

Kapitel 8

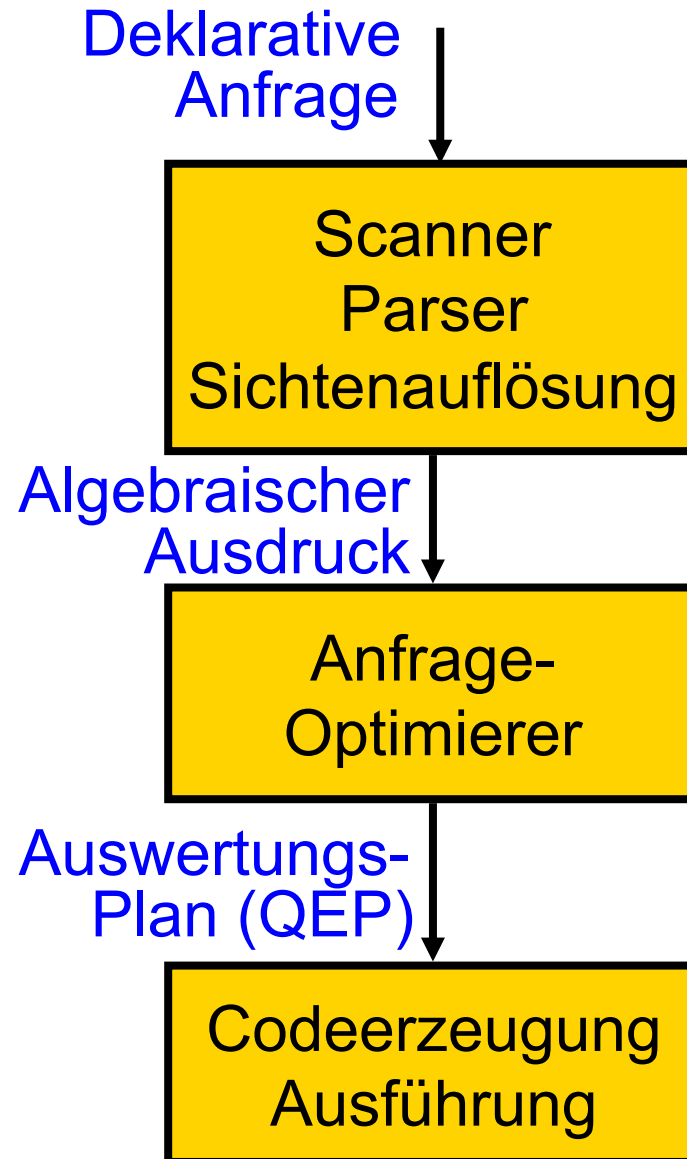
Anfragebearbeitung



Das Bild kann nicht angezeigt werden. Dieser Computer verfügt möglicherweise über zu wenig Arbeitsspeicher, um das Bild zu öffnen, oder das Bild ist beschädigt. Starten Sie den Computer neu, und öffnen Sie dann erneut die Datei. Wenn weiterhin das rote x angezeigt wird, müssen Sie das Bild möglicherweise löschen und dann erneut einfügen.

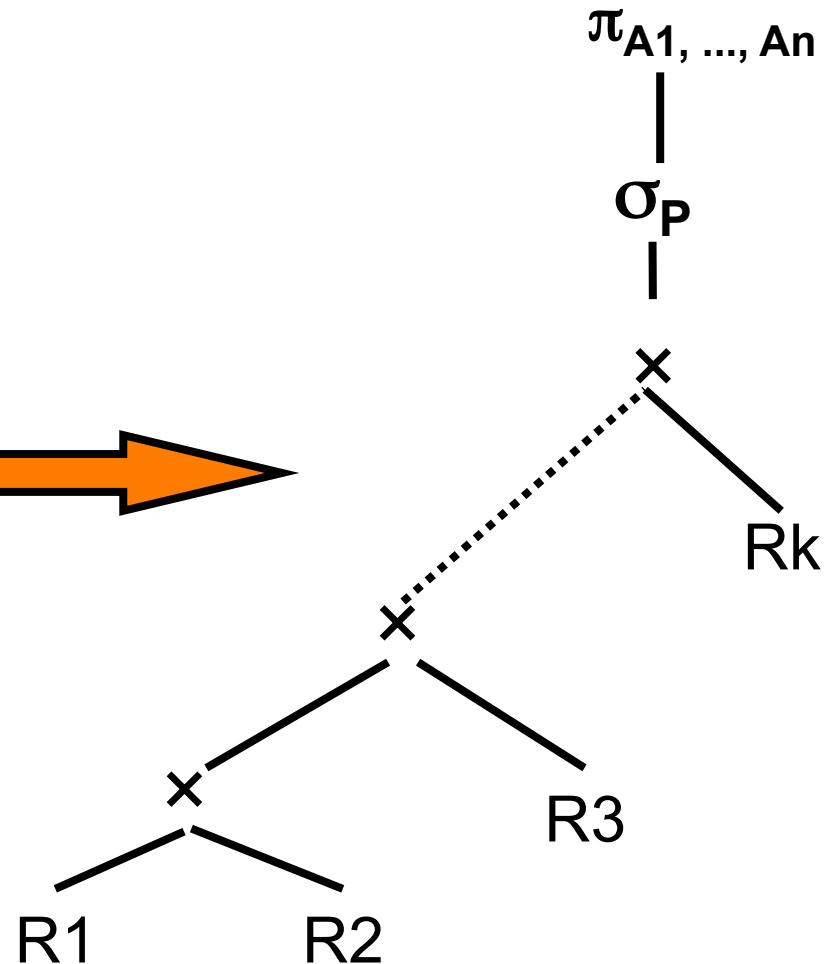
- **Logische Optimierung**
- **Physische Optimierung**
- **Kostenmodelle**
- **„Tuning“**

Ablauf der Anfrageoptimierung



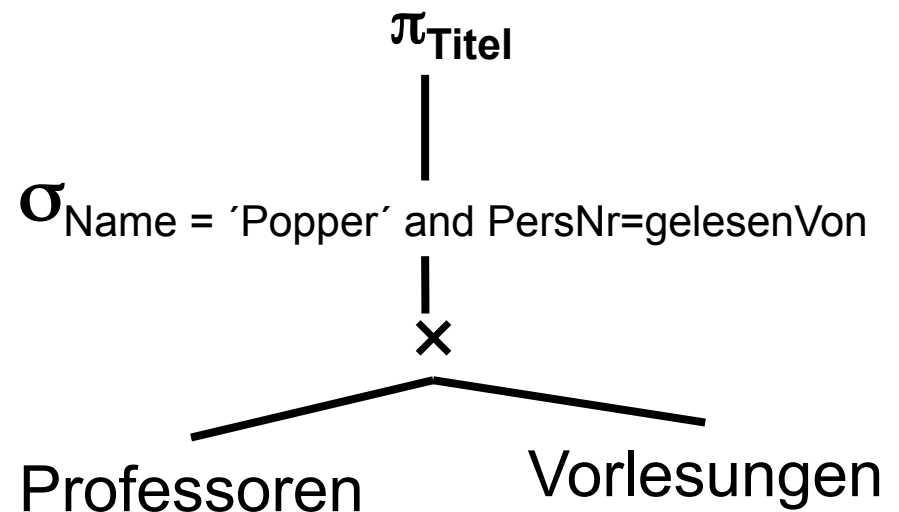
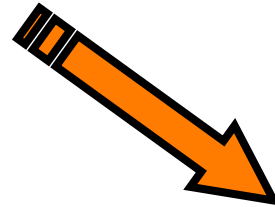
Kanonische Übersetzung

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



Kanonische Übersetzung

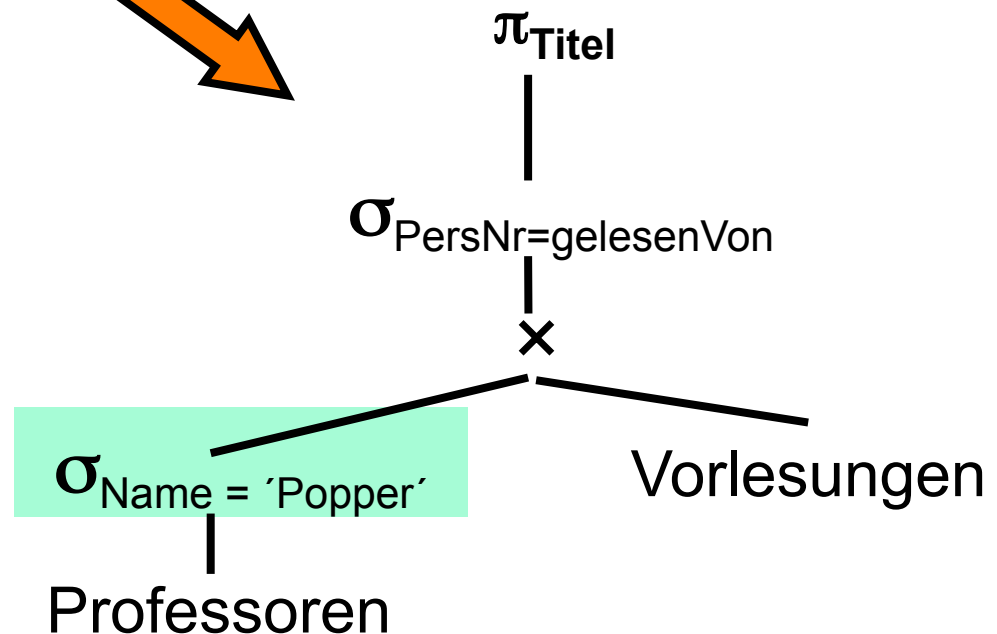
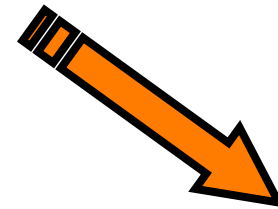
select Titel
from Professoren, Vorlesungen
where Name = 'Popper' and
PersNr = gelesenVon



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

Erste Optimierungsidee

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

$$\sigma_{c_1 \wedge c_2 \wedge \dots \wedge c_n}(R) \equiv \sigma_{c_1}(\sigma_{c_2}(\dots(\sigma_{c_n}(R))\dots))$$

2. σ ist kommutativ

$$\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$$

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

$$\pi_{L_1}(\pi_{L_2}(\dots(\pi_{L_n}(R))\dots)) \equiv \pi_{L_1}(R)$$

4. Vertauschen von σ und π

Falls die Selektion sich nur auf die Attribute A_1, \dots, A_n der Projektionsliste bezieht, können die beiden Operationen vertauscht werden

$$\pi_{A_1, \dots, A_n}(\sigma_c(R)) \equiv \sigma_c(\pi_{A_1, \dots, A_n}(R))$$

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:

$$\sigma_c(R \bowtie_j S) \equiv \sigma_c(R) \bowtie_j (\sigma_{c_2}(S))$$

Äquivalenzerhaltende Transformationsregeln

7. Vertauschung von π mit \bowtie

Die Projektionsliste L sei: $L = \{A_1, \dots, A_n, B_1, \dots, B_m\}$, wobei A_i Attribute aus R und B_j Attribute aus S seien. Falls sich das Joinprädikat c nur auf Attribute aus L bezieht, gilt folgende Umformung:

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

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

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

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

Äquivalenzerhaltende Transformationsregeln

8. Die Operationen \bowtie , X , \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

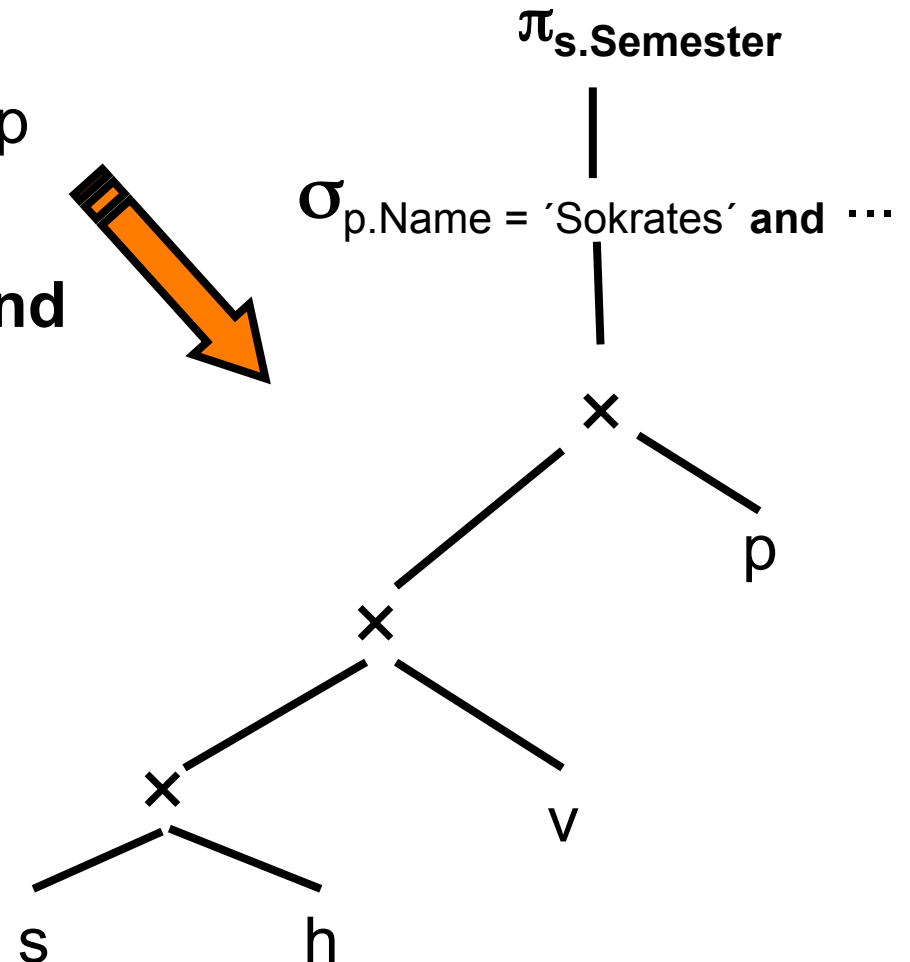
Vorlesungen v, Professoren p

where p.Name = 'Sokrates' **and**

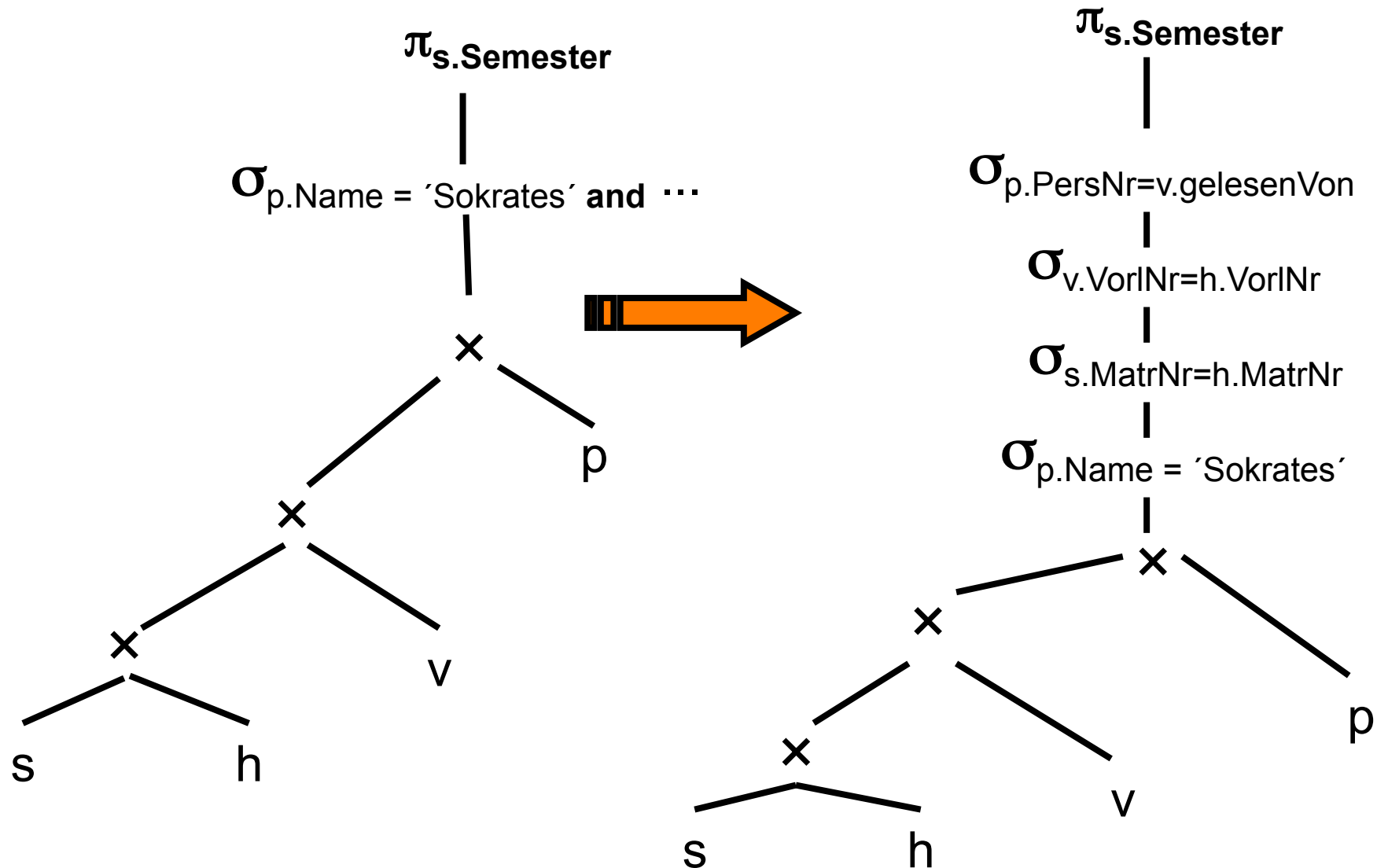
v.gelesenVon = p.PersNr **and**

v.VorlNr = h.VorlNr **and**

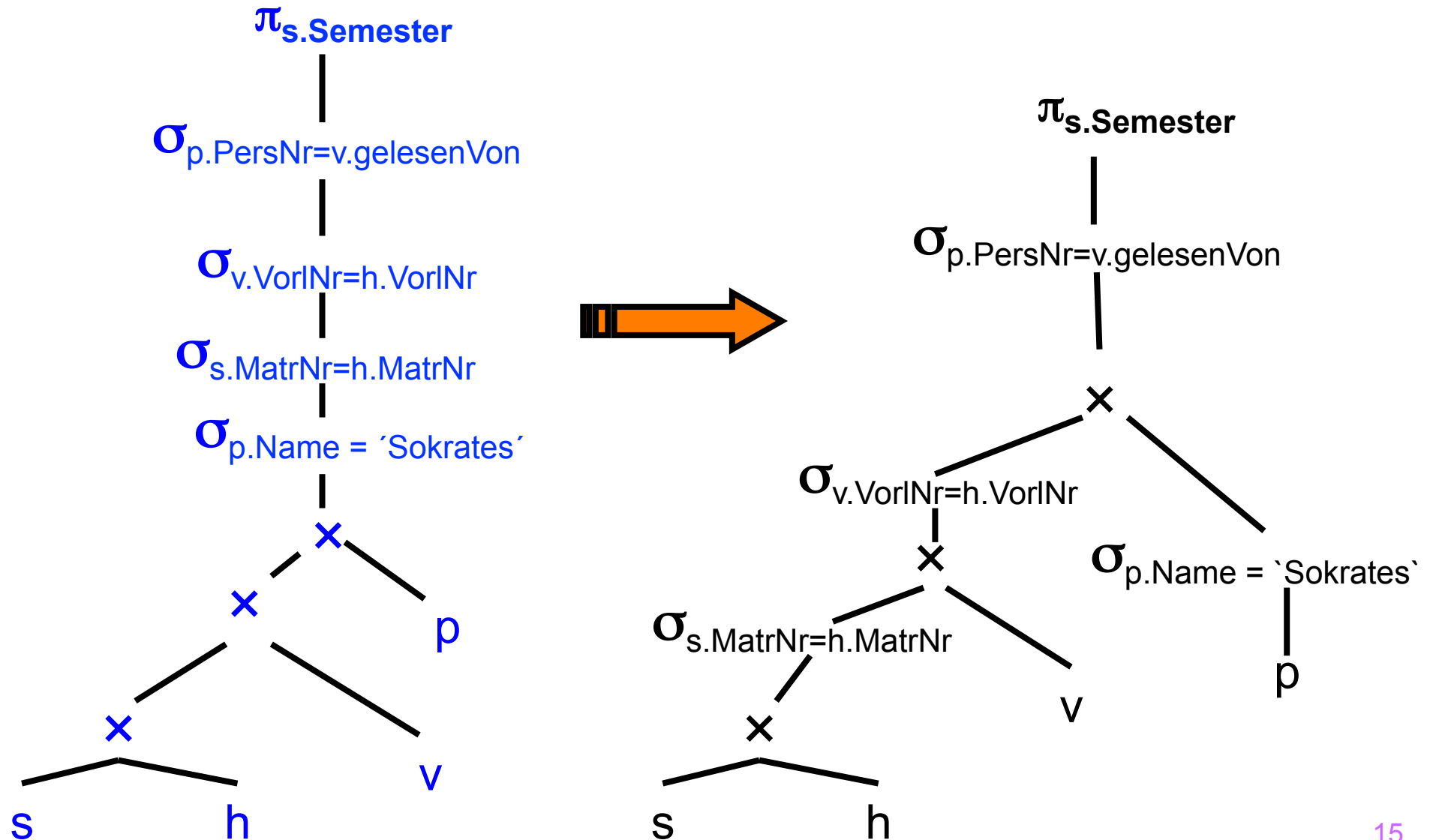
h.MatrNr = s.MatrNr



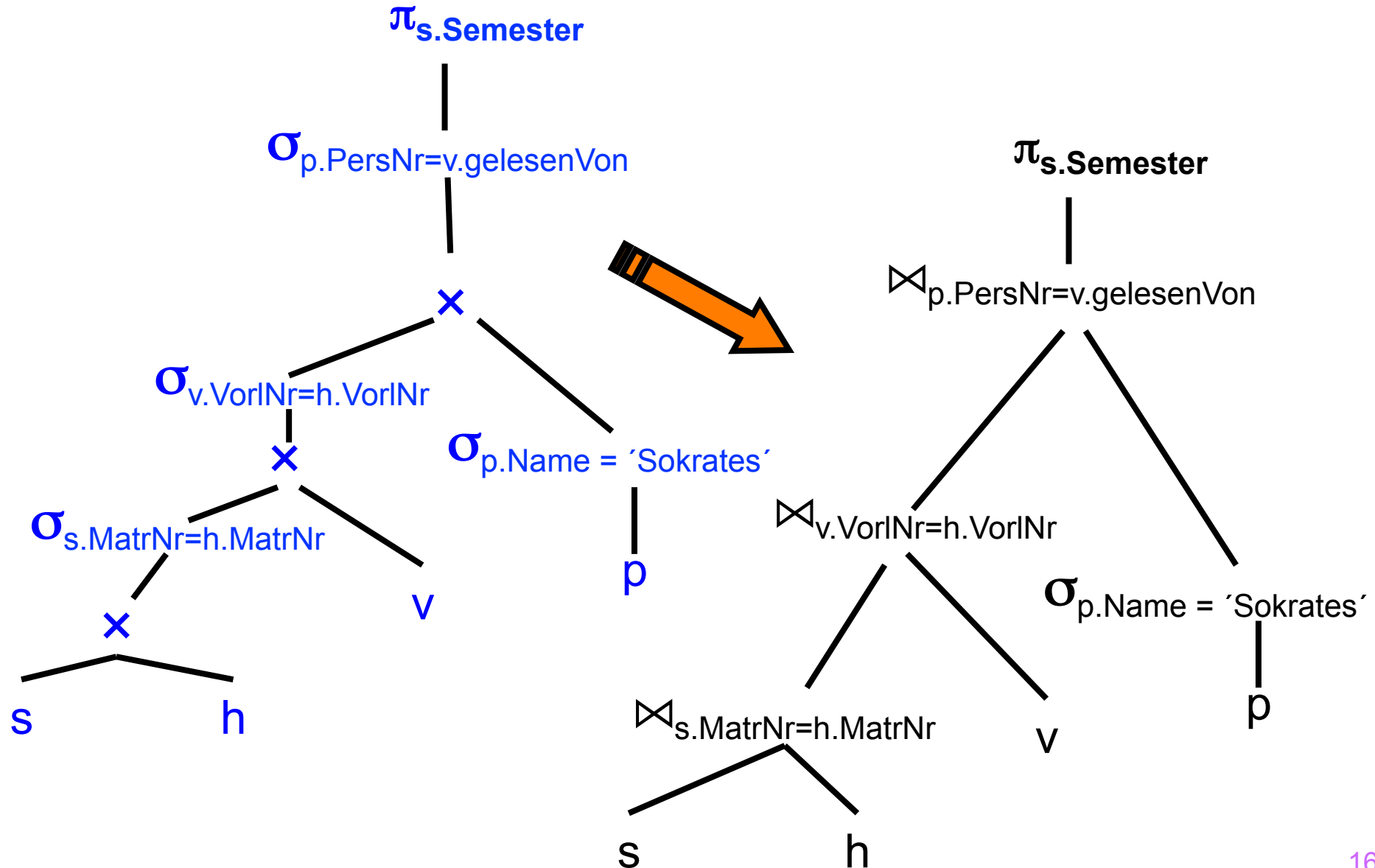
Aufspalten der Selektionsprädikate



Verschieben der Selektionsprädikate „Pushing Selections“

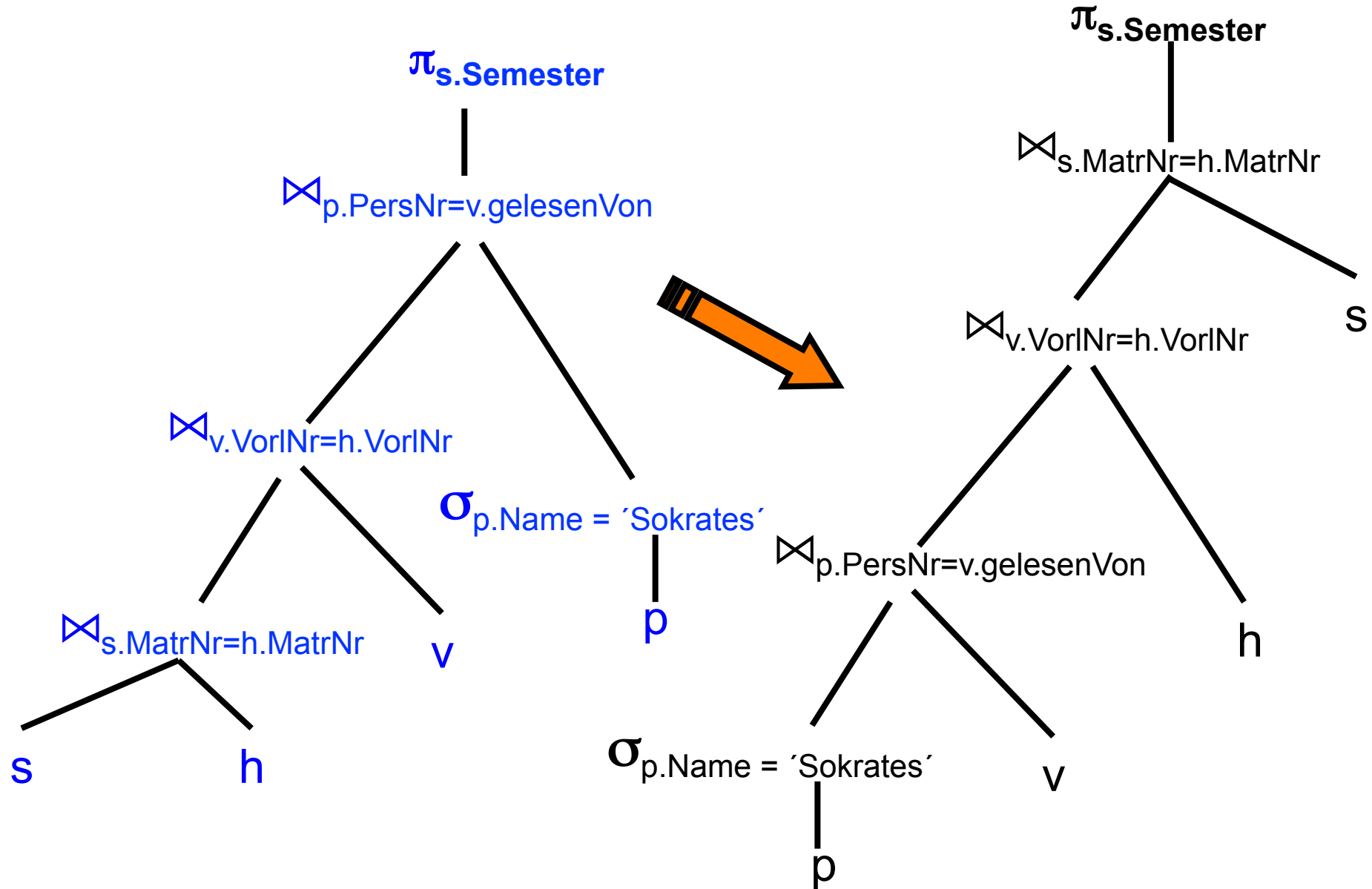


Zusammenfassung von Selektionen und Kreuzprodukten zu Joins

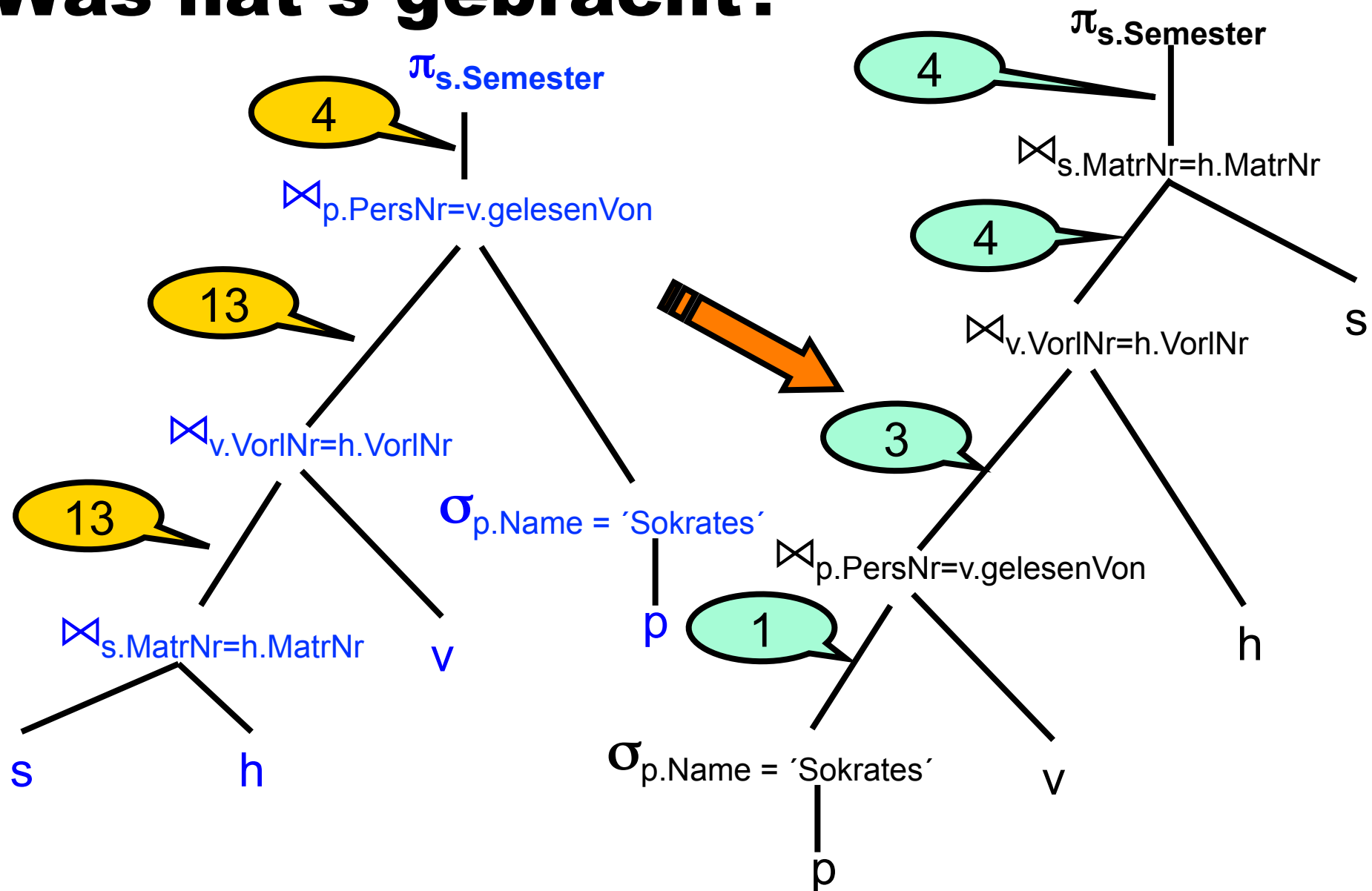


Optimierung der Joinreihenfolge

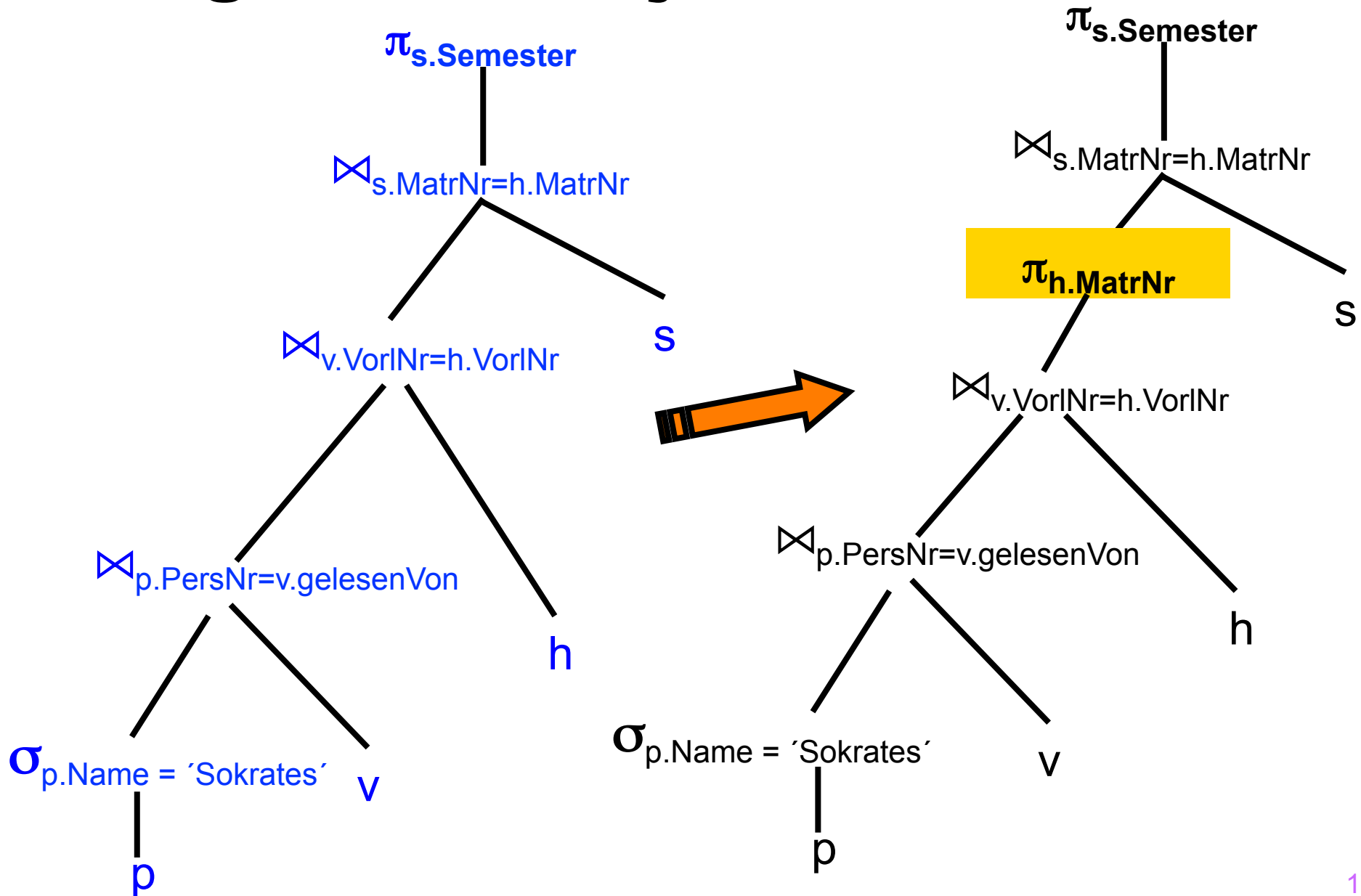
Kommutativität und Assoziativität ausnutzen



Was hat's gebracht?



Einfügen von Projektionen



Organisatorisches

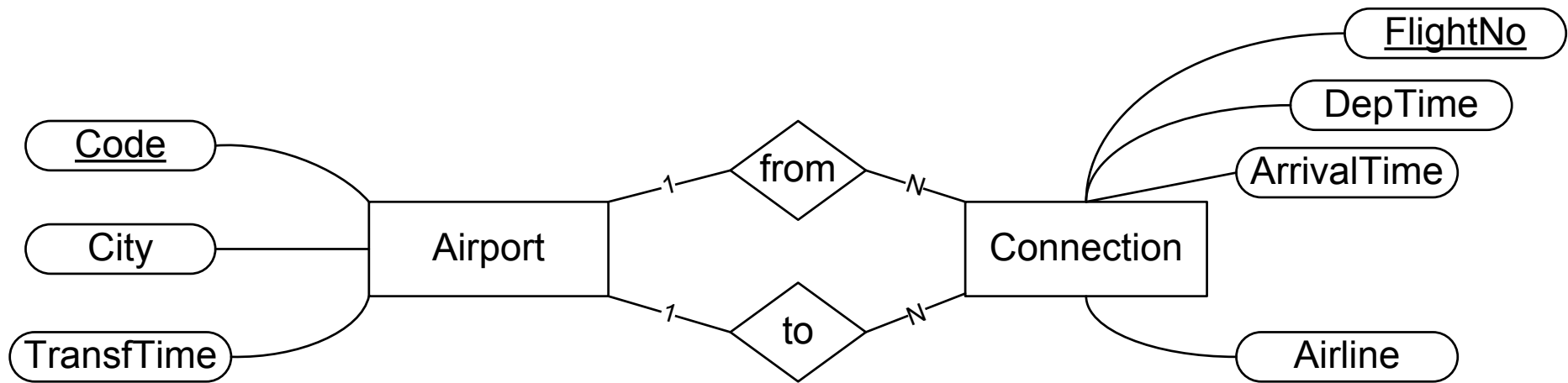
- DB2 Zertifizierungskurs
 - IBM
 - Fast kostenlos
 - Für gute Studierende
- DB Praktikum
 - Vorbesprechung 27. Januar 2009, 14.00 Uhr in unserem Ted Codd Seminarraum MI 02.09.014
 - unter <http://dbpra.in.tum.de/dbpra-aktuell.shtml>
- Vorlesung am kommenden Freitag 23.1.09 (**neuer Stoff**)
 - In Hörsaal 1 (hier!)
- Vorlesung am 30.1.09 ...
 - Findet als Zentralübung statt (**kein neuer Stoff**)

Eine weitere Beispieloptimierung

SQL-Anfrage: Von München direkt nach NY?

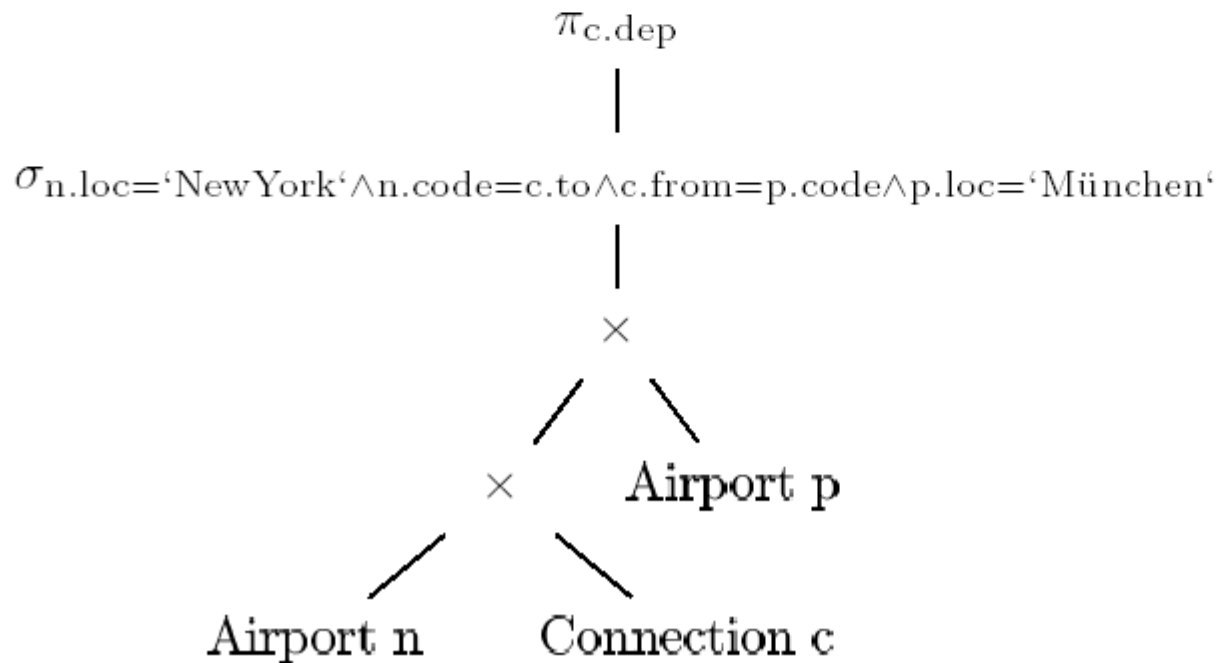
```
select c.dep
from Airport n, Connection c, Airport p
where n.loc = 'New York' and
      n.code = c.to and
      c.from = p.code and
      p.loc = 'München'
```

Airport p			Connection c				Airport n		
p.code	p.loc	c.from	c.to	...	n.code	n.loc	...
...
MUC	München	...					JFK	New York	...
...					LGA	New York	...
						



