

CSE 444: Database Internals

Lectures 19-20
Parallel DBMSs

What We Have Already Learned

- Overall architecture of a DBMS
- Internals of query execution:
 - Data storage and indexing
 - Buffer management
 - Query evaluation including operator algorithms
 - Query optimization
- Internals of transaction processing:
 - Concurrency control: pessimistic and optimistic
 - Transaction recovery: undo, redo, and undo/redo

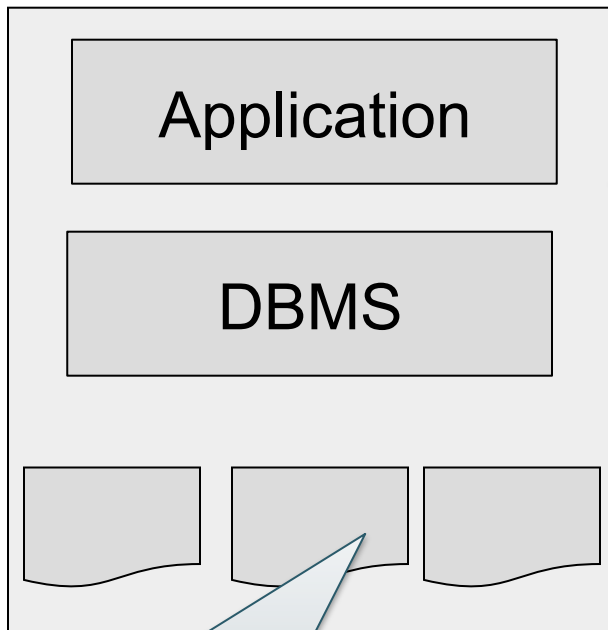
Where We Are Headed Next

- Scaling the execution of a query (this week)
 - Parallel DBMS
 - Distributed query processing
 - MapReduce
- Scaling transactions (next week)
 - Distributed transactions
 - Replication
- Scaling with NoSQL and NewSQL (in two weeks)

Reading Assignments

- Main textbook Chapter 20.1
- Database management systems.
Ramakrishnan&Gehrke.
Third Ed. Chapter 22.11
 - More info than our main textbook

DBMS Deployment: Local

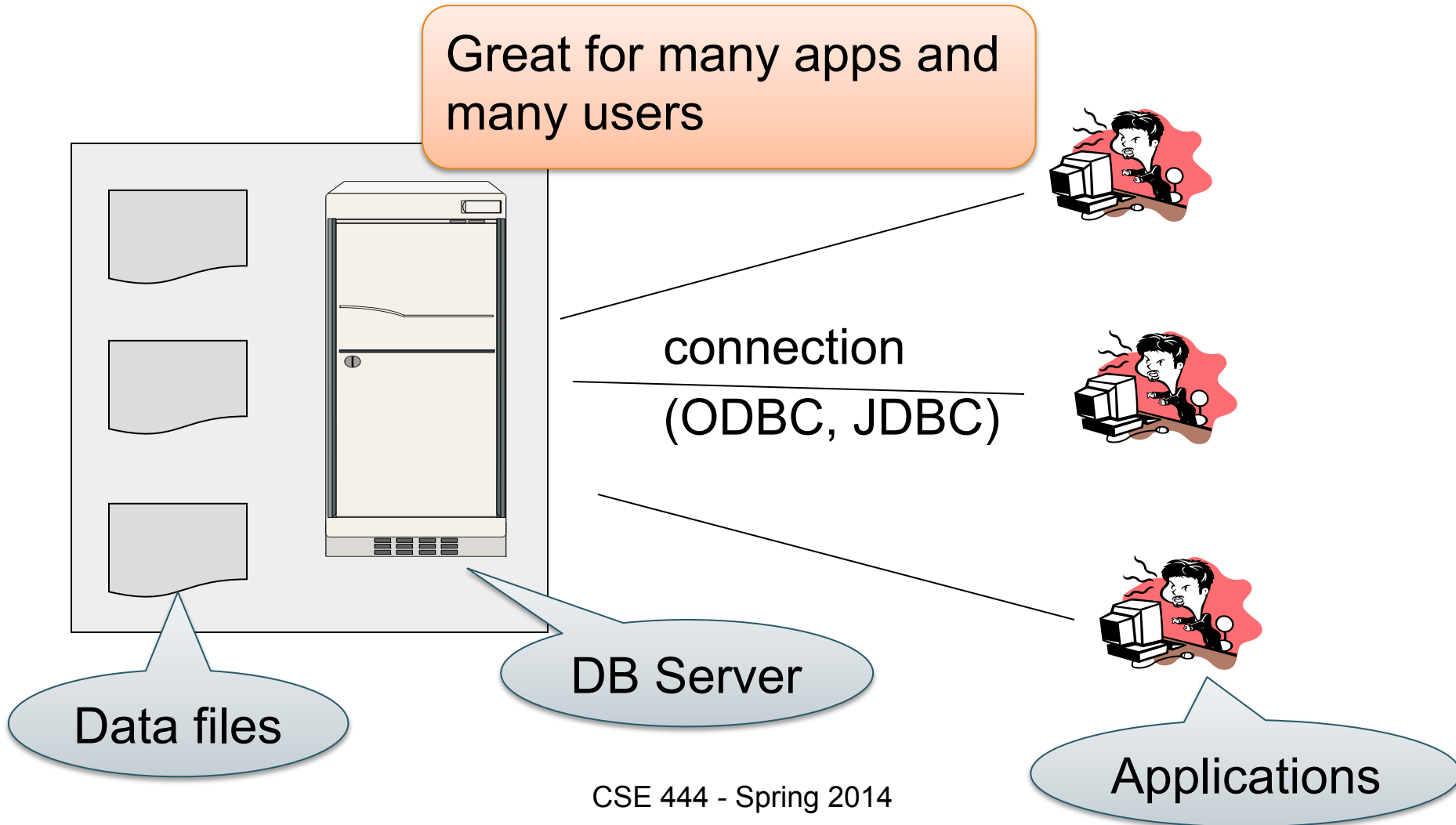


Great for one application (could be more) and one user.

