Introduction to Data Management CSE 344

Lecture 25

Parallel Databases Wrap-up

Announcements

- HW8 due on Thursday (1 late day allowed)
  - Prob#4 may take > 4 hours just to run
  - Check out the posts (lectures + HW8) on discussion board

- Let us know if you have questions on graded hw4-hw6.

- Review session today on BCNF / ER diagram – Monday 3/10, CSE 303, by Vaspol, 4:30-5:30 pm

- Final exam: Next Tuesday

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)

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

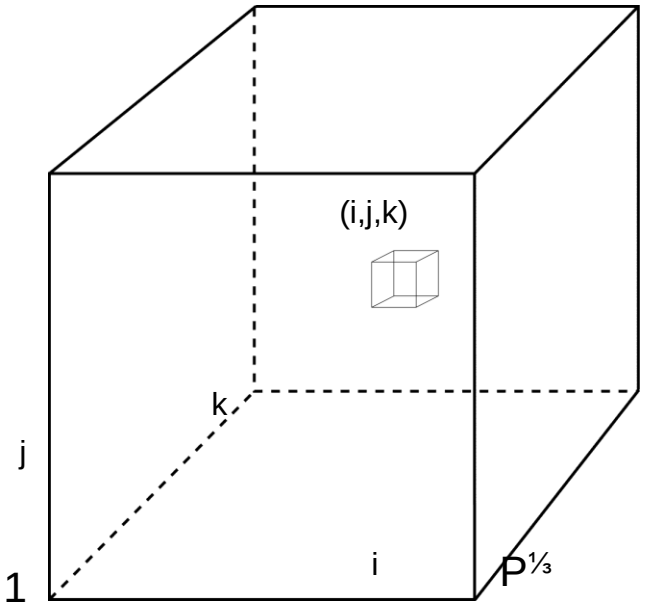
- Have P servers (say P=27 or P=1000)
- How do we compute this query?
  Q(x,y,z) = R(x,y),S(y,z),T(z,x)
- Step 1:
  - Each server sends R(x,y) to server **h(y) mod P**
  - Each server sends S(y,z) to server **h(y) mod P**
- Step 2:
  - Each server computes R⋈S locally
  - Each server sends [R(x,y),S(y,z)]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

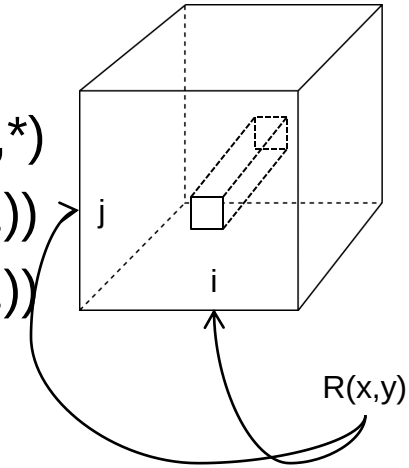
A Challenge

- 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≤$P^{1/3}$



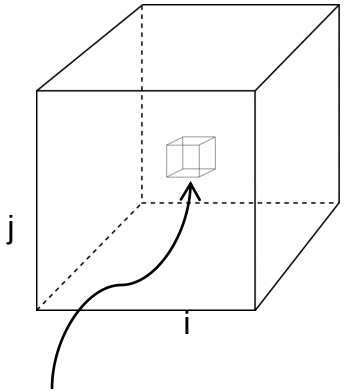
A Challenge

- 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))



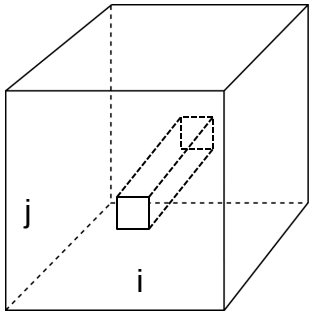
A Challenge

- 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))



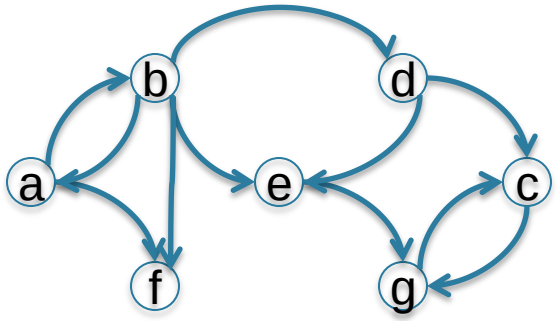
- Final output:
  - Each server (i,j,k) computes the query R(x,y),S(y,z),T(z,x) locally
- Analysis: each tuple R(x,y) is replicated at most $P^{1/3}$ times

Graph Analysis in HW8

Graph Databases

Many large databases are graphs

Examples



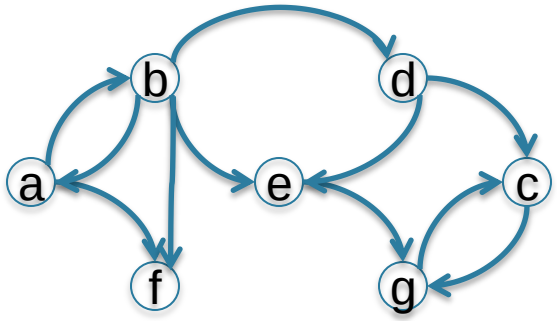
Source	Target
a	b
b	a
a	f
b	f
b	e
b	d
d	e
d	c
e	g
g	c
c	g

Graph Databases

Many large databases are graphs

Examples

- The Web
- The Internet
- Social Networks
- Flights between airports
- Etc.

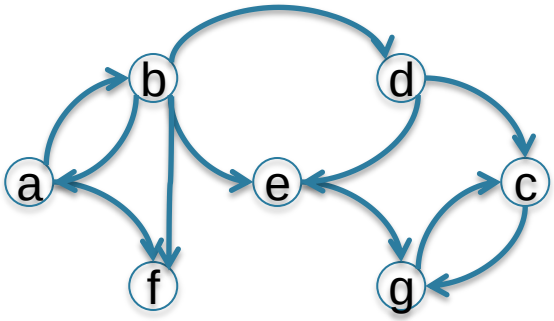


Source	Target
a	b
b	a
a	f
b	f
b	e
b	d
d	e
d	c
e	g
g	c
c	g

Data Analytics on Big Graphs

Queries expressible in SQL:

- How many nodes (edges)?
- How many nodes have > 4 neighbors?
- Which nodes are connected as a triangle?



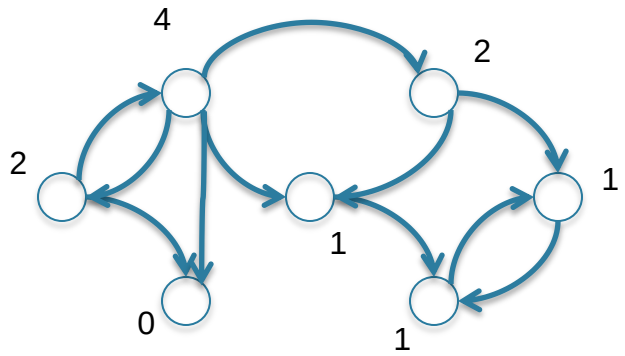
Source	Target
a	b
b	a
a	f
b	f
b	e
b	d
d	e
d	c
e	g
g	c
c	g

Queries requiring recursion:

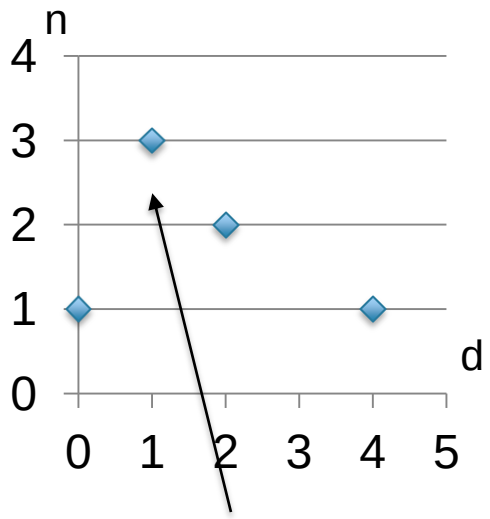
- Is the graph connected?
- What is the diameter of the graph?
- Compute *PageRank*
- Compute the *Centrality* of each node

