Introduction to Data Management CSE 344

Lecture 10: Relational Algebra Wrap-up

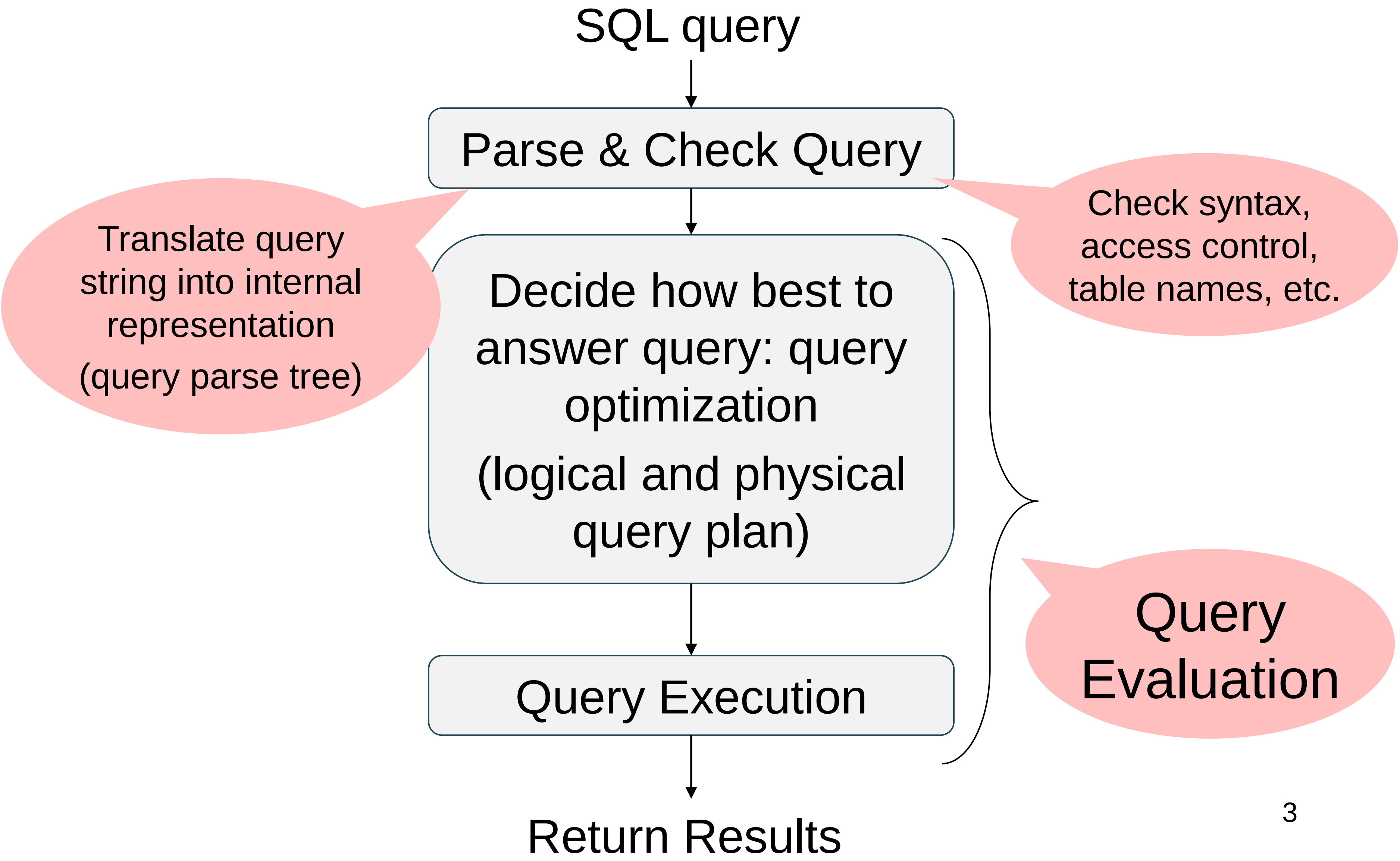
Systems Architecture

Announcements

- WQ4 is due next Tuesday
- HW3 is due next Thursday

- Today's lecture: How DBMSs work
  - Relational algebra and query execution
    - 2.4, 5.1, 16.2-16.3
    - (Optional) Chapter 15, more in CSE 444
  - Client-server architecture
    - 9.1

Query Evaluation Steps



SQL query

Parse & Check Query

Check syntax, access control, table names, etc.

Translate query string into internal representation

(query parse tree)

Decide how best to answer query: query optimization

(logical and physical query plan)

Query Execution

Query Evaluation

Return Results

The WHAT and the HOW

- SQL = WHAT we want to get form the data
- Relational Algebra = HOW to get the data we want
- The passage from WHAT to HOW is called query optimization

Overview: SQL = WHAT

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

```
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’
```

It’s clear WHAT we want, unclear HOW to get it

Query Optimizer = HOW

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

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

1. Which (equivalent) logical plan is the most efficient?

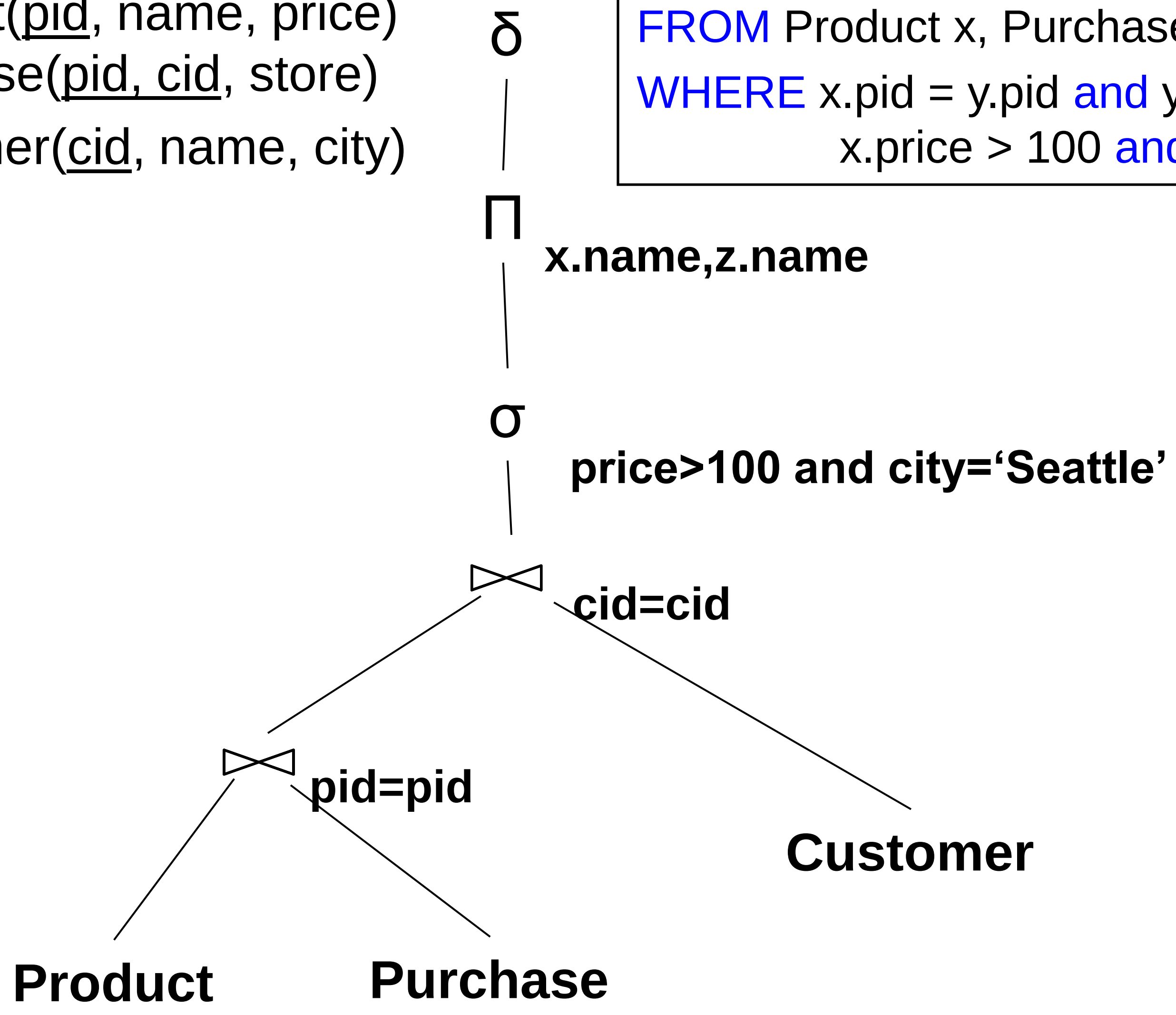
2. Physical plan:
   - How to implement each operation in the plan?
   - How to pass data from one operation to the other?

