ADMINISTRIVIA

- **OQ5 Due Tonight (11:00)**
- **HW6 Due next Wednesday (Feb 28)**
- **HW4 Grades Out**
- **Cost-estimation problems**
  - Fair game for final exam
  - Look at previous midterms early
  - Also in last quarter’s final

"DO YOU KNOW WHAT'S HAPPENING IN 344?"

WHAT IS HAPPENING?

What we have been doing

- How to interface with data (SQL, Datalog, SQL++)
- Query execution on a single node (RA, cost estimation)

Next few lectures

- Parallel query execution across multiple nodes

Why do we care about how the query executes?

- Identify where to speed up (indexes)
- Identify where to eliminate bottlenecks

CSE 344

FEBRUARY 23RD – INTRO TO PARALLELISM

TODAY

Parallel architectures and querying options available

- Shared memory, shared disk, or **shared nothing**
- Inter-query, inter-operator, **intra-operator**

Execution on a shared-nothing, intra-operator model

- Data partitioning in distributed systems
- Grouping execution
- Join execution

WHY COMPUTE IN PARALLEL?

Access to multiple cores

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

“Big Data” size issue

- Too large to fit in main memory
- Too large to fit on a single disk
- Distributed query processing on 100x-1000x servers
- Accessible via cloud services (Azure, AWS, ...)

PERFORMANCE METRICS FOR PARALLEL DBMS

Nodes = processors, computers

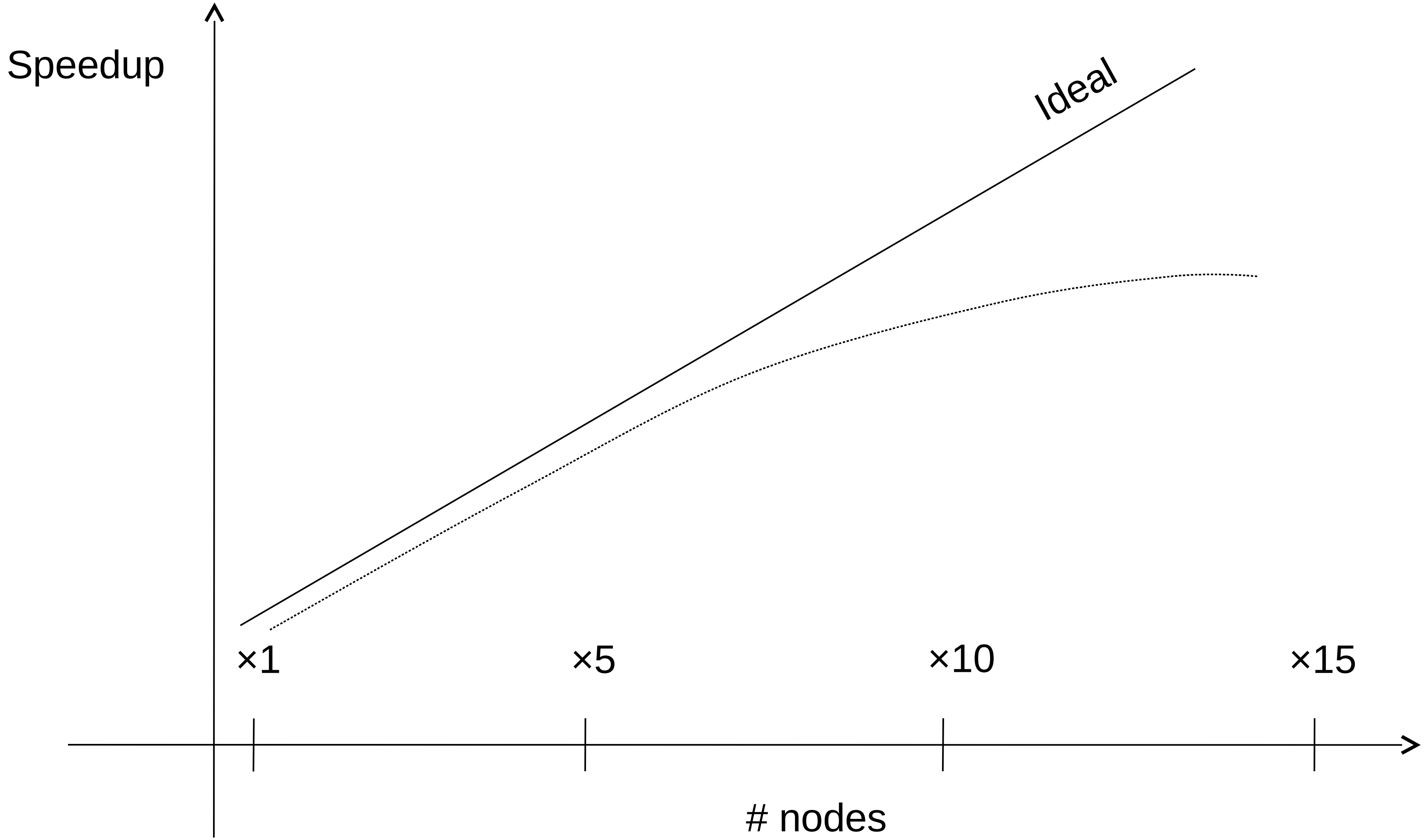
Speedup:

- More nodes, same data → higher speed

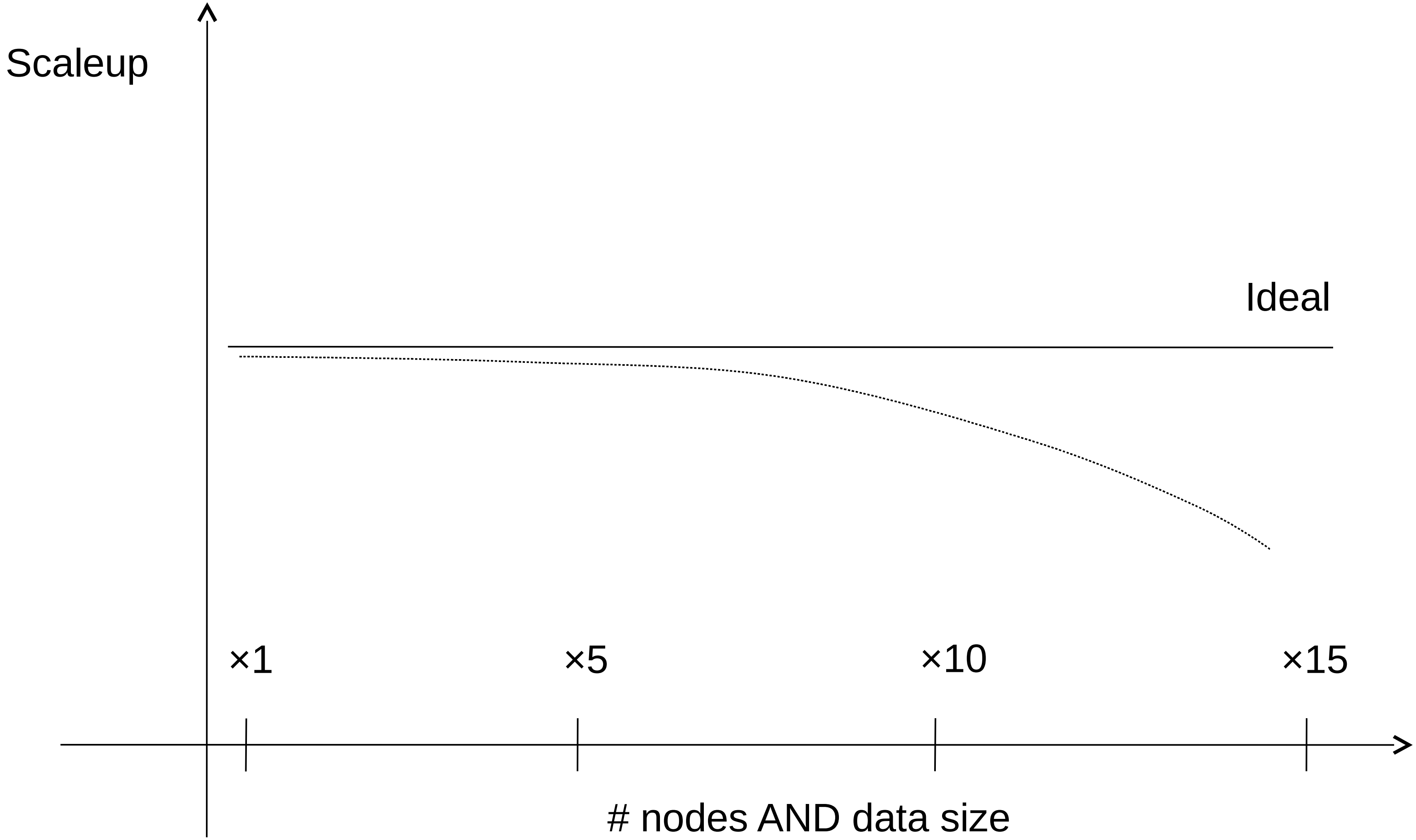
Scaleup:

- More nodes, more data → same speed

LINEAR V.S. NON-LINEAR SPEEDUP



LINEAR V.S. NON-LINEAR SCALEUP



WHY SUB-LINEAR SPEEDUP AND SCALEUP?

Overhead → Startup cost

- Cost of starting an operation on many nodes

Interference

- Contention for resources between nodes
- Waiting for other nodes to finish

Data distribution → Skew

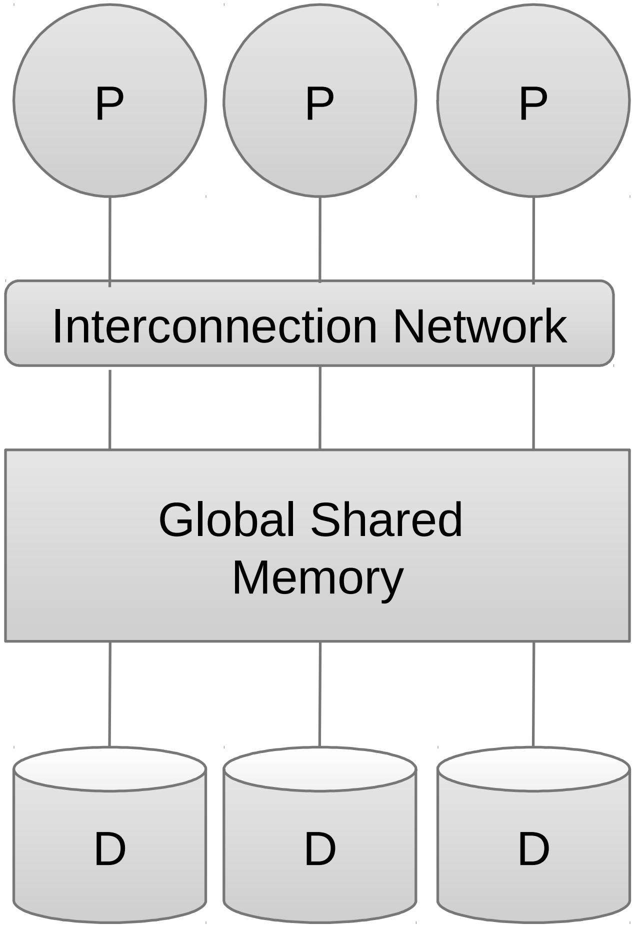
- Slowest node becomes the bottleneck

ARCHITECTURES FOR PARALLEL DATABASES

Solutions:

- **Shared memory**
- **Shared disk**
- **Shared nothing**

SHARED MEMORY



Nodes share both RAM and disk

Dozens to hundreds of processors

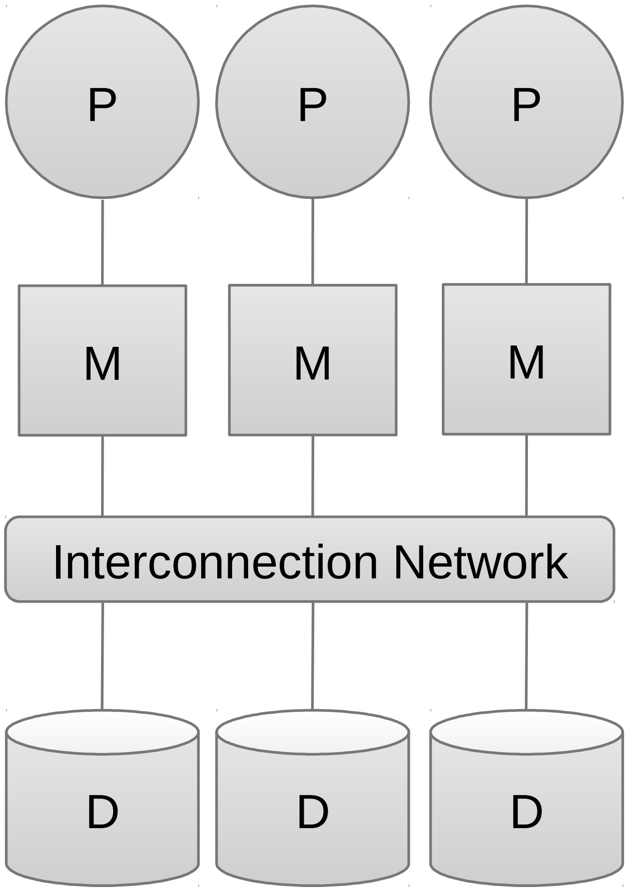
Example: Azure SQL Server

Check out HW3 query plans (SSMS/Datagrip)

