CSE 544
Principles of Database Management Systems

Fall 2016

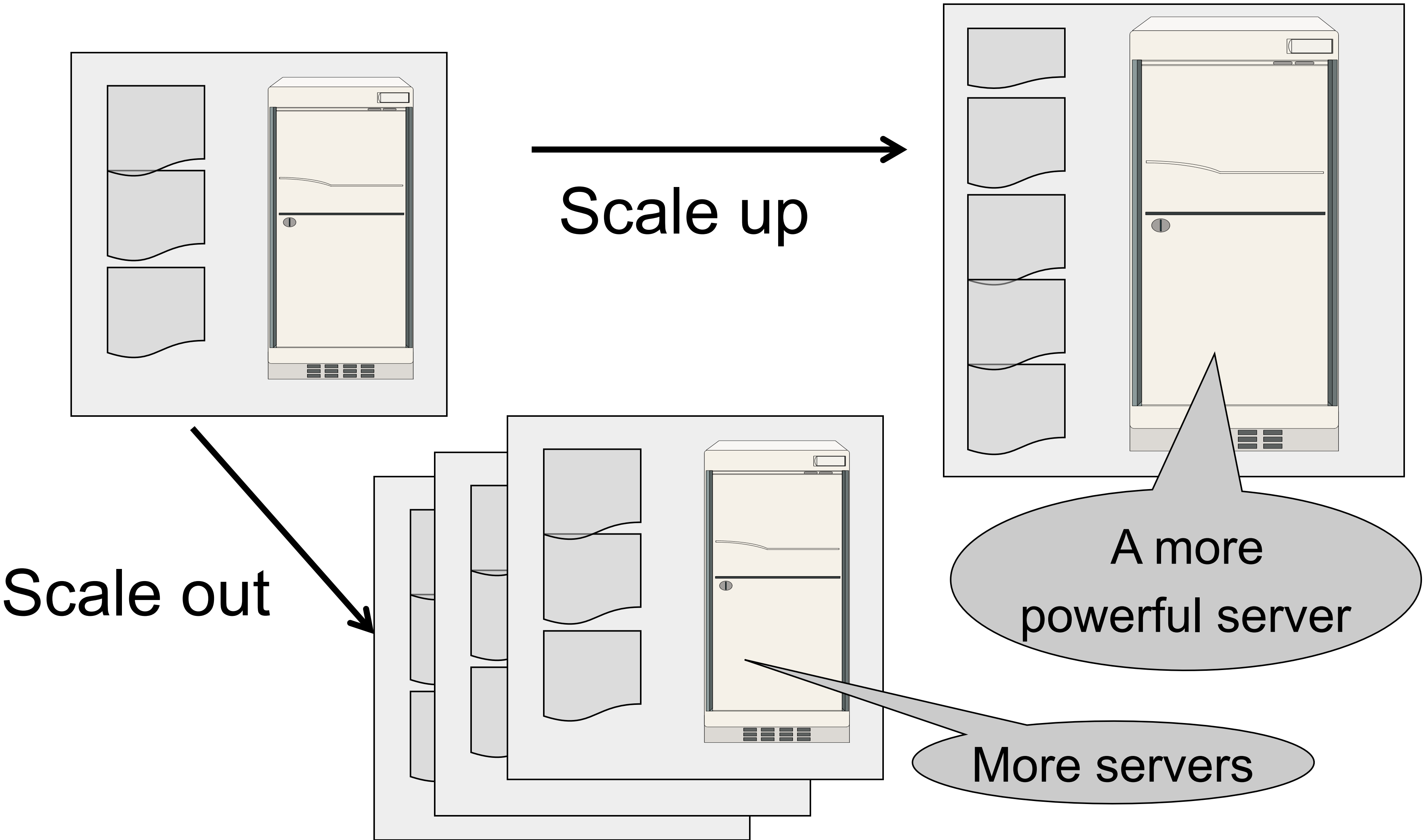
Lecture 13 – Parallel DBMSs

References

- **Parallel Database Systems: The Future of High Performance Database Systems.** Dave DeWitt and Jim Gray. Com. of the ACM. 1992. Sec. 1 and 2.

- **Database management systems.** Ramakrishnan and Gehrke. Third Ed. **Chapter 22.**

Two Ways to Scale a DBMS



Two Ways to Scale a DBMS

- Obviously this can be used to:
  - Execute multiple queries in parallel
  - Speed up a single query

- For now: how to speed up a single query

- We will worry about how to scale to multiple queries later

FYI: Data Analytics Companies

DB analytics companies:

- Greenplum founded in 2003 acquired by EMC in 2010; A parallel shared-nothing DBMS
- Vertica founded in 2005 and acquired by HP in 2011; A parallel, column-store shared-nothing DBMS
- DATAllegro 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. SQL on top of MapReduce
- Netezza founded in 2000 and acquired by IBM in 2010. A parallel, shared-nothing DBMS.

What does this mean? $$$$

Parallel v.s. Distributed Databases

- Distributed database system (later?):
  - Data is stored **across several sites** (geographically speaking), each site managed by a DBMS capable of running independently

- Parallel database system (today):
  - Data is stored **at a single site**, can be used to improve query performance through parallel implementation

