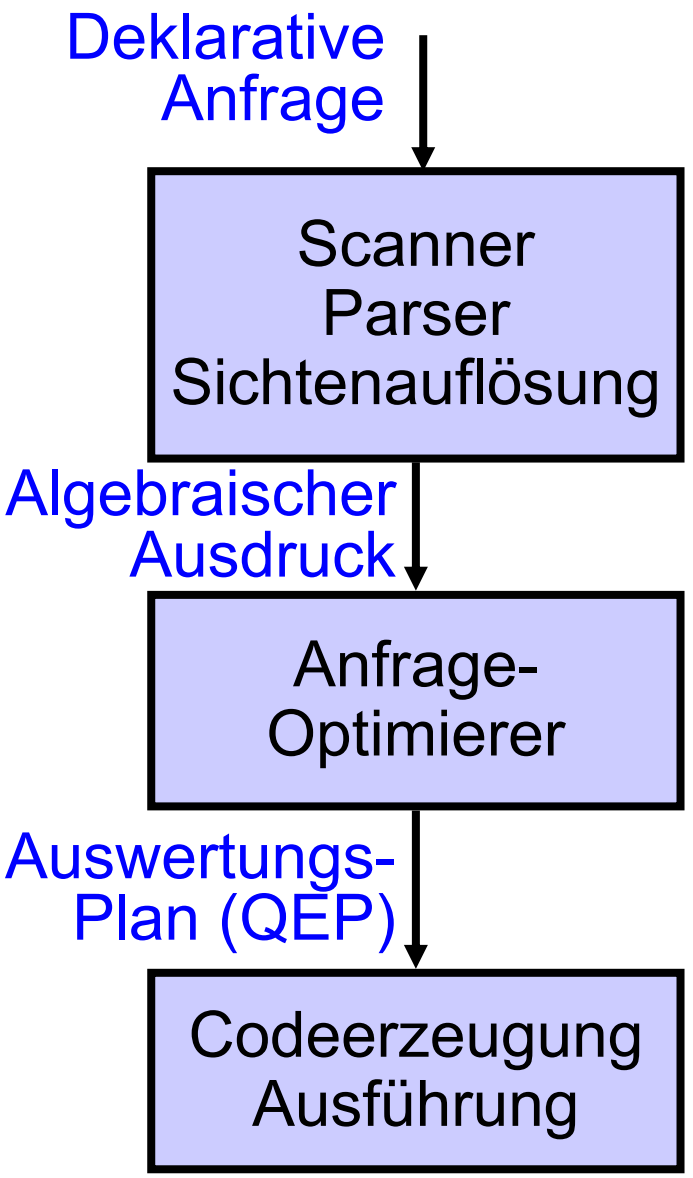
Datenbanken

Gliederung

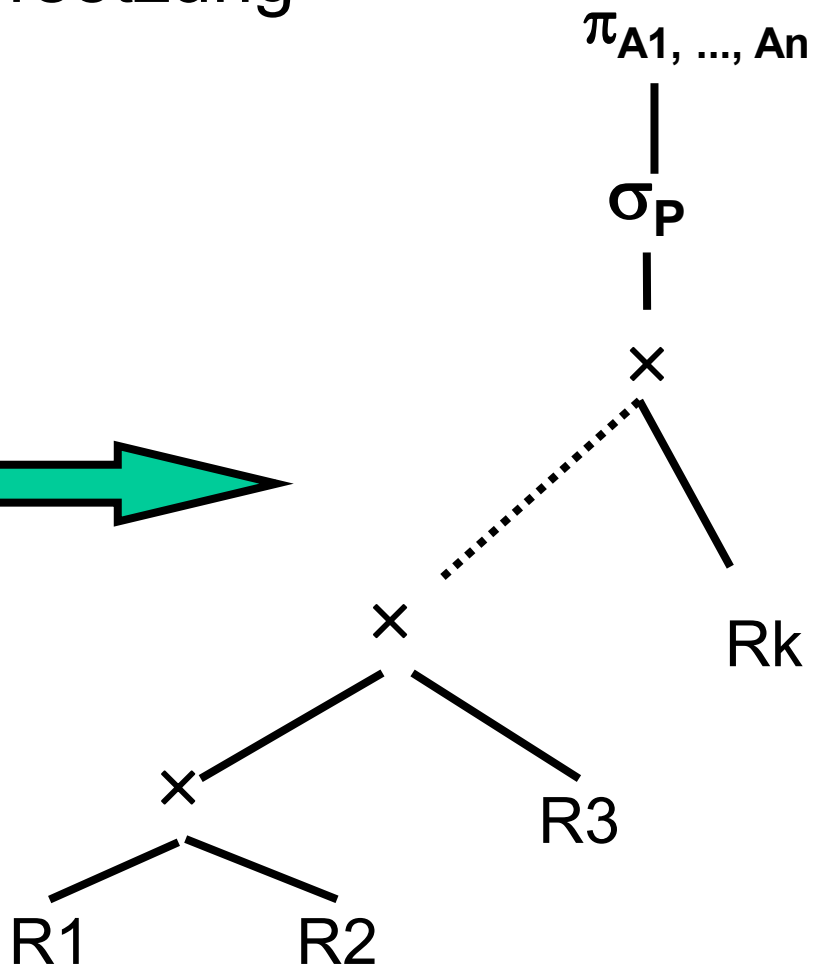
Ablauf der Anfrageoptimierung

Vgl. insbesondere:
Kemper, Datenbanken.
(Umfangreicher Lehrbuchbestand in der Bibliothek!)



Kanonische Übersetzung

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

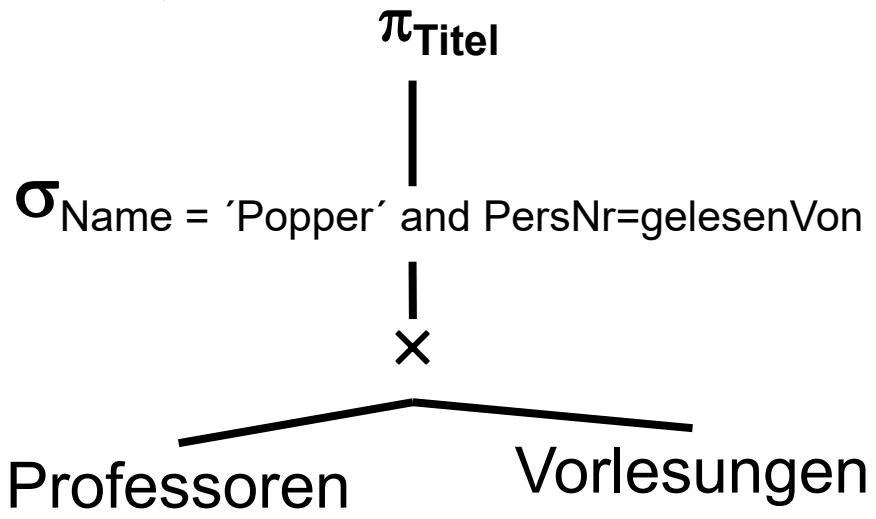


Kanonische Übersetzung

select Titel

from Professoren, Vorlesungen

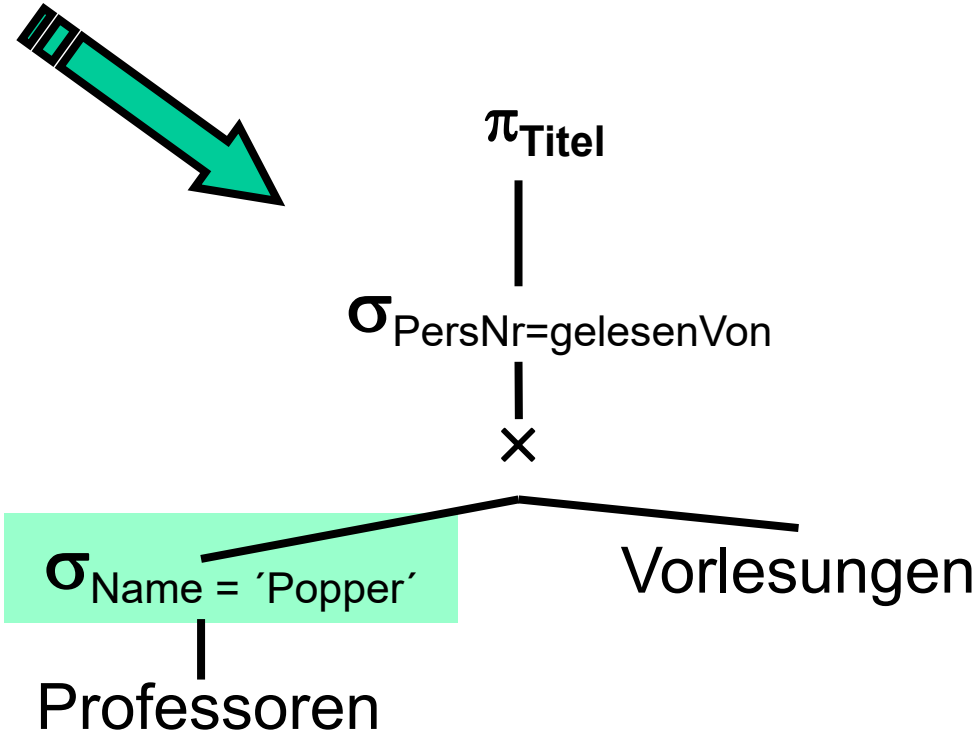
where Name = ´Popper´ **and**
PersNr = gelesenVon



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

Erste Optimierungsidee

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" (N^2!!-ohne Skalierung)

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:

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

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 Umformung:

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

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:

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

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:

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

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

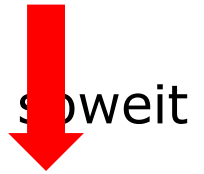
12. Ein kartesisches Produkt, das von einer Selektions-Operation gefolgt wird, deren Selektionsprädikat Attribute aus beiden Operanden des kartesischen Produktes enthält, kann in eine Joinoperation umgeformt werden.

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

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

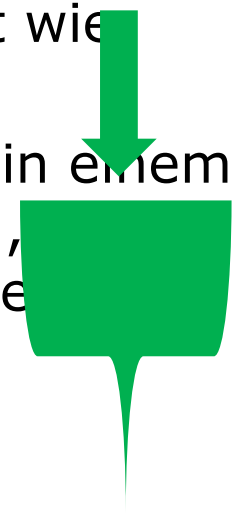
Heuristische Anwendung der Transformationsregeln

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.

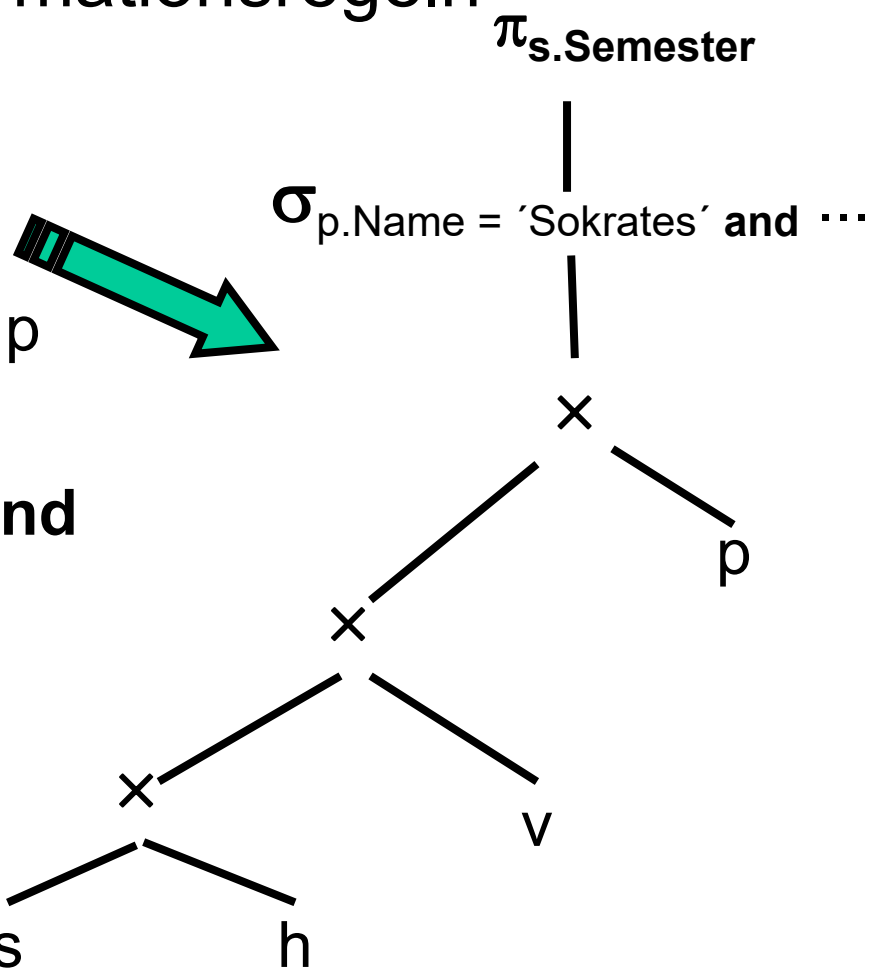
4. Forme eine x-Operation, die von einer σ-Operation gefolgt wird, wenn möglich in eine $\bowtie$-Operation um.
5. Mittels Regeln 3, 4, 7, und 10 werden Projektionen soweit wie möglich nach unten propagiert.



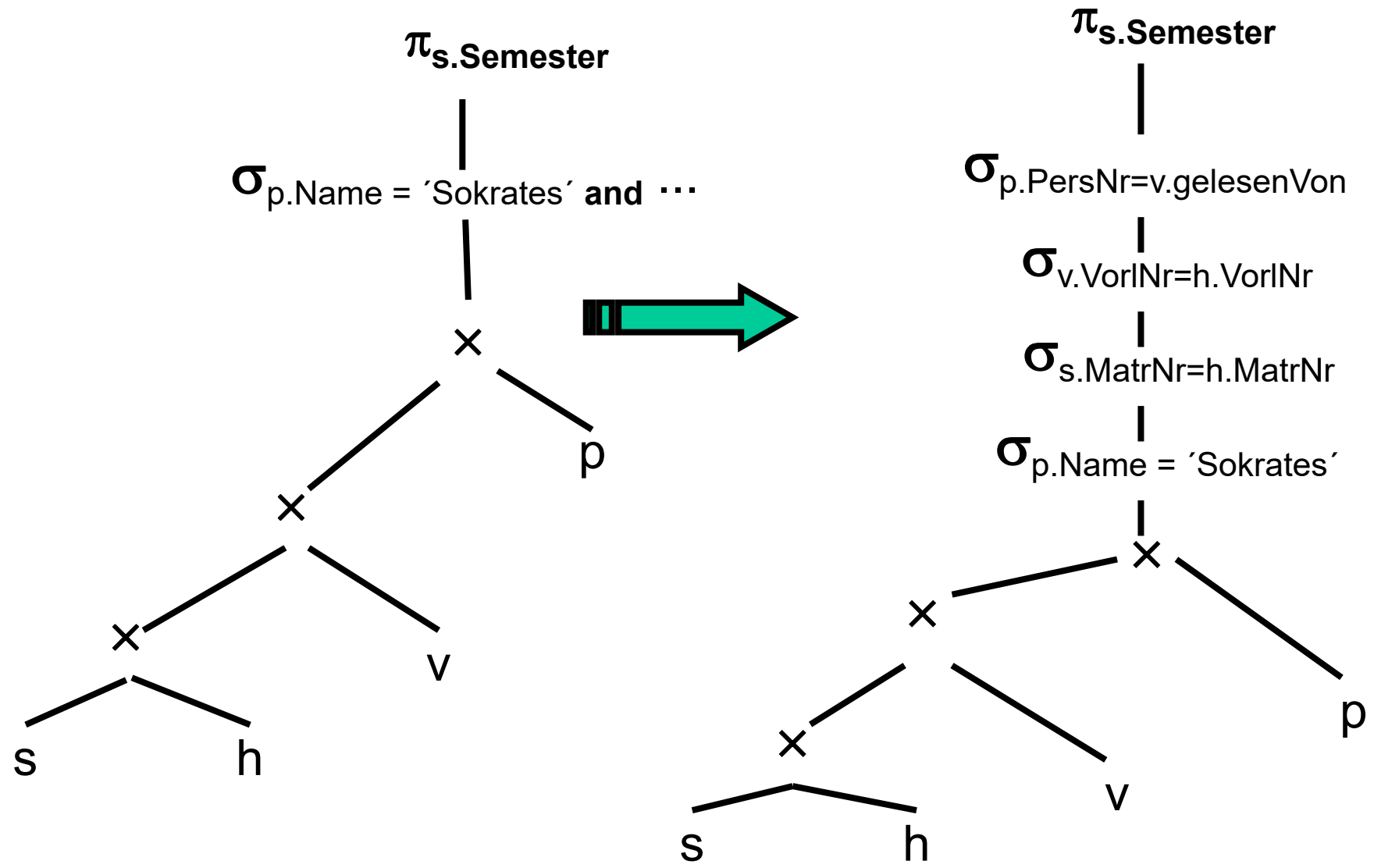
6. Versuche Operationsfolgen zusammenzufassen, wenn sie in einem „Durchlauf“ ausführbar sind (z.B. Anwendung von Regel 1, 3, aber auch Zusammenfassung aufeinanderfolgender Selektionen und Projektionen zu einer „Filter“-Operation).

Anwendung der Transformationsregeln

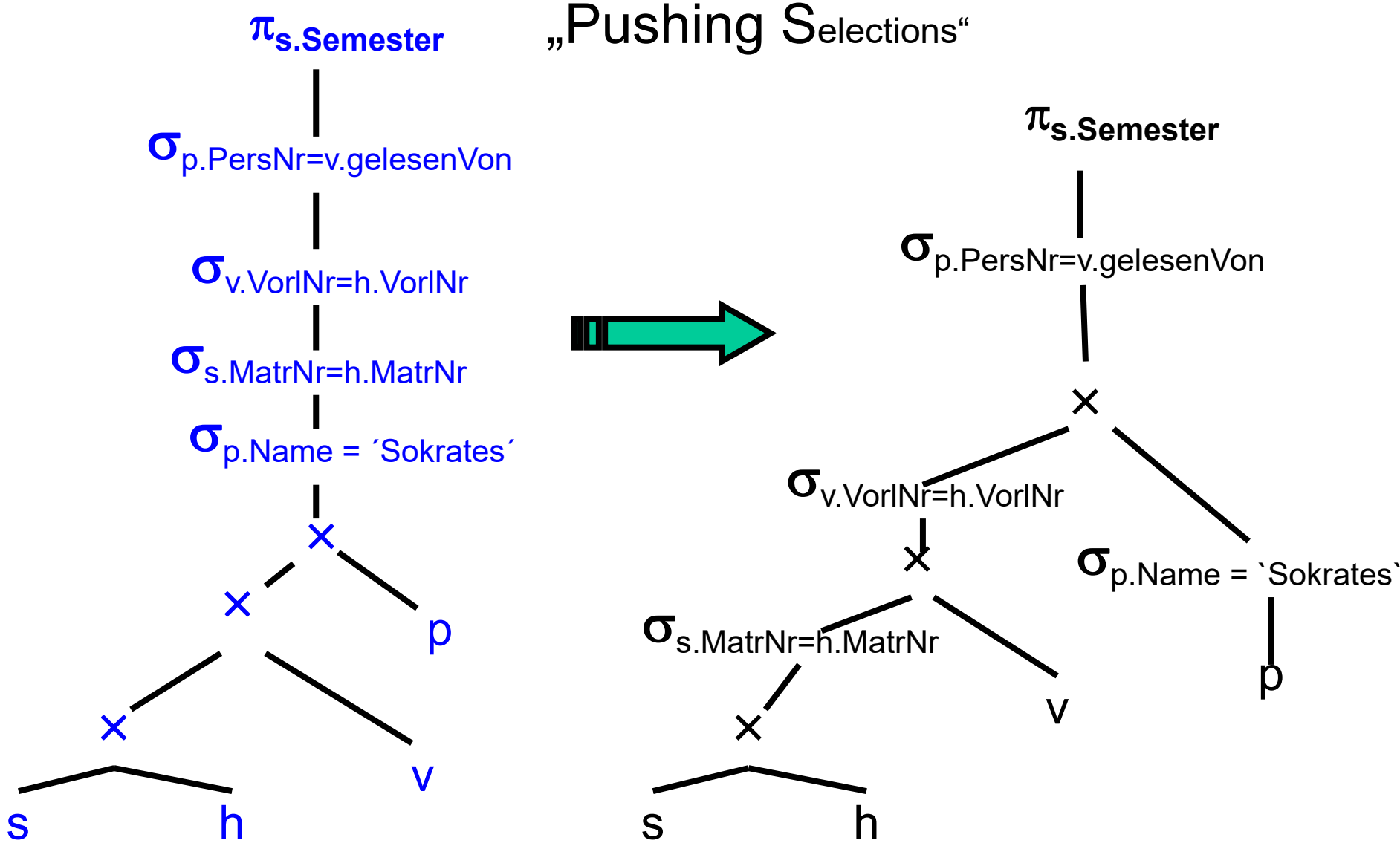
```
select distinct s.Semester
from Studenten s, hören h
        Vorlesungen v, Professoren p
where p.Name = ´Sokrates´ and
        v.gelesenVon = p.PersNr and
        v.VorlNr = h.VorlNr and
        h.MatrNr = s.MatrNr
```



