CSE 444: Database Internals

Section 10:

Parallel Processing, Distributed Processing, and Replication

Review in this section

- Parallel DBMS
- 2-Phase Commit

Parallel DBMS

R(a,b) is horizontally partitioned across N = 3 machines.

Each machine locally stores approximately 1/N of the tuples in R.

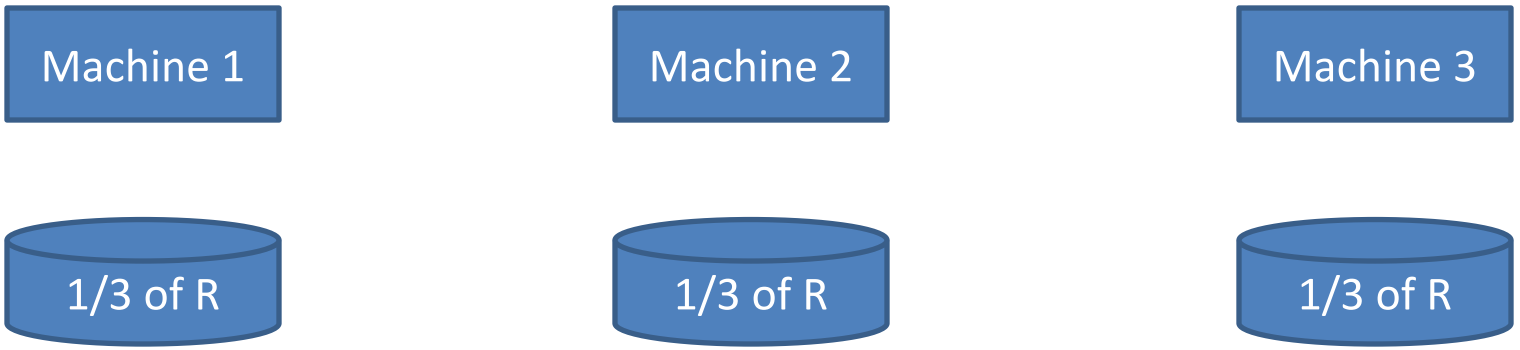
The tuples are randomly organized across machines (i.e., R is block partitioned across machines).

Show a RA plan for this query and how it will be executed across the N = 3 machines.
Pick an efficient plan that leverages the parallelism as much as possible.

SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a

R(a, b)

