CSE 444: Database Internals

Section 8:

Parallel Processing

Review in this section

1. Parallel DBMS + MapReduce
2. 2- Phase Commit (2PC)
3. Demo for parallel SimpleDB (by Jingjing)

1a. 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).

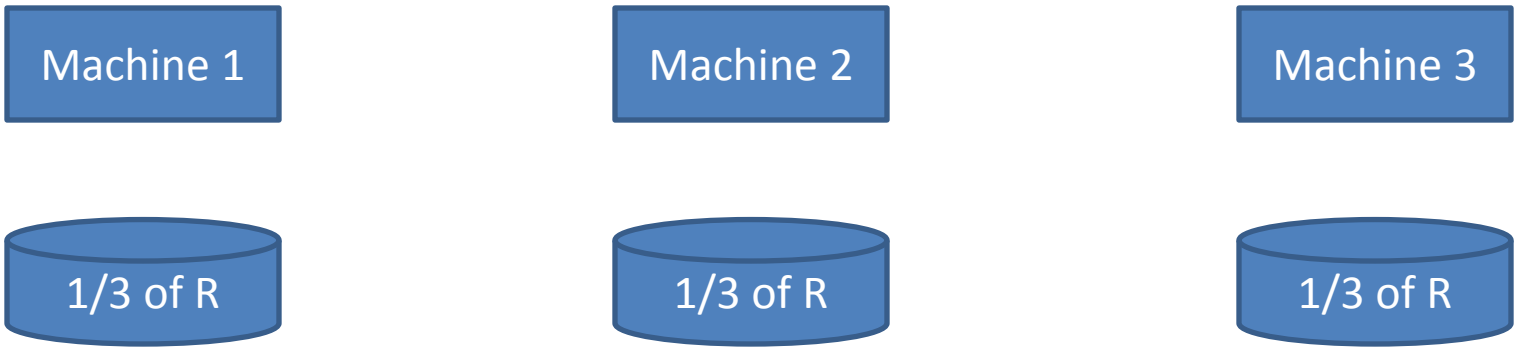
Show a RA plan for this query and how it will be executed across the N = 3 machines.

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)

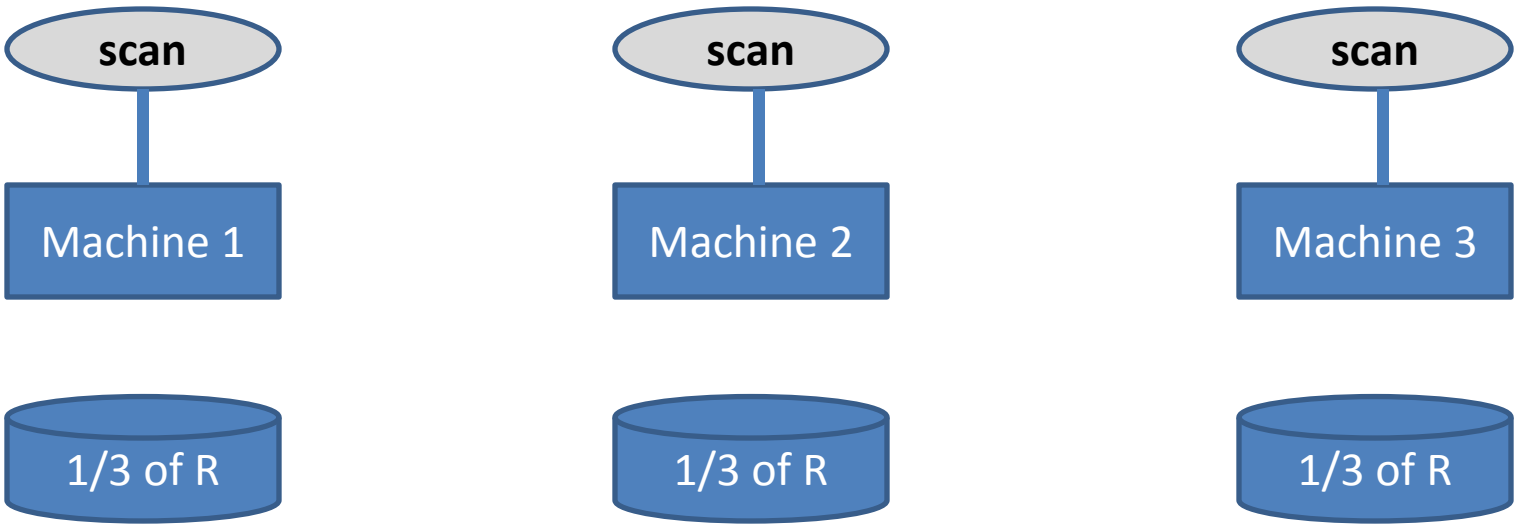
```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```



R(a, b)

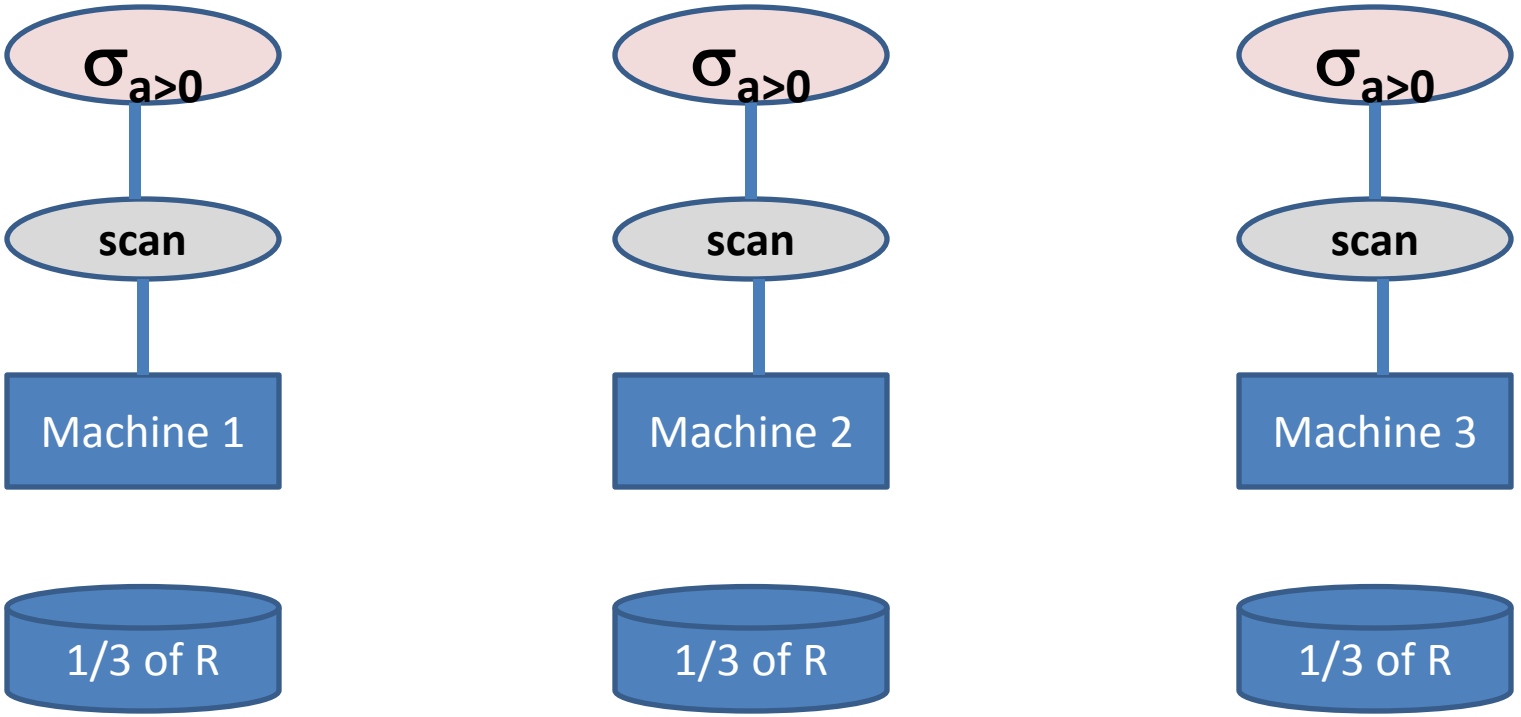
```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```