Parallel DBMSs

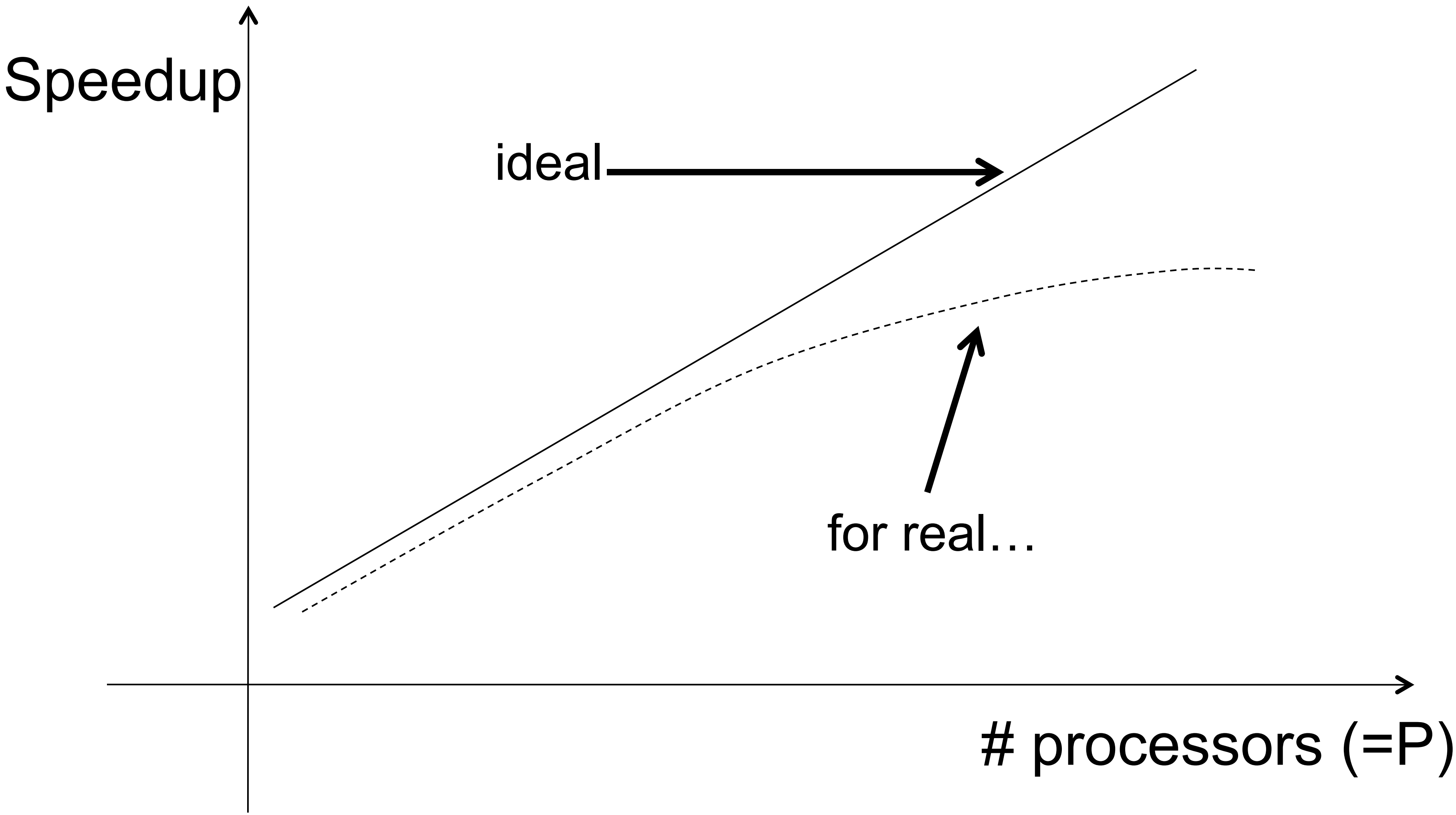
- Goal
  - Improve performance by executing multiple operations in parallel
- Key benefit
  - Cheaper to scale than relying on a single increasingly more powerful processor
- Key challenge
  - Ensure overhead and contention do not kill performance