Example: the Histogram of a Graph

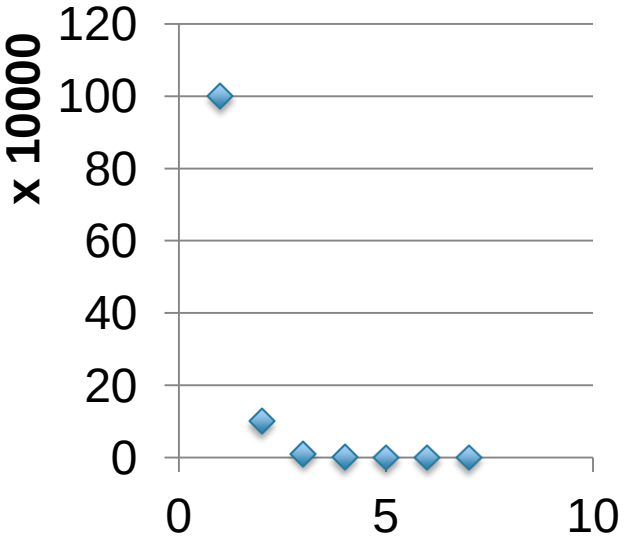
- Outdegree of a node = number of outgoing edges
- For each d, let n(d) = number of nodes with oudegree d
- The outdegree histogram of a graph = the scatterplot (d, n(d))

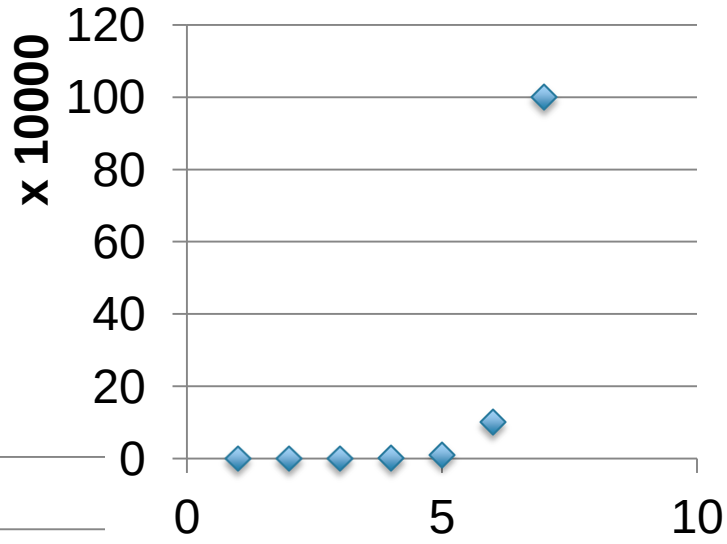


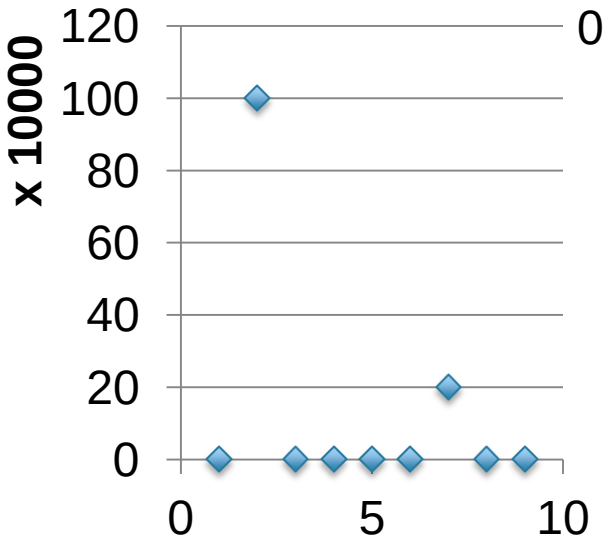
d	n(d)
0	1
1	3
2	2
3	0
4	1



Histograms Tell Us Something About the Graph





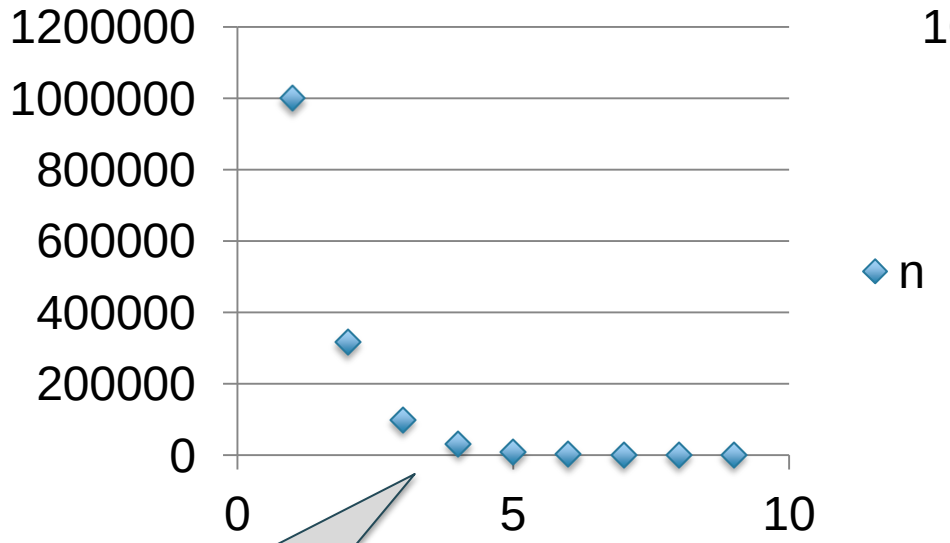


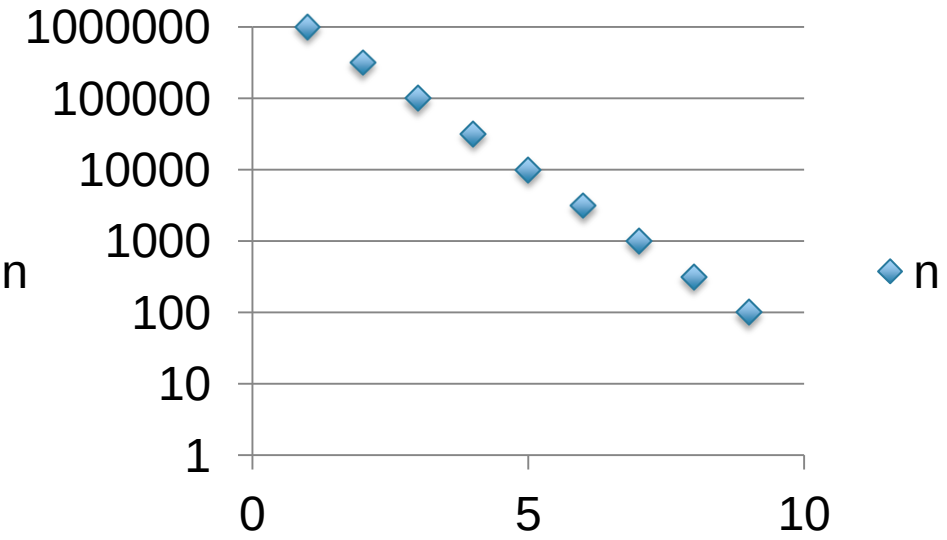
What can you say about these graphs?

