Introduction to Data Management
CSE 344

Lecture 21: Parallel DBMSs

Announcements

- WQ7 due tonight
- HW7 due on Thursday

Welcome to the 2nd half of 344

- Relational data model
  - Instance
  - Schema
  - Query languages
    - SQL, RA, RC, Datalog
- Query processing
  - Logical & physical plans
  - Indexes
  - Cost estimation
  - Query optimization
- Non-relational data model
- Conceptual design
  - E/R diagrams
  - Converting to SQL
  - Normalization
- Transactions
  - ACID
  - Transaction Implementation
  - Writing DB applications
- Parallel query processing
  - MapReduce
  - Spark

Today

- Architecture of parallel DBMSs
- Distributing data to multiple machines
- Executing relational query operators in parallel
- Alternative data models for parallel DBMSs

Why compute in parallel?

- Multi-cores:
  - Most processors have multiple cores
  - This trend will increase in the future

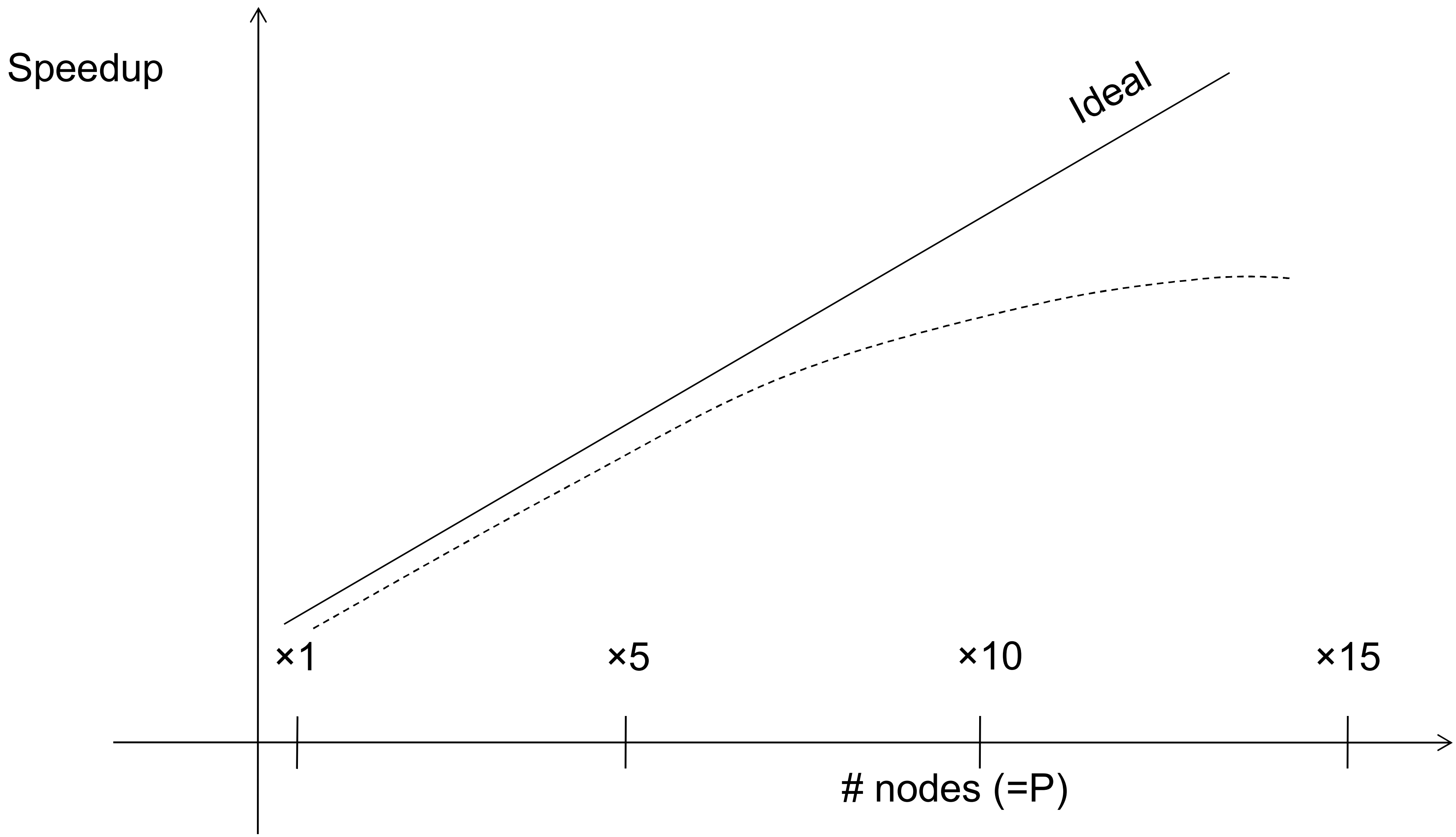
- Big data: too large to fit in main memory
  - Distributed query processing on 100x-1000x servers
  - Widely available now using cloud services