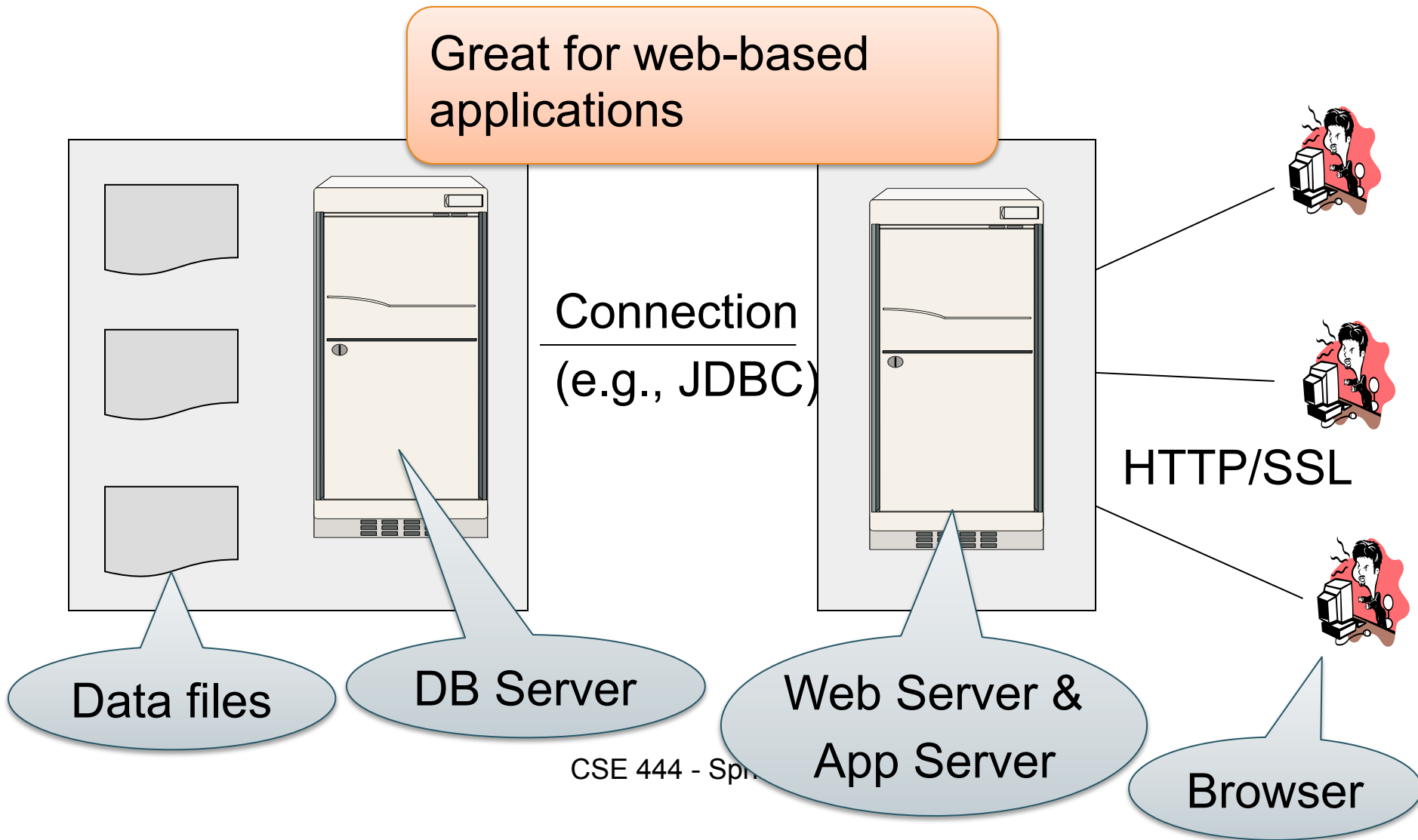
Desktop

Data files on disk

DBMS Deployment: Client/Server

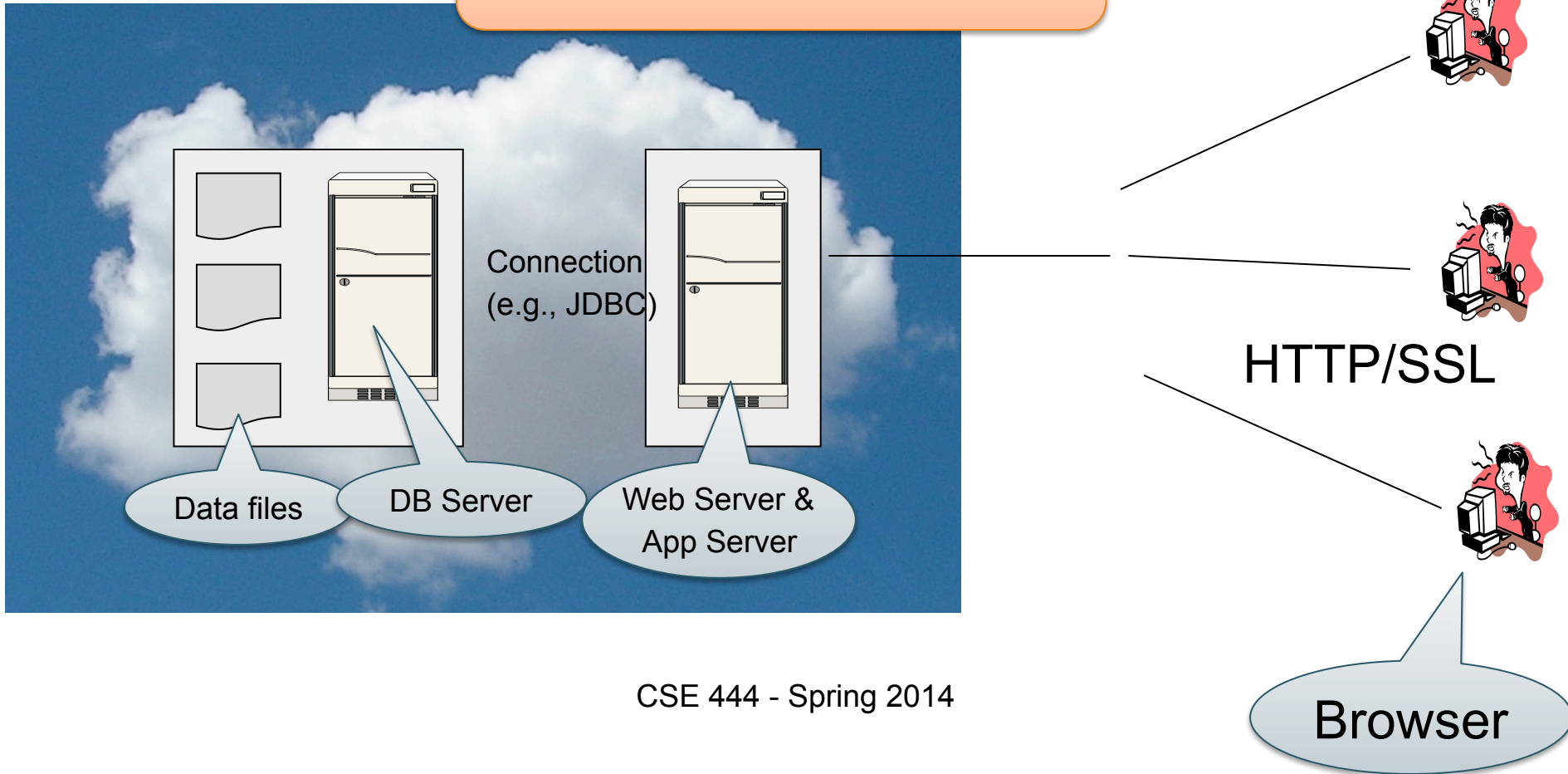


DBMS Deployment: 3 Tiers

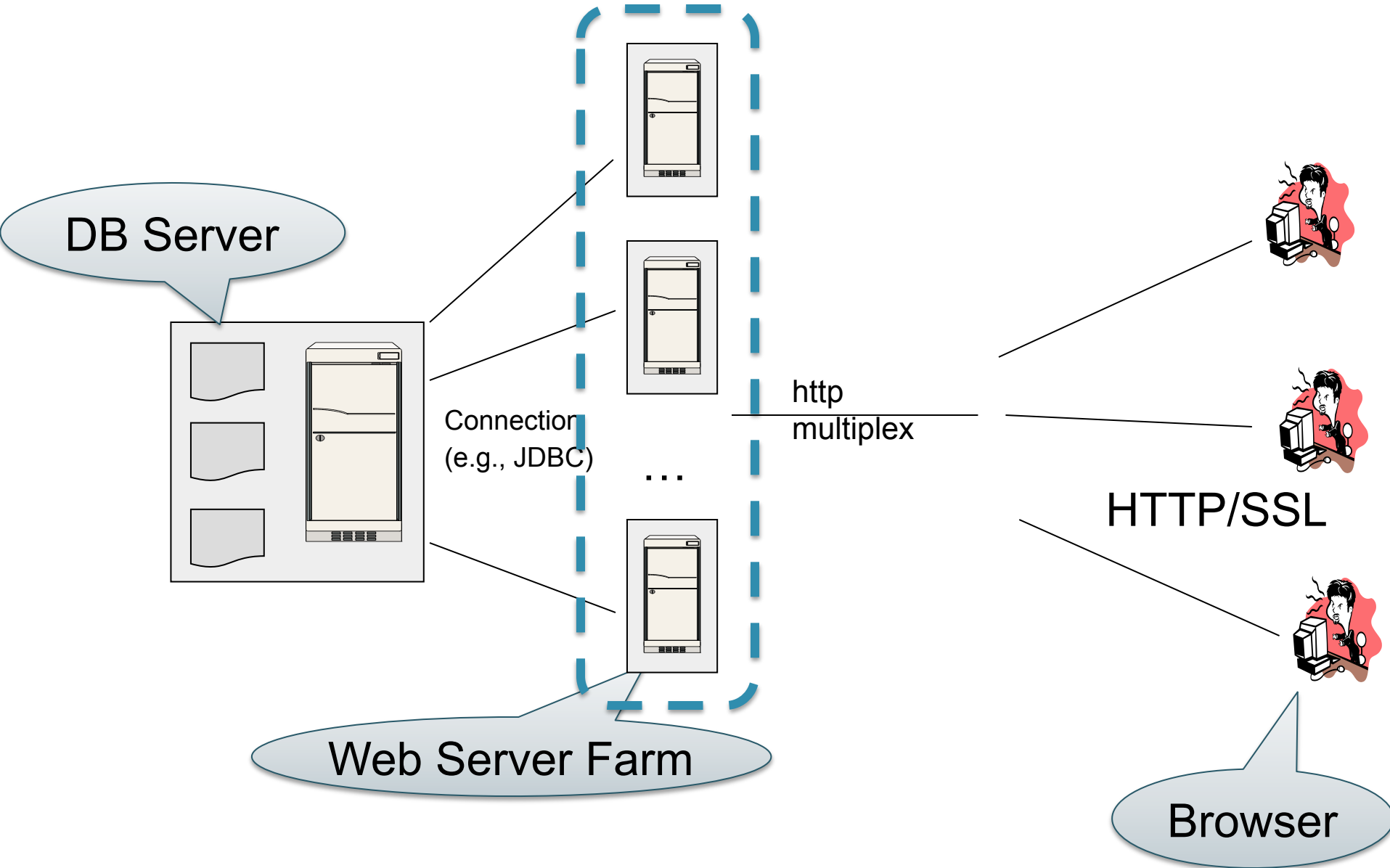


DBMS Deployment: Cloud

Great for web-based applications

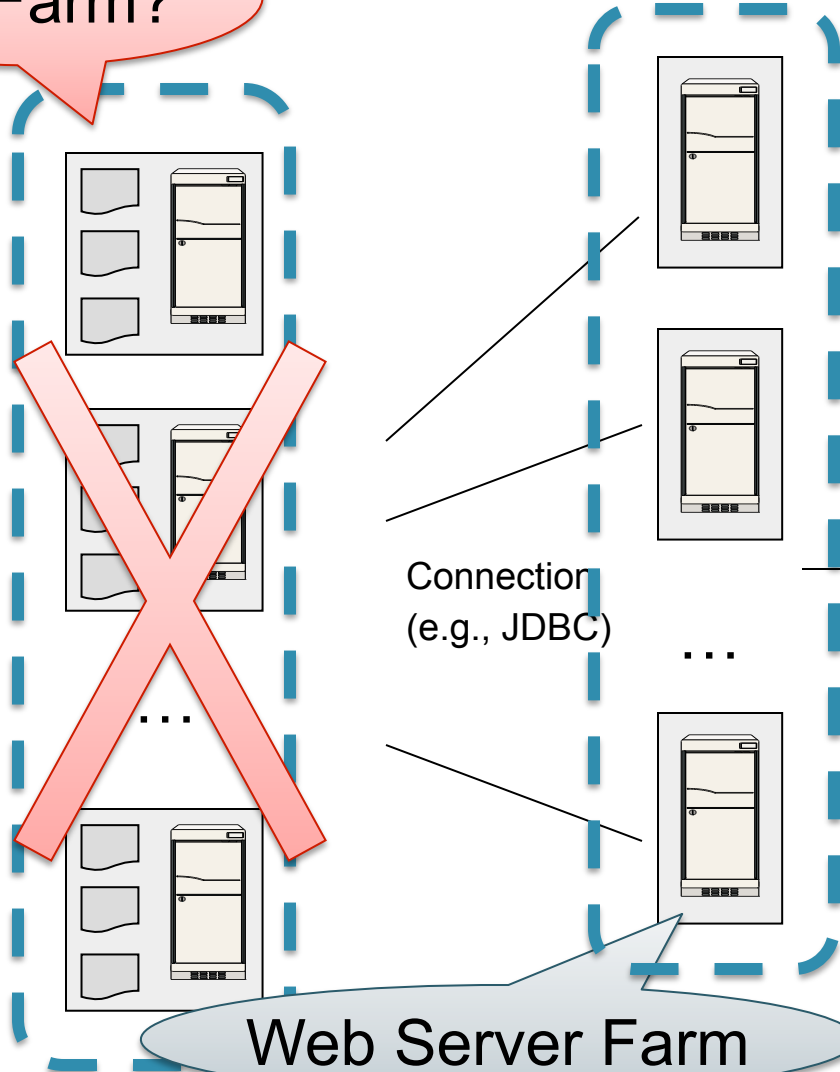


How to Scale?



