DATA516/CSED516
Scalable Data Systems and Algorithms

Lecture 1
Design of a Relational DBMS

Course Staff

- Instructor: Dan Suciu
  suciu@cs.washington.edu

- TA: Brandon Haynes
  bhaynes@cs.washington.edu

- TA: Deepanshu Gupta
  guptad2@cs.washington.edu

Coarse Aims

- Study design of big data systems
  - Historical perspective
  - Sample of modern systems
  - Breadth of designs (relational, streaming, graph, etc.)
- Study key scalable data processing algorithms
- Gain hands-on experience with big data systems
  - Demonstrations and tutorials in sections
  - Assignments and projects

Coarse Content

- Parallel query processing
- MapReduce, legacy, successors
- Some important “Big data” algorithms
- Streaming
- Column stores
- Graph engines

Course Format

- 5pm-7:50pm: Lectures
  - Discuss system architecture & algorithms
- 8pm-8:50pm: Hands-on tutorials
  - Learn how to use big data systems
  - Jump start your homeworks
  - Bring your laptop!

Grading (subject to change!)

- 15%: Reading assigned papers
  - Write short statement/review
- 45%: Three hands-on assignments
  - Redshift, Snowflake, Spark
- 15%: Three mini-assignments
  - Vertica, GraphX, Beam
- 25%: Final project

Project (1/2)

Choose a topic:

- Don’t worry about novelty!
- Highly recommended: Benchmark projects
  - Analyze the performance of some features
  - Compare the performance of different systems
  - Try to implement an interesting workload
- I will post a few ideas (some coming from local startups), but you are strongly encouraged to come up with your own

Project (2/2)

Timeline:

- Saturday Nov. 2nd: Proposal (1 pages)
- Saturday Nov. 23rd: Milestone (2-3 pp)
- Tuesday, Dec. 3rd: Presentations
- Tuesday, Dec. 10th: Final report (3-4 pp)

Web Services

- HW1: Amazon Redshift (the free tier)
- HW2: Snowflake (you will receive credits – stay tuned)
- HW3: AWS (you *received* email invitation to Amazon Educate)

Communication

- Course webpage: all important stuff https://courses.cs.washington.edu/courses/csed516/19au/
- Piazza: post “hello” tonight
- Class email: only for important announcements; if you are registered, you are automatically subscribed

How to Turn In

Gitlab: for homework and project

- Your own repository
- Pull to get homework instructions, starter files
- Push homework solutions, project reports

Google Forms: for paper reviews

- Typically around ½ page
- Goal is only for us to check that you have read the paper

Design of A Relational Database Management System

Outline

- Review of the relational model
- Converting from SQL to Relational Algebra
  - Foundation for going from declarative queries to query plans
- RDBMS architecture
  - All key components of a database system and their purpose
- Steps involved in query evaluation

Review

- Database is a collection of files
- Database management system (DBMS) is a piece of software to help manage that data
- History:
  - Origins in the 1960's
  - Relational model 1970
  - First relational DBMSs (Ingres and System R): 1970's
  - Parallel DBMSs: 1980's

DBMS Functionality

1. Describe real-world entities in terms of a data model
2. Create & persistently store large datasets
3. Efficiently query & update
    1. Must handle complex questions about data
    2. Must handle sophisticated updates
    3. Performance matters
4. Change structure (e.g., add attributes)
5. Concurrency control: enable simultaneous updates
6. Crash recovery
7. Access control, security, integrity

OUR FOCUS: Query large databases

Relational Data Model

- A Database is a collection of relations

- A Relation R is a subset of $\mathbf{S_1 \times S_2 \times \ldots \times S_n}$
  - Where $\mathbf{S_i}$ is the domain of attribute **i**
  - **n** is number of attributes of the relation
  - A relation is a set of tuples

- A Tuple t is an element of $\mathbf{S_1 \times S_2 \times \ldots \times S_n}$

Other names: relation = table; tuple = row

Discussion

- Rows in a relation:
  - Ordering immaterial (a relation is a set)
  - All rows are distinct – set semantics
  - Query answers may have duplicates – bag semantics

Data independence!

- Columns in a tuple:
  - Ordering is significant
  - Applications refer to columns by their names

Or is it?

- Domain of each column is a primitive type

Schema

- Relation schema: describes column heads
  - Relation name
  - Name of each field (or column, or attribute)
  - Domain of each field
  - The *arity* of the relation = # attributes

- Database schema: set of all relation schemas

Instance

- Relation instance: concrete table content
  - Set of tuples (also called records) matching the schema
  - The *cardinality* of the relation = # tuples (a.k.a. size)

- Database instance: set of all relation instances

What is the schema?
What is the instance?

Supplier

sno	sname	scity	sstate
1	s1	city 1	WA
2	s2	city 1	WA
3	s3	city 2	MA
4	s4	city 2	MA

What is the schema?
What is the instance?

Relation schema

Supplier(sno: integer, sname: string, scity: string, sstate: string)

Supplier

sno	sname	scity	sstate
1	s1	city 1	WA
2	s2	city 1	WA
3	s3	city 2	MA
4	s4	city 2	MA

instance

Relational Query Language

- Set-at-a-time:
  - Query inputs and outputs are relations

- Two variants of the query language:
  - Relational algebra: specifies order of operations
  - Relational calculus / SQL: declarative

Note

- We will go very quickly in class over the Relational Algebra and SQL

- Please review at home:
  - Review material from DATA514/CSED514

Structured Query Language: SQL

- Declarative query language

- Data definition language
  - Statements to create, modify tables and views
- Data manipulation language
  - Statements to issue queries, insert, delete data

SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT  <attributes>
FROM    <one or more relations>
WHERE   <conditions>
```

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,qty,price)

Part(pno,pname,psize,pcolor)

Quick Review of SQL

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,qty,price)

Part(pno,pname,psize,pcolor)

Quick Review of SQL

```
SELECT DISTINCT  z.pno, z.pname
FROM      Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
        and x.scity = 'Seattle'
        and y.price < 100
```

What does this query compute?

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,qty,price)

Part(pno,pname,psize,pcolor)

Quick Review of SQL

What about this one?

```
SELECT z.pno, z.pname,   count(*) as cnt,   min(y.price) as m
FROM      Supplier x, Supply y, Part z
WHERE   x.sno = y.sno
        and y.pno = z.pno
GROUP BY z.pname
```

SQL – Summary

- Very complex: >1000 pages,
  - No vendor supports full standard; (in practice, people use postgres as *de facto* standard)
  - Much more than DML
- It is a *declarative* language:
  - we say what we want
  - we don't say how to get it
- Relational algebra says how to get it

Relational Algebra

- Queries specified in an operational manner
  - A query gives a step-by-step procedure

- Relational operators
  - Take one or two relation instances as input
  - Return one relation instance as result
  - Easy to compose into relational algebra expressions

Five Basic Relational Operators

- Selection: $\sigma_{condition}(S)$
  - Condition is Boolean combination ($\wedge$,$\vee$) of atomic predicates (<, <=, =, ≠, >=, >)
- Projection: $\pi_{list\text{-}of\text{-}attributes}(S)$
- Union ($\cup$)
- Set difference (–),
- Cross-product/cartesian product ($\times$),
  Join: $R \bowtie_{\theta} S = \sigma_{\theta}(R \times S)$

Other operators: anti-semijoin, renaming

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,qty,price)

Part(pno,pname,psize,pcolor)

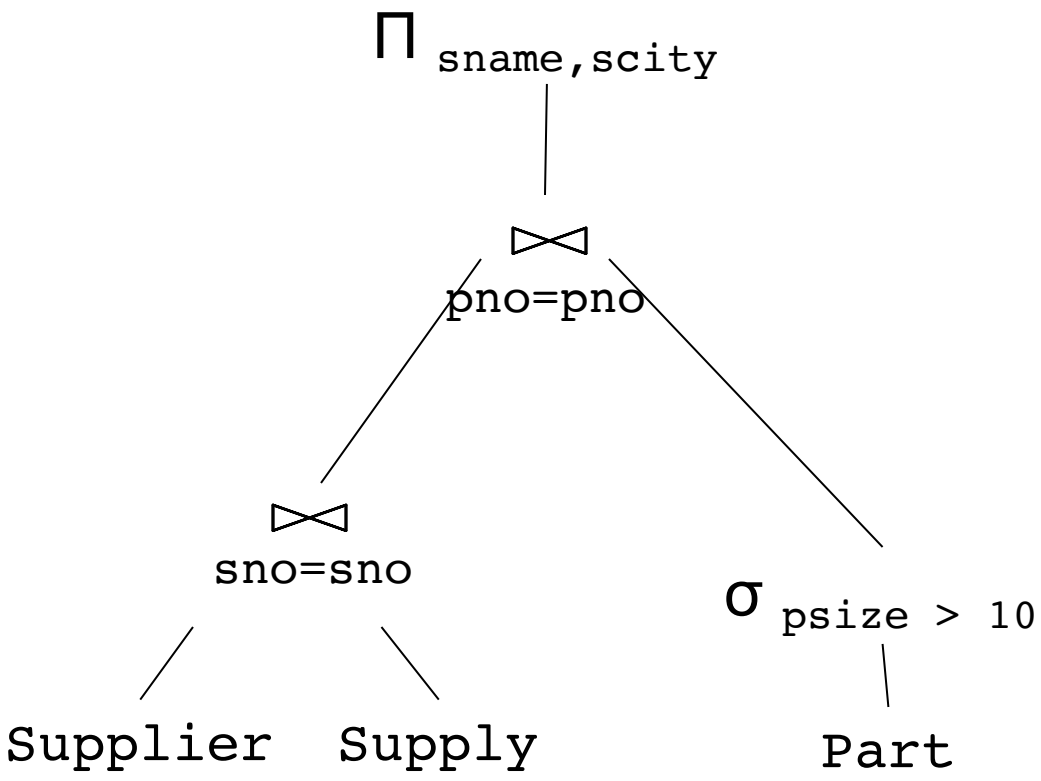
Logical Query Plans

Supplier(sno,sname,scity,sstate)

Supply(sno,pno,qty,price)

Part(pno,pname,psize,pcolor)

Logical Query Plans



What does this query compute?

Extended Operators of Relational Algebra

- Duplicate elimination (δ)
  - Since commercial DBMSs operate on multisets not sets
- Group-by/aggregate (γ)
  - Min, max, sum, average, count
  - Partitions tuples of a relation into "groups"
  - Aggregates can then be applied to groups
- Sort operator (τ)

Relational Model -- Summary

- Schema v.s. Data

Relational Model -- Summary

- Schema v.s. Data
- Data is *normalized* (what is that?)

Relational Model -- Summary

- Schema v.s. Data
- Data is *normalized* (what is that?)
  - 1st NF: relations are flat (also unordered)
  - BCNF (or 3rd or 4th NF…): split large table into many small (why?), need to join back

Relational Model -- Summary

- Schema v.s. Data
- Data is *normalized* (what is that?)
  - 1st NF: relations are flat (also unordered)
  - BCNF (or 3rd or 4th NF…): split large table into many small (why?), need to join back
  - (Consequence: joins are *really* important)

Relational Model -- Summary

- Schema v.s. Data
- Data is *normalized* (what is that?)
  - 1st NF: relations are flat (also unordered)
  - BCNF (or 3rd or 4th NF…): split large table into many small (why?), need to join back
  - (Consequence: joins are *really* important)
- Query language is SQL, or something equivalent, like relational algebra

Outline

- Review of the relational model
- Converting from SQL to Relational Algebra
  - Foundation for going from declarative queries to query plans
- RDBMS architecture
  - All key components of a database system and their purpose
- Steps involved in query evaluation

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

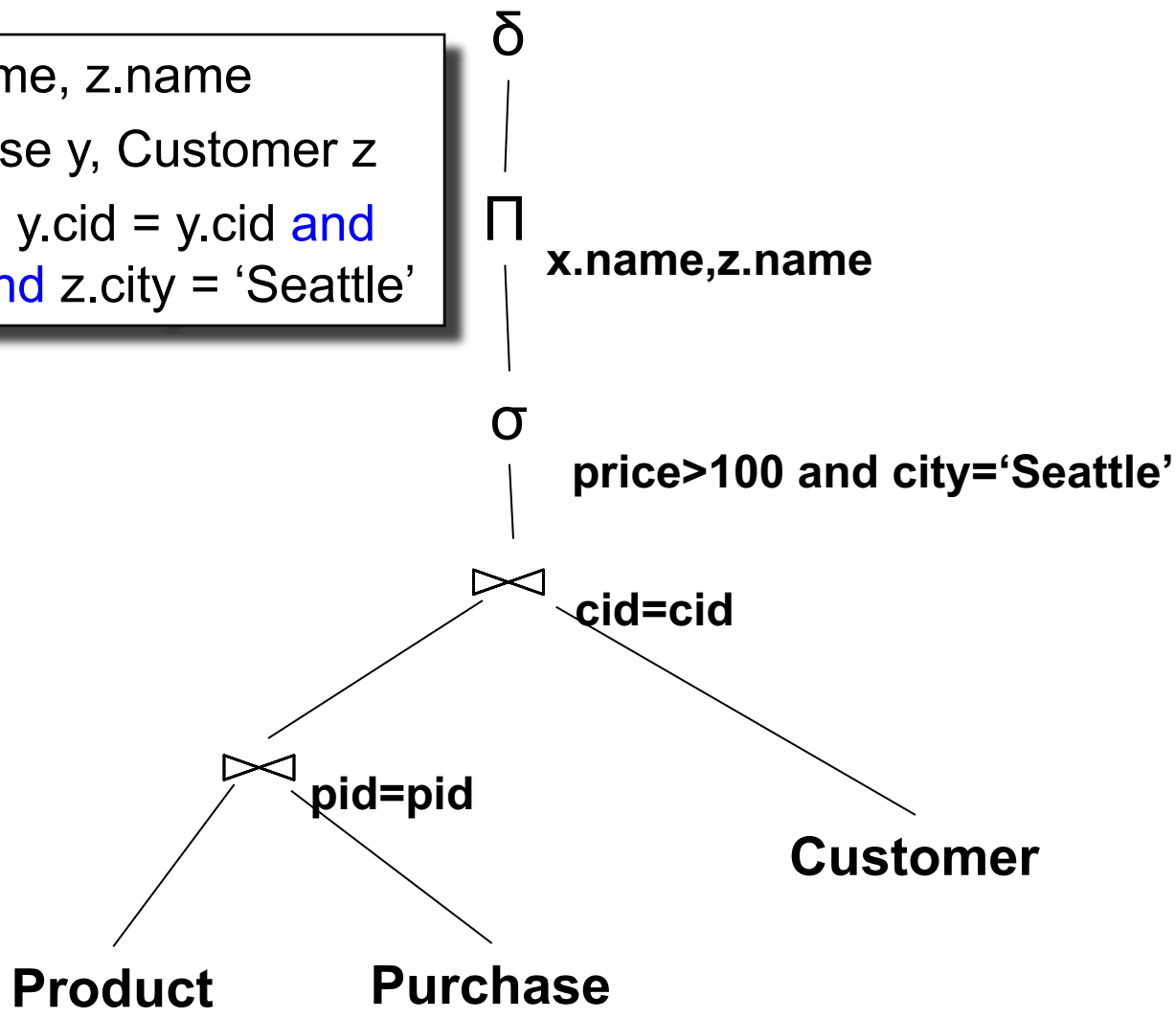
From SQL to RA

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
      x.price > 100 and z.city = 'Seattle'
```

Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

From SQL to RA

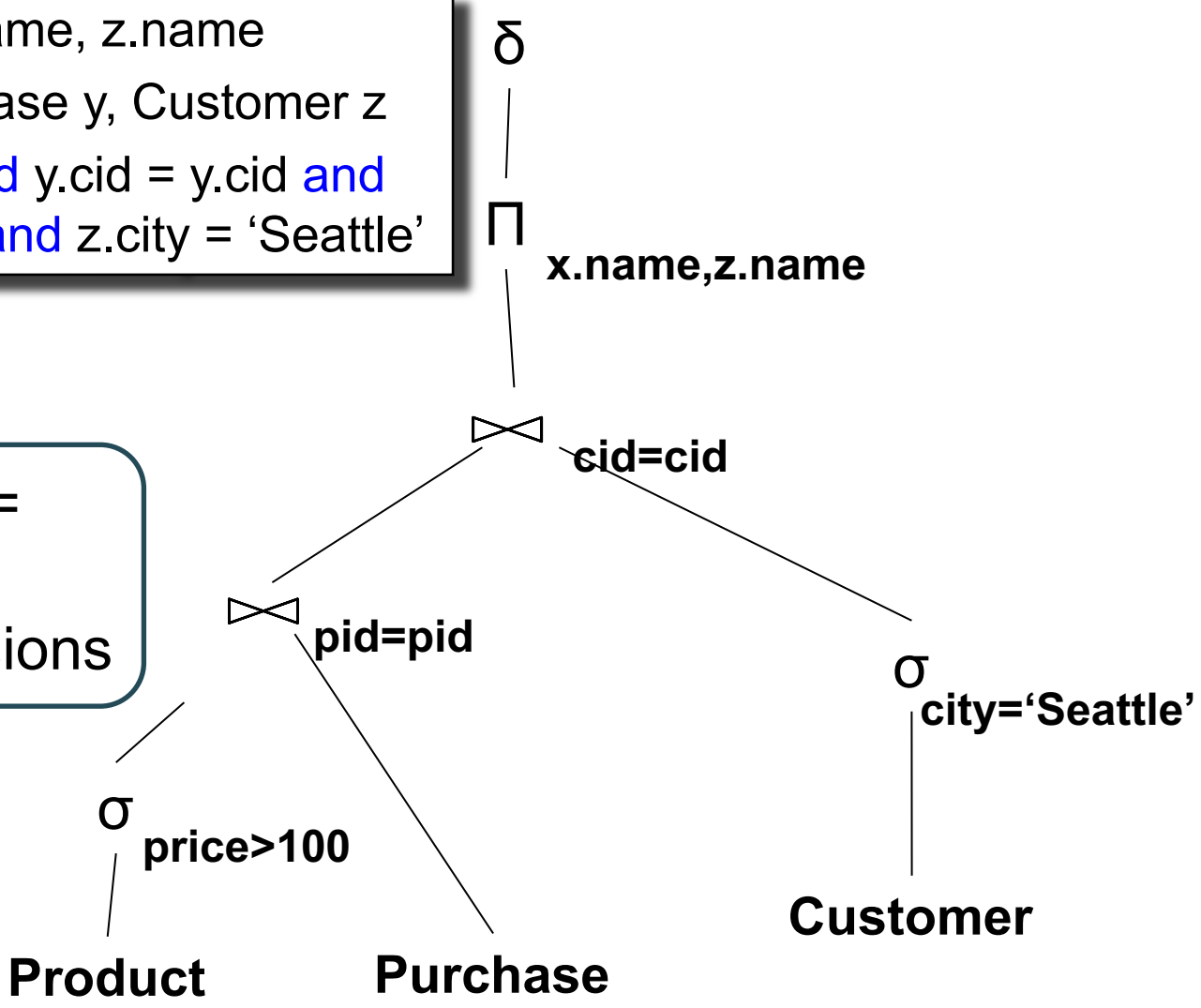
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
      x.price > 100 and z.city = 'Seattle'
```



Product(pid, name, price)
Purchase(pid, cid, store)
Customer(cid, name, city)

Equivalent Expression