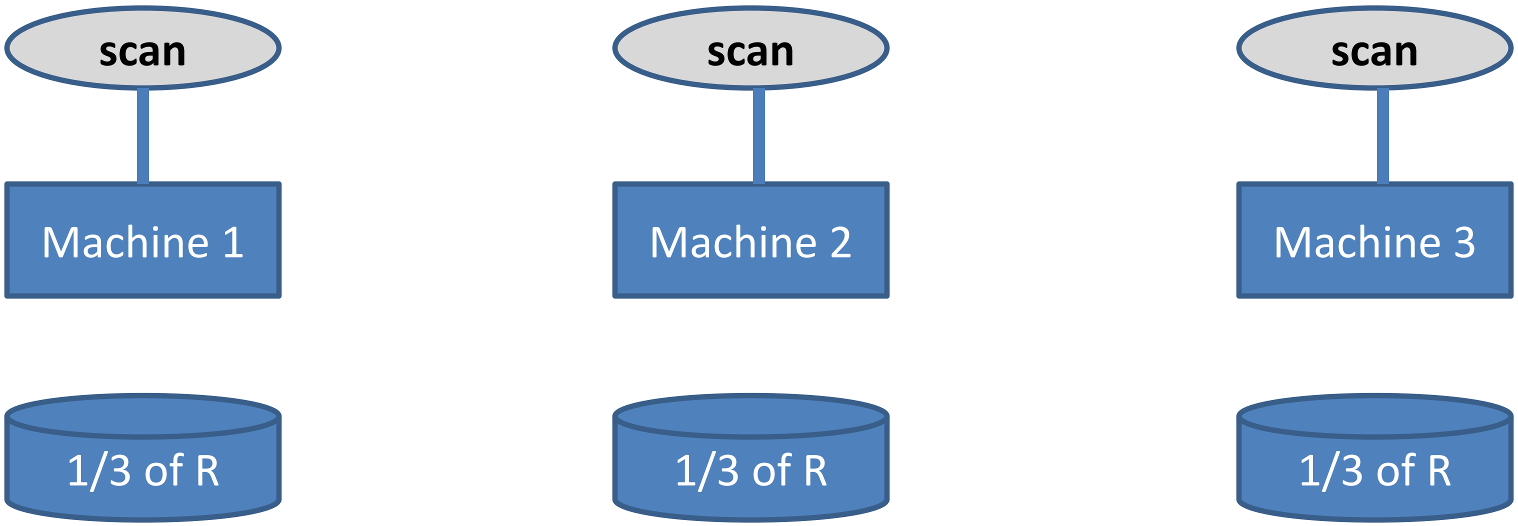
```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```



R(a, b)

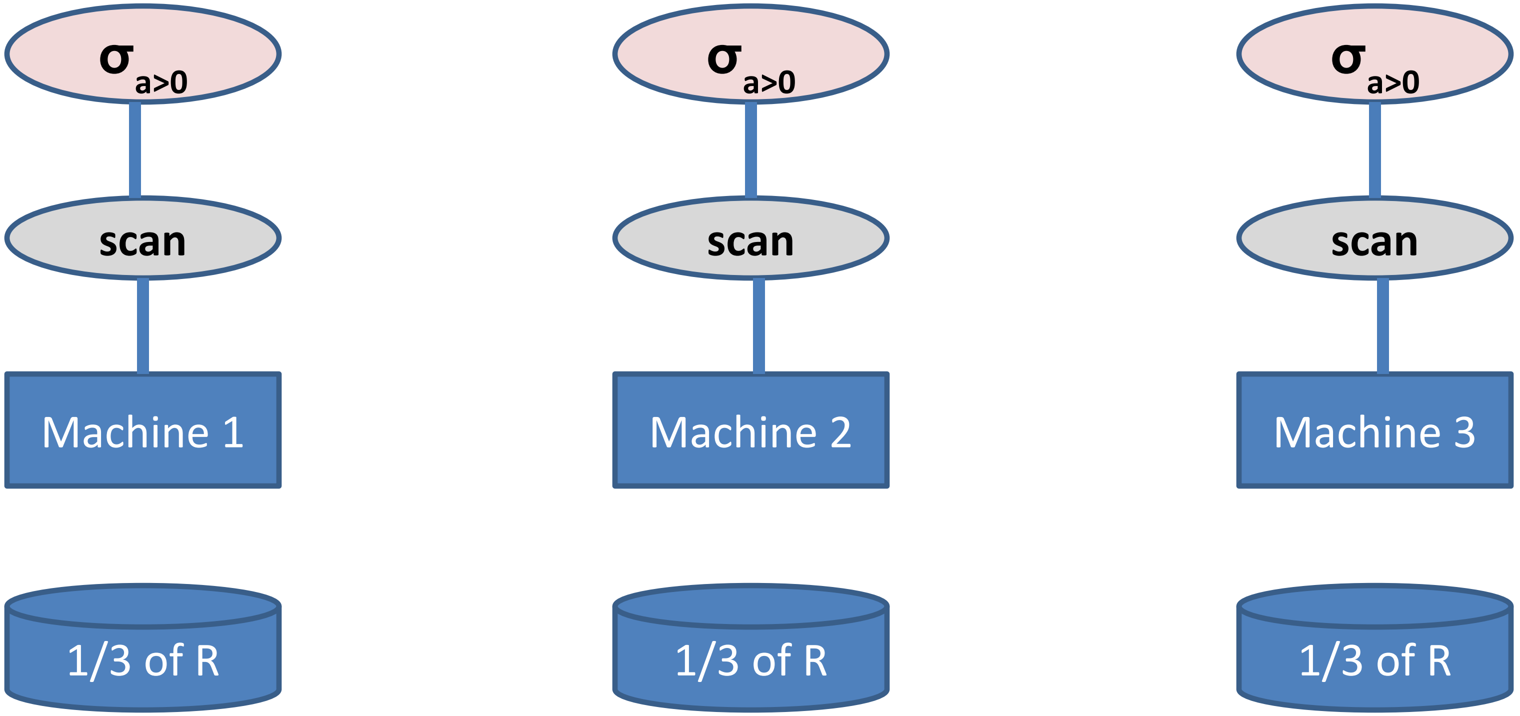
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a

If more than one relation on a machine, then “scan S”, “scan R” etc



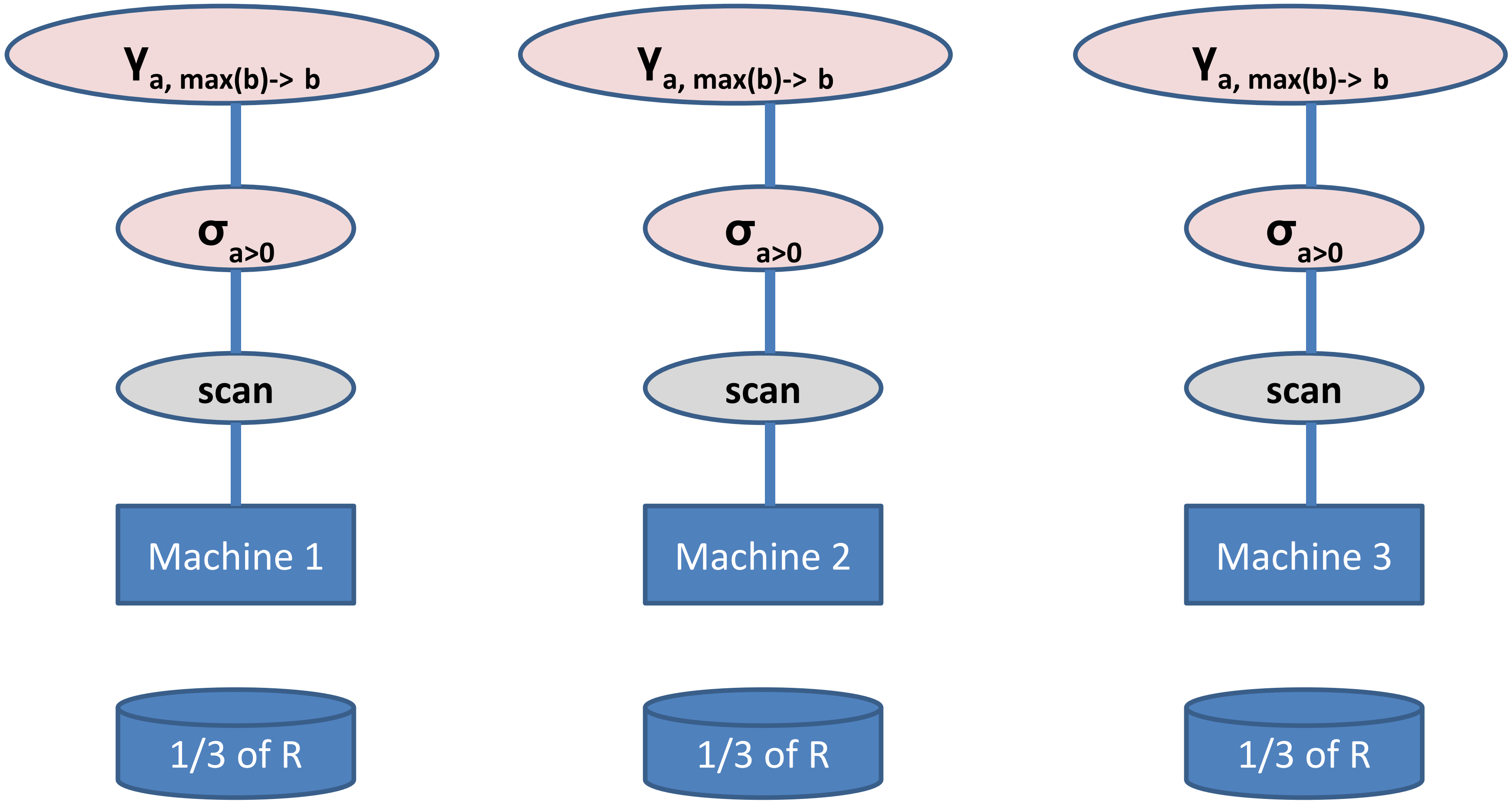
R(a, b)

