Kapitel 2 - Implem. von Datentypen

- Pascal-Maschine
- Begriff der Implementierung
- Implementierung von Arrays
- ... von Records
- ... von Mengen
- ... der Parameterübergabe
- ... von Stacks und Queues
- ... von Rekursion
- Graphen
- Schlußbeispiel: Topologisches Sortieren

Implementierung von Datent.
PASCAL-Maschine

- Grundfrage: Wie bildet man Datenstrukturen auf die Speicherstruktur einer realen Maschine ab, so daß sie effizient ausgeführt bzw. verwaltet werden können?
  - Wie realisiert man die Formularmaschine?
  - Wie bringt man Records, Listen, Bäume in Speicherzellen unter?
- Basismaschine: ASS? Zu kompliziert!
- Basismaschine: PASCAL!

Implementierung von Datent.
PASCAL-Maschine

- Einzelheiten zur Sprache s. Kapitel 3.1 im Skript

Implementierung von Datent.
Begriff der Implementierung - Idee

- Anschaulich: Implementierung D' eines Datentyps D bedeutet:
  - Darstellung der Werte von D durch Werte von D'
  - Simulation der Operationen von D durch Operationen oder Operationsfolgen von D', so daß sich D' nach außen genauso verhält wie D

Implementierung von Datent.
Begriff der Implementierung - Definition

Definition:

Für eine Menge M von Funktionen sei

$$M^{\circ}=\{f_1 \circ f_2 \circ \ldots \circ f_r \mid f_i \in M, r \geq 1\}$$

Menge aller Kompositionen von Funktionen aus M.

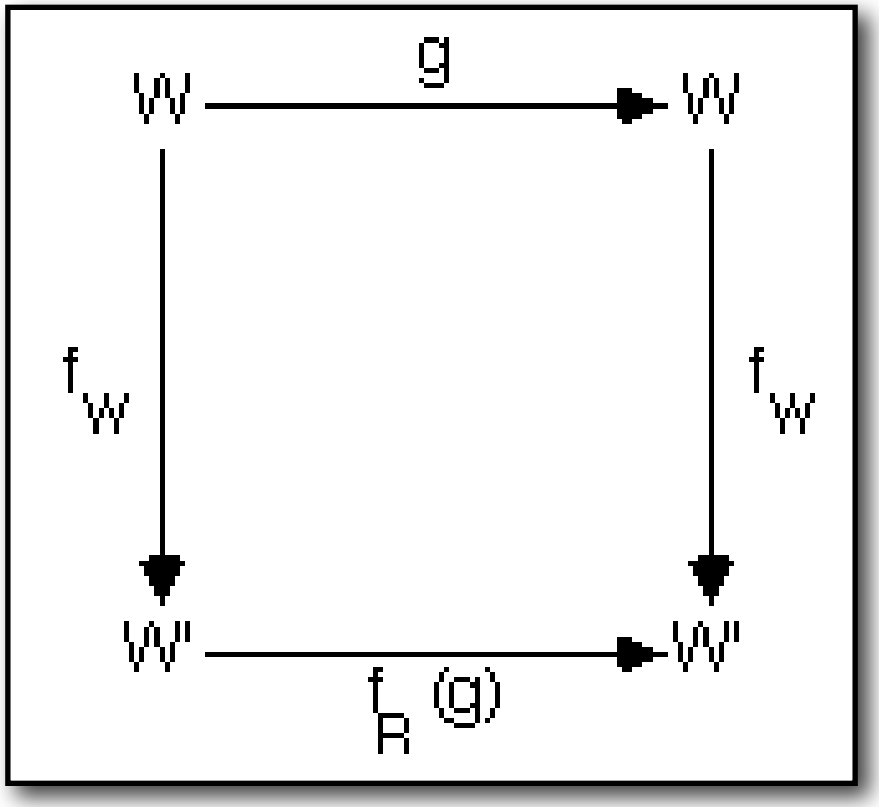
Die **Implementierung** eines Datentyps D=(W,R) durch einen Datentyp D'=(W',R') ist eine Abbildung $f: D \rightarrow D'$ mit $f=(f_W,f_R)$, wobei $f_W: W \rightarrow W'$, $f_R: R \rightarrow R'^{\circ}$, so daß für alle $x \in W, g \in R$ gilt:

$$f_W(g(x))=f_R(g)(f_W(x)).$$

Implementierung von Datent.

Begriff der Implementierung - Definition

kommutierendes Diagramm:



Implementierung von Datent.
Speicherstruktur

- Wie sieht die Speicherstruktur der Basismaschine aus?
  - Folge von einzeln adressierbaren Zellen gleicher Größe
  - Adreßbereich fest
- in PASCAL: homogene Aggregation

  type Speicher=array Adressen of Wort

  type Adressen=0..1048575
- Gesucht: Implementierungen der Form

  $f=(f_W,f_R)$: D → Speicher

Implementierung von Datent.
Implementierung von Arrays - 1-dimens.

- Gegeben 1-dimensionales Array über Grundtyp T

  typ A≡array nat [0..n] of T.

- Annahme: Werte von T passen in ein Wort des Speichers.
- Bestimme Implementierung $f=(f_W,f_R)$:
  A→Speicher mit: Für a∈A und sp∈Speicher mit $f_W(a)$=sp gilt:

  $f_W(a(i))=sp[f_R(i)]$ für alle i mit $0 \leq i \leq n$.

Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß a(i)=sp[i'].

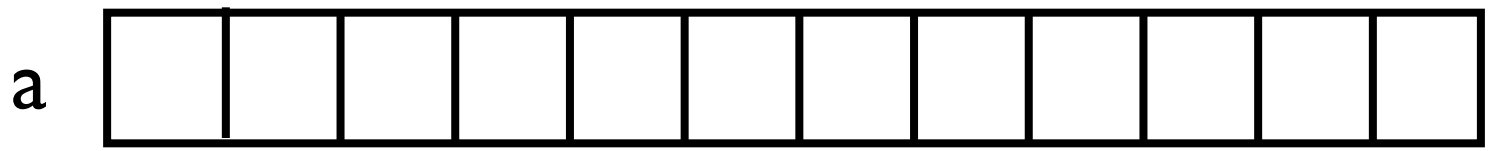
Adreßfunktion

Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß $a(i) = sp[i']$.

Adreßfunktion

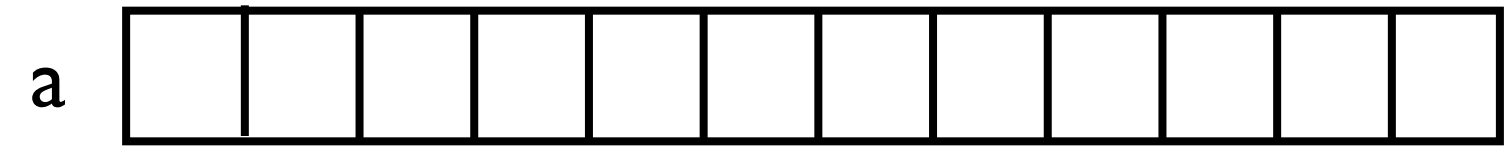


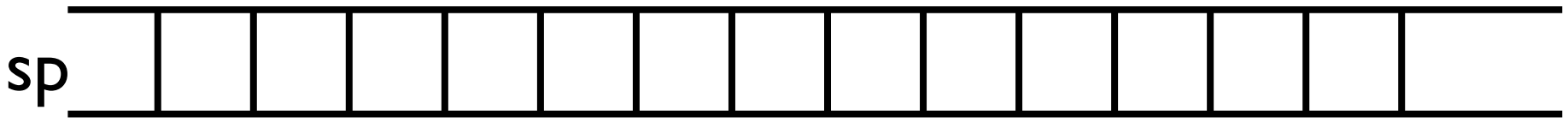
Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß a(i)=sp[i'].

Adreßfunktion



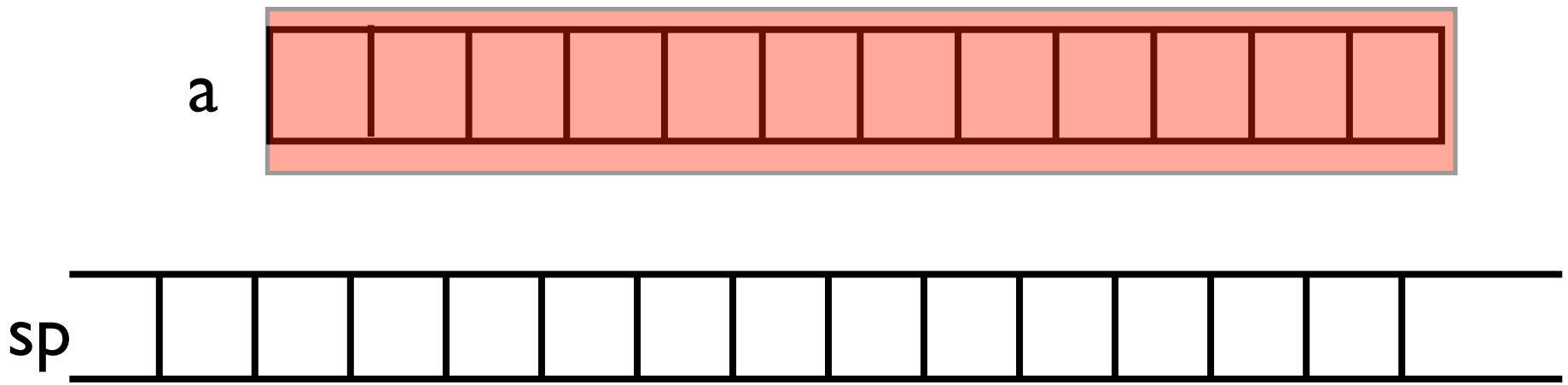


Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß a(i)=sp[i'].

Adreßfunktion

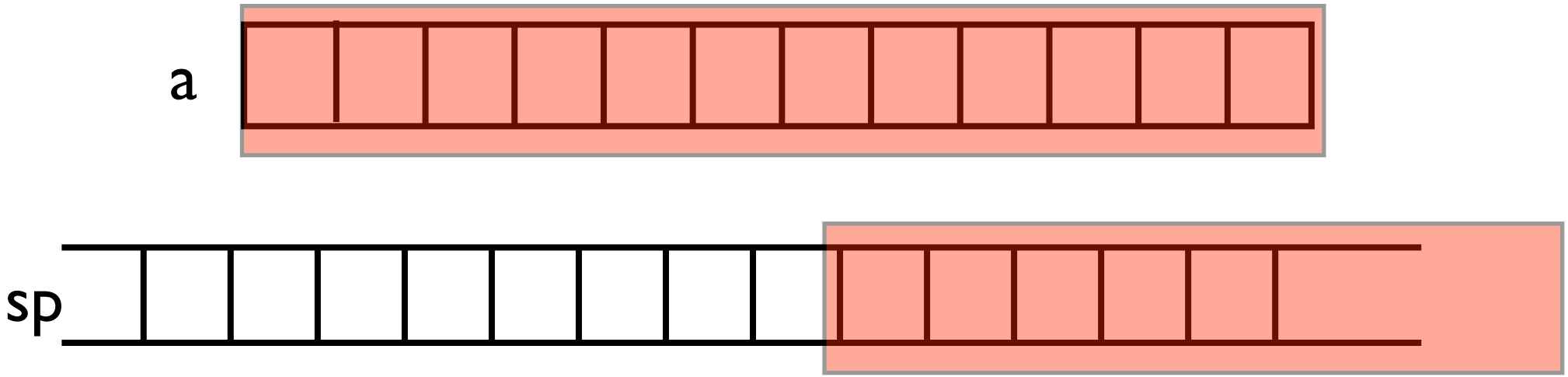


Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß a(i)=sp[i'].

Adreßfunktion

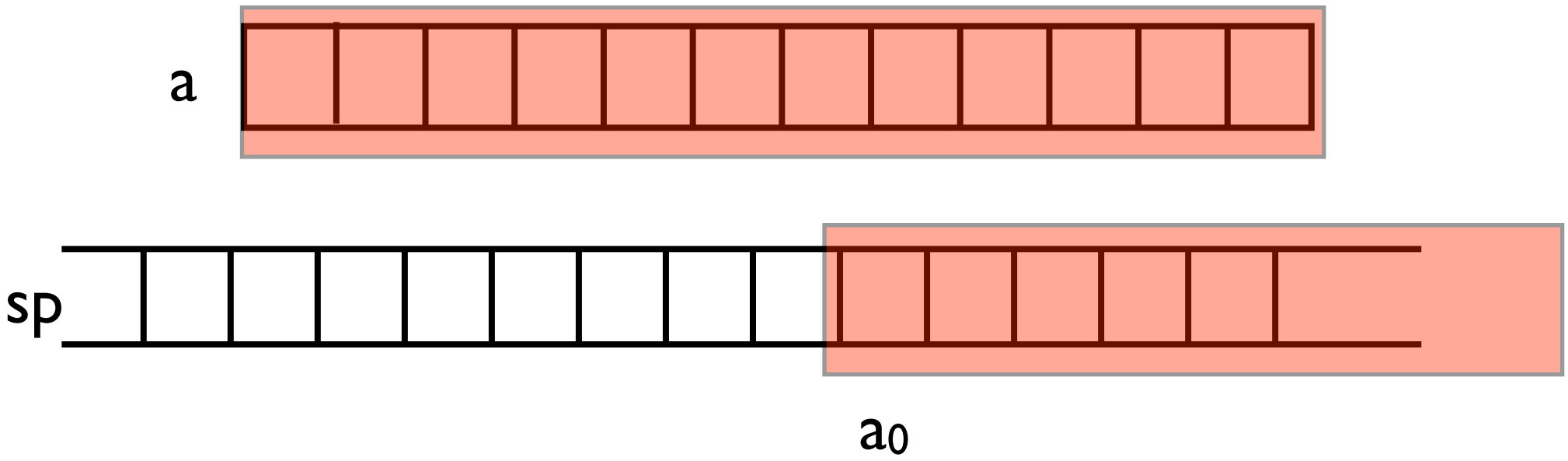


Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß $a(i) = sp[i']$.

Adreßfunktion

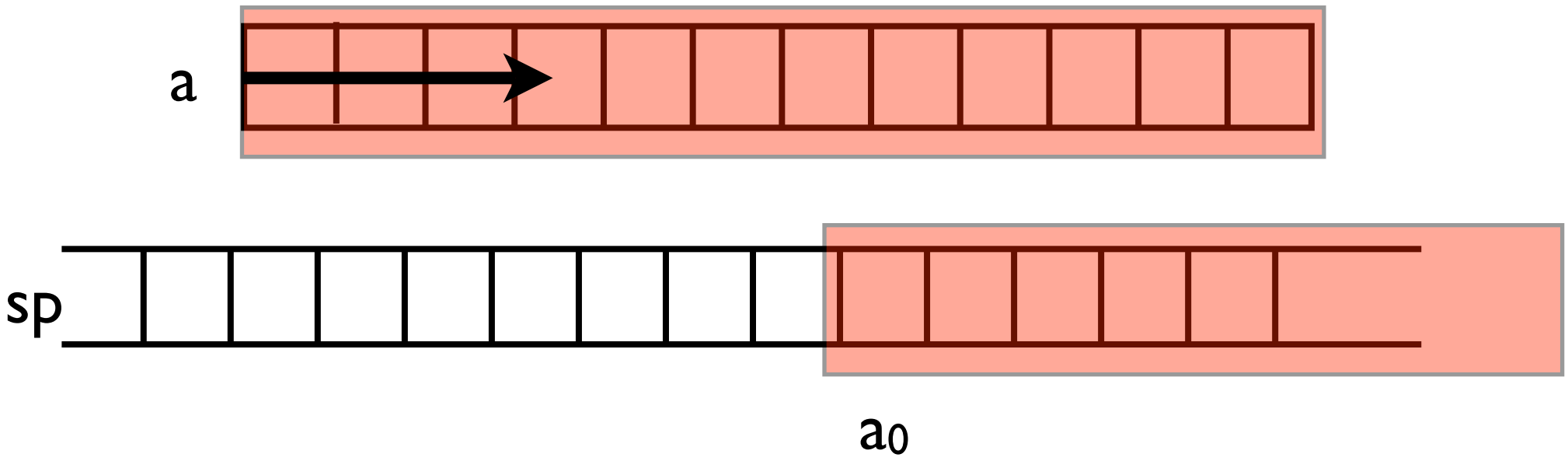


Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß $a(i) = sp[i']$.

Adreßfunktion

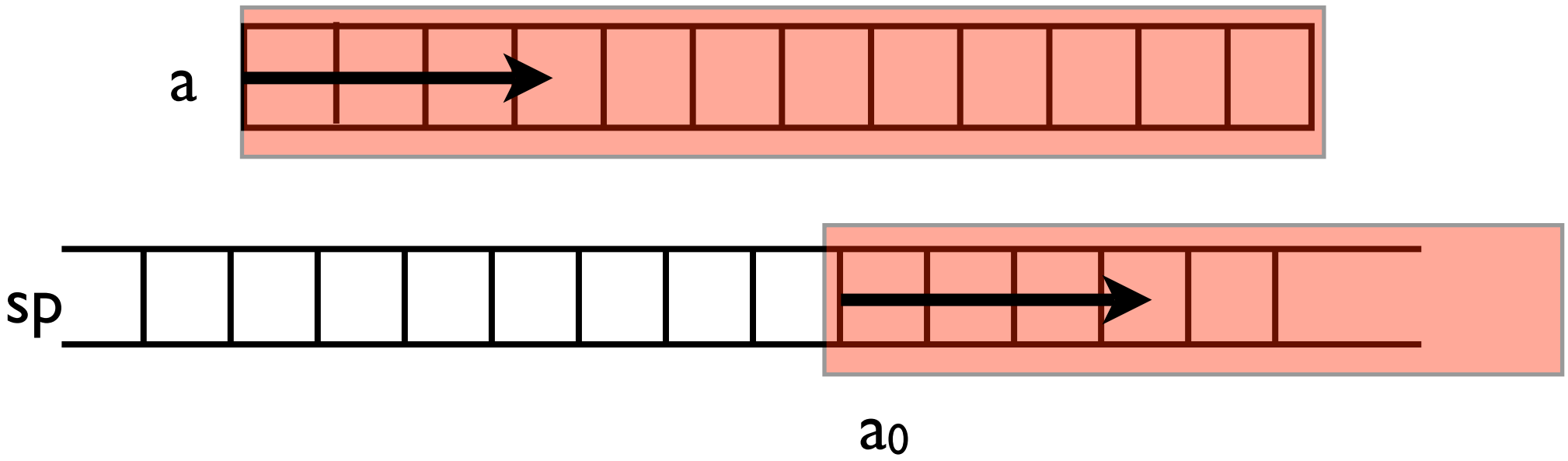


Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß $a(i) = sp[i']$.

Adreßfunktion

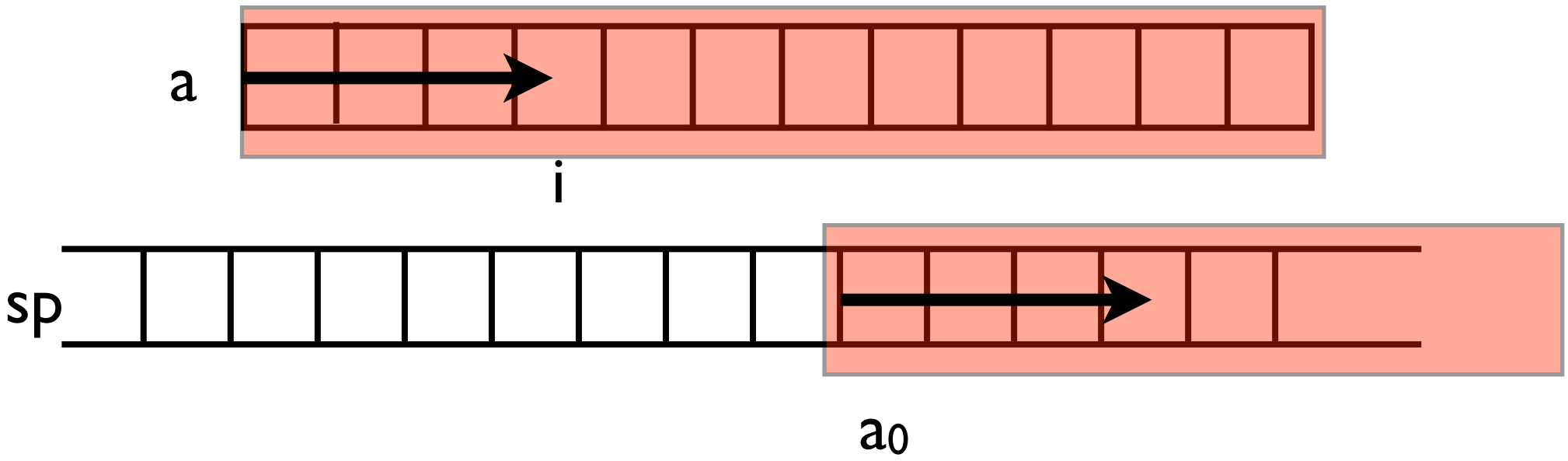


Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß $a(i) = sp[i']$.

Adreßfunktion

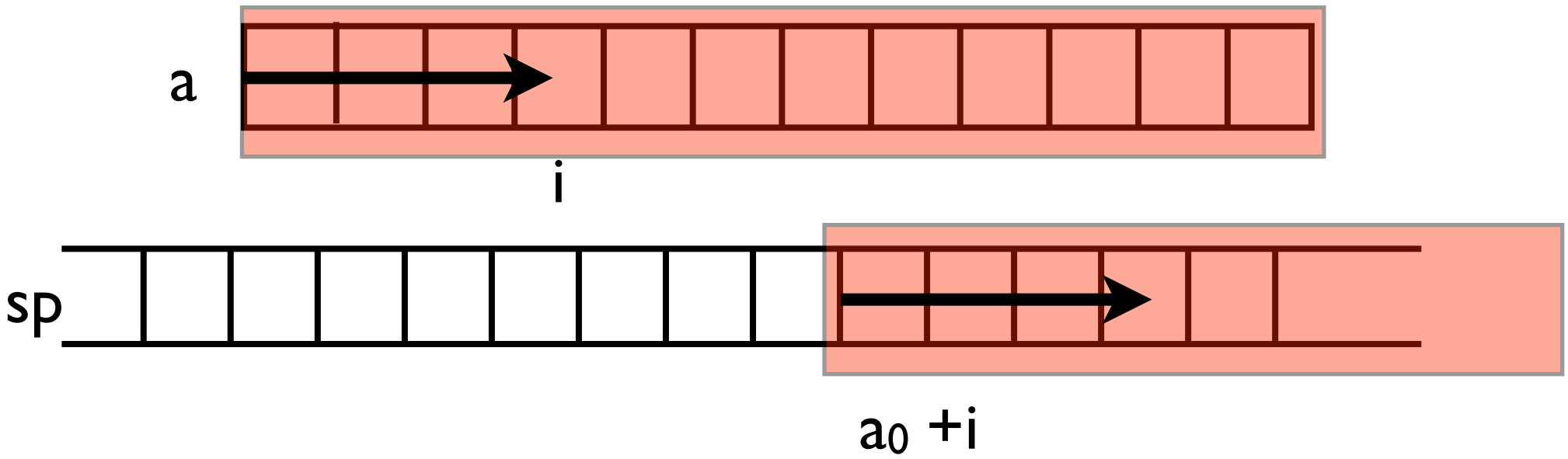


Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß a(i)=sp[i'].

Adreßfunktion

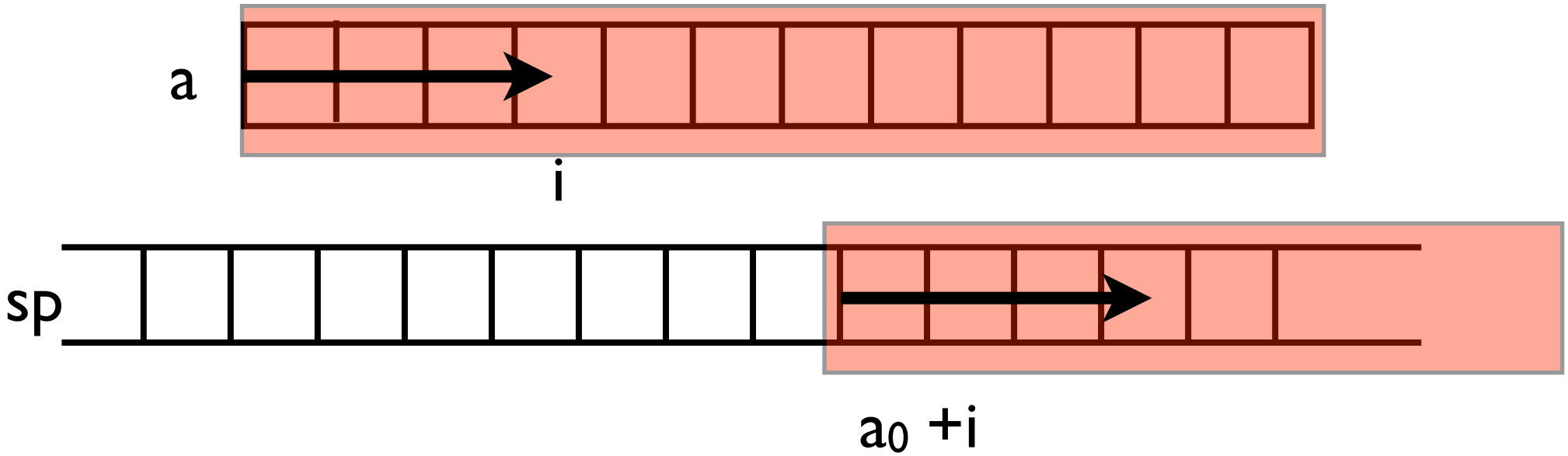


Implementierung von Datent.

Implementierung von Arrays - 1-dimens.

Aufgabe: Bestimme eine geeignete Umrechnung der Indizes $i \rightarrow i' = f_R(\pi_i, n)$ zu bestimmen, so daß $a(i) = sp[i']$.

Adreßfunktion $i' = loc(i) = f_R(i) = a_0 + i,\ 0 \leq i \leq n.$



Implementierung von Datent.
Implementierung von Arrays - 1-dimens.

Erweiterung: c Speicherworte je Wert von T:

$$i' = loc(i) = a_0 + c \cdot i,\ 0 \leq i \leq n,$$

und

$$f_W(a(i)) = sp(f_R(i)) = sp(loc(i)) \ldots sp(loc(i)+c-1).$$

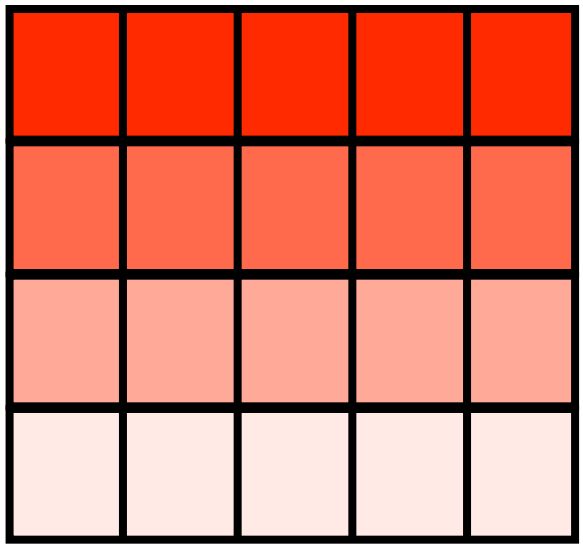
Implementierung von Datent.

Implementierung von Arrays - 2-dimens.

- Gegeben 2-dimensionales Array über Grundtyp T

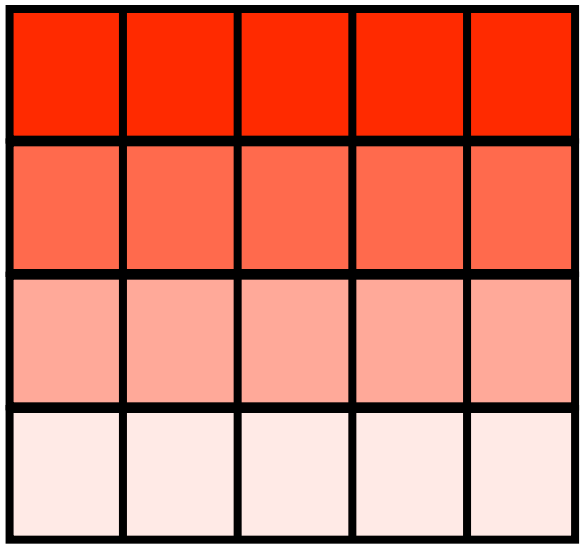
  typ A≡array (nat [0..m],nat [0..n]) of T.

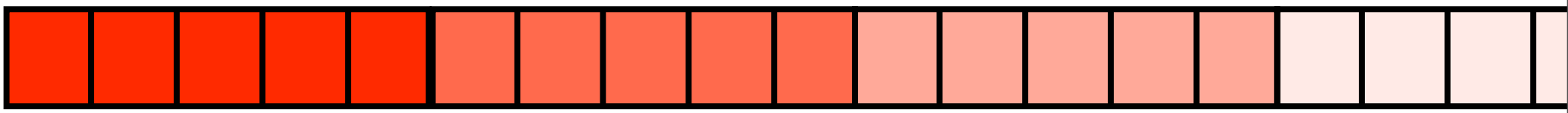
- Aufgabe: lineare Anordnung der Elemente



Implementierung von Datent.

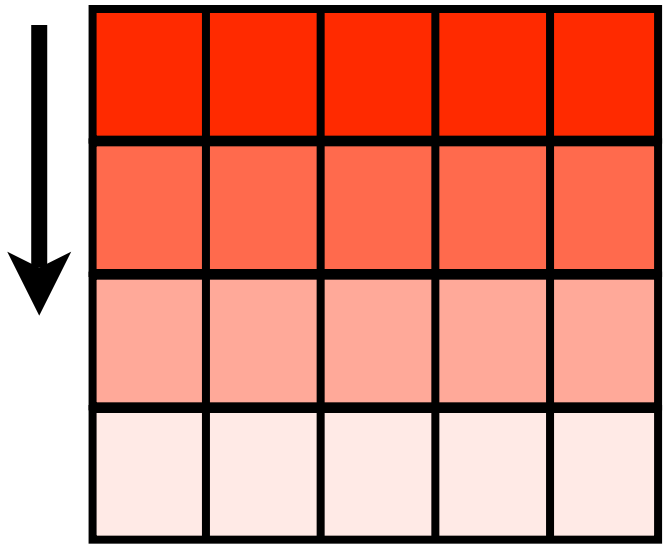
Implementierung von Arrays - 2-dimens.

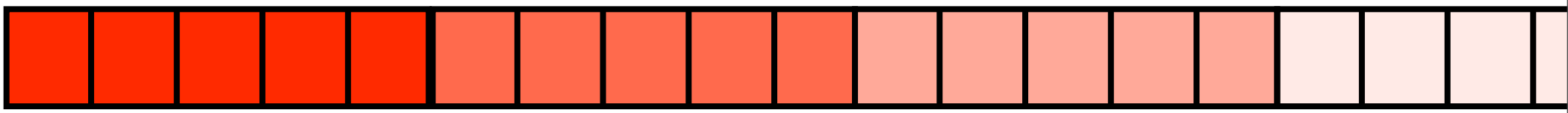




Implementierung von Datent.

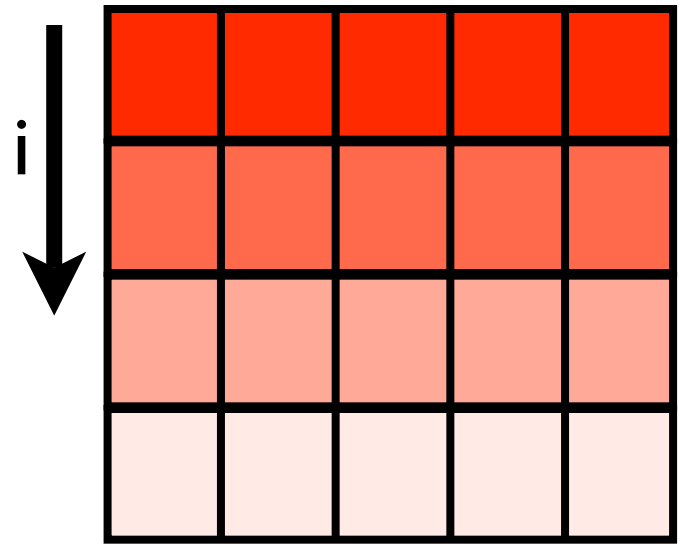
Implementierung von Arrays - 2-dimens.

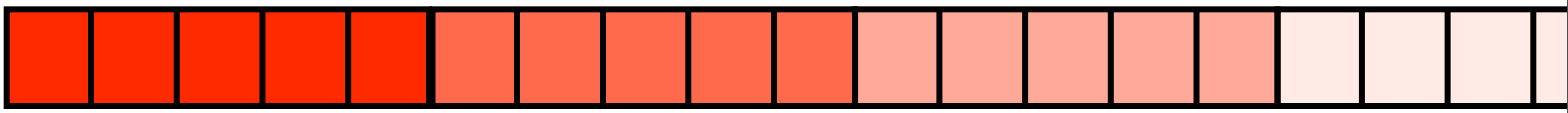




Implementierung von Datent.

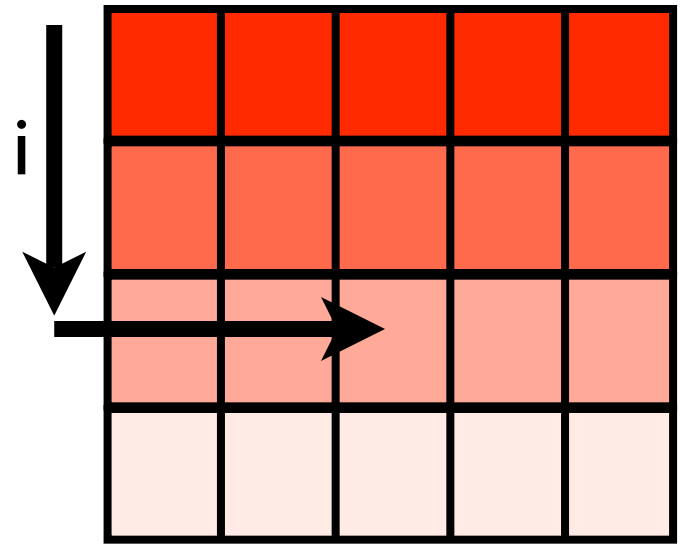
Implementierung von Arrays - 2-dimens.

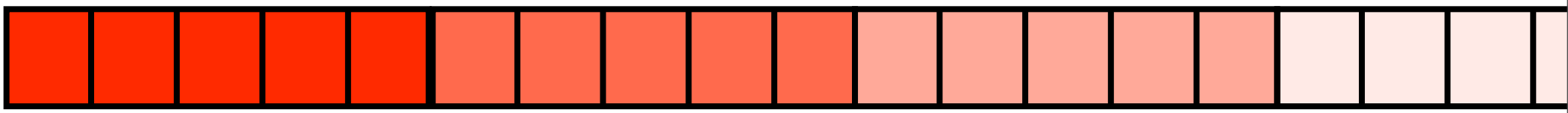




Implementierung von Datent.

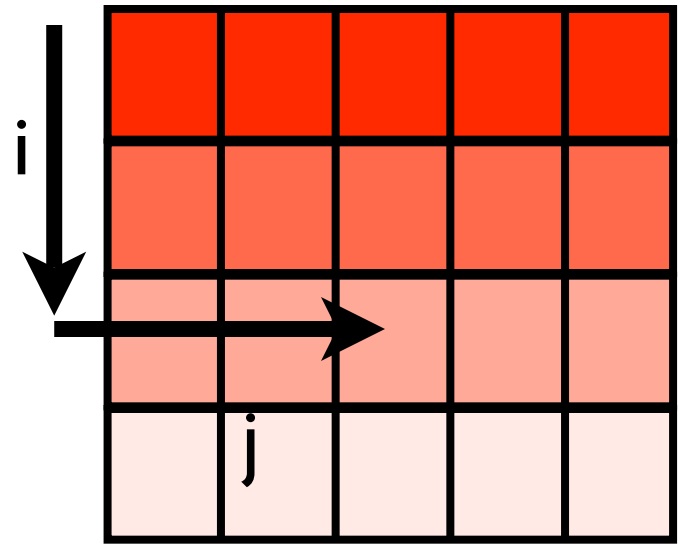
Implementierung von Arrays - 2-dimens.

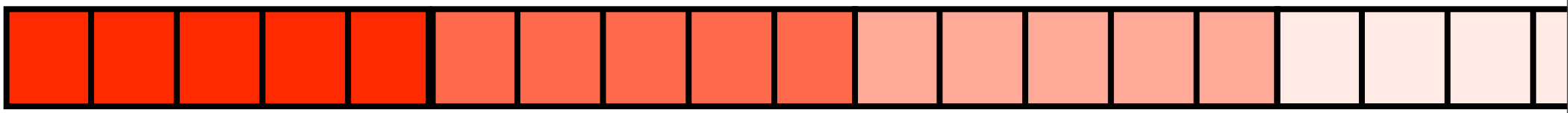




Implementierung von Datent.

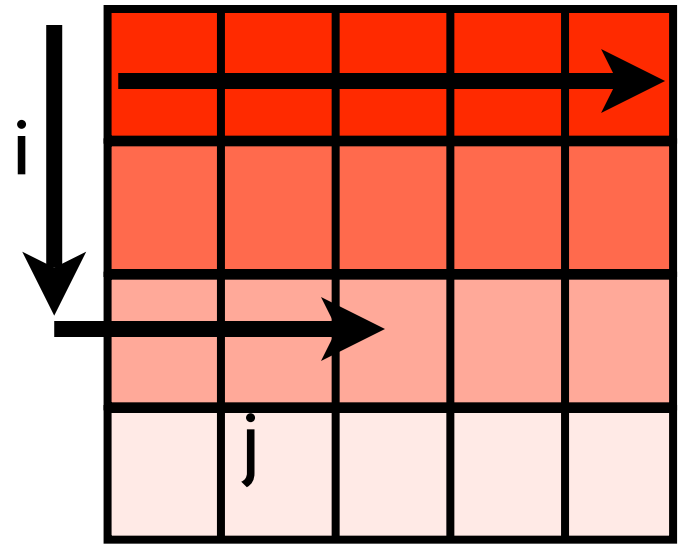
Implementierung von Arrays - 2-dimens.

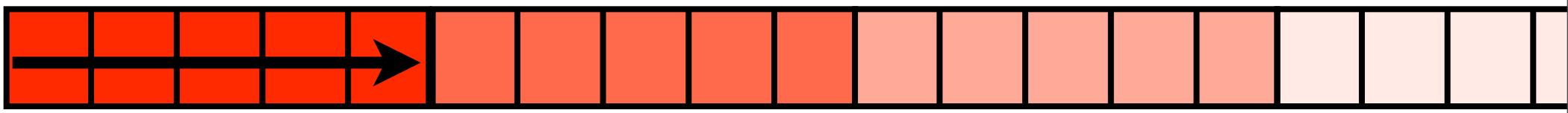




Implementierung von Datent.

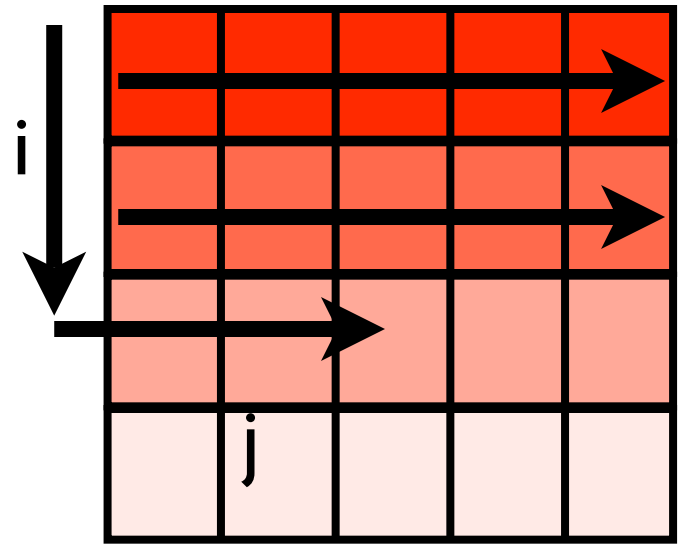
Implementierung von Arrays - 2-dimens.

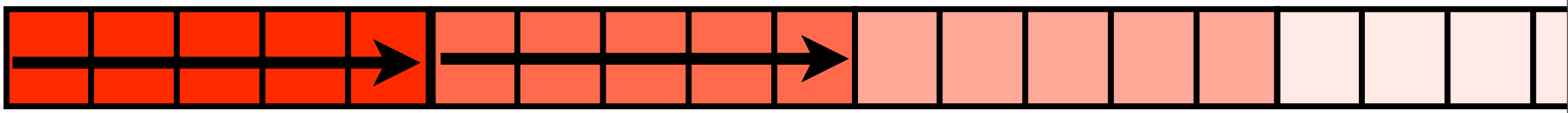




Implementierung von Datent.

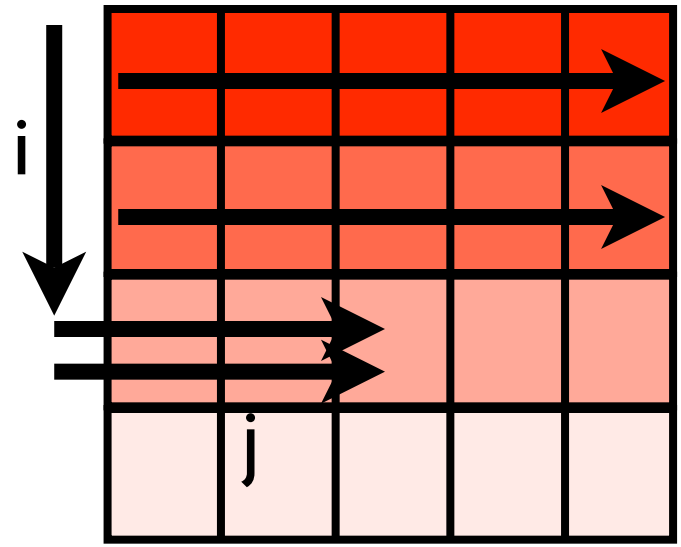
Implementierung von Arrays - 2-dimens.

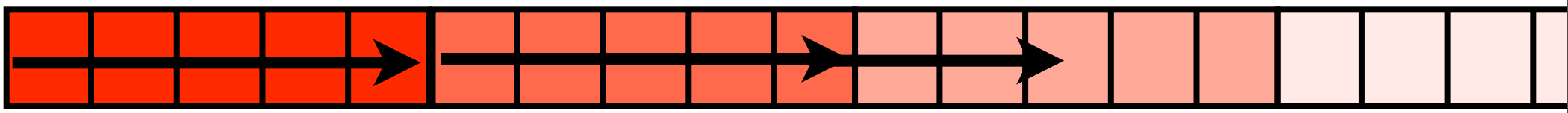




Implementierung von Datent.

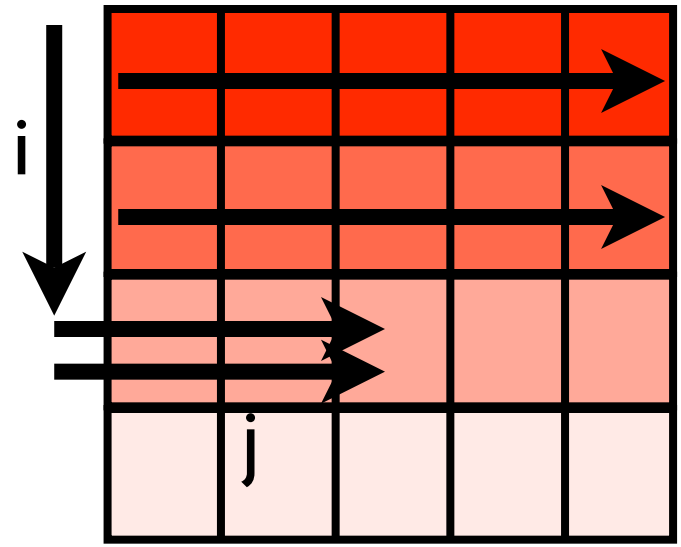
Implementierung von Arrays - 2-dimens.

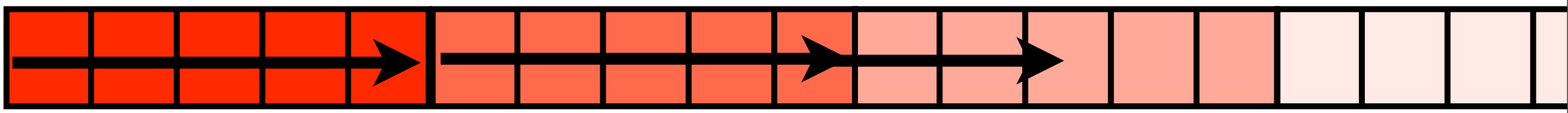




Implementierung von Datent.

Implementierung von Arrays - 2-dimens.

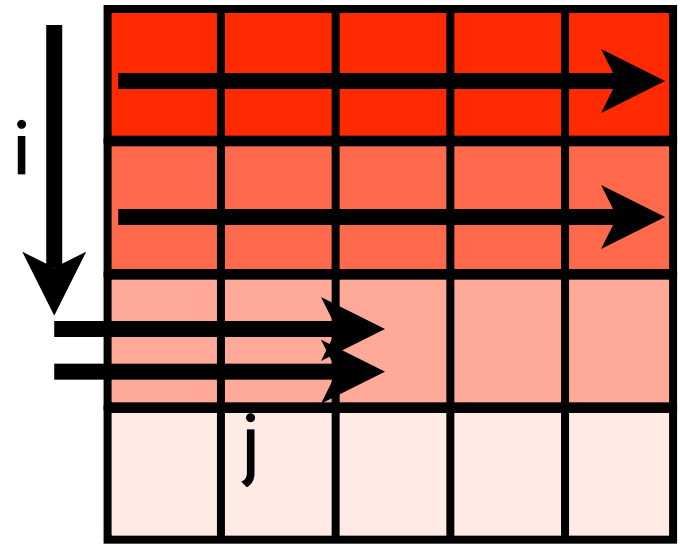


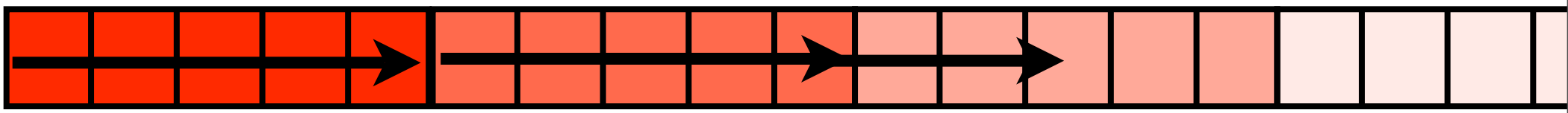


a_0

Implementierung von Datent.

Implementierung von Arrays - 2-dimens.



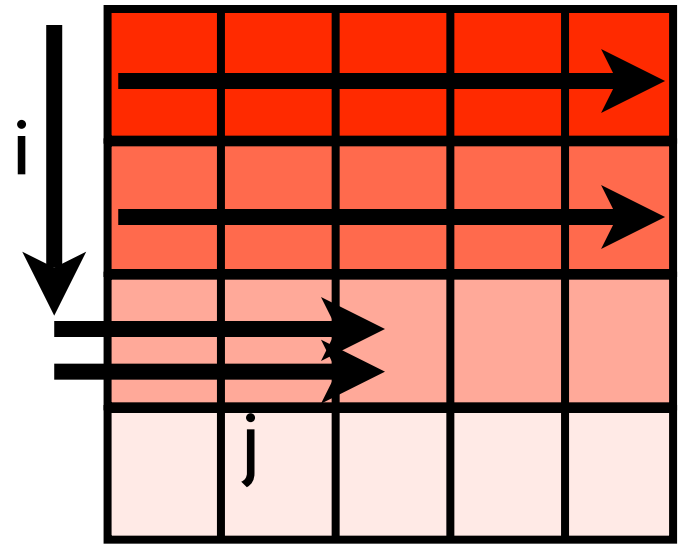


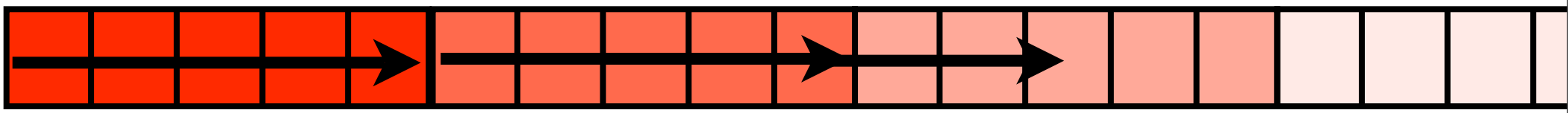
a_0

$$loc(i,j)=a_0+c(n+1)\cdot i+c\cdot j,\ 0\le i\le m,\ 0\le j\le n$$

Implementierung von Datent.

Implementierung von Arrays - 2-dimens.





a_0

$$loc(i,j)=a_0+c(n+1)\cdot i+c\cdot j,\ 0\le i\le m,\ 0\le j\le n$$

$$f_W(a(i,j))=sp(f_R(i,j))=sp(loc(i,j))...sp(loc(i,j)+c-1).$$

Implementierung von Datent.
Implementierung von Arrays - n-dimens.

- Gegeben:

  typ A≡array (nat [0..n_1],nat [0..n_2],...,nat [0..n_k]) of T

- Adreßfunktion:

$$
\begin{aligned}
loc(d_1,d_2,...,d_k) \quad = \quad & a_0 \\
& +c\cdot d_1(n_2+1).....(n_k+1) \\
& +c\cdot d_2(n_3+1).....(n_k+1) \\
& ... \\
& +c\cdot d_{k-1}(n_k+1) \\
& +c\cdot d_k \qquad = a_0+\sum_{j=1}^{k} c_j\cdot d_j
\end{aligned}
$$

Implementierung von Datent.
Implementierung von Arrays - n-dimens.

- Gegeben:

  typ A≡array (nat [0..n_1],nat [0..n_2],...,nat [0..n_k]) of T

- Adreßfunktion:

$$loc(d_1,d_2,...,d_k) = a_0$$
$$+c\cdot d_1(n_2+1).....(n_k+1)$$
$$+c\cdot d_2(n_3+1).....(n_k+1)$$
$$...$$
$$+c\cdot d_{k-1}(n_k+1)$$
$$+c\cdot d_k = a_0+\sum_{j=1}^{k} c_j\cdot d_j$$

$$c_j=c\prod_{l=j+1}^{k}(n_l+1)$$

Implementierung von Datent.
Implementierung von Records

- Analog zu Speicherung von Arrays: lege Komponenten eines Objekts beginnend bei Speicherzelle a_0 sequentiell ab
- Gegeben:

$$\text{typ } T \equiv (t_1 : T_1, t_2 : T_2, ..., t_n : T_n)$$

- Adreßfunktion:

$$loc(t_i) = a_0 + c_1 + c_2 + ... + c_{i-1}$$

Implementierung von Datent.
Implementierung von Records

- Analog zu Speicherung von Arrays: lege Komponenten eines Objekts beginnend bei Speicherzelle a_0 sequentiell ab
- Gegeben:

$$\text{typ } T \equiv (t_1 : T_1, t_2 : T_2, \ldots, t_n : T_n)$$

- Adreßfunktion:

Platzbedarf für Objekte t_i

$$loc(t_i) = a_0 + c_1 + c_2 + \ldots + c_{i-1}$$

Implementierung von Datent.
Implementierung von Records

- Analog zu Speicherung von Arrays: lege Komponenten eines Objekts beginnend bei Speicherzelle a_0 sequentiell ab
- Gegeben:

$$\text{typ } T \equiv (t_1 : T_1, t_2 : T_2, \ldots, t_n : T_n)$$

- Adreßfunktion:

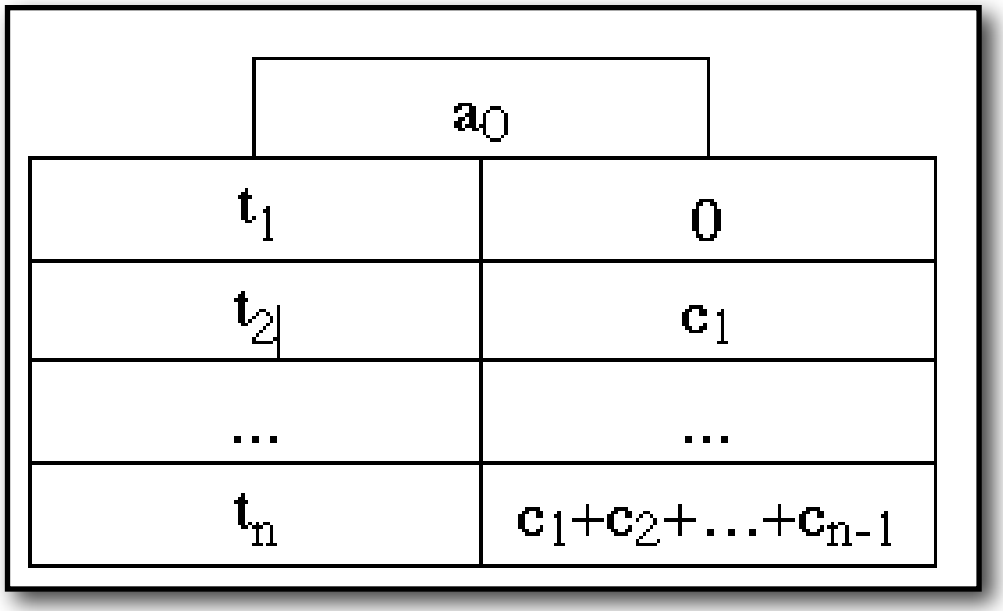
Platzbedarf für Objekte t_i

$$loc(t_i) = a_0 + c_1 + c_2 + \ldots + c_{i-1}$$

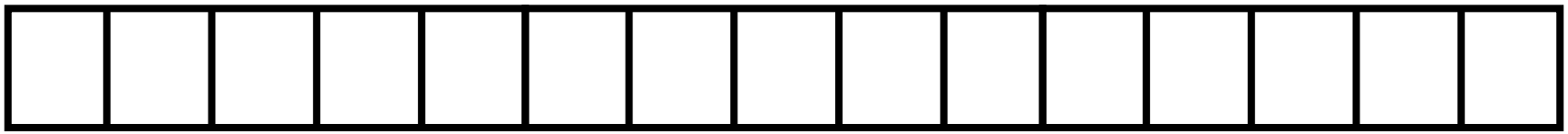
Offsets

Implementierung von Datent.

Implementierung von Records



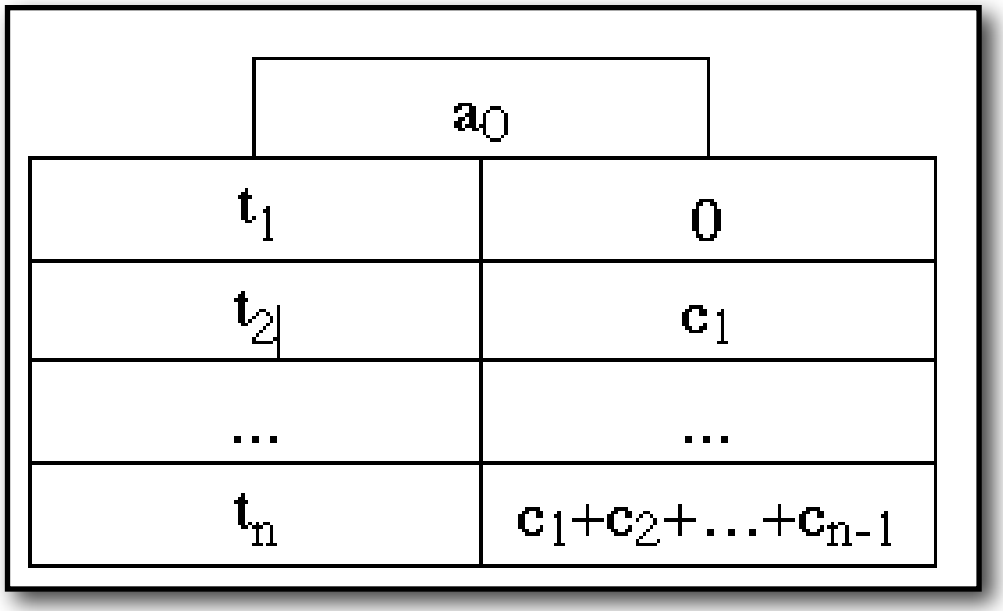
Zugriffstabelle



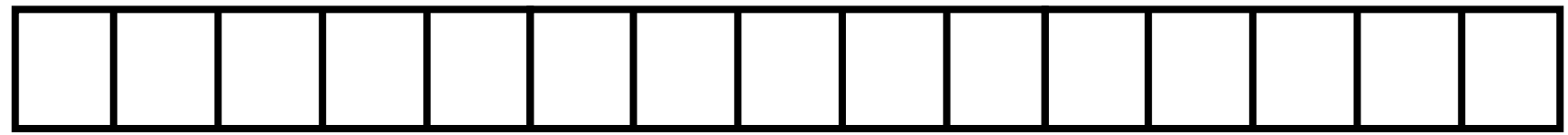
Speicher

Implementierung von Datent.

Implementierung von Records



Zugriffstabelle



Speicher

Implementierung von Datent.
Implementierung von Mengen

- Potenzmengentyp allgemein: typ $D \equiv 2^{D'}$
- Regelvoraussetzung: D' endlich, sei $D'=\{d_1,...,d_n\}$
- Standardimplementierung einer beliebigen Menge $M \in D$ durch **charakteristische Funktion**

$$\chi_M: D' \rightarrow \text{bool mit}$$

$$\chi_M(x) = \begin{cases} 1, \text{ falls } x \in M, \\ 0, \text{ sonst.} \end{cases}$$

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

↑ 1 ist in M

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

↑ 2 ist nicht in M

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

↑ 3 ist in M

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

↑ 4 ist in M

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

↑ 5 ist nicht in M

Implementierung von Datent.
Implementierung von Mengen

- Bei endlichen Grundtypen: charakteristische Funktion ist endliche Folge von n booleschen Werten (0-1-Folgen)

$$\chi_M=(\chi_M(d_1),...,\chi_M(d_n))$$

- Beispiel: Menge $M=\{1,3,4,7\}\in 2^{\{1,2,3,4,5,6,7,8,9,10\}}$ durch 0-1-Folge dargestellt:

1 0 1 1 0 0 1 0 0 0.

Implementierung von Datent.
Implementierung von Mengen

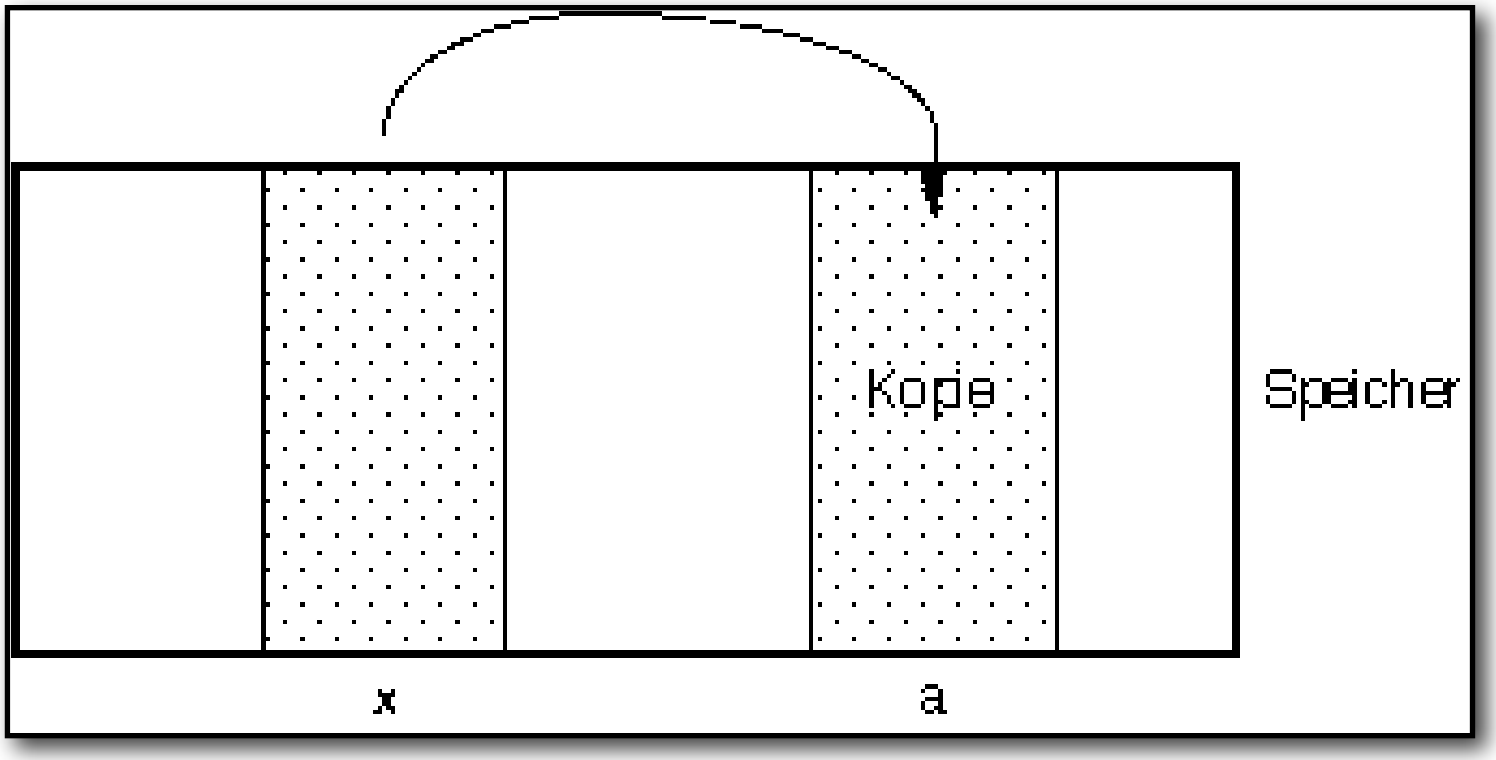
- Operationen auf Mengen einfach:

  Mengenoperation=boolesche Operation
- Vereinigung: stellenweise ODER-Funktion
- Durchschnitt: stellenweise UND-Funktion
- Komplement: stellenweise Negation
- Elementabfrage $d_i \in M$: Test, ob i-te Stelle = 1

- Effiziente Realisierung auf Prozessorebene

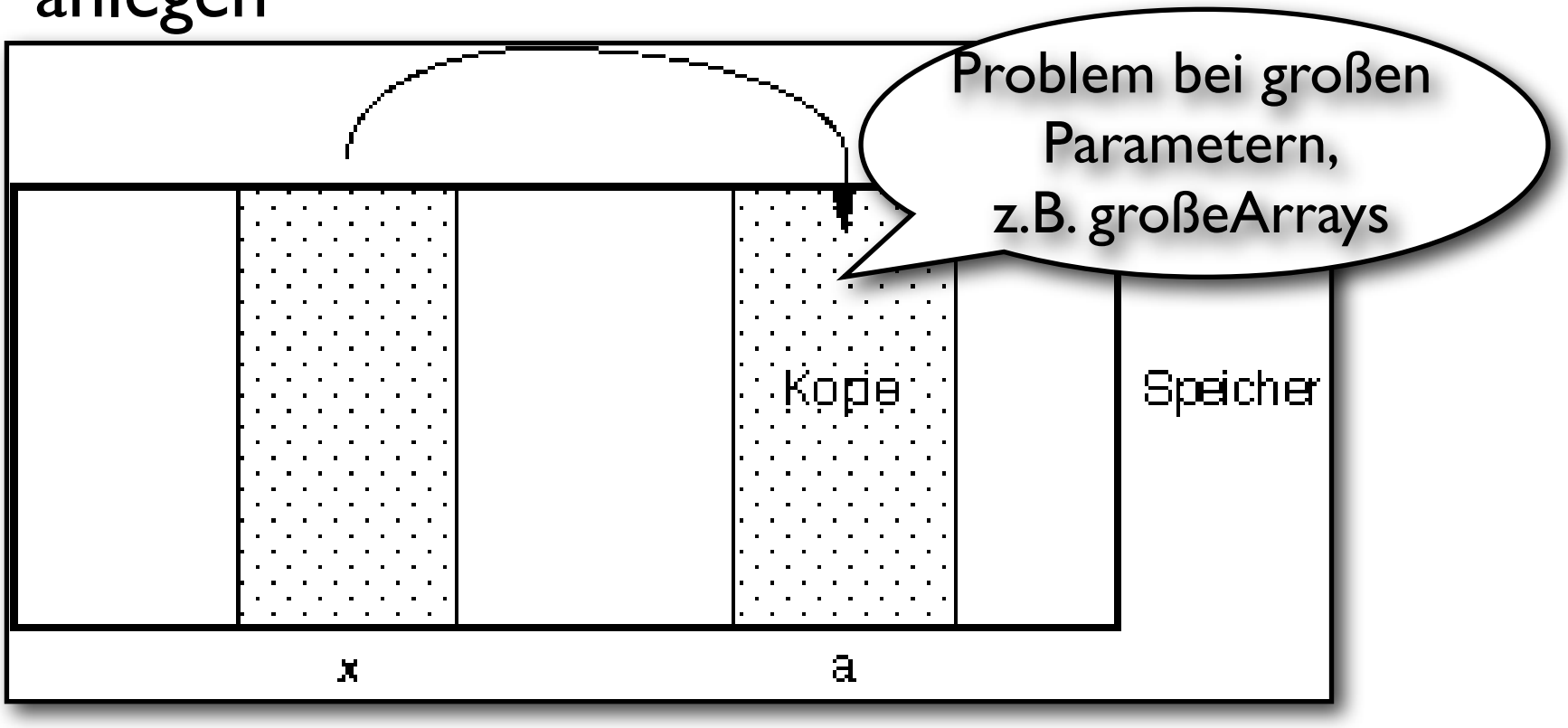
Implementierung von Datent.
Parameterübergabe - call by value

- Gegeben: x aktueller, a formaler Parameter
- Verfahren: Kopie des aktuellen Parameters anlegen



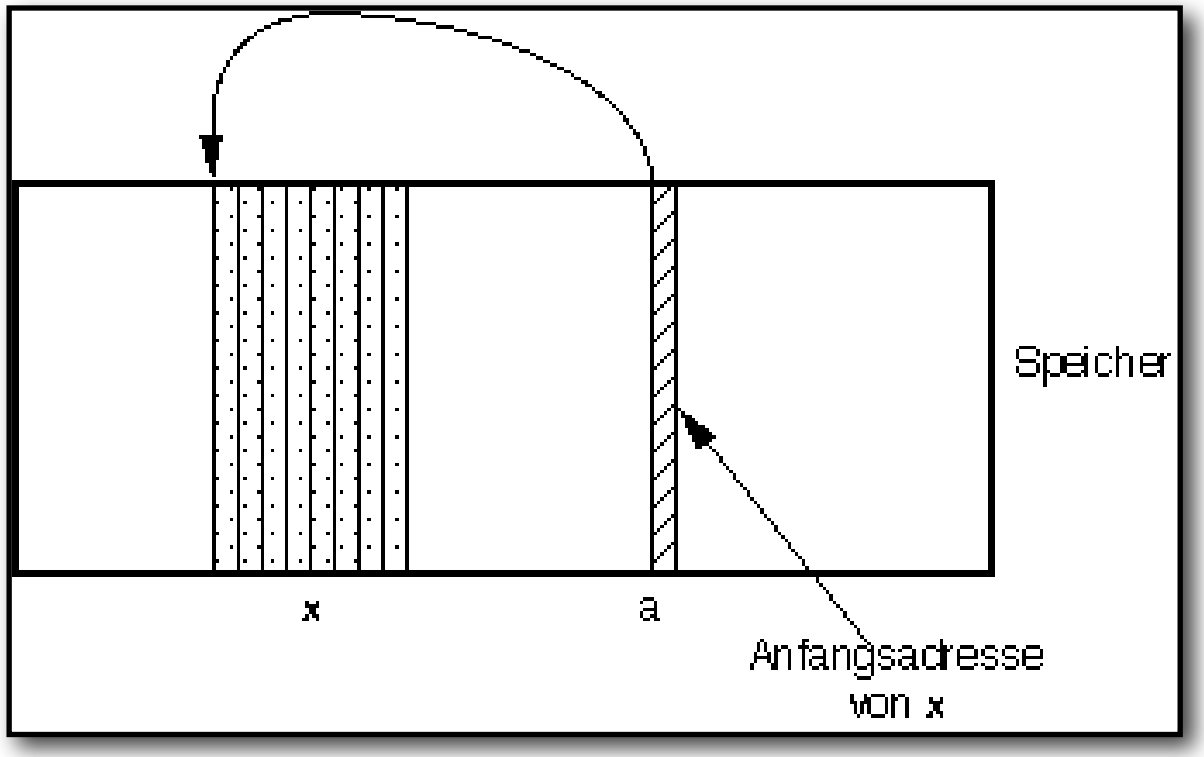
Implementierung von Datent.
Parameterübergabe - call by value

- Gegeben: x aktueller, a formaler Parameter
- Verfahren: Kopie des aktuellen Parameters anlegen



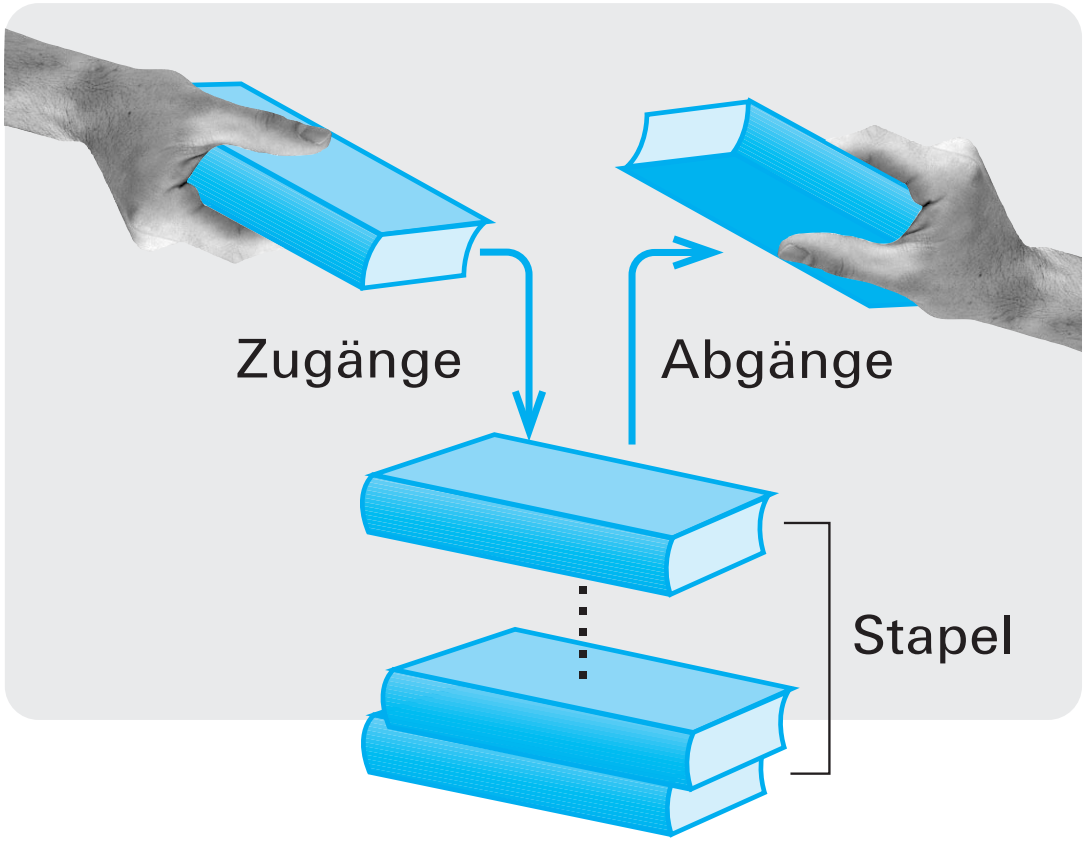
Implementierung von Datent.
Parameterübergabe - call by reference

- Gegeben: x aktueller, a formaler Parameter
- Verfahren: Anlegen einer Referenz (Zeiger) auf den aktuellen Parameter



Implementierung von Datent.

Implementierung von Stacks



Stapel, Keller, Push-down-Speicher, LIFO-Speicher, Stack

Operationen: push, pop, top, empty, is_empty, ...

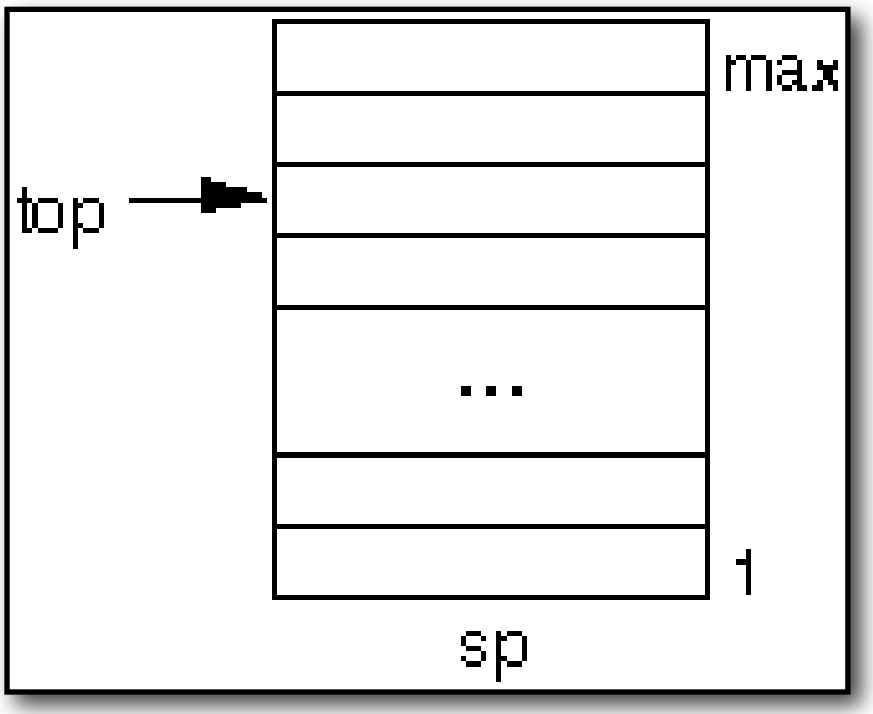
Implementierung von Datent.
Implementierung von Stacks

- Varianten:
  - sequentielle Speicherung: Einträge der Reihe nach in aufeinanderfolgenden Speicherzellen
  - verkettete Speicherung: Einträge im Speicher verstreut, Herstellung der Reihenfolge durch Verweise/Zeiger/ Referenzen

Implementierung von Datent.

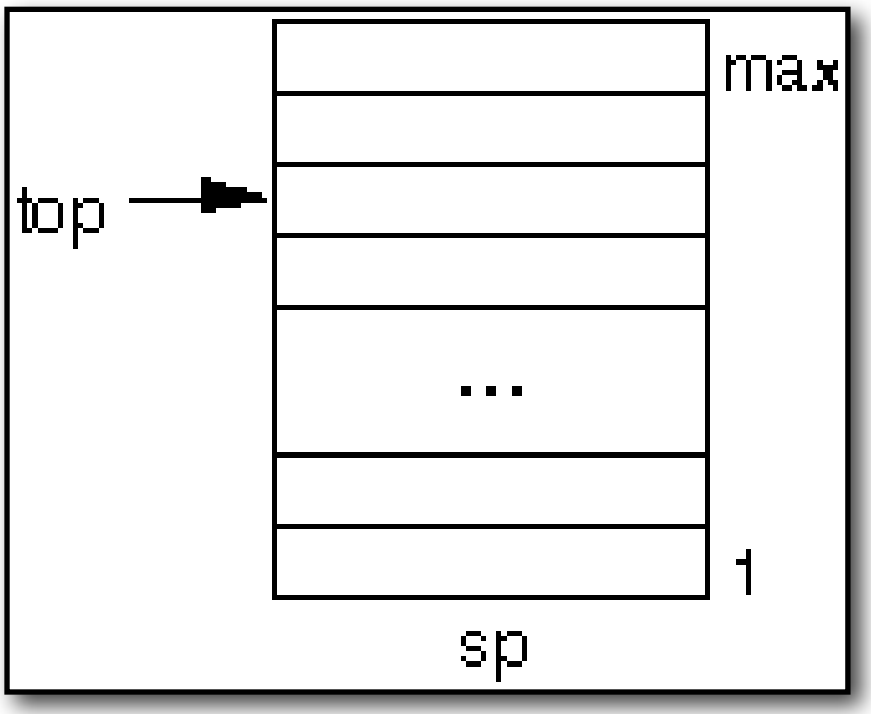
Implementierung von Stacks - sequent.

- bei bekannter maximaler Größe max: Array



Implementierung von Datent.
Implementierung von Stacks - sequent.

- bei bekannter maximaler Größe max: Array

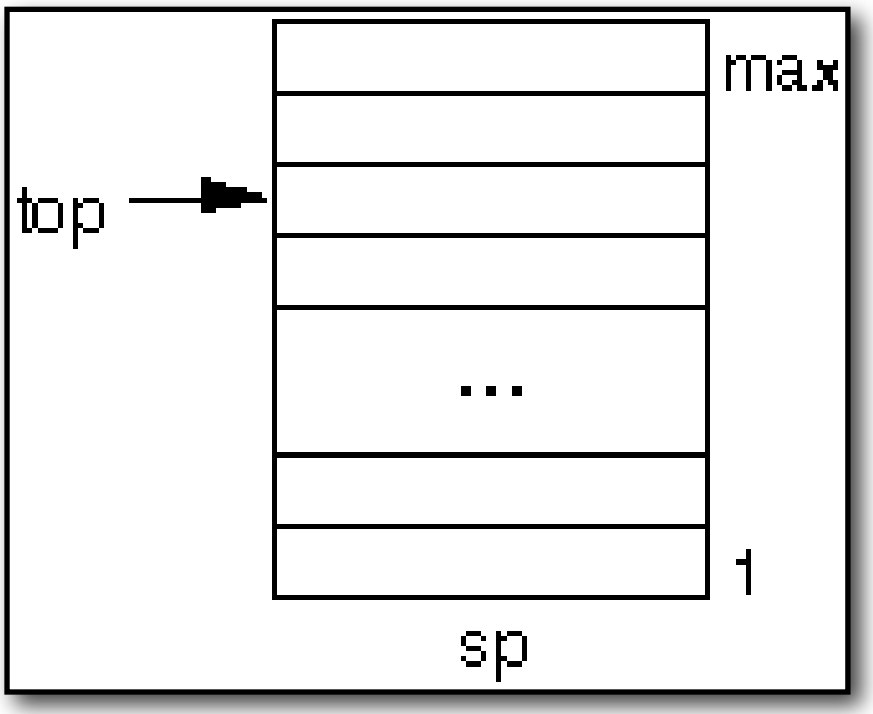



```
type data=...;
     stack=record
               top: 0..max;
               sp: array [1..max] of data
              end;
```

Implementierung von Datent.

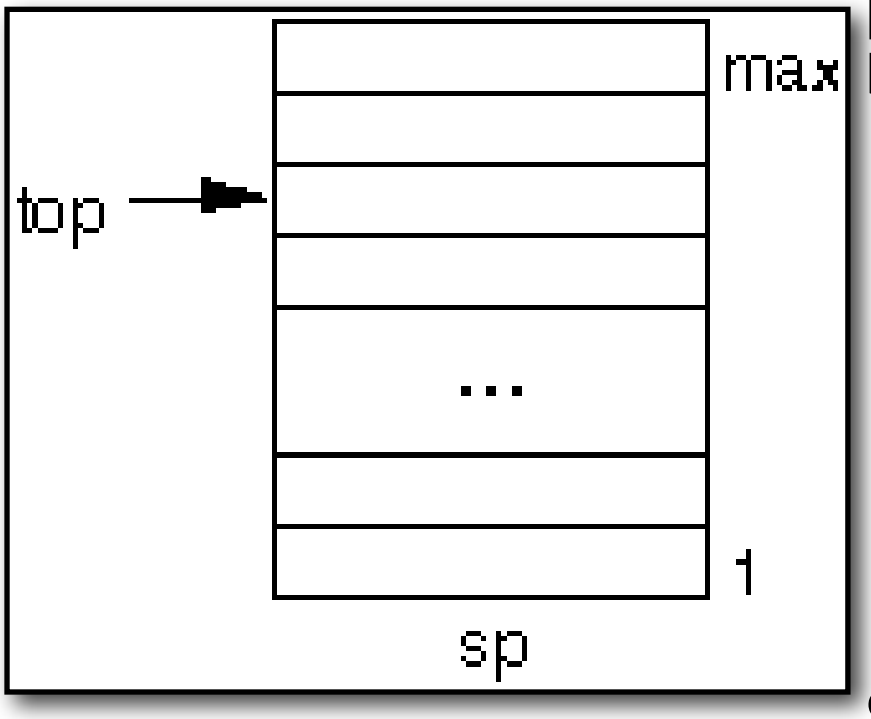
Implementierung von Stacks - sequent.

- bei bekannter maximaler Größe max: Array



Implementierung von Datent.
Implementierung von Stacks - sequent.

- bei bekannter maximaler Größe max: Array

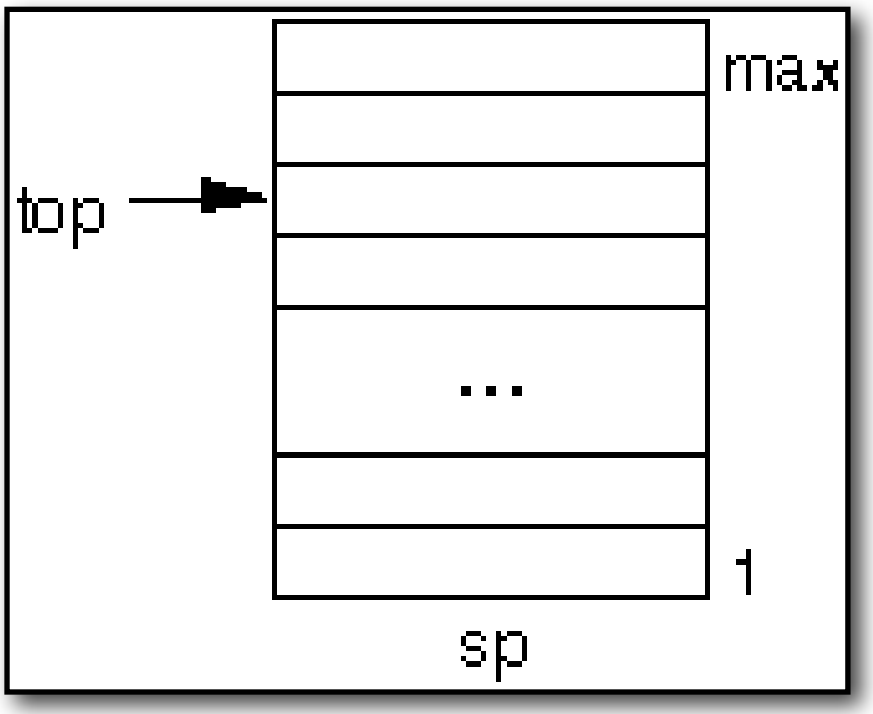



```
procedure push(x: data; var s: stack);
begin
    with s do
    begin
        if top=max then "Overflow" else
        begin
            top:=top+1;
            sp[top]:=x
        end
    end
end;
```

Implementierung von Datent.

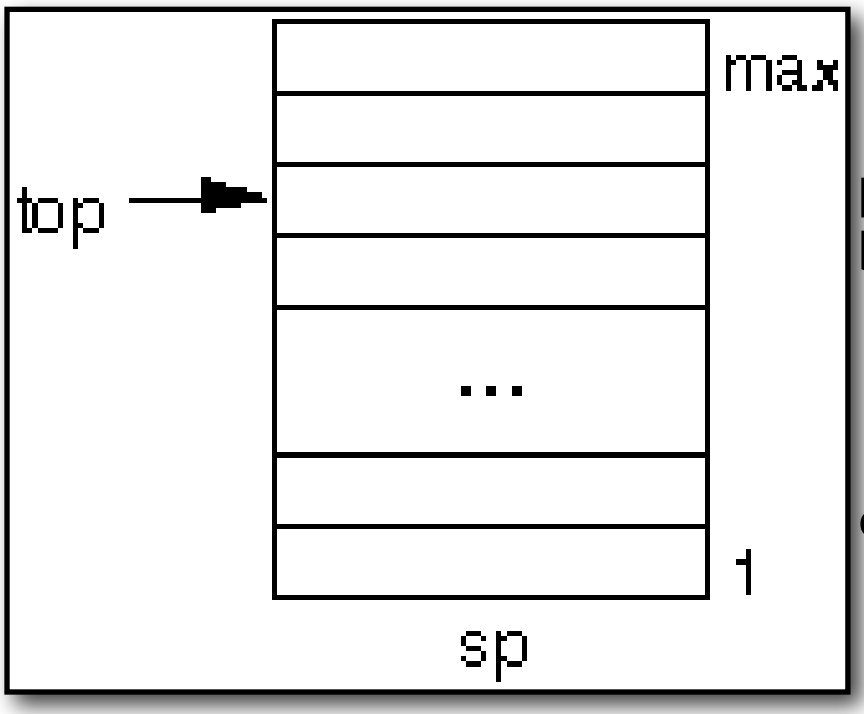
Implementierung von Stacks - sequent.

- bei bekannter maximaler Größe max: Array



Implementierung von Datent.
Implementierung von Stacks - sequent.

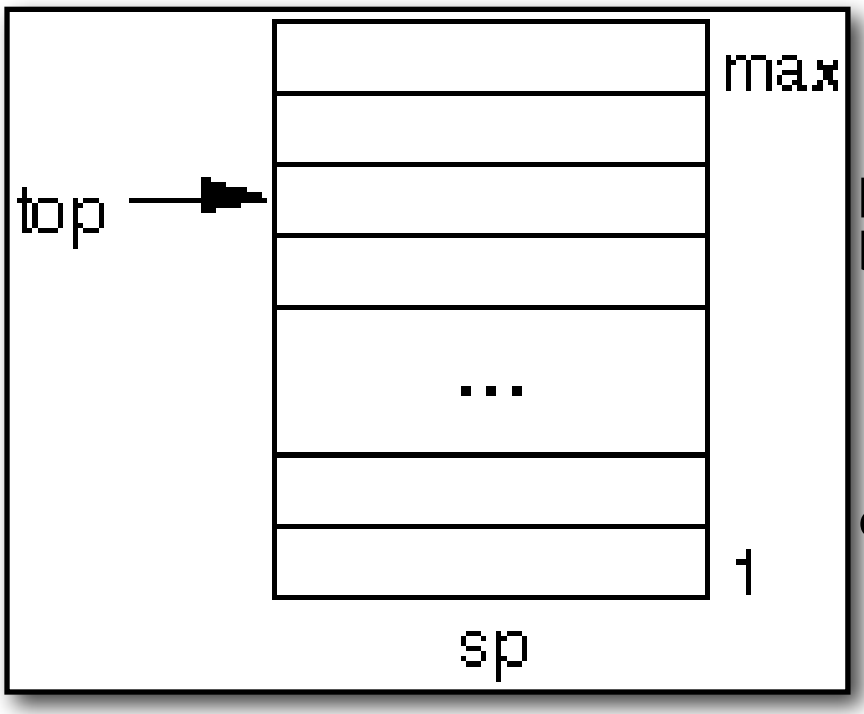
- bei bekannter maximaler Größe max: Array




```
procedure pop(var s: stack);
begin
      with s do
            if is_empty(s) then "Underflow"
                        else top:=top-1
end;
```

Implementierung von Datent.
Implementierung von Stacks - sequent.

- bei bekannter maximaler Größe max: Array




```
procedure pop(var s: stack);
begin
      with s do
            if is_empty(s) then "Underflow"
                        else top:=top-1
end;
```

Initialisierung:
var s: stack; empty(s).

Implementierung von Datent.
Implementierung von Stacks - Beispiel

Programm zur Syntaxanalyse von Klammerausdrücken:

- BNF-Grammatik G=(N,T,P,S) mit: N={<Wort>}, T={(,),[,]}, S=<Wort>, und P enthält Produktion:

  <Wort>::= (<Wort>) <Wort> | [<Wort>] <Wort> | ε.

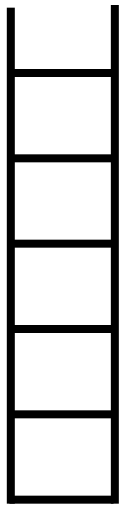
- L(G)=allen Zeichenfolgen mit Klammern (,),[,], die als korrekte Klammerung anzusehen sind:

  (), (())()()(()), [()(([]))][], ...

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

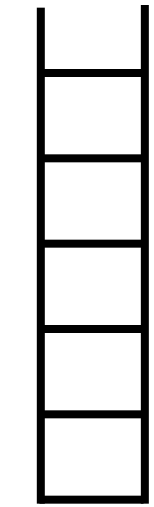
Eingabewort

(( ) [ ])

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

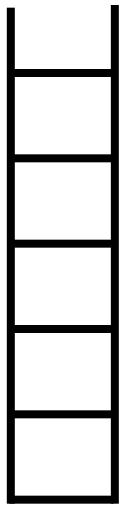
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

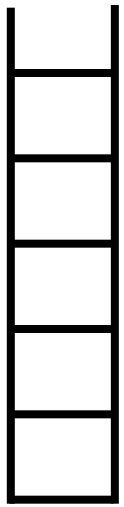
Eingabewort

(( ) [ ])

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

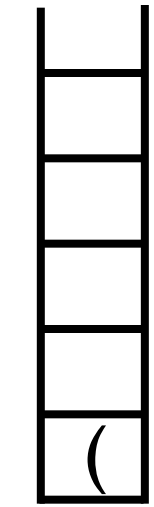
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

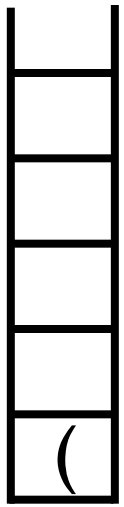
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

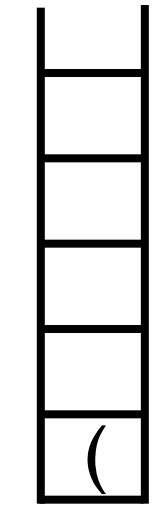
Eingabewort

(( ) [ ])

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

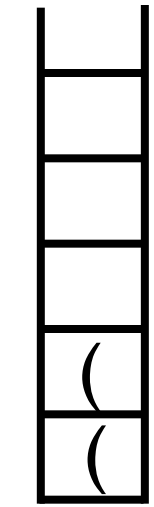
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

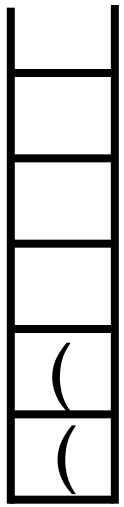
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

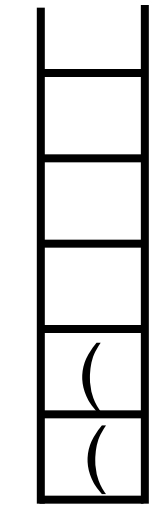
Eingabewort

(( ) [ ])

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

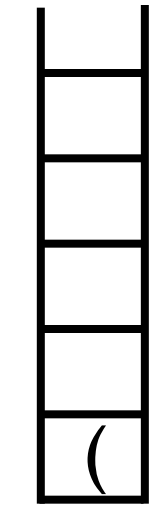
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

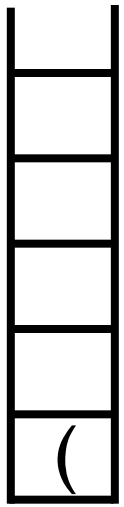
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

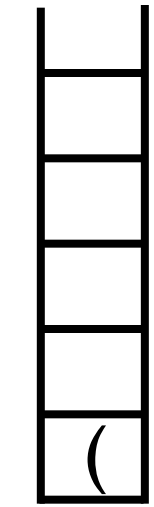
Eingabewort

(( ) [ ])

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

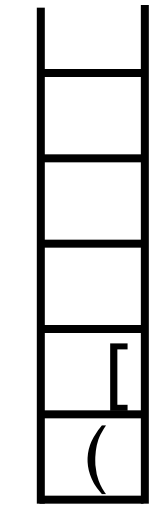
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

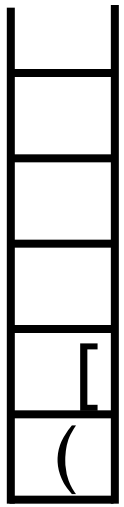
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

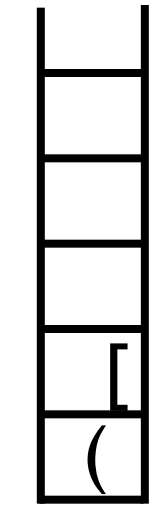
Eingabewort

(( ) [ ])

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

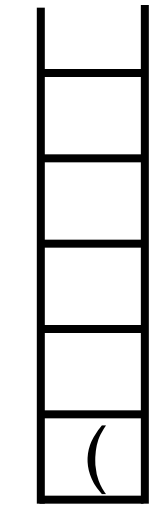
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

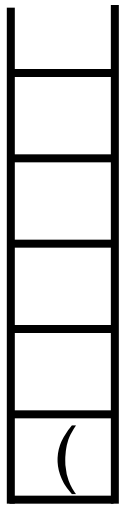
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

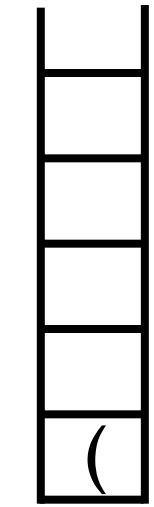
Eingabewort

(( ) [ ])

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

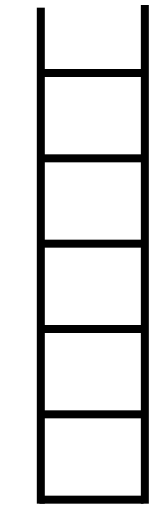
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

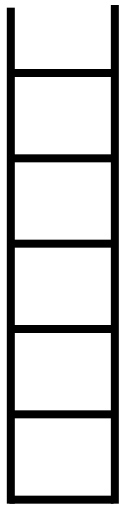
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

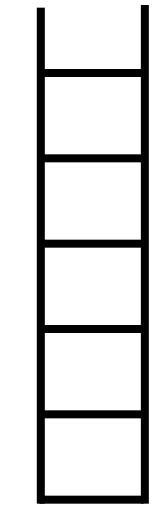
Eingabewort

(( ) [ ])

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

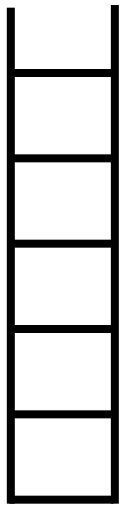
(( ) [ ])

↑

Implementierung von Datent.

Implementierung von Stacks - Beispiel

- Programm s. Skript
- Arbeitsweise:



Stack

Eingabewort

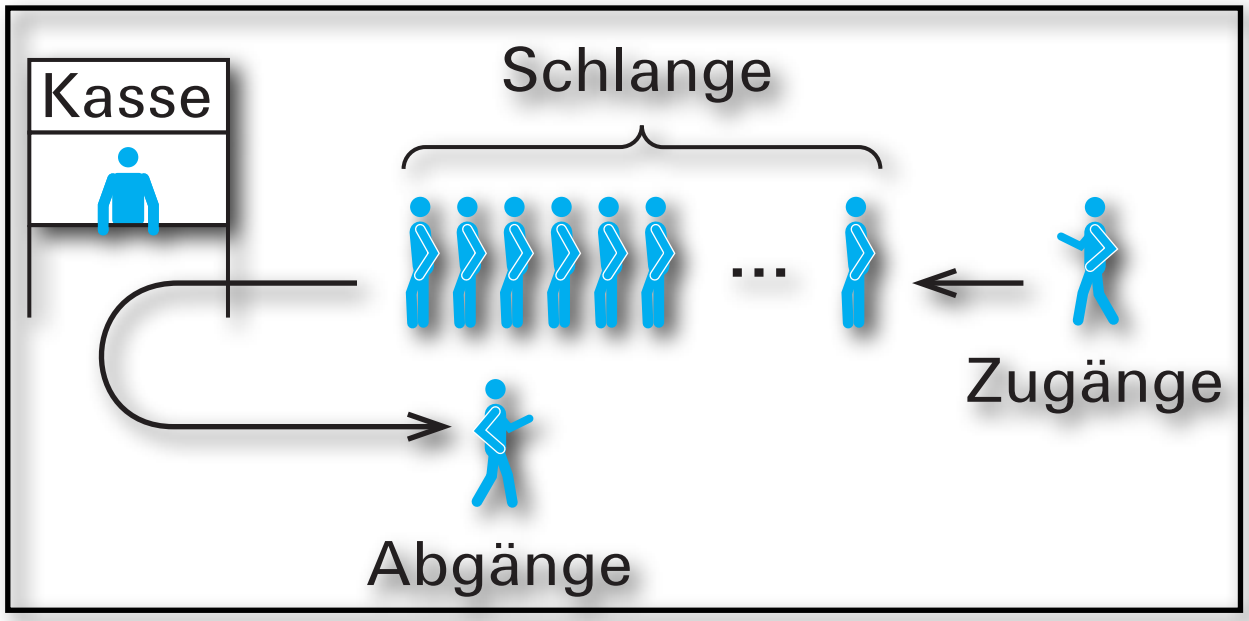
(( ) [ ])

↑

Stacks sind die zentrale Datenstruktur bei der Syntaxanalyse von Programmiersprachen

Implementierung von Datent.

Implementierung von Queues



Schlange, FIFO-Speicher, Queue

Operationen: enter, remove, first, empty, is_empty, ...

Implementierung von Datent.

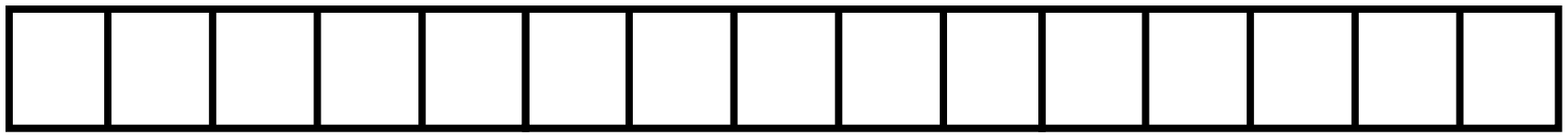
Implementierung von Queues - sequent.

- bei bekannter maximaler Größe max: Array

Implementierung von Datent.

Implementierung von Queues - sequent.

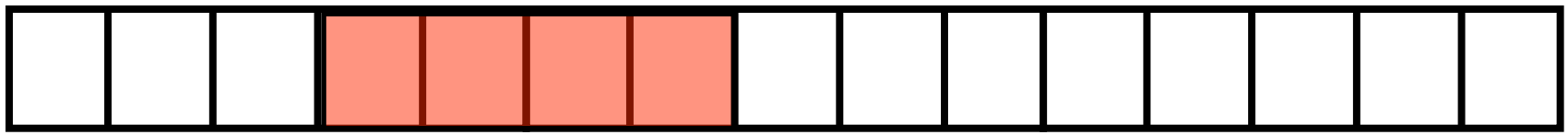
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

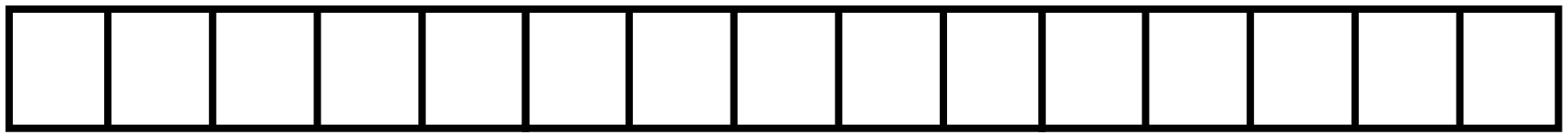
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

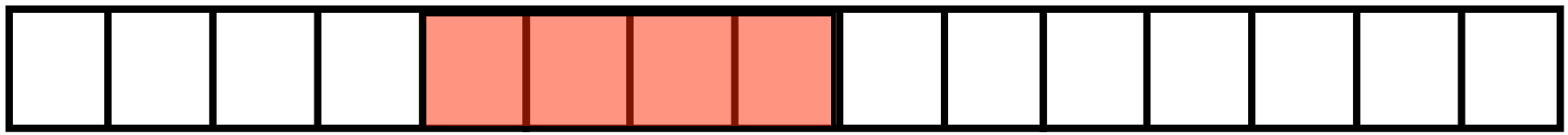
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

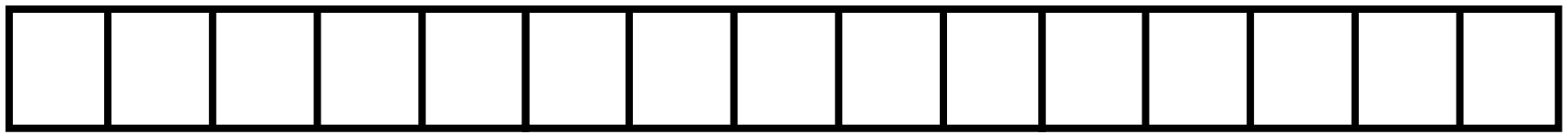
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

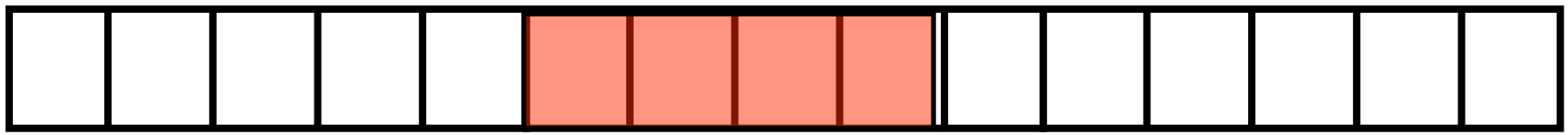
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

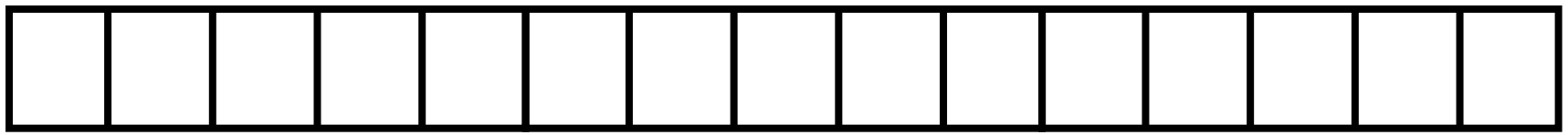
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

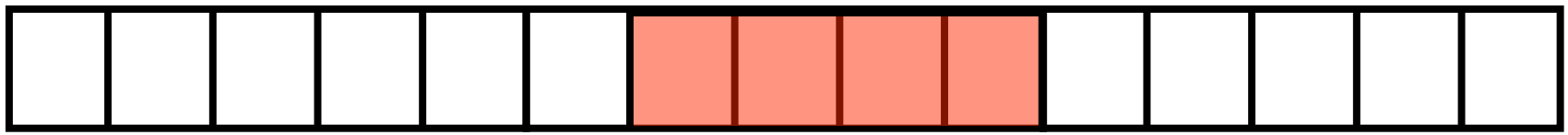
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

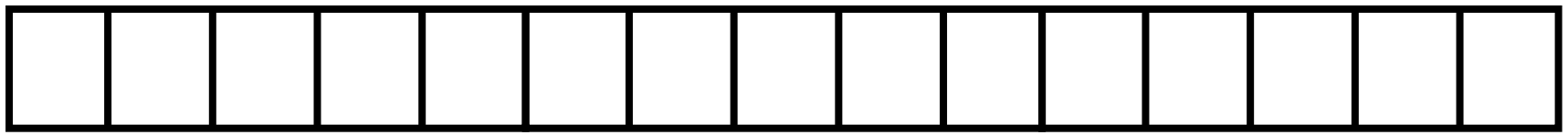
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

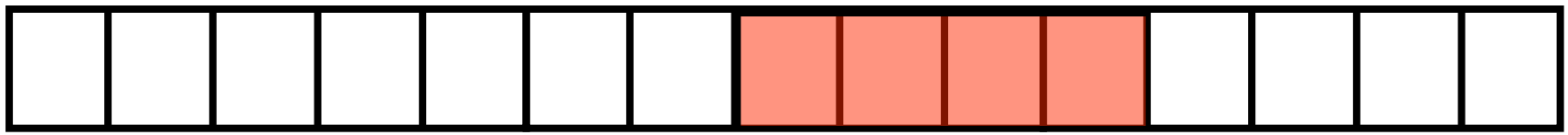
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

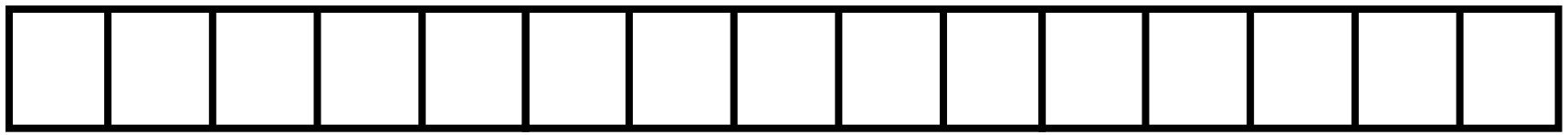
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

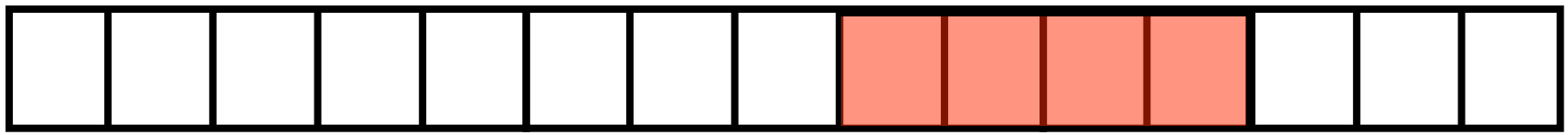
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

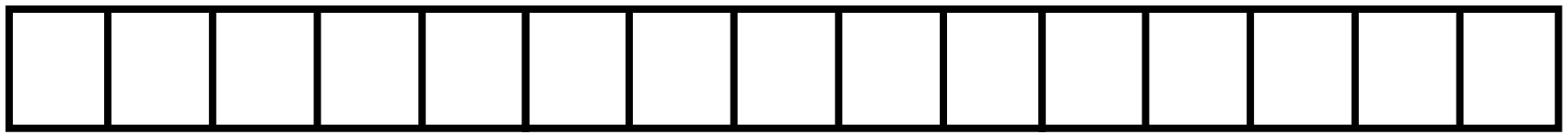
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

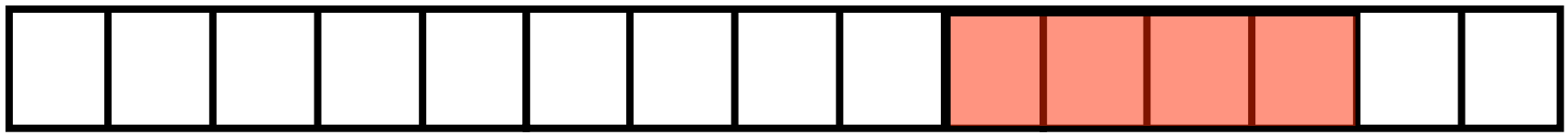
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

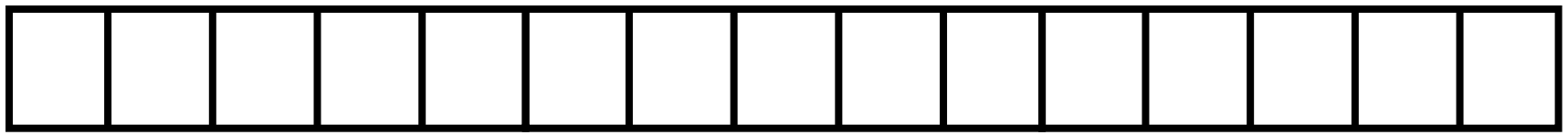
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

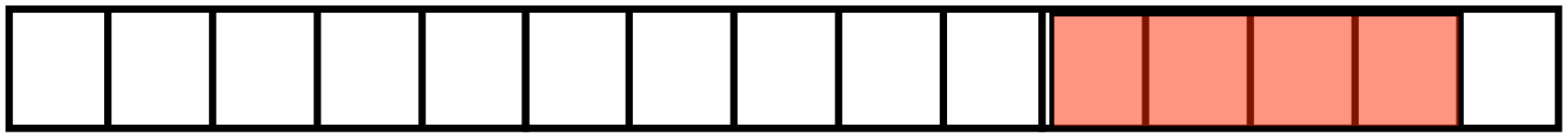
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

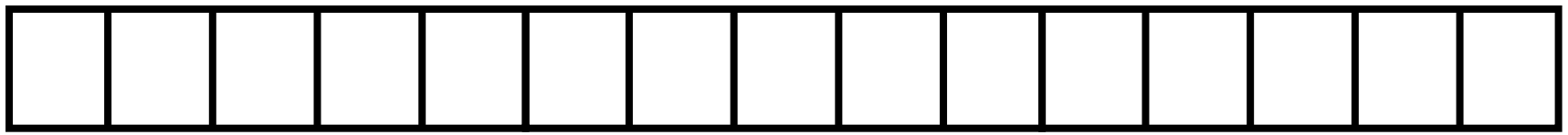
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

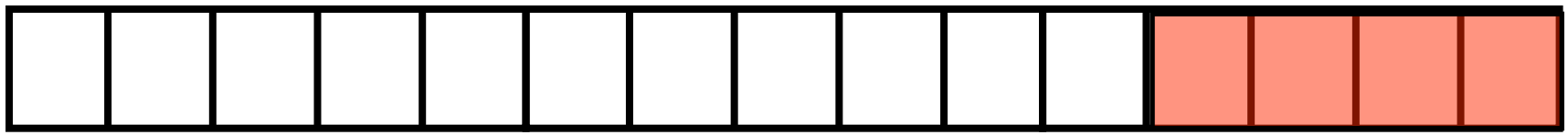
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

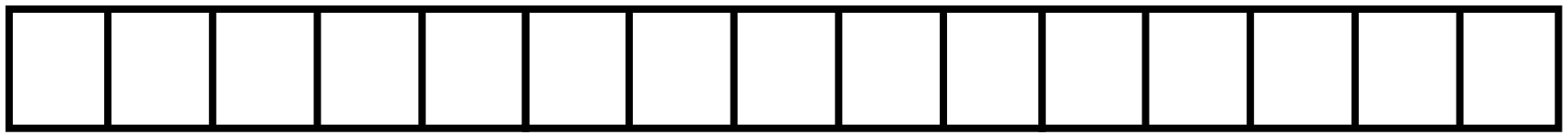
- bei bekannter maximaler Größe max: Array



Implementierung von Datent.

Implementierung von Queues - sequent.

- bei bekannter maximaler Größe max: Array



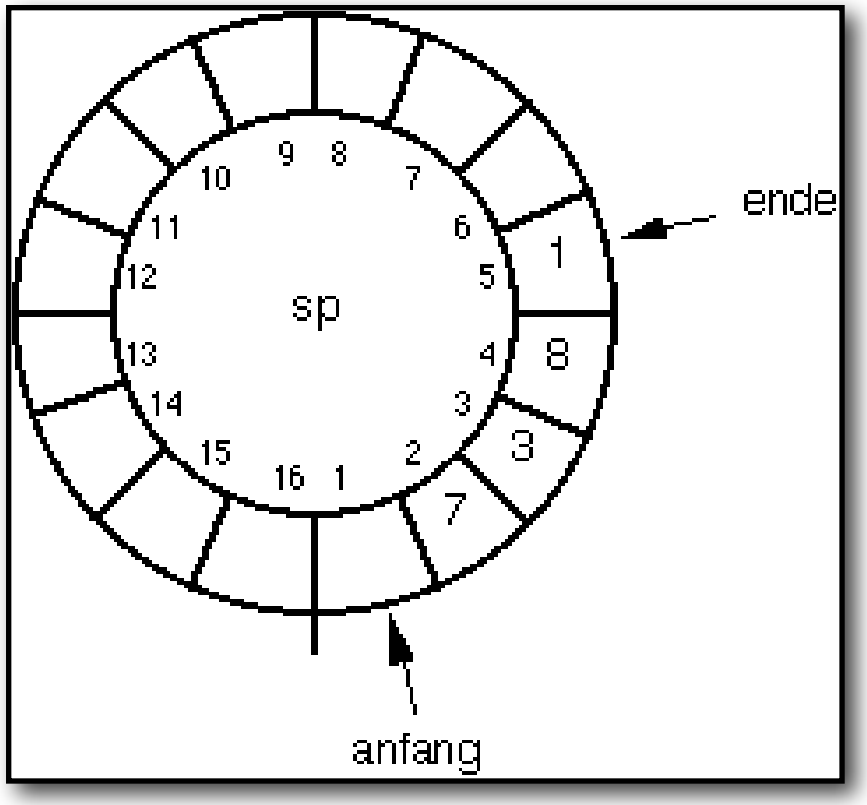
Implementierung von Datent.

Implementierung von Queues - sequent.

- bei bekannter maximaler Größe max: Array

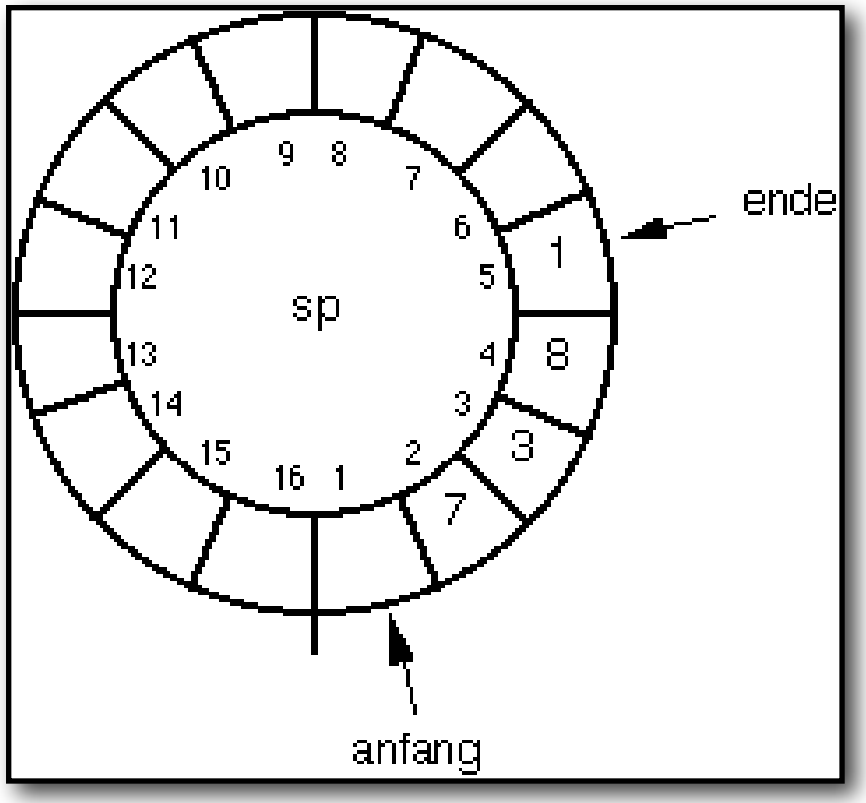
Implementierung von Datent.
Implementierung von Queues - sequent.

- bei bekannter maximaler Größe max: Array



Implementierung von Datent.
Implementierung von Queues - sequent.

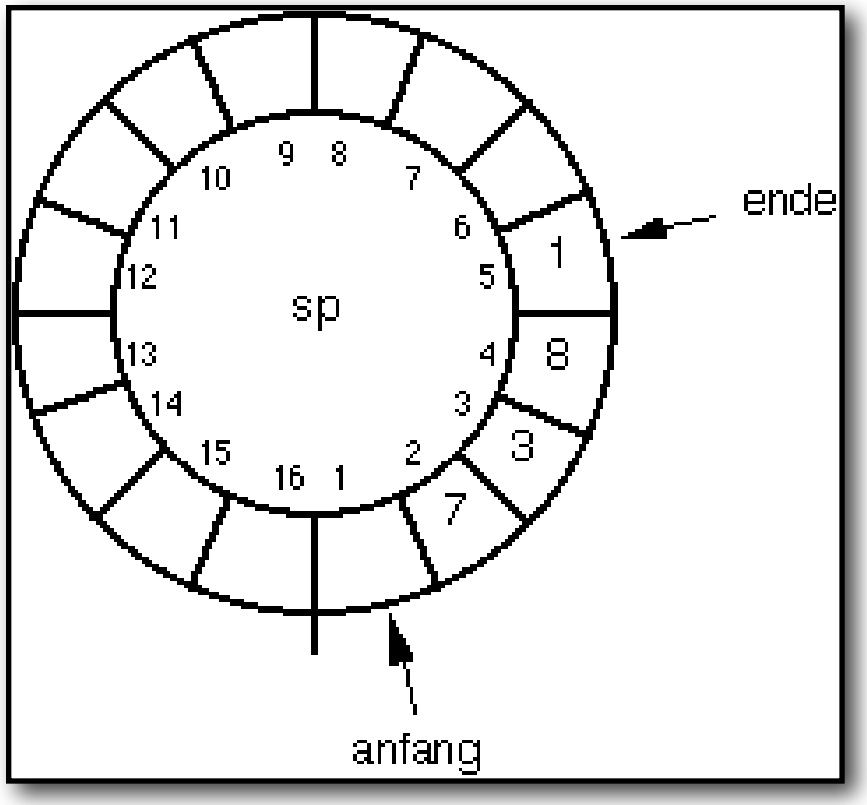
- bei bekannter maximaler Größe max: Array




```
type data=...;
    queue=record
            anfang, ende: 1..max;
            sp: array [1..max] of data
          end;
```

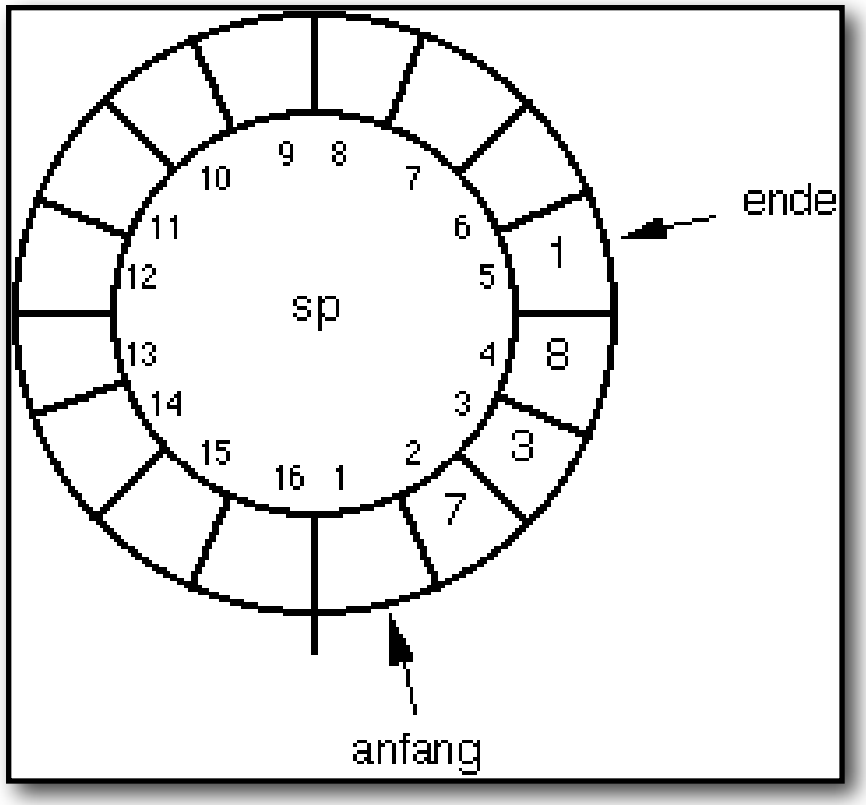
Implementierung von Datent.
Implementierung von Queues - sequent.

- bei bekannter maximaler Größe max: Array



Implementierung von Datent.
Implementierung von Queues - sequent.

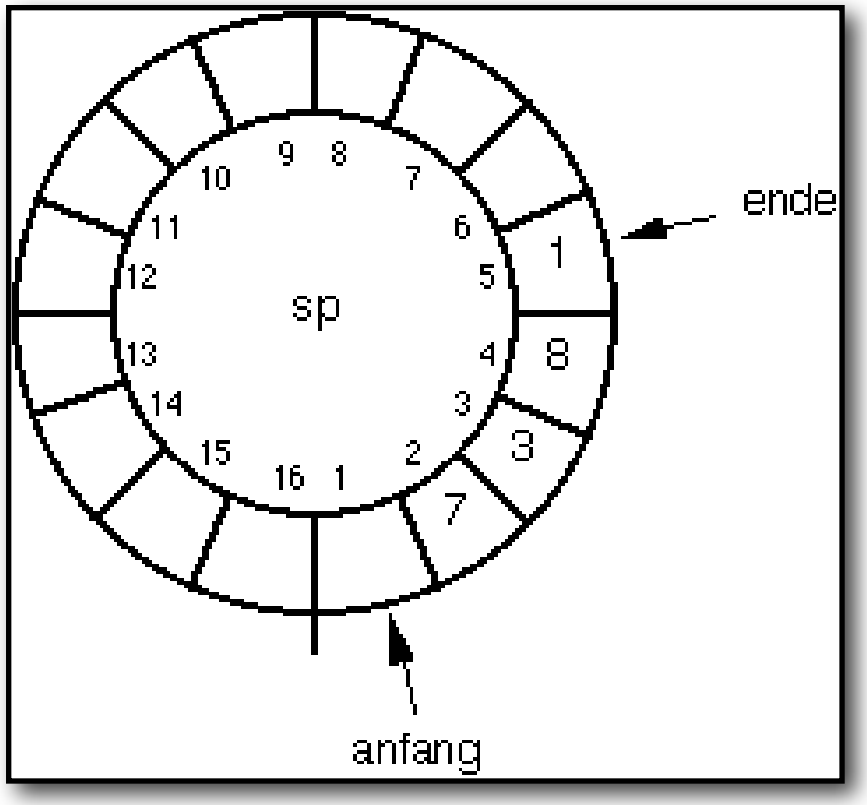
- bei bekannter maximaler Größe max: Array




```
procedure enter(x: data; var q: queue);
begin
      with q do
      begin
            if ende=max then ende:=1
                  else ende:=ende+1;
            if ende=anfang then "Overflow
                  else sp[ende]:=x
      end
end;
```

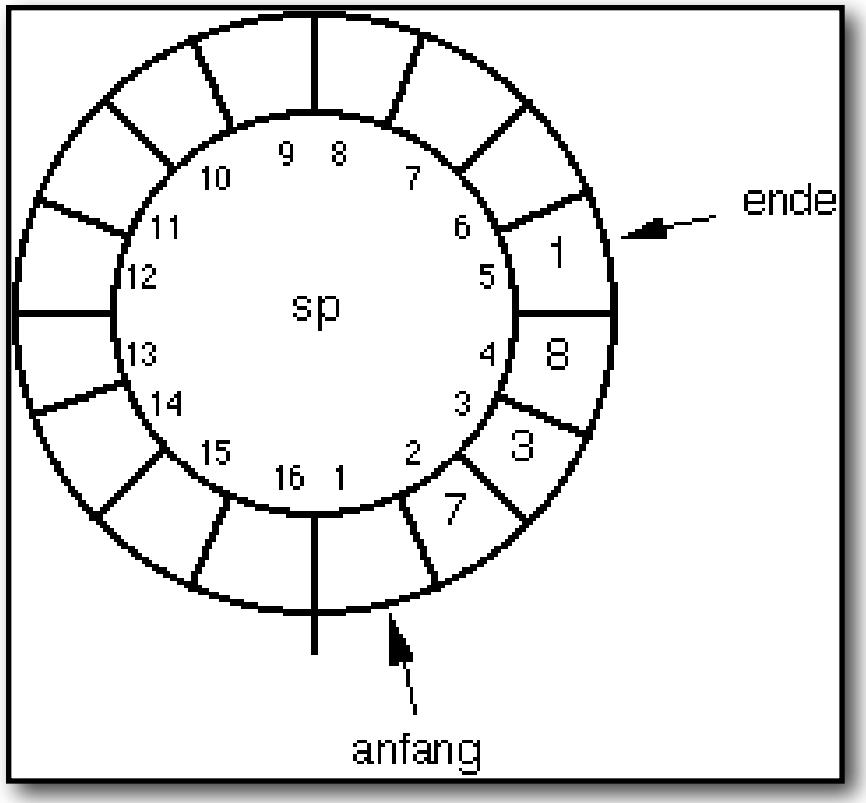
Implementierung von Datent.
Implementierung von Queues - sequent.

- bei bekannter maximaler Größe max: Array



Implementierung von Datent.
Implementierung von Queues - sequent.

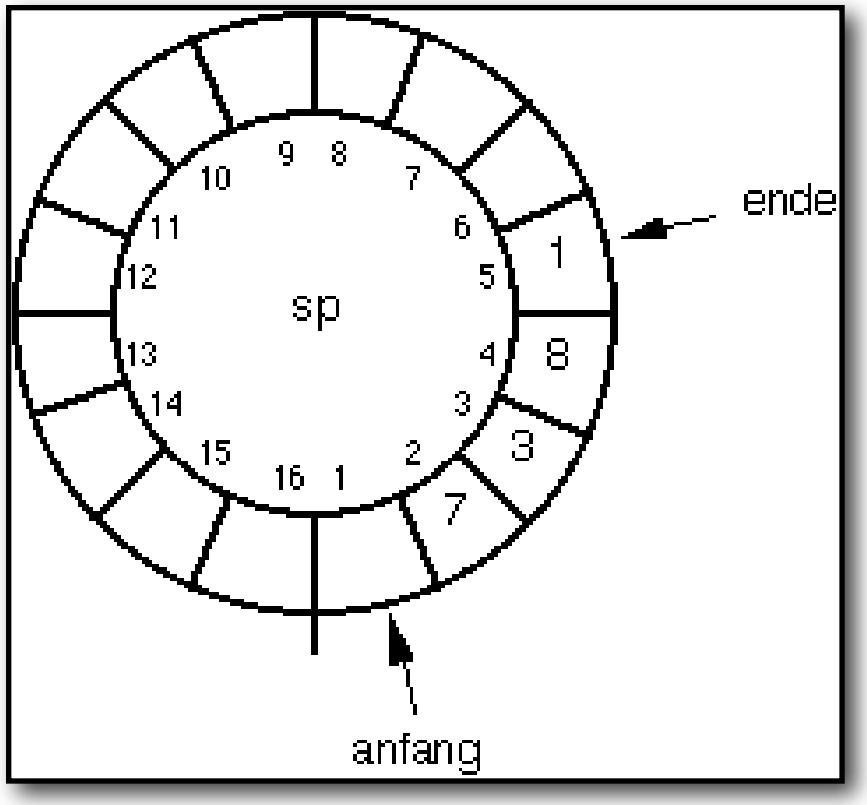
- bei bekannter maximaler Größe max: Array




```
procedure remove(var q: queue);
begin
      with q do
            if is_empty(q) then "Underflow" else
                  if anfang=max then anfang:= 1
                        else anfang:=anfang+ 1
end;
```

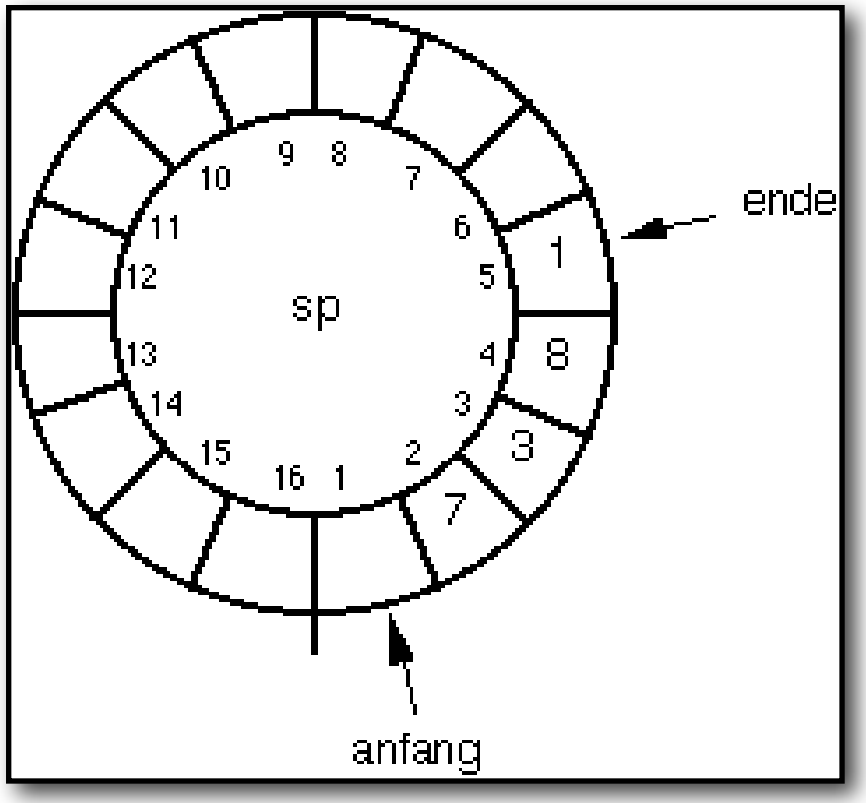
Implementierung von Datent.
Implementierung von Queues - sequent.

- bei bekannter maximaler Größe max: Array



Implementierung von Datent.
Implementierung von Queues - sequent.

- bei bekannter maximaler Größe max: Array



Initialisierung:
var q: queue; empty(q).

Implementierung von Datent.
Stacks und Queues - Overflows

- Fehlersituationen:
  - Overflow: Datenstruktur voll, aber man will noch ein Objekt einfügen (Programm oder Speicherknappheit); Lösung: max vergrößern
  - Underflow: Datenstruktur leer, aber man will noch ein Objekt auslesen (Programmfehler)
- Grundproblem: "Man ist nicht alleine" → viele Stacks zugleich im Speicher (wie verwalten?)

Implementierung von Datent.
Stacks und Queues - Multi-Stacks

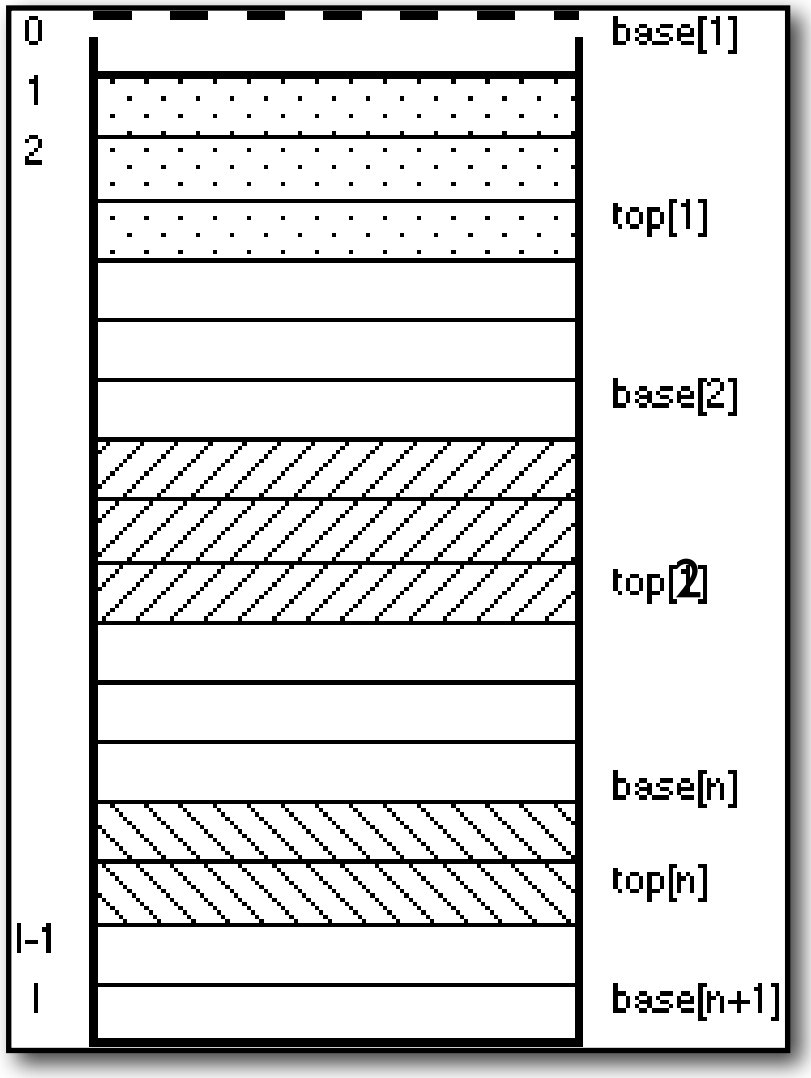
- Grundannahme:

  type Speicher=array [1..l] of data.

