CSE544
Data Management

Lectures 13

Parallel Query Processing

Annoucements

- HW4 due on Friday
- Project Milestone due next Friday
- Mini-HW5 will be posted on Saturday

Distributed/Parallel Query Processing

Parallel DBs since the 80s

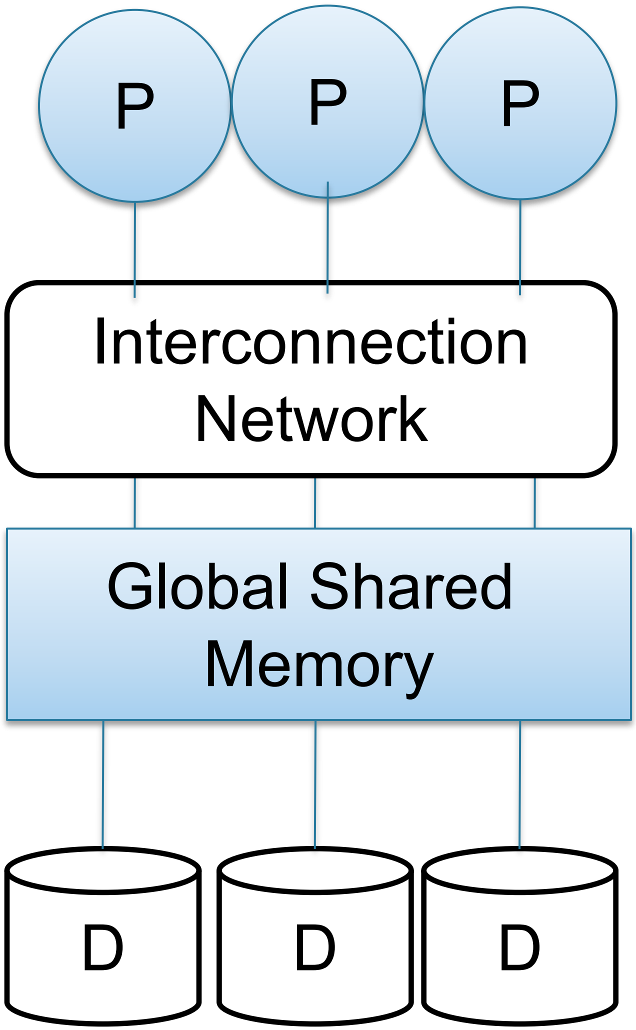
New, strong technology pulls:

- Multi-core
- Cloud computing

Architectures for Parallel Databases

- Shared memory
- Shared disk
- Shared nothing

Shared Memory

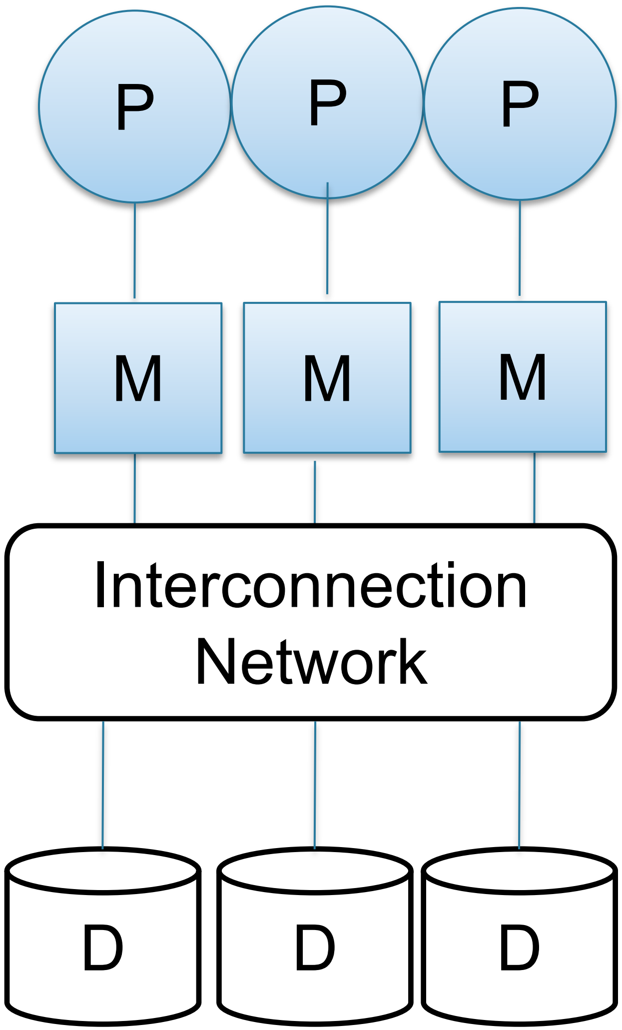


- SMP = symmetric multiprocessor
- Nodes share RAM and disk
- 10x … 100x processors

- Example: SQL Server runs on a single machine and can leverage many threads to speed up a query

- Easy to use and program
- Expensive to scale

Shared Disk



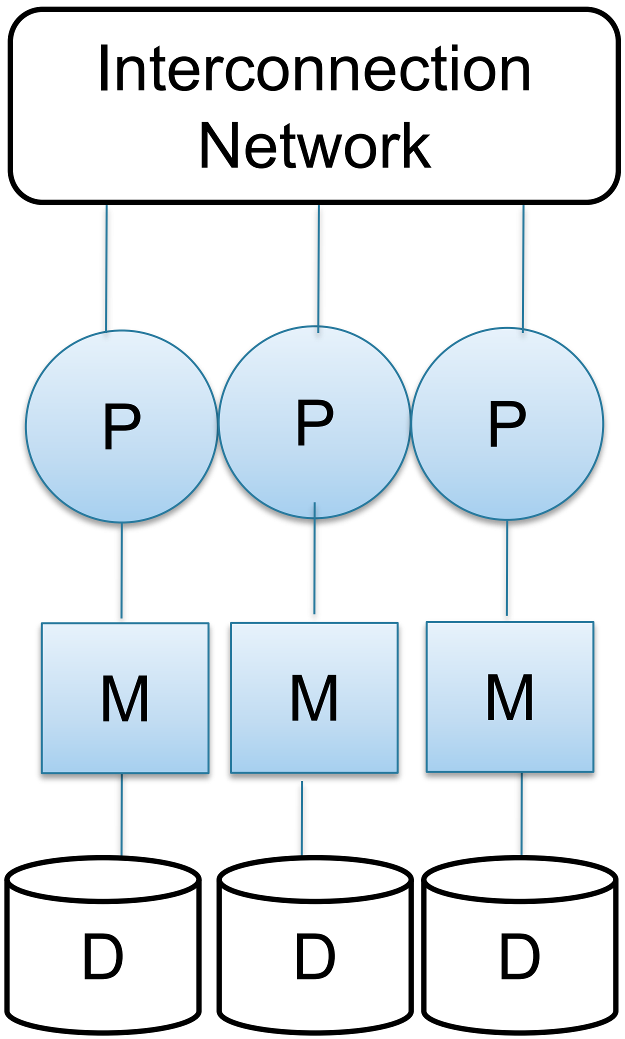
- All nodes access same disks
- 10x processors

- Example: Oracle

- No more memory contention

- Harder to program
- Still hard to scale

Shared Nothing



- Cluster of commodity machines
- Called "clusters" or "blade servers”
- Each machine: own memory&disk
- Up to x1000-x10000 nodes
- Example: redshift, spark, snowflake

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.

Performance Metrics

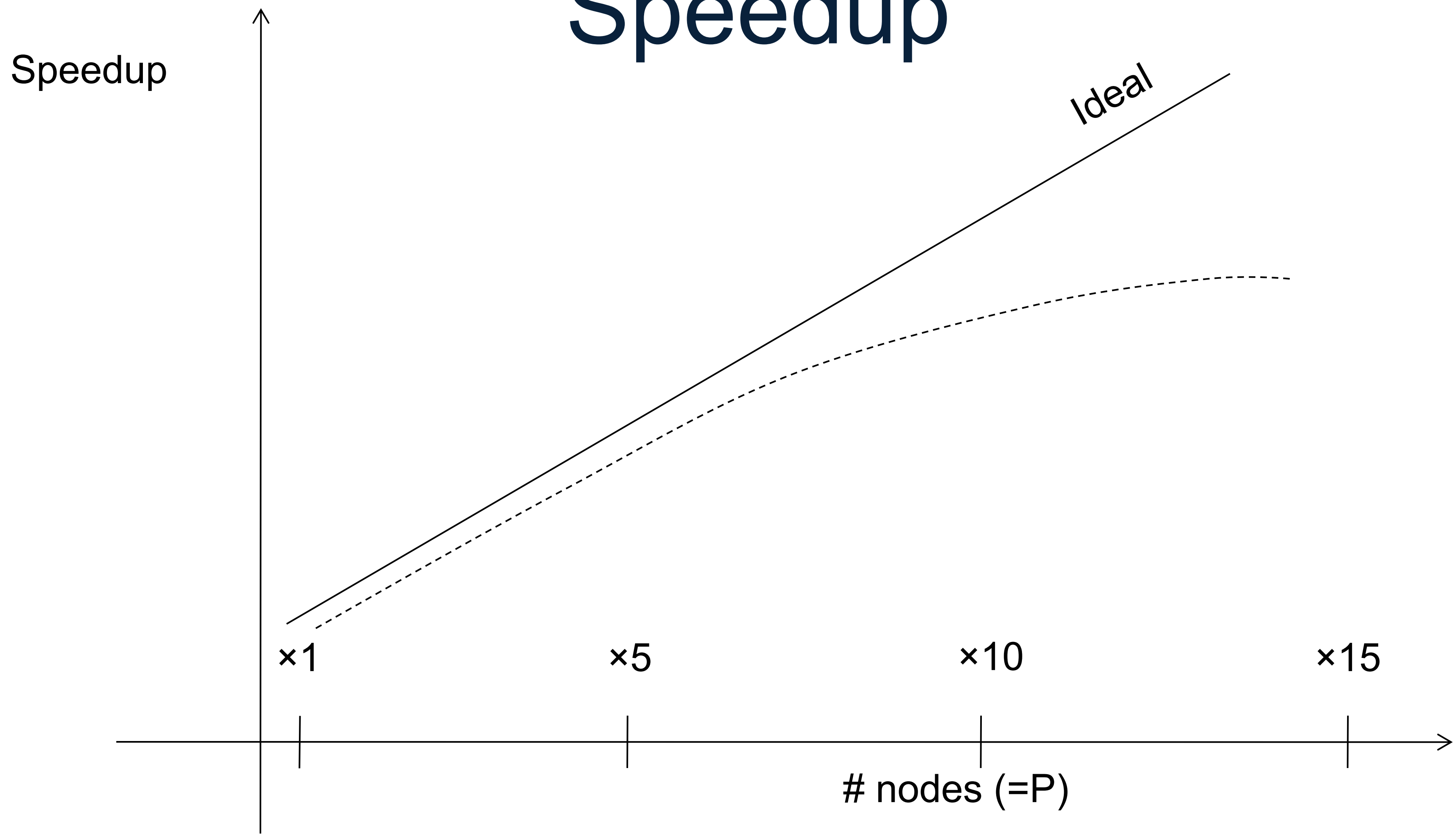
Nodes = processors = computers

- Speedup:
  - More nodes, same data → higher speed

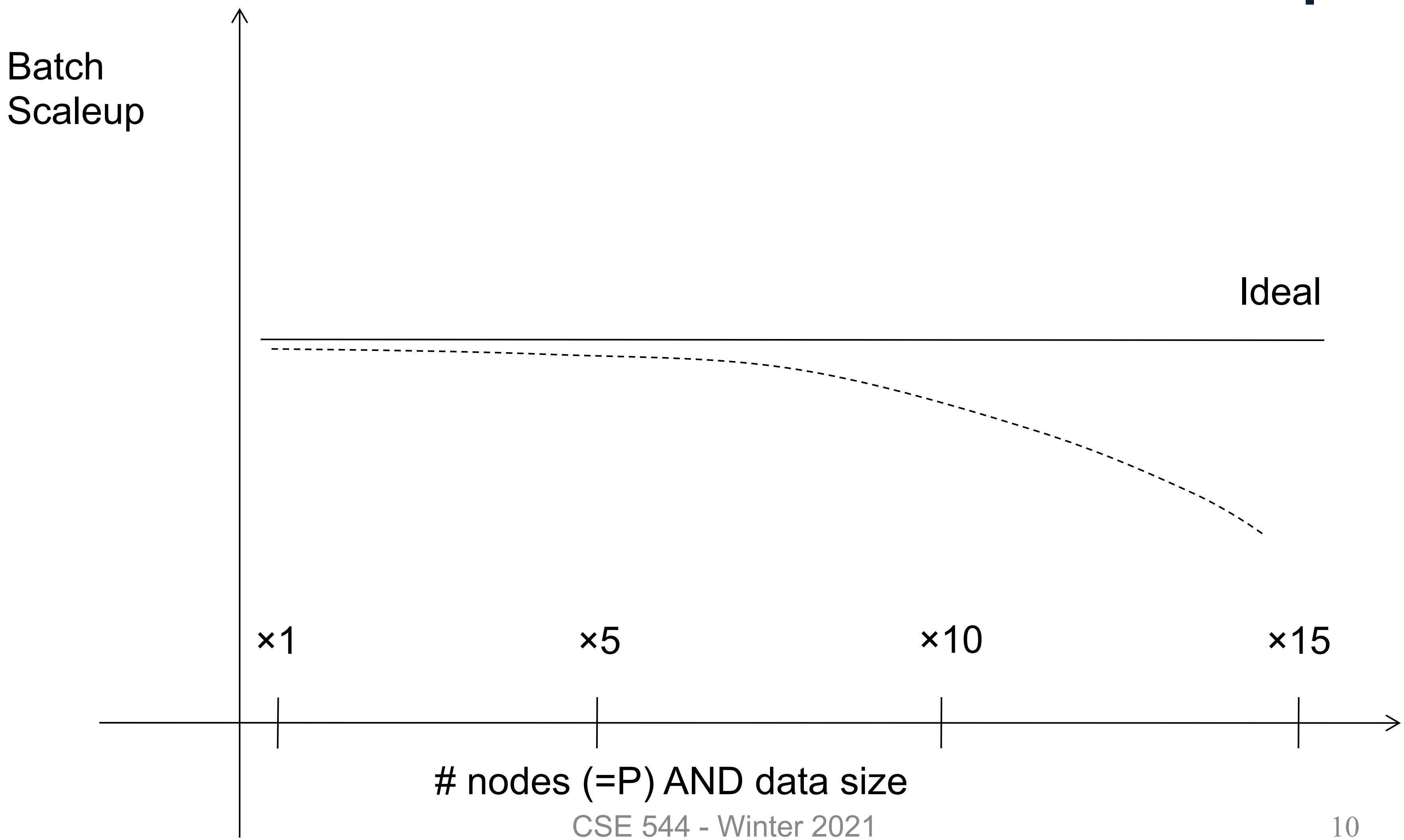
- Scaleup:
  - More nodes, more data → same speed

Warning: sometimes *Scaleup* is used to mean *Speedup*

Linear v.s. Non-linear Speedup



Linear v.s. Non-linear Scaleup



Why Sub-linear?