```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```



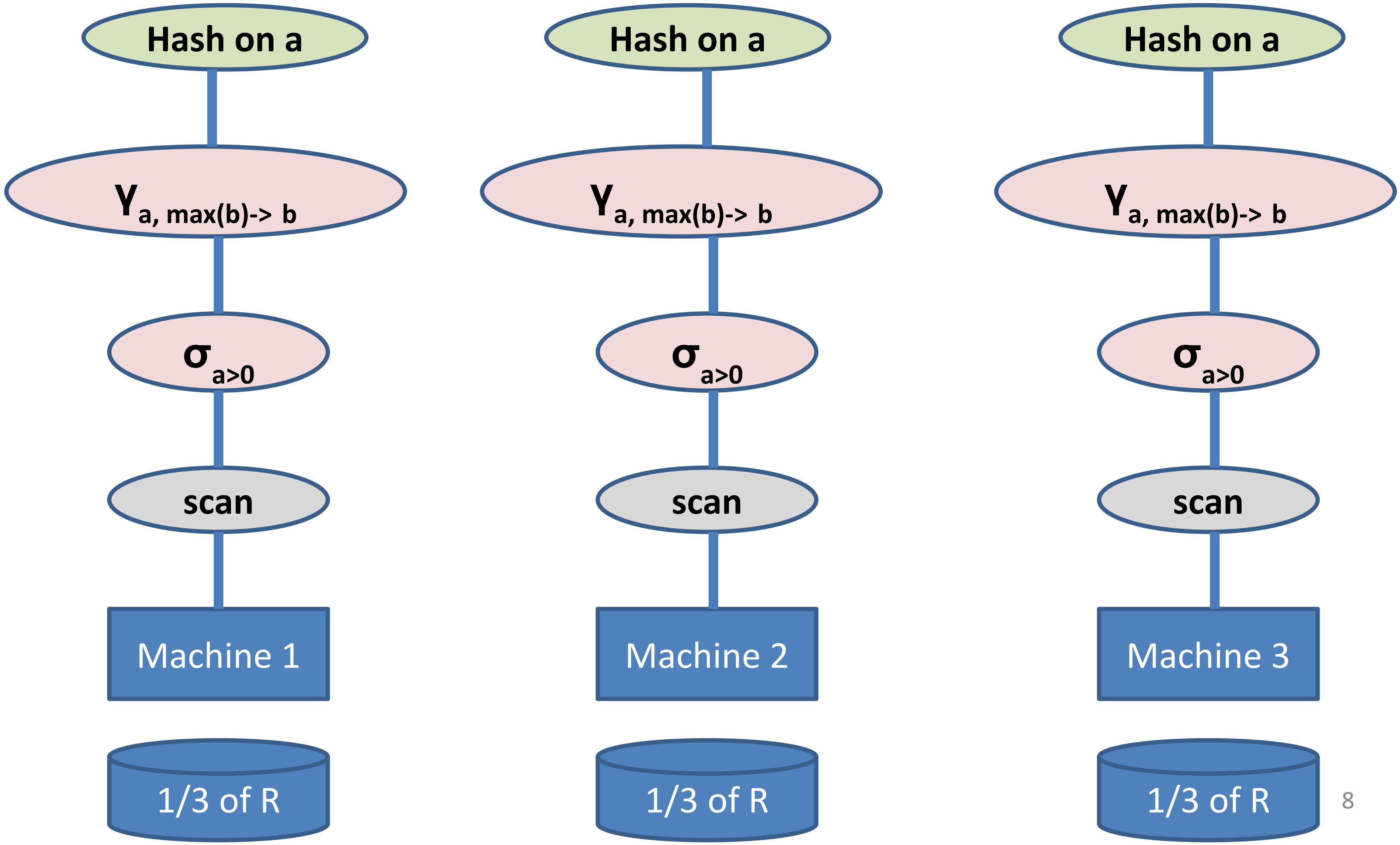
R(a, b)

SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a



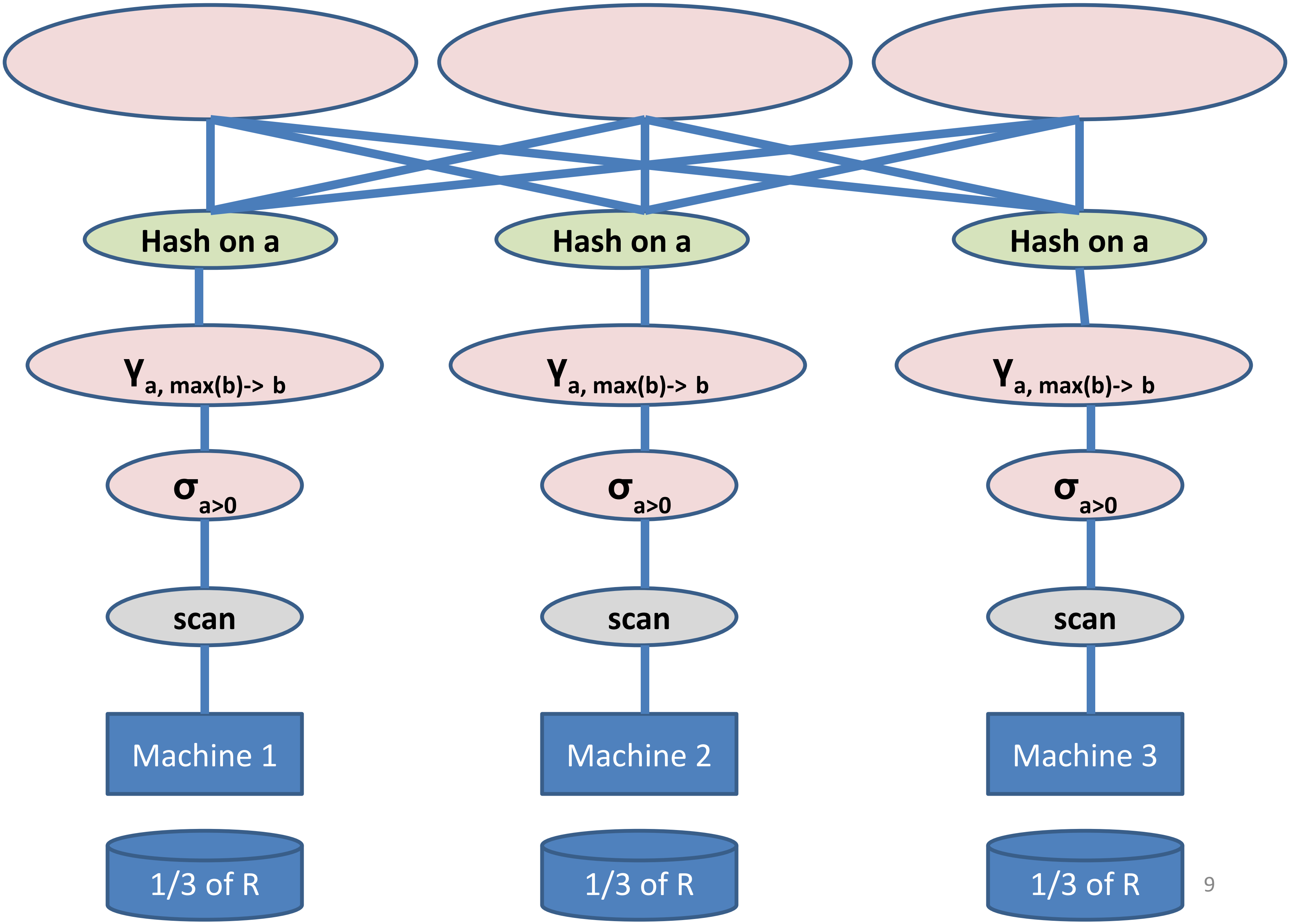
R(a, b)

SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a



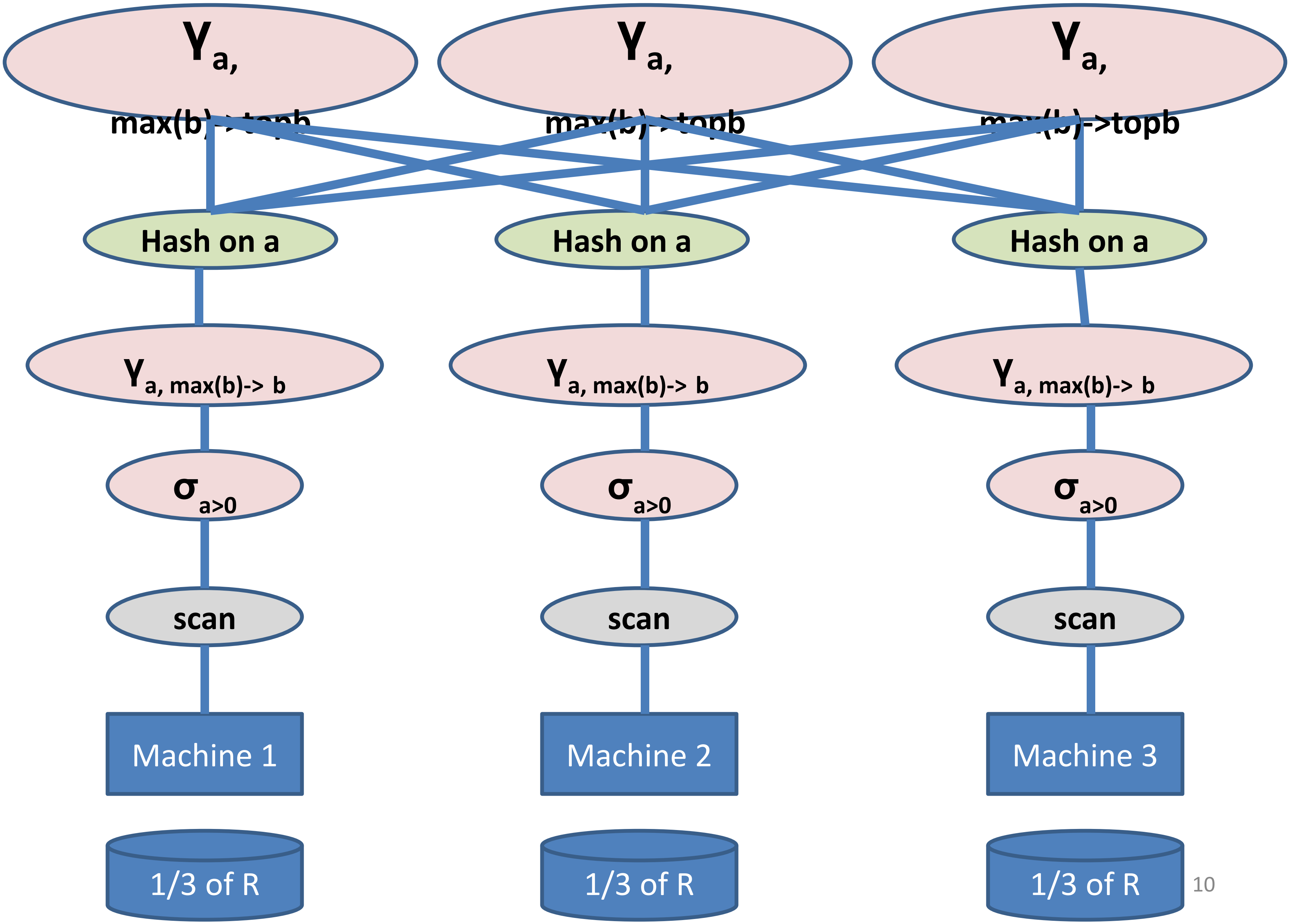
R(a, b)

SELECT a, max(b) as topb FROM R
WHERE a > 0 GROUP BY a



R(a, b)

SELECT a, max(b) as topb FROM R
WHERE a > 0 GROUP BY a



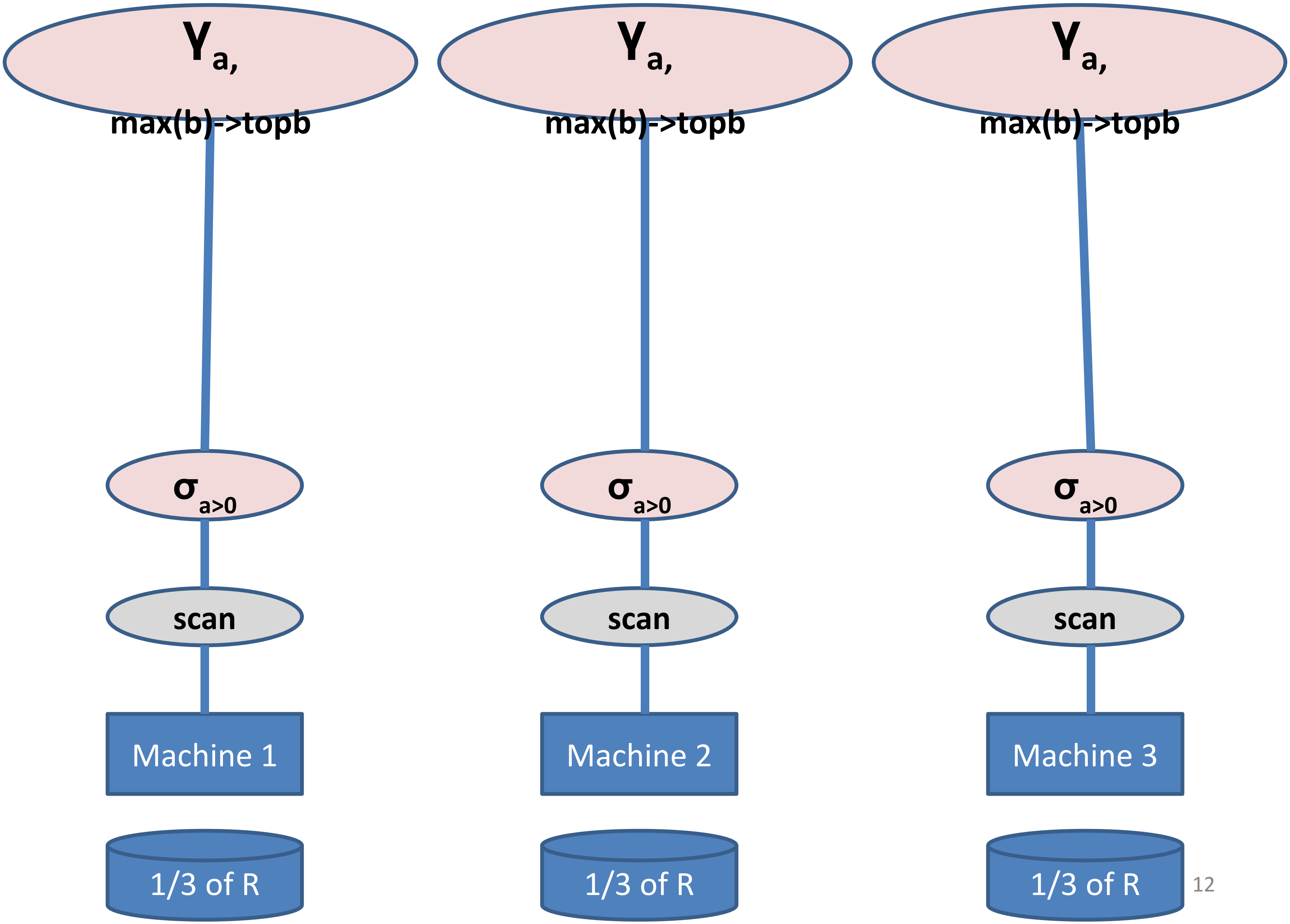
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a

Benefit of hash-partitioning

- **For parallel DBMS**
  - It would avoid the data re-shuffling phase
  - It would compute the aggregates locally

Hash-partition on a for R(a, b)

SELECT a, max(b) as topb FROM R
WHERE a > 0 GROUP BY a



Problem 1)

Consider relations R(a,b), S(c,d), and T(e,f). All three are horizontally partitioned across N = 3 machines.