How to Scale?

DB Farm?



http
multiplex

HTTP/SSL

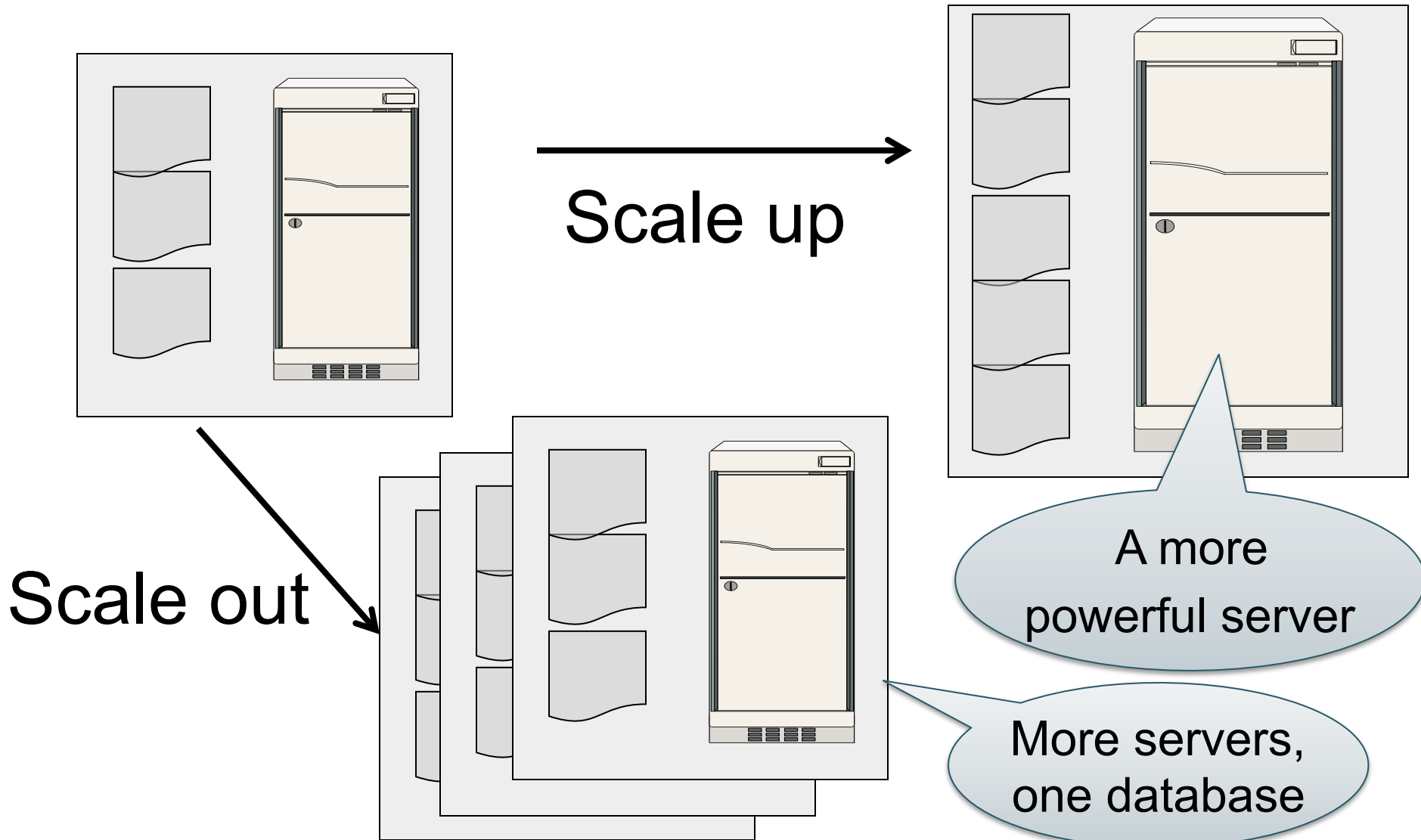
Browser



How to Scale?

- We can easily replicate the web servers and the application servers
- We cannot replicate the database servers, because the database is unique
- Instead we need to design ways to scale up the DBMS itself

How to Scale a DBMS?



What to scale?

- Transactions per second
- Query response time

Scaling Transactions Per Second

- Amazon
- Facebook
- Twitter
- ... your favorite Internet application...

- Goal is to scale OLTP workloads

- We will get back to this next week

Scaling Single Query Response Time

- Goal is to scale OLAP workloads
- That means the analysis of massive datasets

This Week: Focus on Scaling a Single Query

Big Data

- Buzzword?
- Definition from industry:
 - High Volume <http://www.gartner.com/newsroom/id/1731916>
 - High Variety
 - High Velocity

Big Data

Volume is not an issue

- Databases *do* parallelize easily; techniques available from the 80's
 - Data partitioning
 - Parallel query processing
- SQL is *embarrassingly parallel*

Big Data

New workloads are an issue

- Big volumes, small analytics
 - OLAP queries: join + group-by + aggregate
 - Can be handled by today's RDBMS
- Big volumes, big analytics
 - More complex Machine Learning, e.g. click prediction, topic modeling, SVM, k-means
 - Require massive computations outside RDBMS

Data Analytics Companies

Explosion of db analytics companies

- **Greenplum** founded in 2003 acquired by EMC in 2010; A parallel shared-nothing DBMS (this lecture)
- **Vertica** founded in 2005 and acquired by HP in 2011; A parallel, column-store shared-nothing DBMS (see 444 for discussion of column-stores)
- **DATAlegro** founded in 2003 acquired by Microsoft in 2008; A parallel, shared-nothing DBMS
- **Aster Data Systems** founded in 2005 acquired by Teradata in 2011; A parallel, shared-nothing, MapReduce-based data processing system (next lecture). SQL on top of MapReduce
- **Netezza** founded in 2000 and acquired by IBM in 2010. A parallel, shared-nothing DBMS.

Great time to be in the data management, data mining/statistics, or machine learning!

Two Approaches to Parallel Data Processing

- **Parallel databases**, developed starting with the 80s (this lecture)
 - For both **OLTP** (transaction processing)
 - And for **OLAP** (Decision Support Queries)
- **MapReduce**, first developed by Google, published in 2004 (next lecture)
 - Only for **Decision Support Queries**

Today we see convergence of the two approaches (Greenplum, BigQuery)

Parallel DBMSs

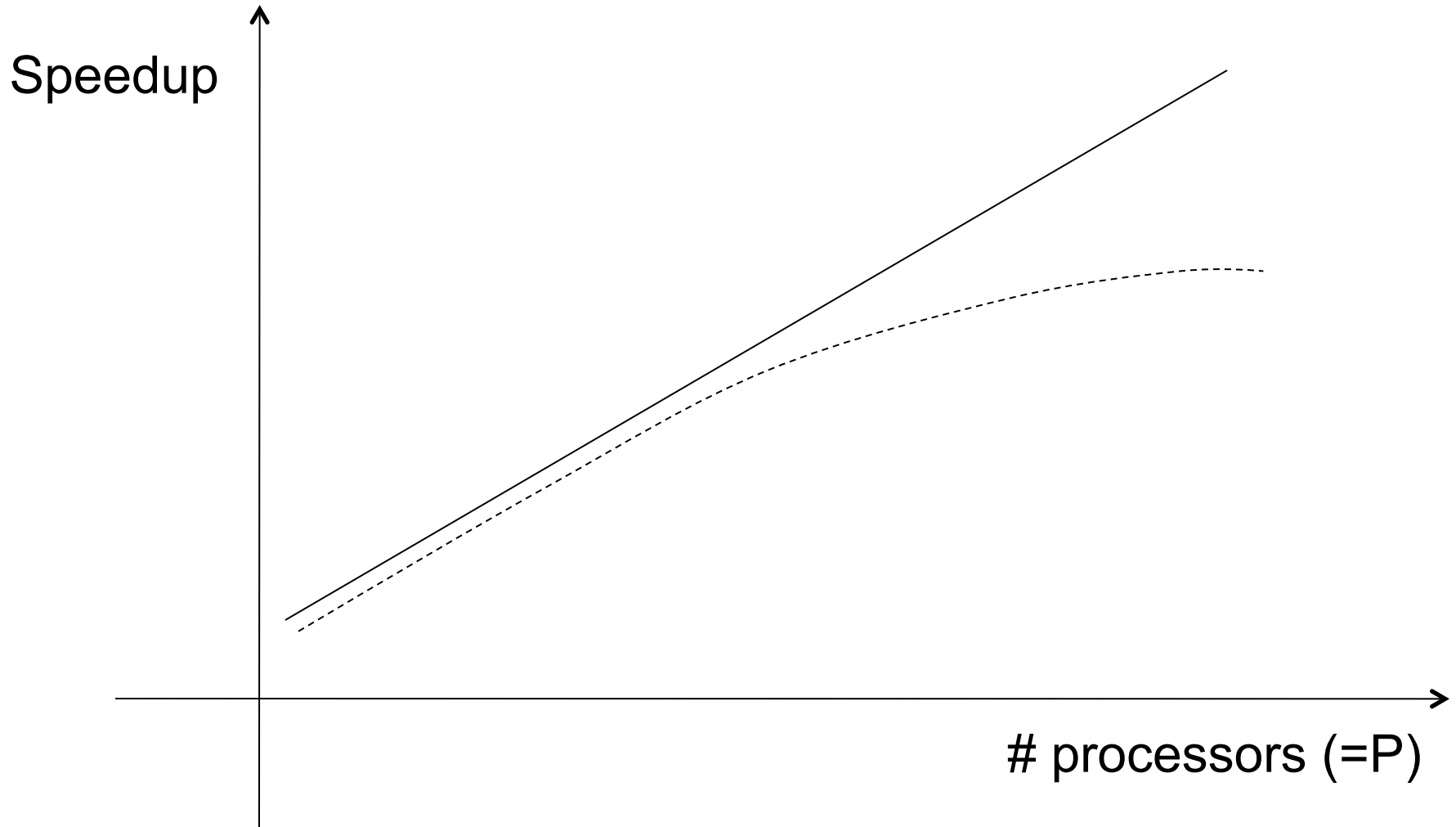
- **Goal**
 - Improve performance by executing multiple operations in parallel
- **Key benefit**
 - Cheaper to scale than relying on a single increasingly more powerful processor
- **Key challenge**
 - Ensure overhead and contention do not kill performance