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

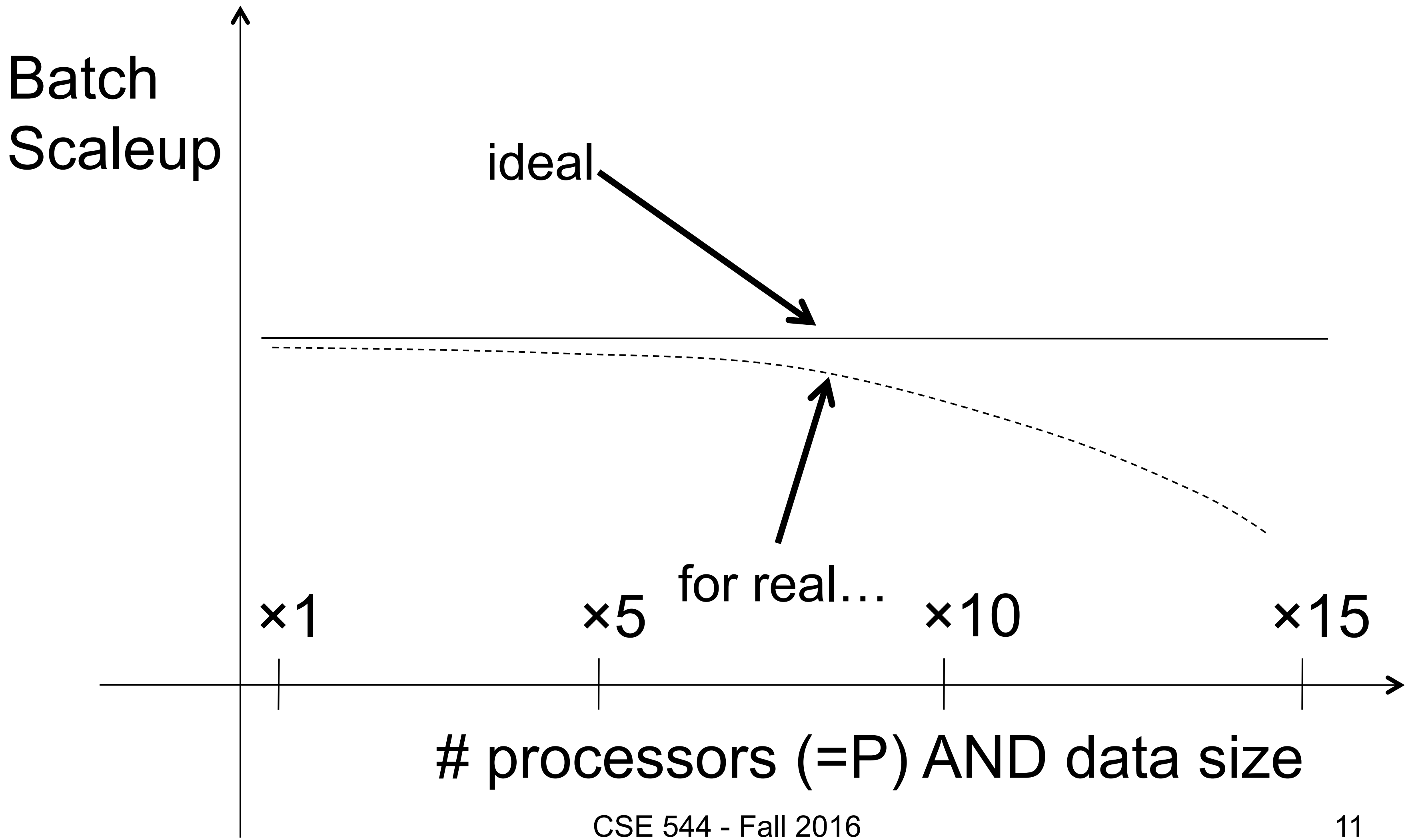


Performance Metrics for Parallel DBMSs

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



Buzzwords, buzzwords

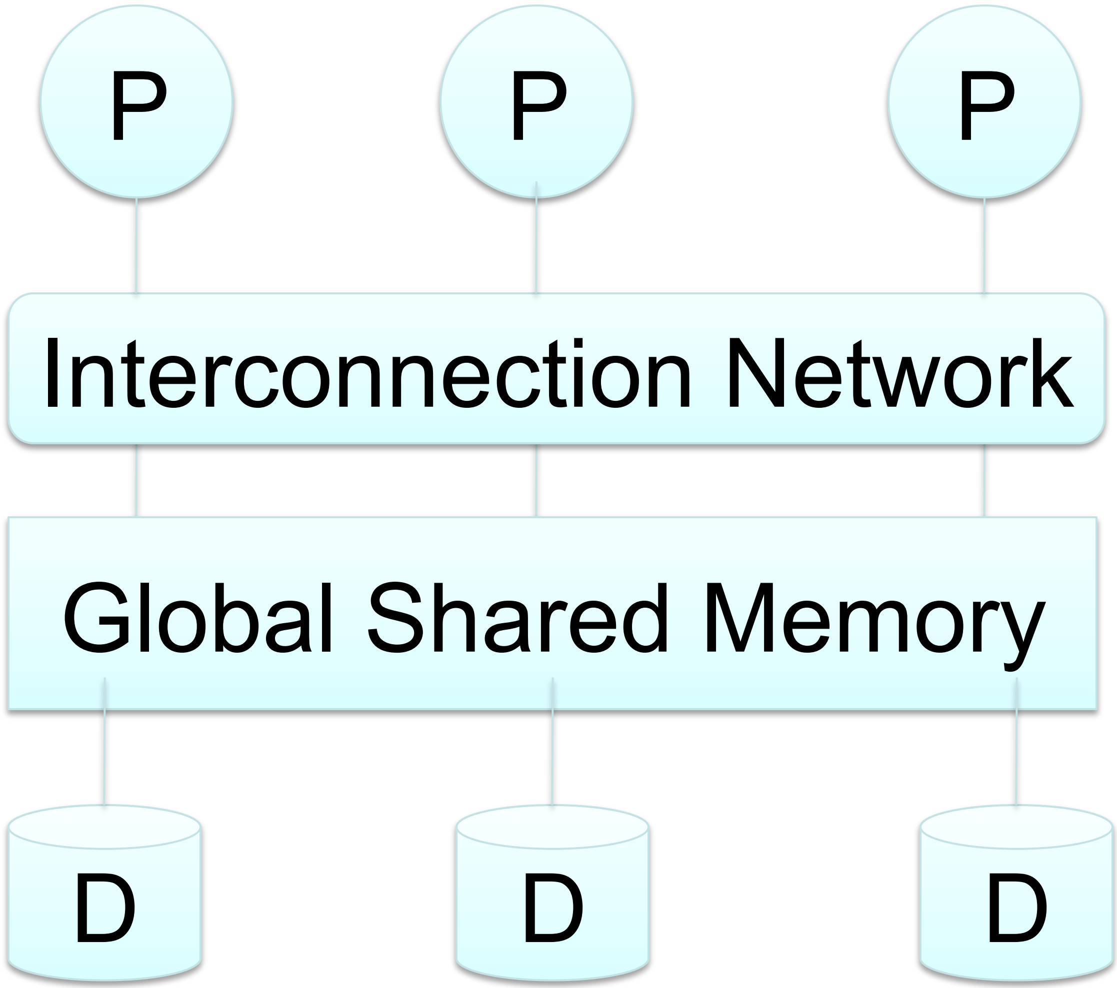
- Be careful. Commonly used terms today:
  - "scale up" = use an increasingly more powerful server
  - "scale out" = use a larger number of servers