Easy to use and program

Expensive to scale

SHARED DISK



All nodes access the same disks

Found in the largest "single-box" (non-cluster) multiprocessors

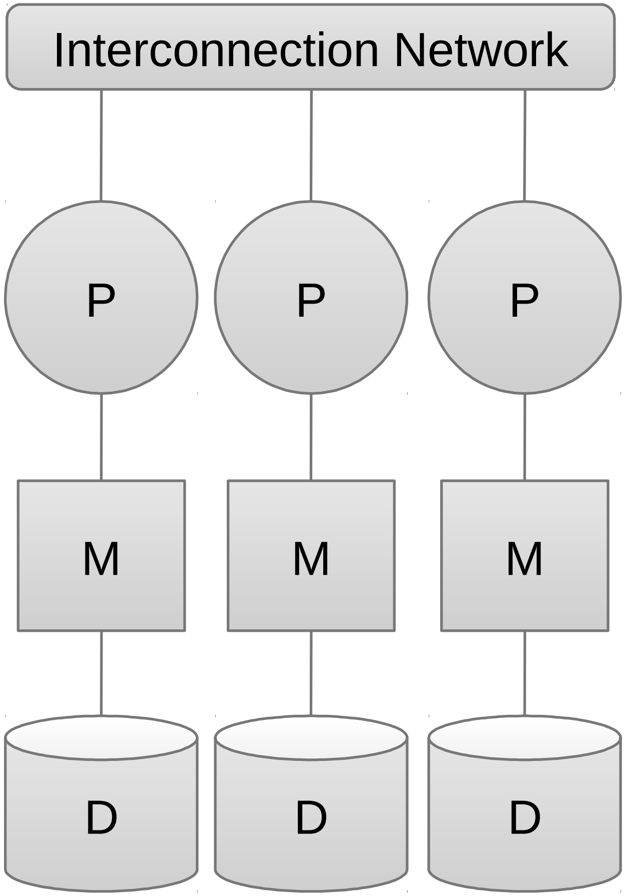
Example: Oracle

No need to worry about shared memory

Still hard to scale

Existing deployments typically have fewer than 10 machines

SHARED NOTHING



Cluster of commodity machines on high-speed network

Each machine has its own memory and disk: lowest contention.

Examples: Amazon EC2, Google Compute Engine

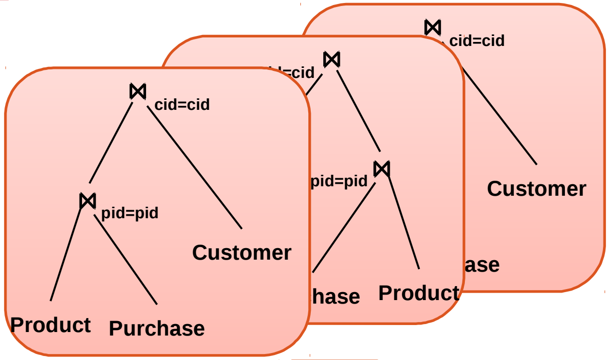
Easy to maintain and scale

Most difficult to administer and tune.

APPROACHES TO PARALLEL QUERY EVALUATION

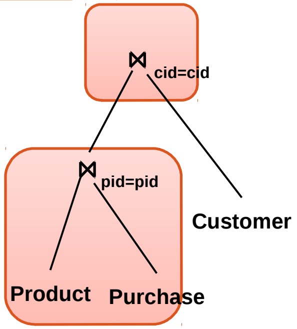
Inter-query parallelism

- Transaction per node
- Good for transactional workloads



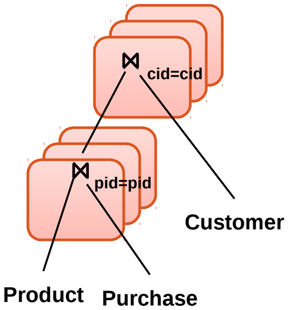
Inter-operator parallelism

- Operator per node
- Good for analytical workloads



Intra-operator parallelism

- Operator on multiple nodes
- Good for both?