Performance Metrics for Parallel DBMSs

Speedup

- More processors → higher speed
- Individual queries should run faster
- Should do more transactions per second (TPS)
- Fixed problem size *overall*, vary # of processors ("strong scaling")

Linear v.s. Non-linear Speedup

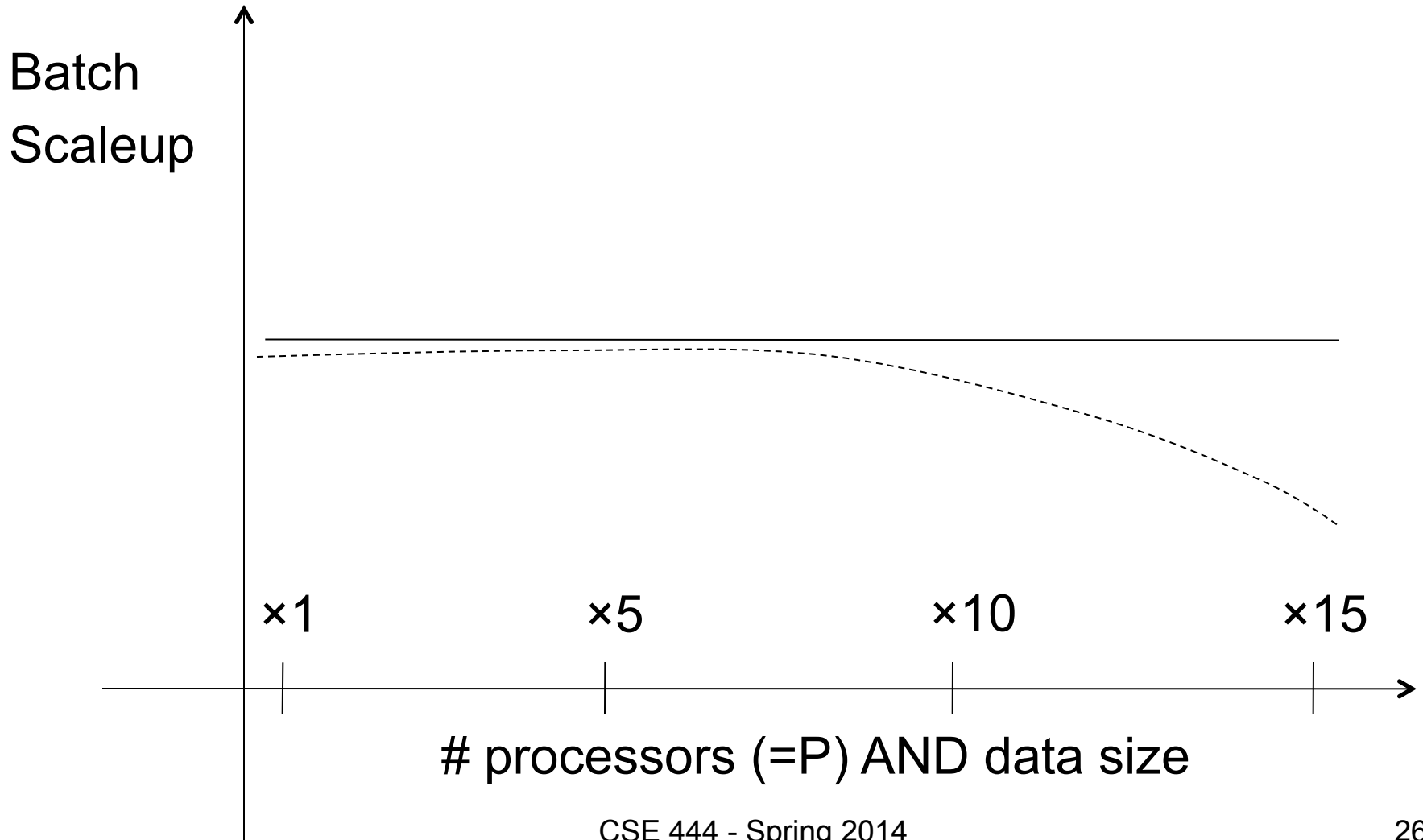


Performance Metrics for Parallel DBMSs

Scaleup

- More processors → can process more data
- Fixed problem size *per processor*, vary # of processors ("weak scaling")
- **Batch scaleup**
 - Same query on larger input data should take the same time
- **Transaction scaleup**
 - N-times as many TPS on N-times larger database
 - But each transaction typically remains small

Linear v.s. Non-linear Scaleup



Warning

- Be careful. Commonly used terms today:
 - “scale up” = use an increasingly more powerful server
 - “scale out” = use a larger number of servers

Challenges to Linear Speedup and Scaleup

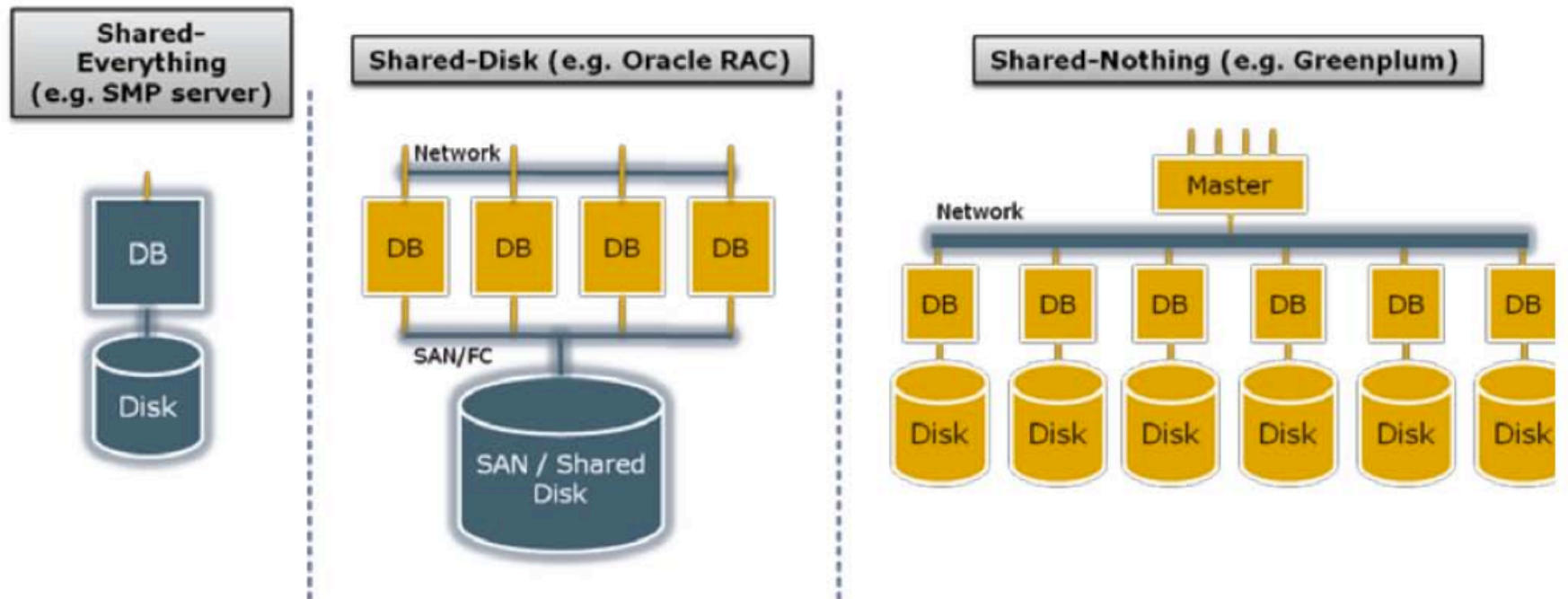
- **Startup cost**
 - Cost of starting an operation on many processors
- **Interference**
 - Contention for resources between processors
- **Skew**
 - Slowest processor becomes the bottleneck

Three Architectures for Parallel DB

- Shared memory
- Shared disk
- Shared nothing

Architectures for Parallel Databases

Figure 1 - Types of database architecture



From: Greenplum Database Whitepaper

SAN = "Storage Area Network"

Shared Memory

- Nodes share both RAM and disk
- Dozens to hundreds of processors

Example: SQL Server runs on a single machine and can leverage many threads to get a query to run faster (see query plans)

- Easy to use and program
- But very expensive to scale

Shared Disk

- All nodes access the same disks
- Found in the largest "single-box" (non-cluster) multiprocessors

Oracle dominates this class of systems

Characteristics:

- Also hard to scale past a certain point: existing deployments typically have fewer than 10 machines

Shared Nothing

- Cluster of machines on high-speed network
- Called "clusters" or "blade servers"
- Each machine has its own memory and disk: lowest contention.

NOTE: Because all machines today have many cores and many disks, then shared-nothing systems typically run many "nodes" on a single physical machine.

Characteristics:

- Today, this is the most scalable architecture.
- Most difficult to administer and tune.

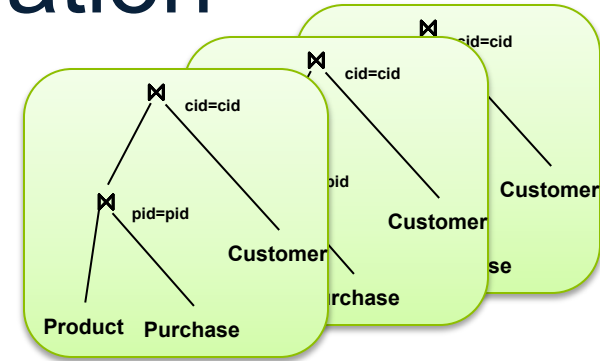
We discuss only Shared Nothing in class

In Class

- You have a parallel machine. Now what?
- How do you speed up your DBMS?