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

Distributed Query Processing Algorithms

Horizontal Data Partitioning

- Block Partition, a.k.a. Round Robin:
  - 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
- 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 $v_{i-1} < t.A < v_i$

Notation

When a relation R is distributed to p servers, we draw the picture like this:

Here R_1 is the fragment of R stored on server 1, etc

$$R = R_1 \cup R_2 \cup \cdots \cup R_P$$

Uniform Load and Skew

- $|R| = N$ tuples, then $|R_1| + |R_2| + \ldots + |R_p| = N$
- We say the load is uniform when:
  $$|R_1| \approx |R_2| \approx \ldots \approx |R_p| \approx N/p$$
- Skew means that some load is much larger:
  $$\max_i |R_i| >> N/p$$

We design algorithms for uniform load, discuss skew later

Parallel Algorithm

- Selection σ
- Join ⋈
- Group by γ

Parallel Selection

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

Query: $\sigma_{A=v}(R)$, or $\sigma_{v1<A<v2}(R)$

- Block partitioned:
  - All servers must scan and filter the data
- Hash partitioned:
  - Can have all servers scan and filter the data
  - Or can optimize and only have some servers do work
- Range partitioned
  - Also only some servers need to do the work

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 or hash-partitioned on K

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:

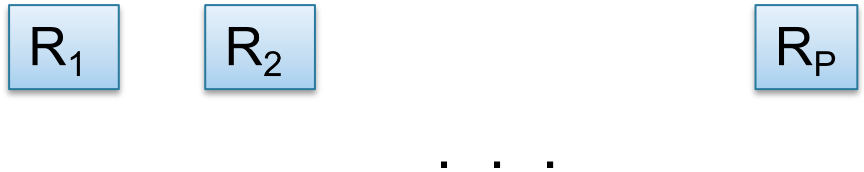
- R is hash-partitioned on A
  - Each server i computes locally $\gamma_{A,sum(C)}(R_i)$
- R is block-partitioned or hash-partitioned on K
  - Need to reshuffle data on A first (next slide)
  - Then compute locally $\gamma_{A,sum(C)}(R_i)$