- n homogene Stacks sequentiell angeordnet
- keine feste Basis mehr (verschieblich)

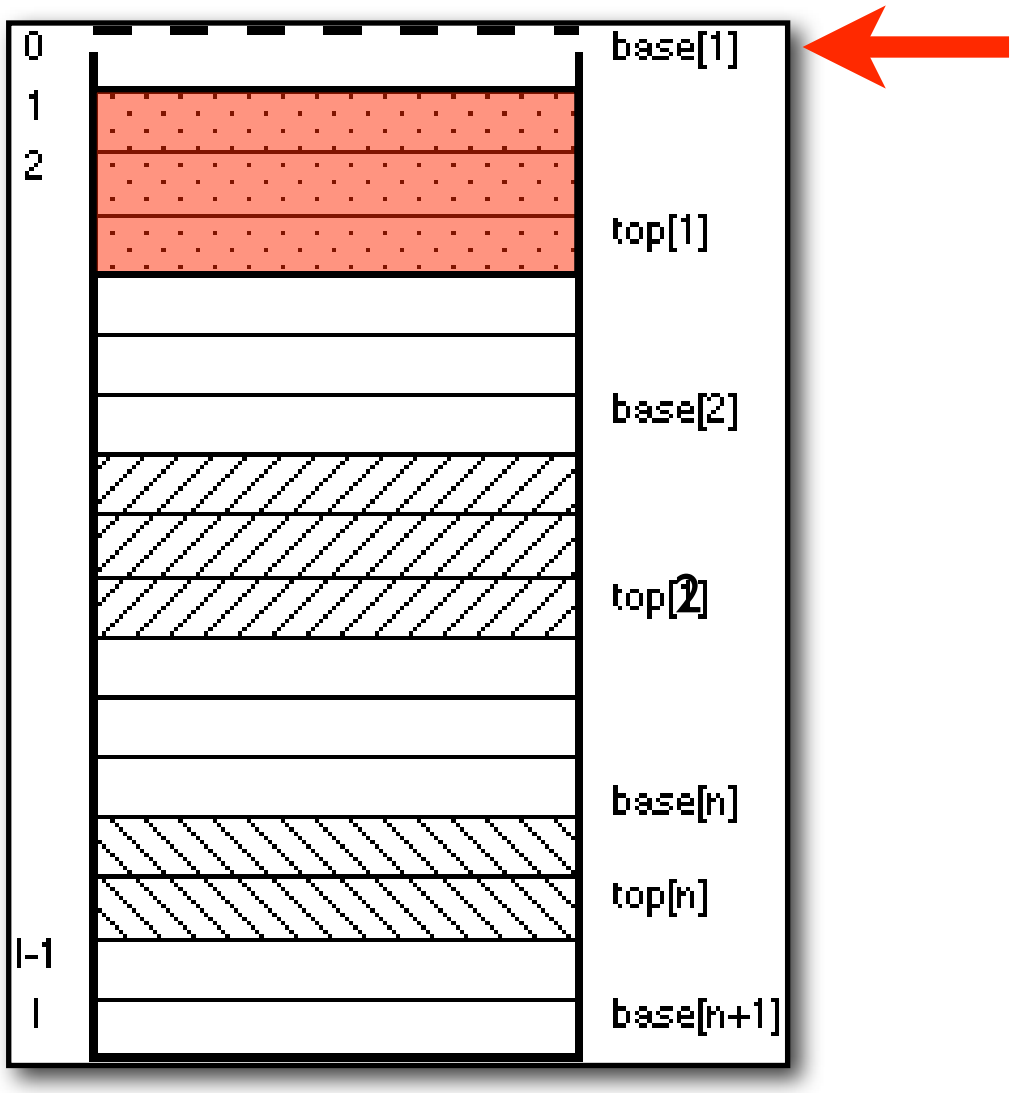
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



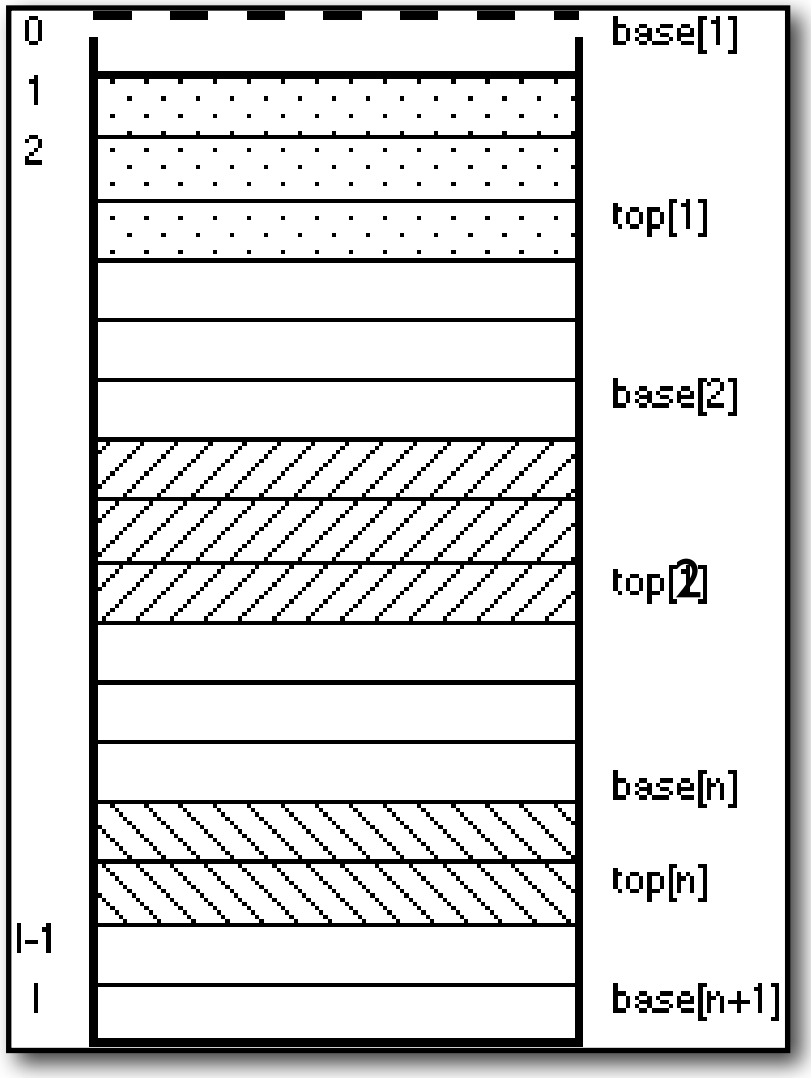
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



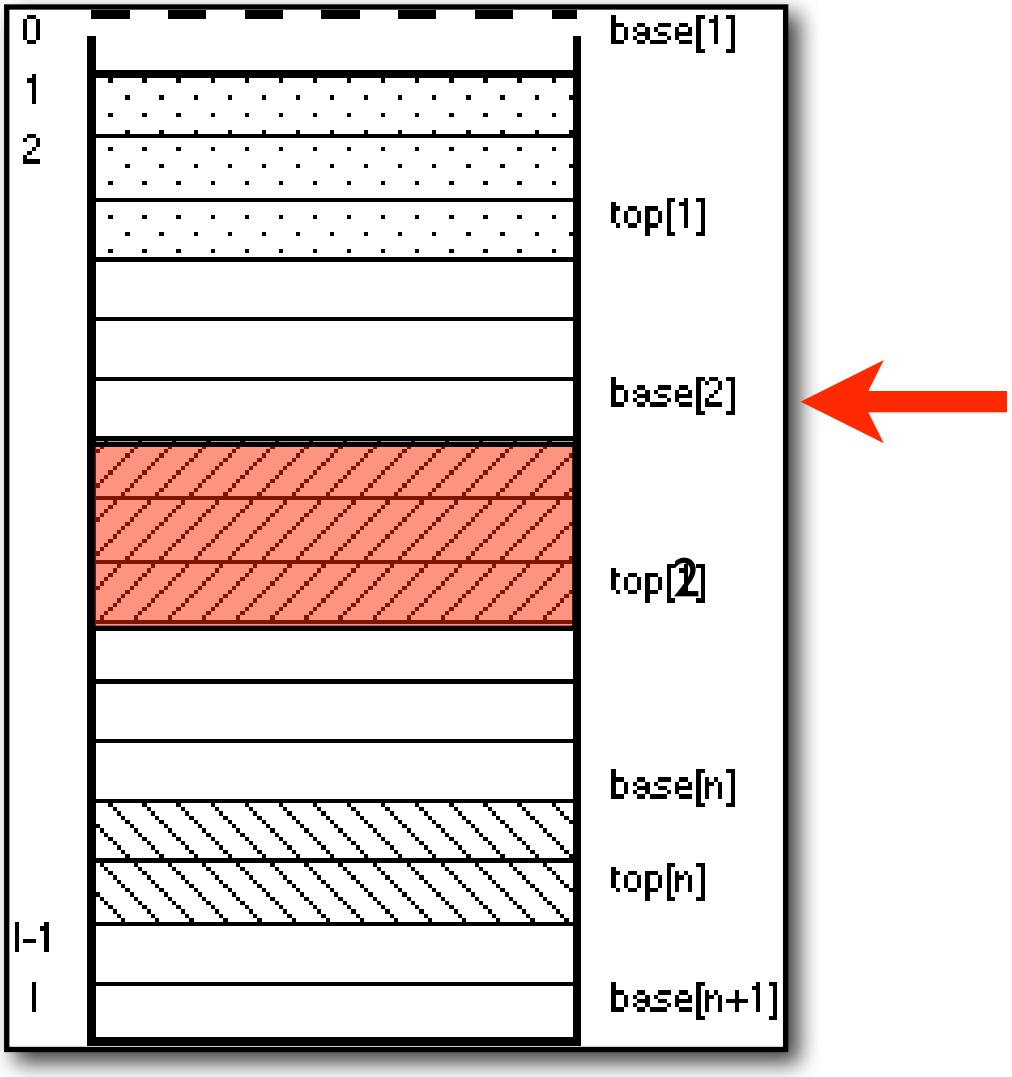
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



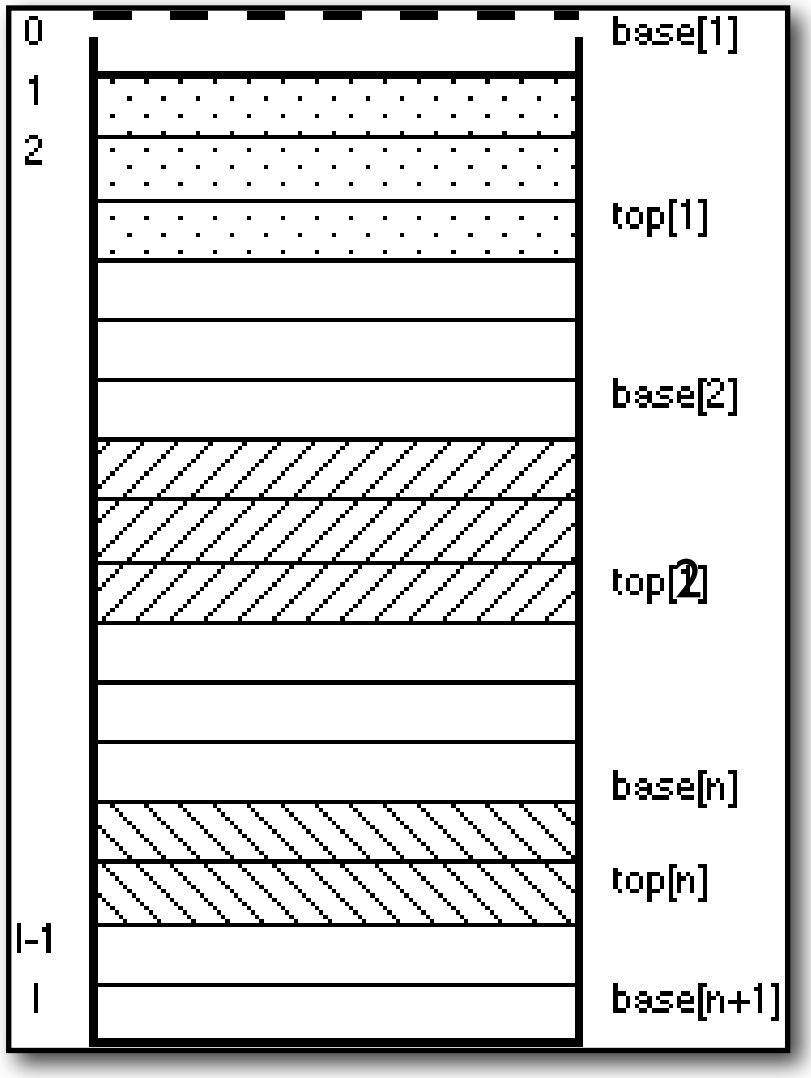
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



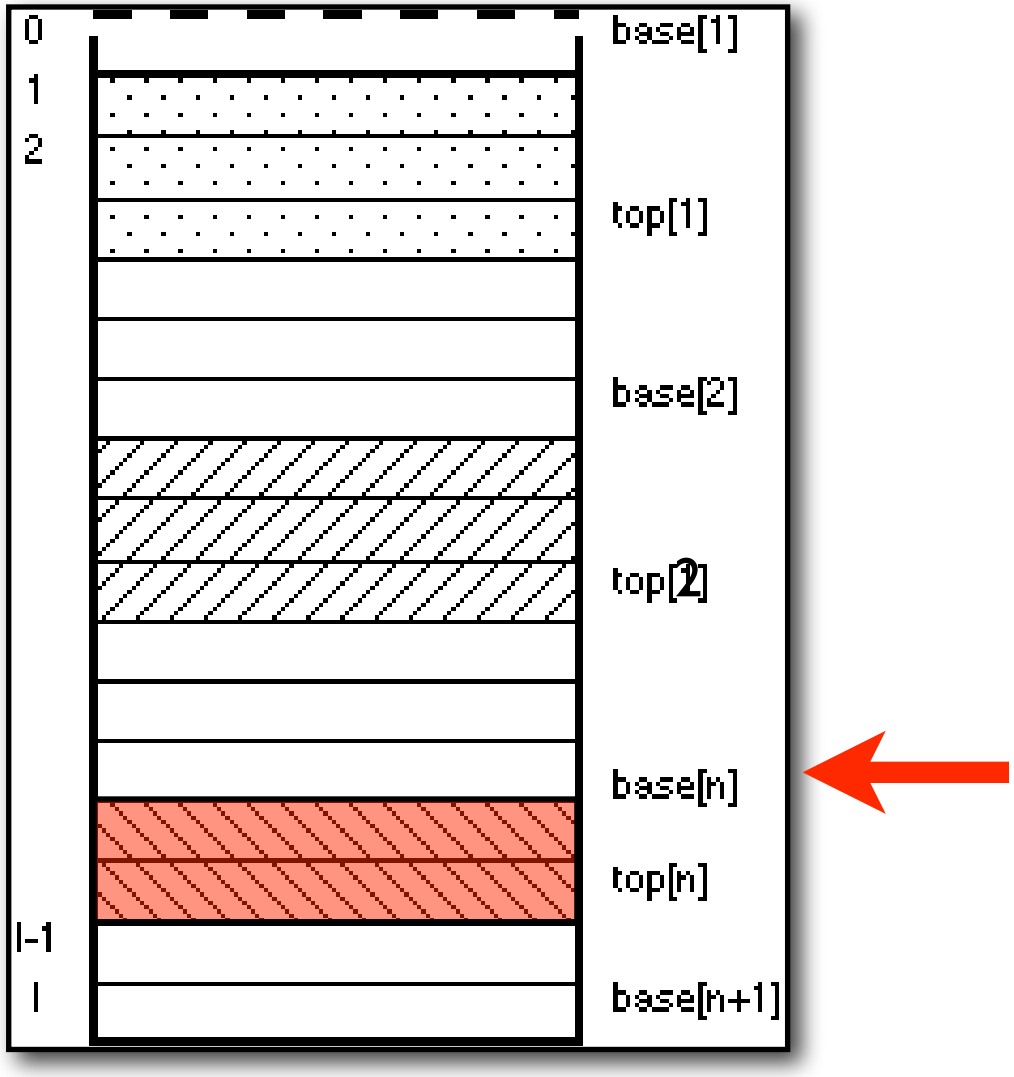
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



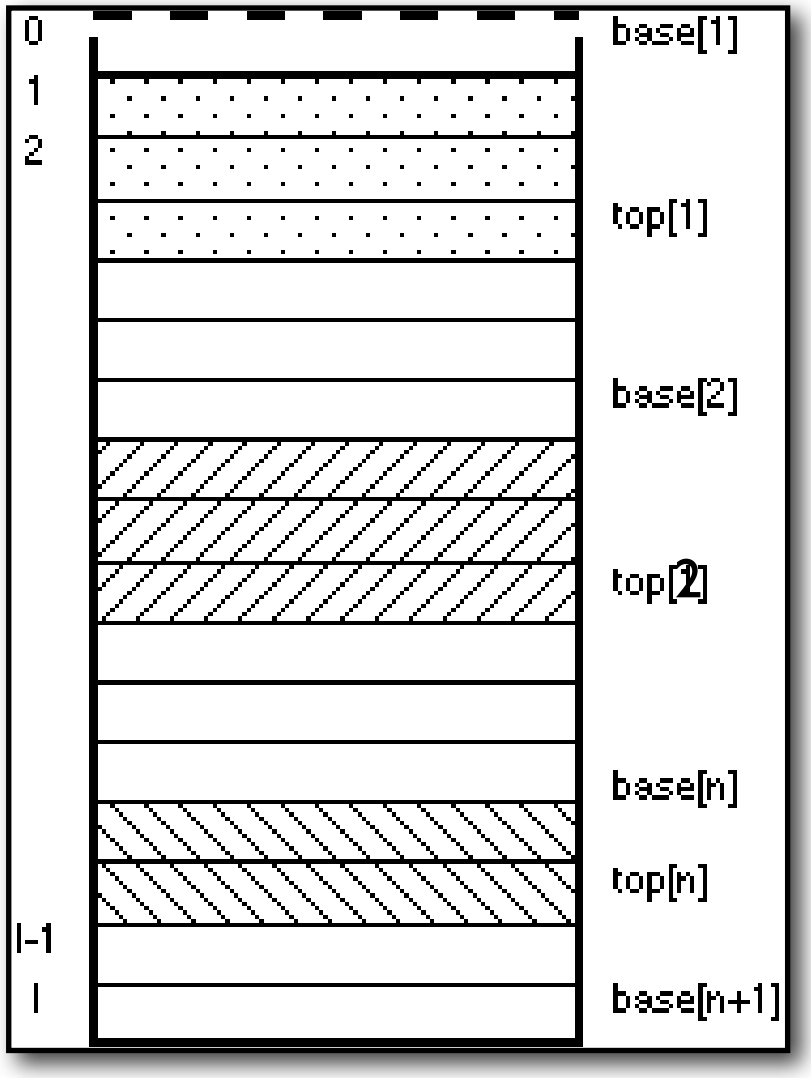
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



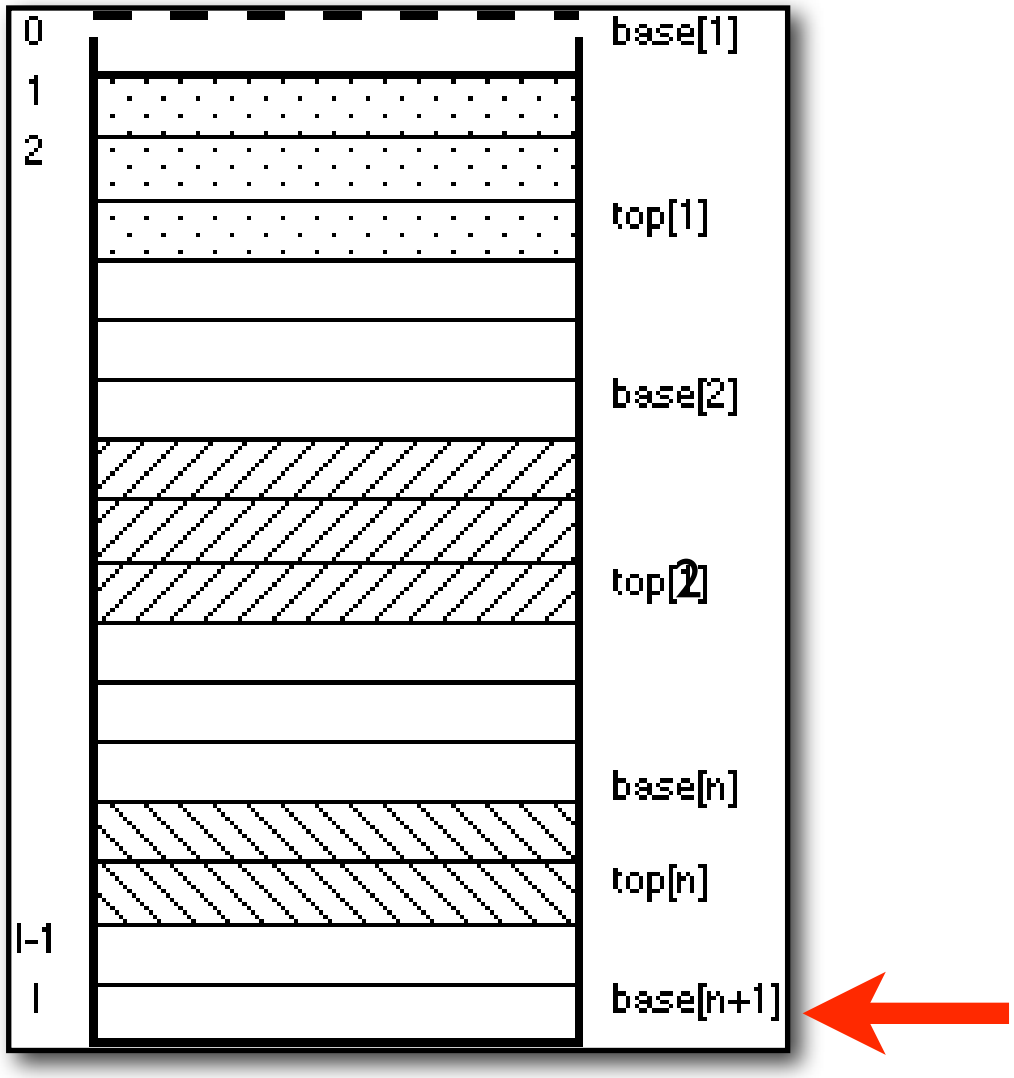
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



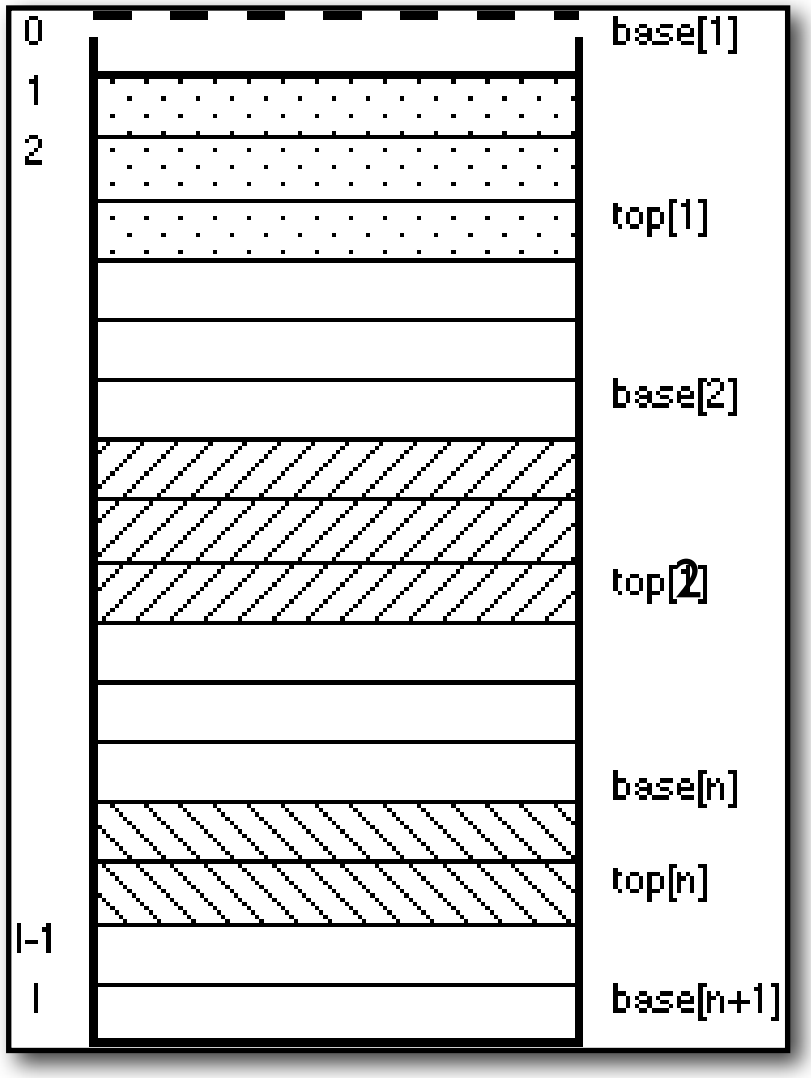
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



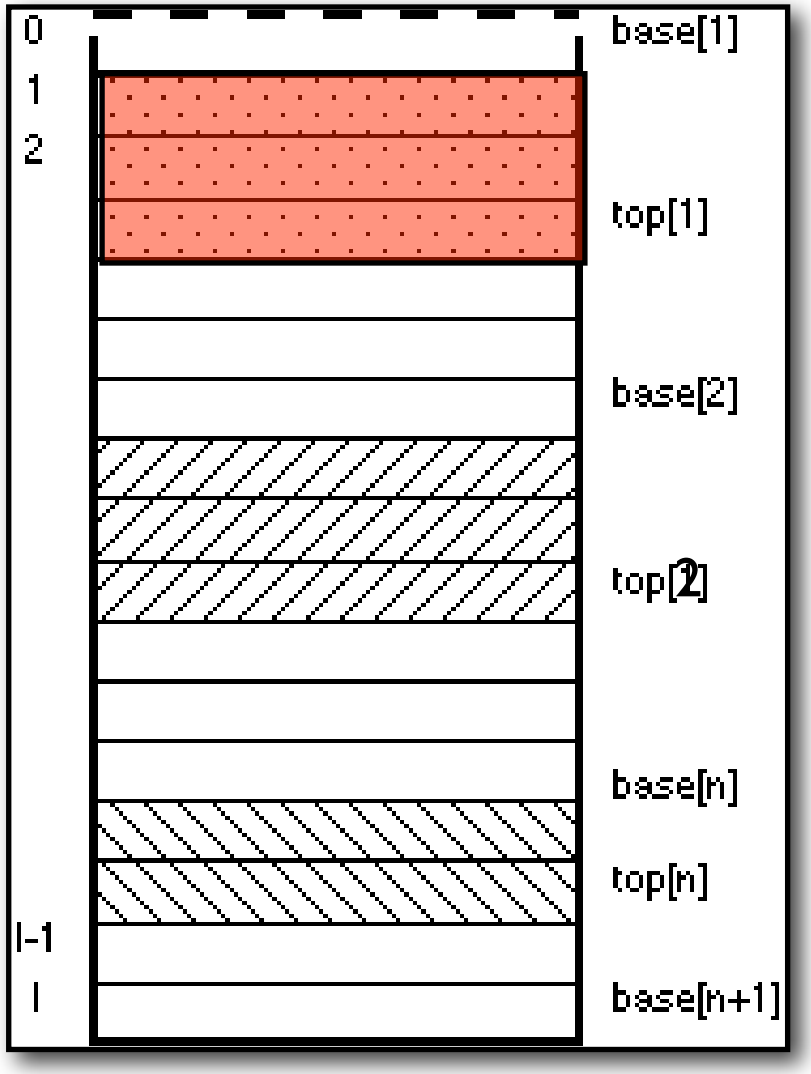
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



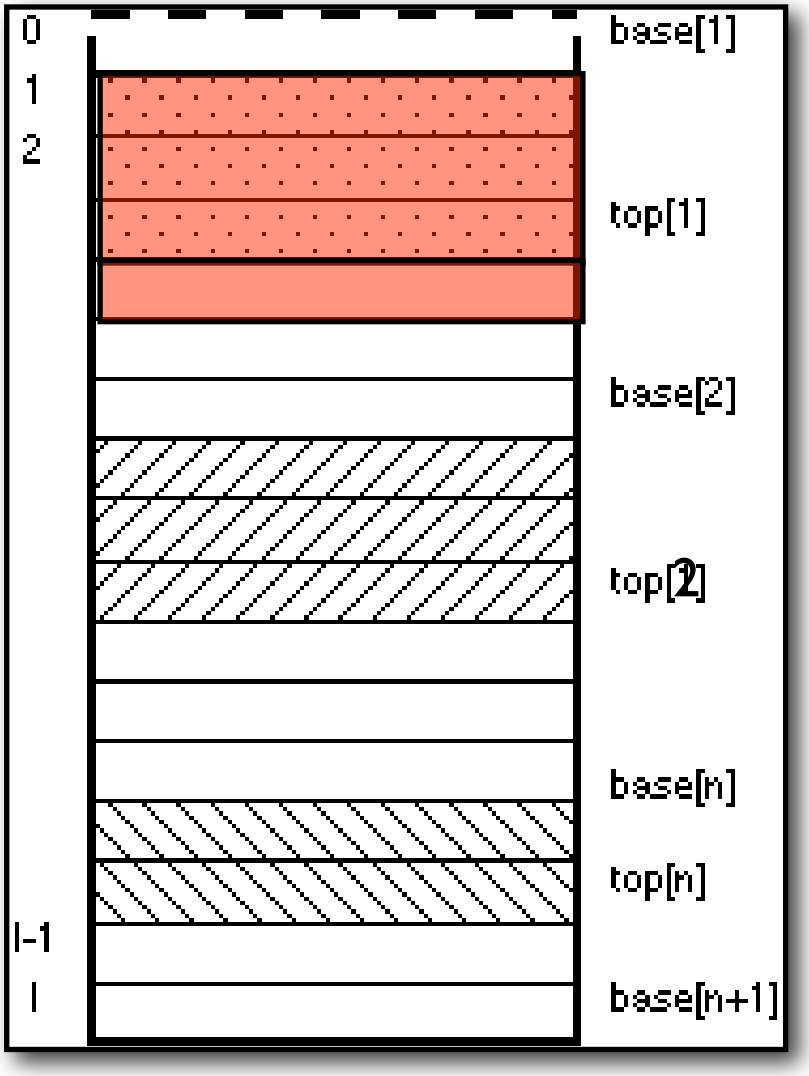
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



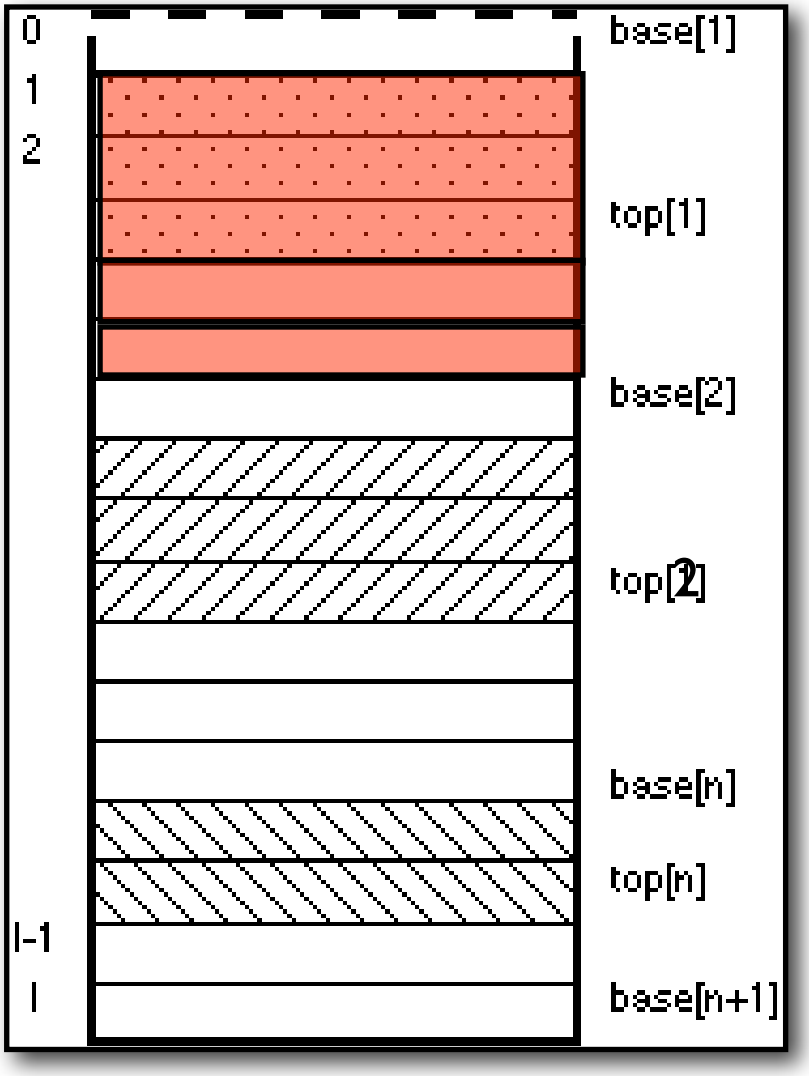
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



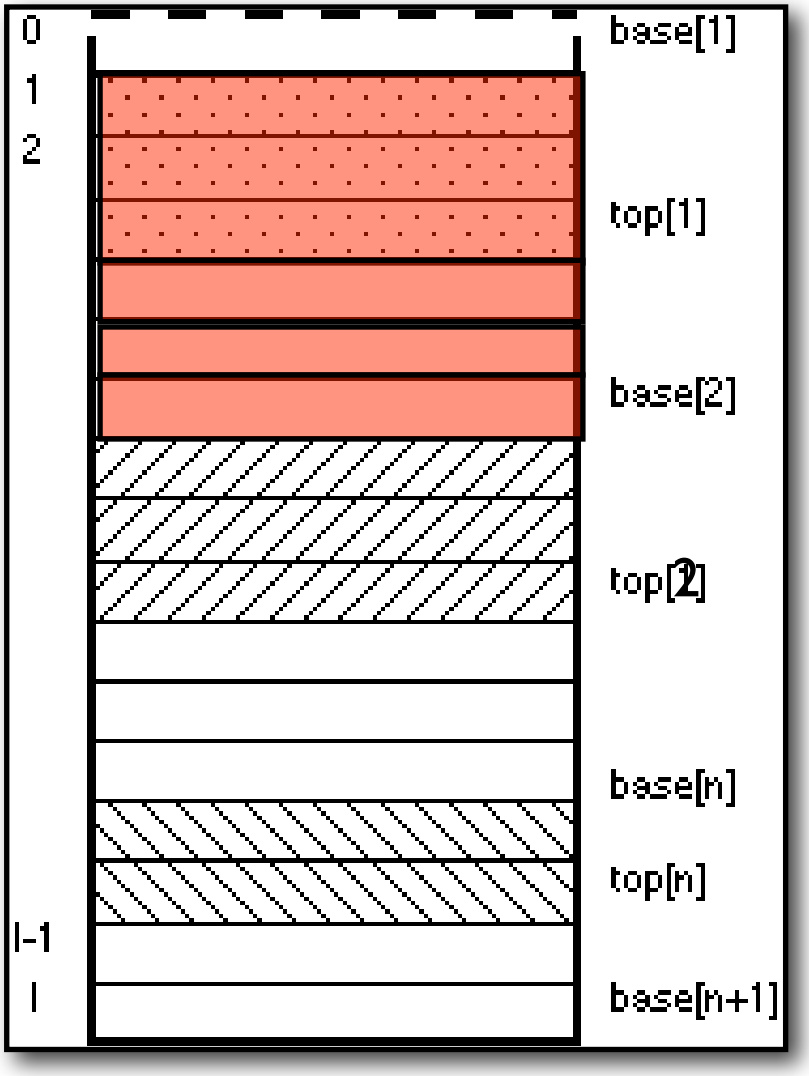
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



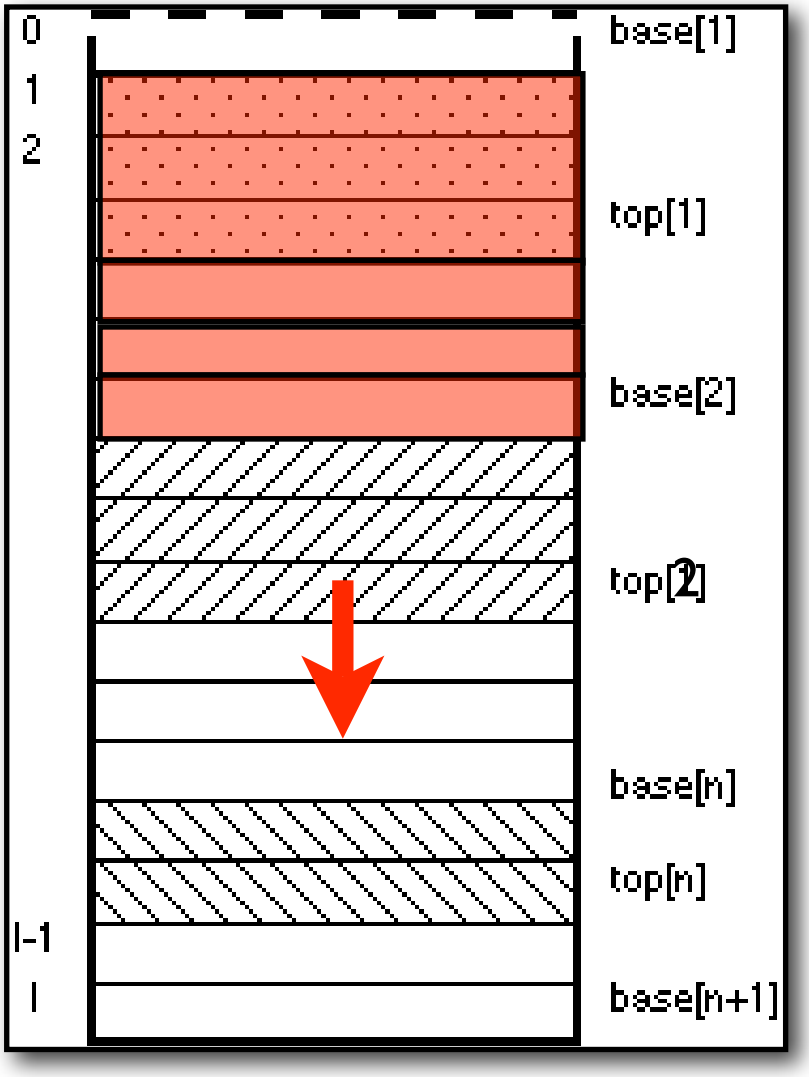
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



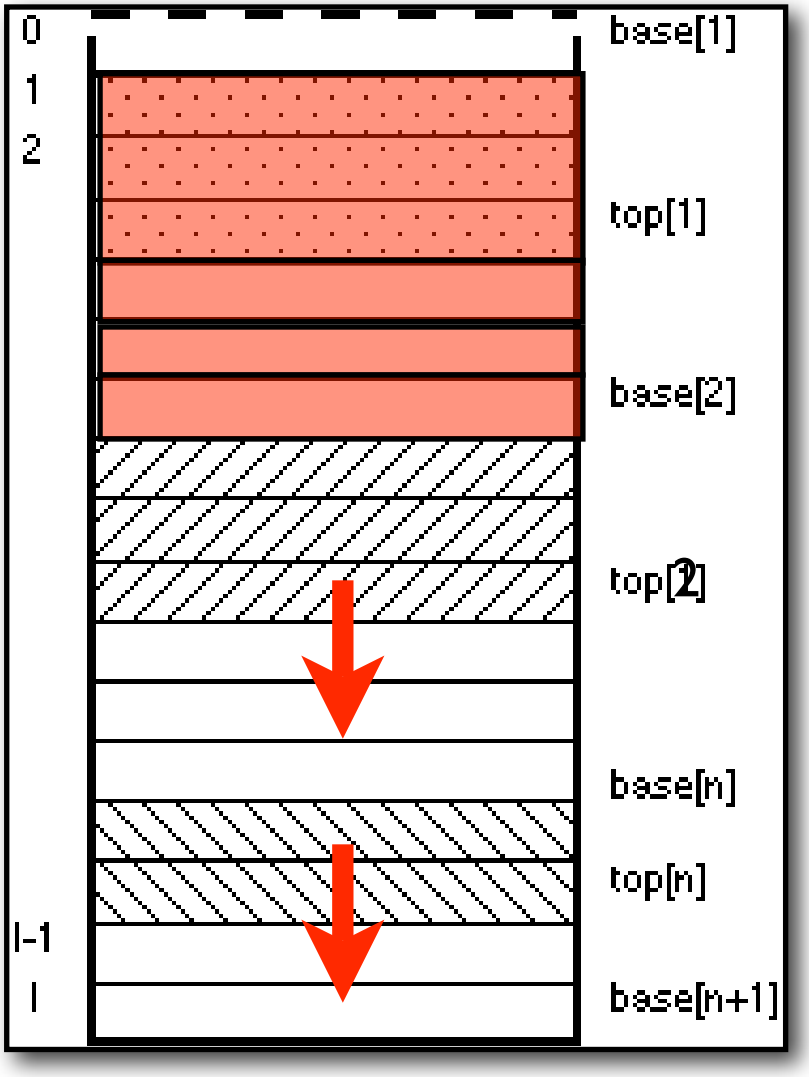
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



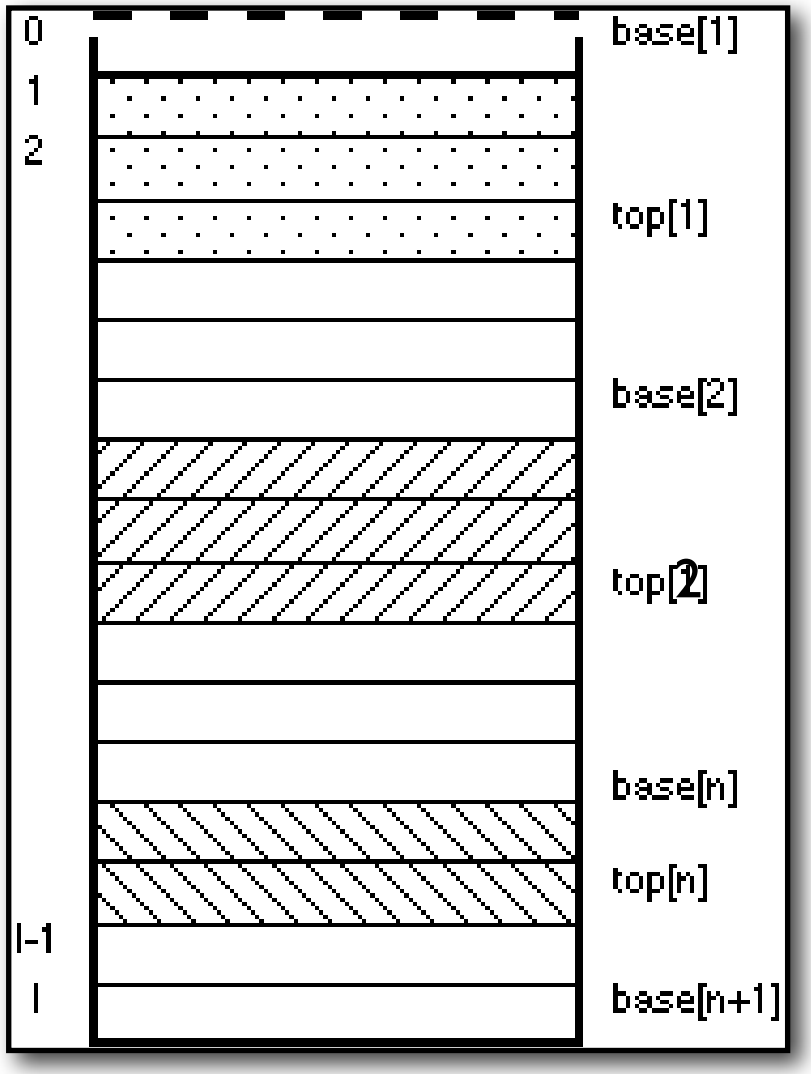
Implementierung von Datent.

Stacks und Queues - Multi-Stacks



Implementierung von Datent.

Stacks und Queues - Multi-Stacks



Modifikationen:

- Overflow: top[i]>base[i+1]
- empty: Verteile Stacks anfangs gleichmäßig über Speicher
- Einzelheiten im Skript

Implementierung von Datent.
Multi-Stacks - Overflow-Problematik

- Annahme: Overflow bei Stack i, aber noch Speicher vorhanden
  - Naive Idee:
    - Stacks umverteilen, damit für Stack i eine Zelle frei wird
    - Nachteil: beim nächsten Schritt vielleicht schon wieder Umordnung (Aufwand!)
  - Besser: Schiebe nächste Umverteilung soweit wie möglich in die Zukunft (**Garwick-Algorithmus**)

Implementierung von Datent.
Multi-Stacks - Garwick-Algorithmus

- Strategie: freie Zellen sind Gemeingut
  - Weise den Stacks je Umordnung *mehrere* Zellen zu oder nimm sie ihnen weg
  - schnell gewachsene Stacks erhalten mehr Zellen als langsam gewachsene
- Grundlage dieser Strategie: **Lokalitätsprinzip**

Programme verhalten sich innerhalb einer gewissen zukünftigen Zeitspanne etwa so, wie sie sich innerhalb ihrer unmittelbaren Vergangenheit verhalten haben

Implementierung von Datent.
Multi-Stacks - Lokalitätsprinzip

- Anschaulich:
  - Ist Stack i stark gewachsen, wächst er ziemlich sicher weiter stark
  - Ist Stack i sparsam, bleibt er ziemlich sicher weiter sparsam
- Warum überhaupt? → viel Beobachtung, Programmtextanalyse, Statistik, ...
- Garwick-Algorithmus: Verteile
  - 10% der freien Zellen gleichmäßig
  - 90% der freien Zellen nach Wachstum seit letzter Umordnung

Implementierung von Datent.
Multi-Stacks - Garwick

```
type n_stack=record
   ... <wie bei Multi-Stacks>
   oldtop: array [anzahl] of 0..l;
   newbase: array [1..n+1] of 0..l;
   d: array [anzahl] of 0..l
end;
```

Algorithmus: s. Skript

Implementierung von Datent.
Multi-Stacks - Garwick

```
type n_stack=record
   ... <wie bei Multi-Stacks>
   oldtop: array [anzahl] of 0..l;
   newbase: array [1..n+1] of 0..l;
   d: array [anzahl] of 0..l
end;
```

top-Werte nach der letzten Umordnung

Algorithmus: s. Skript

Implementierung von Datent.
Multi-Stacks - Garwick

```
type n_stack=record
   ... <wie bei Multi-Stacks>
   oldtop: array [anzahl] of 0..l;
   newbase: array [1..n+1] of 0..l;
   d: array [anzahl] of 0..l
end;
```

Algorithmus: s. Skript

Implementierung von Datent.
Multi-Stacks - Garwick

```
type n_stack=record
   ... <wie bei Multi-Stacks>
   oldtop: array [anzahl] of 0..l;
   newbase: array [1..n+1] of 0..l;
   d: array [anzahl] of 0..l
end;
```

neue Basen der Stacks (werden bei Umordnung berechnet

Algorithmus: s. Skript

Implementierung von Datent.
Multi-Stacks - Garwick

```
type n_stack=record
   ... <wie bei Multi-Stacks>
   oldtop: array [anzahl] of 0..l;
   newbase: array [1..n+1] of 0..l;
   d: array [anzahl] of 0..l
end;
```

Algorithmus: s. Skript

Implementierung von Datent.
Multi-Stacks - Garwick

```
type n_stack=record
   ... <wie bei Multi-Stacks>
   oldtop: array [anzahl] of 0..l;
   newbase: array [1..n+1] of 0
   d: array [anzahl] of 0..l
end;
```

pos. Wachstum jedes Stacks

Algorithmus: s. Skript

Implementierung von Datent.
Multi-Stacks - Garwick

```
type n_stack=record
   ... <wie bei Multi-Stacks>
   oldtop: array [anzahl] of 0..l;
   newbase: array [1..n+1] of 0..l;
   d: array [anzahl] of 0..l
end;
```

Algorithmus: s. Skript

Implementierung von Datent.

Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.

Multi-Stacks - Garwick


```
vara,b: r
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.

Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.

Multi-St[illegible] Garwick

Summe aller pos. Wachstumszahlen

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.

Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
    ...
for j:=1 to n do
begin
    sum:=sum-(top[j]-base[j]);
    if top[j]>oldtop[j] then
    begin
        d[j]:=top[j]-oldtop[j];
        inc:=inc+d[j]
    end else d[j]:=0;
end;
```

je Stack belegte Zellen von sum abziehen

Implementierung von Datent.

Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
    ...
for j:=1 to n do
begin
    sum:=sum-(top[j]-base[j]);
    if top[j]>oldtop[j] then
    begin
        d[j]:=top[j]-oldtop[j];
        inc:=inc+d[j]
    end else d[j]:=0;
end;
```

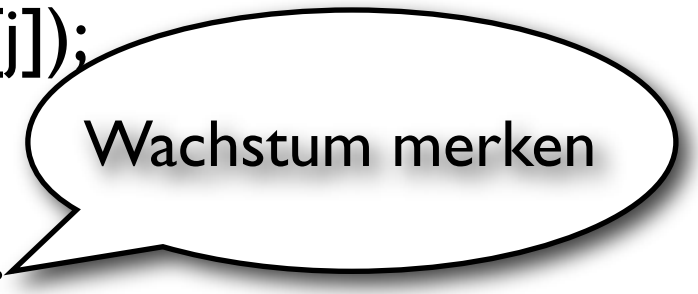
Implementierung von Datent.

Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```



Implementierung von Datent.

Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.

Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.

Multi-Stacks - Garwick

```
vara,b: real;
        sum, inc: integer;
        j: 1..n;
   ...
for j:=1 to n do
begin
   sum:=sum-(top[j]-base[j]);
   if top[j]>oldtop[j] then
   begin
      d[j]:=top[j]-oldtop[j];
      inc:=inc+d[j]
   end else d[j]:=0;
end;
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.

Multi-St

Fehlersituation

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.

Multi-Stacks - Garwick


```
if sum<0 th                                    ist voll" else
begin
  a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.

Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speich
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

wachstumsbezogener Anteil freier Zellen je Stack

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

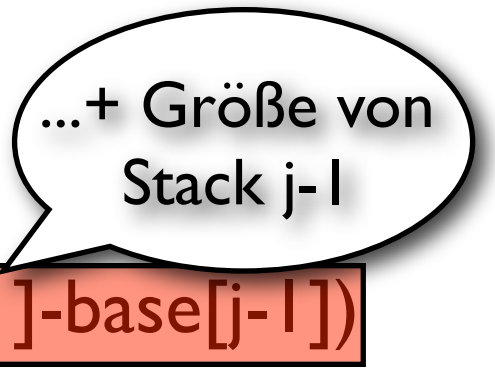
```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```



Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

... + fester Anteil

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+base[j]-base[j-1]
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

... + Wachstums-zuschlag

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

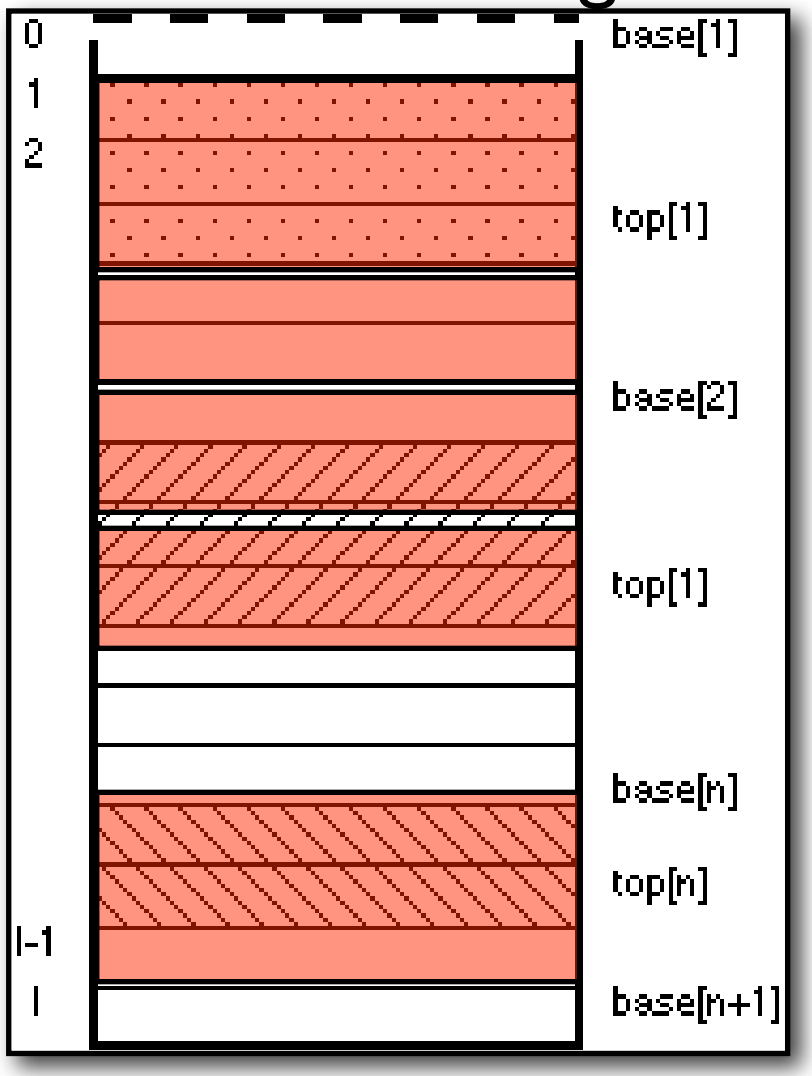
Implementierung von Datent.
Multi-Stacks - Garwick

```
if sum<0 then "Fehler: Speicher ist voll" else
begin
   a:=(0.1*sum)/n; b:=(0.9*sum)/inc;
   newbase[1]:=base[1];
   for j:=2 to n do
      newbase[j]:=newbase[j-1]+(top[j-1]-base[j-1])
      +trunc(a)+trunc(b*d[j-1]);
   umordnen(s);
   for j:=1 to n do oldtop[j]:=top[j]
end
```

Implementierung von Datent.

Multi-Stacks - Umordnung

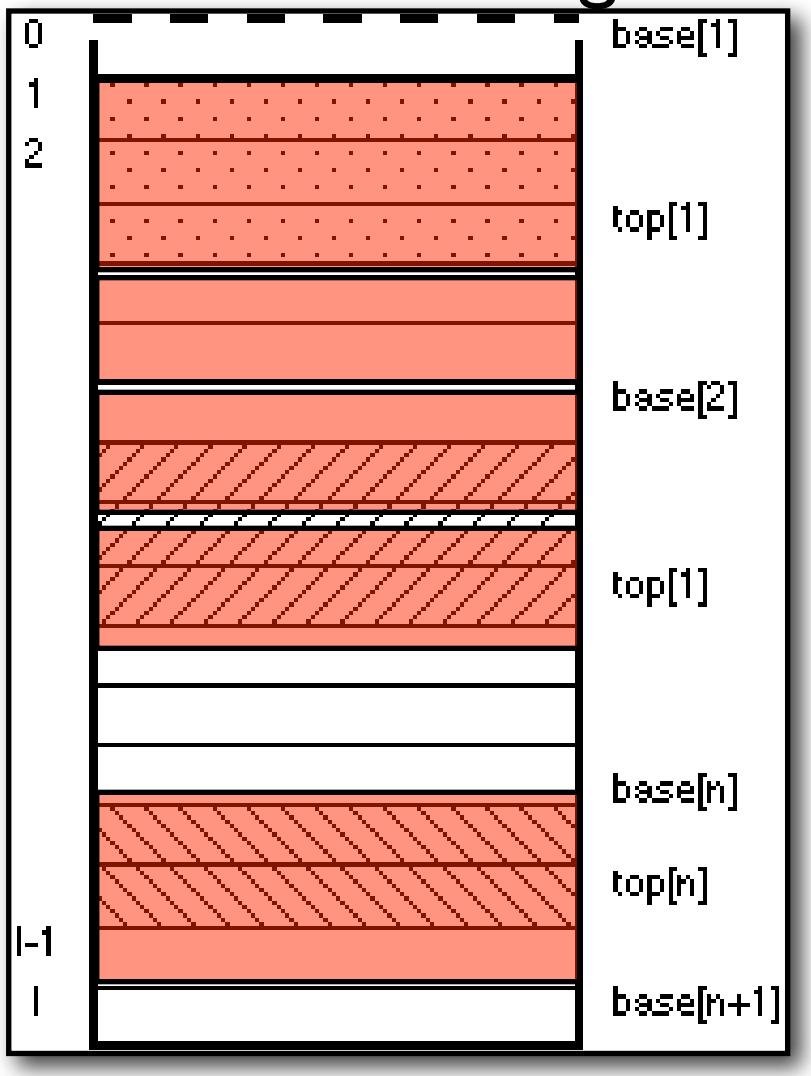
vor Umordnung



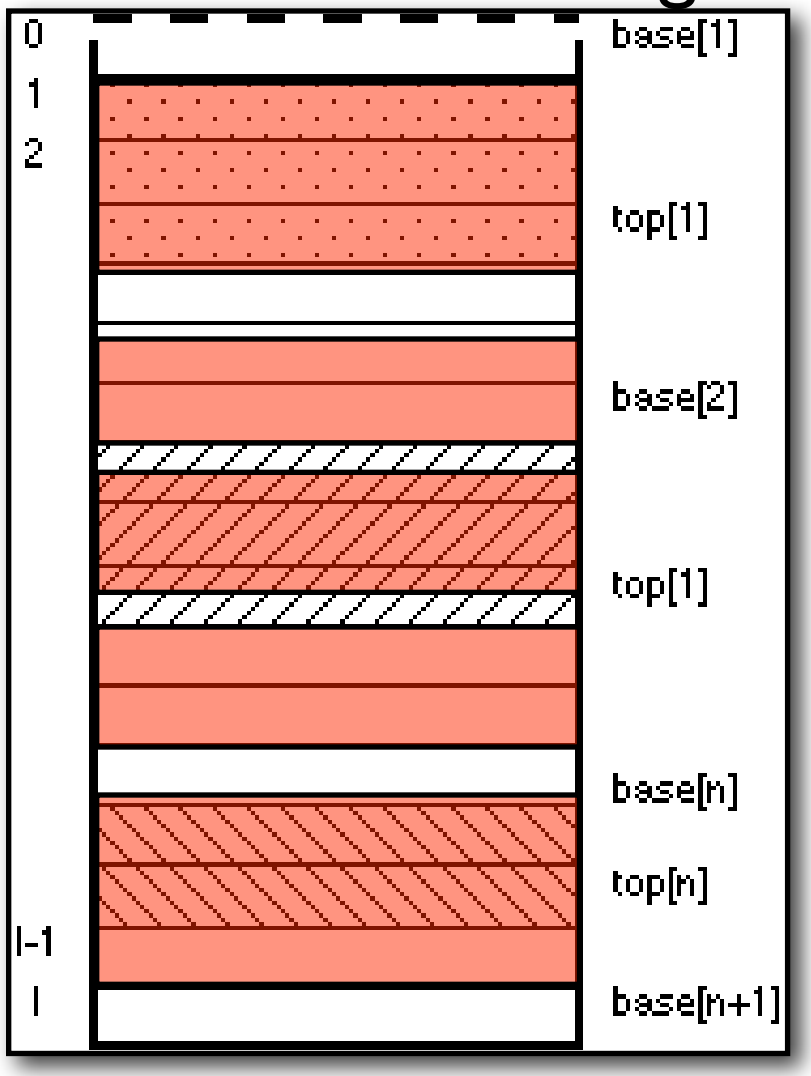
Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung



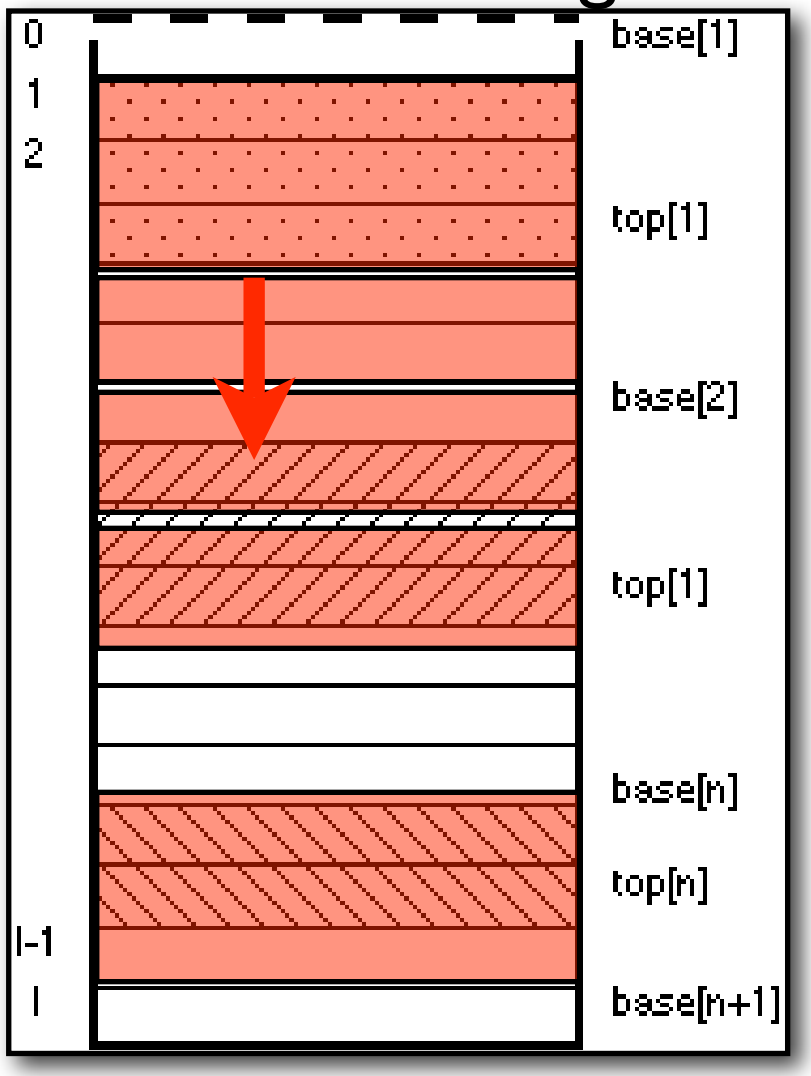
nach Umordnung



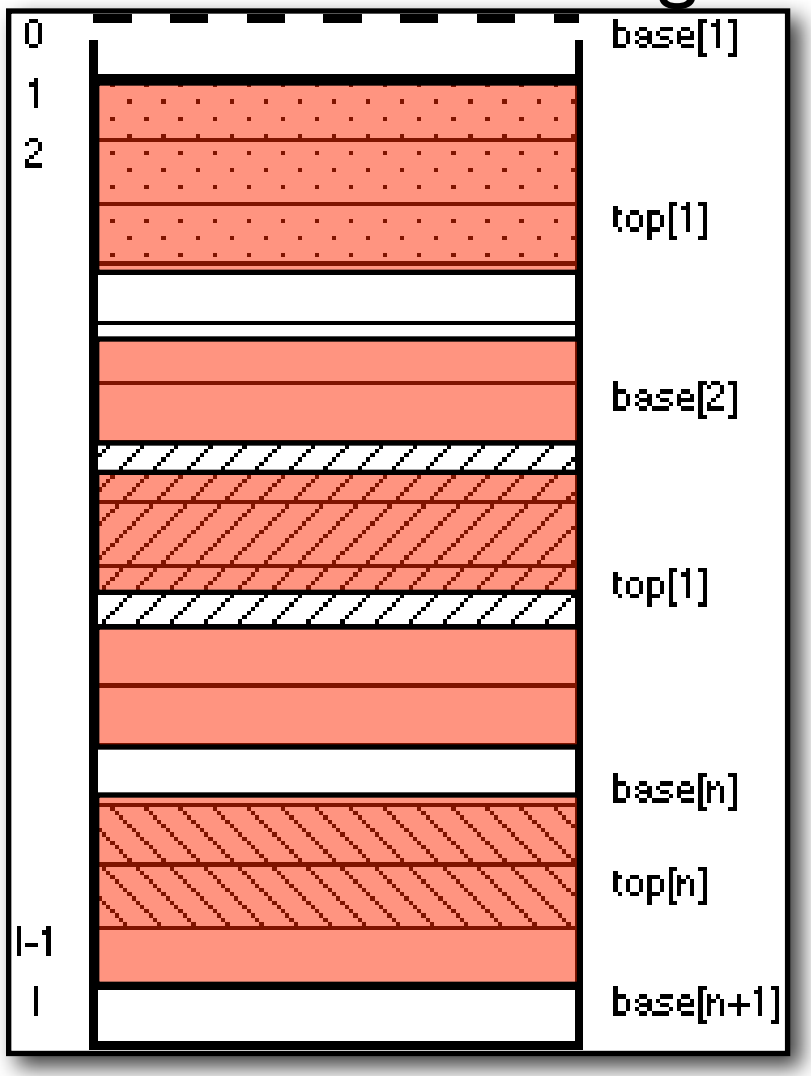
Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung



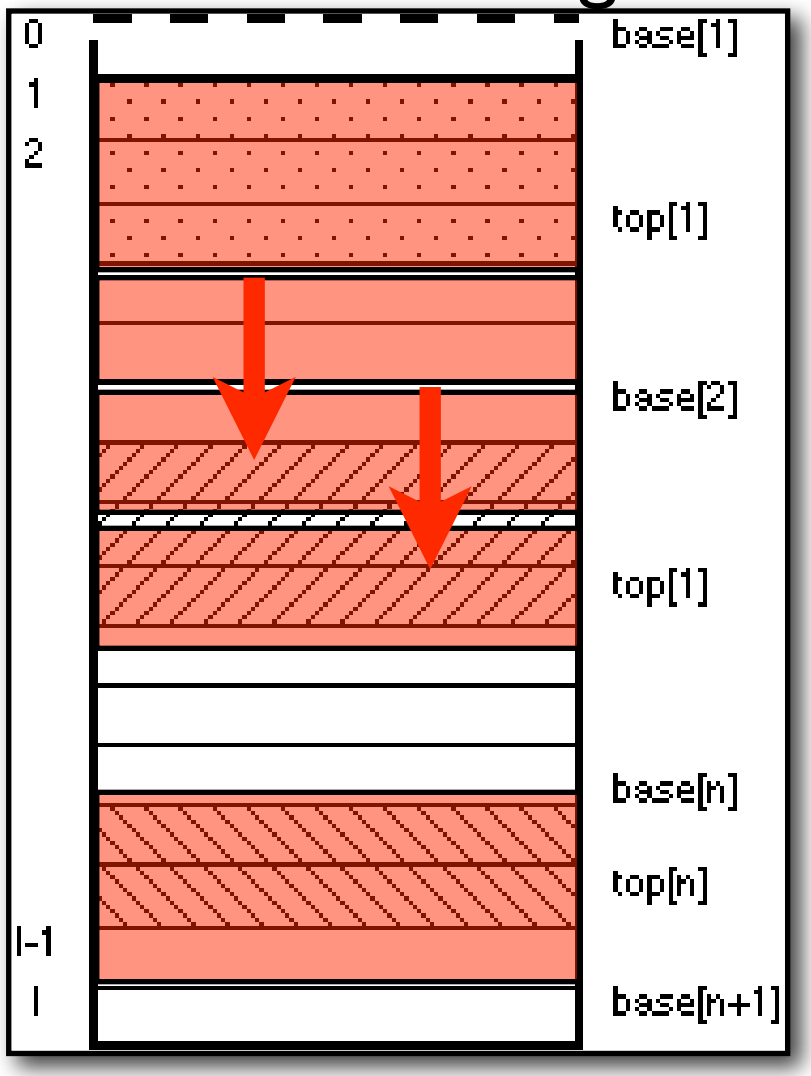
nach Umordnung



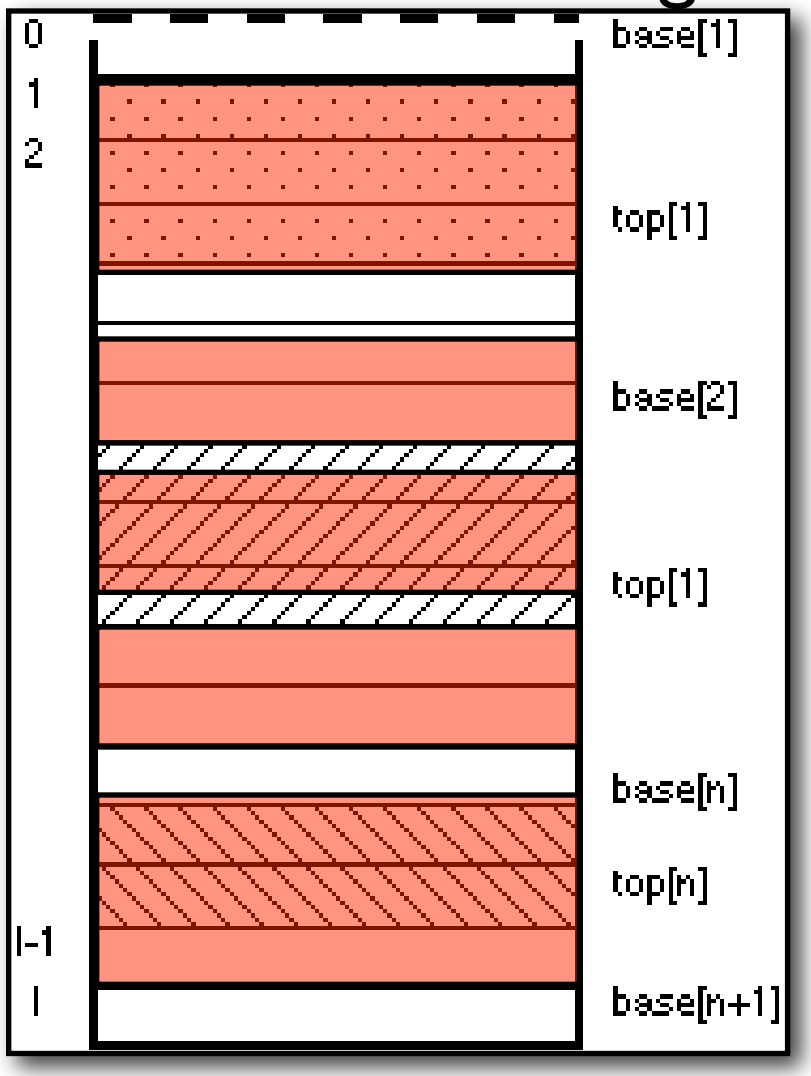
Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung



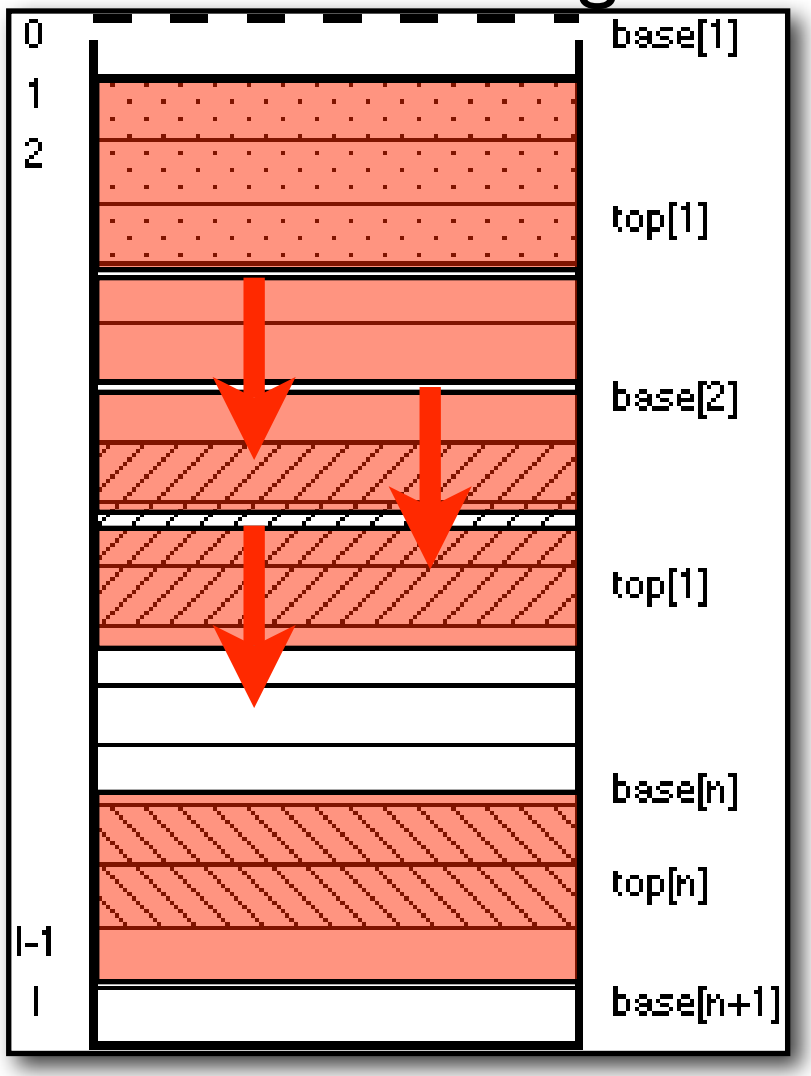
nach Umordnung



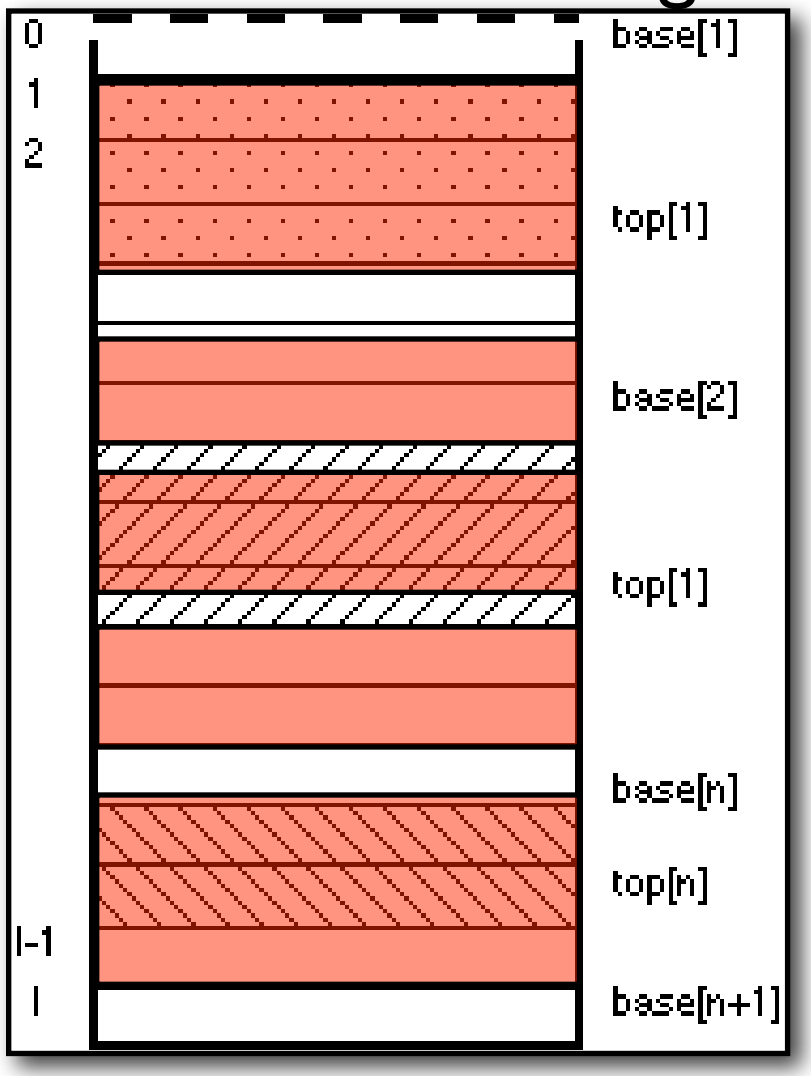
Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung



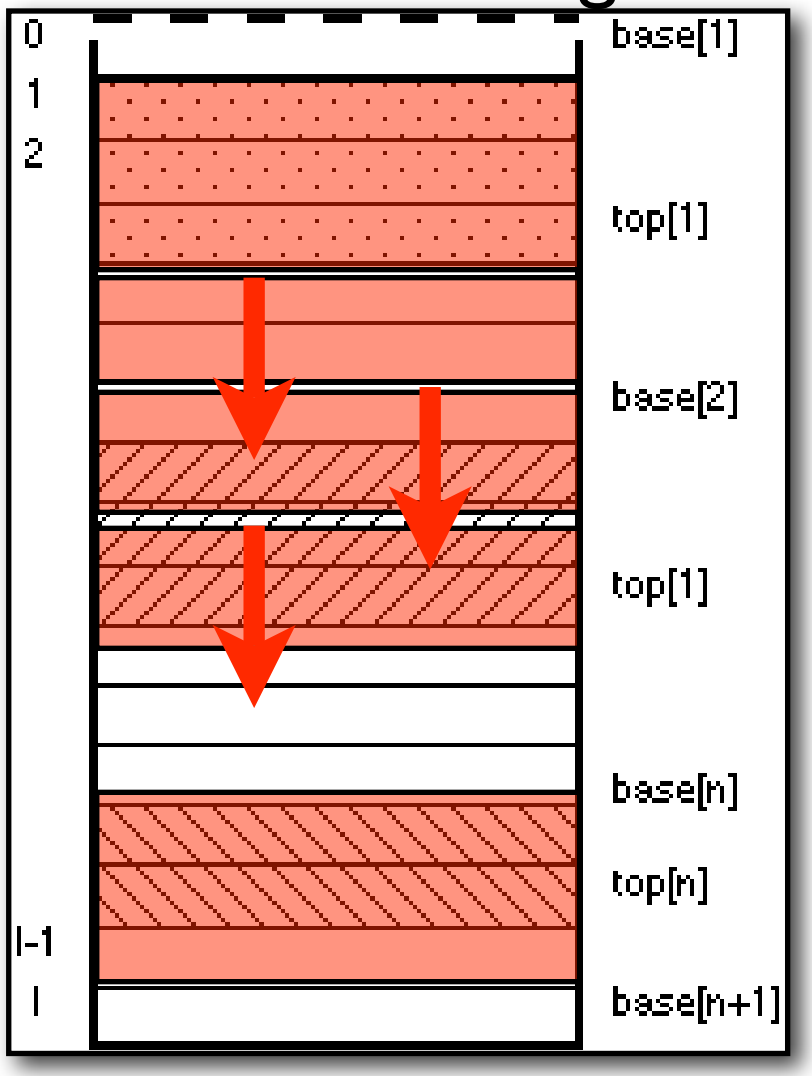
nach Umordnung



Implementierung von Datent.

Multi-Stacks - Umordnung

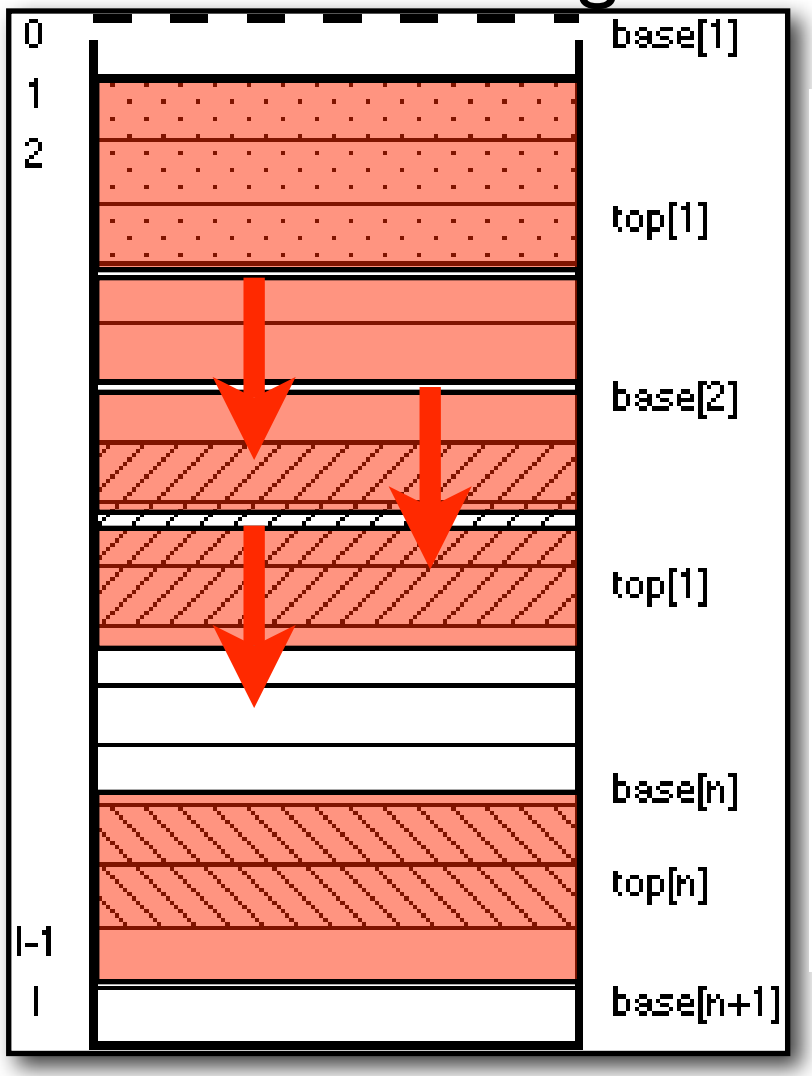
vor Umordnung



Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung

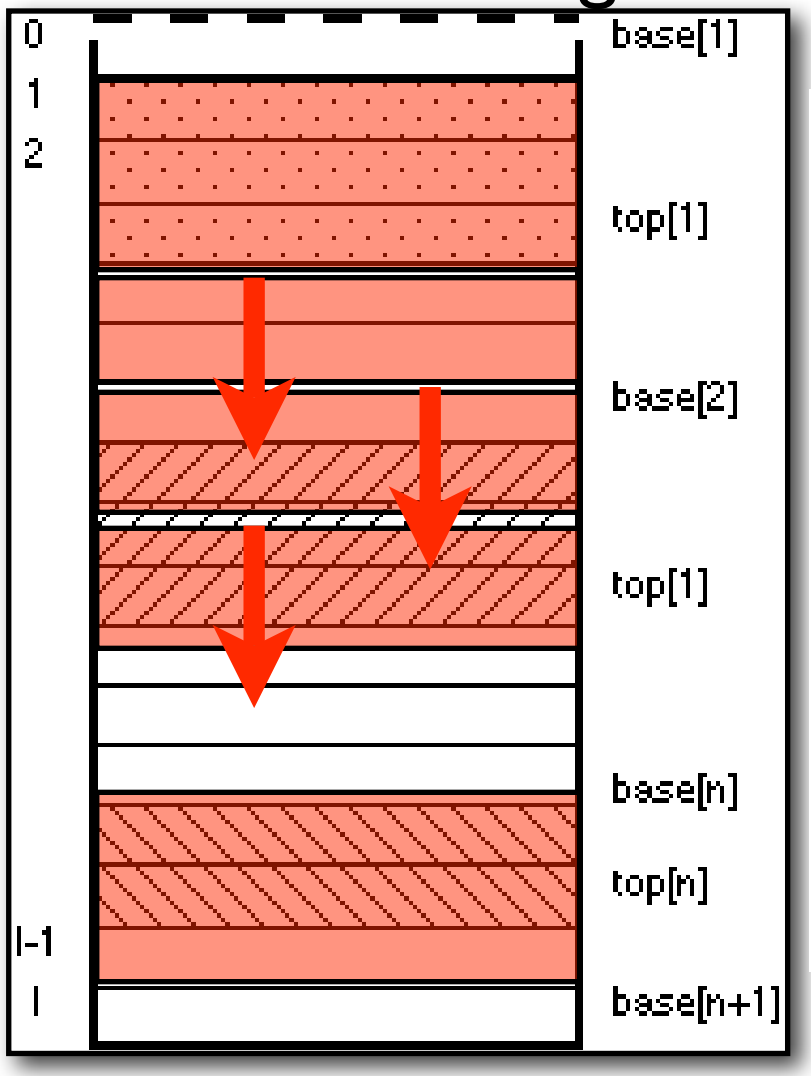



```
while j<=n do
begin
      k:=j;
      if newbase[k]<=base[k] then
            verschieben(s,k) else
      begin
            while newbase[k+1]>base[k+1]
                  do k:=k+1;
            for j1:=k downto j do
                  verschieben(s,j1)
      end;
      j:=k+1
end
```

Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung

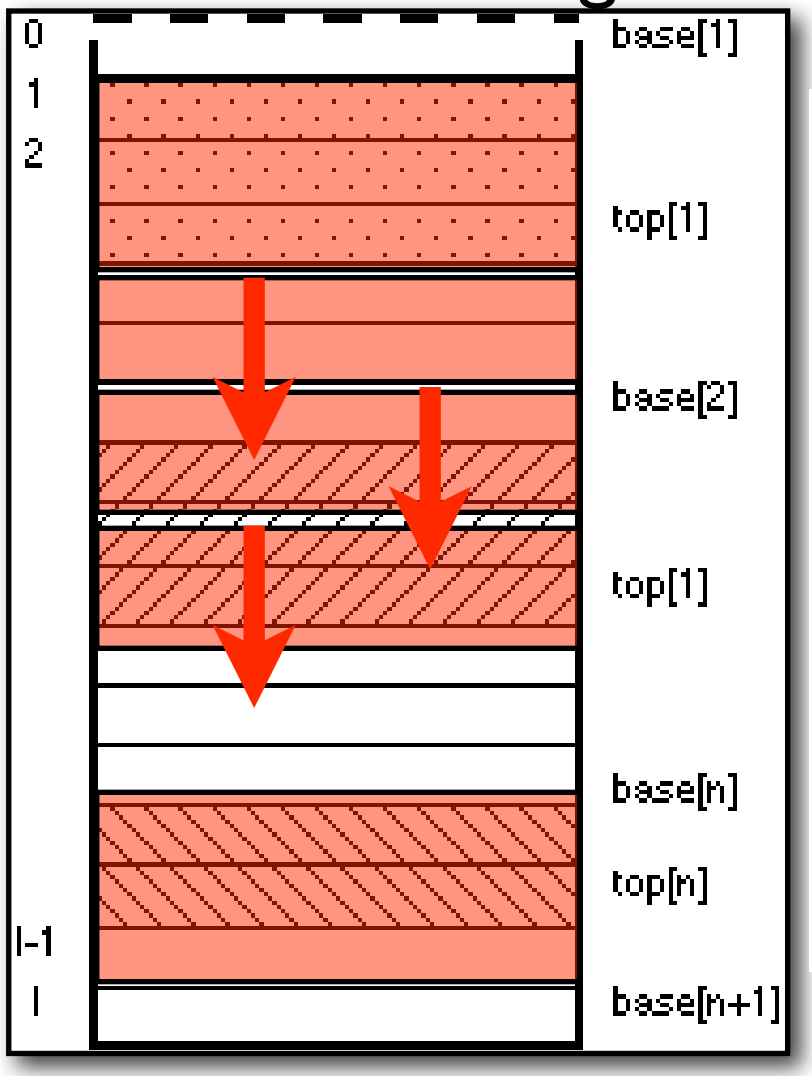



```
while j<=n do
begin
      k:=j;
      if newbase[k]<=base[k] then
         verschieben(s,k) else
      begin
            while newbase[k+1]>base[k+1]
                  do k:=k+1;
            for j1:=k downto j do
                  verschieben(s,j1)
      end;
      j:=k+1
end
```

Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung

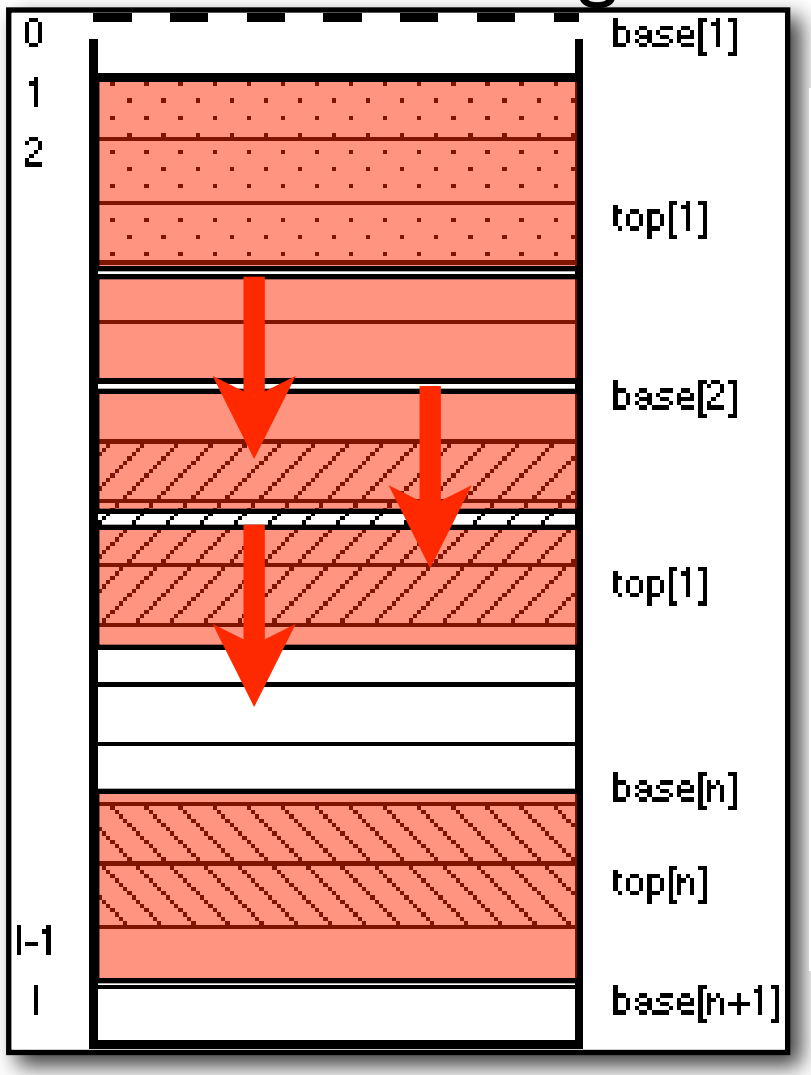



```
while j<=n do
begin
      k:=j;
      if newbase[k]<=base[k] then
            verschieben(s,k) else
      begin
            while newbase[k+1]>base[k+1]
                  do k:=k+1;
            for j1:=k downto j do
                  verschieben(s,j1)
      end;
      j:=k+1
end
```

Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung

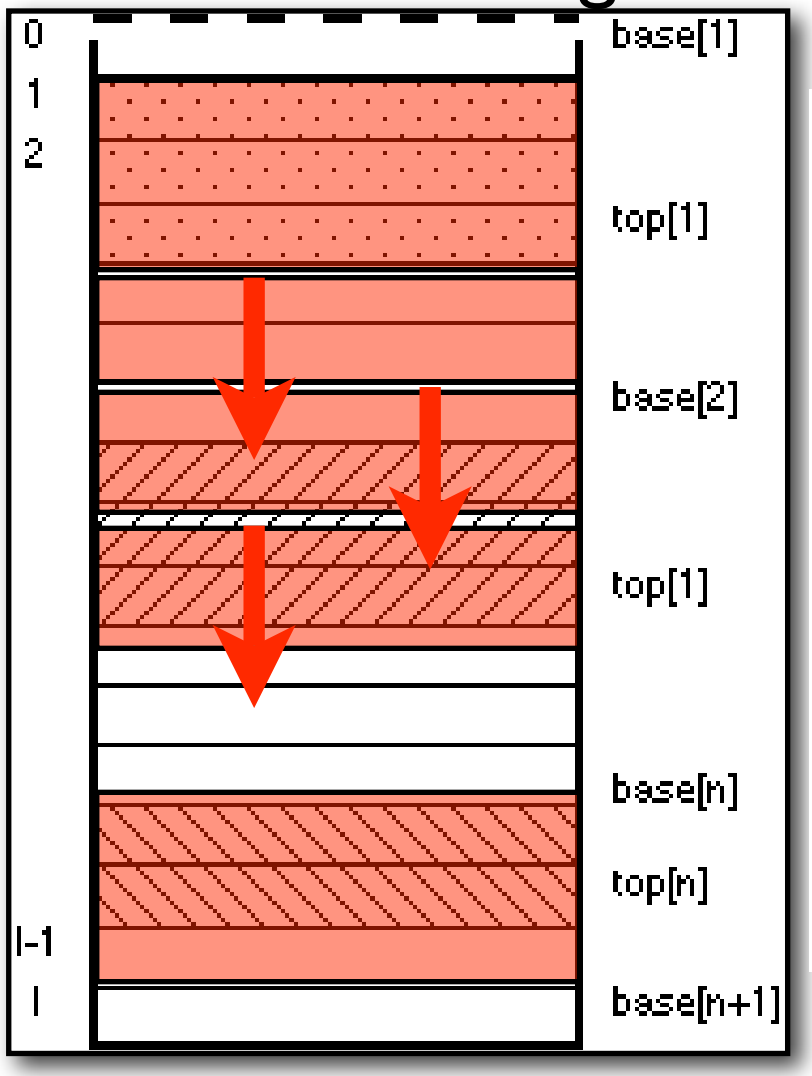



```
while j<=n do
begin
      k:=j;
      if newbase[k]<=base[k] then
            verschieben(s,k) else
      begin
            while newbase[k+1]>base[k+1]
                  do k:=k+1;
            for j1:=k downto j do
                  verschieben(s,j1)
      end;
      j:=k+1
end
```

Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung

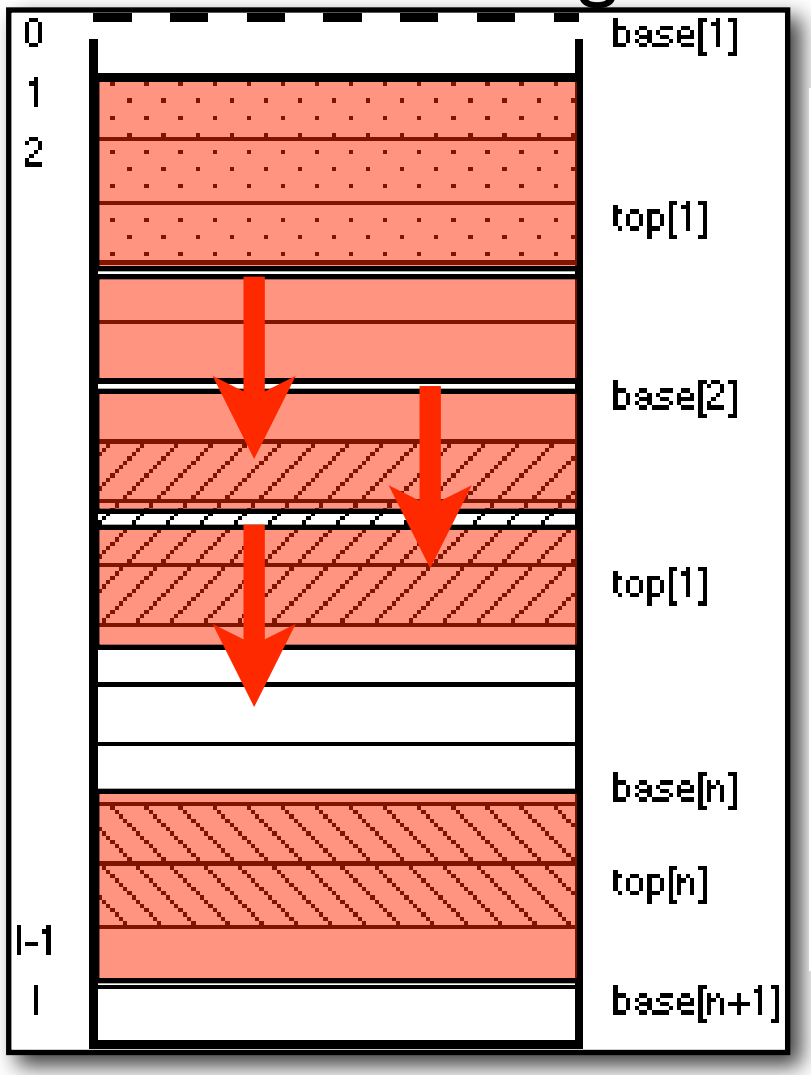



```
while j<=n do
begin
      k:=j;
      if newbase[k]<=base[k] then
            verschieben(s,k) else
      begin
            while newbase[k+1]>base[k+1]
                  do k:=k+1;
            for j1:=k downto j do
                  verschieben(s,j1)
      end;
      j:=k+1
end
```

Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung

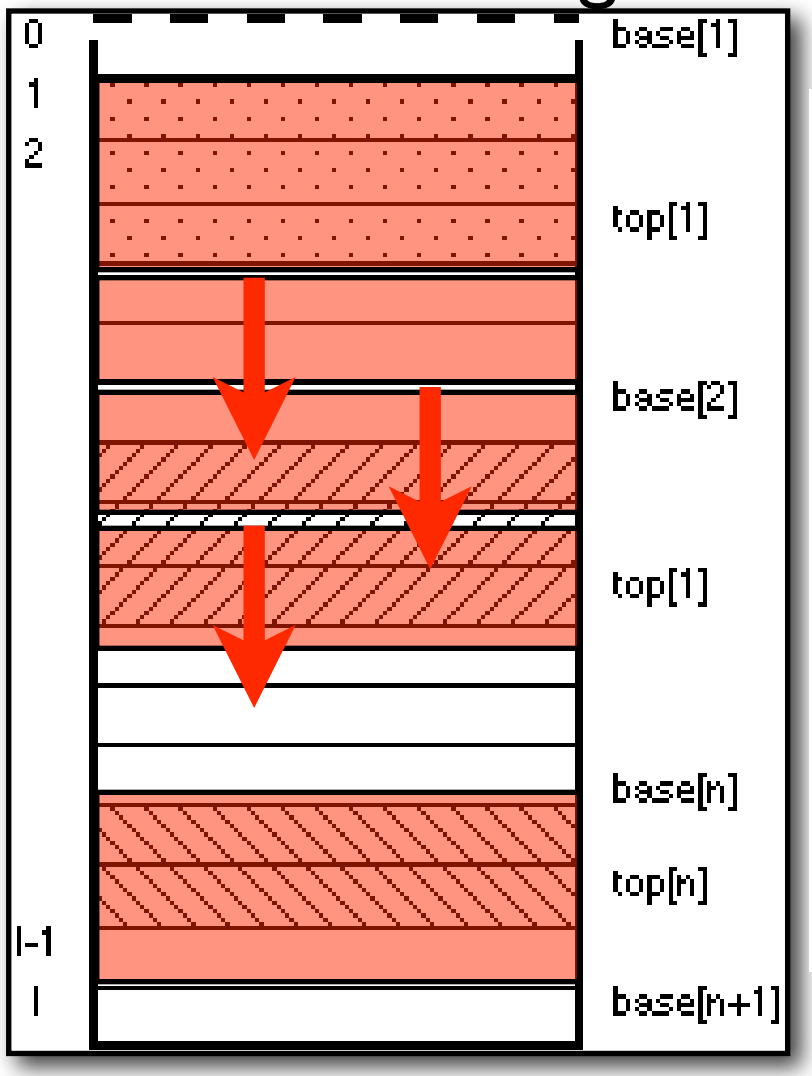



```
while j<=n do
begin
      k:=j;
      if newbase[k]<=base[k] then
            verschieben(s,k) else
      begin
            while newbase[k+1]>base[k+1]
                  do k:=k+1;
            for j1:=k downto j do
                  verschieben(s,j1)
      end;
      j:=k+1
end
```

Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung

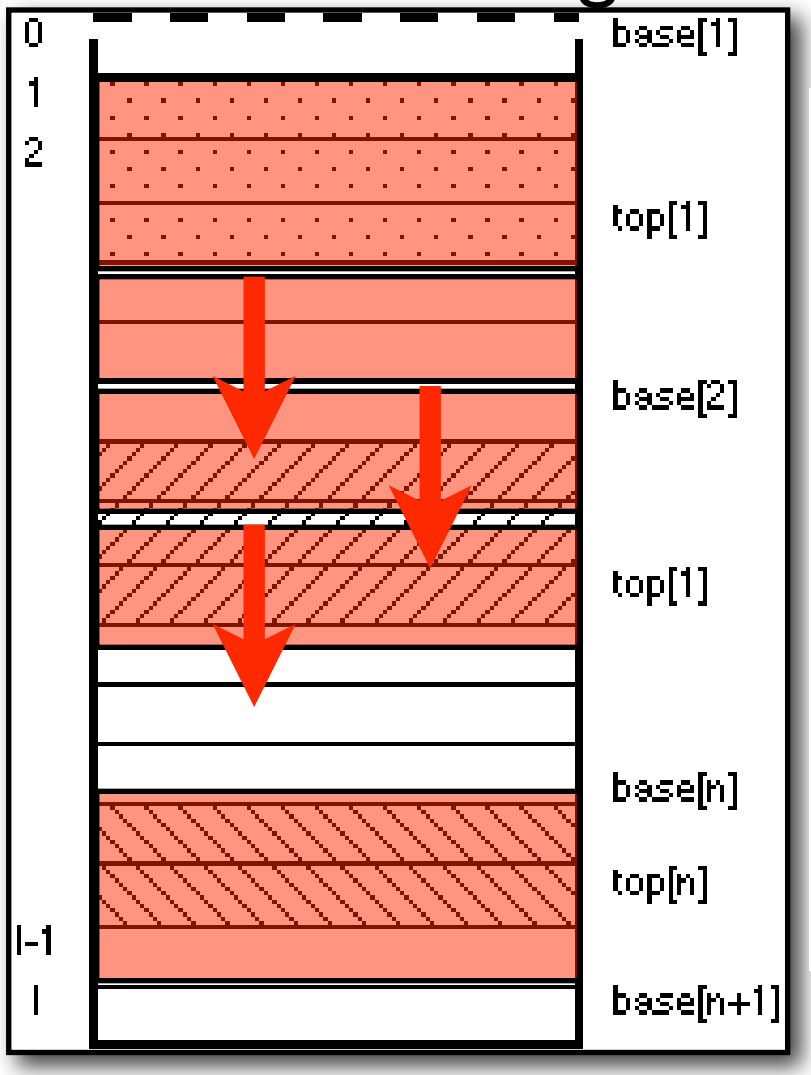



```
while j<=n do
begin
      k:=j;
      if newbase[k]<=base[k] then
            verschieben(s,k) else
      begin
            while newbase[k+1]>base[k+1]
                  do k:=k+1;
            for j1:=k downto j do
                  verschieben(s,j1)
      end;
      j:=k+1
end
```

Implementierung von Datent.

Multi-Stacks - Umordnung

vor Umordnung

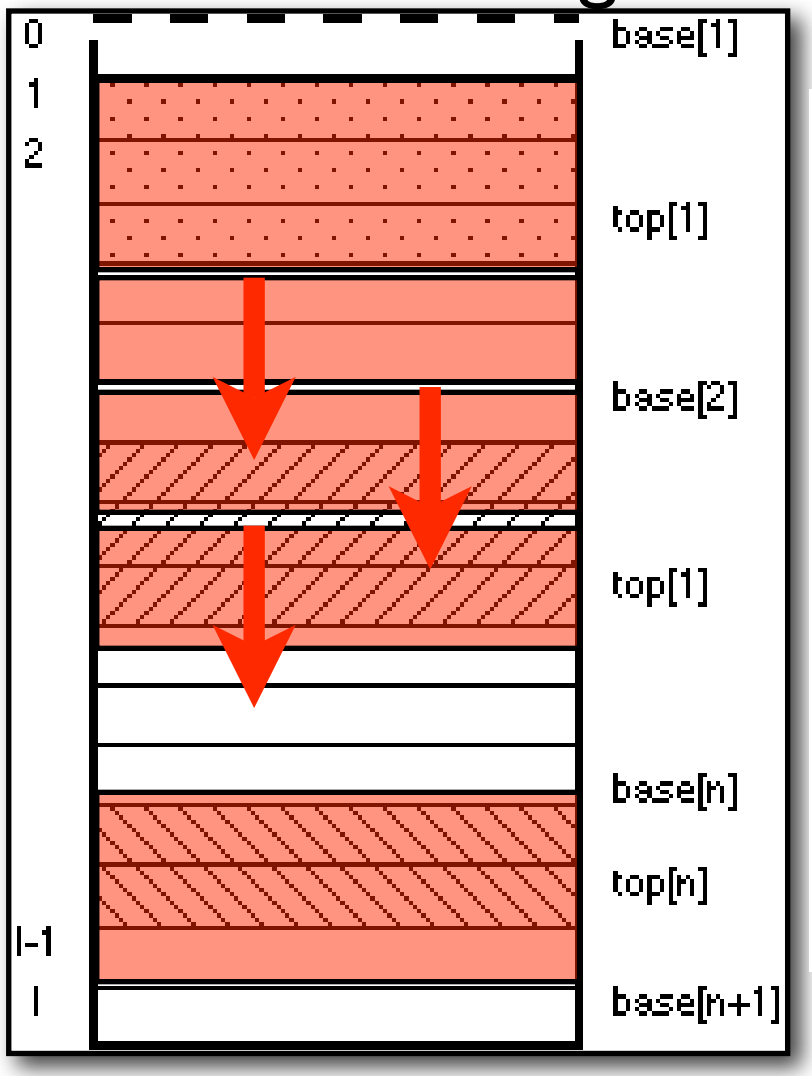



```
while j<=n do
begin
      k:=j;
      if newbase[k]<=base[k] then
            verschieben(s,k) else
      begin
            while newbase[k+1]>base[k+1]
                  do k:=k+1;
            for j1:=k downto j do
                  verschieben(s,j1)
      end;
      j:=k+1
end
```

Implementierung von Datent.

Multi-Stacks - Umordnung

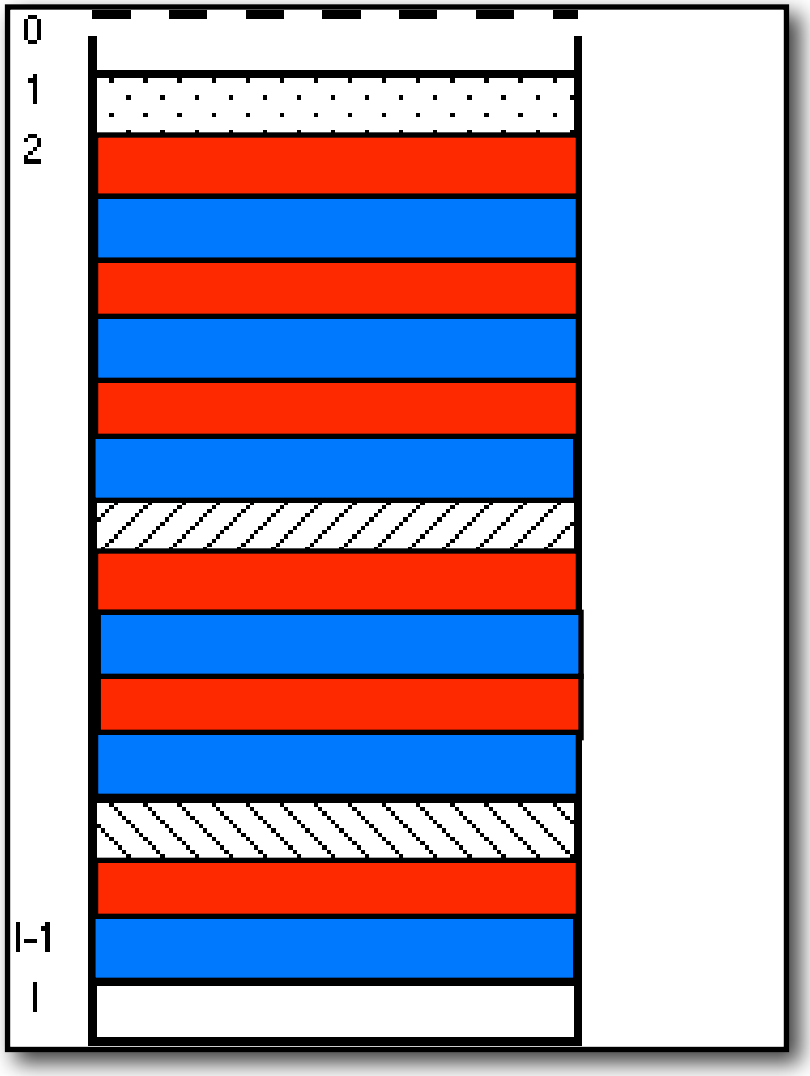
vor Umordnung




```
while j<=n do
begin
      k:=j;
      if newbase[k]<=base[k] then
            verschieben(s,k) else
      begin
            while newbase[k+1]>base[k+1]
                  do k:=k+1;
            for j1:=k downto j do
                  verschieben(s,j1)
      end;
      j:=k+1
end
```

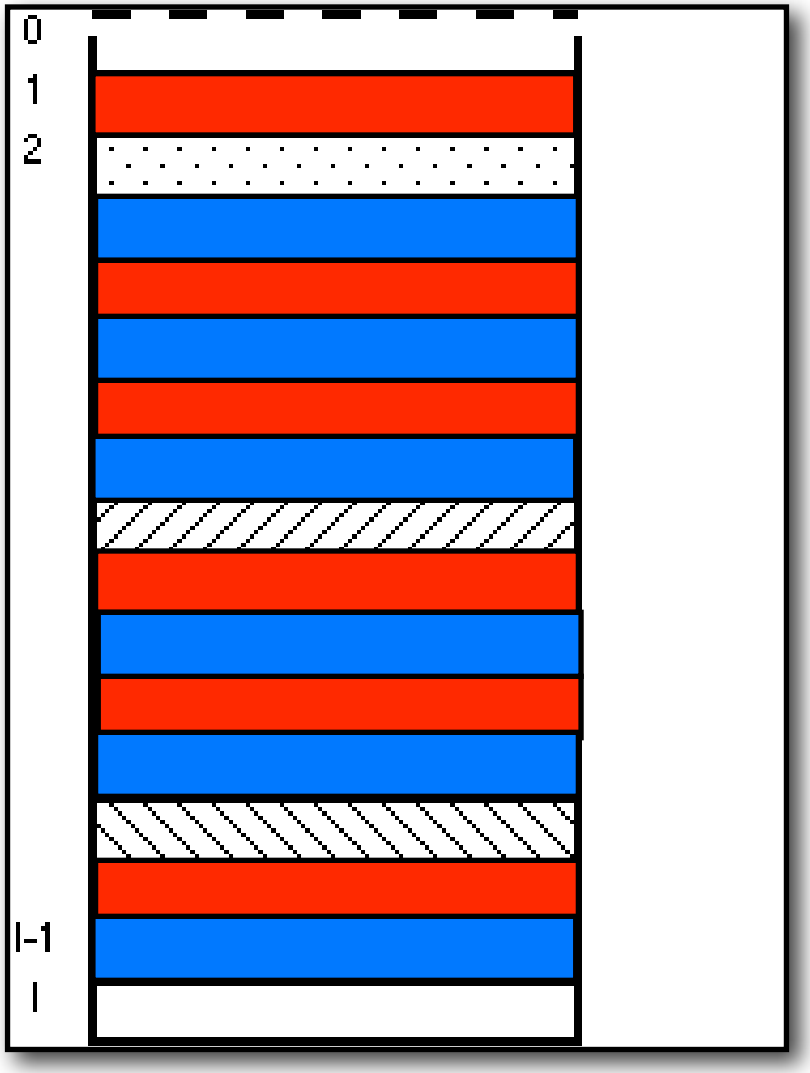
Implementierung von Datent.

Multi-Stacks - Umordnung



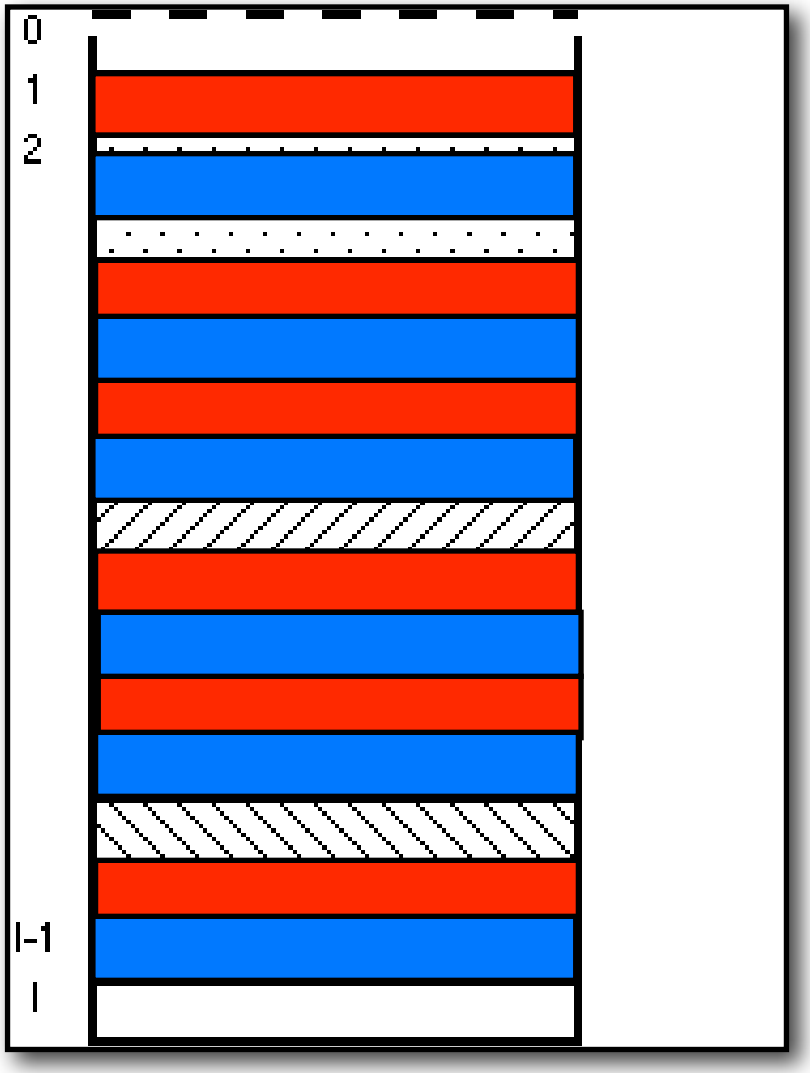
Implementierung von Datent.

Multi-Stacks - Umordnung



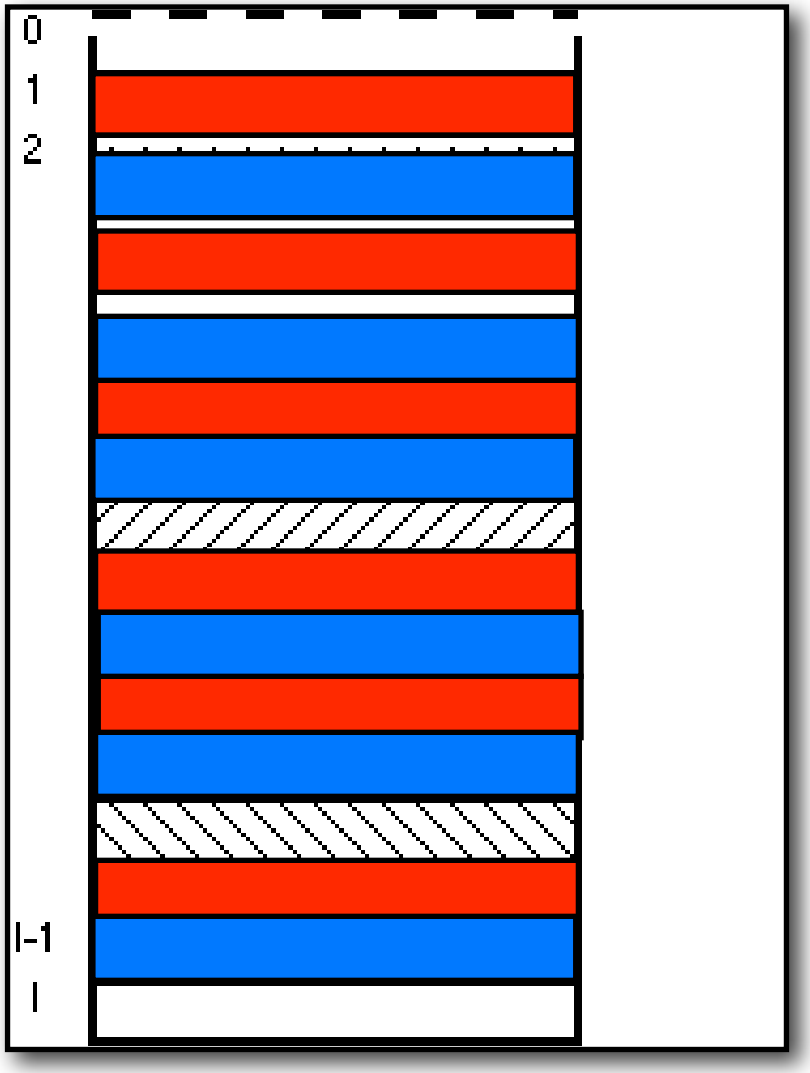
Implementierung von Datent.

Multi-Stacks - Umordnung



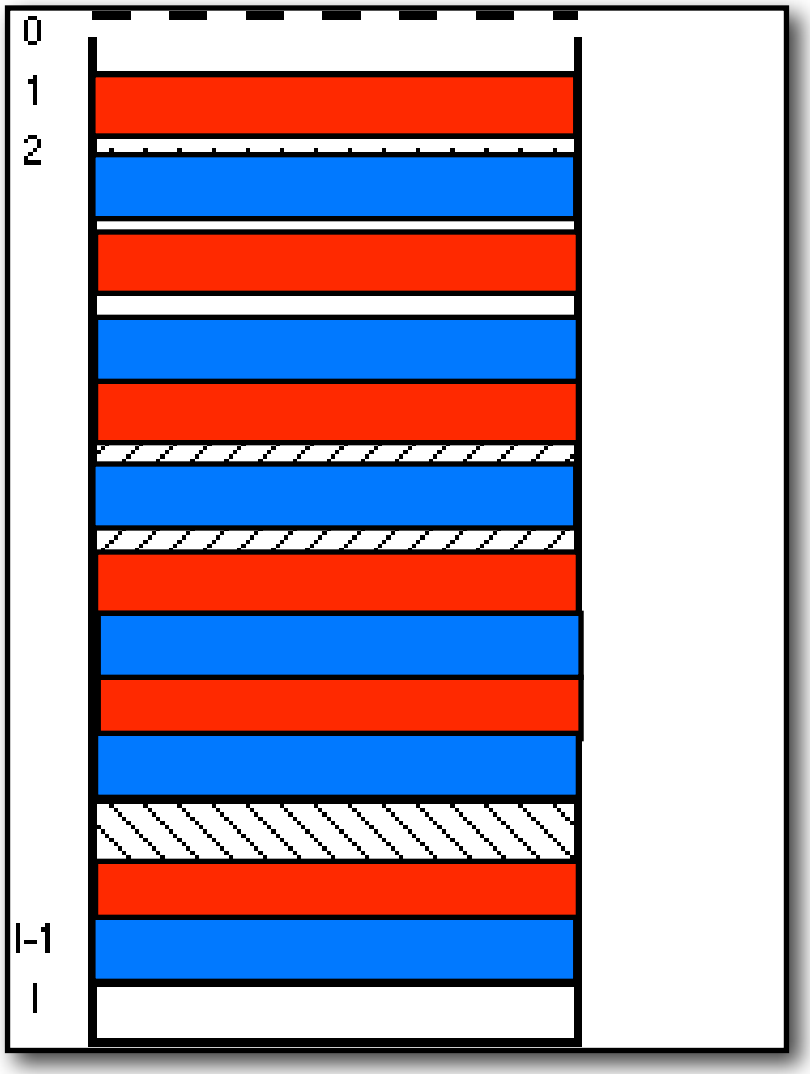
Implementierung von Datent.

Multi-Stacks - Umordnung



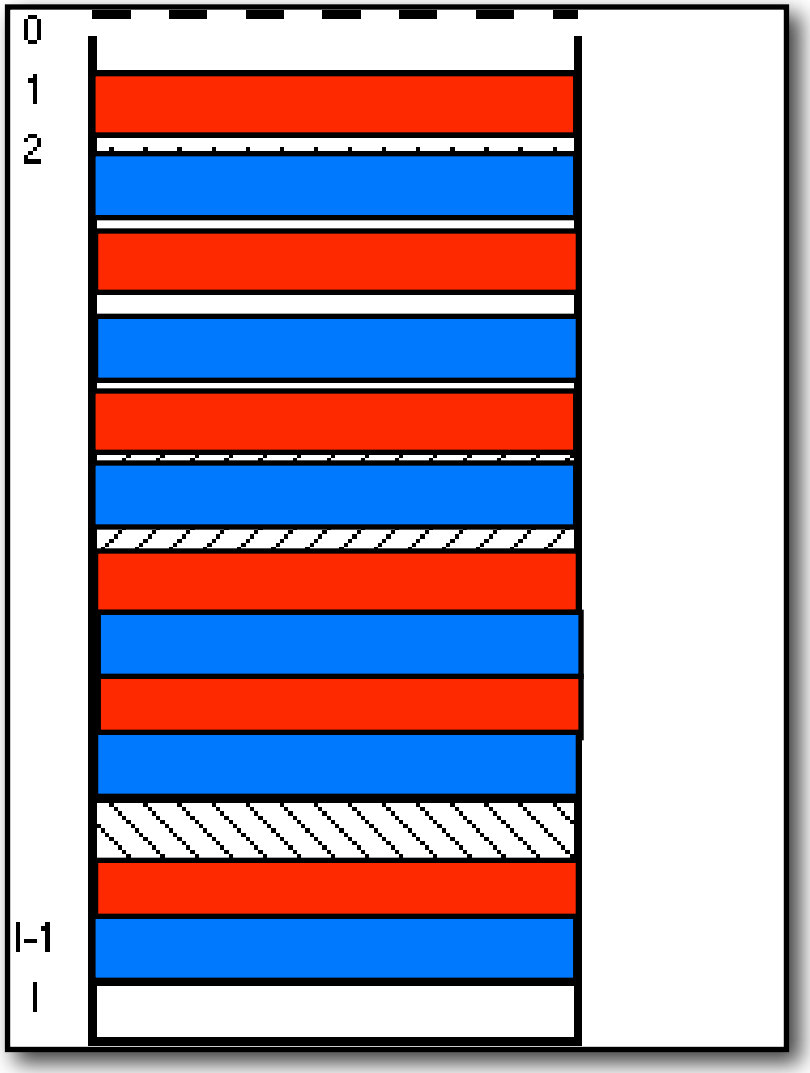
Implementierung von Datent.

Multi-Stacks - Umordnung



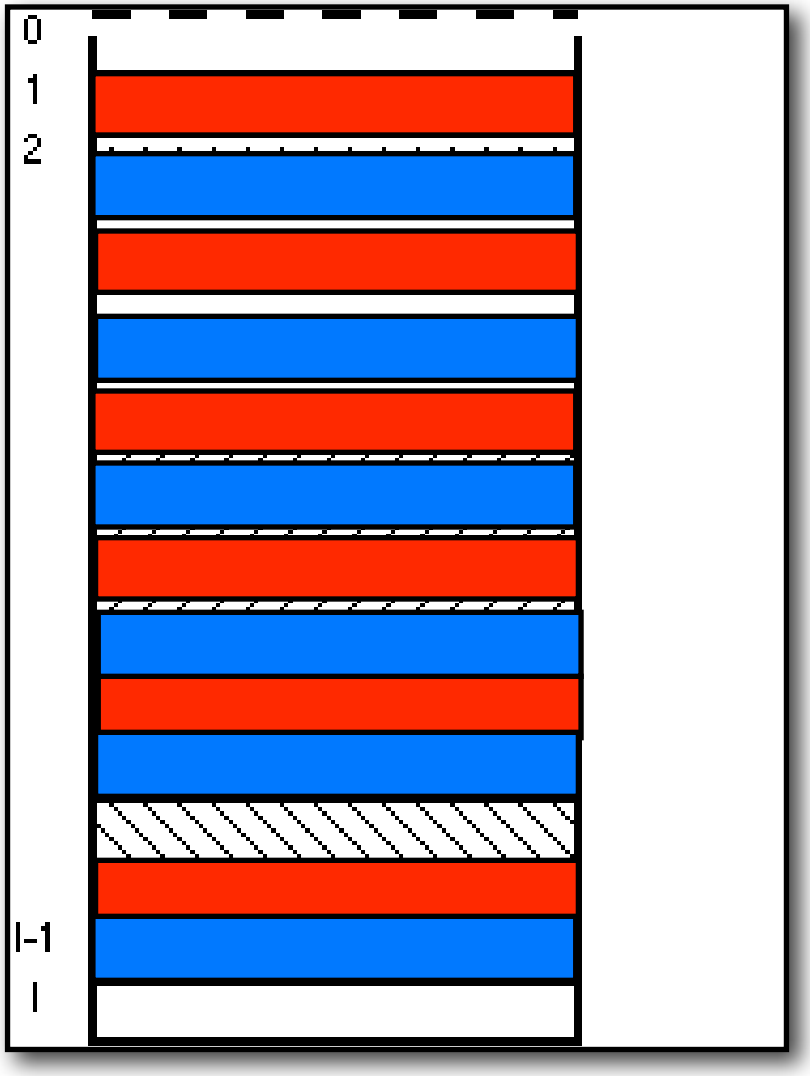
Implementierung von Datent.

Multi-Stacks - Umordnung



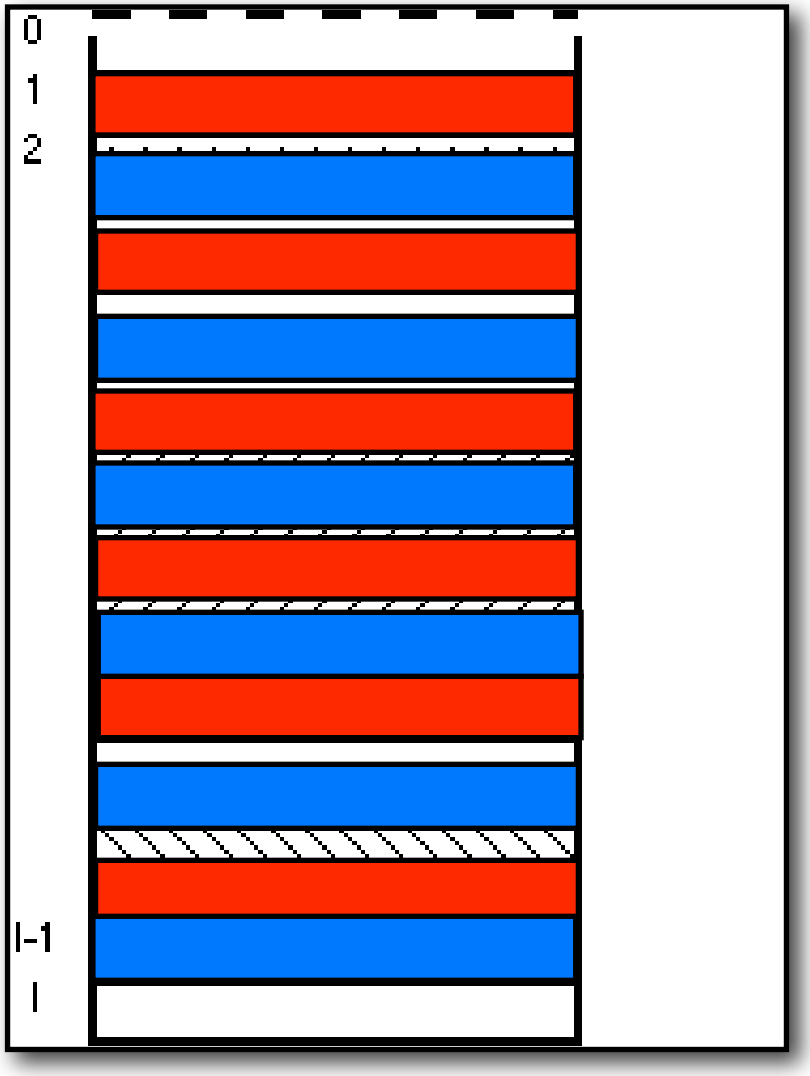
Implementierung von Datent.

Multi-Stacks - Umordnung



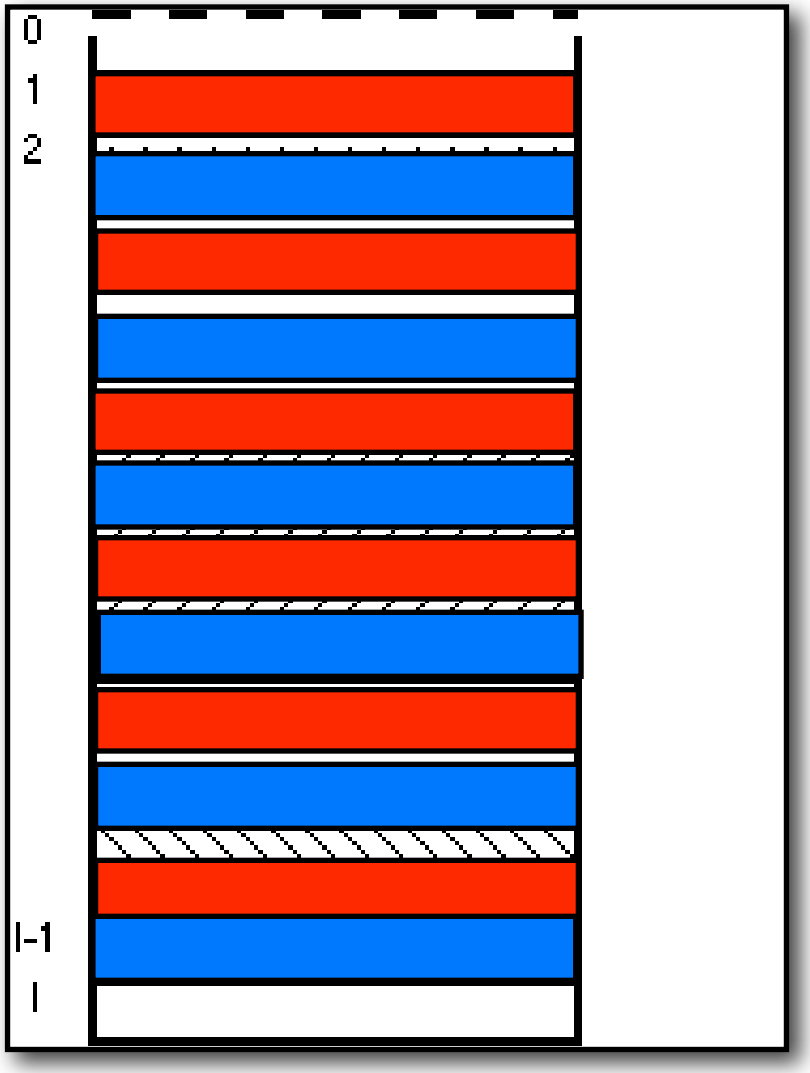
Implementierung von Datent.

Multi-Stacks - Umordnung



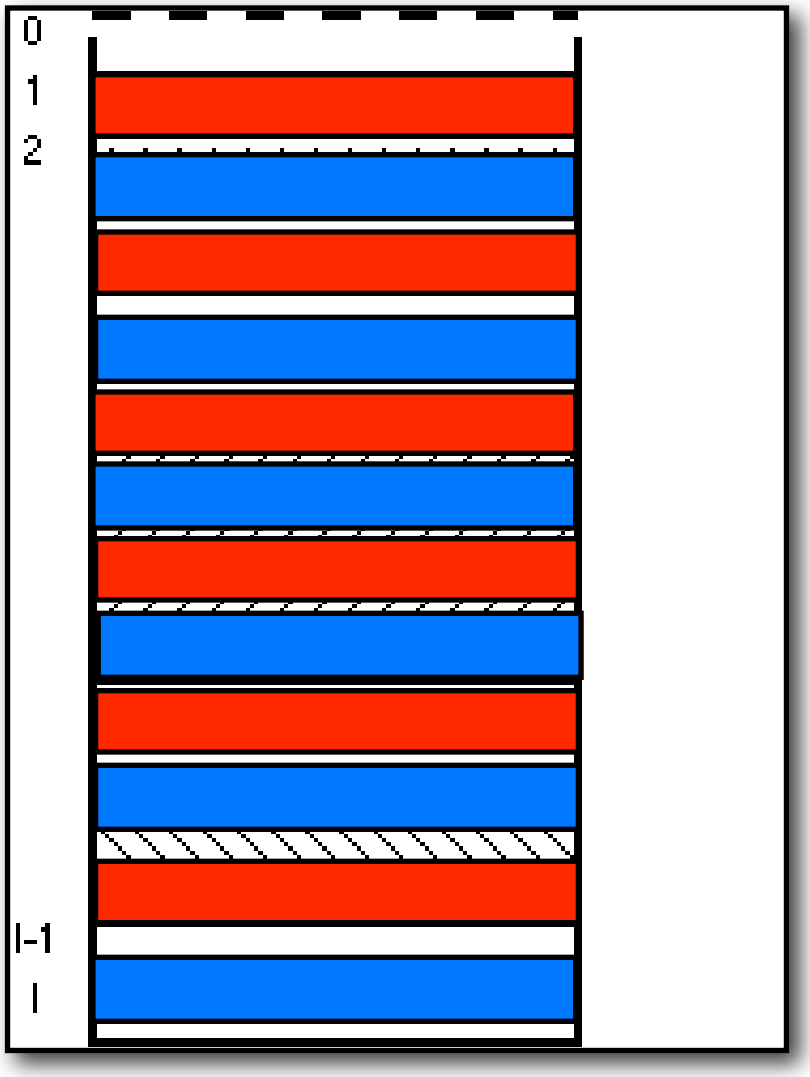
Implementierung von Datent.

Multi-Stacks - Umordnung



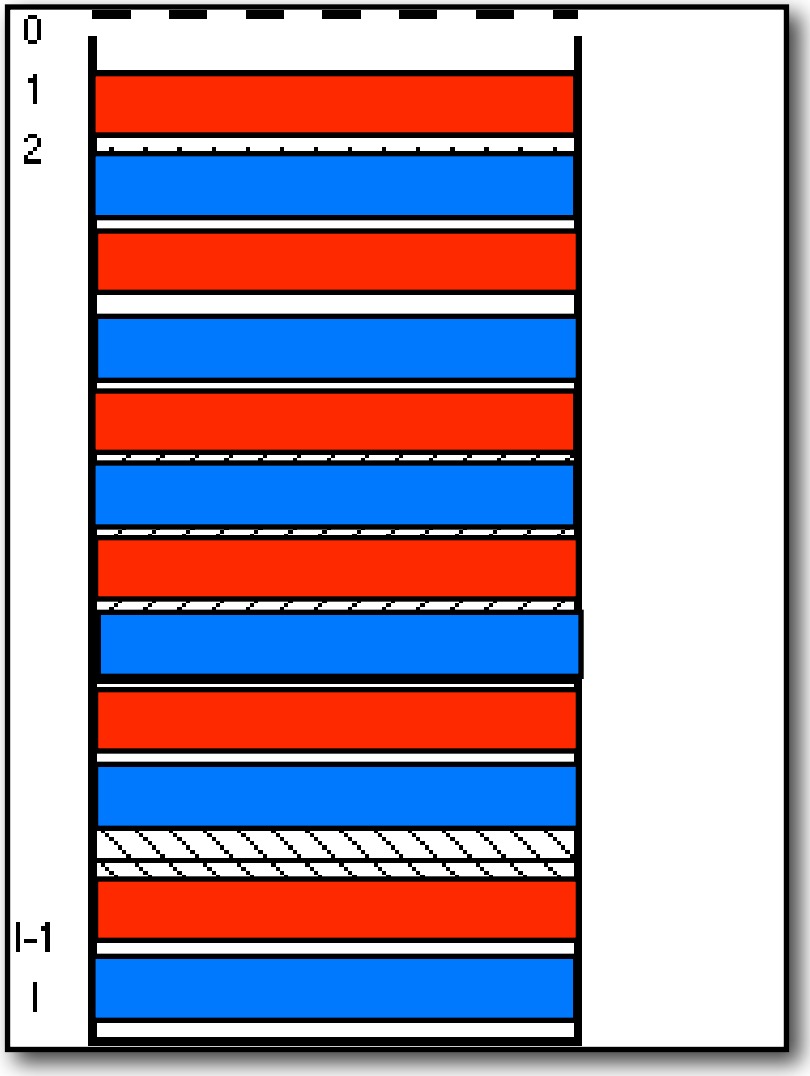
Implementierung von Datent.

Multi-Stacks - Umordnung



Implementierung von Datent.

Multi-Stacks - Umordnung



Implementierung von Datent.
Garwick-Algo. - Übertragung

- Weitere Anwendungen des Garwick-Algorithmus:
  - Queues: statt base und top dann anfang und ende
  - allgemein alle Tafeln/Tabellen, die relativ zu einer Basis adressiert werden

Implementierung von Datent.
Garwick-Algo. - Analyse

- bisher keine exakten theoretischen Effizienzanalysen
- Ergebnisse basieren auf experimentelle Untersuchungen
- Garwick-Algorithmus sehr effizient, wenn Speicher etwa zur Hälfte gefüllt
- Problematisch z.B. bei fehlerhaften Programmen: kurz vor Speicherüberlauf erheblicher Zeitverbrauch für fortlaufende Umordnungen
  - Mögliche Lösung: stoppe Prozedur overflow mit Fehler, wenn Zahl der freien Speicherplätze gewisses Minimum min>0 unterschreitet

Implementierung von Datent.
sequentielle Speicherverf. - Zusammenf.