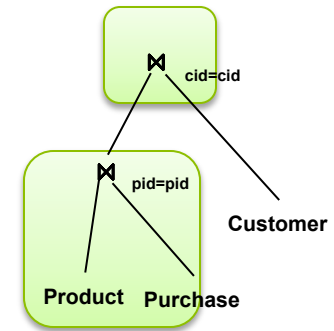
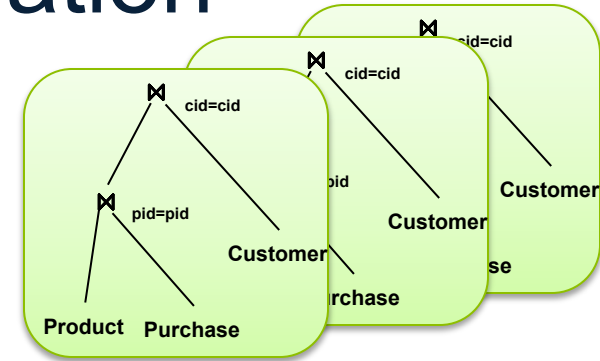
Taxonomy for Parallel Query Evaluation

- Inter-query parallelism
 - Each query runs on one processor



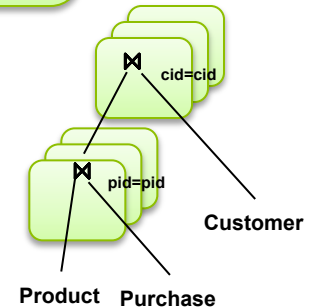
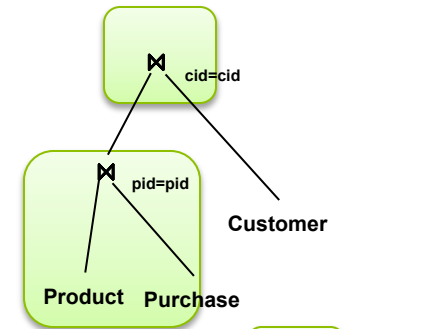
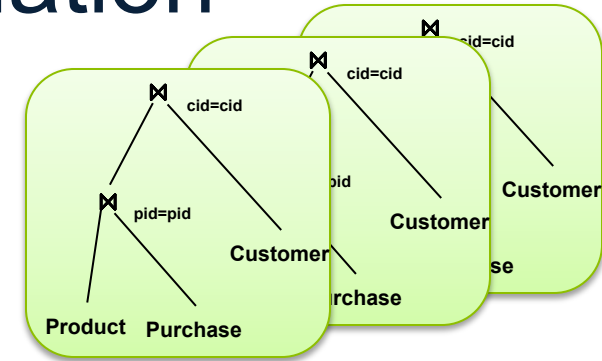
Taxonomy for Parallel Query Evaluation

- **Inter-query parallelism**
 - Each query runs on one processor
- **Inter-operator parallelism**
 - A query runs on multiple processors
 - An operator runs on one processor



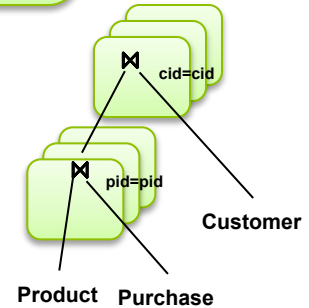
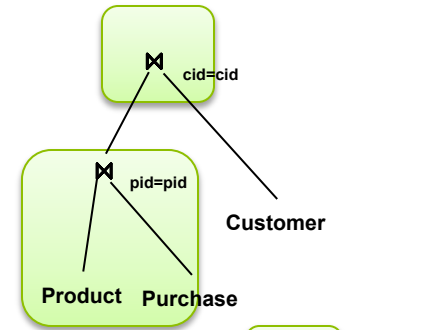
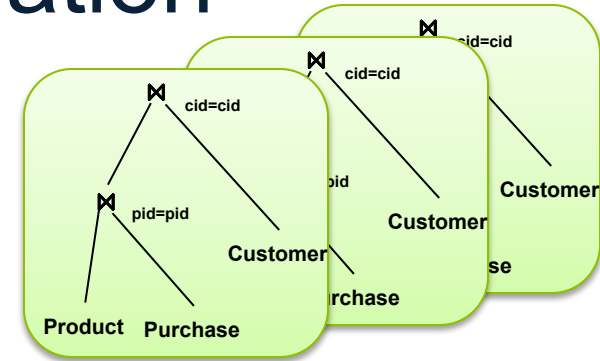
Taxonomy for Parallel Query Evaluation

- **Inter-query parallelism**
 - Each query runs on one processor
- **Inter-operator parallelism**
 - A query runs on multiple processors
 - An operator runs on one processor
- **Intra-operator parallelism**
 - An operator runs on multiple processors



Taxonomy for Parallel Query Evaluation

- **Inter-query parallelism**
 - Each query runs on one processor
- **Inter-operator parallelism**
 - A query runs on multiple processors
 - An operator runs on one processor
- **Intra-operator parallelism**
 - An operator runs on multiple processors



We study only intra-operator parallelism: most scalable

Parallel Query Processing

How do we **compute** these operations on a shared-nothing parallel db?

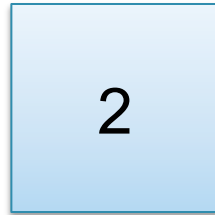
- **Selection:** $\sigma_{A=123}(R)$
- **Group-by:** $\gamma_{A, \text{sum}(B)}(R)$
- **Join:** $R \bowtie S$

Before we answer that: how do we **store** R (and S) on a shared-nothing parallel db?

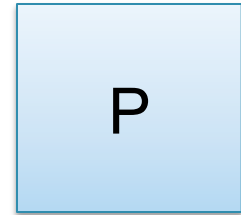
Horizontal Data Partitioning

Data:

Servers:



. . .



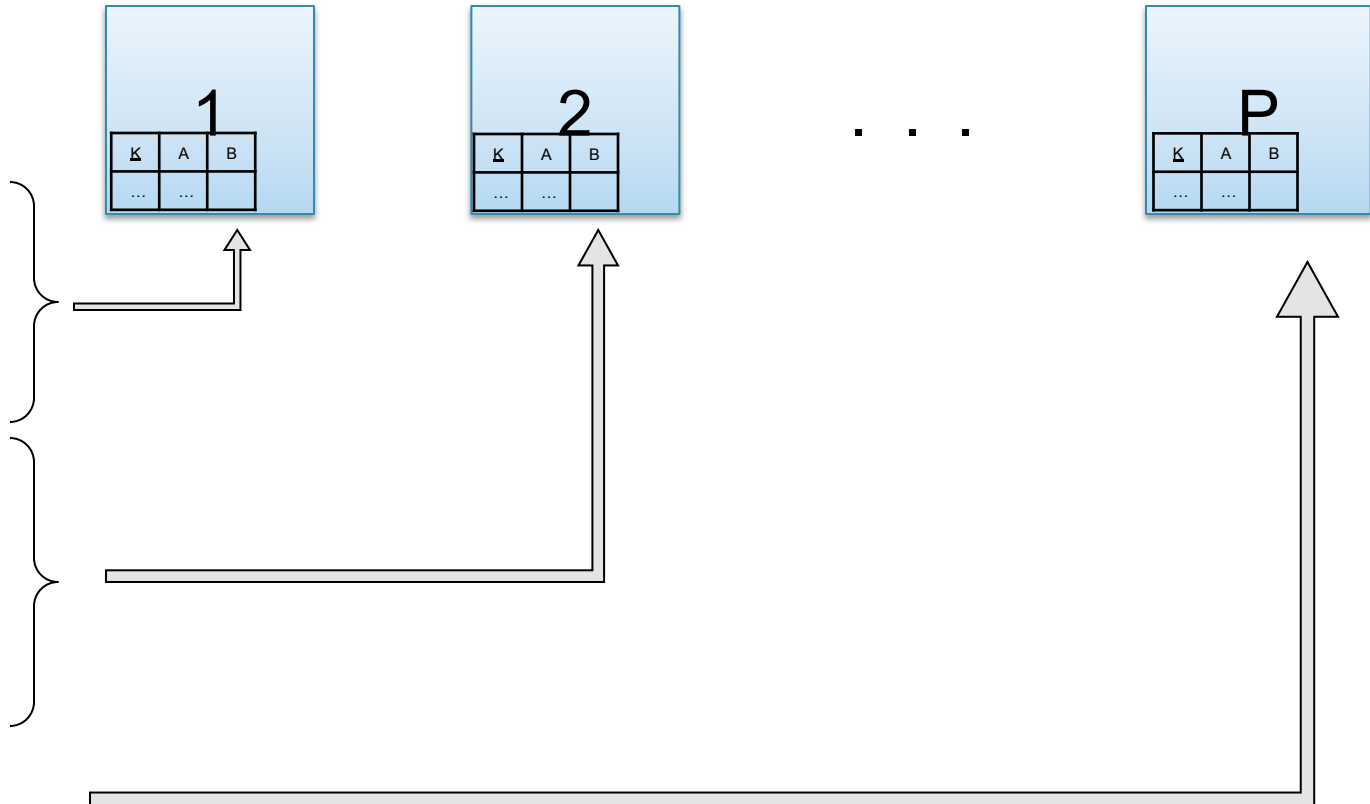
<u>K</u>	A	B
...	...	

Horizontal Data Partitioning

Data:

Servers:

<u>K</u>	A	B
...	...	