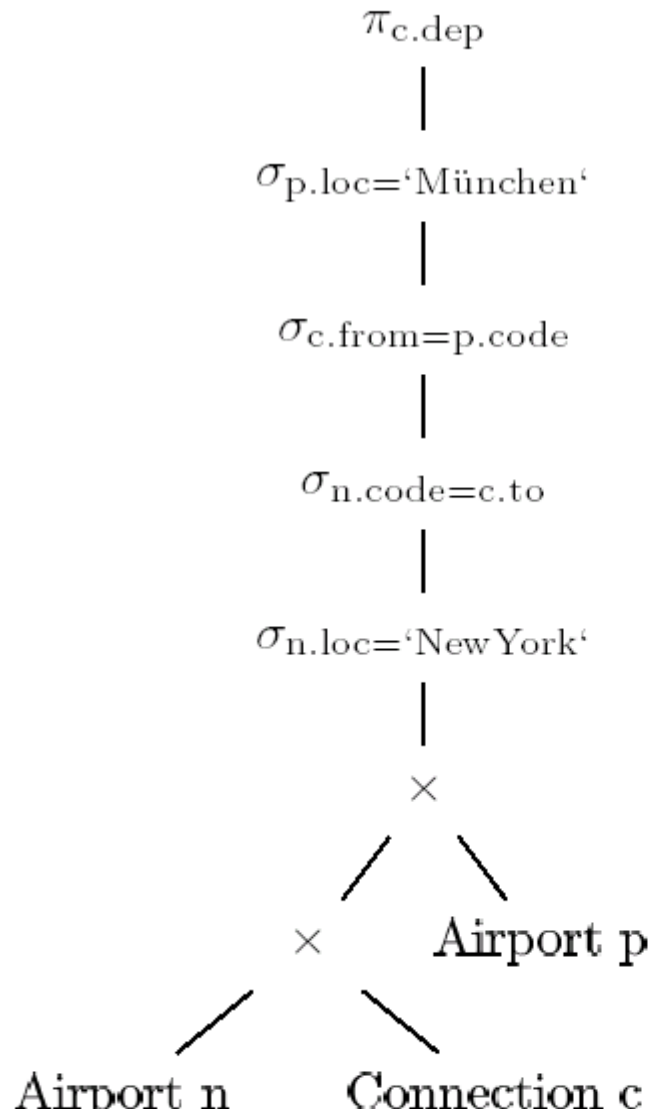
Kanonische Übersetzung

```
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'
```

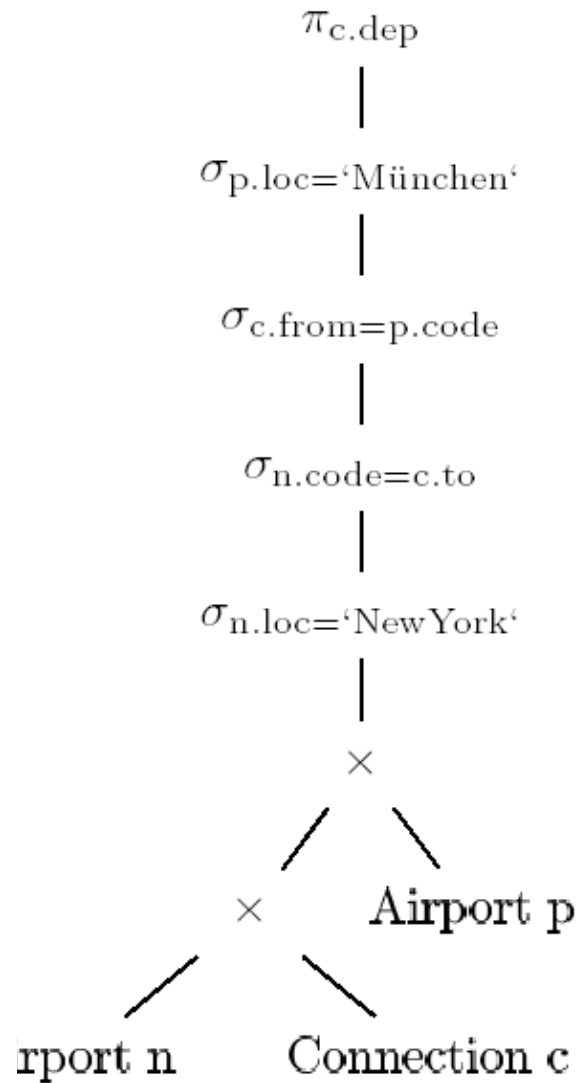


Selektionsprädikate „aufbrechen“

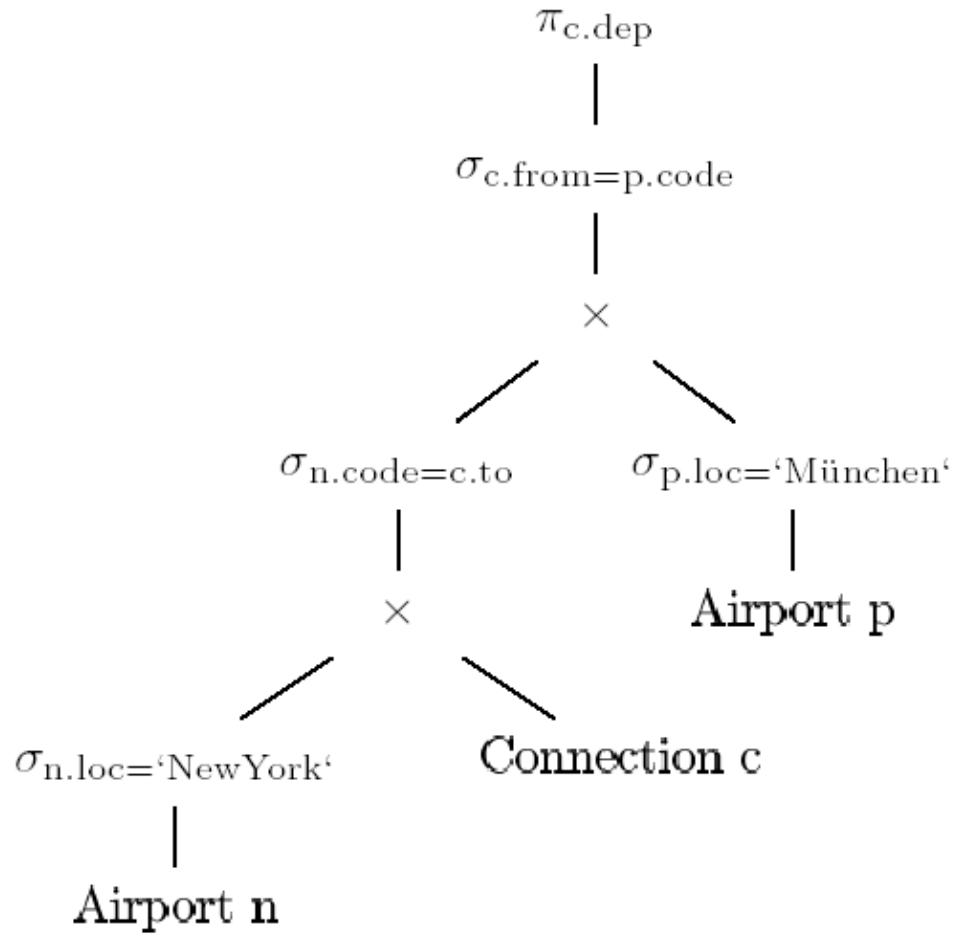
$$\sigma_{p_1 \wedge p_2 \wedge \dots \wedge p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\dots(\sigma_{p_n}(R))\dots))$$



„Pushing Selections“

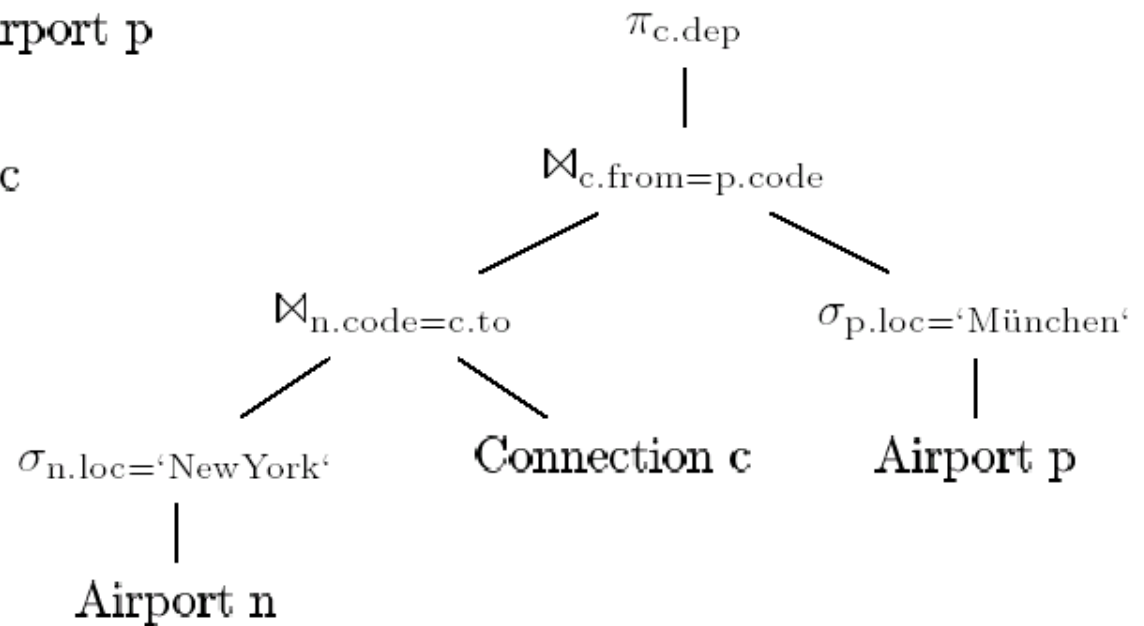
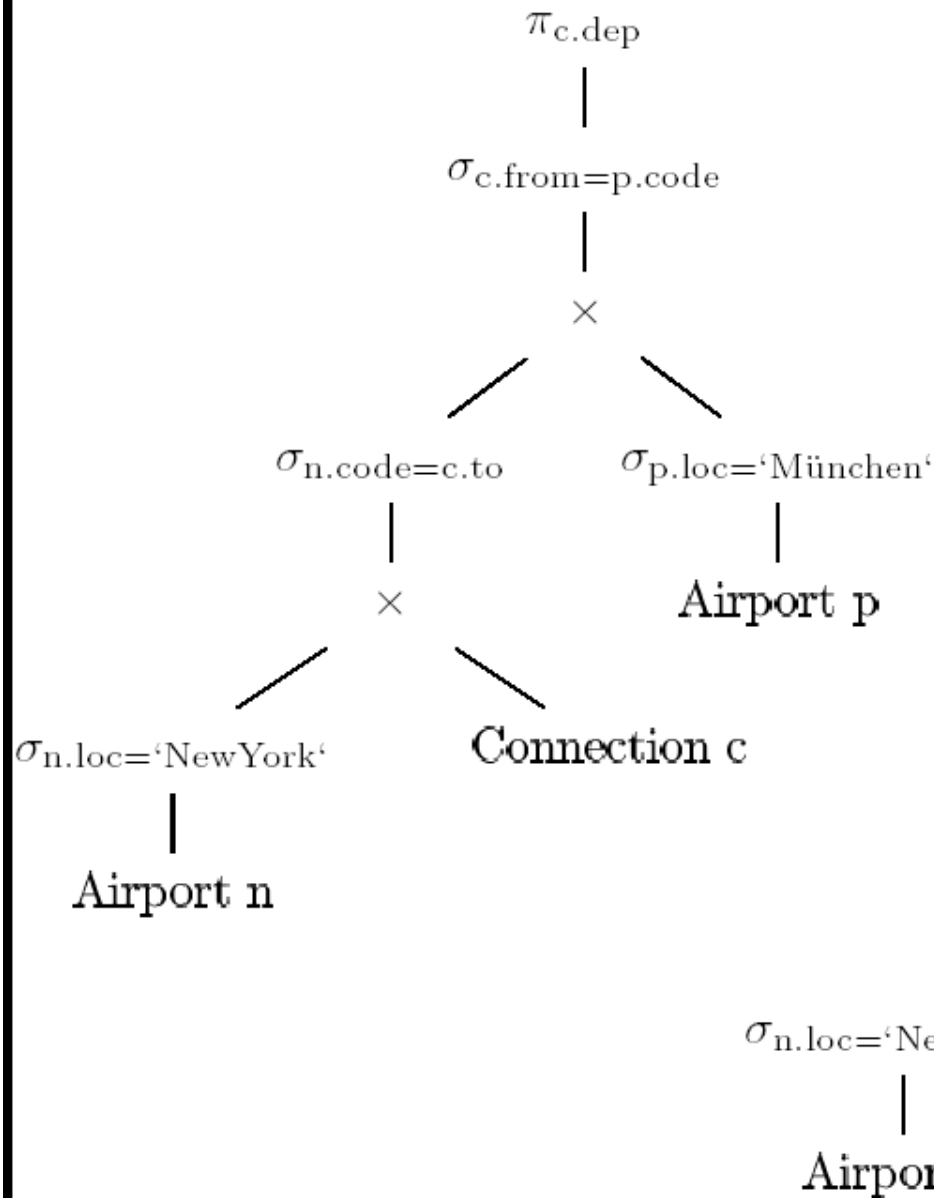


$\sigma_p(\sigma_q(R))$	$=$	$\sigma_q(\sigma_p(R))$
$\sigma_p(R_1 \bowtie R_2)$	$=$	$\sigma_p(R_1) \bowtie R_2$
$\sigma_p(R_1 \times R_2)$	$=$	$\sigma_p(R_1) \times R_2$

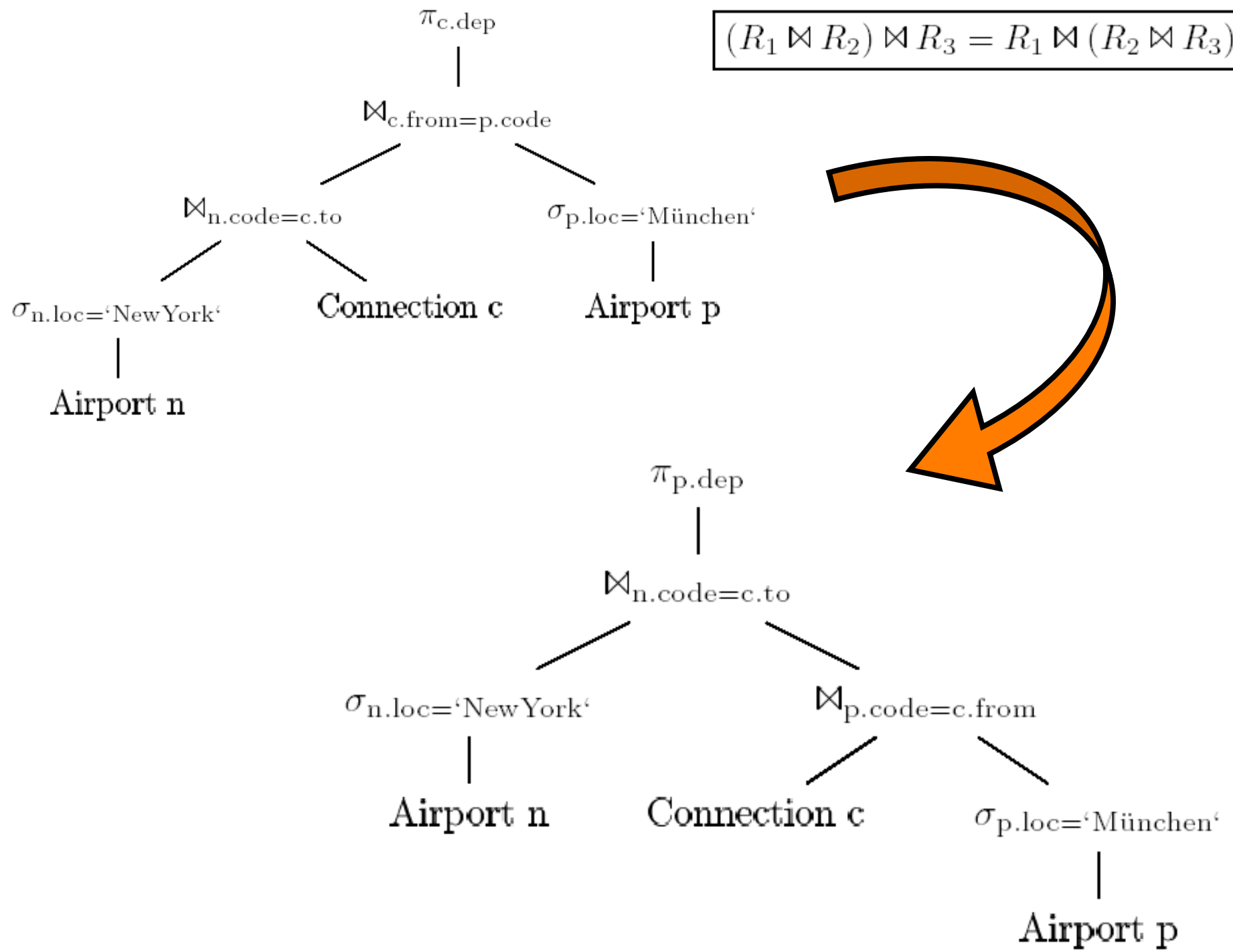


Zusammenfassung von $\sigma \times$ zu \bowtie

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



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



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**

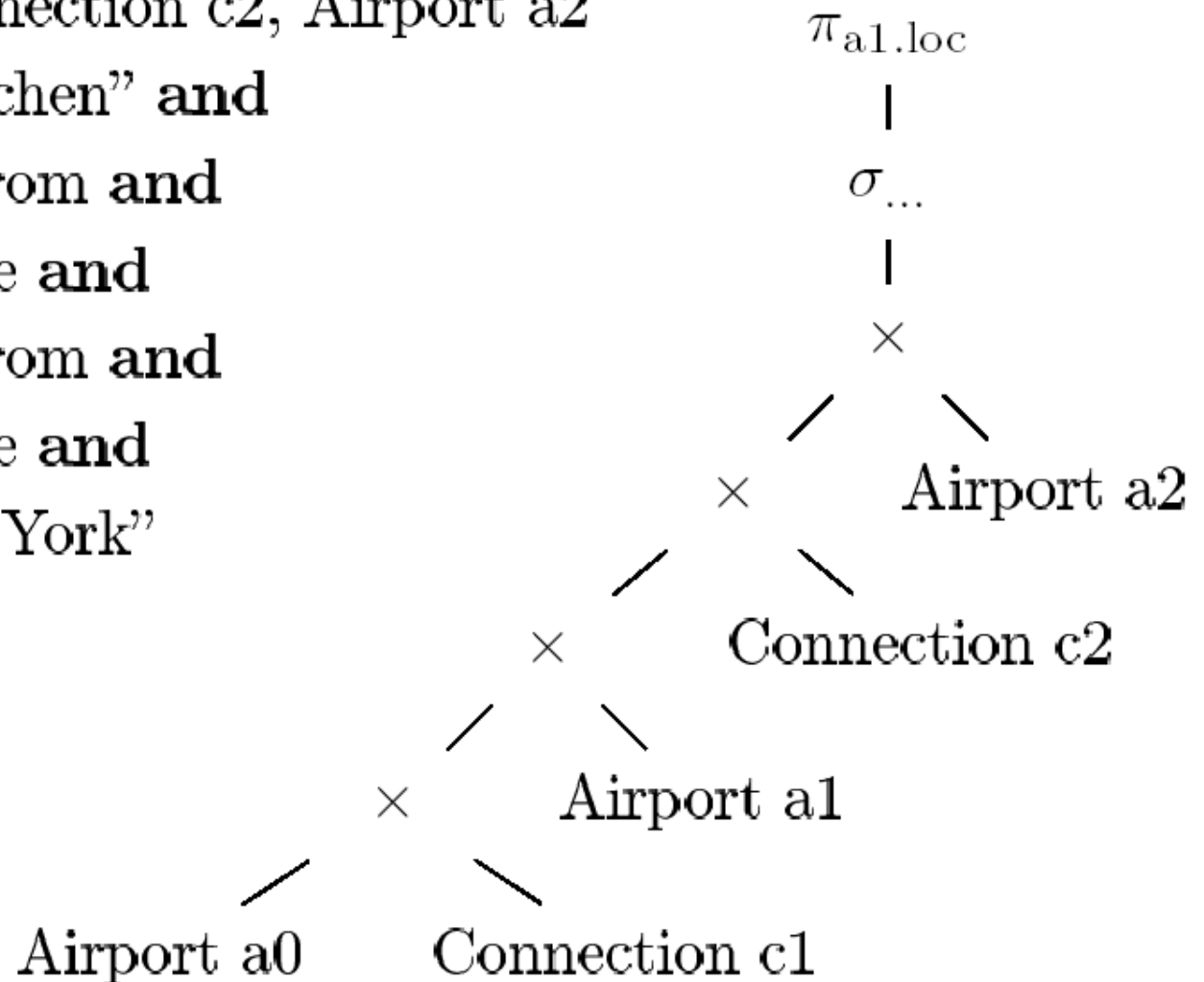
a0.code = c1.from **and**

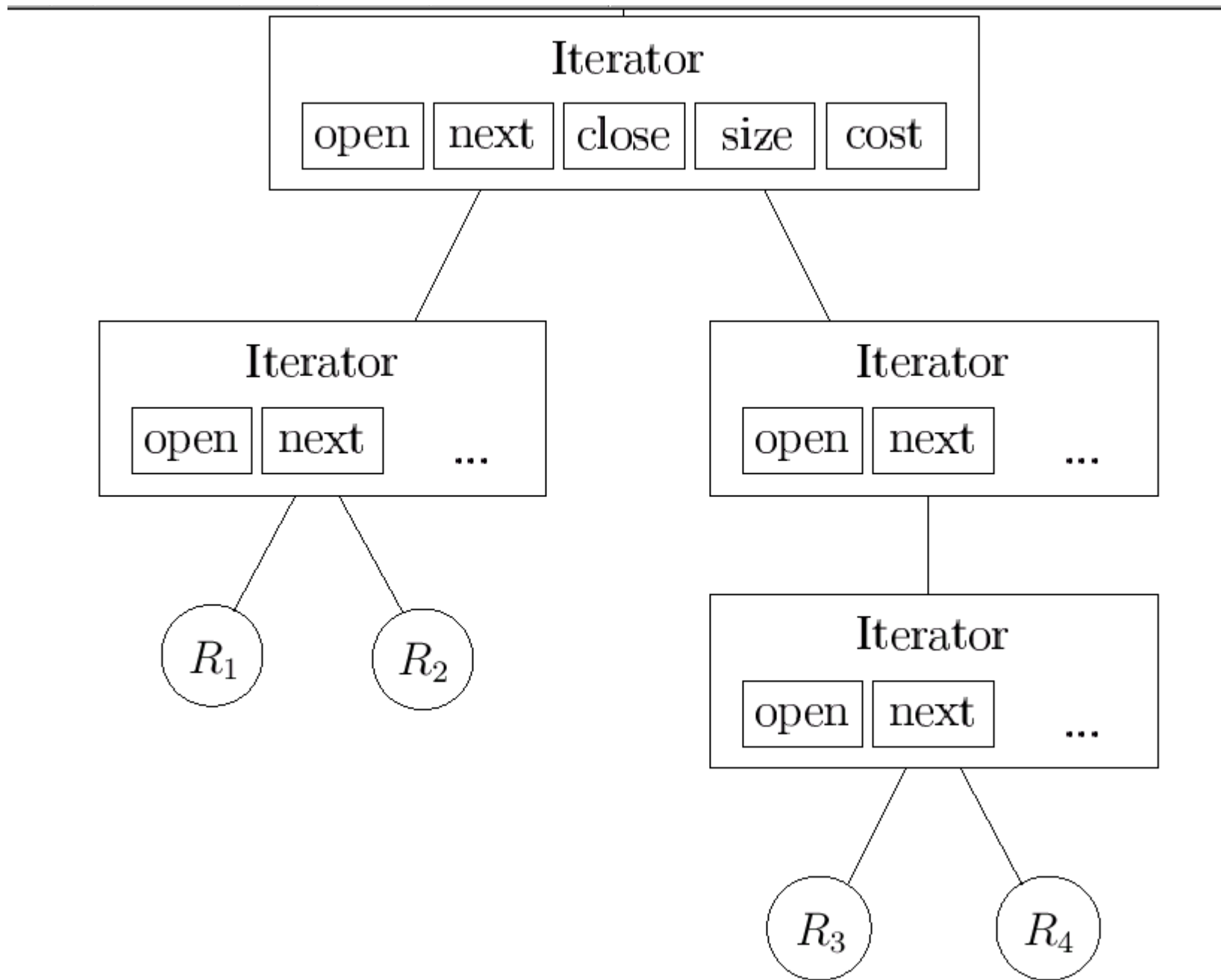
c1.to = a1.code **and**

a1.code = c2.from **and**

c2.to = a2.code **and**

a2.loc = "New York"





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 )
```



automatisch

```
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)

$$T_1 \bowtie_p T_2 := \{t_1 \circ t_2 \mid t_1 \in T_1 \wedge t_2 \in T_2(t_1) \wedge p(t_1 \circ t_2)\}.$$

Einfache Entschachtelung

Q2:

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

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

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

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

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

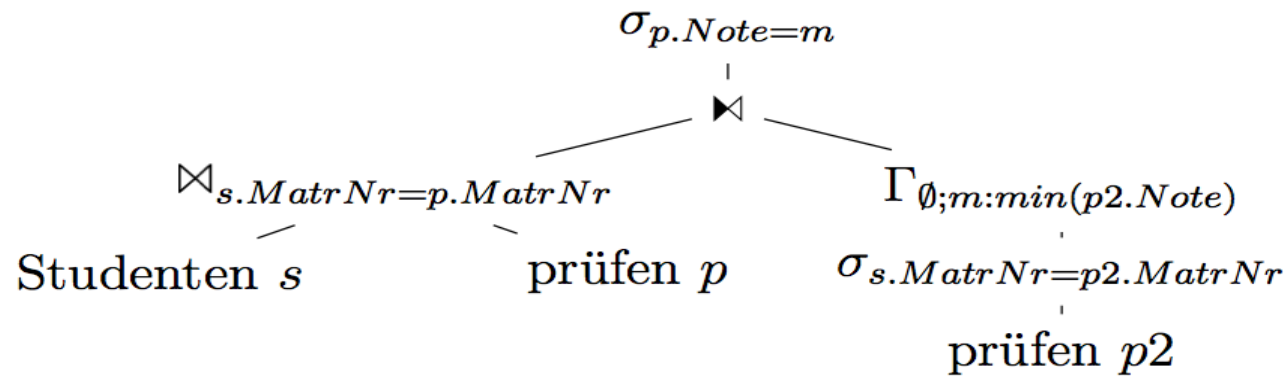
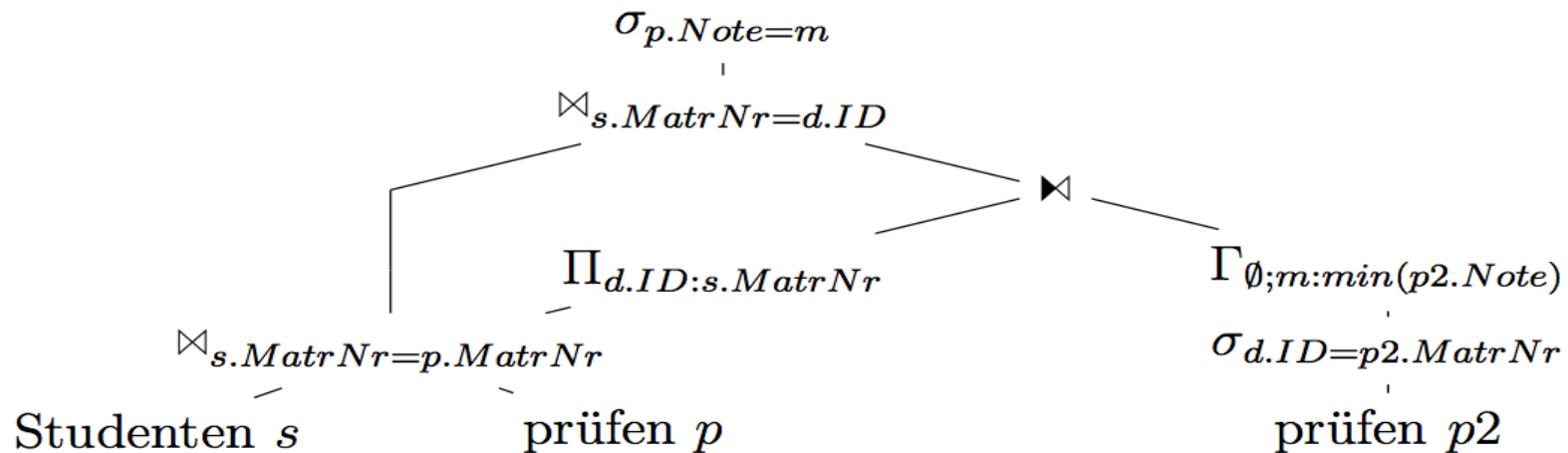


Fig 8.9: Original-Anfrage Q1



Weitere Transformationsregeln

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

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

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

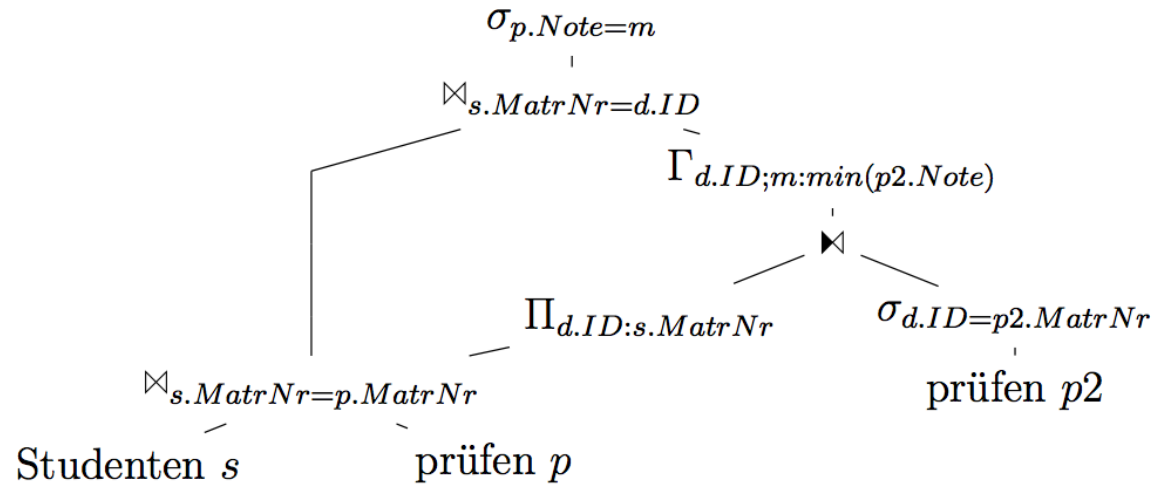
$$D \bowtie (\Pi_A(T)) \equiv \Pi_{A \cup \mathcal{A}(D)}(D \bowtie T)$$

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

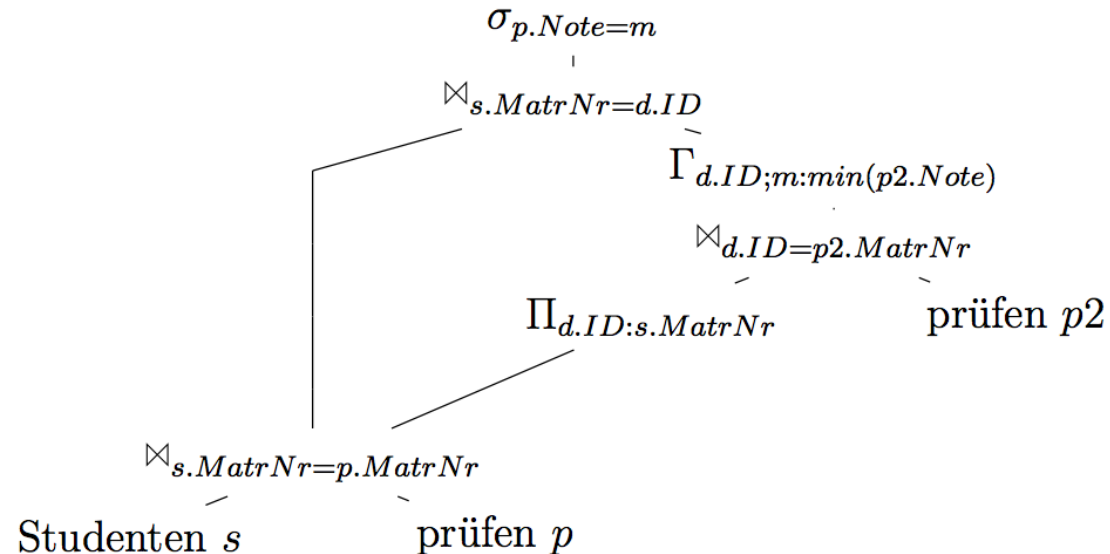
$$D \bowtie (T_1 \cap T_2) \equiv (D \bowtie T_1) \cap (D \bowtie T_2)$$

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

Beispiel

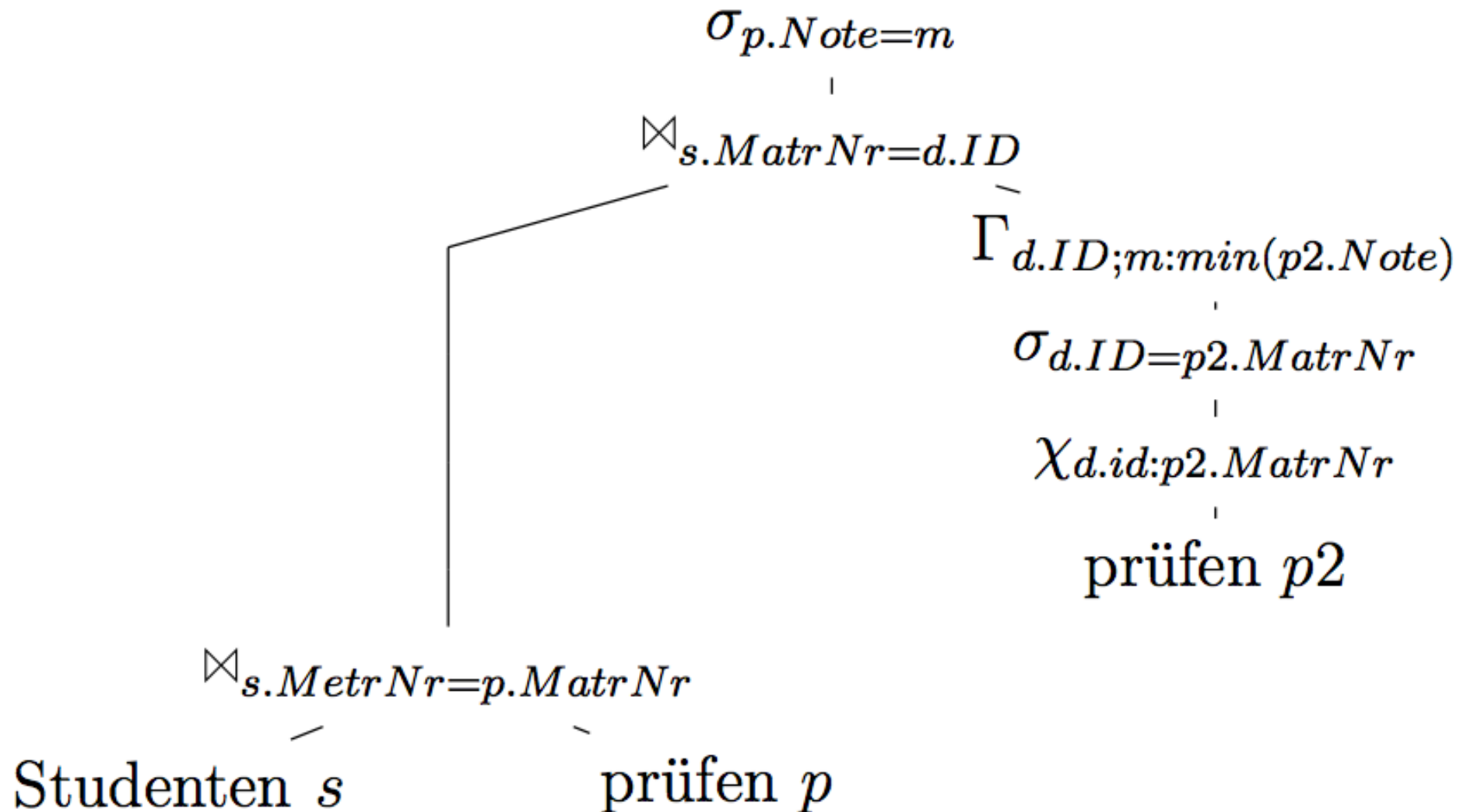


8.11: Vertauschen von Gruppierung/Aggregation mit dem a

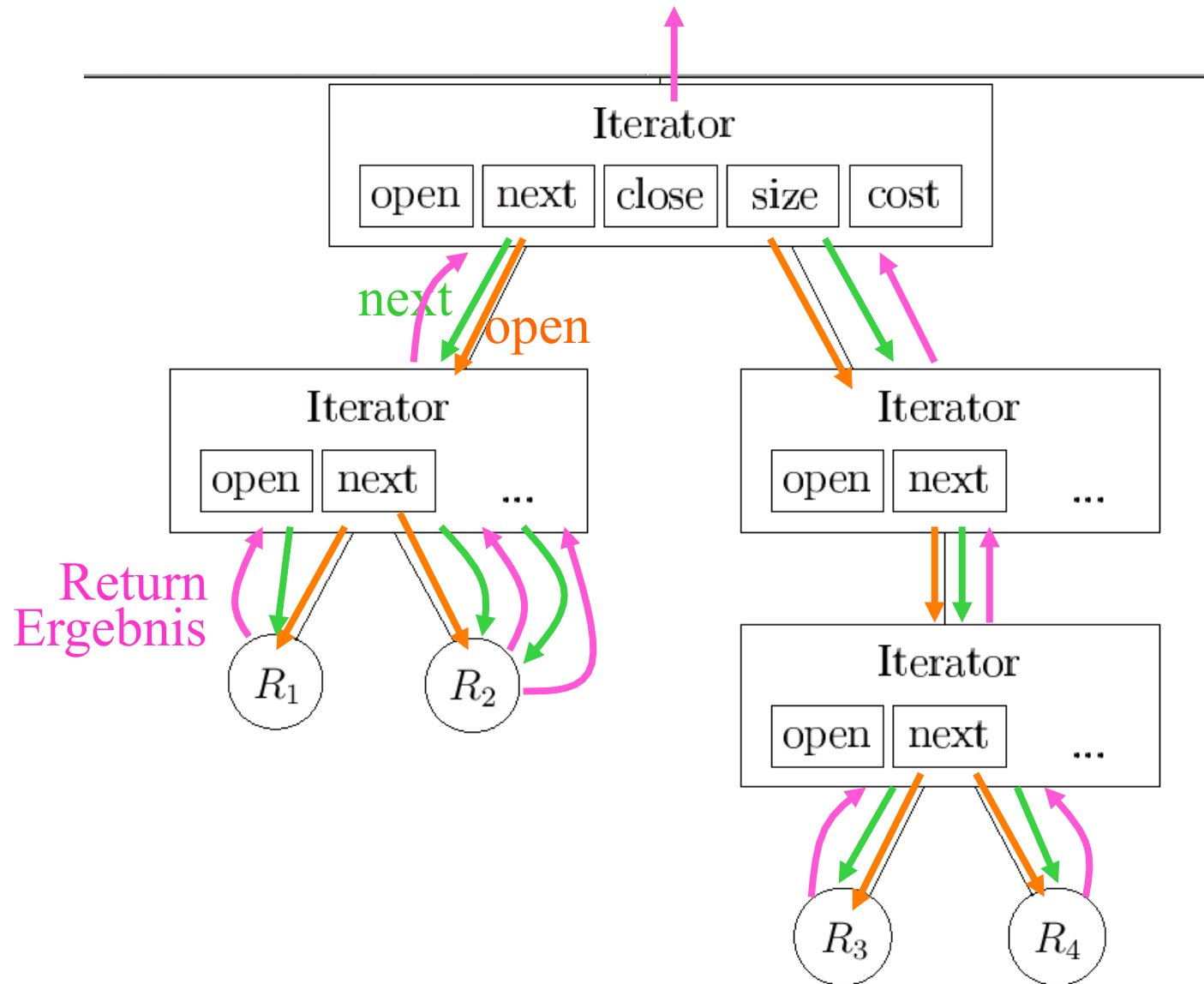


8.12: Elimination des abhängigen Joins

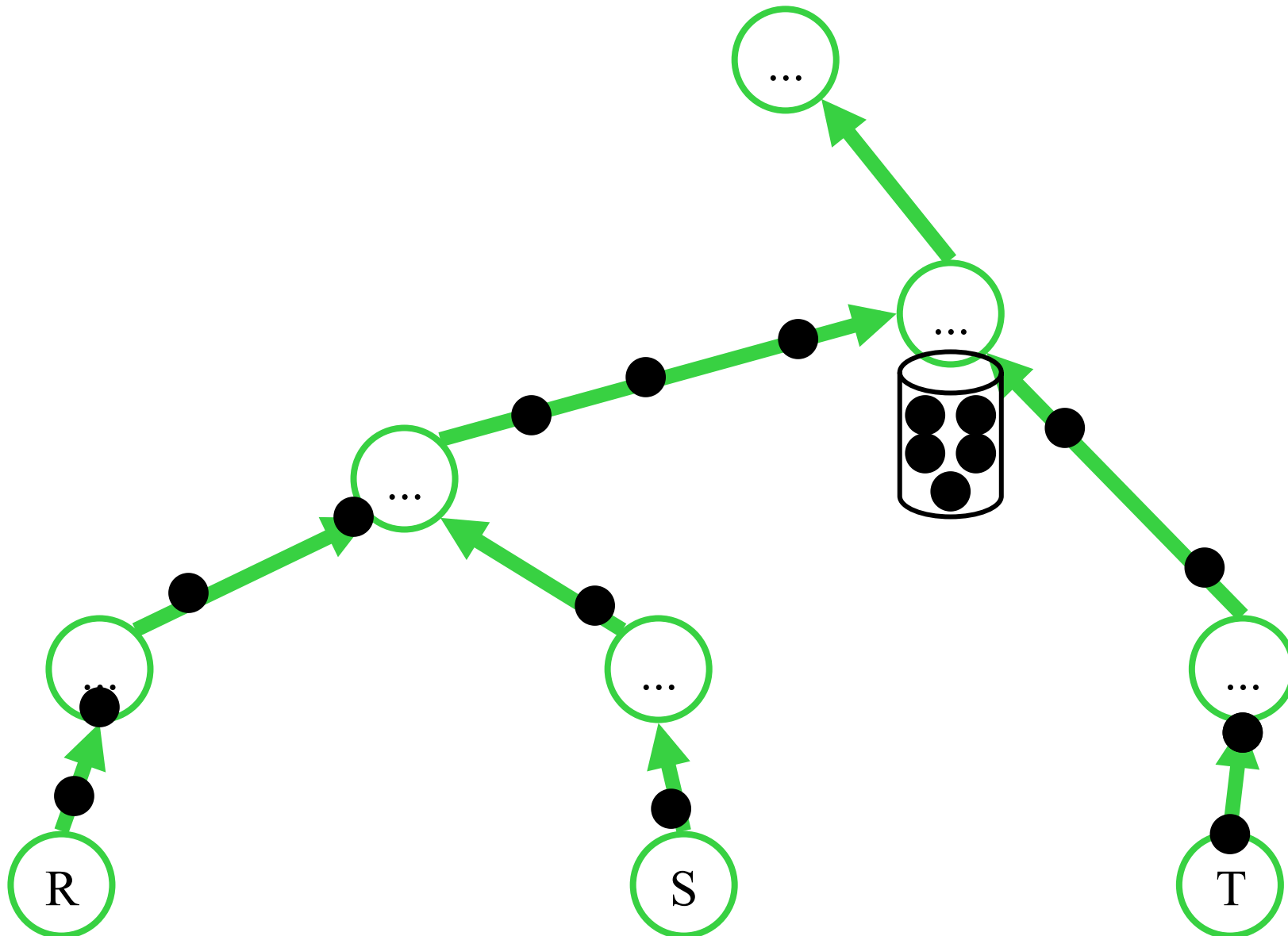
Entkoppelung rechter Seite von linker Seite: optional



Pull-basierte Anfrageauswertung



Pipelining vs. Pipeline-Breaker



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			S		
A	B	C	C	D	E
a_1	b_1	c_1	c_1	d_1	e_1
a_2	b_2	c_2	c_3	d_2	e_2
a_3	b_3	c_1	c_4	d_3	e_3
a_4	b_4	c_2	c_5	d_4	e_4
a_5	b_5	c_3	c_7	d_5	e_5
a_6	b_6	c_2	c_8	d_6	e_6
a_7	b_7	c_6	c_5	d_7	e_7

 \bowtie

$R \bowtie S$				
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

The diagram illustrates the natural join of two relations, R and S. Relation R has attributes A, B, and C, and relation S has attributes C, D, and E. The natural join R ⋈ S is shown as a table with attributes A, B, C, D, and E. The result of the join consists of three rows: (a1, b1, c1, d1, e1), (a3, b3, c1, d1, e1), and (a5, b5, c3, d2, e2). The join is performed by matching the common attribute C between the two relations.

Implementierung der Verbindung: Strategien

J1 nested (inner-outer) loop

- „brute force“-Algorithmus

```
foreach  $r \in R$ 
  foreach  $s \in S$ 
    if  $s.B = r.A$  then  $Res := Res \cup (r \circ s)$ 
```

iterator NestedLoop_{*p*}

open

- Öffne die linke Eingabe

next

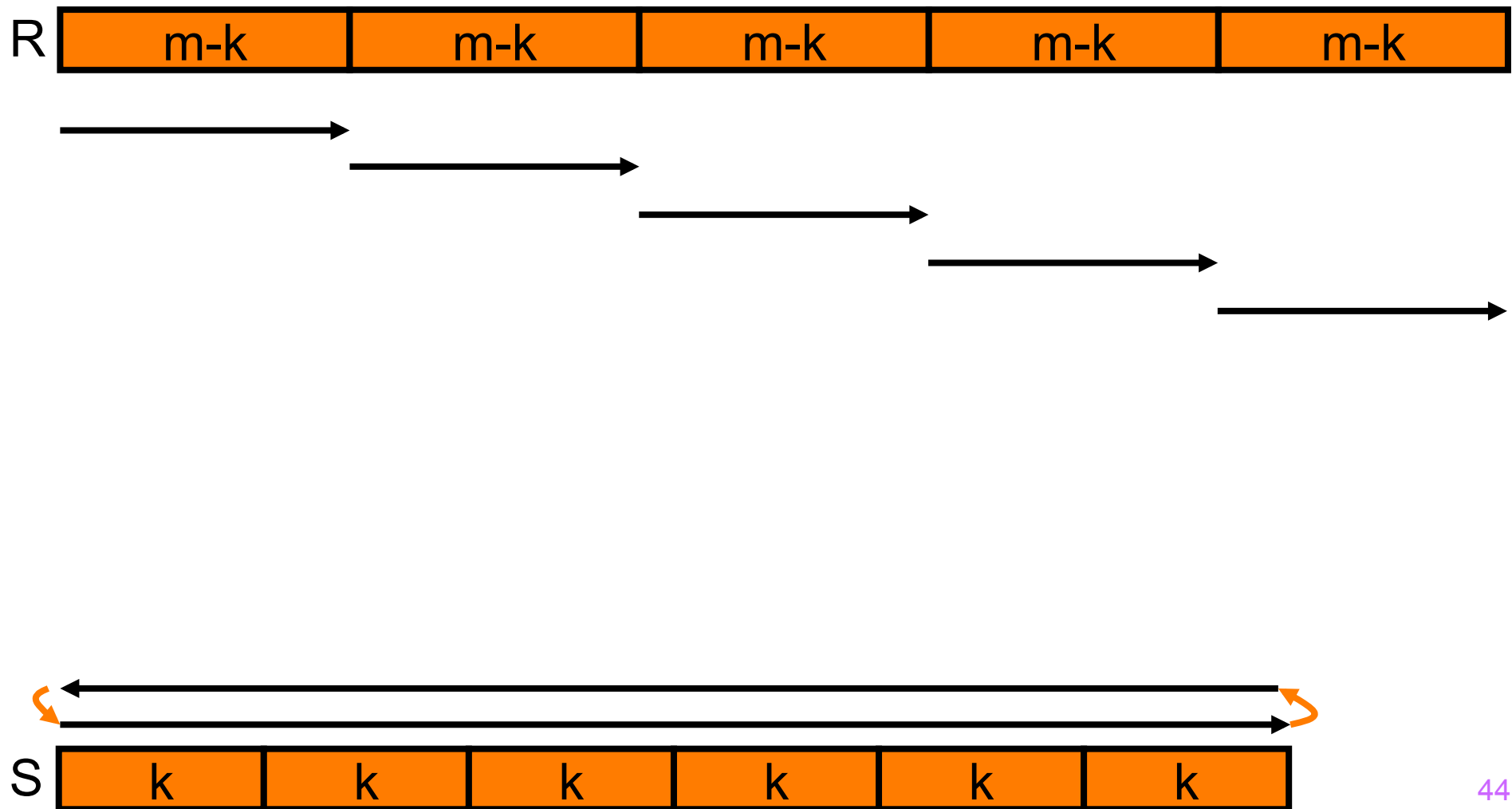
- Rechte Eingabe geschlossen?
 - Öffne sie
- Fordere rechts solange Tupel an, bis Bedingung *p* erfüllt ist
- Sollte zwischendurch rechte Eingabe erschöpft sein
 - Schließe rechte Eingabe
 - Fordere nächstes Tupel der linken Eingabe an
 - Starte **next** neu
- Gib den Verbund von aktuellem linken und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

close

- Schließe beide Eingabequellen

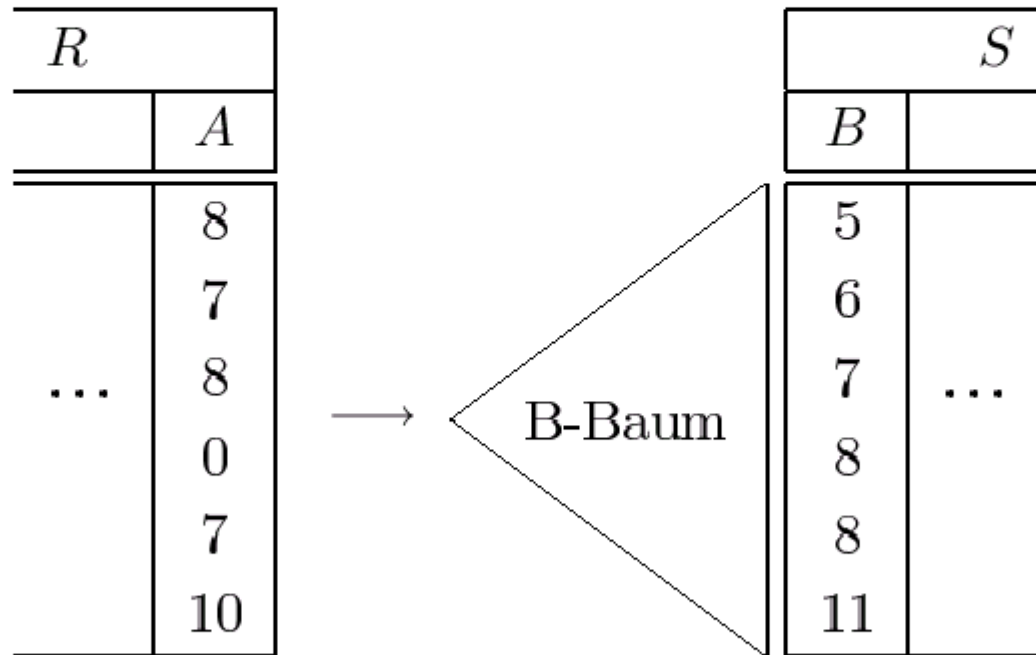
Implementierung der Verbindung: Strategien

Block-Nested Loop Algorithmus



Index-Join

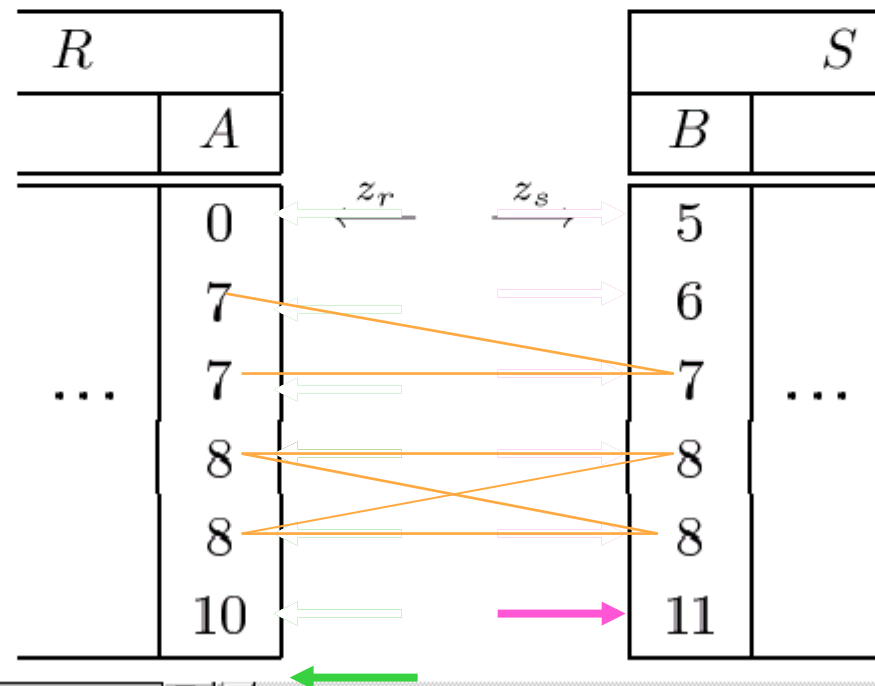
Beispiel:



Der Merge-Join

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

Beispiel:



Implementierung der Verbindung: Strategien

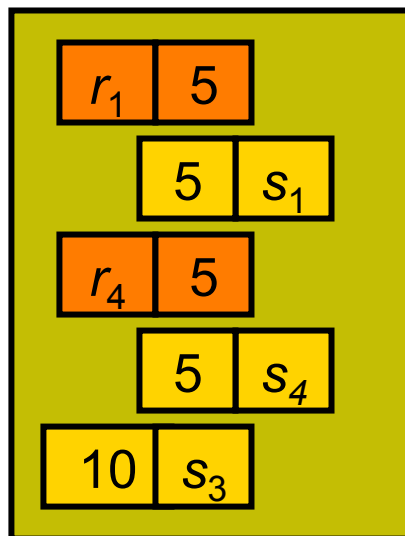
J4 Hash-Join

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

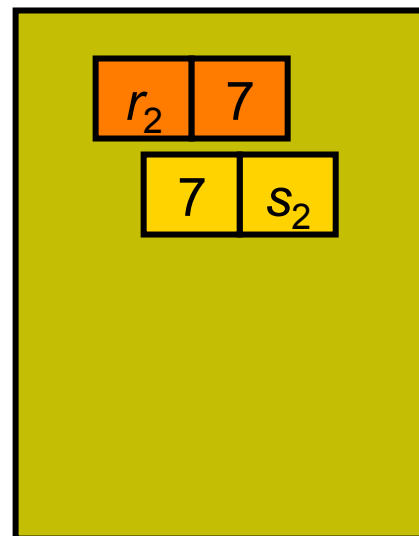
Implementierung der Verbindung: Strategien

R		S	
...	A	B	...
r_1	5	5	s_1
r_2	7	7	s_2
r_3	8	10	s_3
r_4	5	5	s_4

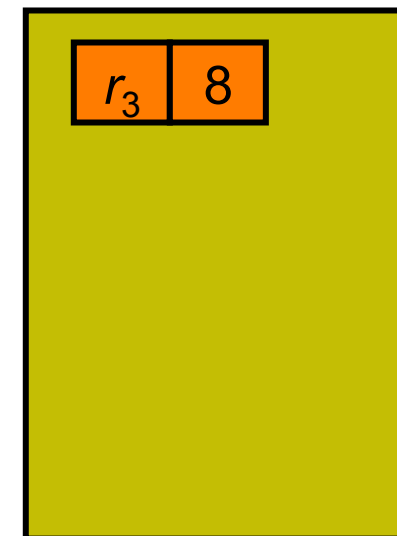
$h(A) \downarrow$ $\downarrow h(B)$



Bucket 1

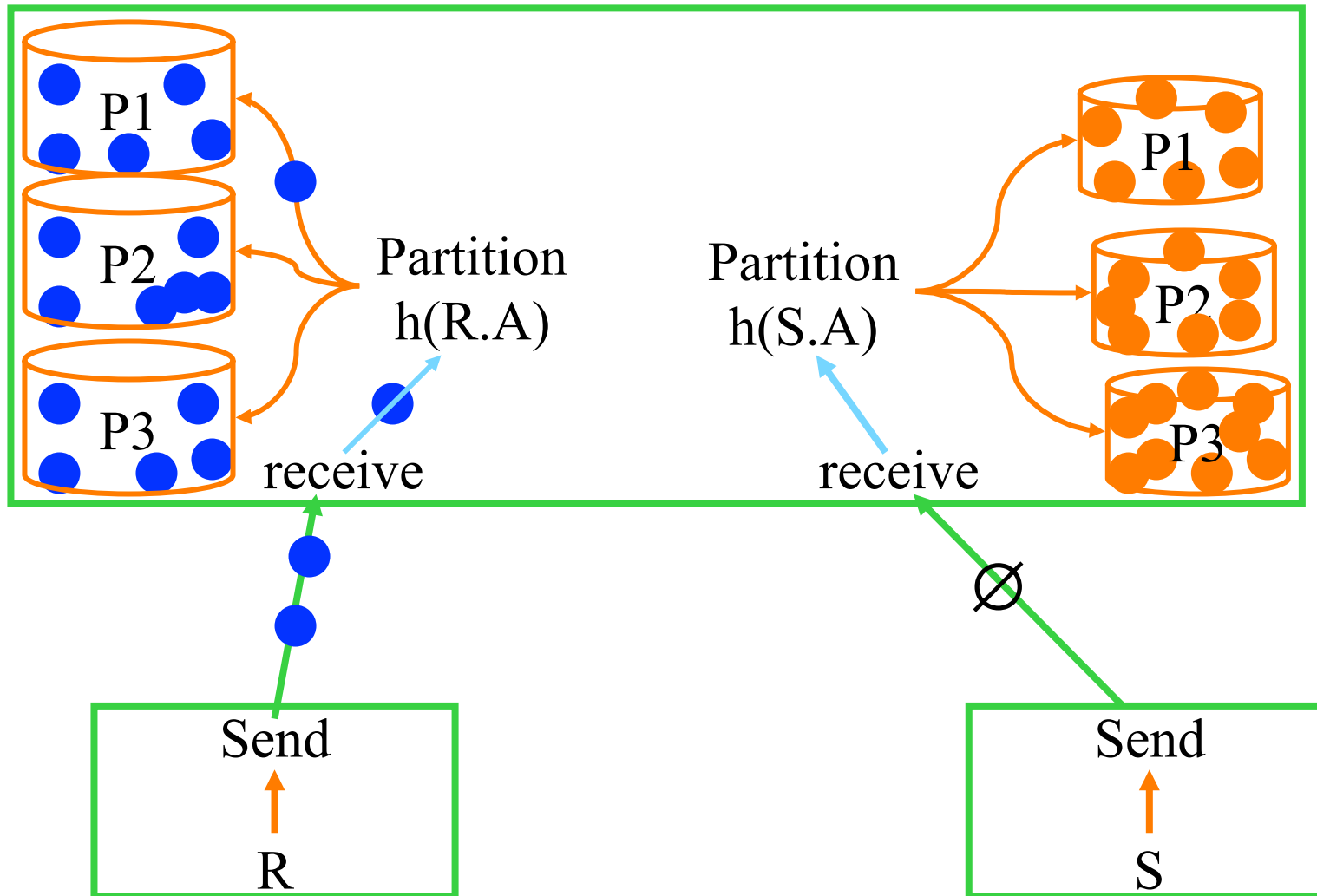


Bucket 2

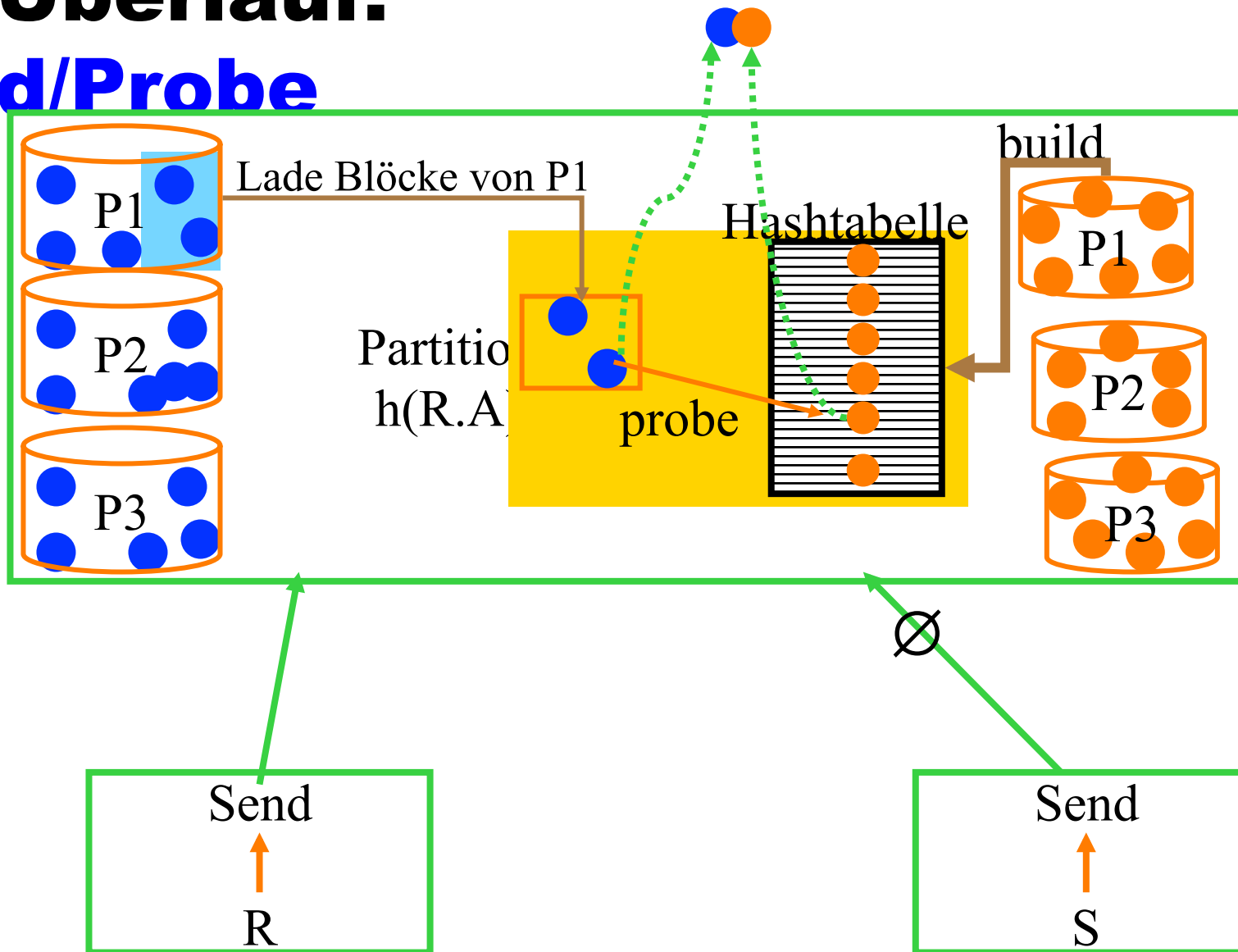


Bucket 3

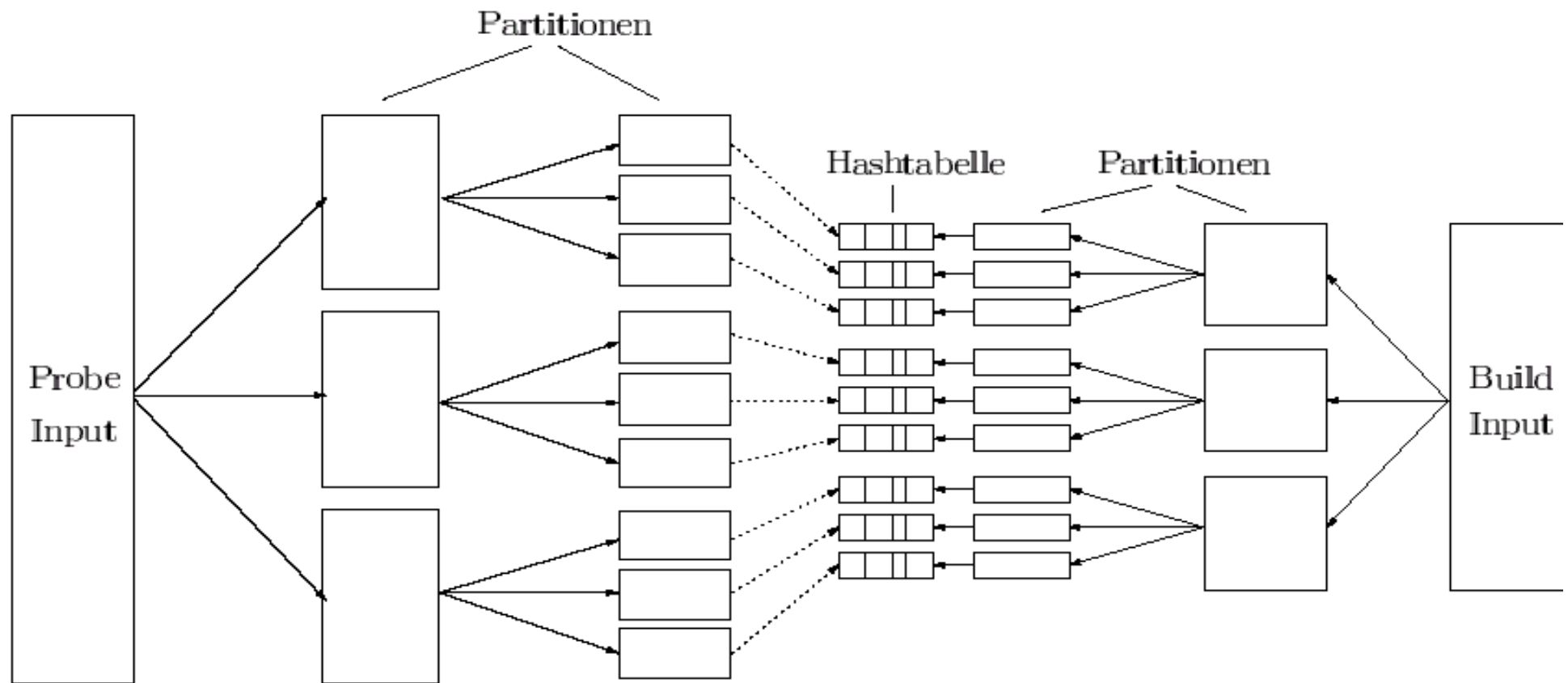
„Normaler“ blockierender Hash-Join mit Überlauf: Partitionieren



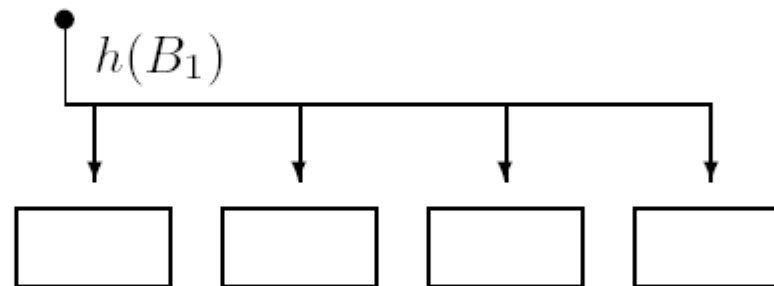
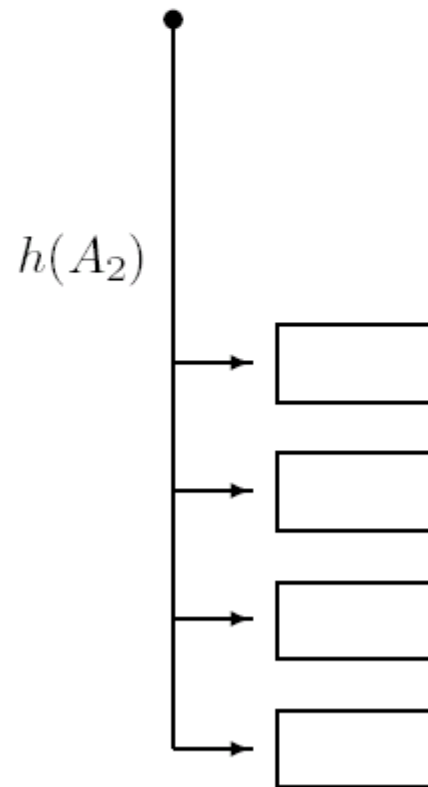
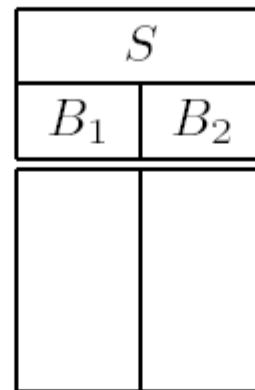
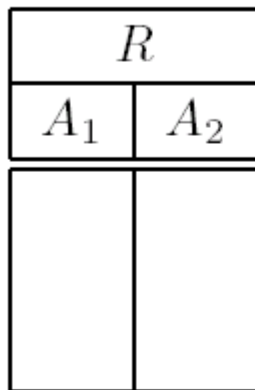
„Normaler“ blockierender Hash-Join mit Überlauf: Build/Probe

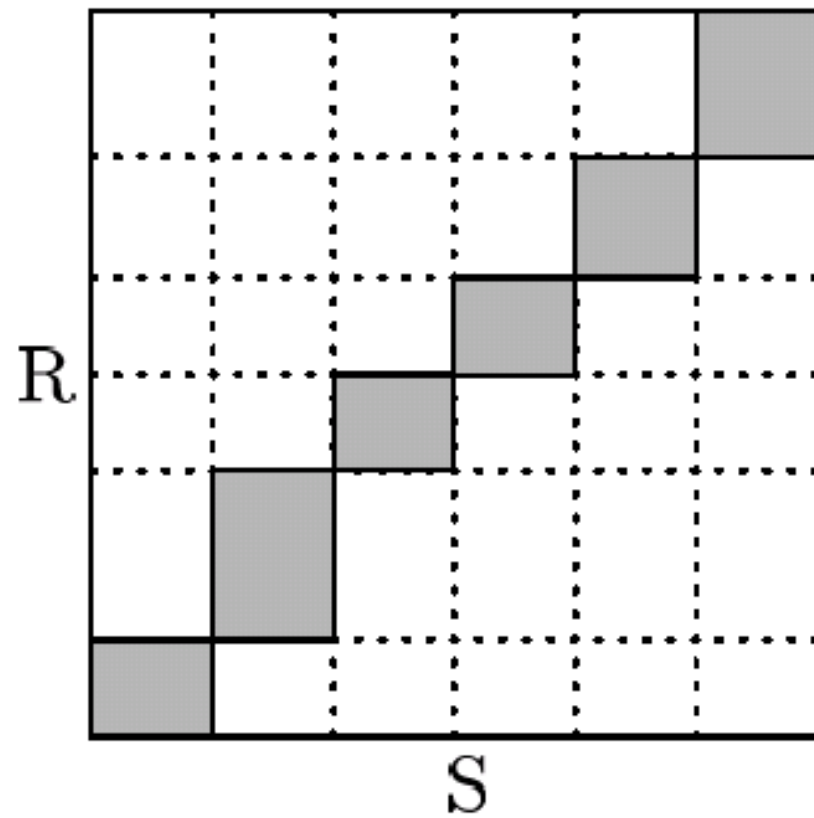
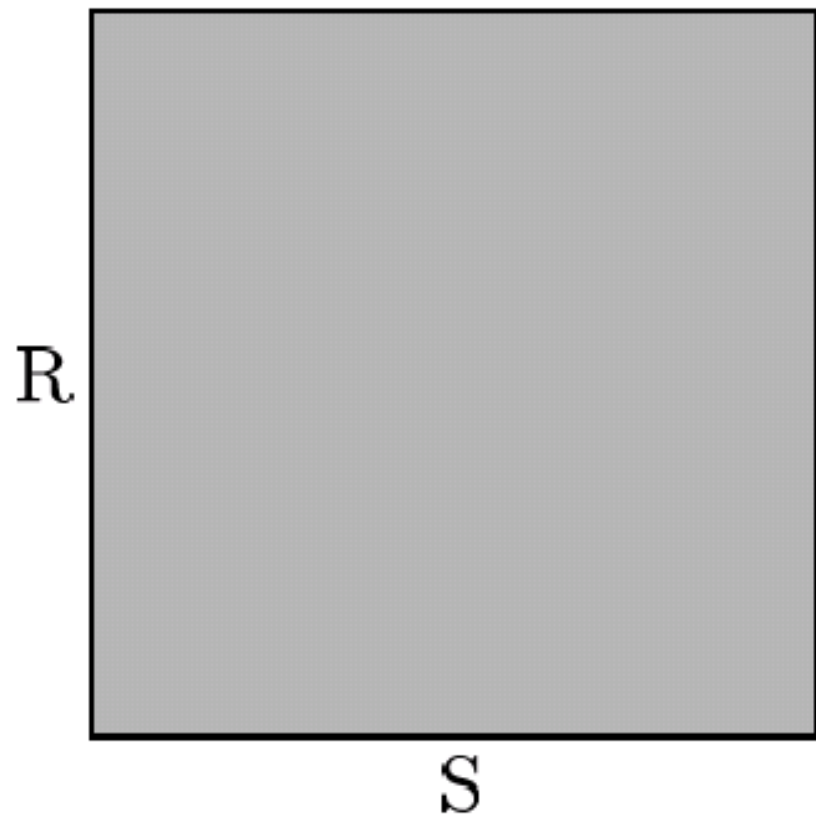


Partitionierung von Relationen

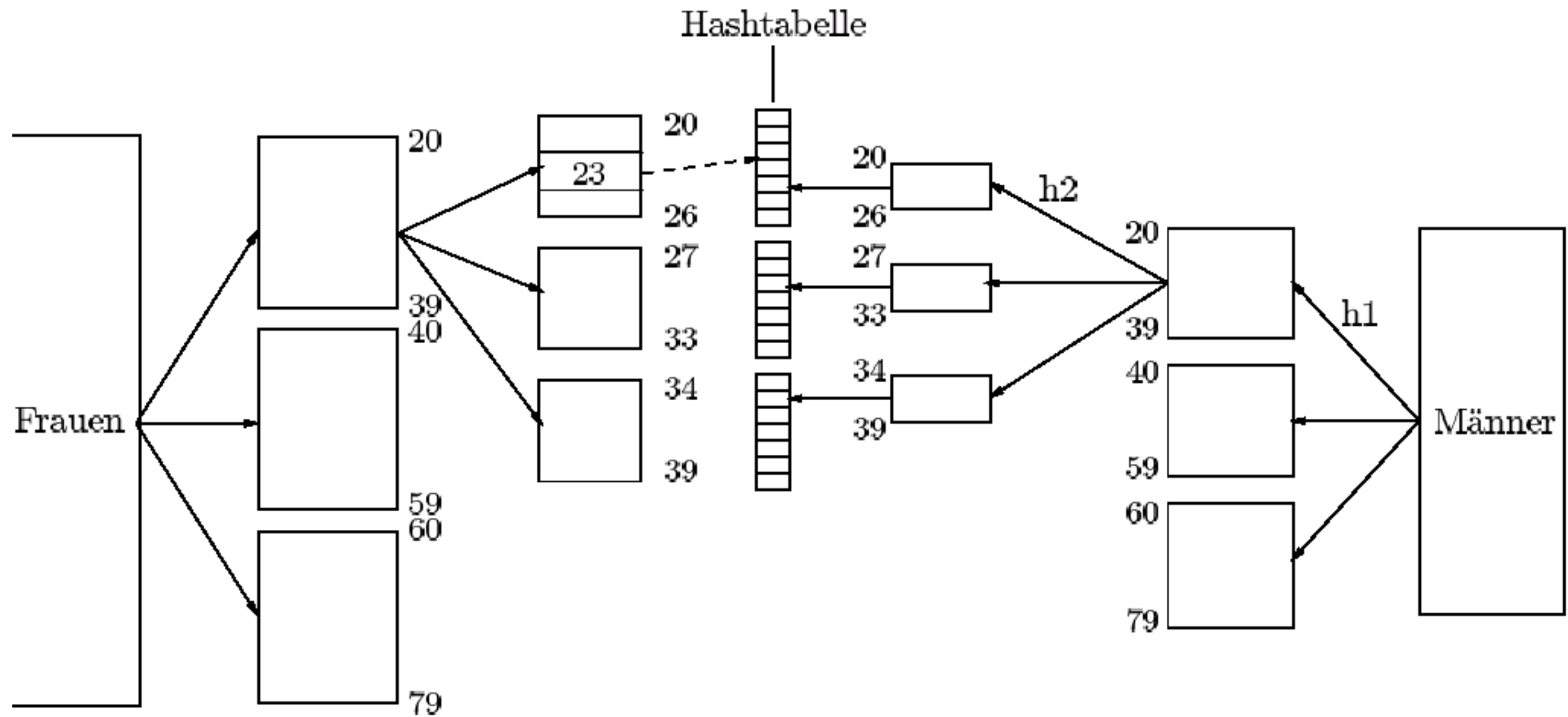


Vergleich der Tupel in der "Diagonalen"





Demonstration der Partitionierung

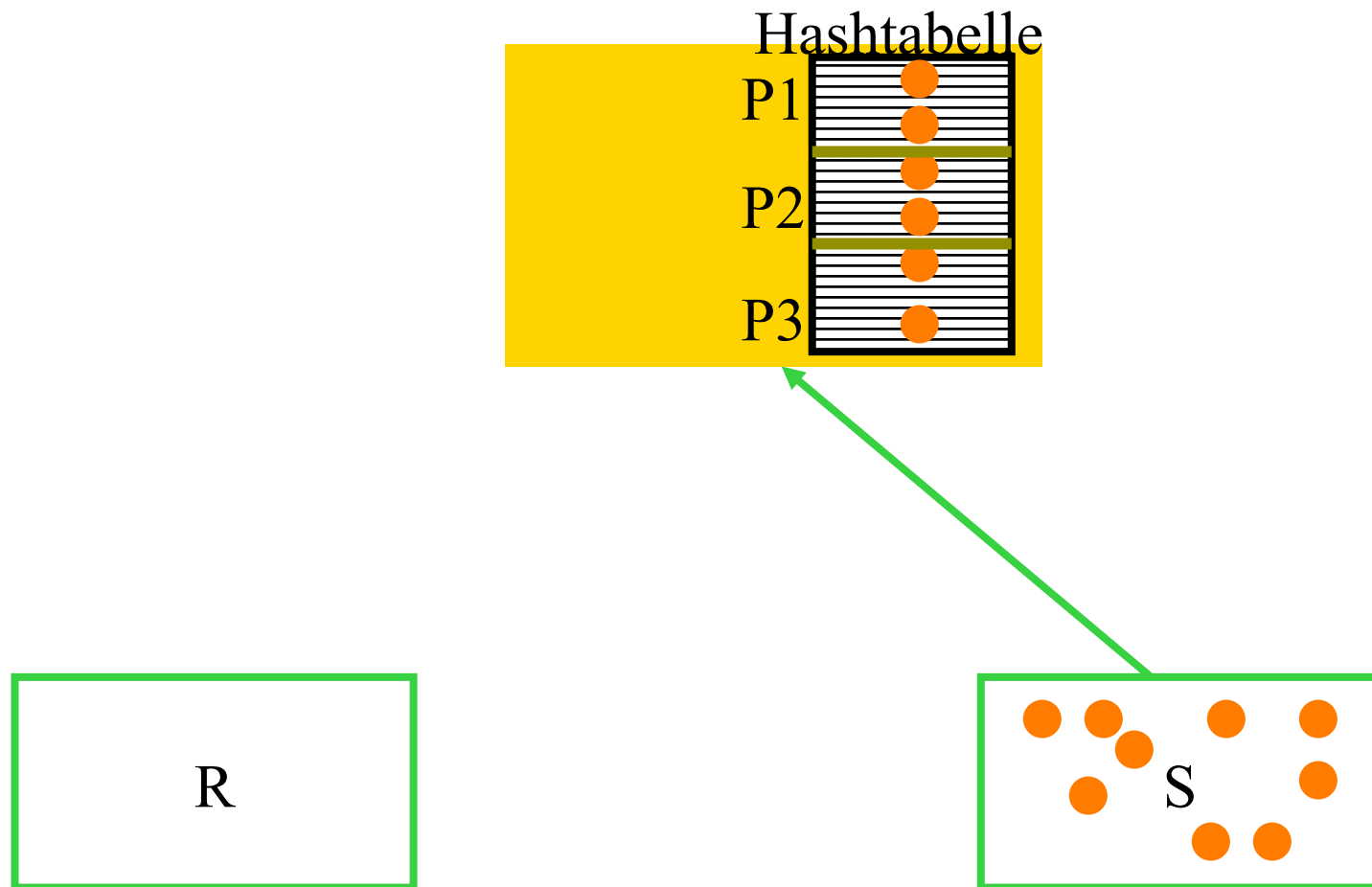


Hybrid Hash-Join

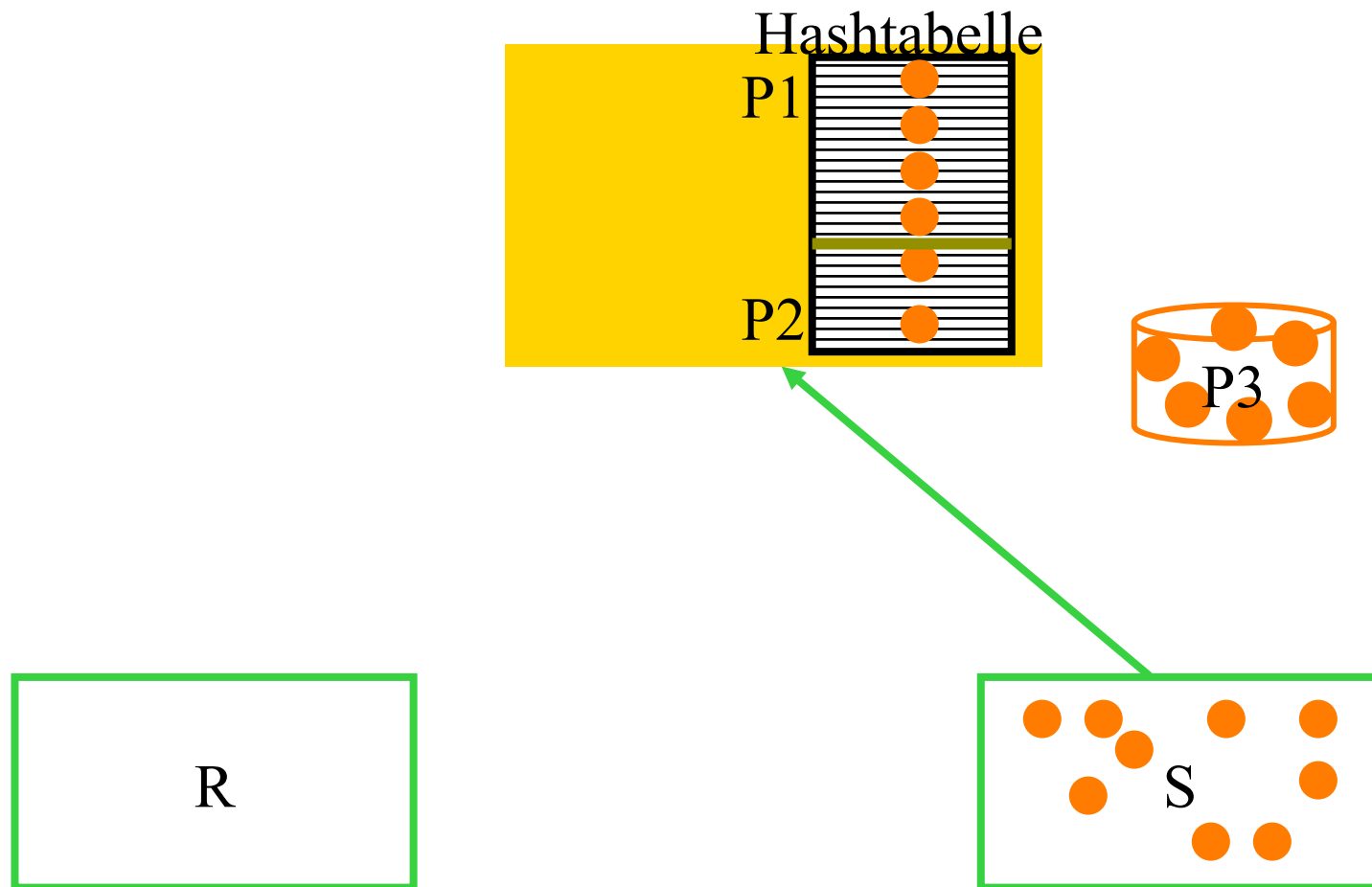
- Fange so an, als wenn der Build-Input S vollständig in den Hauptspeicher passen würde
- Sollte sich dies als zu optimistisch herausstellen, verdränge eine Partition nach der anderen aus dem Hauptspeicher
- Mindestens eine Partition wird aber im Hauptspeicher verbleiben
- Danach beginnt die Probe-Phase mit der Relation R
- Jedes Tupel aus R , dessen potentielle Join-Partner im Hauptspeicher sind, wird sogleich verarbeitet

- Hybrid Hash-Join ist dann besonders interessant, wenn der Build-Input knapp größer als der Hauptspeicher ist
 - Kostensprung beim normalen Hash-Join
 - Wird oft auch Grace-Hash-Join genannt, weil er für die Datenbankmaschine Grace in Japan erfunden wurde

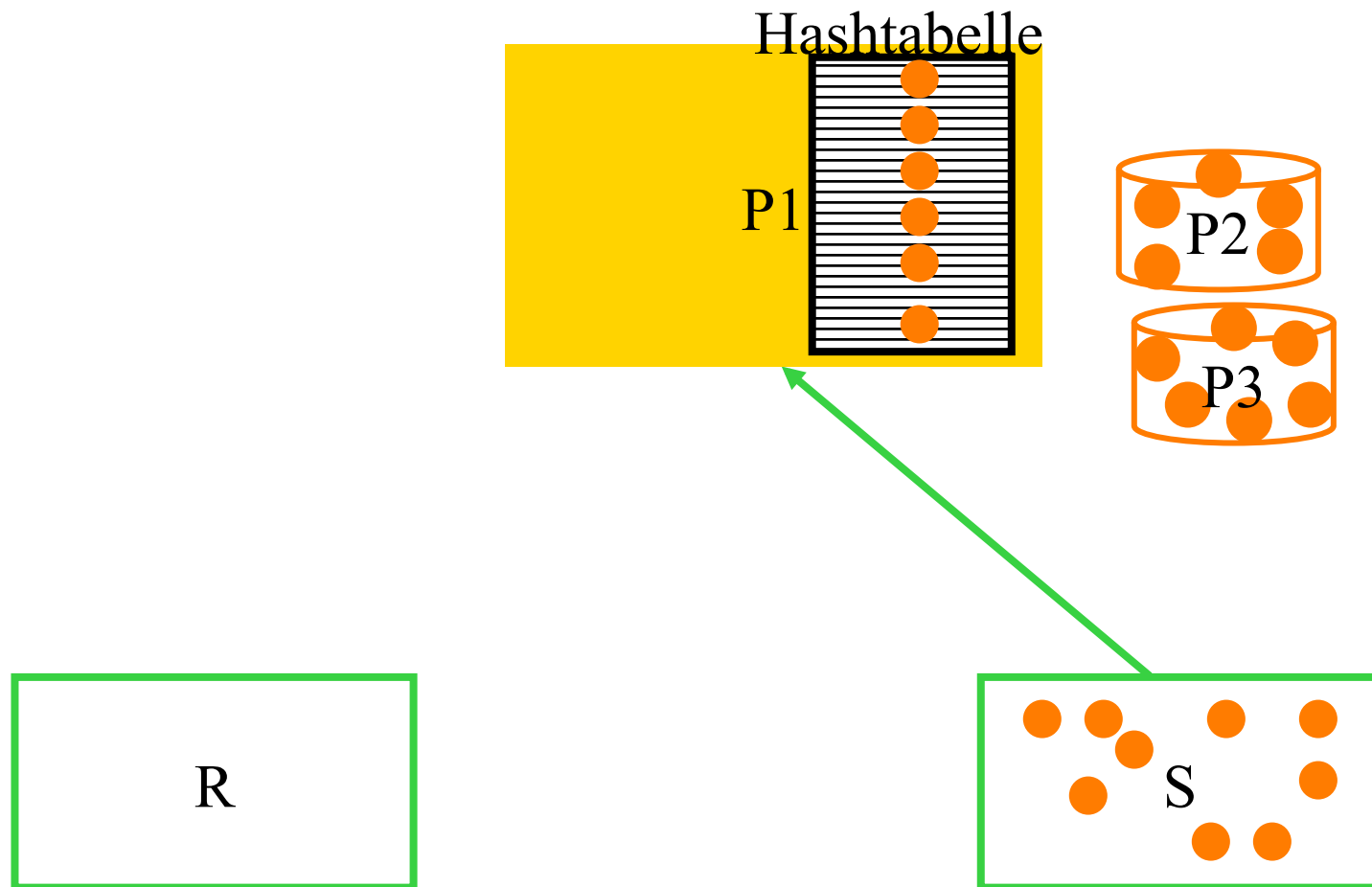
Hybrid Hash-Join



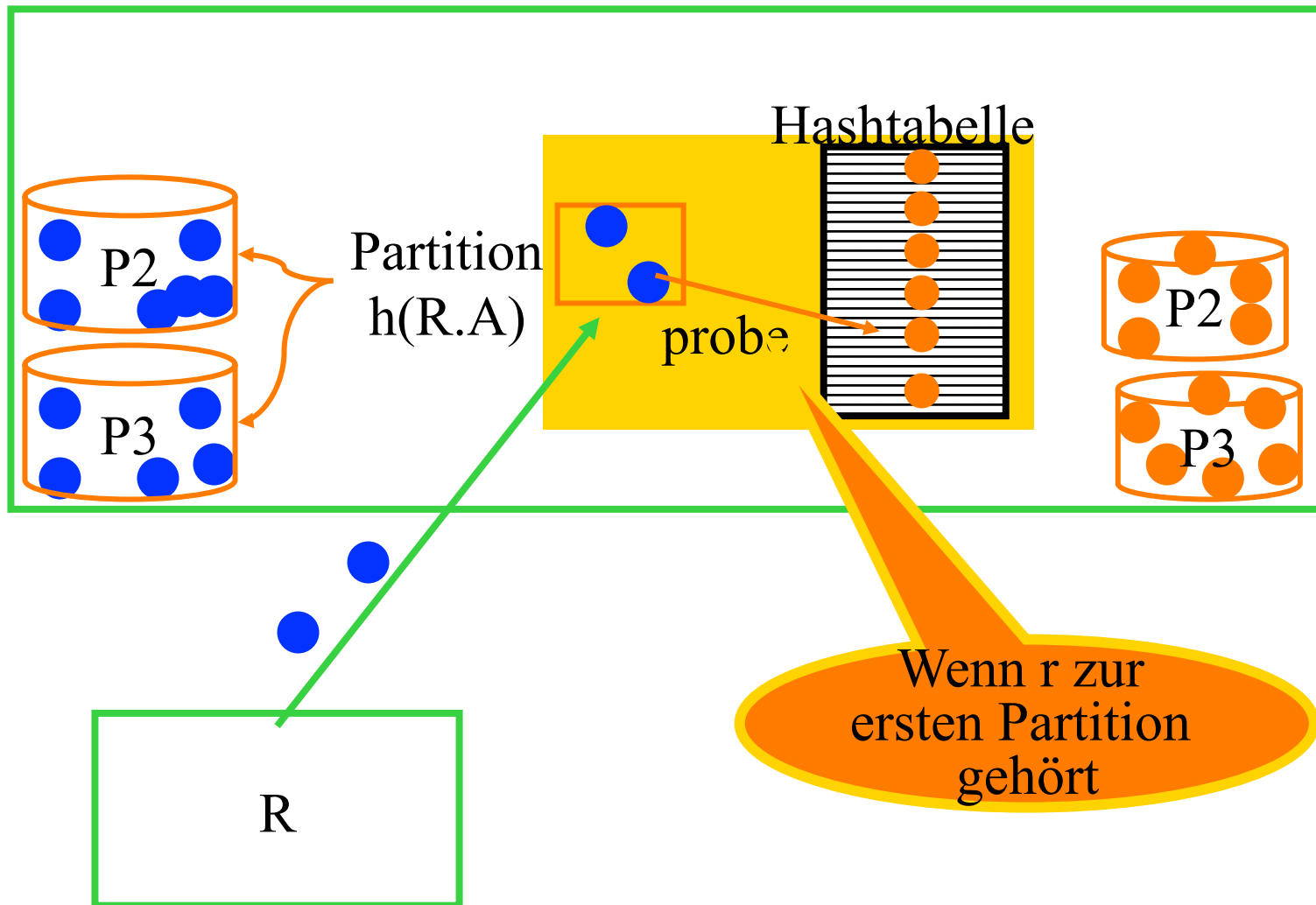
Hybrid Hash-Join



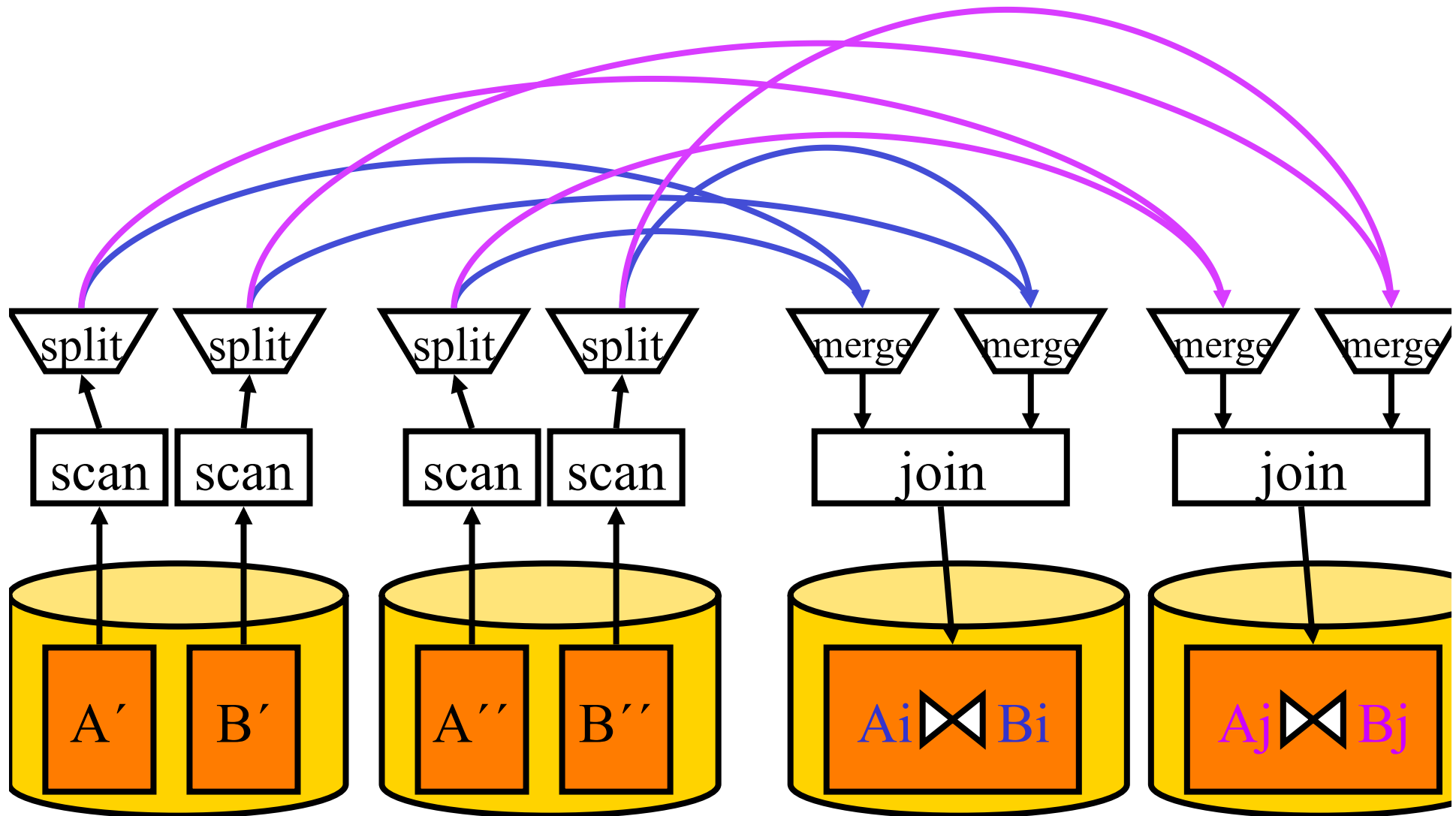
Hybrid Hash-Join



Hybrid Hash-Join



Parallele Anfragebearbeitung: Hash Join



Paralleler Hash Join – im Detail

1. An jeder Station werden mittels Hash-Funktion h_1 die jeweiligen Partitionen von A und B in A_1, \dots, A_k und B_1, \dots, B_k zerlegt
 - h_1 muss so gewählt werden, dass alle A_i 's aller Stationen in den Hauptspeicher passen
2. Für alle $1 \leq i \leq n$: Berechne jetzt den Join von A_i mit B_i wie folgt
 - a. Wende eine weitere Hash-Funktion h_2 an, um A_i auf die l Stationen zu verteilen
 - Sende Tupel t an Station $h_2(t)$
 - b. Eintreffende A_i -Tupel werden in die Hash-Tabelle an der jeweiligen Station eingefügt
 - c. Sobald alle Tupel aus A_i „verschickt“ sind, wird h_2 auf B_i angewendet und Tupel t an Station $h_2(t)$ geschickt
 - d. Sobald ein B_i -Tupel eintrifft, werden in der A_i -Hashtabelle seine Joinpartner ermittelt.