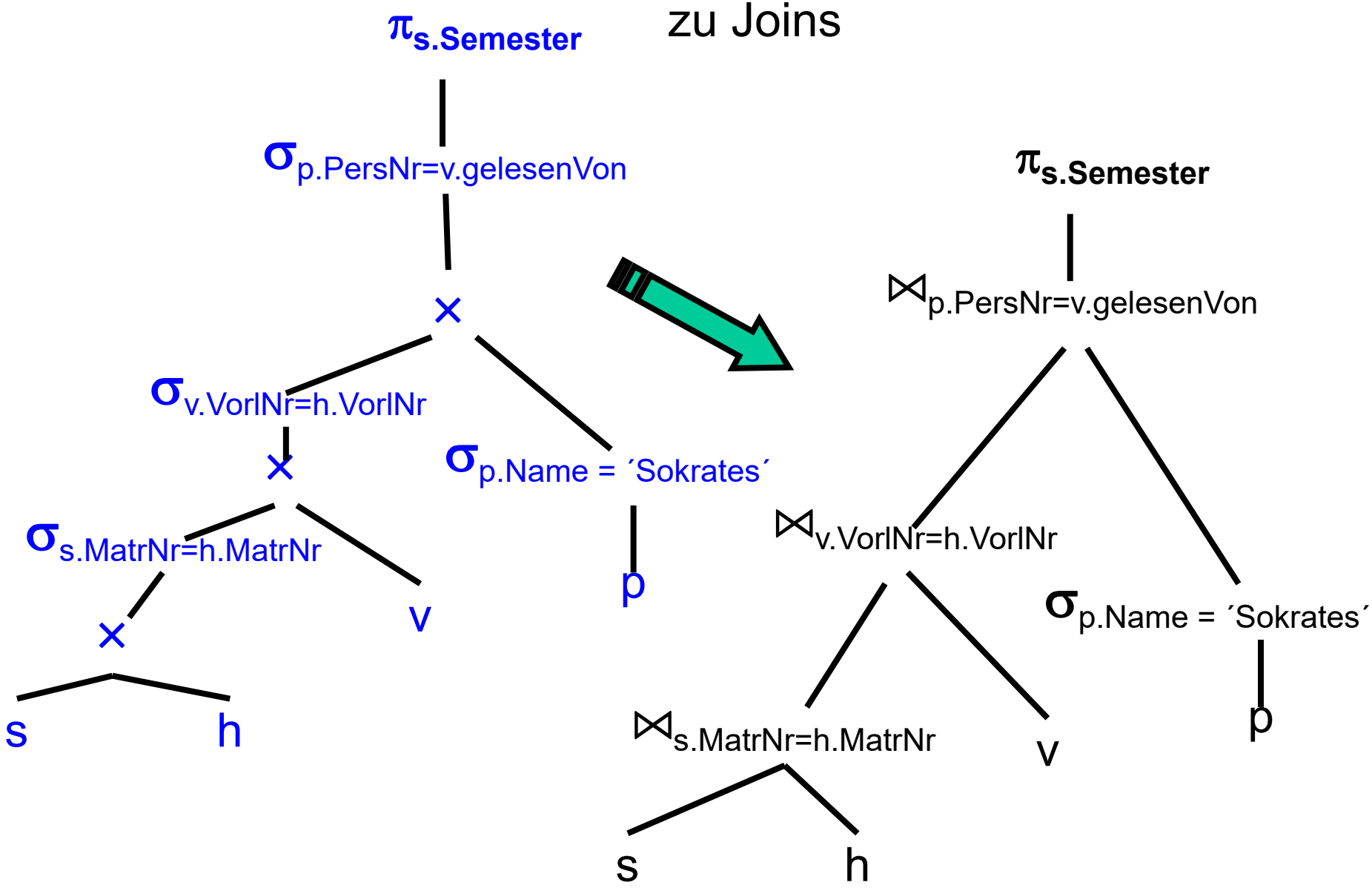
Aufspalten der Selektionsprädikate



Verschieben der Selektionsprädikate „Pushing Selections“

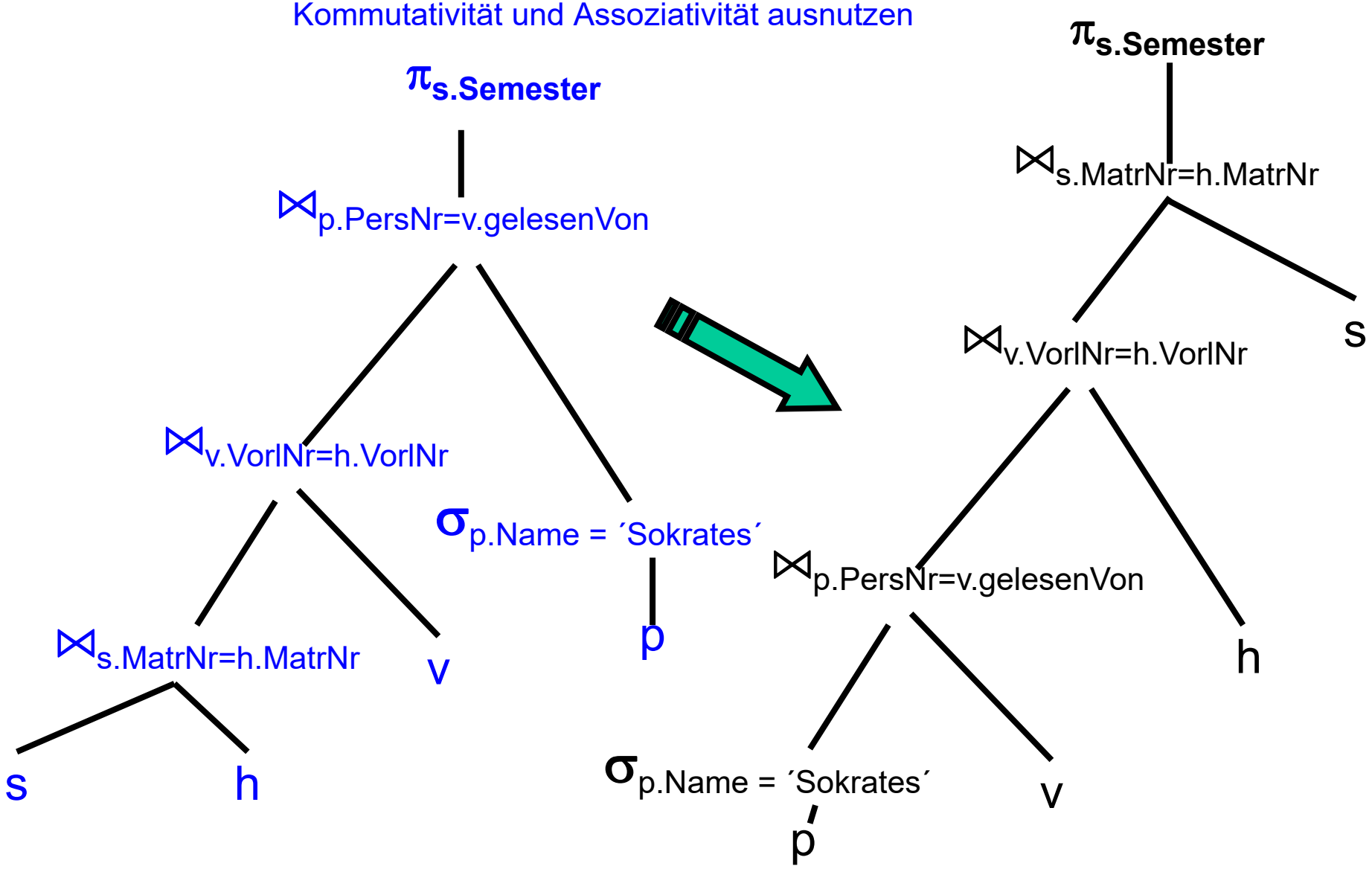


Zusammenfassung von Selektionen und Kreuzprodukten zu Joins

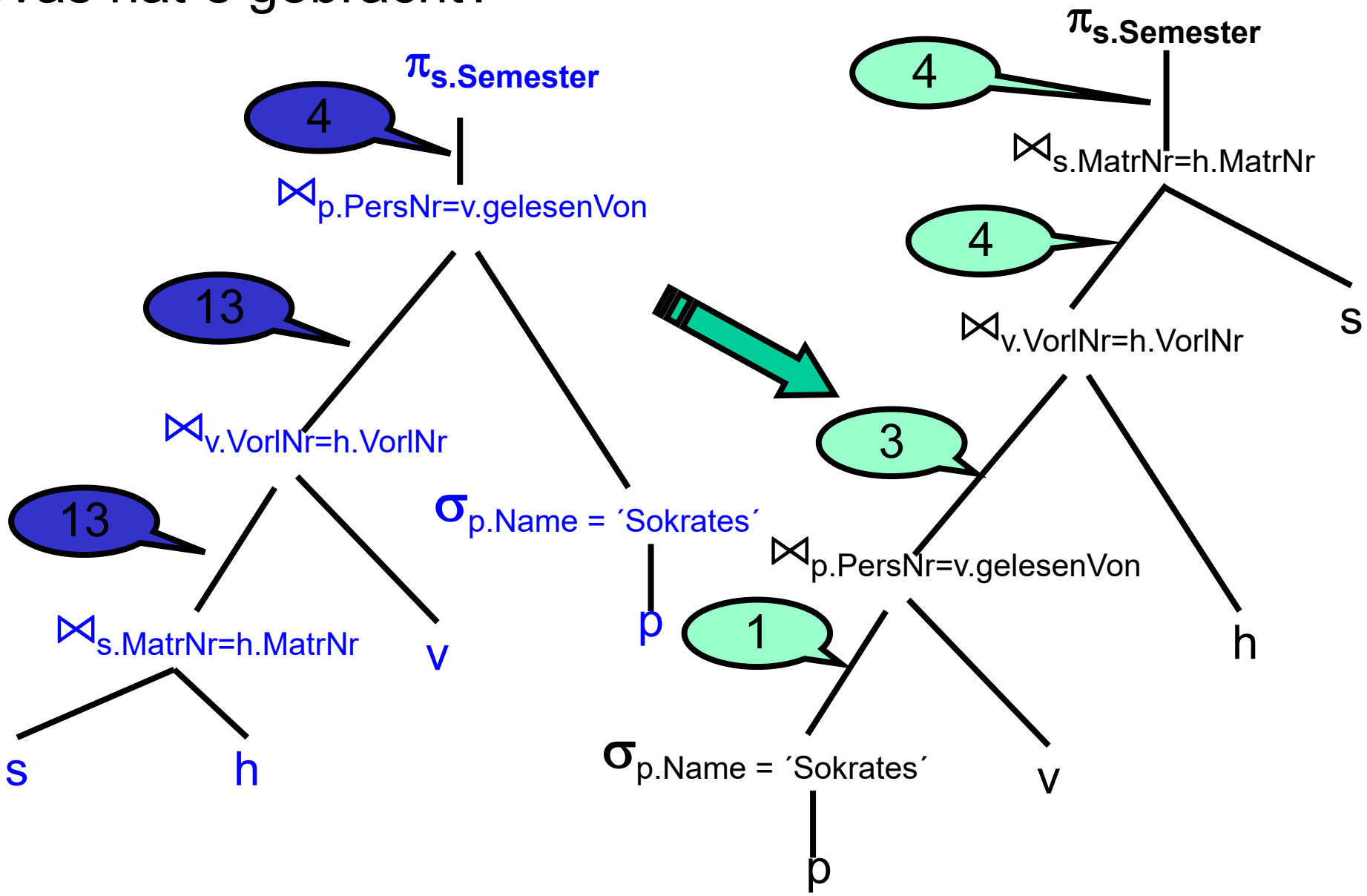


Optimierung der Joinreihenfolge

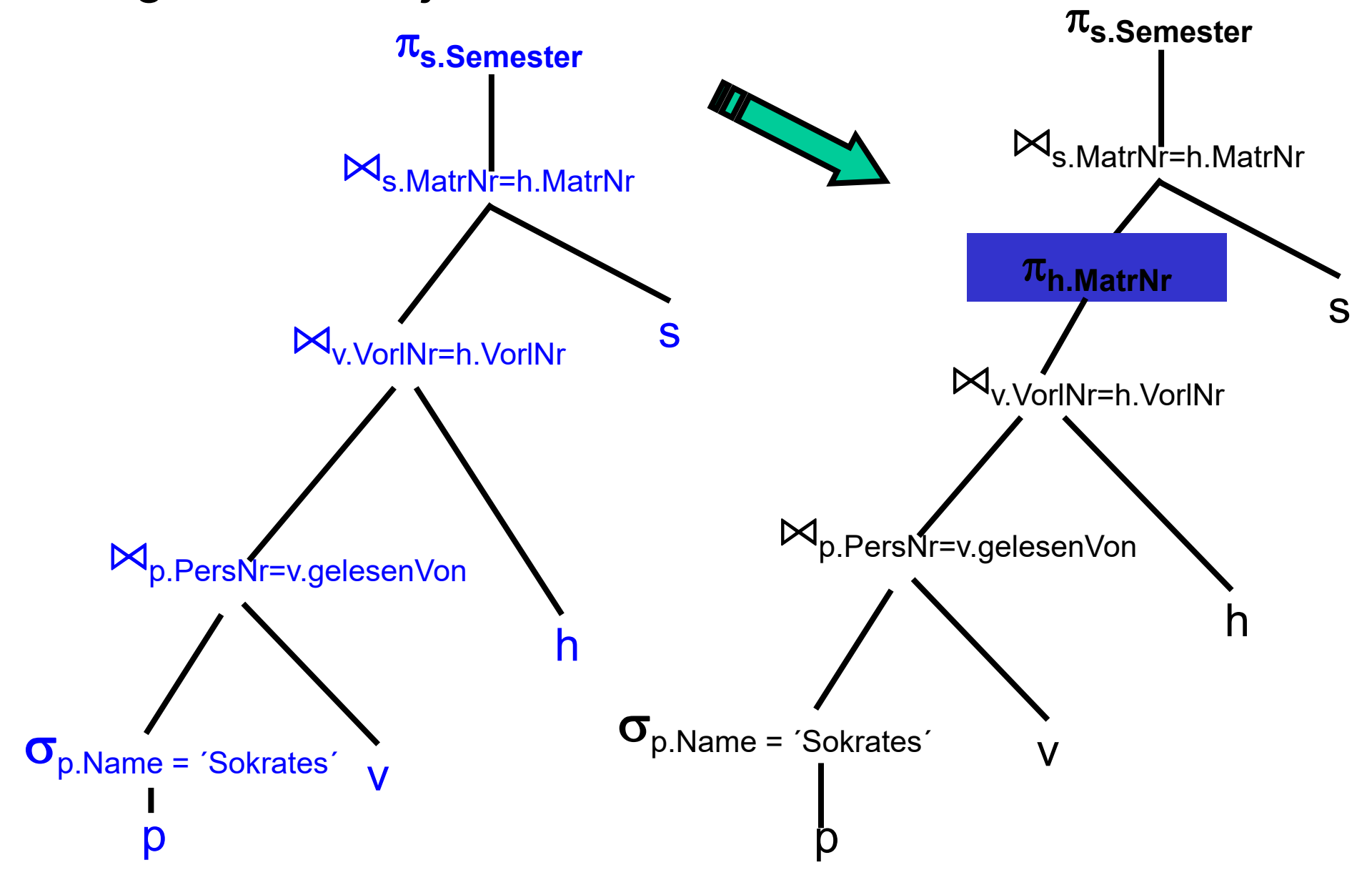
Kommutativität und Assoziativität ausnutzen



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'
```

10^13

10^4

10^5

10^4

Airport p

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

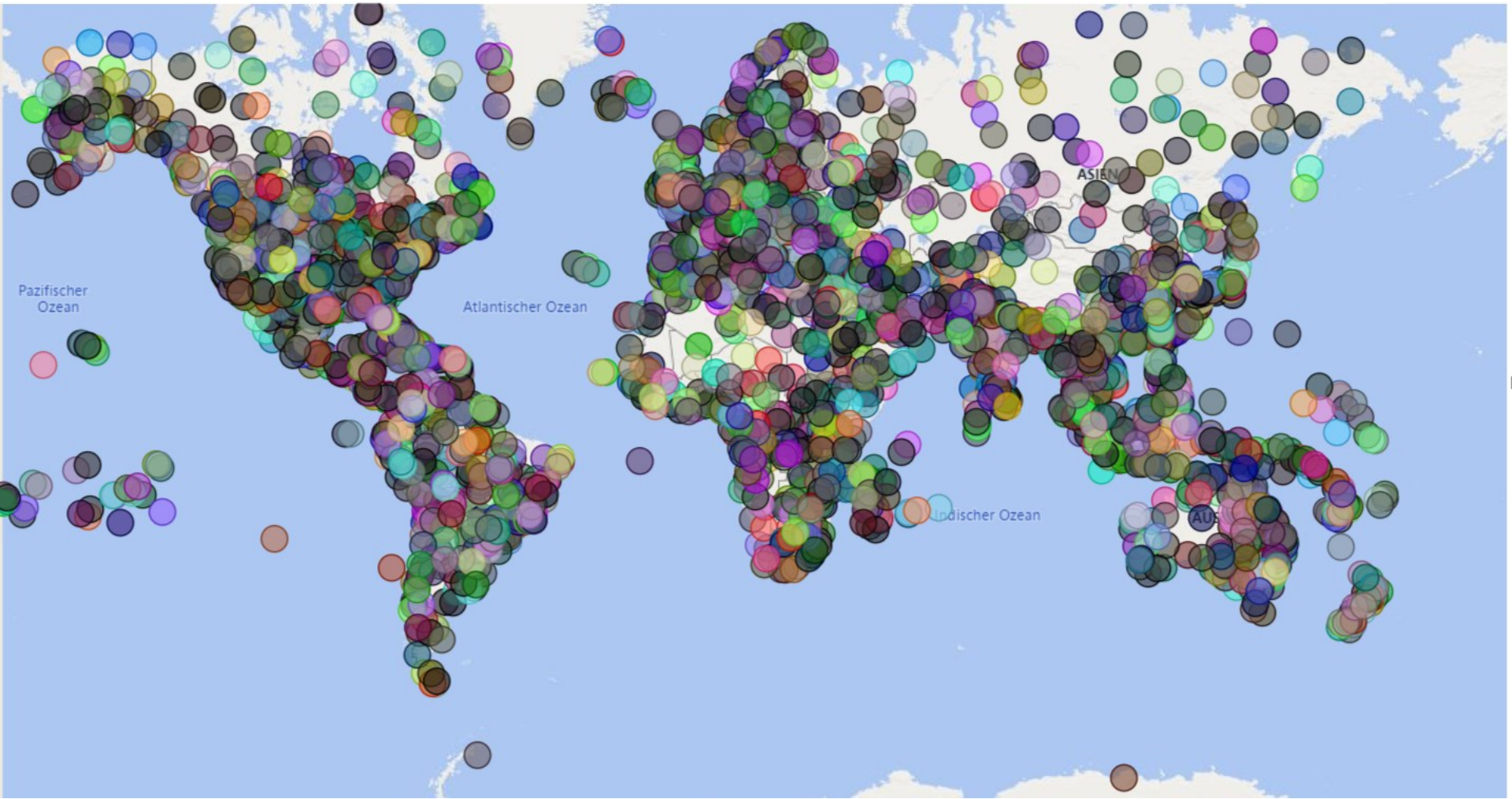
Connection c

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

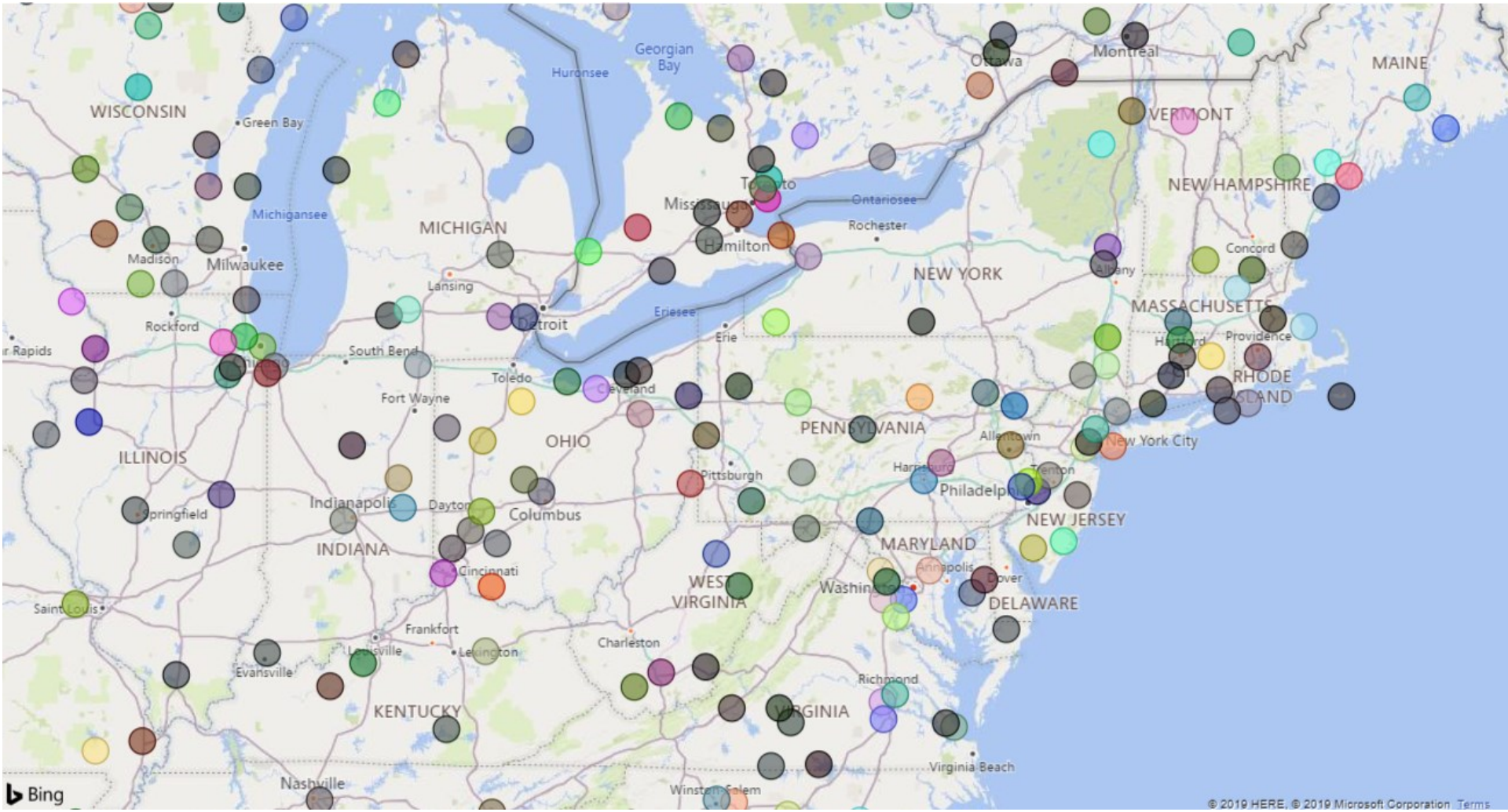
Airport n

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

Count(Airports) = 7128

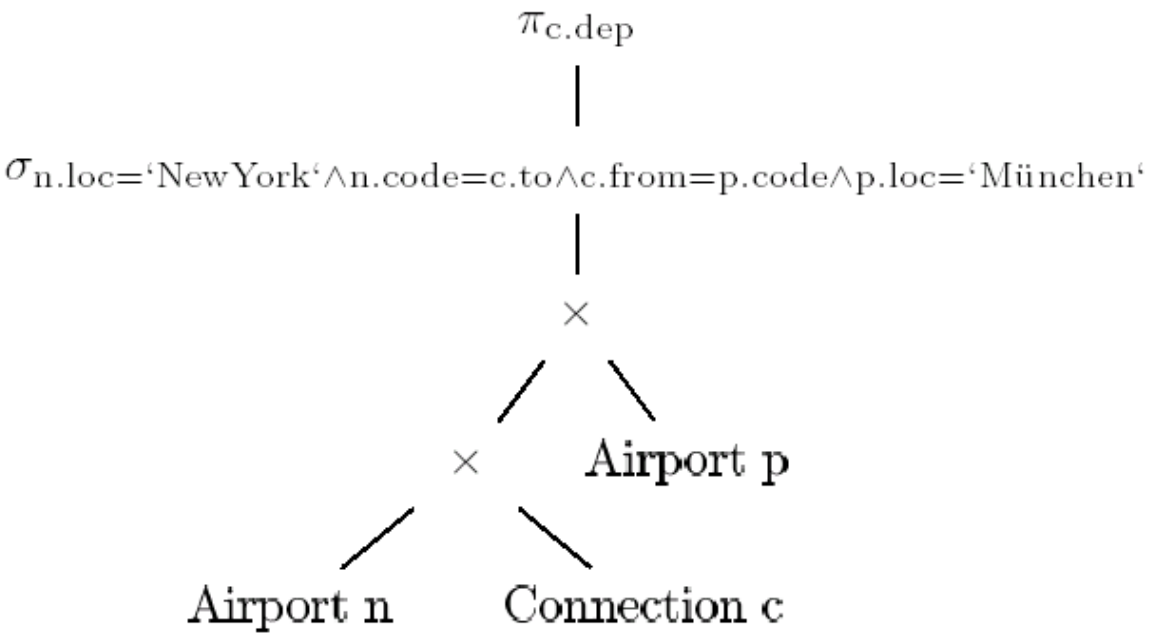


Airports


WISCONSIN
Green Bay
Michigansee
Madison
Milwaukee
Rockford
Huronsee
Georgian Bay
MICHIGAN
Lansing
Detroit
Toledo
South Bend
Fort Wayne
ILLINOIS
Springfield
Indianapolis
Dayton
Columbus
INDIANA
Cincinnati
OHIO
Pittsburgh
Eriesee
Erie
Ontariosee
Rochester
Hamilton
Ottawa
Montreal
NEW YORK
Albany
VERMONT
NEW HAMPSHIRE
MAINE
Concord
MASSACHUSETTS
Providence
RHODE ISLAND
New York City
Allentown
Philadelphia
NEW JERSEY
PENNSYLVANIA
MARYLAND
DELAWARE
WEST VIRGINIA
Charleston
Frankfort
Lexington
Evansville
KENTUCKY
Nashville
Richmond
Virginia Beach
Bing
© 2019 HERE, © 2019 Microsoft Corporation Terms

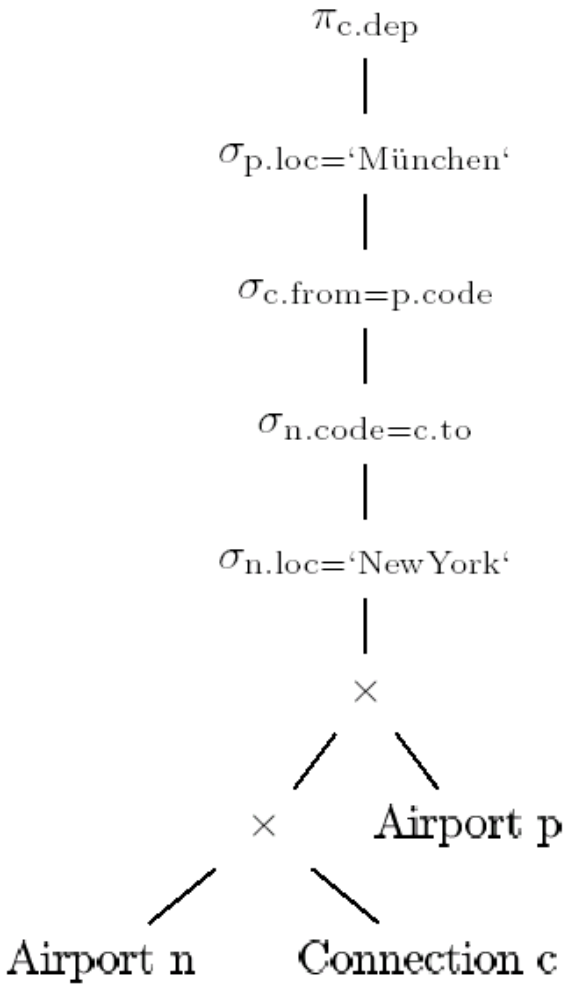
Kanonische Übersetzung