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

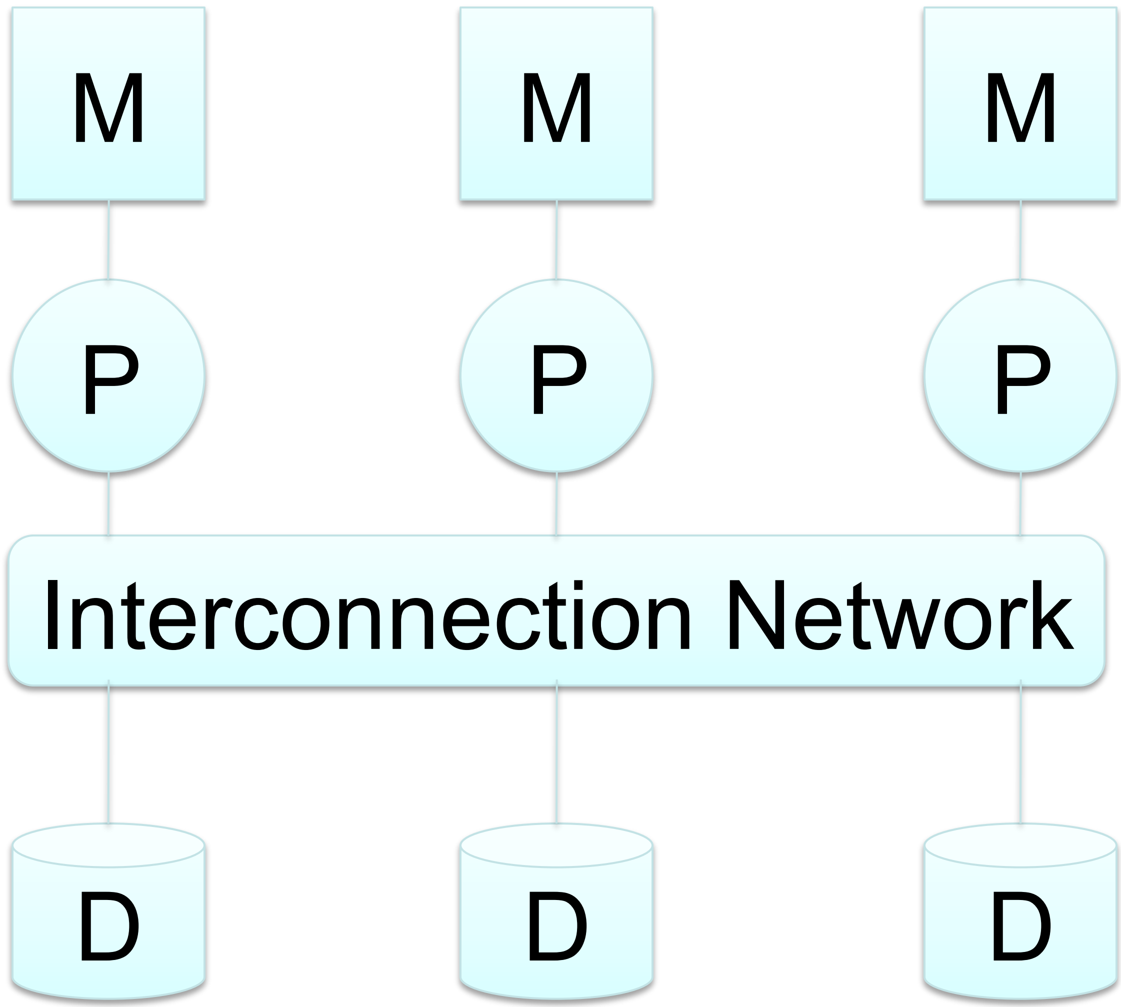
Parallel DBMS Architectures

Architecture for Parallel DBMS: Shared Memory

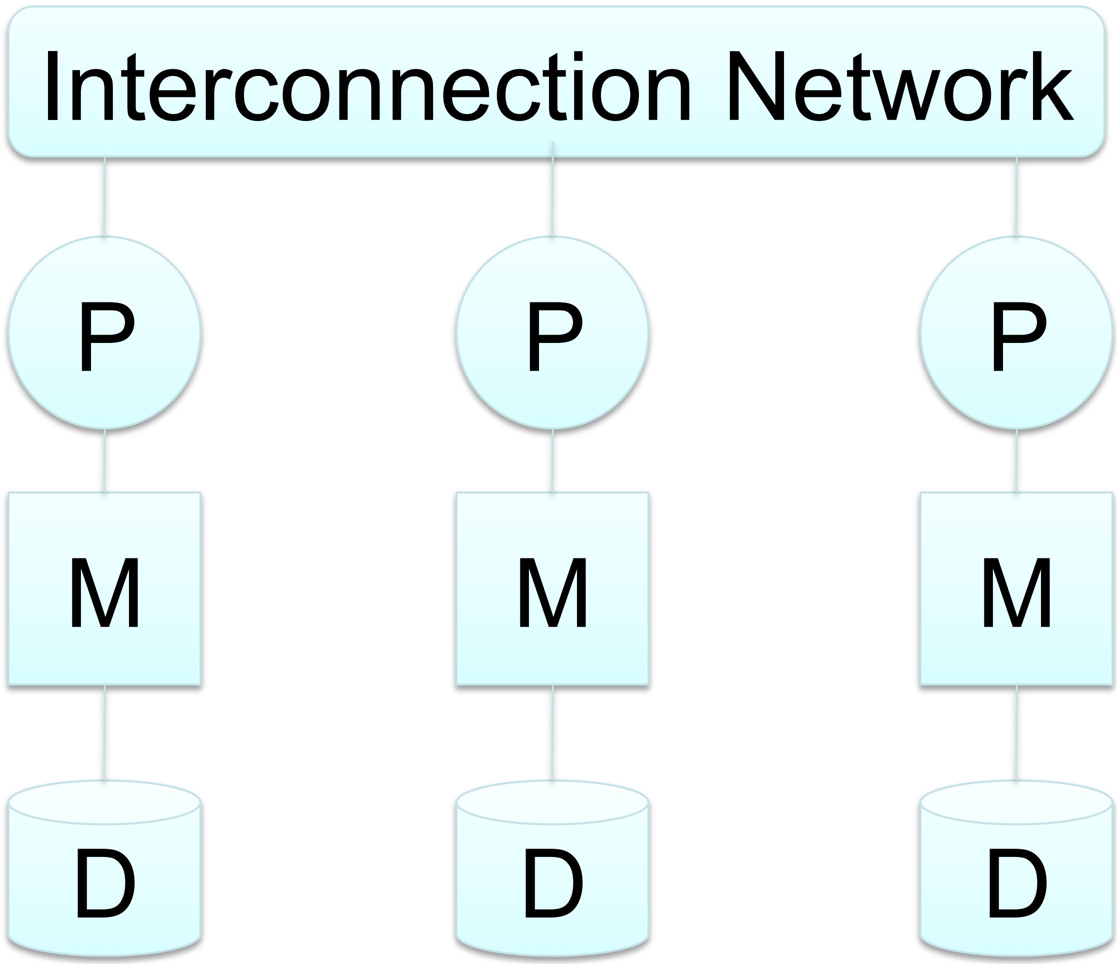


Aka SMP= symmetric multi processor

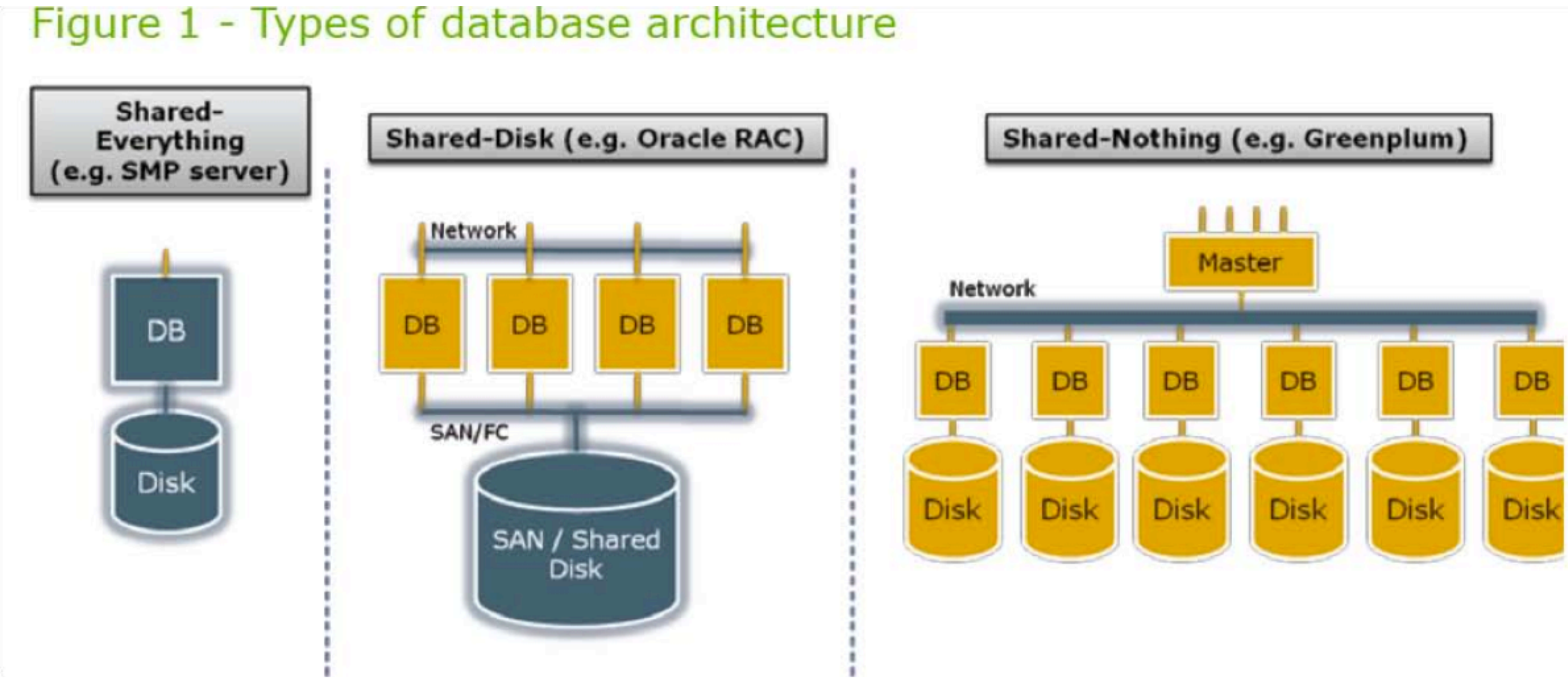
Architecture for Parallel DBMS: Shared Disk



