CSE544
Data Management

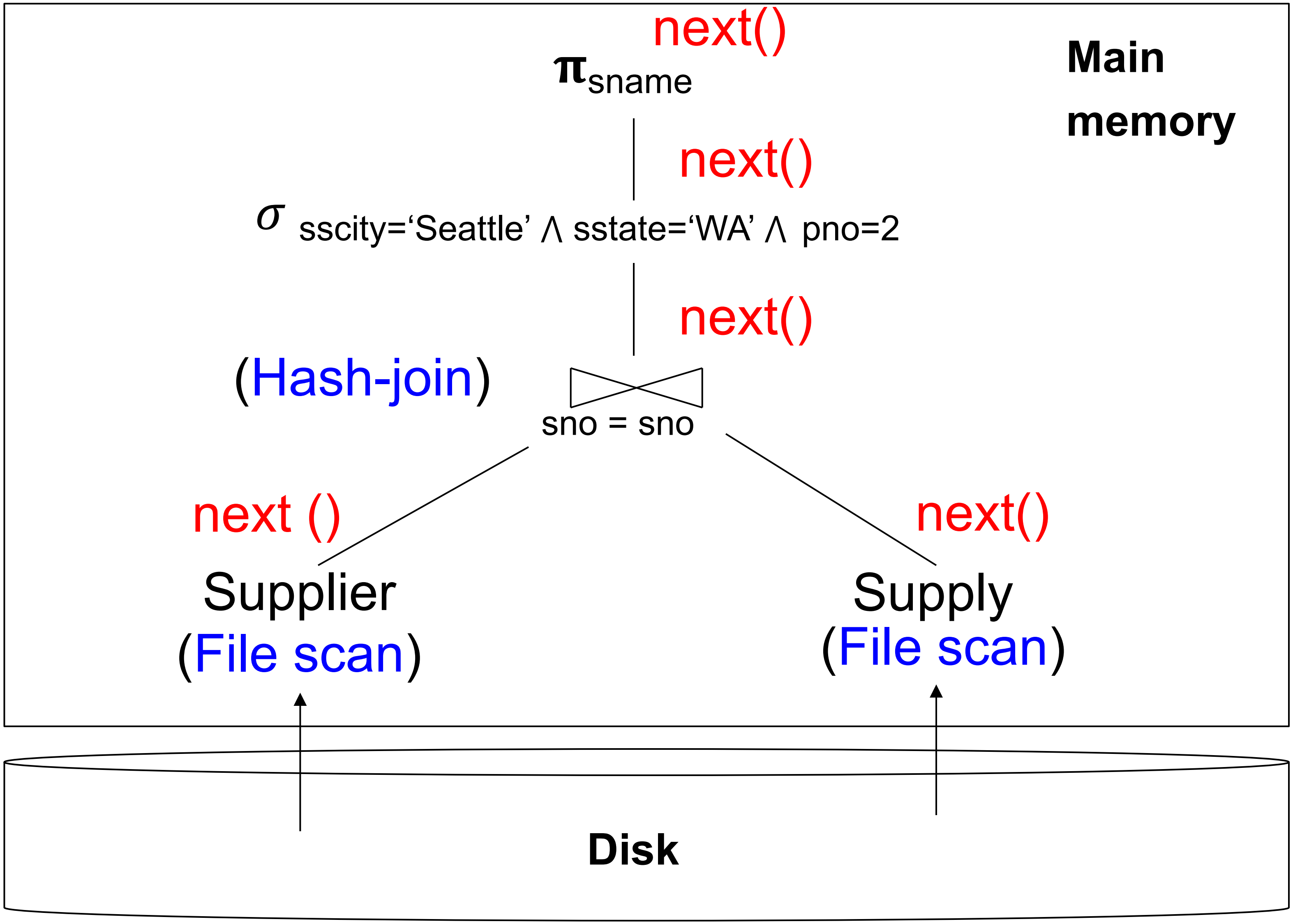
Lectures 13

Parallel Query Processing

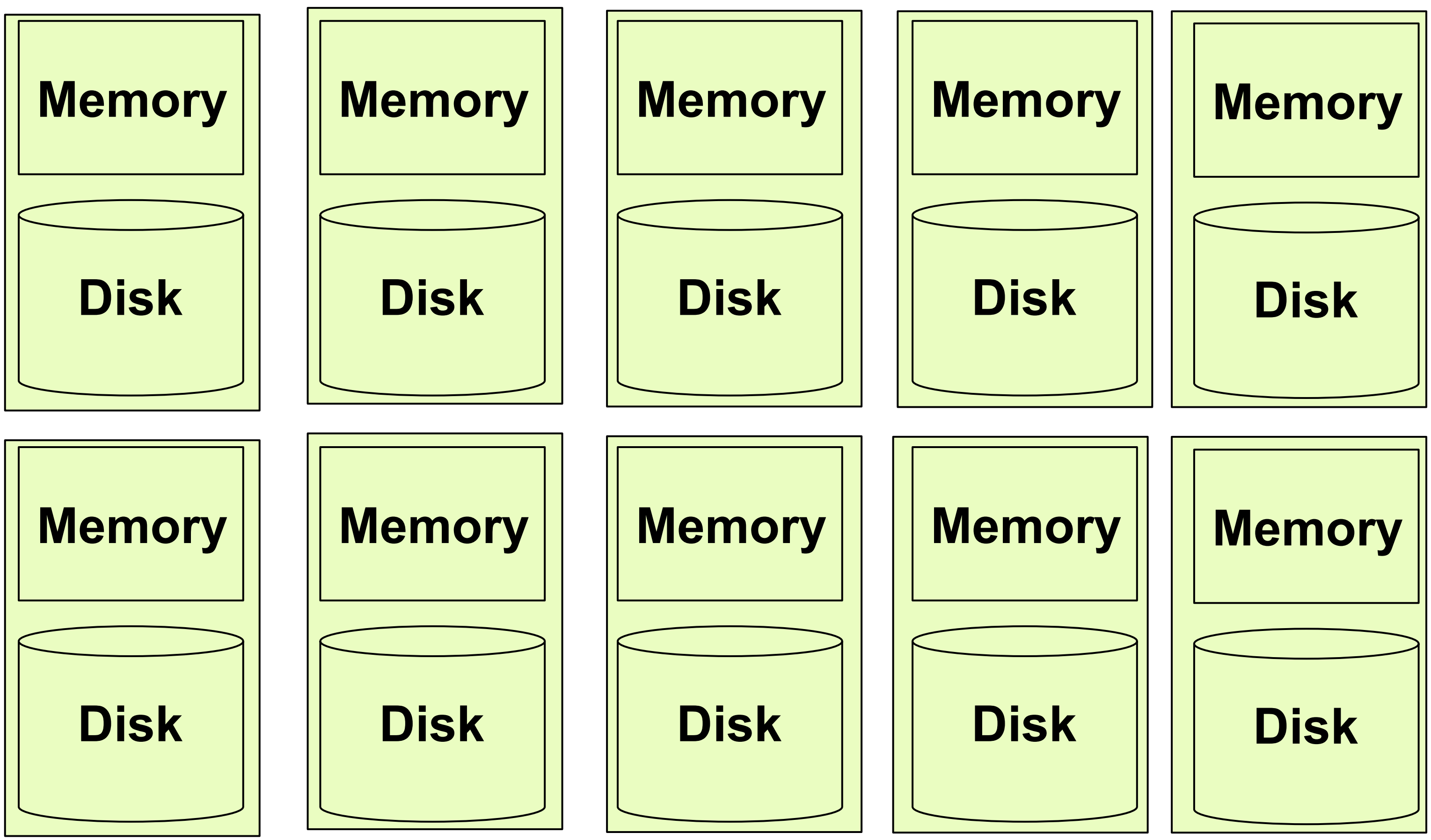
Announcements

- HW 4 is due this Friday
  There was a bug, Walter fixed it
- Review 4, Snowflake, due for Monday
- Project poster presentations next Friday

Serial Query Execution



What if we Have a Cluster and a Large Amount of Data?



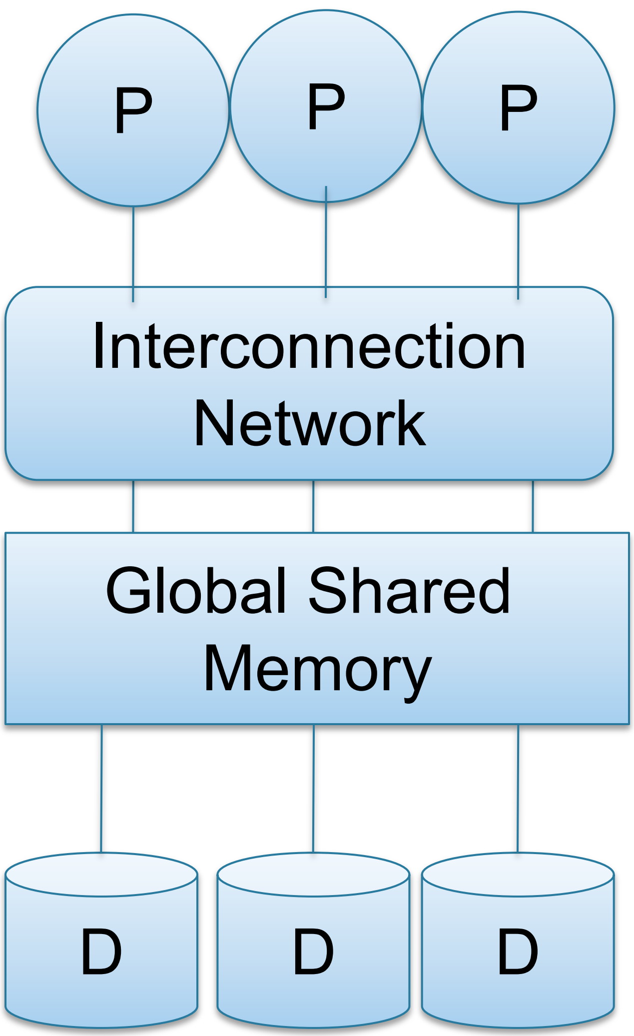
Parallel Query Processing

- Clusters:
  - More servers → more likely to fit data in main memory
  - More servers → more computing power
  - Clusters are now cheaply available in the cloud
- Multicores: the end of Moore's law

Architectures for Parallel Databases

- Shared memory
- Shared disk
- Shared nothing

Shared Memory

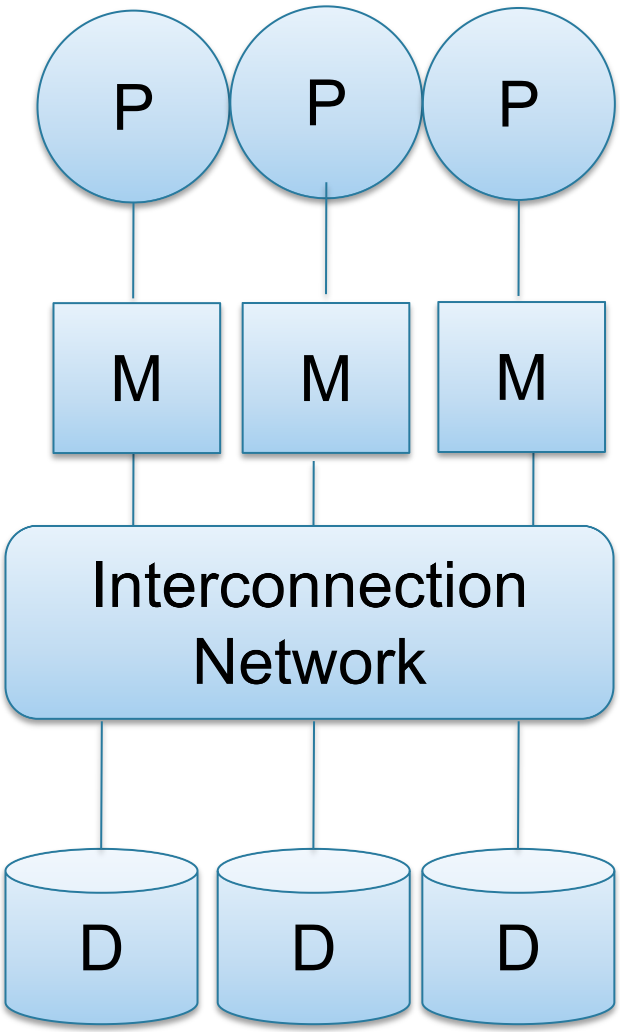


- SMP = symmetric multiprocessor
- Nodes share RAM and disk
- 10x … 100x processors

- 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



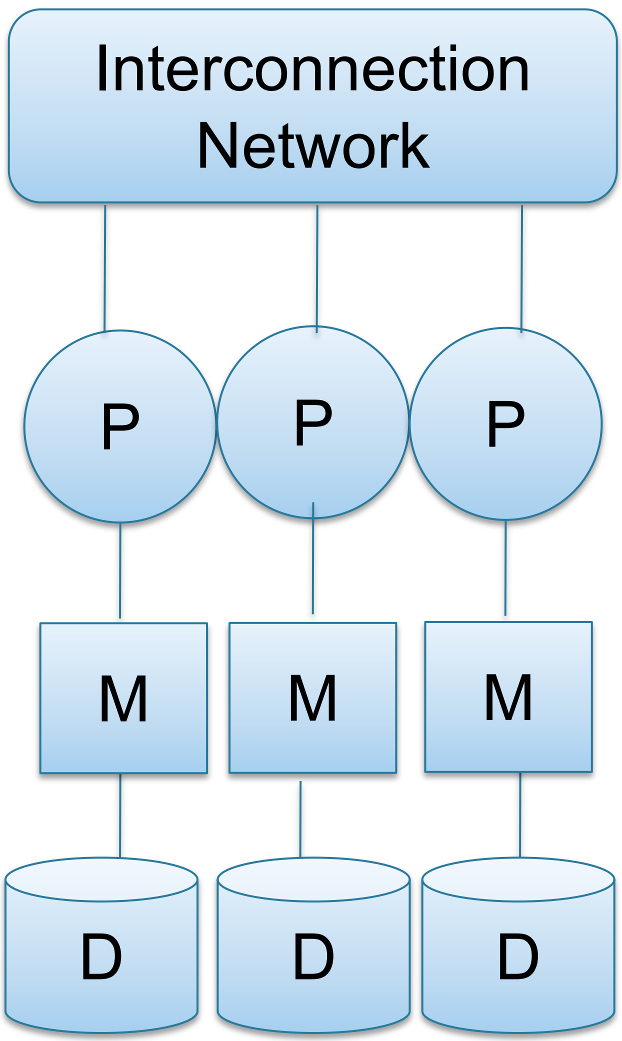
- All nodes access same disks
- 10x processors

- 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.