- schneller direkter Zugriff auf Datenelemente
- kompliziertes Einfügen, meist Umordnen erforderlich, um Platz zu schaffen
- oft unzureichende Speicherausnutzung, weil für evtl. einzutragende Elemente Platz reserviert werden muß, oder Speicherbereinigungsalgorithmen, wie Garwick-Algorithmus, nur bei halbgefülltem Speicher effizient sind

Implementierung von Datent.
Rekursion bei Datentypen

- Motivation: markierter binärer Baum in FUN

```
typ Markierung ≡ {a,b,c,d,e};
typ Baum ≡ {leer} |(Baum,Markierung,Baum)
```

- Ein Objekt vom Typ Baum:

B=((leer,e,leer),a,((leer,b,leer),c,(leer,b,leer)))

Implementierung von Datent.
Rekursion bei Datentypen

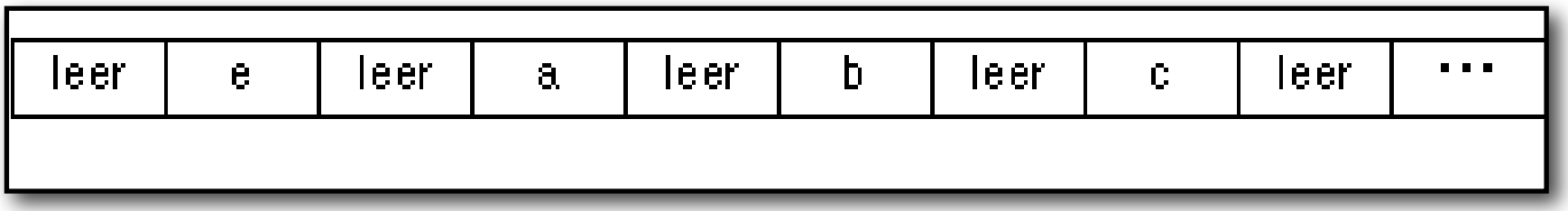
- Implementierungen:

  B=((leer,e,leer),a,((leer,b,leer),c,(leer,b,leer))).

Implementierung von Datent.
Rekursion bei Datentypen

- Implementierungen:

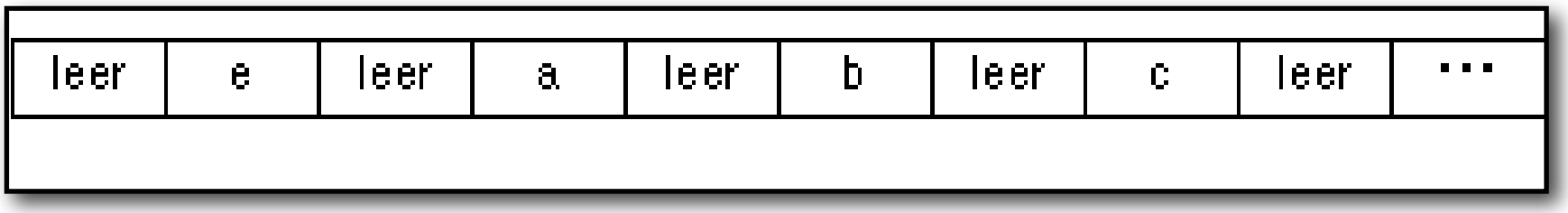
  B=((leer,e,leer),a,((leer,b,leer),c,(leer,b,leer))).



Implementierung von Datent.
Rekursion bei Datentypen

- Implementierungen:

  B=((leer,e,leer),a,((leer,b,leer),c,(leer,b,leer))).



Nachteile: Hierarchie fehlt, ständige Umordnung nötig, ...

Implementierung von Datent.
Rekursion bei Datentypen

- Implementierungen:

  B=((leer,e,leer),a,((leer,b,leer),c,(leer,b,leer))).

Nachteile: Hierarchie fehlt, ständige Umordnung nötig, ...

Implementierung von Datent.
Rekursion bei Datentypen

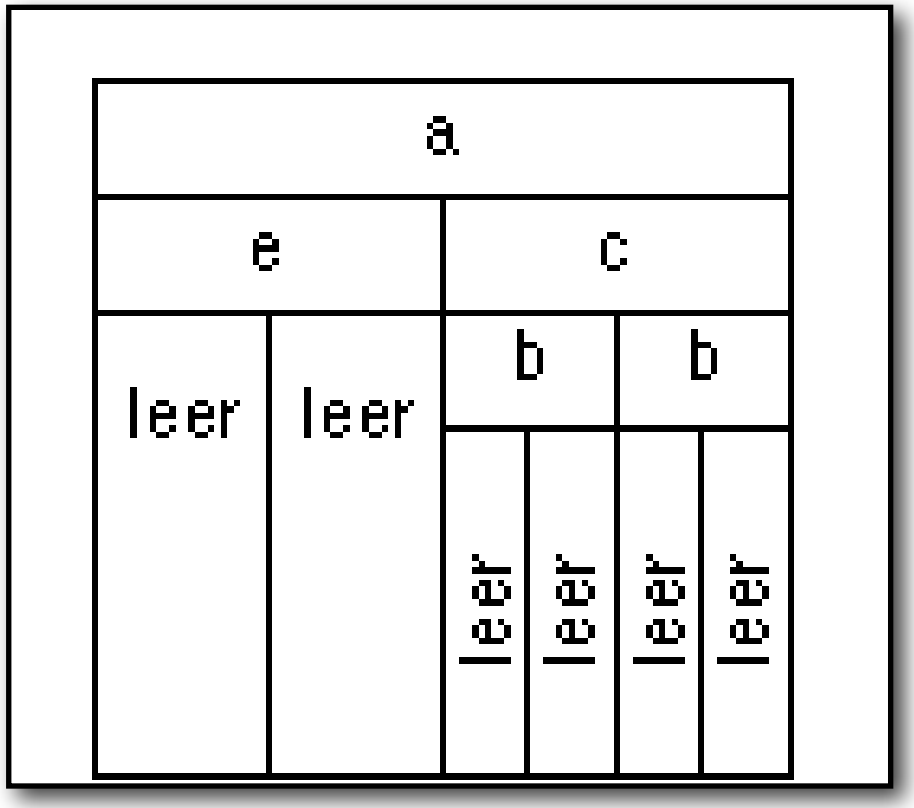
- Implementierungen:

  B=((leer,e,leer),a,((leer,b,leer),c,(leer,b,leer))).

Implementierung von Datent.
Rekursion bei Datentypen

- Implementierungen:

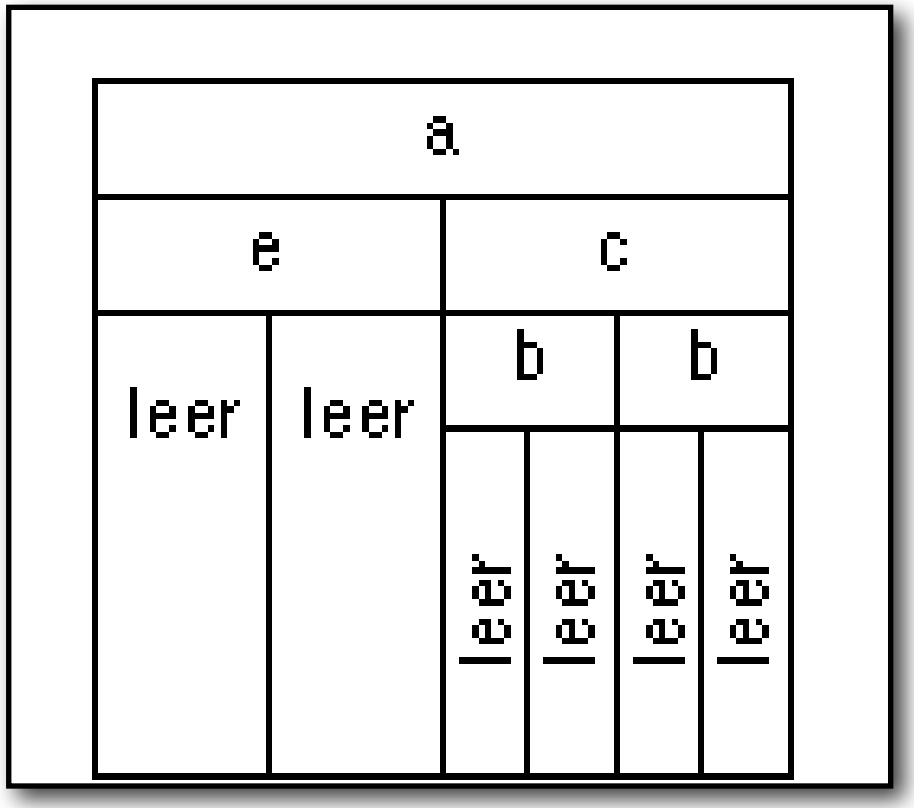
B=((leer,e,leer),a,((leer,b,leer),c,(leer,b,leer))).



Implementierung von Datent.
Rekursion bei Datentypen

- Implementierungen:

B=((leer,e,leer),a,((leer,b,leer),c,(leer,b,leer))).



Nachteile: keine Schachtelung möglich

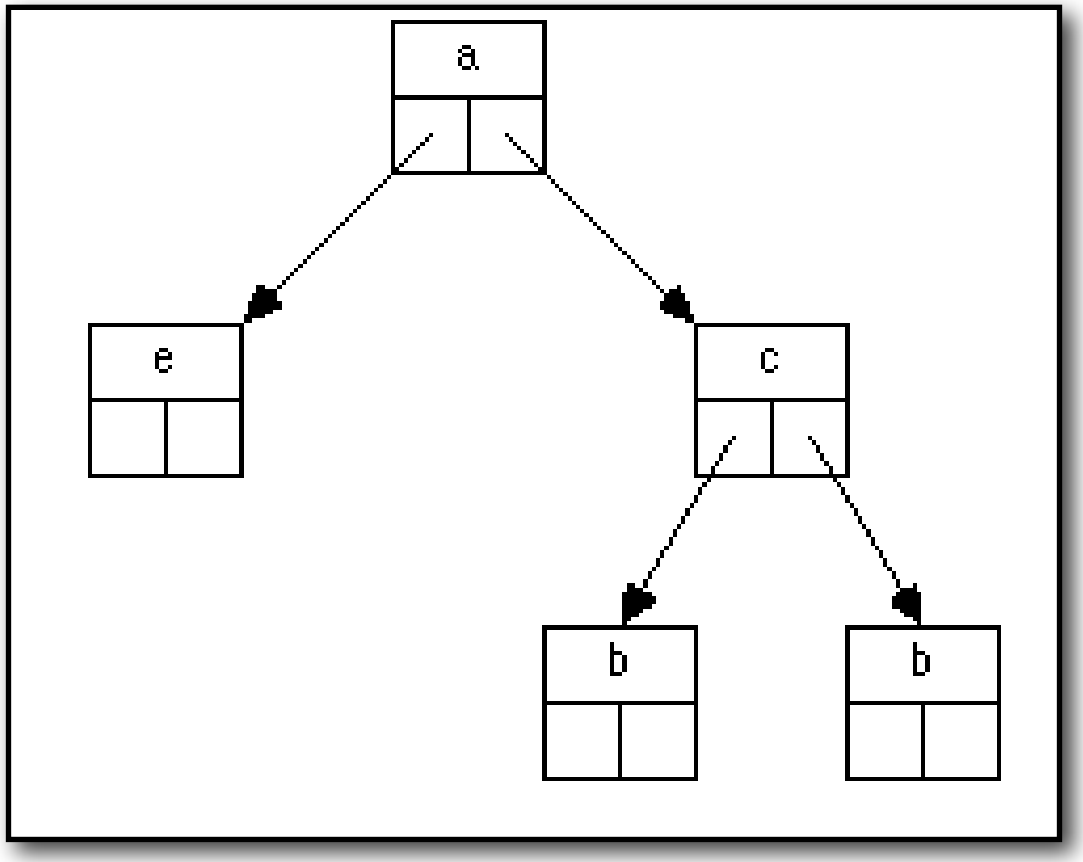
Implementierung von Datent.

Rekursion bei Datentypen - Referenzen

- Referenztechnik: Übergang zu verketteten Strukturen

Implementierung von Datent.
Rekursion bei Datentypen - Referenzen

- Referenztechnik: Übergang zu verketteten Strukturen



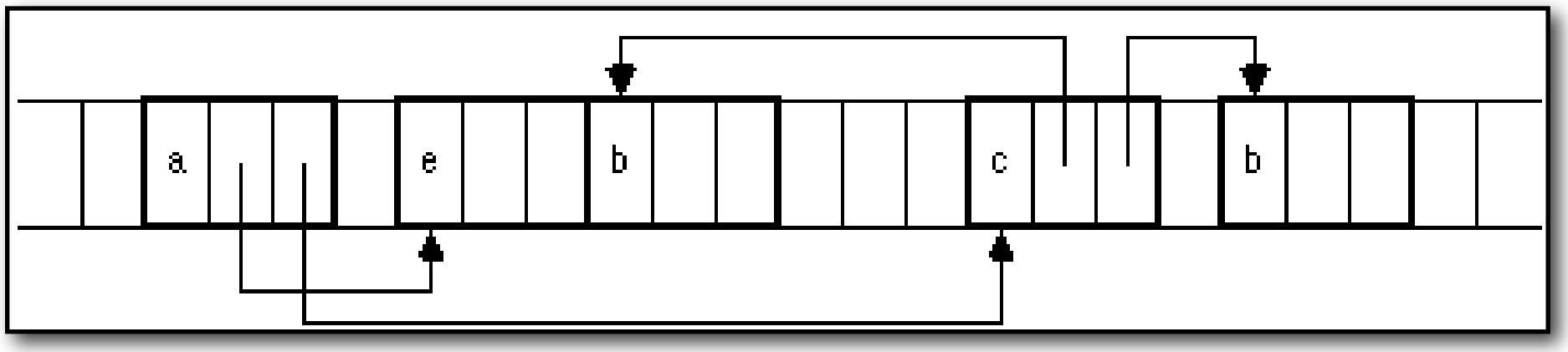
Implementierung von Datent.

Rekursion bei Datentypen - Referenzen

- Referenztechnik: Übergang zu verketteten Strukturen

Implementierung von Datent.
Rekursion bei Datentypen - Referenzen

- Referenztechnik: Übergang zu verketteten Strukturen



mögliche lineare Anordnung im Speicher

Implementierung von Datent.
Rekursion bei Datentypen - Referenzen

- Vorteile:
  - Bausteine aus identischen, einfach zu beschreibenden Objekten desselben Grundtyps
  - Bausteine beliebig im Speicher ablegbar, sofern Verkettung durch Zeiger sichergestellt
  - Ein- und Ausfügen nun vergleichsweise unkompliziert

Implementierung von Datent.
Referenzen in Pascal

- Pascal-Darstellung:

```
type Baum=record
                marke: markierung;
                lt, rt: ↑Baum
          end.
```

Implementierung von Datent.
Referenzen in Pascal

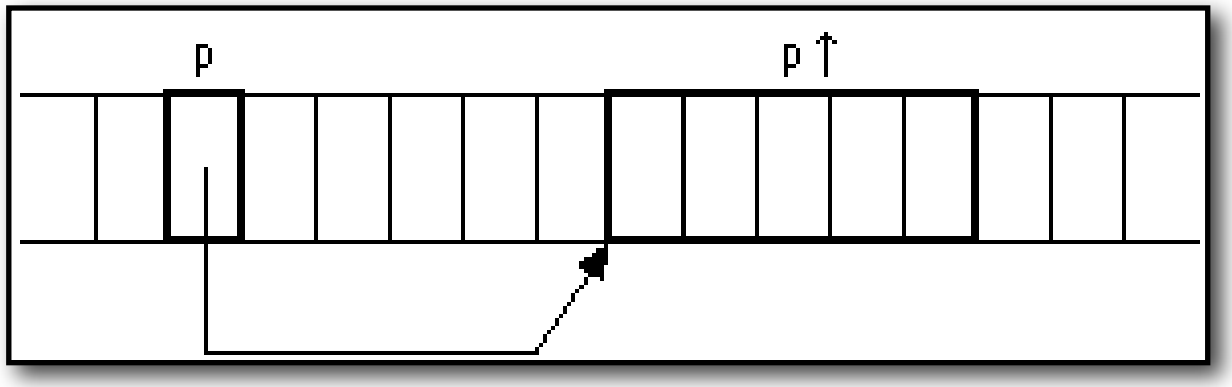
- Pascal-Darstellung:

```
type Baum=record
                marke: markierung;
                lt, rt: ↑Baum
          end.
```

Menge aller Verweise auf Werte vom Typ Baum

Implementierung von Datent.
Referenzen in Pascal - Verweise

- Verweis: Adresse einer Speicherzelle oder von Gruppen aufeinanderfolgender Speicherzellen, in denen Objekte vom Typ Baum abgelegt werden können



- Überprüfung, ob in Speicherzellen Objekte des deklarierten Typs abgelegt werden, erfolgt zur Übersetzungszeit des Programms

Implementierung von Datent.
Referenzen in Pascal - Zeiger

- Allgemeines Definitionsschema für Zeigertypen: T bel. Datentyp:

  type refT = ↑T

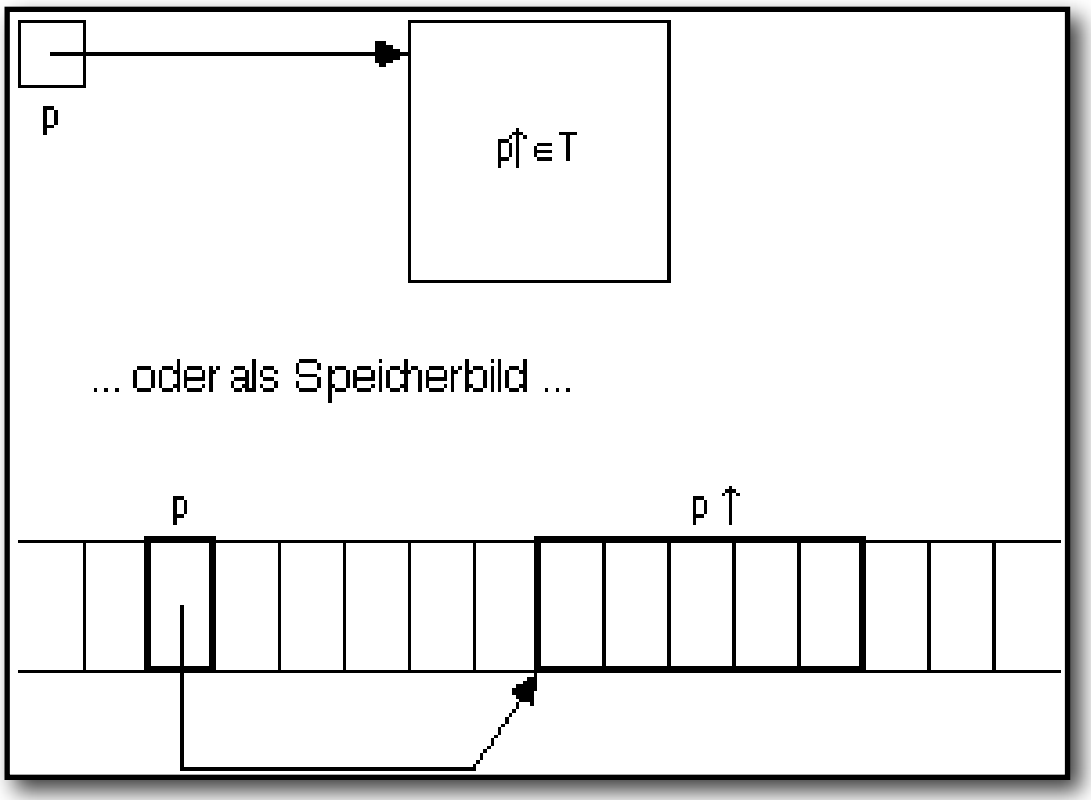
- ist zu T gehöriger Zeigertyp: Wertemenge W: Menge aller Zeiger auf Objekte vom Typ T
- polymorphe Konstante: nil∈W. nil zeigt auf kein Objekt. Beachte: nil≠undefiniert.

Implementierung von Datent.
Referenzen in Pascal - Standardoper.

- Standardoperationen für Zeiger:
  - new
  - dispose
  - ↑ (Dereferenzierung)

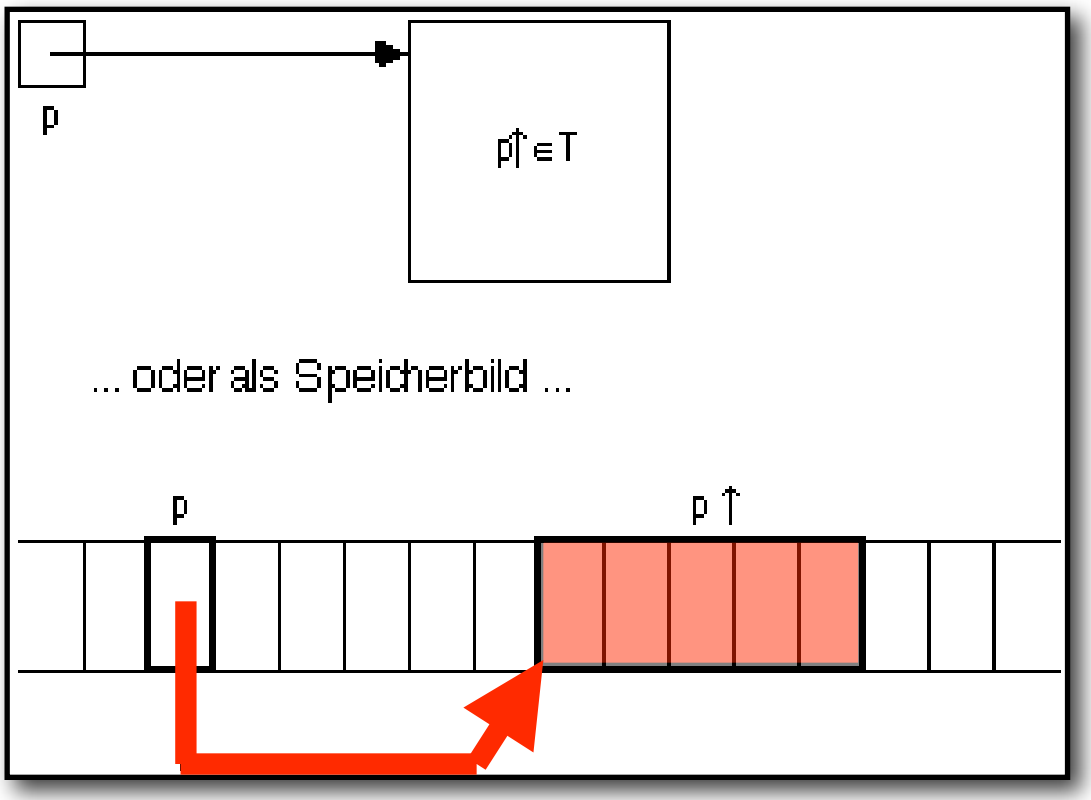
Implementierung von Datent.
Referenzen in Pascal - Standardoper.

- new(p): erzeugt (neues) Objekt vom Typ T und weist p Referenz auf Objekt zu



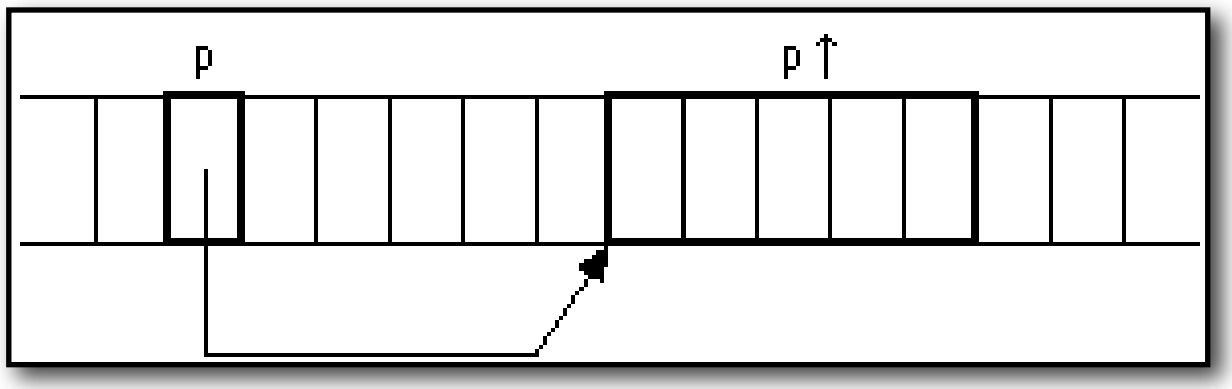
Implementierung von Datent.
Referenzen in Pascal - Standardoper.

- new(p): erzeugt (neues) Objekt vom Typ T und weist p Referenz auf Objekt zu



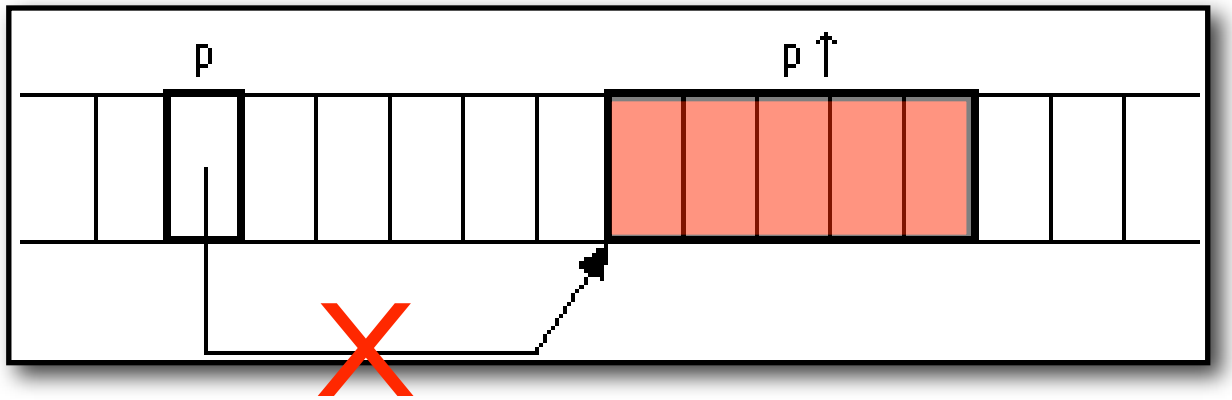
Implementierung von Datent.
Referenzen in Pascal - Standardoper.

- dispose(p): löscht Speicherbereich, den das Objekt belegt, auf das p weist



Implementierung von Datent.
Referenzen in Pascal - Standardoper.

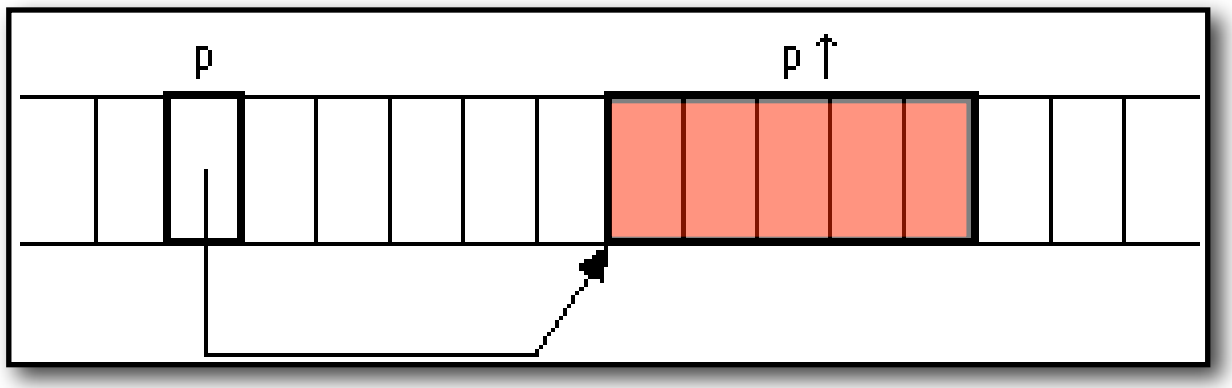
- dispose(p): löscht Speicherbereich, den das Objekt belegt, auf das p weist



keine direkte Löschung, nur als "gelöscht" markieren
später Speicherbereinigung

Implementierung von Datent.
Referenzen in Pascal - Standardoper.

- Dereferenzierung p↑: liefert p↑, also das Objekt, auf das p verweist; also gilt p↑∈T



Achtung: Unterscheide zwischen p und p↑

Implementierung von Datent.

Referenzen in Pascal - Standardoper.

Beispiel:

Implementierung von Datent.
Referenzen in Pascal - Standardoper.

Beispiel:

```
type T=record
        x: integer
end;
var p, q, r: ↑T
```

Implementierung von Datent.
Referenzen in Pascal - Standardoper.

Beispiel:

```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
```

Implementierung von Datent.

Referenzen in Pascal - Standardoper.

Beispiel:

```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
```

Implementierung von Datent.
Referenzen in Pascal - Standardoper.

Beispiel:

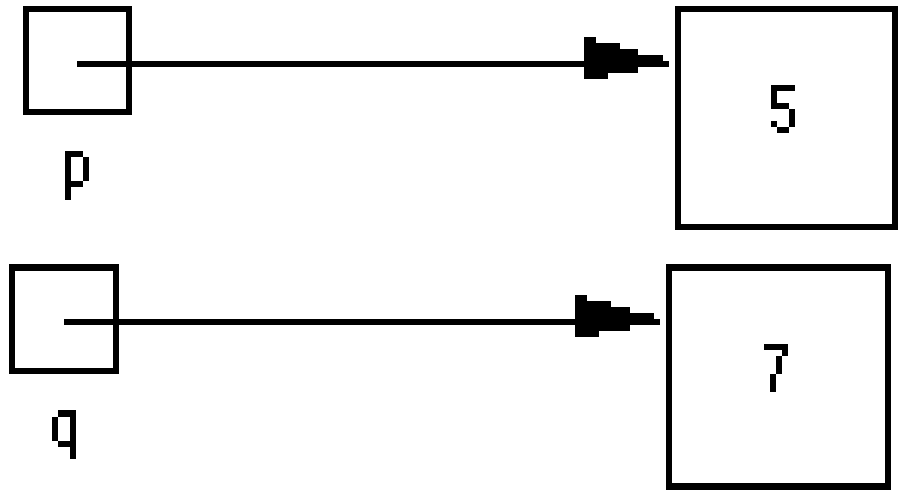
```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
new(q); q↑.x:=7;
```

Implementierung von Datent.

Referenzen in Pascal - Standardoper.

Beispiel:

```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
new(q); q↑.x:=7;
```

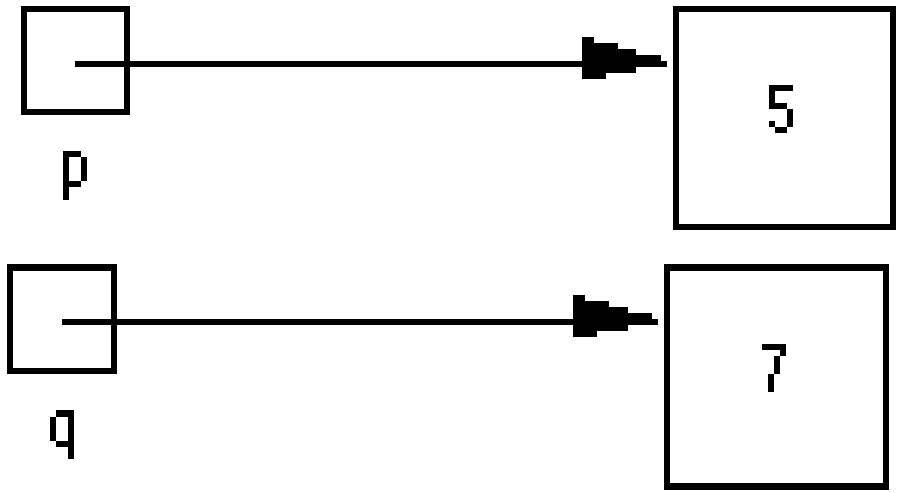


Implementierung von Datent.

Referenzen in Pascal - Standardoper.

Beispiel:

```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
new(q); q↑.x:=7;
new(r); r↑.x:=9;
```

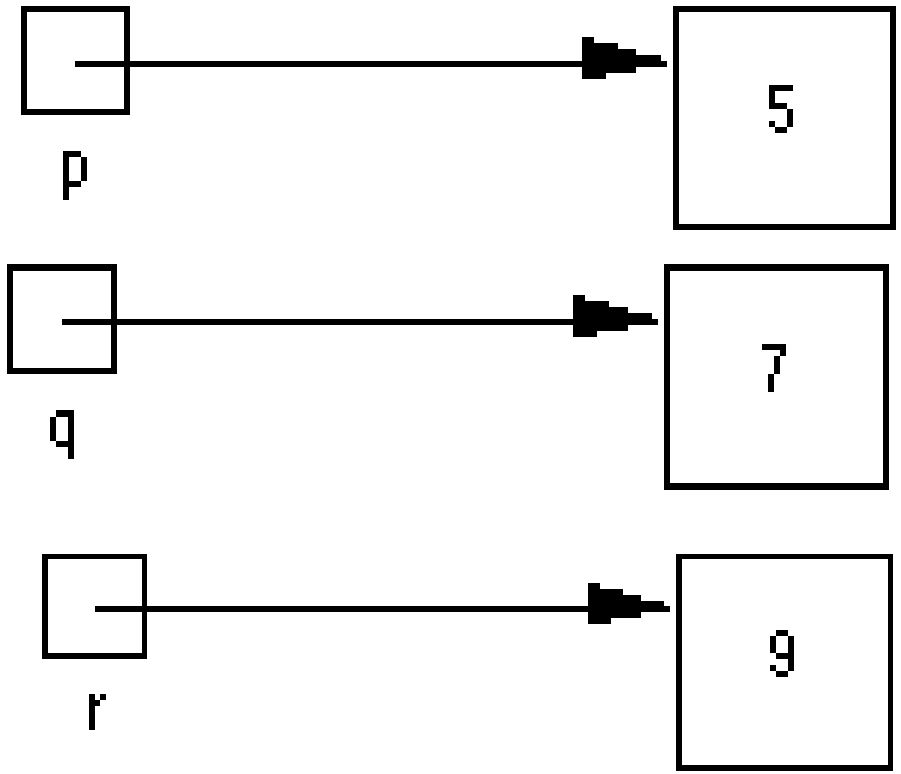


Implementierung von Datent.

Referenzen in Pascal - Standardoper.

Beispiel:

```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
new(q); q↑.x:=7;
new(r); r↑.x:=9;
```



Implementierung von Datent.
Referenzen in Pascal - Standardoper.

Beispiel:

```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
new(q); q↑.x:=7;
new(r); r↑.x:=9;
```

Implementierung von Datent.
Referenzen in Pascal - Standardoper.

Beispiel:

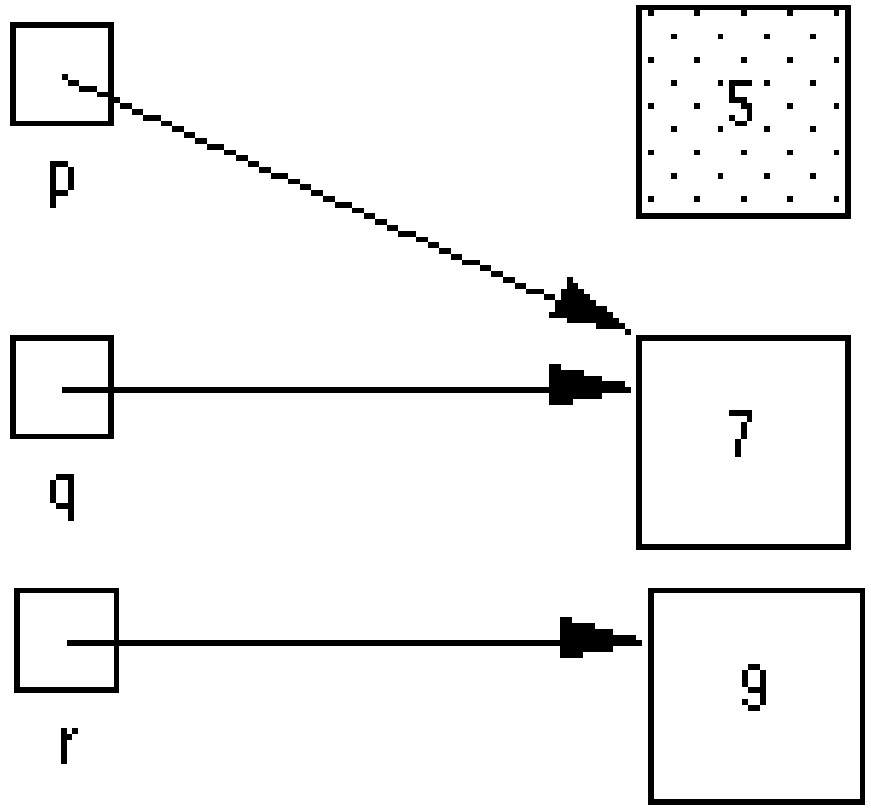
```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
new(q); q↑.x:=7;
new(r); r↑.x:=9;
p:=q;
```

Implementierung von Datent.

Referenzen in Pascal - Standardoper.

Beispiel:

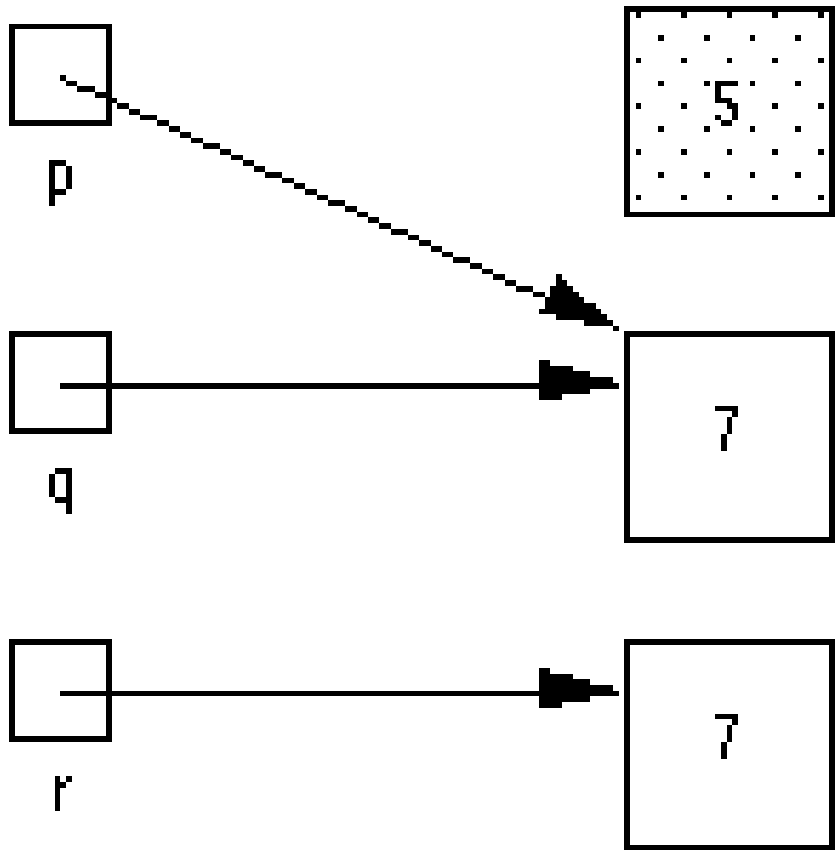
```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
new(q); q↑.x:=7;
new(r); r↑.x:=9;
p:=q;
```



Implementierung von Datent.
Referenzen in Pascal - Standardoper.

Beispiel:

```
type T=record
        x: integer
end;
var p, q, r: ↑T
new(p); p↑.x:=5;
new(q); q↑.x:=7;
new(r); r↑.x:=9;
p:=q;
r↑:=q↑;
```



Implementierung von Datent.
Referenzen in Pascal - Sequenzen

- polymorphe Linkssequenz:

  typ L(D)≡{leer} | (D,L)

- Implementierung in Pascal:

```
type T=record
        inhalt: D
        next: ↑T
end.
```

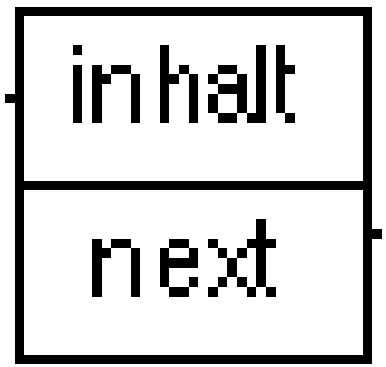
Implementierung von Datent.
Referenzen in Pascal - Sequenzen

- polymorphe Linkssequenz:

  typ L(D)≡{leer} | (D,L)

- Implementierung in Pascal:

```
type T=record
        inhalt: D
        next: ↑T
end.
```

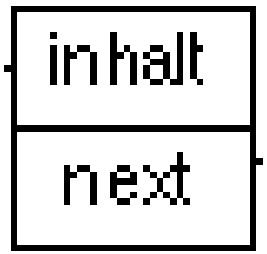


Implementierung von Datent.

Referenzen in Pascal - Sequenzen

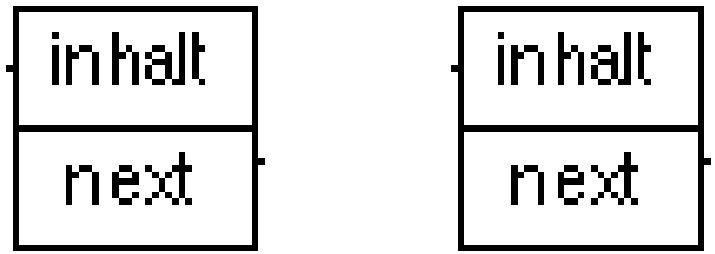
Implementierung von Datent.

Referenzen in Pascal - Sequenzen



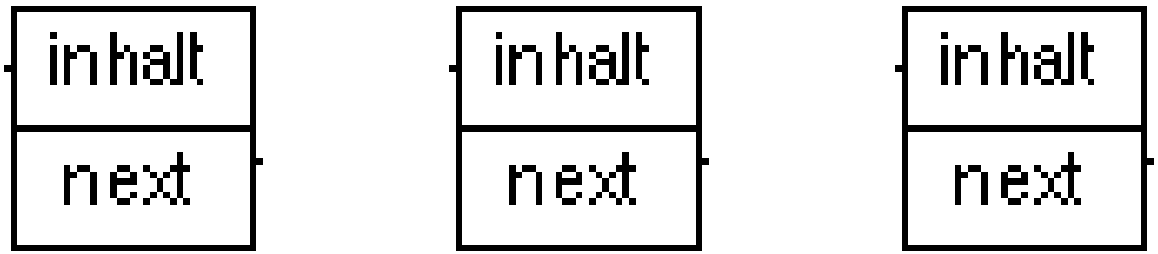
Implementierung von Datent.

Referenzen in Pascal - Sequenzen



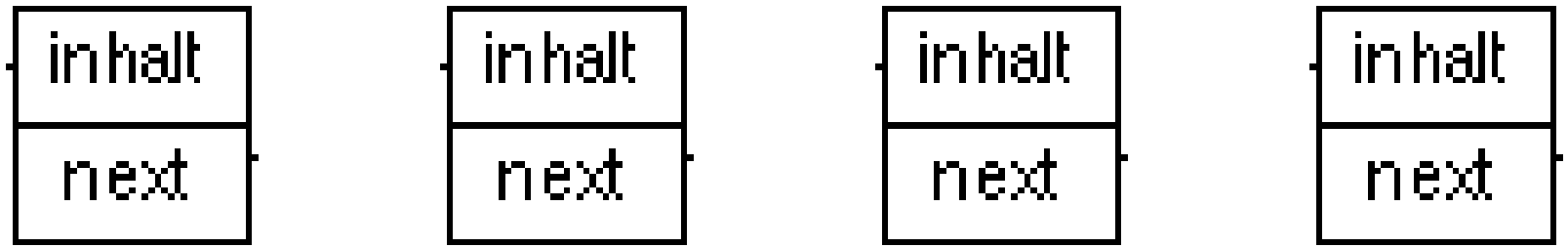
Implementierung von Datent.

Referenzen in Pascal - Sequenzen



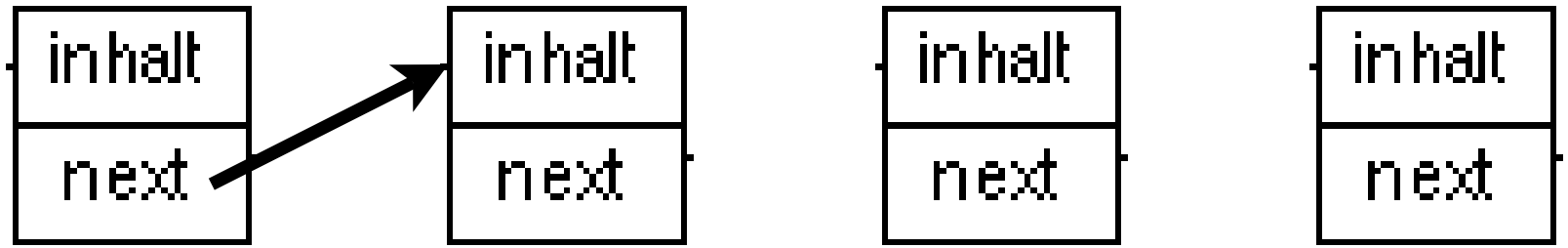
Implementierung von Datent.

Referenzen in Pascal - Sequenzen



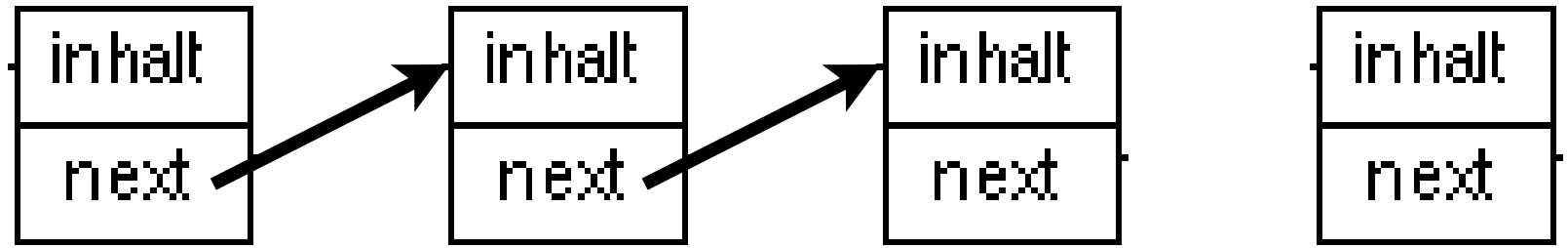
Implementierung von Datent.

Referenzen in Pascal - Sequenzen



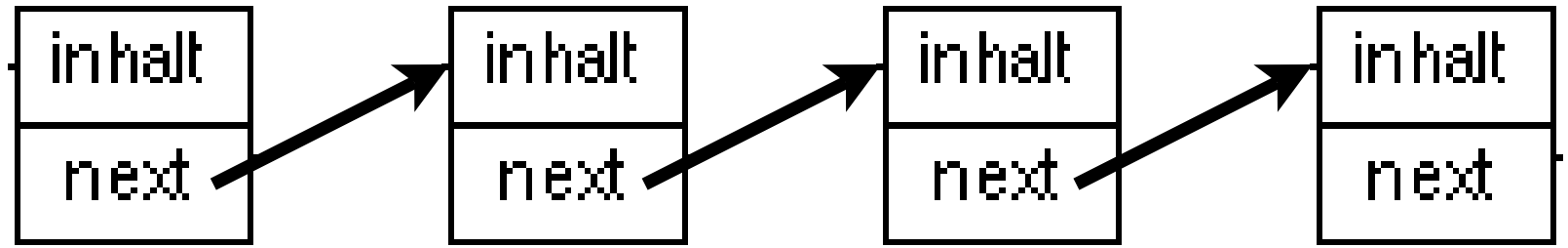
Implementierung von Datent.

Referenzen in Pascal - Sequenzen



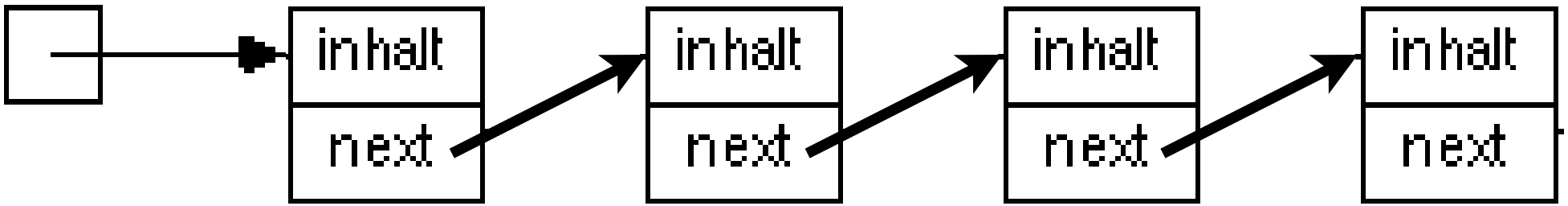
Implementierung von Datent.

Referenzen in Pascal - Sequenzen



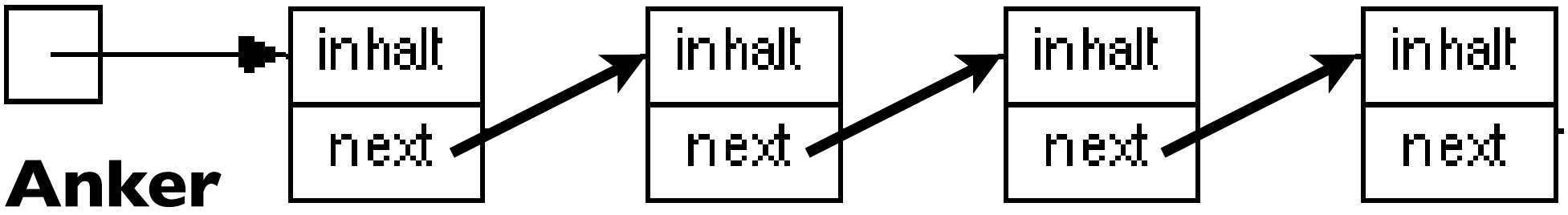
Implementierung von Datent.

Referenzen in Pascal - Sequenzen



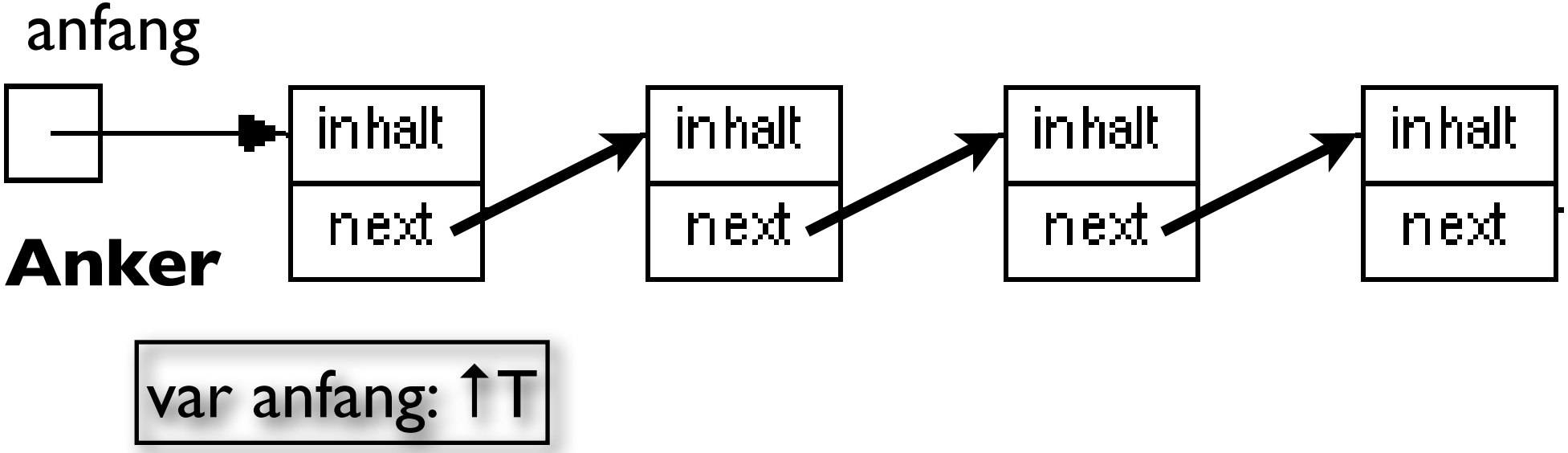
Implementierung von Datent.

Referenzen in Pascal - Sequenzen



Implementierung von Datent.

Referenzen in Pascal - Sequenzen



Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

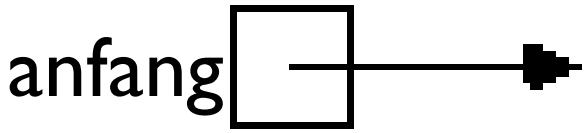
```
var p,q,r,anfang: ↑T
```

Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

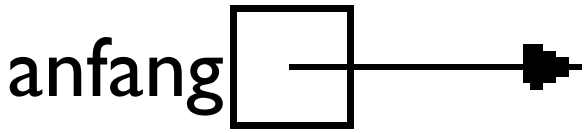


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

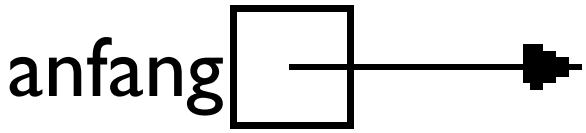


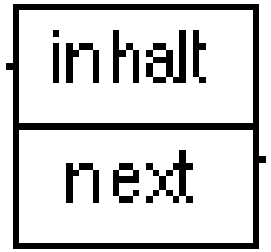
Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

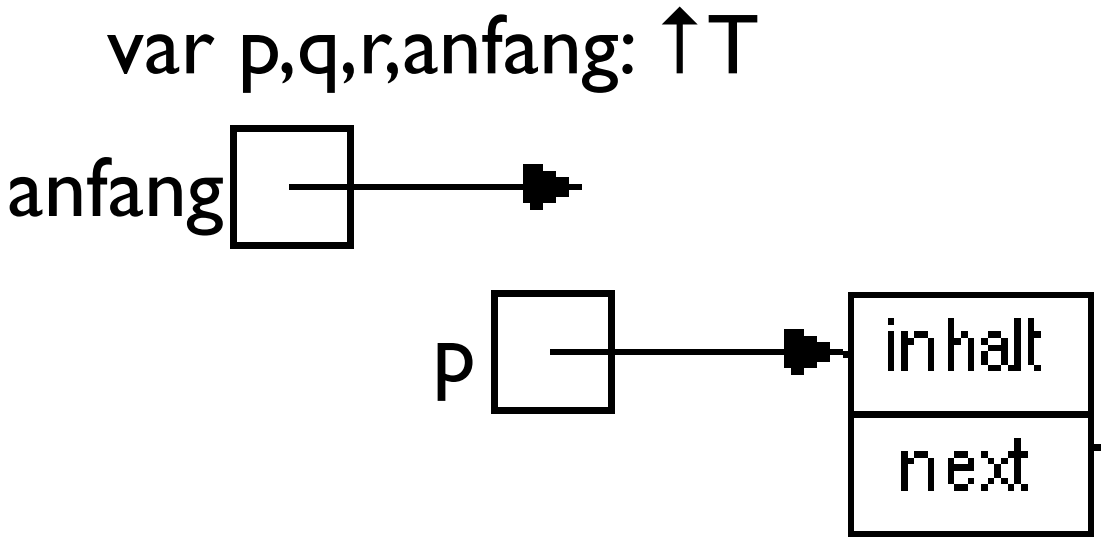




Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

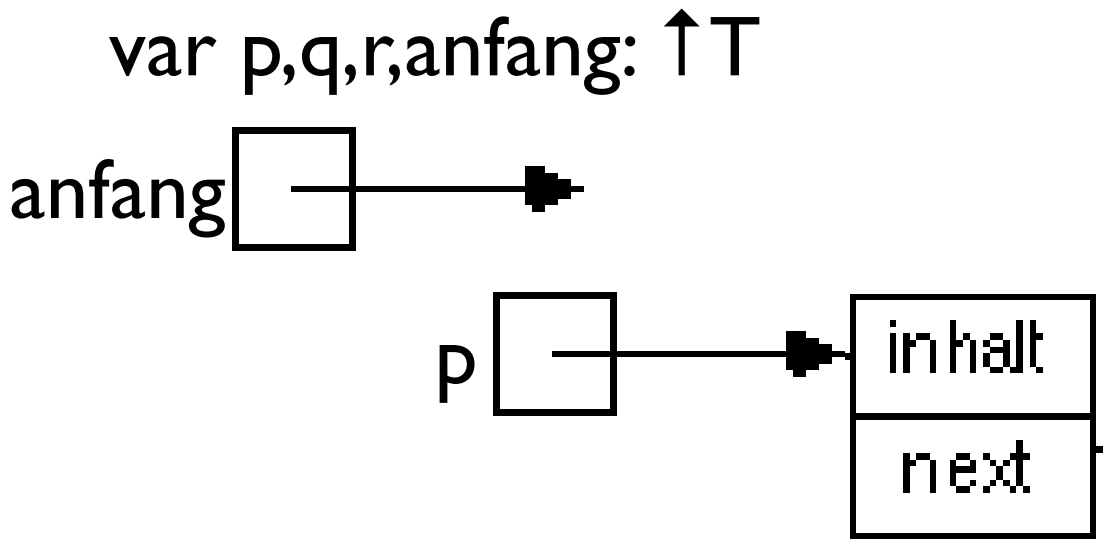
```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```



Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

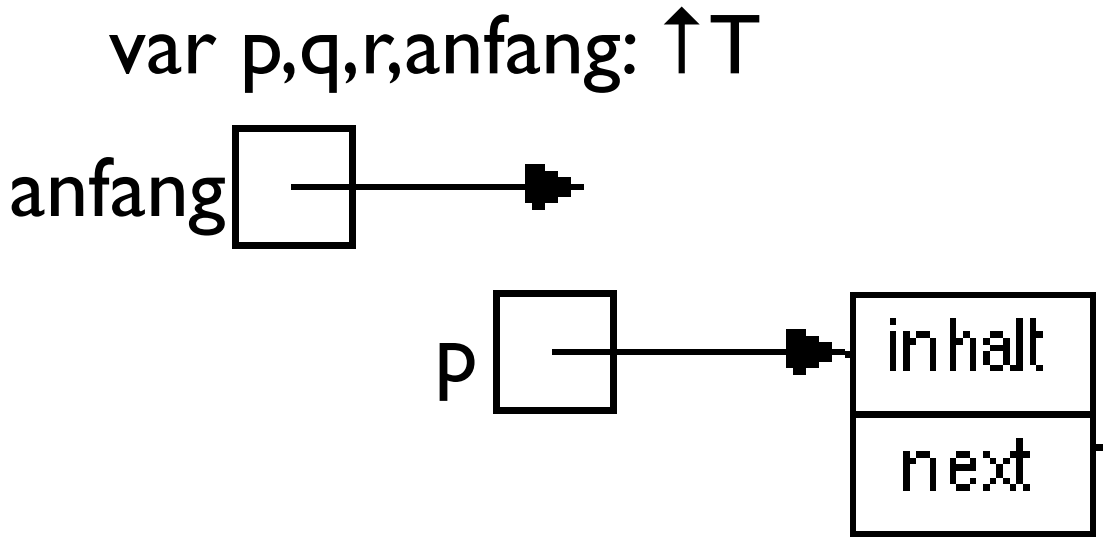
```
anfang:=nil;
for i:=n downto 1 do
begin
     new(p);
     p↑.next:=anfang;
     p↑.inhalt:=ti;
     anfang:=p
end.
```



Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

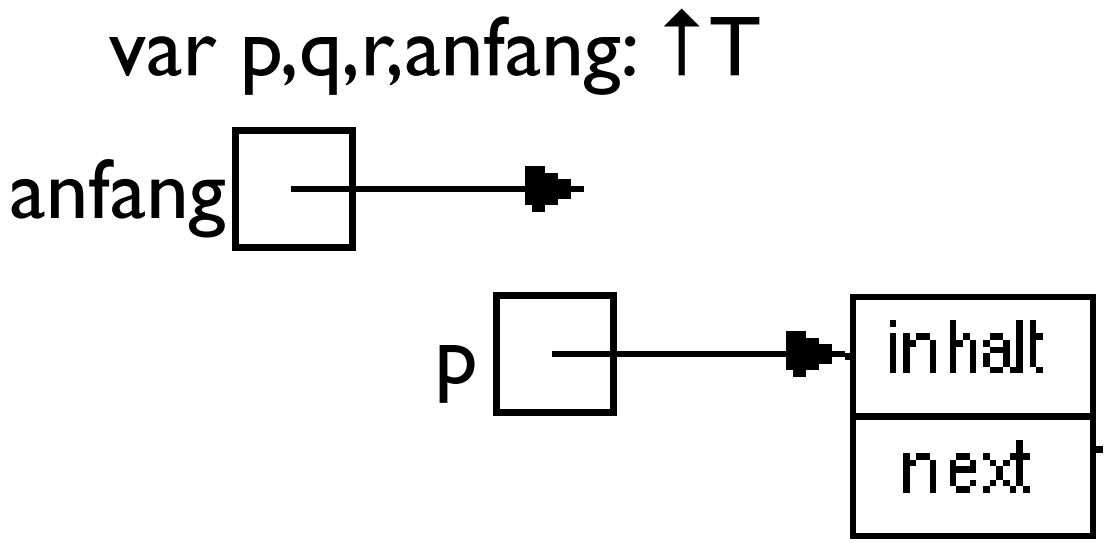
```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```



Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

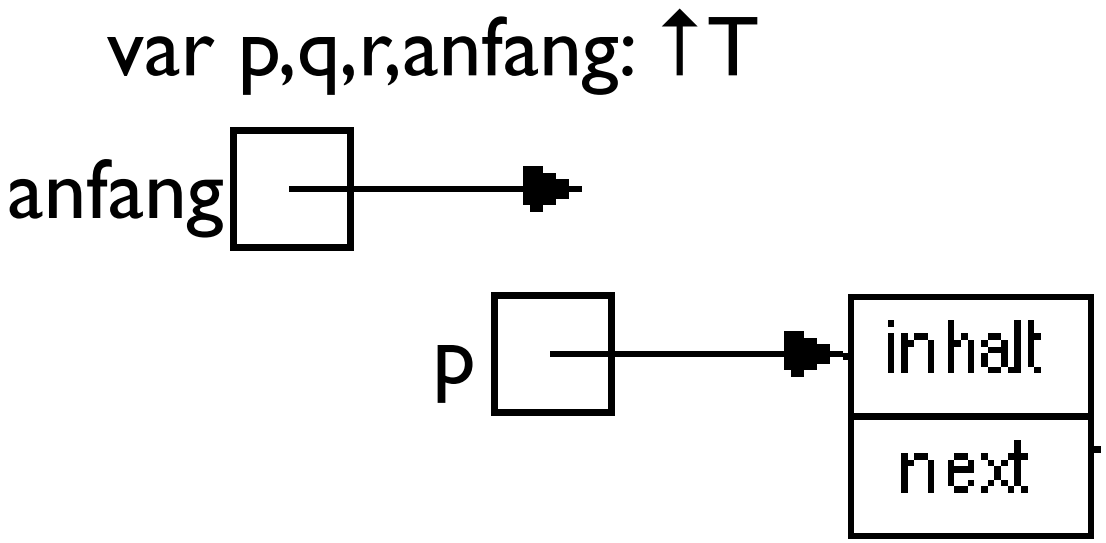
```
anfang:=nil;
for i:=n downto 1 do
begin
     new(p);
     p↑.next:=anfang;
     p↑.inhalt:=ti;
     anfang:=p
end.
```



Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

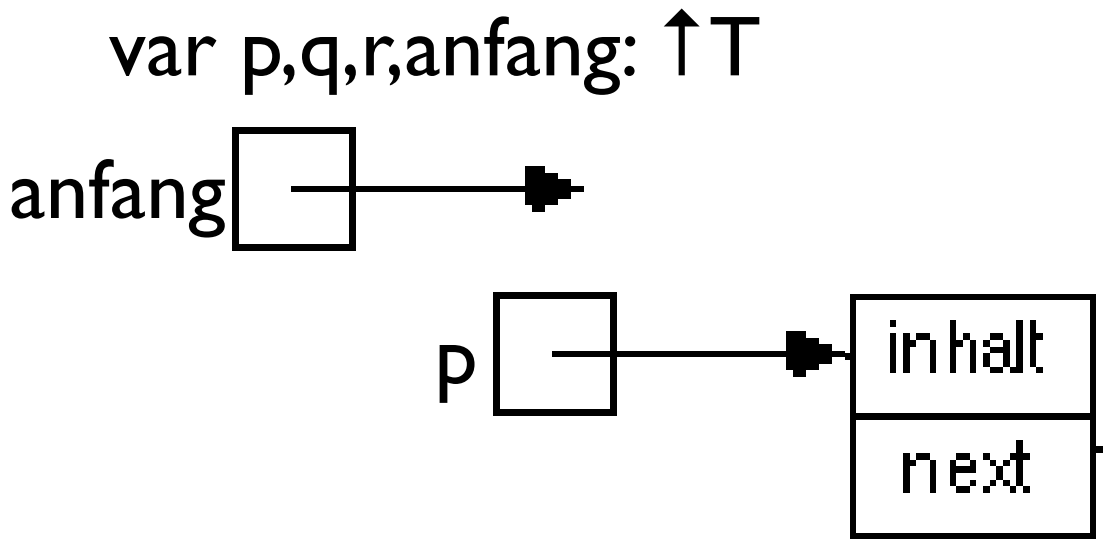
```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```



Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

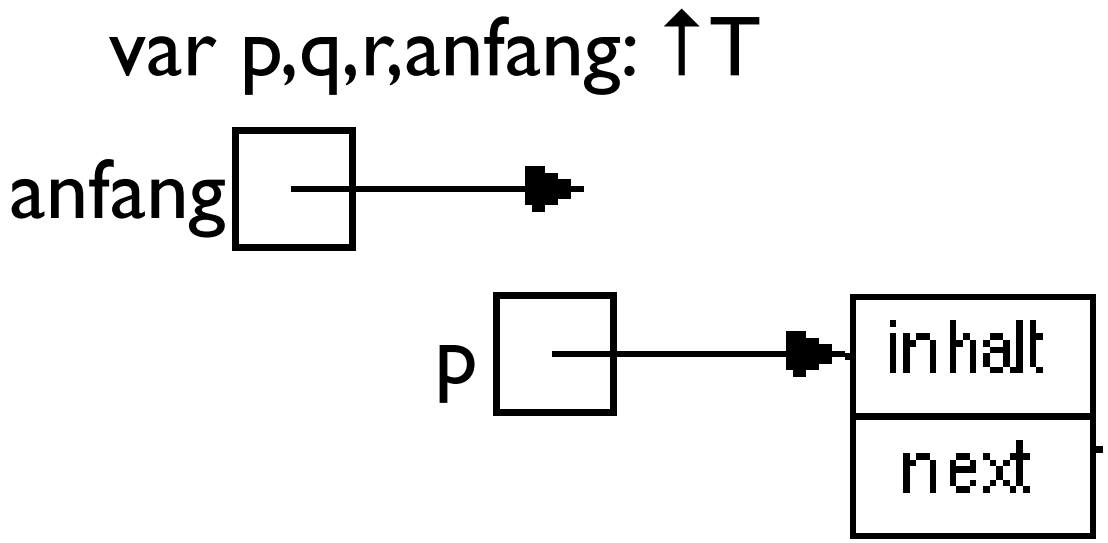
```
anfang:=nil;
for i:=n downto 1 do
begin
     new(p);
     p↑.next:=anfang;
     p↑.inhalt:=ti;
     anfang:=p
end.
```



Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
     new(p);
     p↑.next:=anfang;
     p↑.inhalt:=ti;
     anfang:=p
end.
```

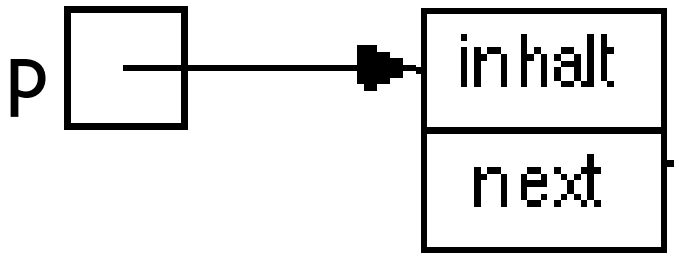


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
     new(p);
     p↑.next:=anfang;
     p↑.inhalt:=ti;
     anfang:=p
end.
```

var p,q,r,anfang: ↑T

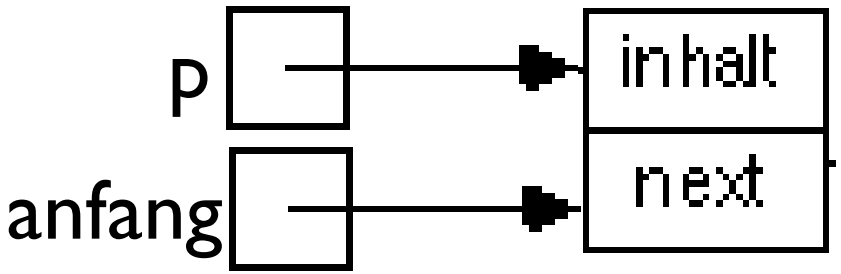


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

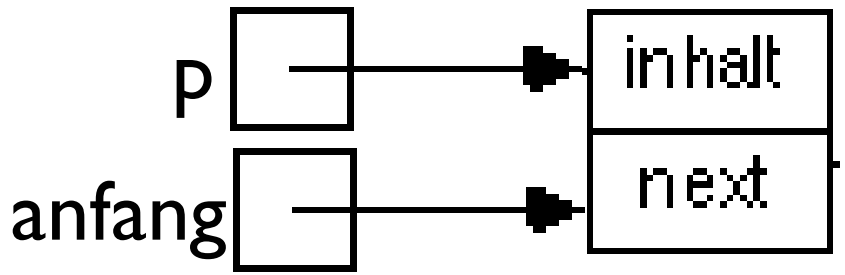


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

```
var p,q,r,anfang: ↑T
```

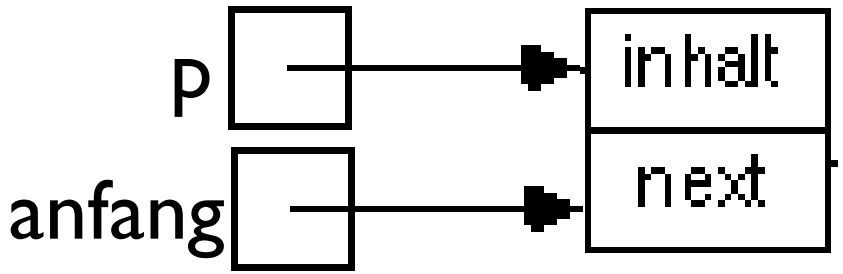


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

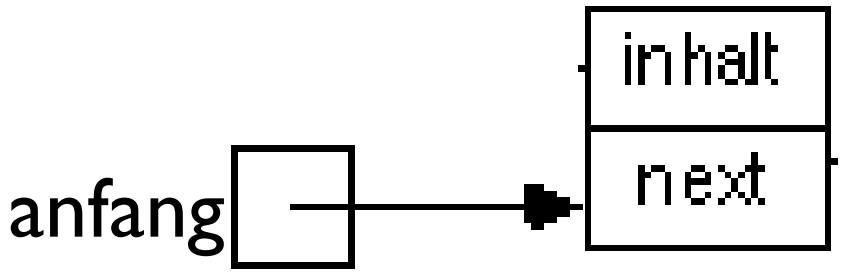


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

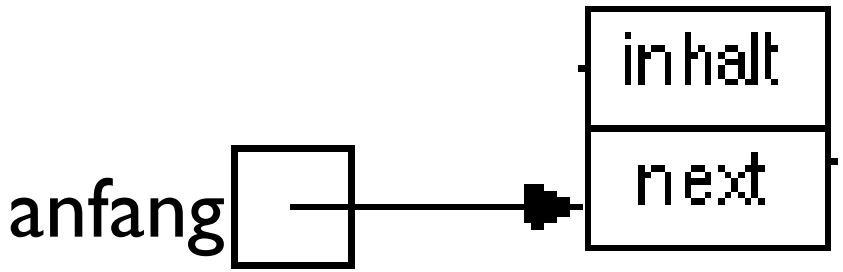


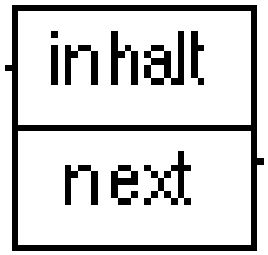
Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T



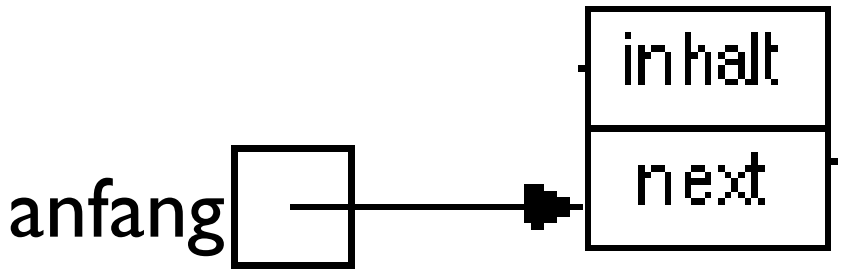


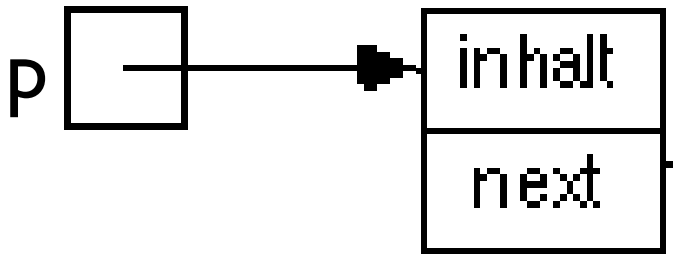
Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T



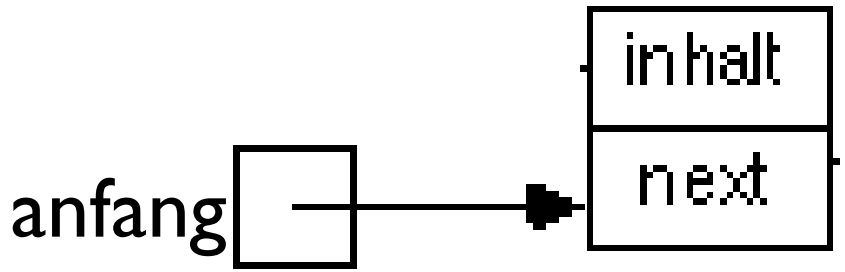


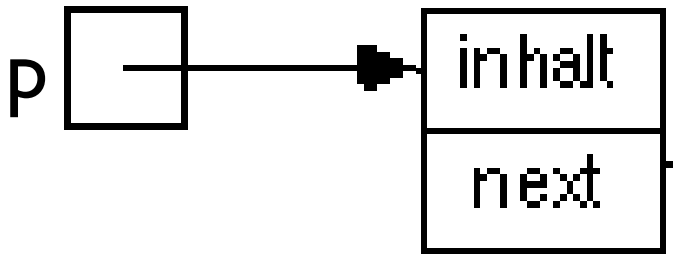
Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

```
var p,q,r,anfang: ↑T
```



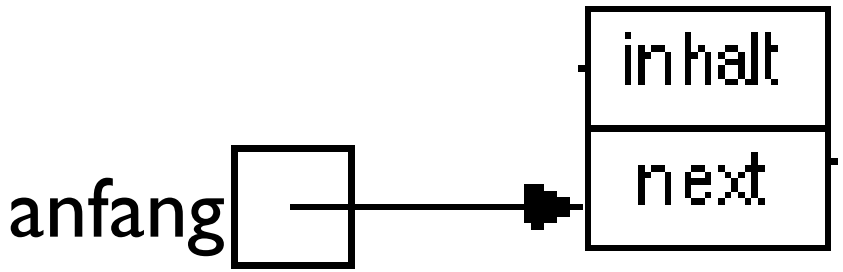


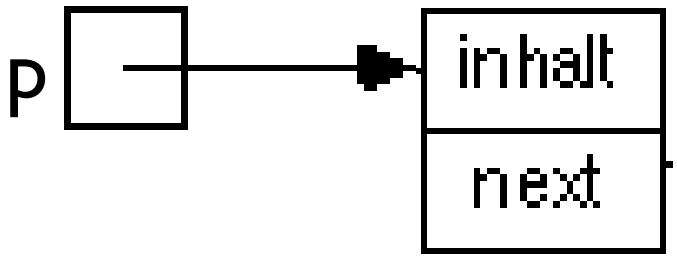
Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T



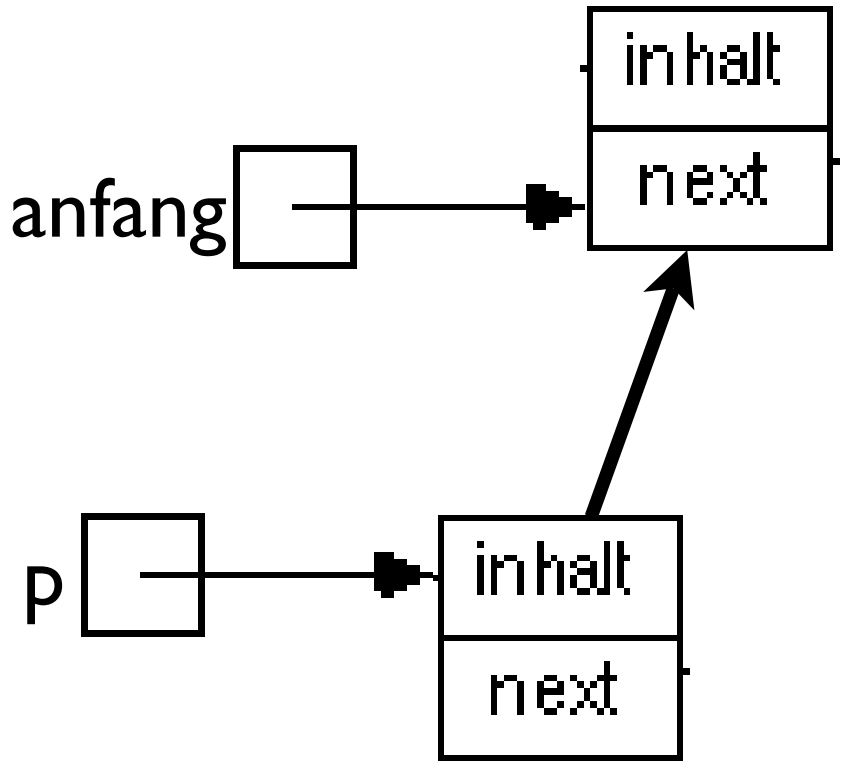


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

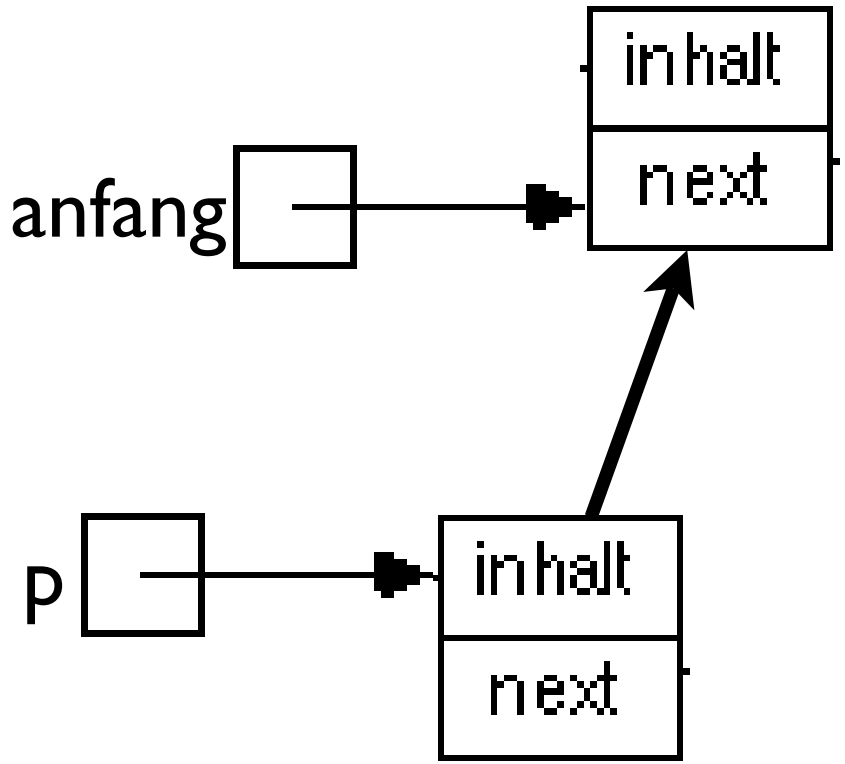


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

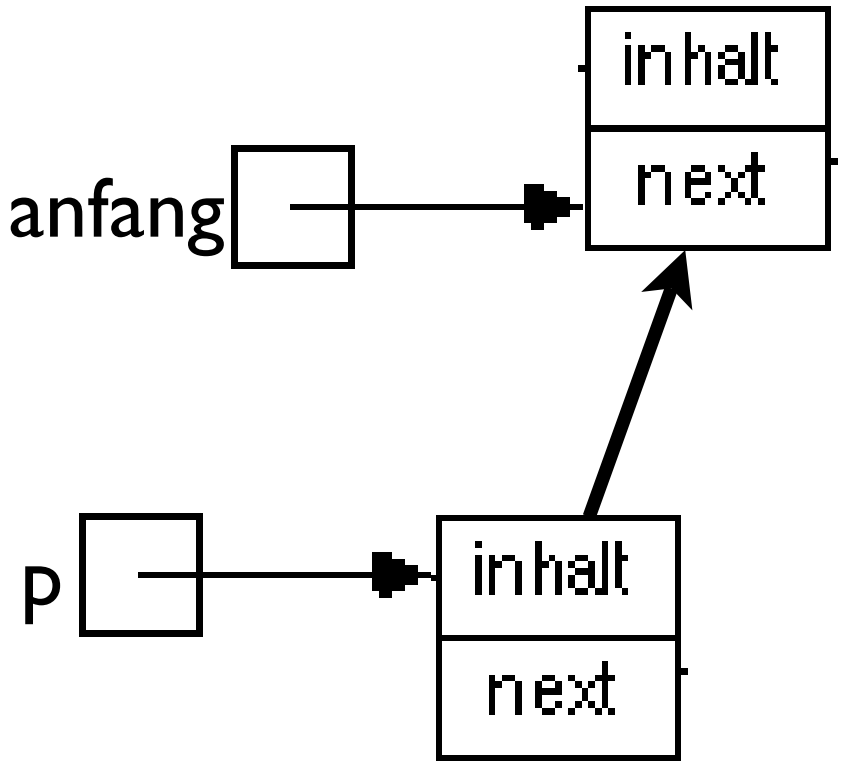


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

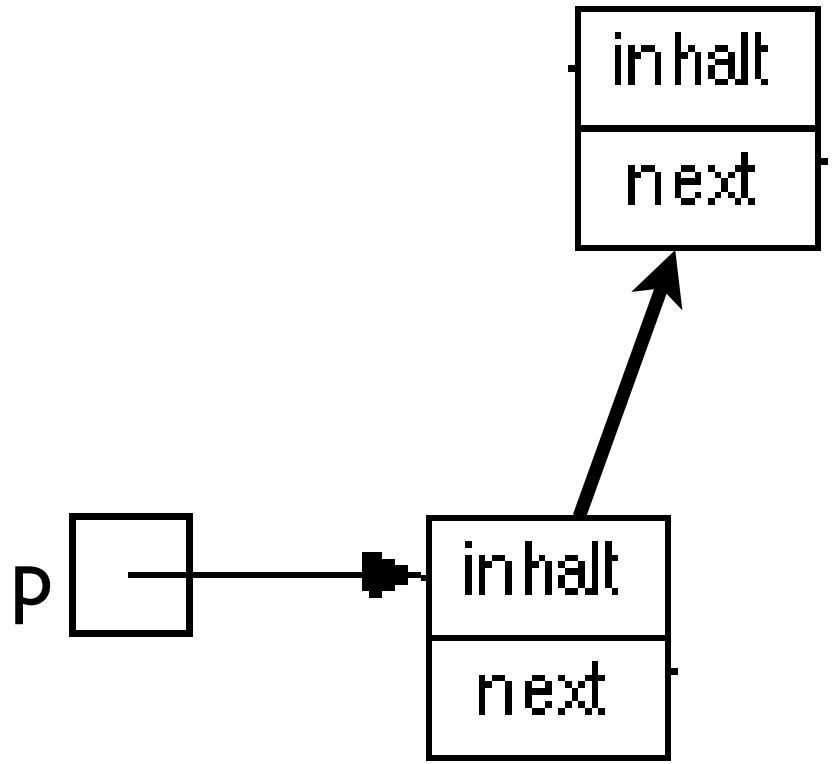


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

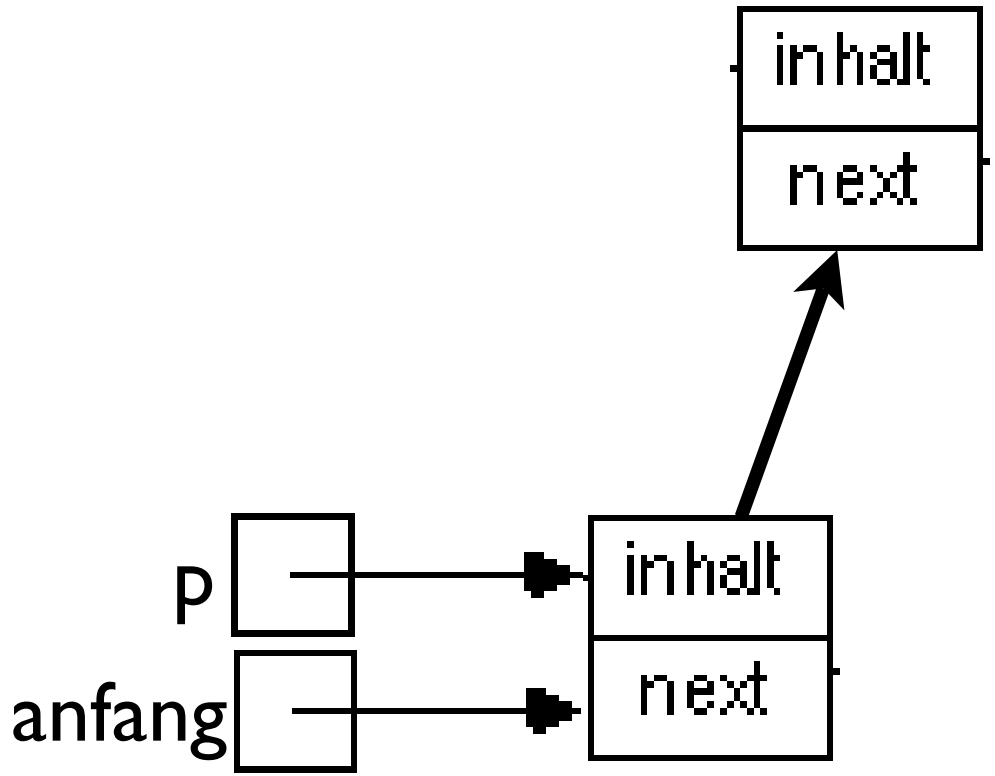


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

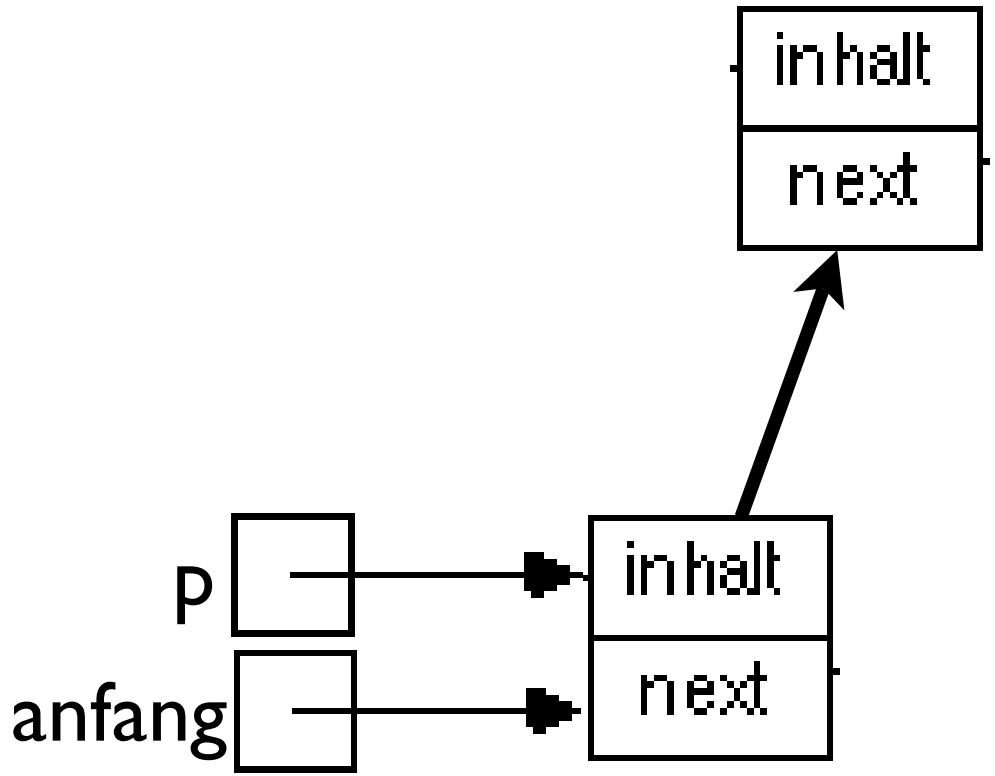


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

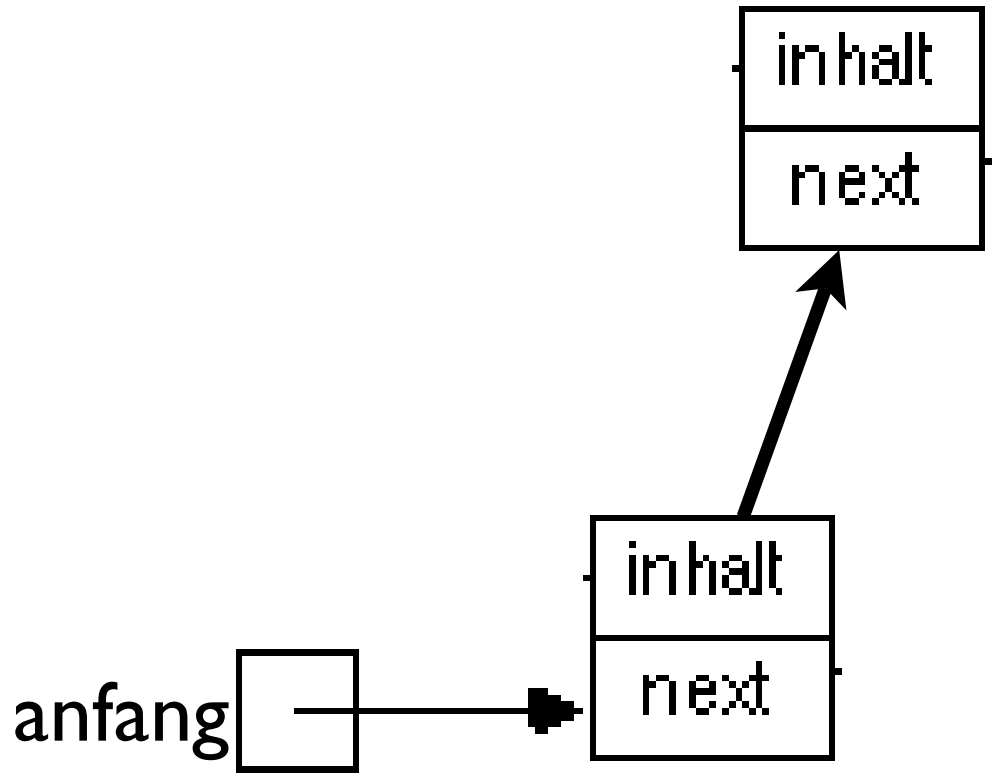


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

```
var p,q,r,anfang: ↑T
```

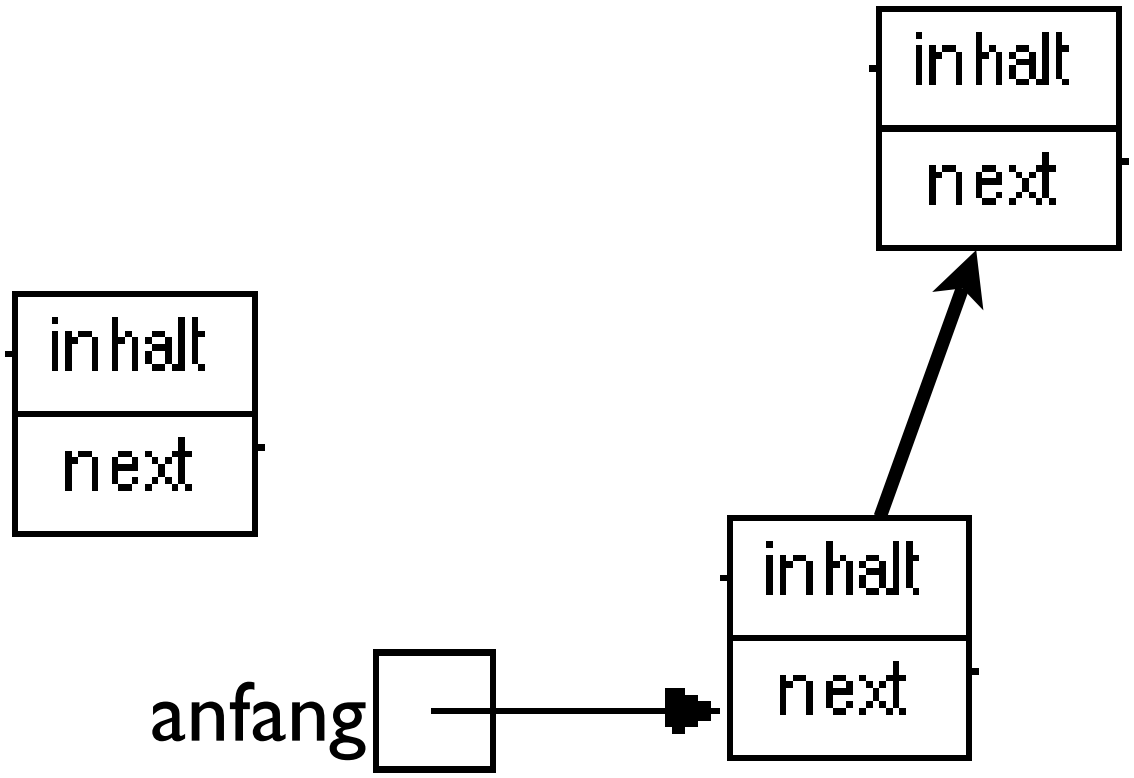


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

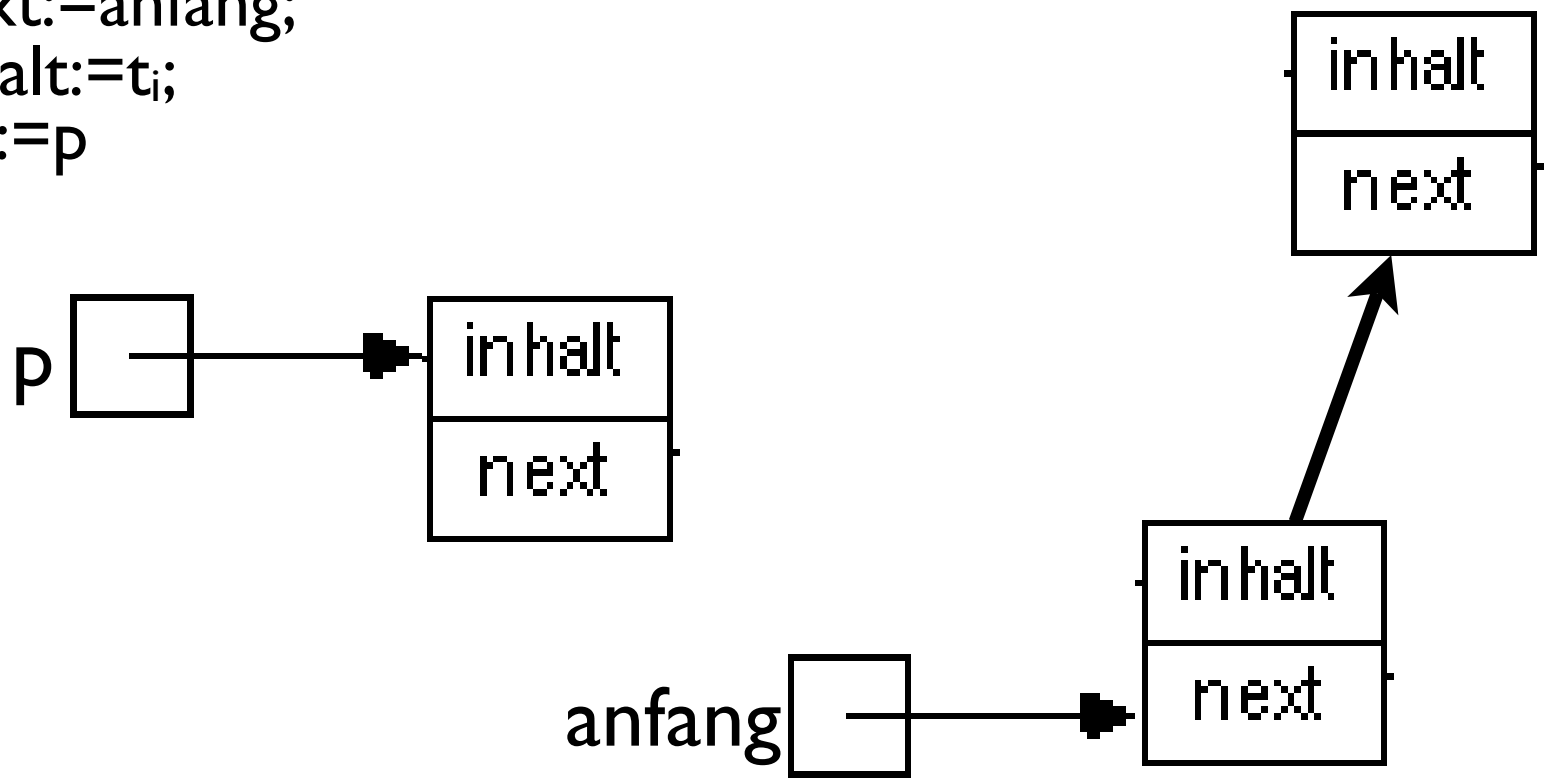


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

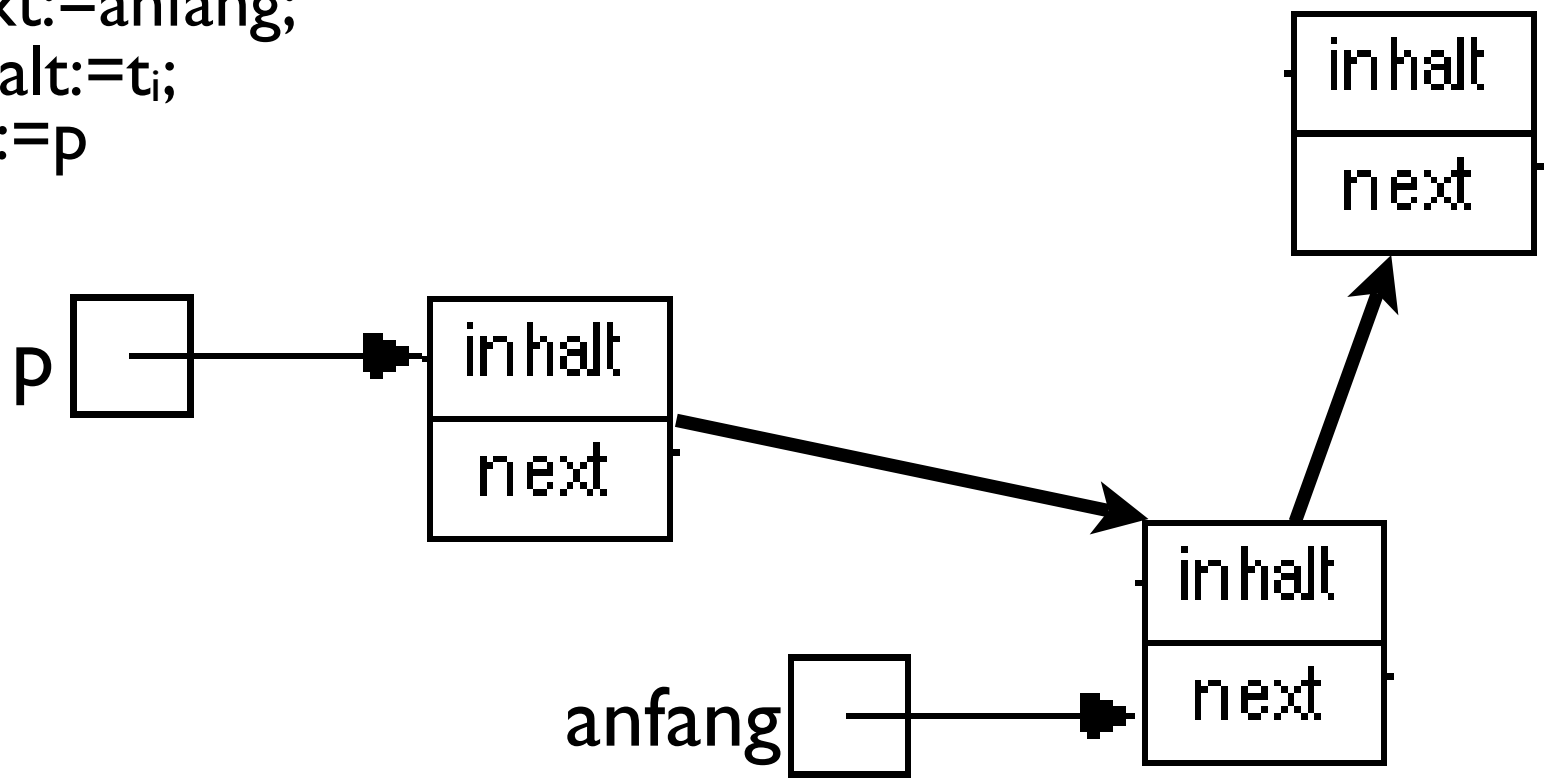


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T

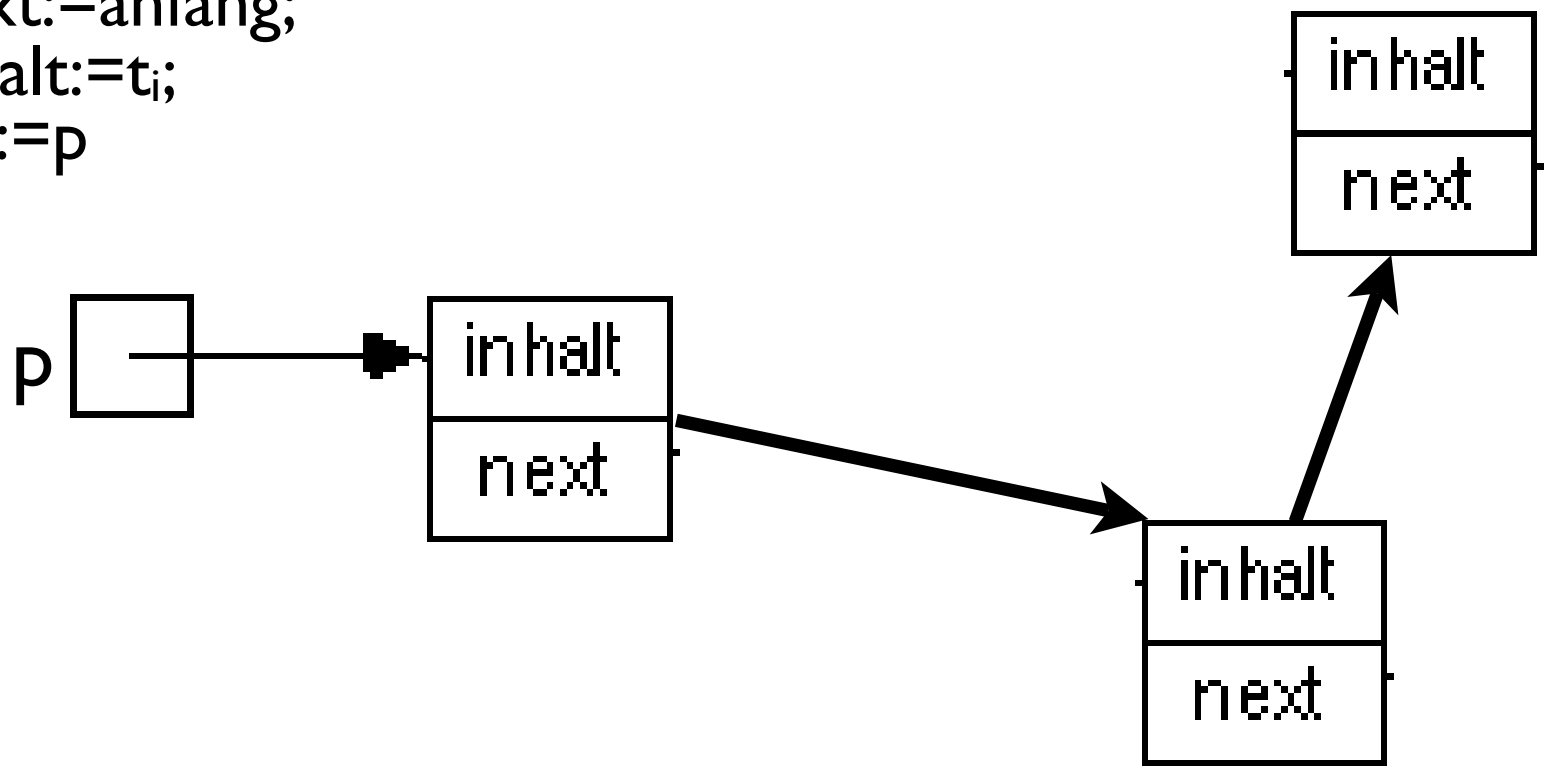


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

```
anfang:=nil;
for i:=n downto 1 do
begin
     new(p);
     p↑.next:=anfang;
     p↑.inhalt:=ti;
     anfang:=p
end.
```

var p,q,r,anfang: ↑T

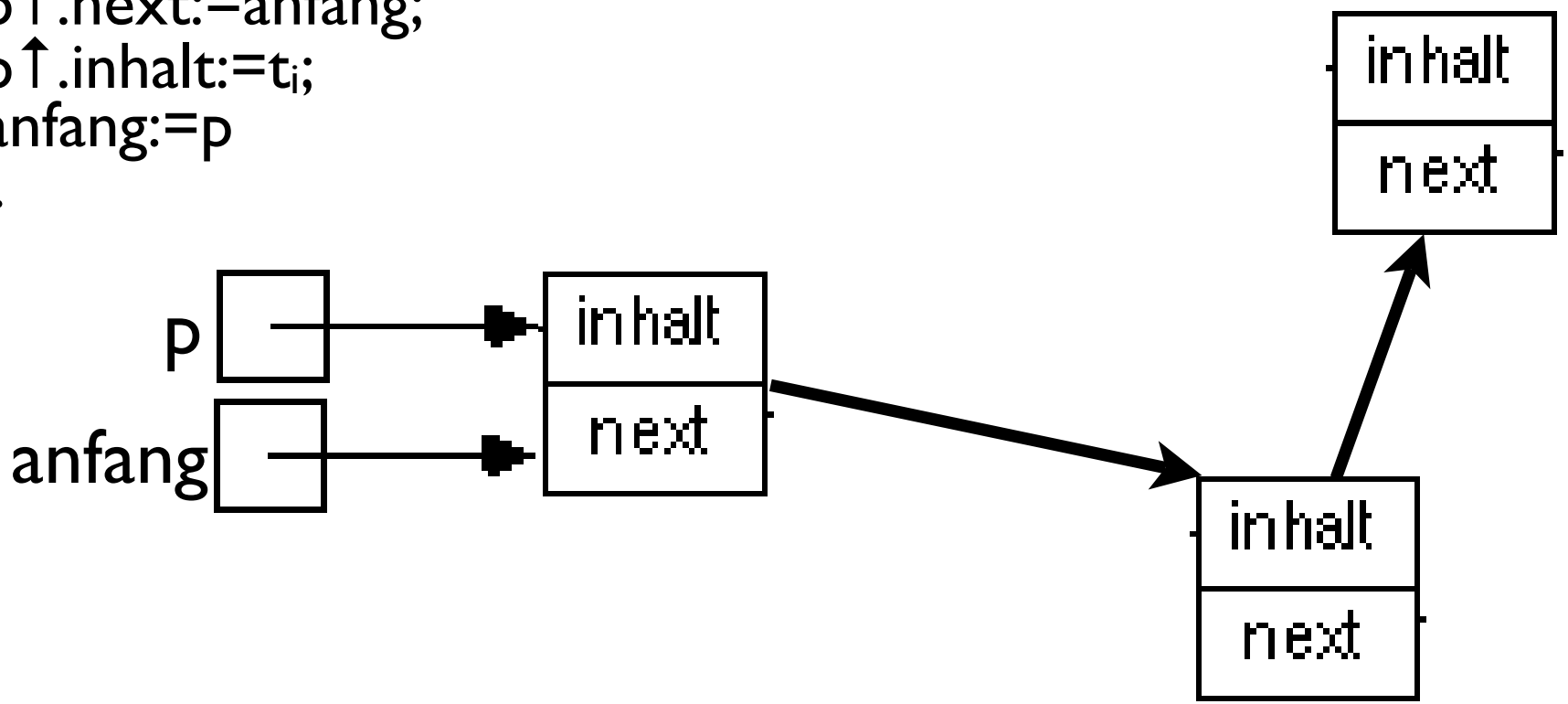


Implementierung von Datent.

Sequenzen - Grundoperationen: Aufbau

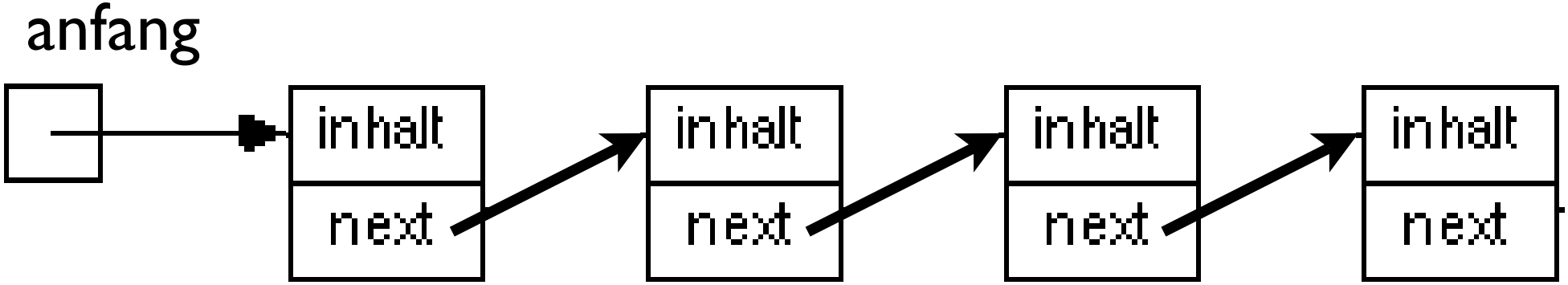
```
anfang:=nil;
for i:=n downto 1 do
begin
    new(p);
    p↑.next:=anfang;
    p↑.inhalt:=ti;
    anfang:=p
end.
```

var p,q,r,anfang: ↑T



Implementierung von Datent.

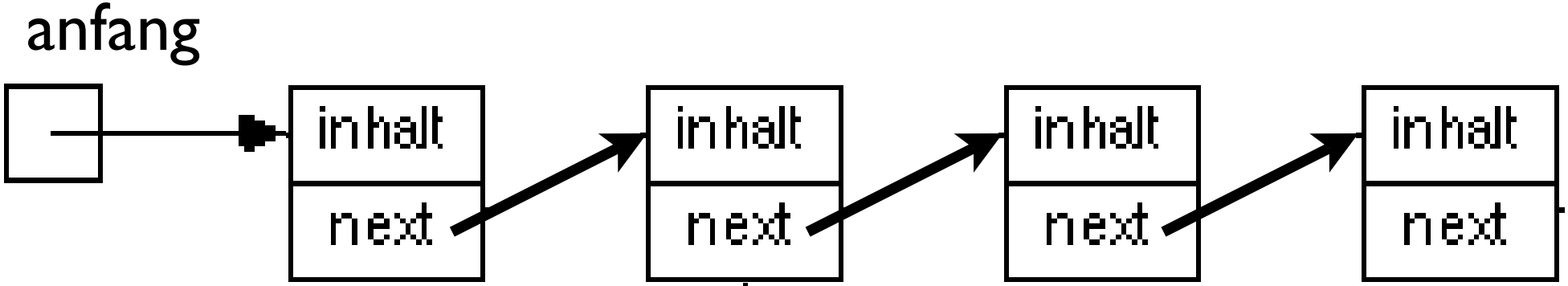
Sequenzen - Einfügen




```
new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.
```

Implementierung von Datent.

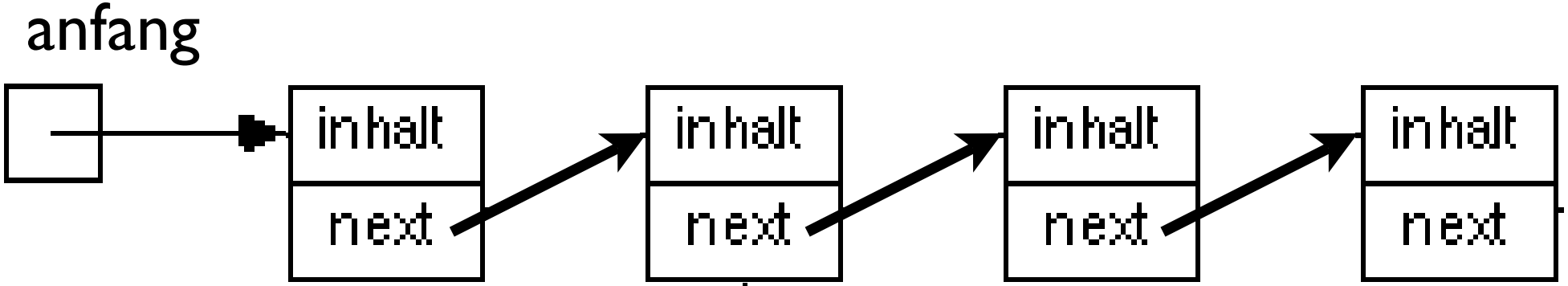
Sequenzen - Einfügen




```
new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.
```

Implementierung von Datent.

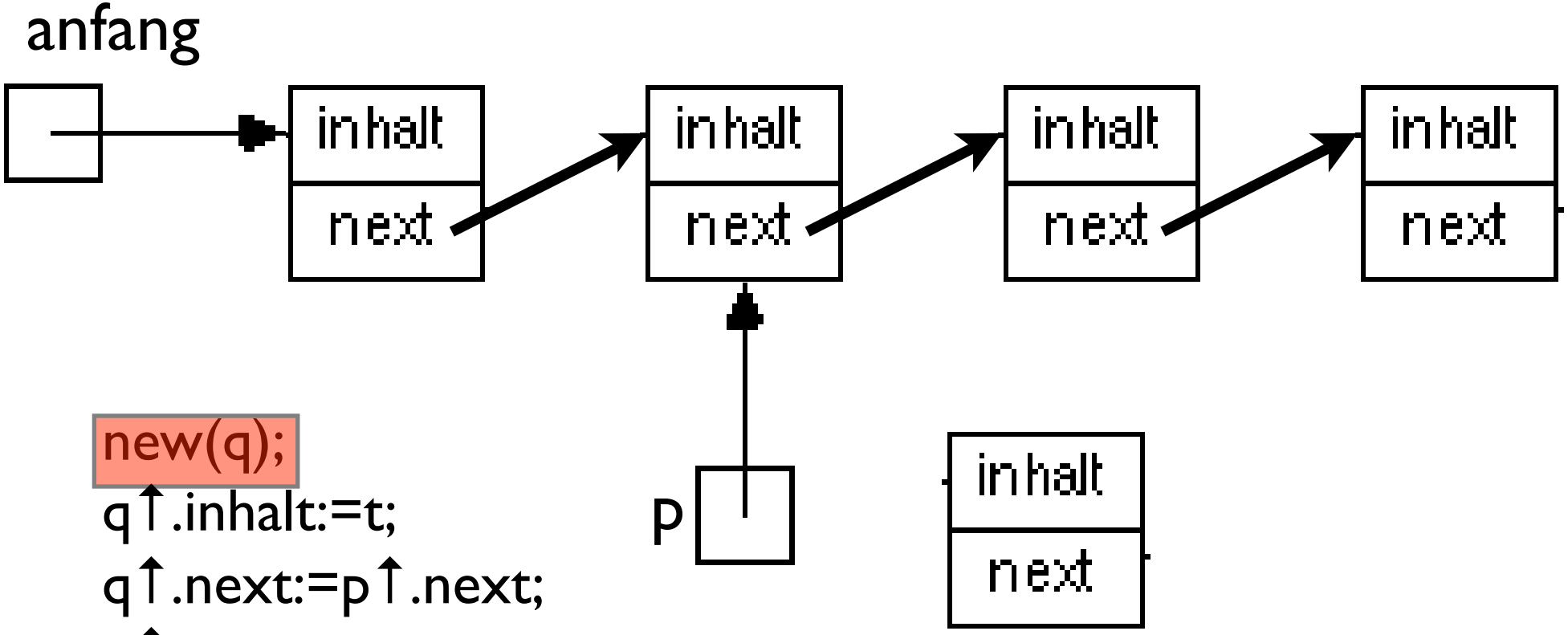
Sequenzen - Einfügen



new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.

Implementierung von Datent.

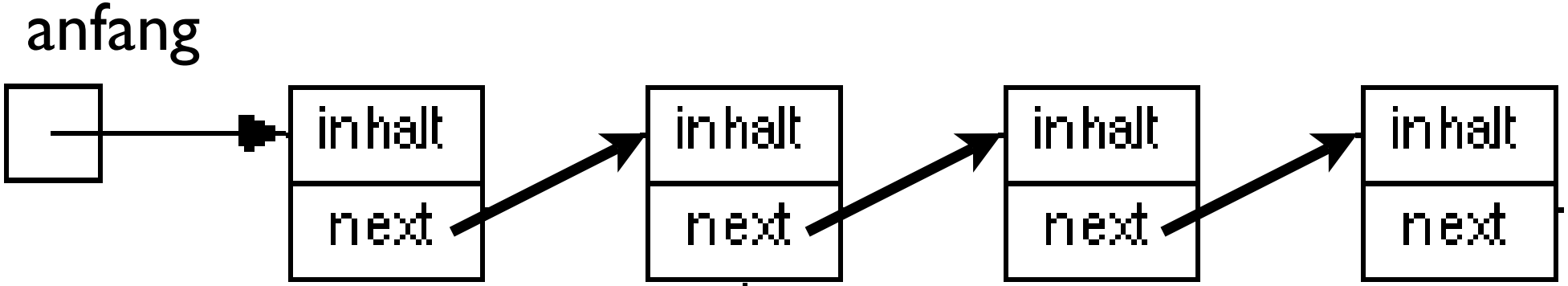
Sequenzen - Einfügen



new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.

Implementierung von Datent.

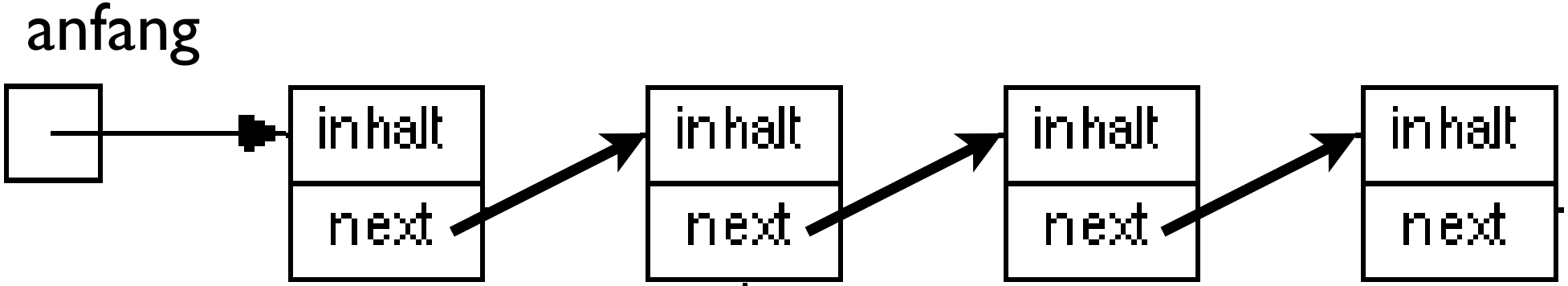
Sequenzen - Einfügen



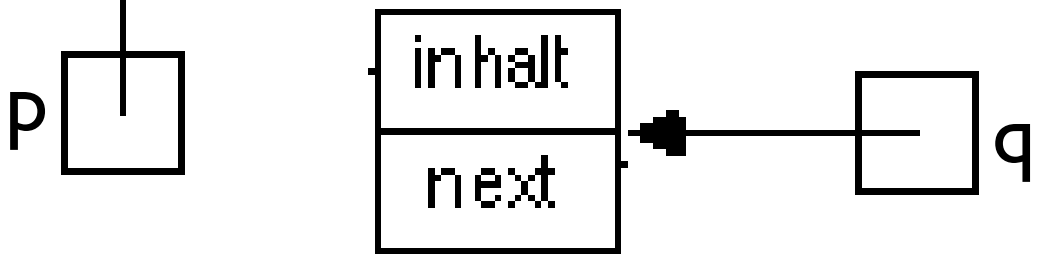
new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.

Implementierung von Datent.

Sequenzen - Einfügen

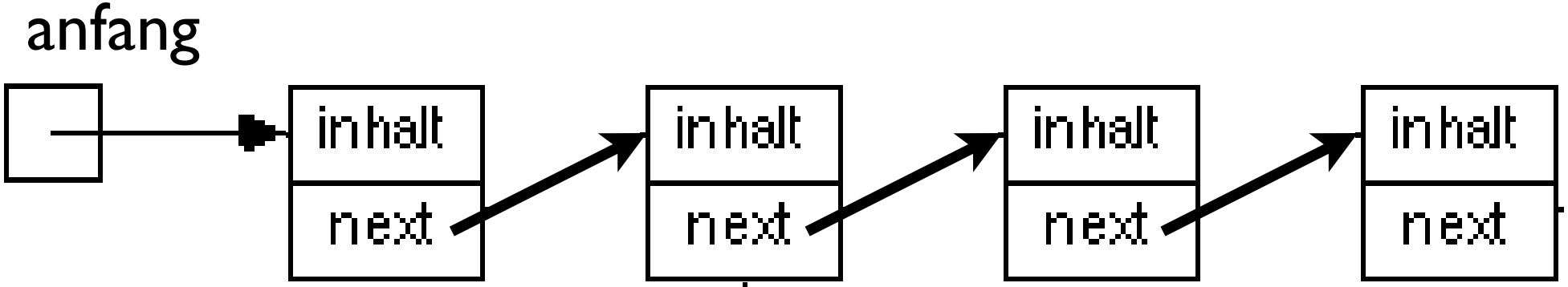


new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.

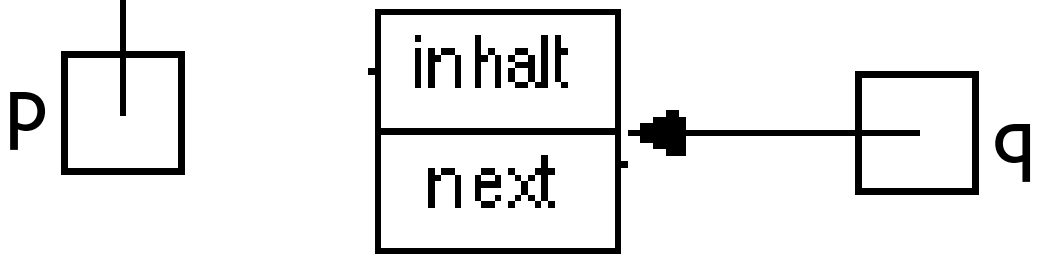


Implementierung von Datent.

Sequenzen - Einfügen

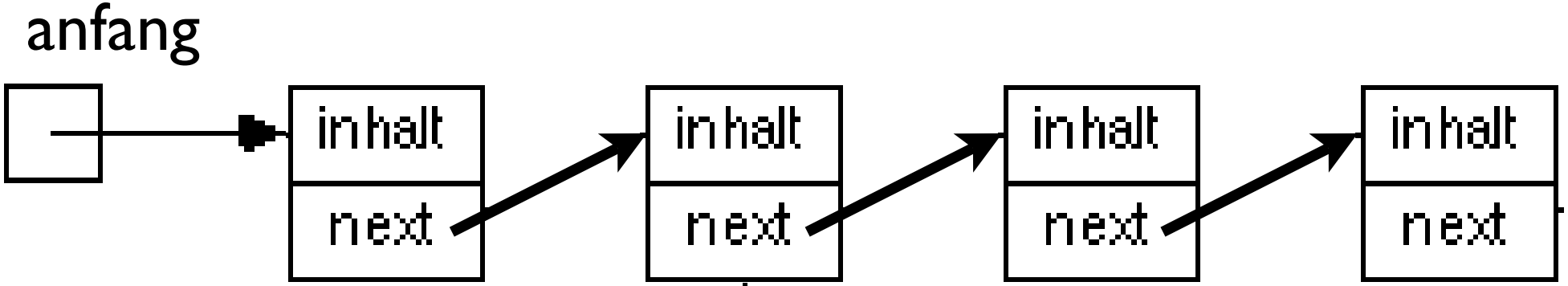



```
new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.
```

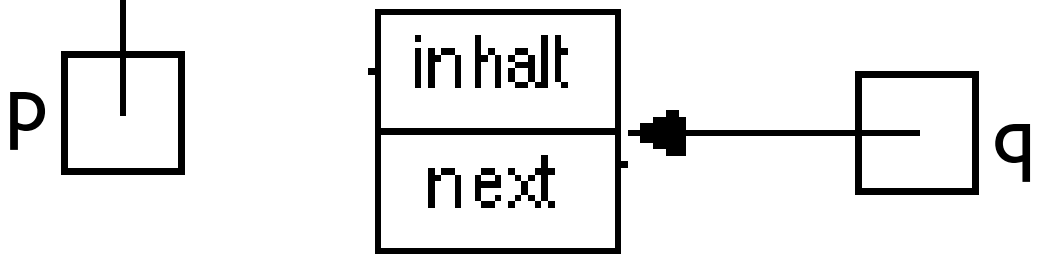


Implementierung von Datent.

Sequenzen - Einfügen

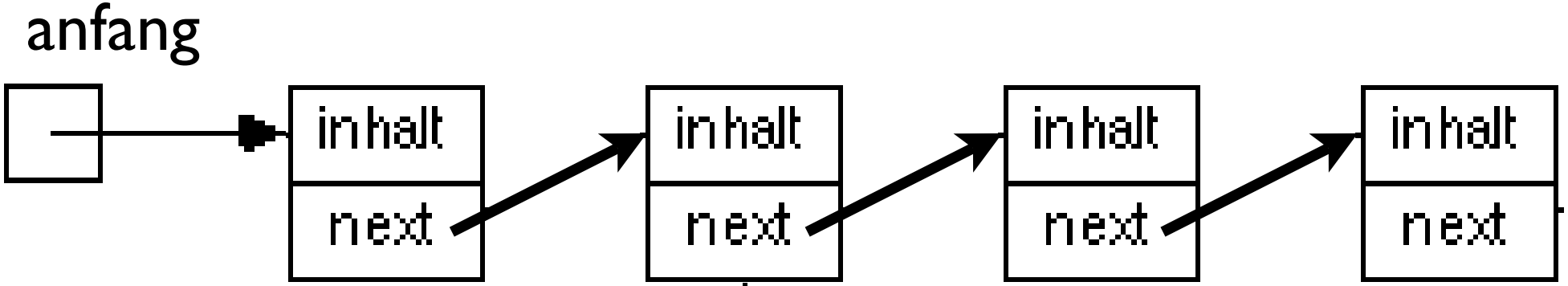


new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.



Implementierung von Datent.

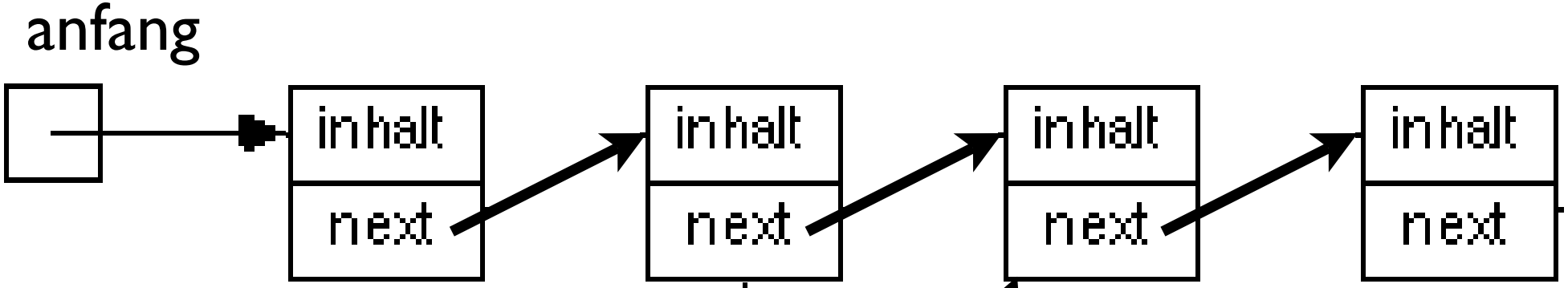
Sequenzen - Einfügen

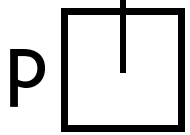


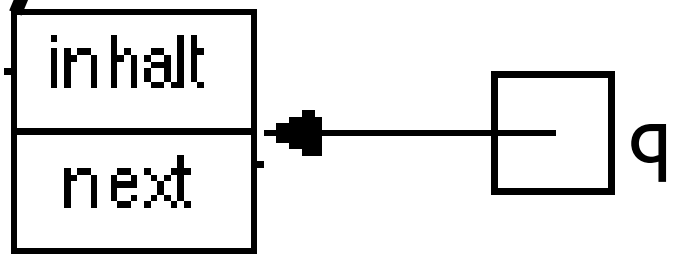

```
new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.
```

Implementierung von Datent.

Sequenzen - Einfügen



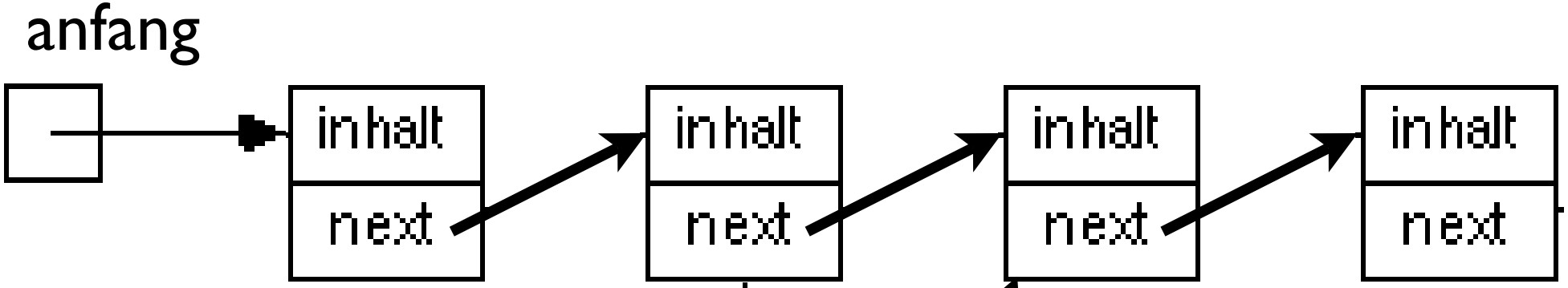


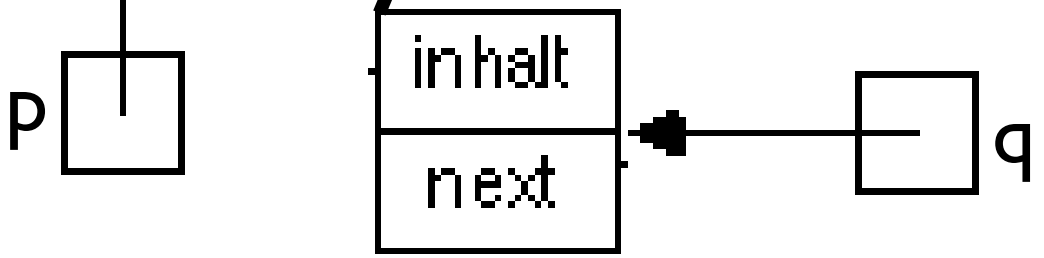



```
new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.
```

Implementierung von Datent.

Sequenzen - Einfügen

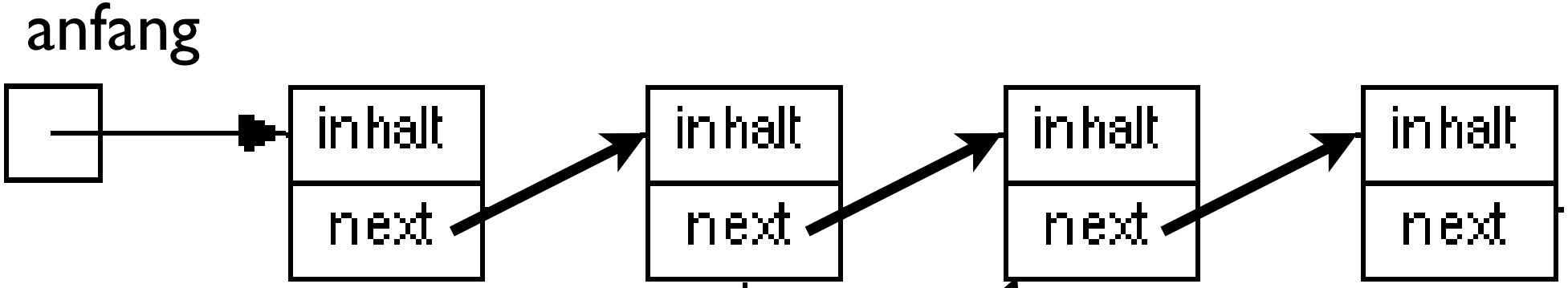





