Introduction to Data Management CSE 344

Lecture 28

Parallel Databases Wrap-up

Announcements

- Homework 8 (last) due on Thursday night
  - Help each other out with configuration funnies
- Final exam next Monday, 2:30
  - Review Sunday afternoon, 2:00

A Challenge

- Have P servers (say P=27 or P=1000)
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)

A Challenge

- Have P servers (say P=27 or P=1000)
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)

- This computes all “triangles”.
- E.g. let Follows(x,y) be all pairs of Twitter users s.t. x follows y. Let R=S=T=Follows. Then Q computes all triples of people that follow each other.

A Challenge

- Have P servers (say P=27 or P=1000)
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Step 1:
  - Each server sends R(x,y) to server h(y) mod P
  - Each server sends S(y,z) to server h(y) mod P

A Challenge

- Have P servers (say P=27 or P=1000)
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Step 1:
  - Each server sends R(x,y) to server h(y) mod P
  - Each server sends S(y,z) to server h(y) mod P
- Step 2:
  - Each server computes R⋈S locally
  - Each server sends [R(x,y),S(y,z)] to h(x) mod P
  - Each server sends T(z,x) to h(x) mod P

A Challenge

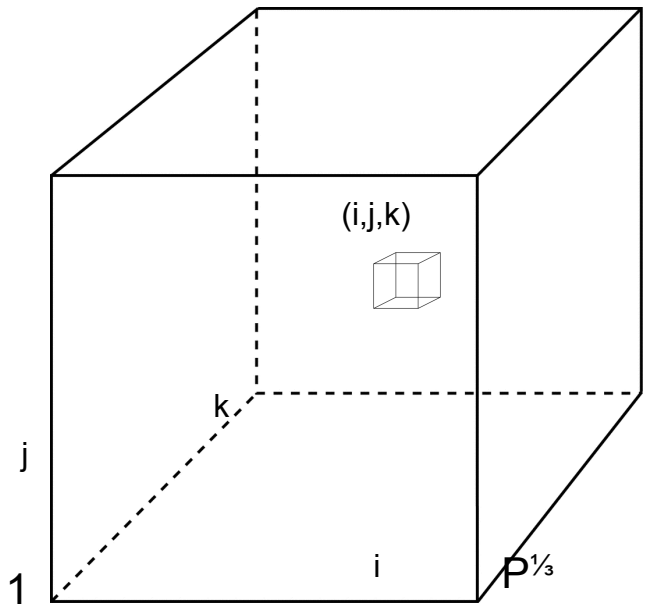
- Have P servers (say P=27 or P=1000)
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Step 1:
  - Each server sends R(x,y) to server h(y) mod P
  - Each server sends S(y,z) to server h(y) mod P
- Step 2:
  - Each server computes R⋈S locally
  - Each server sends [R(x,y),S(y,z)] to h(x) mod P
  - Each server sends T(z,x) to h(x) mod P
- Final output:
  - Each server computes locally and outputs R⋈S⋈T

A Challenge

- Have P servers (say P=27 or P=1000)
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)

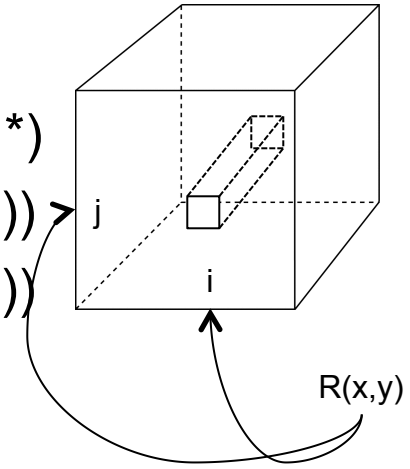
A Challenge

- Have P servers (say P=27 or P=1000)
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Organize the P servers into a cube with side $P^{1/3}$
  - Thus, each server is uniquely identified by (i,j,k), i,j,k≤$P^{1/3}$



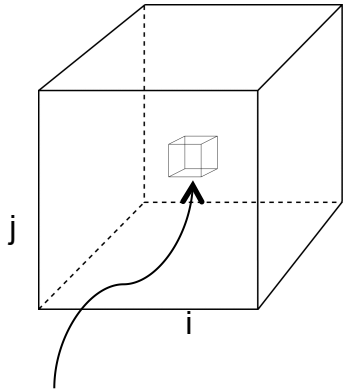
A Challenge

- Have P servers (say P=27 or P=1000)
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Organize the P servers into a cube with side $P^{1/3}$
  - Thus, each server is uniquely identified by (i,j,k), i,j,k≤$P^{1/3}$
- Step 1:
  - Each server sends R(x,y) to all servers (h(x),h(y),*)
  - Each server sends S(y,z) to all servers (*,h(y),h(z))
  - Each server sends T(x,z) to all servers (h(x),*,h(z))



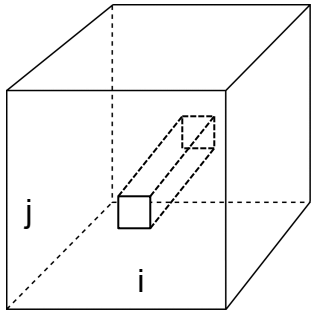
A Challenge

- Have P servers (say P=27 or P=1000)
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Organize the P servers into a cube with side $P^{1/3}$
  - Thus, each server is uniquely identified by (i,j,k), i,j,k≤$P^{1/3}$
- Step 1:
  - Each server sends R(x,y) to all servers (h(x),h(y),*)
  - Each server sends S(y,z) to all servers (*,h(y),h(z))
  - Each server sends T(x,z) to all servers (h(x),*,h(z))
- Final output:
  - Each server (i,j,k) computes the query R(x,y),S(y,z),T(z,x) locally



A Challenge

- Have P servers (say P=27 or P=1000)
- How do we compute this query in one step?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Organize the P servers into a cube with side $P^{1/3}$
  - Thus, each server is uniquely identified by (i,j,k), i,j,k≤$P^{1/3}$
- Step 1:
  - Each server sends R(x,y) to all servers (h(x),h(y),*)
  - Each server sends S(y,z) to all servers (*,h(y),h(z))
  - Each server sends T(x,z) to all servers (h(x),*,h(z))



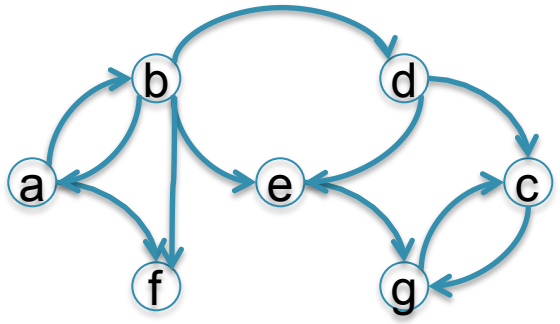
- Final output:
  - Each server (i,j,k) computes the query R(x,y),S(y,z),T(z,x) locally
- Analysis: each tuple R(x,y) is replicated at most $P^{1/3}$ times

Graph Analysis

Graph Databases

Many large databases are graphs

- Give examples in class

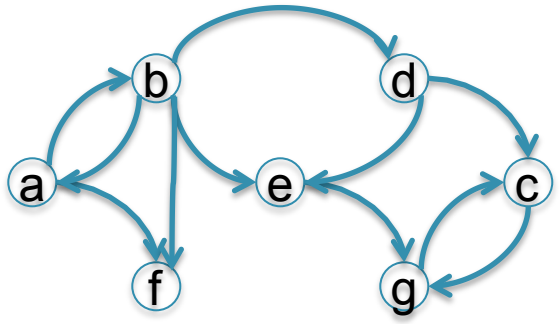


Source	Target
a	b
b	a
a	f
b	f
b	e
b	d
d	e
d	c
e	g
g	c
c	g

Graph Databases

Many large databases are graphs

- Give examples in class
- The Web
- The Internet
- Social Networks
- Flights between airports
- Etc.

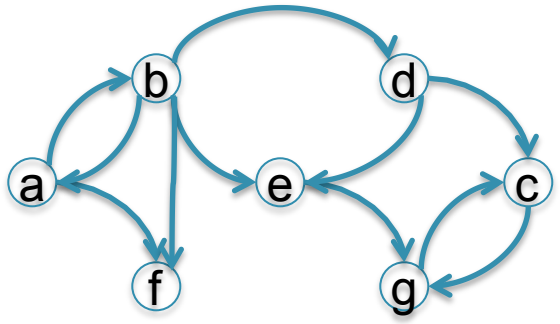


Source	Target
a	b
b	a
a	f
b	f
b	e
b	d
d	e
d	c
e	g
g	c
c	g

Data Analytics on Big Graphs

Queries expressible in SQL:

- How many nodes (edges)?
- How many nodes have > 4 neighbors?
- Which are "most connected nodes"?



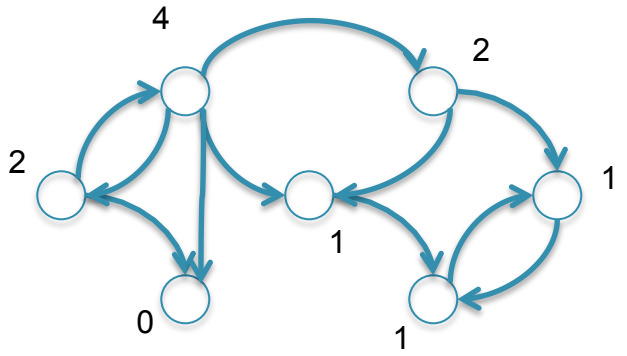
Source	Target
a	b
b	a
a	f
b	f
b	e
b	d
d	e
d	c
e	g
g	c
c	g

Queries requiring recursion:

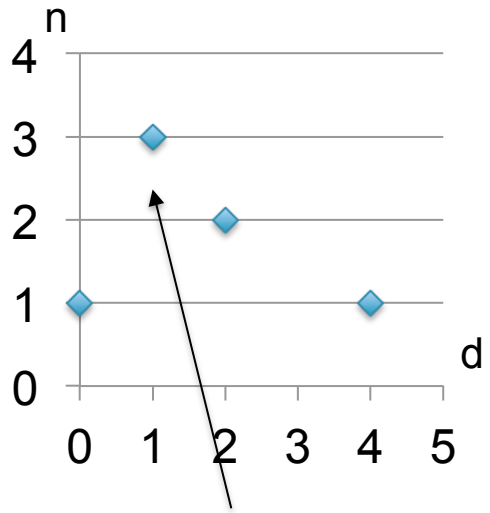
- Is the graph connected?
- What is the diameter of the graph?
- Compute *PageRank*
- Compute the *Centrality* of each node

Example: the Histogram of a Graph

- Outdegree of a node = number of outgoing edges
- For each d, let n(d) = number of nodes with oudegree d
- The outdegree histogram of a graph = the scatterplot (d, n(d))