Basic Parallel GroupBy

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

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

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

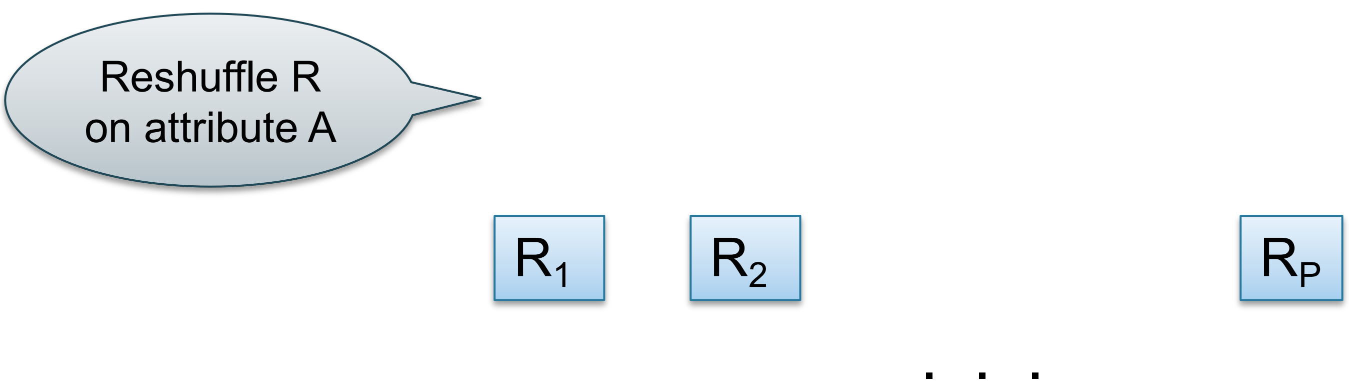


Basic Parallel GroupBy

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

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

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

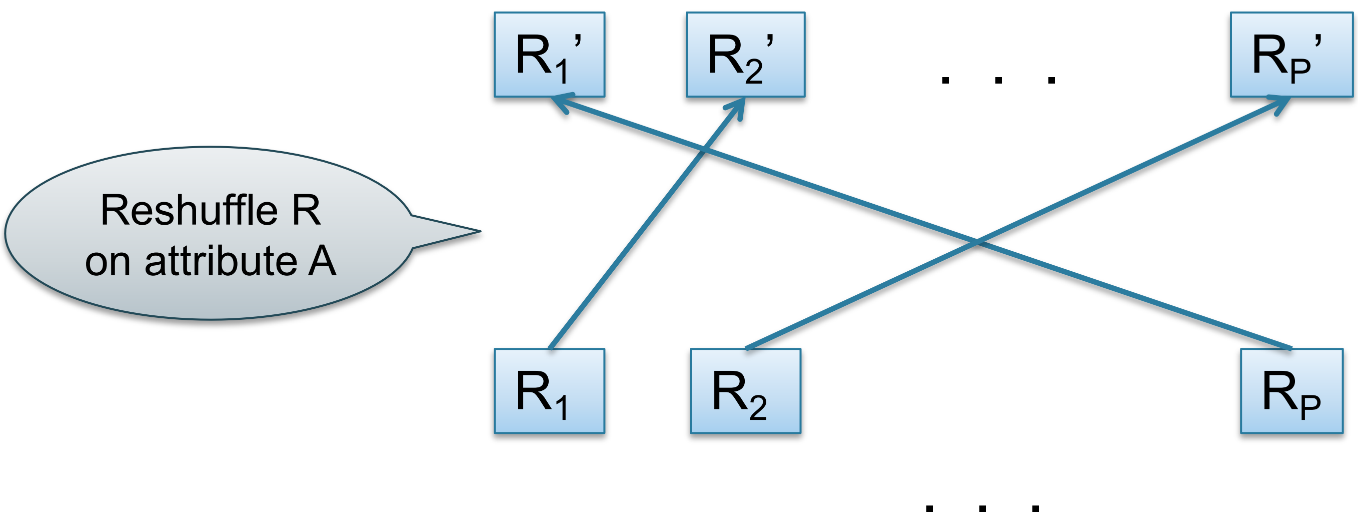


Basic Parallel GroupBy

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

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

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

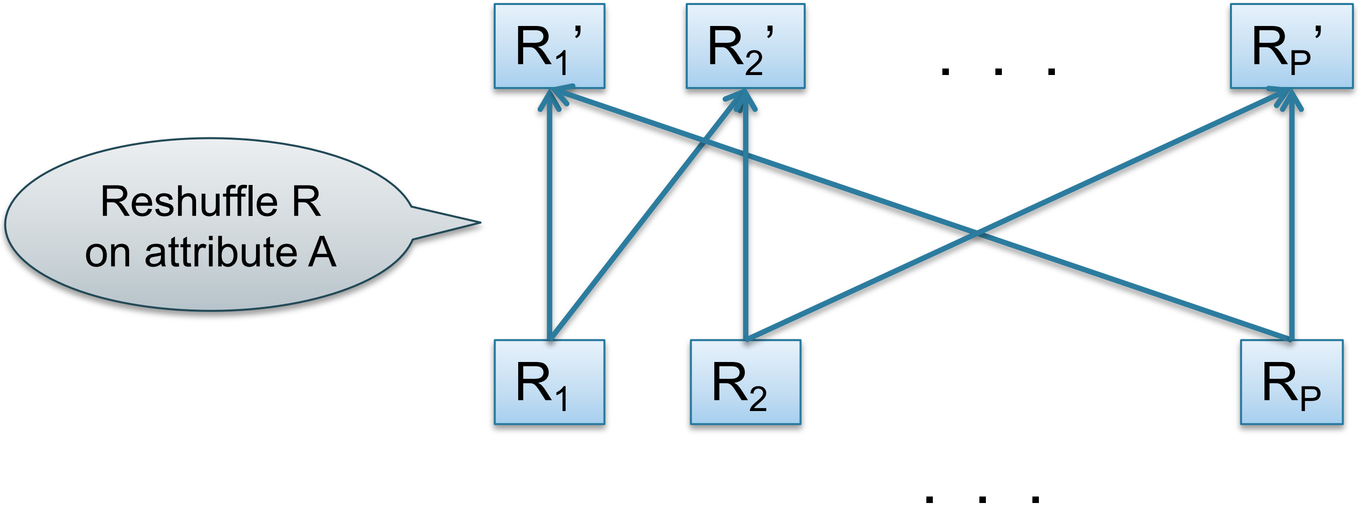


Basic Parallel GroupBy

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

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

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

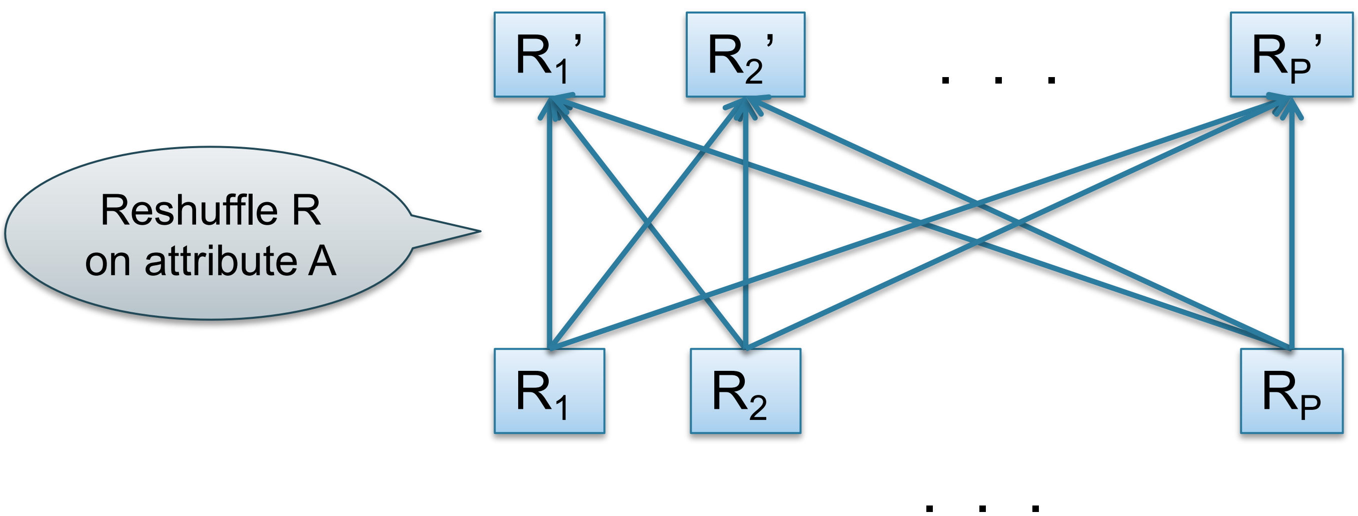


Basic Parallel GroupBy

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

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

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

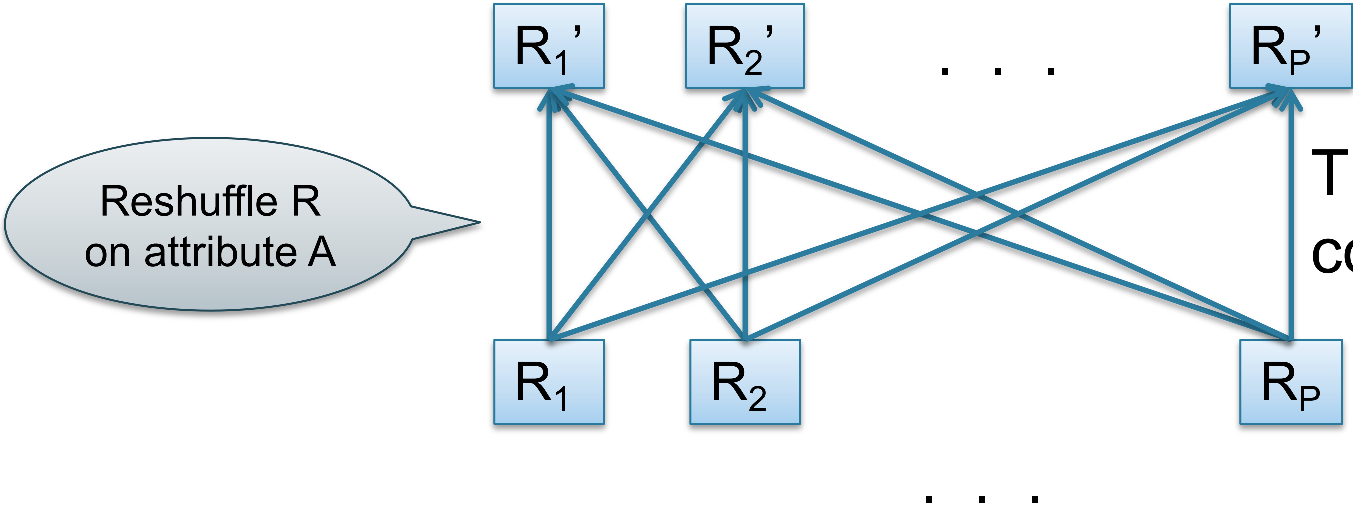


Basic Parallel GroupBy

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

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

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



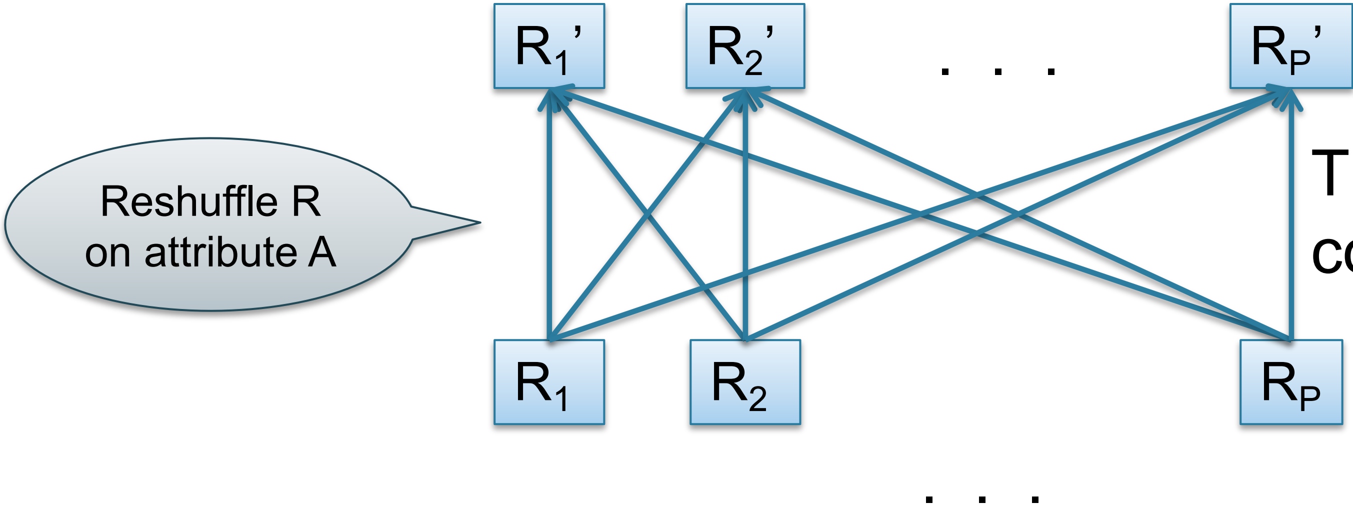
This is done in *<u>one</u>* communication step

Basic Parallel GroupBy

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

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

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



This is done in *one* communication step

Can you think of an optimization?

Basic Parallel GroupBy

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

Basic Parallel GroupBy

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

Step 0: [Optimization] each server i computes local group-by:
$$T_i = \gamma_{A,sum(C)}(R_i)$$

Basic Parallel GroupBy

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

Step 0: [Optimization] each server i computes local group-by:
$T_i = \gamma_{A,sum(C)}(R_i)$

Step 1: partitions tuples in T_i using hash function h(A):
$T_{i,1}, T_{i,2}, \ldots, T_{i,p}$
then send fragment $T_{i,j}$ to server j

Basic Parallel GroupBy

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

Step 0: [Optimization] each server i computes local group-by:
$T_i = \gamma_{A,sum(C)}(R_i)$

Step 1: partitions tuples in T_i using hash function h(A):
$T_{i,1}, T_{i,2}, \ldots, T_{i,p}$
then send fragment $T_{i,j}$ to server j

Step 2: receive fragments, union them, then group-by
$R_j' = T_{1,j} \cup \ldots \cup T_{p,j}$
$Answer_j = \gamma_{A,\ sum(C)}(R_j')$

Example Query with Group By

```
SELECT a, sum(b) as sb
FROM R WHERE c > 0
GROUP BY a
```

Example Query with Group By

```
SELECT a, sum(b) as sb
FROM R WHERE c > 0
GROUP BY a
```

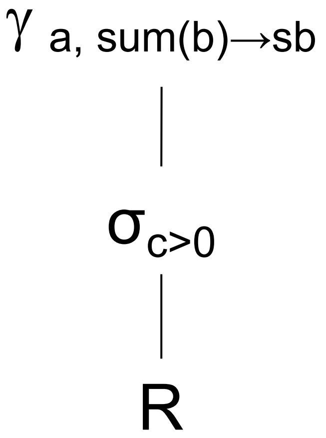
$$\gamma_{a,\ sum(b) \rightarrow sb}$$

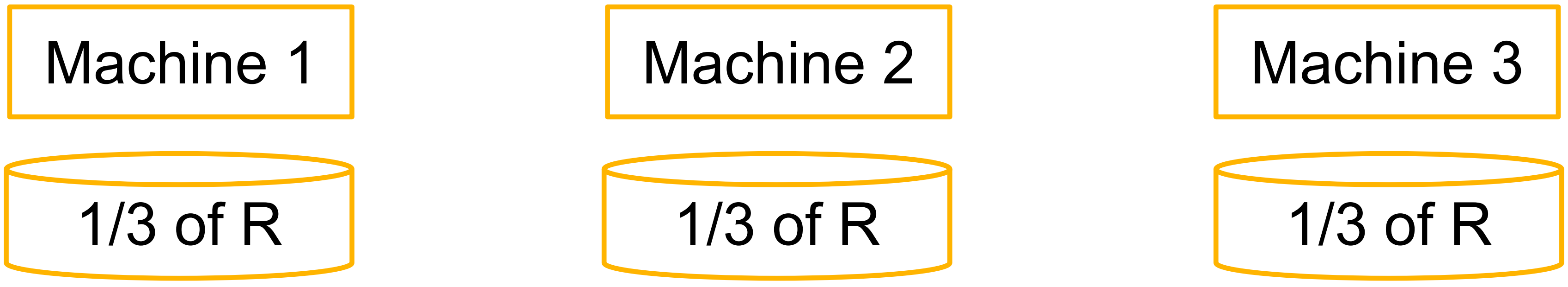
$$\sigma_{c>0}$$

$$R$$

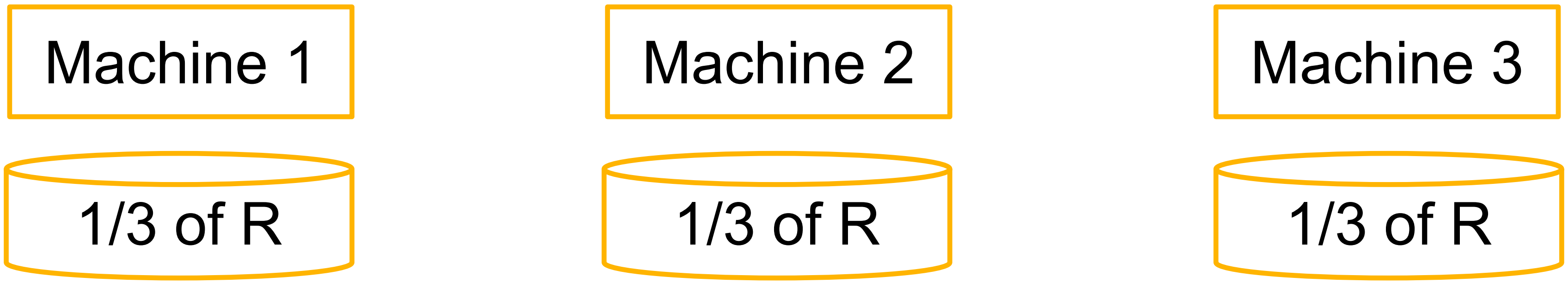
Example Query with Group By

```
SELECT a, sum(b) as sb
FROM R WHERE c > 0
GROUP BY a
```

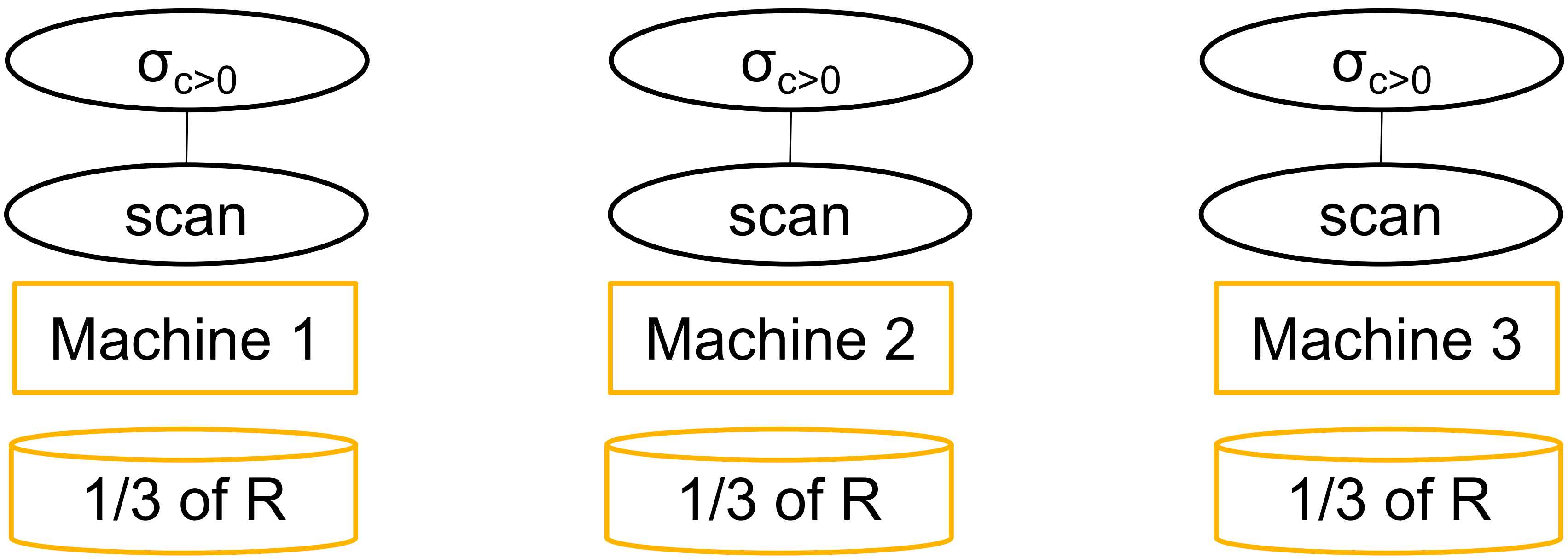




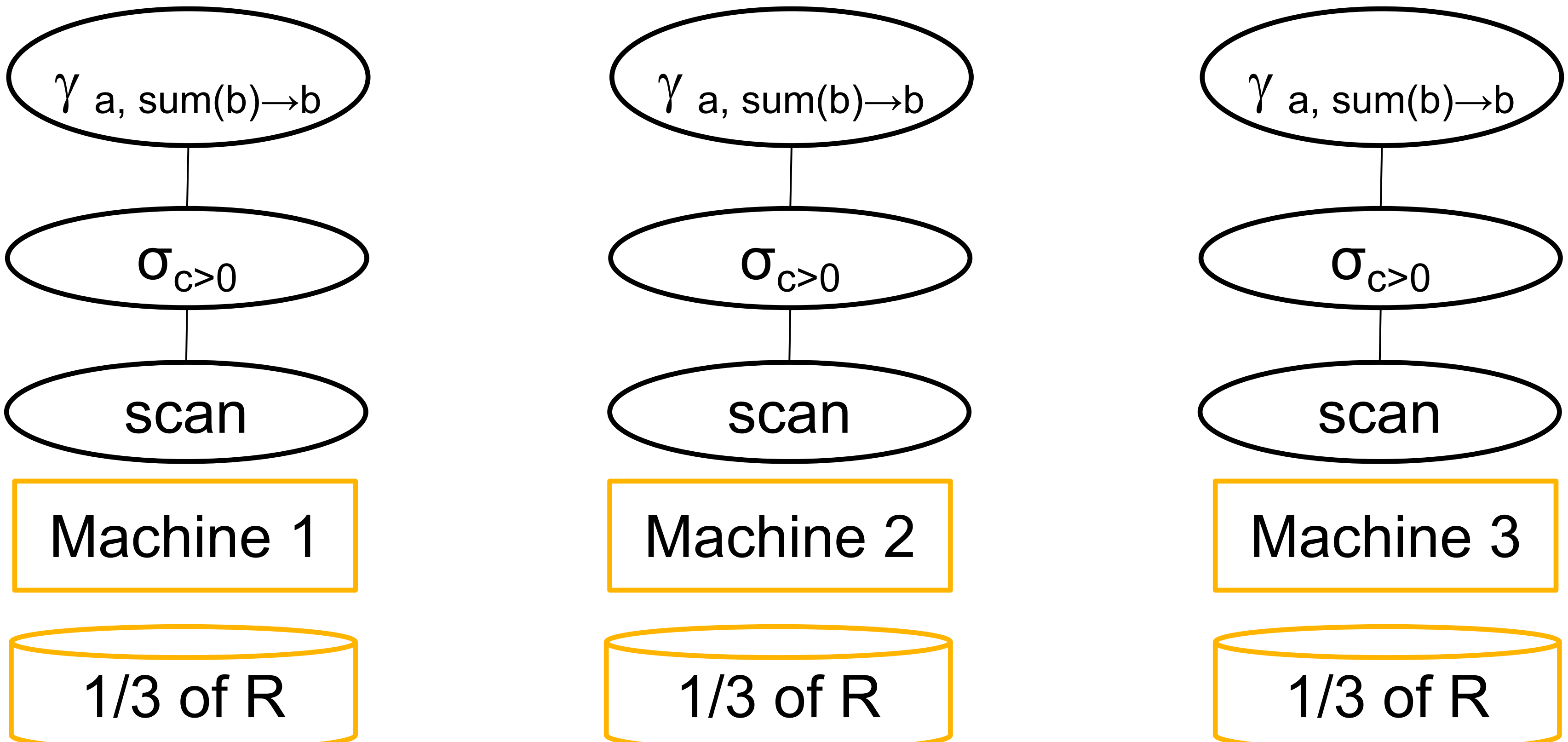
SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a