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



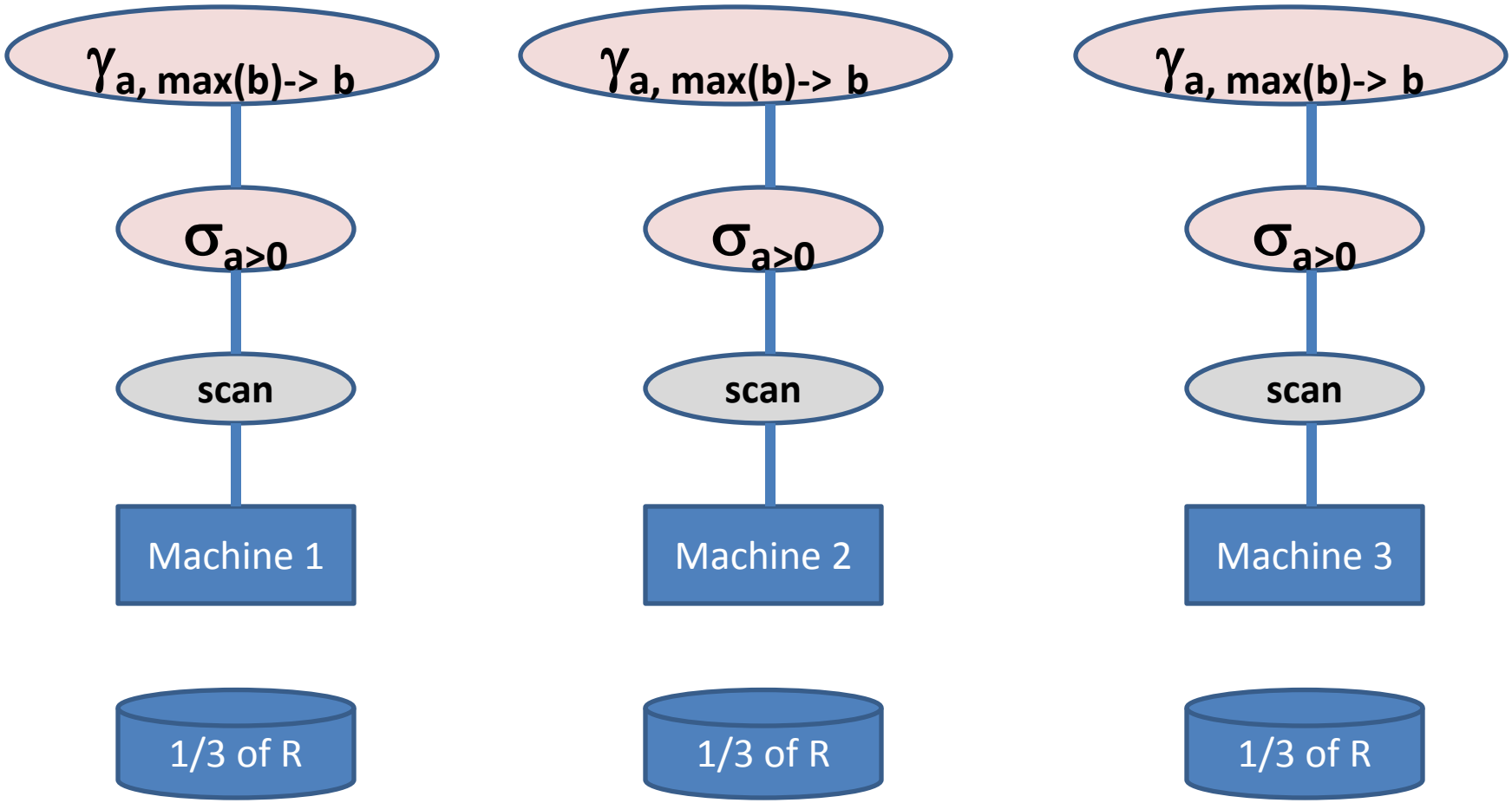
R(a, b)

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



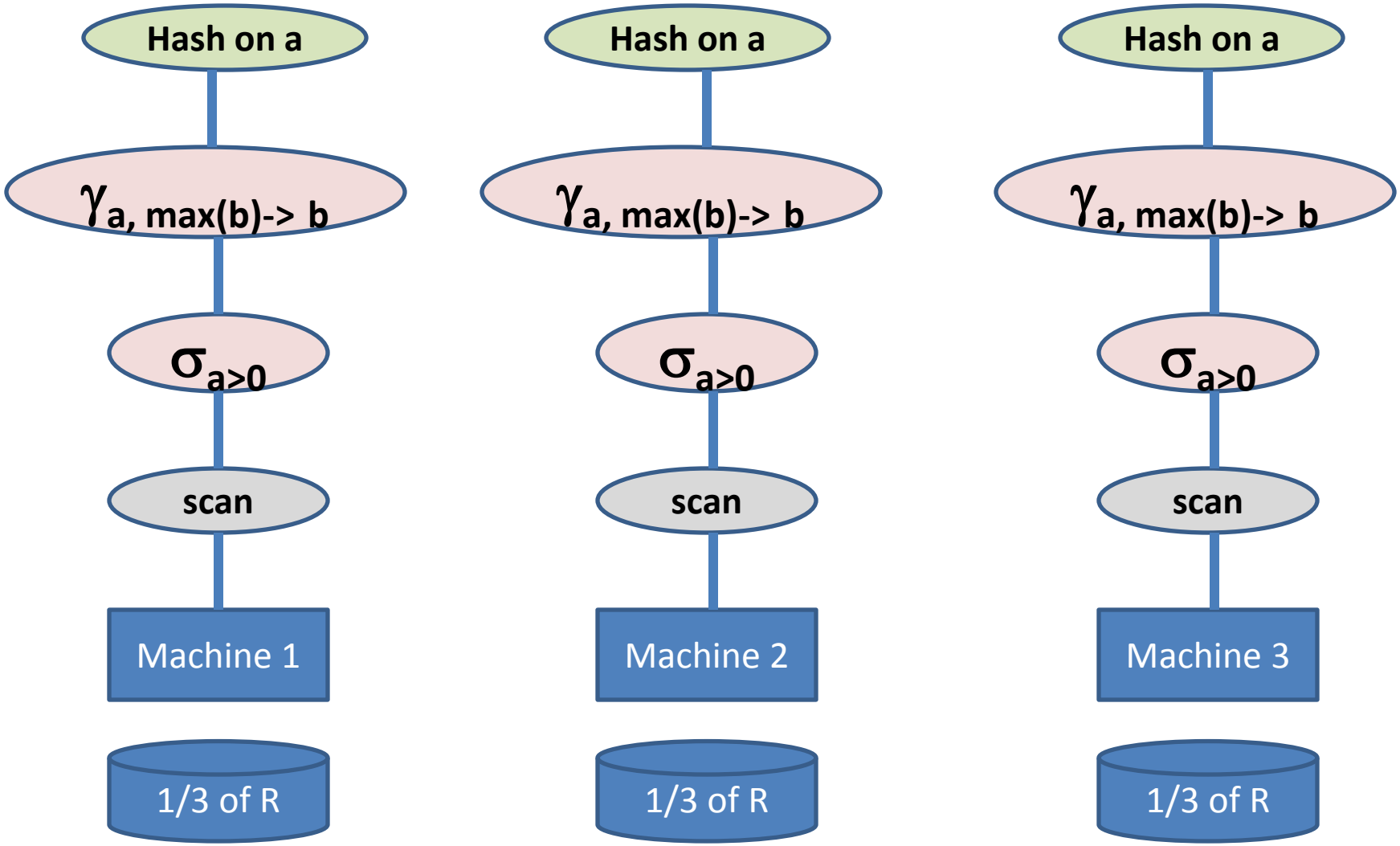
R(a, b)

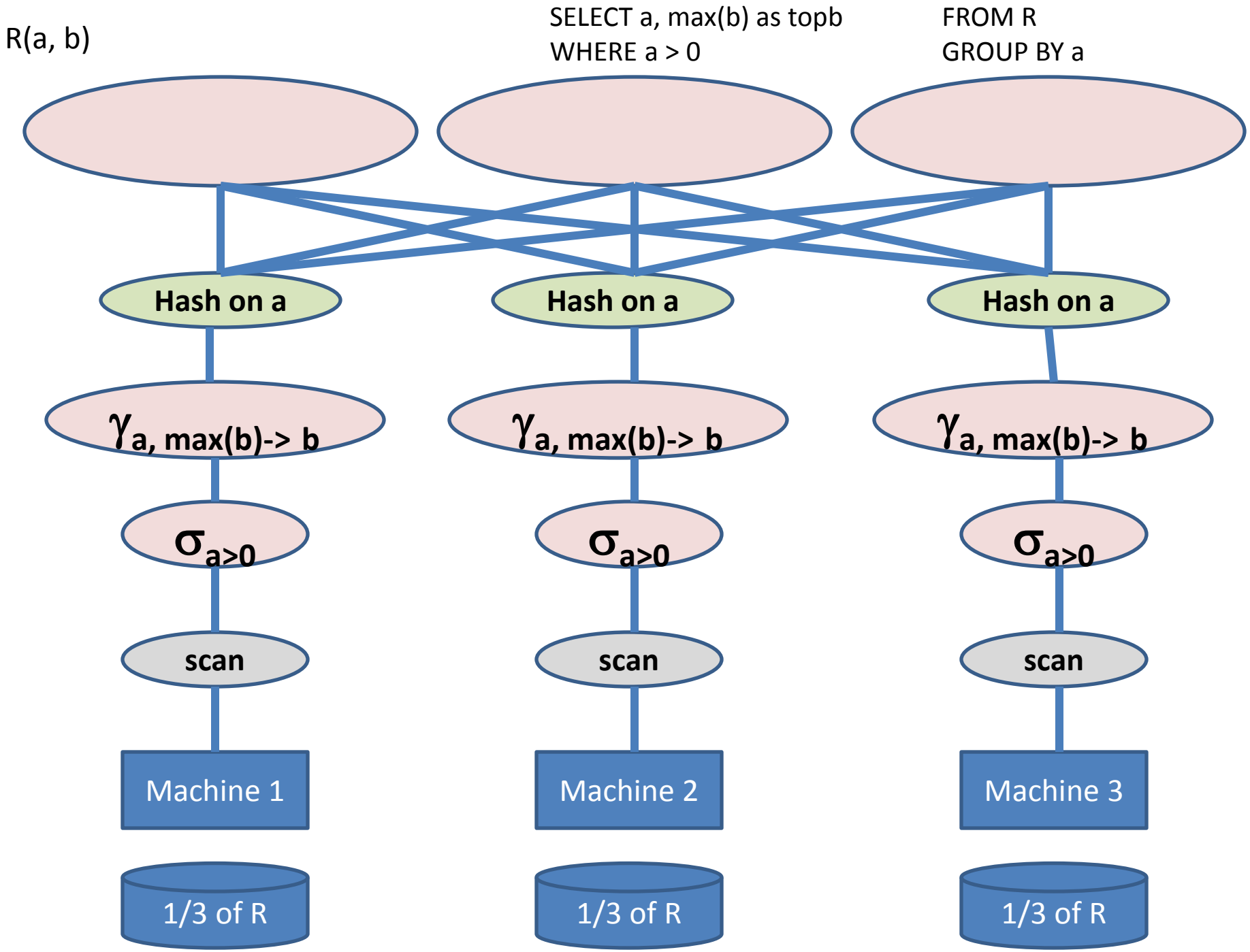
```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```

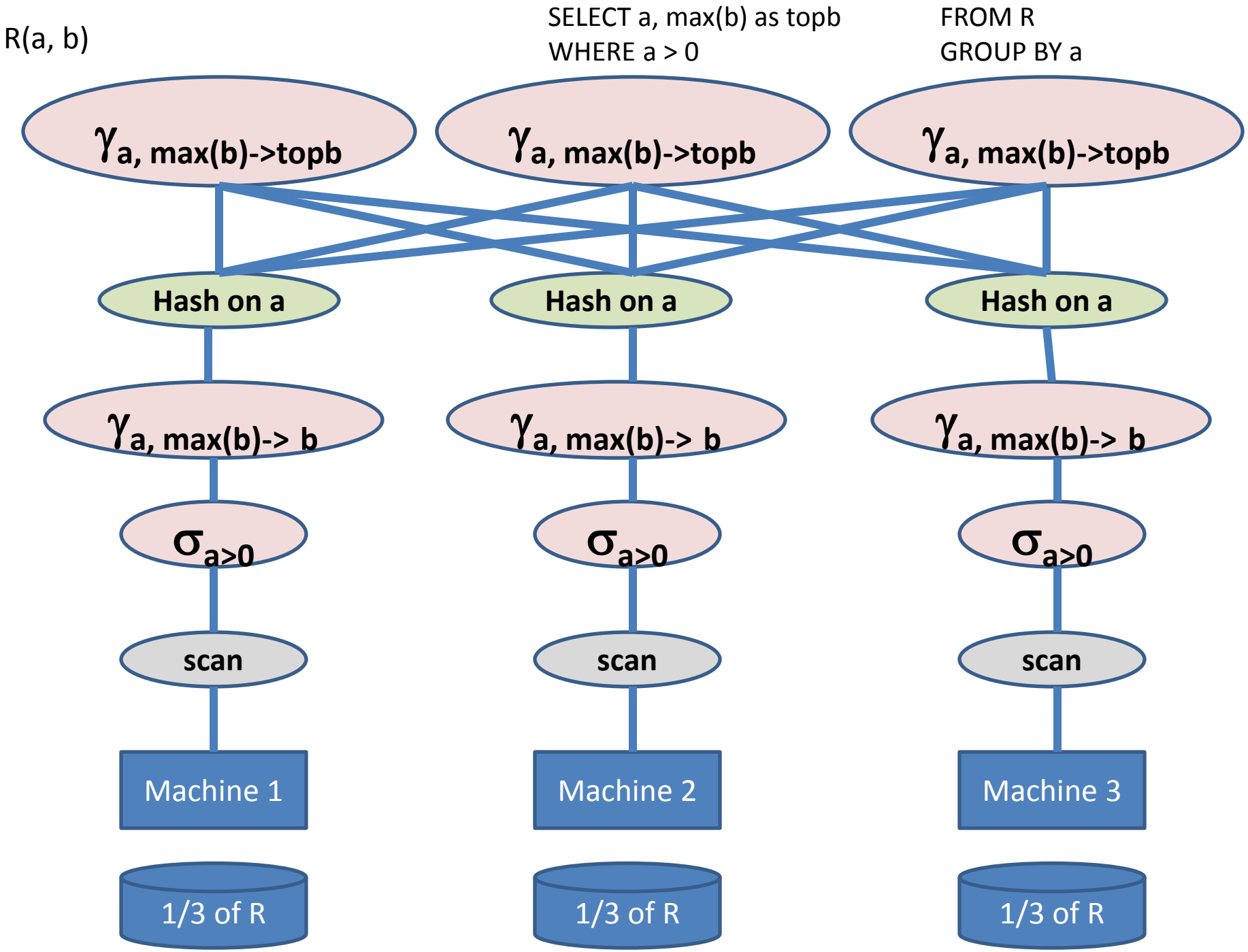


R(a, b)

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




R(a, b)
SELECT a, max(b) as topb
WHERE a > 0
FROM R
GROUP BY a
Hash on a
Hash on a
Hash on a
γa, max(b)-> b
γa, max(b)-> b
γa, max(b)-> b
σa>0
σa>0
σa>0
scan
scan
scan
Machine 1
Machine 2
Machine 3
1/3 of R
1/3 of R
1/3 of R


R(a, b)
SELECT a, max(b) as topb
WHERE a > 0
FROM R
GROUP BY a
γa, max(b)->topb
γa, max(b)->topb
γa, max(b)->topb
Hash on a
Hash on a
Hash on a
γa, max(b)-> b
γa, max(b)-> b
γa, max(b)-> b
σa>0
σa>0
σa>0
scan
scan
scan
Machine 1
Machine 2
Machine 3
1/3 of R
1/3 of R
1/3 of R

1b. Map Reduce

Explain how the query will be executed in MapReduce

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

Specify the computation performed in the map and the reduce functions

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

Map

- Each map task
  - Scans a block of R
  - Calls the map function for each tuple
  - The map function applies the selection predicate to the tuple
  - For each tuple satisfying the selection, it outputs a record with key = a and value = b

•When each map task scans multiple relations, it needs to output something like **key = a and value = (‘R’, b)**
which has the relation name ‘R’

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

Shuffle

- The MapReduce engine reshuffles the output of the map phase and groups it on the intermediate key, i.e. the attribute a

•Note that the programmer has to write only the map and reduce functions, the shuffle phase is done by the MapReduce engine (although the programmer can rewrite the partition function), but you should still mention this in HW6 answers.