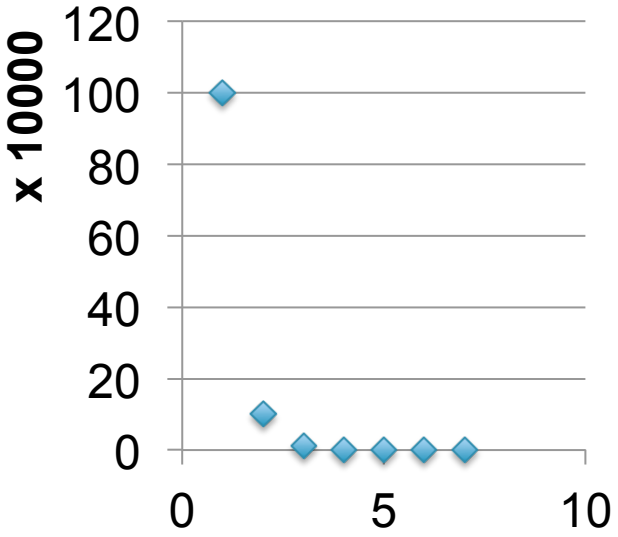


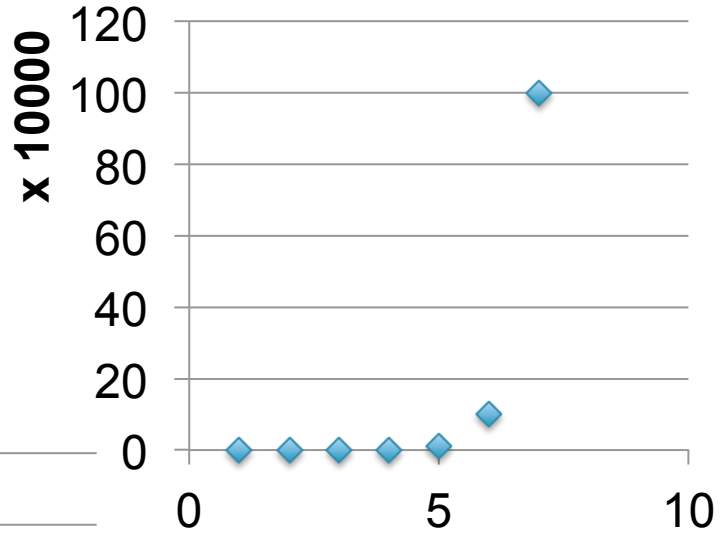
d	n(d)
0	1
1	3
2	2
3	0
4	1

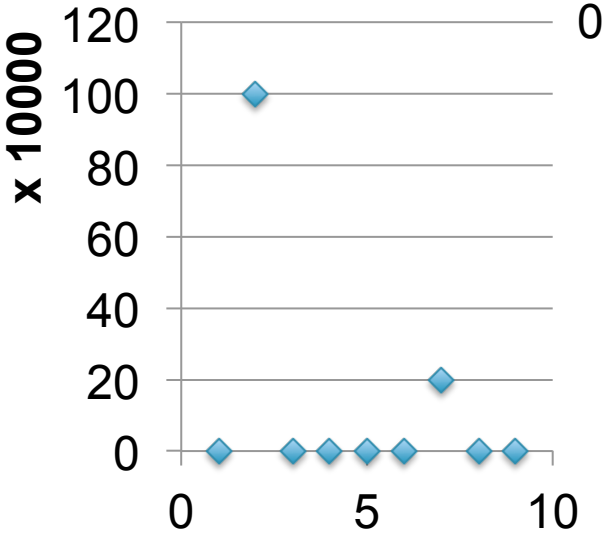


Outdegree 1 is seen at 3 nodes

Histograms Tell Us Something About the Graph





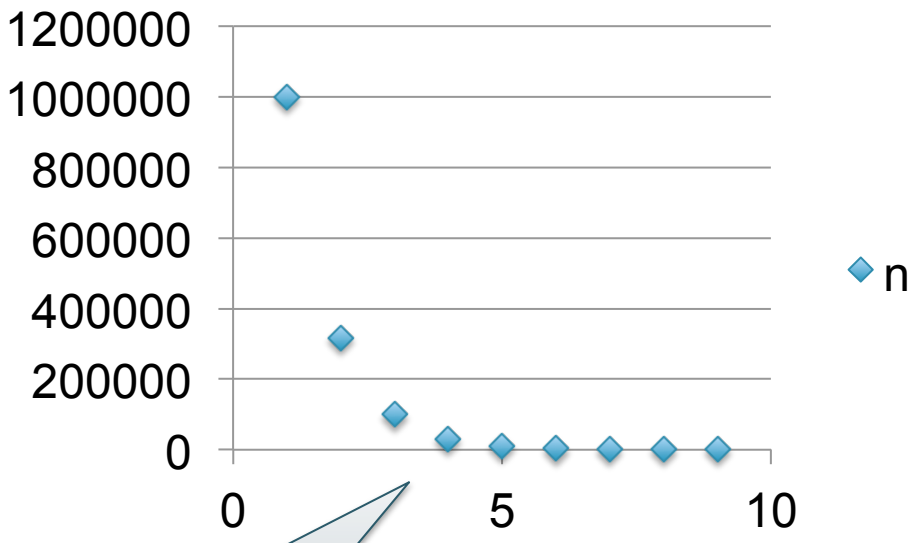


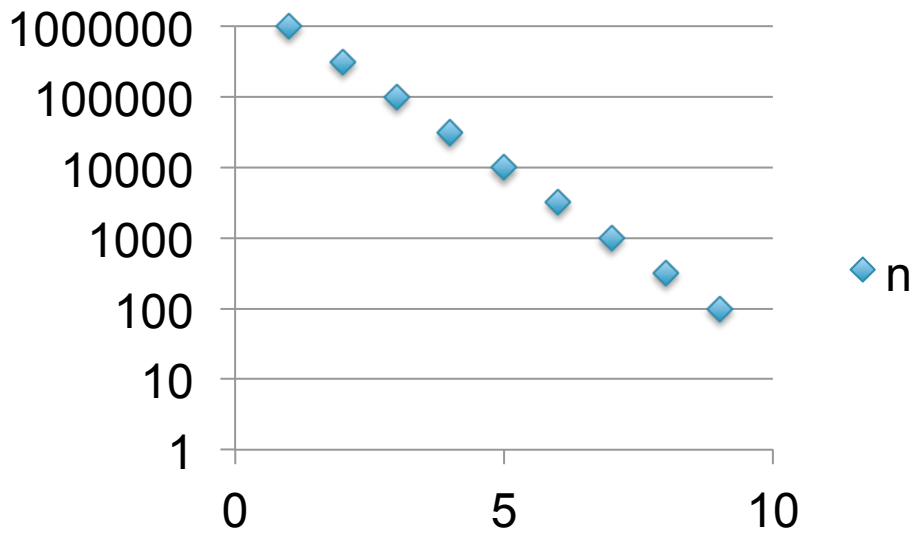
What can you say about these graphs?

Exponential Distribution

nodes with degree d

- $n(d) \cong c/2^d$ (generally, cx^d, for some $x < 1$)
- A *random graph* has exponential distribution
- Best seen when n is on a log scale

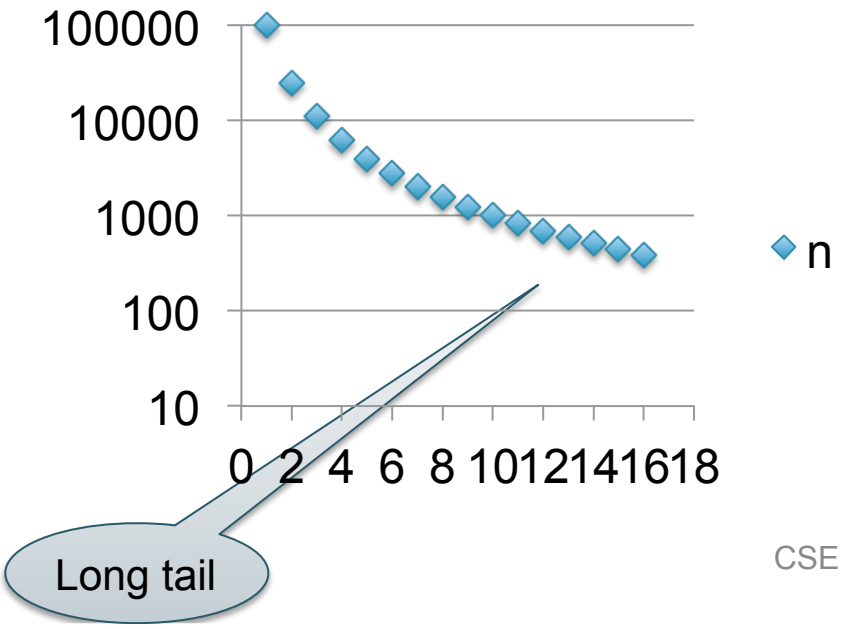


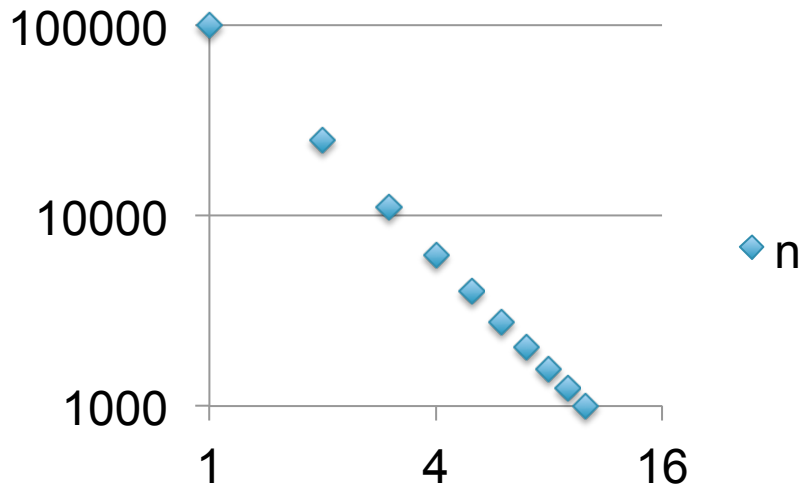


Quickly vanishing

Power Law Distribution (Zipf)

- $n(d) \cong 1/d^x$, for some value $x>0$
- Human-generated data follows power law: letters in alphabet, words in vocabulary, etc.
- Best seen in a log-log scale





