Parallel Databases

When/why do we need them?

Two Kinds to Parallel Data Processing

- Parallel databases, developed starting with the 80s (this lecture)
  - OLTP (Online Transaction Processing)
  - OLAP (Online Analytic Processing, or Decision Support)
- General purpose distributed processing: MapReduce, Spark
  - Mostly for Decision Support Queries

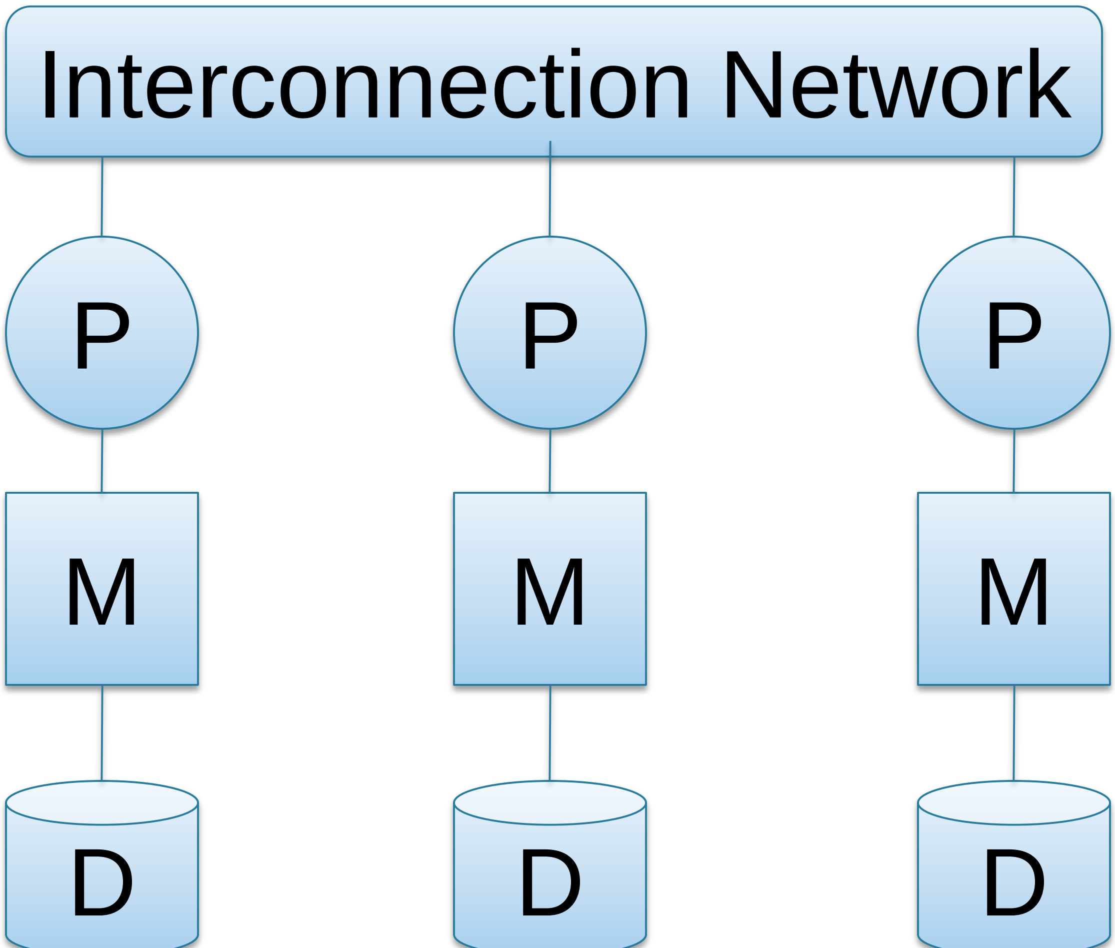
Performance Metrics for Parallel DBMSs

P = the number of nodes (processors, computers)

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

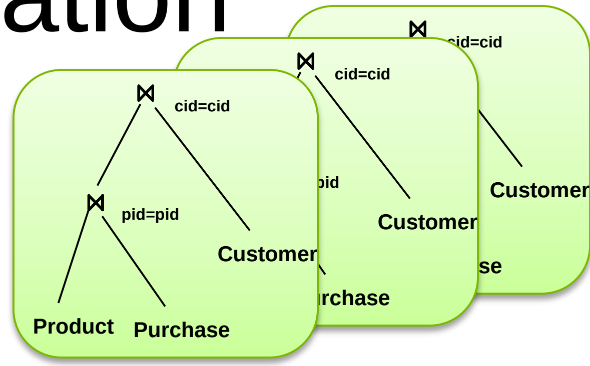
- OLTP: “Speed” = transactions per second (TPS)
- Decision Support: “Speed” = query time