Performance Metrics for Parallel DBMSs

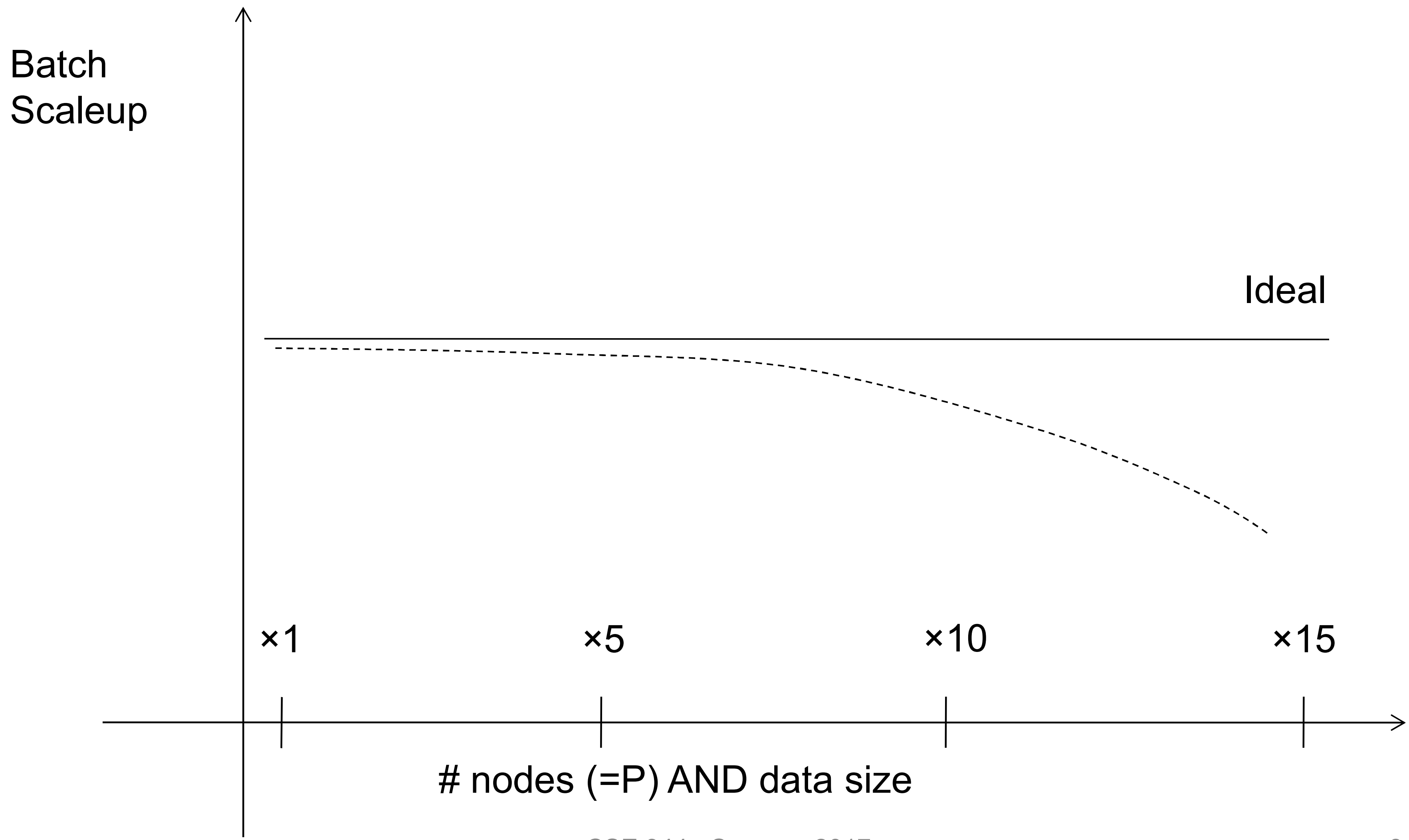
Nodes = processors, computers

- Speedup:
  - More nodes, same data ➔ higher speed
- Scaleup:
  - More nodes, more data ➔ same speed

Linear v.s. Non-linear Speedup



Linear v.s. Non-linear Scaleup



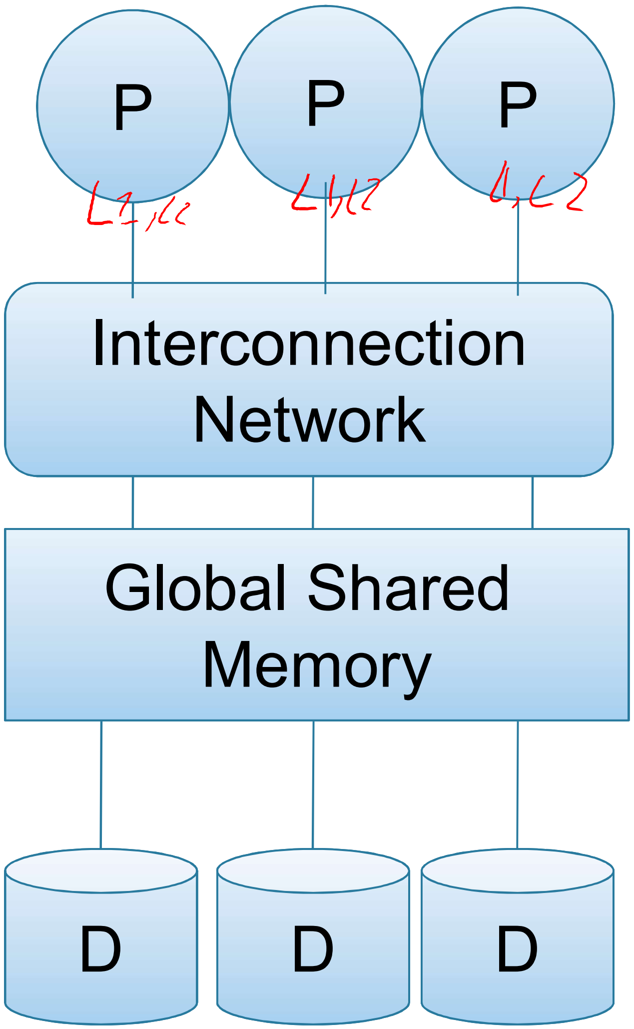
Why Sub-linear Speedup and Scaleup?