The Histogram of the Web

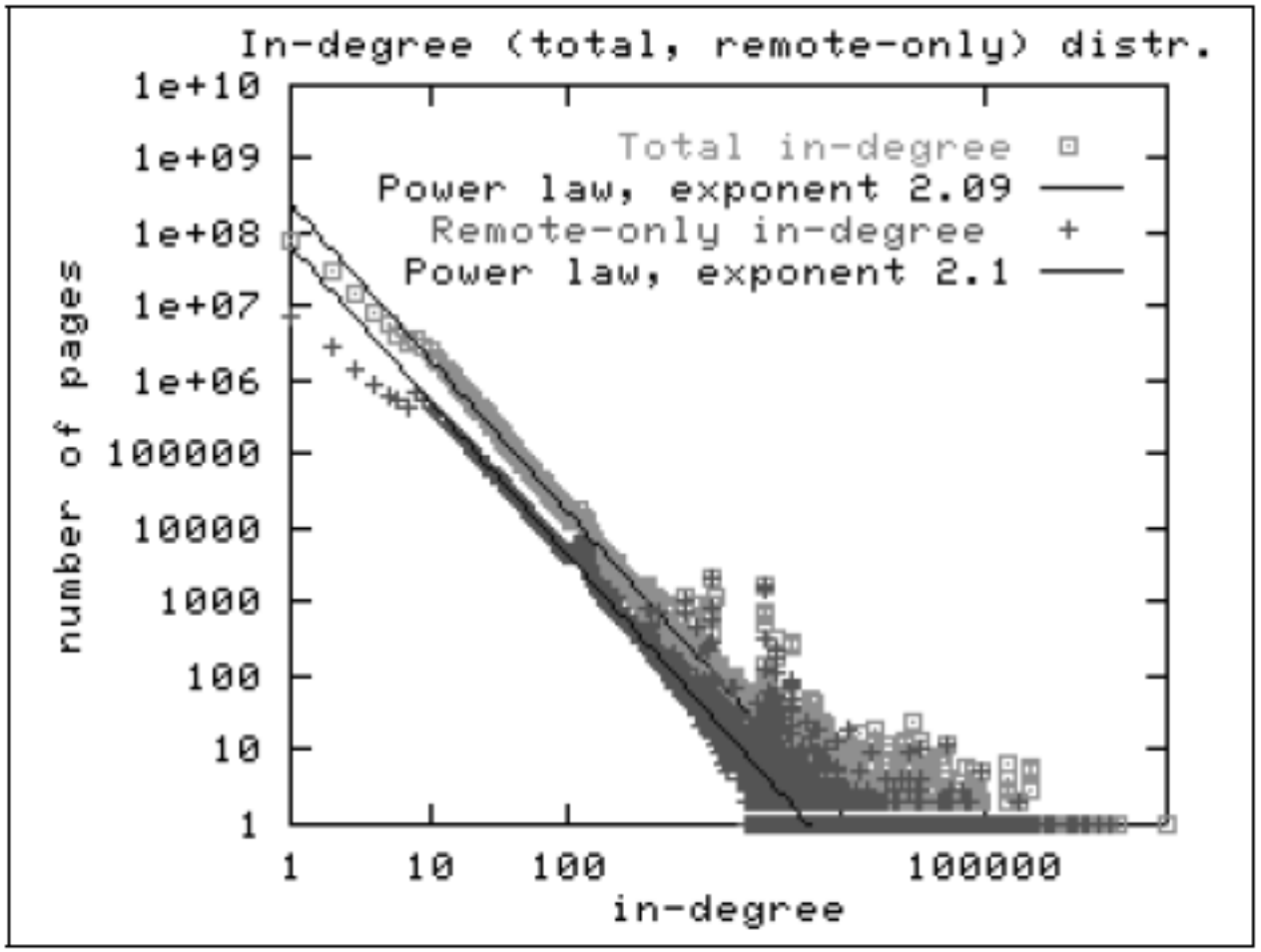


Figure 2: In-degree distribution.

Late 1990's
200M Webpages

Exponential ?

Power Law?

The Bowtie Structure of the Web

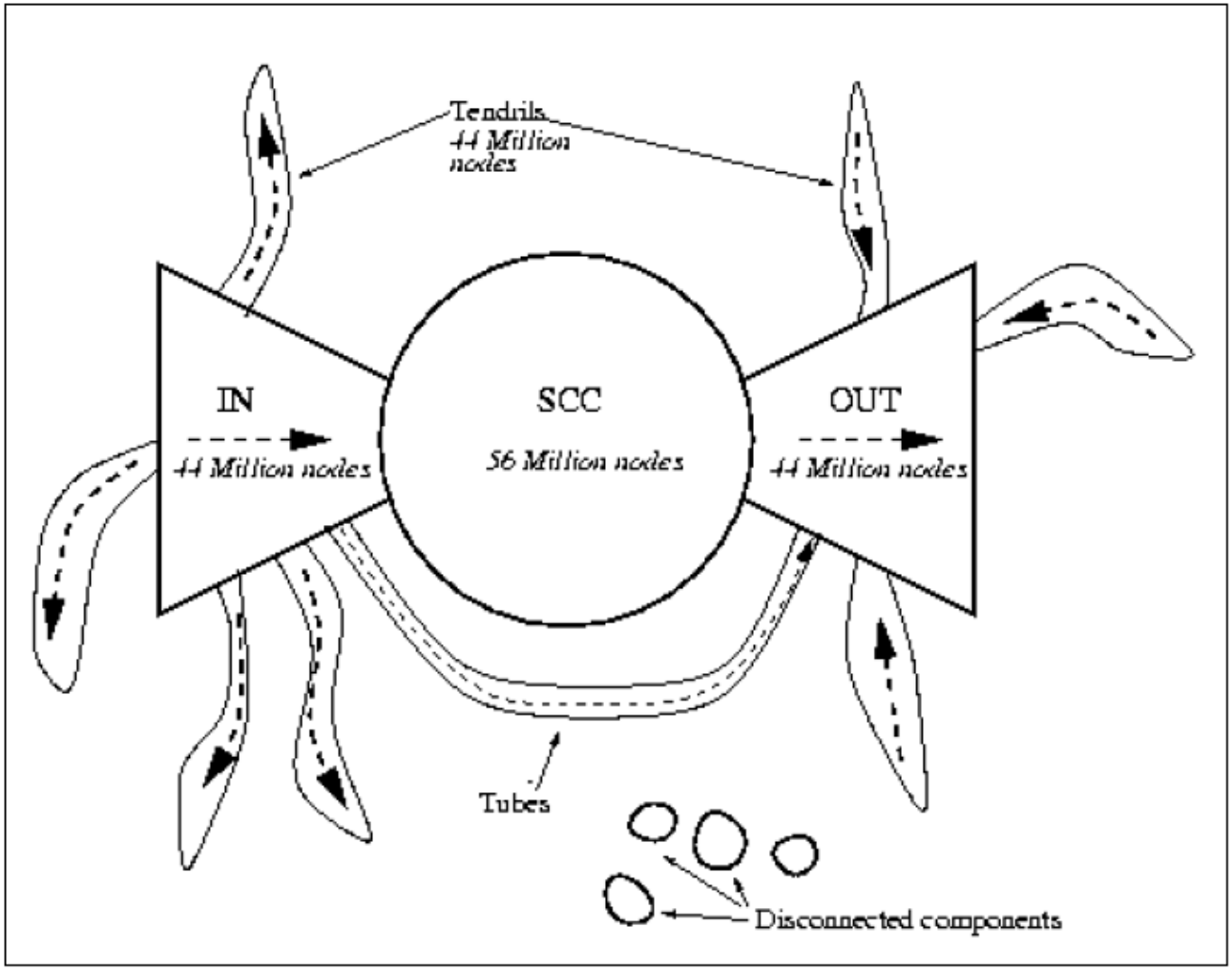


Figure 4: The web as a bowtie. SCC is a giant strongly connected component. IN consists of pages with paths to SCC, but no path from SCC. OUT consists of pages with paths from SCC, but no path to SCC. TENDRILS consists of pages that cannot surf to SCC, and which cannot be reached by surfing from SCC.

users(name, age)
pages(user, url)

Hash Join in MapReduce

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Users by name, Pages by user;
```

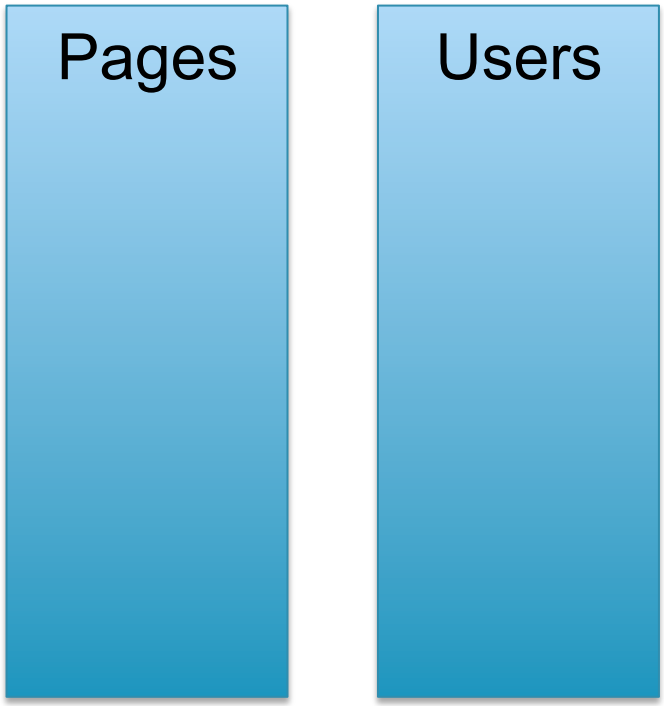
```
Map(String value):
    // value.relation is either 'Users' or 'Pages'
    if value.relation='Users':
      EmitIntermediate(value.name, (1, value));
    else
      EmitIntermediate(value.user, (2, value));
```

```
reduce(String k, Iterator values):
    Users = empty;  Pages = empty;
    for each v in values:
      if v.type = 1: Users.insert(v)
      else Pages.insert(v);
    for v1 in Users, for v2 in Pages
      Emit(v1,v2);
```

users(name, age)
pages(user, url)

Hash Join in Pig Latin

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Users by name, Pages by user;
```

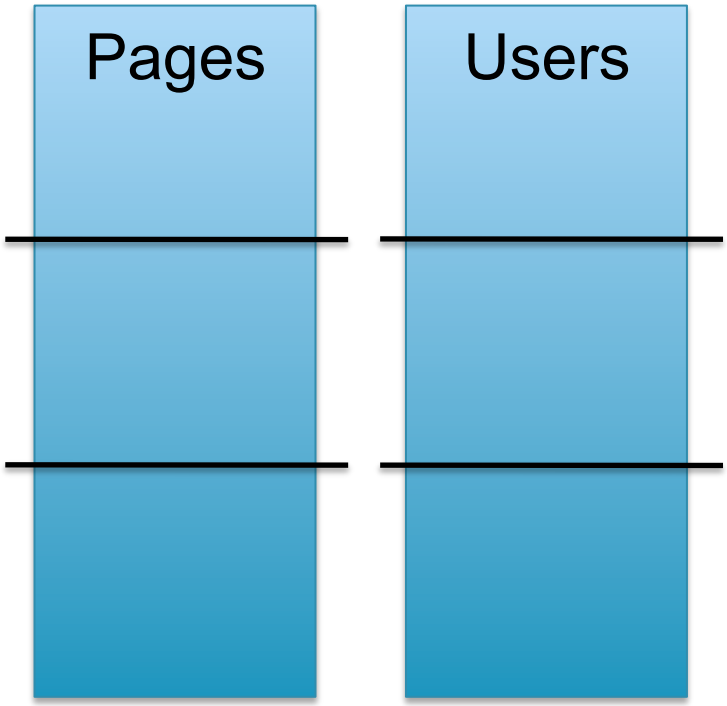


Credit: Alan Gates, Yahoo!

users(name, age)
pages(user, url)