```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = y.cid and
      x.price > 100 and z.city = 'Seattle'
```



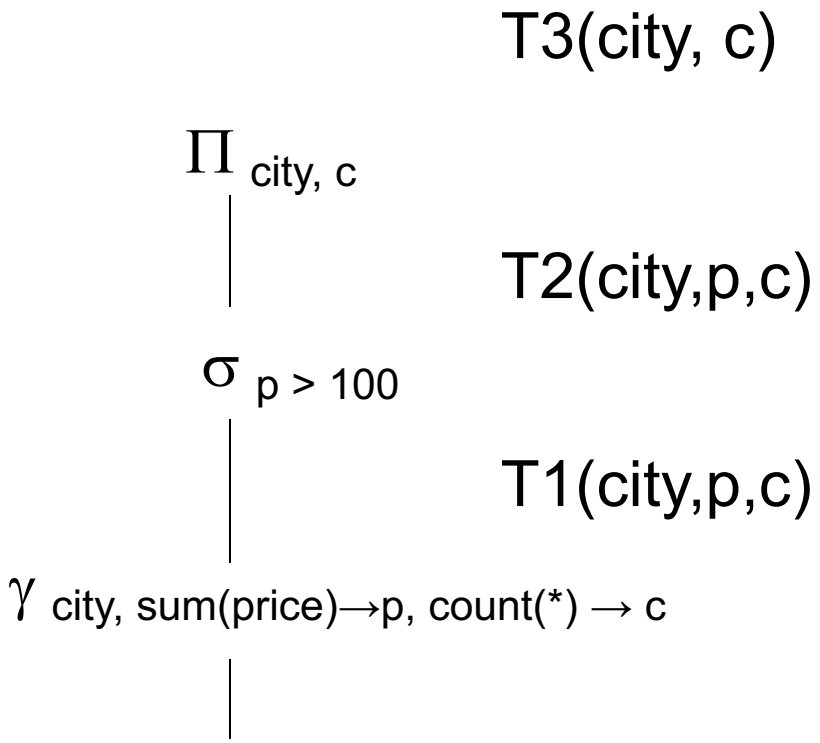
Query optimization = finding cheaper, equivalent expressions

Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
- Sorting τ

Logical Query Plan

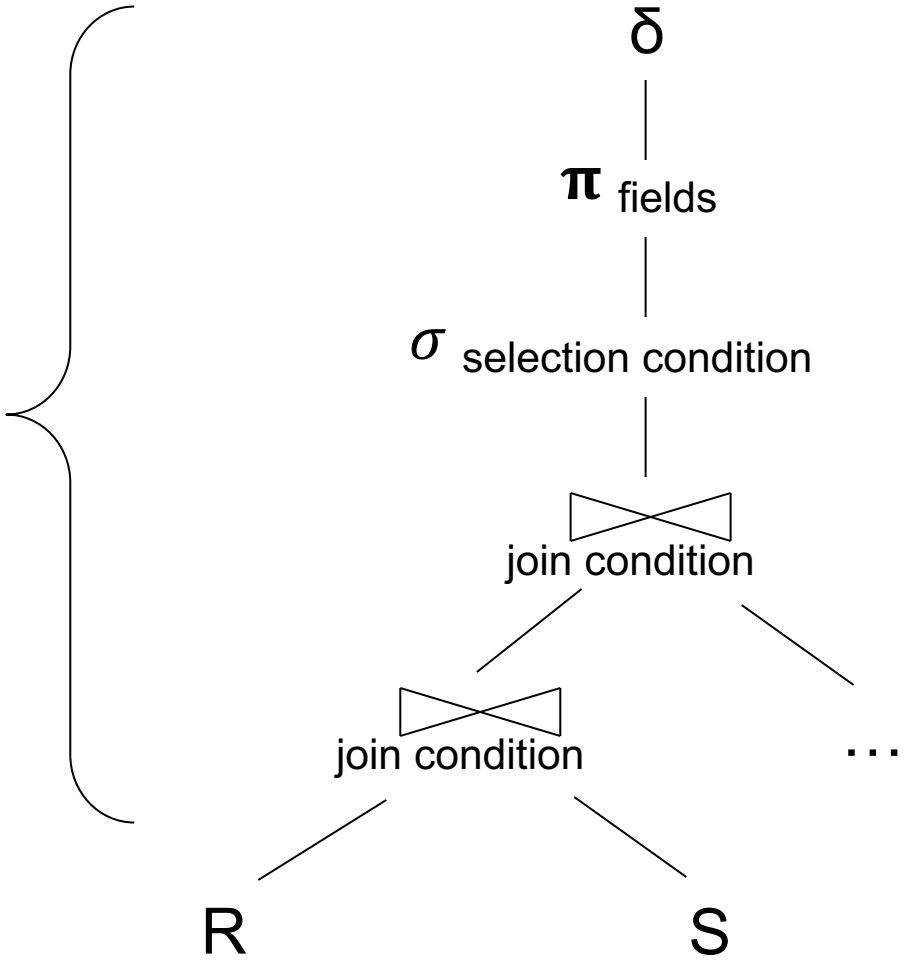
```
SELECT city, count(*)
FROM sales
GROUP BY city
HAVING sum(price) > 100
```



T1, T2, T3 = temporary tables

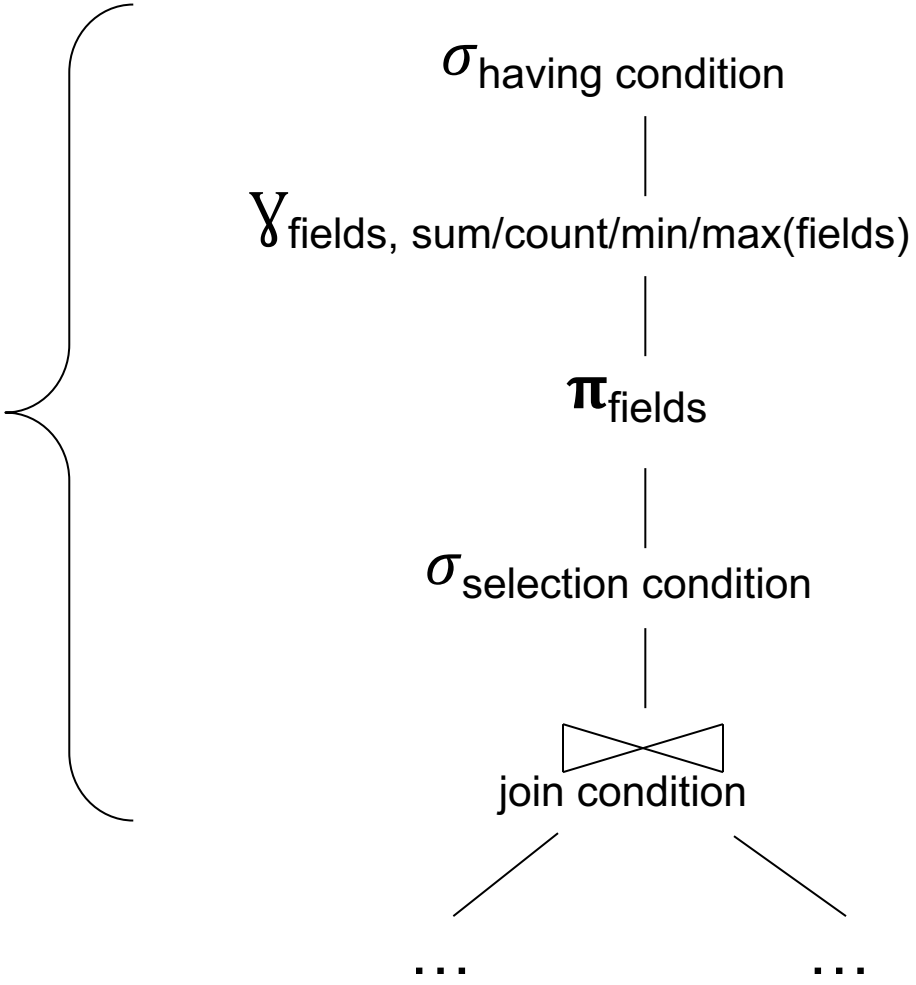
Typical Plan for Block (1/2)

SELECT [DISTINCT] fields…
FROM R, S, …
WHERE join conditions,
selection conditions



Typical Plan For Block (2/2)

SELECT fields, sum/count/…
FROM R, S, …
WHERE join conditions,
selection conditions
GROUP BY fields
HAVING condition



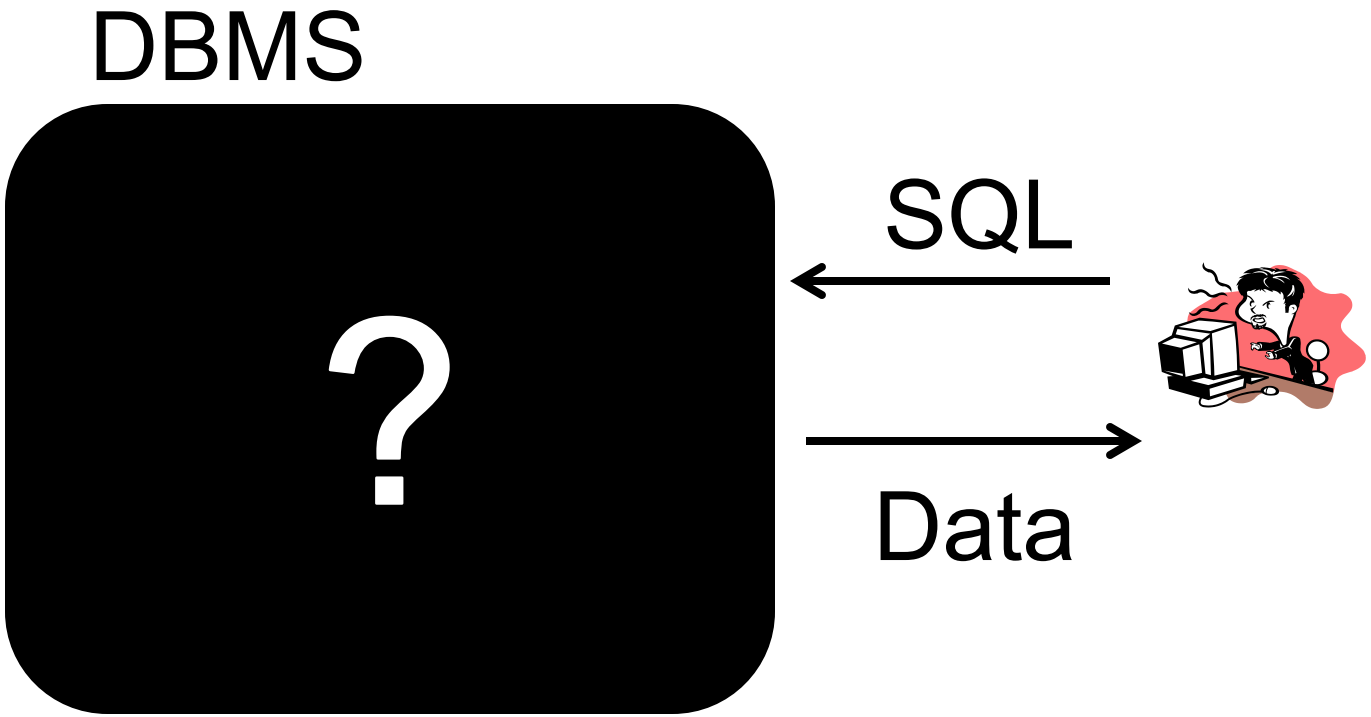
Benefits of Relational Model

- Physical data independence
  - Can change how data is organized on disk without affecting applications
- Logical data independence
  - Can change the logical schema without affecting applications (not 100%... consider updates)

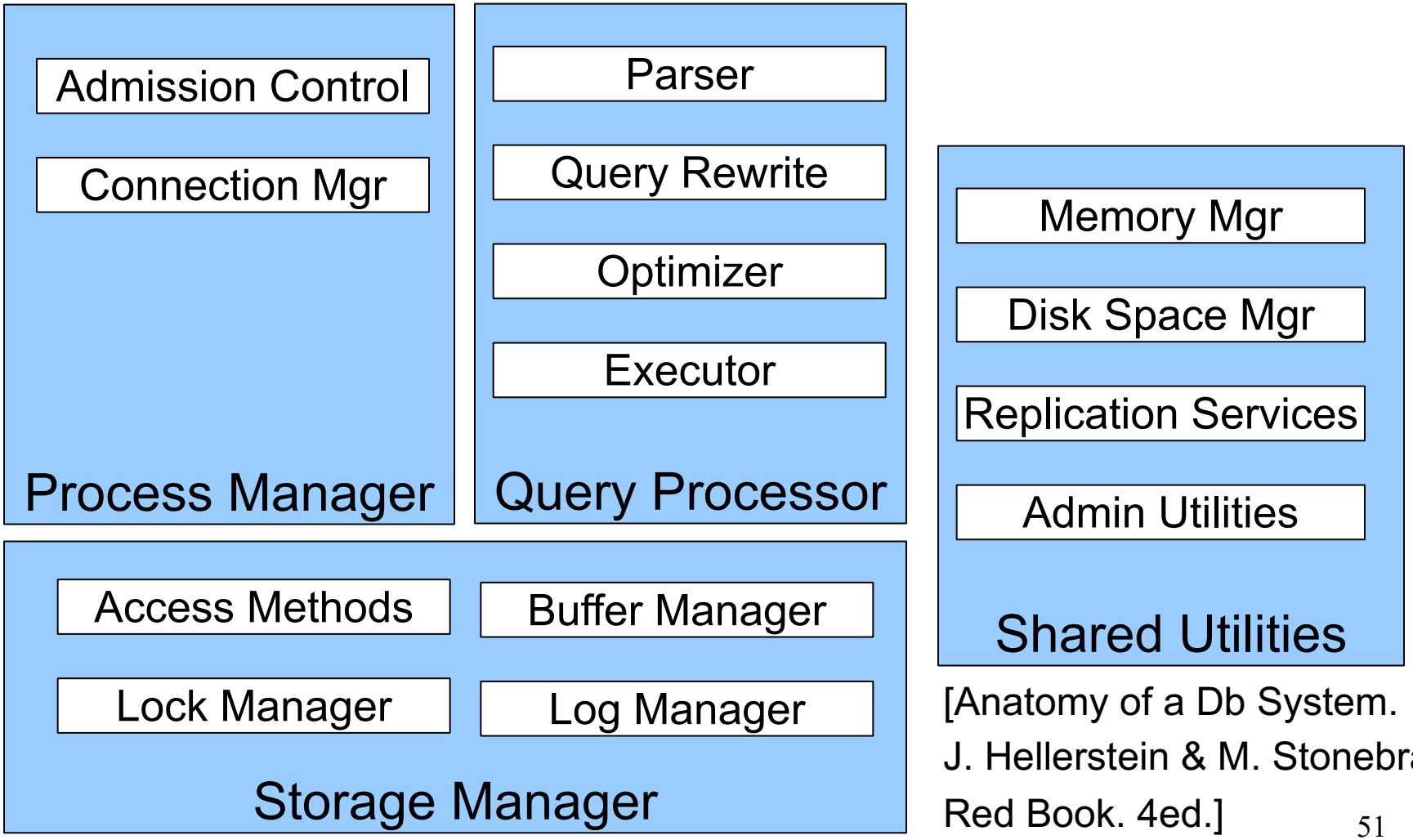
Outline

- Review of the relational model
- Converting from SQL to Relational Algebra
  - Foundation for going from declarative queries to query plans
- RDBMS architecture
  - All key components of a database system and their purpose
- Steps involved in query evaluation

How to Implement a Relational DBMS?



DBMS Architecture

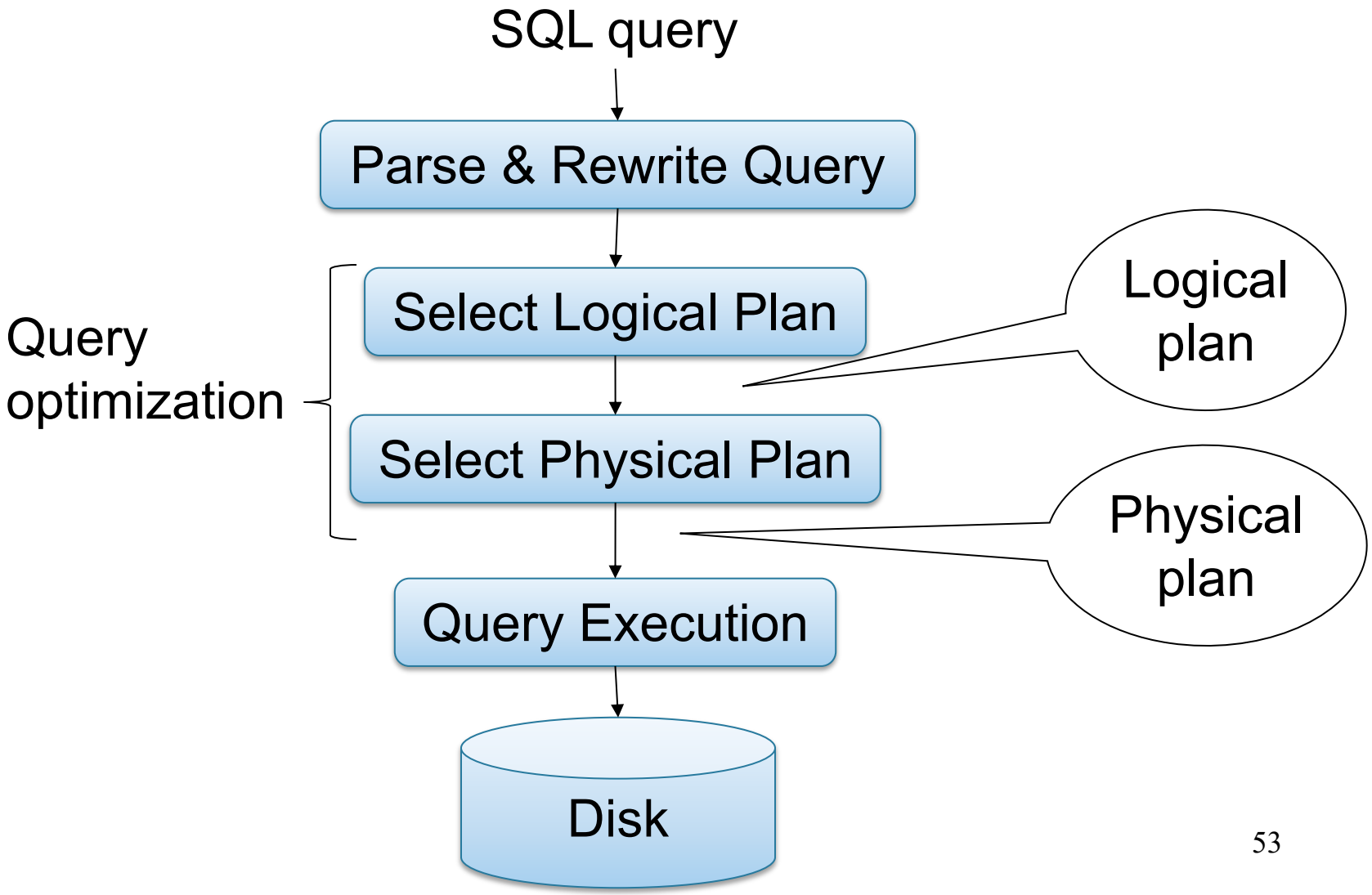


[Anatomy of a Db System. J. Hellerstein & M. Stonebraker. Red Book. 4ed.]

Outline

- Review of the relational model
- Converting from SQL to Relational Algebra
  - Foundation for going from declarative queries to query plans
- RDBMS architecture
  - All key components of a database system and their purpose
- Steps involved in query evaluation

Query Evaluation Steps Review



Example Database Schema

```
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)
```

View: Suppliers in Seattle

```
CREATE VIEW NearbySupp AS
  SELECT sno, sname
  FROM Supplier
  WHERE scity='Seattle' AND sstate='WA'
```

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

Example Query

- Find the names of all suppliers in Seattle who supply part number 2

```
SELECT sno, sname FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Query Processor

- **Step 1: Parser**
  - Parses query into an internal format
  - Performs various checks using **catalog**
    - Correctness, authorization, integrity constraints
    - Typically, catalog is stored in the form of set of relations

- **Step 2: Query rewrite**
  - View rewriting, flattening, etc.

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

Rewritten Version of Our Query

Original query:

```
SELECT sno, sname
FROM NearbySupp
WHERE sno IN ( SELECT sno
               FROM Supplies
               WHERE pno = 2 )
```

Rewritten query:

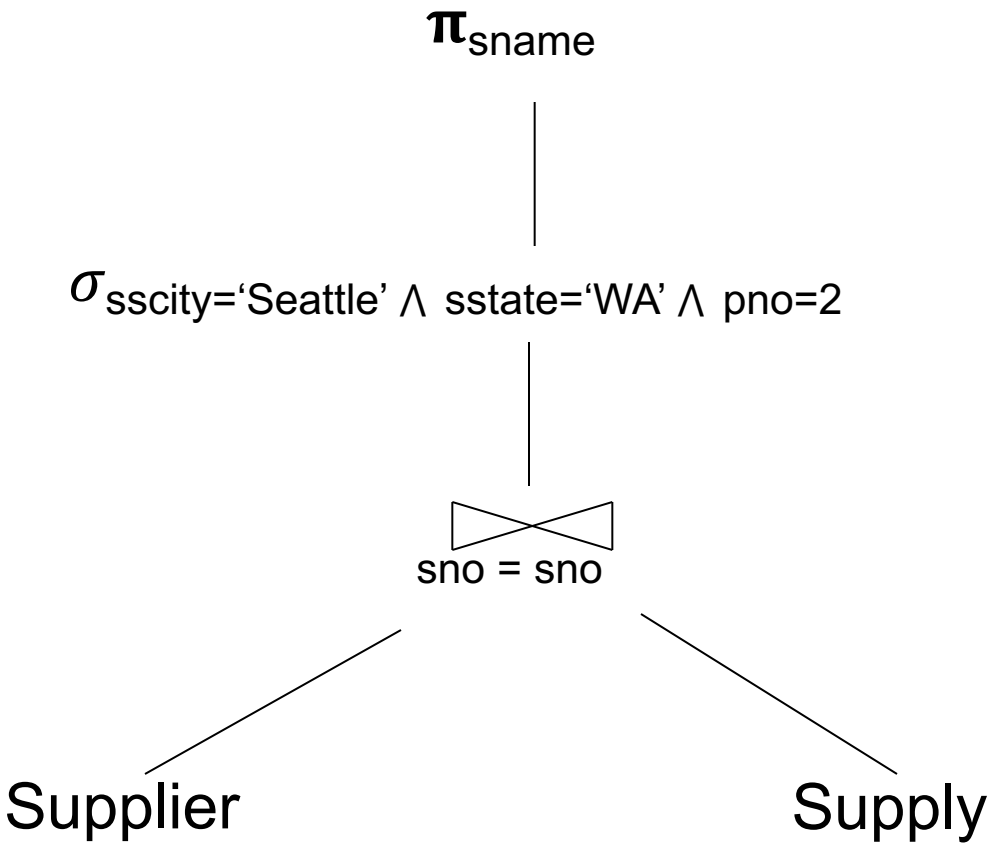
```
SELECT DISTINCT S.sno, S.sname
FROM Supplier S, Supplies U
WHERE S.scity='Seattle' AND S.sstate='WA'
AND S.sno = U.sno
AND U.pno = 2;
```

Query Processor

- **Step 3: Optimizer**
  - Find an efficient query plan for executing the query
  - **Query plan:**
    - **Logical**: An extended relational algebra tree
    - **Physical**: With additional annotations at each node
      - Access method to use for each relation
      - Implementation to use for each relational operator
- **Step 4: Executor**
  - Actually executes the physical plan

Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

Logical Query Plan

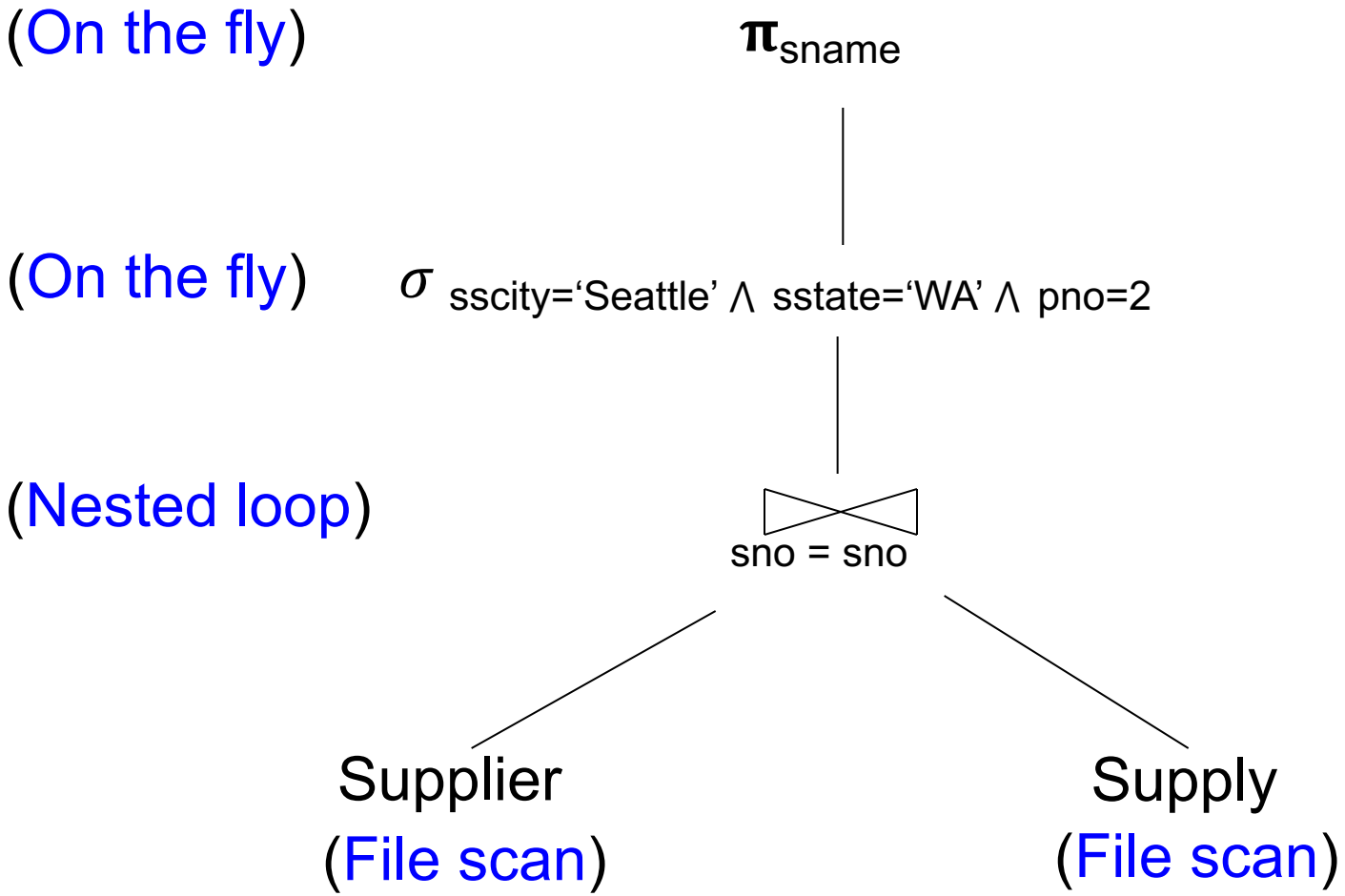


Physical Query Plan

- Logical query plan with extra annotations
- **Access path selection** for each relation
  - Use a file scan or use an index
- **Implementation choice** for each operator
- **Scheduling decisions** for operators