   (pipeline/on-the-fly, main-memory buffer, disk)

From SQL to RA

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

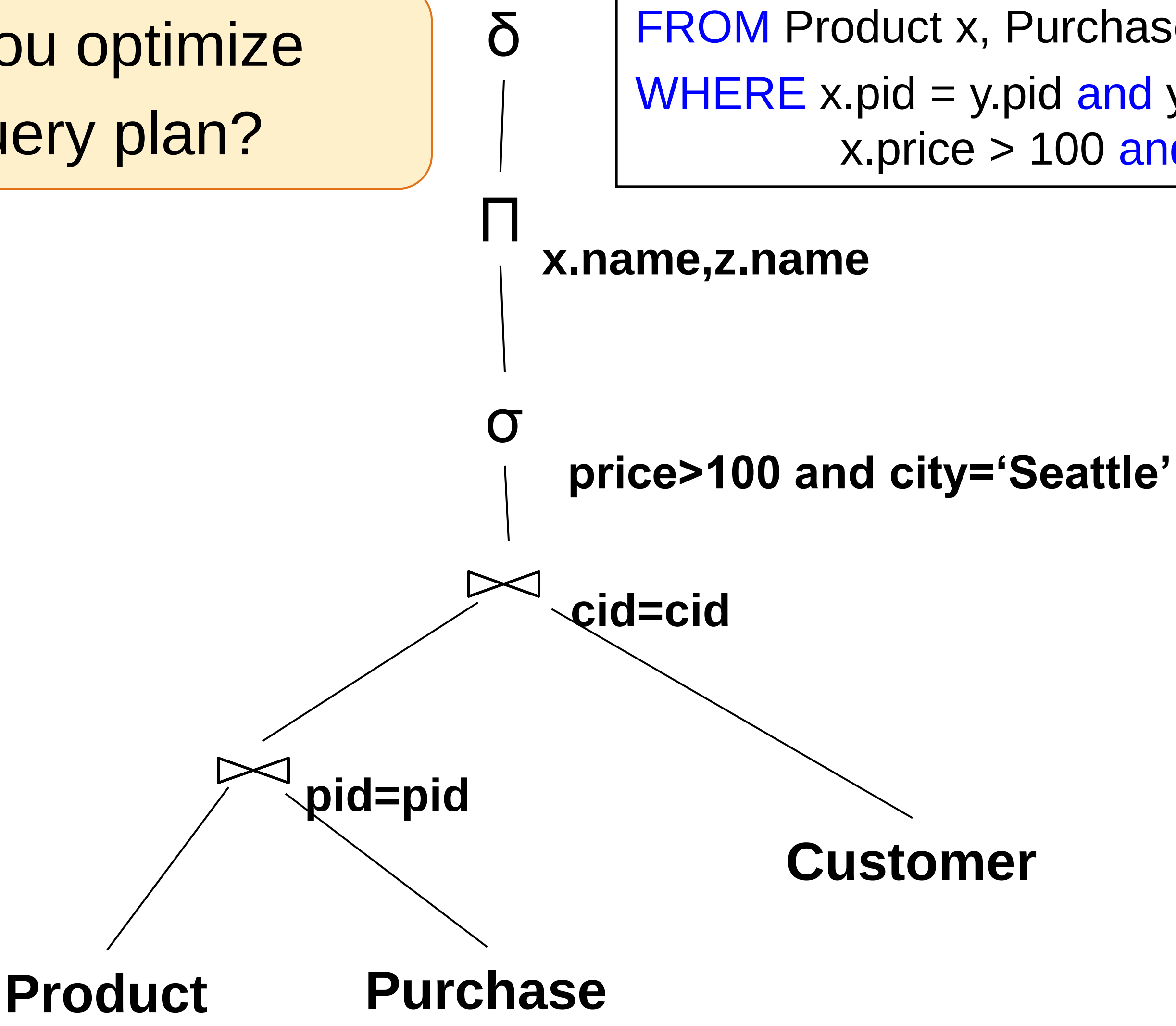
```
SELECT DISTINCT x.name, z.name
FROM Product x, Purchase y, Customer z
WHERE x.pid = y.pid and y.cid = z.cid and
            x.price > 100 and z.city = ‘Seattle’
```



From SQL to RA

Can you optimize this query plan?