Hash Join in Pig Latin

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Users by name, Pages by user;
```

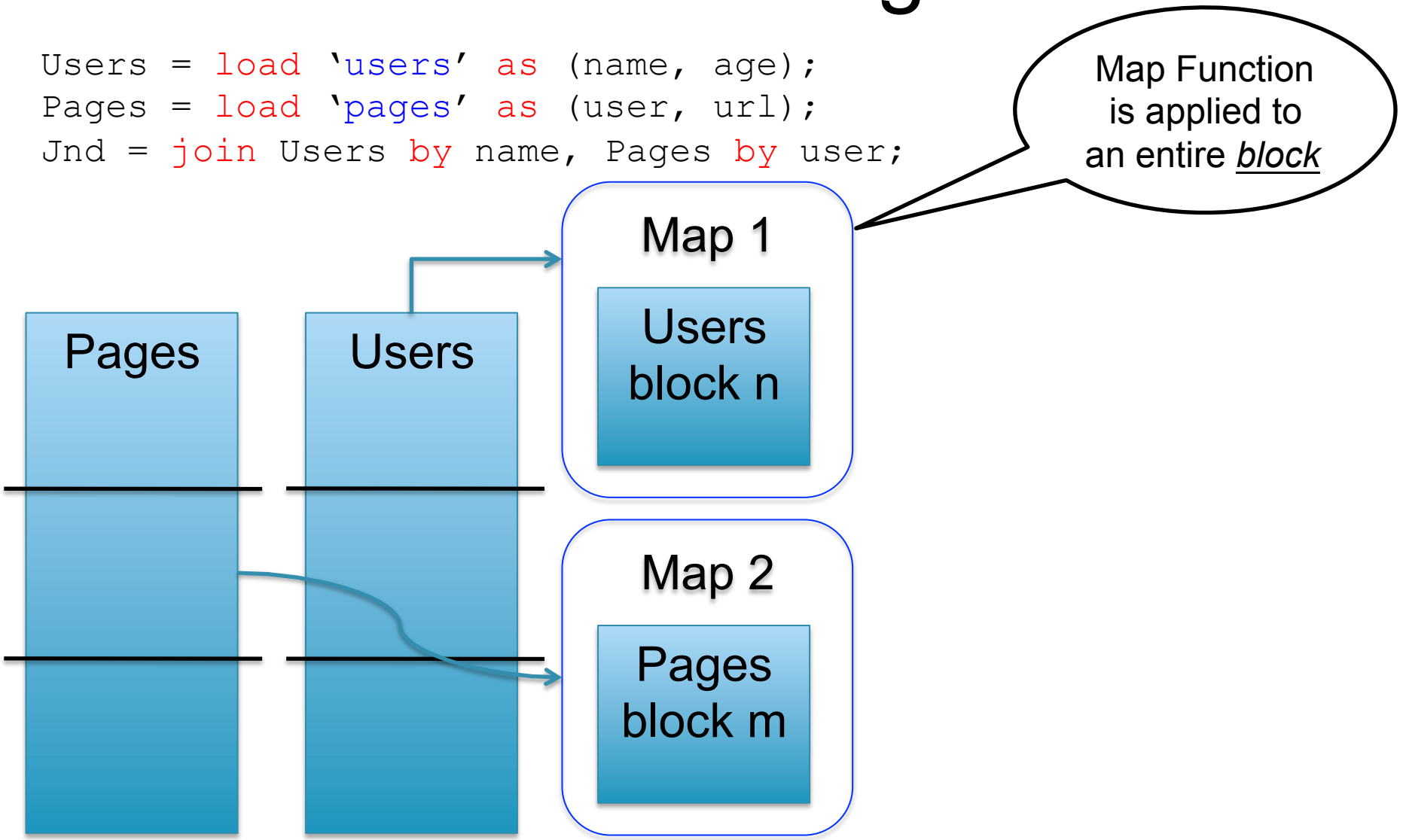


Credit: Alan Gates, Yahoo!

users(name, age)
pages(user, url)

Hash Join in Pig Latin

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Users by name, Pages by user;
```

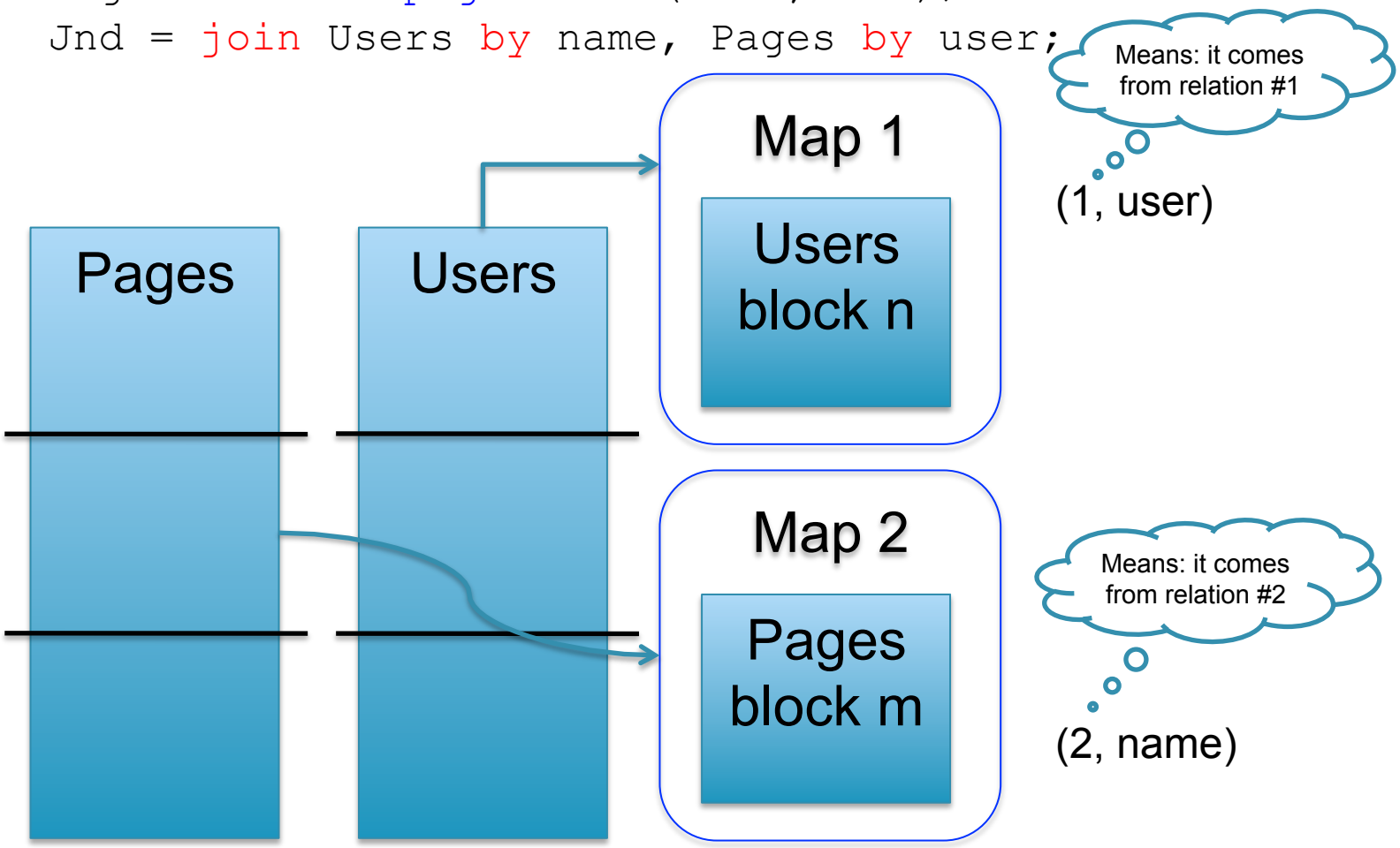


Credit: Alan Gates, Yahoo!

users(name, age)
pages(user, url)

Hash Join in Pig Latin

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Users by name, Pages by user;
```

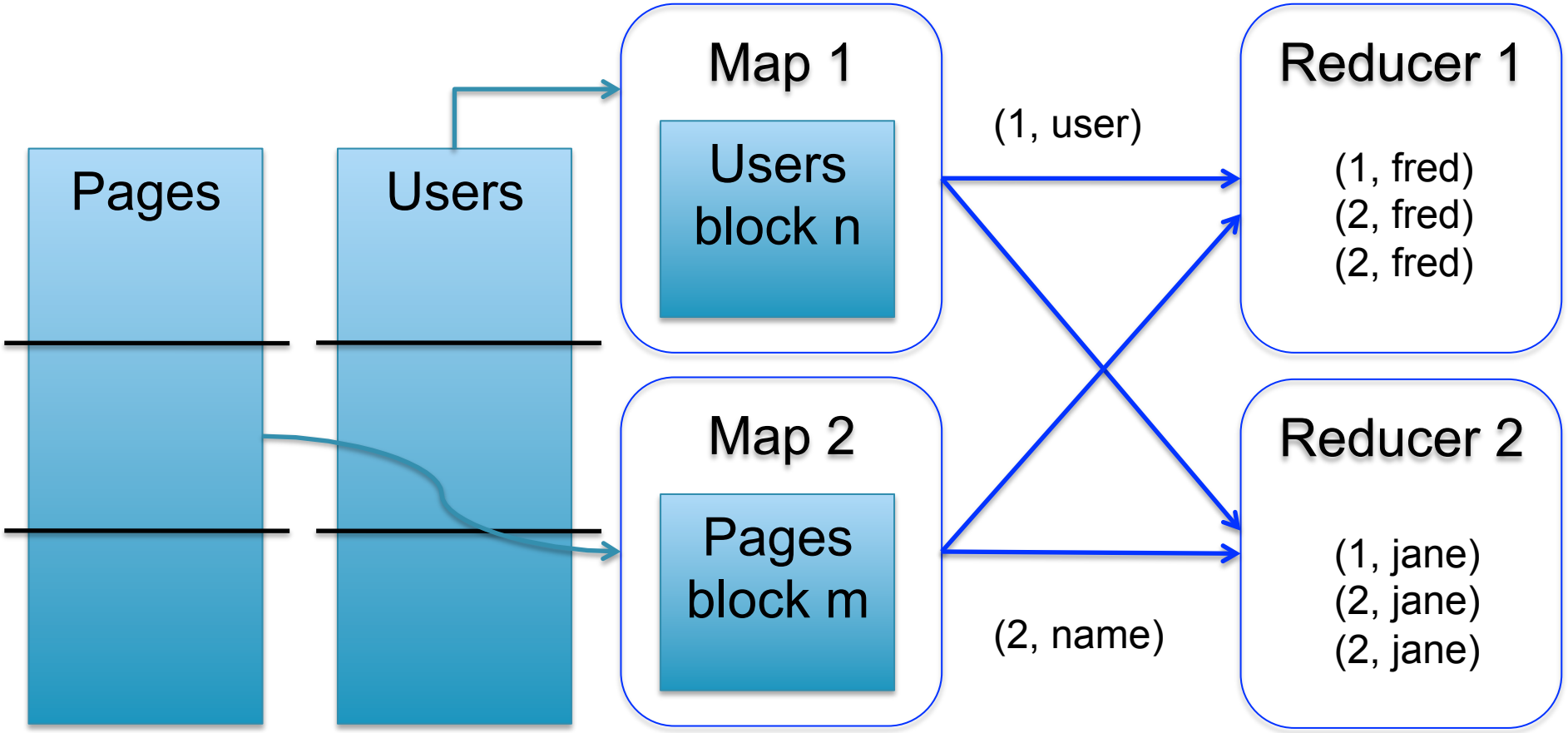


Credit: Alan Gates, Yahoo!

users(name, age)
pages(user, url)

Hash Join in Pig Latin

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Users by name, Pages by user;
```

