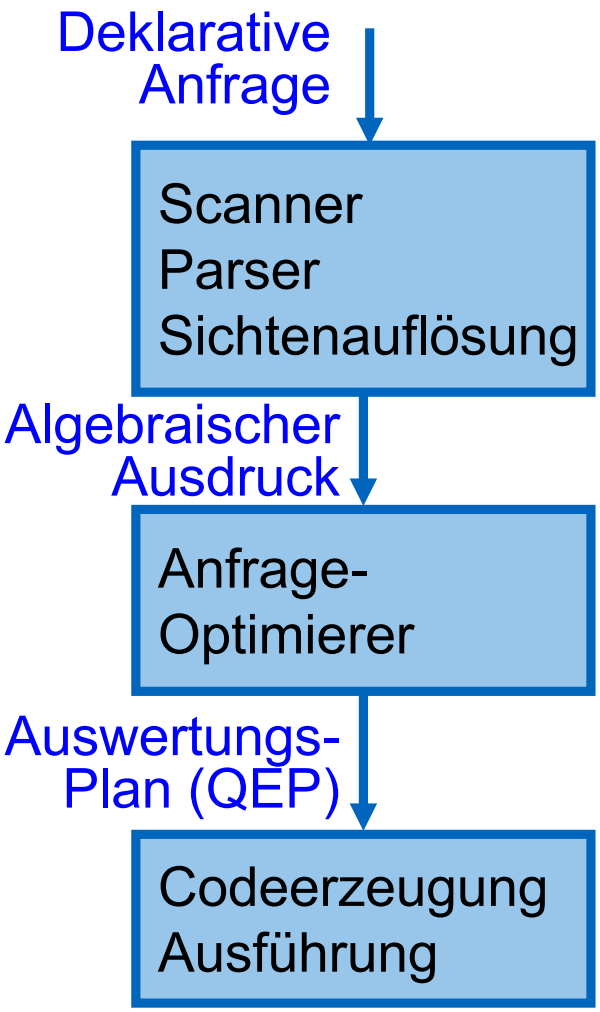
Kapitel 8
Anfragebearbeitung

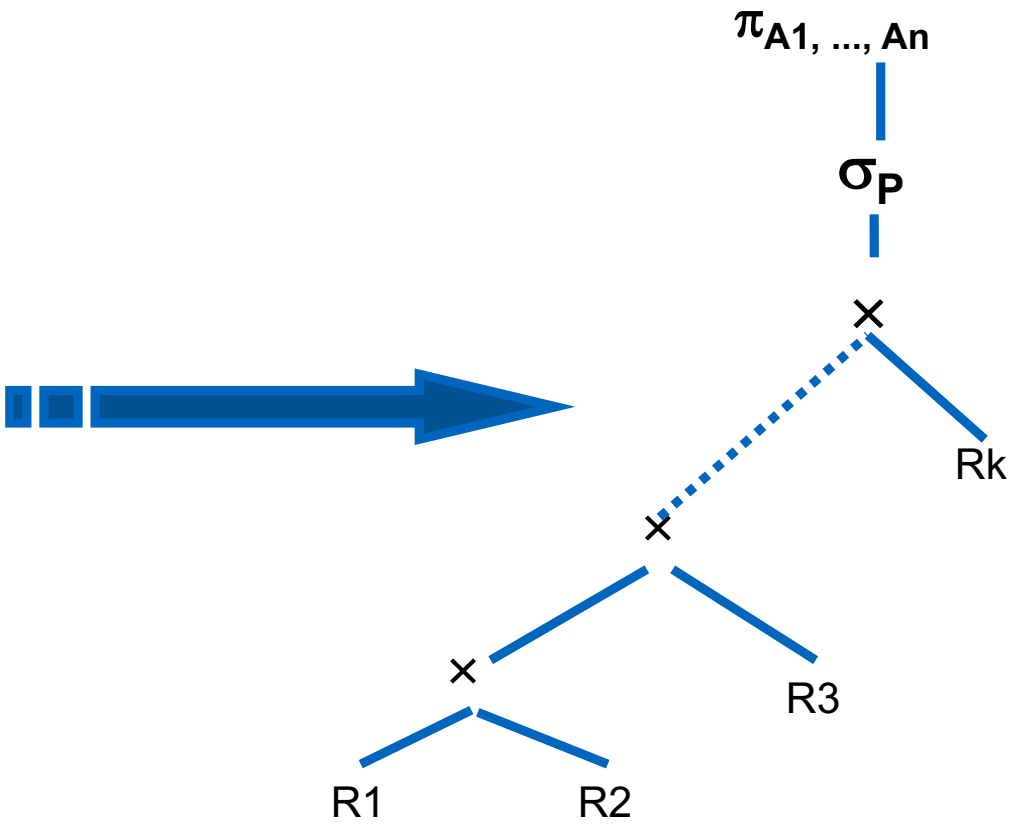
- Logische Optimierung
- Physische Optimierung
- Kostenmodelle
- „Tuning“

Ablauf der Anfrageoptimierung



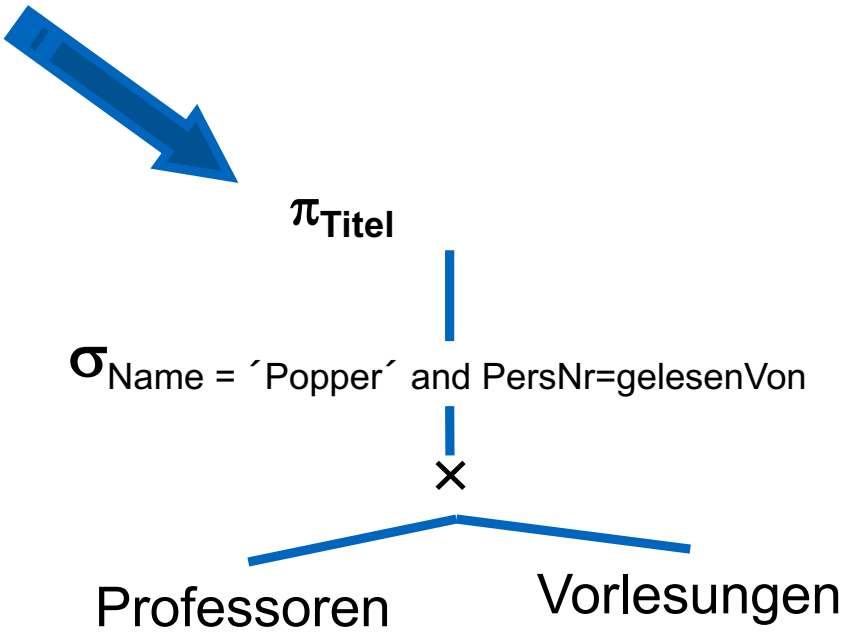
Kanonische Übersetzung

select A1, ..., An
from R1, ..., Rk
where P



Kanonische Übersetzung

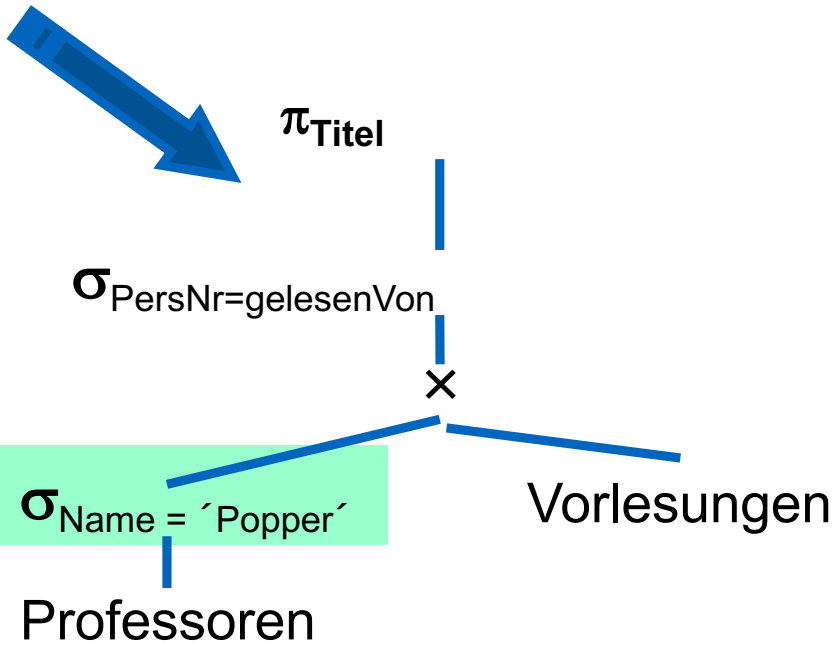
select Titel
from Professoren, Vorlesungen
where Name = ´Popper´ **and**
PersNr = gelesenVon



$\pi_{\text{Titel}}(\sigma_{\text{Name = ´Popper´ and PersNr=gelesenVon}}(\text{Professoren} \times \text{Vorlesungen}))$

Erste Optimierungsidee: „push-down“ Selektionen

select Titel
from Professoren, Vorlesungen
where Name = ´Popper´ **and**
PersNr = gelesenVon



$\pi_{\text{Titel}}(\sigma_{\text{PersNr=gelesenVon}}((\sigma_{\text{Name = ´Popper´}}\ \text{Professoren}) \times \text{Vorlesungen}))$

Optimierung von Datenbank- Anfragen

Grundsätze:

Sehr hohes Abstraktionsniveau der mengenorientierten Schnittstelle (SQL).
Sie ist deklarativ, nicht-prozedural, d.h. es wird spezifiziert, was man finden möchte, aber nicht wie.
Das wie bestimmt sich aus der Abbildung der mengenorientierten Operatoren auf Schnittstellen-Operatoren der internen Ebene (Zugriff auf Datensätze in Dateien, Einfügen/Entfernen interner Datensätze, Modifizieren interner Datensätze).
Zu einem was kann es zahlreiche wie's geben: effiziente Anfrageauswertung durch Anfrageoptimierung.
i.Allg. wird aber nicht die optimale Auswertungsstrategie gesucht (bzw. gefunden) sondern eine einigermaßen effiziente Variante

- Ziel: „avoiding the worst case"

Äquivalenzerhaltende Transformationsregeln

1. Aufbrechen von Konjunktionen im Selektionsprädikat

$$\sigma_{c1 \wedge c2 \wedge \ldots \wedge cn}(R) \equiv \sigma_{c1}(\sigma_{c2}(\ldots(\sigma_{cn}(R))\ldots))$$

2. σ ist kommutativ

$$\sigma_{c1}(\sigma_{c2}((R)) \equiv \sigma_{c2}(\sigma_{c1}((R))$$

3. π -Kaskaden: Falls $L_1 \subseteq L_2 \subseteq \ldots \subseteq L_n$, dann gilt

$$\pi_{L1}(\pi_{L2}(\ldots(\pi_{Ln}(R))\ldots)) \equiv \pi_{L1}(R)$$

4. Vertauschen von σ und π
Falls die Selektion sich nur auf die Attribute $A_1, \ldots, A_n$ der Projektionsliste bezieht, können die beiden Operationen vertauscht werden

$$\pi_{A1, \ldots, An}(\sigma_c(R)) \equiv \sigma_c(\pi_{A1, \ldots, An}(R))$$

5. X, $\cup$, $\cap$ und $\bowtie$ sind kommutativ

$$R \bowtie_c S \equiv S \bowtie_c R$$

Äquivalenzerhaltende Transformationsregeln

6. Vertauschen von σ mit $\bowtie$
Falls das Selektionsprädikat *c* nur auf Attribute der Relation *R* zugreift, kann man die beiden Operationen vertauschen:

$$\sigma_c(R \bowtie_j S) \equiv \sigma_c(R) \bowtie_j S$$

Falls das Selektionsprädikat *c* eine Konjunktion der Form „$c_1 \wedge c_2$" ist und c_1 sich nur auf Attribute aus *R* und c_2 sich nur auf Attribute aus *S* bezieht, gilt folgende Äquivalenz:

$$\sigma_c(R \bowtie_j S) \equiv (\sigma_{c1}(R)) \bowtie_j (\sigma_{c2}(S))$$

Äquivalenzerhaltende Transformationsregeln

7. Vertauschung von π mit $\bowtie$
Die Projektionsliste L sei: $L = \{A_1,...,A_n, B_1,...,B_m\}$, wobei A_i Attribute aus R und B_i Attribute aus S seien. Falls sich das Joinprädikat c nur auf Attribute aus L bezieht, gilt folgende Transformation:

$$\pi_L (R \bowtie_c S) \equiv (\pi_{A_1, ..., A_n} (R)) \bowtie_c (\pi_{B_1, ..., B_n} (S))$$

Falls das Joinprädikat sich auf weitere Attribute, sagen wir A_1', ..., A_p', aus R und B_1', ..., B_q' aus S bezieht, müssen diese für die Join-Operation erhalten bleiben und können erst danach herausprojiziert werden:

$$\pi_L (R \bowtie_c S) \equiv \pi_L (\pi_{A_1, ..., A_n, A_1', ..., A_p'} (R) \bowtie_c \pi_{B_1, ..., B_n, B_1', ..., B_q'} (R))$$

Für die X-Operation gibt es kein Prädikat, so dass die Einschränkung entfällt.

Äquivalenzerhaltende Transformationsregeln

8. Die Operationen $\times$, $\bowtie$, $\cup$, $\cap$ sind jeweils (einzeln betrachtet) assoziativ. Wenn also Φ eine dieser Operationen bezeichnet, so gilt:

$$(R \,\Phi S\,)\,\Phi T \equiv R\,\Phi(S\,\Phi T\,)$$

9. Die Operation σ ist distributiv mit $\cup$, $\cap$, $-$. Falls Φ eine dieser Operationen bezeichnet, gilt:

$$\sigma_c(R\,\Phi S) \equiv (\sigma_c\,(R))\;\Phi\;(\sigma_c\,(S))$$

10. Die Operation π ist distributiv mit $\cup$

$$\pi_c(R \cup S) \equiv (\pi_c\,(R)) \cup (\pi_c\,(S))$$

Äquivalenzerhaltende Transformationsregeln

11. Die Join- und/oder Selektionsprädikate können mittels de Morgan's Regeln umgeformt werden:

$$\neg (c_1 \wedge c_2) \equiv (\neg c_1) \vee (\neg c_2)$$
$$\neg (c_1 \vee c_2) \equiv (\neg c_1) \wedge (\neg c_2)$$

12. Ein kartesisches Produkt, das von einer Selektions-Operation gefolgt wird, deren Selektionsprädikat Attribute aus beiden Operanden des kartesischen Produktes enthält, kann in eine Joinoperation umgeformt werden.

Sei c eine Bedingung der Form $A\ \theta\ B$, mit A ein Attribut von R und B ein Attribut aus S.

$$\sigma_c(R \times S) \equiv R \bowtie_c S$$

Heuristische Anwendung der Transformationsregeln

1. Mittels Regel 1 werden konjunktive Selektionsprädikate in Kaskaden von σ-Operationen zerlegt.

2. Mittels Regeln 2, 4, 6, und 9 werden Selektionsoperationen soweit „nach unten" propagiert wie möglich.

3. Mittels Regel 8 werden die Blattknoten so vertauscht, dass derjenige, der das kleinste Zwischenergebnis liefert, zuerst ausgewertet wird.

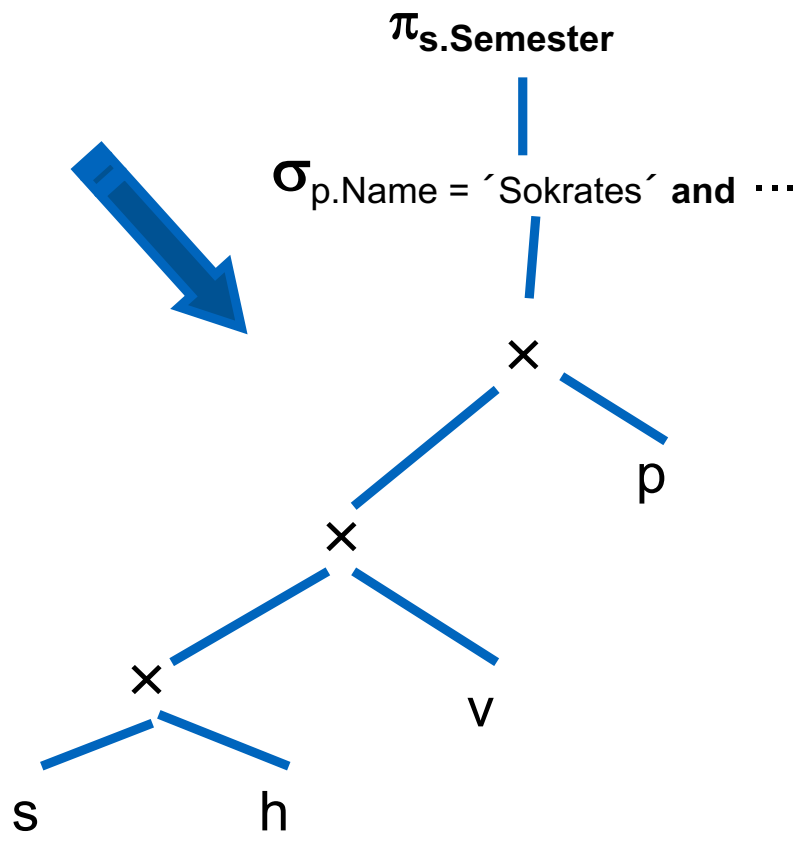
4. Forme eine X-Operation, die von einer σ-Operation gefolgt wird, wenn möglich in eine $\bowtie$-Operation um

5. Mittels Regeln 3, 4, 7, und 10 werden Projektionen soweit wie möglich nach unten propagiert.

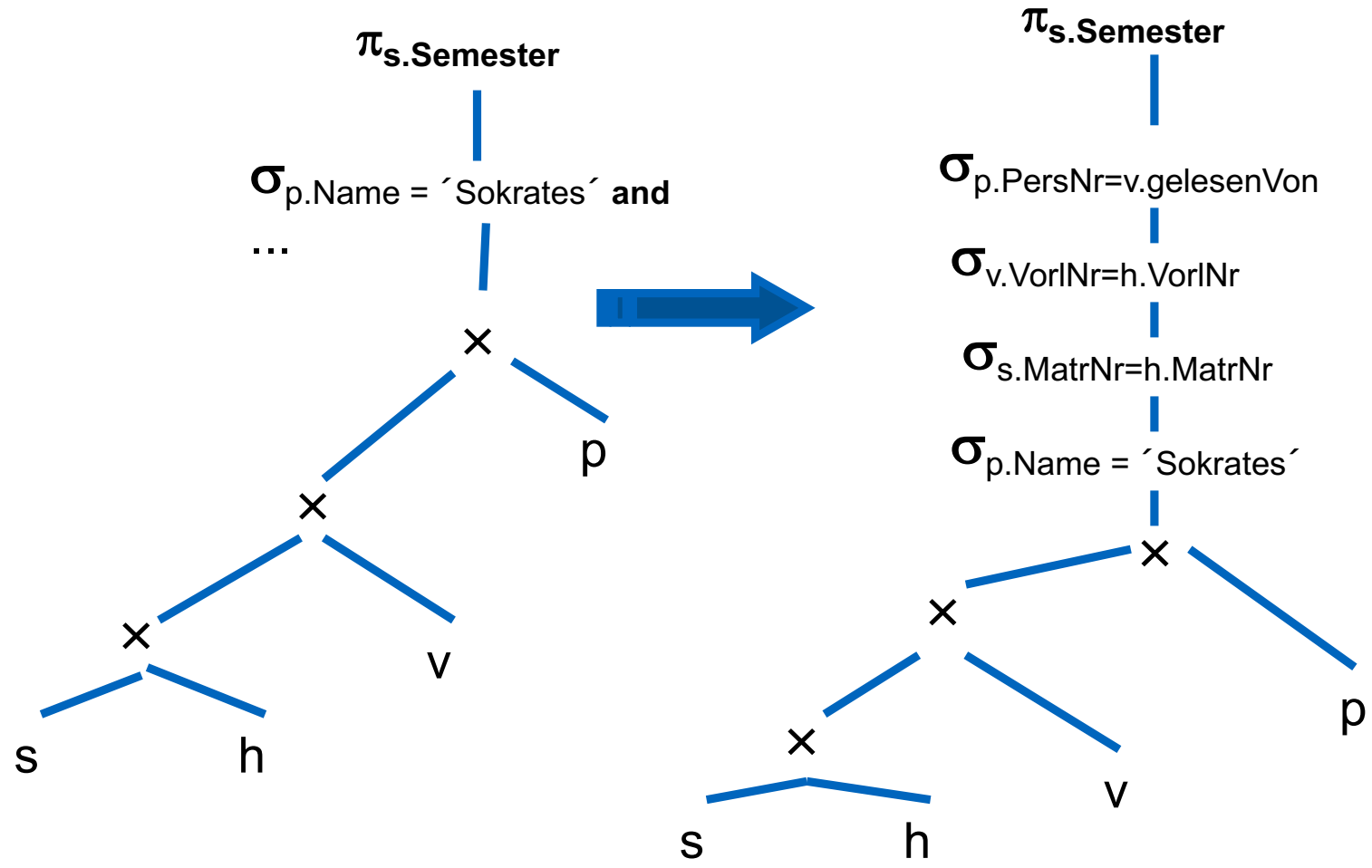
6. Versuche Operationsfolgen zusammenzufassen, wenn sie in einem „Durchlauf" ausführbar sind (z.B. Anwendung von Regel 1, Regel 3, aber auch Zusammenfassung aufeinanderfolgender Selektionen und Projektionen zu einer „Filter"-Operation).

Anwendung der Transformationsregeln

```
select distinct s.Semester
from Studenten s, hören h,
        Vorlesungen v, Professoren p
where p.Name = ´Sokrates´ and
        v.gelesenVon = p.PersNr and
        v.VorlNr = h.VorlNr and
        h.MatrNr = s.MatrNr
```

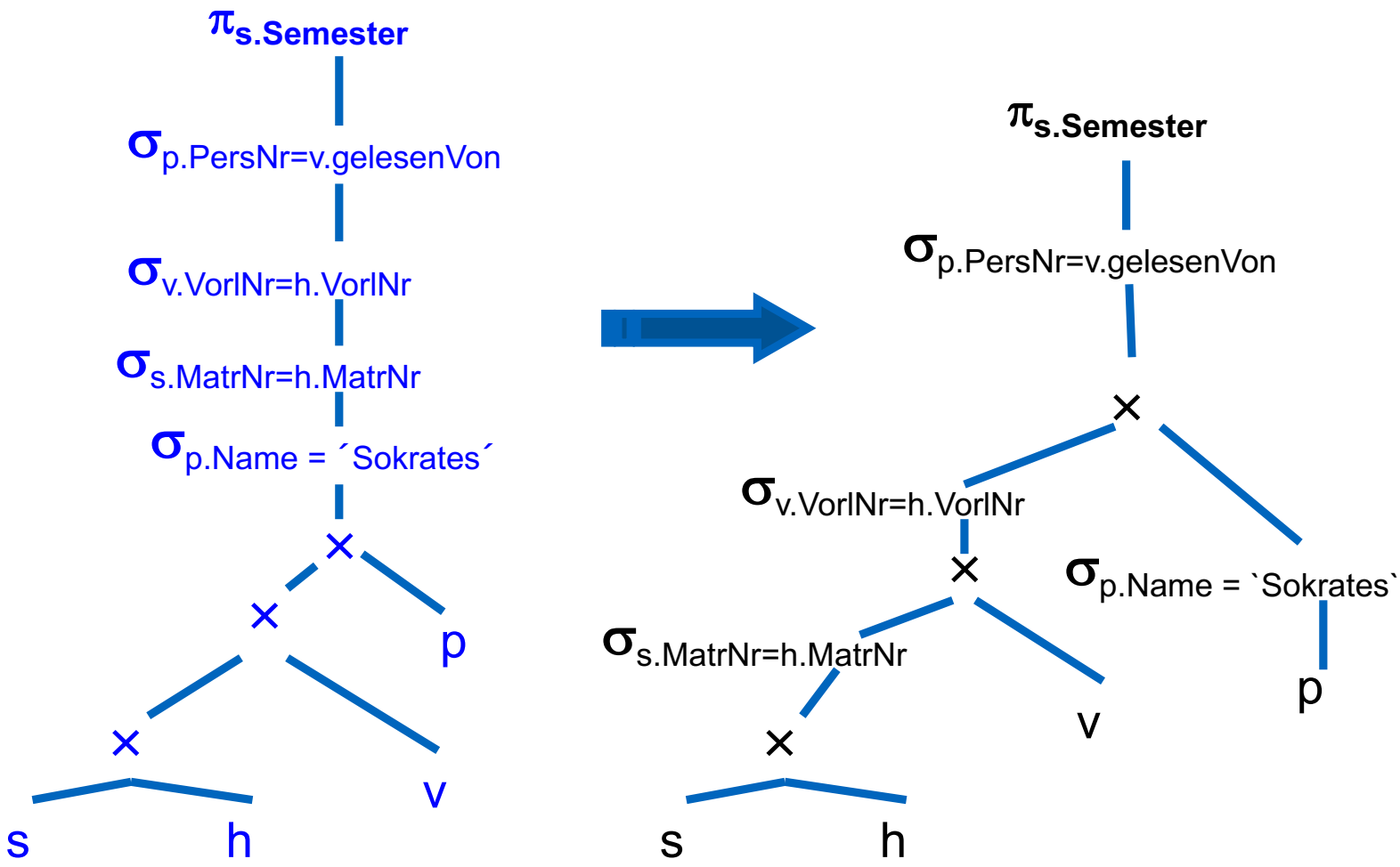


Aufspalten der Selektionsprädikate

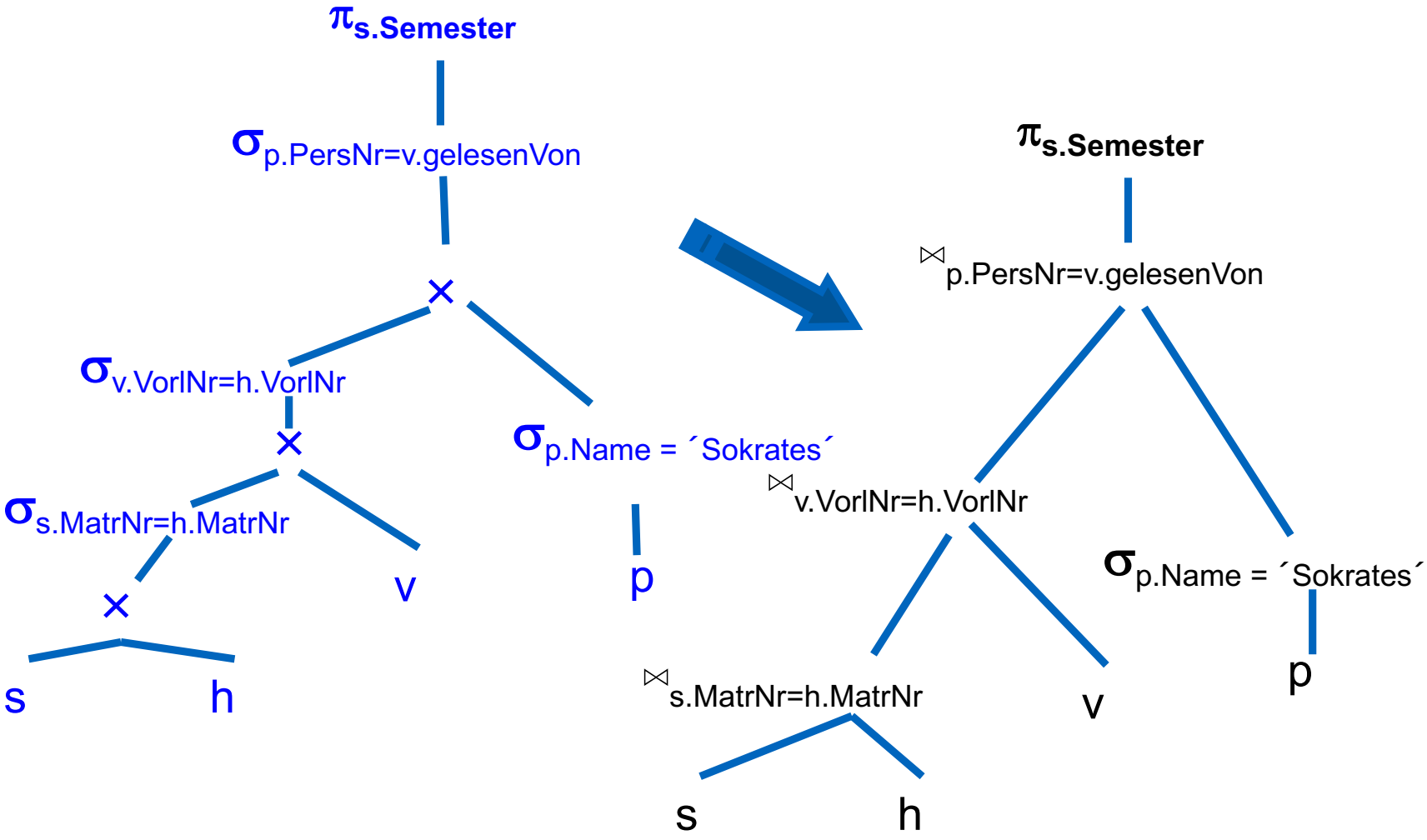


Verschieben der Selektionsprädikate

„Pushing Selections“

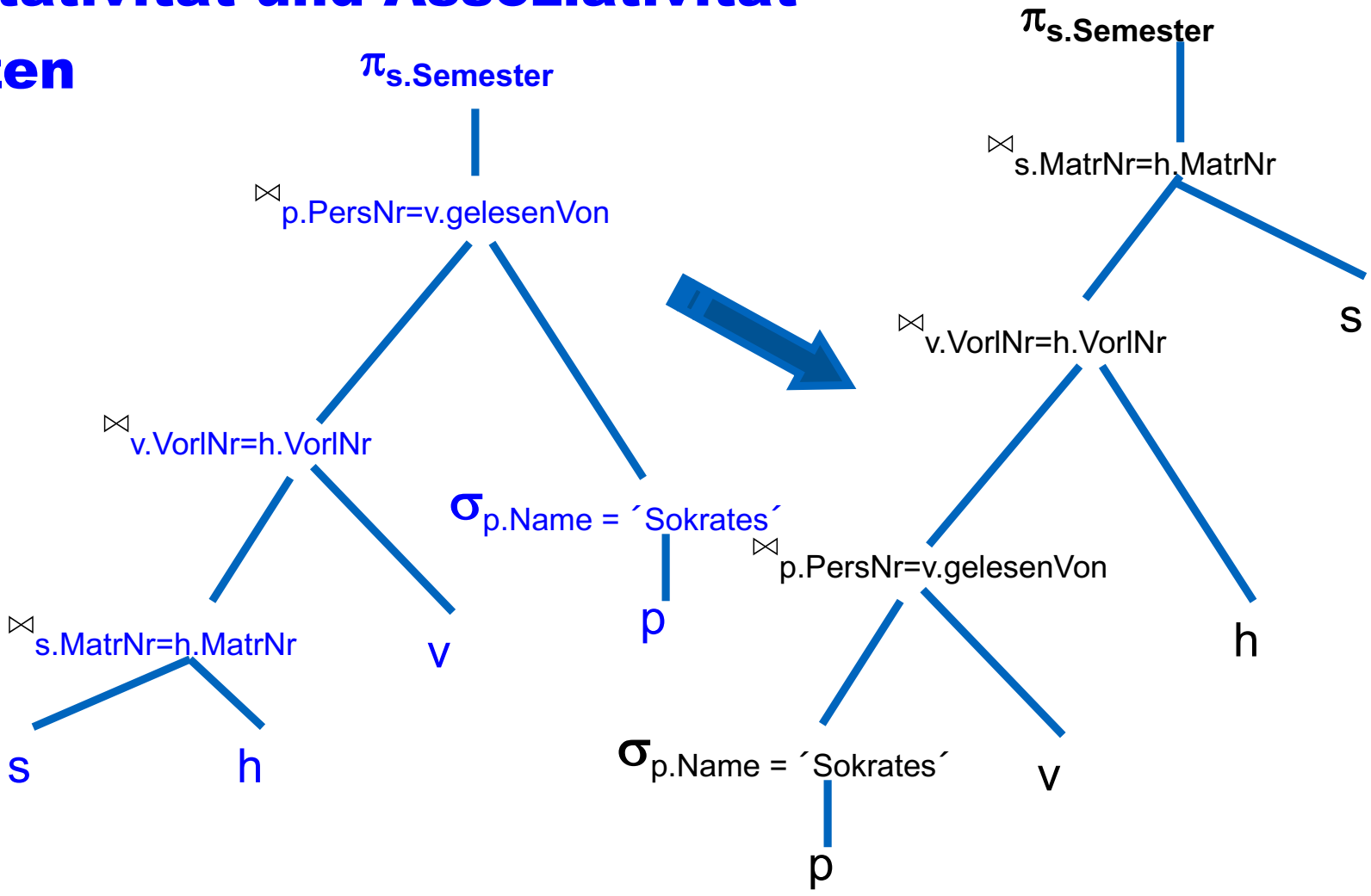


Zusammenfassung von Selektionen und Kreuzprodukten zu Joins

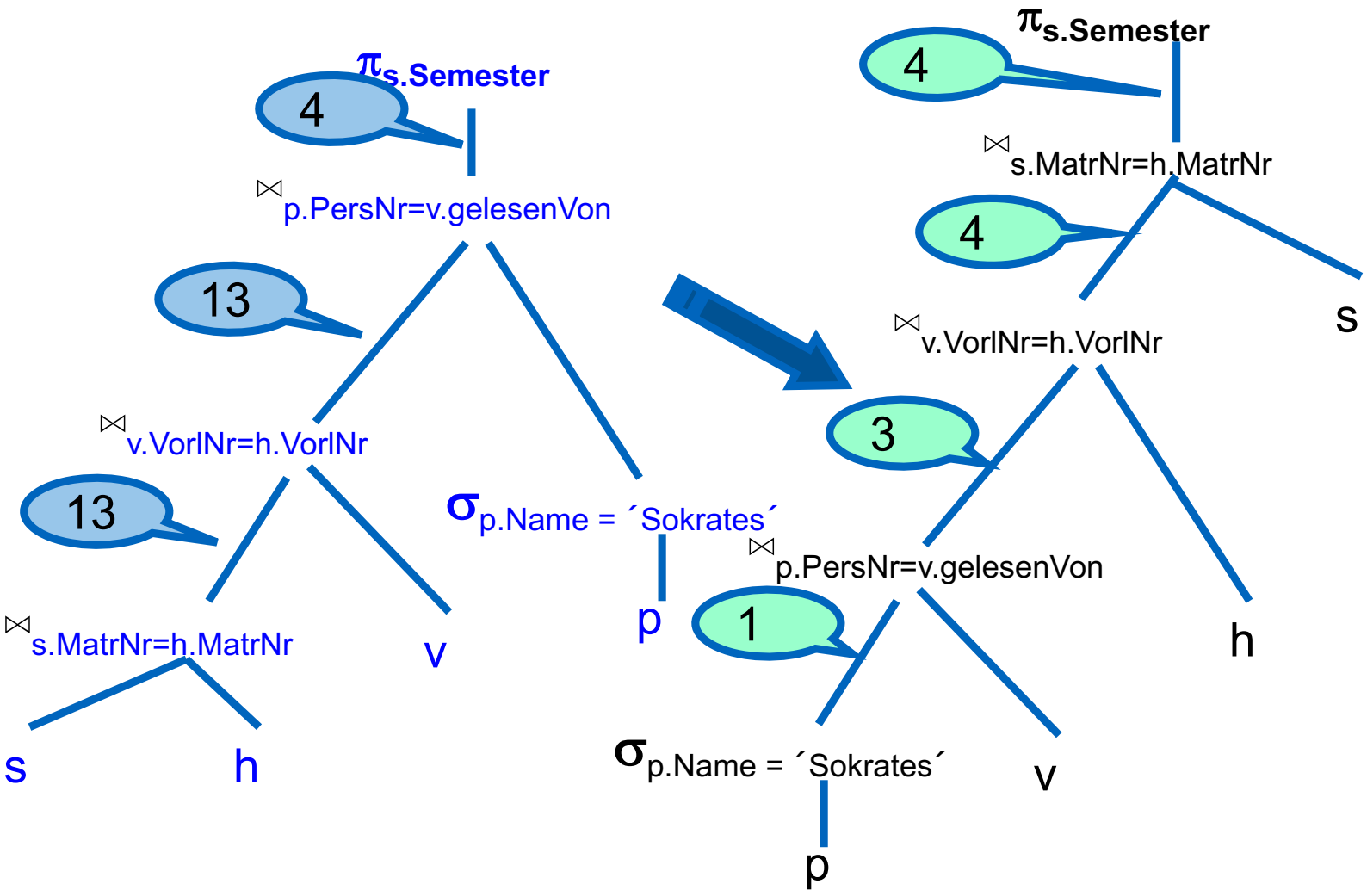


Optimierung der Joinreihenfolge

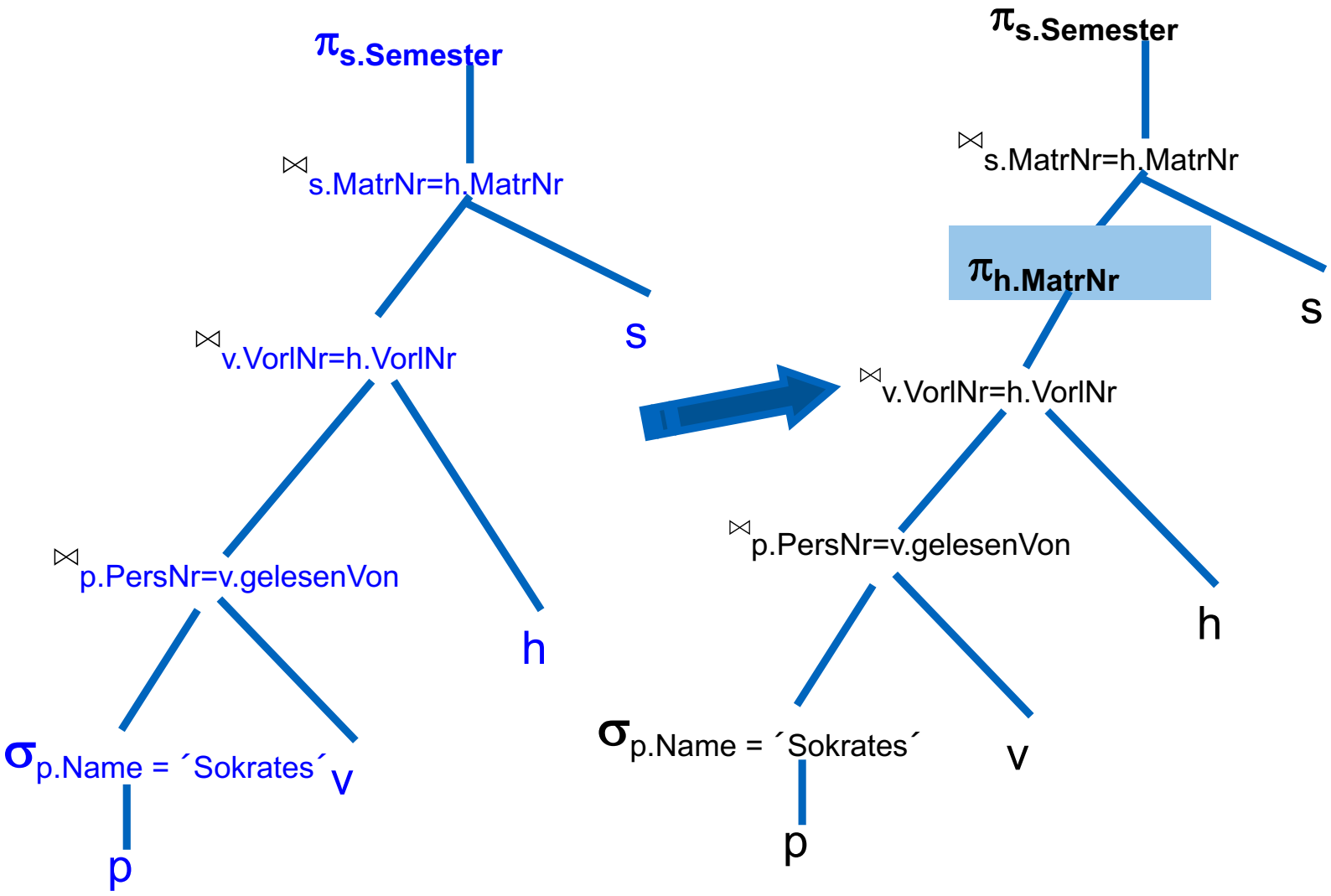
Kommutativität und Assoziativität ausnutzen



Was hat´s gebracht?



Einfügen von Projektionen



Eine weitere Beispieloptimierung

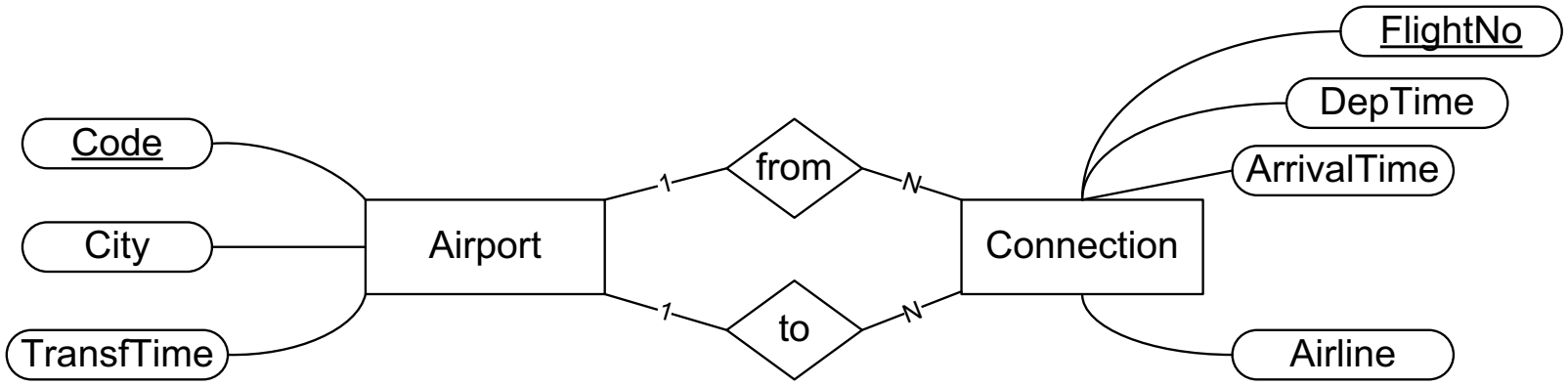
SQL-Anfrage: Von München direkt nach NY?

```
select c.dep
from Airport n, Connection c, Airport p
where n.loc = 'New York' and
      n.code = c.to and
      c.from = p.code and
      p.loc = 'München'
```

Airport p		
p.code	p.loc	...
...	...	...
MUC	München	...
...	...	...

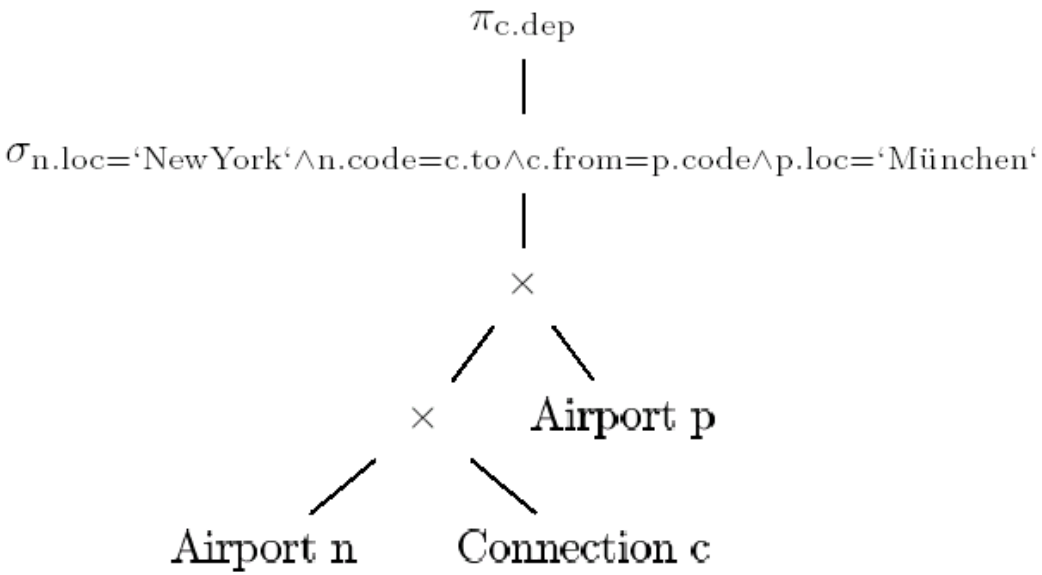
Connection c			
...	c.from	c.to	...

Airport n		
n.code	n.loc	...
...	...	...
JFK	New York	...
LGA	New York	...
...	...	...