```
new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.
```

Implementierung von Datent.

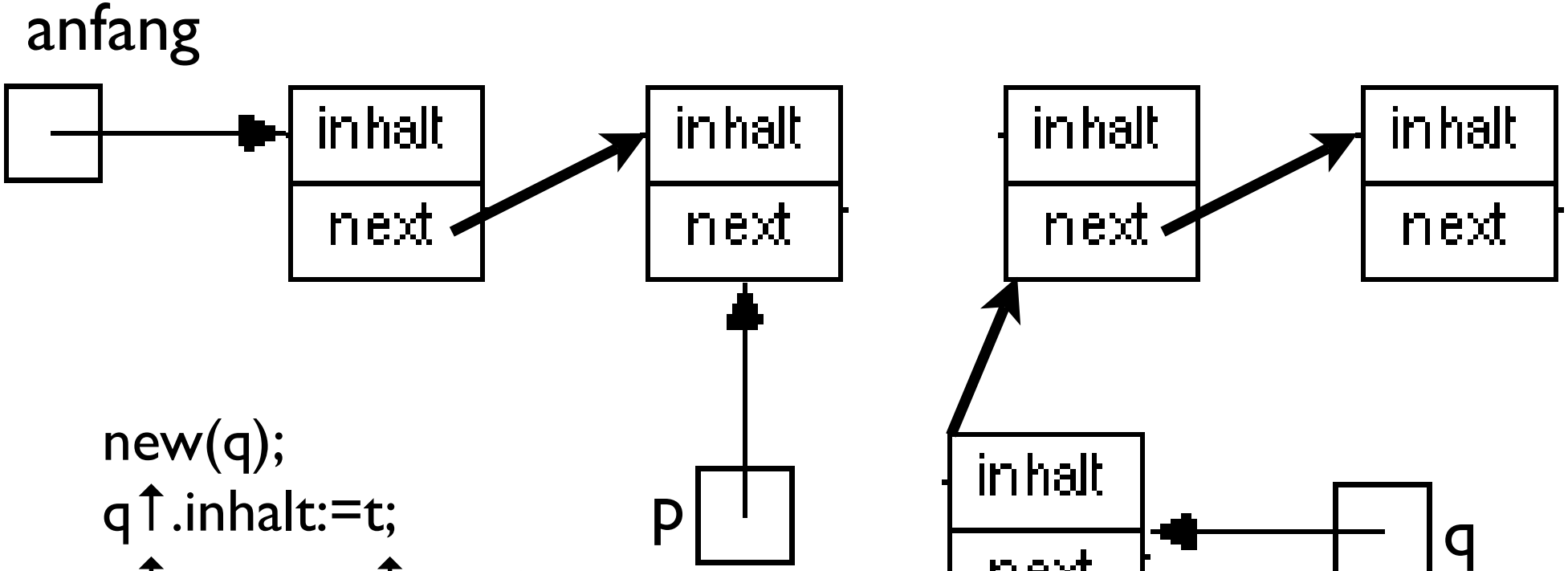
Sequenzen - Einfügen



new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.

Implementierung von Datent.

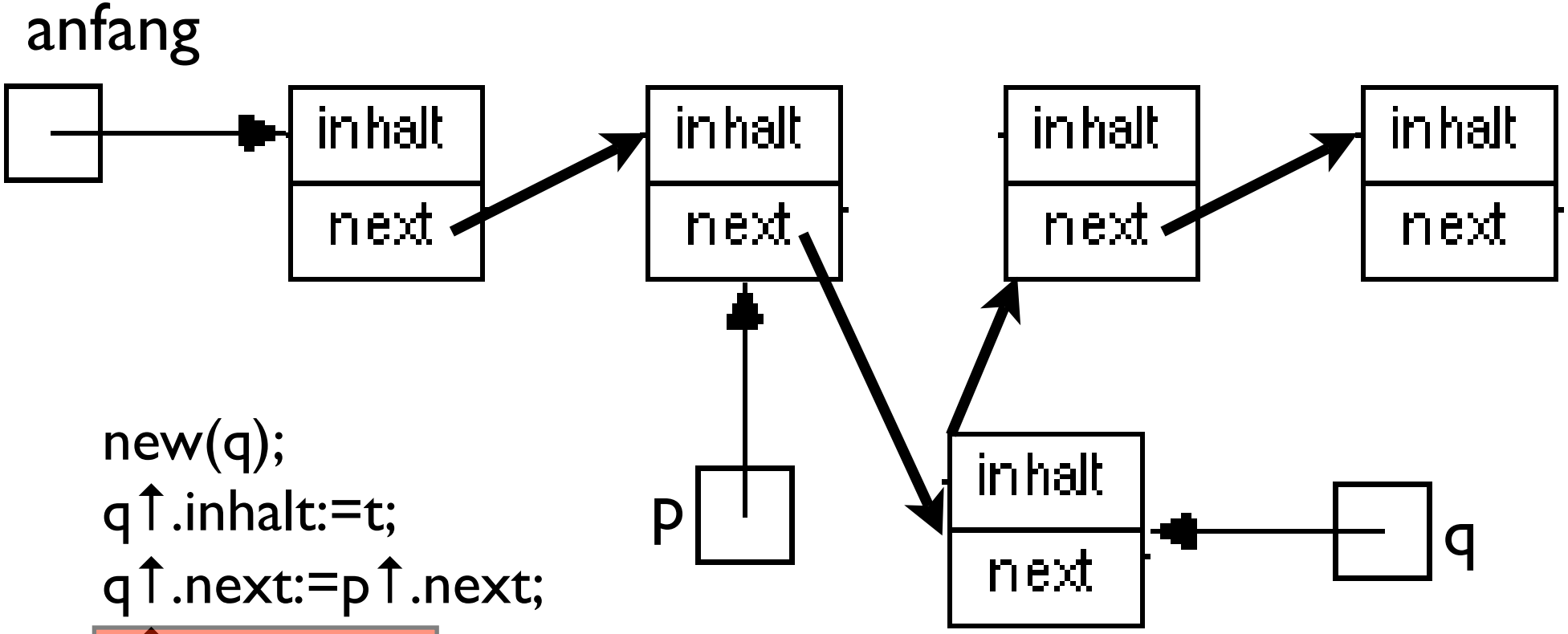
Sequenzen - Einfügen



new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.

Implementierung von Datent.

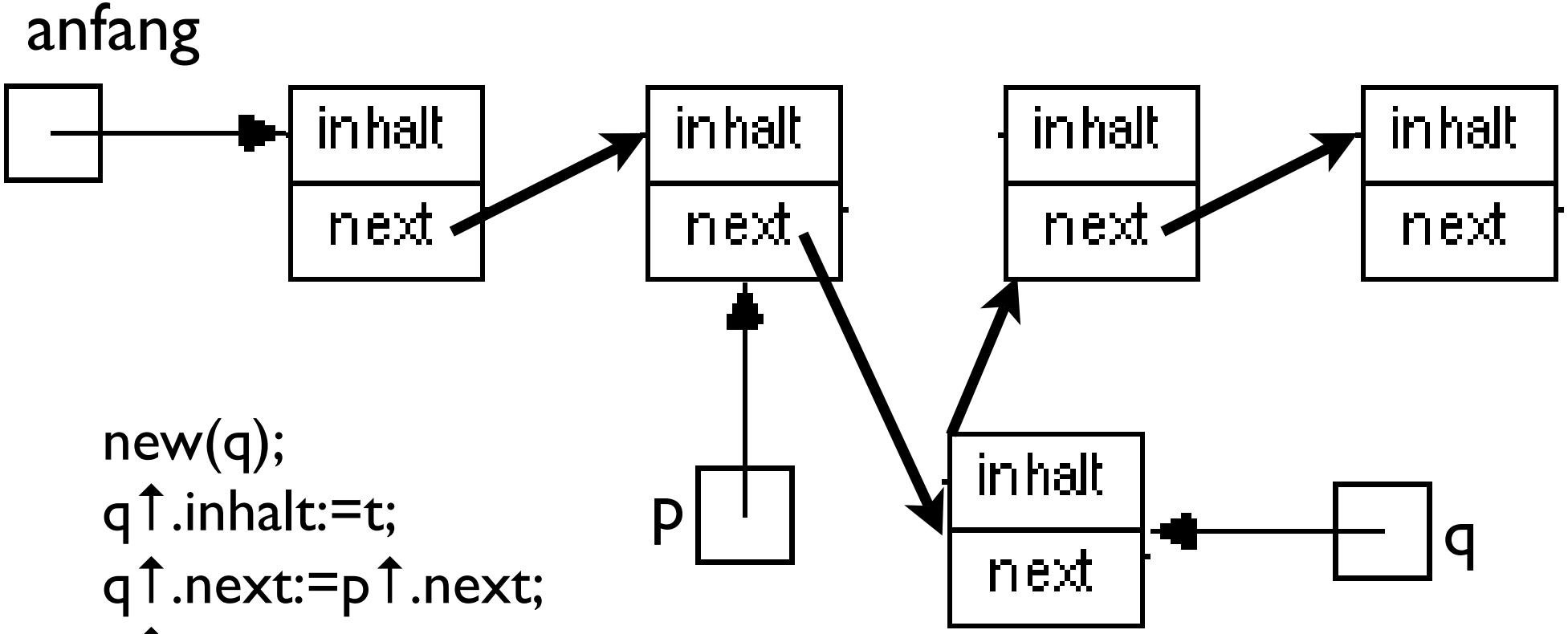
Sequenzen - Einfügen




```
new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.
```

Implementierung von Datent.

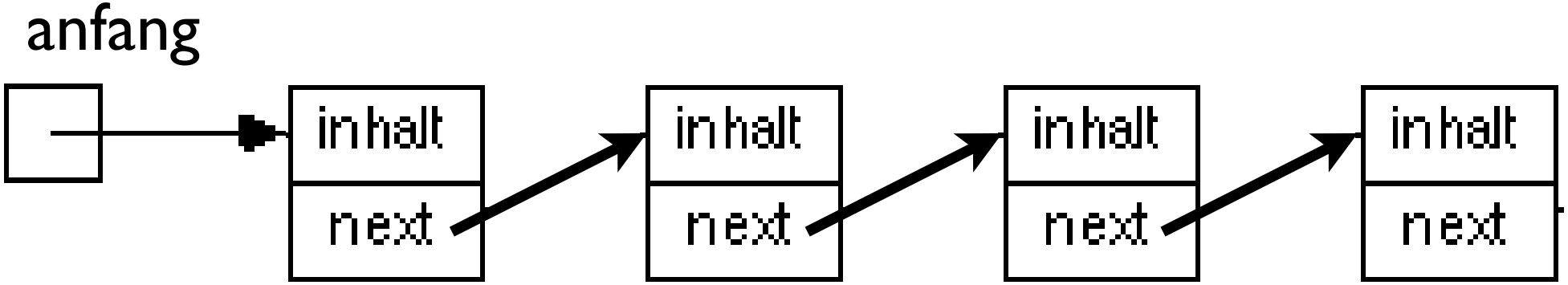
Sequenzen - Einfügen




```
new(q);
q↑.inhalt:=t;
q↑.next:=p↑.next;
p↑.next:=q.
```

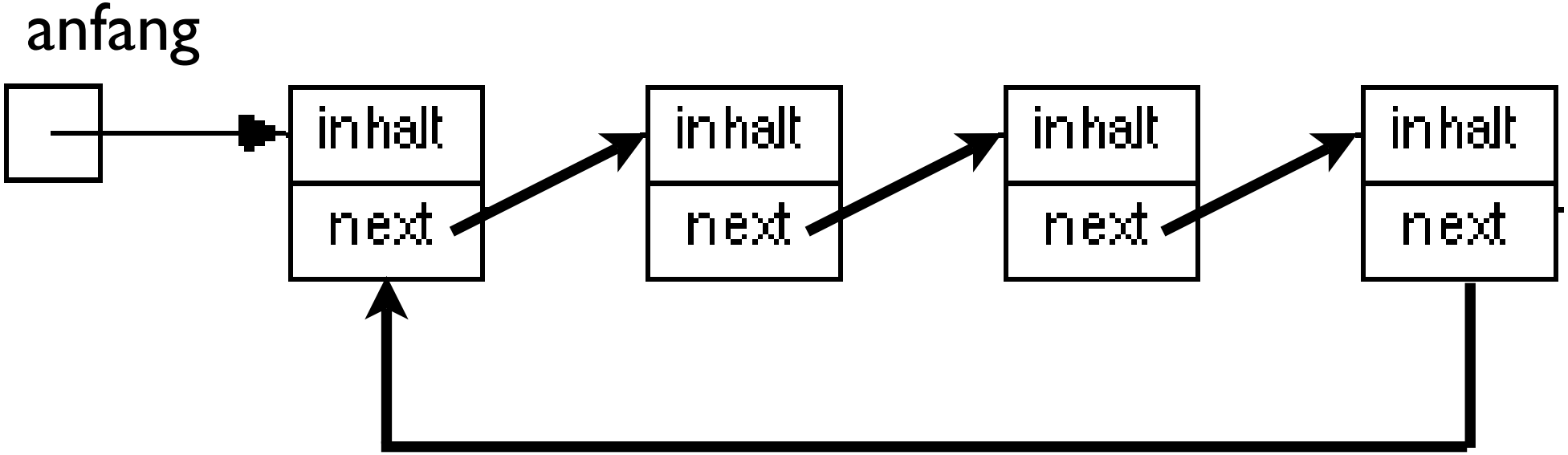
Implementierung von Datent.

Sequenzen - kreisförmige Verkettung



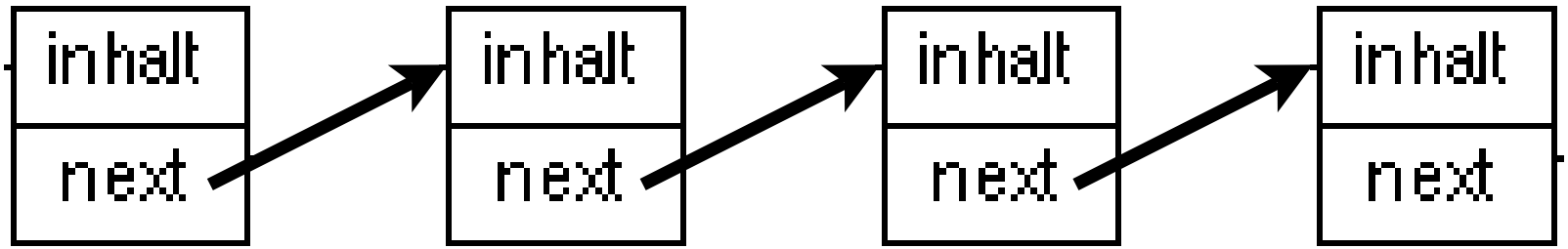
Implementierung von Datent.

Sequenzen - kreisförmige Verkettung



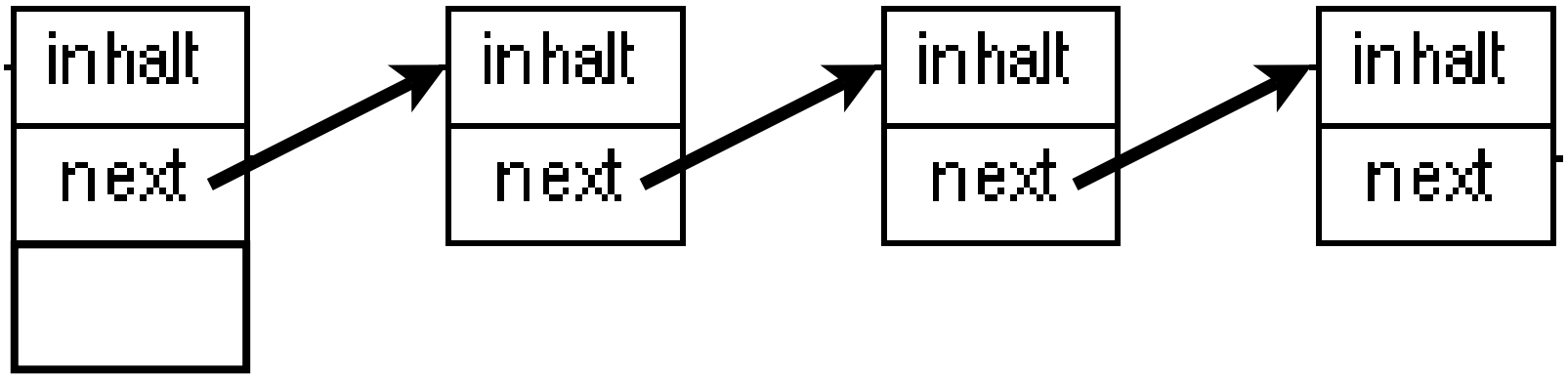
Implementierung von Datent.

Sequenzen - doppelte Verkettung



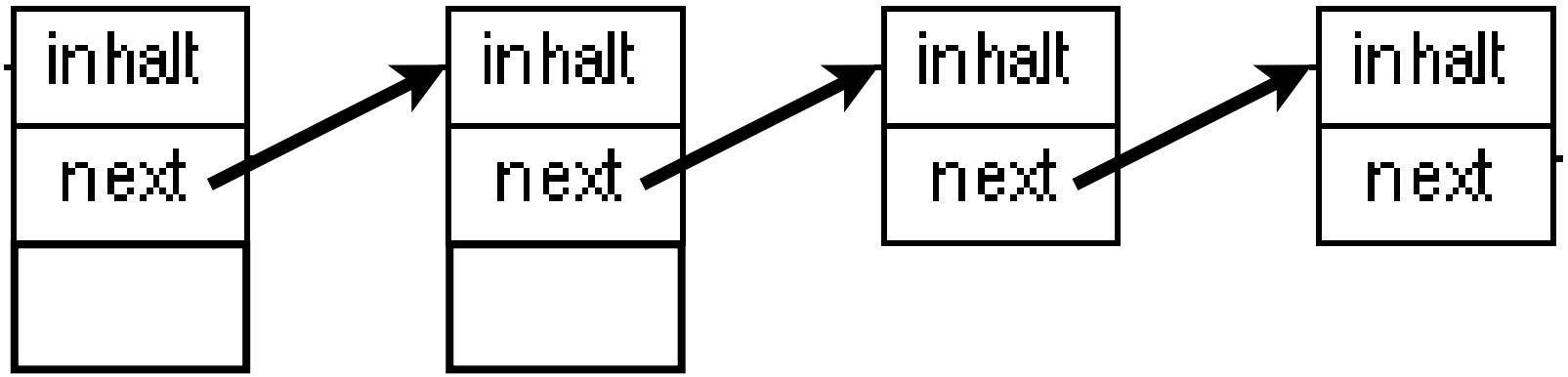
Implementierung von Datent.

Sequenzen - doppelte Verkettung



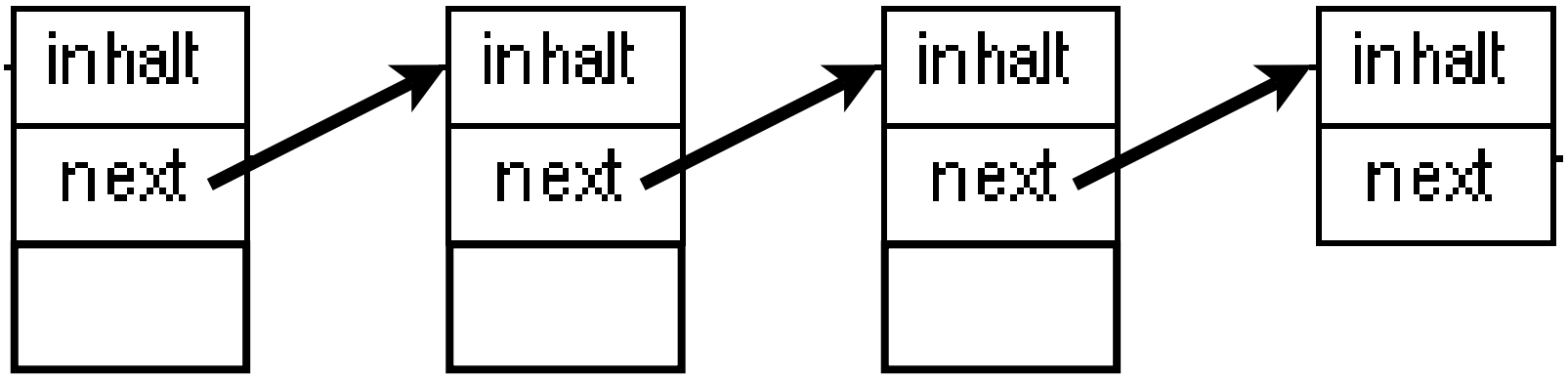
Implementierung von Datent.

Sequenzen - doppelte Verkettung



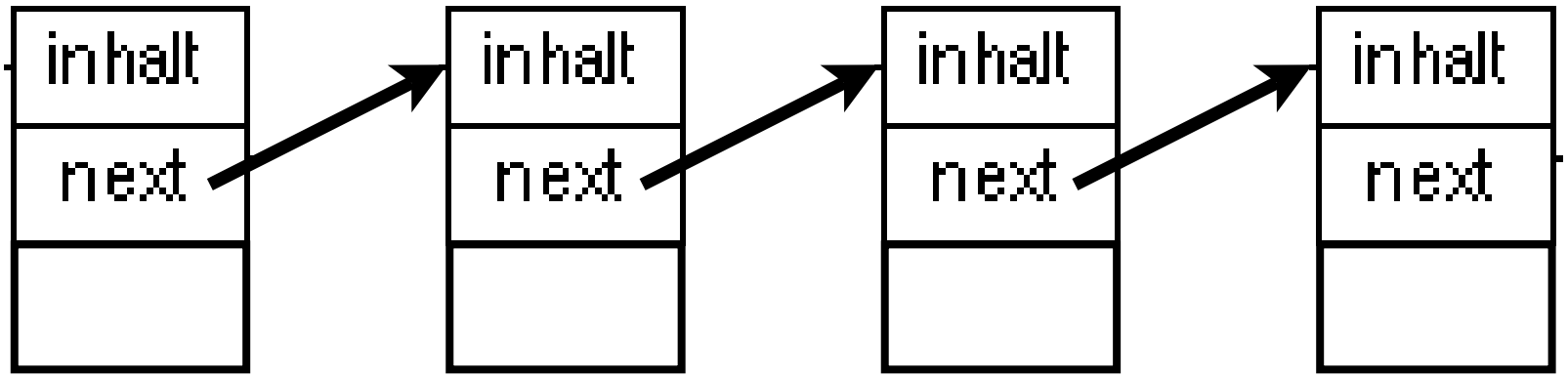
Implementierung von Datent.

Sequenzen - doppelte Verkettung



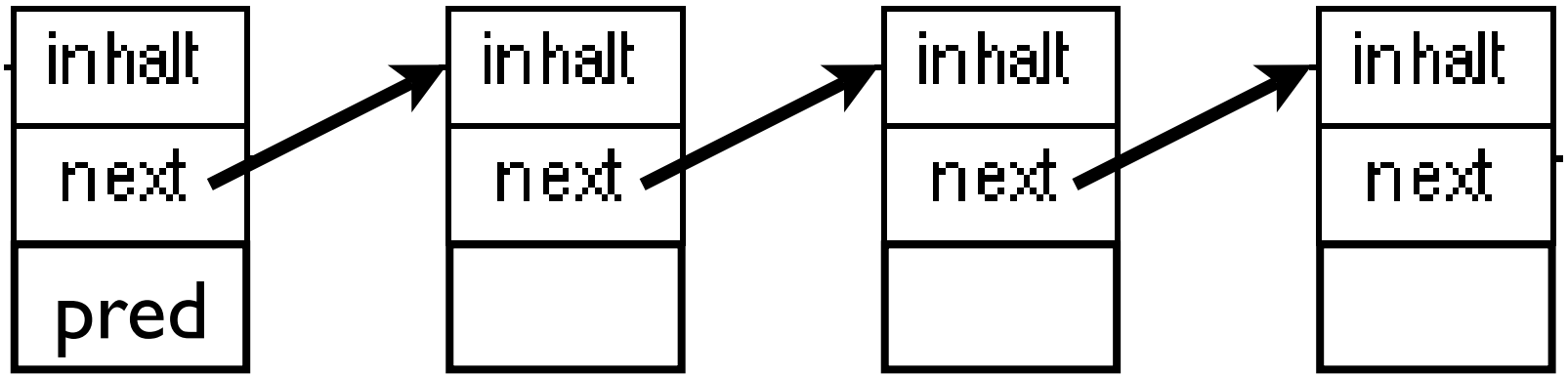
Implementierung von Datent.

Sequenzen - doppelte Verkettung



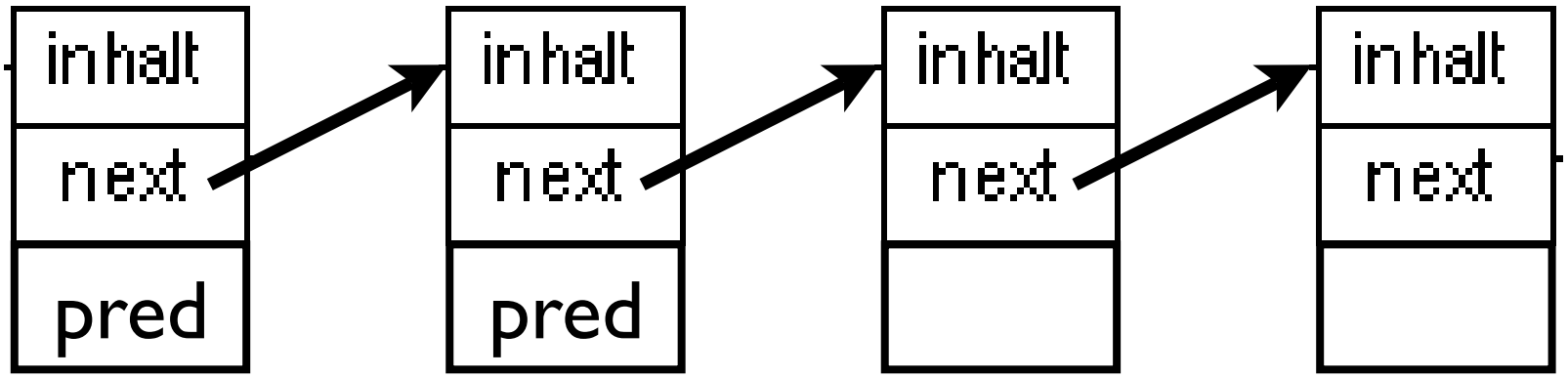
Implementierung von Datent.

Sequenzen - doppelte Verkettung



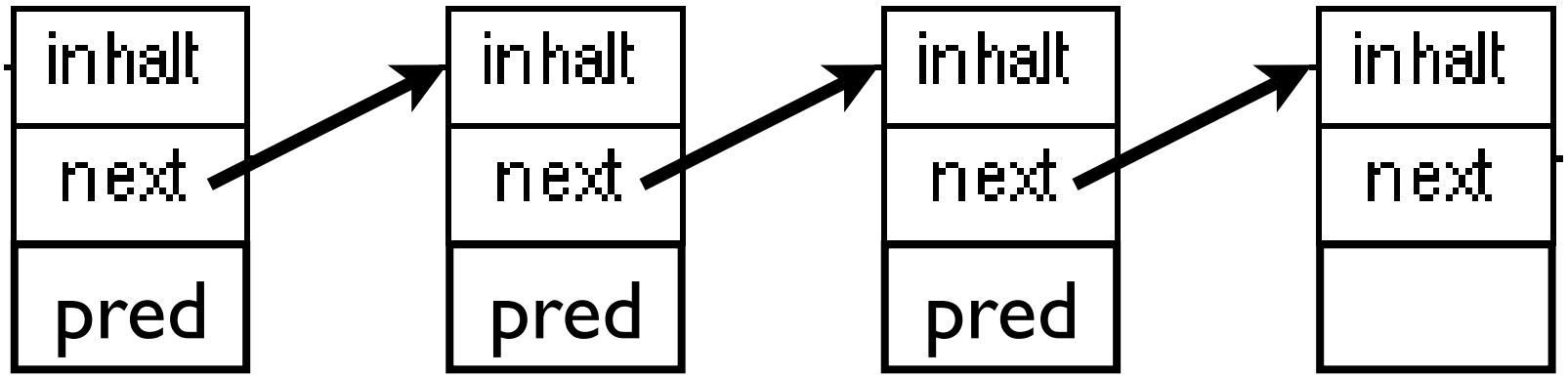
Implementierung von Datent.

Sequenzen - doppelte Verkettung



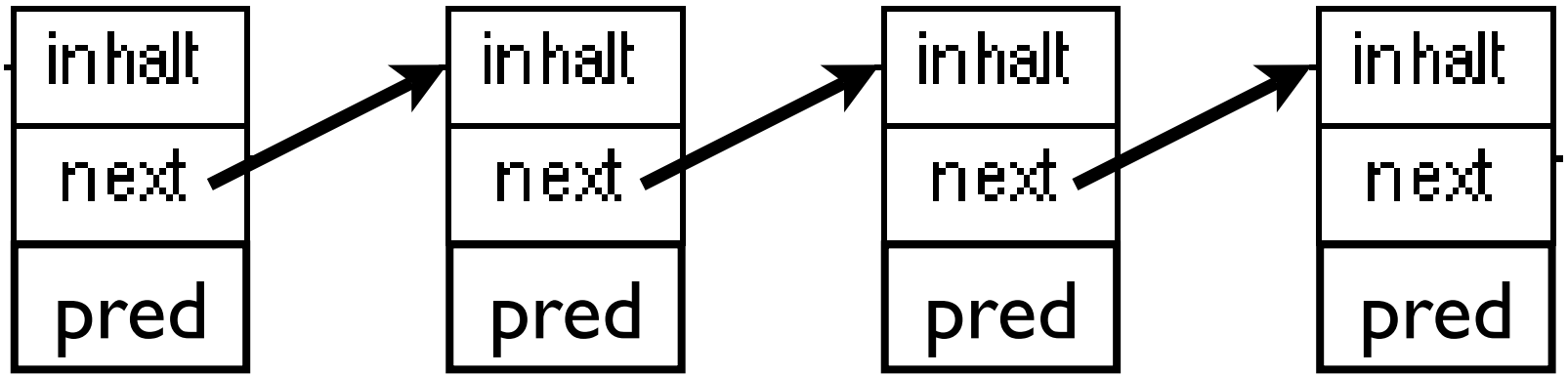
Implementierung von Datent.

Sequenzen - doppelte Verkettung



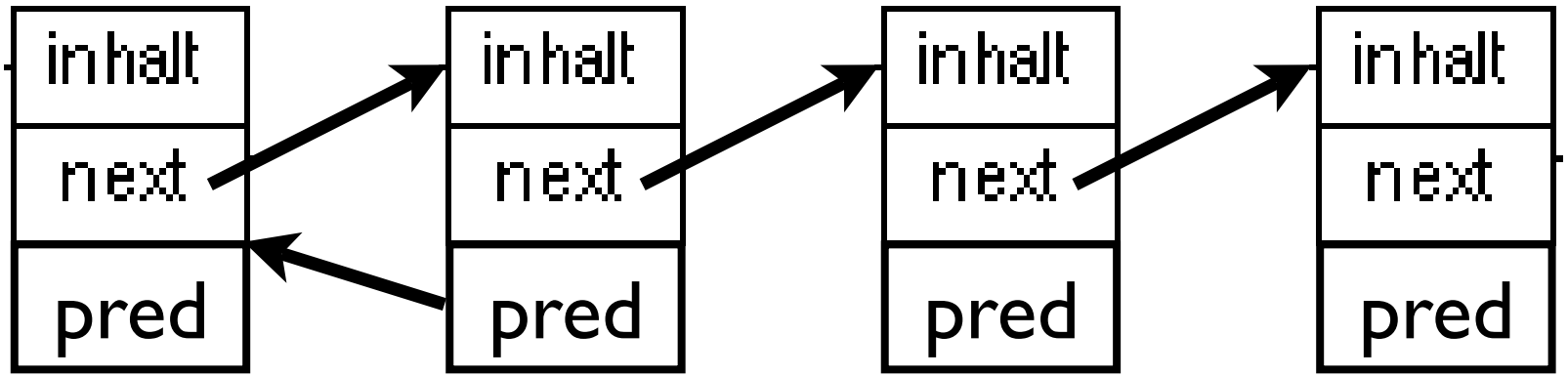
Implementierung von Datent.

Sequenzen - doppelte Verkettung



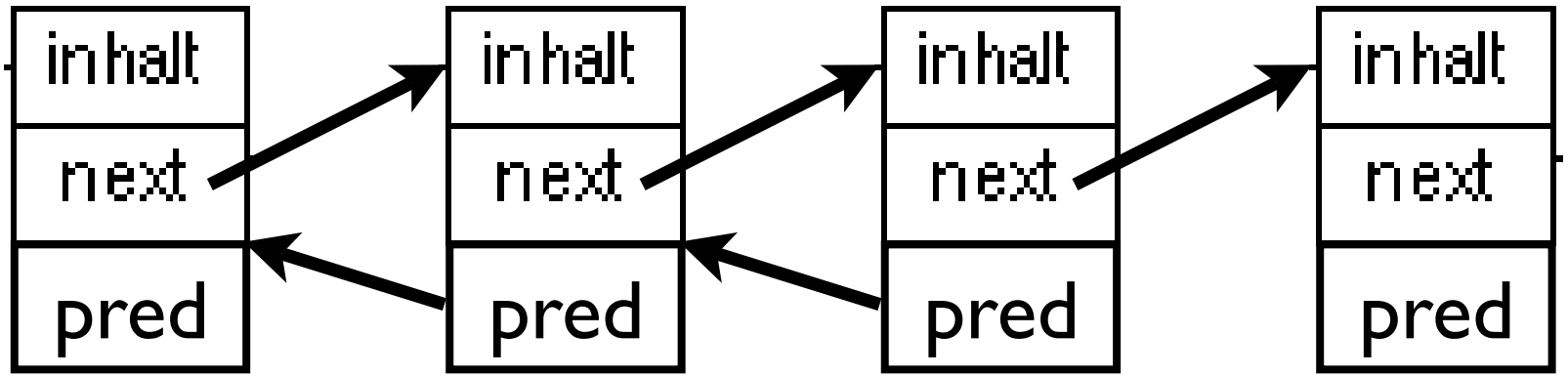
Implementierung von Datent.

Sequenzen - doppelte Verkettung



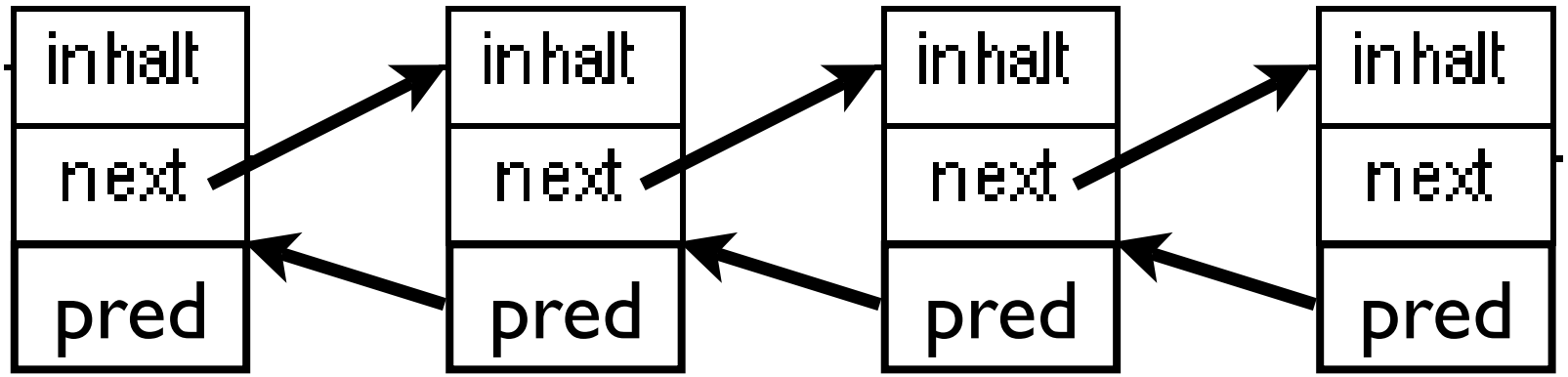
Implementierung von Datent.

Sequenzen - doppelte Verkettung



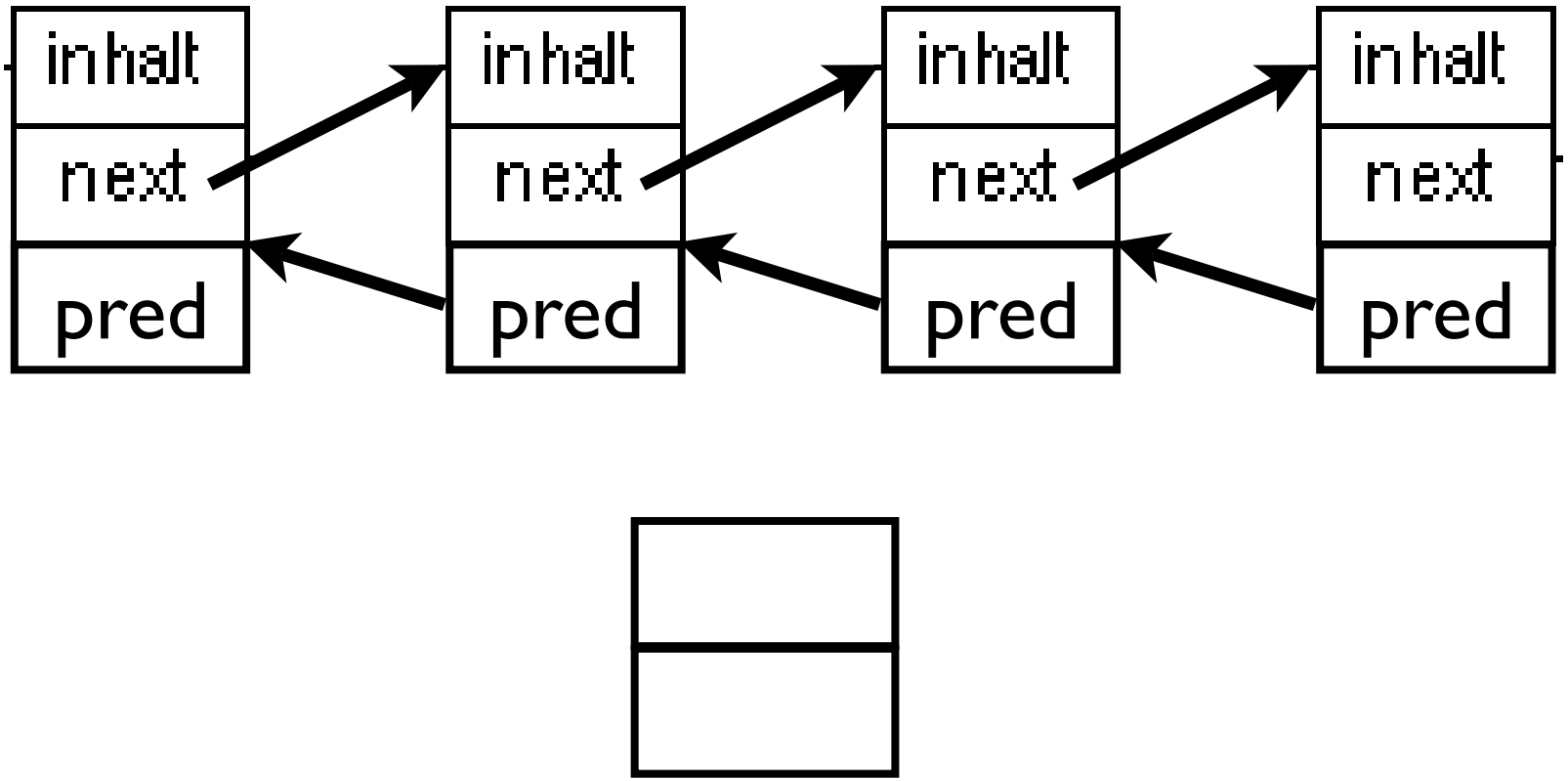
Implementierung von Datent.

Sequenzen - doppelte Verkettung



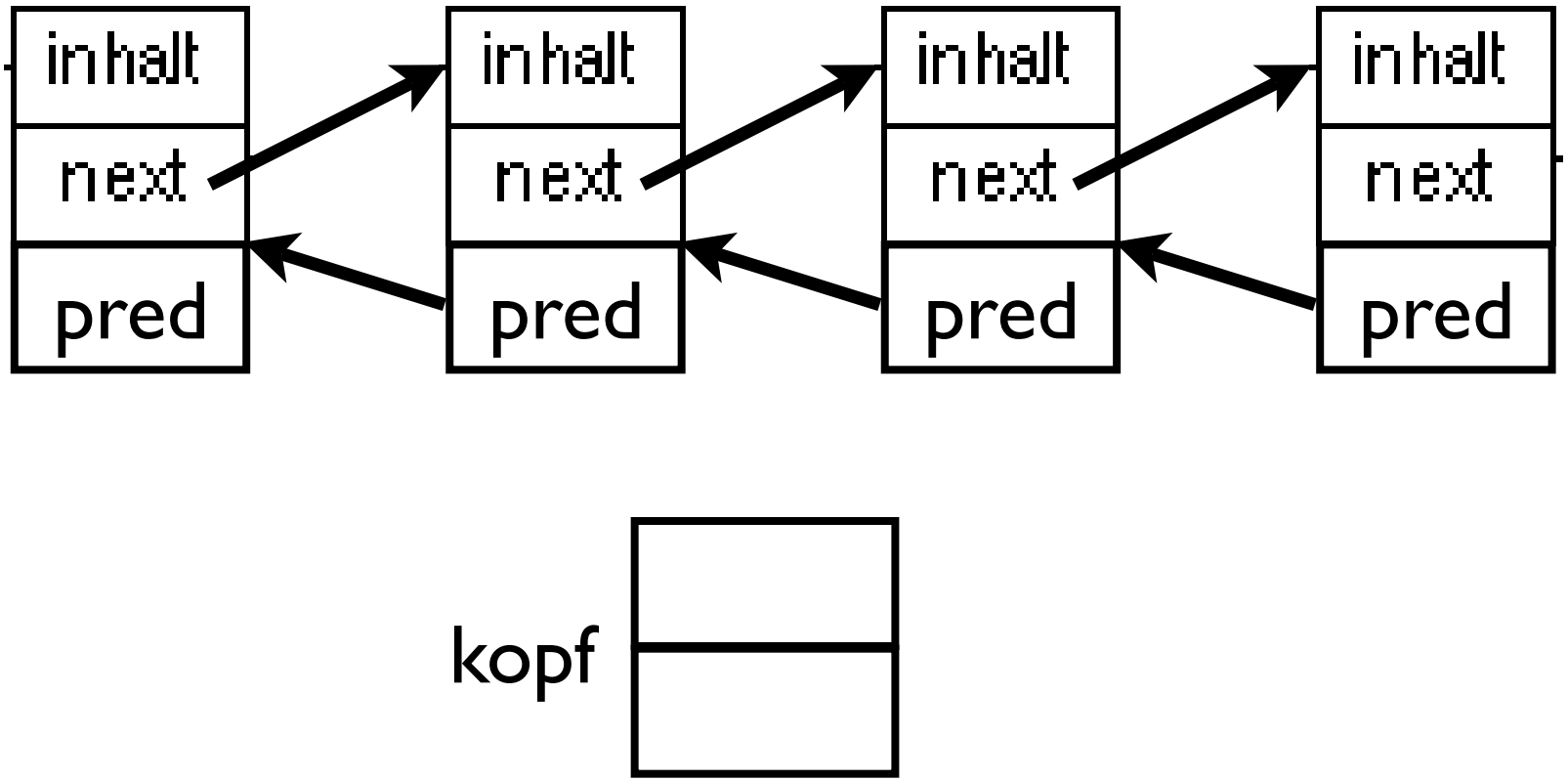
Implementierung von Datent.

Sequenzen - doppelte Verkettung



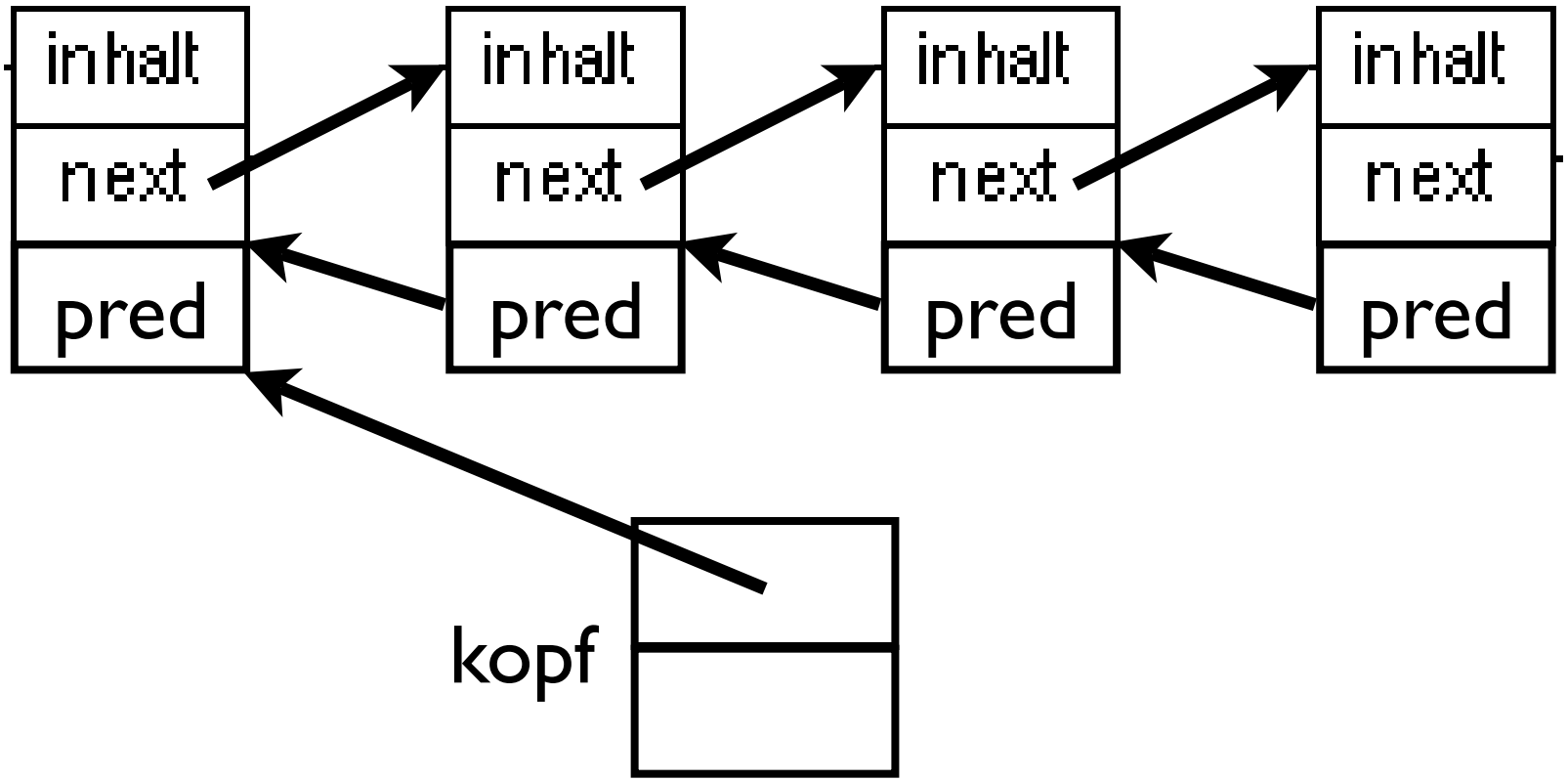
Implementierung von Datent.

Sequenzen - doppelte Verkettung



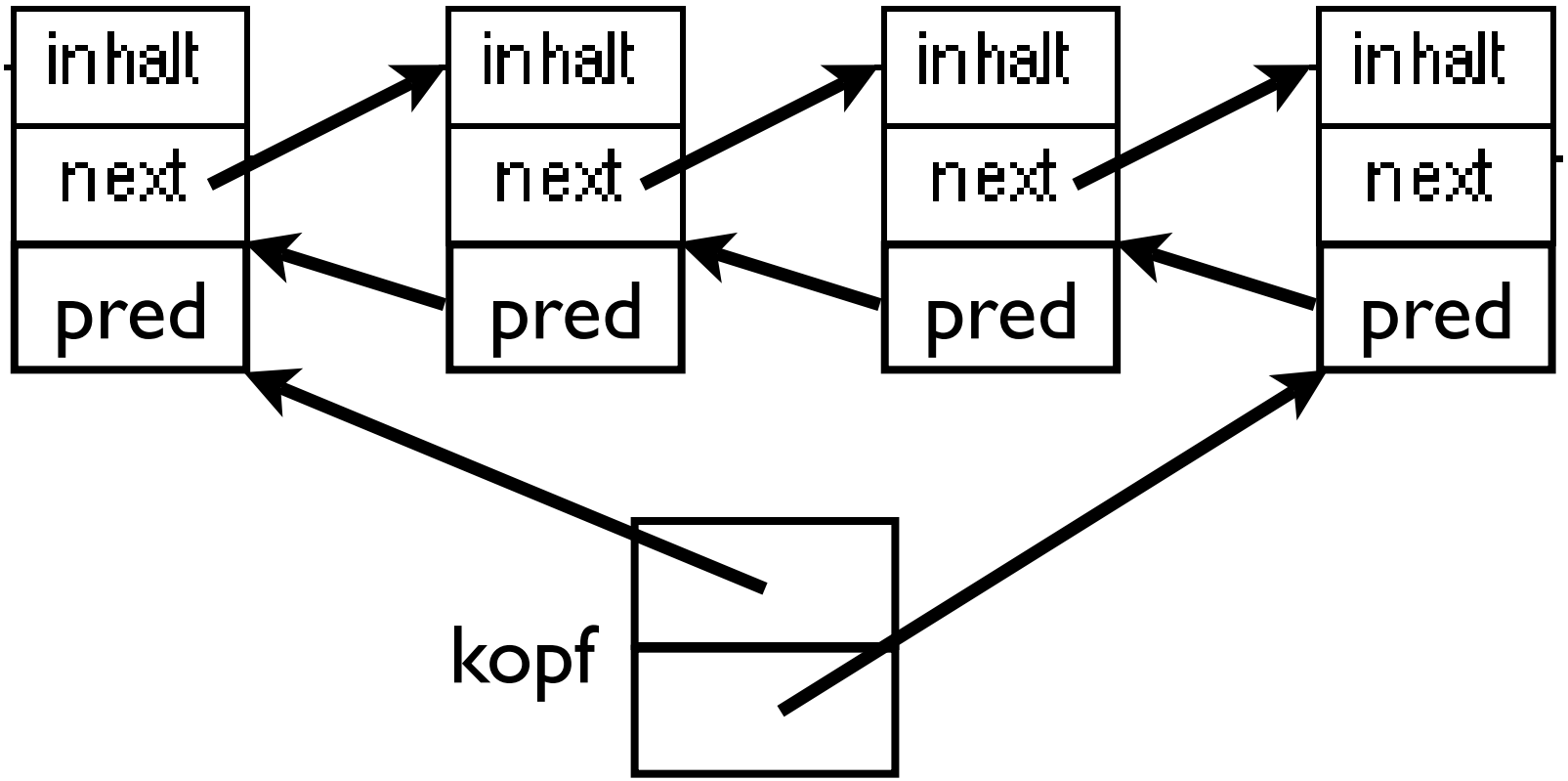
Implementierung von Datent.

Sequenzen - doppelte Verkettung



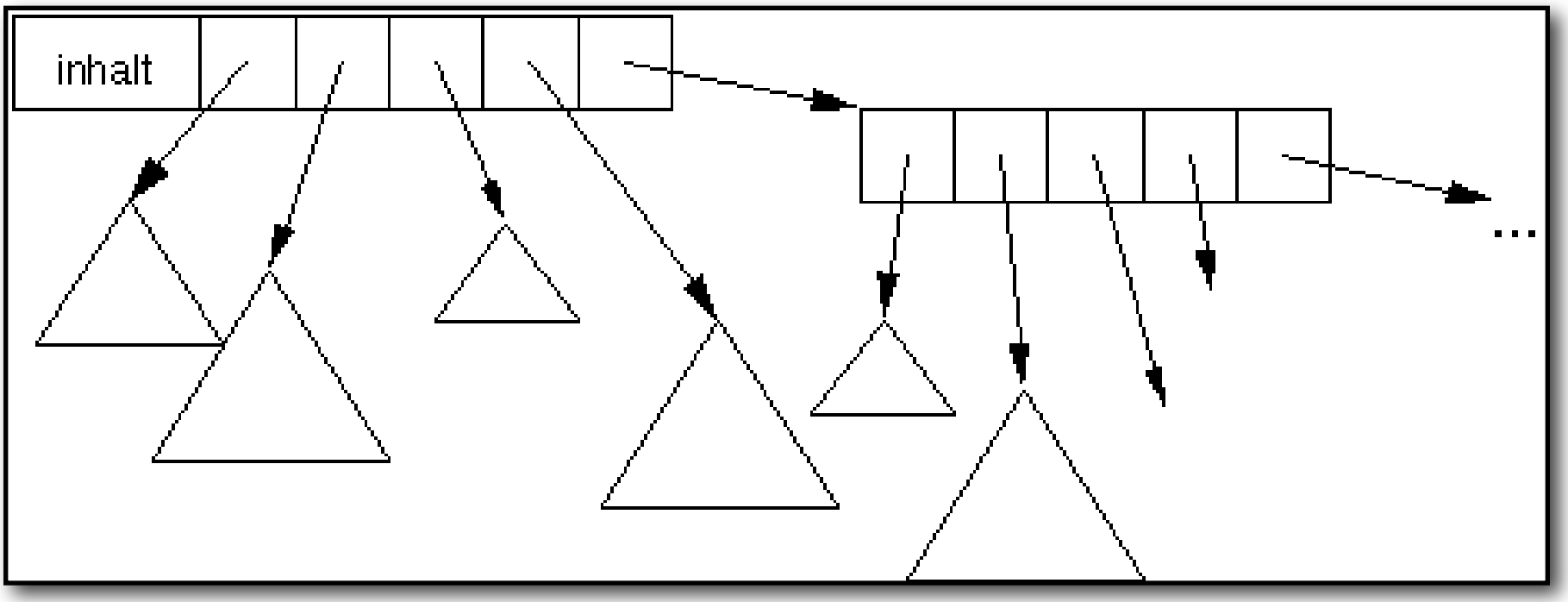
Implementierung von Datent.

Sequenzen - doppelte Verkettung



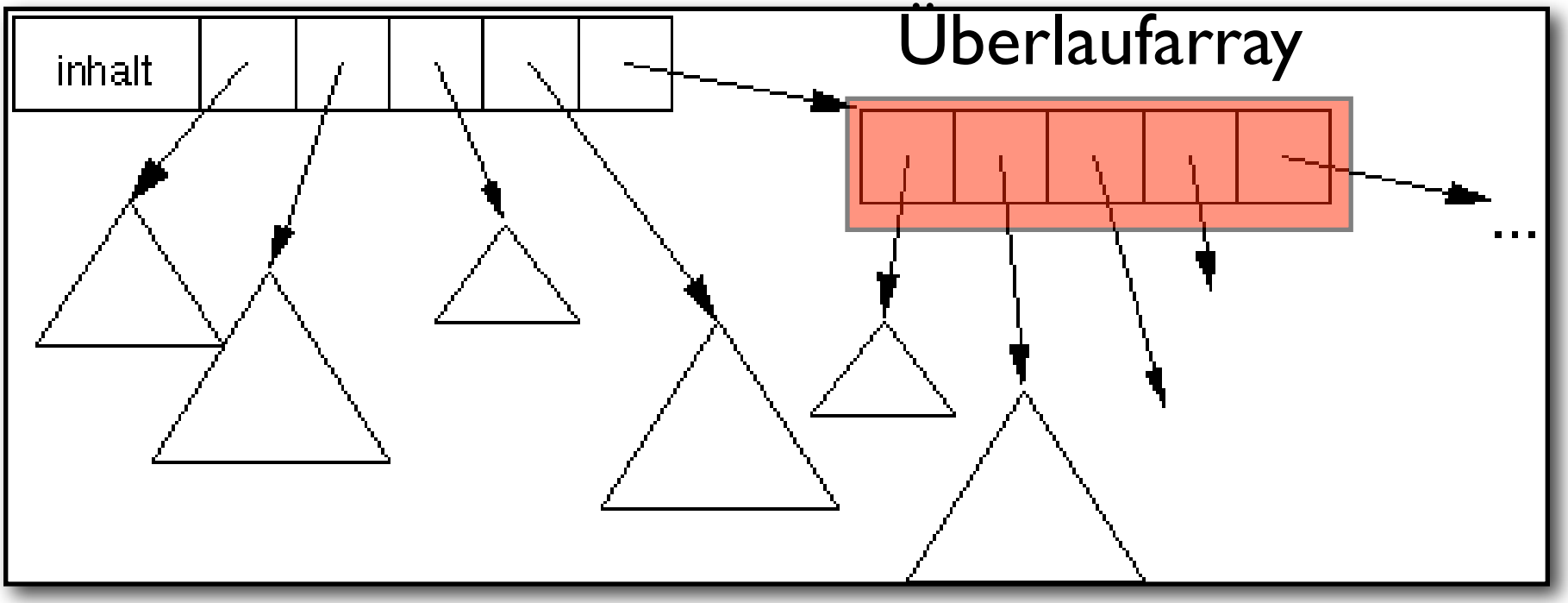
Implementierung von Datent.
Referenzen in Pascal - Bäume

- weitgehend begrenzter Knotengrad:



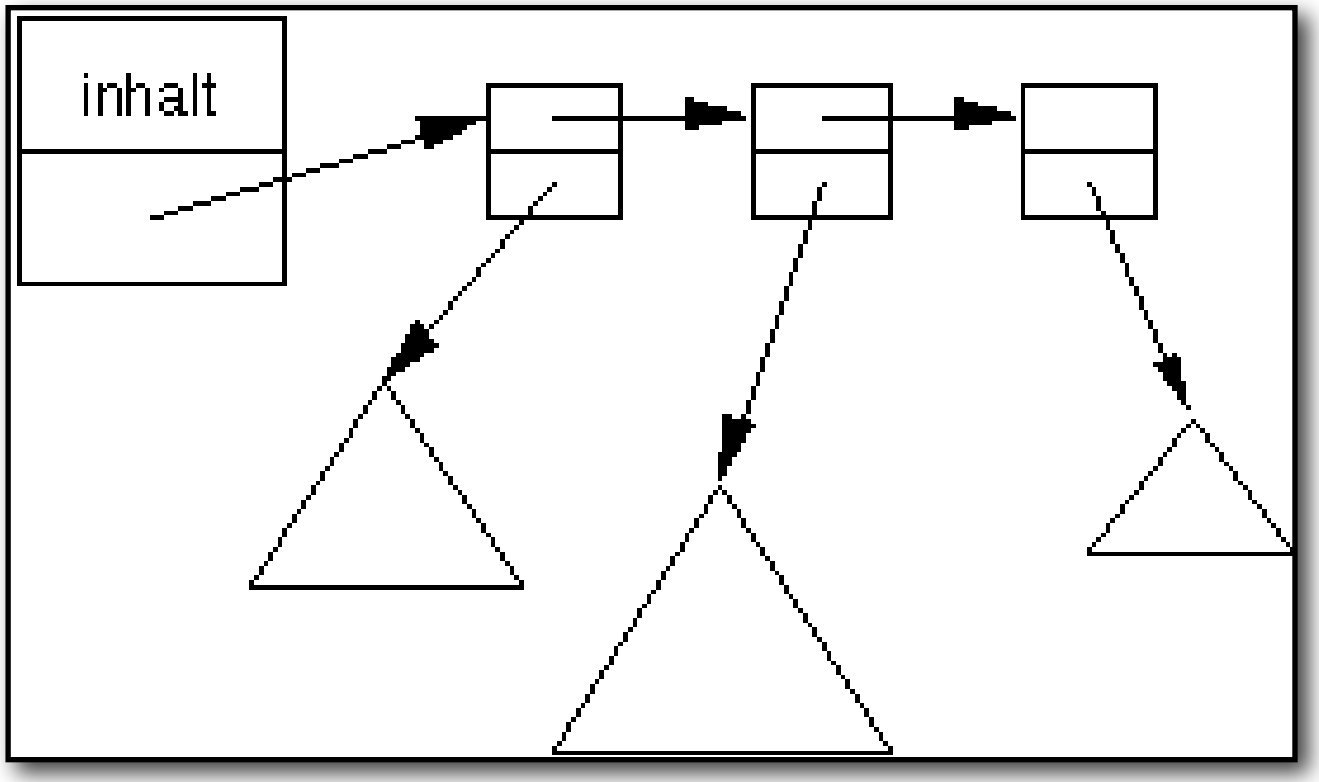
Implementierung von Datent.
Referenzen in Pascal - Bäume

- weitgehend begrenzter Knotengrad:



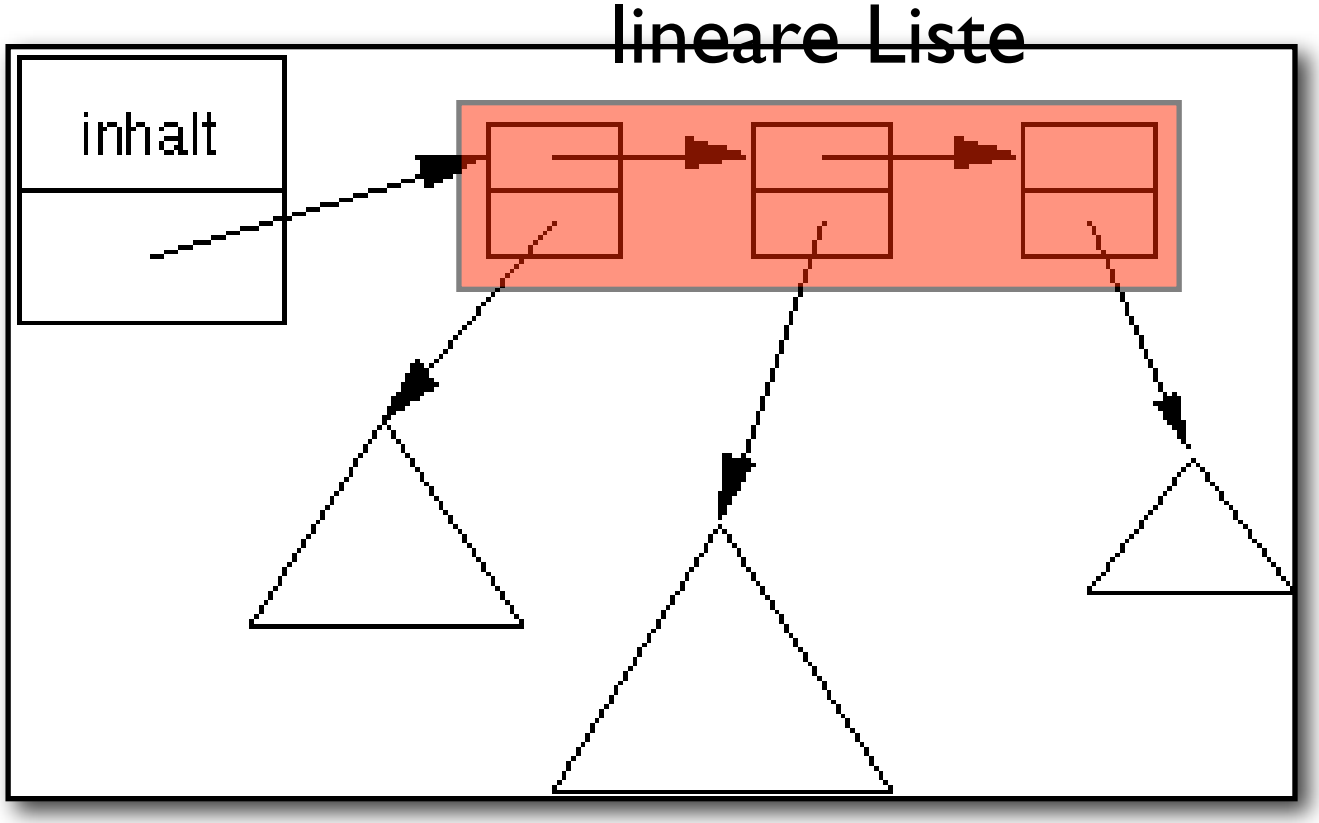
Implementierung von Datent.
Referenzen in Pascal - Bäume

- beliebiger Knotengrad:



Implementierung von Datent.
Referenzen in Pascal - Bäume

- beliebiger Knotengrad:

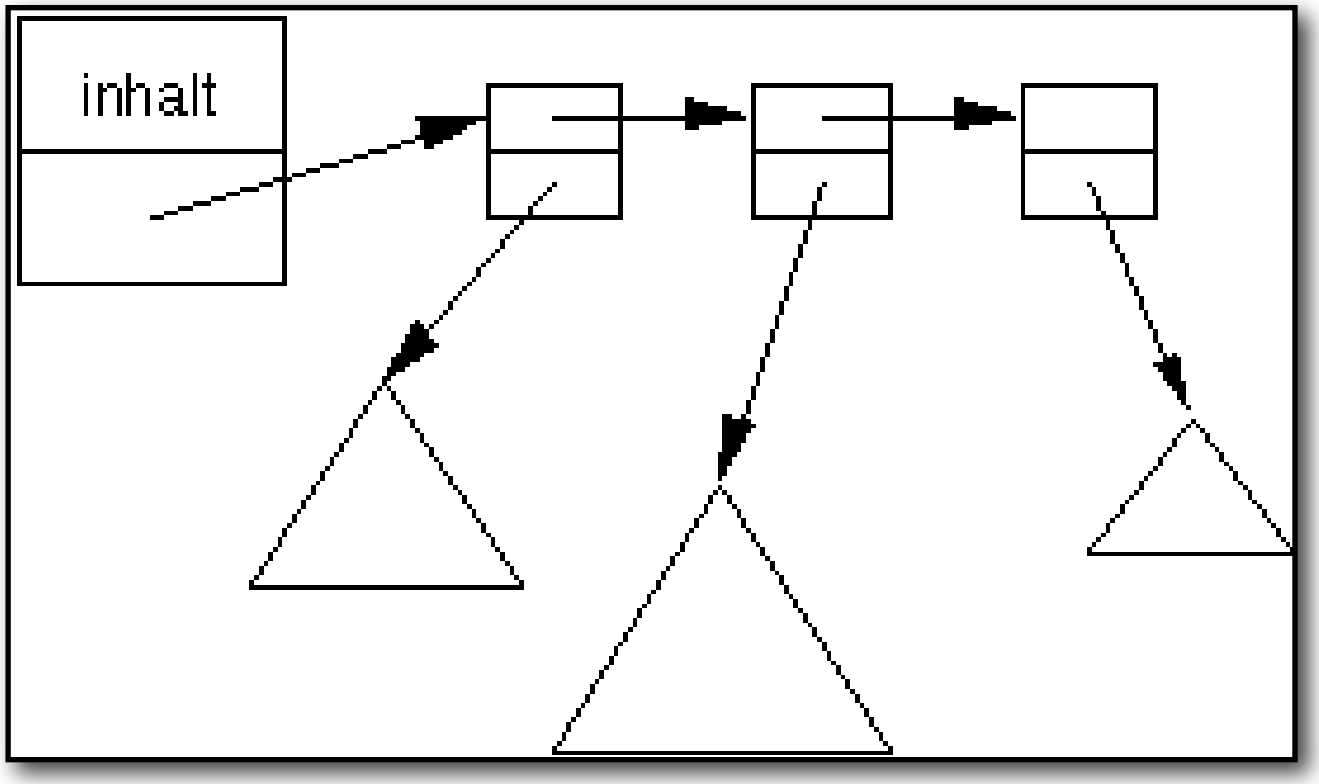


Implementierung von Datent.

Bäume - Speicherausnutz.

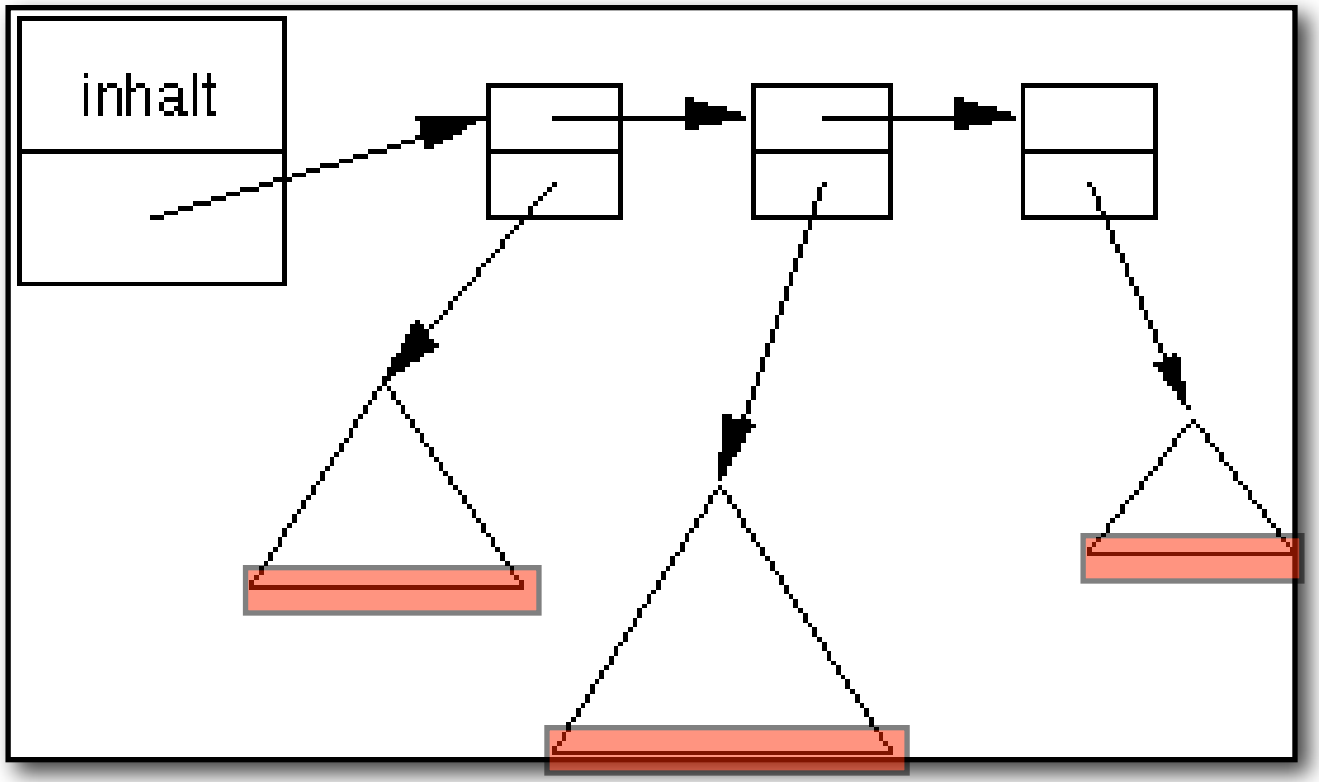
Implementierung von Datent.

Bäume - Speicherausnutz.



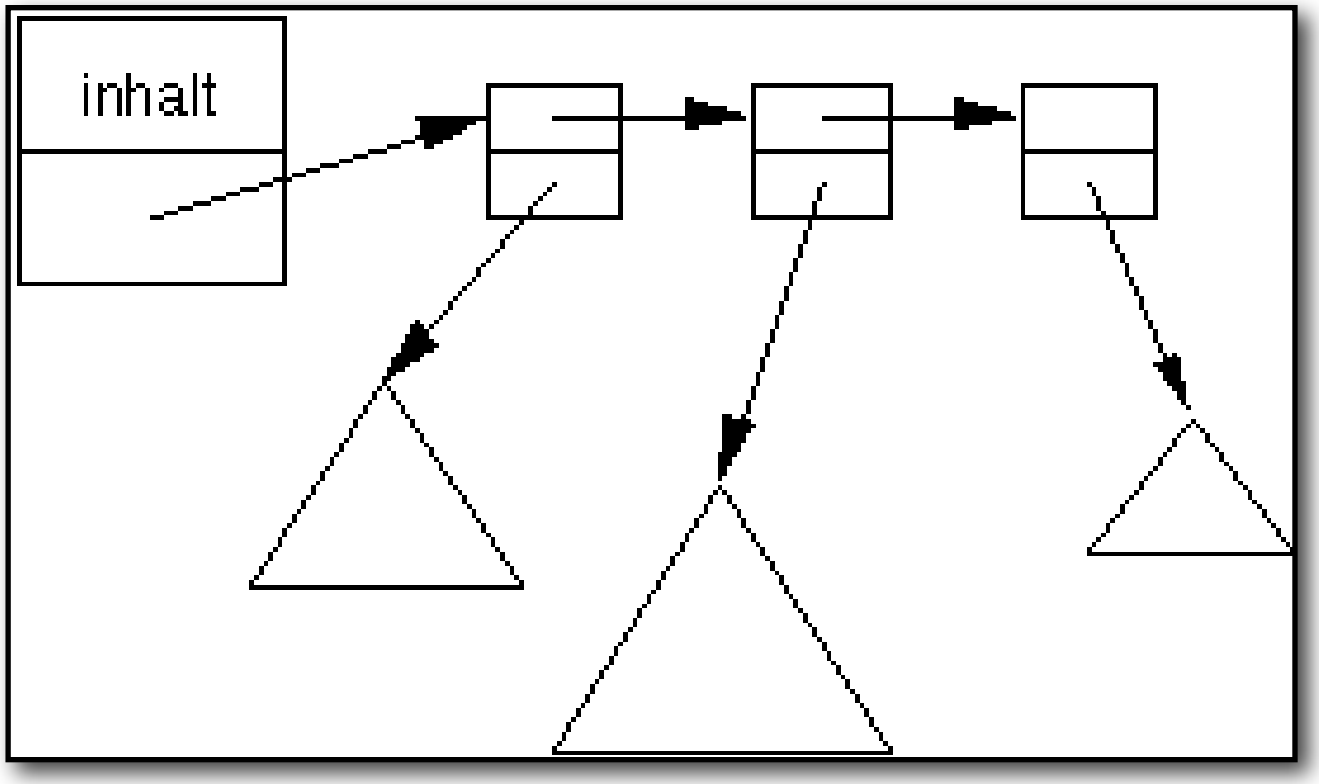
Implementierung von Datent.

Bäume - Speicherausnutz.



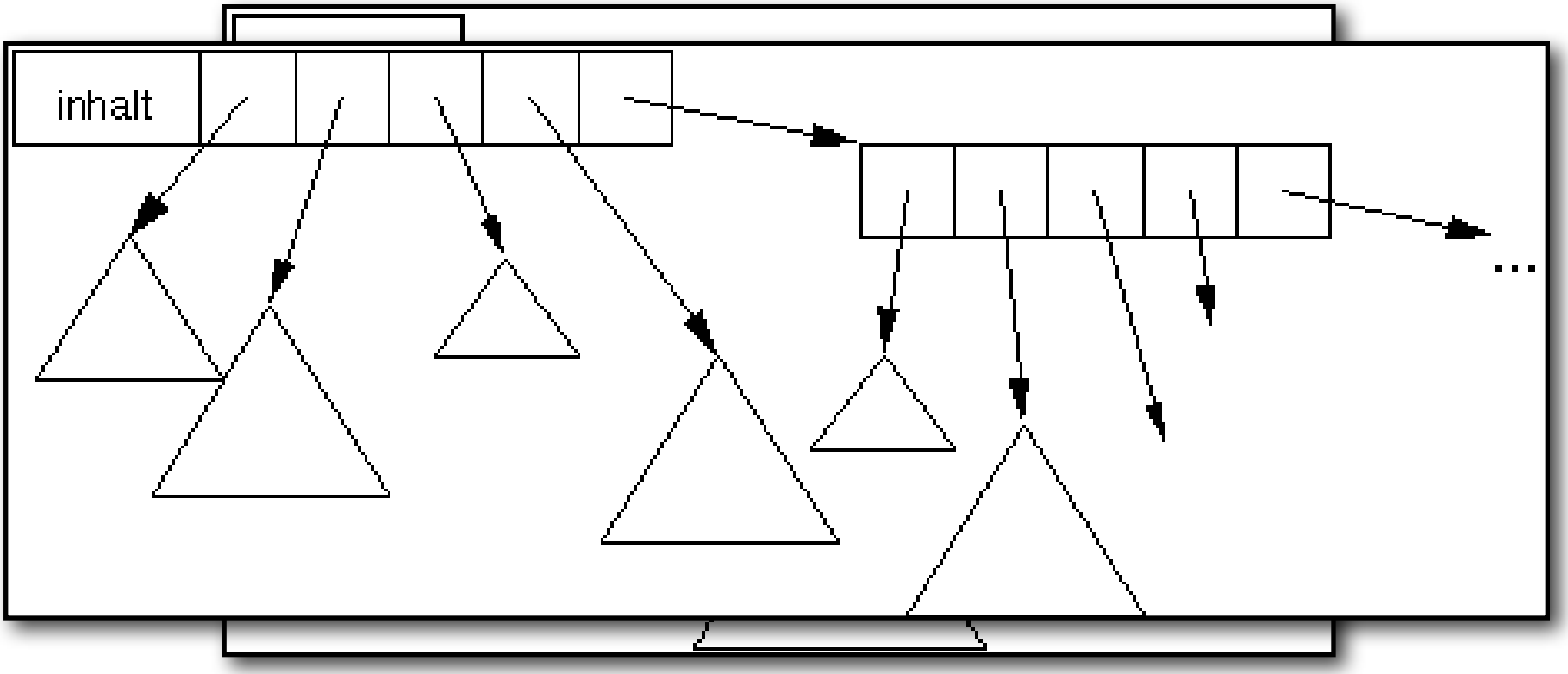
Implementierung von Datent.

Bäume - Speicherausnutz.



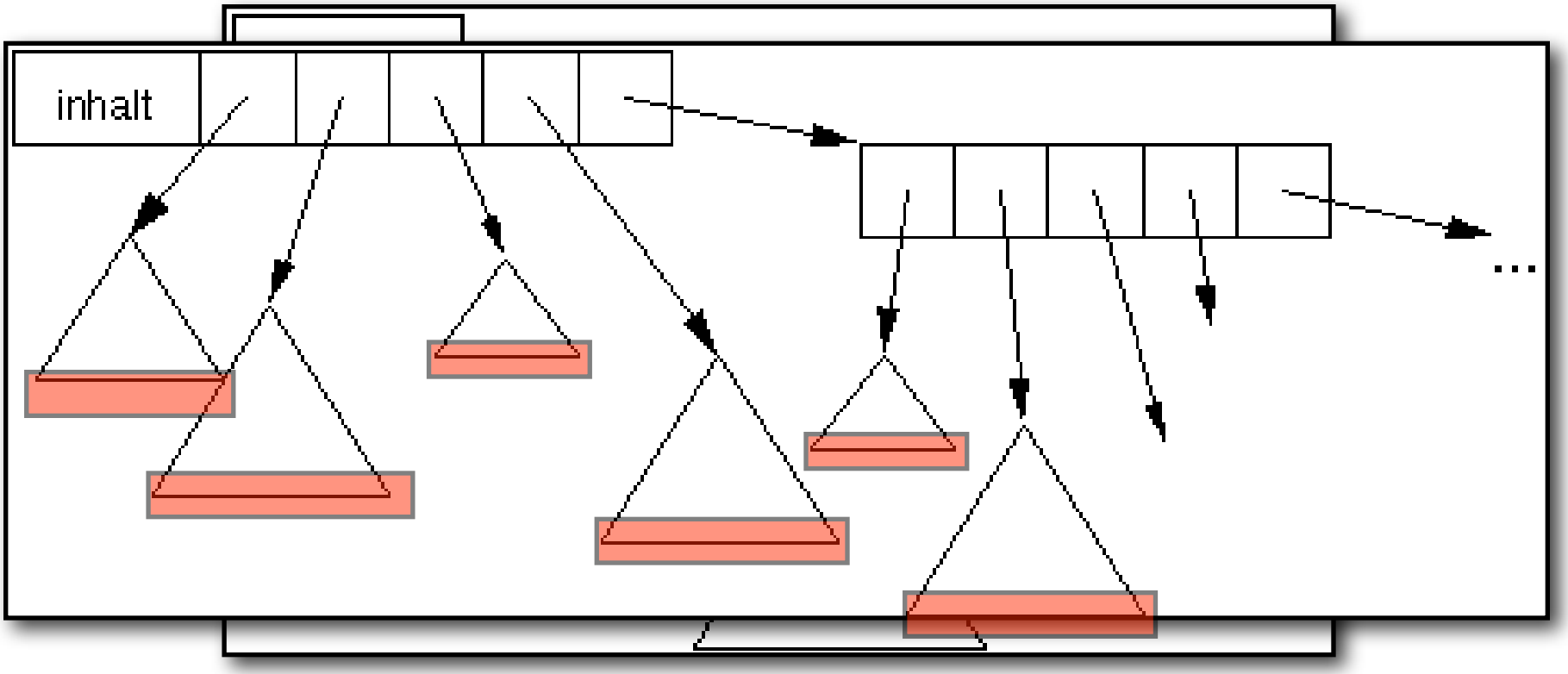
Implementierung von Datent.

Bäume - Speicherausnutz.



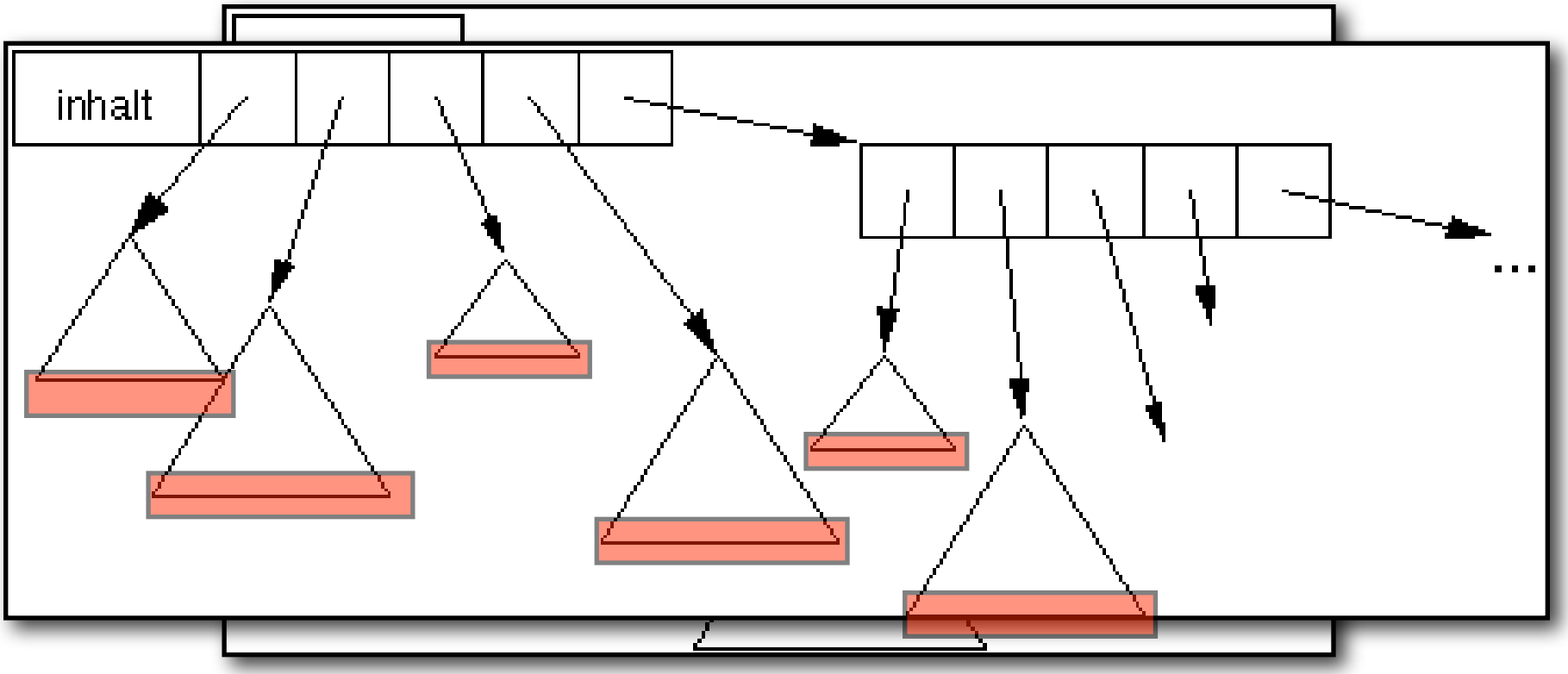
Implementierung von Datent.

Bäume - Speicherausnutz.



Implementierung von Datent.

Bäume - Speicherausnutz.



Viele Zeiger in den Blättern ungenutzt

Implementierung von Datent.
Bäume - Speicherausnutz.

- Genauer:
  - Gegeben: Baum B mit n Knoten
  - je Knoten (außer Blätter) k Söhne
  - B hat also nk Zeiger
  - Wegen n Knoten genau n-1 Zeiger ≠nil.
  - Folglich nk-(n-1)=n(k-1)+1 Zeiger =nil und damit praktisch überflüssig
  - Beispiel: ternäre Bäume (k=3): ca. 2/3 der Zeiger überflüssig
  - binäre Bäume (k=2): bestmögliche Speicherausnutzung: ca. 1/2 der Zeiger=nil

Implementierung von Datent.
Bäume - Implementierung

- binäre Bäume:

```
type data=...
        knoten=record
                inhalt: data;
                lt, rt: ↑knoten
        end.
```

- ein Baum:

```
var b: ↑knoten
```

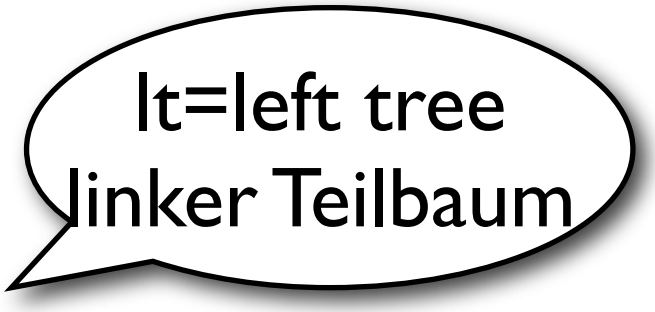
- leerer Baum: b=nil

Implementierung von Datent.

Bäume - Implementierung

- binäre Bäume:

```
type data=...
        knoten=record
                inhalt: data;
                lt, rt: ↑knoten
        end.
```



- ein Baum:

```
var b: ↑knoten
```

- leerer Baum: b=nil

Implementierung von Datent.
Bäume - Implementierung

- binäre Bäume:

```
type data=...
        knoten=record
                inhalt: data;
                lt, rt: ↑knoten
        end.
```

- ein Baum:

```
var b: ↑knoten
```

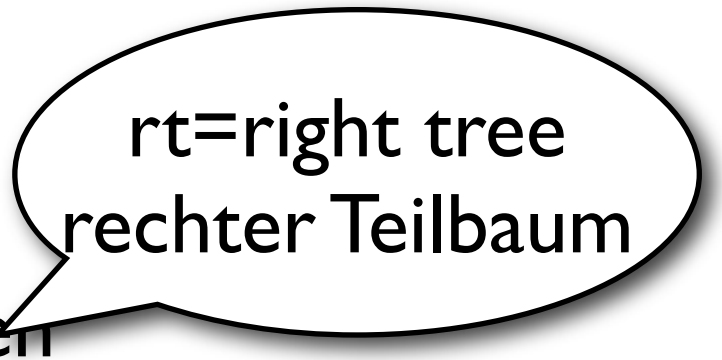
- leerer Baum: b=nil

Implementierung von Datent.

Bäume - Implementierung

- binäre Bäume:

```
type data=...
        knoten=record
                inhalt: data;
                lt, rt: ↑knoten
        end.
```



- ein Baum:

```
var b: ↑knoten
```

- leerer Baum: b=nil

Implementierung von Datent.
Bäume - Implementierung

- binäre Bäume:

```
type data=...
        knoten=record
                inhalt: data;
                lt, rt: ↑knoten
        end.
```

- ein Baum:

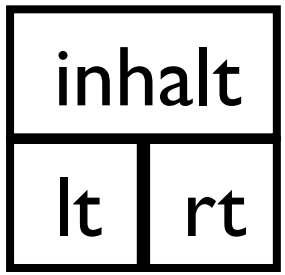
```
var b: ↑knoten
```

- leerer Baum: b=nil

Implementierung von Datent.
Bäume - Implementierung

- binäre Bäume:

```
type data=...
        knoten=record
                inhalt: data;
                lt, rt: ↑knoten
        end.
```



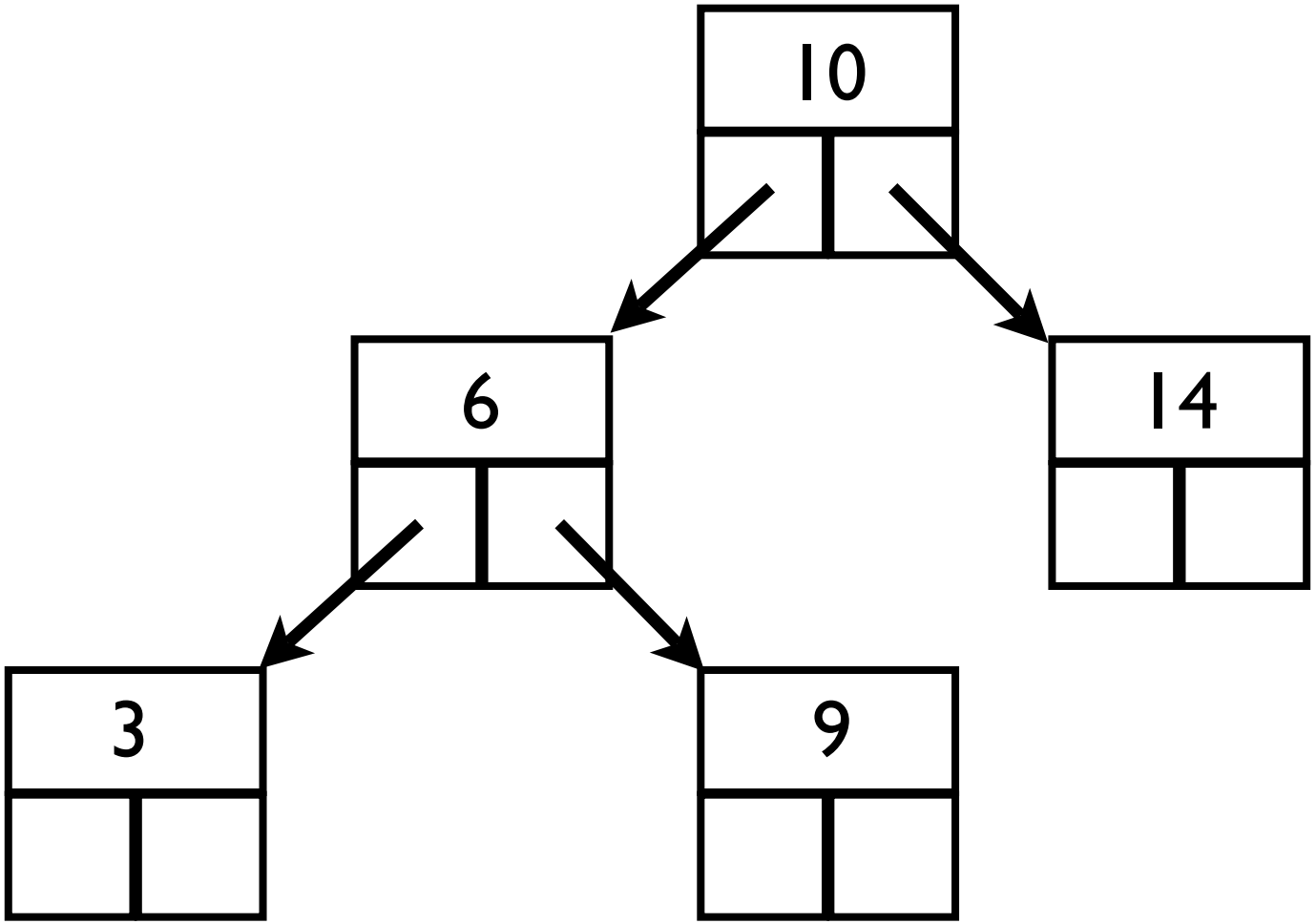
- ein Baum:

```
var b: ↑knoten
```

- leerer Baum: b=nil

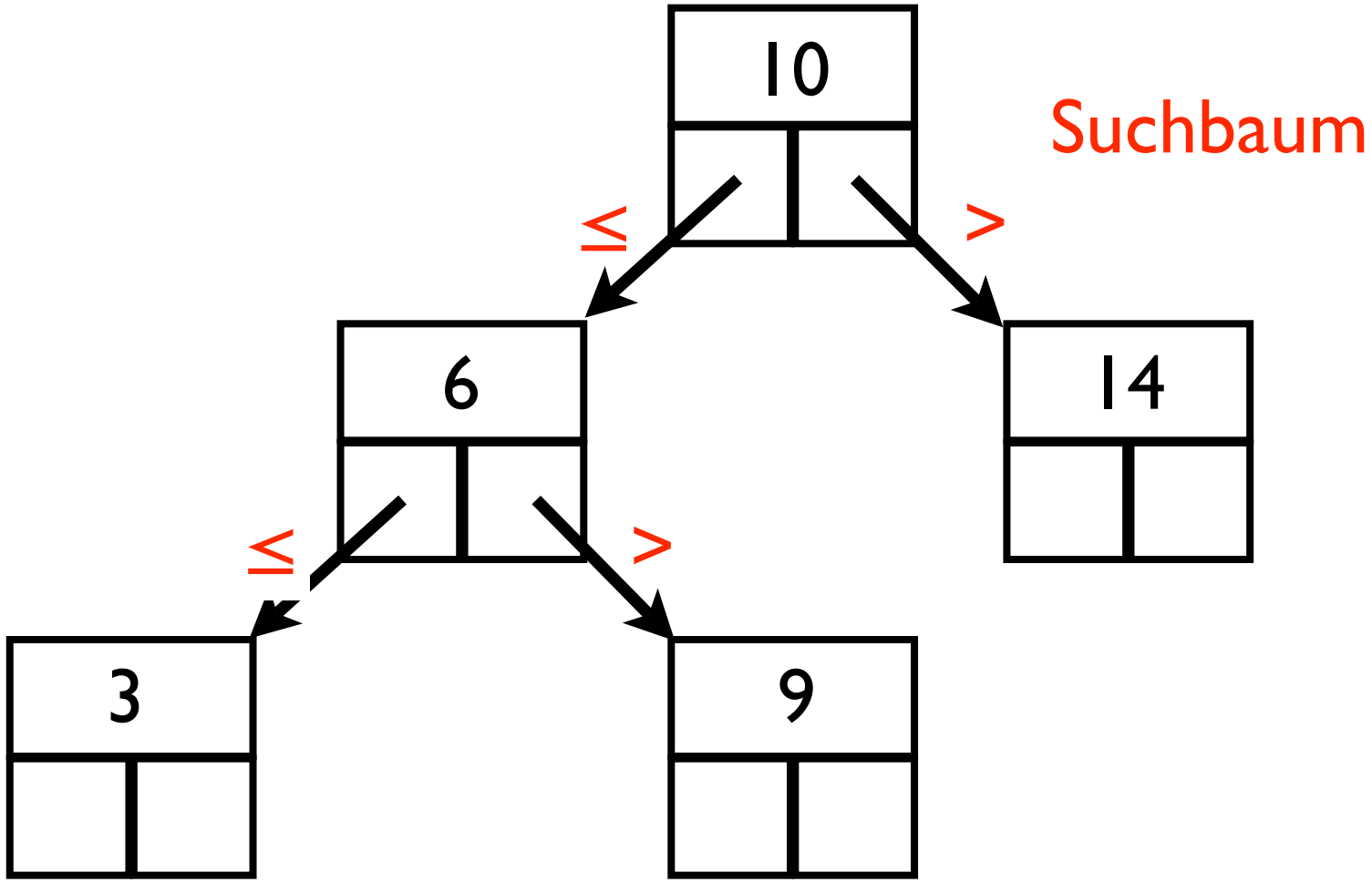
Implementierung von Datent.

Bäume - Implementierung - Beispiel



Implementierung von Datent.

Bäume - Implementierung - Beispiel



Implementierung von Datent.
Bäume - Baumdurchlauf

inorder-Durchlauf:

- besuche linken Teilbaum inorder
- gib Wurzel aus
- besuche rechten Teilbaum inorder

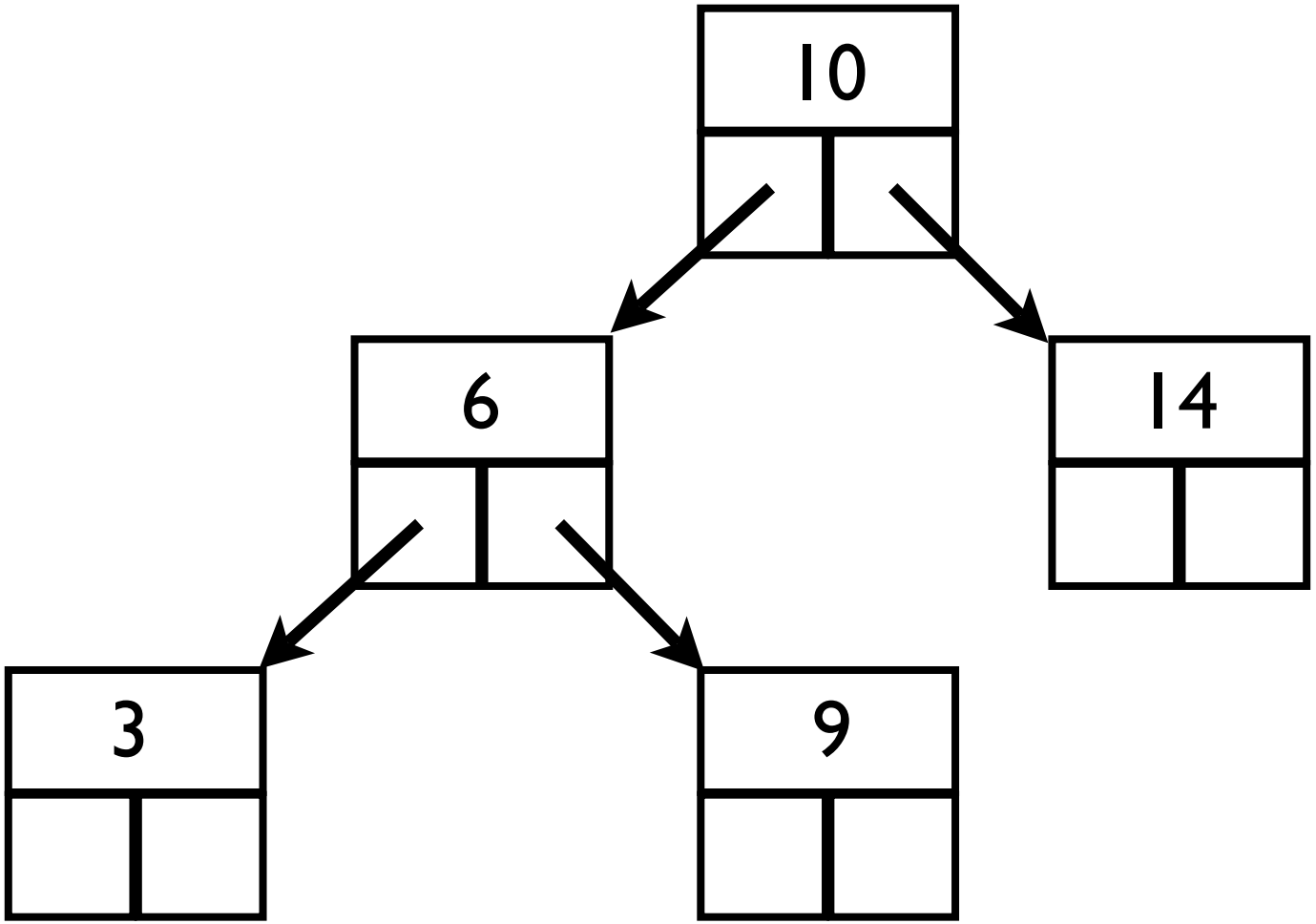
Implementierung von Datent.

Bäume - Algorithmus inorder

```
procedure inorder (b: ↑knoten);
begin
   if b<>nil then
   begin
      inorder(b↑.lt);
      write(b↑.inhalt);
      inorder(b↑.rt)
   end
end.
```

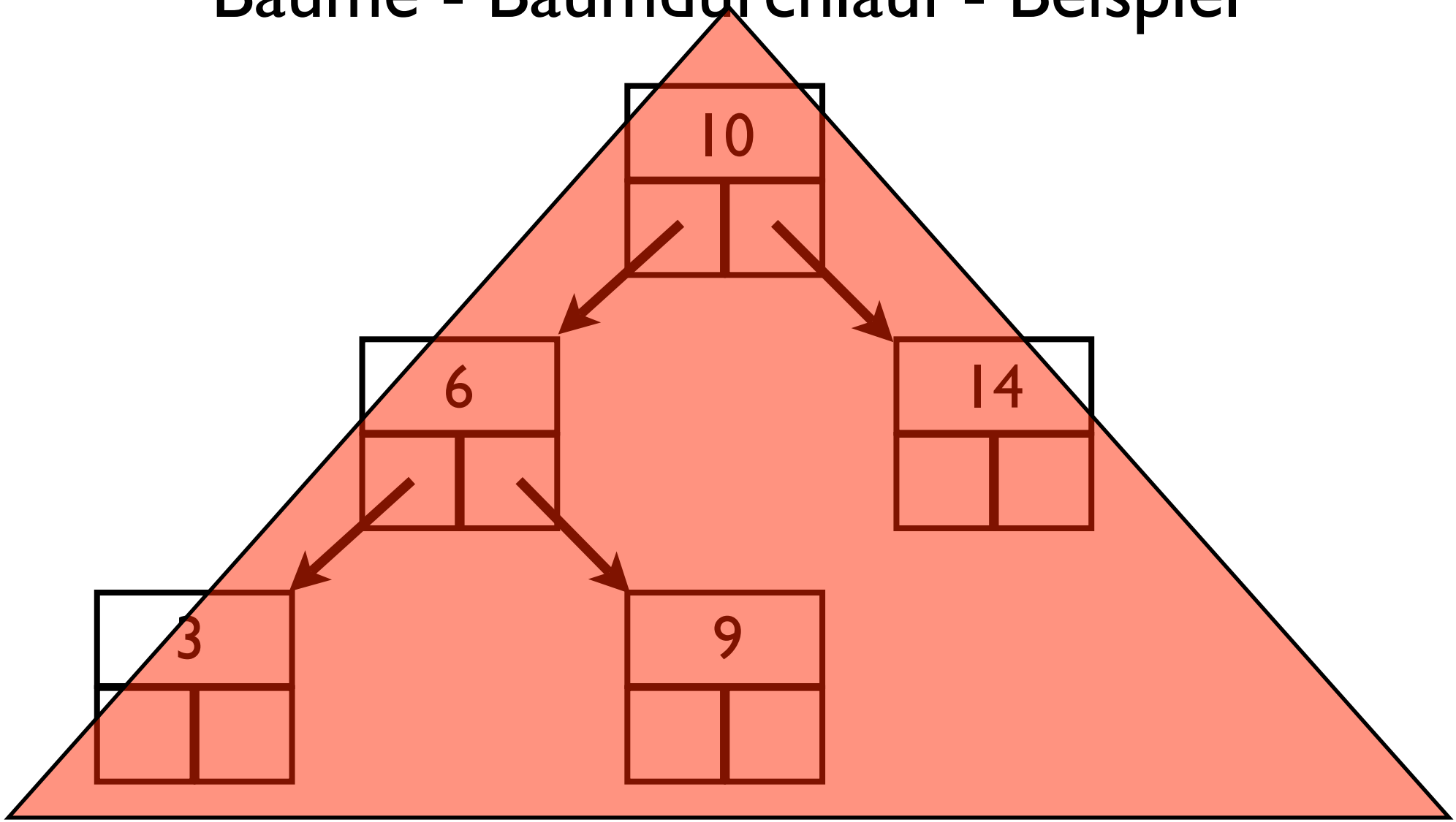
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



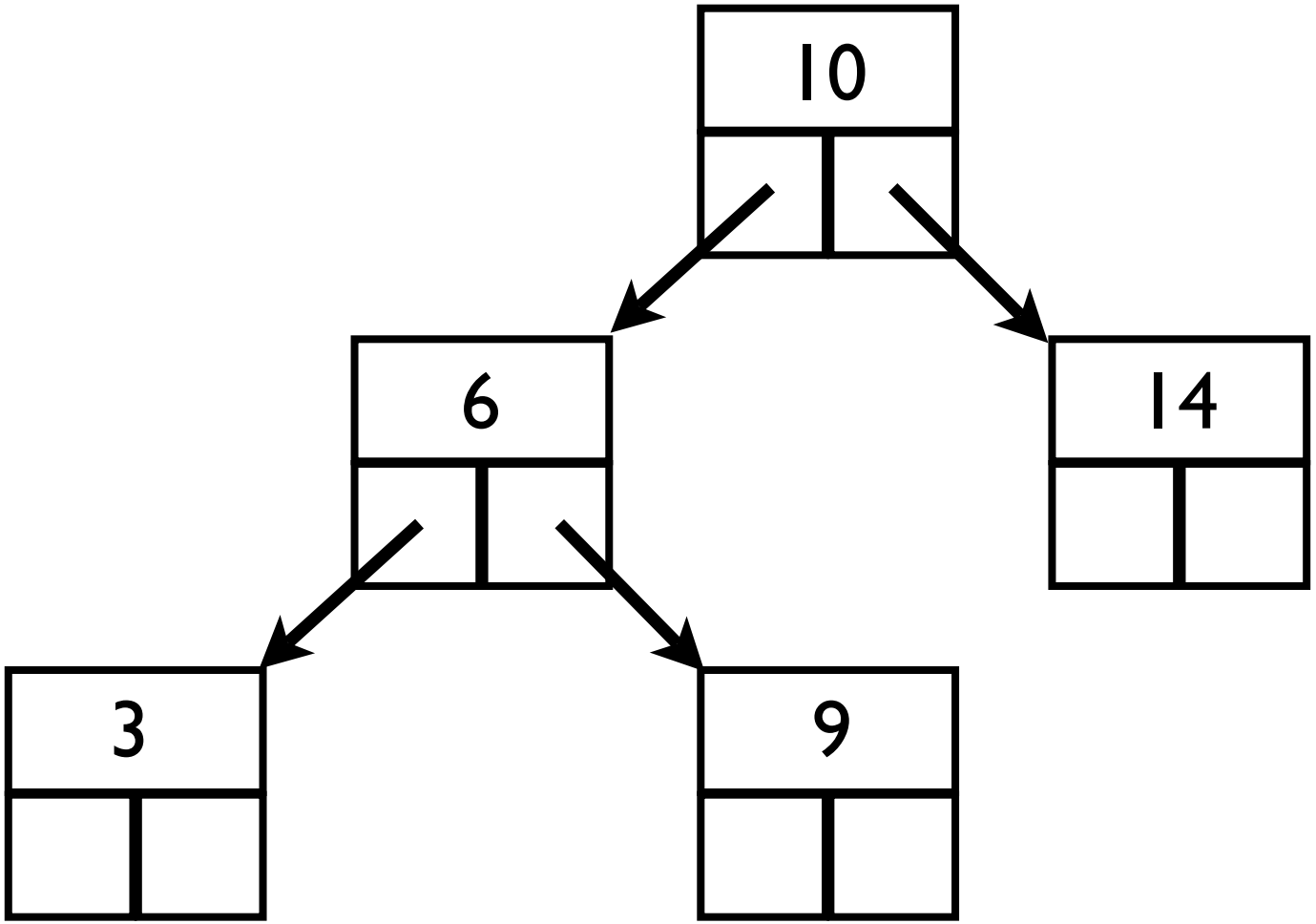
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



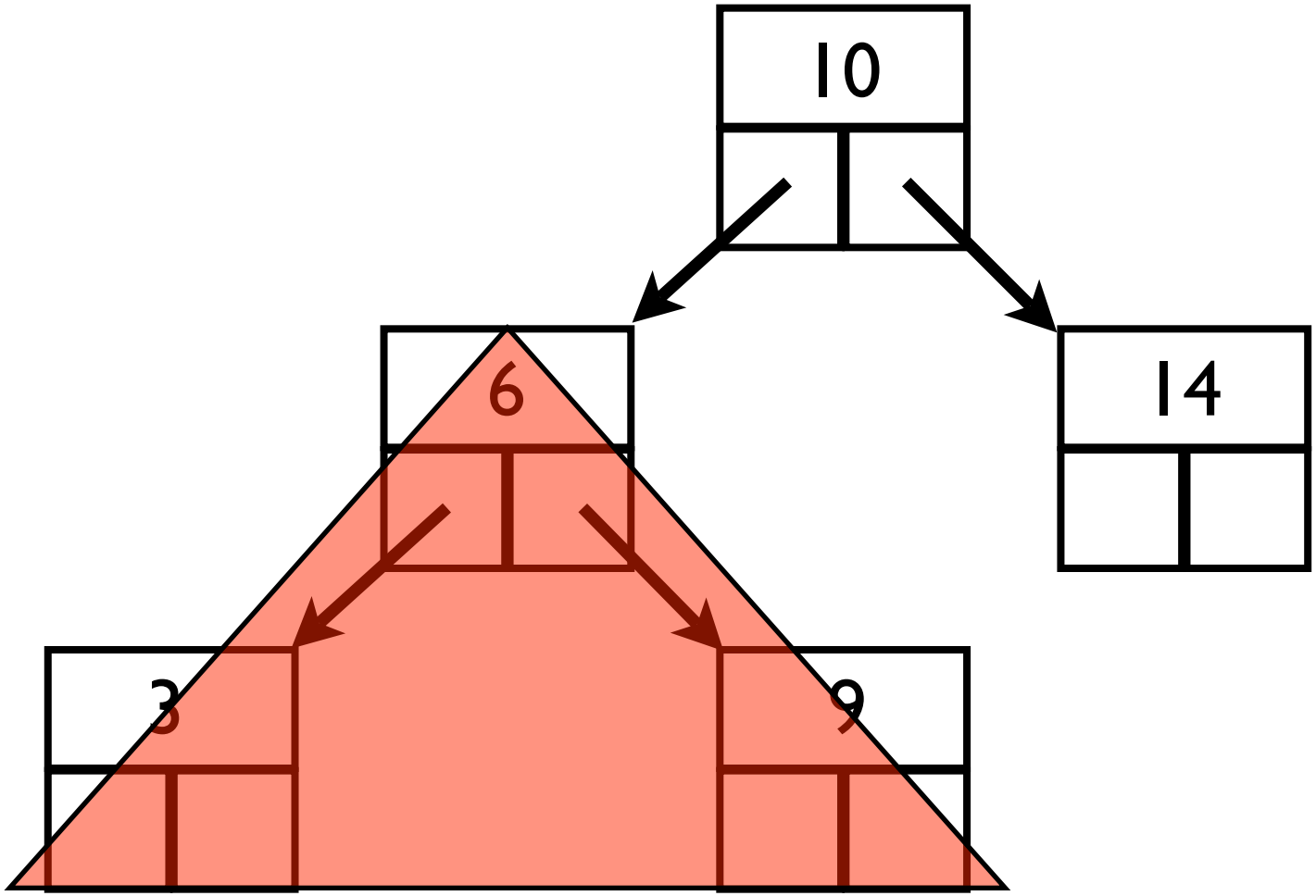
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



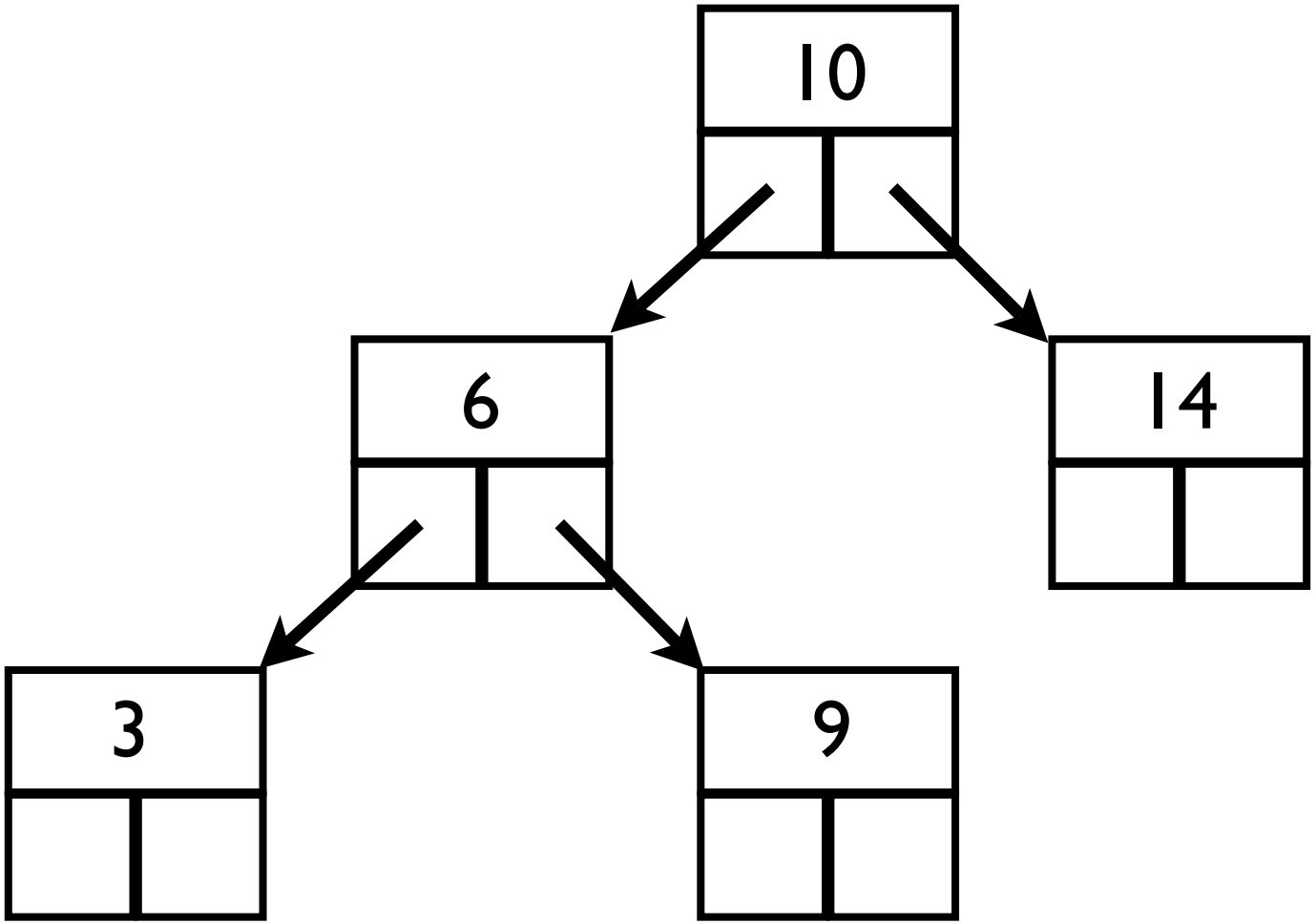
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



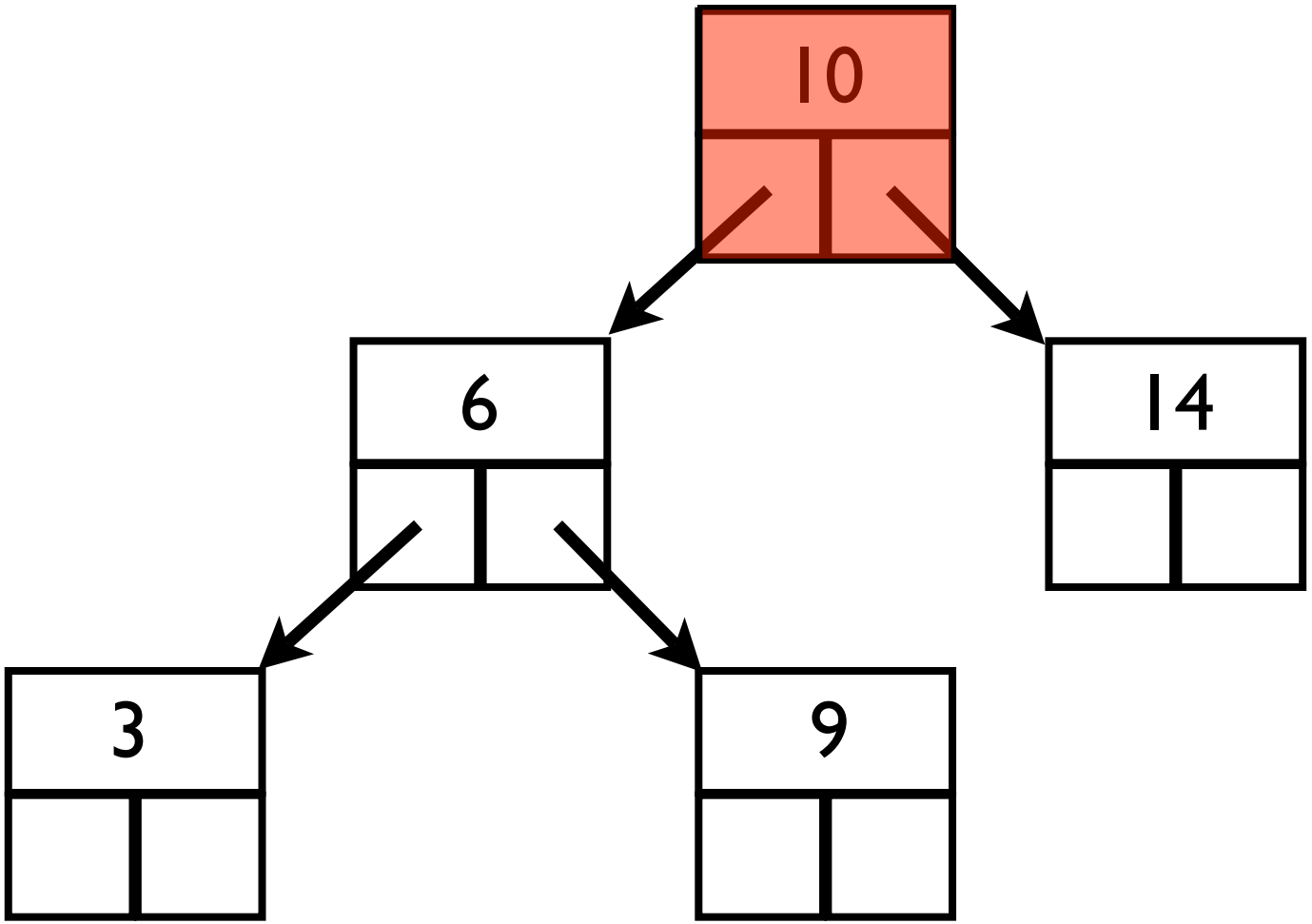
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



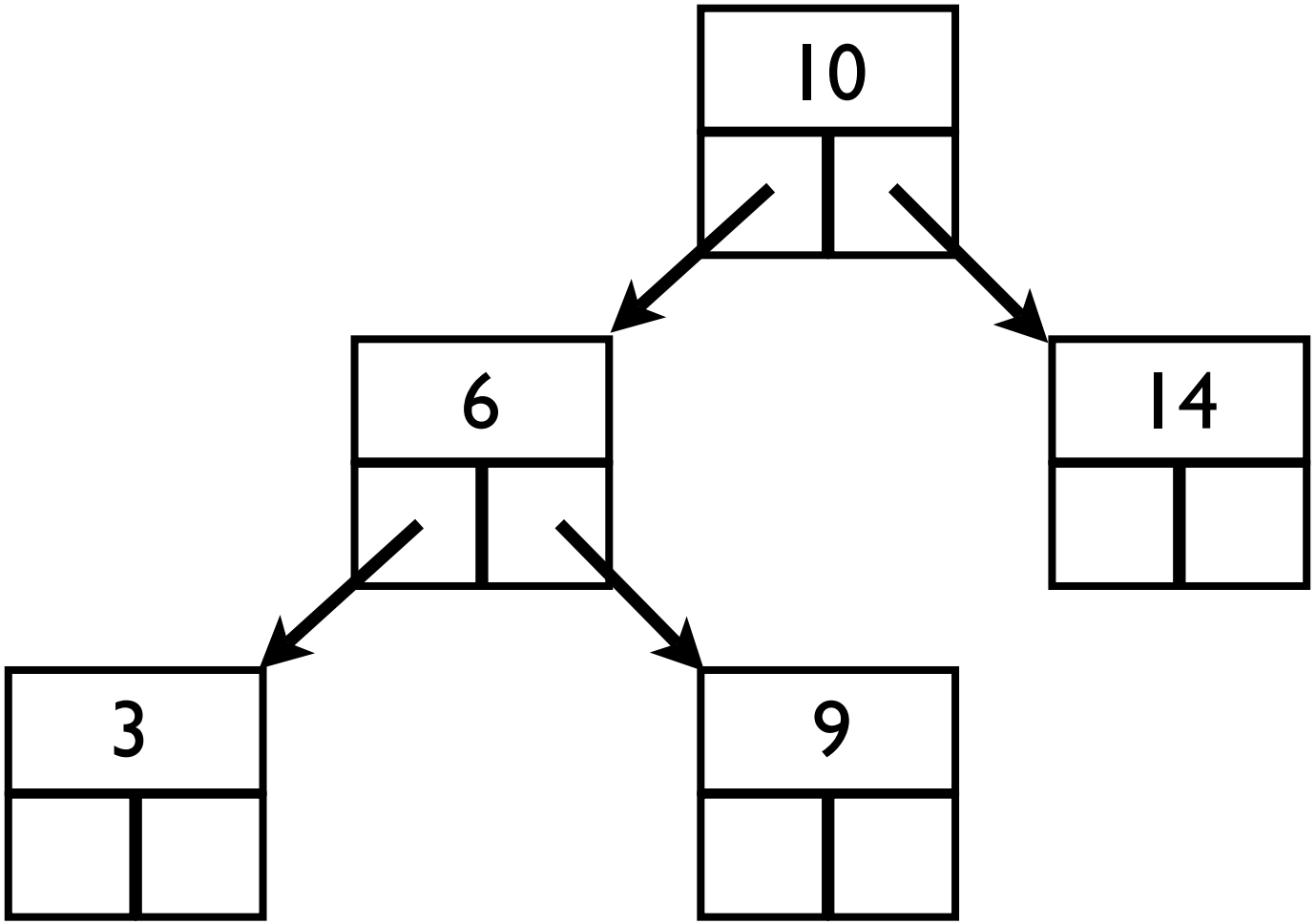
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



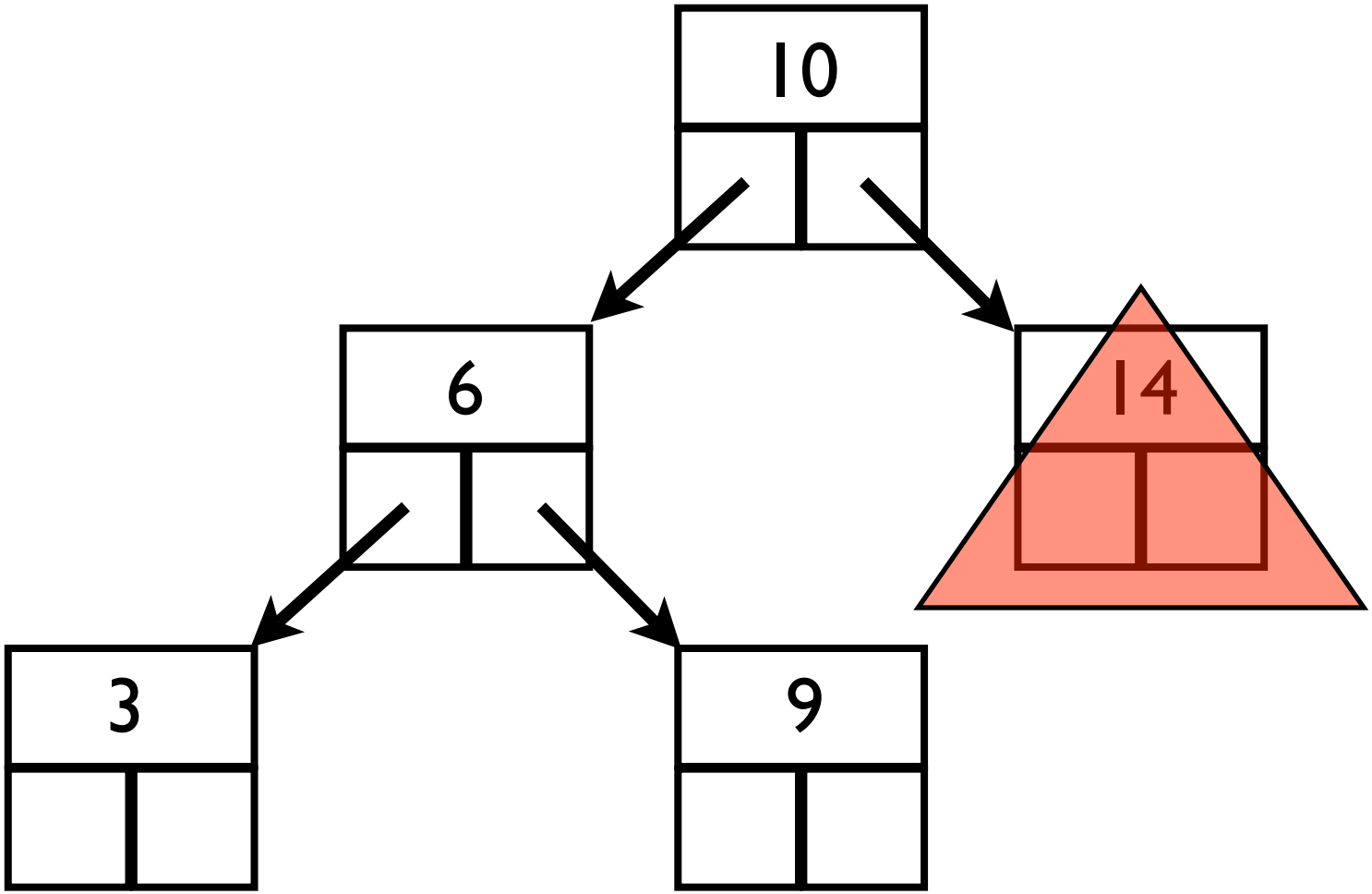
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



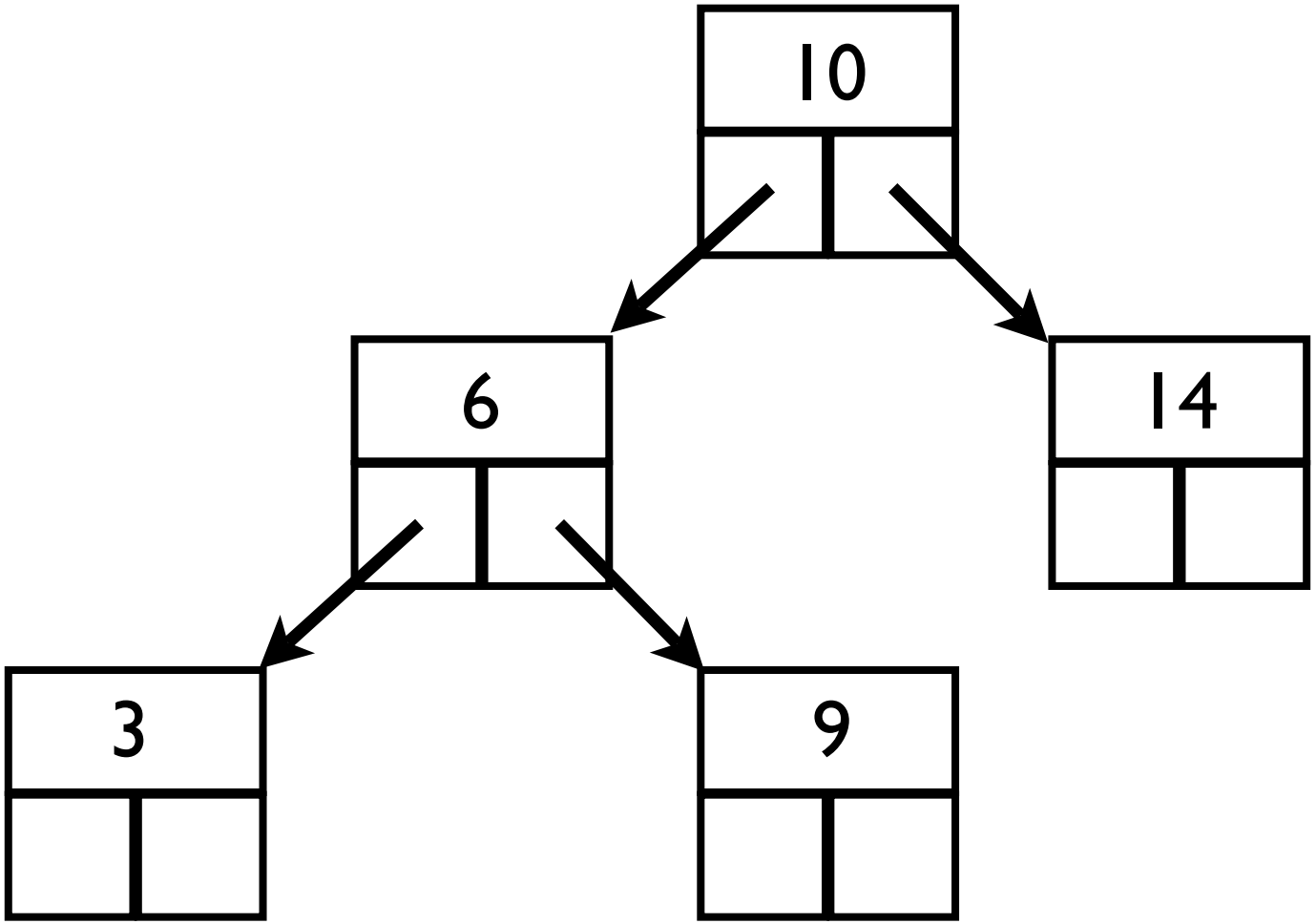
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



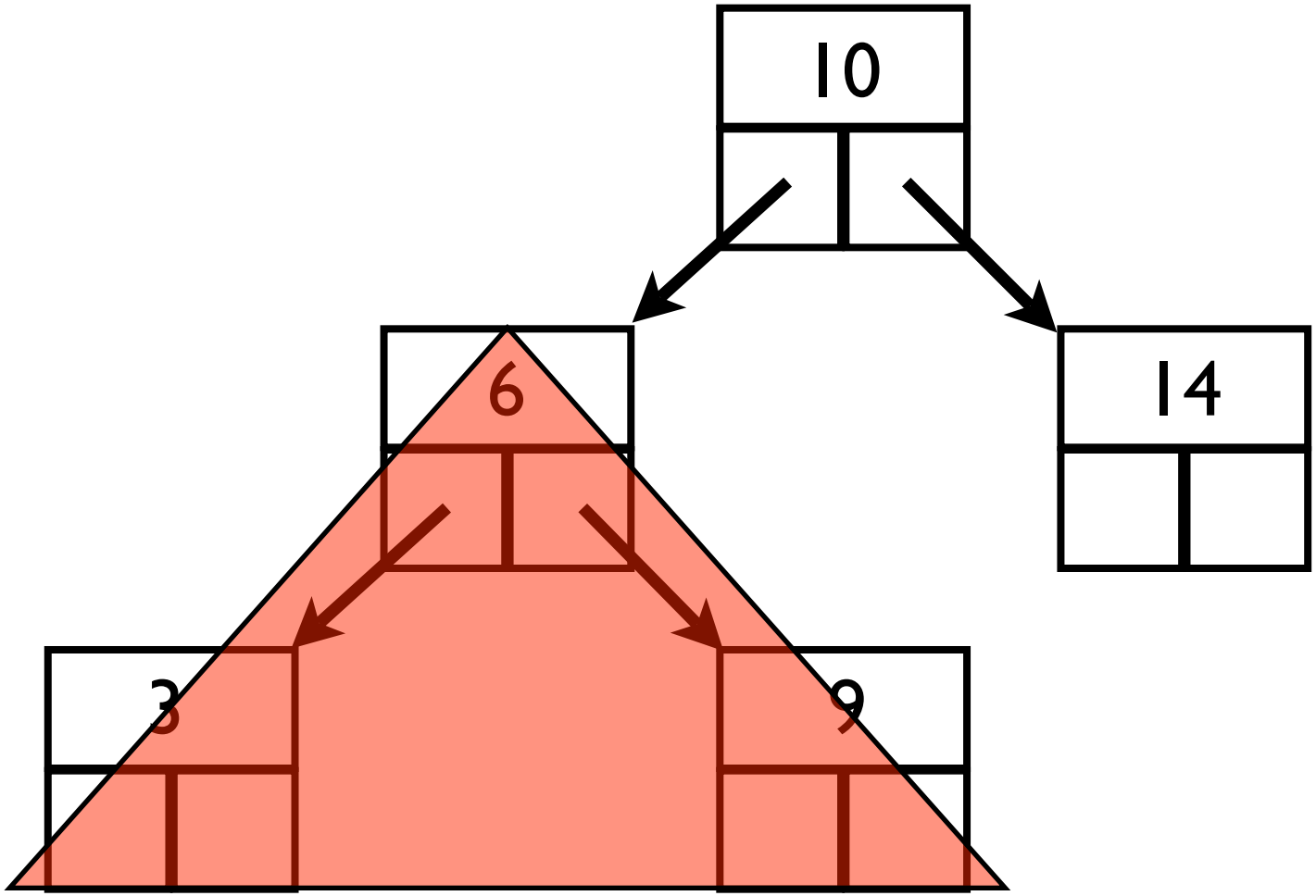
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



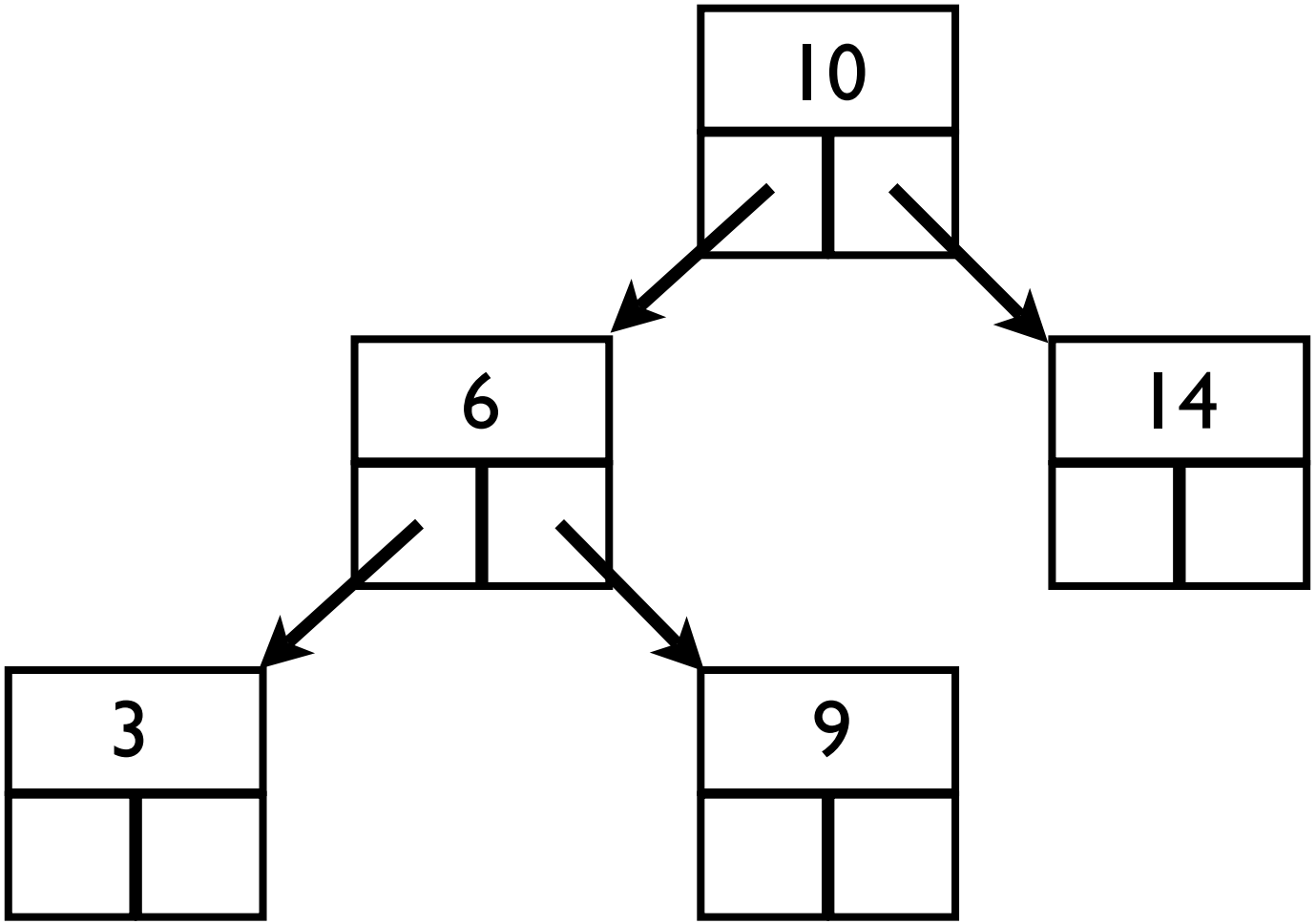
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



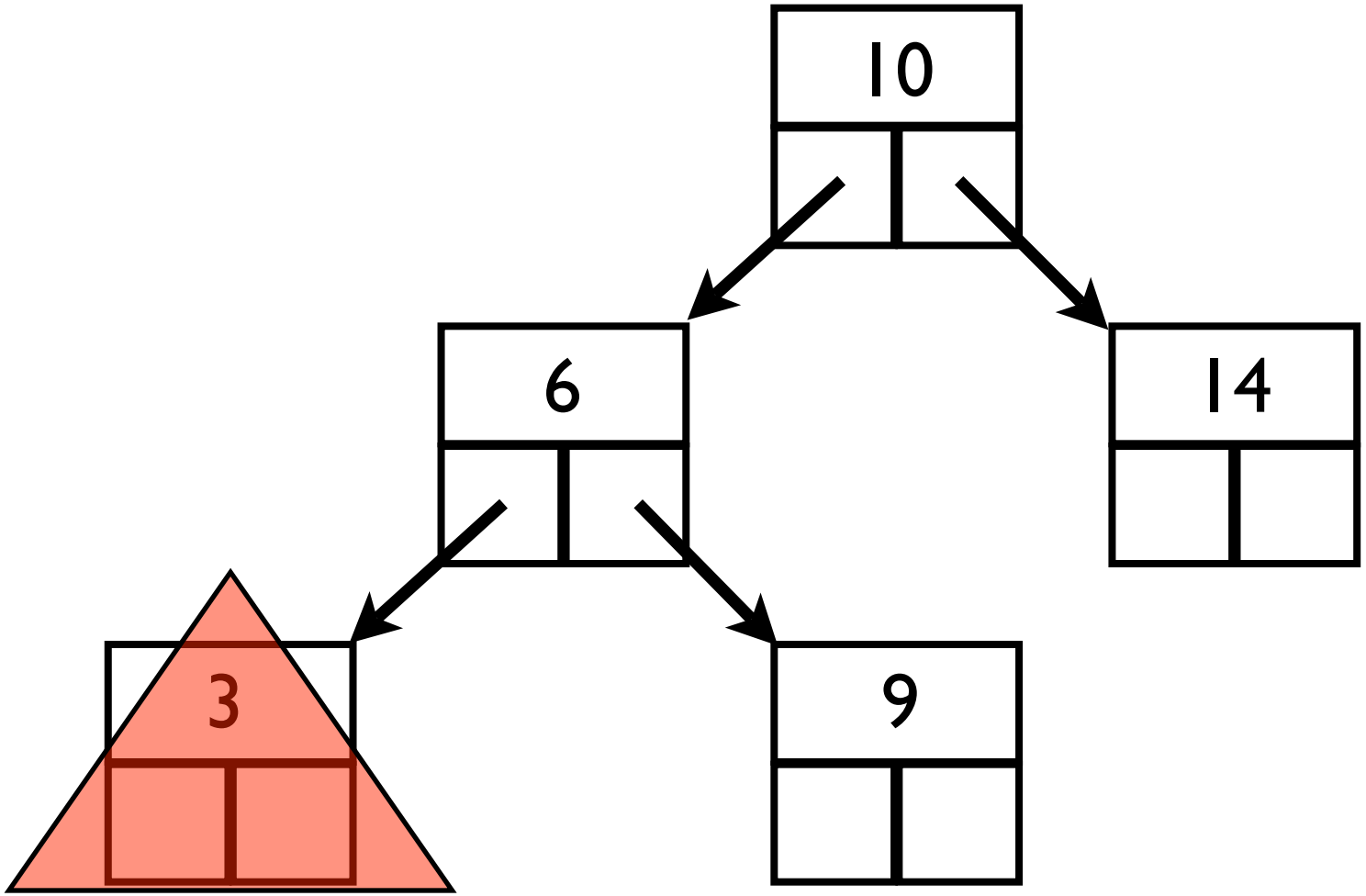
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



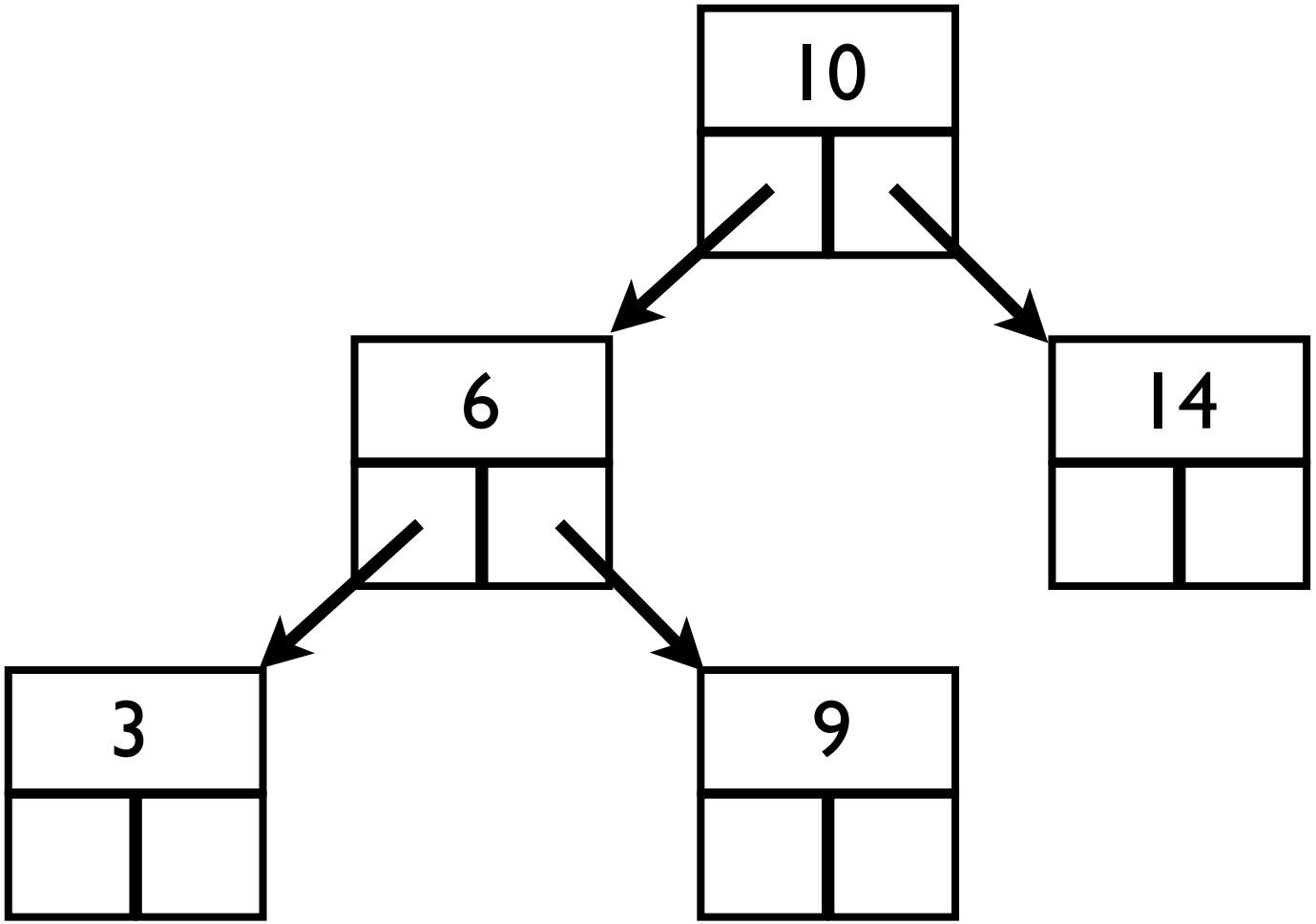
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



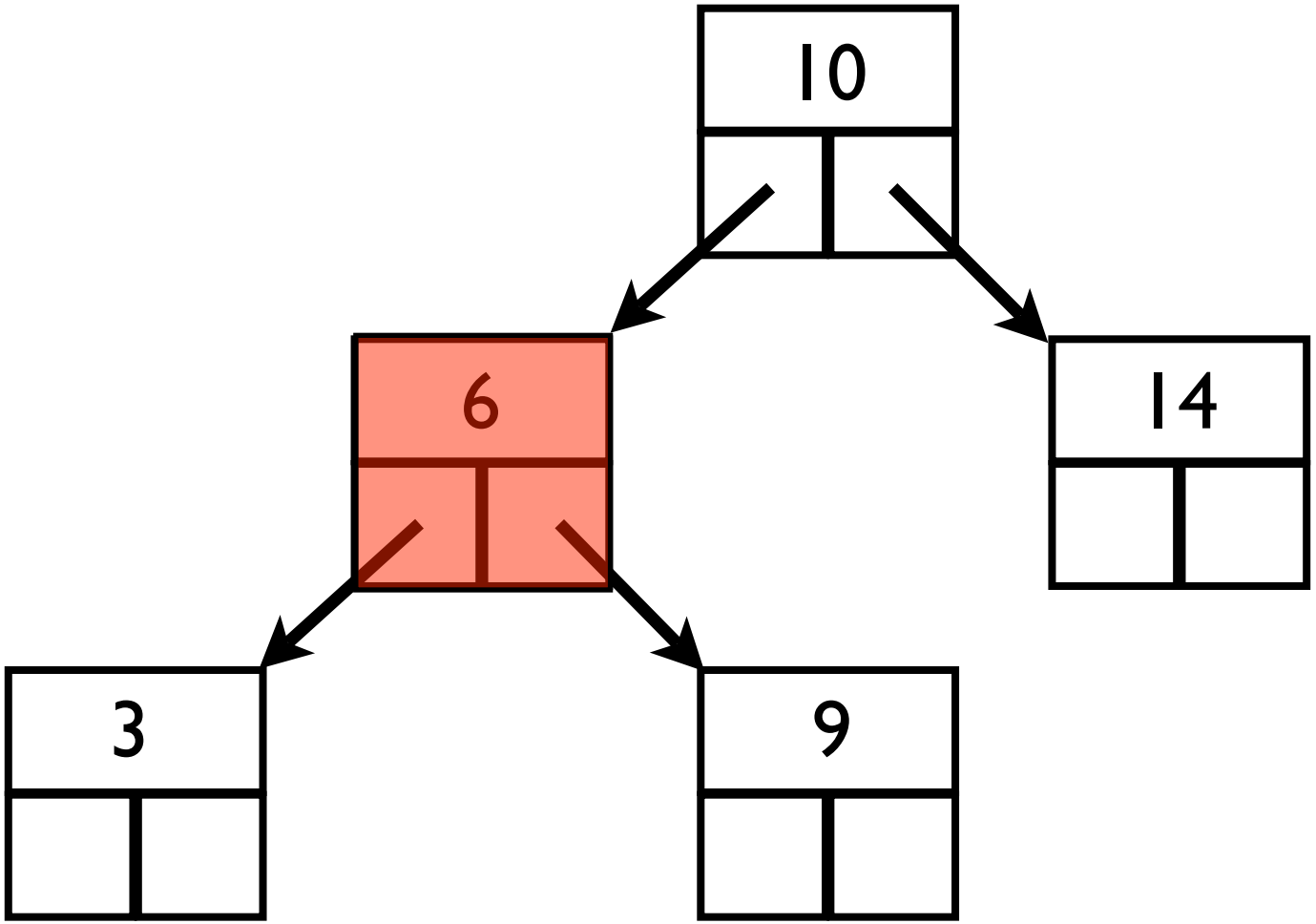
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



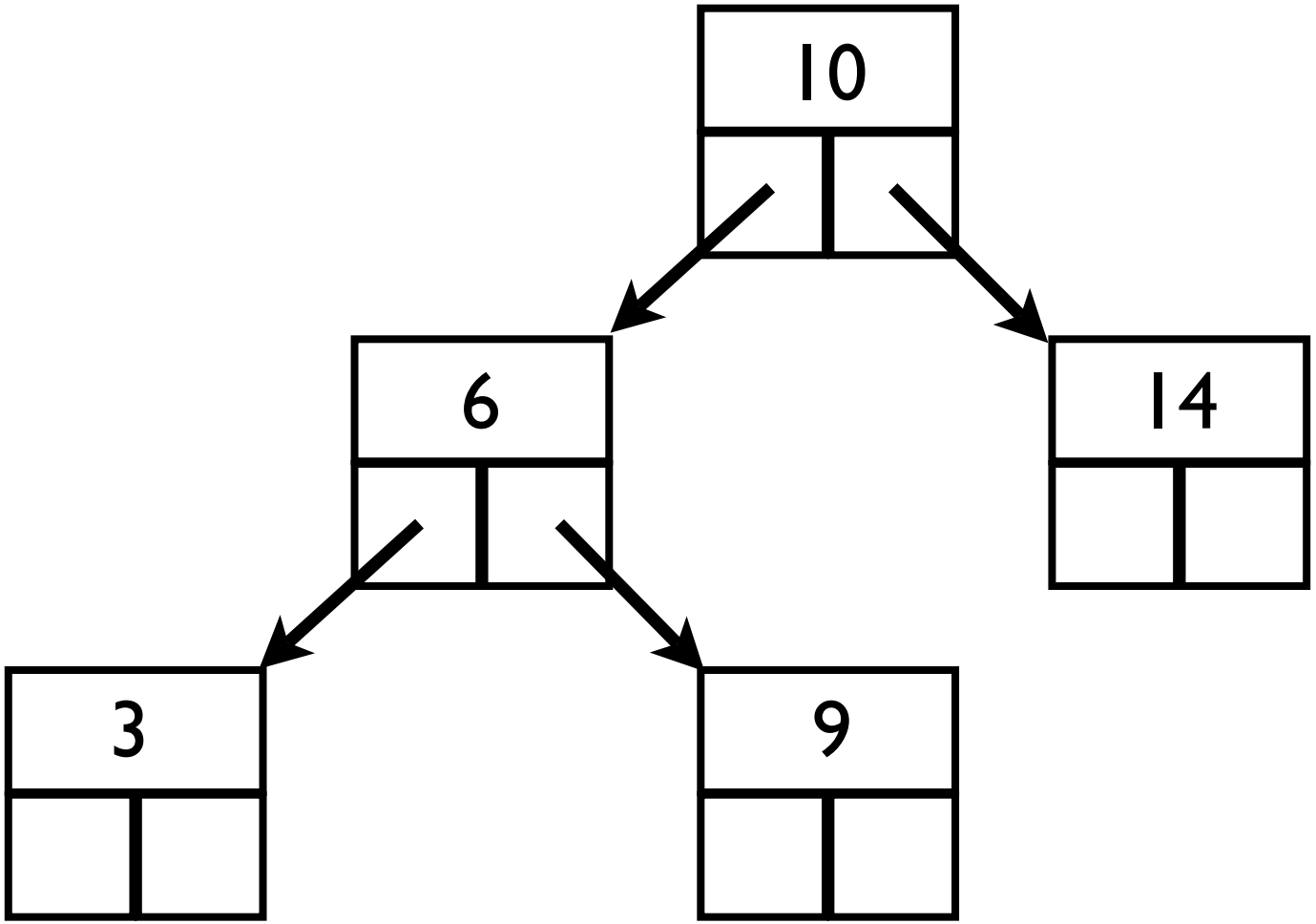
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



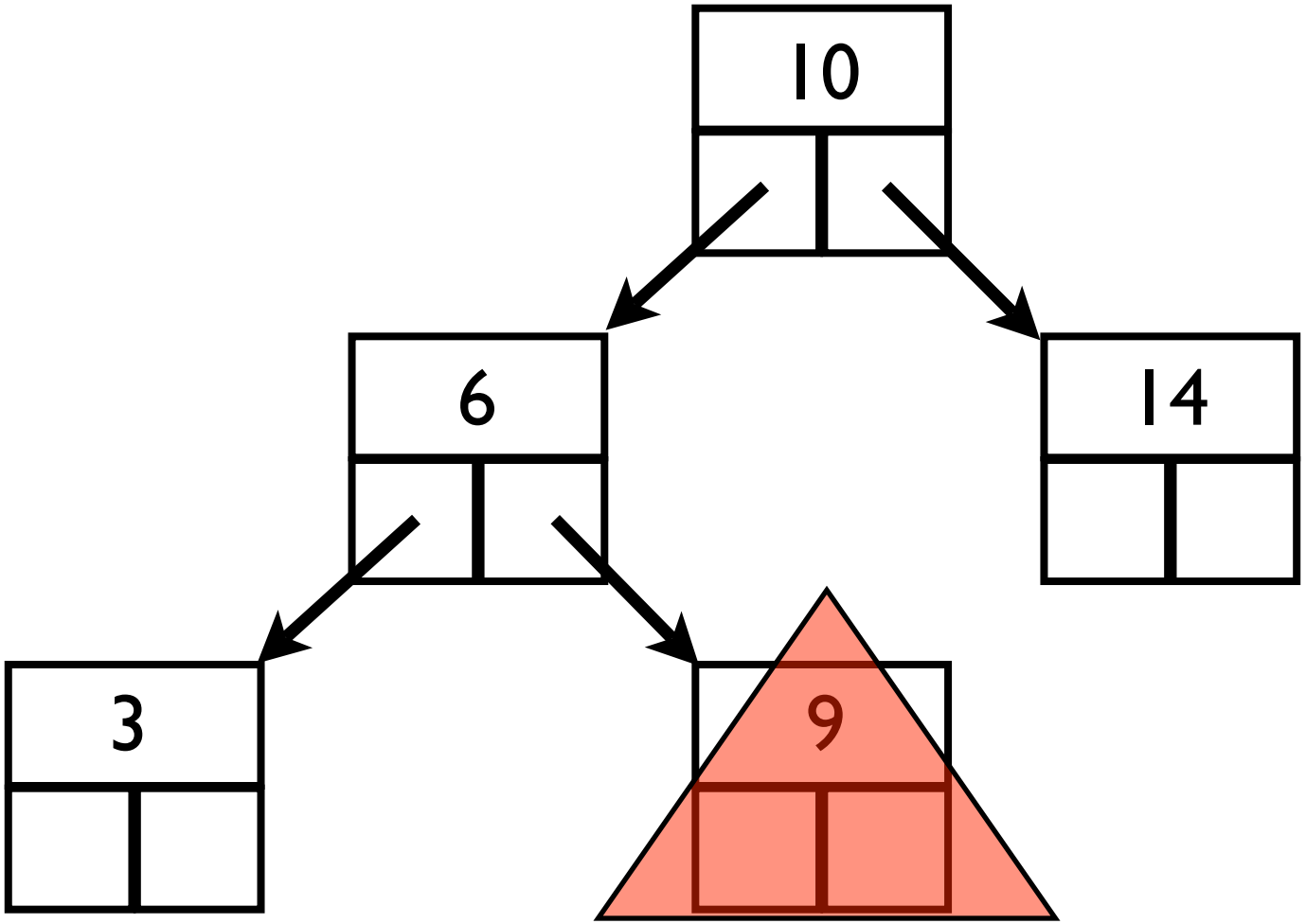
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



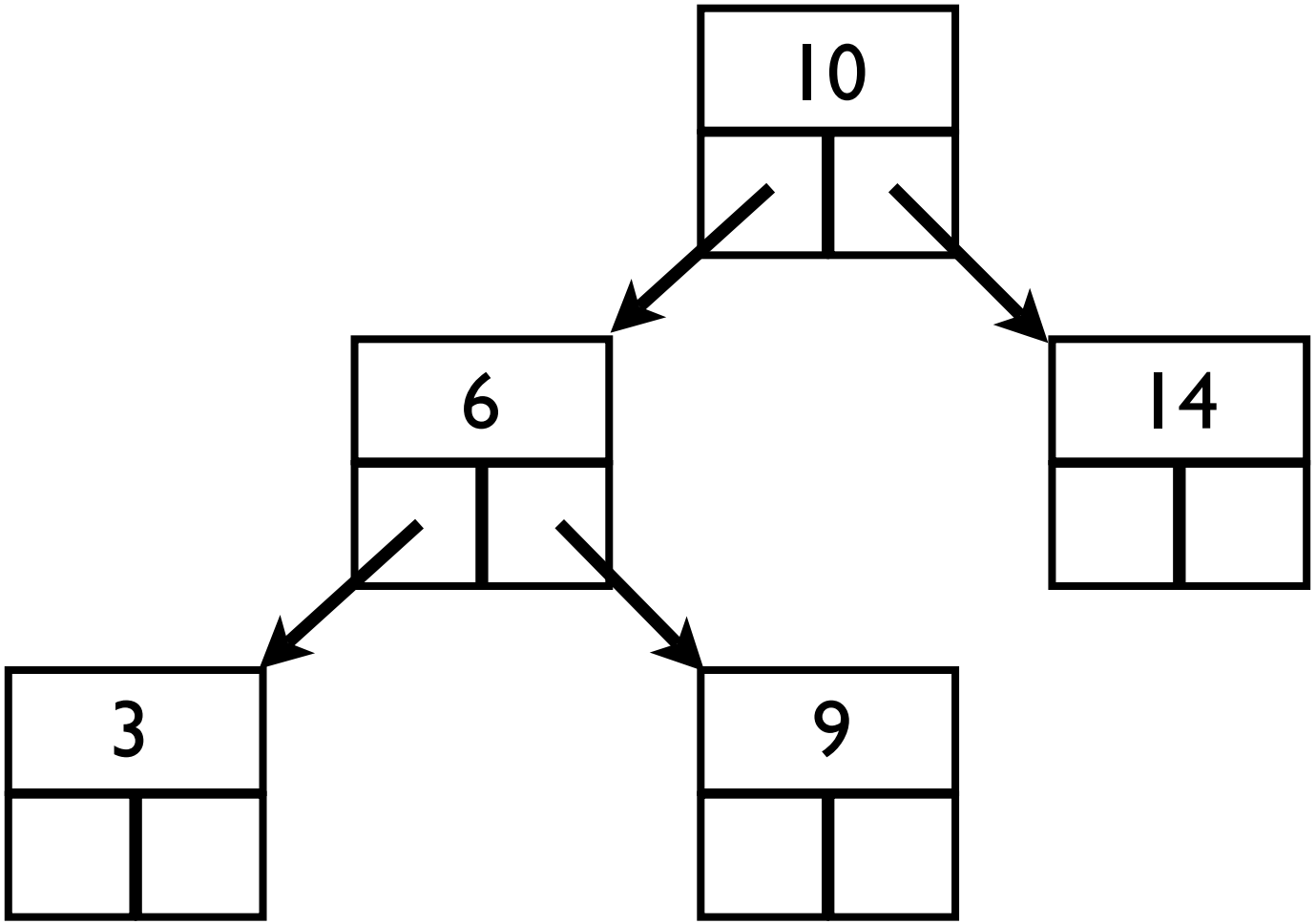
Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



Implementierung von Datent.

Bäume - Baumdurchlauf - Beispiel



Implementierung von Datent.
Bäume - Aufbau

- Aufzählung der Markierungen genügt nicht:

  ABCDEFGHIJKLMN

- Ergänze
  - Durchlauf, z.B. preorder
  - Markierungen für leere Teilbäume:

    – (= Teilbaum ist leer)

Implementierung von Datent.

Bäume - Aufbau

Implementierung von Datent.

Bäume - Aufbau

ABCDEFGHIJKLMN

Implementierung von Datent.

Bäume - Aufbau

ABCDEFGHIJKLMN

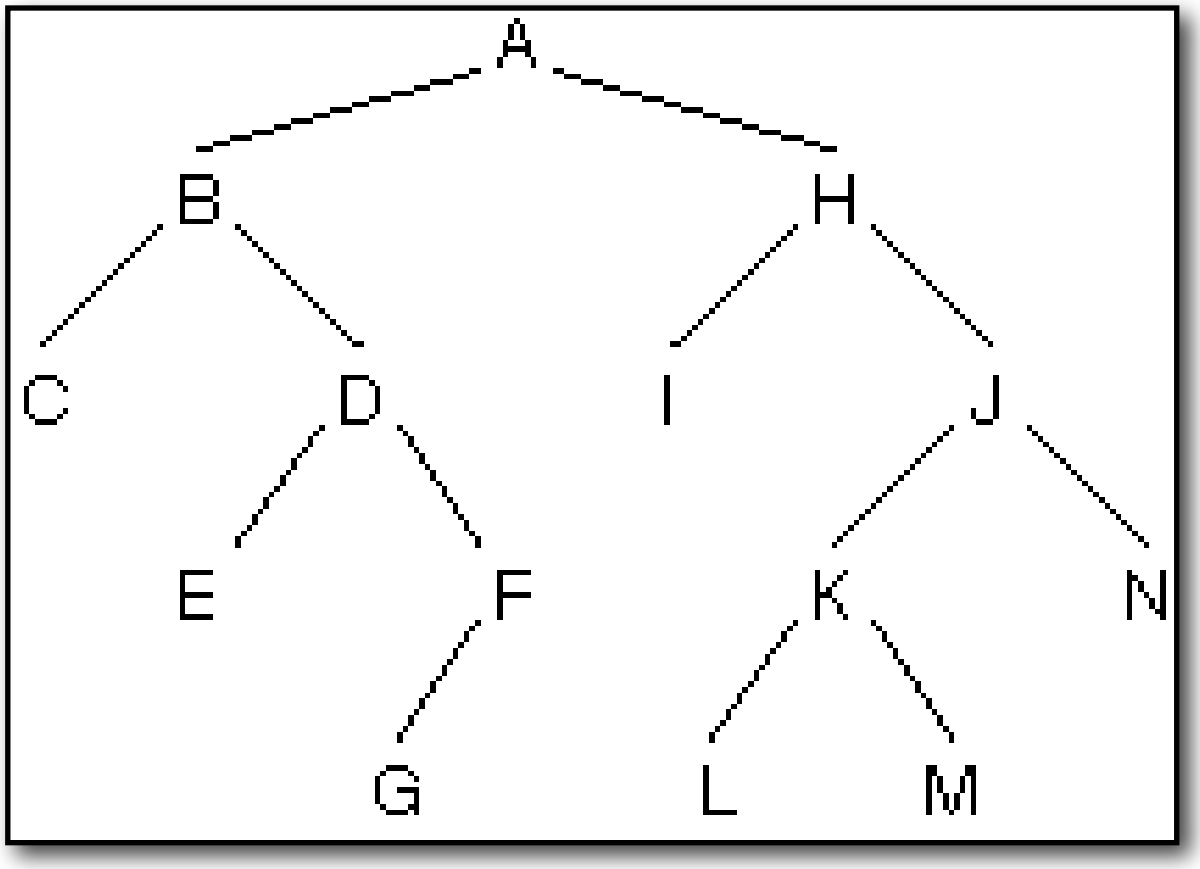
ABC--DE--FG---HI--JKL--M--N--

Implementierung von Datent.

Bäume - Aufbau

ABCDEFGHIJKLMN

ABC--DE--FG---HI--JKL--M--N--



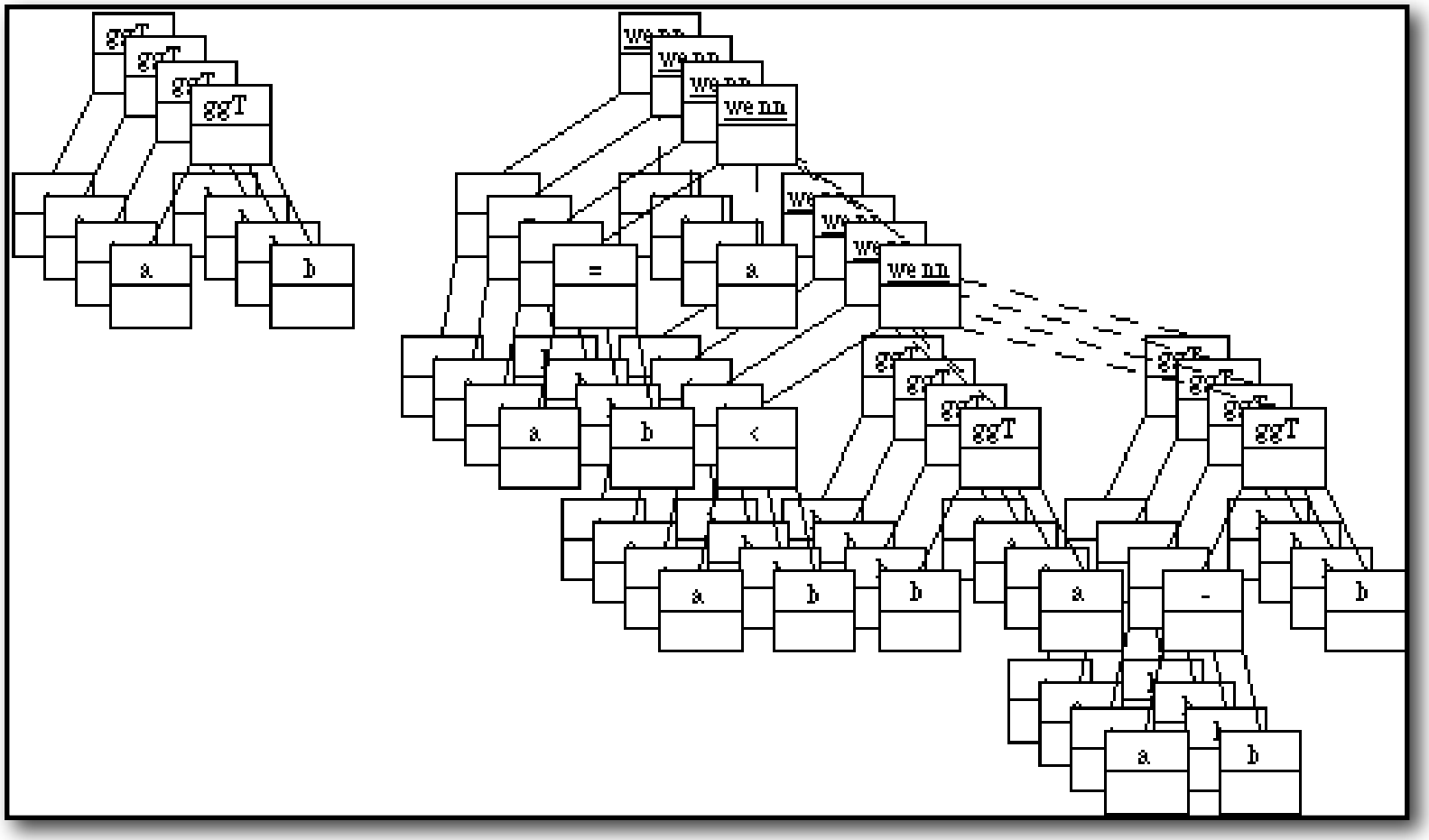
Implementierung von Datent.

Bäume - Aufbau