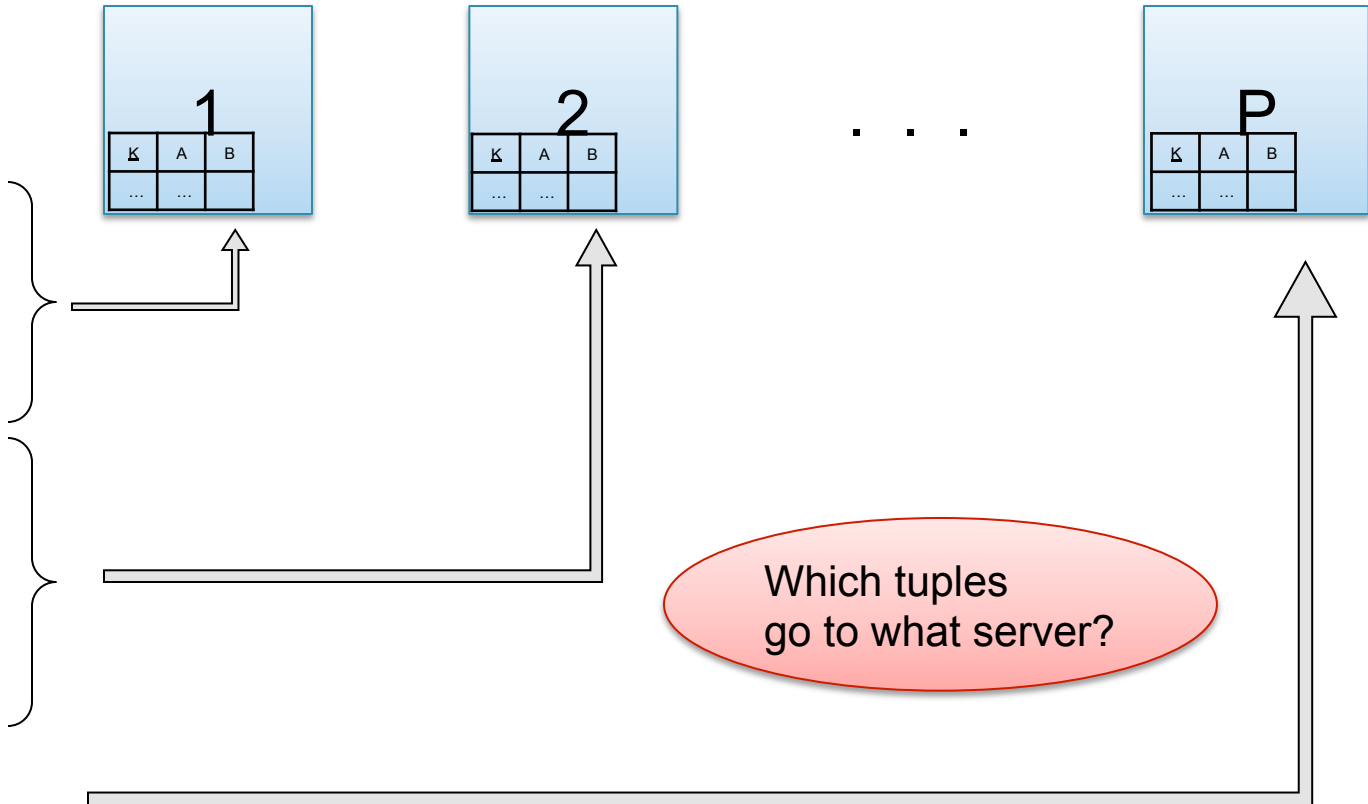


Horizontal Data Partitioning

Data:

Servers:

<u>K</u>	A	B
...	...	



Horizontal Data Partitioning

- Relation R split into P chunks R_0, \dots, R_{P-1} , stored at the P nodes
- **Block partitioned**
 - Each group of k tuples go to a different node
- **Hash based partitioning on attribute A :**
 - Tuple t to chunk $h(t.A) \bmod P$
- **Range based partitioning on attribute A :**
 - Tuple t to chunk i if $v_{i-1} < t.A < v_i$

Uniform Data v.s. Skewed Data

- Let $R(\underline{K}, A, B, C)$; which of the following partition methods may result in skewed partitions?
- Block partition
- Hash-partition
 - On the key K
 - On the attribute A
- Range-partition
 - On the key K
 - On the attribute A

Uniform Data v.s. Skewed Data

- Let $R(\underline{K}, A, B, C)$; which of the following partition methods may result in skewed partitions?

- **Block partition**



- **Hash-partition**

- On the key K
- On the attribute A



- **Range-partition**

- On the key K
- On the attribute A

Uniform Data v.s. Skewed Data

- Let $R(\underline{K}, A, B, C)$; which of the following partition methods may result in skewed partitions?

- **Block partition**

Uniform

- **Hash-partition**

- On the key K
- On the attribute A

Uniform

Assuming uniform hash function

- **Range-partition**

- On the key K
- On the attribute A

May be skewed

E.g. when all records have the same value of the attribute A , then all records end up in the same partition

Uniform Data v.s. Skewed Data

- Let $R(\underline{K}, A, B, C)$; which of the following partition methods may result in skewed partitions?

- Block partition

Uniform

- Hash-partition

- On the key K
- On the attribute A

Uniform

Assuming uniform hash function

May be skewed

E.g. when all records have the same value of the attribute A , then all records end up in the same partition

- Range-partition

- On the key K
- On the attribute A

May be skewed

Difficult to partition the range of A uniformly.

Data Partitioning Revisited

What are the pros and cons ?

- **Block based partitioning**
 - Good load balance but always needs to read all the data
- **Hash based partitioning**
 - Good load balance
 - Can avoid reading all the data for equality selections
- **Range based partitioning**
 - Can suffer from skew (i.e., load imbalances)
 - Can help reduce skew by creating uneven partitions

Horizontal Data Partitioning

All three choices are just special cases:

- For each tuple, compute $bin = f(t)$
- Different properties of the function f determine hash vs. range vs. round robin vs. anything

Parallel Selection

Compute $\sigma_{A=v}(R)$, or $\sigma_{v1 < A < v2}(R)$

- On a conventional database: cost = $B(R)$
- **Q:** What is the cost on a parallel database with P processors ?
 - Block partitioned
 - Hash partitioned
 - Range partitioned

Parallel Selection

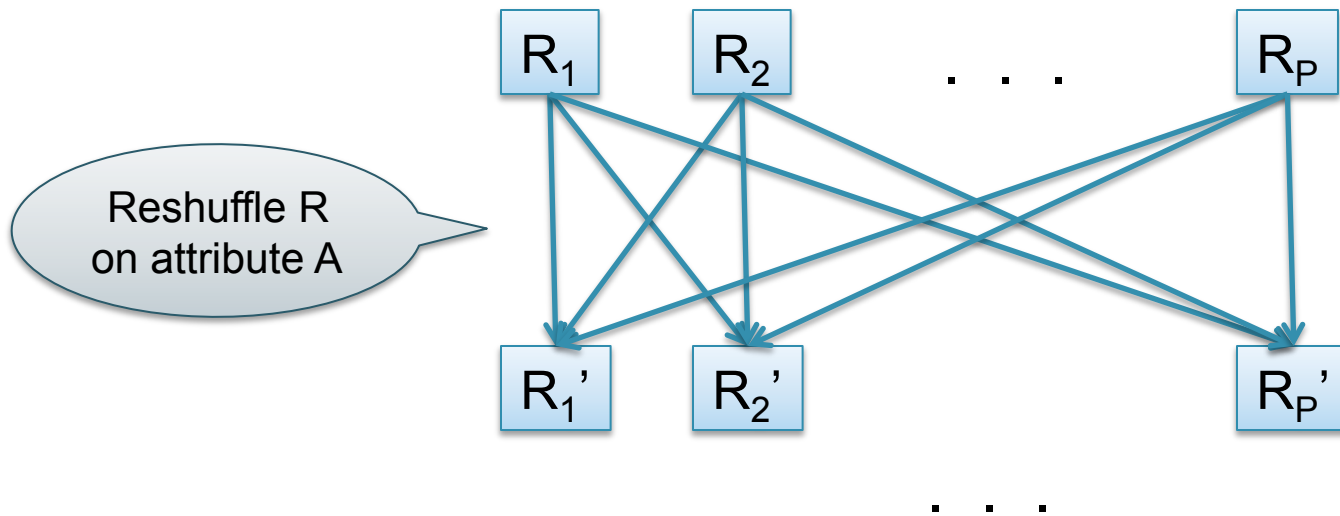
Compute $\sigma_{A=v}(R)$, or $\sigma_{v1 < A < v2}(R)$

- On a conventional database: cost = $B(R)$
- **Q:** What is the cost on a parallel database with P processors ?
A: $B(R) / P$, but
 - Block partitioned -- all servers do the work
 - Hash partitioned -- one server does the work
 - Range partitioned -- some servers do the work

Basic Parallel GroupBy

Data: $R(\underline{K}, A, B, C)$ -- hash-partitioned on K

Query: $\gamma_{A, \text{sum}(B)}(R)$



Basic Parallel GroupBy

- Step 1: each server i partitions its chunk R_i using a hash function $h(t.A) \bmod P$: $R_{i,0}, R_{i,1}, \dots, R_{i,P-1}$
- Step 2: server j computes $Y_{A, \text{sum}(B)}$ on $R_{0,j}, R_{1,j}, \dots, R_{P-1,j}$

Basic Parallel GroupBy

Compute $Y_{A, \text{sum}(B)}(R)$

- On a conventional database: cost = $B(R)$
- **Q:** What is the cost on a parallel database with P processors ?

Basic Parallel GroupBy

Compute $\gamma_{A, \text{sum}(B)}(R)$

- On a conventional database: cost = $B(R)$
- **Q:** What is the cost on a parallel database with P processors ?
- **A:** $B(R) / P$

Basic Parallel GroupBy

Can we do better?

- Sum?
- Count?
- Avg?
- Max?
- Median?

Basic Parallel GroupBy

Can we do better?

- Sum?
- Count?
- Avg?
- Max?
- Median?

