Algorithmen und Datenstrukturen

Prof. Dr. Ralf Möller

Universität zu Lübeck

Institut für Informationssysteme

Magnus Bender (Übungen)

sowie viele Tutoren

Sortierung in linearer Zeit

- Sortieren: Geht es doch noch schneller als in $\Omega(n \log n)$ Schritten?
- Man muss „schärfere“ Annahmen über das Problem machen können ...
  - z.B. Schlüssel in n Feldelementen aus dem Bereich [1..n]
- ... oder Nebenbedingungen „abschwächen“
  - z.B. die In-situ-Einschränkung aufgeben
- Zentrale Idee: Vermeide Vergleiche!

Seward, H. H. (**1954**), "2.4.6 Internal Sorting by Floating Digital Sort", Information sorting in the application of electronic digital computers to business operations, Master's thesis, Report R-232, Massachusetts Institute of Technology, Digital Computer Laboratory, pp. 25–28

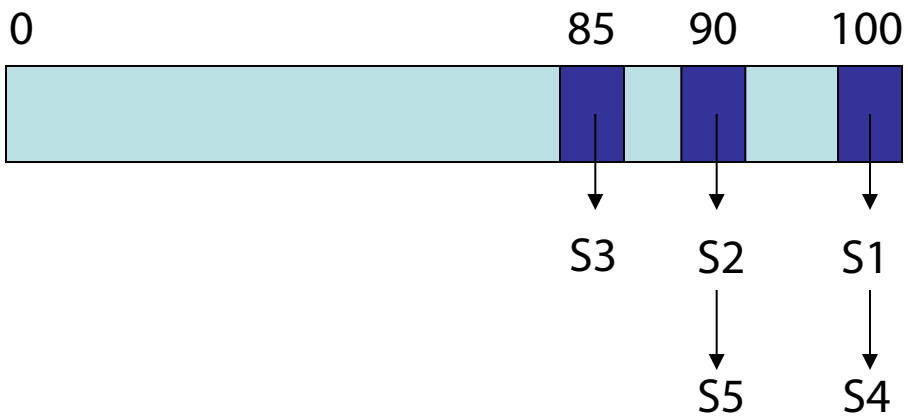
A. Andersson, T. Hagerup, S. Nilsson, R. Raman, Sorting in Linear Time?, J. Comput. Syst. Sci. 57(1): 74-93, **1998**

Sortieren durch Zählen / Counting-Sort

- **Wissen:**
  Schlüssel fallen in einen kleinen Zahlenbereich
- **Beispiel 1:** Sortiere eine Menge von Studierenden nach Examensbewertungen (Scores sind Zahlen)
  - 1000 Studenten
  - Maximum score: 100
  - Minimum score: 0
- **Beispiel 2:** Sortiere Studierende nach dem ersten Buchstaben des Nachnamens
  - Anzahl der Studierenden: viele
  - Anzahl der Buchstaben: 26+wenige

Intuition

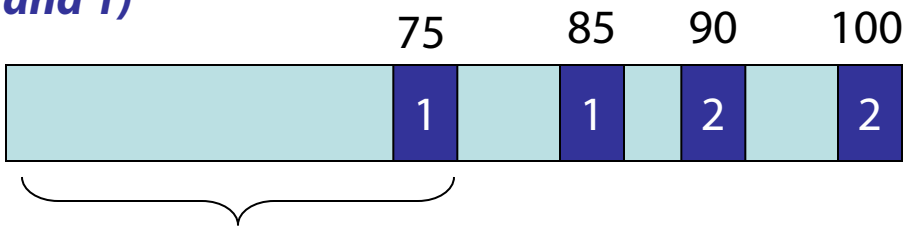
- S1: 100
- S2: 90
- S3: 85
- S4: 100
- S5: 90
- …



… S3 … S2, S5, …, S1, S4

Intuition

Hilfsspeicher (Zustand 1)



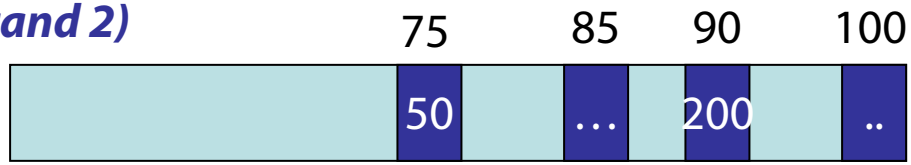
50 Studierende mit Score ≤ 75
Was ist der Rang (von klein auf groß) für einen Studenten mit Score 75?

50

200 Studierende mit Score ≤ 90
Was ist der Rang für einen Studenten mit Score 90?

200

Hilfsspeicher (Zustand 2)



Counting-Sort

- ***Eingabe***: $A[1 \,..\, n]$, wobei $\text{key}(A[\,j]) \in \{1, 2, \ldots, k\}$.
- ***Ausgabe***: $B[1 \,..\, n]$, sortiert.
- ***Hilfsspeicher***: $C[1 \,..\, k]$.

- Kein In-situ-Sortieralgorithmus
- Benötigt θ(n+k) zusätzliche Speicherplätze

Counting-Sort

```
1. for i = 1:k
     C[i] = 0
   end
```

Initialisiere

```
2. for j = 1:n
     C[A[j]] = C[A[j]] + 1
   end
```

Zähle

▷ $C[i] = |\{\text{key} = i\}|$

```
3. for i = 2:k
     C[i] = C[i] + C[i - 1]
   end
```

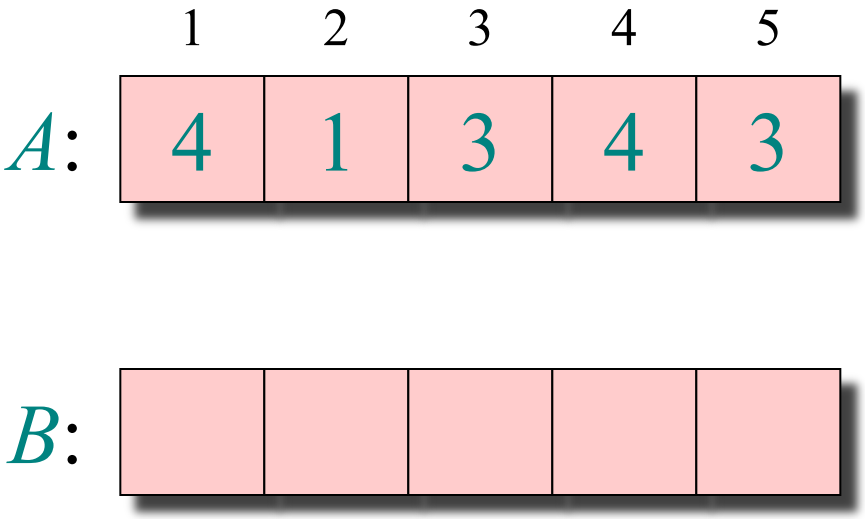
Bestimme Summe

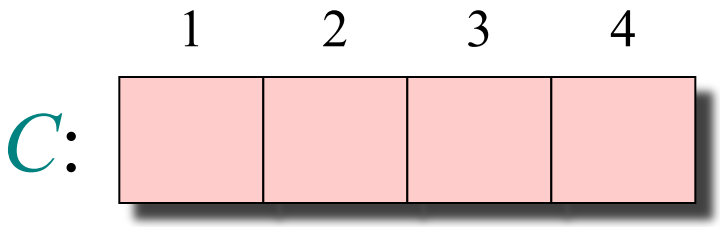
▷ $C[i] = |\{\text{key} \leq i\}|$

```
4. for j = n:-1:1
     B[C[A[j]]] = A[j]
     C[A[j]] = C[A[j]] - 1
   end
```