Shared Nothing

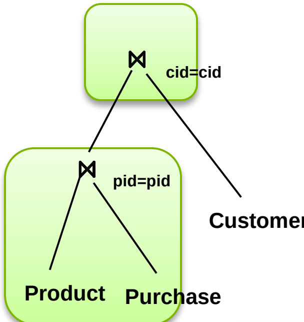


Approaches to Parallel Query Evaluation

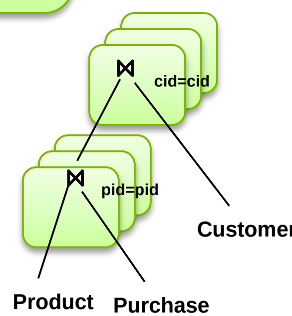
- Inter-query parallelism
  – Transaction per node
  – OLTP



- Inter-operator parallelism
  – Operator per node
  – Both OLTP and Decision Support



- Intra-operator parallelism
  – Operator on multiple nodes
  – Decision Support



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 attr. 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 attr. B as key
  - Scan file R, probe the hash table using attr. B

Distributed Query Processing

- Data is horizontally partitioned on many servers
- Operators may require data reshuffling

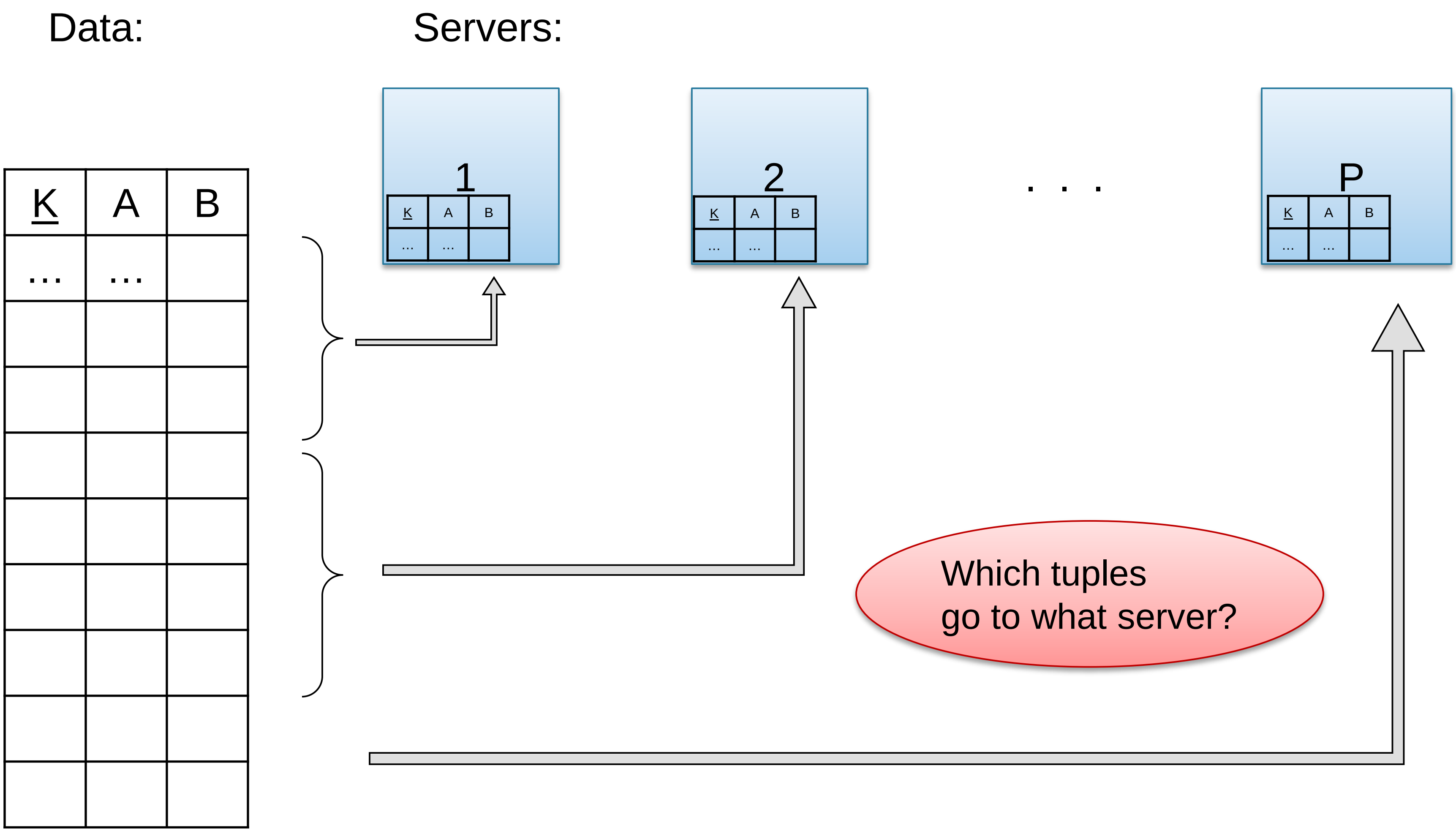
Horizontal Data Partitioning

Data:

K	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
- 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 $v_{i-1} < t.A < v_i$

Parallel GroupBy

Data: R(<u>K</u>,A,B,C)
Query: $\gamma_{A,sum(C)}(R)$
Discuss in class how to compute in each case:

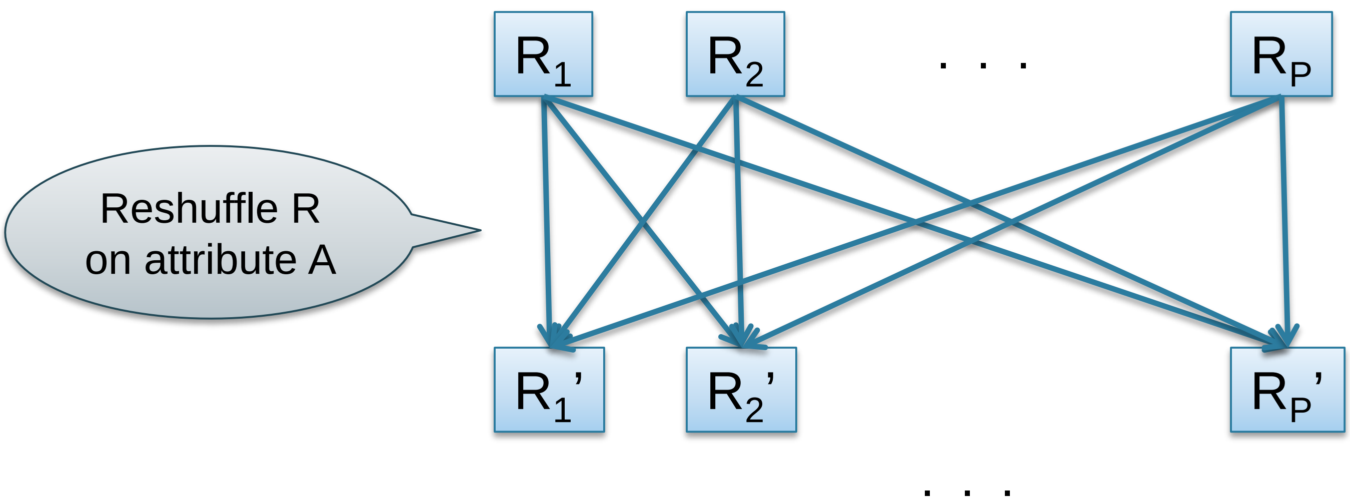
- R is hash-partitioned on A
- R is block-partitioned
- R is hash-partitioned on K