Code
City
TransfTime
Airport
from
to
1
N
1
N
Connection
FlightNo
DepTime
ArrivalTime
Airline

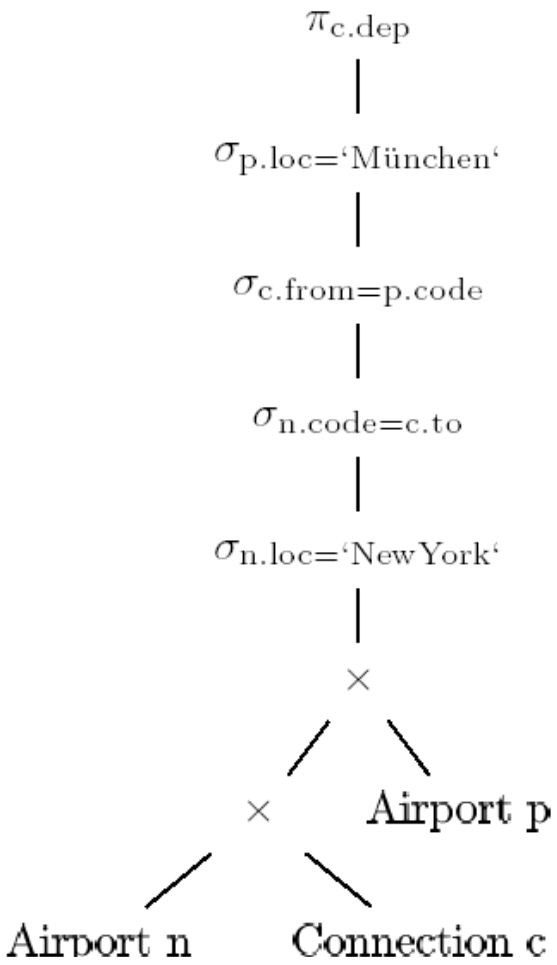
Kanonische Übersetzung

select c.dep
from Airport n, Connection c, Airport p
where n.loc = 'New York' **and**
n.code = c.to **and**
c.from = p.code **and**
p.loc = 'München'



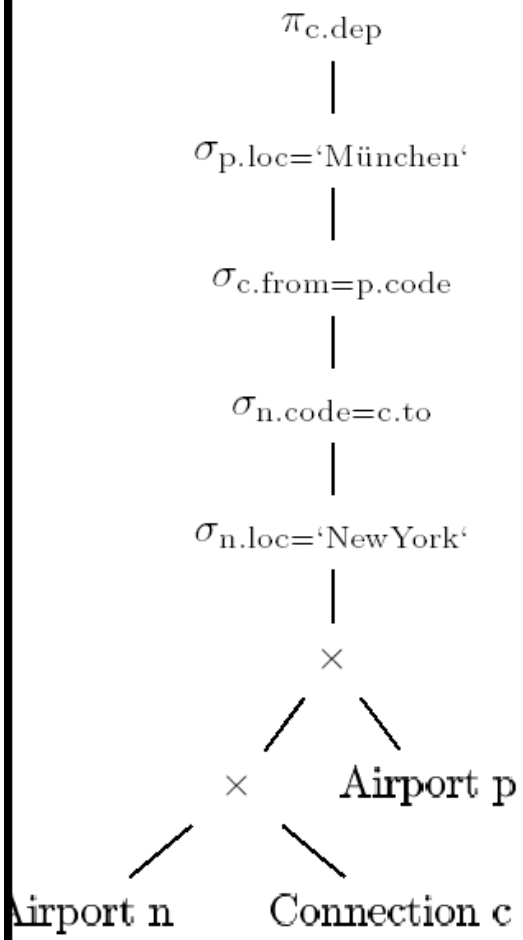
Selektionsprädikate „aufbrechen“

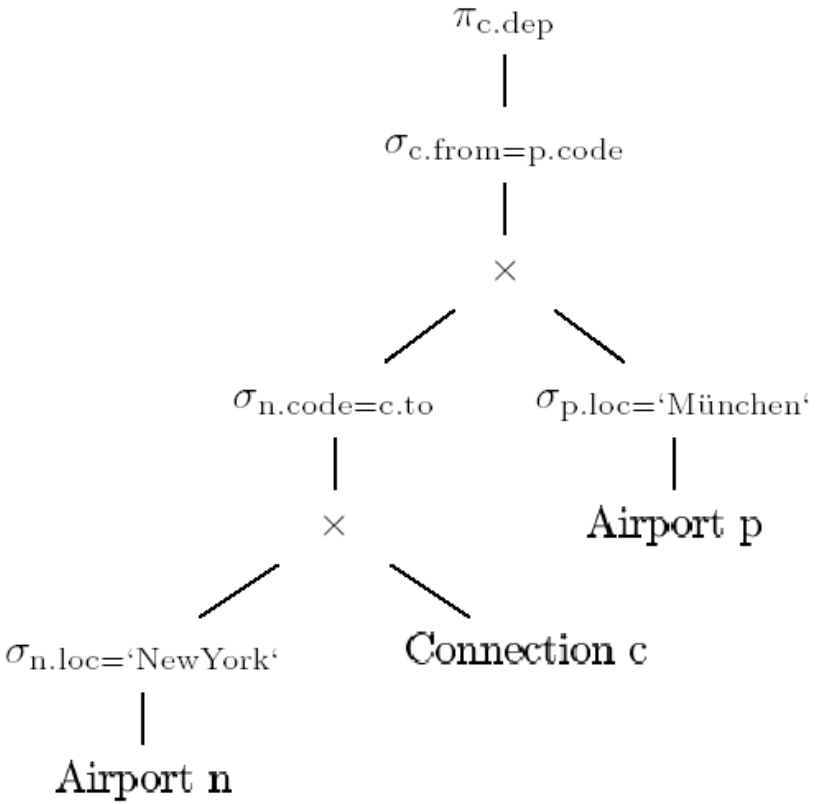
$$\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots(\sigma_{p_n}(R))\ldots))$$



„Pushing Selections“

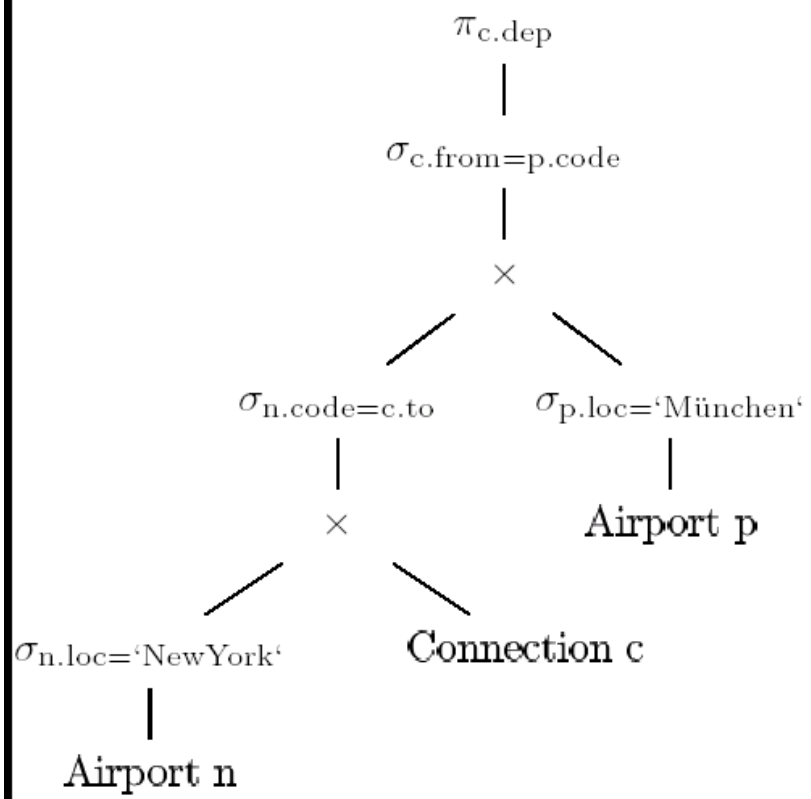
$$
\begin{aligned}
\sigma_p(\sigma_q(R)) &= \sigma_q(\sigma_p(R)) \\
\sigma_p(R_1 \bowtie R_2) &= \sigma_p(R_1) \bowtie R_2 \\
\sigma_p(R_1 \times R_2) &= \sigma_p(R_1) \times R_2
\end{aligned}
$$

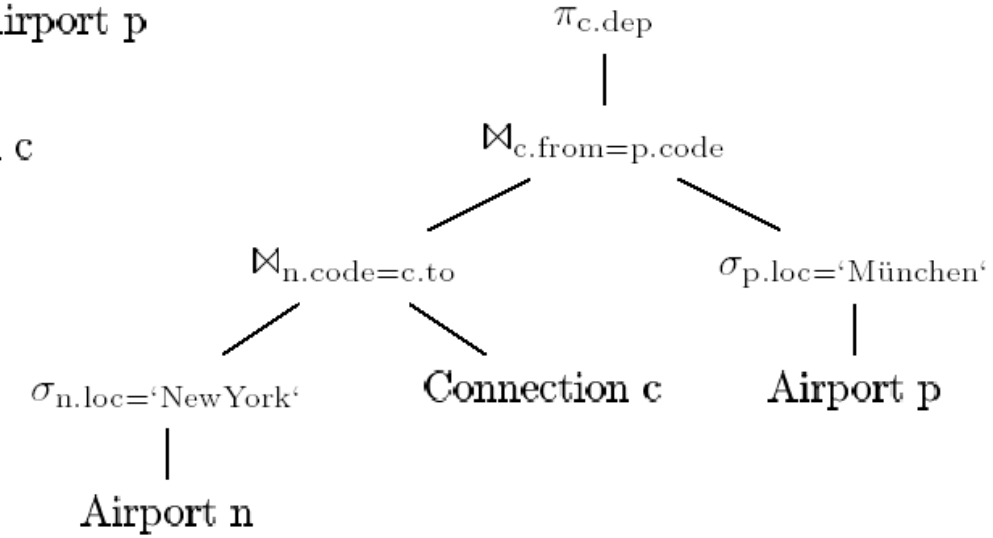


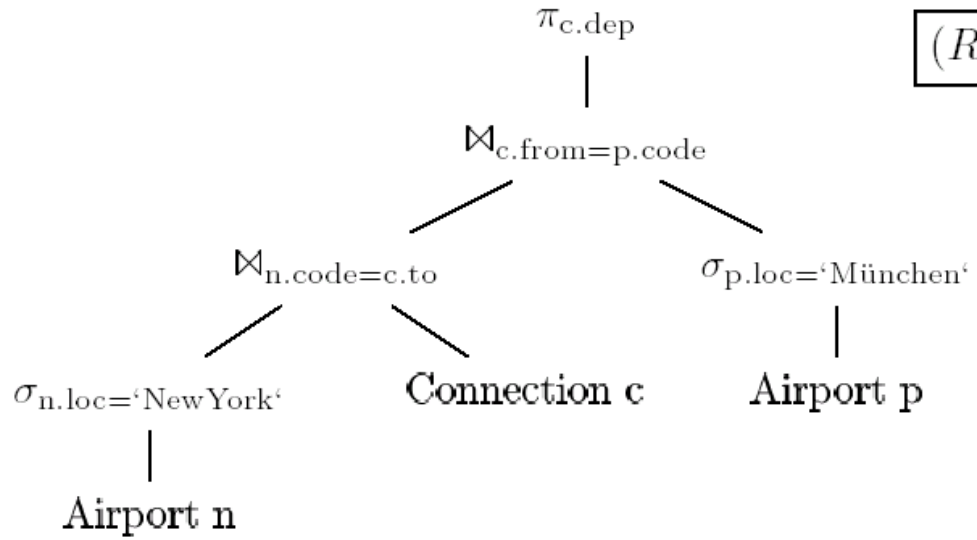


Zusammenfassung von $\sigma\times$ zu $\bowtie$

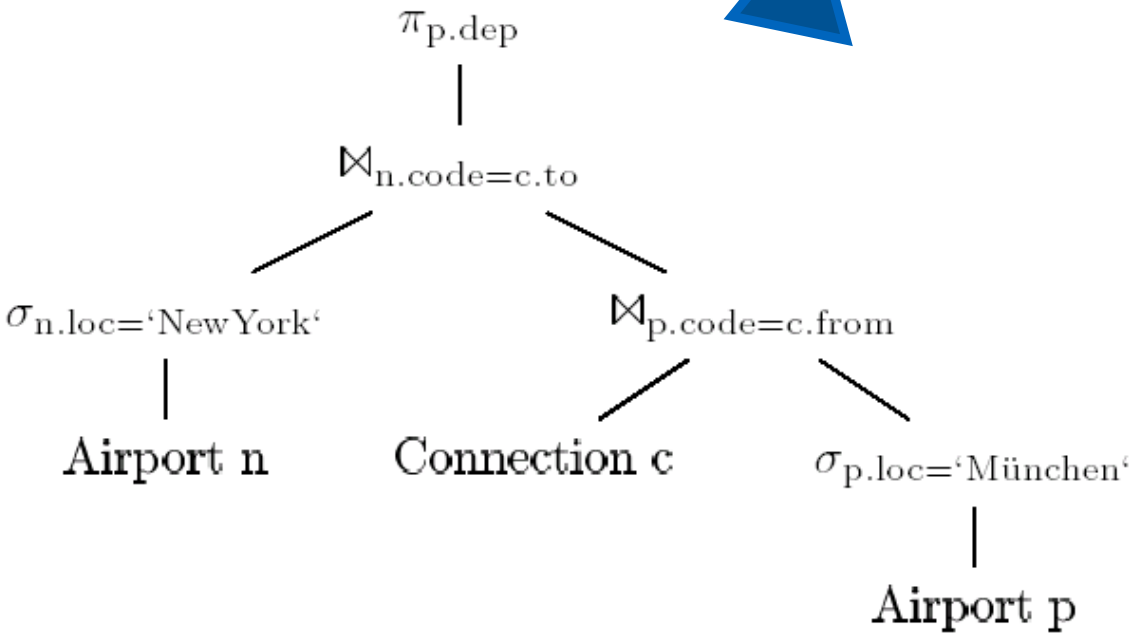
$$\sigma_{R.A=S.B}(R \times S) = R \bowtie_{R.A=S.B} S$$





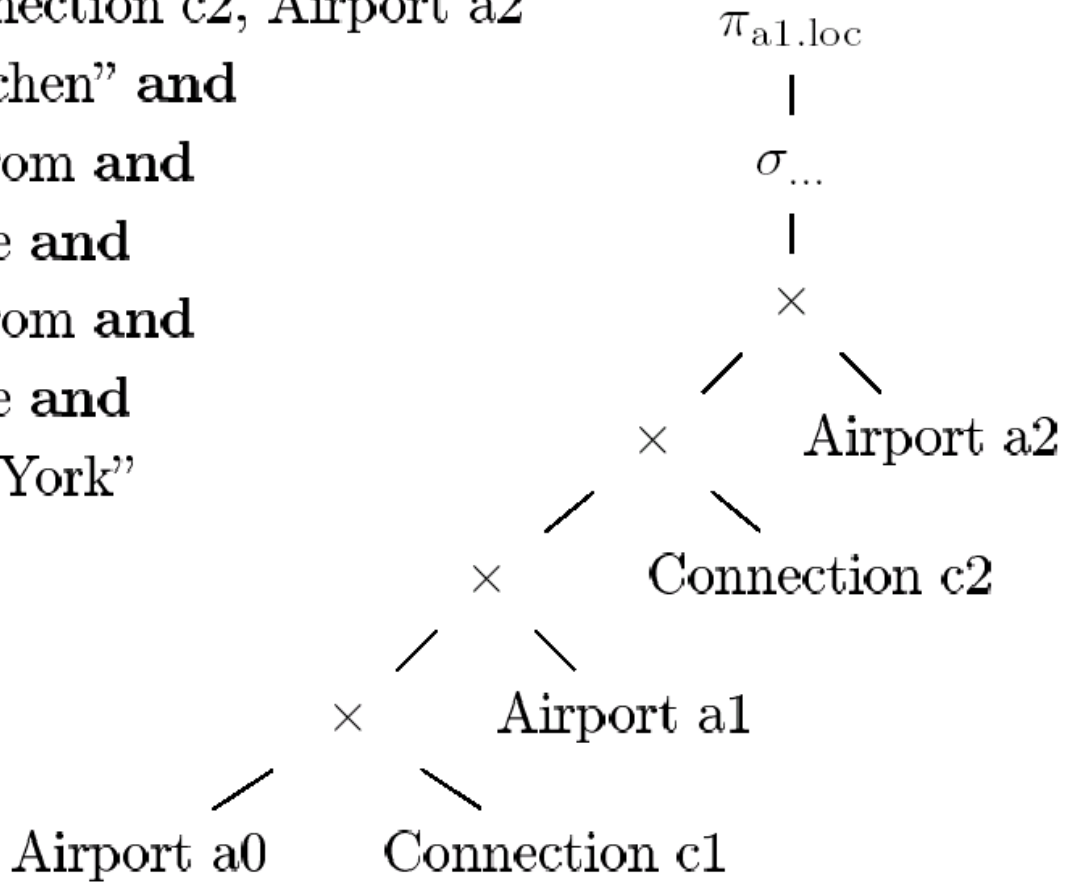


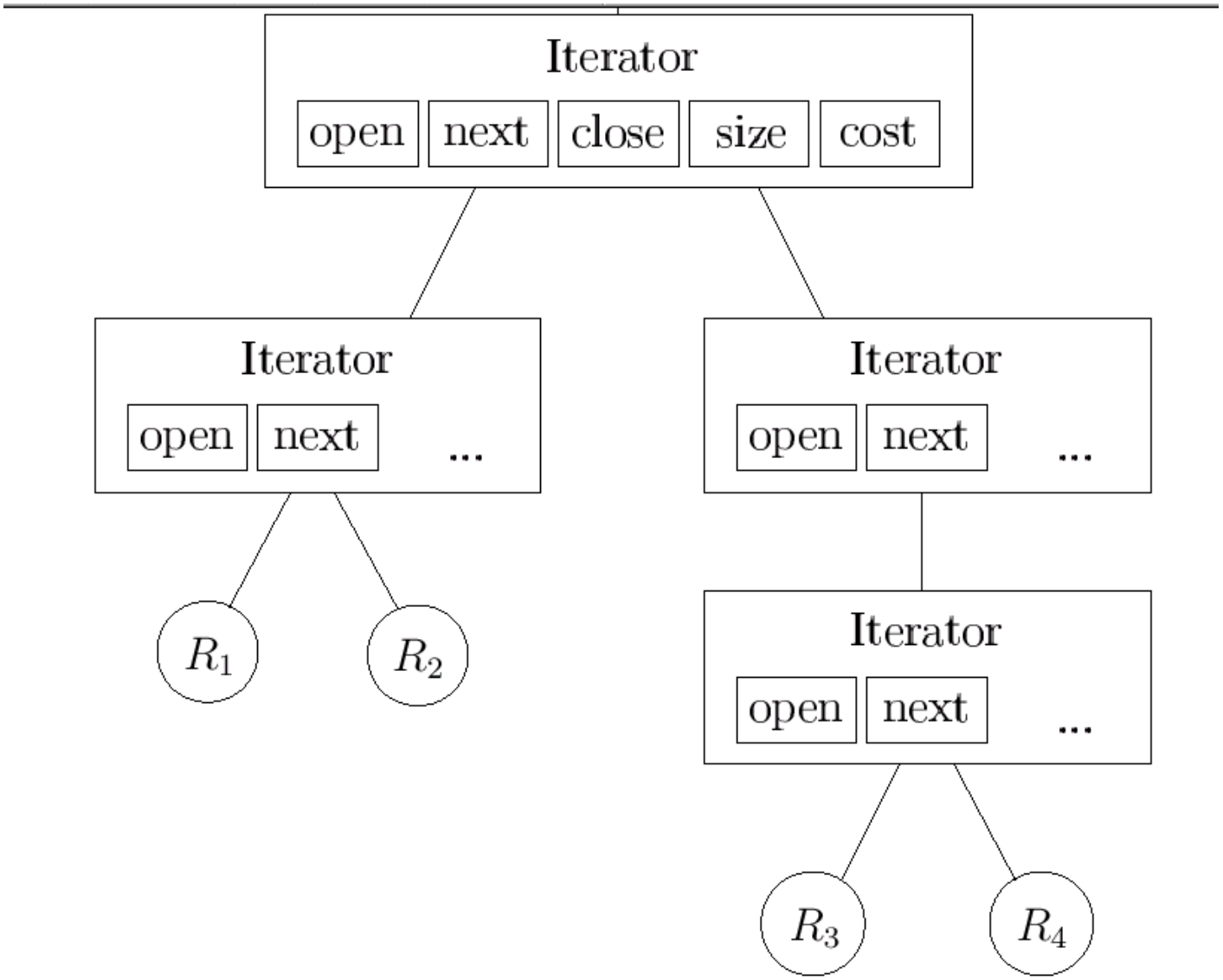
$$(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$$



MUC → NY mit genau einmal Umsteigen

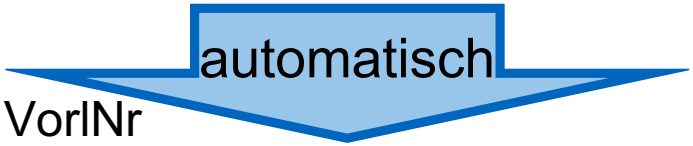
select a1.loc
from Airport a0, Connection c1,
Airport a1, Connection c2, Airport a2
where a0.loc = "München" **and**
a0.code = c1.from **and**
c1.to = a1.code **and**
a1.code = c2.from **and**
c2.to = a2.code **and**
a2.loc = "New York"




Iterator
open
next
close
size
cost
Iterator
open
next
...
Iterator
open
next
...
R_1
R_2
Iterator
open
next
...
R_3
R_4

Entschachtelung / Unnesting

```
select s.Name, p.VorlNr
from Studenten s , prüfen p
where s.MatrNr = p.MatrNr and p.Note = (
            select min(p2.Note)
            from prüfen p2
            where s.MatrNr=p2.MatrNr )
```




```
select s.Name, p.VorlNr
from Studenten s , prüfen p ,
            (select p2.MatrNr as ID, min(p2.Note) as beste
            from prüfen p2
            group by p2.MatrNr) m
where s.MatrNr=p.MatrNr and m.ID=s.MatrNr
        and p.Note=m.beste
```

Dependent Join (nested loop Semantik)

$$T_1 \ltimes_p T_2 \quad := \quad \{t_1 \circ t_2 | t_1 \in T_1 \wedge t_2 \in T_2(t_1) \wedge p(t_1 \circ t_2)\}.$$

Einfache Entschachtelung

Q2:

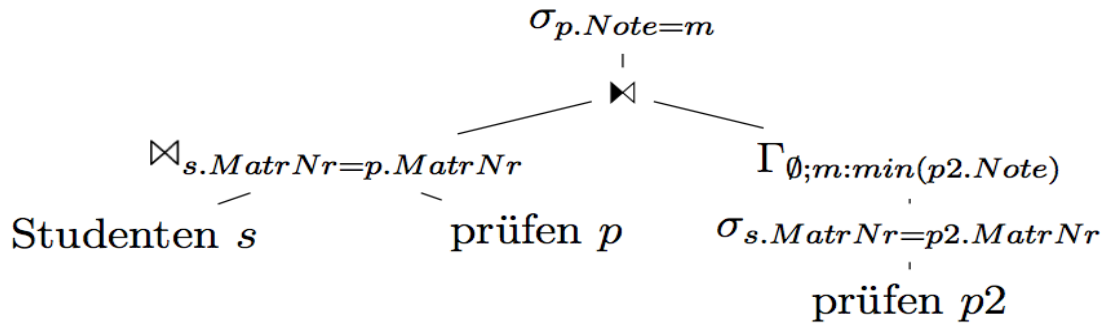
```
select s.*
from Studenten s
where exists (select * from prüfen p
                       where s.MatrNr = p.MatrNr)
```

$$(Studenten\ s) \blacktriangleright (\sigma_{s.MatrNr=p.MatrNr}(prüfen\ p))$$

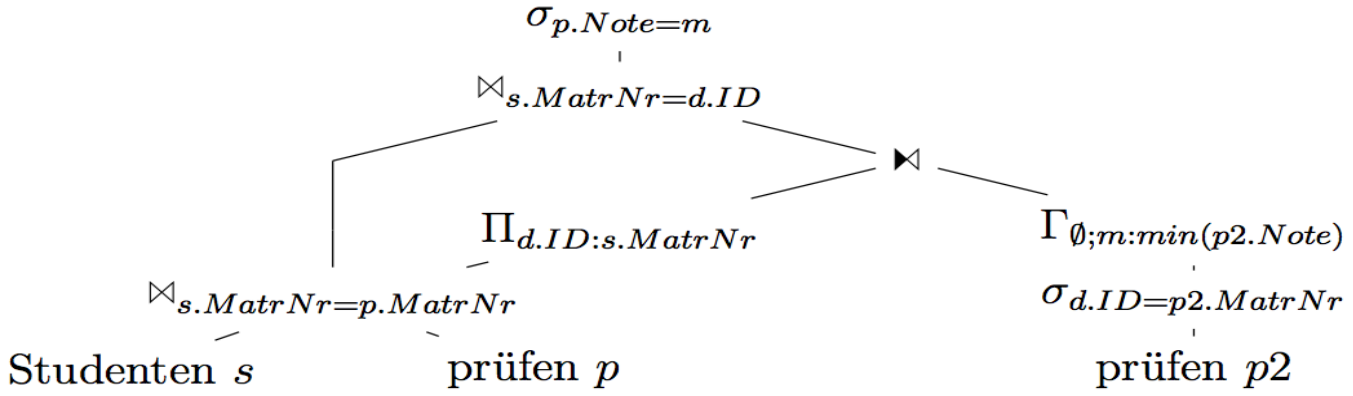
$$(Studenten\ s) \ltimes_{s.MatrNr=p.MatrNr} (prüfen\ p)$$

$$T_1 \bowtie_p T_2 \equiv T_1 \bowtie_{p \wedge T_1 =_{\mathcal{A}(D)} D} (D \bowtie T_2)$$

wobei $D := \Pi_{\mathcal{F}(T_2) \cap \mathcal{A}(T_1)}(T_1)$.



ıg 8.9: Original-Anfrage Q1



Weitere Transformationsregeln

$$D \bowtie \sigma_p(T_2) \equiv \sigma_p(D \bowtie T_2).$$

$$D \bowtie \sigma_p(T_2) \equiv \sigma_p(D \bowtie T_2).$$

$$D \bowtie (\Gamma_{A;a:f}(T)) \equiv \Gamma_{A \cup \mathcal{A}(D);a:f}(D \bowtie T)$$

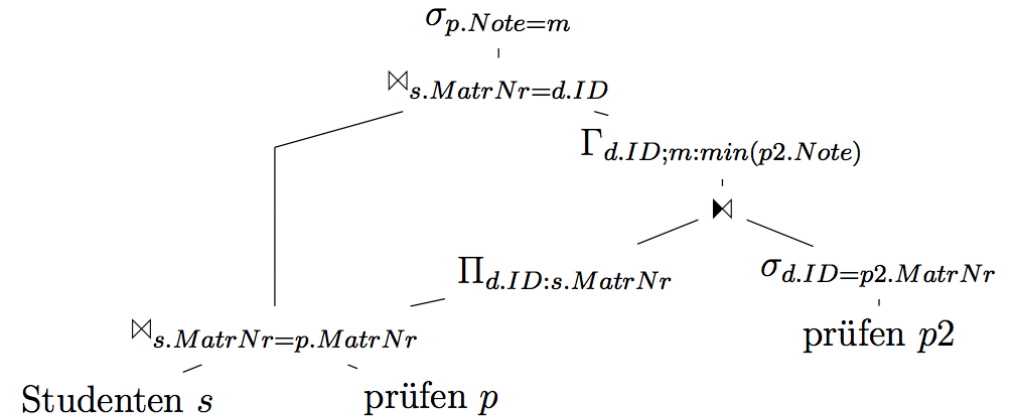
$$D \bowtie (\Pi_A(T)) \equiv \Pi_{A \cup \mathcal{A}(D)}(D \bowtie T)$$

$$D \bowtie (T_1 \cup T_2) \equiv (D \bowtie T_1) \cup (D \bowtie T_2)$$

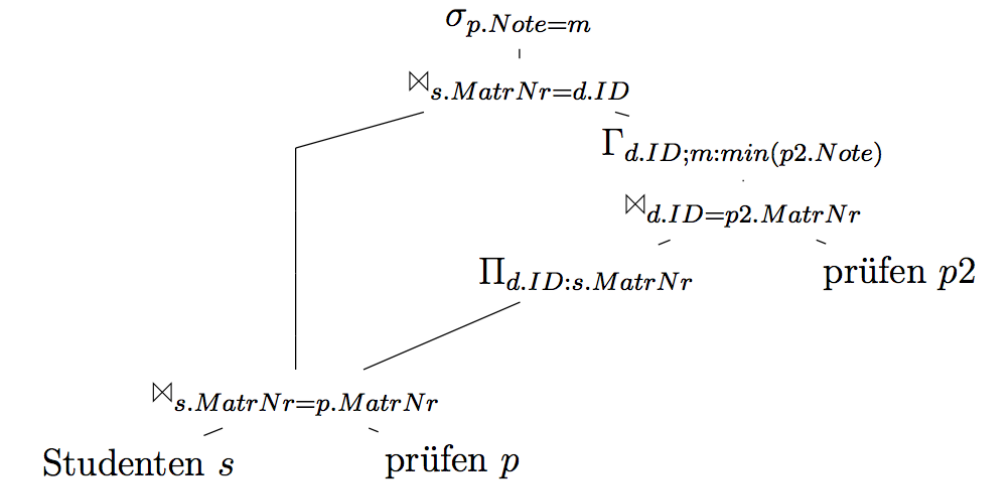
$$D \bowtie (T_1 \cap T_2) \equiv (D \bowtie T_1) \cap (D \bowtie T_2)$$

$$D \bowtie (T_1 \setminus T_2) \equiv (D \bowtie T_1) \setminus (D \bowtie T_2)$$

Beispiel

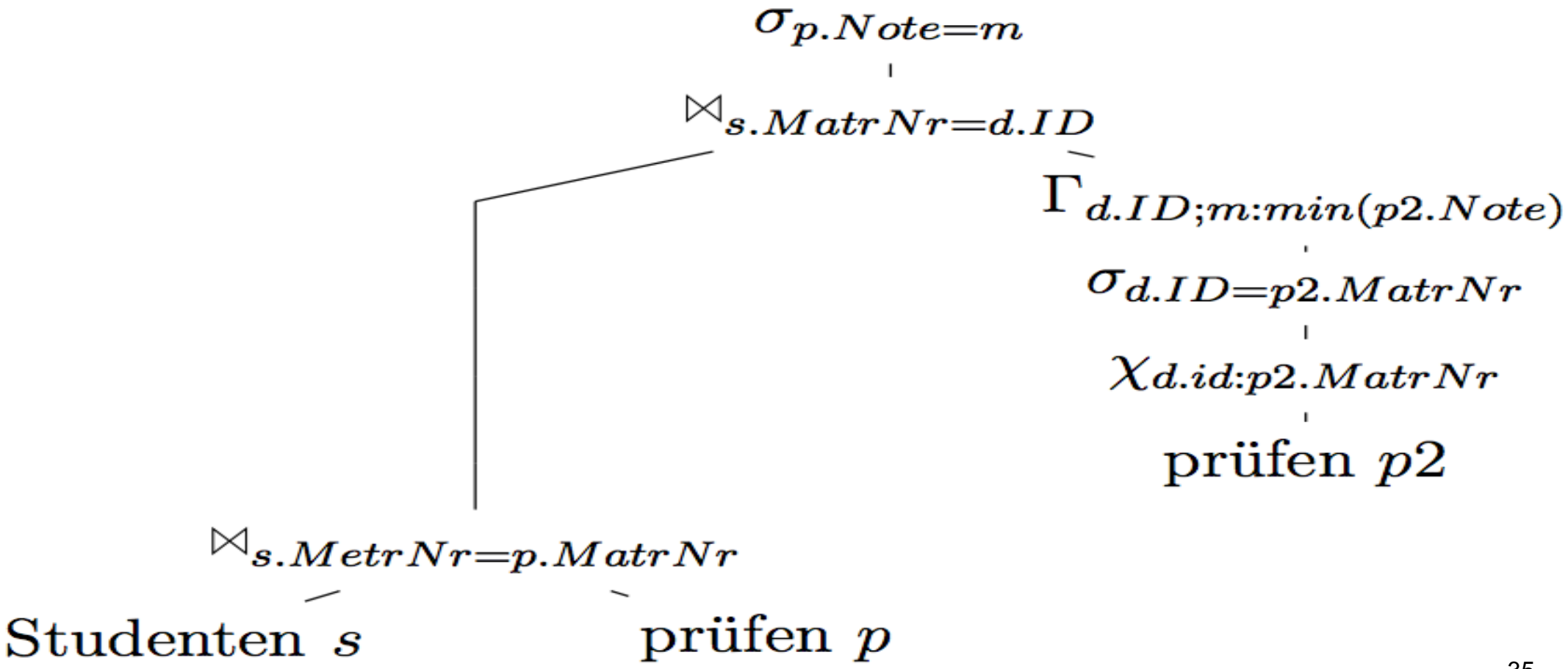


8.11: Vertauschen von Gruppierung/Aggregation mit dem a

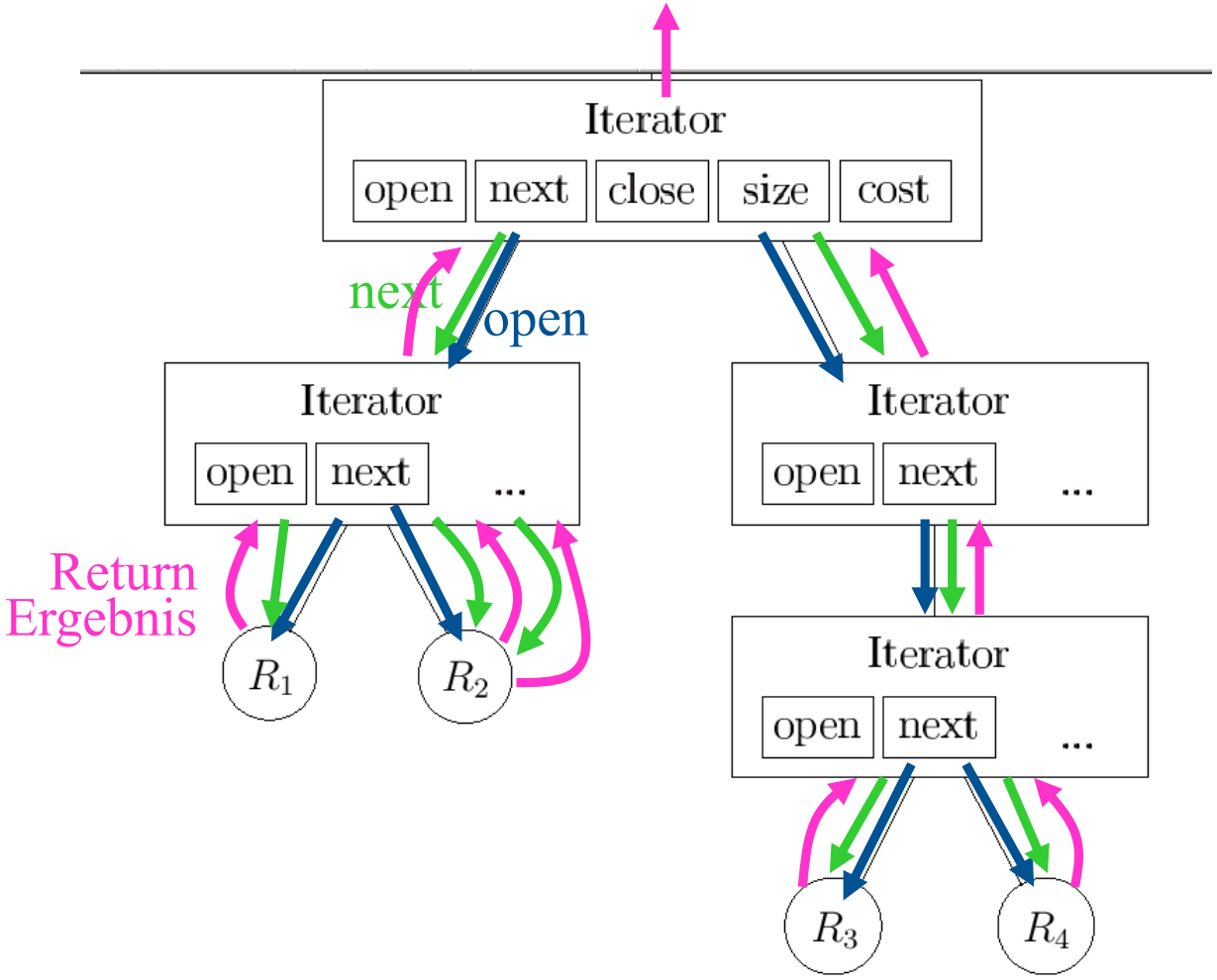


8.12: Elimination des abhängigen Joins

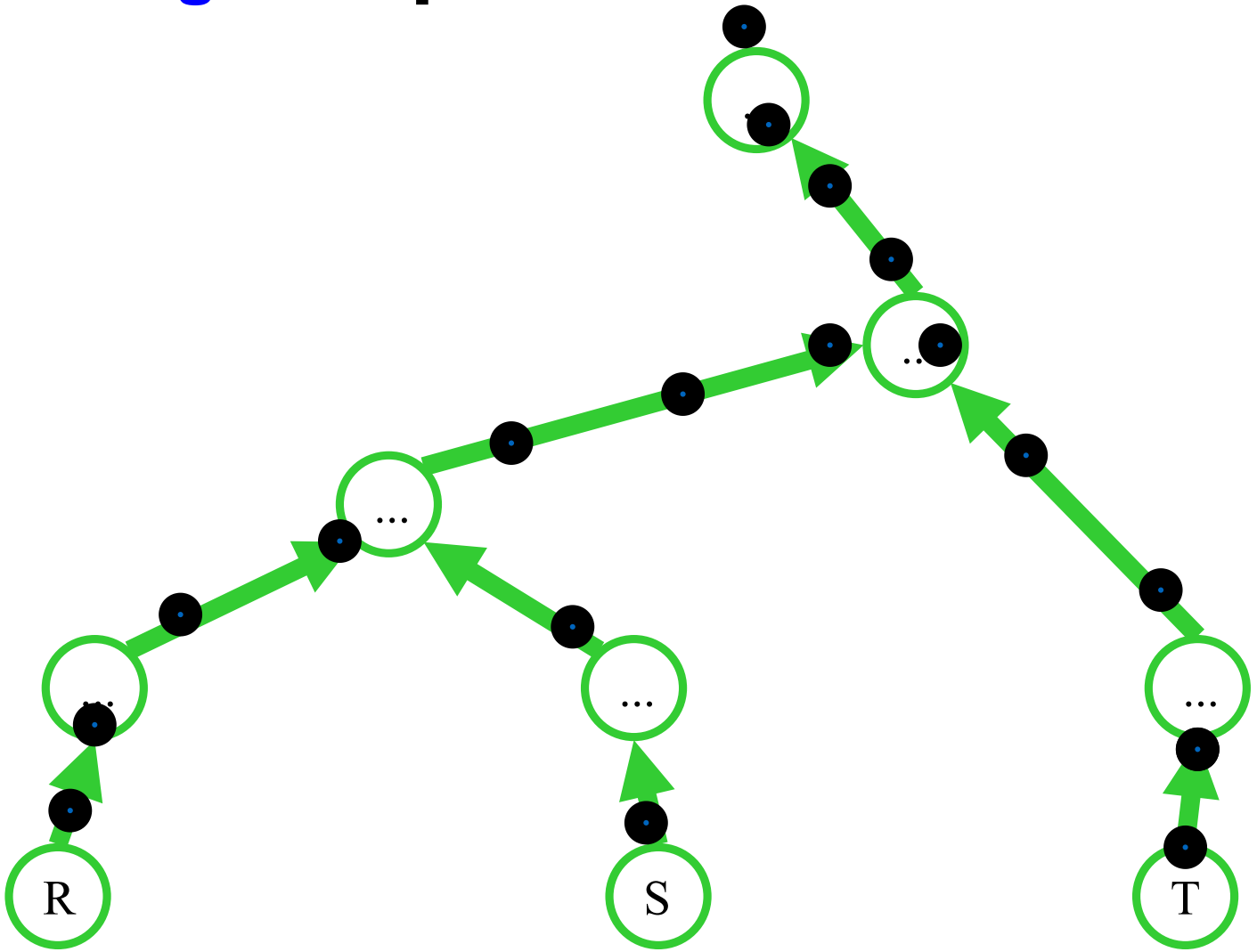
Entkoppelung rechter Seite von linker Seite: optional



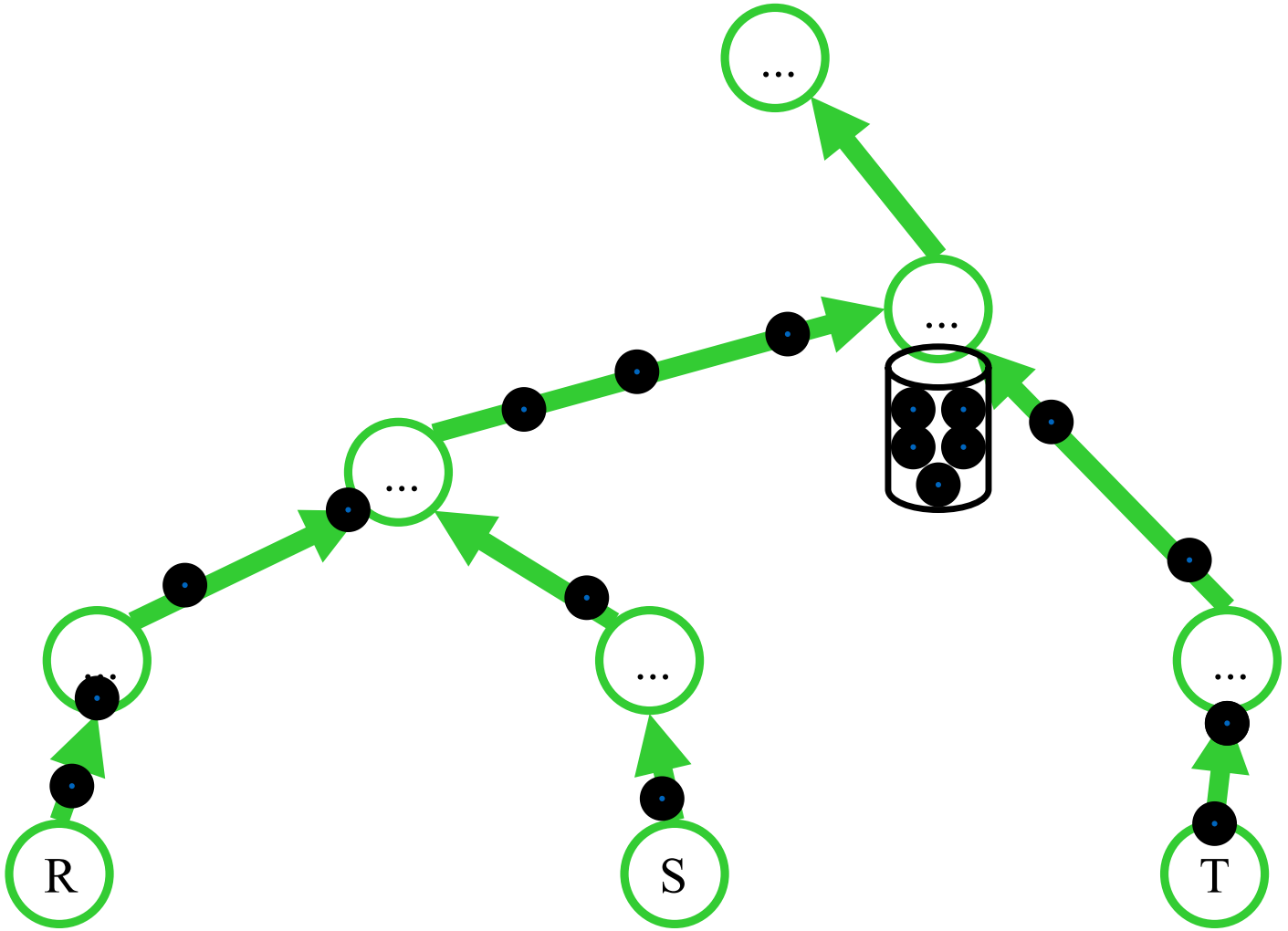
Pull-basierte Anfrageauswertung



Pipelining vs. Pipeline-Breaker



Pipelining vs. Pipeline-Breaker



Pipeline-Breaker

Unäre Operationen
- sort
- Duplikatelimination (unique,distinct)
- Aggregatoperationen (min,max,sum,...)

Binäre Operationen
- Mengendifferenz

Je nach Implementierung
- Join
- Union

Der natürliche Verbund zweier Relationen R und S

R		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_1
a_4	b_4	c_2
a_5	b_5	c_3
a_6	b_6	c_2
a_7	b_7	c_6

$\bowtie$

S		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2
c_4	d_3	e_3
c_5	d_4	e_4
c_7	d_5	e_5
c_8	d_6	e_6
c_5	d_7	e_7

$=$

$R \bowtie S$				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_3	b_3	c_1	d_1	e_1
a_5	b_5	c_3	d_2	e_2

Implementierung des Joins: Strategien

J1 nested (inner-outer) loop

- „brute force"-Algorithmus

```
foreach  r ∈ R
  foreach s ∈ S
    if s.B = r.A  then Res := Res ∪ (r ∘ s)
```

O(N²)-Komplexität

iterator NestedLoop$_p$

open

- Öffne die linke Eingabe

next

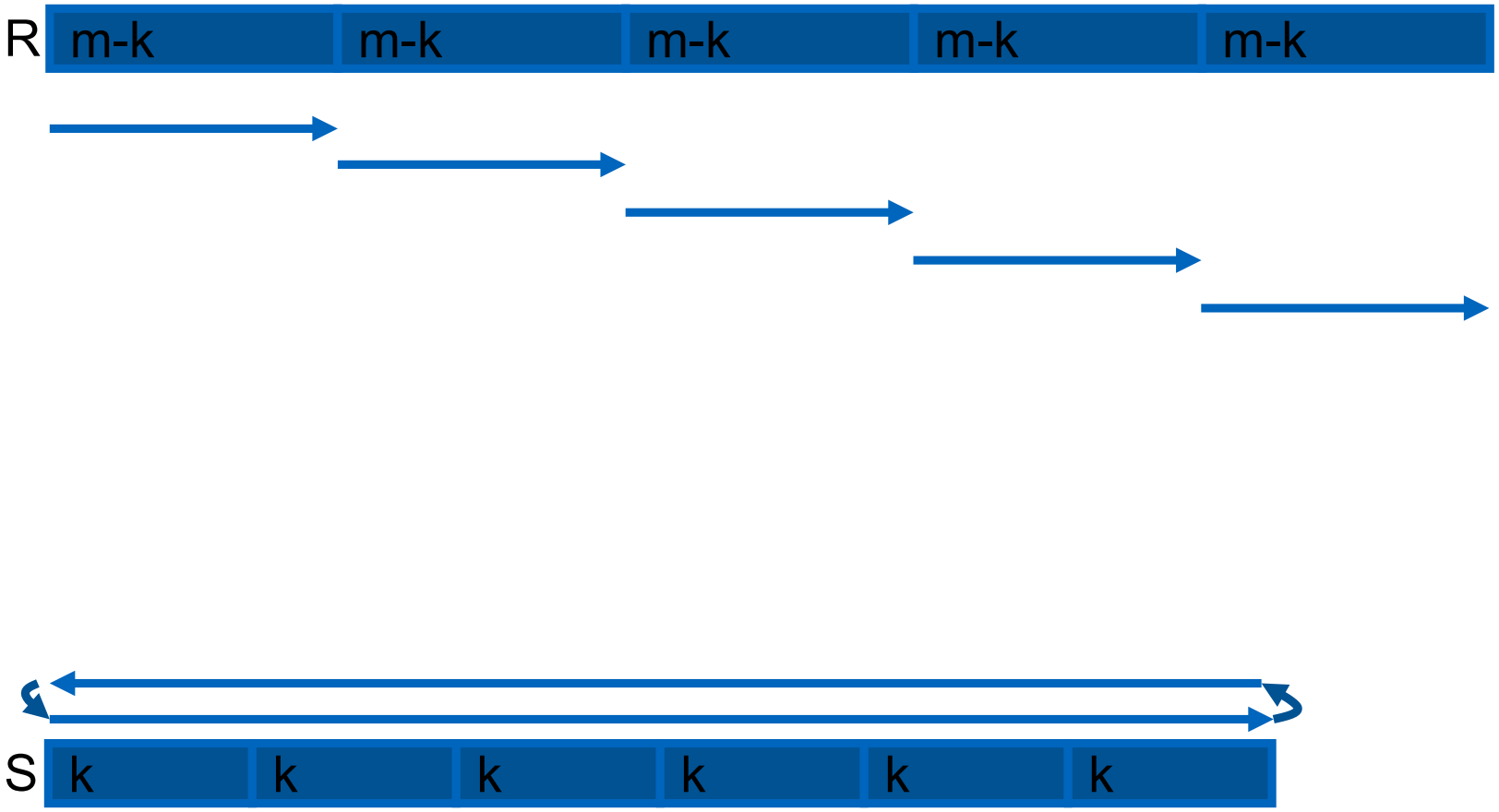
- Rechte Eingabe geschlossen?
  - Öffne sie
- Fordere rechts solange Tupel an, bis Bedingung p erfüllt ist
- Sollte zwischendurch rechte Eingabe erschöpft sein
  - Schließe rechte Eingabe
  - Fordere nächstes Tupel der linken Eingabe an
  - Starte **next** neu
- Gib den Verbund von aktuellem linken und aktuellem rechte Tupel zurück

close

- Schließe beide Eingabequellen

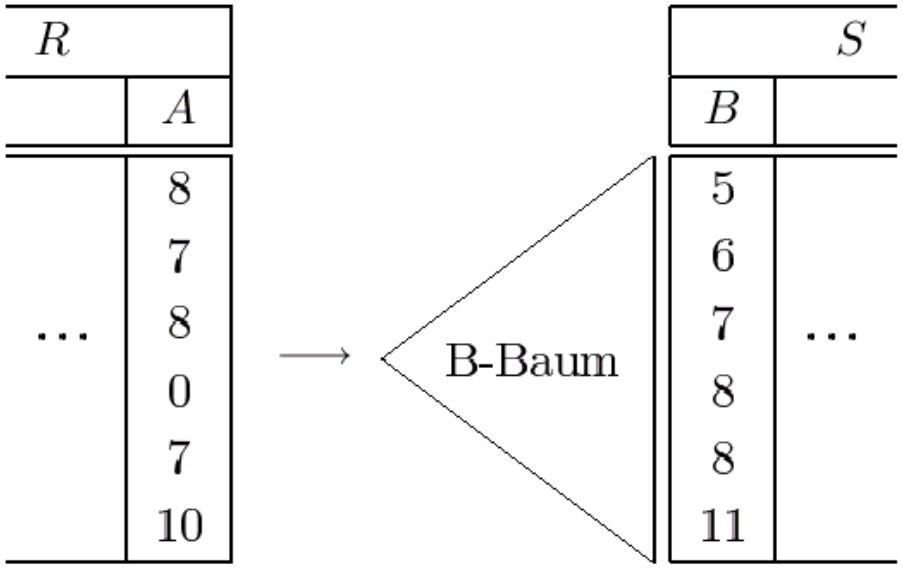
Implementierung des Joins: Strategien

Block-Nested Loop Algorithmus



Index-Join

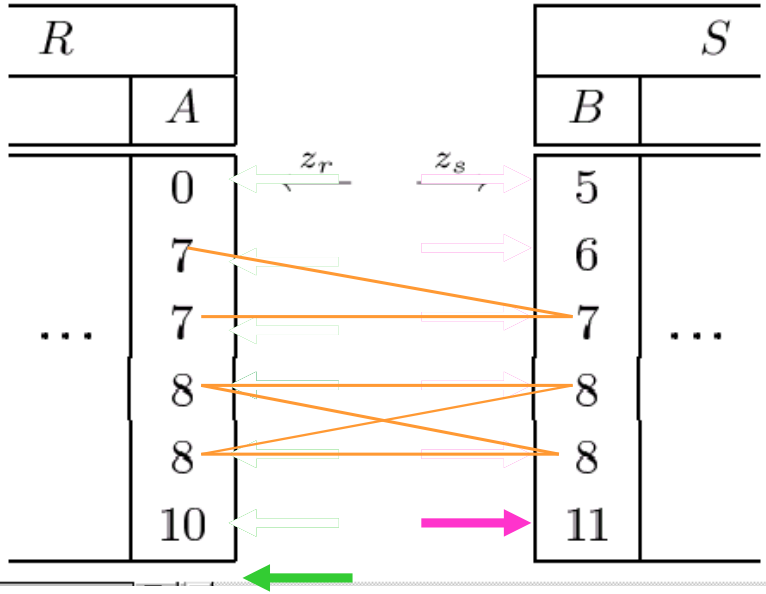
Beispiel:



Der Merge-Join

- Voraussetzung: R und S sind sortiert (notfalls vorher sortieren)

Beispiel:



Implementierung des Joins: Strategien

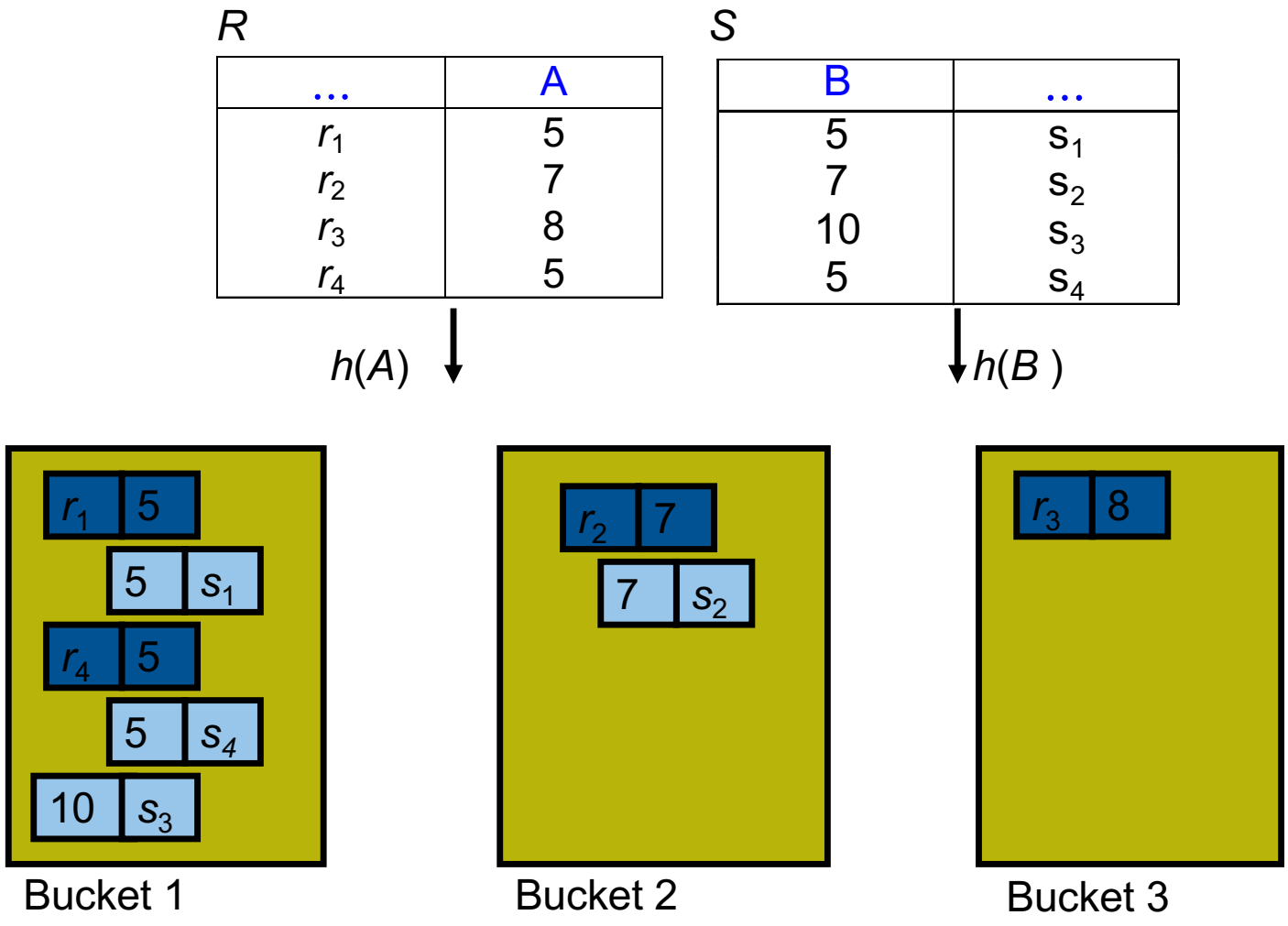
J4 Hash-Join

R und *S* werden mittels der gleichen Hashfunktion *h* – angewendet auf *R.A* und *S.B* – auf (dieselben) Hash-Buckets abgebildet