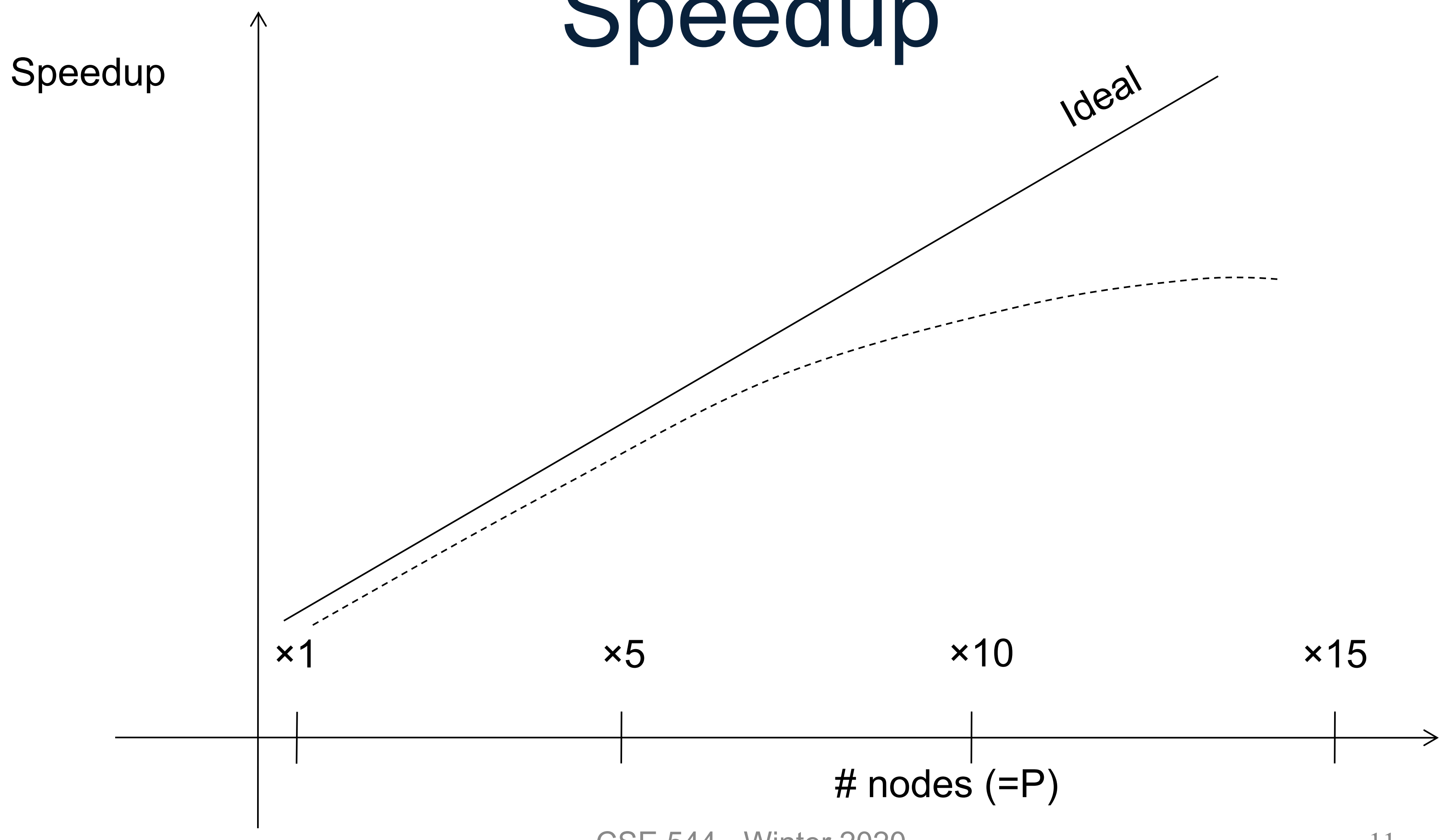
Performance Metrics

Nodes = processors = computers

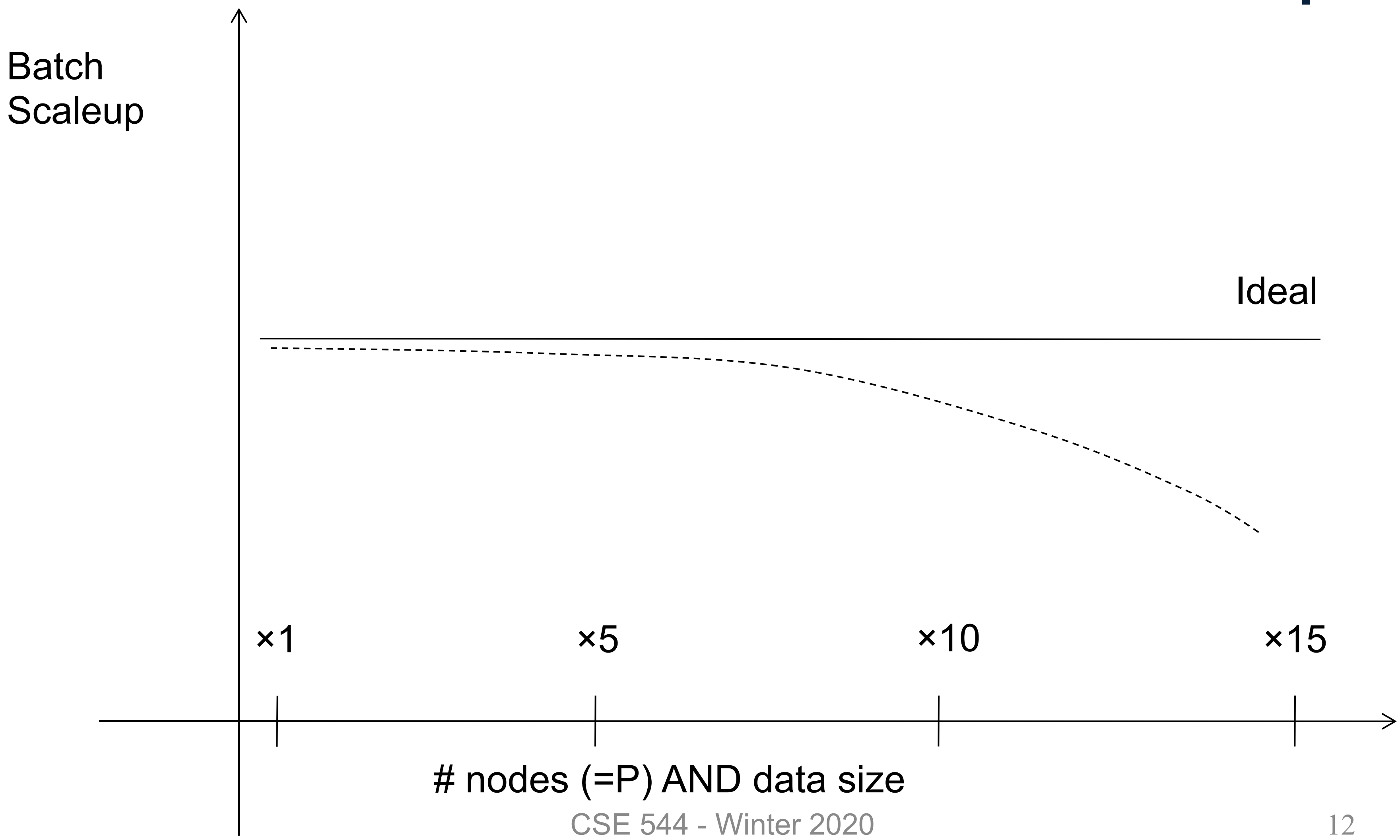
- Speedup:
  - More nodes, same data ➔ higher speed
- Scaleup:
  - More nodes, more data ➔ same speed

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

Super-linear Speedup?

Can we build a machine that achieves super-linear speedup?

Super-linear Speedup?

Can we build a machine that achieves super-linear speedup?

- No! Brent's theorem: If we can run in time T(p) using p processors then we can run in time p*T(p) using 1 processor

Super-linear Speedup?

Can we build a machine that achieves super-linear speedup?

- No! Brent's theorem: If we can run in time T(p) using p processors then we can run in time p*T(p) using 1 processor
- Superlinear means p*T(p) $\rightarrow$ 0;

Super-linear Speedup?

Can we build a machine that achieves super-linear speedup?

- No! Brent's theorem: If we can run in time T(p) using p processors then we can run in time p*T(p) using 1 processor
- Superlinear means p*T(p) $\rightarrow$ 0;
- Then we can run in ≈ 0 time by simulating p ≈ ∞ processors

Parallel Query Execution Algorithms

Basic principle: Data Distribution

- Distribute the n data on the p servers, such that each server only needs to process n/p data items.
- Called *horizontal data partitioning*

Horizontal Data Partitioning

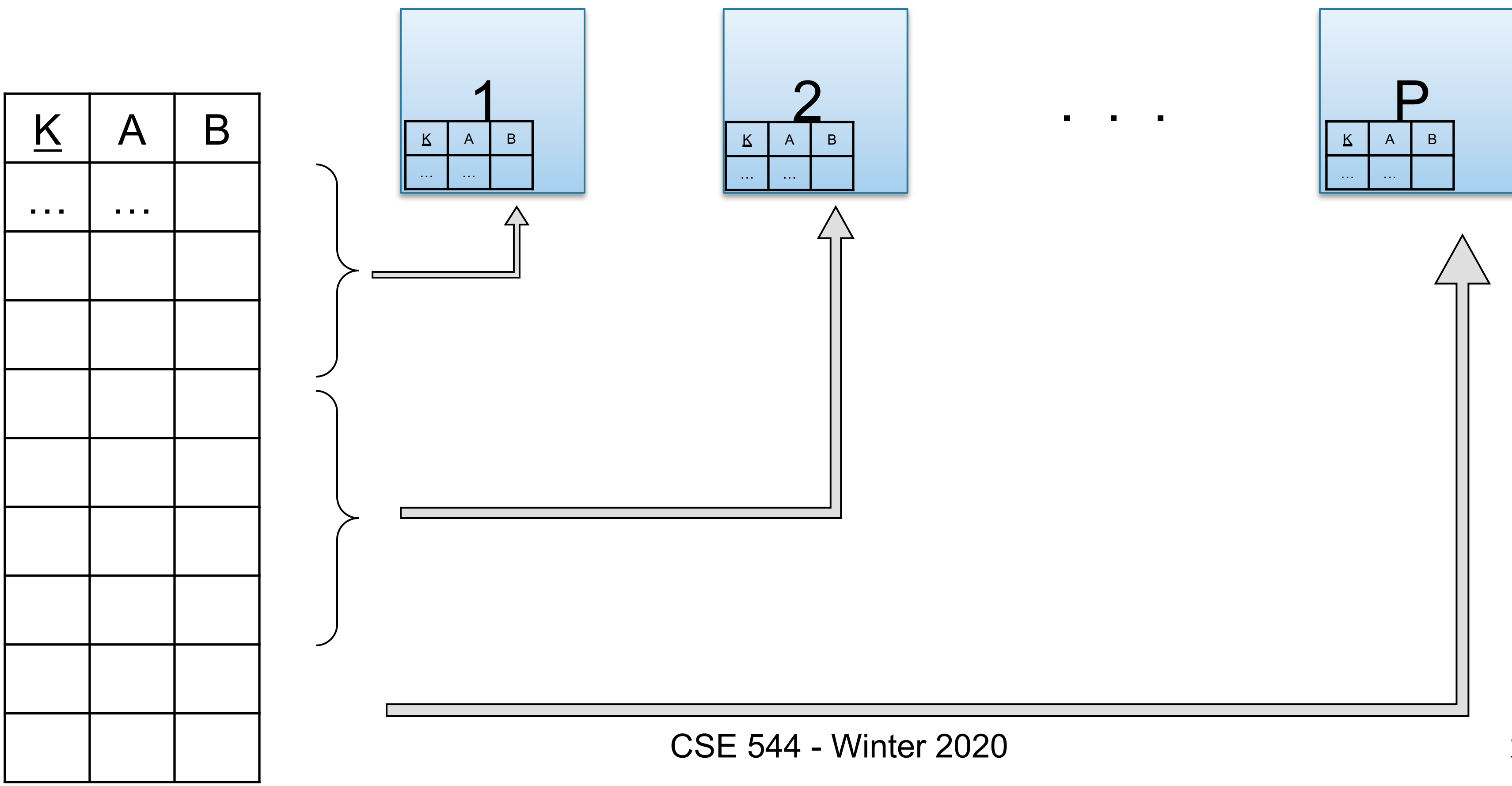
Data:

K	A	B
…	…	

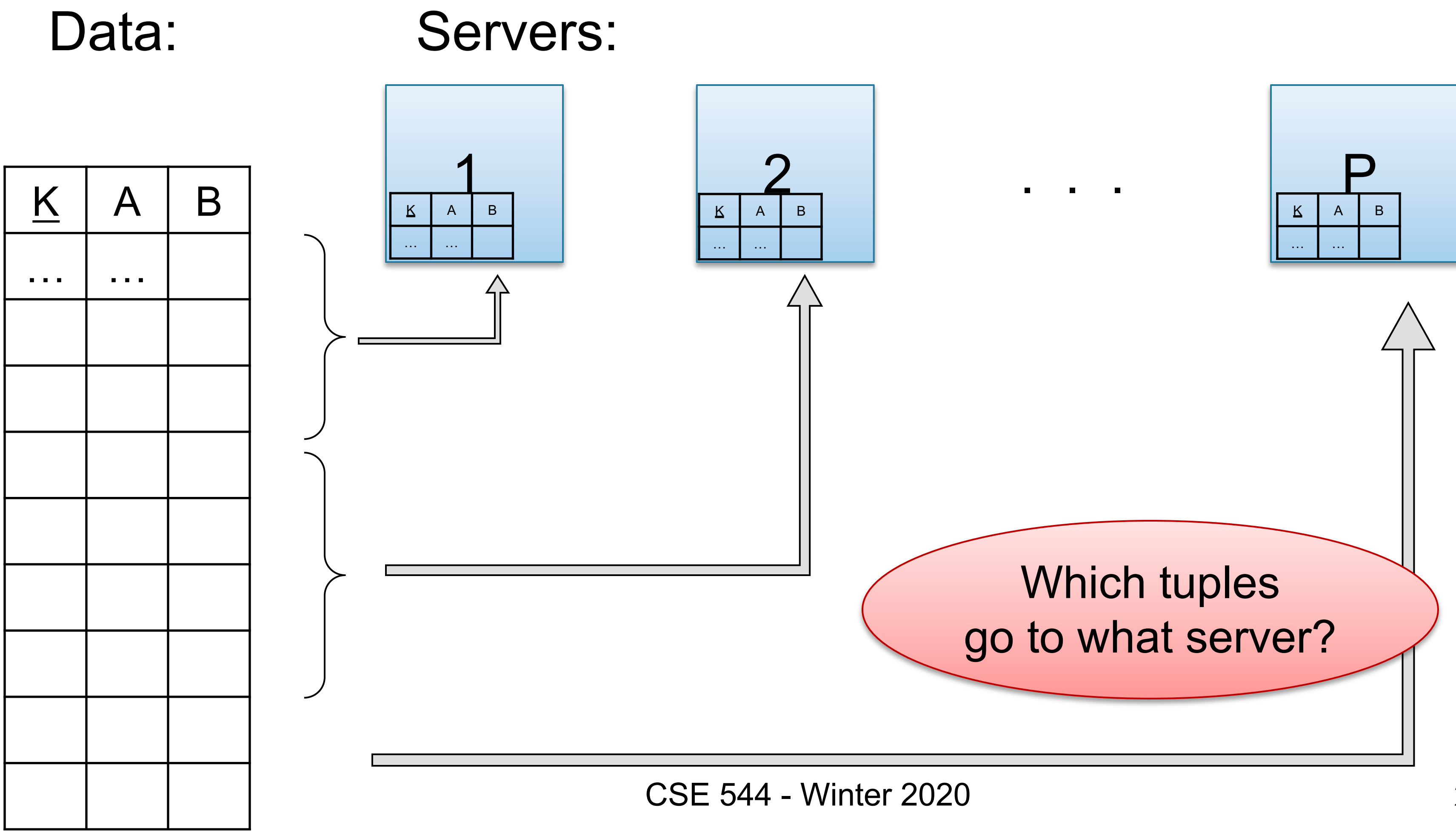
Servers:

Horizontal Data Partitioning

Data: Servers:



Horizontal Data Partitioning



Horizontal Data Partitioning

- Block Partition, a.k.a. Round Robin:
  - 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 Algorithm

- Selection σ
- Join $\bowtie$
- Group by γ

Parallel Selection

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

- Block partitioned:
  - All servers do the work
- Hash partitioned:
  - Only one server does work
- Range partitioned
  - Some servers do the work

Parallel GroupBy

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

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

How do we compute in each case:

- R is hash-partitioned on A
- R is 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