Architecture for Parallel DBMS: Shared Nothing



A Professional Picture…



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

- leverage many threads to get a query to run faster

Characteristics:

- 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

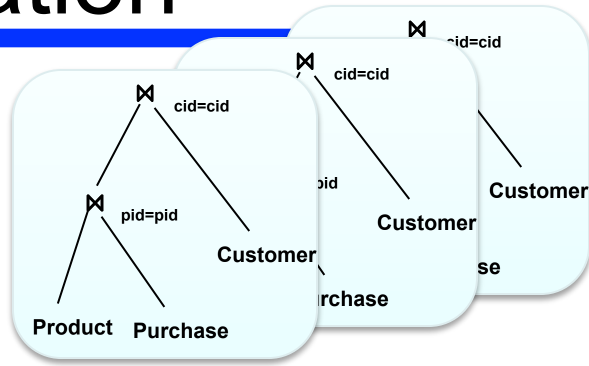
So…

- You have a parallel machine. Now what?

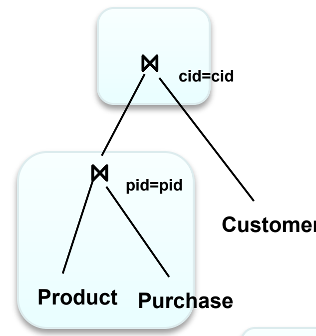
- How do you speed up your DBMS given a shared-nothing architecture?

Approaches to Parallel Query Evaluation

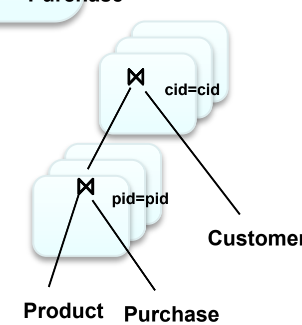
- Inter-query parallelism
  - Each query runs on one processor
  - Only for running multiple queries (OLTP)



- Inter-operator parallelism
  - A query runs on multiple processors
  - An operator runs on one processor
  - For both OLTP and Decision Support



- Intra-operator parallelism
  - An operator runs on multiple processors
  - For both OLTP and Decision Support



We study only intra-operator parallelism: most scalable

Data Partitioning

Horizontal Data Partitioning

- Relation R split into P chunks $R_0, \ldots, R_{P-1}$, 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) mod P

- Range based partitioning on attribute A:
  - Tuple t to chunk i if $v_{i-1} < t.A < v_i$

Need to worry about **data skew**

Uniform Data v.s. Skewed Data

- Let R(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.

All You Need to Know About Skew

Hash-partition a m data values (with duplicates!) to p bins

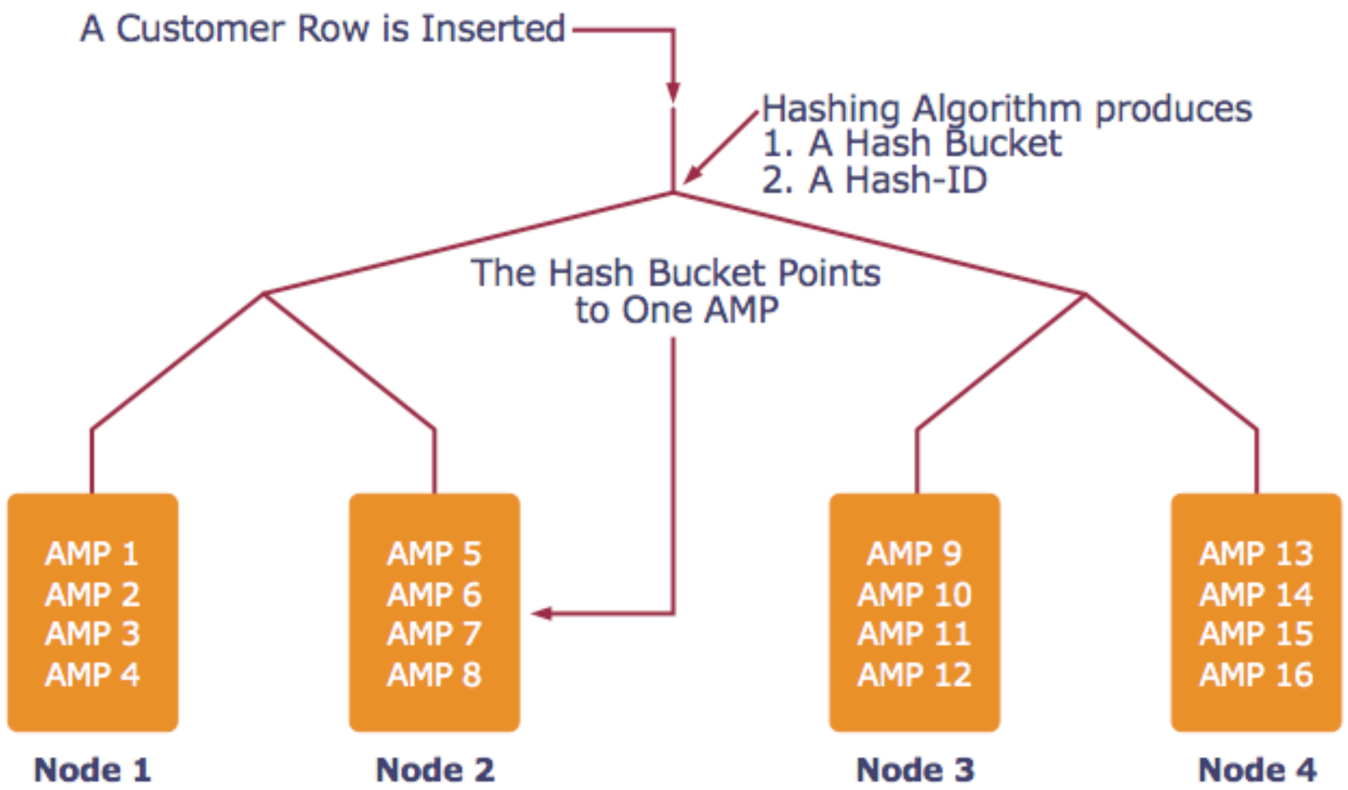
Fact 1 Expected size of any **one** fixed bin is m/p