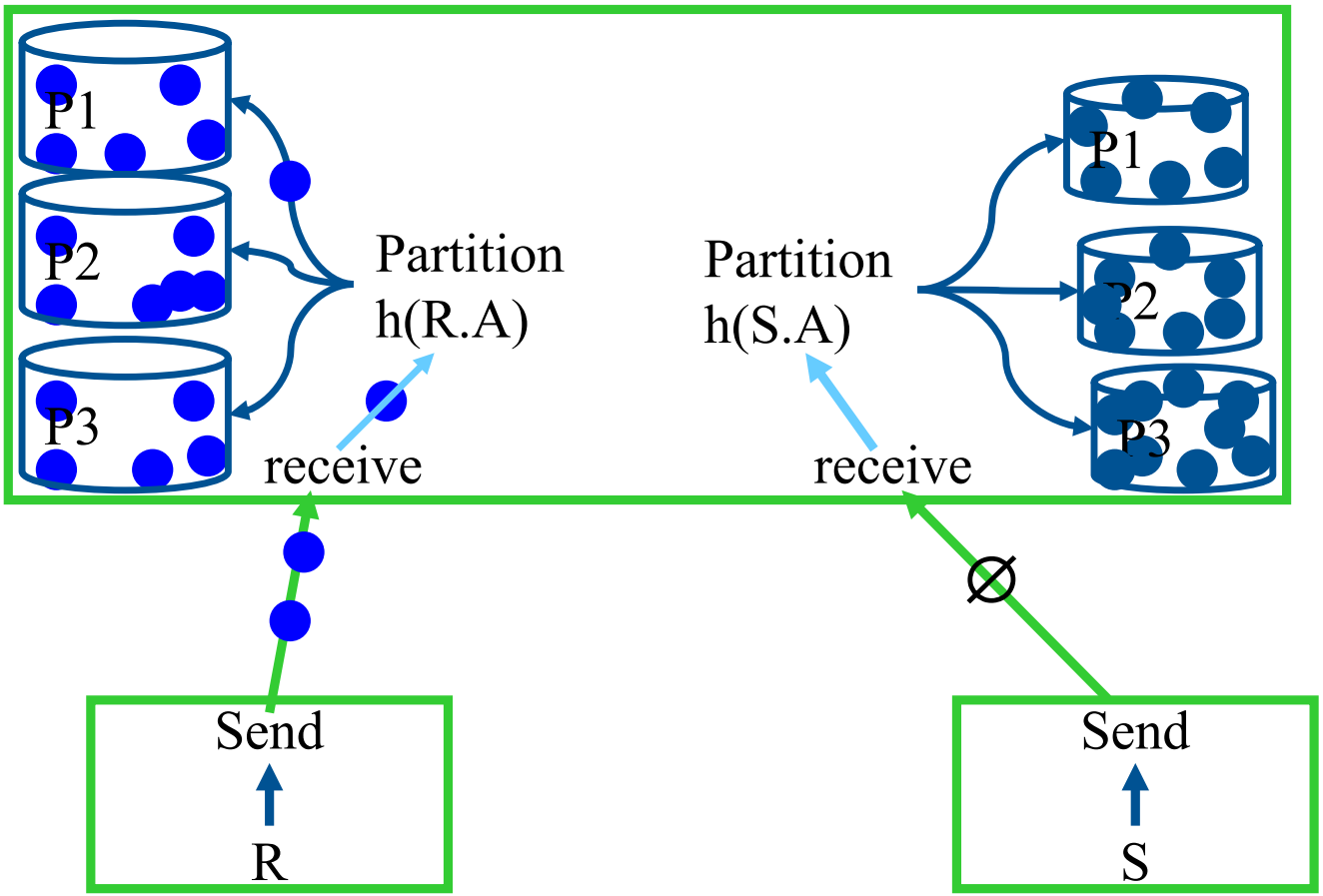
Hash-Buckets sind i.Allg. auf Hintergrundspeicher (abhängig von der Größe der Relationen)

Zu „joinende" Tupel befinden sich dann im selben Bucket
Wird (nach praktischen Tests) nur vom Merge-Join „geschlagen", wenn die Relationen schon vorsortiert sind

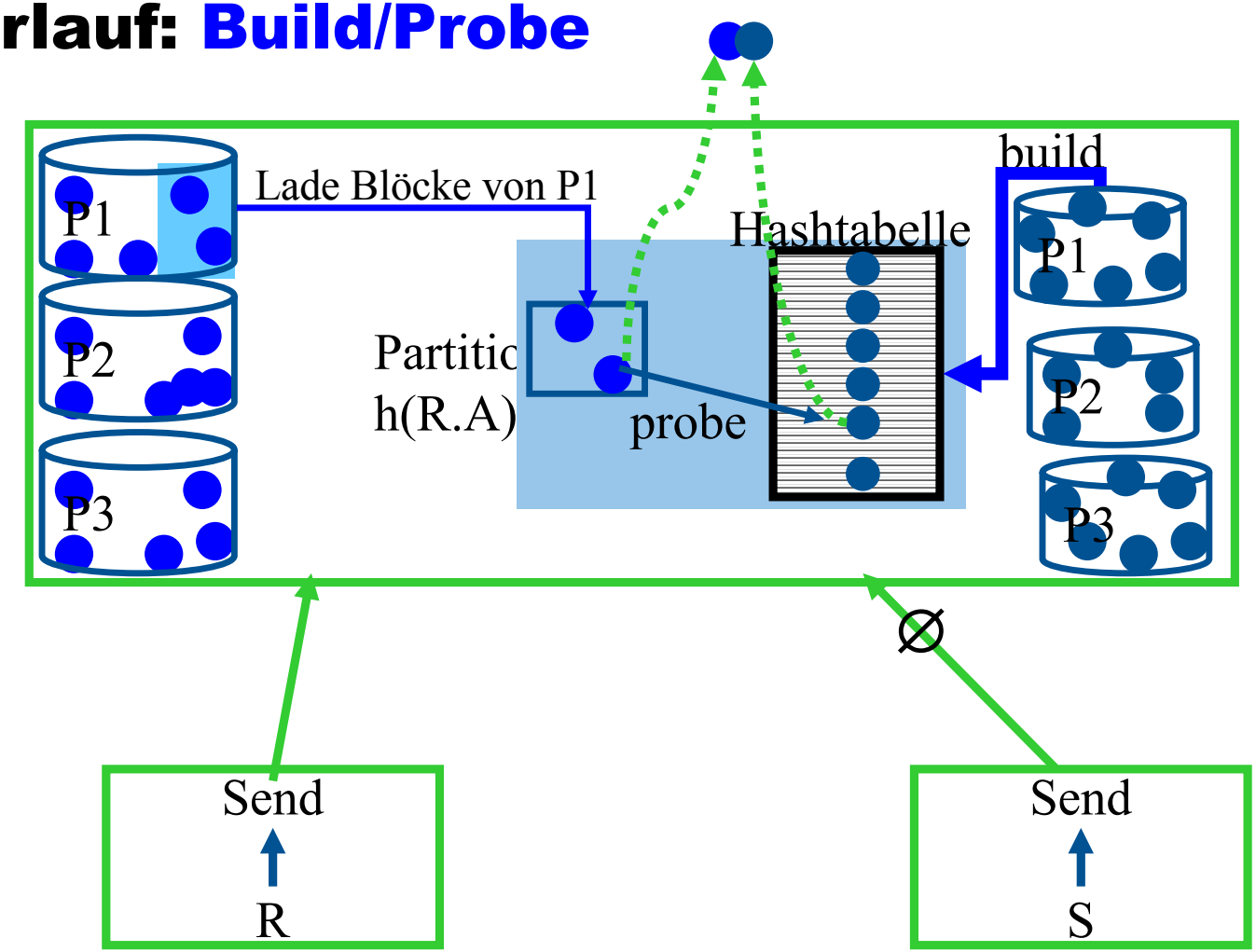
Konzeptuelle Idee des Hash-Joins



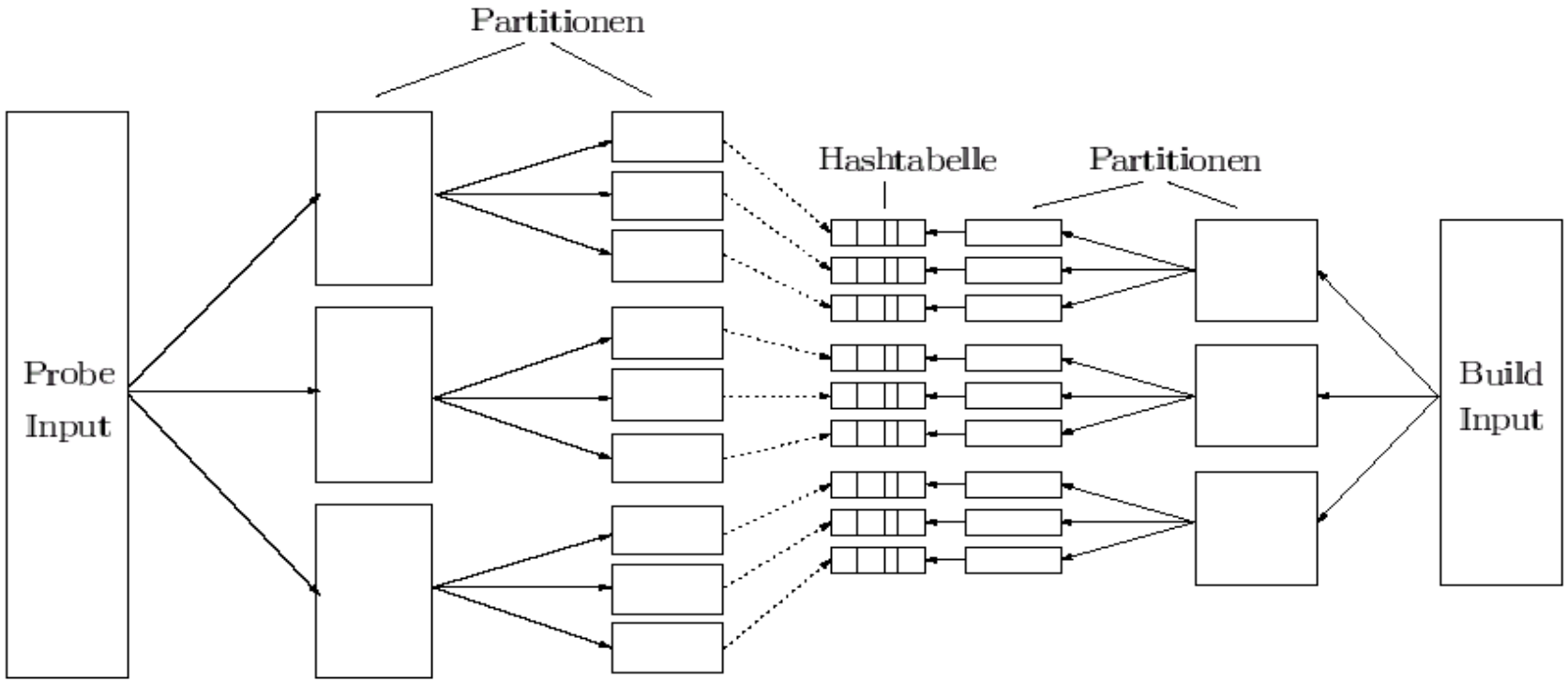
„Normaler“ blockierender Hash-Join mit Überlauf: Partitionieren



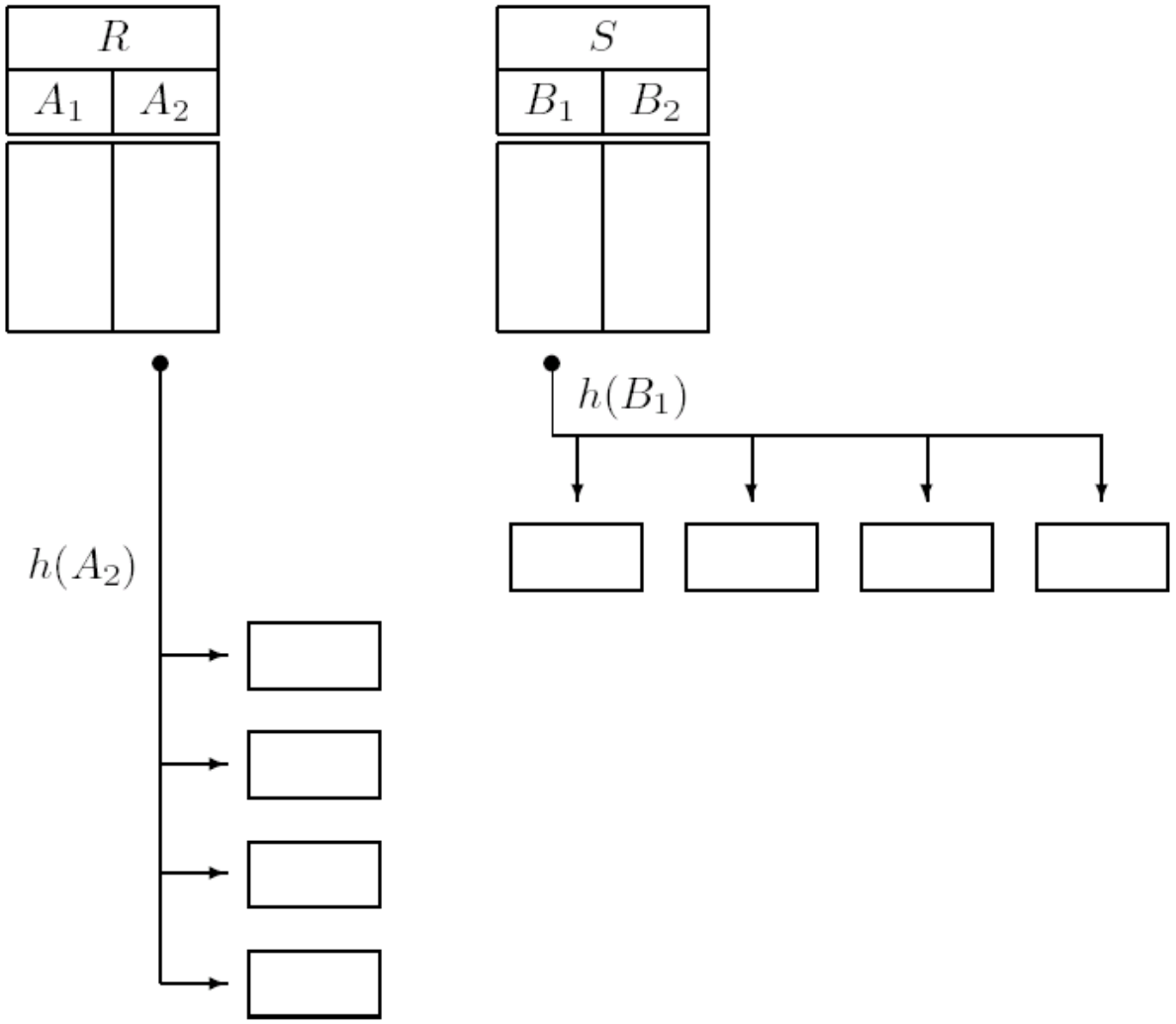
„Normaler“ blockierender Hash-Join mit Überlauf: Build/Probe

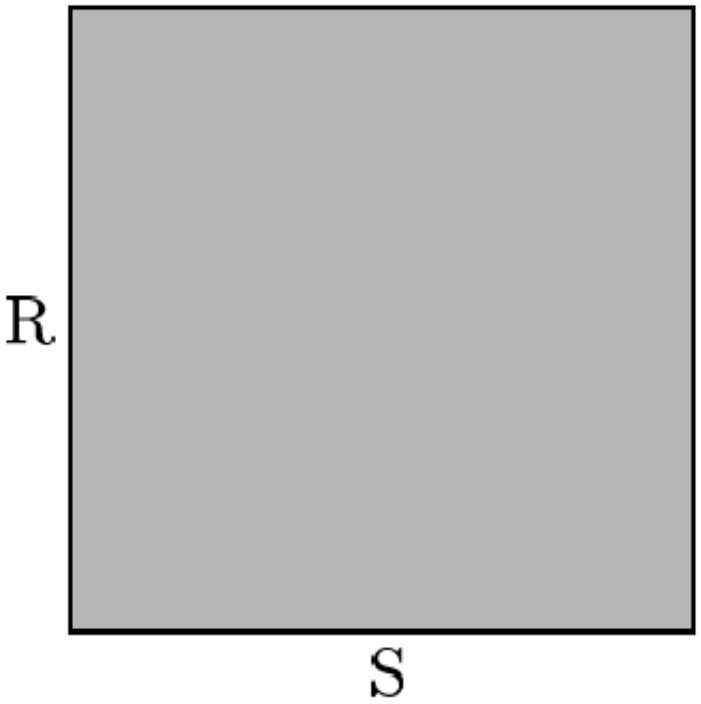


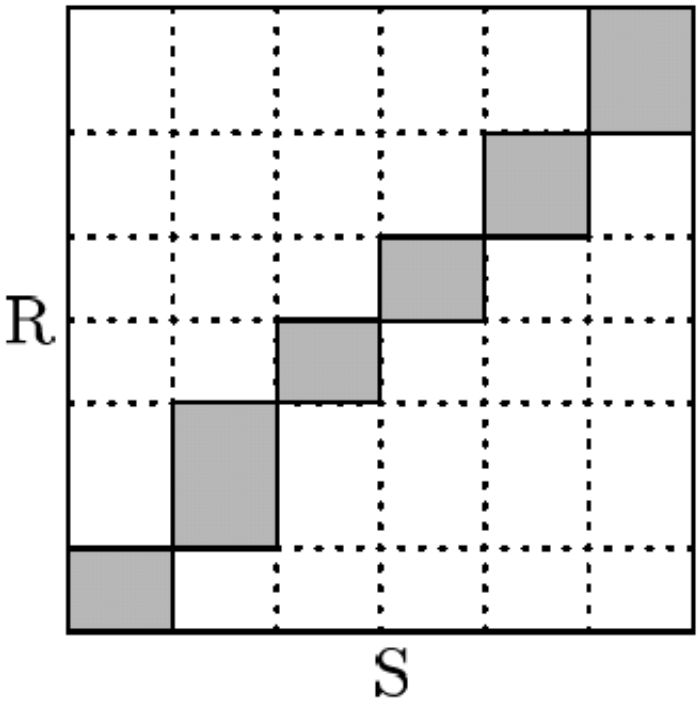
Partitionierung von Relationen



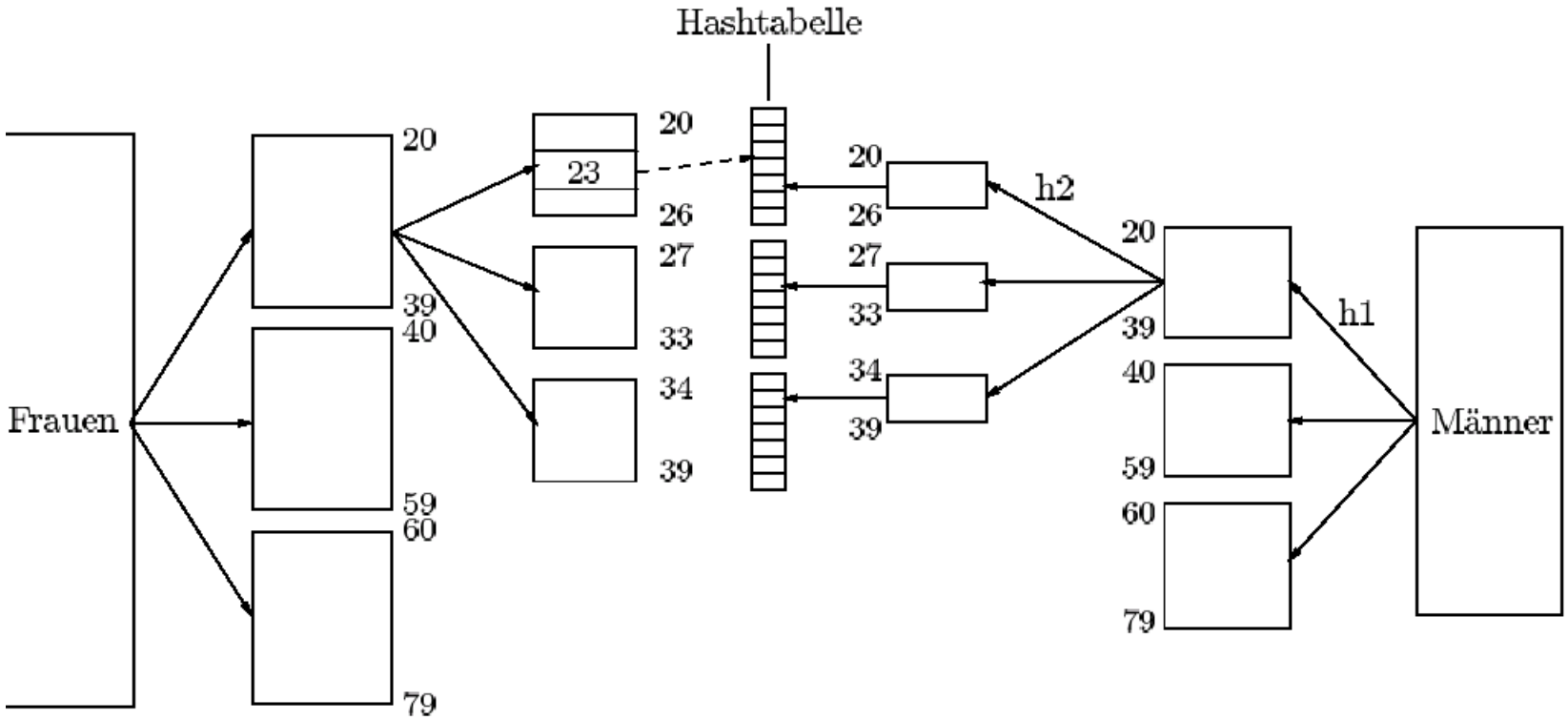
Vergleich der Tupel in der "Diagonalen"



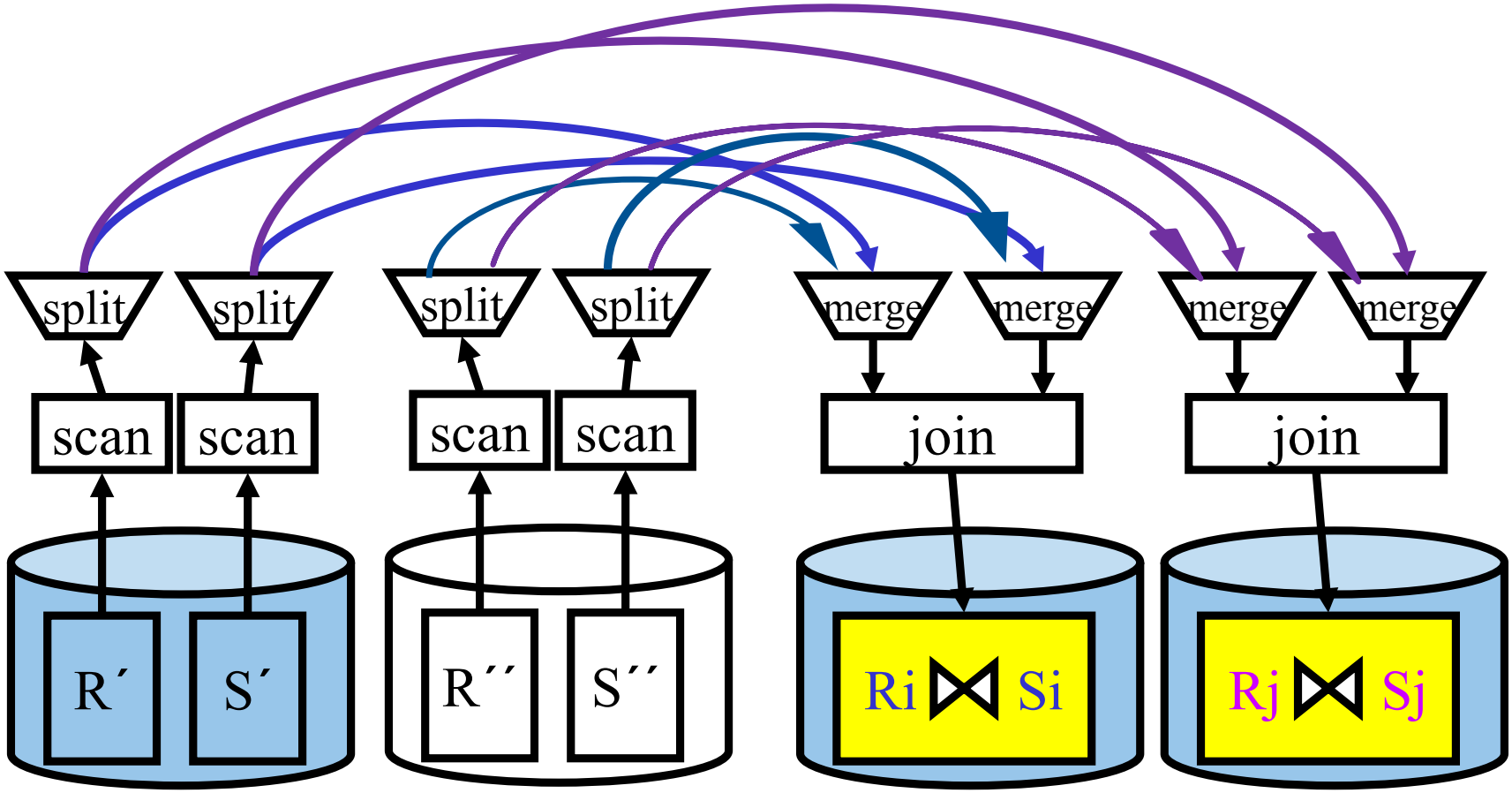

R
S


R
S

Demonstration der Partitionierung



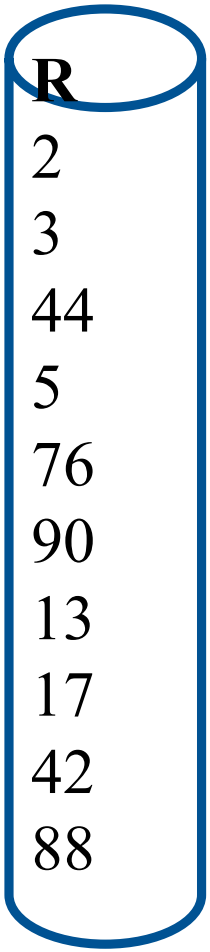
Parallele Anfragebearbeitung:
Hash Join



Paralleler Hash Join – im Detail

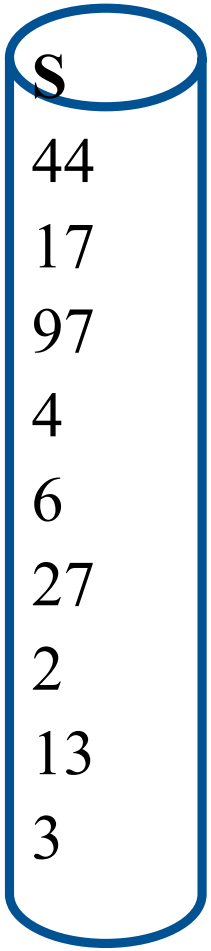
1. An jeder Station werden mittels Hash-Funktion h1 die jeweiligen Partitionen von R und S in R1,...,Rk und S1,...,Sk zerlegt
   - h1 muss so gewählt werden, dass alle R_i´s aller Stationen in den Hauptspeicher passen
2. Für alle 1 <= i <= n: Berechne jetzt den Join von Ri mit Si wie folgt
   a. Wende eine weitere Hash-Funktion h2 an, um Ri auf die l Stationen zu verteilen
      - Sende Tupel t an Station h2(t)
   b. Eintreffende Ri-Tupel werden in die Hash-Tabelle an der jeweiligen Station eingefügt
   c. Sobald alle Tupel aus Ri „verschickt" sind, wird h2 auf S angewendet und Tupel t an Station h2(t) geschickt
   d. Sobald ein Si-Tupel eintrifft, werden in der Ri-Hashtabelle seine Joinpartner ermittelt.

Mengendurchschnitt (~Join) mit einem Hash/Partitionierungs-Algorithmus

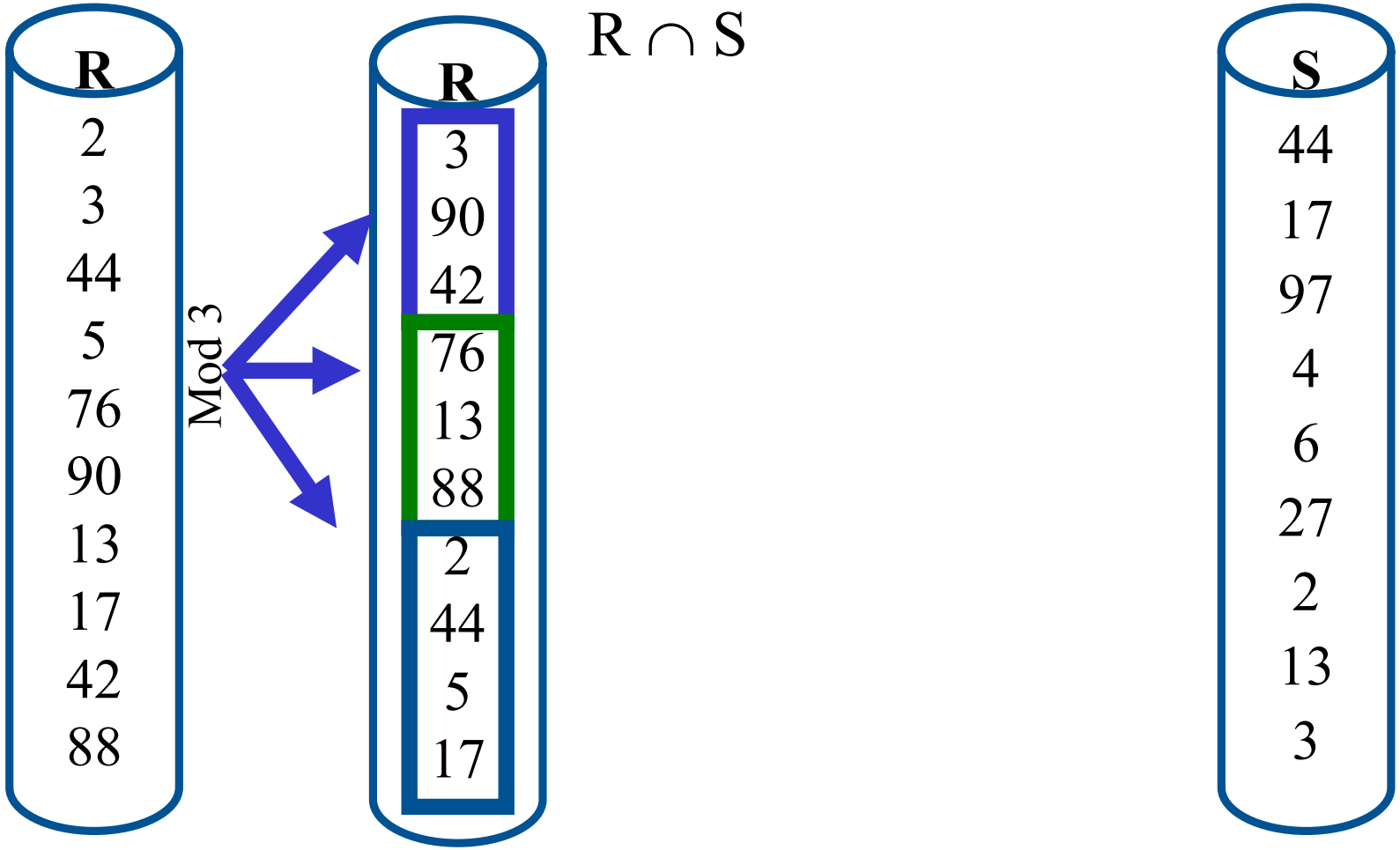


$R \cap S$

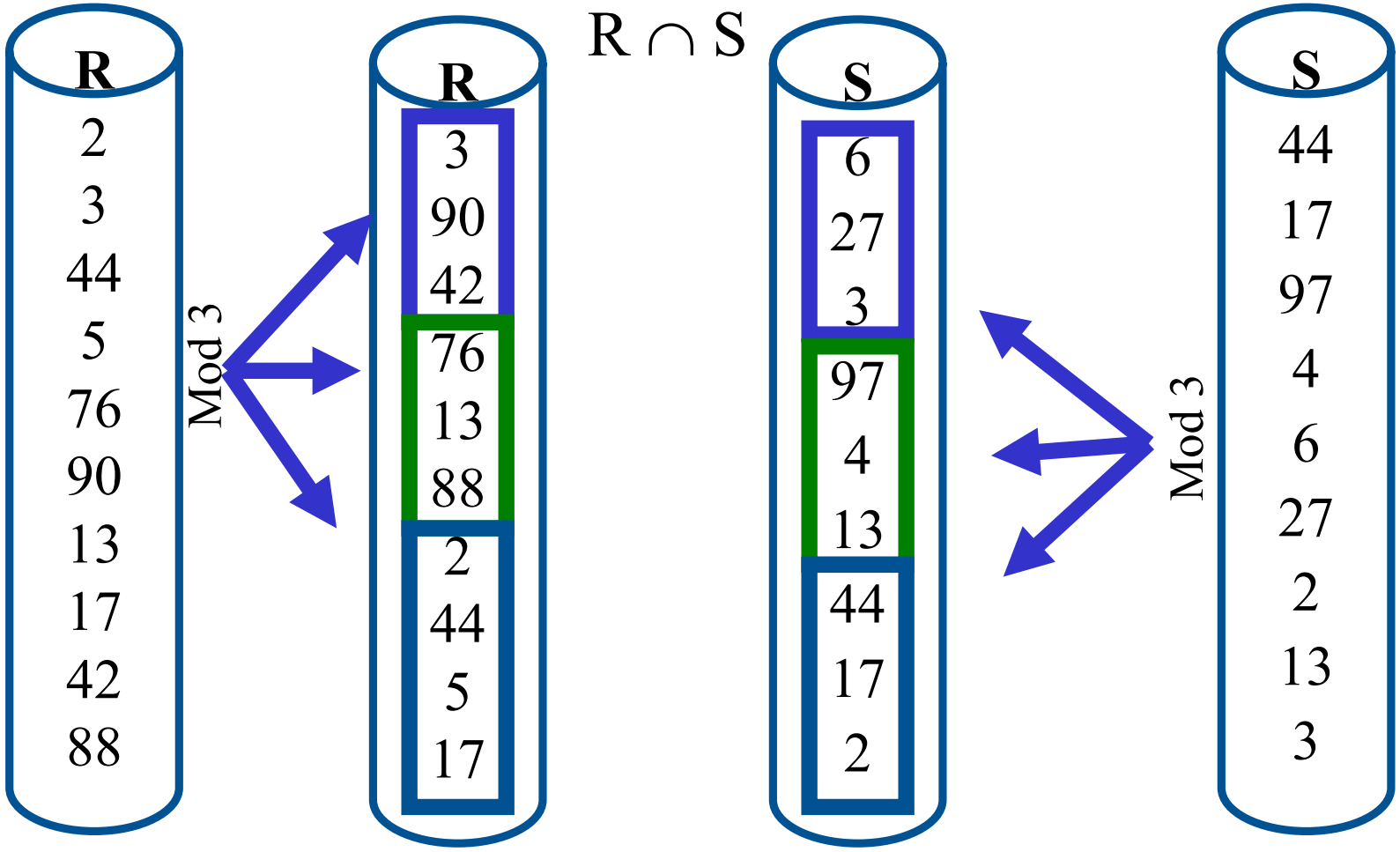
- Nested Loop: $O(N^2)$
- Sortieren: $O(N \log N)$
- Partitionieren und Hashing



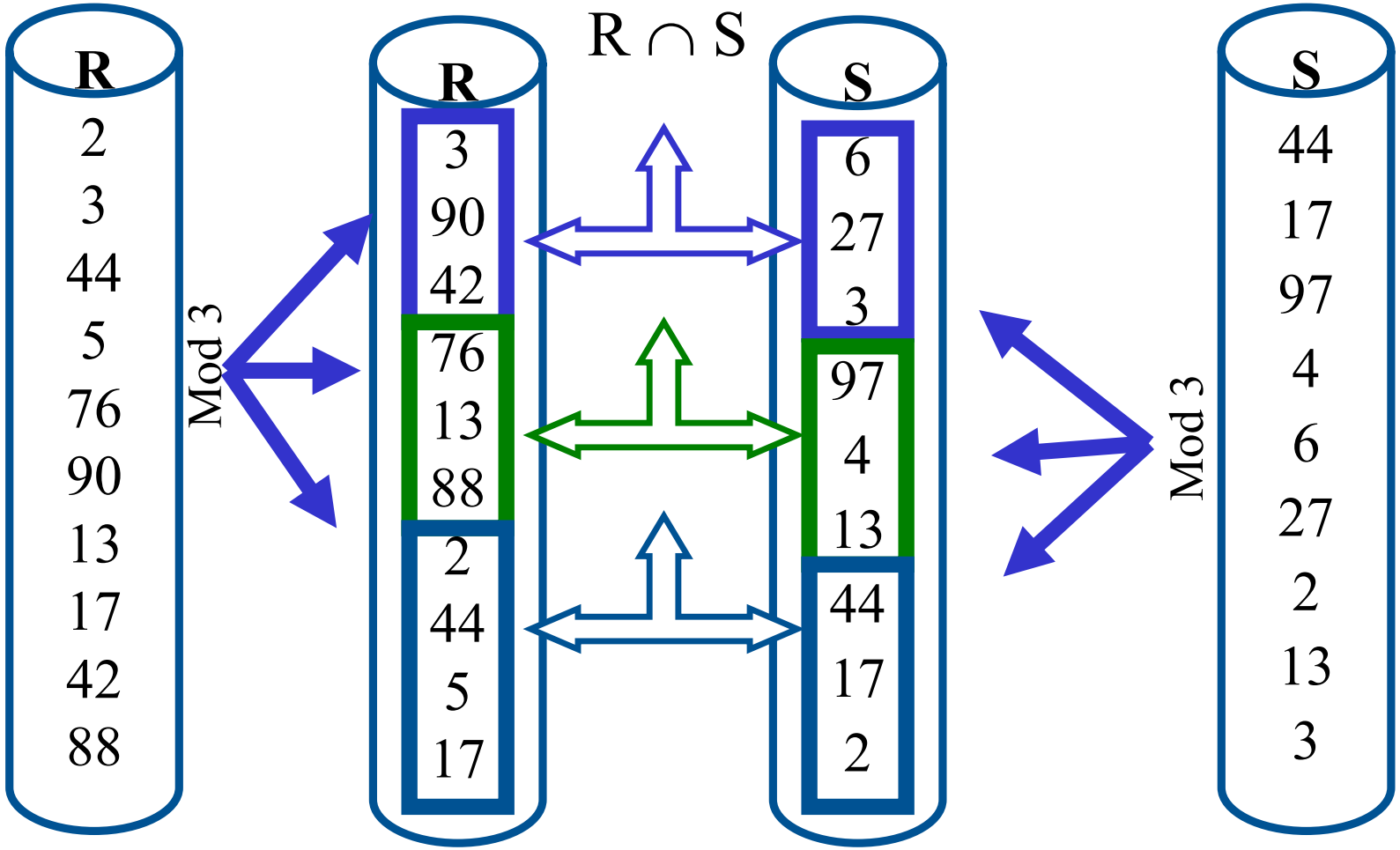
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



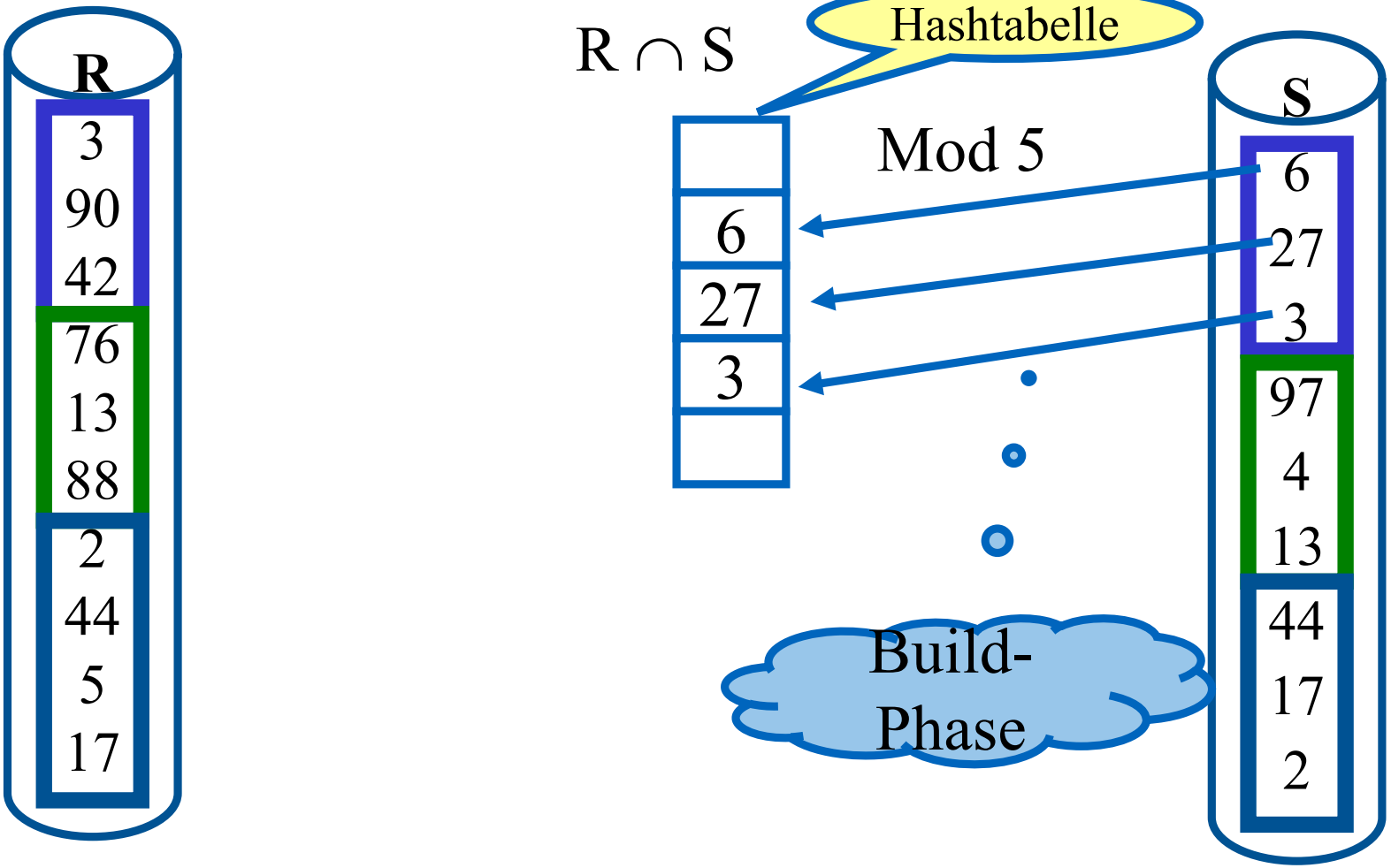
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



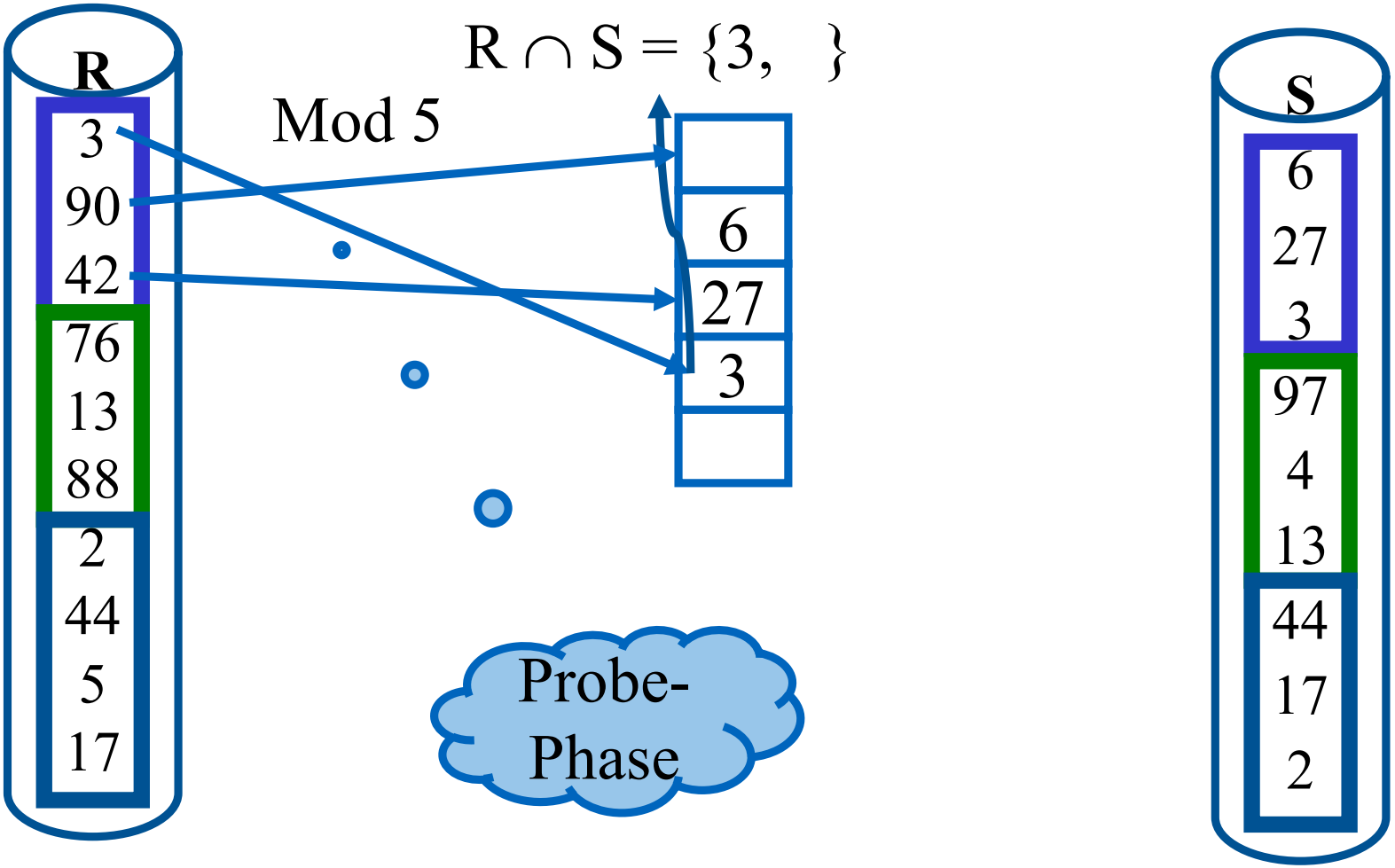
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



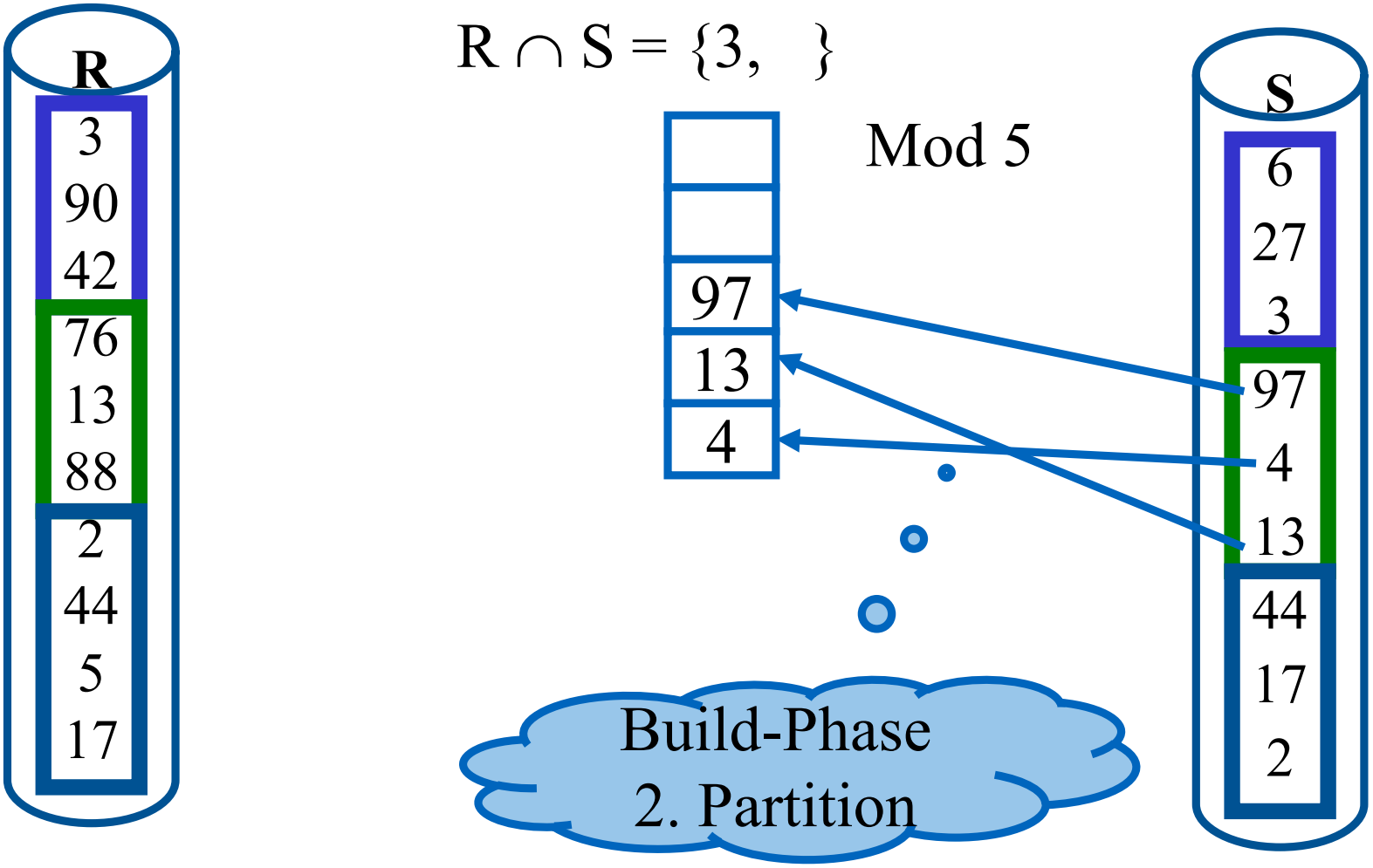
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