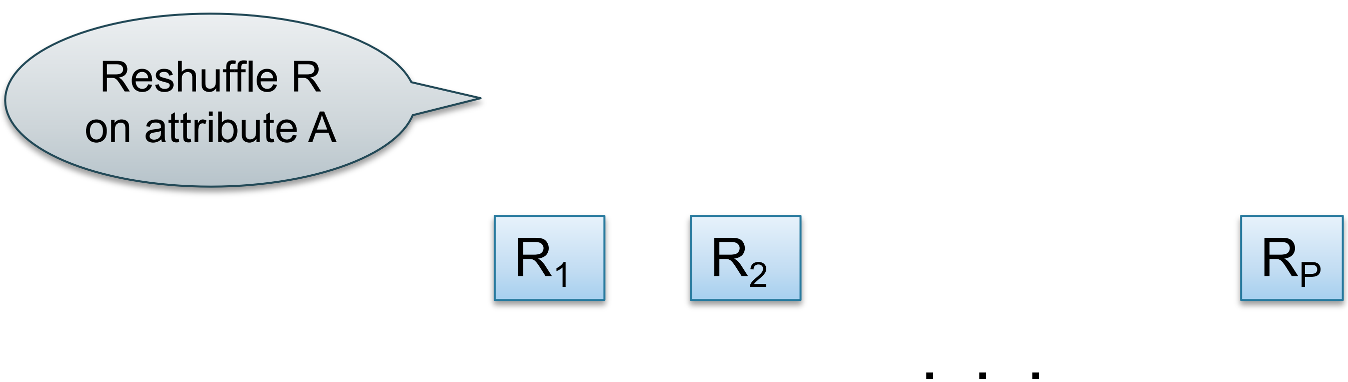
R_1 R_2 . . . R_P

Basic Parallel GroupBy

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

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

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

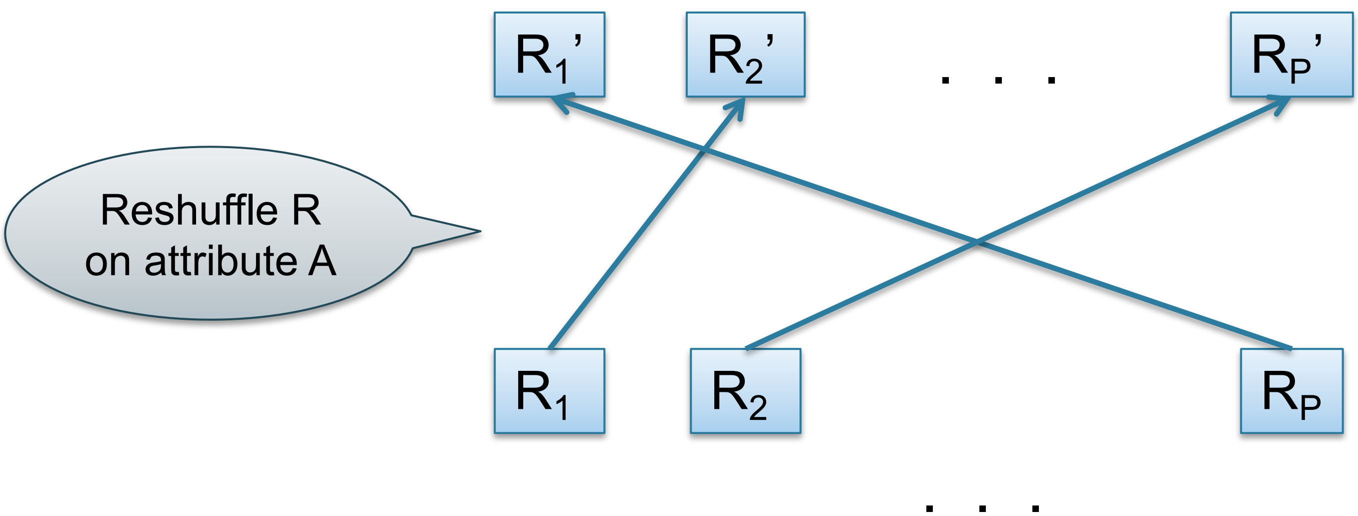


Basic Parallel GroupBy

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

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

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

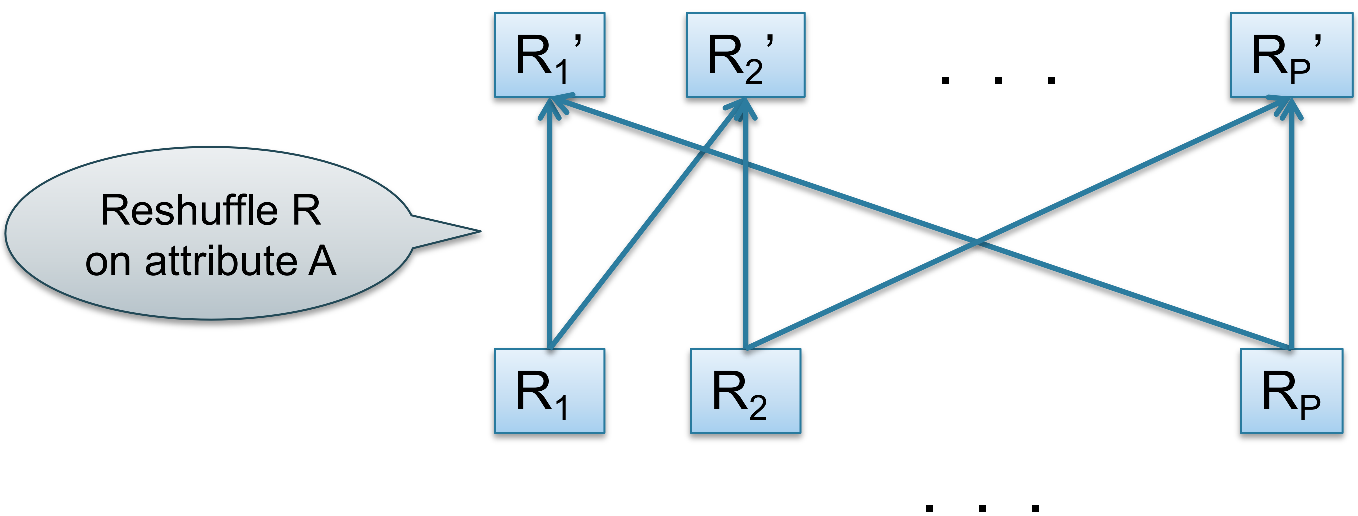


Basic Parallel GroupBy

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

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

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

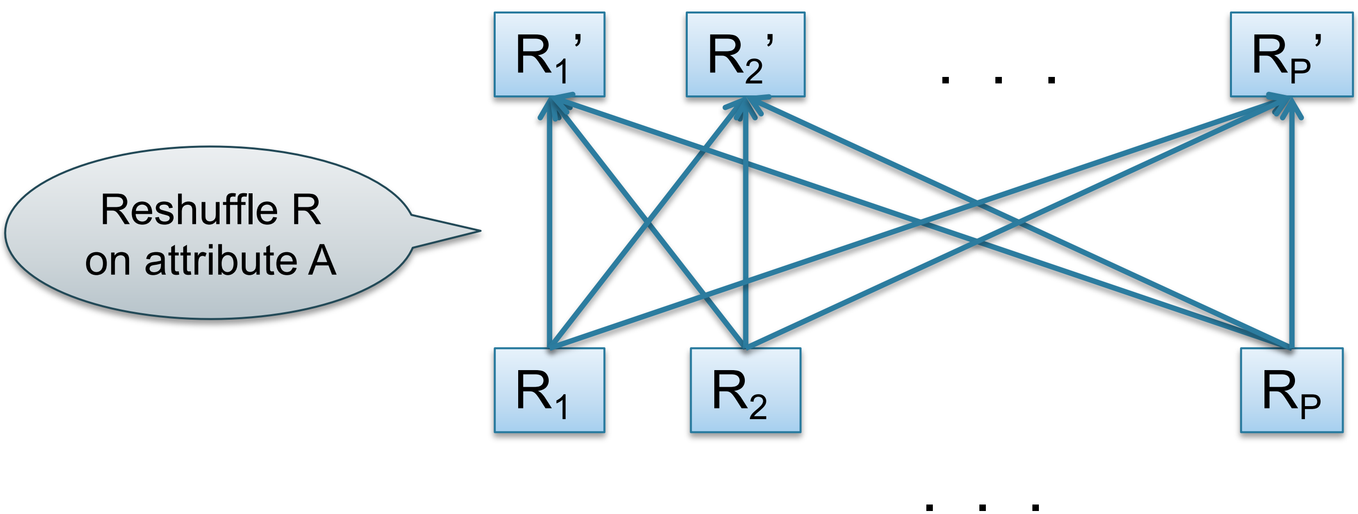


Basic Parallel GroupBy

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

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

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

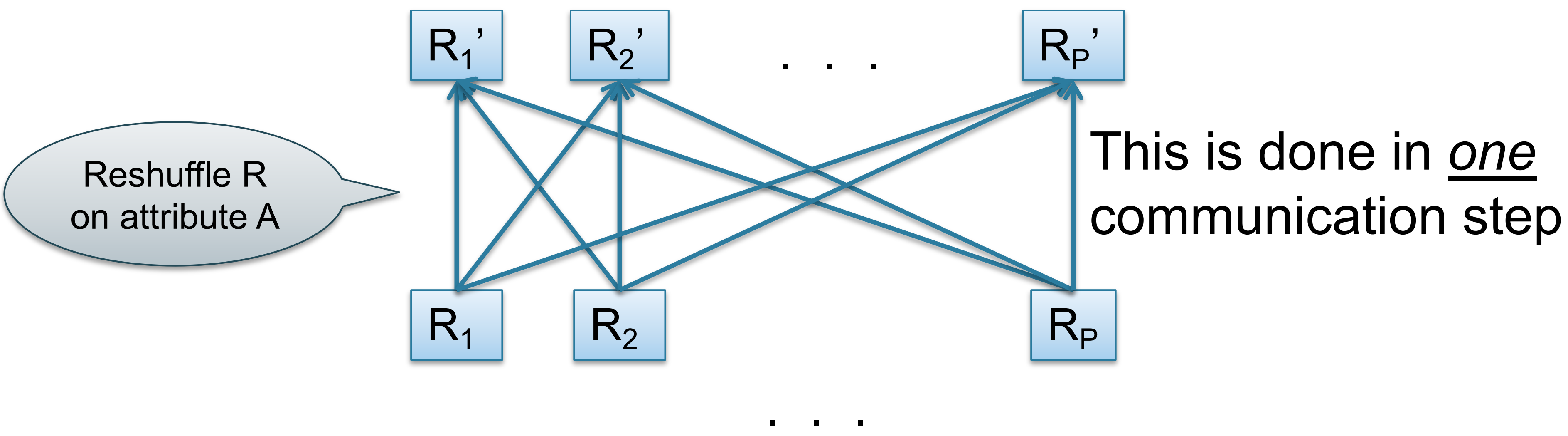


Basic Parallel GroupBy

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

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

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

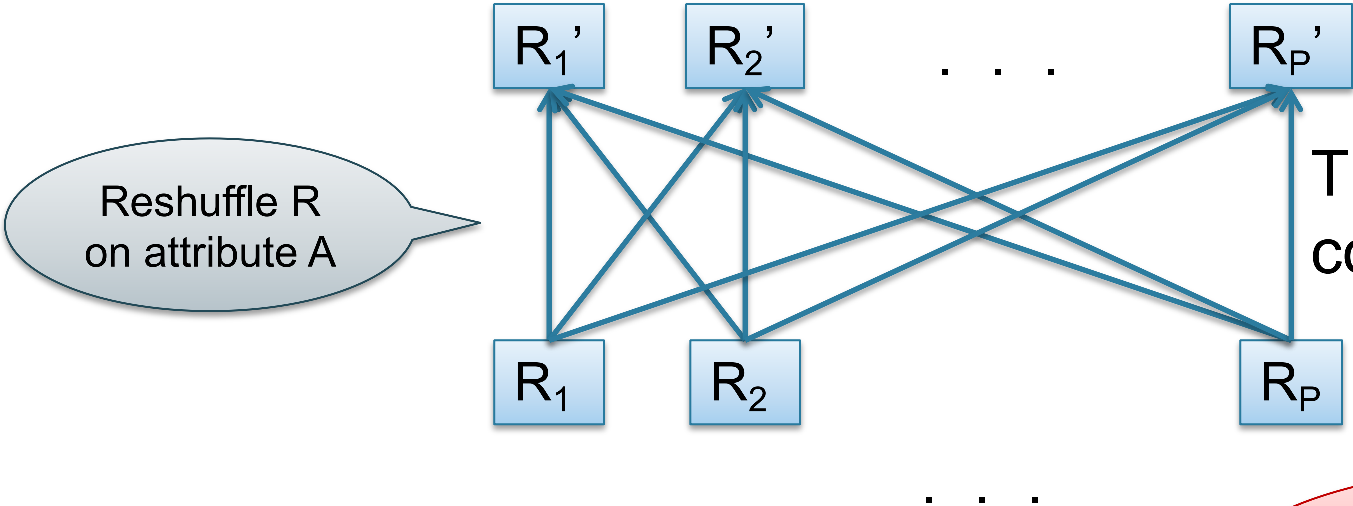


Basic Parallel GroupBy

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

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

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



This is done in *one* communication step

Can you think of an optimization?

Basic Parallel GroupBy

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

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

- Step 0: [Optimization] each server i computes a 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 a 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(<u>K</u>, A, B, C)

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

- Step 0: [Optimization] each server i computes a local group-by:

  $T_i = \gamma_{A,sum(C)}(R_i)$

- Step 1: partition 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

  $R_j' = T_{1,j} \cup \ldots \cup T_{p,j}$

  $Answer_j = \gamma_{A, sum(B)}(R_j')$

Example Query with Group By

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

Machine 1

1/3 of R

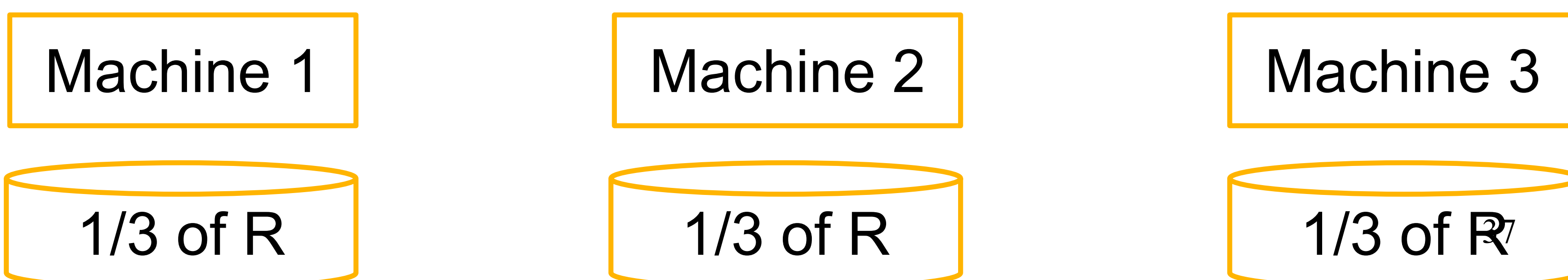
Machine 2

1/3 of R

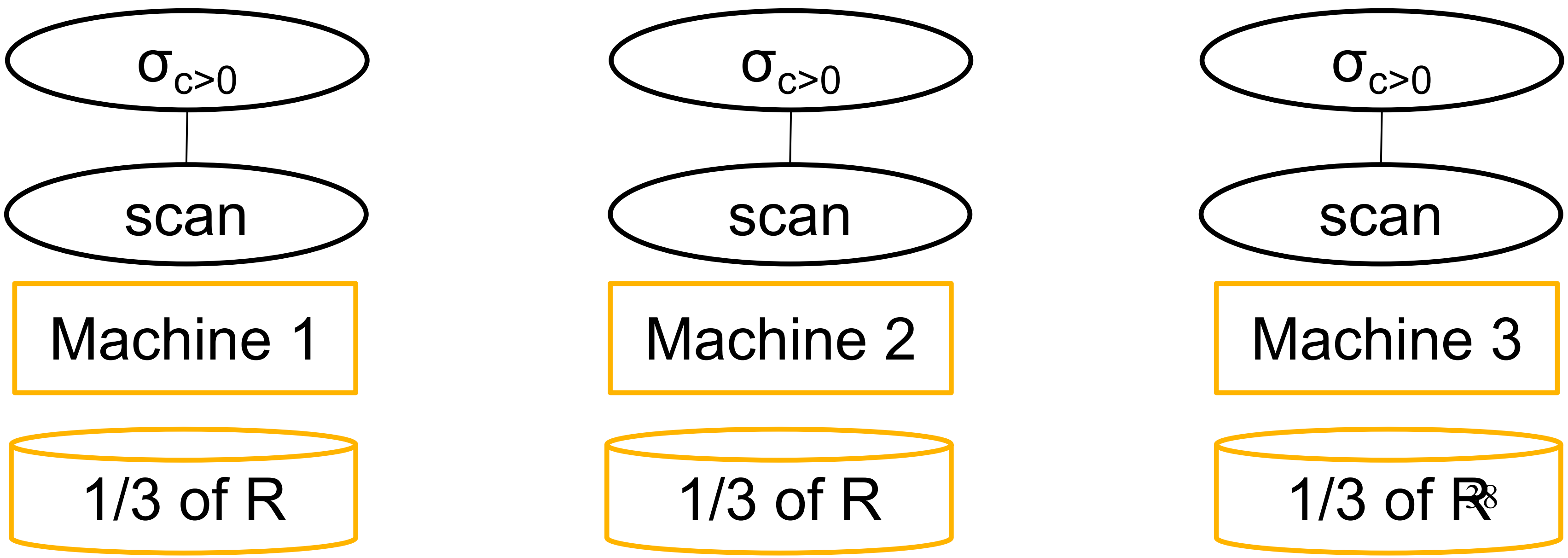
Machine 3

1/3 of R

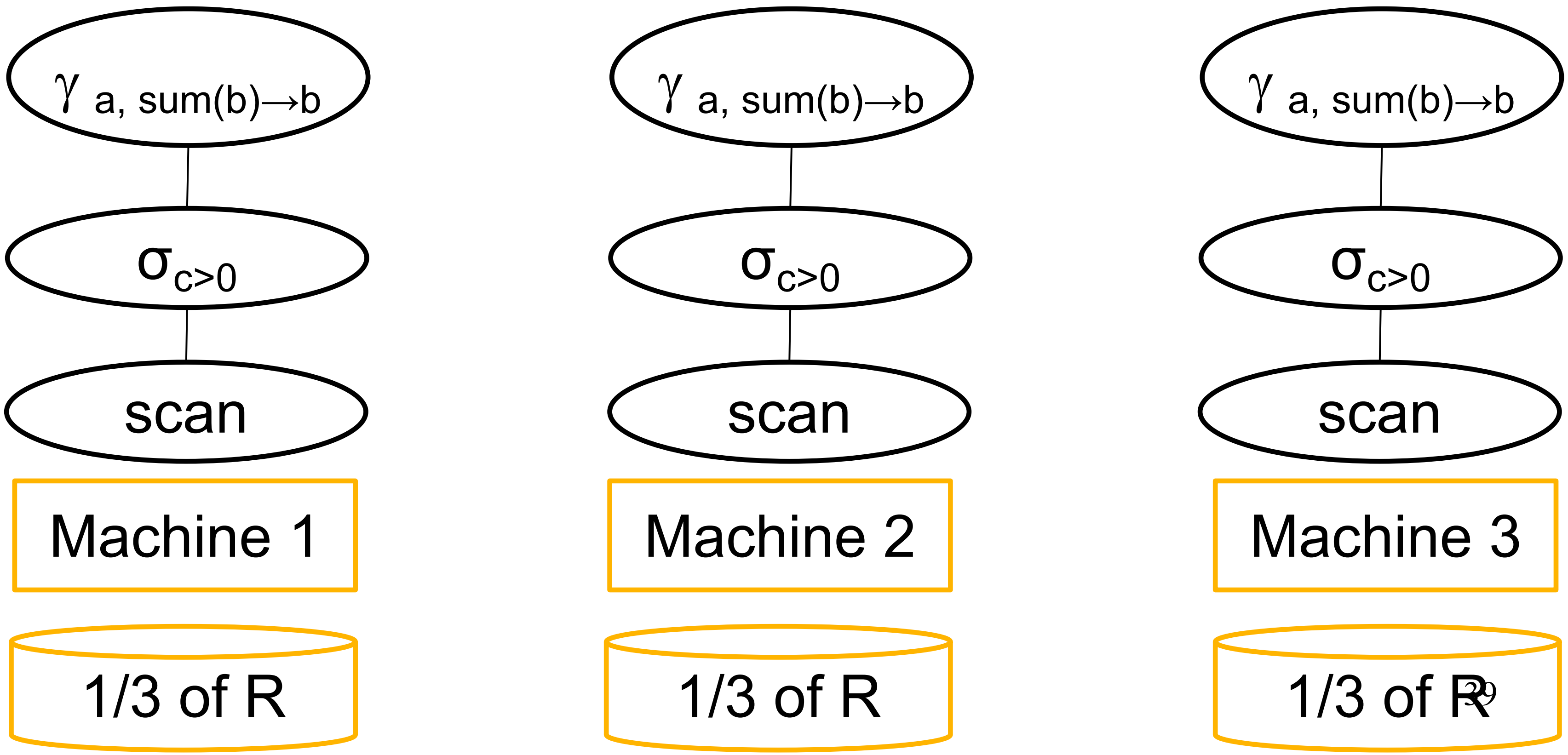
SELECT a, sum(b) as sumb FROM R WHERE c > 0 GROUP BY a



