CSE544
Data Management

Lecture 4
Data Models

Announcements

- Tomorrow (Thursday): makeup lecture at 10:30am, CSE2 371
- Homework 1 is due on Friday
- Start to think about class projects

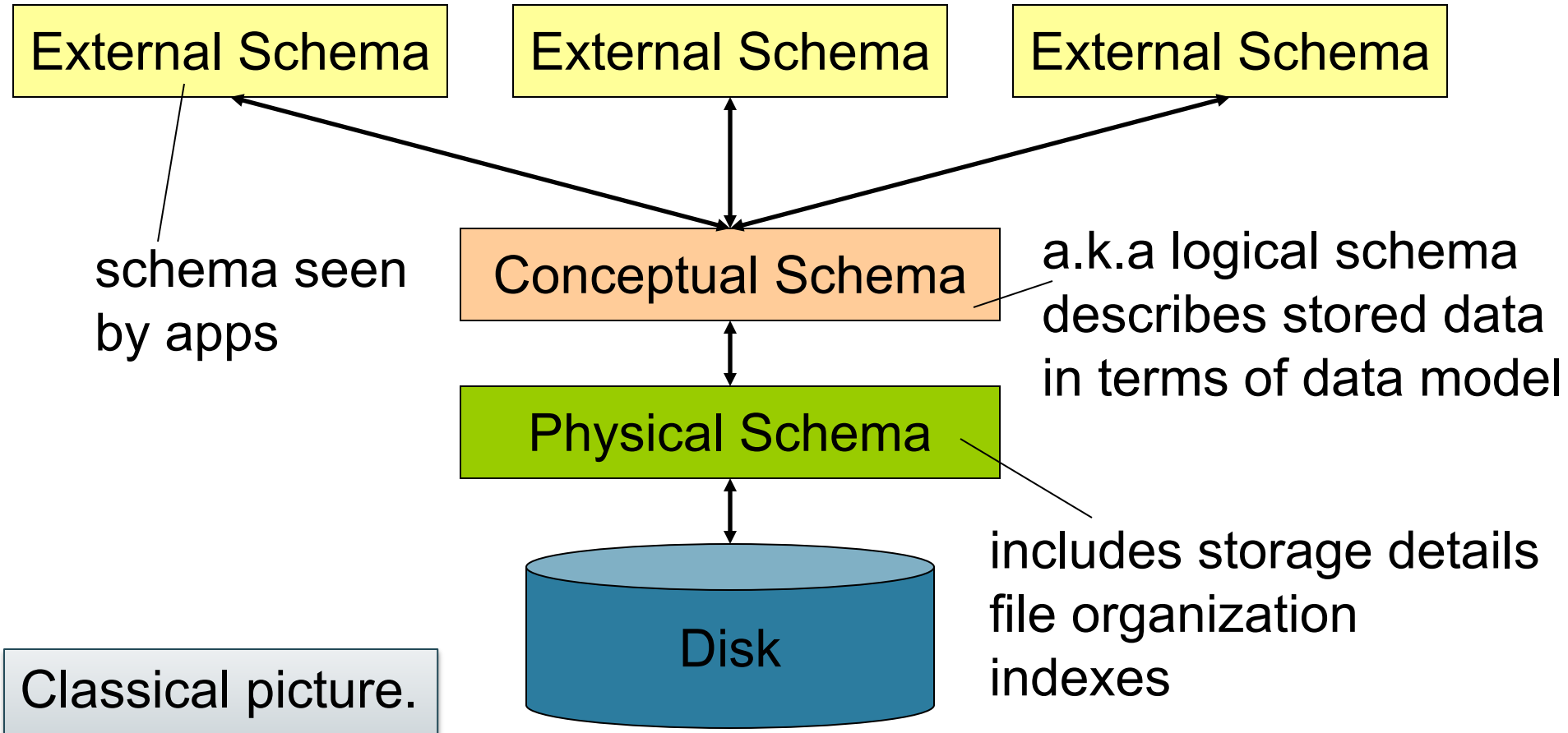
References

- M. Stonebraker and J. Hellerstein. What Goes Around Comes Around. In "Readings in Database Systems" (aka the Red Book). 4th ed.

Data Model Motivation

- Applications need to model real-world data
- User somehow needs to define data to be stored in DBMS
- Data model enables a user to define the data using high-level constructs without worrying about many low-level details of how data will be stored on disk

Levels of Abstraction



Classical picture.
Remember it !

Different Types of Data

- **Structured data**
  - All data conforms to a schema. Ex: business data
- **Semistructured data**
  - Some structure in the data but implicit and irregular
- **Unstructured data**
  - No structure in data. Ex: text, sound, video, images
- Our focus: structured data & relational DBMSs

Outline

- Early data models
  - IMS
  - CODASYL

- Physical and logical independence in the relational model

- Data models that followed the relational model

Early Proposal 1: IMS

- What is it?

Early Proposal 1: IMS

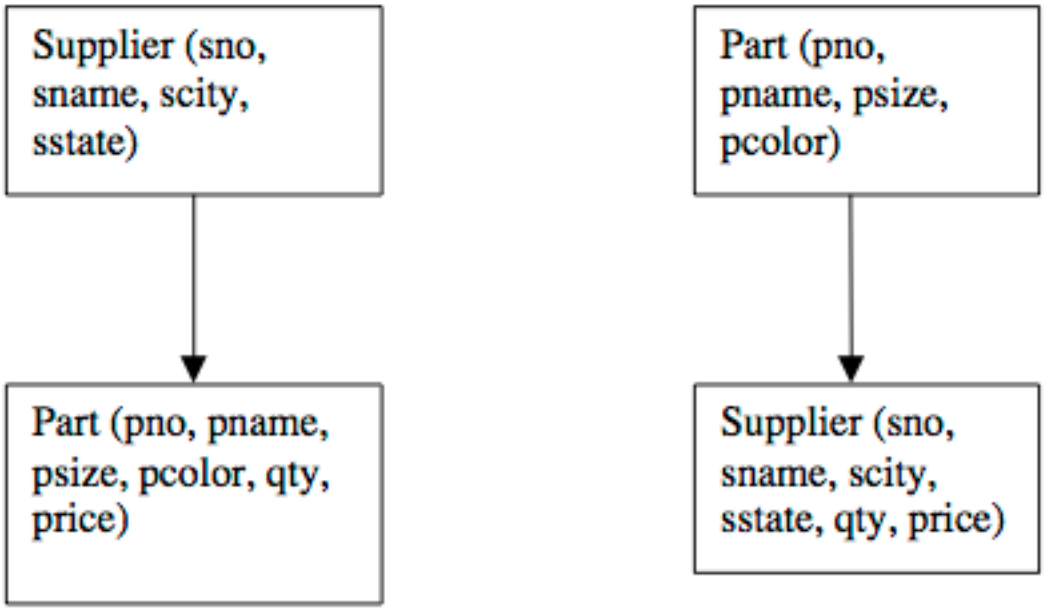
- **Hierarchical data model**

- **Record**
  - **Type**: collection of named fields with data types
  - **Instance**: must match type definition
  - Each instance has a **key**
  - Record types arranged in a **tree**

- **IMS database** is collection of instances of record types organized in a tree

IMS Example

- Figure 2 from “What goes around comes around”



Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

Data Manipulation Language: DL/1

How does a programmer retrieve data in IMS?

- Each record has a hierarchical sequence key (HSK)
- HSK defines semantics of commands:
  - get_next; get_next_within_parent
- **DL/1 is a record-at-a-time language**
  - Programmers construct algorithm, worry about optimization

Data storage

How is data physically stored in IMS?

Data storage

How is data physically stored in IMS?

- Root records
  - Stored sequentially (sorted on key)
  - Indexed in a B-tree using the key of the record
  - Hashed using the key of the record
- Dependent records
  - Physically sequential
  - Various forms of pointers
- Selected organizations restrict DL/1 commands
  - No updates allowed due to sequential organization
  - No “get-next” for hashed organization

Data Independence

What is it?

Data Independence

What is it?

- **Physical data independence**: Applications are insulated from changes in **physical storage details**

- **Logical data independence**: Applications are insulated from changes to **logical structure of the data**

IMS Limitations

- **Tree-structured data model**
  - Redundant data; existence depends on parent
- **Record-at-a-time** user interface
  - User must specify algorithm to access data
- **Very limited physical independence**
  - Phys. organization limits possible operations
  - Application programs break if organization changes
- **Some logical independence but limited**

Early Proposal 2: CODASYL

What is it?