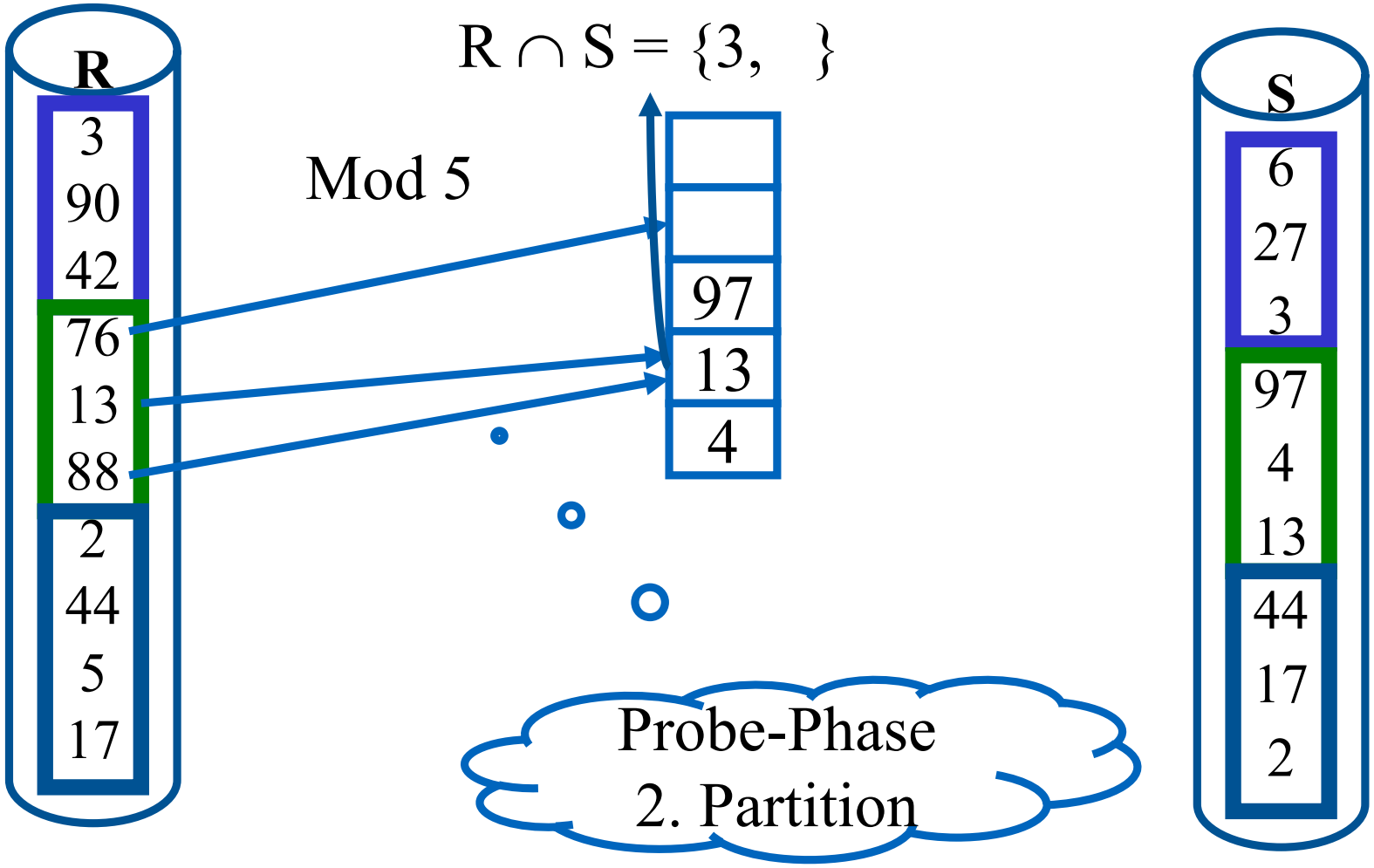
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



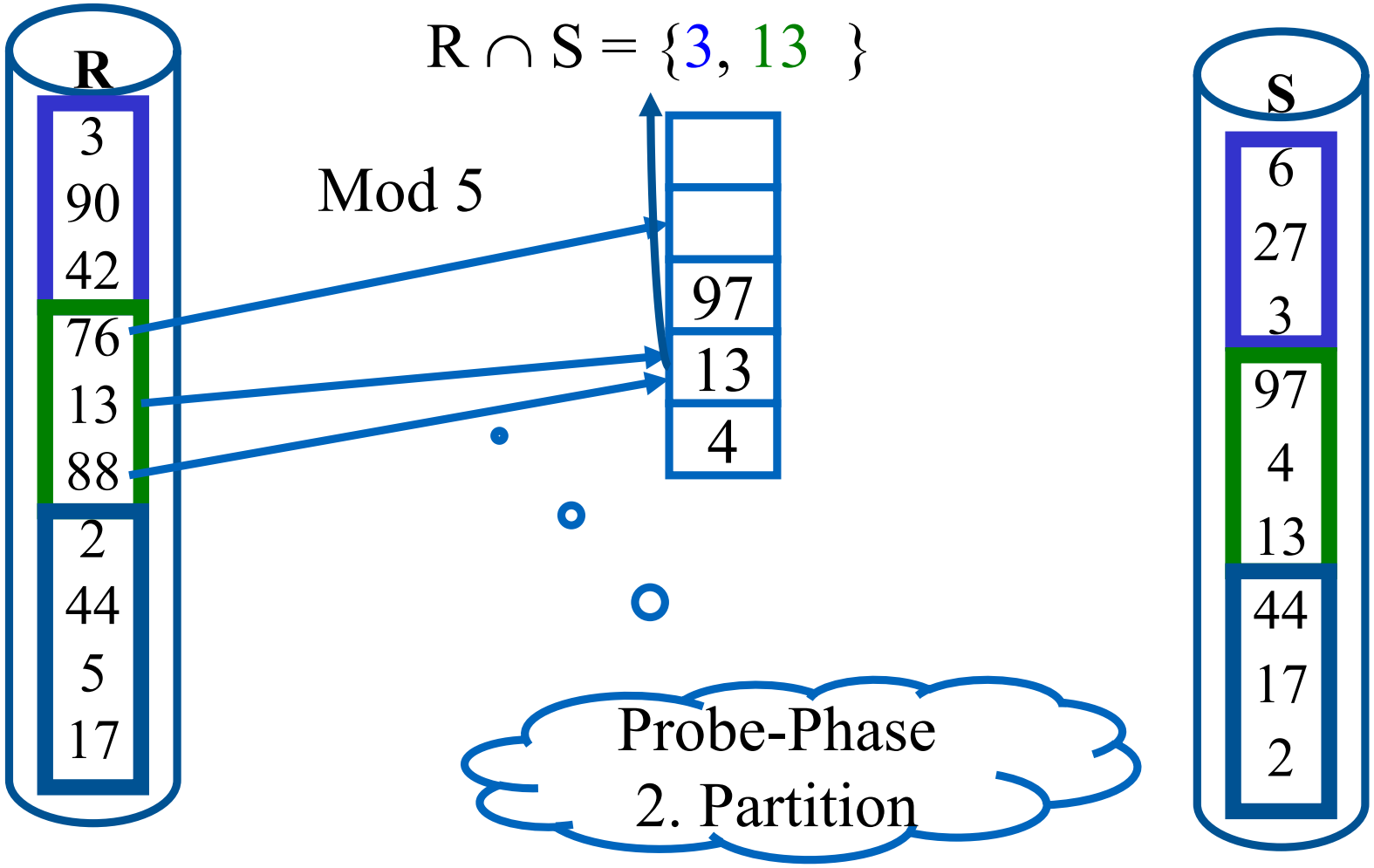
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

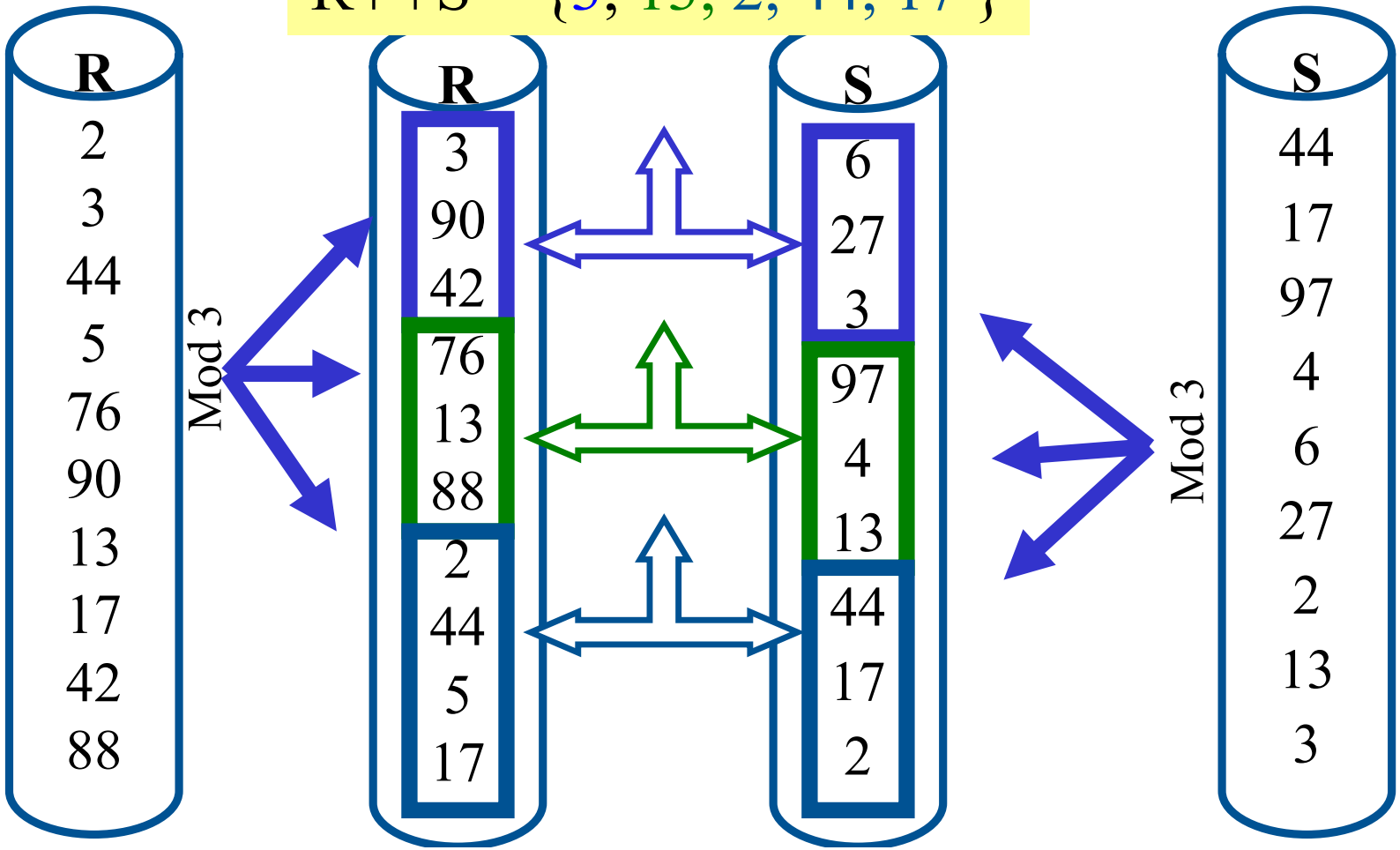


Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

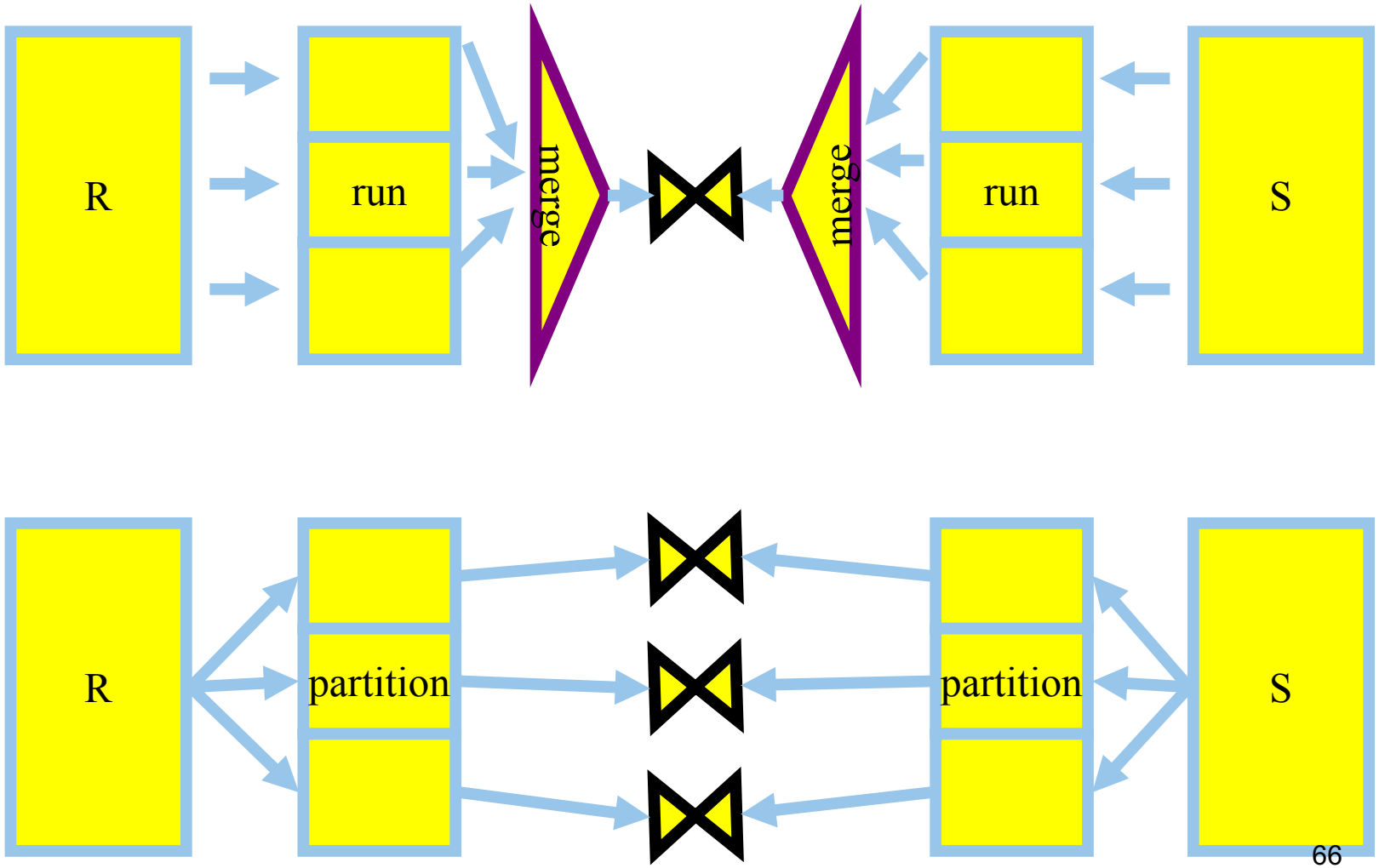


Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

$R \cap S = \{3, 13, 2, 44, 17\}$



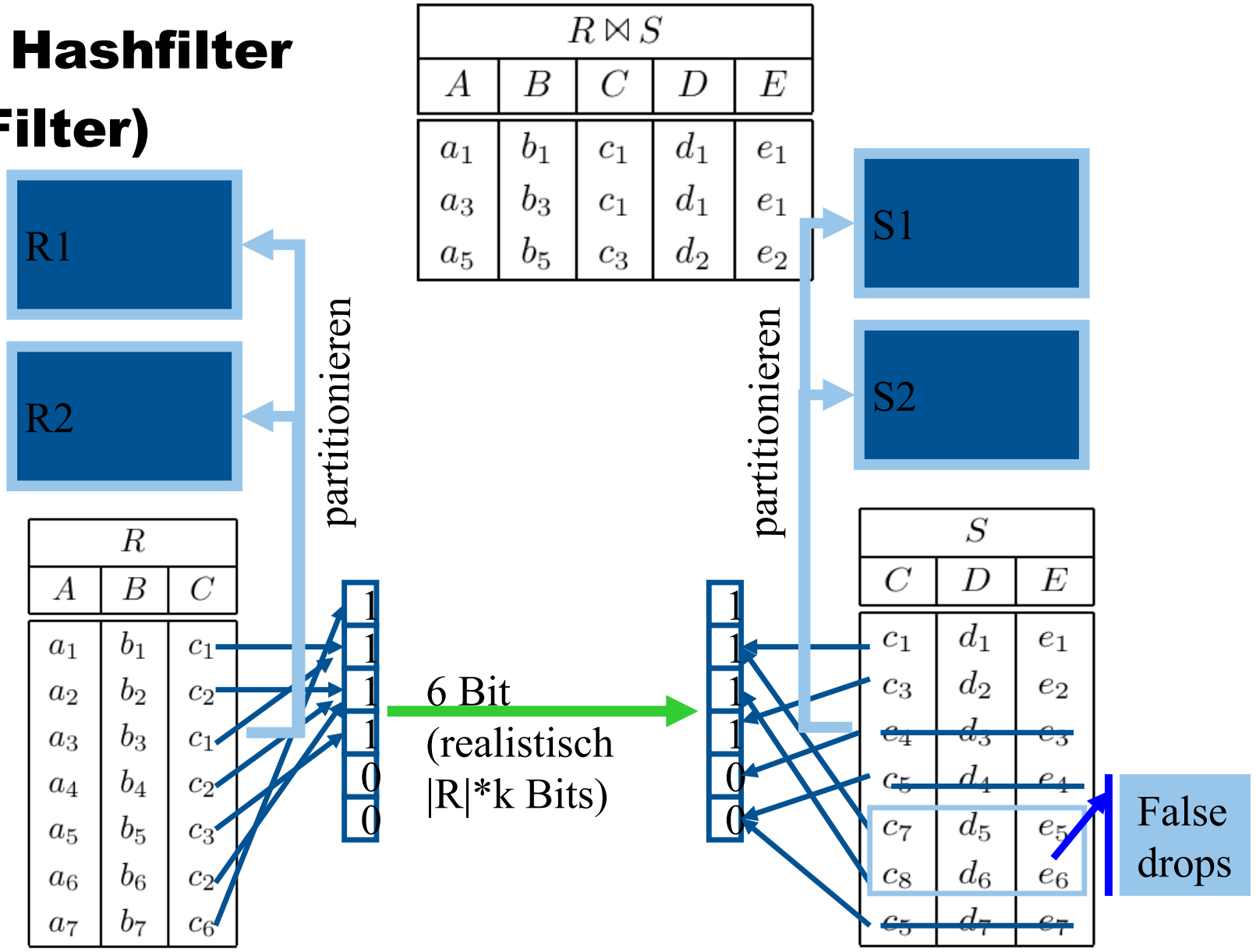
Vergleich: Sort/Merge-Join versus Hash-Join



Prallelausführung von Aggregat-Operationen

Min: Min(R.A) = Min (Min(R1.A), ... , Min(Rn.A))
Max: analog
Sum: Sum(R.A) = Sum (Sum(R1.a), ..., Sum(Rn.A))
Count: analog
Avg: man muß die Summe und die Kardinalitäten der Teilrelationen kennen; aber vorsicht bei Null-Werten!
Avg(R.A) = Sum(R.A) / Count(R) gilt nur wenn A keine Nullwerte enthält.

Join mit Hashfilter (Bloom-Filter)



Join mit Hashfilter

(False Drop Abschätzung)

Wahrscheinlichkeit, dass ein bestimmtes Bit j gesetzt ist

- W. dass ein bestimmtes r∈R das Bit setzt: 1/b
- W. dass kein r∈R das Bit setzt: $(1-1/b)^{|R|}$
- W. dass ein r∈R das Bit gesetzt hat: $1- (1-1/b)^{|R|}$

