Version March 4, 2011

Introduction to Database Systems
CSE 444, Winter 2011

Lectures 21-23: Query Optimization

http://www.cs.washington.edu/education/courses/cse444/11wi/

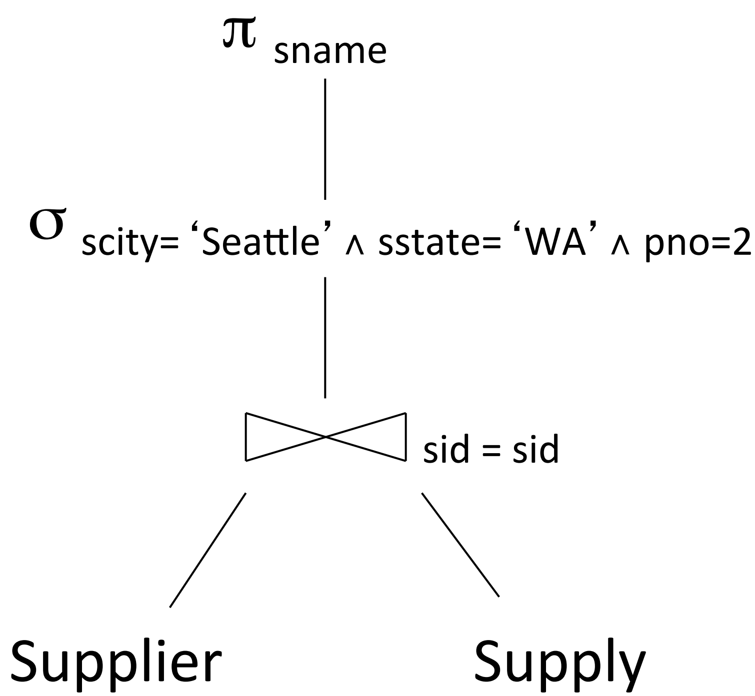
Where we are / and where we go

Feb 7	Transactions: Concurrency Control lecture 14-15 Midterm review on the board	Midterm	Data Storage and Indexing lecture 16 **Homework 2 due**
Feb 14	**Database Tuning** lecture 17	**Relational Algebra** lecture 18	**Query Processing Overview** lecture 19 **Project 3 due**
Feb 21	**No class (Presidents Day)**	**Operator Algorithms**	**Query Optimization** **Homework 3 due**
Feb 28	**Query Optimization**	**Query Optimization**	Parallel and Distributed DBMSs
Mar 7	Pig Latin	TBA	Wrap-up Project 4 due
Mar 14	Final Exam Thursday, March 17, 8:30am-10:20am, in class		

Review Relational Algebra

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

```
SELECT  sname
FROM    Supplier x, Supply y
WHERE   x.sid = y.sid
   and  y.pno = 2
   and  x.scity = 'Seattle'
   and  x.sstate = 'WA'
```



Give a relational algebra expression for this query:

$$\pi_{\text{sname}}(\sigma_{\text{scity= 'Seattle'} \wedge \text{sstate= 'WA'} \wedge \text{pno=2}}(\text{Supplier} \bowtie_{\text{sid = sid}} \text{Supply}))$$

Key Idea: Algebraic Optimization

$$N = ((z*2)+((z*3)+y))/x$$

Given x = 1, y = 0, and z = 4, solve for N.

In what order did you perform the operations?

And how many operations?

Key Idea: Algebraic Optimization

$N = ((z*2)+((z*3)+0))/1$

Given x = 1, y = 0, and z = 4, solve for N again, but now assume:

* costs 10 units

+ costs 2 units

/ costs 50 units

Which execution plan offers the lowest cost?

Key Idea: Algebraic Optimization

N = ((z*2)+((z*3)+0))/1

Algebraic Laws:

1. (+) identity: x+0 = x
2. (/) identity: x/1 = x
3. (*) distributes: (n*x+n*y) = n*(x+y)
4. (*) commutes: x*y = y*x

Apply rules 1, 3, 4, 2:

N = (2+3)*z

two operations instead of five, no division operator

Optimization with Relational Algebra

```
SELECT sname
FROM   Supplier x, Supply y
WHERE  x.sid = y.sid
   and y.pno = 2
   and x.scity = 'Seattle'
   and x.sstate = 'WA'
```

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

$\pi_{sname}(\sigma_{scity=\text{'Seattle'} \wedge sstate=\text{'WA'} \wedge pno=2}(\text{Supplier} \bowtie_{sid = sid} \text{Supply}))$

Here is a different relational algebra expression for this query:

$\pi_{sname}((\sigma_{scity=\text{'Seattle'} \wedge sstate=\text{'WA'}}\text{Supplier}) \bowtie_{sid = sid} (\sigma_{pno=2}\text{Supply}))$

Query Optimization Goal:
For a query, there may exist many logical and physical query plans. Query Optimizer needs to pick a "good" one.

Hands-on Example

```
SELECT  sname
FROM    Supplier x, Supply y
WHERE   x.sid = y.sid
    and y.pno = 2
    and x.scity = 'Seattle'
    and x.sstate = 'WA'
```

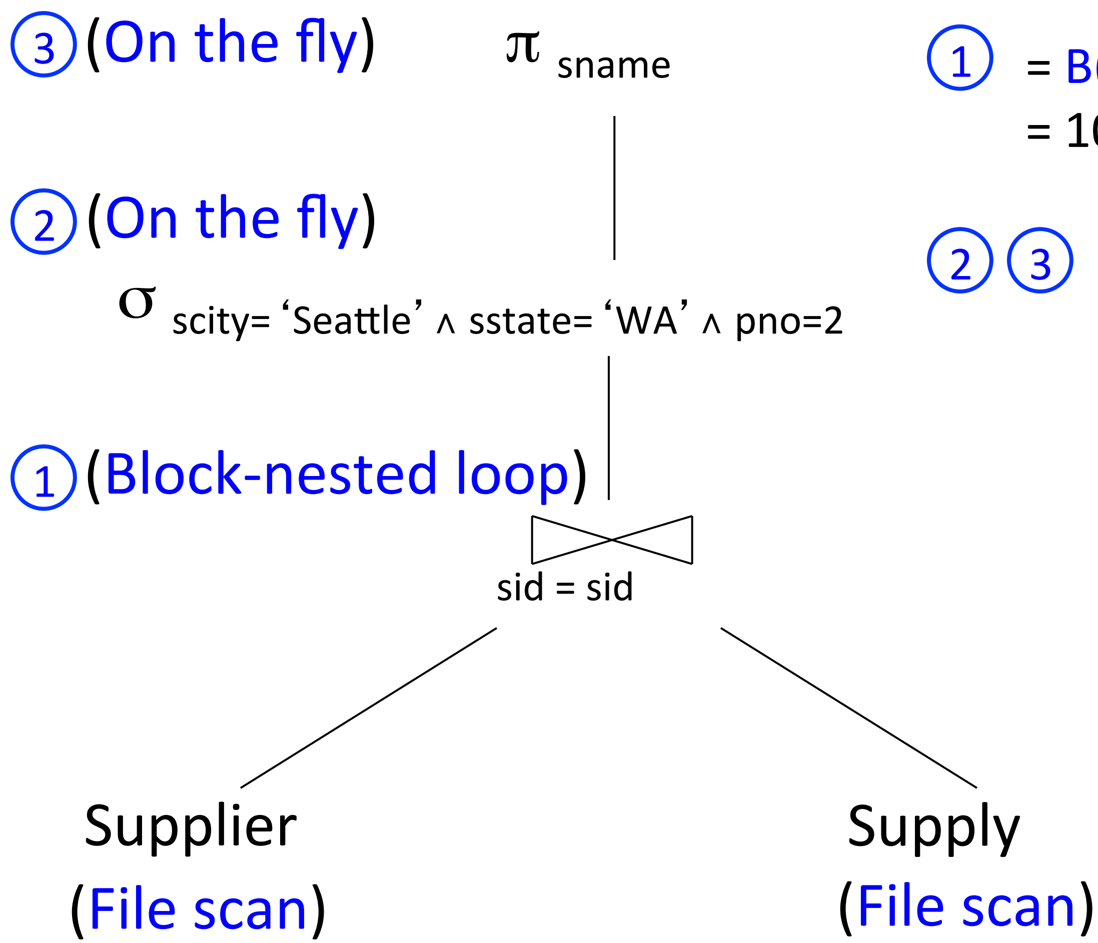
Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

- Some statistics
  - T(Supplier) = 1000 records
  - T(Supply) = 10,000 records
  - B(Supplier) = 100 pages
  - B(Supply) = 100 pages
  - V(Supplier,scity) = 20
  - V(Supplier,state) = 10
  - V(Supply,pno) = 2,500
  - Both relations are clustered
  - M = 10

Physical Query Plan 1

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

T(Supplier) = 1,000 B(Supplier) = 100 V(Supplier, scity) = 20 M = 10
T(Supply) = 10,000 B(Supply) = 100 V(Supplier, state) = 10
V(Supply, pno) = 2,500



① = B(Supplier)+B(Supplier)•B(Supply)/M
= 100 + 100•100/10 = 1,100 I/Os

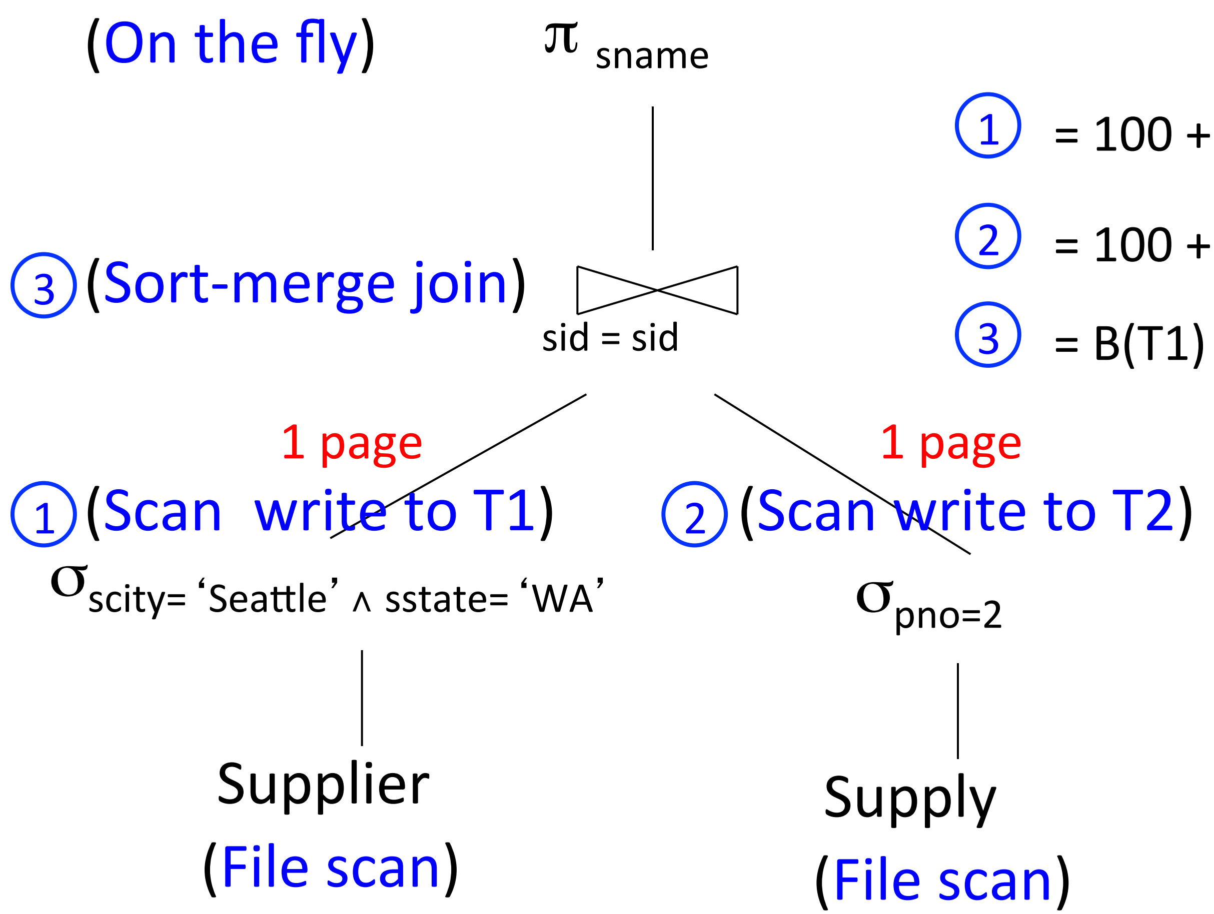
② ③ Selection and project on-the-fly
-> No additional cost.

Cost = **1,100** I/Os

Physical Query Plan 2

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

T(Supplier) = 1,000 B(Supplier) = 100 V(Supplier, scity) = 20 M = 10
T(Supply) = 10,000 B(Supply) = 100 V(Supplier, state) = 10
V(Supply, pno) = 2,500



Independence assumption

① $= 100 + 100 \cdot 1/20 \cdot 1/10 = 100.5 \approx 101$

② $= 100 + 100 \cdot 1/2500 \approx 101$

③ $= B(T1) + B(T2) = 1 + 1 = 2$

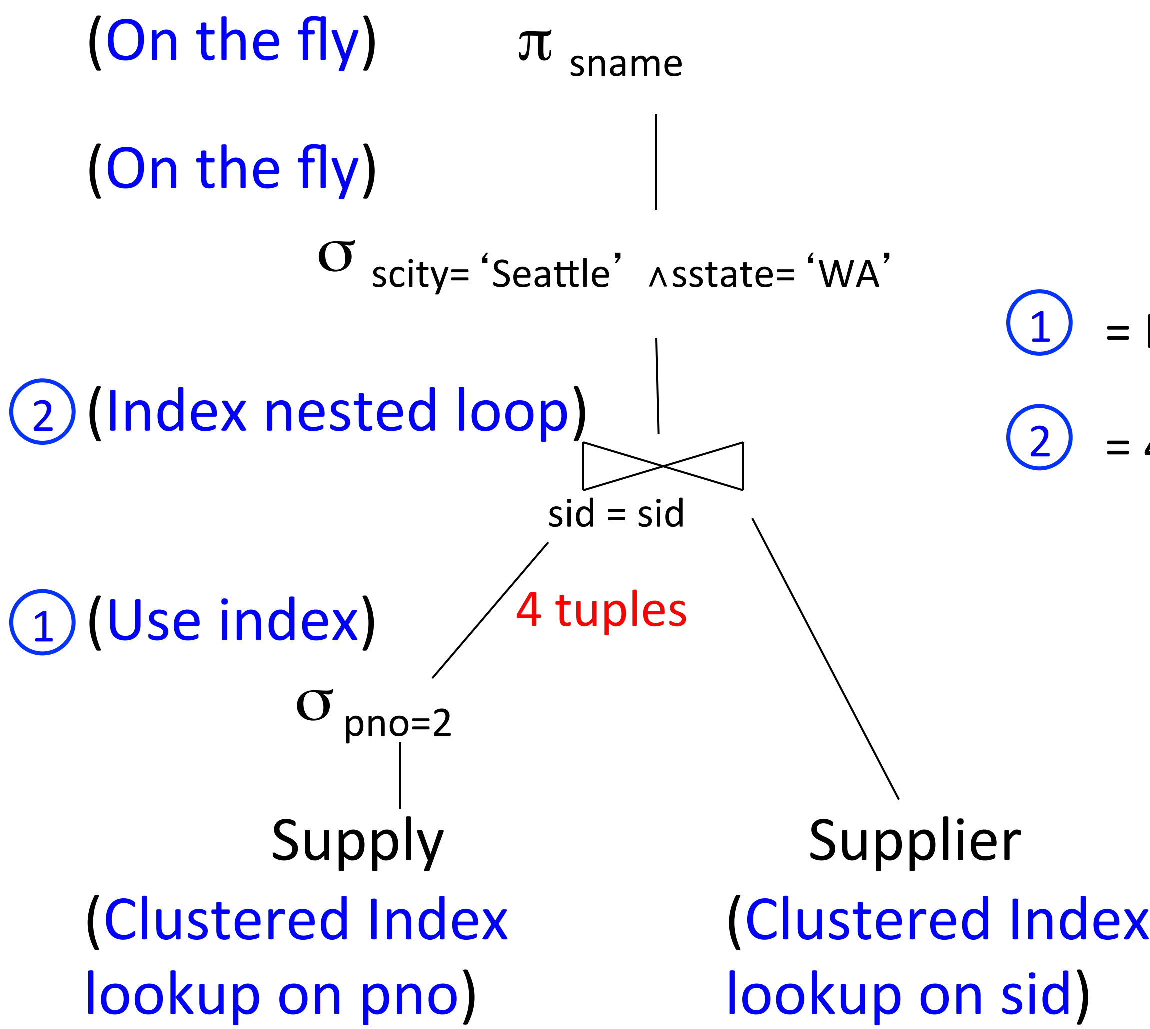
Cost ≈ **204** I/Os

x

Physical Query Plan 3

Supplier(<u>sid</u>, sname, scity, sstate)
Supply(<u>sid, pno</u>, quantity)

T(Supplier) = 1,000 B(Supplier) = 100 V(Supplier, scity) = 20 M = 10
T(Supply) = 10,000 B(Supply) = 100 V(Supplier, state) = 10
V(Supply, pno) = 2,500



① = B(Supply)/V(Supply,pno) ≈ 1

② = 4

each of the 4 tuples will have a different sid

Cost ≈ **5** I/Os

x

Simplifications

- In the previous examples, we assumed that all index pages were in memory
- When this is not the case, we need to add the cost of fetching index pages from disk

Query Optimization Goal / Algorithm

- Query Optimization Goal
  - For a query, there exist many logical and physical plans. Query optimizer needs to pick a good one. How?

- Query Optimization Algorithm
  - Enumerate alternative plans
  - Compute estimated cost of each plan
    - Compute both number of I/Os, and CPU cost
  - Choose plan with lowest cost
    - This is called cost-based optimization

Lessons

- Need to consider several physical plan
  - even for one, simple logical plan
- No magic "best" plan: depends on the data
- In order to make the right choice
  - need to have **statistics** over the data
  - the B's, the T's, the V's

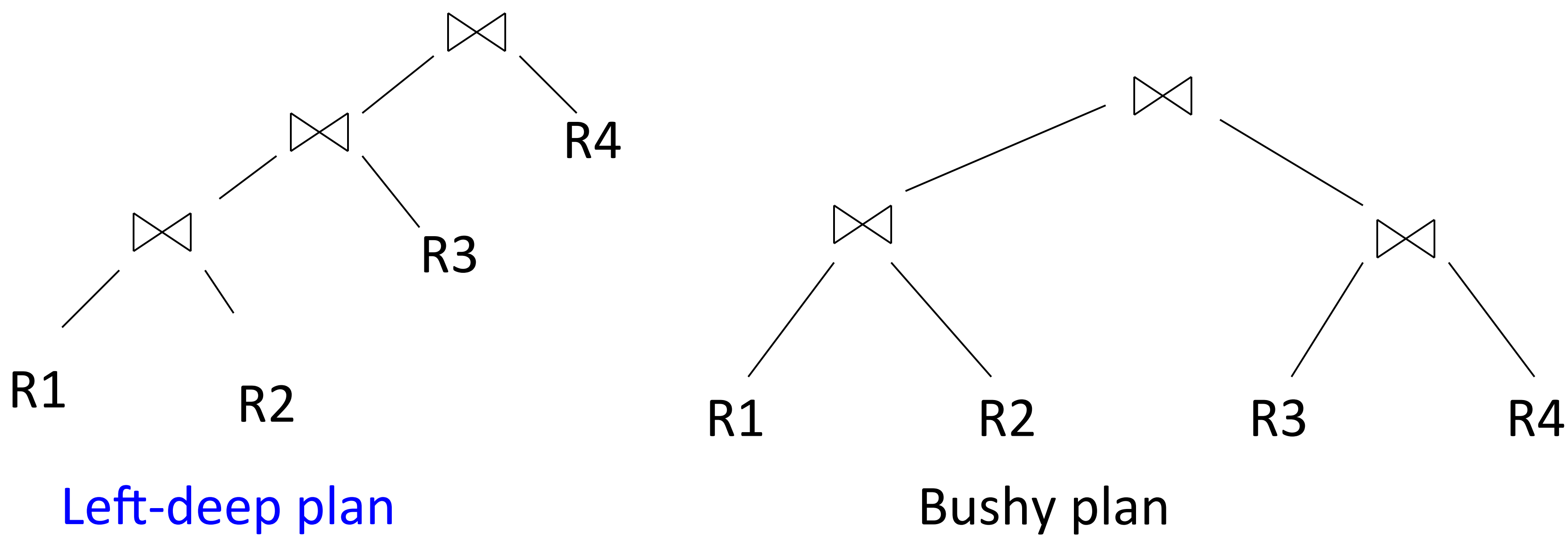
Outline

- **Search space**
- Algorithm for enumerating query plans
- Estimating the cost of a query plan

Relational Algebra Equivalences

- Selections
  - Commutative: $\sigma_{c1}(\sigma_{c2}(R))$ same as $\sigma_{c2}(\sigma_{c1}(R))$
  - Cascading: $\sigma_{c1 \wedge c2}(R)$ same as $\sigma_{c2}(\sigma_{c1}(R))$
- Projections
  - projections can be added as long as all attributes are kept that are used in later operators or the results
- Joins
  - Commutative : $R \bowtie S$ same as $S \bowtie R$
  - Associative: $R \bowtie (S \bowtie T)$ same as $(R \bowtie S) \bowtie T$

Left-Deep Plans and Bushy Plans



4 relations:

- # different tree shapes = 5
- # different orders = 4! = 24
- # different join trees = 5 * 24 = 120

x

Commutativity, Associativity, Distributivity

$R \cup S = S \cup R$, $R \cup (S \cup T) = (R \cup S) \cup T$
$R \bowtie S = S \bowtie R$, $R \bowtie (S \bowtie T) = (R \bowtie S) \bowtie T$

$R \bowtie (S \cup T) = (R \bowtie S) \cup (R \bowtie T)$

Example

Which plan is more efficient:
$R \bowtie (S \bowtie T)$ or $(R \bowtie S) \bowtie T$?

- Assumptions:
  Note: sometimes defined differently!
  - Every join selectivity is 10%
    - That is: $T(R \bowtie S) = 0.1 * T(R) * T(S)$ etc.
  - B(R)=100, B(S) = 50, B(T)=500
  - All joins are main memory joins
  - All intermediate results are materialized

Laws involving selection:

$$\sigma_{C \text{ AND } C'}(R) = \sigma_C(\sigma_{C'}(R)) = \sigma_C(R) \cap \sigma_{C'}(R)$$
$$\sigma_{C \text{ OR } C'}(R) = \sigma_C(R) \cup \sigma_{C'}(R)$$

$$\sigma_C(R - S) = \sigma_C(R) - S$$
$$\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$$
$$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$$

When C involves
only attributes of R

Example: Simple Algebraic Laws

- Example: R(A, B, C, D), S(E, F, G)

$\sigma_{F=3} (R \bowtie_{D=E} S) = ?$

$= R \bowtie_{D=E} (\sigma_{F=3} S)$

$\sigma_{A=5 \text{ AND } G=9} (R \bowtie_{D=E} S) = ?$

$= \sigma_{A=5} (\sigma_{G=9} (R \bowtie_{D=E} S))$

$= (\sigma_{A=5} R) \bowtie_{D=E} (\sigma_{G=9} S))$

Laws Involving Projections

$$\Pi_M(R \bowtie S) = \Pi_M(\Pi_P(R) \bowtie \Pi_Q(S))$$

$$\Pi_M(\Pi_N(R)) = \Pi_M(R)$$

/* note that $M \subseteq N$ */

- Example R(A,B,C,D), S(E, F, G)

$$\Pi_{A,B,G}(R \bowtie_{D=E} S) = \Pi_{?}(\Pi_{?}(R) \bowtie_{D=E} \Pi_{?}(S))$$

Laws involving grouping and aggregation

$$\delta(\gamma_{A,\ agg(B)}(R)) = \gamma_{A,\ agg(B)}(R)$$

$$\gamma_{A,\ agg(B)}(\delta(R)) = \gamma_{A,\ agg(B)}(R)$$

if agg is "duplicate insensitive"

Which of the following are "duplicate insensitive" ?
sum, count, avg, min, max

Laws Involving Constraints

Product(pid, pname, price, cid)
Company(cid, cname, city, state)

Foreign key

$$\Pi_{pid,\ price}(\text{Product} \bowtie_{cid=cid} \text{Company}) = \Pi_{pid,\ price}(\text{Product})$$

Need a second constraint for this law to hold. Which ?

Example

Product(pid, pname, price, cid)
Company(cid, cname, city, state)

Foreign key
& not null

```
CREATE VIEW CheapProductCompany
    SELECT *
    FROM    Product x, Company y
    WHERE  x.cid = y.cid and x.price < 100
```

```
SELECT  pname, price
FROM    CheapProductCompany
```

⇒

```
SELECT pname, price
FROM    Product
WHERE  price < 100
```

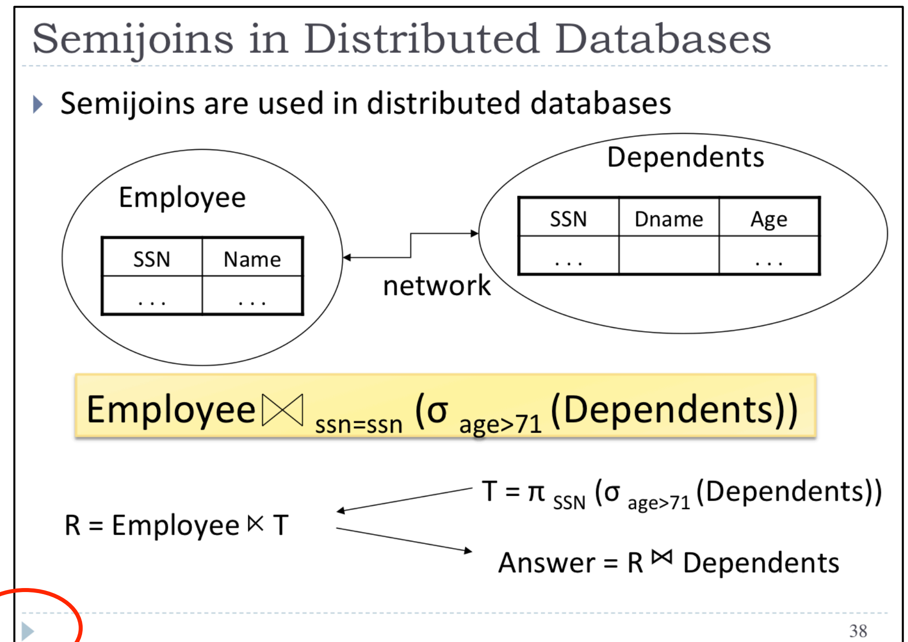

x

Laws with Semijoins

Recall the definition of a semijoin:

- $R \ltimes S = \Pi_{A1,\ldots,An}(R \bowtie S)$
- Where the schemas are:
  - Input: R(A1,…An), S(B1,…,Bm)
  - Output: T(A1,…,An)

Remember from lecture 18:



$R \bowtie S = (R \ltimes S) \bowtie S$

Observe the "dangling" triangle,
doesn't "join" with any content,
poor lonely triangle ☹

Laws with Semijoins

- Example:

$$Q = R(A,B) \bowtie S(B,C)$$

- A reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$

- The rewritten query is:

$$Q = R_1(A,B) \bowtie S(B,C)$$

$R \bowtie S = (R \ltimes S) \bowtie S$

Why else would we do this ?

Why Would We Do This ?