Early Proposal 2: CODASYL

What is it?

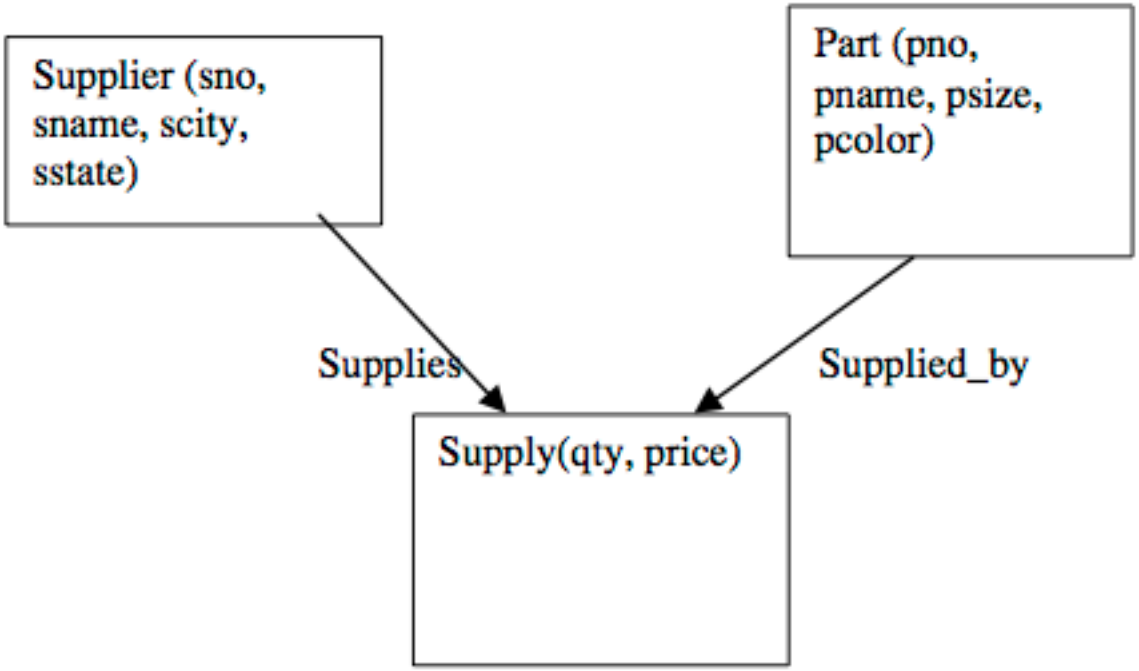
- **Networked data model**

- Primitives are also **record types** with **keys**
- Record types are organized into **network**
  - Multiple parents; arcs = “sets”; at least one entry point
- More flexible than hierarchy

- **Record-at-a-time** data manipulation language

CODASYL Example

- Figure 5 from “What goes around comes around”



CODASYL Limitations

- **No physical data independence**
  - Application programs break if organization changes
- **No logical data independence**
  - Application programs break if organization changes
- Very complex
- Programs must “navigate the hyperspace”
- Load and recover as one gigantic object

The Programmer as Navigator

by Charles W. Bachman

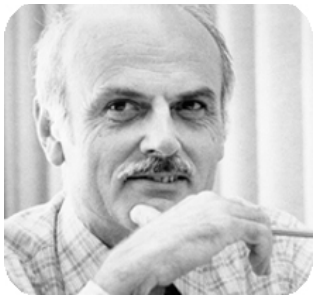
Outline

- Early data models
  - IMS
  - CODASYL
- Physical and logical independence in the relational model
- Data models that followed the relational model

Relational Model Overview

- Proposed by Ted Codd in 1970
- Motivation: logical and physical data independence
- Overview
  - Store data in a **simple data structure** (table)
  - Access data through **set-at-a-time** language
  - **No need for physical storage proposal**

Relational Database: A Practical Foundation for Productivity



Great Debate

- Pro relational
  - What were the arguments?
- Against relational
  - What were the arguments?
- How was it settled?

Great Debate

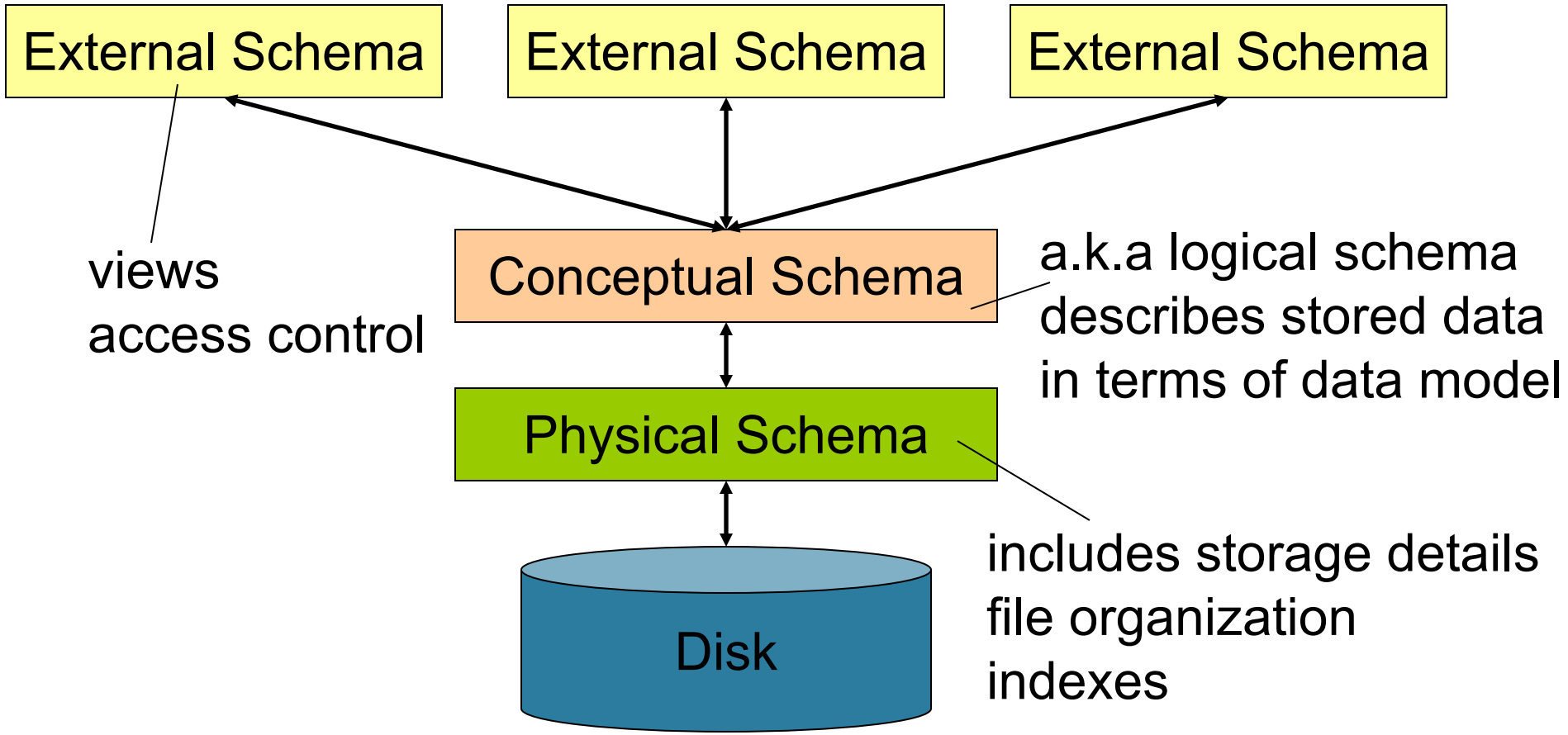
- Pro relational
  - CODASYL is too complex
  - CODASYL does not provide sufficient data independence
  - Record-at-a-time languages are too hard to optimize
  - Trees/networks not flexible enough to represent common cases
- Against relational
  - COBOL programmers cannot understand relational languages
  - Impossible to represent the relational model efficiently
- Ultimately settled by the market place

Data Independence

How is data independence is achieved in RDBMS today:

- Physical independence:
  - SQL to Query Plan
- Logical independence
  - SQL Views

Levels of Abstraction



Query Optimizer

- SQL → Relational Algebra Plan (RA)
- RA → Optimized RA
- Algebraic identities of RA; examples:
  1. Pushing selections down
  2. Join reorder