Fact 2 Say that data is *skewed* if some value has degree > m/p. Then **some** bin has load > m/p

Hiding log p factors

Fact 3 Conversely, if the database is *skew-free* then max size of **all** bins = $O(m/p)$ w.h.p.

Example from Teradata



AMP = Access Module Processor = unit of parallelism

Parallelizing Operator Implementations

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

Q: What is the cost on a parallel database with P nodes ?

A: B(R) / P in all cases (except range) if cost is response time

However, not all processors are equal (workwise):

- Block: all servers do the same amount of work
- Hash: one server for $\sigma_{A=v}(R)$, all for $\sigma_{v1<A<v2}(R)$
- Range: some servers only

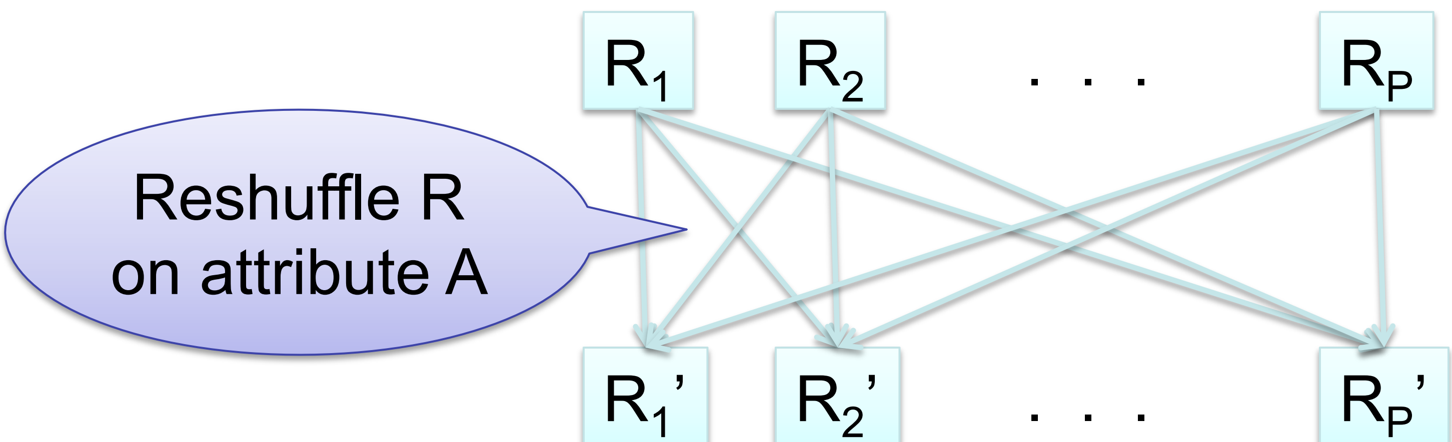
Data Partitioning Revisited

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

Parallel Group By: $\gamma_{A,\ sum(B)}(R)$

- If R is partitioned on A, then each node computes the group-by locally
- Otherwise, hash-partition R(<u>K</u>,A,B,C) on A, then compute group-by locally:



Parallel Group By: $\gamma_{A,\ sum(B)}(R)$

- Step 1: server i partitions chunk R_i using a hash function h(t.A) mod P: R_{i0}, R_{i1}, …, $R_{i,P-1}$ (there are P servers total)

- Step 2: server i sends partition R_{ij} to server j

- Step 3: server j computes $\gamma_{A,\ sum(B)}$ on R_{0j}, R_{1j}, …, $R_{P-1,j}$

Parallel Group By: $\gamma_{A,\ sum(B)}(R)$

Can we do better?

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

Parallel Group By: $\gamma_{A,\ sum(B)}(R)$

- $Sum(B) = Sum(B_0) + Sum(B_1) + \ldots + Sum(B_n)$
- $Count(B) = Count(B_0) + Count(B_1) + \ldots + Count(B_n)$
- $Max(B) = Max(Max(B_0), Max(B_1), \ldots, Max(B_n))$

distributive

- $Avg(B) = Sum(B) / Count(B)$

algebraic

- Median(B) = ???

holistic

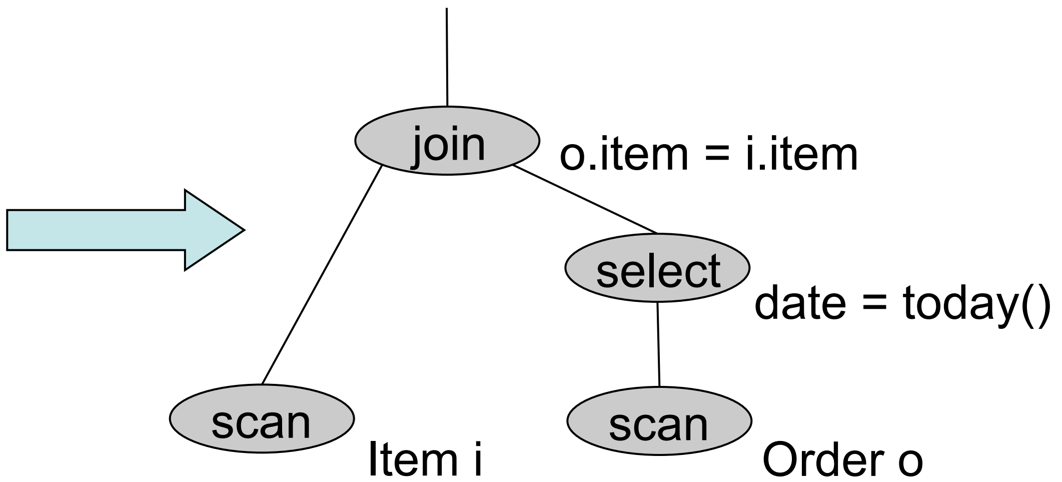
Parallel Join: $R \bowtie_{A=B} S$

- Step 1
  - For all servers in [0,k], server i partitions chunk R_i using a hash function h(t.A) mod P: R_{i0}, R_{i1}, …, $R_{i,P-1}$
  - For all servers in [k+1,P], server j partitions chunk S_j using a hash function h(t.A) mod P: S_{j0}, S_{j1}, …, $R_{j,P-1}$
- Step 2:
  - Server i sends partition R_{iu} to server u
  - Server j sends partition S_{ju} to server u
- Steps 3: Server u computes the join of R_{iu} with S_{ju}