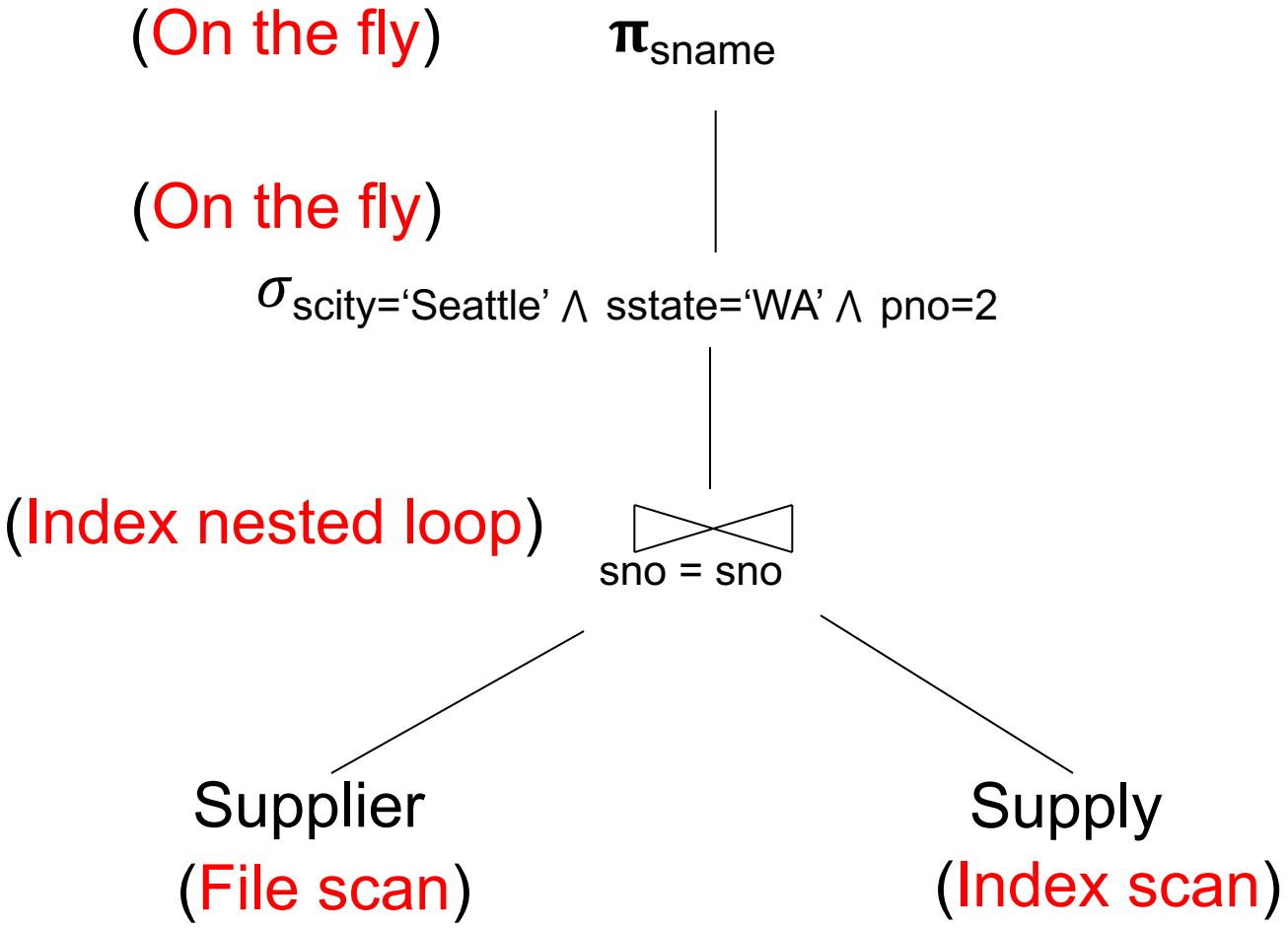
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

Physical Query Plan



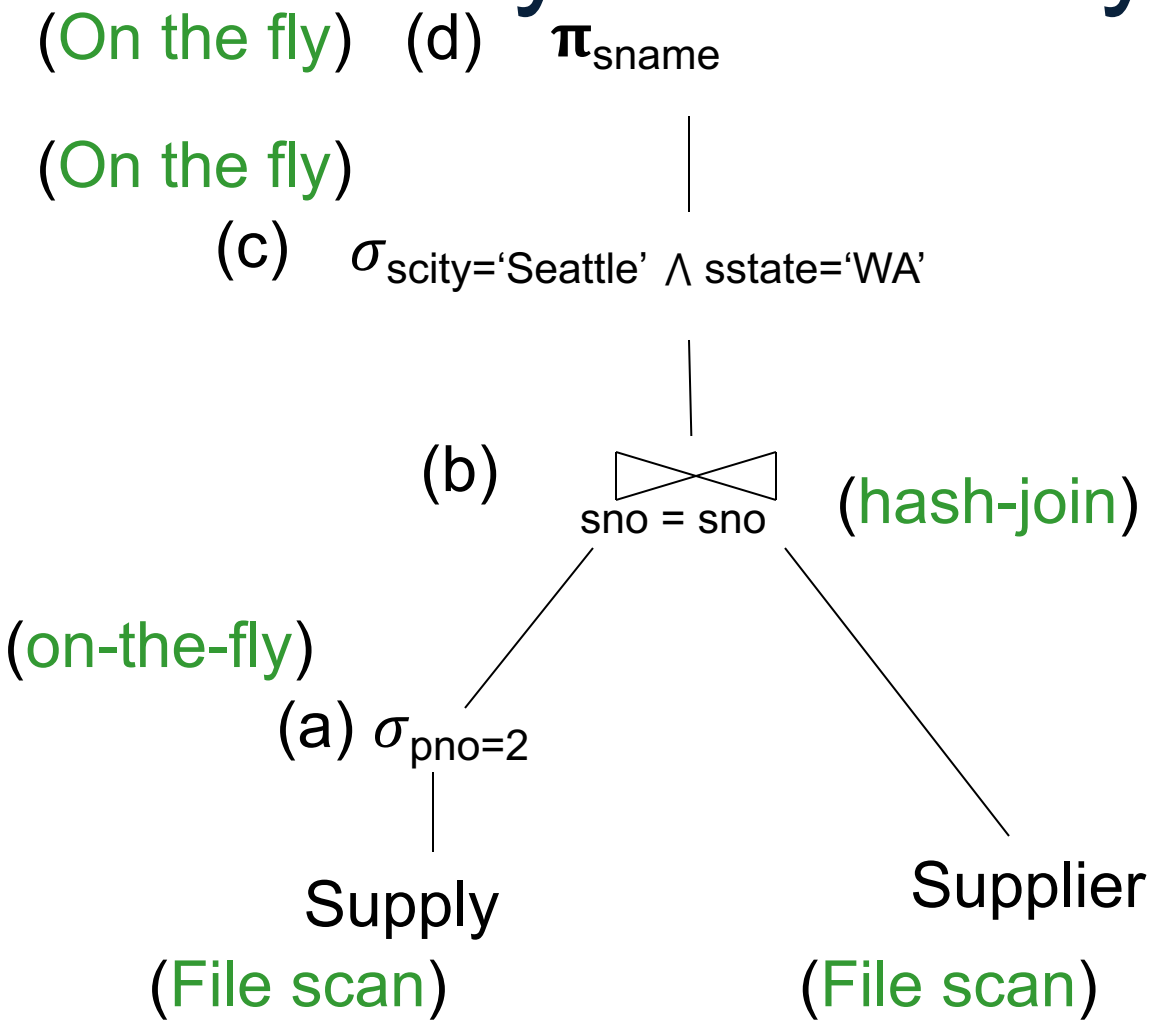
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

Physical Query Plan 2



Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

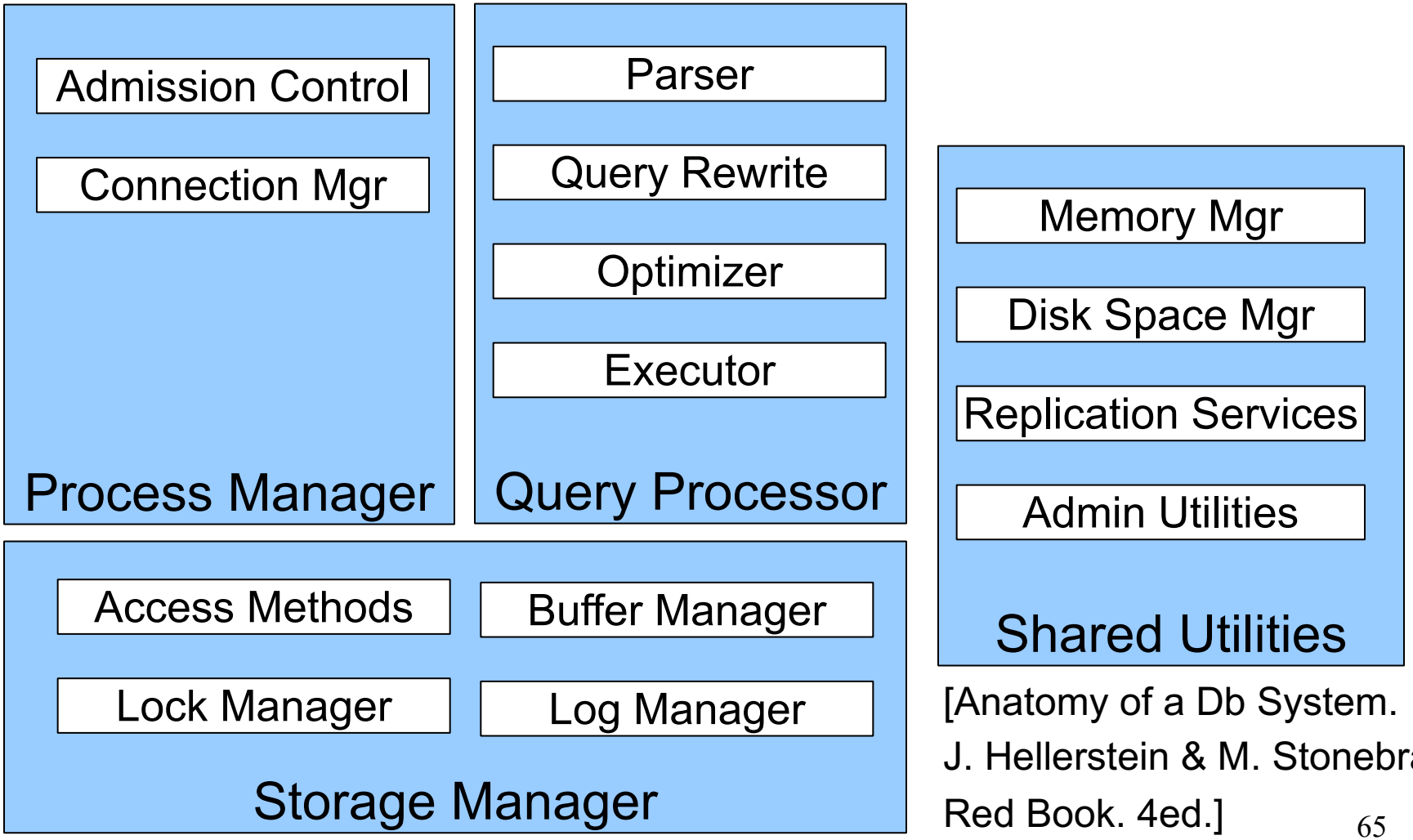
Physical Query Plan 3



Query Plans – Summary

- SQL query → query plan
- Logical query plan = a relational algebra expression
- Physical query plan = choose an implementation for each operator
- The cost of computing the SQL query is affected by both choices: of the logical and the physical plan
- Query optimizer choose optimal plan

DBMS Architecture



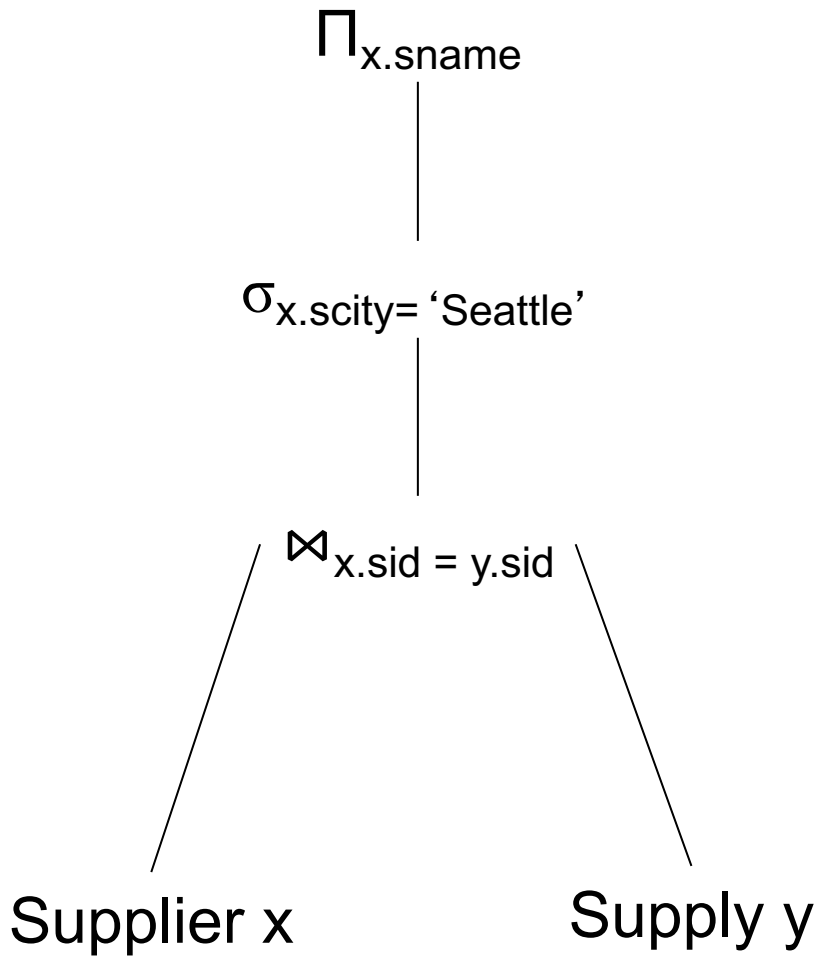
[Anatomy of a Db System. J. Hellerstein & M. Stonebraker. Red Book. 4ed.]

Query Optimization

- Main idea: replace a query plan with another one that is equivalent, but cheaper
- Algebraic identities of the relational algebra
- Will discuss:
  1. Pushing selections down
  2. Join reorder

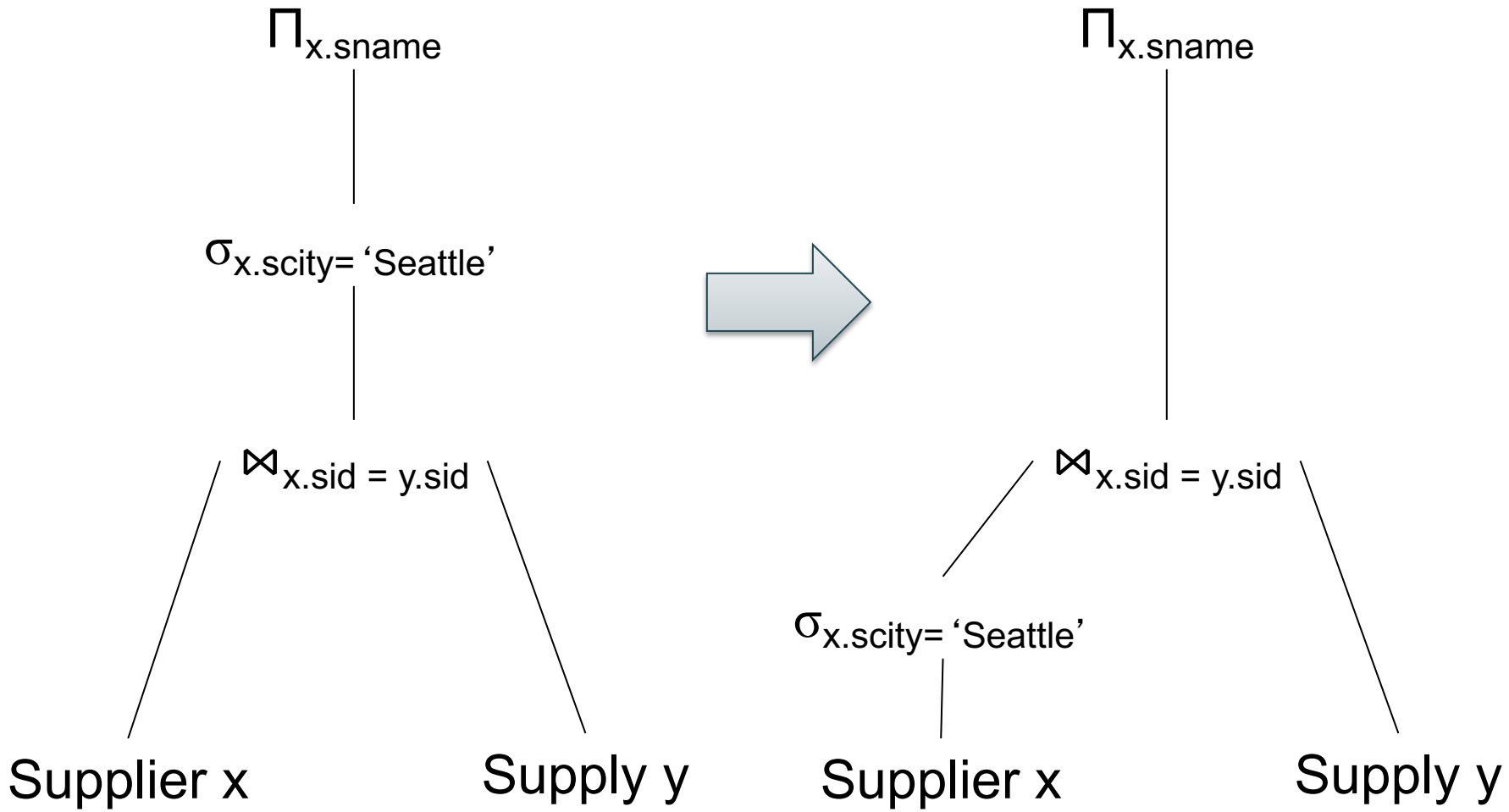
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



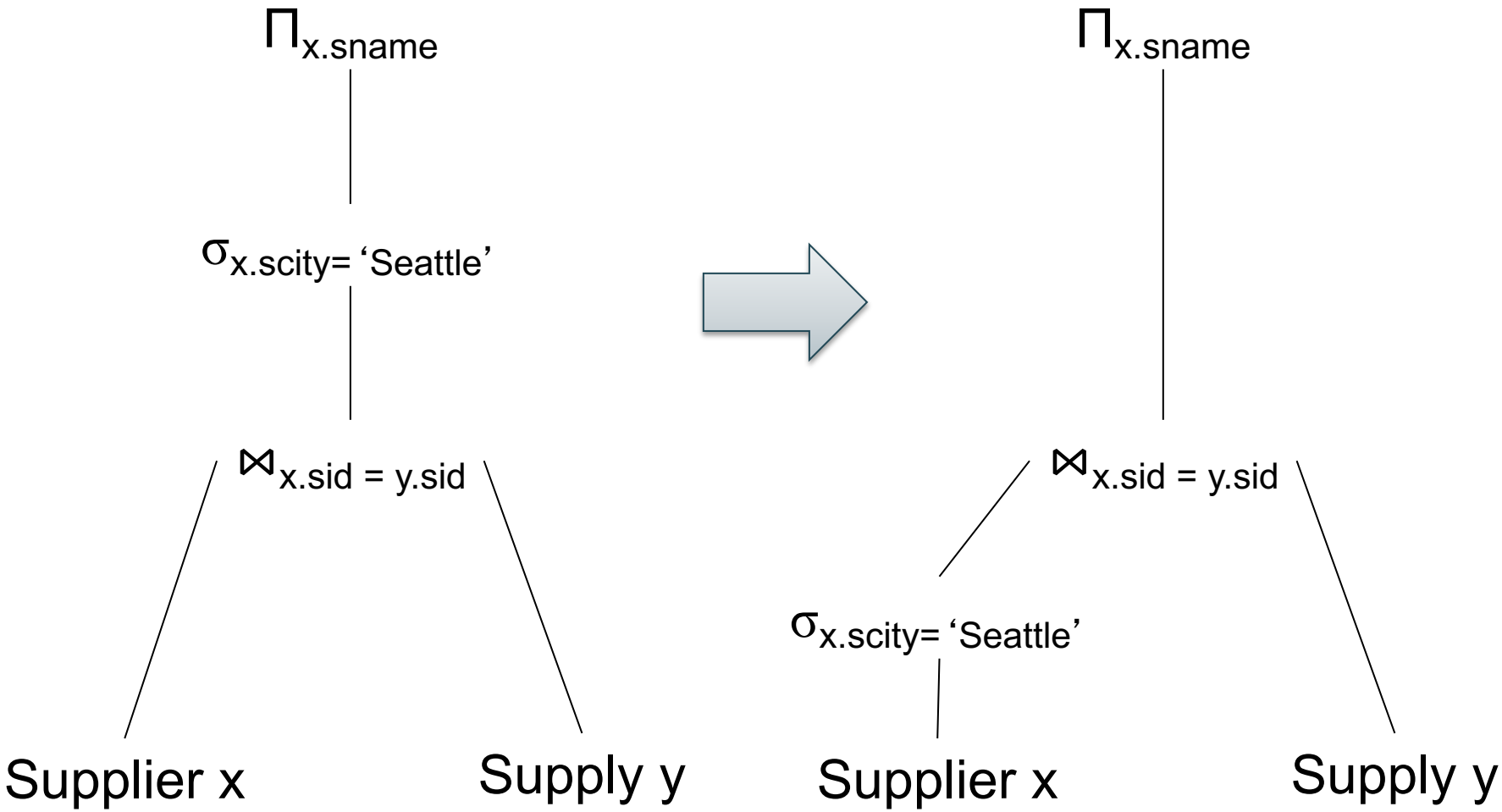
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