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



δ

Π **x.name,z.name**

σ **price>100 and city=‘Seattle’**

$\bowtie$ **cid=cid**

$\bowtie$ **pid=pid**

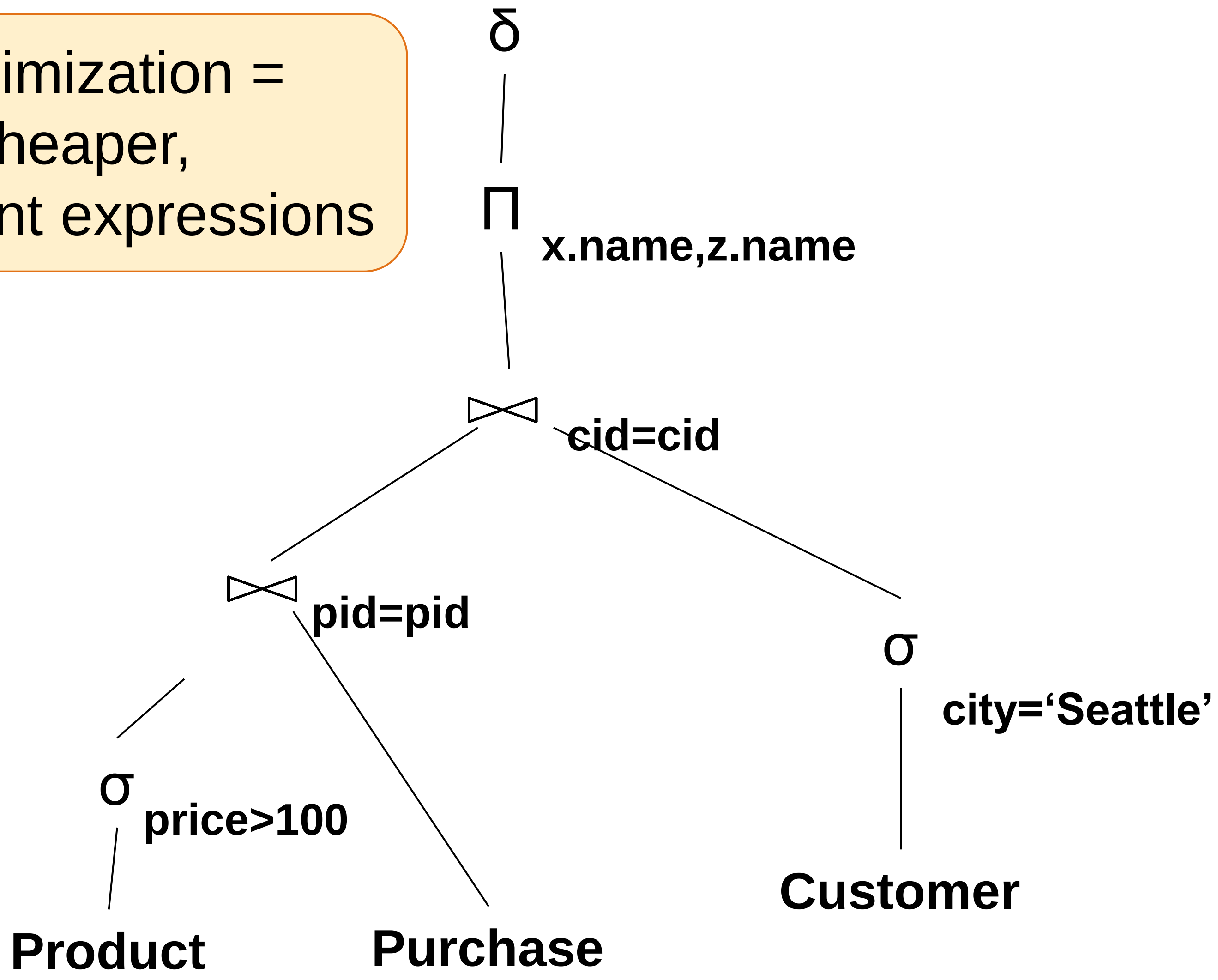
Product

Purchase

Customer

An Equivalent Expression

Query optimization = finding cheaper, equivalent expressions

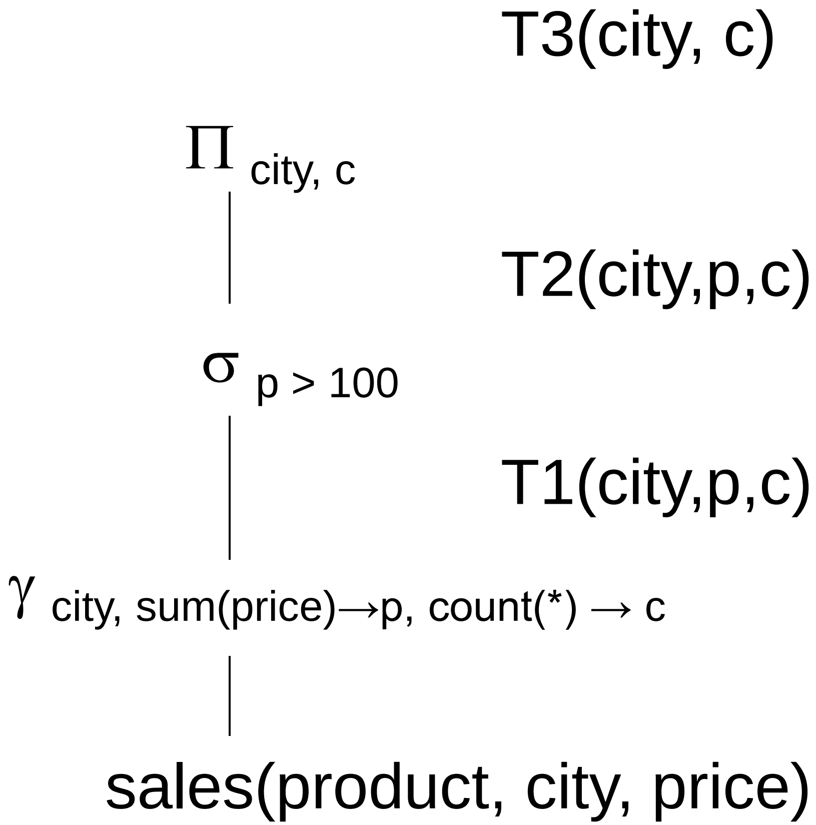


Extended RA: Operators on Bags

- Duplicate elimination δ
- Grouping γ
- Sorting τ

Logical Query Plan

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



T3(city, c)

$\Pi_{\text{city, c}}$

T2(city,p,c)

$\sigma_{\text{p > 100}}$

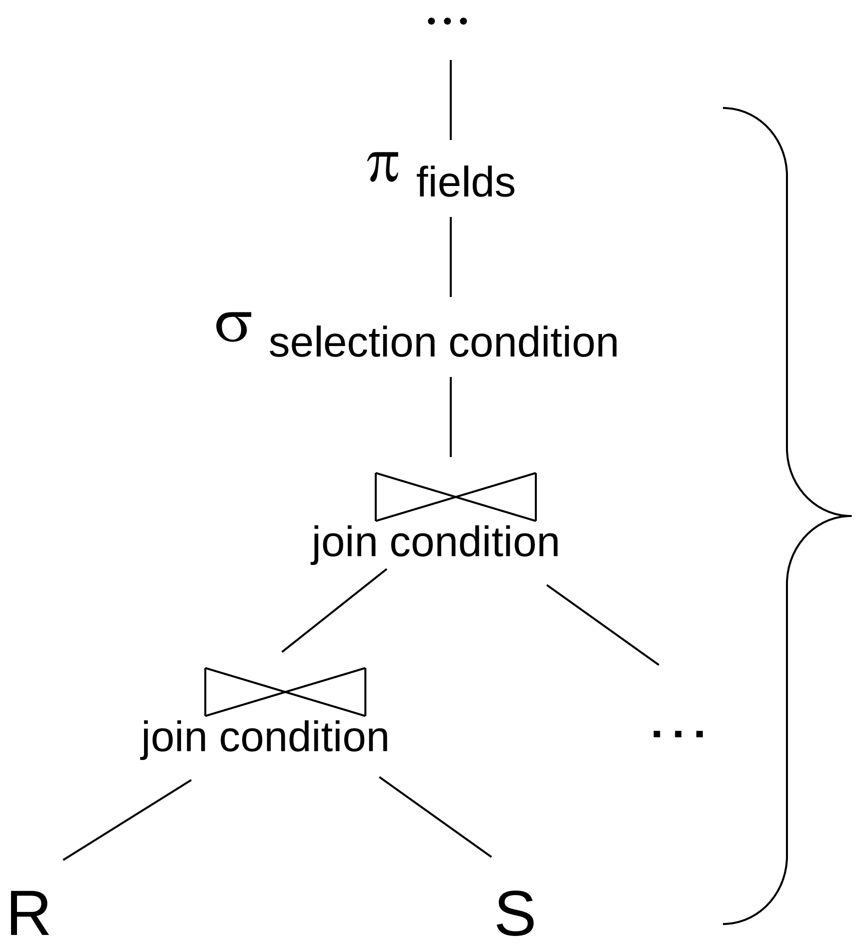
T1(city,p,c)

$\gamma_{\text{city, sum(price)}\rightarrow\text{p, count(*)} \rightarrow \text{c}}$

sales(product, city, price)

T1, T2, T3 = temporary tables

Typical Plan for Block (1/2)



...