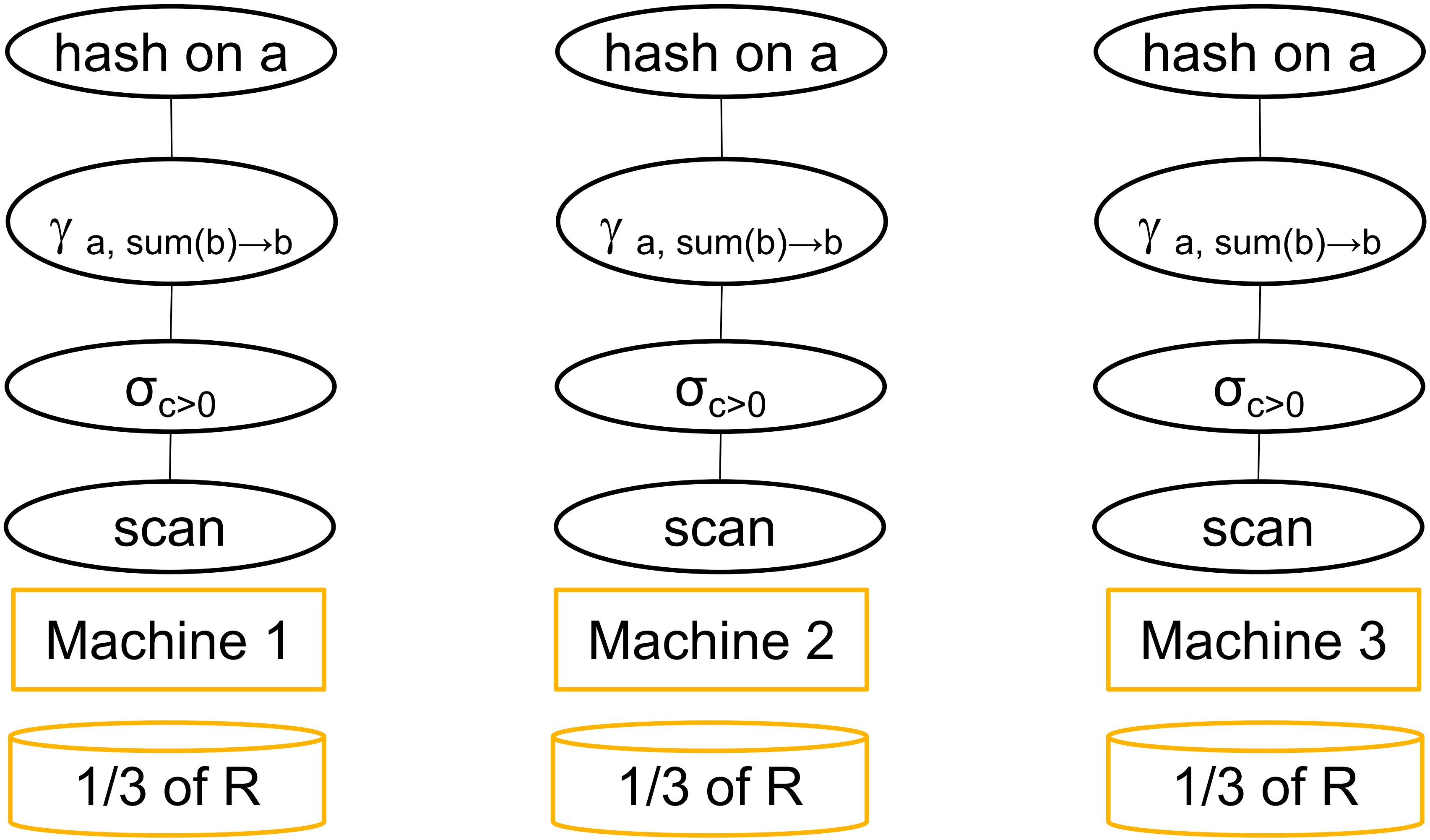
SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a



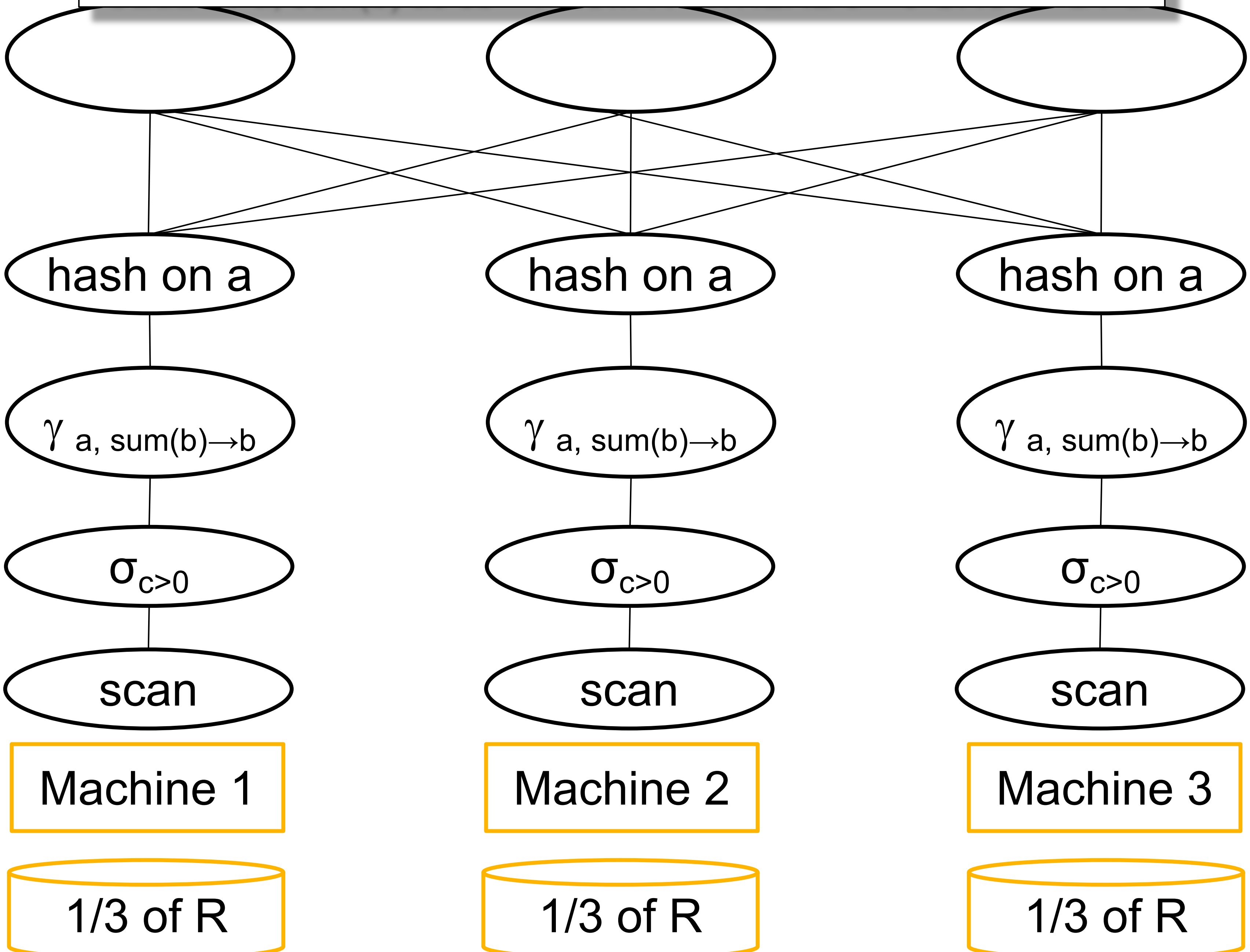
SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a

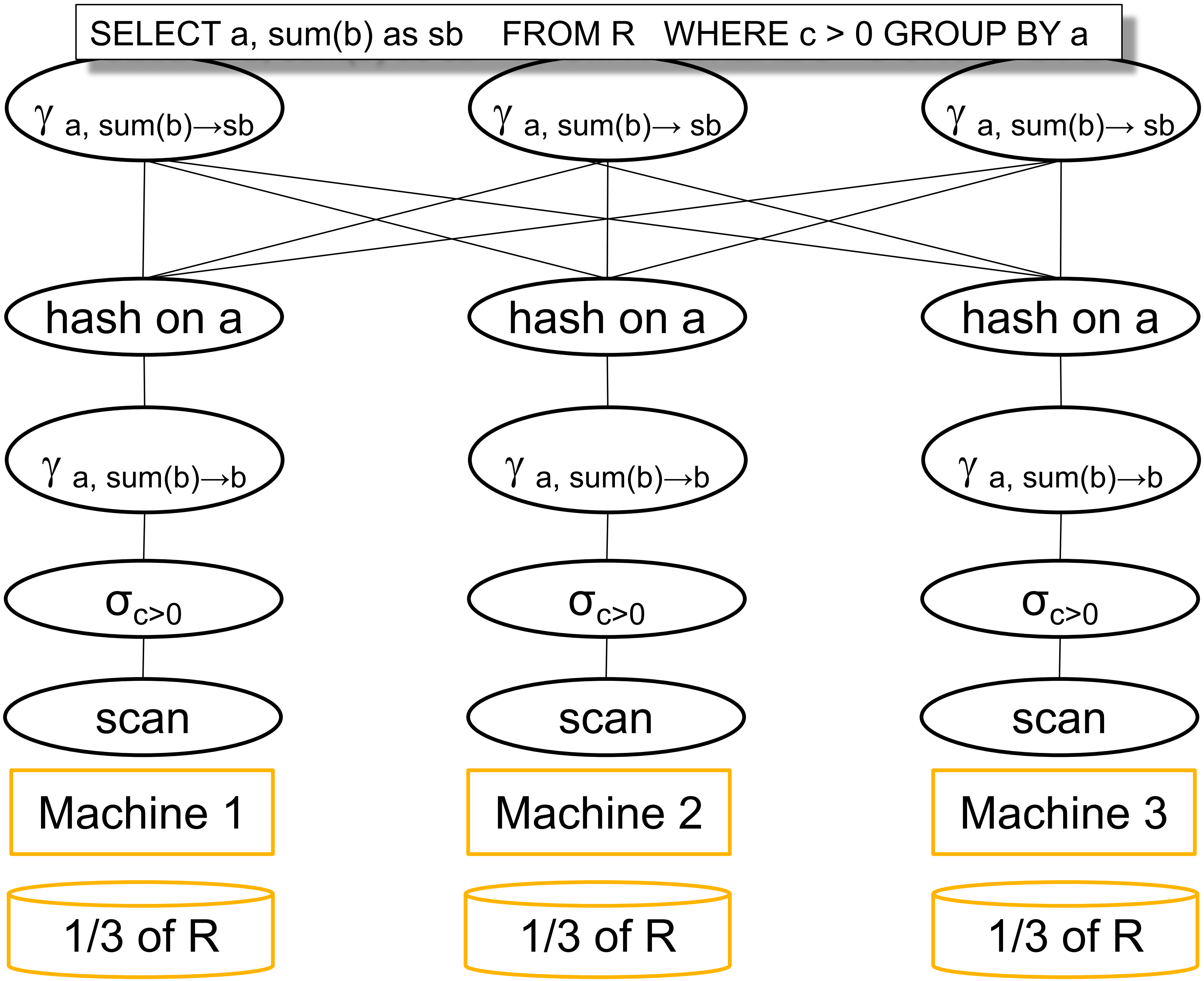


SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a



SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a




SELECT a, sum(b) as sb FROM R WHERE c > 0 GROUP BY a
γ a, sum(b)→sb
γ a, sum(b)→ sb
γ a, sum(b)→ sb
hash on a
hash on a
hash on a
γ a, sum(b)→b
γ a, sum(b)→b
γ a, sum(b)→b
σc>0
σc>0
σc>0
scan
scan
scan
Machine 1
Machine 2
Machine 3
1/3 of R
1/3 of R
1/3 of R

Pushing Aggregates Past Union

The rule that allowed us to do early summation is:

$$\gamma_{A,sum(B)\to C}(R_1 \cup R_2) =$$

$$= \gamma_{A,sum(D)\to C}(\ \gamma_{A,sum(B)\to D}(R_1)\ \cup\ \gamma_{A,sum(B)\to D}(R_2)\ )$$

For example:

- R_1 has B= x,y,z; R_2 has B=u,w
- Then: x+y+z+u+w = (x+y+z) + (u+w)

Pushing Aggregates Past Union

Which other rules can we push past union?

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

Pushing Aggregates Past Union

Which other rules can we push past union?

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

Distributive	Algebraic	Holistic
$sum(a_1+a_2+\ldots+a_9)=$ $sum(sum(a_1+a_2+a_3)+$ $sum(a_4+a_5+a_6)+$ $sum(a_7+a_8+a_9))$	avg(B) = sum(B)/count(B)	median(B)

Speedup and Scaleup

Consider the query $\gamma_{A,sum(C)}(R)$
Assume the local runtime for group-by is linear O(|R|)

If we double number of nodes P, what is the runtime?

If we double both P and size of R, what is the runtime?

Speedup and Scaleup

Consider the query $\gamma_{A,sum(C)}(R)$
Assume the local runtime for group-by is linear $O(|R|)$

If we double number of nodes P, what is the runtime?

- Half (chunk sizes become ½)

If we double both P and size of R, what is the runtime?

- Same (chunk sizes remain the same)

Speedup and Scaleup

Consider the query $\gamma_{A,sum(C)}(R)$
Assume the local runtime for group-by is linear O(|R|)

If we double number of nodes P, what is the runtime?

- Half (chunk sizes become ½)

If we double both P and size of R, what is the runtime?

- Same (chunk sizes remain the same)

But only if the data is without skew!