```
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'
```



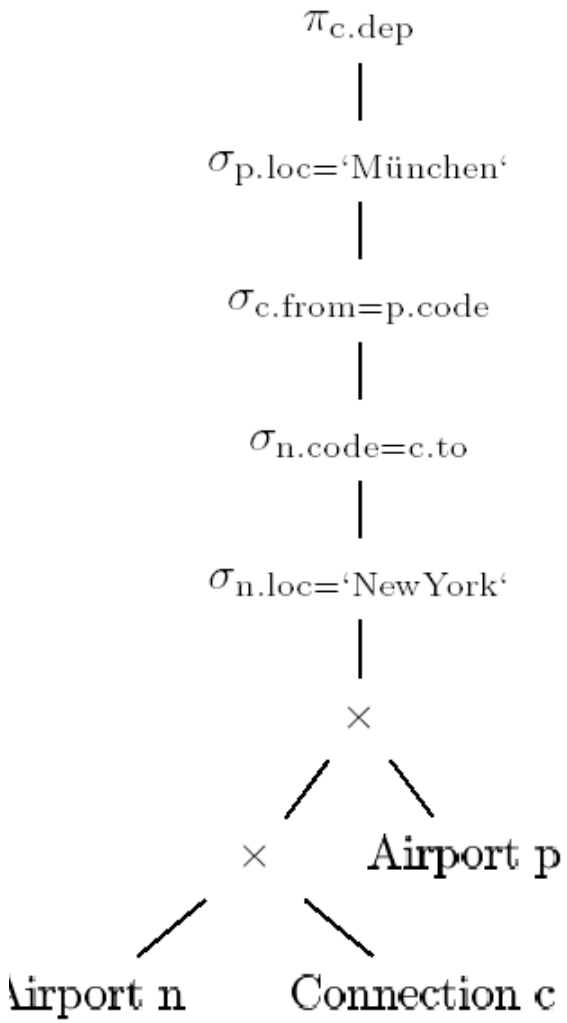
Selektionsprädikate „aufbrechen“

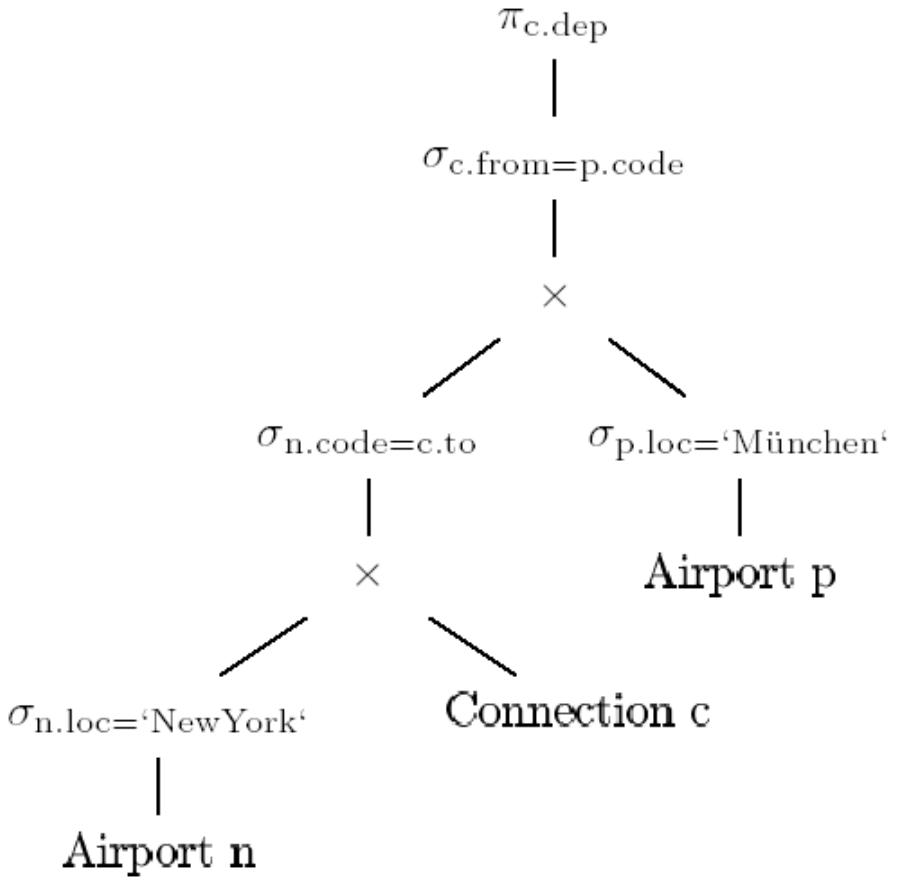
$$\sigma_{p_1 \wedge p_2 \wedge \ldots \wedge p_n}(R) = \sigma_{p_1}(\sigma_{p_2}(\ldots(\sigma_{p_n}(R))\ldots))$$



„Pushing Selections“

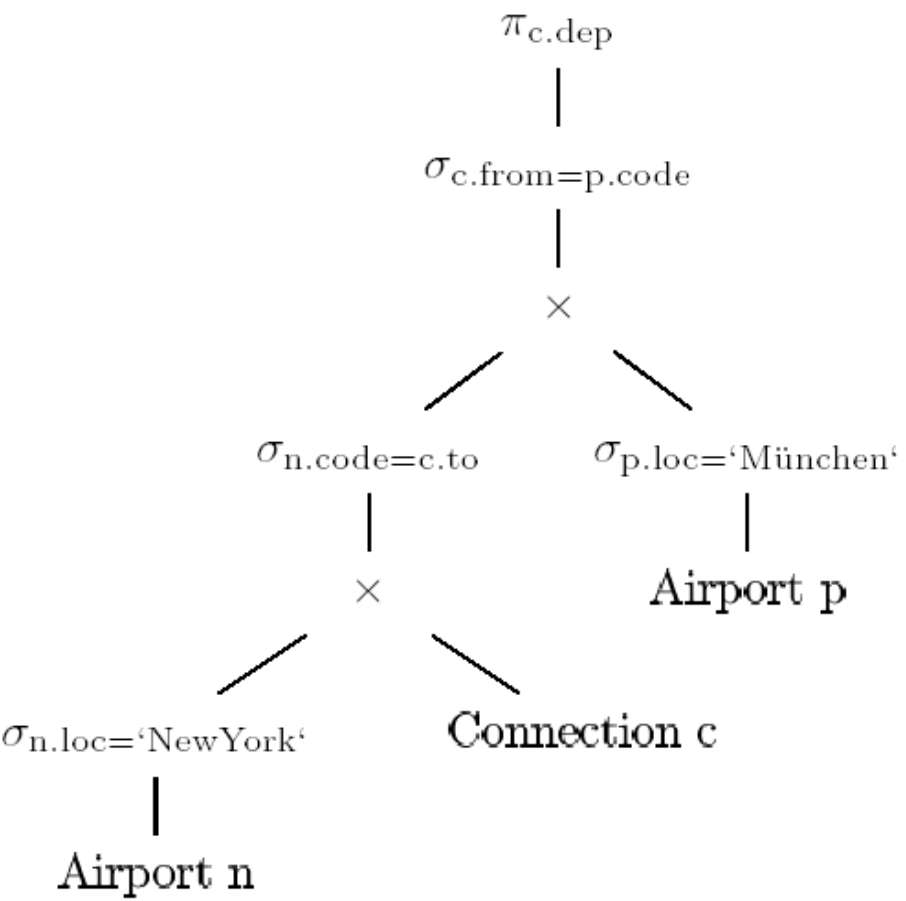
$$
\begin{aligned}
\sigma_p(\sigma_q(R)) &= \sigma_q(\sigma_p(R)) \\
\sigma_p(R_1 \bowtie R_2) &= \sigma_p(R_1) \bowtie R_2 \\
\sigma_p(R_1 \times R_2) &= \sigma_p(R_1) \times R_2
\end{aligned}
$$

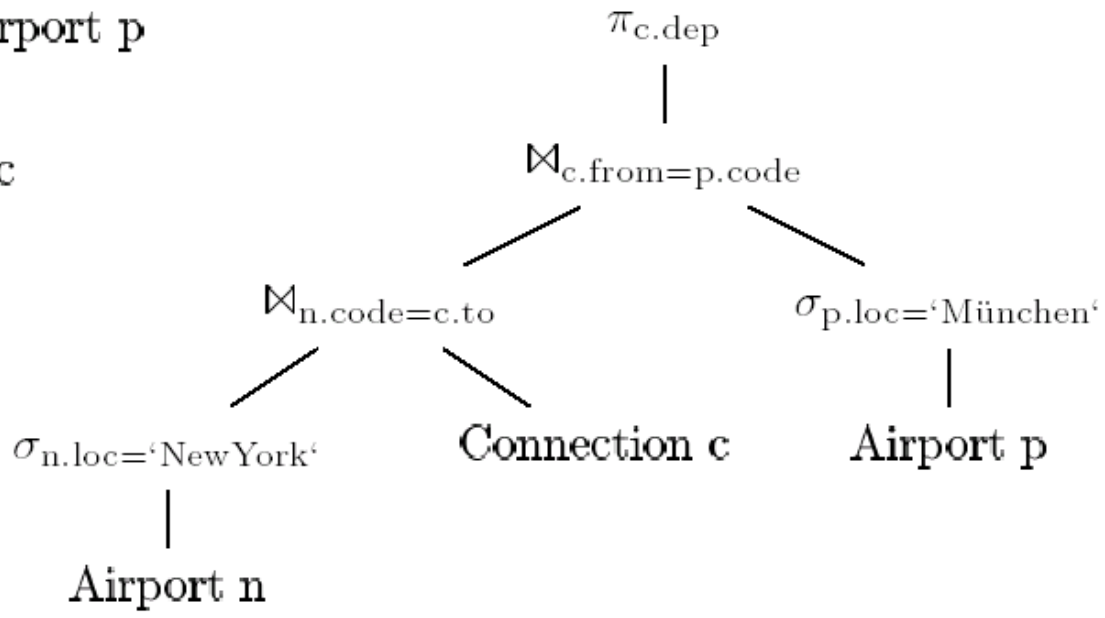




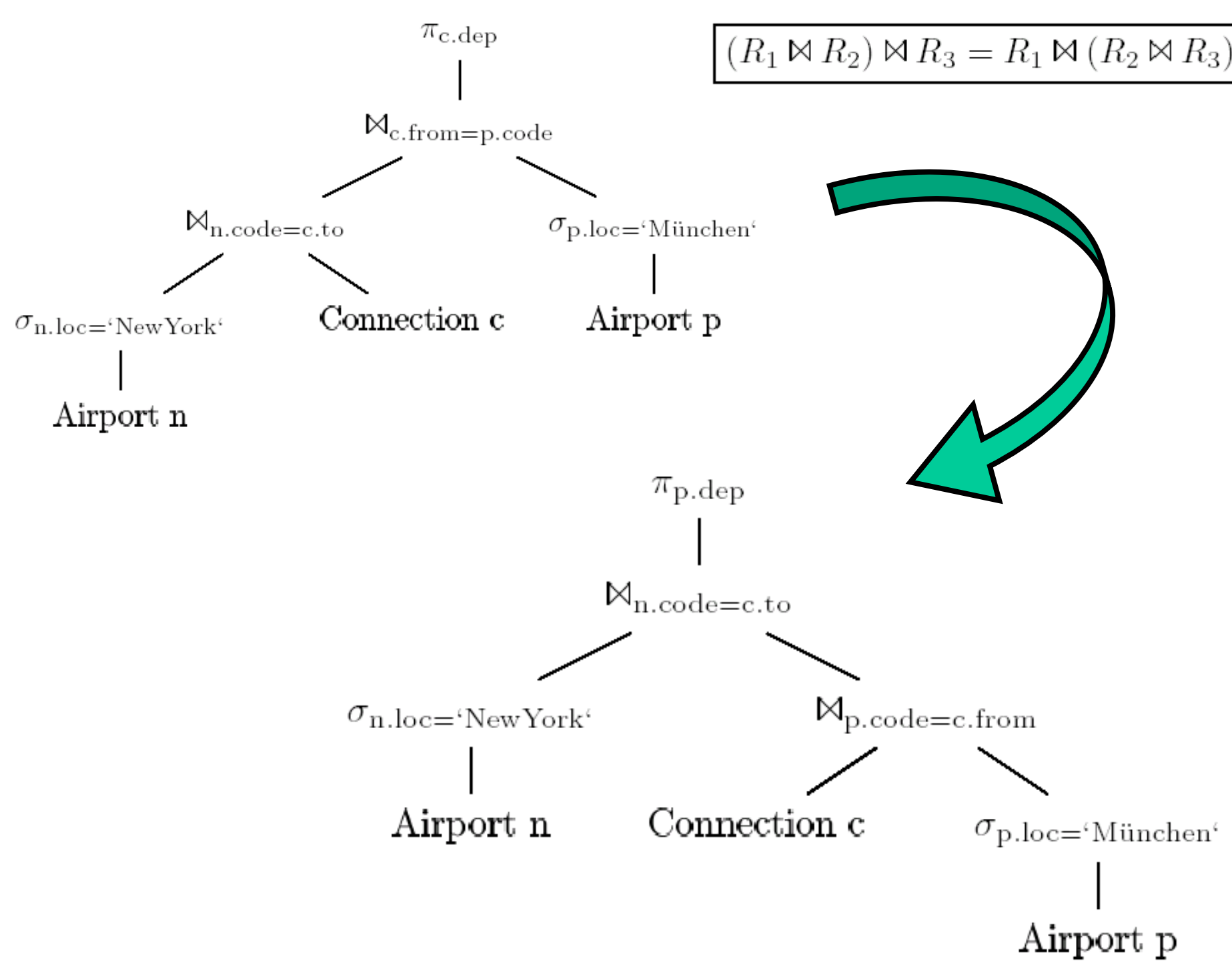
Zusammenfassung von $\sigma \times$ zu $\bowtie$

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



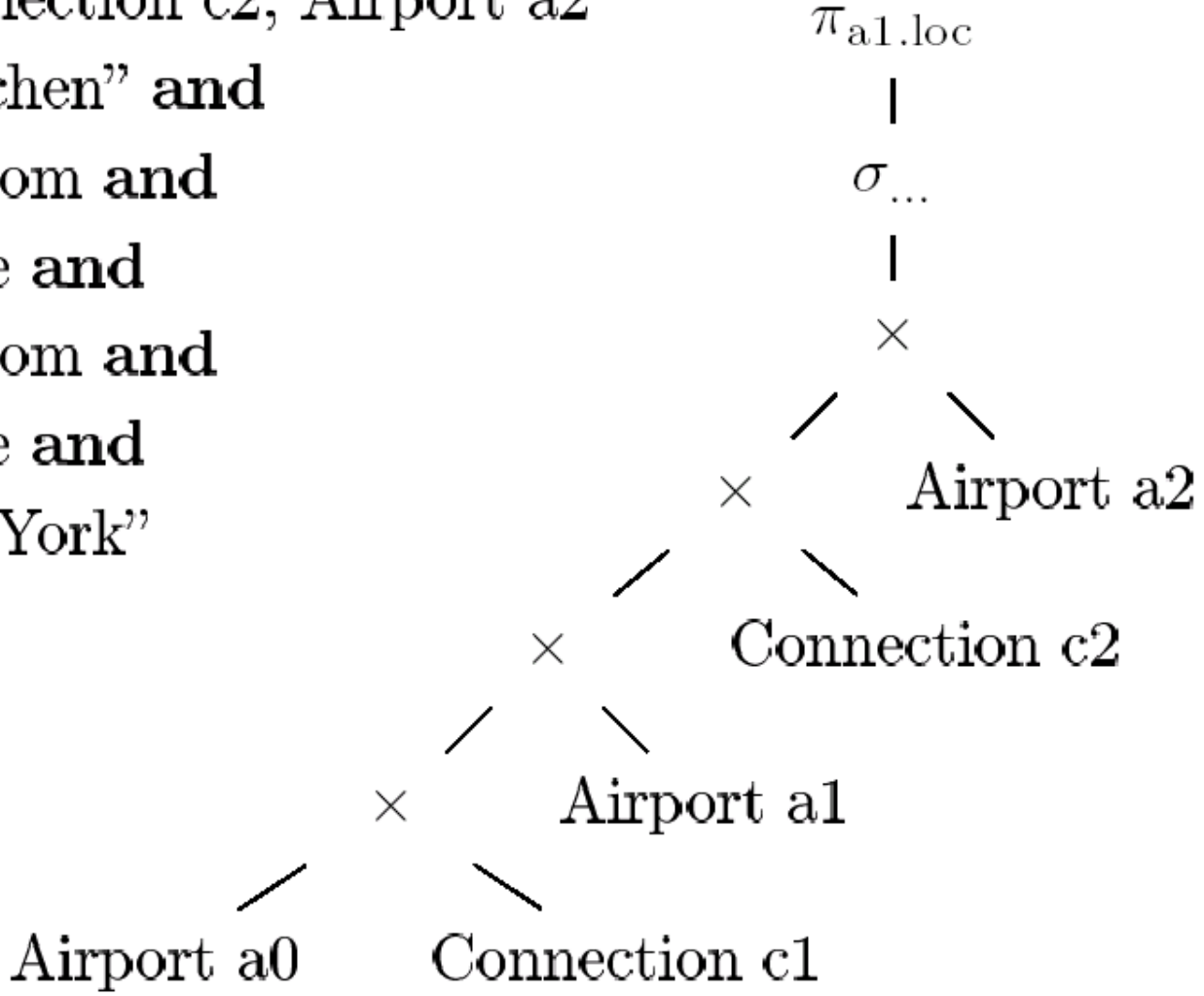


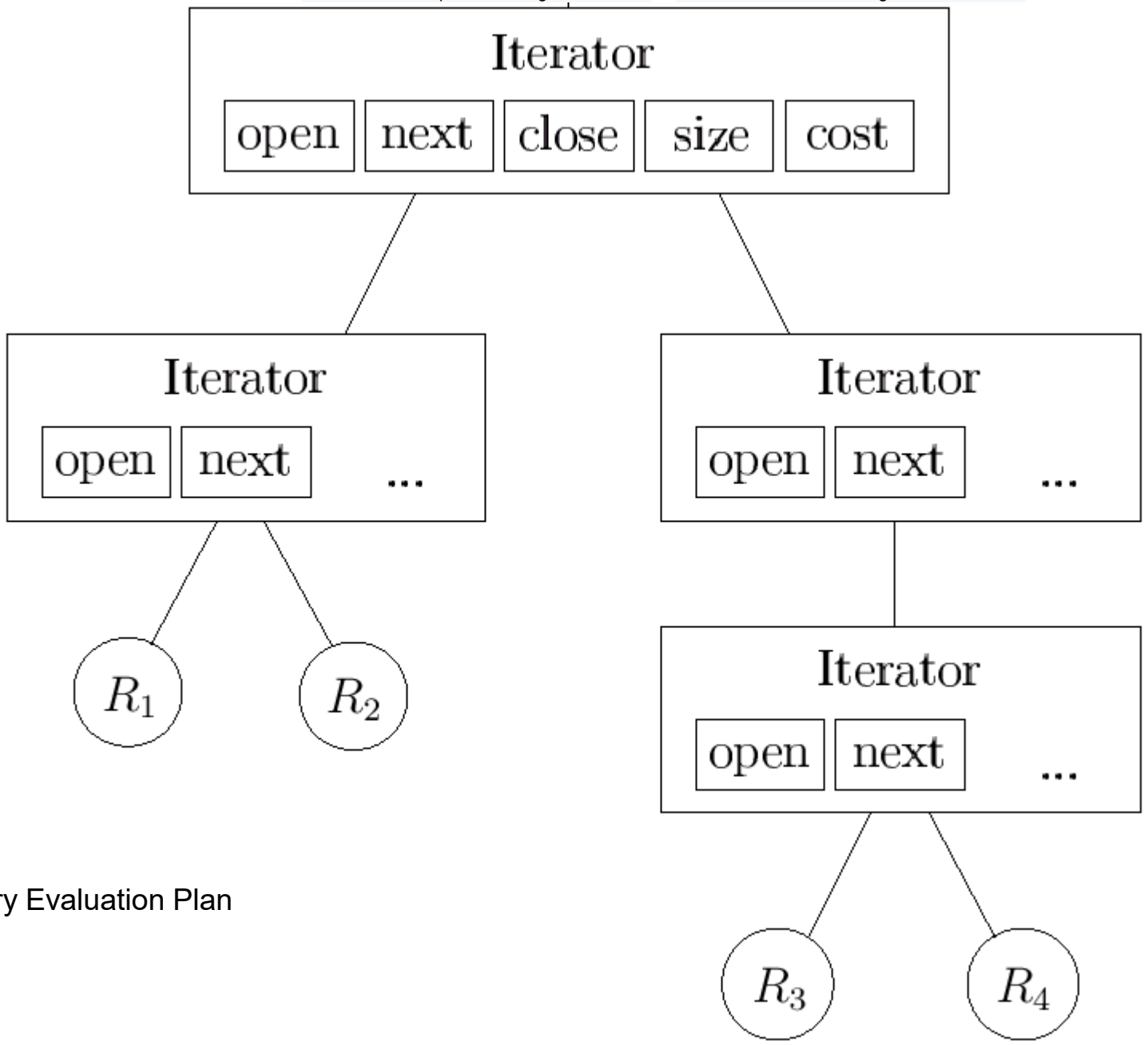
$$(R_1 \bowtie R_2) \bowtie R_3 = R_1 \bowtie (R_2 \bowtie R_3)$$



MUC → NY mit genau einmal Umsteigen

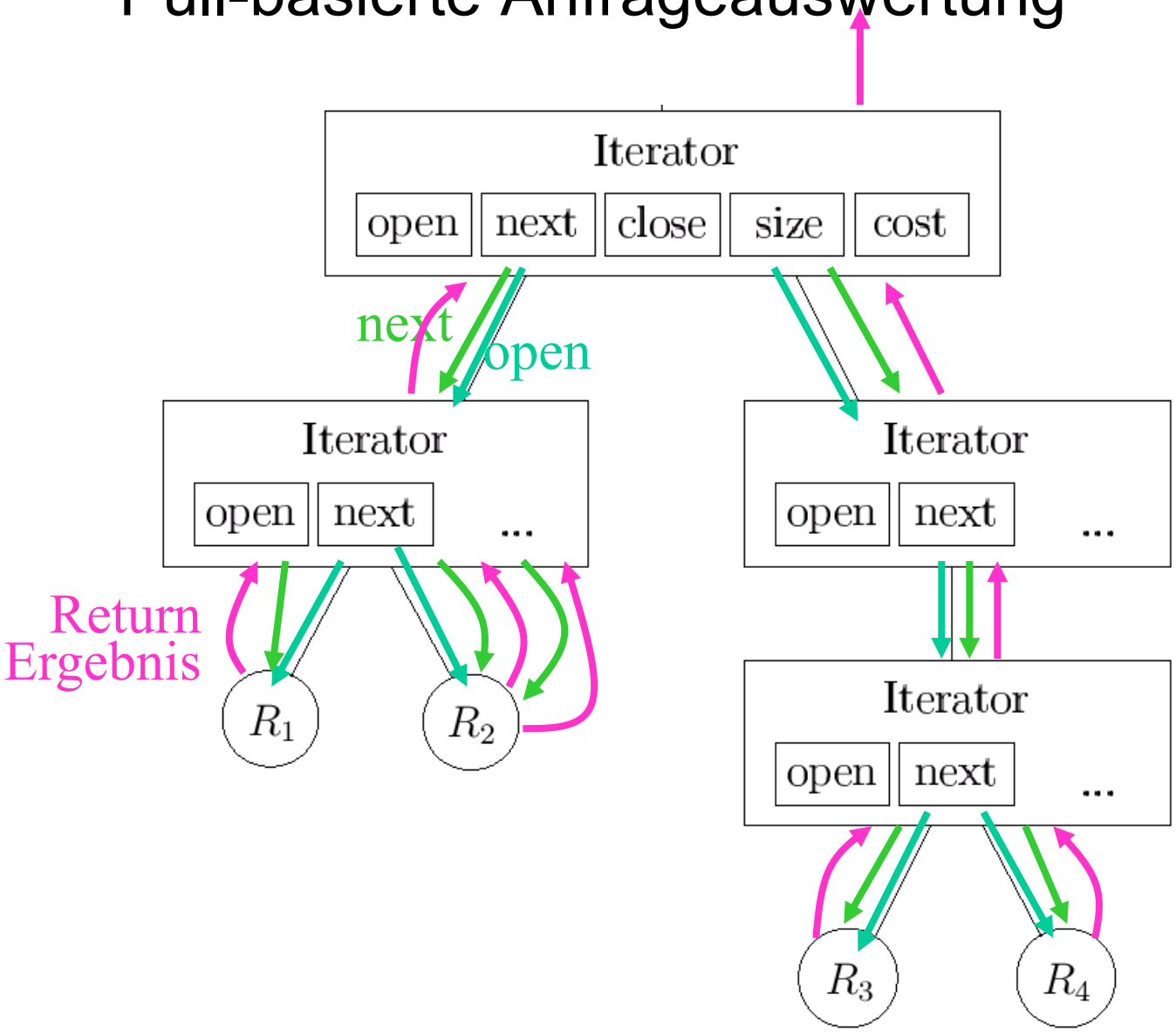
select a1.loc
from Airport a0, Connection c1,
Airport a1, Connection c2, Airport a2
where a0.loc = "München" **and**
a0.code = c1.from **and**
c1.to = a1.code **and**
a1.code = c2.from **and**
c2.to = a2.code **and**
a2.loc = "New York"



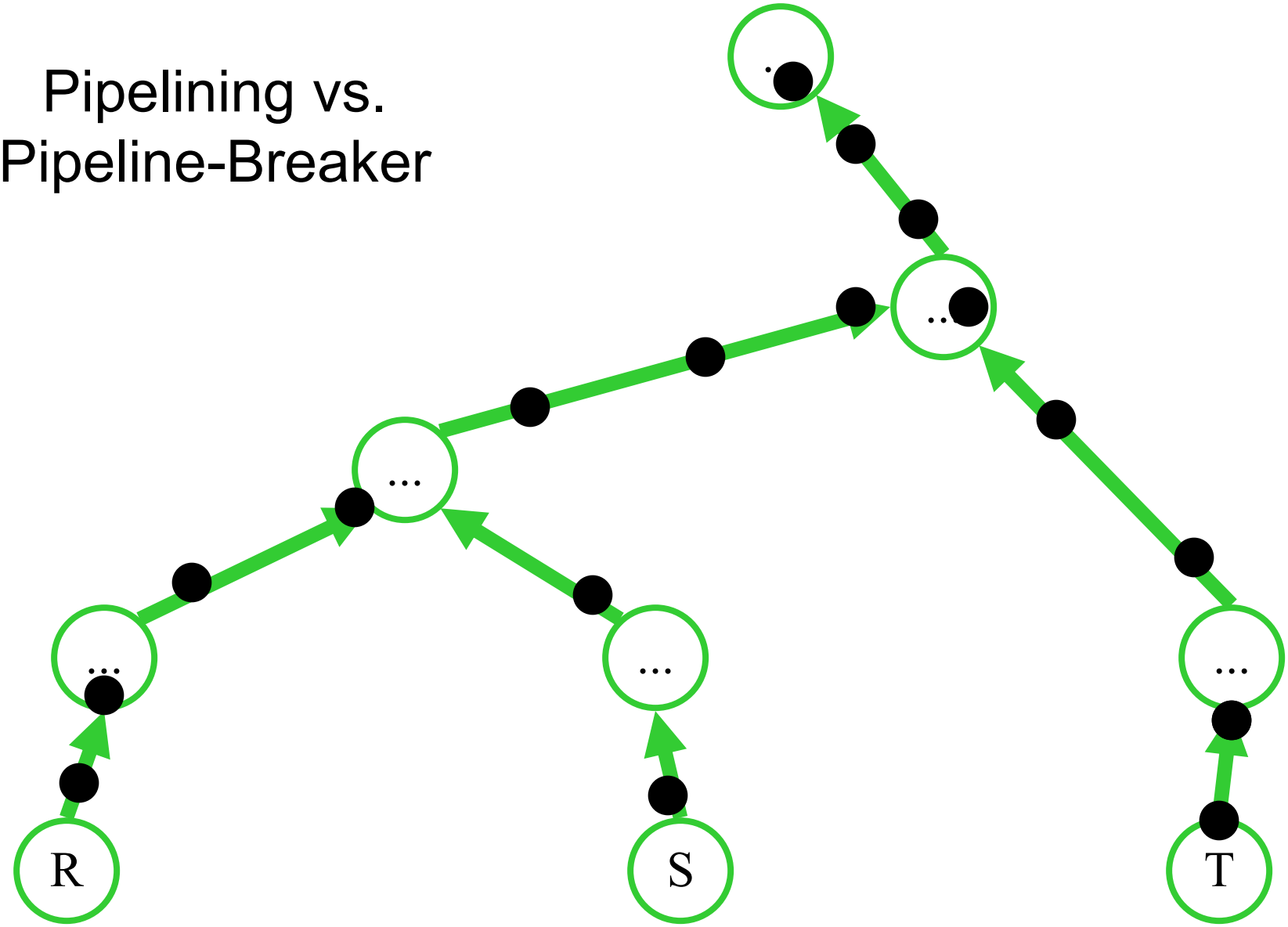


QEP
Query Evaluation Plan

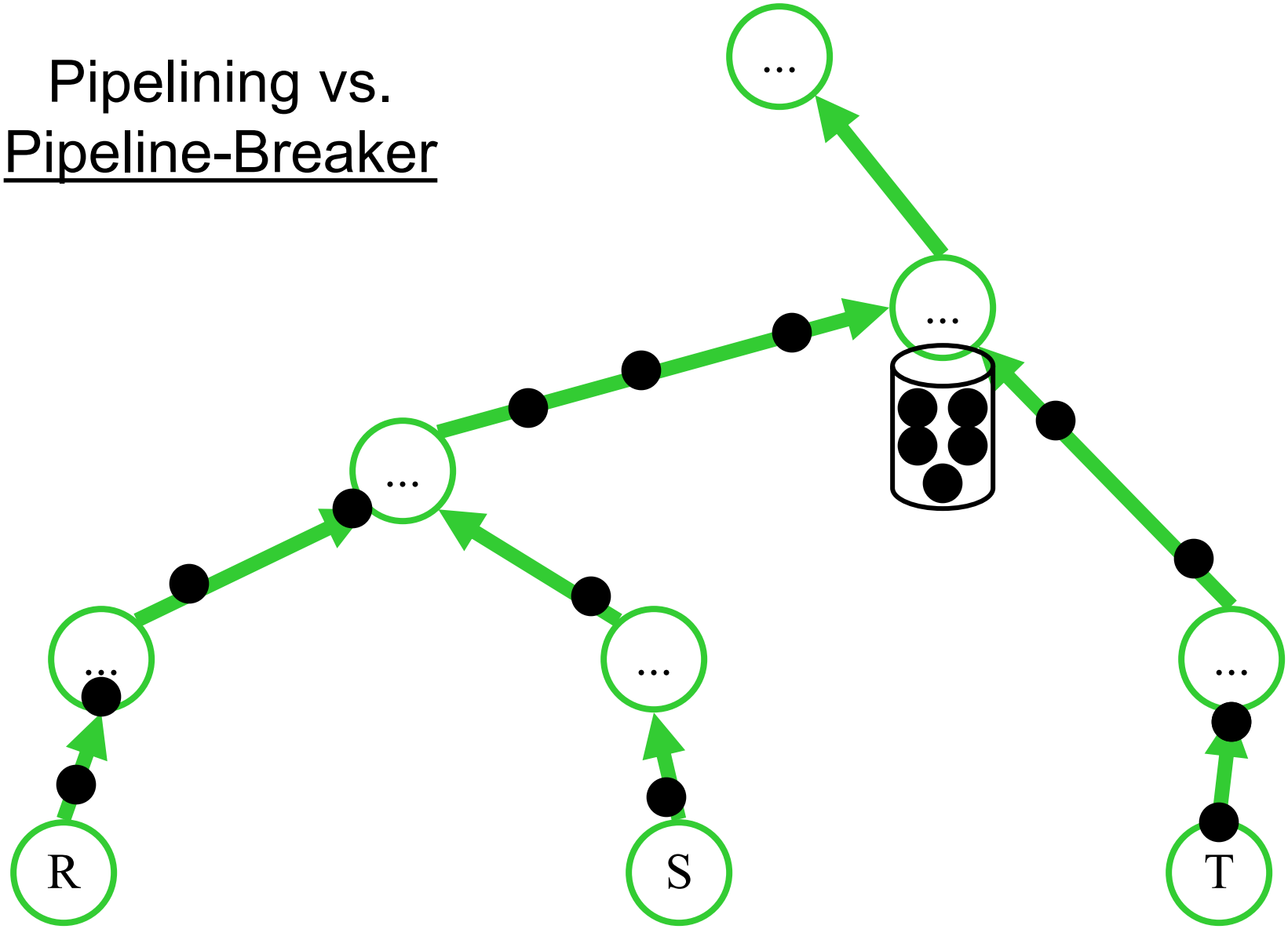
Pull-basierte Anfrageauswertung



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 der Verbindung: 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)
```

iterator NestedLoop$_p$

open

- Öffne die linke Eingabe

next

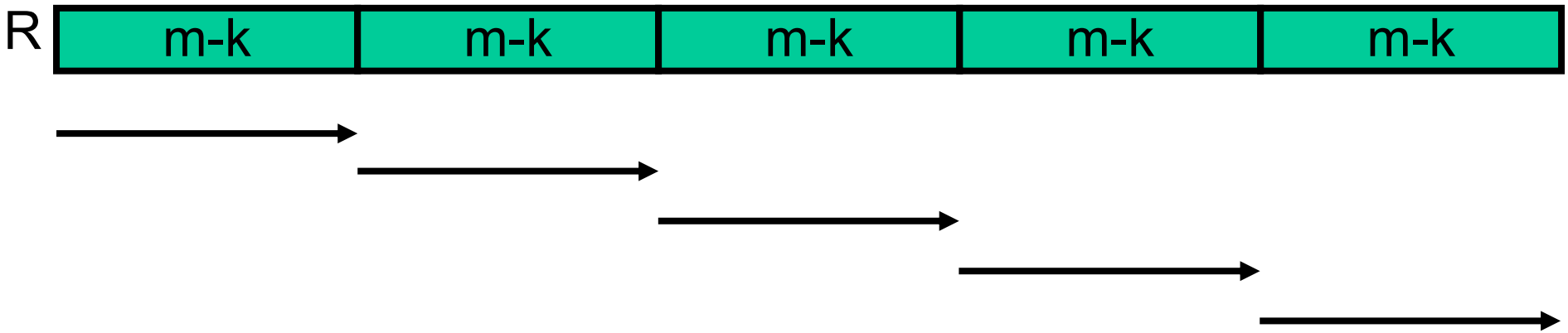
- Rechte Eingabe geschlossen?
  - Öffne sie
- Fordere rechts solange Tupel an, bis Bedingung p erfüllt ist
- Sollte zwischendurch rechte Eingabe erschöpft sein
  - Schließe rechte Eingabe
  - Fordere nächstes Tupel der linken Eingabe an
  - Starte **next** neu
- Gib den Verbund von aktuellem linken und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

close

- Schließe beide Eingabequellen

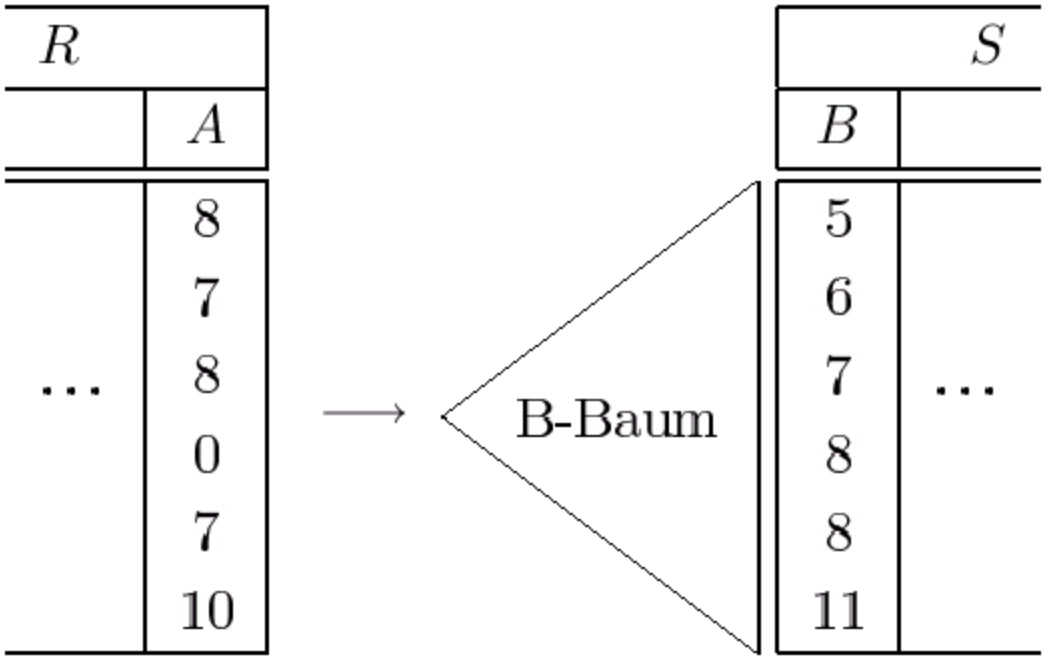
Implementierung der Verbindung: Strategien