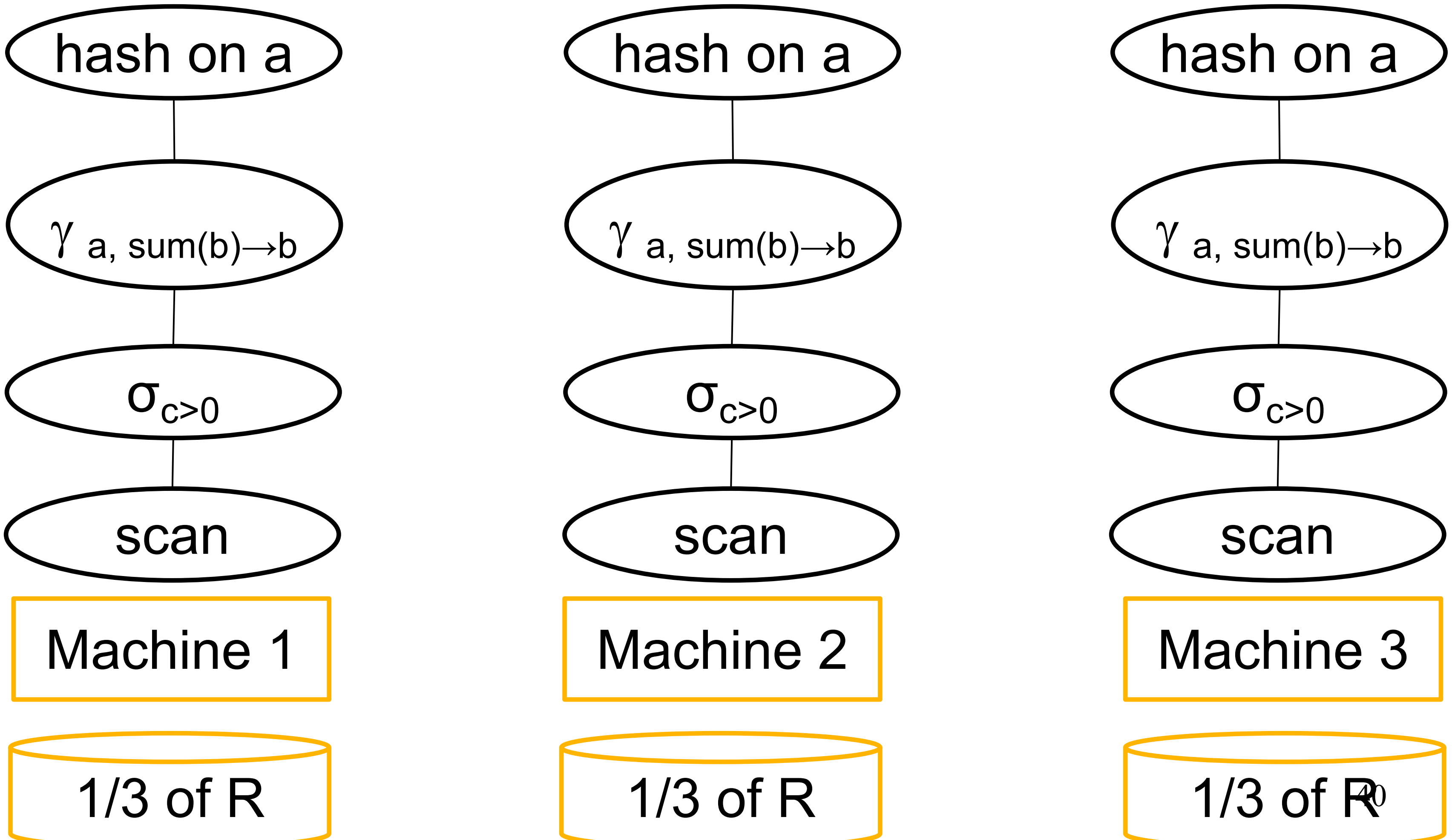
SELECT a, sum(b) as sumb FROM R WHERE c > 0 GROUP BY a



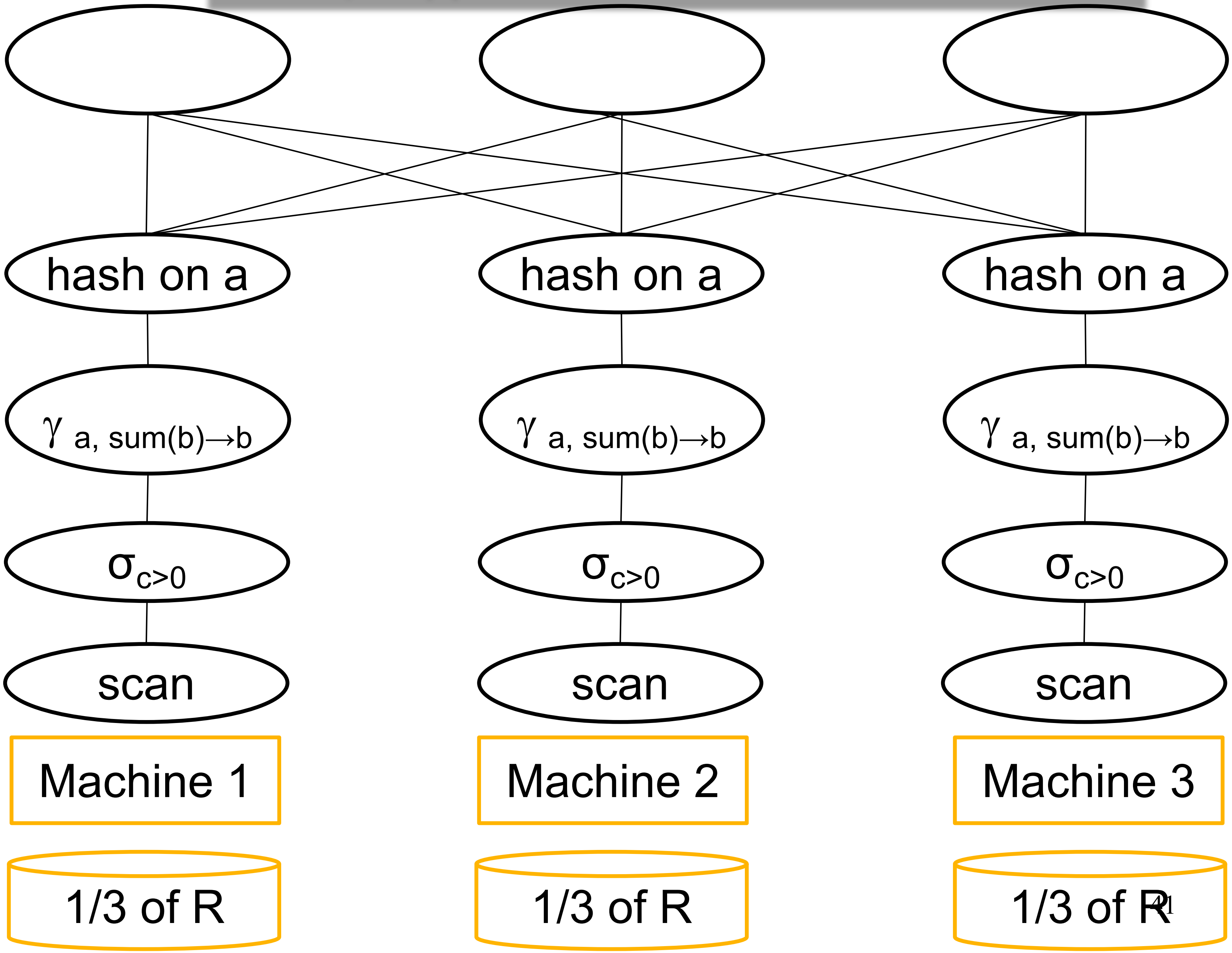
SELECT a, sum(b) as sumb FROM R WHERE c > 0 GROUP BY a



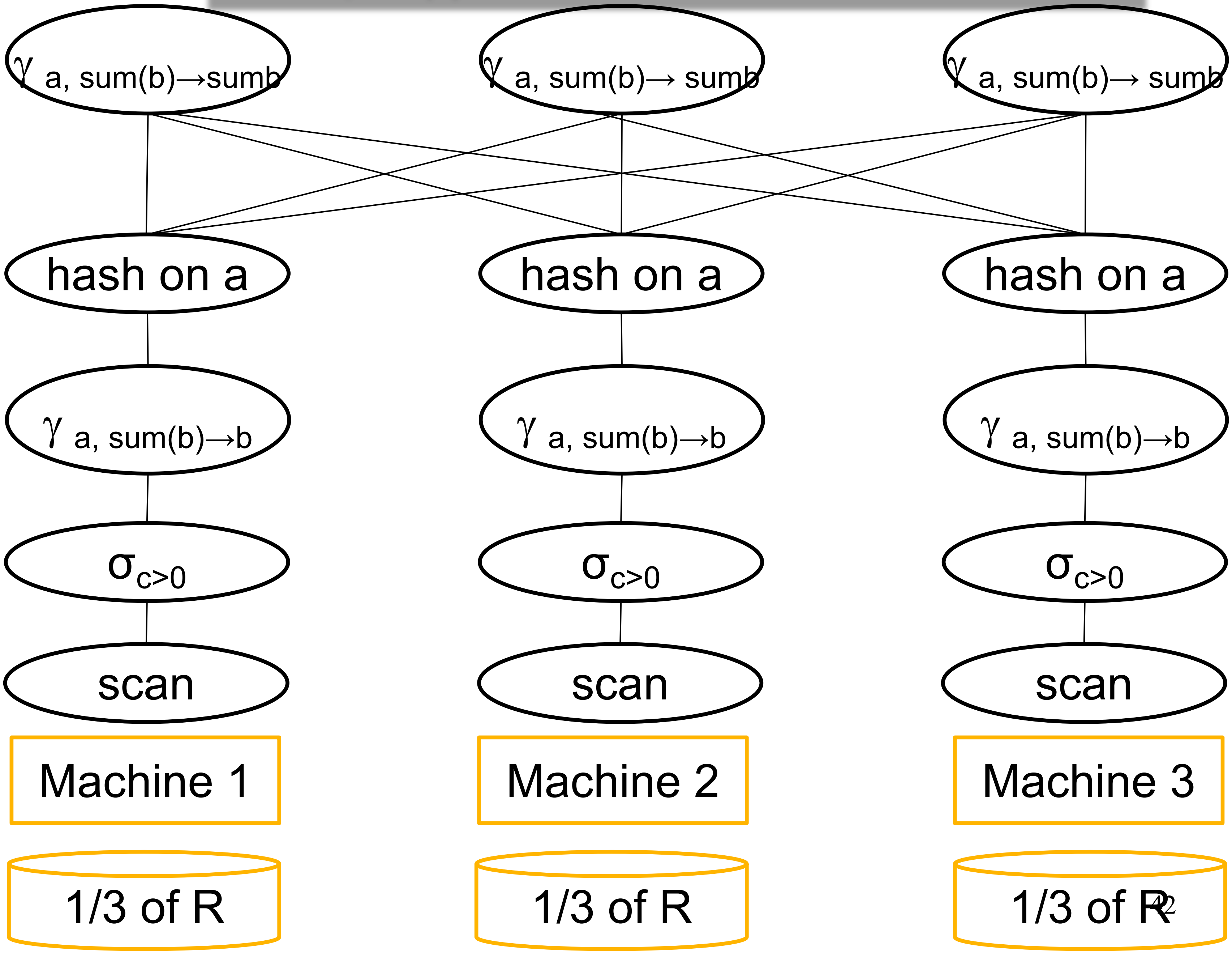
SELECT a, sum(b) as sumb FROM R WHERE c > 0 GROUP BY a



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



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



Basic Parallel GroupBy

Can we apply the local optimization to:

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

Basic Parallel GroupBy

Can we apply the local optimization to:

- 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)

YES: for distributive and algebraic only

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 new 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 new runtime?

- Half (each server holds ½ as many chunks)

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

- Same (the chunk size at each server remains 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 new runtime?