Parallel/Distributed Join

Three “algorithms”:

- Hash-partitioned
- Broadcast
- Combined: “skew-join” or other names

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

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

Query: $R \bowtie_{A=B} S$

R_1, S_1 | R_2, S_2 | . . . | R_P, S_P

Initially, R and S are block partitioned.
Notice: they may be stored in DFS (recall MapReduce)

Some servers hold R-chunks, some hold S-chunks, some hold both

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

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

Query: $R \bowtie_{A=B} S$

Reshuffle R on R.A
and S on S.B

R_1, S_1 R_2, S_2 . . . R_P, S_P

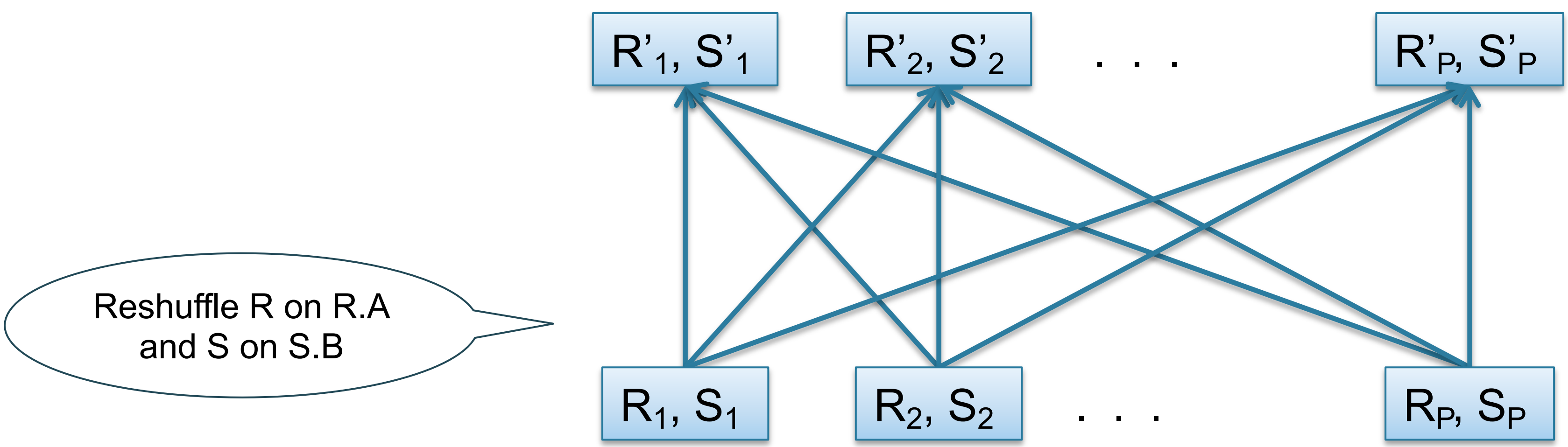
Initially, R and S are block partitioned.
Notice: they may be stored in DFS (recall MapReduce)

Some servers hold R-chunks, some hold S-chunks, some hold both

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

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

Query: $R \bowtie_{A=B} S$



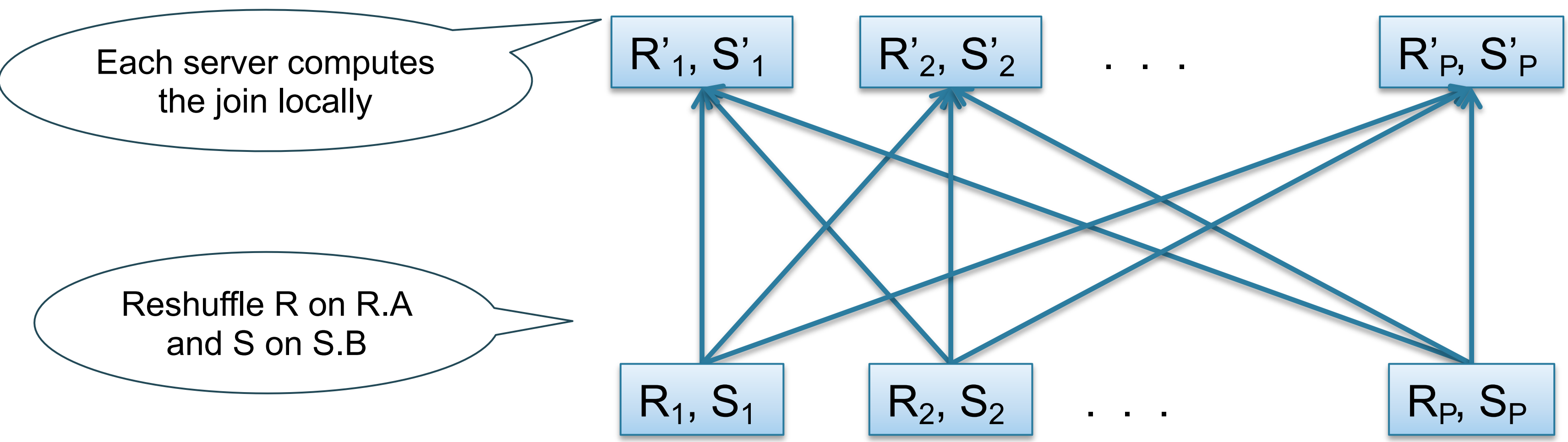
Initially, R and S are block partitioned.
Notice: they may be stored in DFS (recall MapReduce)

Some servers hold R-chunks, some hold S-chunks, some hold both

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

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

Query: R $\bowtie_{A=B}$ S



Initially, R and S are block partitioned.
Notice: they may be stored in DFS (recall MapReduce)

Some servers hold R-chunks, some hold S-chunks, some hold both

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

- Step 1
  - Every server holding any chunk of R partitions its chunk using a hash function h(t.A)
  - Every server holding any chunk of S partitions its chunk using a hash function h(t.B)

- Step 2:
  - Each server computes the join of its local fragment of R with its local fragment of S

Broadcast Join

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

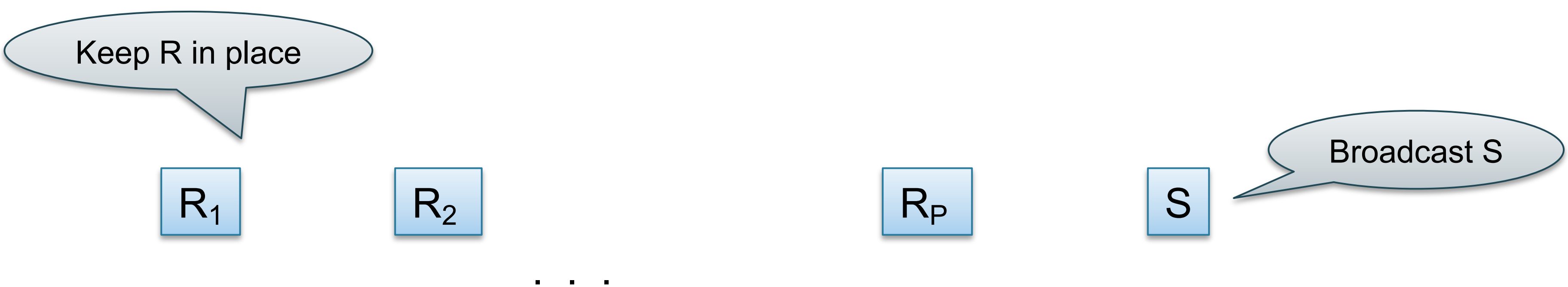
- Also called a small join or a broadcast join

Query: $R \bowtie S$

Broadcast Join

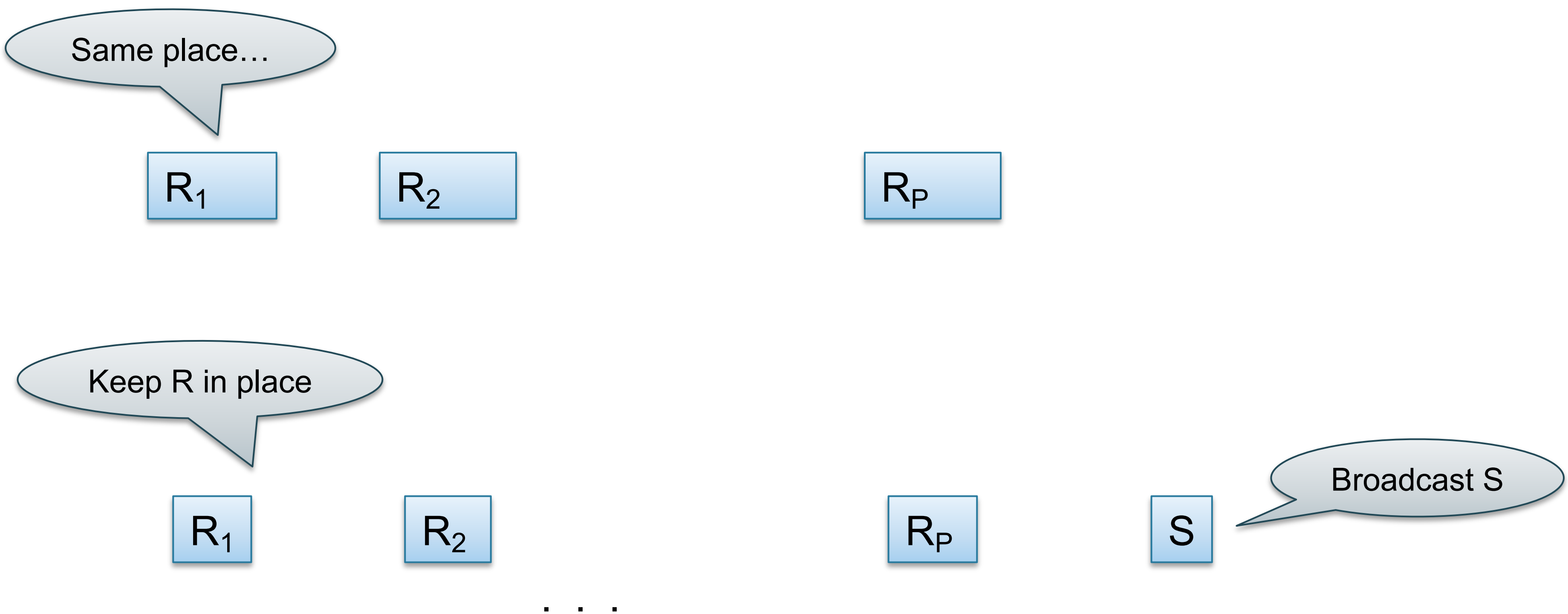
Query: $R \bowtie S$

Broadcast Join



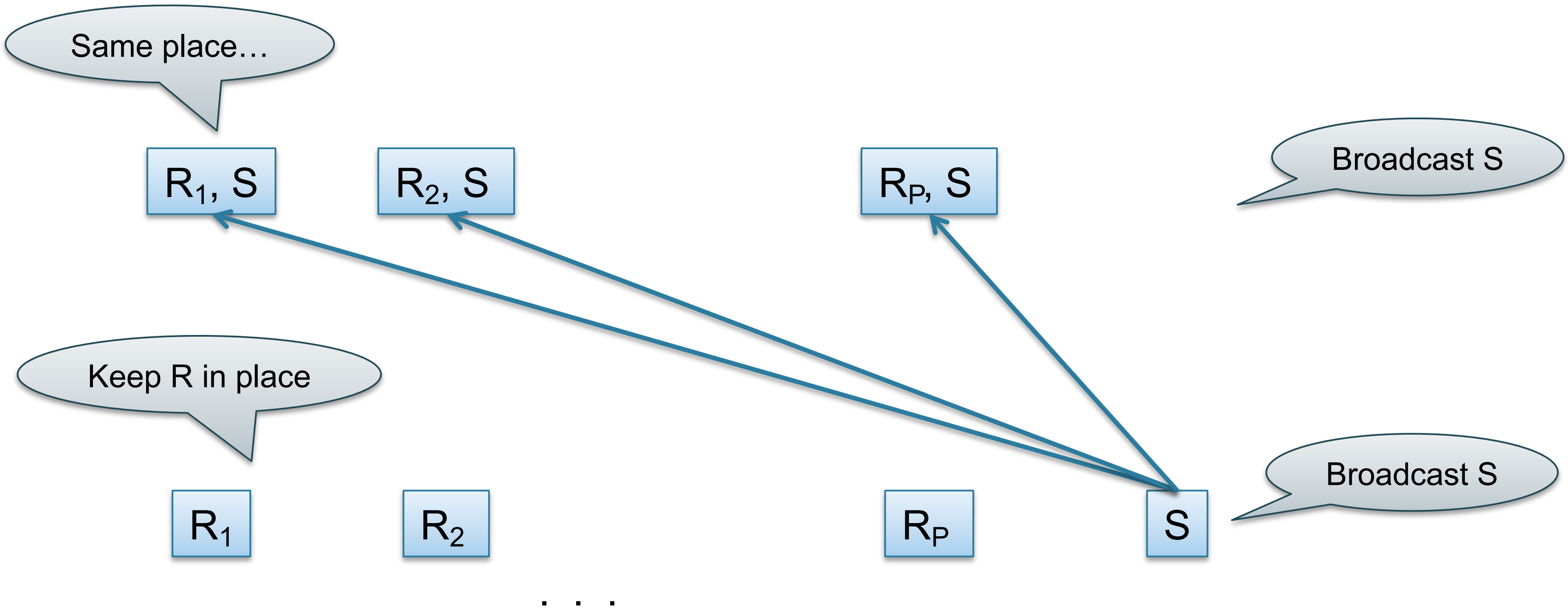
Query: $R \bowtie S$

Broadcast Join



Query: $R \bowtie S$

Broadcast Join



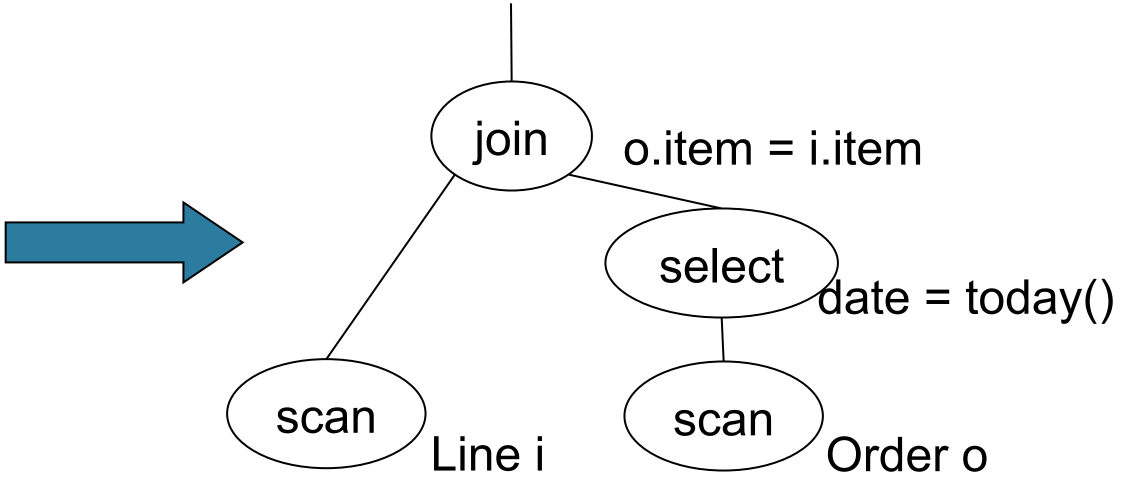
Skew-Join

- Hash-join:
  - Both relations are partitioned (good)
  - May have skew (bad)
- Broadcast join
  - One relation must be broadcast (bad)
  - No worry about skew (good)
- Skew join (has other names):
  - Combine both: in class