- Large attributes:

$$Q = R(A, B, D, E, F,\ldots) \bowtie S(B, C, M, K, L, \ldots)$$

- Expensive side computations

$$Q = (\gamma_{A,B,count(*)} R(A,B,D)) \bowtie (\sigma_{C=value} S(B,C))$$

$$R_1(A,B,D) = R(A,B,D) \ltimes \sigma_{C=value}(S(B,C))$$
$$Q = (\gamma_{A,B,count(*)} R_1(A,B,D)) \bowtie (\sigma_{C=value} S(B,C))$$

x

Laws with Semijoins

- Example:

$$Q = R(A,B) \bowtie S(B,C)$$

- A reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$

- The rewritten query is:

$$Q = R_1(A,B) \bowtie S(B,C)$$

Are there dangling tuples ?

Laws with Semijoins

- Example:

$$Q = R(A,B) \bowtie S(B,C)$$

- A full reducer is:

$$R_1(A,B) = R(A,B) \ltimes S(B,C)$$
$$S_1(B,C) = S(B,C) \ltimes R_1(A,B)$$

- The rewritten query is:

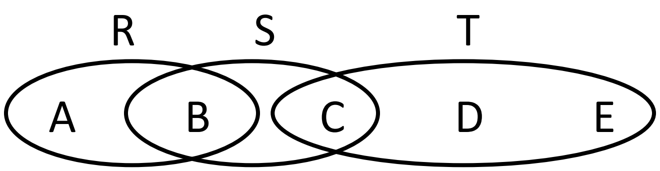
$$Q = R_1(A,B) \bowtie S_1(B,C)$$

No more dangling tuples

Laws with Semijoins

- More complex example:

$$Q = R(A,B) \bowtie S(B,C) \bowtie T(C,D,E)$$



Query Hypergraph

- A full reducer is:

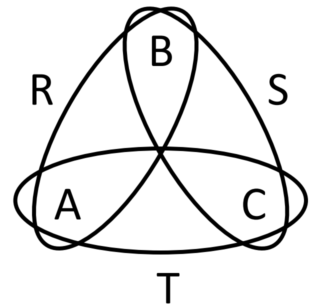
$$S'(B,C) = S(B,C) \ltimes R(A,B)$$
$$T'(C,D,E) = T(C,D,E) \ltimes S'(B,C)$$
$$S''(B,C) = S'(B,C) \ltimes T'(C,D,E)$$
$$R'(A,B) = R(A,B) \ltimes S''(B,C)$$

$$Q = R'(A,B) \bowtie S''(B,C) \bowtie T'(C,D,E)$$

Laws with Semijoins

- Example:

  $Q = R(A,B) \bowtie S(B,C) \bowtie T(A,C)$



Cyclic Query Hypergraph

- Doesn't have a full reducer (we can reduce forever)

- **Theorem:** A query has a full reducer iff it is acyclic (see Chapter 20.4)
  - (if interested, you find the proof in the book [1995, Database Theory, by Abiteboul, Hull, Vianu])

Laws with Semijoins

After all the examples, now the overview slide at the end!

Semijoins

- Given a query: $Q = R_1 \bowtie R_2 \bowtie \ldots \bowtie R_n$
- A semijoin reducer for Q is

$$R_{i1} = R_{i1} \ltimes R_{j1}$$
$$R_{i2} = R_{i2} \ltimes R_{j2}$$
$$\ldots\ldots$$
$$R_{ip} = R_{ip} \ltimes R_{jp}$$

- such that the query is equivalent to:

$$Q = R_{k1} \bowtie R_{k2} \bowtie \ldots \bowtie R_{kn}$$

- A full reducer is such that no dangling tuples remain

Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

[PODS'98, by Chaudhuri]

View:

```
CREATE VIEW DepAvgSal As (
    SELECT   E.did, Avg(E.Sal) AS avgsal
    FROM     Emp E
    GROUP BY E.did)
```

Query:

```
SELECT   E.eid, E.sal
FROM     Emp E, Dept D, DepAvgSal V
WHERE    E.did = D.did and D.budget > 100k
    and  E.age < 30 and E.did = V.did
    and  E.sal > V.avgsal
```

Goal: compute only the necessary part of the view

Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

[PODS'98, by Chaudhuri]

New view uses a reducer:

```
CREATE VIEW LimitedAvgSal As (
    SELECT   E.did, Avg(E.Sal) AS avgsal
    FROM     Emp E, Dept D
    WHERE    E.did = D.did and D.buget > 100k
    GROUP BY E.did)
```

New Query:

```
SELECT   E.eid, E.sal
FROM     Emp E, Dept D, LimitedAvgSal V
WHERE    E.did = D.did and D.budget > 100k
   and   E.age < 30 and E.did = V.did
   and   E.sal > V.avgsal
```

Example with Semijoins

Emp(eid, ename, sal, did)
Dept(did, dname, budget)
DeptAvgSal(did, avgsal) /* view */

[PODS'98, by Chaudhuri]

Full reducer:

```
CREATE VIEW PartialResult AS
    (SELECT  E.eid, E.sal, E.did
     FROM    Emp E, Dept D
     WHERE   E.did=D.did and E.age < 30
        and  D.budget > 100k)

CREATE VIEW Filter AS
    (SELECT DISTINCT P.did FROM PartialResult P)

CREATE VIEW LimitedDepAvgSal AS
    (SELECT  E.did, Avg(E.Sal) AS avgsal
     FROM    Emp E, Filter F
     WHERE   E.did = F.did
     GROUP BY E.did)
```

Example with Semijoins

New query:

```
SELECT   P.eid, P.sal
FROM     PartialResult P, LimitedDepAvgSal V
WHERE    P.did = V.did
  and    P.sal > V.avgsal
```

Original query:

```
SELECT   E.eid, E.sal
FROM     Emp E, Dept D, DepAvgSal V
WHERE    E.did = D.did and E.did = V.did
   and   E.age < 30 and D.budget > 100k
   and   E.sal > V.avgsal
```

Search Space Challenges

- Search space is huge!
  - Many possible equivalent trees
  - Many implementations for each operator
  - Many access paths for each relation
    - File scan or index + matching selection condition
- Cannot consider ALL plans
  - Heuristics: only partial plans with "low" cost

Outline

- Search space
- **Algorithm for enumerating query plans**
- Estimating the cost of a query plan

Key Decisions

- Logical plan
  - What logical plans do we consider (left-deep, bushy ?)
    Search Space
  - Which algebraic laws do we apply, and in which context(s) ?
    Optimization rules
  - In what order do we explore the search space ?
    Optimization algorithm

- Physical plan
  - What physical operators to use?
  - What access paths to use (file scan or index)?

Optimizers

- Heuristic-based optimizers:
  - Apply greedily rules that always improve
    - Typically: push selections down
  - Very limited: no longer used today
- Cost-based optimizers
  - Use a cost model to estimate the cost of each plan
  - Select the "cheapest" plan

The Search Space

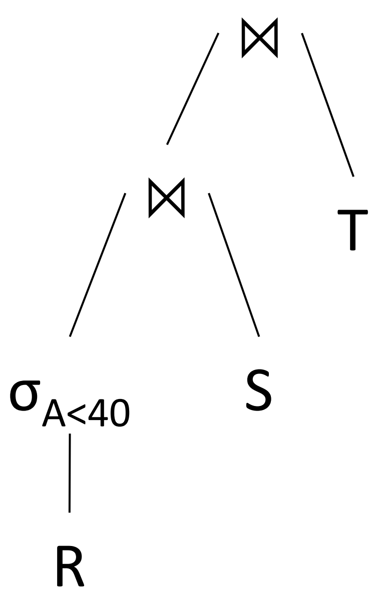
- 1. Complete plans
- 2. Bottom-up plans
- 3. Top-down plans

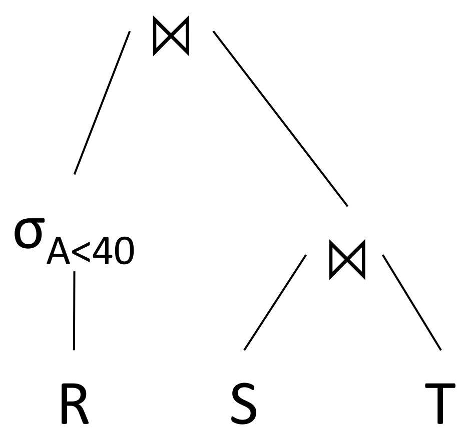
Seach Space 1: Complete Plans

```
SELECT   *
FROM     R, S, T
WHERE    R.B=S.B
    and  S.C=T.C
    and  R.A<40
```

R(A,B)
S(B,C)
T(C,D)

Why is this search space inefficient ?





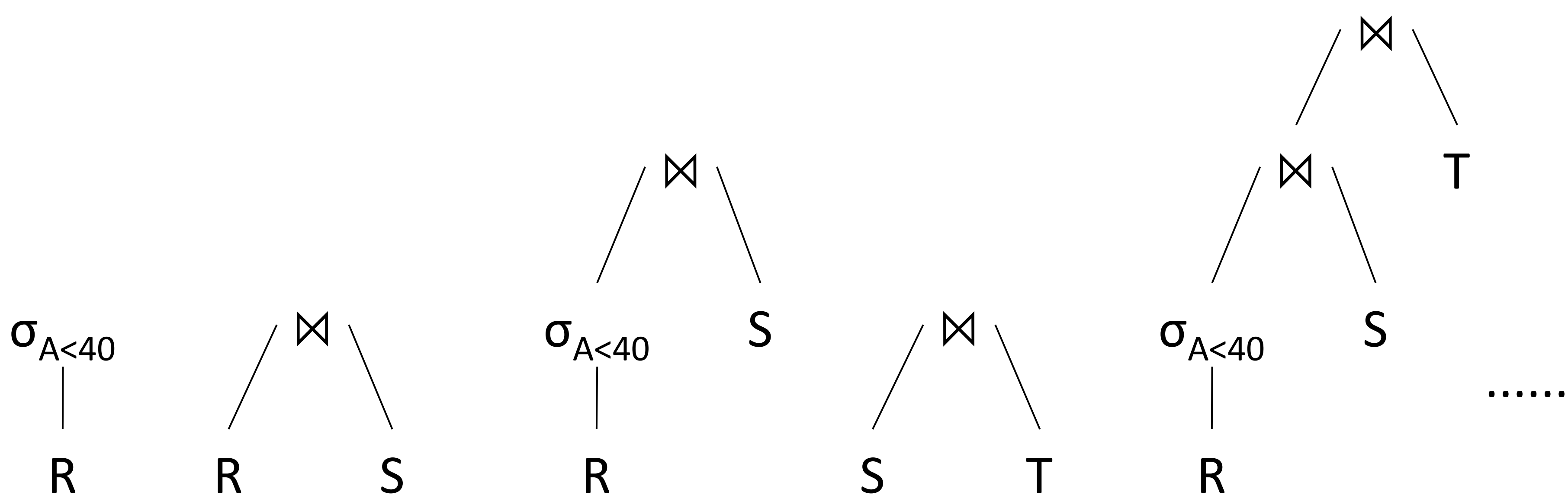
......

Seach Space 2: Bottom-up Partial Plans

```
SELECT   *
FROM     R, S, T
WHERE    R.B=S.B
    and  S.C=T.C
    and  R.A<40
```

R(A,B)
S(B,C)
T(C,D)

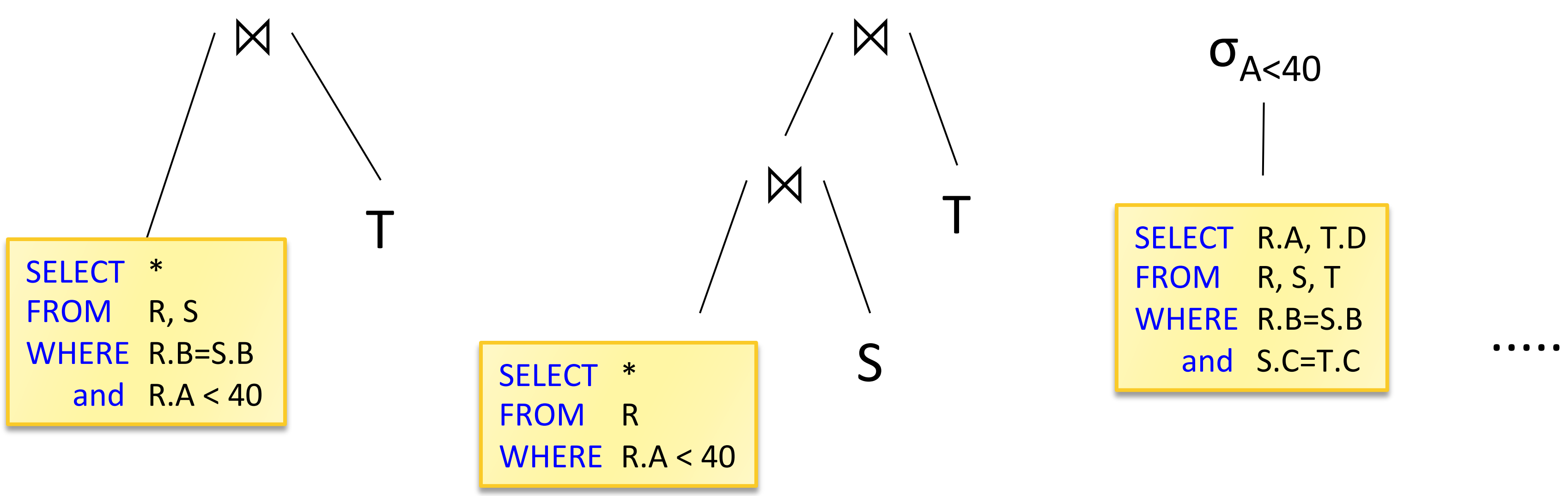
Why is this better ?



Seach Space 3: Top-down Partial Plans

```
SELECT   *
FROM     R, S, T
WHERE    R.B=S.B
    and  S.C=T.C
    and  R.A<40
```

R(A,B)
S(B,C)
T(C,D)



Plan Enumeration Algorithms

- Dynamic programming (in class)
  - Classical algorithm [1979]
  - Limited to joins: join reordering algorithm
  - Bottom-up

- Rule-based algorithm (will not discuss)
  - Database of rules (=algebraic laws)
  - Usually: dynamic programming
  - Usually: top-down

Dynamic Programming

Originally proposed in System R [1979]

- Only handles single block queries:

```
SELECT   list
FROM     R1, …, Rn
WHERE    cond_1
   and   cond_2
   and   . . .
   and   cond_k
```

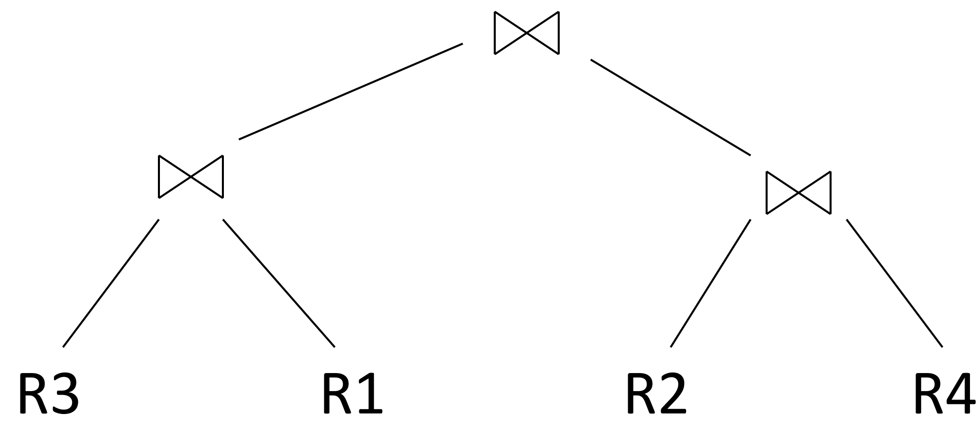
- Heuristics: selections down, projections up

Dynamic Programming

- Search space = join trees
- Algebraic laws = commutativity, associativity
- Algorithm = dynamic programming ☺

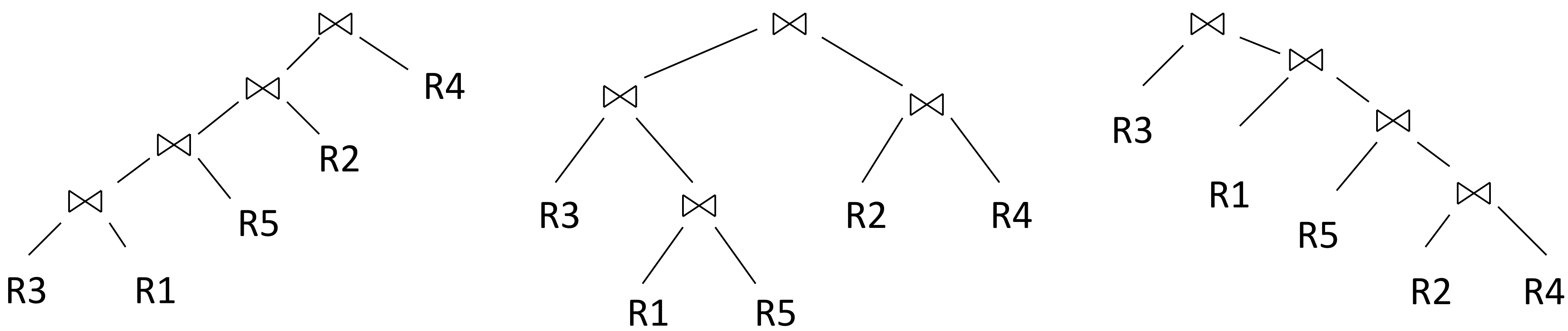
Join Trees

- $R1 \bowtie R2 \bowtie .... \bowtie Rn$
- Join tree:



- A plan = a join tree
- A partial plan = a subtree of a join tree

Types of Join Trees



Left-deep plan   Bushy plan   Right-deep plan

# relations	2	3	4	5	6	7
# tree shapes	1	2	5	14	42	132
# permutations	2	6	24	120	720	5040
# join trees	2	12	120	1680	>30k	>665k

x

Dynamic Programming

```
SELECT  list
FROM    R1, …, Rn
WHERE   cond1 and  . . . and condk
```

Join ordering:

- Given: a query R1 ⋈ R2 ⋈ . . . ⋈ Rn
- Find optimal order

- Assume we have a function cost() that gives us the cost of every join tree

Dynamic Programming

```
SELECT  list
FROM    R1, …, Rn
WHERE   cond_1 and . . . and cond_k
```

- For each subquery Q $\subseteq$ {R1, …, Rn} compute the following:
  - Size(Q) = the estimated size of Q
  - Plan(Q) = a best plan for Q
  - Cost(Q) = the estimated cost of that plan

Dynamic Programming

```
SELECT  list
FROM    R1, …, Rn
WHERE   cond1 and . . . and condk
```

- **Step 1**: For each $\{R_i\}$, set:
  - $Size(\{R_i\}) = B(R_i)$
  - $Plan(\{R_i\}) = R_i$
  - $Cost(\{R_i\})$ = (cost of scanning R_i)

Dynamic Programming

SELECT list
FROM R1, …, Rn
WHERE cond_1 and . . . and cond_k

- **Step 2**: For each Q $\subseteq \{R_1, \ldots, R_n\}$ involving *i* relations:
  - Size(Q) = estimate it recursively
  - For every pair of subqueries Q', Q'' s.t. Q = Q' ∪ Q'' compute cost(Plan(Q') ⋈ Plan(Q''))
    - Cost(Q) = the smallest such cost
    - Plan(Q) = the corresponding plan

What's a reasonable estimate?

- **Step 3**: Return Plan($\{R_1, \ldots, R_n\}$)

Example: $R \bowtie S \bowtie T \bowtie U$

To illustrate, ad-hoc cost model (from the book ☺):

- $Cost(P_1 \bowtie P_2) = Cost(P_1) + Cost(P_2) +$ size(intermediate results for P_1, P_2)
- Cost of a scan = 0

- Further assumptions:

All join selectivities = 1%

$T(R) = 2000$
$T(S) = 5000$
$T(T) = 3000$
$T(U) = 1000$

$T(R \bowtie S) = 0.01 * T(R) * T(S)$
$T(S \bowtie T) = 0.01 * T(S) * T(T)$
etc.

x

T(R) = 2000
T(S) = 5000
T(T) = 3000
T(U) = 1000

Subquery	Size	Cost	Plan
RS			
RT			
RU			
ST			
SU			
TU			
RST			
RSU			
RTU			
STU			
RSTU			

T(R) = 2000
T(S) = 5000
T(T) = 3000
T(U) = 1000

Subquery	Size	Cost	Plan
RS	100k	0	RS
RT	60k	0	RT
RU	20k	0	RU
ST	150k	0	ST
SU	50k	0	SU
TU	30k	0	TU
RST	3M	60k	(RT)S
RSU	1M	20k	(RU)S
RTU	0.6M	20k	(RU)T
STU	1.5M	30k	(TU)S
RSTU	30M	60k +50k=110k	(RT)(SU)

x

Reducing the Search Space

- Restriction 1: only linear trees (no bushy)

  Most systems restrict the search space to left-deep plans. Note that for some join algorithms, there exist different conventions about which is the build and which is the probe relation. The convention of our textbook (see example 16.31 p.818) assumes the build relation on the left, and hence calls right-deep plans those with several build relations in main memory. Don't let this detail confuse you. In practice it does not matter, as the optimizer does not actually "draw" these trees. The fact that they are linear is the only thing that matters.
- Restriction 2: no trees with cartesian product

$$R(A,B) \bowtie S(B,C) \bowtie T(C,D)$$

Plan: $(R(A,B) \bowtie T(C,D)) \bowtie S(B,C)$
has a cartesian product.
Most query optimizers will not consider it

x

Dynamic Programming: Summary

- Handles only join queries:
  - Selections are pushed down (i.e. early)
  - Projections are pulled up (i.e. late)
- Takes exponential time in general, BUT:
  - Left linear joins may reduce time
  - Non-cartesian products may reduce time further

Rule-Based Optimizers

- Extensible collection of rules
  - Rule = Algebraic law with a direction
- Algorithm for firing these rules
  - Generate many alternative plans, in some order
  - Prune by cost

- Volcano (later SQL Server)
- Starburst (later DB2)

Completing the Physical Query Plan

- Choose algorithm for each operator
  - How much memory do we have ?
  - Are the input operand(s) sorted ?
- Access path selection for base tables
- Decide for each intermediate result:
  - To materialize
  - To pipeline

Access Path Selection

- Access path: a way to retrieve tuples from a table
  - A file scan
  - An index *plus* a matching selection condition

- Index matches selection condition if it can be used to retrieve just tuples that satisfy the condition
  - Example: Supplier(sid,sname,scity,sstate)
  - B+-tree index on (scity,sstate)
    - matches scity='Seattle'
    - does not match sid=3, does not match sstate='WA'

Access Path Selection

- Relation: Supplier(sid,sname,scity,sstate)
- Selection condition: sid > 300 ∧ scity='Seattle'
- Indexes: B+-tree on sid and B+-tree on scity

- Which access path should we use?
  - We should pick the most selective access path

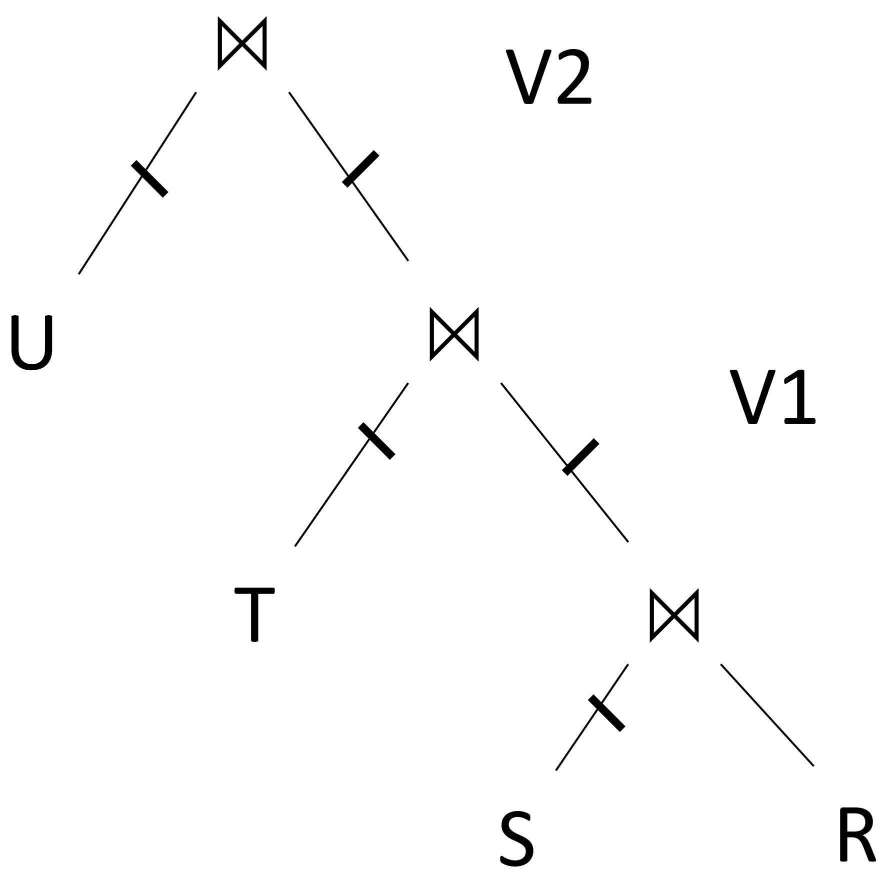
Access Path Selectivity

- Access path selectivity:
  - number of pages retrieved if we use this access path
  - Most selective retrieves fewest pages

- As we saw earlier, for equality predicates:
  - Selection on equality: $\sigma_{a=v}(R)$
  - V(R,a) = # of distinct values of attribute a
  - 1/V(R,a) is thus the reduction factor
  - Clustered index on a: cost B(R)/V(R,a)
  - Unclustered index on a: cost T(R)/V(R,a)
  - (we are ignoring I/O cost of index pages for simplicity)

x

Materialize Intermediate Results b/w Operators




```
HashTable ← S
repeat    read(R, x)
          y ← join(HashTable, x)
          write(V1, y)

HashTable ← T
repeat    read(V1, y)
          z ← join(HashTable, y)
          write(V2, z)

HashTable ← U
repeat    read(V2, z)
          u ← join(HashTable, z)
          write(Answer, u)
```

Convention of the book: build relations on the left.

x

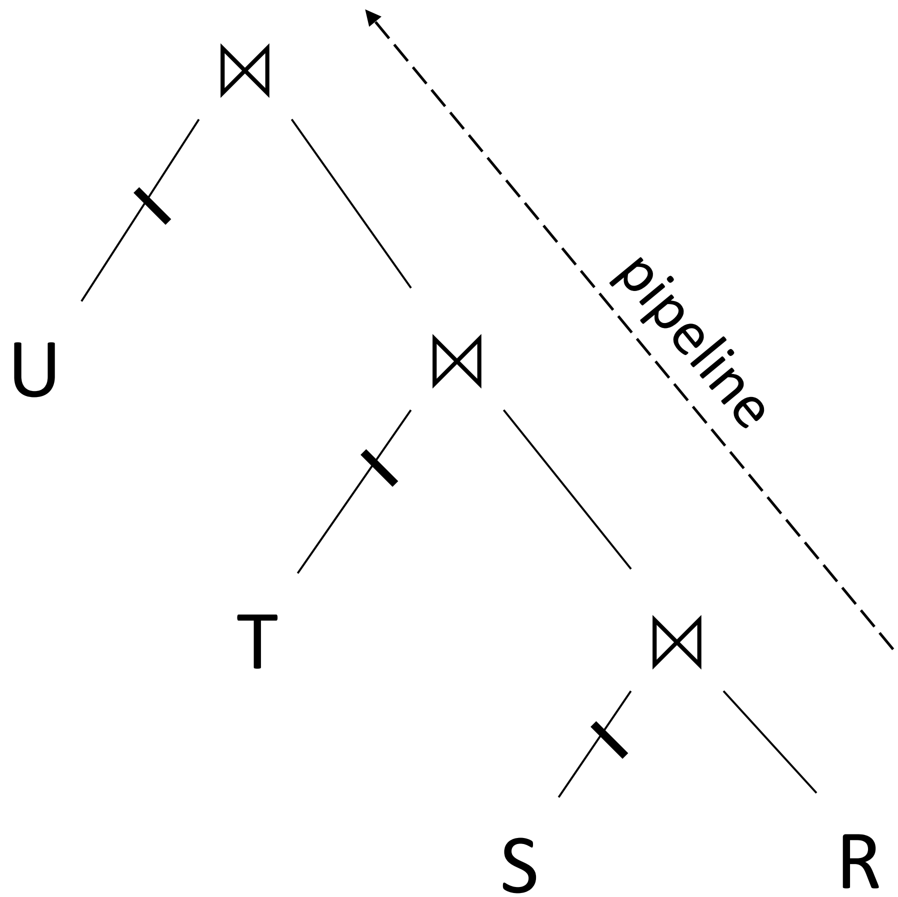
Materialize Intermediate Results b/w Operators