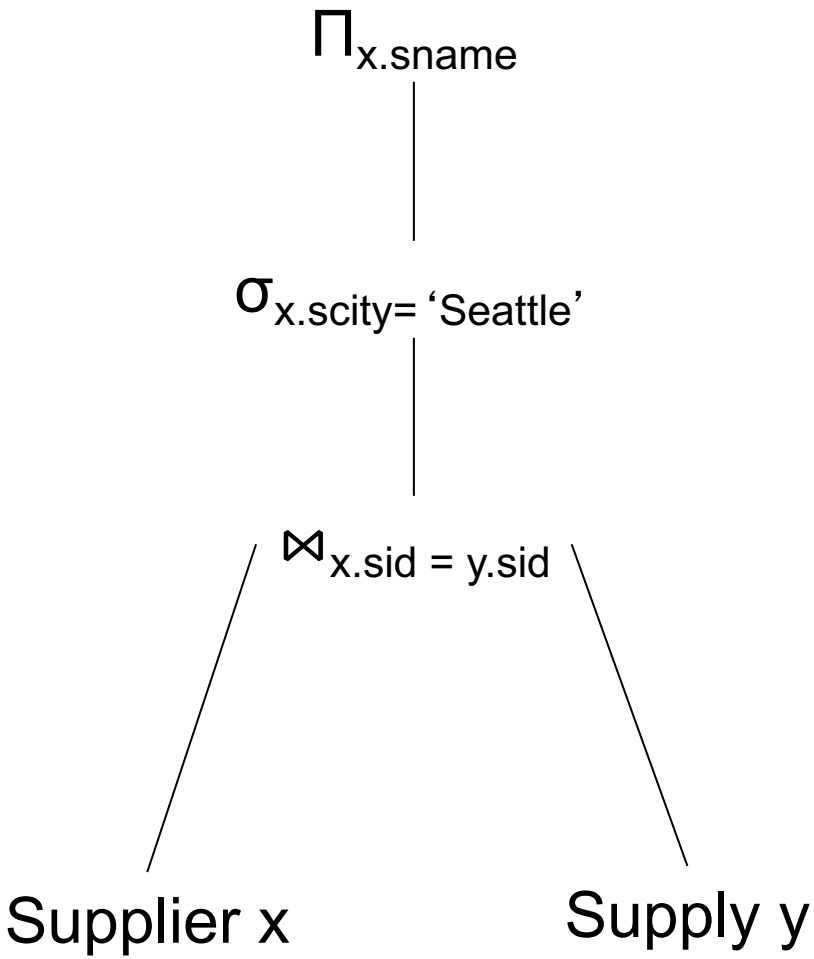
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Example Optimization

```
SELECT x.name
FROM.   Supplier x, Supply y
WHERE x.sid = y.sid
        and x.scity = 'Seattle'
```

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

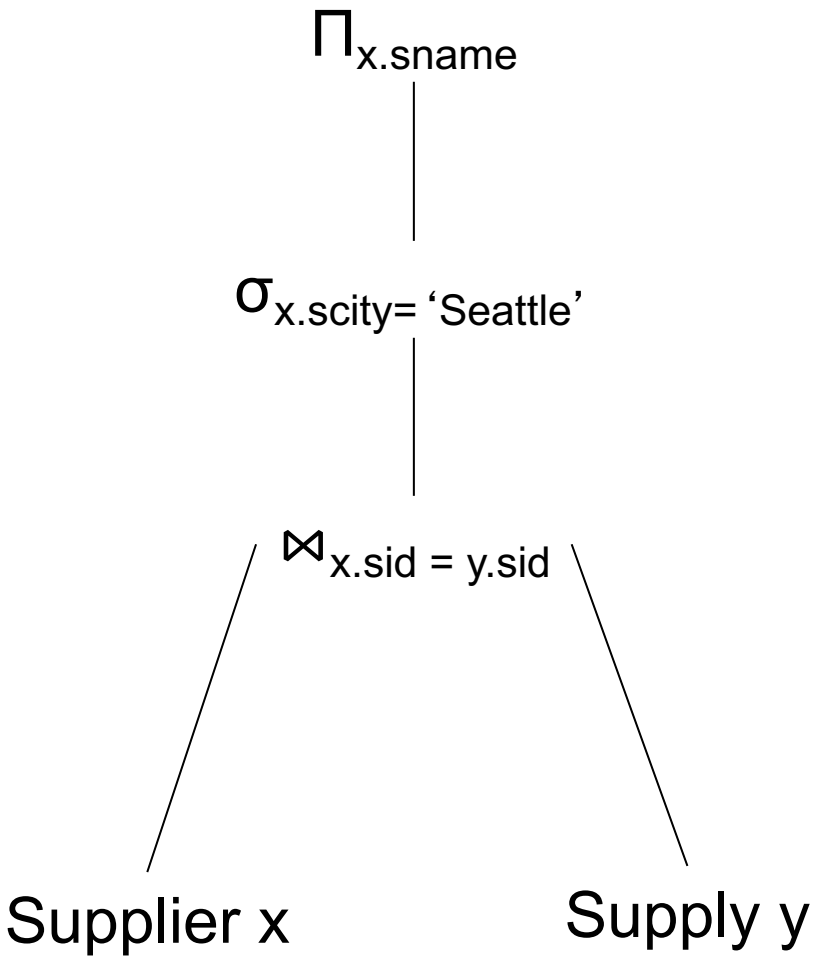
Example Optimization




```
SELECT x.name
FROM.   Supplier x, Supply y
WHERE x.sid = y.sid
       and x.scity = 'Seattle'
```

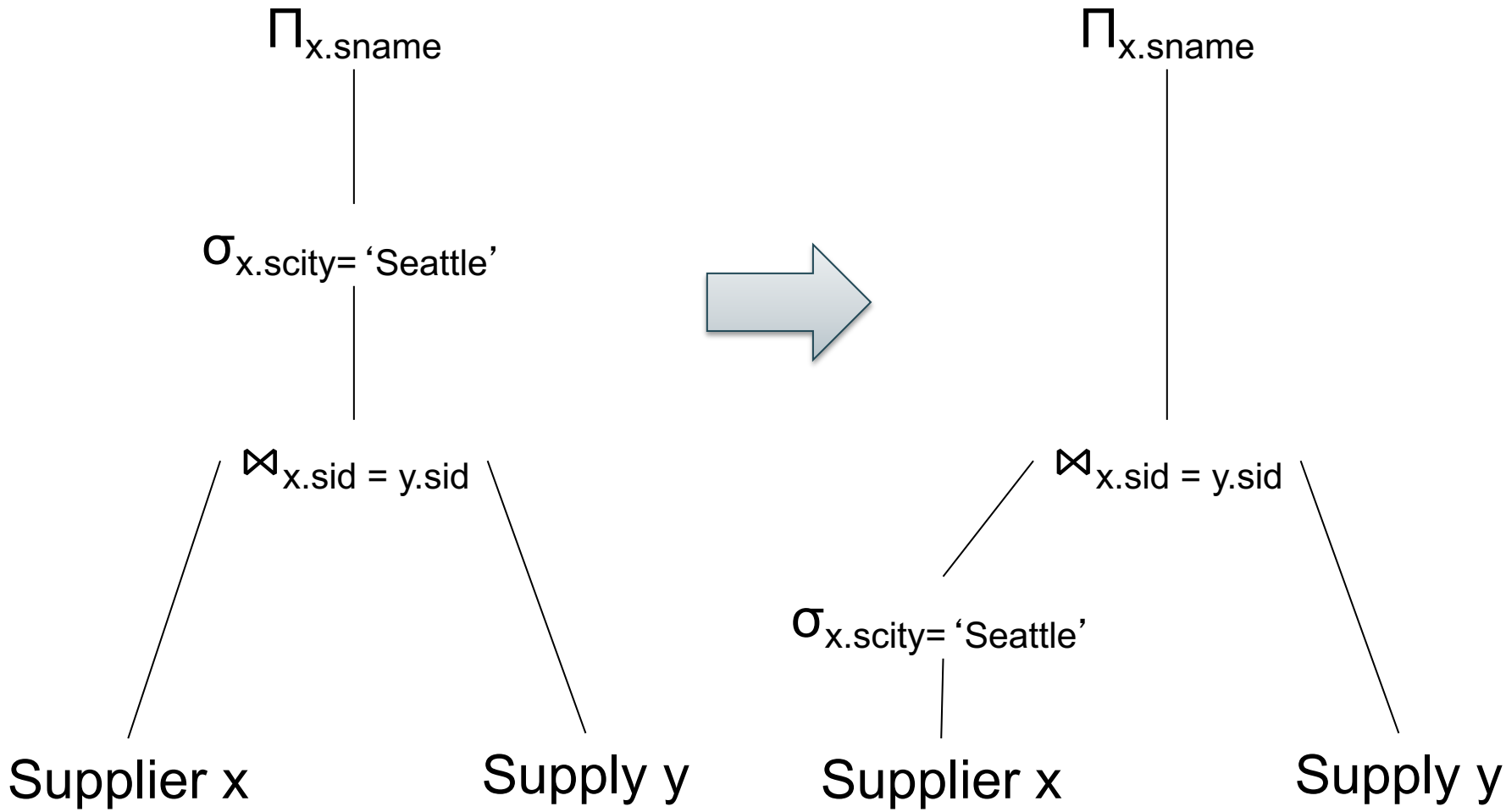
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