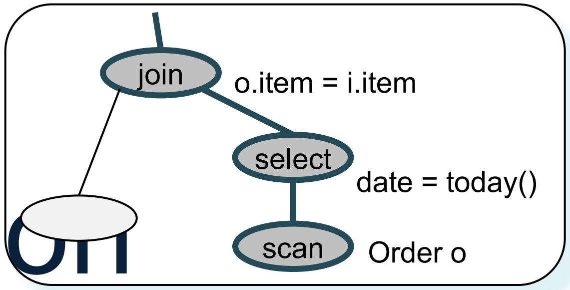
Example Query Execution

Find all orders from today, along with the items ordered

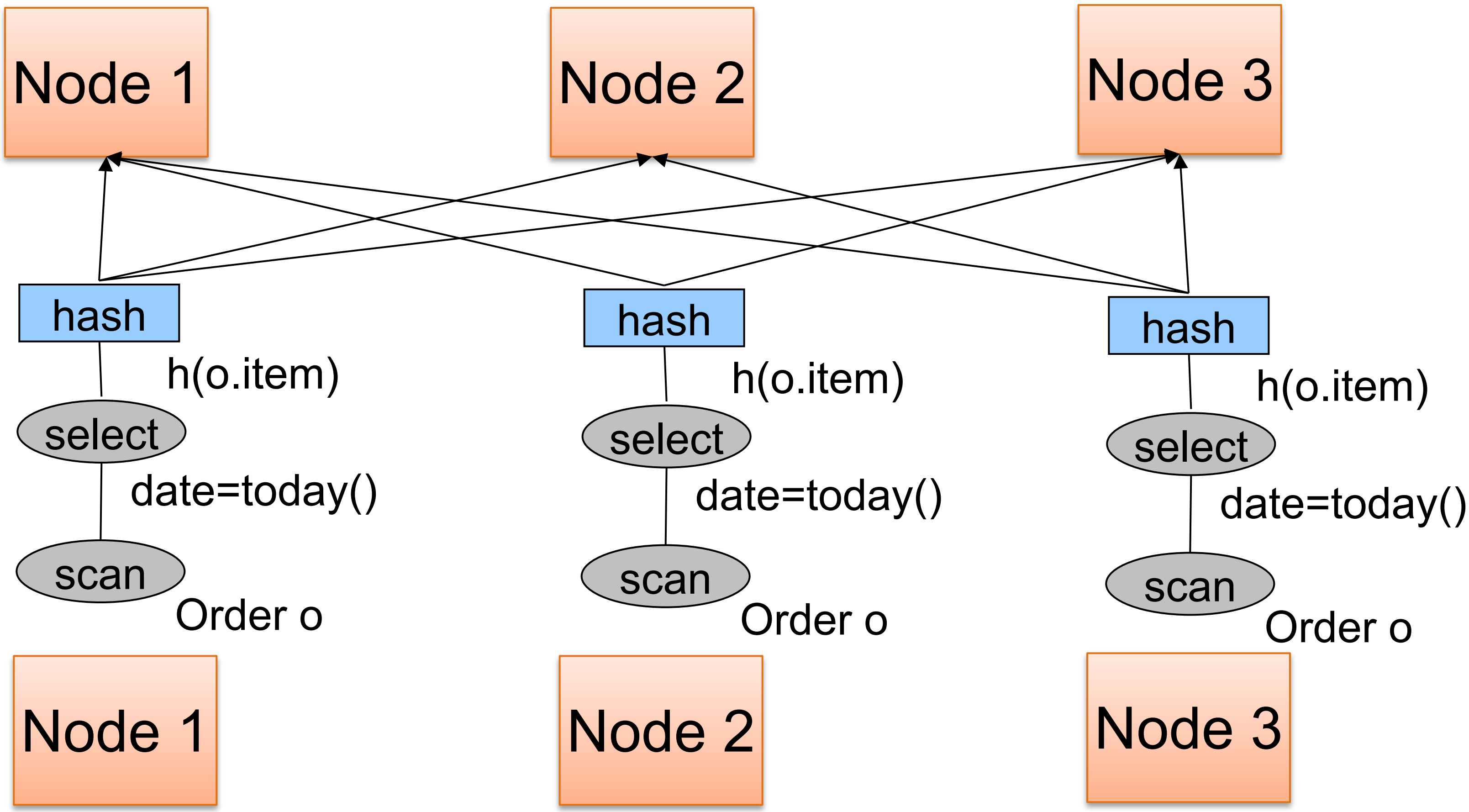
```
SELECT *
FROM Order o, Line i
WHERE o.item = i.item
    AND o.date = today()
```



Order(oid, item, date), Line(item, …)

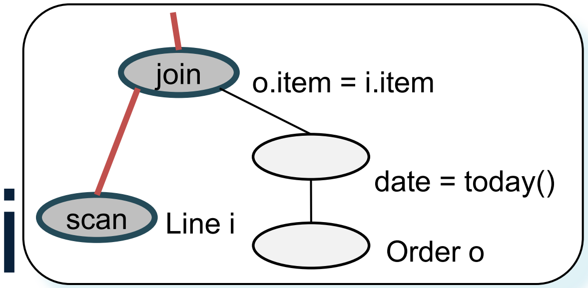


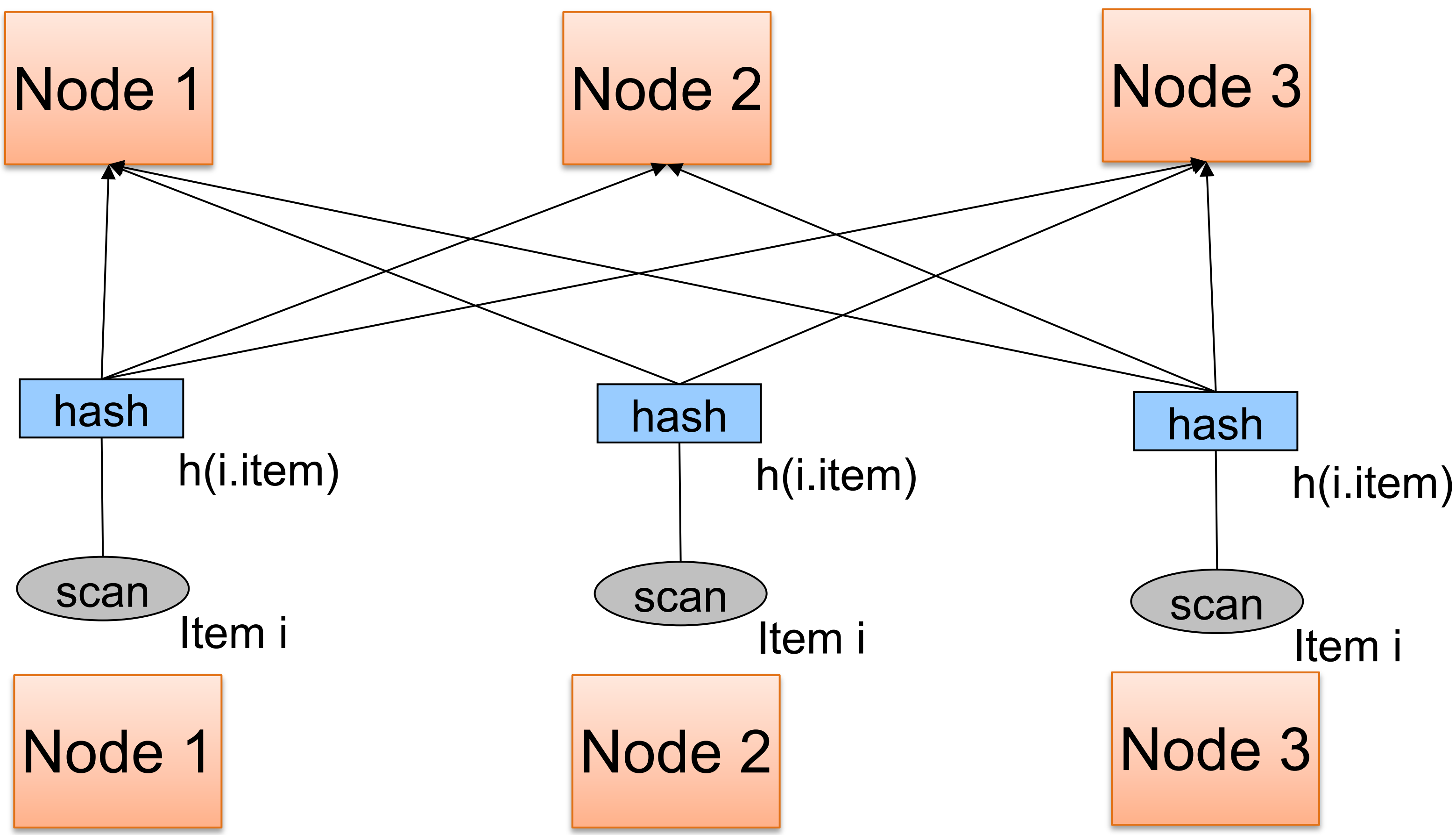
Query Execution



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

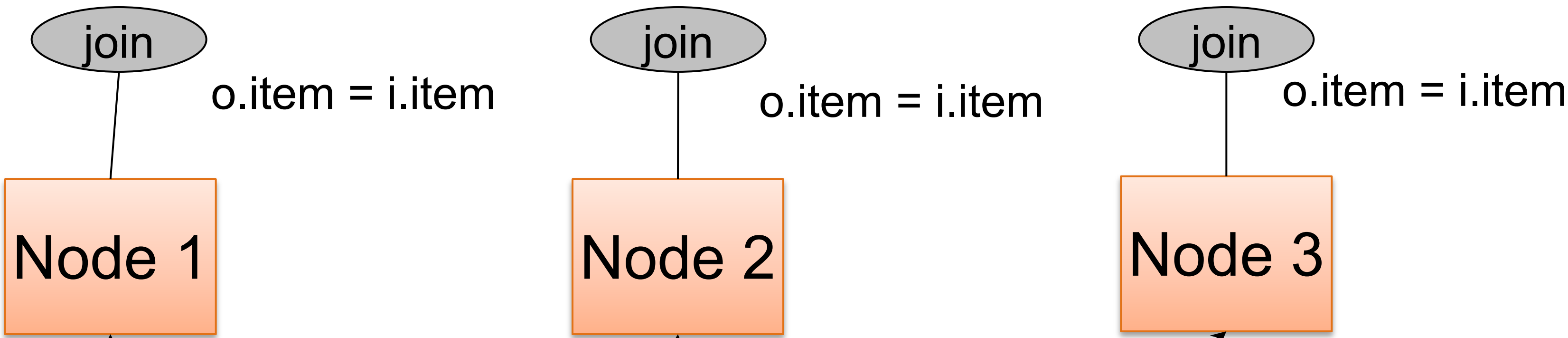
Query Executi





Order(oid, item, date), Line(item, …)

Query Execution



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

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

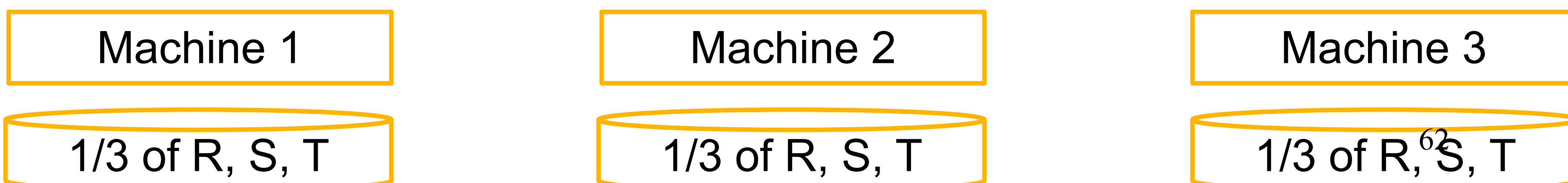
contains all orders and all lines where hash(item) = 1