π fields

σ selection condition

$\bowtie$ join condition

$\bowtie$ join condition

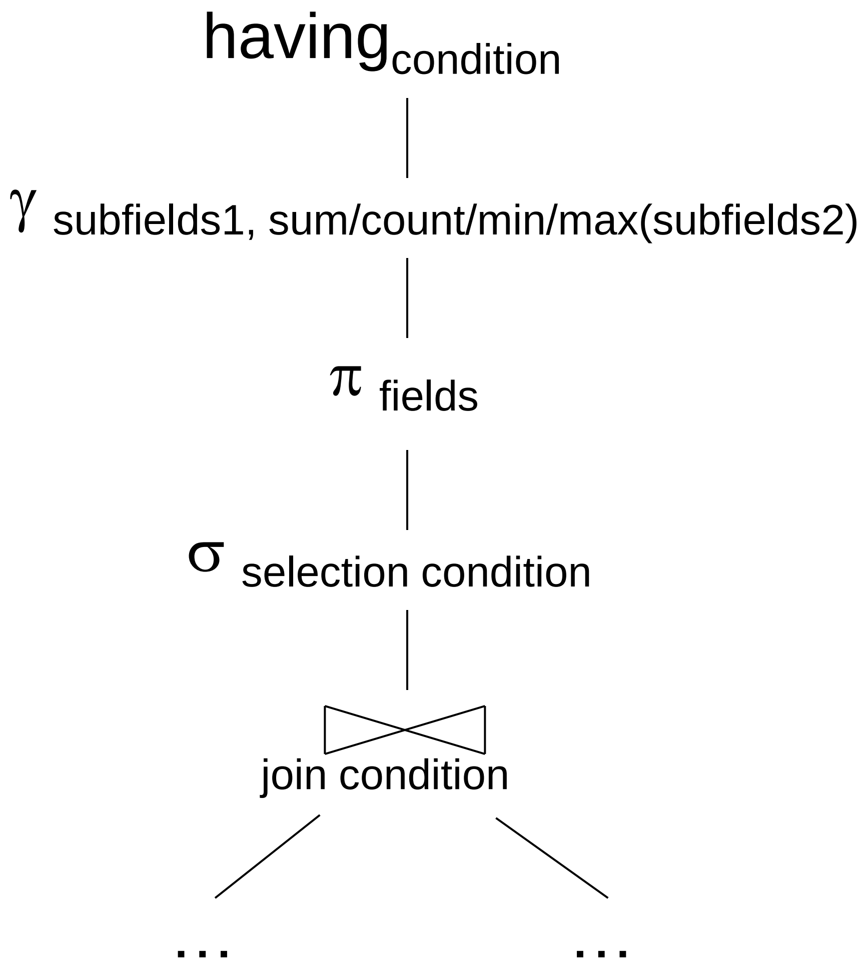
...

R

S

SELECT-PROJECT-JOIN Query

Typical Plan For Block (2/2)



$\text{having}_{\text{condition}}$

γ subfields1, sum/count/min/max(subfields2)

π fields

σ selection condition

$\bowtie$ join condition

…

…

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

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and not exists
       (SELECT *
        FROM Supply P
        WHERE P.sno = Q.sno
               and P.price > 100)
```

Sno of Suppliers
from WA
who did not
Supply a part
with price > 100

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

How about Subqueries?

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and not exists
      (SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
             and P.price > 100)
```

Correlation !

Sno of Suppliers
from WA
who did not
Supply a part
with price > 100

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

Let’s try to De-Correlate! (soln-1)

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and not exists
       (SELECT *
        FROM Supply P
        WHERE P.sno = Q.sno
               and P.price > 100)
```

De-Correlation

(Solution from class)

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and not in (SELECT P.sno
        FROM Supply P
        WHERE P.sno = Q.sno
               and P.price > 100)
```

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

Let’s try to De-Correlate! (soln-2)

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and not exists
       (SELECT *
        FROM Supply P
        WHERE P.sno = Q.sno
               and P.price > 100)
```

De-Correlation

(Solution from class)

```
SELECT  Q.sno
FROM Supplier Q, Supply P
WHERE  Q.sstate = ‘WA’
And Q.sno = P.sno
GROUP BY Q.sno
HAVING MAX(P.price) <= 100
```

How about Subqueries? (Decorrelation)

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

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and not exists
      (SELECT *
       FROM Supply P
       WHERE P.sno = Q.sno
             and P.price > 100)
```

De-Correlation

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and Q.sno not in
      (SELECT P.sno
       FROM Supply P
       WHERE P.price > 100)
```

How to model “not in” (nested)
Using RA operators?

How about Subqueries? (Un-nesting)

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

Un-nesting

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = ‘WA’)
    EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

```
SELECT  Q.sno
FROM Supplier Q
WHERE  Q.sstate = ‘WA’
    and Q.sno not in
       (SELECT P.sno
        FROM Supply P
        WHERE P.price > 100)
```

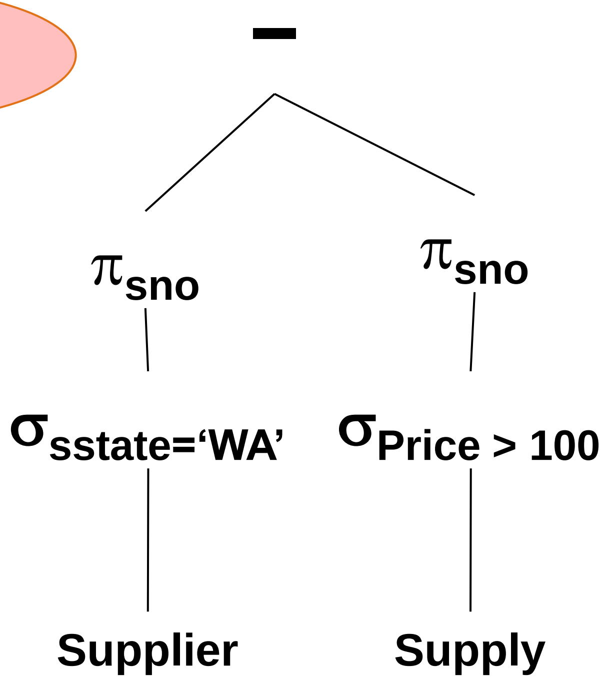
EXCEPT = set difference

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

Conversion to RA, finally!

```
(SELECT  Q.sno
 FROM Supplier Q
 WHERE  Q.sstate = ‘WA’)
   EXCEPT
 (SELECT P.sno
   FROM Supply P
   WHERE P.price > 100)
```

Finally…



From Logical Plans to Physical Plans

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

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’
```

Give a relational algebra expression for this query

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

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’
```

Give a relational algebra expression for this query

$\pi_{\text{--------}}(\sigma_{\text{----------------}}(\text{Supplier} \bowtie_{\text{-----------}} \text{Supply}))$

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

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’
```

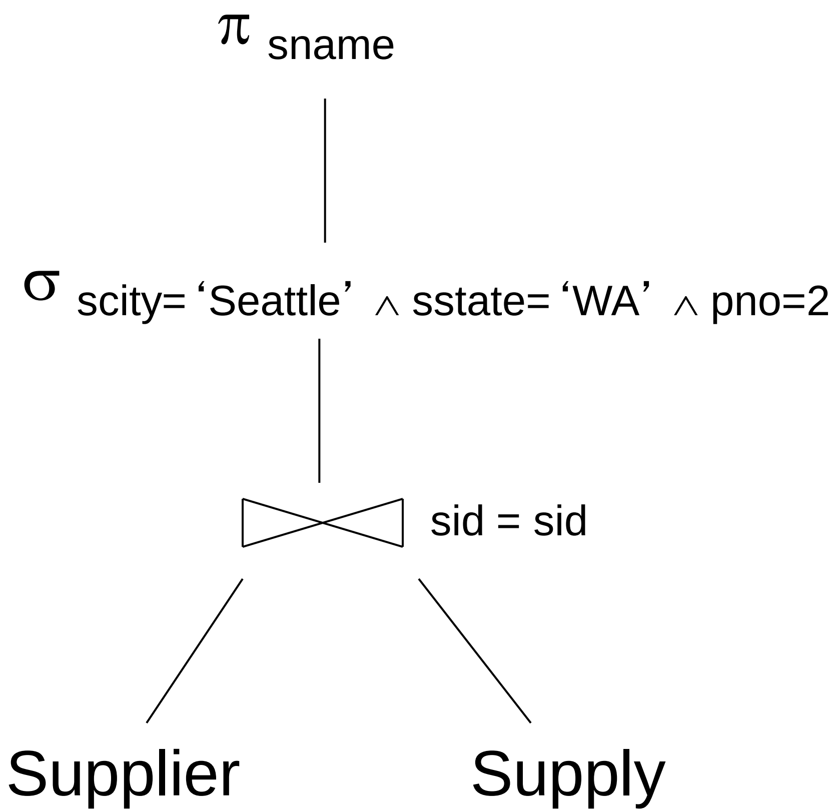
Give a relational algebra expression for this query