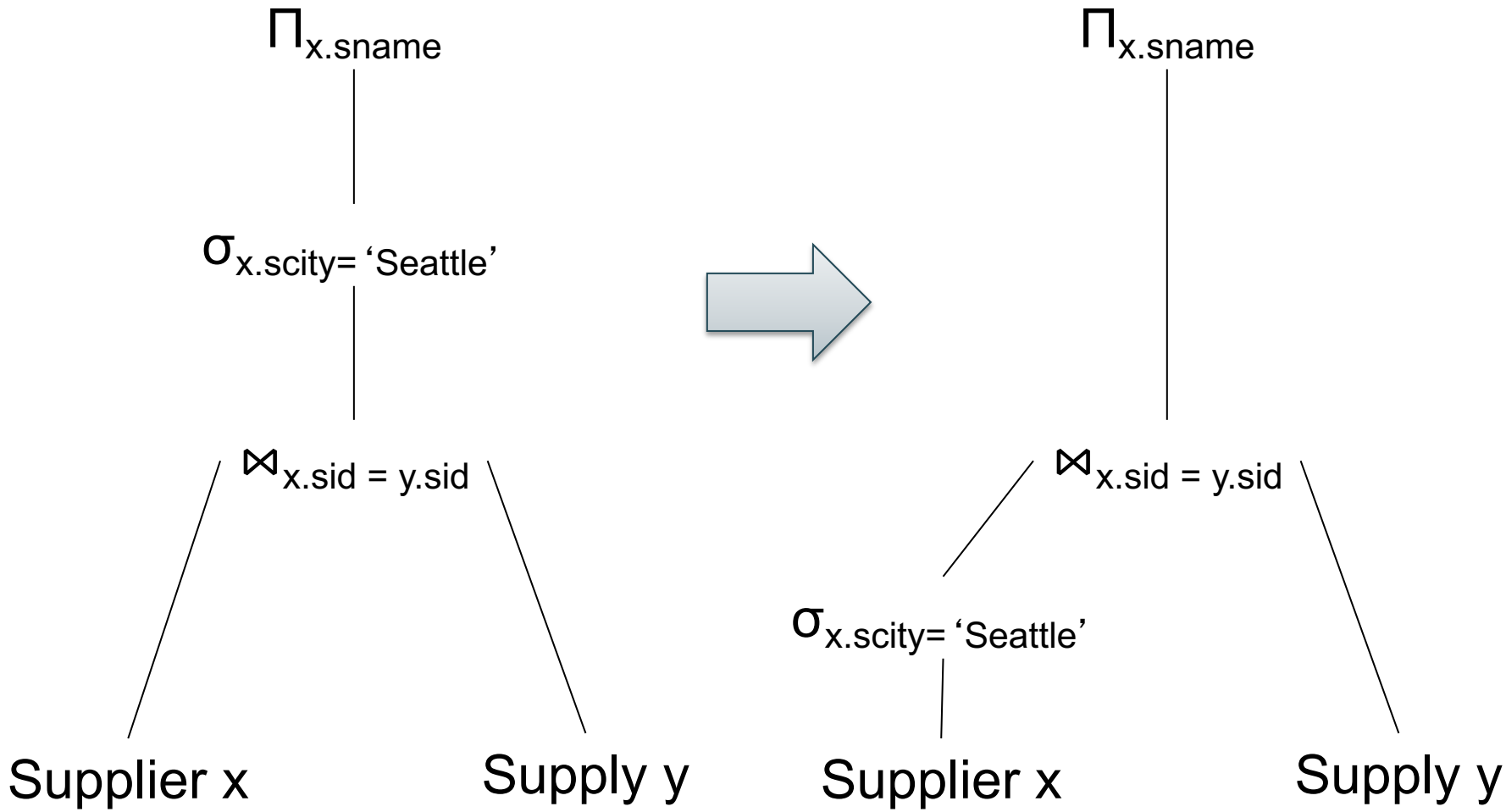
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