Example 2

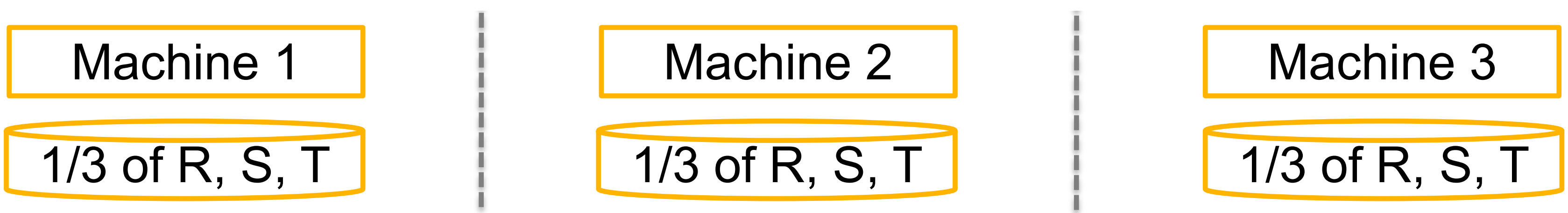
SELECT *

FROM R, S, T

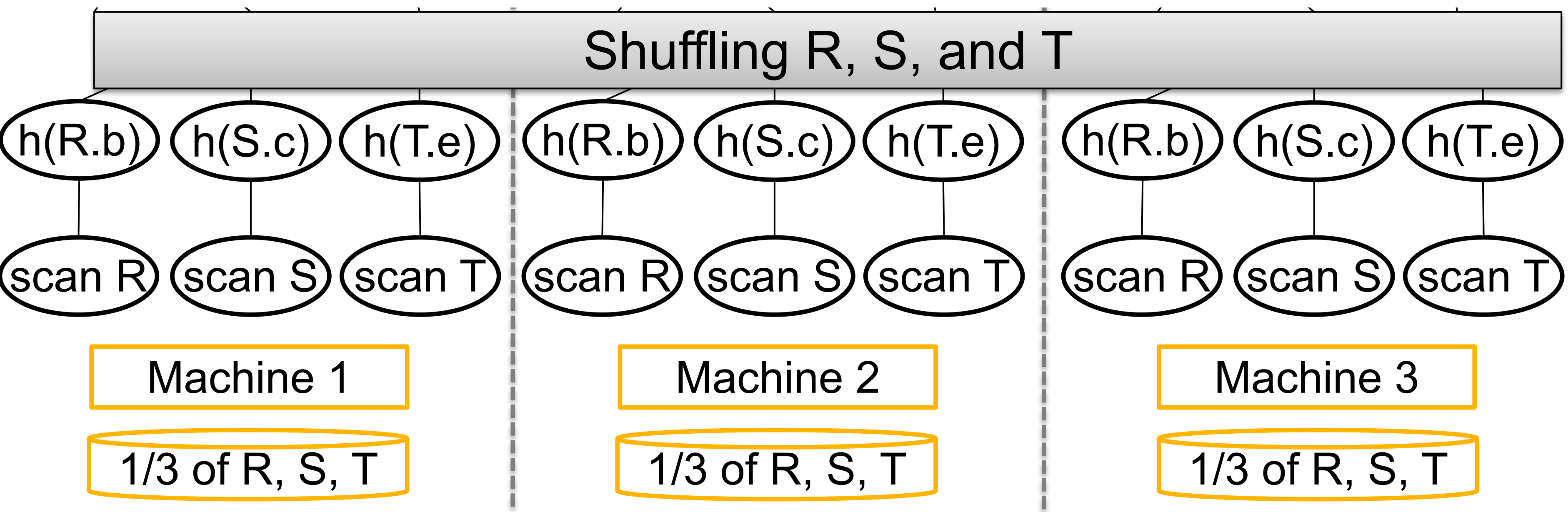
WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100



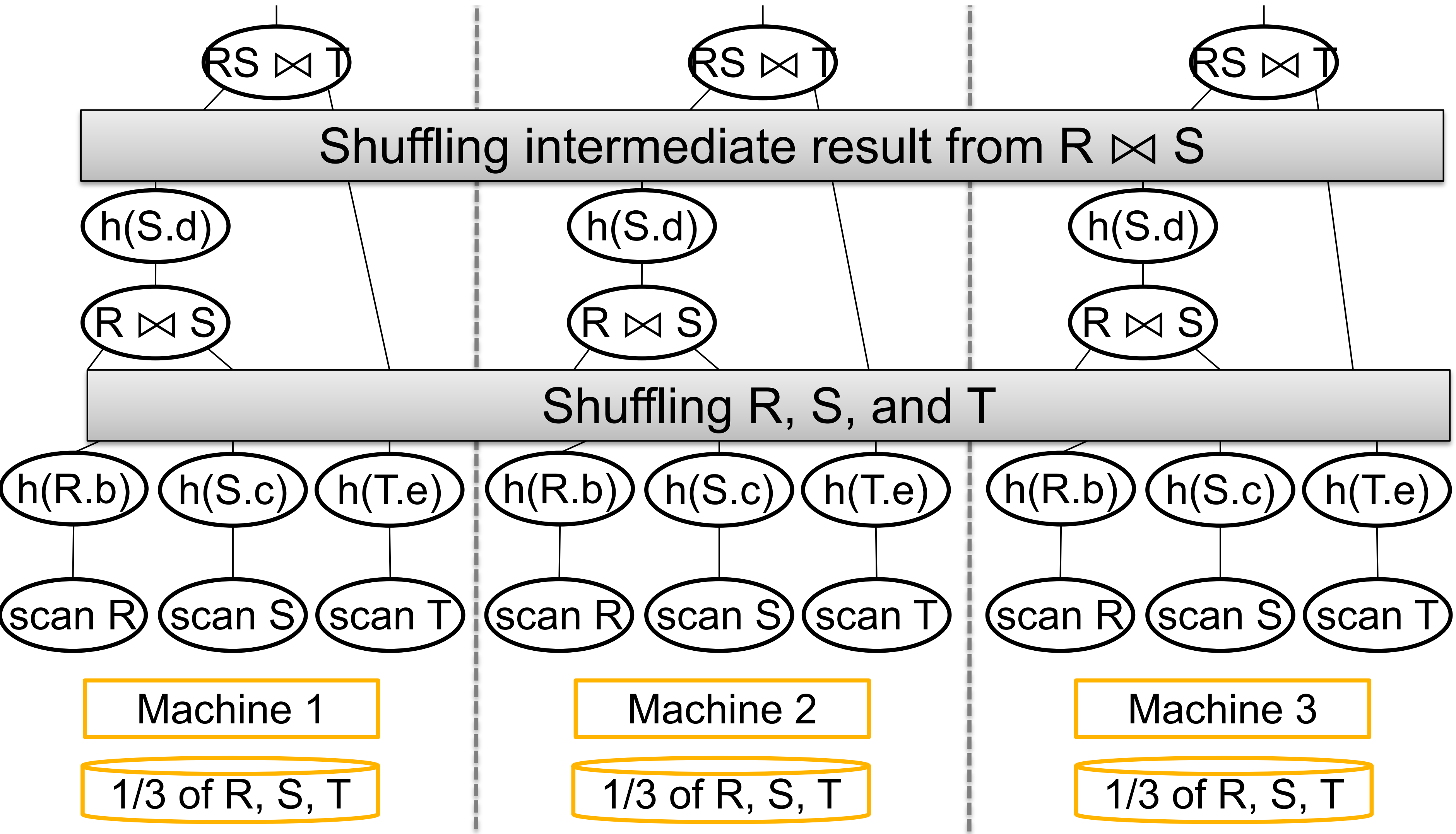
… WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100



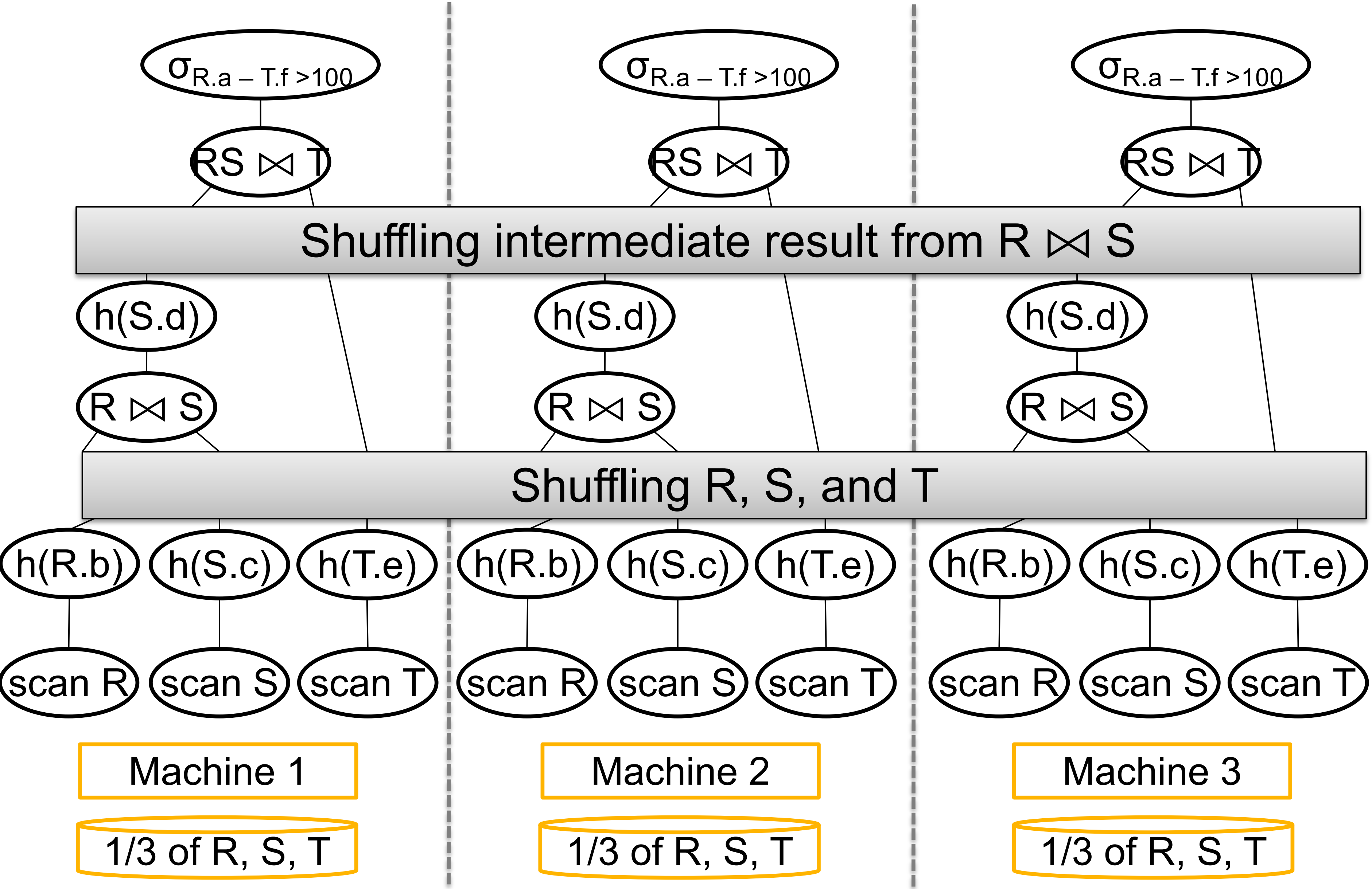
… WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100



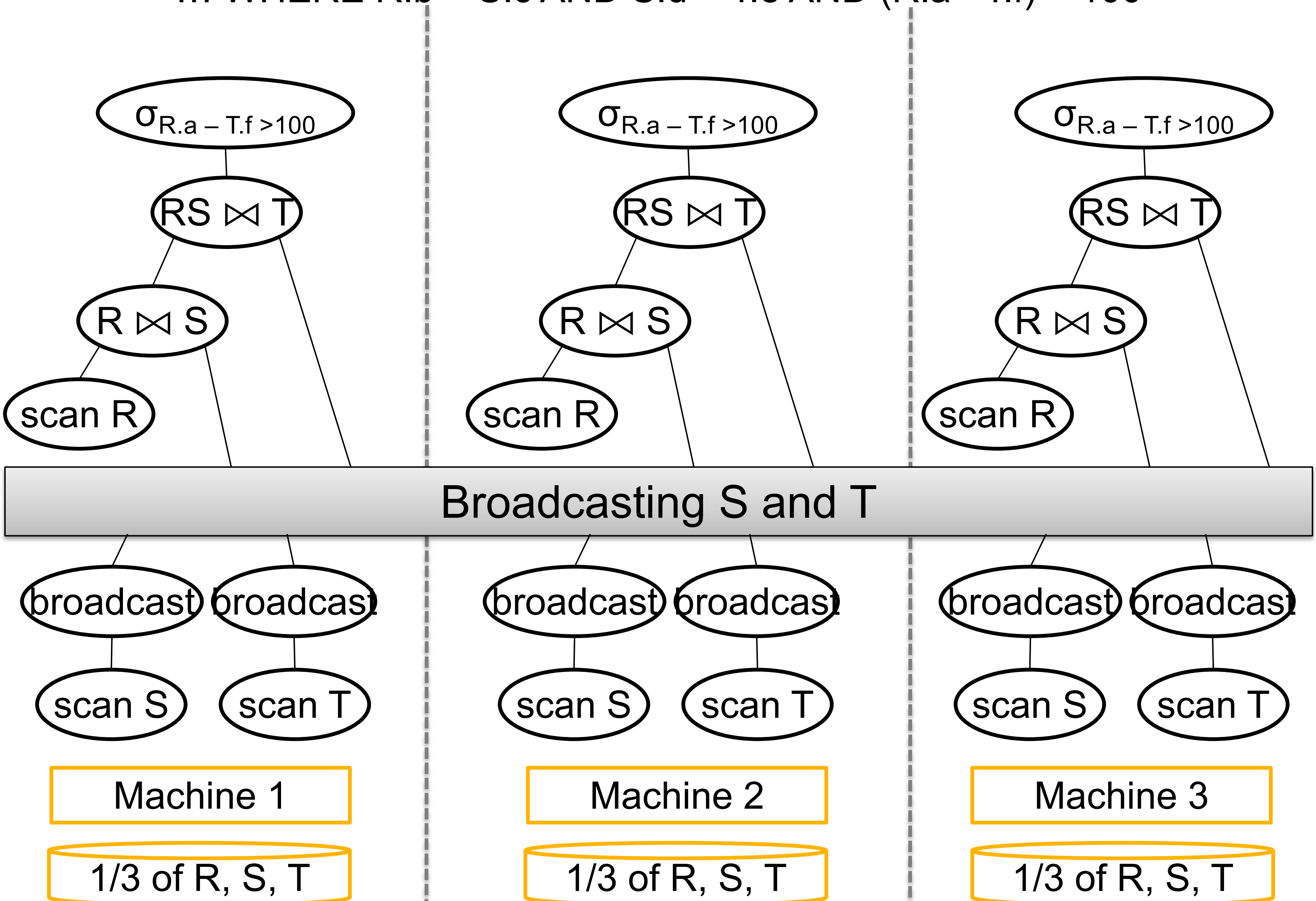
… WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100



… WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100



… WHERE R.b = S.c AND S.d = T.e AND (R.a - T.f) > 100



Skew

Skew

- Skew in the input: a data value has much higher frequency than others
- Skew in the output: a server generates many more values than others, e.g. join
- Skew in the computation

Simple Skew Handling Techniques

For range partition:

- Ensure each range gets same number of tuples

- E.g.: {1, 1, 1, 2, 3, 4, 5, 6 } → [1,2] and [3,6]

- Eq-depth v.s. eq-width histograms

Simple Skew Handling Techniques

Skew in the computation:

- Create more partitions than nodes
  - "virtual servers"
- And be smart about scheduling the partitions
- Note: MapReduce uses this technique

Skew for Hash Partition

Relation R(A,B,C,…), we hash-partition on A

If A is a key: we expect a uniform partition

Skew for Hash Partition

Relation R(A,B,C,…), we hash-partition on A

If A is a key: we expect a uniform partition

If A is not a key:

- Some value A=v may occur very many times
  - The "Justin Bieber" effect ☺
  - v is called a "heavy hitter"

Skew for Hash Partition

Relation R(A,B,C,…), we hash-partition on A

If A is a key: we expect a uniform partition

If A is not a key:

- Some value A=v may occur very many times
  - The “Justin Bieber” effect ☺
  - v is called a “heavy hitter”
- Records with value v hashed to same server i
- Partition R_i is much larger than |R|/p; skew!!