Block-Nested Loop Algorithmus



Index-Join

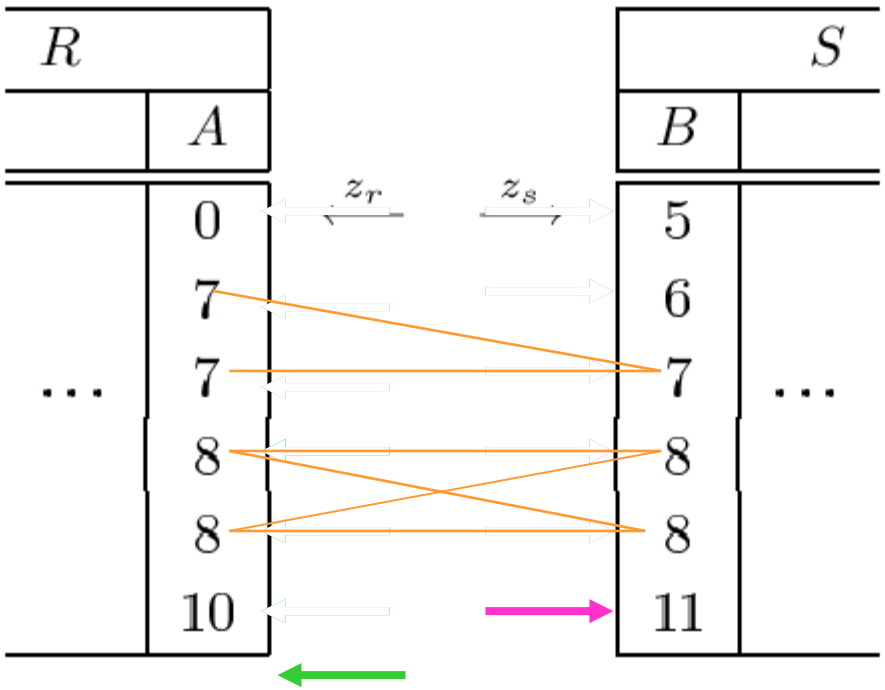
Beispiel:



Der Merge-Join

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

Beispiel:

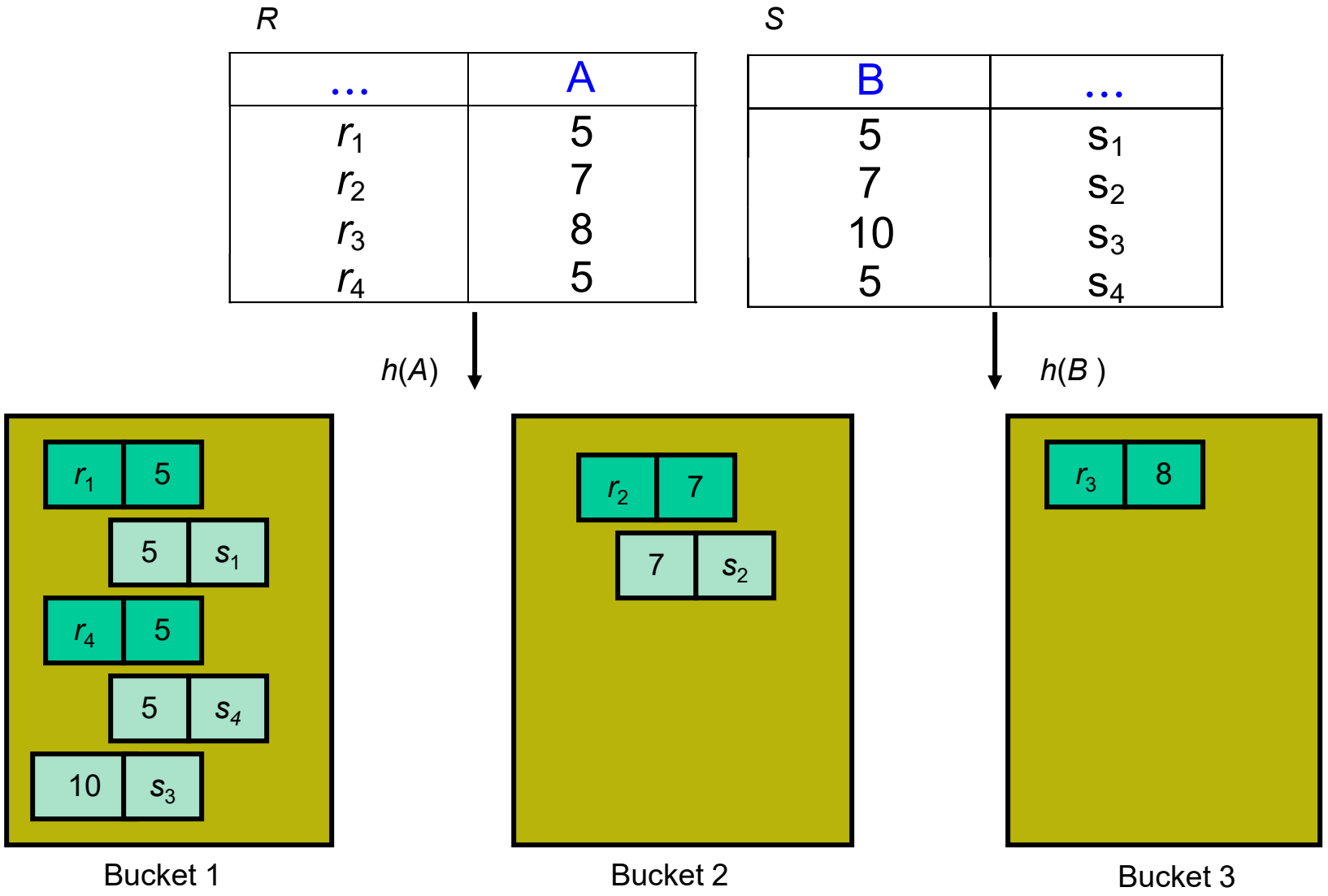


Implementierung der Verbindung: Strategien

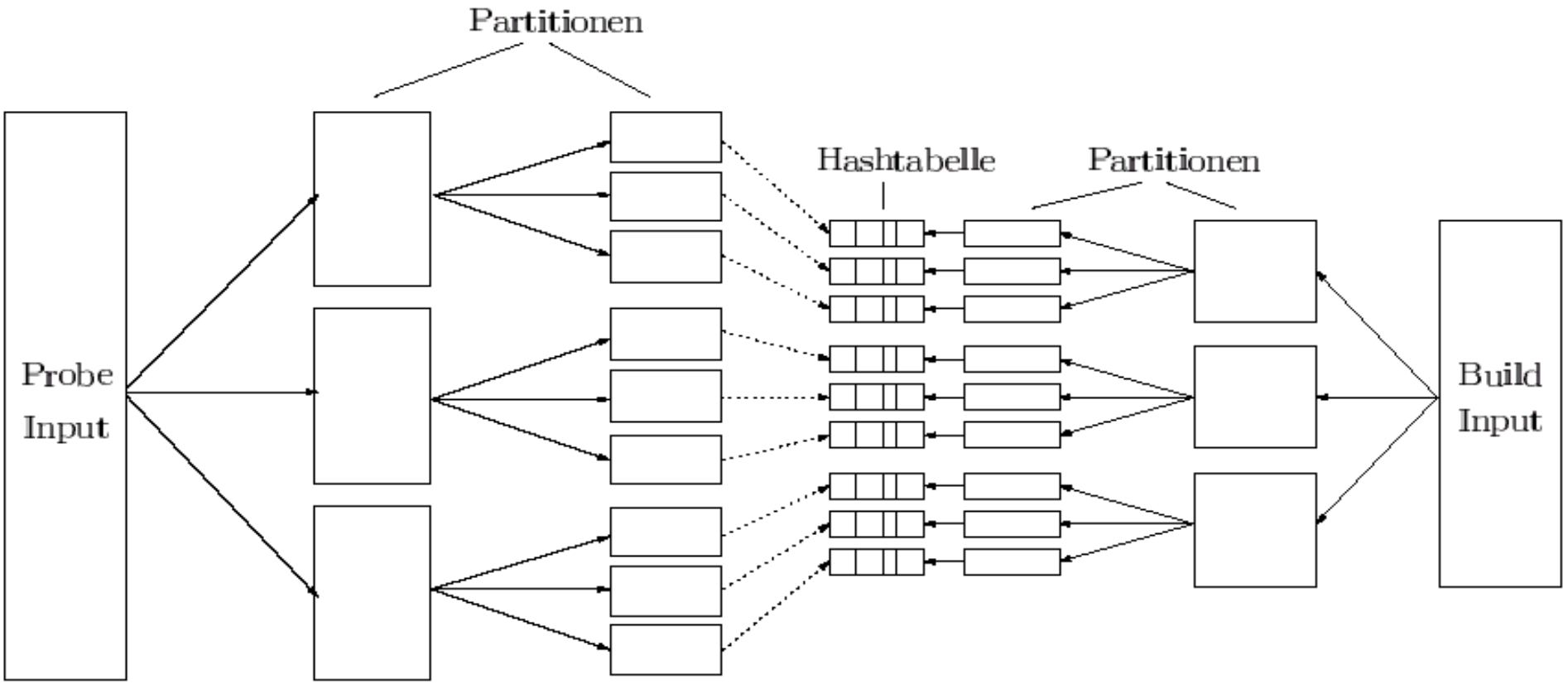
J4 Hash-Join

- *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

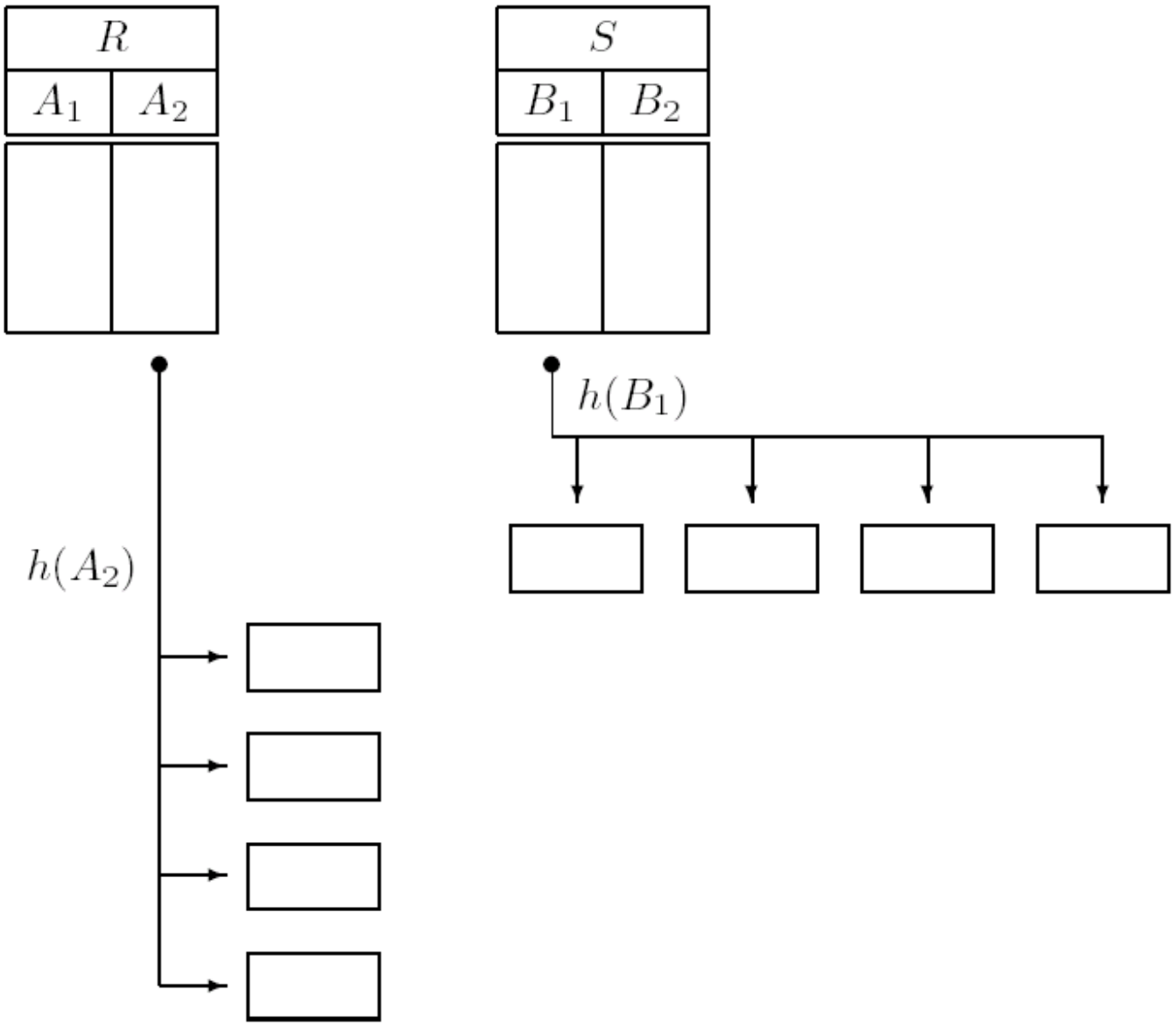
Implementierung der Verbindung: Strategien

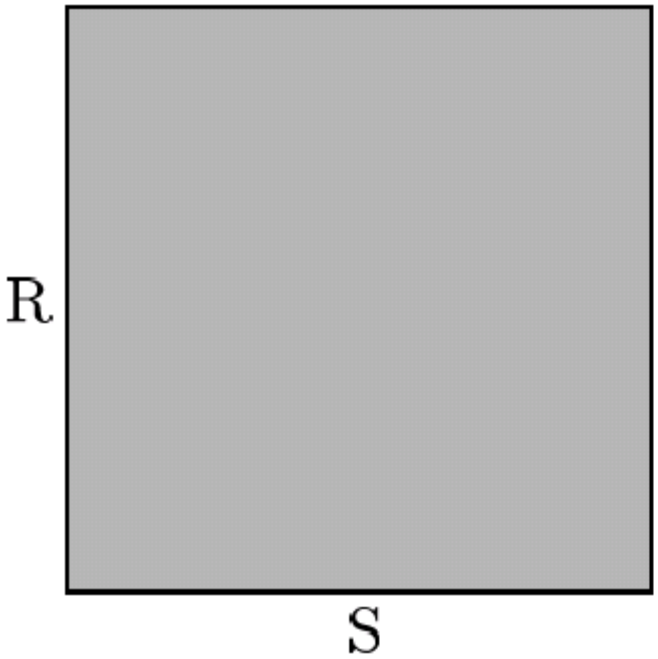


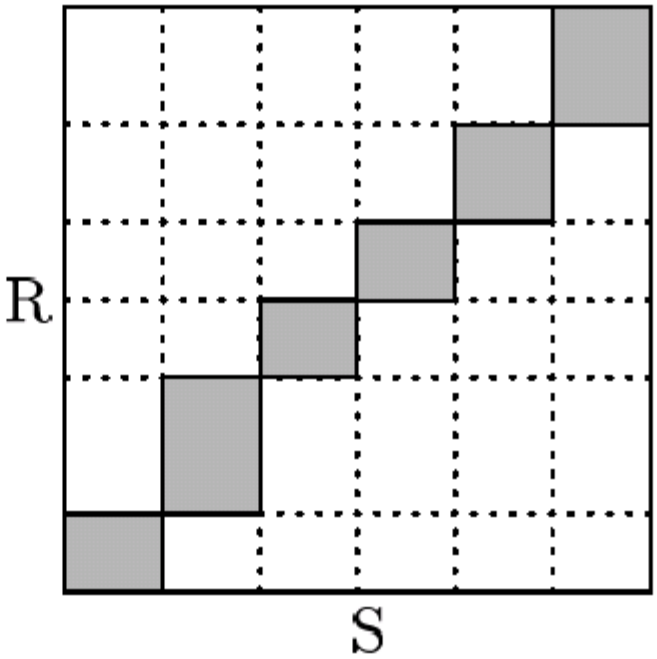
Partitionierung von Relationen



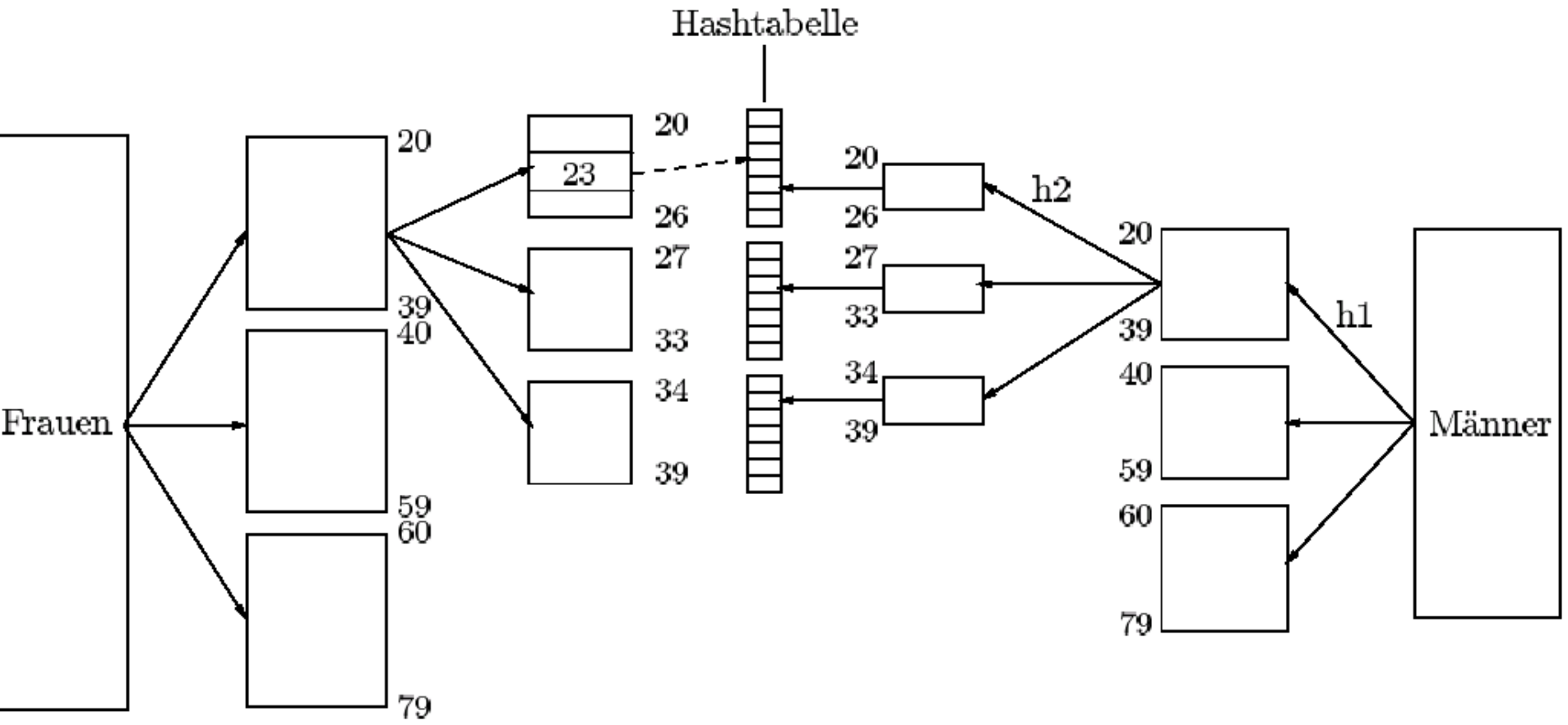
Vergleich der Tupel in der "Diagonalen"



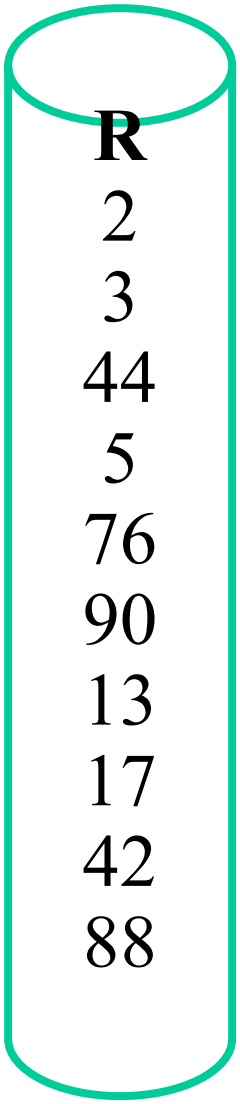

R
S


R
S

Demonstration der Partitionierung

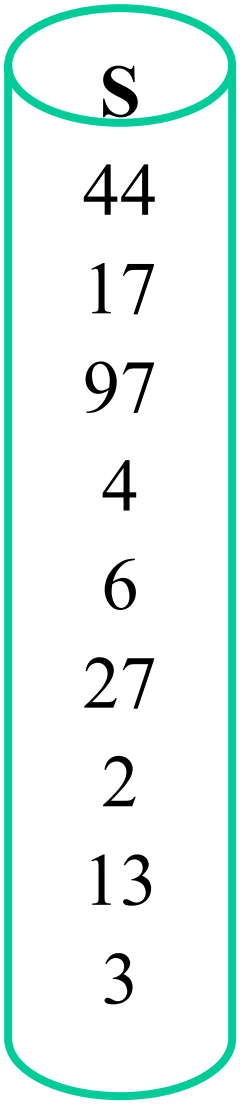


Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

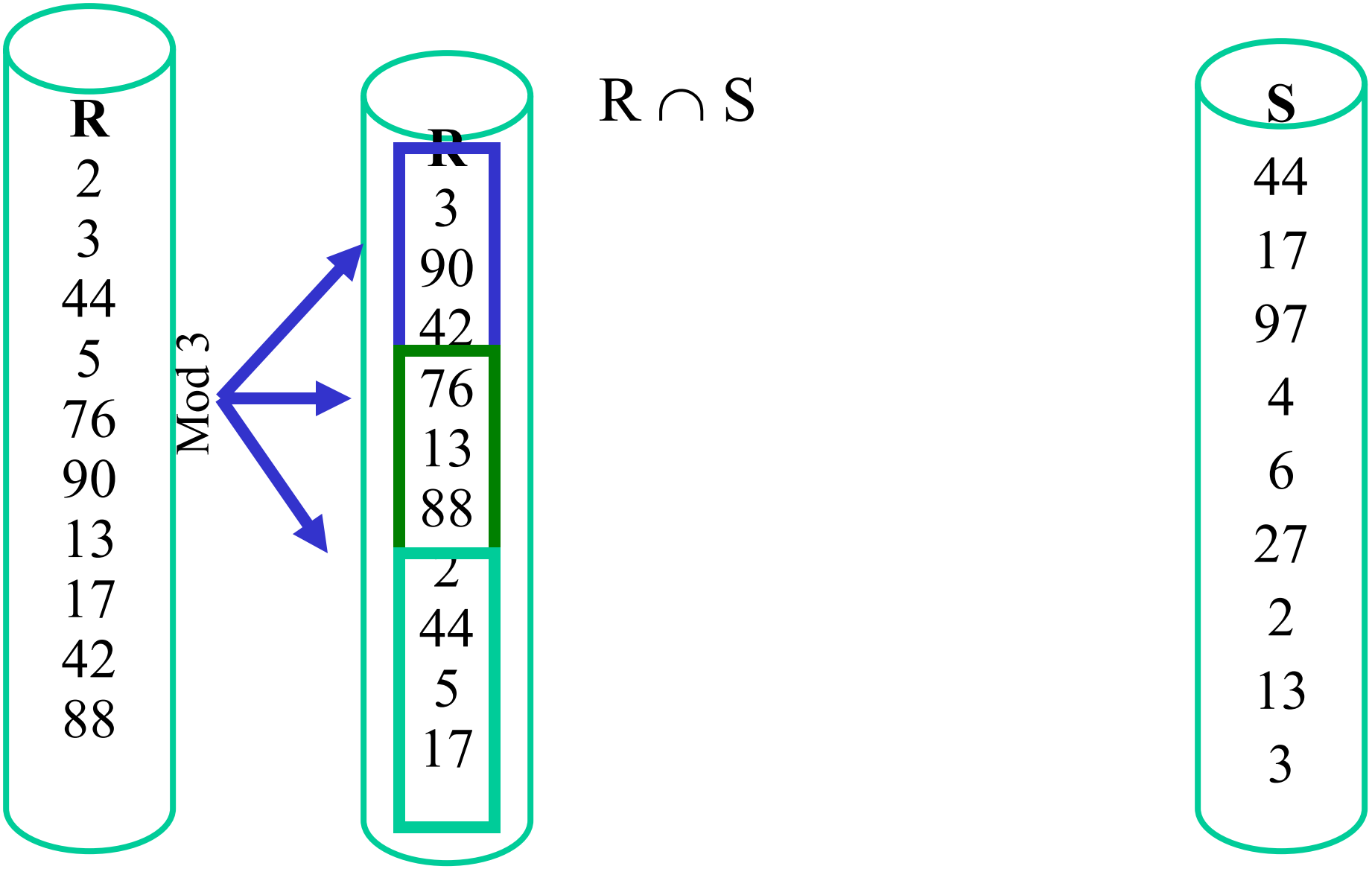


$R \cap S$

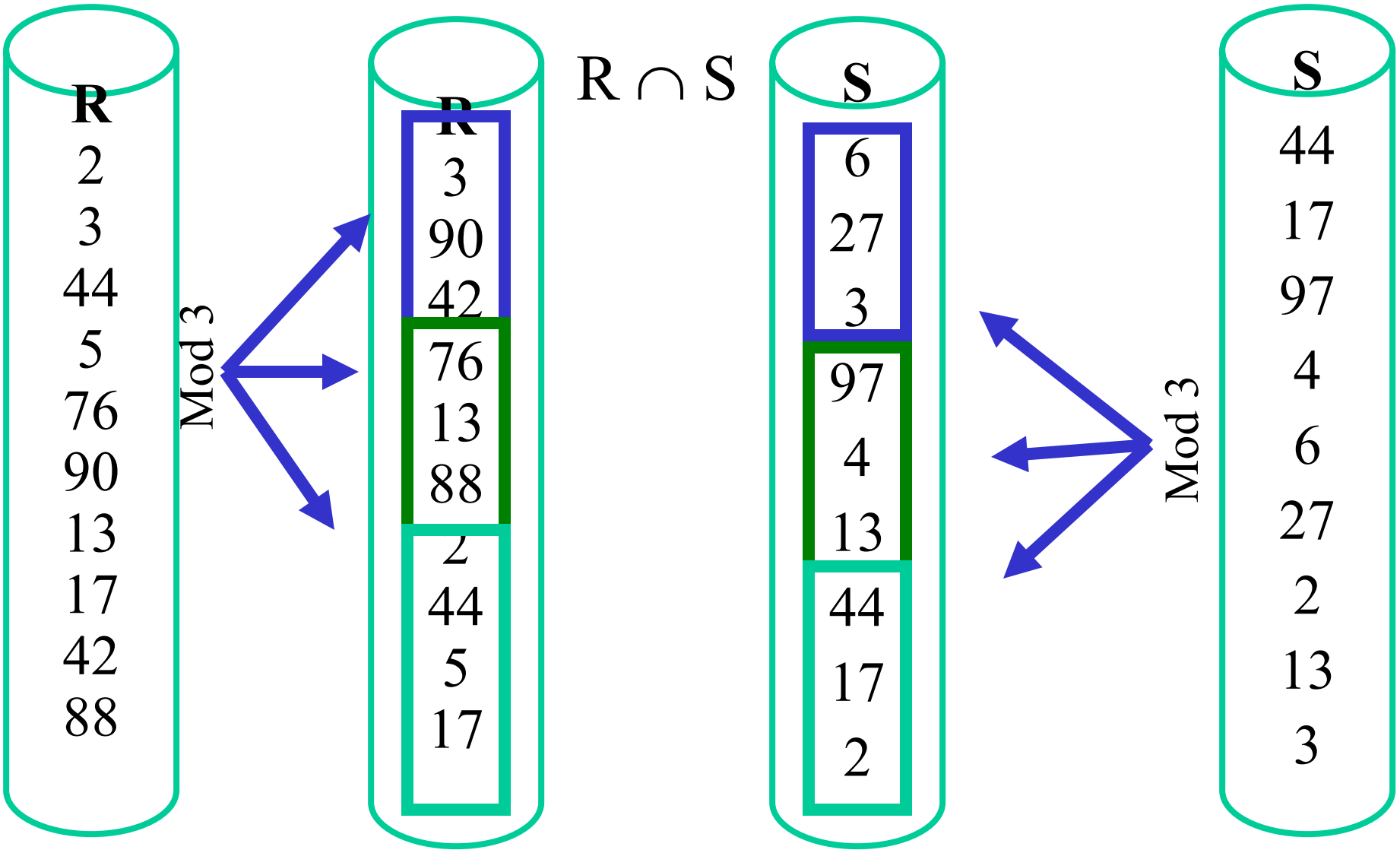
- Nested Loop: $O(N^2)$
- Sortieren: $O(N \log N)$
- Partitionieren und Hashing