- Startup cost
  - Cost of starting an operation on many nodes
- Interference
  - Contention for resources between nodes
- Skew/Stragglers
  - Slowest node becomes the bottleneck

Architectures for Parallel Databases

- Shared memory
- Shared disk
- Shared nothing

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 speed up a query

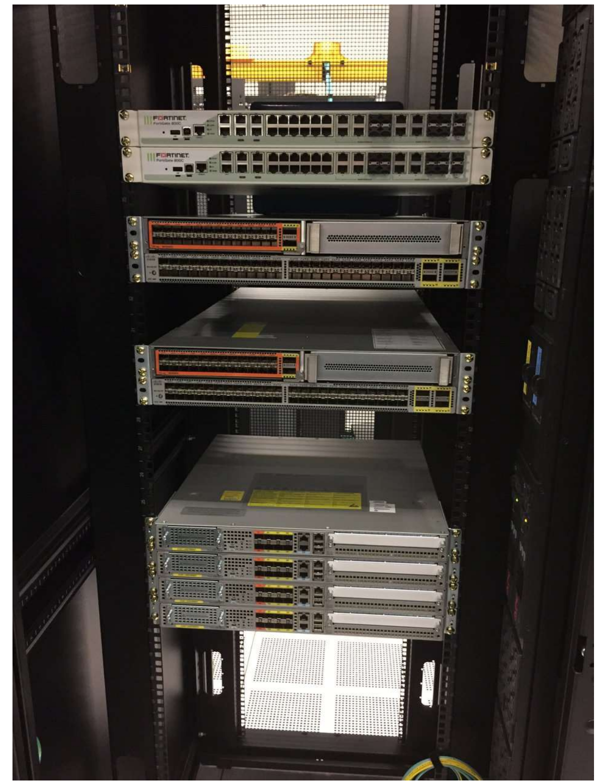
- check your HW3 query plans

- Easy to use and program
- Expensive to scale
  - last remaining cash cows in the hardware industry

StackOverflow Hardware

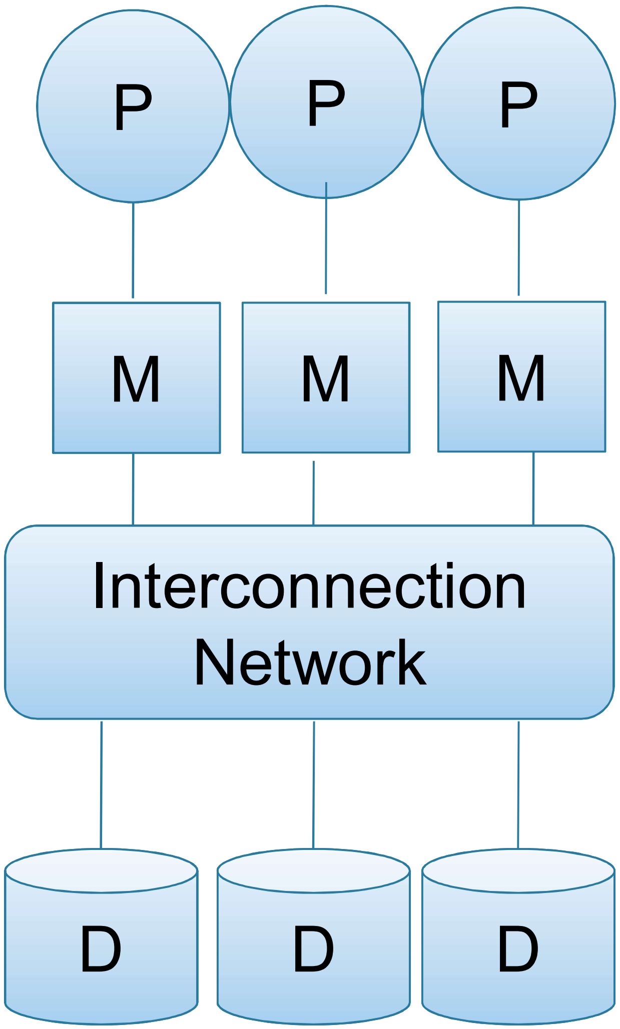
SQL Servers (Stack Overflow Cluster)

- 2 Dell R720xd Servers, each with:
- Dual E5-2697v2 Processors (12 cores @2.7–3.5GHz each)
- 384 GB of RAM (24x 16 GB DIMMs)
- 1x Intel P3608 4 TB NVMe PCIe SSD (RAID 0, 2 controllers per card)
- 24x Intel 710 200 GB SATA SSDs (RAID 10)
- Dual 10 Gbps network (Intel X540/I350 NDC)



https://nickcraver.com/blog/2016/03/29/stack-overflow-the-hardware-2016-edition/

Shared Disk



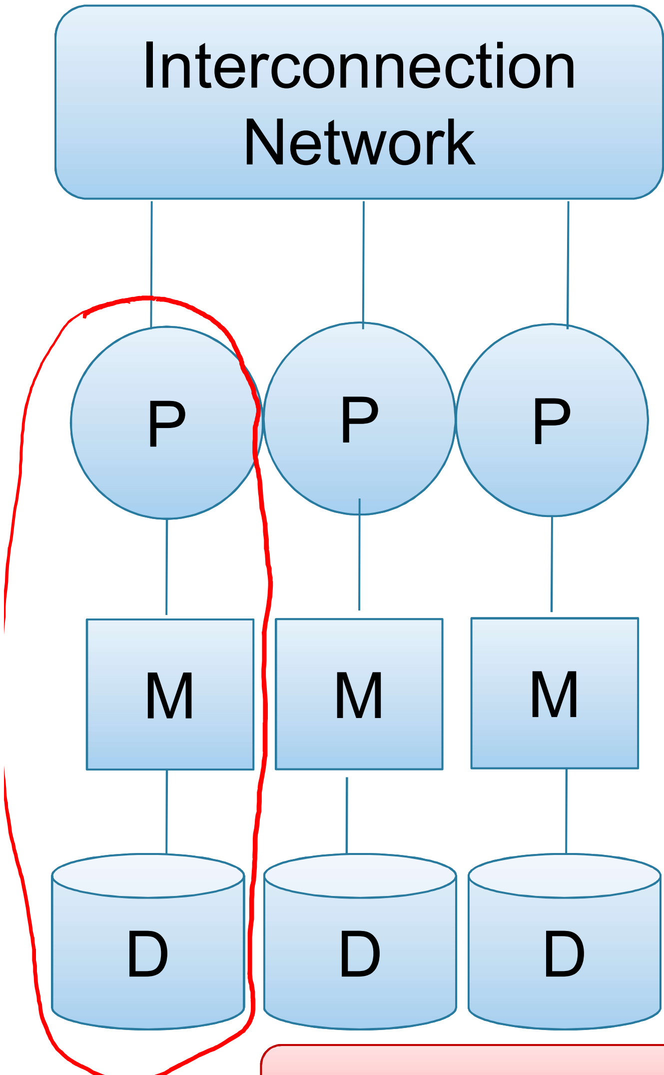
- All nodes access the same disks
- Found in the largest "single-box" (non-cluster) multiprocessors

