CSE 544: Principles of Database Systems

Parallel Databases

Announcements

- Paper reviews:
  - Join processing paper was due yesterday
  - MapReduce paper due on Monday, May 6th
- HW2 is due on Monday, May 6th
  - You should have made lots of progress by now!

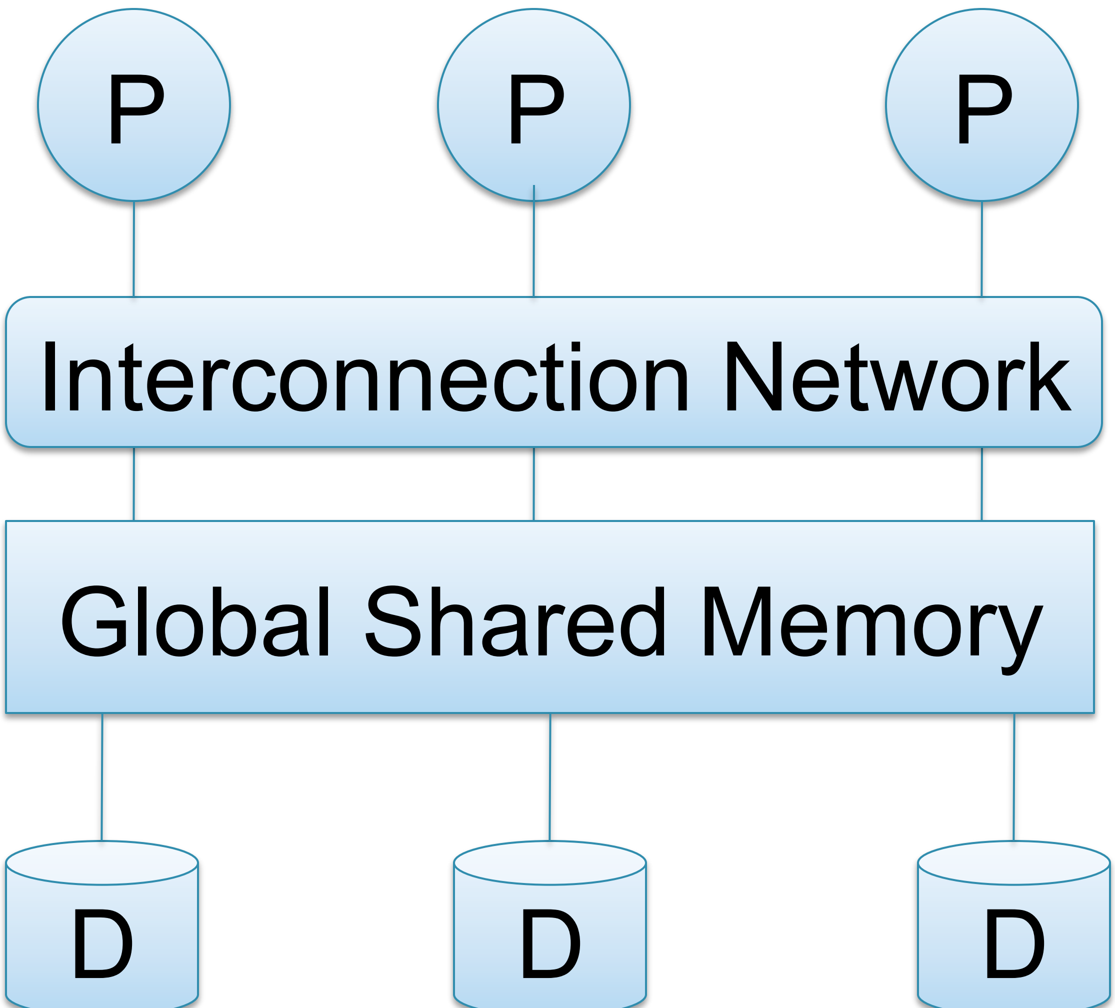
Overview of Today's Lecture

- Parallel databases (Chapter 22.1 – 22.5)
- MapReduce – base on the paper

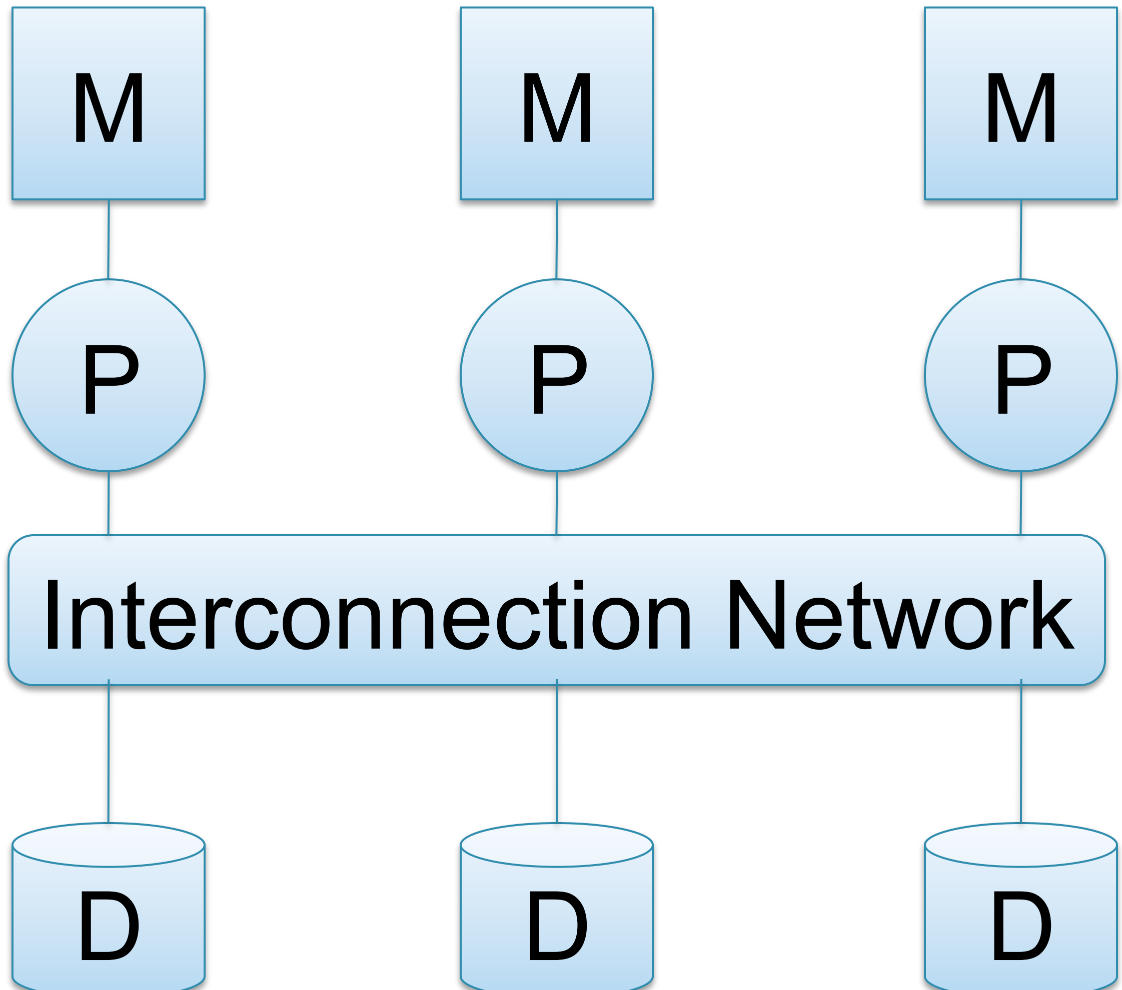
Architectures for Parallel Databases

- Shared memory
- Shared disk
- Shared nothing

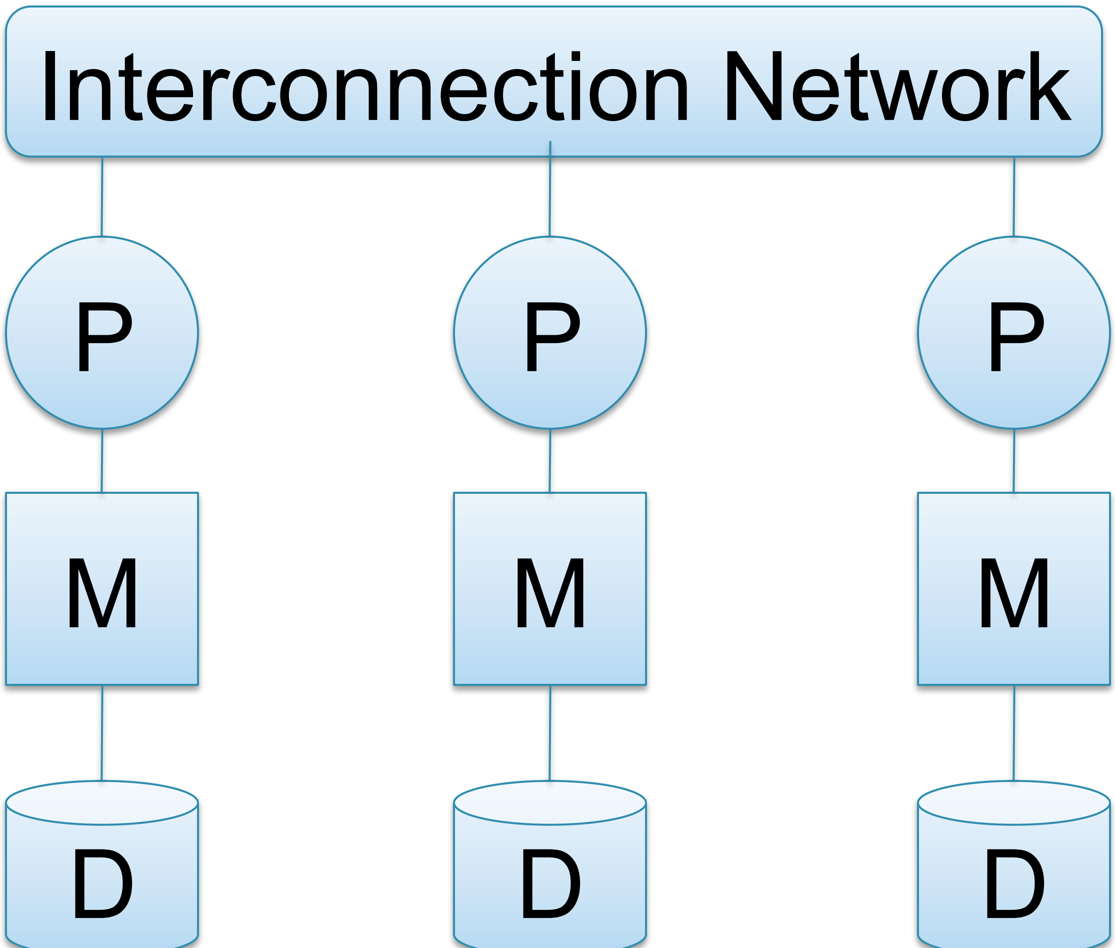
Shared Memory



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 get a query to run faster (see query plans)

- Easy to use and program