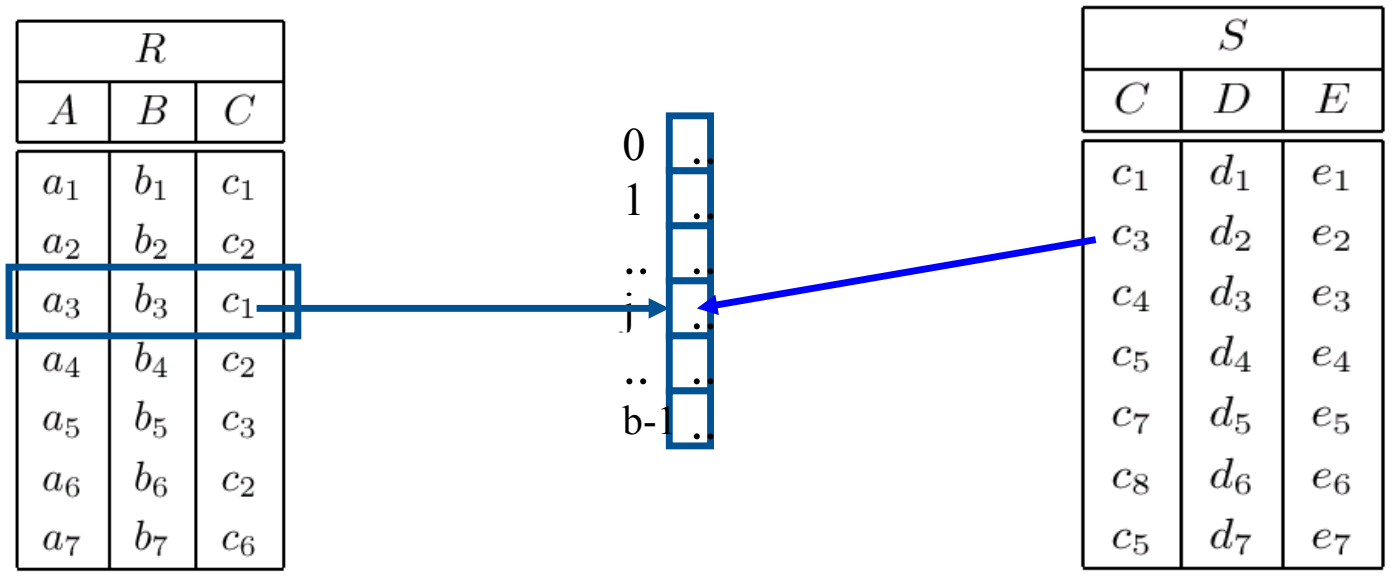


Illustration: Externes Sortieren

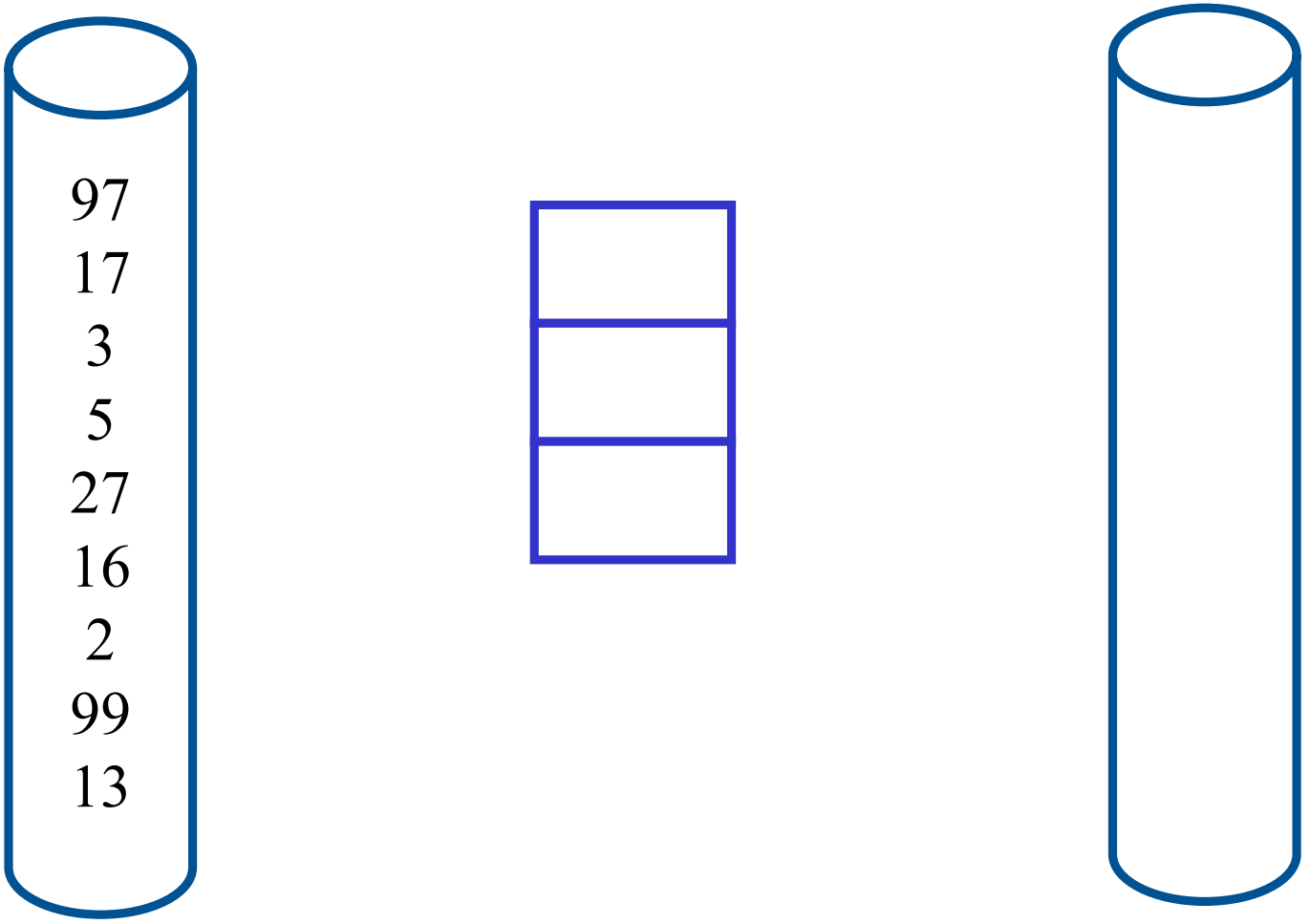


Illustration: Externes Sortieren

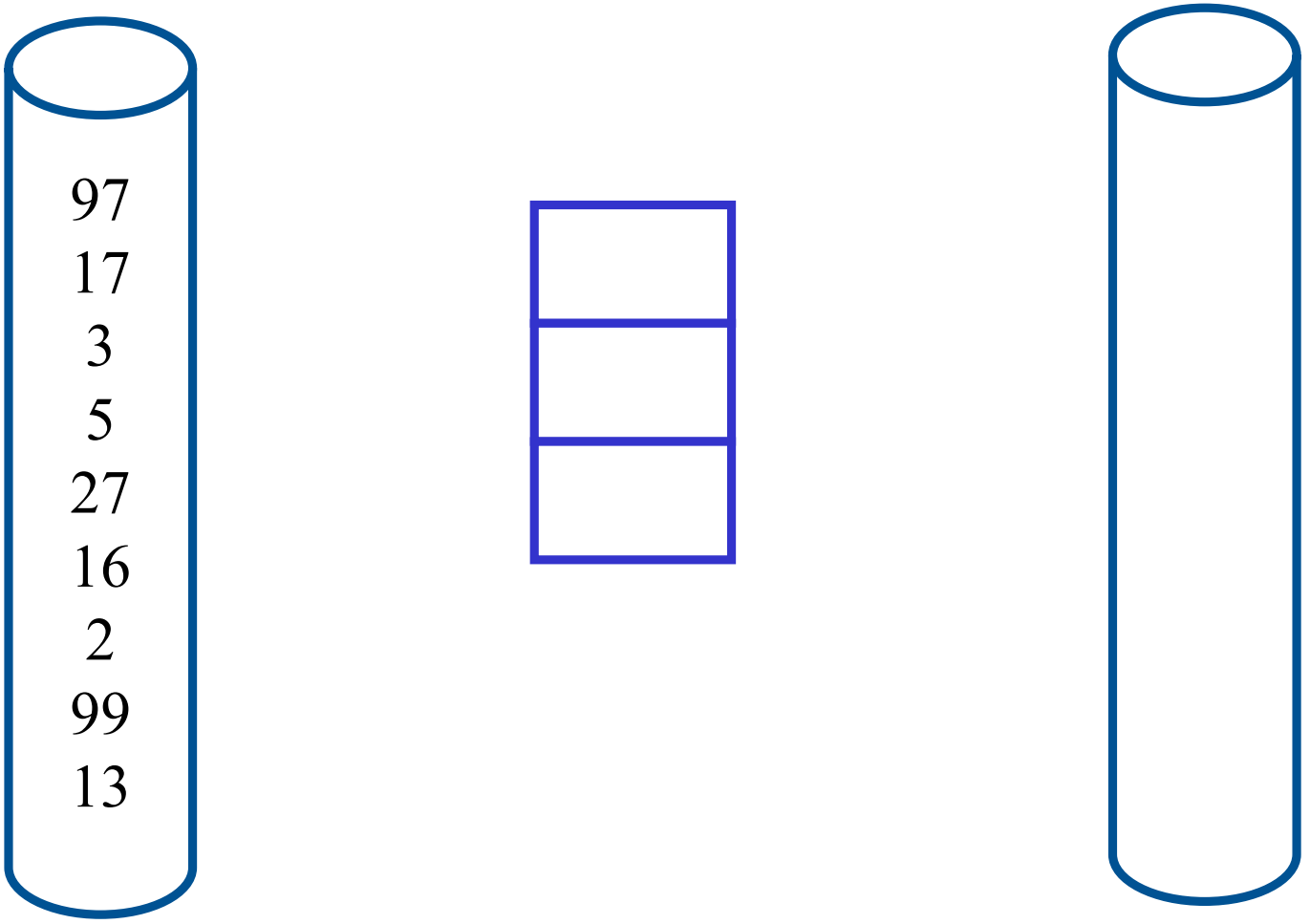


Illustration: Externes Sortieren

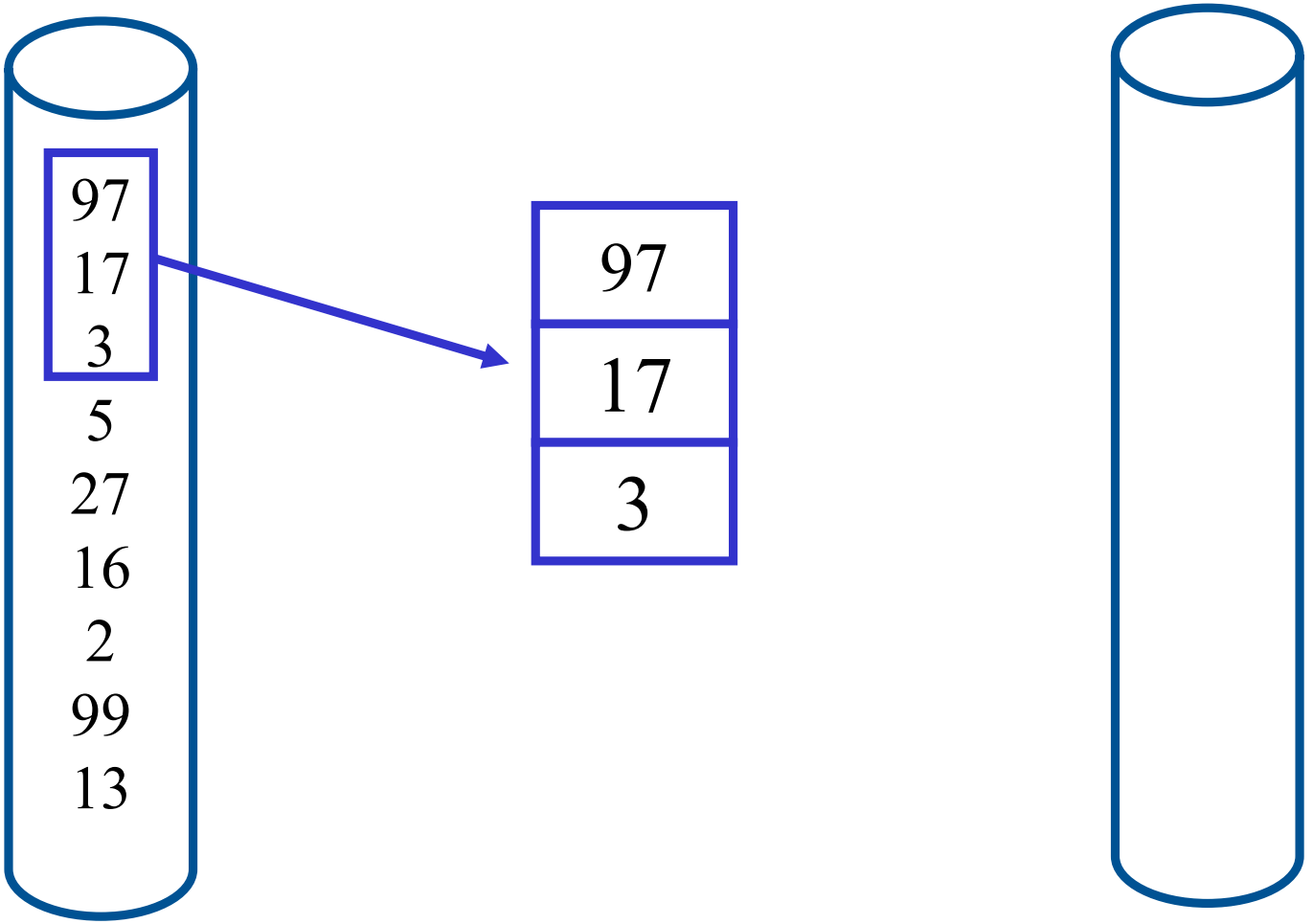


Illustration: Externes Sortieren

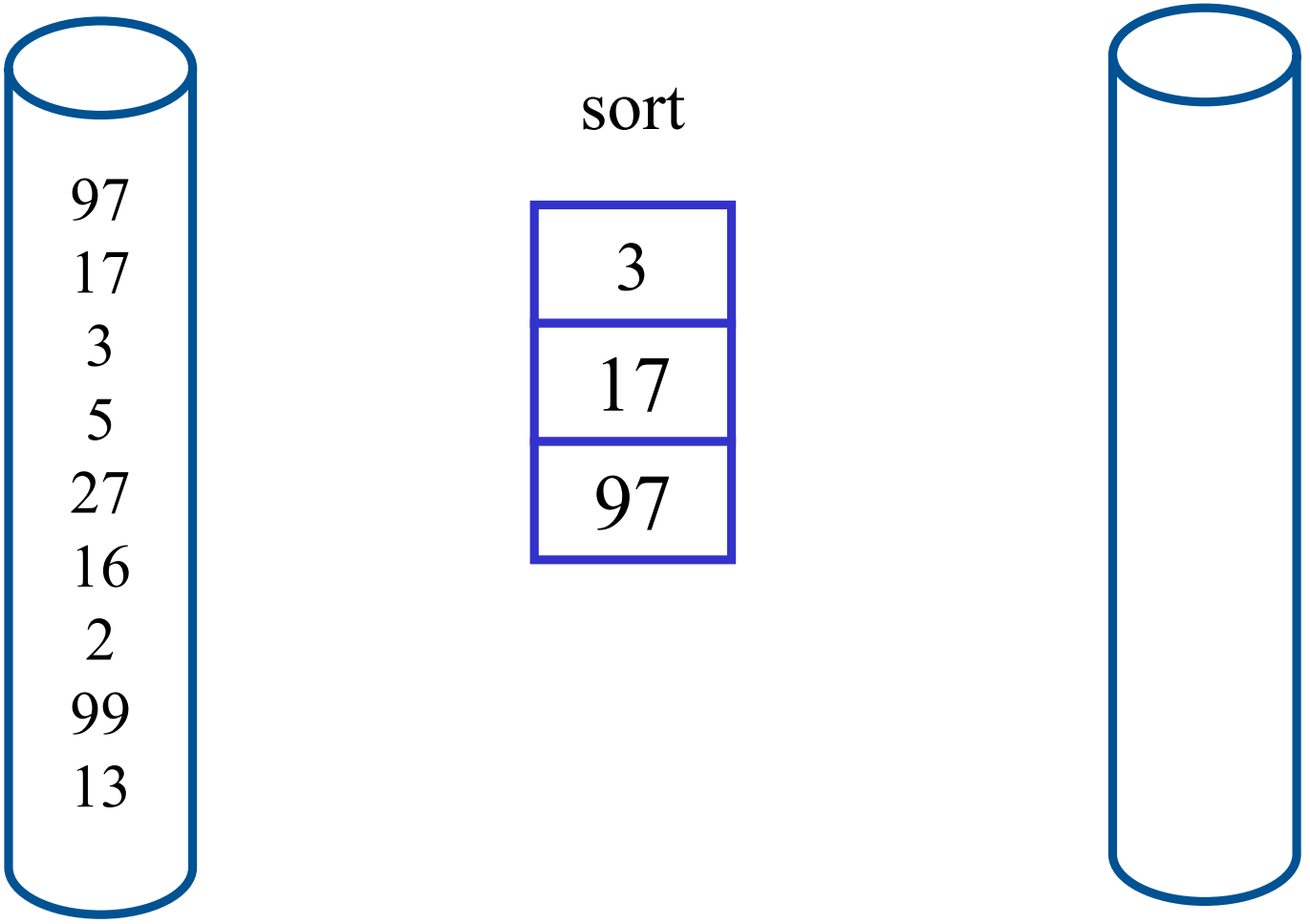


Illustration: Externes Sortieren

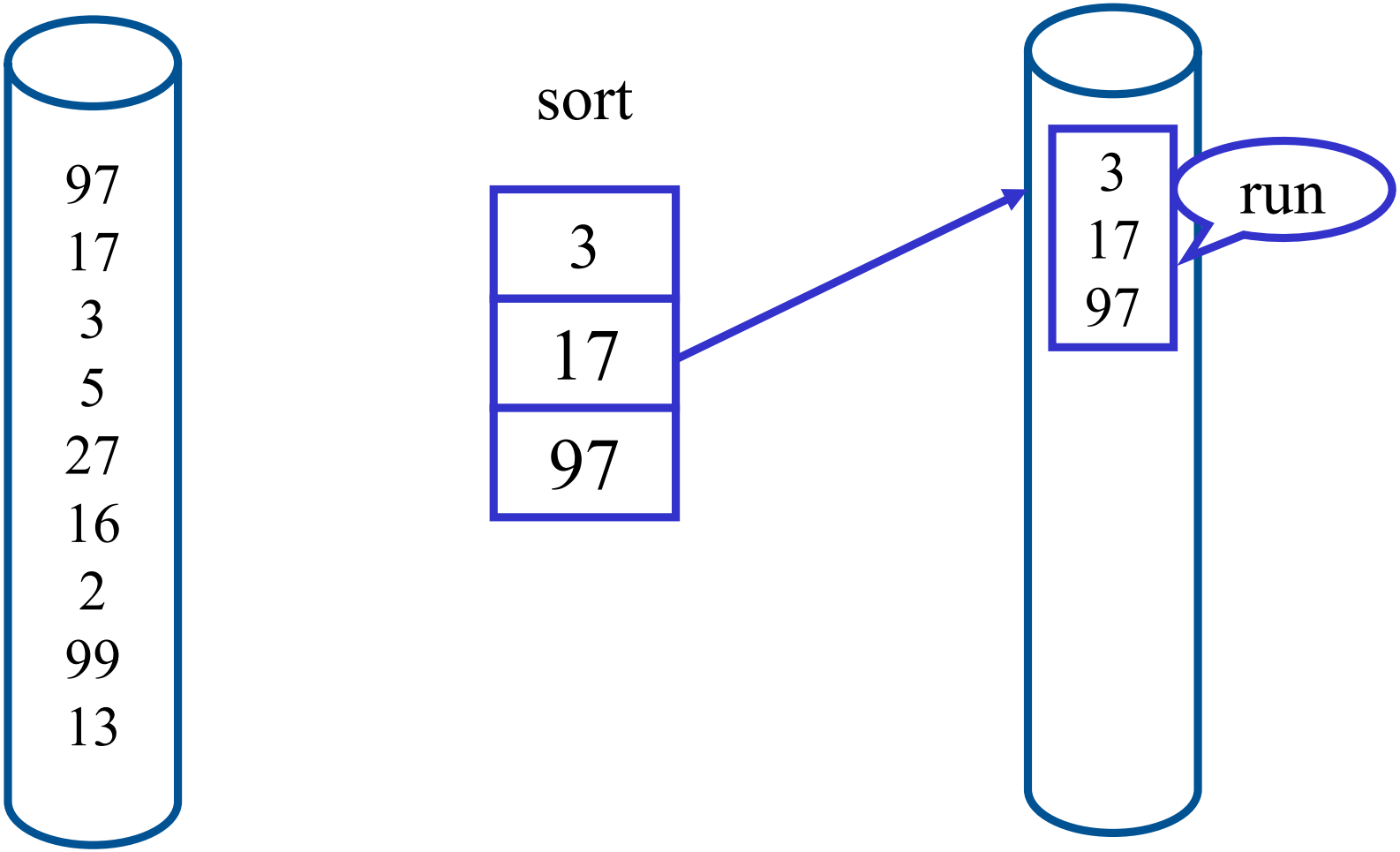


Illustration: Externes Sortieren

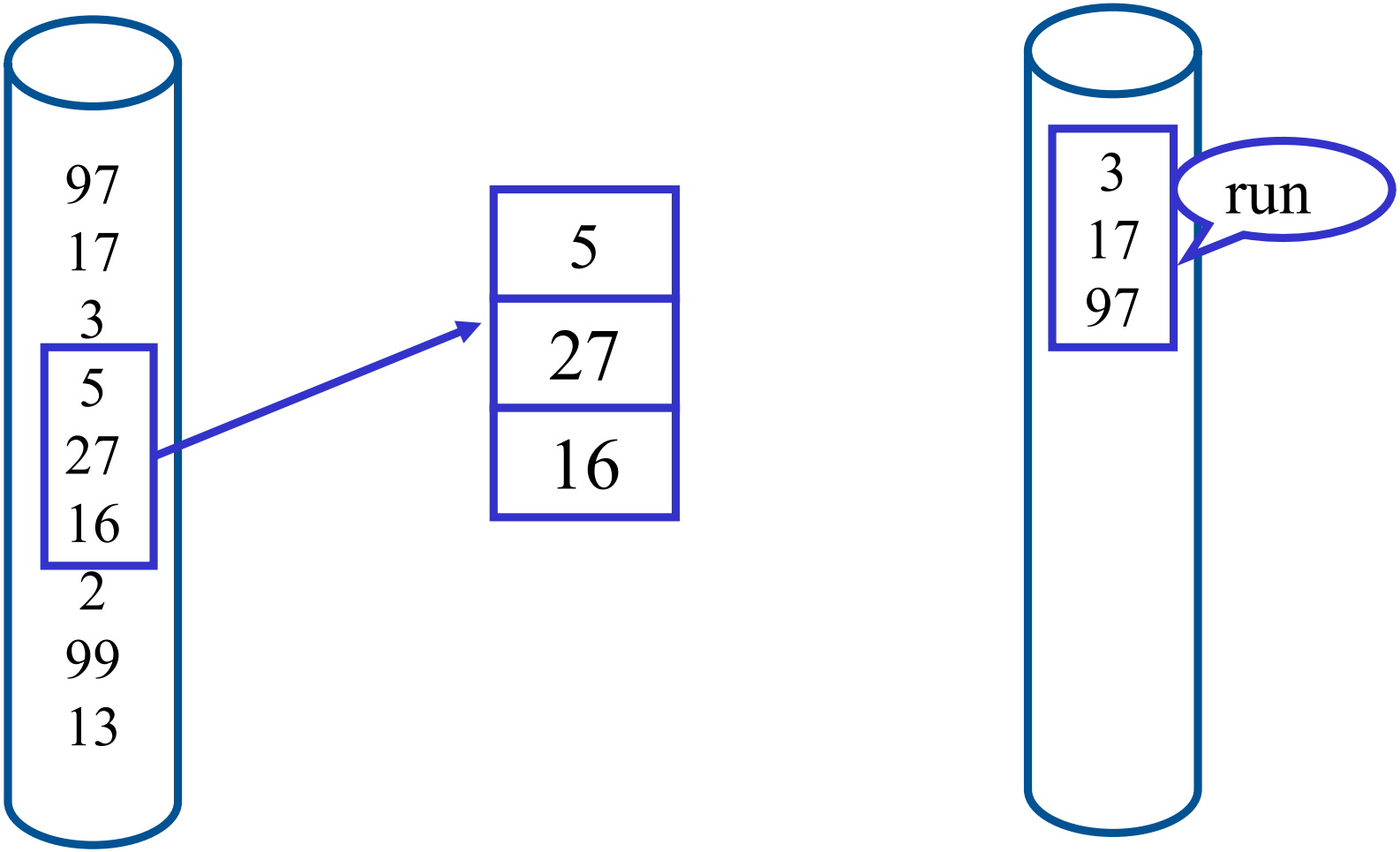


Illustration: Externes Sortieren

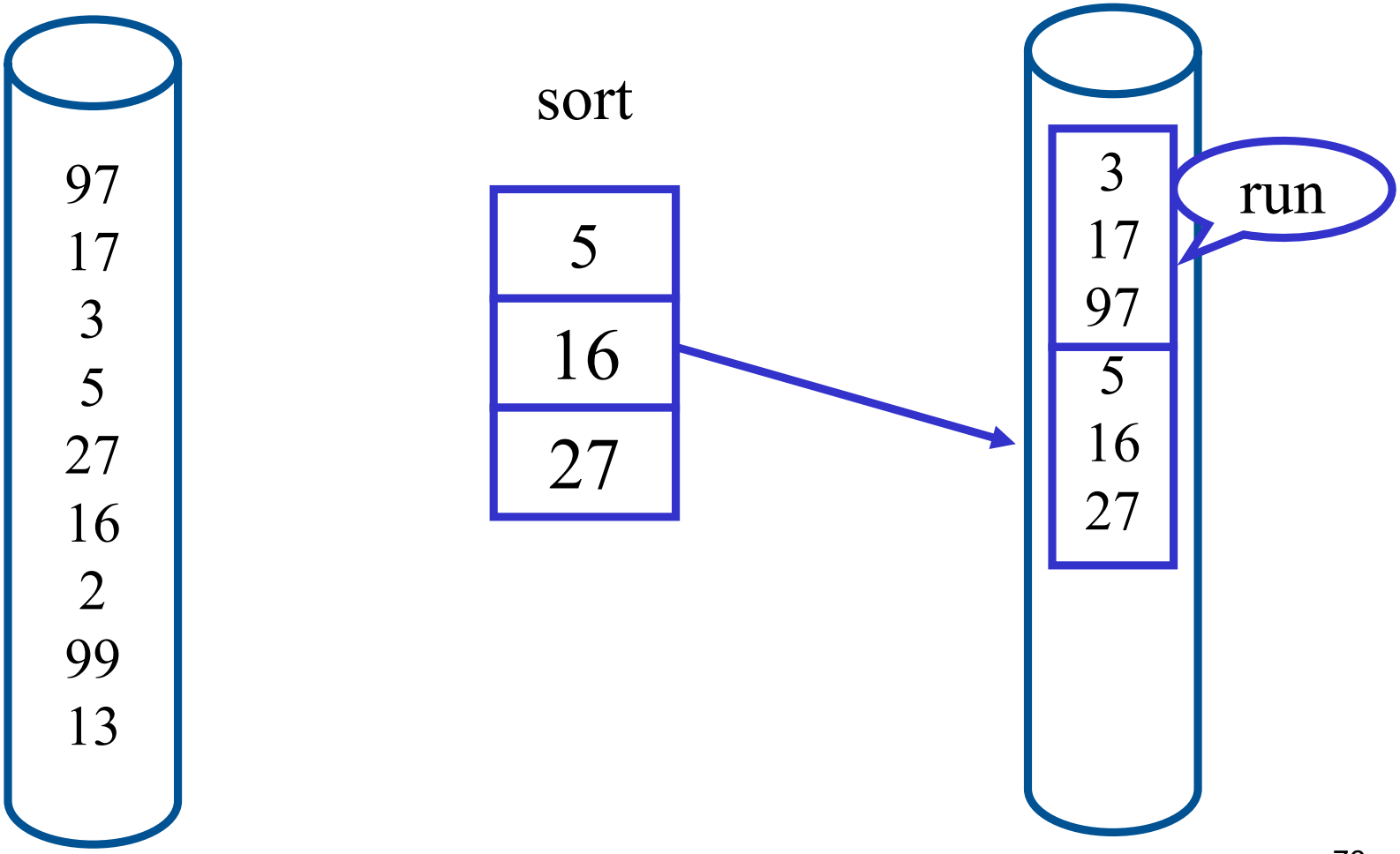


Illustration: Externes Sortieren

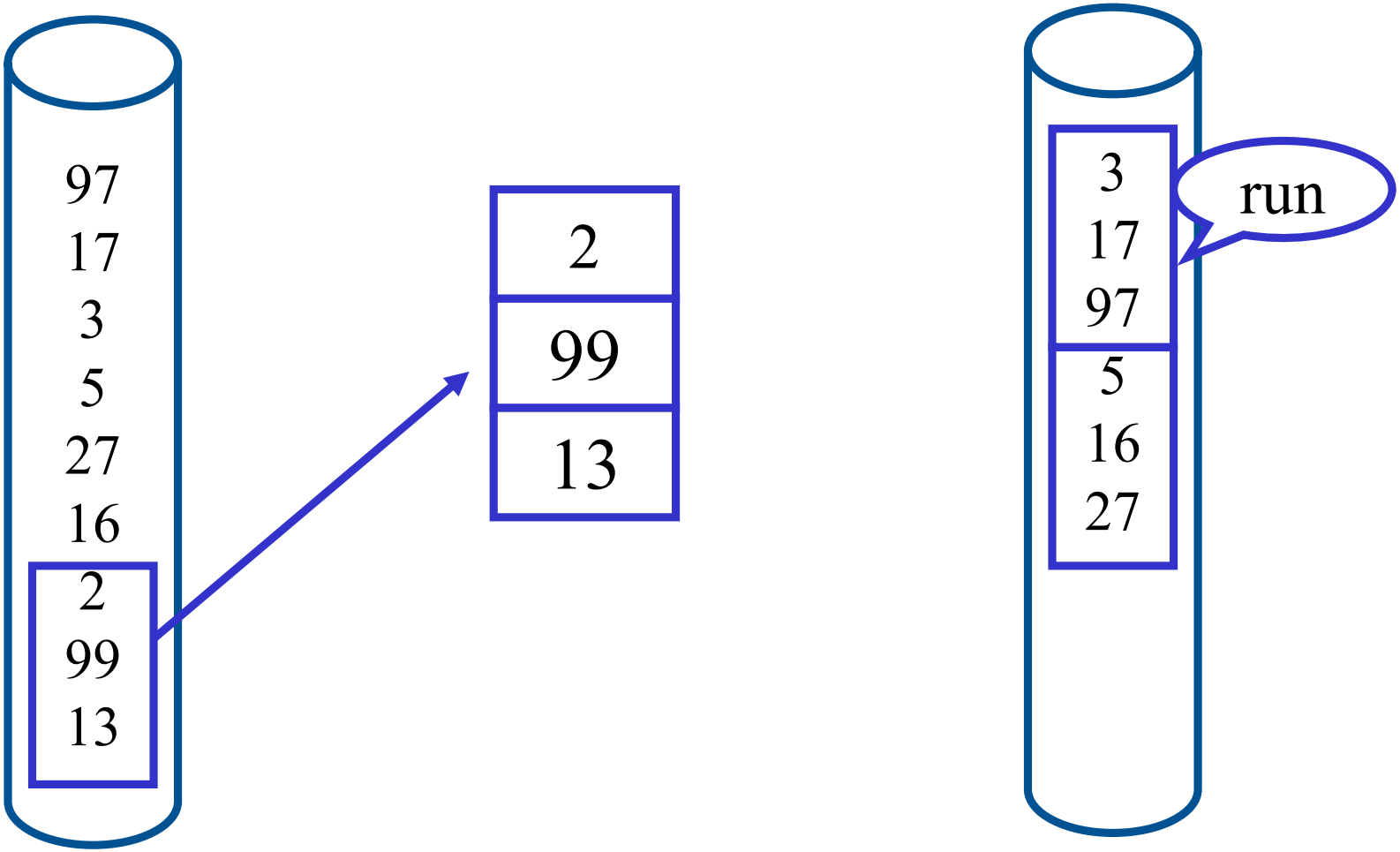


Illustration: Externes Sortieren

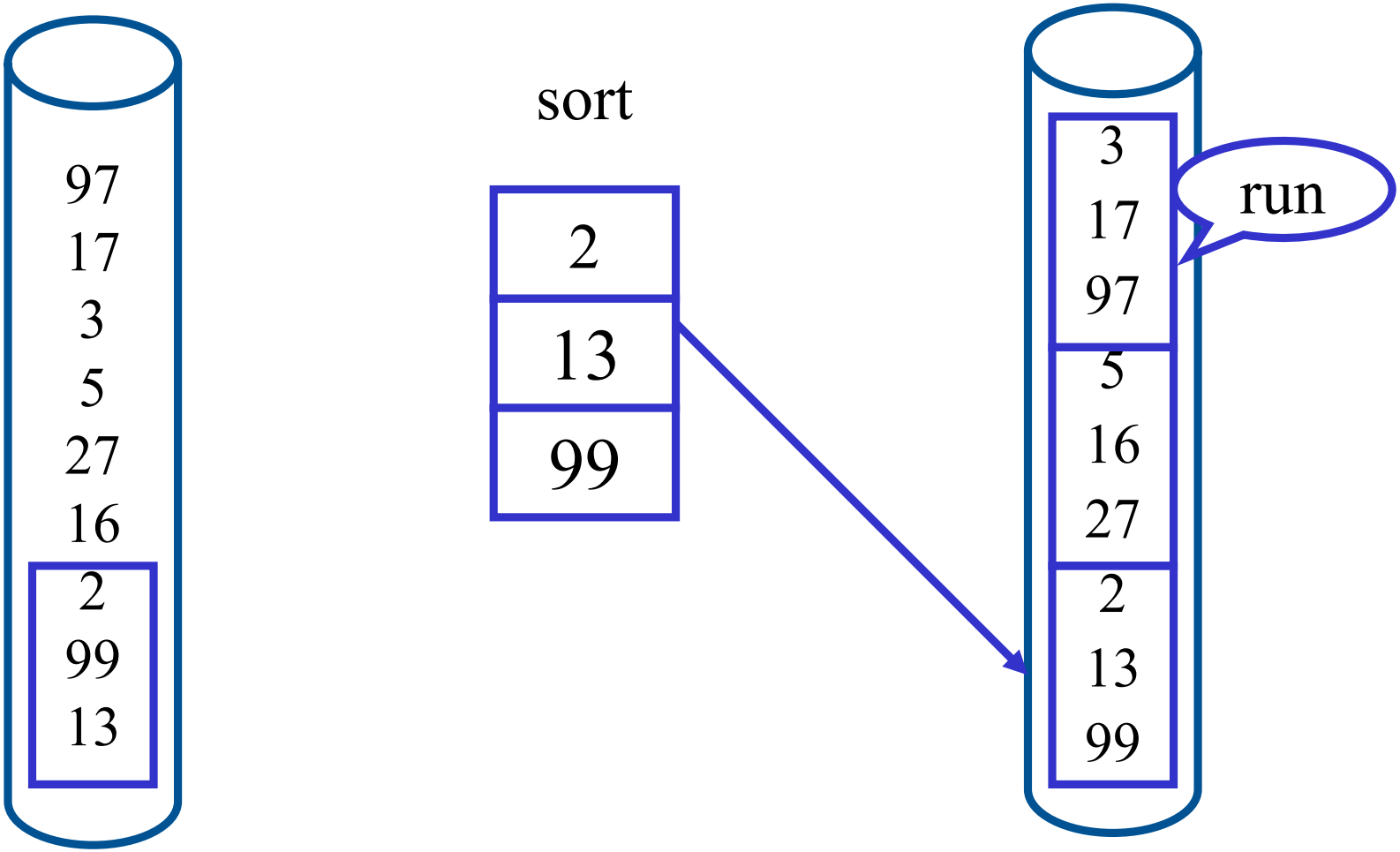


Illustration: Externes Sortieren

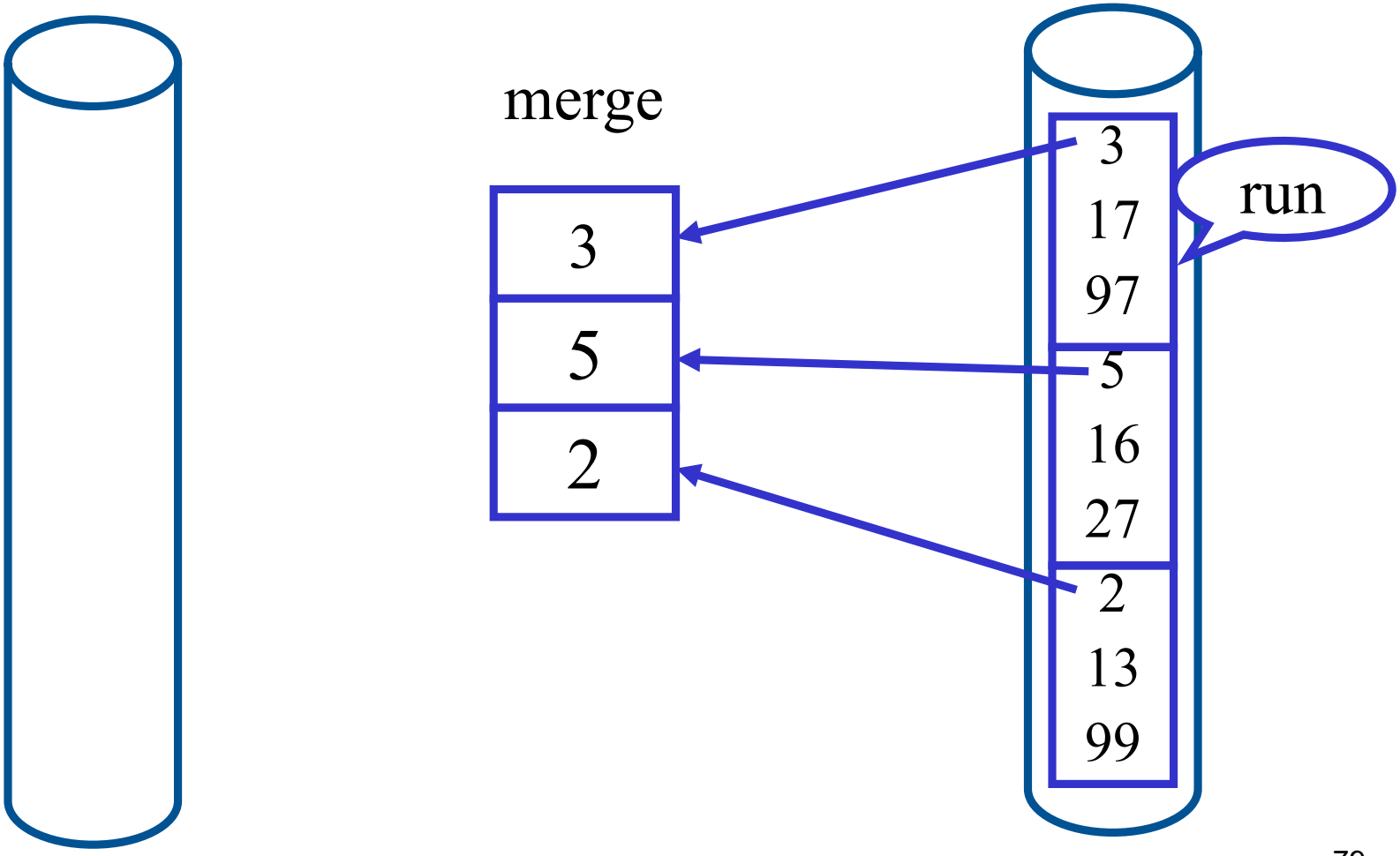


Illustration: Externes Sortieren

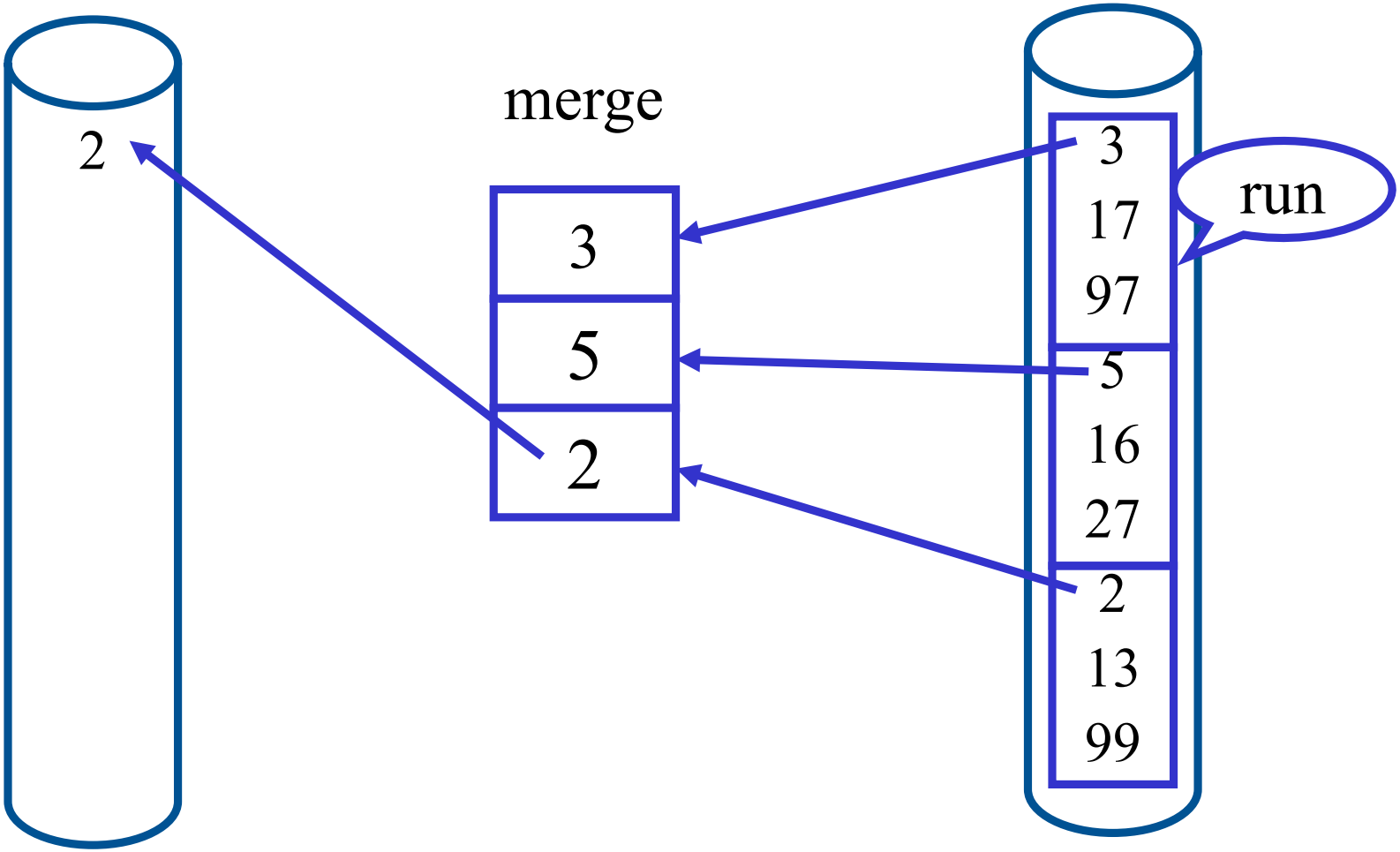


Illustration: Externes Sortieren

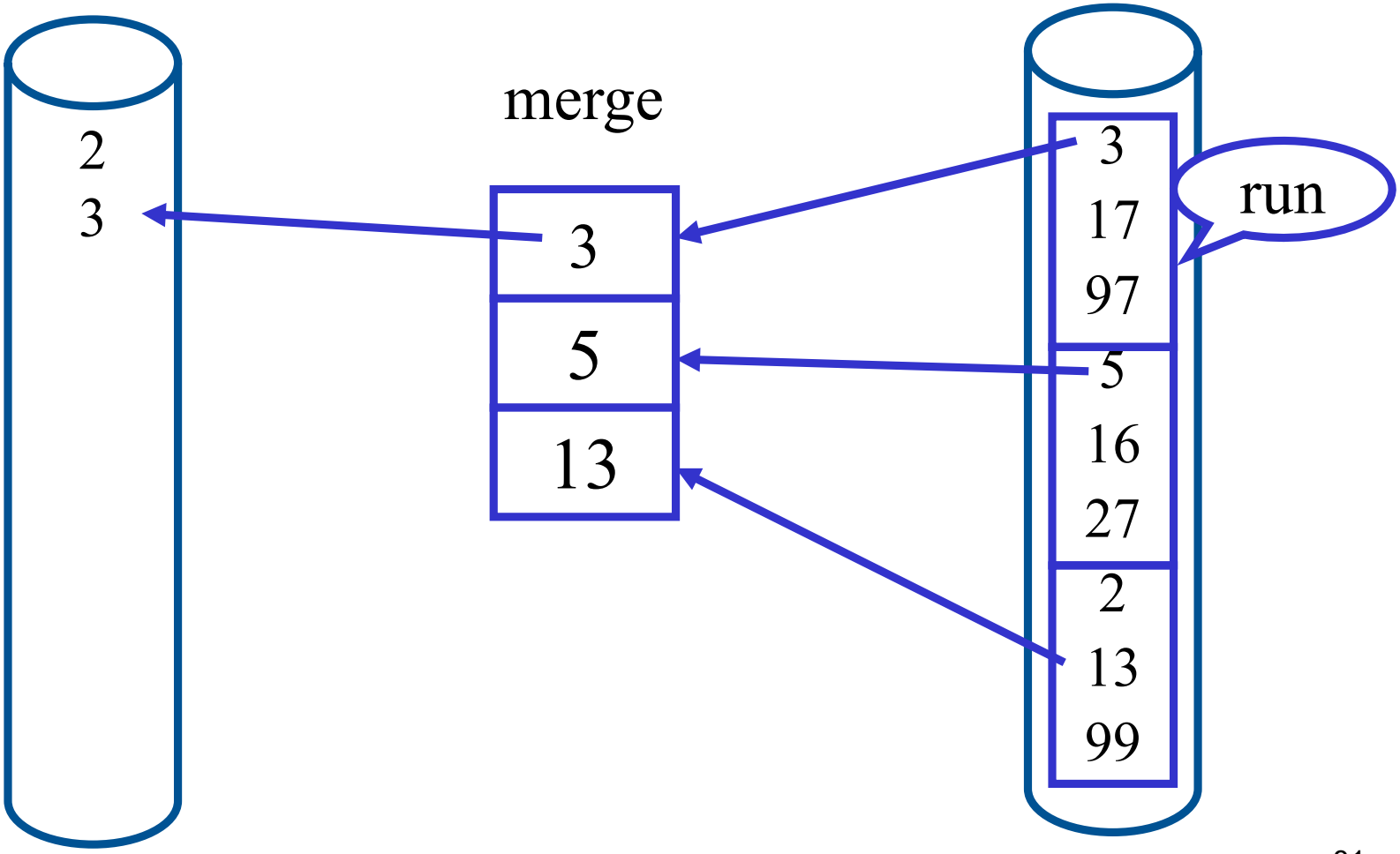


Illustration: Externes Sortieren

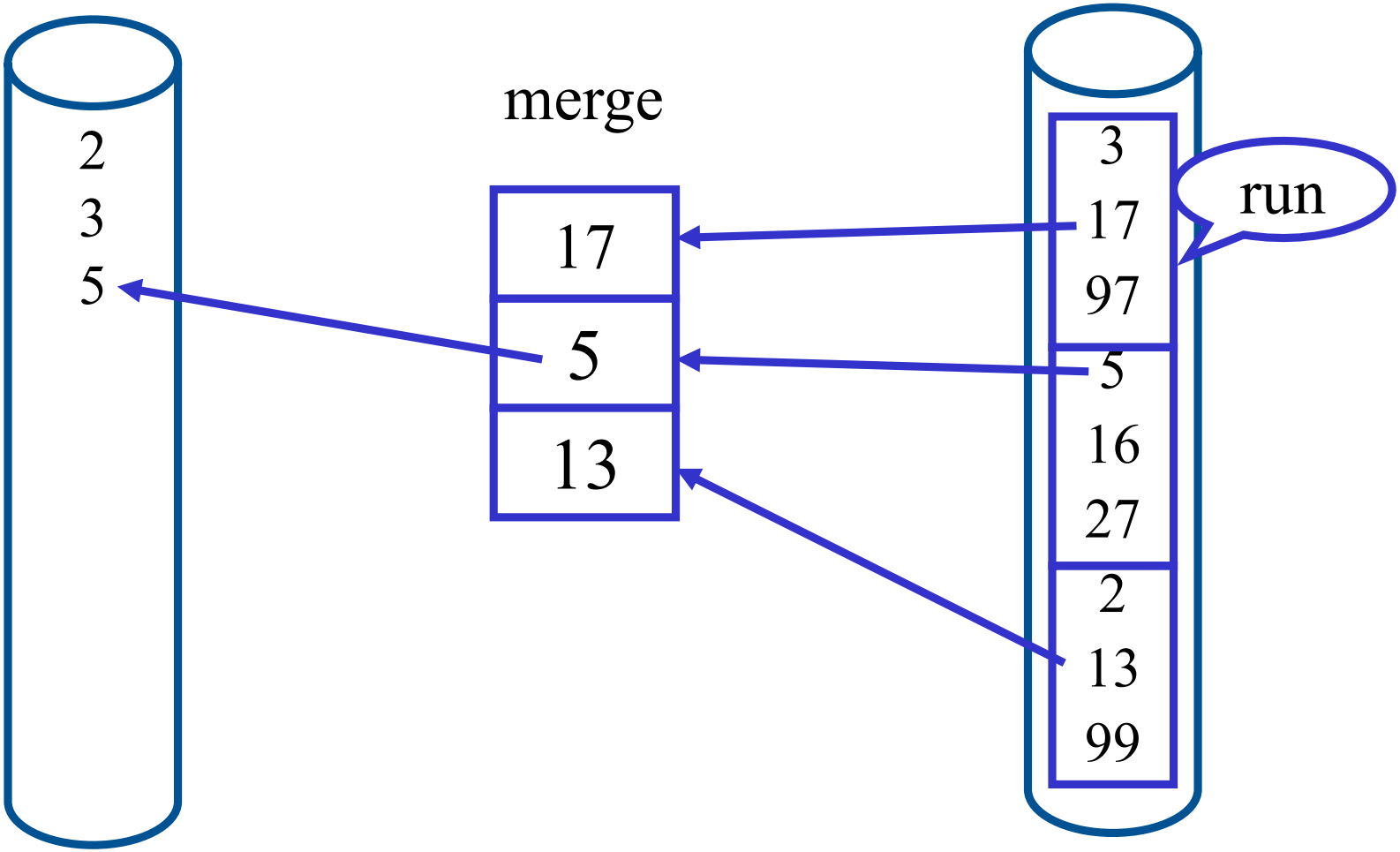


Illustration: Externes Sortieren

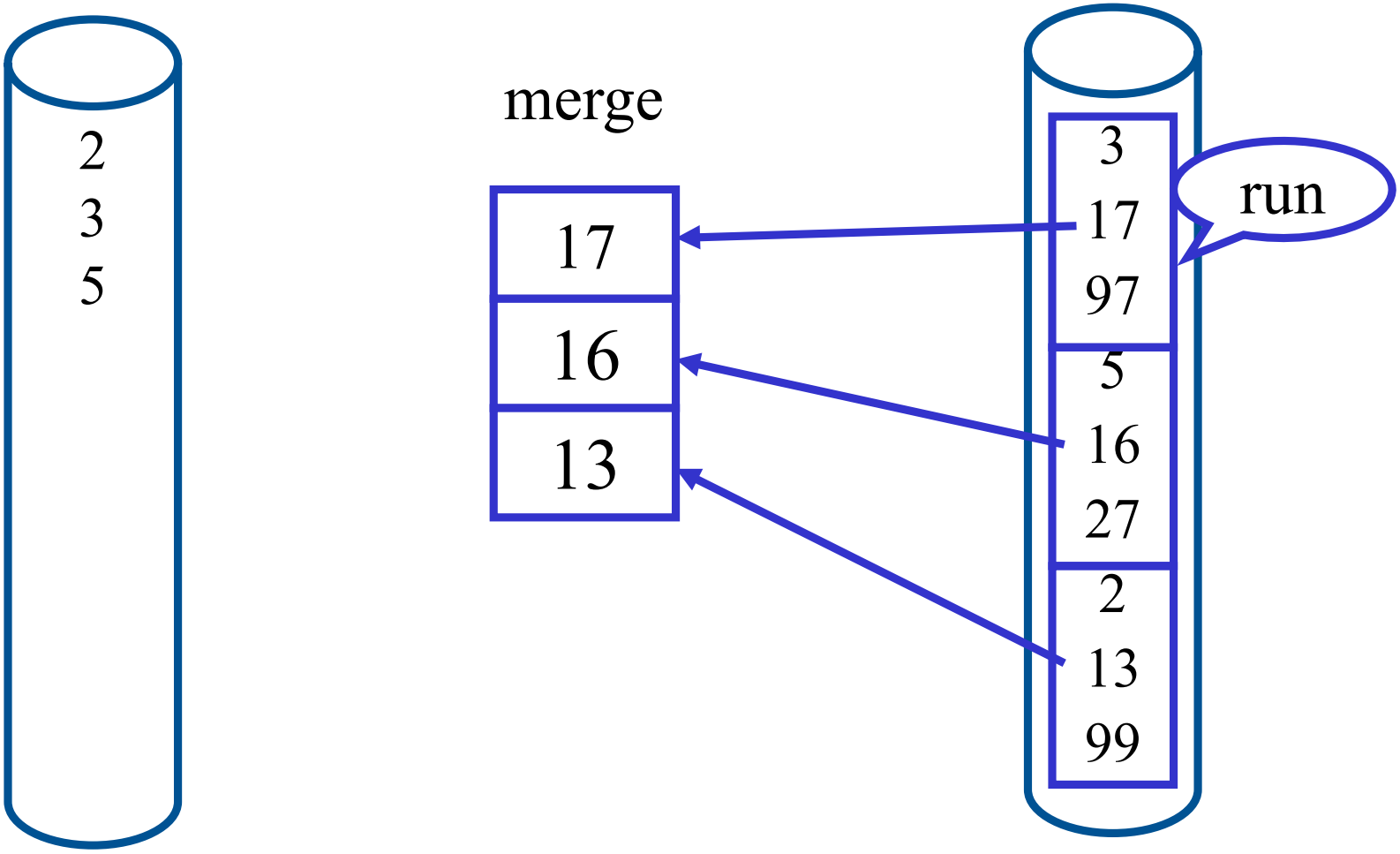
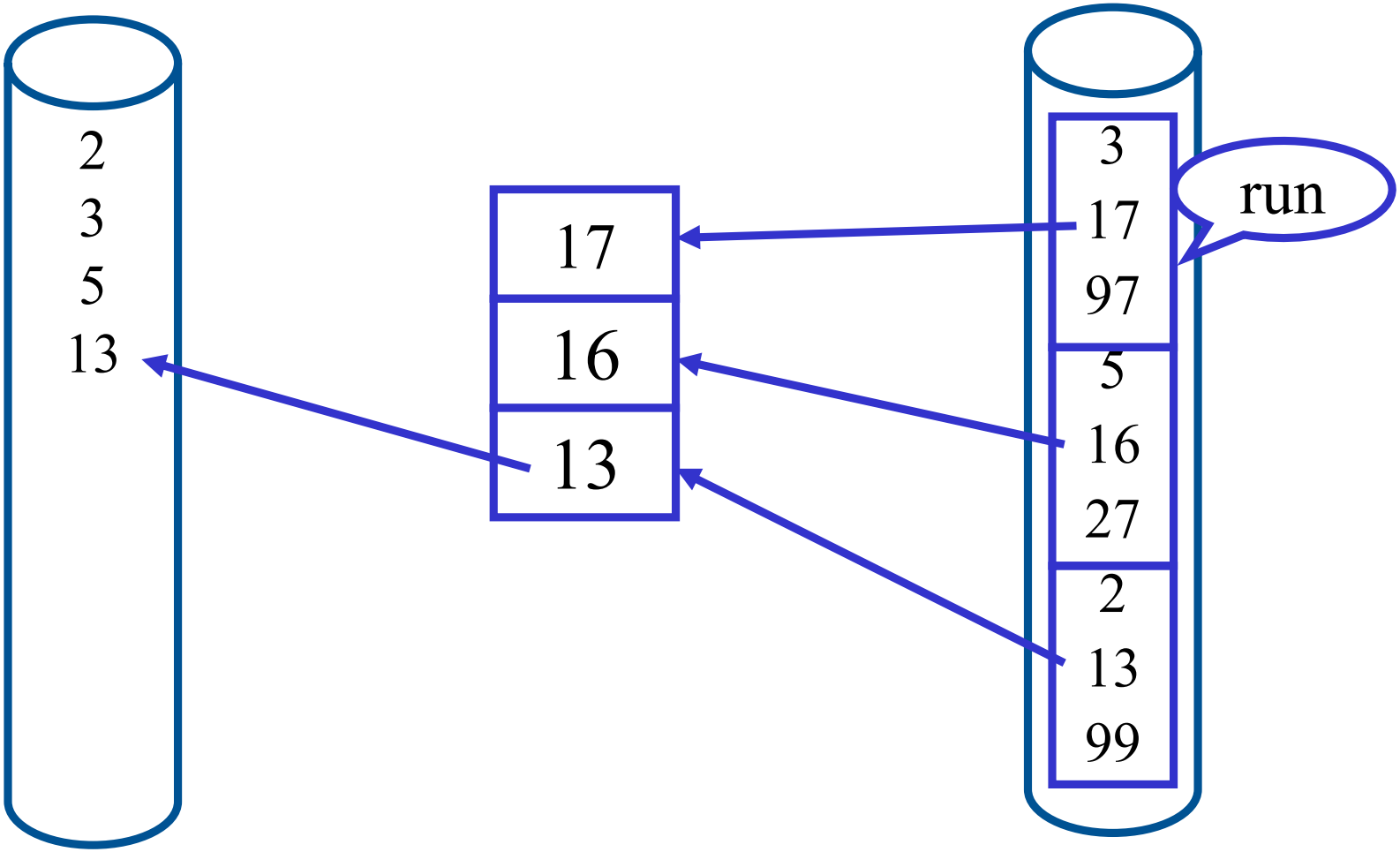
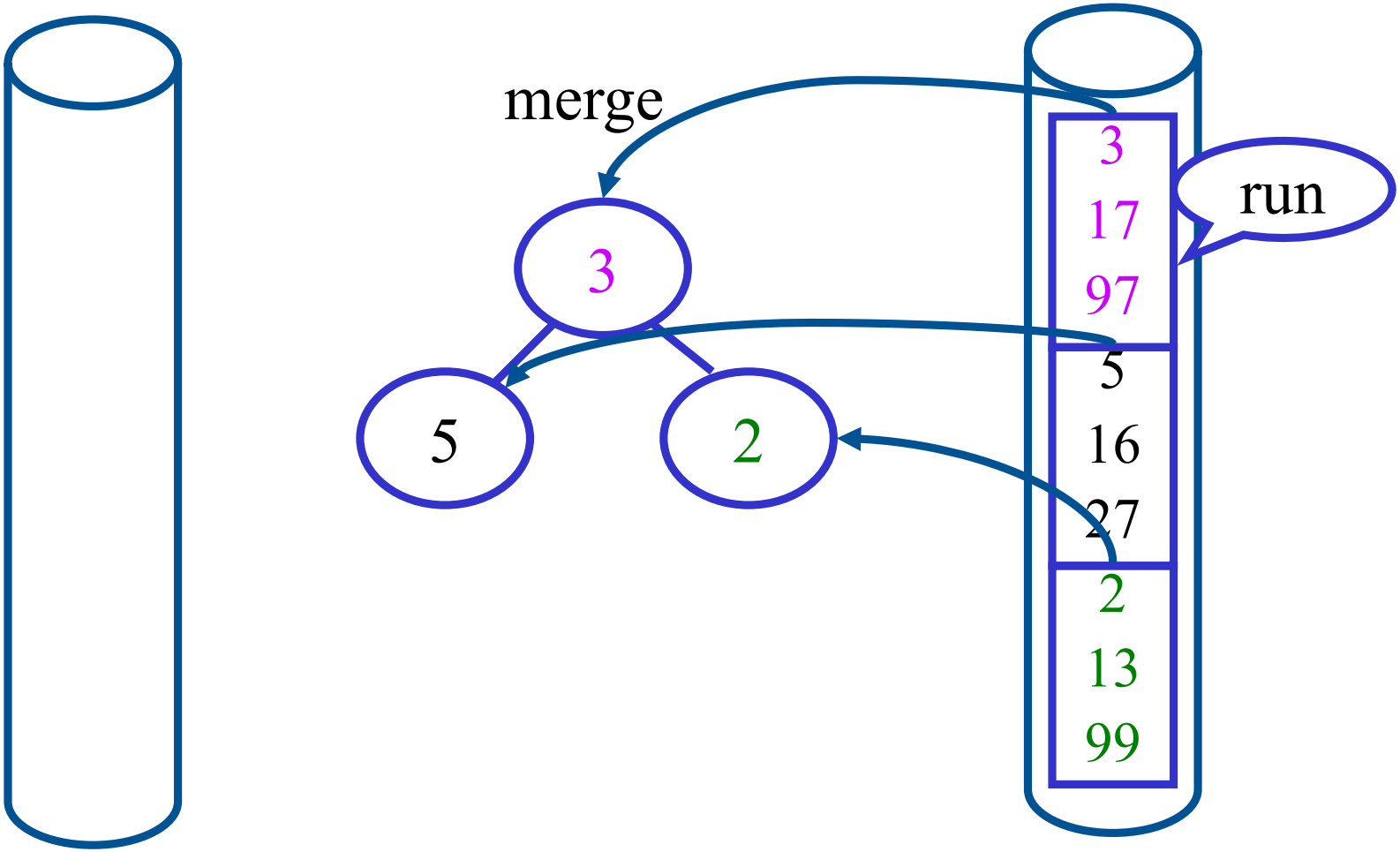


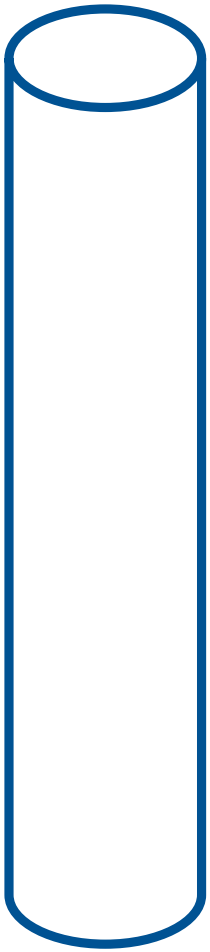
Illustration: Externes Sortieren

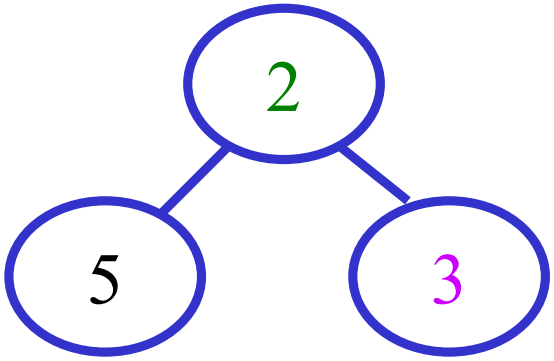


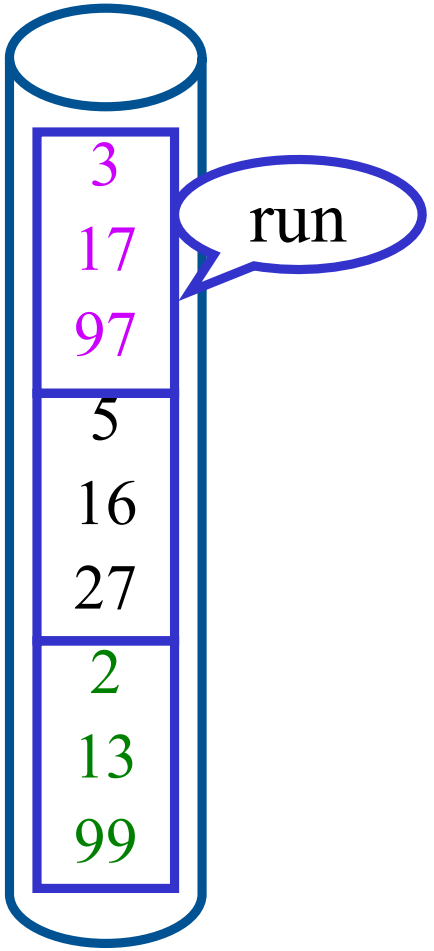
Externes Sortieren: Merge mittels Heap/Priority Queue



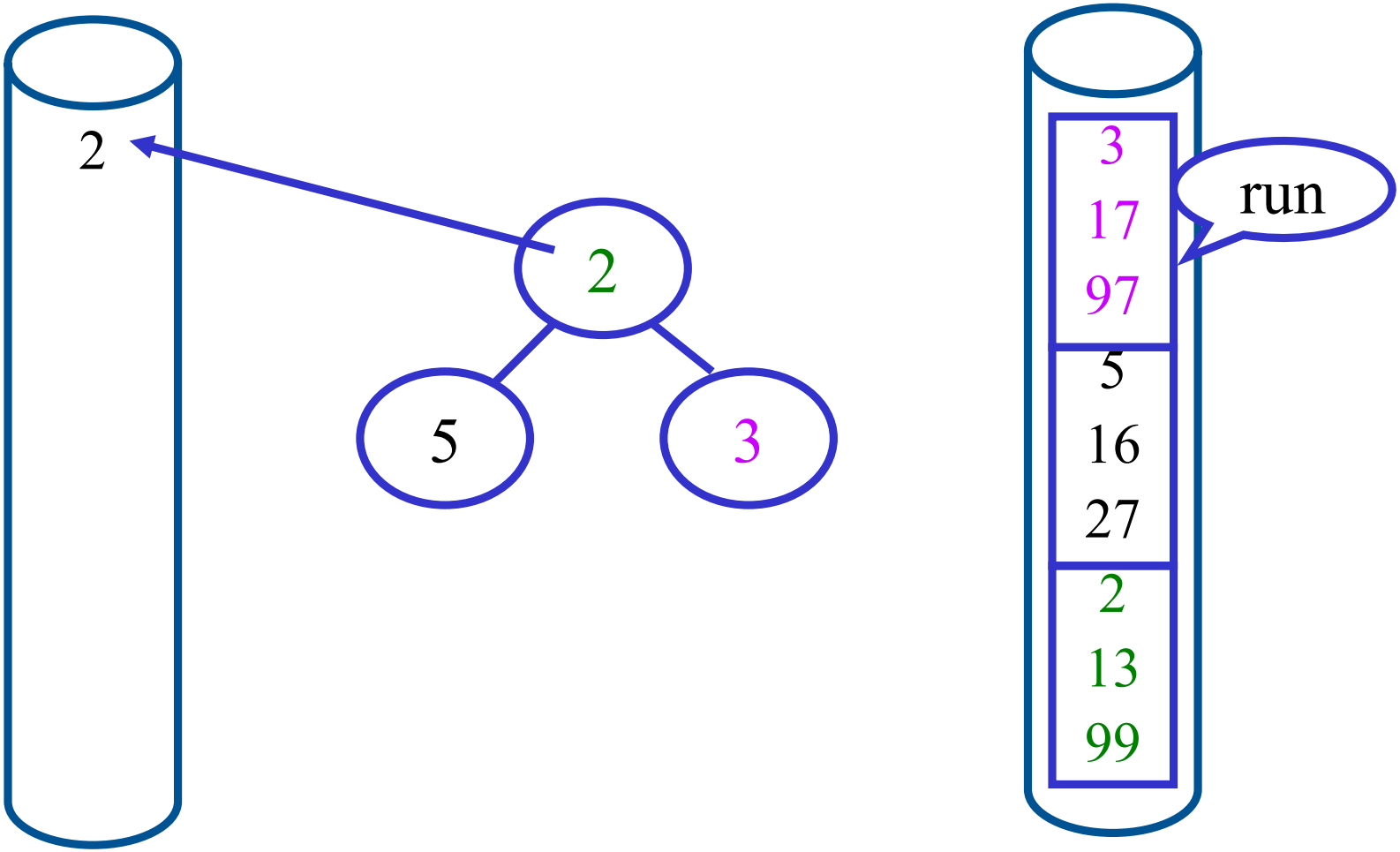
Externes Sortieren: Merge mittels Heap/Priority Queue



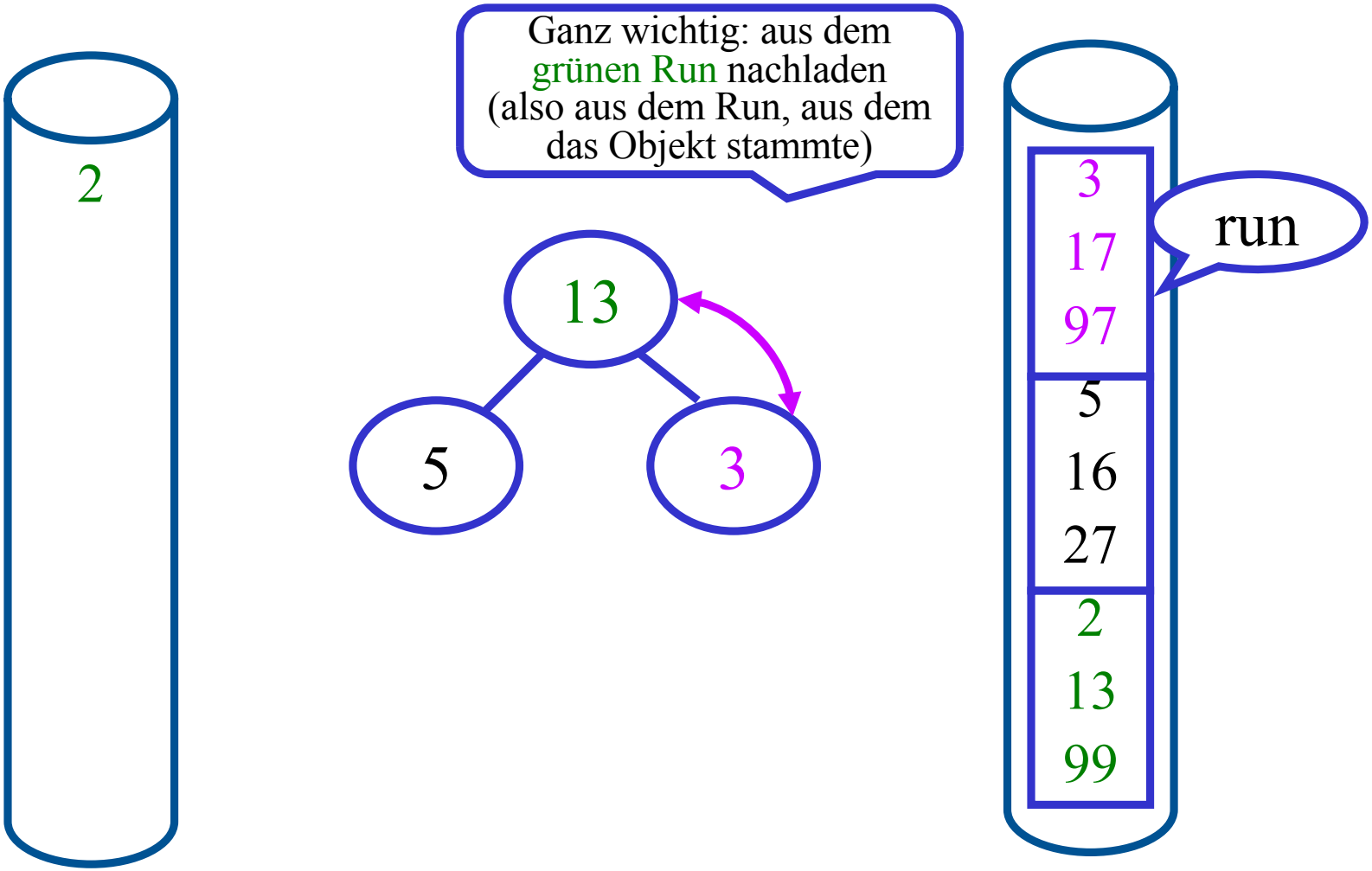




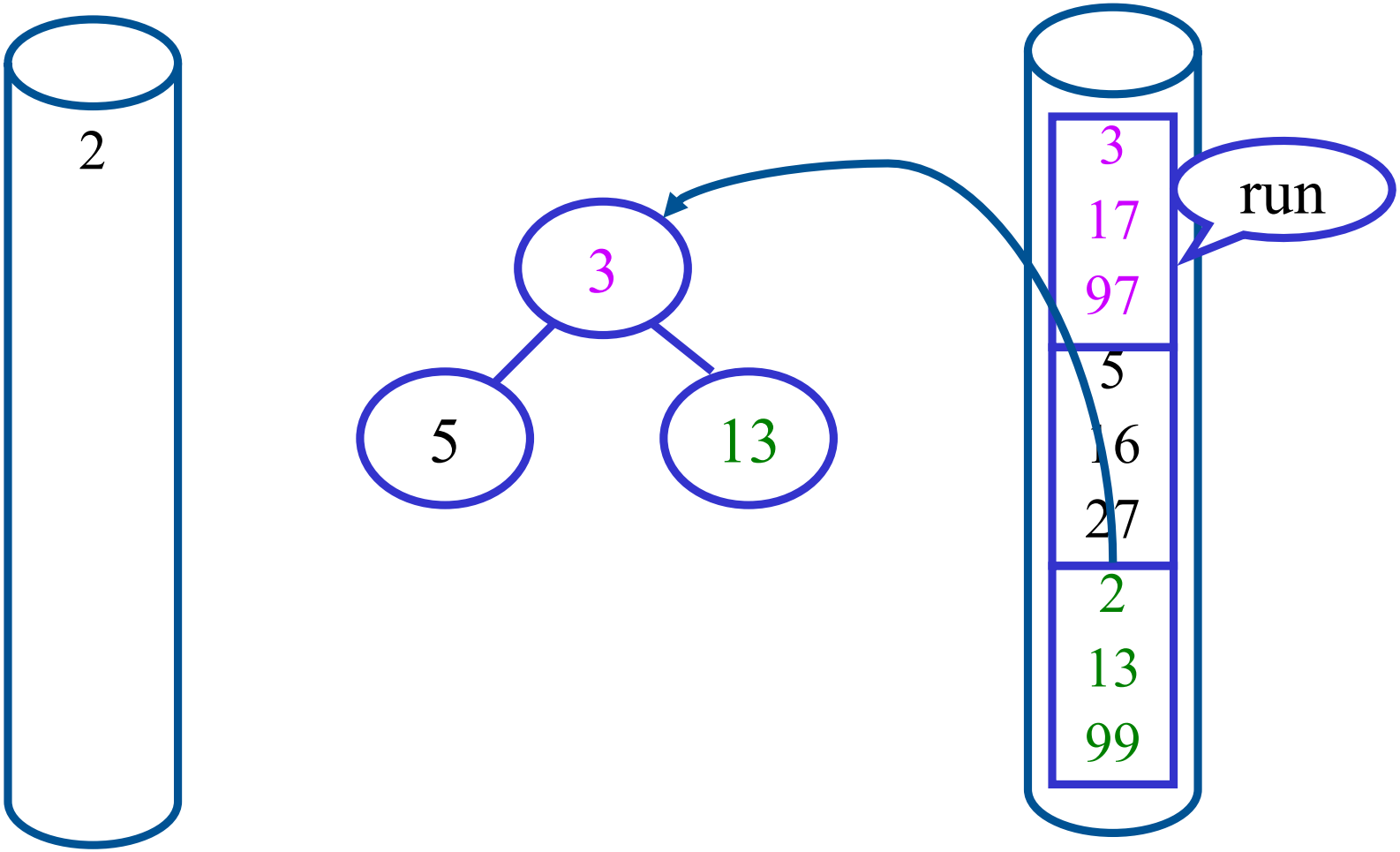
Externes Sortieren: Merge mittels Heap/Priority Queue



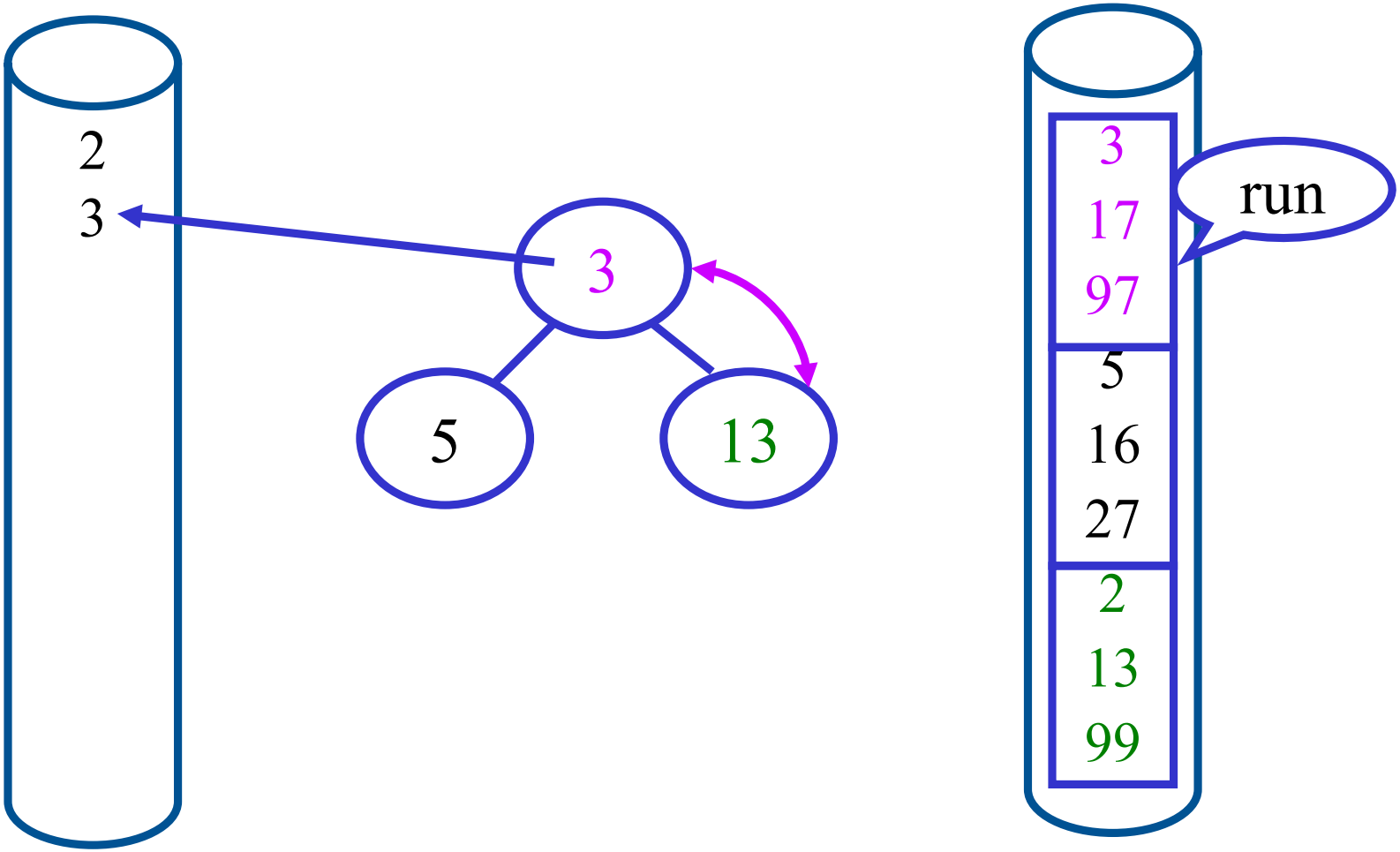
Externes Sortieren: Merge mittels Heap/Priority Queue