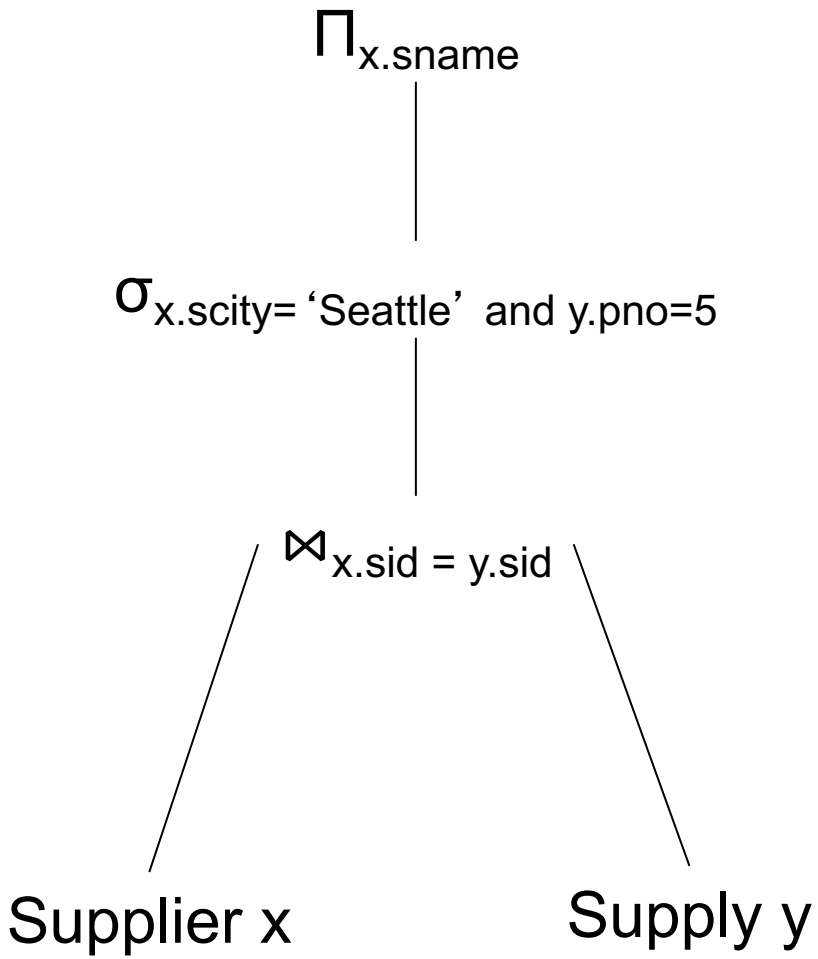
Push Selections Down



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

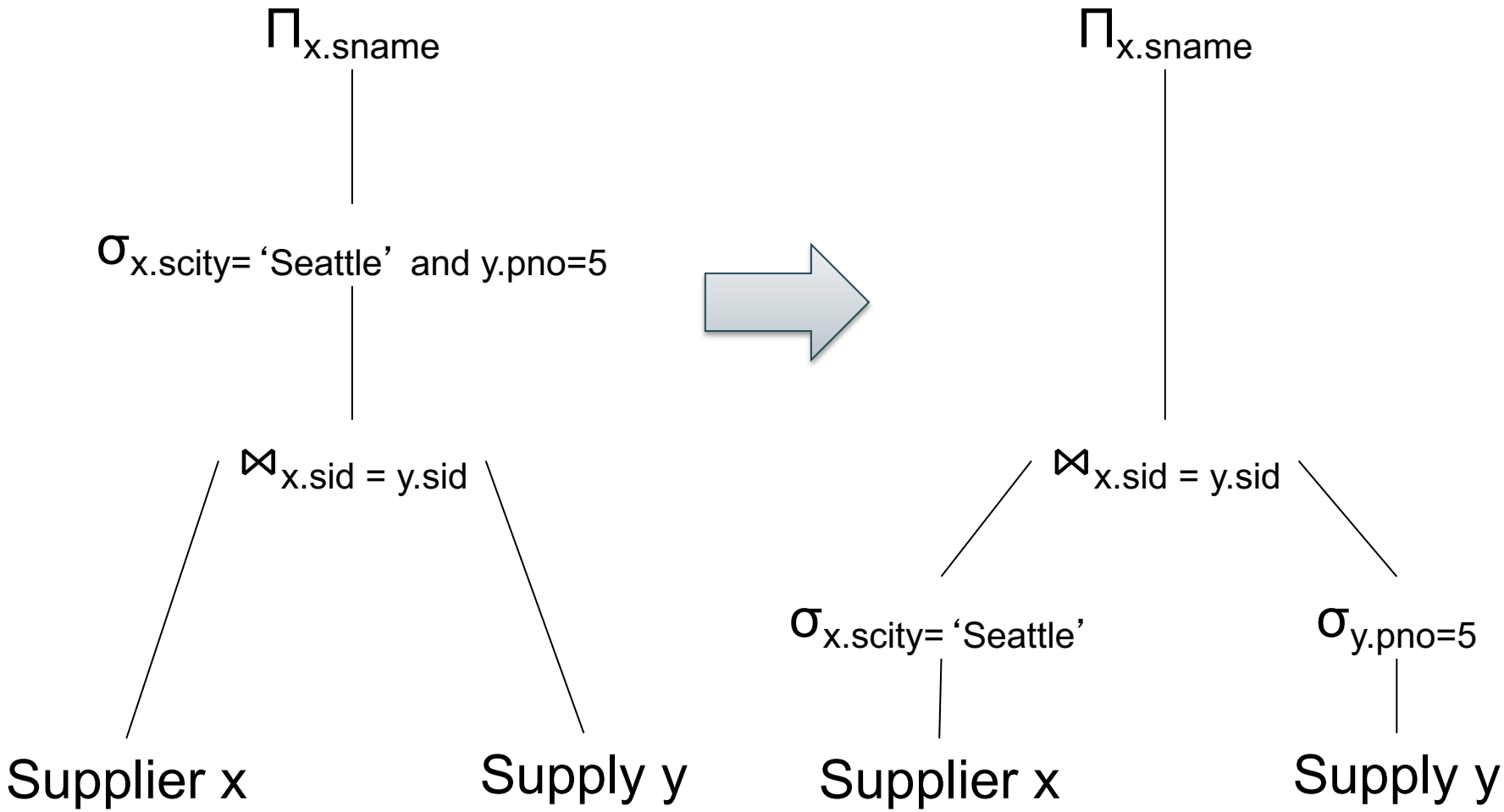
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



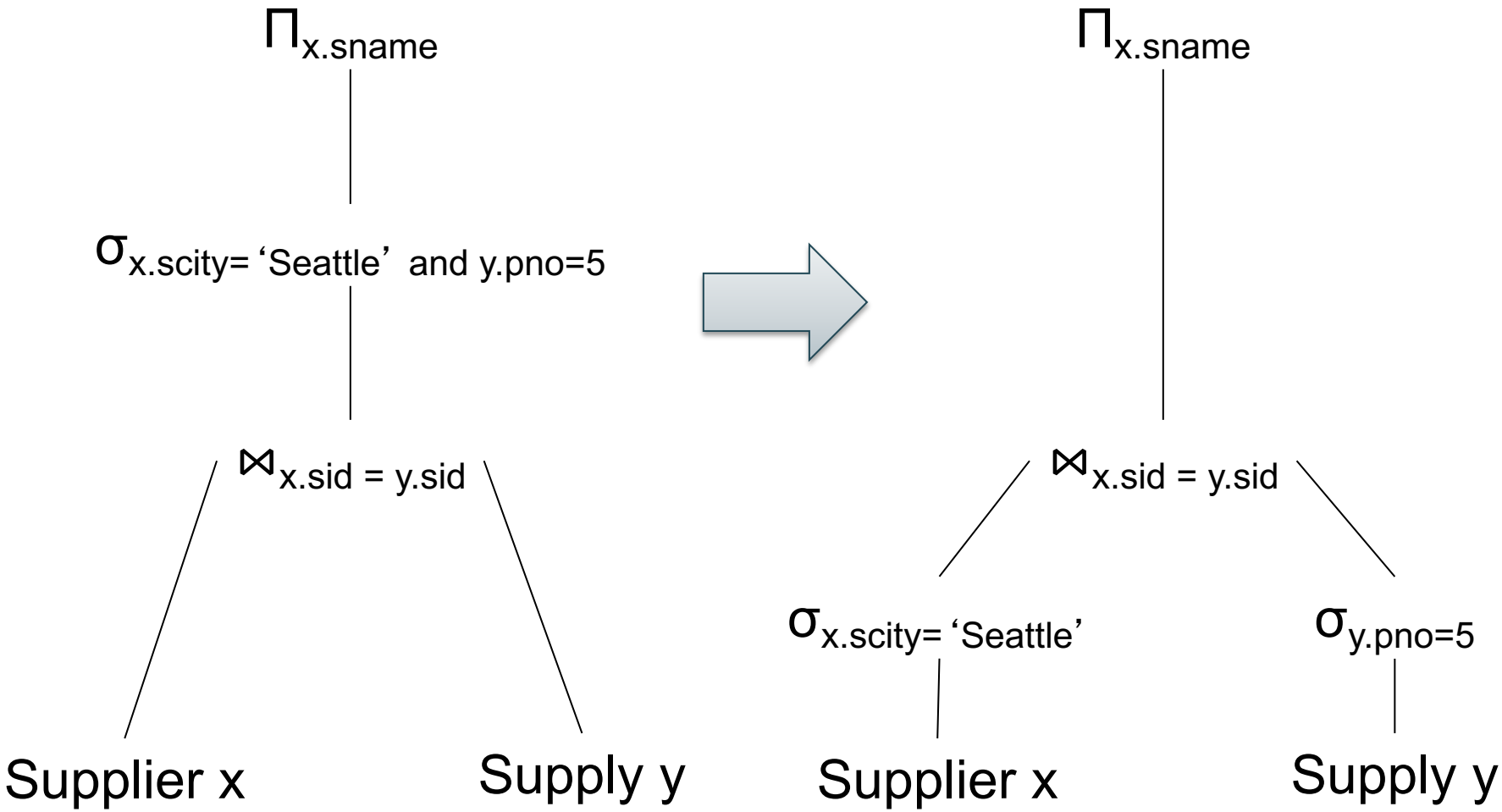
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)