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

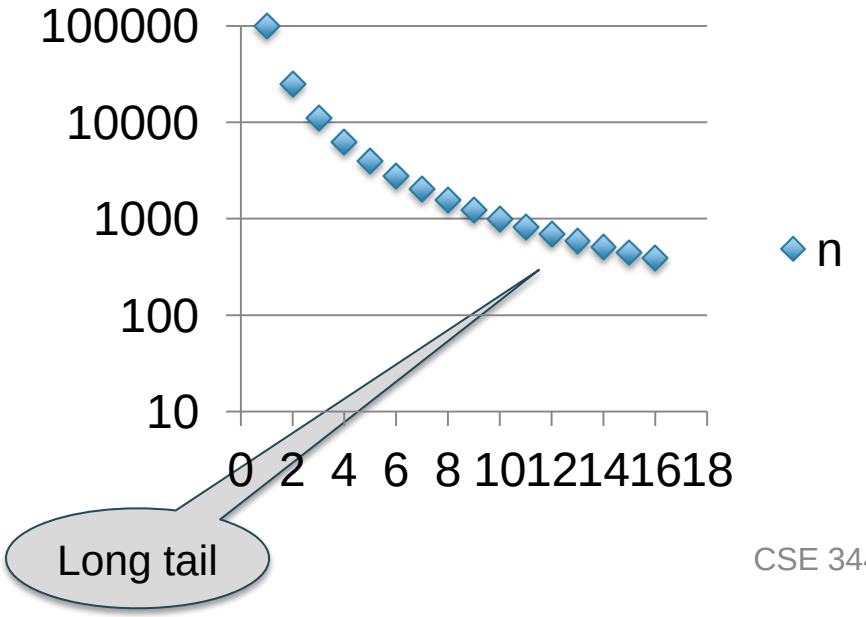


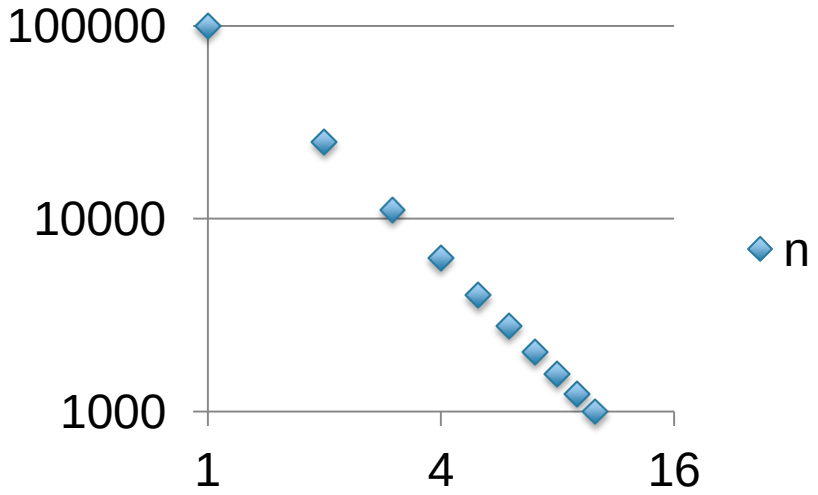


Quickly vanishing

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

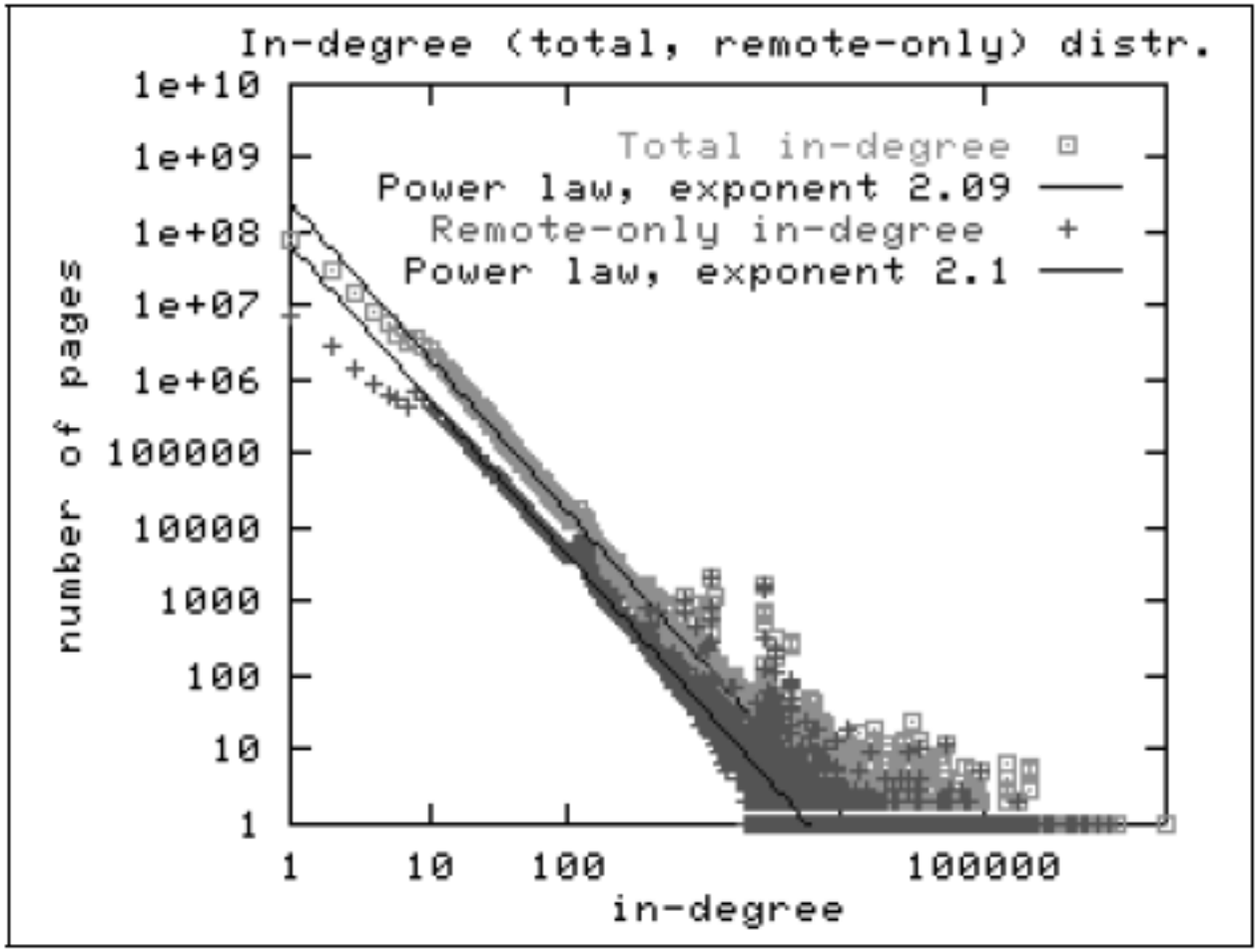


Figure 2: In-degree distribution.

Late 1990's
200M Webpages

Exponential ?

Power Law?

From PODS 2000 paper:
The Web as a graph by Ravi Kumar et . al.