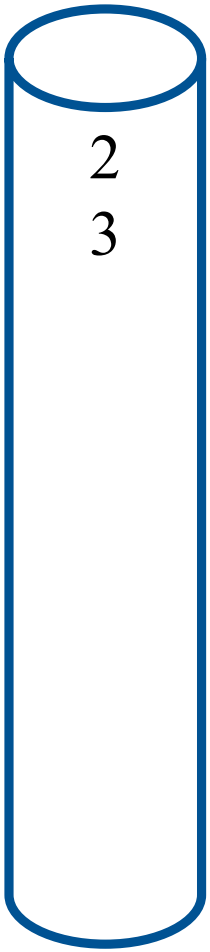
Externes Sortieren: Merge mittels Heap/Priority Queue

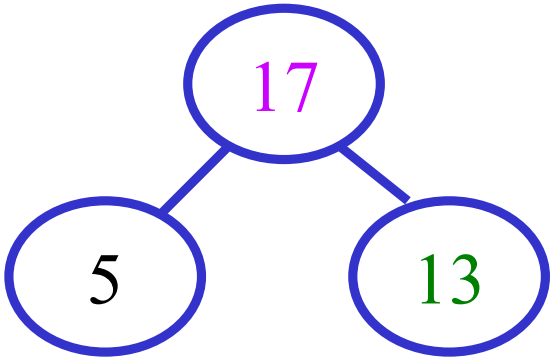


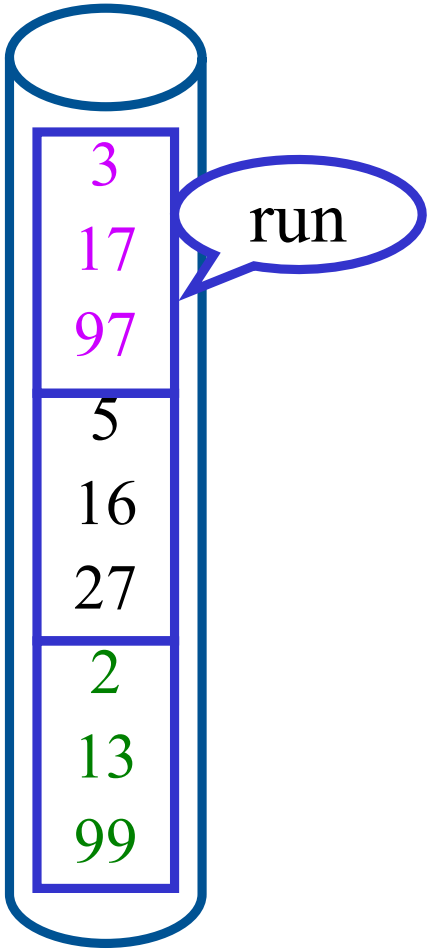
Externes Sortieren: Merge mittels Heap/Priority Queue



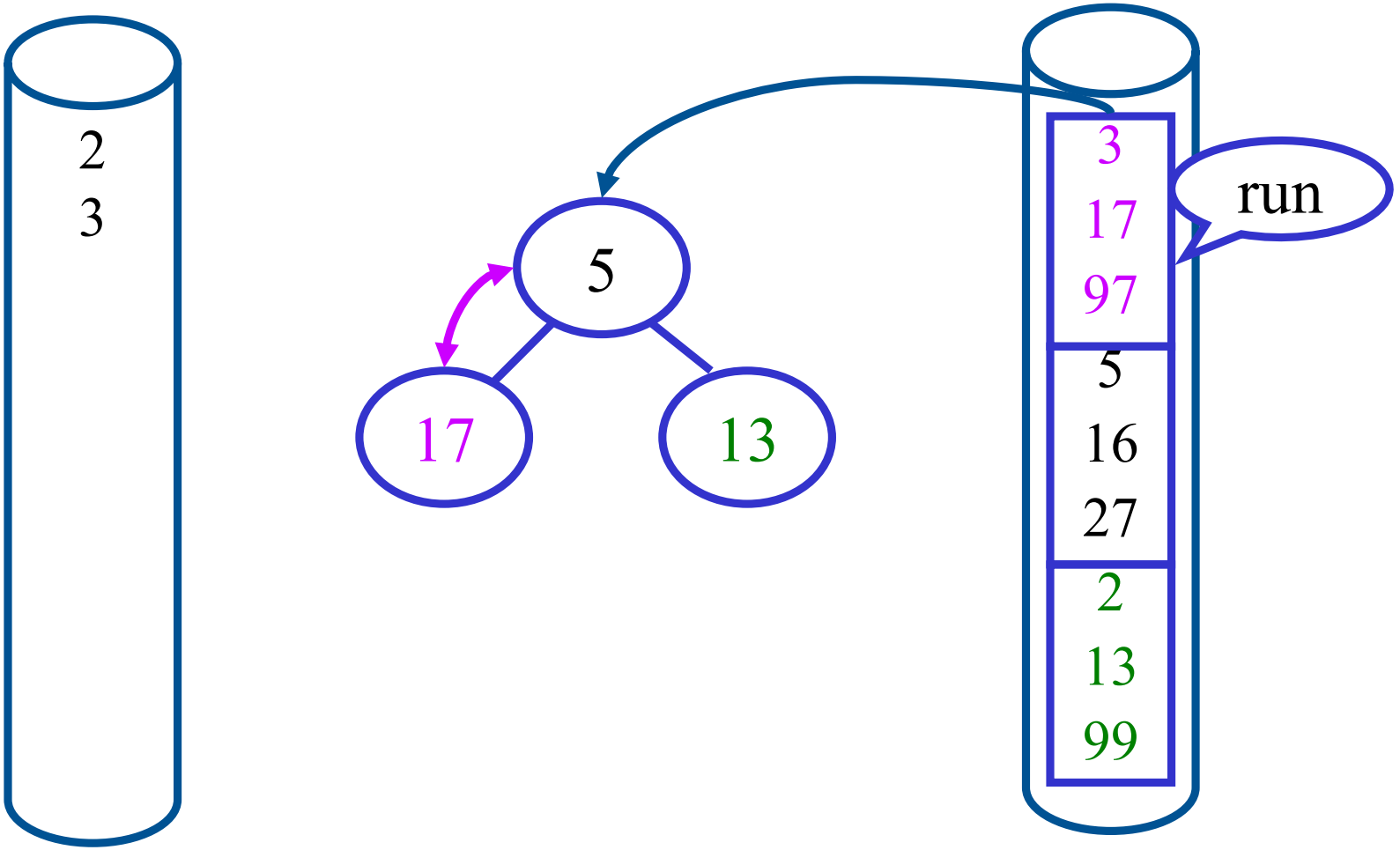
Externes Sortieren: Merge mittels Heap/Priority Queue



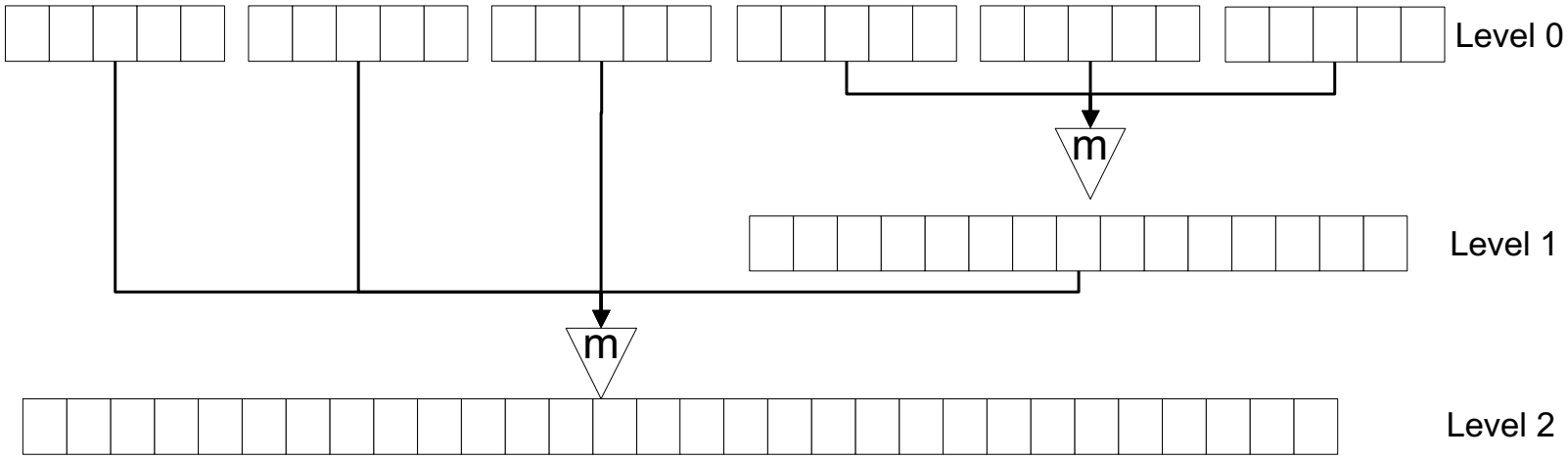




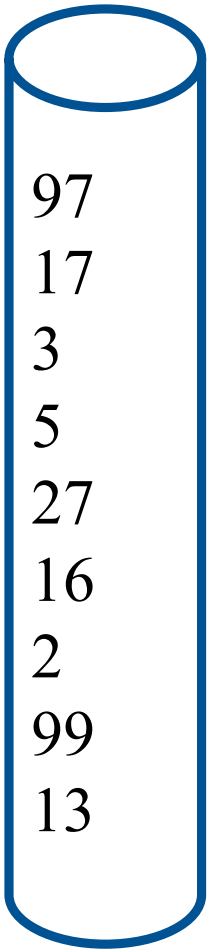
Externes Sortieren: Merge mittels Heap/Priority Queue



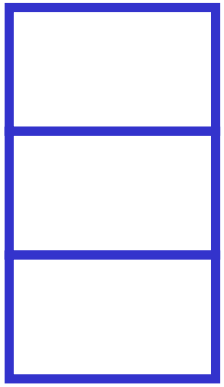
Mehrstufiges Mischen / Merge

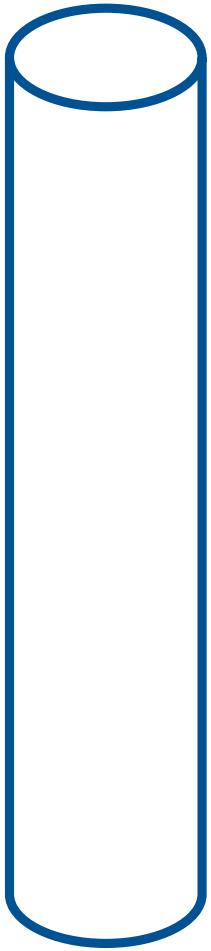


Replacement Selection während der Run-Generierung

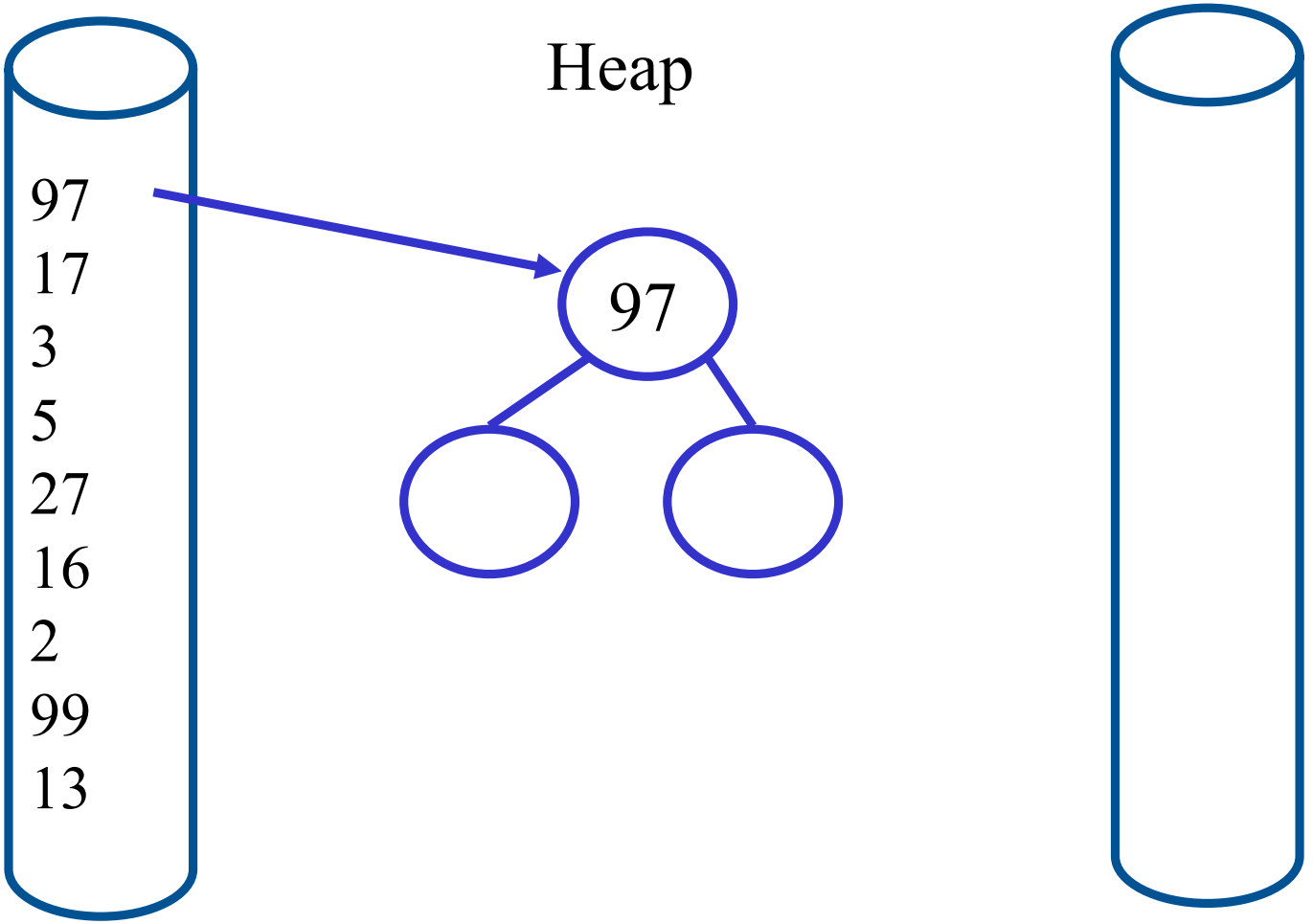


Ersetze Array durch
Einen Heap

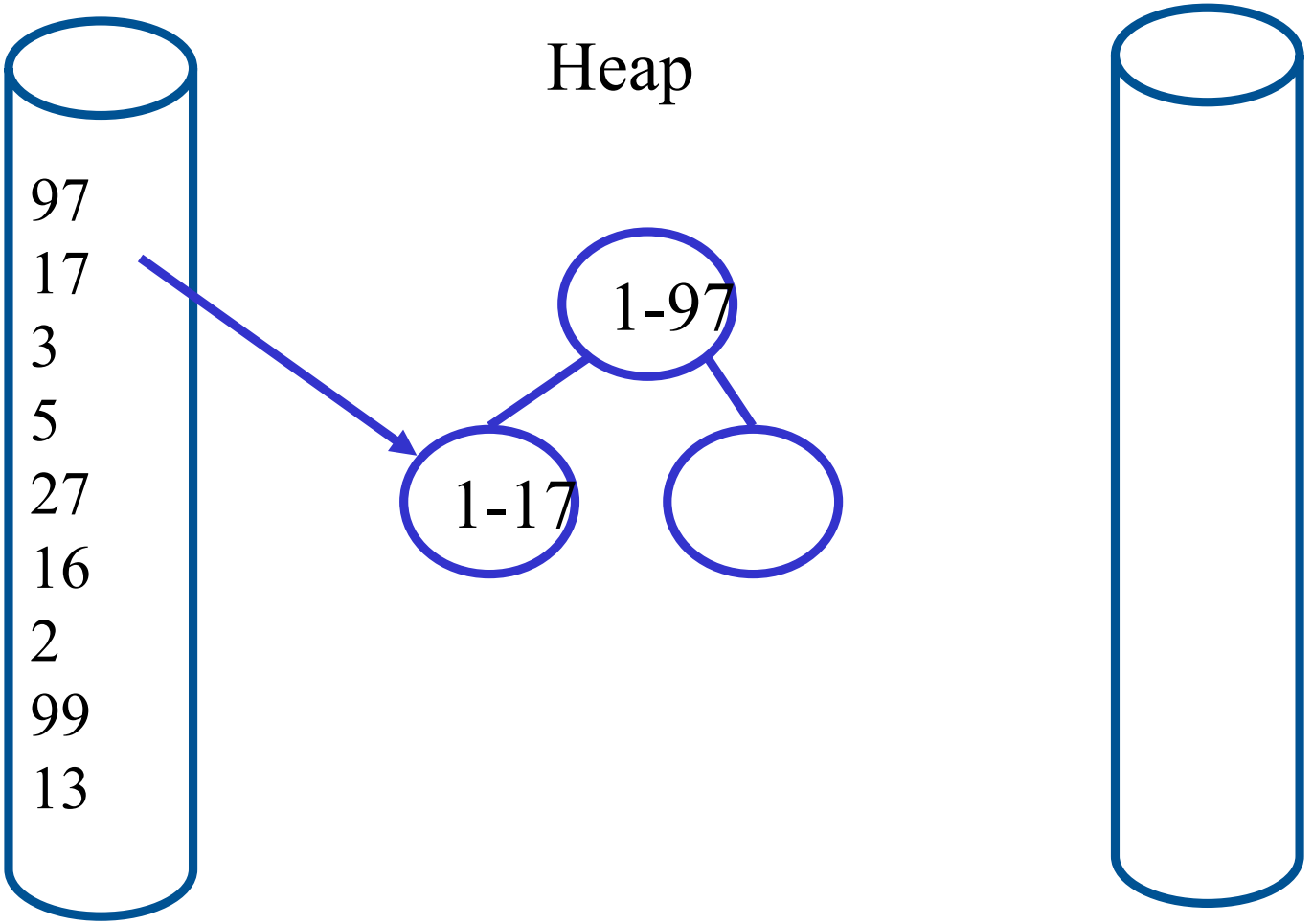




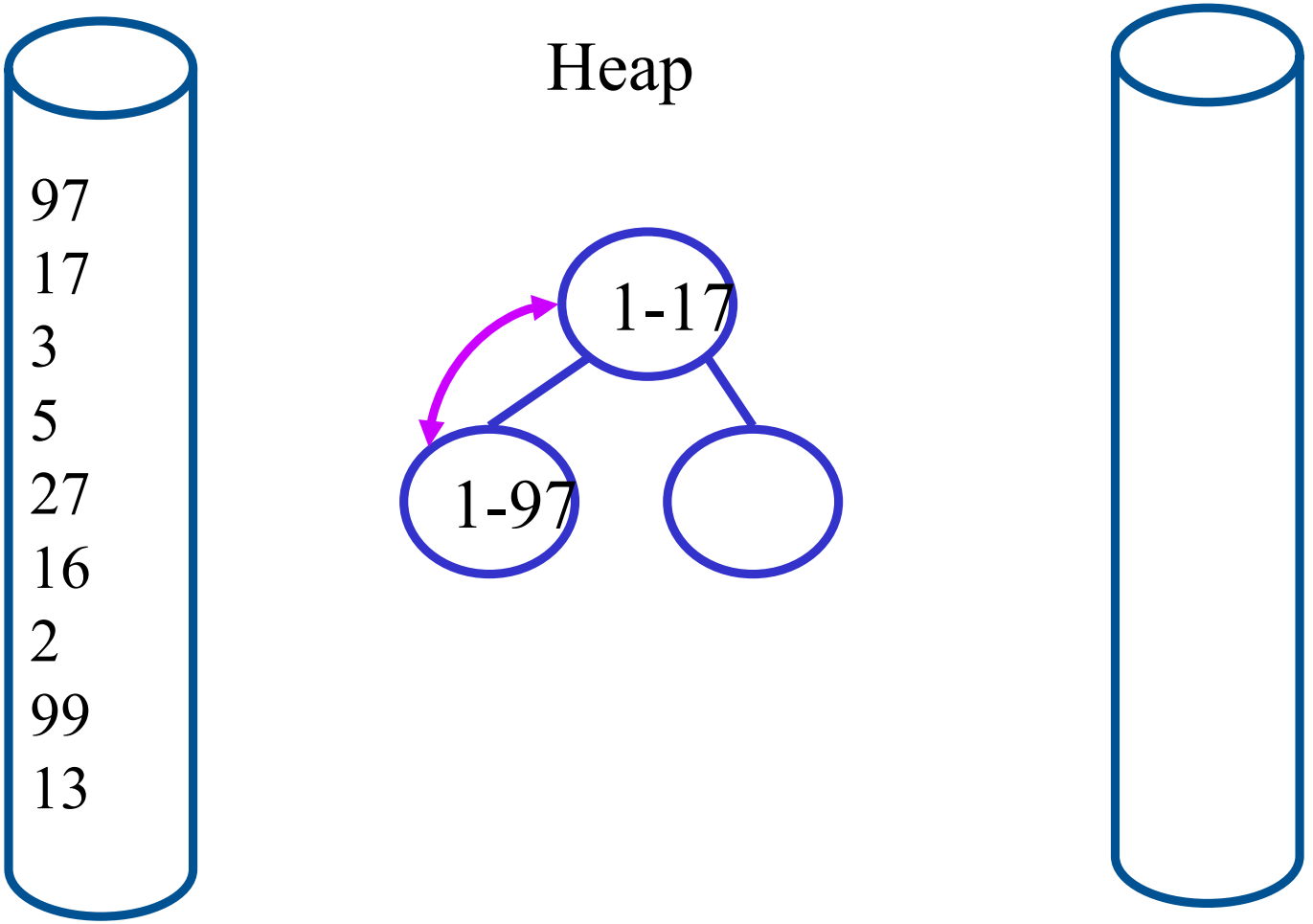
Replacement Selection während der Run-Generierung



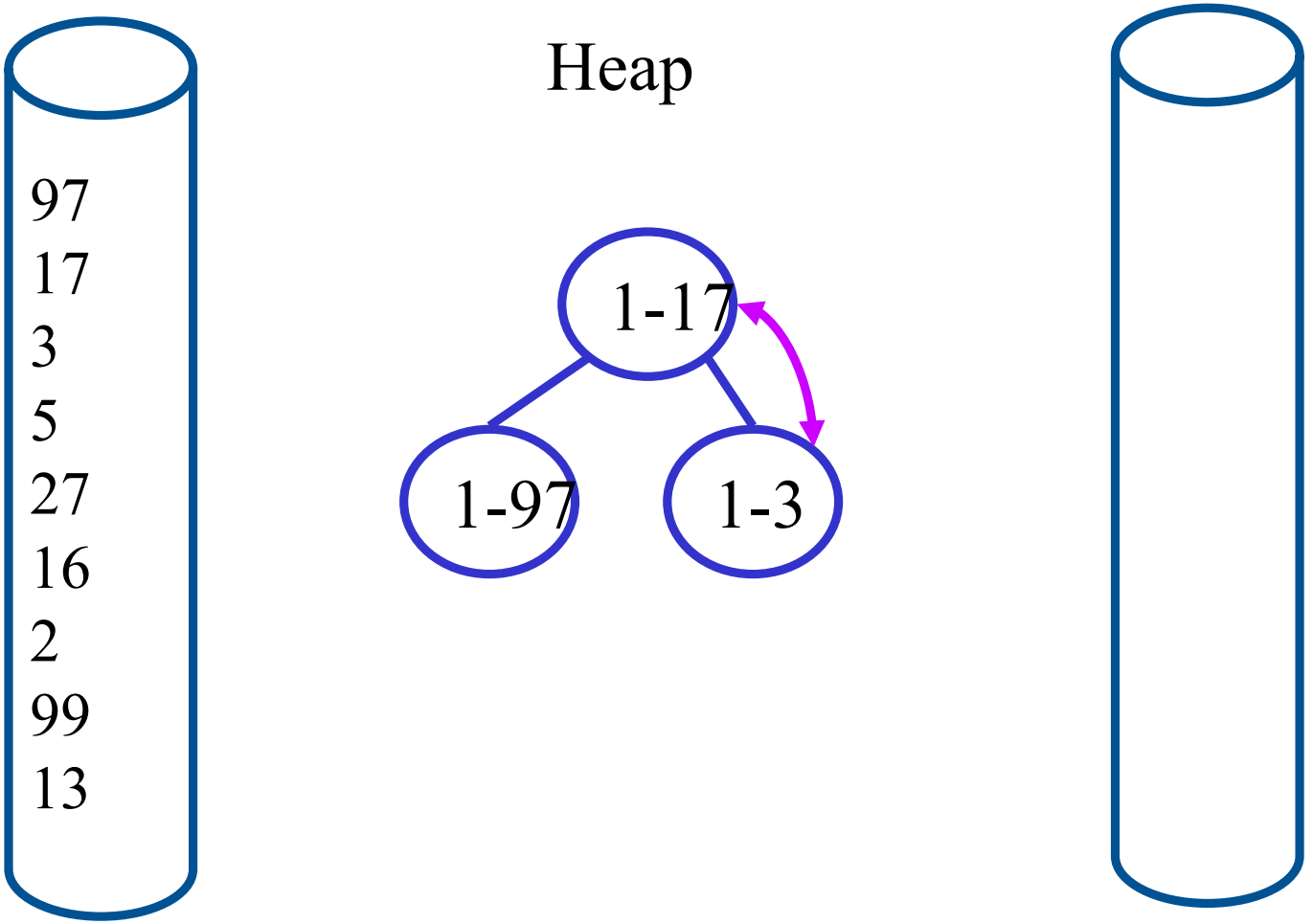
Replacement Selection während der Run-Generierung



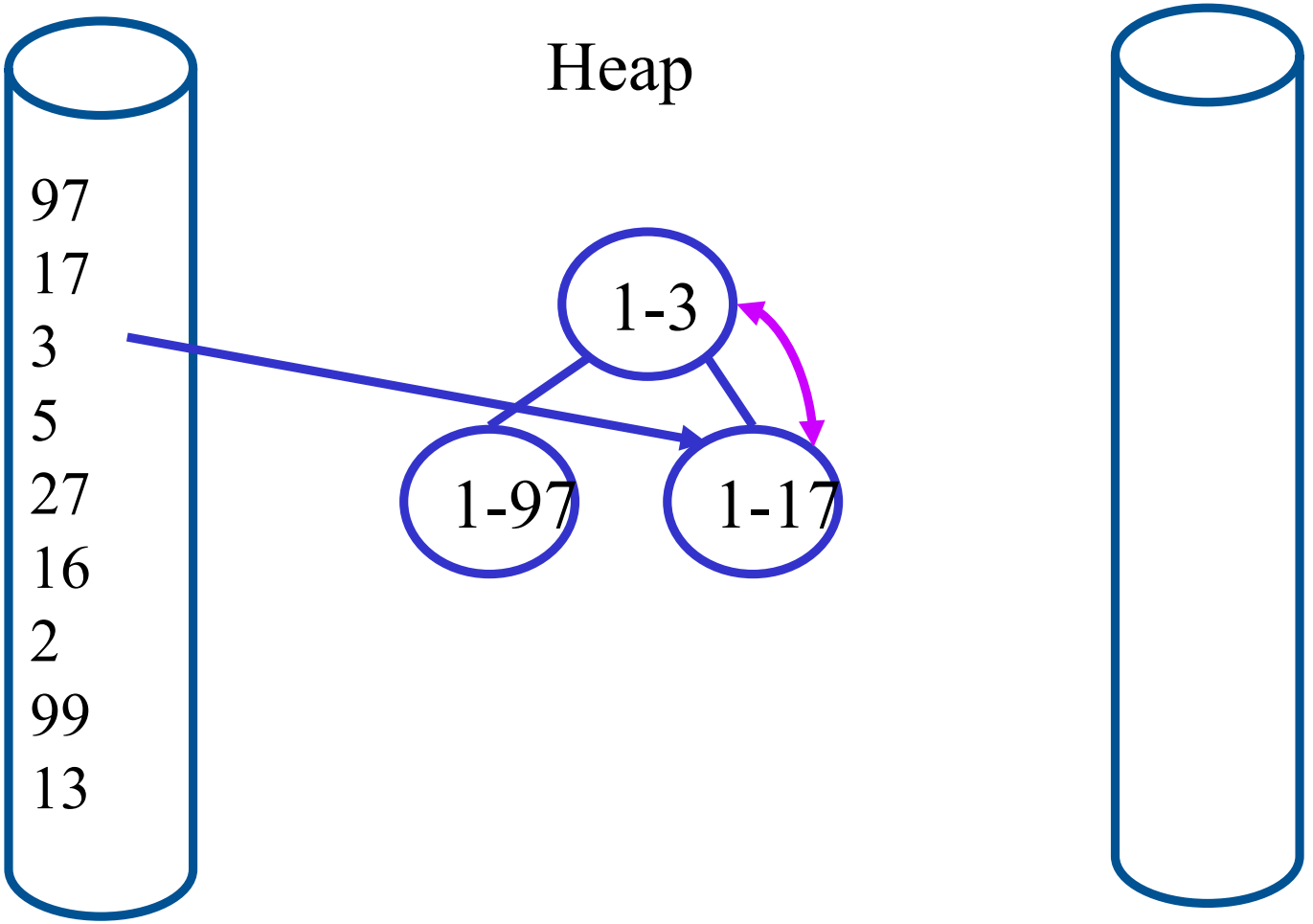
Replacement Selection während der Run-Generierung



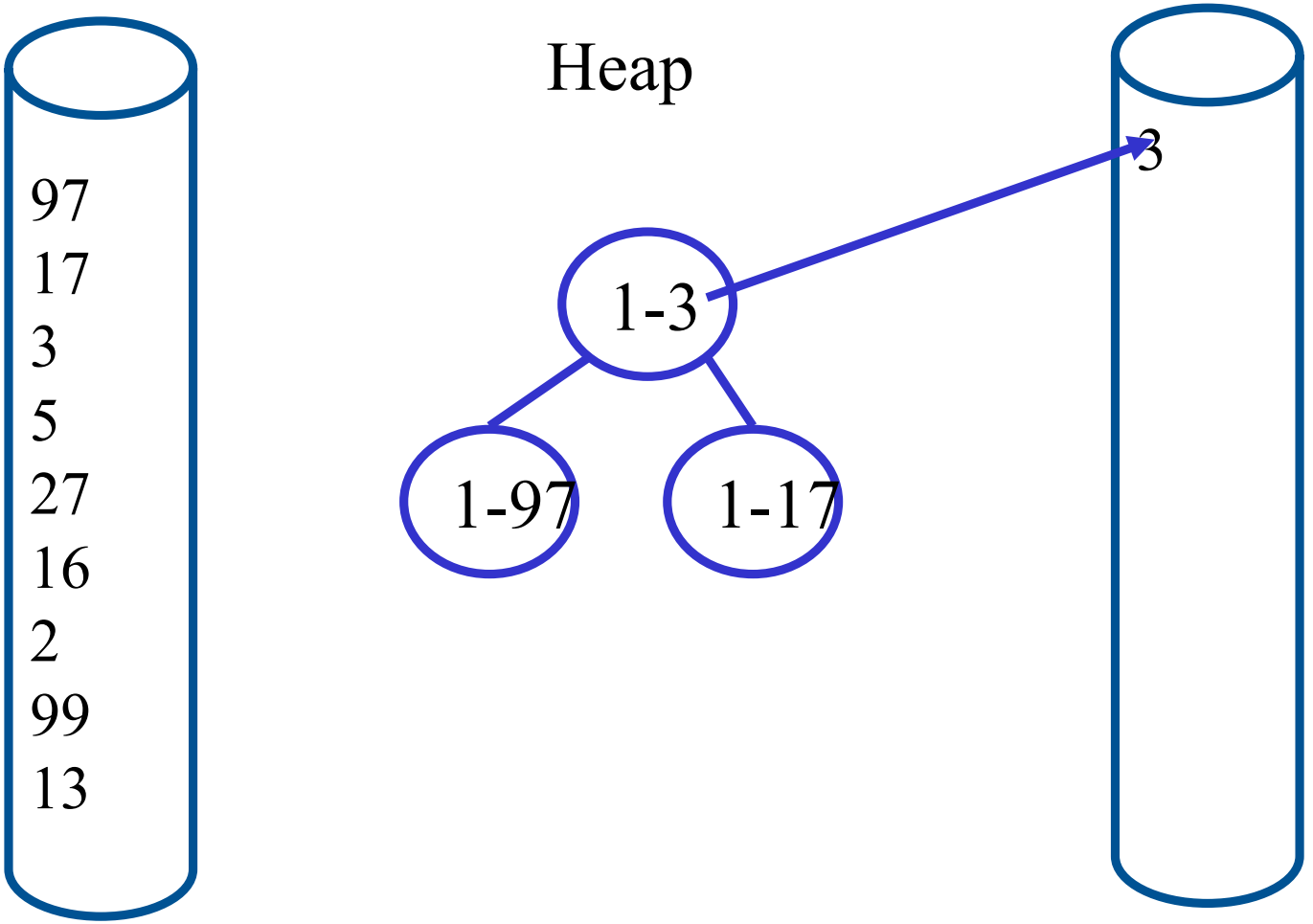
Replacement Selection während der Run-Generierung



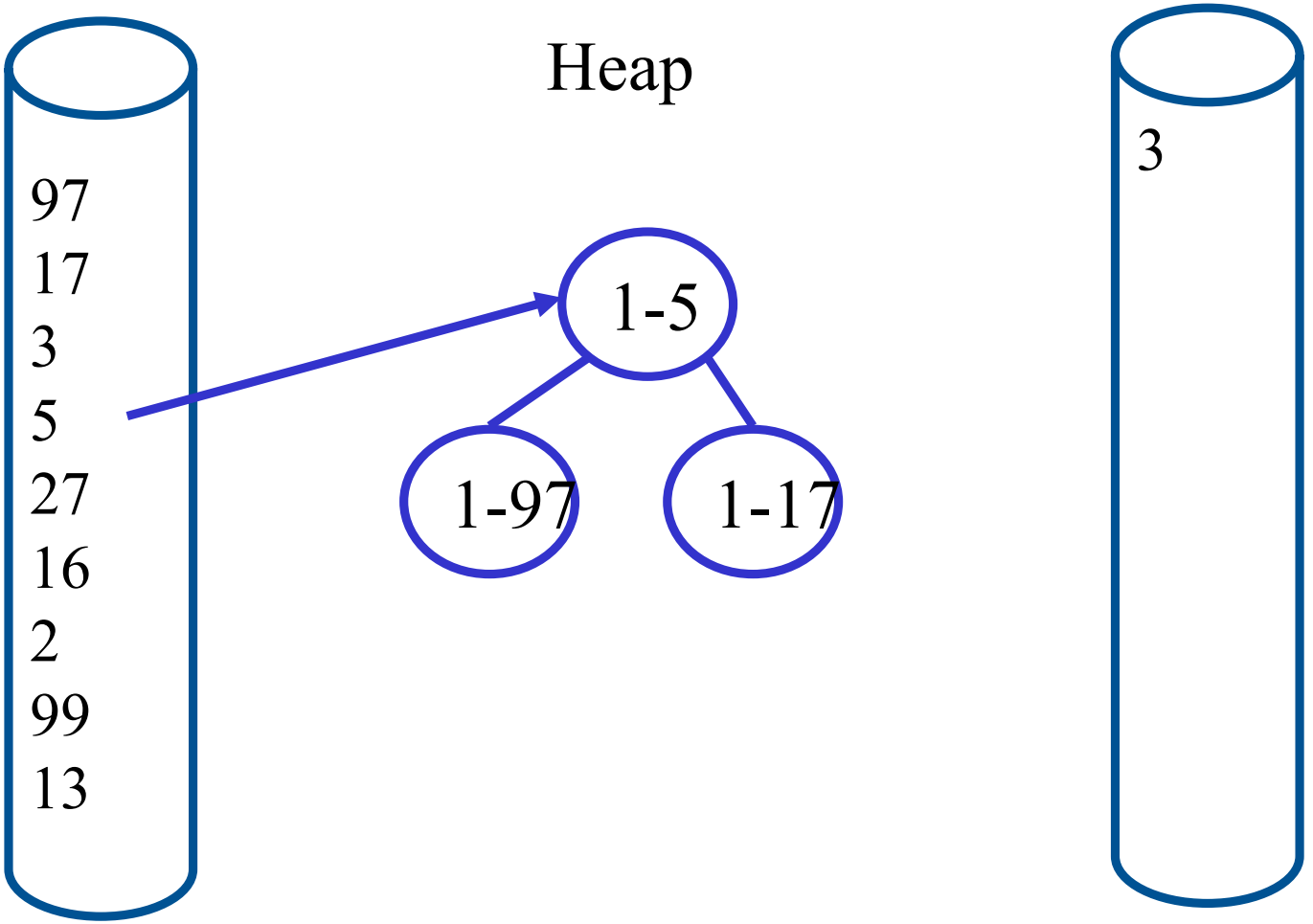
Replacement Selection während der Run-Generierung



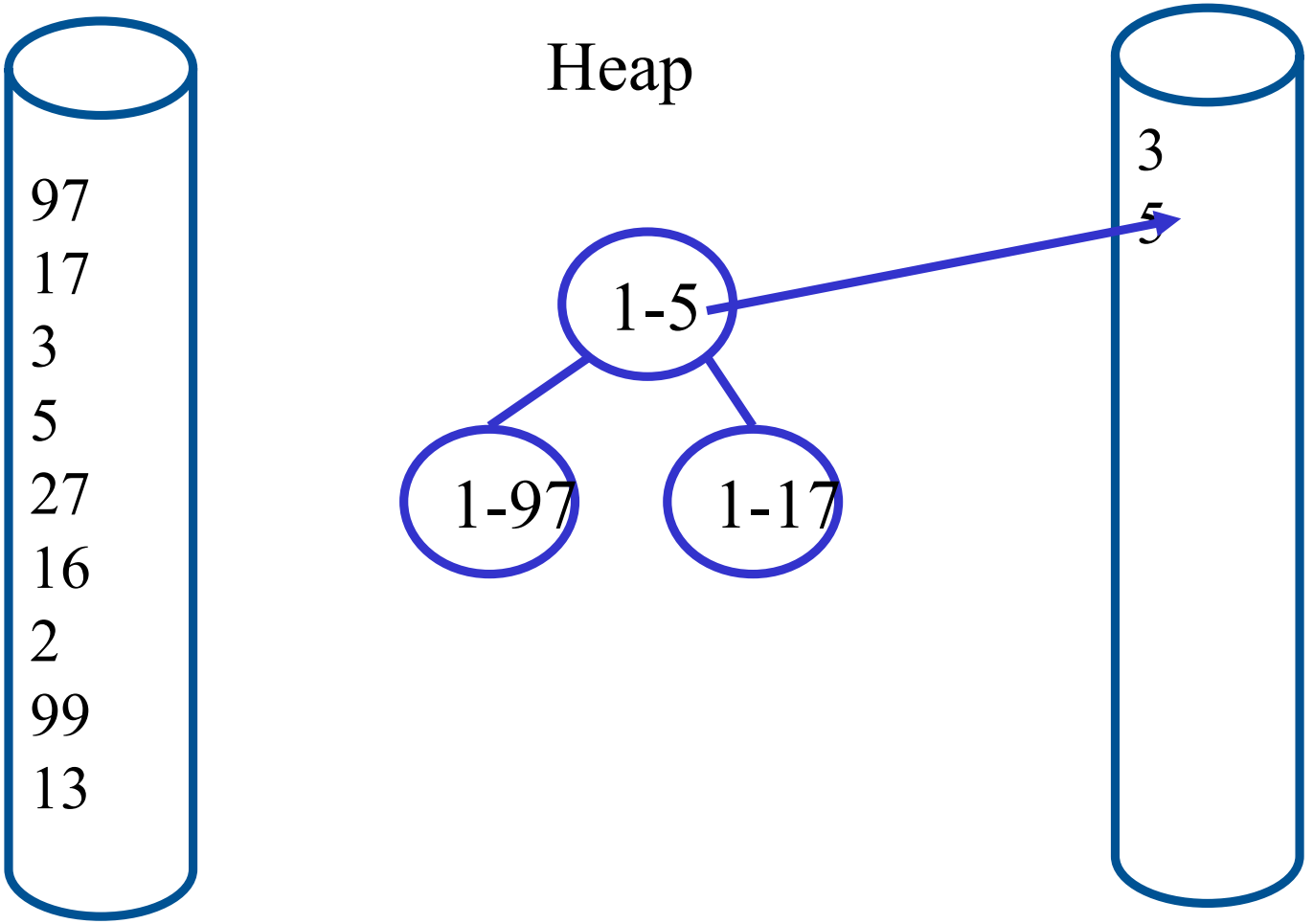
Replacement Selection während der Run-Generierung



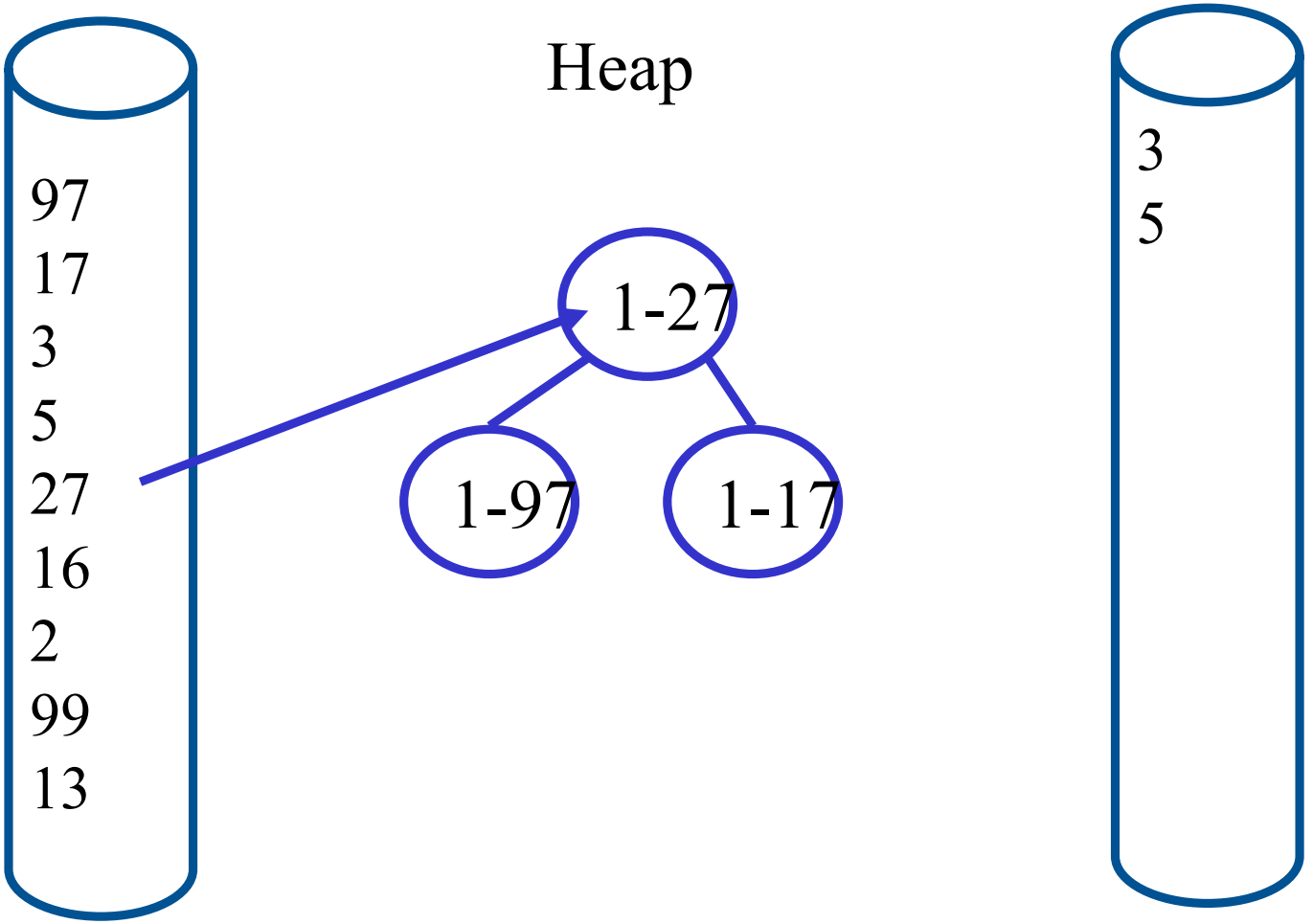
Replacement Selection während der Run-Generierung



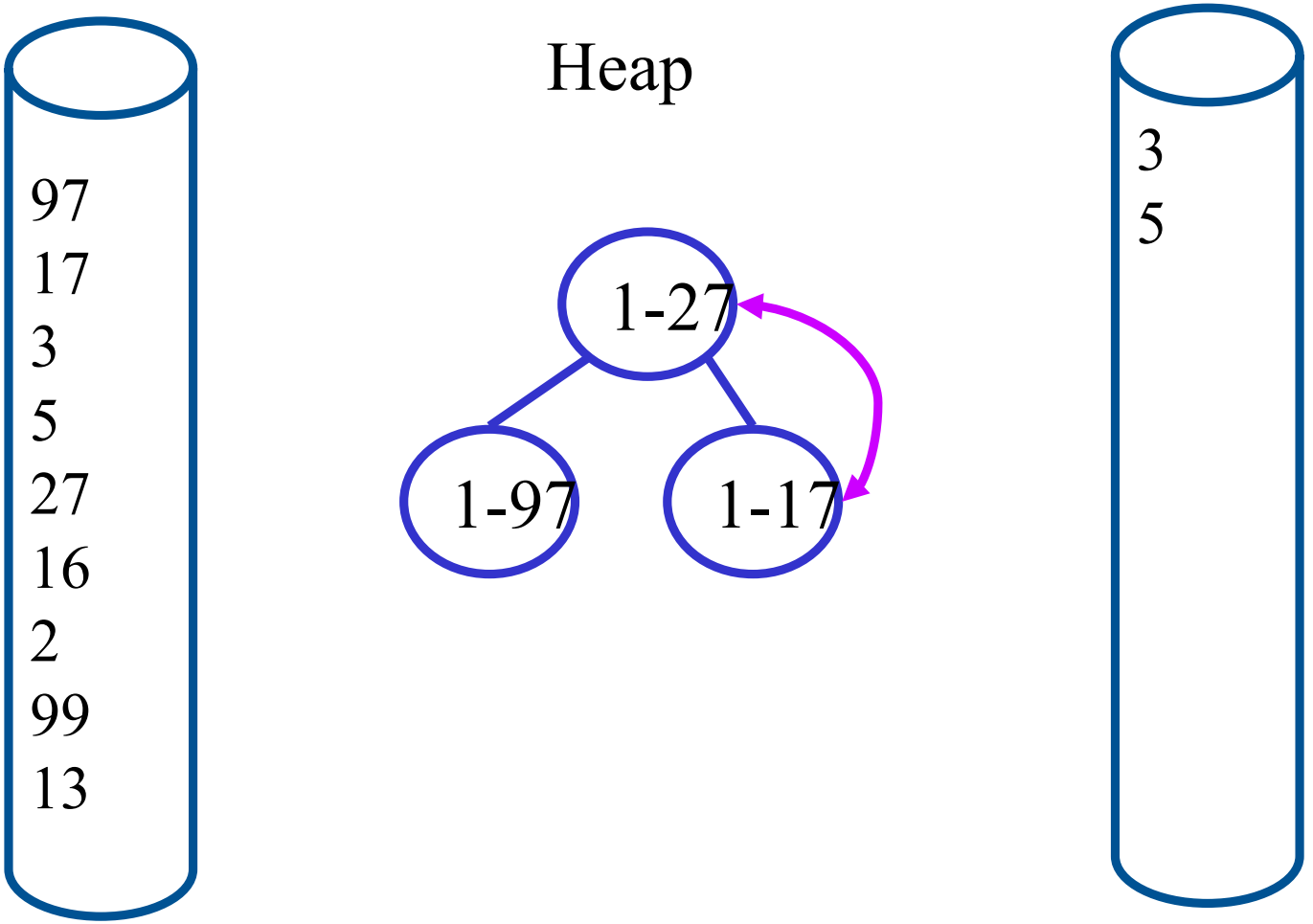
Replacement Selection während der Run-Generierung



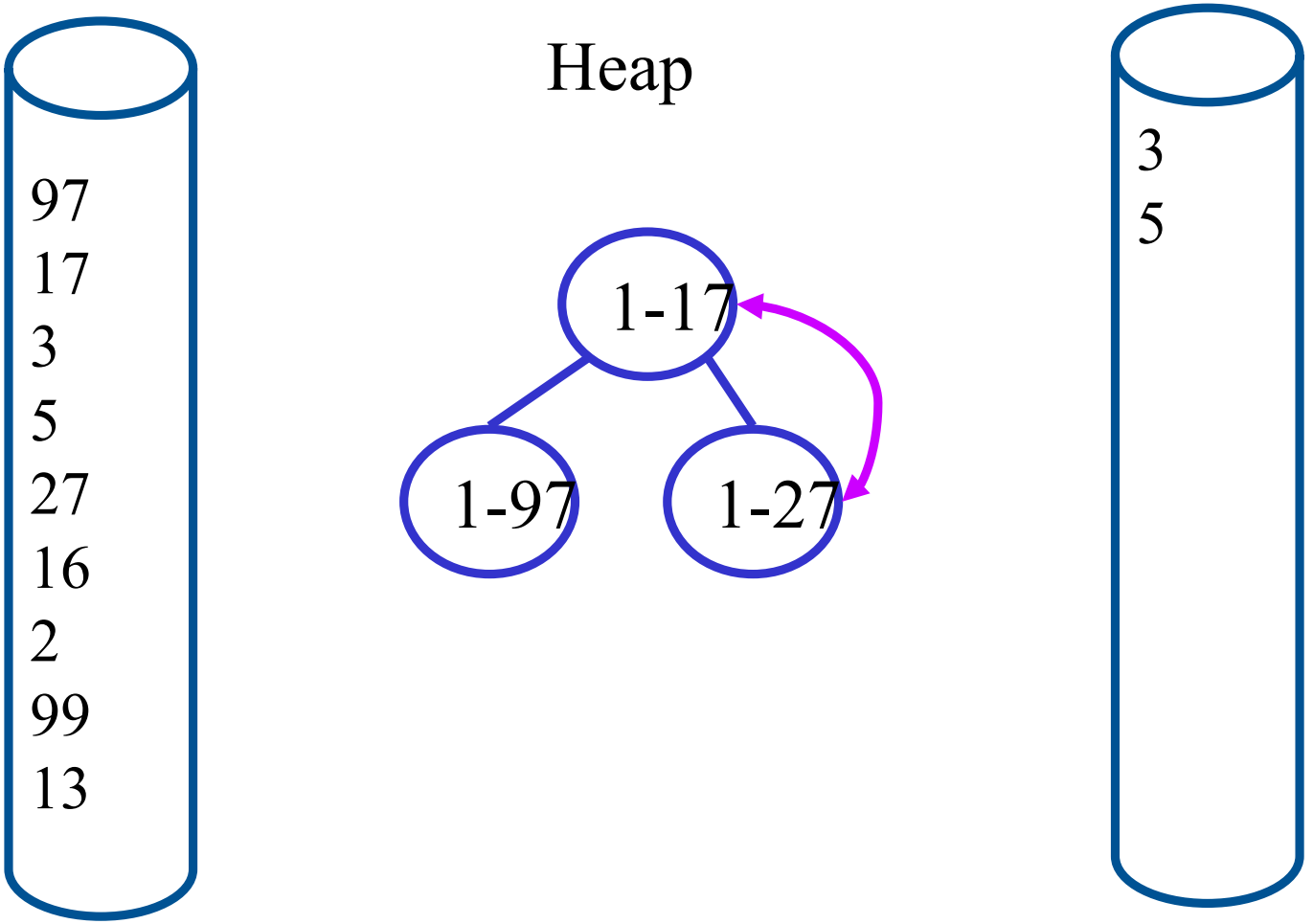
Replacement Selection während der Run-Generierung