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

Reduce

- Each reduce task
  - computes the aggregate value **max(b) = topb** for each group (i.e. ***a***) assigned to it (by calling the reduce function)
  - outputs the final results: **(a, topb)**

A local combiner can be used to compute local max before data gets reshuffled (in the map tasks)

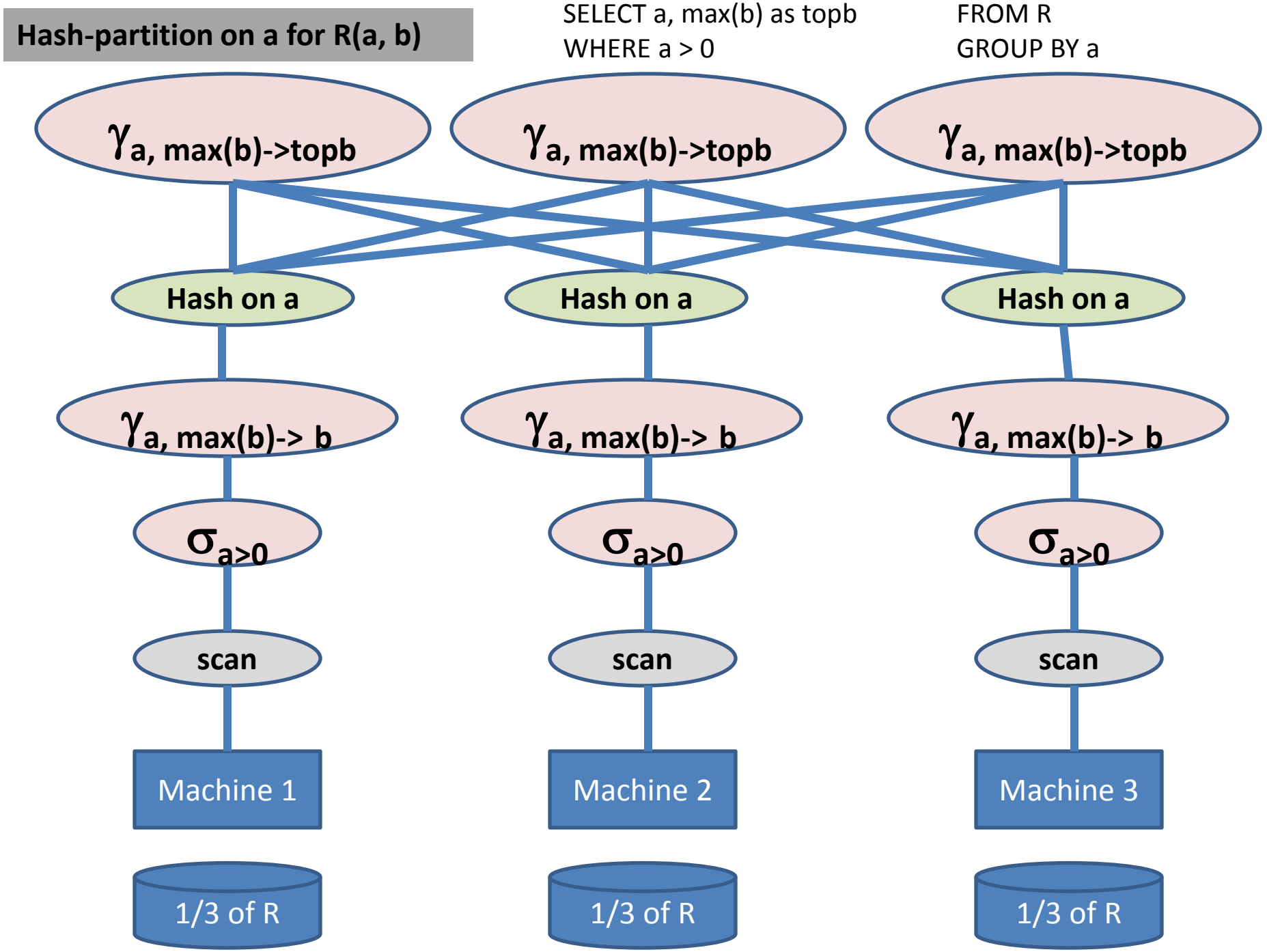
- Multiple aggregates can be output by the reduce phase like **key = a and value = (sum(b), min(b))** etc.

- Sometimes a second (third etc) level of Map-Reduce phase might be needed

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

1c. Benefit of hash-partitioning

- What would change if we hash-partitioned R on R.a before executing this query
  - For parallel DBMS
  - For MapReduce


Hash-partition on a for R(a, b)
SELECT a, max(b) as topb
WHERE a > 0
FROM R
GROUP BY a
γa, max(b)->topb
γa, max(b)->topb
γa, max(b)->topb
Hash on a
Hash on a
Hash on a
γa, max(b)-> b
γa, max(b)-> b
γa, max(b)-> b
σa>0
σa>0
σa>0
scan
scan
scan
Machine 1
Machine 2