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

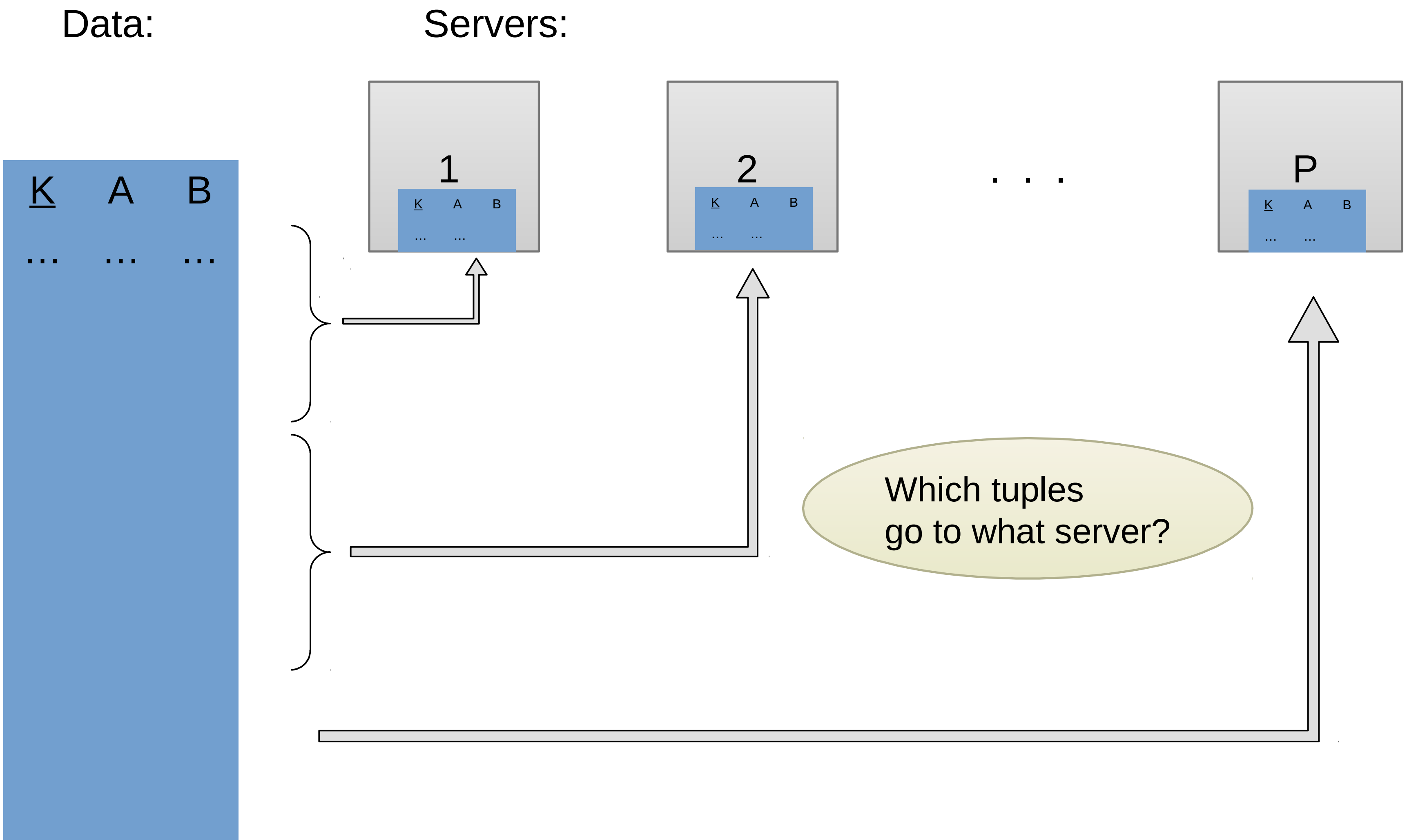
HORIZONTAL DATA PARTITIONING

Data:

<u>K</u>	A	B
…	…	…

Servers:

HORIZONTAL DATA PARTITIONING



HORIZONTAL DATA PARTITIONING

Block Partition:

- Partition tuples arbitrarily s.t. size(R_1)≈ … ≈ 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 is free in this class

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 — Uniform
- On the attribute A — May be skewed

Assuming good hash function

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

Keep this in mind in the next few slides

PARALLEL EXECUTION OF RA OPERATORS: GROUPING

Data: R(<u>K</u>,A,B,C)

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

How to compute group by if:

R is hash-partitioned on A ?

R is block-partitioned ?

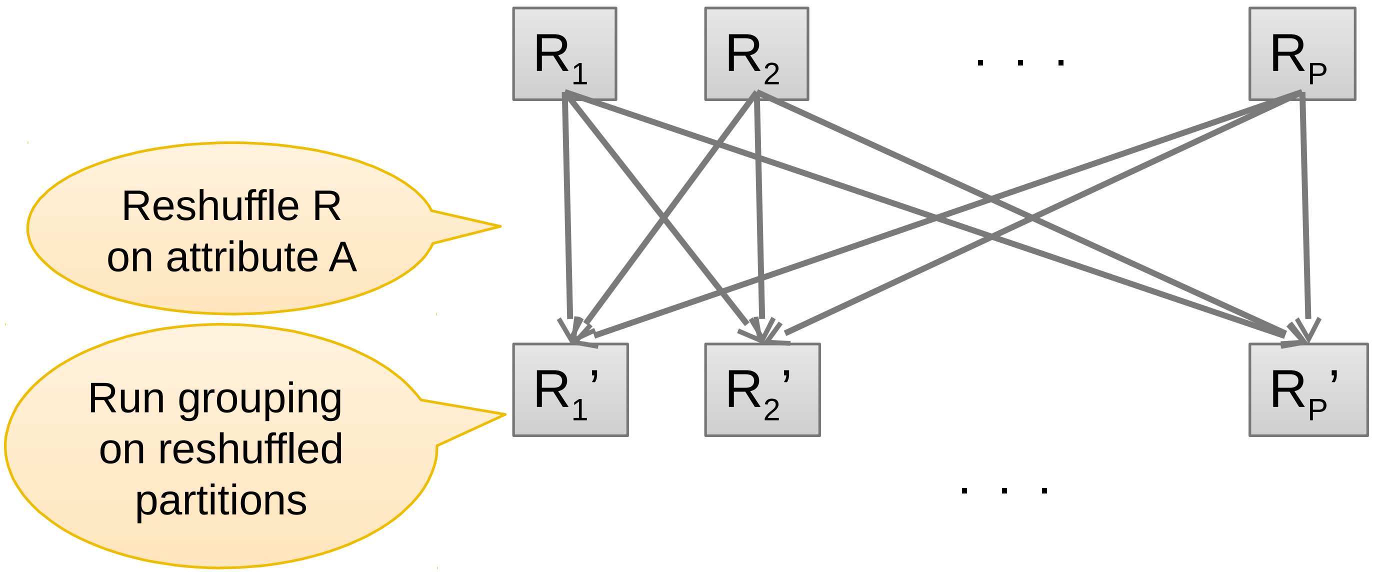
R is hash-partitioned on K ?

PARALLEL EXECUTION OF RA OPERATORS: GROUPING

Data: **R(<u>K</u>,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)

If we double both P and the size of R, what is the new running time?

- Same (each server holds the same # of chunks)

But only if the data is without skew!

SKEWED DATA

R(K,A,B,C)

Informally: we say that the data is skewed if one server holds much more data than the average

E.g. we hash-partition on A, and some value of A occurs very many times ("Justin Bieber")