The Bowtie Structure of the Web

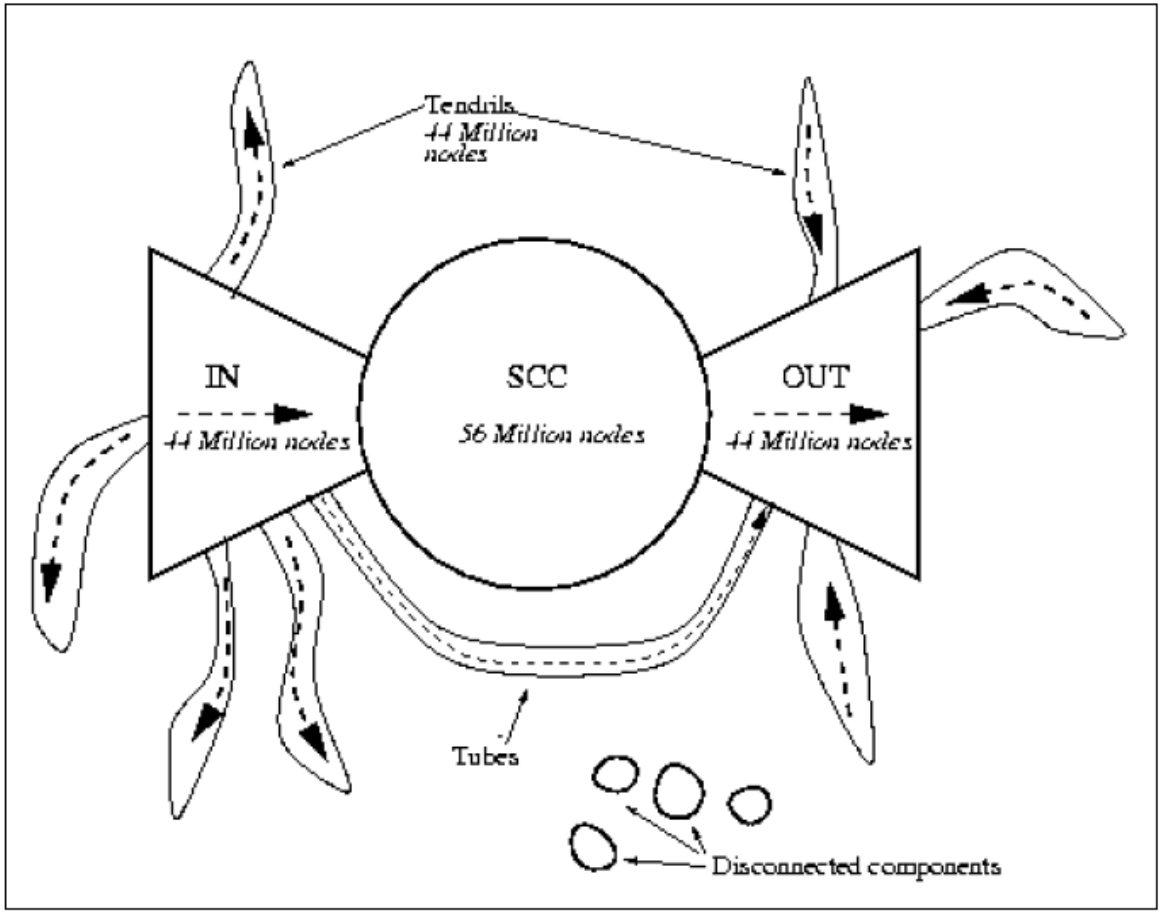


Figure 4: The web as a bowtie. SCC is a giant strongly connected component. IN consists of pages with paths to SCC, but no path from SCC. OUT consists of pages with paths from SCC, but no path to SCC. TENDRILS consists of pages that cannot surf to SCC, and which cannot be reached by surfing from SCC.

For about 75% of the webpages pairs u, v, no paths exist from u to v!

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

Hash Join in MapReduce

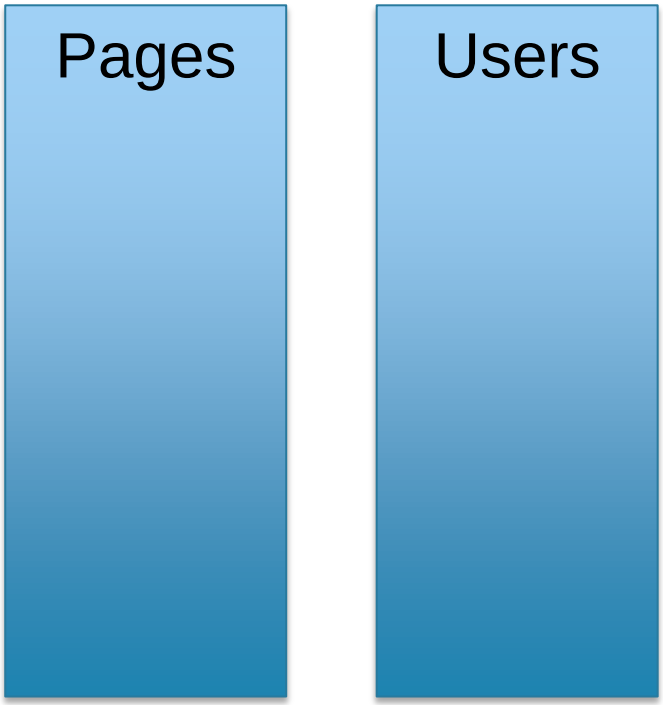
```
Map(String value):
    // value.relation is either 'Users' or 'Pages'
    if value.relation='Users':
     EmitIntermediate(value.name, (1, value));
    else
     EmitIntermediate(value.user, (2, value));
```

```
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;
```

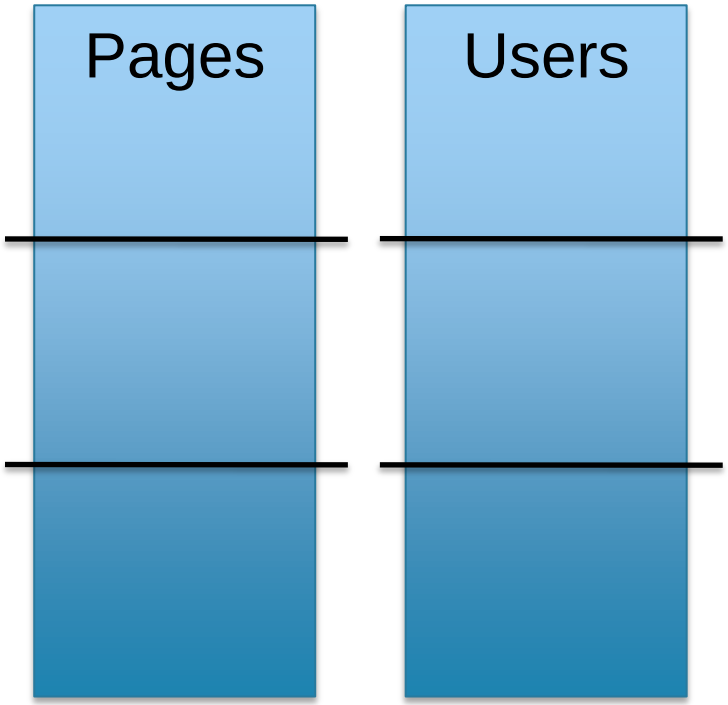


Credit: Alan Gates, Yahoo!

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;
```

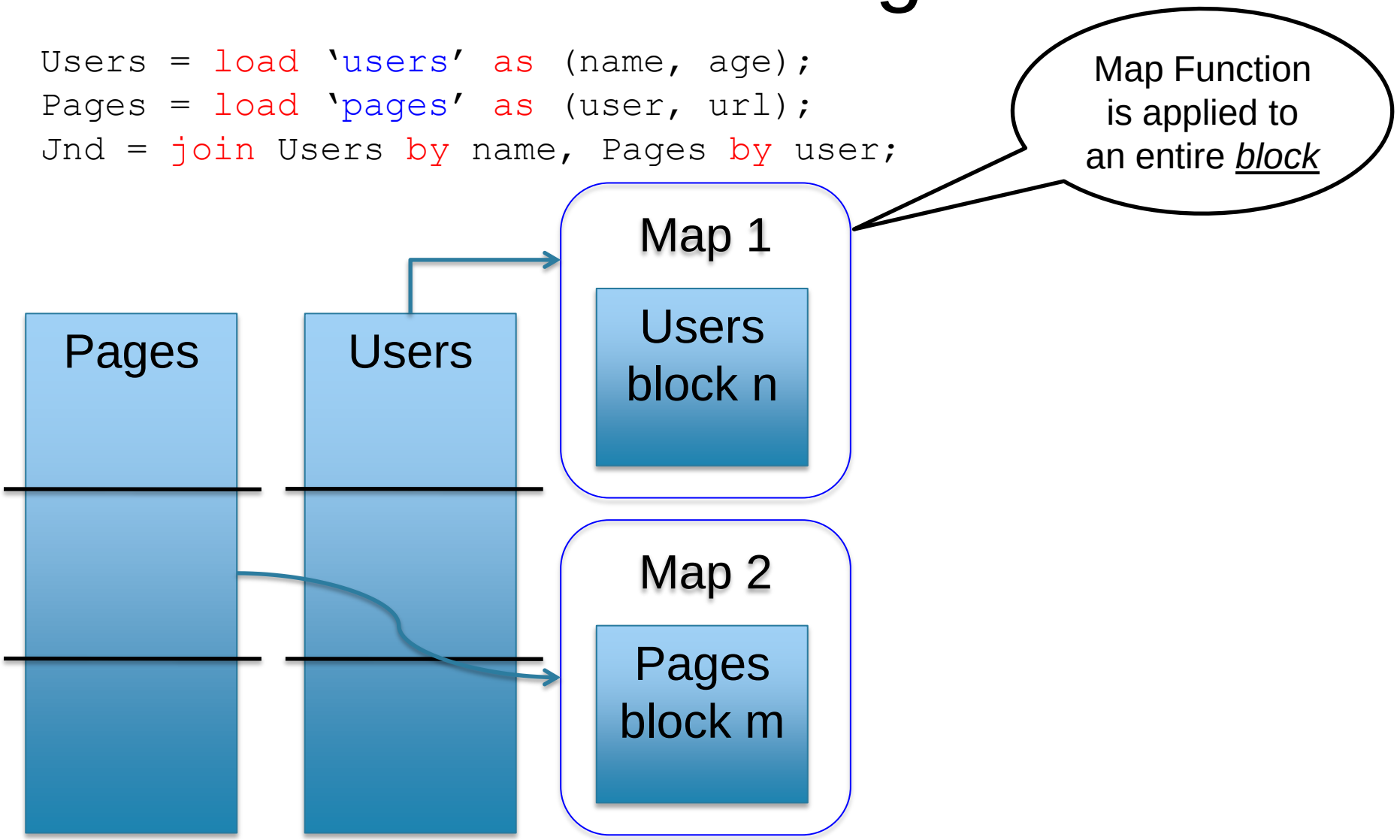


Credit: Alan Gates, Yahoo!

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;
```