- Half (each server holds ½ as many chunks)

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

- Same (the chunk size at each server remains the same)

But only if data is without skew! – discuss later

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

- Data: R(<u>K1</u>,A, C), S(<u>K2</u>, B, D)
- Query: R(<u>K1</u>,A,C) $\bowtie$ S(<u>K2</u>,B,D)

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

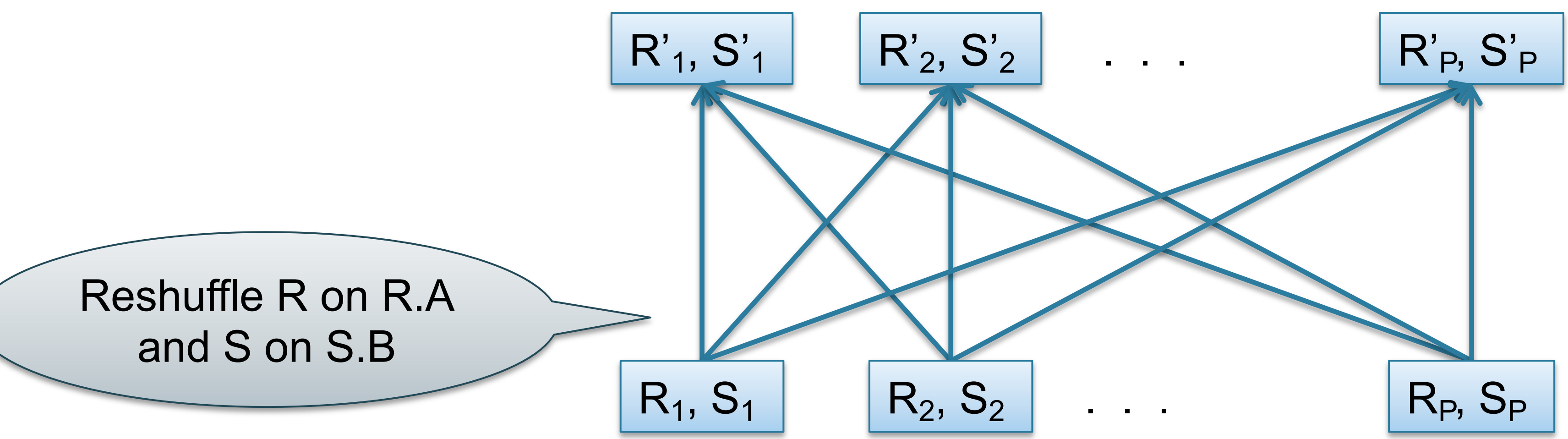
- Data: R(<u>K1</u>,A, C), S(<u>K2</u>, B, D)
- Query: R(<u>K1</u>,A,C) $\bowtie$ S(<u>K2</u>,B,D)

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

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

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

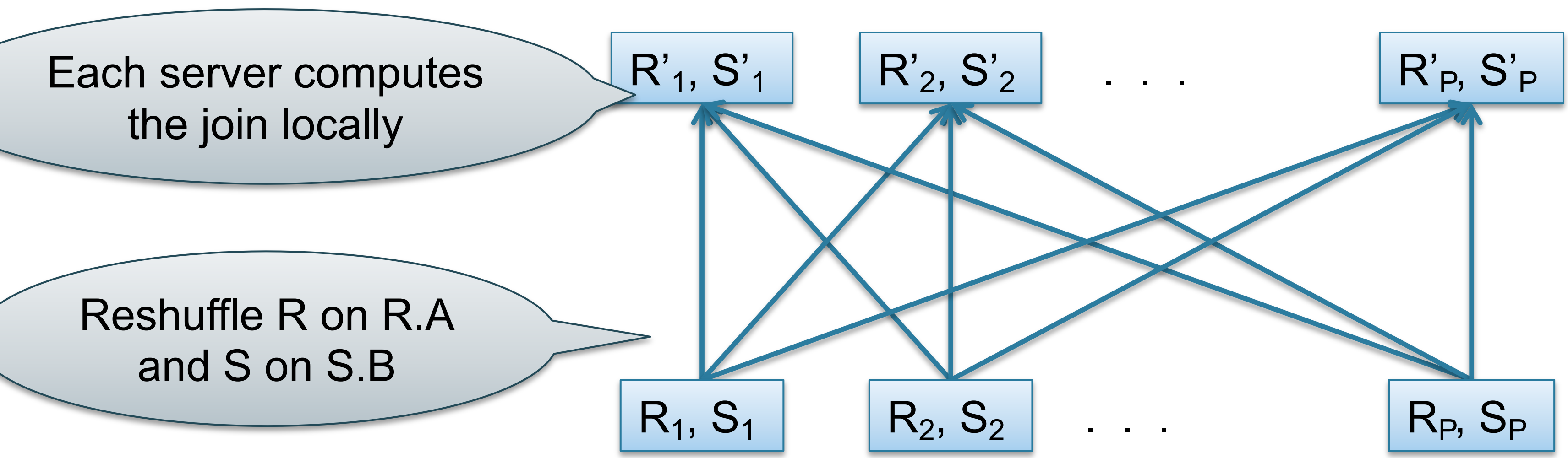
- Data: R($\underline{\text{K1}}$,A, C), S($\underline{\text{K2}}$, B, D)
- Query: R($\underline{\text{K1}}$,A,C) $\bowtie$ S($\underline{\text{K2}}$,B,D)



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

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

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



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

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

Partitioned-Hash-Join:

- Step 1
  - Every server holding a chunk of R reshuffles it using a hash function h(t.A)
  - Every server holding a chunk of S reshuffles it using a hash function h(t.B)
- Step 2:
  - Each server computes a join locally

Optimization for Small Relations

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

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

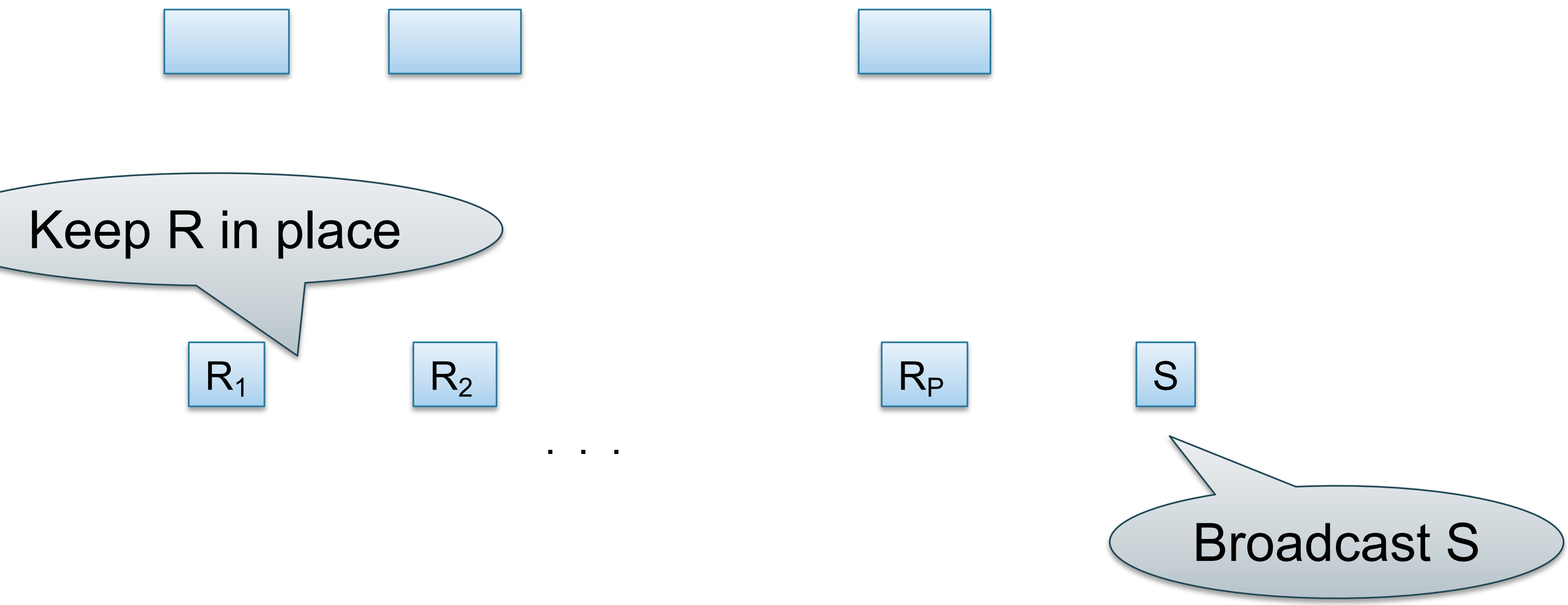
Broadcast Join

R_1 R_2 . . . R_P S

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

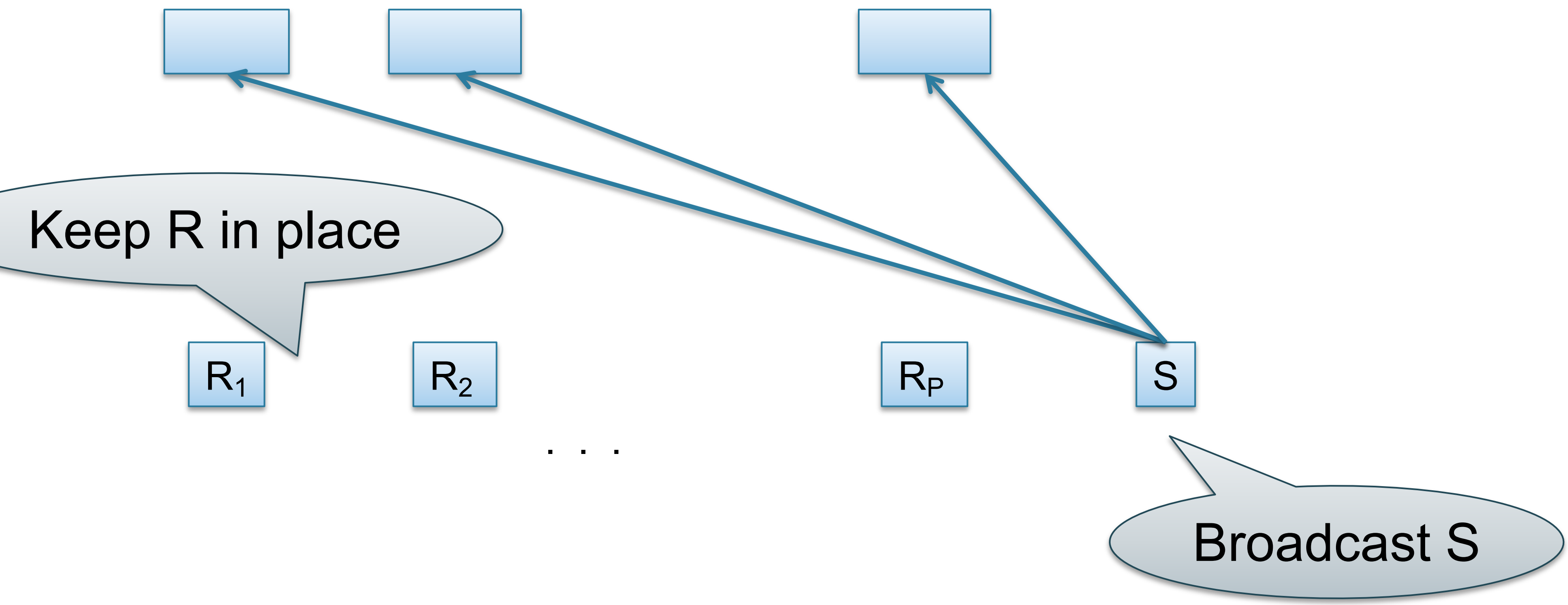
Query: R(A,B) $\bowtie_{B=C}$ S(C,D)

Broadcast Join



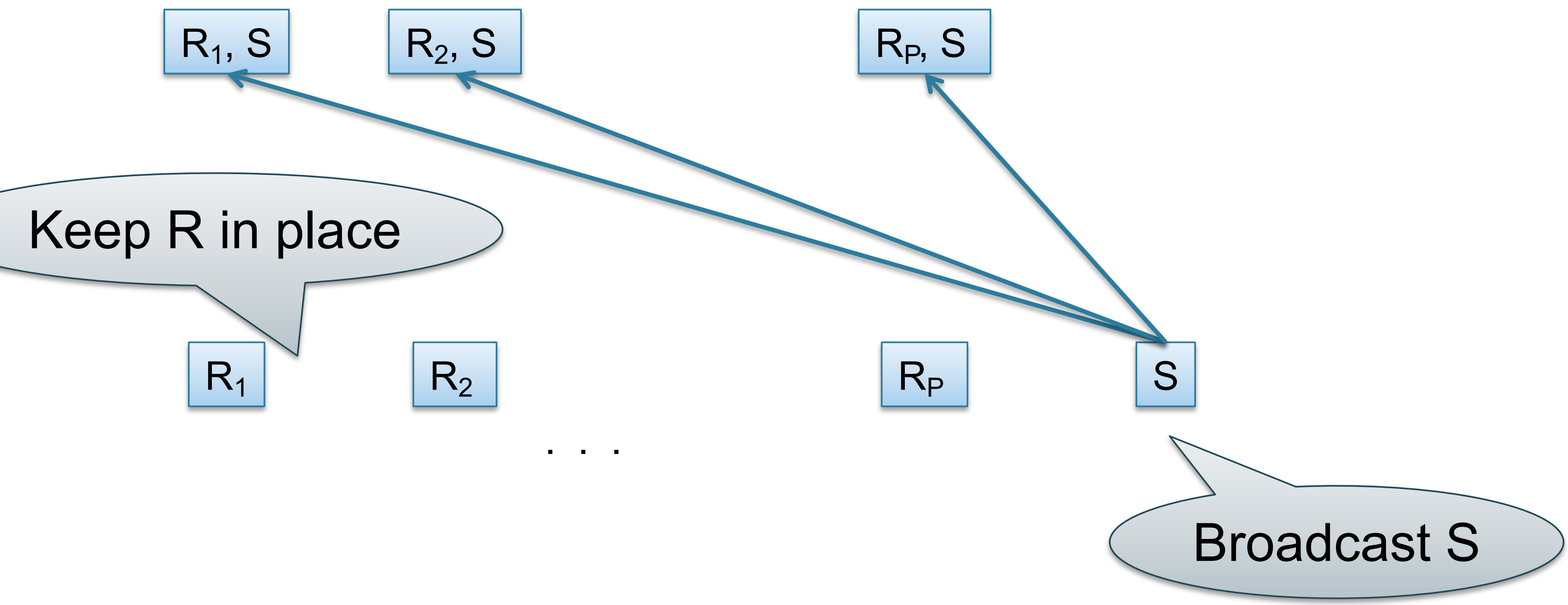
Data: R(A, B), S(C, D)
Query: $R(A,B) \bowtie_{B=C} S(C,D)$