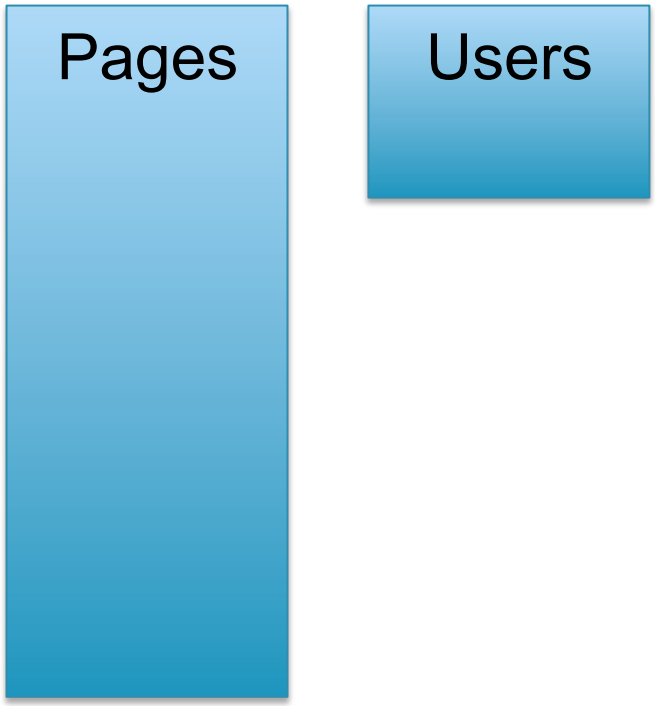


Credit: Alan Gates, Yahoo!

users(name, age)
pages(user, url)

Broadcast Join

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Pages by user, Users by name using "replicated";
```

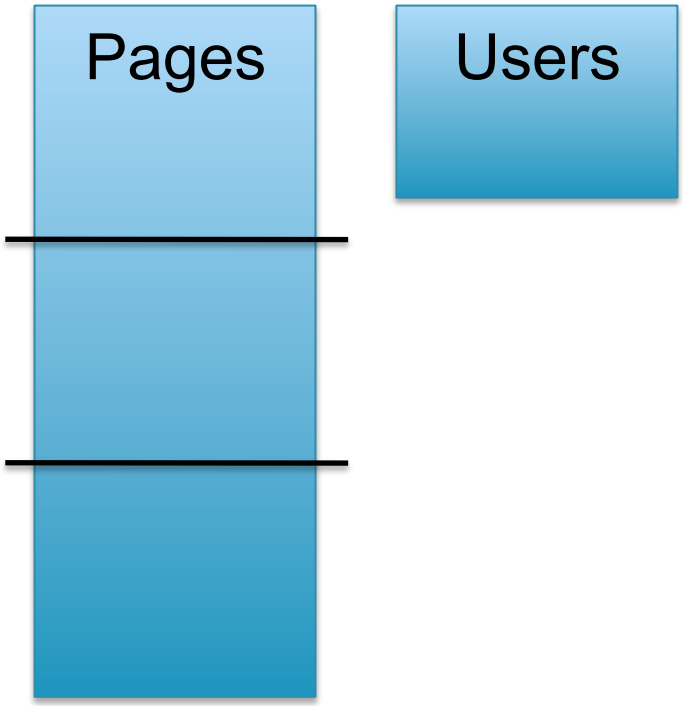


Credit: Alan Gates, Yahoo!

users(name, age)
pages(user, url)

Broadcast Join

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Pages by user, Users by name using "replicated";
```

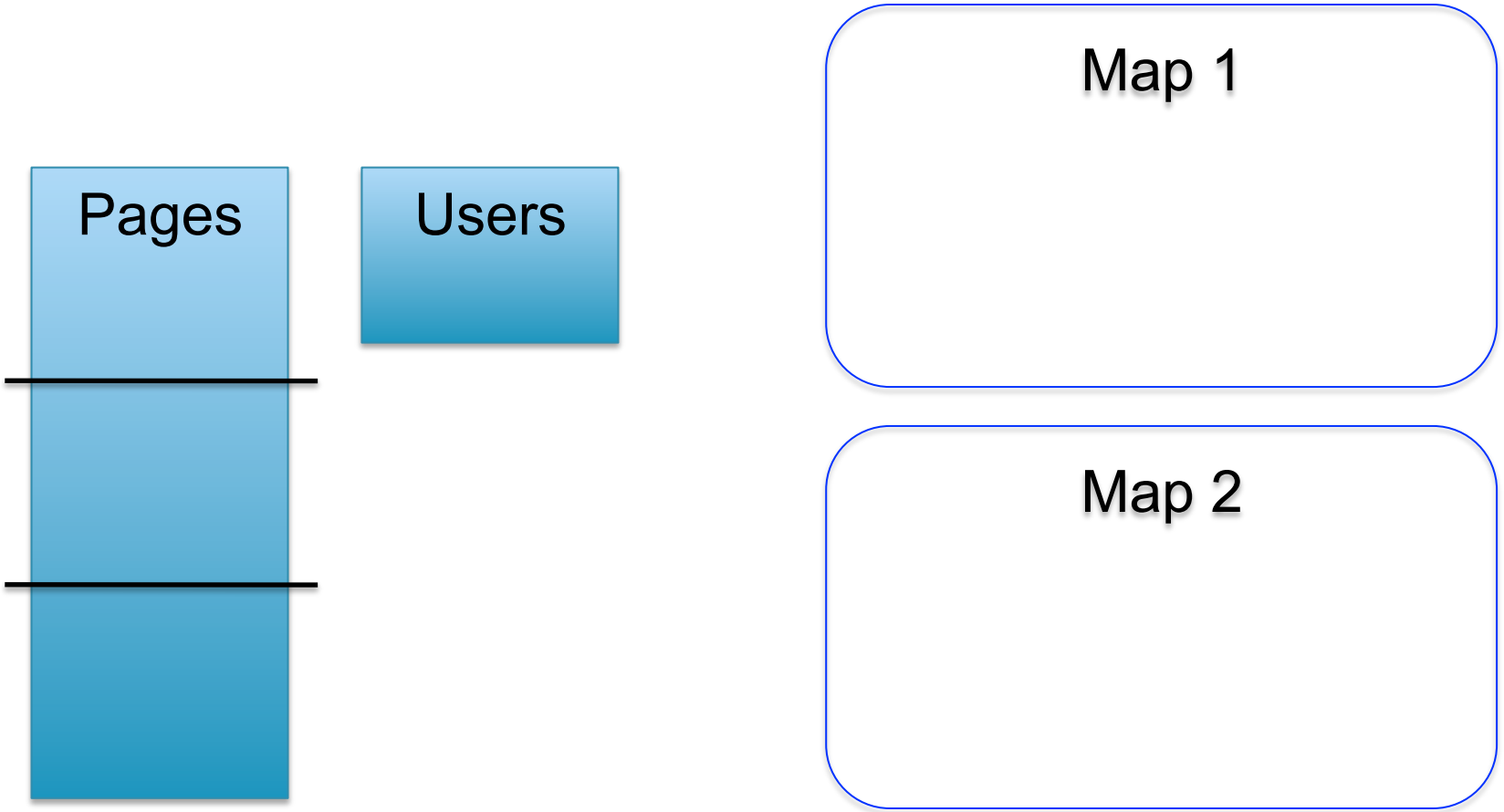


Credit: Alan Gates, Yahoo!

users(name, age)
pages(user, url)

Broadcast Join

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Pages by user, Users by name using "replicated";
```

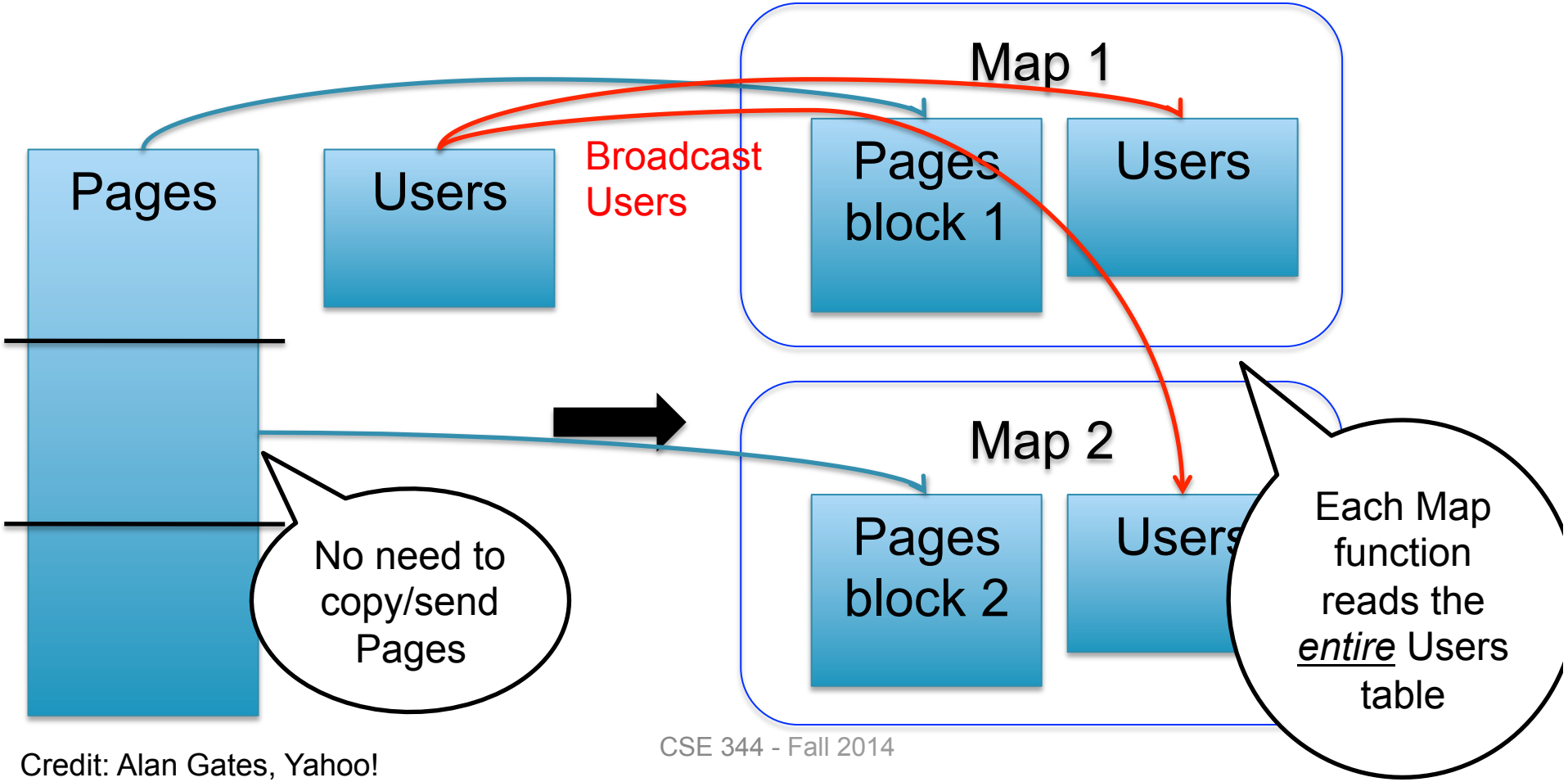


Credit: Alan Gates, Yahoo!

users(name, age)
pages(user, url)

Broadcast Join