Machine 3
1/3 of R
1/3 of R
1/3 of R

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

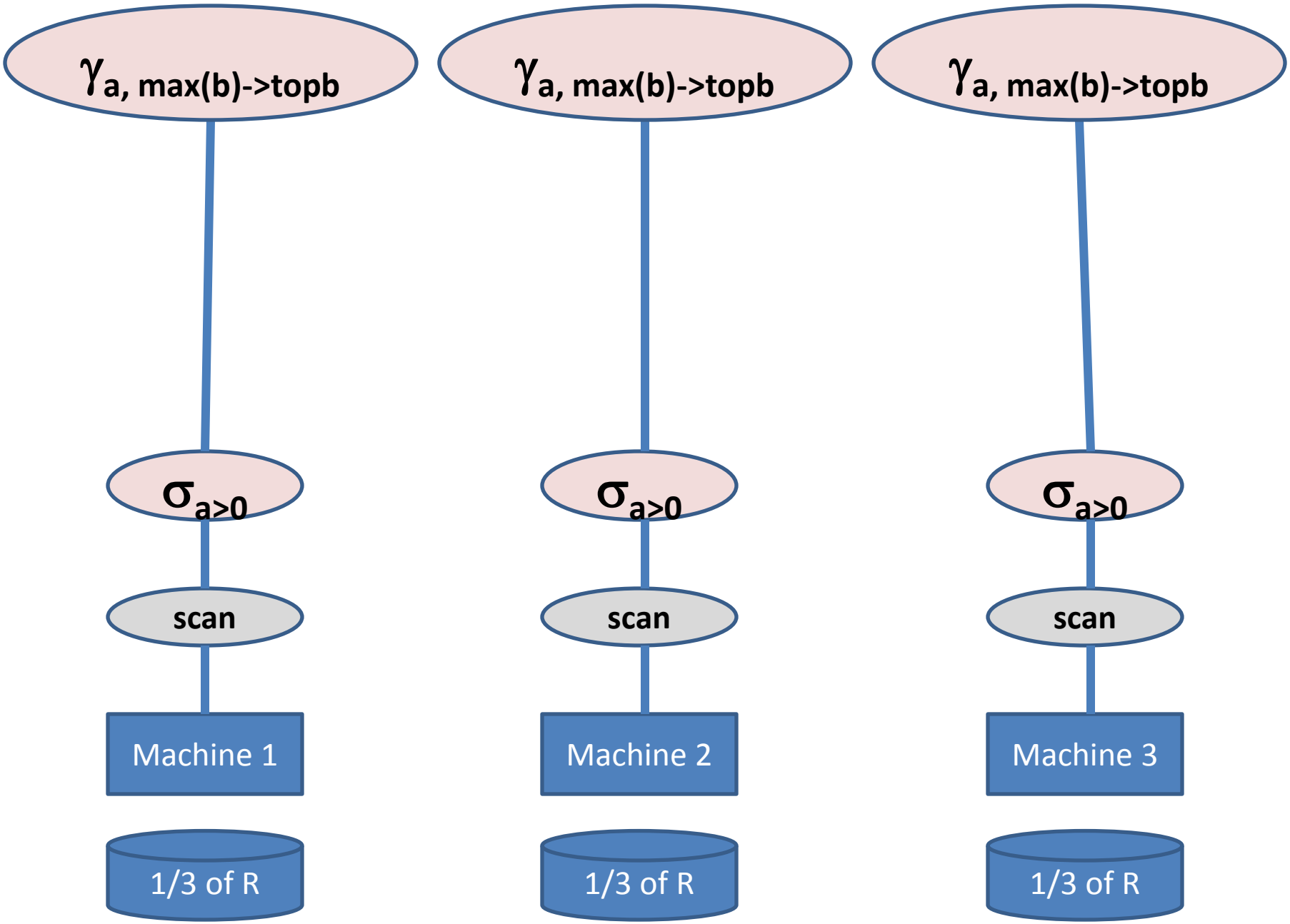
1c. Benefit of hash-partitioning

- **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
WHERE a > 0

FROM R
GROUP BY a



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

1c. Benefit of hash-partitioning

- **For MapReduce**
  - Logically, MR won't know that the data is hash-partitioned
  - MR treats map and reduce functions as black-boxes and does not perform any optimizations on them
- But, if a local combiner is used
  - Saves communication cost:
    - fewer tuples will be emitted by the map tasks
  - Saves computation cost in the reducers:
    - the reducers would have to do anything

2. 2PC

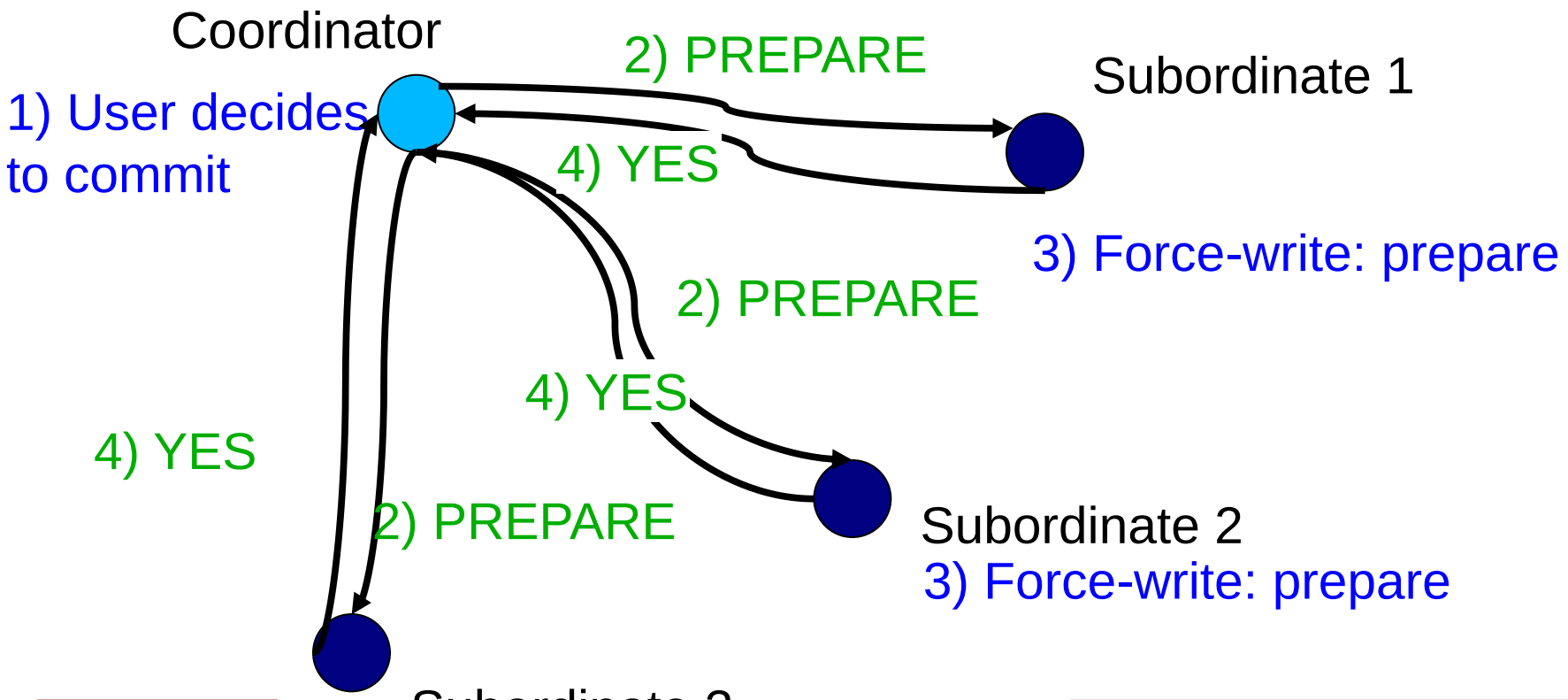
- In the 2PC protocol, what happens if
  - a coordinator sends PREPARE messages
  - all but one subordinate vote to commit the transaction
  - the last subordinate also wants to commit, but it crashes before receiving the PREPARE message from the coordinator
- Review?