Broadcast Join



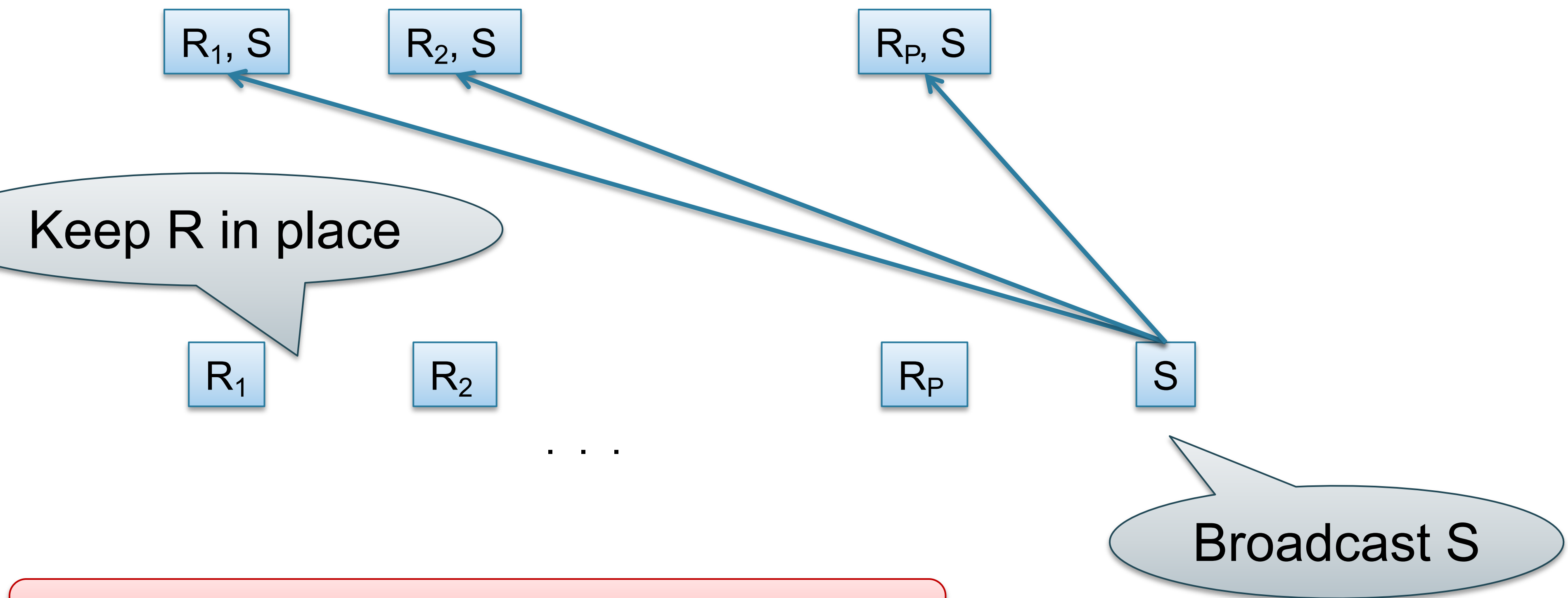
Data: R(A, B), S(C, D)
Query: R(A,B) $\bowtie_{B=C}$ S(C,D)

Broadcast 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?

Skew Join

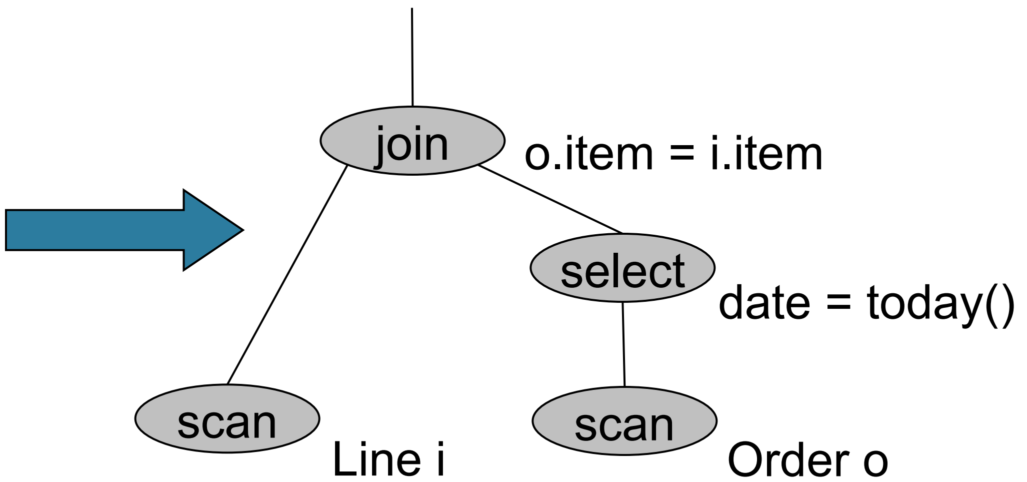
$R(A,B) \bowtie_{B=C} S(C,D)$

- Problem: skewed values C in S
- Preprocessing: identify the heavy hitter values C (i.e. occur > threshold times)
- Partition S into S^{light} and S^{heavy}
- Use partition hash-join for $R \bowtie S^{light}$
- Use broadcast join for $R \bowtie S^{heavy}$

Example Query Execution

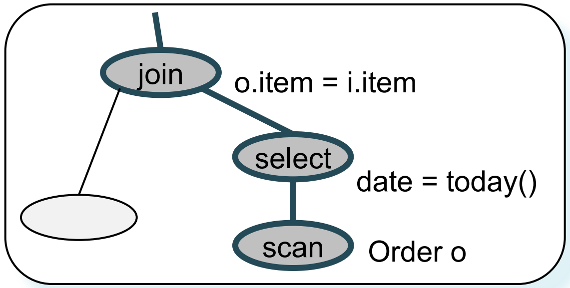
Find all orders from today, along with the items ordered

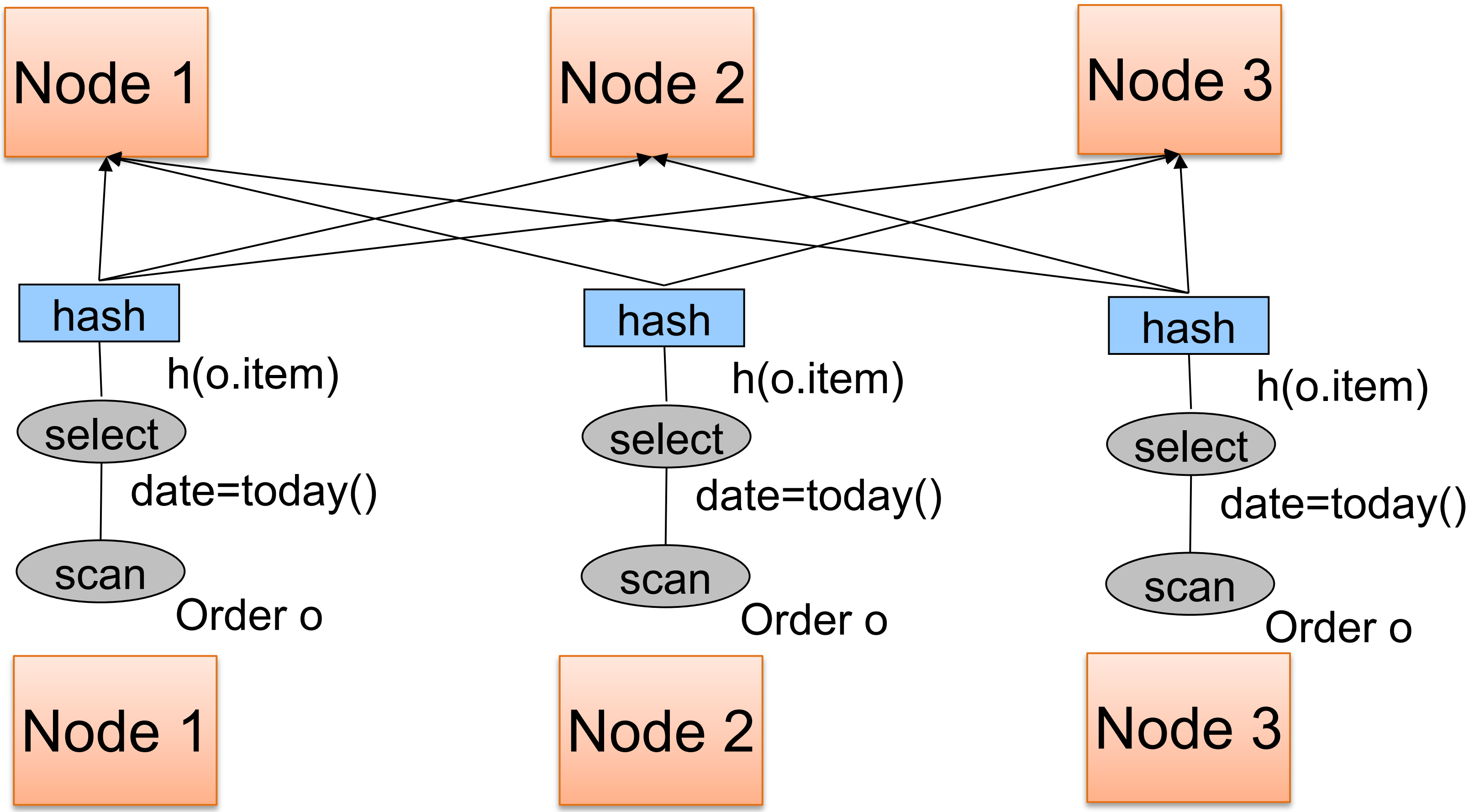
```
SELECT *
  FROM Order o, Line i
 WHERE o.item = i.item
   AND o.date = today()
```



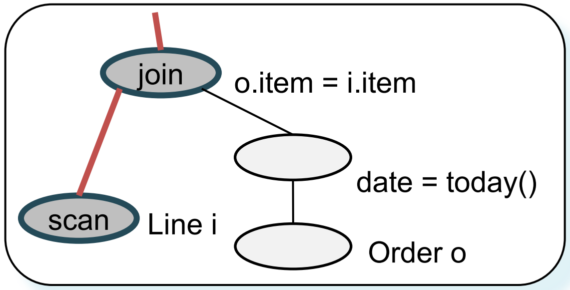
Order(oid, item, date), Line(item, …)

Query Execution

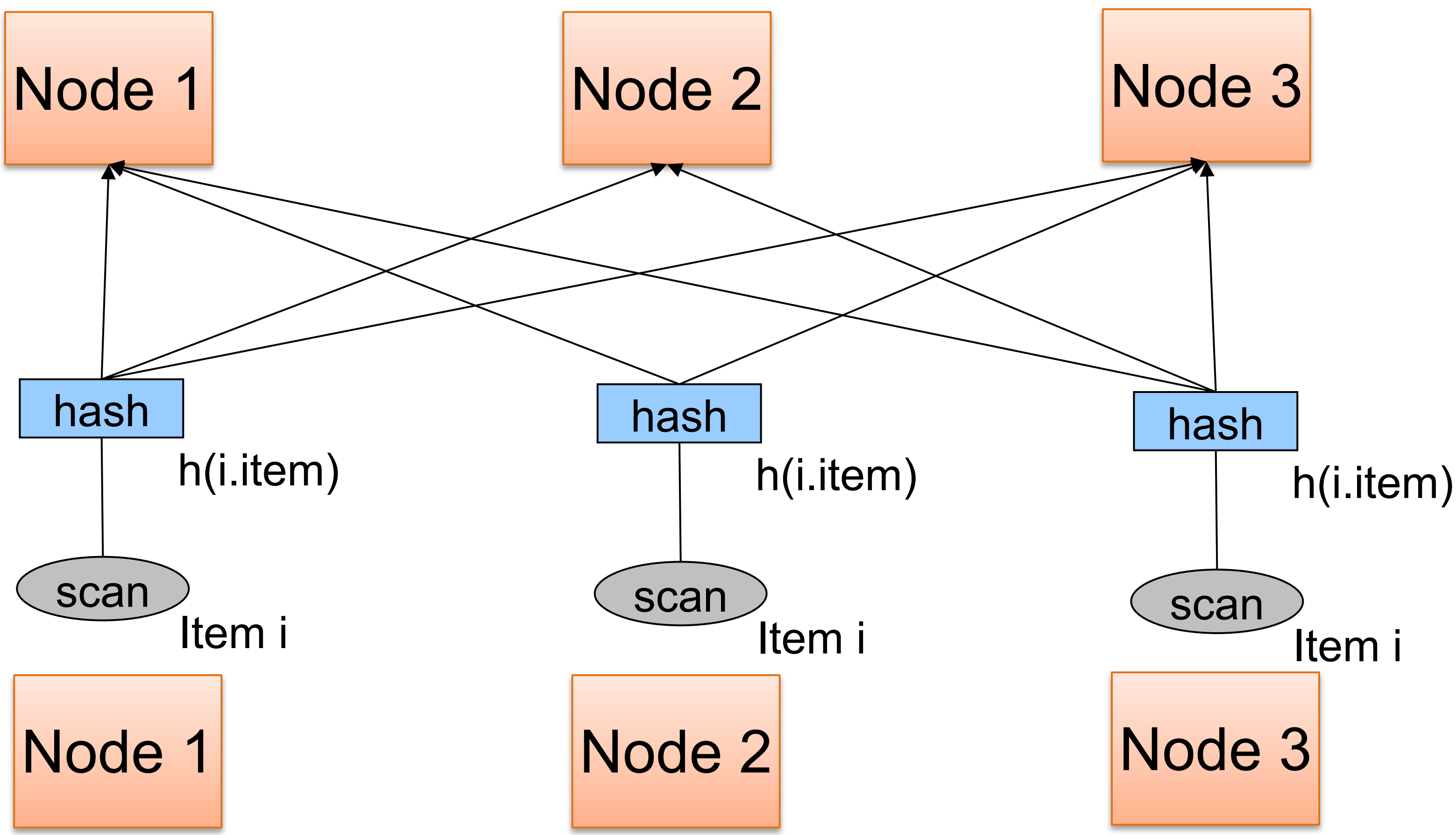




Order(<u>oid</u>, item, date), Line(item, …)