Distributive	Algebraic	Holistic
$\text{sum}(a_1+a_2+\dots+a_9)=$ $\text{sum}(\text{sum}(a_1+a_2+a_3)+$ $\text{sum}(a_4+a_5+a_6)+$ $\text{sum}(a_7+a_8+a_9))$	$\text{avg}(B) =$ $\text{sum}(B)/\text{count}(B)$	$\text{median}(B)$

Parallel Join: $R \bowtie_{A=B} S$

- **Data:** $R(\underline{K1}, A, C), S(\underline{K2}, B, D)$
- **Query:** $R(\underline{K1}, A, C) \bowtie S(\underline{K2}, B, D)$

Initially, both R and S are horizontally partitioned on K1 and K2

R_1, S_1

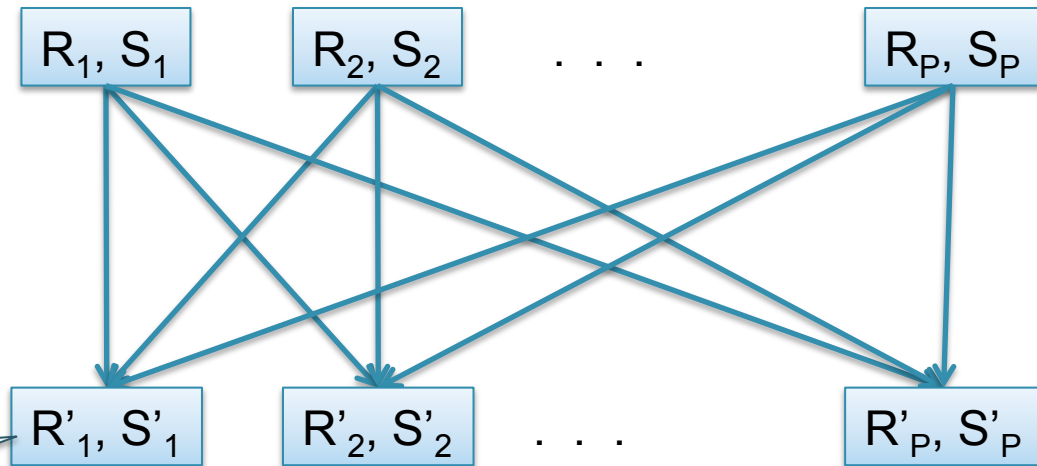
R_2, S_2

R_p, S_p

Parallel Join: $R \bowtie_{A=B} S$

- **Data:** $R(\underline{K1}, A, C)$, $S(\underline{K2}, B, D)$
- **Query:** $R(\underline{K1}, A, C) \bowtie S(\underline{K2}, B, D)$

Initially, both R and S are horizontally partitioned on K1 and K2



Reshuffle R on R.A
and S on S.B

Each server computes
the join locally

Parallel Join: $R \bowtie_{A=B} S$

- Step 1
 - Every server holding any chunk of R partitions its chunk using a hash function $h(t.A) \bmod P$
 - Every server holding any chunk of S partitions its chunk using a hash function $h(t.B) \bmod P$
- Step 2:
 - Each server computes the join of its local fragment of R with its local fragment of S

Parallel Join: $R \bowtie_{A=B} S$

Compute $R \bowtie_{A=B} S$

- On a conventional database: cost = $B(R)+B(S)$
- **Q:** What is the cost on a parallel database with P processors ?

Parallel Join: $R \bowtie_{A=B} S$

Compute $R \bowtie_{A=B} S$

- On a conventional database: cost = $B(R)+B(S)$
- **Q:** What is the cost on a parallel database with P processors ?
- **A:** $(B(R)+B(S)) / P$

Speedup and Scaleup

- Consider:
 - Query: $Y_{A, \text{sum}(C)}(R)$
 - Runtime: dominated by reading chunks from disk
- **If we double the number of nodes P** , what is the new running time?
- **If we double both P and the size of R** , what is the new running time?

Speedup and Scaleup

- Consider:
 - Query: $Y_{A, \text{sum}(C)}(R)$
 - Runtime: dominated by reading chunks from disk
- If we double the number of nodes P , what is the new running time?
 - Half (each server holds $\frac{1}{2}$ as many chunks)
- If we double both P and the size of R , what is the new running time?
 - Same (each server holds the same # of chunks)

Optimization for Small Relations

When joining R and S

- If $|R| \gg |S|$
 - Leave R where it is
 - Replicate entire S relation across nodes
- Also called a **small join** or a **broadcast join**

Other Interesting Parallel Join Implementation

Skew:

- Some partitions get more **input** tuples than others

Reasons:

- Range-partition instead of hash
 - Some values are very popular:
 - Heavy hitters values; e.g. ‘Justin Bieber’
 - Selection before join with different selectivities
-
- Some partitions generate more **output** tuples than others

Some Skew Handling Techniques

If using range partition:

- Ensure each range gets same number of tuples
- E.g.: {1, 1, 1, 2, 3, 4, 5, 6 } \rightarrow [1,2] and [3,6]
- Eq-depth v.s. eq-width histograms

Some Skew Handling Techniques

Create more partitions than nodes

- And be smart about scheduling the partitions
- Note: MapReduce uses this technique

Some Skew Handling Techniques

Use subset-replicate (a.k.a. “skewedJoin”)

- Given $R \bowtie_{A=B} S$
- Given a heavy hitter value $R.A = 'v'$
(i.e. $'v'$ occurs very many times in R)
- Partition R tuples with value $'v'$ across all nodes
e.g. block-partition, or hash on other attributes
- Replicate S tuples with value $'v'$ to all nodes
- R = the build relation
- S = the probe relation

Parallel Query Evaluation

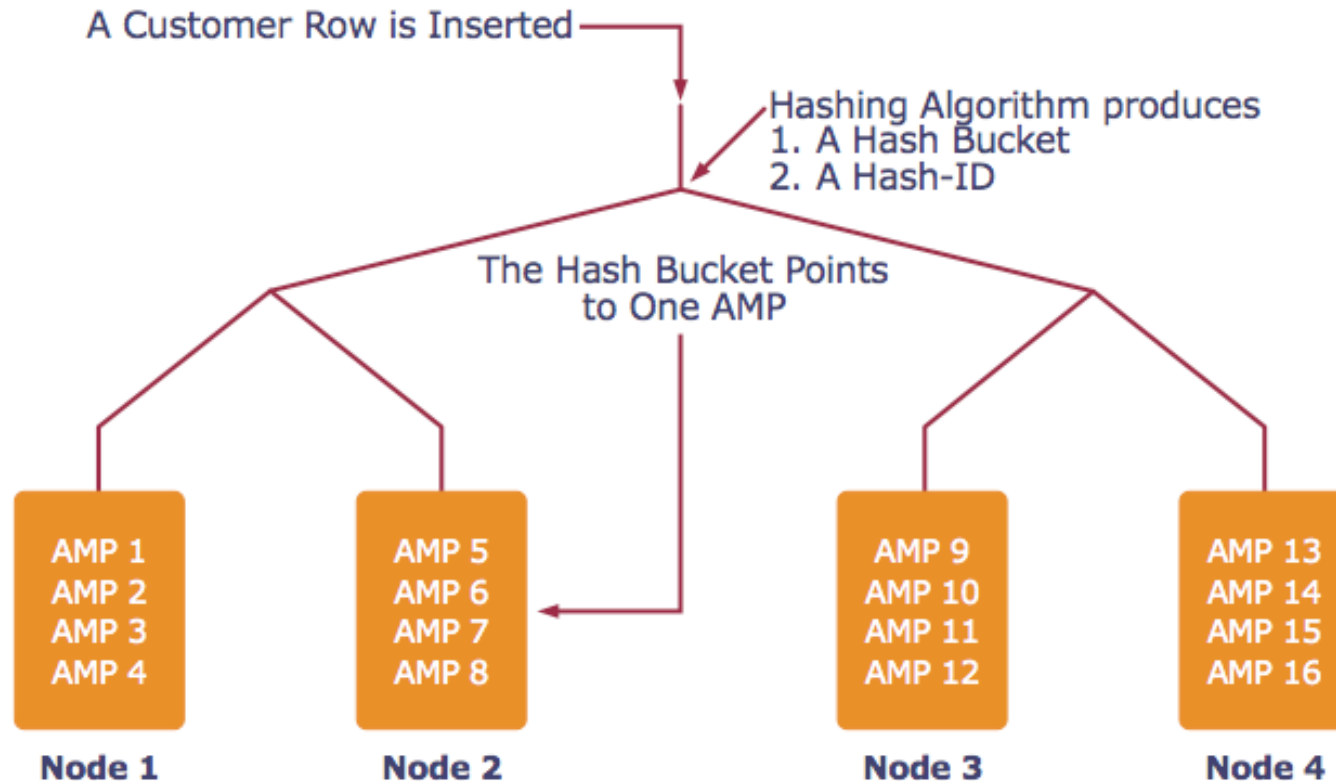
- Parallel query plan: tree of parallel operators
Intra-operator parallelism
 - Data streams from one operator to the next
 - Typically all cluster nodes process all operators
- Can run multiple queries at the same time
Inter-query parallelism
 - Queries will share the nodes in the cluster

Parallel Query Evaluation

New operator: **Shuffle**

- Handles data routing, buffering, and flow control
- Inserted between consecutive operators in the query plan
- Two components:
ShuffleProducer and **ShuffleConsumer**
- Producer:
 - Pulls data from operator and sends to n consumers
 - Producer acts as driver for operators below it in query plan
- Consumer:
 - Buffers input data from n producers and makes it available to operator through getNext interface