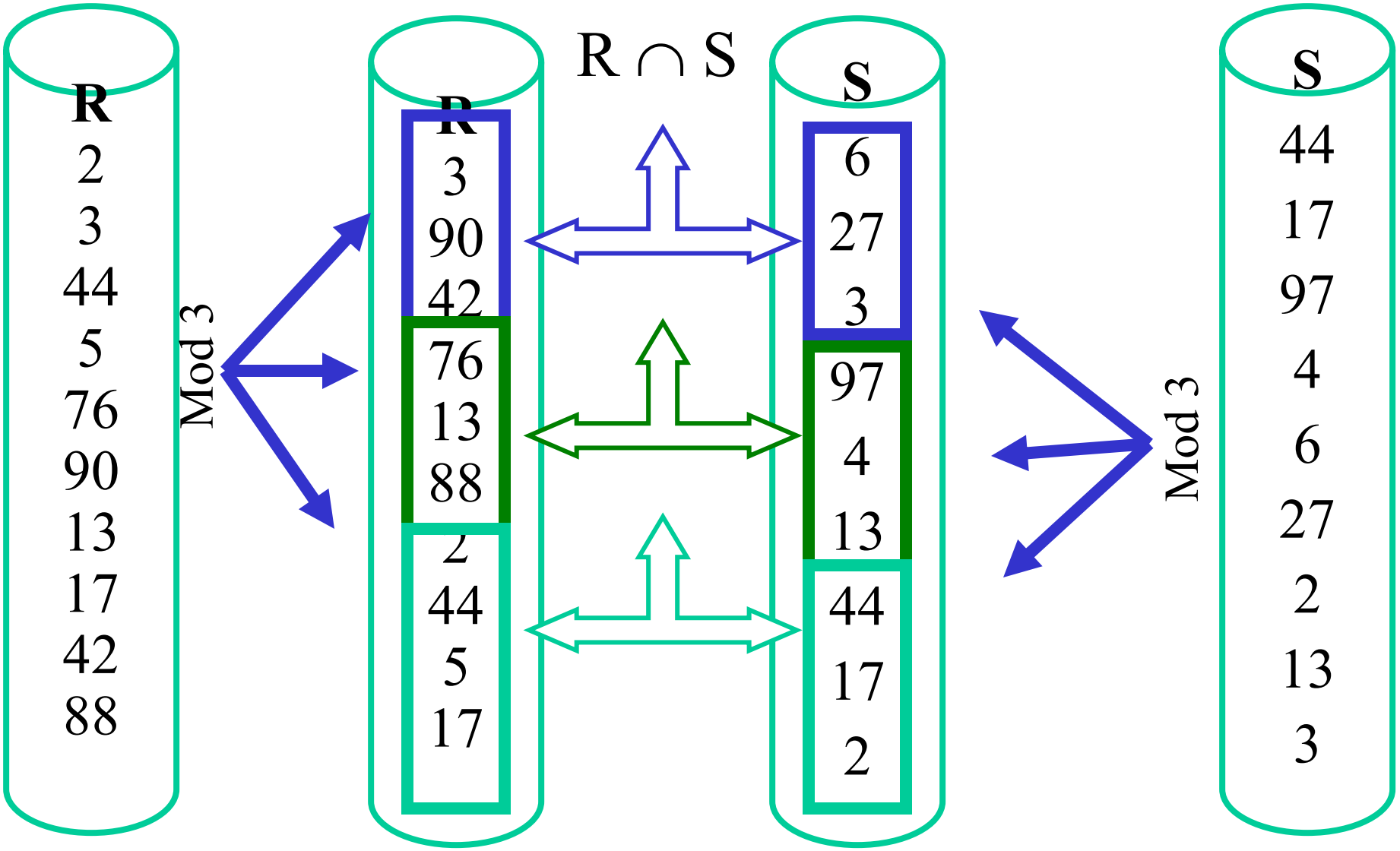
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



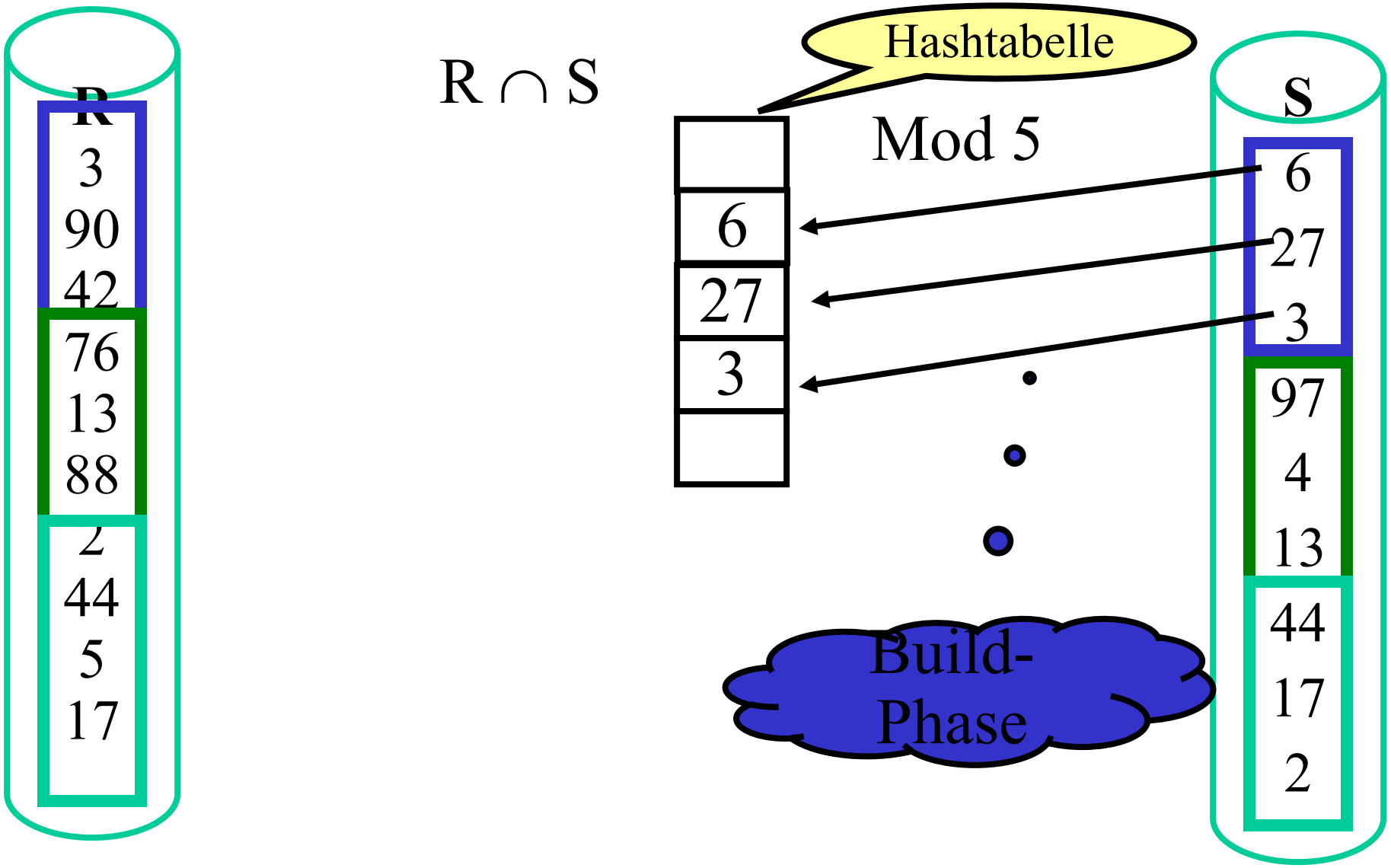
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



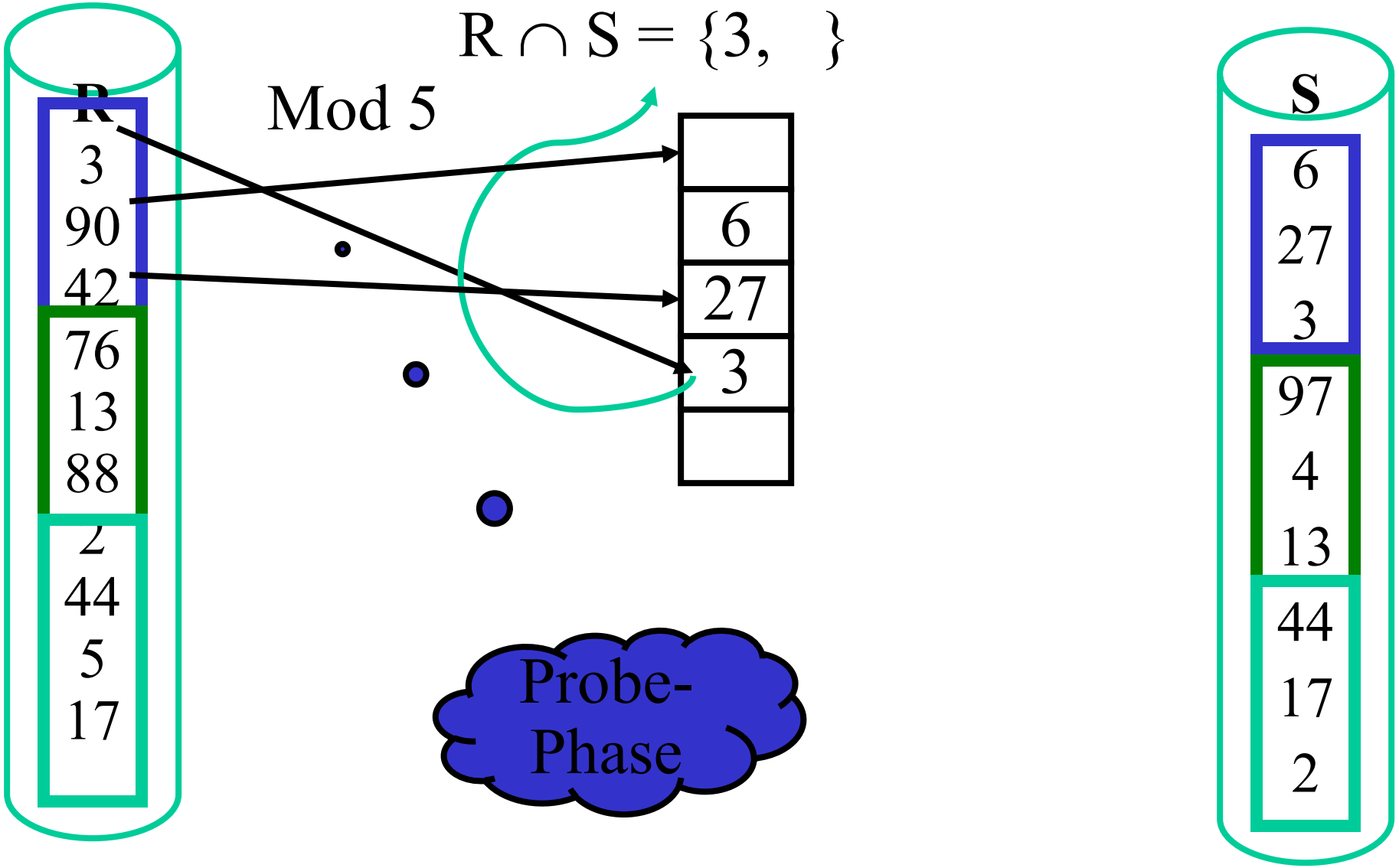
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



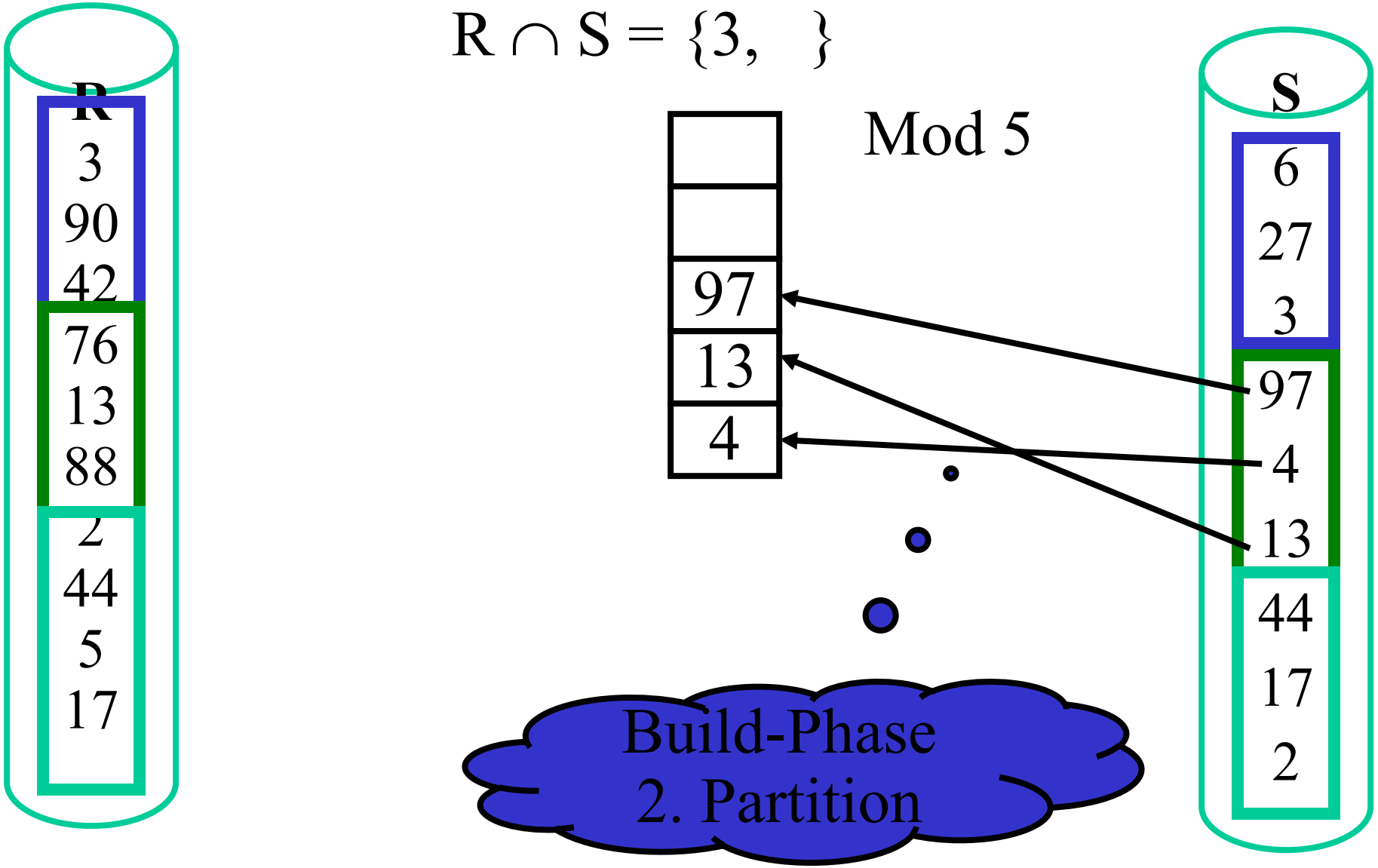
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



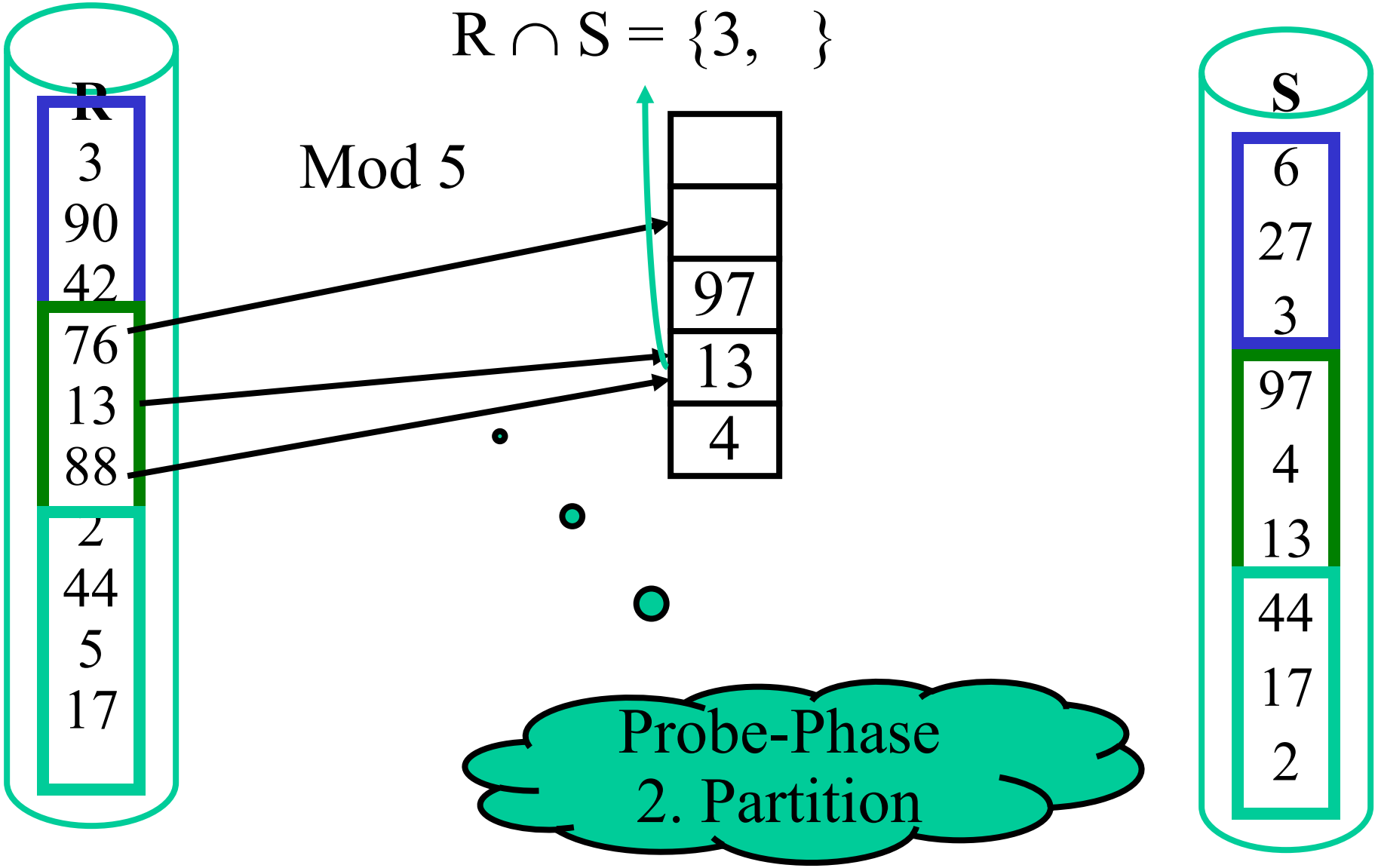
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



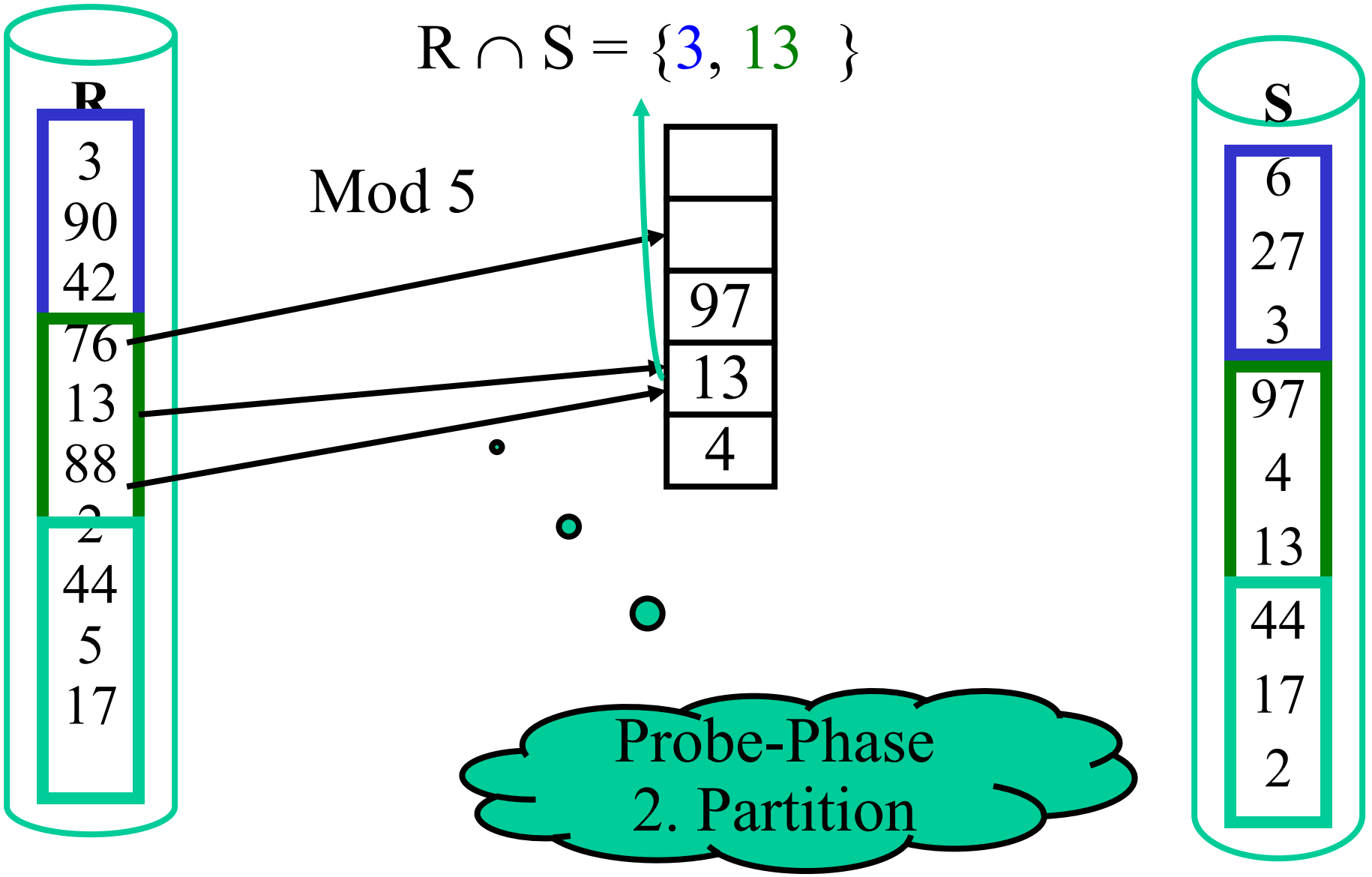
Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus



Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

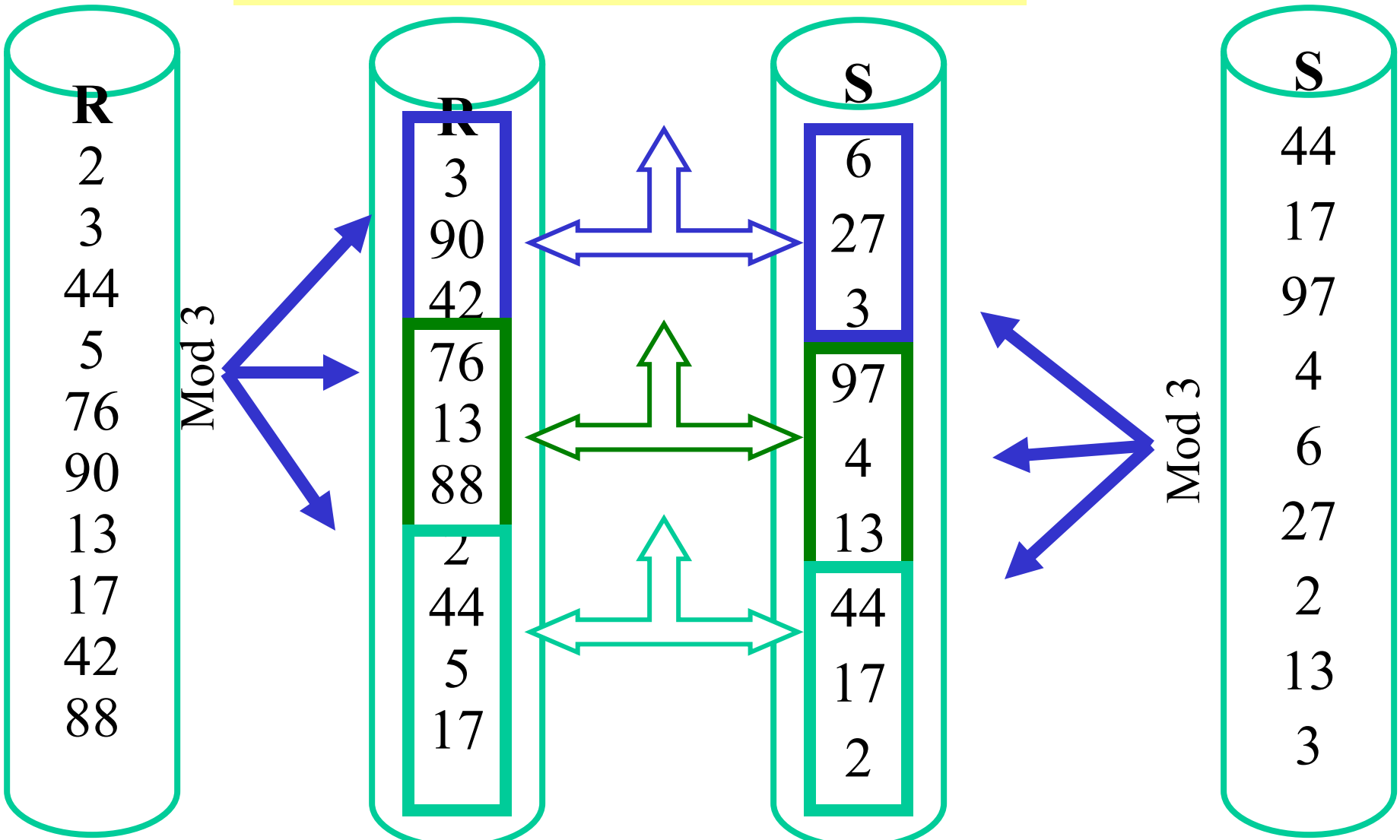


Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

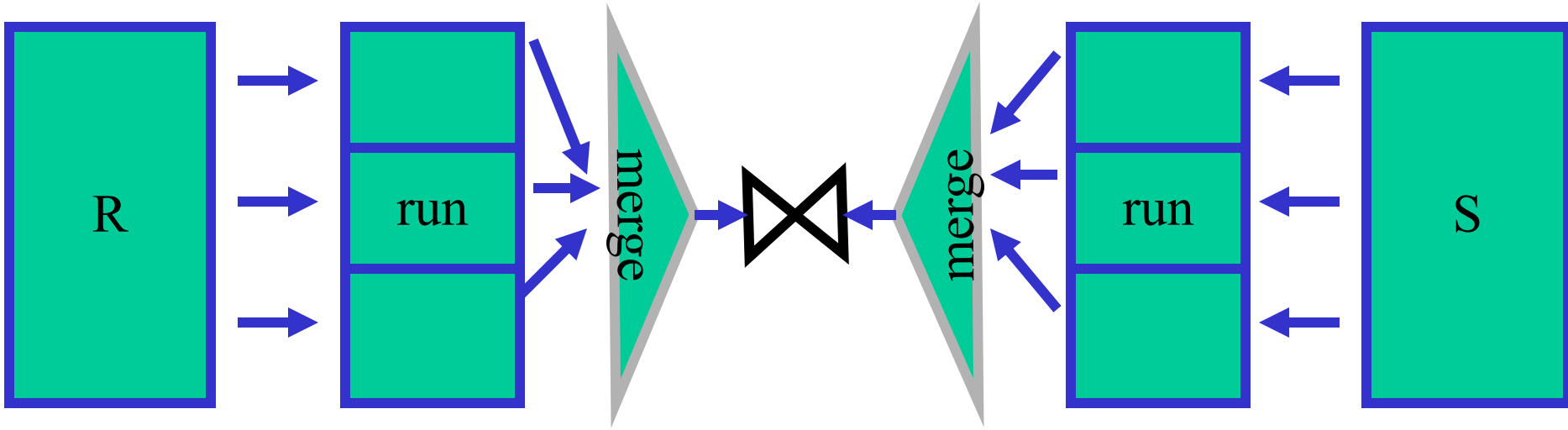


Mengendurchschnitt mit einem Hash/Partitionierungs-Algorithmus

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



Vergleich: Sort/Merge-Join versus Hash-Join



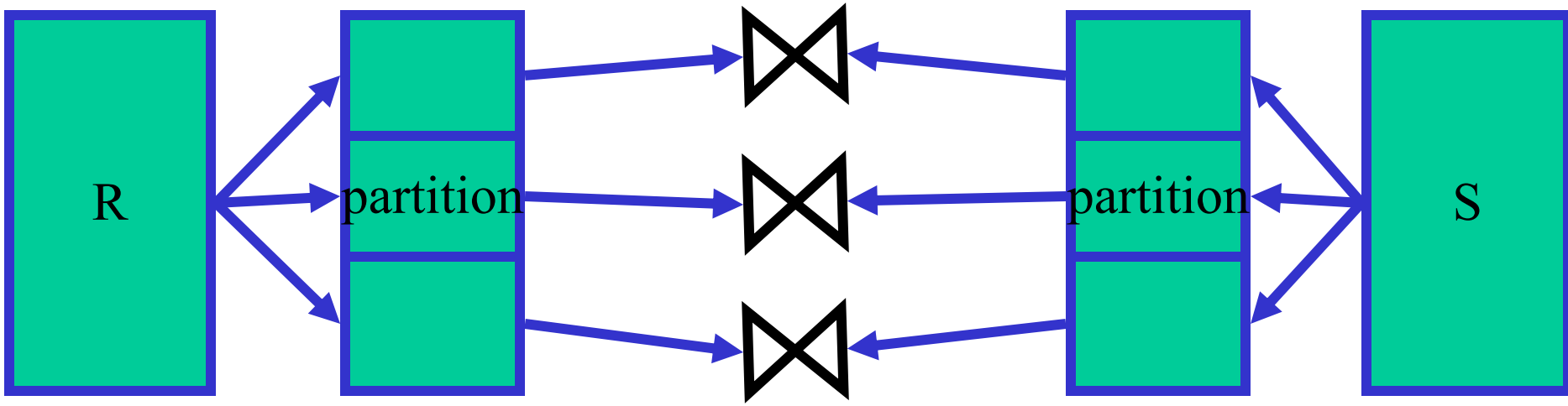


Illustration: Externes Sortieren

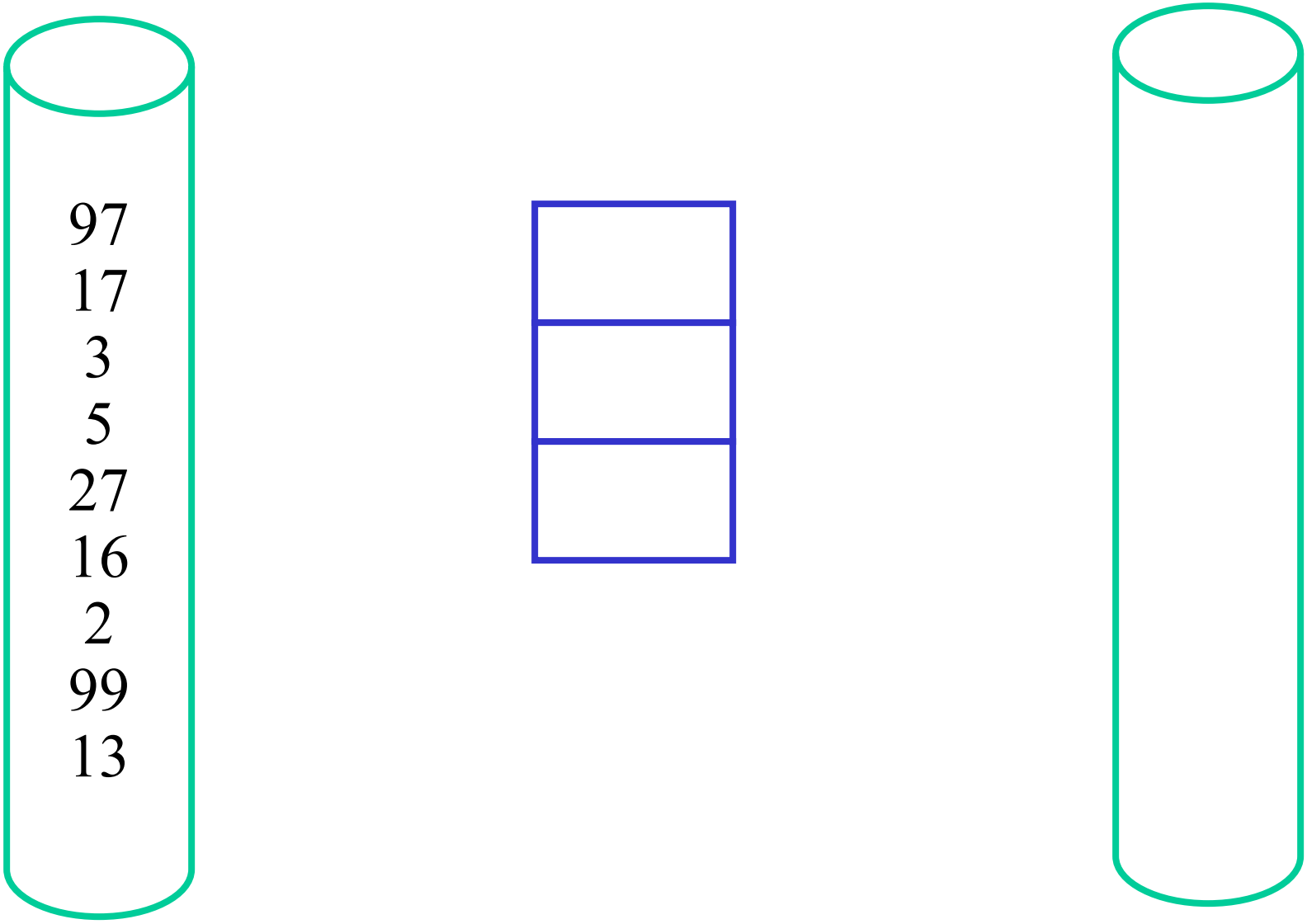


Illustration: Externes Sortieren

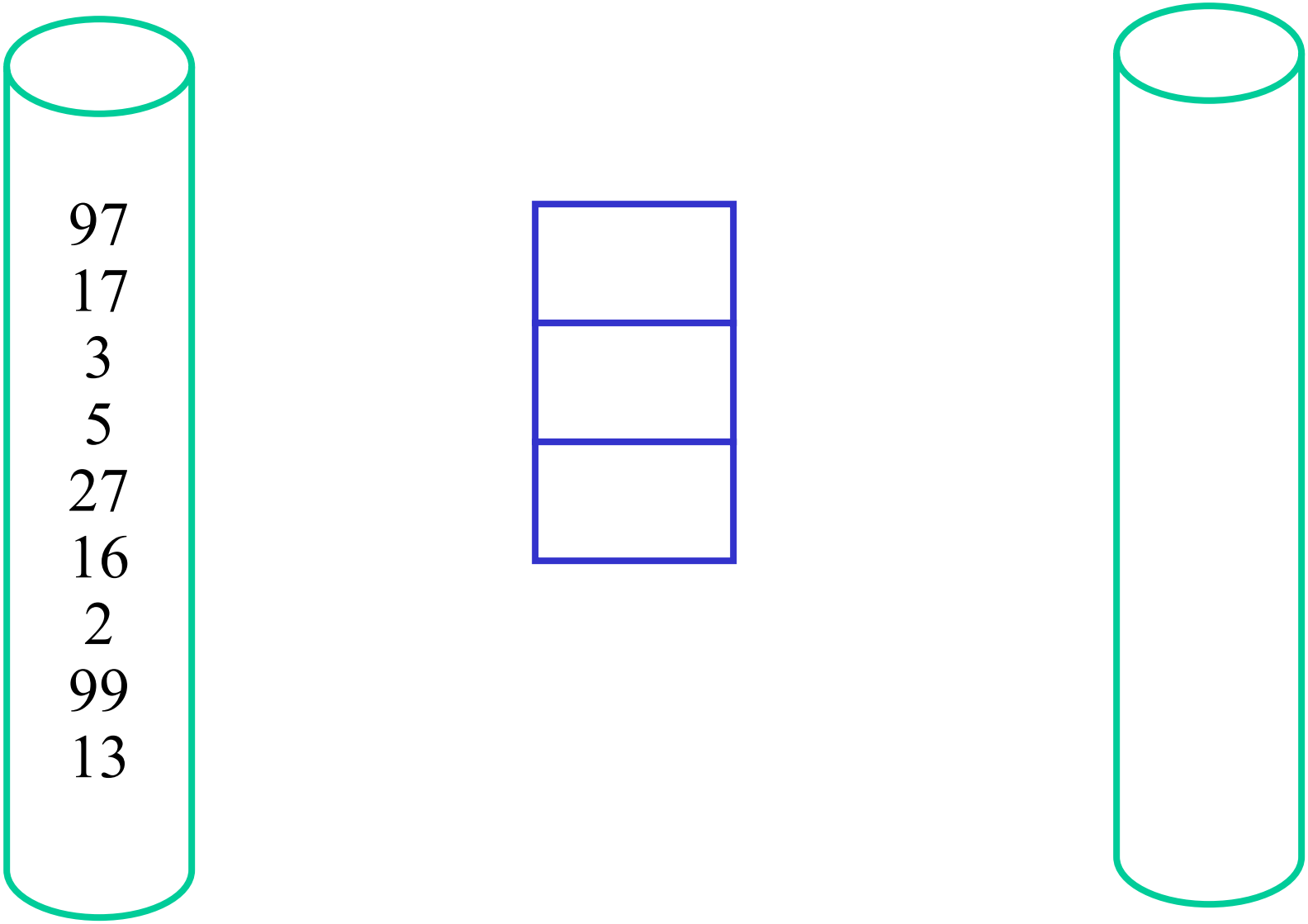


Illustration: Externes Sortieren

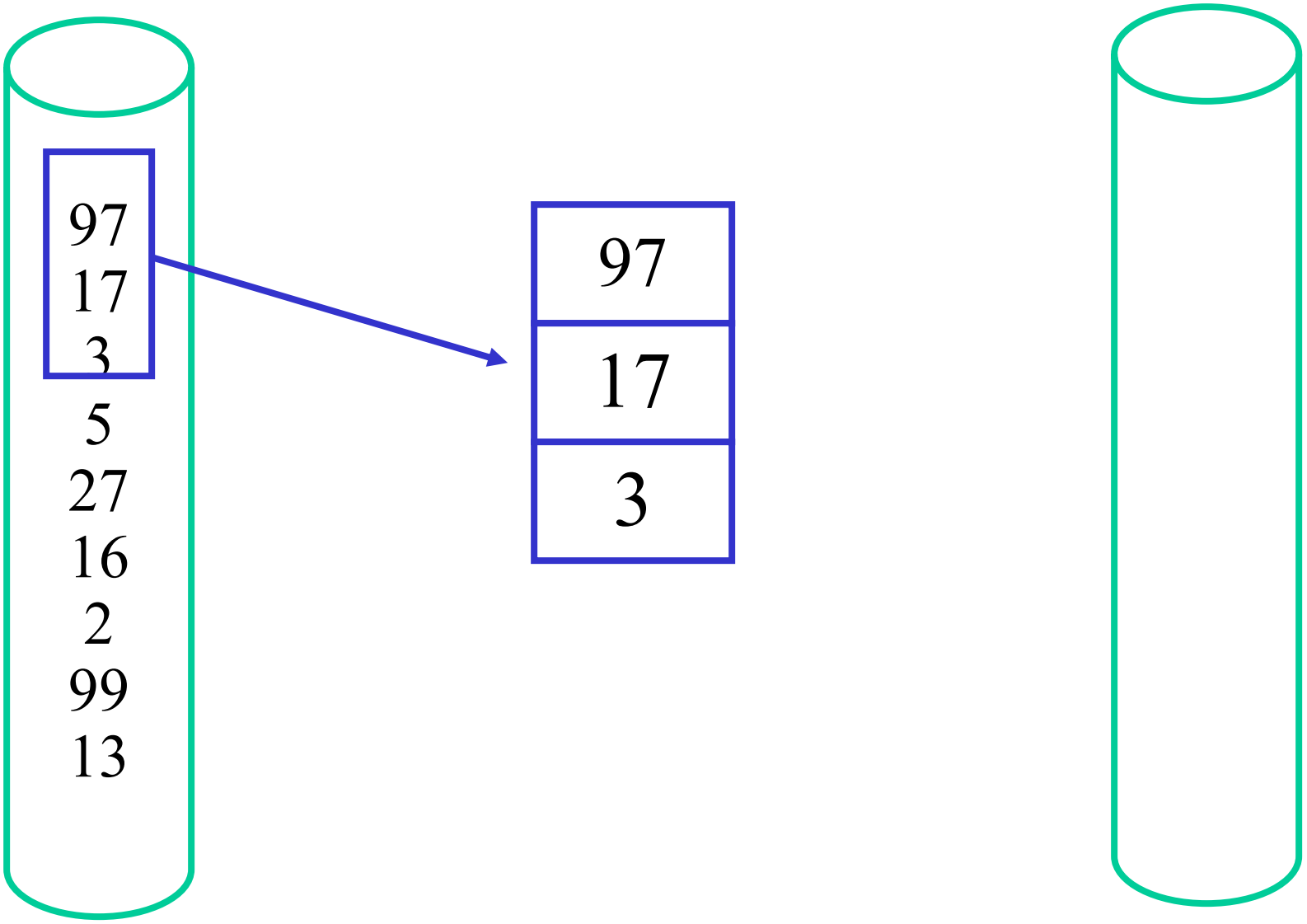


Illustration: Externes Sortieren

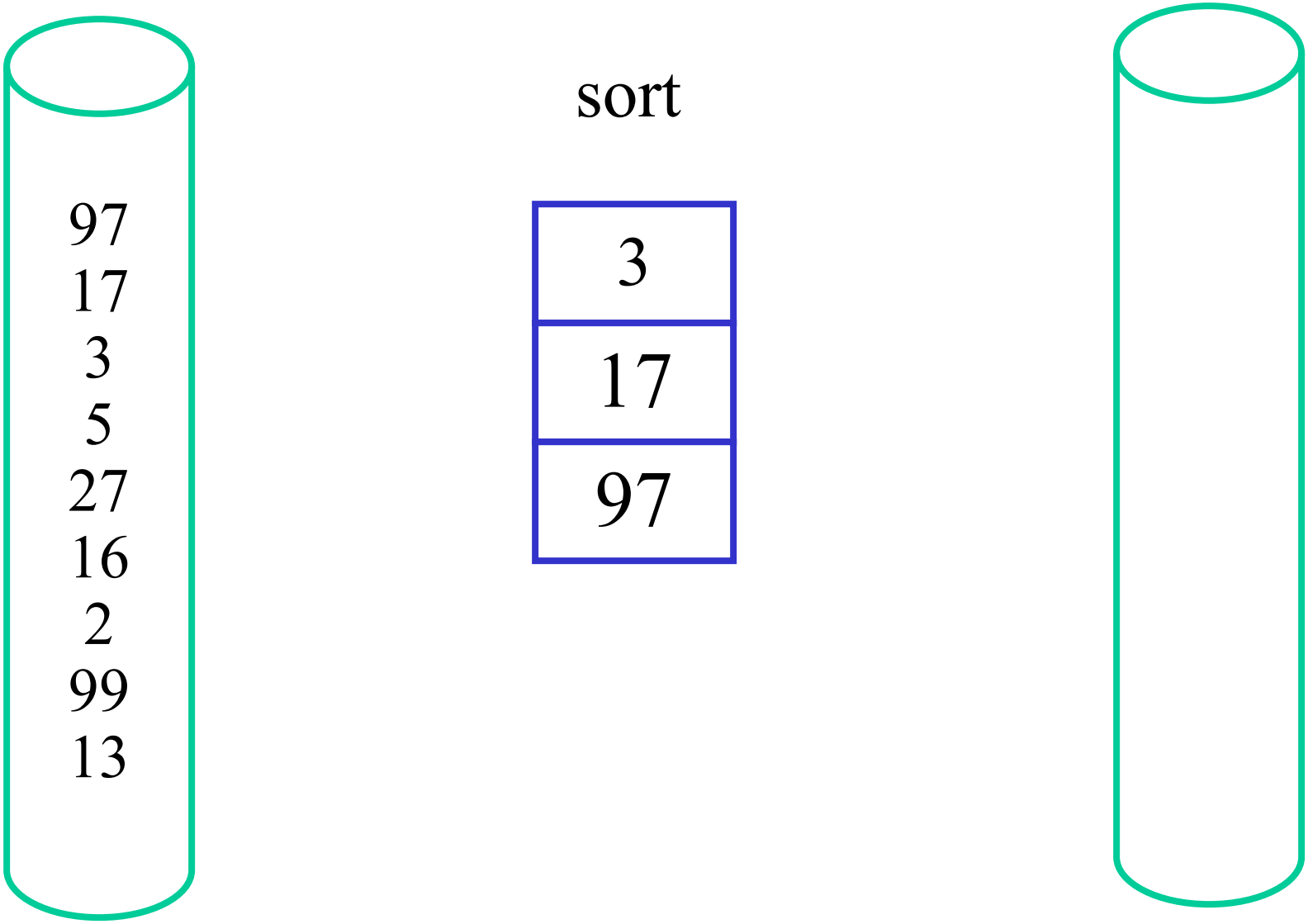


Illustration: Externes Sortieren