The tuples are randomly organized across machines.

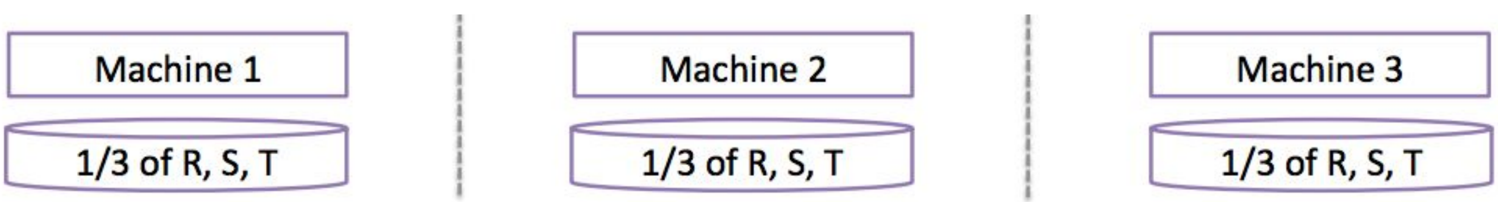
Show a relational algebra plan for the following query and how it will be executed across the N = 3 machines:

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

Problem 1)

R(a,b)
S(c,d)
T(e,f)

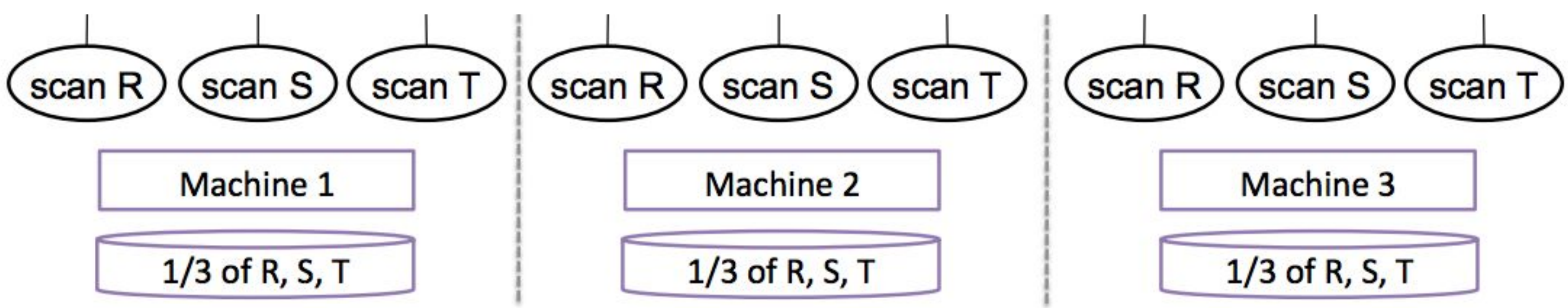
```
SELECT *
FROM R, S, T
WHERE R.b = S.c
      AND S.d = T.e
      AND (R.a - T.f) > 100
```



Problem 1)

R(a,b)
S(c,d)
T(e,f)

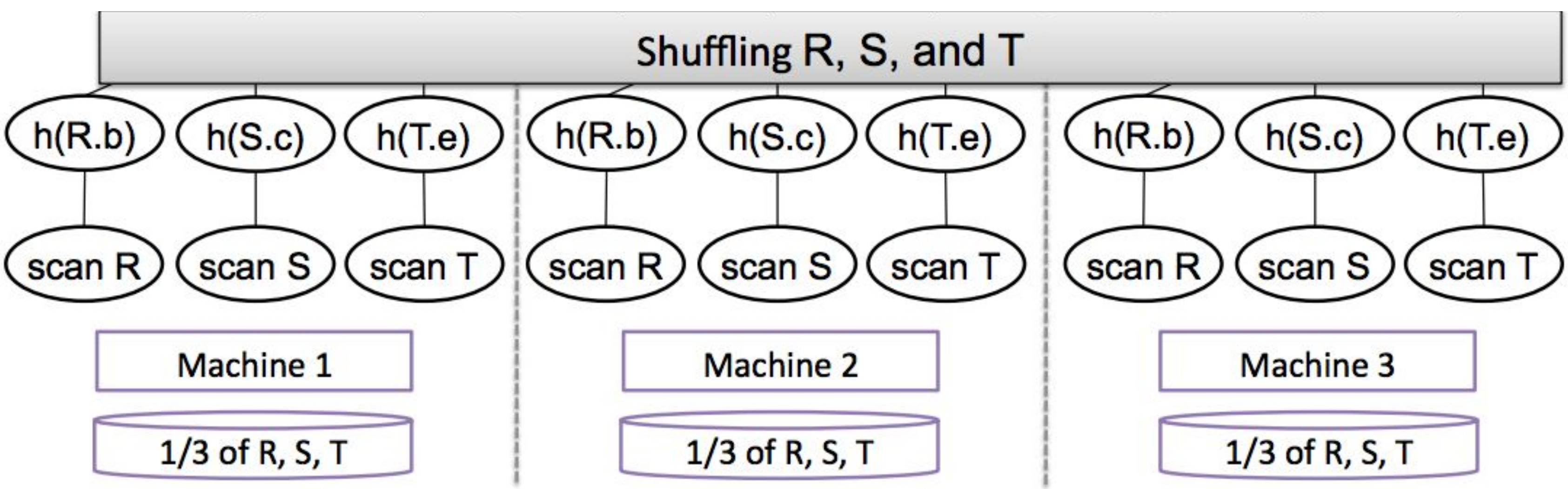
```
SELECT *
FROM R, S, T
WHERE R.b = S.c
      AND S.d = T.e
      AND (R.a - T.f) > 100
```



Problem 1)

R(a,b)
S(c,d)
T(e,f)