- But very expensive to scale: last remaining cash cows in the hardware industry

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.

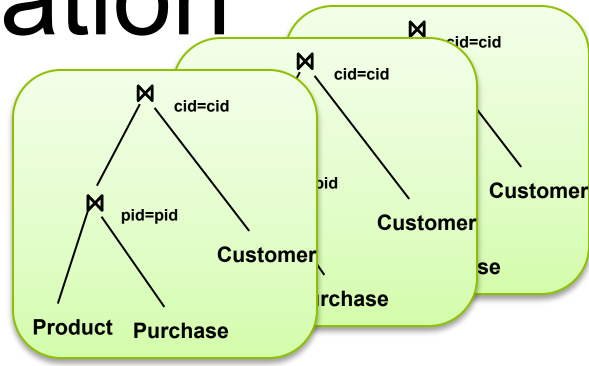
We discuss only Shared Nothing in class

In Class

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

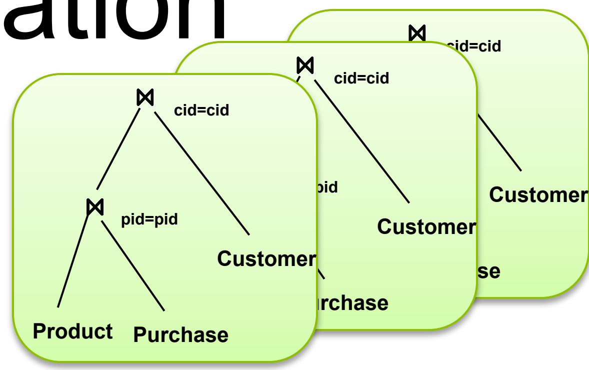
Approaches to Parallel Query Evaluation

- Inter-query parallelism
  - Transaction per node
  - OLTP

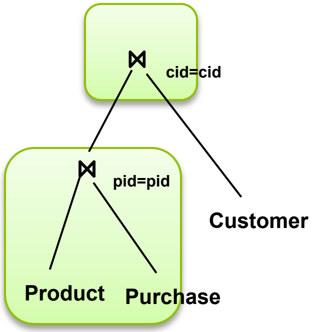


Approaches to Parallel Query Evaluation

- Inter-query parallelism
  - Transaction per node
  - OLTP

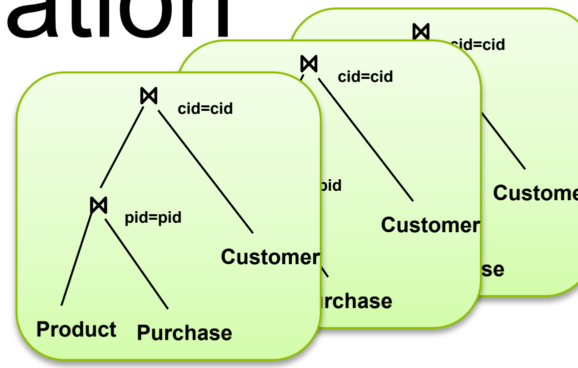


- Inter-operator parallelism
  - Operator per node
  - Both OLTP and Decision Support