Example: Oracle

- No need to worry about shared memory

- Hard to scale: existing deployments typically have fewer than 10 machines

Shared Nothing



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

Example: Google

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

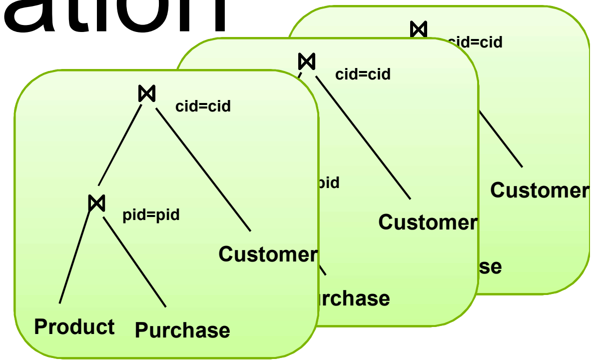
- Easy to maintain and scale
- Most difficult to administer and tune.

We discuss only Shared Nothing in class

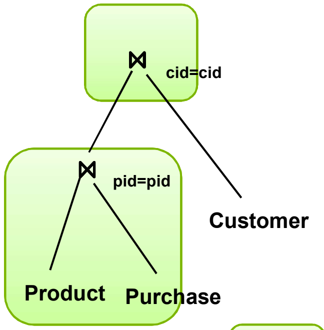
Parallel Data Processing @ 1990

Approaches to Parallel Query Evaluation

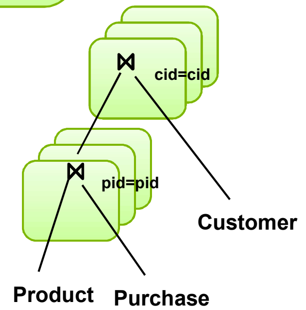
- Inter-query parallelism
  - Transaction per node
  - Good for transactional workloads



- Inter-operator parallelism
  - Operator per node
  - Good for analytical workloads



- Intra-operator parallelism
  - Operator on multiple nodes
  - Good for both?



We study only intra-operator parallelism: most scalable

Single Node Query Processing (Review)

Given relations R(A,B) and S(B, C), no indexes:

- Selection: $\sigma_{A=123}(R)$
  - Scan file R, select records with A=123
- Group-by: $\gamma_{A,sum(B)}(R)$
  - Scan file R, insert into a hash table using A as key
  - When a new key is equal to an existing one, add B to the value
- Join: $R \bowtie S$
  - Scan file S, insert into a hash table using B as key
  - Scan file R, probe the hash table using B

Distributed Query Processing

- Data is horizontally partitioned on many servers
- Operators may require data reshuffling
- First let's discuss how to distribute data across multiple nodes / servers