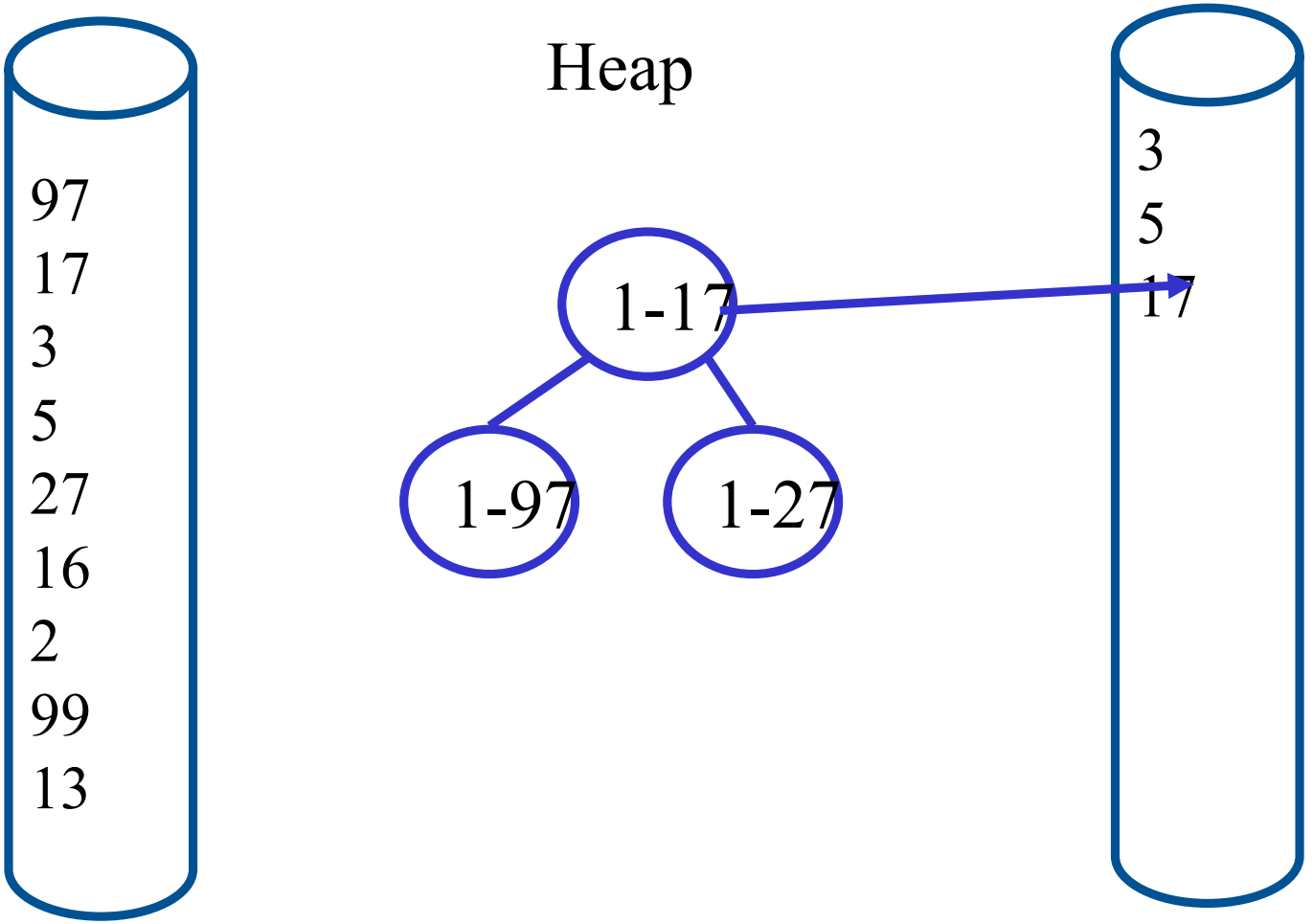
Replacement Selection während der Run-Generierung



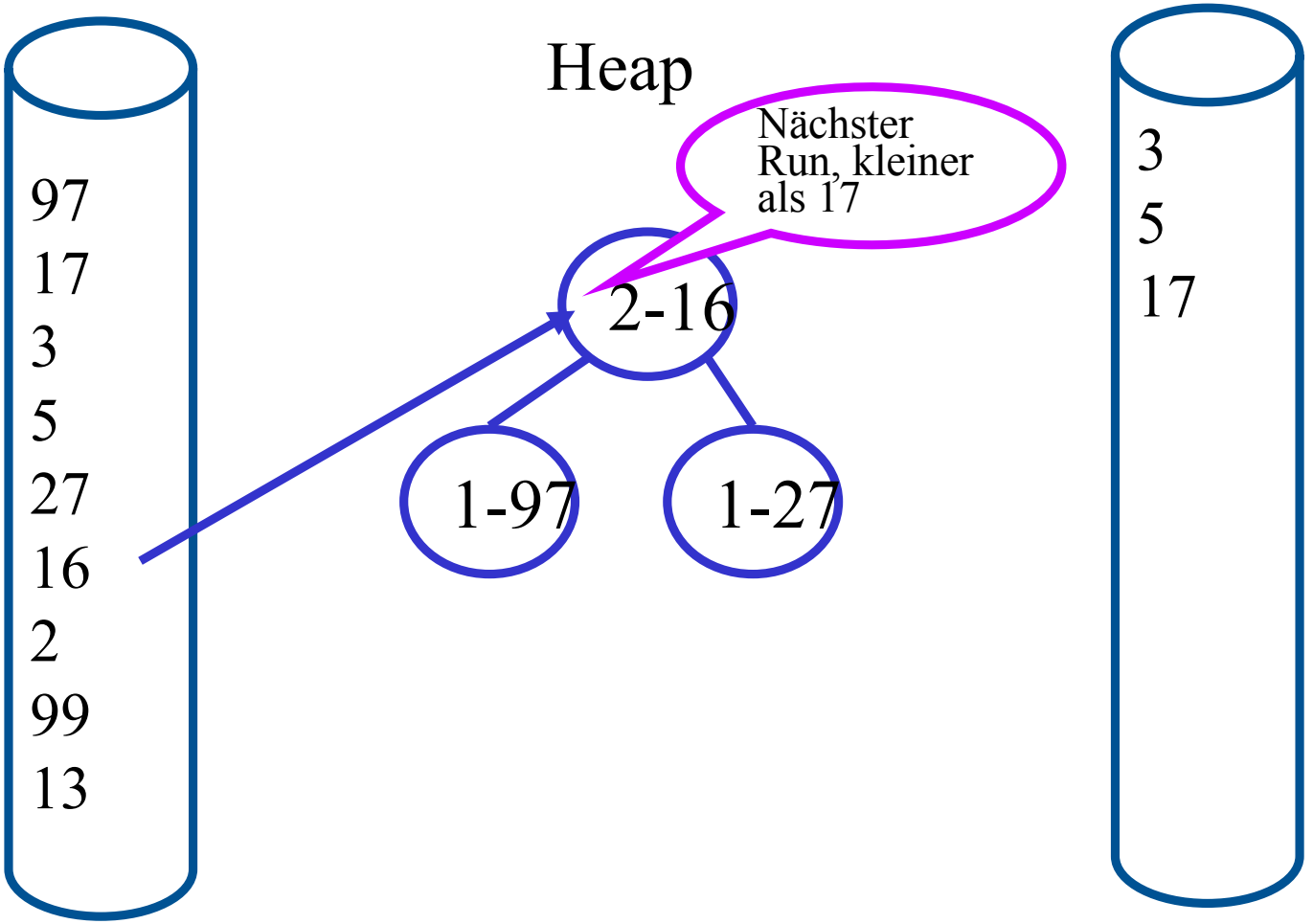
Replacement Selection während der Run-Generierung



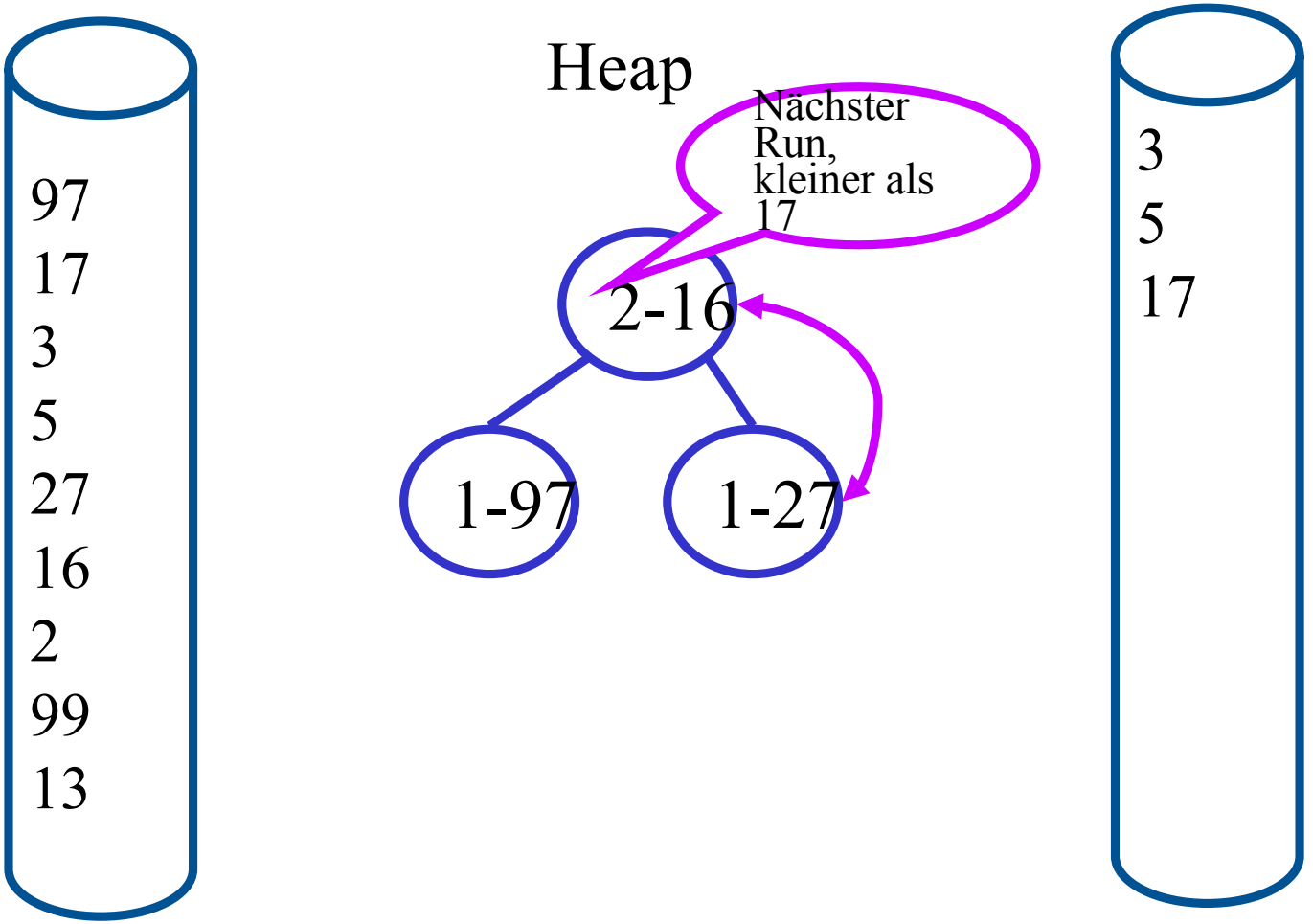
Replacement Selection während der Run-Generierung



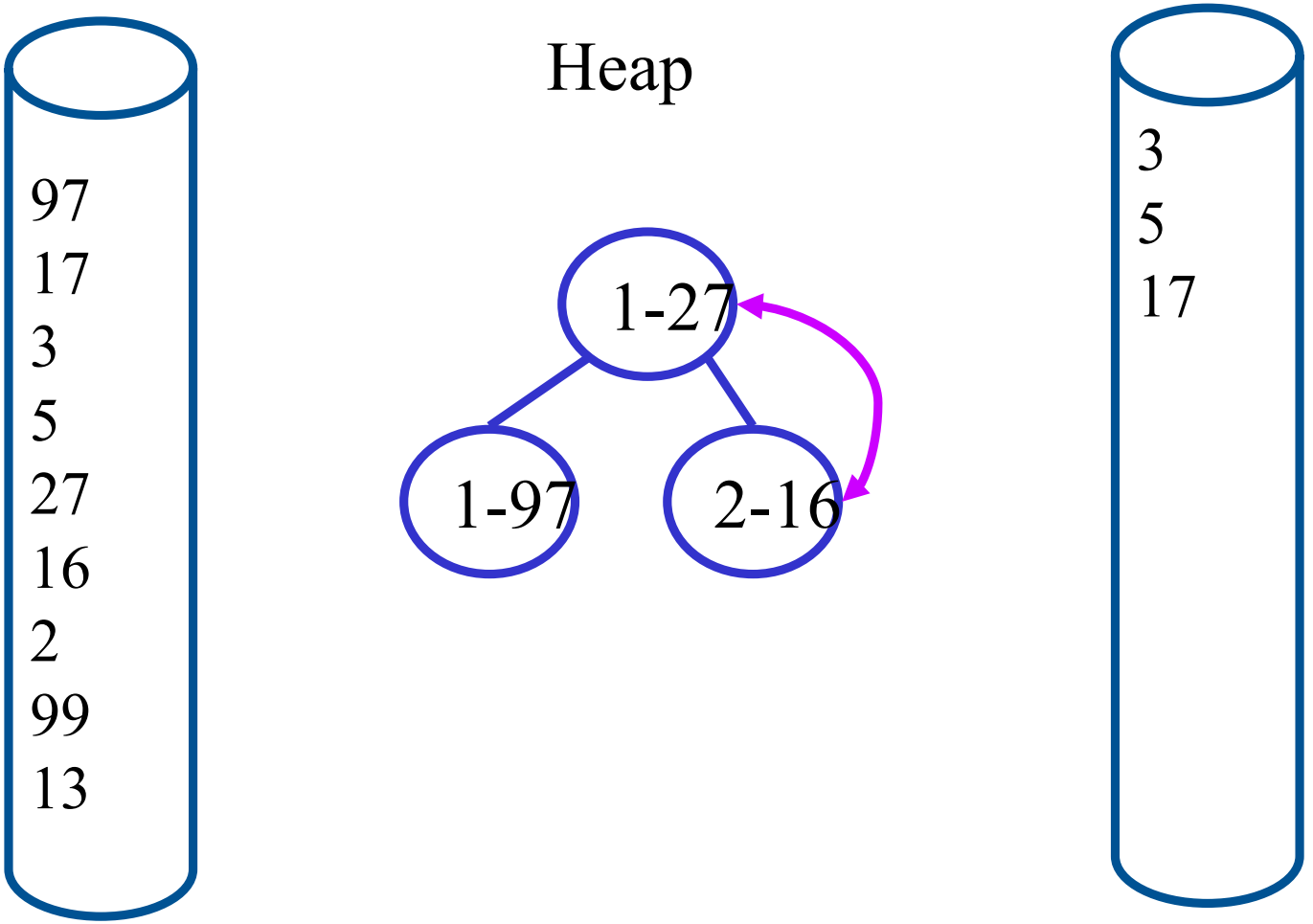
Replacement Selection während der Run-Generierung



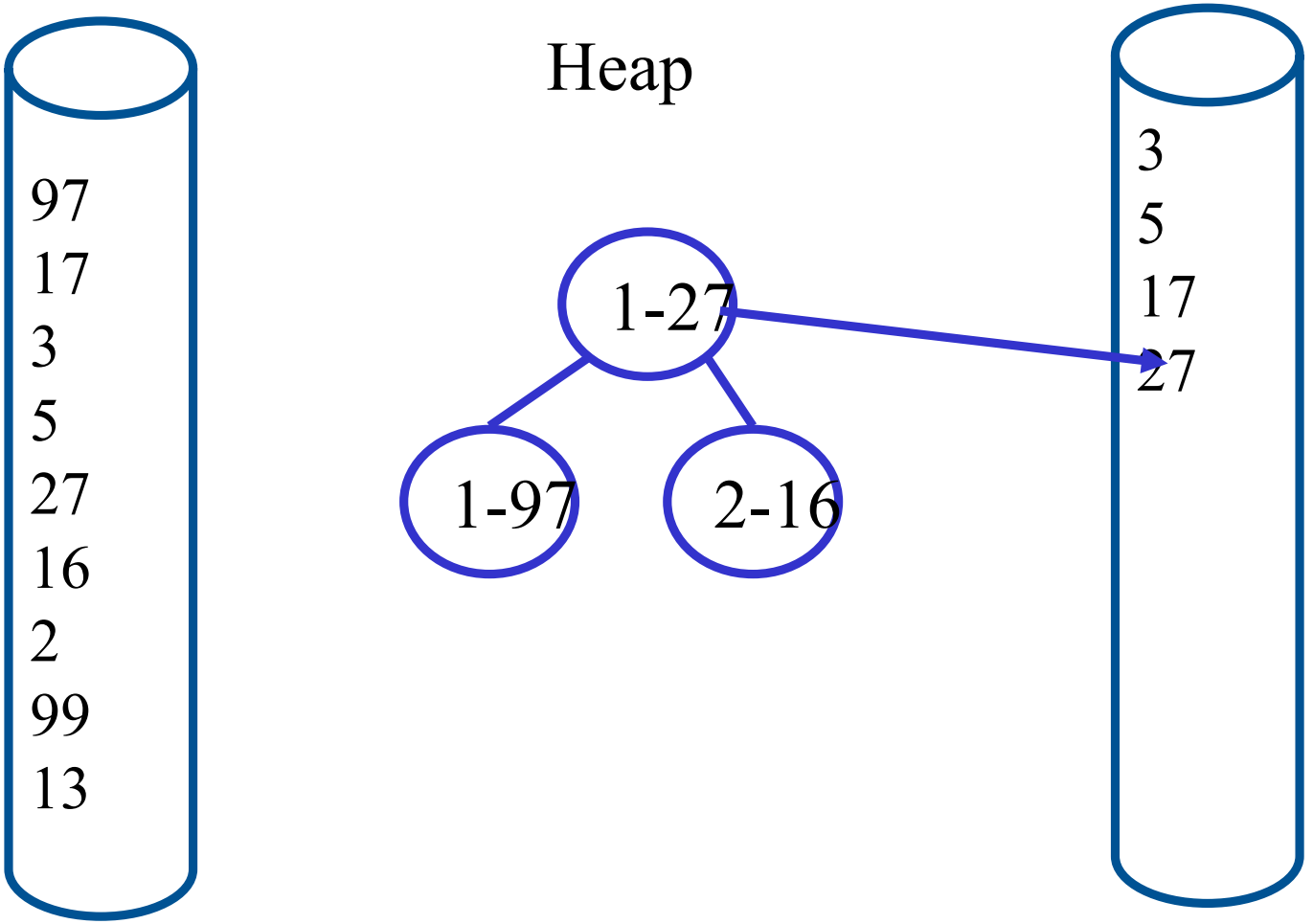
Replacement Selection während der Run-Generierung



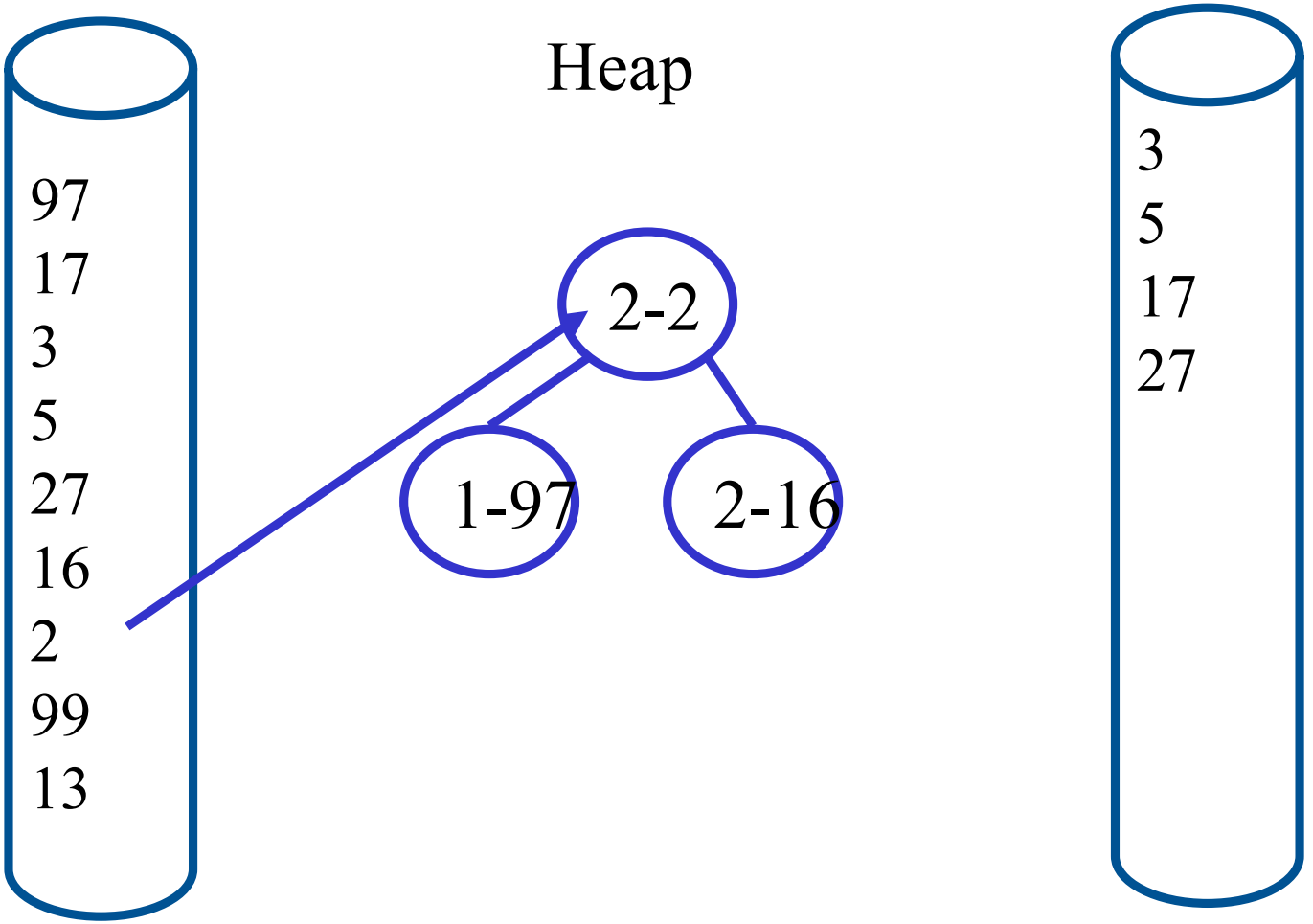
Replacement Selection während der Run-Generierung



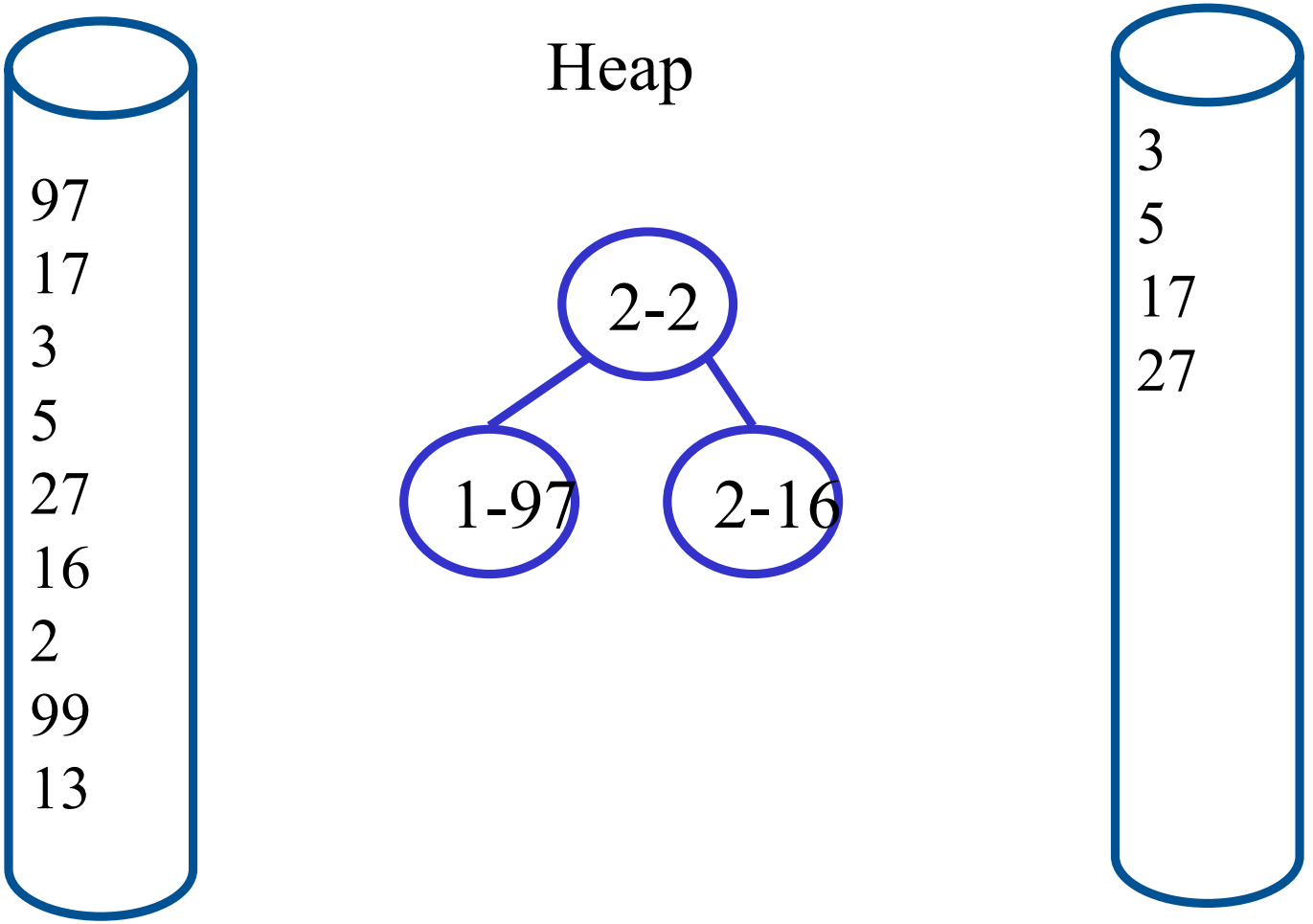
Replacement Selection während der Run-Generierung



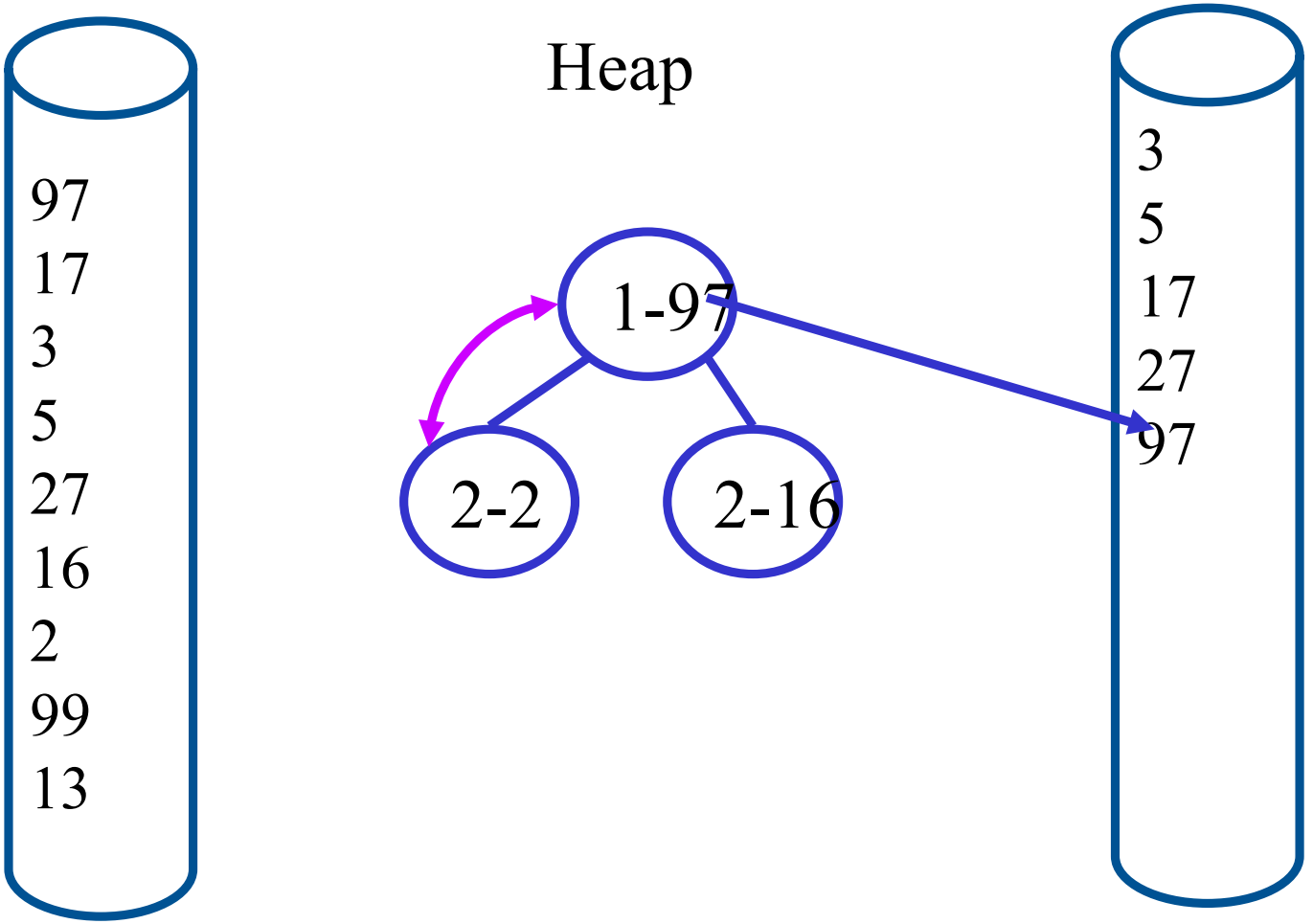
Replacement Selection während der Run-Generierung



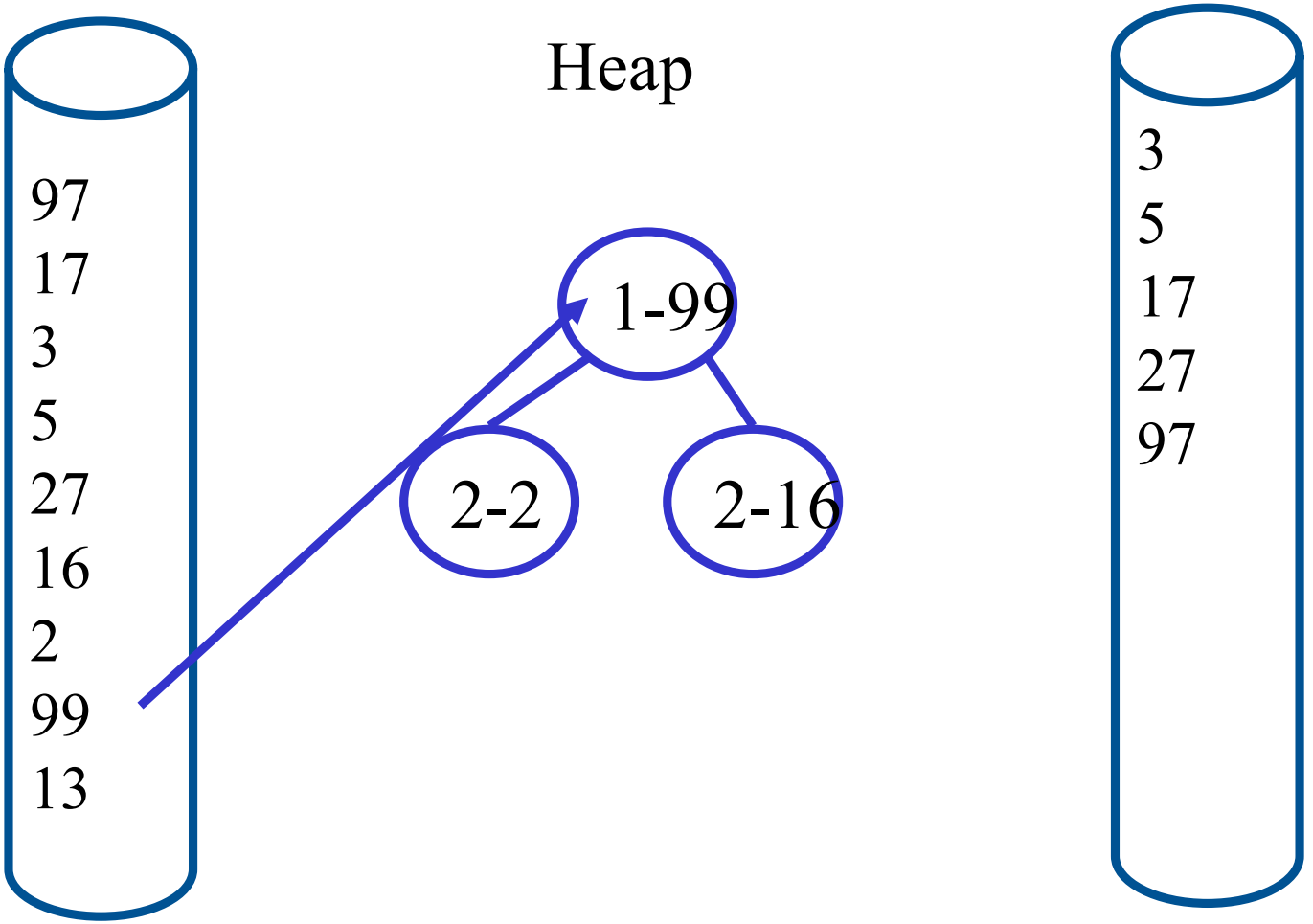
Replacement Selection während der Run-Generierung



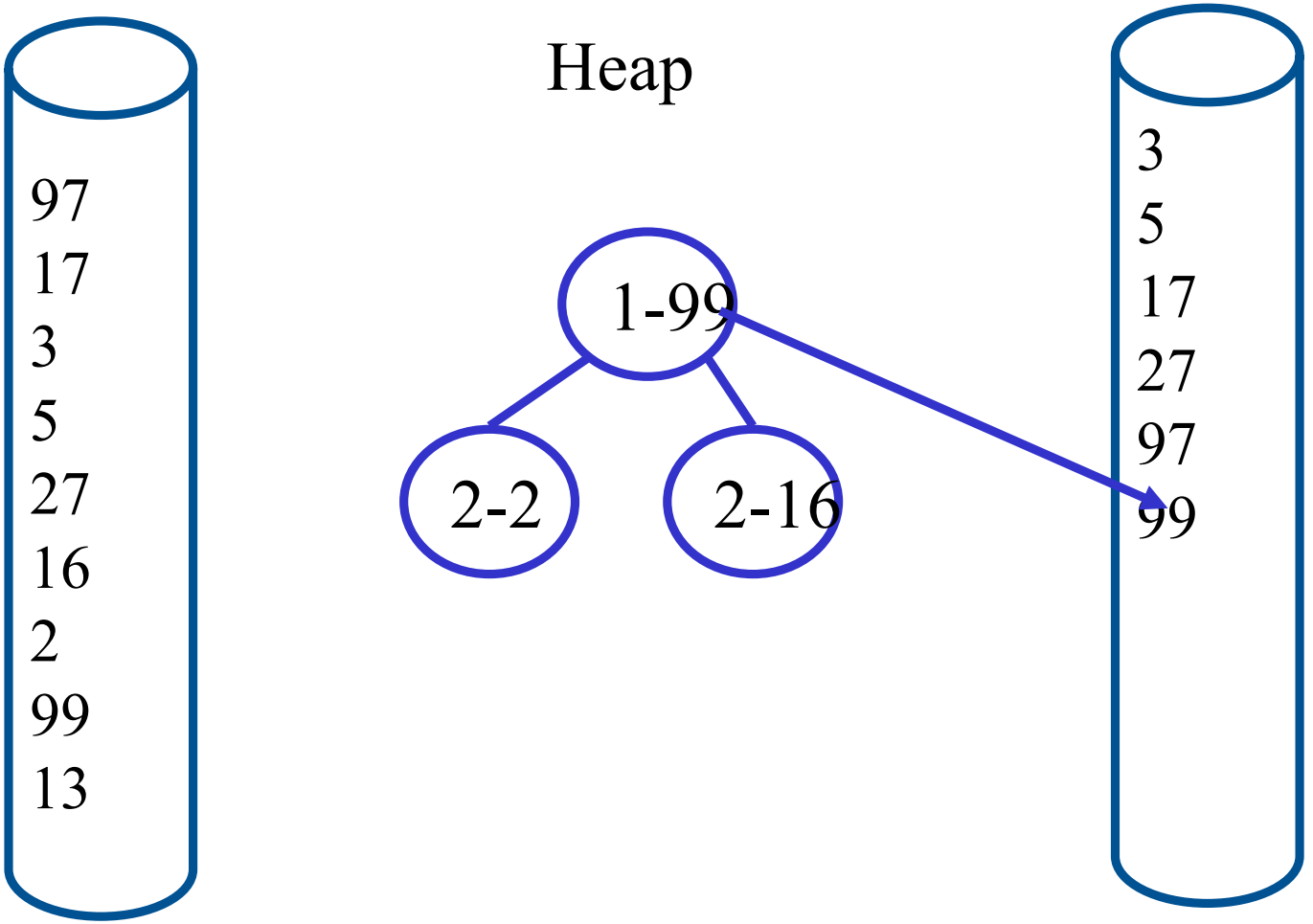
Replacement Selection während der Run-Generierung



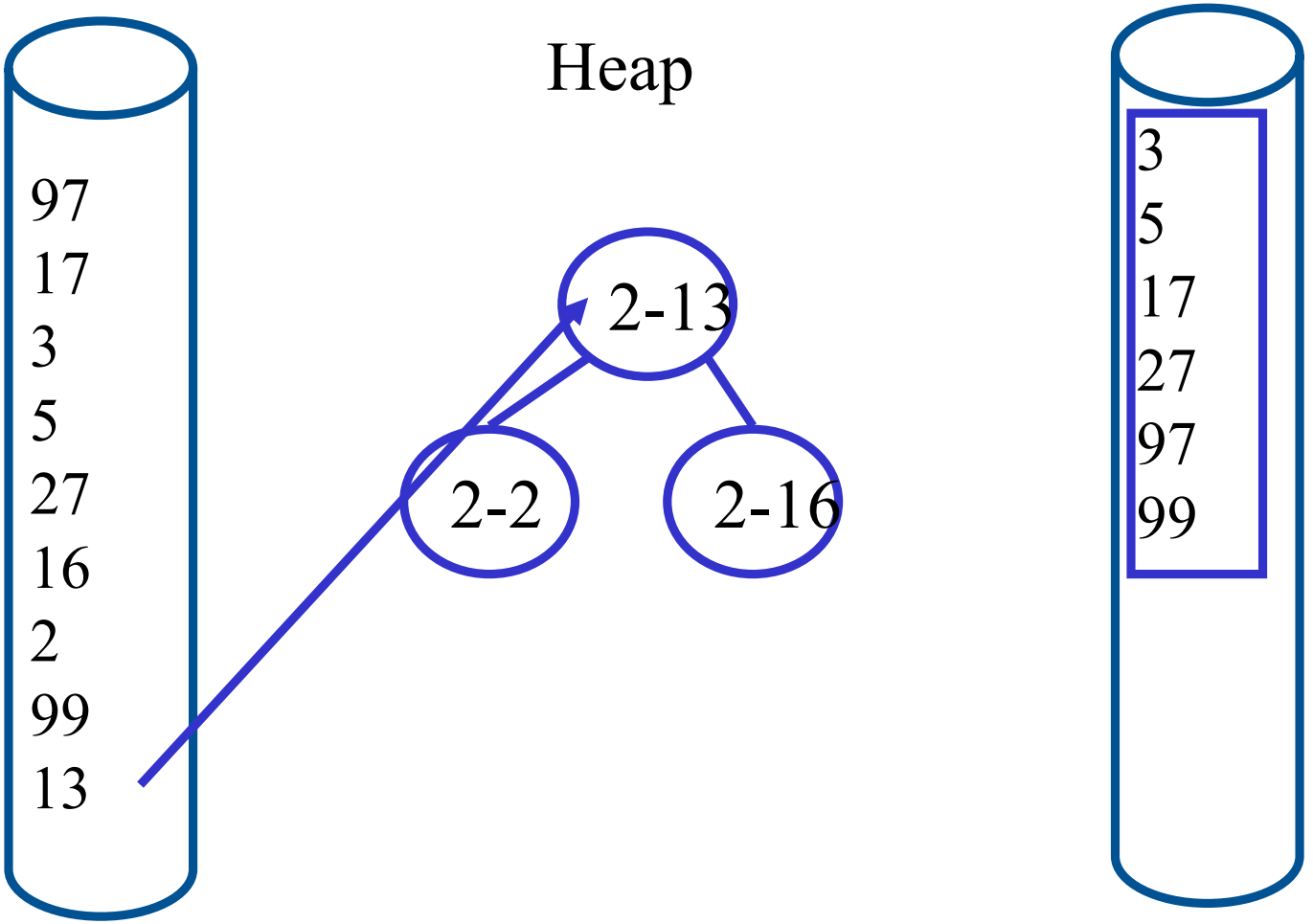
Replacement Selection während der Run-Generierung



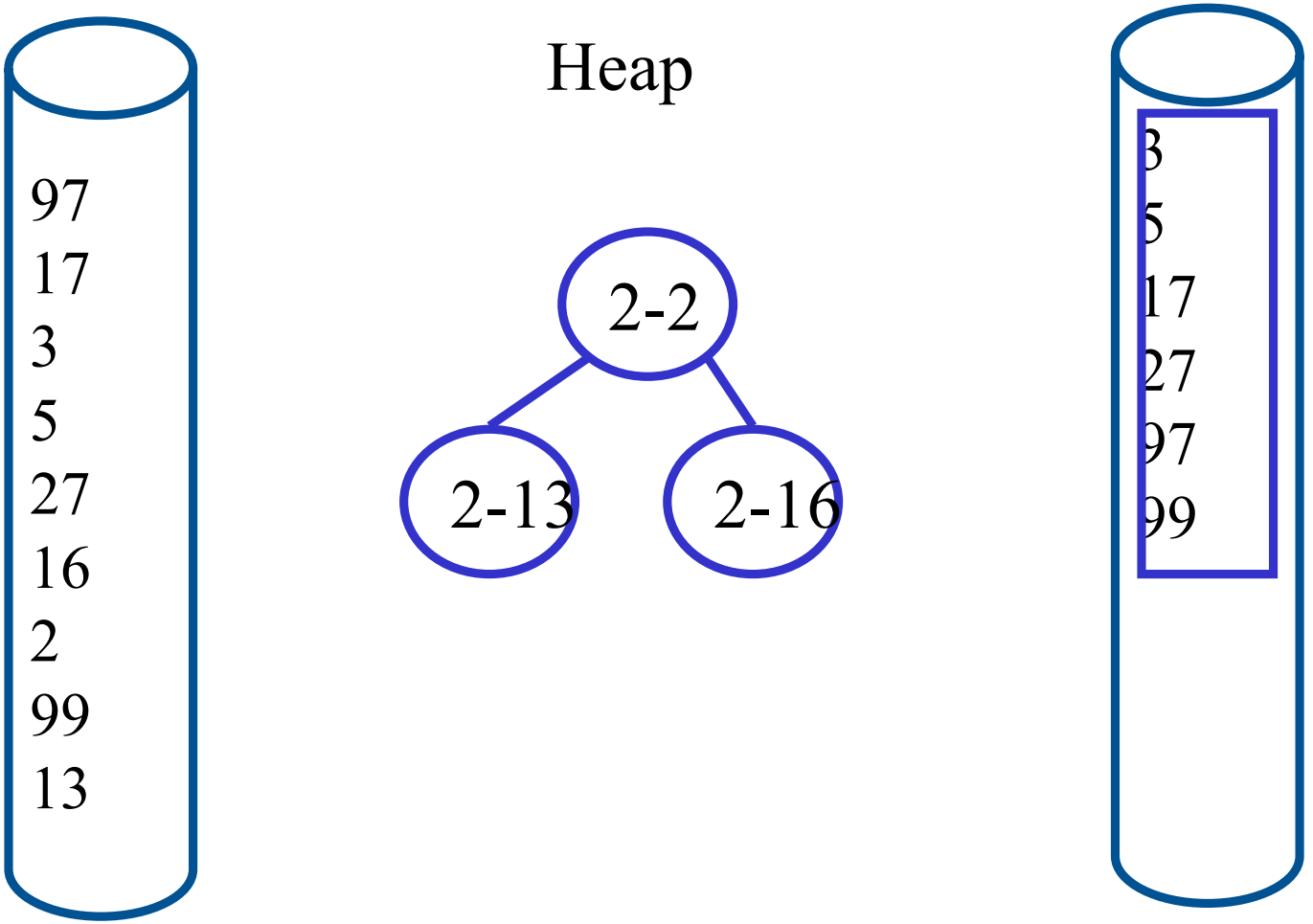
Replacement Selection während der Run-Generierung



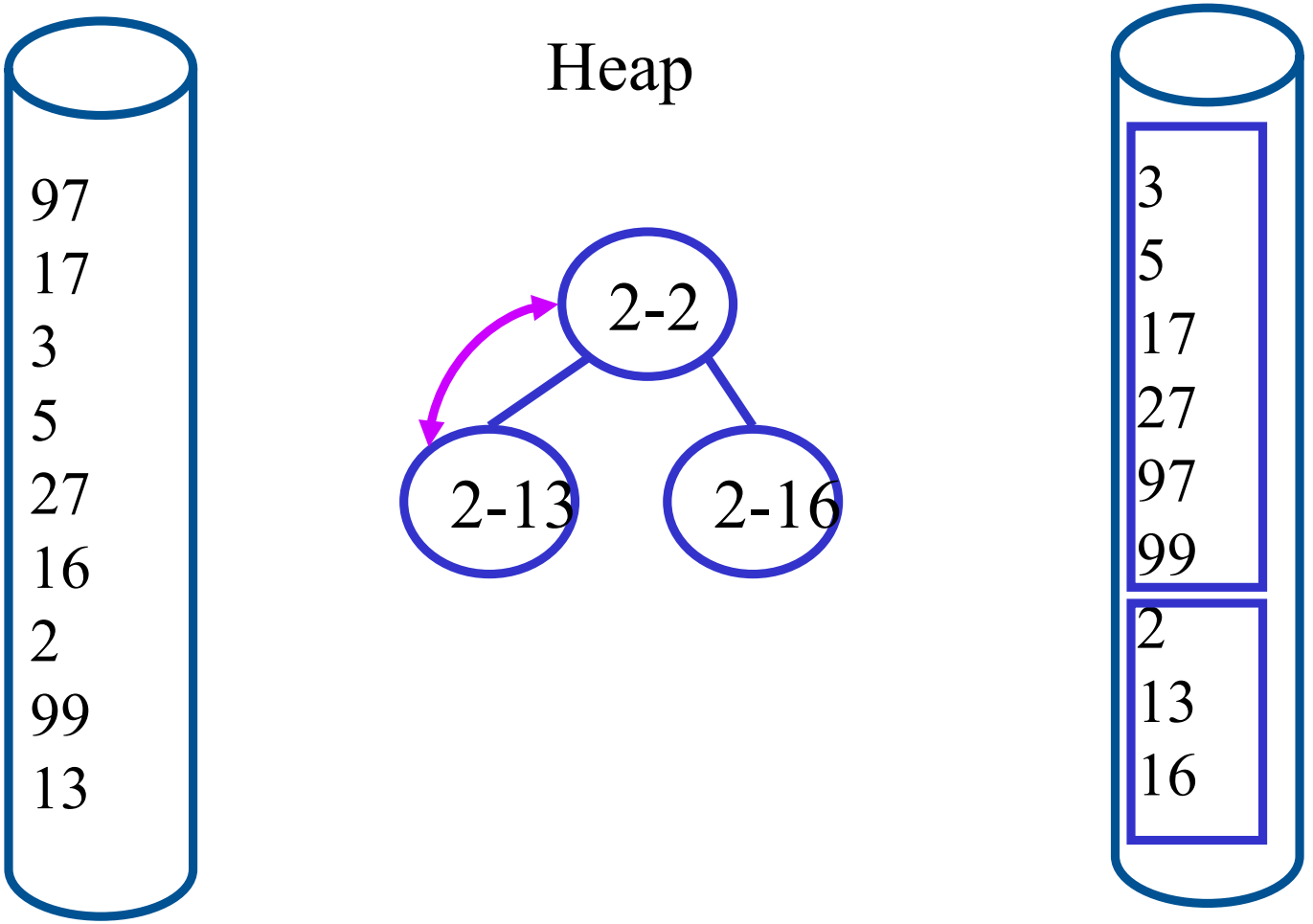
Replacement Selection während der Run-Generierung



Replacement Selection während der Run-Generierung



Replacement Selection während der Run-Generierung



Implementierungs-Details

Natürlich darf man nicht einzelne Datensätze zwischen Hauptspeicher und Hintergrundspeicher transferieren
- Jeder „Round-Trip" kostet viel Zeit (ca 10 ms)

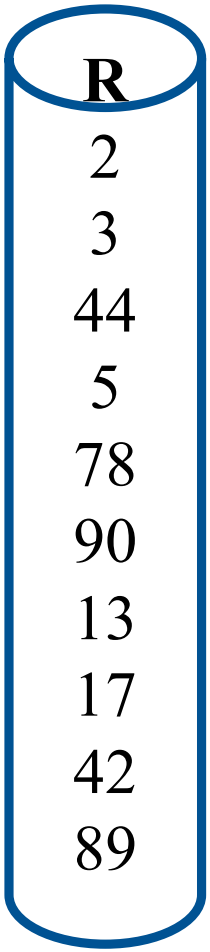
Man transferiert größere Blöcke
- Mindestens 8 KB Größe

Replacement Selection ist problematisch, wenn die zu sortierenden Datensätze variable Größe habe
- Der neue Datensatz passt dann nicht unbedingt in den frei gewordenen Platz, d.h., man benötigt eine aufwendigere Freispeicherverwaltung

Replacement Selection führt im Durchschnitt zu einer Verdoppelung der Run-Länge
- Beweis findet man im [Knuth]

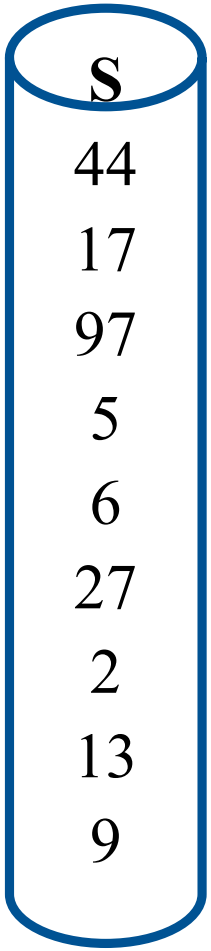
Komplexität des externen Sortierens? O(N log N) ??