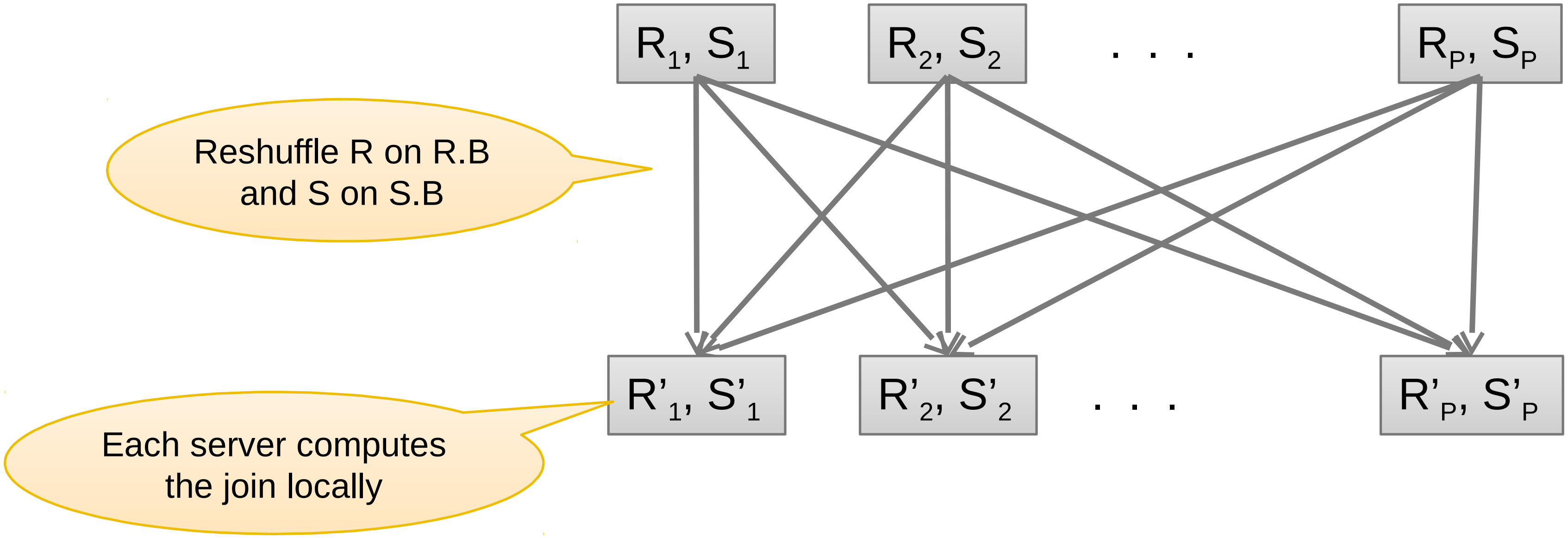
Then the server holding that value will be skewed

PARALLEL EXECUTION OF RA OPERATORS: PARTITIONED HASH-JOIN

Data: R(<u>K1</u>, A, B), S(<u>K2</u>, B, C)

Query: R(<u>K1</u>, A, B) ⋈ S(<u>K2</u>, B, C)

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



Data: R(<u>K1</u>,A, B), S(<u>K2</u>, B, C)

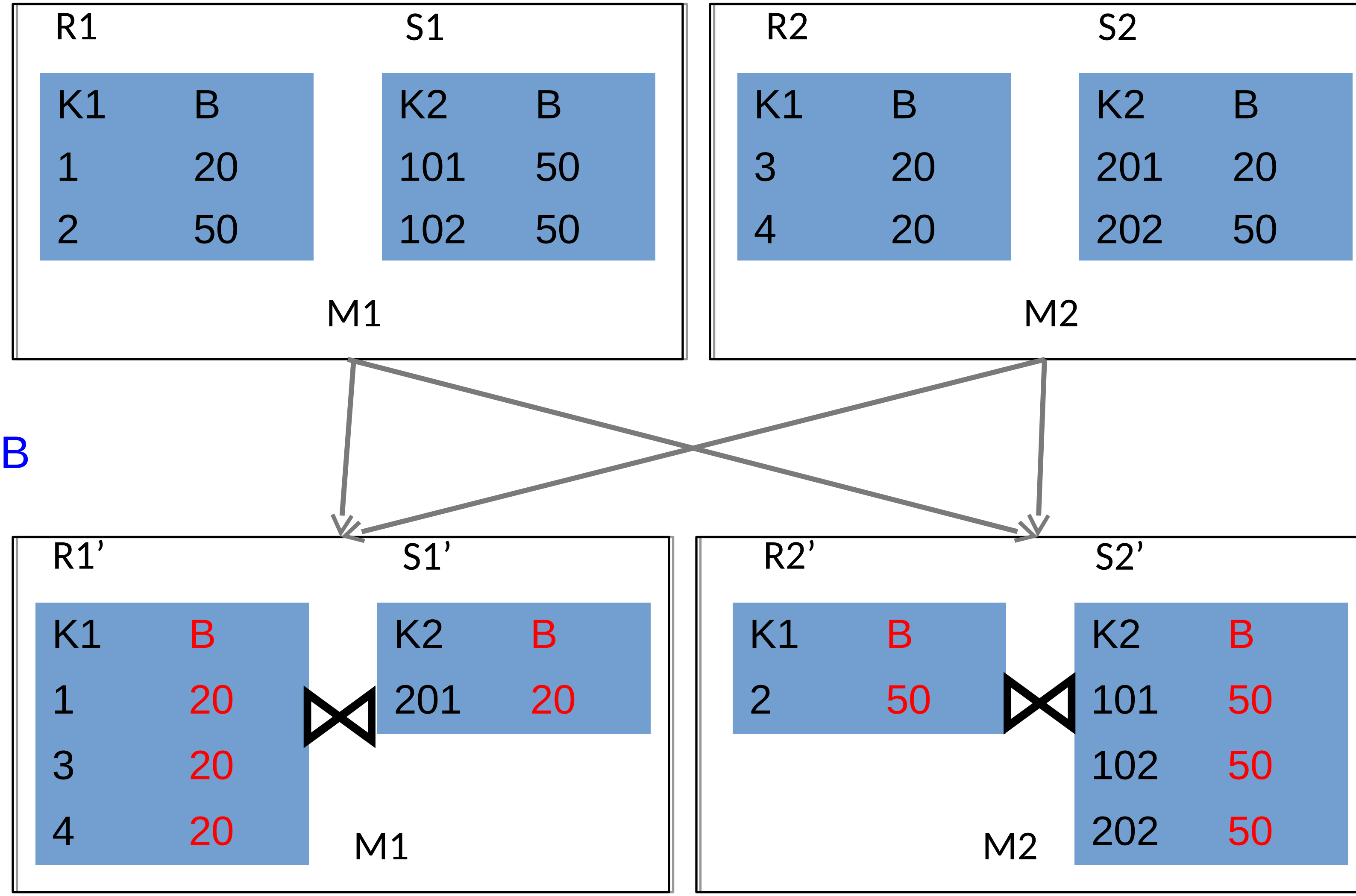
Query: R(<u>K1</u>,A,B) ⋈ S(<u>K2</u>,B,C)