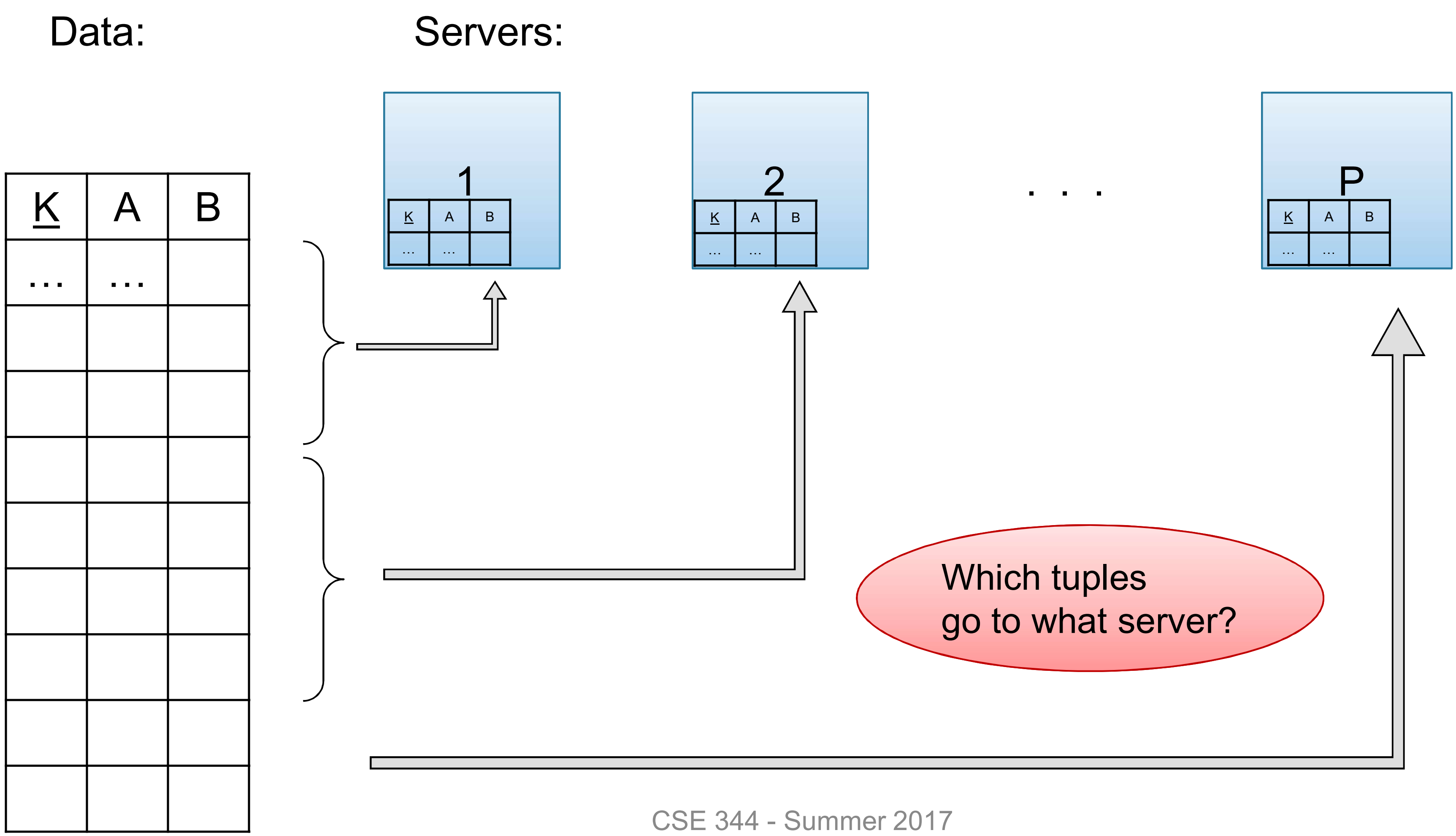
Horizontal Data Partitioning

Data:

K	A	B
…	…	

Servers:

Horizontal Data Partitioning



Horizontal Data Partitioning

- Block Partition:
  - Partition tuples arbitrarily s.t. $size(R_1) \approx \ldots \approx size(R_P)$

- Hash partitioned on attribute A:
  - Tuple t goes to chunk i, where i = h(t.A) mod P + 1
  - Recall: calling hash fn's is free in this class

↳ most common

- Range partitioned on attribute A:
  - Partition the range of A into $-\infty = v_0 < v_1 < \ldots < v_P = \infty$
  - Tuple t goes to chunk i, if $v_{i-1} < t.A < v_i$

Uniform Data v.s. Skewed Data

- Let R(<u>K</u>,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 good 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 tition

Keep this in mind in the next few slides

Parallel GroupBy

Data: R(K,A,B,C)

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

How can we compute in each case?

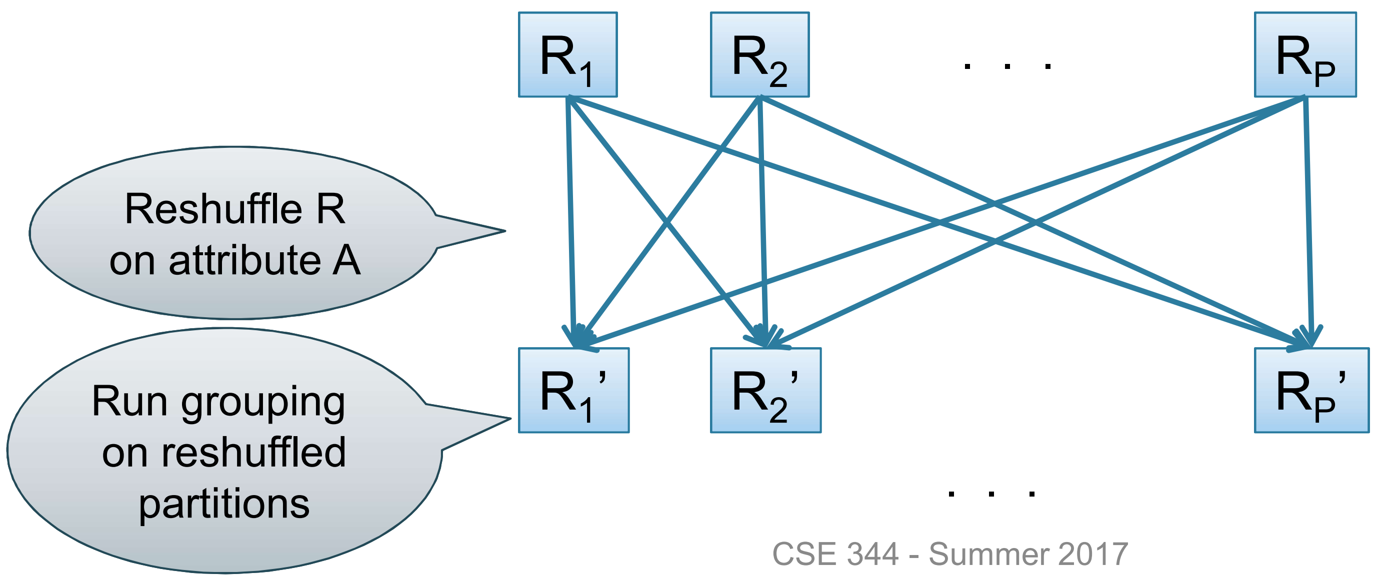
- R is hash-partitioned on A ← easy case!
- R is block-partitioned
- R is hash-partitioned on K

Parallel Execution of RA Operators: Grouping

Data: R(K,A,B,C)