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

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Relational Algebra



π sname

σ scity= ‘Seattle’ $\wedge$ sstate= ‘WA’ $\wedge$ pno=2

$\bowtie$ sid = sid

Supplier

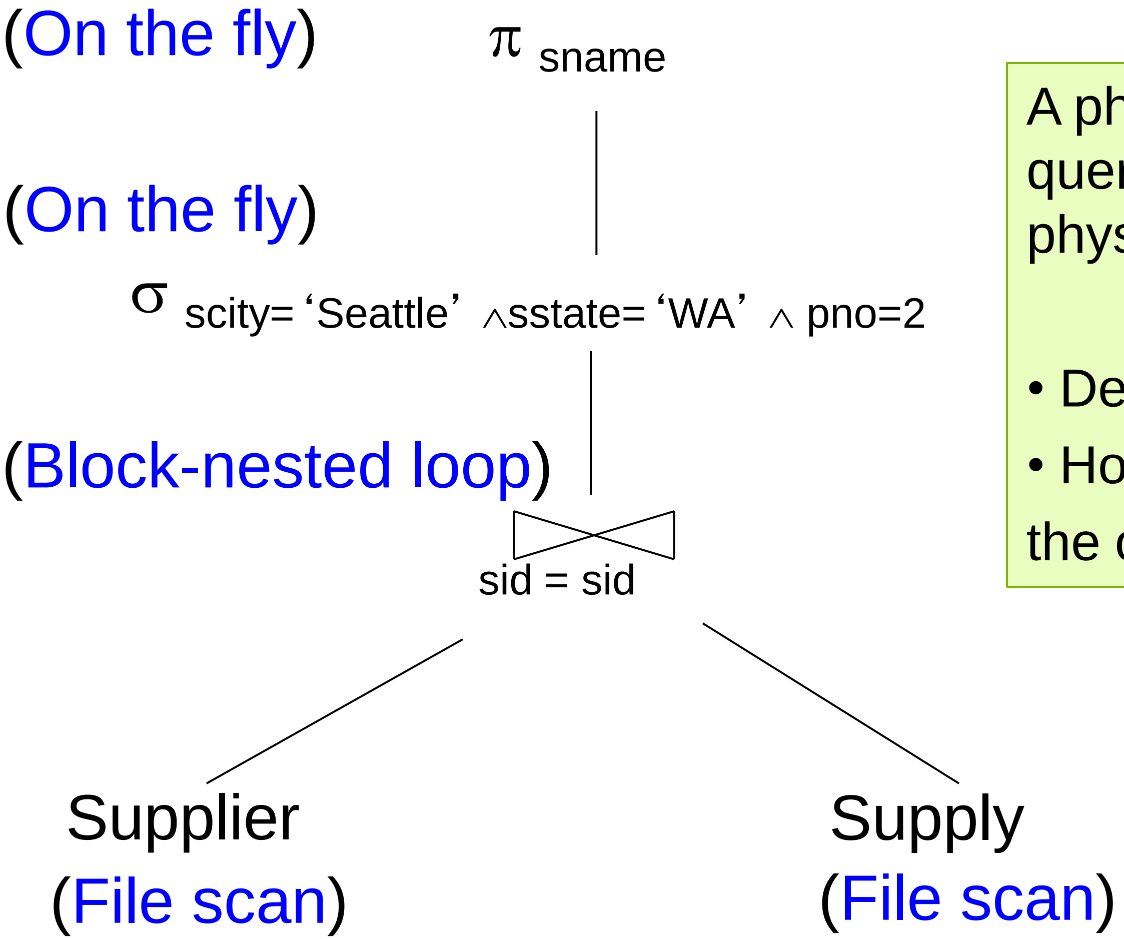
Supply

Relational algebra expression is also called the “logical query plan”

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

Supply(<u>sid, pno</u>, quantity)

Physical Query Plan 1



(On the fly) π_{sname}

(On the fly) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'} \wedge \text{pno=2}}$

(Block-nested loop) $\bowtie_{\text{sid = sid}}$

Supplier (File scan)

Supply (File scan)

A physical query plan is a logical query plan annotated with physical implementation details

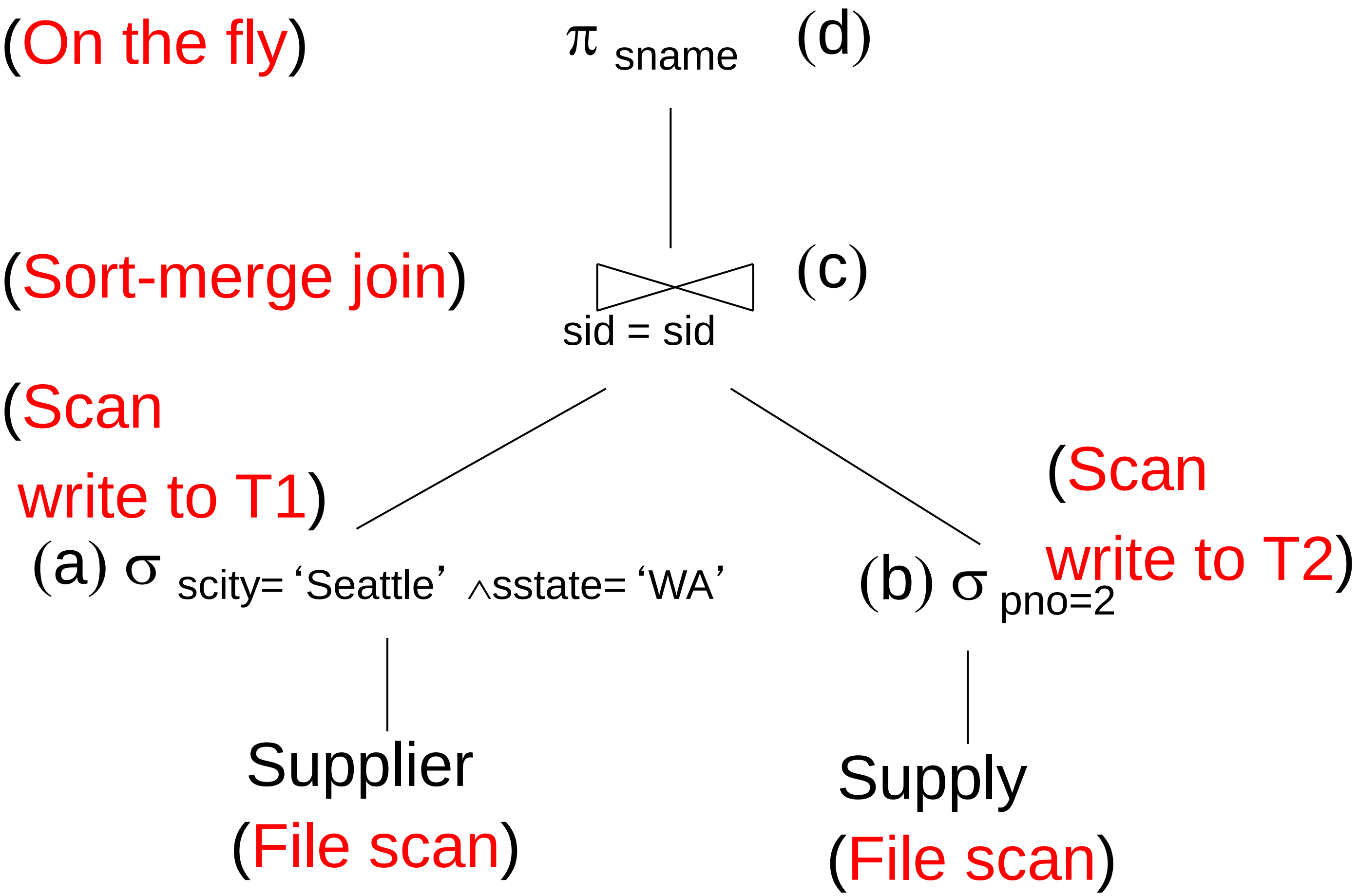
- Details of each operator
- How info is passed between the operators