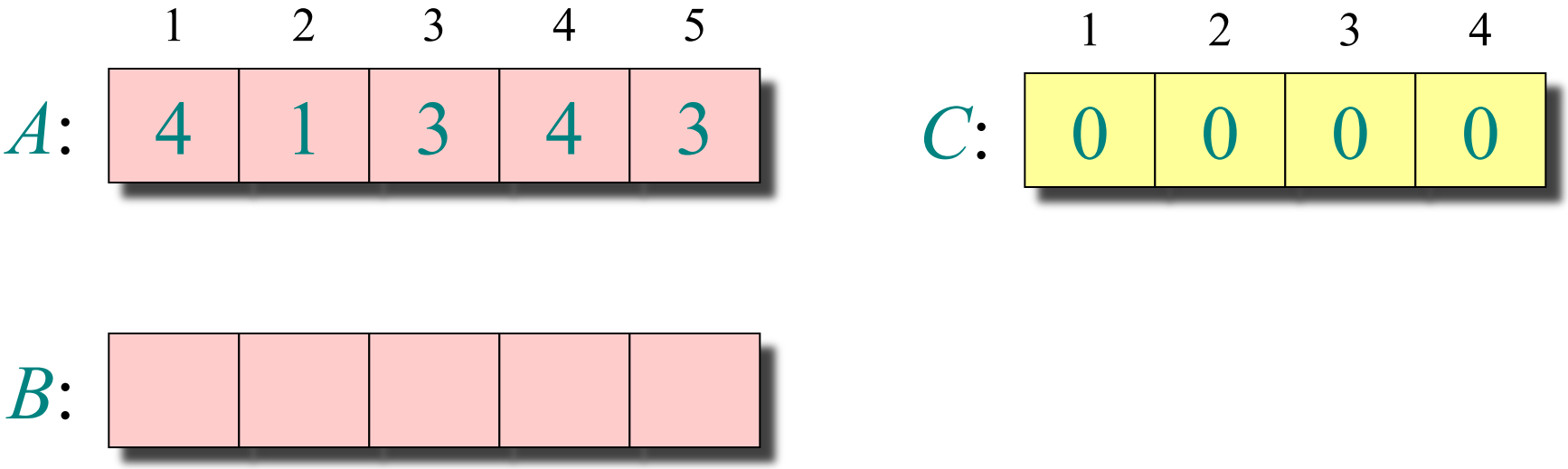
Ordne neu

Counting-Sort Beispiel



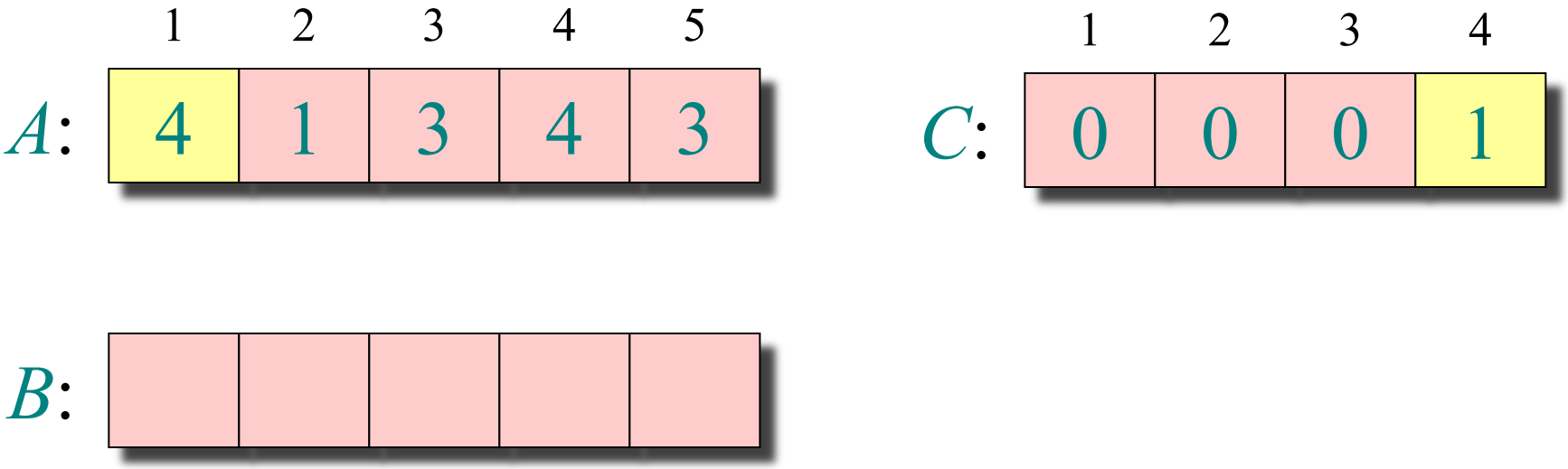


Schleife 1: Initialisierung




```
1. for i = 1:k
       C[i] = 0
   end
```

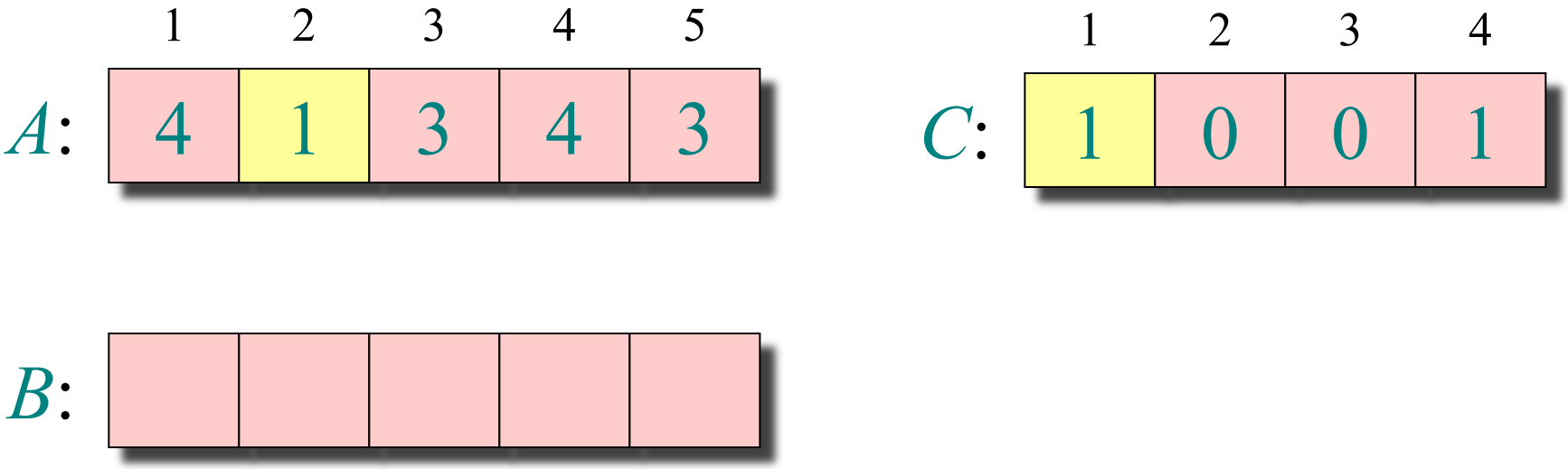
Schleife 2: Zähle



2.
```
for j = 1:n
    C[A[j]] = C[A[j]] + 1
end
```

▷ $C[i] = |\{\text{key} = i\}|$

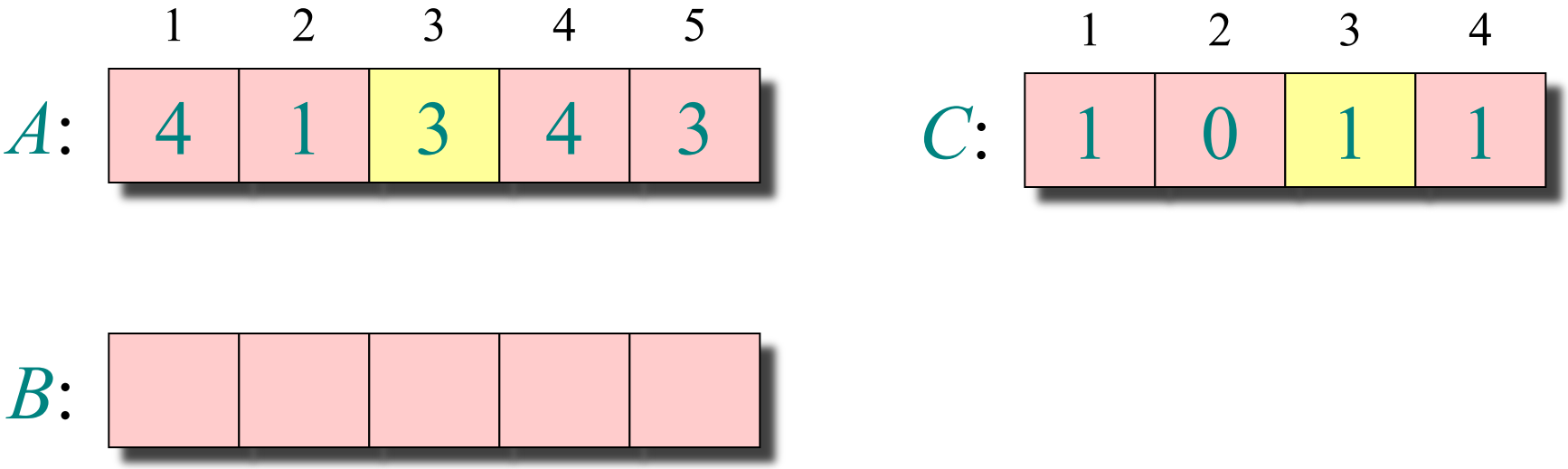
Schleife 2: Zähle



2.
```
for j = 1:n
    C[A[j]] = C[A[j]] + 1
end
```

▷ $C[i] = |\{\text{key} = i\}|$

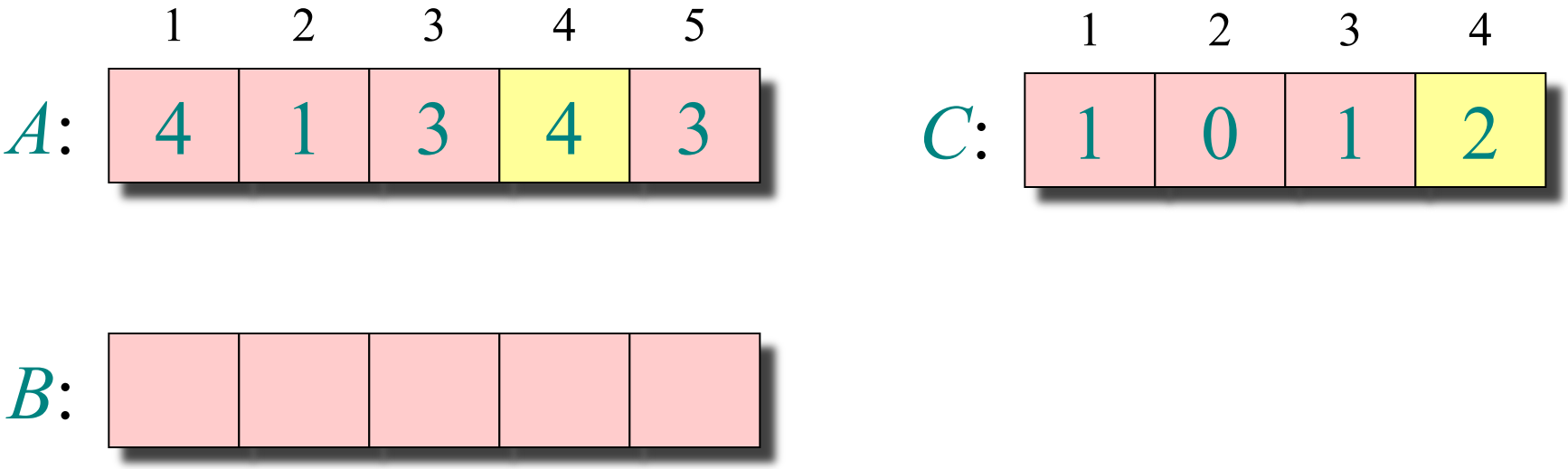
Schleife 2: Zähle



2.
```
for j = 1:n
    C[A[j]] = C[A[j]] + 1
end
```

▷ $C[i] = |\{\text{key} = i\}|$

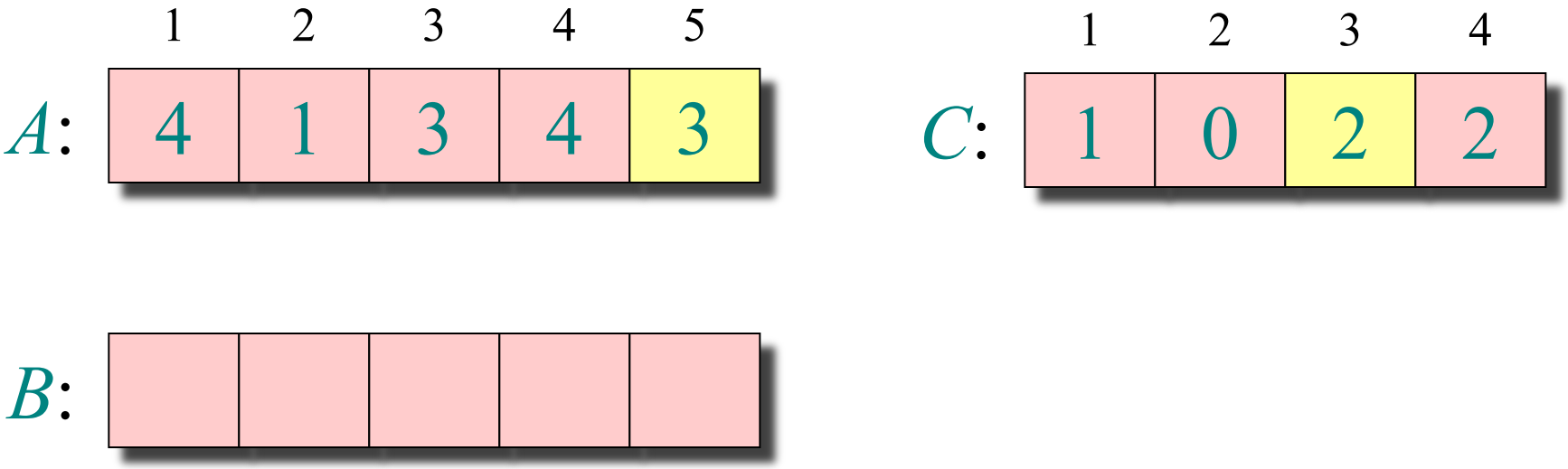
Schleife 2: Zähle



2.
```
for j = 1:n
    C[A[j]] = C[A[j]] + 1
end
```

▷ $C[i] = |\{\text{key} = i\}|$

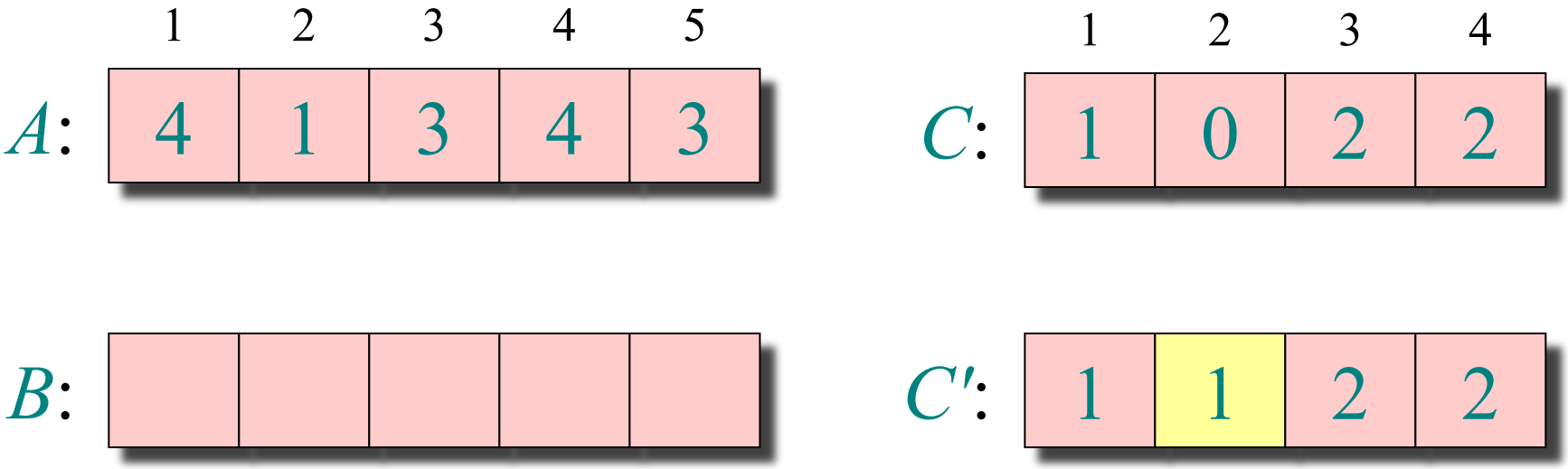
Schleife 2: Zähle



2.
```
for j = 1:n
    C[A[j]] = C[A[j]] + 1
end
```

▷ $C[i] = |\{\text{key} = i\}|$

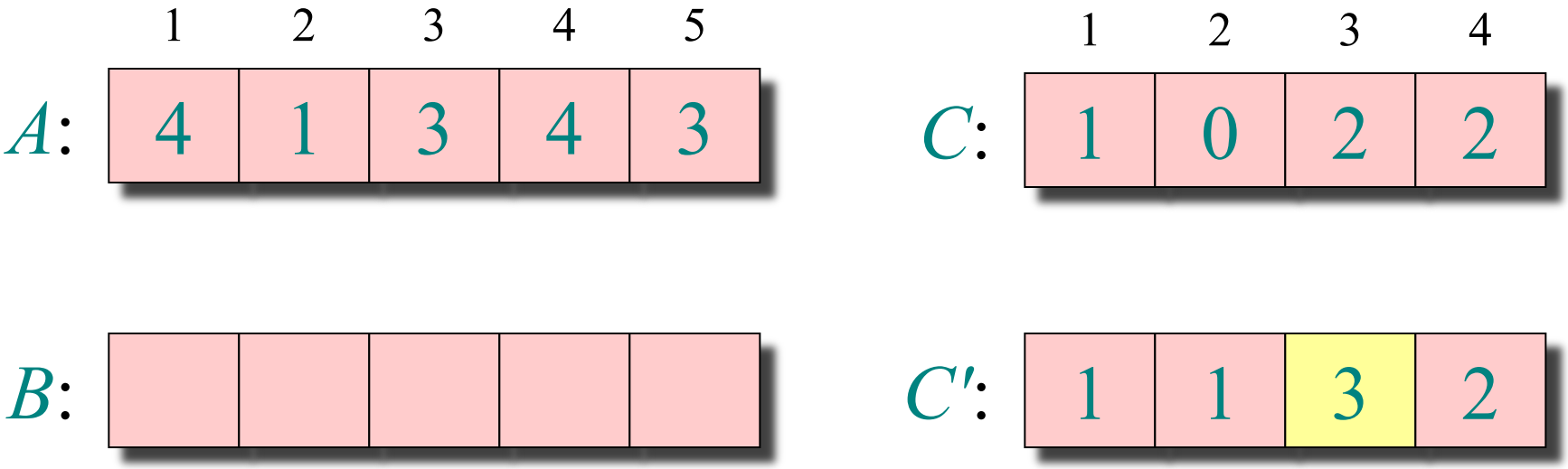
Schleife 3: Berechne Summe



3.
```