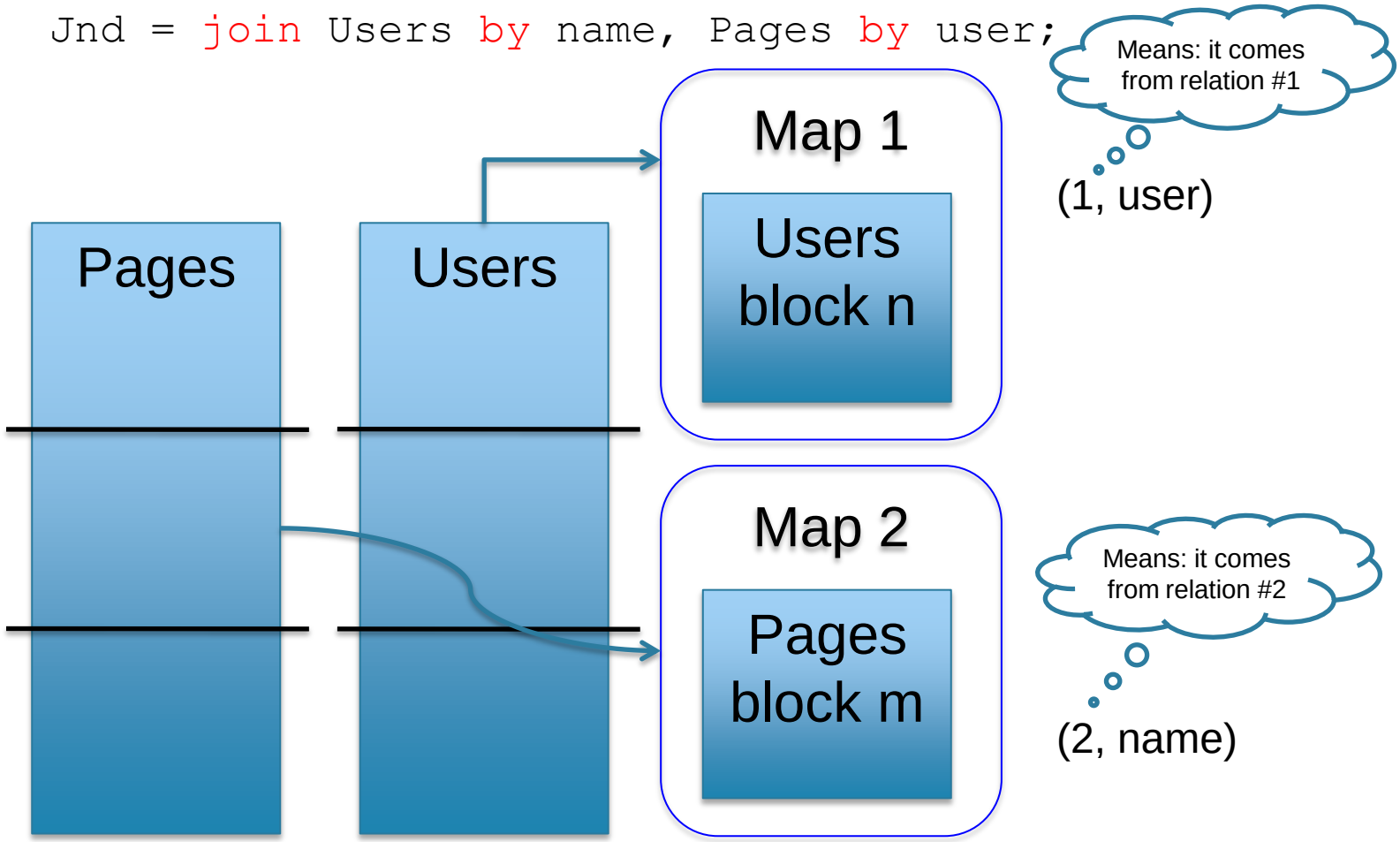


Credit: Alan Gates, Yahoo!

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;
```

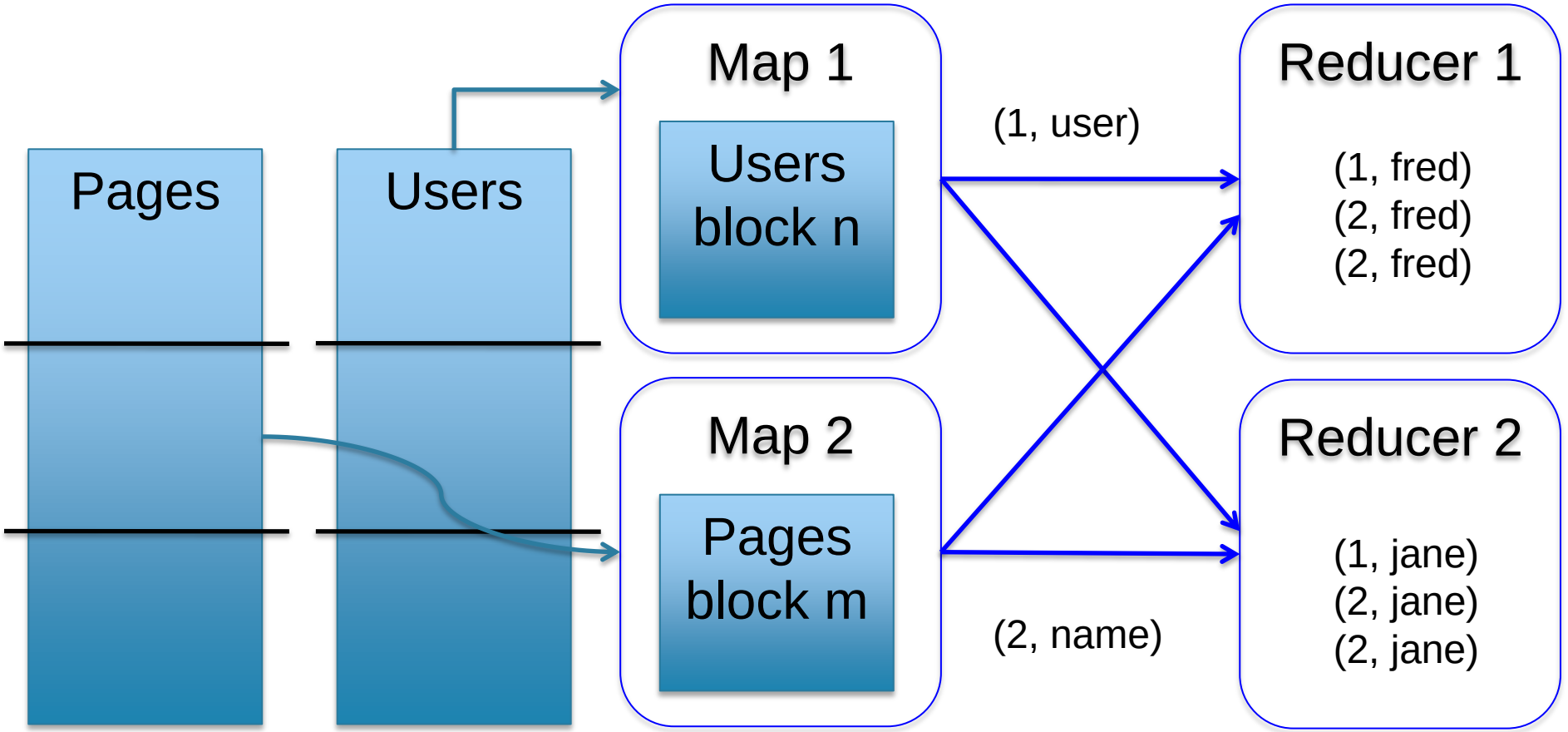


Credit: Alan Gates, Yahoo!

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;
```