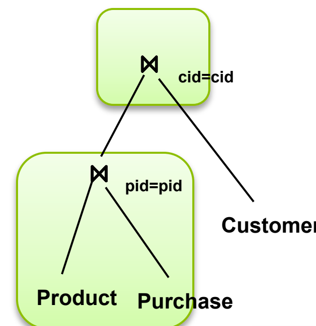


Approaches to Parallel Query Evaluation

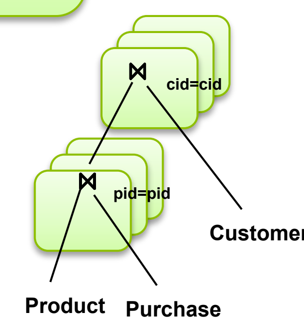
- Inter-query parallelism
  - Transaction per node
  - OLTP



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

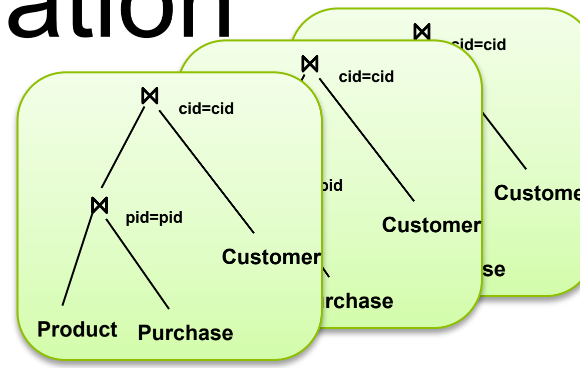


- Intra-operator parallelism
  - Operator on multiple nodes
  - Decision Support

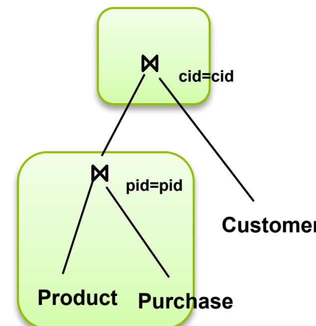


Approaches to Parallel Query Evaluation

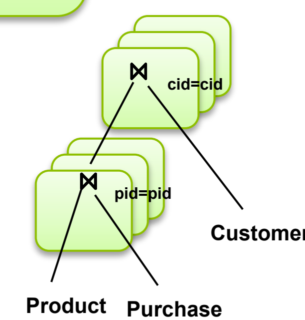
- Inter-query parallelism
  - Transaction per node
  - OLTP



- 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

Basic Query Processing: Quick Review in Class

Basic query processing on one node.

Given relations R(A,B) and S(B, C), no indexes, how do we compute:

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

Basic Query Processing: Quick Review in Class

Basic query processing on one node.

Given relations R(A,B) and S(B, C), no indexes, how do we compute:

- 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

Parallel Query Processing

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

- Selection: $\sigma_{A=123}(R)$ (that's easy, won't discuss…)
- Group-by: $\gamma_{A,sum(B)}(R)$
- Join: $R \bowtie S$

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

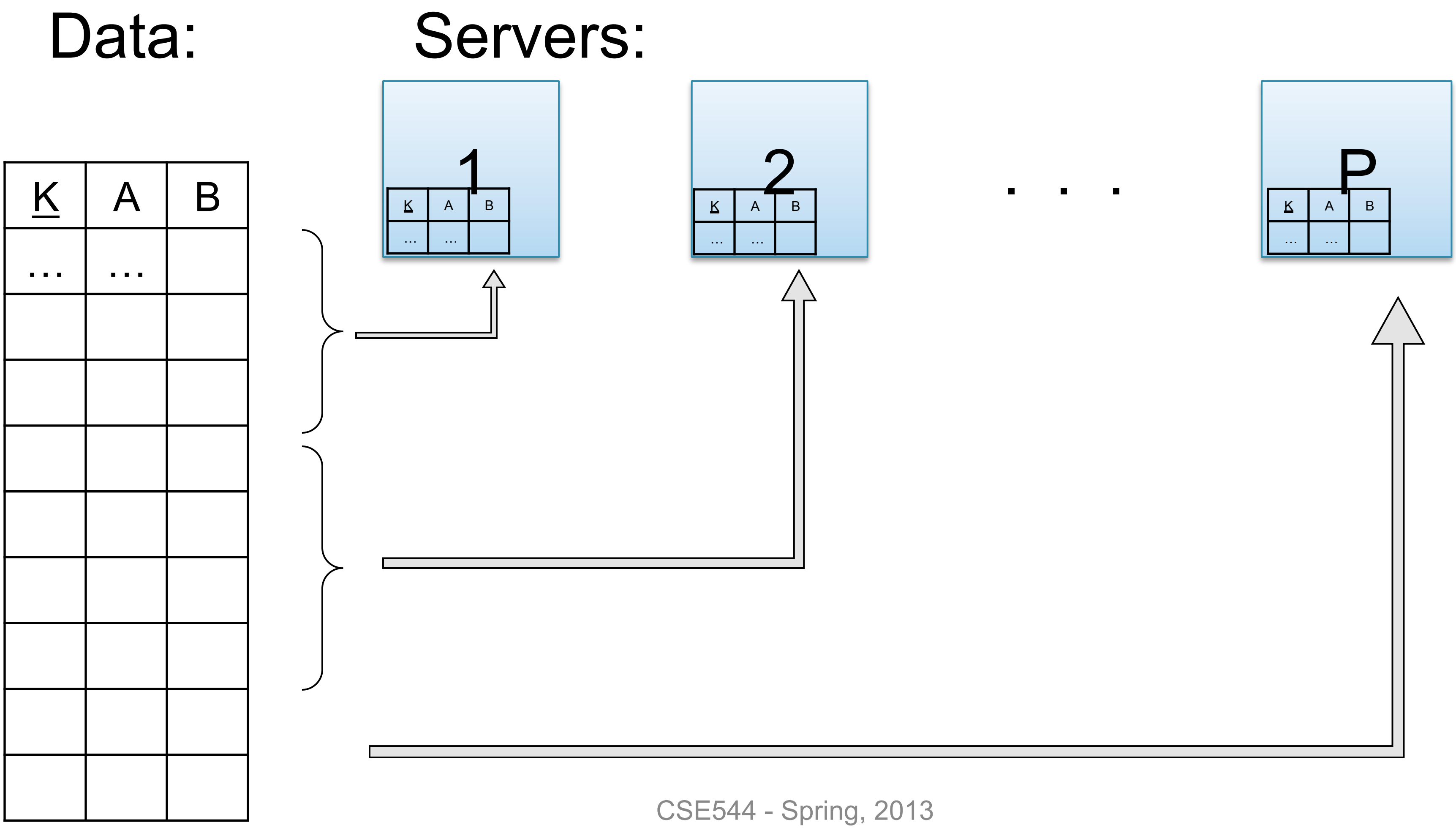
Horizontal Data Partitioning

Data:

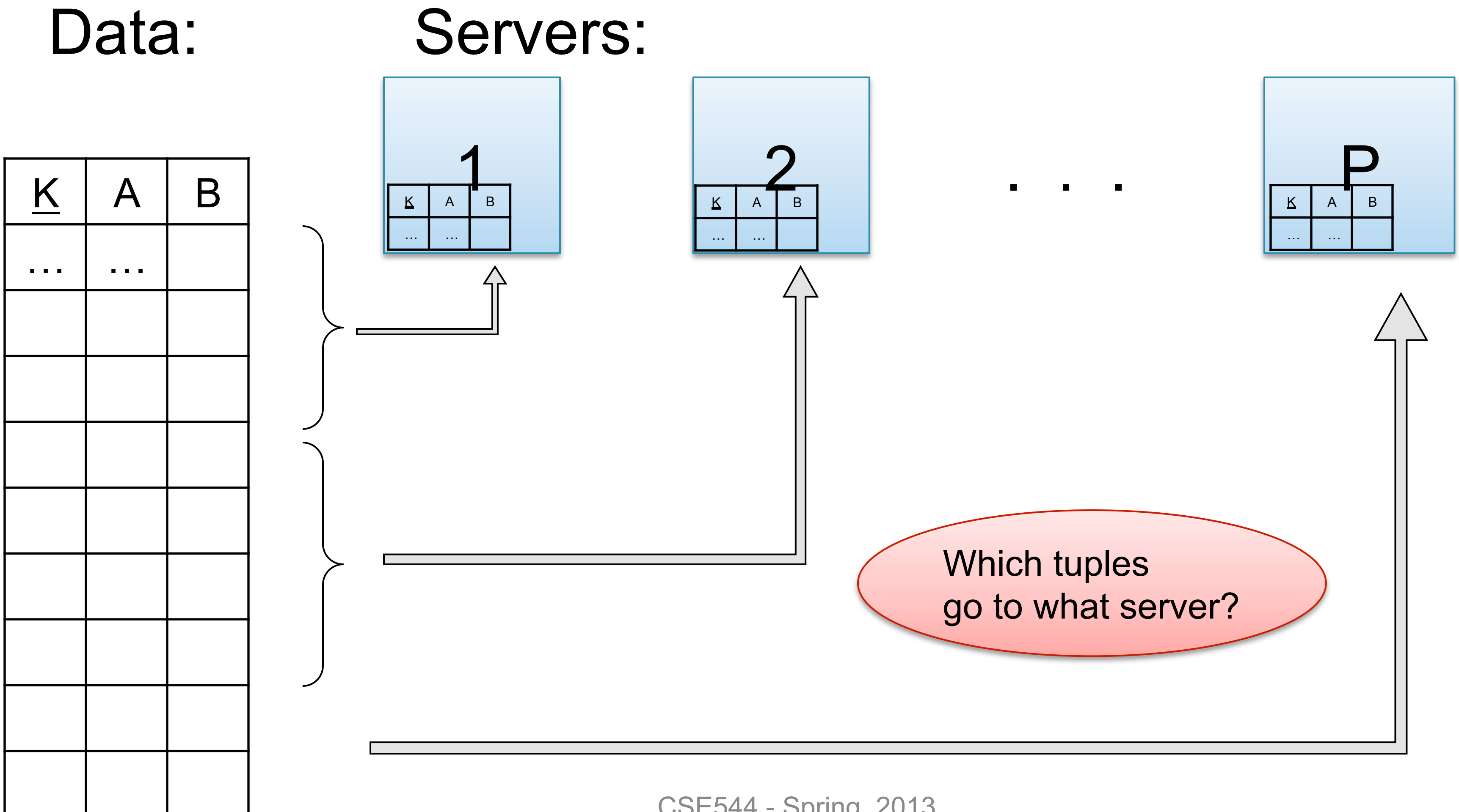
K	A	B
…	…	

Servers:

Horizontal Data Partitioning



Horizontal Data Partitioning



Horizontal Data Partitioning

- Block Partition:
  - Partition tuples arbitrarily s.t. $\text{size}(R_1) \approx \ldots \approx \text{size}(R_P)$
- Hash partitioned on attribute A:
  - Tuple t goes to chunk i, where $i = h(t.A) \bmod P + 1$
- 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:

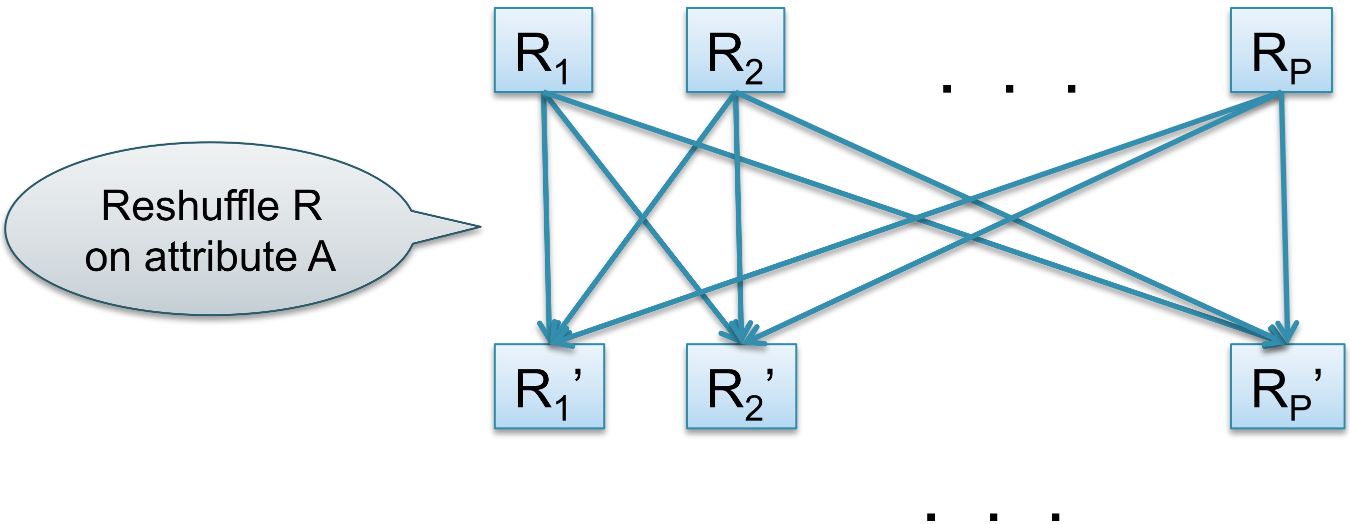
- R is hash-partitioned on A
- R is block-partitioned
- R is hash-partitioned on K