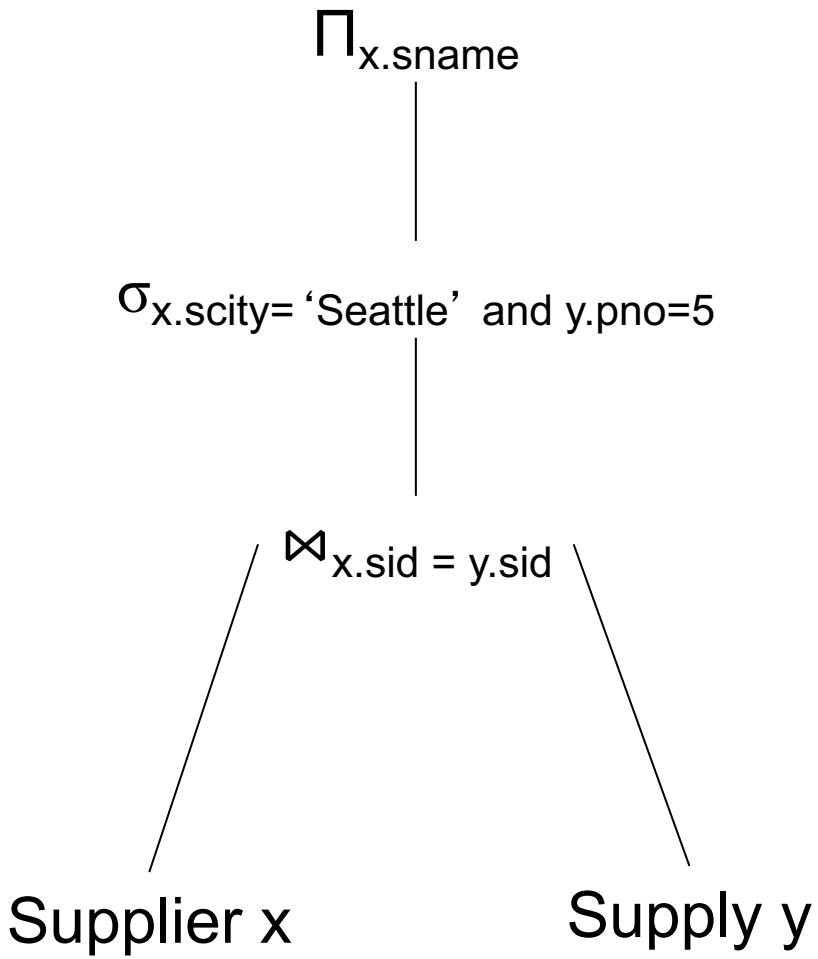
Push Selections Down



$\sigma_C(R \bowtie S) = \sigma_C(R) \bowtie S$ when C refers only to R

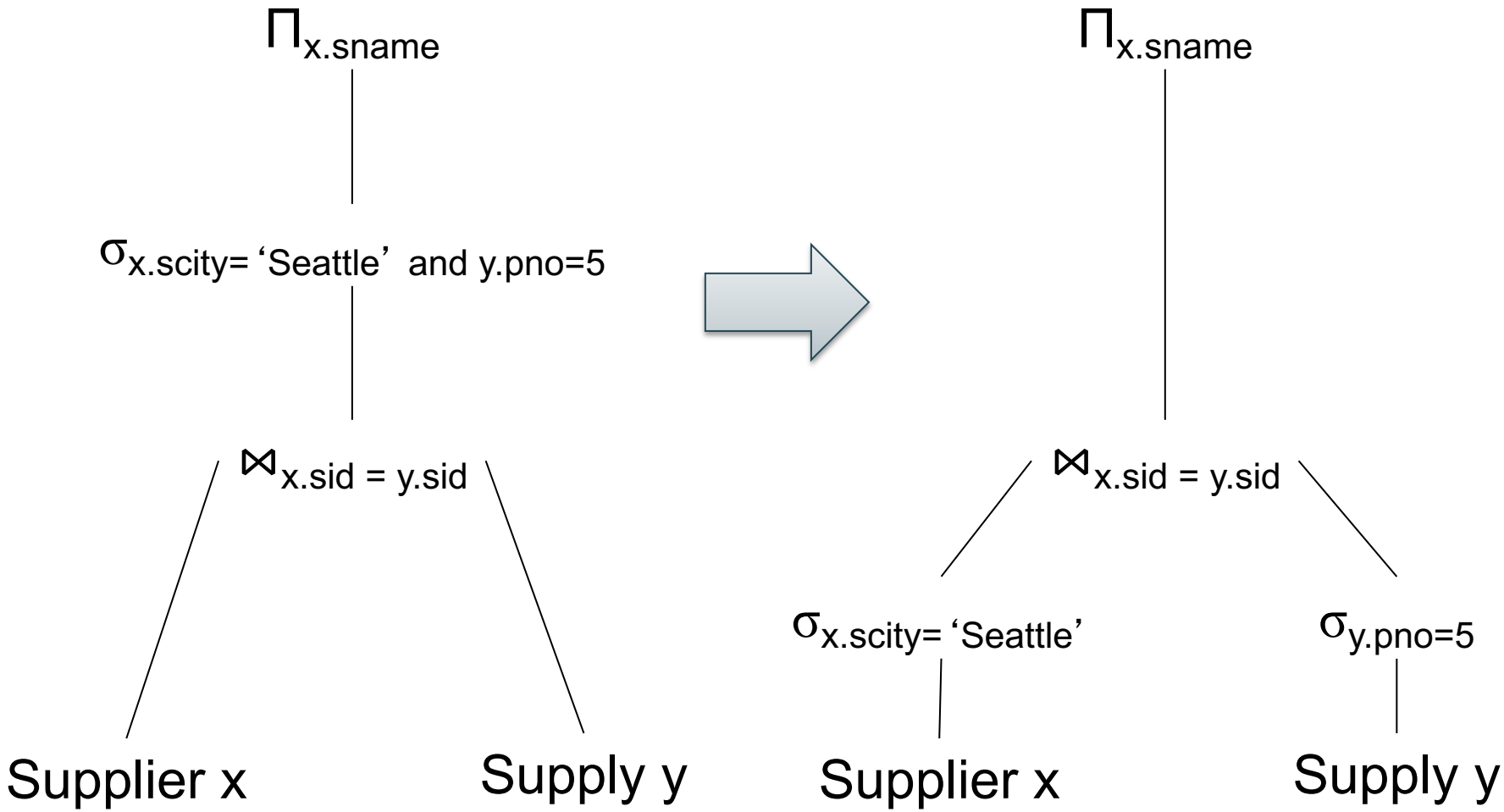
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



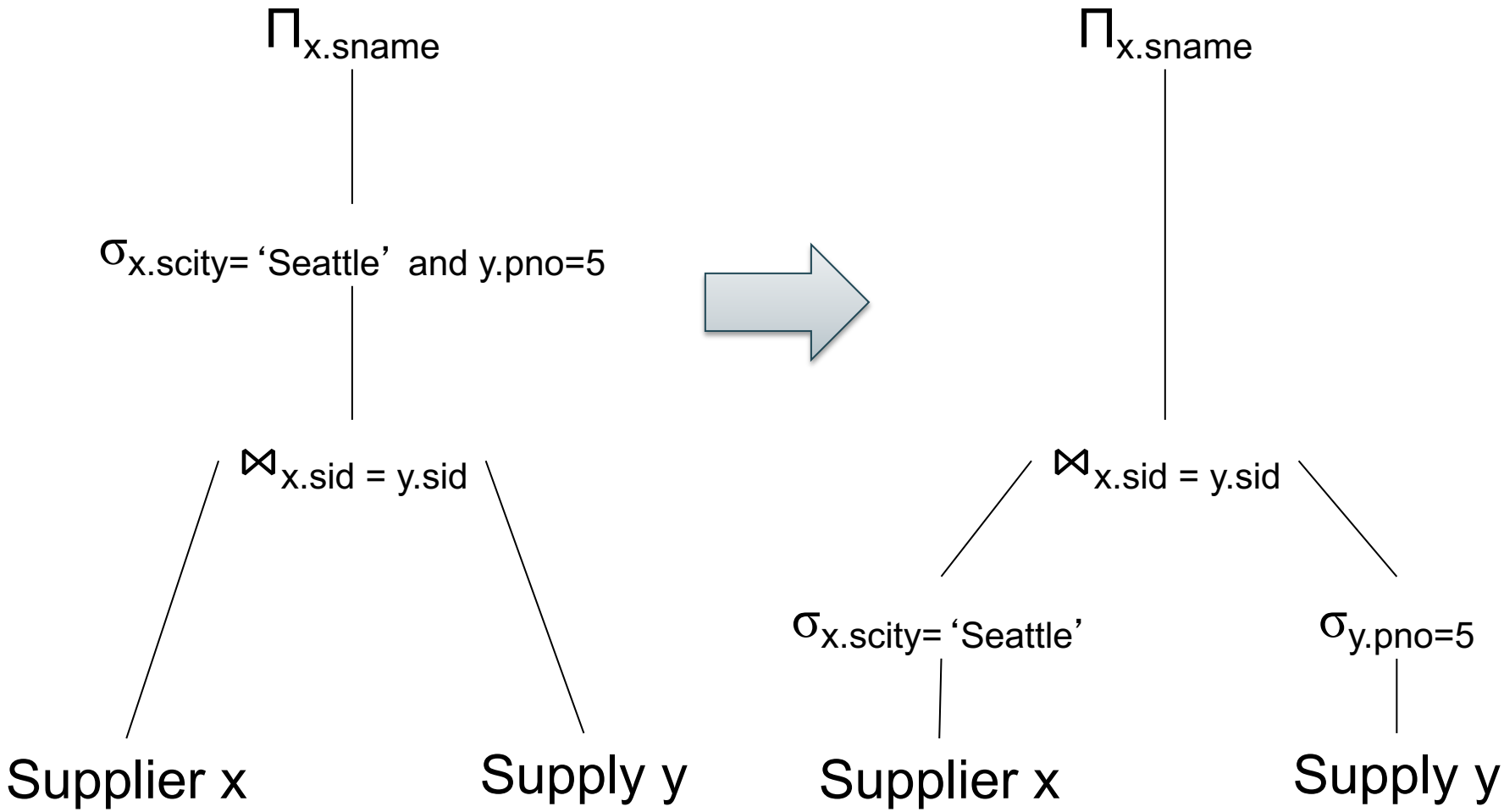
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

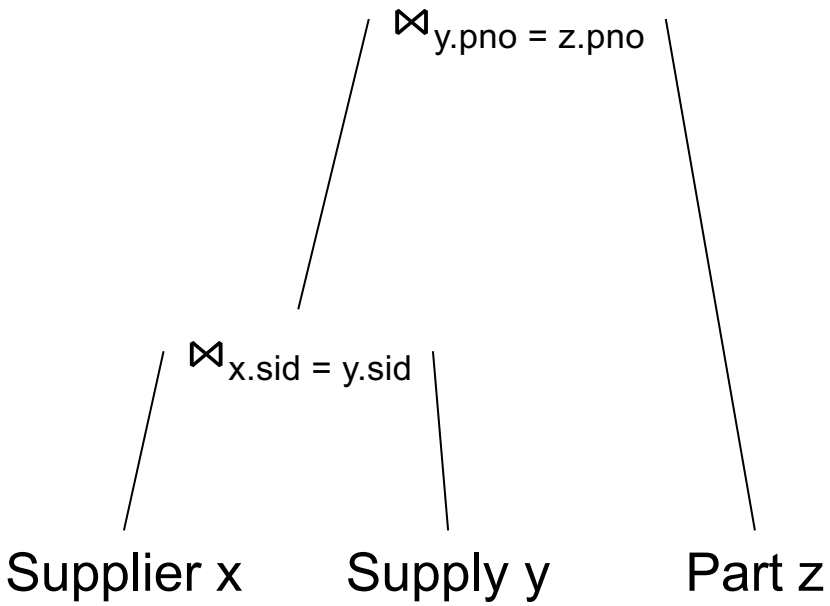
Push Selections Down



$$\sigma_{C1 \text{ and } C2}(R \bowtie S) = \sigma_{C1}(\sigma_{C2}(R \bowtie S)) = \sigma_{C1}(R \bowtie \sigma_{C2}(S)) = \sigma_{C1}(R) \bowtie \sigma_{C2}(S)$$

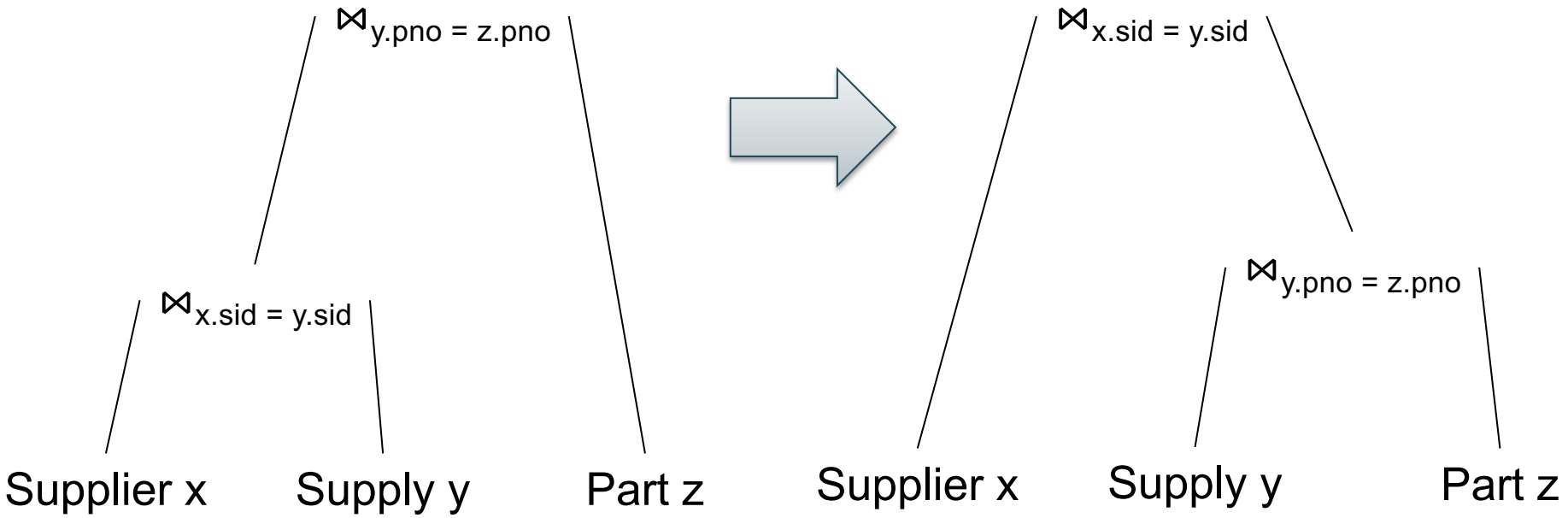
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



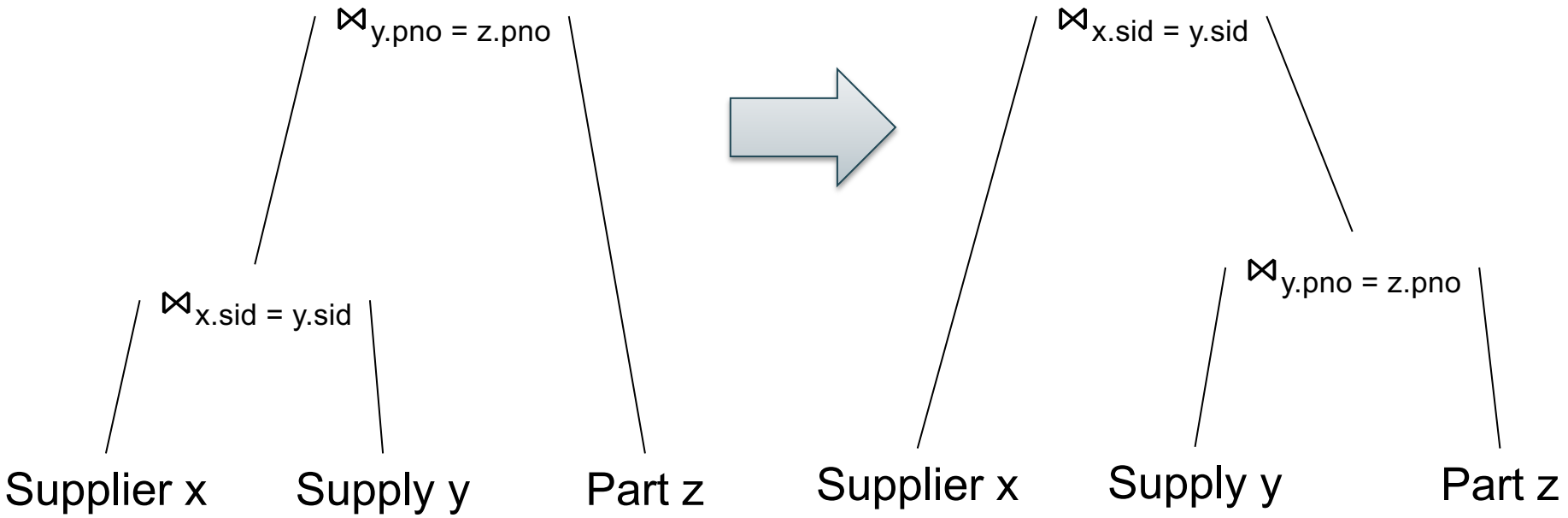
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

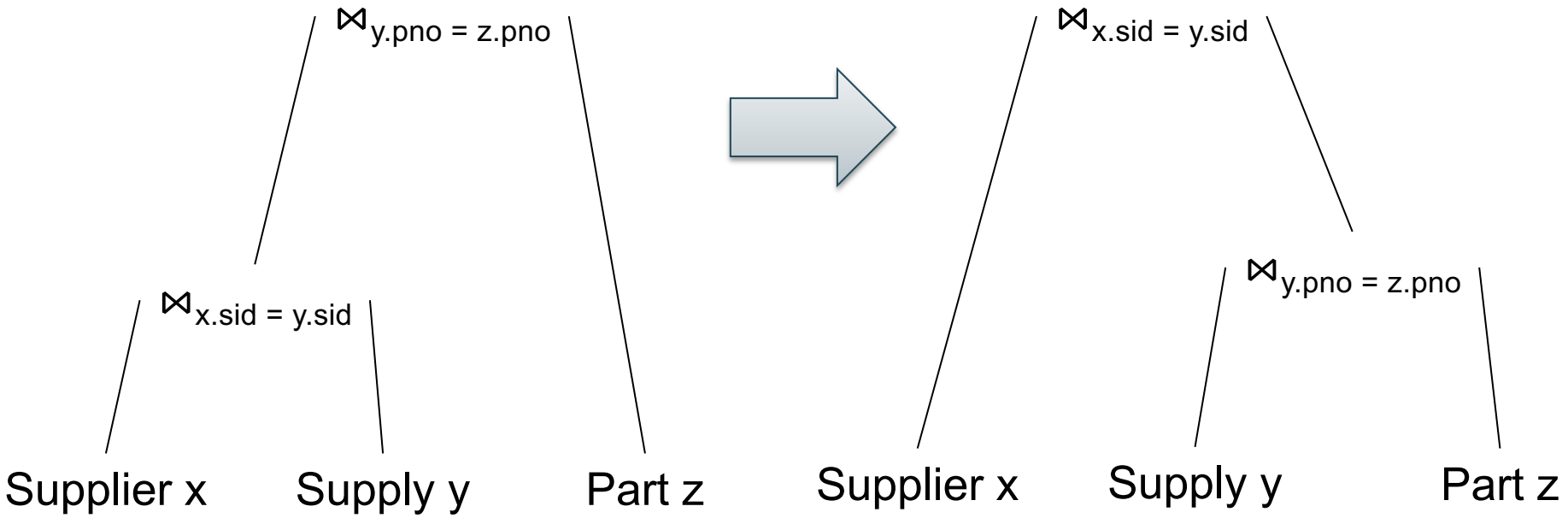
Join Reorder



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

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



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

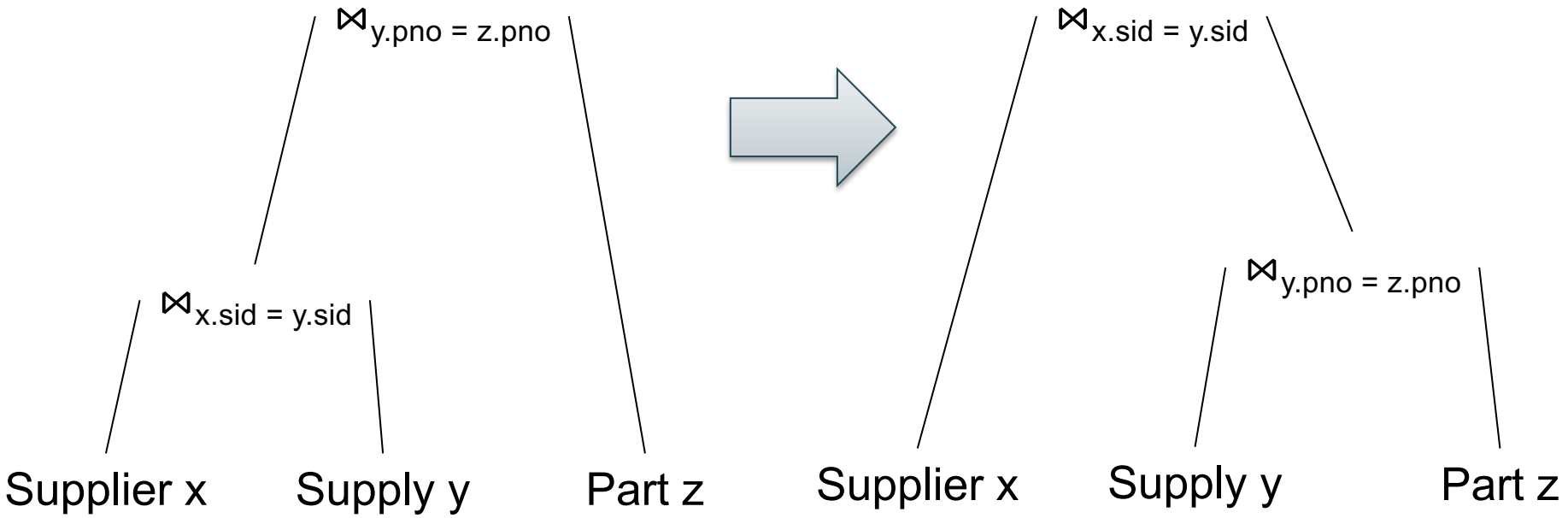
Also:

$$R \bowtie S = S \bowtie R$$

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder

When is one plan better than the other?