PARALLEL JOIN ILLUSTRATION

Partition

Shuffle on B

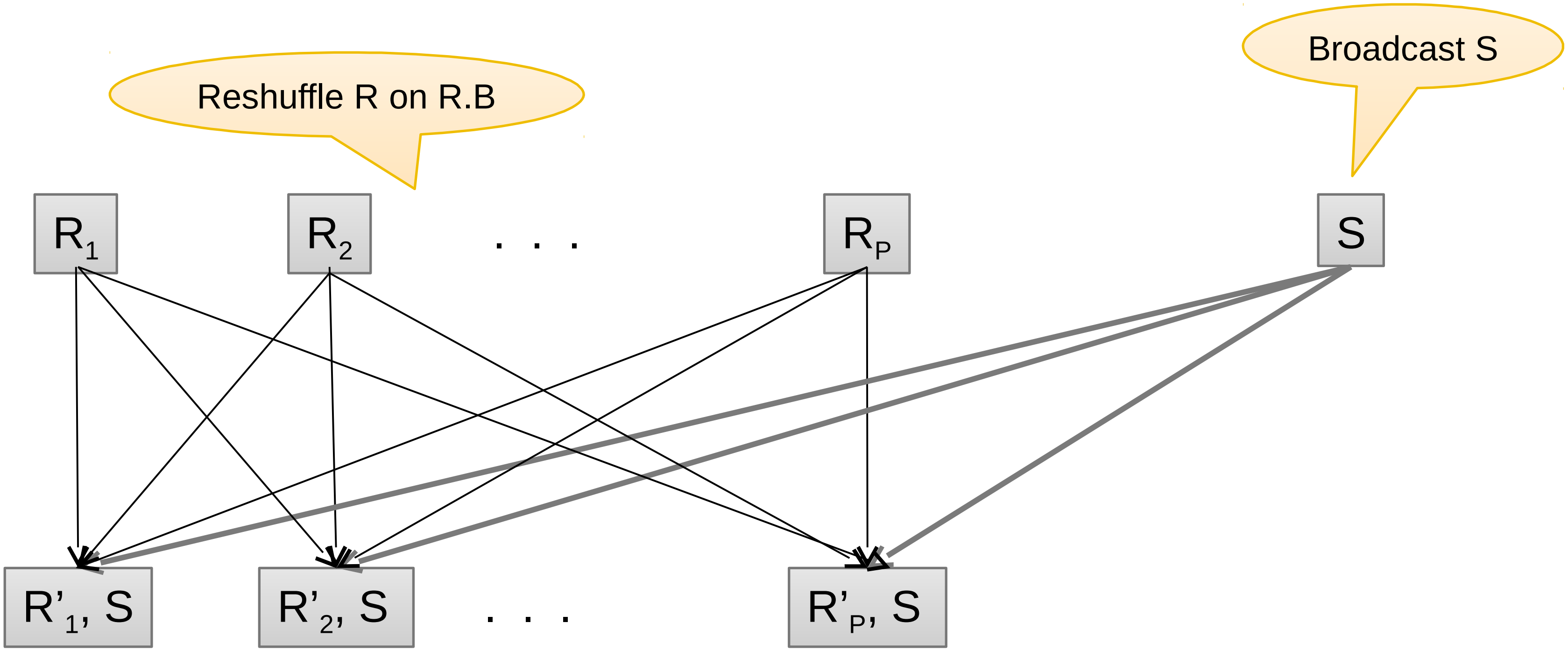
Local Join



Data: R(A, B), S(C, D)

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

BROADCAST JOIN



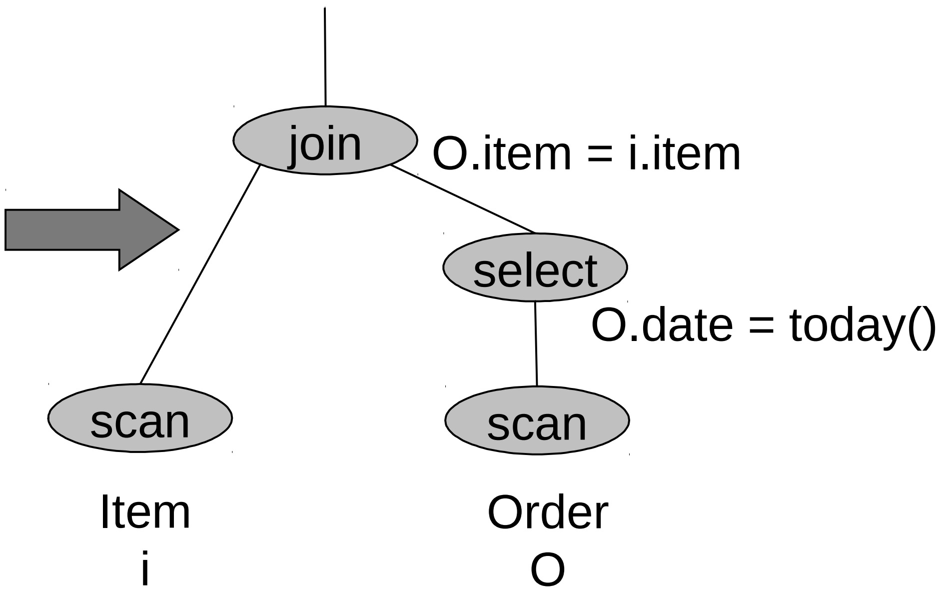
Why would you want to do this?

Order(oid, item, date), Item(item, …)

EXAMPLE PARALLEL QUERY PLAN

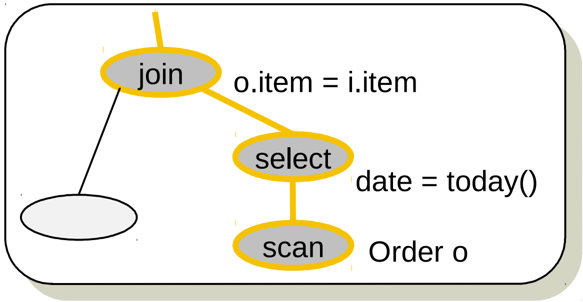
Find all orders from today, along with the items ordered