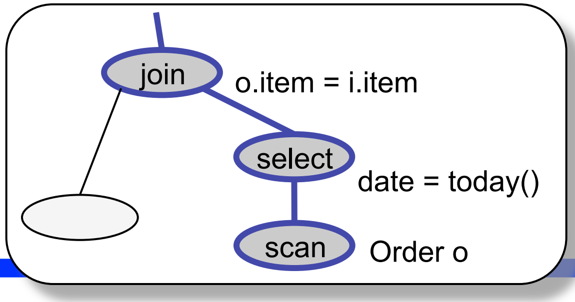
Example of Parallel Query Plan

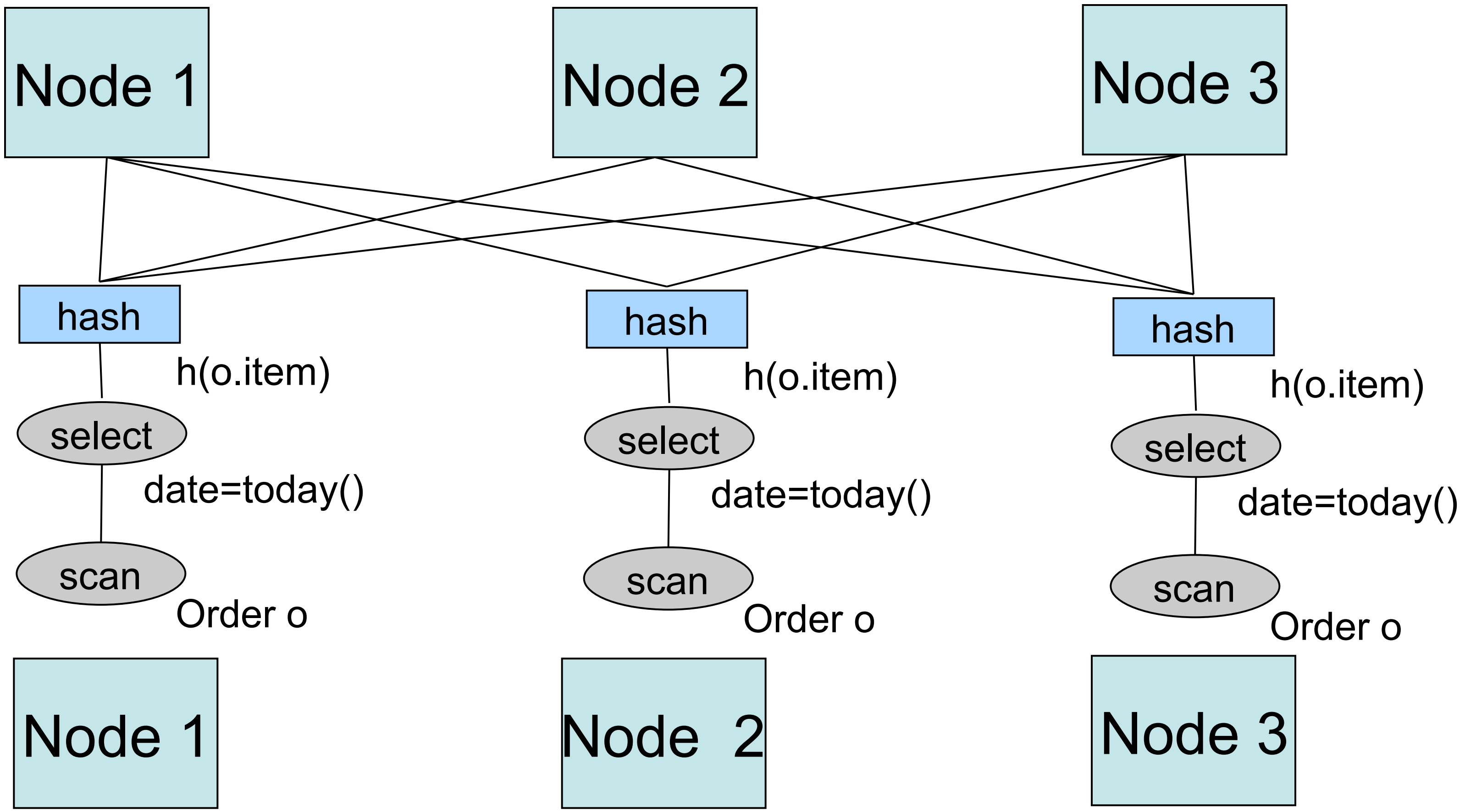
Find all orders from today, along with the items ordered