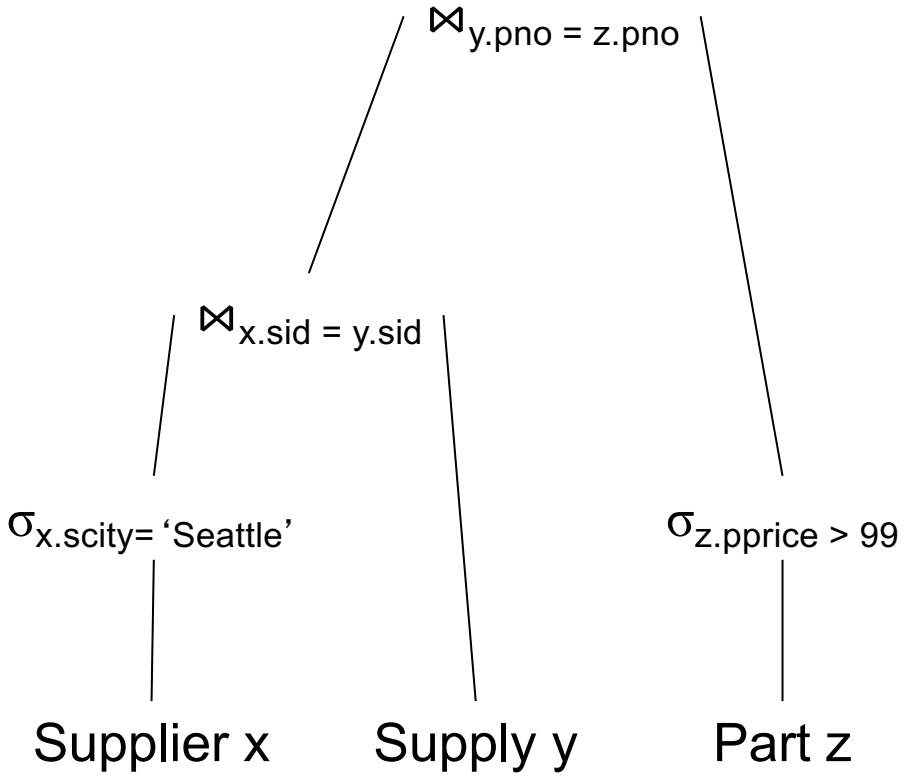


$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

Also: $R \bowtie S = S \bowtie R$

Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

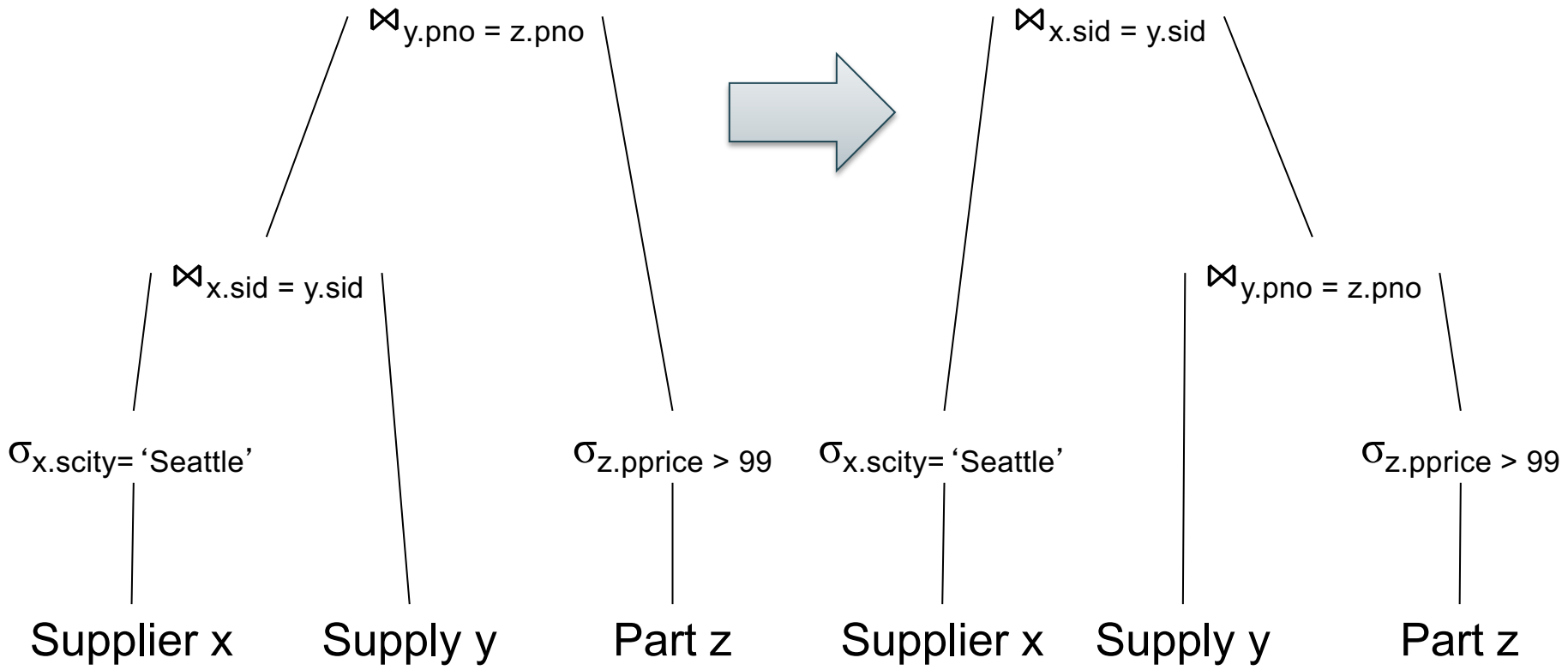
Join Reorder



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder

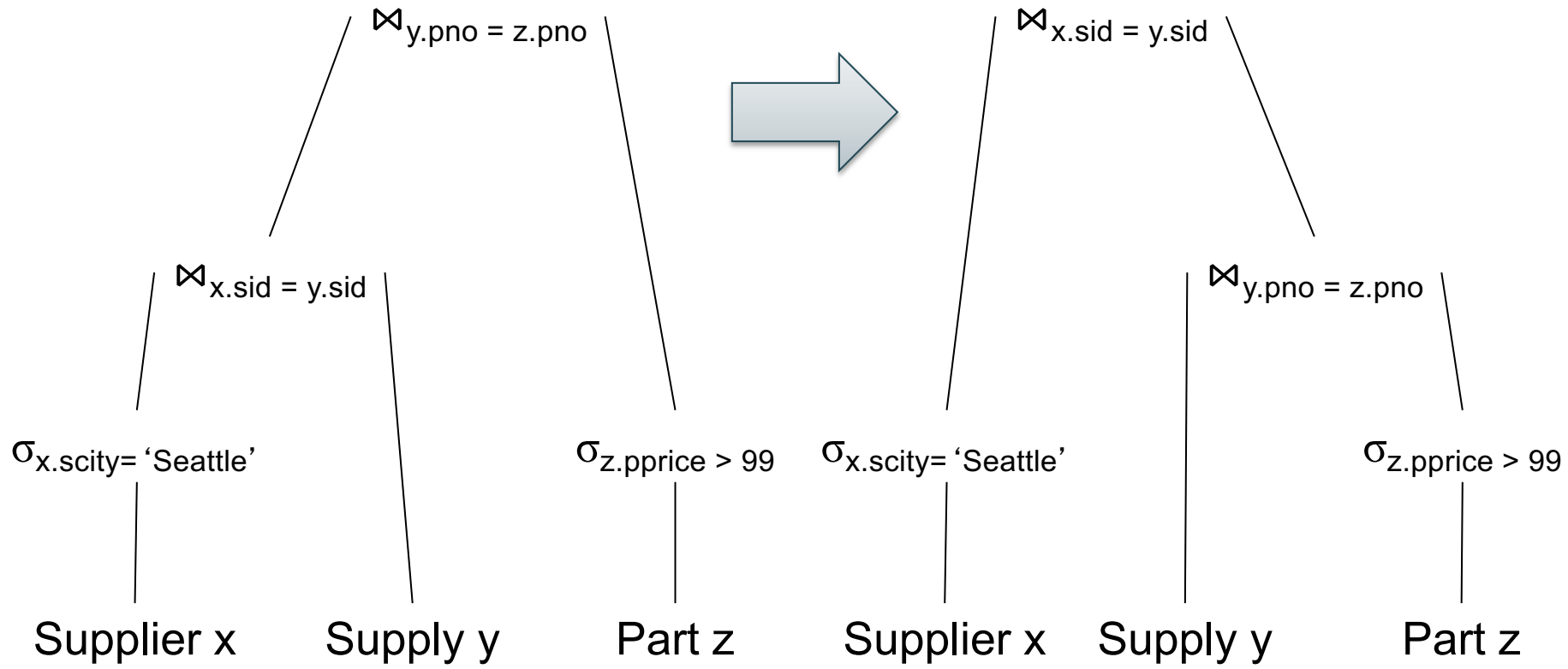
When is one plan better than the other?



Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder

When is one plan better than the other?



Lesson: need sizes of $\sigma_{x.scity= \text{'Seattle'}}$(Supplier), $\sigma_{z.pprice > 99}$(Part)

Query Executor

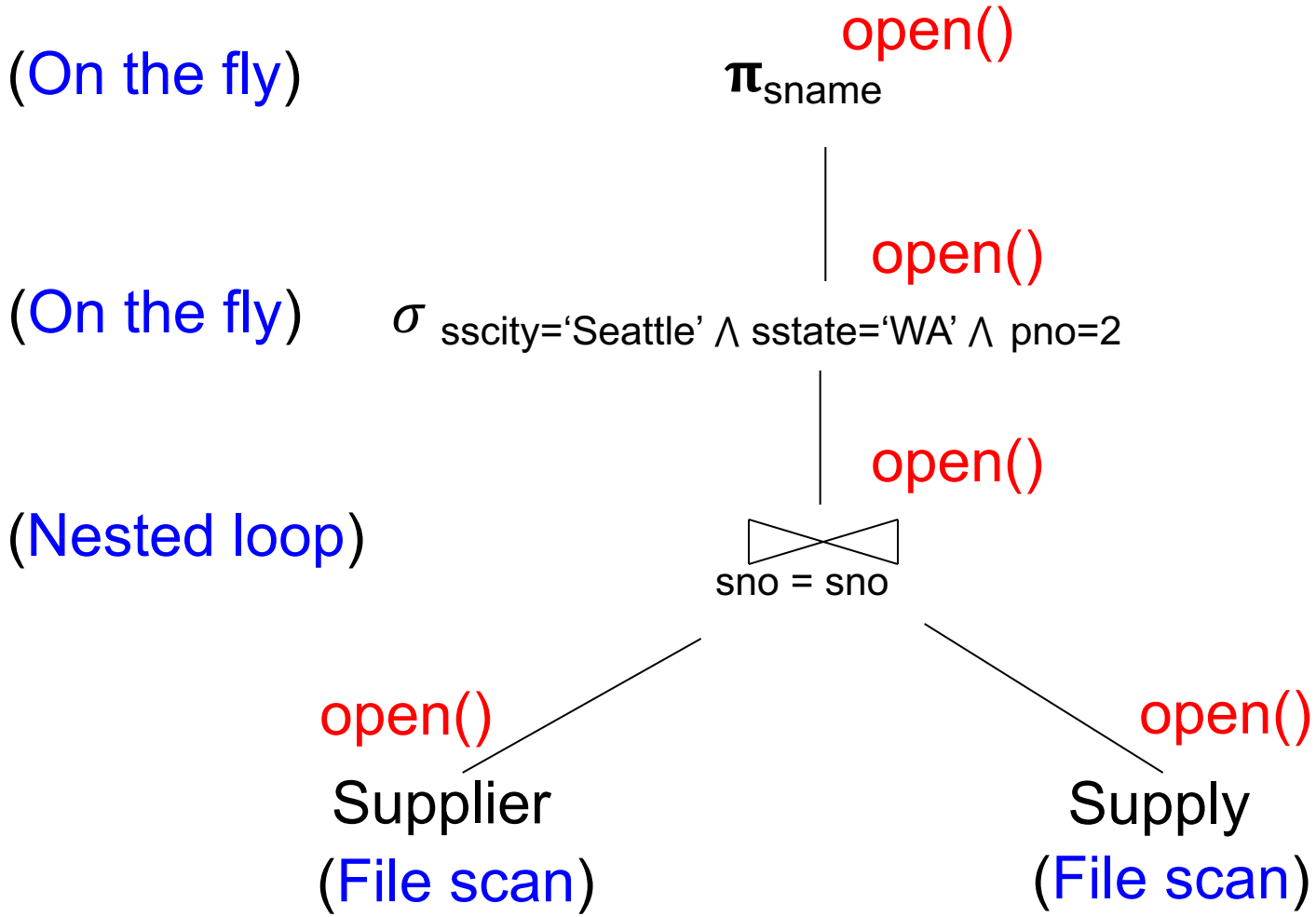
Iterator Interface

- Each **operator implements this interface**

- **open()**
  - Initializes operator state
  - Sets parameters such as selection condition
- **next()**
  - Operator invokes next() recursively on its inputs
  - Performs processing and produces an output tuple
- **close()**: clean-up state

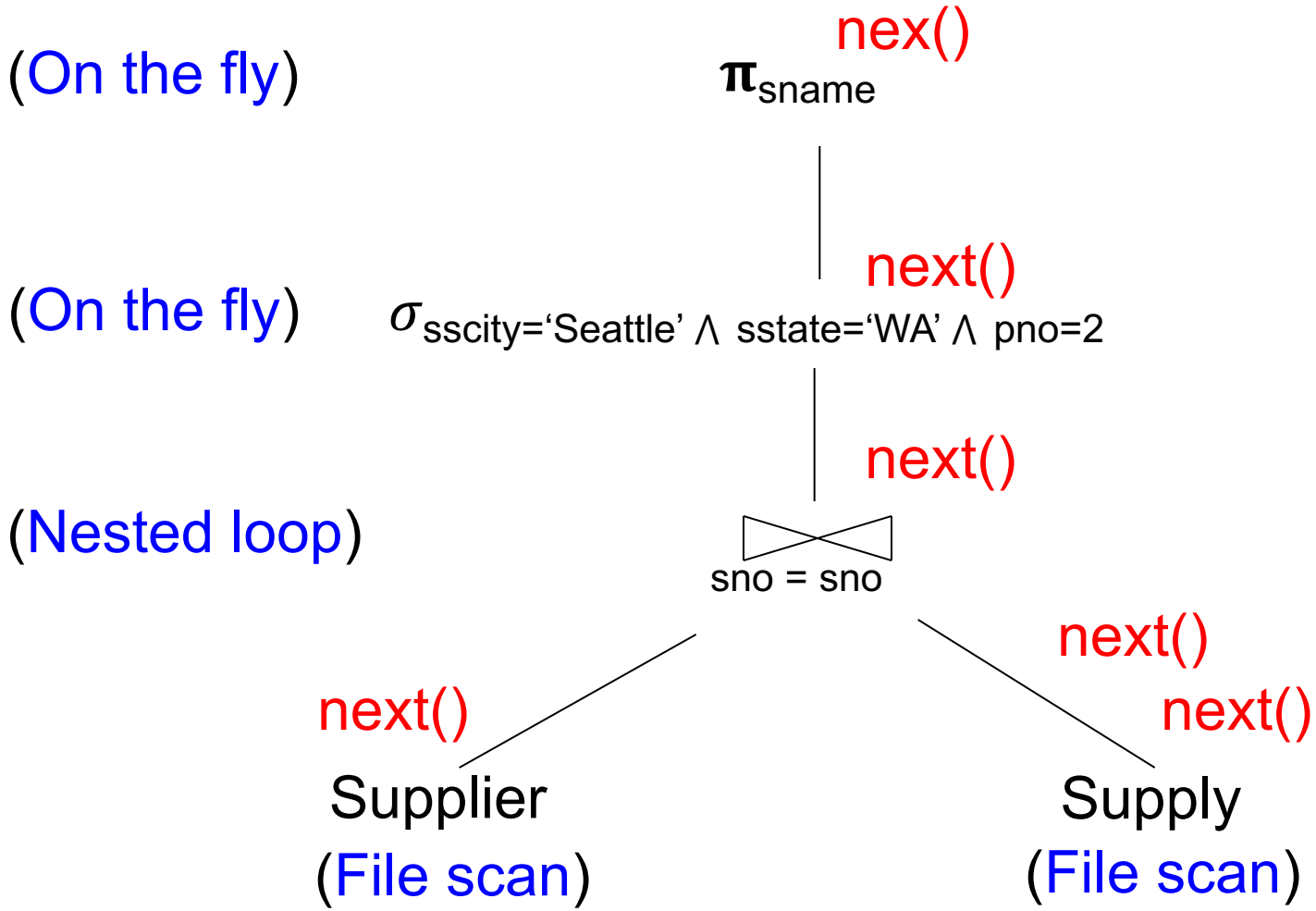
Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

Query Execution

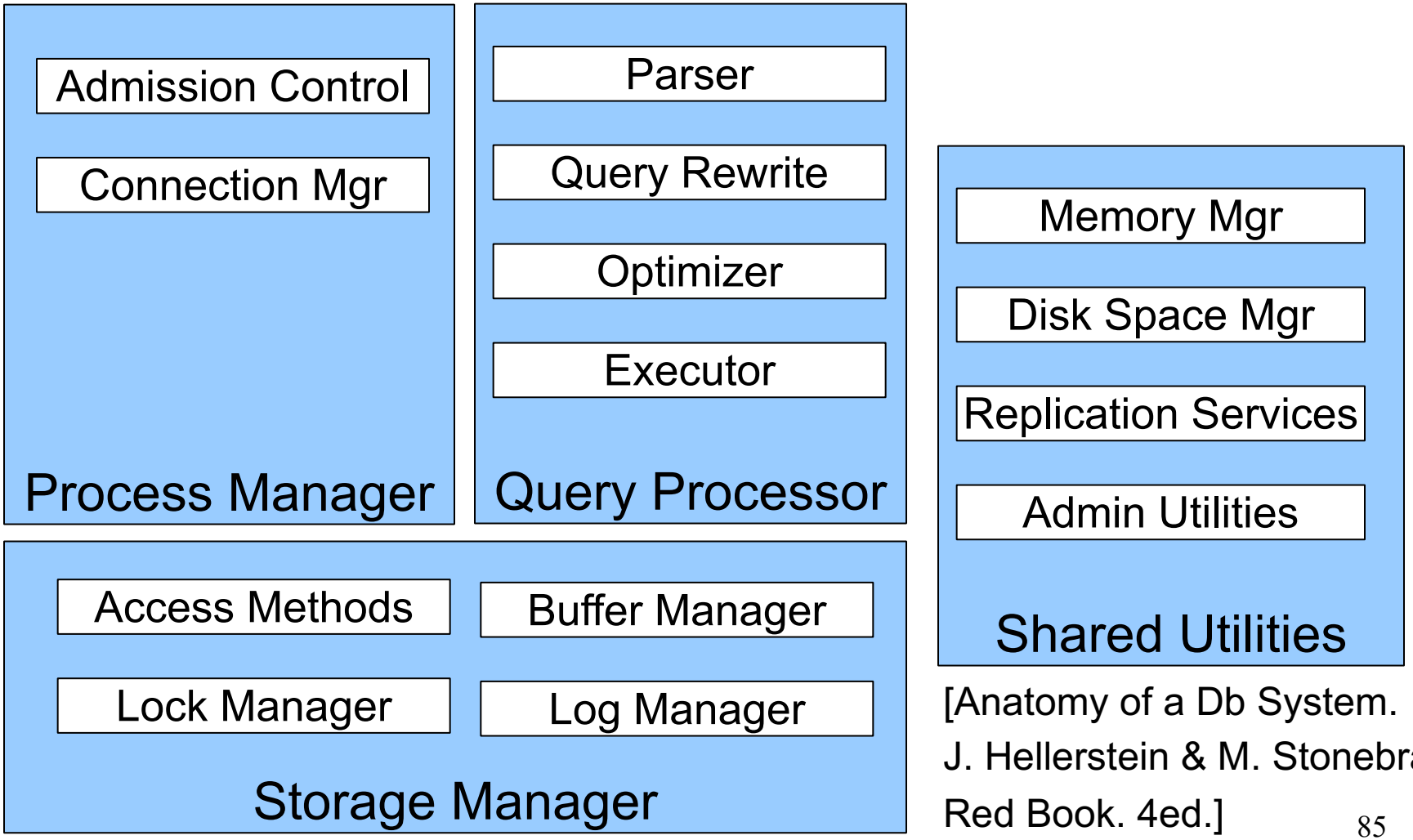


Supplier(sno,sname,scity,sstate)
Part(pno,pname,psize,pcolor)
Supply(sno,pno,price)