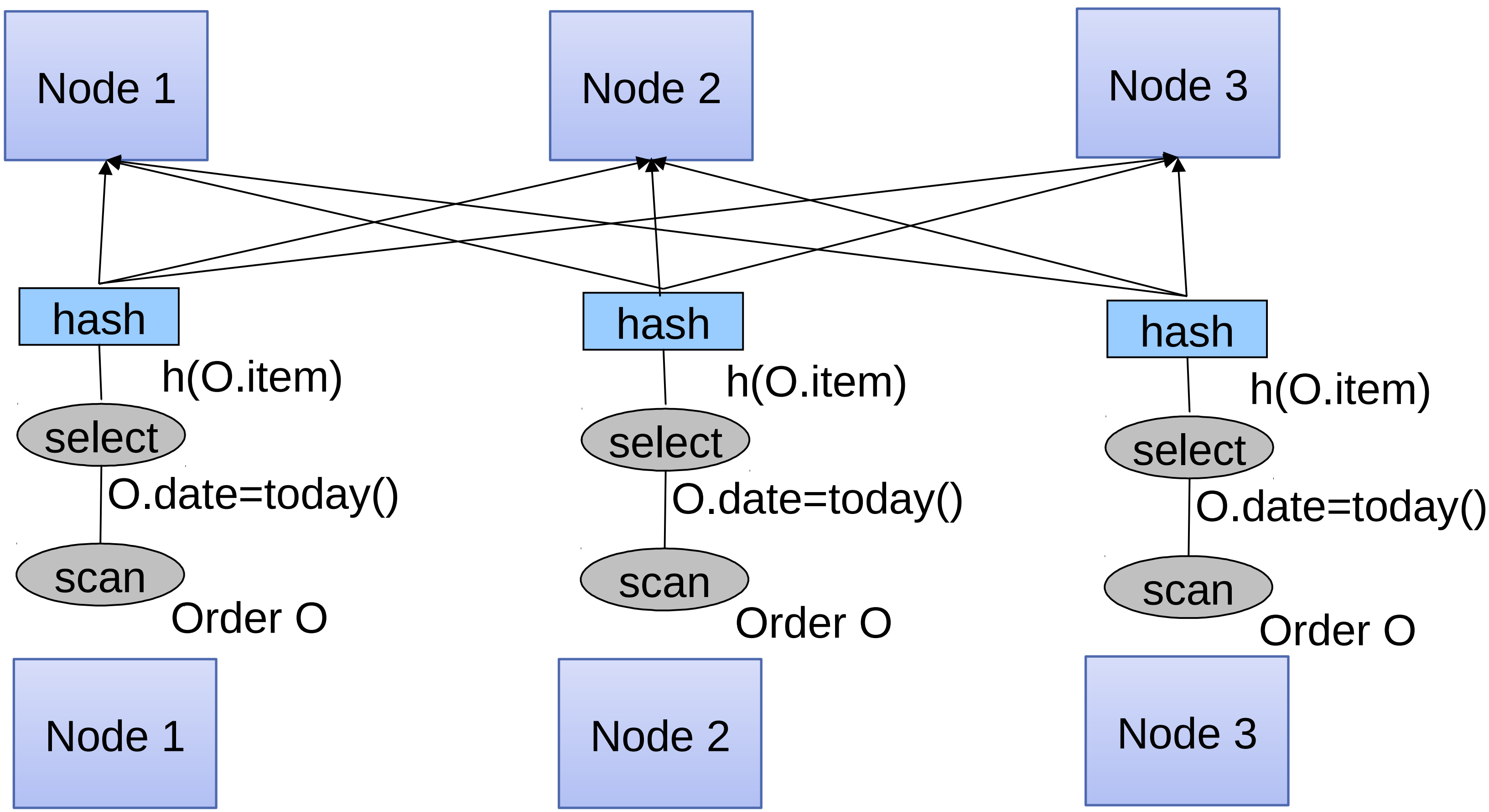
```
SELECT *
  FROM Order AS O, Item AS i
 WHERE O.item = i.item
   AND O.date = today()
```



Order(oid, item, date), Line(item, …)