Parallel GroupBy

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

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

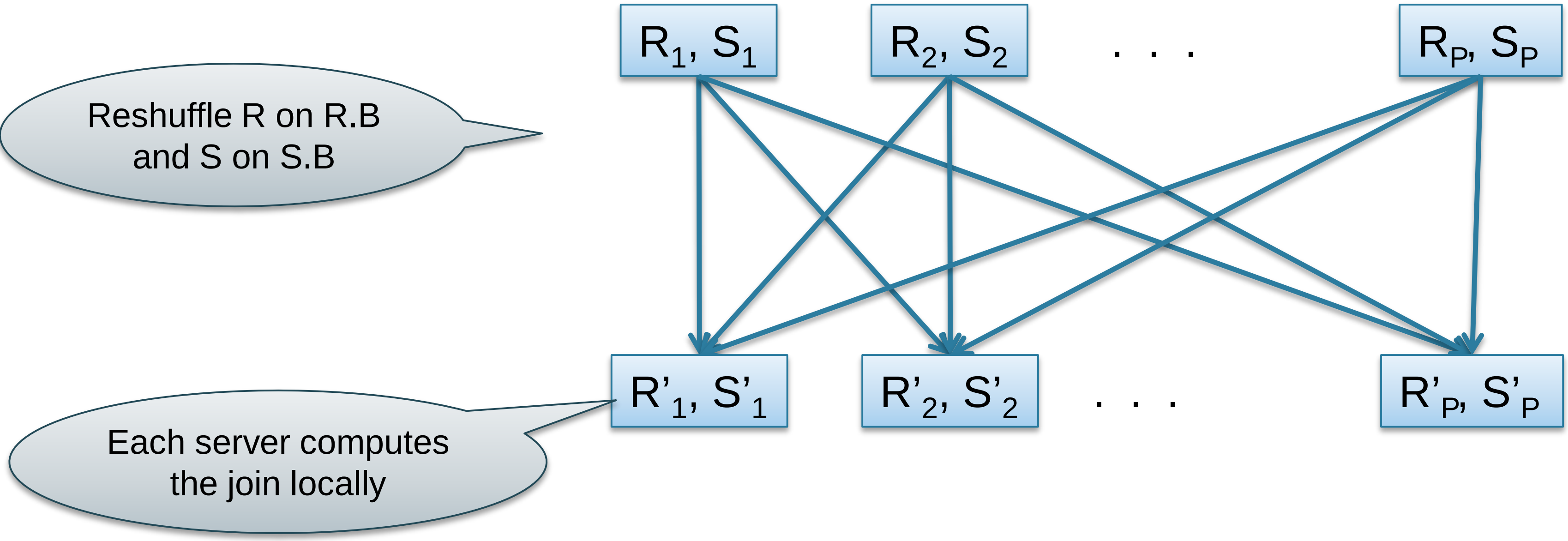
- R is block-partitioned or hash-partitioned on K



Parallel Join

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

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



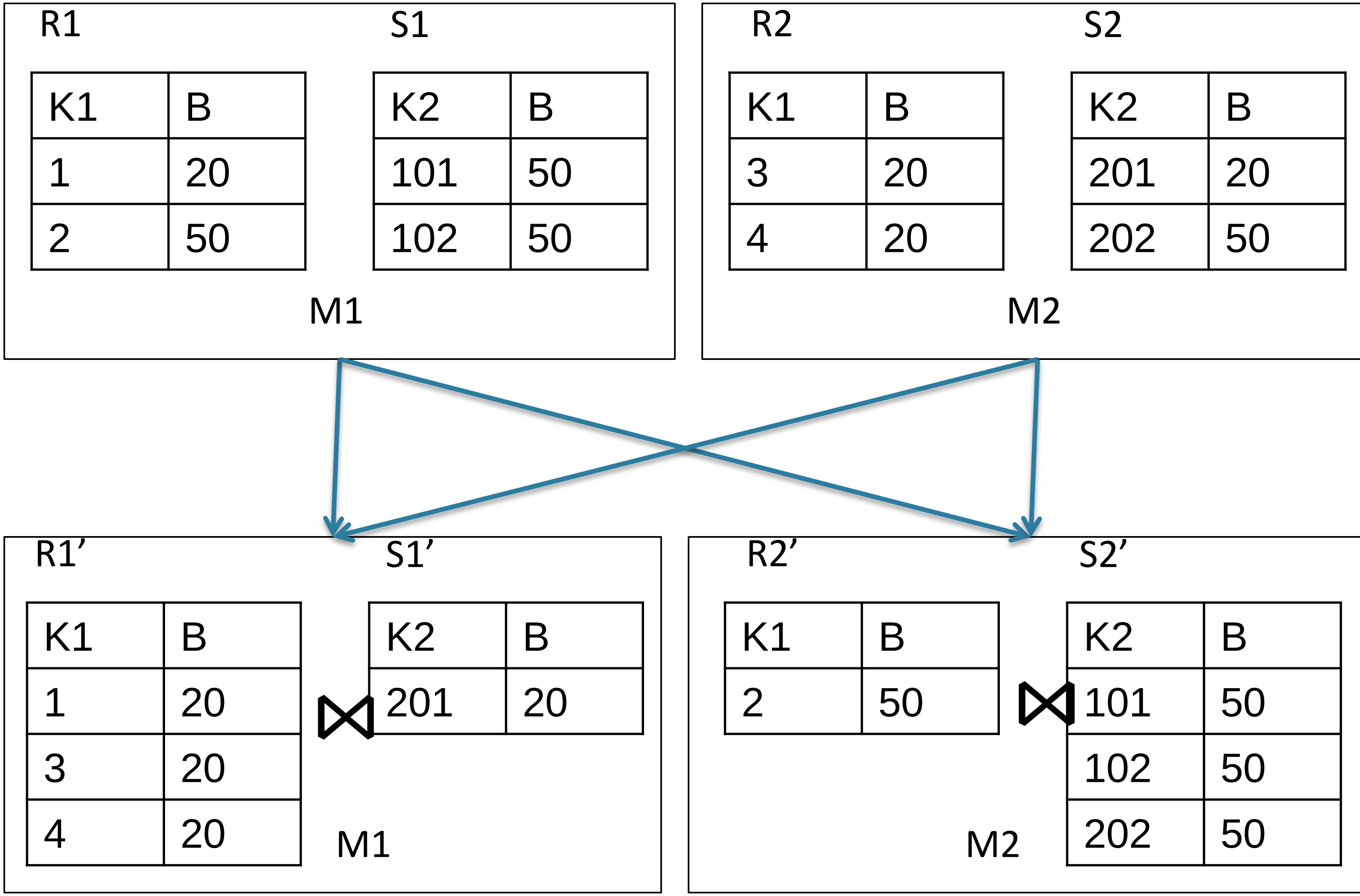
Data: R(K1,A, B), S(K2, B, C)