for i = 2:k
    C[i] = C[i] + C[i - 1]
end
```

▷ $C[i] = |\{\text{key} \leq i\}|$

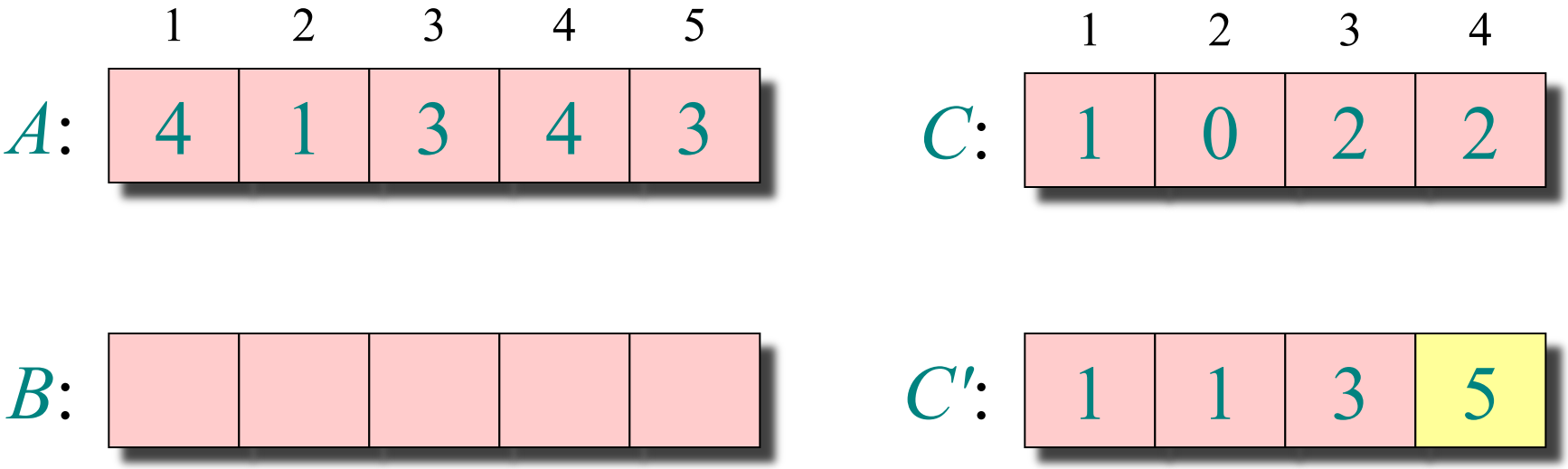
Schleife 3: Berechne Summe



3.
```
for i = 2:k
    C[i] = C[i] + C[i - 1]
end
```

$\triangleright\ C[i] = |\{\text{key} \le i\}|$

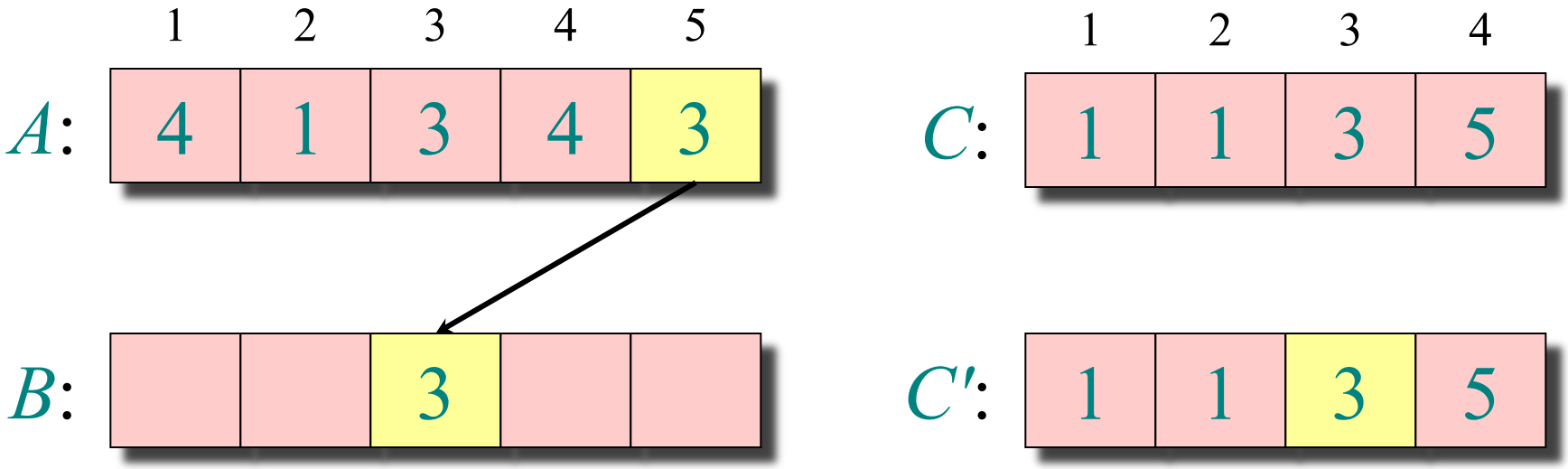
Schleife 3: Berechne Summe



3.
```
for i = 2:k
    C[i] = C[i] + C[i - 1]
end
```

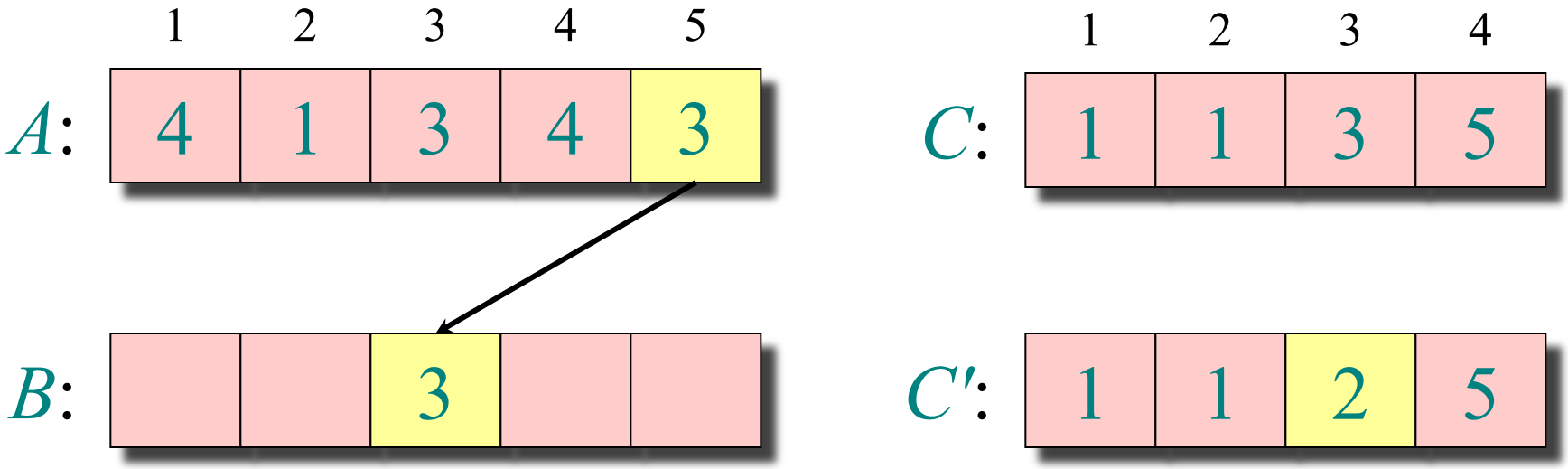
$\triangleright\ C[i] = |\{\text{key} \le i\}|$

Schleife 4: Ordne neu



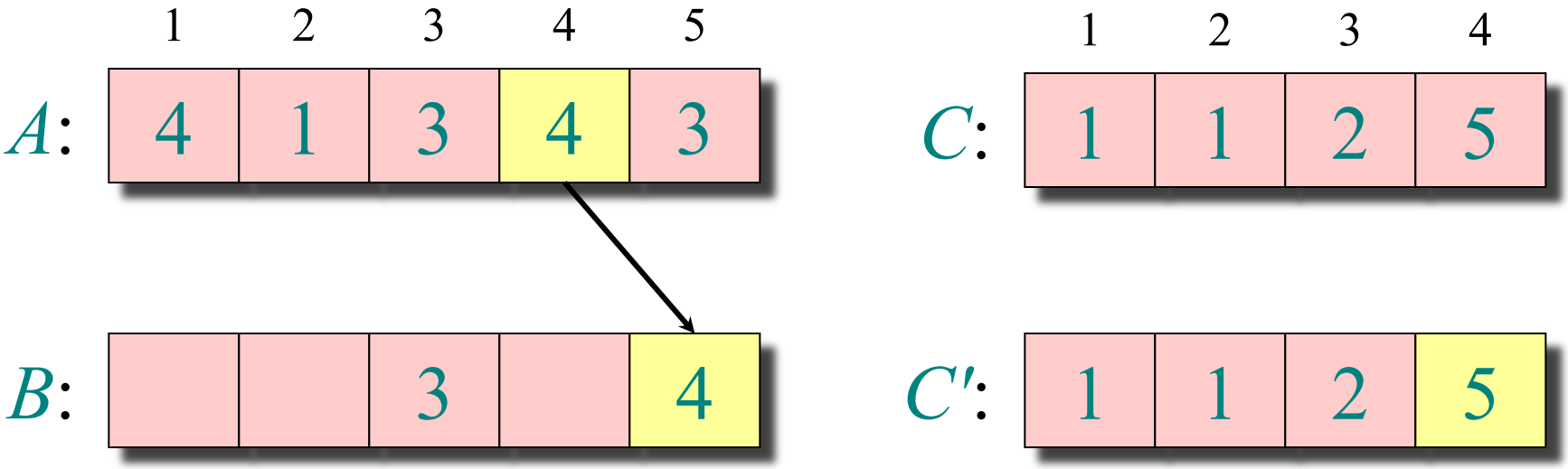
4.
```
for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
end
```

Schleife 4: Ordne neu



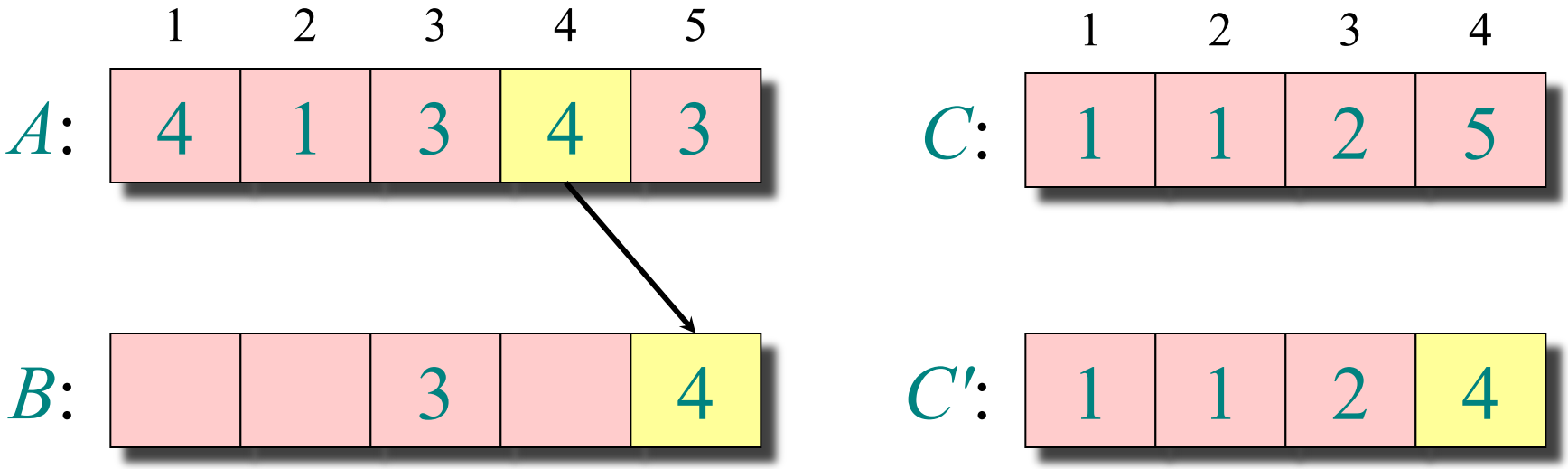
4.
```
for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
end
```

Schleife 4: Ordne neu



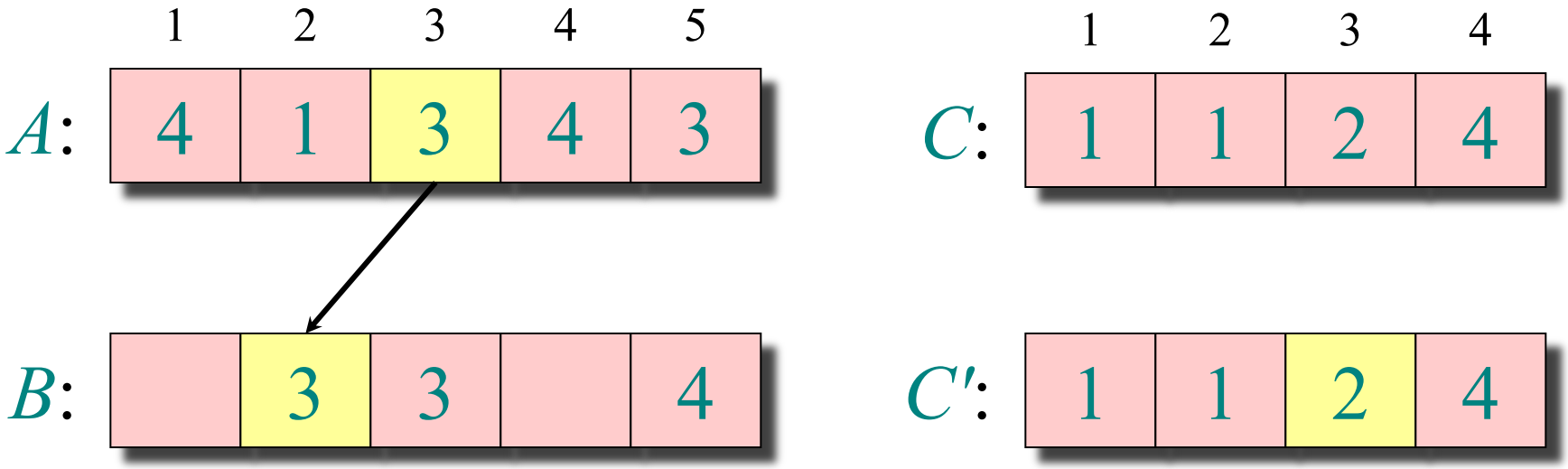
4.
```
for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
end
```

Schleife 4: Ordne neu



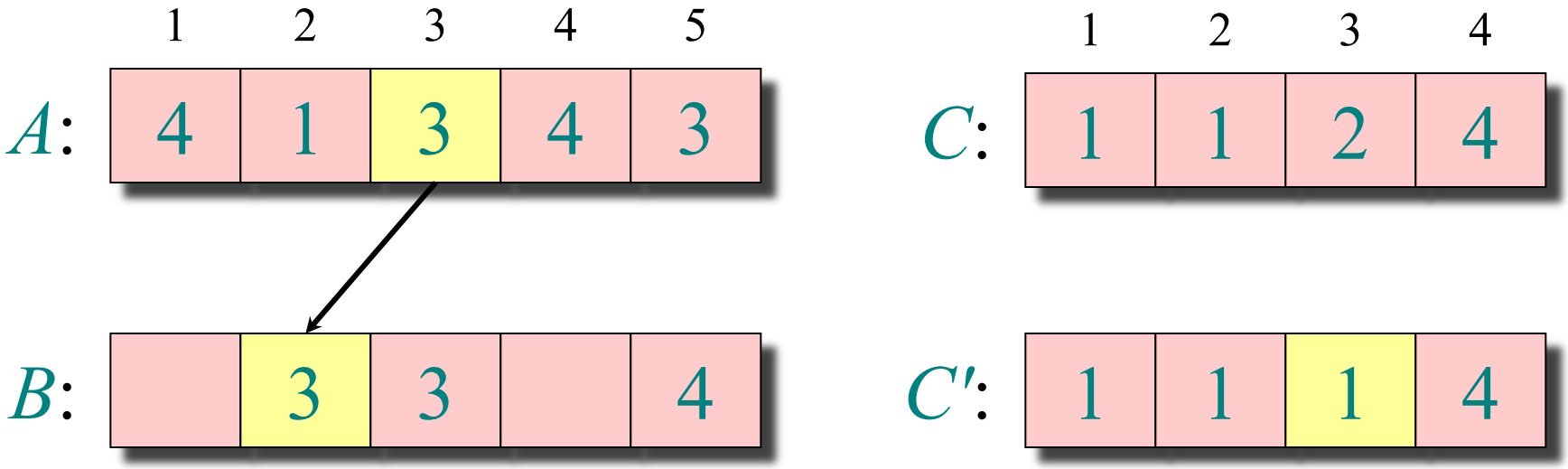
4.
```
for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
end
```