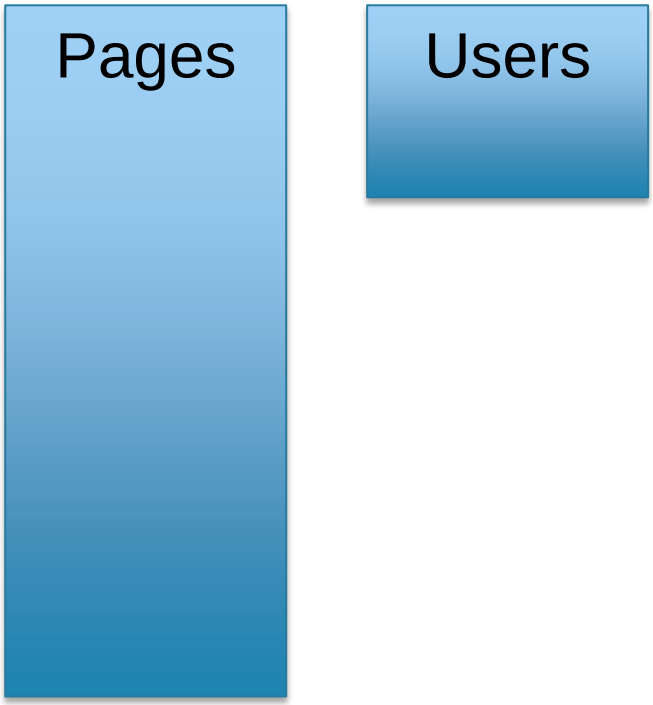


Credit: Alan Gates, Yahoo!

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";
```

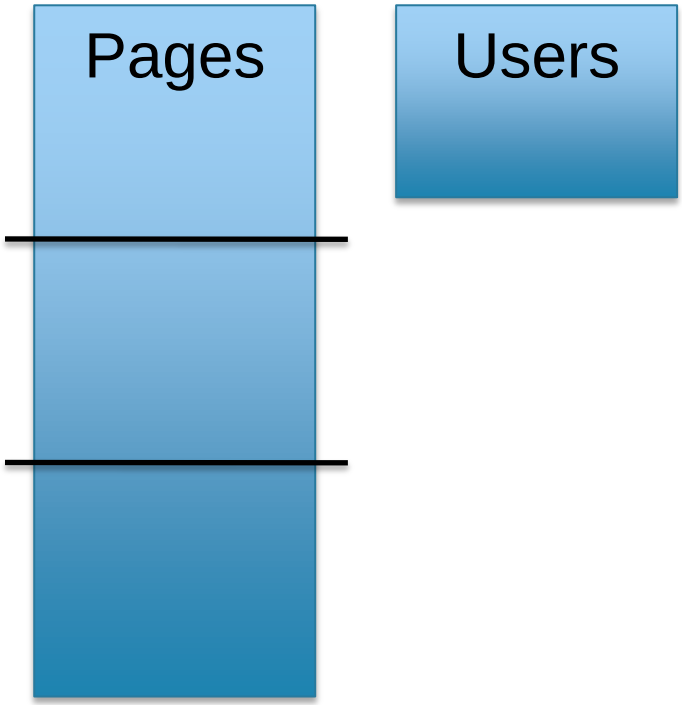


Credit: Alan Gates, Yahoo!

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";
```

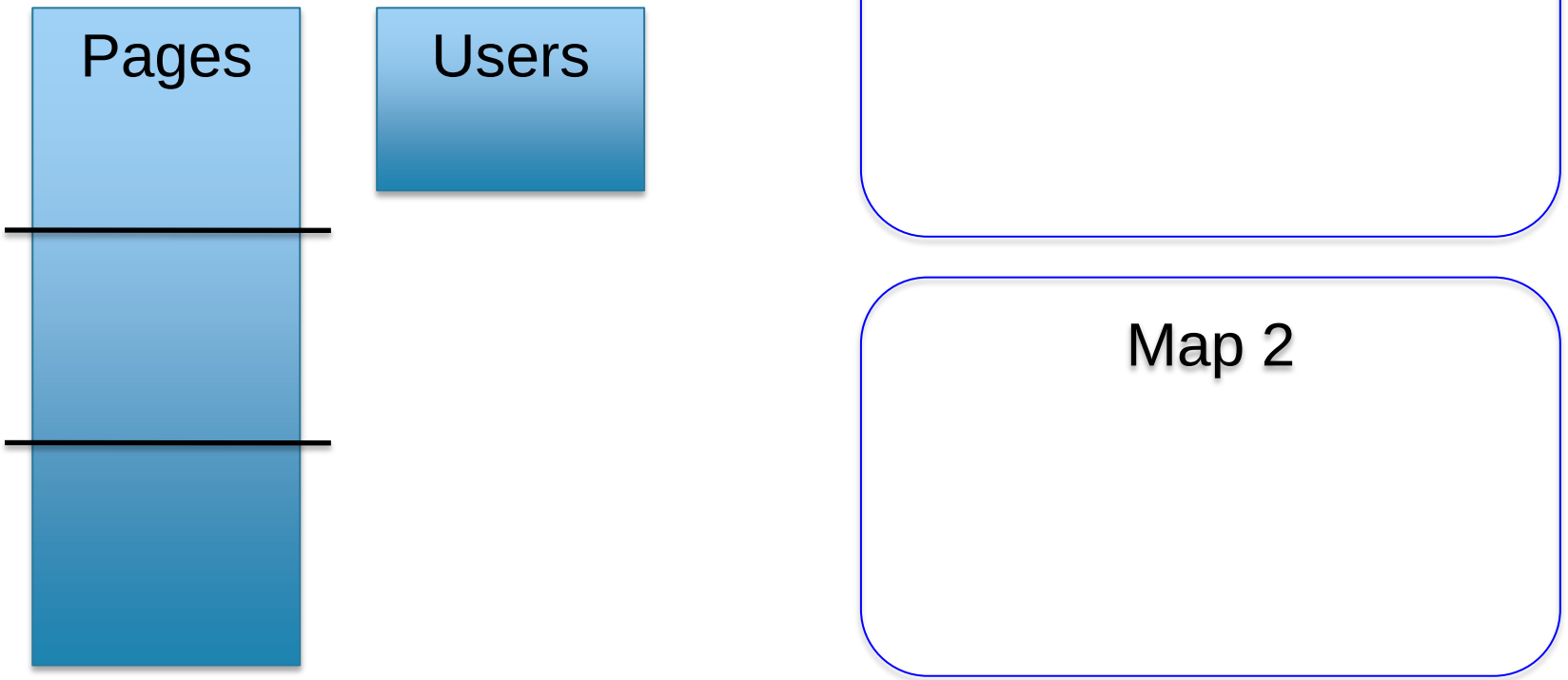


Credit: Alan Gates, Yahoo!