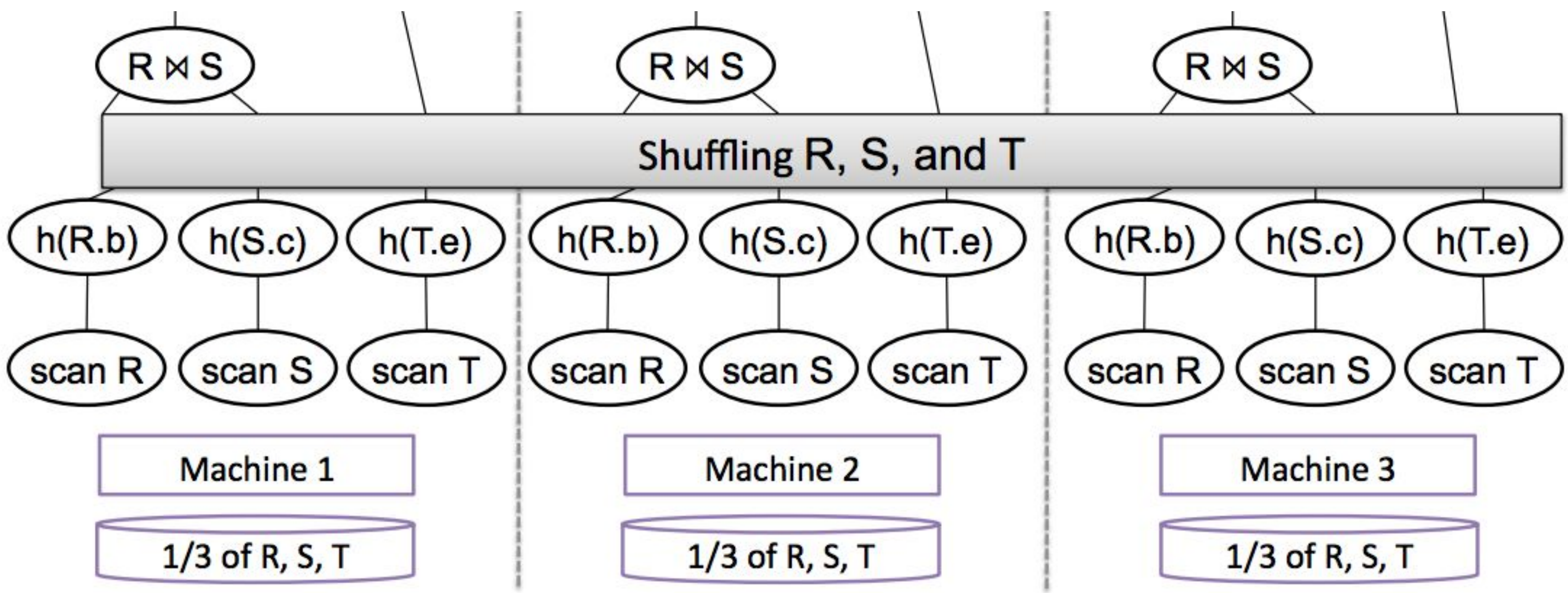
```
SELECT *
FROM R, S, T
WHERE R.b = S.c
      AND S.d = T.e
      AND (R.a - T.f) > 100
```



Problem 1)

R(a,b)
S(c,d)
T(e,f)

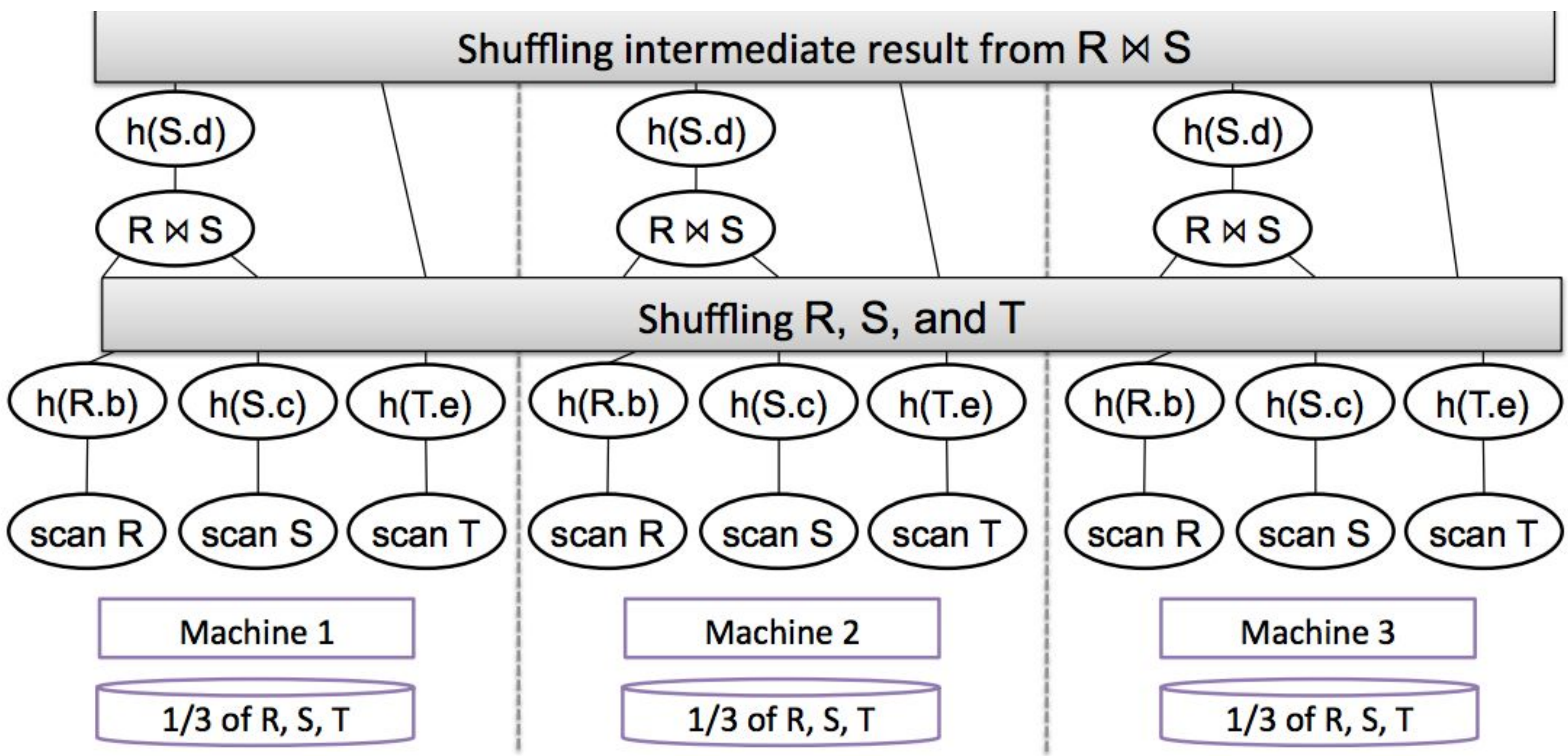
SELECT *
FROM R, S, T
WHERE R.b = S.c
AND S.d = T.e
AND (R.a - T.f) > 100



Problem 1)

R(a,b)
S(c,d)
T(e,f)