Analyzing Heavy Hitters

- We will discuss how to choose the threshold such that a value that occurs more times than the threshold becomes a “heavy hitters”
- This analysis is based on Cernoff bounds, which is a general technique that is useful in statistics and randomized algorithm

Problem Statement

Given: N data items $v_1, \ldots, v_N$

- We hash-partition them to P nodes
- When is the partitioning uniform?

Problem Statement

Given: N data items $v_1, \ldots, v_N$

- We hash-partition them to P nodes
- When is the partitioning uniform?

Uniform: each node has $O(N/P)$ items

Problem Statement

Given: N data items $v_1, \ldots, v_N$

- We hash-partition them to P nodes
- When is the partitioning uniform?

Uniform: each node has O(N/P) items

Skew: some node has >> N/P items

Problem Statement

Given: N data items $v_1, \ldots, v_N$

- We hash-partition them to P nodes
- When is the partitioning uniform?

Uniform: each node has O(N/P) items

Skew: some node has >> N/P items

1. Due to the hash function h, or
2. Due to skew in the data

Role of the Hash Function

Assume $v_1, \ldots, v_N$ are distinct

Hash function computes $h(v_i) \in \{1,\ldots,P\}$

- If h is *fixed* then we can find bad items that will overload one server; how?
- If h is *random*: *balls-in-bins* problem; we analyze it using the Cernoff bound

Note:
very many
variants

The Cernoff Bound

Bernoulli r.v.: $X_1, \ldots, X_N \in \{0,1\}$

For all i, $\Pr(X_i = 1) = \mu \in (0,1)$

We are interested in $Y = X_1 + X_2 + \cdots + X_N$

Fact: $E[Y] = N\mu$

Theorem (Cernoff bound). If they are iid then:

$$\Pr(Y > (1+\delta)E[Y]) \leq exp\left(-\frac{\delta^2}{3}E[Y]\right)$$

Role of the Hash Function

Fix one server j;

Define indicator variables:

$$X_1 = [h(v_1) = j], \ldots, X_N = [h(v_N) = j]$$
$$\Pr(X_1 = 1) = \cdots = \Pr(X_N = 1) = 1/P$$

Load of server j: $\mathrm{Load}(\mathrm{j}) = X_1 + X_2 + \cdots + X_N$

Expected load: $\mathrm{E}[\mathrm{Load}(\mathrm{j})] = N/P$

Role of the Hash Function

Load of server j: $\text{Load}(\text{j}) = X_1 + X_2 + \cdots + X_N$

Expected load: $\text{E}[\text{Load}(\text{j})] = \frac{N}{P}$

Role of the Hash Function

Load of server j: $\mathrm{Load}(j) = X_1 + X_2 + \cdots + X_N$

Expected load: $\mathrm{E}[\mathrm{Load}(j)] = \frac{N}{P}$

Why?

Case 1: $v_1, \ldots, v_N$ distinct; then $X_1, \ldots, XN$ are iid.

Role of the Hash Function

Load of server j: $\mathrm{Load(j)} = X_1 + X_2 + \cdots + X_N$

Expected load: $\mathrm{E[Load(j)]} = \frac{N}{P}$

Why?

Case 1: v_1, …, v_N distinct; then $X_1, \ldots, XN$ are iid.

Skew at j

Cernoff: $\Pr\left(\mathrm{Load(j)} > (1+\delta)\frac{N}{P}\right) \leq exp\left(-\frac{\delta^2}{3}\frac{N}{P}\right)$

Role of the Hash Function

Load of server j: $\mathrm{Load}(\mathrm{j}) = X_1 + X_2 + \cdots + X_N$

Expected load: $\mathrm{E}[\mathrm{Load}(\mathrm{j})] = \frac{N}{P}$

Why?

Case 1: $v_1, \ldots, v_N$ distinct; then $X_1, \ldots, XN$ are iid.

Skew at j

Cernoff: $\Pr\left(\mathrm{Load}(\mathrm{j}) > (1+\delta)\frac{N}{P}\right) \le exp\left(-\frac{\delta^2}{3}\frac{N}{P}\right)$

Union bound: $\Pr(Skew) \le P \cdot exp\left(-\frac{\delta^2}{3}\frac{N}{P}\right)$

Skew at 1 or at 2 … or at P

Role of the Hash Function

Case 1: $v_1, \ldots, v_N$ distinct:

$$\Pr(Skew) \leq P \cdot exp\left(-\frac{\delta^2}{3}\frac{N}{P}\right)$$

Discussion: usually N >> P

Role of the Hash Function

Case 1: $v_1, \ldots, v_N$ distinct:

$$\Pr(Skew) \leq P \cdot exp\left(-\frac{\delta^2}{3}\frac{N}{P}\right)$$

Discussion: usually N >> P

- E.g. want load/server < 30% above expected, then $\delta = 0.3$ Assume N=10^9 and P=10^3

Role of the Hash Function

Case 1: $v_1, \ldots, v_N$ distinct:

$$\Pr(Skew) \leq P \cdot exp\left(-\frac{\delta^2}{3}\frac{N}{P}\right)$$

Discussion: usually N >> P

- E.g. want load/server < 30% above expected, then $\delta = 0.3$ Assume N=10^9 and P=10^3

$$\Pr(Skew) \leq 1000 \cdot \mathrm{e}^{-\frac{0.09}{3}10^6} = 1000 \cdot e^{-3\cdot 10^4} \approx 0$$

Role of the Hash Function

Case 1: $v_1, \ldots, v_N$ distinct:

$$\Pr(Skew) \leq P \cdot exp\left(-\frac{\delta^2}{3}\frac{N}{P}\right)$$

Discussion: usually N >> P

- Start worrying only when $N \approx P \ln P$ (why?)

Role of the Hash Function

- Don't write your own has function!
- Randomize it (how?)
- Make sure N >> P (if not, why parallelize?)

Take away: a good hash function shall not cause skew!

Role of the Data Skew

Case 2: $v_1, \ldots, v_N$ have duplicates

Call v_i a *heavy hitter* if it occurs >> N/P times

Role of the Data Skew

Case 2: $v_1, \ldots, v_N$ have duplicates

Call v_i a ***heavy hitter*** if it occurs >> N/P times

Fact if there exists a heavy hitter, then there exists a server j s.t. $\text{Load}(j) \gg \frac{N}{P}$

Role of the Data Skew

Case 2: $v_1, \ldots, v_N$ have duplicates

Call v_i a *heavy hitter* if it occurs >> N/P times

Fact if there exists a heavy hitter, then there exists a server j s.t. $\mathrm{Load}(j) \gg \frac{N}{P}$

Therefore: $\Pr(Skew)$=1

Role of the Data Skew

Case 2: $v_1, \ldots, v_N$ have duplicates

Call v_i a ***heavy hitter*** if it occurs >> N/P times

Fact if there exists a heavy hitter, then there exists a server j s.t. $\mathrm{Load}(j) \gg \frac{N}{P}$

Therefore: $\Pr(Skew) = 1$

No hash function can handle heavy hitters

Role of the Data Skew

Case 3: $v_1, \ldots, v_N$ have duplicates, no heavy hitters

Assume each value occurs $\frac{N}{cP}$ times, for $c > 1$

$$\underbrace{v_1, v_1, \ldots, v_1}_{\frac{N}{cP}}, \underbrace{v_2, v_2, \ldots, v_2}_{\frac{N}{cP}}, \ldots$$

cP distinct values

Role of the Data Skew

Case 3: $v_1, \ldots, v_N$ have duplicates, no heavy hitters

Assume each value occurs $\frac{N}{cP}$ times, for $c > 1$

$$\underbrace{v_1, v_1, \ldots, v_1}_{\frac{N}{cP}}, \underbrace{v_2, v_2, \ldots, v_2}_{\frac{N}{cP}}, \ldots$$

cP distinct values

$$X_1 = [h(v_1) = j], X_2 = [h(v_2) = j], \ldots$$

Role of the Data Skew

Case 3: $v_1, \ldots, v_N$ have duplicates, no heavy hitters

Assume each value occurs $\frac{N}{cP}$ times, for $c > 1$

$$\underbrace{v_1, v_1, \ldots, v_1}_{\frac{N}{cP}}, \underbrace{v_2, v_2, \ldots, v_2}_{\frac{N}{cP}}, \ldots$$

cP distinct values

$X_1 = [h(v_1) = j], X_2 = [h(v_2) = j], \ldots$

$Y = \sum_i X_i \qquad E[Y] = c \qquad Load(j) = Y\frac{N}{cP}$

$\Pr(\text{Skew}) \leq P \cdot \Pr(Y > (1+\delta)E[Y])$

Role of the Data Skew

Case 3: $v_1, \ldots, v_N$ have duplicates, no heavy hitters

Assume each value occurs $\frac{N}{cP}$ times, for $c > 1$

$$\underbrace{v_1, v_1, \ldots, v_1}_{\frac{N}{cP}}, \underbrace{v_2, v_2, \ldots, v_2}_{\frac{N}{cP}}, \ldots$$

cP distinct values

$$X_1 = [h(v_1) = j], X_2 = [h(v_2) = j], \ldots$$

$$Y = \sum_i X_i \qquad E[Y] = c \qquad Load(j) = Y\frac{N}{cP}$$

$$\Pr(\text{Skew}) \le P \cdot \Pr(Y > (1+\delta)E[Y]) \le P \cdot exp\left(-\frac{\delta^2 c}{3}\right)$$

Role of the Data Skew

Case 3: $v_1, \ldots, v_N$ have duplicates, no heavy hitters

Assume each value occurs $\frac{N}{cP}$ times, for $c > 1$

$$\underbrace{v_1, v_1, \ldots, v_1}_{\frac{N}{cP}}, \underbrace{v_2, v_2, \ldots, v_2}_{\frac{N}{cP}}, \ldots$$

cP distinct values

$X_1 = [h(v_1) = j], X_2 = [h(v_2) = j], \ldots$

$Y = \sum_i X_i \qquad E[Y] = c \qquad Load(j) = Y\frac{N}{cP}$

$$\Pr(\text{Skew}) \le P \cdot \Pr(Y > (1+\delta)E[Y]) \le P \cdot exp\left(-\frac{\delta^2 c}{3}\right)$$

Need $c \gtrsim \ln P$

Discussion

Use library hash function! Randomize!

- When each value occurs $\leq \frac{N}{P \cdot ln\ P}$ times, then $Load \leq (1+\delta)\frac{N}{P}$ with high probability
- When some value occurs $\gg \frac{N}{P}$ times, the load will be skewed
- Gray area: when values occur $\approx \frac{N}{P}$ times: it can be shown that $Load \approx \frac{N \cdot \ln(P)}{P}$

SkewJoin

Main idea: separate the heavy hitters from the light hitters

- Hash join the light hitters: the partition is uniform because they are light
- Broadcast join the heavy hitters: works because there are very few heavy hitters

SkewJoin: Details

Query: $R \bowtie_{A=B} S$, R.A = foreign key, S.A=key

SkewJoin: Details

Query: R $\bowtie_{A=B}$ S, R.A = foreign key, S.A=key

- Step 1: find the *heavy hitters* in R.A
  - I.e. find the values v=R.A that occur $\geq \frac{N}{P}$ times
  - There are ≤ P heavy hitters! Broadcast them

SkewJoin: Details

Query: R $\bowtie_{A=B}$ S, R.A = foreign key, S.A=key

- Step 1: find the ***heavy hitters*** in R.A
  - I.e. find the values v=R.A that occur $\geq \frac{N}{P}$ times
  - There are ≤ P heavy hitters! Broadcast them
- Step 2: each sever partitions locally:

  $$R = R_{light} \cup R_{heavy}, S = S_{light} \cup S_{heavy}$$

  Notice: $|S_{heavy}| \leq P$ (i.e. it is small)

SkewJoin: Details

Query: R $\bowtie_{A=B}$ S, R.A = foreign key, S.A=key

- Step 1: find the <u>*heavy hitters*</u> in R.A
  - I.e. find the values v=R.A that occur $\geq \frac{N}{P}$ times
  - There are ≤ P heavy hitters! Broadcast them
- Step 2: each sever partitions locally:

  $$R = R_{light} \cup R_{heavy}, S = S_{light} \cup S_{heavy}$$

  Notice: $|S_{heavy}| \leq P$ (i.e. it is small)
- Step 3: hash-join $R_{light} \bowtie S_{light}$

SkewJoin: Details

Query: $R \bowtie_{A=B} S$, R.A = foreign key, S.A=key

- Step 1: find the ***heavy hitters*** in R.A
  - I.e. find the values v=R.A that occur $\geq \frac{N}{P}$ times
  - There are ≤ P heavy hitters! Broadcast them
- Step 2: each sever partitions locally:

  $$R = R_{light} \cup R_{heavy}, S = S_{light} \cup S_{heavy}$$

  Notice: $|S_{heavy}| \leq P$ (i.e. it is small)
- Step 3: hash-join $R_{light} \bowtie S_{light}$
- Step 4: broadcast join $R_{heavy} \bowtie S_{heavy}$

Discussion

- Many distributed query processors do not handle skew well
- (Project idea: how does your favorite engine handle skewed data?)
- In practice, you may need to partition skewed data manually