Push Selections Down



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

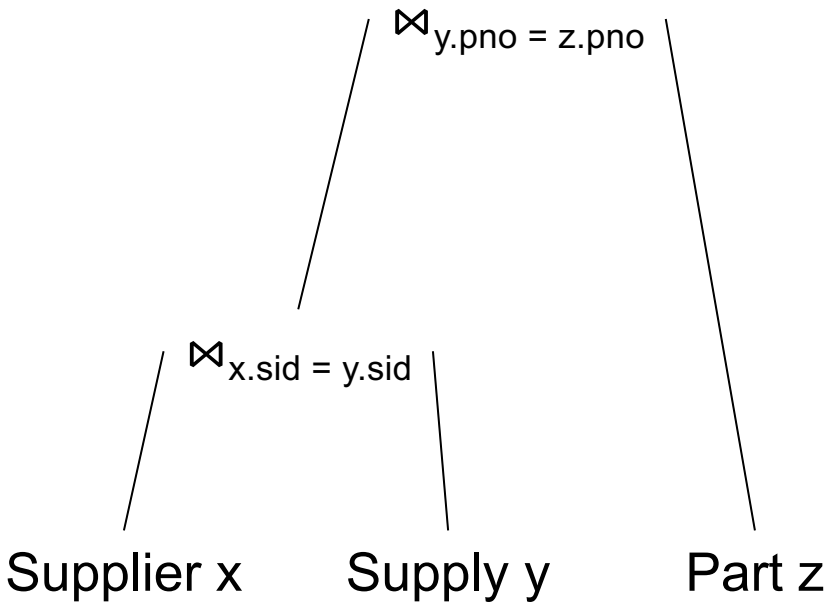
Push Selections Down



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

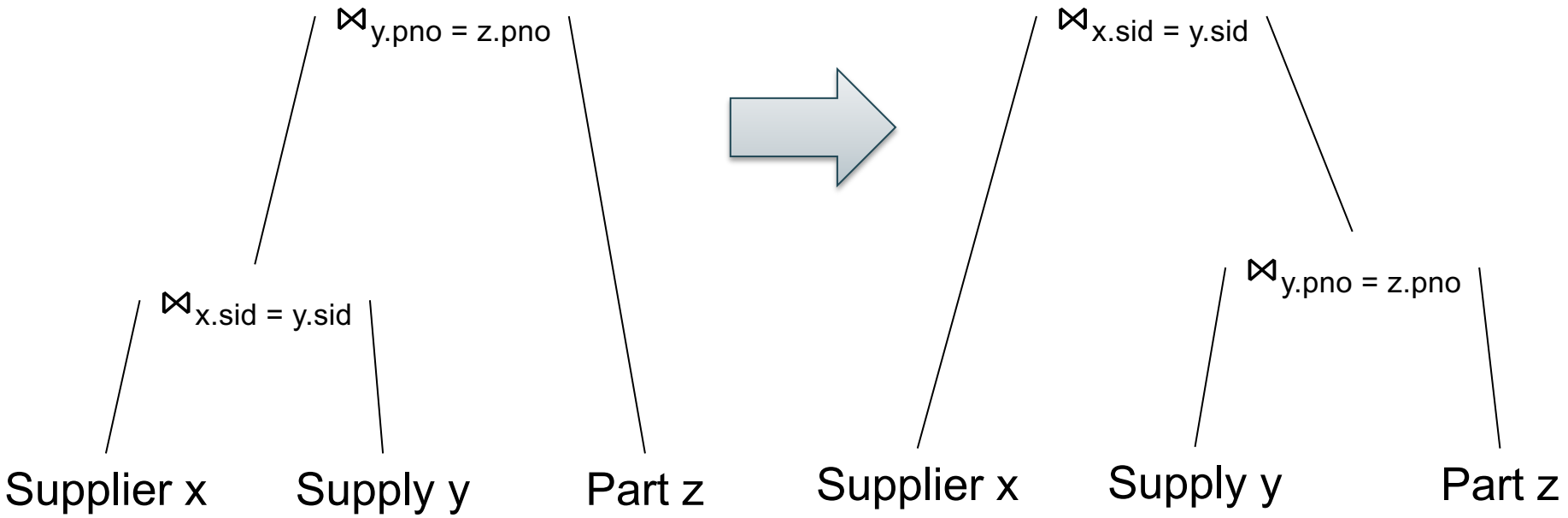
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



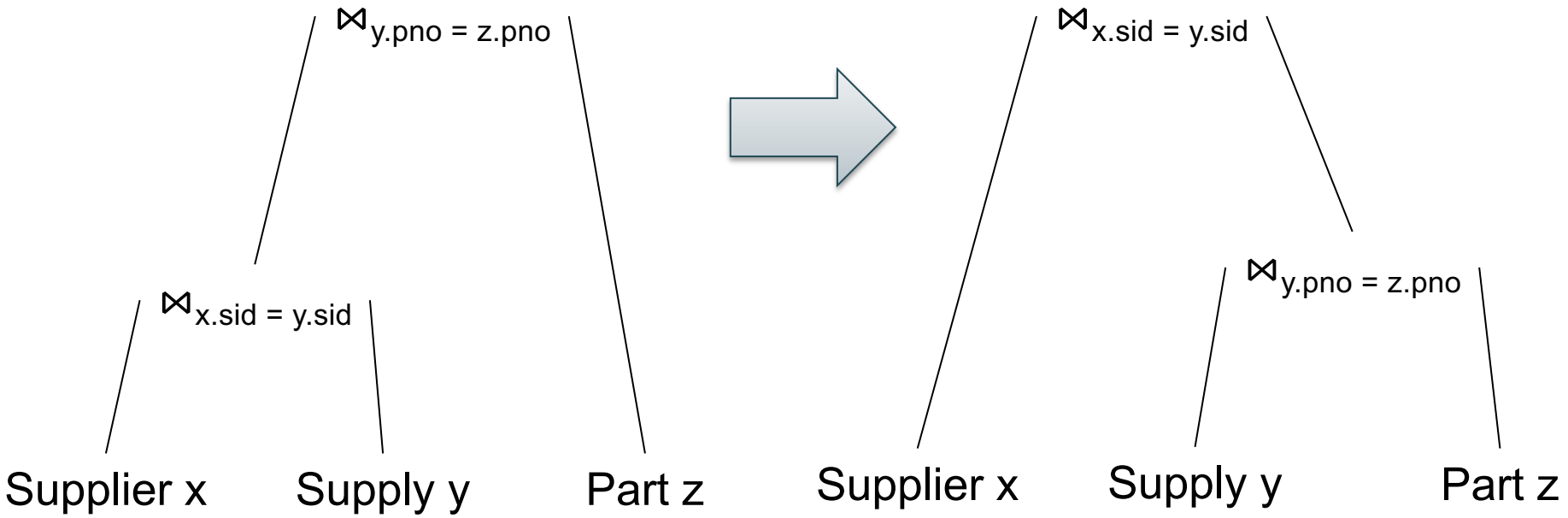
Supplier(sid, sname, scity, sstate)
Supply(sid, pno, quantity)
Part(pno, pname, pprice)