```
procedure Aufbau(var p: ↑knoten);
var ch: char;
begin
   read(ch);
   if ch<>'-' then
   begin
      new(p); p↑.inhalt:=ch;
      Aufbau(p↑.lt);
      Aufbau(p↑.rt)
   end else p:=nil
end.
```

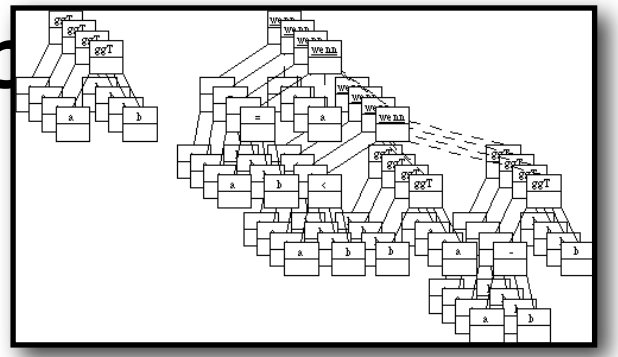
Implementierung von Datent.
Rekursion im Kontrollbereich

- Ziel: Implementierung der Formularmaschine



Implementierung von Datent.

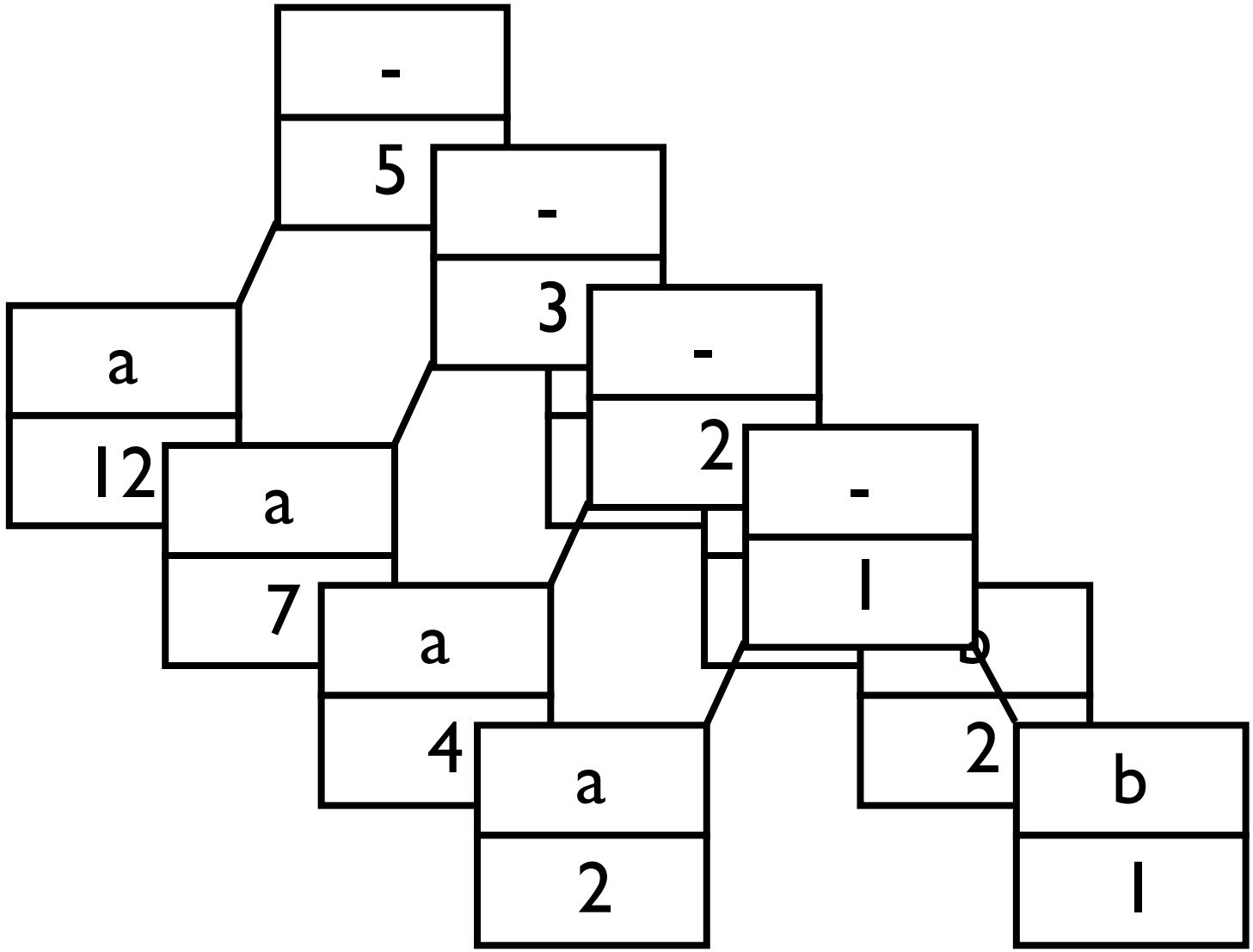
Rekursion im Kontrollb



- Lege Formulare je Inkarnation
- Verschmelze die festen (identischen) Einträge
- Behalte Werteeinträge der Inkarnationen bei (als Folge organisiert)
- Je Aufruf neue Wertezelle an jedem Knoten; diese ist aktuelle gültig
- Bei Beendigung einer Funktion Löschen der aktuellen (zuletzt zugefügten) Wertezellen
- Beobachte: Zugriffsschema eines Stacks

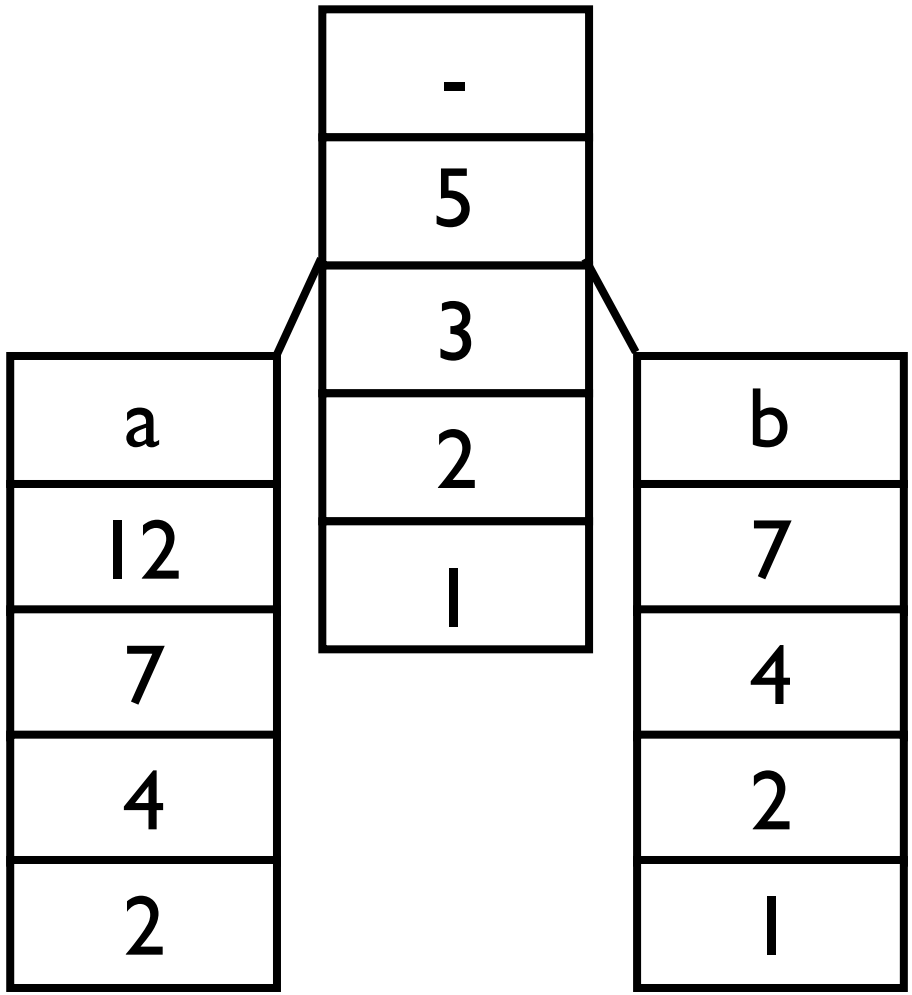
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



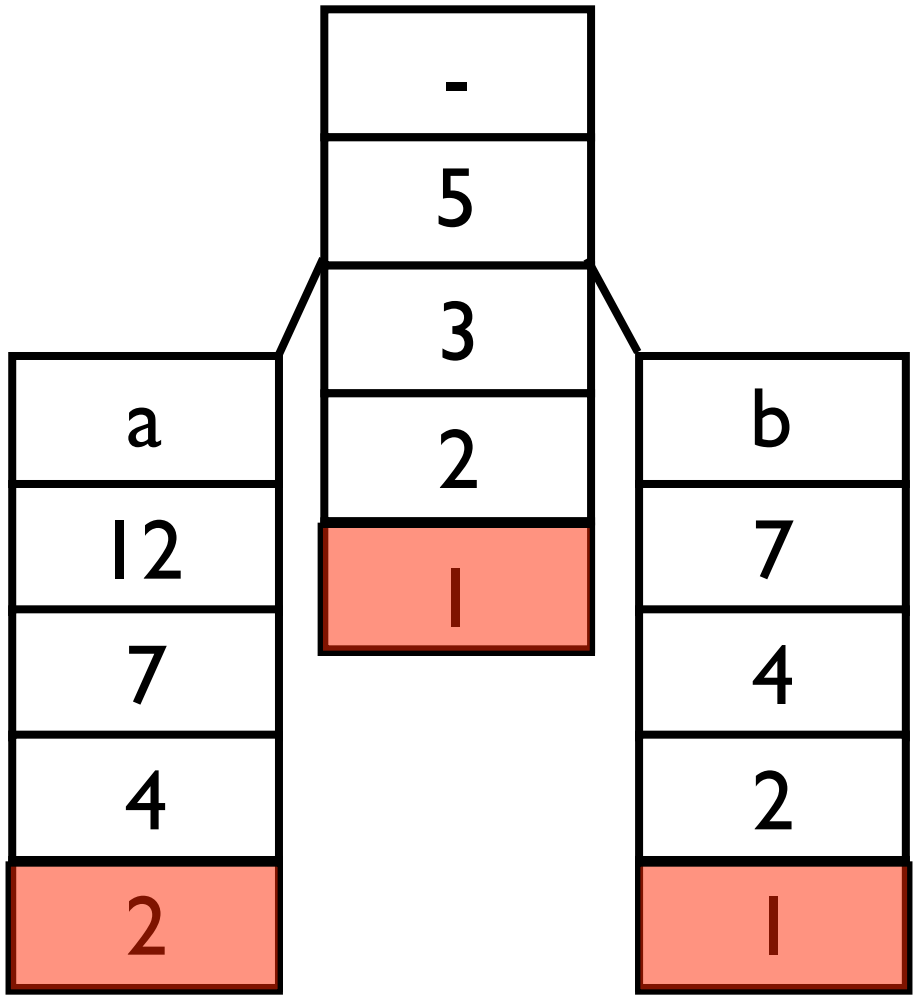
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



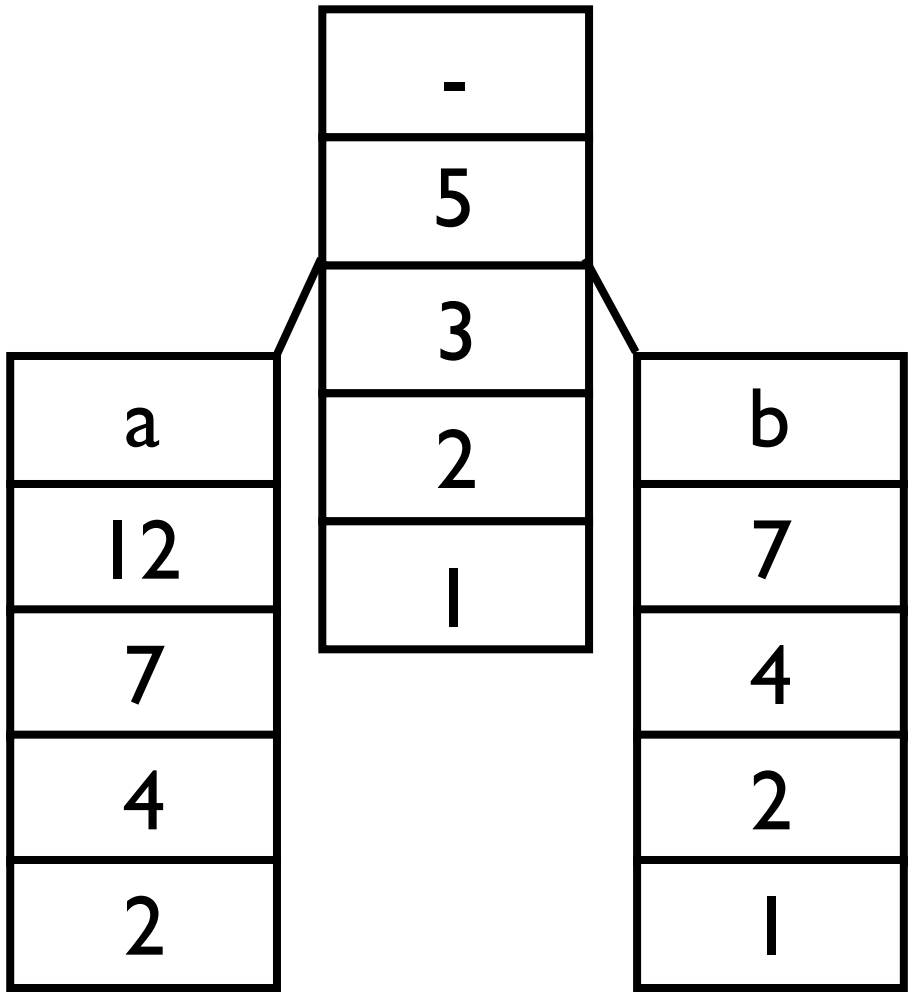
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



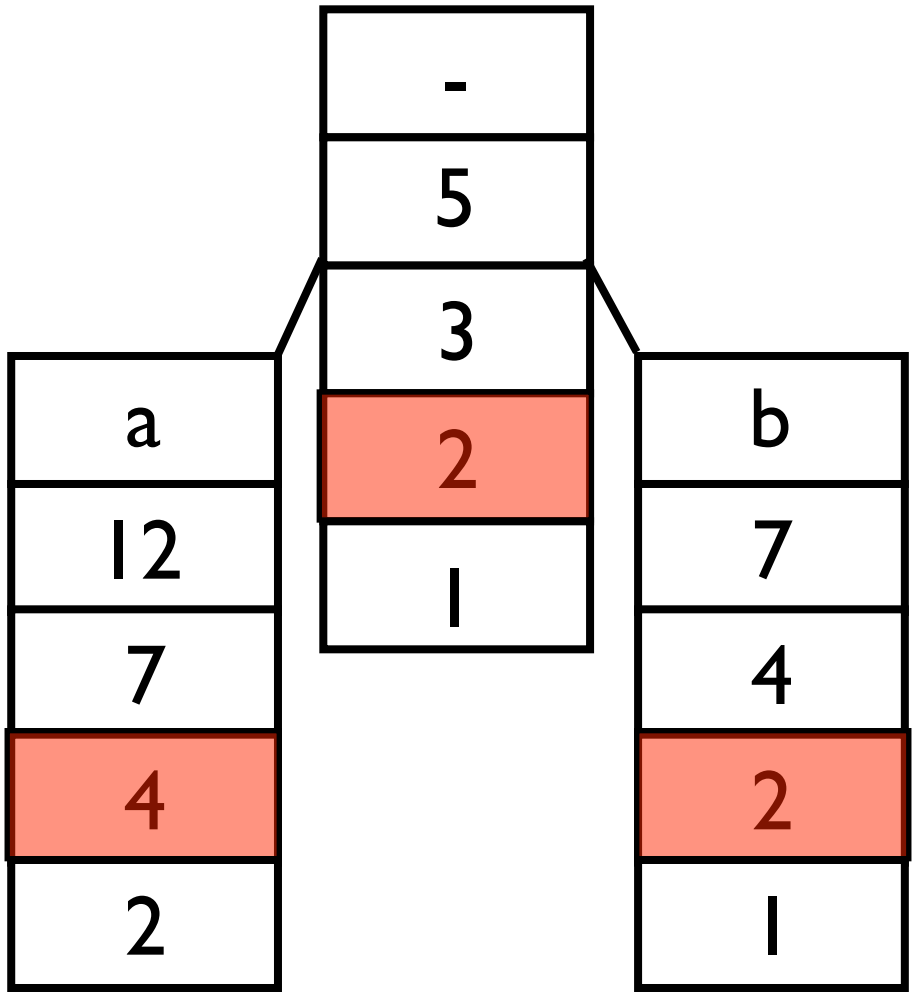
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



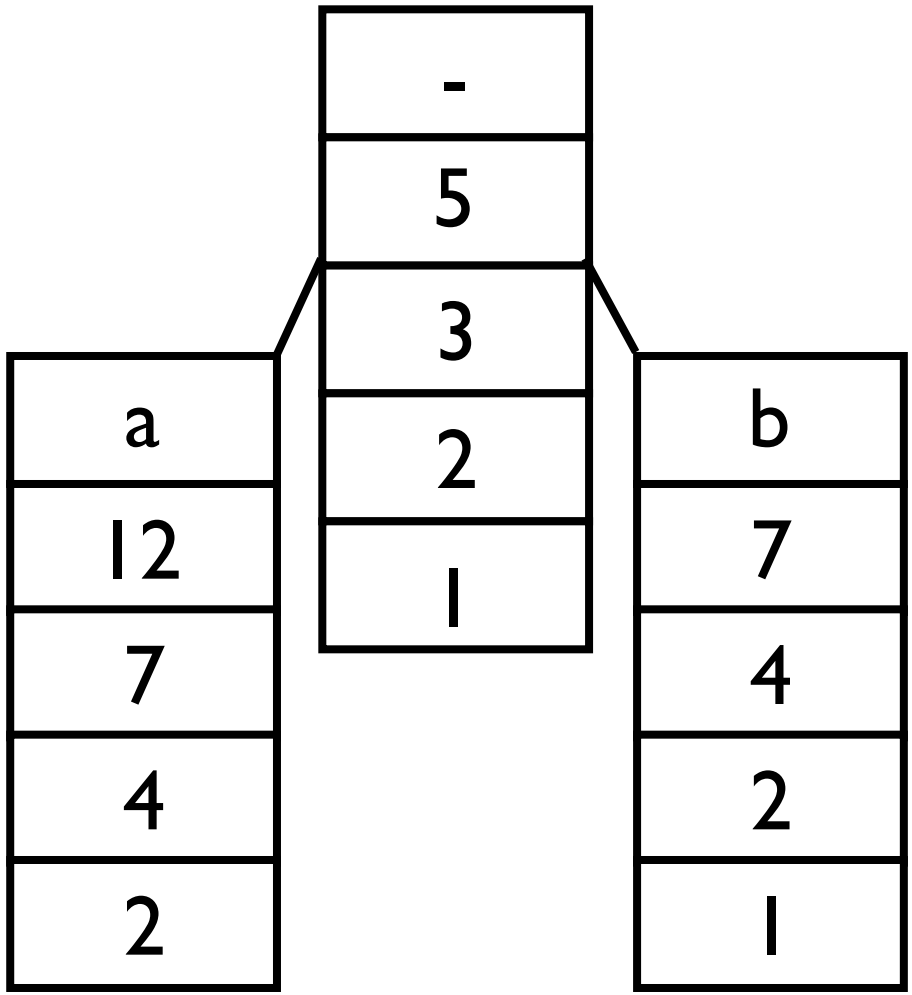
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



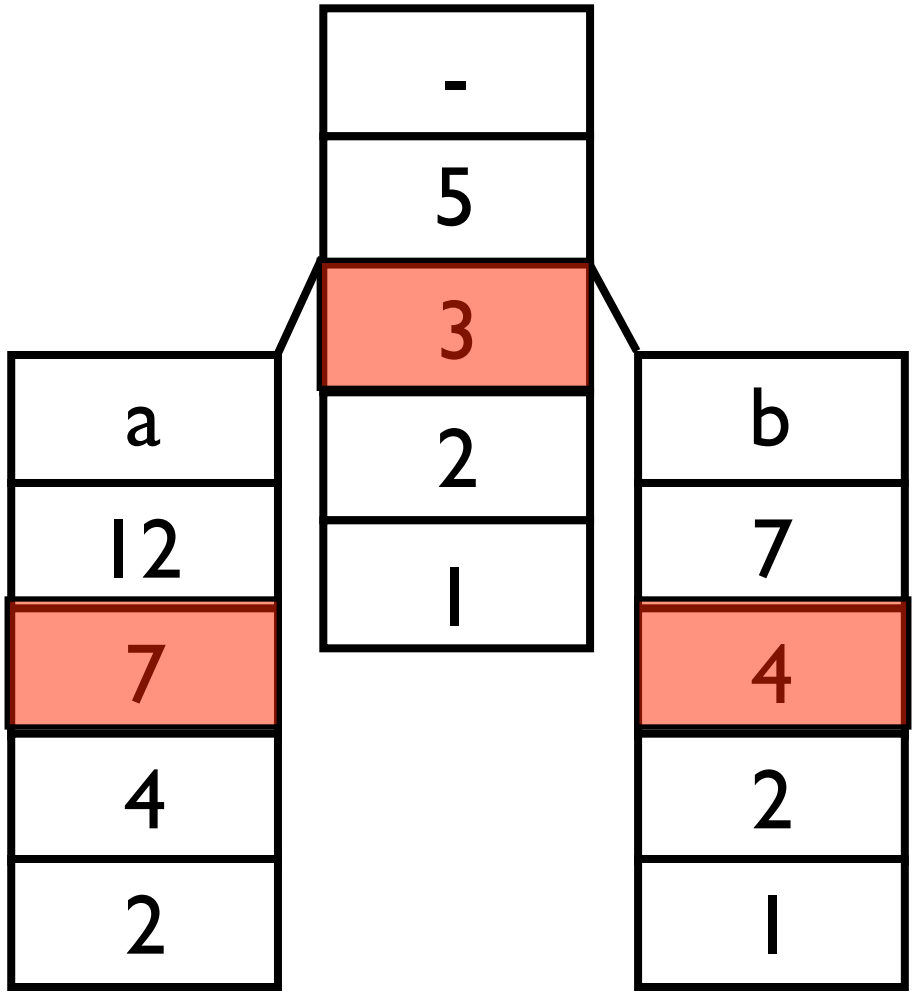
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



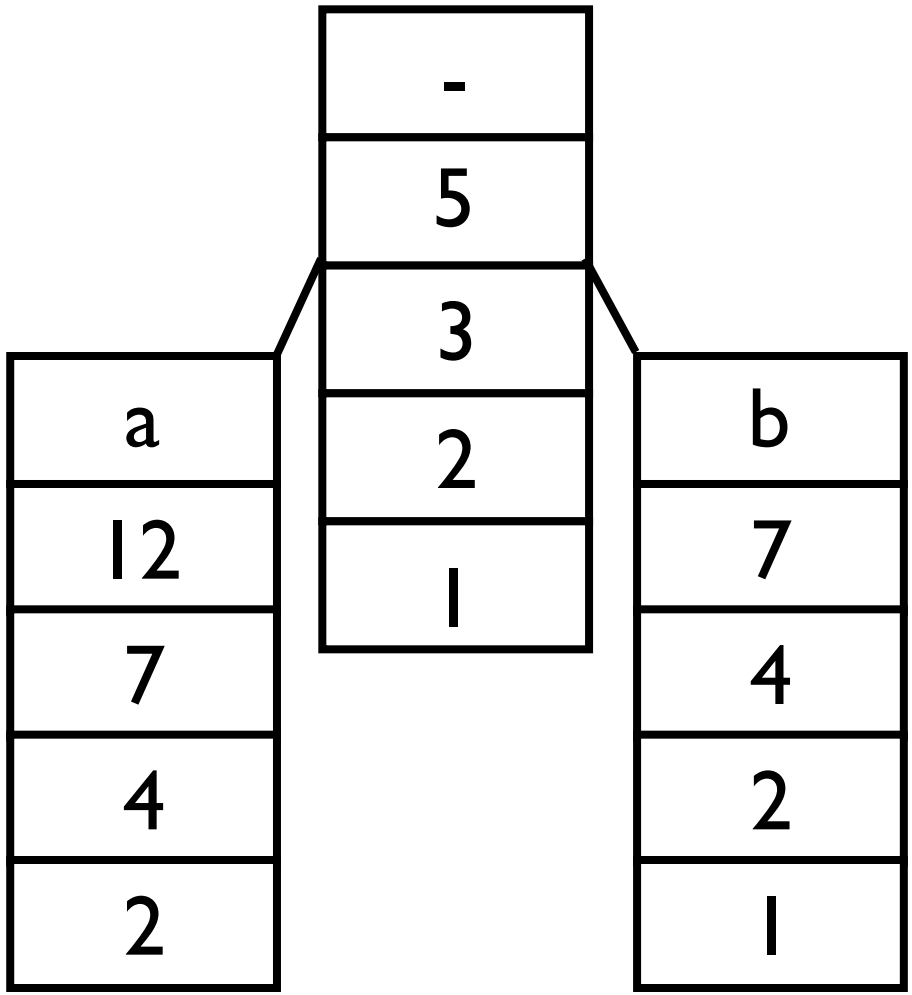
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



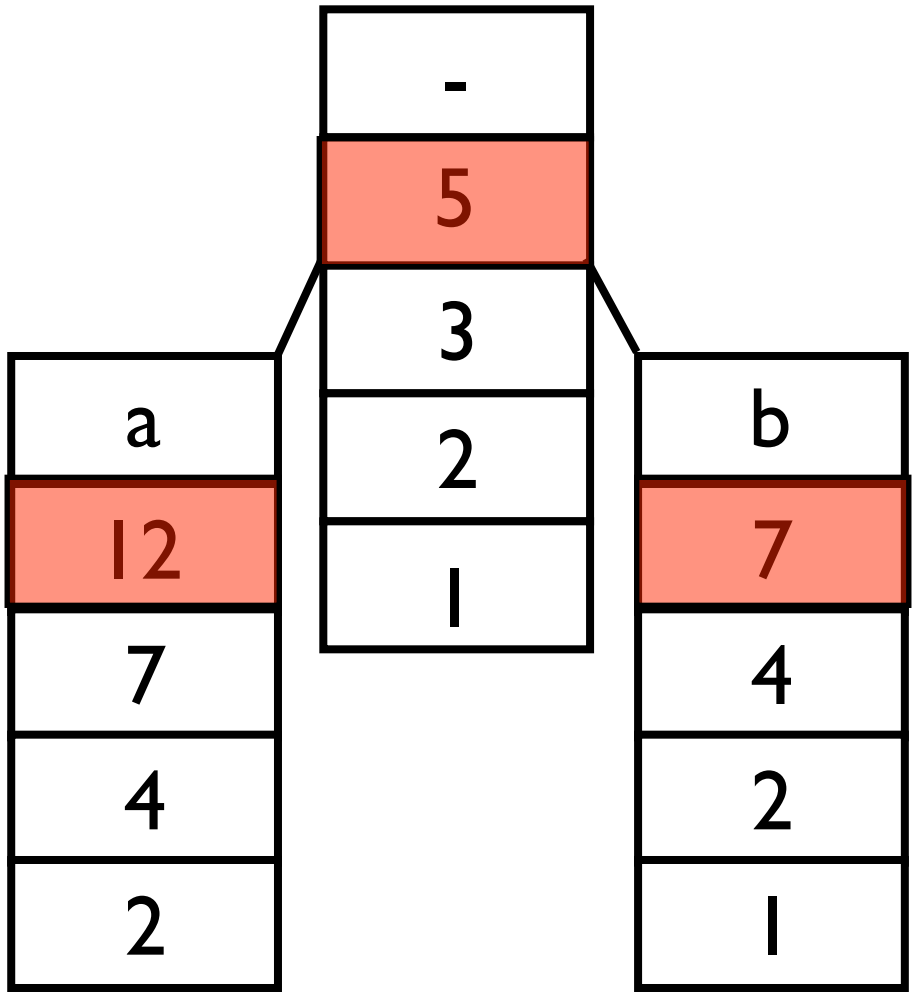
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



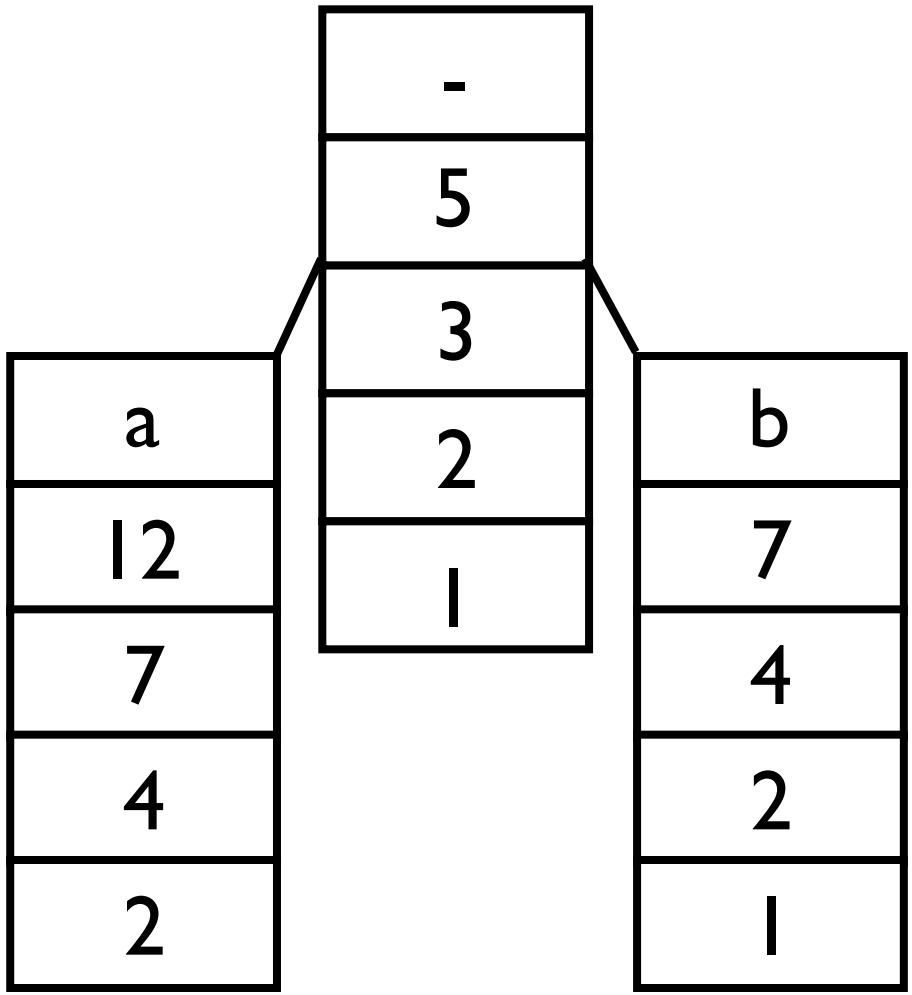
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



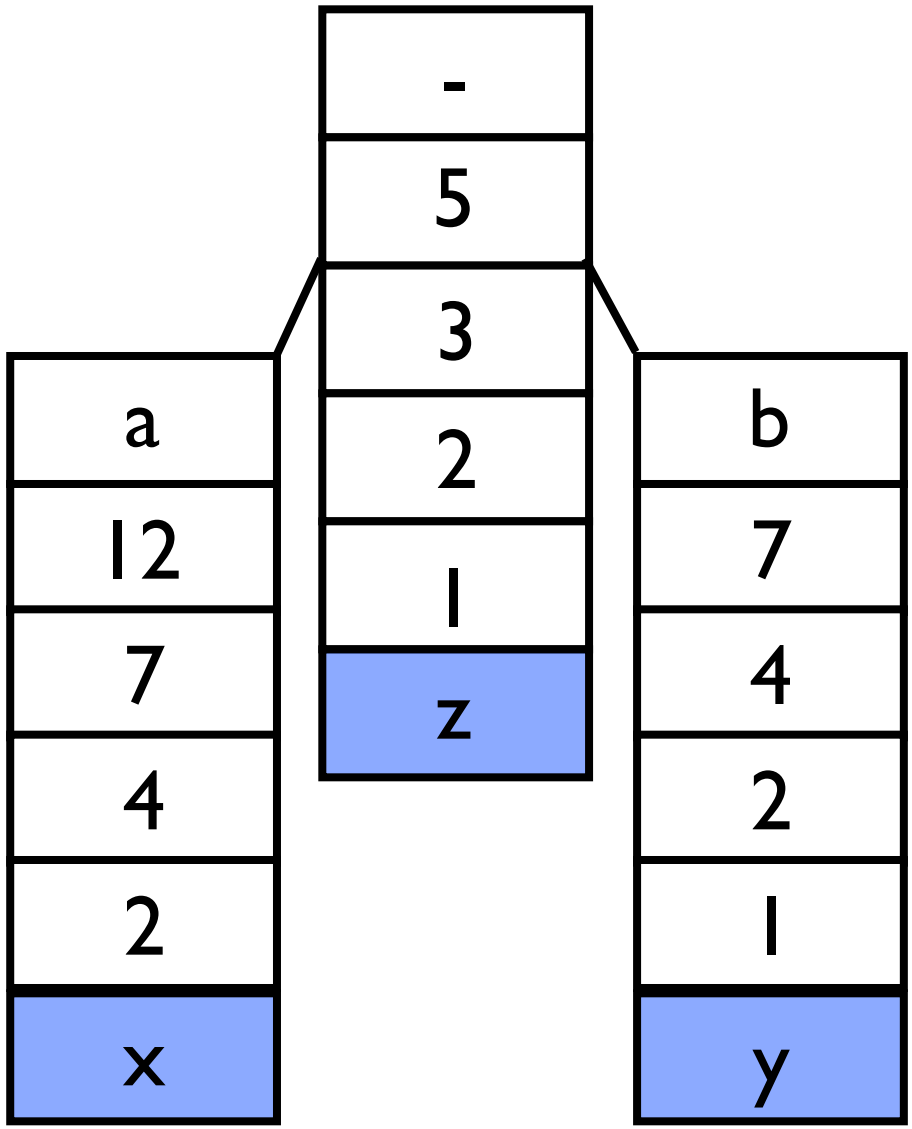
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



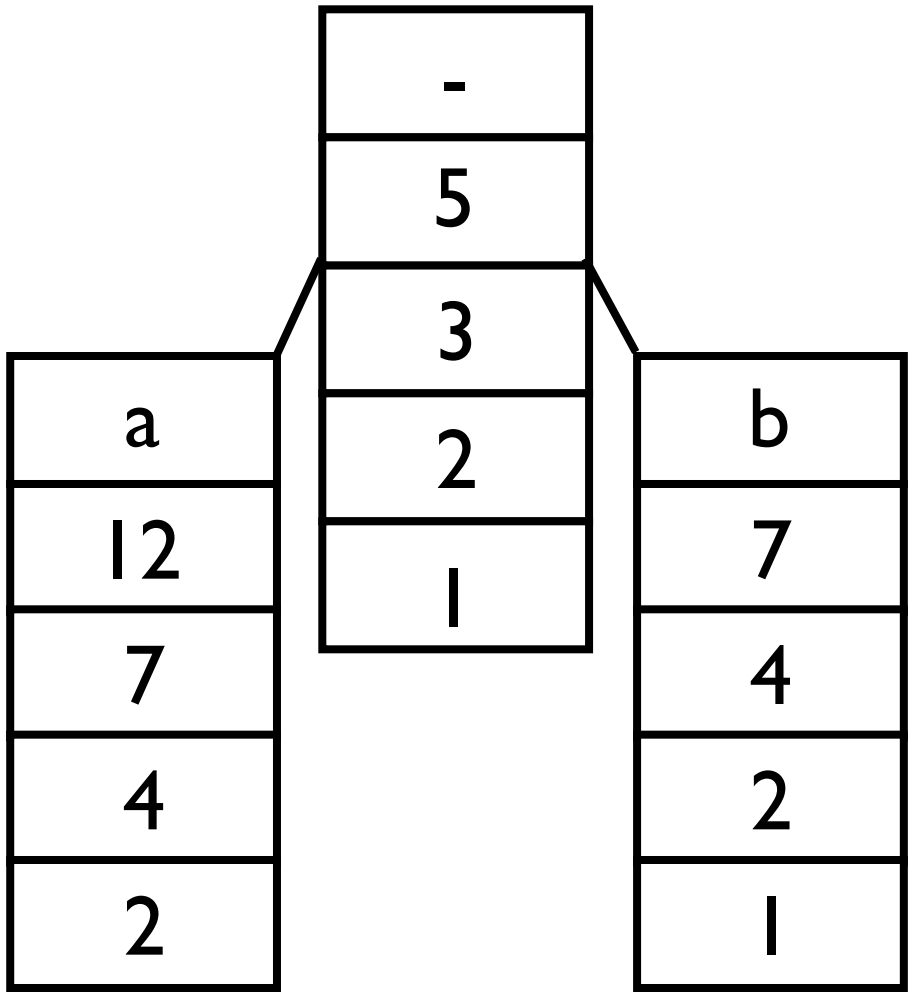
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



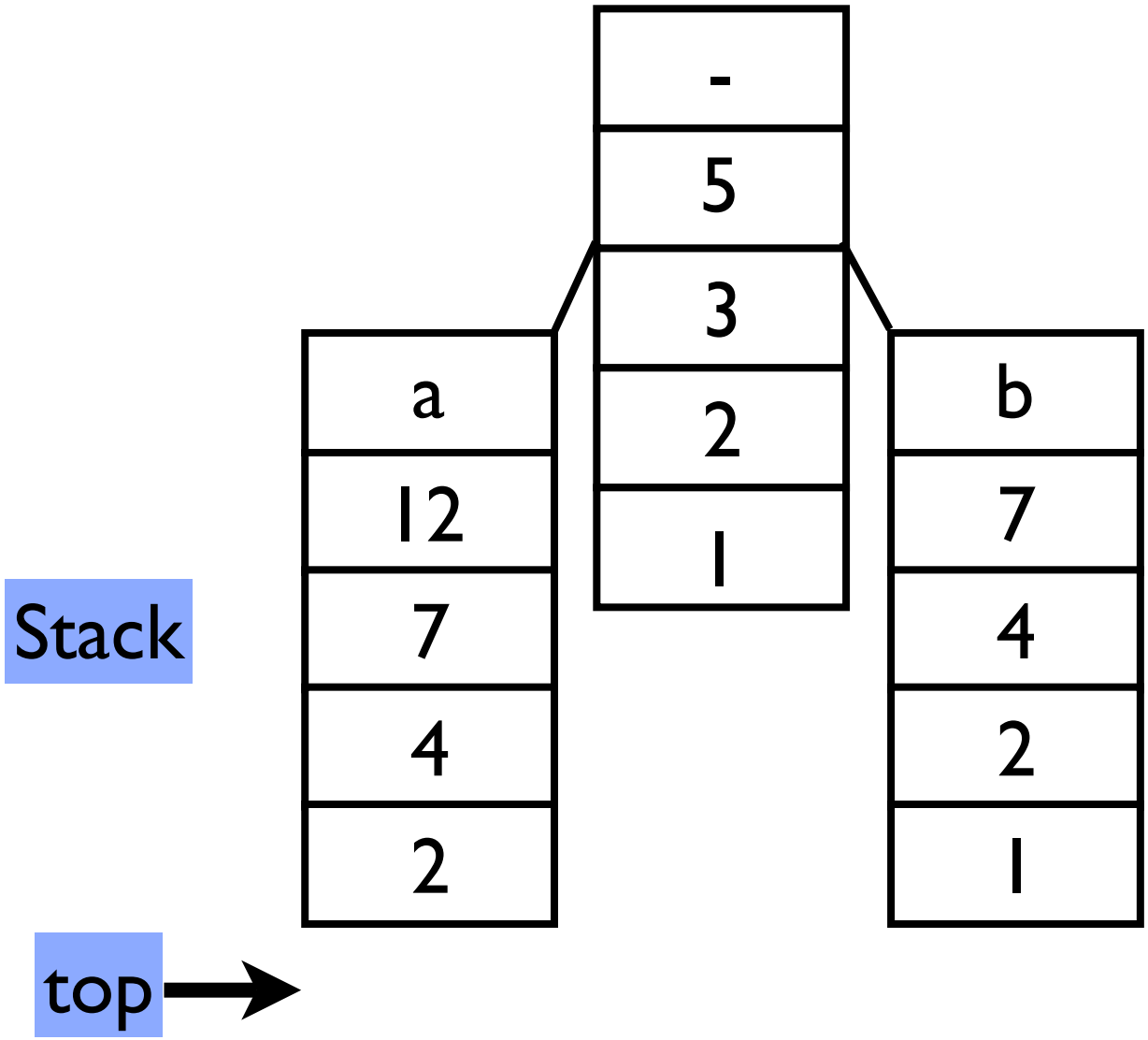
Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



Implementierung von Datent.

Rekursion im Kontrollbereich - Beispiel



Implementierung von Datent.
Rekursion - praktisches Vorgehen

Implementierung einer rekursiven Funktion:

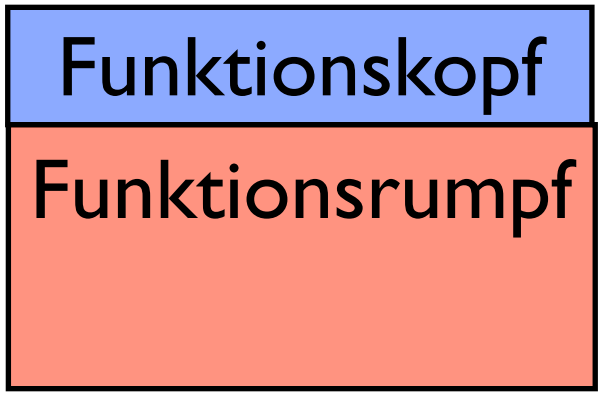
- entferne geschachtelte Funktionsaufrufe:

  f(g(x)) → h:=g(x); f(h)

- Rumpf der Funktion an Programmanfang
- zu Beginn Marke X als Sprungziel
- weitere Marke Y unmittelbar hinter dem Rumpf
- vor den Rumpf ein Sprung auf Y

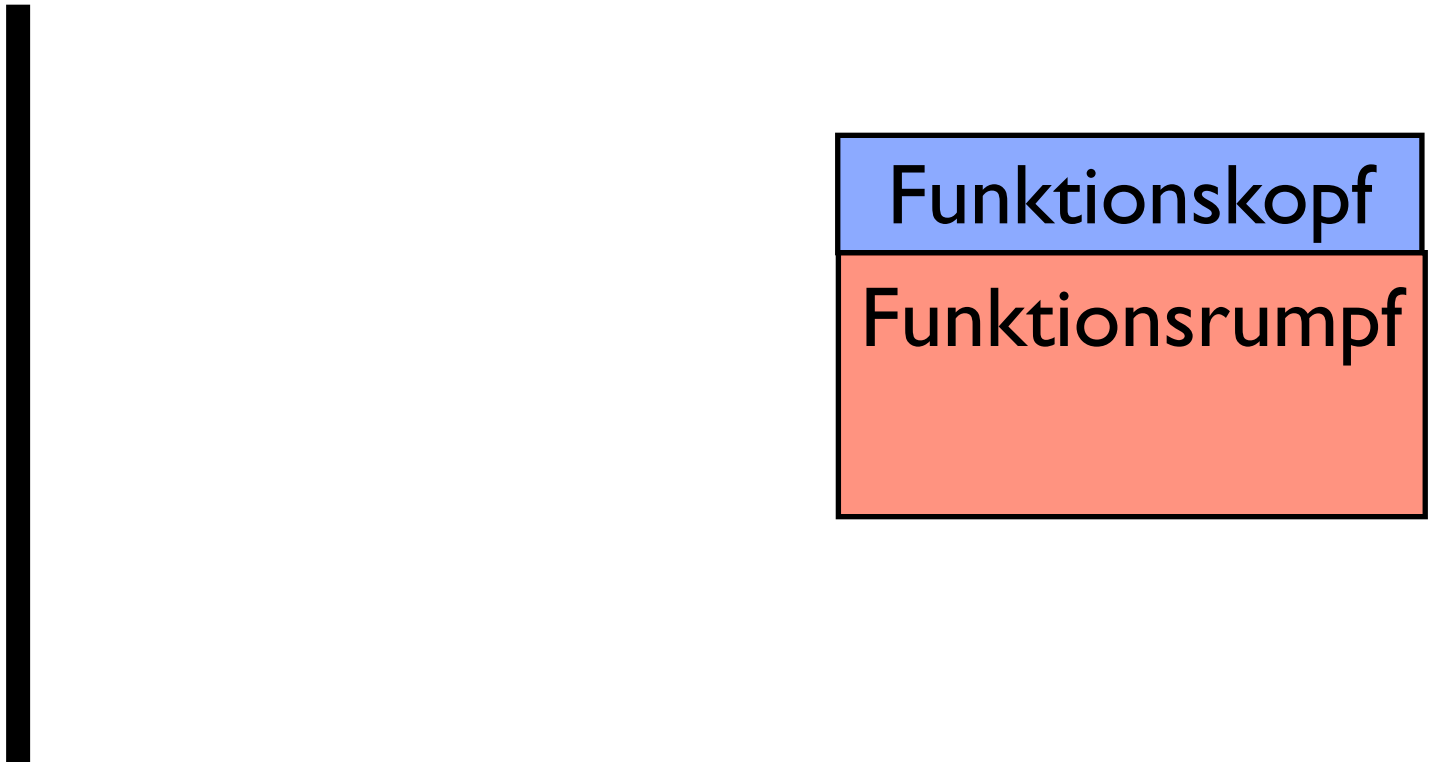
Implementierung von Datent.

Rekursion - praktisches Vorgehen



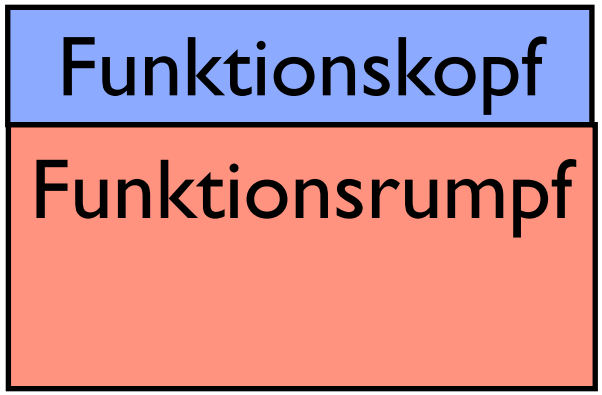
Implementierung von Datent.

Rekursion - praktisches Vorgehen



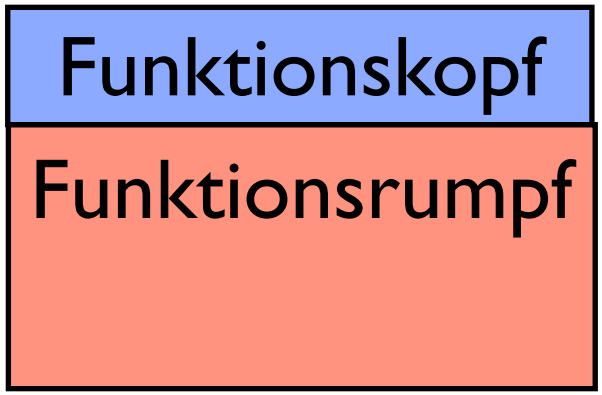
Implementierung von Datent.

Rekursion - praktisches Vorgehen



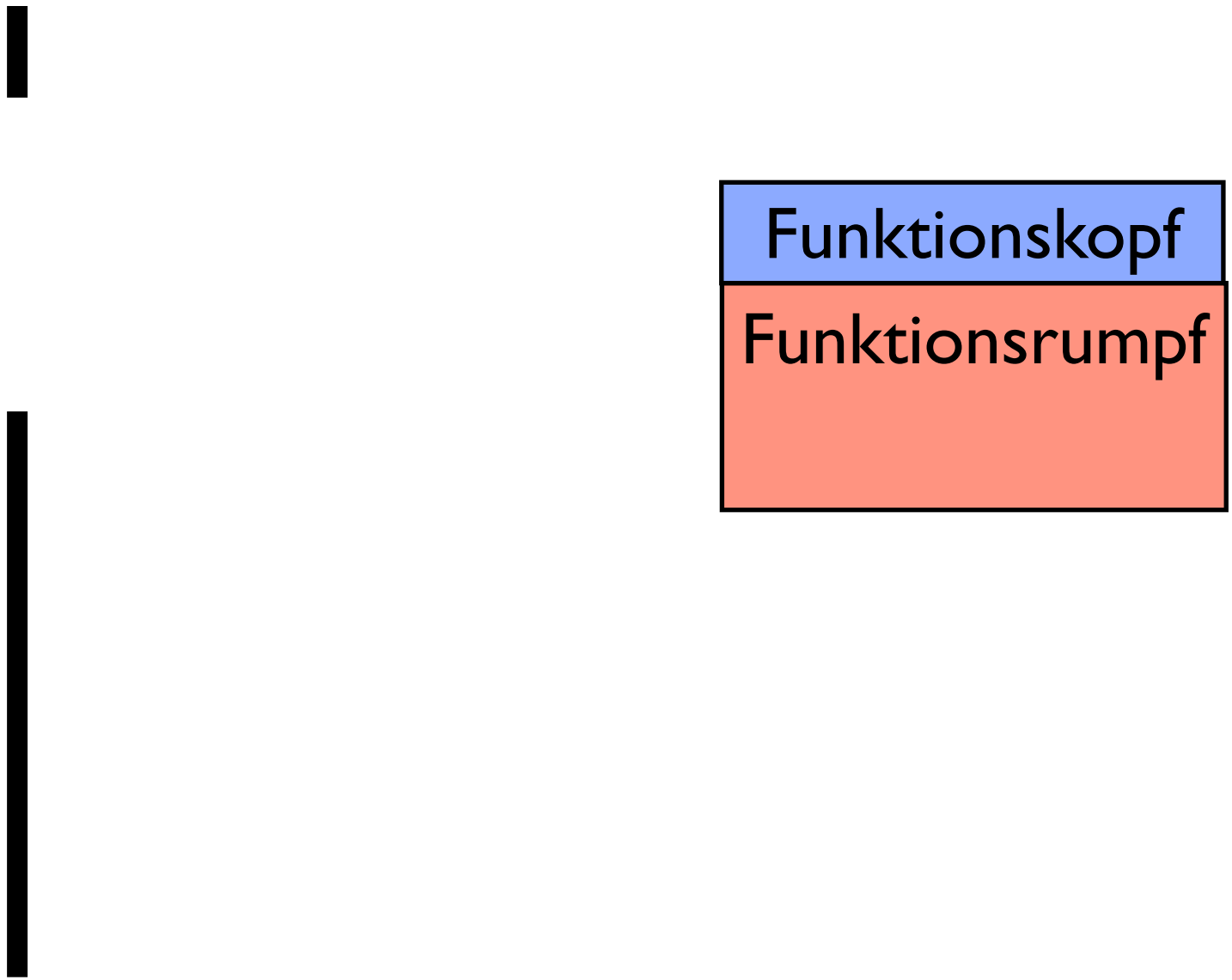
Implementierung von Datent.

Rekursion - praktisches Vorgehen



Implementierung von Datent.

Rekursion - praktisches Vorgehen



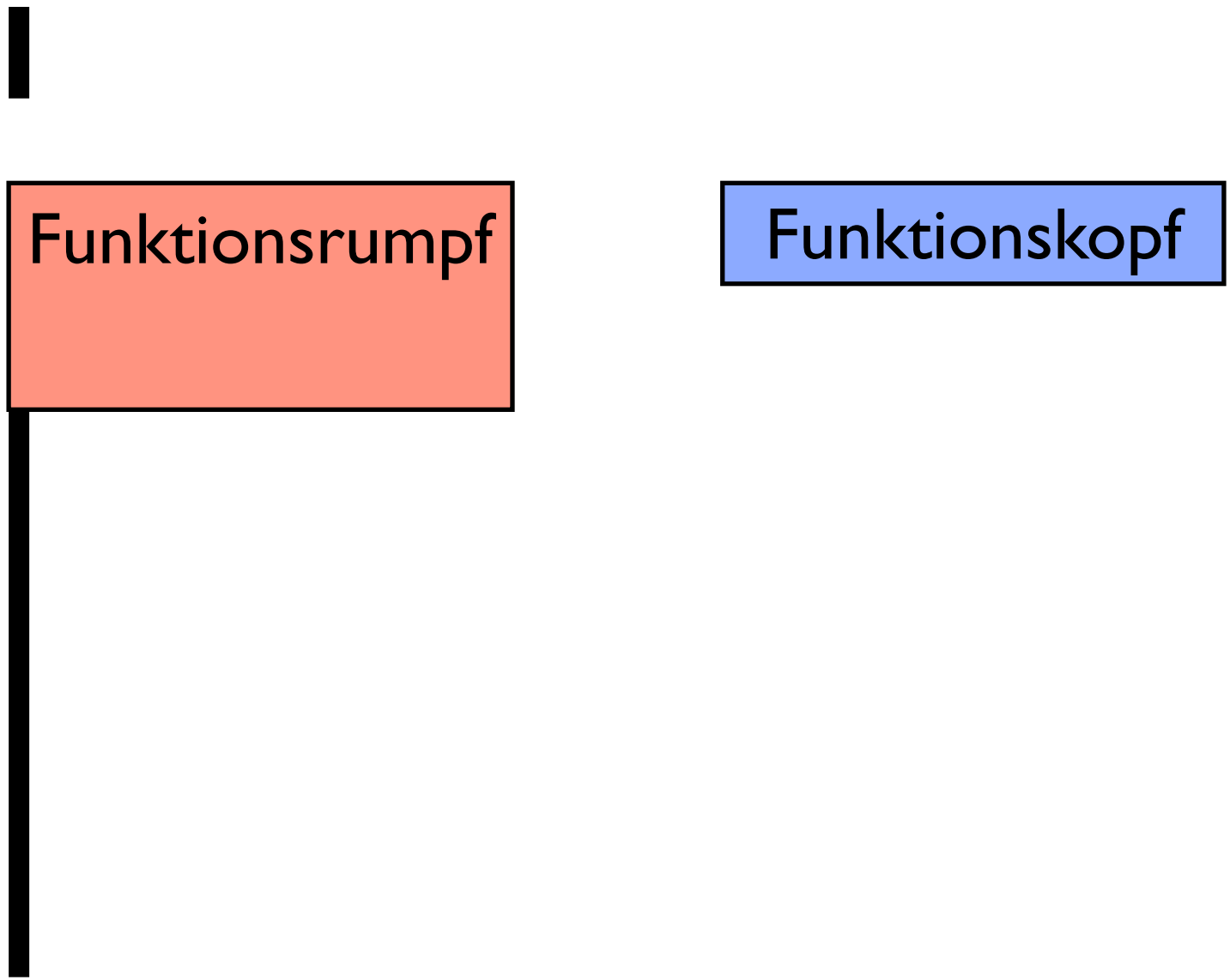
Implementierung von Datent.

Rekursion - praktisches Vorgehen

Funktionskopf

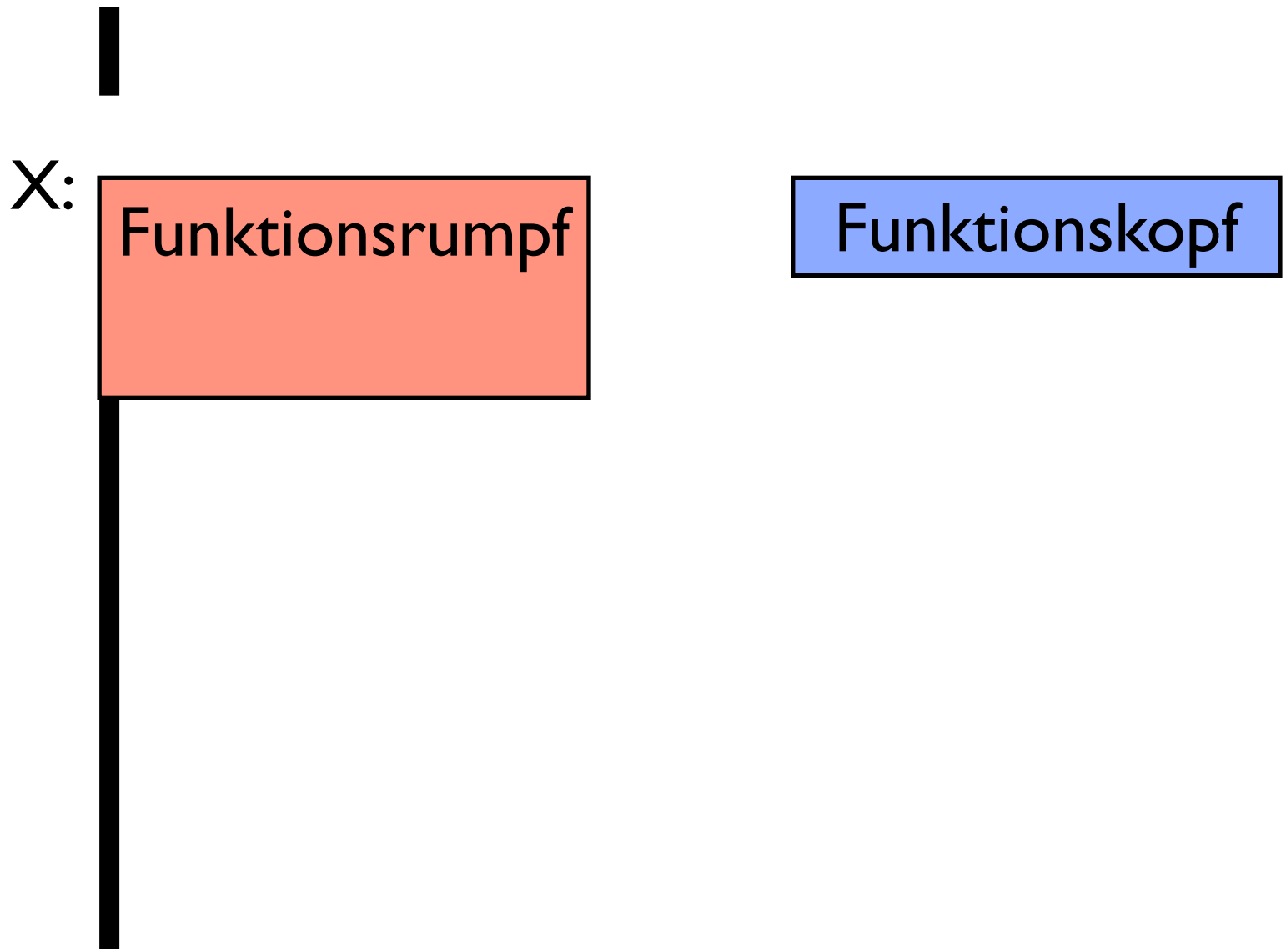
Implementierung von Datent.

Rekursion - praktisches Vorgehen



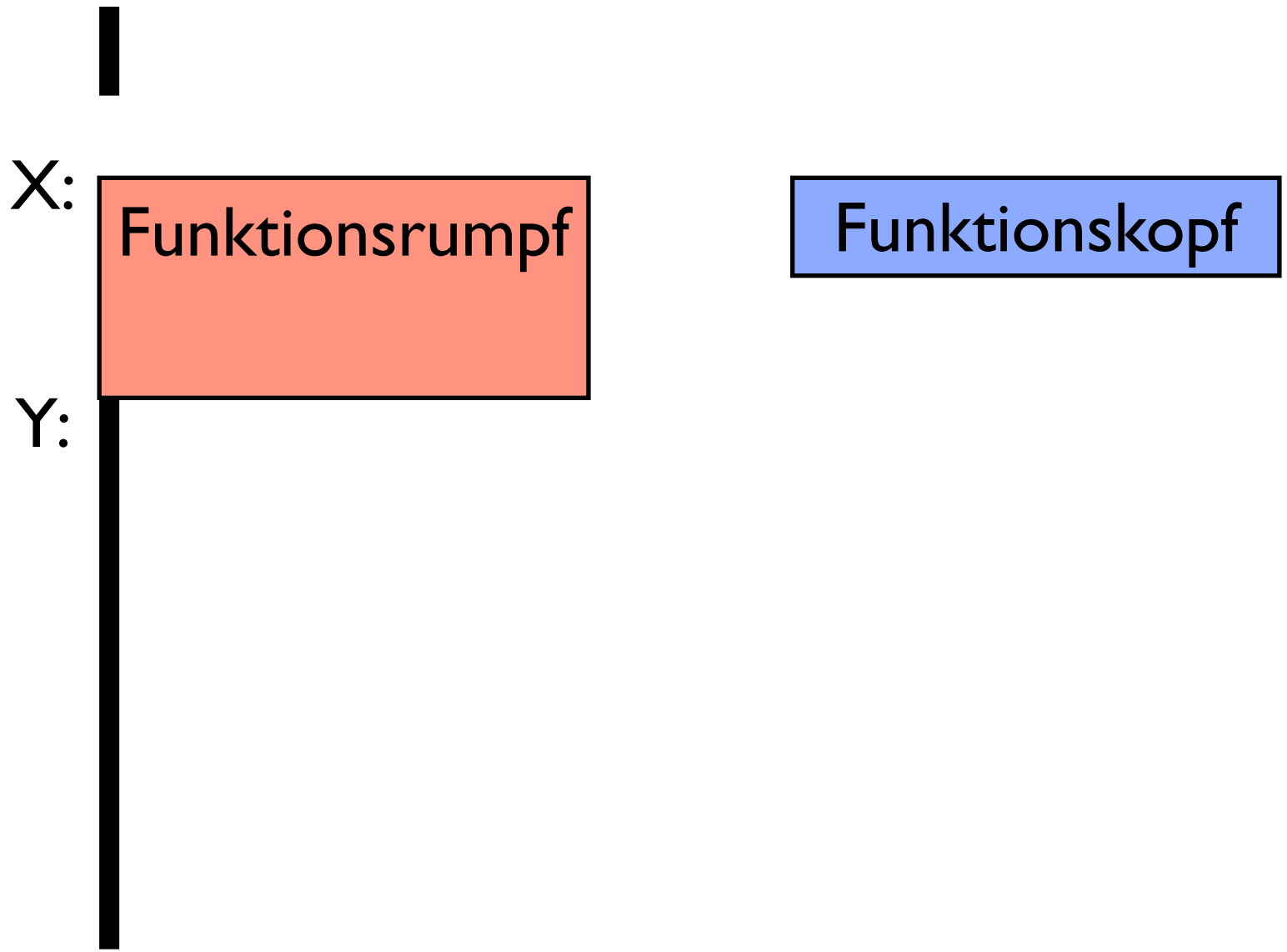
Implementierung von Datent.

Rekursion - praktisches Vorgehen



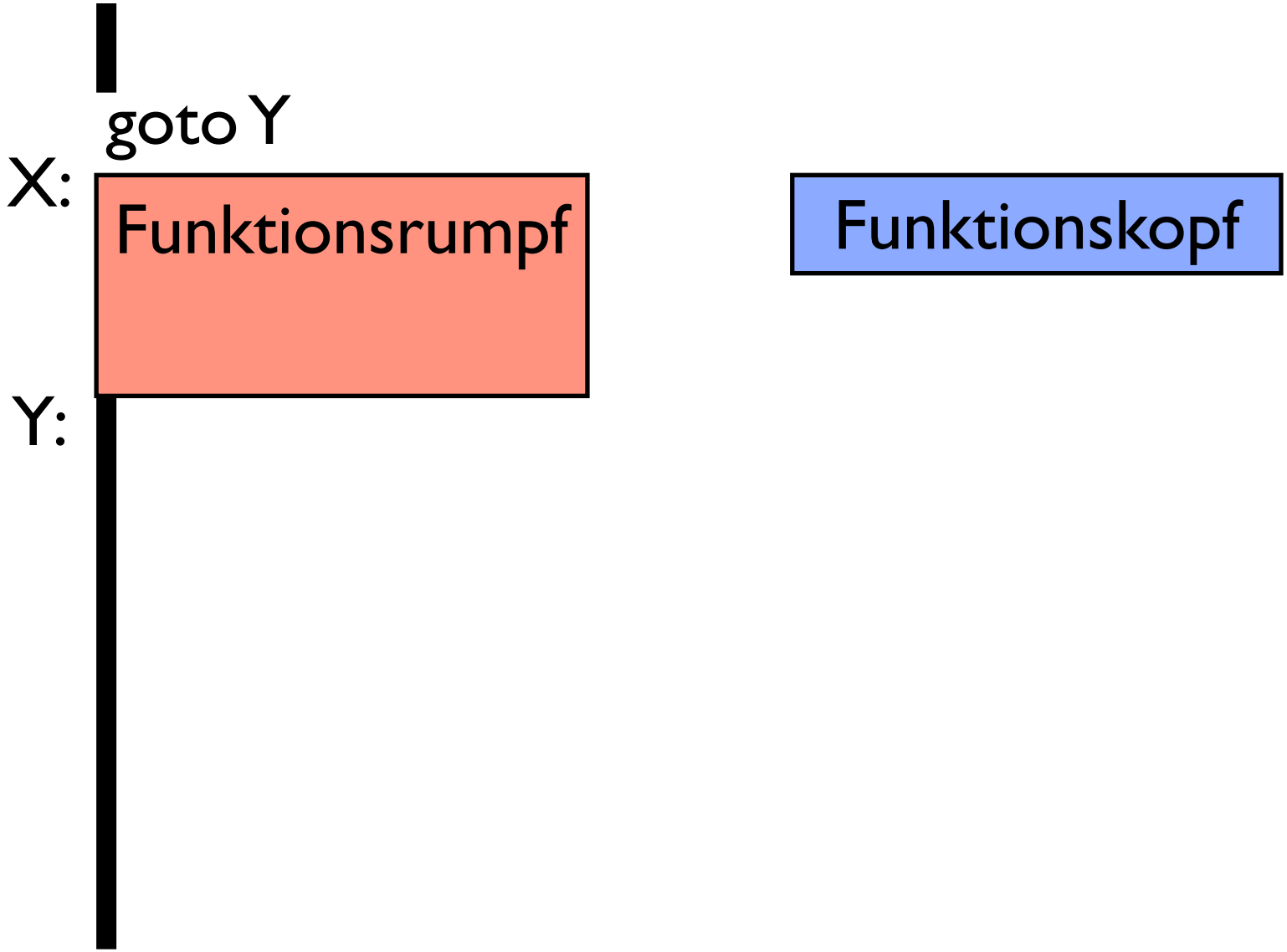
Implementierung von Datent.

Rekursion - praktisches Vorgehen



Implementierung von Datent.

Rekursion - praktisches Vorgehen



Implementierung von Datent.
Rekursion - praktisches Vorgehen

- Definiere Stack: je Stackeintrag ein Record mit
  - allen Parametern
  - allen lokalen Variablen
  - Variable für den berechneten Funktionswert einer Inkarnation
  - Rücksprungadresse, zu der nach Abarbeitung der Funktion verzweigt werden muß (direkt hinter der Aufrufstelle).
- Der Funktionsrumpf arbeitet nur mit dem obersten Stackelement. Eigene Variablen oder Parameter besitzt er nicht.

Implementierung von Datent.

Rekursion - praktisches Vorgehen

Beispiel: ggT

```
type data=record
                    a,b: integer;
                    marke: (M1,M2,M3);
                    ggT: integer
                end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                    a,b: integer;
                    marke: (M1,M2,M3);
                    ggT: integer
                end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                a,b: integer;
                marke: (M1,M2,M3);
                ggT: integer
              end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                    a,b: integer;
                    marke: (M1,M2,M3);
                    ggT: integer
                end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                a,b: integer;
                marke: (M1,M2,M3);
                ggT: integer
            end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                    a,b: integer;
                    marke: (M1,M2,M3);
                    ggT: integer
                end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                a,b: integer;
                marke: (M1,M2,M3);
                ggT: integer
              end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                    a,b: integer;
                    marke: (M1,M2,M3);
                    ggT: integer
                end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                    a,b: integer;
                    marke: (M1,M2,M3);
                    ggT: integer
                end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                    a,b: integer;
                    marke: (M1,M2,M3);
                    ggT: integer
                end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                    a,b: integer;
                    marke: (M1,M2,M3);
                    ggT: integer
                end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

```
funktion ggT (a:nat,b:nat) → nat ≡
wenn a=b dann a sonst
    wenn a<b dann ggT (b,a) sonst ggT (a-b,b) ende ende.
```

Beispiel: ggT

```
type data=record
                    a,b: integer;
                    marke: (M1,M2,M3);
                    ggT: integer
                end.
type rekstack = stack of data
```

Implementierung von Datent.
Rekursion - praktisches Vorgehen

Funktionsaufrufe ersetzen:

- Zuweisung der Parameter und Rücksprungadresse an neues Record-Element
- push-Operation des Records auf den Stack
- Sprung zur Anfangsmarke des Funktionsrumpfs
- Einführung einer Rücksprungmarke.

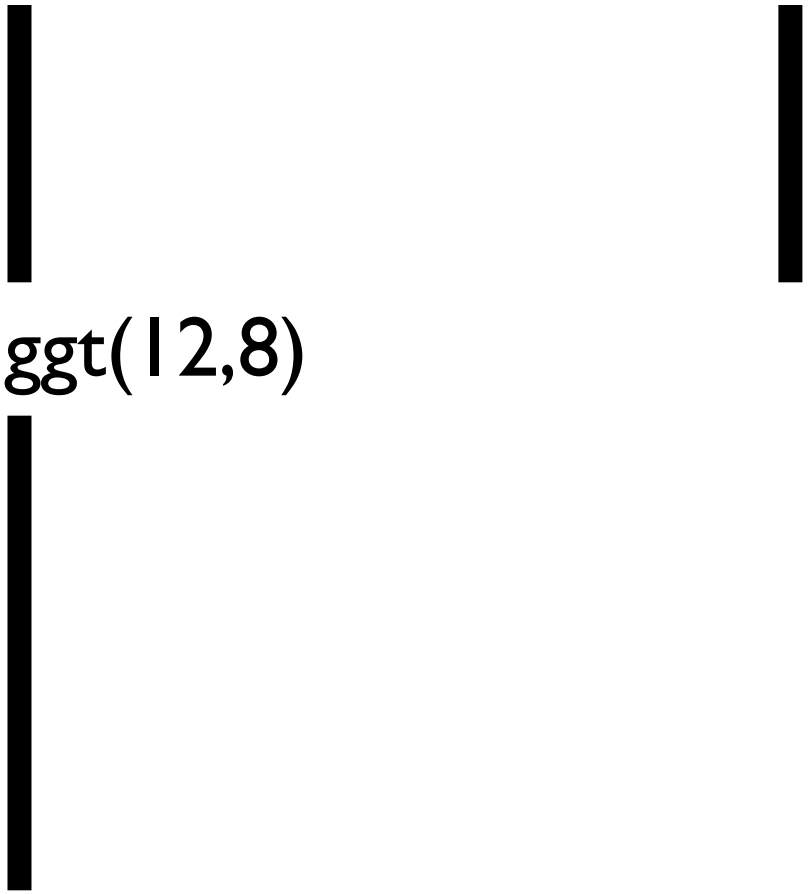
Implementierung von Datent.

Rekursion - praktisches Vorgehen

ggt(12,8)

Implementierung von Datent.

Rekursion - praktisches Vorgehen

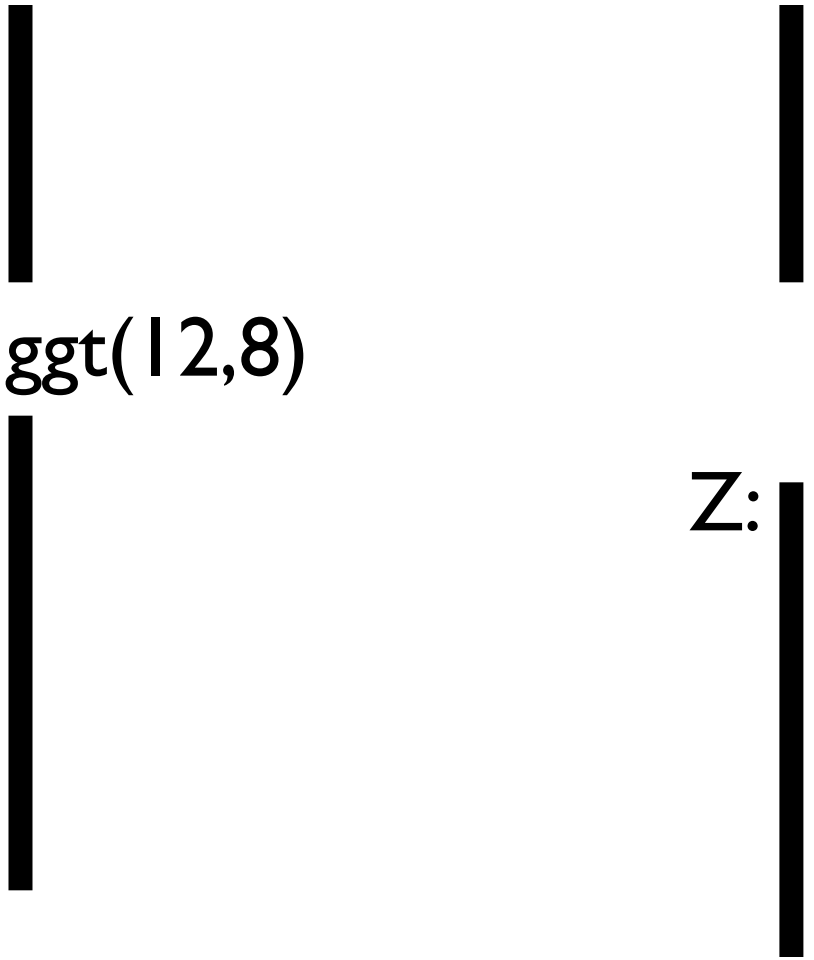


Implementierung von Datent.

Rekursion - praktisches Vorgehen

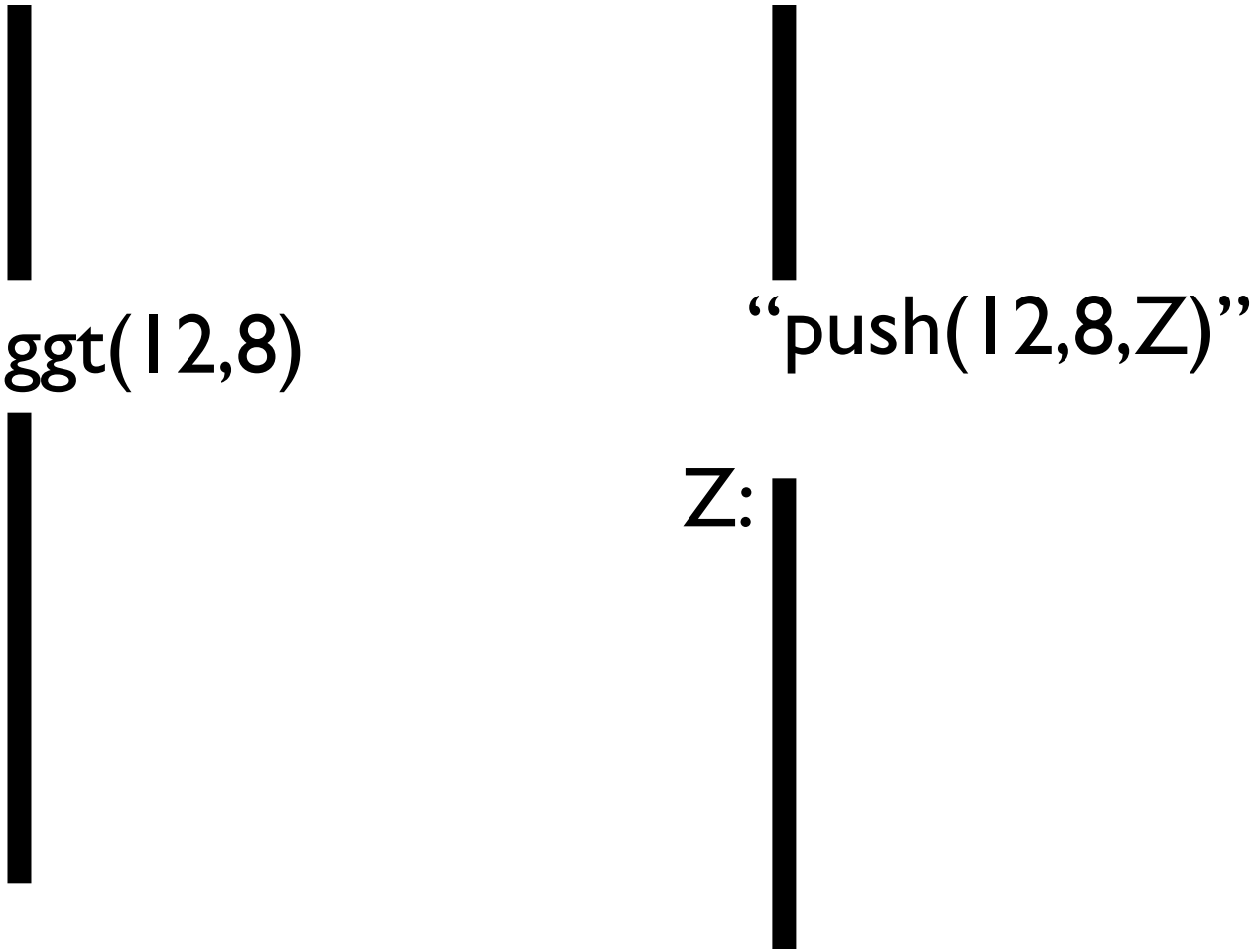
Implementierung von Datent.

Rekursion - praktisches Vorgehen



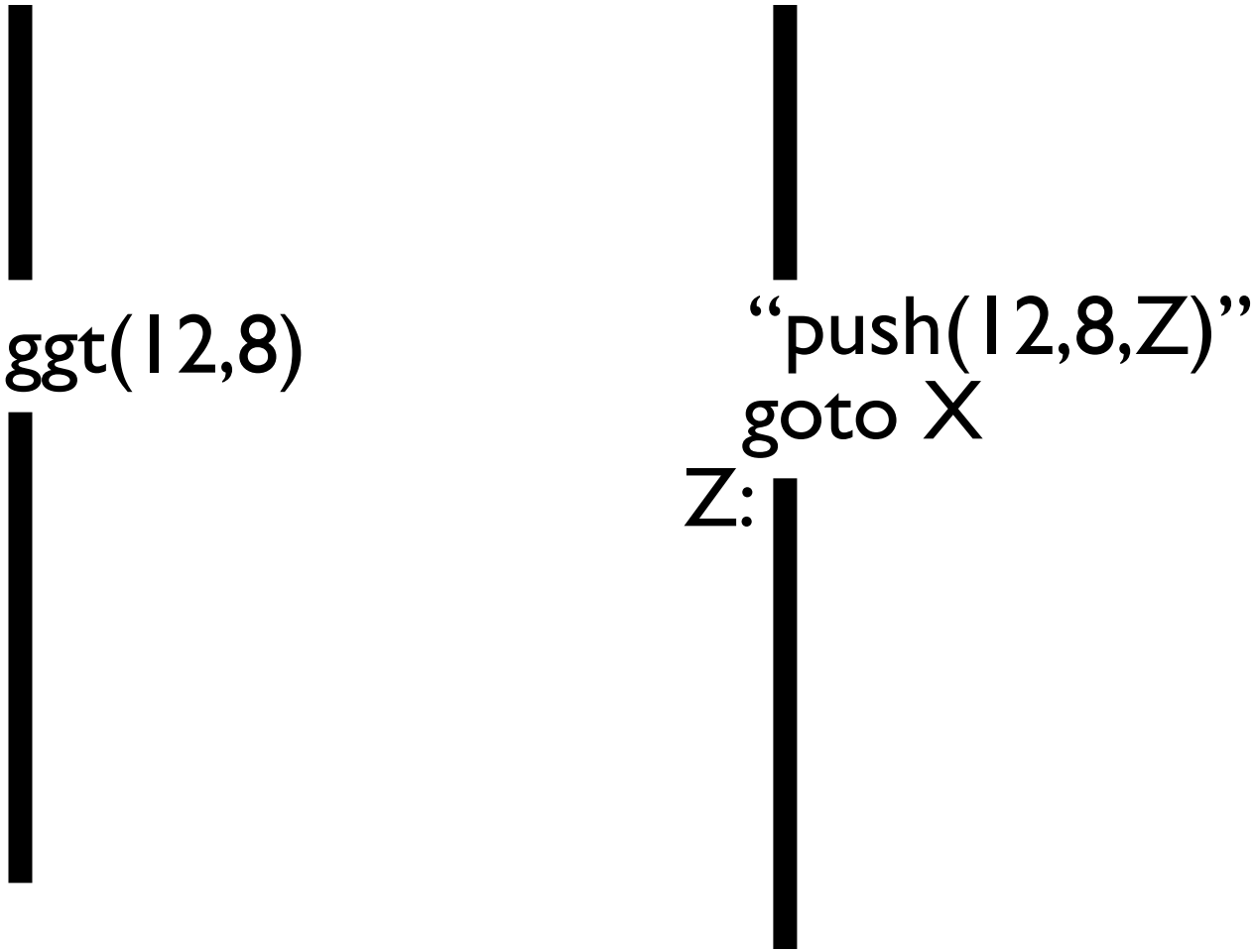
Implementierung von Datent.

Rekursion - praktisches Vorgehen



Implementierung von Datent.

Rekursion - praktisches Vorgehen



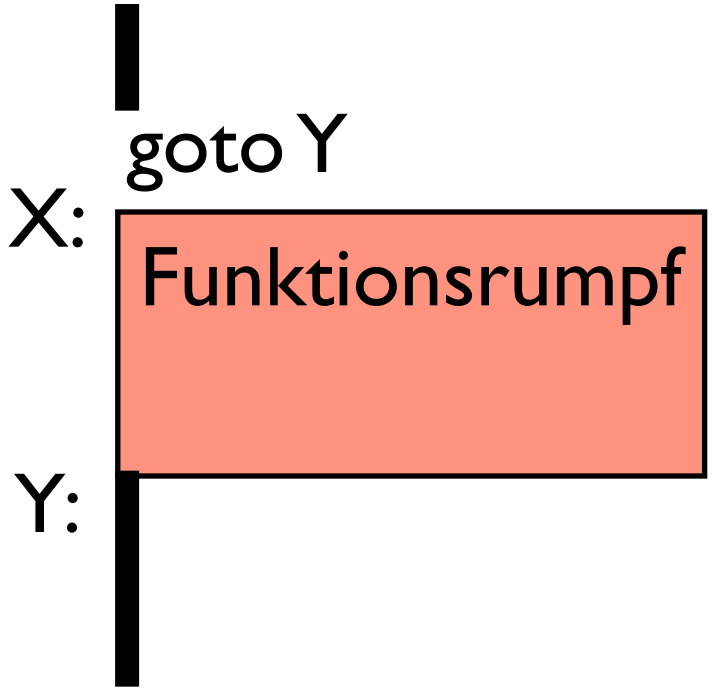
Implementierung von Datent.
Rekursion - praktisches Vorgehen

Funktionsende modifizieren:

- Auslesen der Rücksprungadresse aus oberstem Stackelement
- Auslesen des Funktionswertes aus oberstem Stackelement
- Entferne oberstes Stackelementes
- Weitergabe des Funktionswertes an jetzt oberste Stackelement
- Sprung zur Rücksprungadresse.

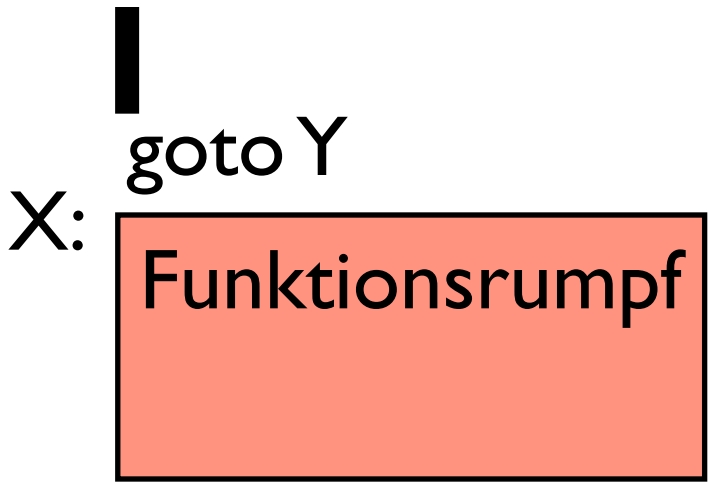
Implementierung von Datent.

Rekursion - praktisches Vorgehen



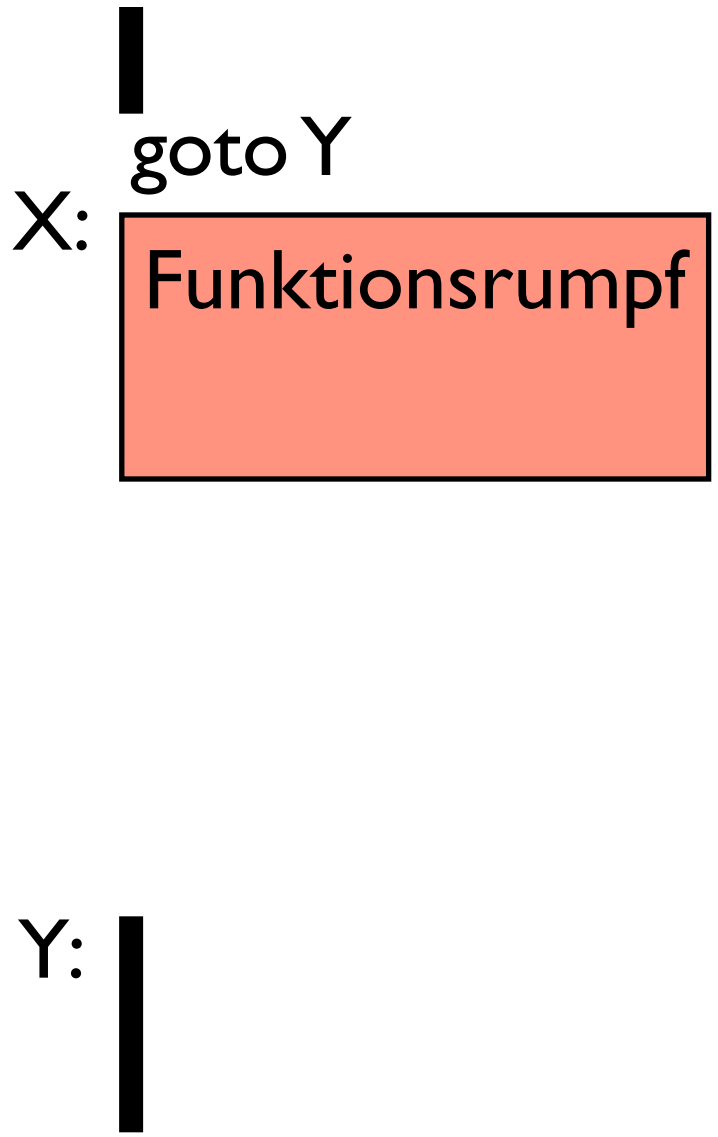
Implementierung von Datent.

Rekursion - praktisches Vorgehen



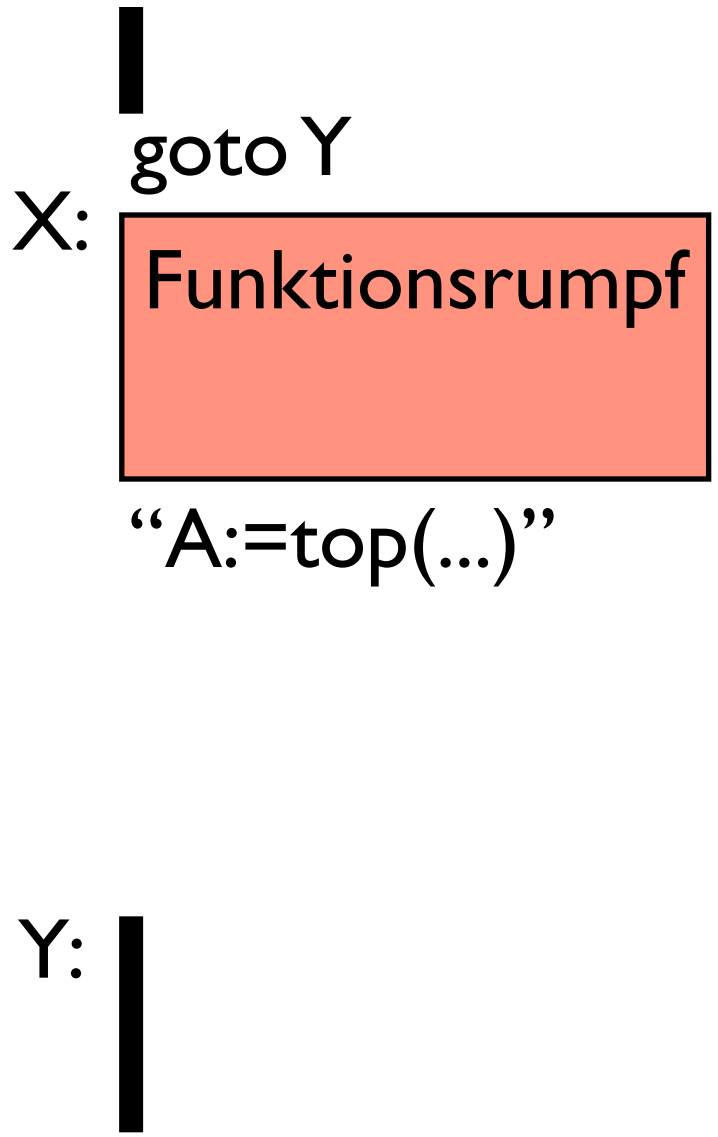
Implementierung von Datent.

Rekursion - praktisches Vorgehen



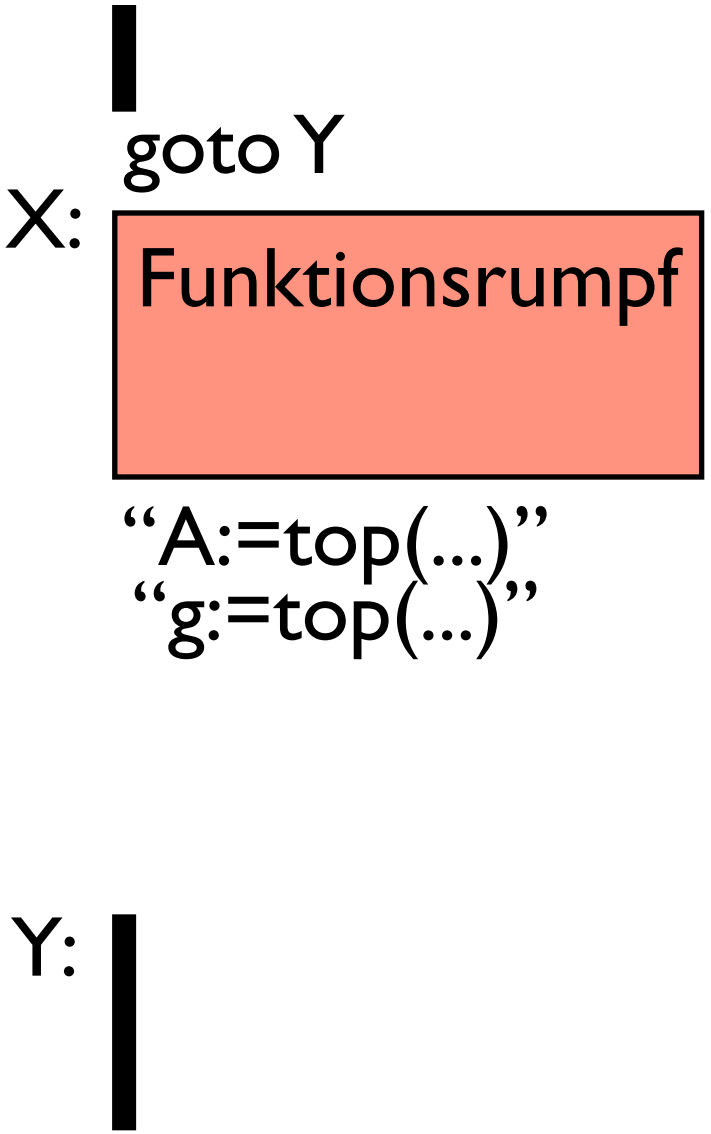
Implementierung von Datent.

Rekursion - praktisches Vorgehen



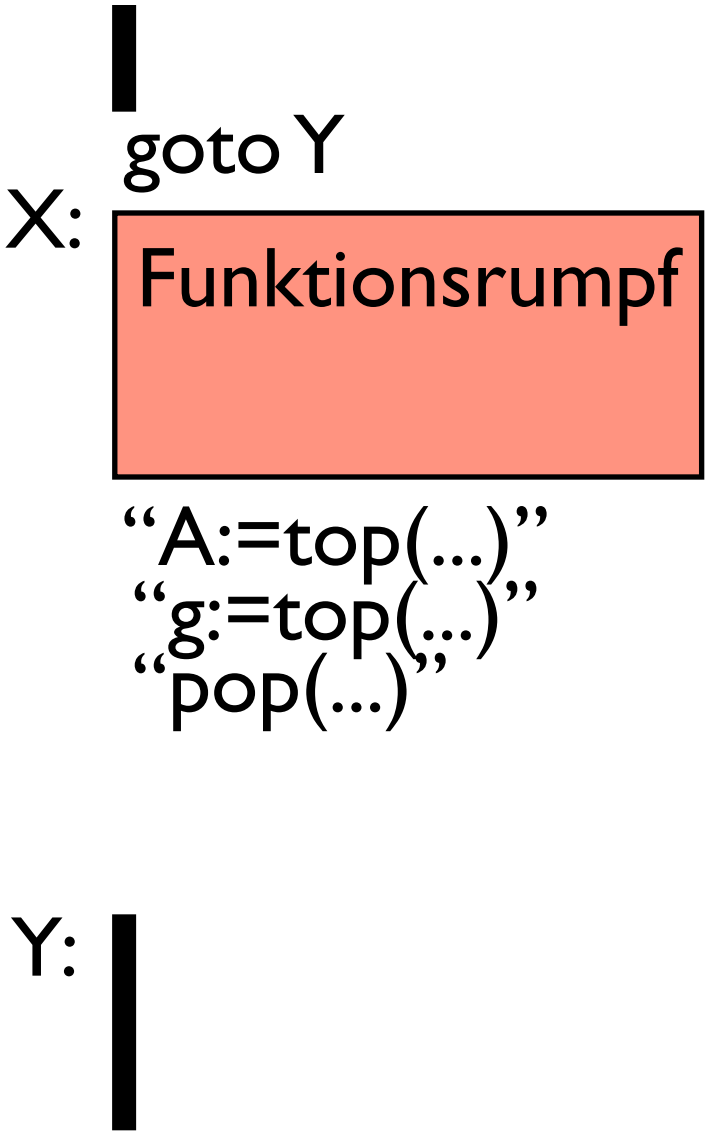
Implementierung von Datent.

Rekursion - praktisches Vorgehen



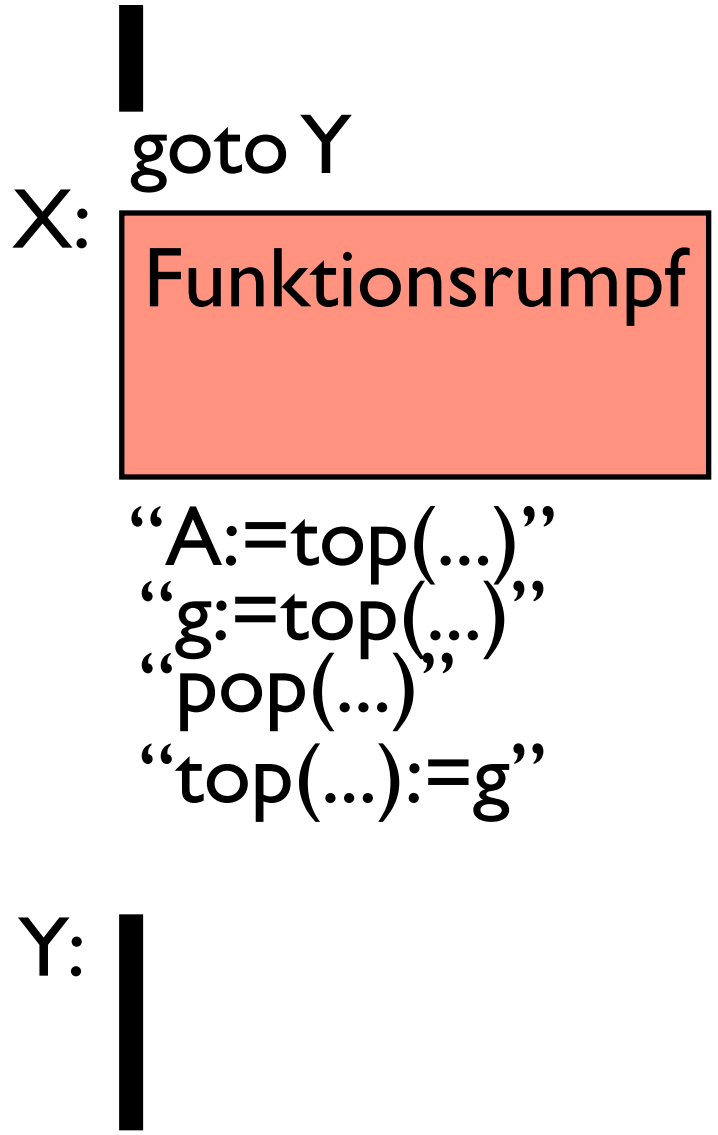
Implementierung von Datent.

Rekursion - praktisches Vorgehen



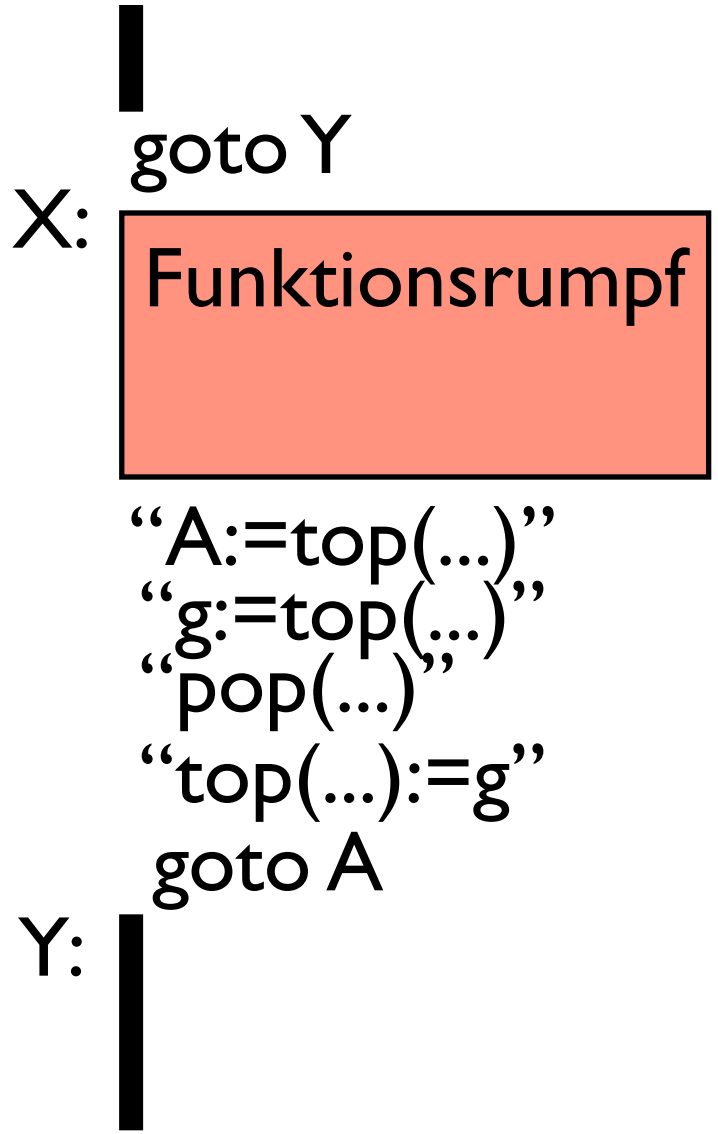
Implementierung von Datent.

Rekursion - praktisches Vorgehen



Implementierung von Datent.

Rekursion - praktisches Vorgehen



Implementierung von Datent.
Rekursion und Iteration

"**Satz**"

Zu jedem rekursiven Algorithmus gibt es einen äquivalenten nicht-rekursiven Algorithmus (mit Stack) und umgekehrt.

Die Maschine realisiert Rekursion immer nicht-rekursiv und verwaltet den Stack automatisch.

Implementierung von Datent.
Rekursion - der Stack

- Stack: kritische Größe (reicht er aus?)
- erwartete Rekursionstiefe und -breite analysieren: paßt sie zur Problemgröße? (genaue Analyse später)
- Negativbeispiel:
  - Fibonacci-Funktion

    $f(0)=0, f(1)=1,$

    $f(n)=f(n-1)+f(n-2)$ für $n \geq 2$
  - mit n exponentiell wachsender Stackbedarf
  - es gibt einfache nicht-rekursive Lösung (o. Stack)

Implementierung von Datent.
Rekursion - der Stack

- Wann geht es ohne Stack?
- Rekursive Funktionen, bei denen der rekursive Aufruf stets am Anfang (*head recursion*) oder stets am Ende (*tail recursion*) des Rumpfs erfolgt, können ohne Stack implementiert werden

Implementierung von Datent.

Rekursion - der Stack

Funktionsschema:

funktion P x → ... ≡
wenn B dann a sonst g(x,P(f(x))) ende

Implementierung:

```
P:=a; j:=y;
while not B do
begin
    P:=g(j,P);
    j:=f(j)
end.
```

Implementierung von Datent.

Rekursion - der Stack

Funktionsschema:

funktion P x → ... ≡

wenn B dann a sonst g(x,P(f(x))) ende

Implementierung:

```
P:=a; j:=y;
while not B do
begin
    P:=g(j,P);
    j:=f(j)
end.
```

Implementierung von Datent.