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

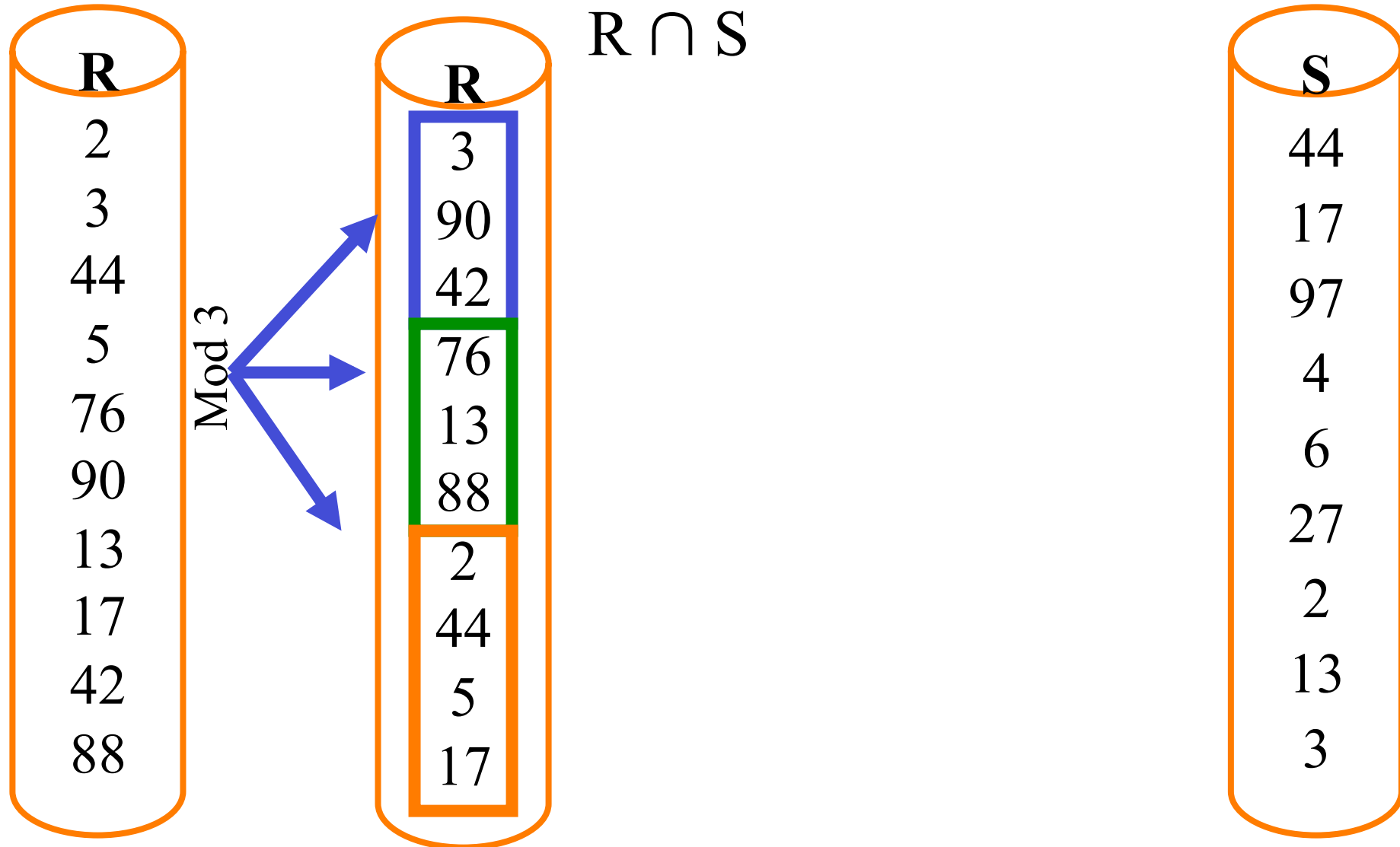
- R**
- 2
 - 3
 - 44
 - 5
 - 76
 - 90
 - 13
 - 17
 - 42
 - 88

$R \cap S$

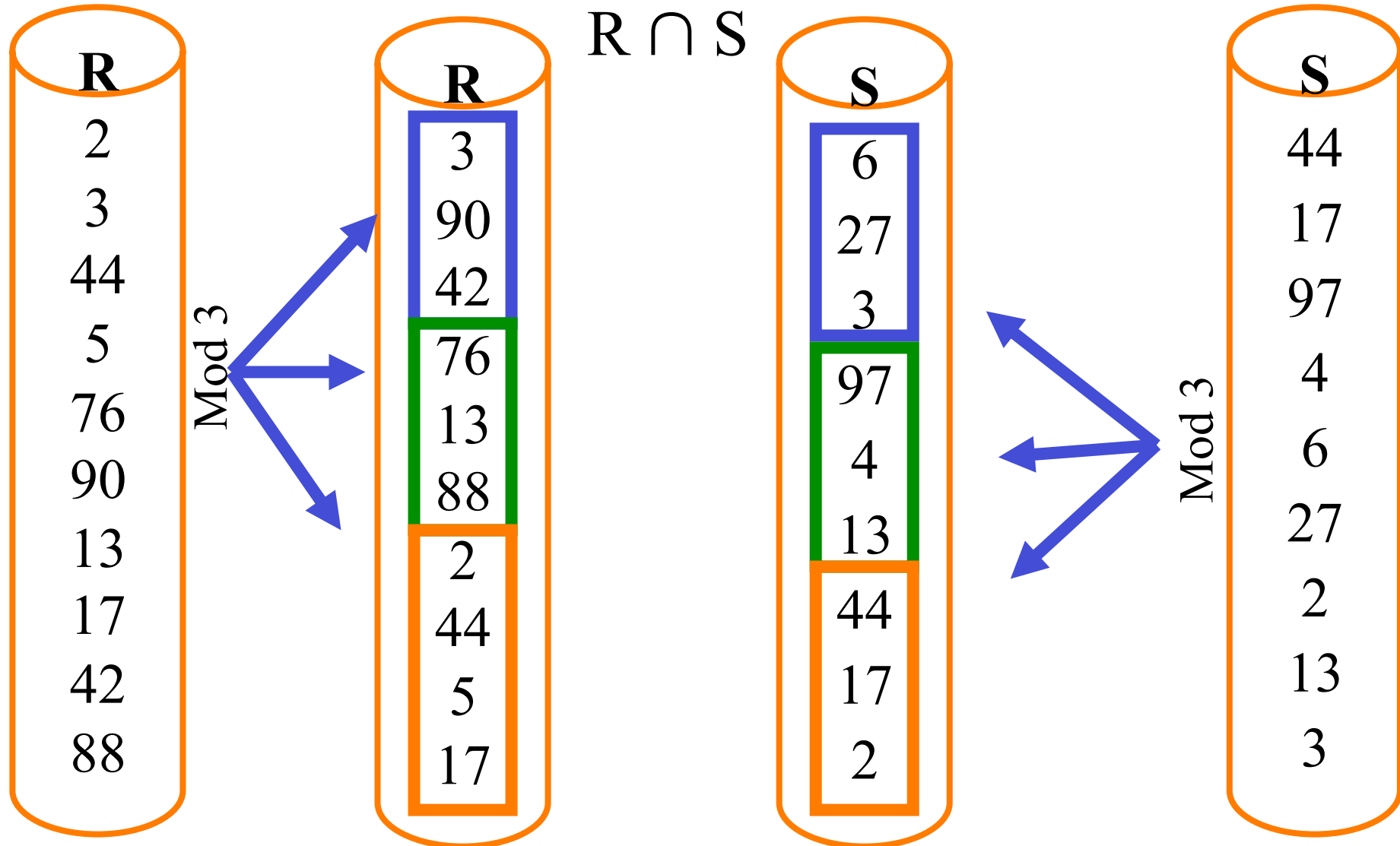
- Nested Loop: $O(N^2)$
- Sortieren: $O(N \log N)$
- Partitionieren und Hashing

- S**
- 44
 - 17
 - 97
 - 4
 - 6
 - 27
 - 2
 - 13
 - 3

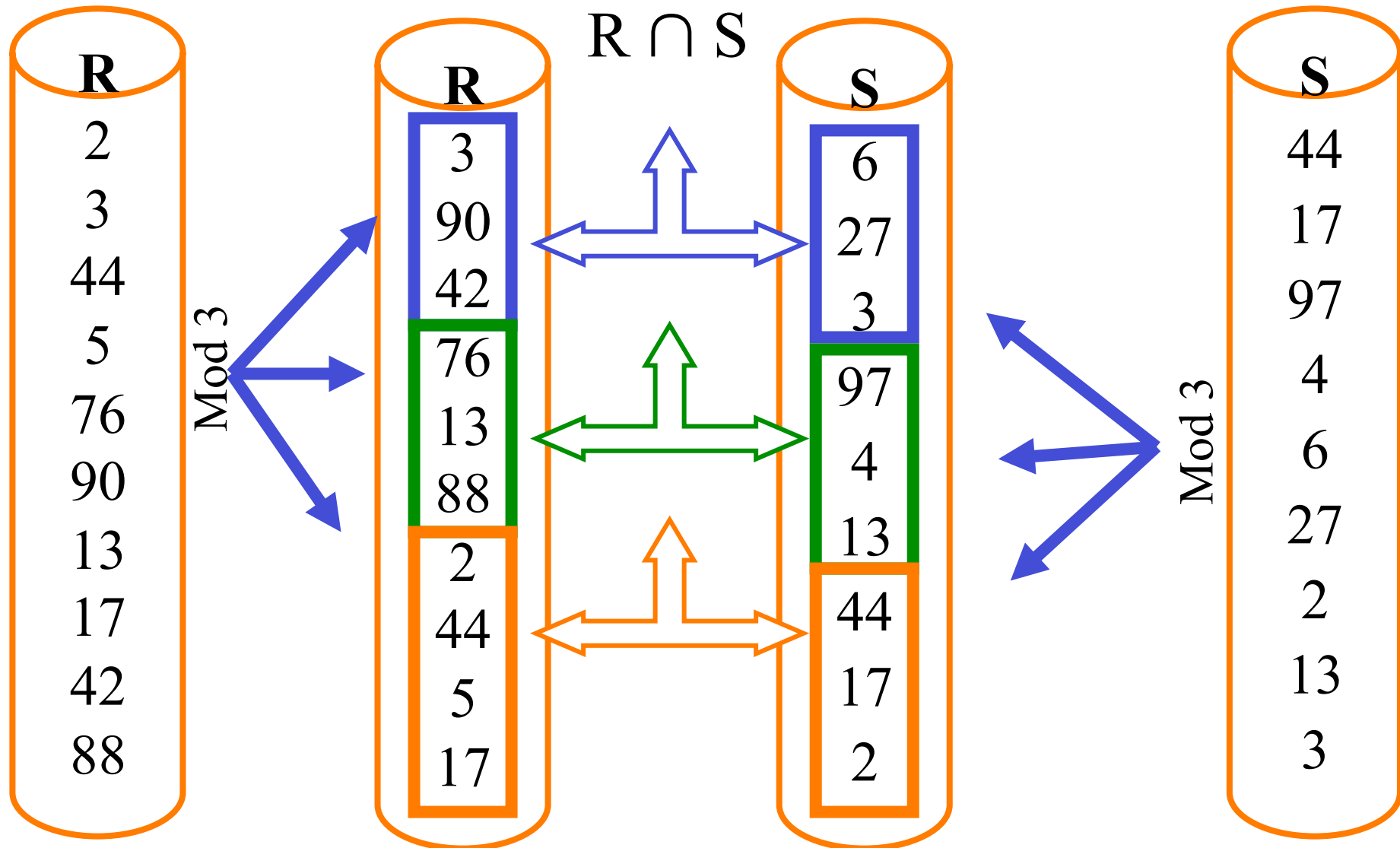
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



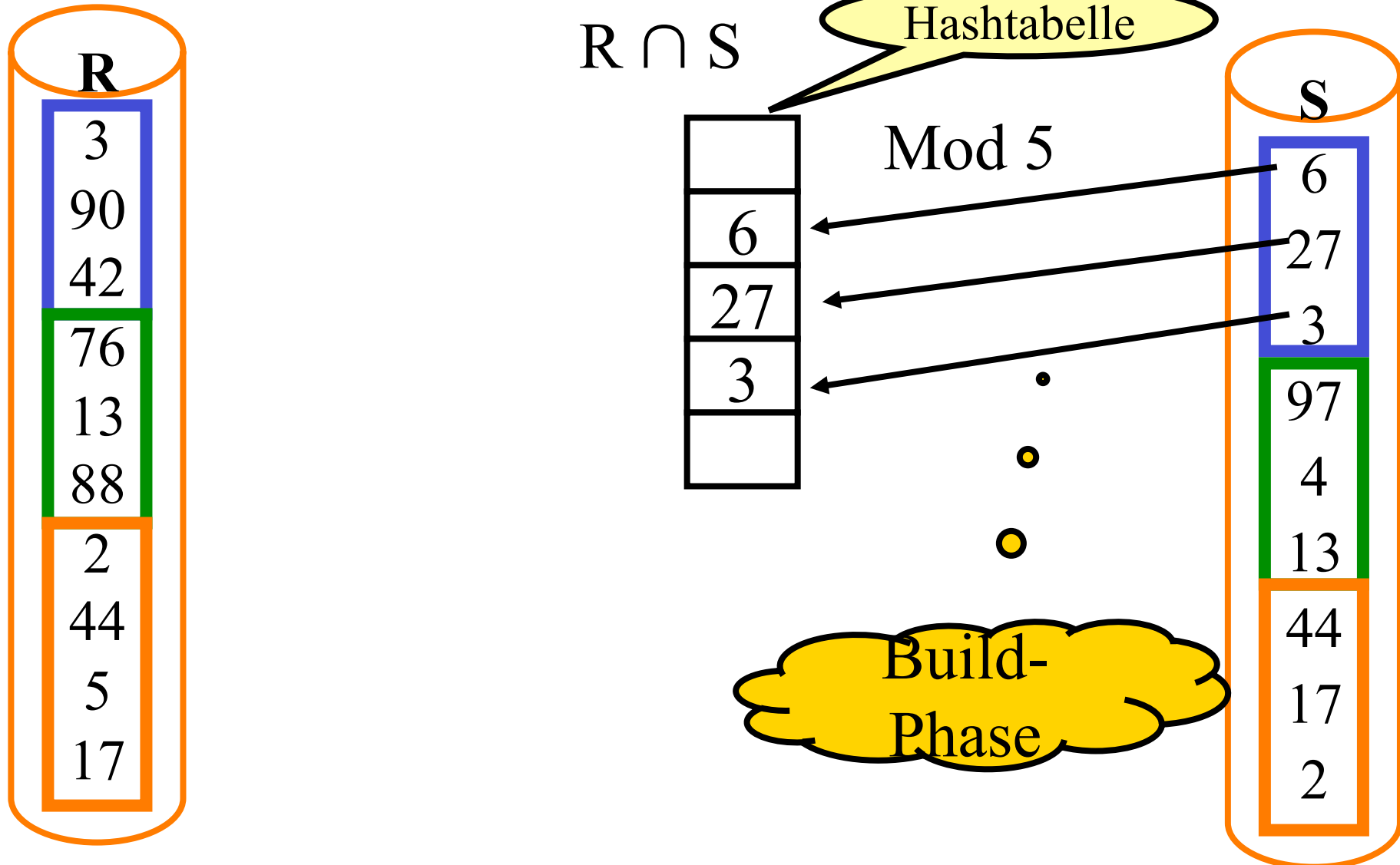
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



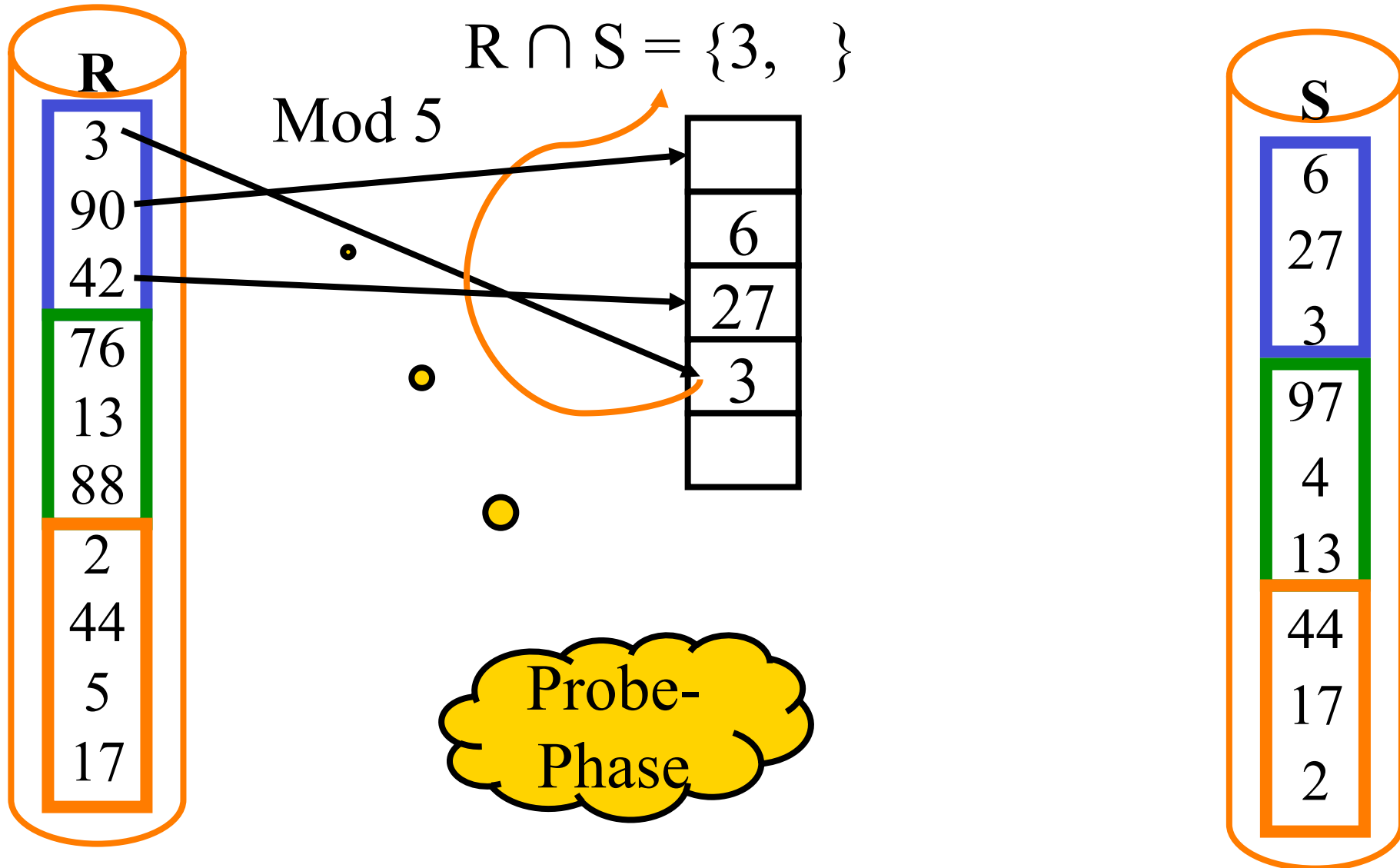
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



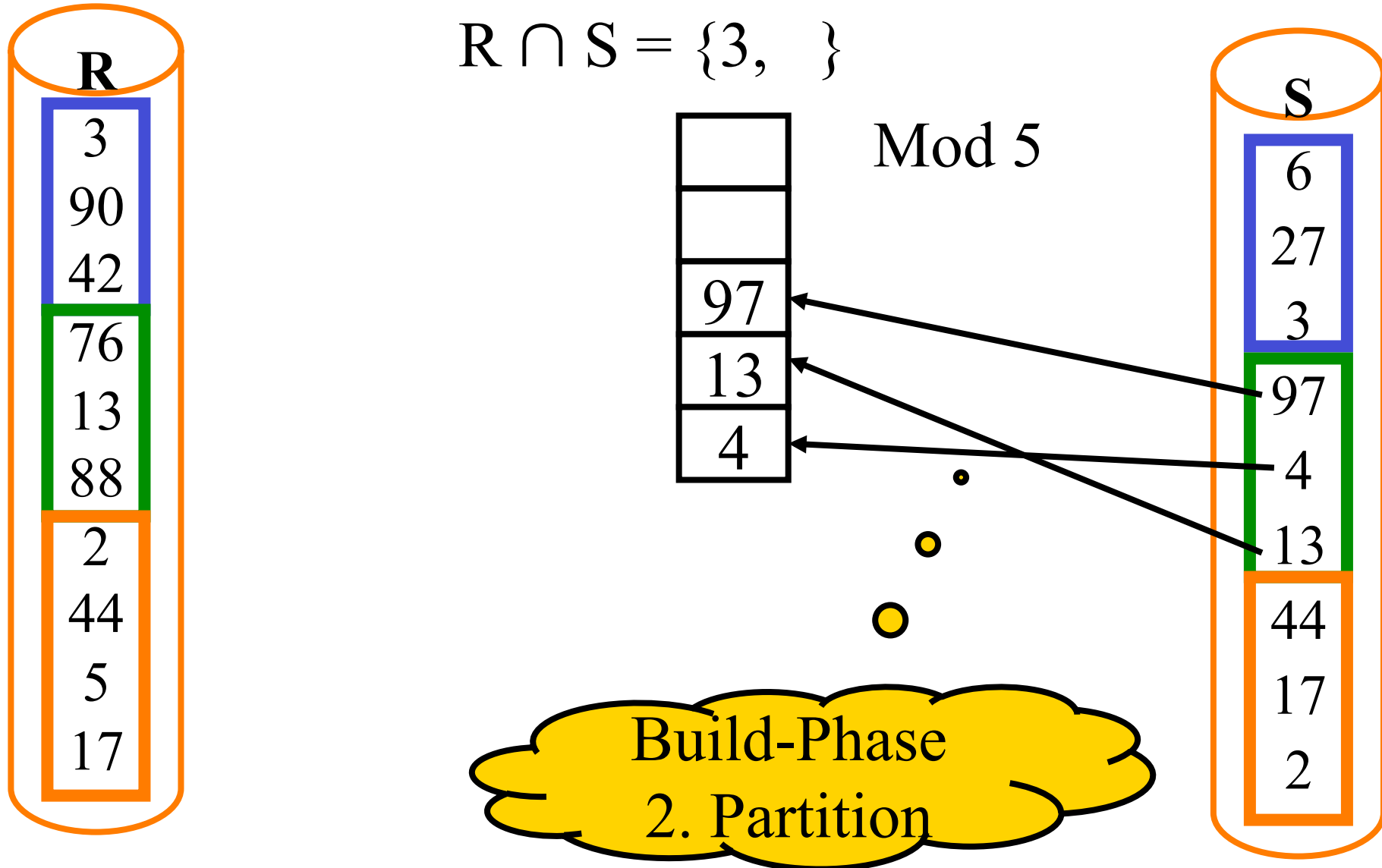
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



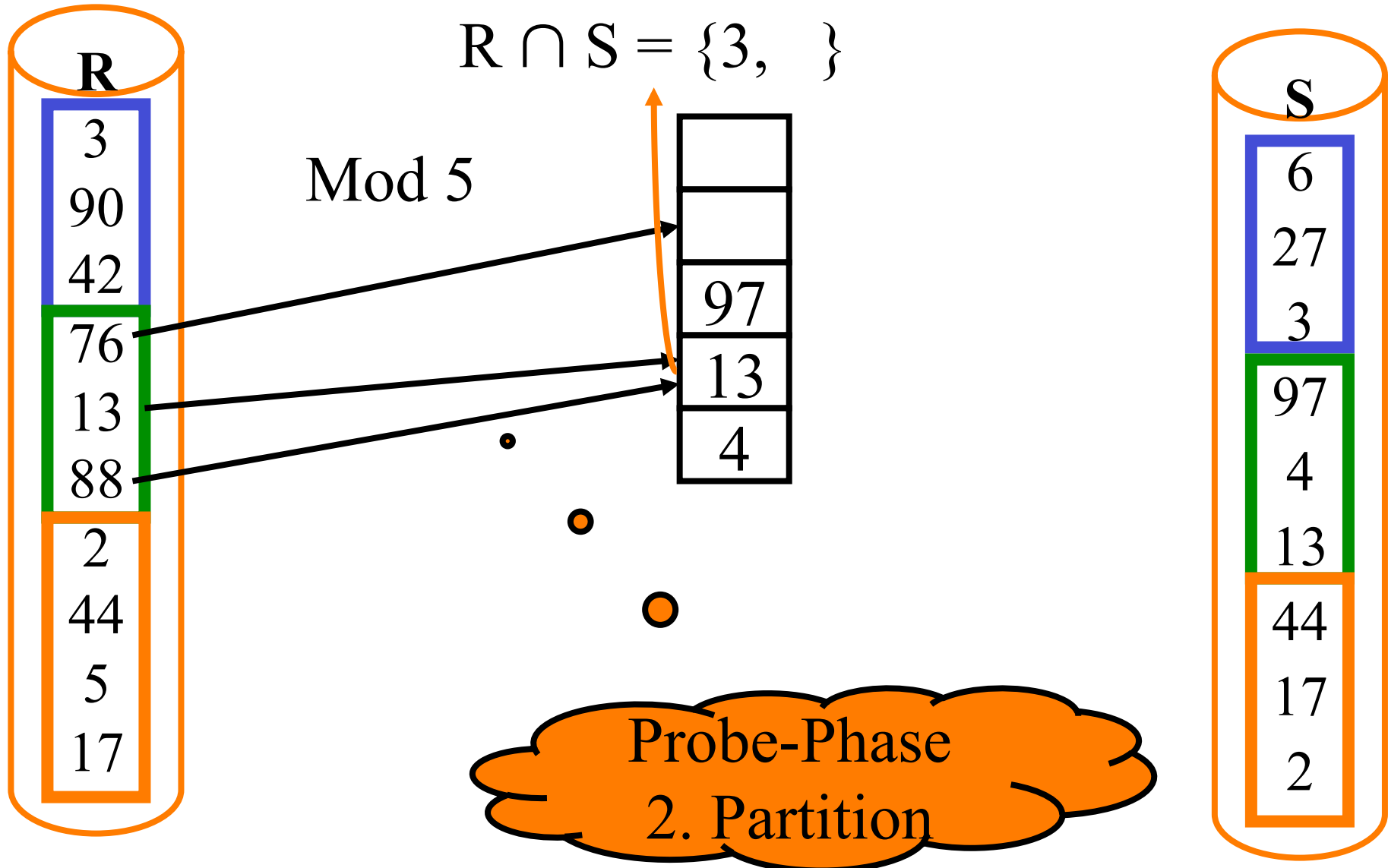
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



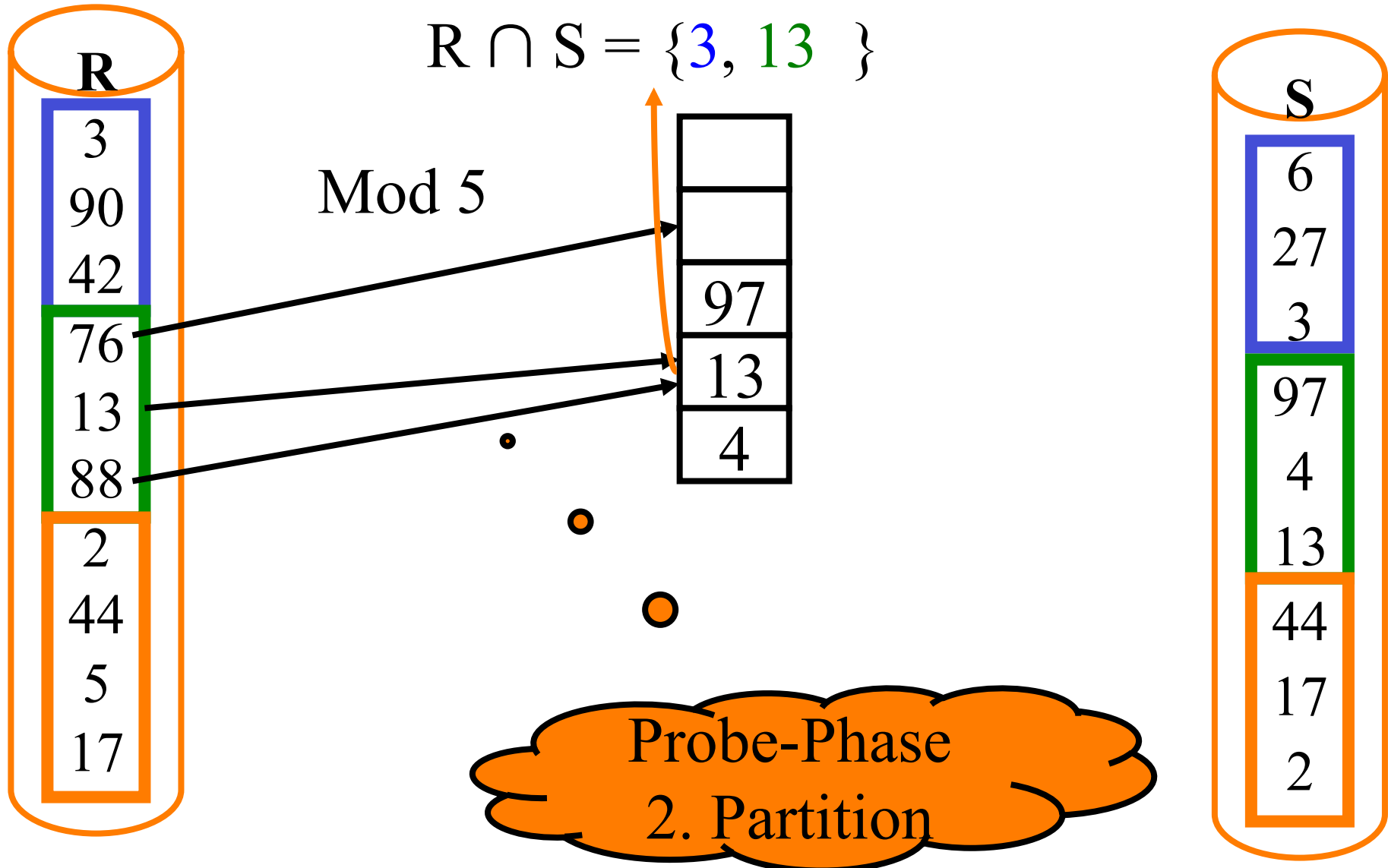
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

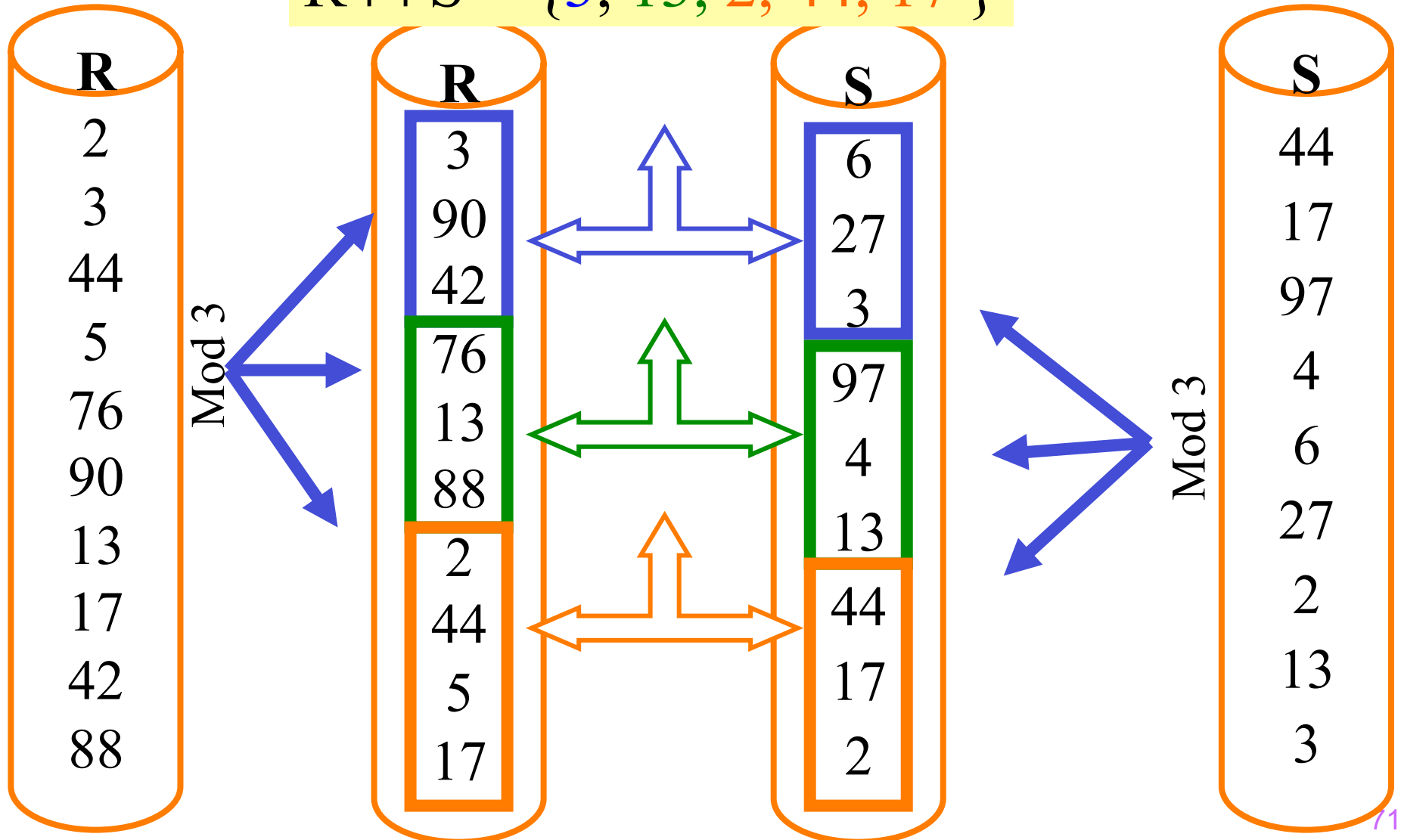


Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

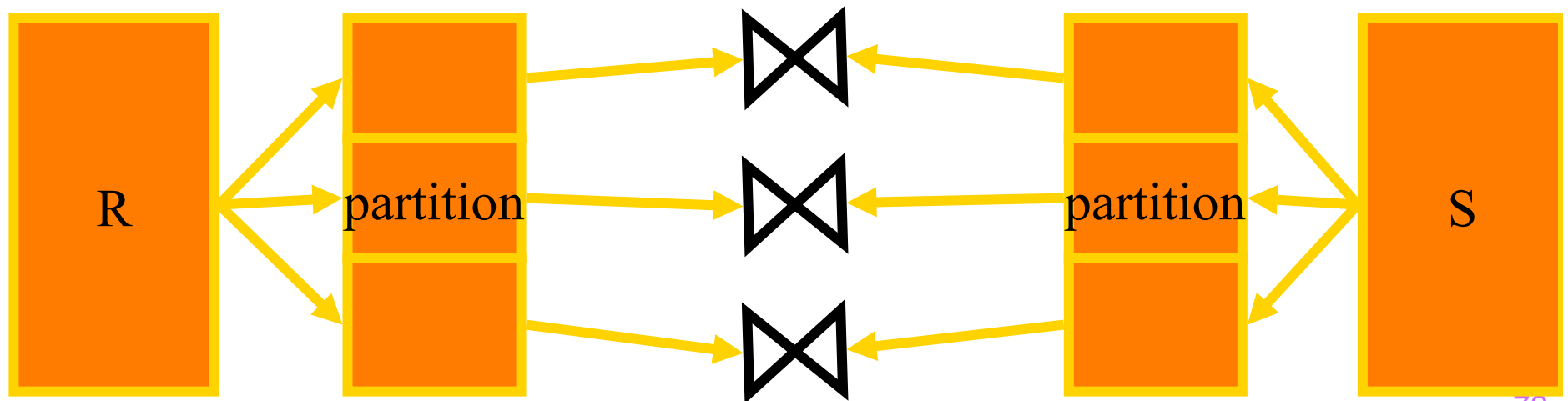
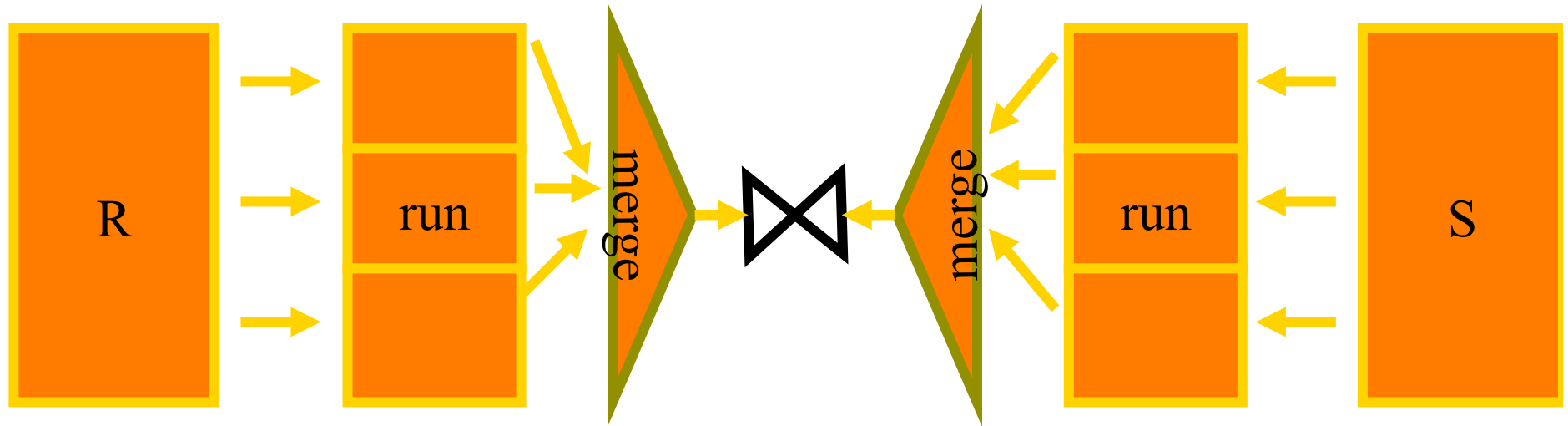


Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

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



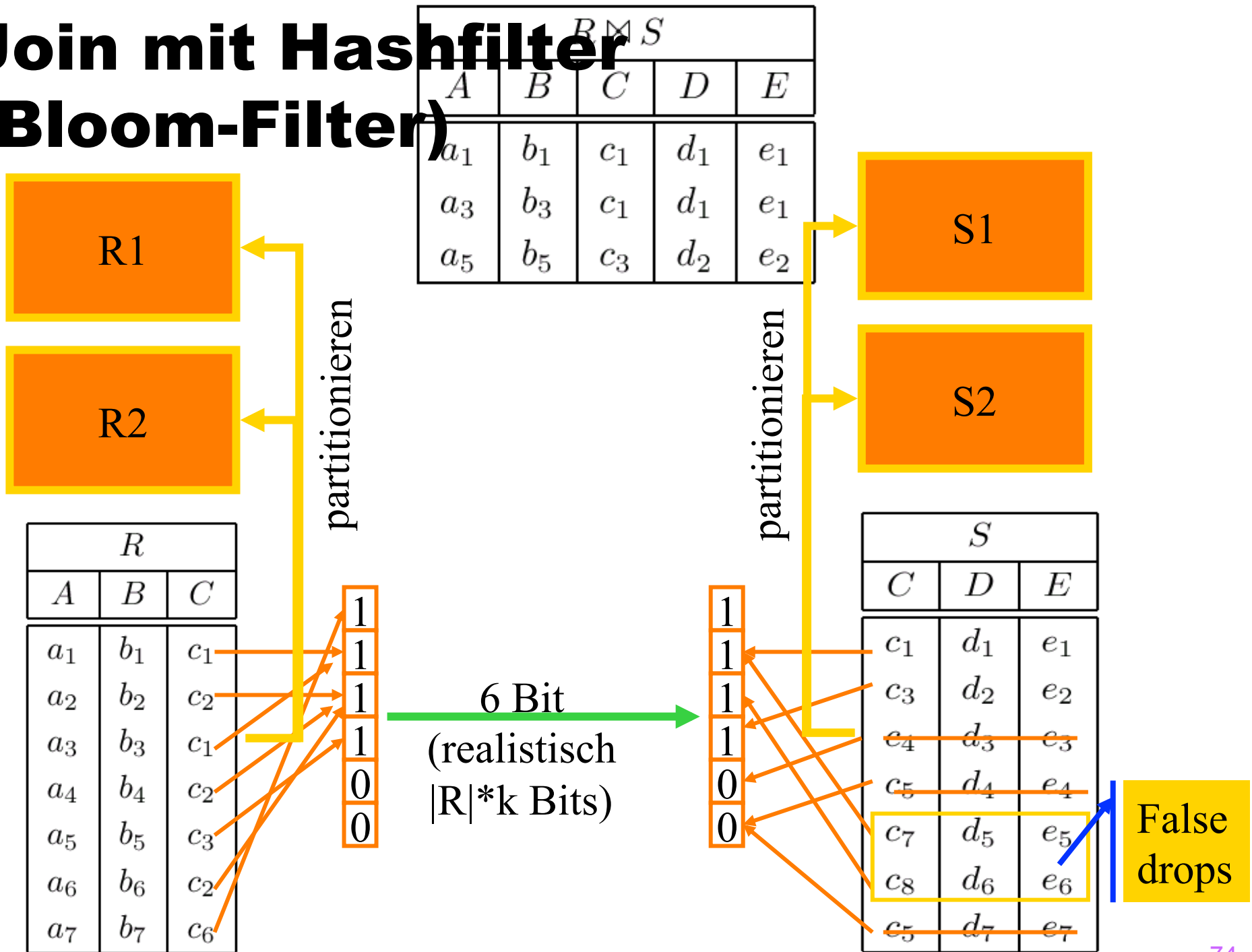
Vergleich: Sort/Merge-Join versus Hash-Join



Prallelausführung von Aggregat-Operationen

- **Min:** $\text{Min}(R.A) = \text{Min} (\text{Min}(R1.A), \dots , \text{Min}(Rn.A))$
- **Max:** analog
- **Sum:** $\text{Sum}(R.A) = \text{Sum} (\text{Sum}(R1.a), \dots, \text{Sum}(Rn.A))$
- **Count:** analog
- **Avg:** man muß die Summe und die Kardinalitäten der Teilrelationen kennen; aber vorsicht bei Null-Werten!
- $\text{Avg}(R.A) = \text{Sum}(R.A) / \text{Count}(R)$ 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 $r \in R$ das Bit setzt: $1/b$
 - W. dass kein $r \in R$ das Bit setzt: $(1-1/b)^{|R|}$
 - W. dass ein $r \in R$ das Bit gesetzt hat: $1 - (1-1/b)^{|R|}$

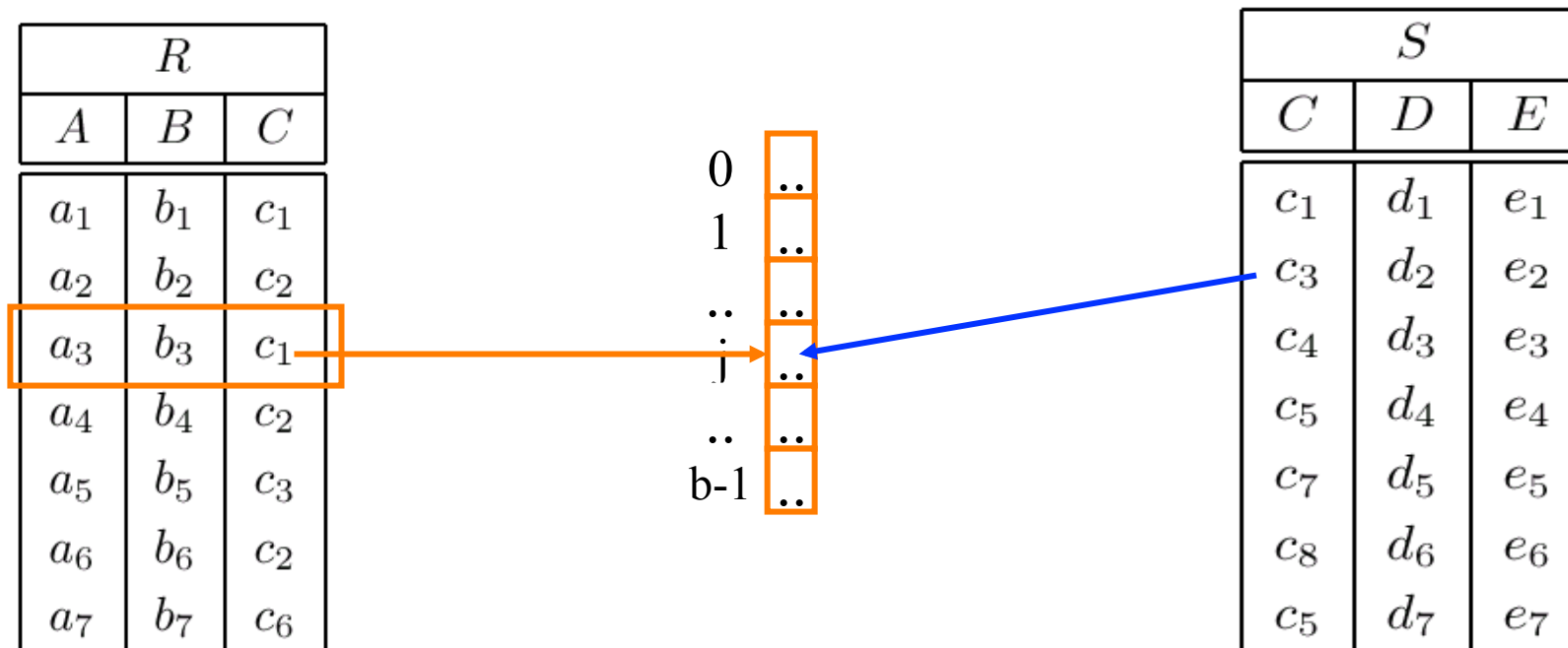


Illustration: Externes Sortieren

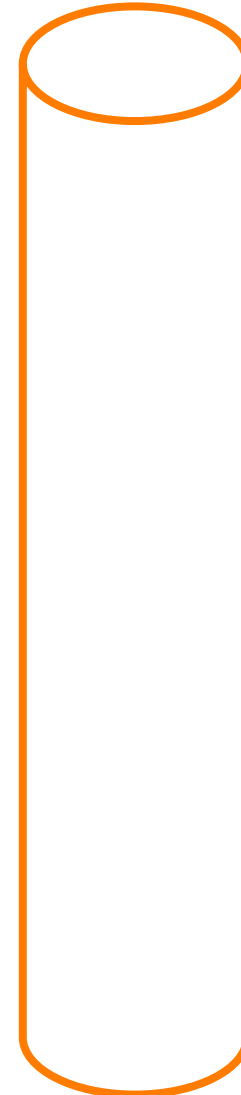
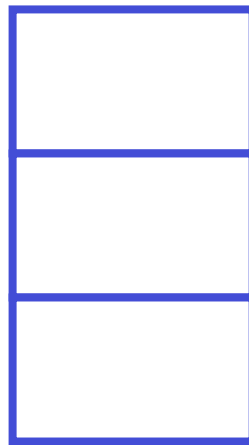
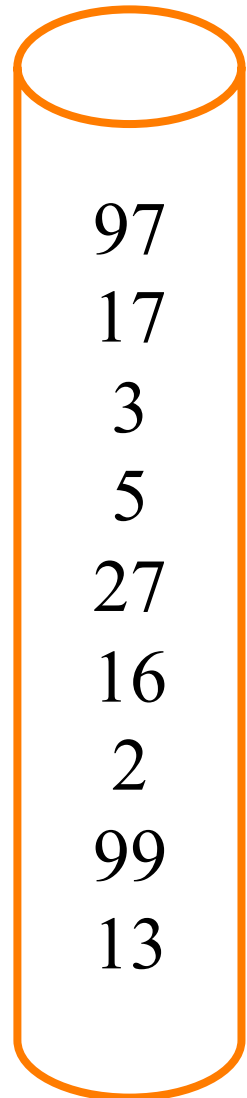


Illustration: Externes Sortieren

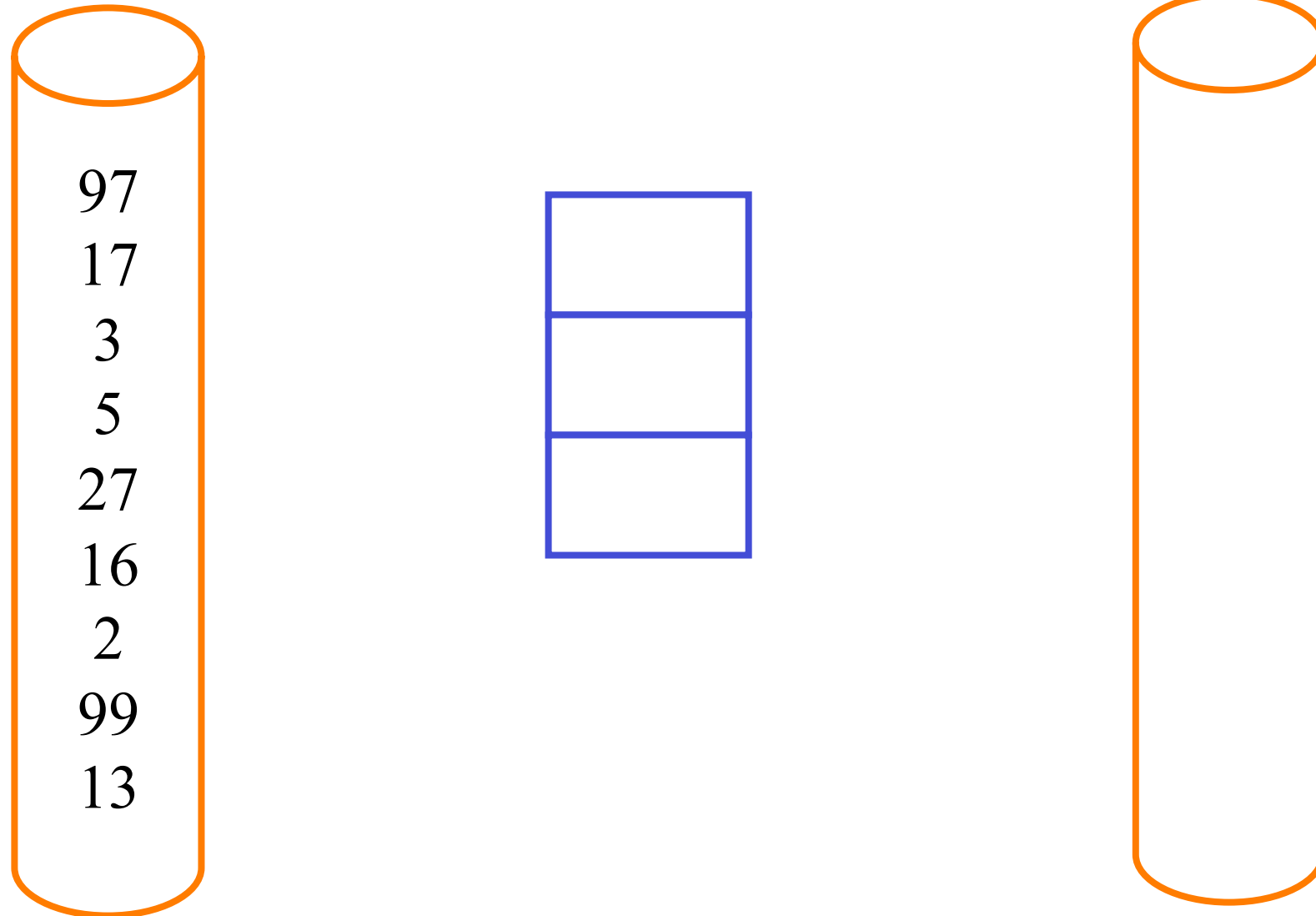


Illustration: Externes Sortieren

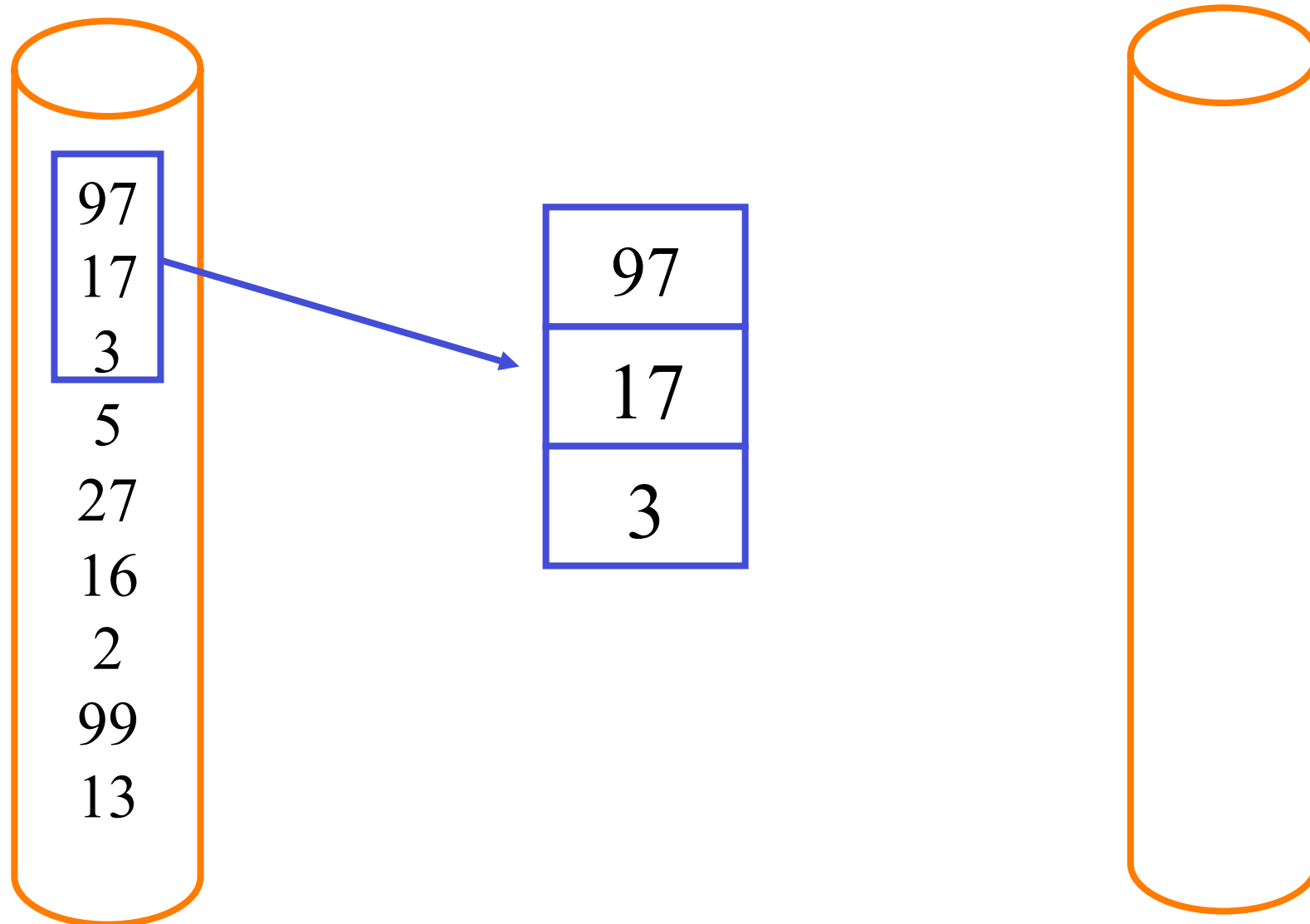
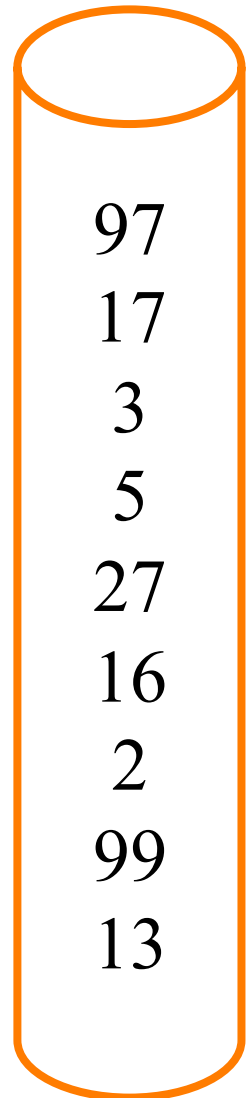