Question in class

Given B(R), B(S), B(T), B(U)

- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

Pipeline Between Operators




```
HashTable1 ← S
HashTable2 ← T
HashTable3 ← U
repeat    read(R, x)
          y ← join(HashTable1, x)
          z ← join(HashTable2, y)
          u ← join(HashTable3, z)
          write(Answer, u)
```

Convention of the book: build relations on the left.

x

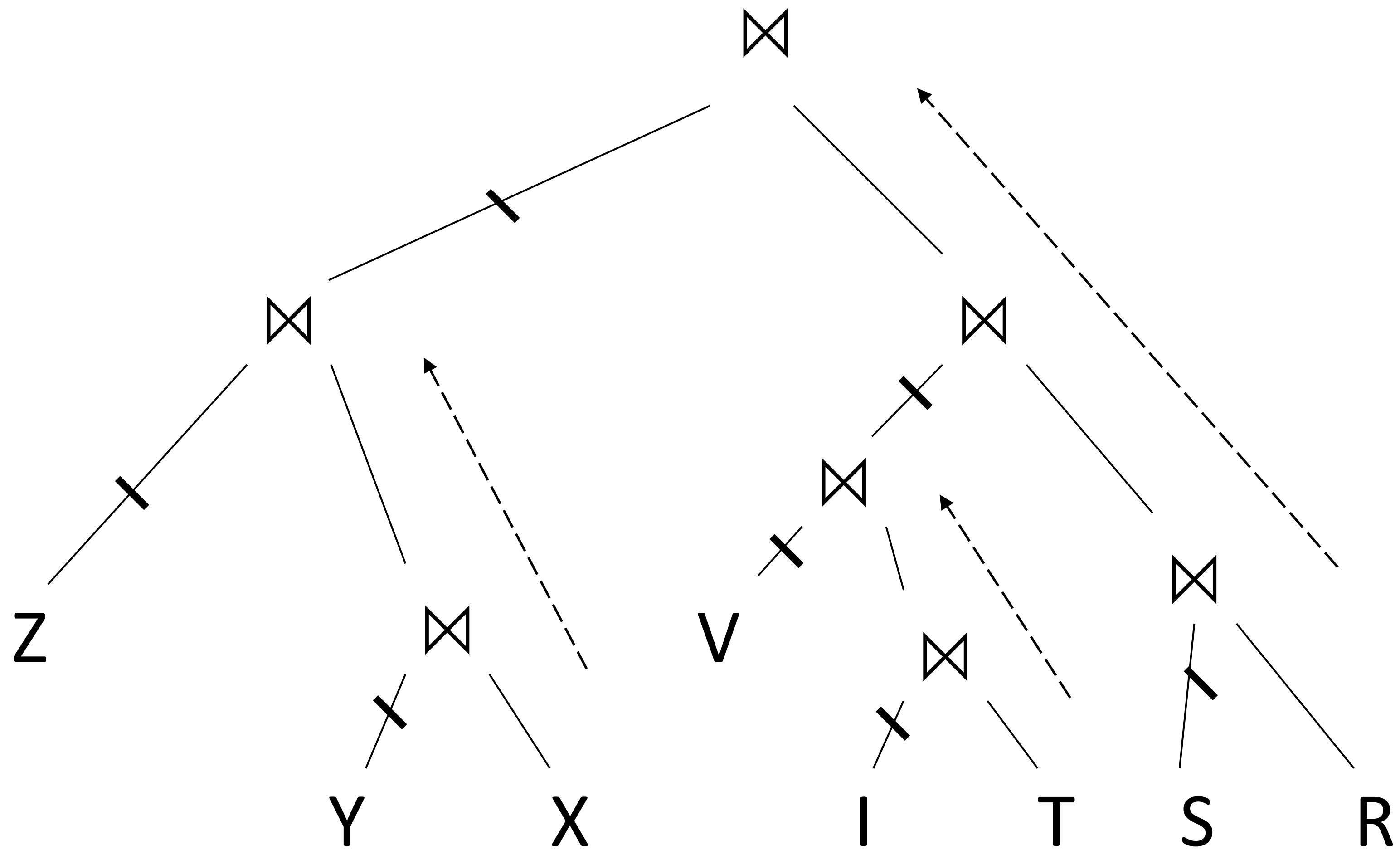
Pipeline Between Operators

Question in class

Given B(R), B(S), B(T), B(U)

- What is the total cost of the plan ?
  - Cost =
- How much main memory do we need ?
  - M =

Pipeline in Bushy Trees



x

Example "Star Schema"

Customers

Cid	...
Alice	...
...	...
...	...

small

Purchases

Cid	Pid	Store	...
Alice	00143	U-district	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...
...	...	...	...

big

Items

Pid	Name	...
00143	Banana	...
...	...	...
...	...	...

small

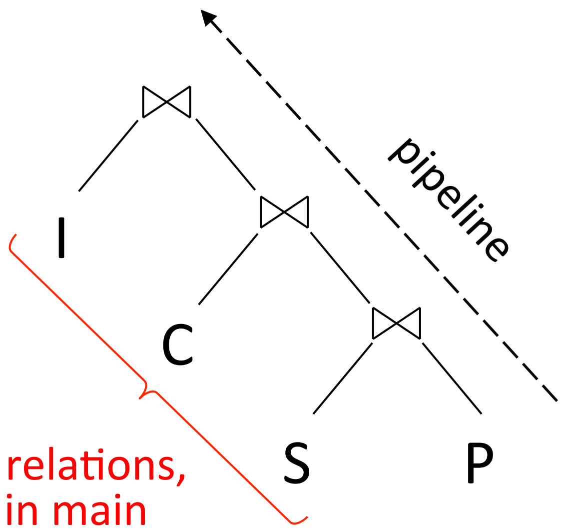
Stores

Sname	...
U-district	...
...	...
...	...

small

x

Possible Naming Confusion



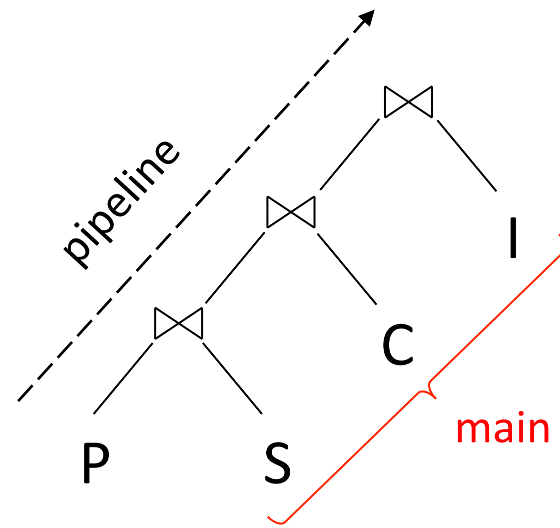
Build relations, fit all in main memory at the same time

Probe relation, too big for main memory

Right-deep plan

Convention in the book and in this class!

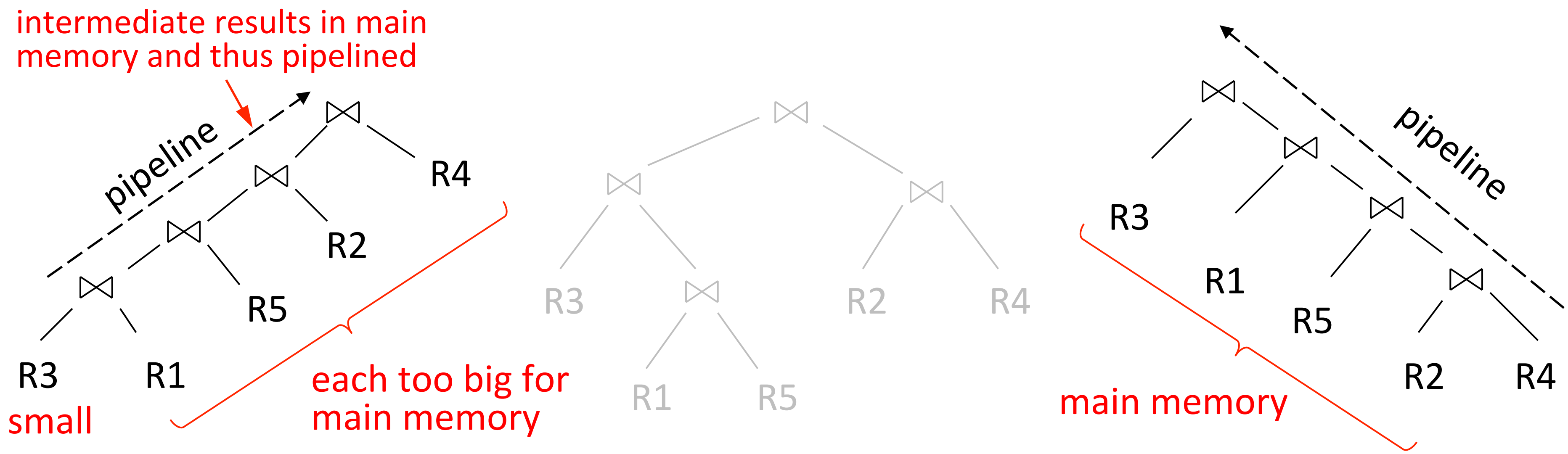
See our textbook example 16.31 p.818



main memory

Note that you may find the same evaluation strategy (all 3 build relations in main memory) at other places depicted as above. Reason is that build and probe are reversed. Hence they call this a left-deep plan. Don't get confused, just so you know.

Types of Join Trees



Left-deep plan

Bushy plan

Right-deep plan

E.g. good for "non-expansive" joins

E.g. good for "expansive" joins

Both are called "linear plans"

Convention in the book and in this class!

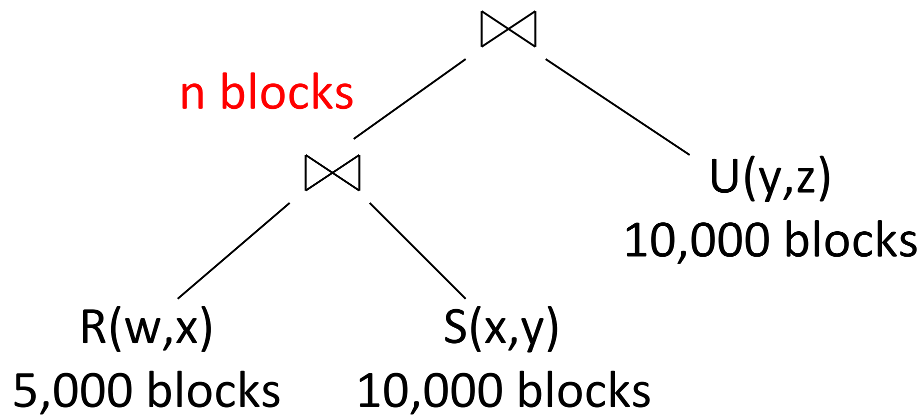
Def. "expansive join": $|R \bowtie S| > \max(|A|,|B|)$
Source: [Stocker et al. ICDE 2001]

x

Example

Logical plan

M = 101



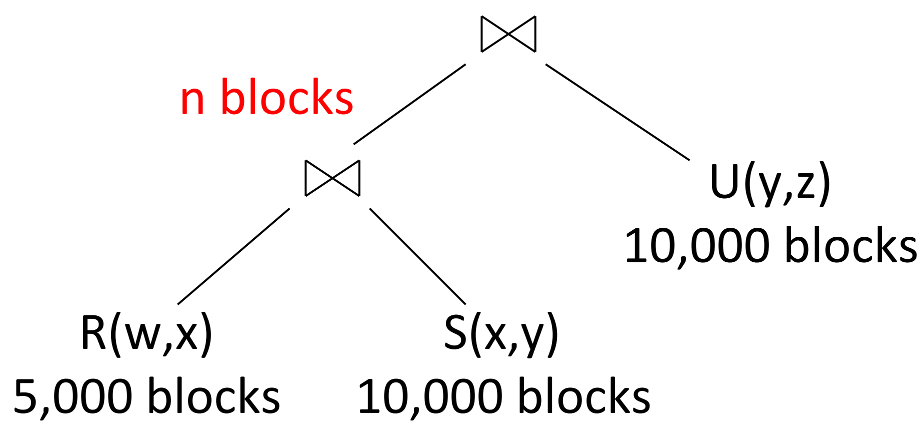
Naïve evaluation:

- 2 partitioned hash-joins
- Cost: [3B(R) + 3B(S) + n] + [3n + 3B(U)]
  = 3B(R) + 3B(R) + 3B(U) + 4n
  = 75,000 + 4n

x

Example

Logical plan



M = 101

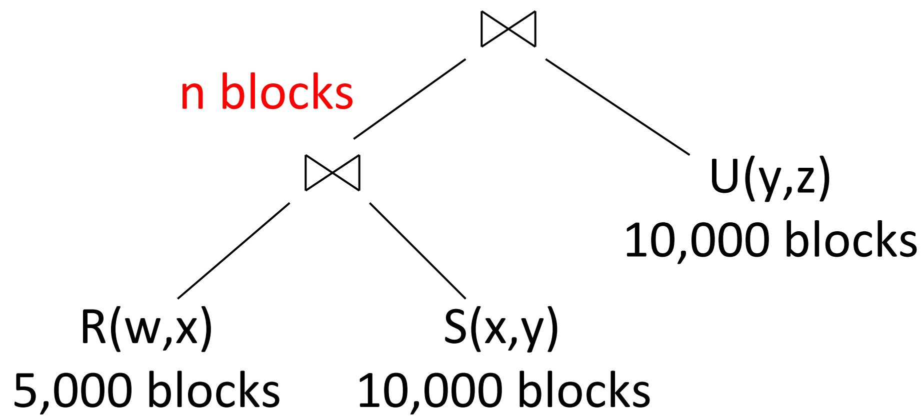
Smarter:

- Step 1: hash R on x into 100 buckets, each of 50 blocks; to disk
- Step 2: hash S on x into 100 buckets, each of 100 blocks; to disk
- Step 3: read each R_i in memory (50 buffer) join with S_i (1 buffer); hash result on y into 50 buckets (50 buffers) -- here we pipeline
- Cost so far: 3B(R) + 3B(S)

Example

Logical plan

M = 101



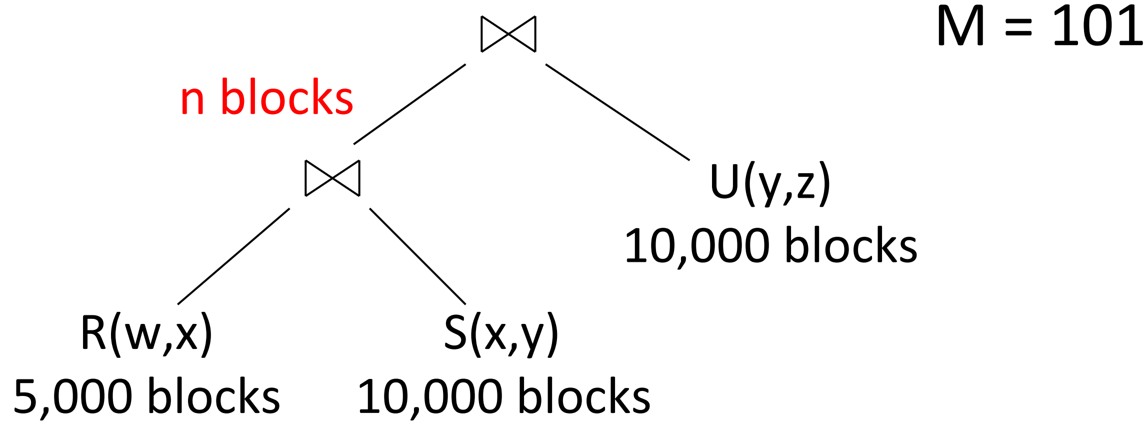
Continuing:

- How large are the 50 buckets on y ? Answer: n/50.
- If n <= 50 then keep all 50 buckets in Step 3 in memory, then:
- Step 4: read U from disk, hash on y and join with memory
- Total cost: 3B(R) + 3B(S) + B(U) = 55,000

x

Example

Logical plan



Continuing:

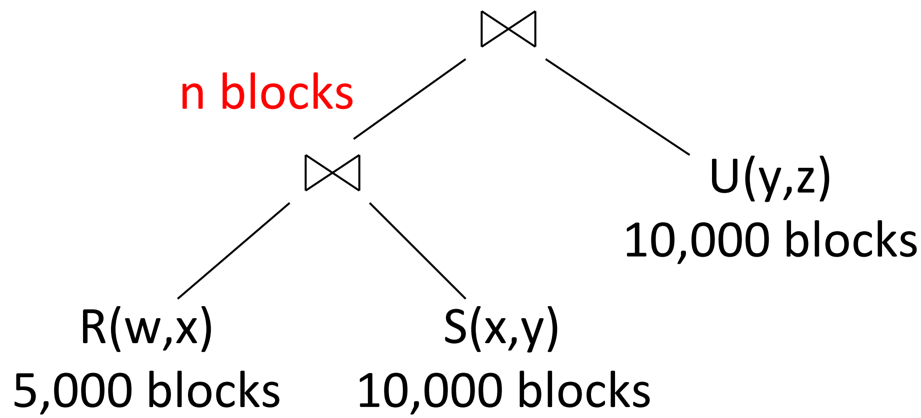
- If 50 < n <= 5000 then send the 50 buckets in Step 3 to disk
  - Each bucket has size n/50 <= 100
- Step 4: partition U into 50 buckets
- Step 5: read each partition and join in memory
- Total cost: 3B(R) + 3B(S) + 2n + 3B(U) = 75,000 + 2n

x

Example

Logical plan

M = 101

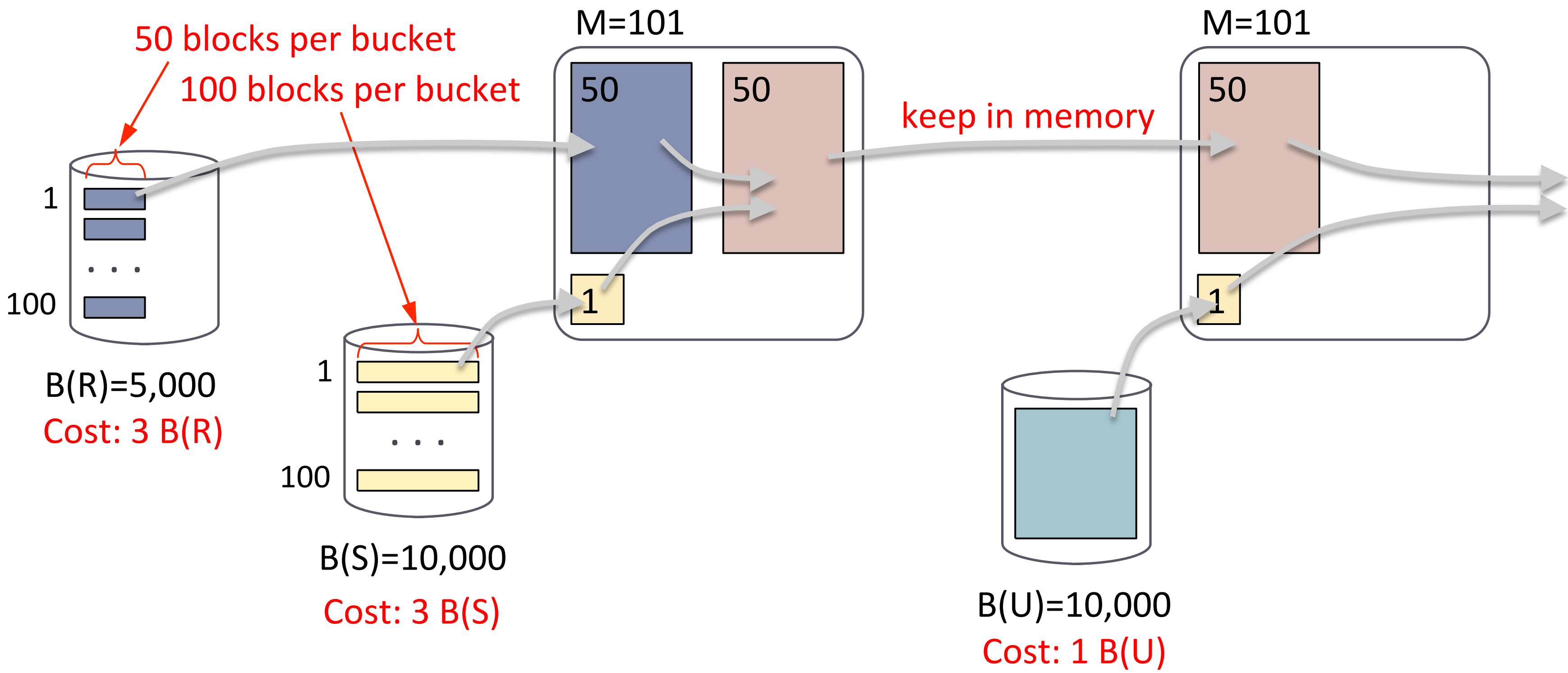


Continuing:

- If n > 5000 then materialize instead of pipeline
- 2 partitioned hash-joins
- Cost 3B(R) + 3B(S) + 4k + 3B(U) = 75000 + 4n

x

Example in pictures 1

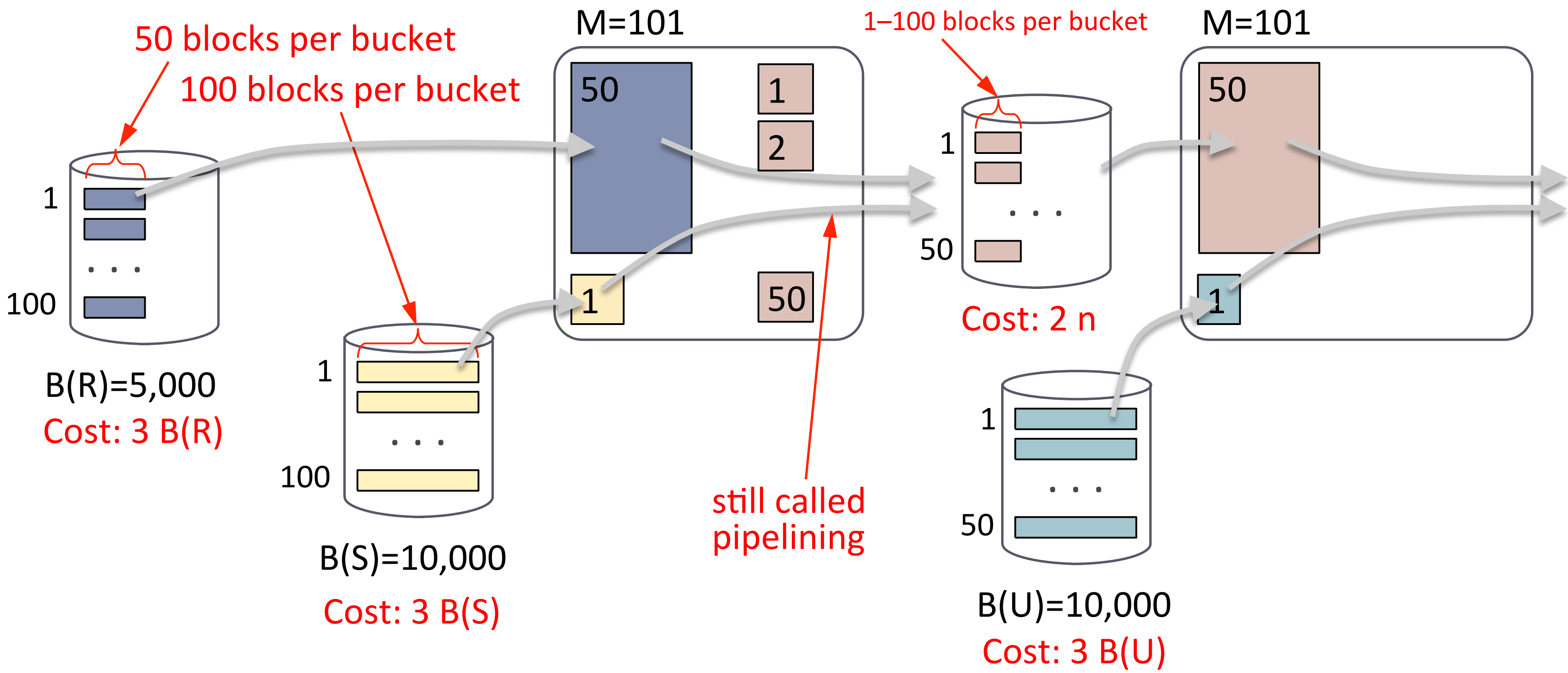


If $n \leq 50$: 3B(R) + 3B(S) + B(U)

Source: Variation on example 16.36 from book; all cost units are in "blocks" = I/O

x

Example in pictures 2

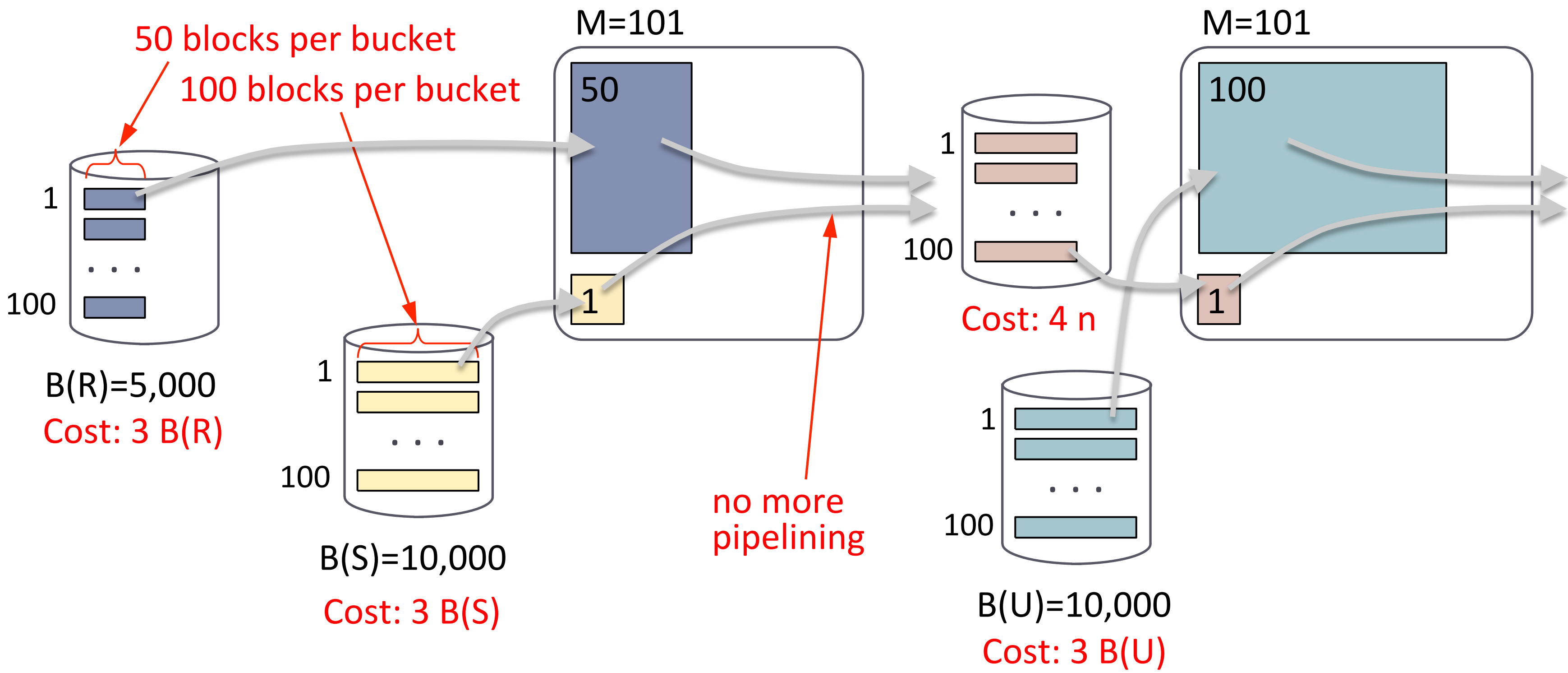


If $50 < n \leq 5000$: 3B(R) + 3B(S) + 2n + 3B(U)

Source: Variation on example 16.36 from book; all cost units are in "blocks" = I/O

x

Example in pictures 3



If 5000 < n: 3B(R) + 3B(S) + 4n + 3B(U)

Source: Variation on example 16.36 from book; all cost units are in "blocks" = I/O

x

Outline

- Search space
- Algorithm for enumerating query plans
- **Estimating the cost of a query plan**

Computing the Cost of a Plan

- Collect statistical summaries of stored data
- Estimate size in a bottom-up fashion
- Estimate cost by using the estimated size

Statistics on Base Data

- Collected information for each relation
  - Number of tuples (cardinality)
  - Indexes, number of keys in the index
  - Number of physical pages, clustering info
  - Statistical information on attributes
    - Min value, max value, number distinct values
    - Histograms
  - Correlations between columns (hard)
- Collection approach
  - periodic
  - using sampling

Size Estimation Problem

```
S = SELECT  list
    FROM    R1, …, Rn
    WHERE   cond1 AND cond2 AND . . . AND condk
```

Given T(R1), T(R2), …, T(Rn)
Estimate T(S)

How can we do this ? Note: doesn't have to be exact.