Can you think of any Other types of Join algo? ☺

Supplier(sid, sname, scity, sstate)

Supply(sid, pno, quantity)

Physical Query Plan 2



(On the fly)

π_{sname} (d)

(Sort-merge join)

$\bowtie_{sid = sid}$ (c)

(Scan write to T1)

(a) $\sigma_{scity='Seattle' \wedge sstate='WA'}$

(Scan write to T2)

(b) $\sigma_{pno=2}$

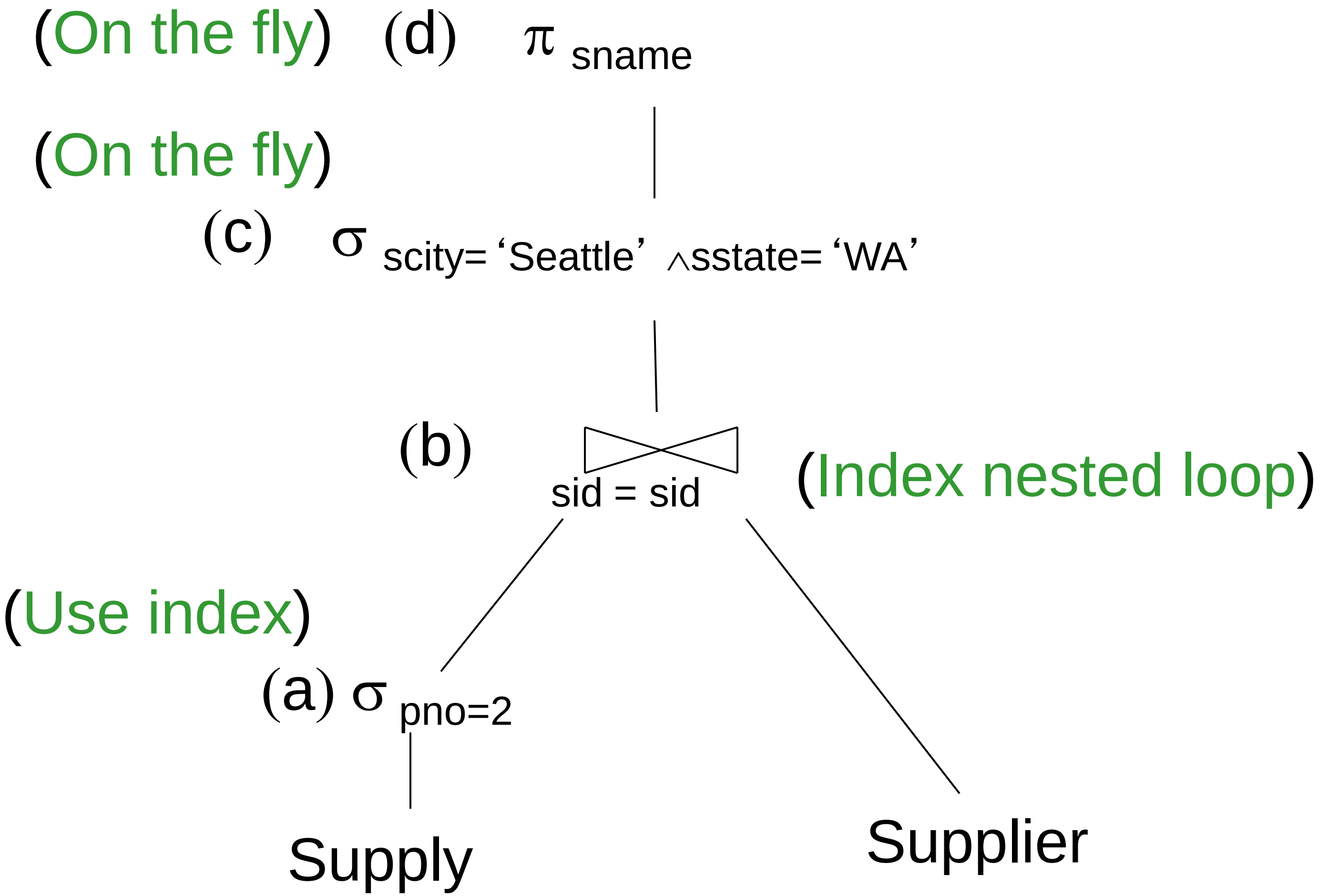
Supplier (File scan)

Supply (File scan)

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

Supply(<u>sid, pno</u>, quantity)

Physical Query Plan 3



(On the fly) (d) π_{sname}

(On the fly)

(c) $\sigma_{\text{scity='Seattle'} \wedge \text{sstate='WA'}}$

(b) $\bowtie_{\text{sid = sid}}$ (Index nested loop)

(Use index)

(a) $\sigma_{\text{pno=2}}$

Supply

Supplier

(Index lookup on pno)

(Index lookup on sid)

Why?

Assume: clustered

Doesn’t matter if clustered or not

Physical Data Independence

- Means that applications are insulated from changes in physical storage details
  - E.g., can add/remove indexes without changing apps
  - Can do other physical tunings for performance

- SQL and relational algebra facilitate physical data independence because both languages are “set-at-a-time”: Relations as input and output