  - Reading Ramakrishnan-Gehrke book and/or the IBM paper might be useful (only 2-3 pages)

Review: Two-Phase Commit Protocol

- One coordinator and many subordinates
  - Phase 1: prepare
  - Phase 2: commit or abort
  - Log records for 2PC include transaction and coordinator ids
  - Coordinator also logs ids of all subordinates

- Principle
  - When a process makes a decision: vote yes/no or commit/abort
  - Or when a subordinate wants to respond to a message: ack
  - First force-write a log record (to make sure it survives a failure)
  - Only then send message about decision

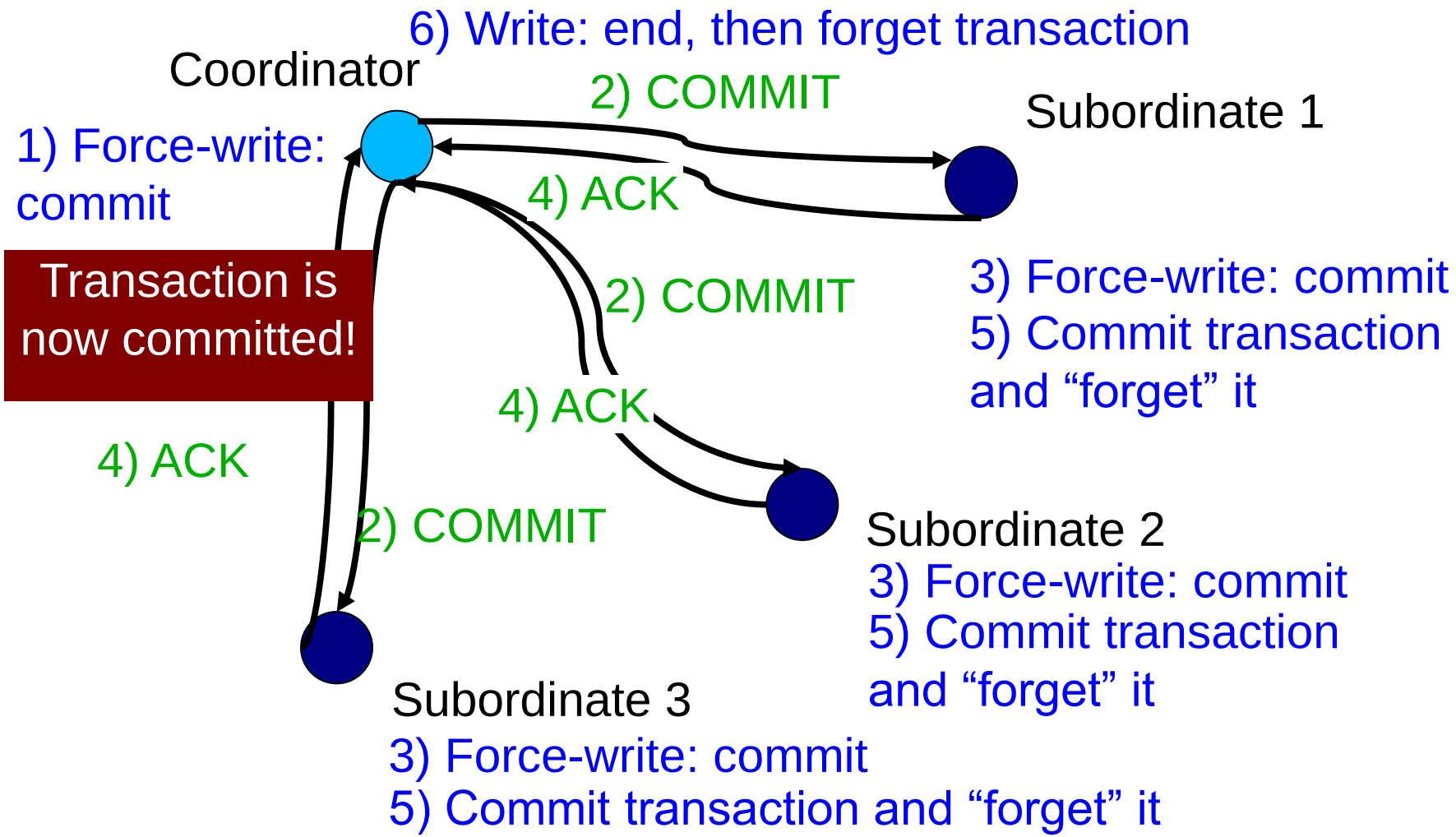
Review: 2PC: Phase 1, Prepare



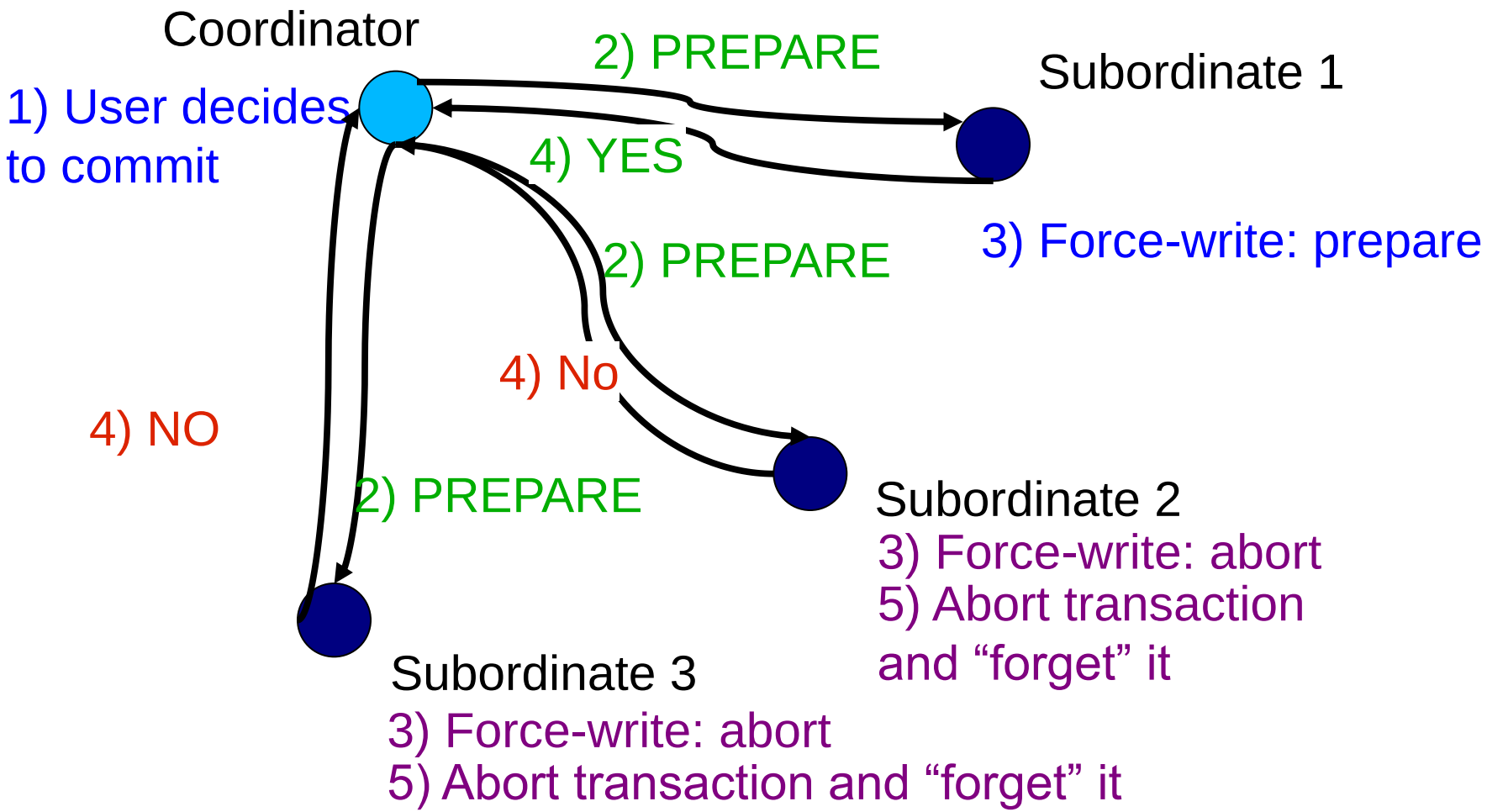
NO vote is like a veto

Coordinator/subordinate for a specific transaction

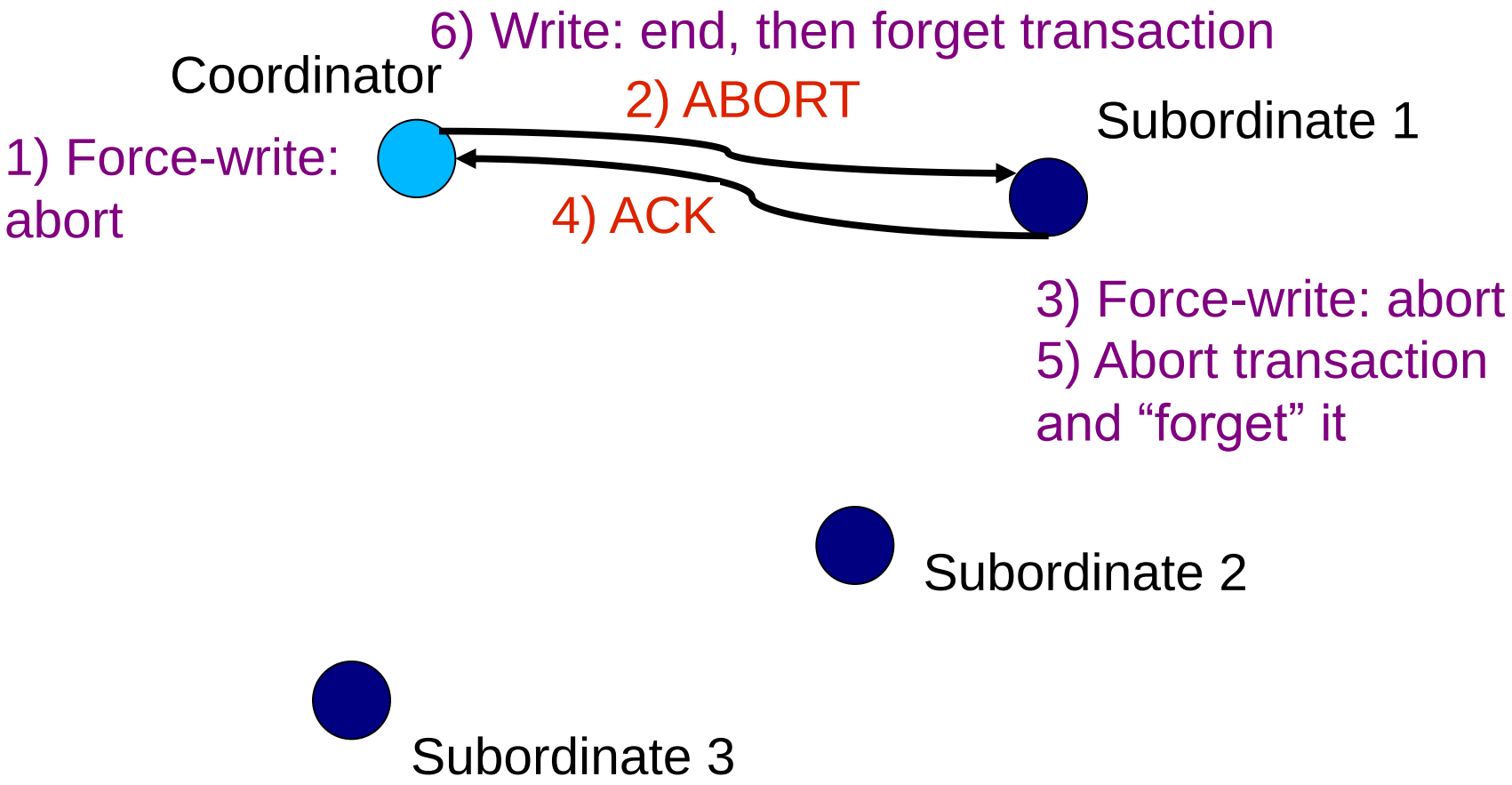
Review: 2PC: Phase 2, Commit



Review: 2PC with Abort



Review: 2PC with Abort



2. 2PC - Solution

a coordinator sends PREPARE messages, all but one subordinate vote to commit, the last subordinate crashes before receiving the PREPARE msg

- Coordinator
  - Time-out waiting for the reply from the failed subordinate
  - Will decide to abort the transaction
  - Write an abort log record
  - Will send a ABORT message to the subordinates

2. 2PC - Solution

a coordinator sends PREPARE messages, all but one subordinate vote to commit, the last subordinate crashes before receiving the PREPARE msg

- The subordinates that did not crash
  - Will receive the ABORT from the coordinator
  - Write an abort log record
  - Will abort the transaction and “forget” it

2. 2PC - Solution

a coordinator sends PREPARE messages, all but one subordinate vote to commit, the last subordinate crashes before receiving the PREPARE msg

- The subordinate that crashed
  - After recovery will find that a transaction was executing at the time of the crash, but no commit log been written
  - The recovery process will abort the transaction
  - Write an abort log record
  - Will abort the transaction and “forget” it