Schleife 4: Ordne neu



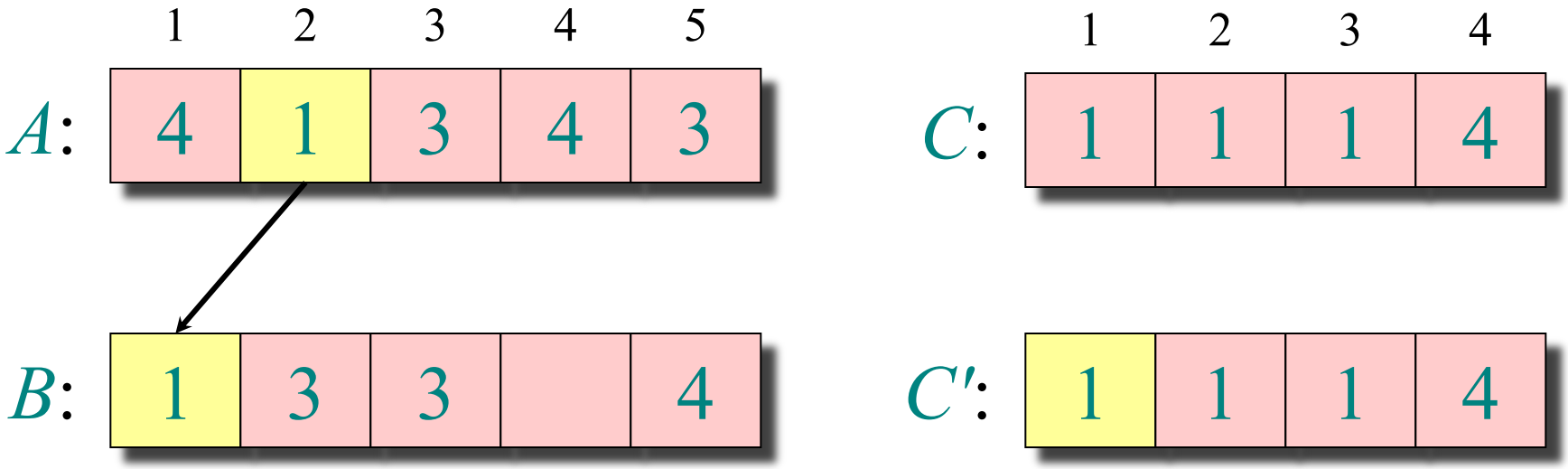
4.
```
for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
end
```

Schleife 4: Ordne neu



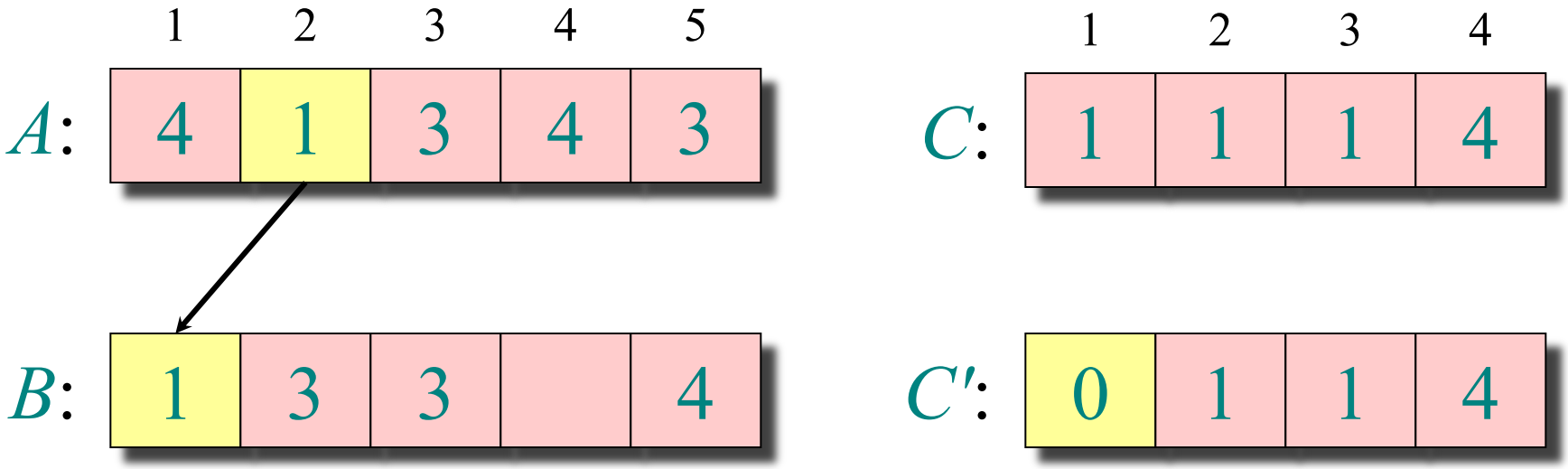

```
4. for j = n:-1:1
     B[C[A[j]]] = A[j]
     C[A[j]] = C[A[j]] - 1
   end
```

Schleife 4: Ordne neu



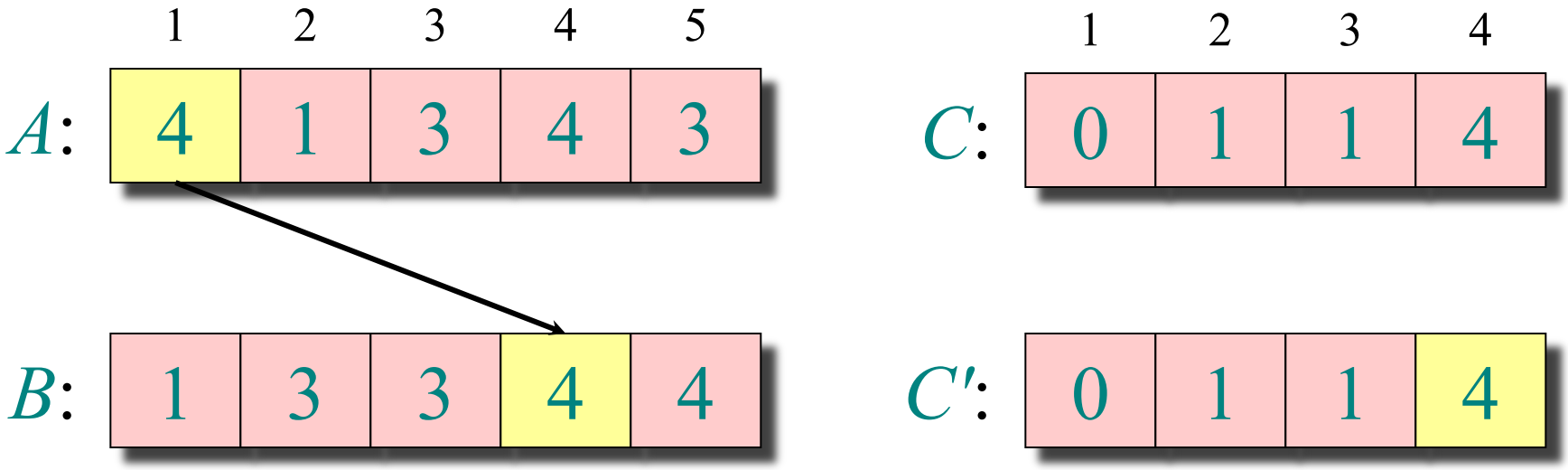
4.
```
for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
end
```

Schleife 4: Ordne neu



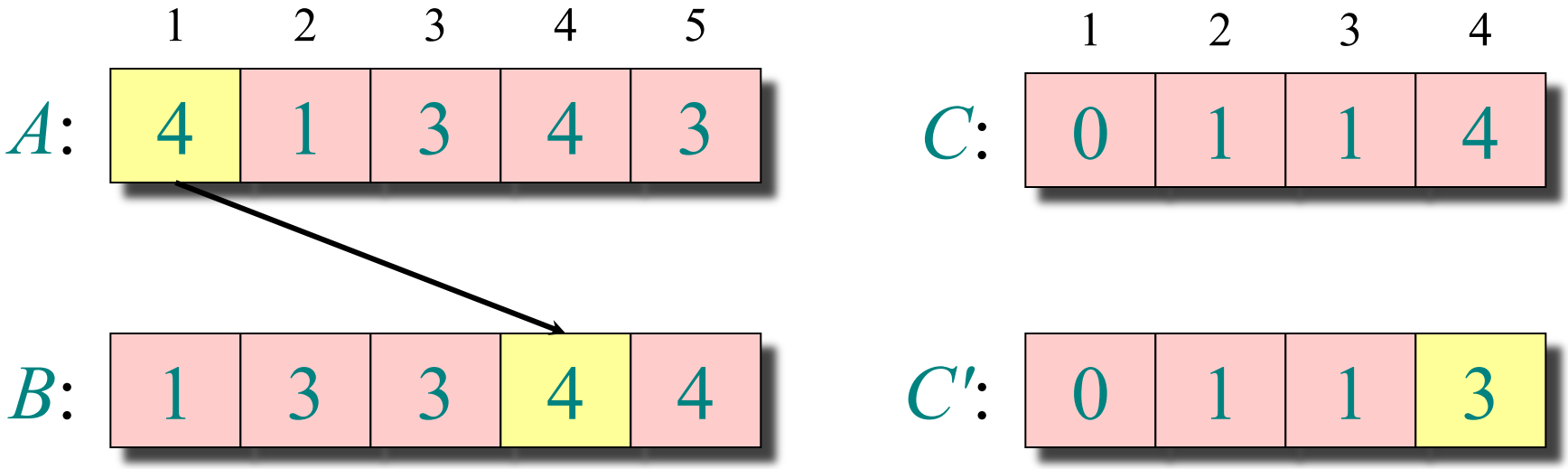
4.
```
for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
end
```

Schleife 4: Ordne neu



4.
```
for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
end
```

Schleife 4: Ordne neu



4.
```
for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
end
```

Counting-Sort Algorithmus

```
function counting_sort(A)
  k = maximum(A); n = length(A)
  C = fill(0, k)
  for i = 1:k
    insert!(C, i, 0)
  end
  for j = 1:n
    C[A[j]] = C[A[j]] + 1
  end
  # C[i] enthält nun die Anzahl der Elemente, die gleich i sind
  for i = 2:k
    C[i] = C[i] + C[i - 1]
  end
  # C[i] enthält nun die Anzahl der Elemente, die kleiner oder gleich i sind
  B = fill(0, n)
  for j = n:-1:1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
  end
  return B
end
```

Analyse

$\Theta(k)$

```
1. for i = 1:k
     C[i] = 0
   end
```

$\Theta(n)$

```
2. for j = 1:n
     C[A[j]] = C[A[j]] + 1
   end
```

$\Theta(k)$

```
3. for i = 2:k
     C[i] = C[i] + C[i - 1]
   end
```

$\Theta(n)$

```
4. for j = n:-1:1
     B[C[A[j]]] = A[j]
     C[A[j]] = C[A[j]] - 1
   end
```

$\Theta(n+k)$

Laufzeit: Wodurch wird sie reduziert?

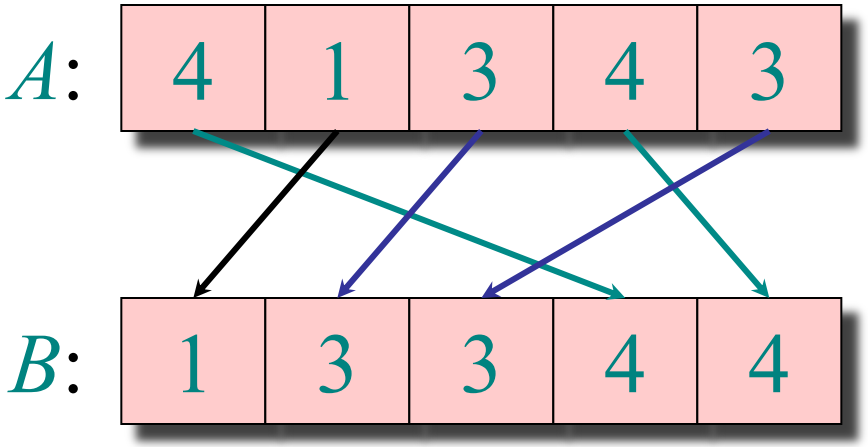
- Falls $k = O(n)$, dann braucht Counting-Sort $\Theta(n)$ viele Schritte.
- Aber theoretisch braucht doch Sortierung $\Omega(n \log n)$ viele Schritte!
- Gibt es ein Problem mit der Theorie?

Antwort:

- ***Sortieren durch Vergleichen*** liegt in $\Omega(n \log n)$
- Counting-Sort macht keine Vergleiche
- Counting-Sort verteilt einfach

Stabiles Sortieren

Counting-Sort ist stabil: die Eingabeordnung für gleiche Schlüssel bleibt bestehen

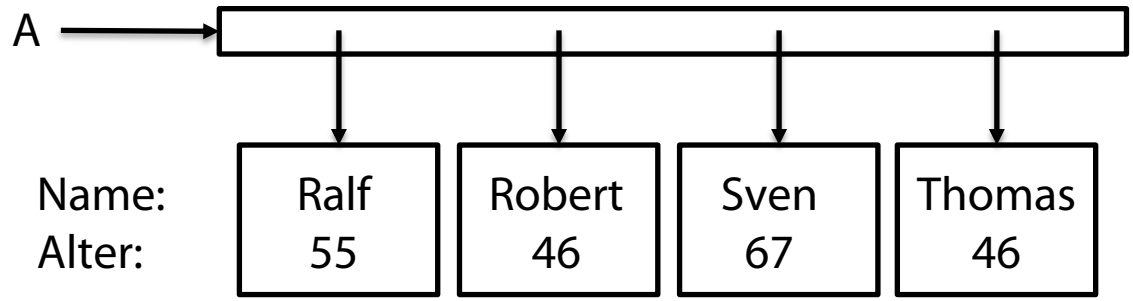


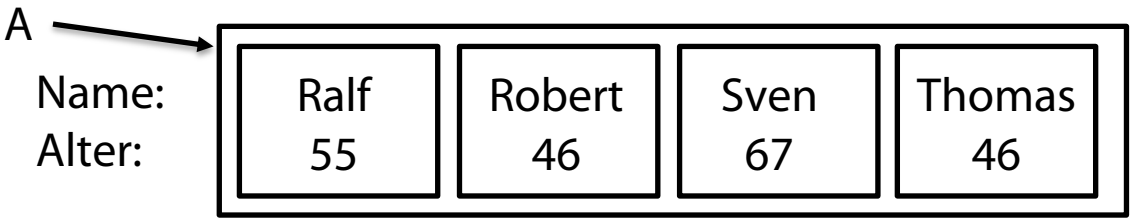
Warum ist das wichtig?

Welche andere Algorithmen haben diese Eigenschaft?

Sortieren nach mehreren Kriterien

- Zusammengesetze Objekte in einem Feld





Nur in C++ und
vergleichbaren Sprachen

Name		Address		
Last	First	Street	City	State
Bayless	Andrew	West Ave	Houston	TX
Benitez	Michael	North Ave	Los Angeles	CA
Chu	Henry	East Ave	San Diego	CA
Dangelo	David	Third St	Detroit	MI
Dawood	Hussam	Lincoln Rd	Detroit	MI
Devineni	Soujanya	Northwestern Ave	Houston	TX
Dunne	Brendan	EAST AVE.	Dallas	TX
Edwards	Brian	De Zavala Rd	San Antonio	TX
Godfrey	Daryl	MAIN ST	Austin	TX
Guerra	John	DALLAS AVE.	Austin	TX
Guevara	Clovis	University Pkwy	San Antonio	TX
Halbeisen	Gerald	FOREST CIRCLE	Los Angeles	CA
Hawkins	Richard	RIVERSIDE ST	San Antonio	TX
Hernandez	Monica	COLLEGE PKWY	San Antonio	TX
Hohmann	Shawn	COLLEGE PKWY	Cleveland	OH
Honeycutt	Richard	Southwest Ave	San Antonio	TX
Martinez	Juan	OAK CLIFF	Pheonix	AZ
Mayo	Nathan	UTSA BLVD	San Antonio	TX
Mirabal	Renato	FIRST ST	Columbus	OH
Modebe	Nnaemeka	RIVERSIDE ST	Houston	TX
Moon	Ryan	EAST AVE.	Madison	WI

Aufgabe: Sortiere Studierende nach (state, city, street).