Query Execution



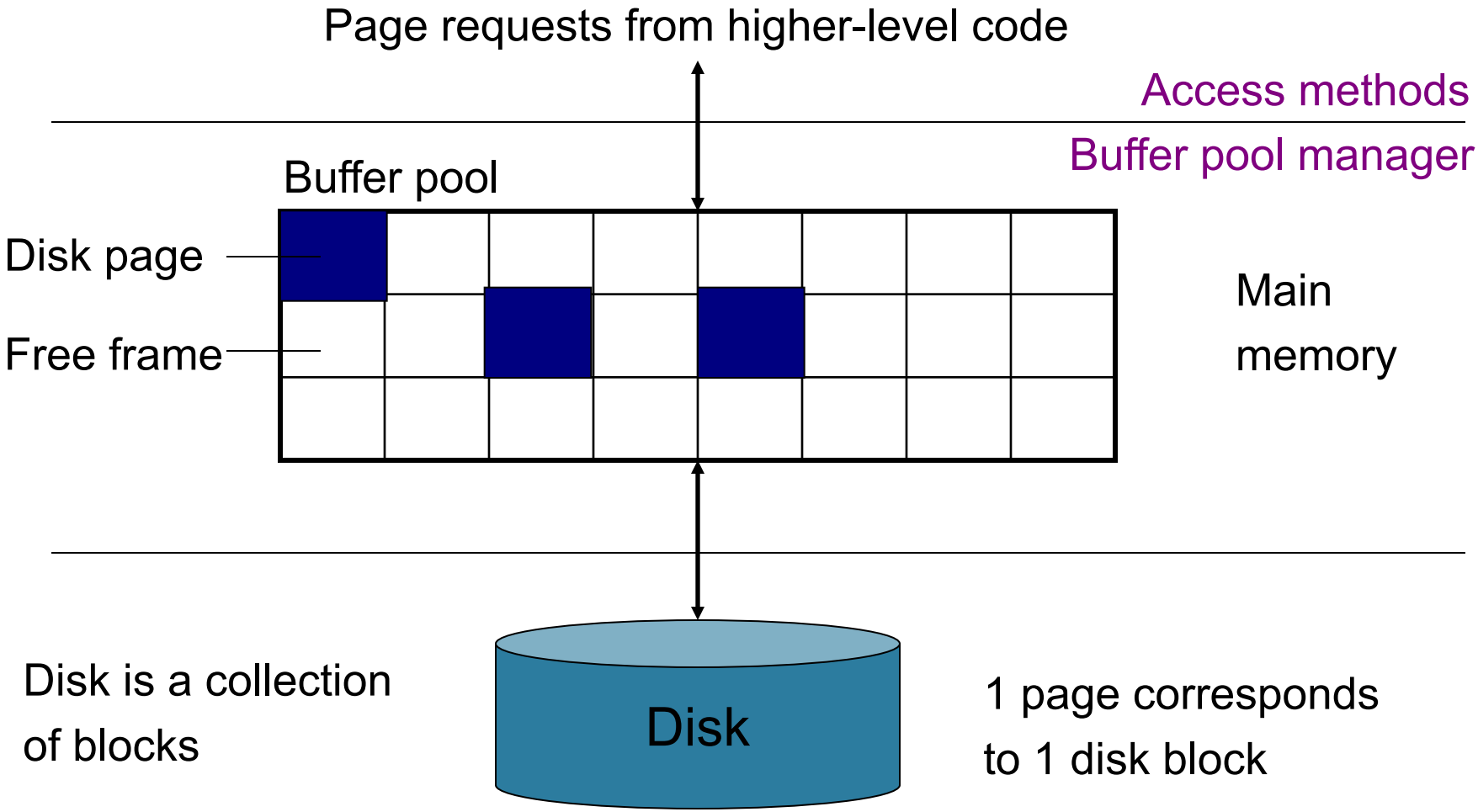
DBMS Architecture



[Anatomy of a Db System. J. Hellerstein & M. Stonebraker. Red Book. 4ed.]

Storage Manager

Buffer Manager



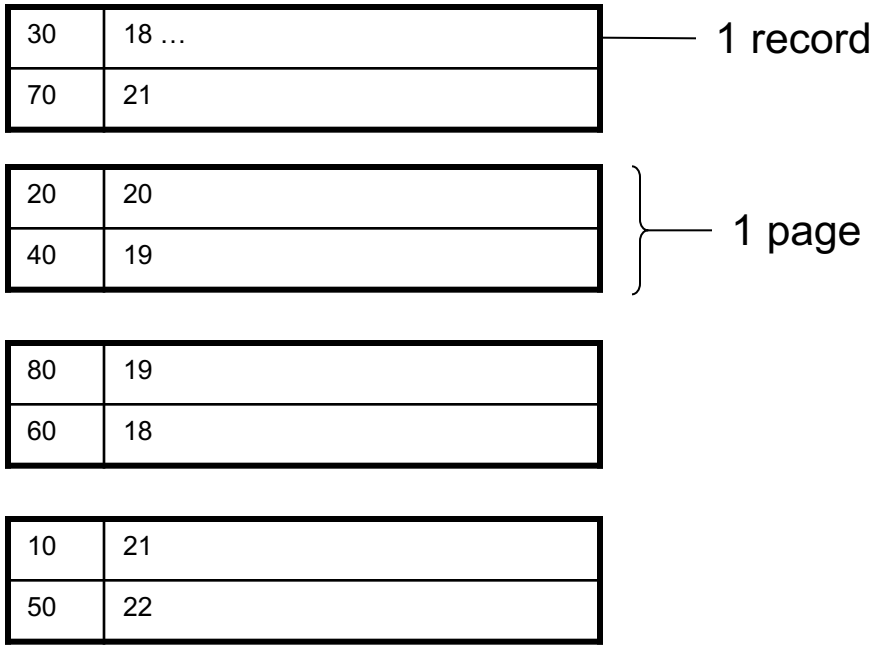
Heap File

Data on disk is stored in *files*

Files consist of *pages* filled with *records*

A *heap file* is not sorted on any attribute

`Student(sid: int, age: int, …)`

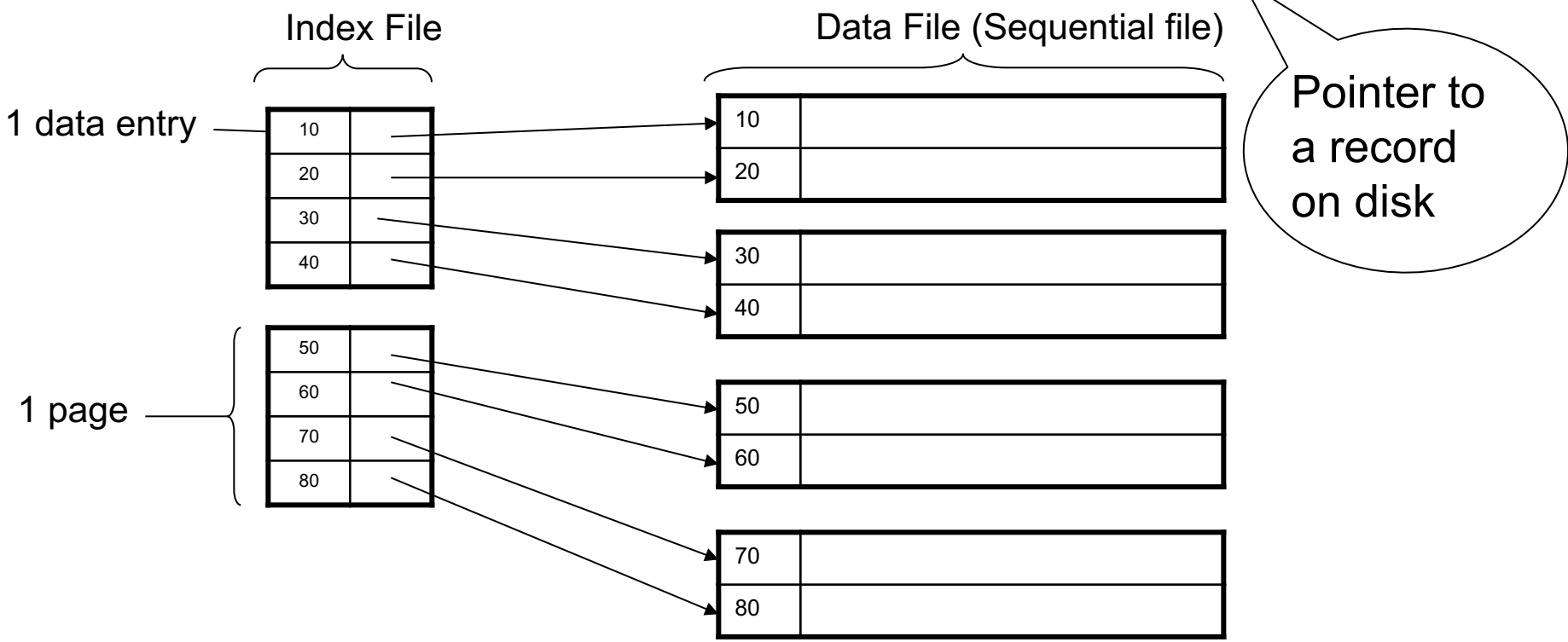


Index

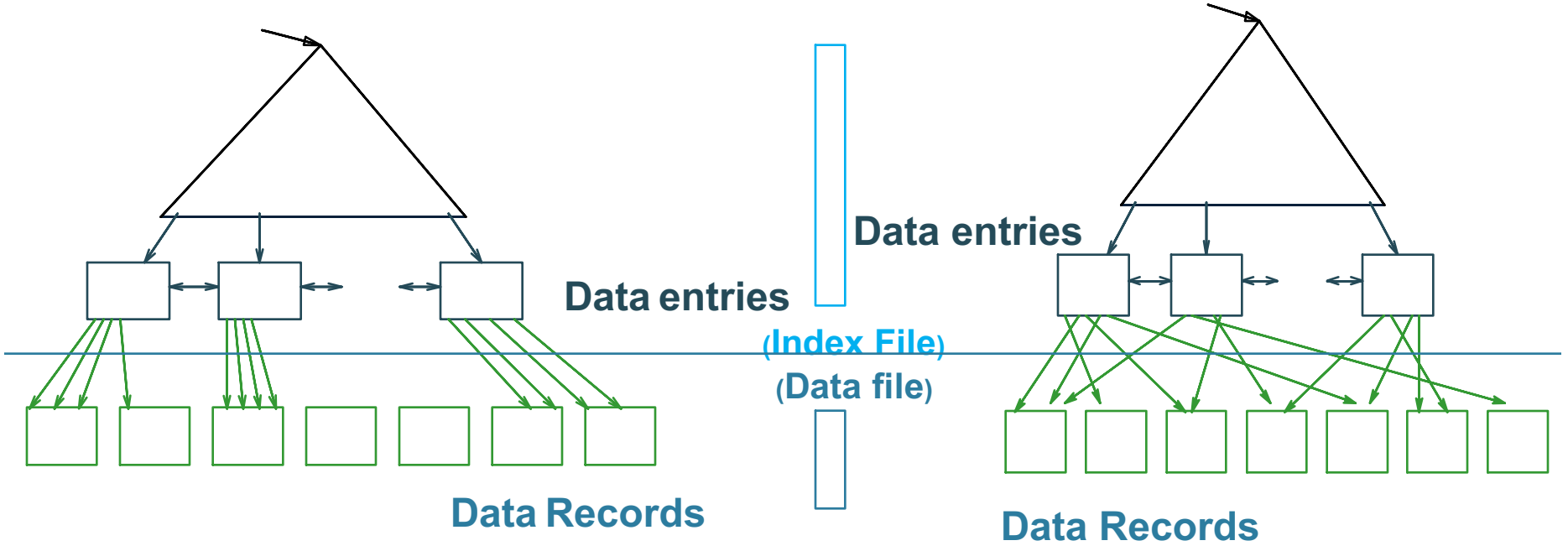
An index is a data structure stored on disk

It is stored in a file consisting of pages & records

But records are (search key value, record ID)



Clustered vs. Unclustered Index



CLUSTERED

UNCLUSTERED

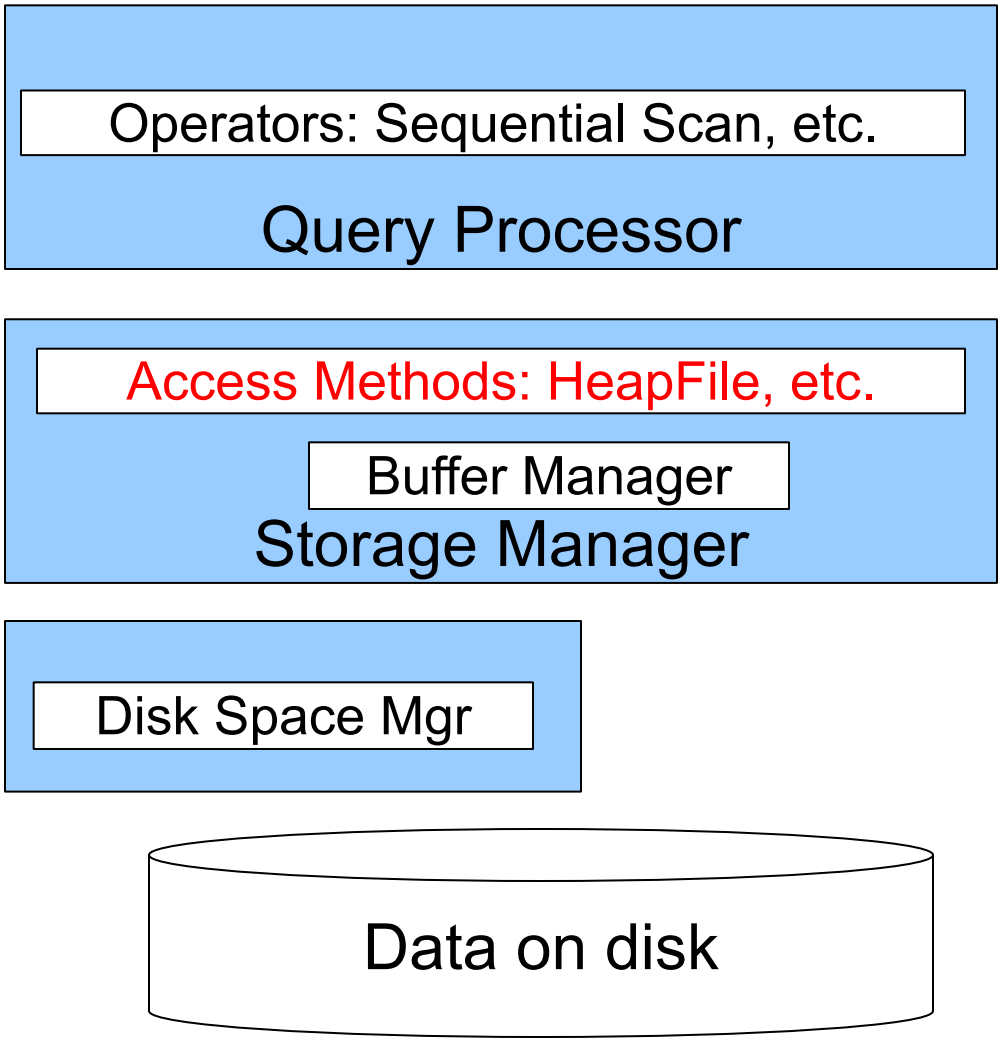
Clustered = records close in index are close in data

Access Method API

Access methods manage access to these disk-based data structures (heap files & indexes)

- **Create** or **destroy** a file
- **Insert** a record
- **Delete** a record with a given rid (rid)
  - rid: unique tuple identifier (more later)
- **Get** a record with a given rid
  - Not necessary for sequential scan operator
  - But used with indexes
- **Scan** all records in the file [matching a predicate]

How it Fits Together



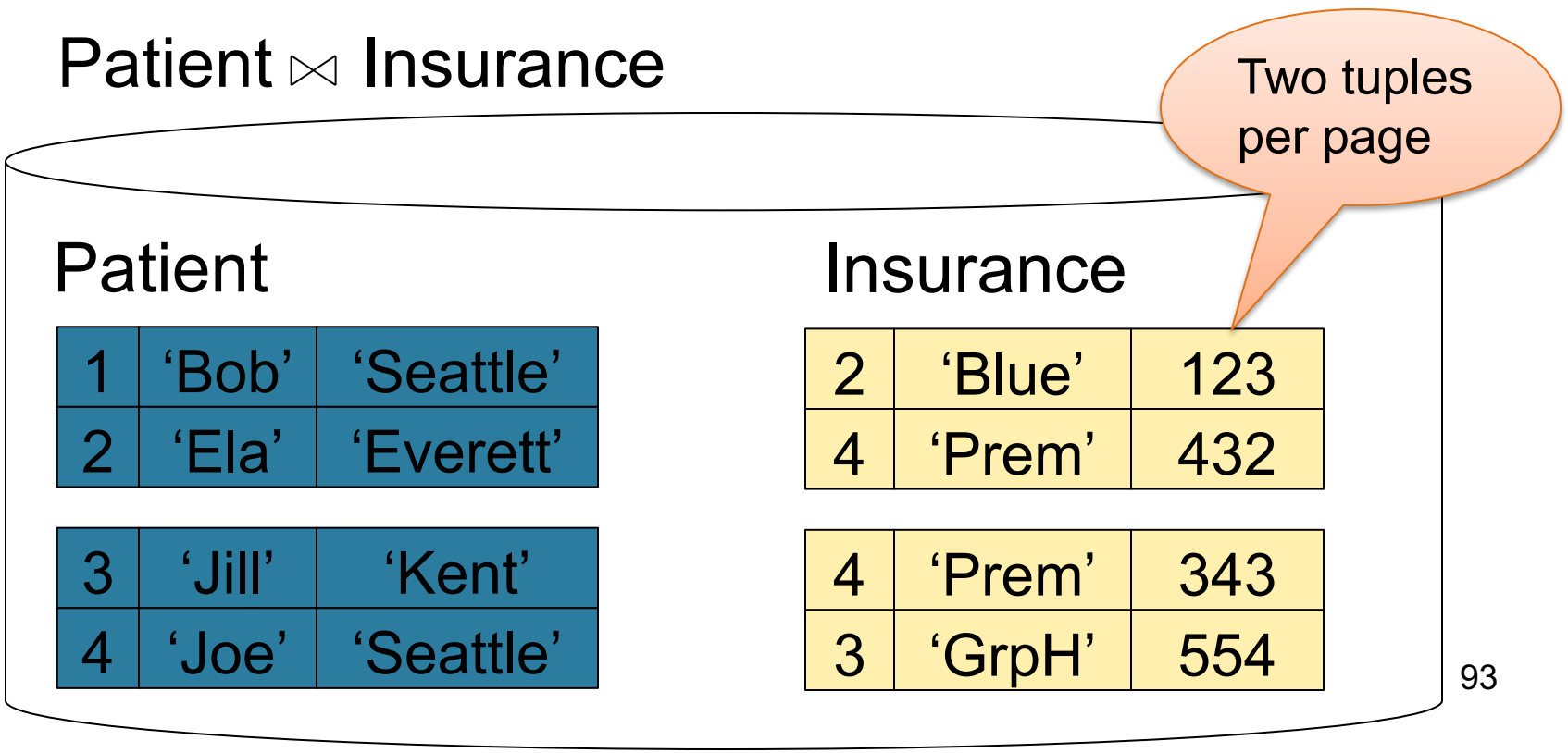
- **Operators:** Process data
- **Access methods**: Organize data to support fast access to desired subsets of records
- **Buffer manager**: Caches data in memory. Reads/writes data to/from disk as needed
- **Disk-space manager**: Allocates space on disk for files/access methods

Hash Join Example

Patient(pid, name, address)

Insurance(pid, provider, policy_nb)

Patient ⋈ Insurance



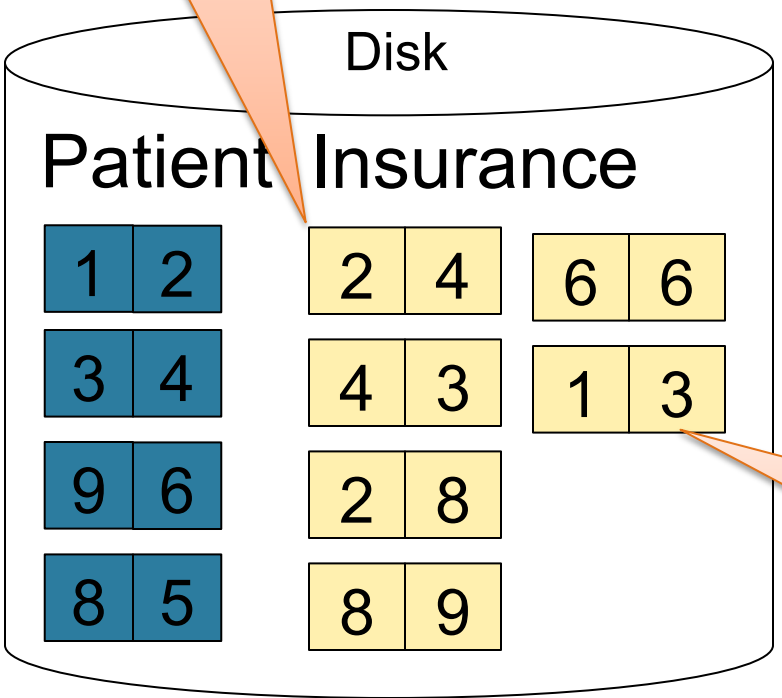
Hash Join Example

Patient ⋈ Insurance

Some large-enough nb

Memory M = 21 pages

Showing pid only

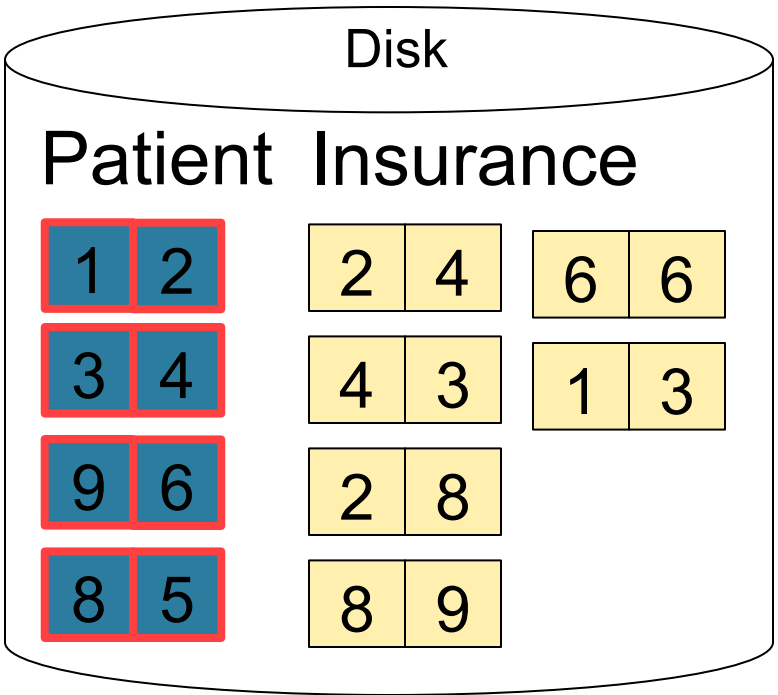


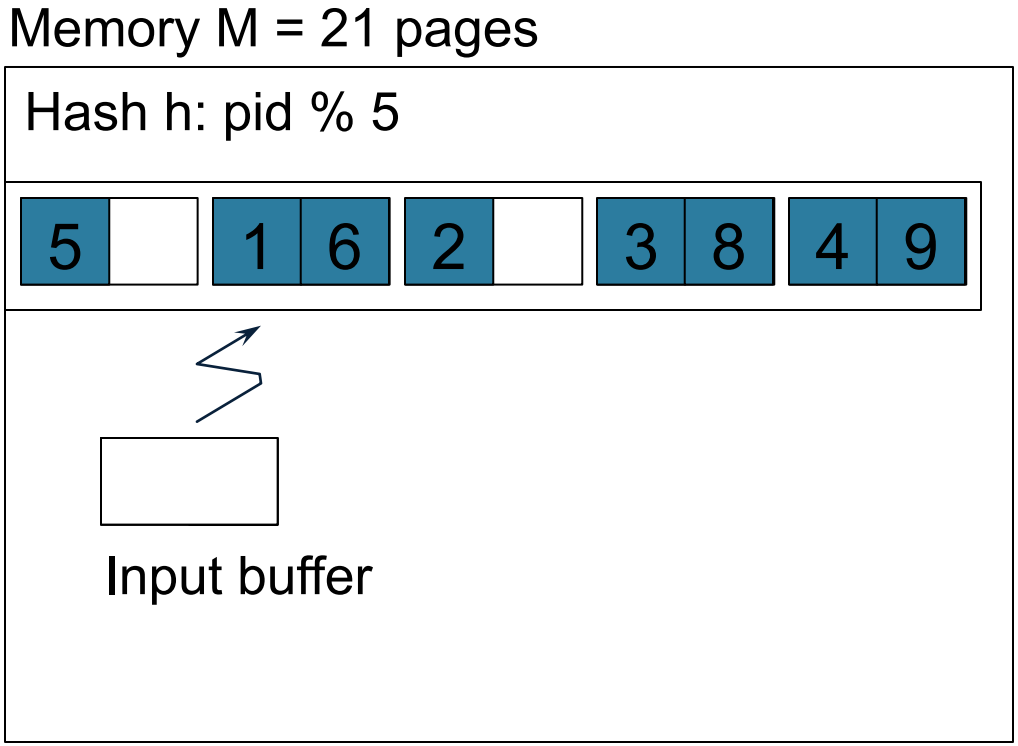
This is one page with two tuples

Hash Join Example

Step 1: Scan Patient and build hash table in memory

Can be done in method open()