Rekursion - der Stack

Funktionsschema:

funktion P x → ... ≡

wenn B dann a sonst g(x,P(f(x))) ende

g kommutativ

Implementierung:

```
P:=a; j:=y;
while not B do
begin
    P:=g(j,P);
    j:=f(j)
end.
```

Beispiel im Skript:
Fakultät

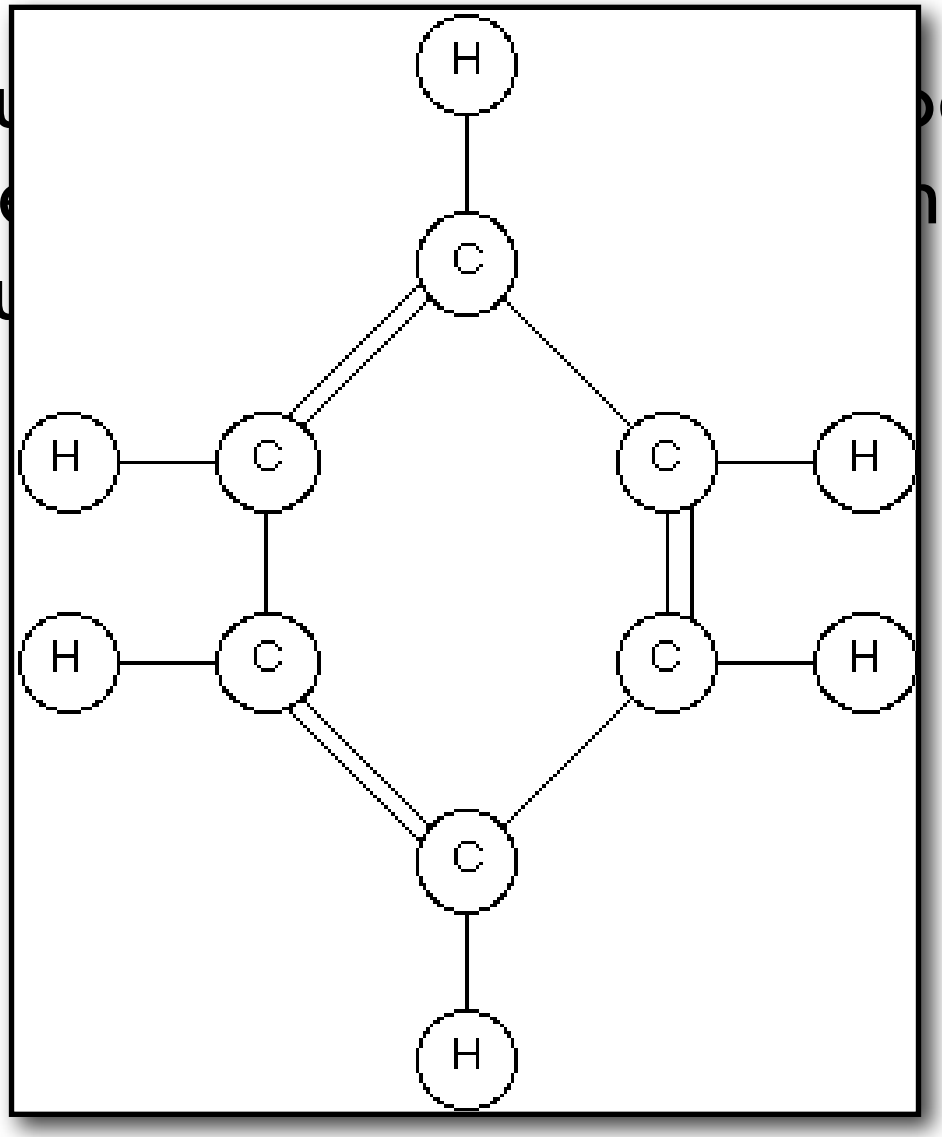
Implementierung von Datent.
Graphen

- anschauliches mathematisches Modell zur Beschreibung von Objekten, die in gewisser Beziehung stehen

Implementierung von Datent.

Graphen

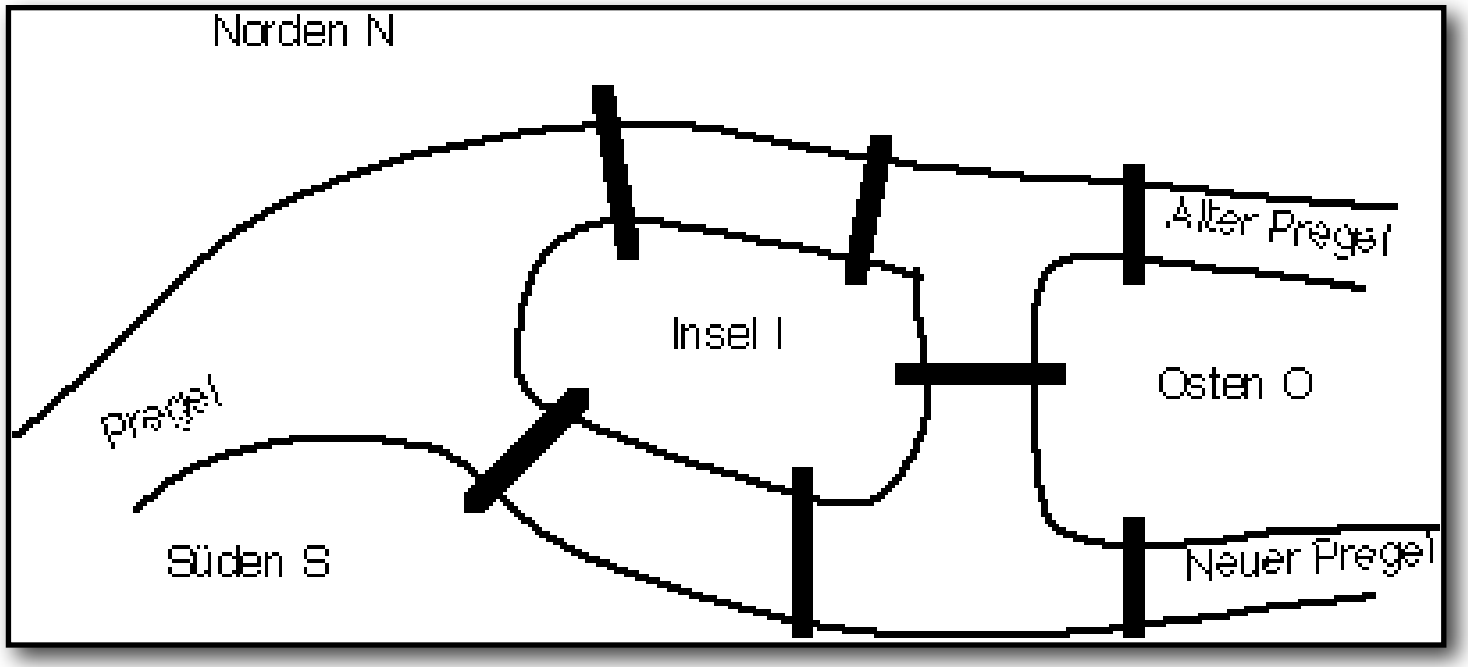
- anschau odell zur Beschre n gewisser Beziehu

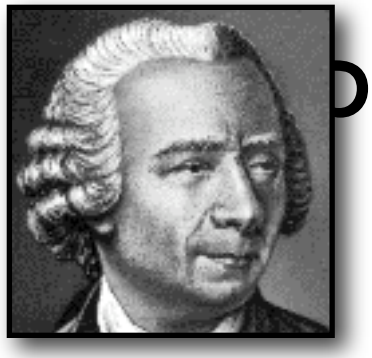


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

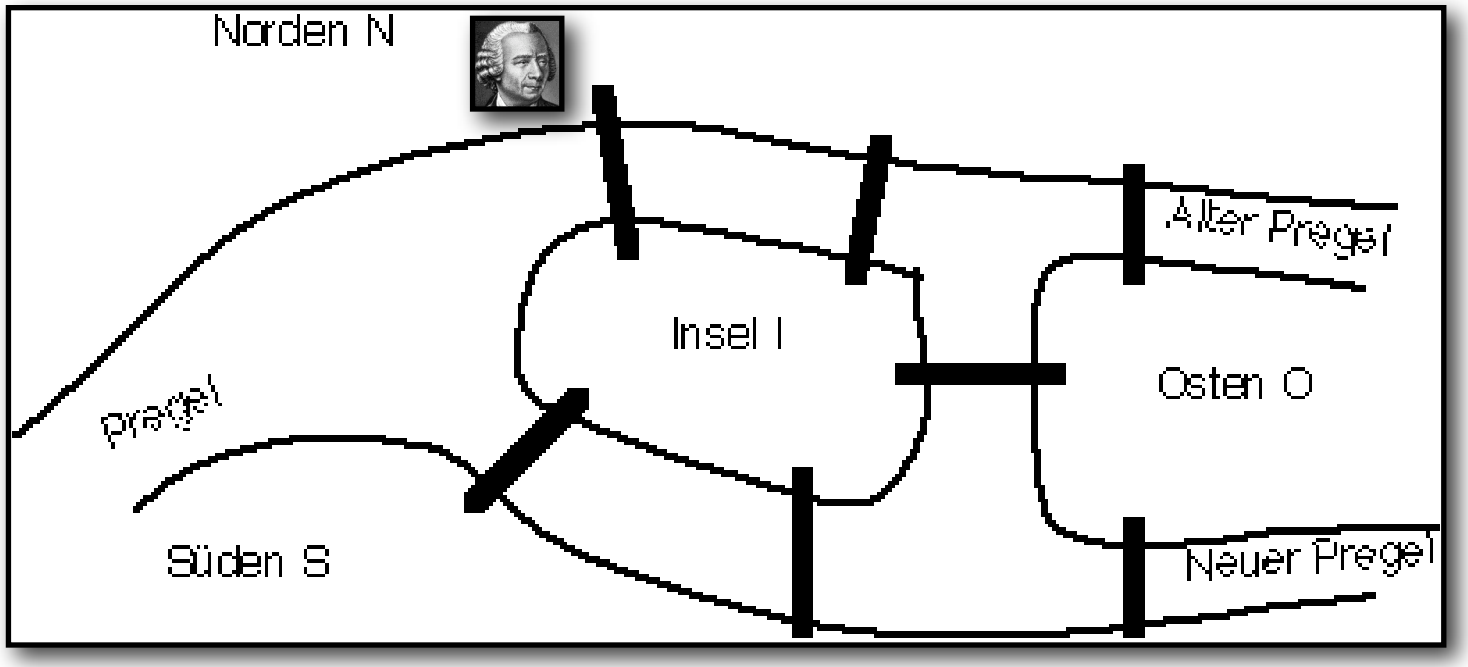




blementierung von Datent.

Graphen

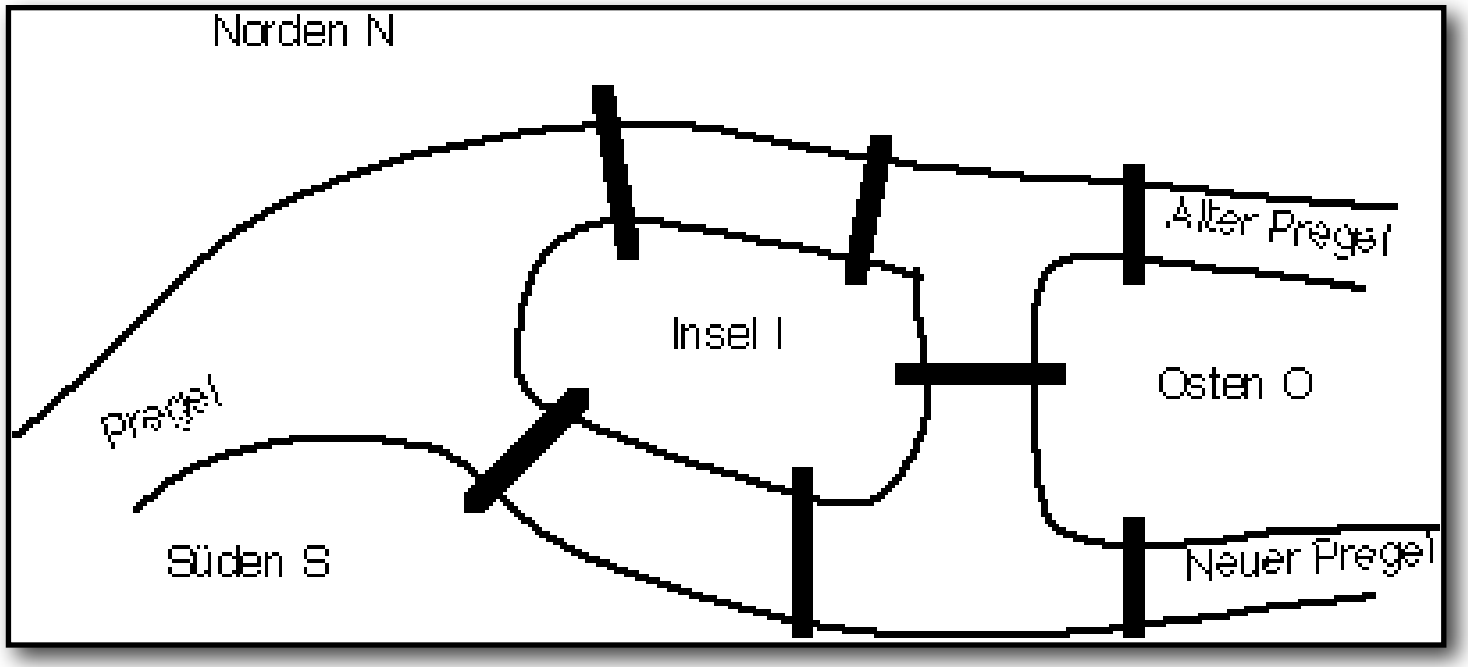
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

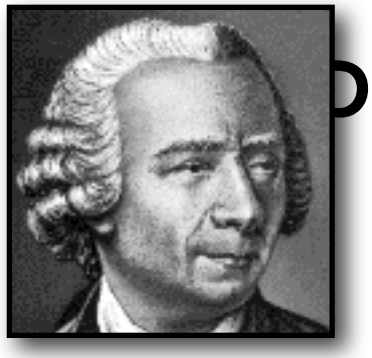


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

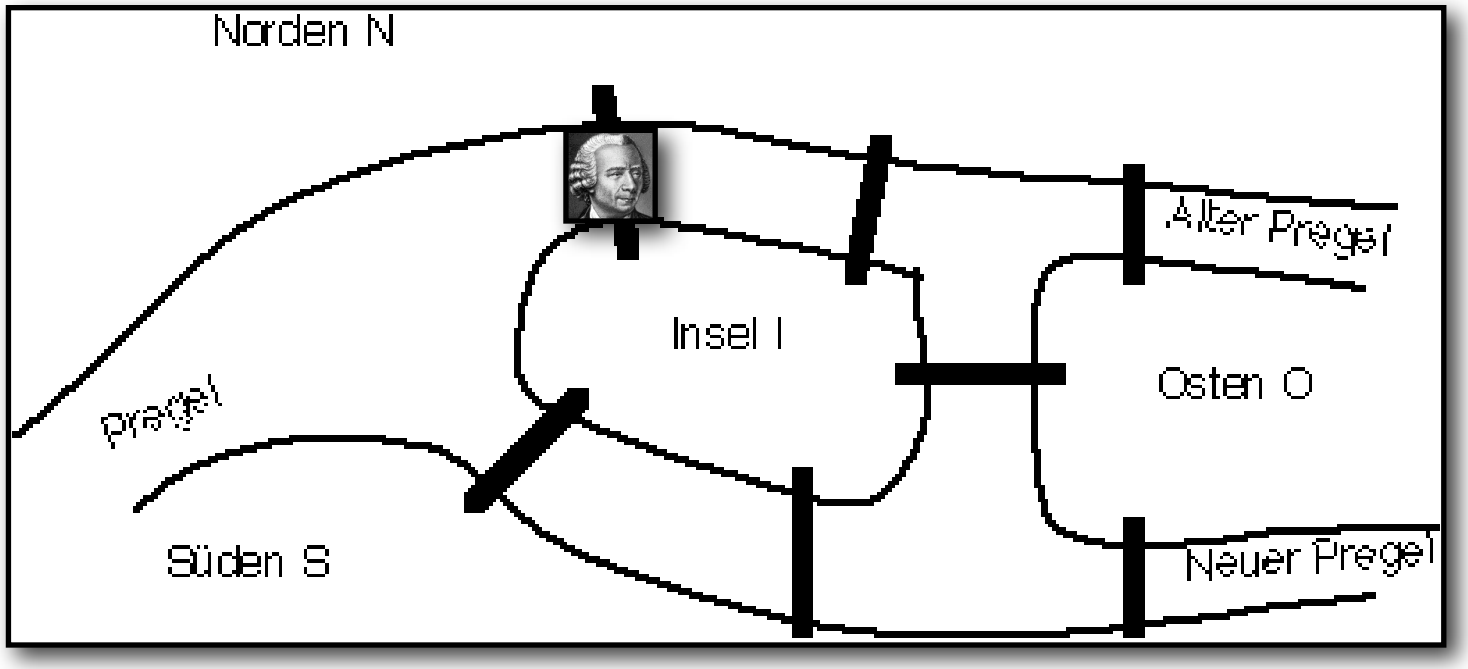




blementierung von Datent.

Graphen

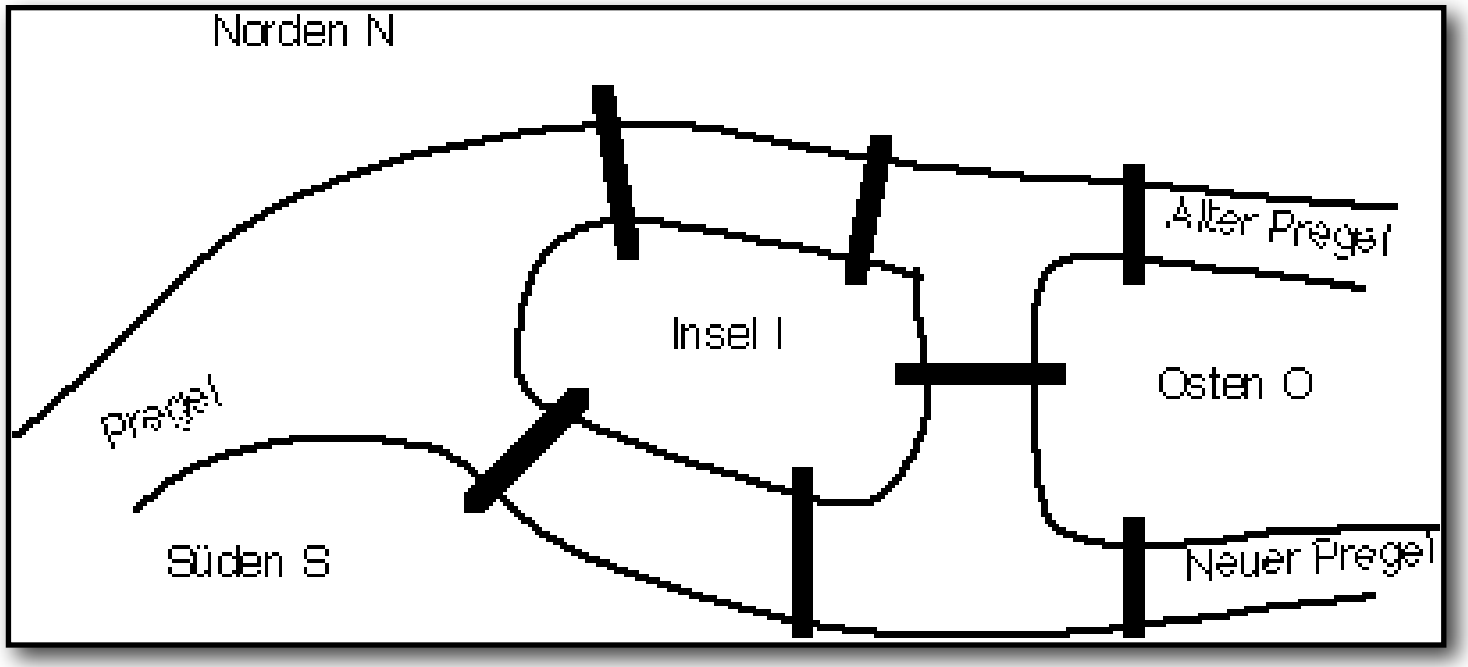
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

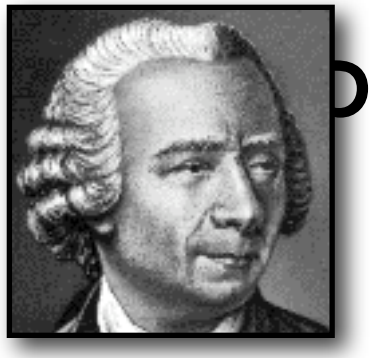


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

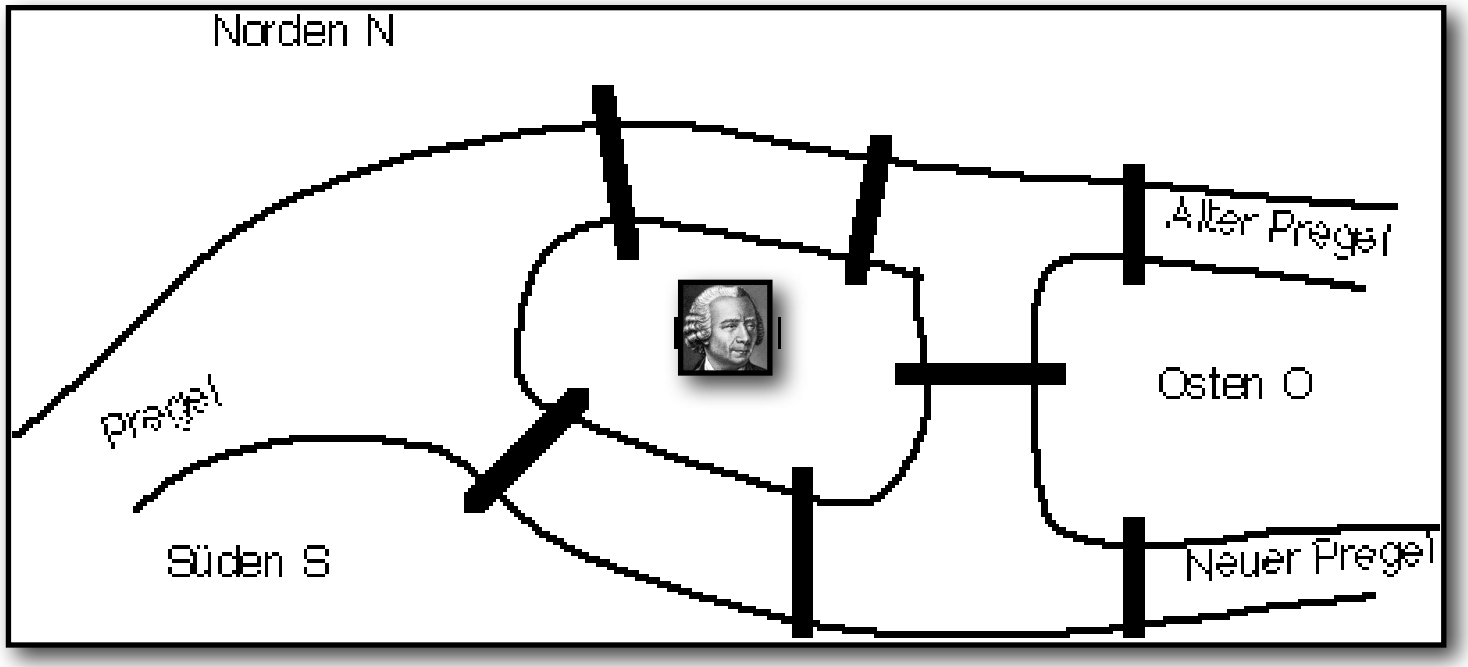




blementierung von Datent.

Graphen

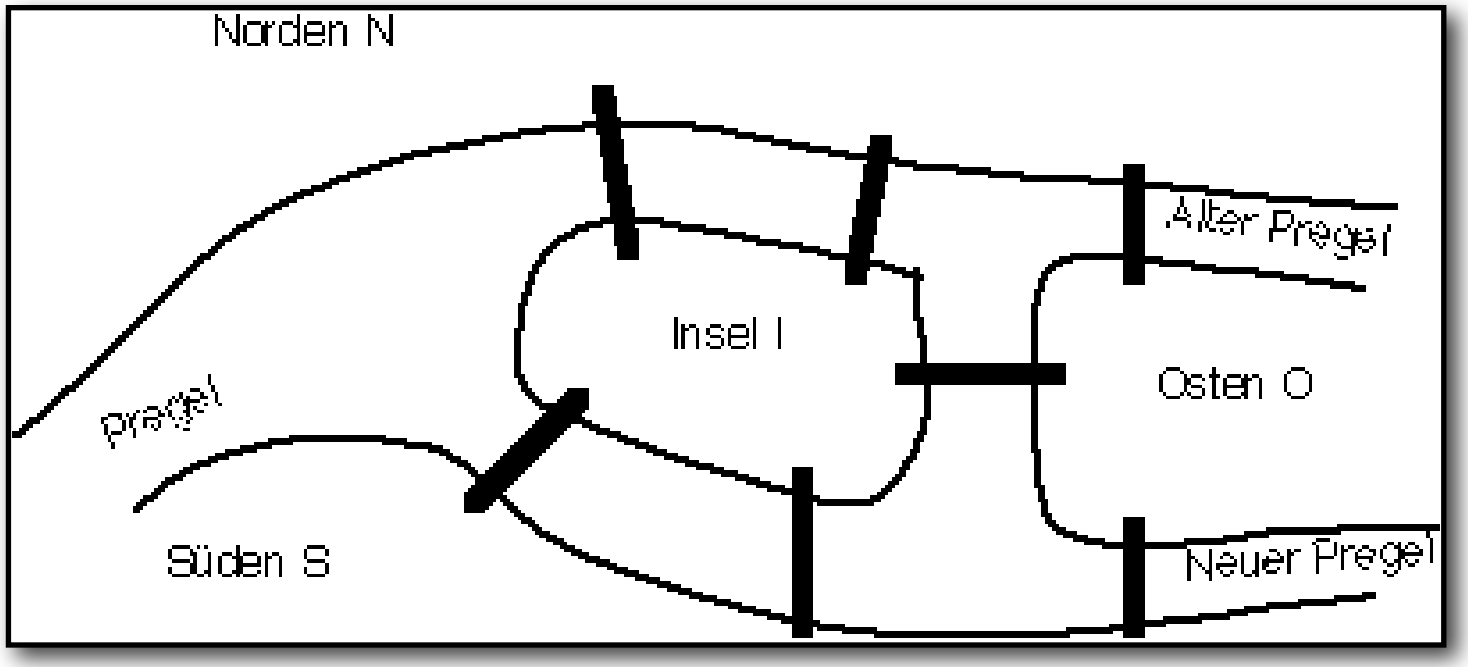
- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m

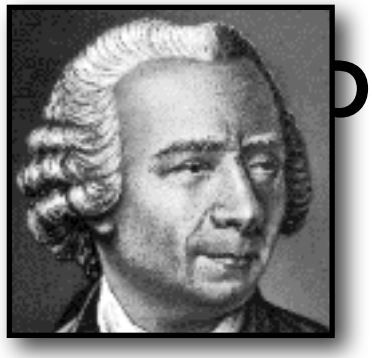


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

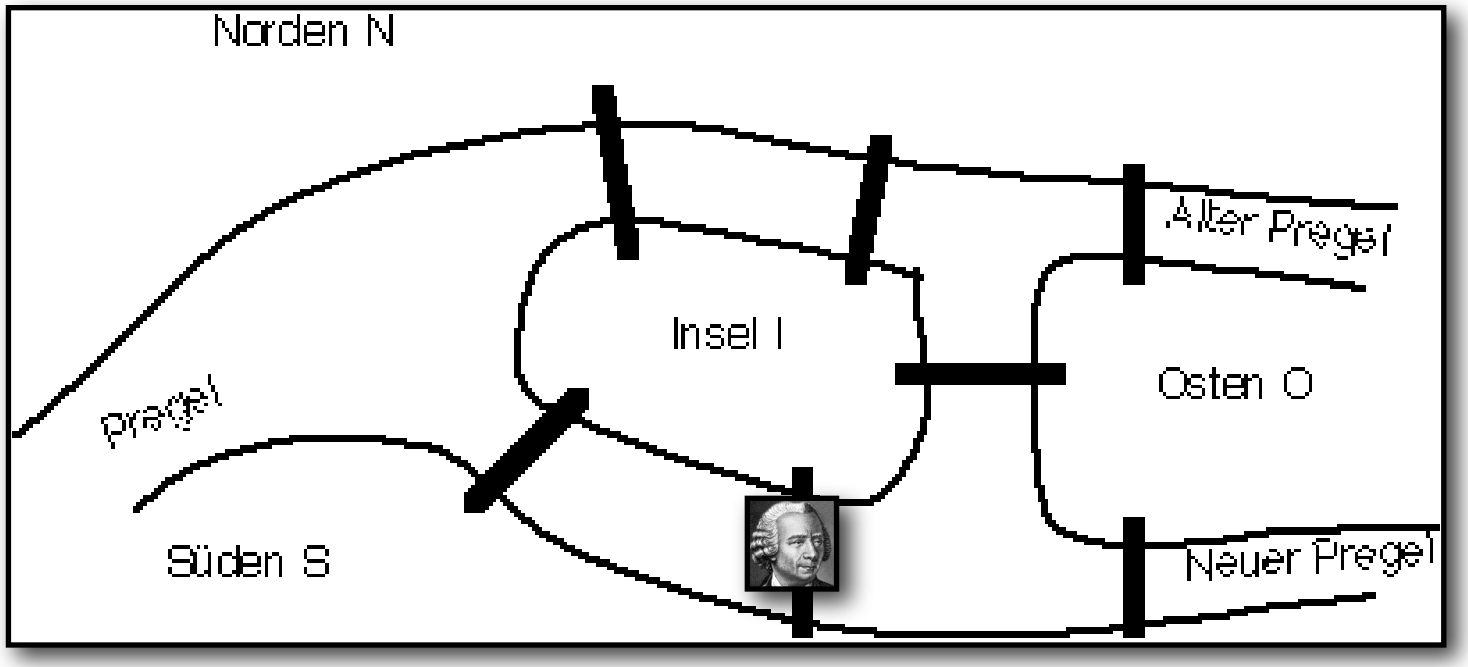




blementierung von Datent.

Graphen

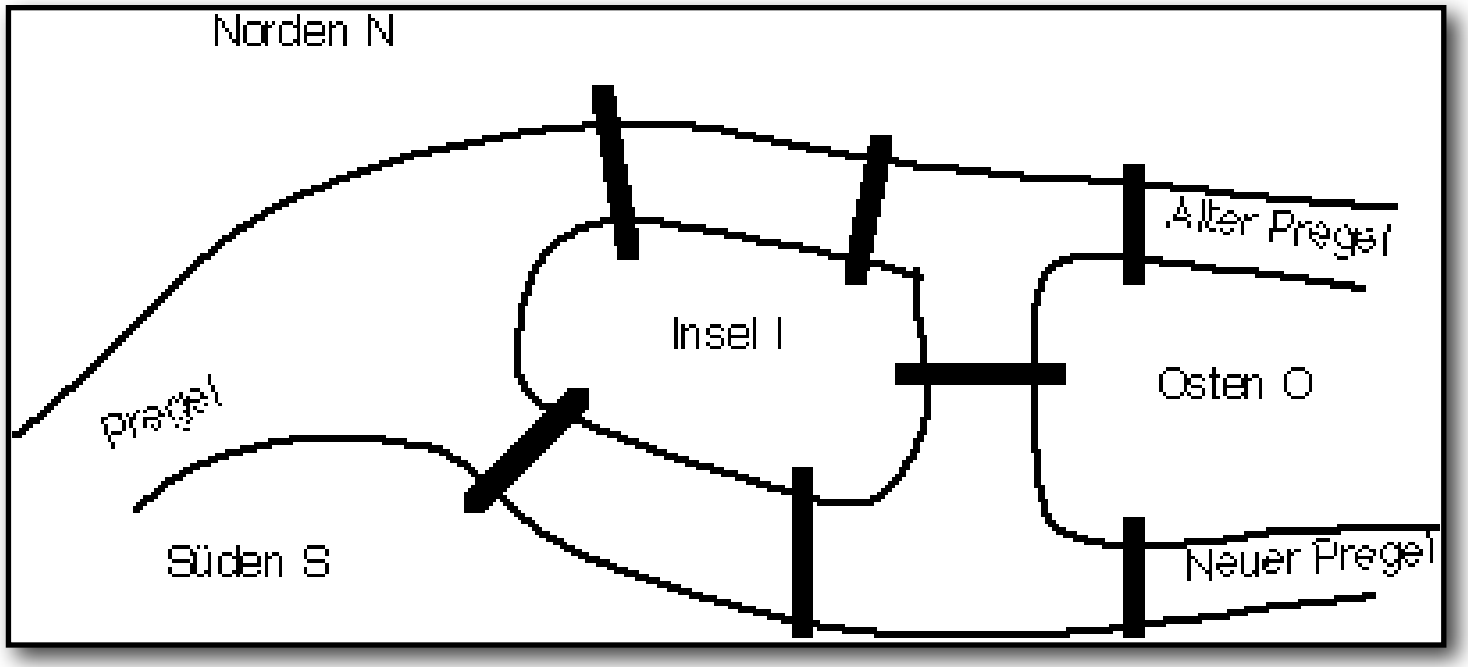
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

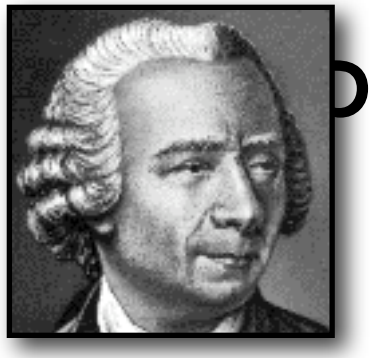


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

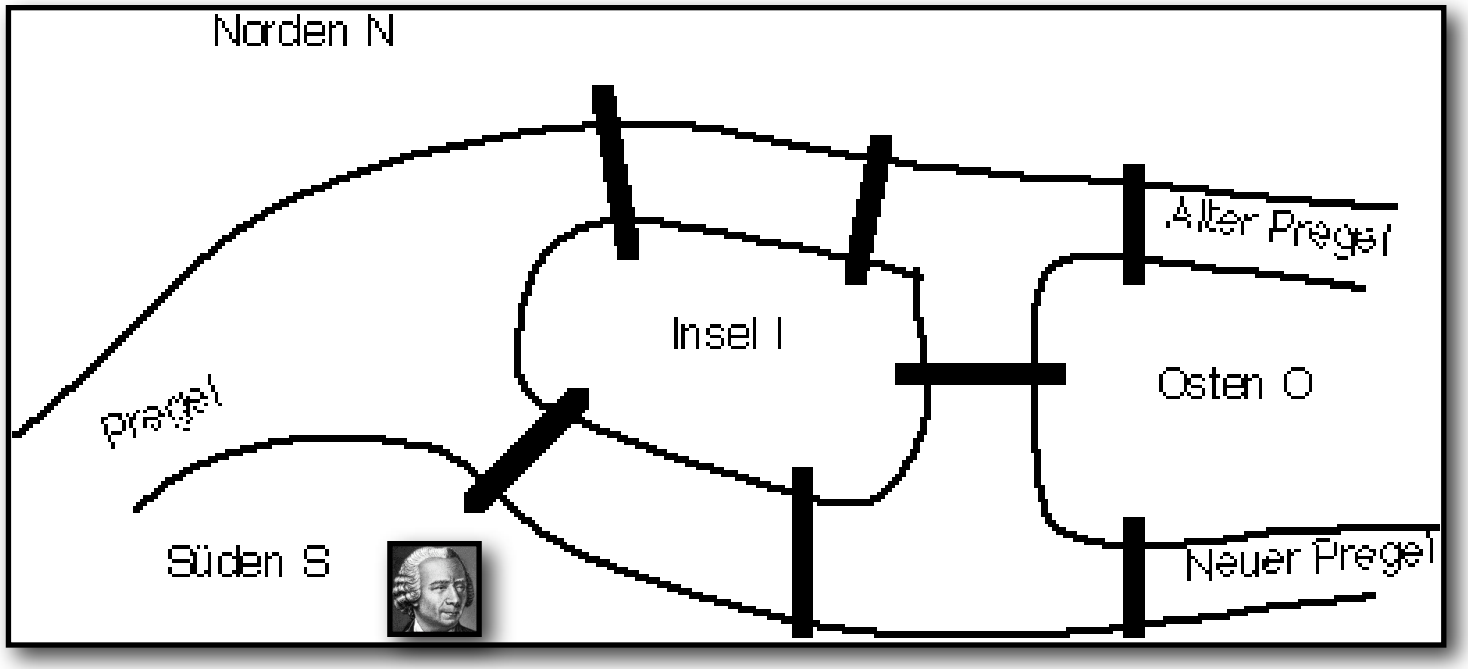




blementierung von Datent.

Graphen

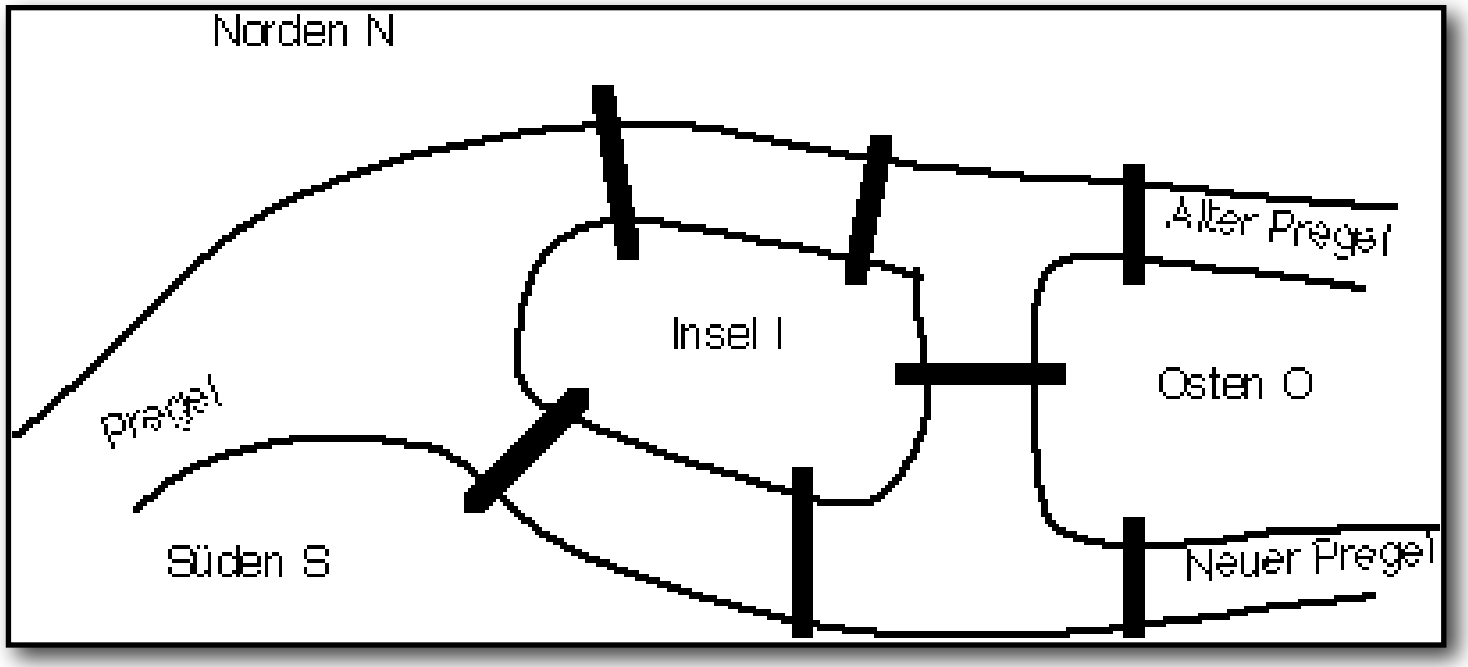
- Geschichte: L. Euler (1736): Königsberger Brückenproblem



blementierung von Datent.

Graphen

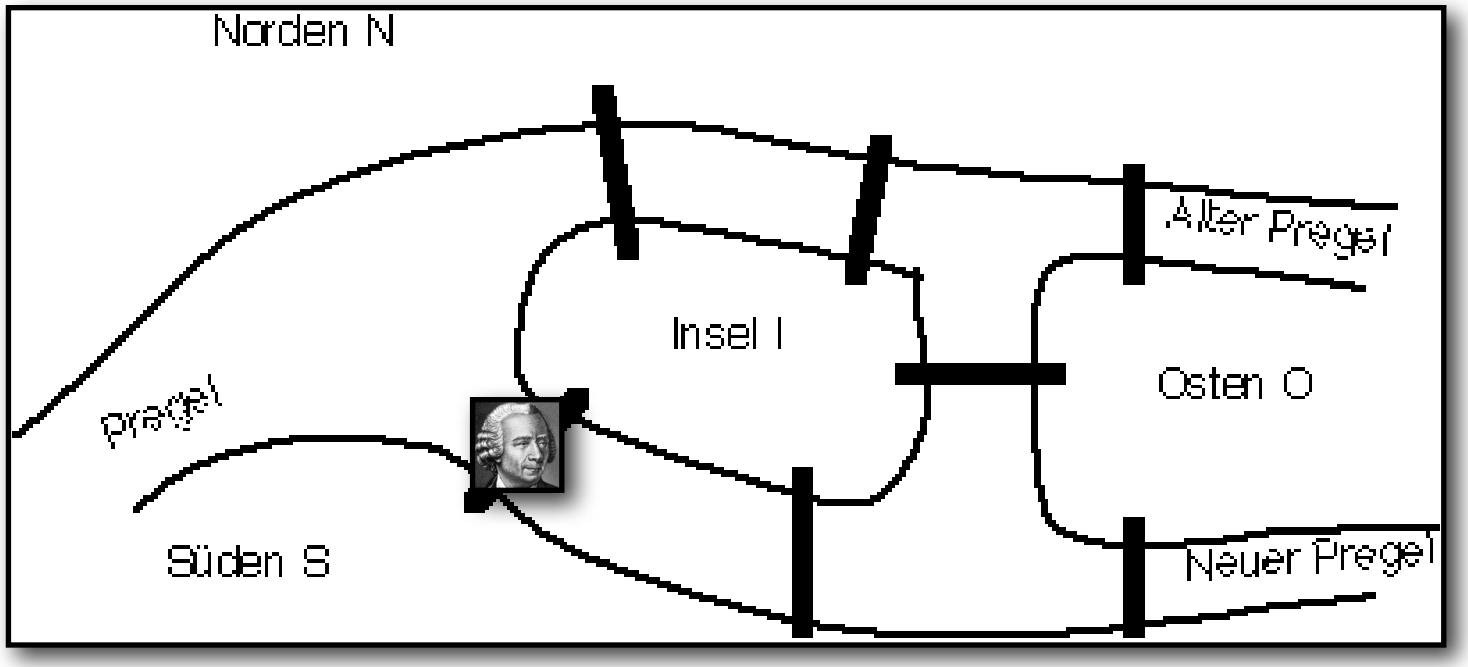
- Geschichte: L. Euler (1736): Königsberger Brückenproblem



blementierung von Datent.

Graphen

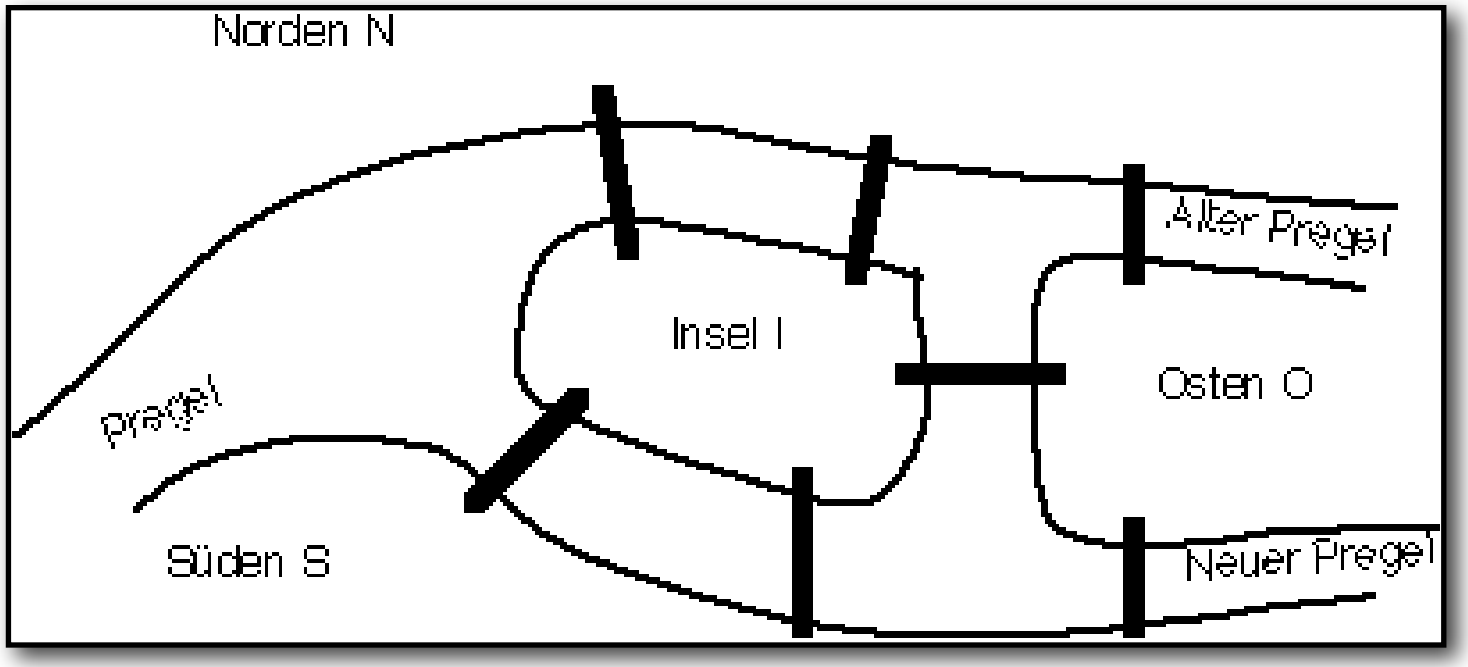
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

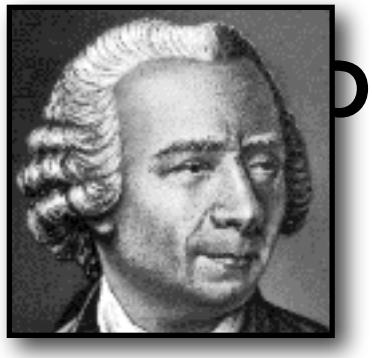


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

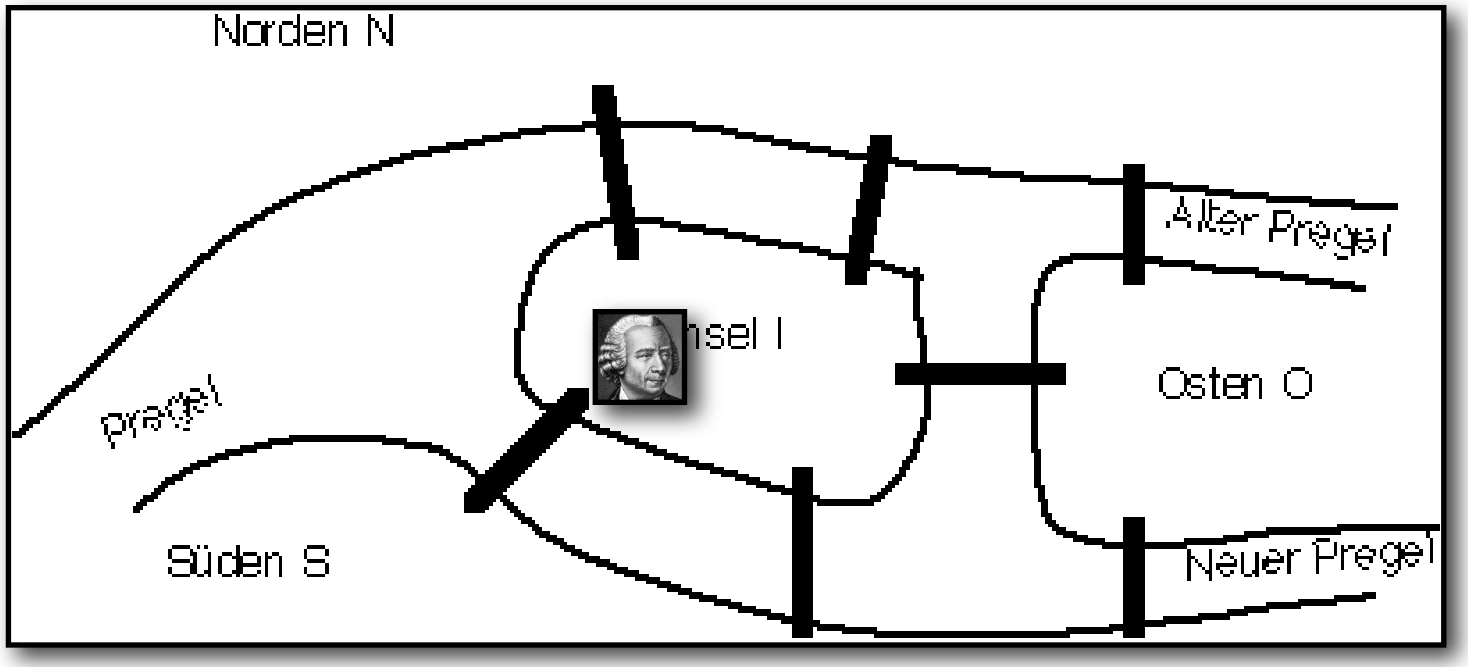




blementierung von Datent.

Graphen

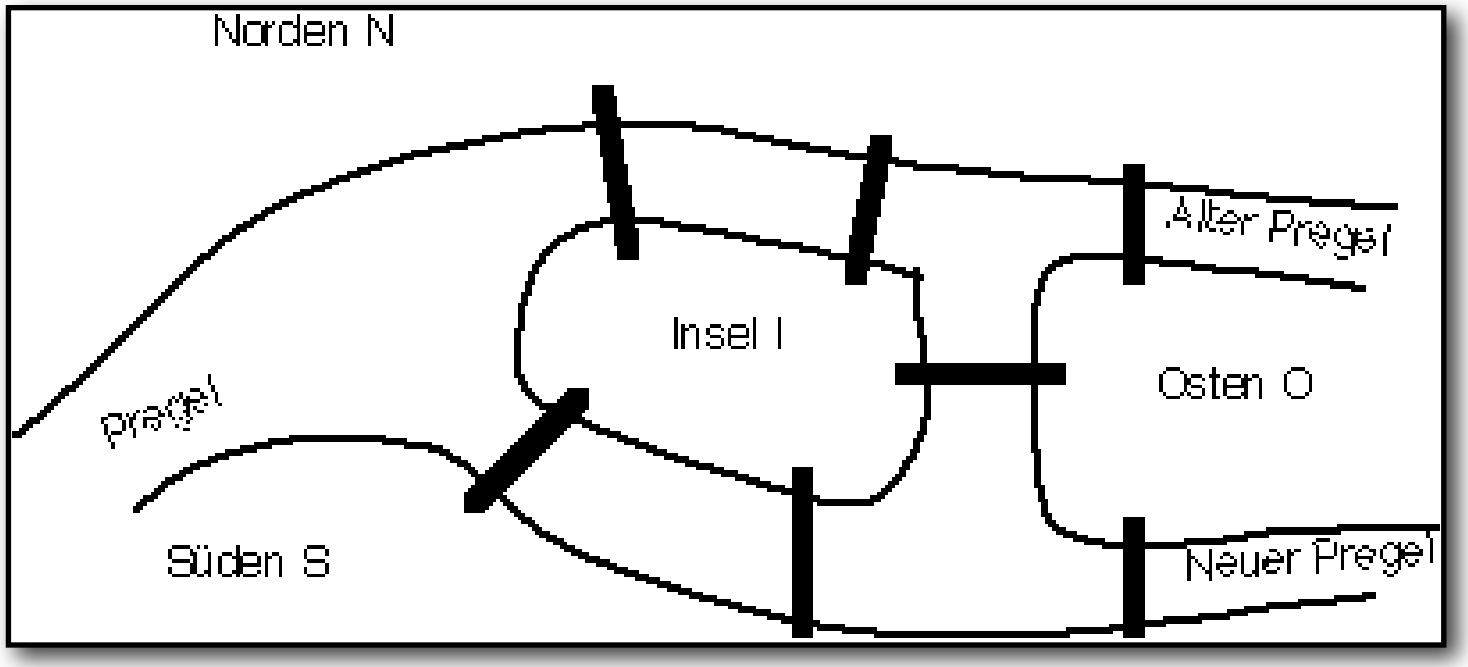
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

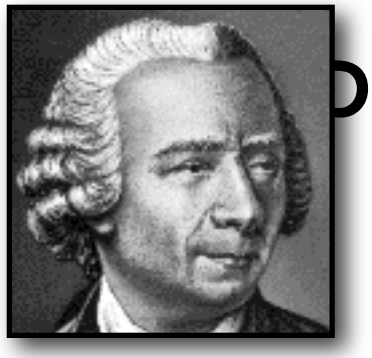


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

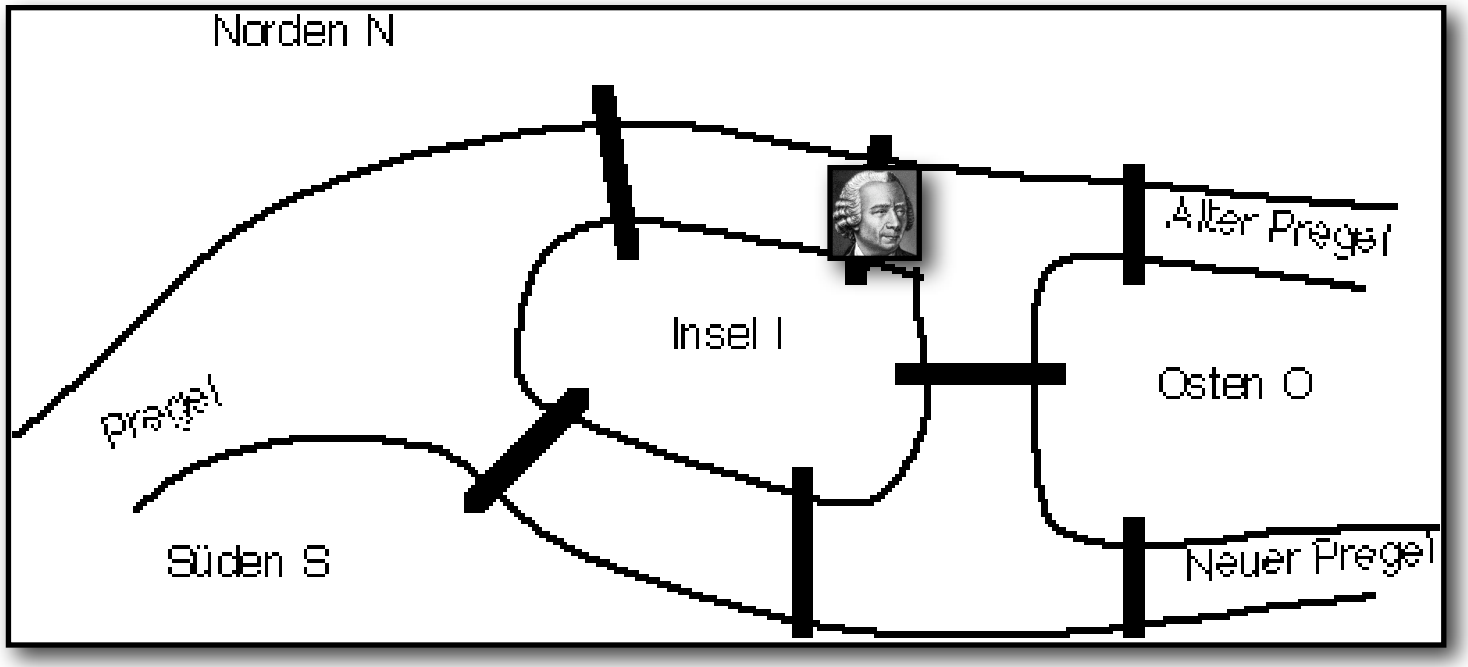




blementierung von Datent.

Graphen

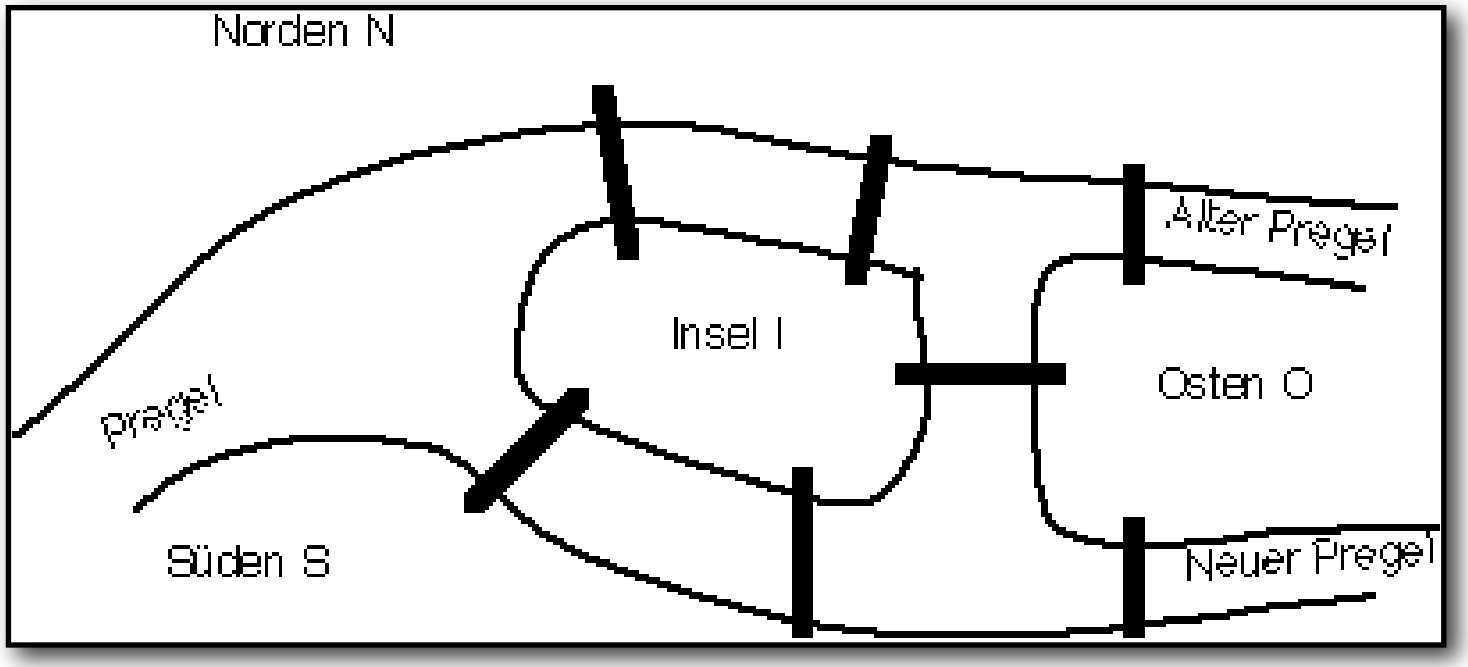
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

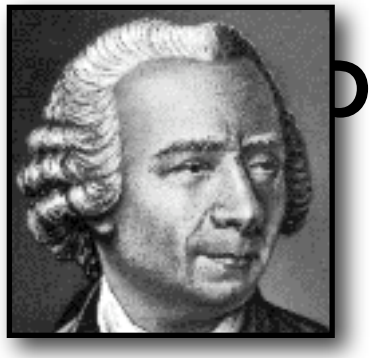


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

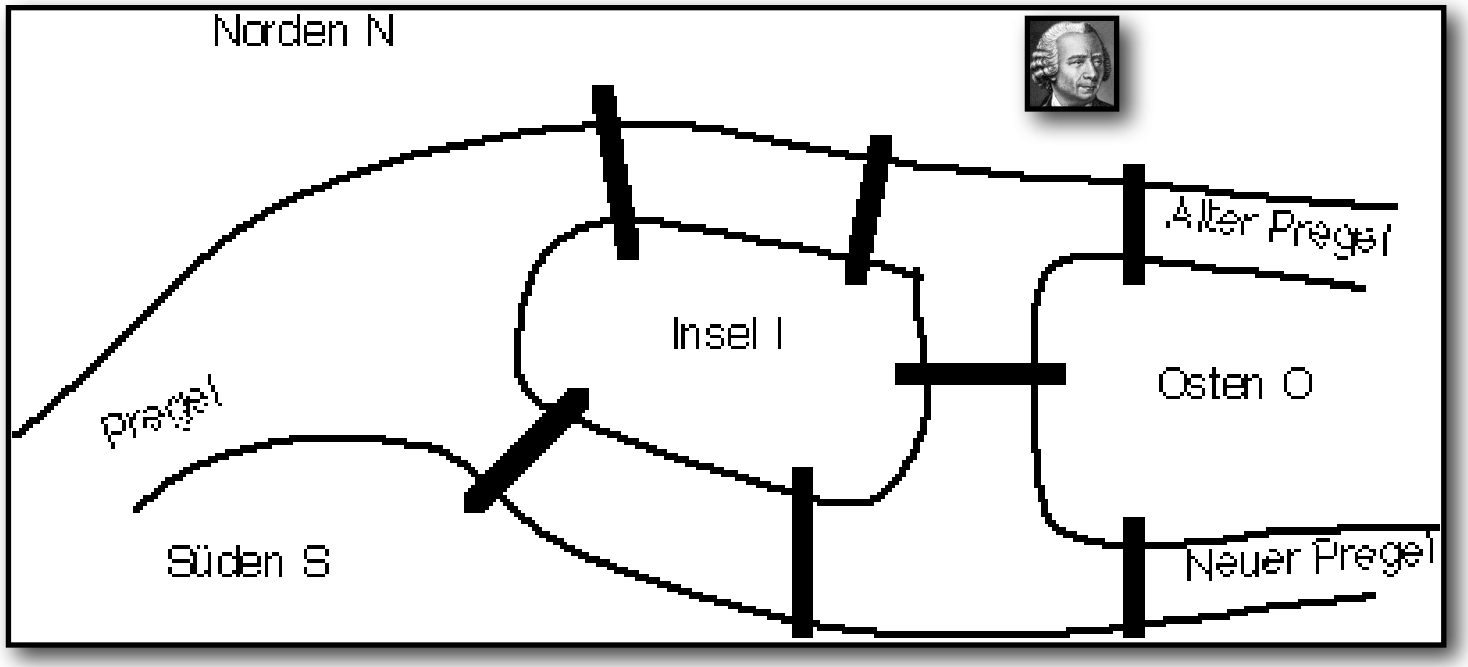




blementierung von Datent.

Graphen

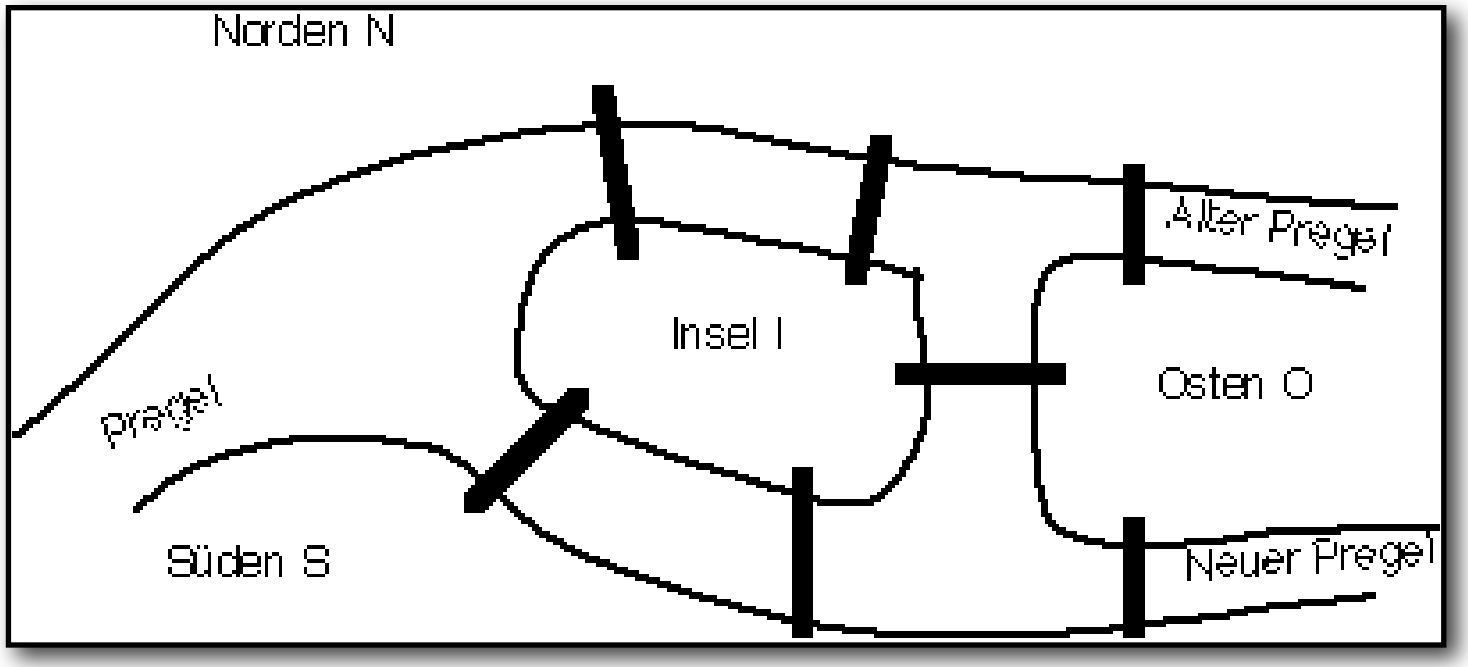
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

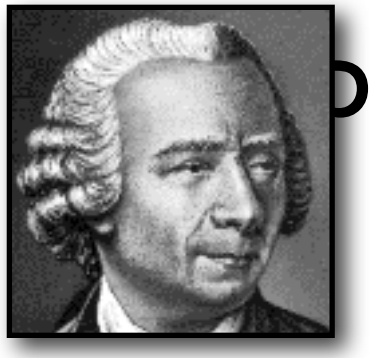


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

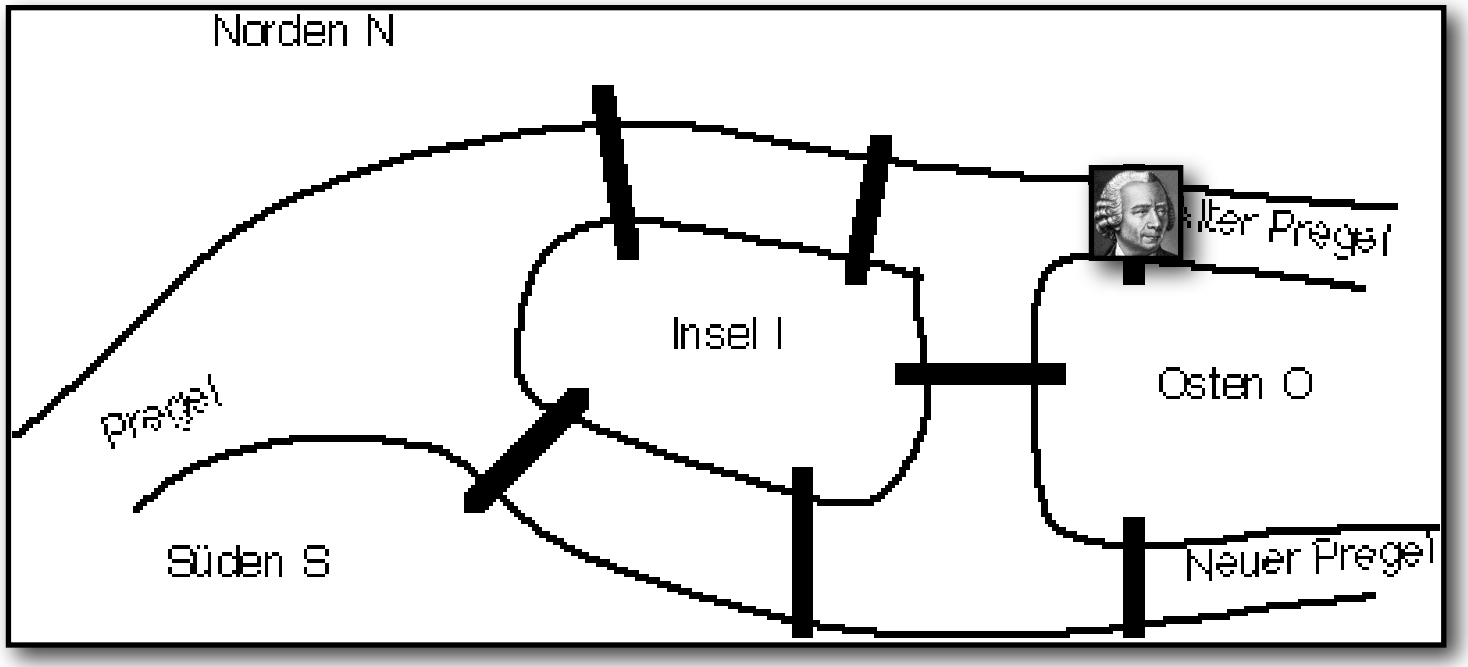




blementierung von Datent.

Graphen

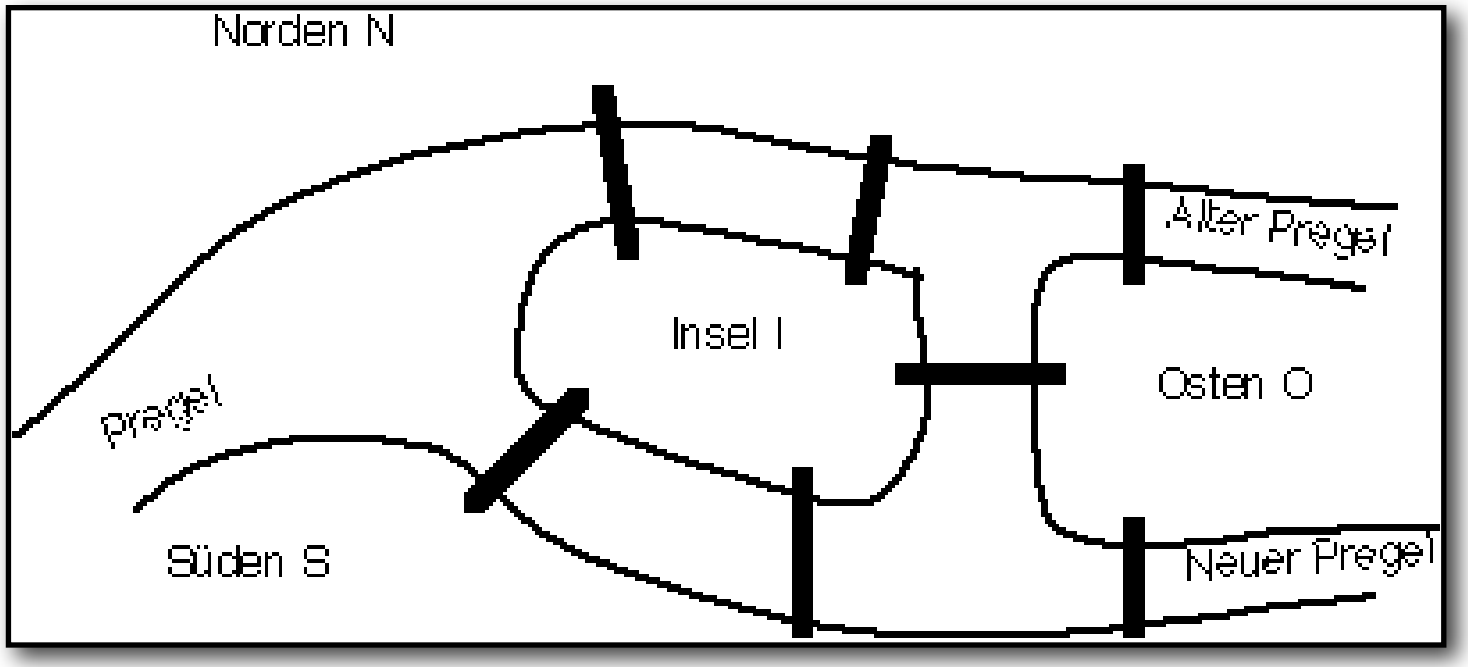
- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m

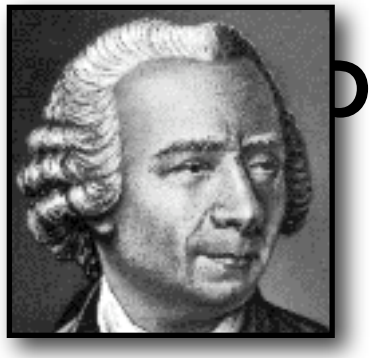


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

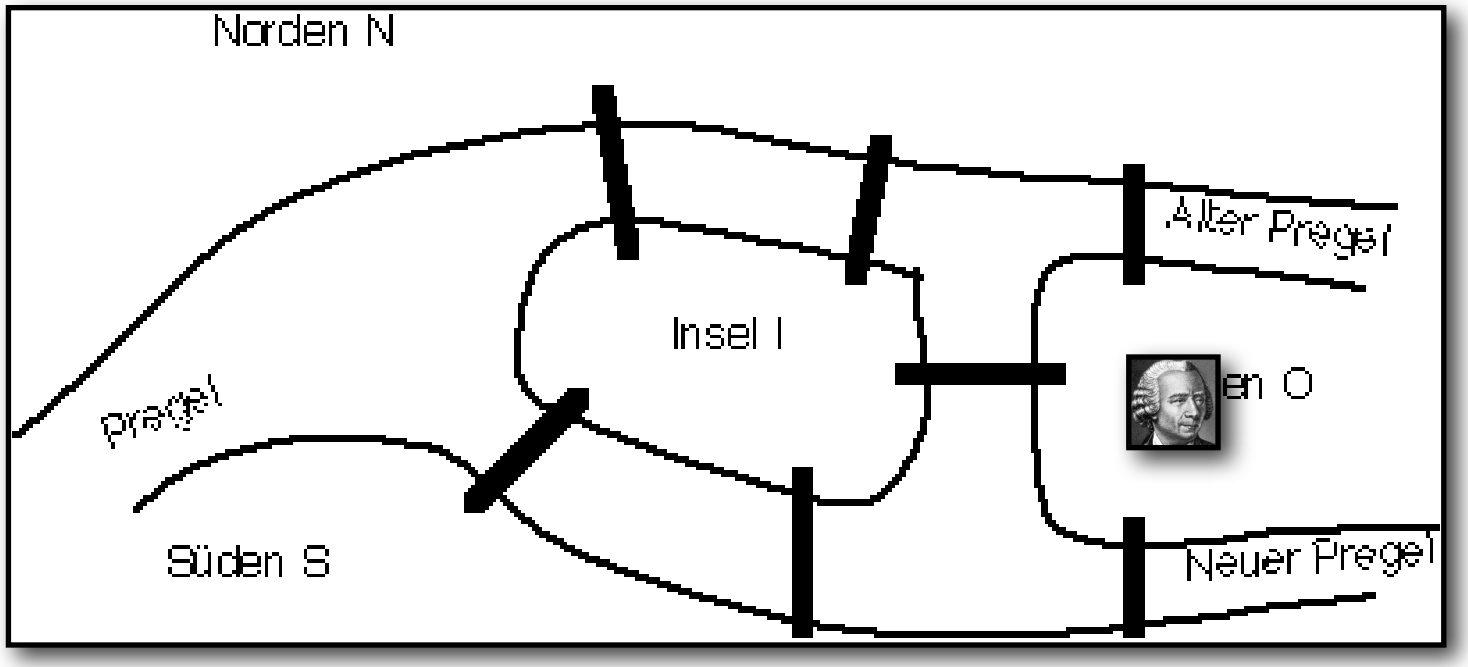




blementierung von Datent.

Graphen

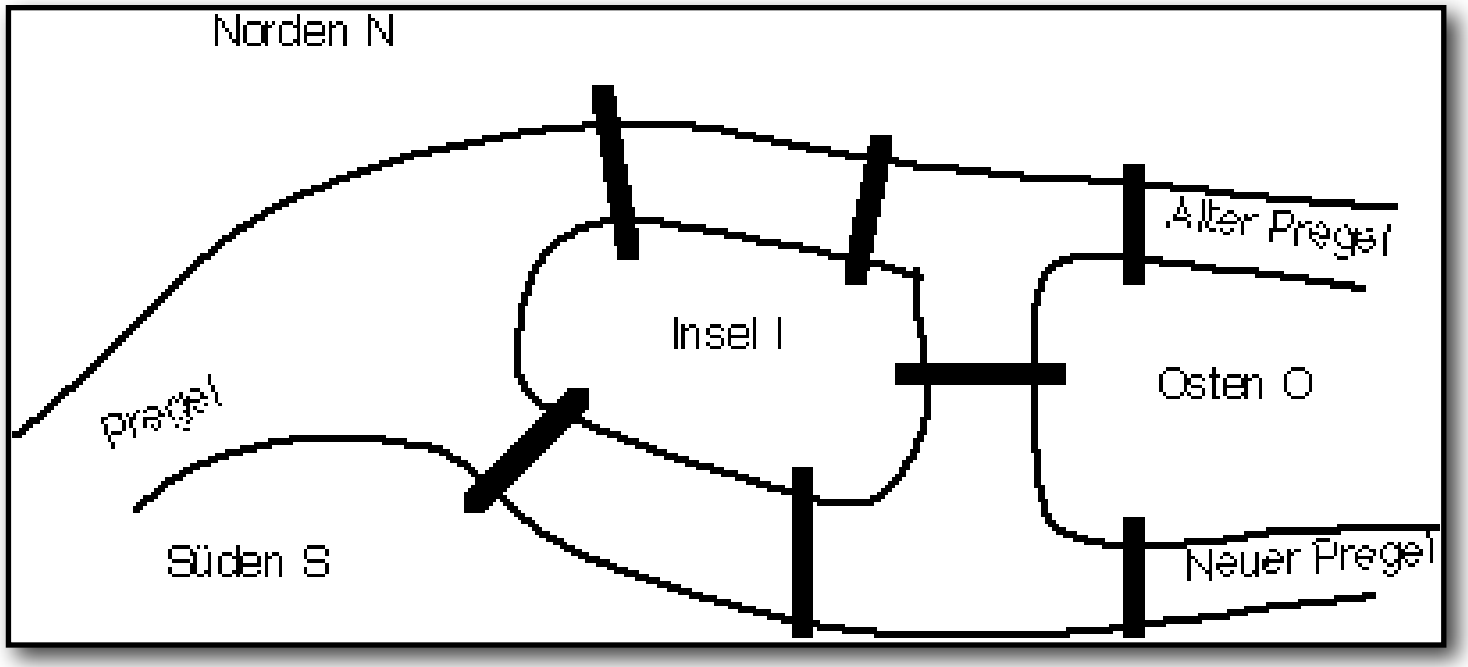
- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m

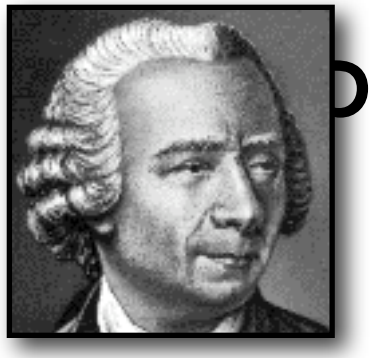


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

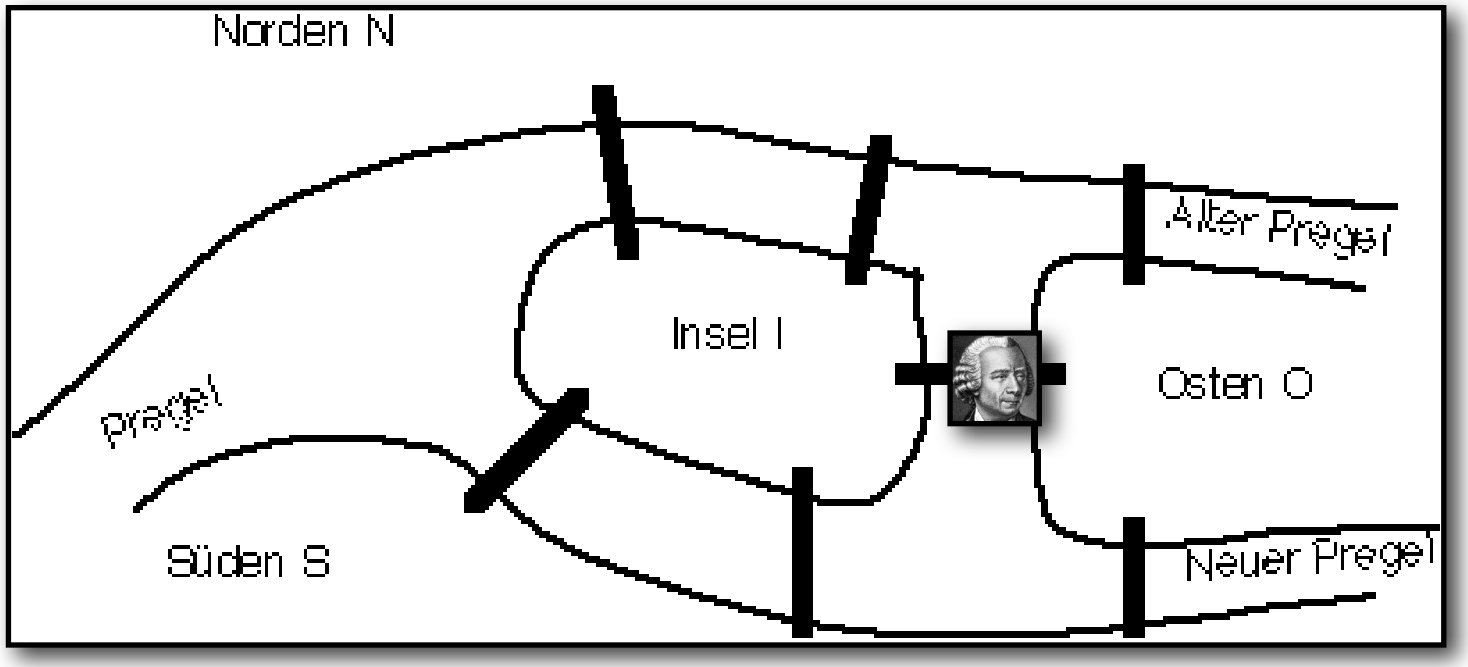




blementierung von Datent.

Graphen

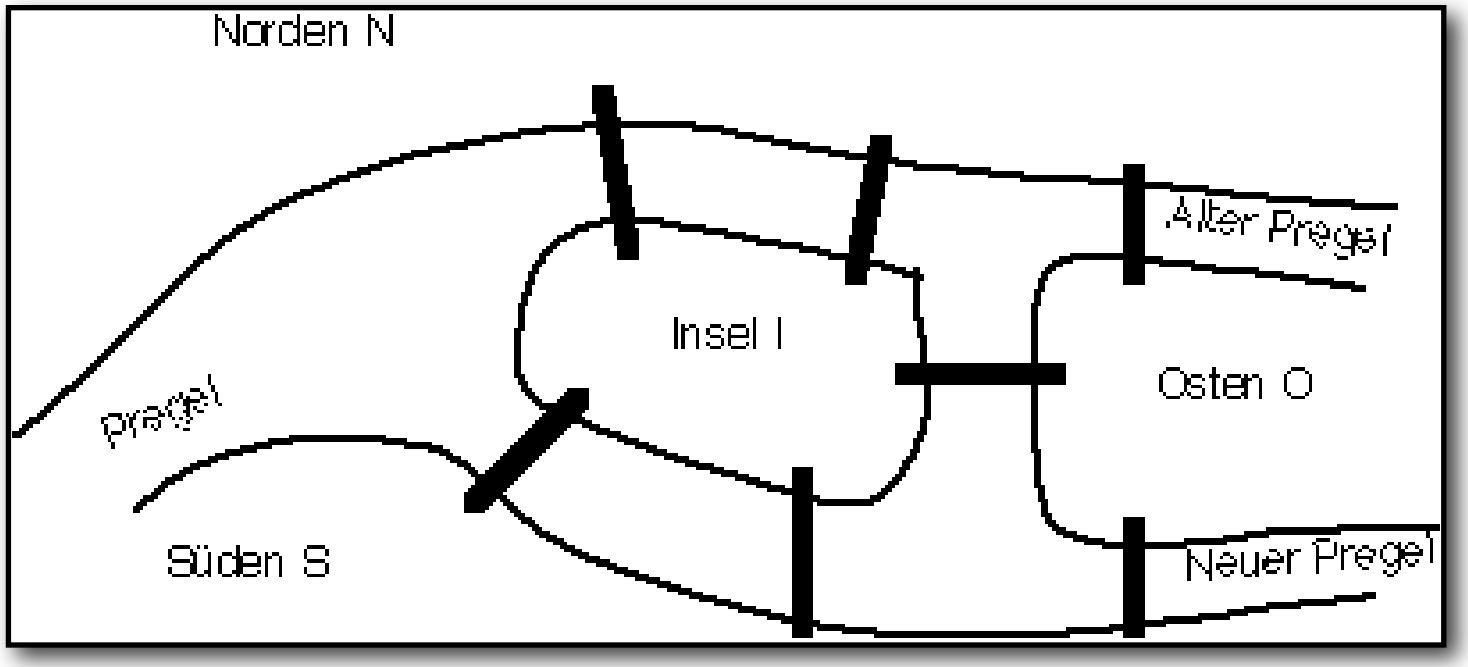
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

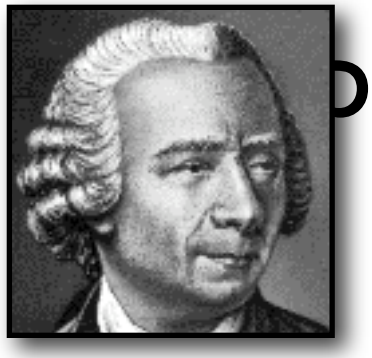


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

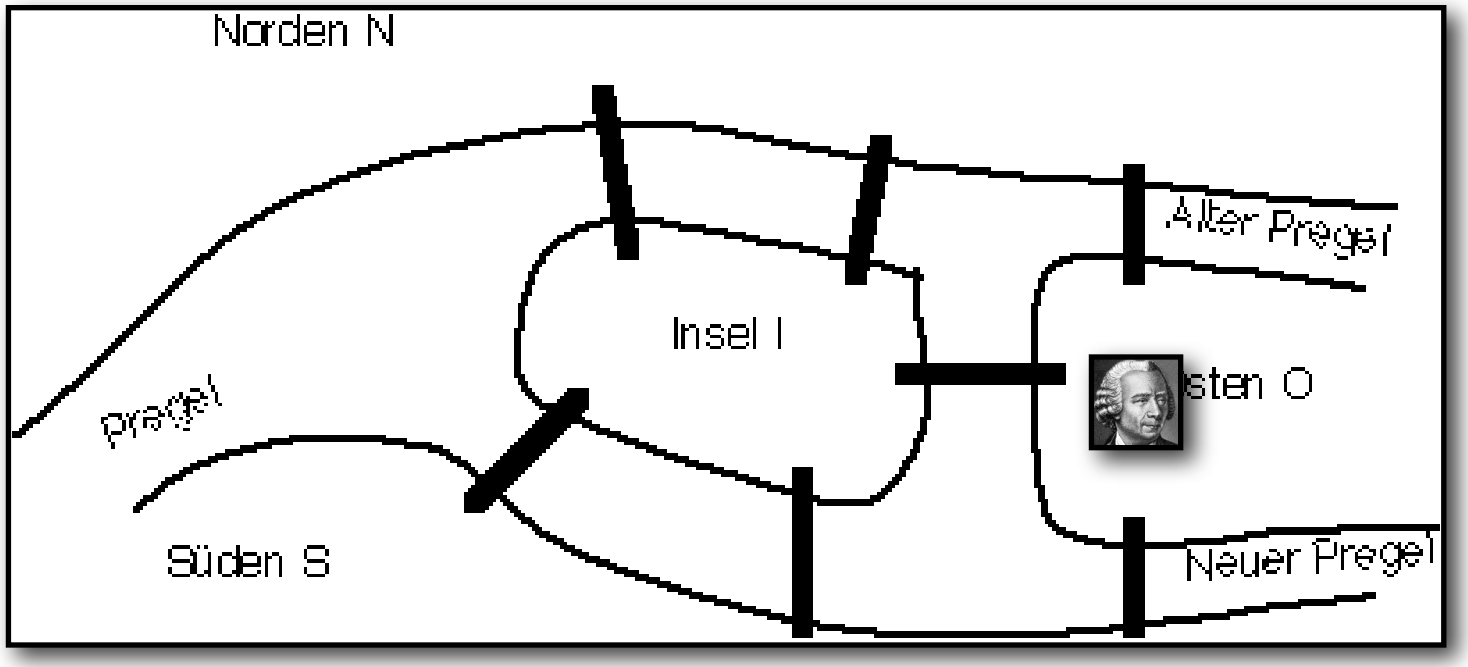




blementierung von Datent.

Graphen

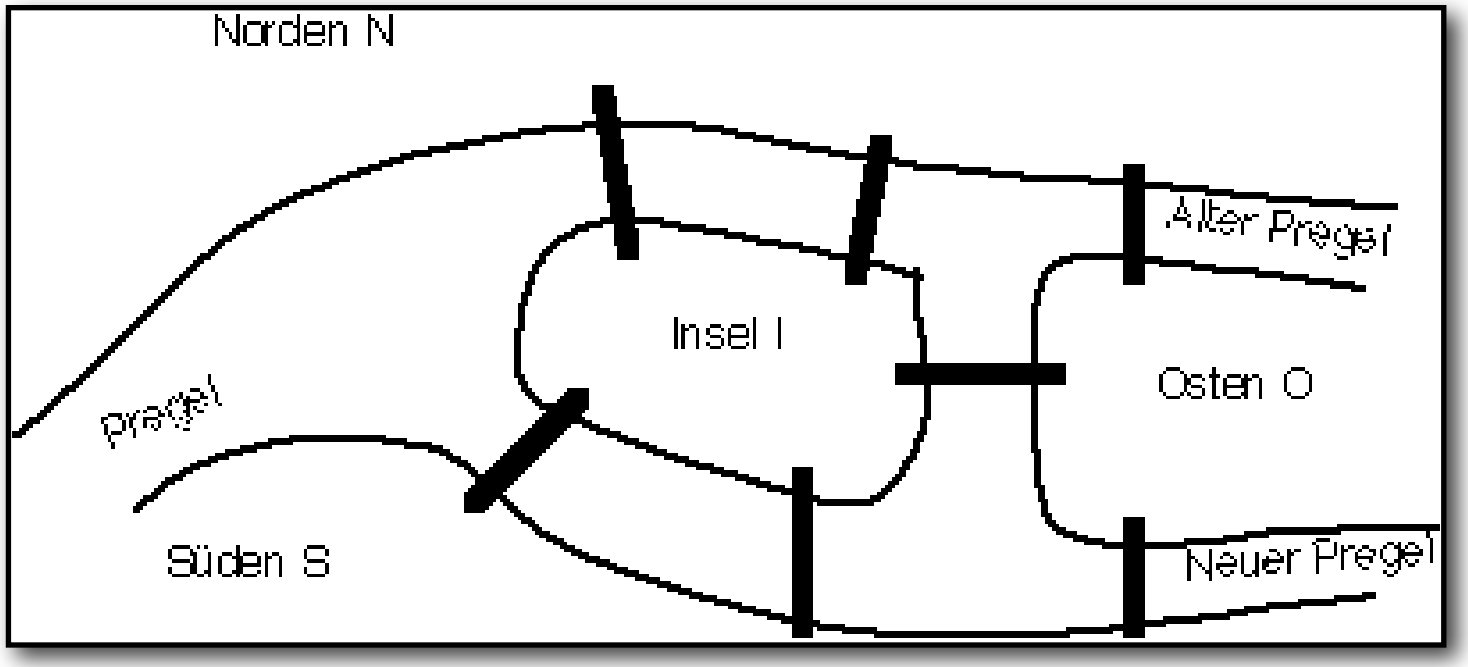
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

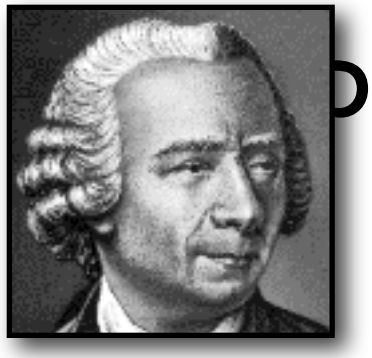


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

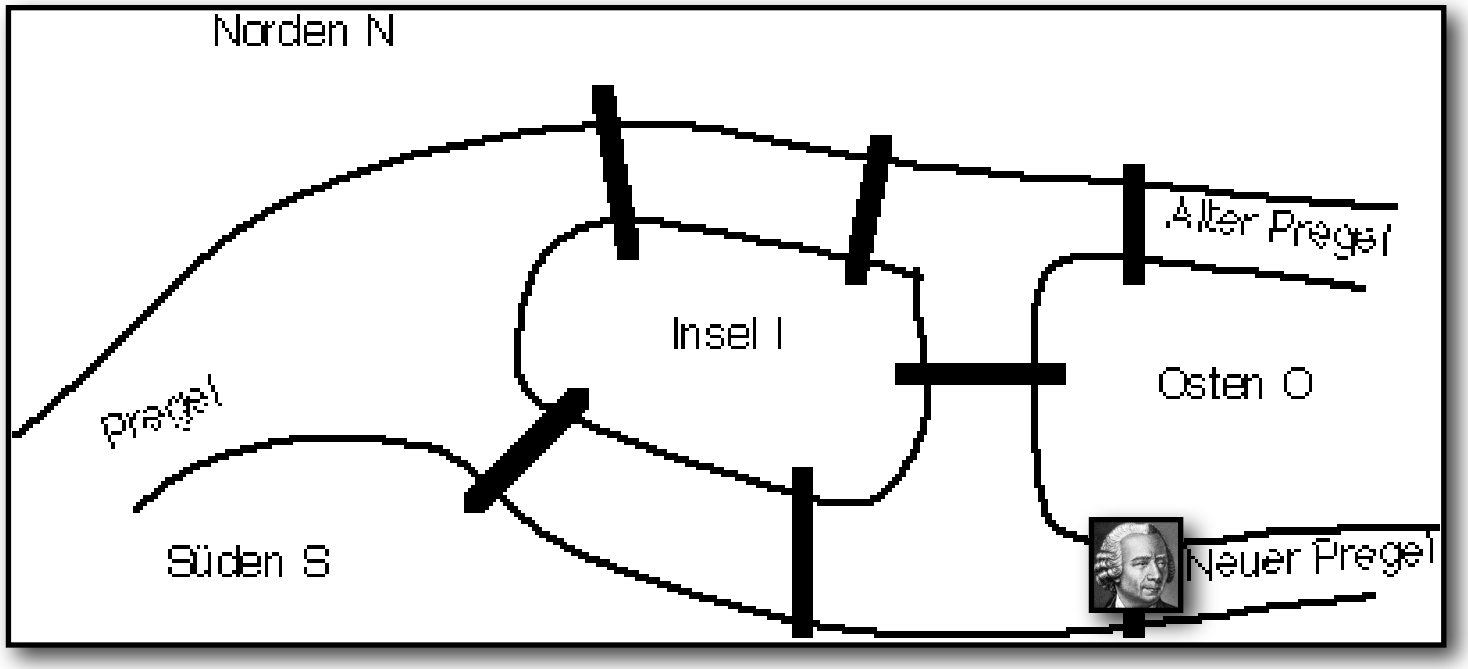




blementierung von Datent.

Graphen

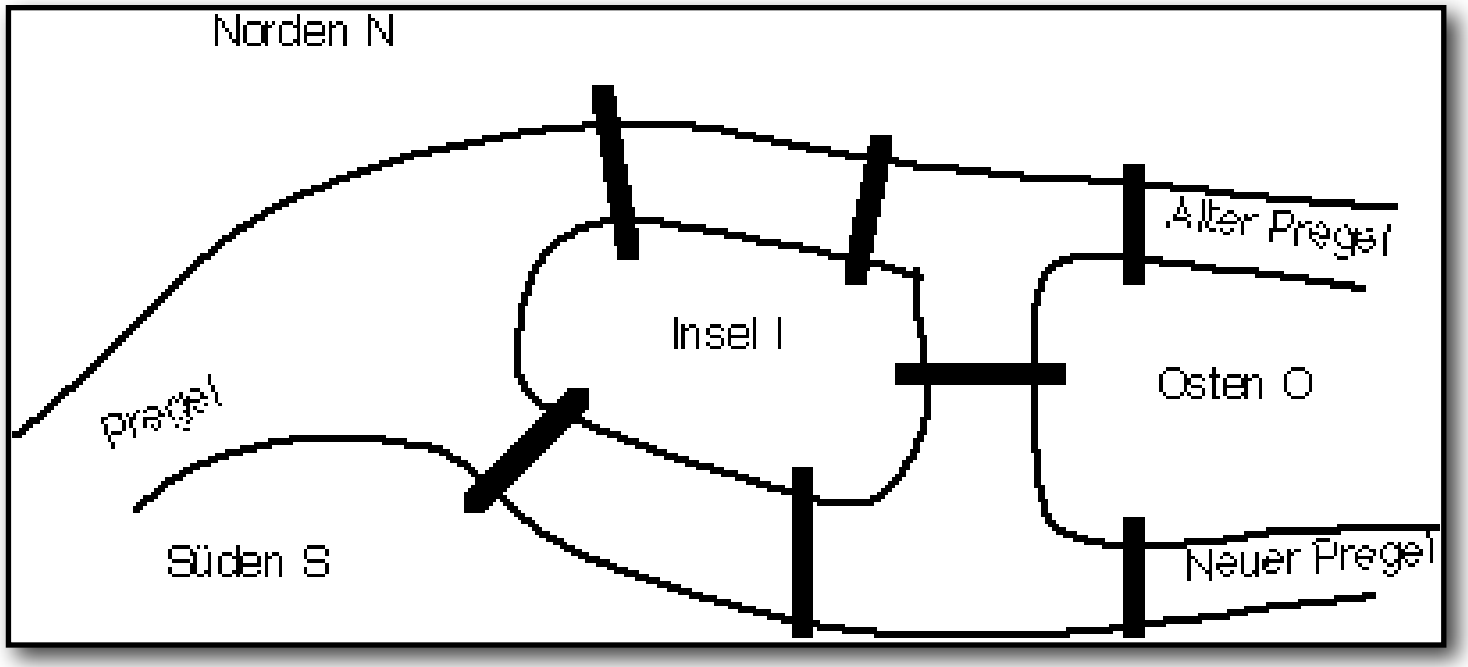
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

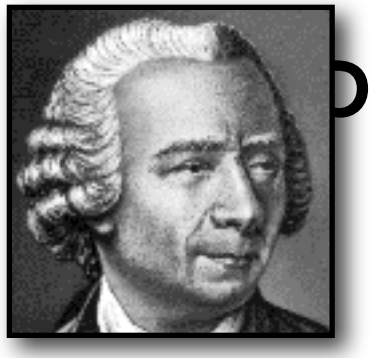


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

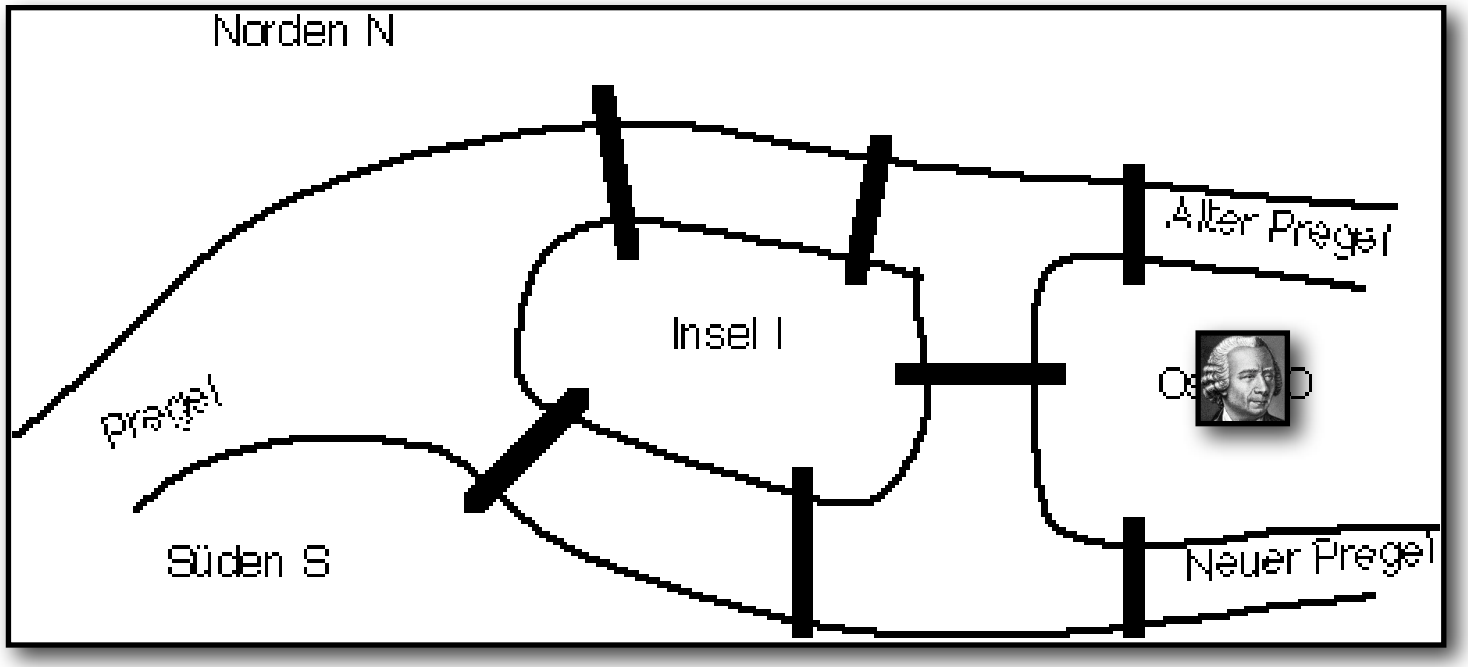


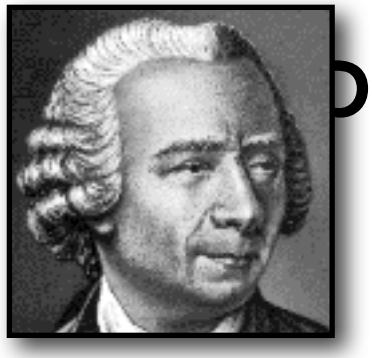


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenprobem

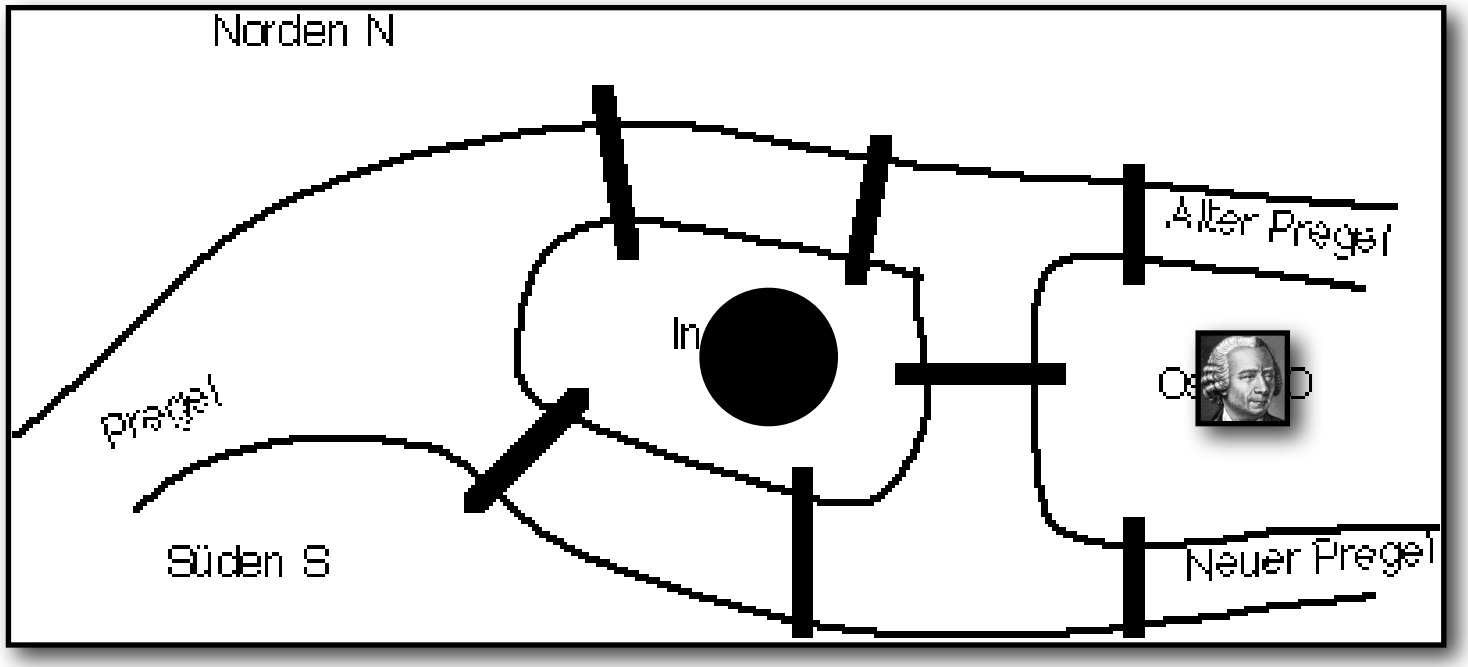


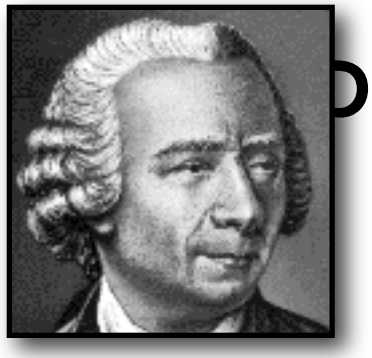


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m

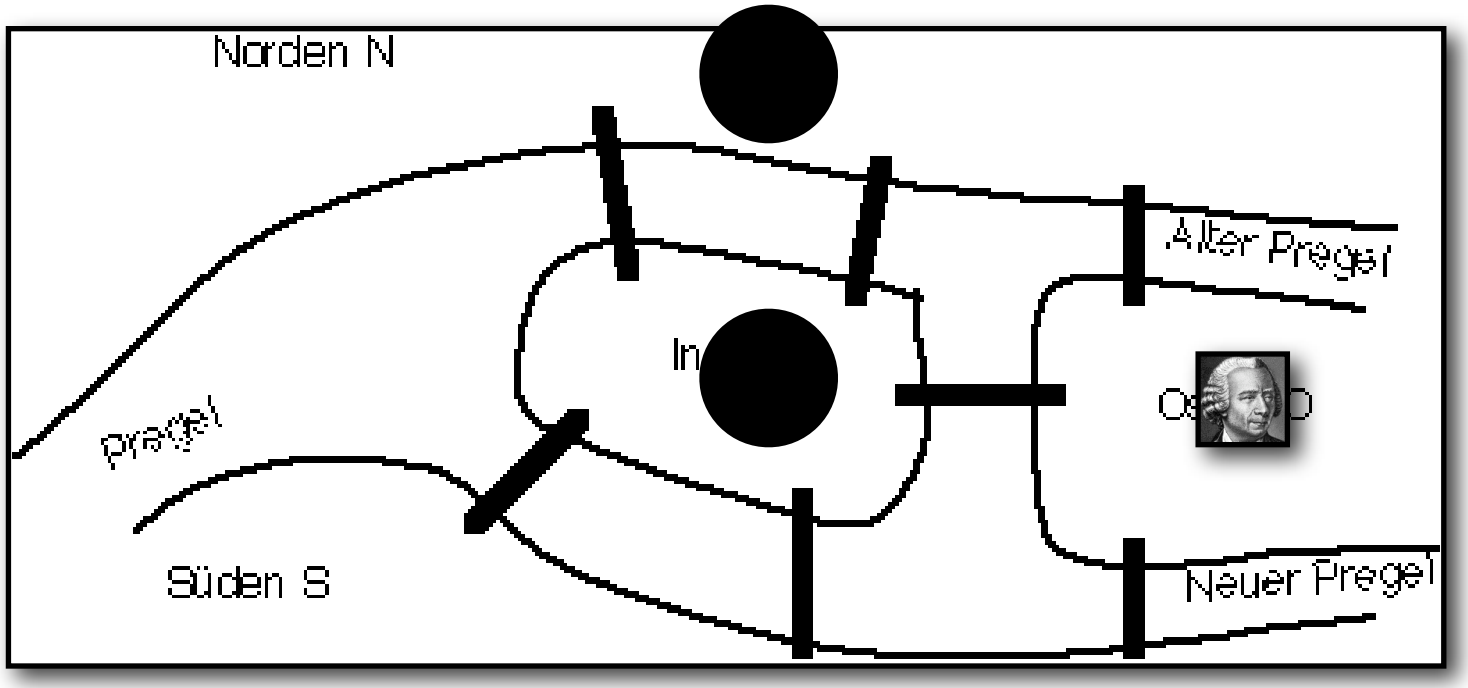




blementierung von Datent.

Graphen

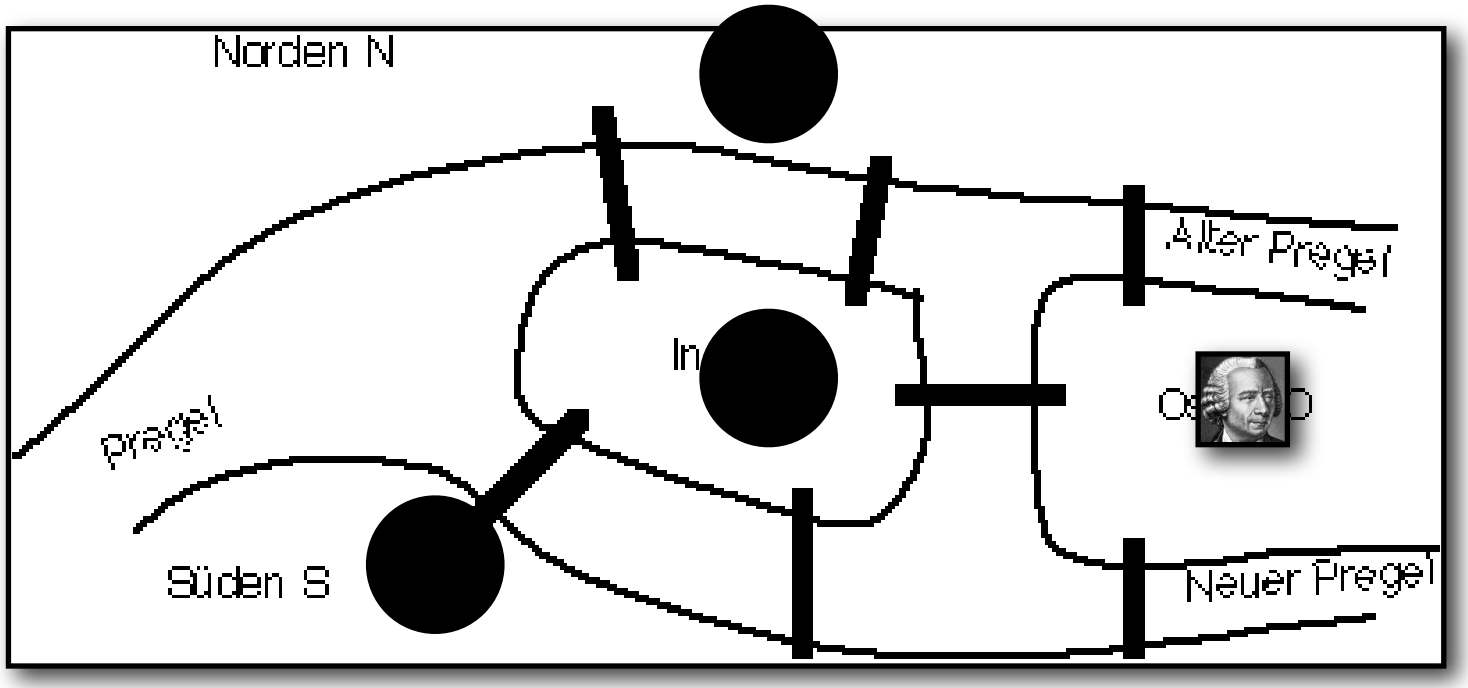
- Geschichte: L. Euler (1736): Königsberger Brückenproblem

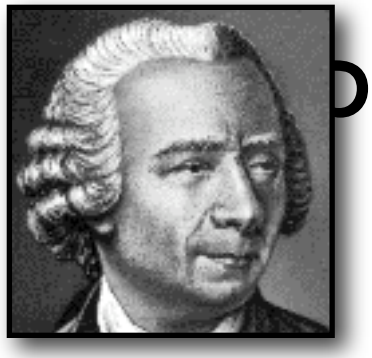


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

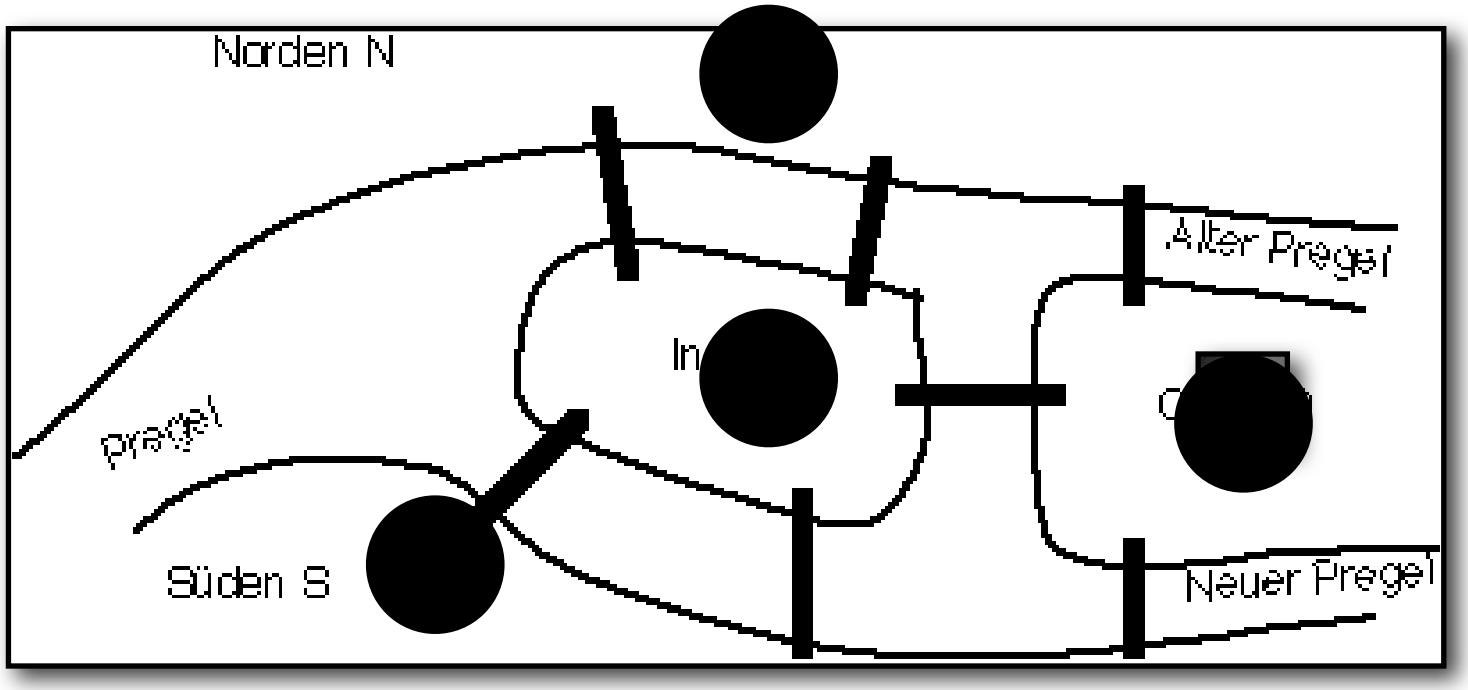


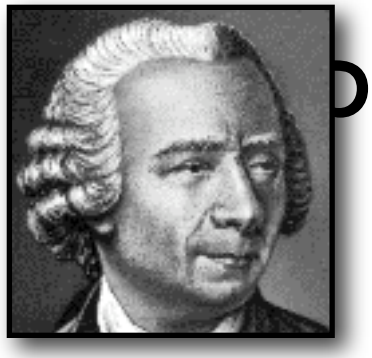


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

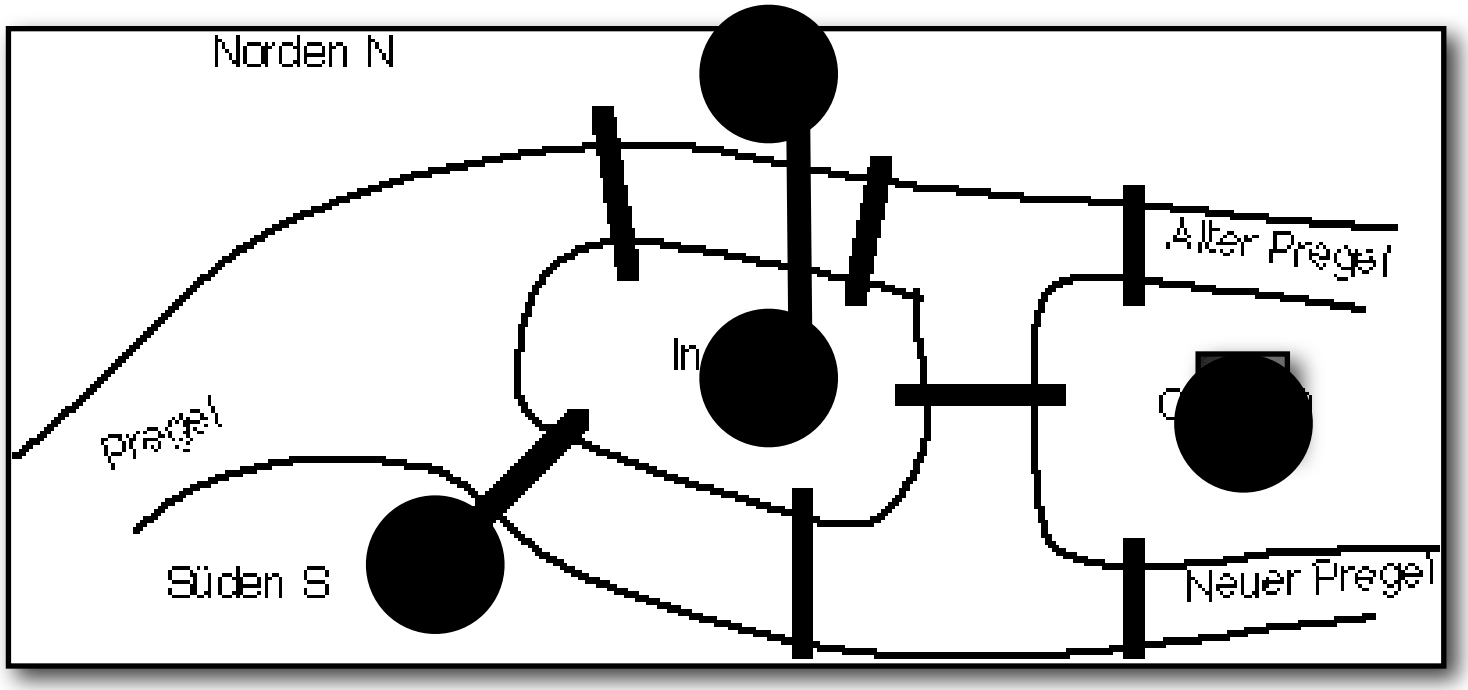


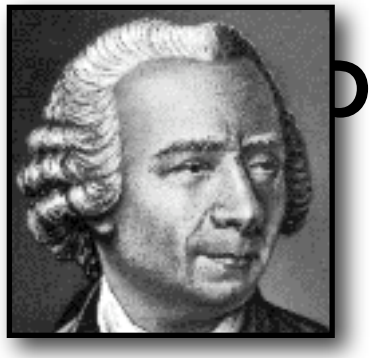


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m

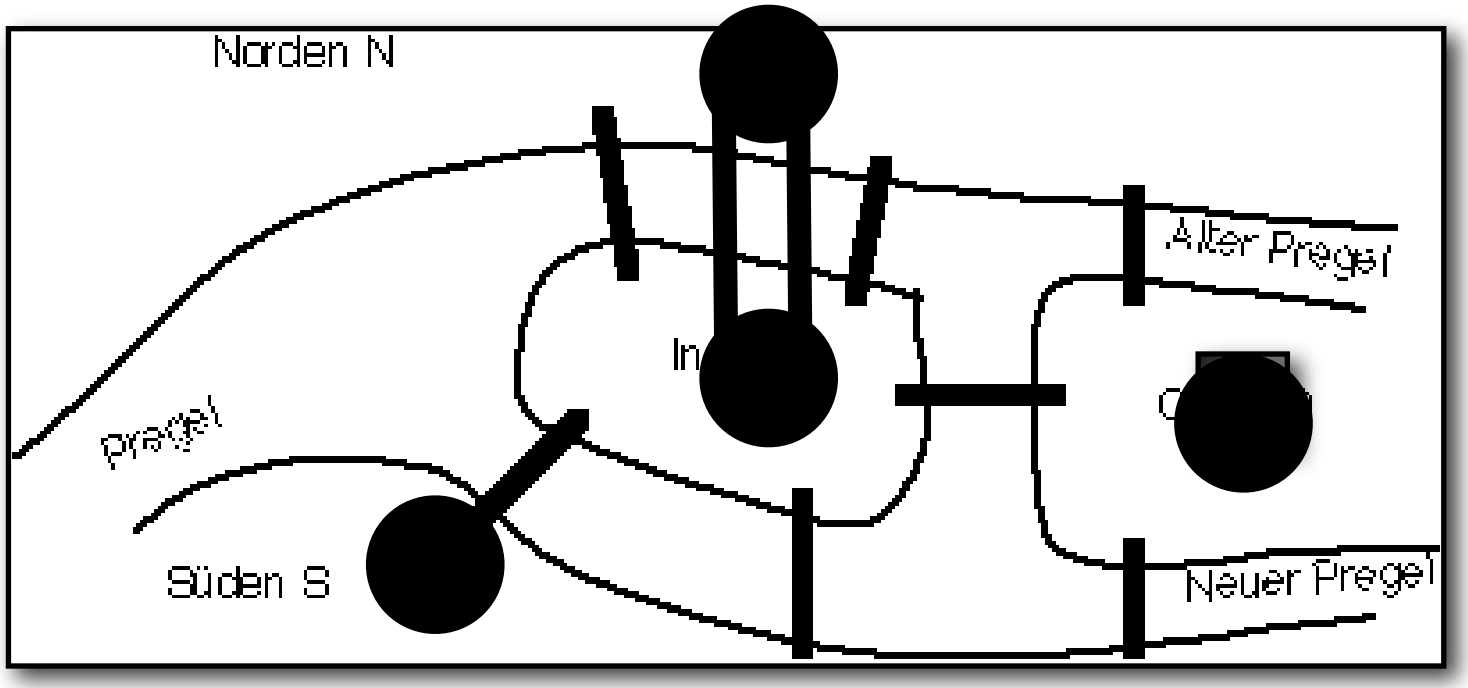


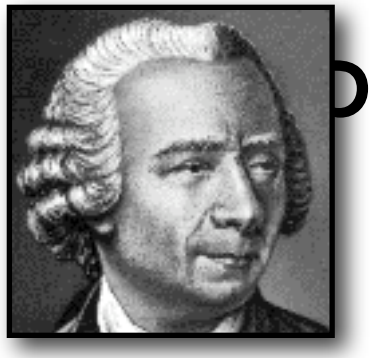


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenproblem

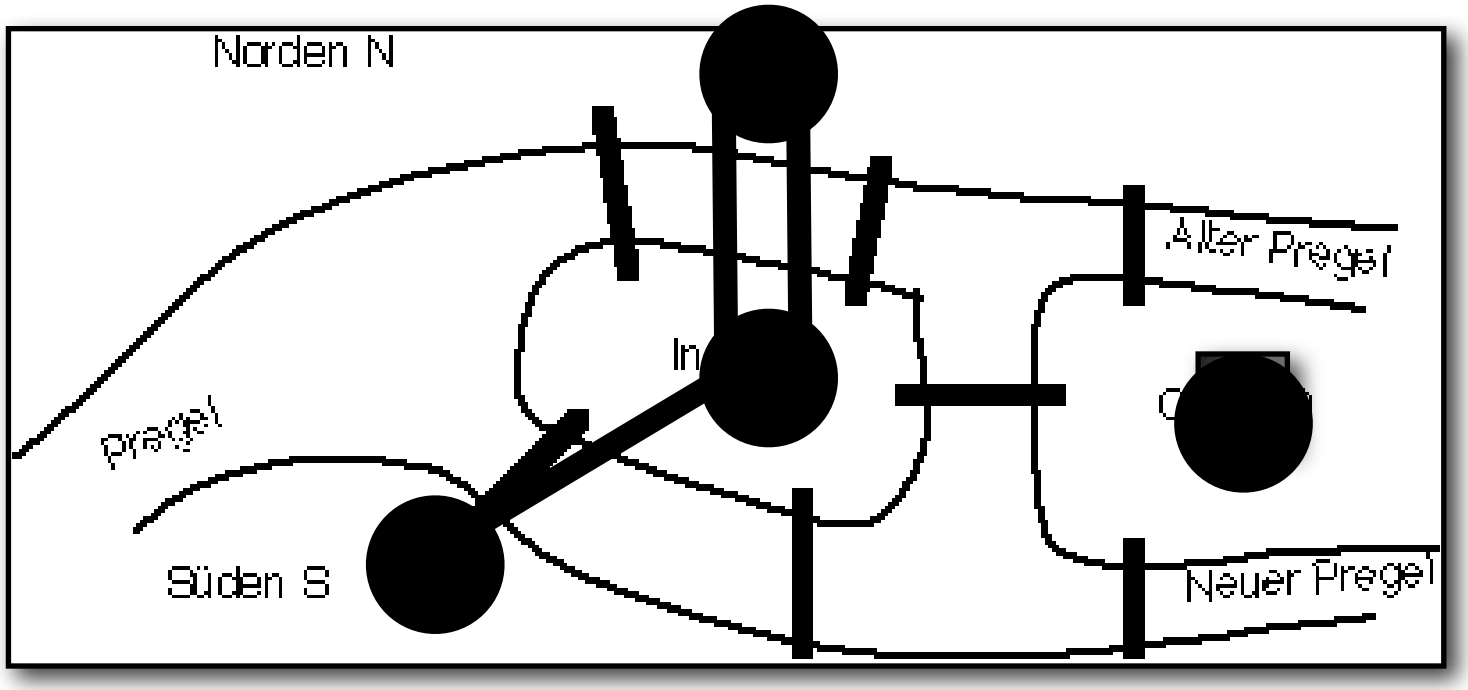




blementierung von Datent.

Graphen

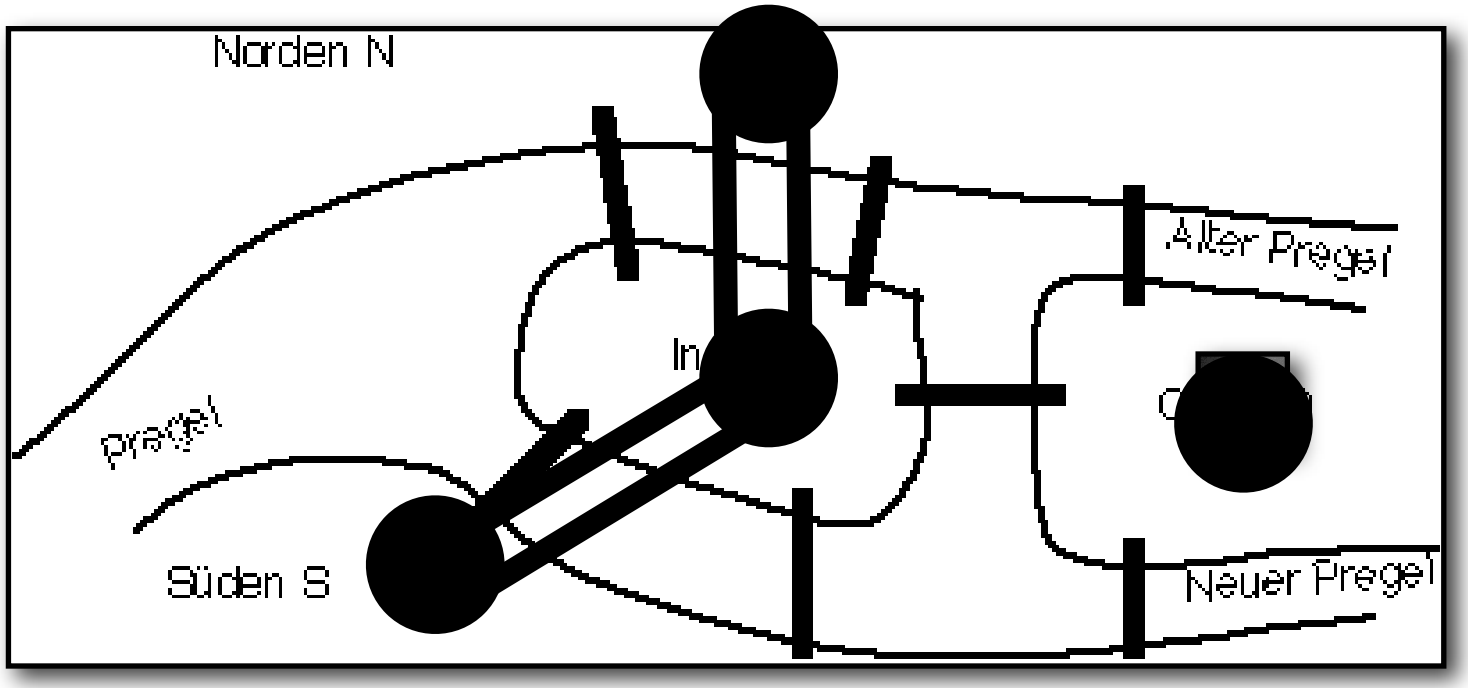
- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m

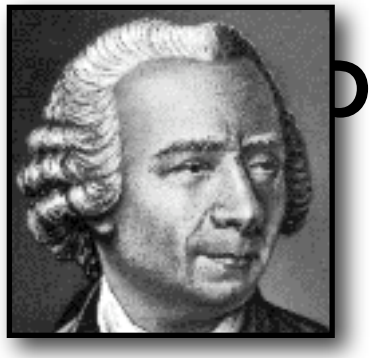


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m

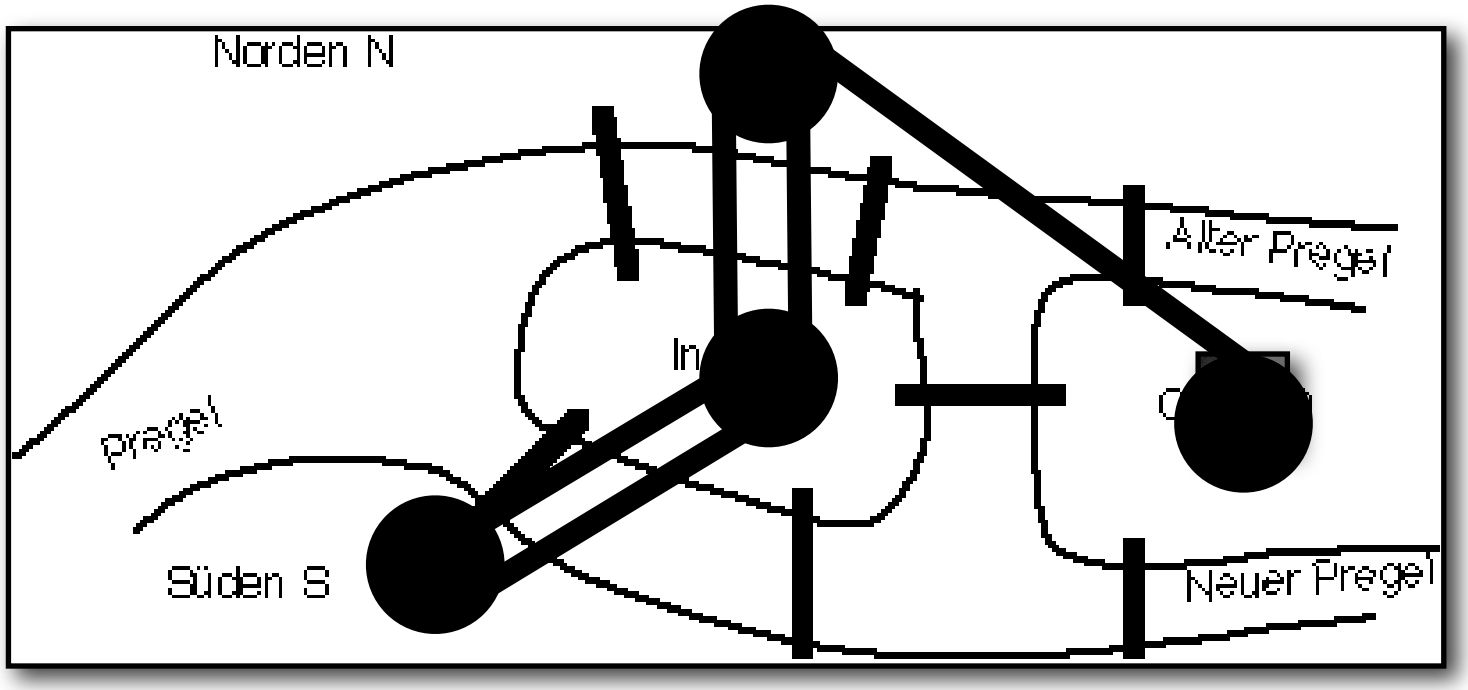


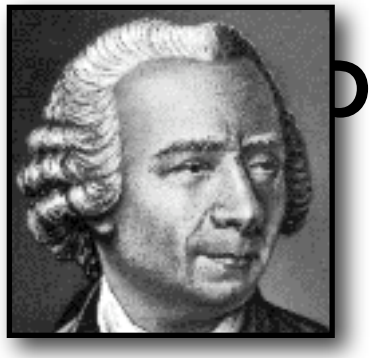


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m

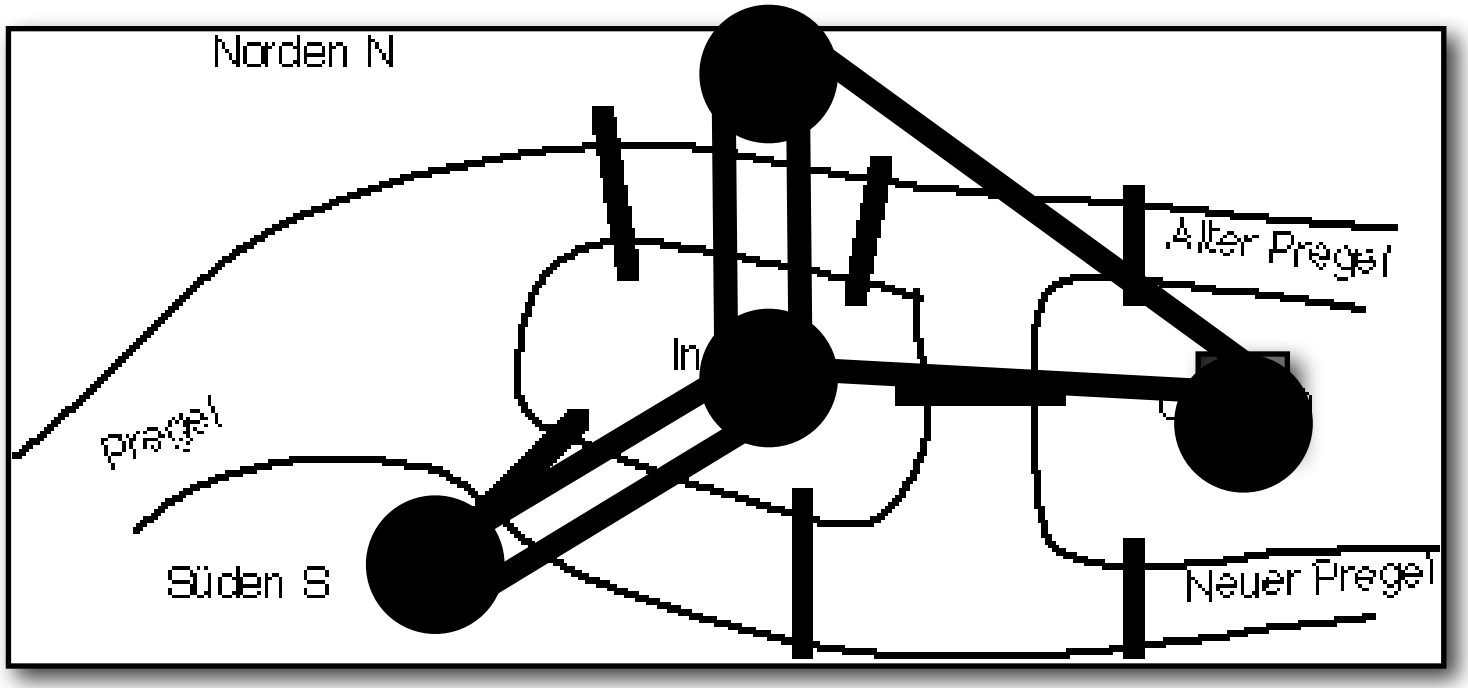


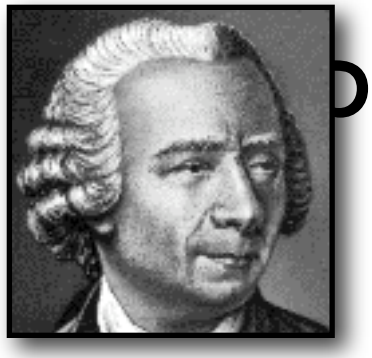


blementierung von Datent.

Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m

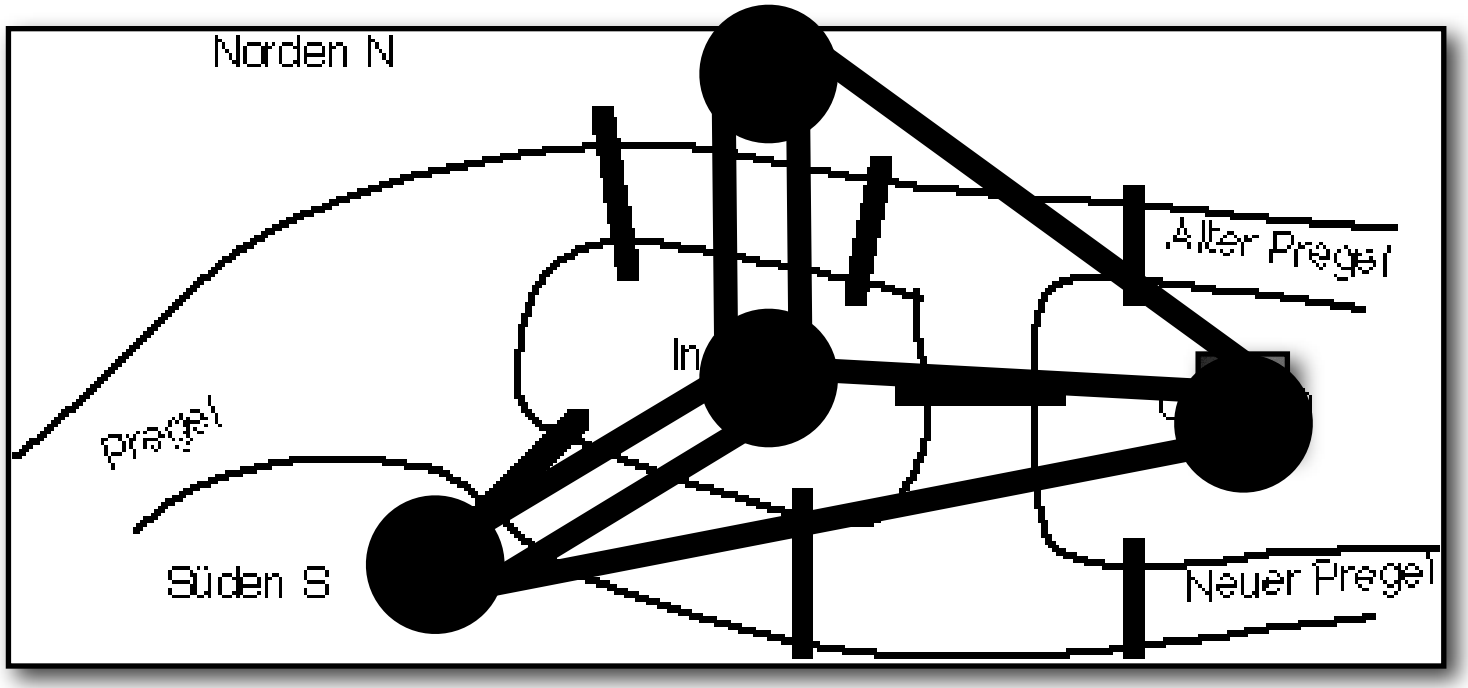




blementierung von Datent.

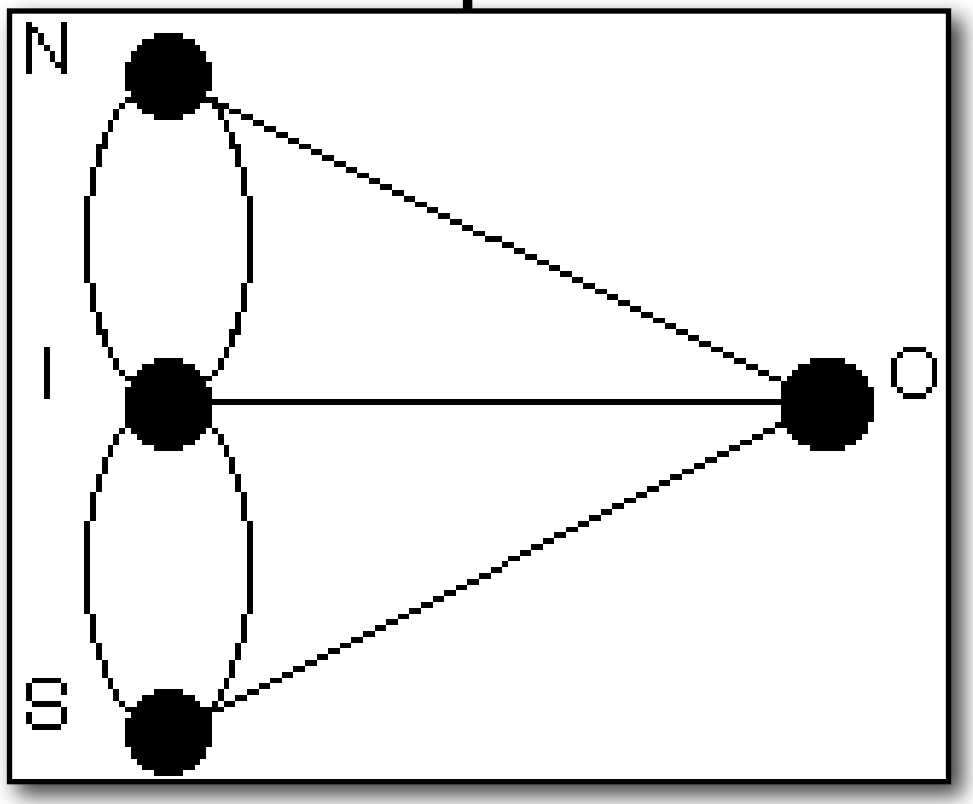
Graphen

- Geschichte: L. Euler (1736): Königsberger Brückenprobl**e**m



Implementierung von Datent.

Graphen



Gibt es zu jedem Knoten einen Weg, der jede Kante genau einmal durchläuft und am Ende wird am Ausgangsknoten ankommt?

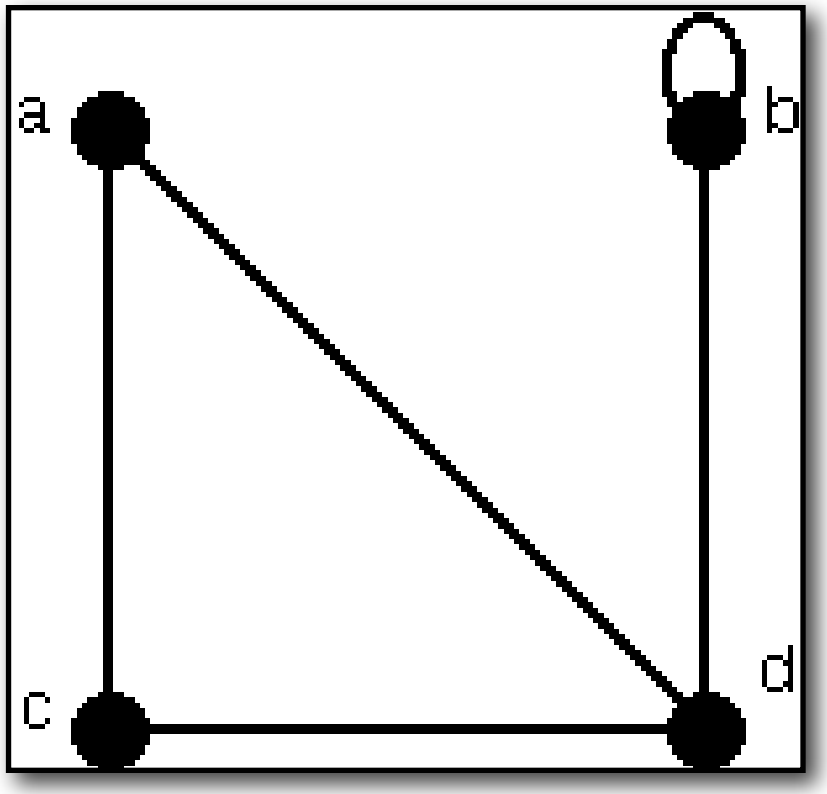
Eulerscher Kreis

Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel



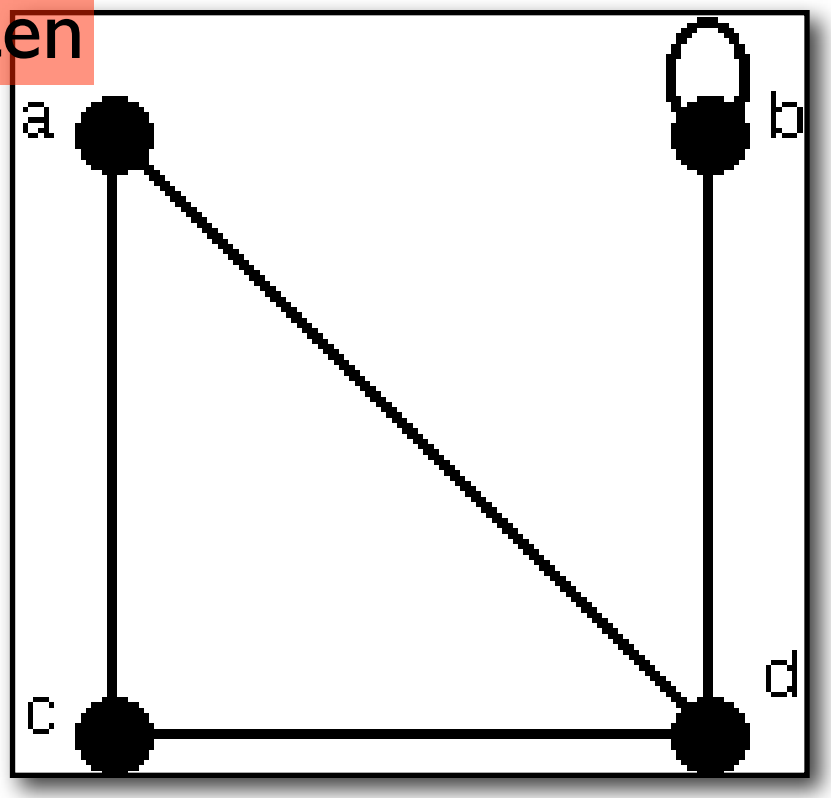
Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

Knoten

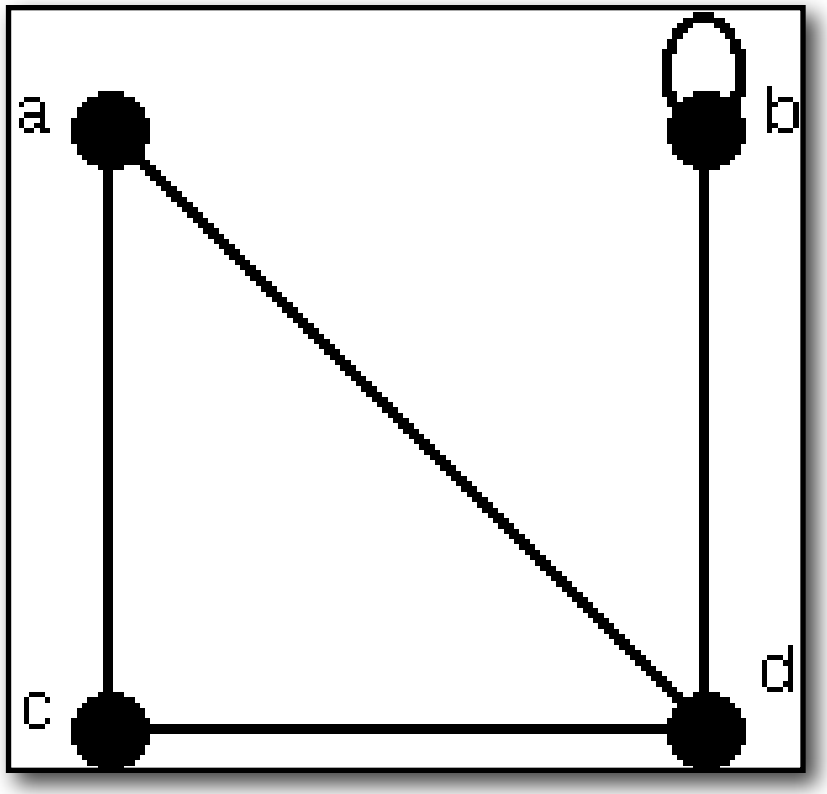


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

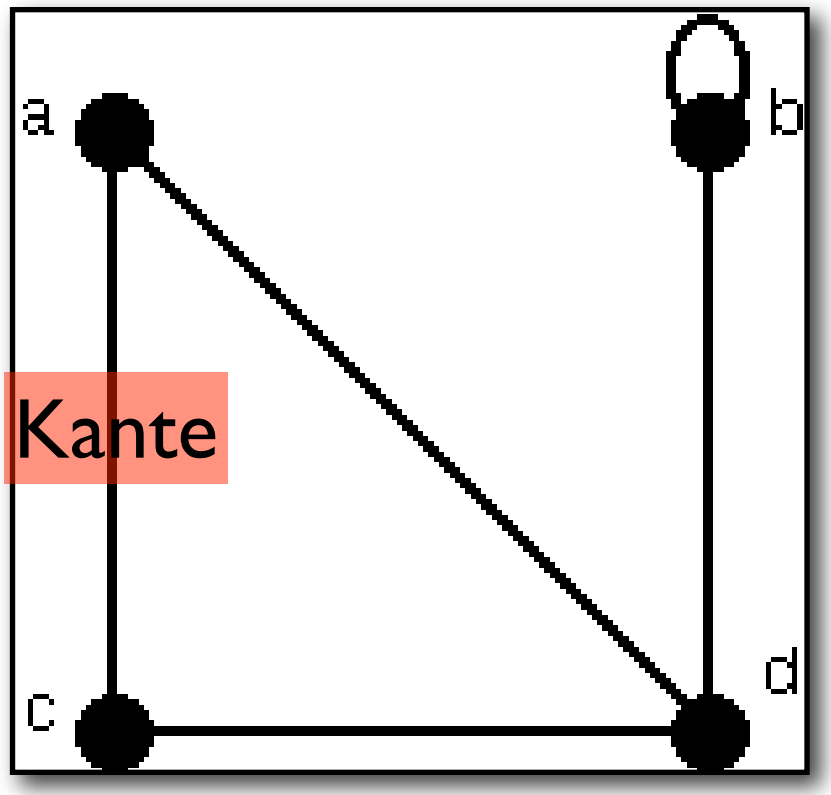


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

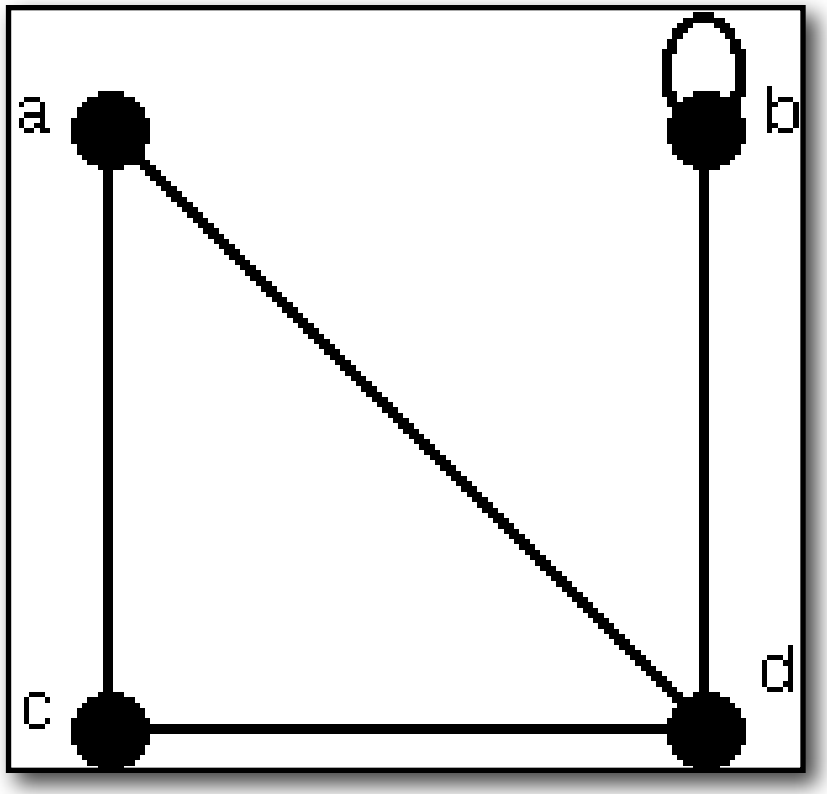


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

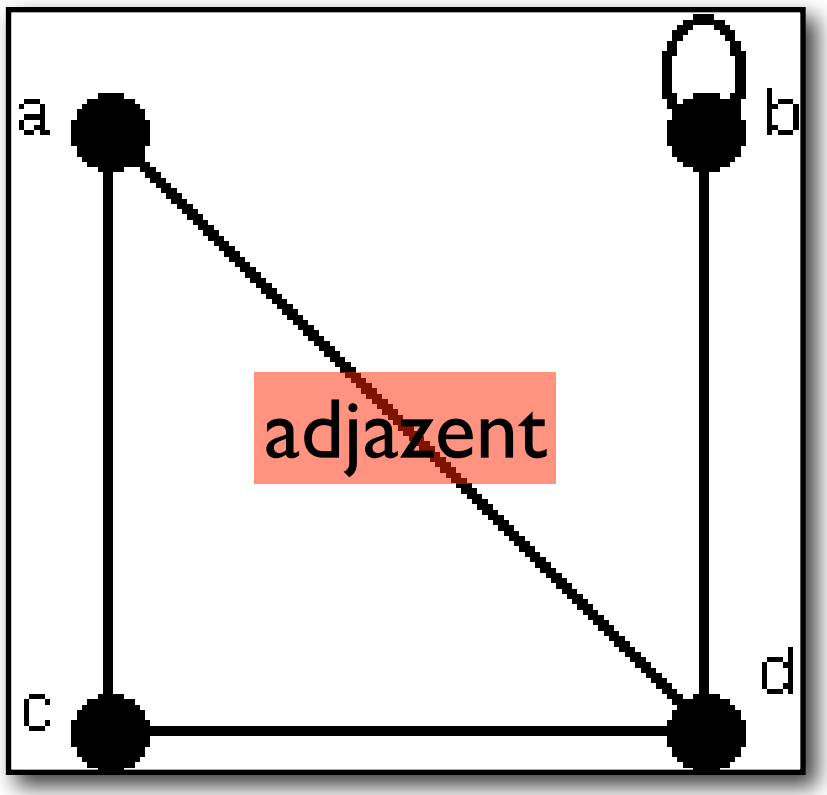


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

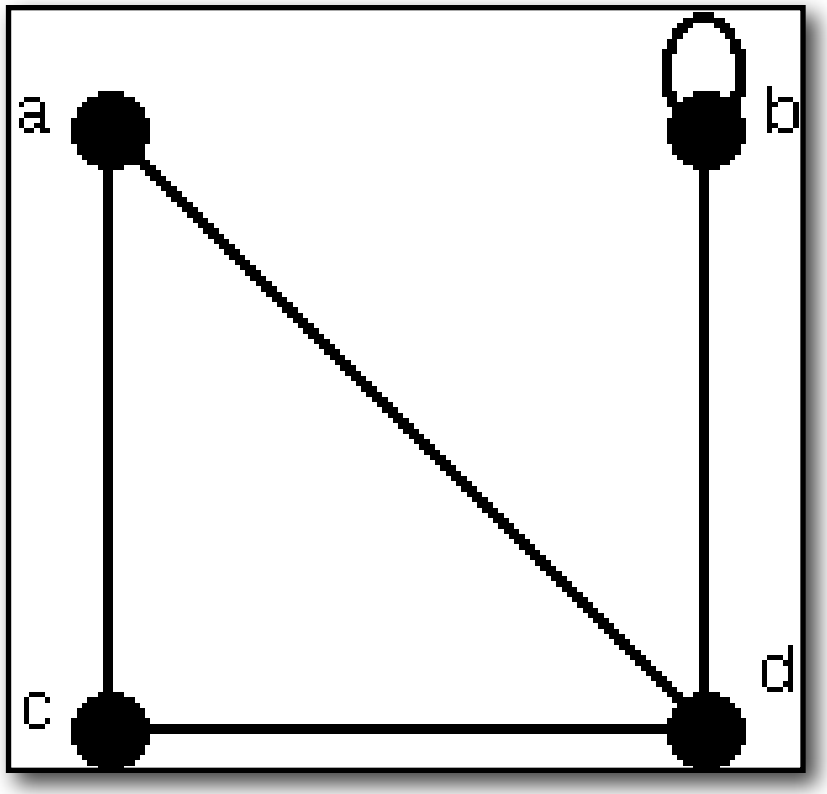


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

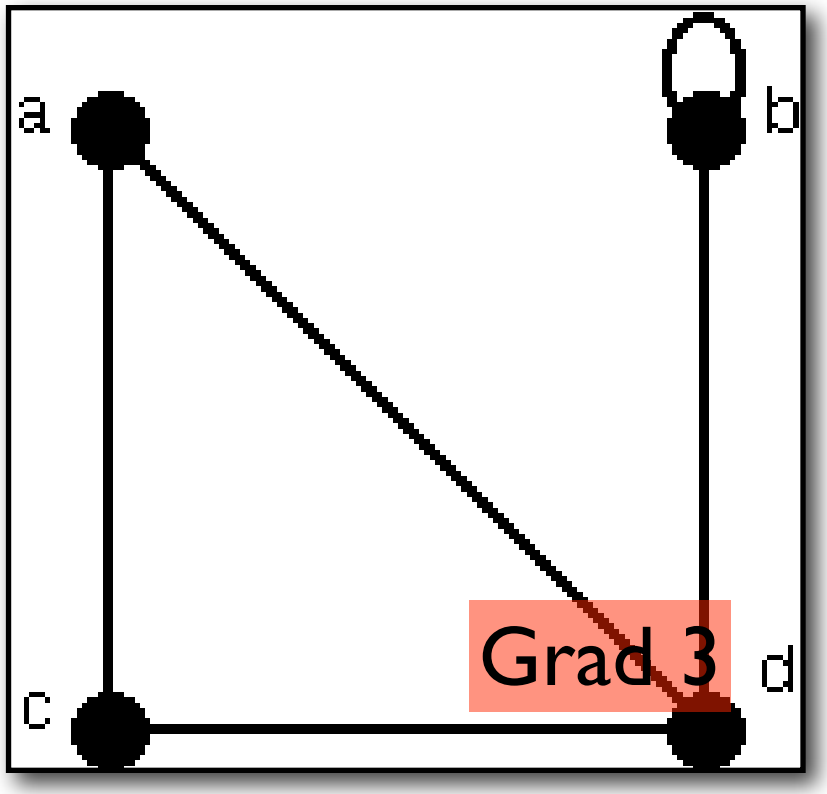
hier nur Begriffe am Beispiel



Implementierung von Datent.
Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

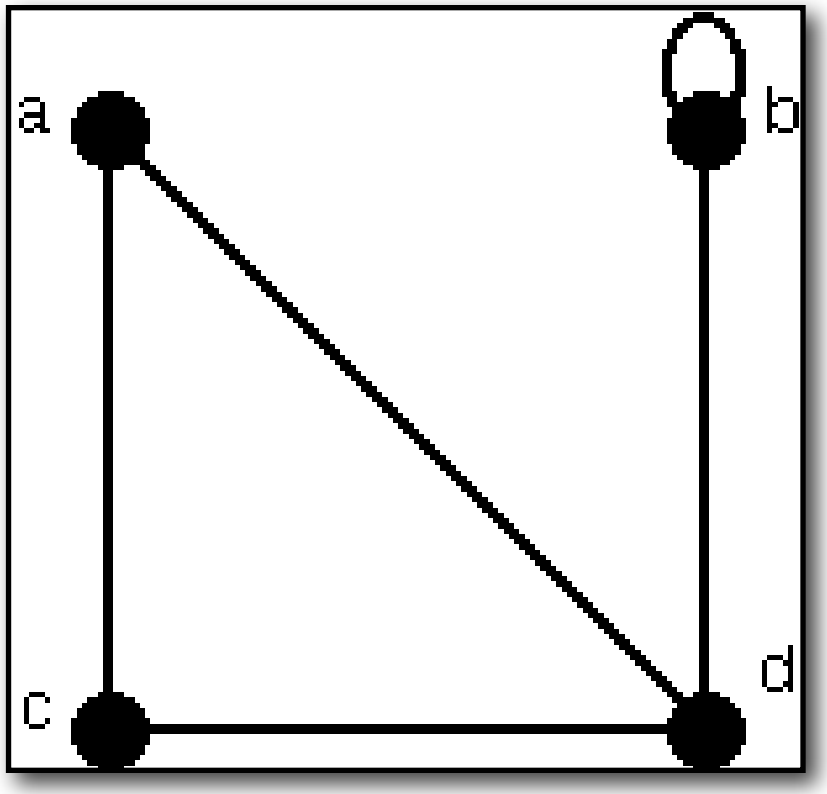


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

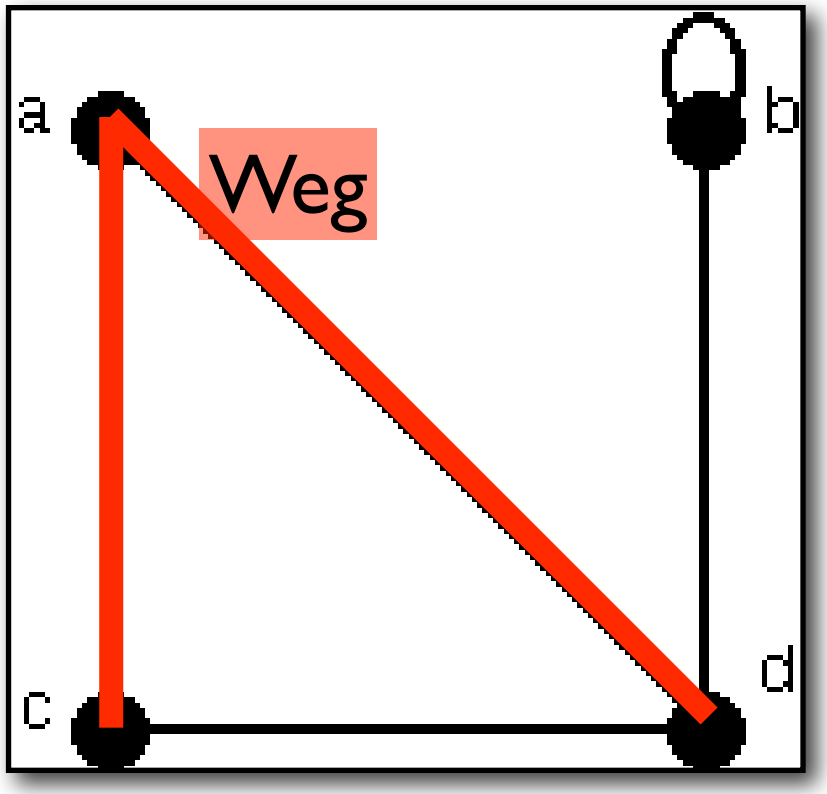


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

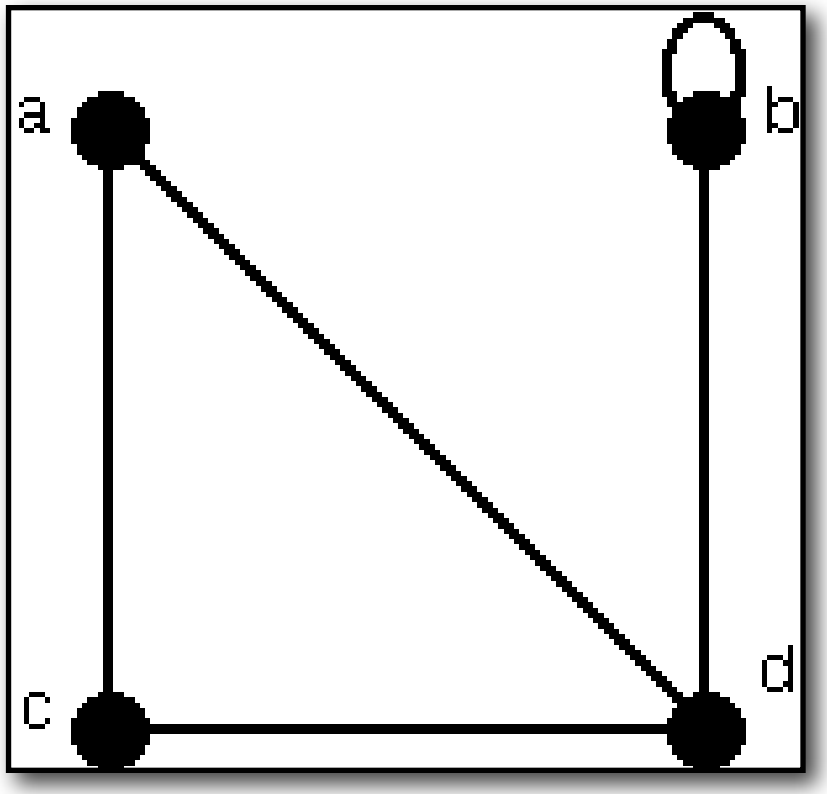


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

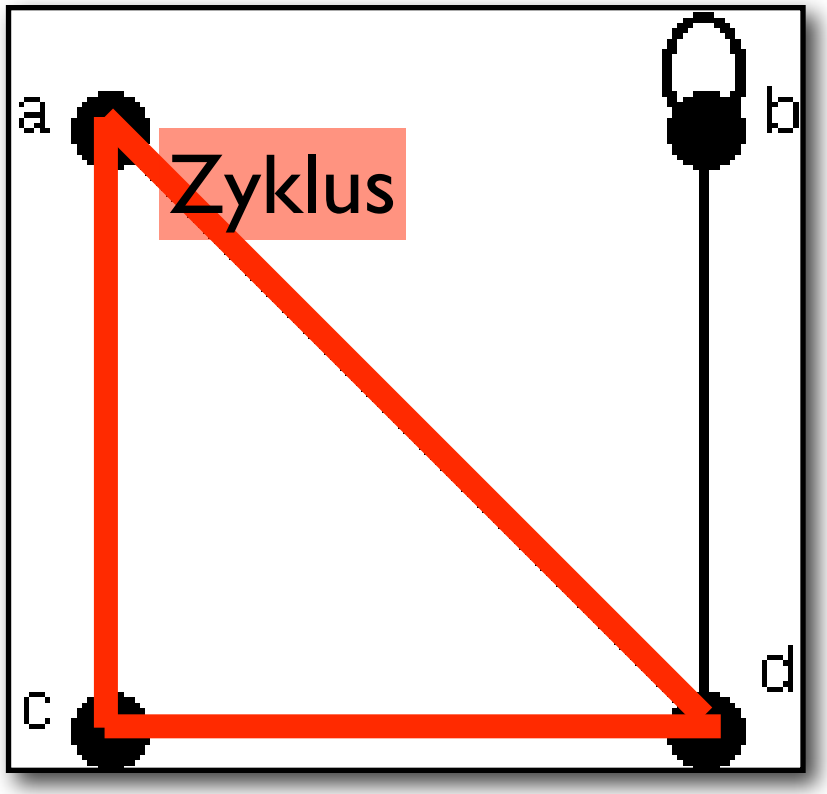


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

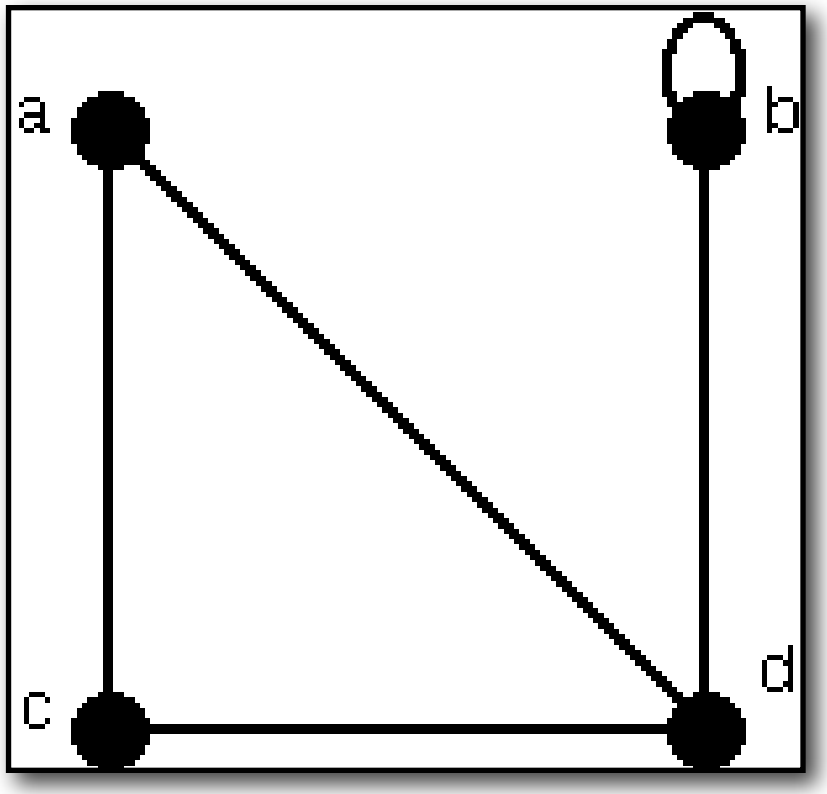


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel



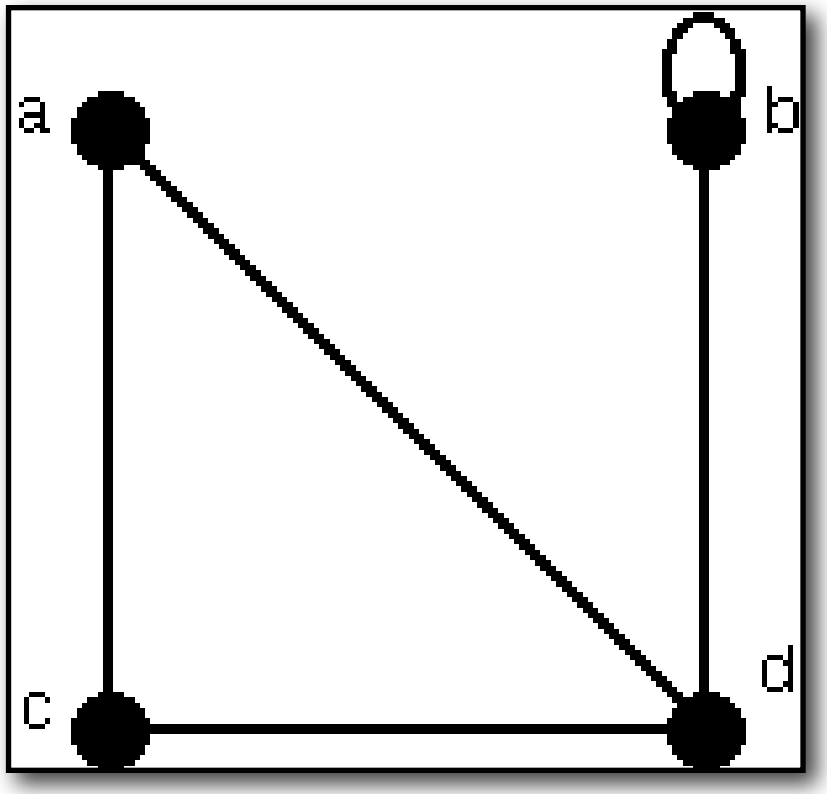
Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

azyklisch

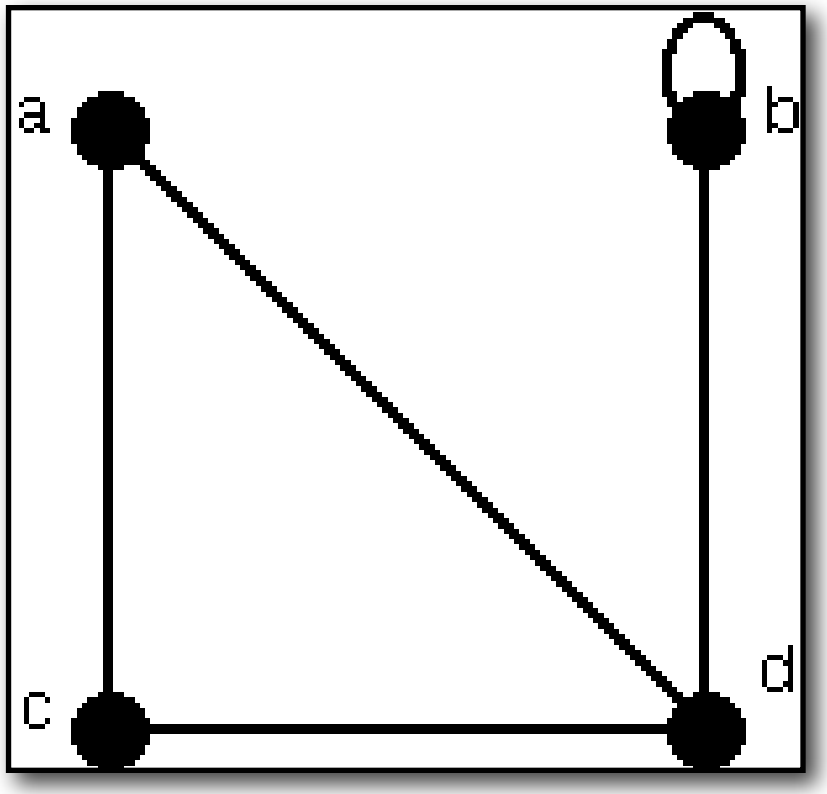


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

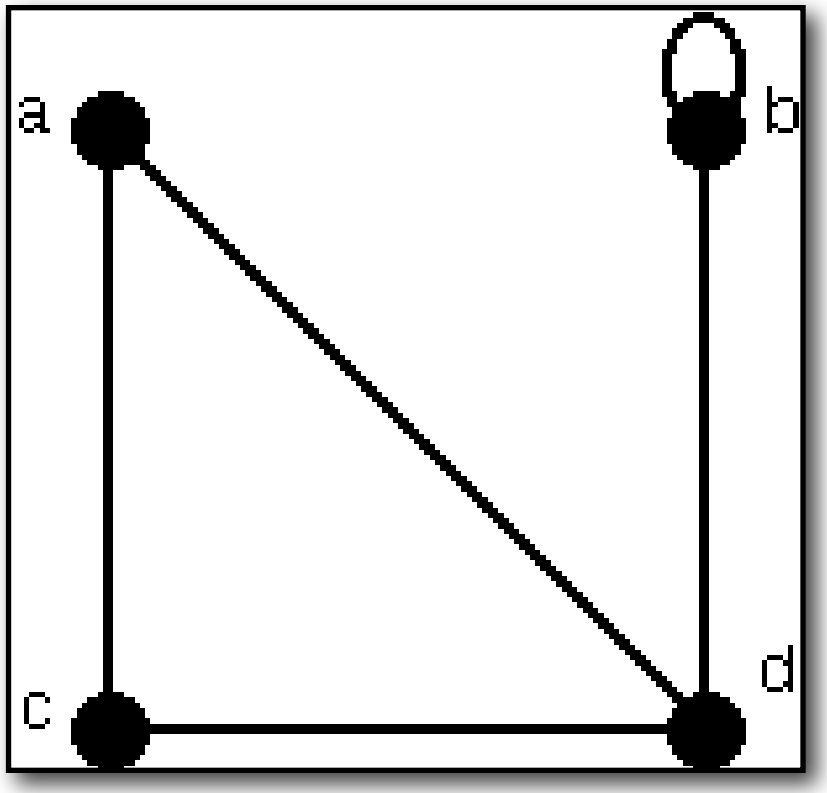


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel



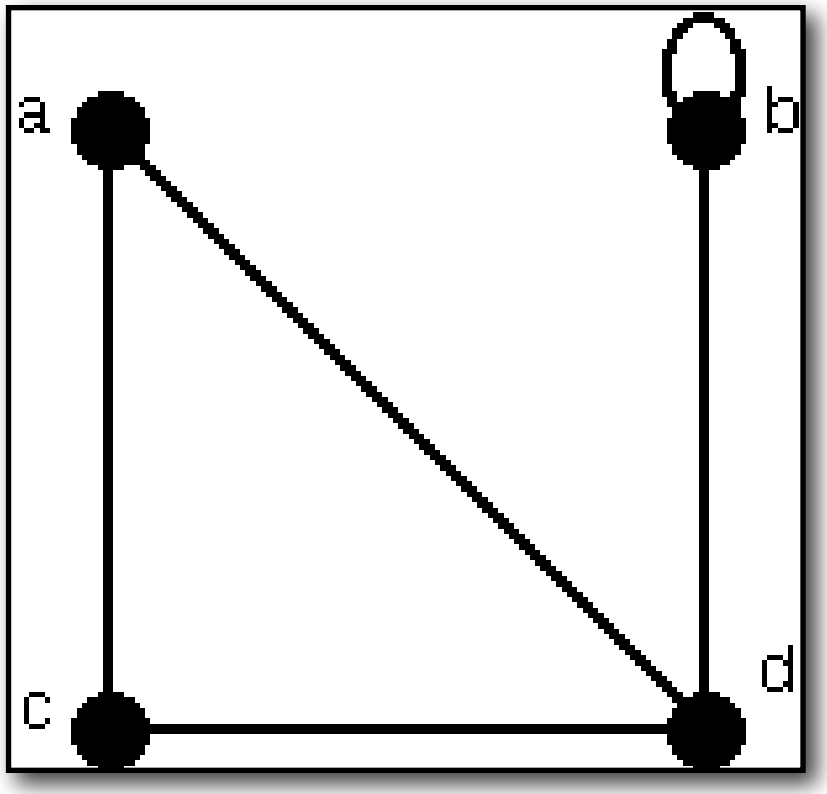
gerichteter Graph

Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel

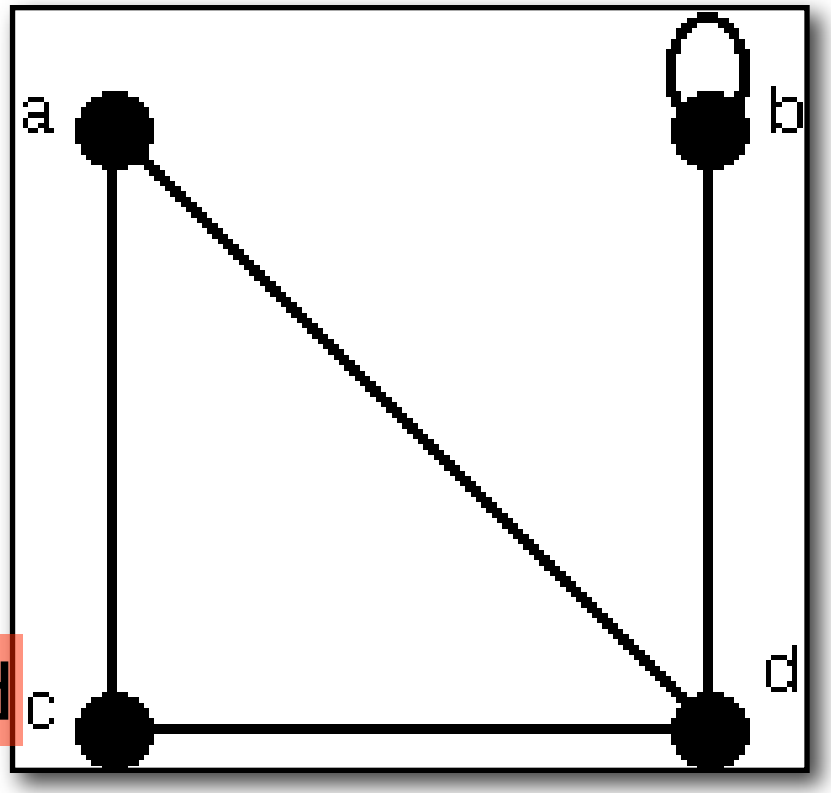


Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel



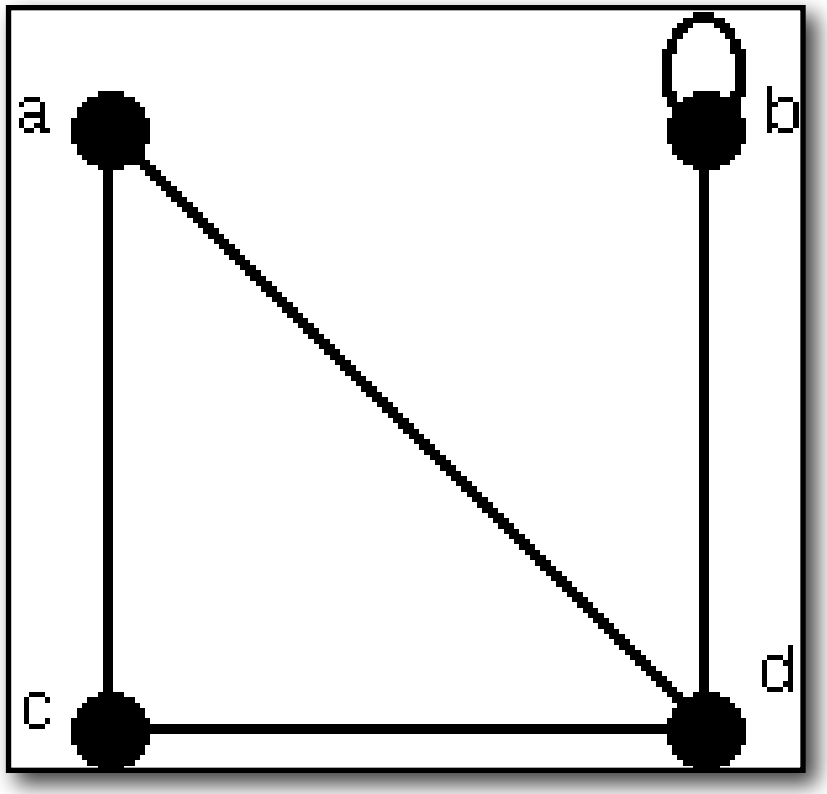
zusammenhängend

Implementierung von Datent.

Graphen - Definition

Definition s. Skript

hier nur Begriffe am Beispiel



Implementierung von Datent.
Graphen - Implementierung

"sequentielle" Implementierung: 2-dim. Array

- Knoten durchnumerieren
- **Adjazenzmatrix** $A=(a_{ij})$ definieren:

$$a_{ij} = \begin{cases} \text{true, falls } \{i,j\} \in E \text{ im ungerichteten Graphen bzw.} \\ (i,j) \in E \text{ im gerichteten Graphen} \\ \\ \text{false, sonst.} \end{cases}$$

Bei ungerichteten Graphen Adjazenzmatrix symmetrisch, d.h. $a_{ij}=a_{ji}$ für alle i,j

Implementierung von Datent.
Graphen - Implementierung

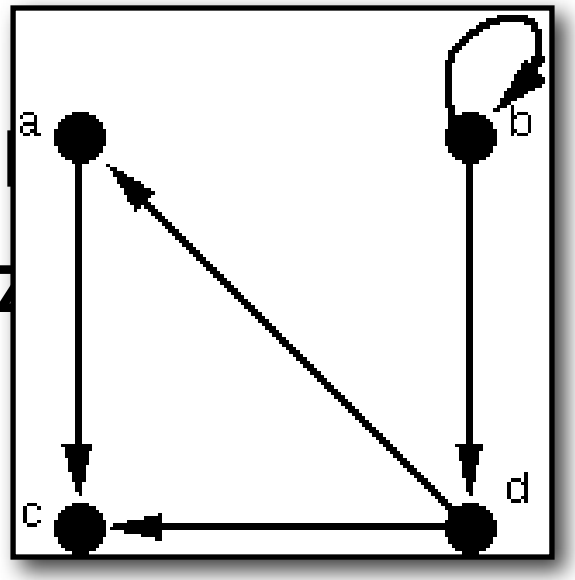
- Vorteil: leichtert Zugriff auf Knoten und Kanten
- Nachteil: Speicherplatzbedarf stets $|V|^2$ (V Knotenmenge, auch z.B. bei $V=\varnothing$)
- Wunsch: Speicherbedarf wächst mit Größe des Graphen, ungefähr $c(|V|+|E|)$ (E Kantenmenge)
- Lösung: verkettete Implementierung durch **Adjazenzlisten**

Implementierung von Datent.
Graphen - Adjazenzlisten

- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  $3(|V|+|E|)$

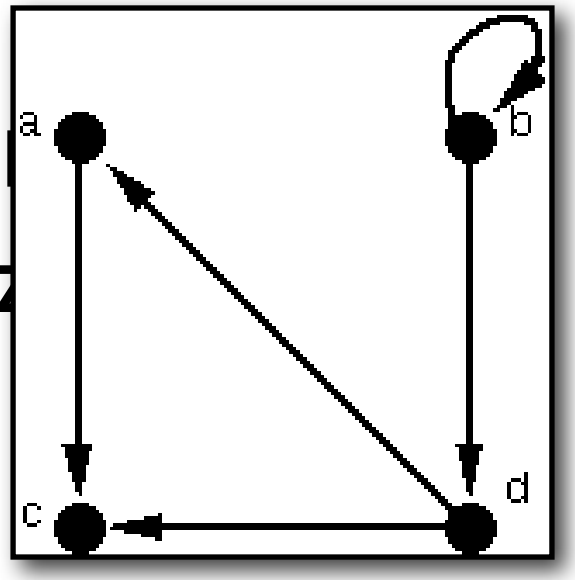
Implementierung vo
Graphen - Adjazenz



- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

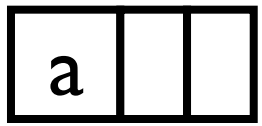
  $3(|V|+|E|)$

Implementierung von Graphen - Adjazenz

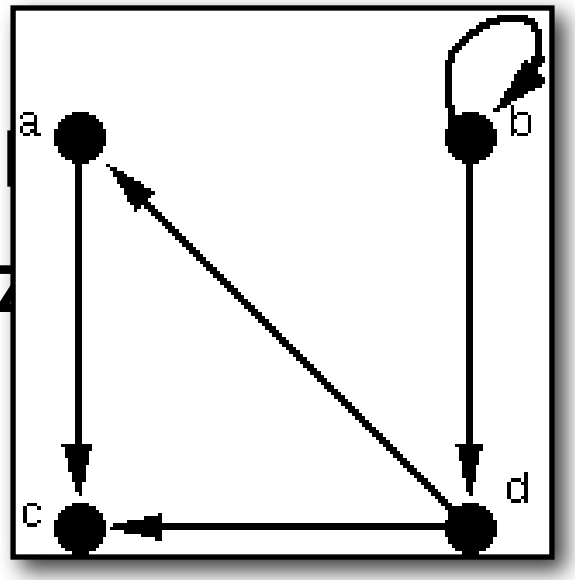


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

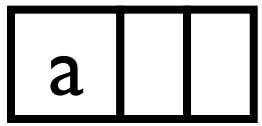
  3(|V|+|E|)

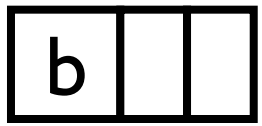


Implementierung vo
Graphen - Adjazenz



- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten

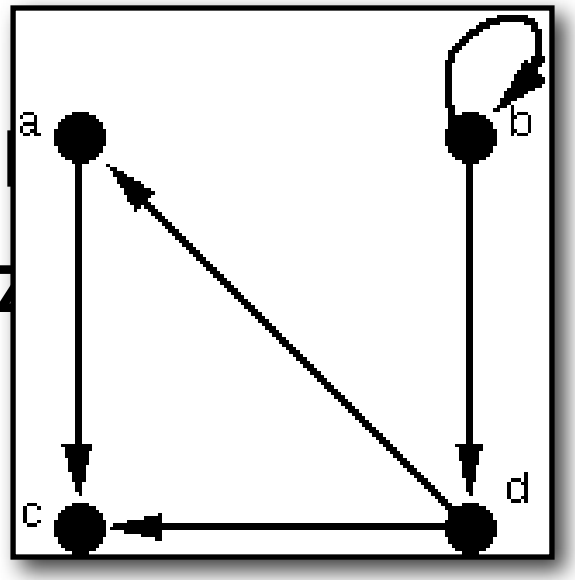




- je Knoten 3 Zellen, je Kante 2 Zellen

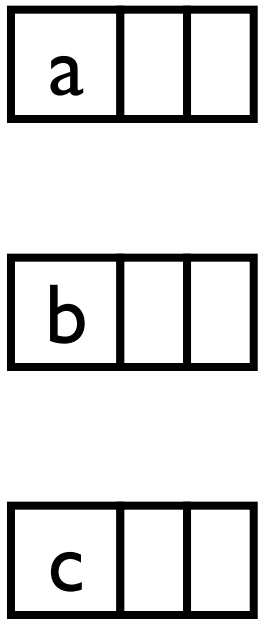
  $3(|V|+|E|)$

Implementierung vo
Graphen - Adjazenz

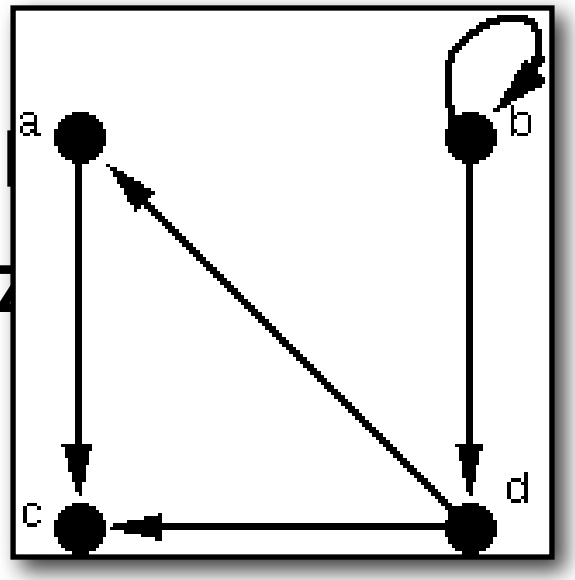


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

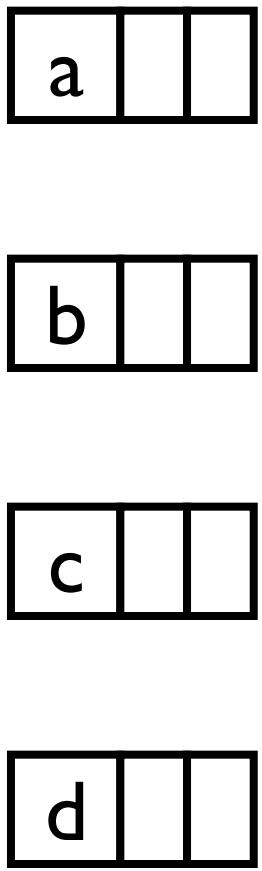


Implementierung vo
Graphen - Adjazenz

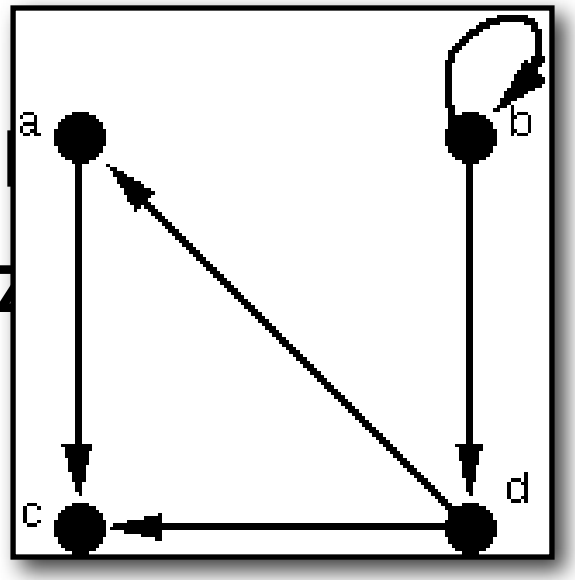


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

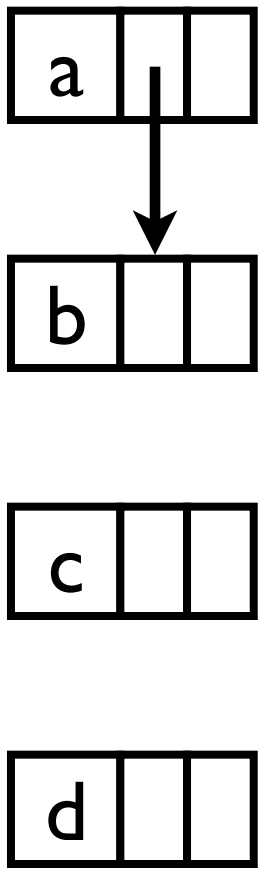


Implementierung vo
Graphen - Adjazenz

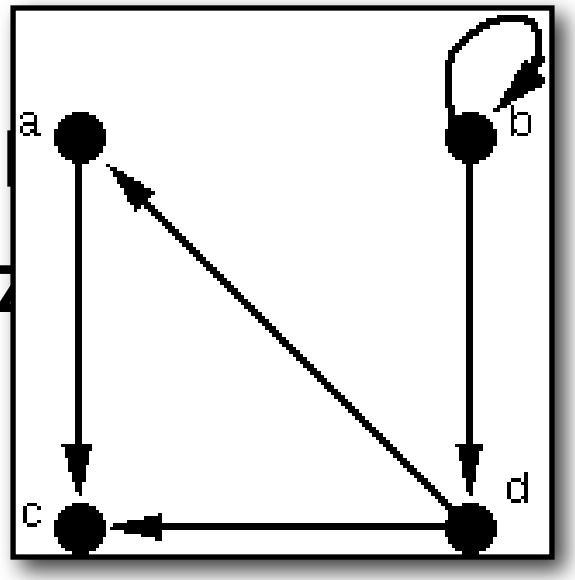


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

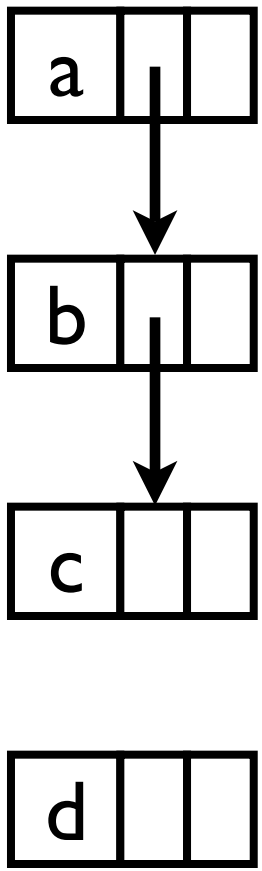


Implementierung vo
Graphen - Adjazenz

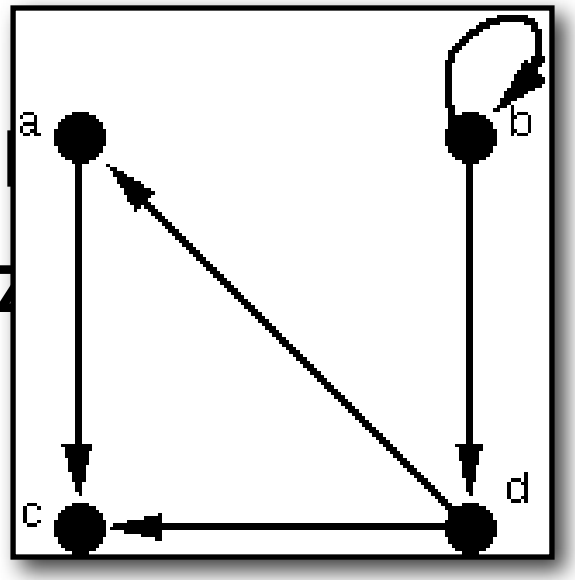


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

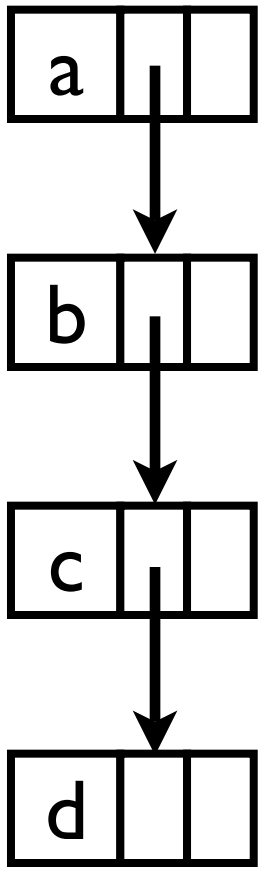


Implementierung vo
Graphen - Adjazenz

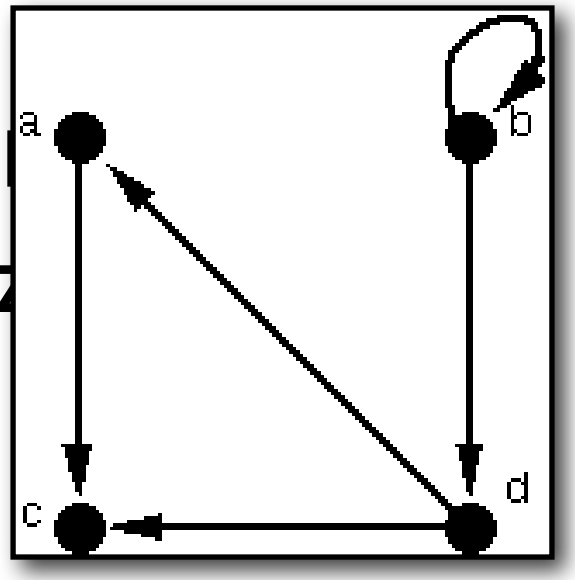


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

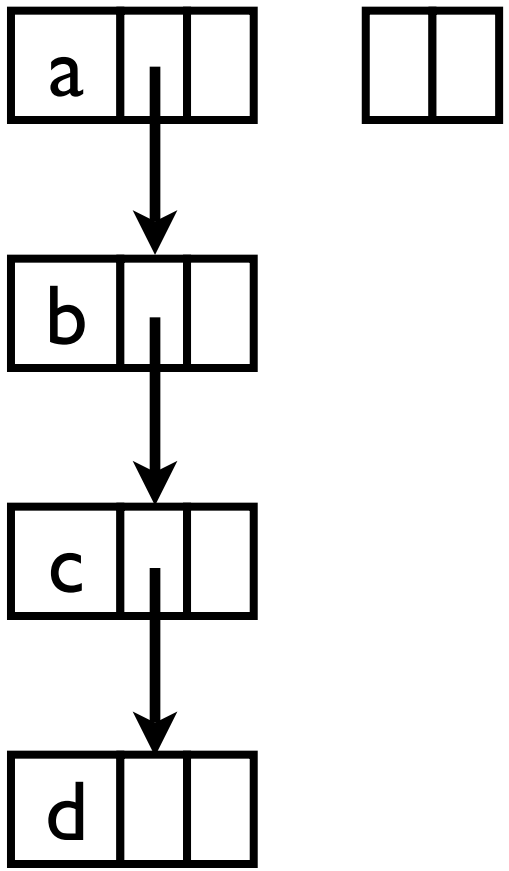


Implementierung vo
Graphen - Adjazenz

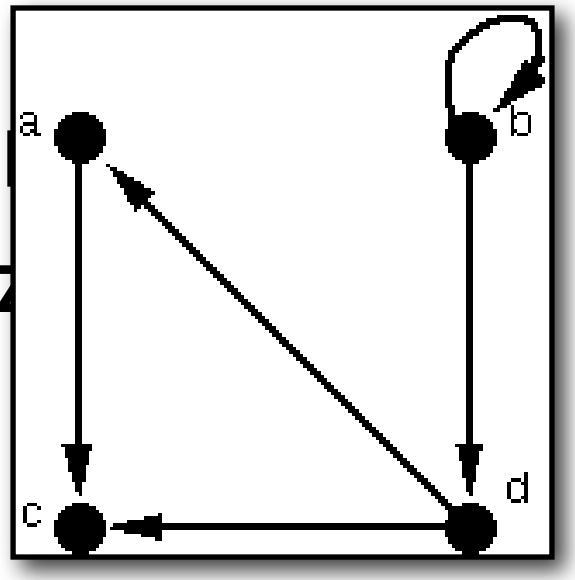


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

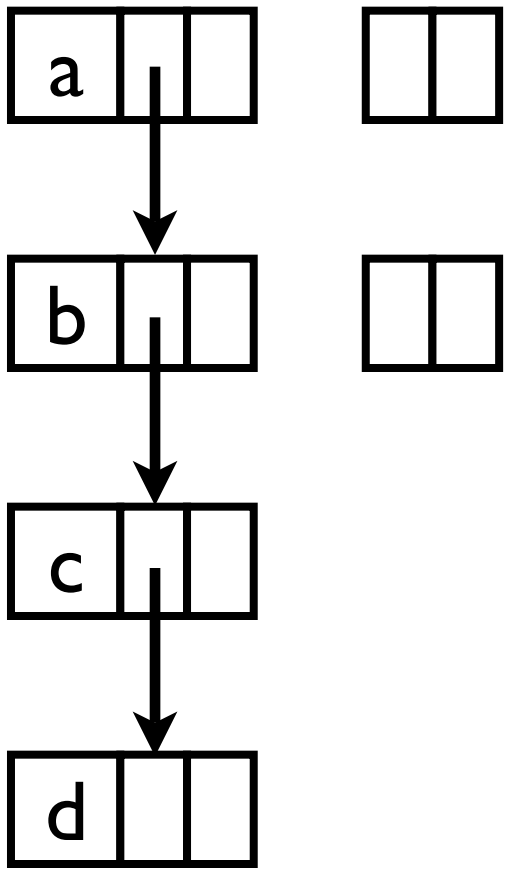


Implementierung vo
Graphen - Adjazenz

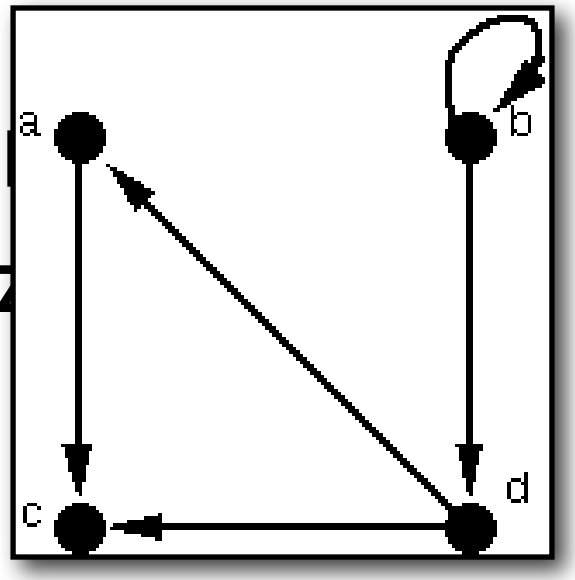


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

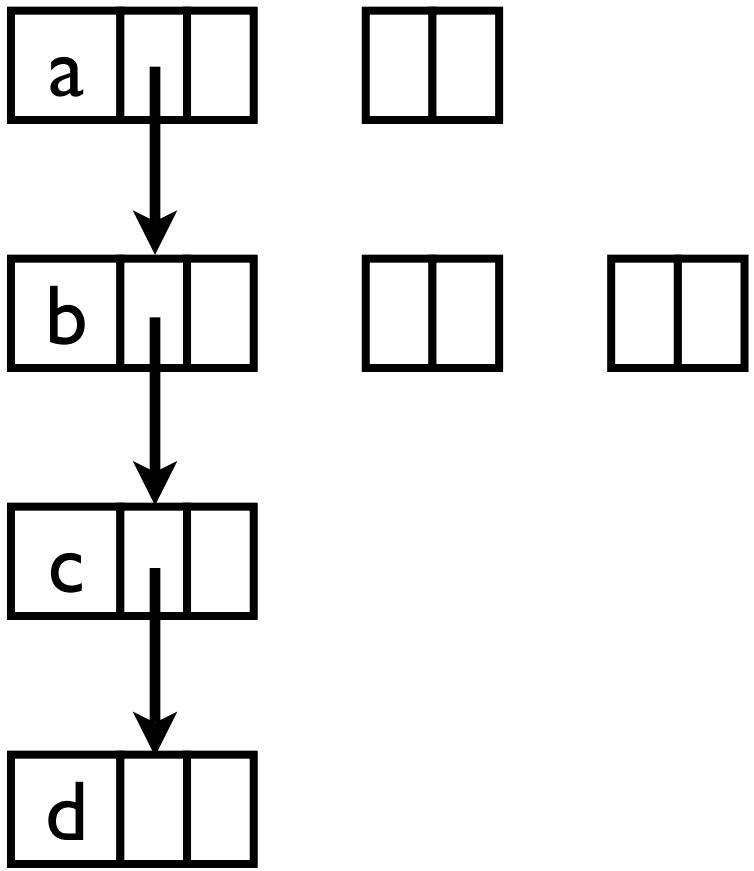


Implementierung vo
Graphen - Adjazenz

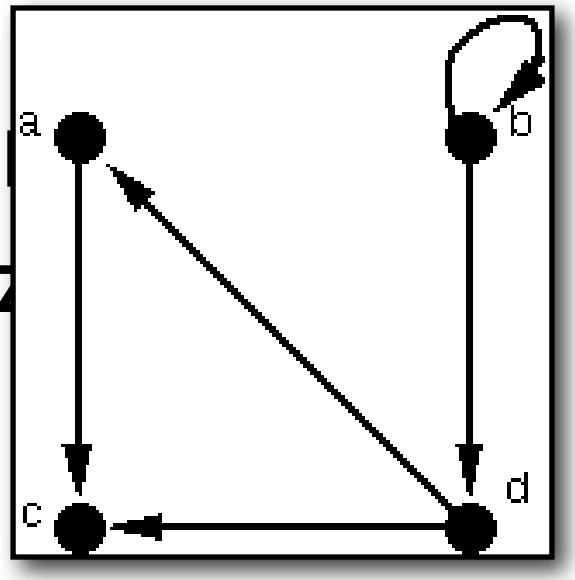


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

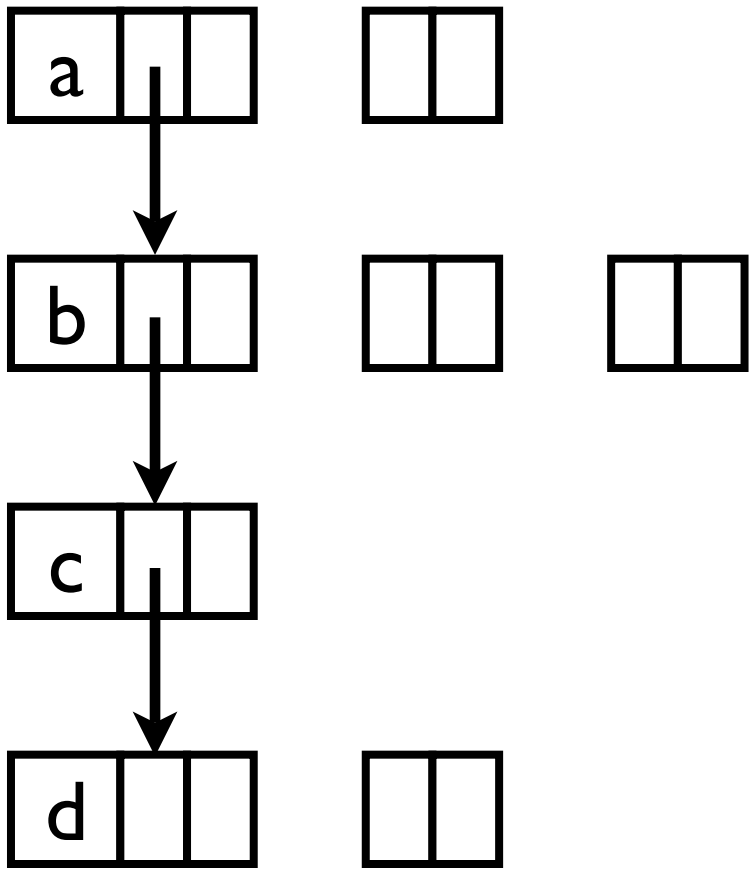


Implementierung vo
Graphen - Adjazenz

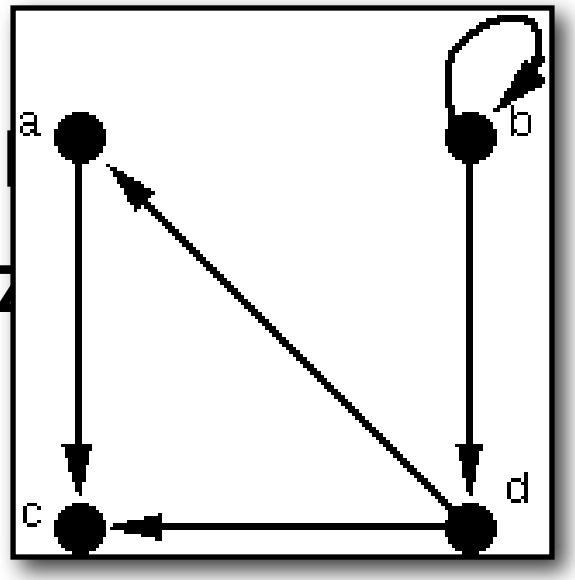


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

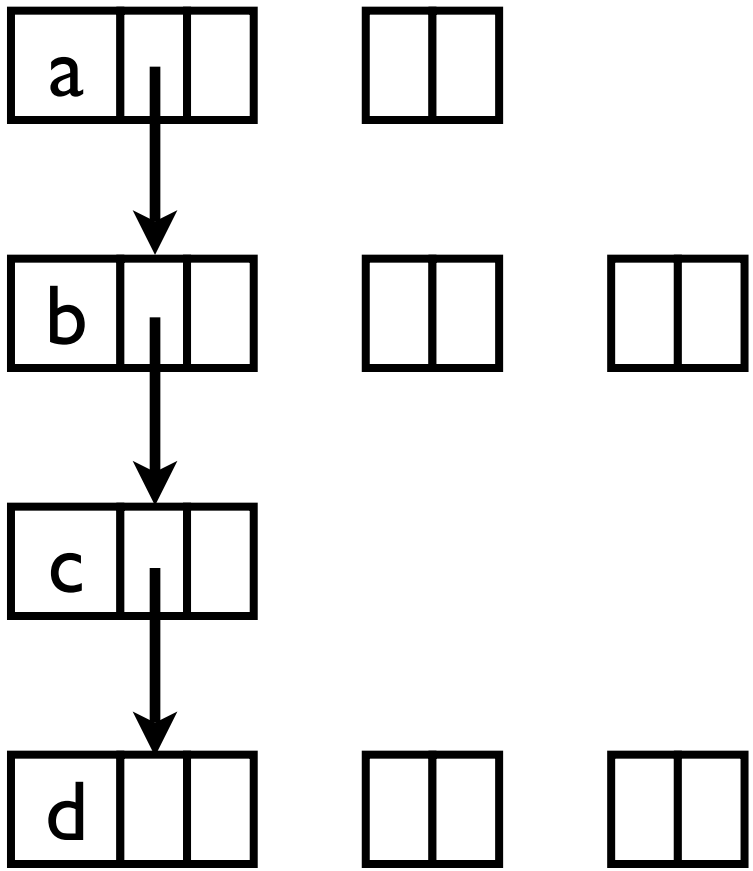


Implementierung vo
Graphen - Adjazenz

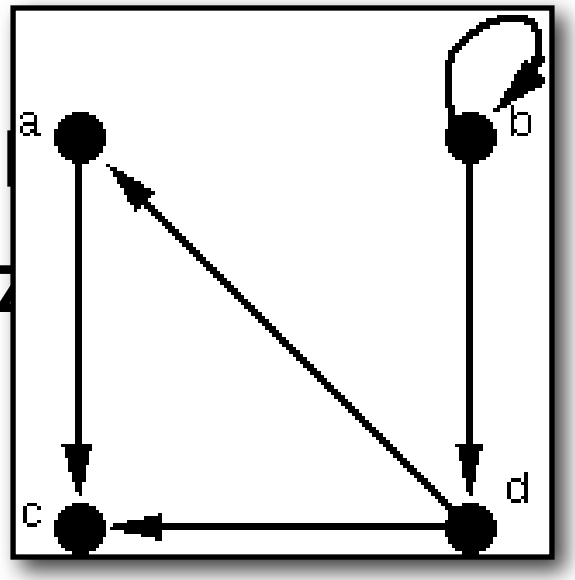


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

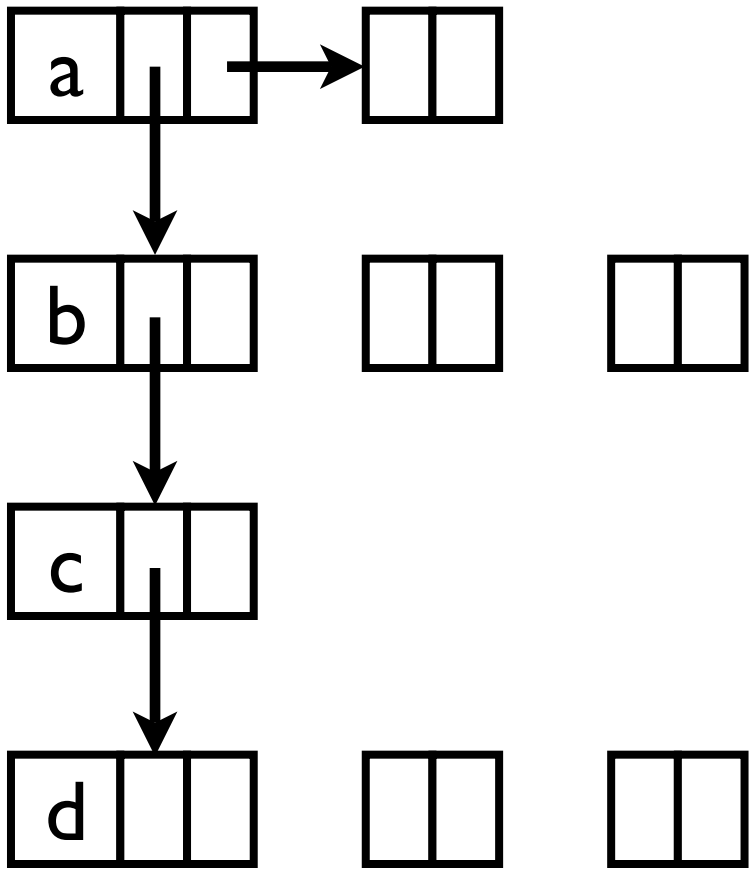


Implementierung vo
Graphen - Adjazenz

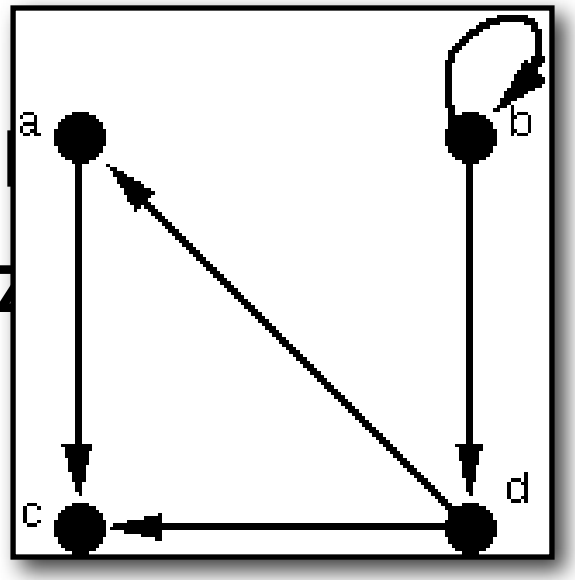


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

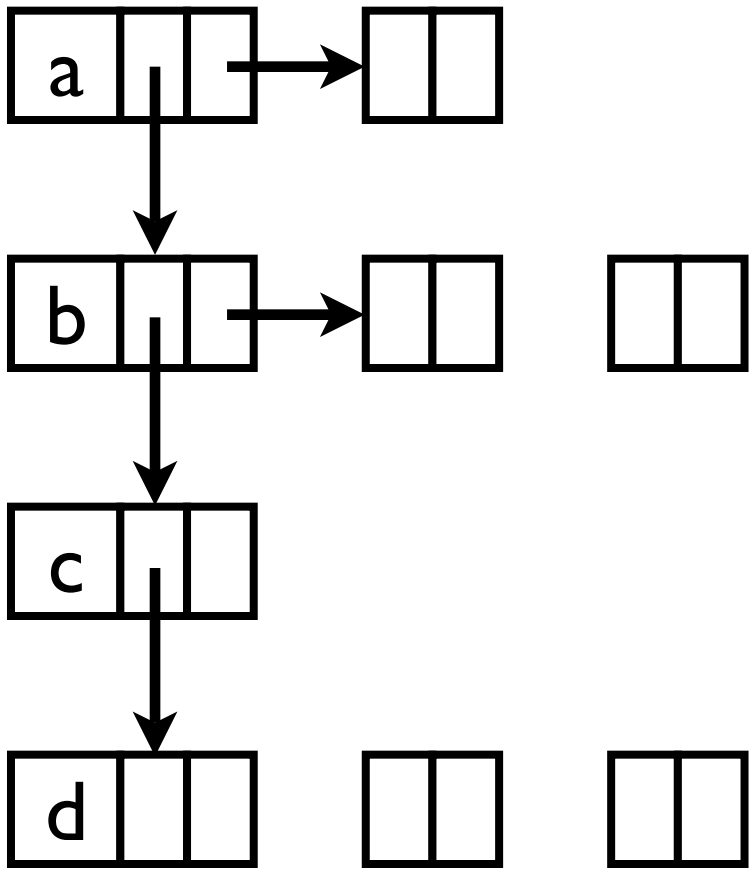


Implementierung vo
Graphen - Adjazenz

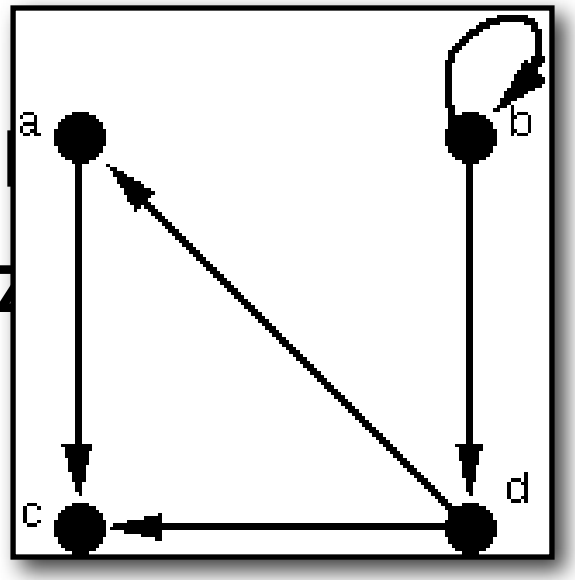


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

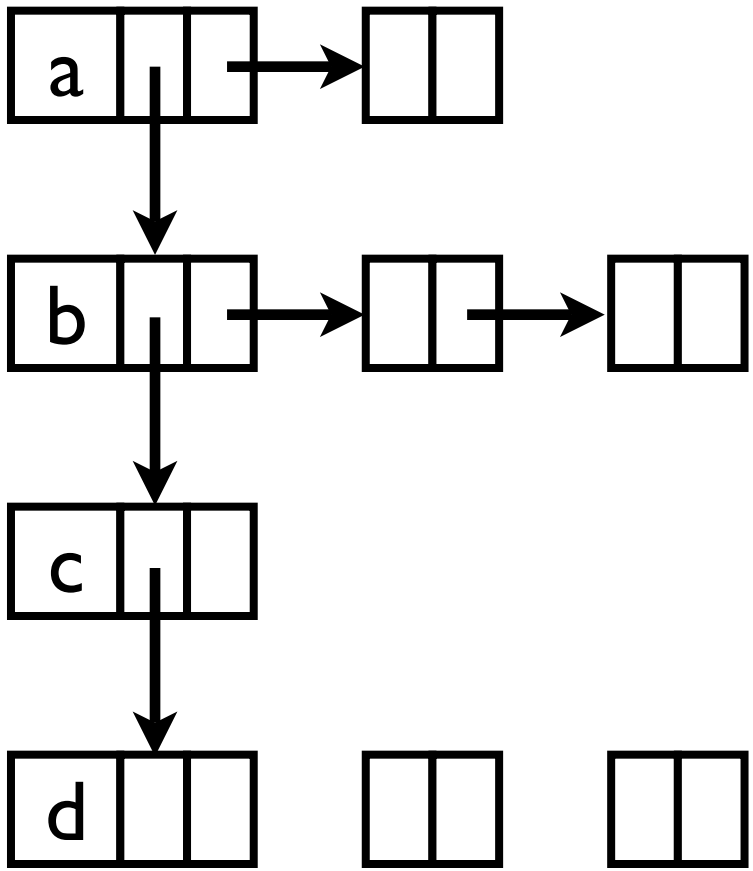


Implementierung vo
Graphen - Adjazenz

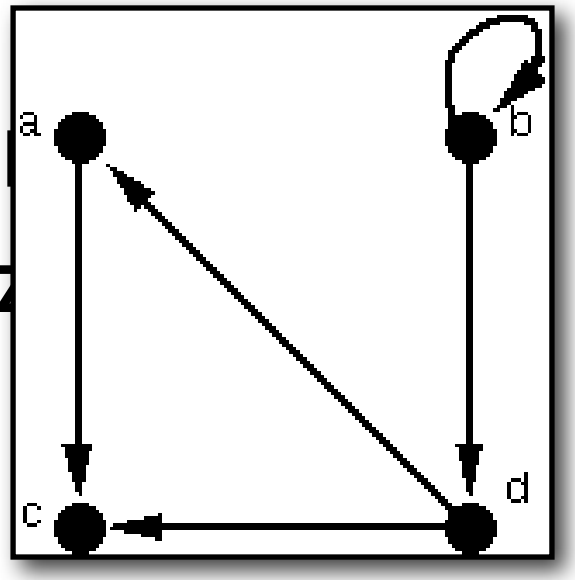


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

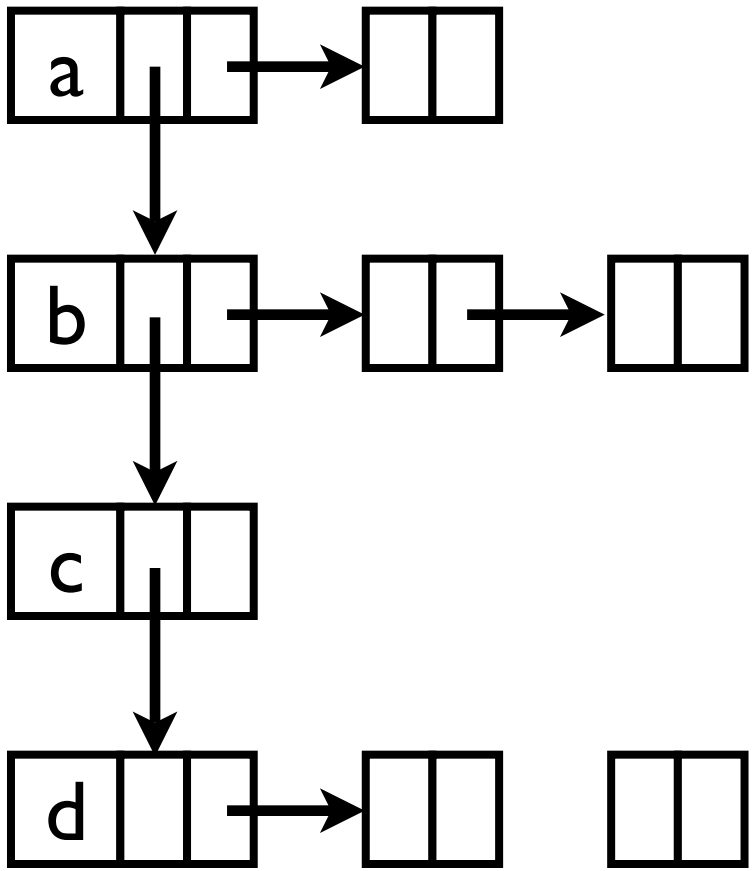


Implementierung vo
Graphen - Adjazenz

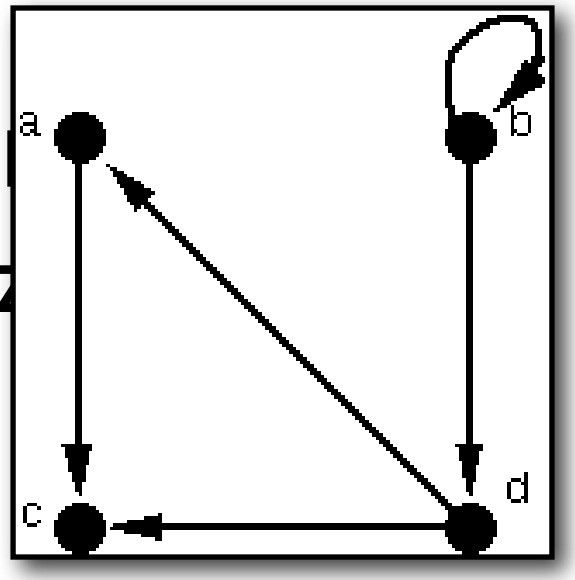


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

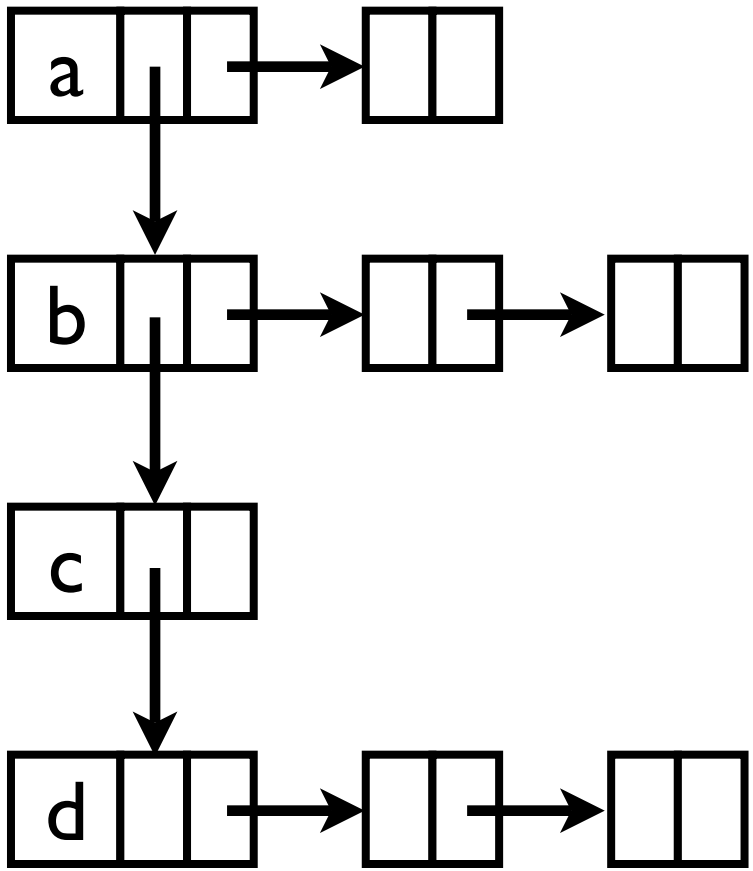


Implementierung vo
Graphen - Adjazenz

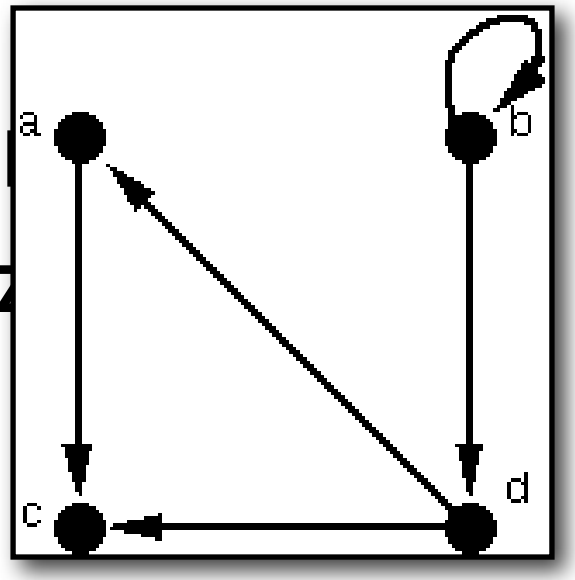


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

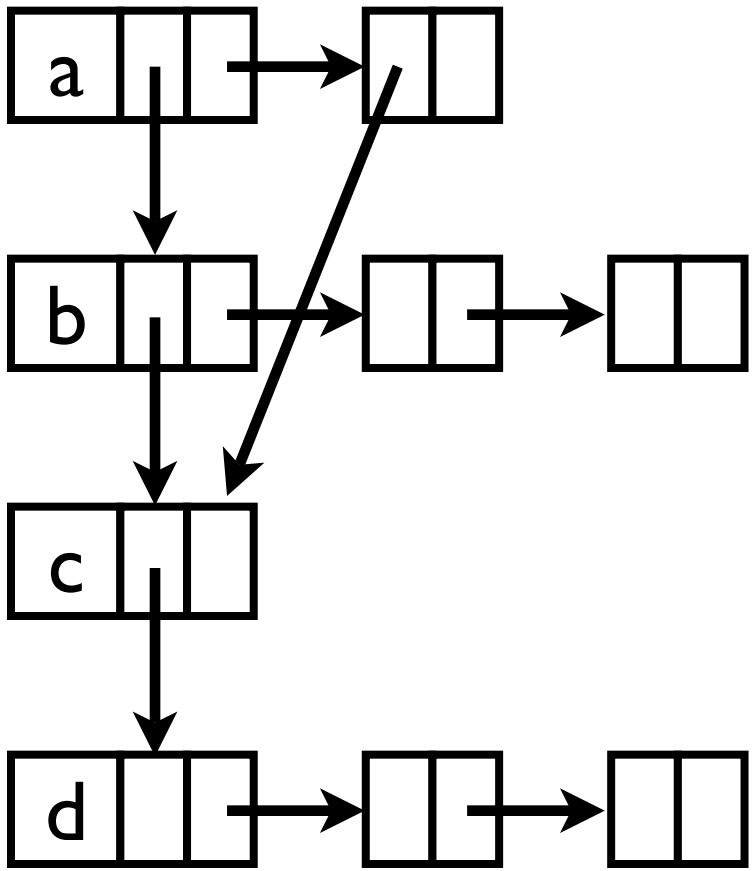


Implementierung vo
Graphen - Adjazenz

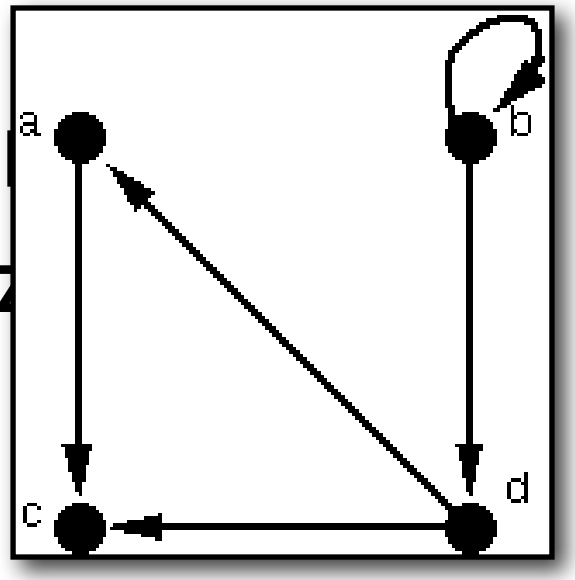


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

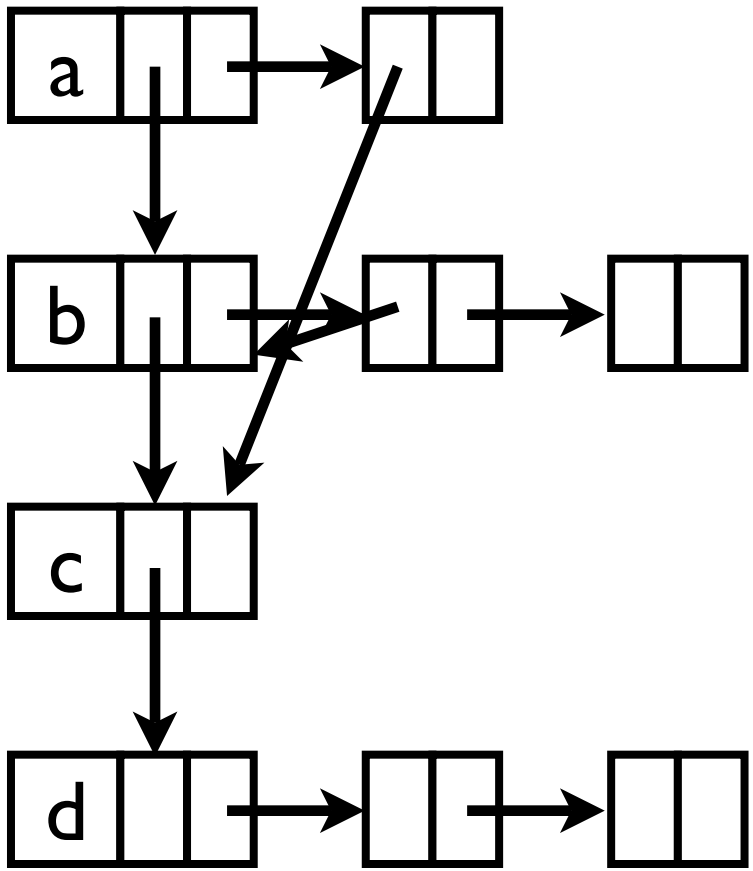


Implementierung vo
Graphen - Adjazenz

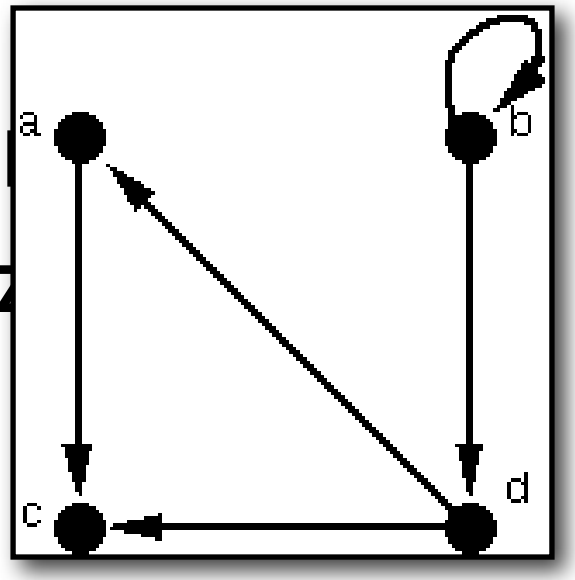


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

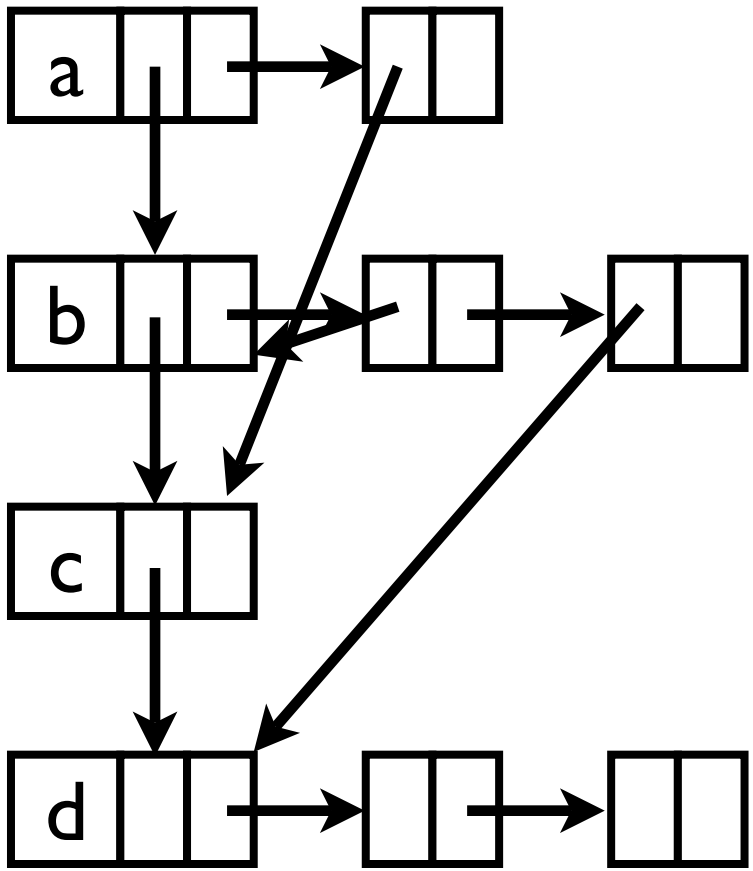


Implementierung vo
Graphen - Adjazenz

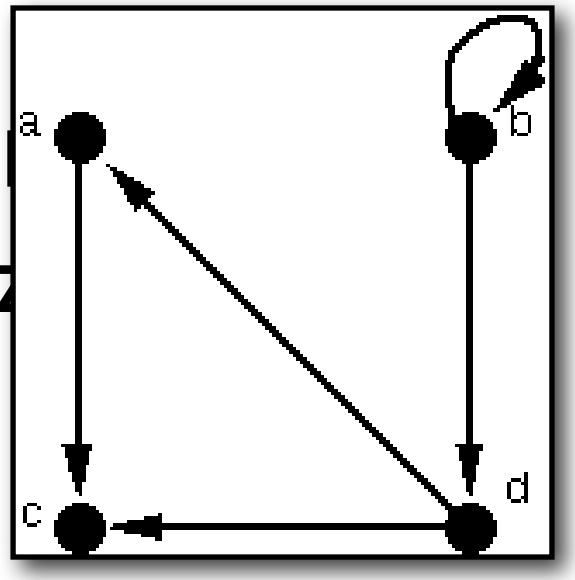


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

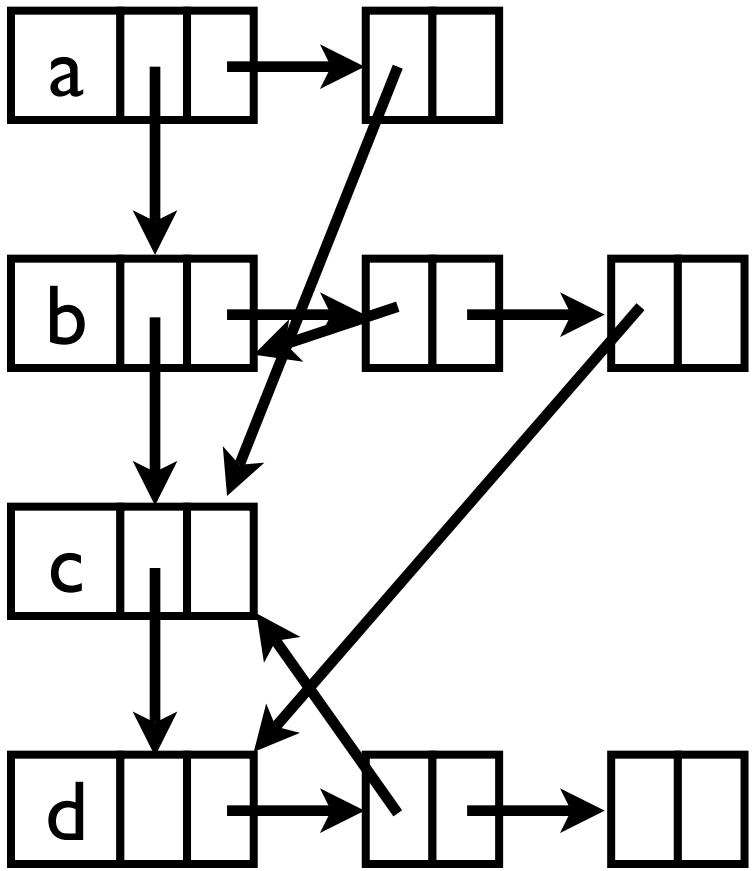


Implementierung vo
Graphen - Adjazenz

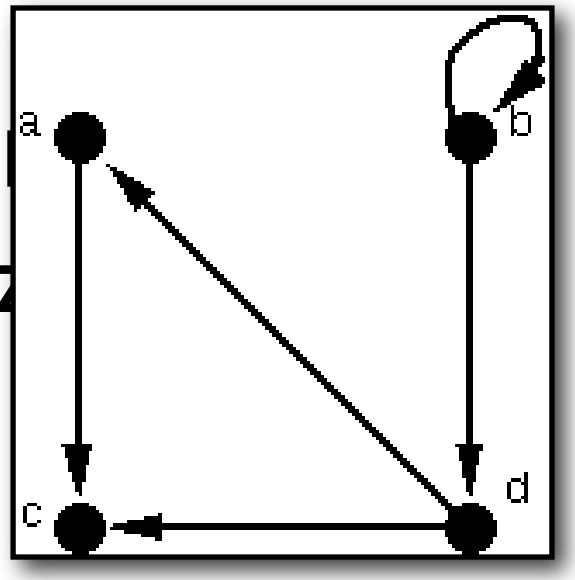


- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

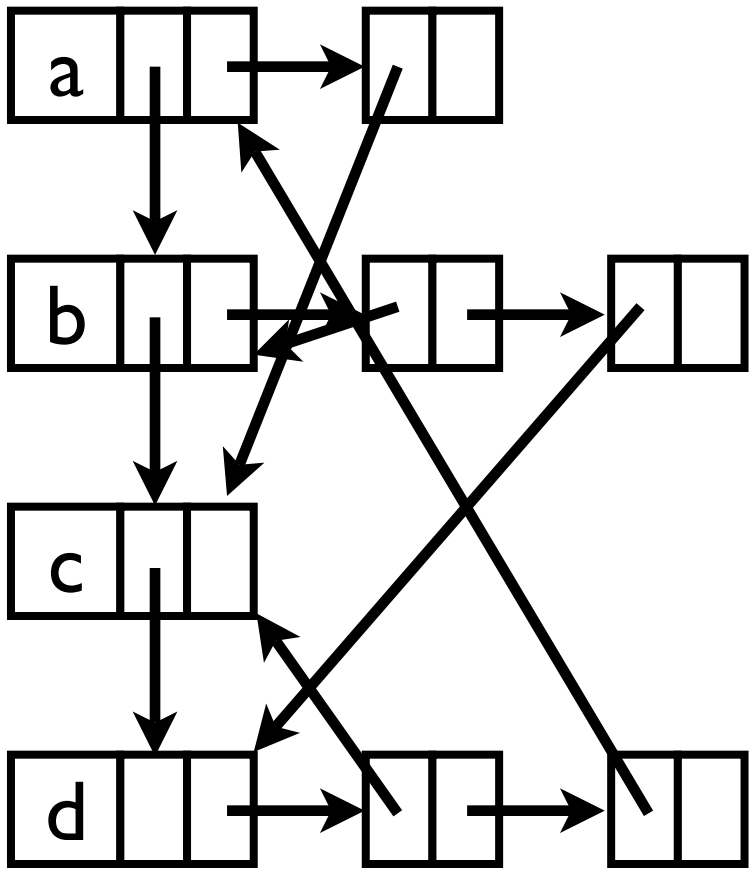


Implementierung vo
Graphen - Adjazenz



- Vorgehen:
  - lineare Liste von Knoten
  - je Knoten eine lineare Liste von Verweisen auf die zu diesem adjazenten Knoten
- je Knoten 3 Zellen, je Kante 2 Zellen

  3(|V|+|E|)

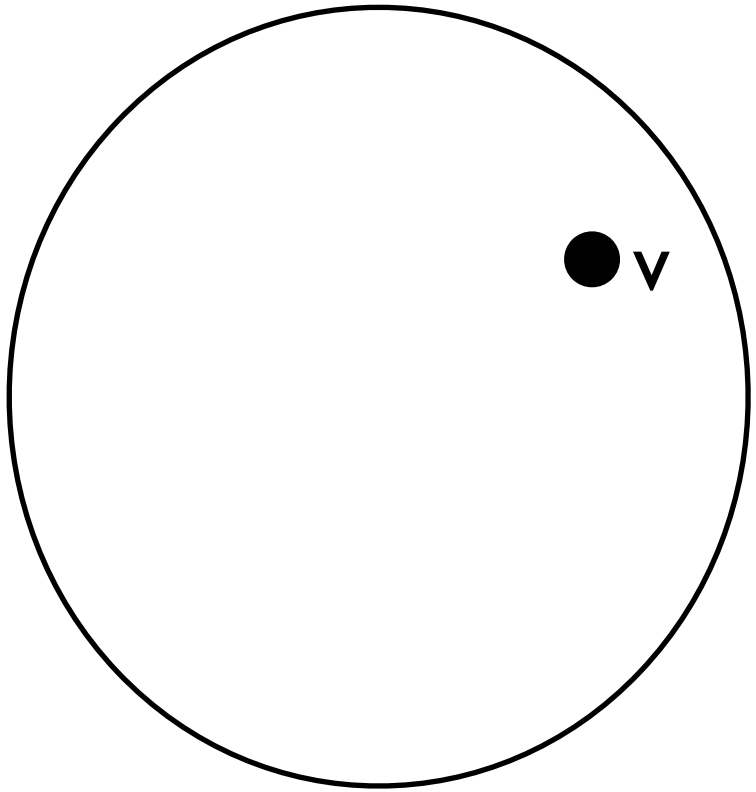


Implementierung von Datent.
Graphen - Durchlaufen

- systematisches Durchsuchen eines Graphen und Ausgabe aller gespeicherten Daten (Knotenmarkierungen)
- Gegeben: zusammenhängender Graph, Startknoten v
- Zwei bekannte Verfahren
  - Tiefendurchlauf (depth first search, dfs)
  - Breitendurchlauf (breadth first search, bfs)

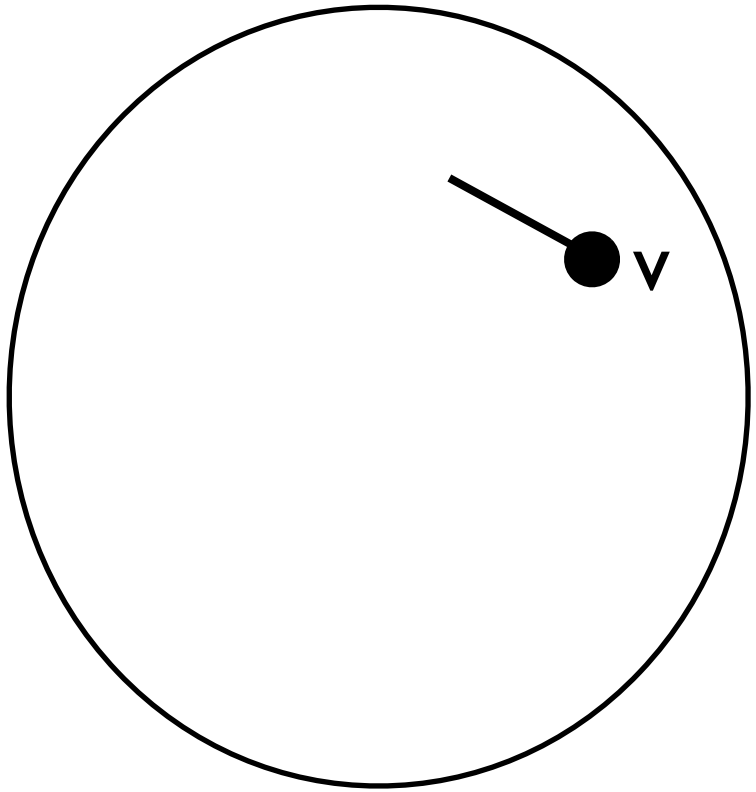
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



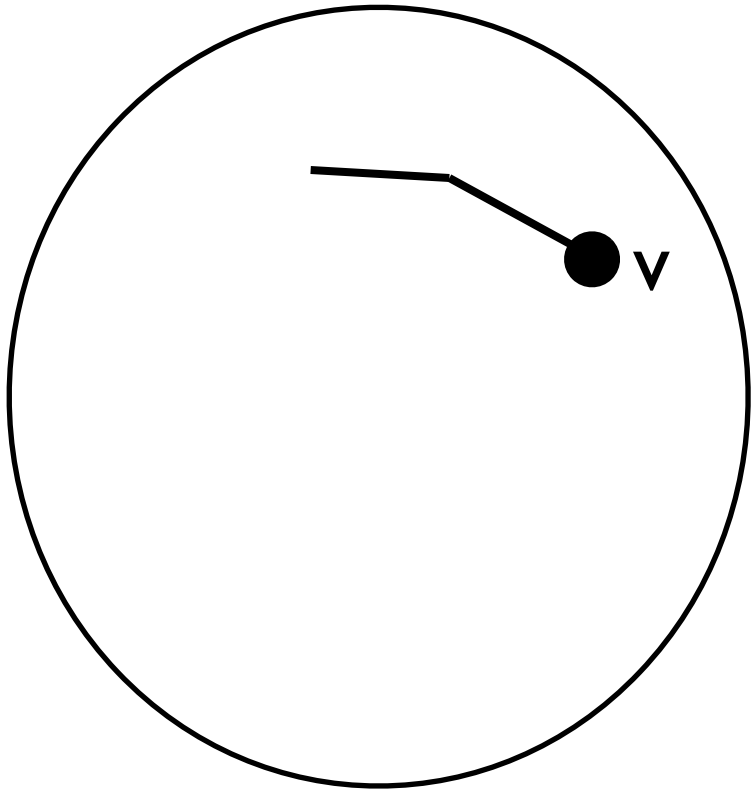
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



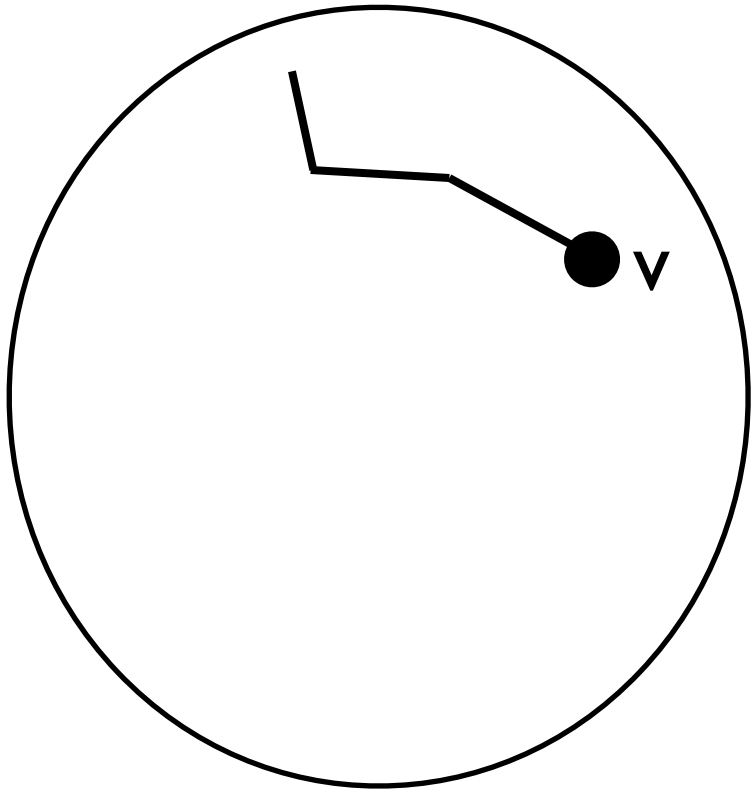
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



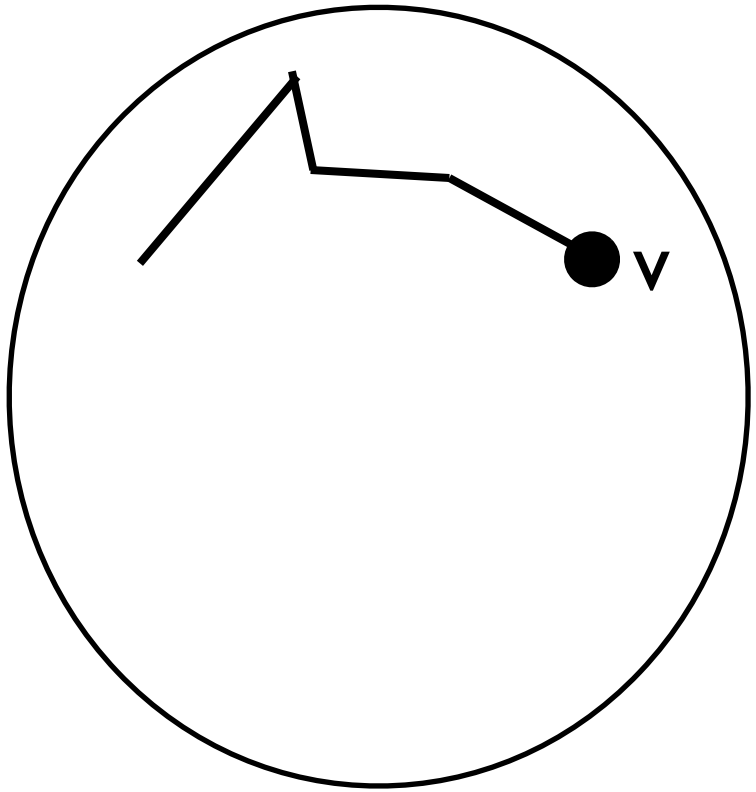
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



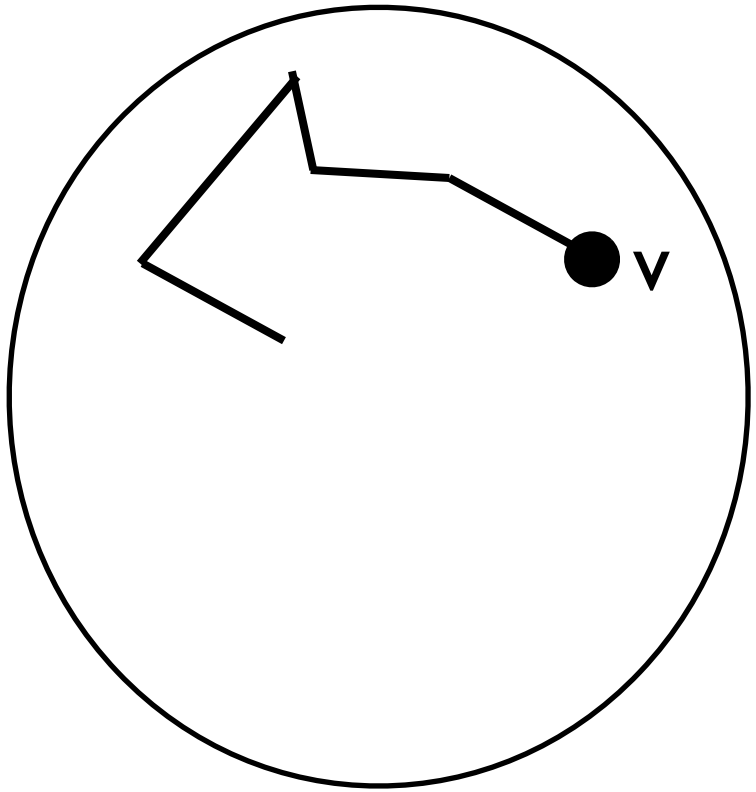
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



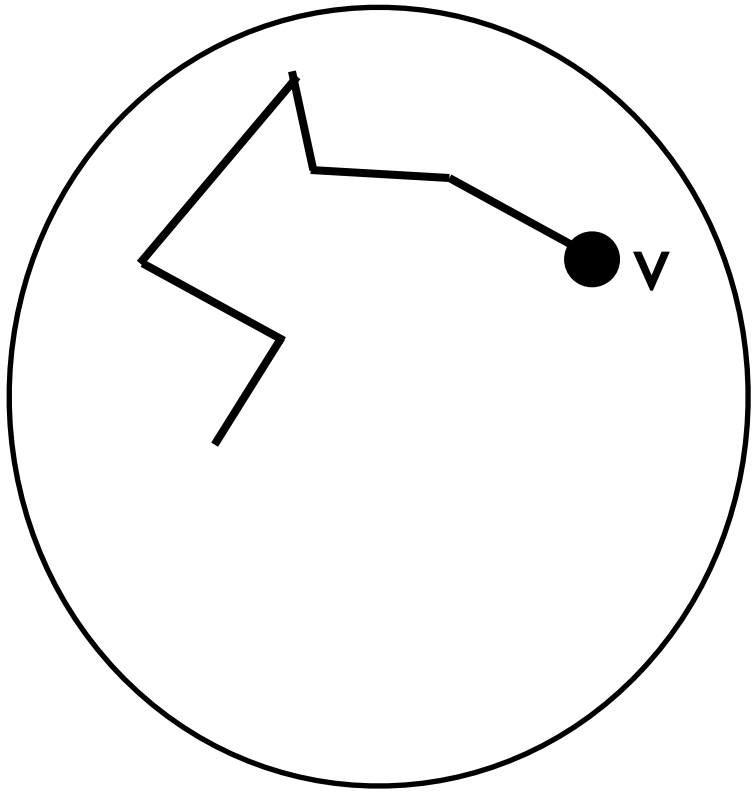
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



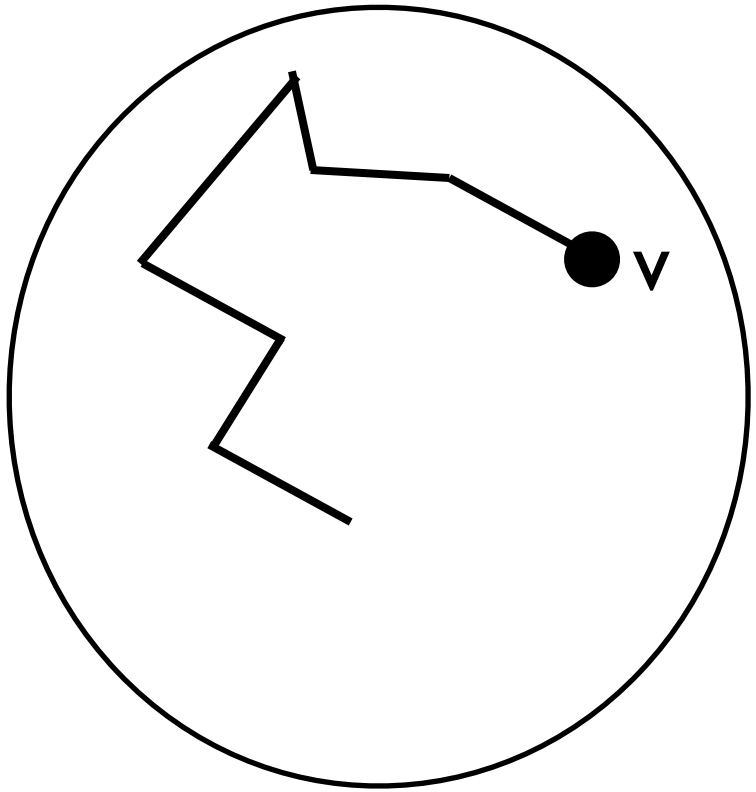
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



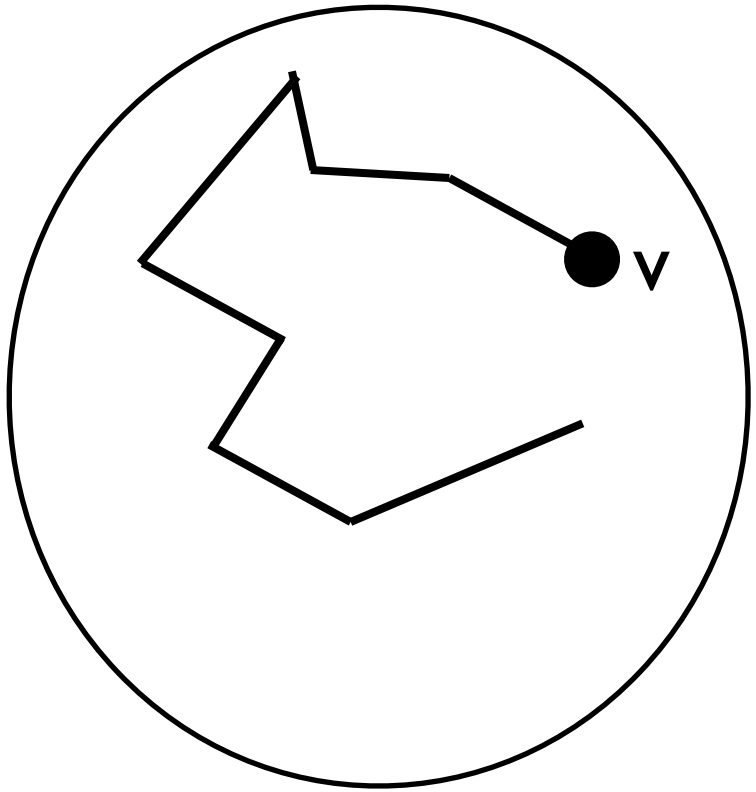
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



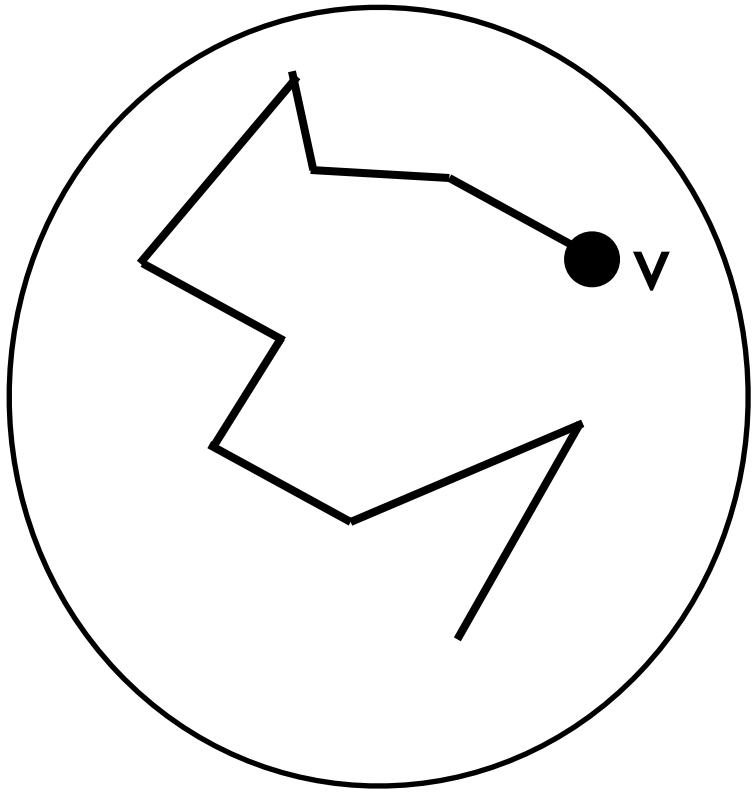
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



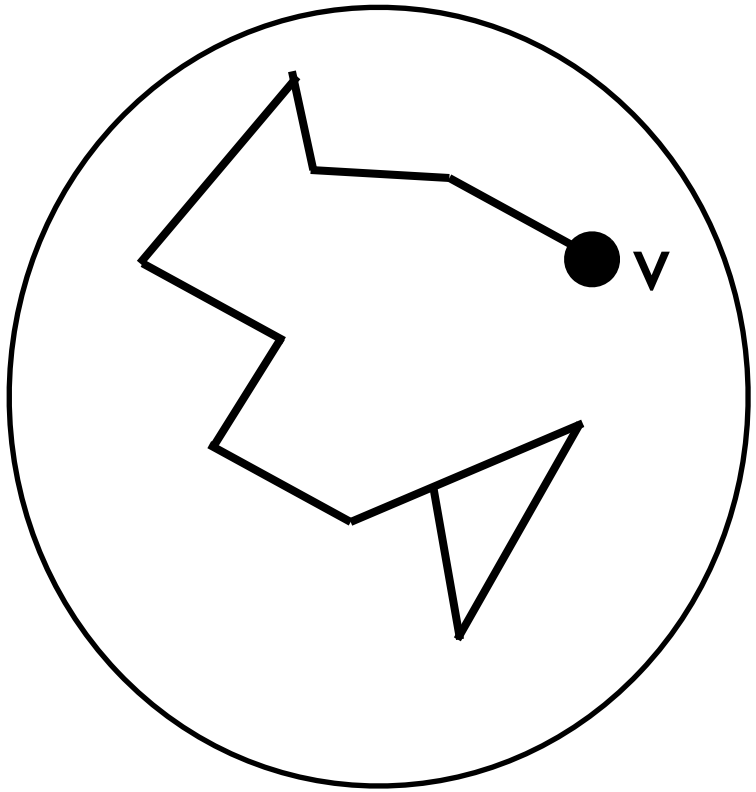
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



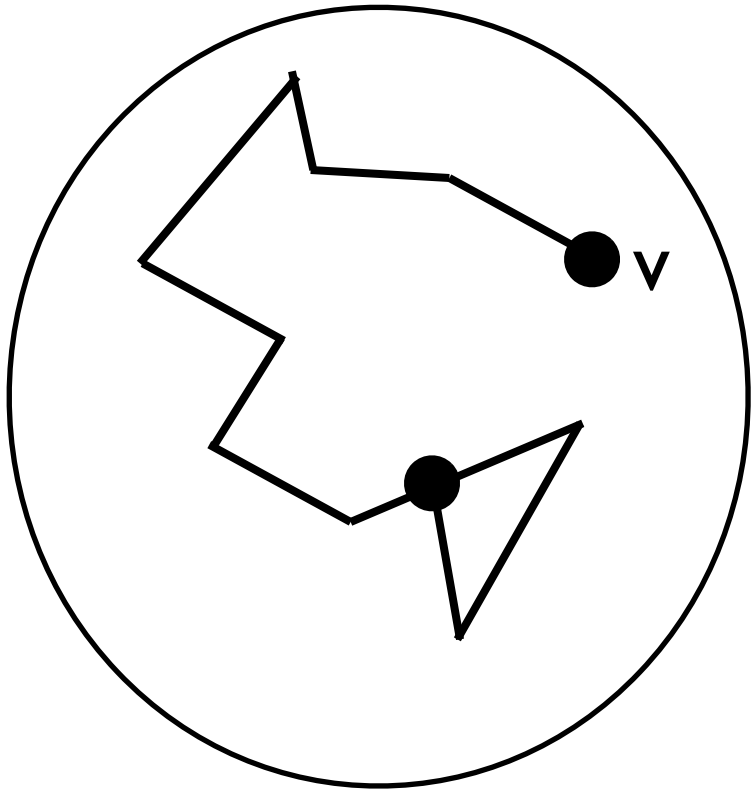
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



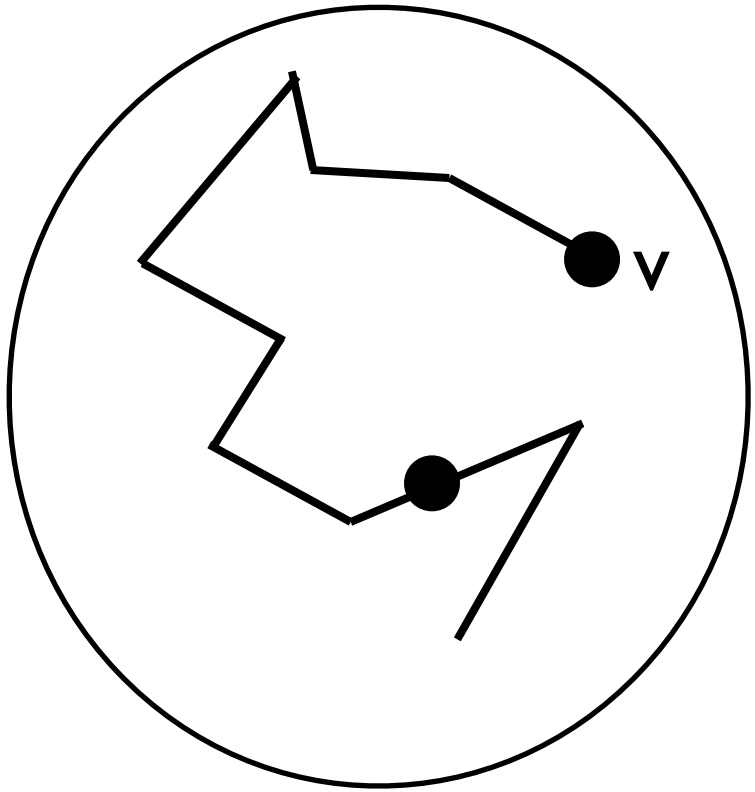
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



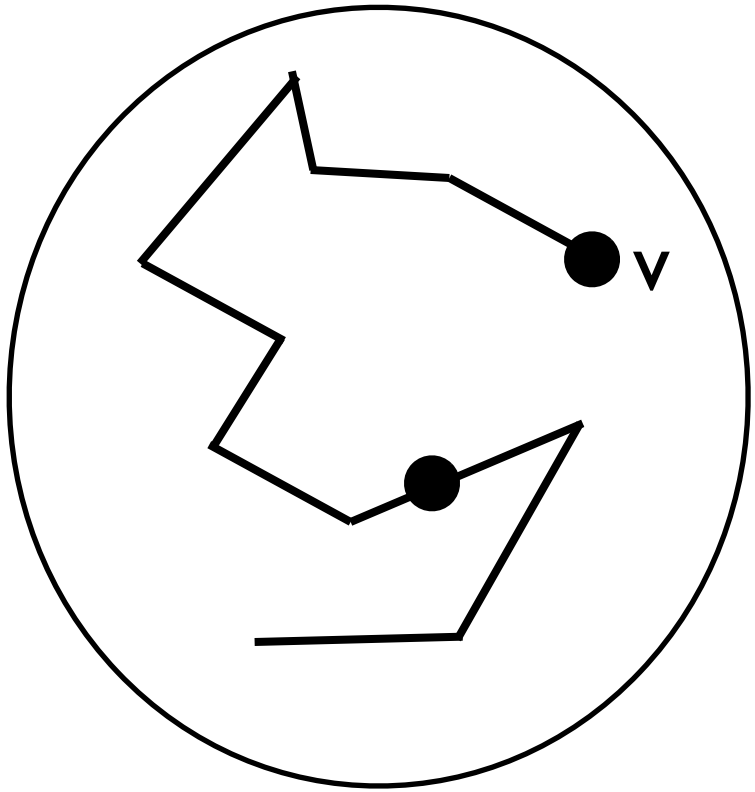
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