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";
```

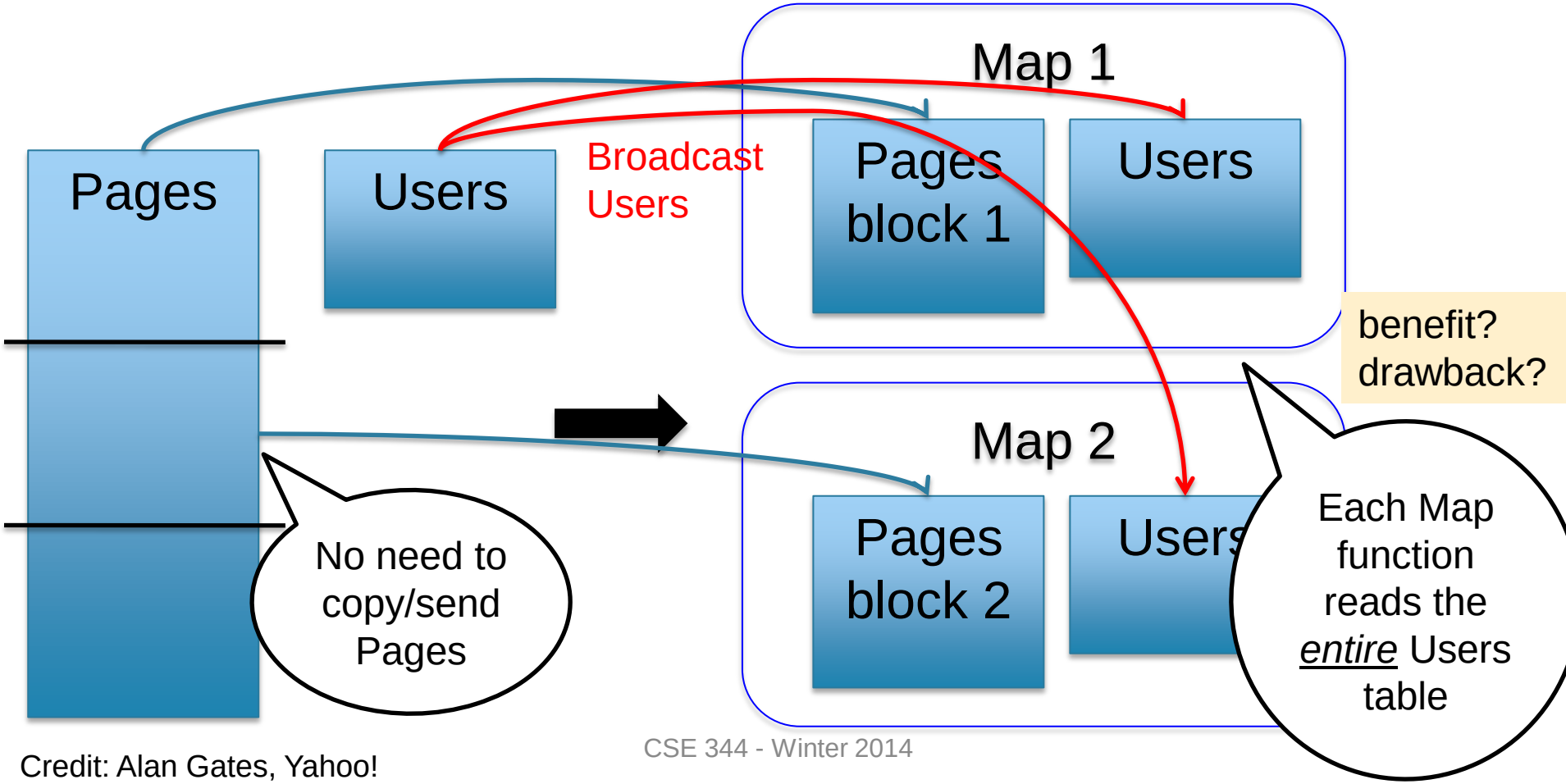


Credit: Alan Gates, Yahoo!

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:

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

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 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

Review: A sample run from last lecture

Anatomy of a Query Execution

- Running problem #4
- 20 nodes = 1 master + 19 workers
- Using PARALLEL 50
- Recall:
  - Multiple map/reduce functions -> single map/reduce task
  - Multiple map/reduce tasks -> single node
  - Map workers write to their local disks, Reduce workers read (copy/sort) from there.

Take a look at the Hadoop tutorial:
https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

From Hadoop Tutorial…

How many maps?

- "The number of maps is usually driven by the total size of the inputs, that is, the total number of blocks of the input files.

- The right level of parallelism for maps seems to be around 10-100 maps per-node,… sometimes 300.."

From Hadoop Tutorial…

How many reduces?

- "The right number of reduces seems to be 0.95 or 1.75 multiplied by (<*no. of nodes*> * mapred.tasktracker.reduce.tasks.maximum).
- With 0.95 all of the reduces can launch immediately and start transfering map outputs as the maps finish. With 1.75 the faster nodes will finish their first round of reduces and launch a second wave of reduces doing a much better job of load balancing.
- Increasing the number of reduces increases the framework overhead, but increases load balancing and lowers the cost of failures.
- The scaling factors above are slightly less than whole numbers to reserve a few reduce slots in the framework for speculative-tasks and failed tasks
- ."

March 2013

Hadoop job_201303091944_0001 on domU-12-31-39-06-75-A1

User: hadoop
Job Name: PigLatin:DefaultJobName
Job File: hdfs://10.208.122.79:9000/mnt/var/lib/hadoop/tmp/mapred/staging/hadoop/.staging/job_201303091944_0001/job.xml
Submit Host: domU-12-31-39-06-75-A1.compute-1.internal
Submit Host Address: 10.208.122.79
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Sat Mar 09 19:49:21 UTC 2013