Parallel GroupBy

Data: $R(\underline{K},A,B,C)$

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

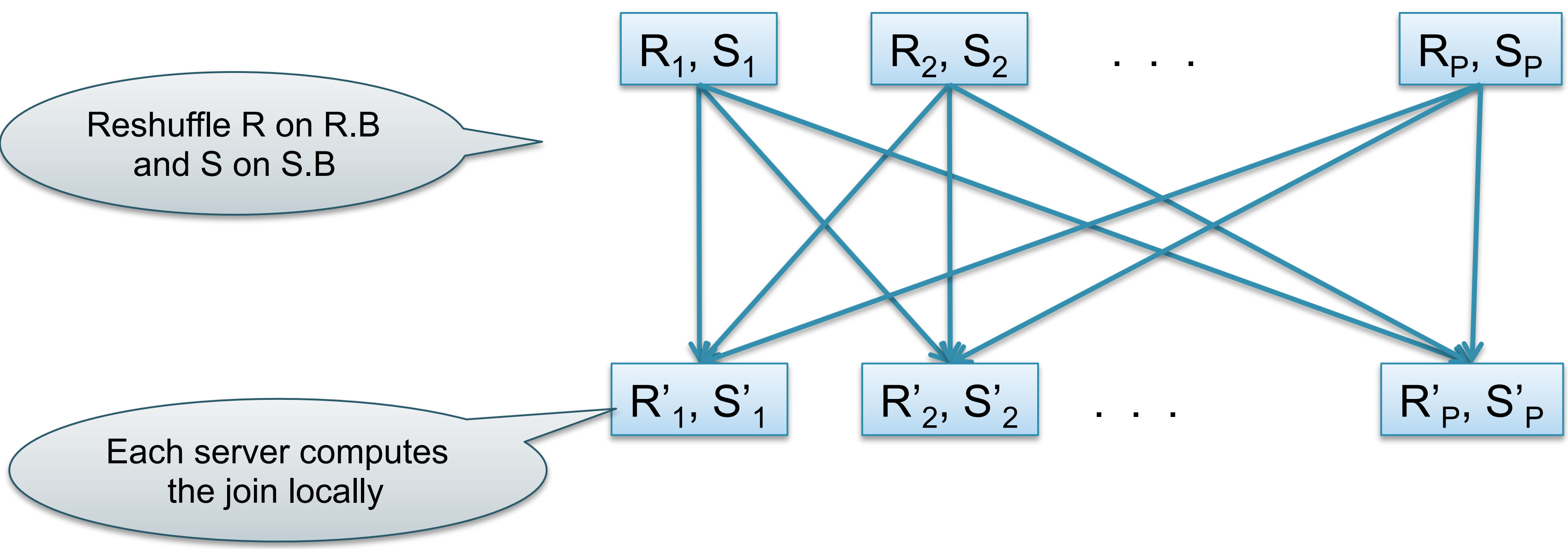
Initially, both R and S are horizontally partitioned on K1 and K2

R_1, S_1 $\quad$ R_2, S_2 $\quad$ R_P, S_P

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



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

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

- Hash-partition
  - On the key K
  - On the attribute A

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

  May be skewed

Assuming good hash function

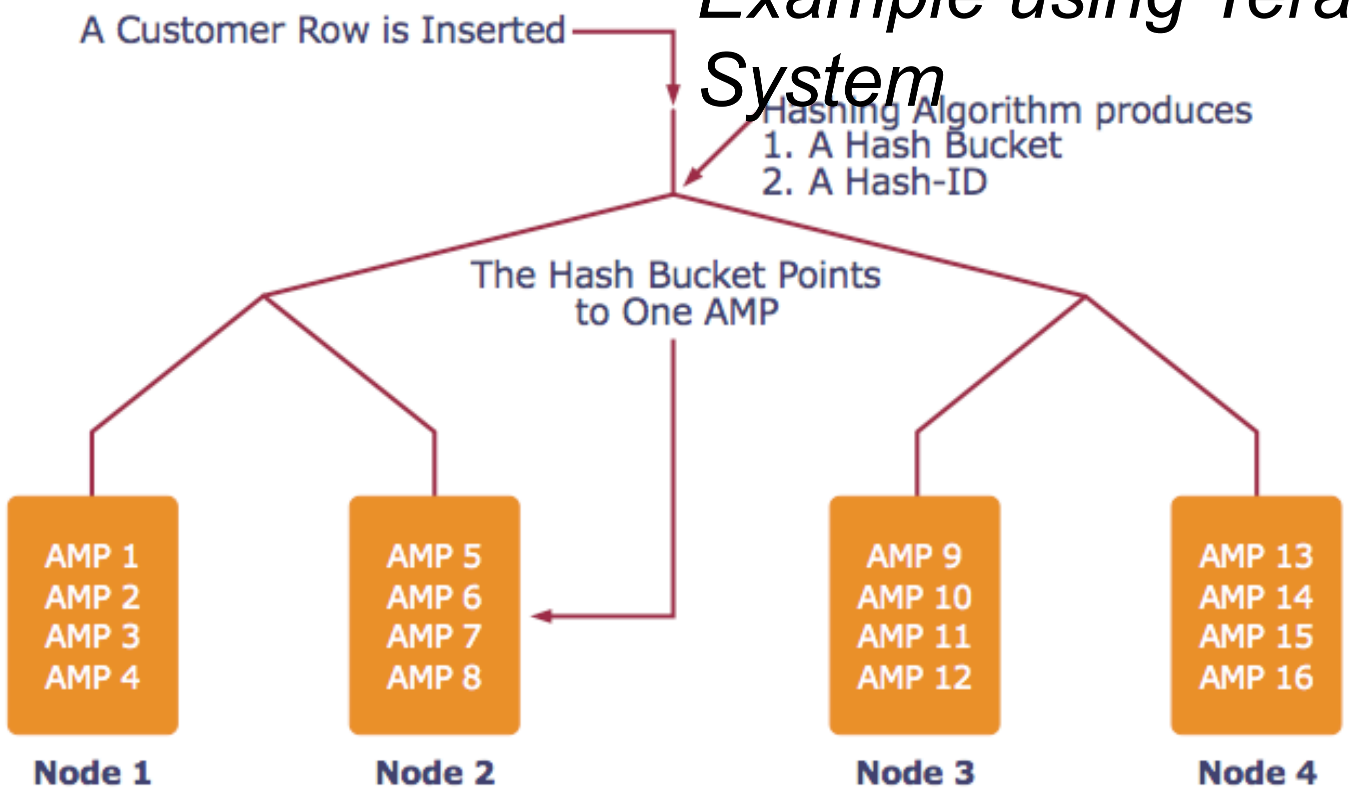
E.g. when all records have the same value of the attribute A, then all records end up in the same partition

Parallel DBMS

- 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
- Notice that user does not need to know how his/her SQL query was processed

Loading Data into a Parallel DBMS

Example using Teradata System

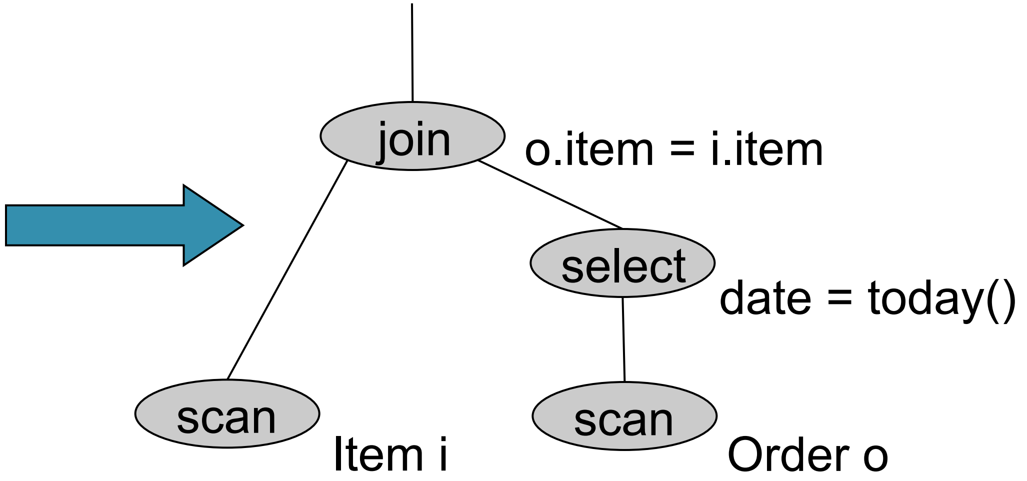


AMP = "Access Module Processor" = unit of parallelis

Example Parallel Query Execution

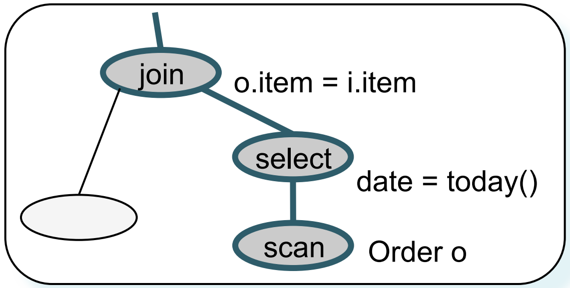
Find all orders from today, along with the items ordered