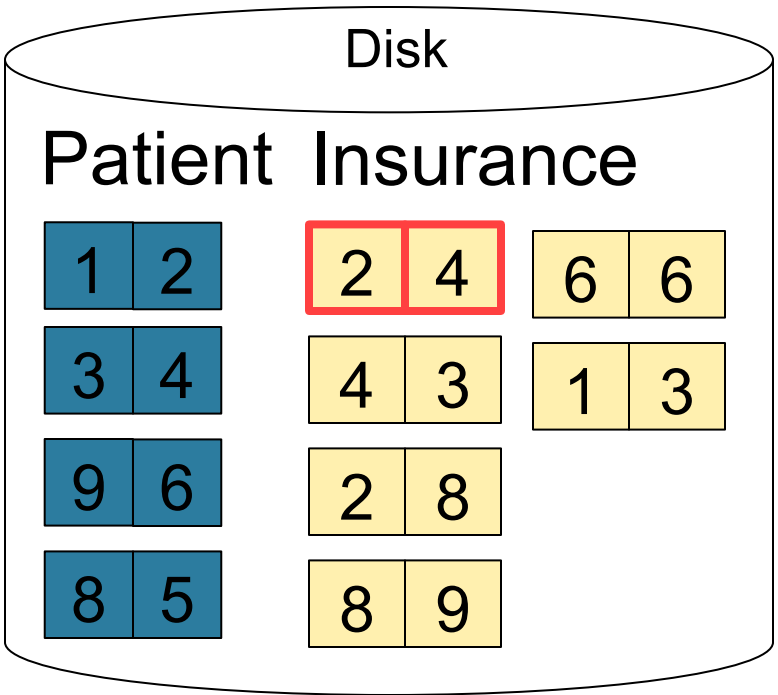


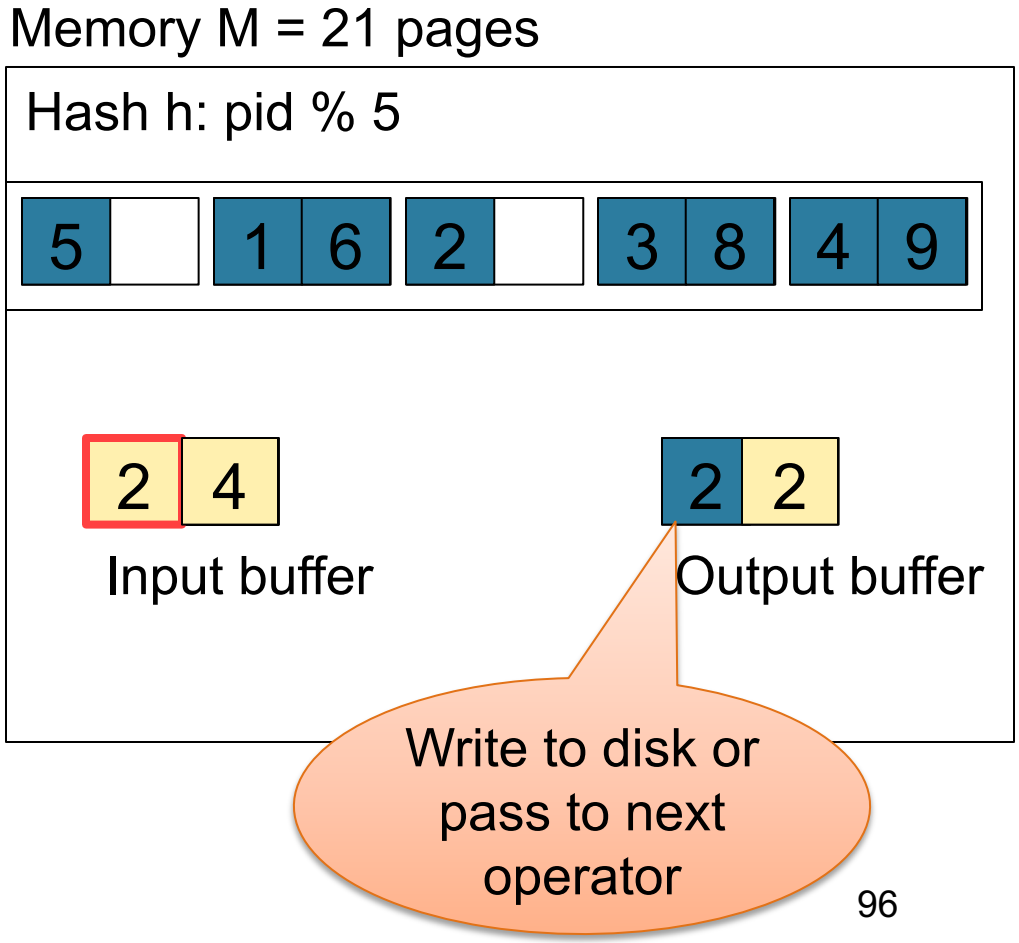
Hash Join Example

Step 2: Scan Insurance and probe into hash table

Done during

calls to next()



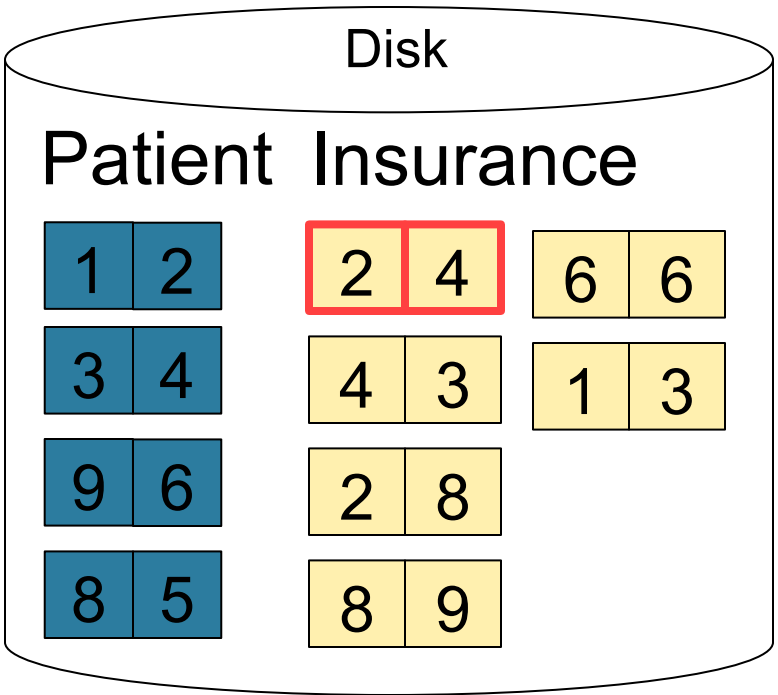


Hash Join Example

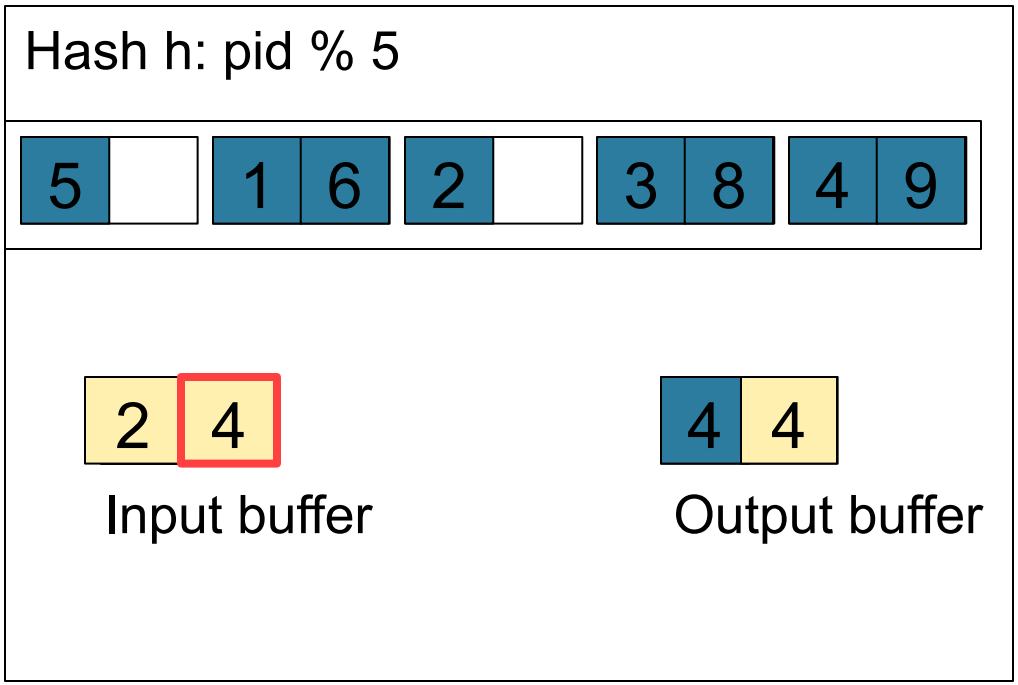
Step 2: Scan Insurance and probe into hash table

Done during

calls to next()



Memory M = 21 pages

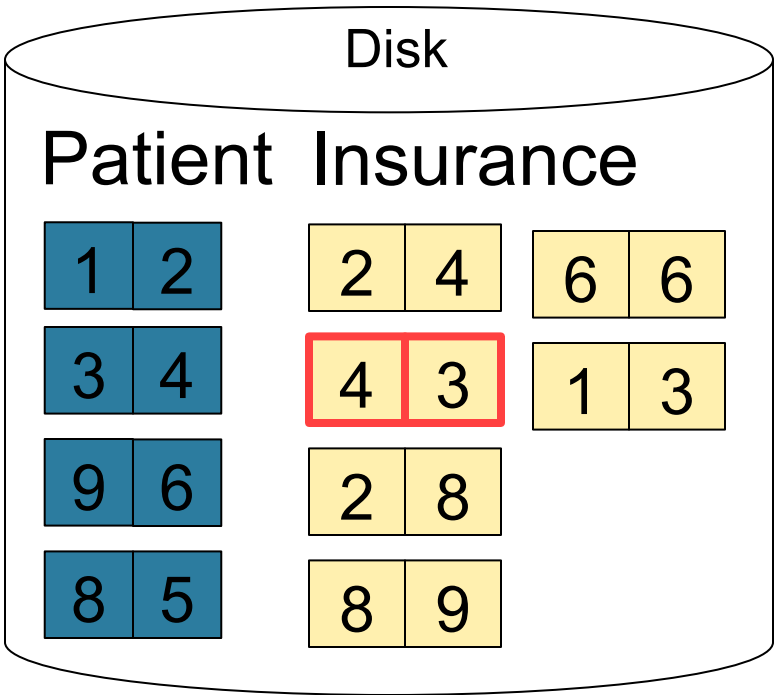


Hash Join Example

Step 2: Scan Insurance and probe into hash table

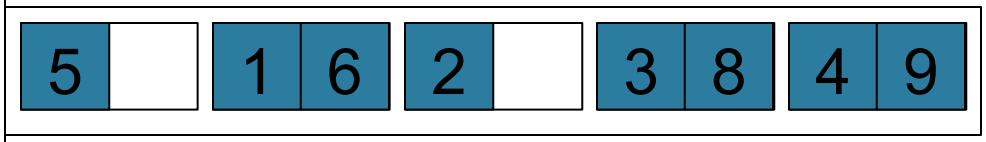
Done during

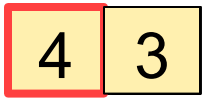
calls to next()



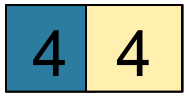
Memory M = 21 pages

Hash h: pid % 5





Input buffer



Output buffer

Keep going until read all of Insurance

Nested Loop Joins

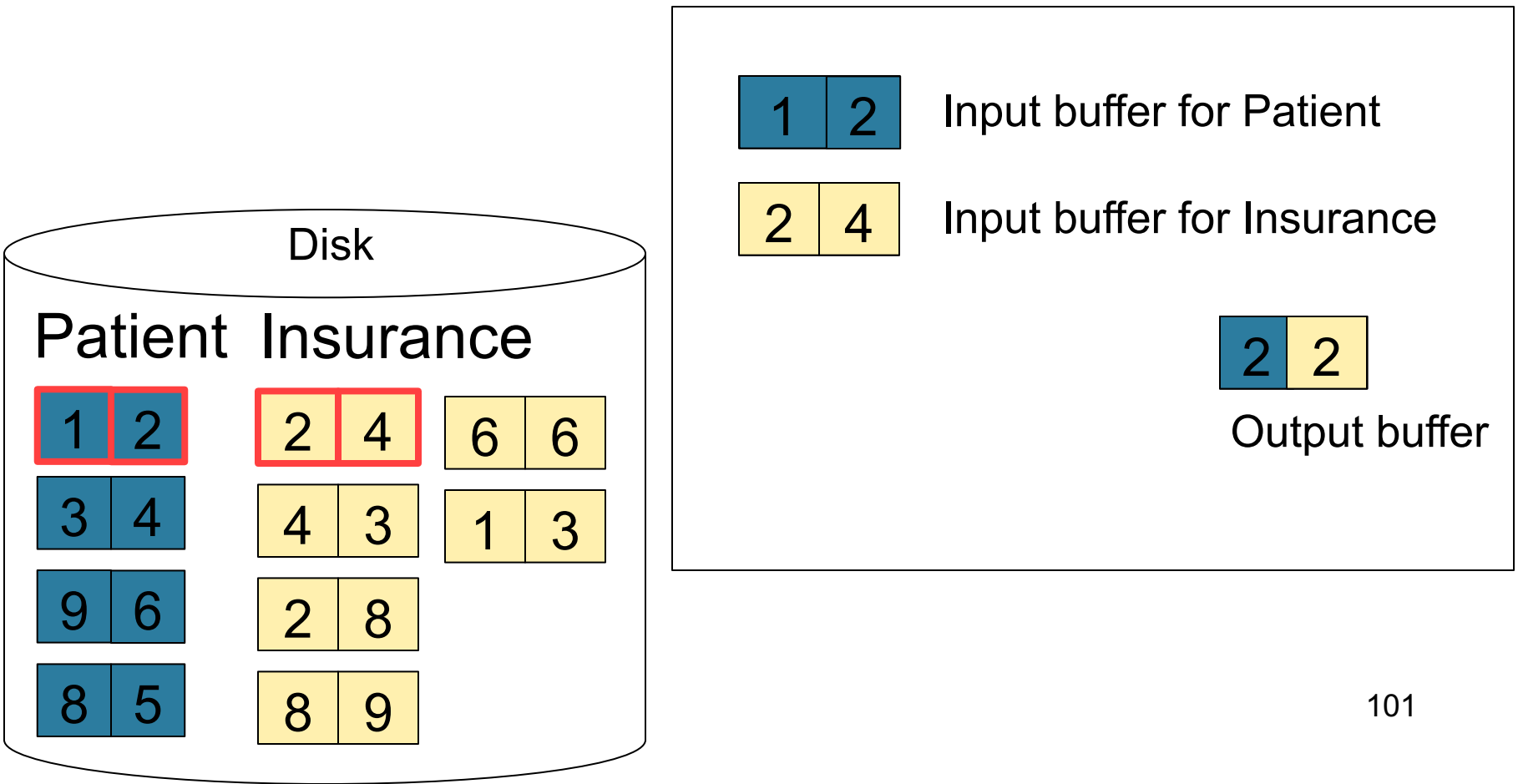
- Tuple-based nested loop R ⋈ S
- R is the outer relation, S is the inner relation

```
for each tuple t1 in R do
   for each tuple t2 in S do
      if t1 and t2 join then output (t1,t2)
```

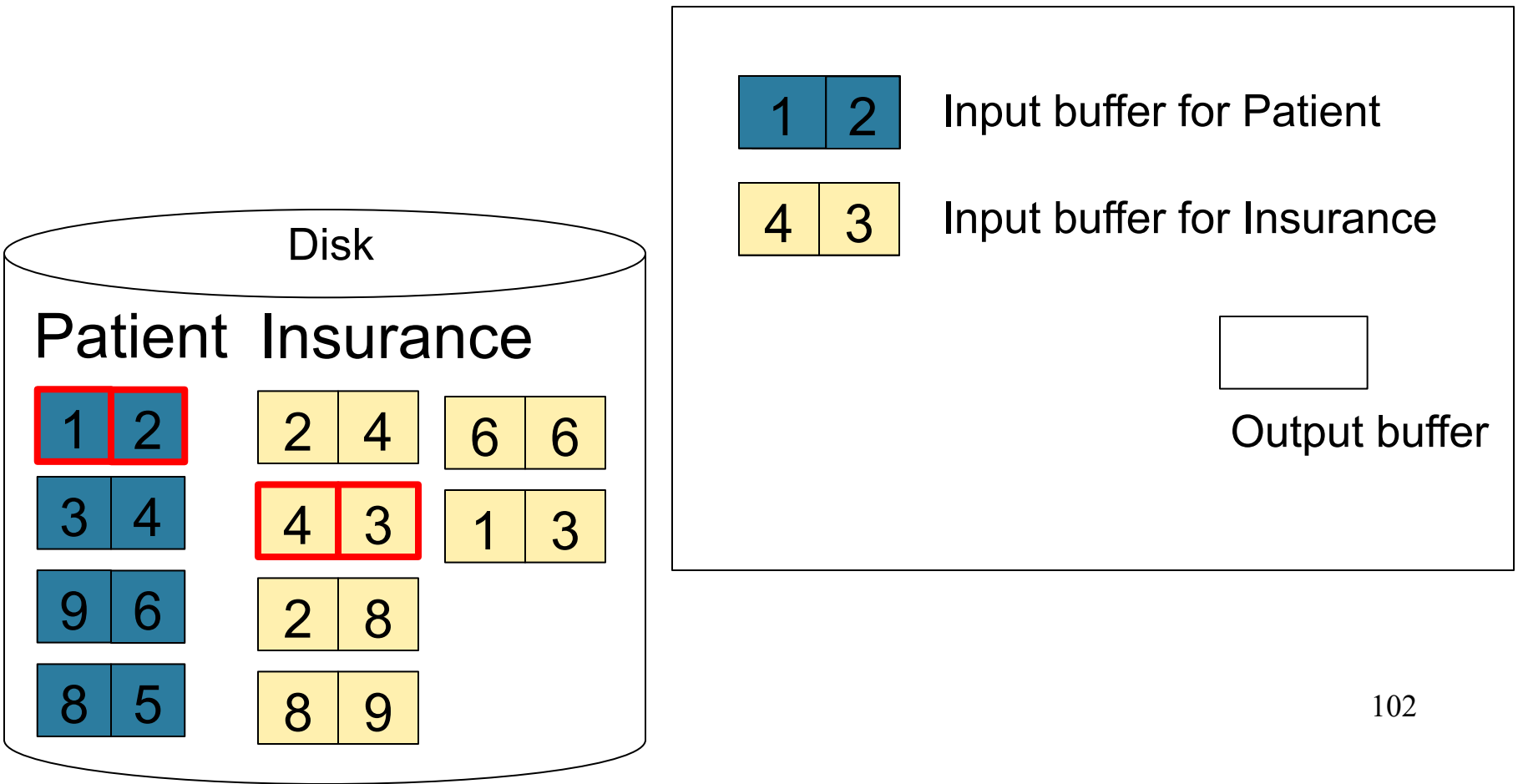
Page-at-a-time Refinement

```
for each page of tuples r in R do
   for each page of tuples s in S do
         for all pairs of tuples t1 in r, t2 in s
               if t1 and t2 join then output (t1,t2)
```