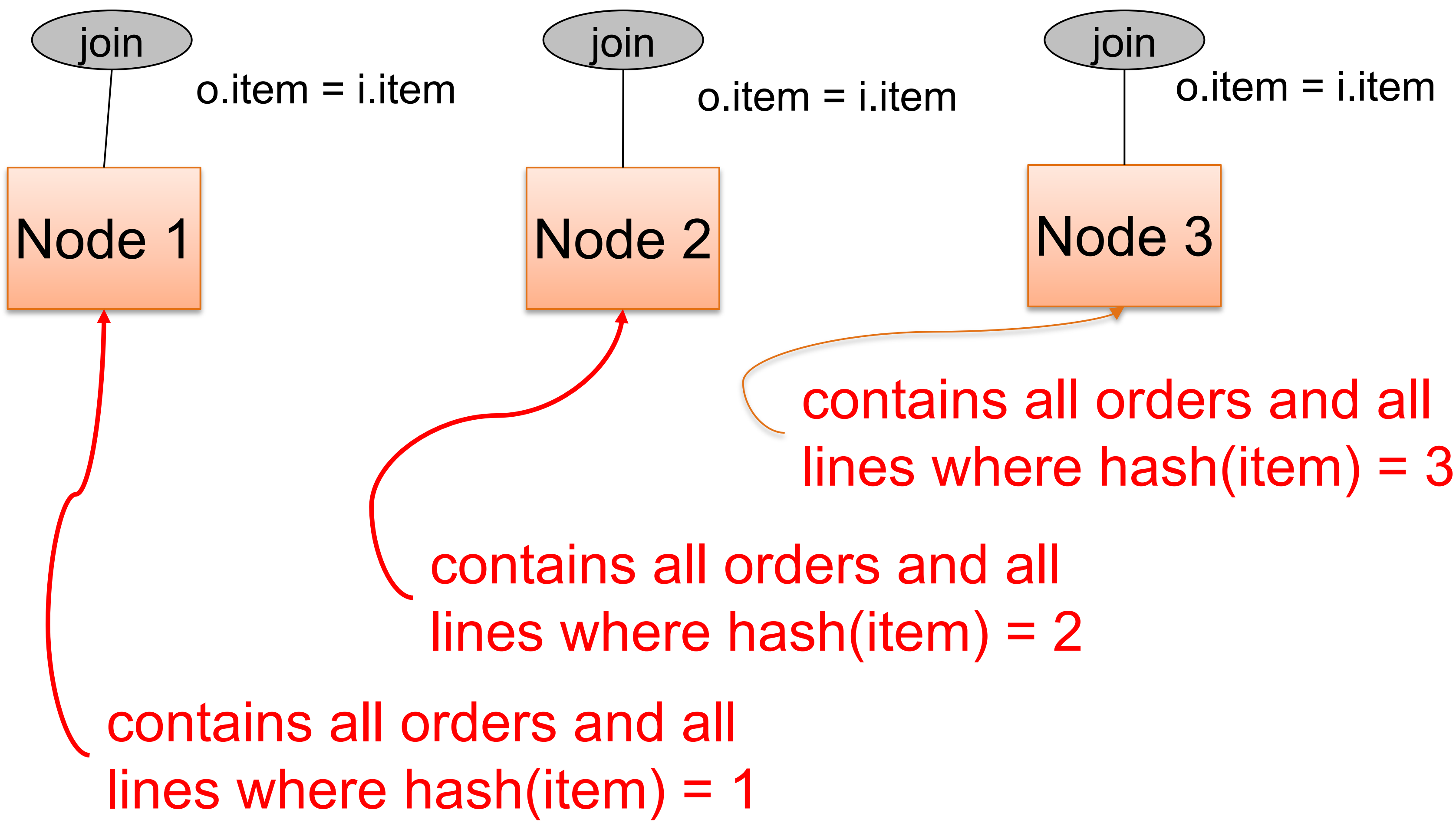


Query Execution



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

Query Execution

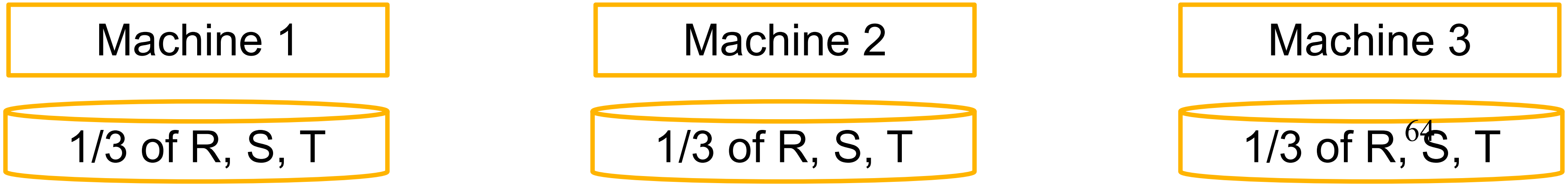


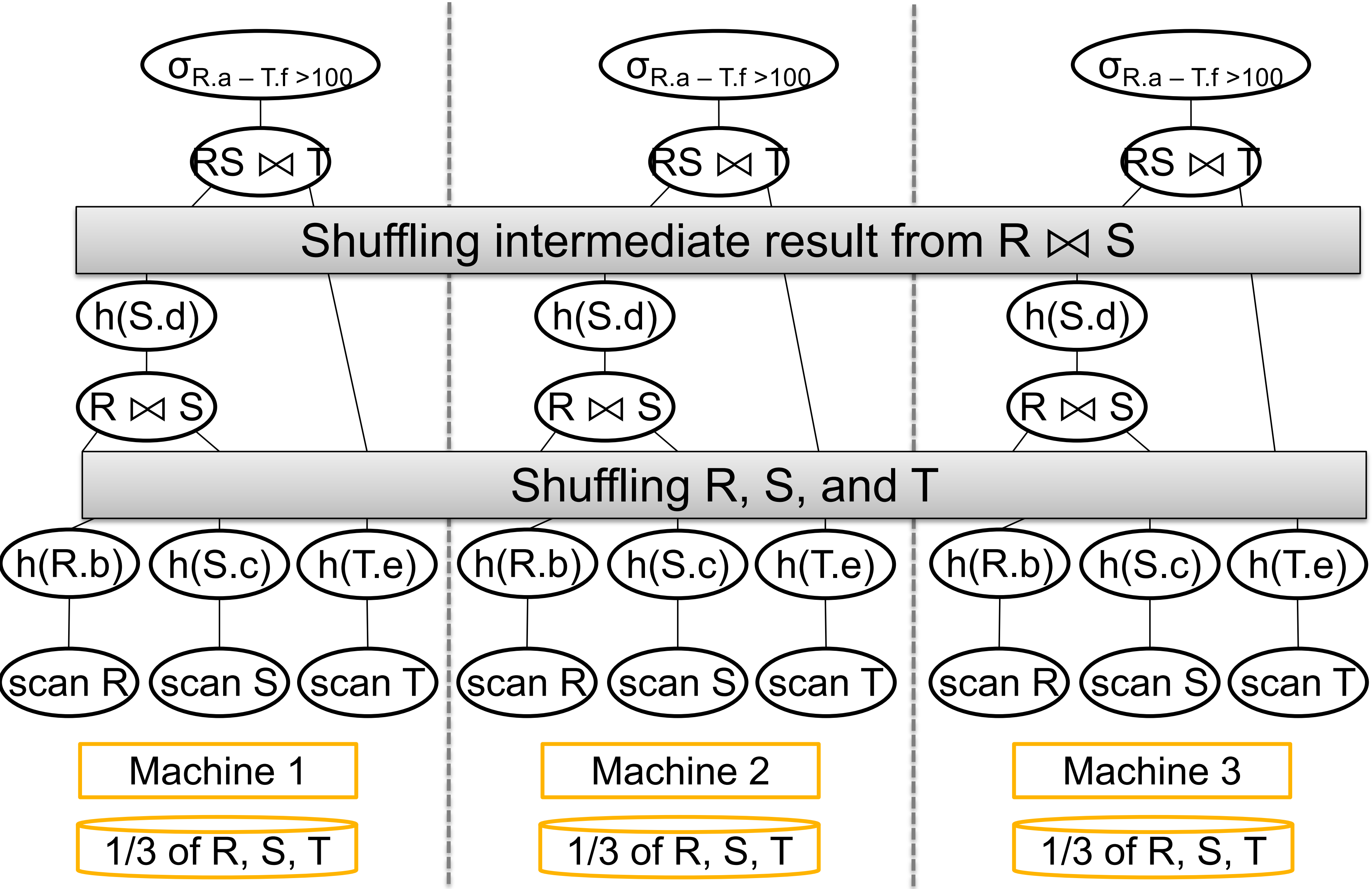
Example 2

SELECT *

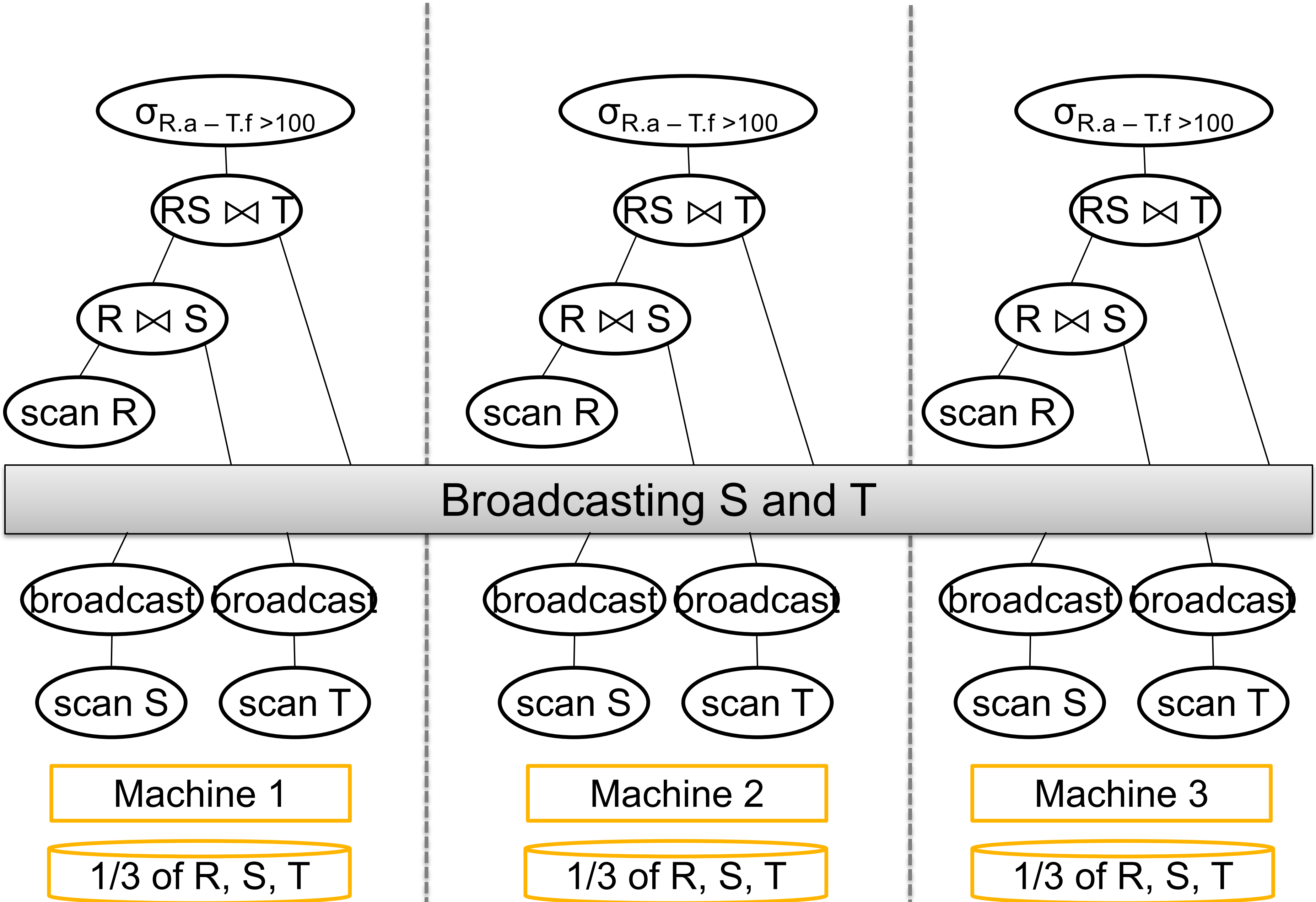
FROM R, S, T

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





$\sigma_{R.a - T.f > 100}$
RS ⋈ T
Shuffling intermediate result from R ⋈ S
h(S.d)
R ⋈ S
Shuffling R, S, and T
h(R.b)
h(S.c)
h(T.e)
scan R
scan S
scan T
Machine 1
Machine 2
Machine 3
1/3 of R, S, T


σR.a – T.f >100
RS ⋈ T
R ⋈ S
scan R
σR.a – T.f >100
RS ⋈ T
R ⋈ S
scan R
σR.a – T.f >100
RS ⋈ T
R ⋈ S
scan R
Broadcasting S and T
broadcast
broadcast
scan S
scan T
broadcast
broadcast
scan S
scan T
broadcast
broadcast
scan S
scan T
Machine 1
Machine 2
Machine 3
1/3 of R, S, T
1/3 of R, S, T
1/3 of R, S, T