Name		Address		
Last	First	Street	City	State
Martinez	Juan	OAK CLIFF	Pheonix	AZ
Benitez	Michael	North Ave	Los Angeles	CA
Chu	Henry	East Ave	San Diego	CA
Halbeisen	Gerald	FOREST CIRCLE	Los Angeles	CA
Dangelo	David	Third St	Detroit	MI
Dawood	Hussam	Lincoln Rd	Detroit	MI
Hohmann	Shawn	COLLEGE PKWY	Cleveland	OH
Mirabal	Renato	FIRST ST	Columbus	OH
Bayless	Andrew	West Ave	Houston	TX
Devineni	Soujanya	Northwestern Ave	Houston	TX
Dunne	Brendan	EAST AVE.	Dallas	TX
Edwards	Brian	De Zavala Rd	San Antonio	TX
Godfrey	Daryl	MAIN ST	Austin	TX
Guerra	John	DALLAS AVE.	Austin	TX
Guevara	Clovis	University Pkwy	San Antonio	TX
Hawkins	Richard	RIVERSIDE ST	San Antonio	TX
Hernandez	Monica	COLLEGE PKWY	San Antonio	TX
Honeycutt	Richard	Southwest Ave	San Antonio	TX
Mayo	Nathan	UTSA BLVD	San Antonio	TX
Modebe	Nnaemeka	RIVERSIDE ST	Houston	TX
Moon	Ryan	EAST AVE.	Madison	WI

Aufgabe: Sortiere Studierende nach (state, city, street).

Name		Address		
Last	First	Street	City	State
Godfrey	Daryl	MAIN ST	Austin	TX
Guerra	John	DALLAS AVE.	Austin	TX
Hohmann	Shawn	COLLEGE PKWY	Cleveland	OH
Mirabal	Renato	FIRST ST	Columbus	OH
Dunne	Brendan	EAST AVE.	Dallas	TX
Dangelo	David	Third St	Detroit	MI
Dawood	Hussam	Lincoln Rd	Detroit	MI
Bayless	Andrew	West Ave	Houston	TX
Devineni	Soujanya	Northwestern Ave	Houston	TX
Modebe	Nnaemeka	RIVERSIDE ST	Houston	TX
Benitez	Michael	North Ave	Los Angeles	CA
Halbeisen	Gerald	FOREST CIRCLE	Los Angeles	CA
Moon	Ryan	EAST AVE.	Madison	WI
Martinez	Juan	OAK CLIFF	Pheonix	AZ
Edwards	Brian	De Zavala Rd	San Antonio	TX
Guevara	Clovis	University Pkwy	San Antonio	TX
Hawkins	Richard	RIVERSIDE ST	San Antonio	TX
Hernandez	Monica	COLLEGE PKWY	San Antonio	TX
Honeycutt	Richard	Southwest Ave	San Antonio	TX
Mayo	Nathan	UTSA BLVD	San Antonio	TX
Chu	Henry	East Ave	San Diego	CA

Aufgabe: Sortiere Studierende nach (state, city, street).

Name		Address		
Last	First	Street	City	State
Hohmann	Shawn	COLLEGE PKWY	Cleveland	OH
Hernandez	Monica	COLLEGE PKWY	San Antonio	TX
Guerra	John	DALLAS AVE.	Austin	TX
Edwards	Brian	De Zavala Rd	San Antonio	TX
Chu	Henry	East Ave	San Diego	CA
Dunne	Brendan	EAST AVE.	Dallas	TX
Moon	Ryan	EAST AVE.	Madison	WI
Mirabal	Renato	FIRST ST	Columbus	OH
Halbeisen	Gerald	FOREST CIRCLE	Los Angeles	CA
Dawood	Hussam	Lincoln Rd	Detroit	MI
Godfrey	Daryl	MAIN ST	Austin	TX
Benitez	Michael	North Ave	Los Angeles	CA
Devineni	Soujanya	Northwestern Ave	Houston	TX
Martinez	Juan	OAK CLIFF	Pheonix	AZ
Modebe	Nnaemeka	RIVERSIDE ST	Houston	TX
Hawkins	Richard	RIVERSIDE ST	San Antonio	TX
Honeycutt	Richard	Southwest Ave	San Antonio	TX
Dangelo	David	Third St	Detroit	MI
Guevara	Clovis	University Pkwy	San Antonio	TX
Mayo	Nathan	UTSA BLVD	San Antonio	TX
Bayless	Andrew	West Ave	Houston	TX

Aufgabe: Sortiere Studierende nach (state, city, street).

Name		Address		
Last	First	Street	City	State
Bayless	Andrew	West Ave	Houston	TX
Benitez	Michael	North Ave	Los Angeles	CA
Chu	Henry	East Ave	San Diego	CA
Dangelo	David	Third St	Detroit	MI
Dawood	Hussam	Lincoln Rd	Detroit	MI
Devineni	Soujanya	Northwestern Ave	Houston	TX
Dunne	Brendan	EAST AVE.	Dallas	TX
Edwards	Brian	De Zavala Rd	San Antonio	TX
Godfrey	Daryl	MAIN ST	Austin	TX
Guerra	John	DALLAS AVE.	Austin	TX
Guevara	Clovis	University Pkwy	San Antonio	TX
Halbeisen	Gerald	FOREST CIRCLE	Los Angeles	CA
Hawkins	Richard	RIVERSIDE ST	San Antonio	TX
Hernandez	Monica	COLLEGE PKWY	San Antonio	TX
Hohmann	Shawn	COLLEGE PKWY	Cleveland	OH
Honeycutt	Richard	Southwest Ave	San Antonio	TX
Martinez	Juan	OAK CLIFF	Pheonix	AZ
Mayo	Nathan	UTSA BLVD	San Antonio	TX
Mirabal	Renato	FIRST ST	Columbus	OH
Modebe	Nnaemeka	RIVERSIDE ST	Houston	TX
Moon	Ryan	EAST AVE.	Madison	WI

Aufgabe: Sortiere Studierende nach (state, city, street).

Name		Address		
Last	First	Street	City	State
Hernandez	Monica	COLLEGE PKWY	San Antonio	TX
Hohmann	Shawn	COLLEGE PKWY	Cleveland	OH
Guerra	John	DALLAS AVE.	Austin	TX
Edwards	Brian	De Zavala Rd	San Antonio	TX
Chu	Henry	East Ave	San Diego	CA
Dunne	Brendan	EAST AVE.	Dallas	TX
Moon	Ryan	EAST AVE.	Madison	WI
Mirabal	Renato	FIRST ST	Columbus	OH
Halbeisen	Gerald	FOREST CIRCLE	Los Angeles	CA
Dawood	Hussam	Lincoln Rd	Detroit	MI
Godfrey	Daryl	MAIN ST	Austin	TX
Benitez	Michael	North Ave	Los Angeles	CA
Devineni	Soujanya	Northwestern Ave	Houston	TX
Martinez	Juan	OAK CLIFF	Pheonix	AZ
Hawkins	Richard	RIVERSIDE ST	San Antonio	TX
Modebe	Nnaemeka	RIVERSIDE ST	Houston	TX
Honeycutt	Richard	Southwest Ave	San Antonio	TX
Dangelo	David	Third St	Detroit	MI
Guevara	Clovis	University Pkwy	San Antonio	TX
Mayo	Nathan	UTSA BLVD	San Antonio	TX
Bayless	Andrew	West Ave	Houston	TX

Aufgabe: Sortiere Studierende nach (state, city, street).

Name		Address		
Last	First	Street	City	State
Guerra	John	DALLAS AVE.	Austin	TX
Godfrey	Daryl	MAIN ST	Austin	TX
Hohmann	Shawn	COLLEGE PKWY	Cleveland	OH
Mirabal	Renato	FIRST ST	Columbus	OH
Dunne	Brendan	EAST AVE.	Dallas	TX
Dawood	Hussam	Lincoln Rd	Detroit	MI
Dangelo	David	Third St	Detroit	MI
Devineni	Soujanya	Northwestern Ave	Houston	TX
Modebe	Nnaemeka	RIVERSIDE ST	Houston	TX
Bayless	Andrew	West Ave	Houston	TX
Halbeisen	Gerald	FOREST CIRCLE	Los Angeles	CA
Benitez	Michael	North Ave	Los Angeles	CA
Moon	Ryan	EAST AVE.	Madison	WI
Martinez	Juan	OAK CLIFF	Pheonix	AZ
Hernandez	Monica	COLLEGE PKWY	San Antonio	TX
Edwards	Brian	De Zavala Rd	San Antonio	TX
Hawkins	Richard	RIVERSIDE ST	San Antonio	TX
Honeycutt	Richard	Southwest Ave	San Antonio	TX
Guevara	Clovis	University Pkwy	San Antonio	TX
Mayo	Nathan	UTSA BLVD	San Antonio	TX
Chu	Henry	East Ave	San Diego	CA

Aufgabe: Sortiere Studierende nach (state, city, street).

Name		Address		
Last	First	Street	City	State
Martinez	Juan	OAK CLIFF	Pheonix	AZ
Halbeisen	Gerald	FOREST CIRCLE	Los Angeles	CA
Benitez	Michael	North Ave	Los Angeles	CA
Chu	Henry	East Ave	San Diego	CA
Dawood	Hussam	Lincoln Rd	Detroit	MI
Dangelo	David	Third St	Detroit	MI
Hohmann	Shawn	COLLEGE PKWY	Cleveland	OH
Mirabal	Renato	FIRST ST	Columbus	OH
Guerra	John	DALLAS AVE.	Austin	TX
Godfrey	Daryl	MAIN ST	Austin	TX
Dunne	Brendan	EAST AVE.	Dallas	TX
Devineni	Soujanya	Northwestern Ave	Houston	TX
Modebe	Nnaemeka	RIVERSIDE ST	Houston	TX
Bayless	Andrew	West Ave	Houston	TX
Hernandez	Monica	COLLEGE PKWY	San Antonio	TX
Edwards	Brian	De Zavala Rd	San Antonio	TX
Hawkins	Richard	RIVERSIDE ST	San Antonio	TX
Honeycutt	Richard	Southwest Ave	San Antonio	TX
Guevara	Clovis	University Pkwy	San Antonio	TX
Mayo	Nathan	UTSA BLVD	San Antonio	TX
Moon	Ryan	EAST AVE.	Madison	WI

Aufgabe: Sortiere Studierende nach (state, city, street).

Name		Address		
Last	First	Street	City	State
Martinez	Juan	OAK CLIFF	Pheonix	AZ
Halbeisen	Gerald	FOREST CIRCLE	Los Angeles	CA
Benitez	Michael	North Ave	Los Angeles	CA
Chu	Henry	East Ave	San Diego	CA
Dawood	Hussam	Lincoln Rd	Detroit	MI
Dangelo	David	Third St	Detroit	MI
Hohmann	Shawn	COLLEGE PKWY	Cleveland	OH
Mirabal	Renato	FIRST ST	Columbus	OH
Guerra	John	DALLAS AVE.	Austin	TX
Godfrey	Daryl	MAIN ST	Austin	TX
Dunne	Brendan	EAST AVE.	Dallas	TX
Devineni	Soujanya	Northwestern Ave	Houston	TX
Modebe	Nnaemeka	RIVERSIDE ST	Houston	TX
Bayless	Andrew	West Ave	Houston	TX
Hernandez	Monica	COLLEGE PKWY	San Antonio	TX
Edwards	Brian	De Zavala Rd	San Antonio	TX
Hawkins	Richard	RIVERSIDE ST	San Antonio	TX
Honeycutt	Richard	Southwest Ave	San Antonio	TX
Guevara	Clovis	University Pkwy	San Antonio	TX
Mayo	Nathan	UTSA BLVD	San Antonio	TX
Moon	Ryan	EAST AVE.	Madison	WI

Aufgabe: Sortiere Studierende nach (state, city, street).

Stabiles Sortieren

- Die meisten $O(n^2)$-Sortieralgorithmen sind stabil
  - oder können "einfach" stabil gemacht werden
- O(n log n)-Sortieralgorithmen sind nicht immer stabil
  - Ist Heap-Sort stabil?
- Generischer Ansatz, Stabilität zu erzeugen:
  - Verwende zwei Schlüssel,
    der zweite ist der originale Index des Elements
  - Wenn zwei Elemente gleich, vergleiche zweite Schlüsselkomponenten

[5, 6, 5, 1, 2, 3, 2, 6]

[(5, 1), (6, 2), (5, 3), (1, 4), (2, 5), (3, 6), (2, 7), (6, 8)]

(5, 1) < (5, 3)

(2, 5) < (2, 7)

Wie kann man sehr große Zahlen sortieren?

19809910912351818359934019954038012811529538470010159453961469638240836020103925853861438650762868132893673814865209099036919798708408709665302029918566412442123451645478539207574785913188553099522359313739735426705749044361811176779529358191483755781558501764221221110674142522204403312937607718098797338329180836856504702326654684056982119770959427525245528076153239047050820305445639847201611168478334240651199238019

Jede Zeile sei eine "lange" Zahl, eine Gensequenz oder….