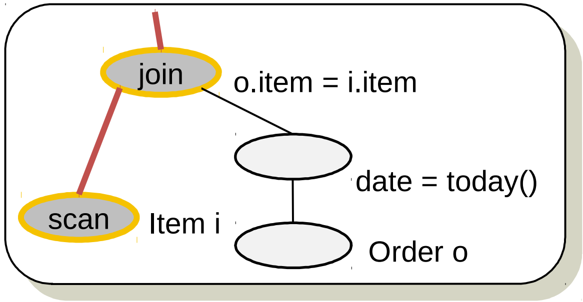
PARALLEL QUERY PLAN

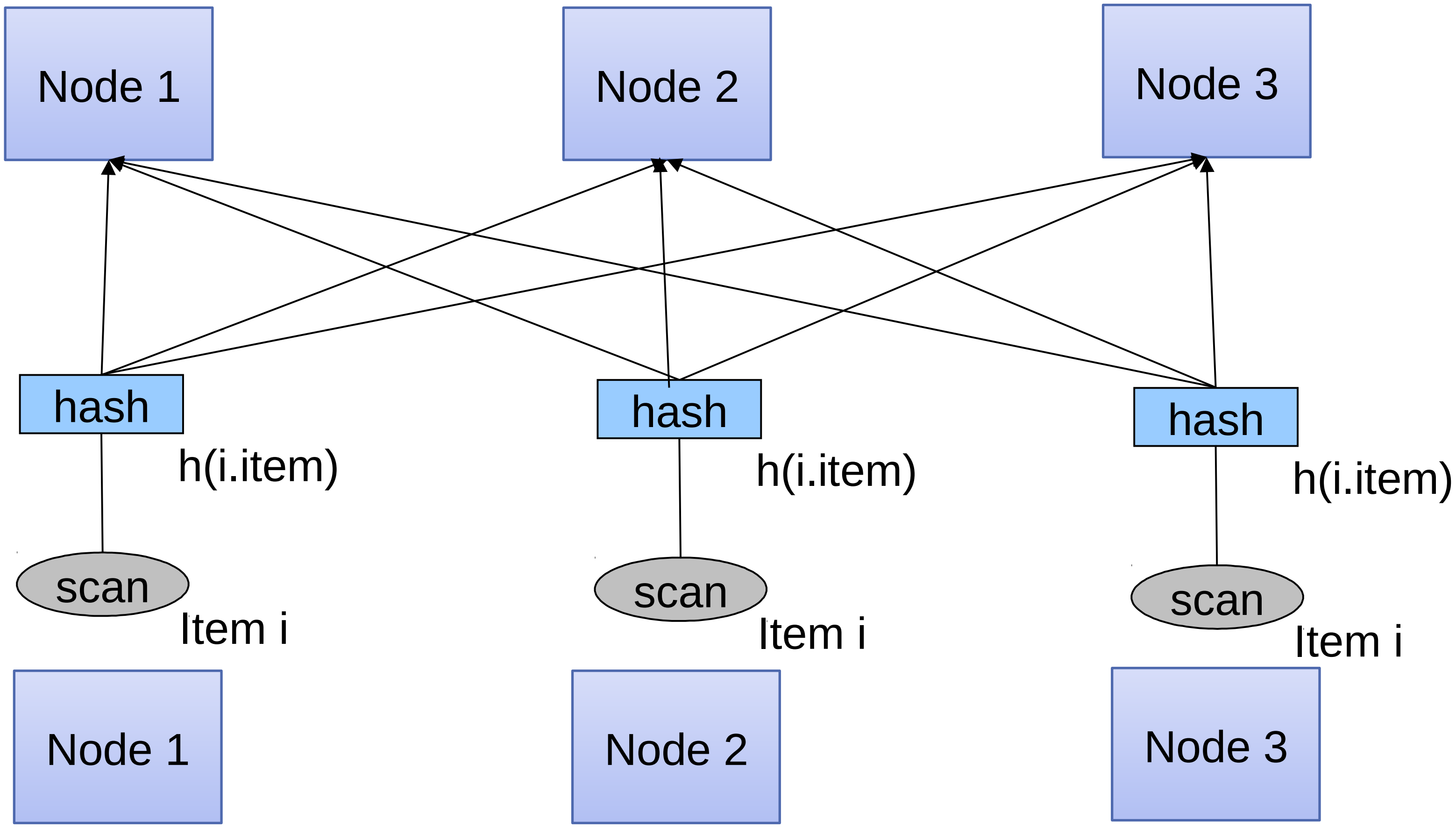




Order(oid, item, date), Line(item, …)

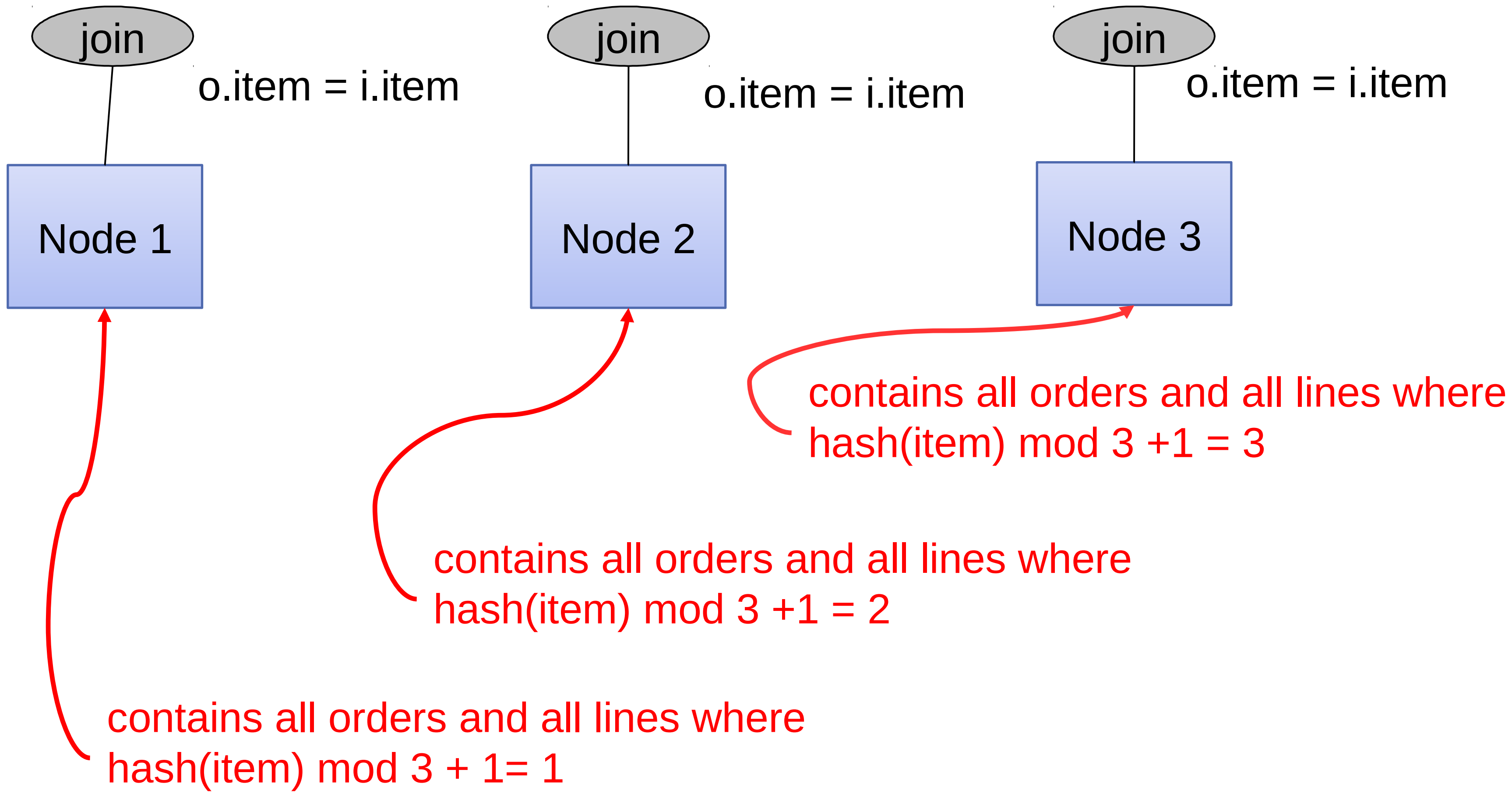
PARALLEL QUERY PLAN





Order(oid, item, date), Line(item, …)

PARALLEL QUERY PLAN



A CHALLENGE

Have P number of servers (e.g. P=1000)

How do we compute this Datalog query in one step?

Q(x,y,z) :- R(x,y), S(y,z), T(z,x)

HYPERCUBE JOIN

Have P number of servers (e.g. P=1000)

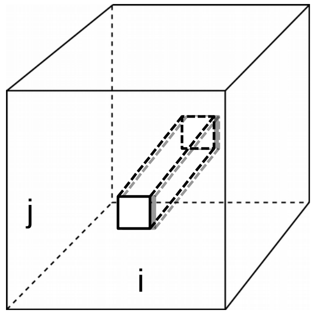
How do we compute this Datalog query in one step?
Q(x,y,z) = R(x,y),S(y,z),T(z,x)

Organize the P servers into a cube with side $P^{1/3}$

- Thus, each server is uniquely identified by (i,j,k), i,j,k≤$P^{1/3}$

Step 1:

- Each server sends R(x,y) to all servers (h(x),h(y),*)
- Each server sends S(y,z) to all servers (*,h(y),h(z))
- Each server sends T(x,z) to all servers (h(x),*,h(z))



Final output:

- Each server (i,j,k) computes the query R(x,y),S(y,z),T(z,x) locally

Analysis: each tuple R(x,y) is replicated at most $P^{1/3}$ times