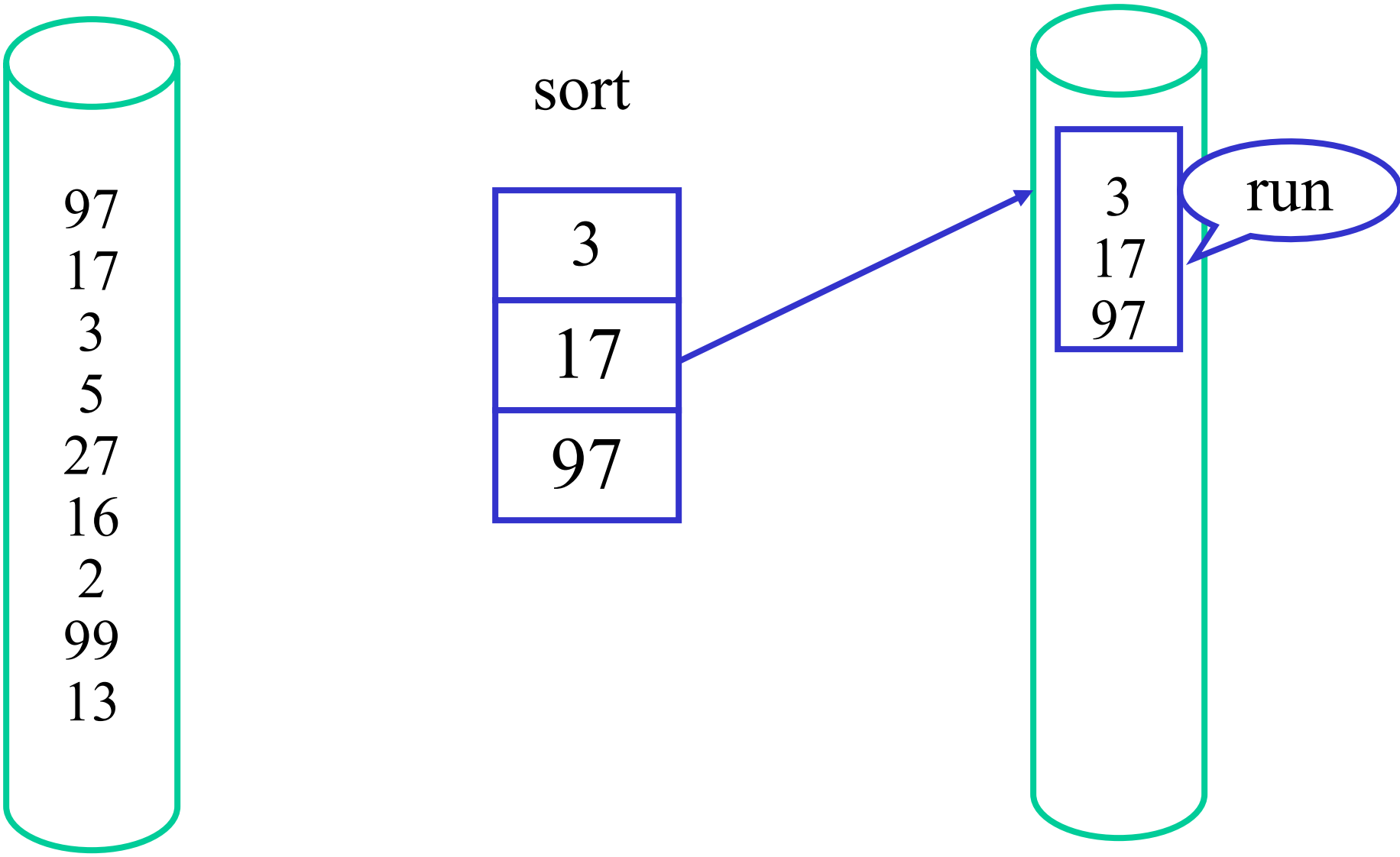


Illustration: Externes Sortieren

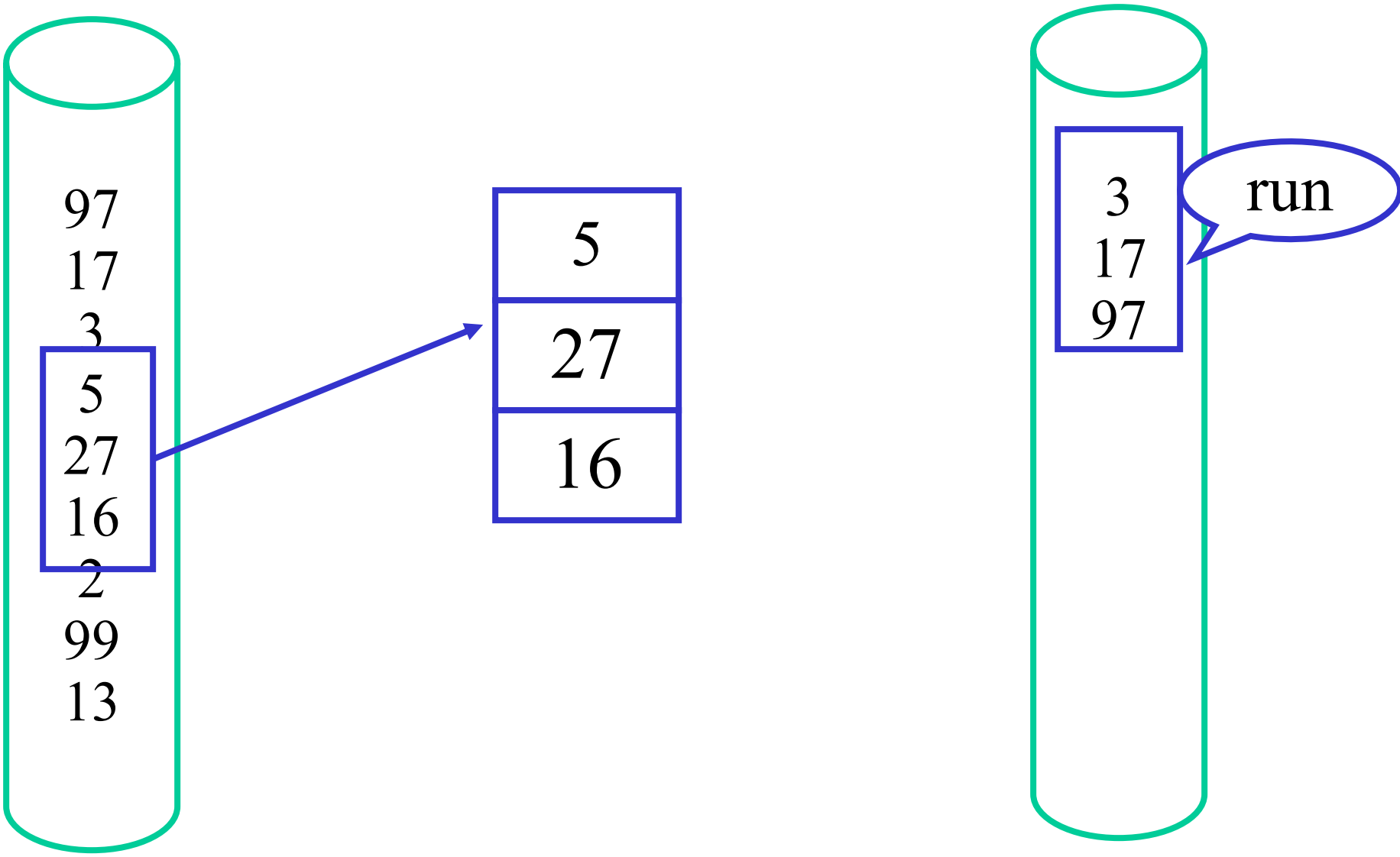


Illustration: Externes Sortieren

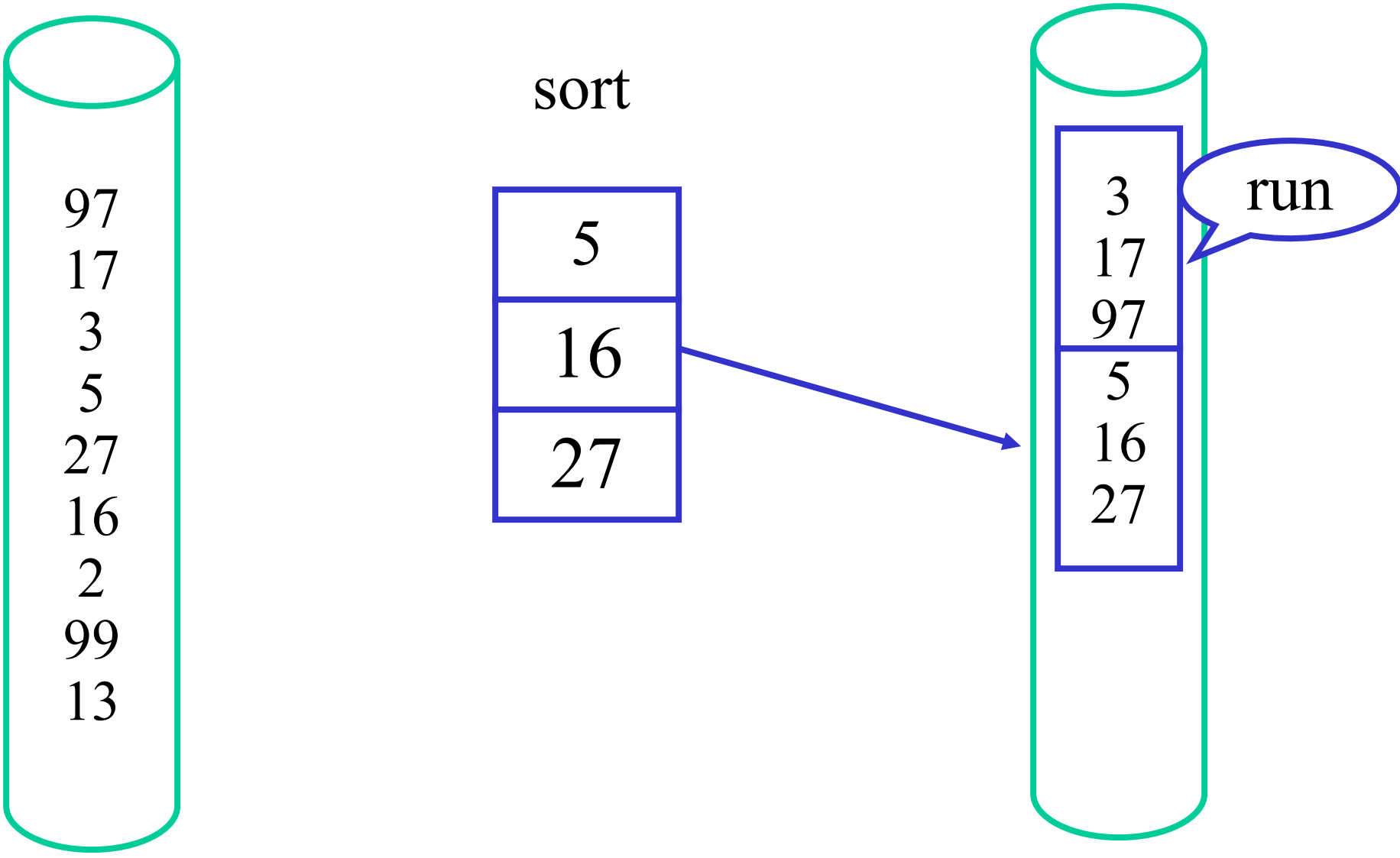


Illustration: Externes Sortieren

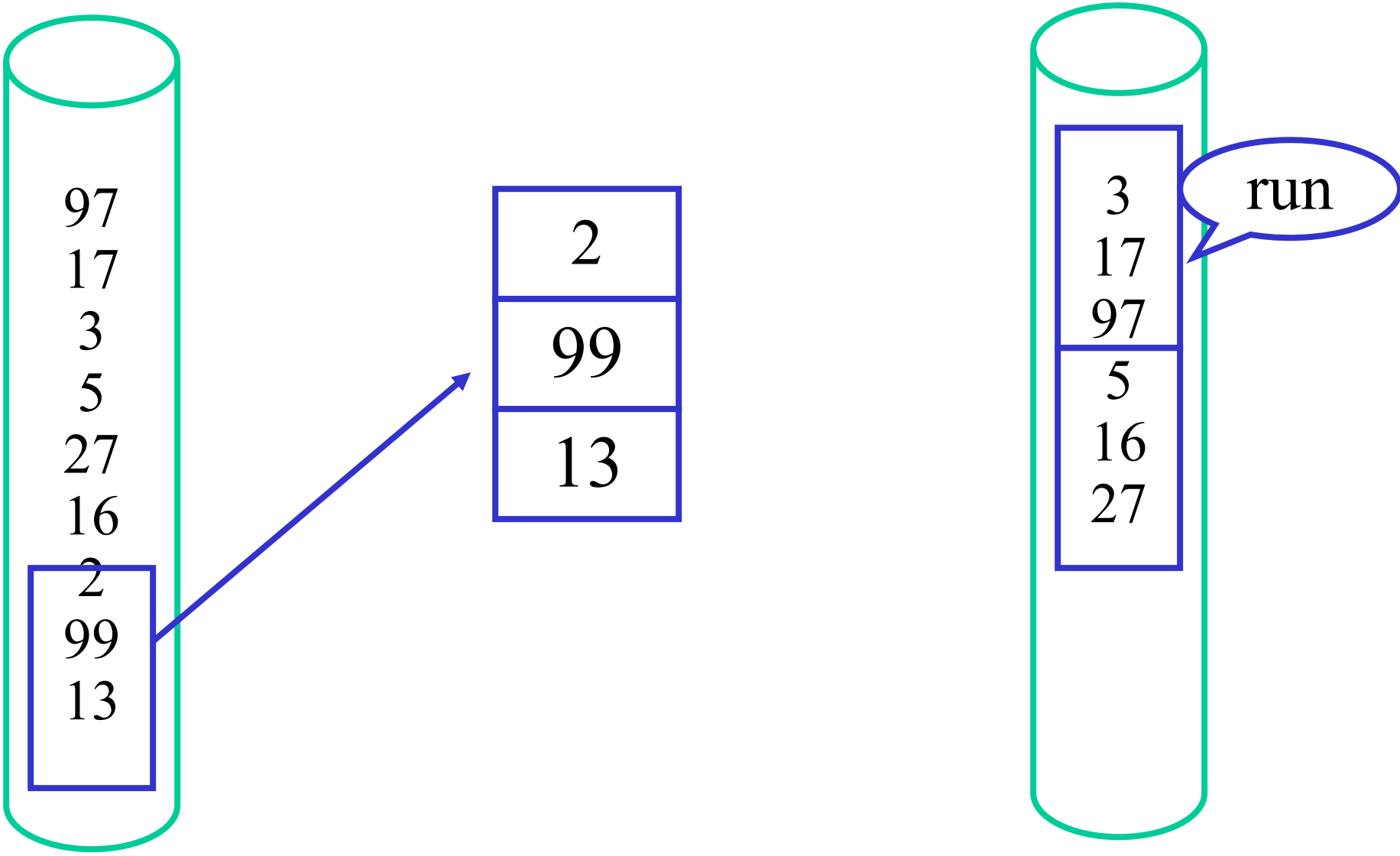


Illustration: Externes Sortieren

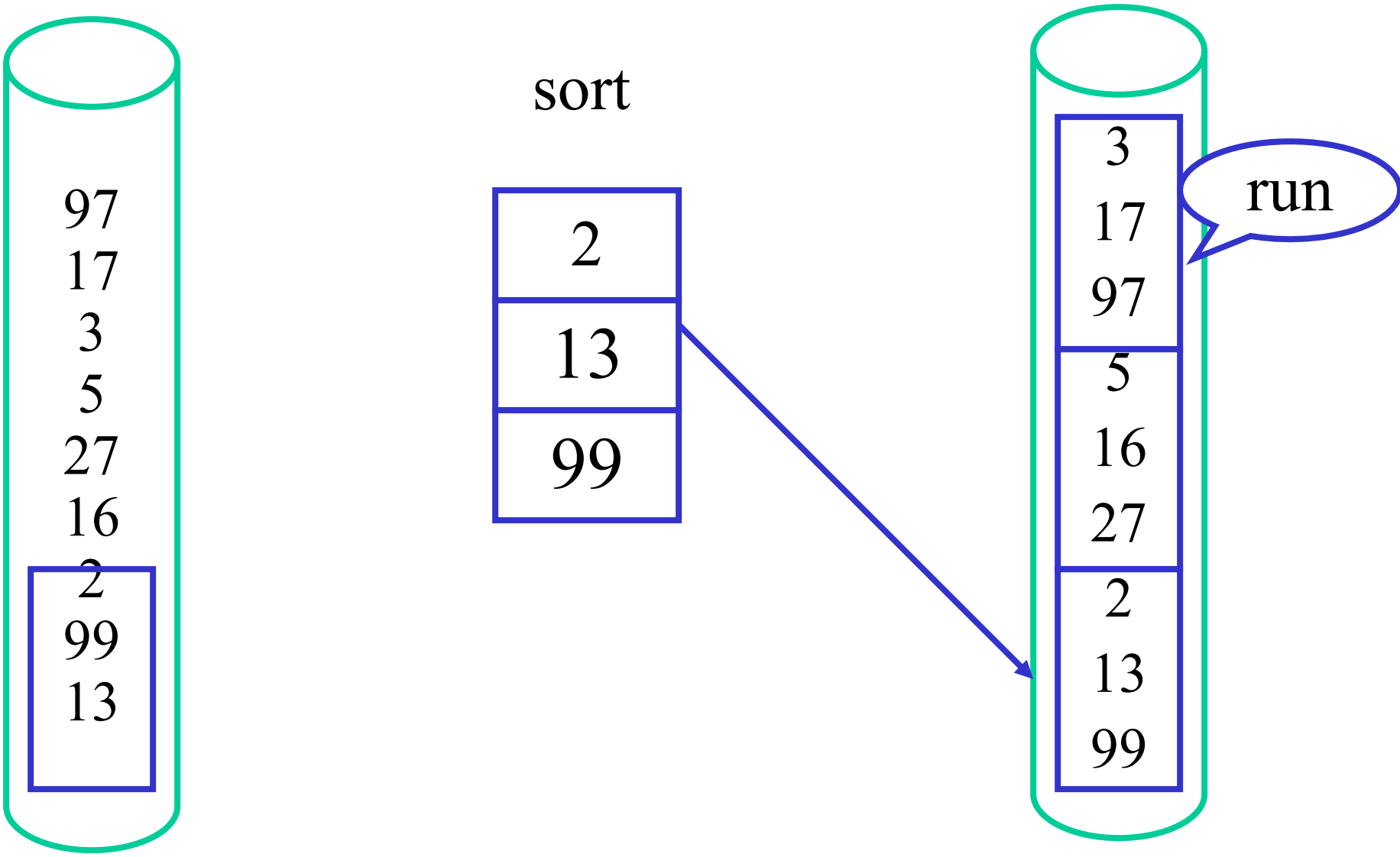


Illustration: Externes Sortieren

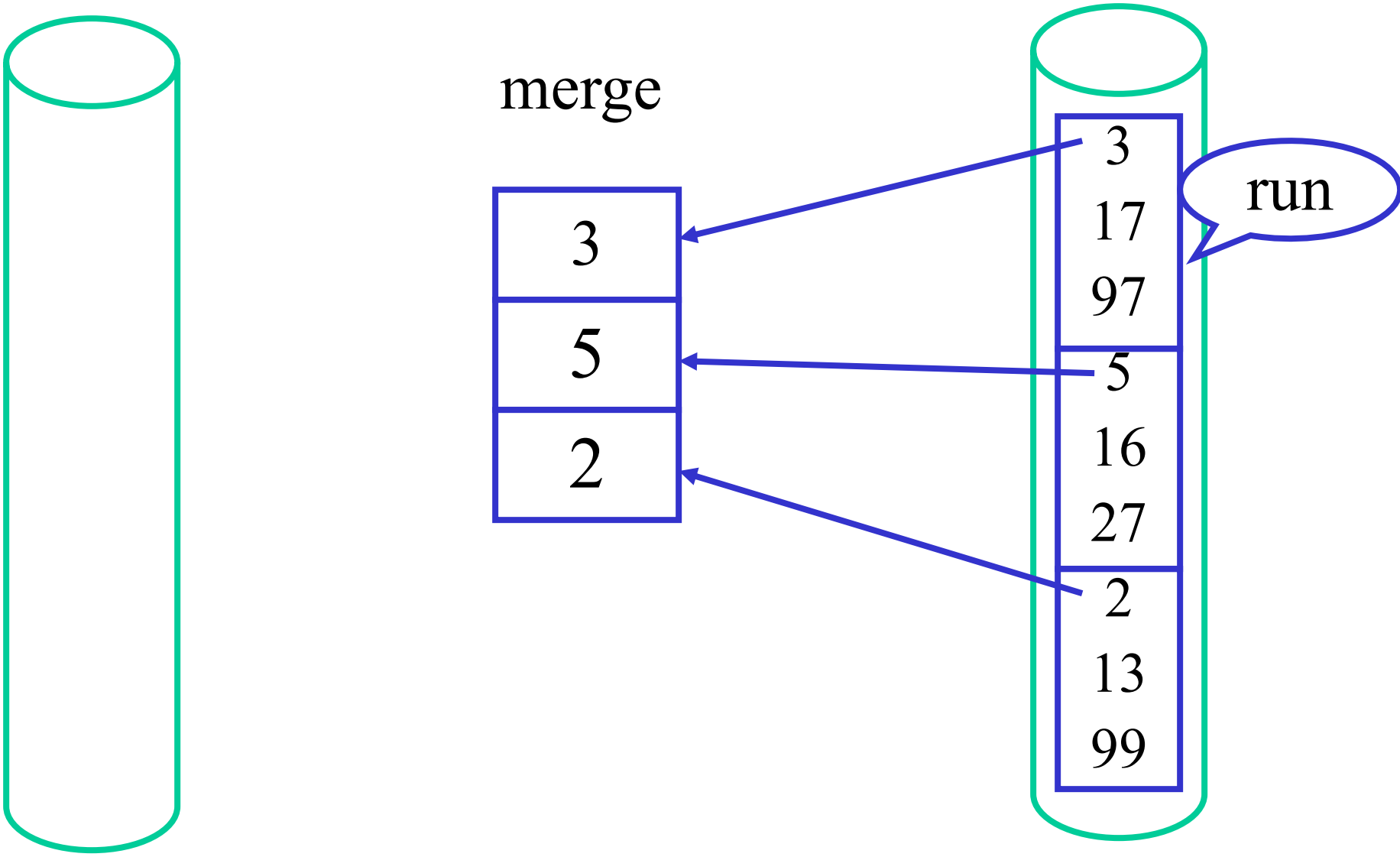


Illustration: Externes Sortieren

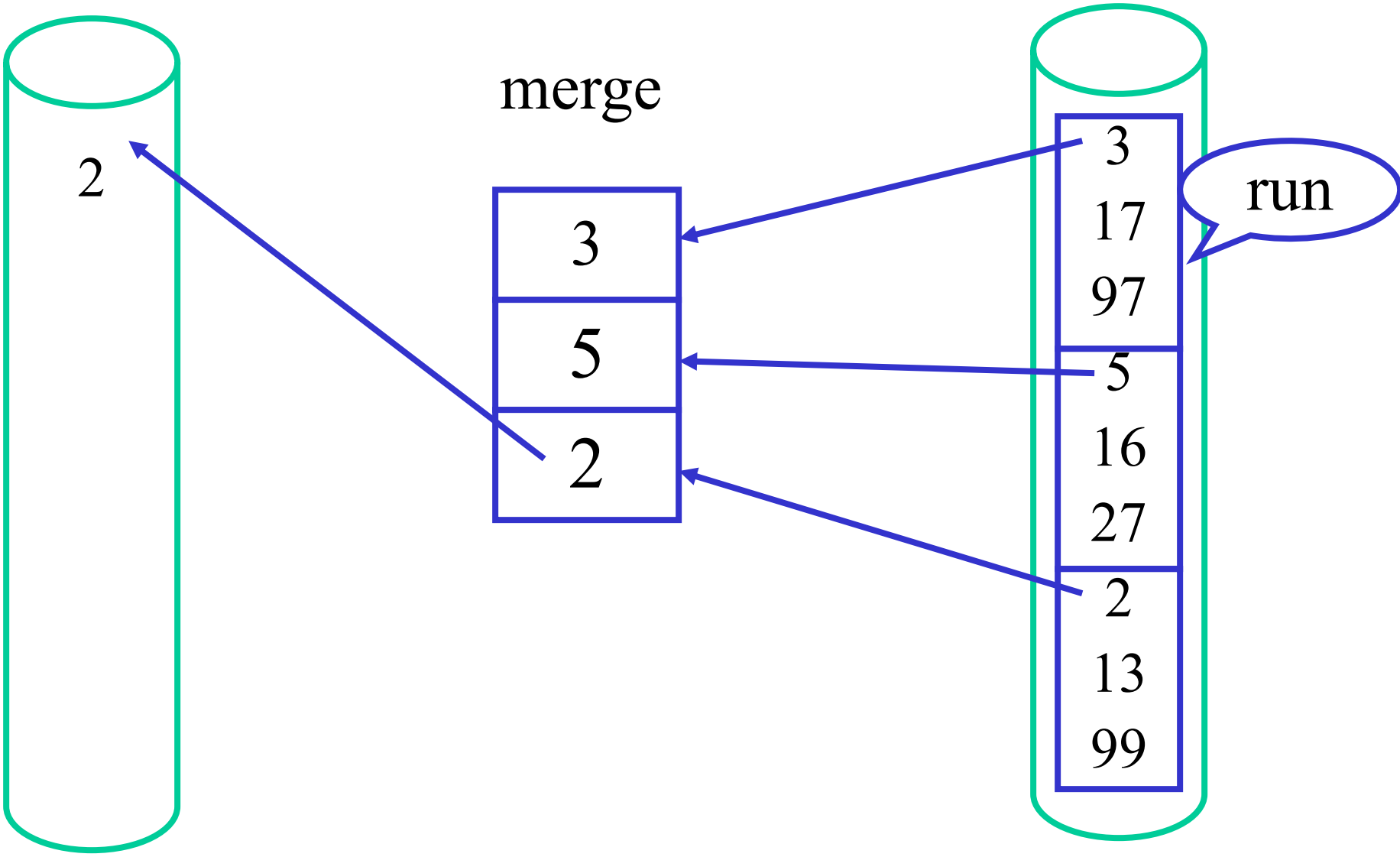


Illustration: Externes Sortieren

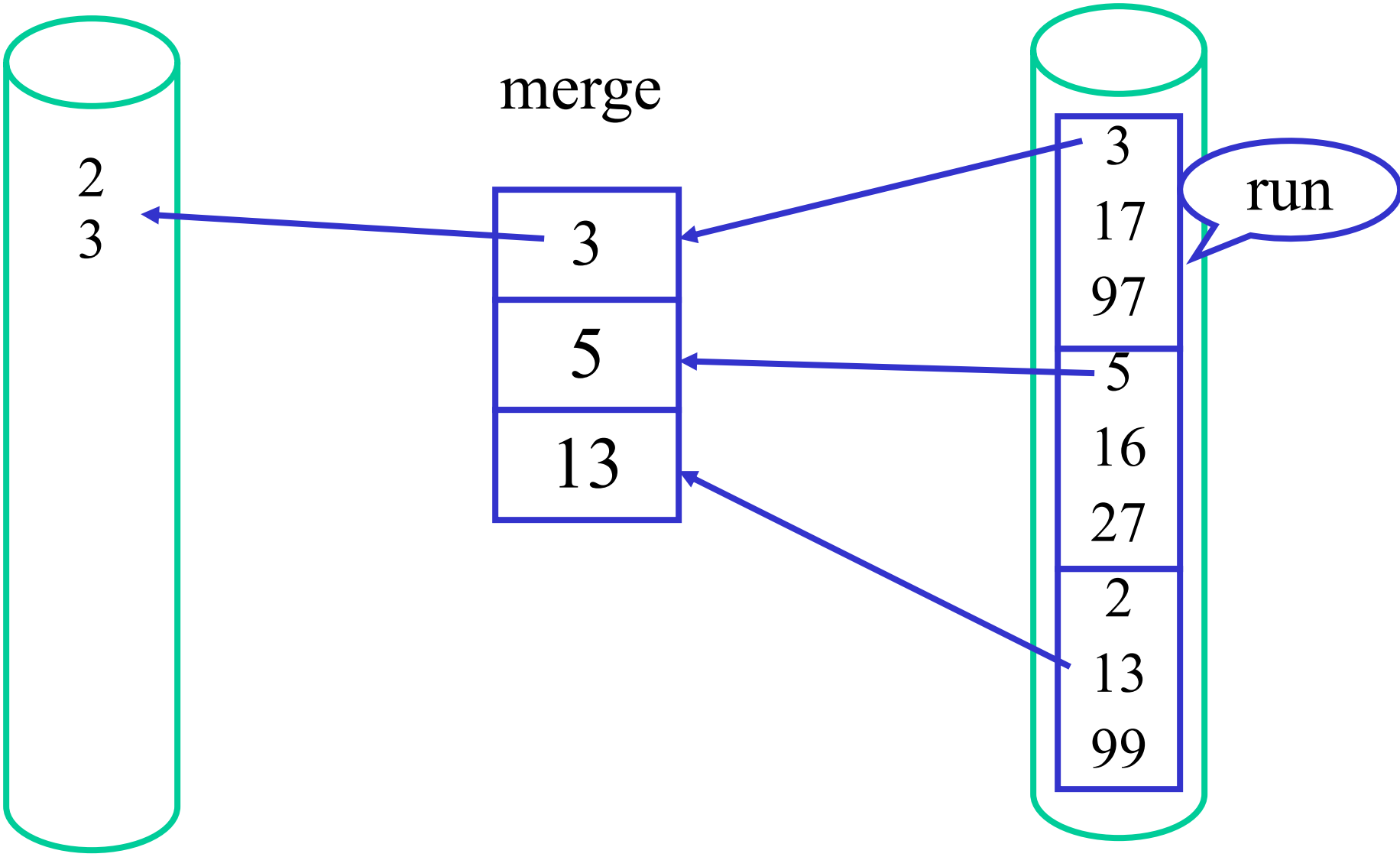


Illustration: Externes Sortieren

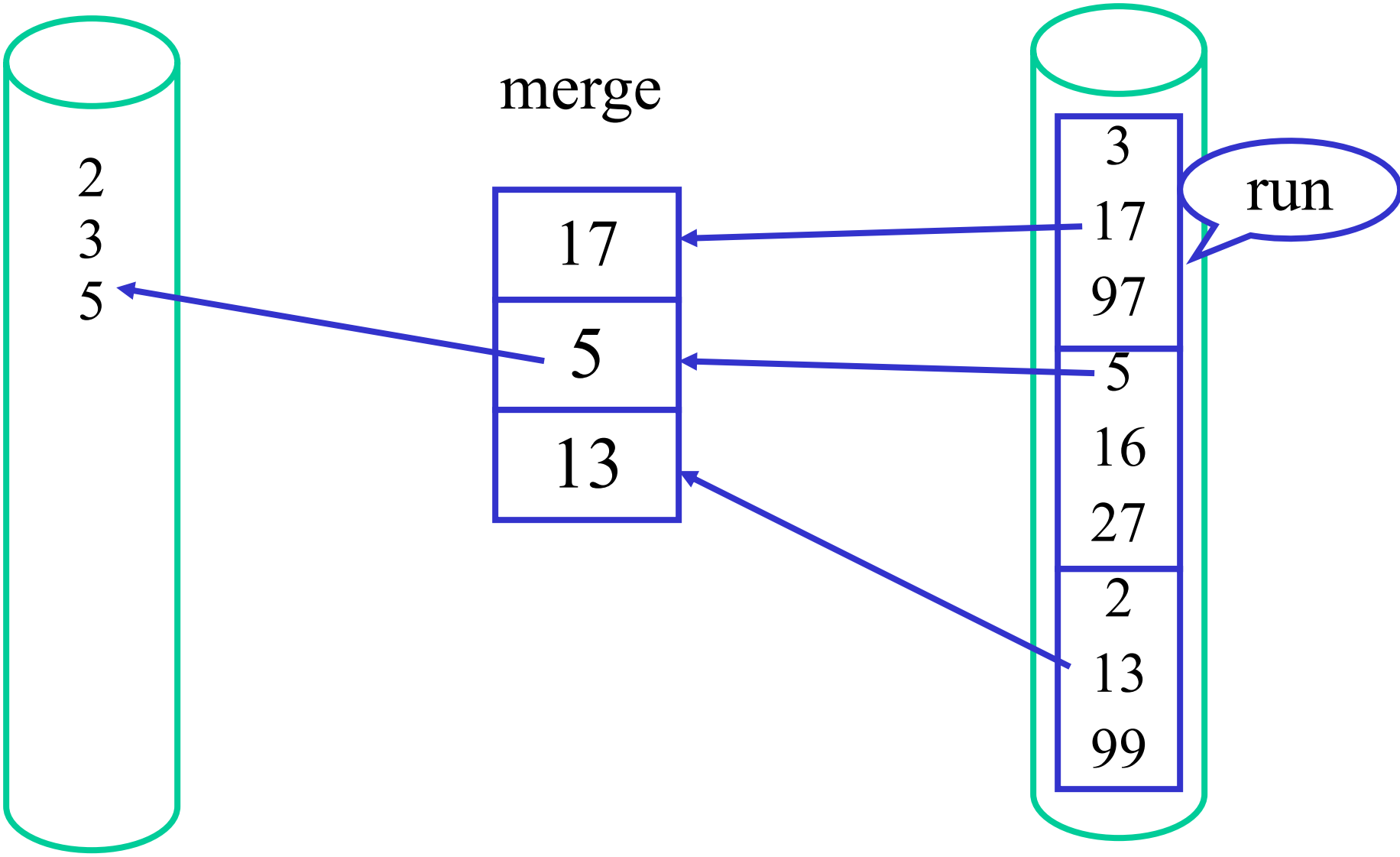


Illustration: Externes Sortieren

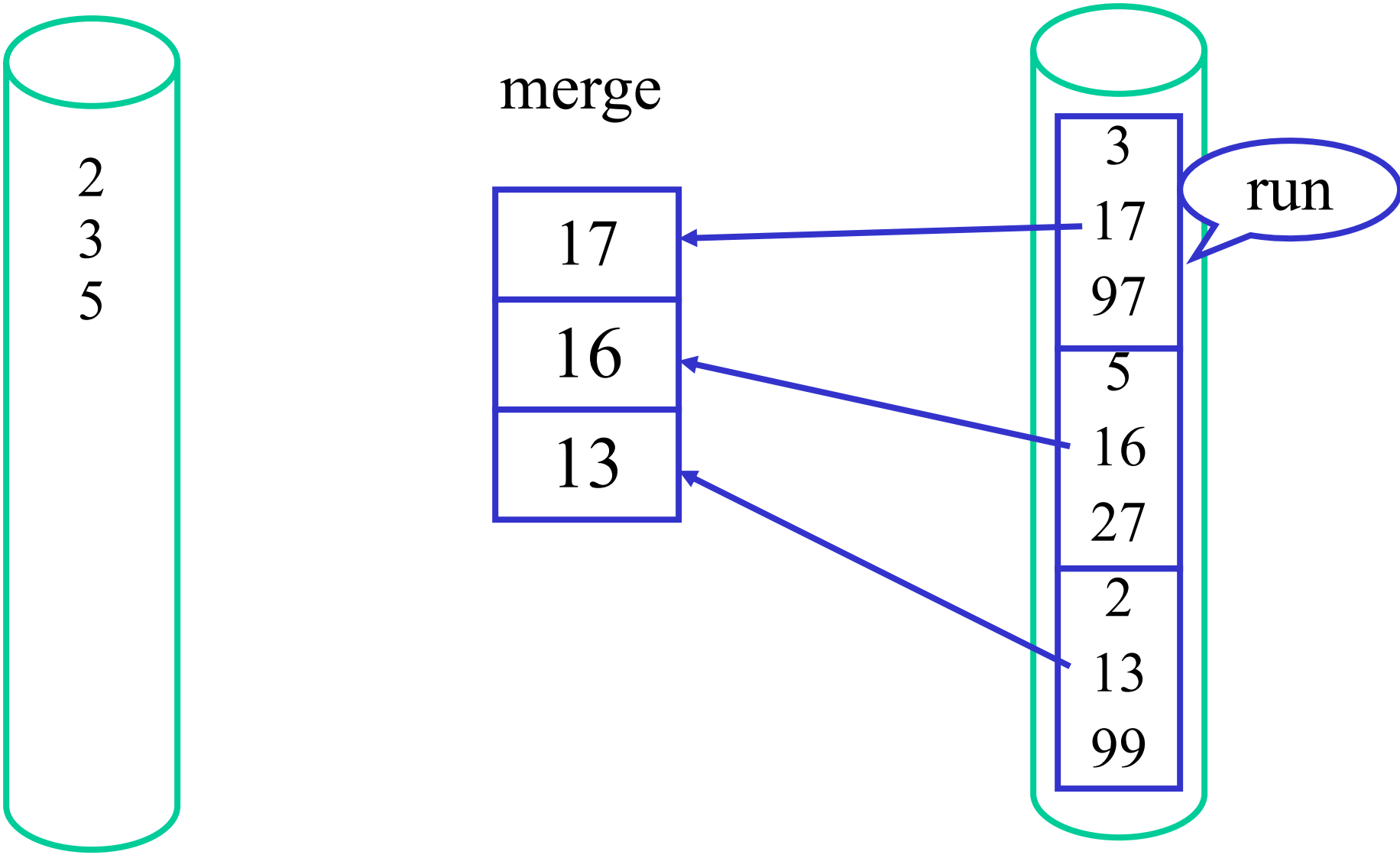
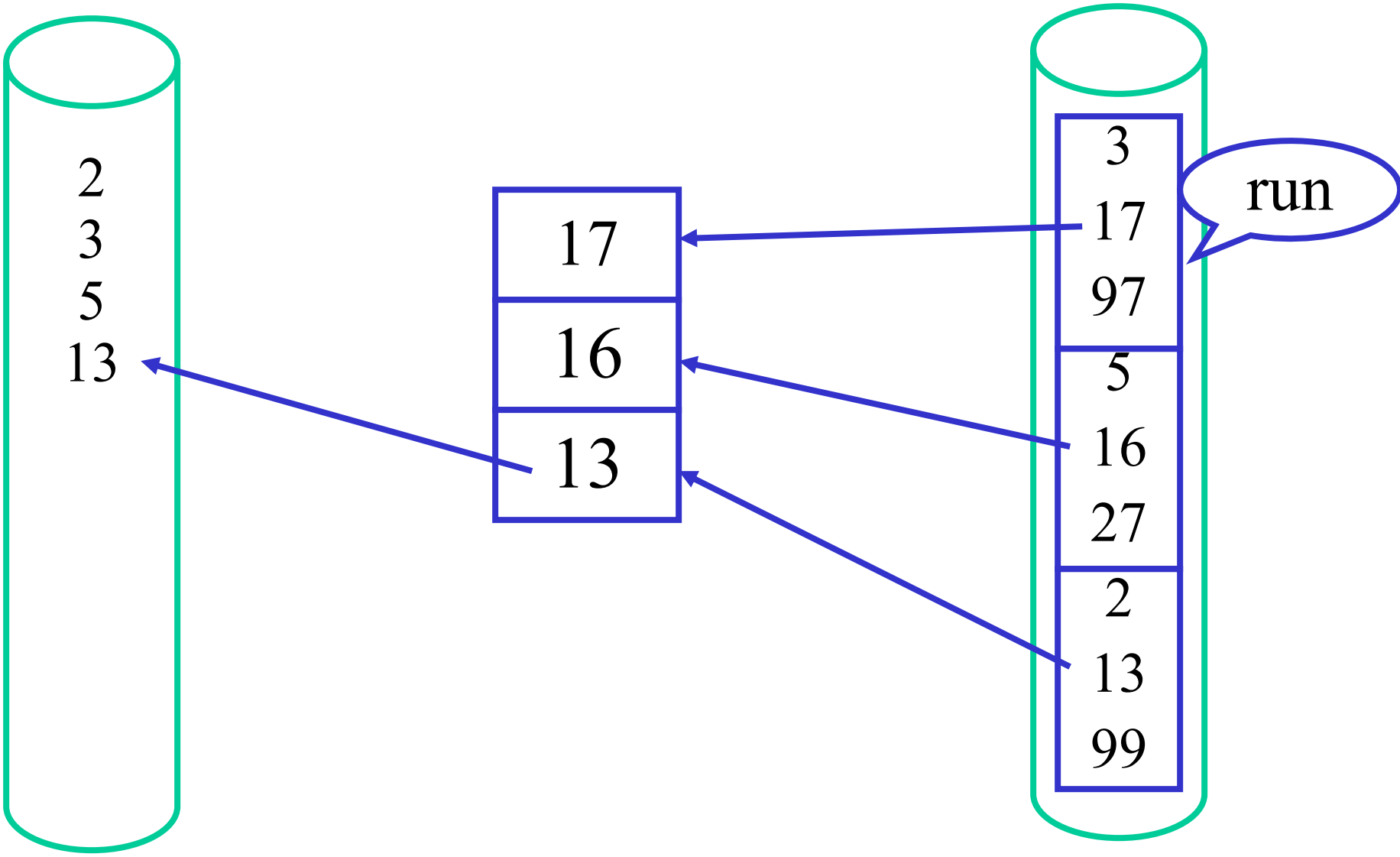