Remark: $T(S) \leq T(R1) \times T(R2) \times \ldots \times T(Rn)$

Selectivity Factor

- Each condition "cond" reduces the size by some factor called selectivity (factor)
  - selection

$$sel_{\sigma} = \frac{|\sigma_C(R)|}{|R|}$$

  - join

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

- Assuming independence, multiply the selectivity factors

x

Example

```
SELECT   *
FROM     R, S, T
WHERE    R.B=S.B and S.C=T.C and R.A<40
```

R(A,B)
S(B,C)
T(C,D)

T(R) = 300, T(S) = 2000, T(T) = 100

Selectivity of R.B = S.B is 1/3
Selectivity of S.C = T.C is 1/10
Selectivity of R.A < 40 is ½

What is the estimated size of the query output ?

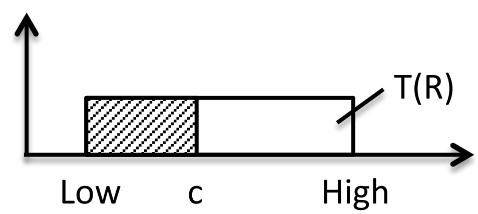
x

Rule of Thumb

- If selectivities are unknown, then:
  selectivity factor = 1/10
  [System R, 1979]

Selectivities from Statistics

- Condition is A = c   /* value selection on R */
  - Selectivity = 1/V(R,A)
- Condition is A < c   /* range selection on R */
  - Selectivity = (c - Low(R, A)) / (High(R,A) - Low(R,A)) T(R)



- Condition is A = B   /* R $\bowtie_{A=B}$ S */
  - Selectivity = 1 / max(V(R,A),V(S,A))
  - (will explain next)

Selectivity of R $\bowtie_{A=B}$ S

R

	A
	1
	1
	2
	2
	2
	2
	2
	3
	3
	3

T(R) = 10
V(R,A) = 3

S

B	
1	
1	
2	
3	
4	
4	

T(S) = 6
V(S,B) = 4

|R x S| = 60
|R $\bowtie_{A=B}$ S| = 12

$sel_{\bowtie}$ = 12/60 = 1/5

Assumption:

Containment of values:

- if V(R,A) ≤ V(S,B), then the set of A values of R is subset of B values of S
- Here: {1,2,3} ⊆ {1,2,3,4}

When does this hold for sure?

Conclusion 1:

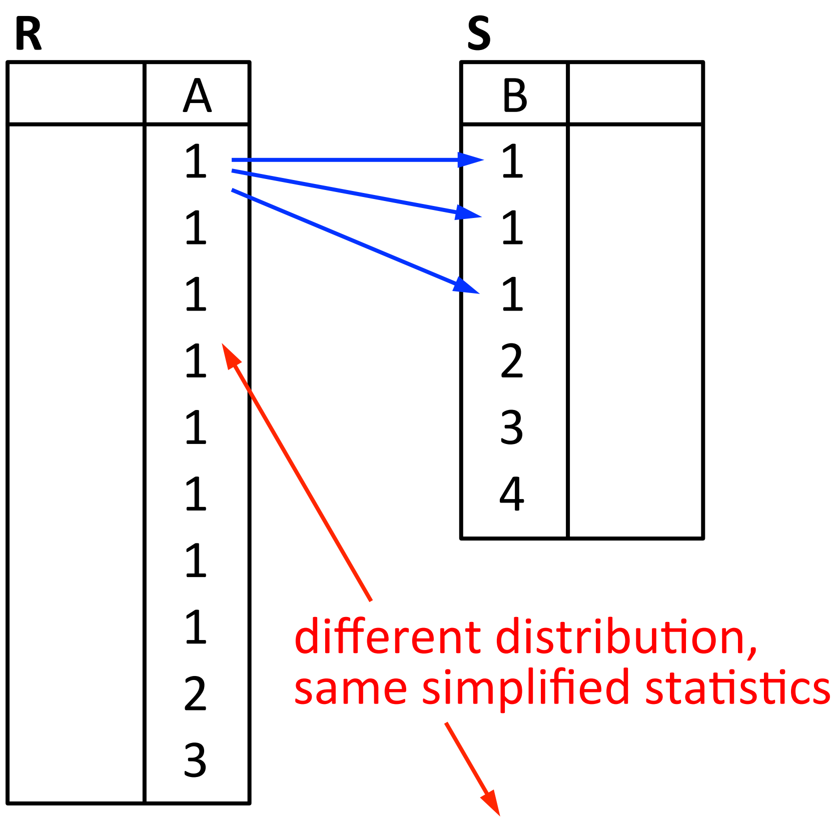
- A tuple from R joins with expected T(S)/V(S,B) tuples from S

Conclusion 2:

- Expected join size is T(S) T(R) / V(S,B) = 12.5

Why different?

Selectivity of R $\bowtie_{A=B}$ S



T(R) = 10
V(R,A) = 3

T(S) = 6
V(S,B) = 4

|R x S| = 60
|R $\bowtie_{A=B}$ S| = 26

$\text{sel}_{\bowtie}$ = 12/60 = 0.43

Assumption:

Containment of values:

- if V(R,A) ≤ V(S,B), then the set of A values of R is subset of B values of S
- Here: {1,2,3} ⊆ {1,2,3,4}
- Holds for sure if A in R is a foreign key on B in S (not the case here)

Conclusion 1:

- A tuple from R joins with expected T(S)/V(S,B) tuples from S

Conclusion 2:

- Expected join size is T(S) T(R) / V(S,B) = 12.5

Expected sel_{RS} = 1/V(S,B) = 0.25

x

Selectivity of R $\bowtie_{A=B}$ S

R

	A
	1
	2
	2
	2
	2
	2
	2
	2
	2
	3

T(R) = 10
V(R,A) = 3

S

B	
1	
2	
3	
4	
4	
4	

T(S) = 6
V(S,B) = 4

different distribution,
same simplified statistics

|R x S| = 60
|R $\bowtie_{A=B}$ S| = 10

$sel_{\bowtie}$ = 12/60 = 0.17

Assumption:

Containment of values:

- if V(R,A) ≤ V(S,B), then the set of A values of R is subset of B values of S
- Here: {1,2,3} ⊆ {1,2,3,4}
- Holds for sure if A in R is a foreign key on B in S (not the case here)

Conclusion 1:

- A tuple from R joins with expected T(S)/V(S,B) tuples from S

Conclusion 2:

- Expected join size is T(S) T(R) / V(S,B) = 12.5

Expected sel_{RS} = 1/V(S,B) = 0.25

x

Assumptions

- 1: Containment of values: if $V(R,A) \leq V(S,B)$, then the set of A values of R is included in the set of B values of S
  - Note: this indeed holds when A is a foreign key in R, and B is a key in S

- 2: Preservation of values: for any other attribute C, $V(R \bowtie_{A=B} S, C) = V(R,C)$ if C is attribute of R

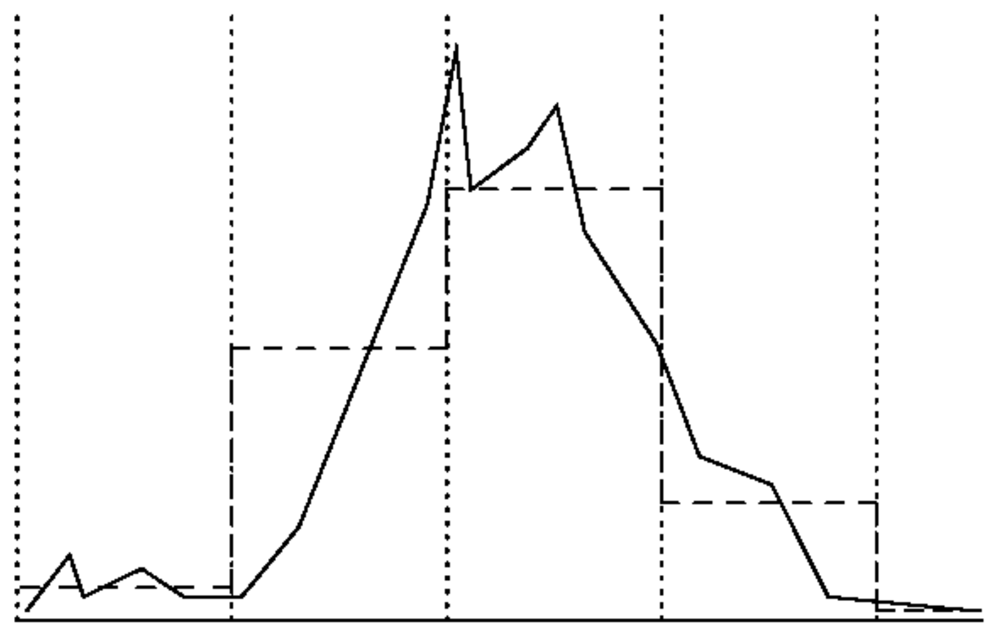
Size Estimation for Join

Example:

- $T(R) = 10{,}000$, $T(S) = 20{,}000$
- $V(R,A) = 100$, $V(S,B) = 200$
- How large is $R \bowtie_{A=B} S$?

Histograms

- Statistics on data maintained by the RDBMS
- Makes size estimation much more accurate
  - hence, cost estimations are more accurate



Histograms

Employee(ssn, name, age)

T(Employee) = 25,000
V(Employee, age) = 50
min(age) = 18
max(age) = 77

$\sigma_{age=18}$(Employee) = ?

$\sigma_{age>28 \text{ and } age<35}$(Employee) = ?

Histograms

Employee(<u>ssn</u>, name, age)

T(Employee) = 25,000
V(Employee, age) = 50
min(age) = 18
max(age) = 77

$\sigma_{age=18}$(Employee) = ?

Estimate = 25,000 / 50 = 500

$\sigma_{age>28 \text{ and } age<35}$(Employee) = ?

Estimate = 25,000 * 6 / 60 = 2,500

Histograms

Employee(<u>ssn</u>, name, age)

T(Employee) = 25,000
V(Employee, age) = 50
min(age) = 18
max(age) = 77

$\sigma_{age=18}$(Employee) = ?

$\sigma_{age>28 \text{ and } age<35}$(Employee) = ?

Age:	10..19	20..29	30-39	40-49	50-59	> 60
Tuples	200	800	5,000	12,000	6,500	500

x

Histograms

Employee(ssn, name, age)

T(Employee) = 25,000
V(Employee, age) = 50
min(age) = 18
max(age) = 77

$\sigma_{age=18}$(Employee) = ?

$\sigma_{age>28 \text{ and } age<35}$(Employee) = ?

Age:	10..19	20..29	30-39	40-49	50-59	> 60
Tuples	200	800	5,000	12,000	6,500	500

Estimate = 20

Estimate = 1*80 + 5*500 = 2580

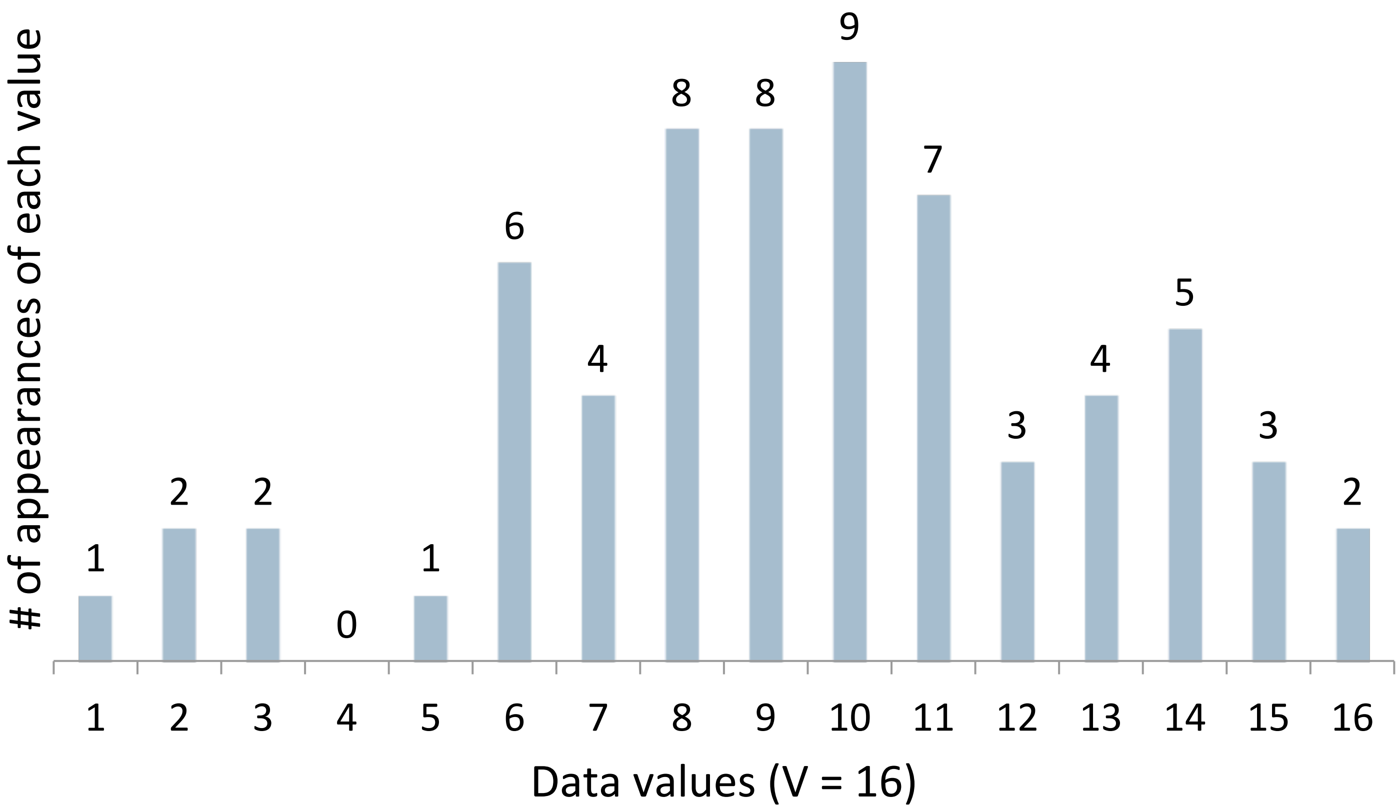
x

Types of Histograms

- How should we determine the bucket boundaries in a histogram ?

- Equi-Width
- Equi-Depth
  - also called equi-height or equi-sum
- Compressed

Histograms

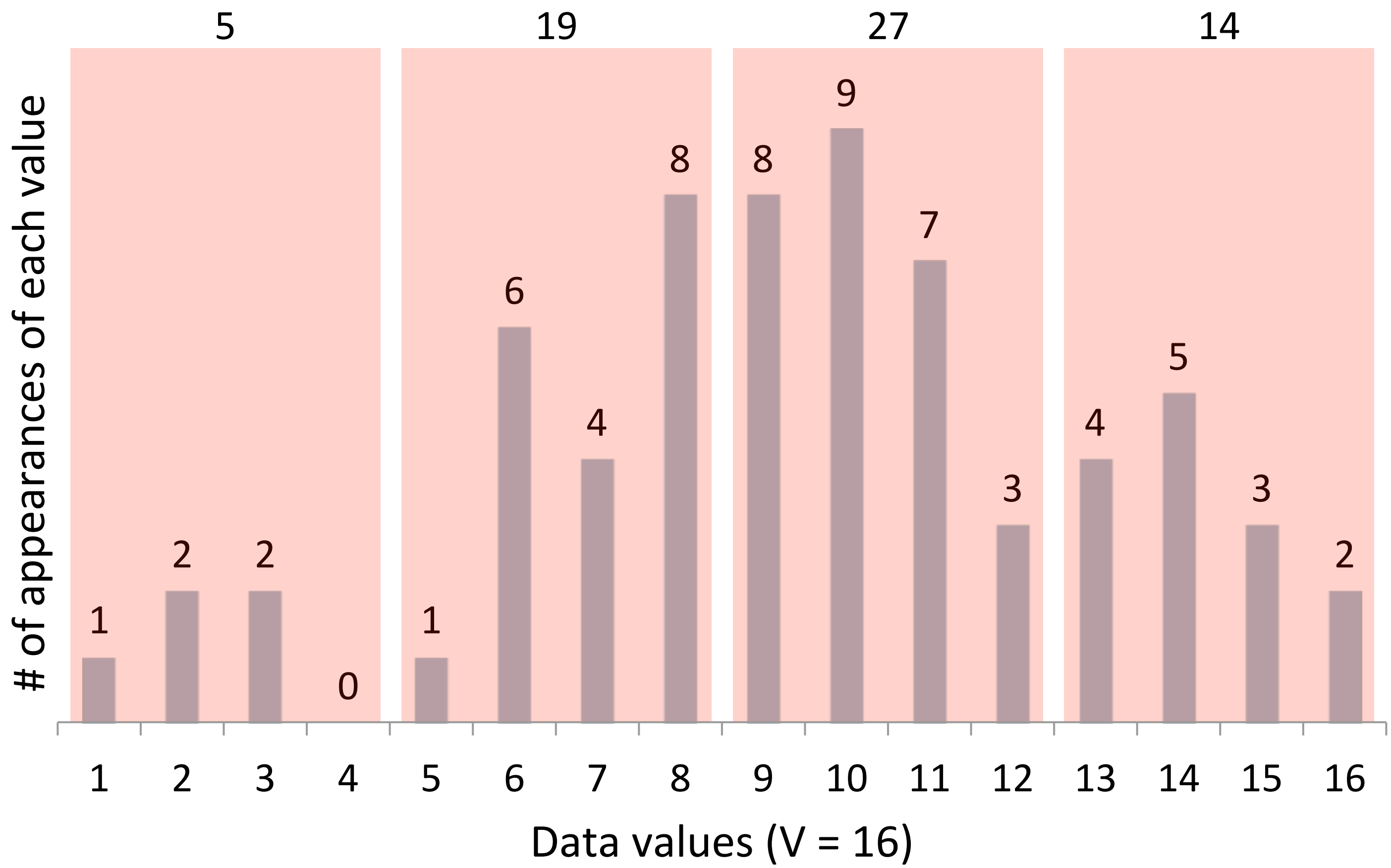


Source: Numerical values slightly varied from:
https://grape.ics.uci.edu/wiki/public/wiki/cs222-2010-fall-lecture17 (March 2011)

x

Histograms

1. Equi-Width Histogram

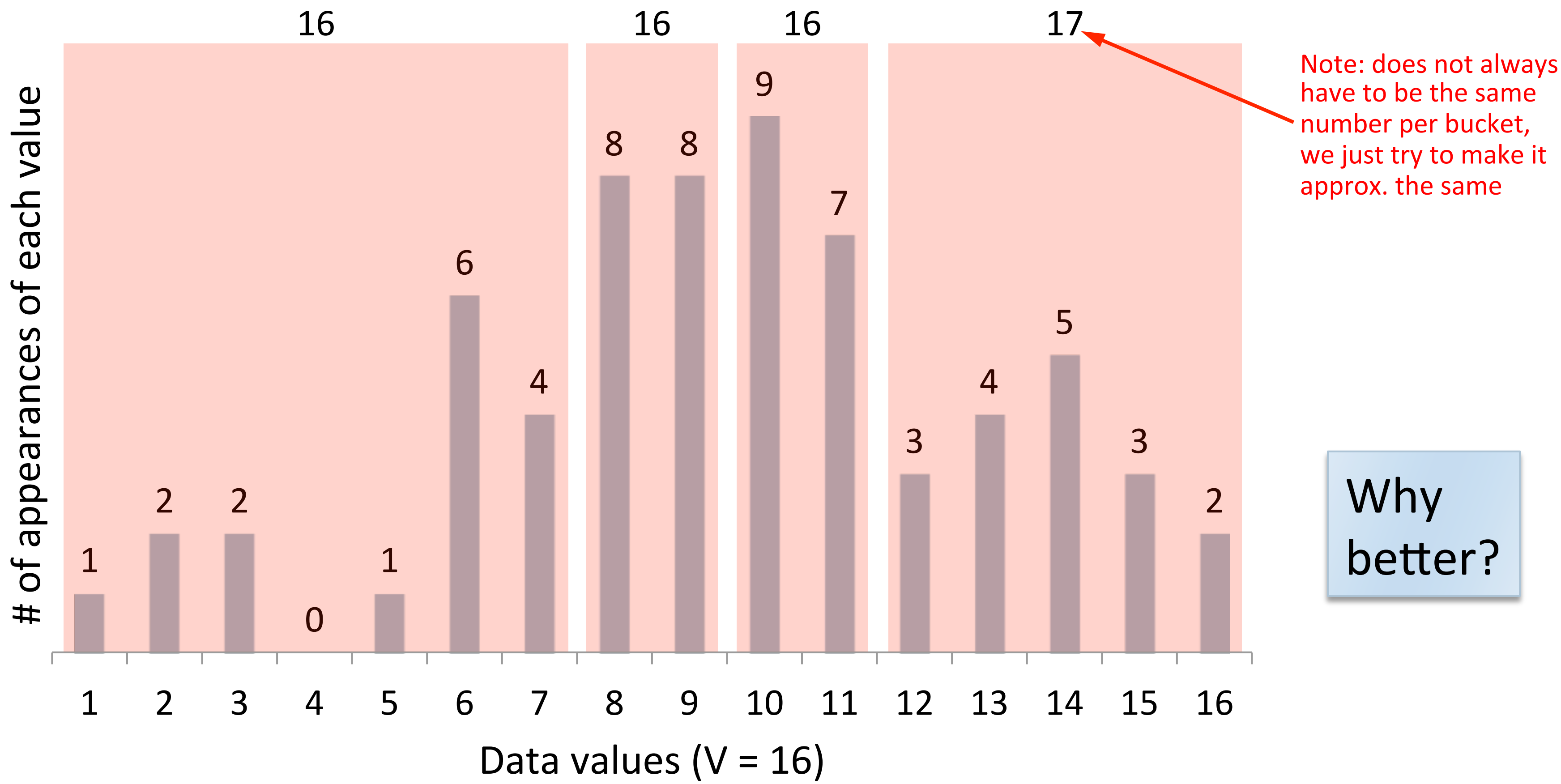


Source: Numerical values slightly varied from:
https://grape.ics.uci.edu/wiki/public/wiki/cs222-2010-fall-lecture17 (March 2011)

x

Histograms

2. Equi-Depth Histogram (also Equi-Height / Equi-sum)



Why better?

3. Compressed: store separately some highly frequent values: e.g. (10,9)

Source: Numerical values slightly varied from:
https://grape.ics.uci.edu/wiki/public/wiki/cs222-2010-fall-lecture17 (March 2011)

x

Histograms

Employee(ssn, name, age)

Equi-width

Age:	10..19	20..29	30-39	40-49	50-59	> 60
Tuples	200	800	5,000	12,000	6,500	500

Equi-depth

Age:	10..34	35..41	42-45	46-48	49-54	> 55
Tuples	4,200	4,100	4,200	4,300	3,900	4,300

x

Difficult Questions on Histograms

- Small number of buckets
  - Hundreds, or thousands, but not more
  - WHY ?
- *Not* updated during database update, but recomputed periodically
  - WHY ?
- Multidimensional histograms rarely used
  - WHY ?