Join Reorder



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

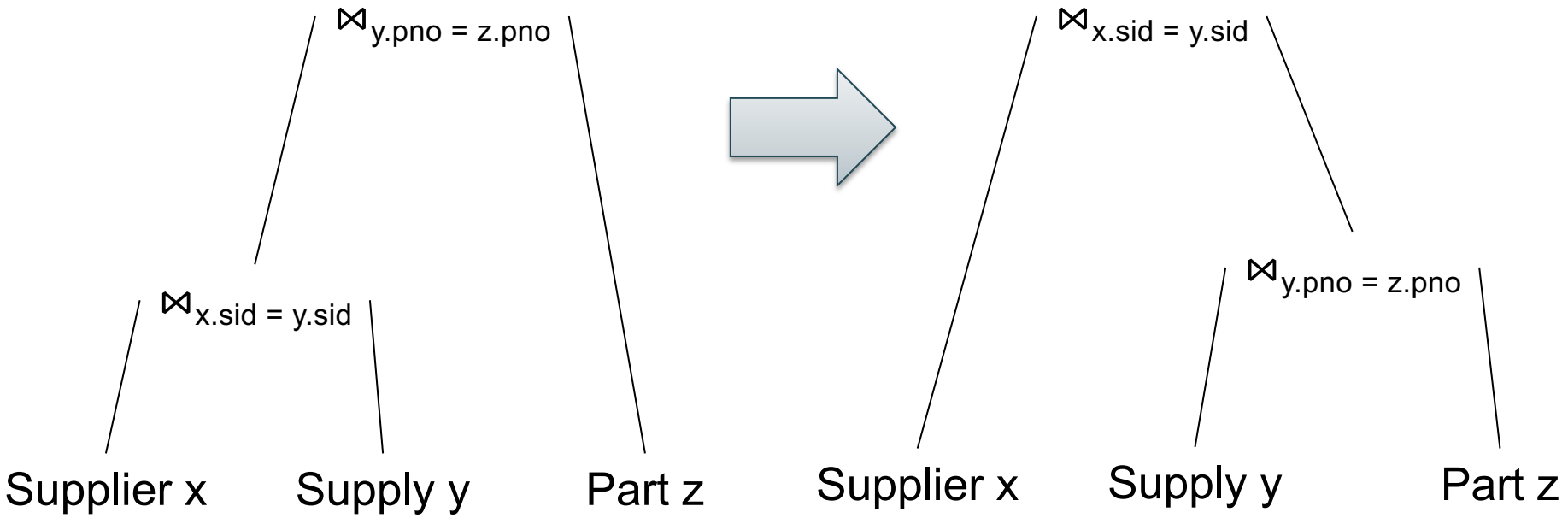
Join Reorder



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

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

Join Reorder



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

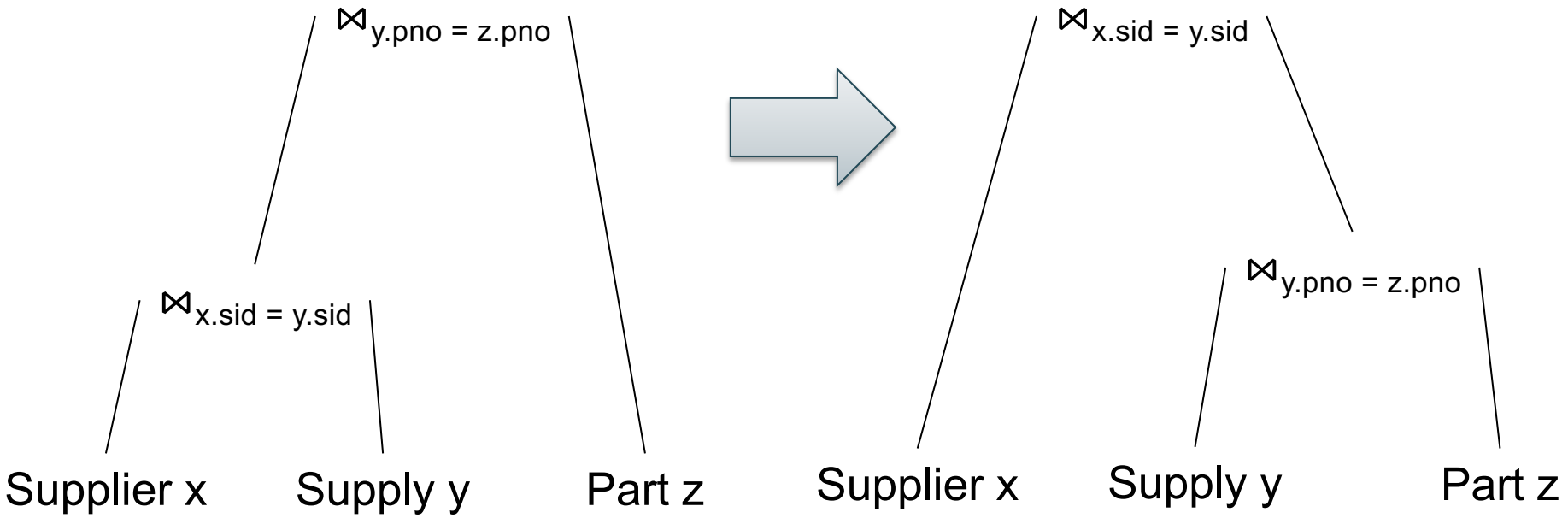
Also:

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

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

Join Reorder

When is one plan better than the other?