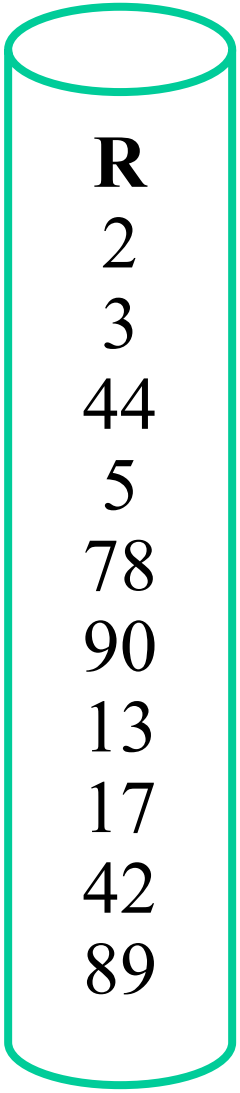
Illustration: Externes Sortieren



Implementierungs-Details

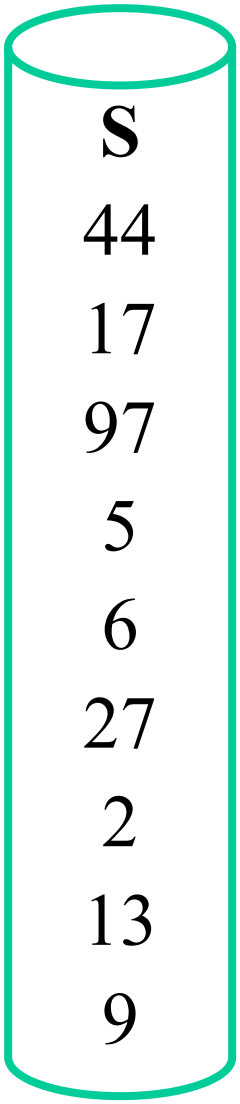
- Natürlich darf man nicht einzelne Datensätze zwischen Hauptspeicher und Hintergrundspeicher transferieren
  - Jeder „Round-Trip" kostet viel Zeit (ca 10 ms)
- Man transferiert größere Blöcke
  - Mindestens 8 KB Größe
- Replacement Selection ist problematisch, wenn die zu sortierenden Datensätze variable 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

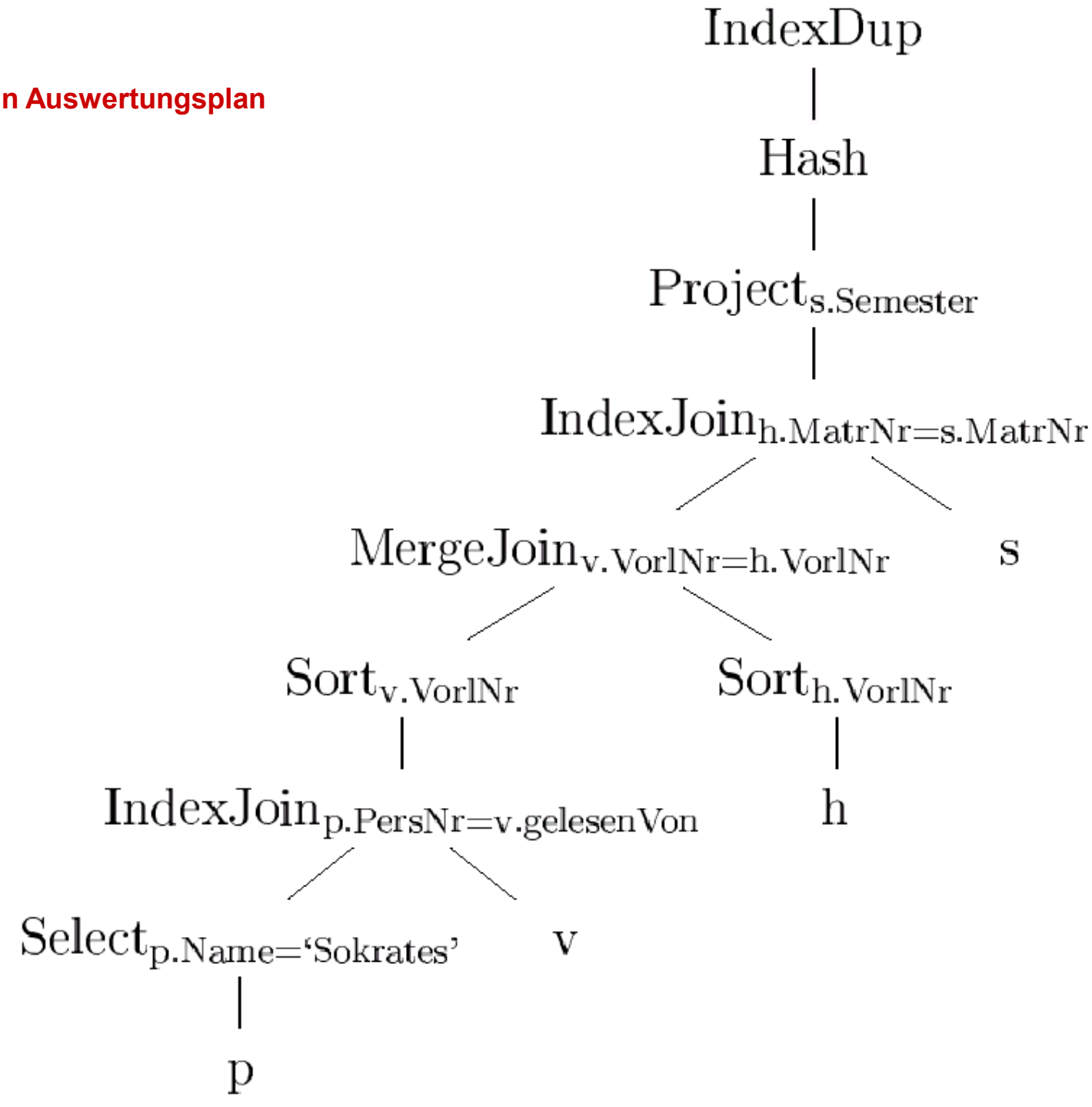


$$R \cap S$$

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

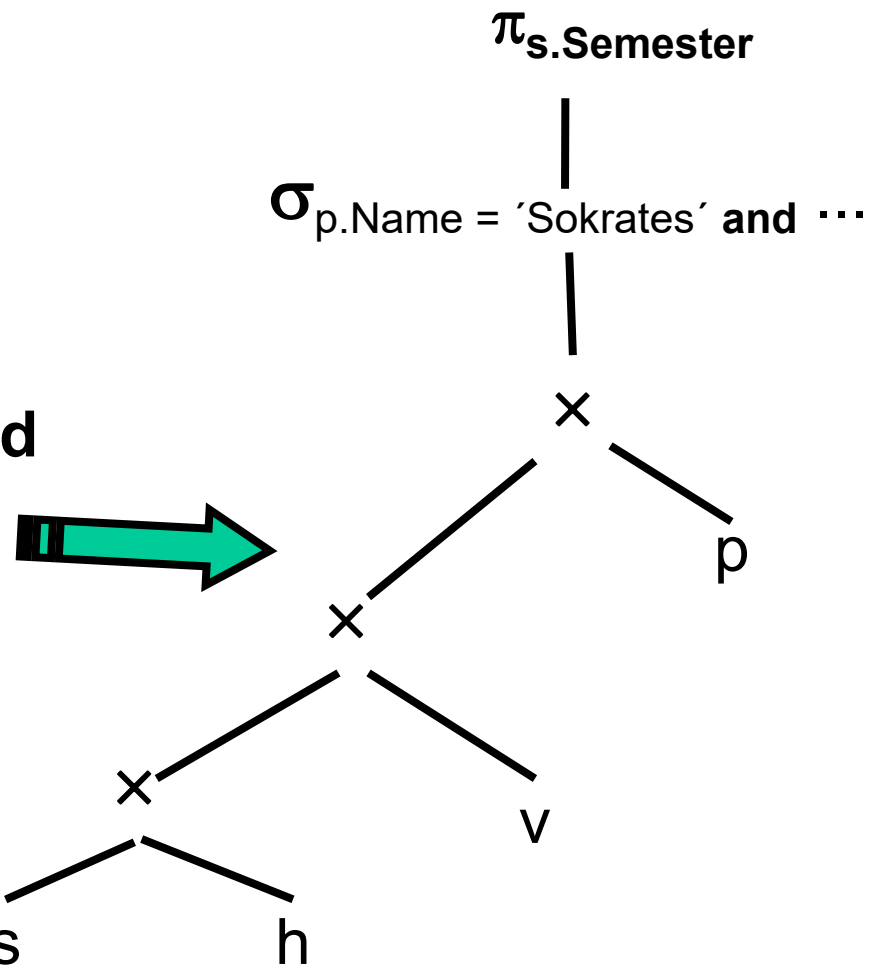


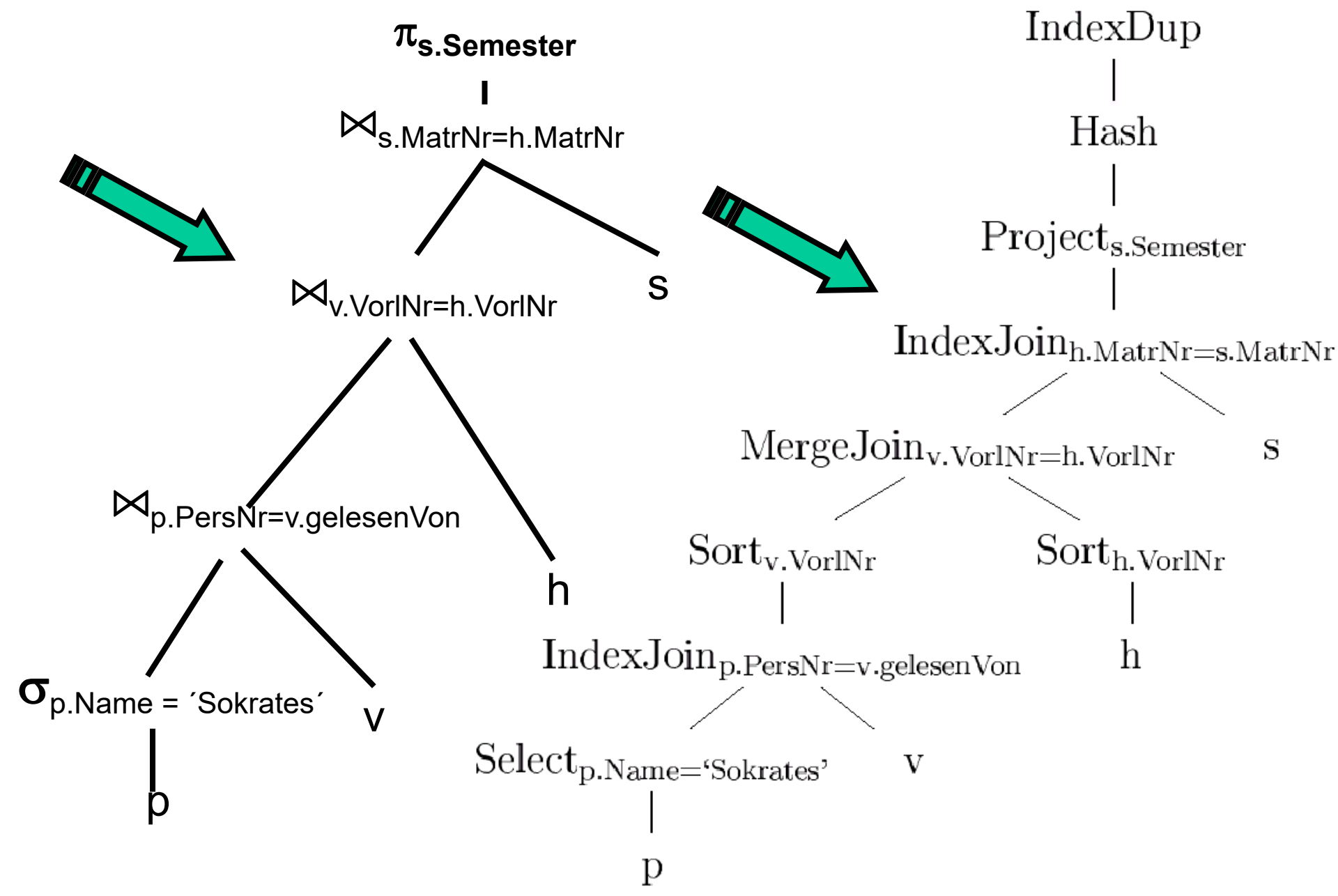
Ein Auswertungsplan



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
```

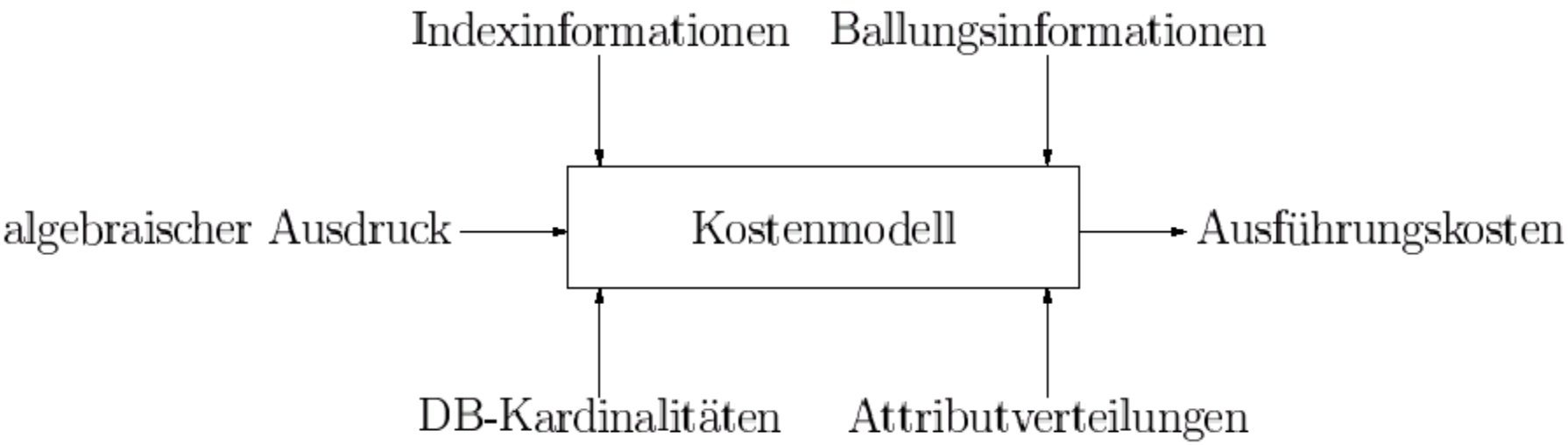



$\pi_{s.Semester}$
$\bowtie_{s.MatrNr=h.MatrNr}$
$\bowtie_{v.VorlNr=h.VorlNr}$
s
$\bowtie_{p.PersNr=v.gelesenVon}$
h
$\sigma_{p.Name = ´Sokrates´}$
v
p
IndexDup
Hash
$Project_{s.Semester}$
$IndexJoin_{h.MatrNr=s.MatrNr}$
$MergeJoin_{v.VorlNr=h.VorlNr}$
s
$Sort_{v.VorlNr}$
$Sort_{h.VorlNr}$
$IndexJoin_{p.PersNr=v.gelesenVon}$
h
$Select_{p.Name='Sokrates'}$
v
p

Kostenbasierte Optimierung

- Generiere alle denkbaren Anfrageauswertungspläne
  - 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

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:

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

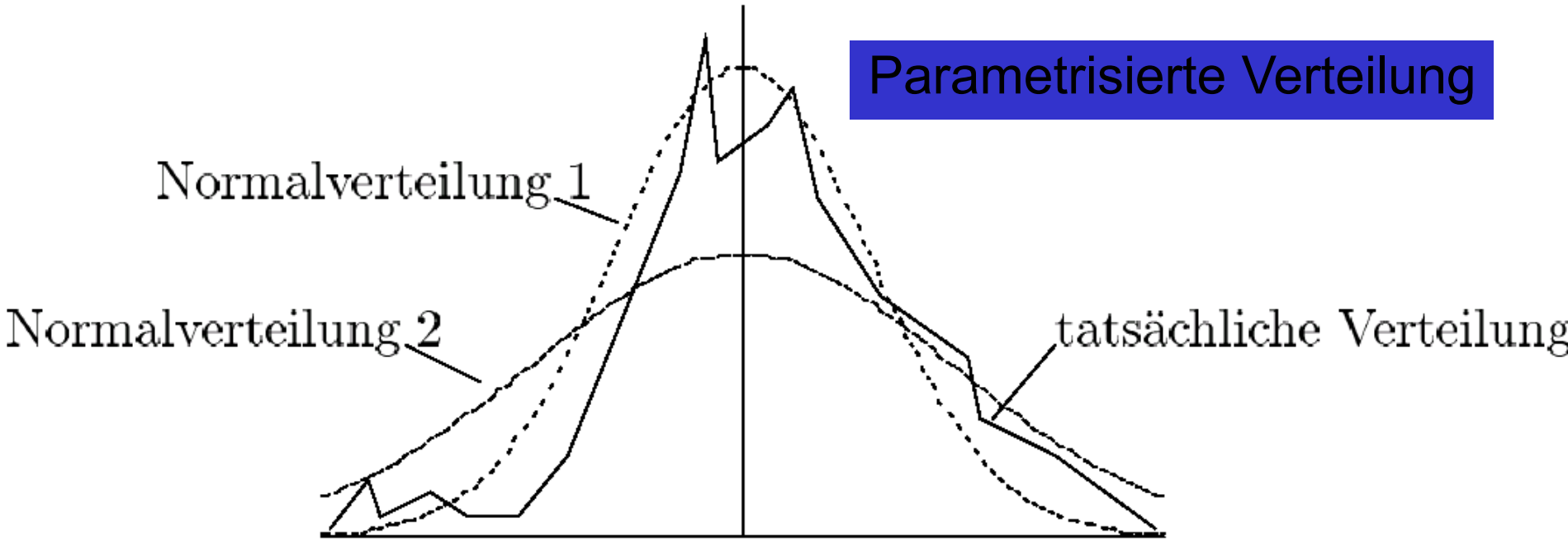
Abschätzung für einfache Fälle

Abschätzung der Selektivität:

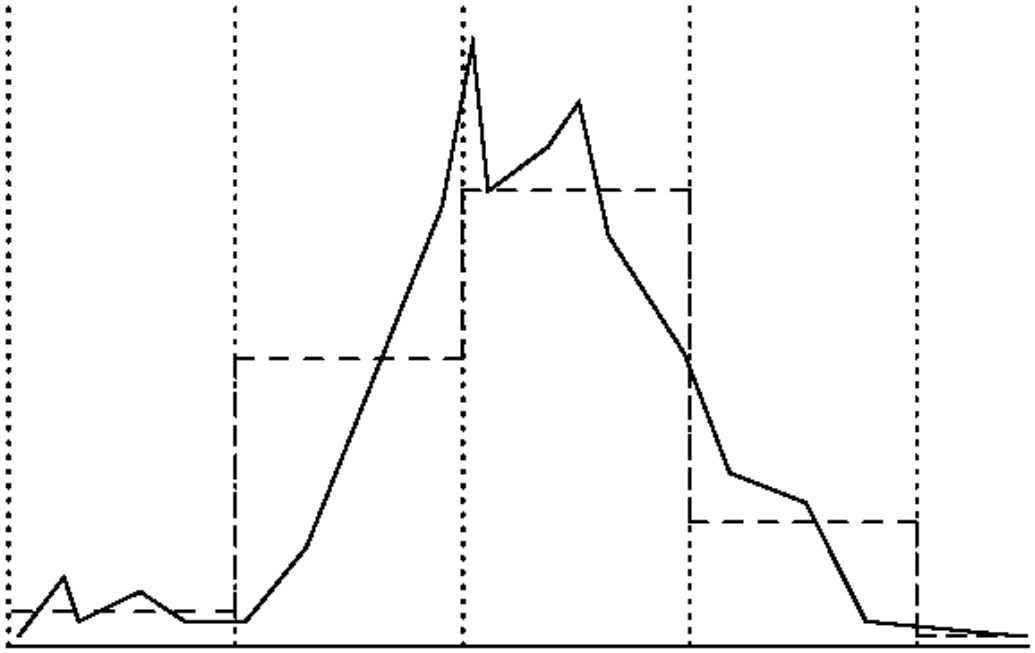
- $sel_{R.A=C} = \frac{1}{|R|}$
  falls A Schlüssel von R
- $sel_{R.A=C} = \frac{1}{i}$
  falls i die Anzahl der Attributwerte von $R.A$ ist (Gleichverteilung)
- $sel_{R.A=S.B} = \frac{1}{|R|}$
  bei Equijoin von R mit S über Fremdschlüssel in S

Ansonsten z.B. Stichprobenverfahren

Parametrisierte Verteilung



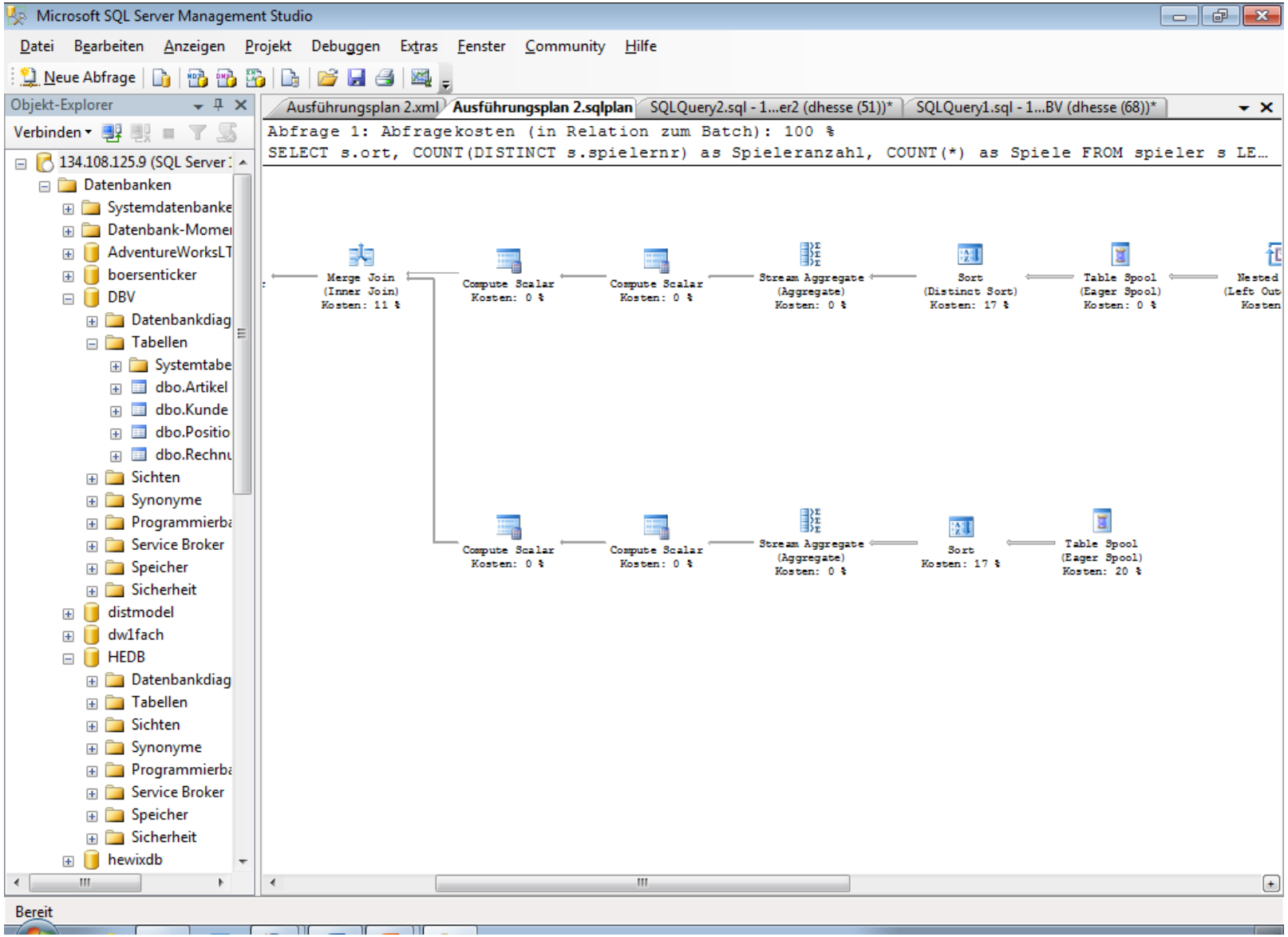
Histogramm



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 …
- In MSSQL
  - update statistics
- In MySQL
  - analyze table

Ein Auswertungsplan



```
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