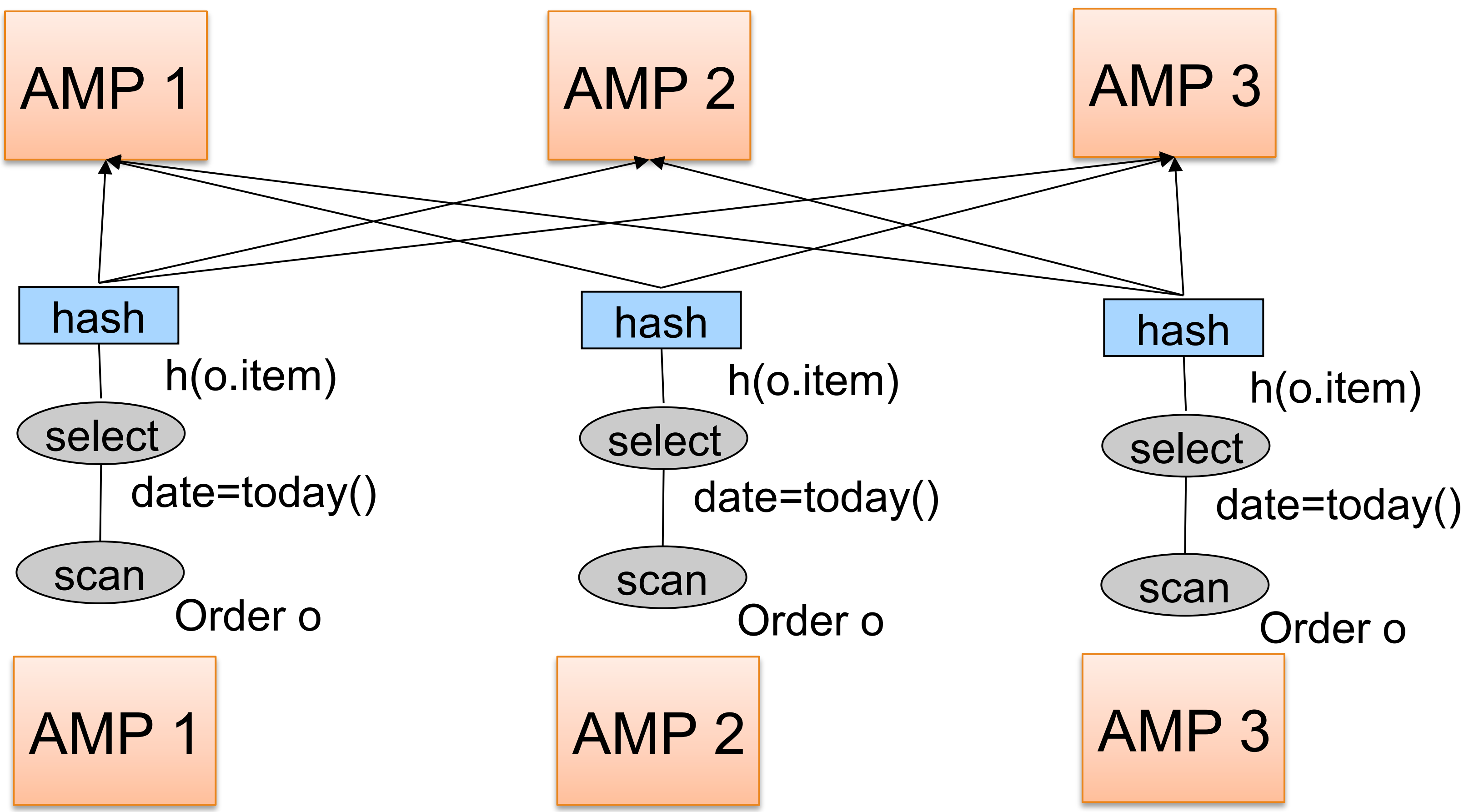
```
SELECT *
  FROM Order o, Line i
 WHERE o.item = i.item
   AND o.date = today()
```



Order(oid, item, date), Line(item, …)