Illustration: Externes Sortieren



sort

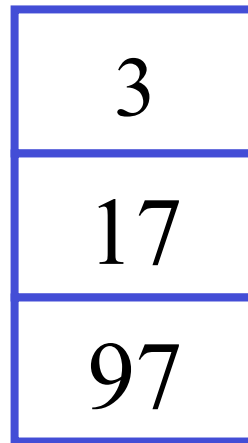


Illustration: Externes Sortieren

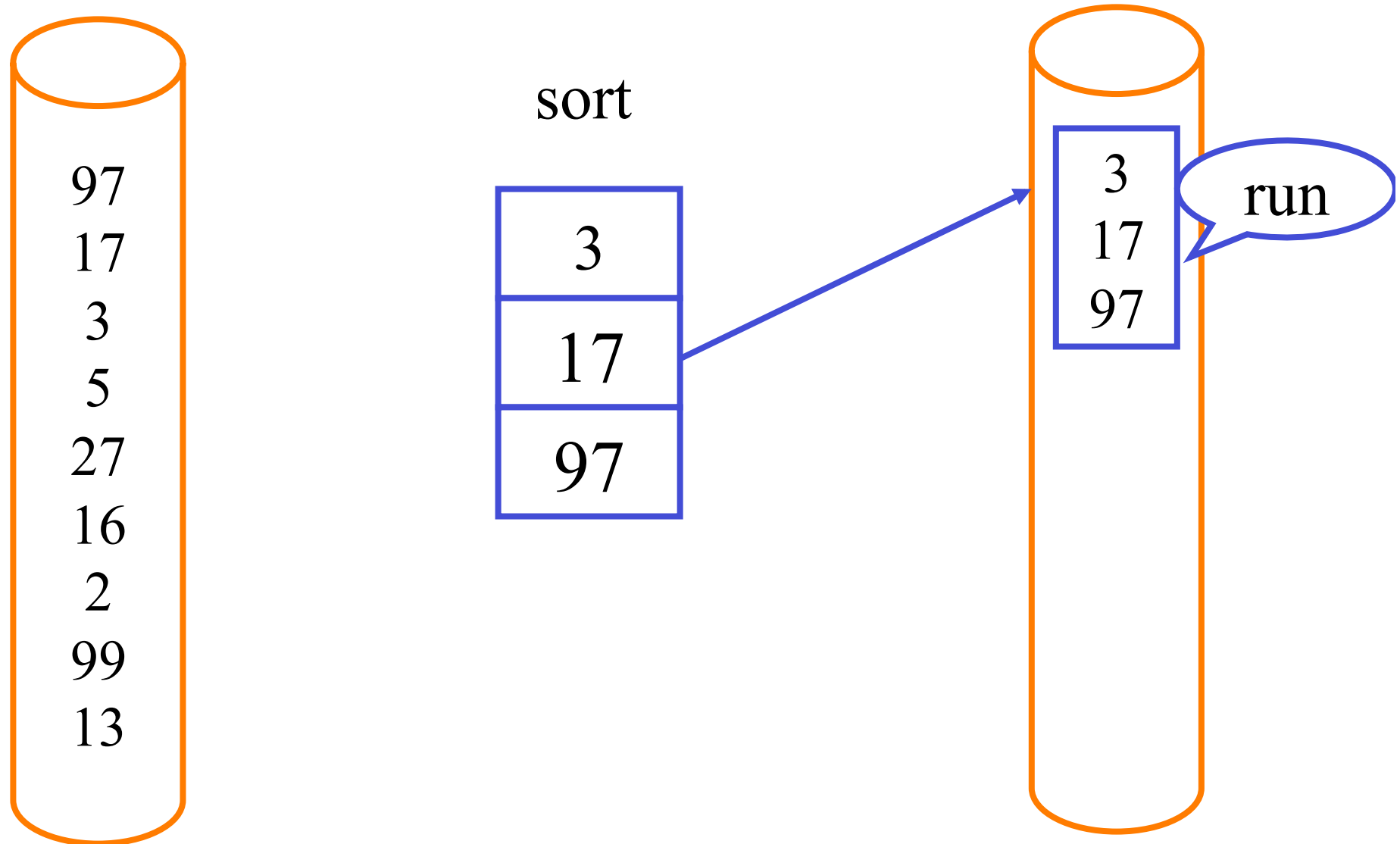


Illustration: Externes Sortieren

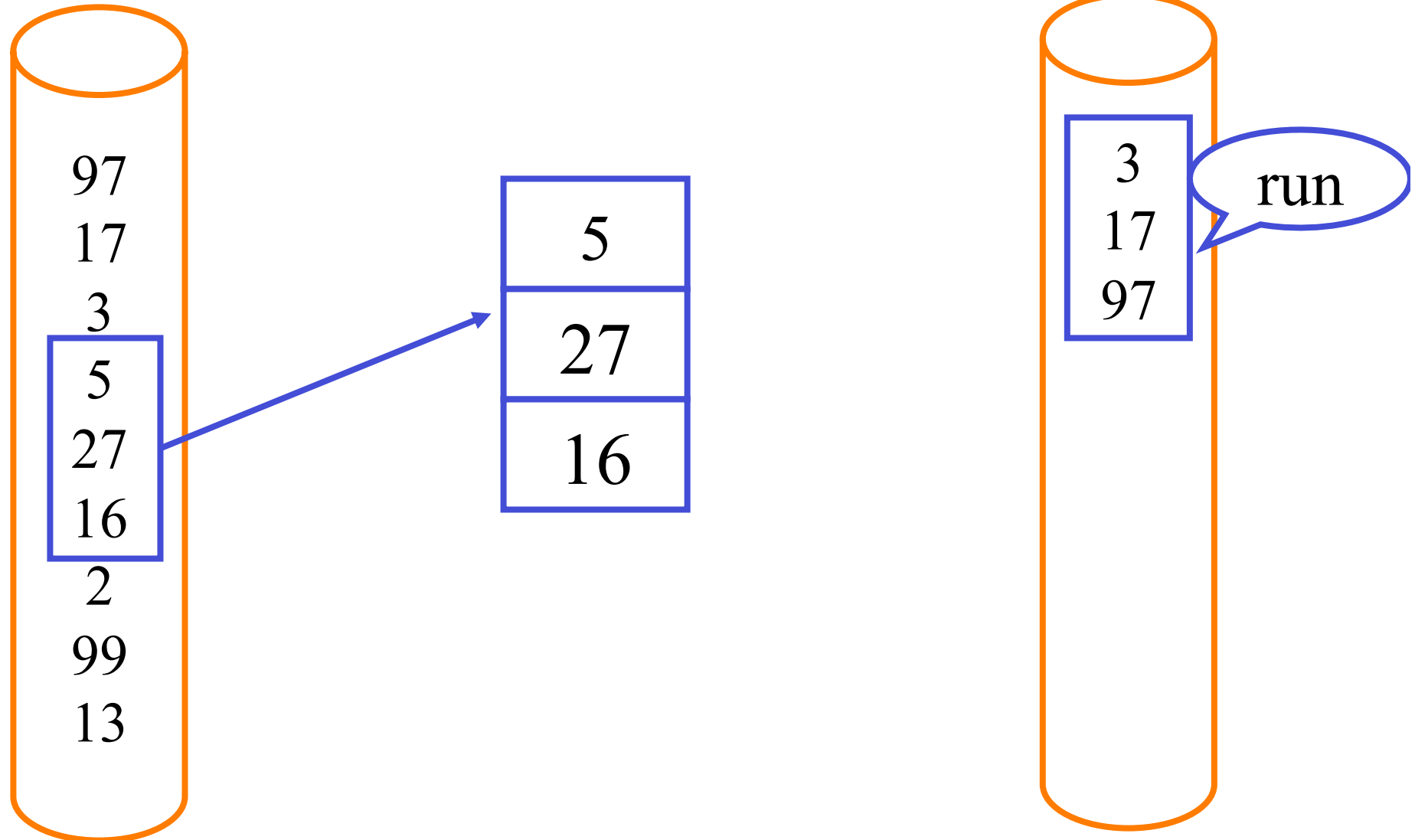


Illustration: Externes Sortieren

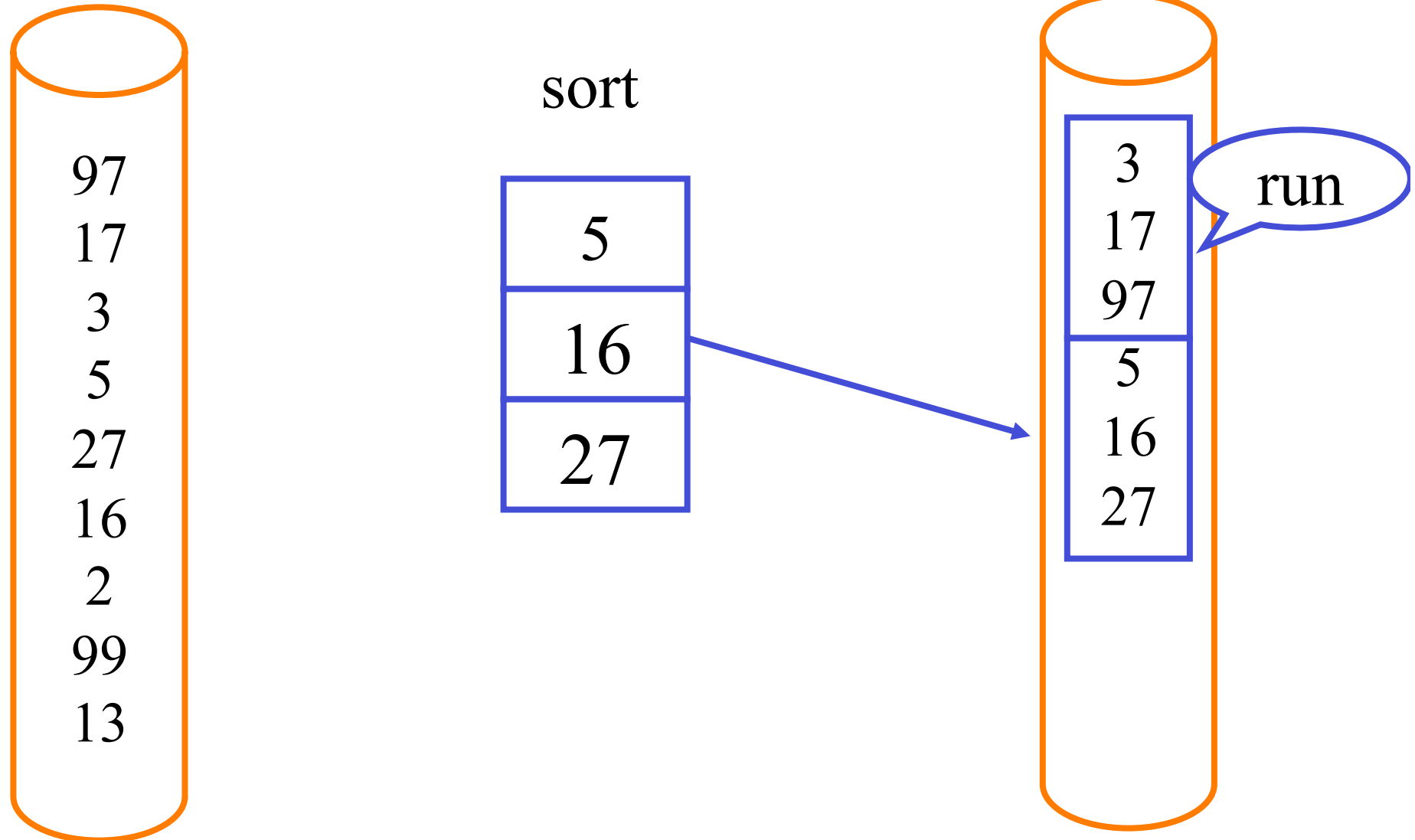


Illustration: Externes Sortieren

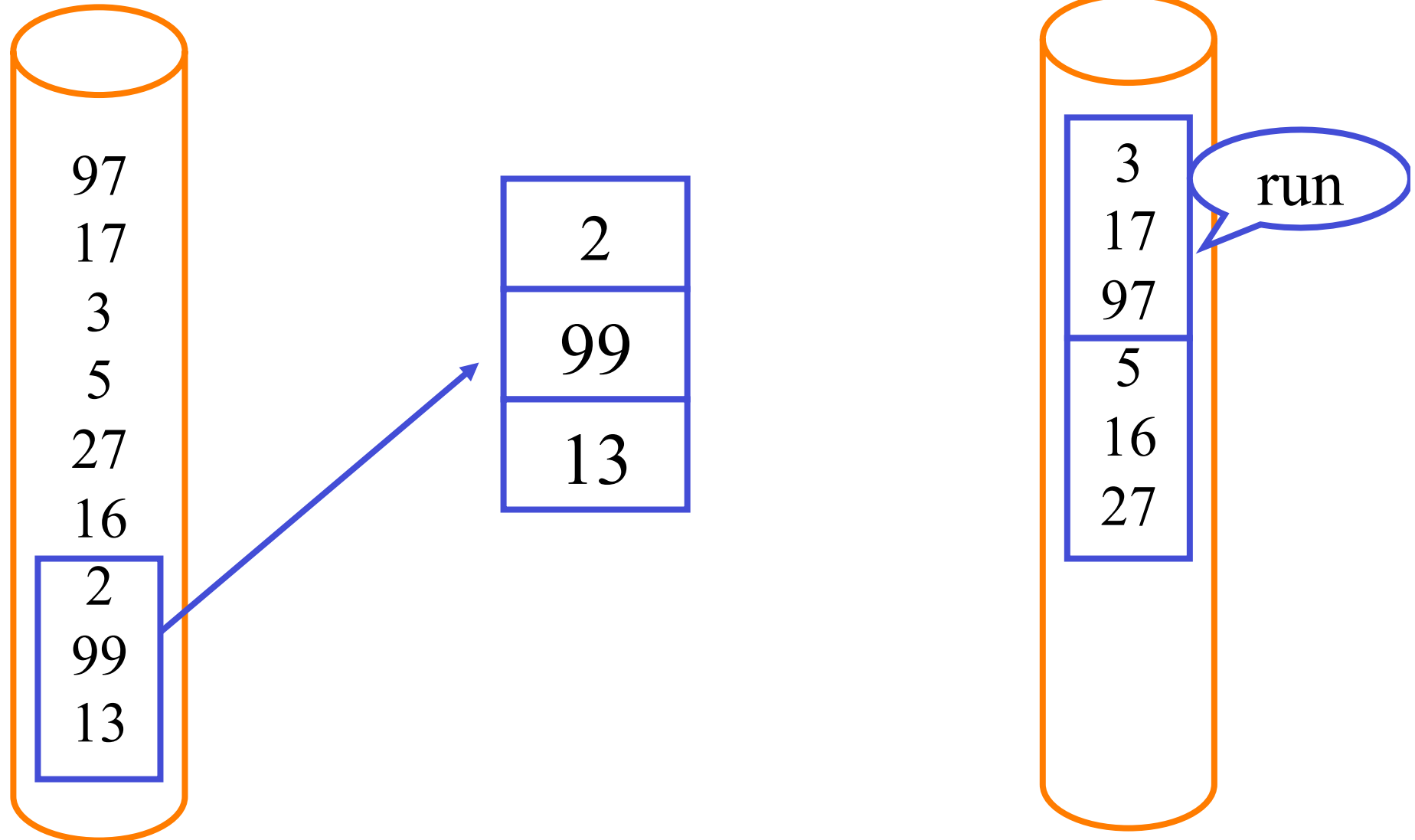


Illustration: Externes Sortieren

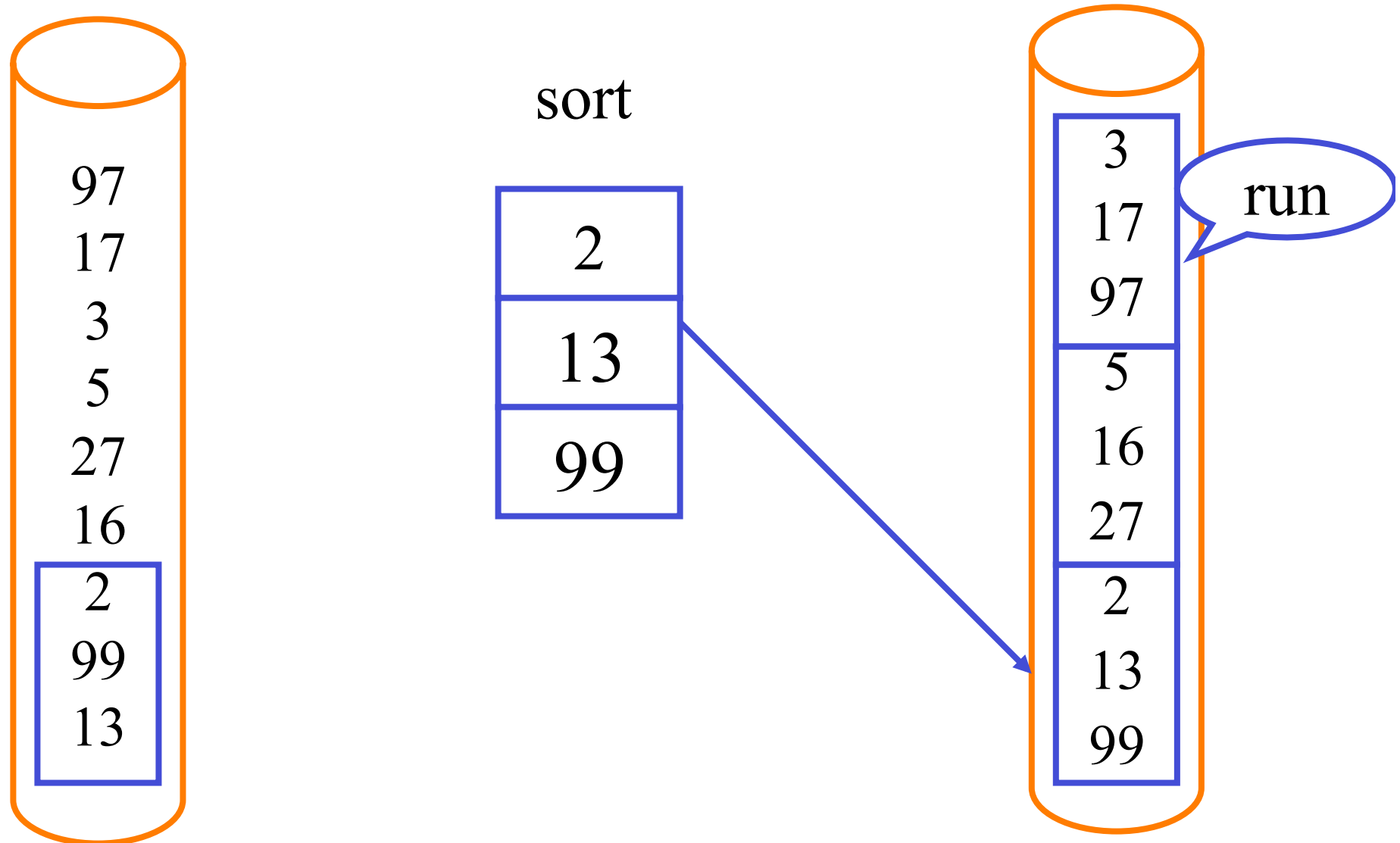
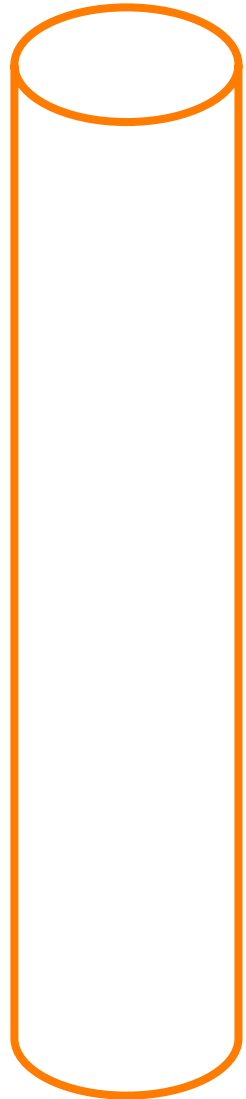
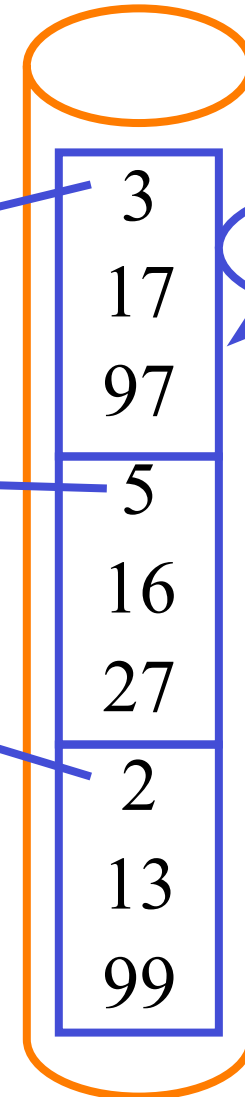
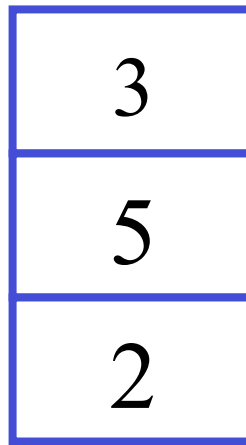


Illustration: Externes Sortieren



merge



run

Illustration: Externes Sortieren

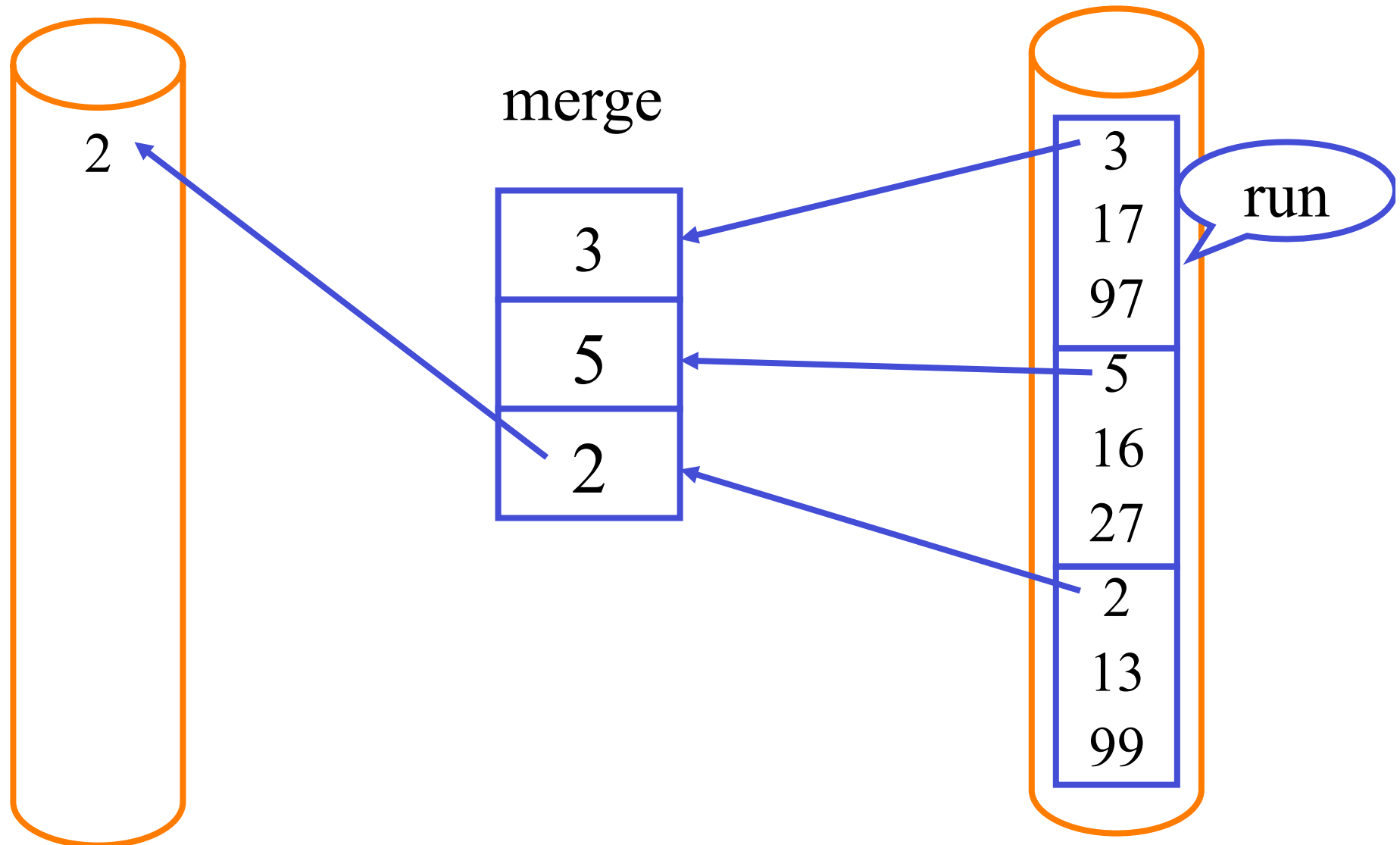


Illustration: Externes Sortieren

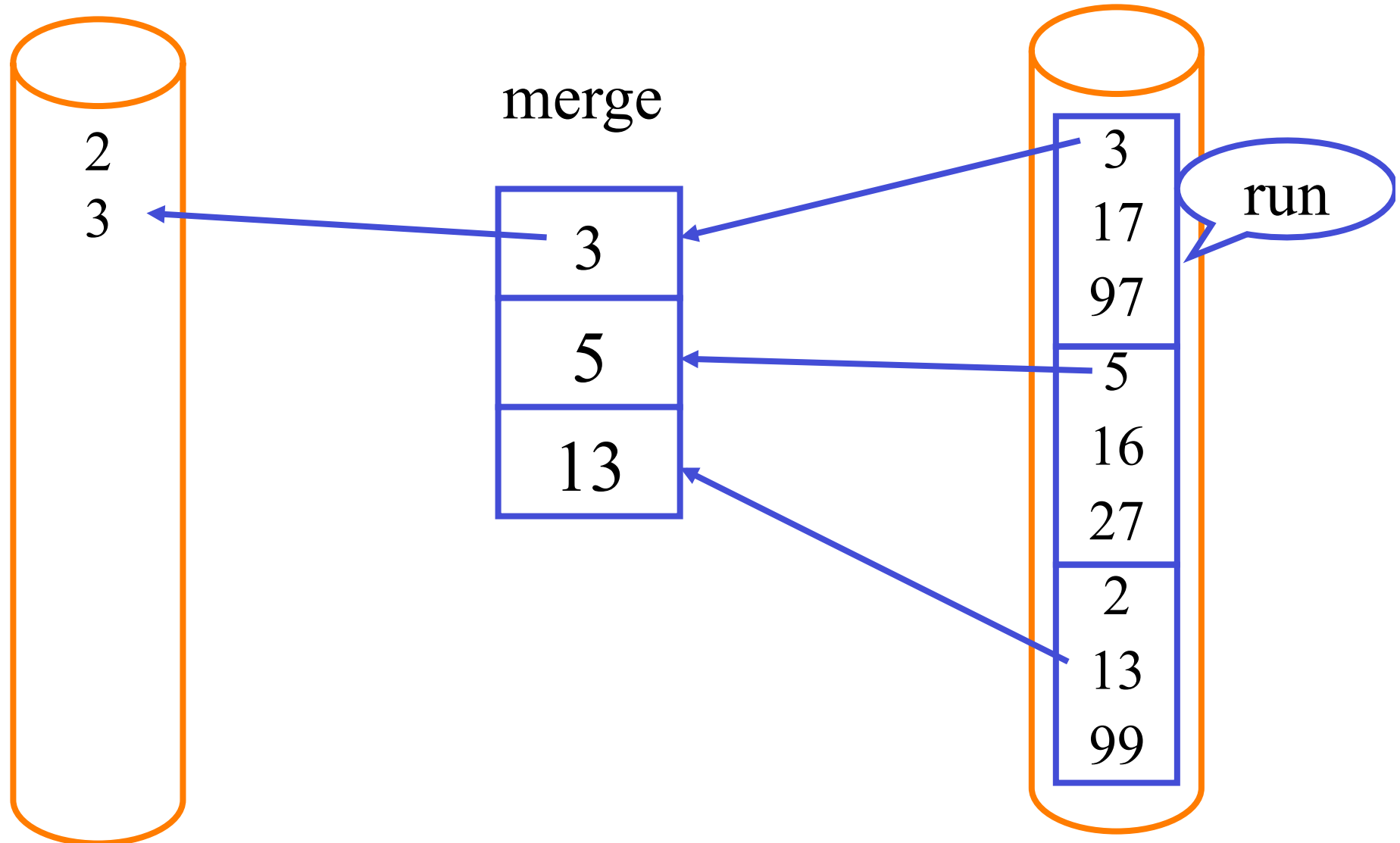


Illustration: Externes Sortieren

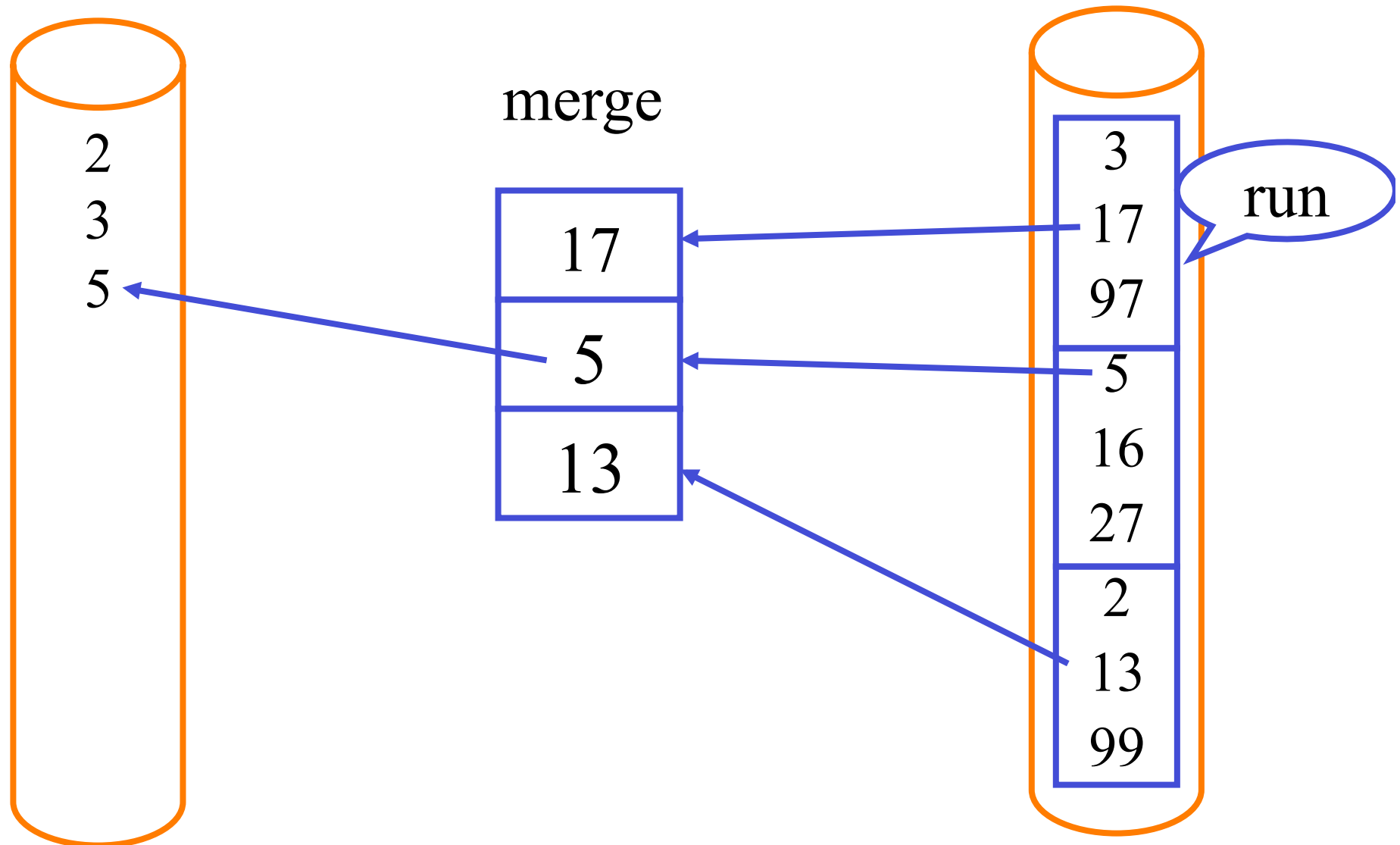


Illustration: Externes Sortieren

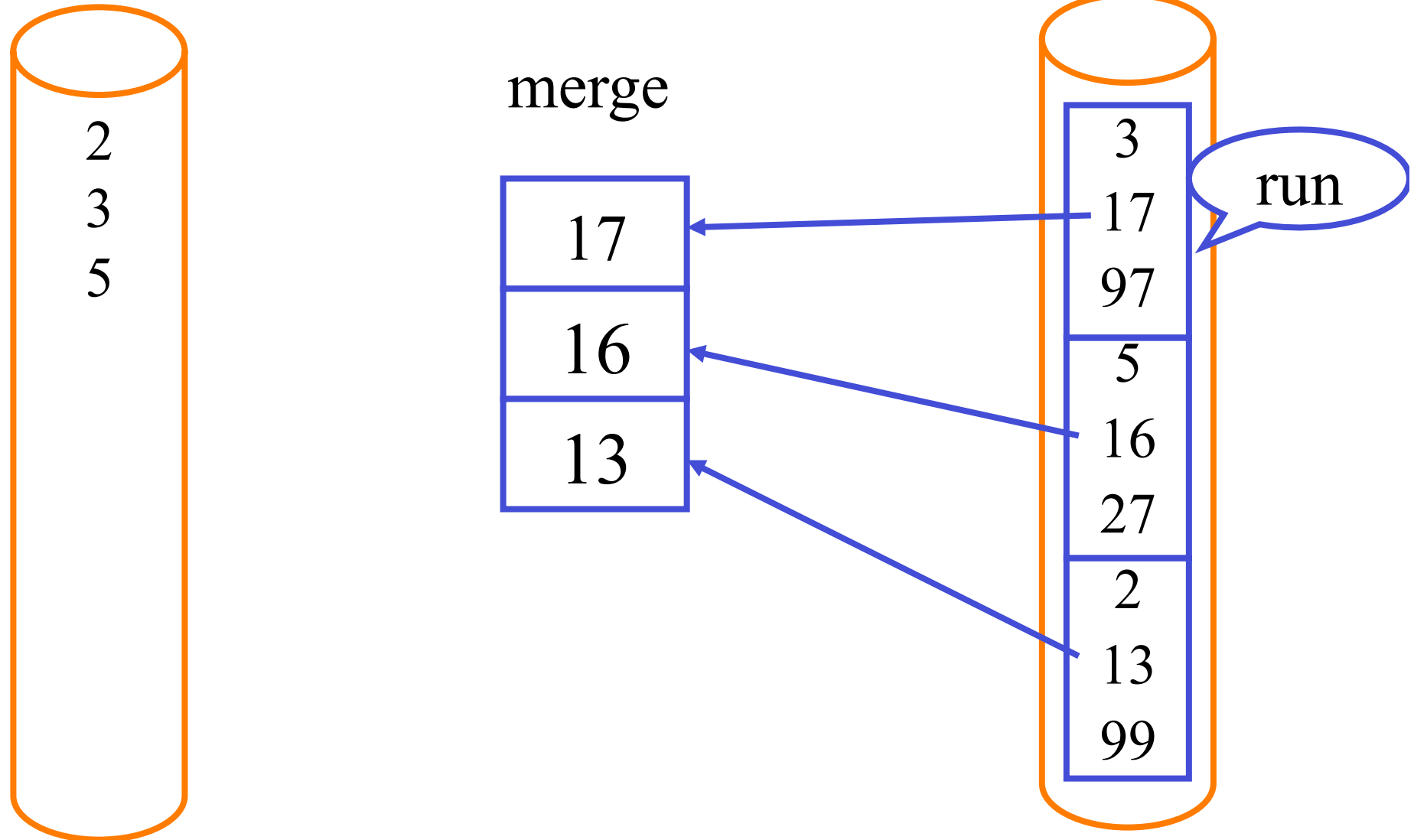
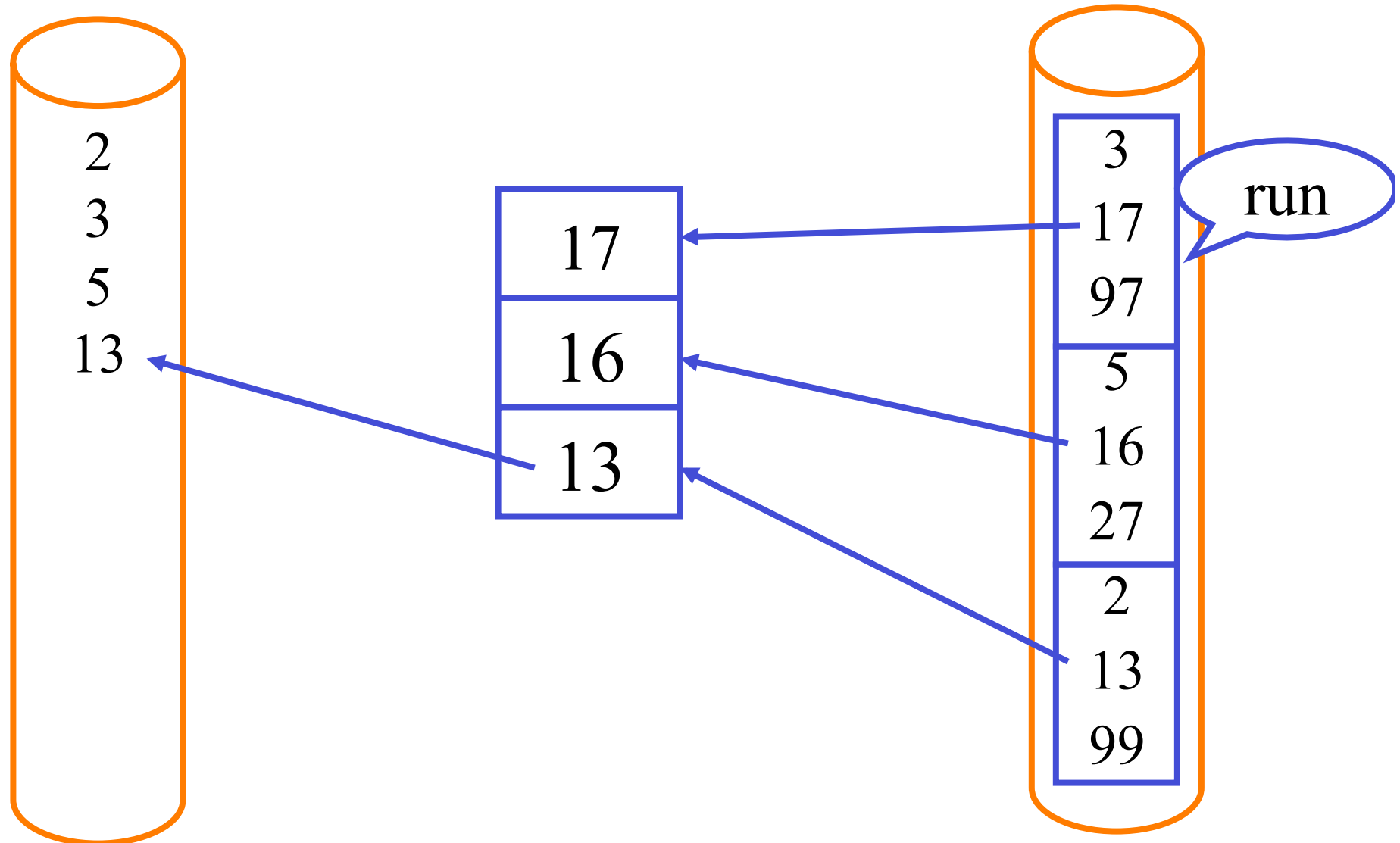
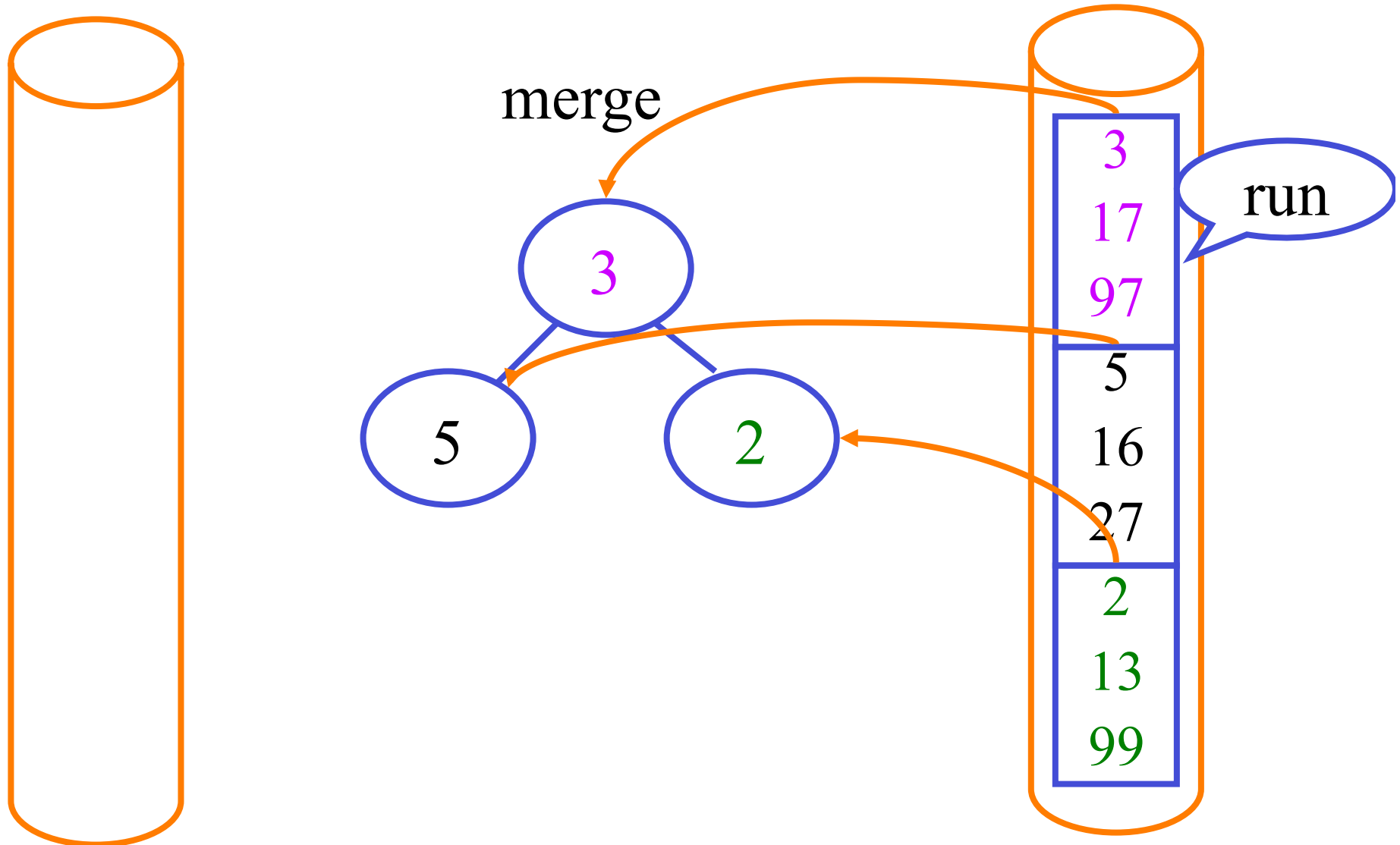