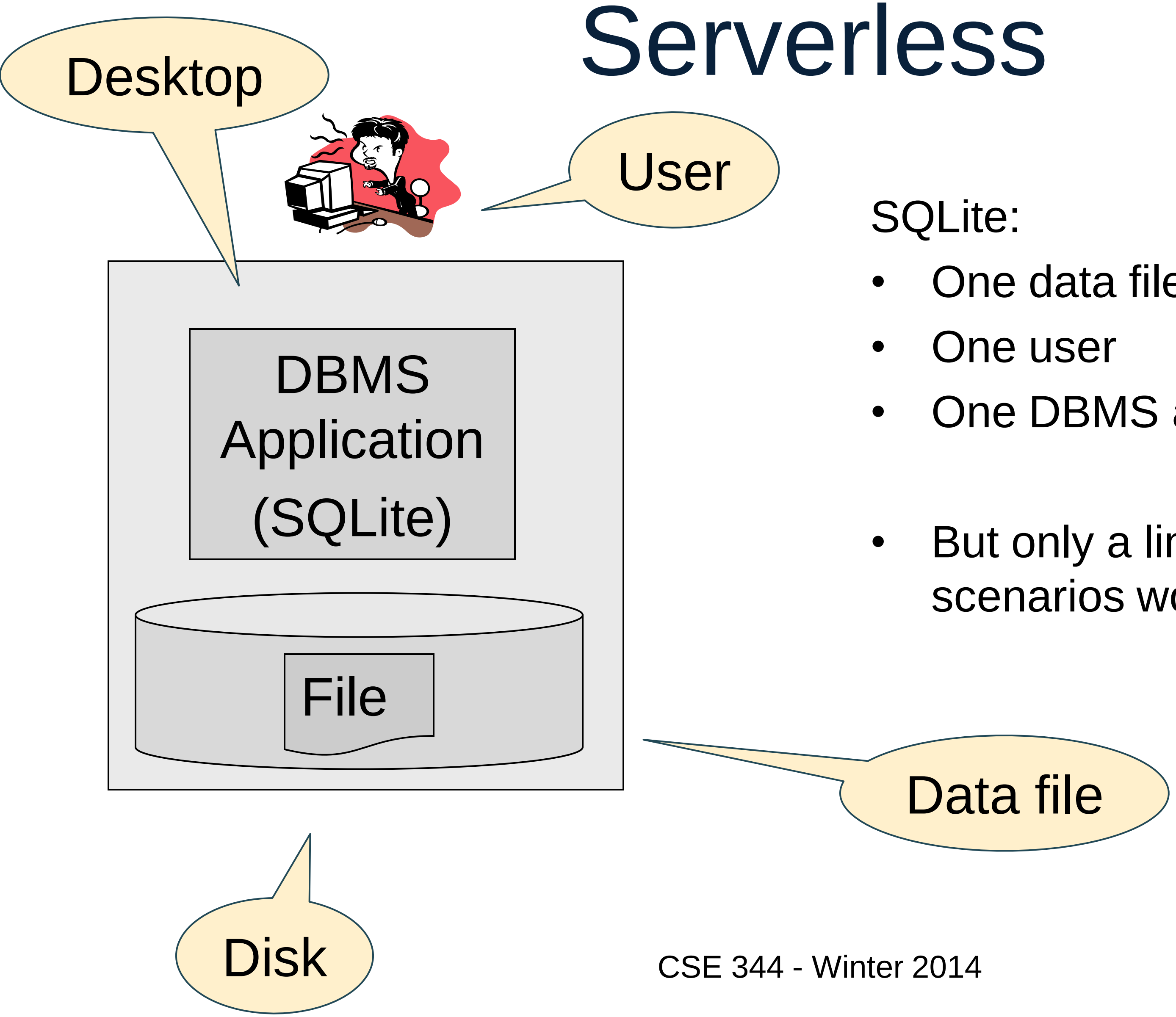
Architectures

1. Serverless

2. Two tier: client/server

3. Three tier: client/app-server/db-server

Serverless

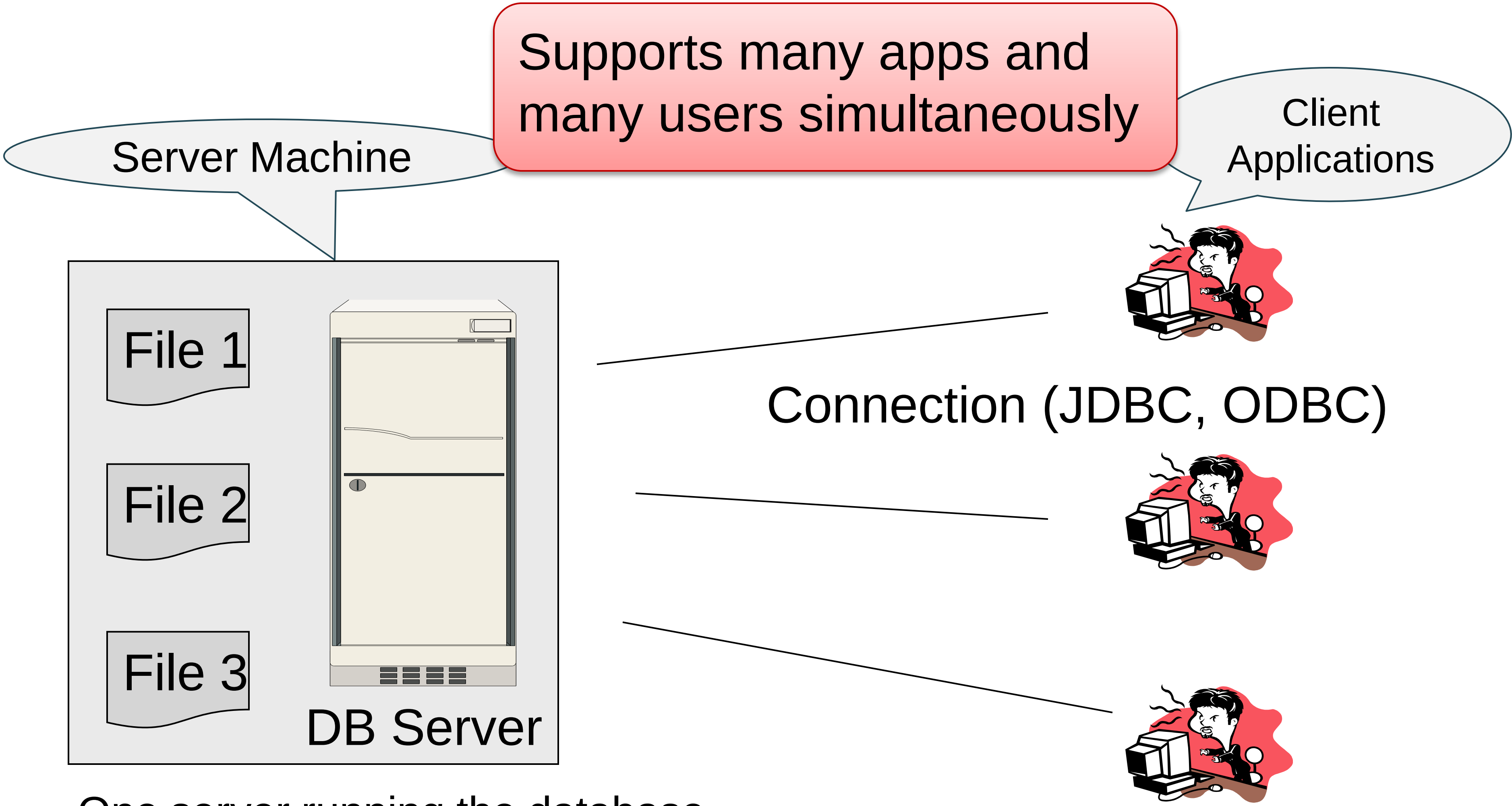


SQLite:

- One data file
- One user
- One DBMS application

- But only a limited number of scenarios work with such model

Client-Server


Supports many apps and many users simultaneously

Server Machine

Client Applications

File 1

File 2

File 3

DB Server

Connection (JDBC, ODBC)

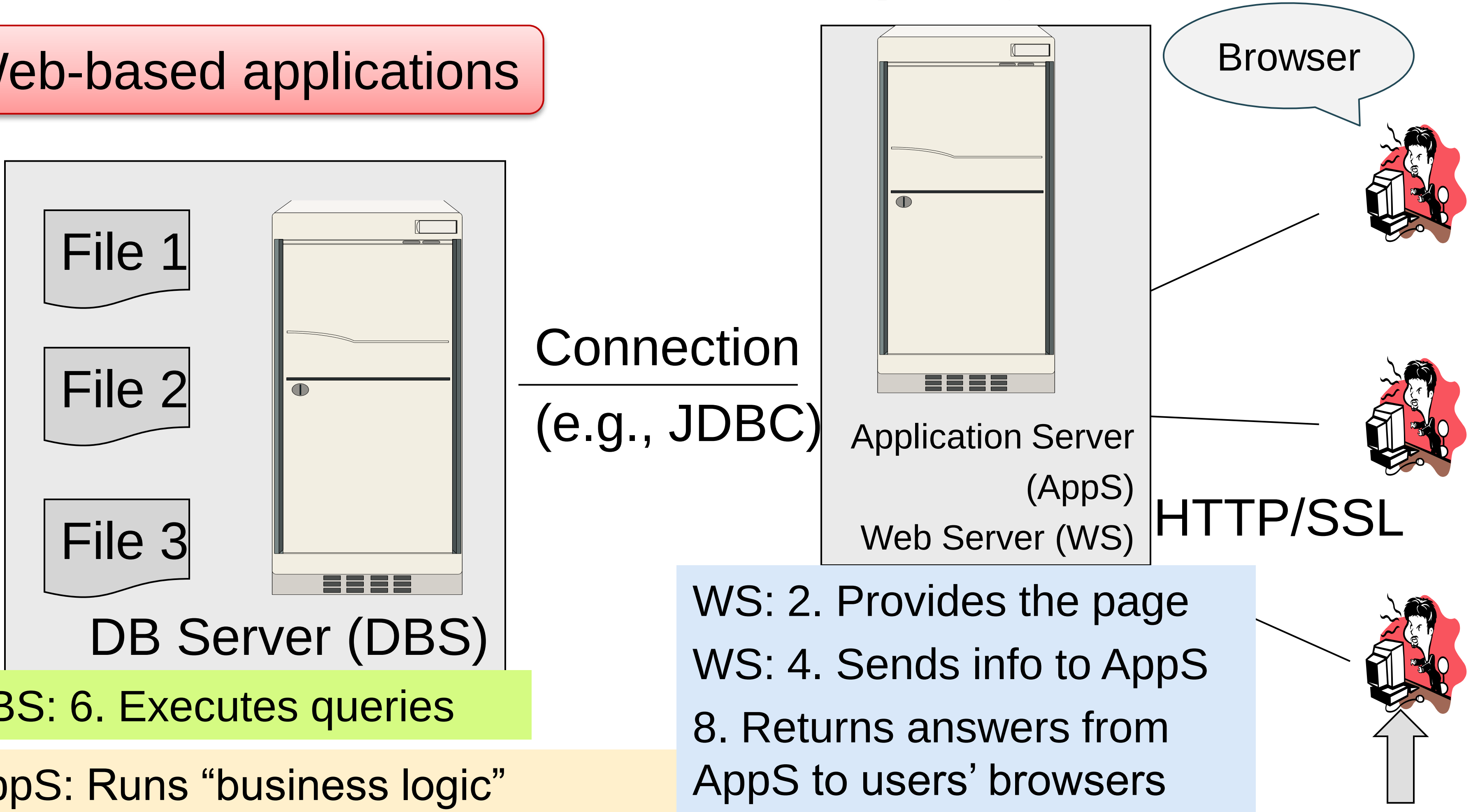
- One server running the database
- Many clients, connecting via the ODBC or JDBC (Java Database Connectivity) protocol