```
Users = load 'users' as (name, age);
Pages = load 'pages' as (user, url);
Jnd = join Pages by user, Users by name using "replicated";
```



Credit: Alan Gates, Yahoo!

Matrix Multiplication v.s. Join

Dense matrices:

$$\begin{bmatrix} 6 & 6 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 3 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

```
forall i,k do
    C[i,k] = Σ_j A[i,j] * B[j,k]
```

Matrix Multiplication v.s. Join

Dense matrices:

$$\begin{bmatrix} 6 & 6 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 3 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

```
forall i,k do
    C[i,k] = Σ_j A[i,j] * B[j,k]
```

Sparse matrices as relations:

B(j,k,v)

j	k	v
1	1	1
1	3	3
2	2	1
3	1	2

A(i,j,v)

i	j	v
1	2	3
1	3	3
2	1	1
3	1	2

```
SELECT A.i, B.k, sum(A.v*B.v)
FROM A, B
WHERE A.j=B.j
GROUP BY A.i,B.i
```

Matrix Multiplication v.s. Join

Dense matrices:

$$\begin{bmatrix} 6 & 6 & 0 \\ 1 & 0 & 0 \\ 2 & 0 & 6 \end{bmatrix} = \begin{bmatrix} 0 & 3 & 3 \\ 1 & 0 & 0 \\ 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 2 & 0 \\ 2 & 0 & 0 \end{bmatrix}$$

Sparse matrices as relations:

B(j,k,v)

j	k	v
1	1	1
1	3	3
2	2	1
3	1	2

A(i,j,v)

i	j	v
1	2	3
1	3	3
2	1	1
3	1	2

forall i,k do
$C[i,k] = \Sigma_j A[i,j] * B[j,k]$

```
SELECT A.i, B.k, sum(A.v*B.v)
FROM A, B
WHERE A.j=B.j
GROUP BY A.i,B.i
```

Matrix multiplication = a join + a group by

Parallel DBs v.s. MapReduce

Parallel DB

- Plusses
  - Efficient binary format
  - Indexes, physical tuning
  - Cost-based optimization
- Minuses
  - Difficult to import data
  - Lots of baggage: logging, transactions

MapReduce

- Minuses
  - Lots of time spent parsing!
  - Text files
  - “Optimizers is between your eyes and your keyboard”
- Plusses
  - Any data
  - Lightweight, easy to speedup
  - Arguably more scalable

Example: Parallel DBMS vs. MR

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.

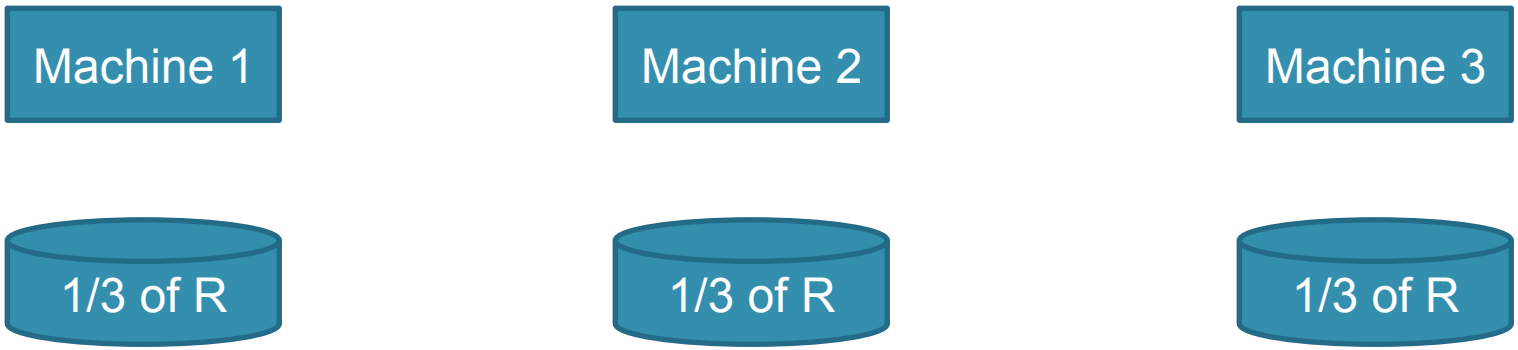
The tuples are randomly organized across machines (i.e., R is **block partitioned** across machines).

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)

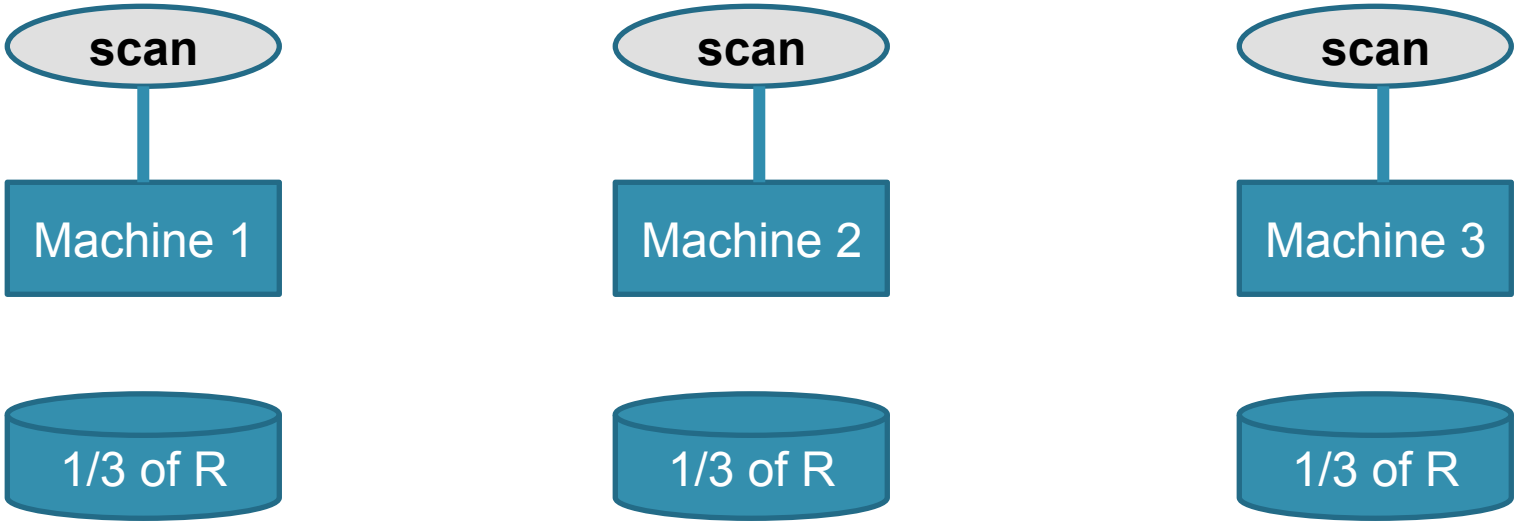
```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```



R(a, b)