Summary of Query Optimization

- Three parts:
  - search space, algorithms, size/cost estimation
- Ideal goal: find optimal plan. But
  - Impossible to estimate accurately
  - Impossible to search the entire space
- Goal of today's optimizers:
  - Avoid very bad plans

Outline

- Search space
- Algorithm for enumerating query plans
- Estimating the cost of a query plan
- **Some extra slides (optional)**

The following slides are taken from this German Database textbook. They may provide some alternative intuitions for some of the operators. Topics:
Hash Join / value of 2 passes / Merge join / External sort / semi-join / operators

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

x

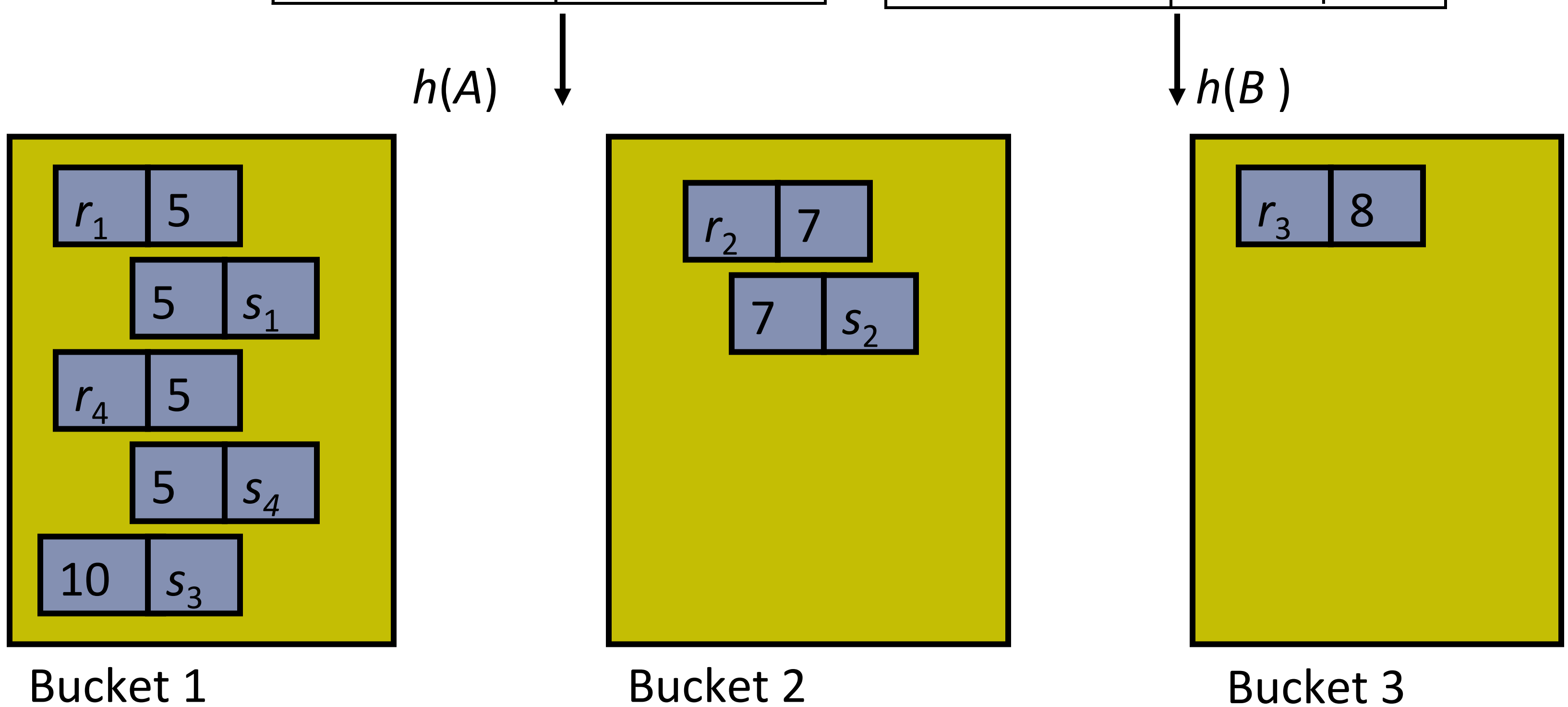
Partition relations into buckets

R

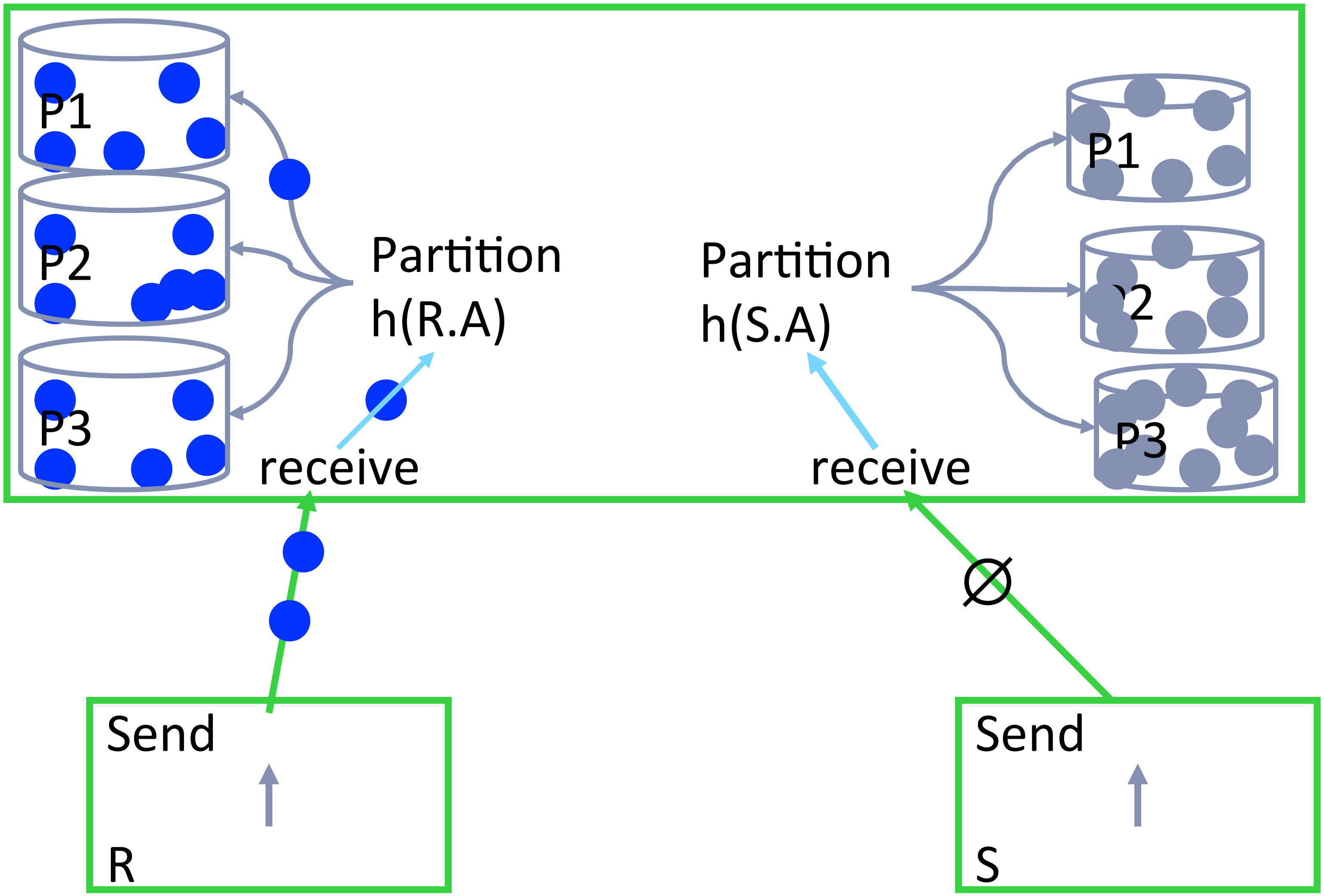
…	A
r_1	5
r_2	7
r_3	8
r_4	5

S

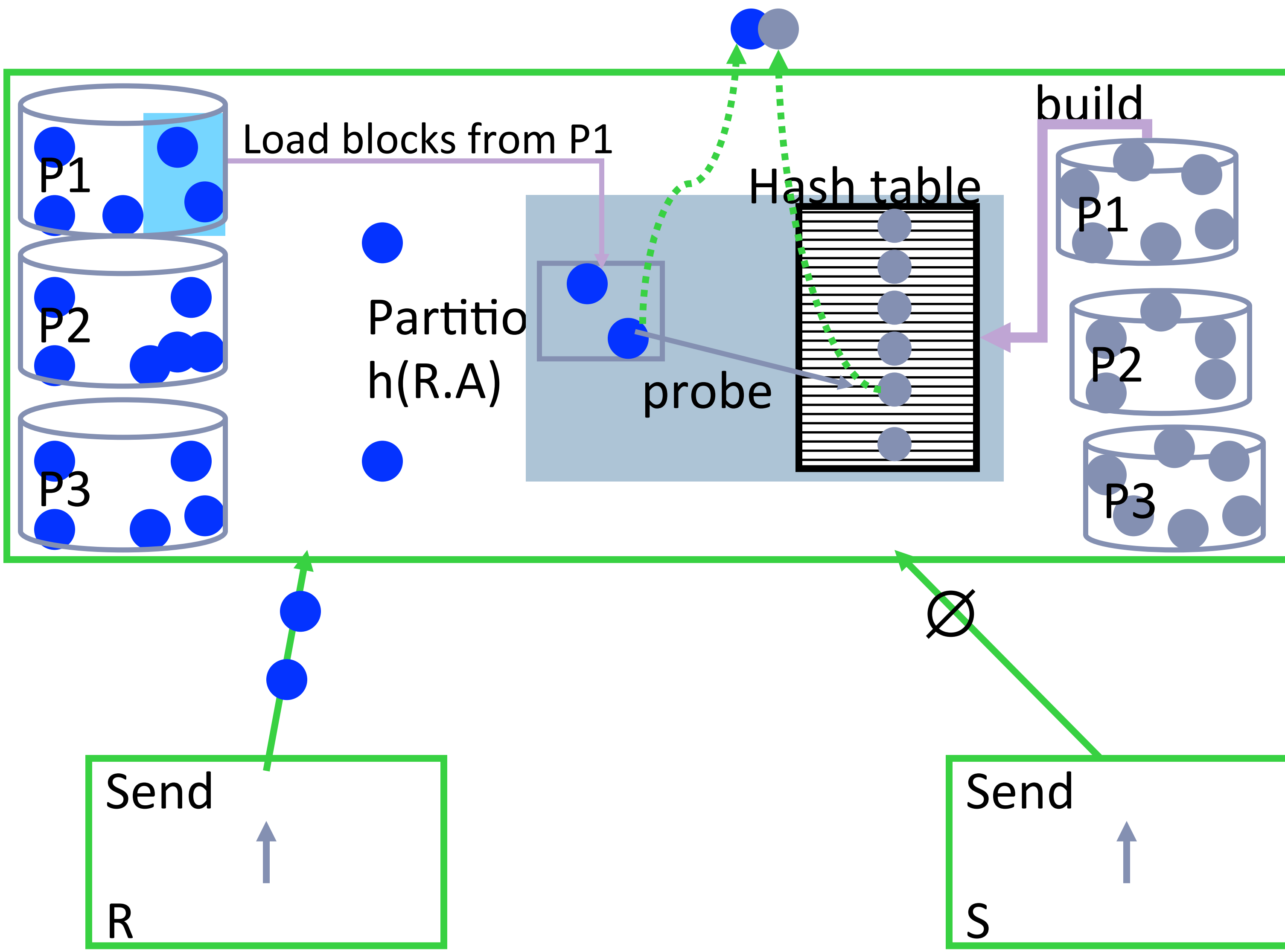
B	…
5	s_1
7	s_2
10	s_3
5	s_4



Partitioned Hash Join: Partitioning



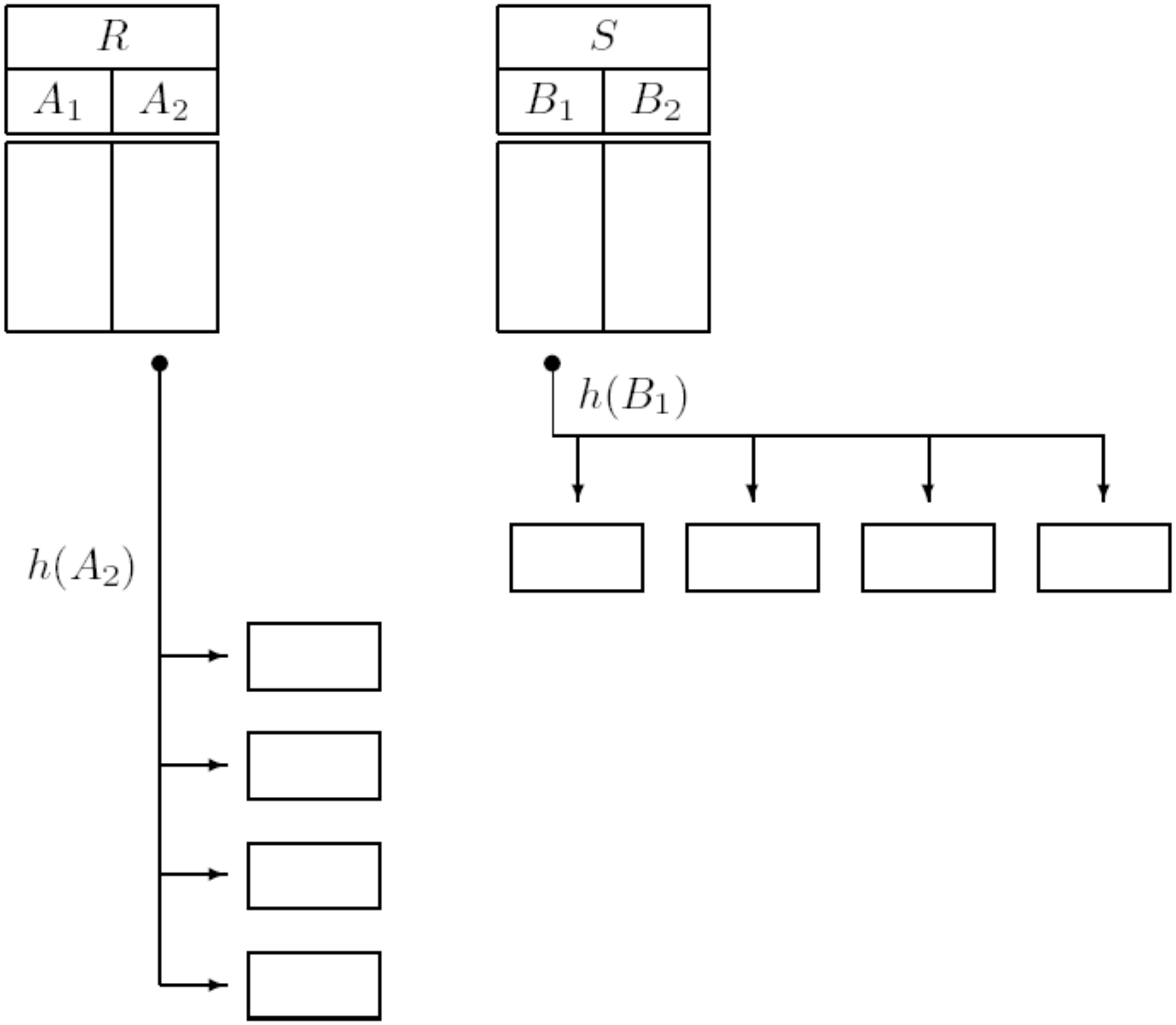
Partitioned Hash Join: Build/Probe



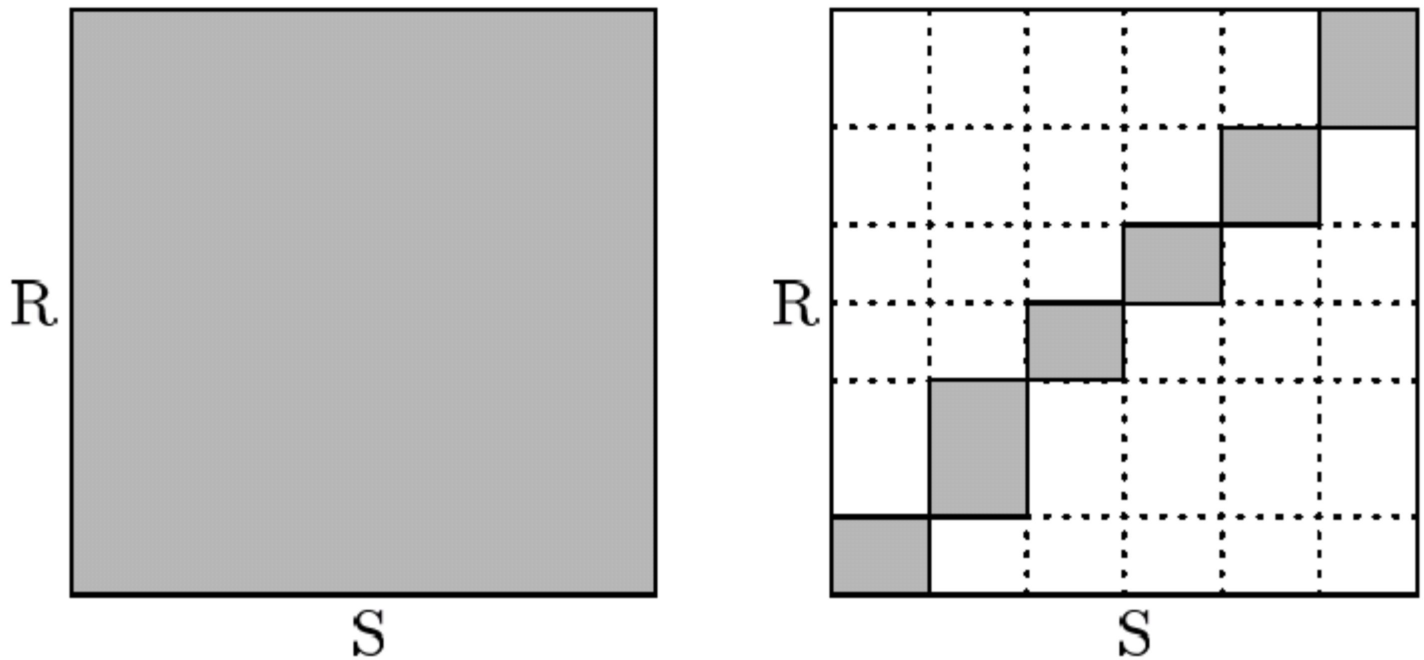
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

x

Comparing Tuples "on the Diagonal"



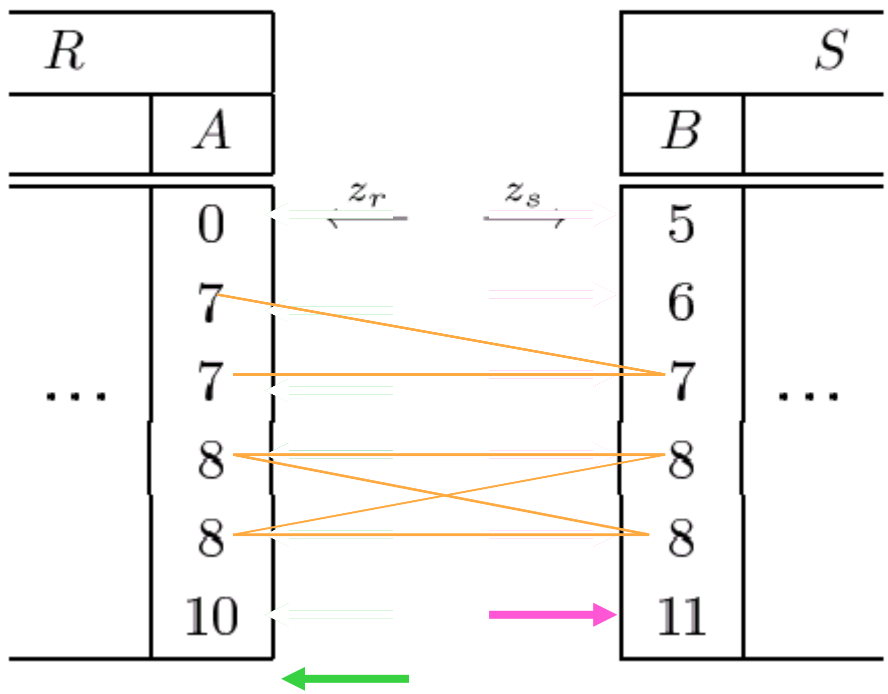
Comparing Tuples "on the Diagonal"



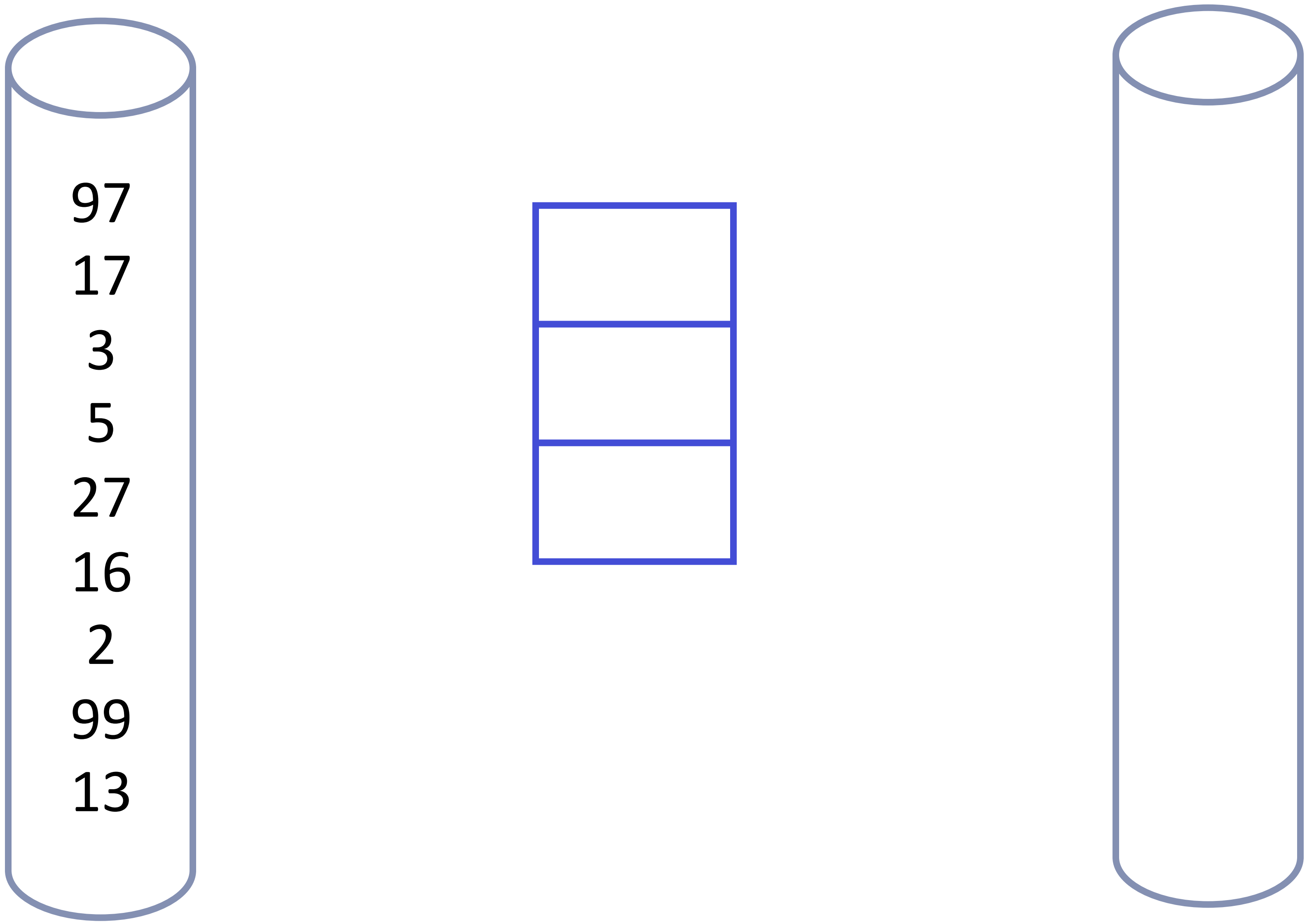
Merge-Join

- Assume R and S are already sorted:
  - Then each relation needs to be read only once

Beispiel:

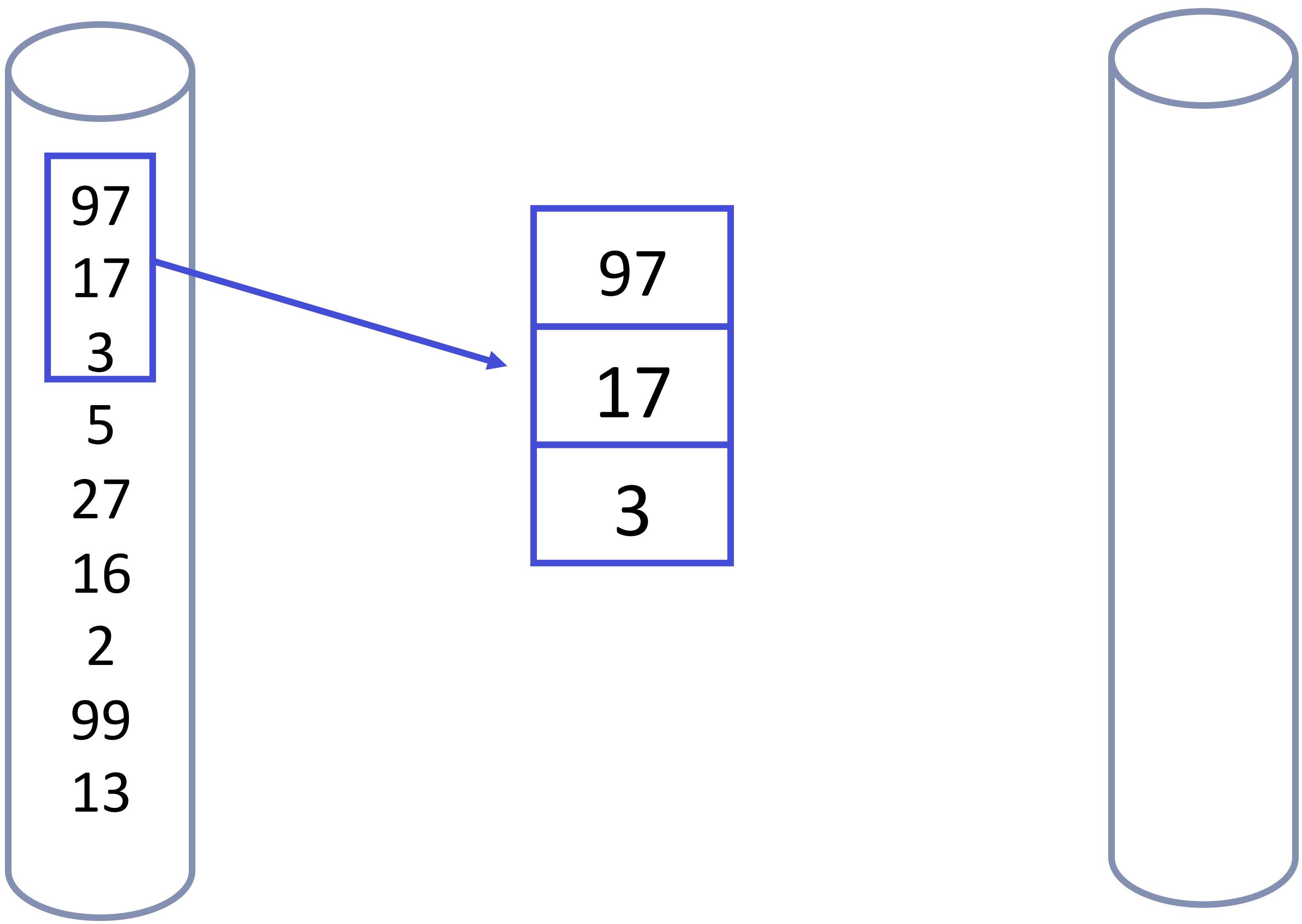


External Sorting



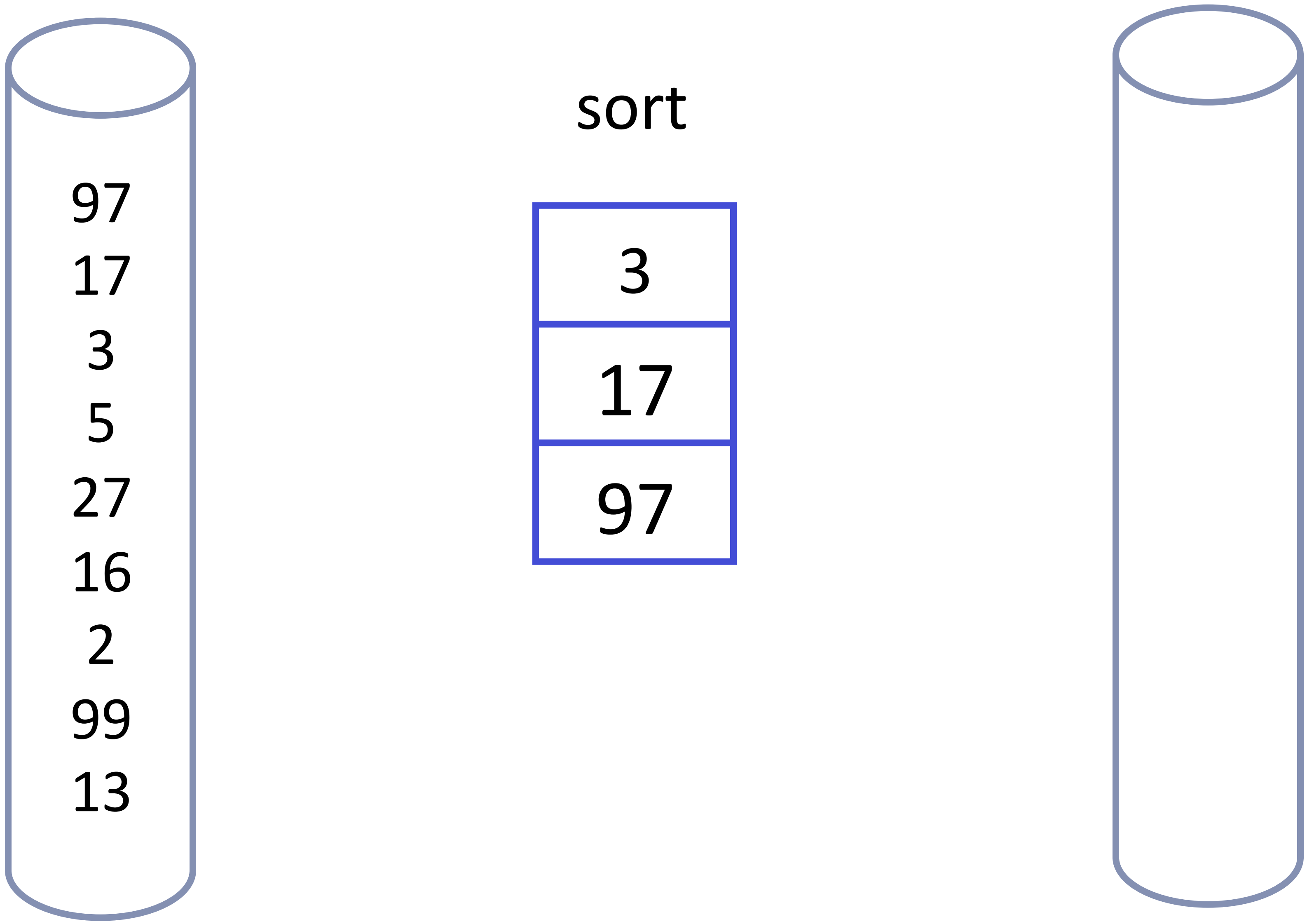
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting



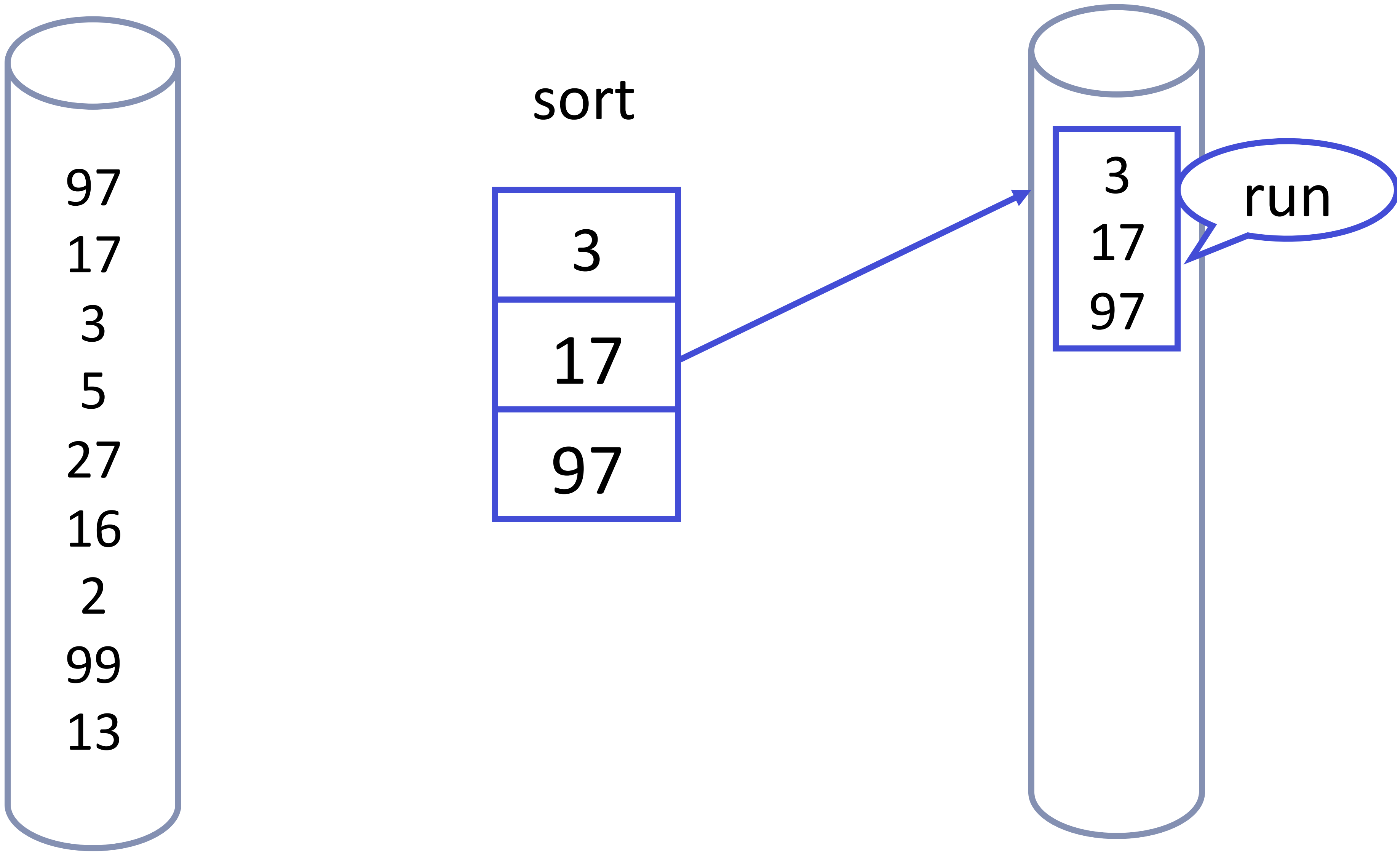
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting

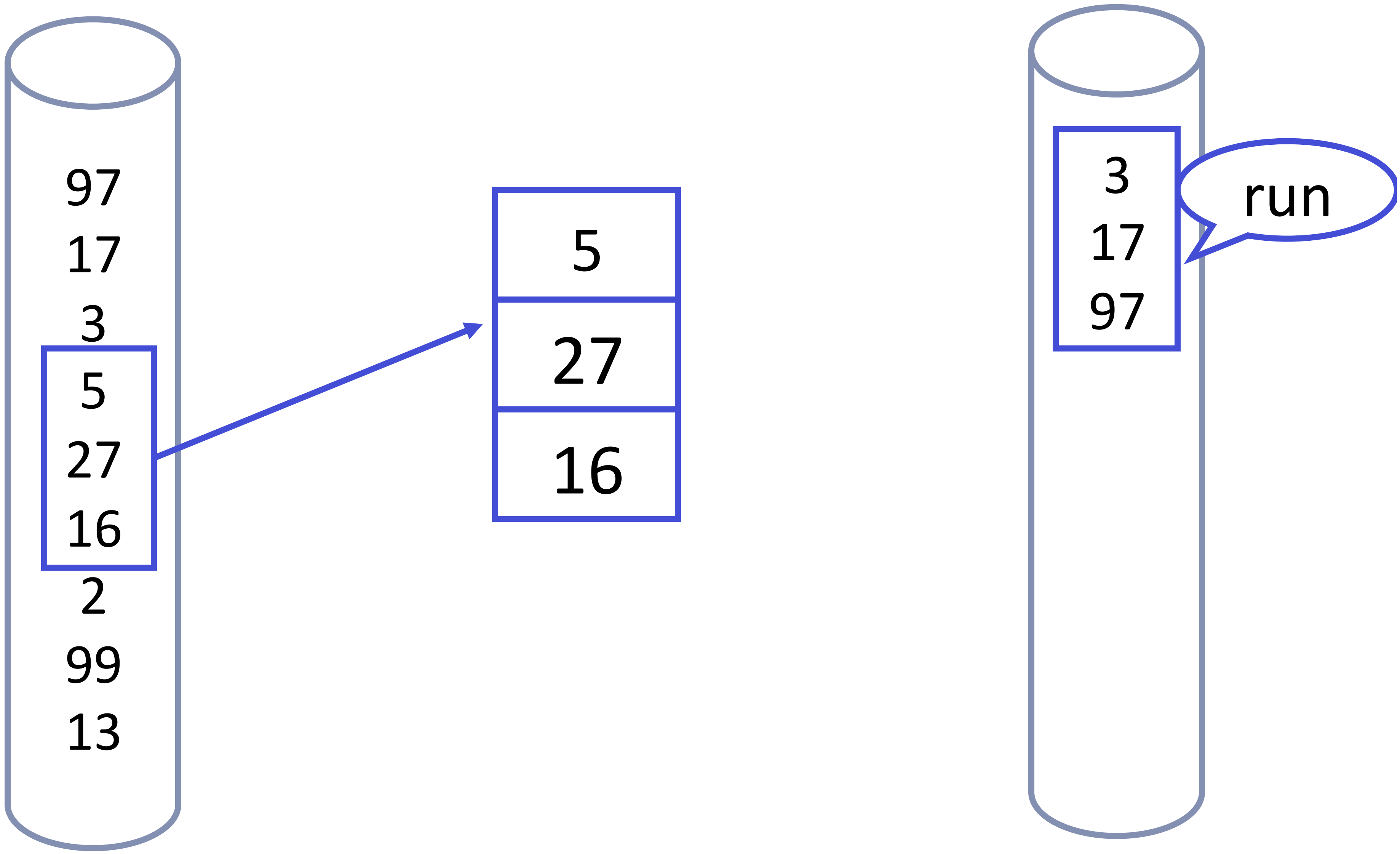


Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

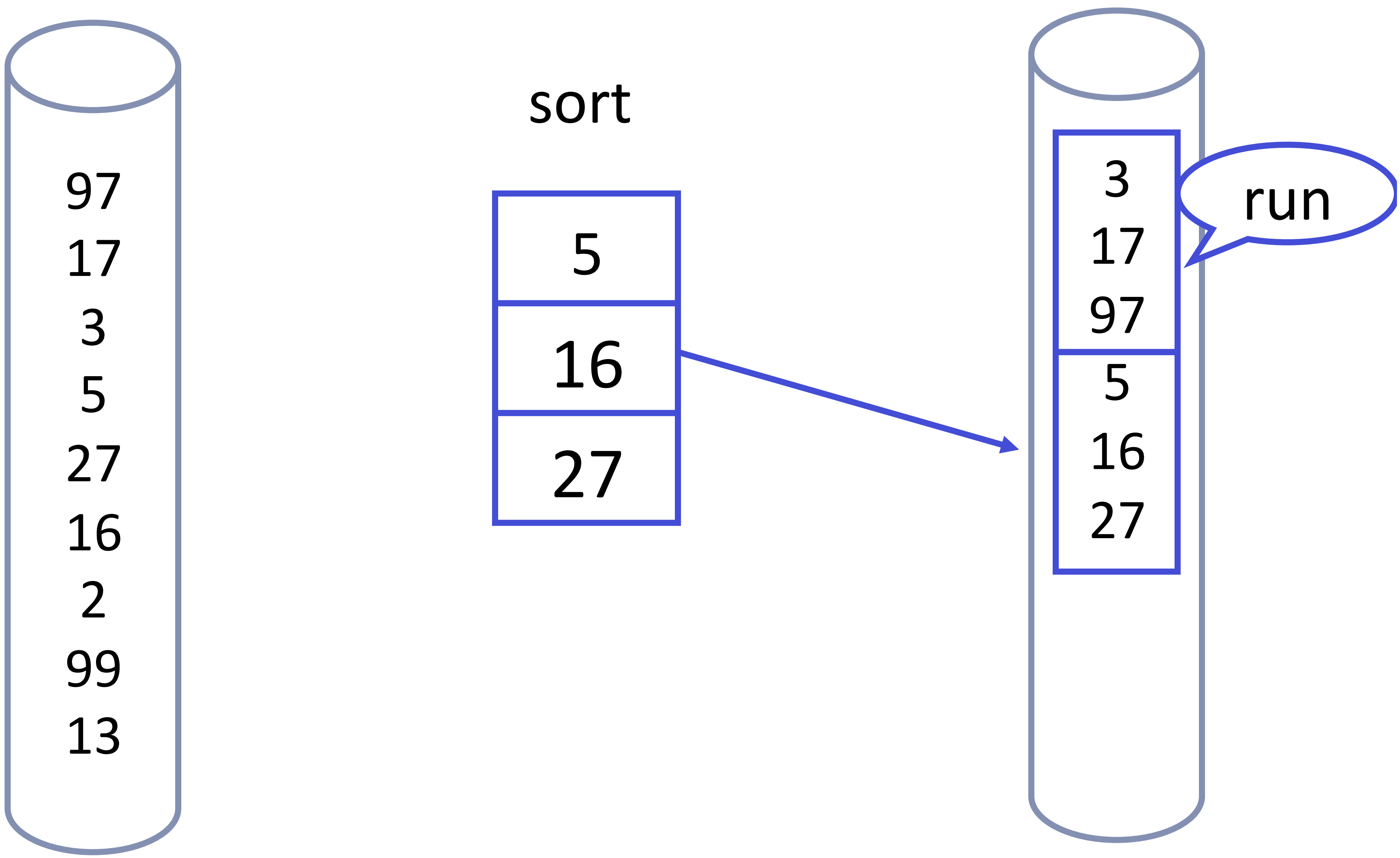
External Sorting



External Sorting

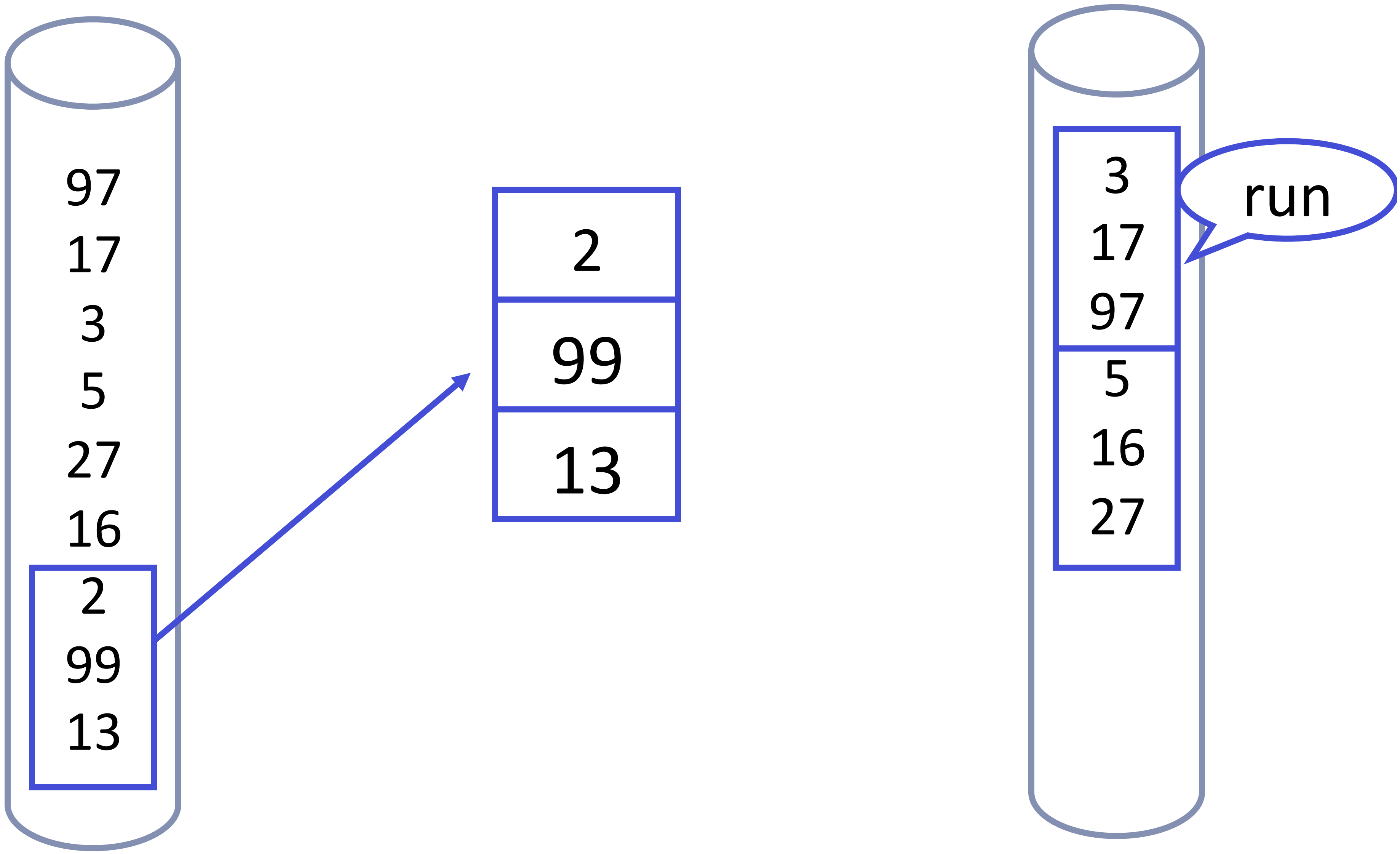


External Sorting



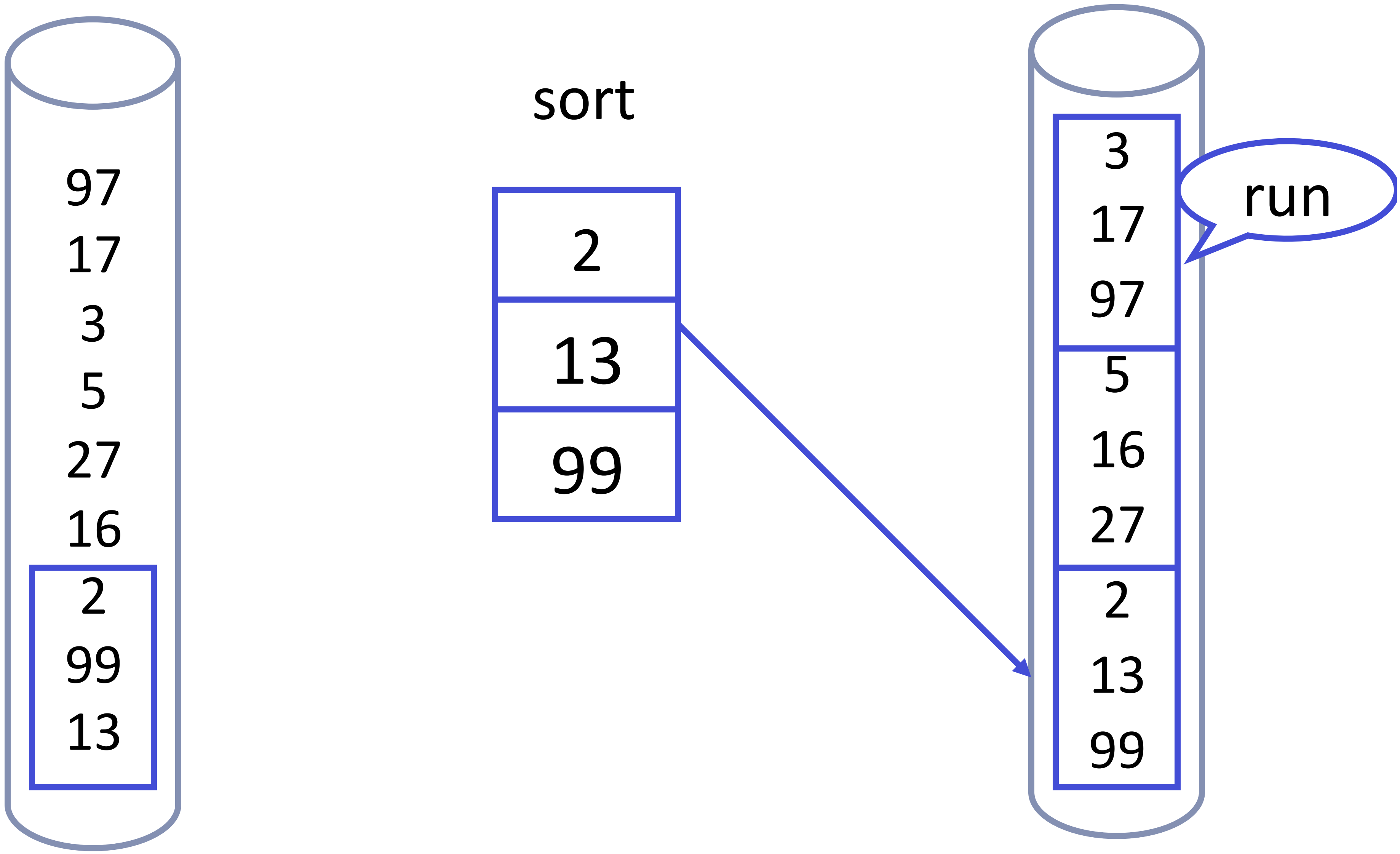
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting



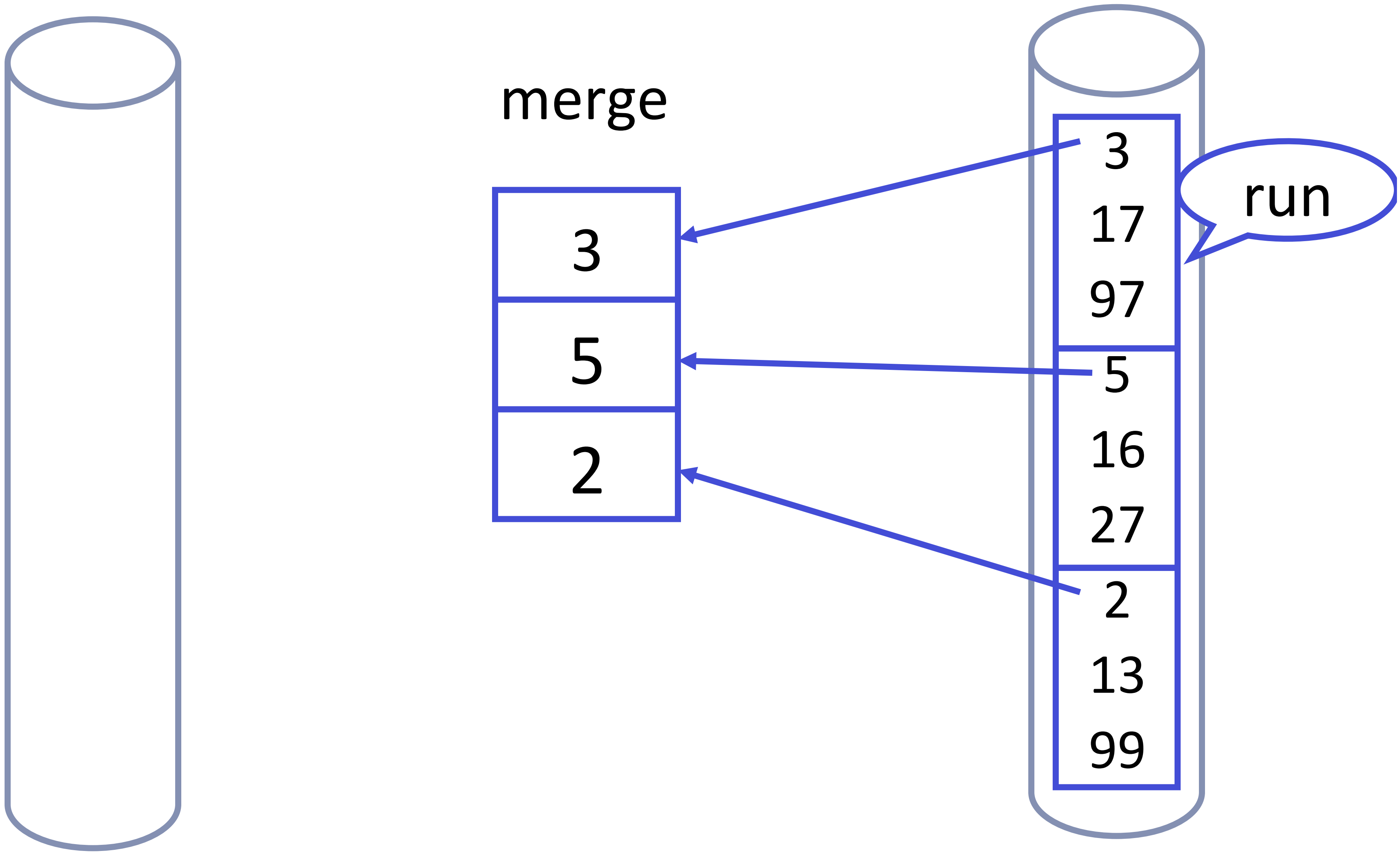
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting



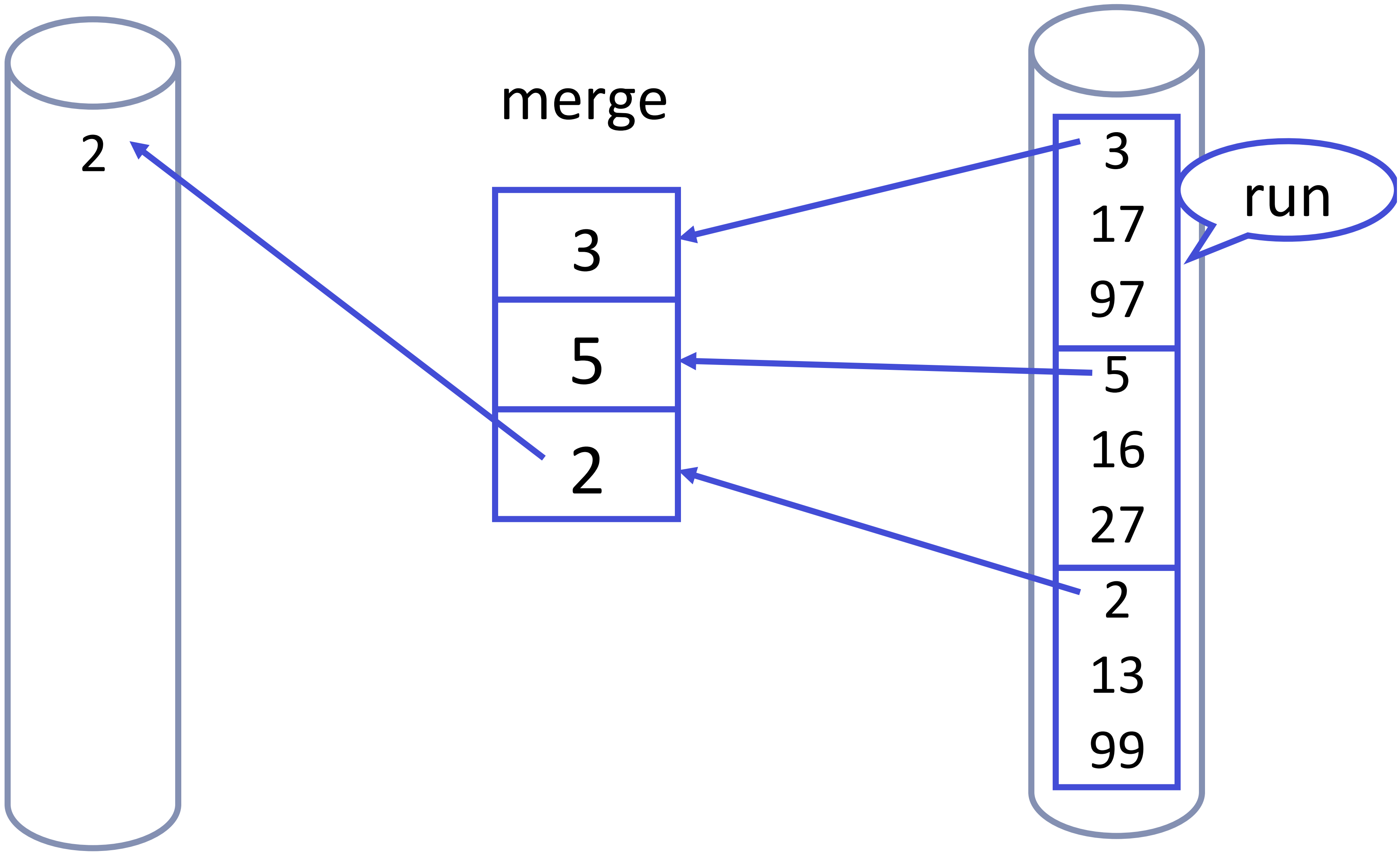
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting



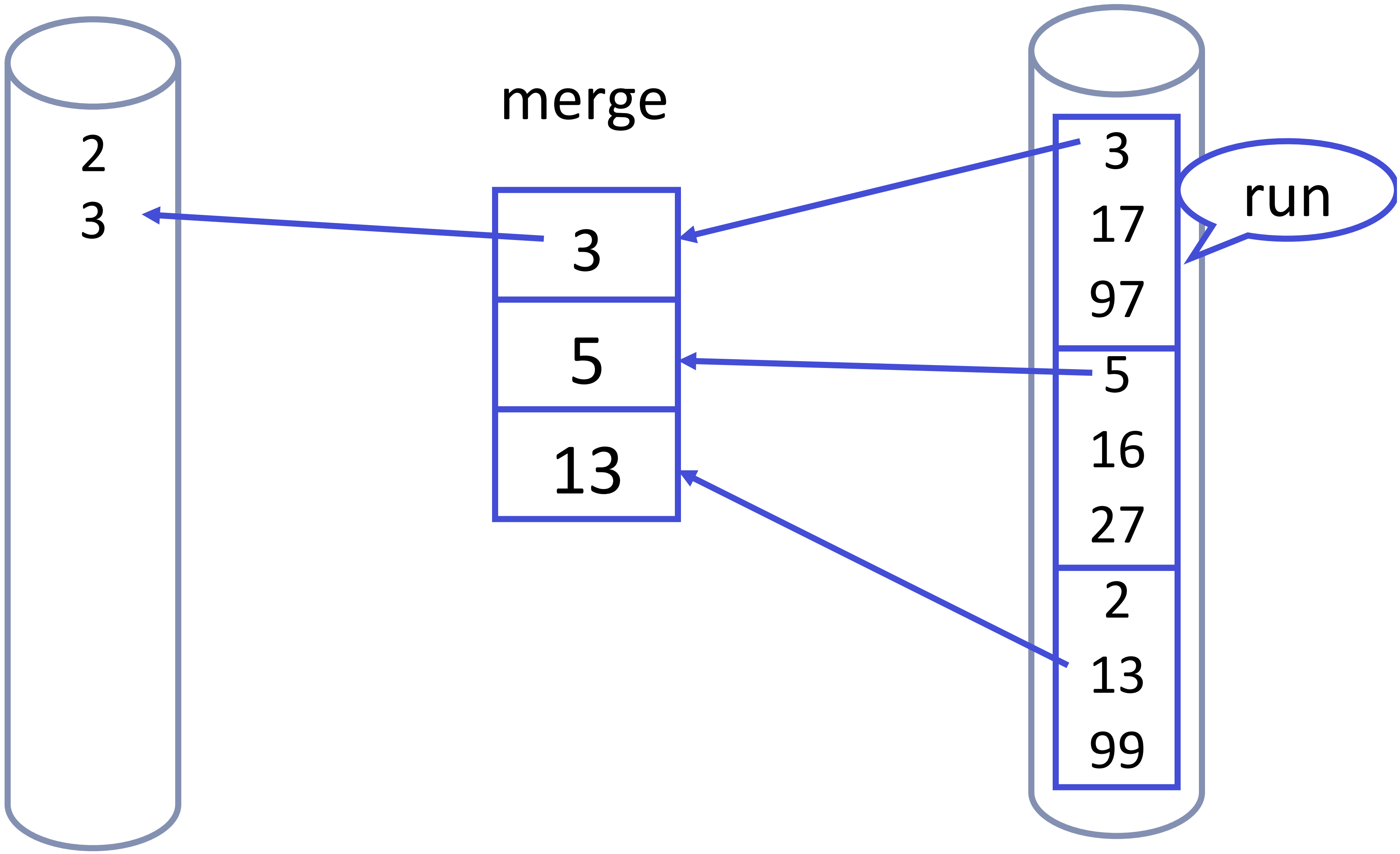
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting



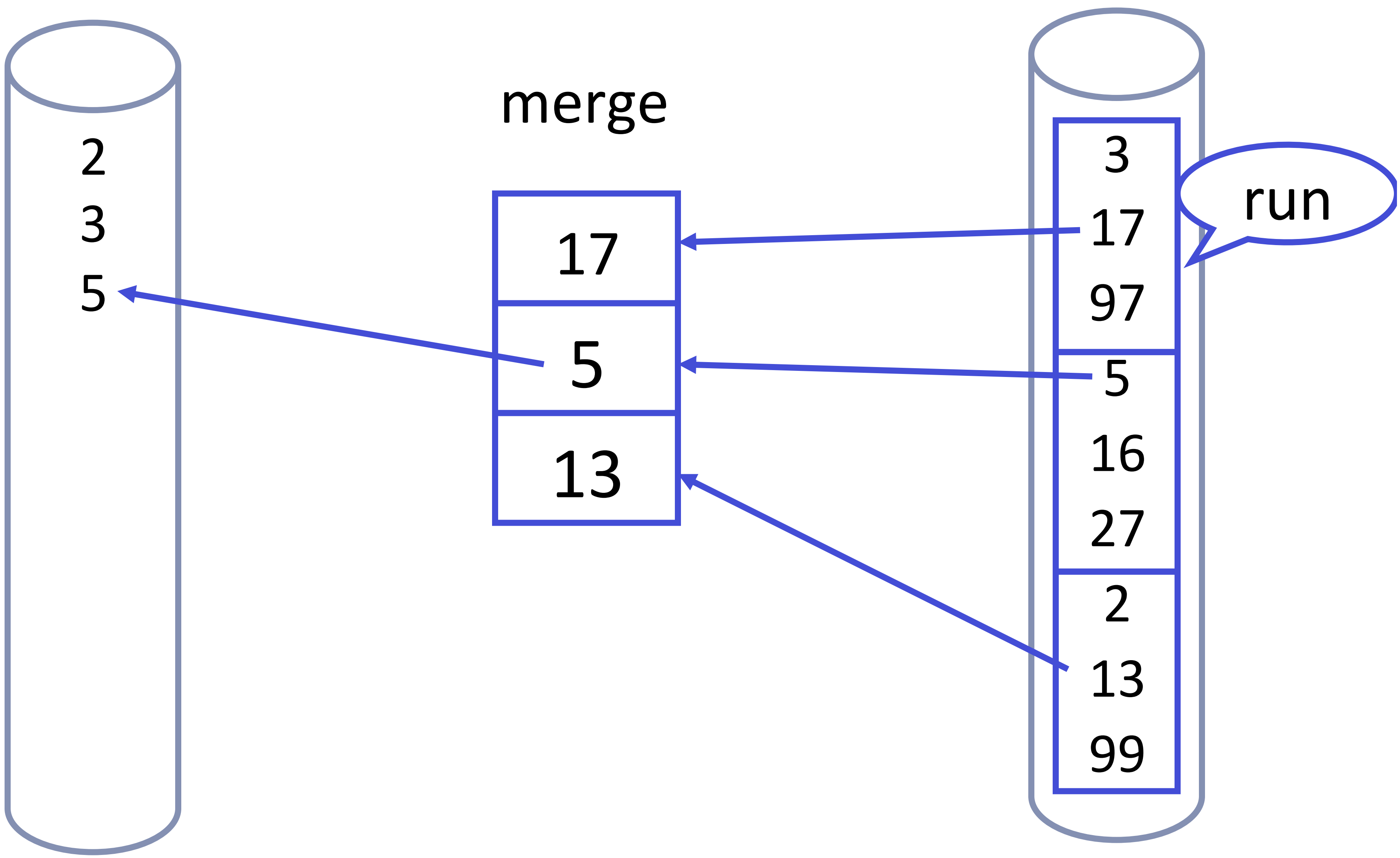
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting



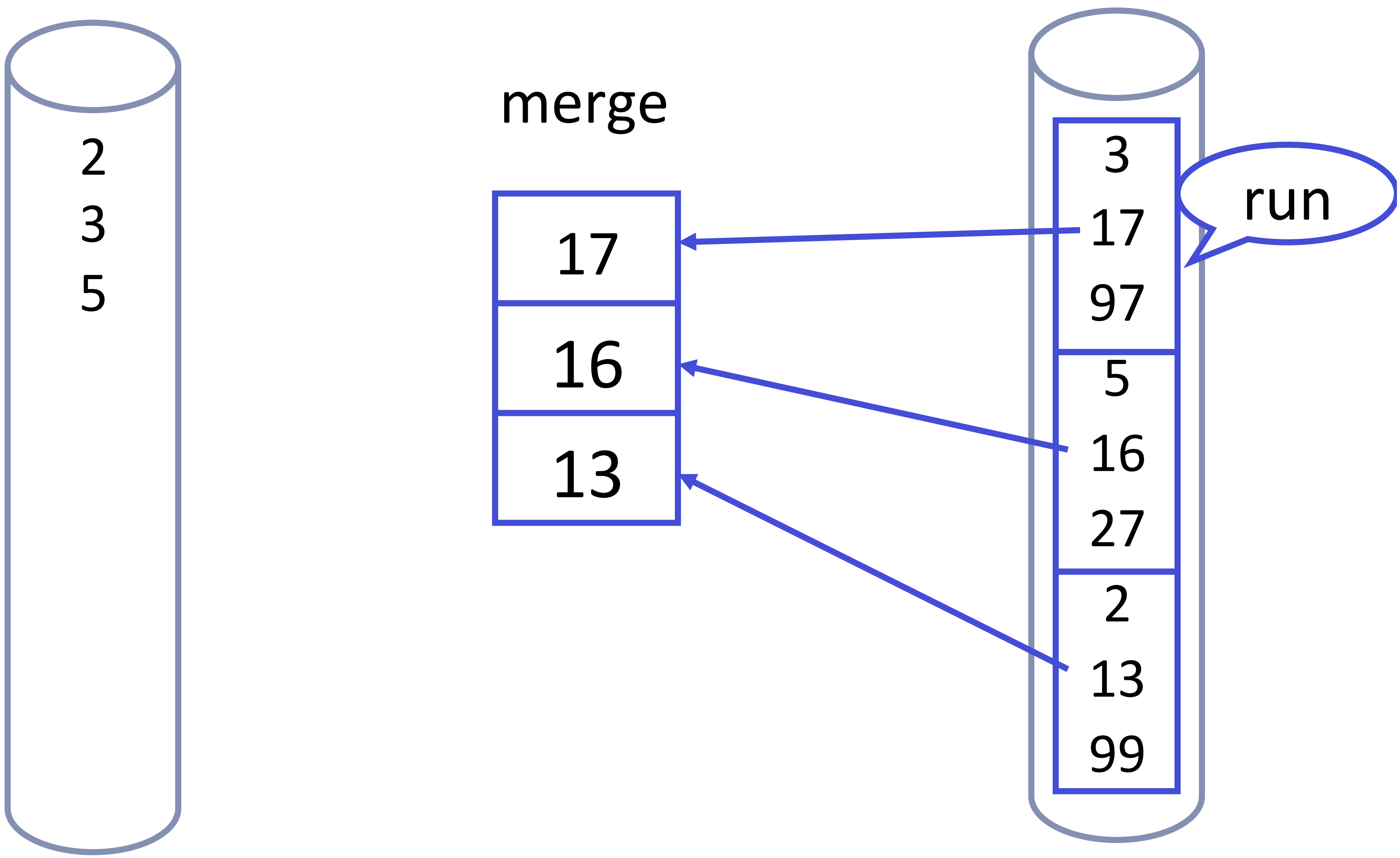
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting



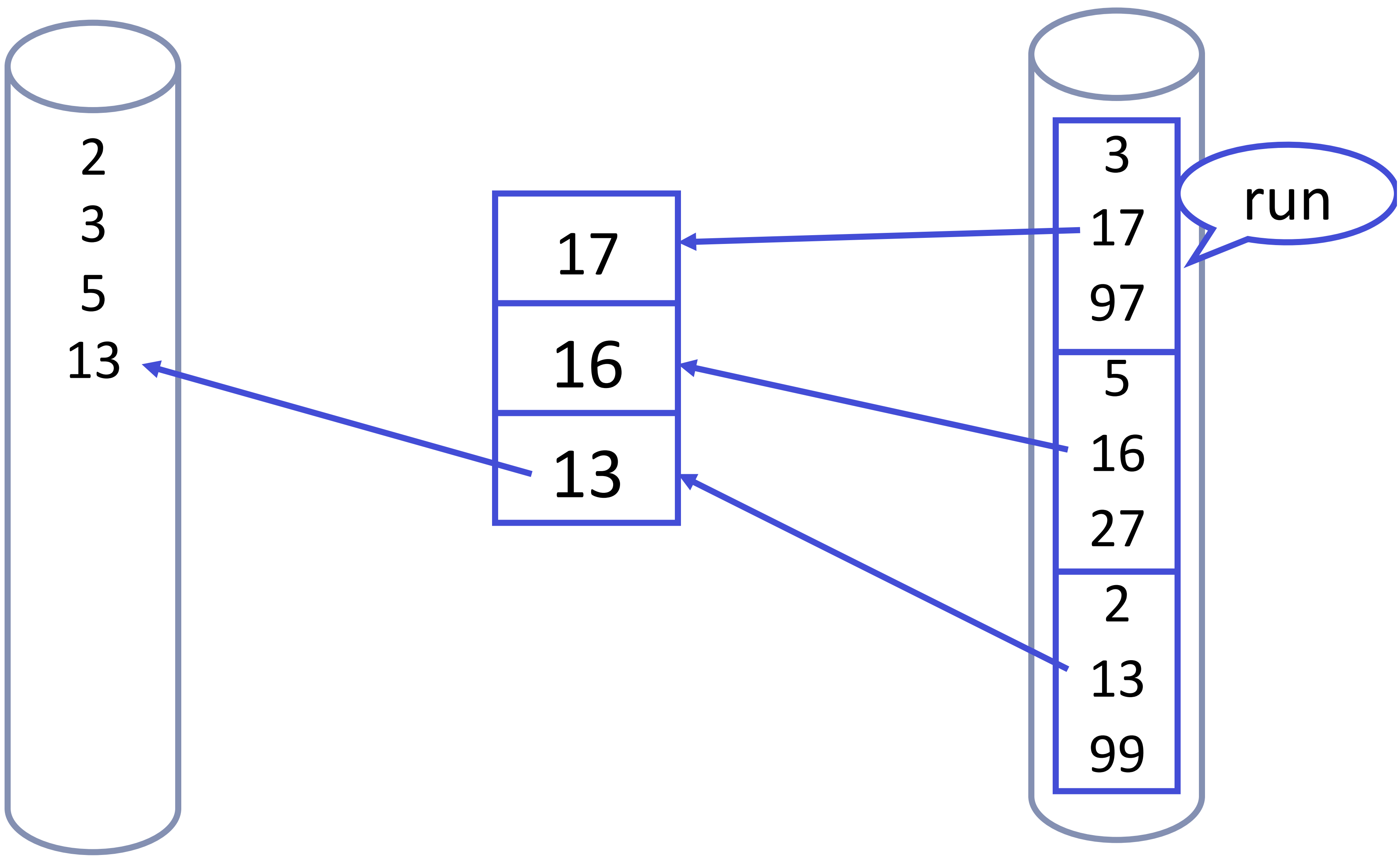
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting



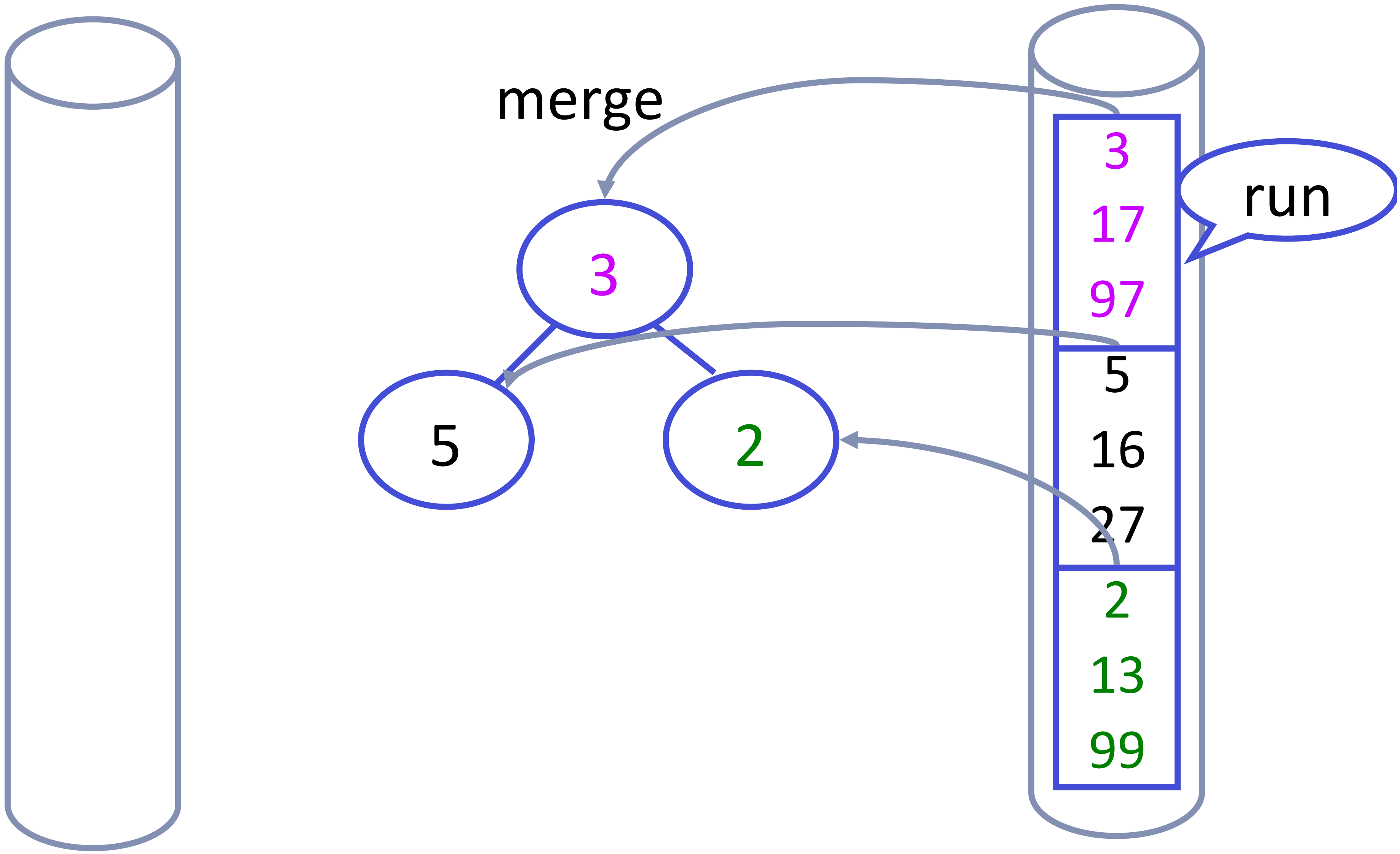
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sorting



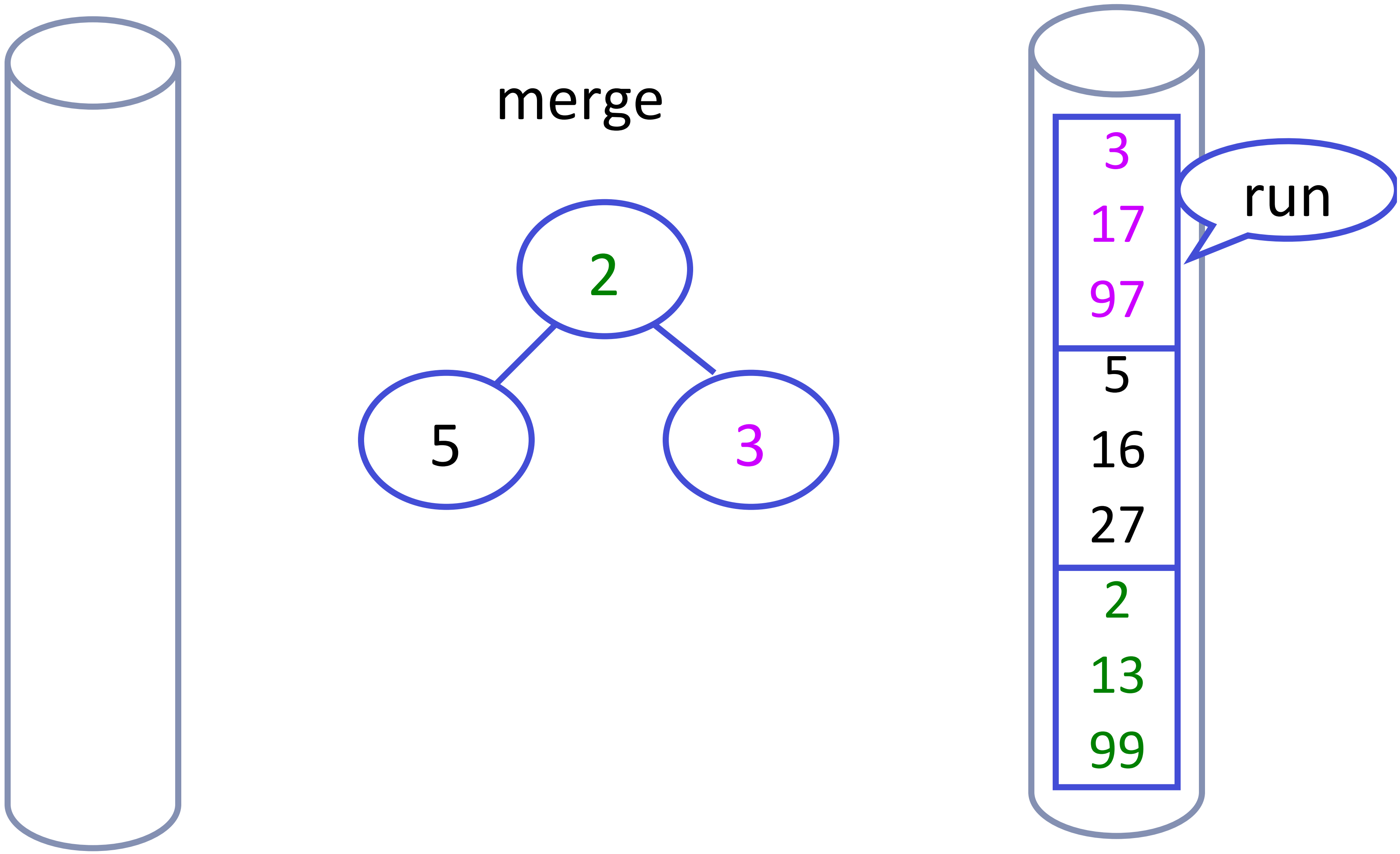
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sort w/ Heap-Priority Queue