Client-Server

- One *server* that runs the DBMS (or RDBMS):
  - Your own desktop, or
  - Some beefy system, or
  - A cloud service (SQL Azure)
- Many *clients* run apps and connect to DBMS
  - Microsoft's Management Studio (for SQL Server), or
  - psql (for postgres)
  - Some Java program (HW5) or some C++ program
- Clients "talk" to server using JDBC/ODBC protocol

3-Tiers DBMS Deployment

Web-based applications



DBS: 6. Executes queries

AppS: Runs “business logic”

e.g. converts price from USD to EUR

5. Forms and asks queries to DBS,

7. Returns results to WS

WS: 2. Provides the page

WS: 4. Sends info to AppS

8. Returns answers from AppS to users’ browsers

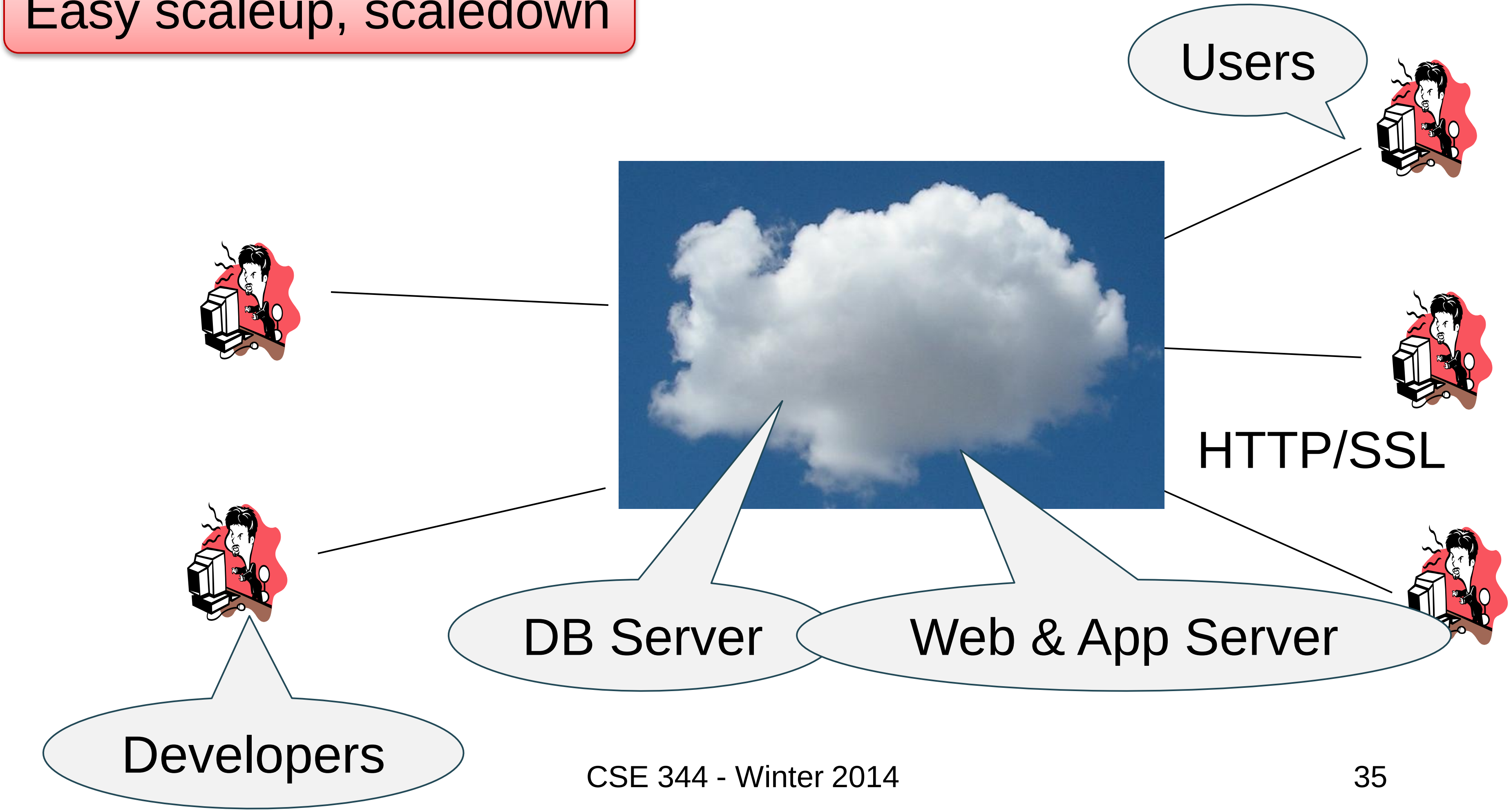
1. Open amazon.com

3. Search for books

9. Gets the results ☺

DBMS Deployment: Cloud

Easy scaleup, scaledown



Users

HTTP/SSL

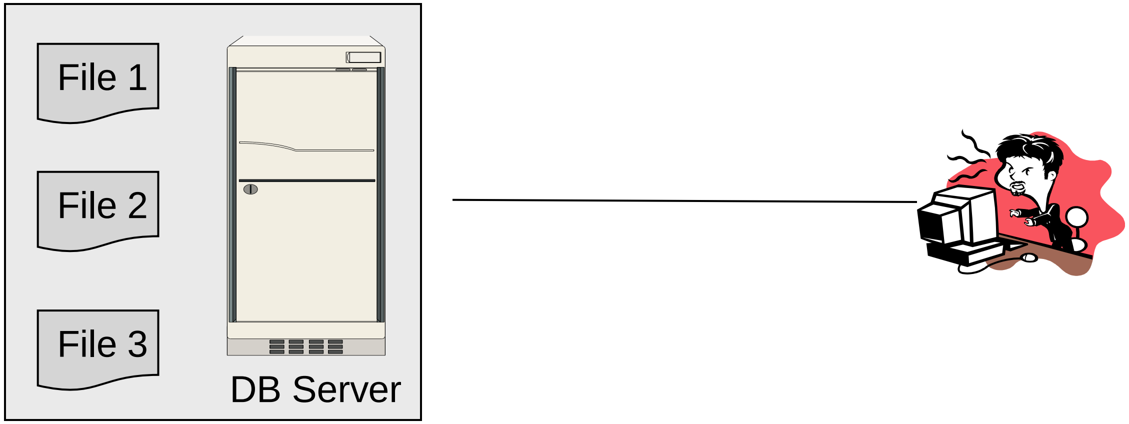
DB Server

Web & App Server

Developers

Using a DBMS Server

1. Client application establishes connection to server
2. Client must authenticate self
3. Client submits SQL commands to server
4. Server executes commands and returns results


File 1
File 2
File 3
DB Server