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

- R is block-partitioned or hash-partitioned on K

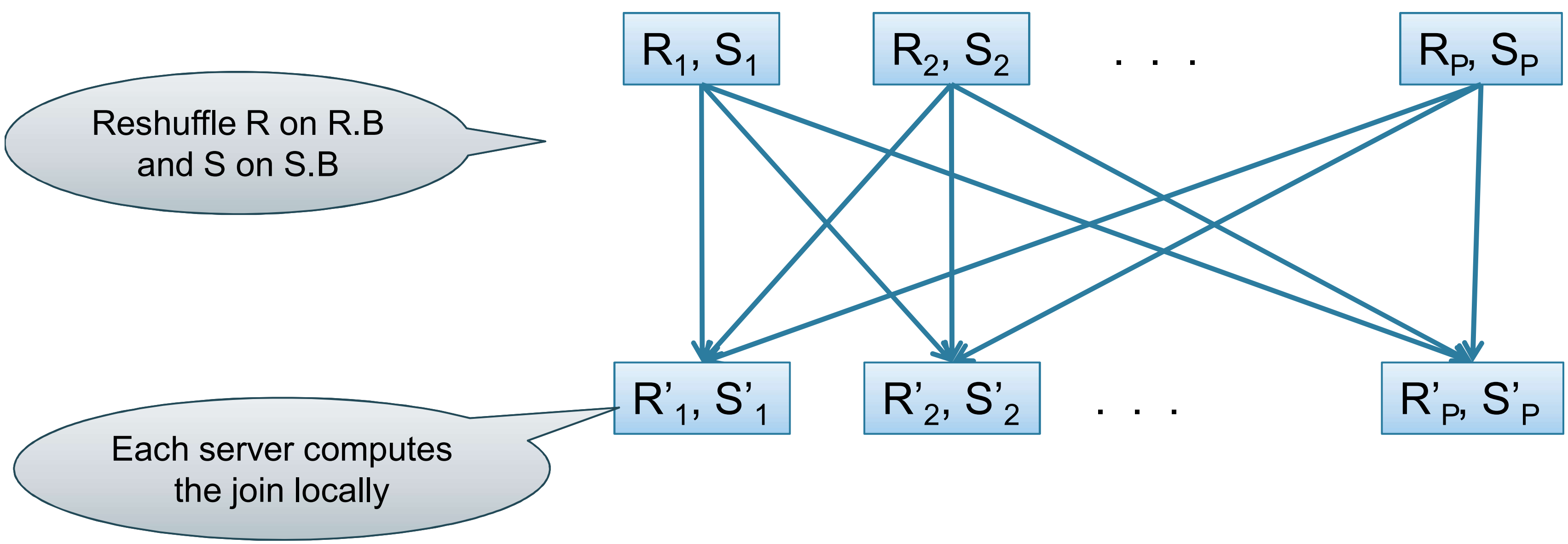


Speedup and Scaleup

- Consider:
  - Query: $\gamma_{A,sum(C)}(R)$
  - Runtime: only consider I/O costs
- If we double the number of nodes P, what is the new running time?
  - Half (each server holds ½ as many chunks) ideal only
- If we double both P and the size of R, what is the new running time?
  - Same (each server holds the same # of chunks)

Parallel Execution of RA Operators: Partitioned Hash-Join

- **Data**: R(K1, A, B), S(K2, B, C)
- **Query**: R(K1, A, B) ⋈ S(K2, B, C)
  - Initially, both R and S are partitioned on K1 and K2



Data: R(K1,A, B), S(K2, B, C)
Query: R(K1,A,B) ⋈ S(K2,B,C)

Parallel Join Illustration

Partition

M1

R1

K1	B
1	20
2	50

S1

K2	B
101	50
102	50

M2

R2

K1	B
3	20
4	20

S2

K2	B
201	20
202	50

Shuffle on B

Local Join

M1

R1'

K1	B
1	20
3	20
4	20

⋈

S1'

K2	B
201	20

M2

R2'

K1	B
2	50

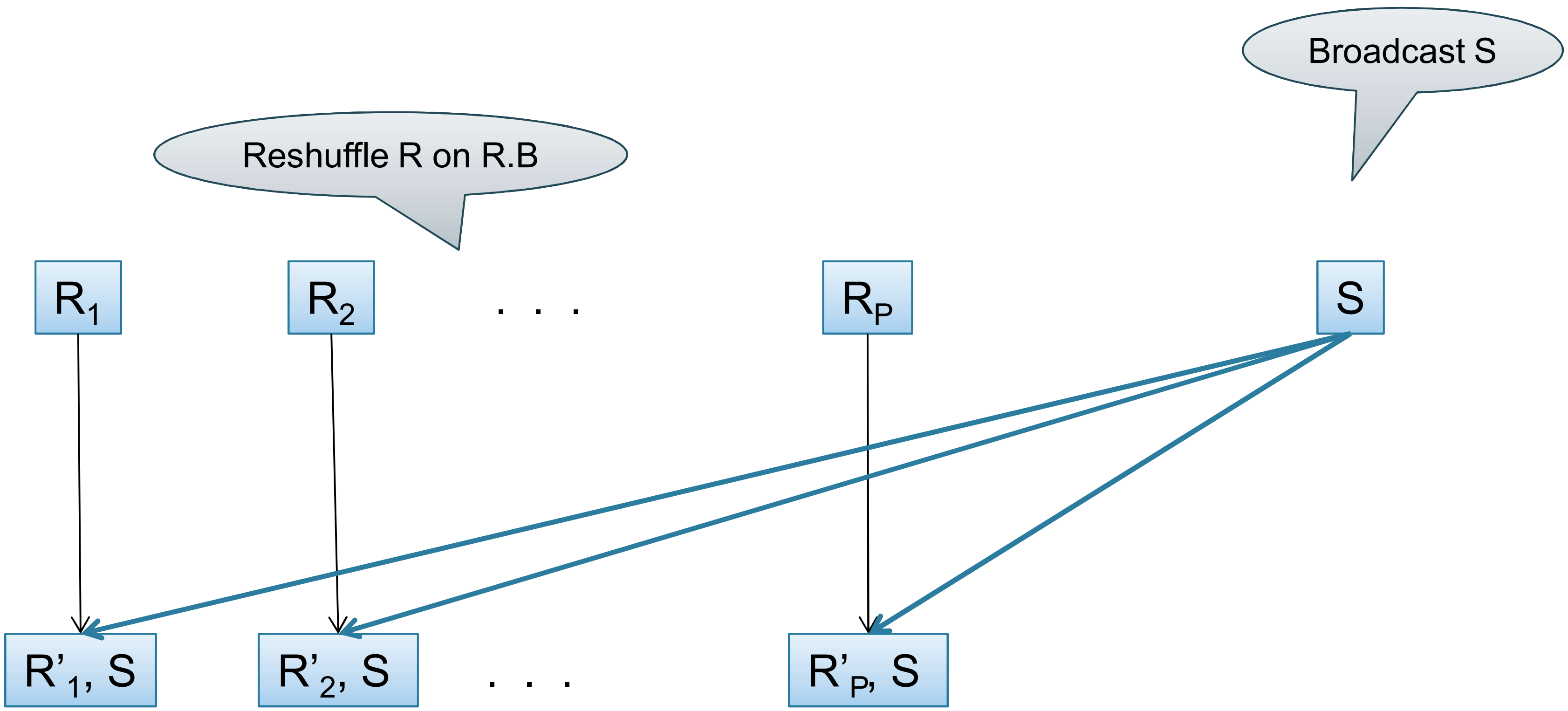
⋈

S2'