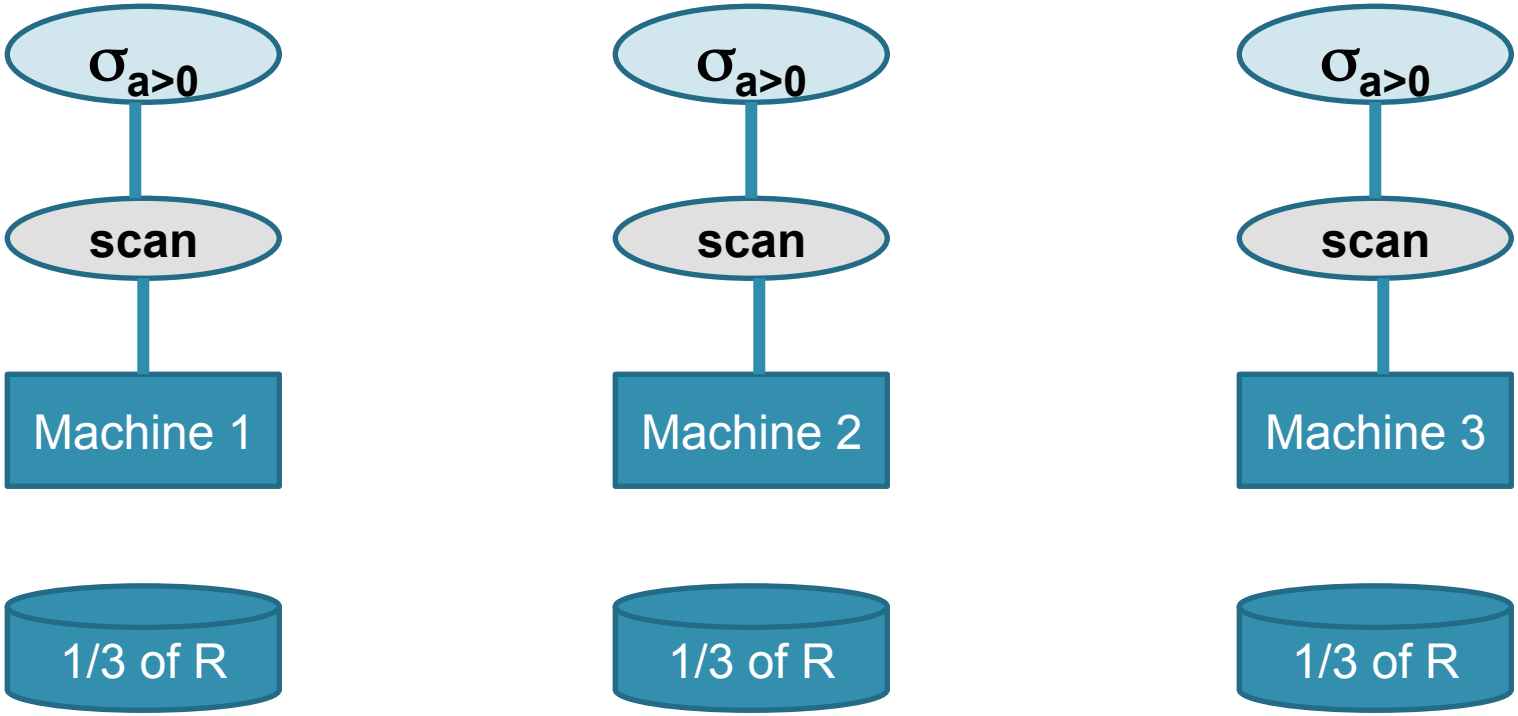
```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```

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



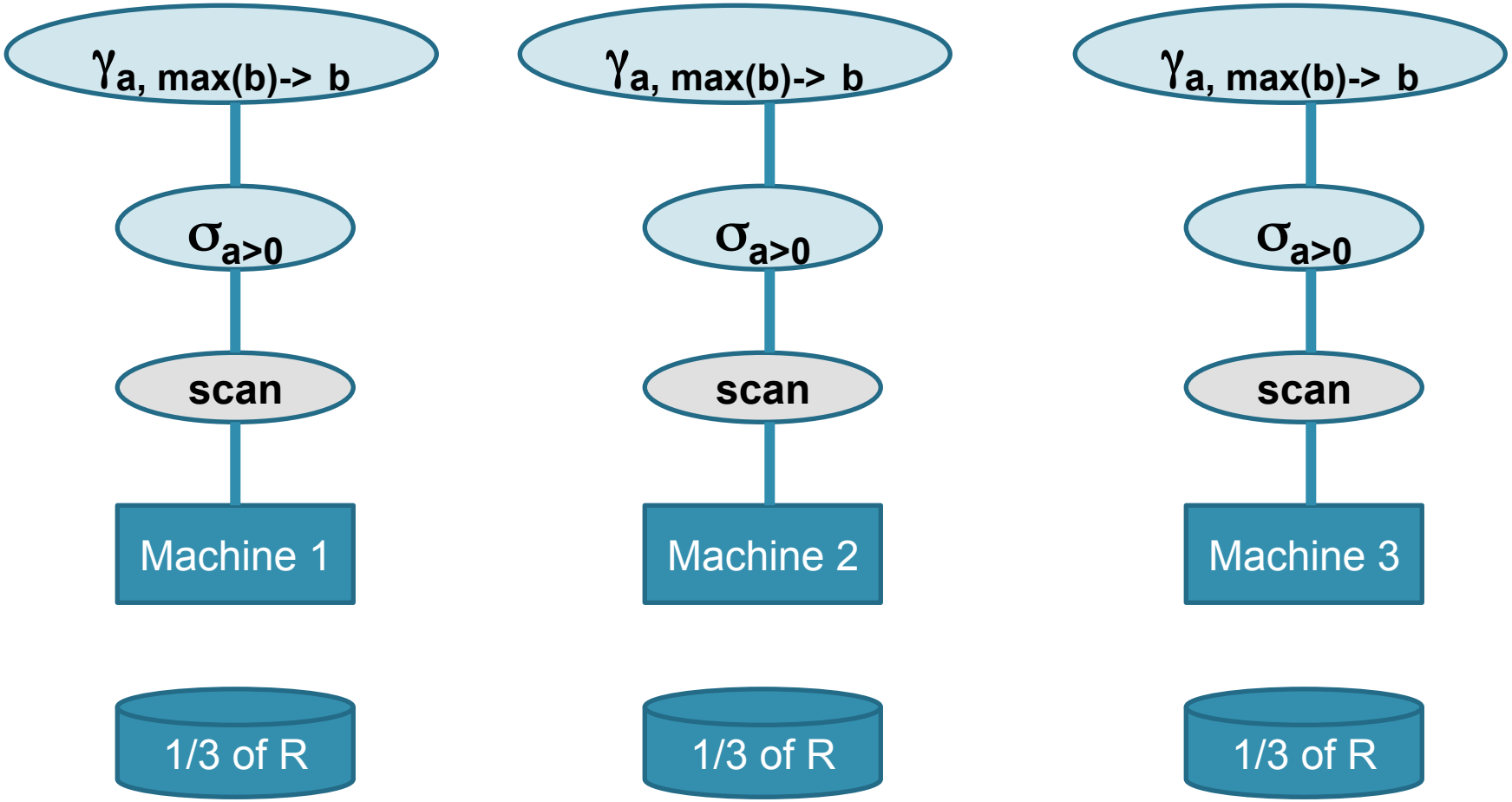
R(a, b)

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



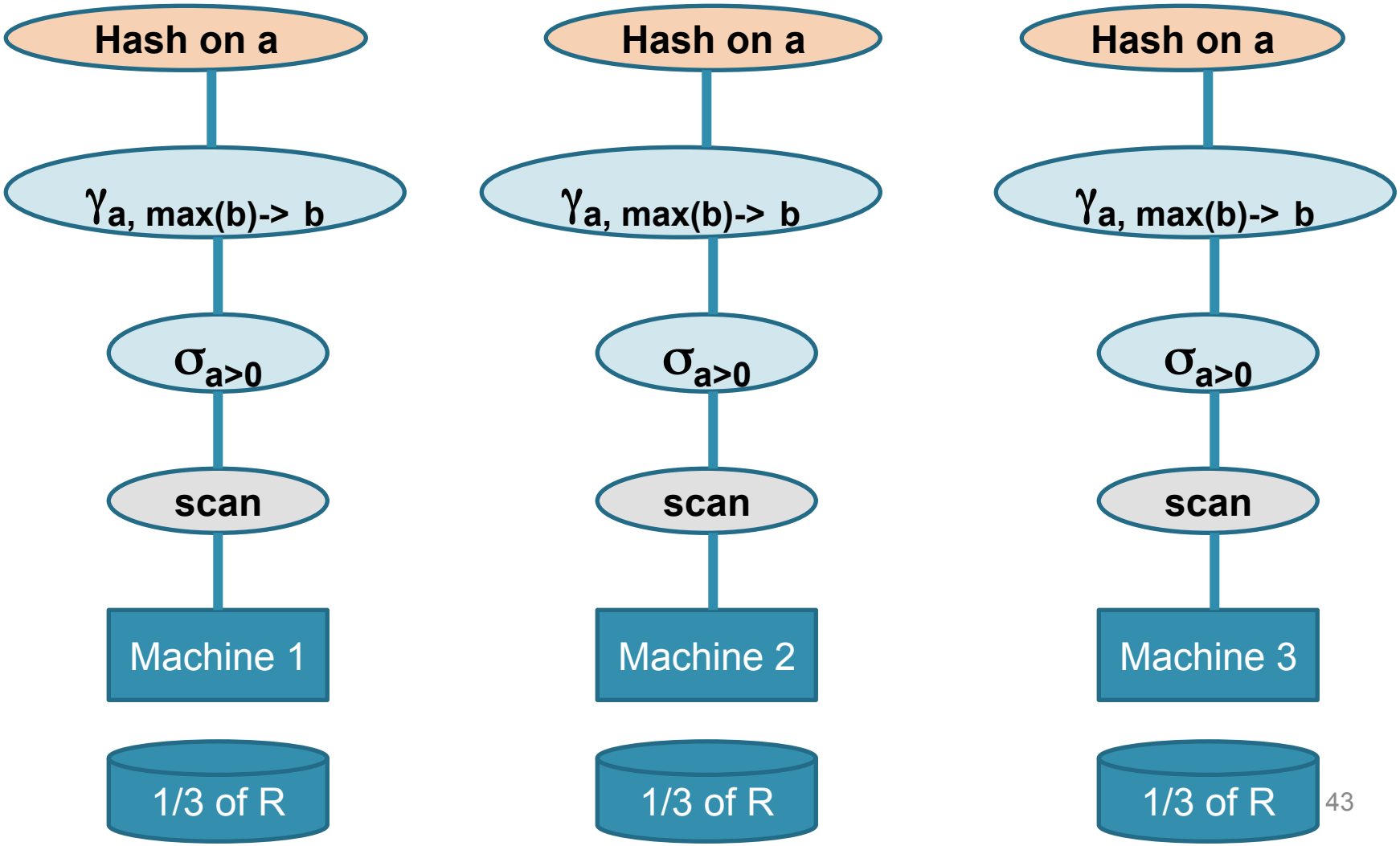
R(a, b)

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



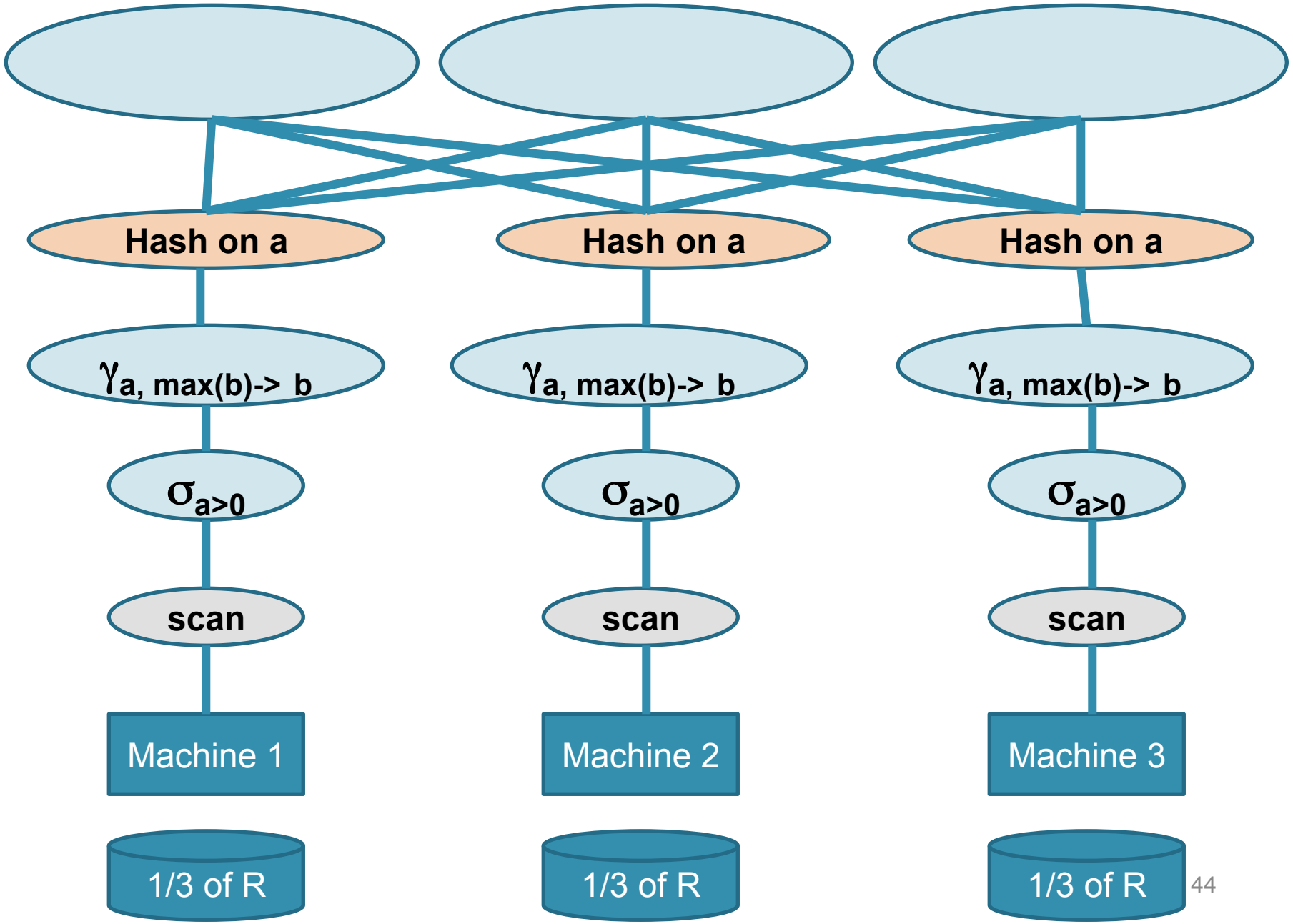
R(a, b)

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



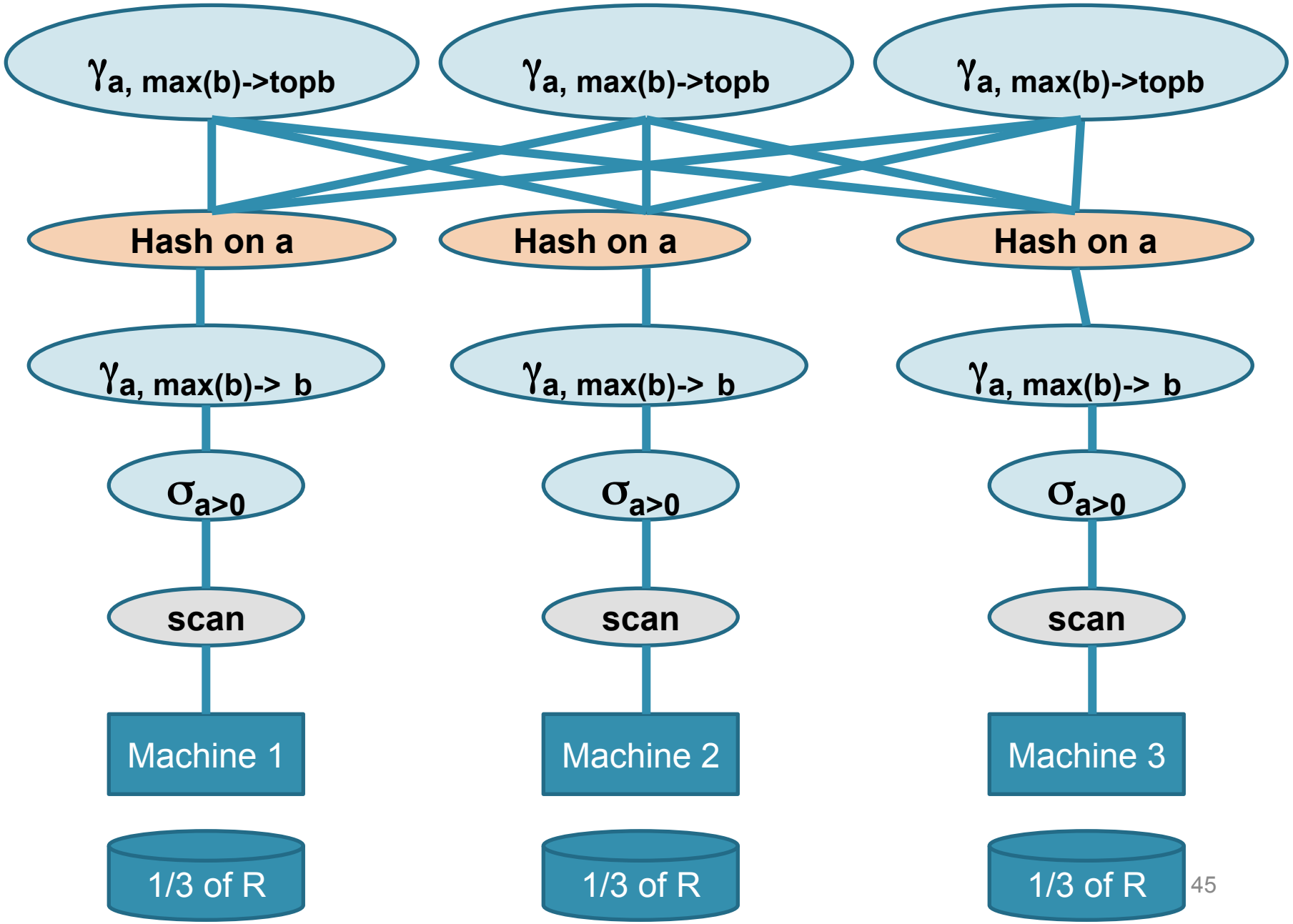
R(a, b)

SELECT a, max(b) as topb
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



1b. Map Reduce

Explain how the query will be executed in MapReduce (not PIG)

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

Map

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

- 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'