Query: R(K1,A,B) ⋈ S(K2,B,C)

Partition

Shuffle

Local Join



Speedup and Scaleup

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

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 good hash function

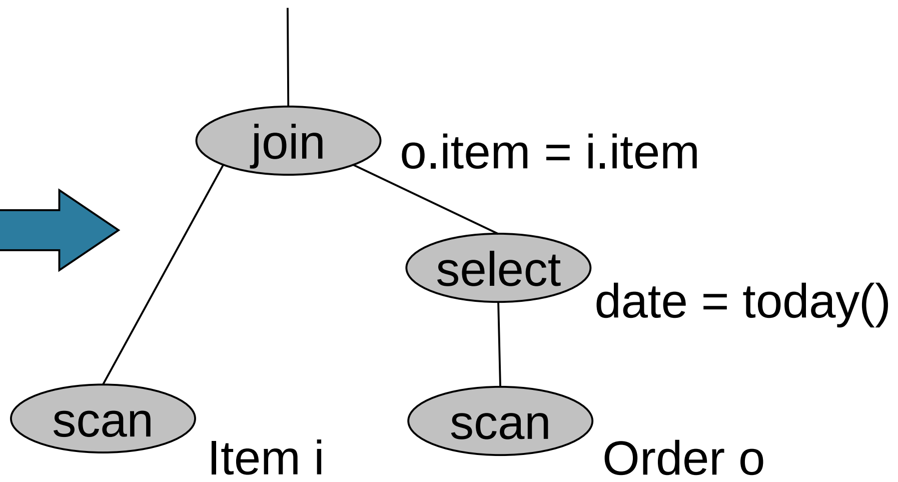
May be skewed

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

Example Parallel Query Execution

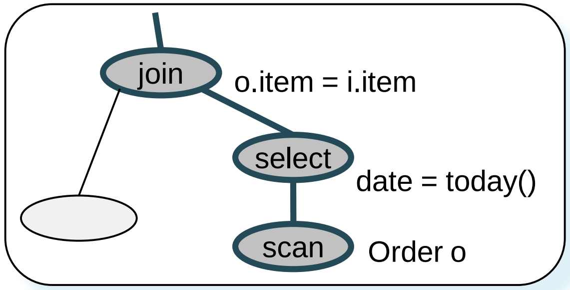
Find all orders from today, along with the items ordered