K2	B
101	50
102	50
202	50

Data: R(A, B), S(C, D)
Query: R(A,B) $\bowtie_{B=C}$ S(C,D)

Broadcast Join

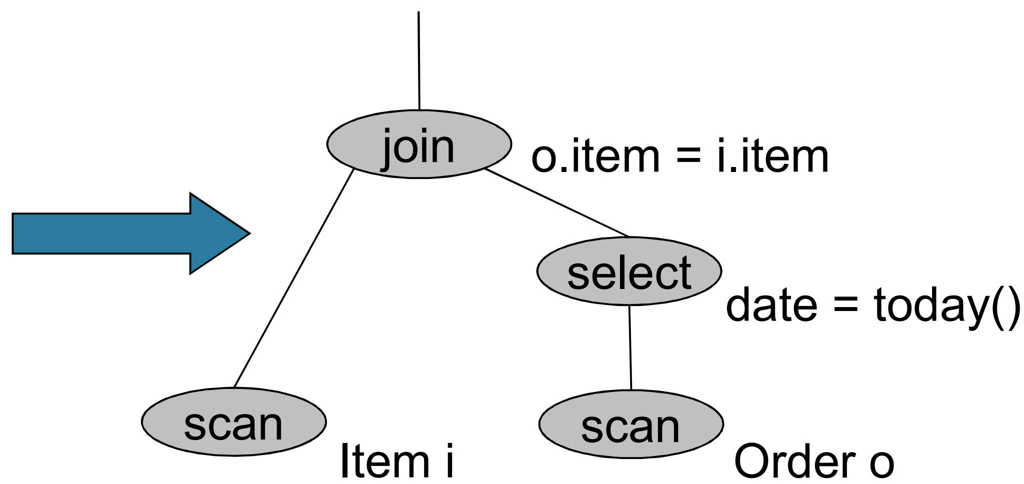


Why would you want to do this?

Putting it Together: Example Parallel Query Plan

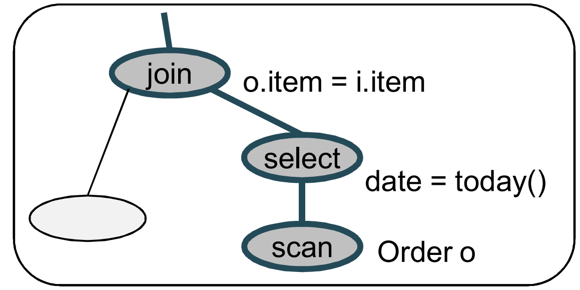
Find all orders from today, along with the items ordered

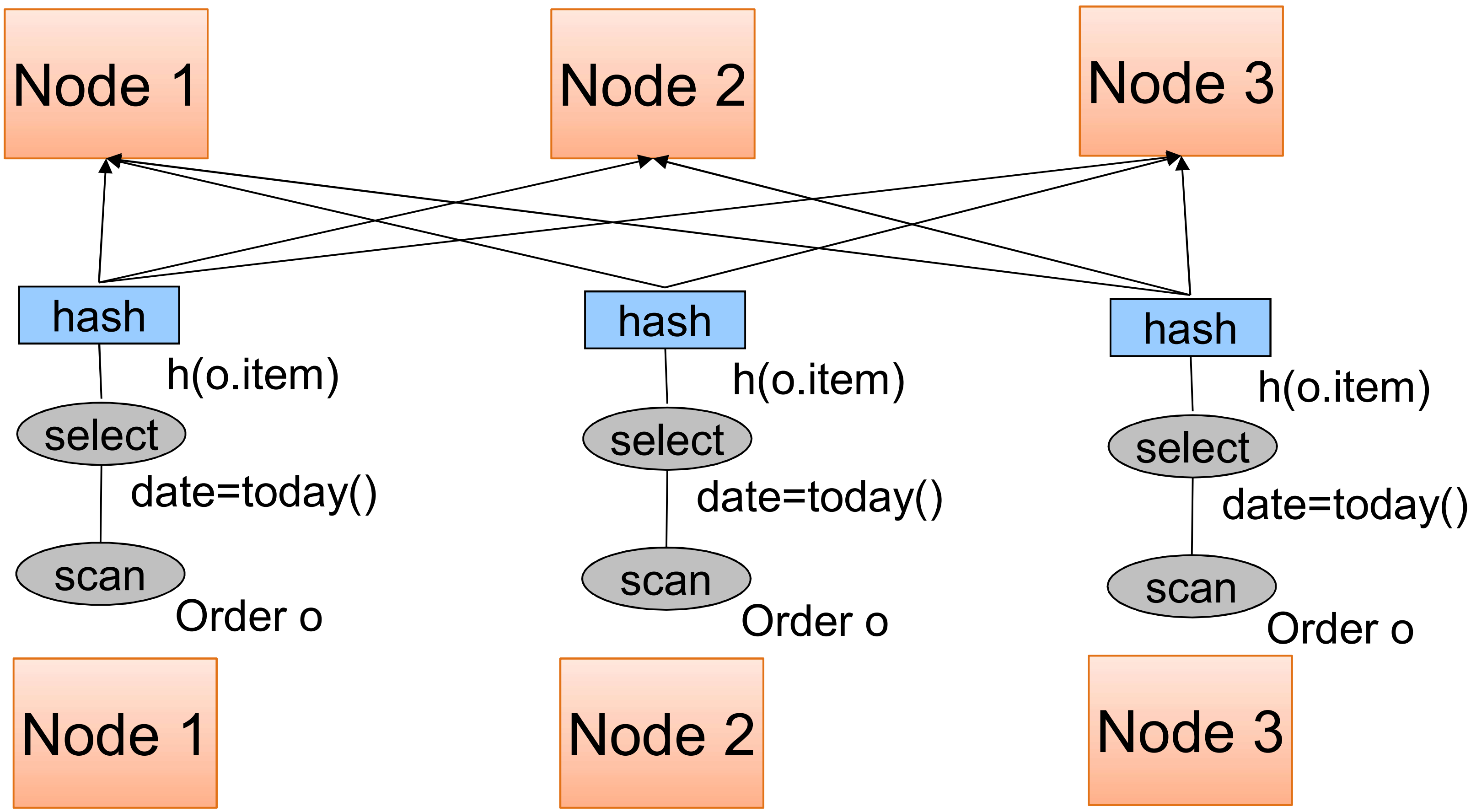
```
SELECT *
  FROM Order o, Line i
 WHERE o.item = i.item
   AND o.date = today()
```