Shuffle

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

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

Reduce

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

- 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

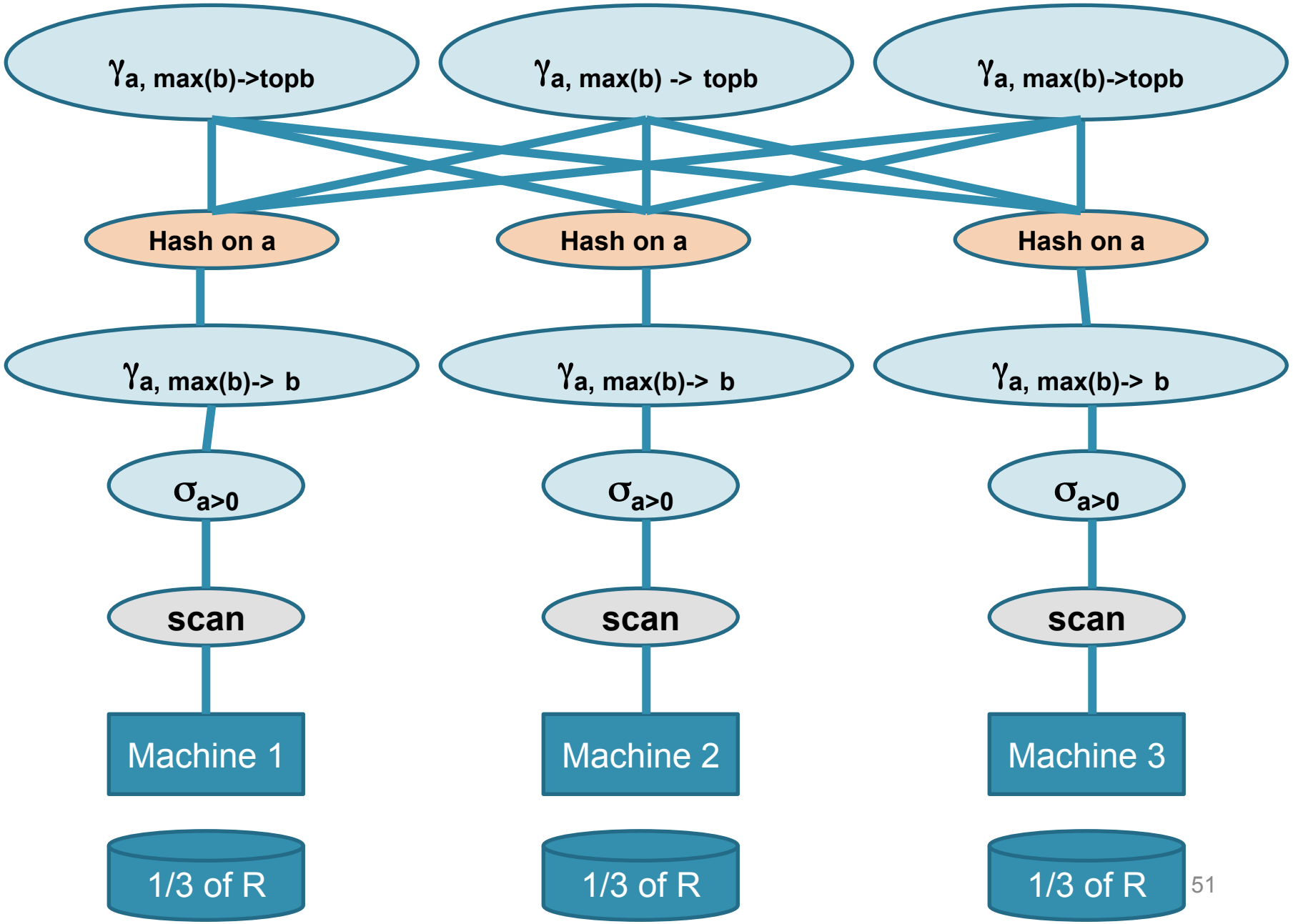
SELECT a, max(b) as topb
FROM R WHERE a > 0 GROUP BY a

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

Block partition

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



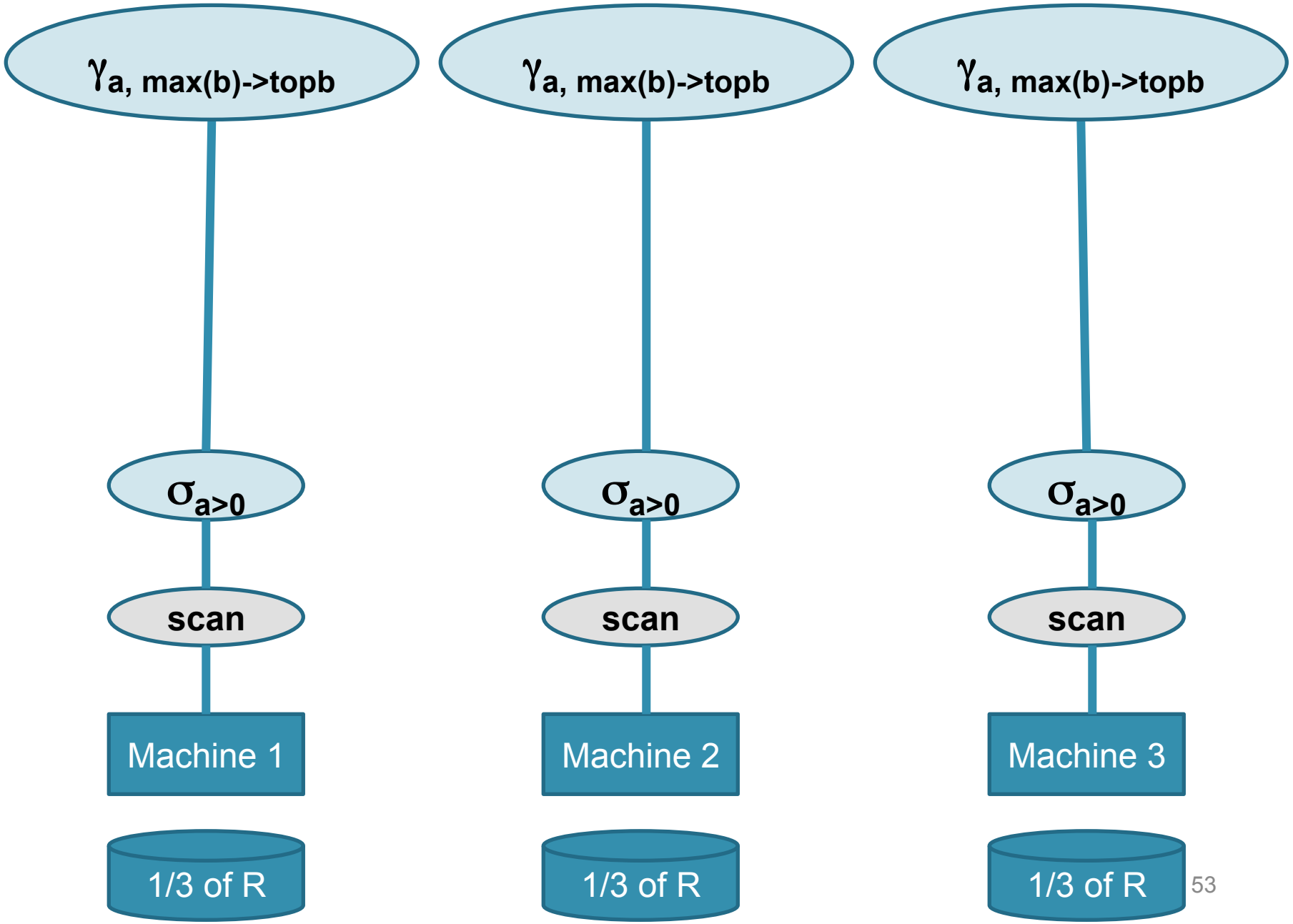
SELECT a, max(b) as topb
FROM R WHERE a > 0 GROUP BY a

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
FROM R WHERE a > 0 GROUP BY a



SELECT a, max(b) as topb
FROM R WHERE a > 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 not have to do anything (if one map task/ node) or less computation (multiple map tasks/node)