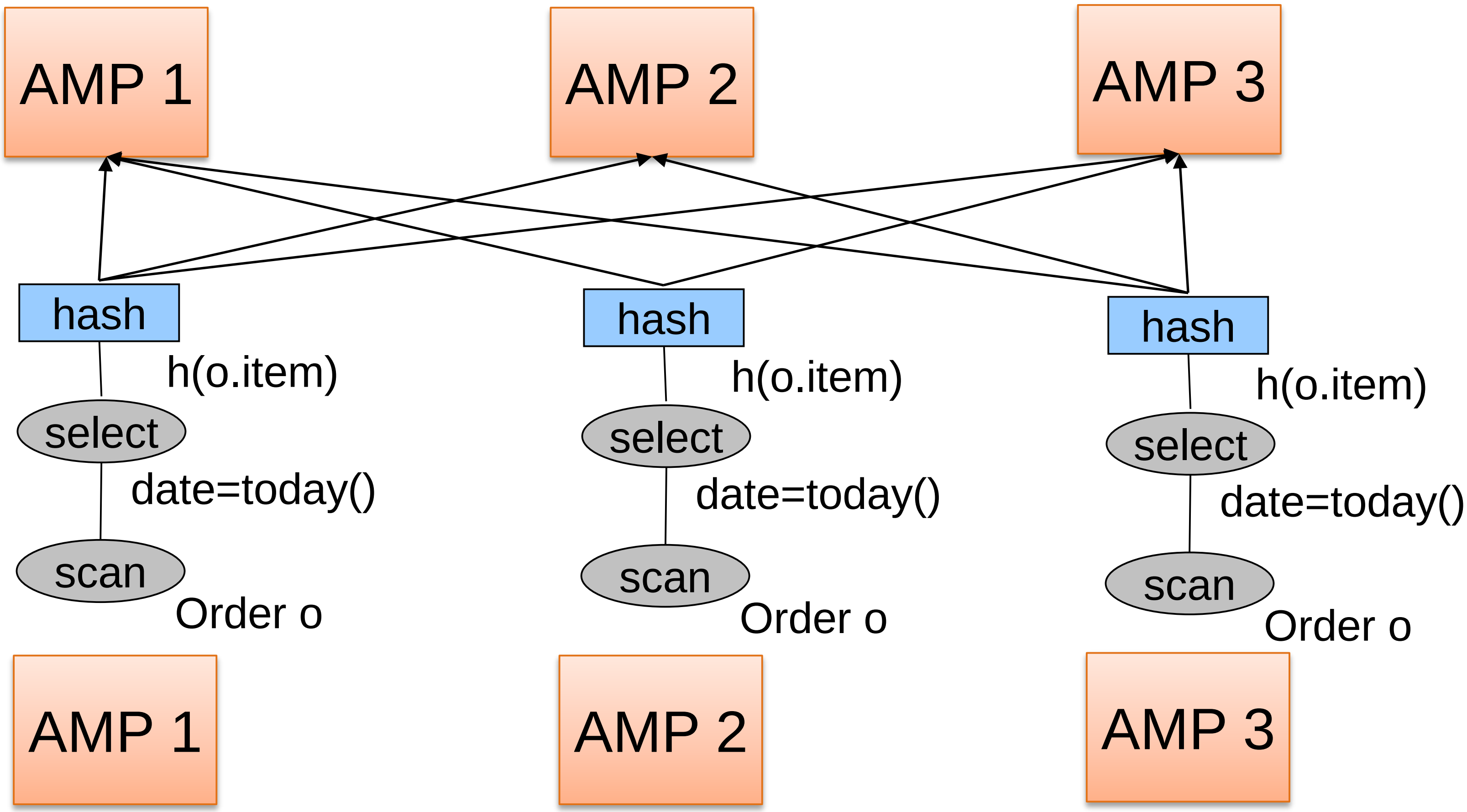
```
SELECT *
  FROM Order o, Line i
 WHERE o.item = i.item
   AND o.date = today()
```



Order(oid, item, date), Line(item, …)