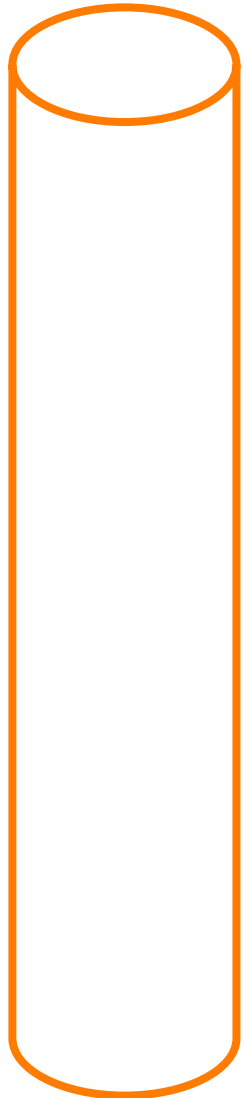
Illustration: Externes Sortieren



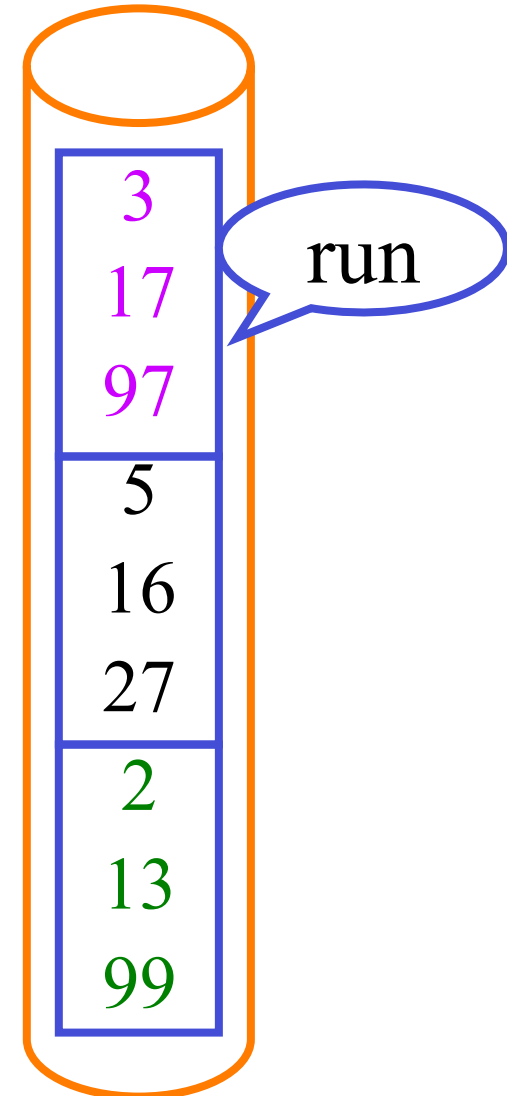
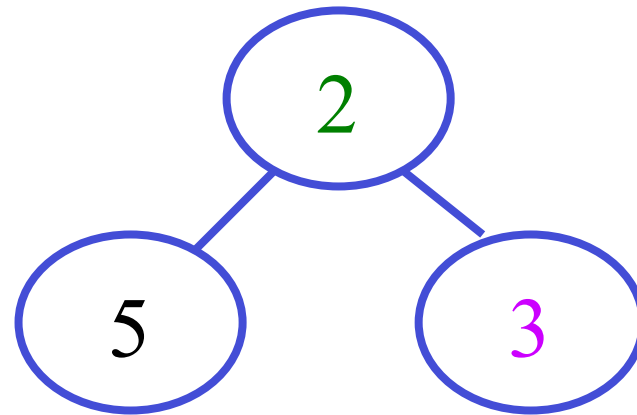
Externes Sortieren: Merge mittels Heap/Priority Queue



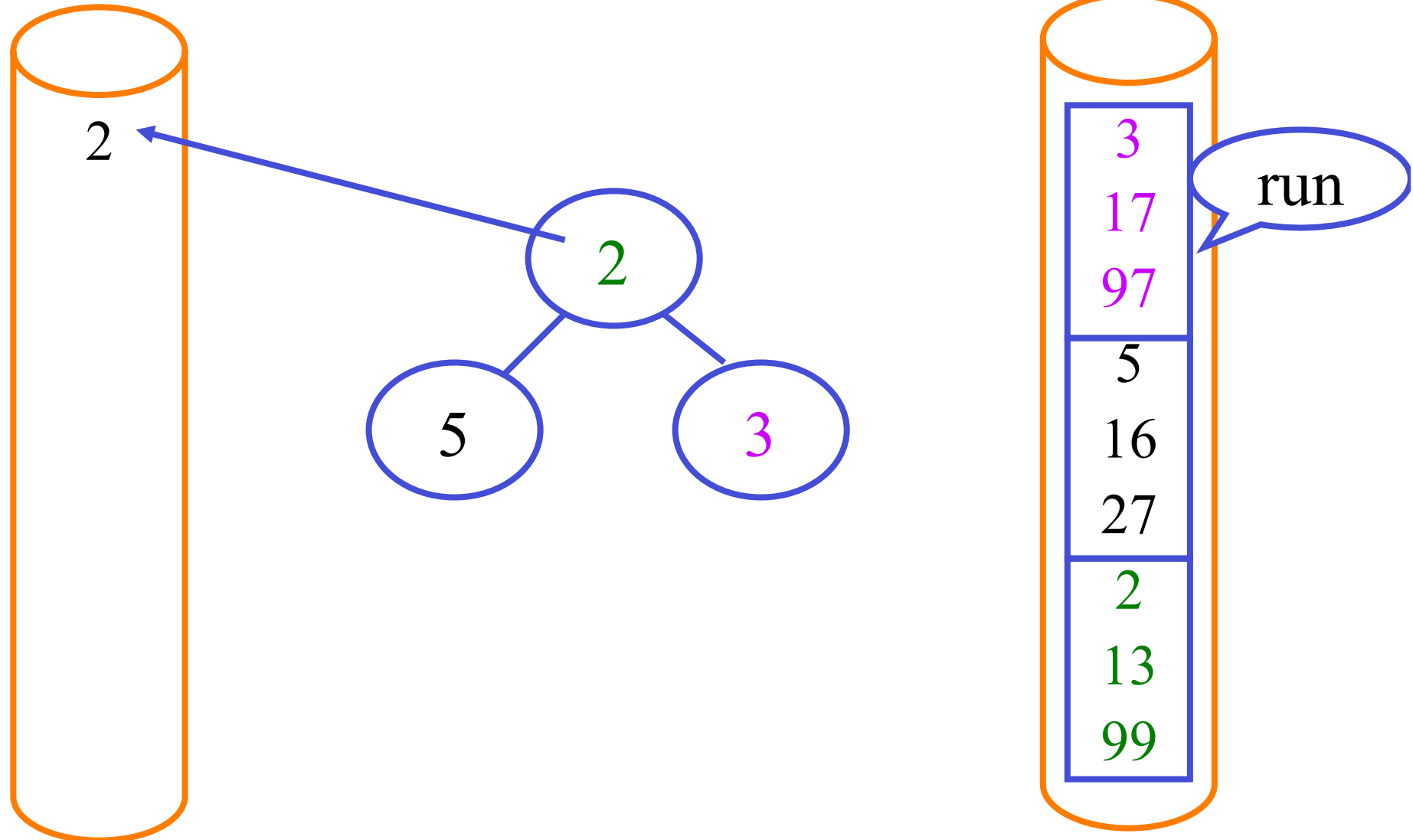
Externes Sortieren: Merge mittels Heap/Priority Queue



merge

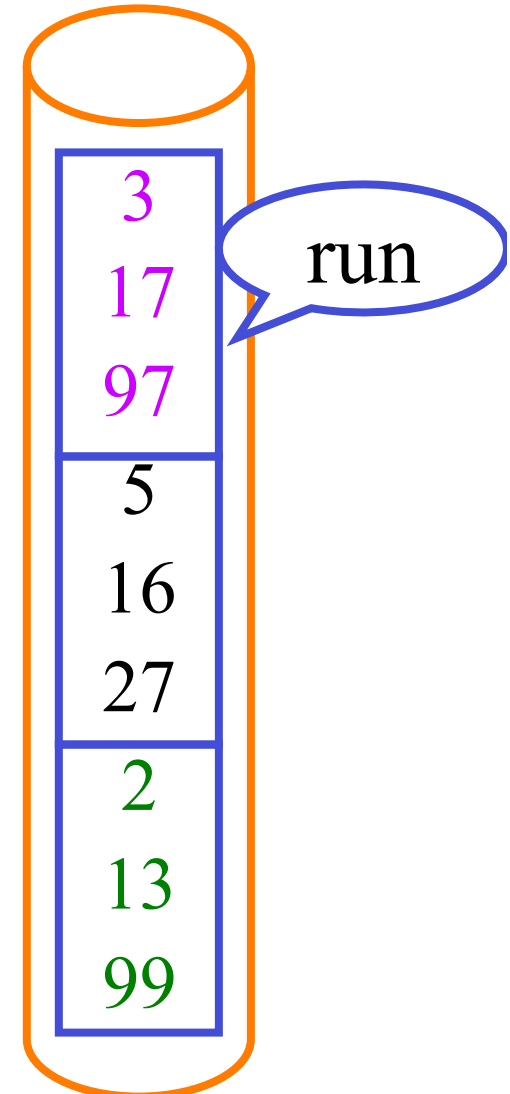
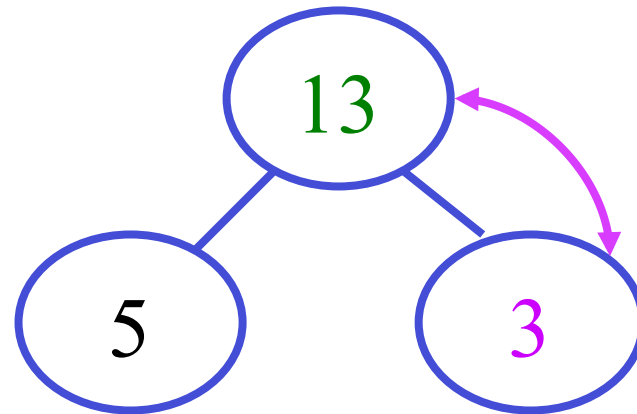
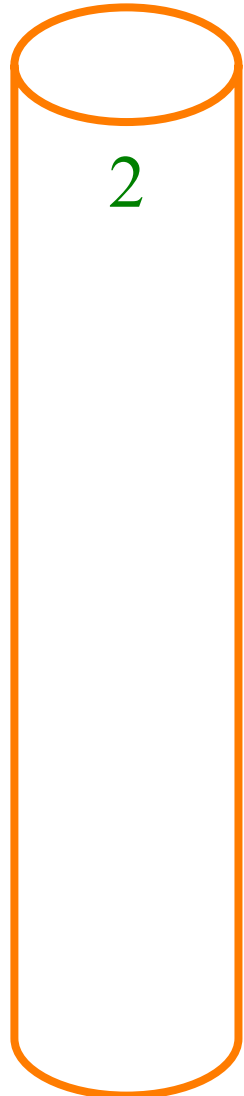


Externes Sortieren: Merge mittels Heap/Priority Queue

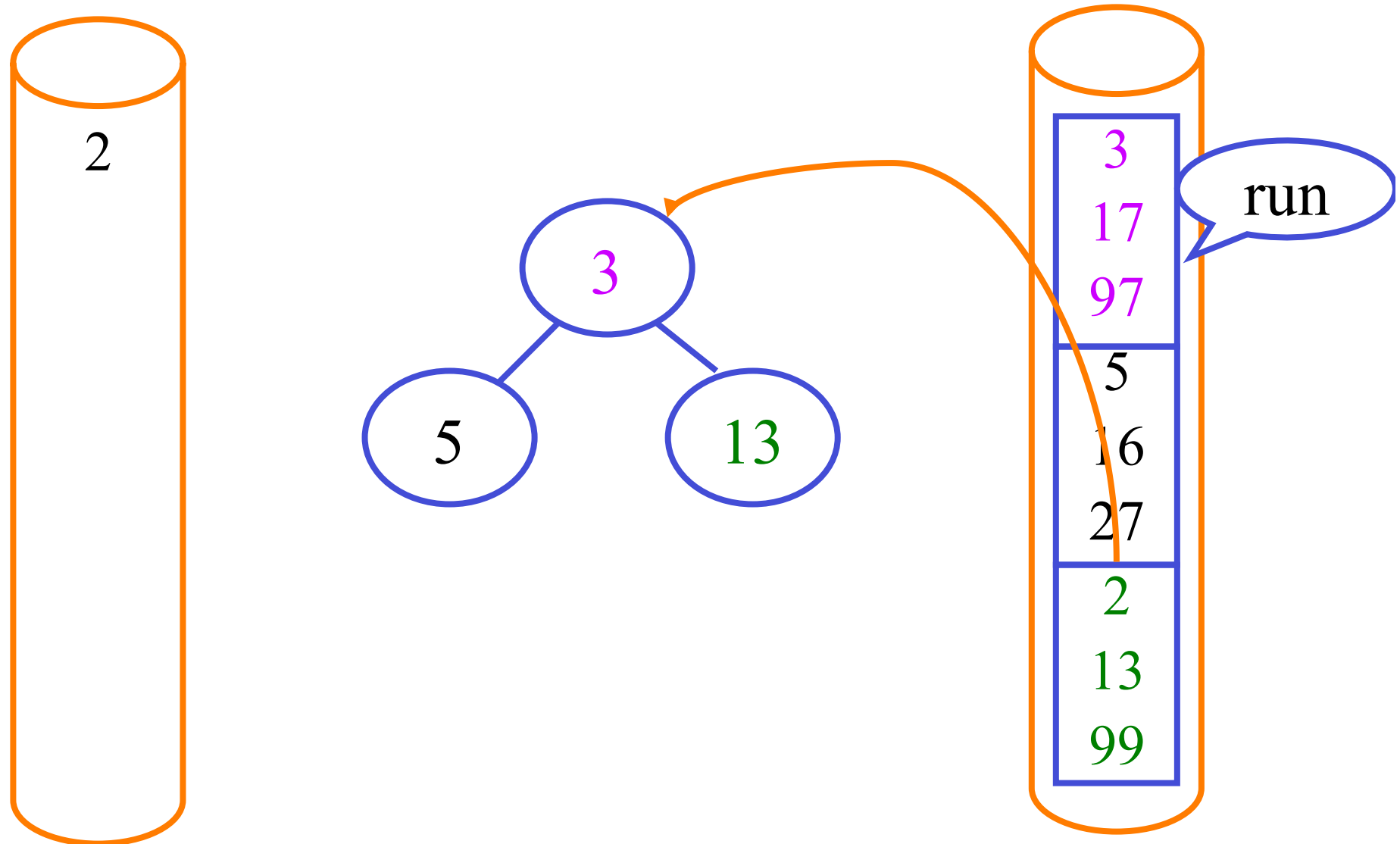


Externes Sortieren: Merge mittels Heap/Priority Queue

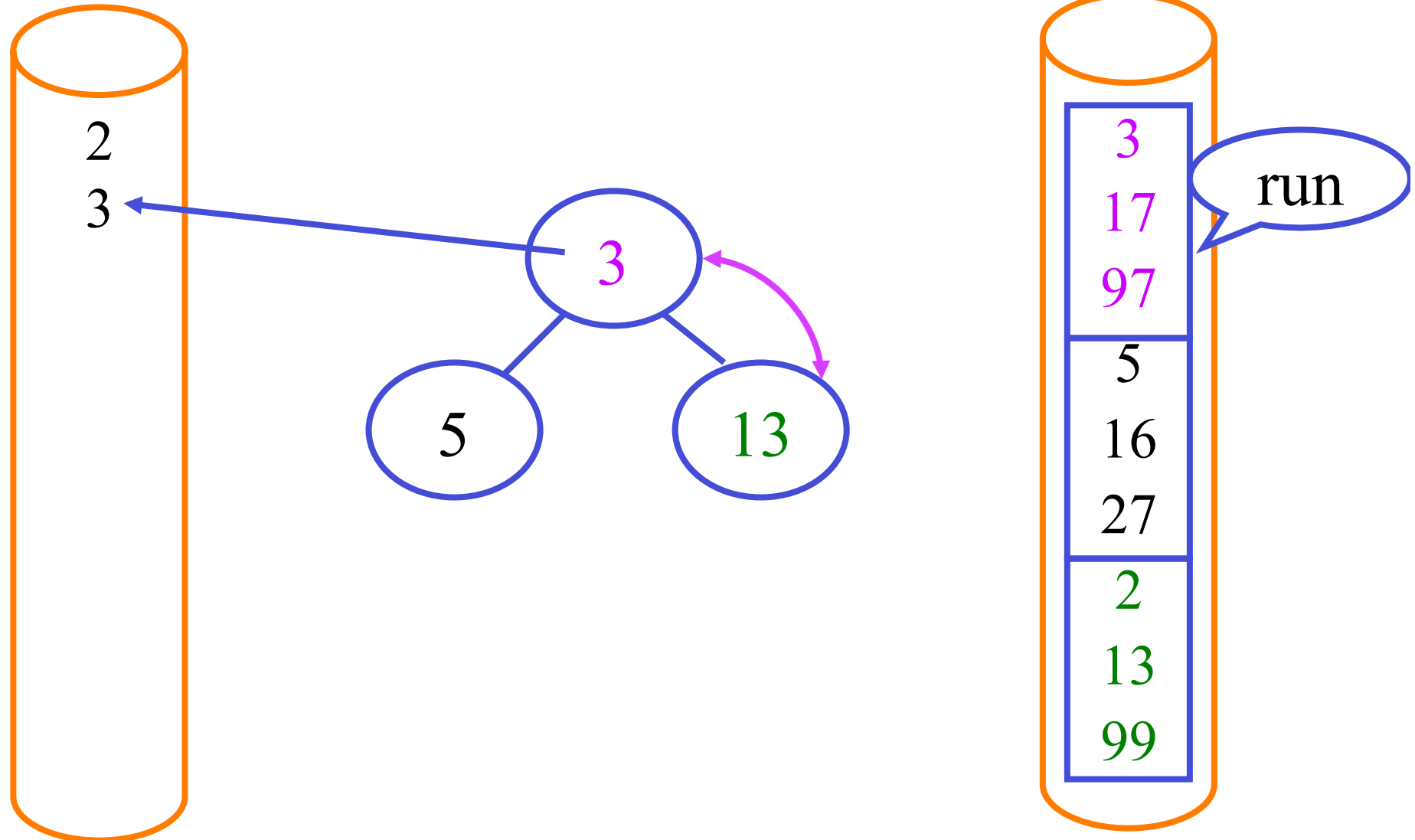
Ganz wichtig: aus dem **grünen Run** nachladen (also aus dem Run, aus dem das Objekt stammte)



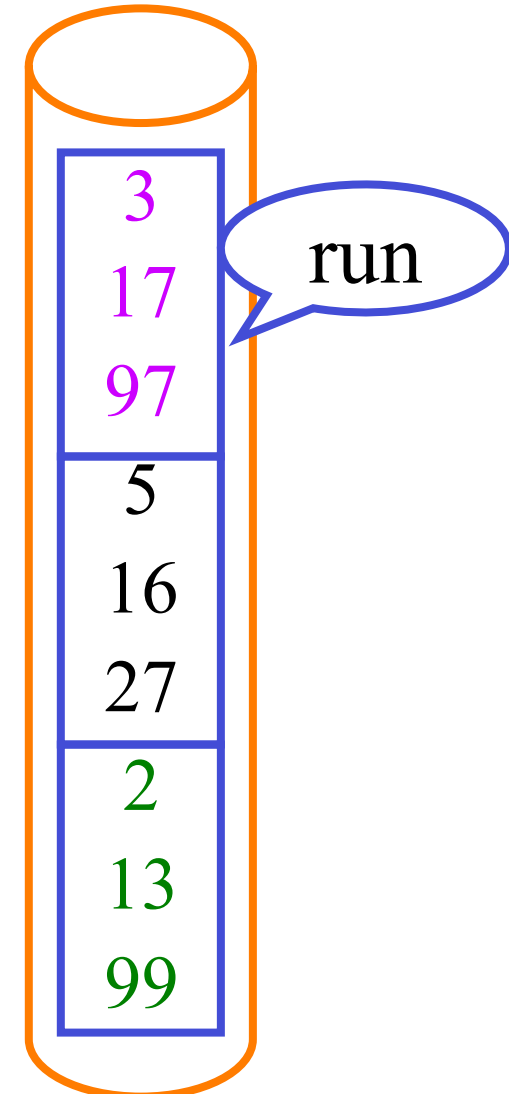
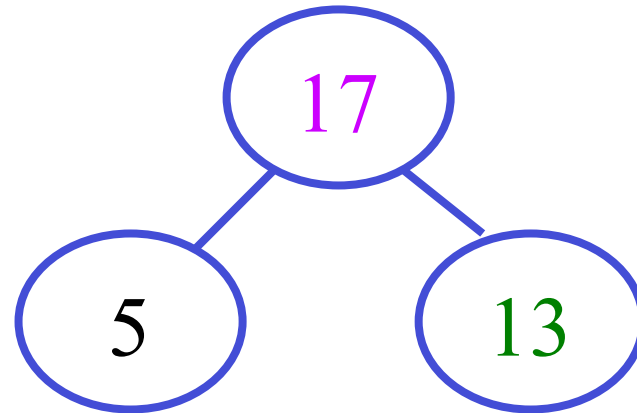
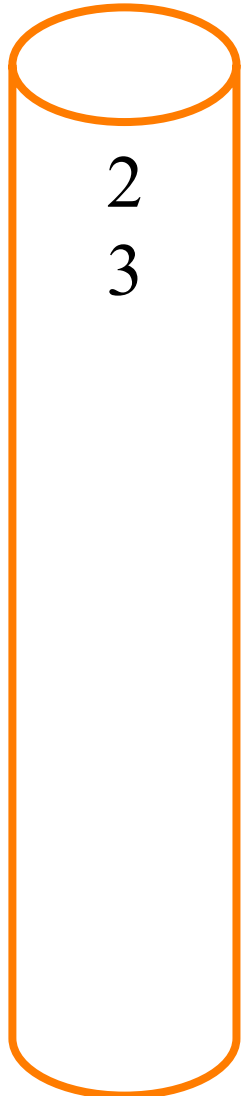
Externes Sortieren: Merge mittels Heap/Priority Queue



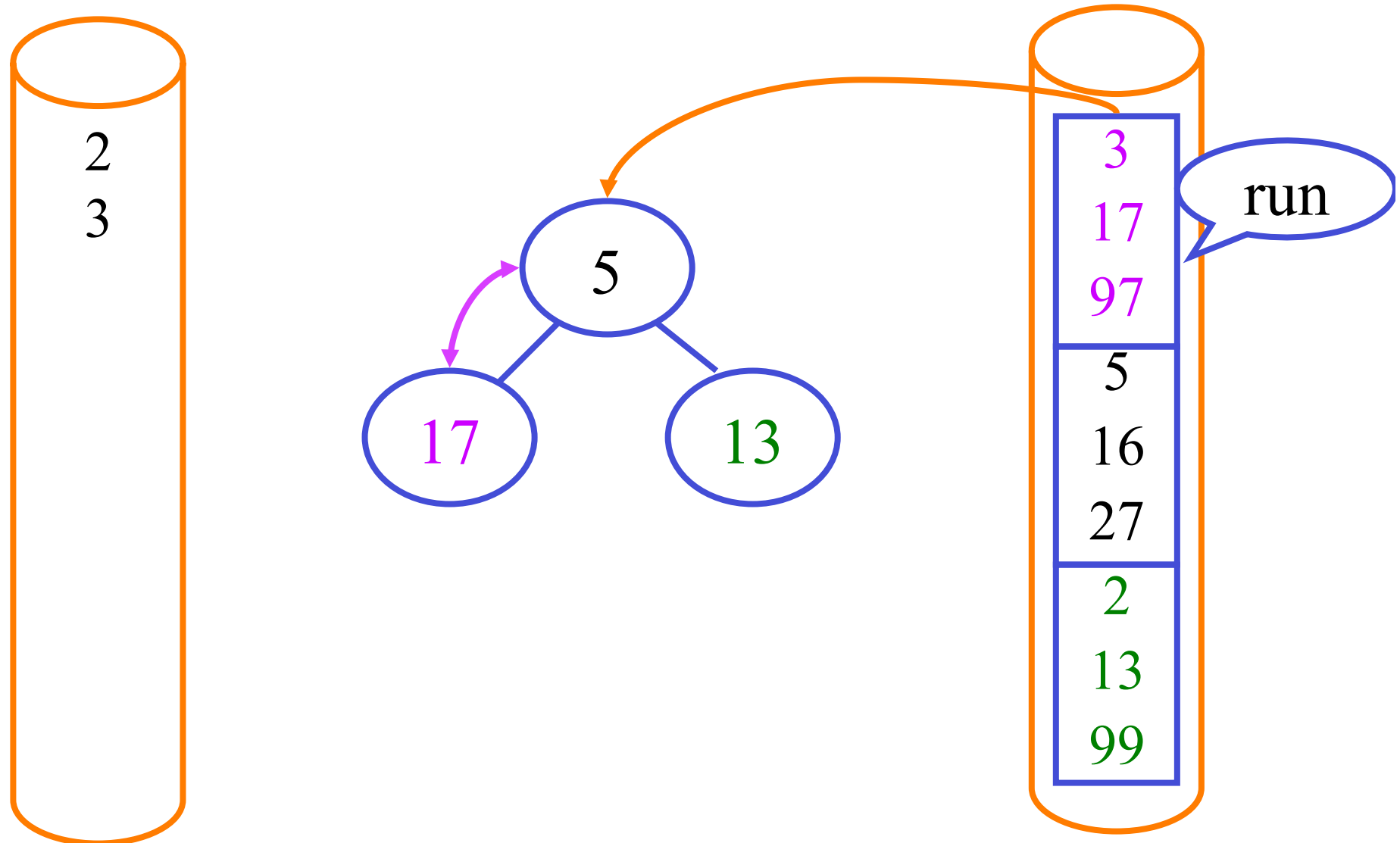
Externes Sortieren: Merge mittels Heap/Priority Queue



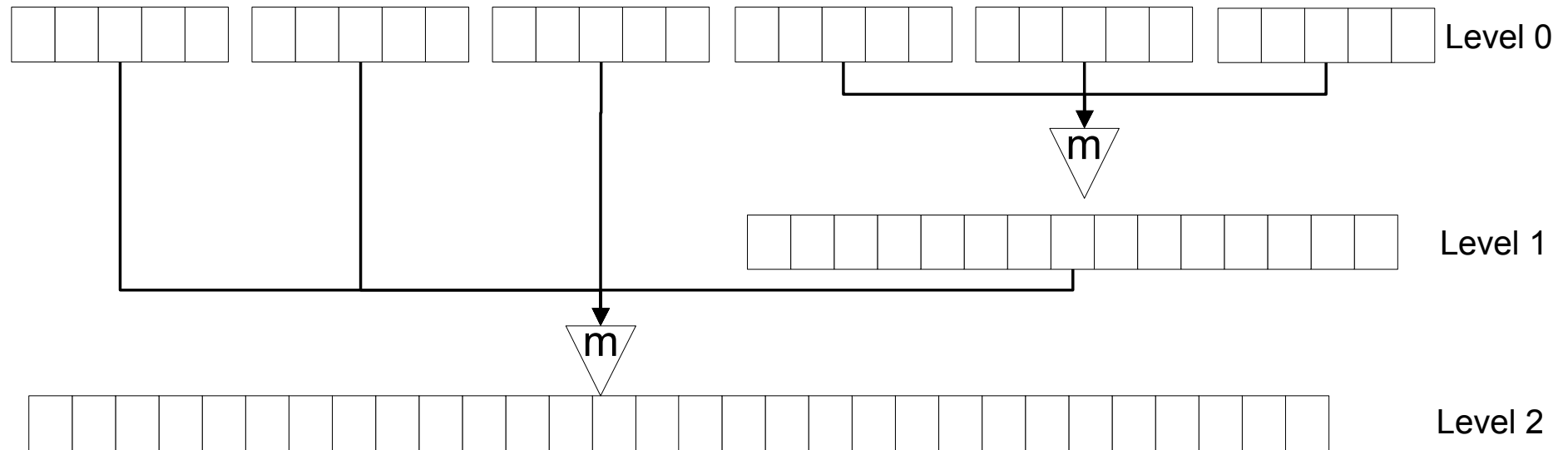
Externes Sortieren: Merge mittels Heap/Priority Queue



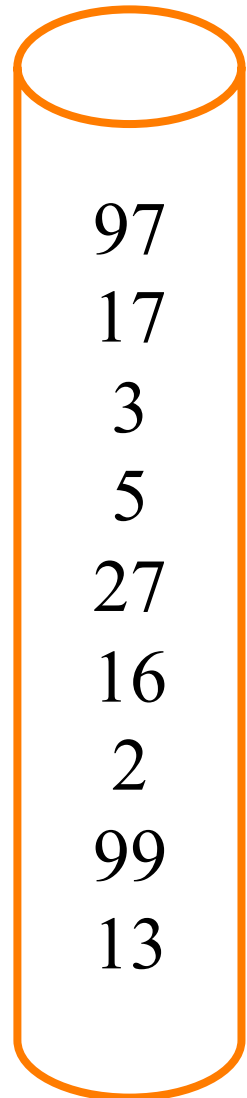
Externes Sortieren: Merge mittels Heap/Priority Queue



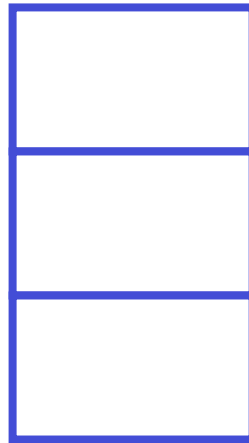
Mehrstufiges Mischen / Merge



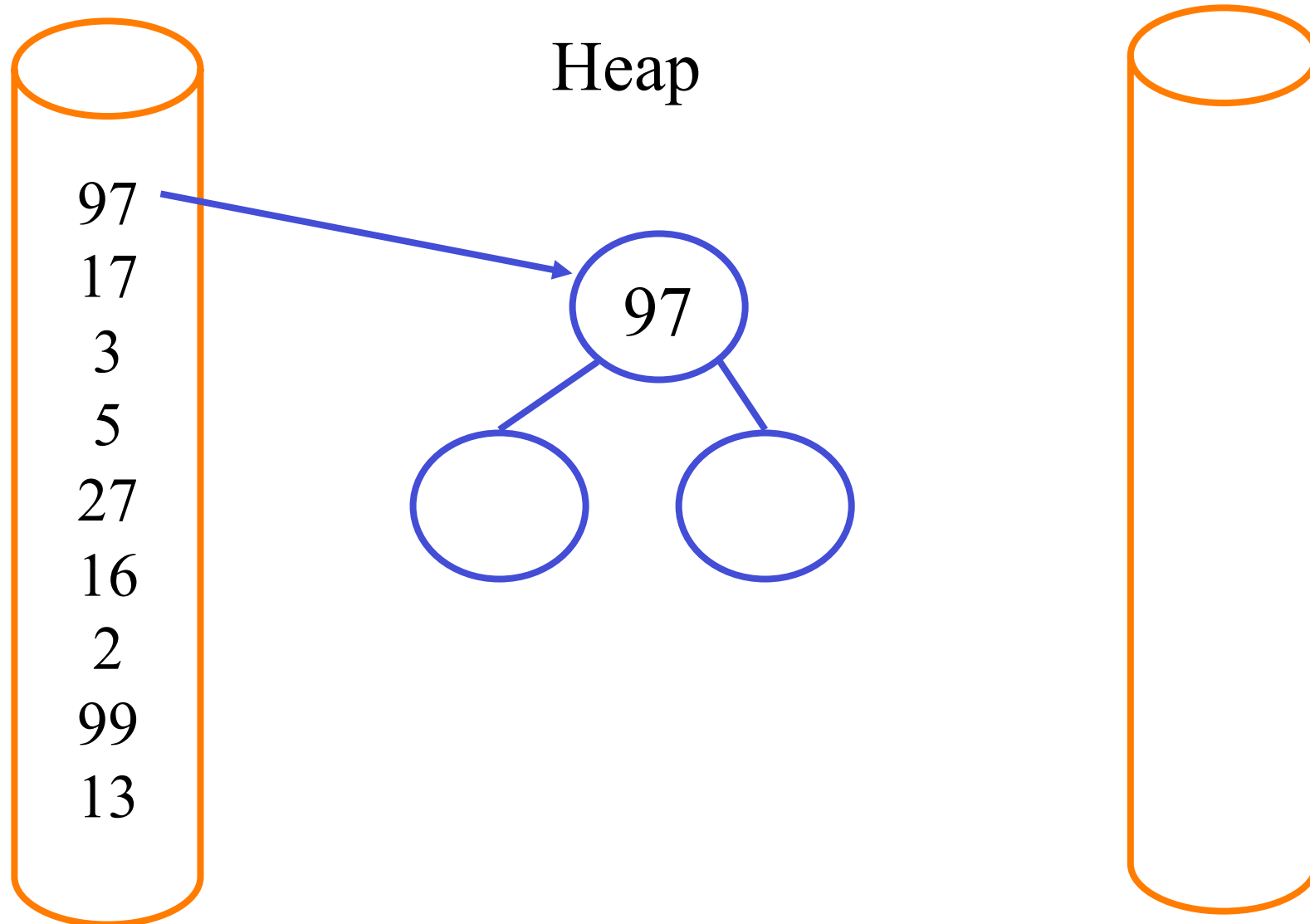
Replacement Selection während der Run-Generierung



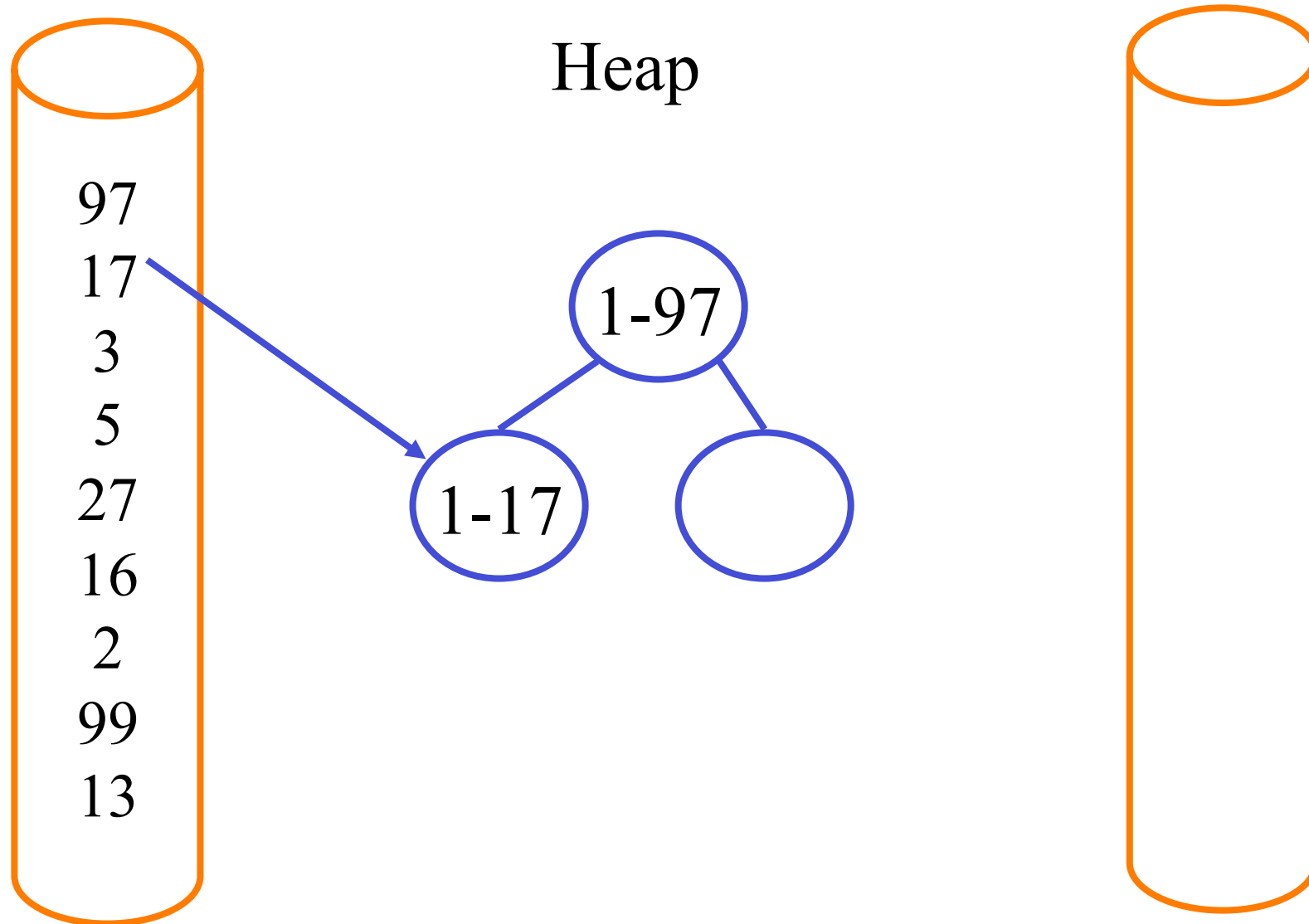
Ersetze Array durch
Einen Heap



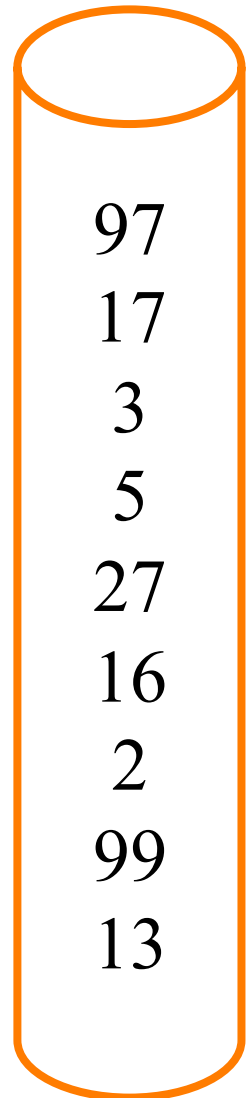
Replacement Selection während der Run-Generierung



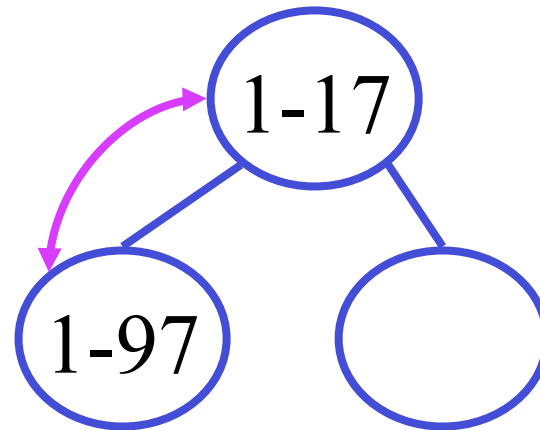
Replacement Selection während der Run-Generierung



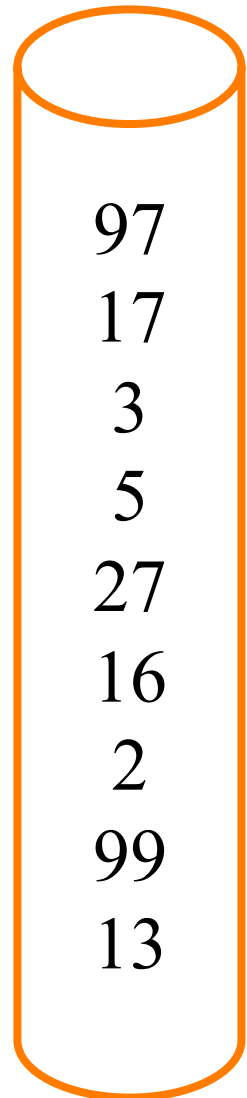
Replacement Selection während der Run-Generierung



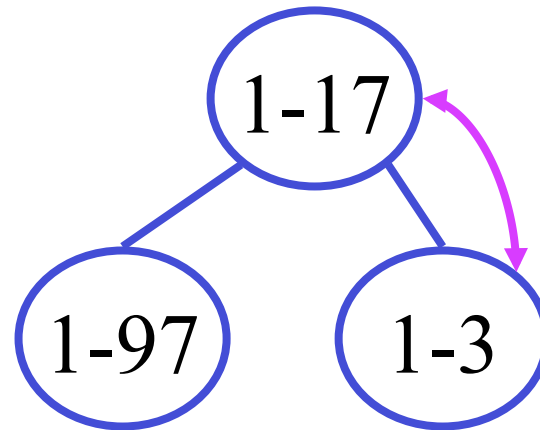
Heap



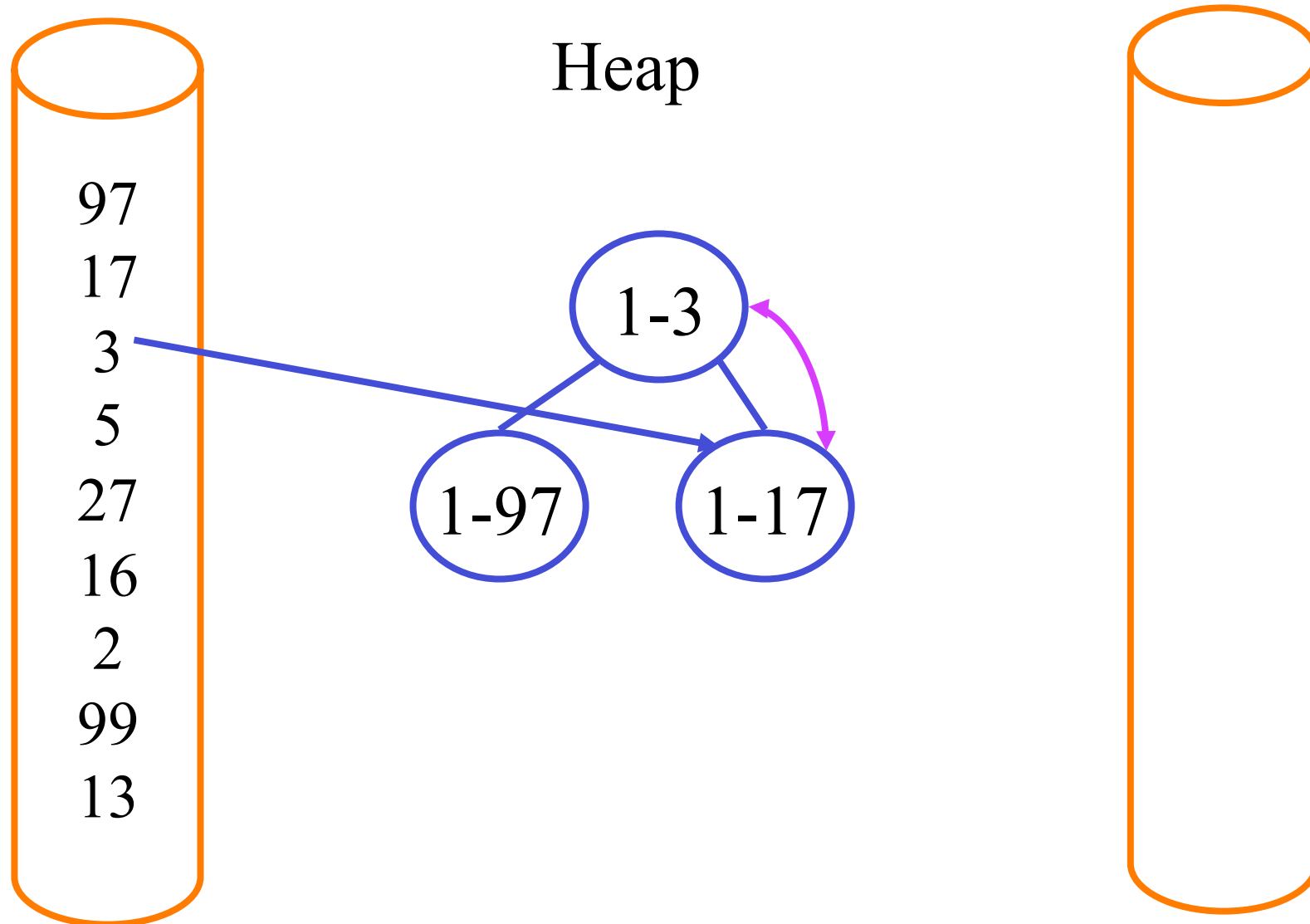
Replacement Selection während der Run-Generierung



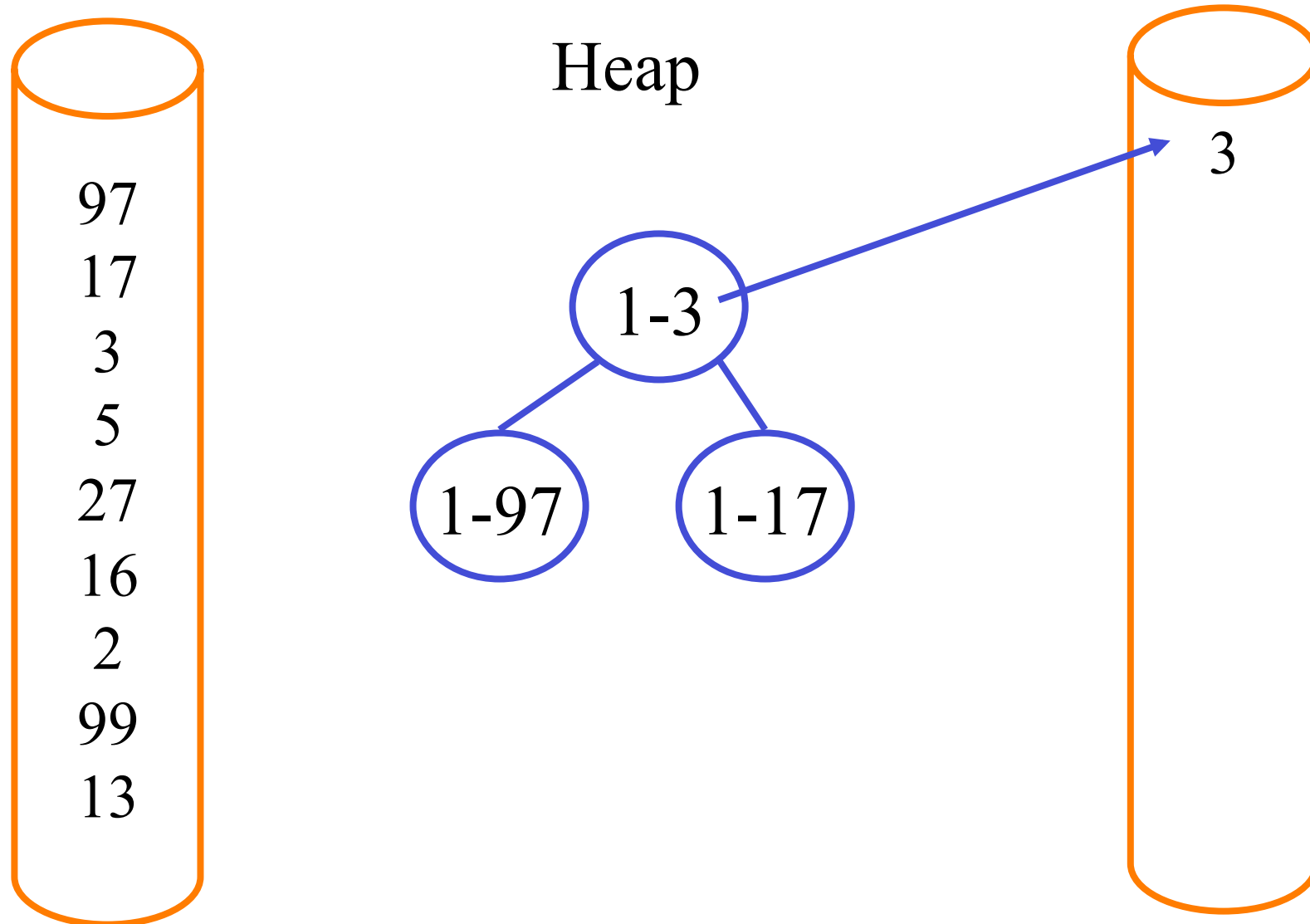
Heap



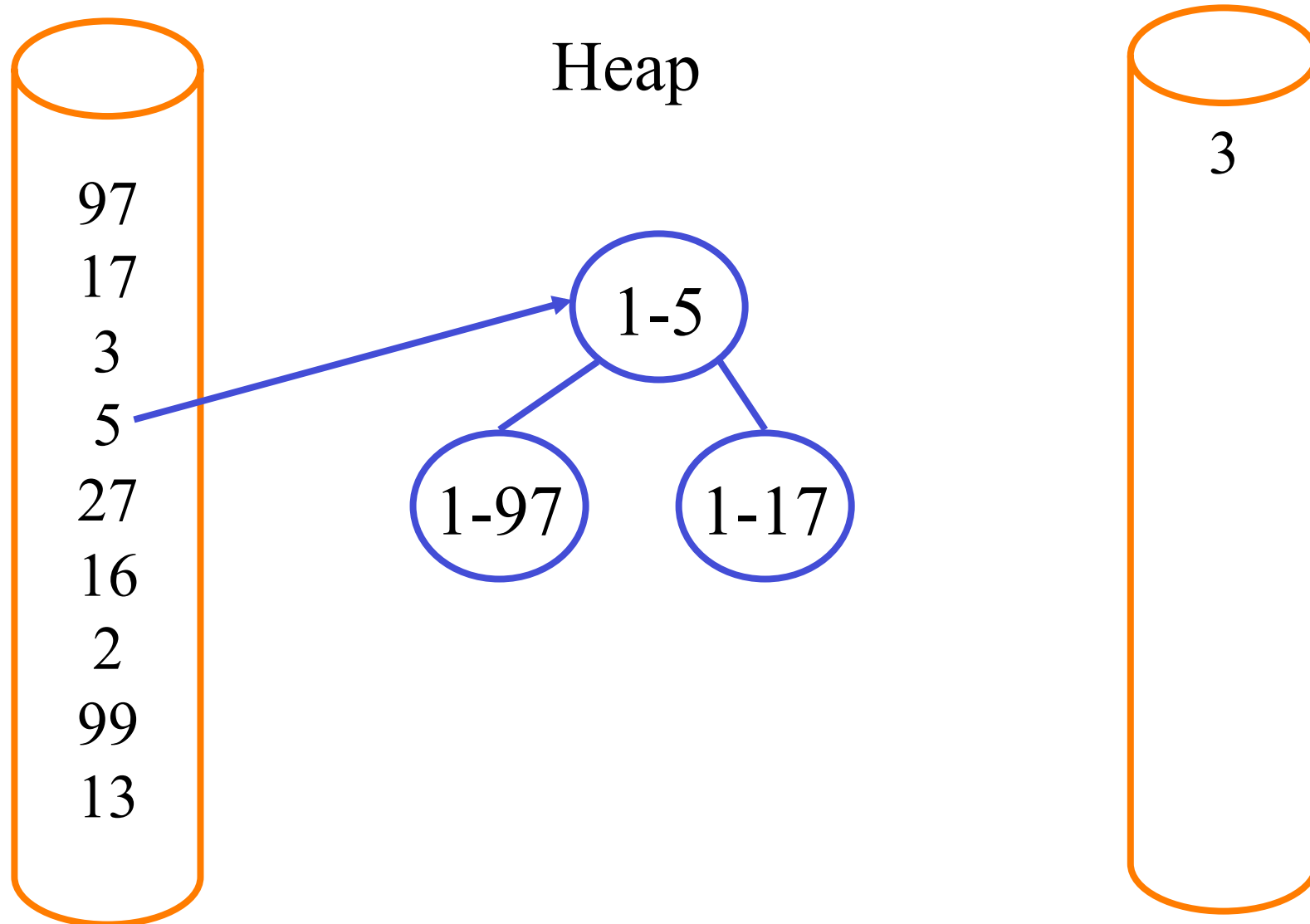
Replacement Selection während der Run-Generierung