Example: Teradata – Loading

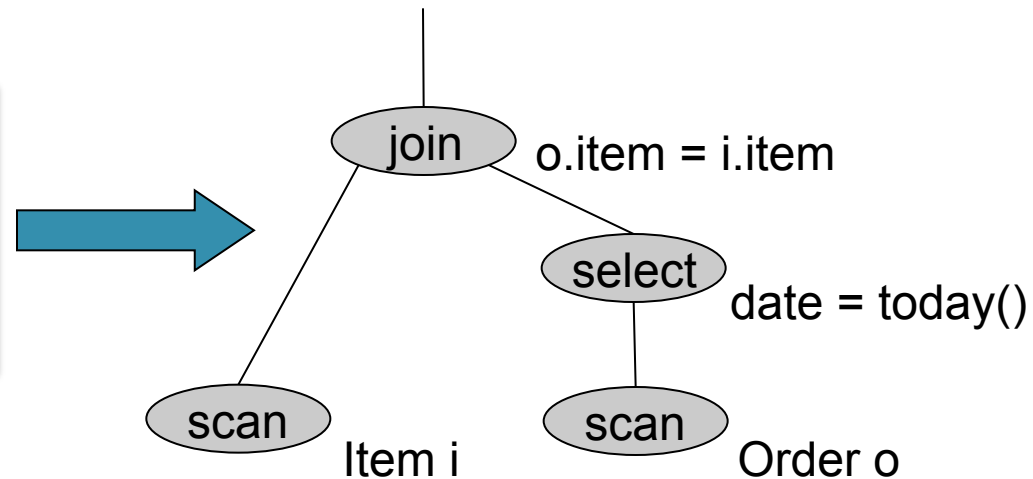


AMP = “Access Module Processor” = unit of parallelism

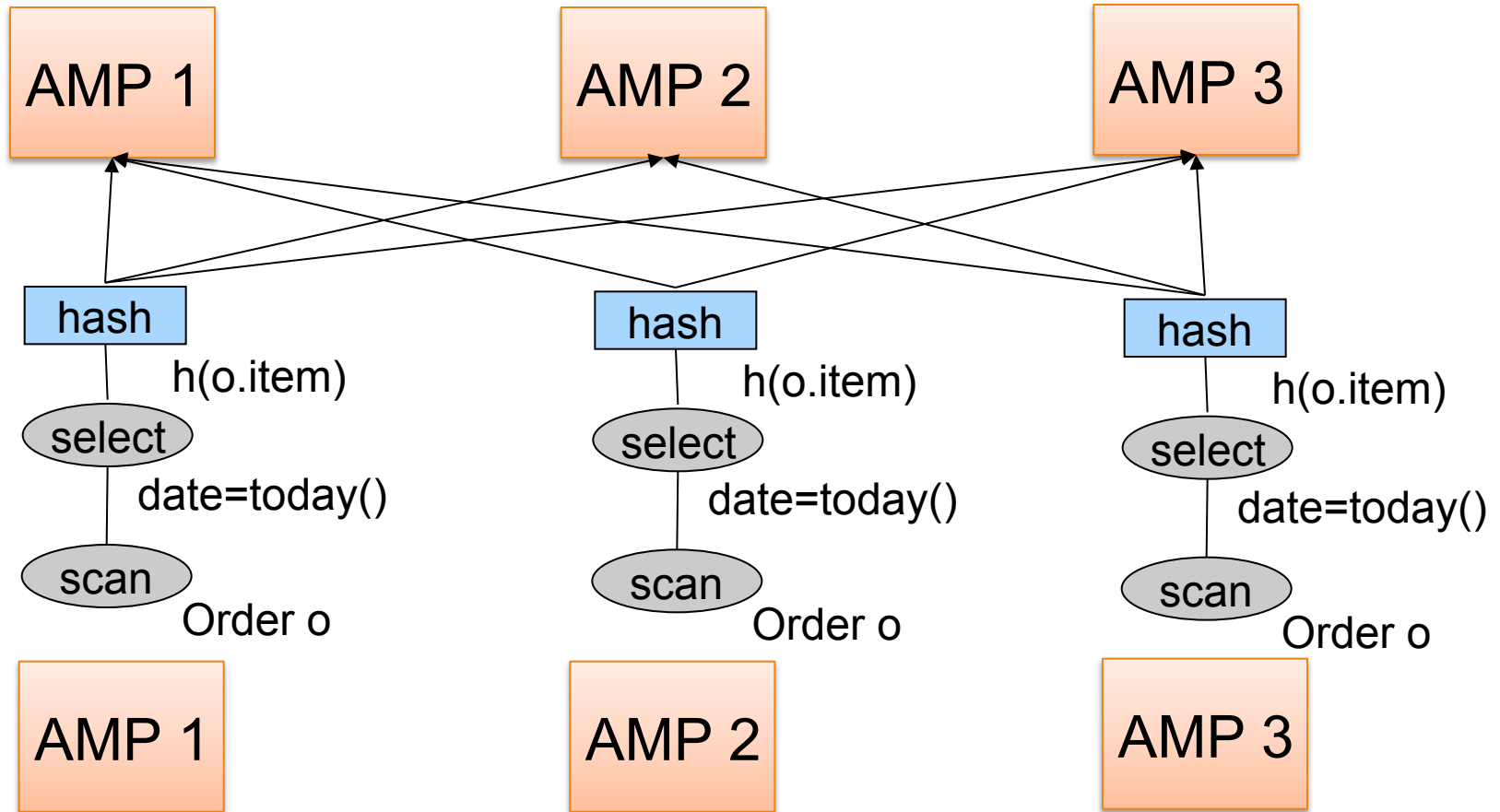
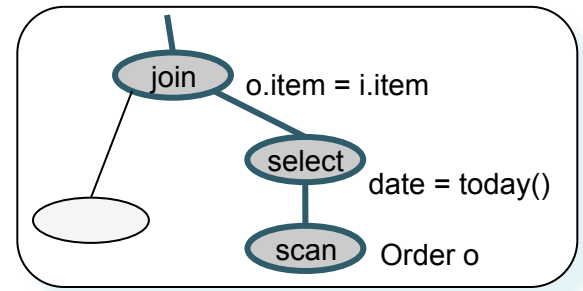
Example: Teradata – Query Execution

Find all orders from today, along with the items ordered

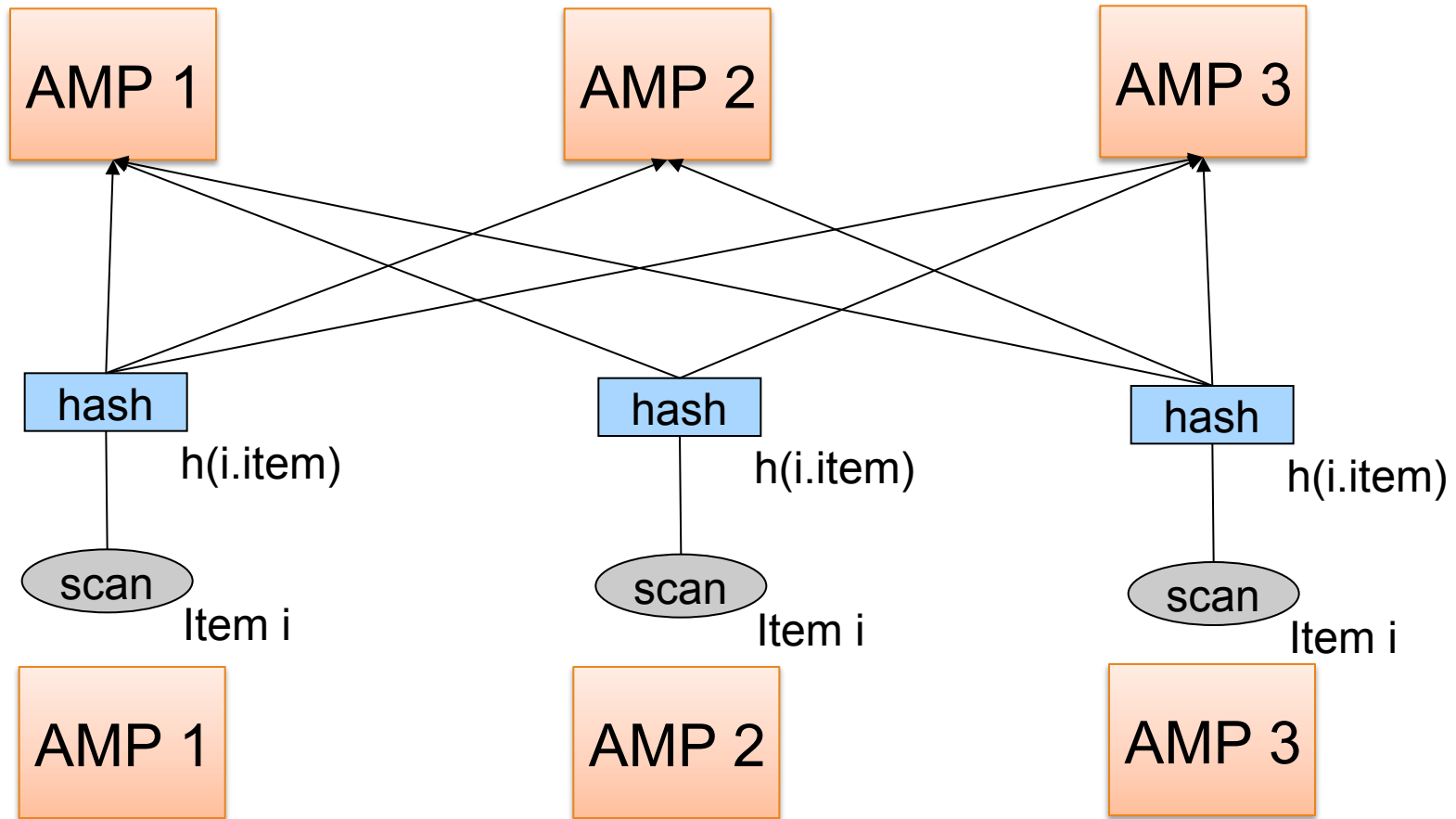
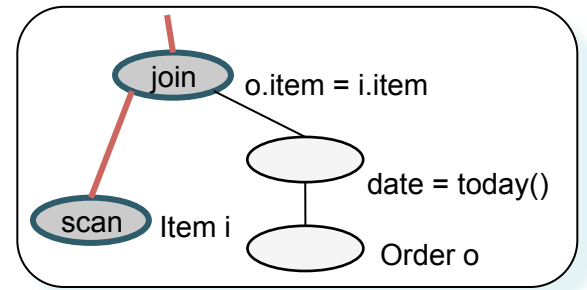
```
SELECT *  
  FROM Order o, Line i  
 WHERE o.item = i.item  
    AND o.date = today()
```



Query Execution



Query Execution



Query Execution