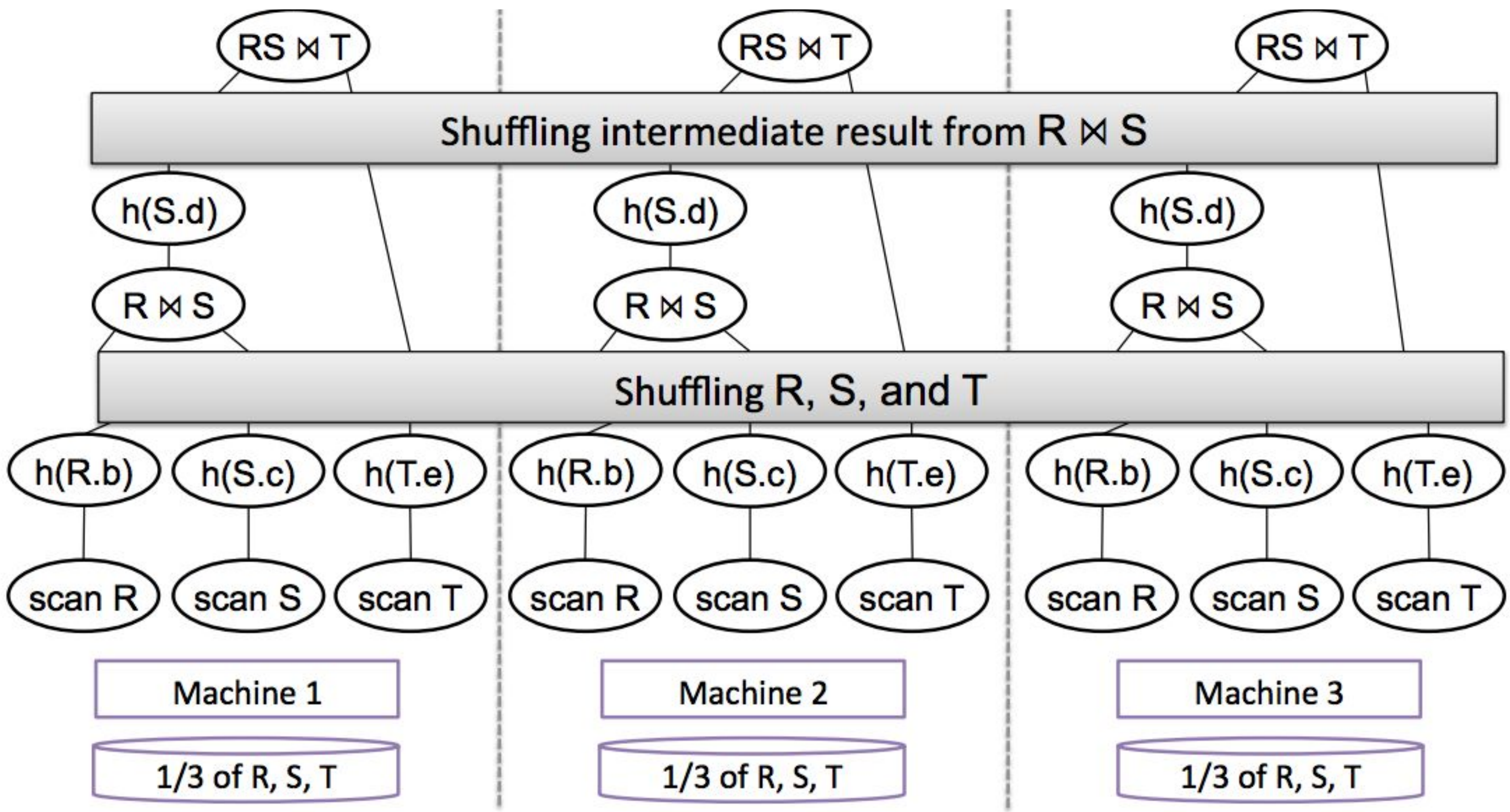
```
SELECT *
FROM R, S, T
WHERE R.b = S.c
      AND S.d = T.e
      AND (R.a - T.f) > 100
```



Problem 1)

R(a,b)
S(c,d)
T(e,f)

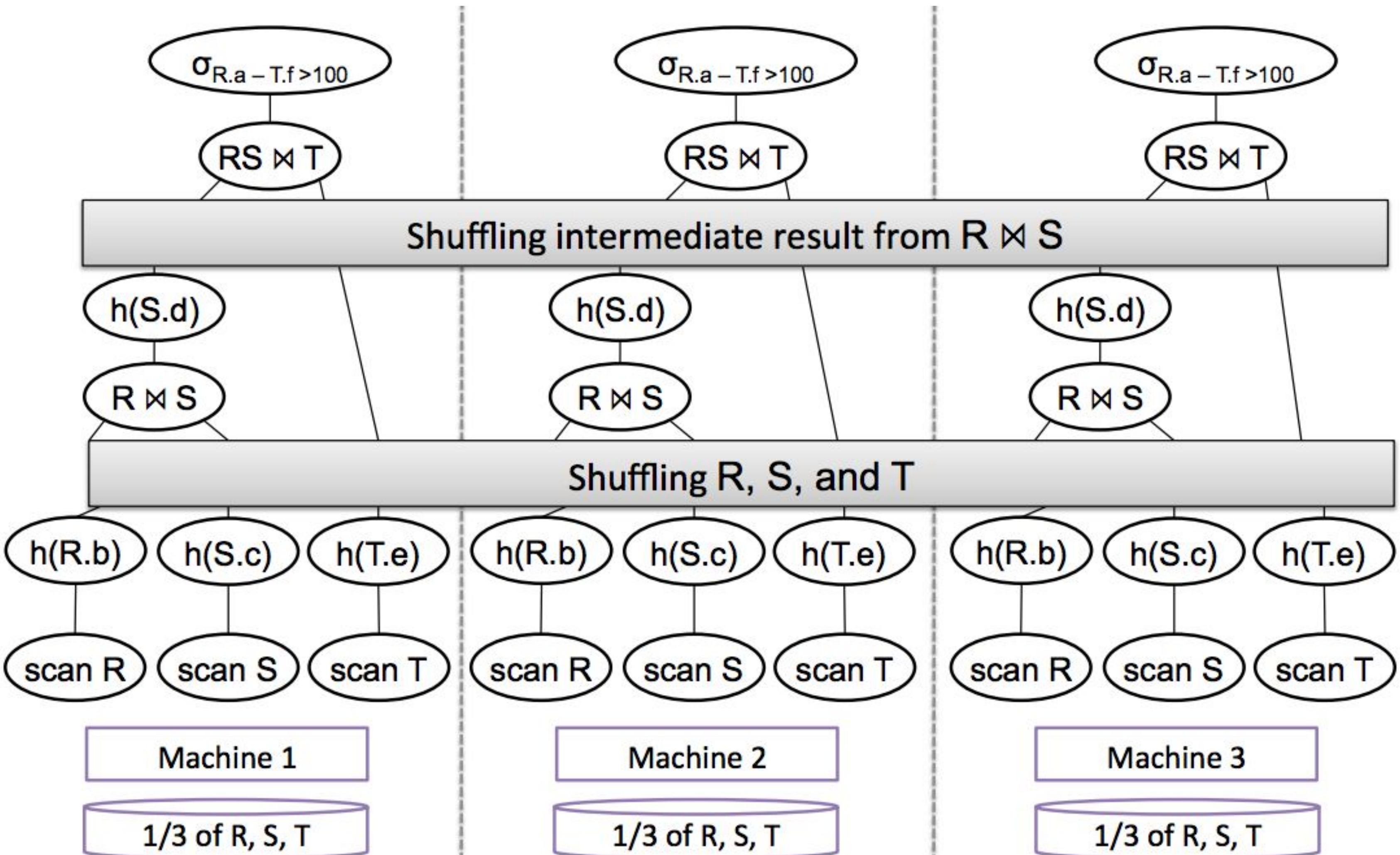
SELECT *
FROM R, S, T
WHERE R.b = S.c
AND S.d = T.e
AND (R.a - T.f) > 100