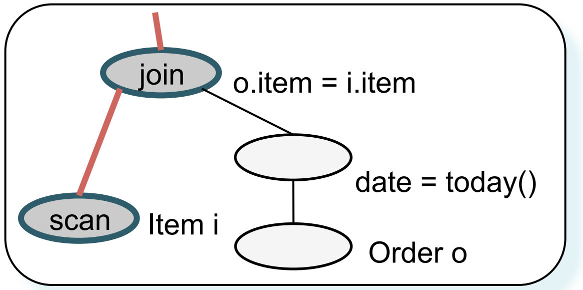
Example Parallel Query Execution

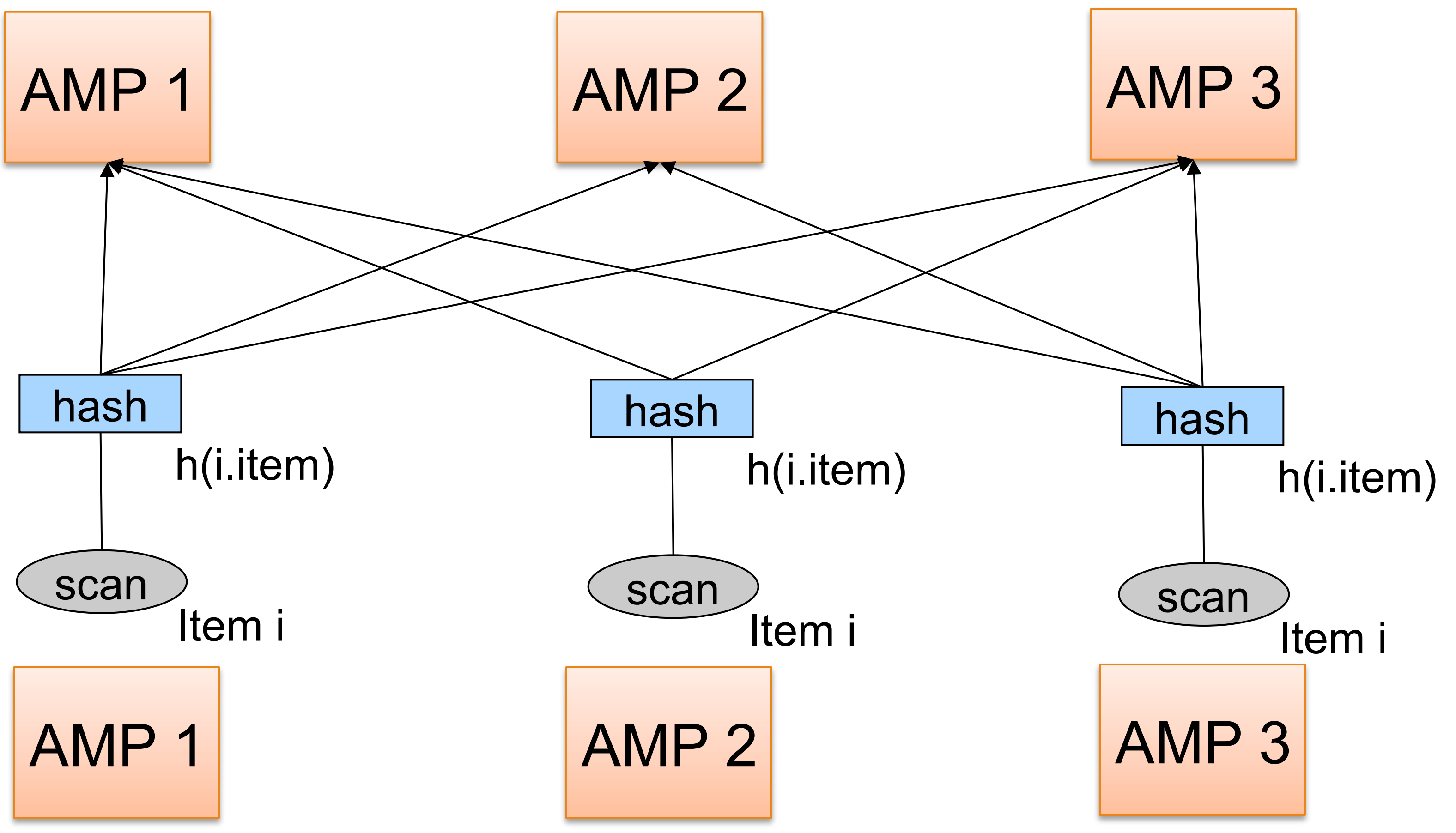




Order(oid, item, date), Line(item, …)

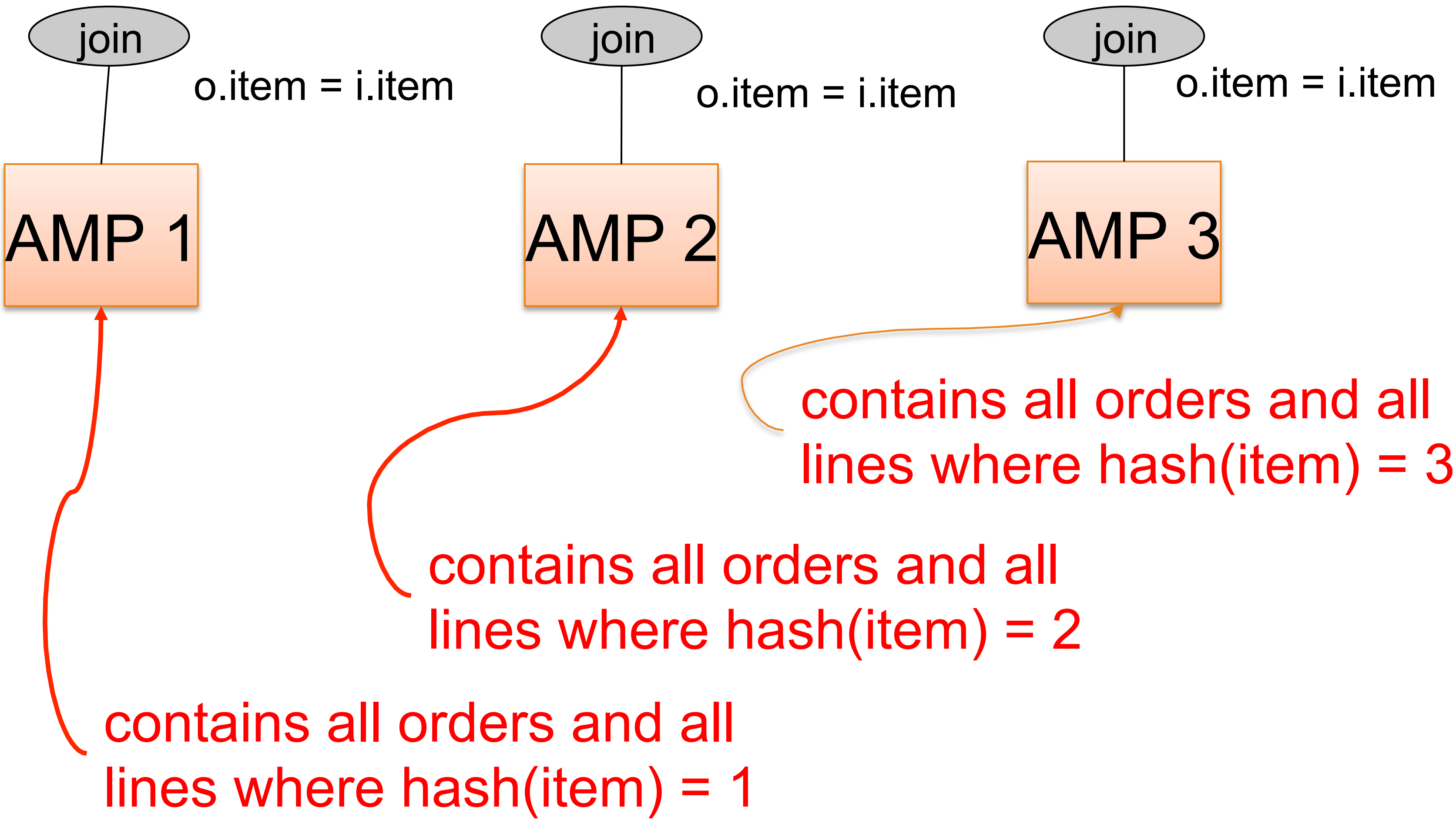
Example Parallel Query Execution





Order(<u>oid</u>, item, date), Line(item, …)

Example Parallel Query Execution



Parallel Query Plans

- Same relational operators
- Add special split and merge operators
  - Handle data routing, buffering, and flow control
- Example: exchange operator
  - Inserted between consecutive operators in the query plan

Time Permitting….

- Discussion of Shapiro's paper on join algorithms

Partitioned Hash Join, or GRACE Join

$R \bowtie S$

How does it work?