Order(oid, item, date), Line(item, …)

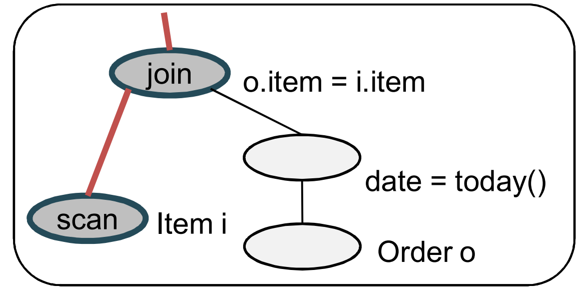
Example Parallel Query Plan

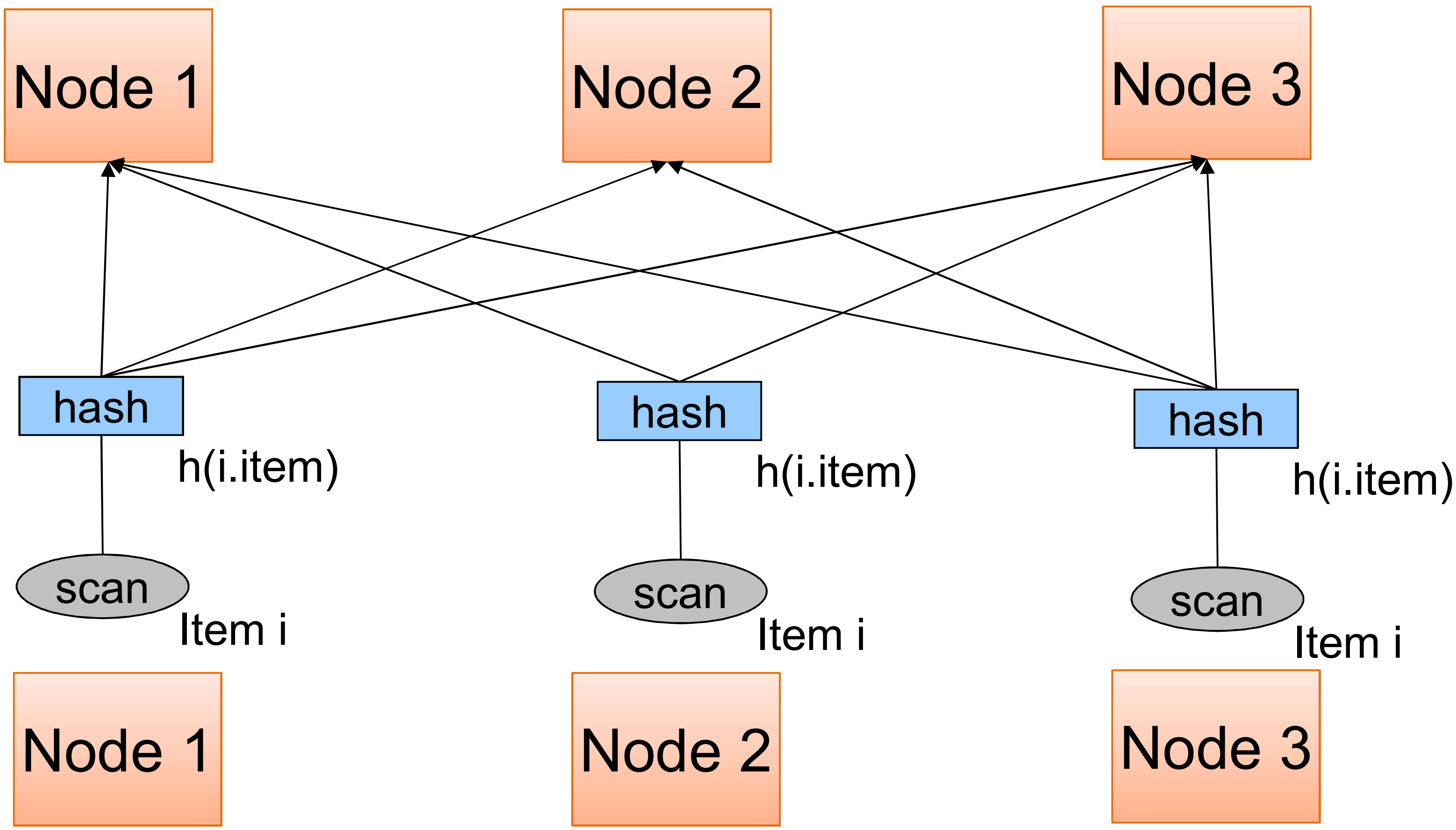




Order(oid, item, date), Line(item, …)

Example Parallel Query Plan





Order(oid, item, date), Line(item, …)

Example Parallel Query Plan