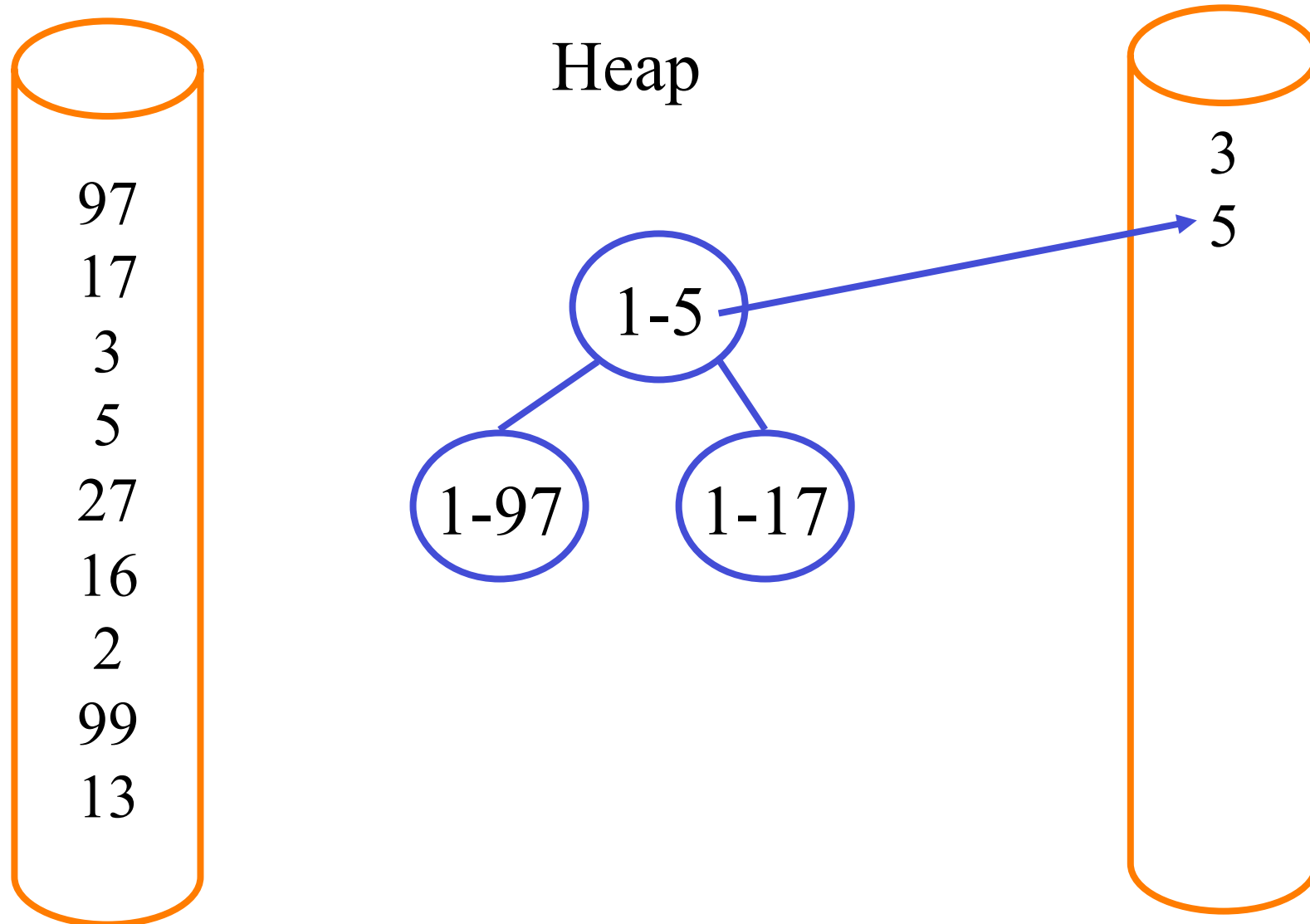
Replacement Selection während der Run-Generierung



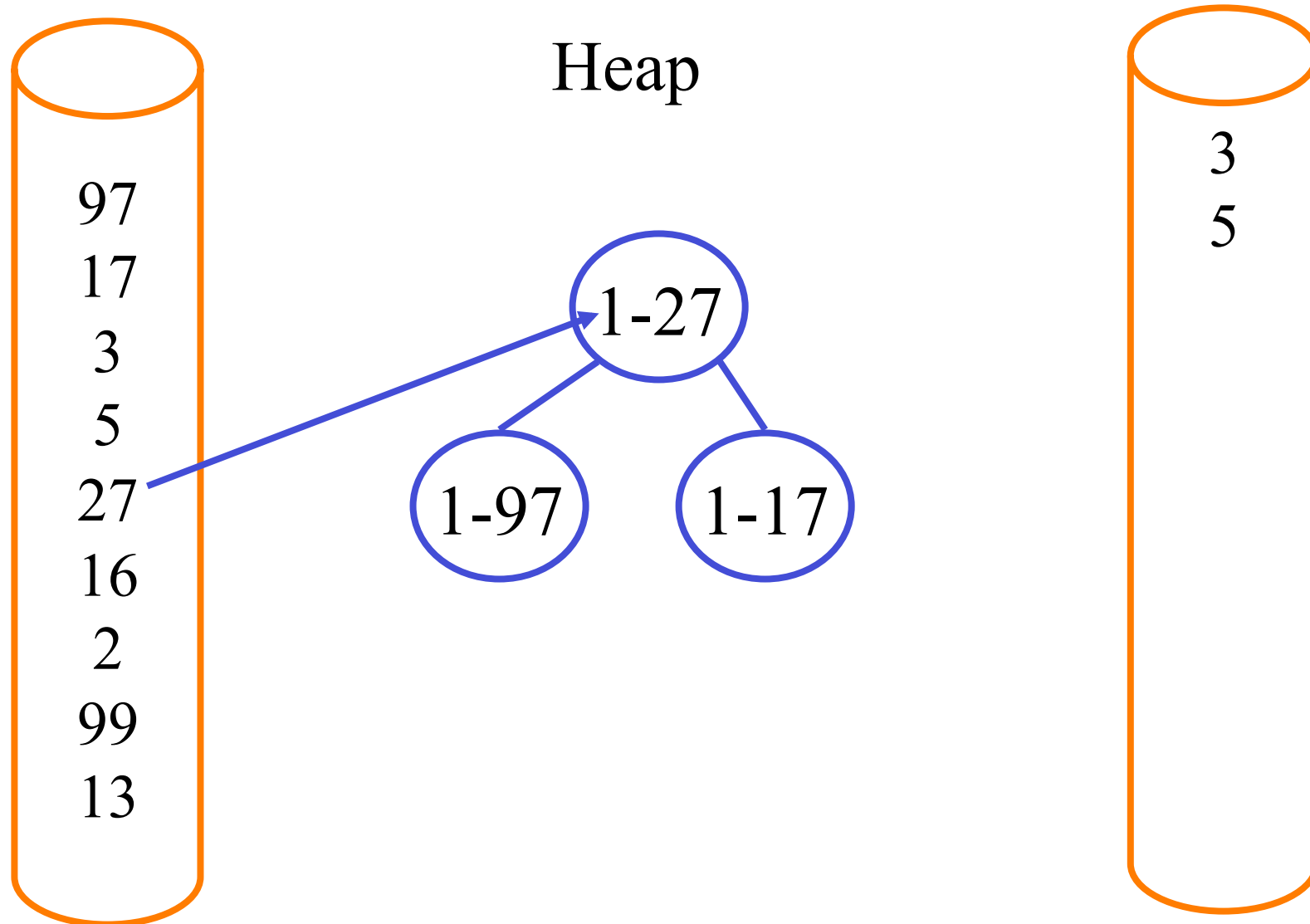
Replacement Selection während der Run-Generierung



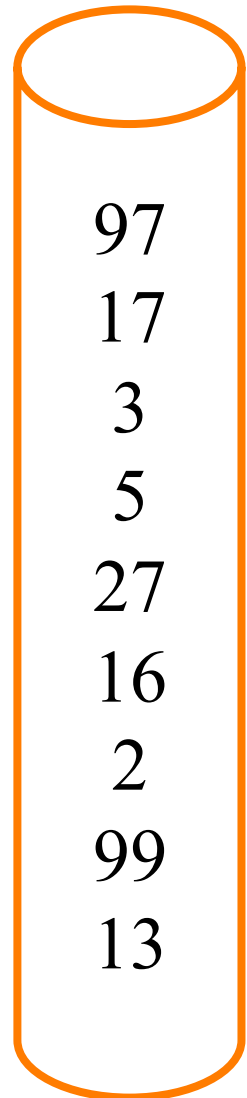
Replacement Selection während der Run-Generierung



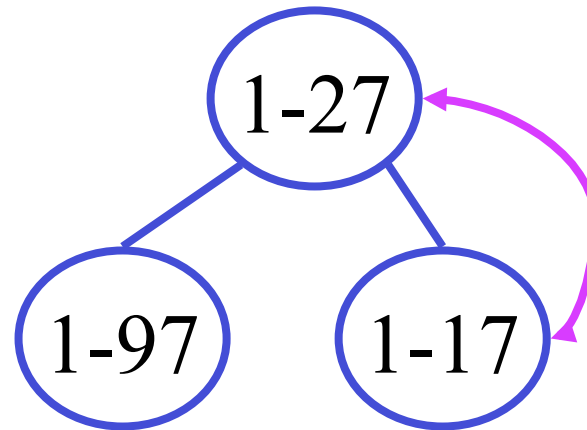
Replacement Selection während der Run-Generierung



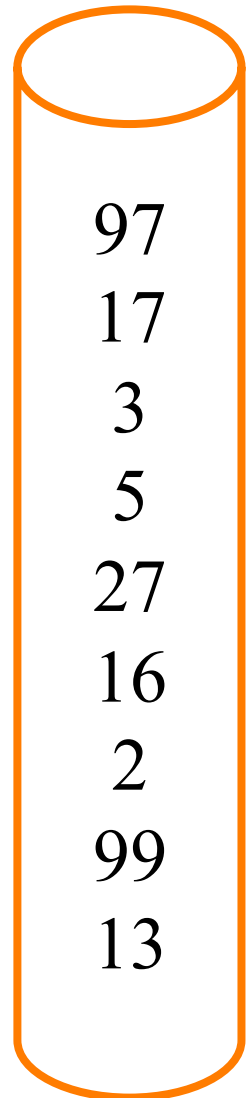
Replacement Selection während der Run-Generierung



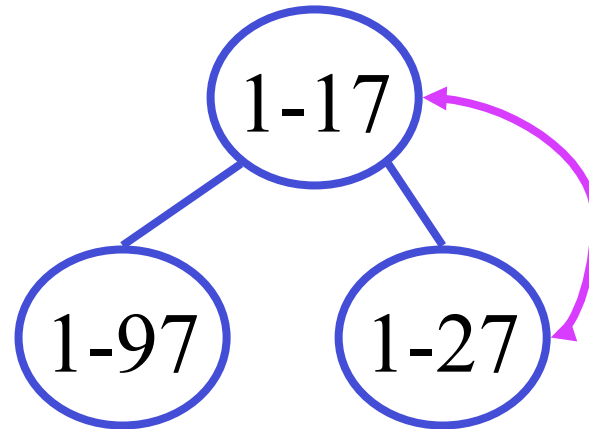
Heap



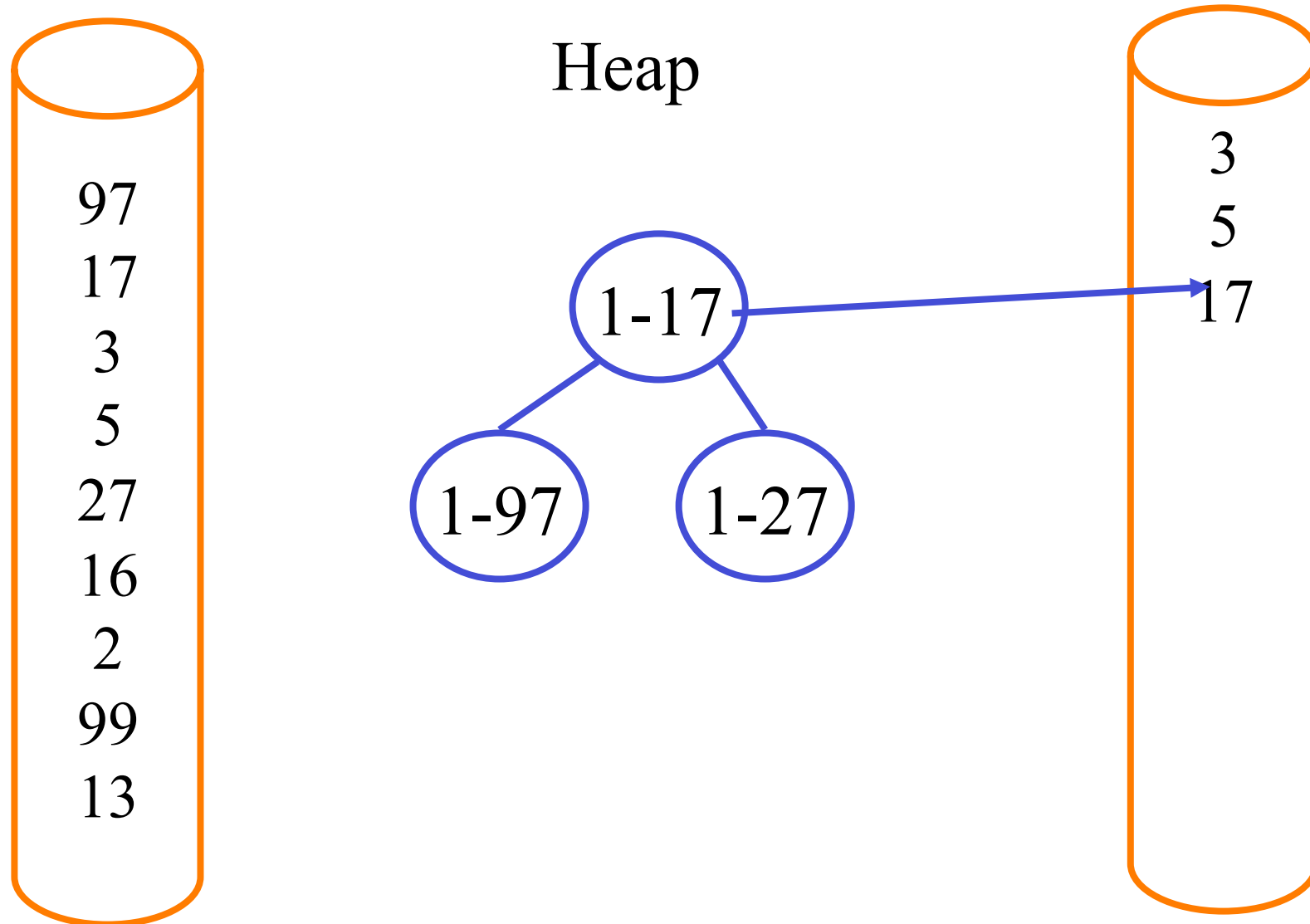
Replacement Selection während der Run-Generierung



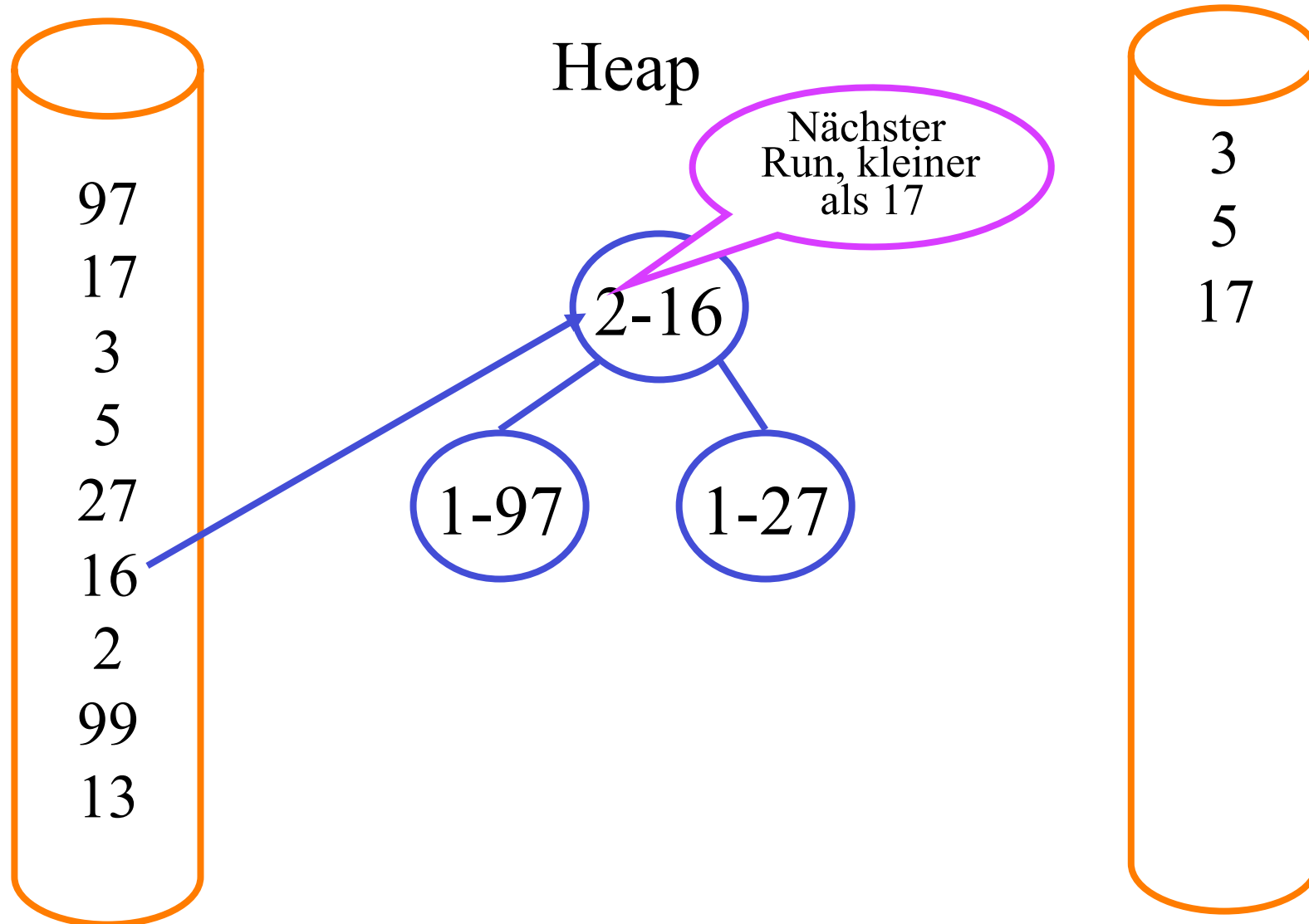
Heap



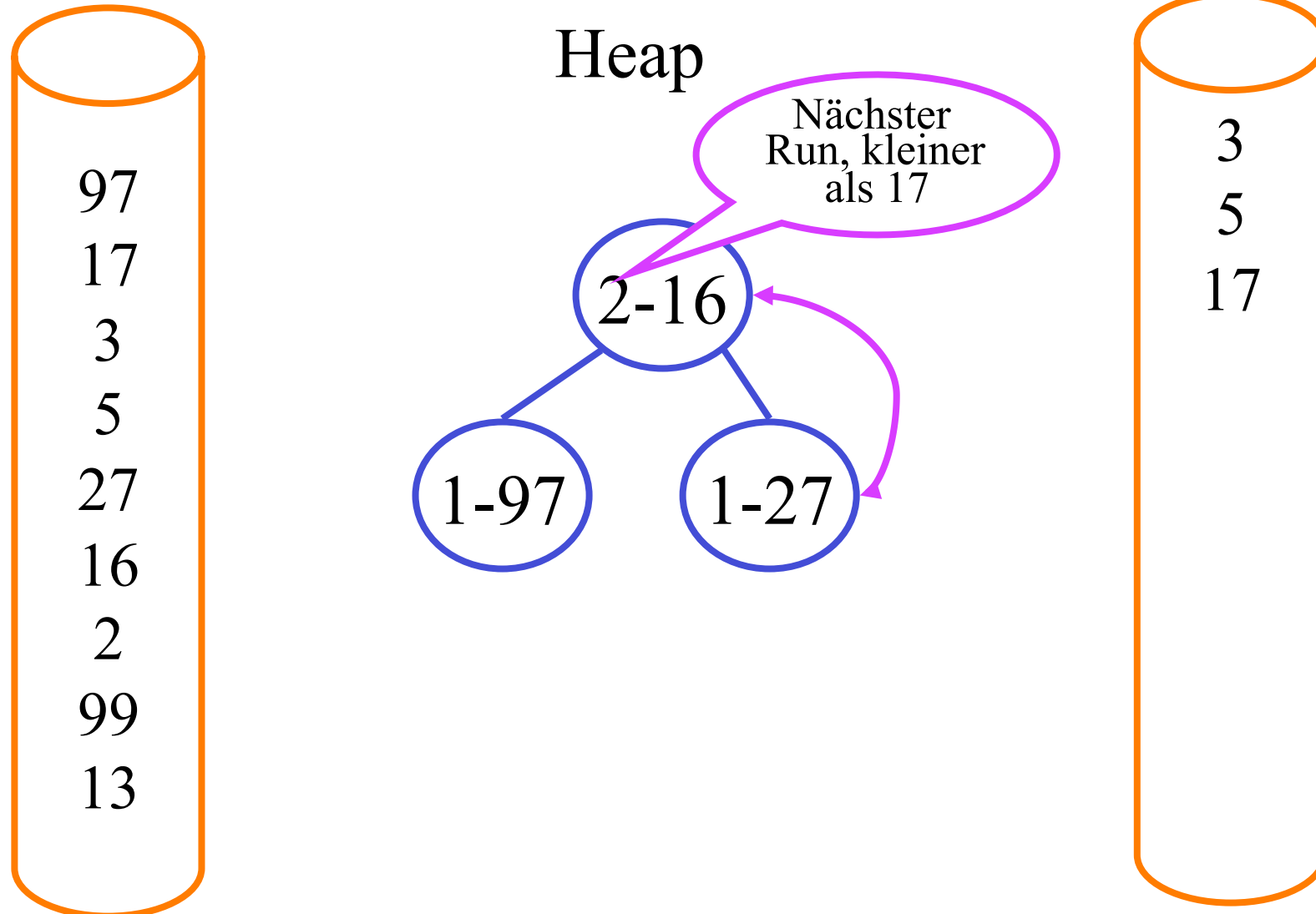
Replacement Selection während der Run-Generierung



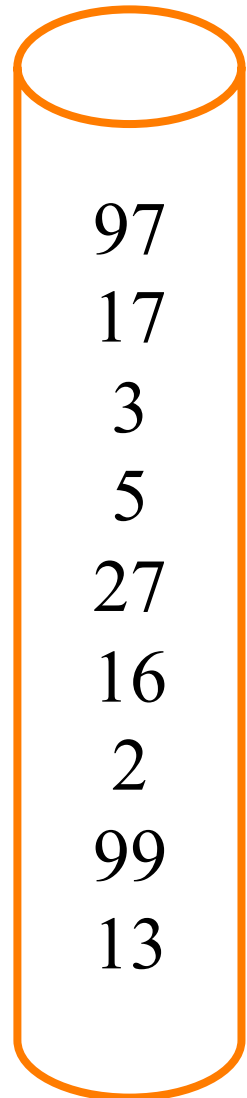
Replacement Selection während der Run-Generierung



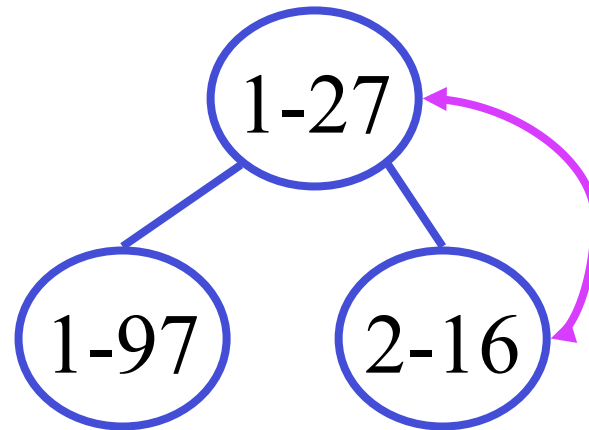
Replacement Selection während der Run-Generierung



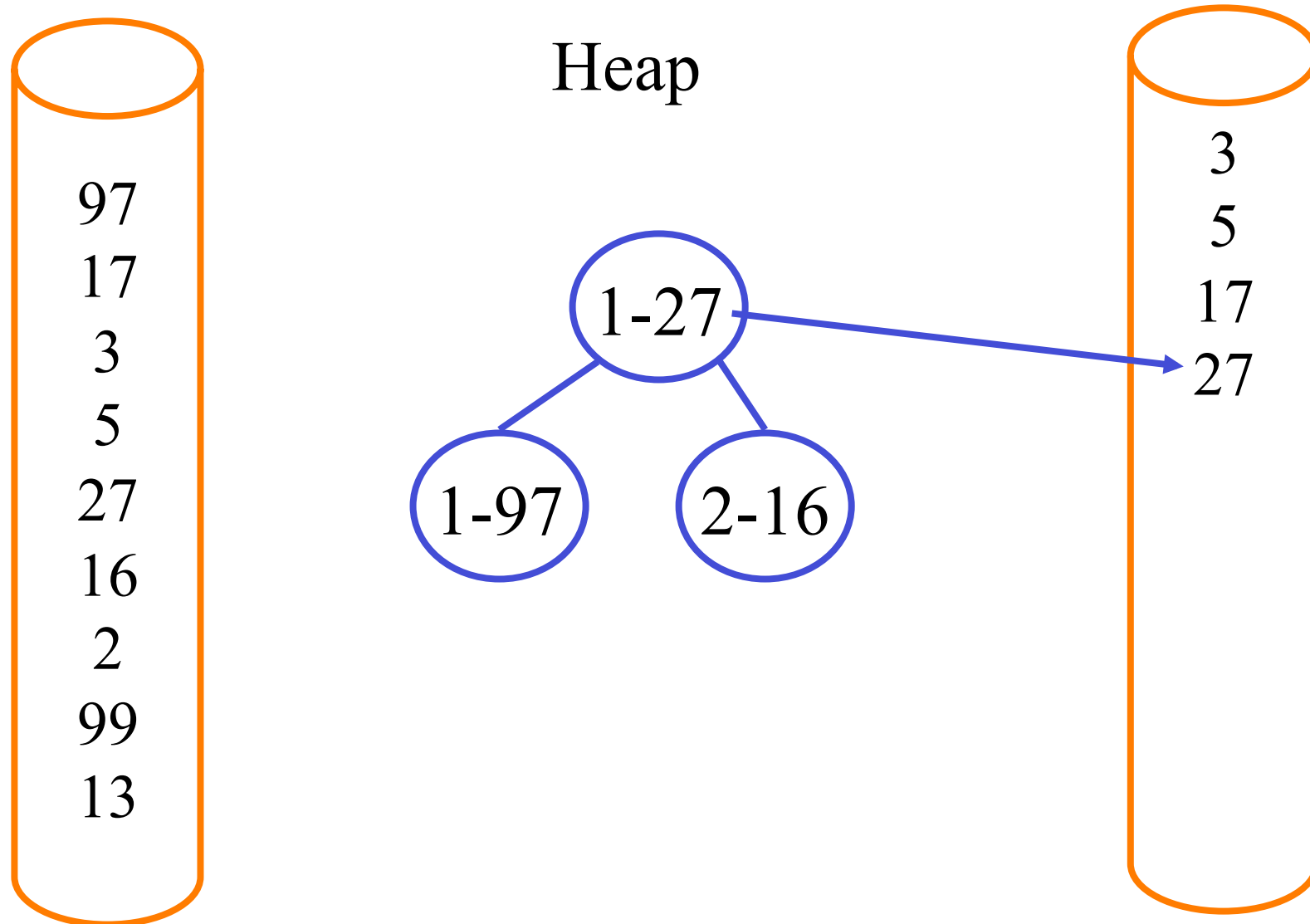
Replacement Selection während der Run-Generierung



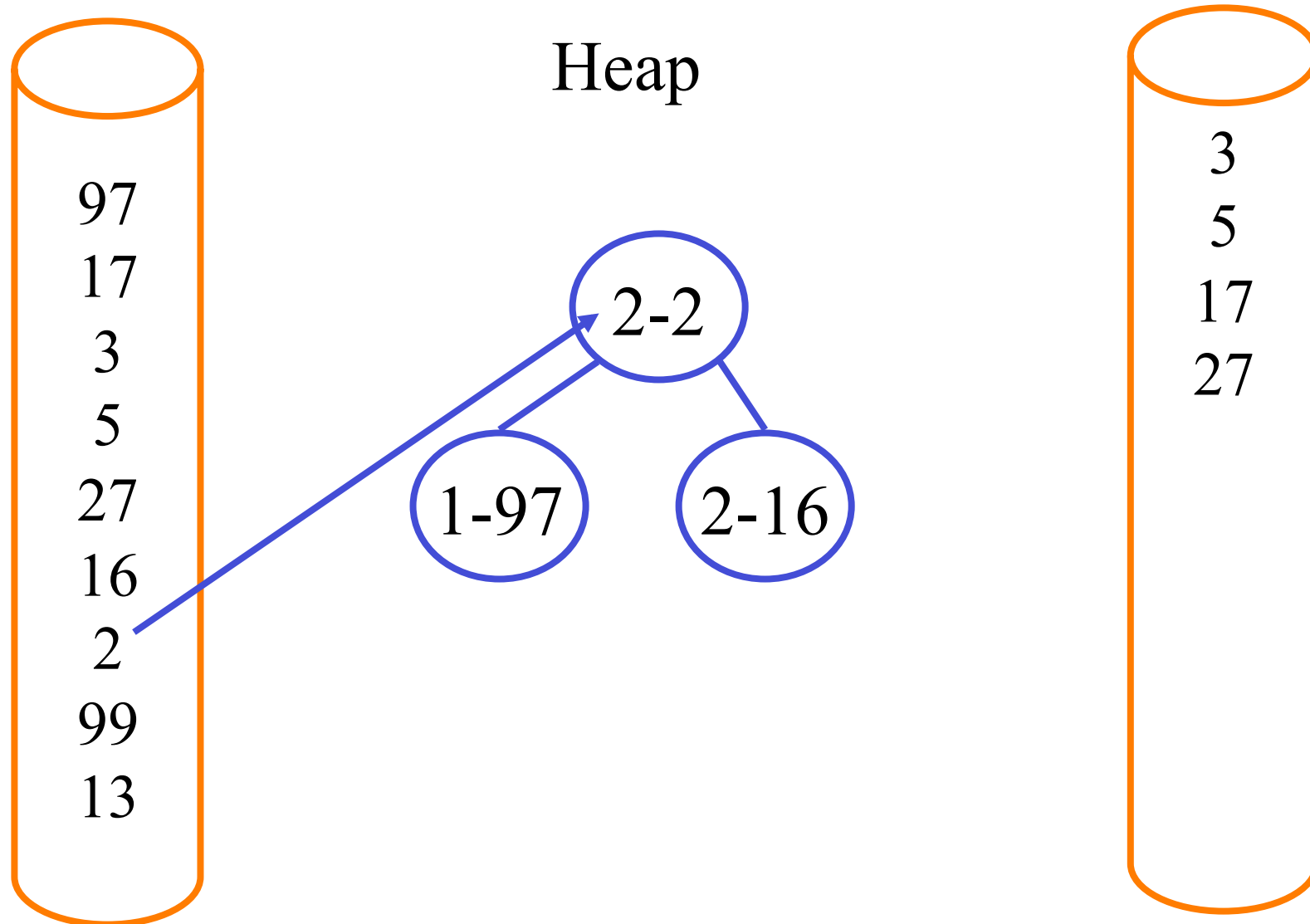
Heap



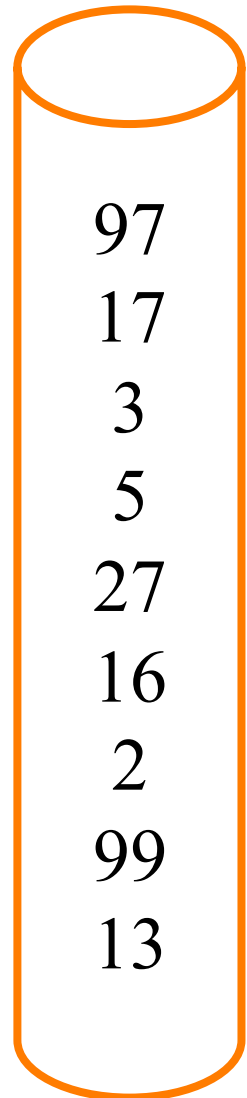
Replacement Selection während der Run-Generierung



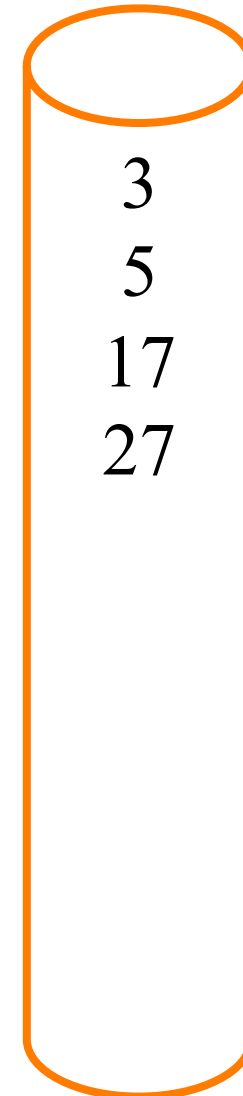
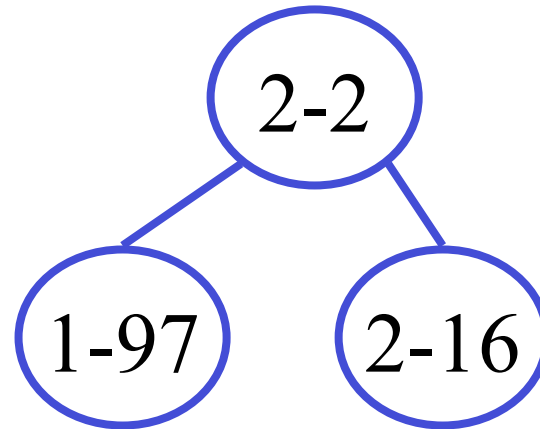
Replacement Selection während der Run-Generierung



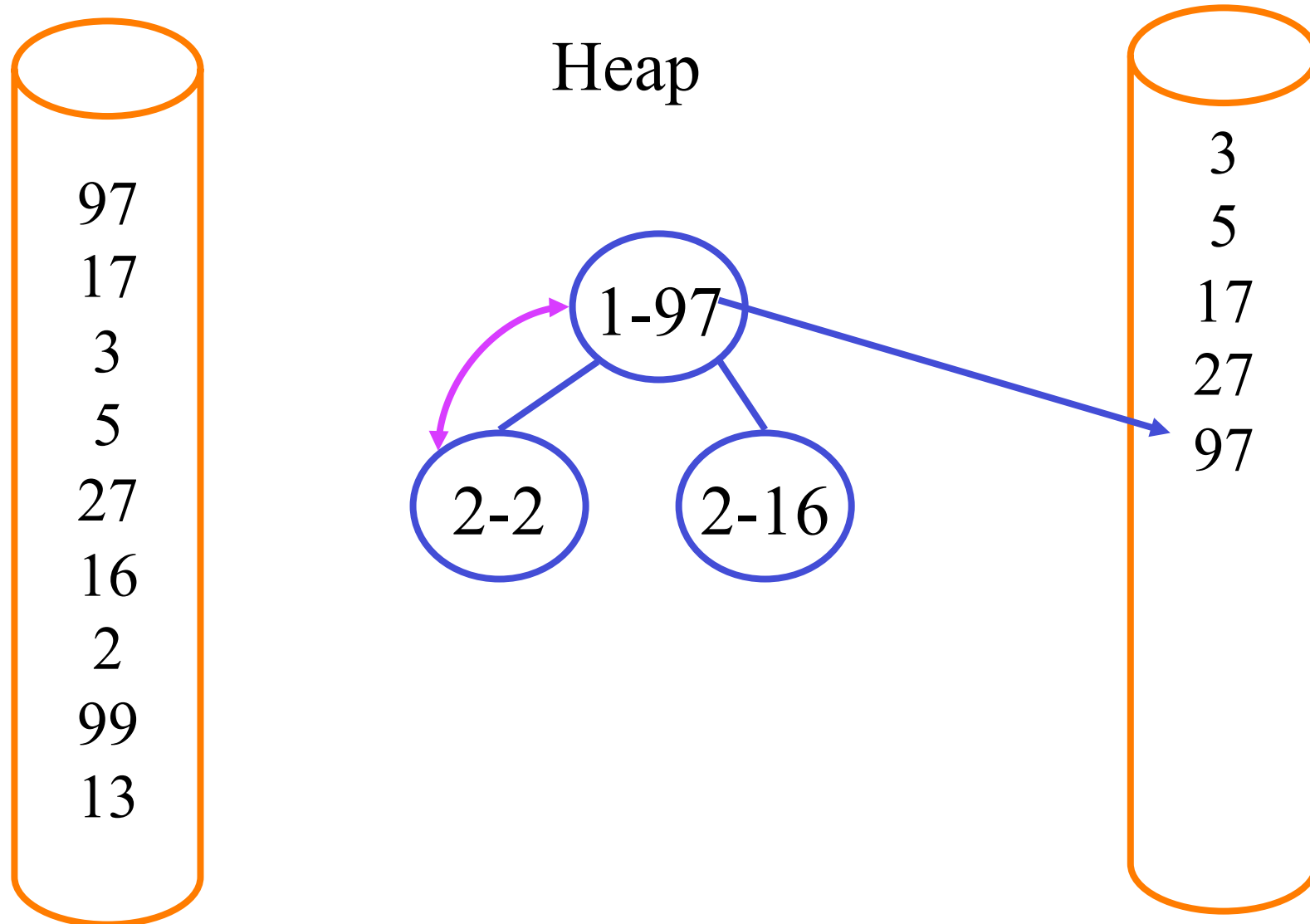
Replacement Selection während der Run-Generierung



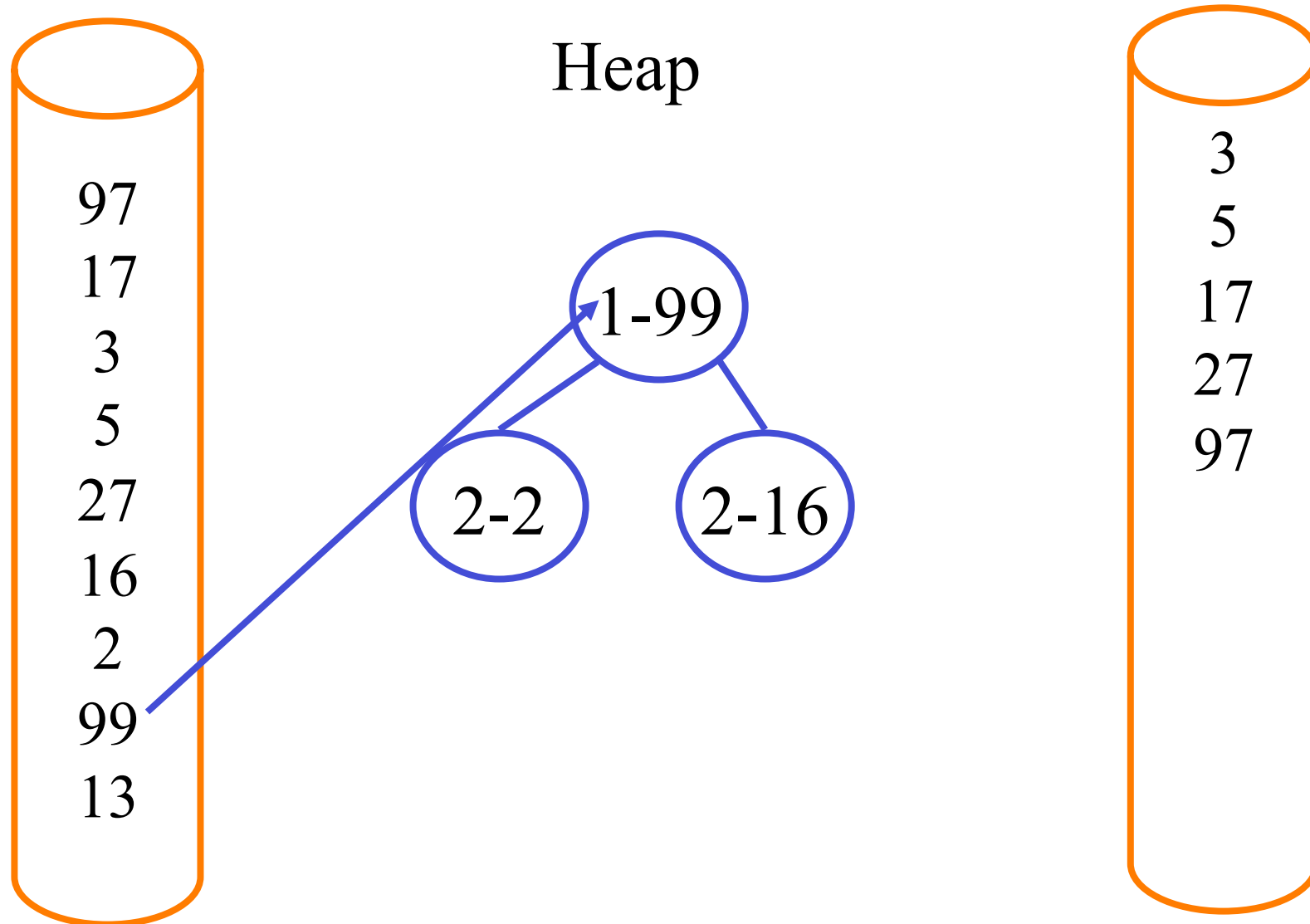
Heap



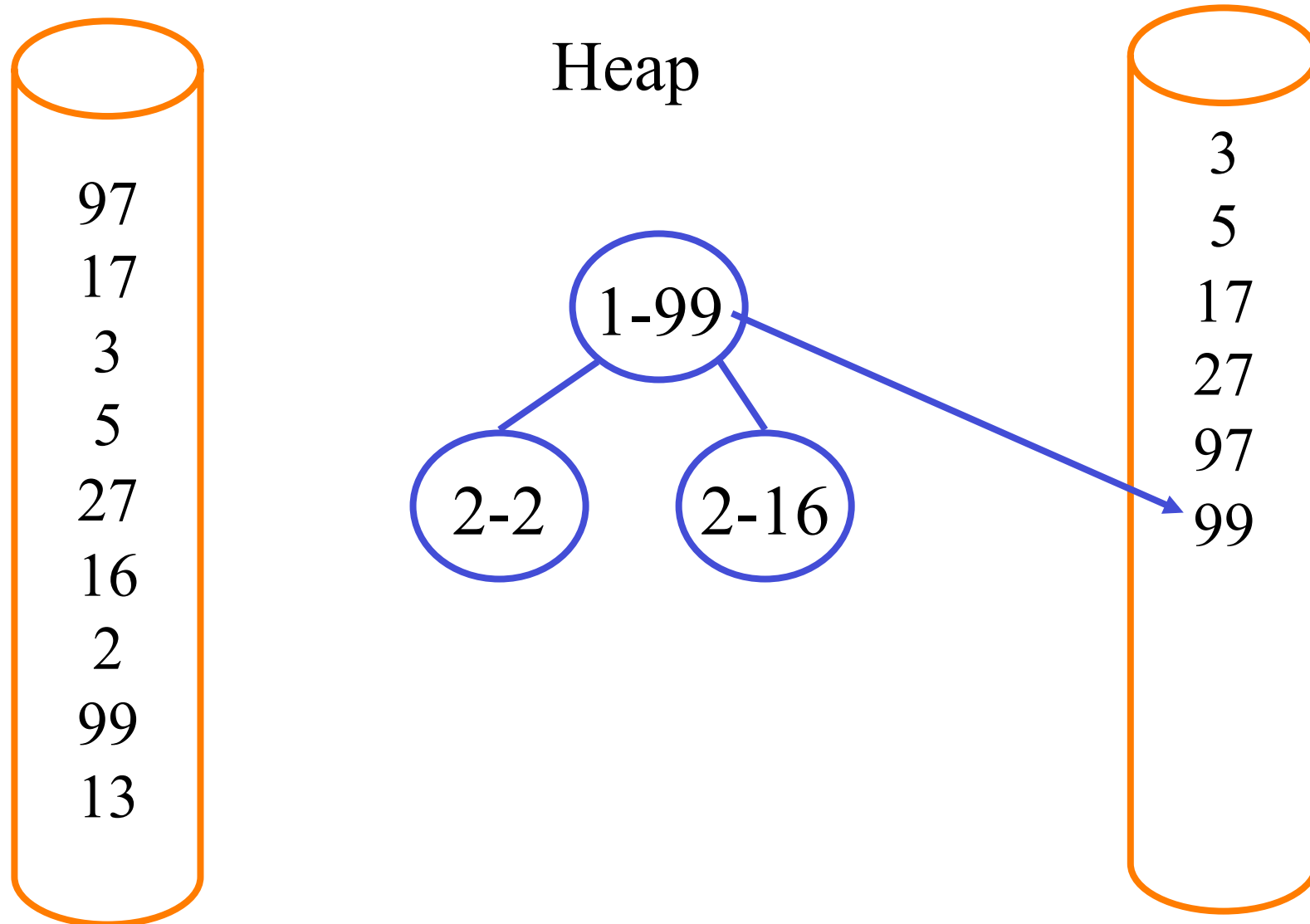
Replacement Selection während der Run-Generierung