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

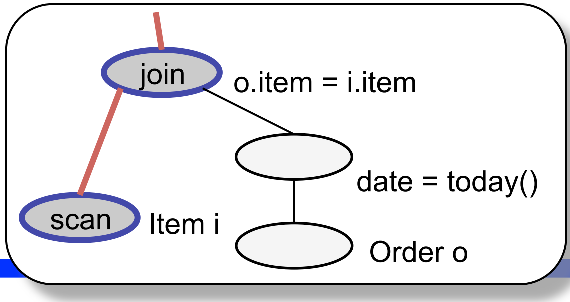


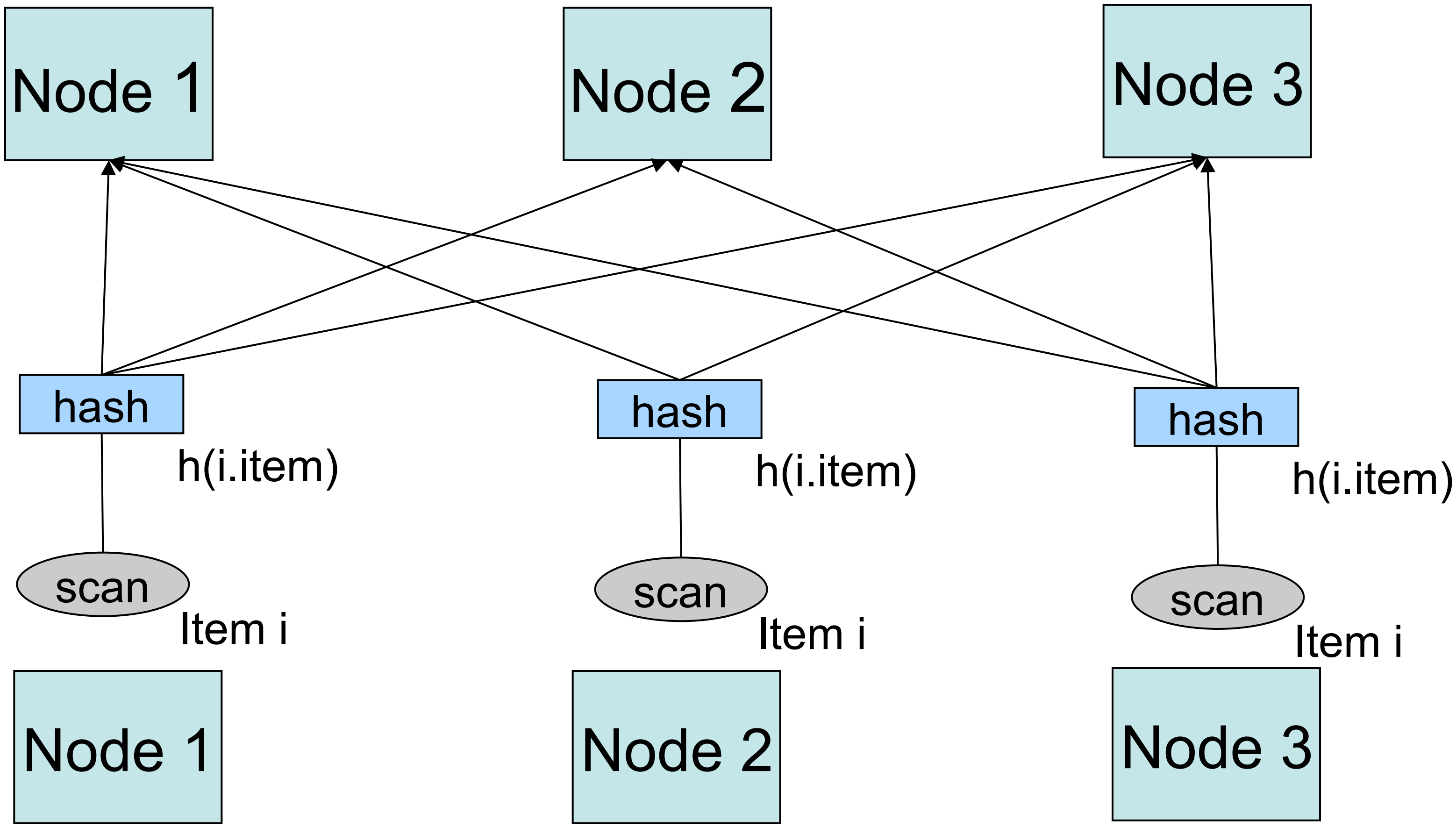
Example Parallel Plan



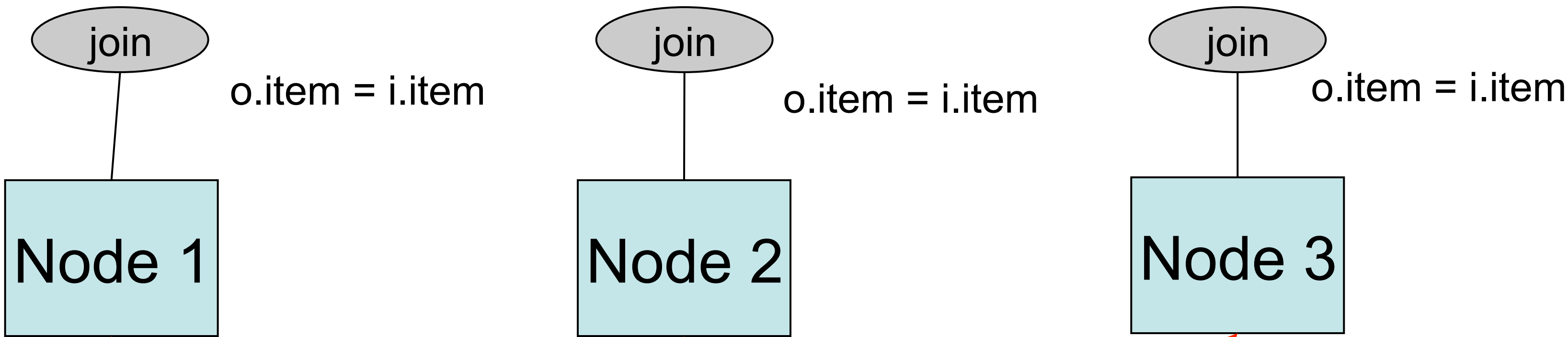


Example Parallel Plan





Example Parallel Plan



contains all orders and all lines where hash(item) = 3

contains all orders and all lines where hash(item) = 2

contains all orders and all lines where hash(item) = 1

Optimization for Small Relations

- When joining R and S
- If |R| >> |S|
  - Leave R where it is
  - Replicate entire S relation across nodes

- Sometimes called a “small join” or “broadcast join”

Other Interesting Parallel Join Implementation

Problem of skew during join computation

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

- Reason 1: Base data unevenly distributed
  - Because used a range-partition function
  - Or used hashing but some values are very popular (Skew)
- Reason 2: Selection before join with different selectivities
- Reason 3: Input data got unevenly rehashed (or otherwise repartitioned before the join)

Some partitions **output** more tuples than others

Some Skew Handling Techniques

1. Use range- instead of hash-partitions
   - Ensure that each range gets same number of tuples
   - Example: {1, 1, 1, 2, 3, 4, 5, 6 } → [1,2] and [3,6]
2. Create more partitions than nodes
   - And be smart about scheduling the partitions
3. Use subset-replicate (i.e., "skewedJoin")
   - Given an extremely common value 'v'
   - Distribute R tuples with value v randomly across k nodes (R is the build relation)
   - Replicate S tuples with value v to same k machines (S is the probe relation)

Parallel Dataflow Implementation

Use relational operators unchanged

Add a special *shuffle* operator

- Handle 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

Conclusion

- Making databases parallel is another way to speed up query processing
- Many algorithms for parallelizing different relational operators
- Next time: column stores