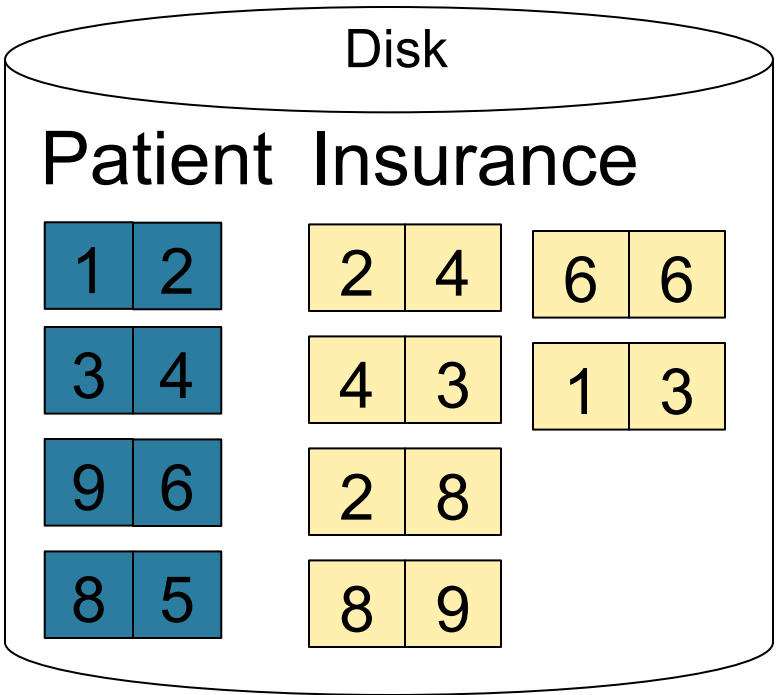
Page-at-a-time Refinement

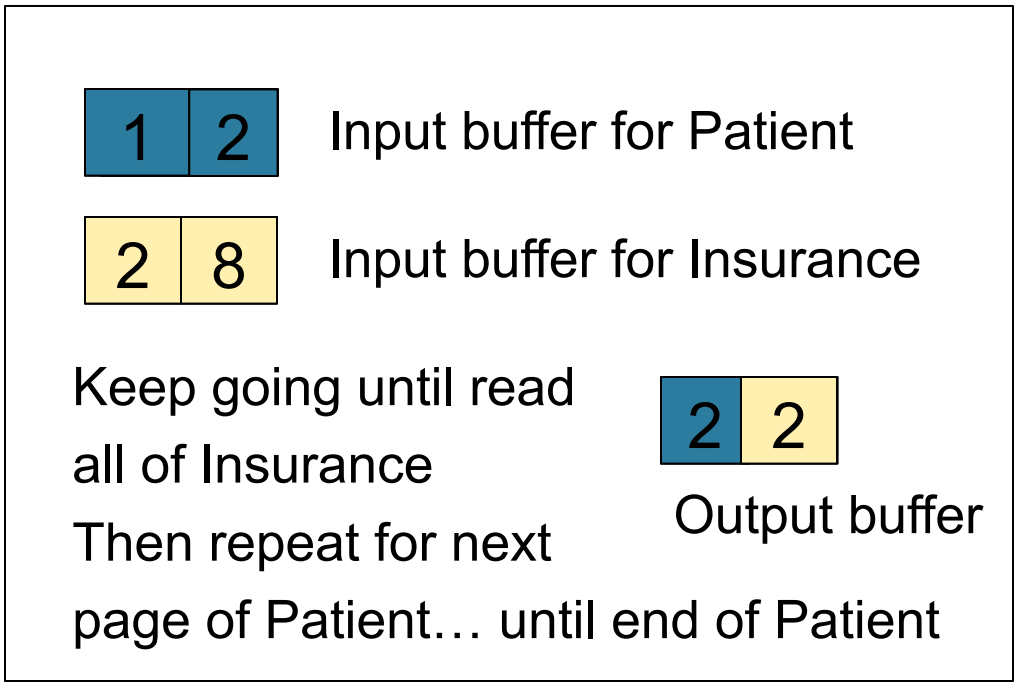


Page-at-a-time Refinement



Page-at-a-time Refinement





Block-Nested-Loop Refinement

```
for each group of M-1 pages r in R do
   for each page of tuples s in S do
         for all pairs of tuples t1 in r, t2 in s
               if t1 and t2 join then output (t1,t2)
```

Sort-Merge Join

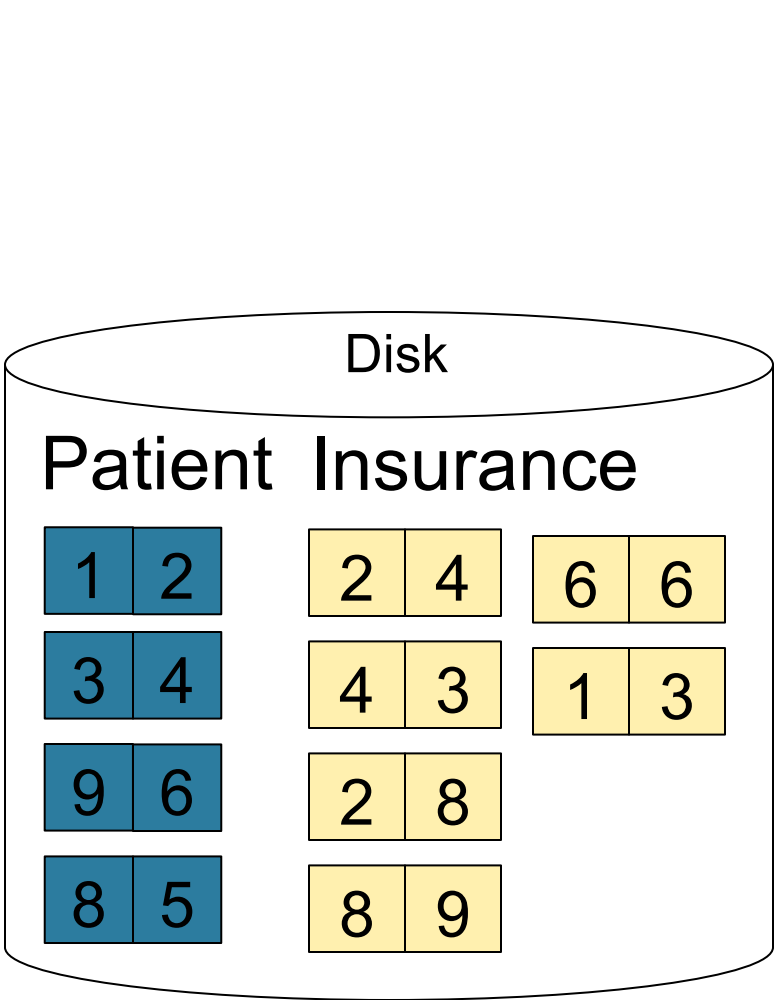
Sort-merge join: R ⋈ S

- Scan R and sort in main memory
- Scan S and sort in main memory
- Merge R and S

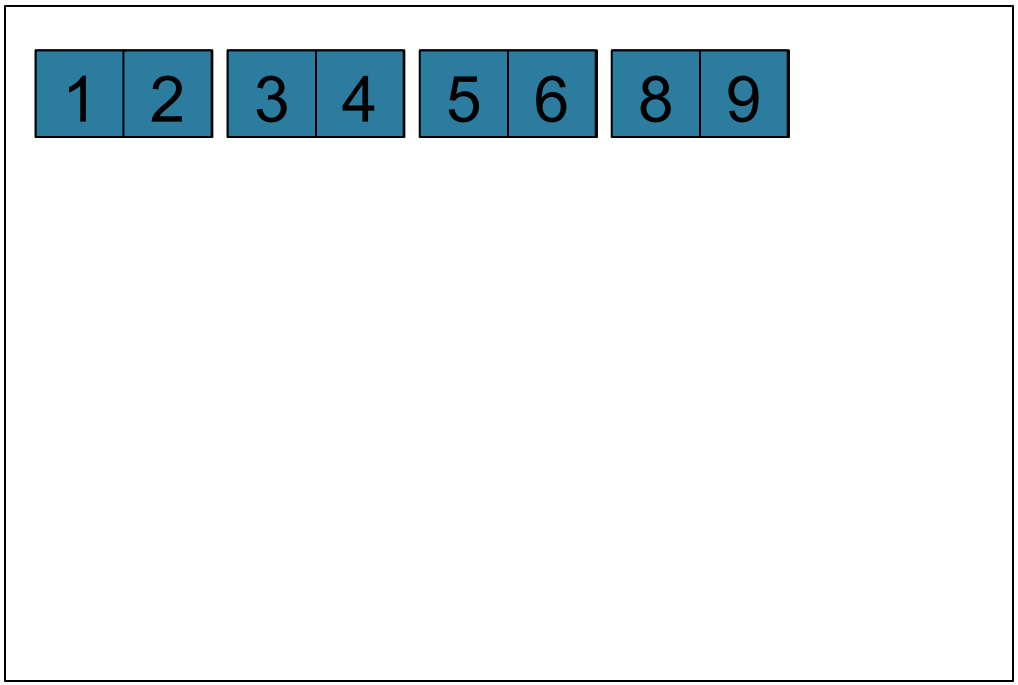
- Typically, not enough memory
  - Need to spill to disk during sorting

Sort-Merge Join Example

Step 1: Scan Patient and sort in memory

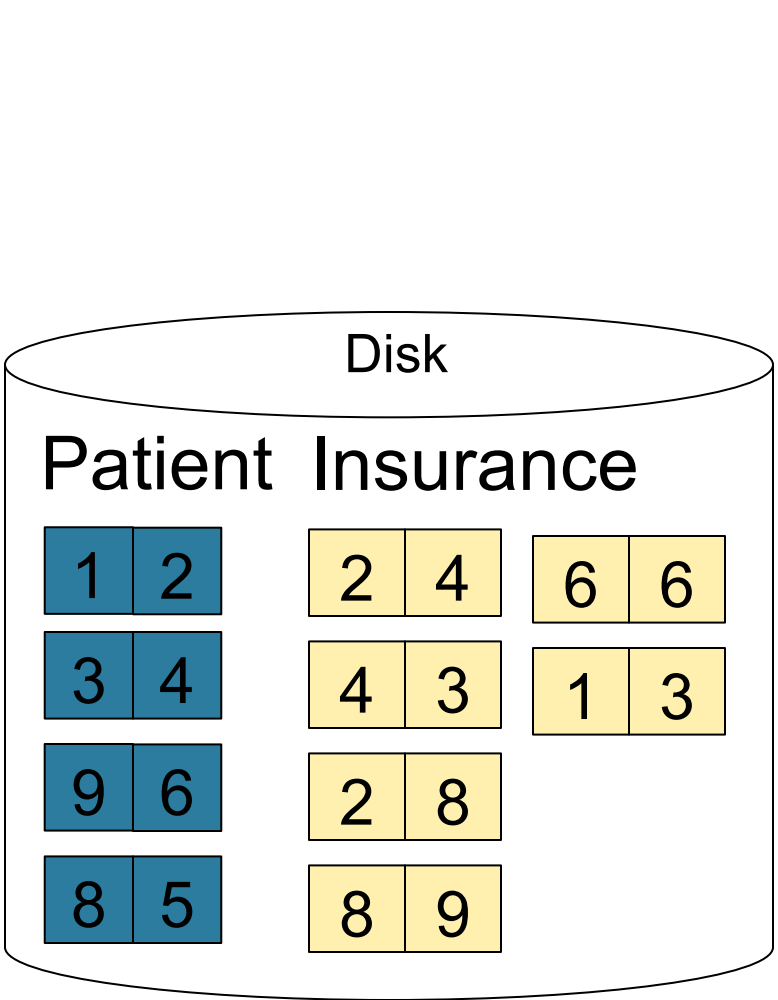


Memory M = 21 pages

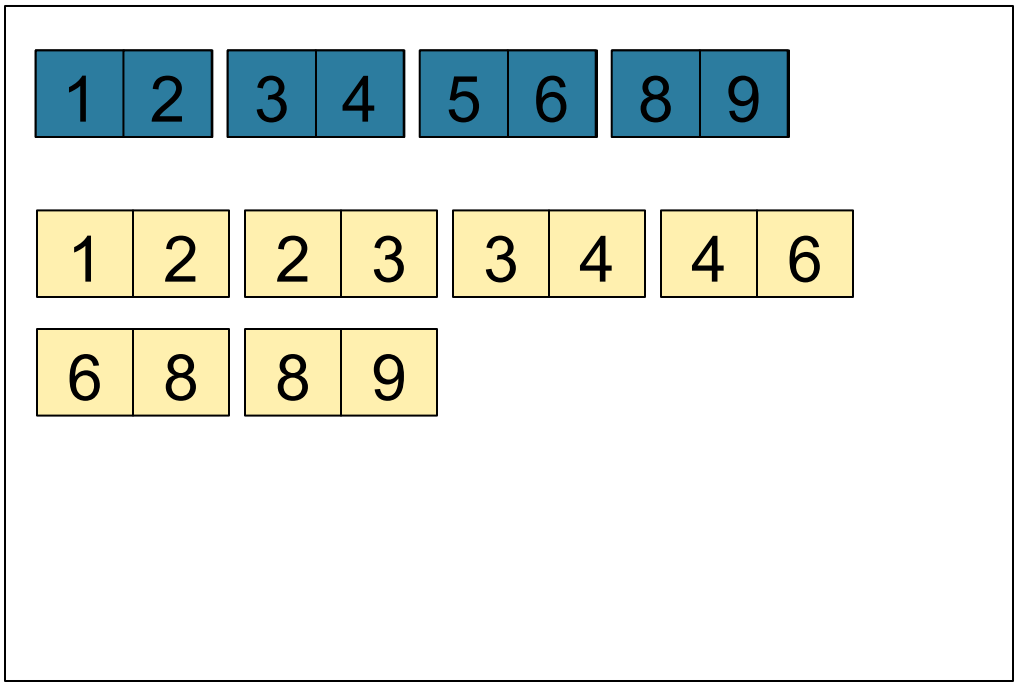


Sort-Merge Join Example

Step 2: Scan Insurance and sort in memory

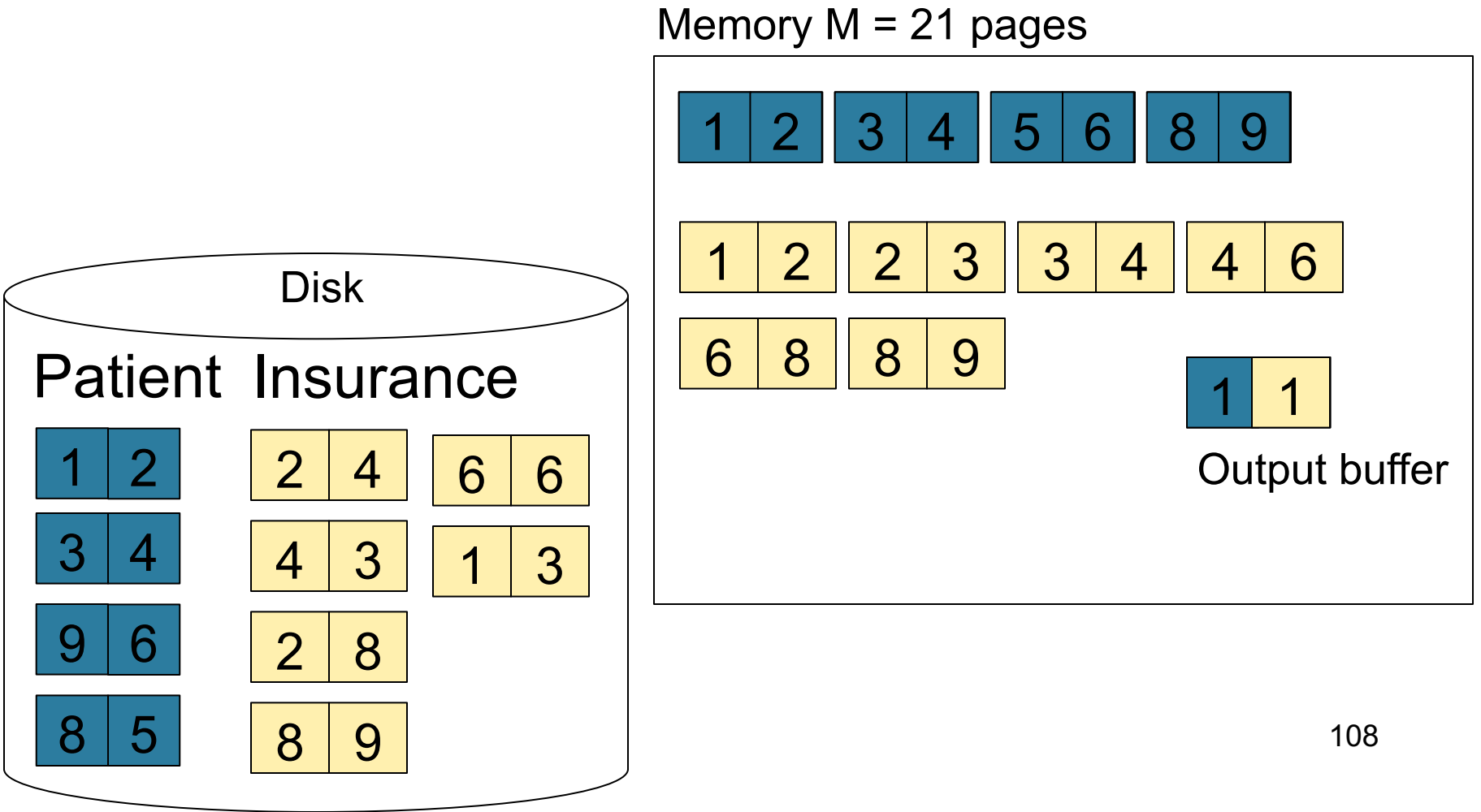


Memory M = 21 pages



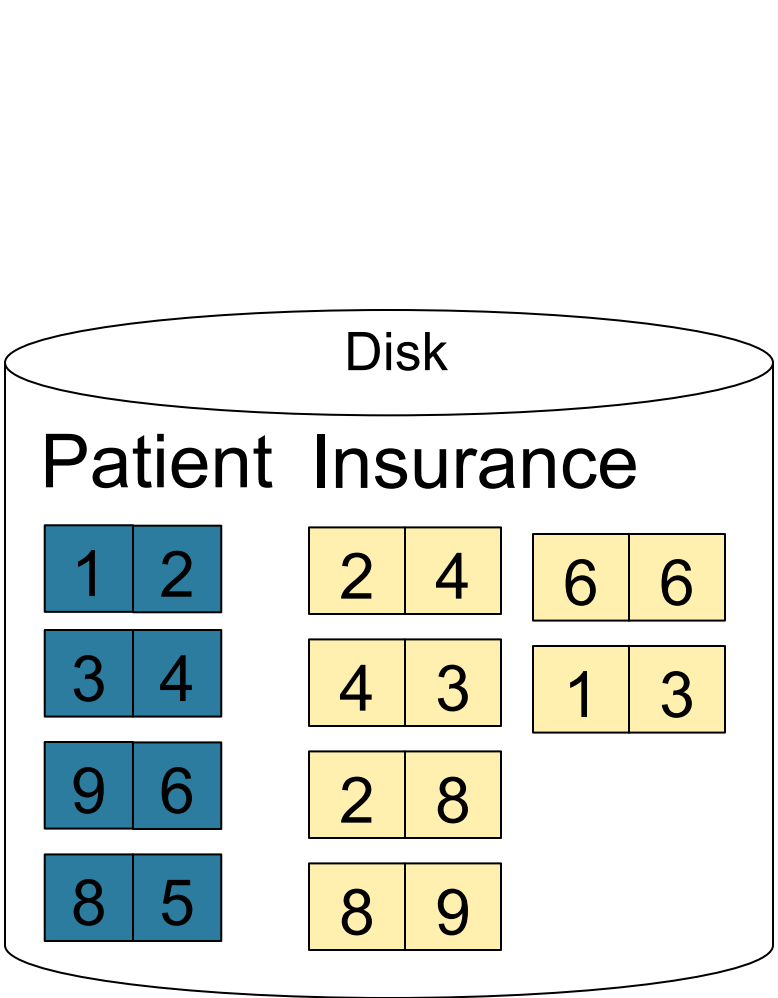
Sort-Merge Join Example

Step 3: Merge Patient and Insurance

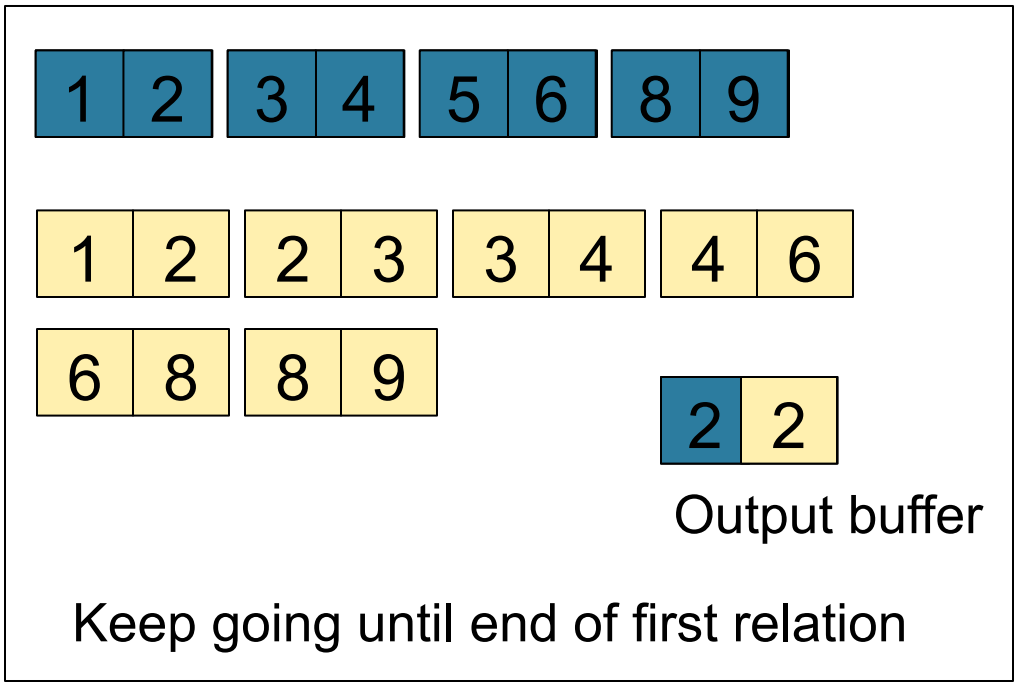


Sort-Merge Join Example

Step 3: Merge Patient and Insurance



Memory M = 21 pages



Index Nested Loop Join

R ⋈ S

- Assume S has an index on the join attribute
- Iterate over R, for each tuple fetch corresponding tuple(s) from S

Summary

- RDMBS are complex systems
- Need to know some of their basics inner workings in order to understand query performance

Next week: we start discussing parallel, shared-nothing databases