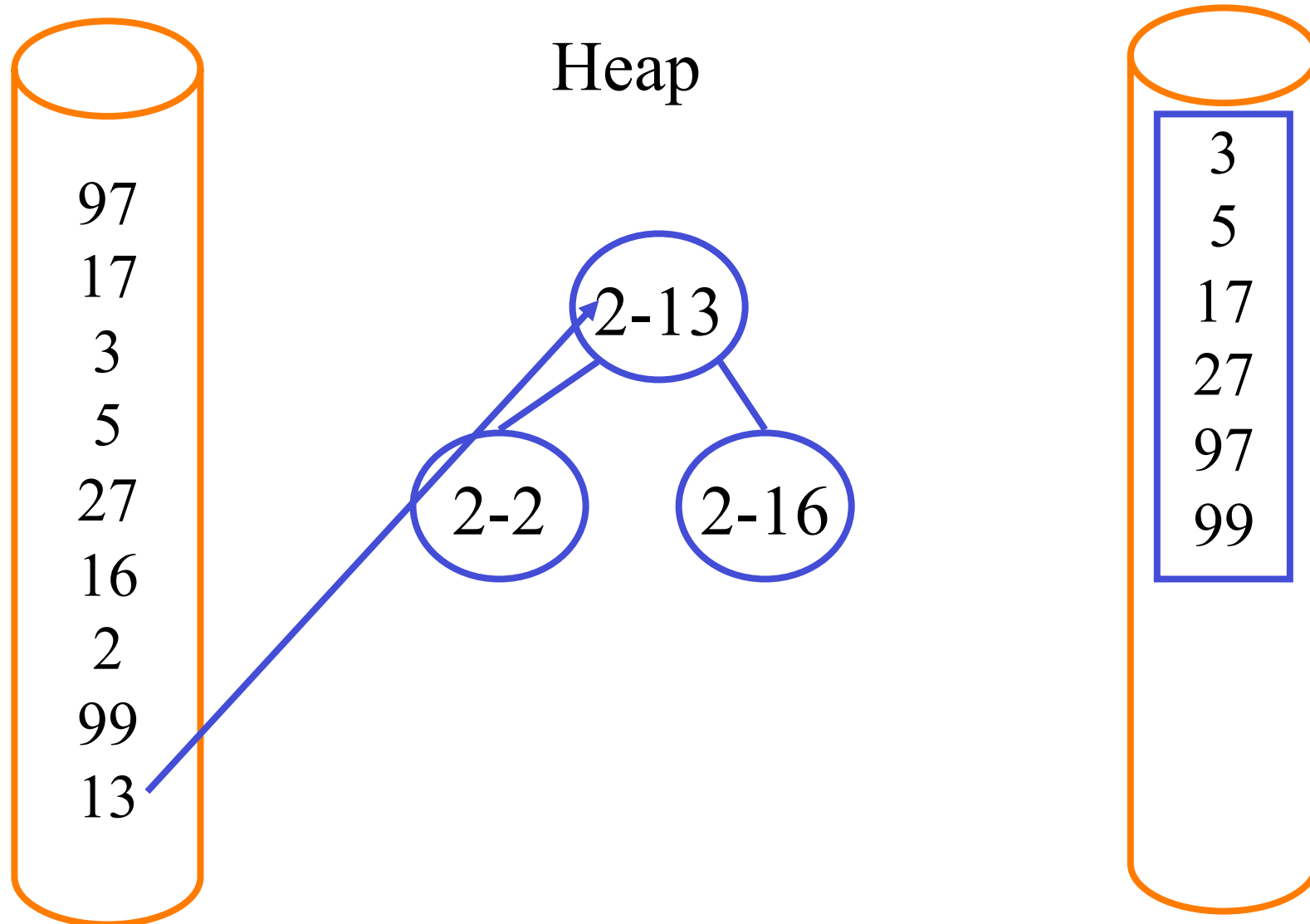
Replacement Selection während der Run-Generierung



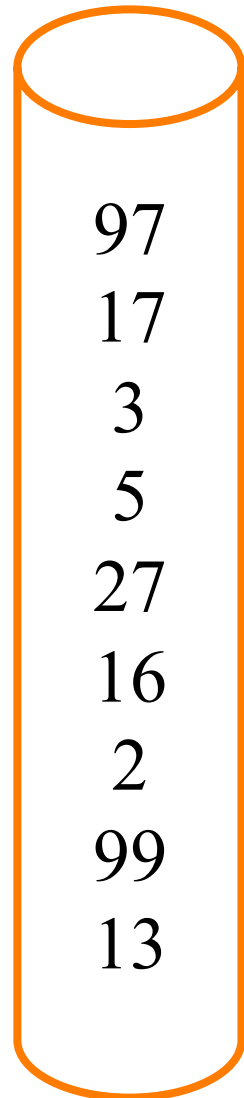
Replacement Selection während der Run-Generierung



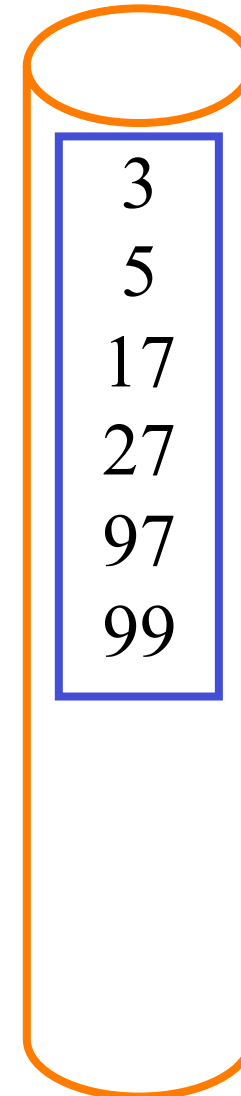
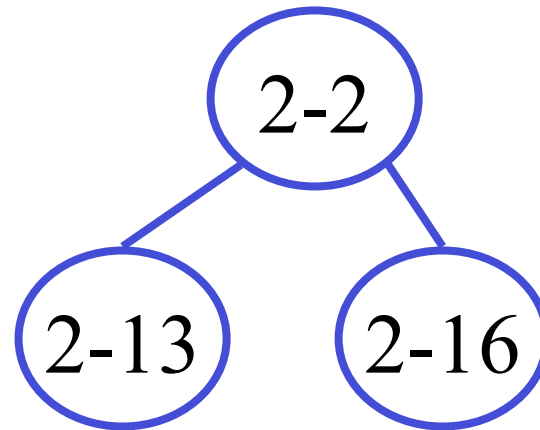
Replacement Selection während der Run-Generierung



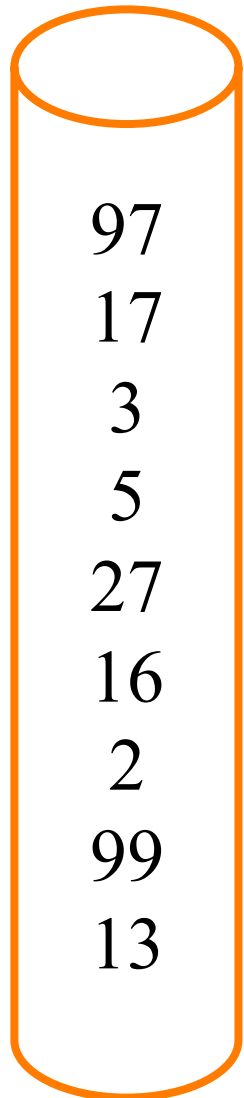
Replacement Selection während der Run-Generierung



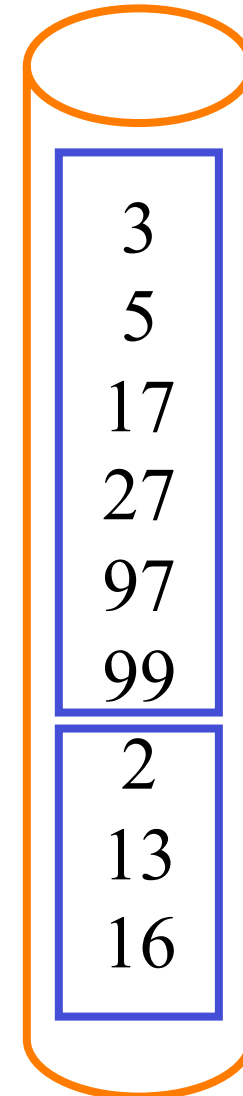
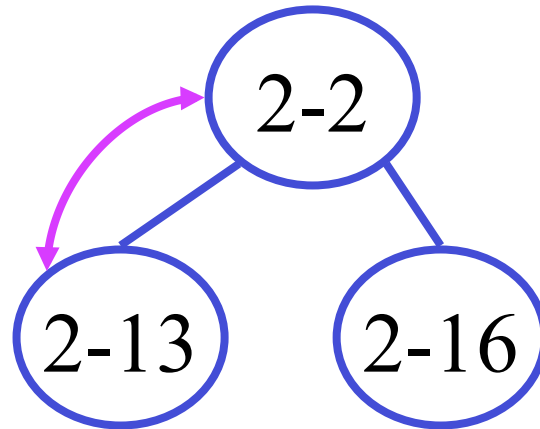
Heap



Replacement Selection während der Run-Generierung



Heap



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 haben
 - Der neue Datensatz passt dann nicht unbedingt in den freigewordenen 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

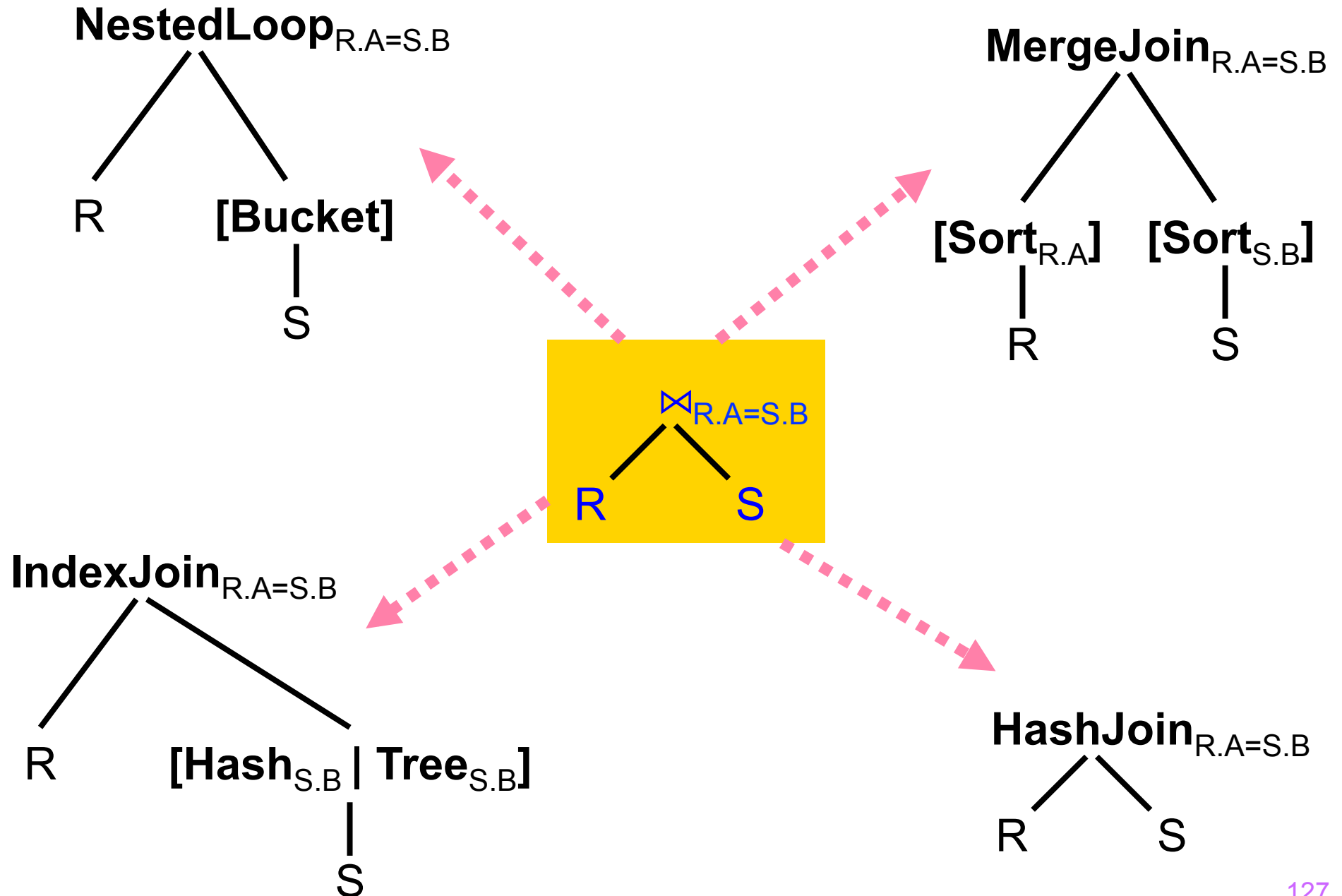
- R**
- 2
 - 3
 - 44
 - 5
 - 78
 - 90
 - 13
 - 17
 - 42
 - 89

$R \cap S$

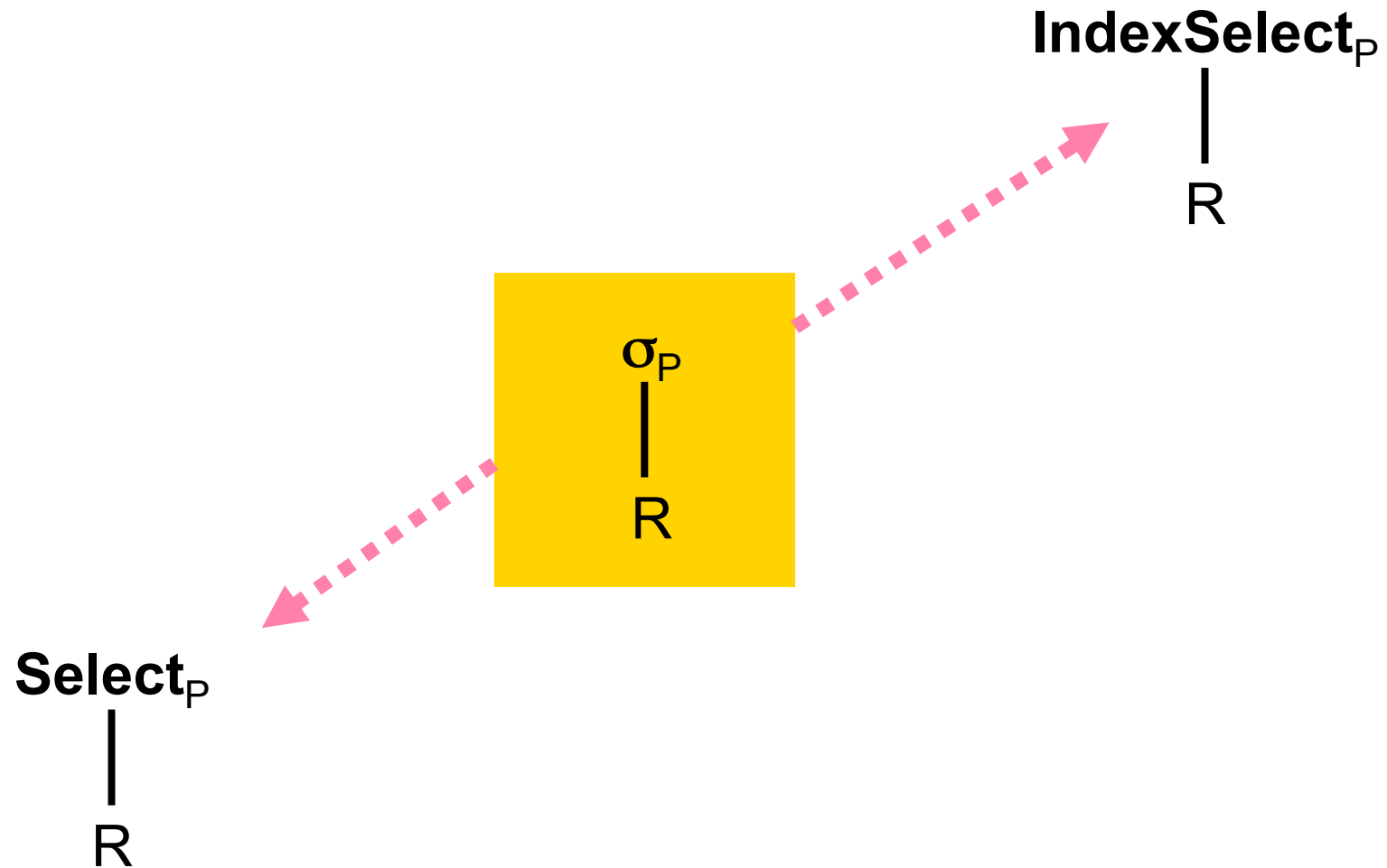
- Nested Loop: $O(N^2)$
- Sortieren: $O(N \log N)$
- Partitionieren und Hashing

- S**
- 44
 - 17
 - 97
 - 5
 - 6
 - 27
 - 2
 - 13
 - 9

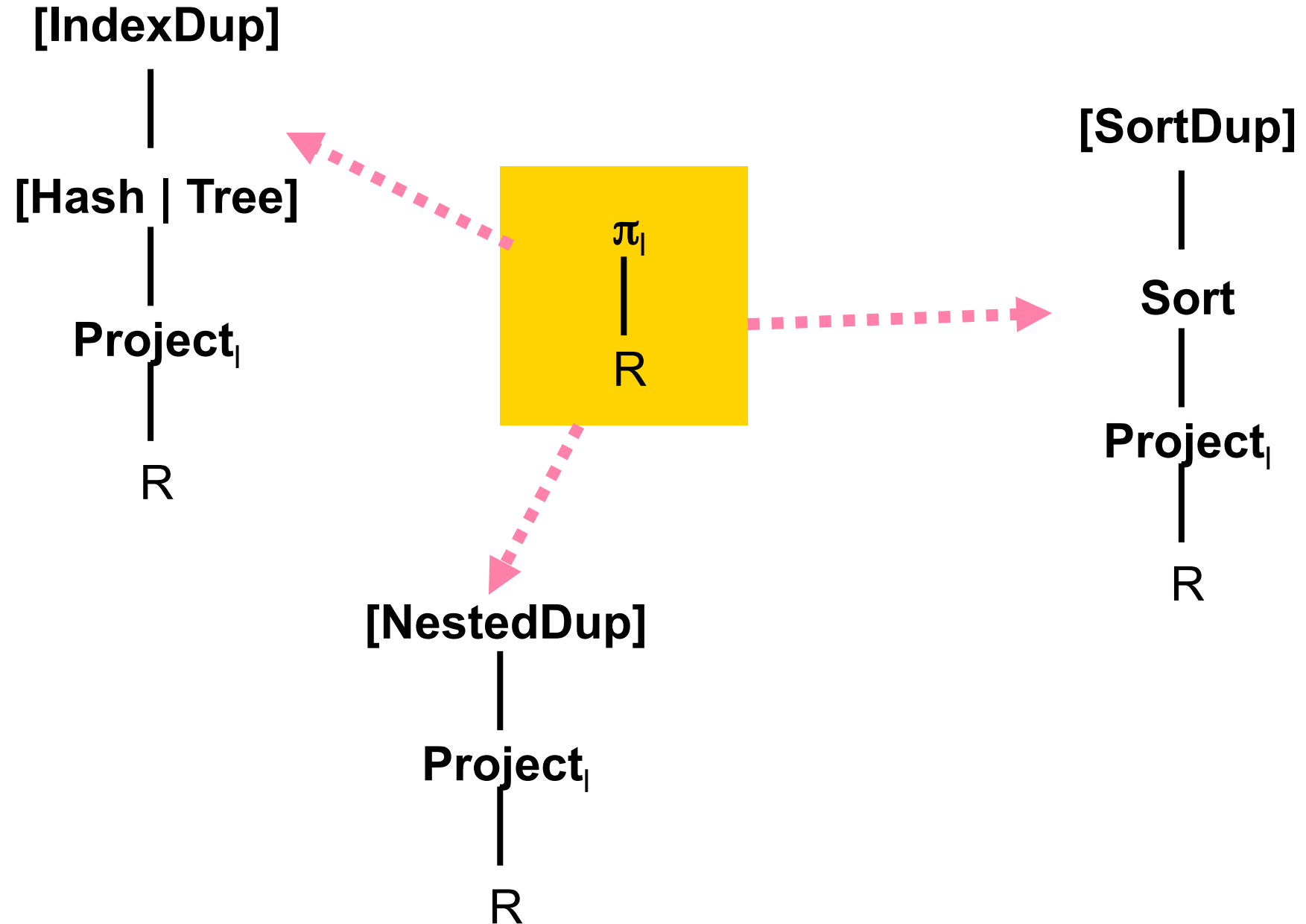
Übersetzung der logischen Algebra



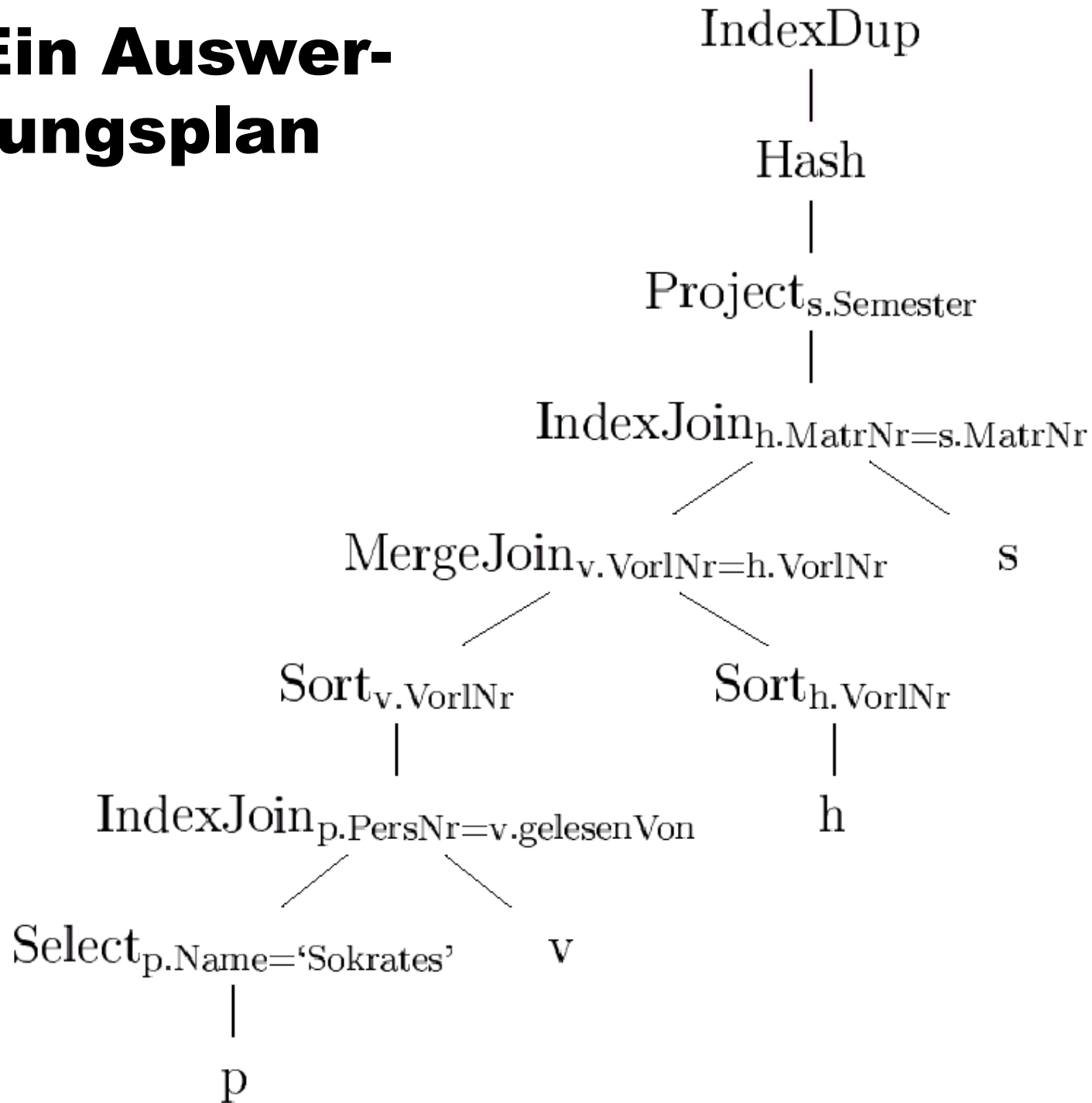
Übersetzung der logischen Algebra



Übersetzung der logischen Algebra

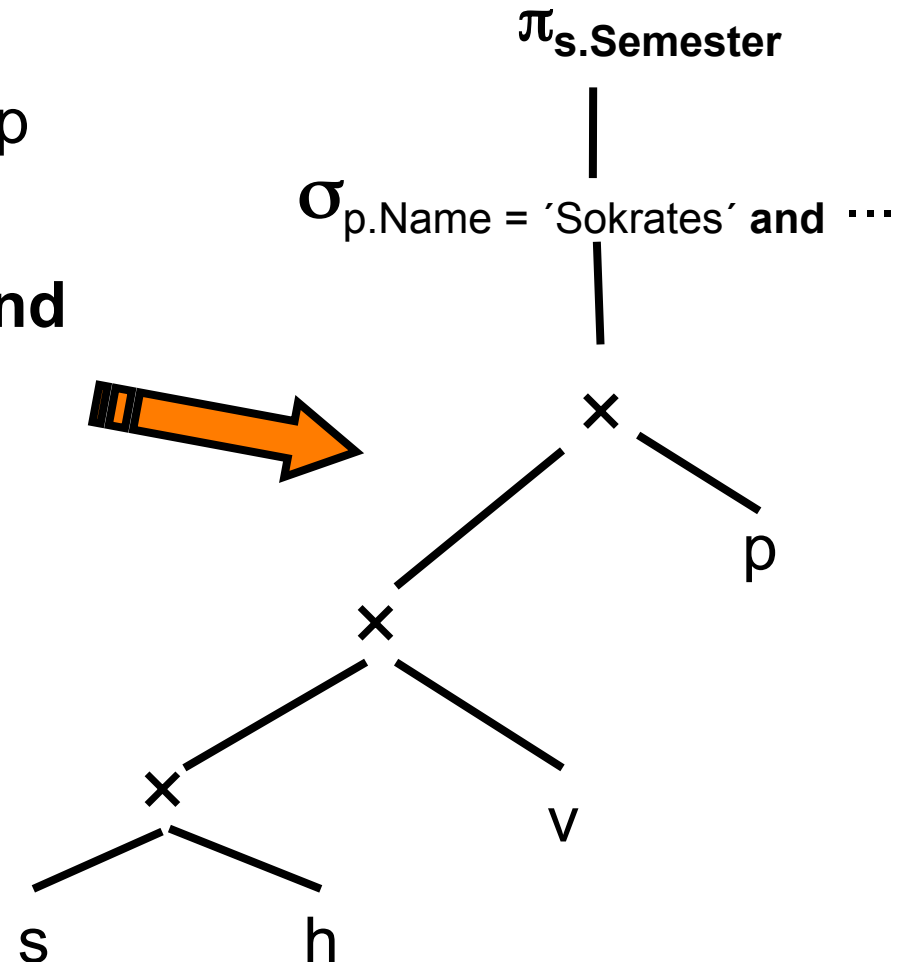


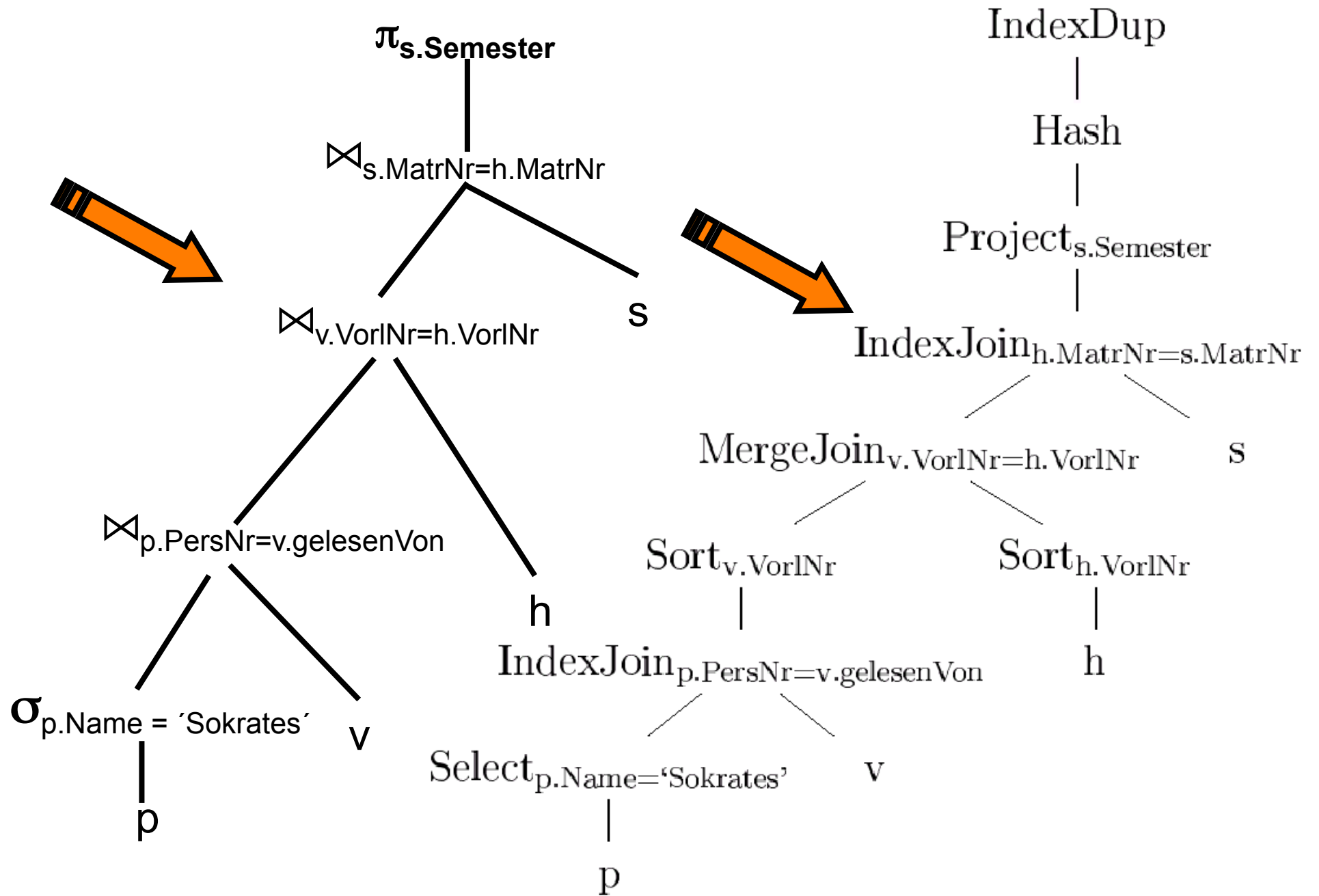
Ein Auswertungsplan



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





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 Antwortzeit-minimierend
- Behalte den billigsten Plan

Problemgröße

Suchraum (Planstruktur)

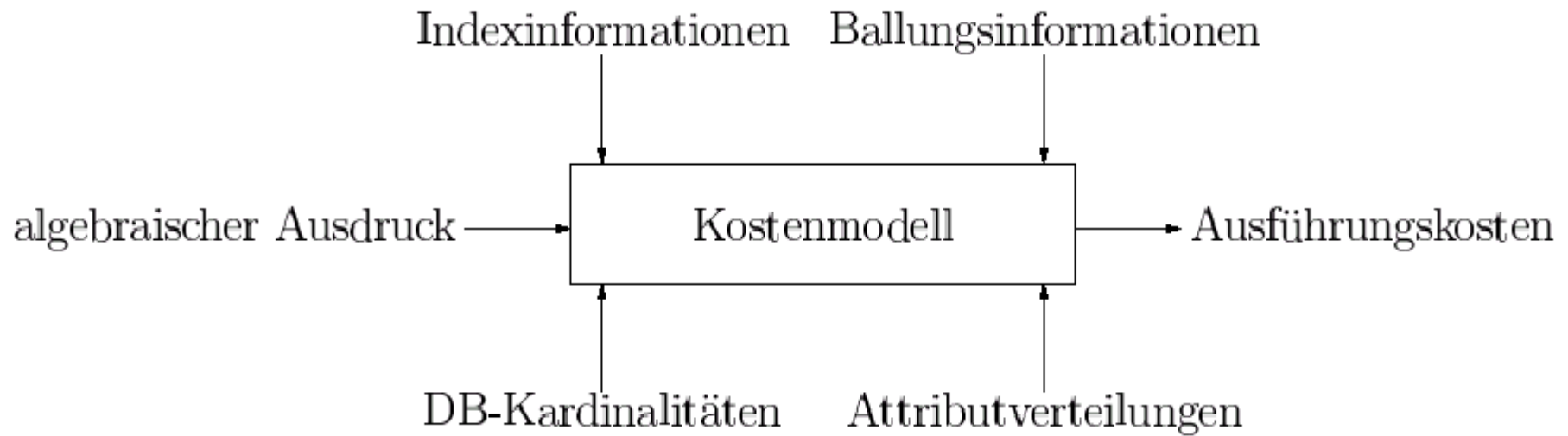
1. # Bushy-Pläne mit n Tabellen [Ganguly et al. 1992]:

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

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

2. Plankosten unterscheiden sich um Größenordnungen
3. 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
$$(\#R/i) / \#R \quad \text{oder} \quad 1/i$$
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 :

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

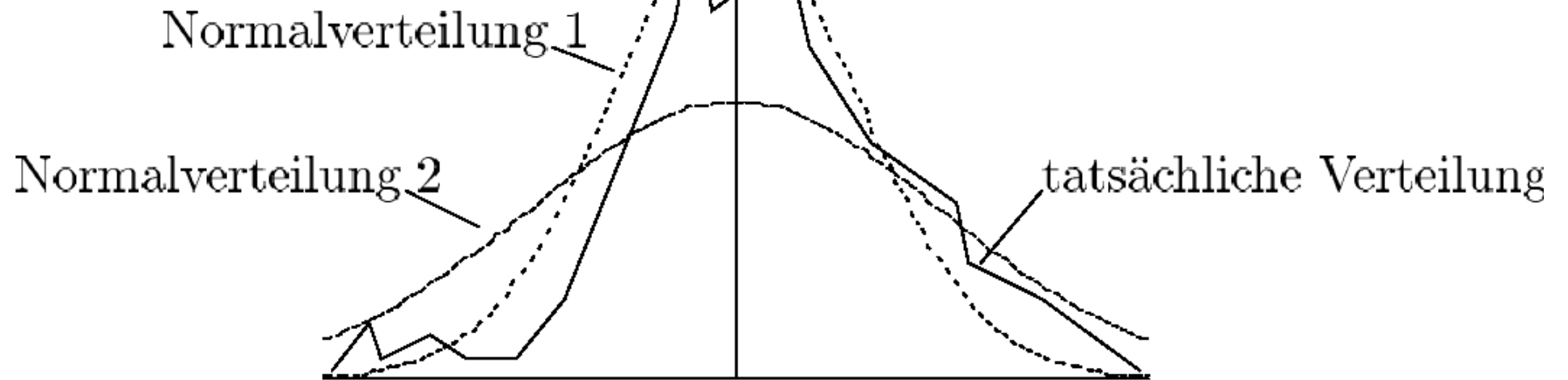
Abschätzung für einfache Fälle

Abschätzung der Selektivität:

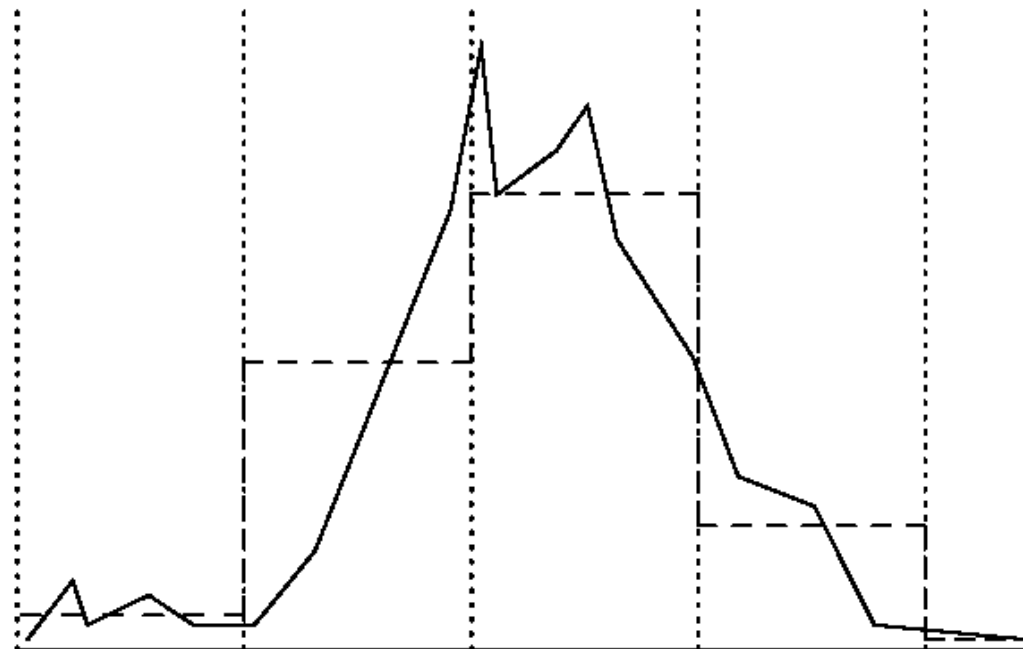
- $sel_{R.A=C} = \frac{1}{|R|}$
falls A Schlüssel von R
- $sel_{R.A=C} = \frac{1}{i}$
falls i die Anzahl der Attributwerte von $R.A$ ist (Gleichwert)
- $sel_{R.A=S.B} = \frac{1}{|R|}$
bei Equijoin von R mit S über Fremdschlüssel in S

Ansonsten z.B. Stichprobenverfahren

Parametrisierte Verteilung



Histogramm

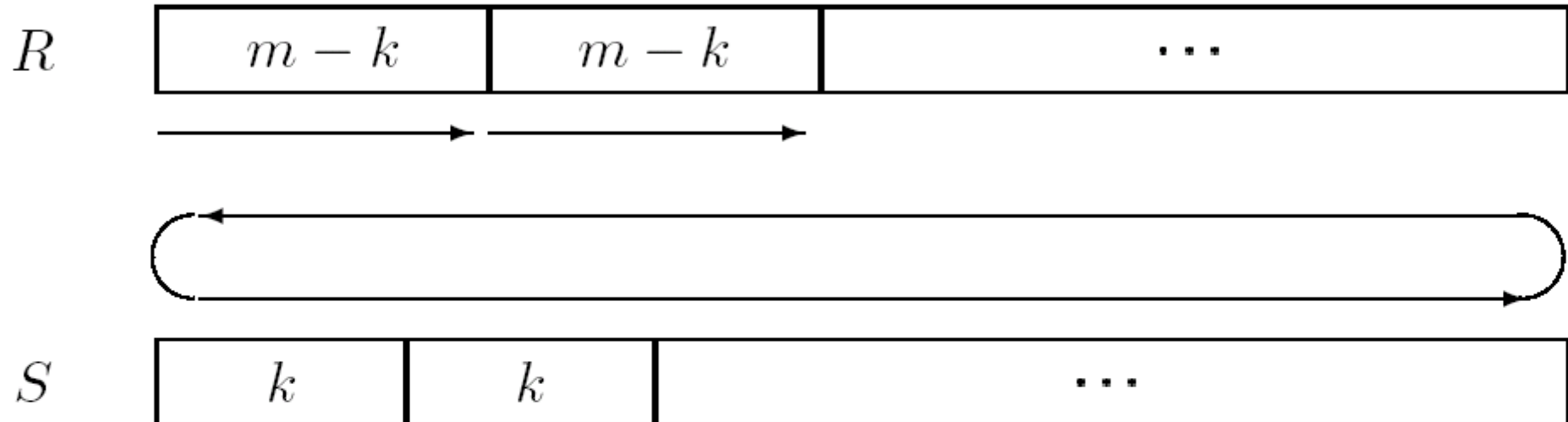


Kostenabschätzungen

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 einer 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 Leistungspässen

explain plan for

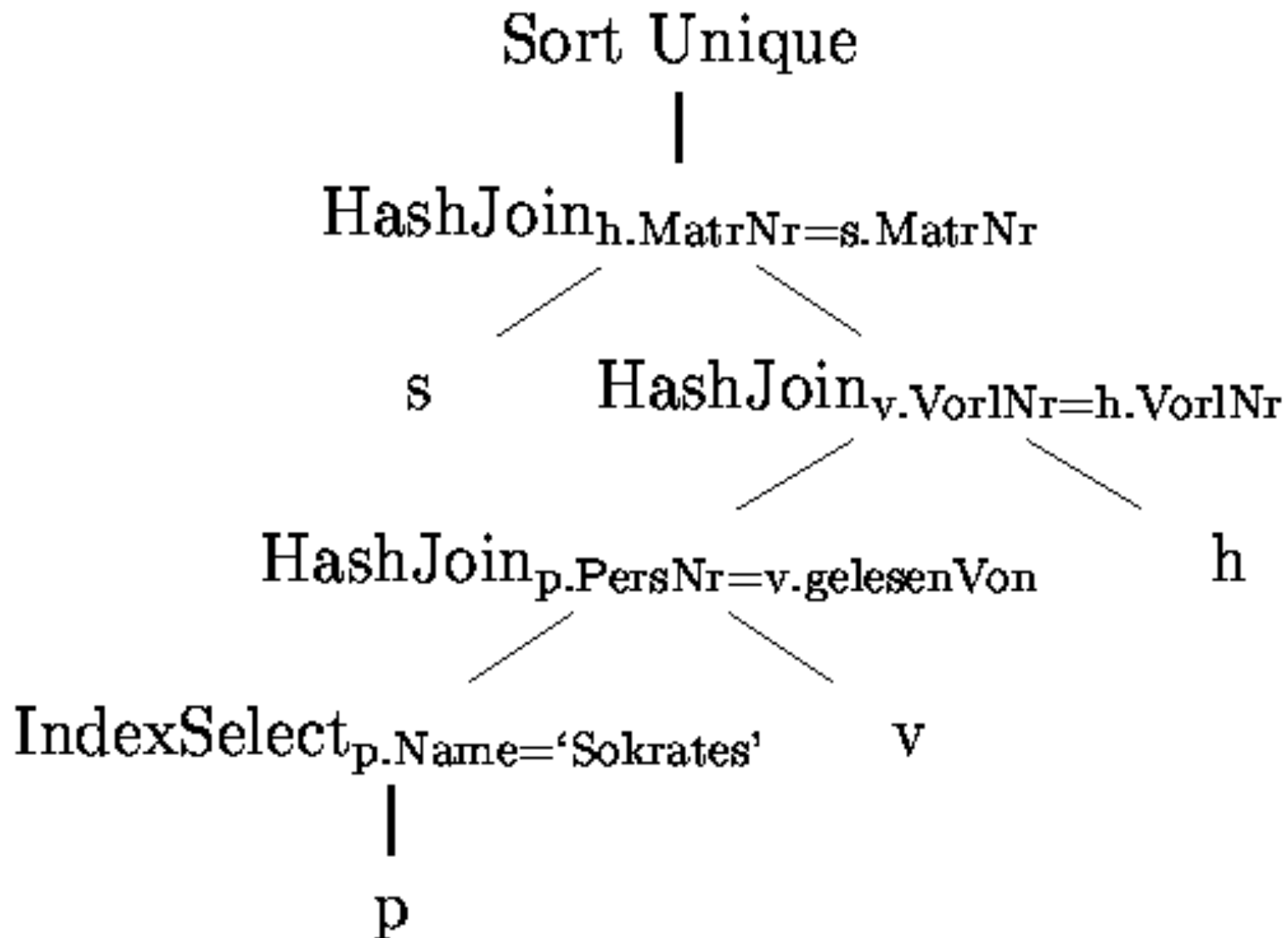
```
select distinct s.Semester
from Studenten s, hören h, Vorlesungen v, Professoren p
where p.Name = 'Sokrates' and v.gelesenVon = p.PersNr and
      v.VorlNr = h.VorlNr and h.MatrNr = s.MatrNr;
```

```
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



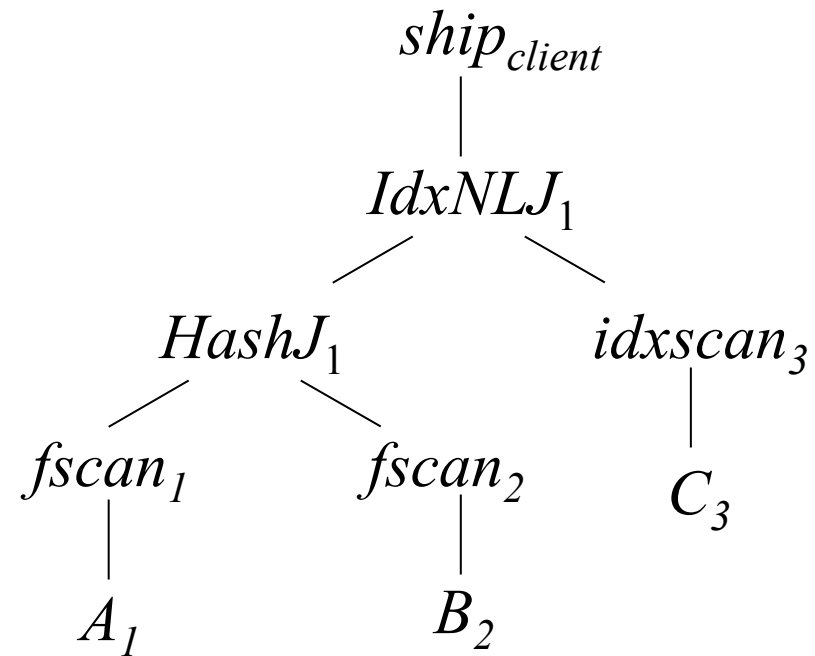
Beispiel

Anfrage

```
SELECT *  
FROM   A, B, C  
WHERE  A.a = B.a AND  
       B.b = C.a ;
```

- Blätter → Tabellen
- innere Knoten → Operatoren
- Annotation → Ausführungsorte

Auswertungsplan



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]:

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

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

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

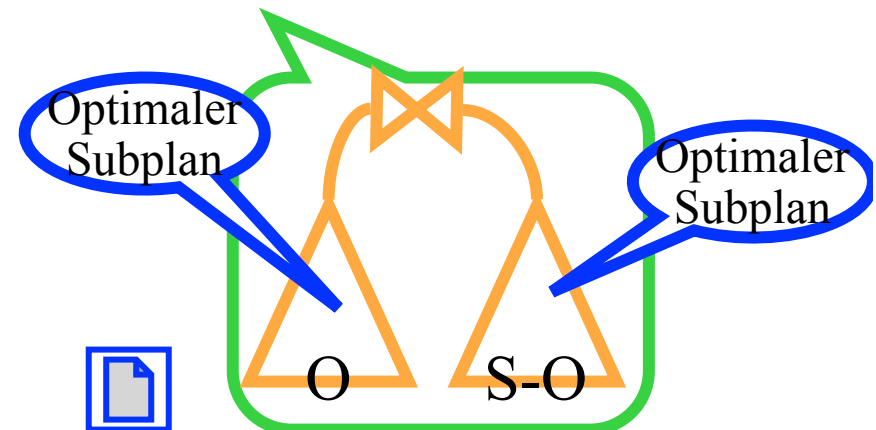
Dynamische Programmierung II

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}	$s(A) \bowtie s(C), s(C) \bowtie s(A)$	
{AB}	$s(A) \bowtie s(B), s(A) \bowtie is(B), is(B) \bowtie s(A), \dots$	
{C}	scan(C)	
{B}	scan(B), iscan(B)	
{A}	scan(A)	

DP - Beispiel

3. Phase: Finalisierung

Index	Pläne
{ABC}	$(\text{is}(B) \bowtie \text{s}(A)) \bowtie \text{s}(C)$
{BC}	...
{AC}	$\text{s}(A) \bowtie \text{s}(C)$
{AB}	$\text{s}(A) \bowtie \text{is}(B), \text{is}(B) \bowtie \text{s}(A)$
{C}	$\text{scan}(C)$
{B}	$\text{scan}(B), \text{iscan}(B)$
{A}	$\text{scan}(A)$

Algorithmus DynProg

Function DynProg

input A query q over relations R_1, \dots, R_n // *Set of relations to be joined*

output A query execution plan for 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  $S \subseteq \{R_1, \dots, R_n\}$  such that  $|S| == i$  do {
7:           BestPlansTable( $S$ ) =  $\emptyset$ 
8:           for all  $O \subset S$  do {
9:               BestPlansTable( $S$ ) = BestPlansTable( $S$ )  $\cup$ 
                joinPlans(BestPlansTable( $O$ ), BestPlansTable( $S - O$ ))
10:            prunePlans(BestPlansTable( $S$ ))
11:        }
12:    }
13: }
14: prunePlans(BestPlansTable( $\{R_1, \dots, R_n\}$ ))
15: return BestPlansTable( $\{R_1, \dots, R_n\}$ )
```

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

Enumeration

- Effiziente Enumeration [Vance 96]
 - anstatt zunächst alle 2-elem, 3-elem, ..., n-elem Pläne sequentiell zu enumerieren: effizientes Interleaving
 - nur Pläne aus bereits berechneten Zeilen notwendig
- Beispiel:
 1. A → 2. B → 3. AB → 4. C → 5. AC → 6. BC → 7. ABC