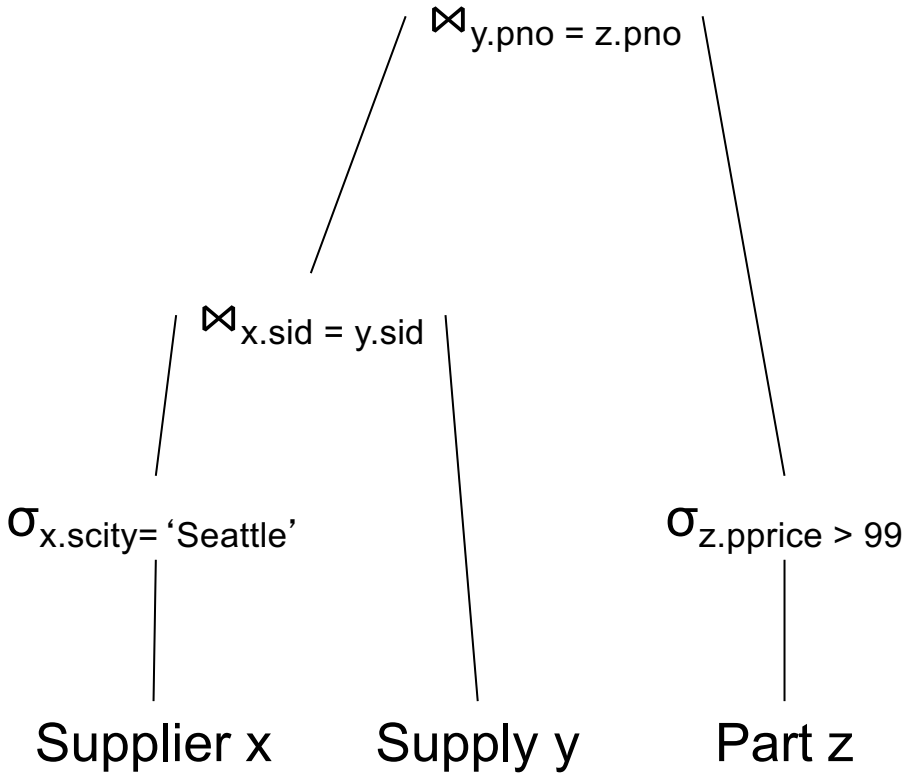


$(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

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

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

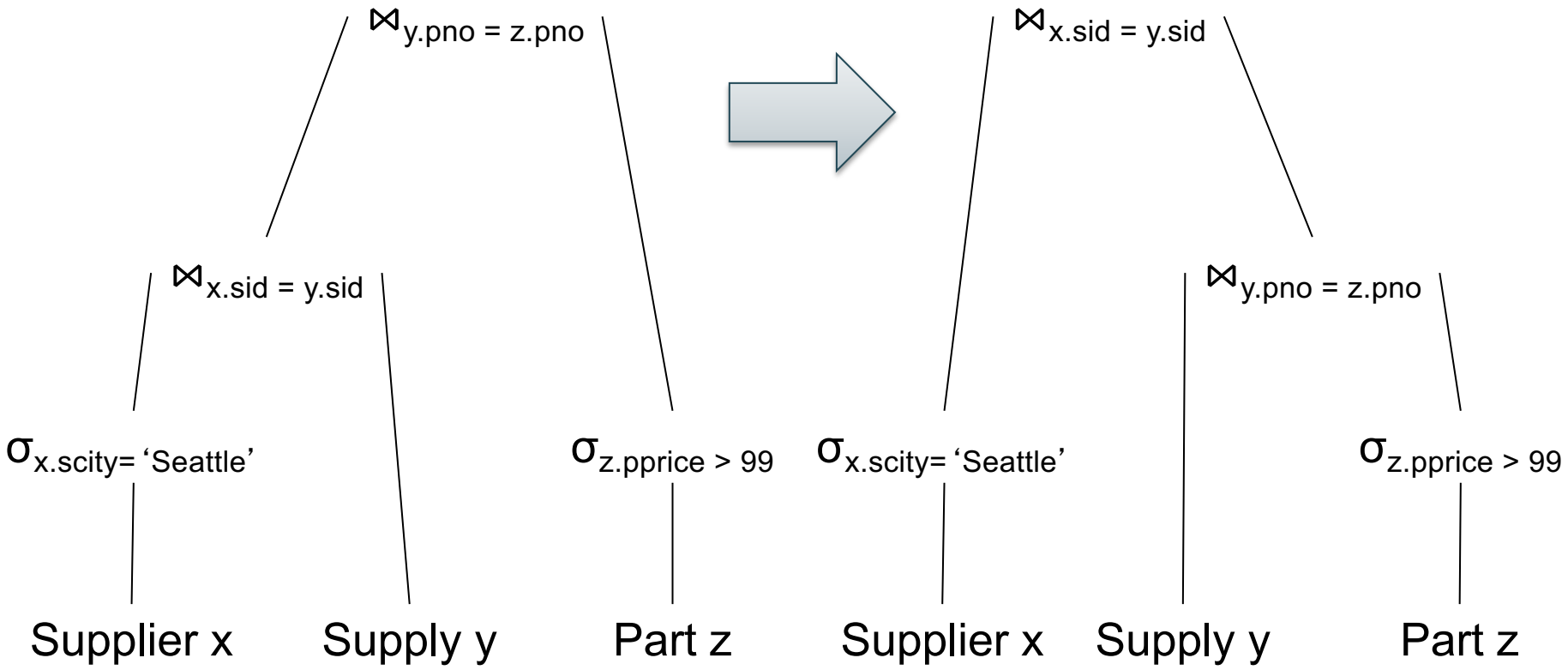
Join Reorder



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

Join Reorder

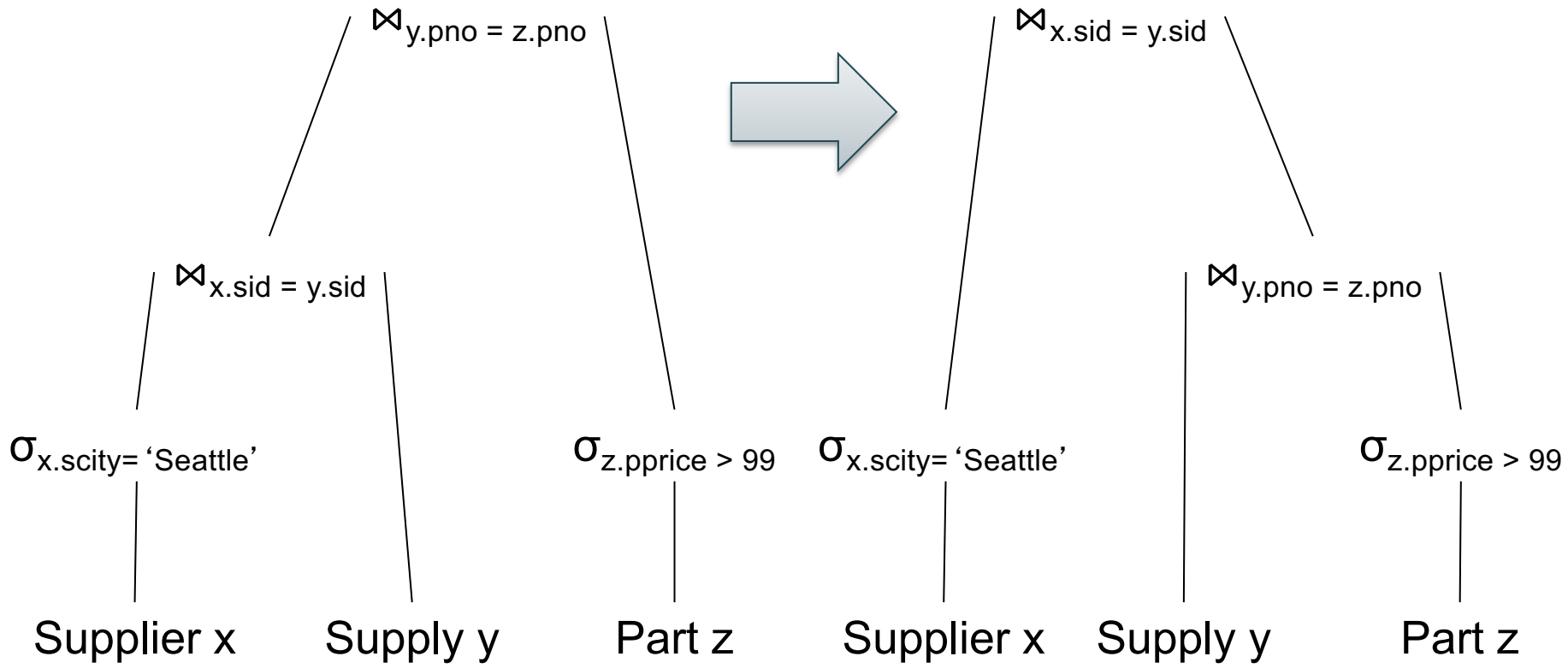
When is one plan better than the other?



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)

Summary of Phys. Data Indep

Physical data independence based on:

- SQL to Query Plan
- Query Plan is optimized
- Optimized Query Plan to Physical Plan

Will discuss this later in this course

Logical Independence

- Applications are insulated from changes to logical structure of the data

- Relational model
  - Logical independence through views

Logical Data Independence

In SQL: based on views

A View is a relation

- Virtual views:
  - Computed on demand, at query time
  - Default in SQL, and what Stonebraker means in the paper
- Materialized views:
  - Computed and stored persistently
- Pros and cons?

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

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

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

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

Virtual table: Big_Parts(pno,pname,psize,pcolor)

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

View Example

View definition:

```
CREATE VIEW Big_Parts AS
        SELECT * FROM Part
        WHERE psize > 10;
```

Virtual table: Big_Parts(pno,pname,psize,pcolor)

Querying the view:

```
SELECT *
FROM Big_Parts
WHERE pcolor='blue';
```

Updating Through Views

- **Updatable views** (SQL-92)
  - Defined on single base relation
  - No aggregation in definition
  - Inserts have NULL values for missing fields
  - Better if view definition includes primary key
- Non-Updatable views (SQL-99)
  - May be defined on multiple tables
- **Messy issue in general**

Summary of Logical Data Indep

- SQL views

- Beyond Logical data indep
  - An index is a view (why?)
  - Any query result can be stored, and considered a view
  - New problem: given views V1, V2, … find optimal way to compute a query Q

Outline

- Early data models
  - IMS
  - CODASYL
- Physical and logical independence in the relational model
- Data models that followed the relational model

Other Data Models

- Entity-Relationship: 1970's
  - Successful in logical database design (last lecture)
- Extended Relational: 1980's
- Semantic: late 1970's and 1980's
- Object-oriented: late 1980's and early 1990's
  - Address impedance mismatch: relational dbs ←→ OO languages
  - Interesting but ultimately failed (several reasons, see references)
- Object-relational: late 1980's and early 1990's
  - User-defined types, ops, functions, and access methods
- Semi-structured: late 1990's to the present

Semistructured vs Relational

- Relational data model
  - “Schema first”
- Semistructured data model: XML, Json, Protobuf
  - ”Schema last”
  - Hierarchical (trees)

XML Syntax

```
<article mdate="2011-01-11" key="journals/acta/GoodmanS83">
    <author>Nathan Goodman</author>
    <author>Oded Shmueli</author>
    <title>NP-complete Problems Simplified on Tree Schemas.</title>
    <pages>171-178</pages>
    <year>1983</year>
    <volume>20</volume>
    <journal>Acta Inf.</journal>
    <url>db/journals/acta/acta20.html#GoodmanS83</url>
    <ee>http://dx.doi.org/10.1007/BF00289414</ee>
</article>
```

Semistructured, self-describing schema

JSon

Example from: http://www.jsonexample.com/

```
myObject = {
   "first": "John",
   "last": "Doe",
   "salary": 70000,
   "registered": true,
   "interests": [ "Reading", “Biking”, "Hacking" ]
}
```

Semistructured, self-describing schema

Discussion

- Stonebraker (circa 1998)
  - "schema last" is a niche market
- Today (circa 2020)
  - Major vendors scramble to offer efficient schema discovery while ingesting Json
- Why? What changed?

Discussion

- Stonebraker (circa 1998)
  - "schema last" is a niche market
- Today (circa 2020)
  - Major vendors scramble to offer efficient schema discovery while ingesting Json
- Why? What changed?
  - Today datasets are available in text format, often in Json; ingest first, process later

NoSQL Data Model(s)

- Web boom in the 2000's created a scalability crises
  - Startup X grows from 10 customers to 10M customers overnight, how does the MySQL database grow?
- NoSQL answer:
  - "Shard" data, i.e. distribute AWS
  - Simple data mode: key/value pairs

Key-Value Pair Data Model

- **Data model**: (key,value) pairs
  - Key = string/integer, unique for the entire data
  - Value = can be anything (very complex object)

Key-Value Pair Data Model

- **Data model**: (key,value) pairs
  - Key = string/integer, unique for the entire data
  - Value = can be anything (very complex object)
- **Operations**
  - `get(key)`, `put(key,value)`
  - Operations on value not supported

Key-Value Pair Data Model

- **Data model**: (key,value) pairs
  - Key = string/integer, unique for the entire data
  - Value = can be anything (very complex object)
- **Operations**
  - `get(key)`, `put(key,value)`
  - Operations on value not supported
- **Distribution / Partitioning** – w/ hash function
  - No replication: key k is stored at server h(k)
  - 3-way replication: key k stored at h1(k),h2(k),h3(k)

Flights(fid, date, carrier, origin, dest, ...)
Carriers(cid, name)

Example

- How would you represent the Flights data as key, value pairs?

How does query processing work?

Flights(fid, date, carrier, origin, dest, ...)
Carriers(cid, name)

Example

- How would you represent the Flights data as key, value pairs?

- Option 1: key=fid, value=entire flight record

How does query processing work?

Flights(fid, date, carrier, origin, dest, ...)
Carriers(cid, name)

Example

- How would you represent the Flights data as key, value pairs?
- Option 1: key=fid, value=entire flight record
- Option 2: key=date, value=all flights that day

How does query processing work?

Flights(fid, date, carrier, origin, dest, ...)
Carriers(cid, name)

Example

- How would you represent the Flights data as key, value pairs?
- Option 1: key=fid, value=entire flight record
- Option 2: key=date, value=all flights that day
- Option 3: key=(origin,dest), value=all flights between

How does query processing work?

No physical data independence!

Conclusion

- Data independence is desirable
  - Both physical and logical
  - People keep reinventing data models with predefined physical schema
  - Relational model is the only one that provides independence
- Query optimizer:
  - Is the critical piece that ensures performance with physical independence