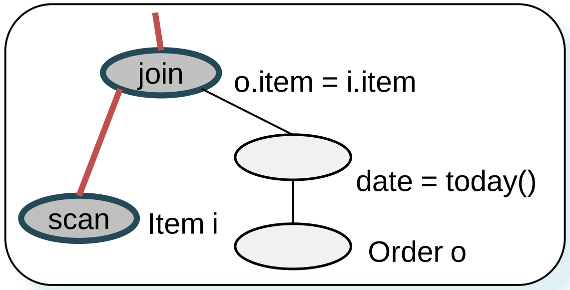
Example Parallel Query Execution

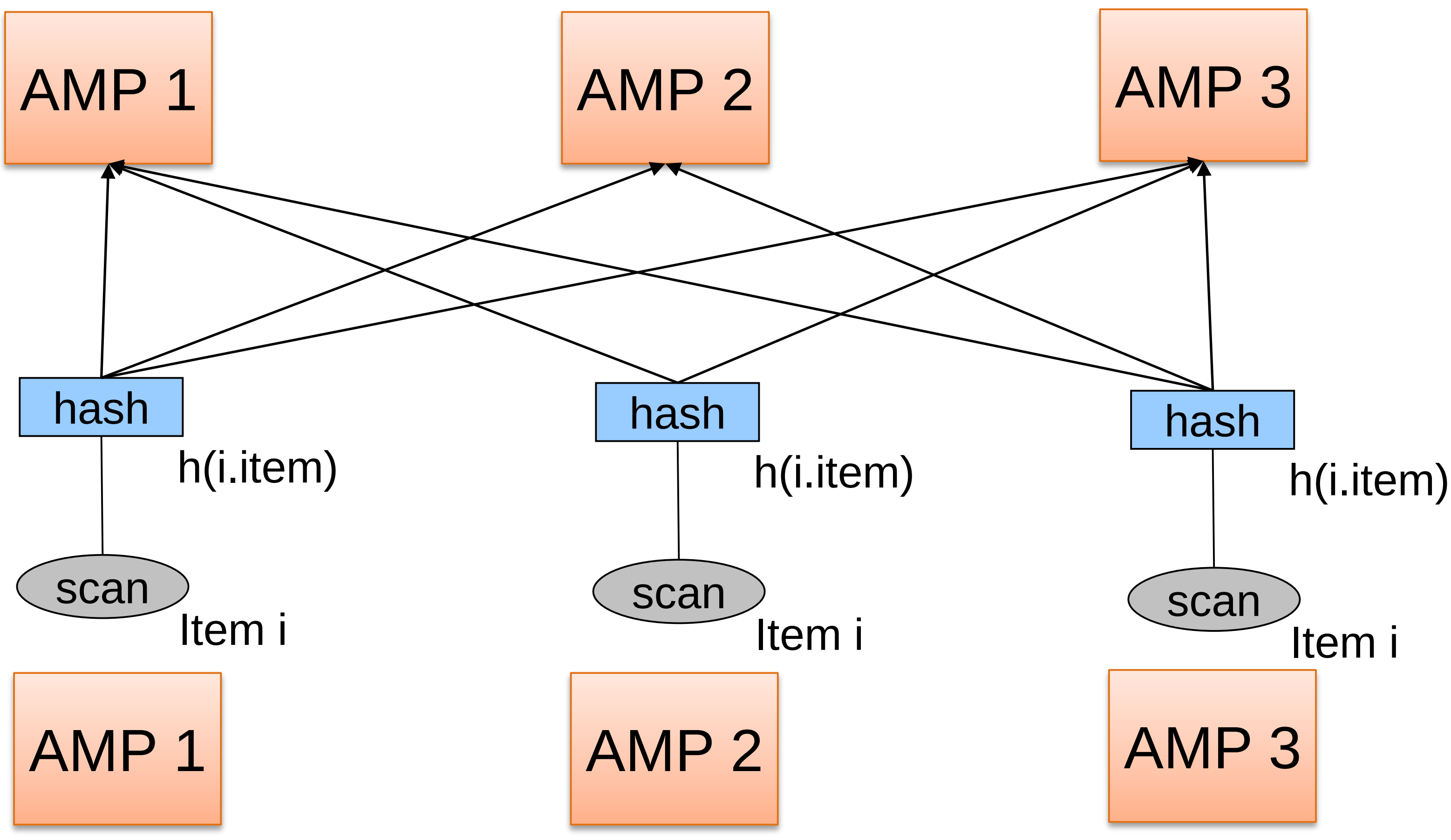




Order(oid, item, date), Line(item, …)

Example Parallel Query Execution





Order(oid, item, date), Line(item, …)

Example Parallel Query Execution