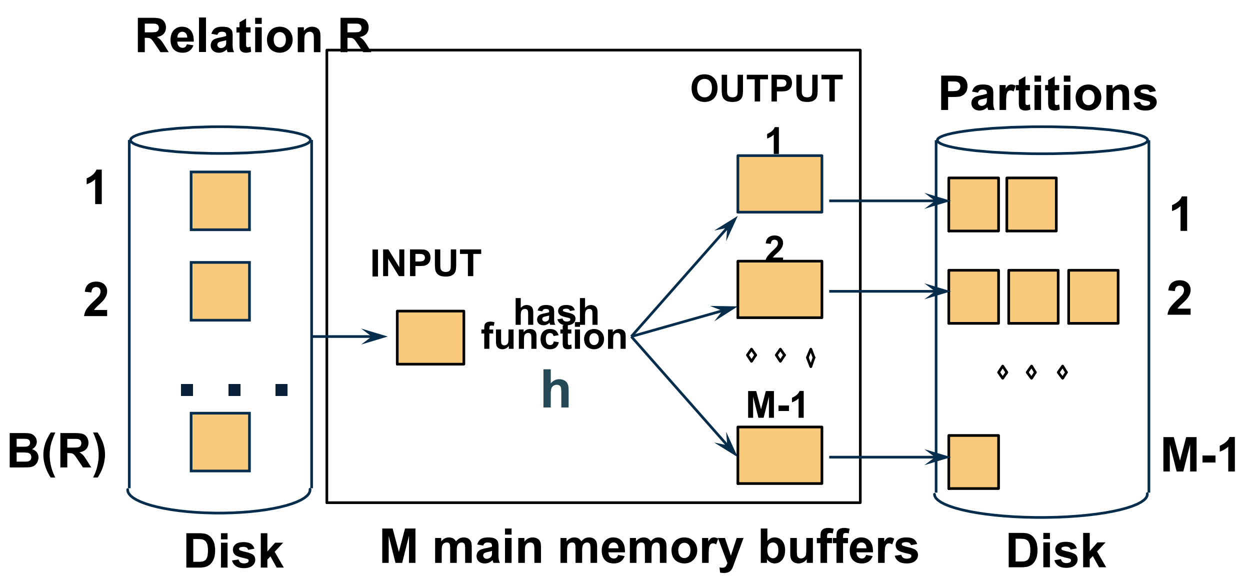
Partitioned Hash Join, or GRACE Join

$R \bowtie S$

- Step 1:
  - Hash S into M buckets
  - send all buckets to disk
- Step 2
  - Hash R into M buckets
  - Send all buckets to disk
- Step 3
  - Join every pair of buckets

The Idea of Hash-Based Partitioning

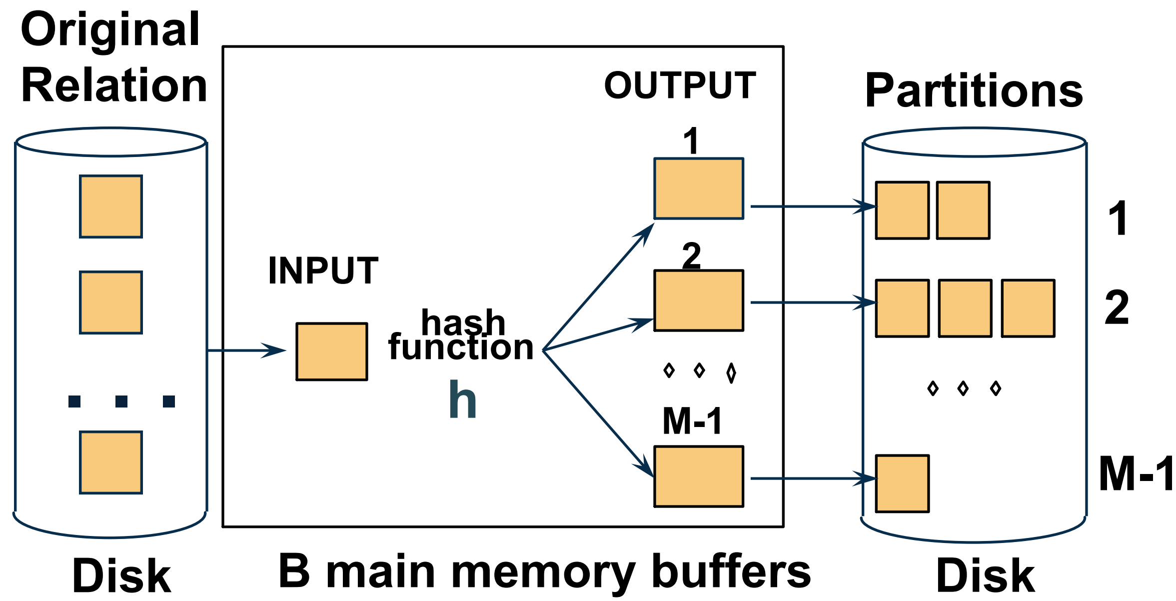
- Idea: partition a relation R into M-1 buckets, on disk
- Each bucket has size approx. B(R)/(M-1) ≈ B(R)/M



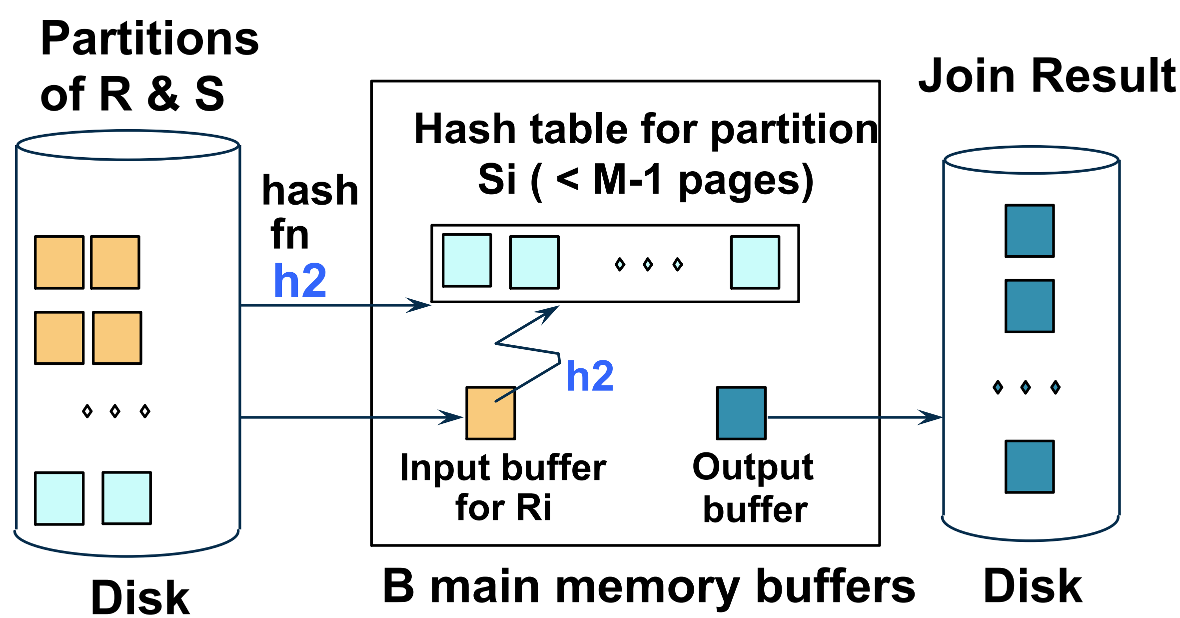
Assumption: $B(R)/M \le M$, i.e. $B(R) \le M^2$

Grace-Join

- Partition both relations using hash fn **h**: R tuples in partition i will only join S tuples in partition i.



- Read in a partition of R, hash it using **h2 (<> h!)**. Scan matching partition of S, search for matches.



Grace Join

- Cost: 3B(R) + 3B(S)
- Assumption: min(B(R), B(S)) <= M^2

Hybrid Hash Join

- What problem does it address?

Hybrid Hash Join

- What problem does it address?
- If $B(R) \leq M$ then we can use main memory hash-join: cost = $B(R) + B(S)$
- If $B(R) >\approx M$ then we must use Grace join: cost jumps to $3*B(R) + 3*B(S)$

Hybrid Hash Join

- How does it work?

Hybrid Hash Join

- How does it work?
- Use B(R)/M buckets
- Since B(R)/M << M, there is enough space left in main memory: use it to store a few buckets
- Fuzzy math to make this work, but best done adaptively:
  - Start by keeping ***all*** buckets in main memory
  - When the remaining memory (M - B(R)/M) fills up, spill one bucket to disk