Problem 1)

R(a,b)
S(c,d)
T(e,f)

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



2PC Crash/Recovery Scenarios

Recovery Process

- At each active site a recovery process (RP) exists
  - It processes messages from RPs at other sites, and
  - handles all the transactions that were executing 2PC at the time of the last failure of the site.
- At recovery from a crash, the RP at the recovering site
  - reads the log on stable storage, and
  - accumulates in virtual storage information relating to transactions executing 2PC at the time of the crash.
- This information in virtual storage is used to
  - answer queries from other sites about transactions that had their coordinators at this site, and
  - to send unsolicited information to other 'subordinate sites' for transactions at this 'coordinator site'

2PC Recovery Scenarios

1. **If the recovery process finds that**
   - **a transaction was executing at the time of the crash,**
   - **And that no commit/prepare protocol log record had been written**

- Then the recovery process neither knows nor cares whether it is dealing with a subordinate or the coordinator of the transaction.
- It aborts that transaction by
  - “undoing” its actions, if any, using the UNDO log records,
  - writing an abort record,
  - and “forgetting” it.

2PC Recovery Scenarios

2. **If the recovery process at a <u>coordinator</u> finds a transaction in the <u>committing (resp. aborting)</u> state**

- It periodically tries to send the COMMIT (ABORT) to all the subordinates that have not acknowledged and awaits their ACKs.
- Once all the ACKs are received, the recovery process writes the end record and “forgets” the transaction.

2PC Recovery Scenarios

3. If the coordinator process notices the failure of a subordinate while waiting for the latter to send its vote

- then the former aborts the transaction (and follows the necessary steps).

4. If the failure occurs when the coordinator is waiting to get an ACK

- then the coordinator hands the transaction over to the recovery process.
- (commit/abort state must be maintained)

2PC Recovery Scenarios

5. **If a <u>subordinate notices</u> the <u>failure of the coordinator</u> <u>before</u> the former sent a YES VOTE and moved into the prepared state**

- then it aborts the transaction (unilateral abort)

6. **If the failure (of the coordinator) occurs <u>after</u> the <u>subordinate is in prepared state</u>**

- then the subordinate hands the transaction over to the recovery process.

2PC Recovery Scenarios

7. When a recovery process receives an inquiry message from a prepared subordinate site

- it looks at its information in virtual storage.
- If it has information that says the transaction is in the aborting or committing state, then it sends the appropriate response.

- **What if no information is found?**

2PC Recovery Scenarios

7 contd.

- **How can such a situation arise when no info is found**

- Both COMMITS and ABORTS are being acknowledged
- Inquiry is being made means that the inquirer had not received and processed a COMMIT/ABORT before the inquiree "forgot" the transaction.
- Such a situation comes about when
  - (1) the inquiree sends out PREPARES,
  - (2) it crashes before receiving all the votes and deciding to commit/abort, and
  - (3) on restart, it aborts the transaction and does not inform any of the subordinates.

2PC Recovery Scenarios

7 contd.

- **What to do?**

- On restart, the recipient of an inquiry cannot tell whether it is a coordinator or subordinate, if no commit protocol log records exist for the transaction.
- Given this fact, the correct response to an inquiry in the no information case is an ABORT.

2PC Recovery Scenarios

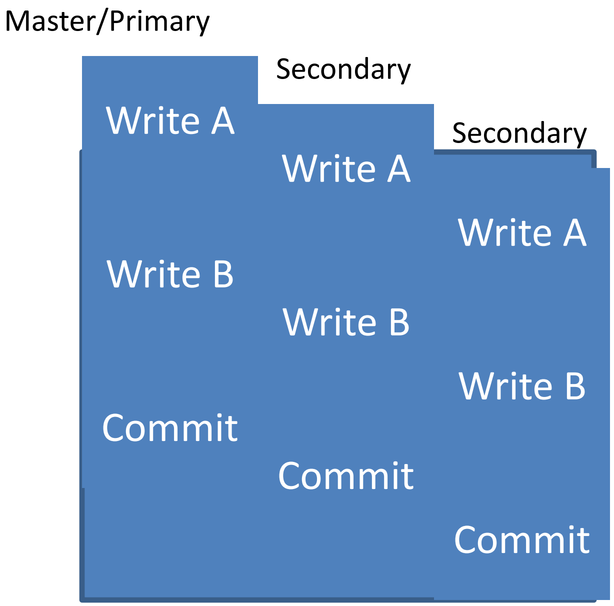
8. When the recovery process finds that it (the subordinate) is in the prepared state for a particular transaction

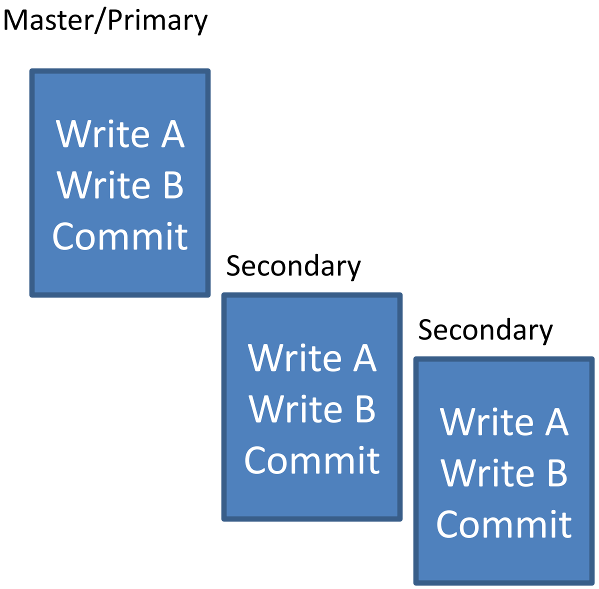
- It cannot take a unilateral decision
- The rest you have to figure out ☺
- Write in HW6 what happens from receiving the PREPARE message at this subordinate
- Note that we assumed that a site will recover at some point after a crash.

Replication

Eager (Synchronous) vs. Lazy (Asynchronous)

- **Eager:** Updates are applied to all replicas of an object as part of the original transaction (needs global locks, 2PC).
- **Lazy:** One replica is updated by the originating transaction. Updates to other replicas propagate asynchronously, typically as a <u>separate transaction</u> for each node.





Master vs. Group

- **Master:**
  - Each object has a master node. Only the master can update the primary copy of the object.
  - All other replicas are read-only. If they want to update the object request the master do the update.
- **Group**:
  - Any node with a copy of a data item can update it (also called "update anywhere")

Propagation vs. Ownership

	Eager	Lazy
Master	1 transaction 1 object owner	N transactions 1 object owner
Group	1 transactions N object owners	N transactions N object owners

Summary

A. Synchronous/Eager

- Option A1: Use a master
- Option A2: Use a quorum (cluster)

HW6, 2b

B. Asynchronous/Lazy:

- Option B1: Use a master. All updates have to go to the master first.
- Option B2: Allow updates to go everywhere. This is **multi-master**.

Ref.

Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The Dangers of Replication and a Solution. ACM SIGMOD Record (25)2, 1996