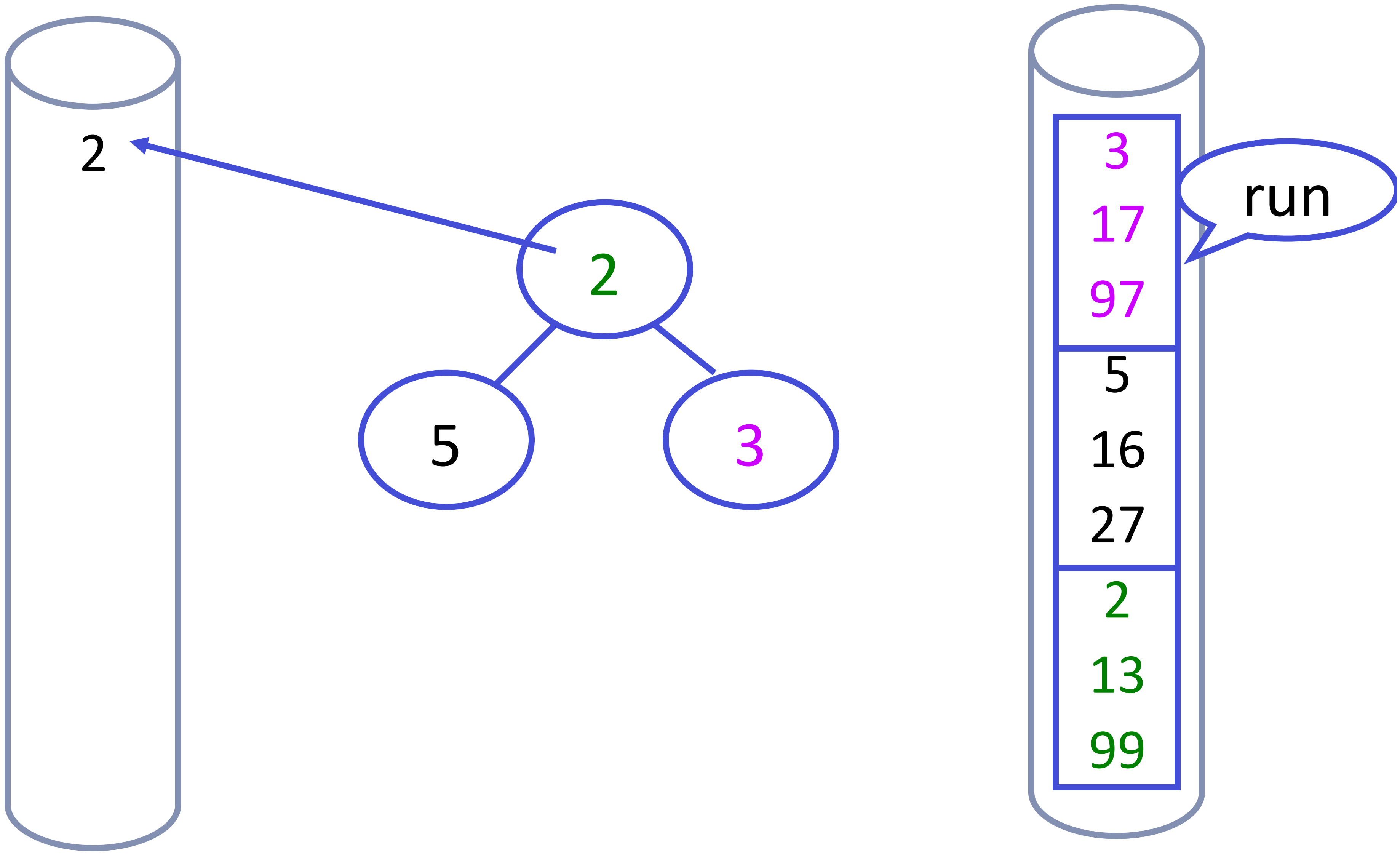


Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sort w/ Heap-Priority Queue

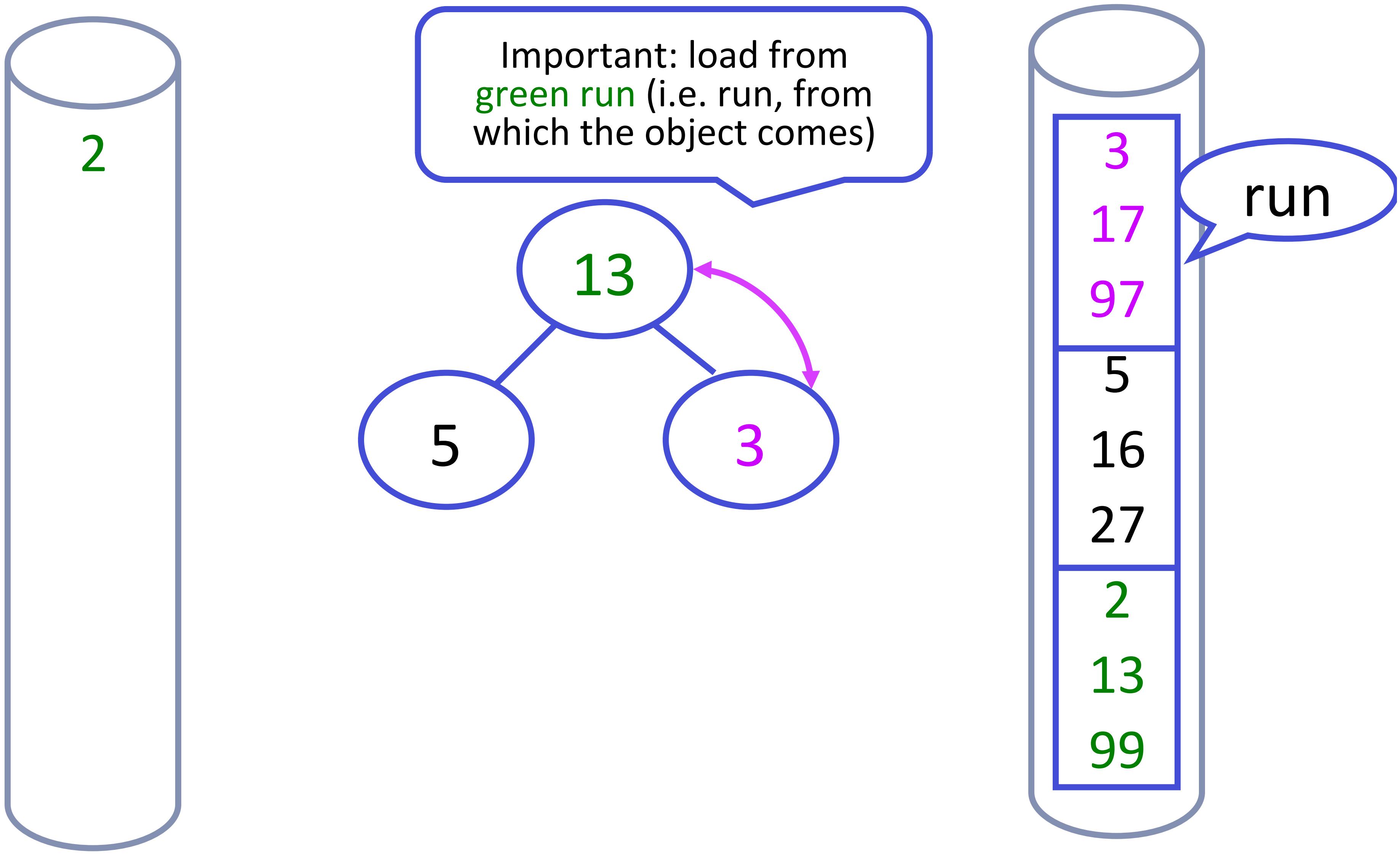


External Sort w/ Heap-Priority Queue

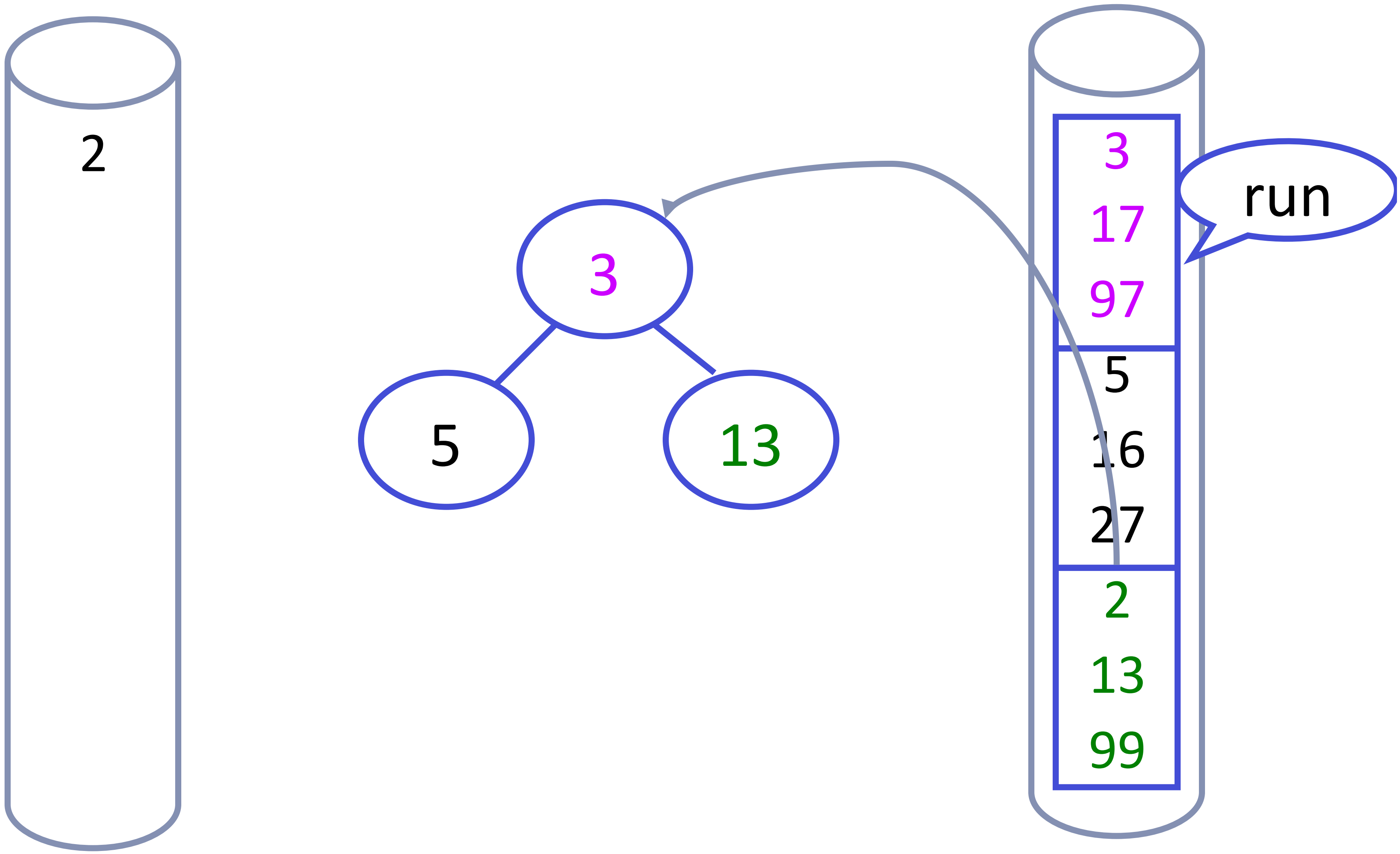


Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

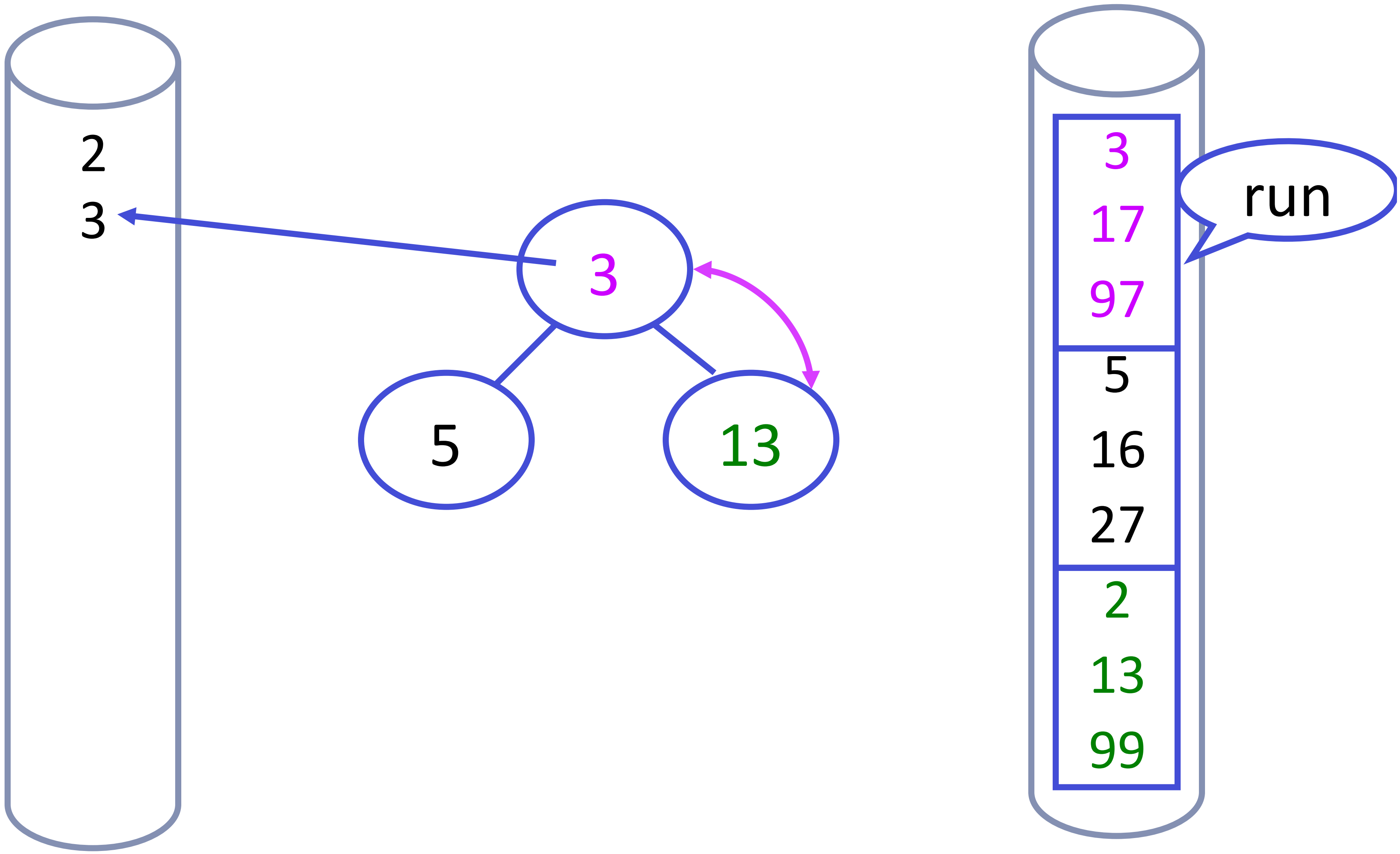
External Sort w/ Heap-Priority Queue



External Sort w/ Heap-Priority Queue

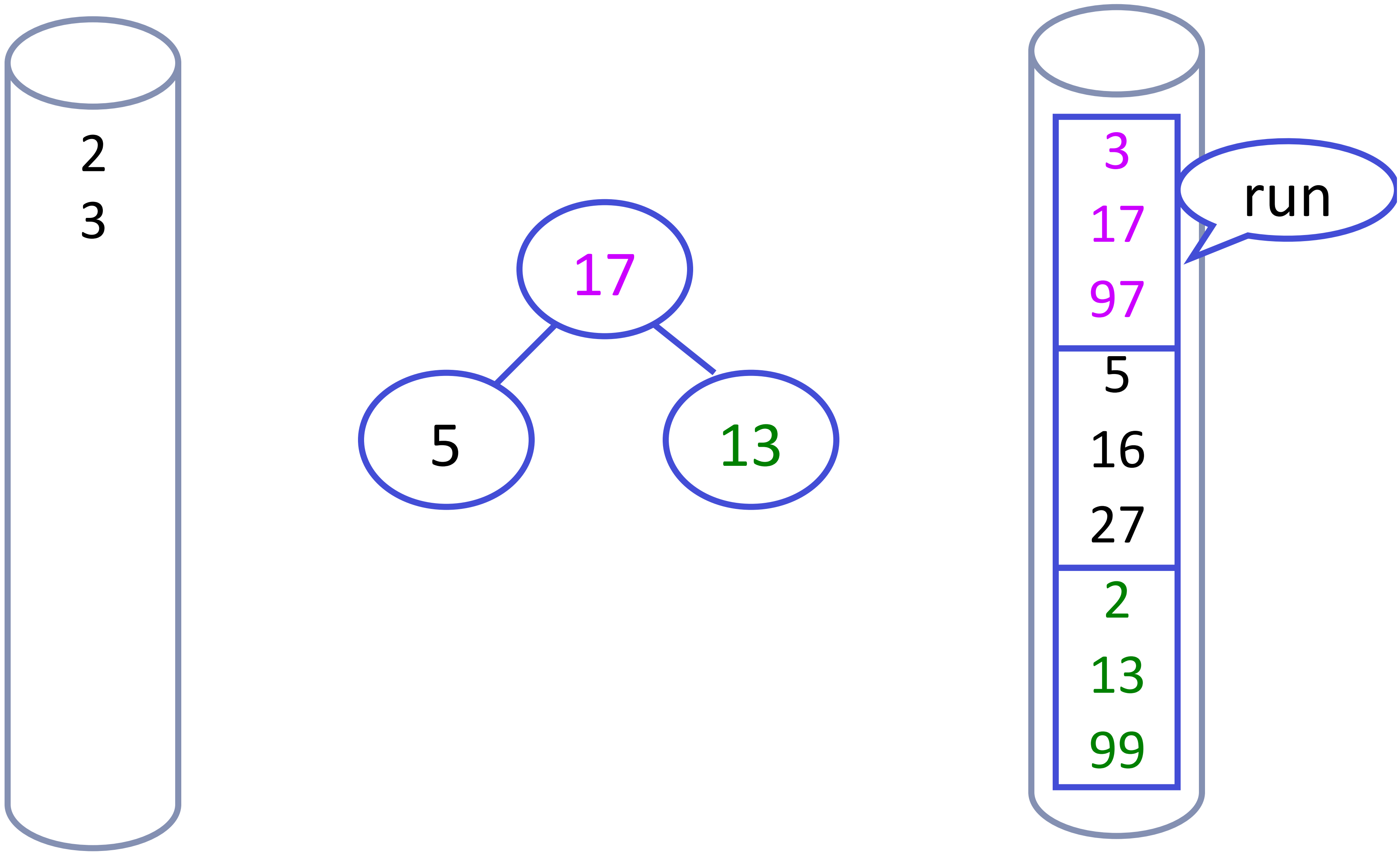


External Sort w/ Heap-Priority Queue



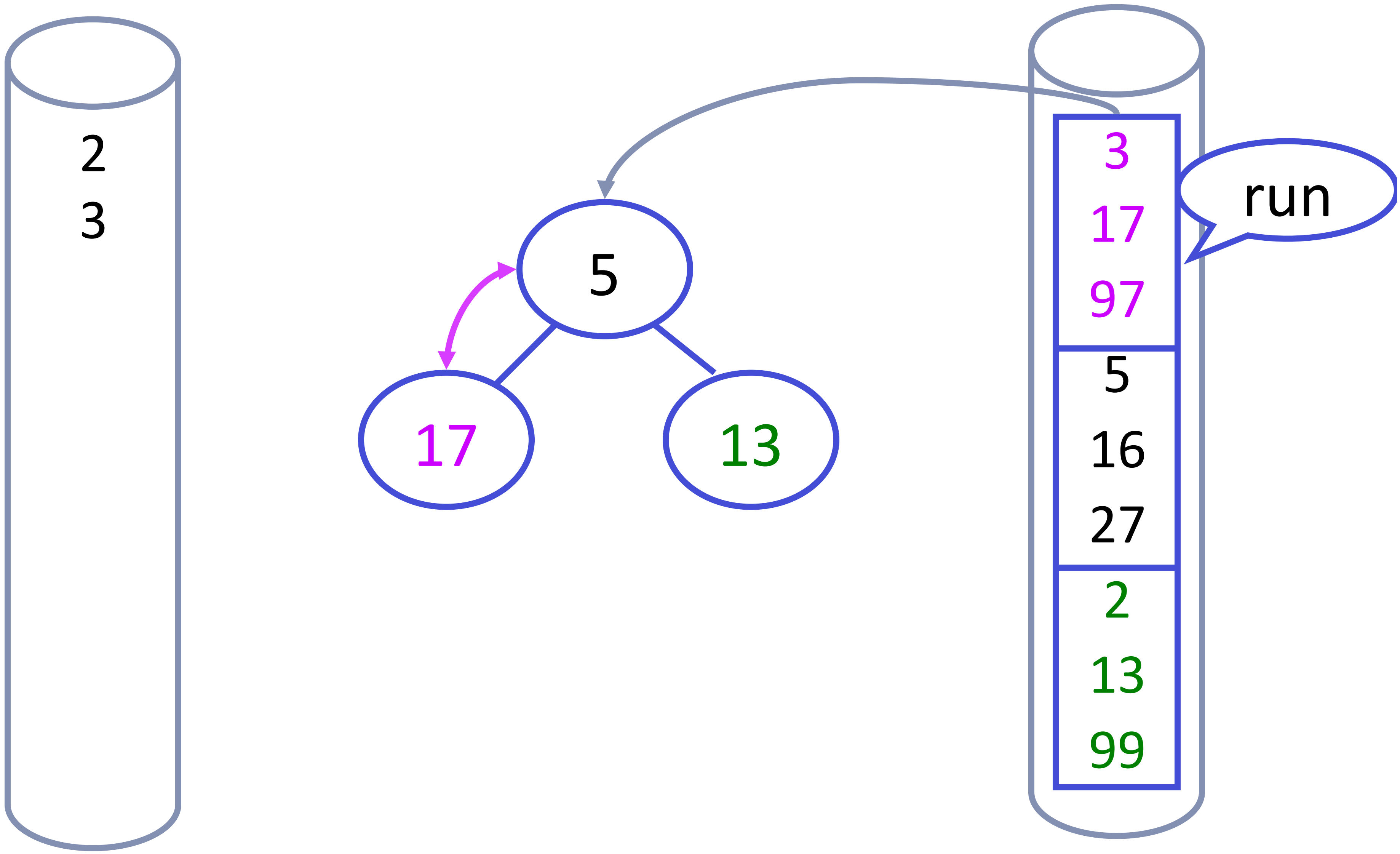
Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sort w/ Heap-Priority Queue



Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

External Sort w/ Heap-Priority Queue



R ⋈ S (Natural Join)

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

⋈

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

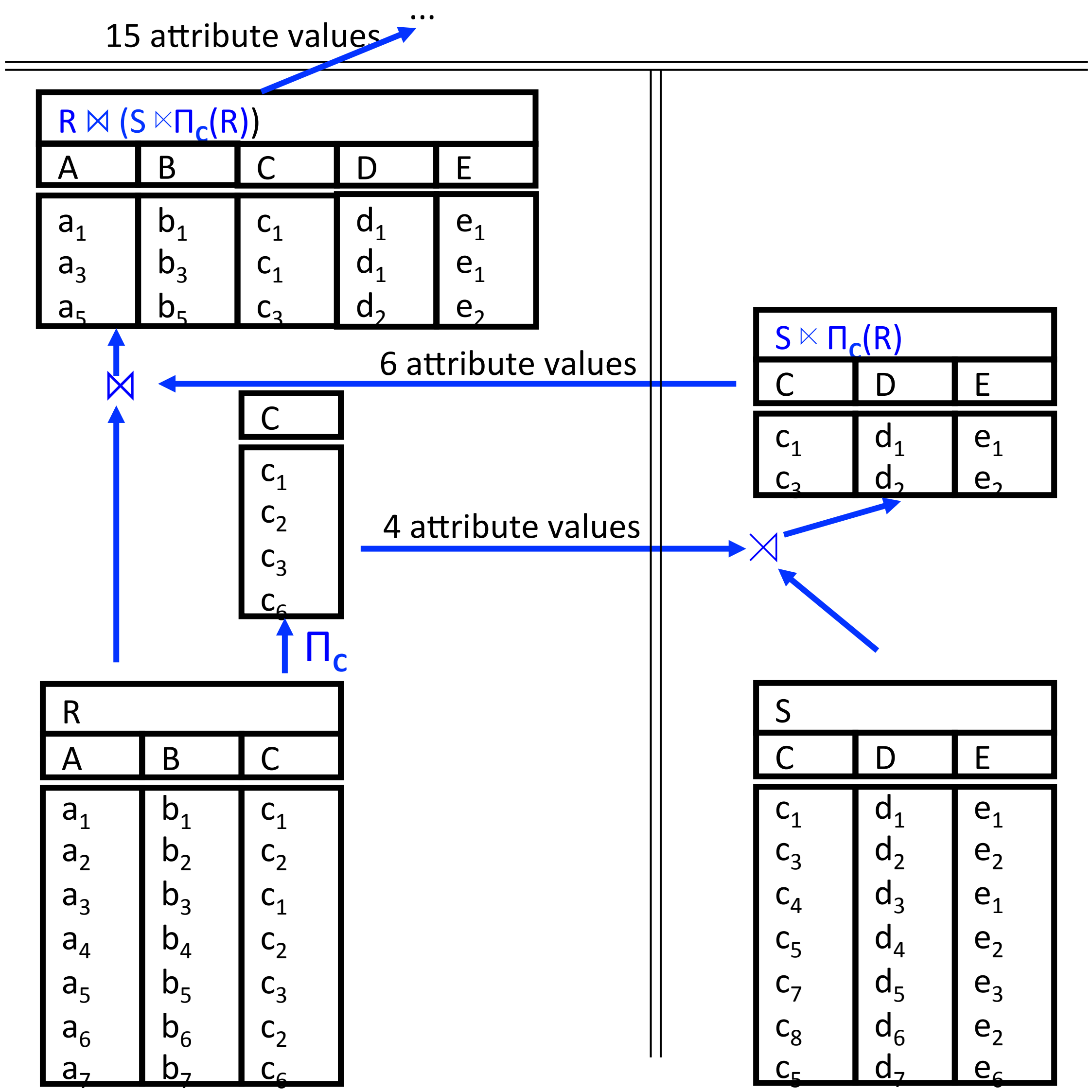
=

R ⋈ S

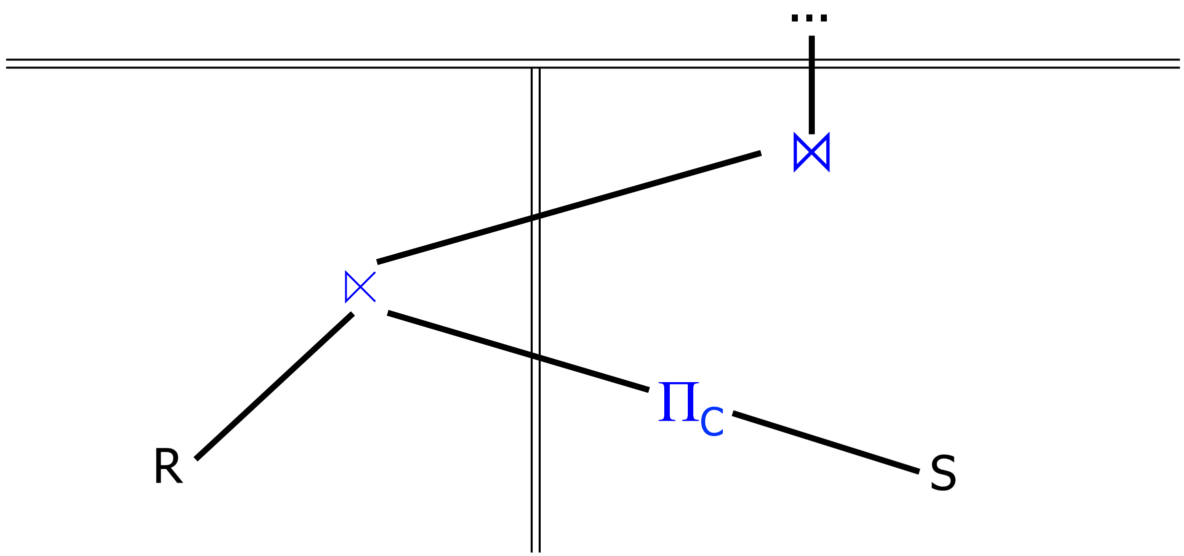
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_3	b_3	c_1	d_1	e_1
a_5	b_5	c_3	d_2	e_2

Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)

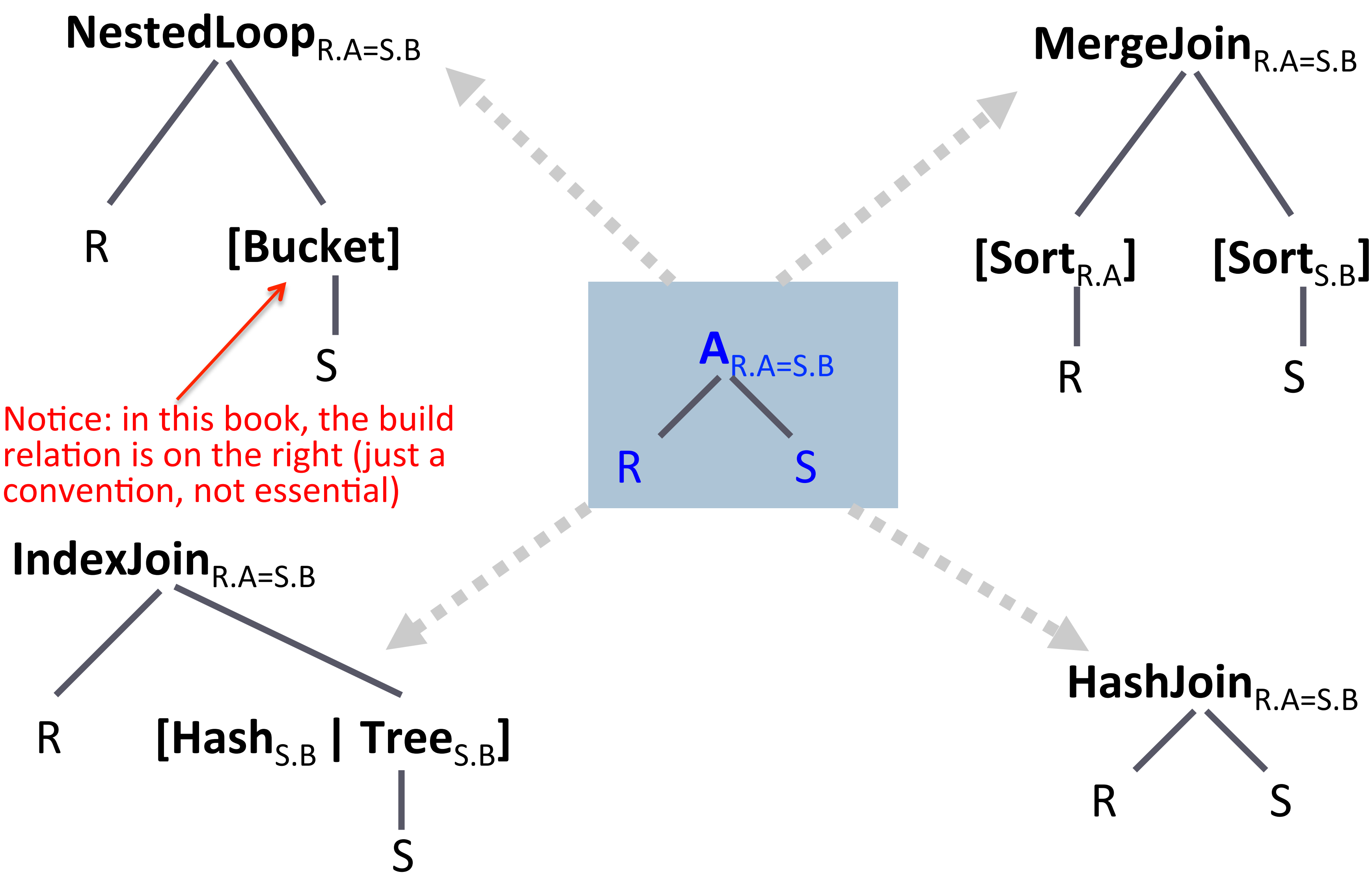
$R \bowtie S = R \bowtie (S \ltimes \Pi_C R)$



Alternative: R ⋈ S = (R ⋉ S) ⋈ S



Logical Algebra → Physical Operators



Source: Slides taken and adapted from: Kemper, Eickler: "Datenbanksysteme - Eine Einführung"
http://www3.in.tum.de/research/publications/books/DBMSeinf/EIS_4_Auflage/index.html (March 2011)