Zahlen dieser Art sind zu groß für den Integer-Datentyp, sie werden als "Zeichenkette" repräsentiert

Verwende vergleichsbasiertes Sortieren mit einer Zeichenkettenvergleichsfunktion

if A[i] < A[j] wird zu if vergleiche(A[i], A[j]) < 0 mit

```
function vergleiche(s, t)
  for i = 1:minimum([length(s), length(t)])
    if s[i] < t[i] return -1
    elseif s[i] > t[i] return 1
    end
  end
  if length(s) == length(t) return 0
  elseif length(s) < length(t) return -1
  else return 1
  end
end
```

Kosten des Vergleichs von zwei Zeichenketten der Länge d? O(d)
Gesamtkosten: O(d n log n)

Radix-Sort

- Ähnlich wie das Sortieren von Adressbüchern
- Behandle jede Zahl als Sortierschlüssel
- Starte vom Least-significant-Bit

Most significant ↓ Least significant ↓

19809910912351818359 9
34019954038012811529 5
38470010159453961469 6
38240836020103925853 8
61438650762868132893 6

Jede Zeile entspricht einer “langen” Zahl

Radix-Sort wurde schon **1887** in Arbeiten von Herman Hollerith zu Volkszählungsmaschinen verwendet

Radix-Sort: Illustration

- Hier ein vereinfachtes Beispiel:

Jede Zeile entspricht einer langen Zahl

7 4 2
7 4 8
0 5 4
6 8 8
4 1 2
2 3 0
9 3 5
1 1 6
1 6 1
4 3 4
3 8 5
6 6 6
0 3 1
0 1 3
3 6 5
1 7 3
0 1 6

Radix-Sort: Illustration

- Sortiere nach letzer Zahl:

2 3 0
1 6 1
0 3 1
7 4 2
4 1 2
0 1 3
1 7 3
0 5 4
4 3 4
9 3 5
3 8 5
3 6 5
1 1 6
6 6 6
0 1 6
7 4 8
6 8 8

Radix-Sort: Illustration

- Sortiere nach zweitletzer Zahl:

4 1 2
0 1 3
1 1 6
0 1 6
2 3 0
0 3 1
4 3 4
9 3 5
7 4 2
7 4 8
0 5 4
1 6 1
3 6 5
6 6 6
1 7 3
3 8 5
6 8 8

Radix-Sort: Illustration

- Sortiere nach erster Zahl:

0 1 3
0 1 6
0 3 1
0 5 4
1 1 6
1 6 1
1 7 3
2 3 0
3 6 5
3 8 5
4 1 2
4 3 4
6 6 6
6 8 8
7 4 2
7 4 8
9 3 5

Zeitkomplexität

- Sortierung jeder der d Spalten mit Counting-Sort
- Gesamtkosten: $d\ (n + k)$
  - Wähle $k = 10$ konstant
  - Gesamtkosten: $\Theta(dn)$
- Partitionierung der d Zahlen in z.B. in Dreiergruppen
  - Gesamtkosten: $(n+10^3)d/3$
- Wir arbeiten mit Binärzahlen anstelle von Dezimalen
  - Partitionierung der d Bits in Gruppen von r Bits
  - Gesamtkosten: $(n+2^r)d/r$
  - Wähle $r = log\ n$
  - Gesamtkosten: O($dn\ /\ log\ n$)
  - Vergleiche mit O($dn\ log\ n$) für das einfache Verfahren
- Aber: Radix-Sort hat hohen konstanten Faktor

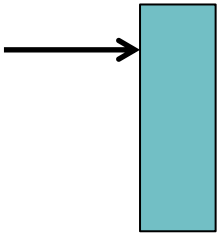
Platzkomplexität

- Verwendung von Counting-Sort
- Daher zusätzlicher Speicher nötig: $\theta(n)$

Strukturen zur Gruppierung von Daten

auch vertikale Darstellung möglich

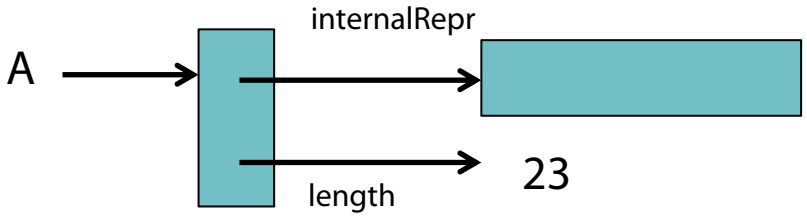
- Arrays
  - Zugriff über Index (wir schreiben A[i] oder auch A[i] = …)
  - Funktion length ist definiert
  - Zeichenketten als spezielle Arrays (Notation "…")
  - Funktion A: I → D Notation: [3, 42, 55, 6]
- Tupel (Reihung von Komponenten)
  - Beispiel: ("Ralf", 55, 1.8) n-Tupel
  - (p, age, height) = ("Ralf", 55, 1.8)
  - Zugriff über Indexschreibweise (42, 23)[1] = 42
    - Namen von Funktionen, die auf Komponenten zugreifen, können wie immer vereinbart werden
    - Anzahl der Komponenten üblicherweise klein



auch horizontale Darstellung möglich

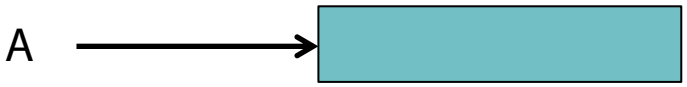
Wenn wir length effizient realisieren wollen…

- … müssten wir uns Arrays so vorstellen




```
julia> foo = [4,2,1];

julia> insert!(foo, 2, 3);

julia> foo
4-element Array{Int64,1}:
 4
 3
 2
 1
```

- A[i] muss der Compiler entsprechend umsetzen
- Wir bleiben aber in der Darstellung bei



auch vertikale Darstellung möglich

Listen als abstrakte Datentypen (ADTs)

Operationen:

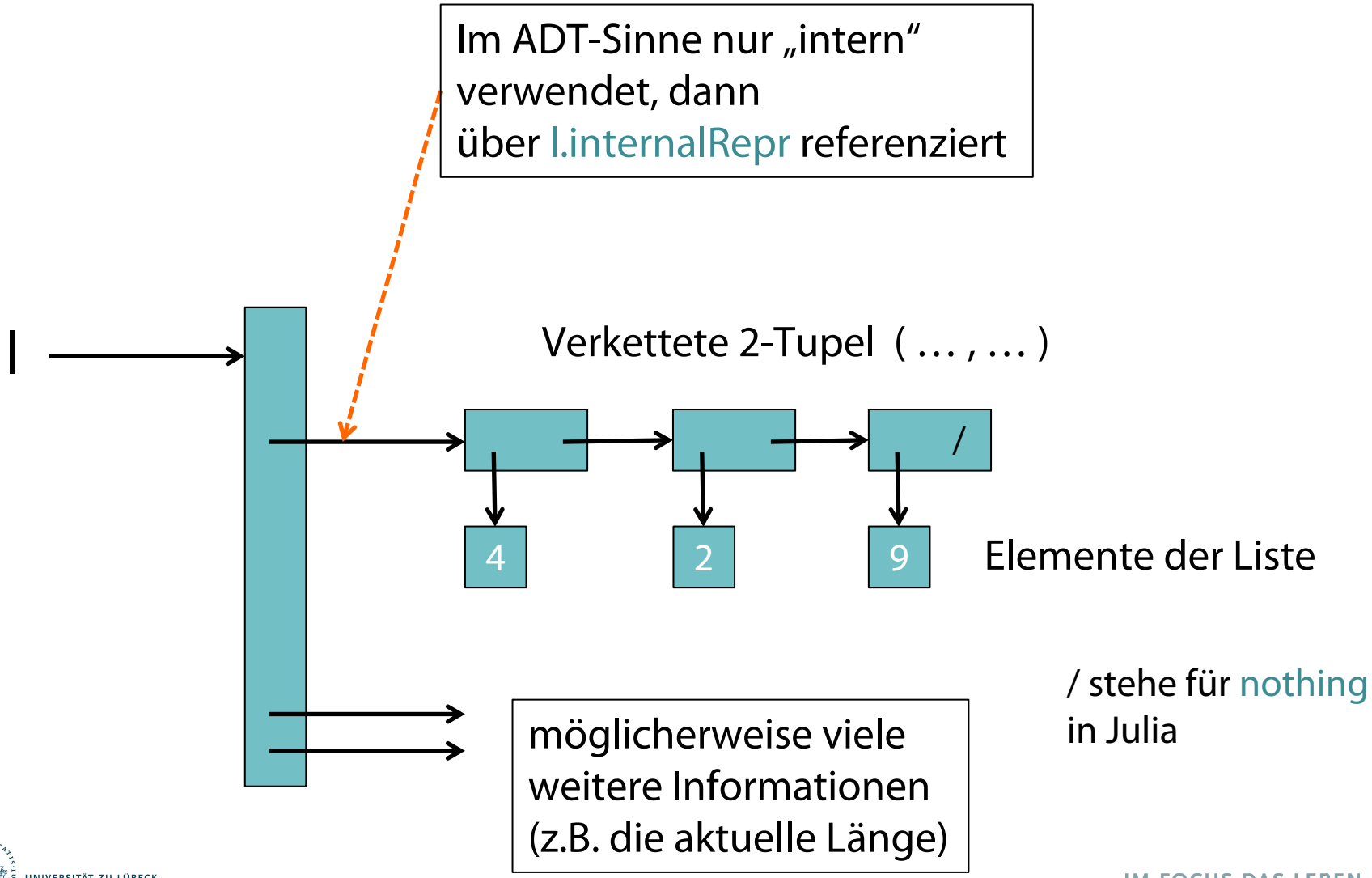
- function makeList() liefert neue Liste (am Anfang leer)
- procedure insert(e, l) fügt Element e am Anfang in Liste l ein, verändert l
- procedure delete(e, l) löscht Element e sofern enthalten, verändert l, wenn ein Element gelöscht wird
- function first(l) gibt Last-in-Element zurück (Fehler, wenn l leer)
- procedure deleteFirst(l) löscht Last-in-Element in l (Fehler, wenn l leer)
- function length(l) gibt Anzahl der Elemente in l zurück
- function mtList?(l) gibt true zurück, wenn l leer ist, sonst false

Iteration (last-in first-out):

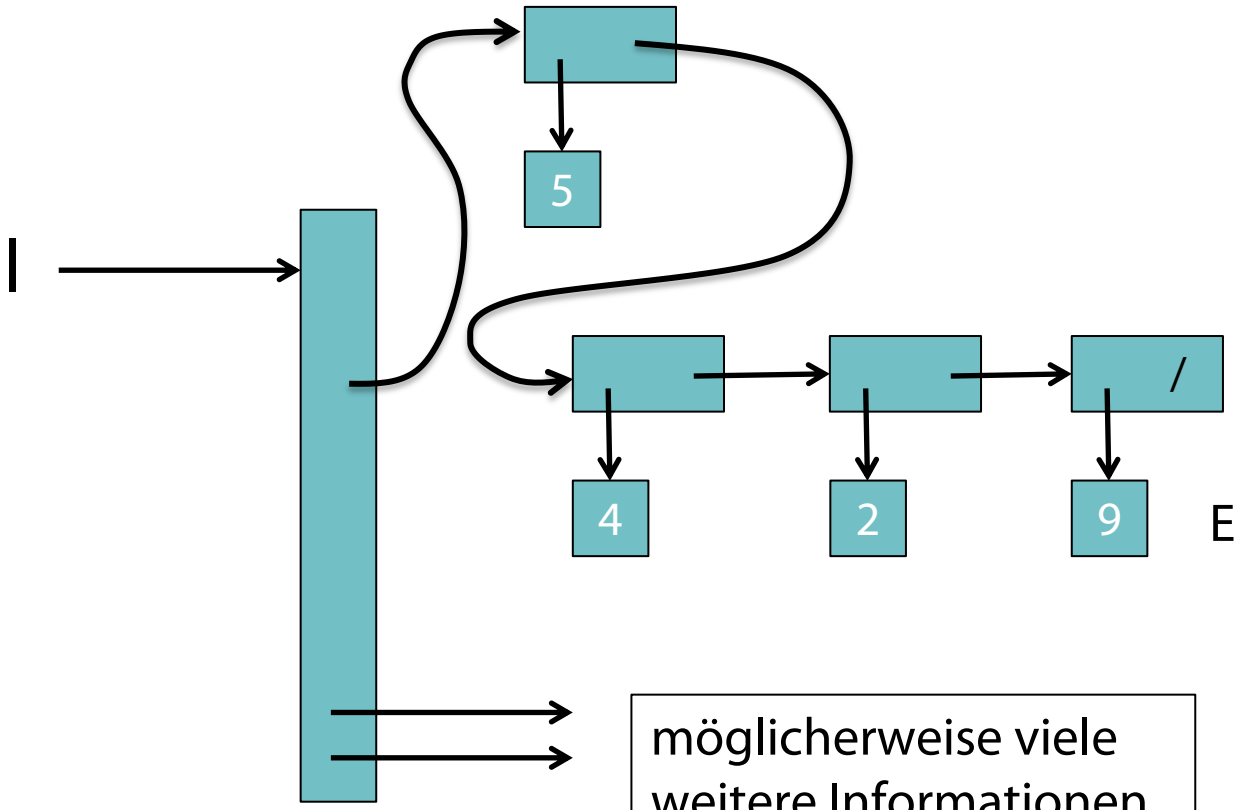
- for e in l ... end *oder auch* for e ∈ l ... end

Beispielimplementierung in Julia vorhanden.

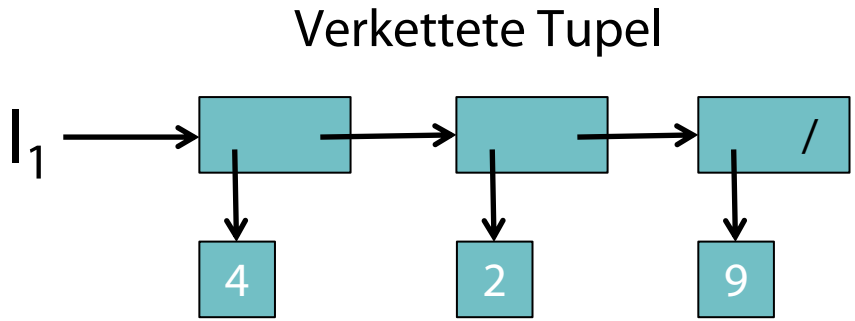
Listen intern realisiert als Tupel (mutable struct)



insert(5, l)


l
5
4
2
9
/
Elemente der Liste
möglicherweise viele
weitere Informationen
(z.B. die aktuelle Länge)

Listen als Glaskästen (“verkettete Liste”)



- Ausdruck (e, l) liefert Tupel mit Element e und Liste l
- Beispiele:
  - (4, (2, (9, nothing)))
- Sei l_1 = (4, (2, (9, nothing))), dann Zugriff mit (e, l_2) = l_1 dann gilt: e = 4 und l_2 = (2, (9, nothing))
- Zugriff auch über first(l) und rest(l) im API

Listen als Glaskästen

Notation (Beispiel): (1, (2, nothing)) mit nothing für die leere Liste

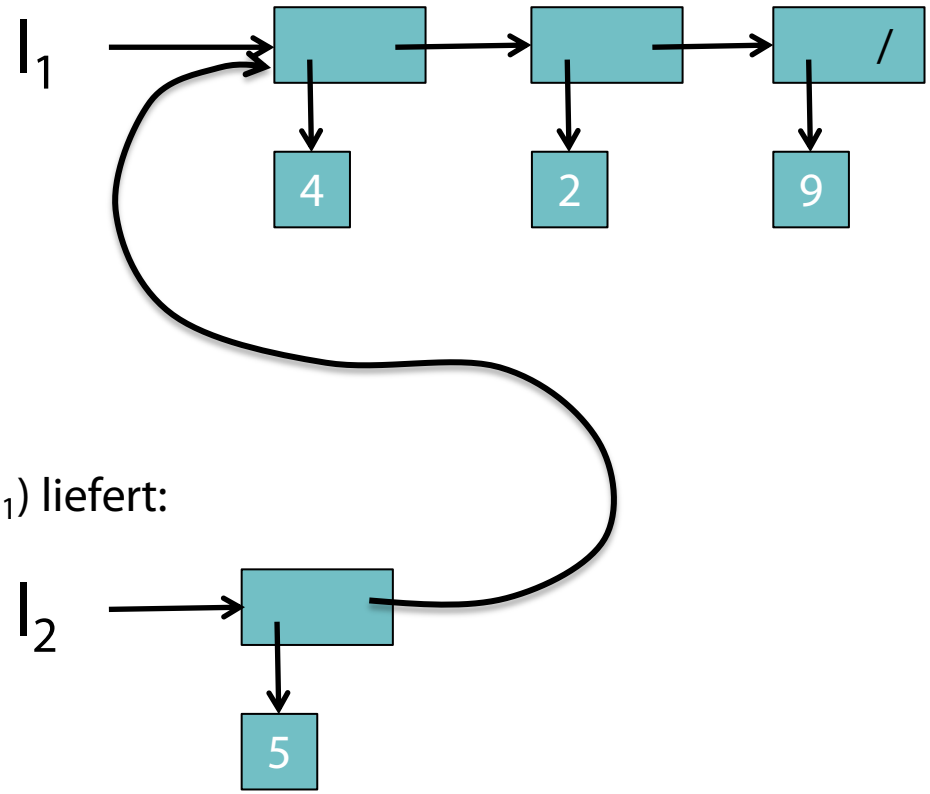
Operationen:

- function cons(e, l) fügt Element e am Anfang in Liste l ein,
  verändert l nicht, gibt eine neue, erweiterte Liste zurück
- function first(l) gibt die erste Komponente l[1] des Tupels zurück
  (Fehler, wenn l leer), Manipulation mit l[1] = ...
- function rest(l) gibt die zweite Komponente l[2] des Tupels zurück
  (Fehler, wenn l leer), Manipulation mit l[2] = ...
- function length(l) gibt Anzahl der Elemente in l zurück
- function mt?(l) gibt true zurück, wenn l = nothing ist, sonst false

Iteration:

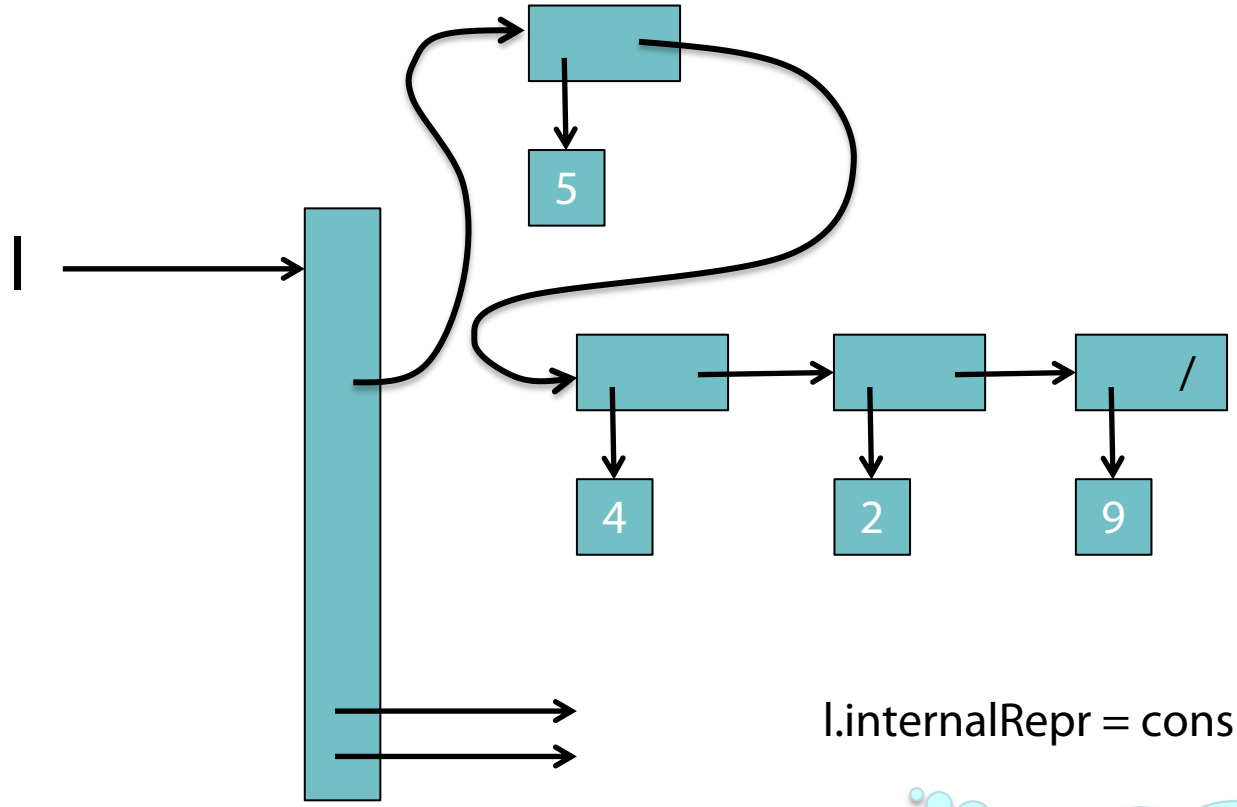
- for e in l ... end *oder auch* for e ∈ l ... end

Cons



$l_2 = cons(5, l_1)$

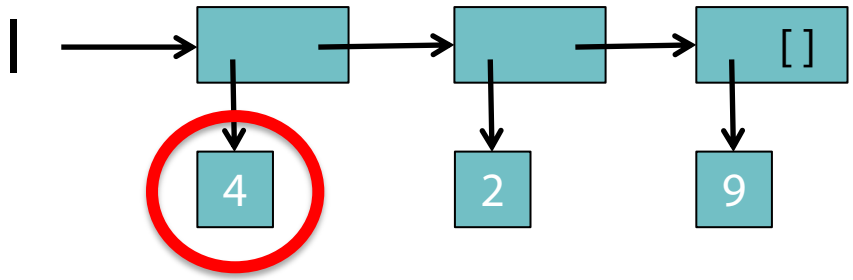
Wiederholung ADT-Listen: insert(5, l)



l.internalRepr = cons(5, l.internalRepr)

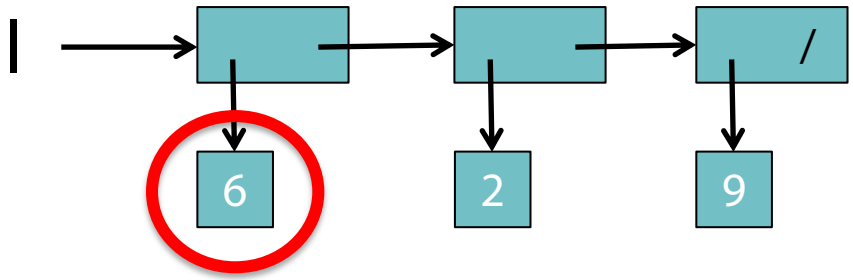
Aber von außen nicht sichtbar

Zugriff auf erste Komponente mit first



- first(l) oder l[1]

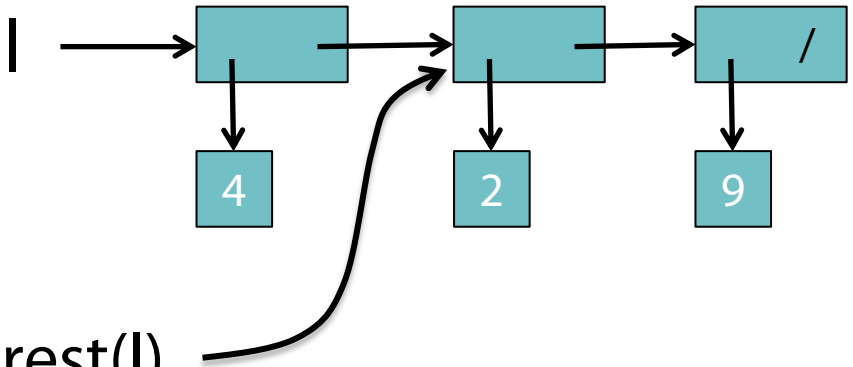
Manipulaton der ersten Komponente



- l[1] = 6

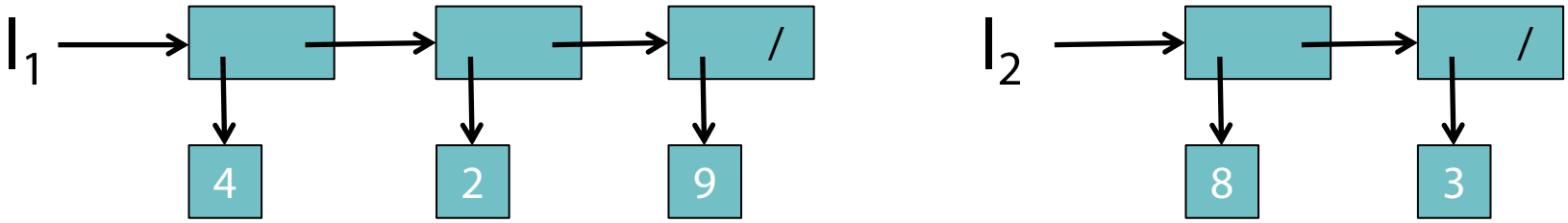
- Vergleiche das Setzen von Arrayelementen: A[i] = …

Zugriff auf zweite Komponente mit rest



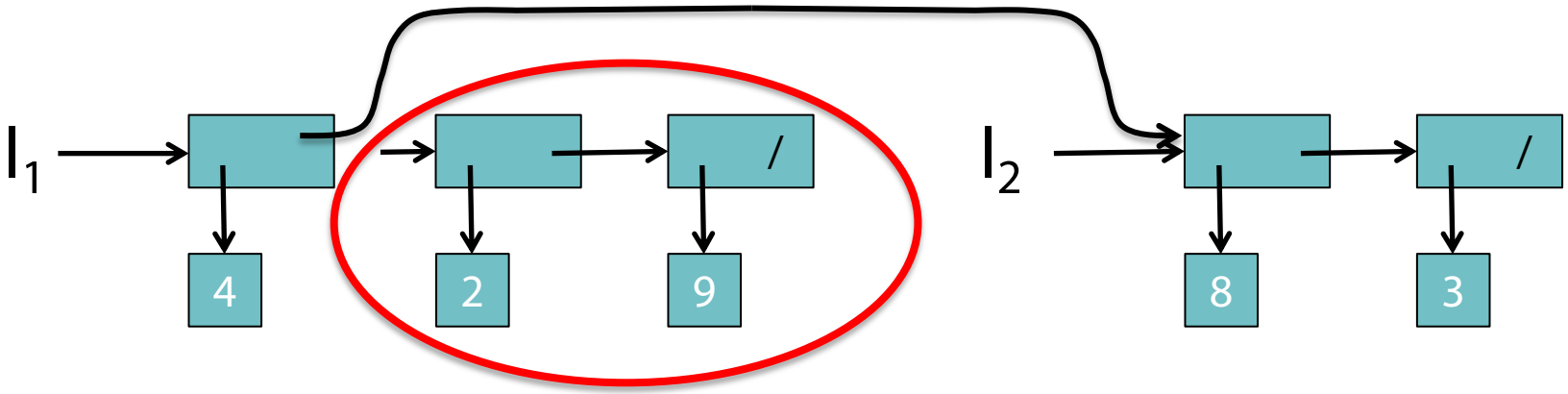
- rest(l)
- l[2]

Manipulaton der zweiten Komponente (1)



- Was bewirkt $l_1[2] = l_2$?

Manipulaton der zweiten Komponente (2)



- $l_1[2] = l_2$

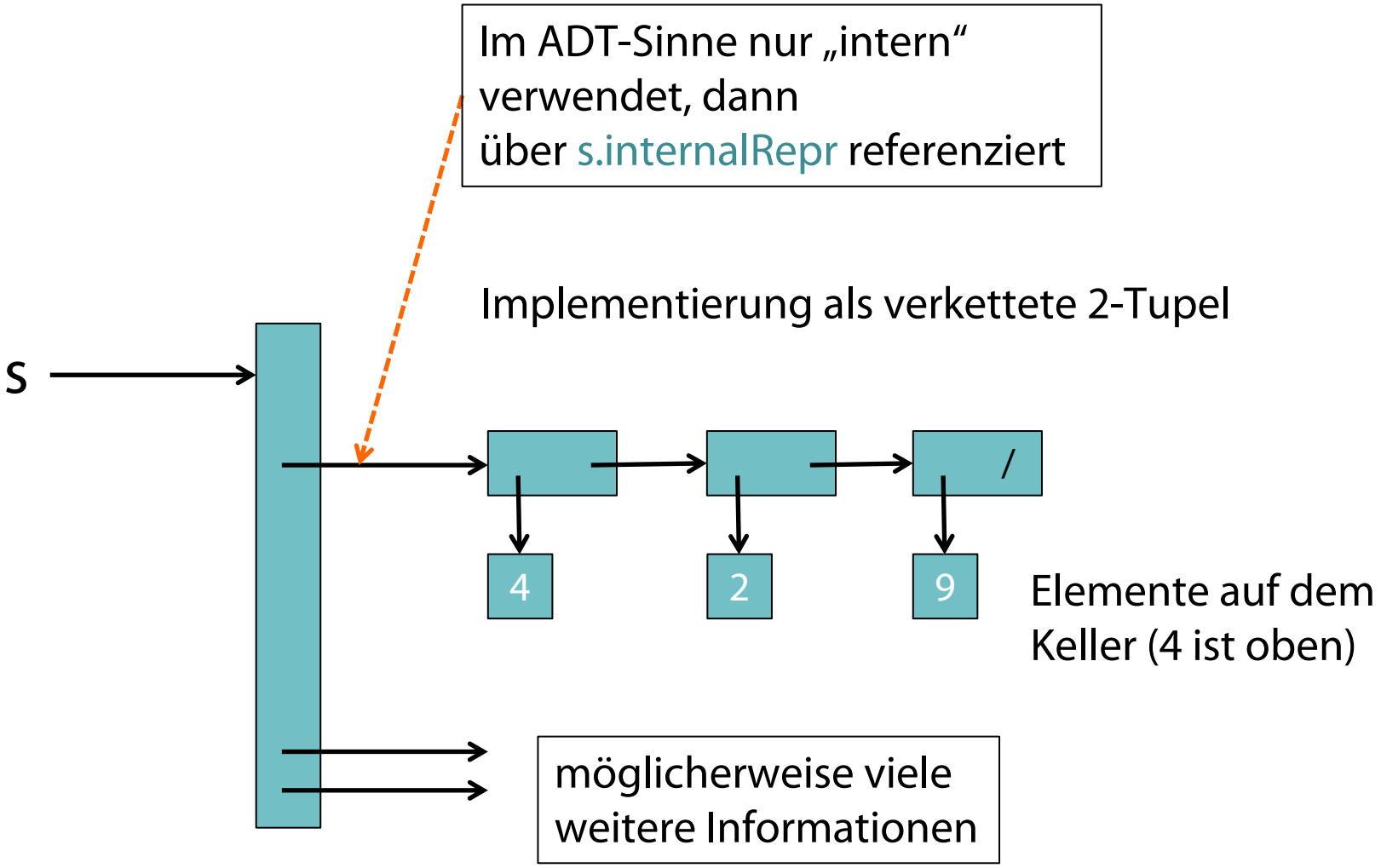
Kellerspeicher / Stapelspeicher / Stack

Operationen:

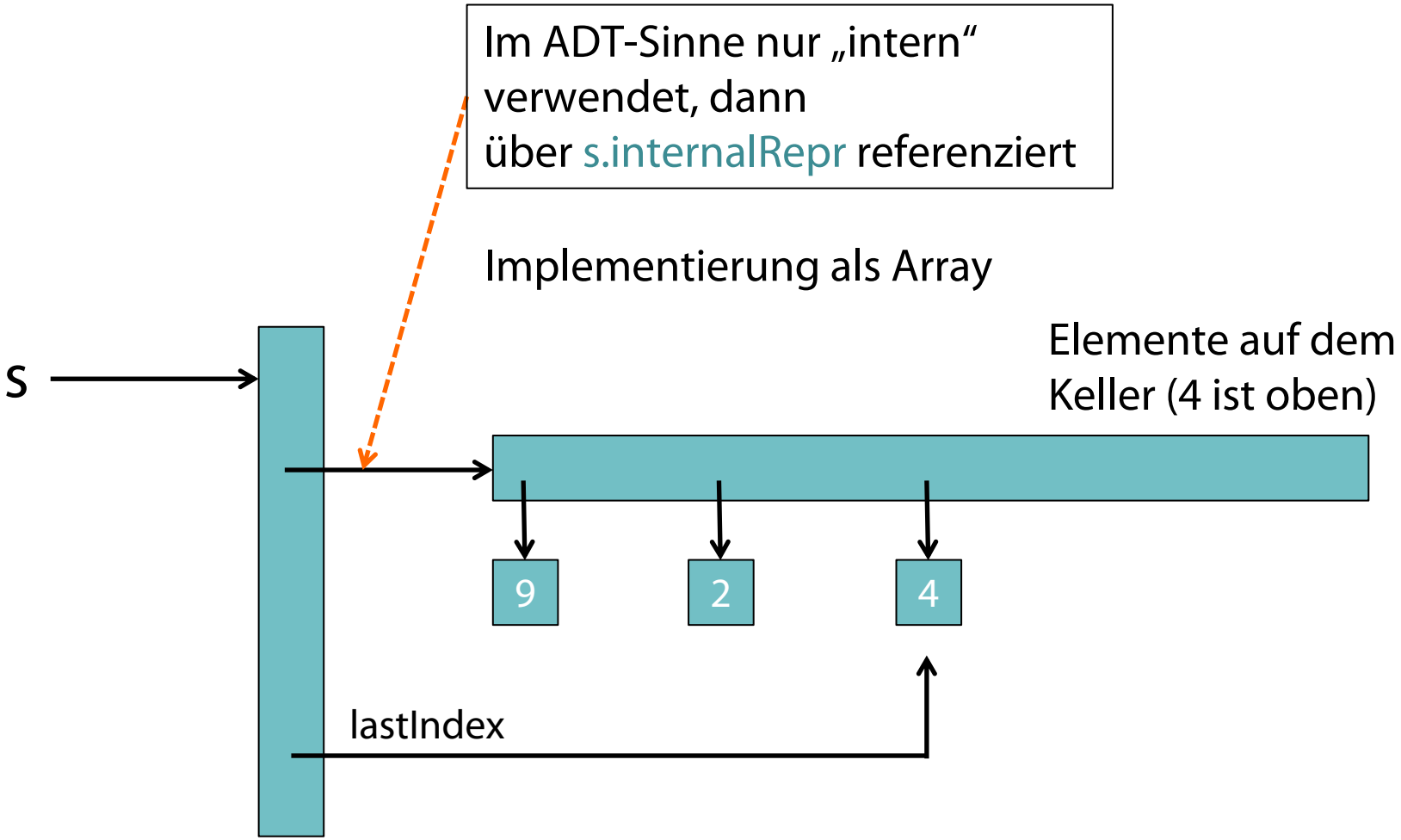
- function makeStack() liefert leeren Keller
- procedure push(e, s) fügt Element e oben in den Keller s ein, verändert s
- function top(s) gibt oberes Element zurück (Fehler, wenn s leer)
- procedure pop(s) löscht Top-Element in s (Fehler, wenn s leer)
- function mtStack?(s) gibt true zurück, wenn s leer ist, sonst false

Iteration: eigentlich nicht vorgesehen (aber im Prinzip realisierbar)

Keller intern (Beispiel: als Liste)



Keller intern (Beispiel 2: als Array)



Realisierung von Kellerspeichern

- Arrays
  - Initiale Größe muss vorher festgelegt werden
  - Keller kann “vollaufen”
  - Neues, größeres Array und Umkopieren
- Verkettete Liste
  - Speicherbedarf entspricht dem Füllgrad des Kellers

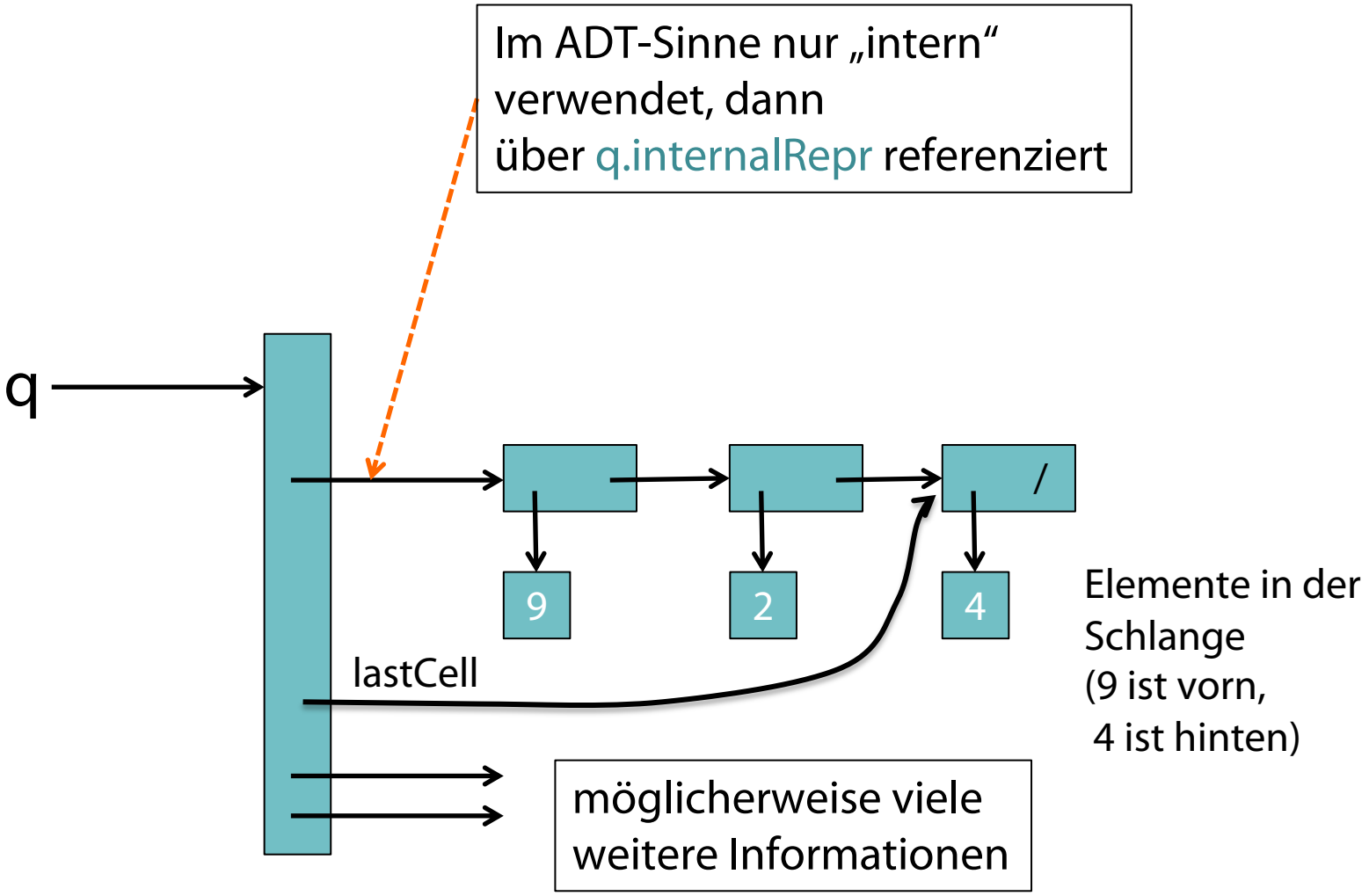
Schlange / Queue (First-in-First-out-Speicher)

Operationen:

- function makeQueue() liefert leere Warteschlange
- procedure enqueue(e, q) fügt Element e hinten in die Schlange q ein, verändert q
- function next(q) gibt vorderes Element zurück, verändert q nicht
- function dequeue (q) gibt vorderes Element zurück, verändert q
- function mtQueue?(q) gibt true zurück, wenn q leer ist, sonst false

Iteration: nicht vorgesehen (evtl. wie Liste)

Queue intern (Beispiel)



Sortieren durch verallgemeinerte Gruppierung

Bucket-Sort

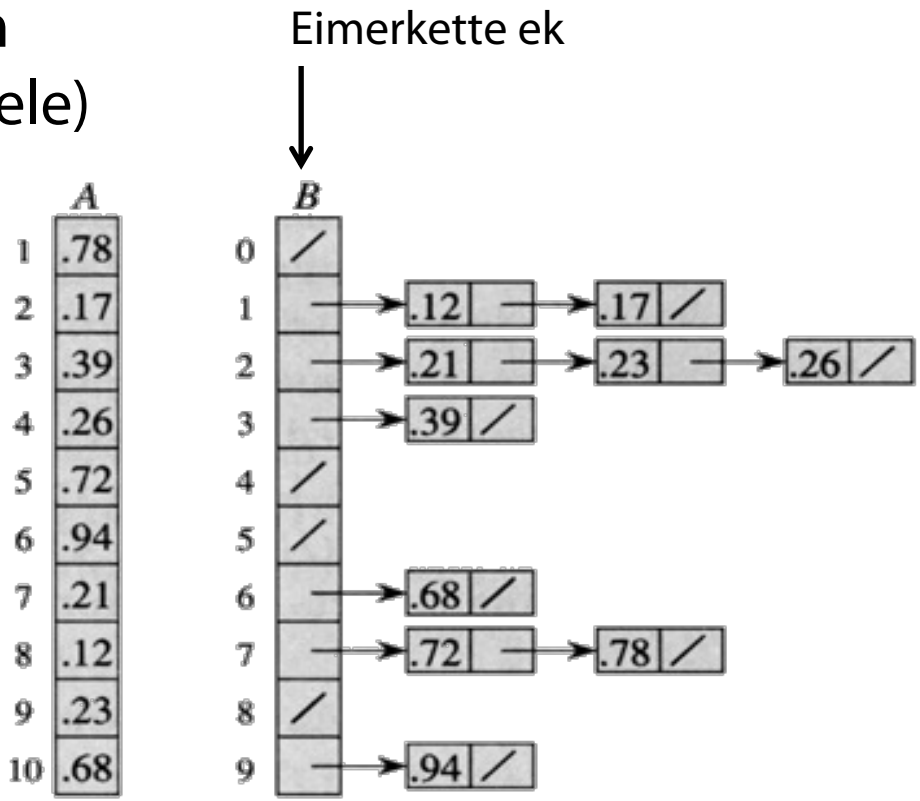
```
function bucket_sort(A, waehle_eimer)
    # Bestimme Anzahl der benötigten Eimer:
    k = waehle_eimer(maximum(A))
    # Eimerkette erstellen
    E = []
    for i = 1:k
        insert!(E, i, [])
    end
    # in die Eimer aufteilen
    for i = 1:length(A)
        push!(E[waehle_eimer(A[i])], A[i])
    end
    # Eimer sortieren
    for i = 1:k
        sort!(E[i])
    end
    # Eimer zusammenfuegen
    B = []
    for i = 1:k
        append!(B, E[i])
    end
    return B
end
```


```
A=[4,2,1,3,2,1,3,2,1,2,3,2,1,4]
bucket_sort(A,identity)

A=[40,20,10,30,20,10,30,20,10,20,30,20,10
,40]
bucket_sort(A,(x)->x÷10)

A=[41,22,17,33,25,11,38,20,19,22,31,24,16
,42]
bucket_sort(A,(x)->x÷10)
```


Bildquelle: Portugiesisches Wikipedia

Wie wollen wir die Eimerkette implementieren?

- Verkettete Liste oder Feld für Eimer**kette**?
- Verkettete Listen oder Felder für **Einzel**eimer?
  - Verkettete Listen sparen Platz (einige Eimer haben kaum Einträge, andere haben viele)
  - Aber mit verketteten Listen können wir "schnelle" Sortierverfahren wie Heap-Sort oder Quicksort nicht verwenden
  - **Sortierte Listen**!



Analyse von Bucketsort mit sort! aus O(n log n)

- Sei S(m) die Anzahl der Vergleiche für einen Eimer mit m Schlüsseln
- Setze n_i auf die Anzahl der Schlüssel im i-ten Eimer
- Gesamtzahl der Vergleiche = $\sum^{k}_{i=1} S(n_i)$ bei k Eimern

Analyse mit sort! aus O(n log n)

- Sei $S(m) \in O(m \log m)$
- Falls die Schlüssel gleichmäßig verteilt sind, beträgt die Eimergröße n/k
- Gesamtzahl der Vergleiche für alle k Eimer

  $= k(n/k) \log(n/k)$

  $= n \log(n/k)$
- Falls $k=n/10$, dann reichen $n \log(10)$ Vergleiche (Laufzeit ist linear in n)

Analyse mit sort! aus $O(n^2)$

- Sei $S(m) \in O(m^2)$
- Falls die Schlüssel gleichmäßig verteilt sind, beträgt die Eimergröße n/k
- Gesamtzahl der Vergleiche für all k Eimer

  $= k(n/k)^2$

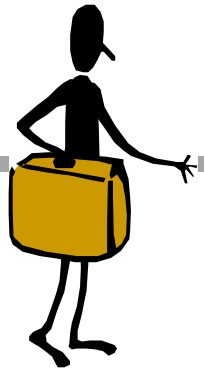
  $= n^2/k$
- Falls $k=n/\log(10)$, dann reichen $n \log(10)$ Vergleiche (Laufzeit ist linear in n, aber man muss mehr Speicher bereitstellen als bei $S(m) \in O(m \log m)$)

Lineare Sortierung: Einsicht

Je mehr man über das Problem weiß, desto eher kann man einen optimalen Algorithmus entwerfen

- Gesucht ist ein Verfahren **S**, so dass **{ P } S { Q }** gilt (Notation nach Hoare)
  - Vorbedingung: **P =?**
  - Invarianten („Axiome“): **I = ?**
  - Nachbedingung: **Q = ∀1≤i<j≤n: A[i] ≤ A[j]**
  - Nebenbedingungen: **?**

Zusammenfassung



- Bisher behandelt:
  - Sortieren durch Vergleichen (vorige Sitzungen)
  - Sortieren durch Verteilen (lineares Sortieren)
  - Wiederholung von elementaren Datenstrukturen
    - Listen, Keller, Warteschlangen
- Es kommt:
  - Prioritätswarteschlangen
  - MinHeaps (zum Vergleich mal anders herum)
  - Binomiale Heaps (effiziente Vereinigung von Heaps)
  - Fibonacci Heaps (Einführung der amortisierten Analyse)