Algorithmen auf sehr großen Datenmengen

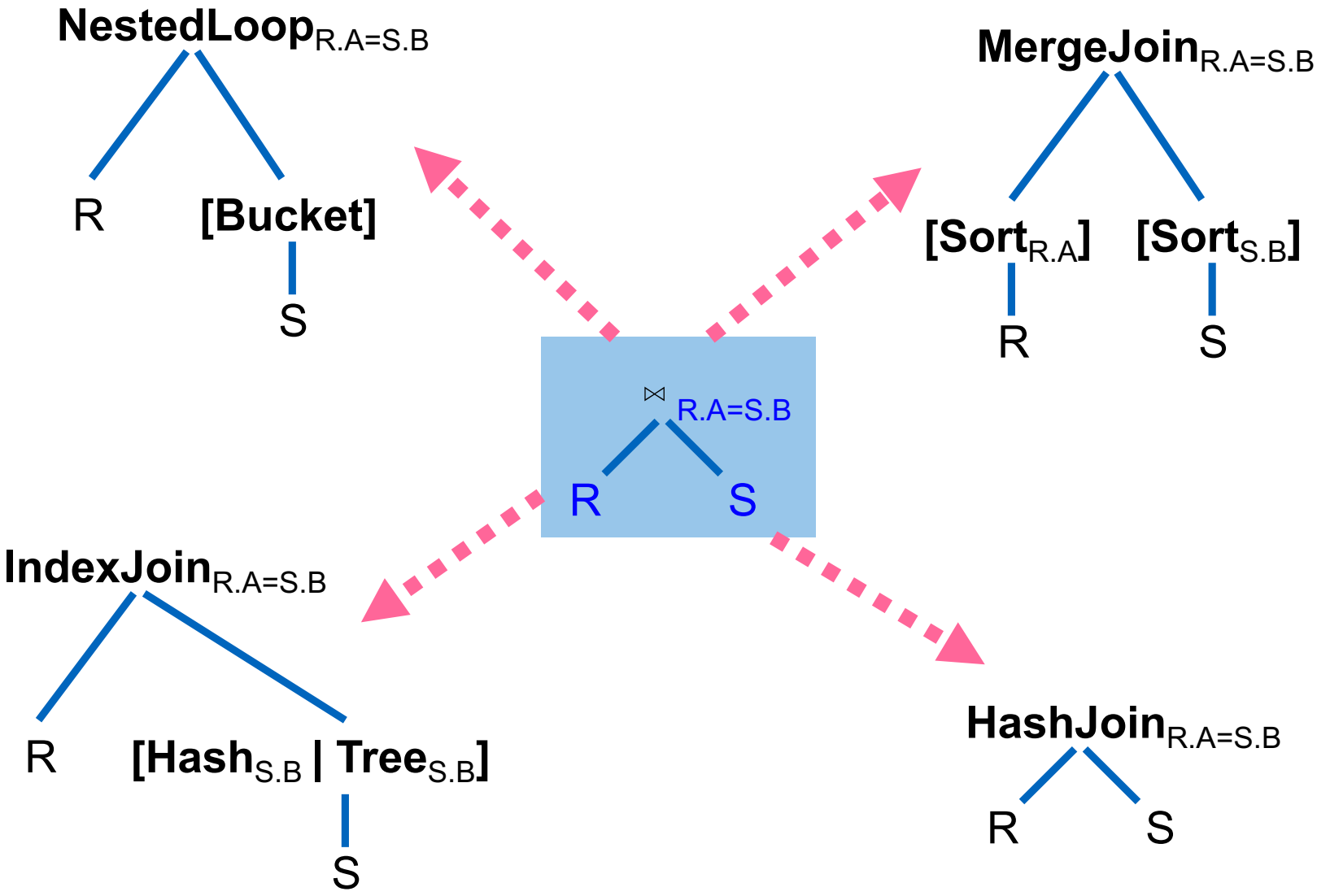


$R \cap S$

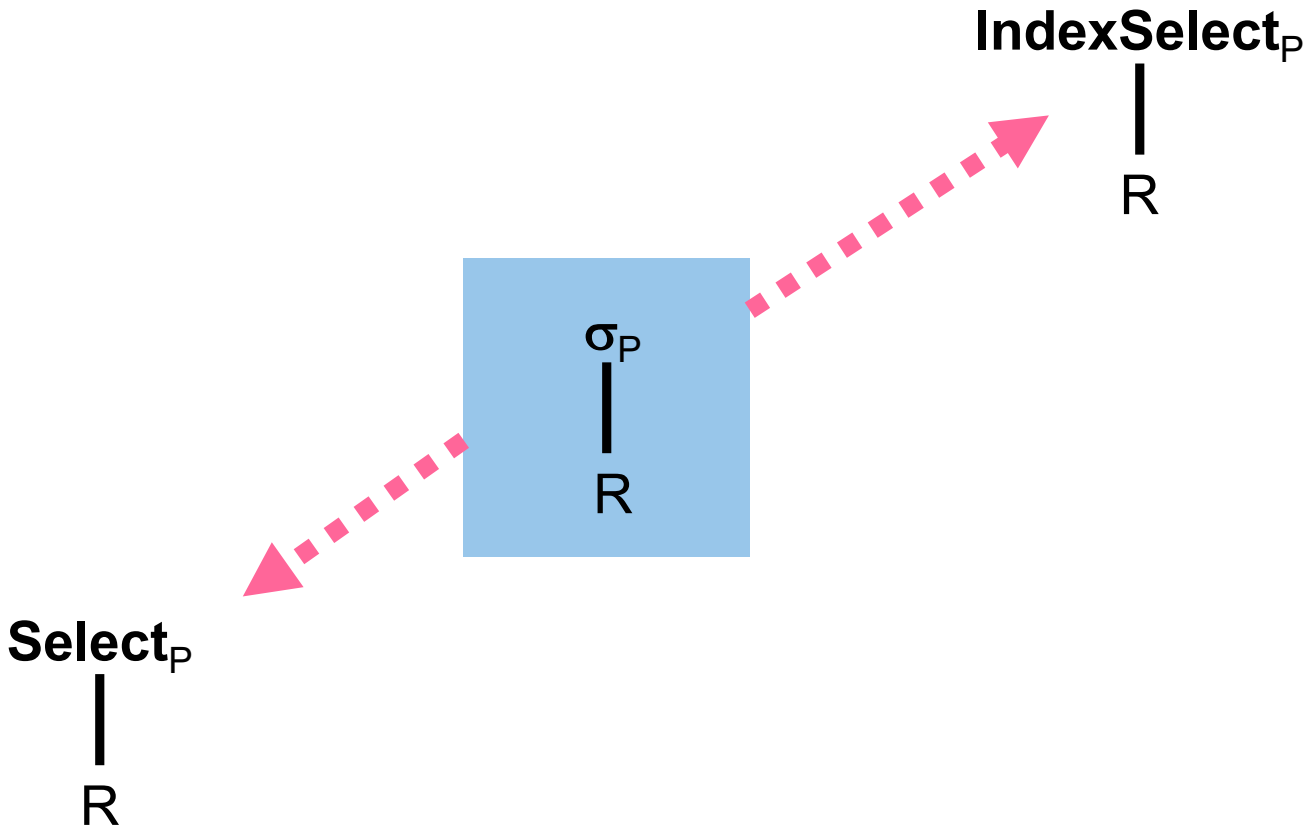
- Nested Loop: $O(N^2)$
- Sortieren: $O(N \log N)$
- Partitionieren und Hashing



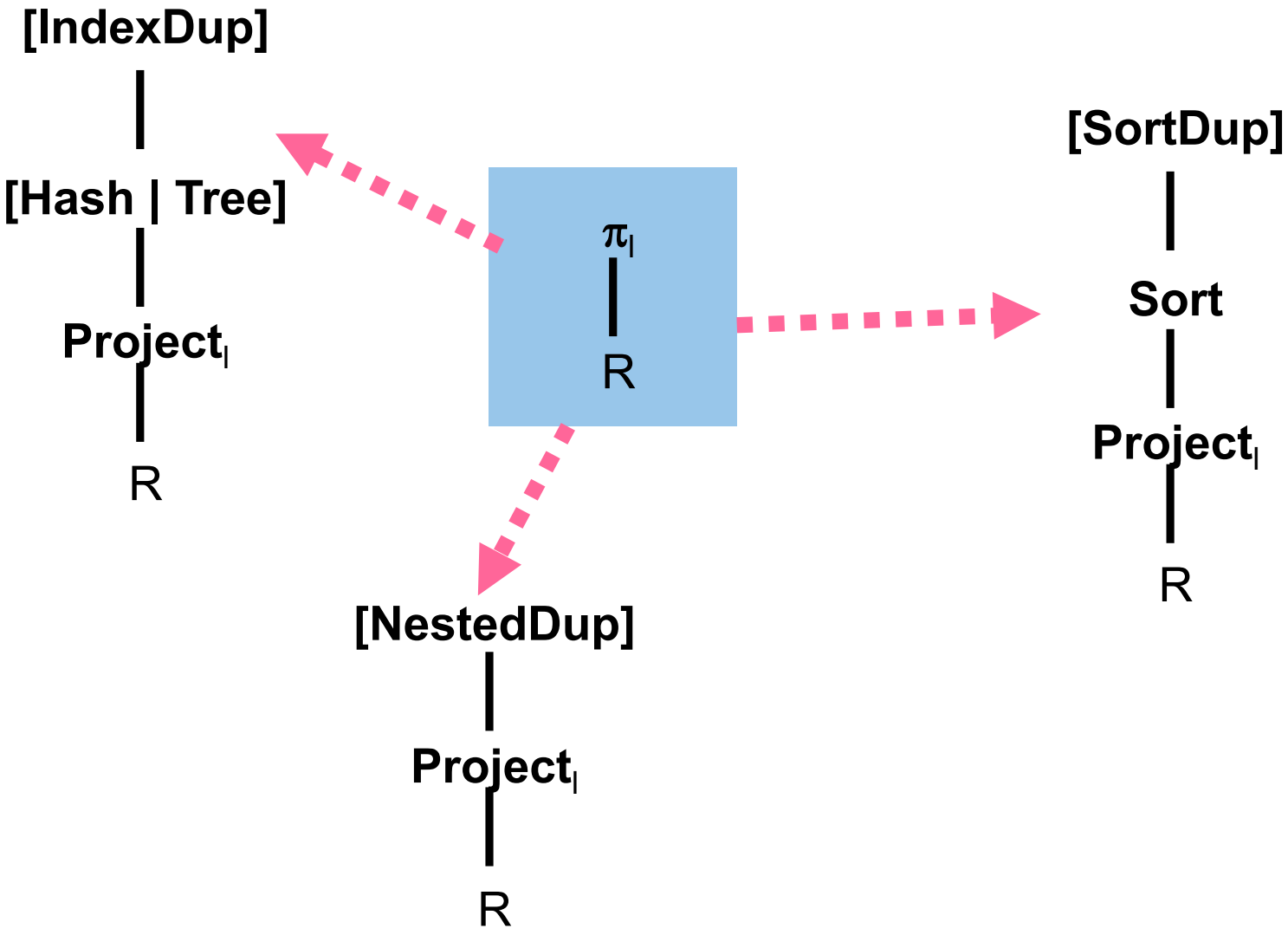
Übersetzung der logischen Algebra



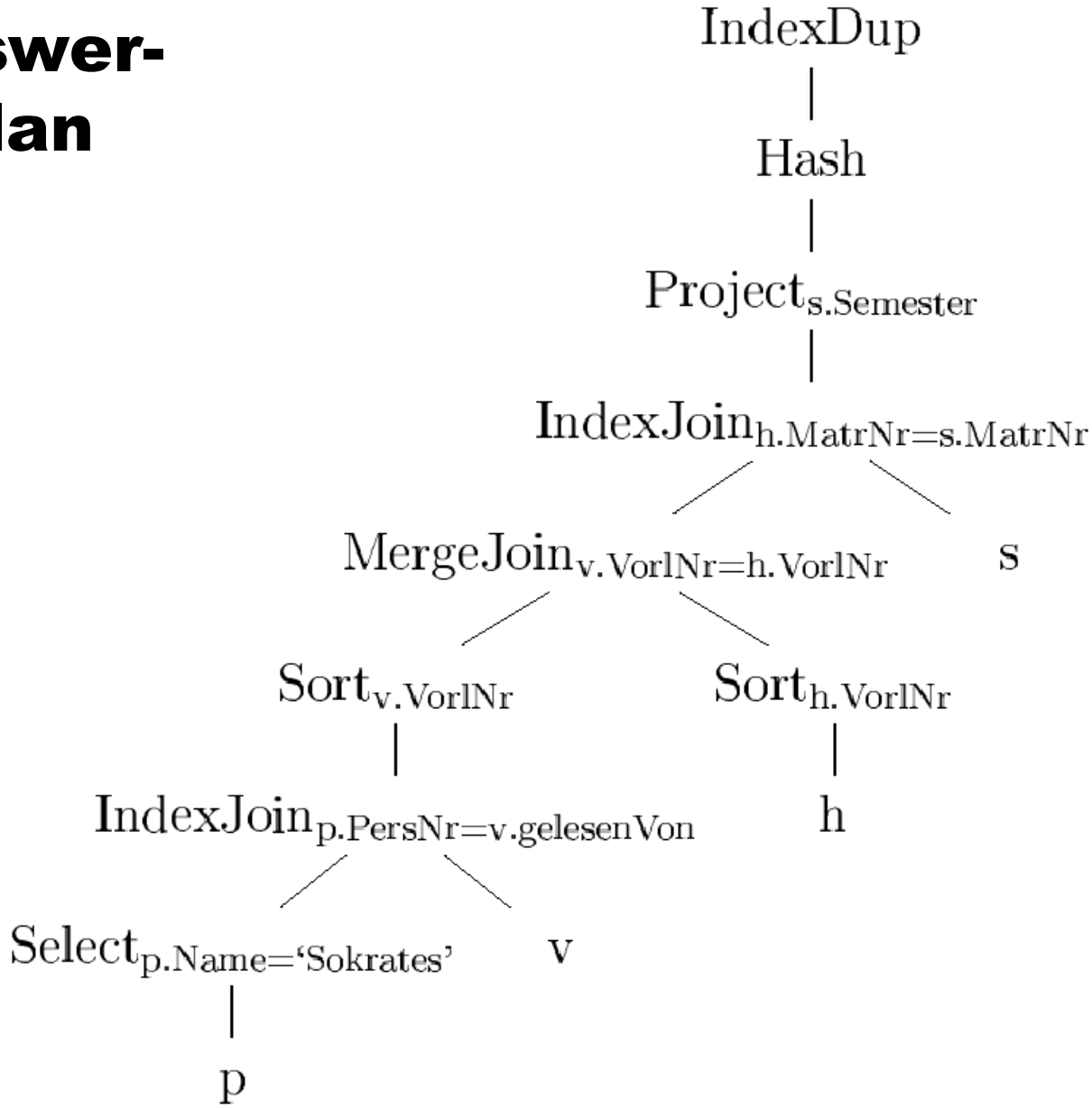
Übersetzung der logischen Algebra



Übersetzung der logischen Algebra

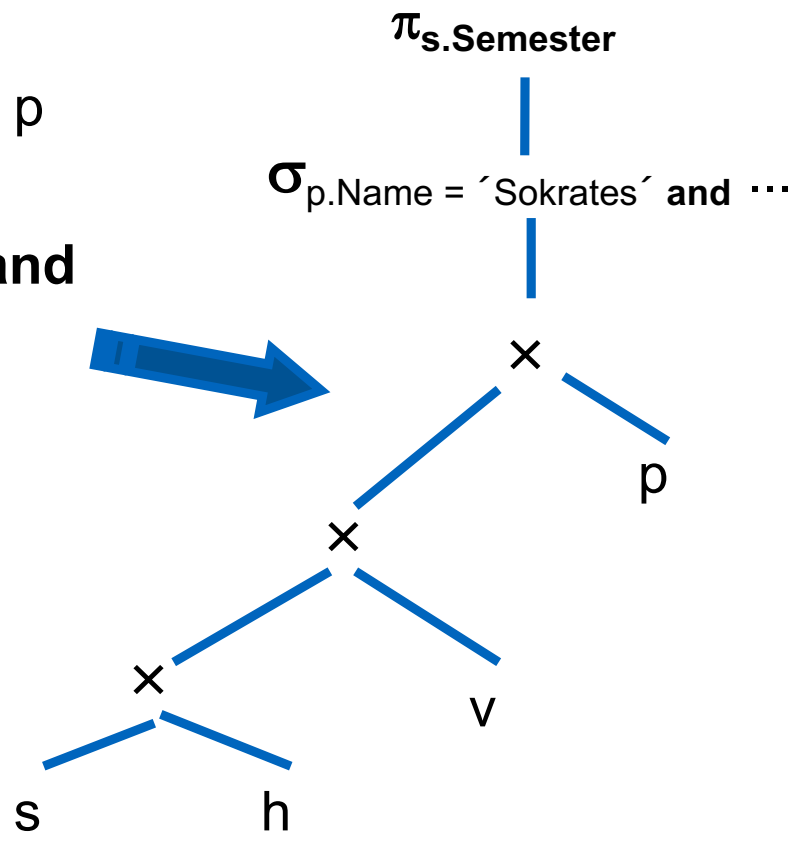


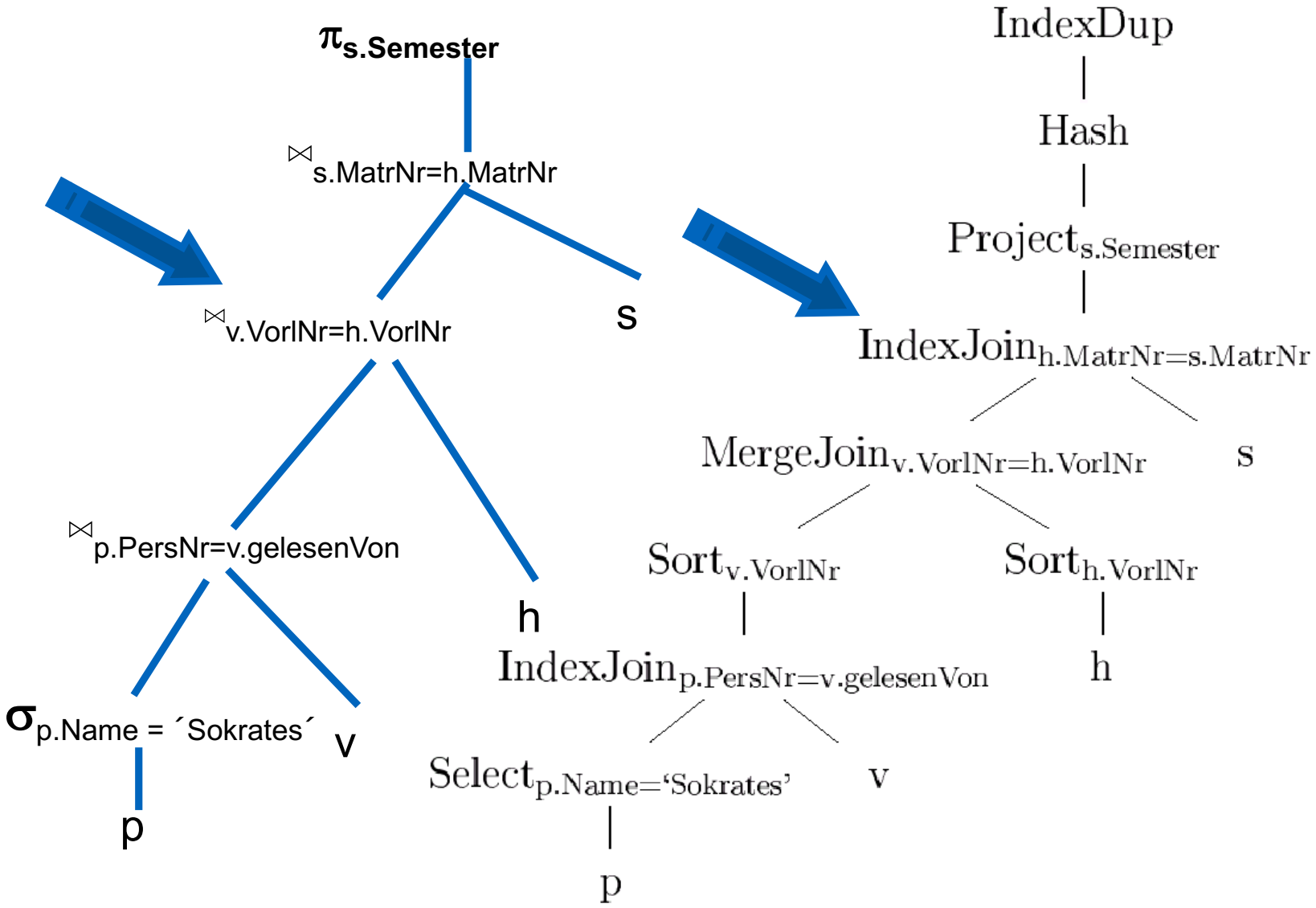
Ein Auswer-tungsplan



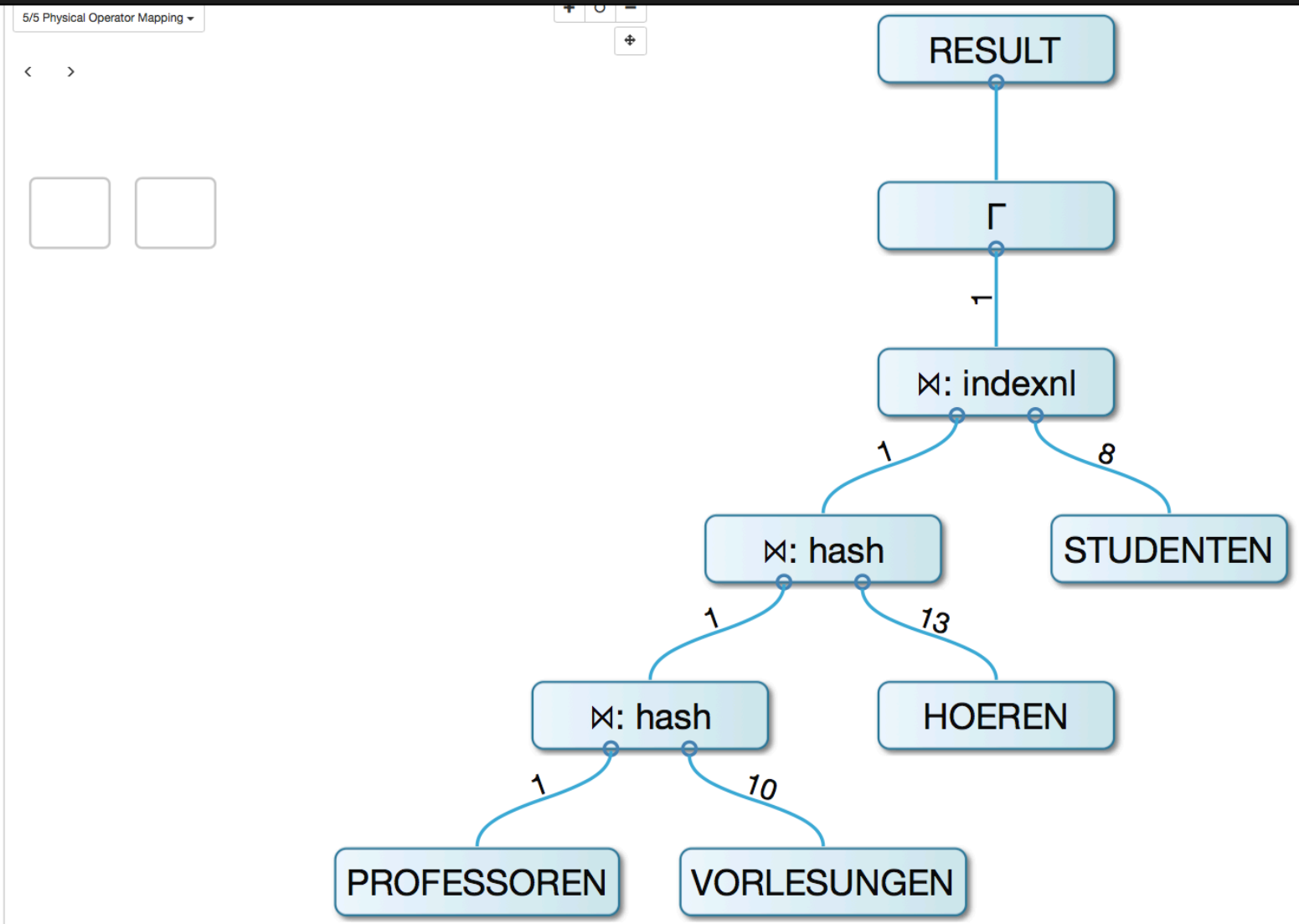
Wiederholung der Optimierungsphasen

```
select distinct s.Semester
from Studenten s, hören h,
        Vorlesungen v, Professoren p
where p.Name = 'Sokrates' and
        v.gelesenVon = p.PersNr and
        v.VorlNr = h.VorlNr and
        h.MatrNr = s.MatrNr
```




πs.Semester
⋈s.MatrNr=h.MatrNr
⋈v.VorlNr=h.VorlNr
s
⋈p.PersNr=v.gelesenVon
h
σp.Name = ´Sokrates´
v
p
IndexDup
Hash
Projects.Semester
IndexJoinh.MatrNr=s.MatrNr
MergeJoinv.VorlNr=h.VorlNr
s
Sortv.VorlNr
Sorth.VorlNr
IndexJoinp.PersNr=v.gelesenVon
h
Selectp.Name='Sokrates'
v
p

HyPer


5/5 Physical Operator Mapping
RESULT
Γ
1
⋈: indexnl
1
8
⋈: hash
STUDENTEN
1
13
⋈: hash
HOEREN
1
10
PROFESSOREN
VORLESUNGEN

Kostenbasierte Optimierung

Generiere alle denkbaren Anfrageausertungspläne
- Enumeration

Bewerte deren Kosten
- Kostenmodell
- Statistiken
- Histogramme
- Kalibrierung gemäß verwendetem Rechner
- Abhängig vom verfügbaren Speicher
- Aufwands-Kostenmodell
  - Durchsatz-maximierend
  - Nicht (immer) Antwortzeit-minimierend

Behalte den billigsten Plan

Problemgröße

Suchraum (Planstruktur)

Bushy-Pläne mit *n* Tabellen [Ganguly et al. 1992]:

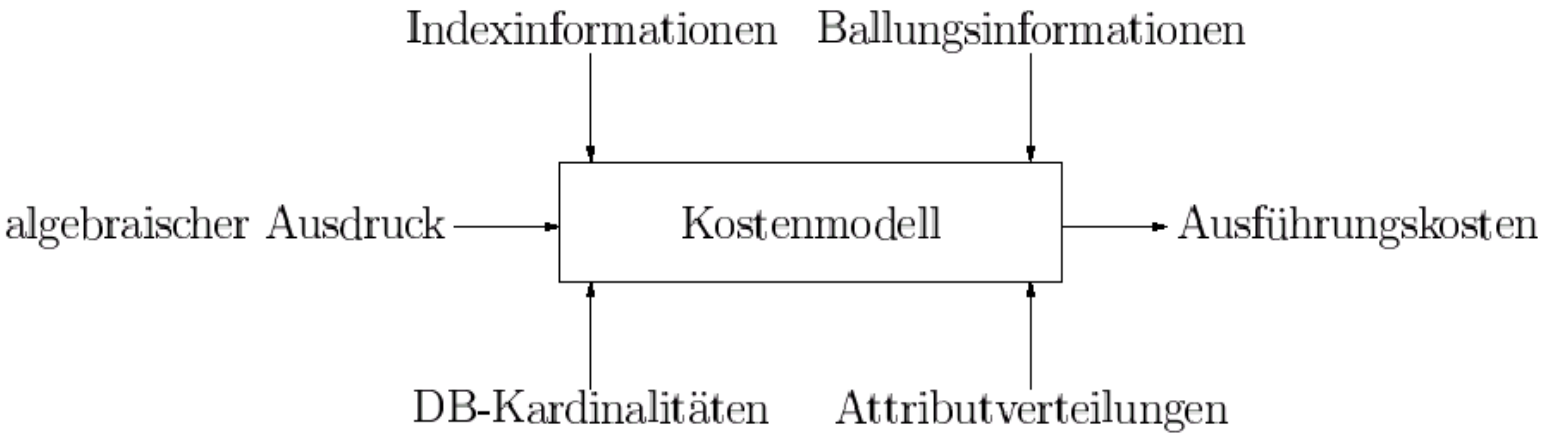
$$\frac{(2(n\text{-}1))!}{(n\text{-}1)!}$$

n	e^n	$(2(n\text{-}1))!/(n\text{-}1)!$
2	7	2
5	146	1680
10	22026	$1{,}76*10^{10}$
20	$4{,}85 * 10^9$	$4{,}3*10^{27}$

Plankosten unterscheiden sich um Größenordnungen

Optimierungsproblem ist *NP*-hart [Ibaraki 1984]

Kostenmodelle



Selektivität

Sind verschiedene Strategien anwendbar, so benötigt man zur Auswahl eine Kostenfunktion. Sie basiert auf dem Begriff der Selektivität.

Die Selektivität eines Suchprädikats schätzt die Anzahl der qualifizierenden Tupel relativ zur Gesamtanzahl der Tupel in der Relation.

Beispiele:

- die Selektivität einer Anfrage, die das Schlüsselattribut einer Relation R spezifiziert, ist $1/\ \#R$, wobei $\#R$ die Kardinalität der Relation R angibt.

- Wenn ein Attribut A spezifiziert wird, für das i verschiedene Werte existieren, so kann die Selektivität als

  $(\#R/i)\ /\ \#R$ oder $1/i$

  abgeschätzt werden.

Selektivitäten

- Anteil der qualifizierenden Tupel einer Operation
- Selektion mit Bedingung p:

$$sel_p := \frac{|\sigma_p(R)|}{|R|}$$

- Join von R mit S:

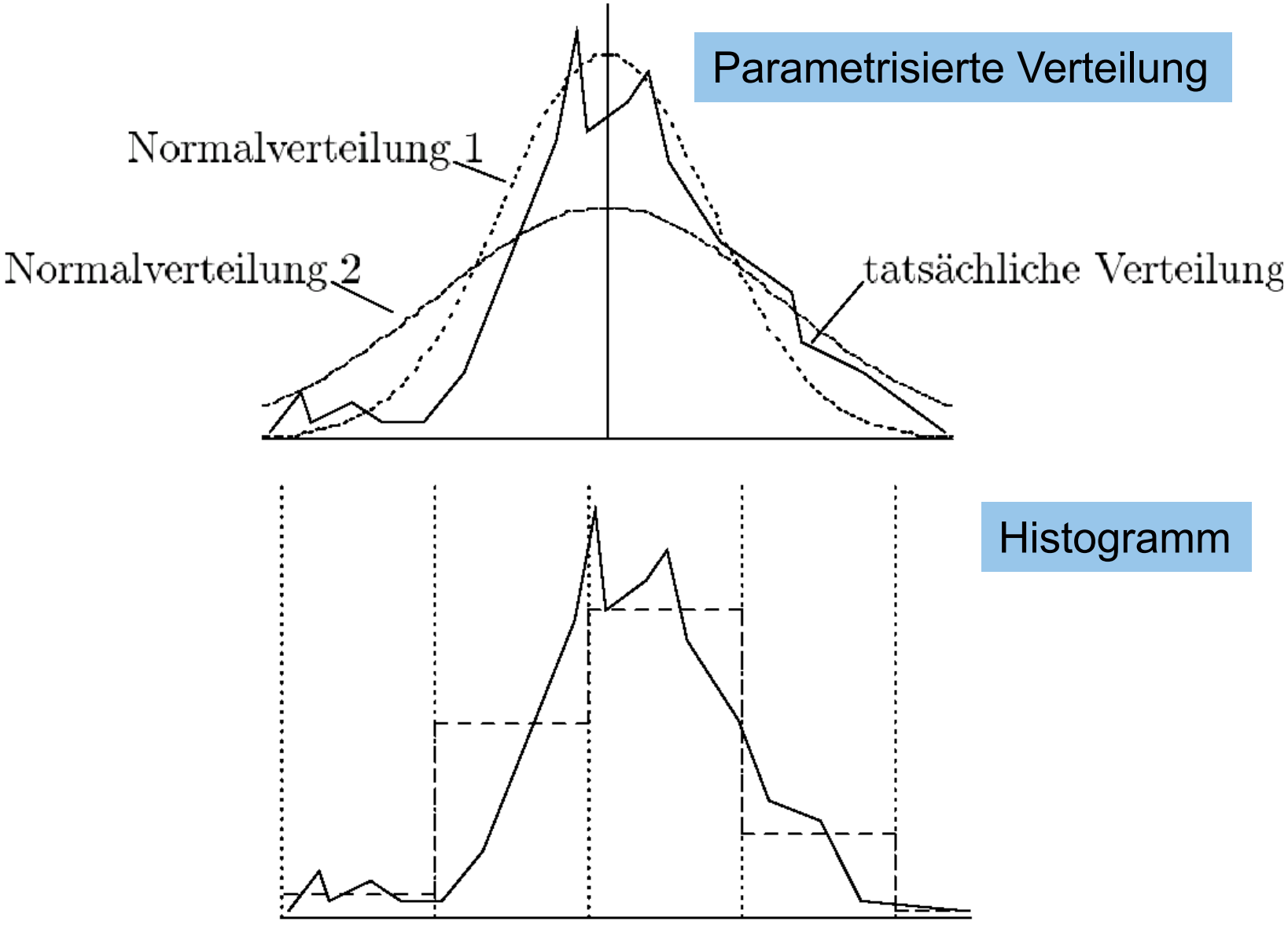
$$sel_{RS} := \frac{|R \bowtie S|}{|R \times S|} = \frac{|R \bowtie S|}{|R| \cdot |S|}$$

Abschätzung für einfache Fälle

Abschätzung der Selektivität:

- $sel_{R.A=C} = \frac{1}{|R|}$
  falls A Schlüssel von R
- $sel_{R.A=C} = \frac{1}{i}$
  falls i die Anzahl der Attributwerte von $R.A$ ist (Gleichverteilung)
- $sel_{R.A=S.B} = \frac{1}{|R|}$
  bei Equijoin von R mit S über Fremdschlüssel in S

Ansonsten z.B. Stichprobenverfahren

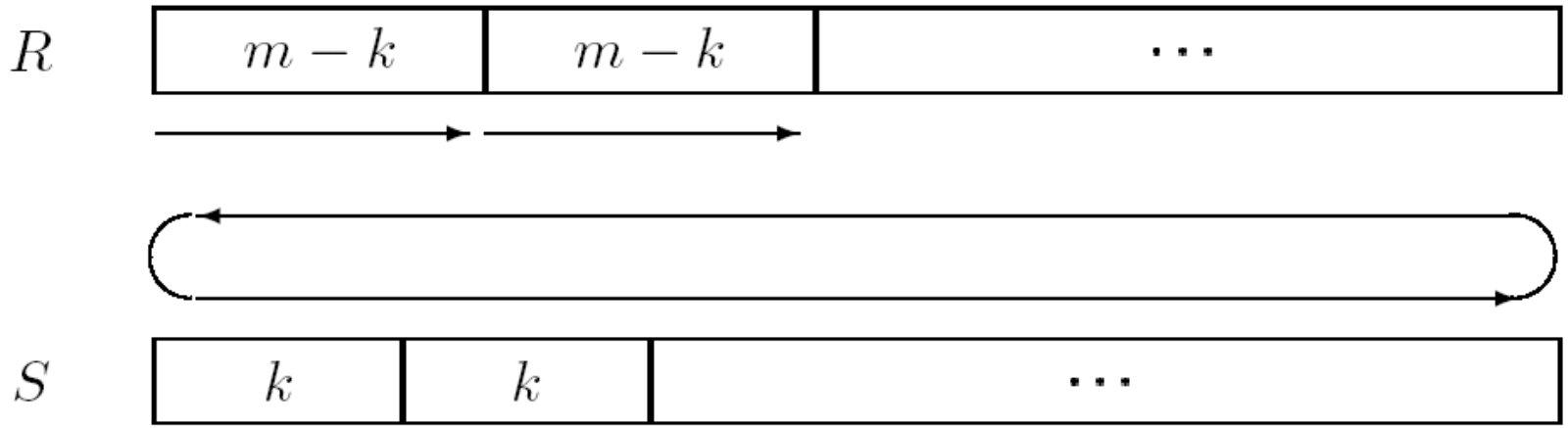

Parametrisierte Verteilung
Normalverteilung 1
Normalverteilung 2
tatsächliche Verteilung
Histogramm

Kostenabschätzungen

Selektion:

- Brute Force: Lesen aller Seiten von R
- B$^+$-Baum-Index: $t + \lceil sel_{A\theta c} \cdot b_R \rceil$
  - Absteigen der Indexstruktur
  - Lesen der qualifizierenden Tupel
- Hash-Index: für jeden die Bedingung erfüllenden Wert einen Look-up

I/O-Kosten: Block Nested Loop Join



- Durchlaufen aller Seiten von R: b_R
- Durchläufe der inneren Schleife: $\lceil b_R/(m-k) \rceil$
- Insgesamt: $b_R + k + \lceil b_R/(m-k) \rceil \cdot (b_S - k)$
- minimal, falls $k = 1$ und R die kleinere Relation

Tuning von Datenbanken

Statistiken (Histogramme, etc.) müssen explizit angelegt werden
Anderenfalls liefern die Kostenmodelle falsche Werte
In Oracle …

- analyze table Professoren compute statistics for table;
- Man kann sich auch auf approximative Statistiken verlassen
  - Anstatt compute verwendet man estimate

In DB2 …

- runstats on table …

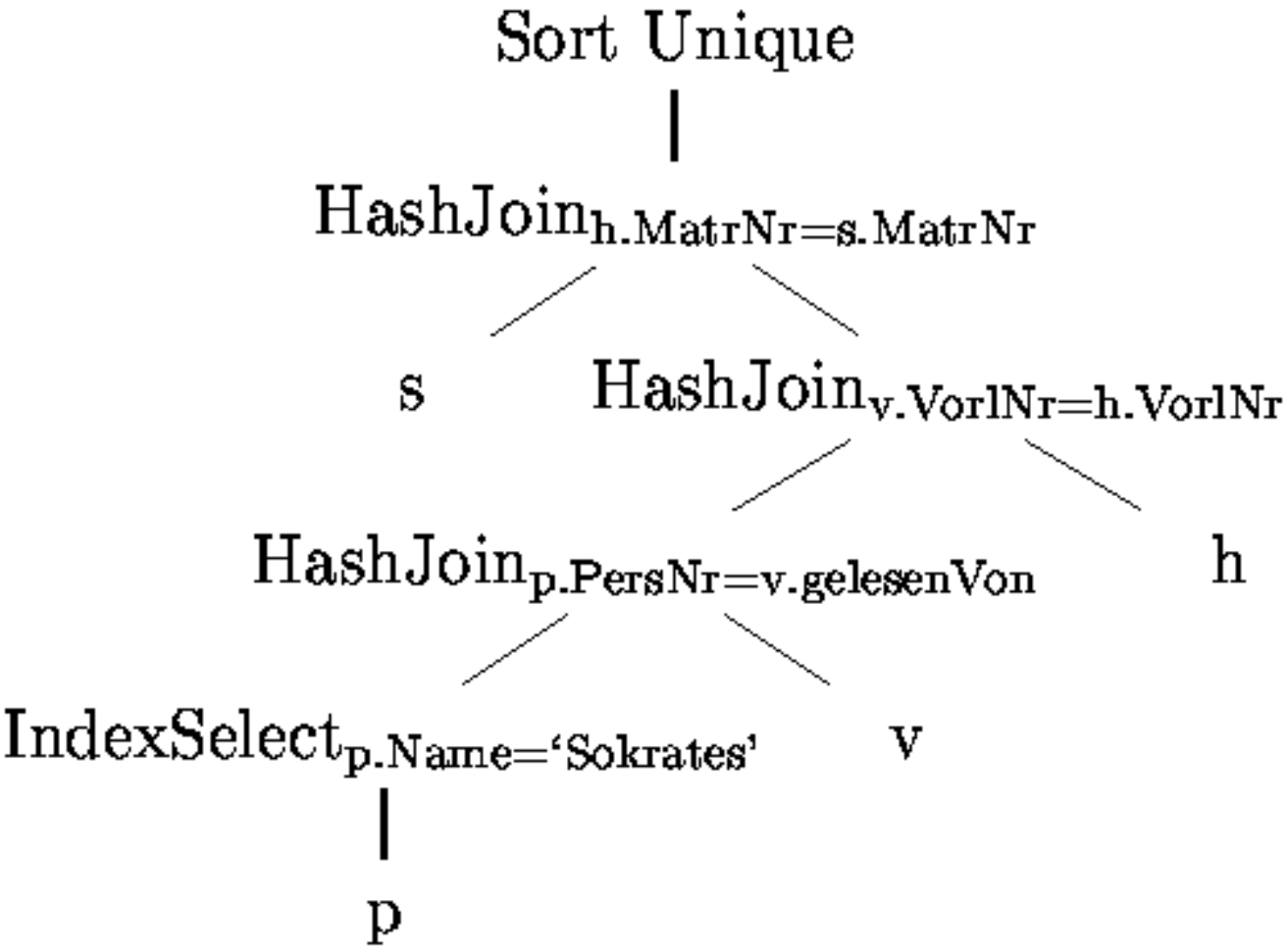
Analysieren von Leistungsengpässen

```
explain plan for
   select distinct s.Semester
   from Studenten s, hören h, Vorlesungen v, Professoren p
   where p.Name = 'Sokrates' and v.gelesenVon = p.PersNr and
         v.VorlNr = h.VorlNr and h.MatrNr = s.MatrNr;

SELECT STATEMENT   Cost = 37710
  SORT UNIQUE
    HASH JOIN
      TABLE ACCESS FULL STUDENTEN
      HASH JOIN
        HASH JOIN
          TABLE ACCESS BY ROWID PROFESSOREN
            INDEX RANGE SCAN PROFNAMEINDEX
          TABLE ACCESS FULL VORLESUNGEN
        TABLE ACCESS FULL HOEREN
```

Geschätzte Kosten von Oracle

Baumdarstellung



Algorithmen - Ansätze

Erschöpfende Suche
- Dynamische Programmierung (System R)
- A* Suche

Heuristiken (Planbewertung nötig)
- Minimum Selectivity, Intermediate Result,...
- KBZ-Algorithmus, AB-Algorithmus

Randomisierte Algorithmen
- Iterative Improvement
- Simulated Annealing

Problemgröße

Suchraum (Planstruktur)

1. # Bushy-Pläne mit *n* Tabellen [Ganguly et al. 1992]:

$$\frac{(2(n\text{-}1))!}{(n\text{-}1)!}$$

n	e^n	(2(*n*-1))!/(*n*-1)!
2	7	2
5	146	1680
10	22026	$1{,}76*10^{10}$
20	$4{,}85 * 10^9$	$4{,}3*10^{27}$

2. Plankosten unterscheiden sich um Größenordnungen
3. Optimierungsproblem ist *NP*-hart [Ibaraki 1984]

Dynamische Programmierung

Identifikation von 3 Phasen

1. Access Root - Phase: Aufzählen der Zugriffspläne

2. Join Root - Phase: Aufzählen der Join-Kombinationen

3. Finish Root - Phase: sort, group-by, etc.

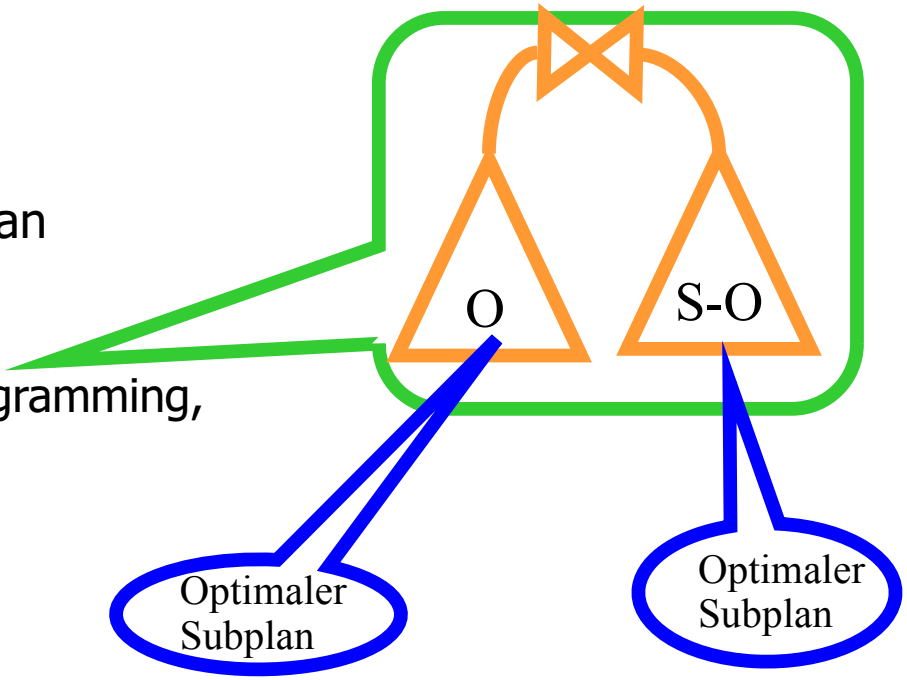
Optimierung durch Dynamische Programmierung

Standardverfahren in heutigen relationalen Datenbanksystemen

Voraussetzung ist ein Kostenmodell als Zielfunktion
- I/O-Kosten
- CPU-Kosten

DP basiert auf dem Optimalitätskriterium von Bellman
Literatur zu DP:

- D. Kossmann und K. Stocker: Iterative Dynamic Programming, TODS, 2000 to appear (online)



DP - Beispiel

1. Phase: Zugriffspläne ermitteln

Index	Pläne
{ABC}	
{BC}	
{AC}	
{AB}	
{C}	
{B}	
{A}	

DP - Beispiel

1. Phase: Zugriffspläne ermitteln

Index	Pläne
{ABC}	
{BC}	
{AC}	
{AB}	
{C}	scan(C)
{B}	scan(B), iscan(B)
{A}	scan(A)

DP - Beispiel

2. Phase: Join-Pläne ermitteln (2-fach,...,n-fach)

Index	Pläne	**Pruning**
{ABC}		
{BC}	...	
{AC}	s(A) ⋈ s(C), s(C) ⋈ s(A)	
{AB}	s(A) ⋈ s(B), s(A) ⋈ is(B), is(B) ⋈ s(A),...	
{C}	scan(C)	
{B}	scan(B), iscan(B)	
{A}	scan(A)	

DP - Beispiel

3. Phase: Finalisierung

Index	Pläne
{ABC}	(is(B) ⋈ s(A)) ⋈ s(C)
{BC}	...
{AC}	s(A) ⋈ s(C)
{AB}	s(A) ⋈ is(B), is(B) ⋈ s(A)
{C}	scan(C)
{B}	scan(B), iscan(B)
{A}	scan(A)

Algorithmus DynProg

Function DynProg
input A query q over relations $R_1, \ldots, R_n$ *//Set of relations to be joined*
output A query execution plan fpr q *//Processing Tree*

```
1:    for i = 1 to n do {
2:        BestPlansTable({R_i}) = accessPlans(R_i)
3:        prunePlans(BestPlansTable({R_i}))
4:    }
5:    for i = 2 to n do {
6:        for all S ⊆ {R_1, ..., R_n} such that |S| == i do {
7:            BestPlansTable(S) = ∅
8:            for all O ⊂ S do {
9:                BestPlansTable(S) = BestPlansTable(S) ∪
                          joinPlans(BestPlansTable(O), BestPlansTable(S − O))
10:               prunePlans(BestPlansTable(S))
11:           }
12:       }
13:   }
14:   prunePlans(BestPlansTable({R_1, ..., R_n})
15:   return BestPlansTable({R_1, ..., R_n})
```

BestPlansTable	
Index (S)	Alternative Plans
n, c, m	$(\text{iseek}_{\text{City}='\text{Munich}'}(\text{m}) \bowtie \text{iseek}_{\text{From}}(\text{c})) \bowtie \text{iseek}_{\text{City}='\text{NY}'}(\text{n})$
c, m	$\text{iseek}_{\text{City}='\text{Munich}'}(\text{m}) \bowtie \text{iseek}_{\text{From}}(\text{c})$
n, m	$\text{iseek}_{\text{City}='\text{NY}'}(\text{n}) \times \text{iseek}_{\text{City}='\text{Munich}'}(\text{m})$
n, c	$\text{iseek}_{\text{City}='\text{NY}'}(\text{n}) \bowtie \text{iseek}_{\text{To}}(\text{c})$
Airport m	$\text{scan}(\text{m})$, $\text{iseek}_{\text{City}='\text{Munich}'}(\text{m})$, $\text{iscan}_{\text{Code}}(\text{m})$
Connection c	$\text{scan}(\text{c})$, $\text{iseek}_{\text{From}}(\text{c})$, $\text{iseek}_{\text{To}}(\text{c})$
Airport n	$\text{scan}(\text{n})$, $\text{iseek}_{\text{City}='\text{NY}'}(\text{n})$, $\text{iscan}_{\text{Code}}(\text{n})$