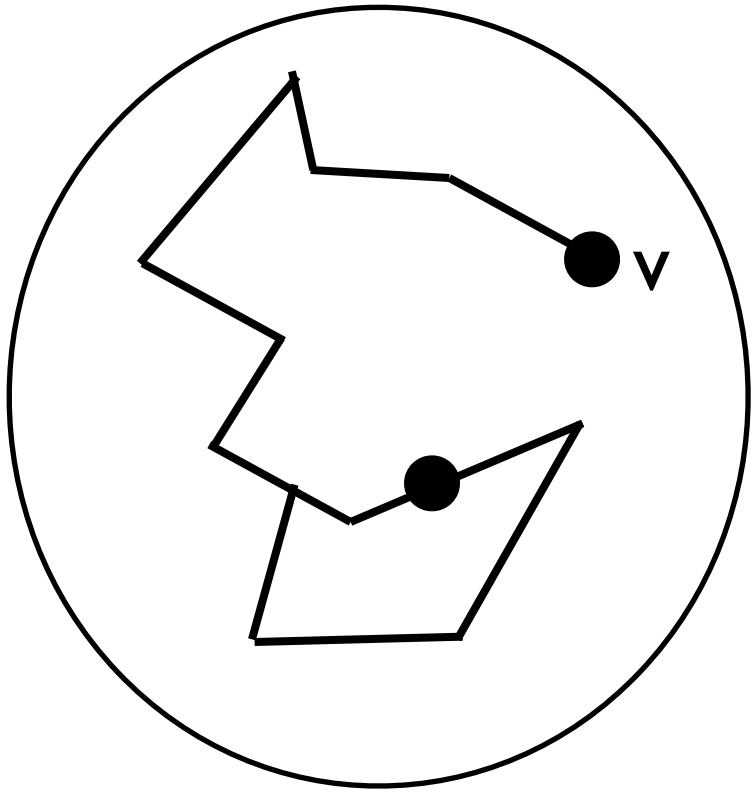
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



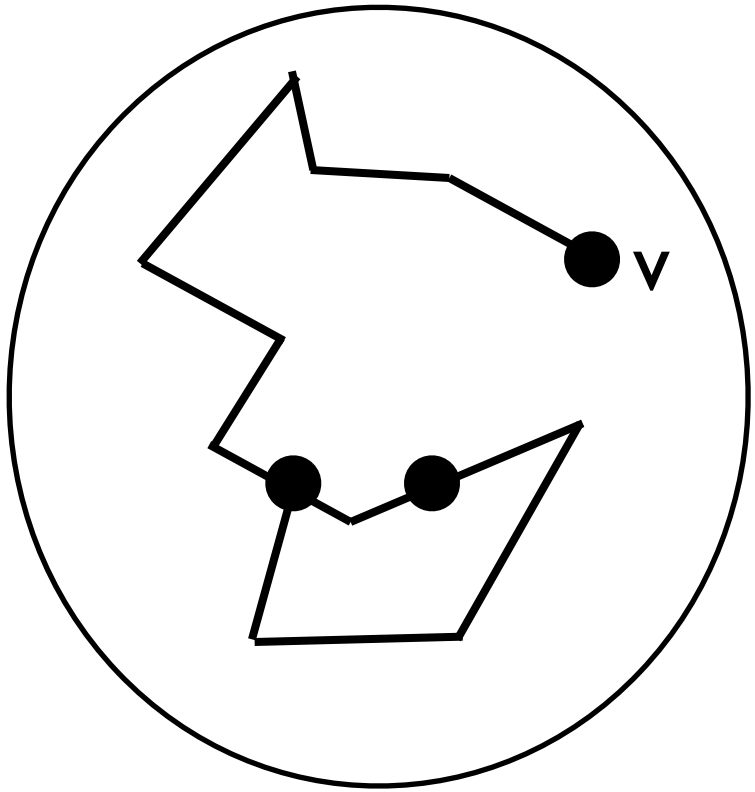
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



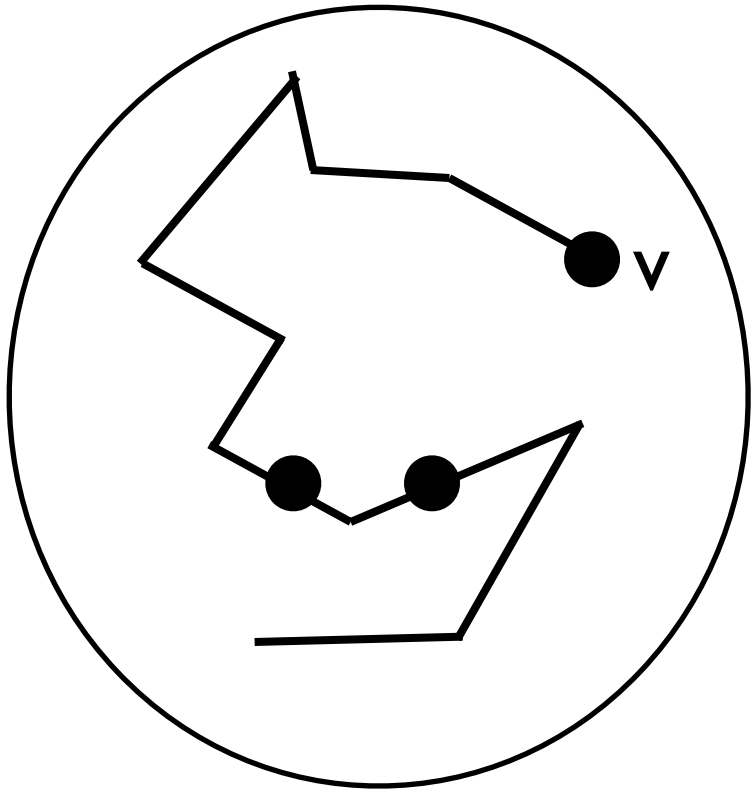
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



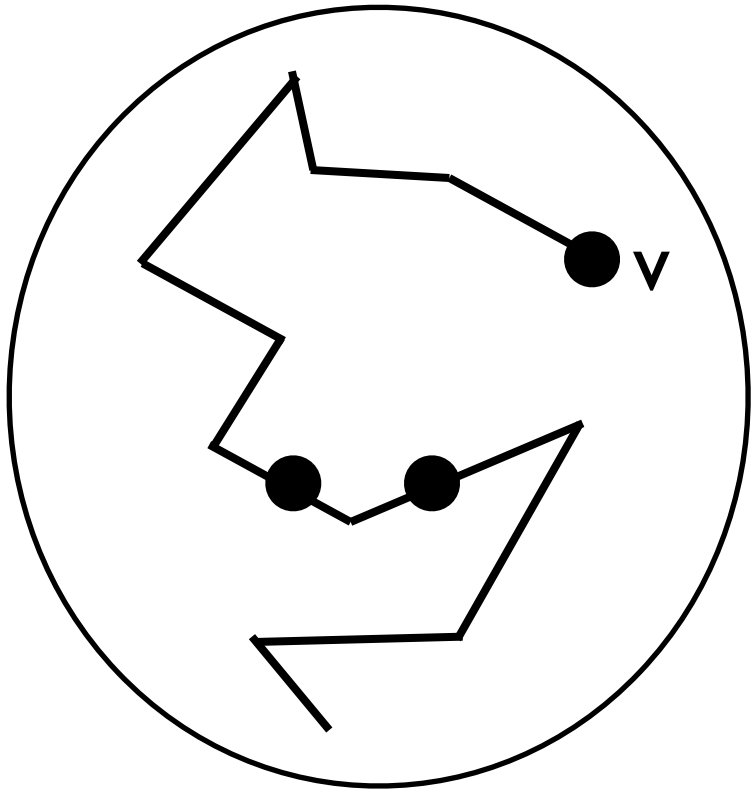
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



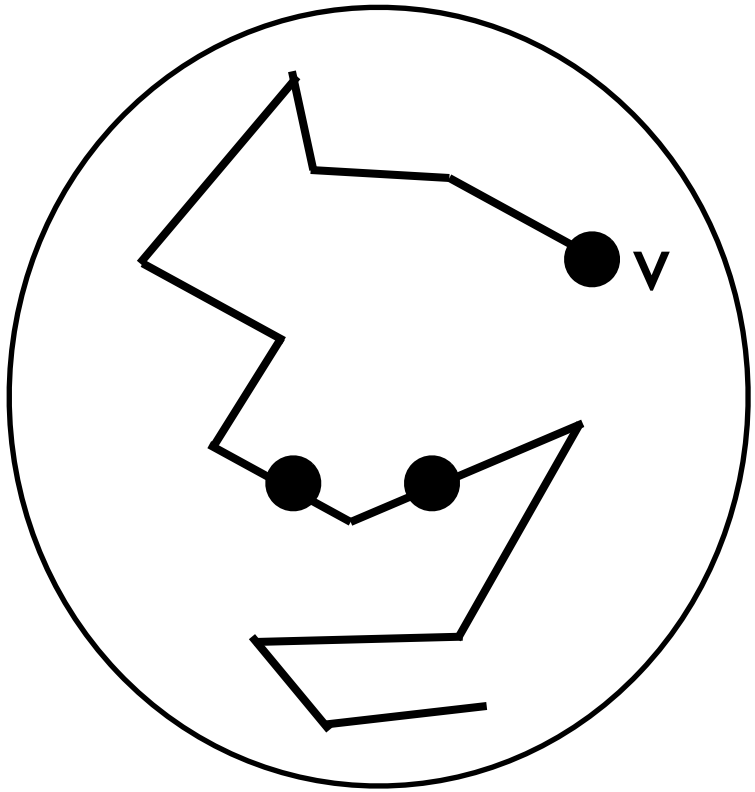
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Tiefendurchlauf



Implementierung von Datent.
Graphen - Durchlaufen

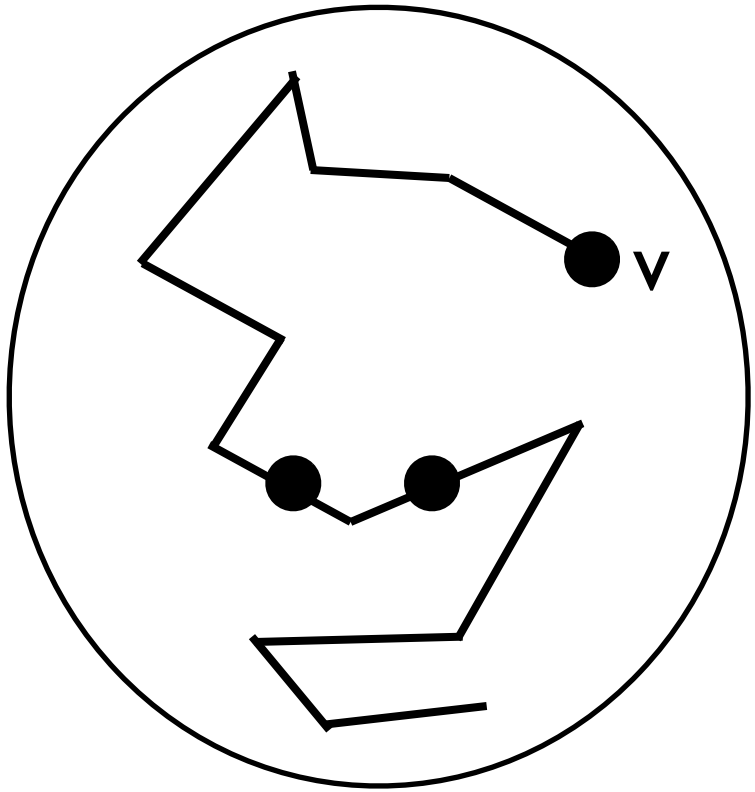
- Idee: Tiefendurchlauf



Implementierung von Datent.
Graphen - Durchlaufen

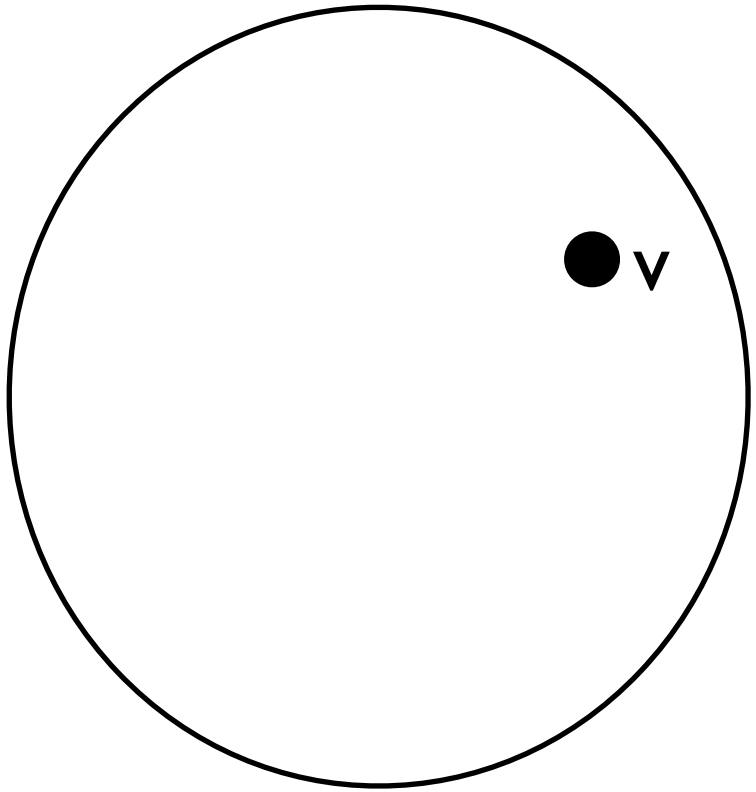
- Idee: Tiefendurchlauf

Markieren,
wo man schon
war



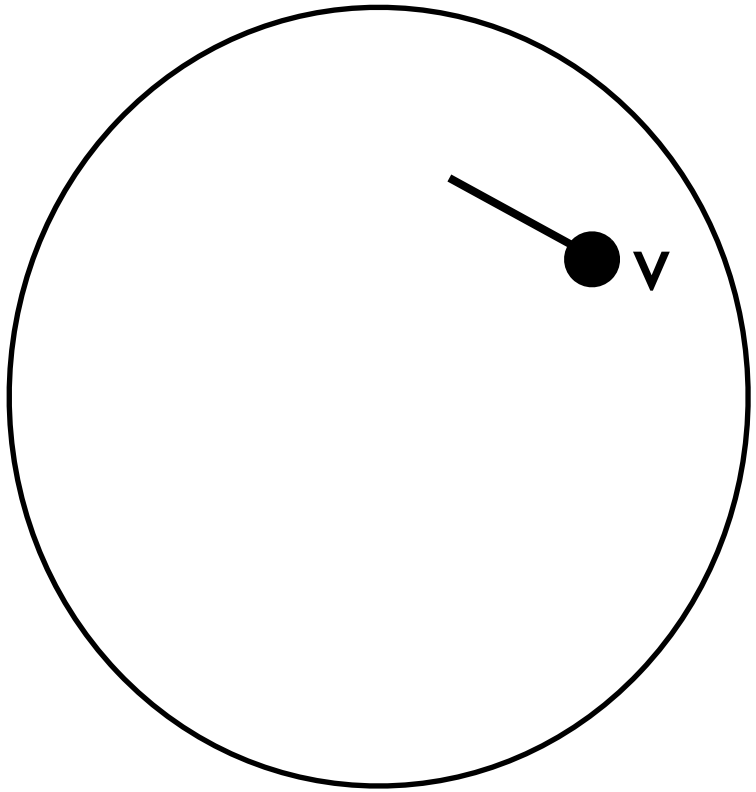
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



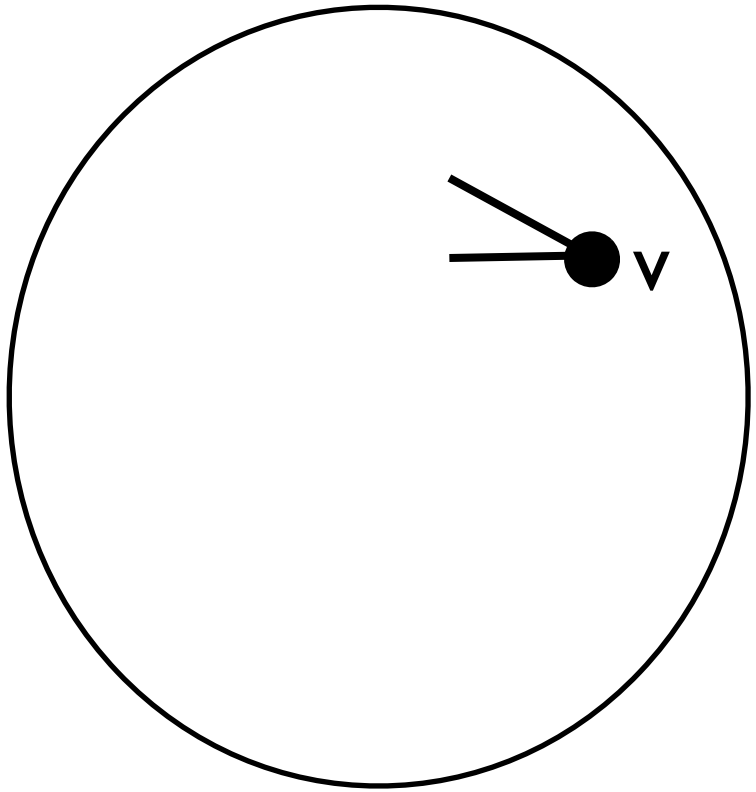
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



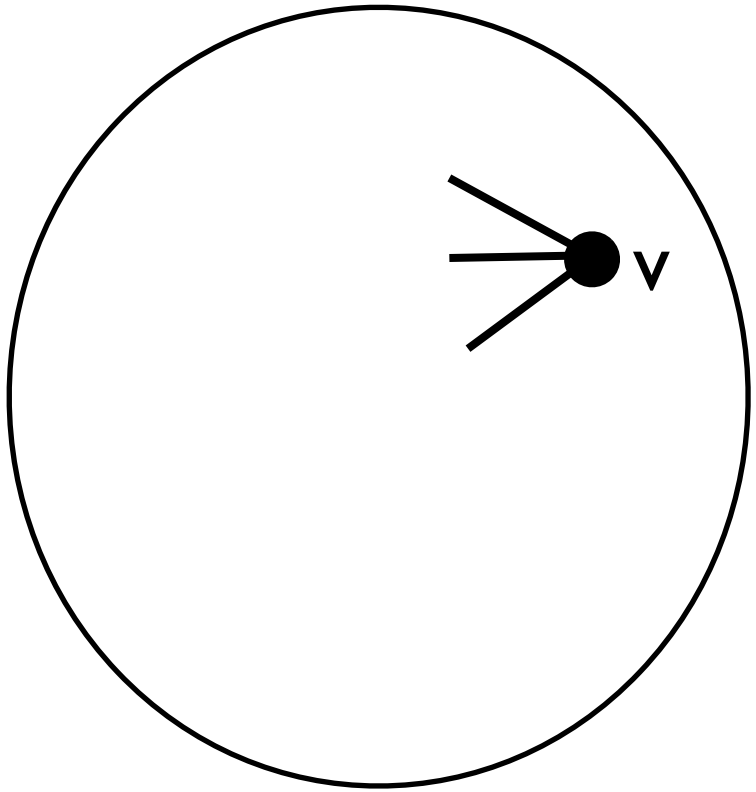
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



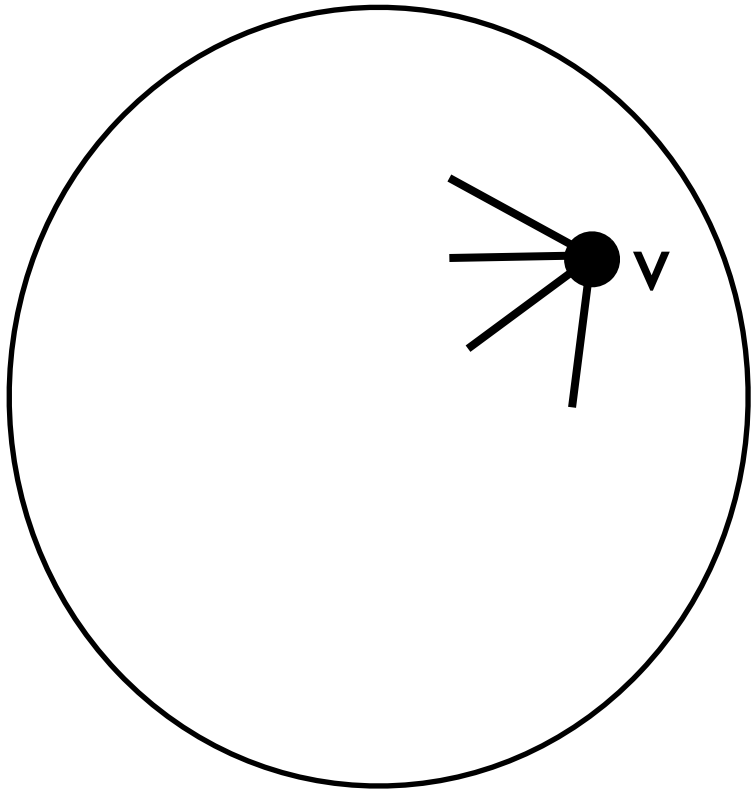
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



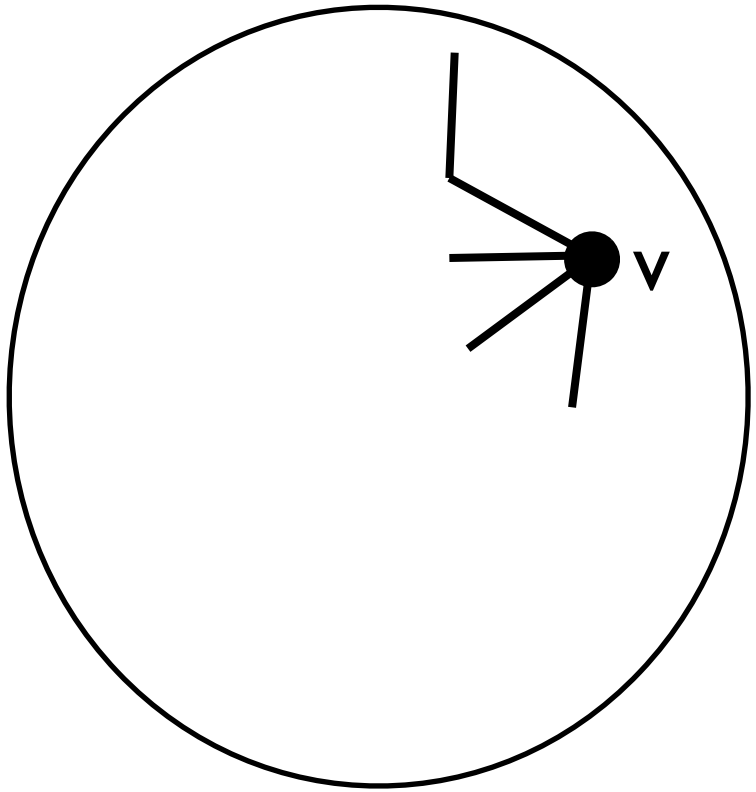
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



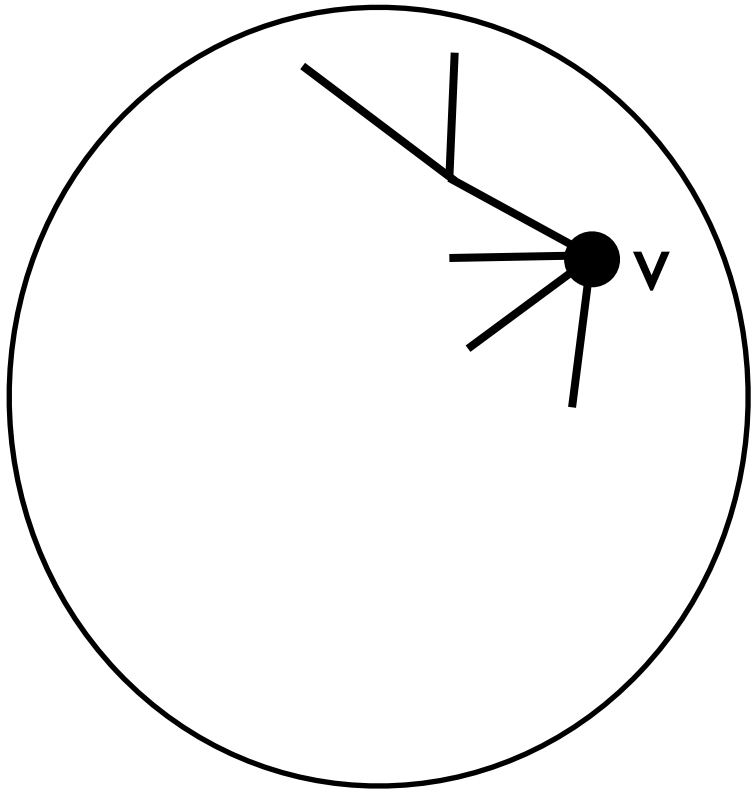
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



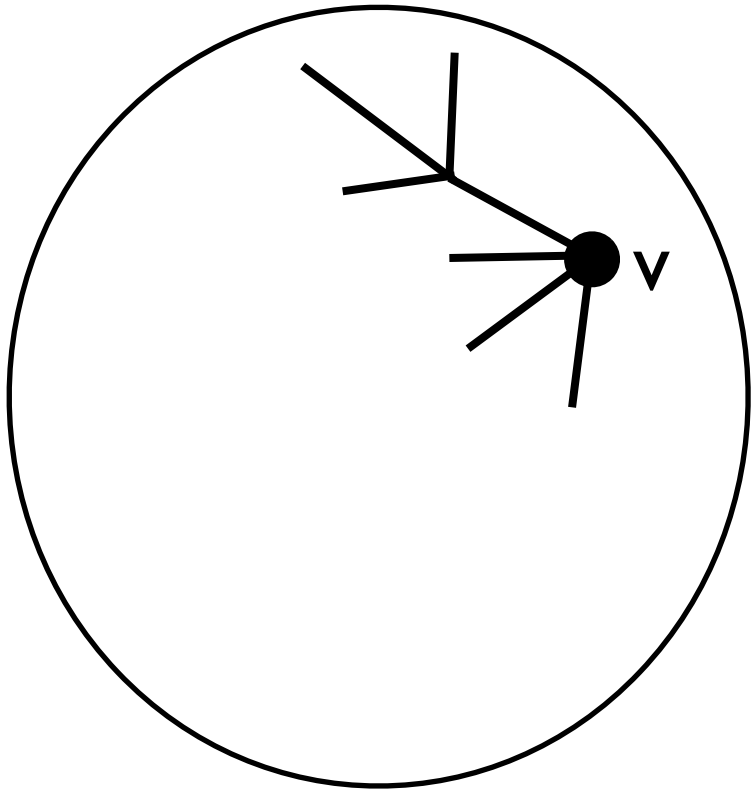
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



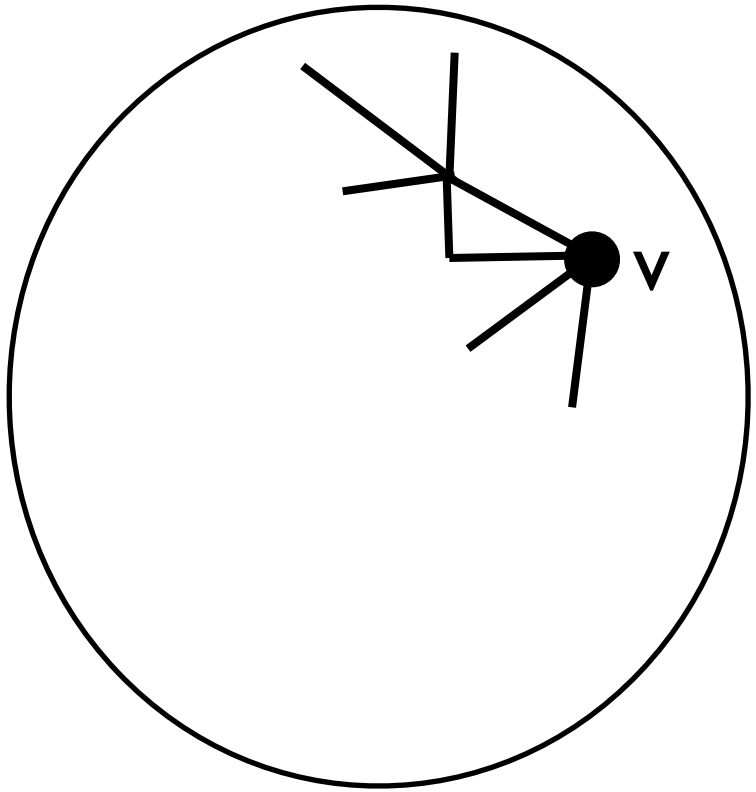
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



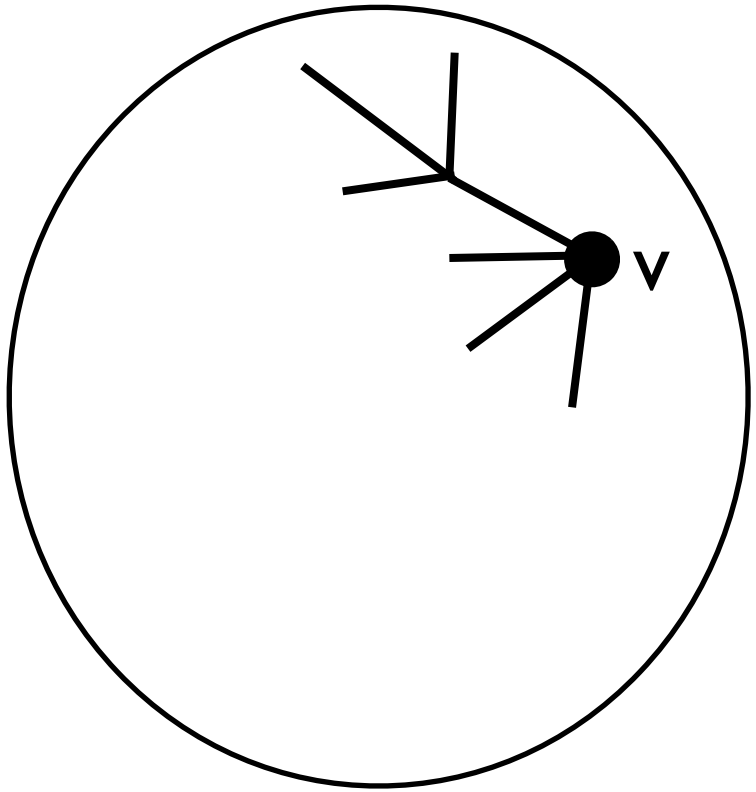
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



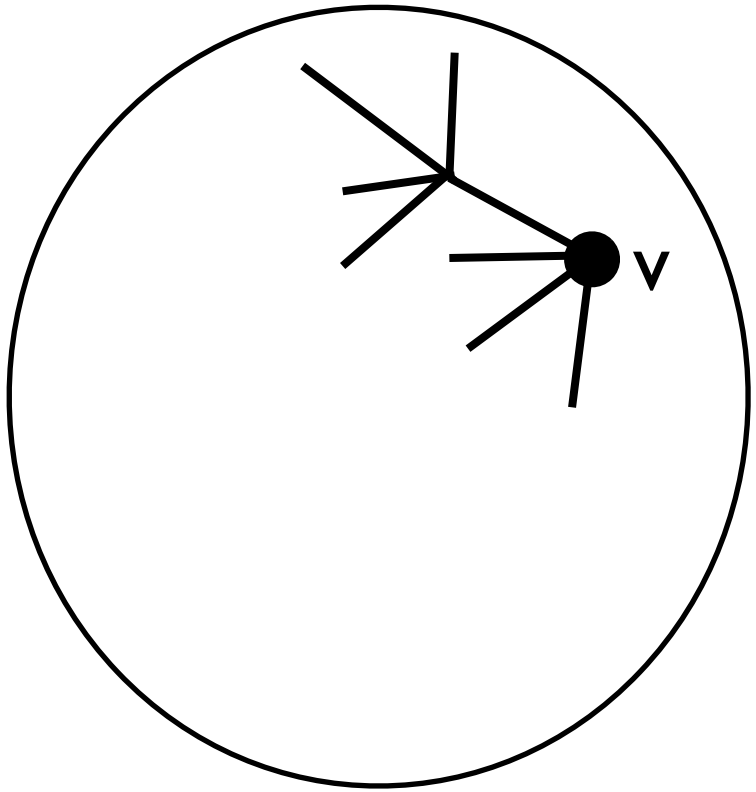
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



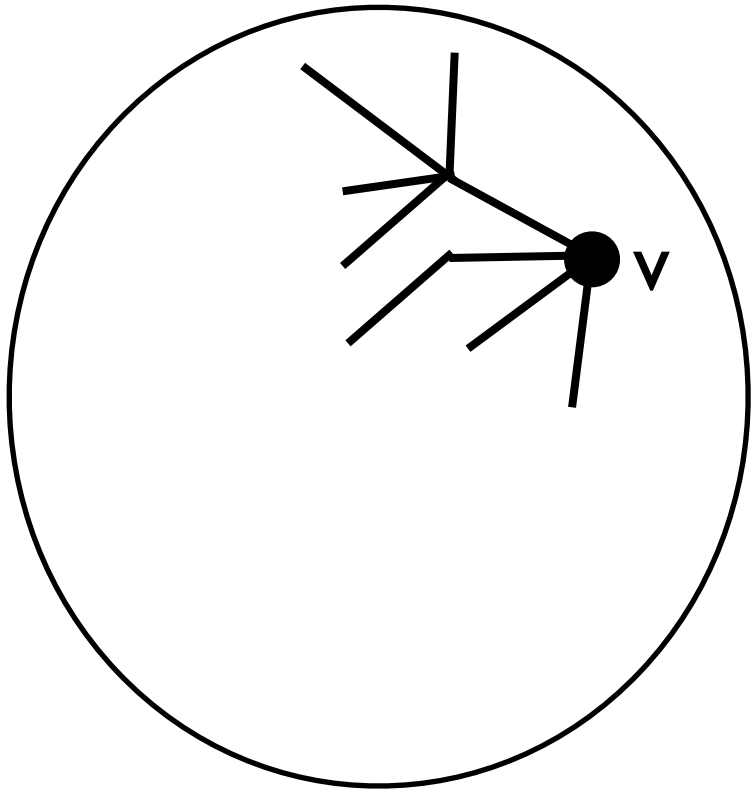
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



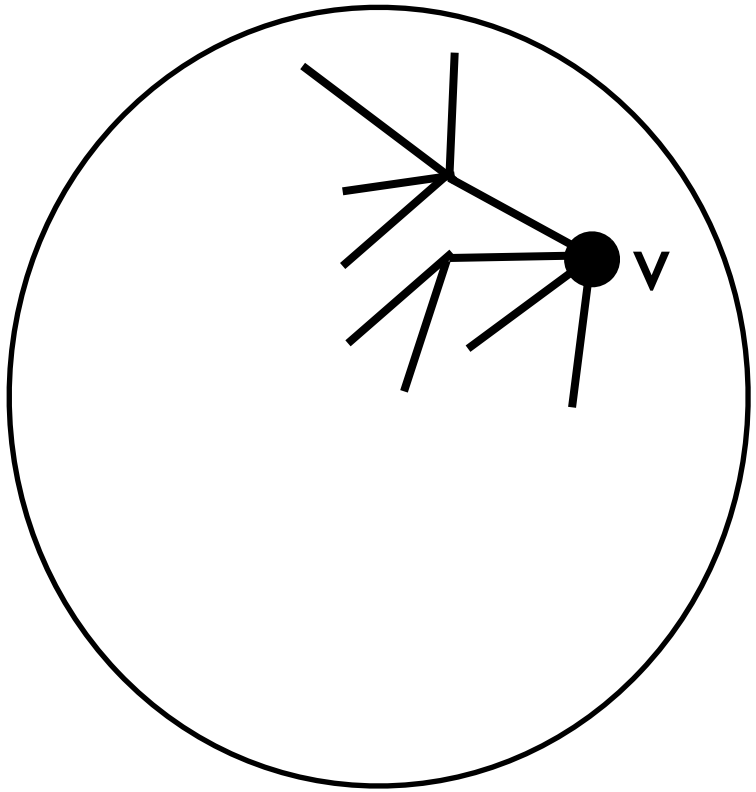
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



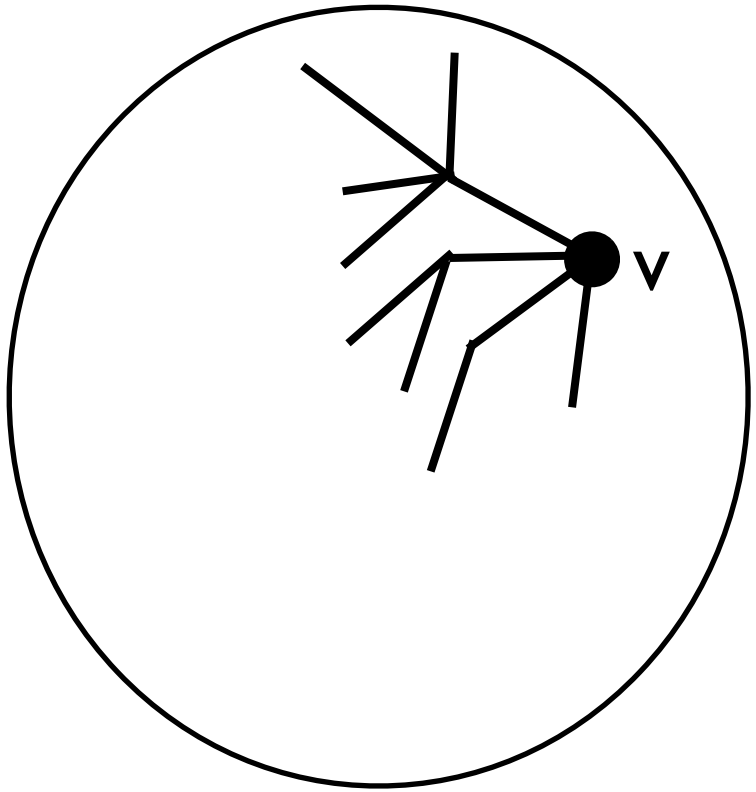
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



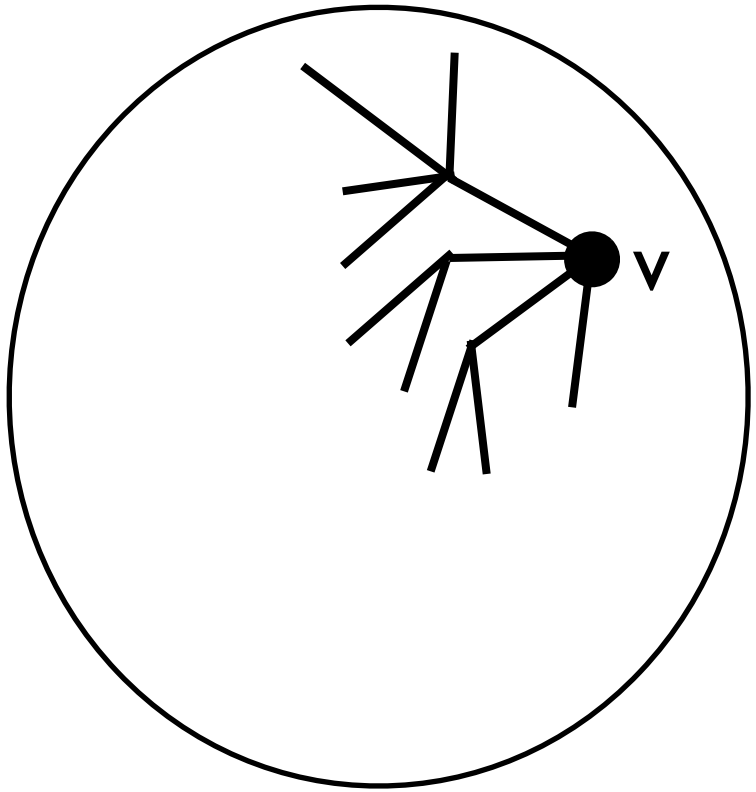
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



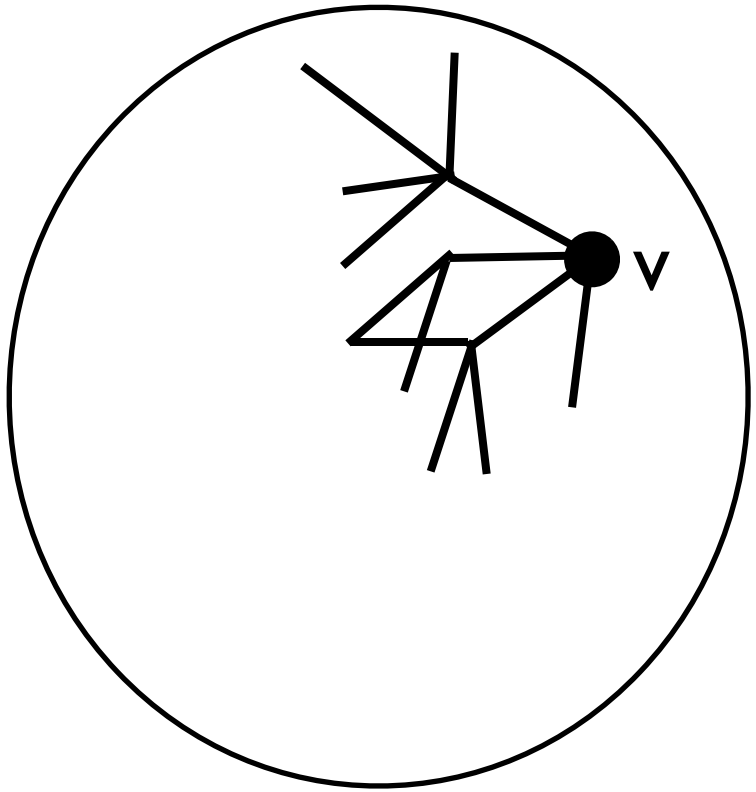
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



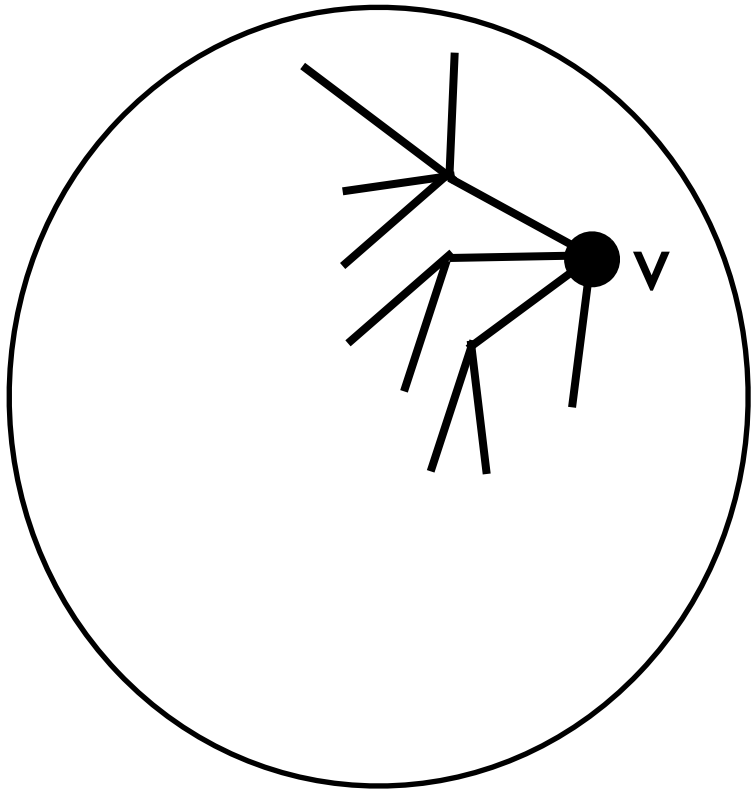
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



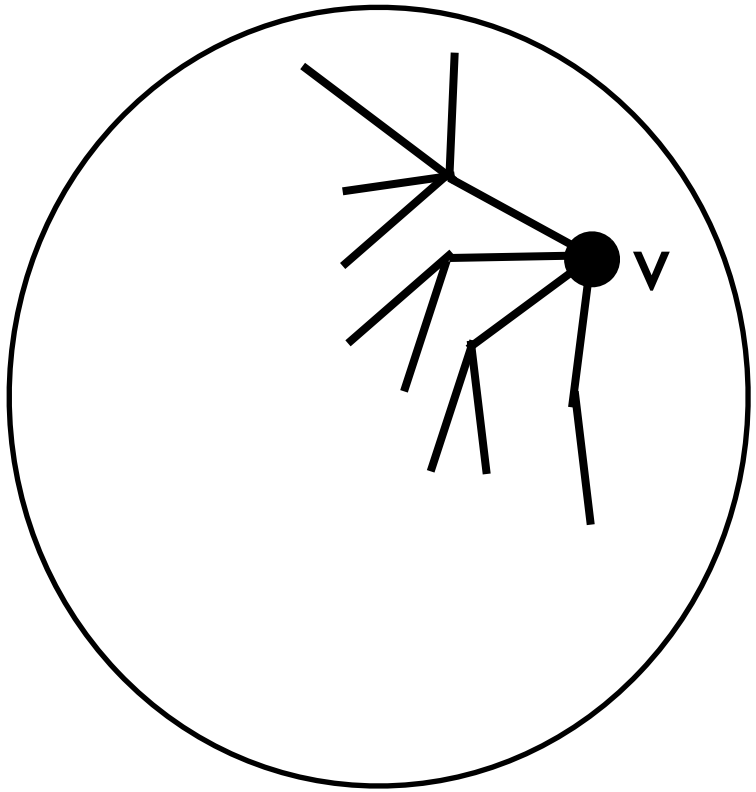
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



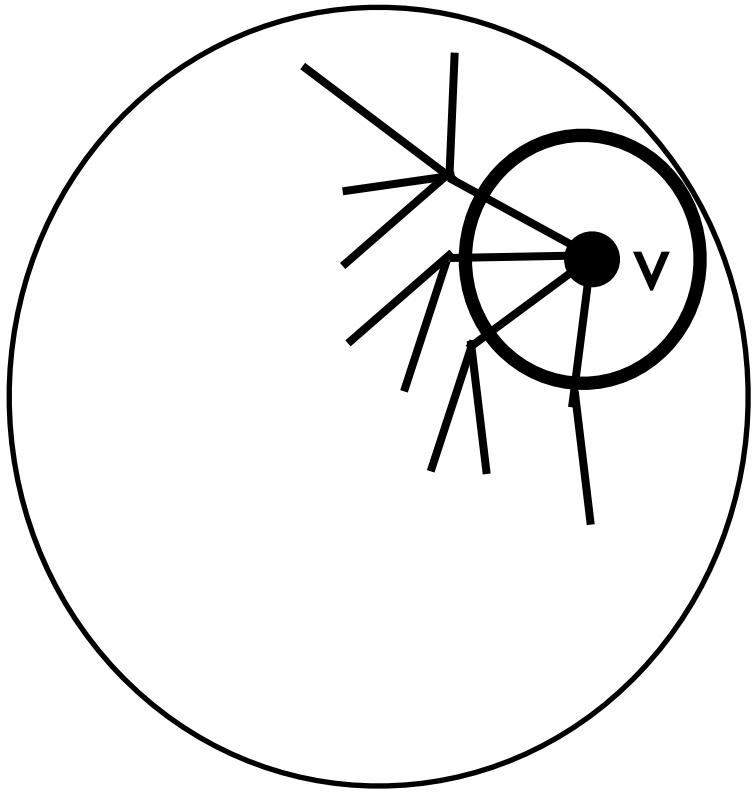
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



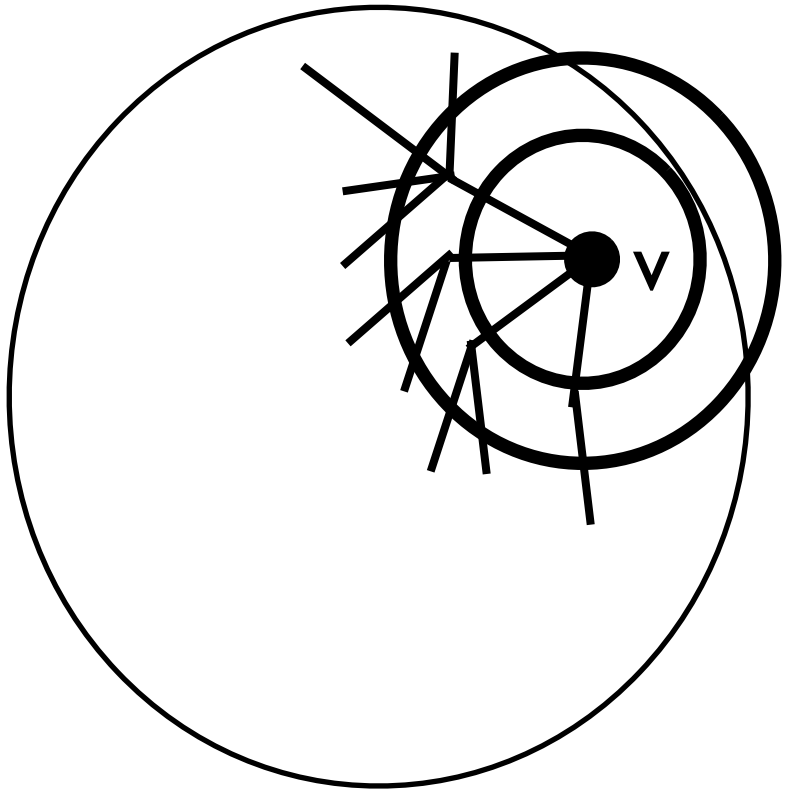
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



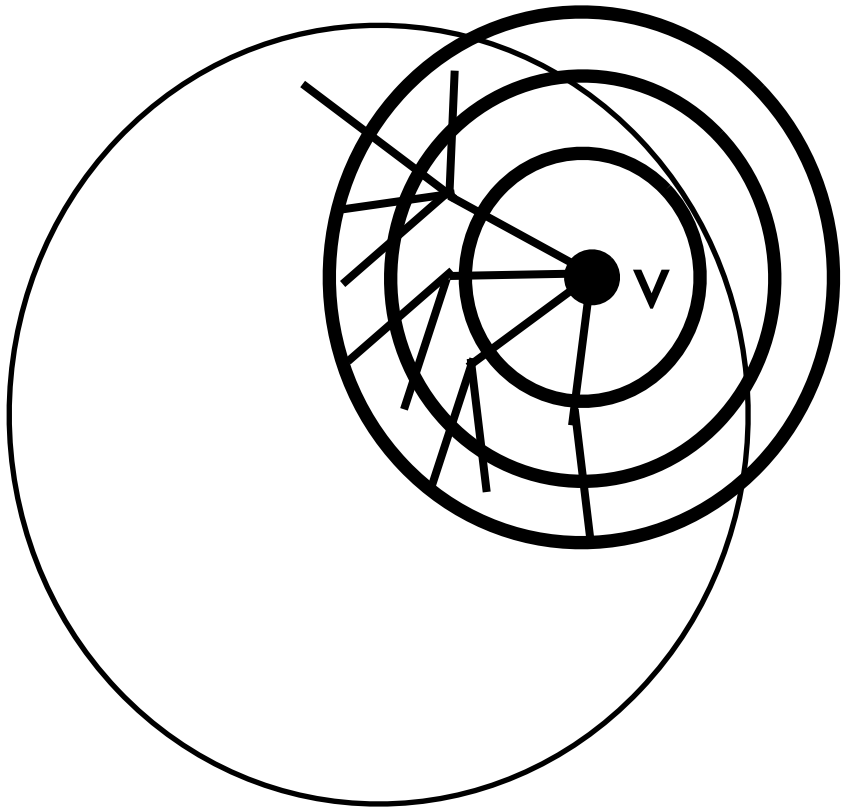
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



Implementierung von Datent.
Graphen - Durchlaufen

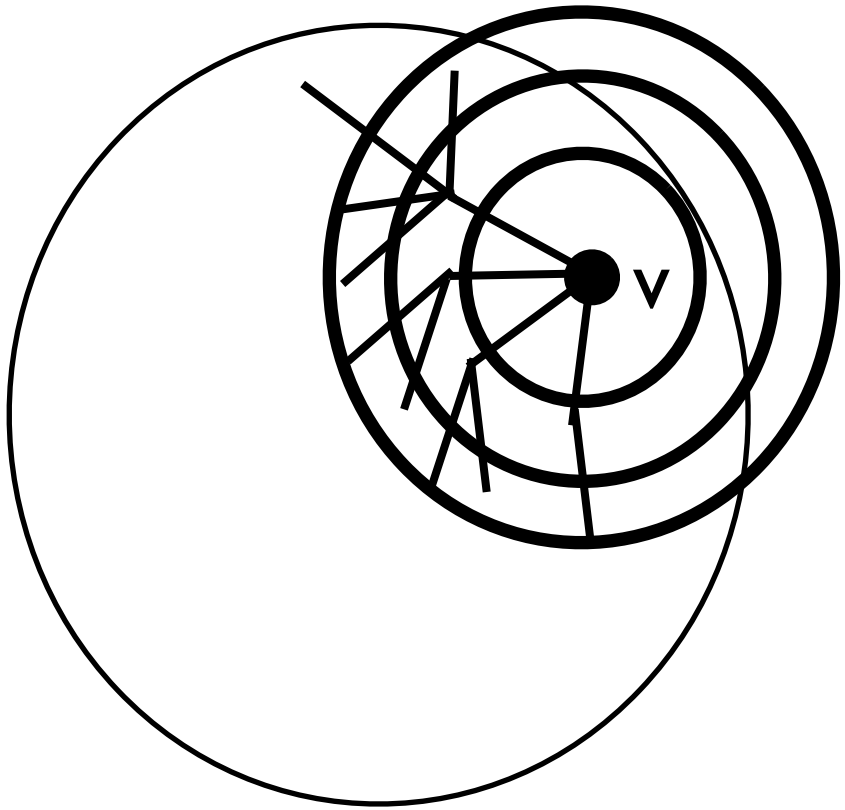
- Idee: Breitendurchlauf



Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf

Markieren,
wo man schon
war



Implementierung von Datent.
Graphen - Implementierung dfs

- dfs:

```
var      besucht: array [V] of boolean.
procedure dfs(v: knoten);
begin
     besucht[v]:=true; write(v);
     for all w∈V: {v,w}∈E do
     if not besucht[w] then dfs(w)
end.
```

- Laufzeit proportional zur Größe des Graphen (=|V|+|E|)

Implementierung von Datent.
Graphen - Implementierung bfs

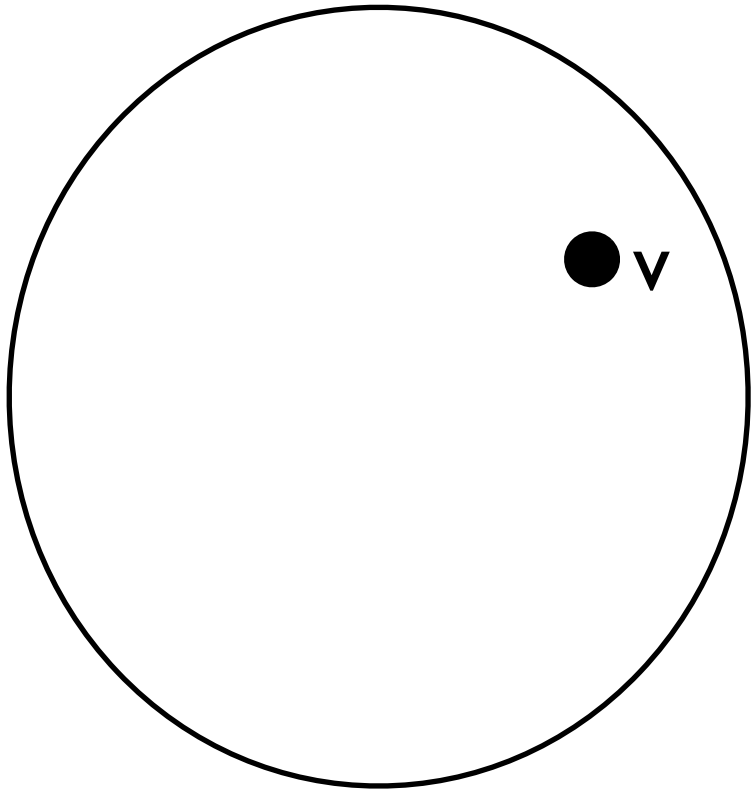
- bfs:

```
var besucht: array [V] of boolean;
            q: queue of V;
procedure bfs(v: knoten);
begin
    besucht[v]:=true; enter(v,q);
    while not is_empty(q) do
    begin
        v:=first(q); write(v); remove(q);
        for all w∈V: {v,w}∈E do
            if not besucht[w] then
            begin
                besucht[w]:=true;
                enter(w,q)
            end
    end
end.
```

Laufzeit proportional zur Größe des Graphen (=|V|+|E|)

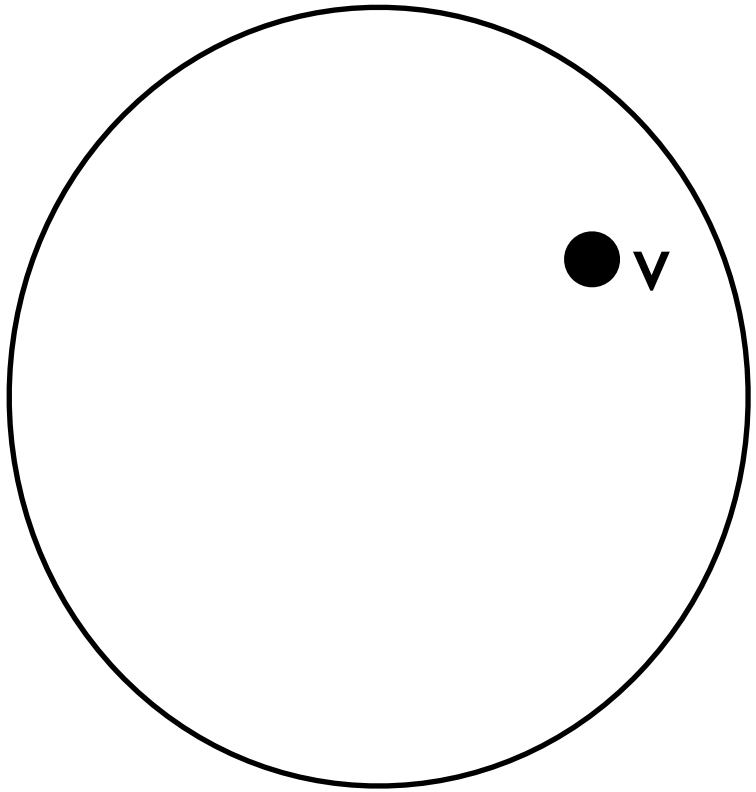
Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



Implementierung von Datent.
Graphen - Durchlaufen

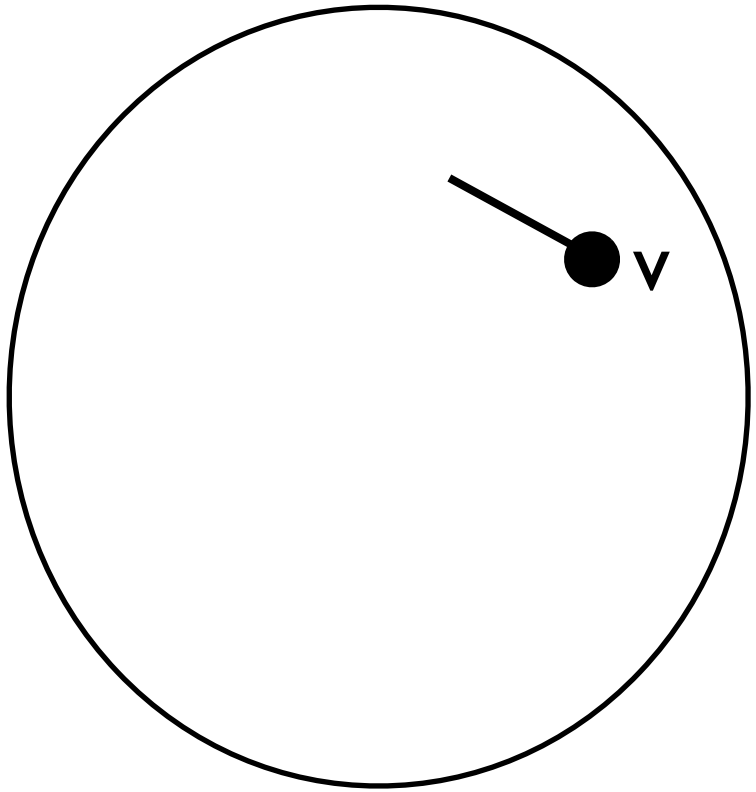
- Idee: Breitendurchlauf



Queue:

Implementierung von Datent.
Graphen - Durchlaufen

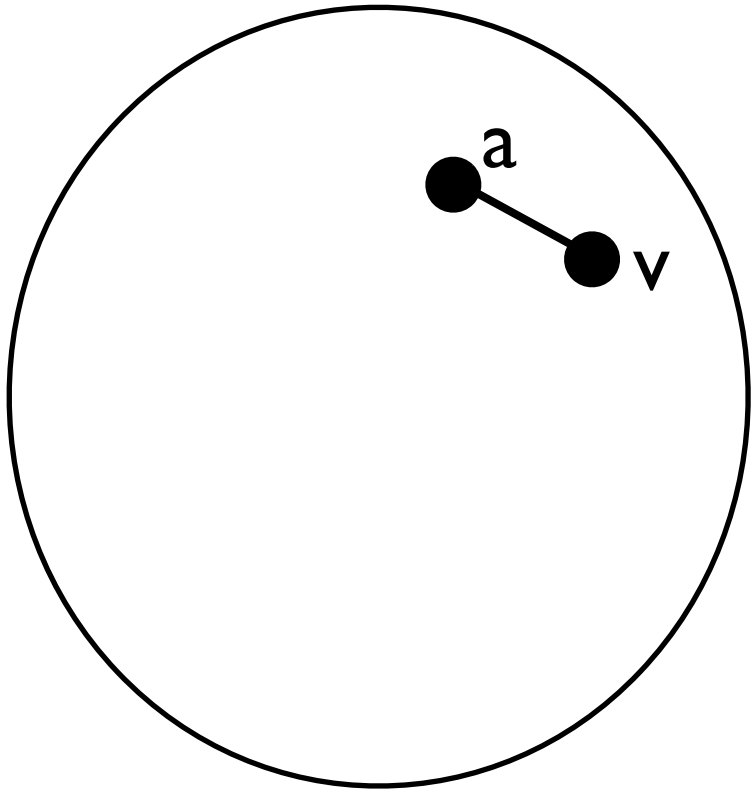
- Idee: Breitendurchlauf



Queue:

Implementierung von Datent.
Graphen - Durchlaufen

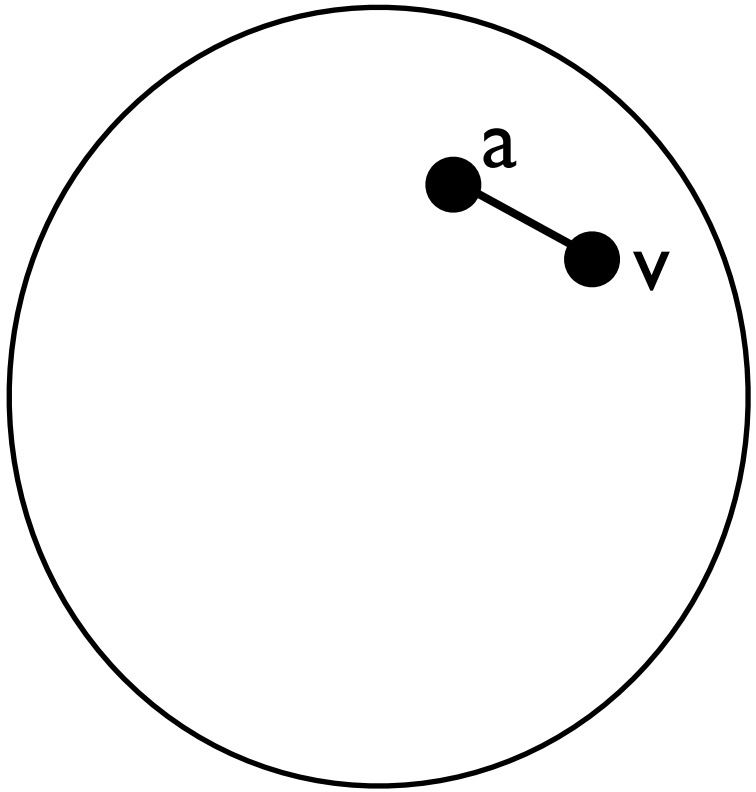
- Idee: Breitendurchlauf



Queue:

Implementierung von Datent.
Graphen - Durchlaufen

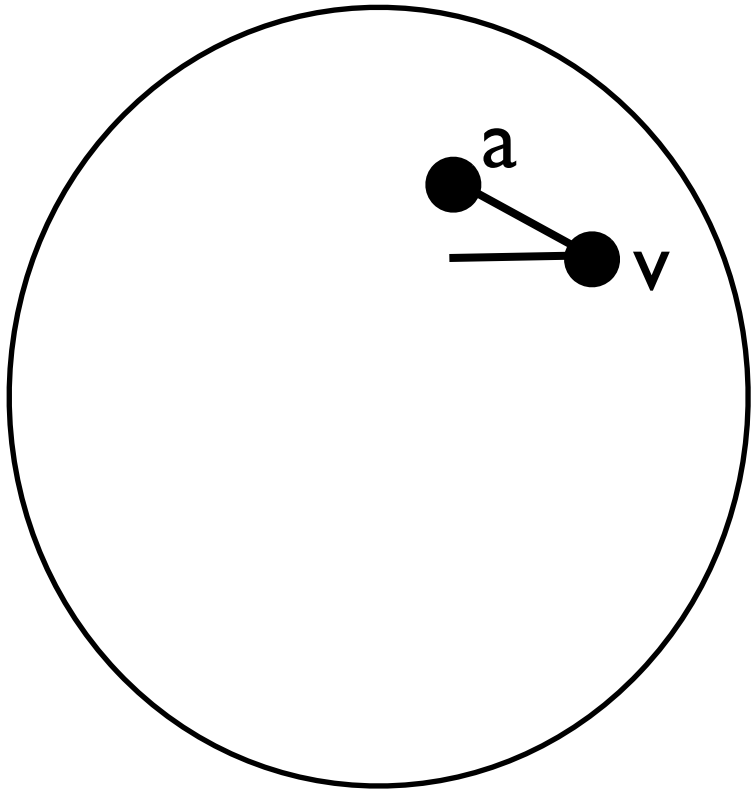
- Idee: Breitendurchlauf



Queue: a

Implementierung von Datent.
Graphen - Durchlaufen

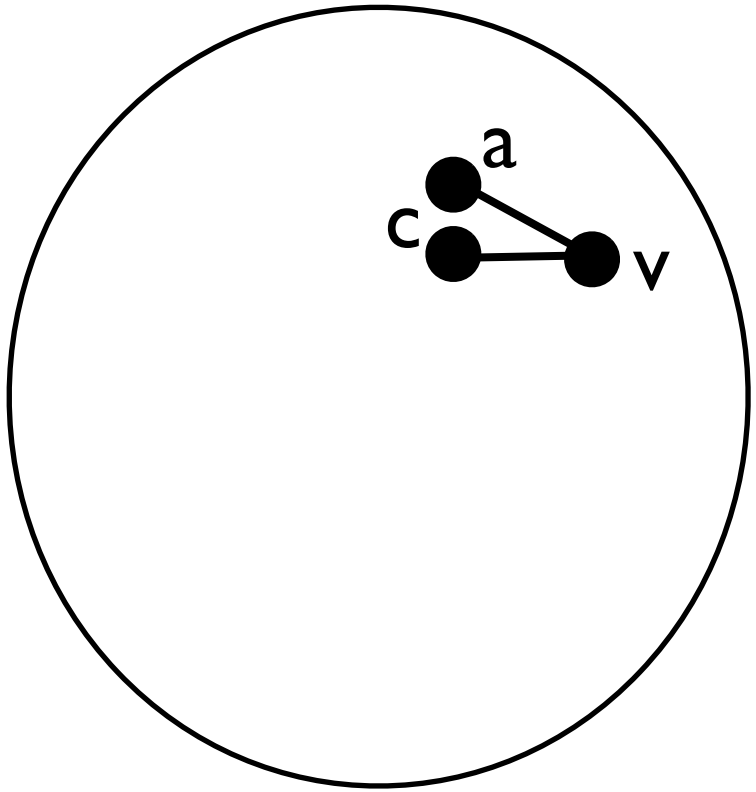
- Idee: Breitendurchlauf



Queue: a

Implementierung von Datent.
Graphen - Durchlaufen

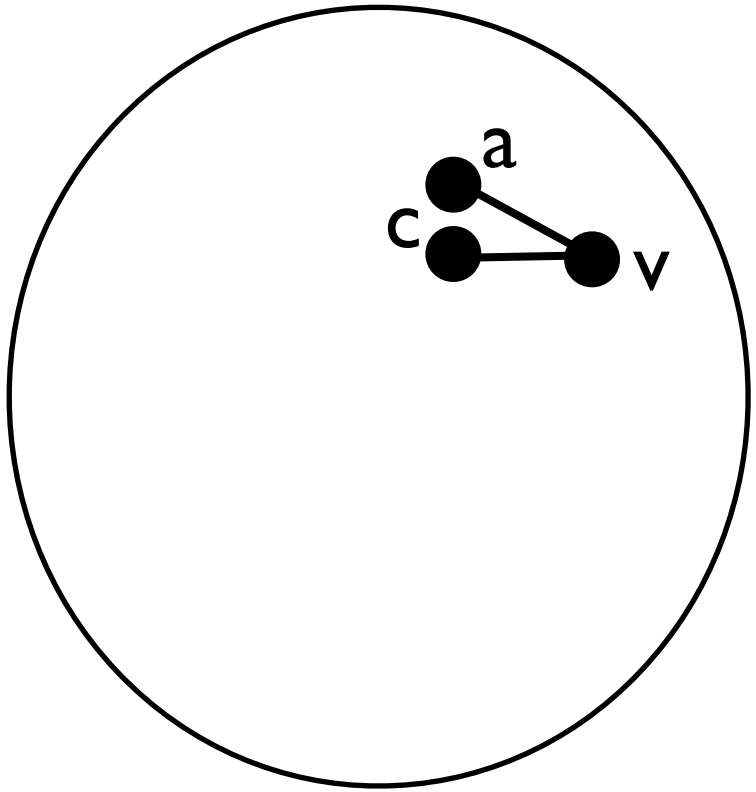
- Idee: Breitendurchlauf



Queue: a

Implementierung von Datent.
Graphen - Durchlaufen

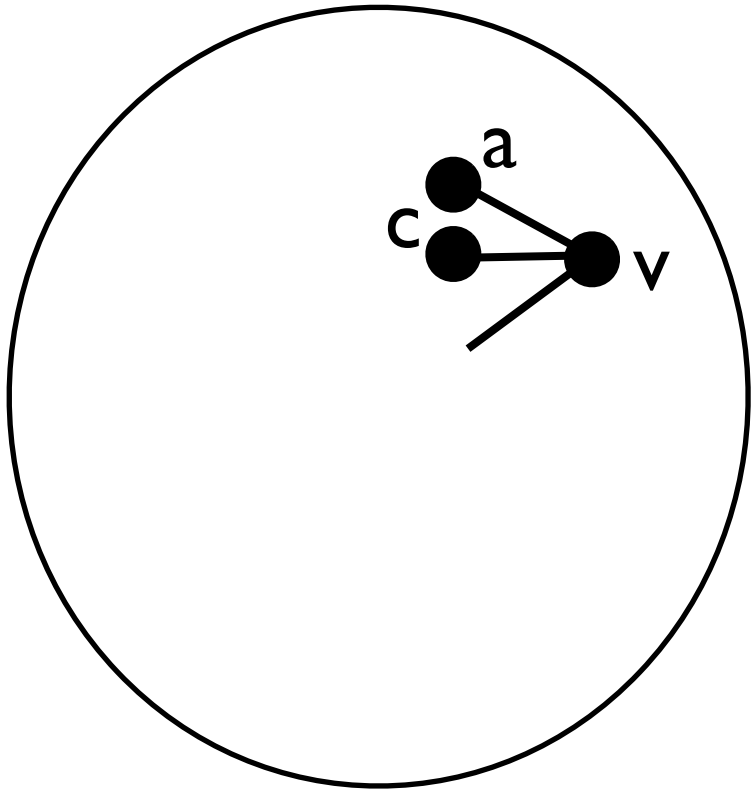
- Idee: Breitendurchlauf



Queue: a c

Implementierung von Datent.
Graphen - Durchlaufen

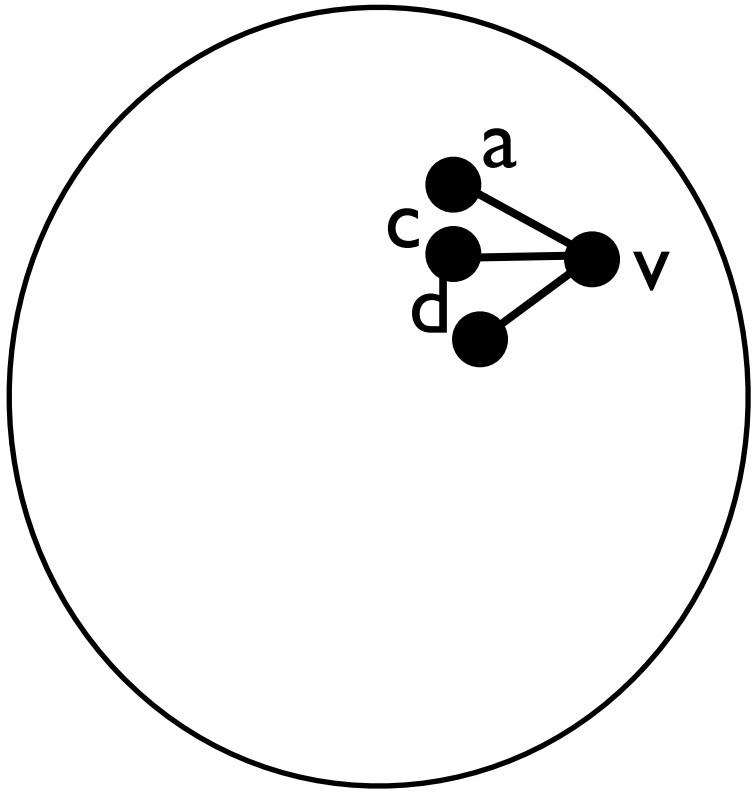
- Idee: Breitendurchlauf



Queue: a c

Implementierung von Datent.
Graphen - Durchlaufen

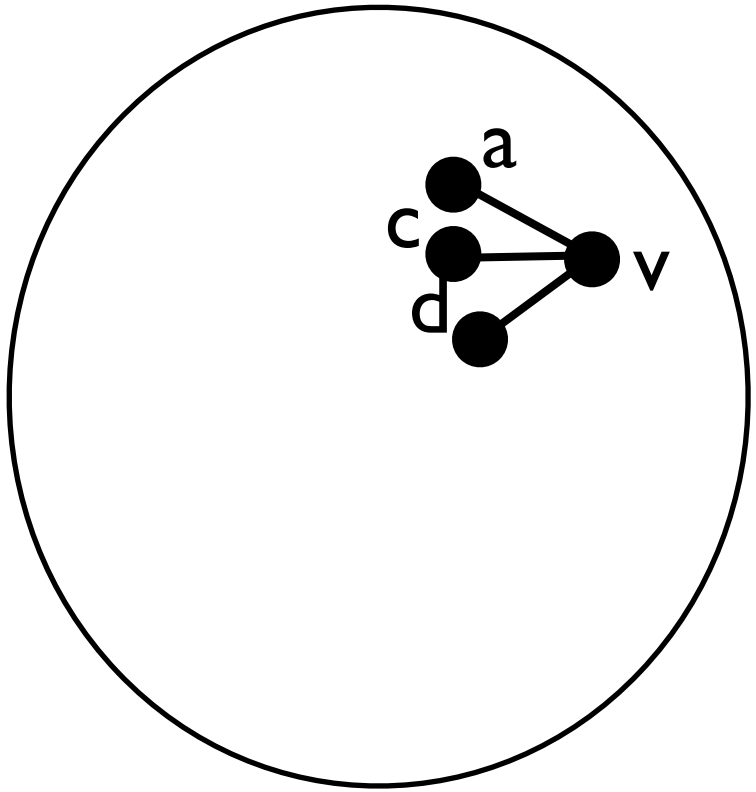
- Idee: Breitendurchlauf



Queue: a c

Implementierung von Datent.
Graphen - Durchlaufen

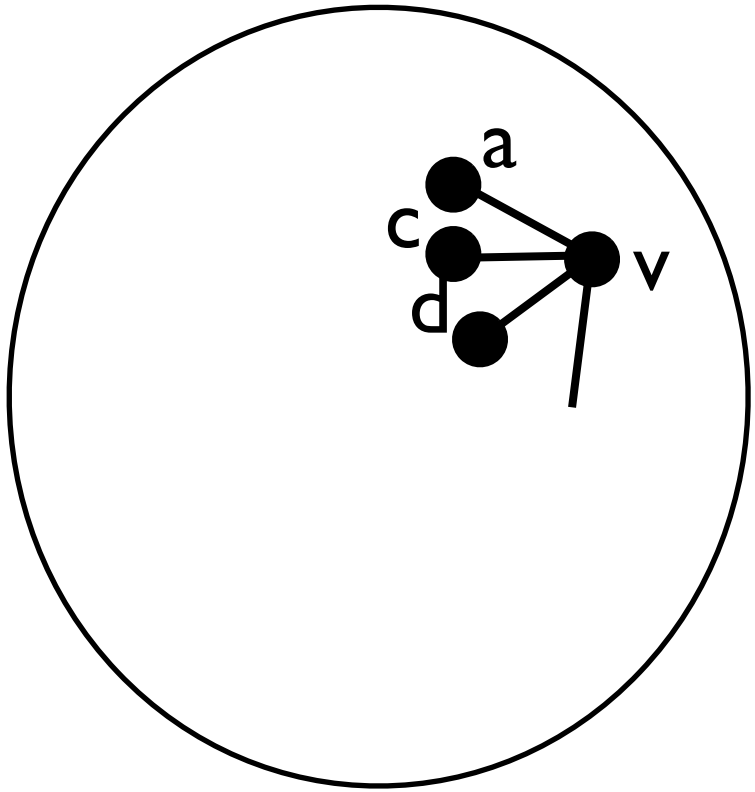
- Idee: Breitendurchlauf



Queue: a c d

Implementierung von Datent.
Graphen - Durchlaufen

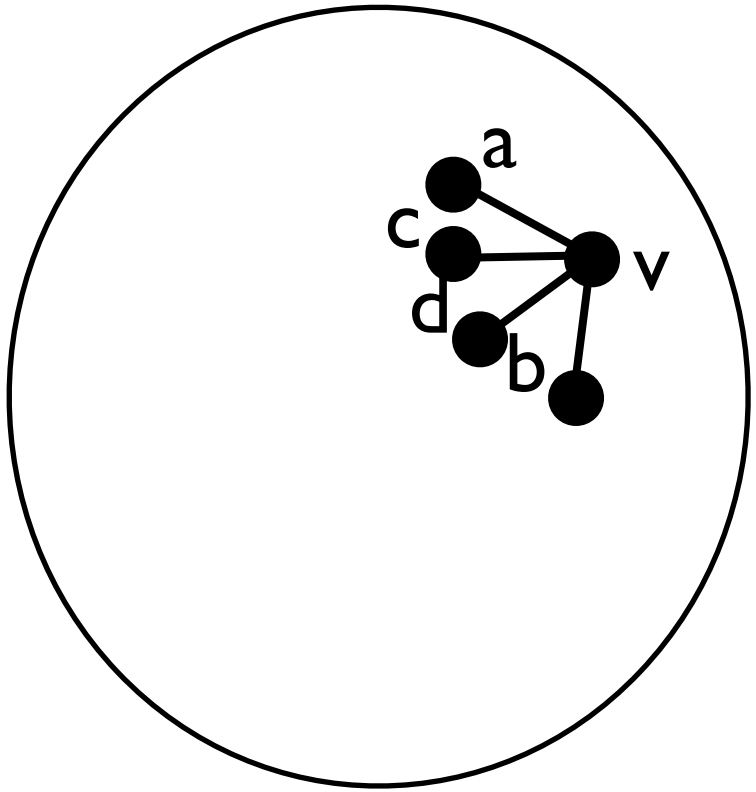
- Idee: Breitendurchlauf



Queue: a c d

Implementierung von Datent.
Graphen - Durchlaufen

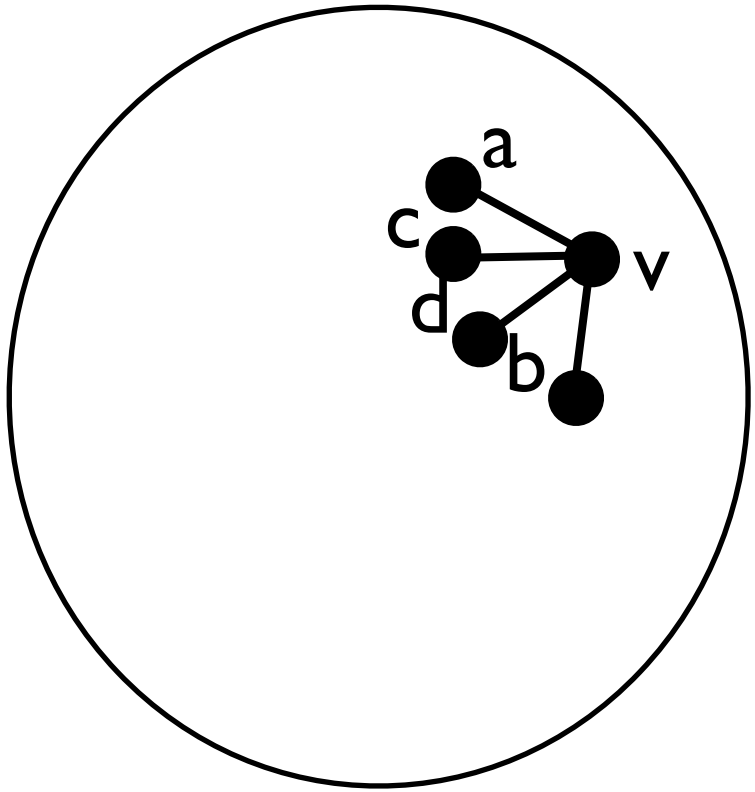
- Idee: Breitendurchlauf



Queue: a c d

Implementierung von Datent.
Graphen - Durchlaufen

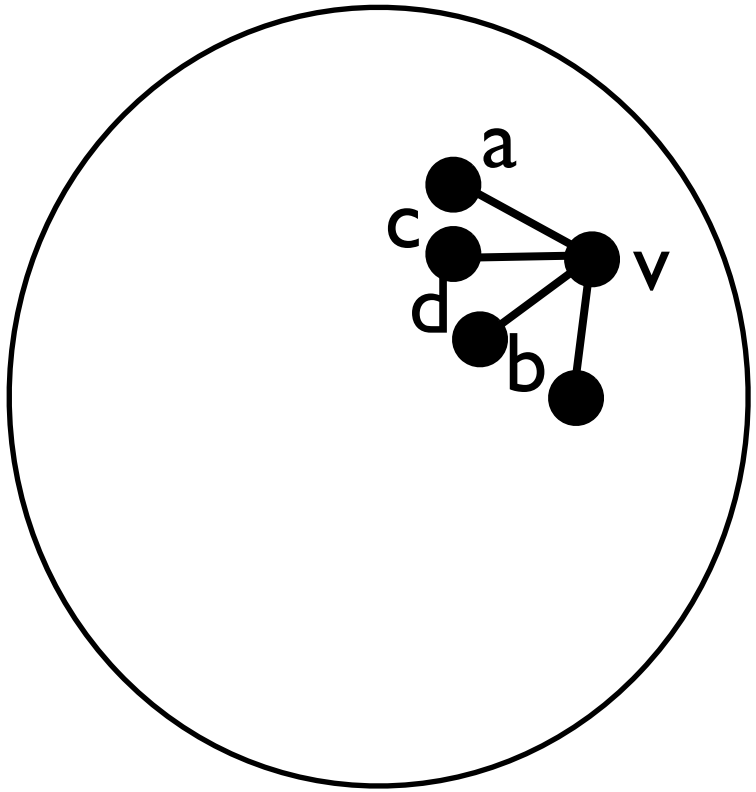
- Idee: Breitendurchlauf



Queue: a c d b

Implementierung von Datent.
Graphen - Durchlaufen

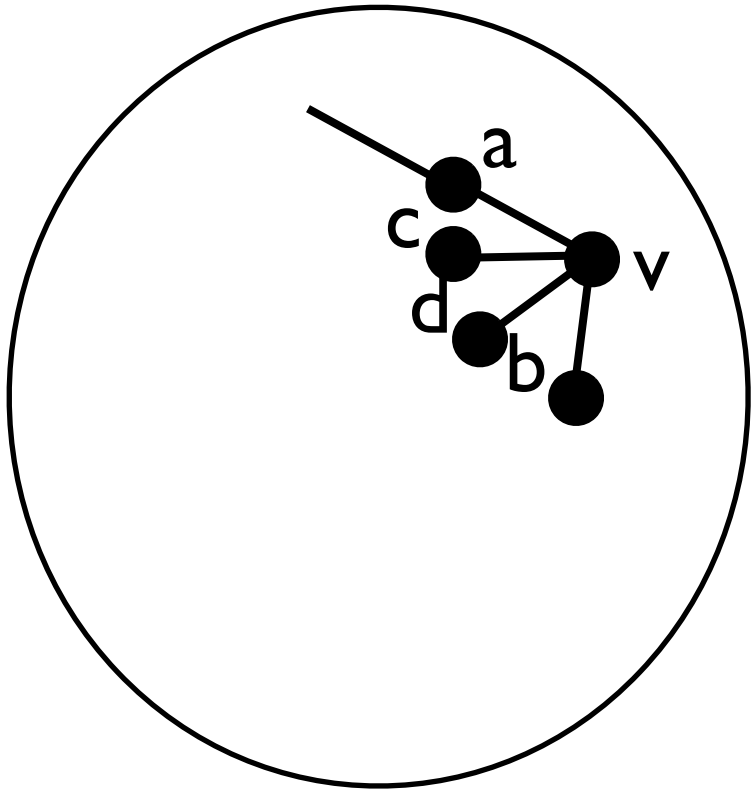
- Idee: Breitendurchlauf



Queue: c d b

Implementierung von Datent.
Graphen - Durchlaufen

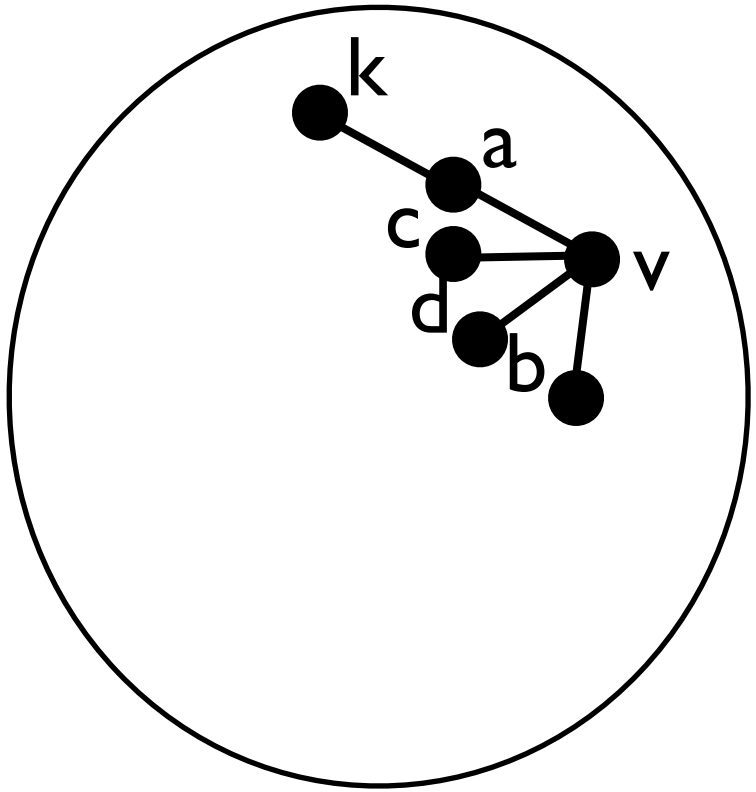
- Idee: Breitendurchlauf



Queue: c d b

Implementierung von Datent.
Graphen - Durchlaufen

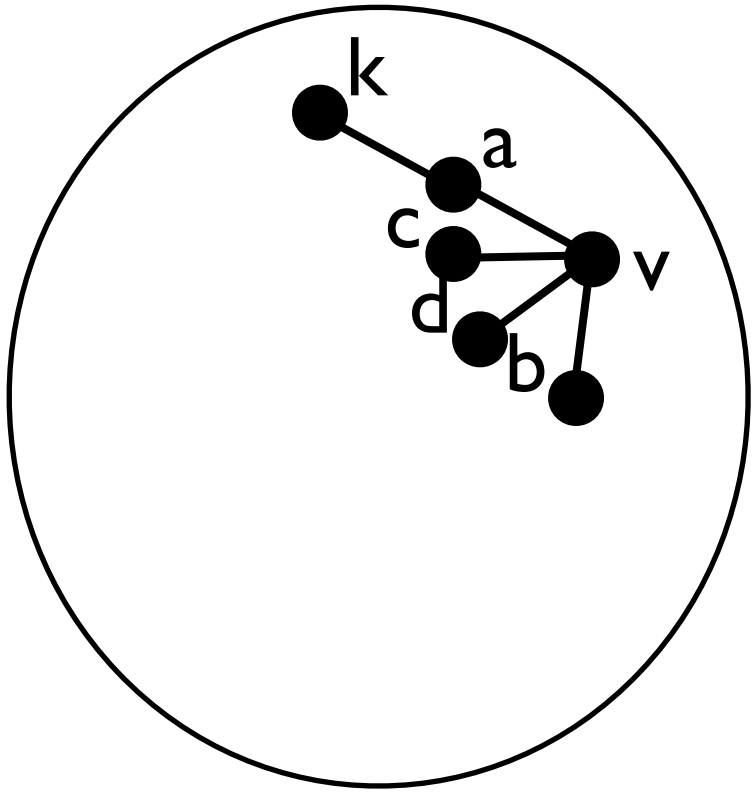
- Idee: Breitendurchlauf



Queue: c d b

Implementierung von Datent.
Graphen - Durchlaufen

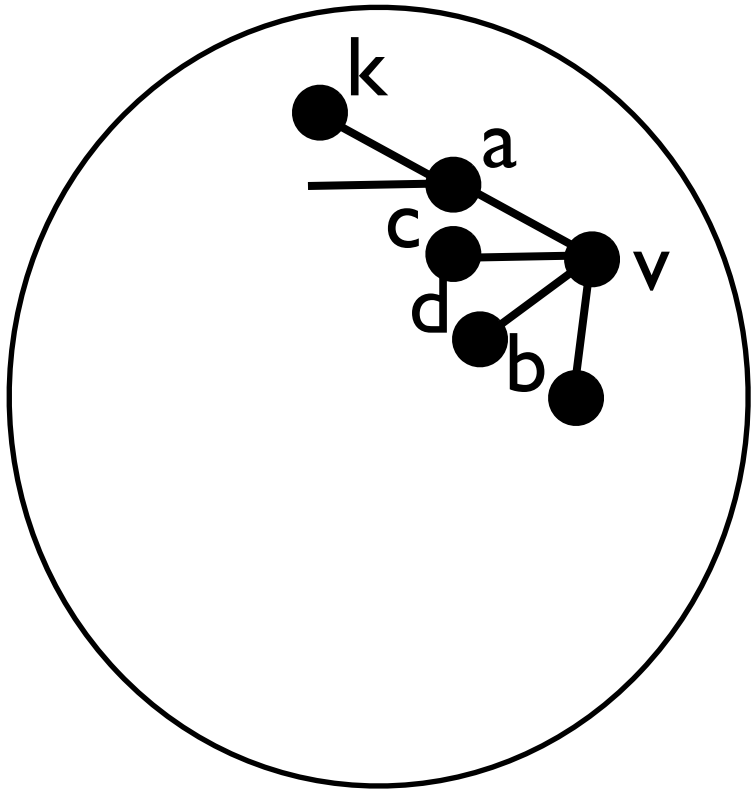
- Idee: Breitendurchlauf



Queue: c d b k

Implementierung von Datent.
Graphen - Durchlaufen

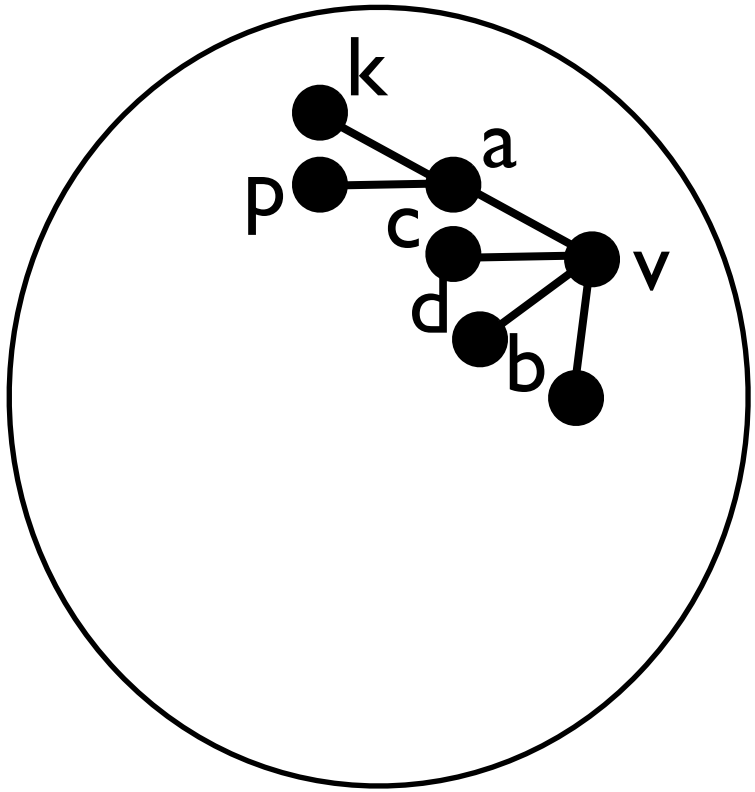
- Idee: Breitendurchlauf



Queue: c d b k

Implementierung von Datent.
Graphen - Durchlaufen

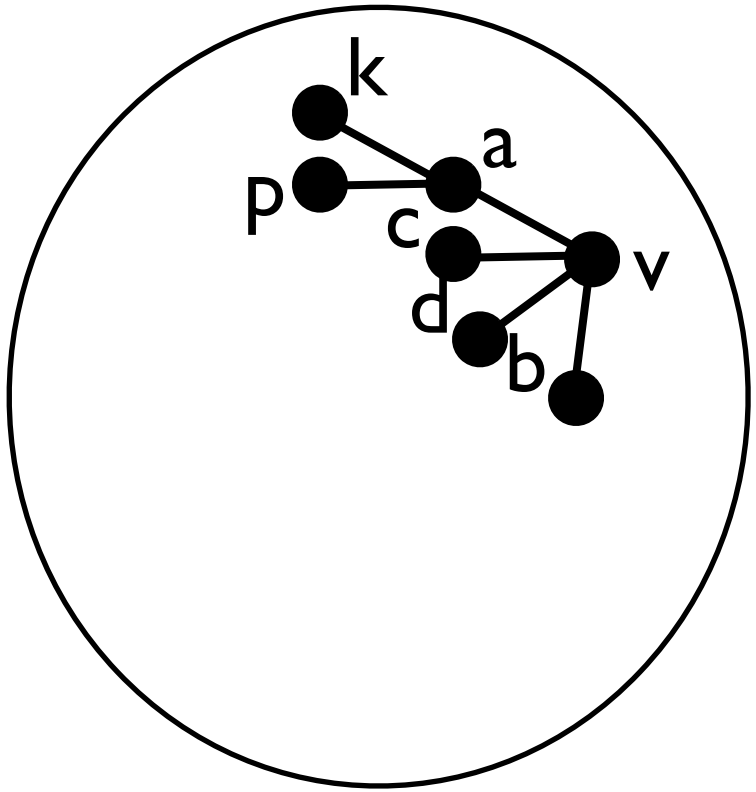
- Idee: Breitendurchlauf



Queue: c d b k

Implementierung von Datent.
Graphen - Durchlaufen

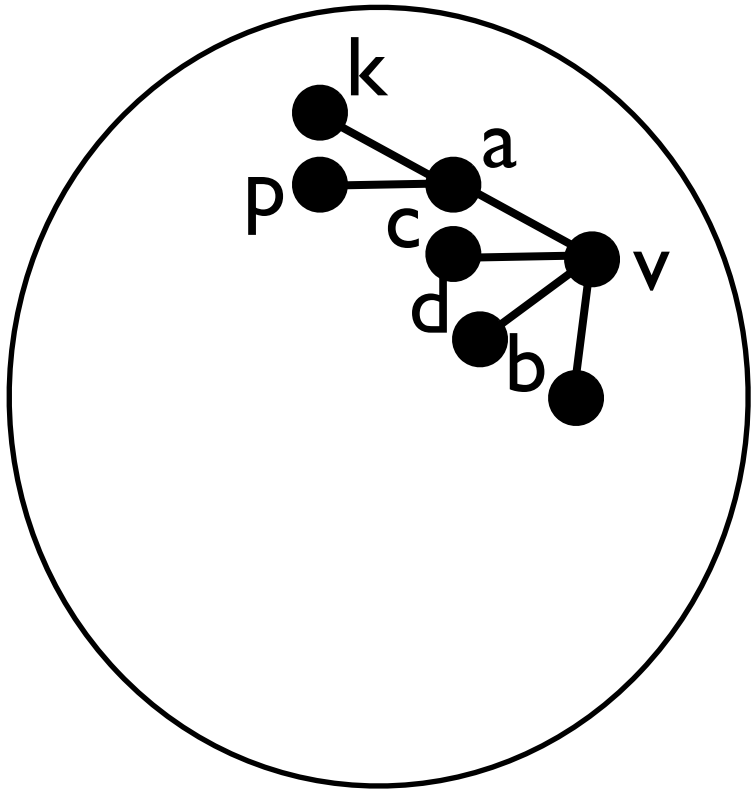
- Idee: Breitendurchlauf



Queue: c d b k p

Implementierung von Datent.
Graphen - Durchlaufen

- Idee: Breitendurchlauf



Queue: d b k p

Implementierung von Datent.
Schlußbeispiel - topol. Sortieren

- Sortieren von Daten, auf denen *keine* vollständige Ordnung vorliegt, sondern *partielle* Ordnung: einige Elemente vergleichbar, andere nicht
- Aufgabe: Sortiere Elemente der Menge so, daß für alle $i,j \in \{1,...,n\}$ mit $i \neq j$ aus $a_i < a_j$ immer $i < j$ folgt

Implementierung von Datent.
topol. Sortieren - Beispiel

Beispiel:

- geg.: $M=\{a_1,...,a_9\}$ mit $a_1<a_3$, $a_3<a_7$, $a_7<a_4$, $a_7<a_5$, $a_4<a_6$, $a_9<a_2$, $a_9<a_5$, $a_2<a_8$, $a_5<a_8$, $a_8<a_6$
- z.B.: a_4 und a_8 nicht vergleichbar
- eine topologische Sortierung:

  $a_1,a_3,a_7,a_4,a_9,a_2,a_5,a_8,a_6$

- andere topologische Sortierung:

  $a_9,a_1,a_2,a_3,a_7,a_5,a_8,a_4,a_6$

Implementierung von Datent.
topol. Sortieren - Anwendung

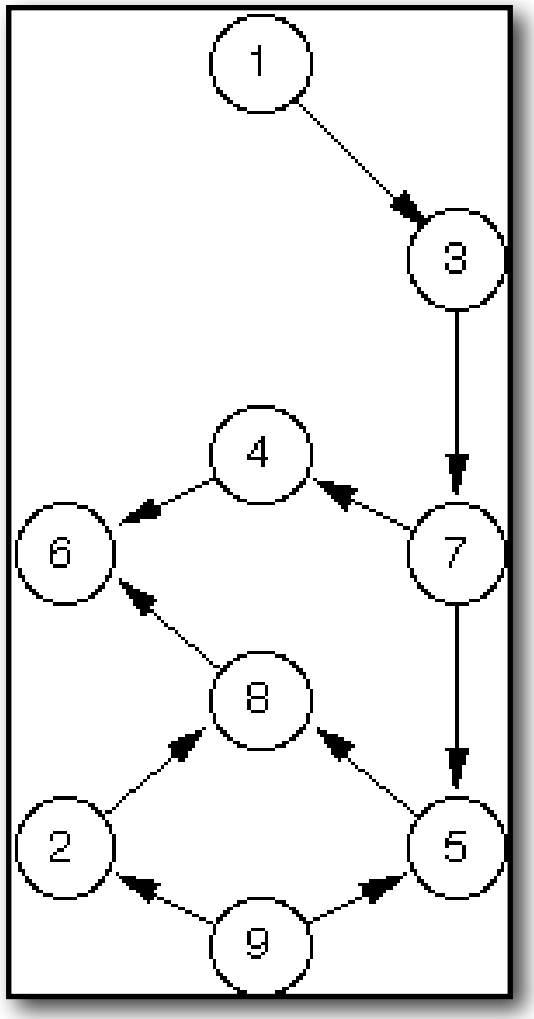
Anwendung:

- Projektplanung, z.B. ordne Arbeitsvorgänge beim Hausbau
- Schreiben eine Lehrbuchs, z.B. ordne Definition und Gebrauch von Begriffen
- Studienplan., z.B. ordne Vorlesungen hinsichtlich erforderlicher Vorkenntnisse

Implementierung von Datent.

topol. Sortieren - Graphdarstellung

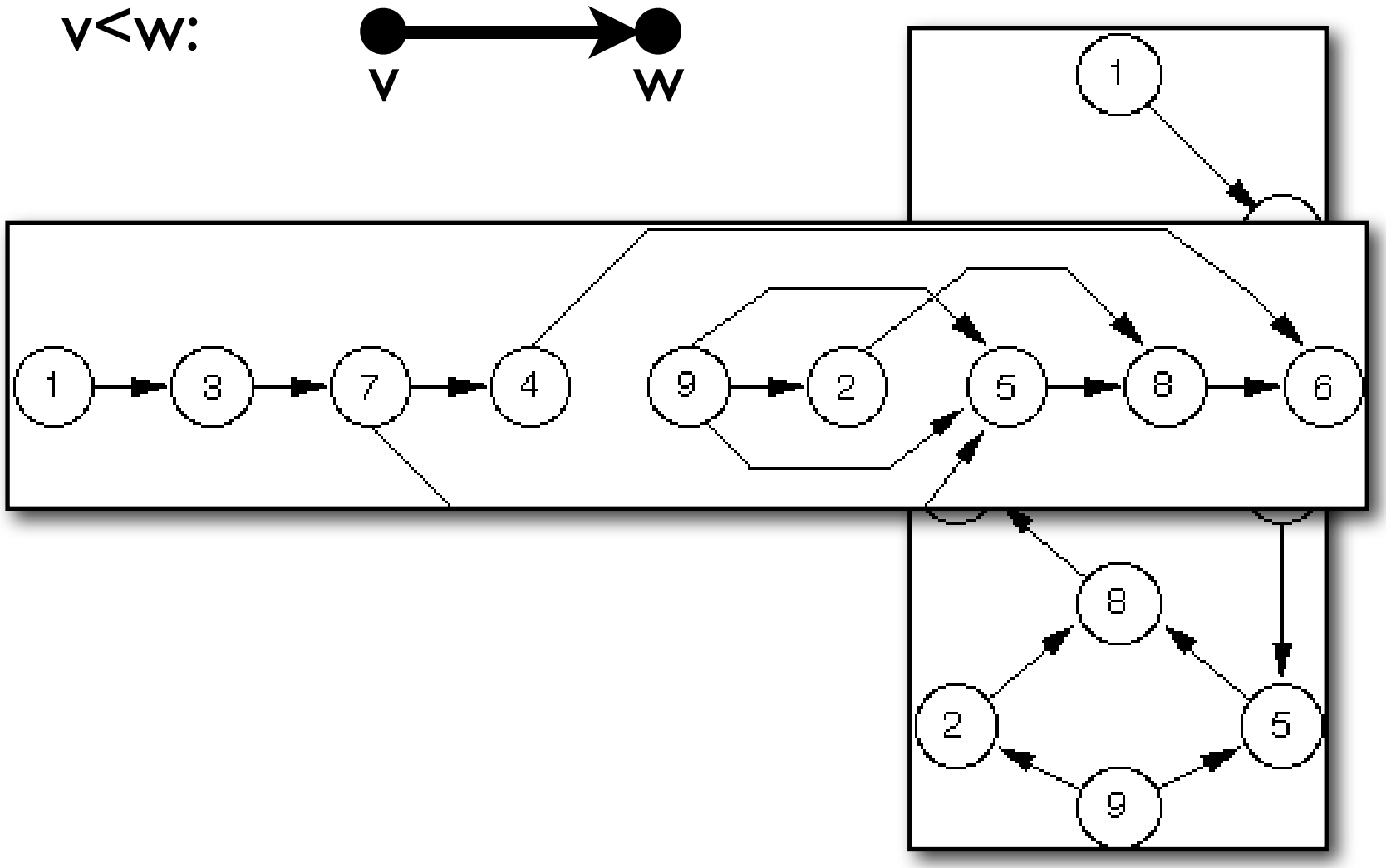
v<w:



Implementierung von Datent.

topol. Sortieren - Graphdarstellung

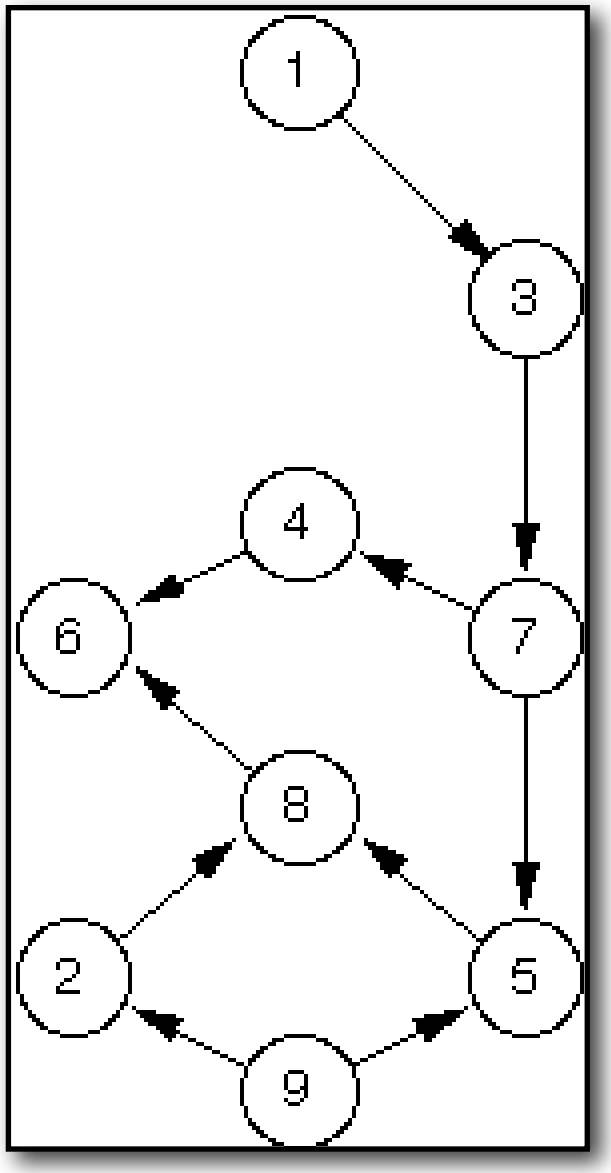
v<w:



Implementierung von Datent.

topol. Sortieren - Lösungsidee

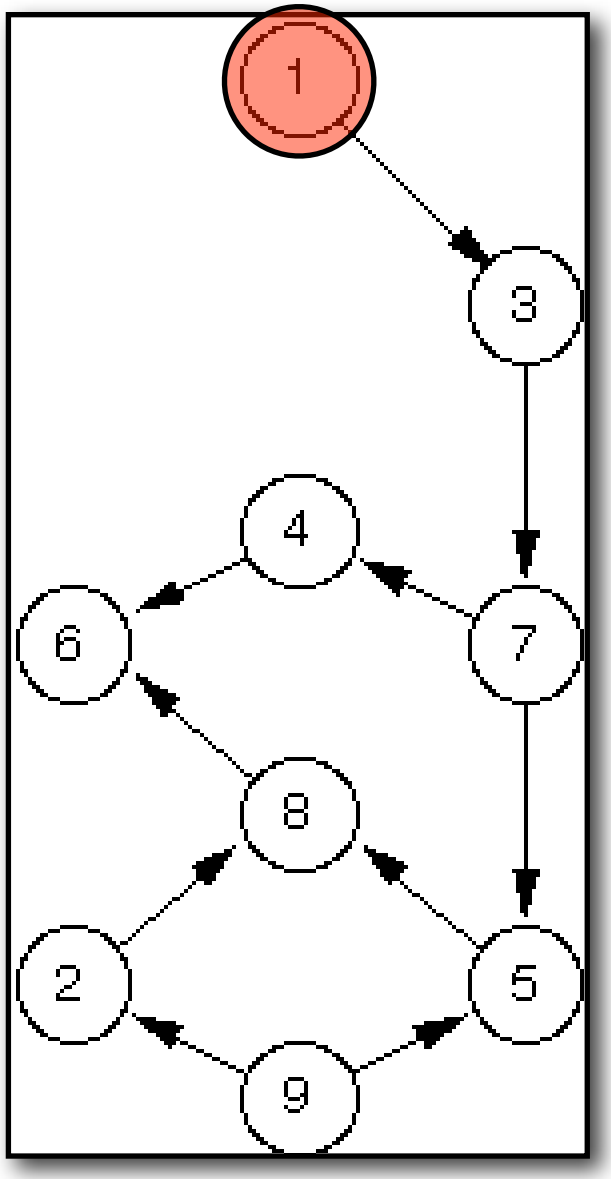
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

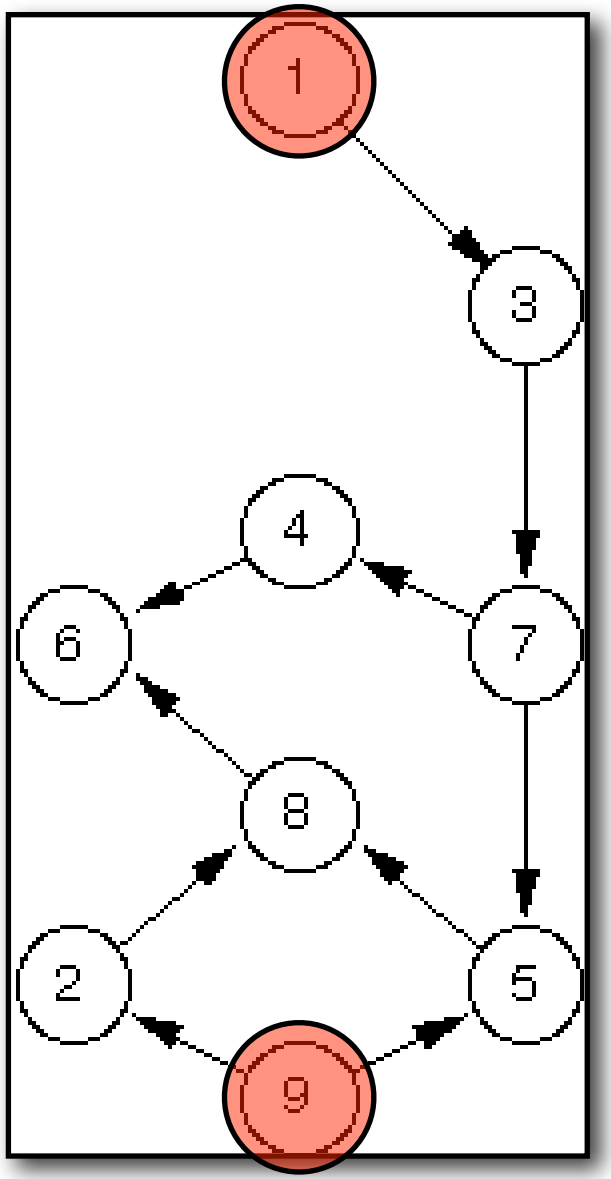
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

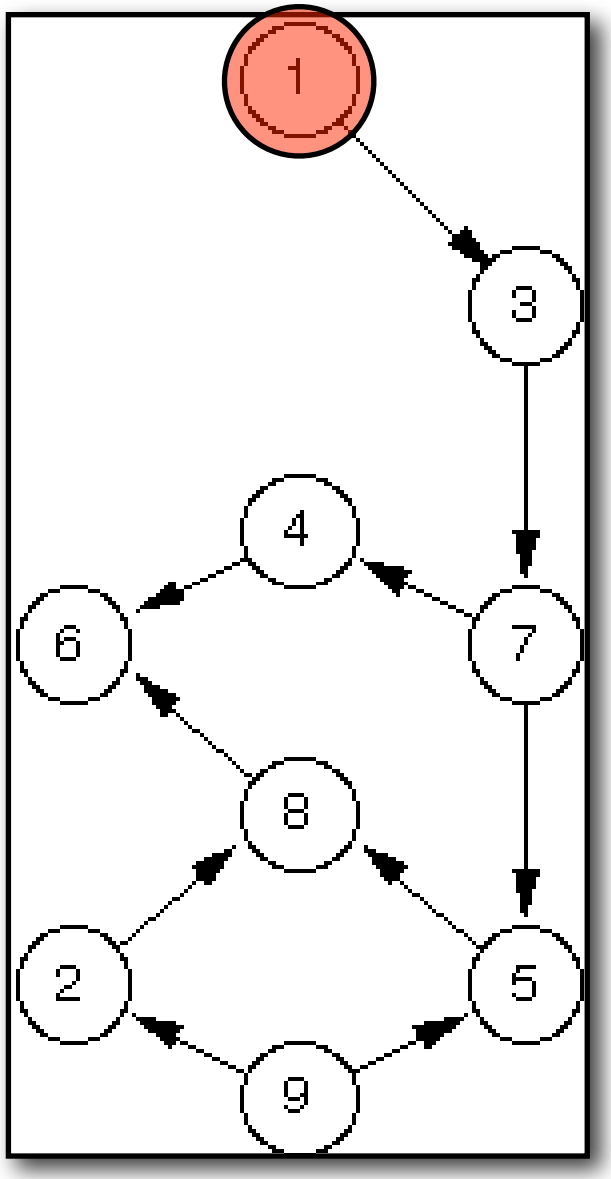
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

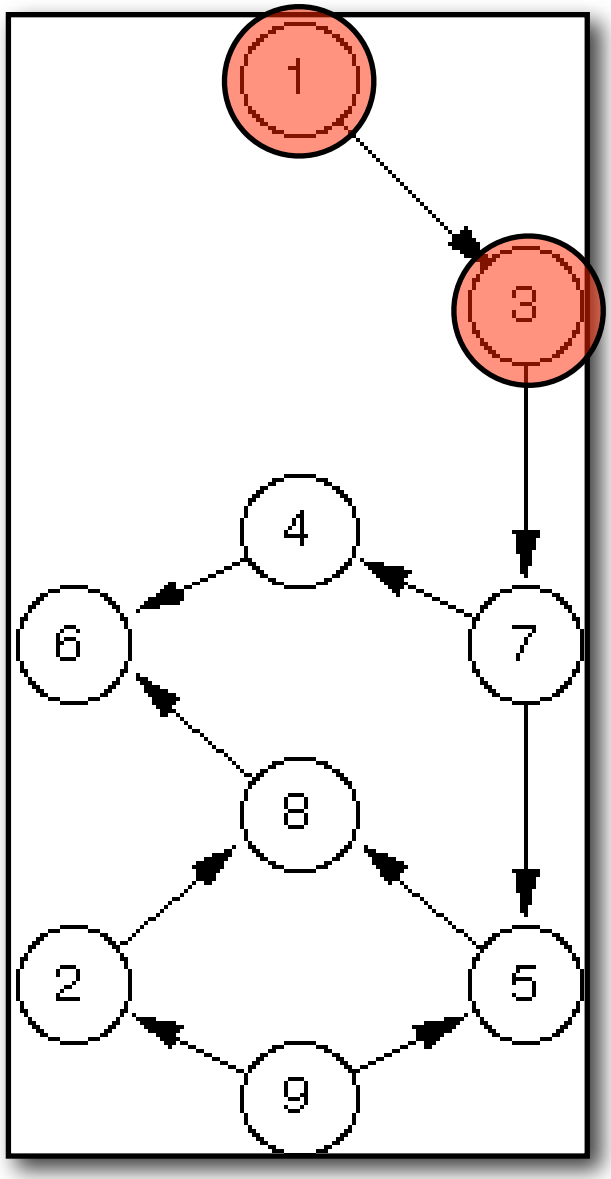
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

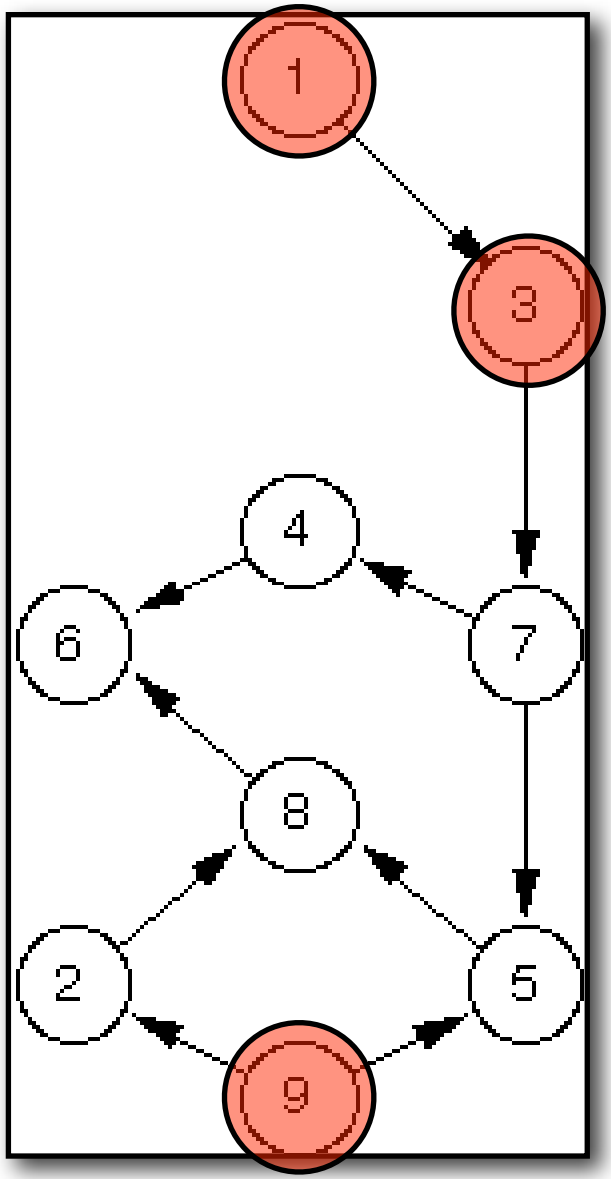
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

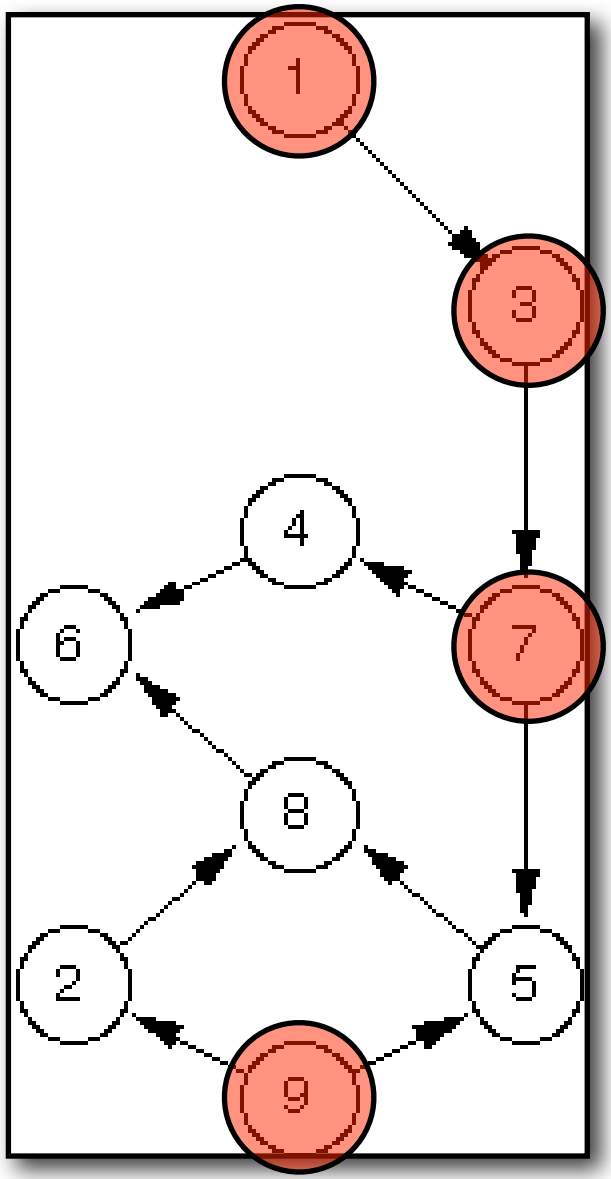
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

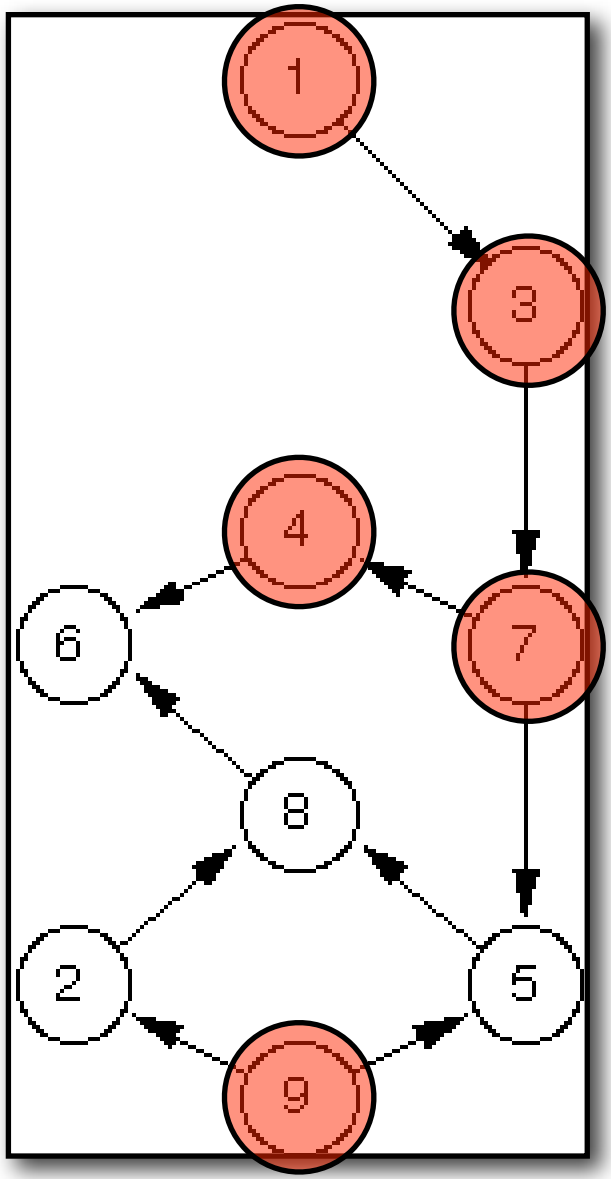
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

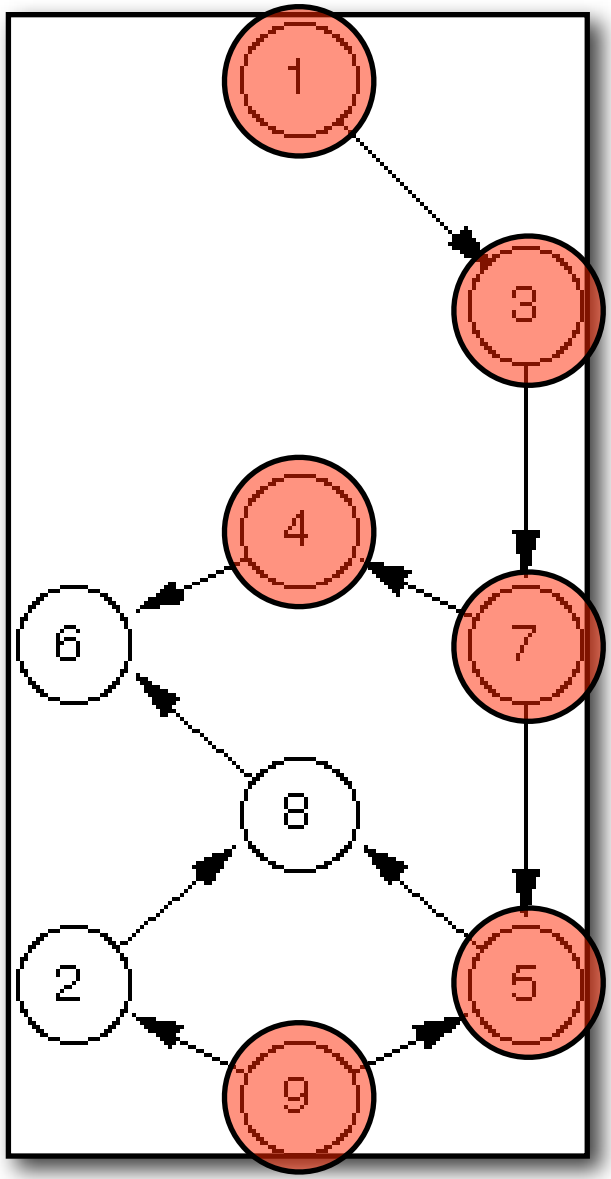
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

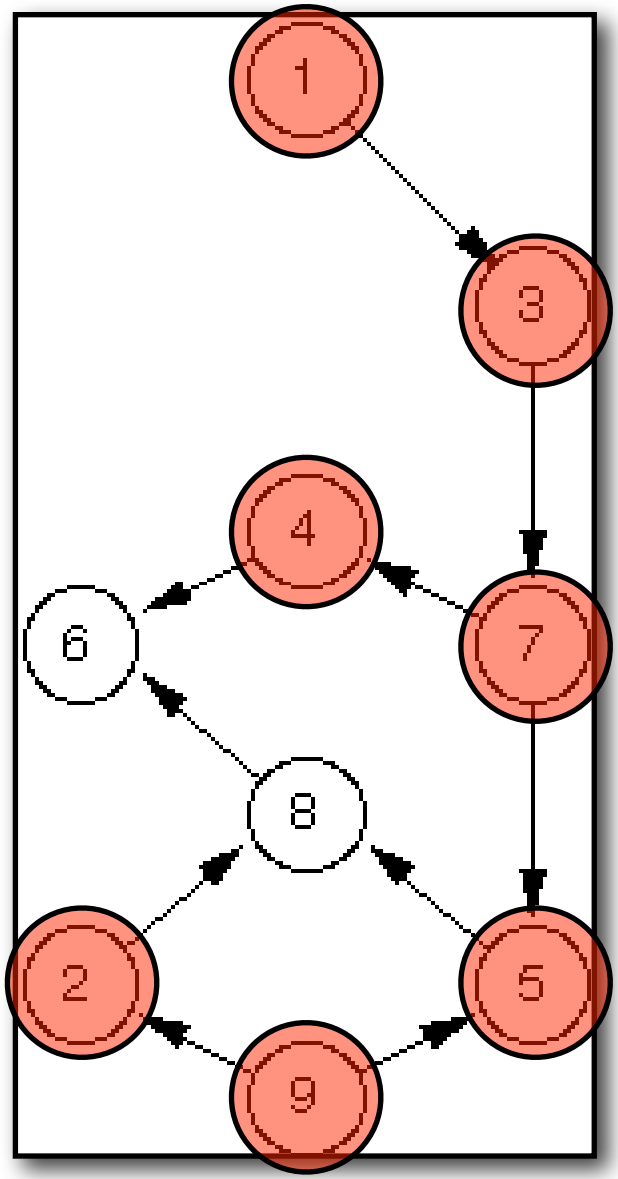
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

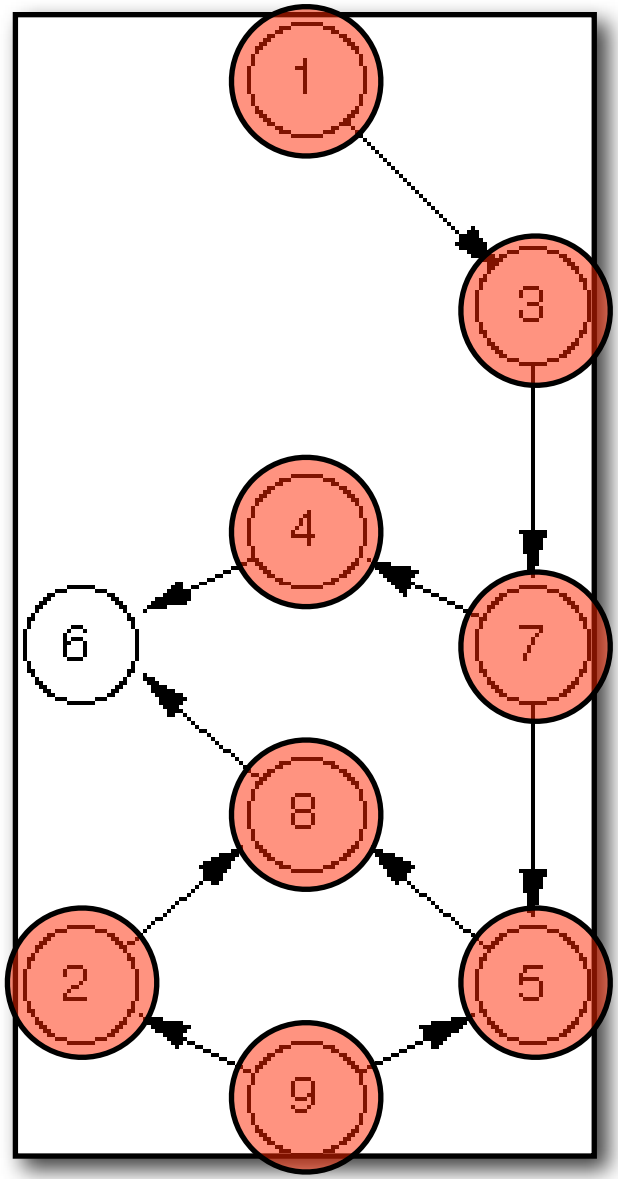
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

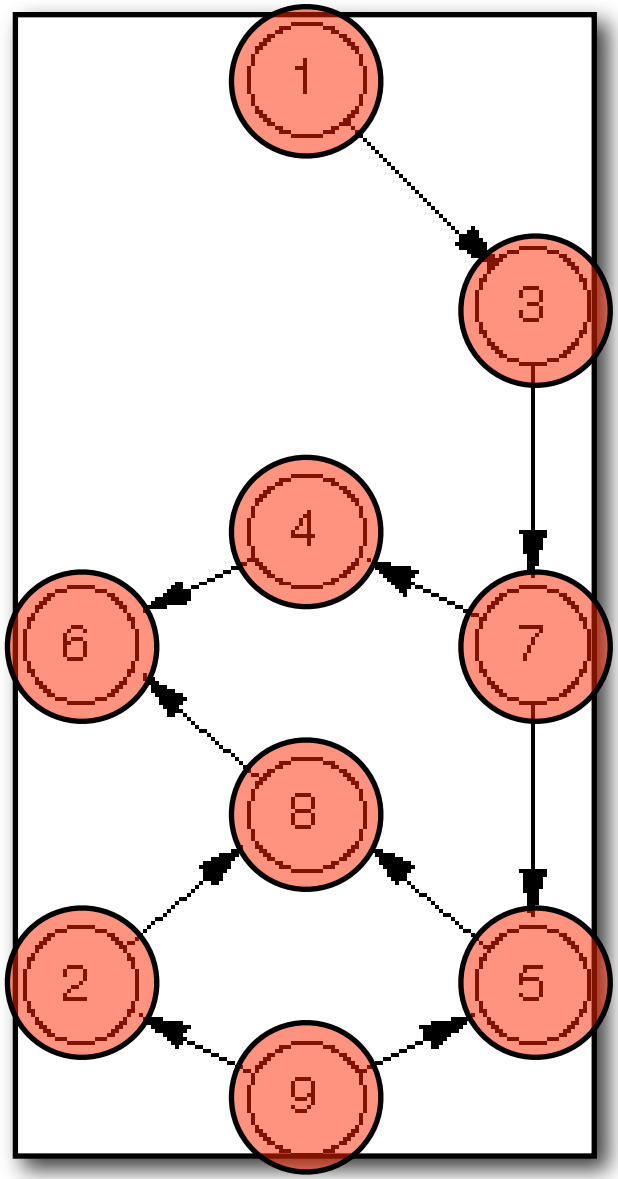
- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Lösungsidee

- Suche Knoten mit Eingangsgrad 0
- dieser Knoten hat keinen Vorgänger
- gib Markierung aus
- entferne Knoten
- wiederhole Verfahren bis Graph leer



Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
  if V≠∅ then
   begin
     Wähle v∈V mit d+(v)=0;
     writeln(v);
     topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

geschickte Imple-mentierung nötig

Methode 1:
jedesmal Graphen nach Knoten mit Eingangsgrad 0 absuchen
=> zeitaufwendig, schlecht

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

geschickte Imple-mentierung nötig

Methode 2:
zu Beginn alle Eingangsgrade ermitteln; beim Löschen nur Eingangsgrade adjazenter Knoten aktualisieren
=> deutlich schneller

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

Implementierung von Datent.
topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

geschickte Imple-mentierung nötig

Ergänzung:
Knoten mit Eingangs-grad 0 in Queue sammeln und der Reihe nach abarbeiten

Implementierung von Datent.

topol. Sortieren - Algorithmus

```
procedure topsort (G=(V,E): graph);
begin
   if V≠∅ then
   begin
      Wähle v∈V mit d+(v)=0;
      writeln(v);
      topsort(G\{v})
   end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

```
var d+: array [V] of integer;
    q: queue of V;
begin
    read(G=(V,E));
    empty(q);
    for all v∈V do d+(v):=0;
    for all v∈V do
    begin
        for all w∈V: (w,v)∈E do d+(v):=d+(v)+1;
        if d+(v)=0 then enter(v,q)
    end;
    while not is_empty(q) do
    begin
        v:=first(q); remove(q);
        writeln(v,N);
        for all w∈V: (v,w)∈E do
        begin
            d+(w):=d+(w)-1;
            if d+(w)=0 then enter(w,q)
        end
    end
end.
```

Implementierung von Datent.
topol. Sortieren - Auswertung

- obiger Algorithmus effizienter
- Effizienz im Moment noch nicht genau bestimmbar
- exakte numerische Maße fehlen, um Algorithmen genauer zu vergleichen
- Maße und Meßverfahren im nächsten Kapitel