Finished at: Sat Mar 09 23:33:14 UTC 2013
Finished in: 3hrs, 43mins, 52sec
Job Cleanup: Successful
Black-listed TaskTrackers: 1

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	7908	0	0	7908	0	14 / 16
reduce	100.00%	50	0	0	50	0	0 / 8

	Counter	Map	Reduce	Total
Job Counters	SLOTS_MILLIS_MAPS	0	0	454,162,761
	Launched reduce tasks	0	0	58
	Total time spent by all reduces waiting after reserving slots (ms)	0	0	0
	Rack-local map tasks	0	0	7,938
	Total time spent by all maps waiting after reserving slots	0	0	0

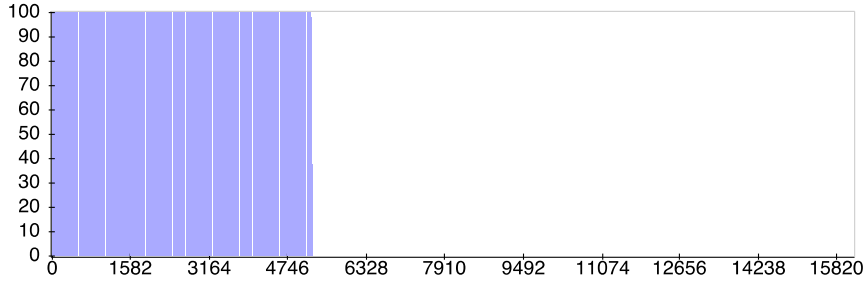
Some other time (March 2012)

- Let's see what happened…

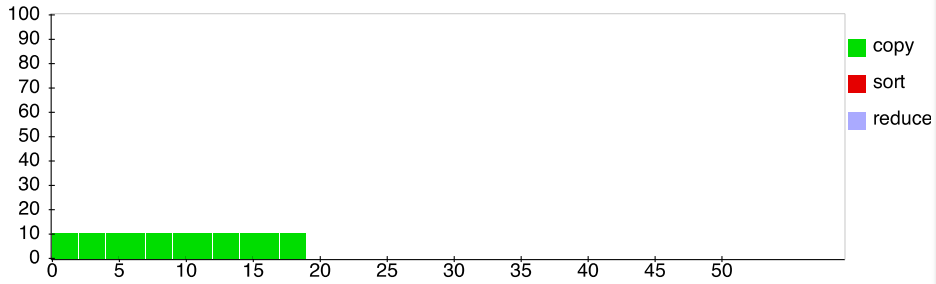
Take a look at the Hadoop tutorial:
https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html

1h 16min

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	33.17%	15816	10549	38	5229	0	0 / 0
reduce	4.17%	50	31	19	0	0	0 / 0



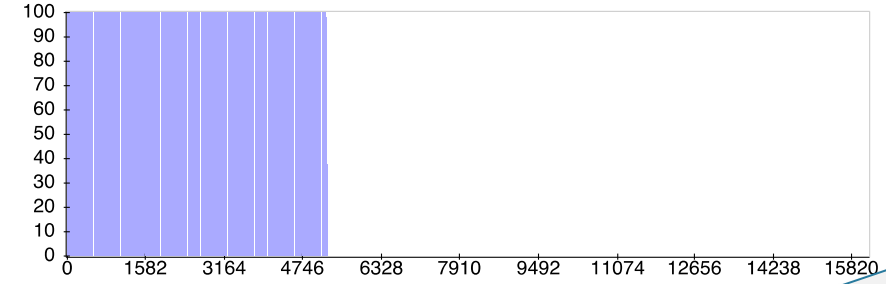
luce Completion Graph - close



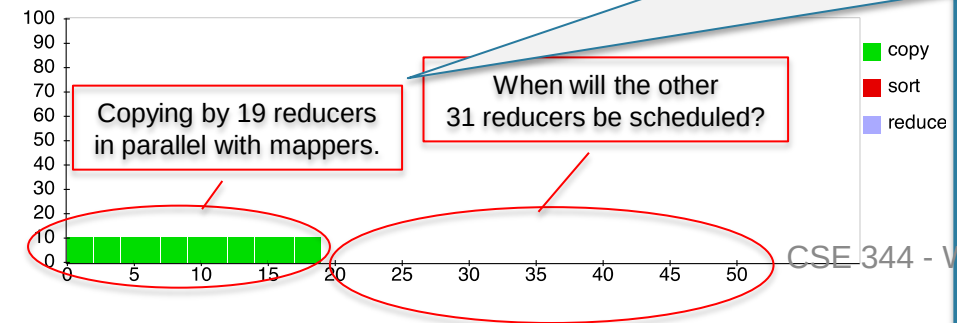
1h 16min

Only 19 reducers active, out of 50. Why?

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	33.17%	15816	10549	38	5229	0	0 / 0
reduce	4.17%	50	31	19	0	0	0 / 0



luce Completion Graph - close

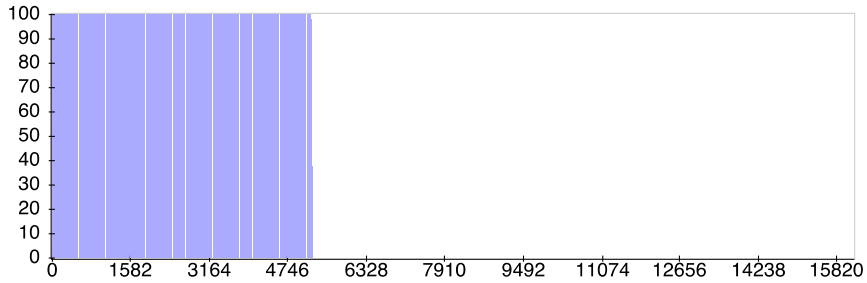


- Map workers keep writing data to local disk
- Reduce workers can start "copy"-ing in parallel
- But cannot start "reduce" functions until the map workers are done.

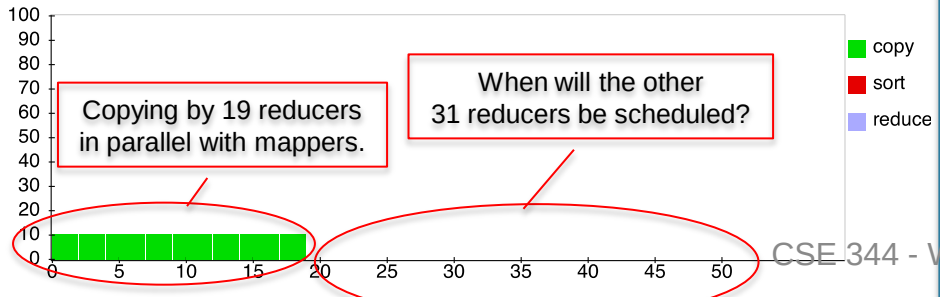
1h 16min

Only 19 reducers active, out of 50. Why?

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	33.17%	15816	10549	38	5229	0	0 / 0
reduce	4.17%	50	31	19	0	0	0 / 0

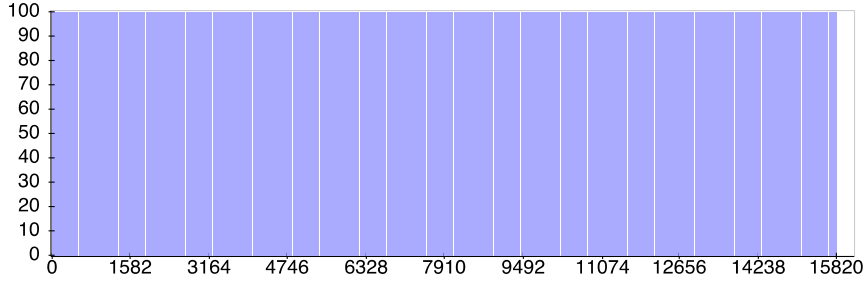


luce Completion Graph - close

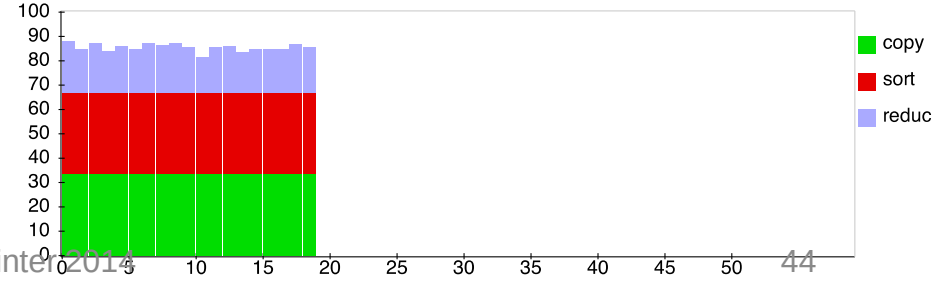


3h 50min

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	15816	0	0	15816	0	0 / 18
reduce	32.42%	50	31	19	0	0	0 / 0



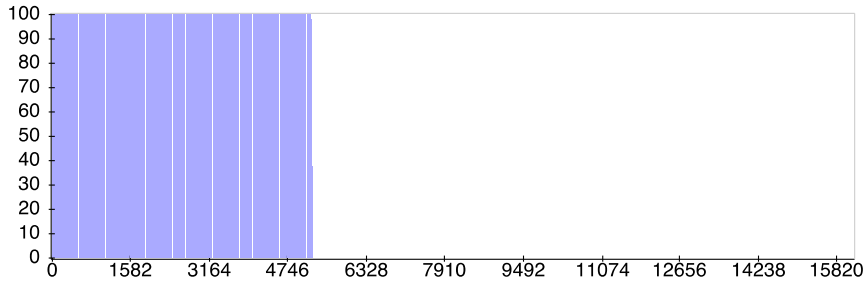
uce Completion Graph - close



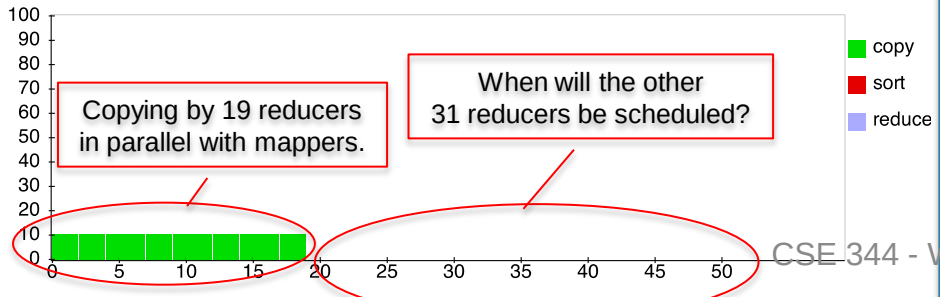
1h 16min

Only 19 reducers active, out of 50. Why?

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	33.17%	15816	10549	38	5229	0	0 / 0
reduce	4.17%	50	31	19	0	0	0 / 0



luce Completion Graph - close

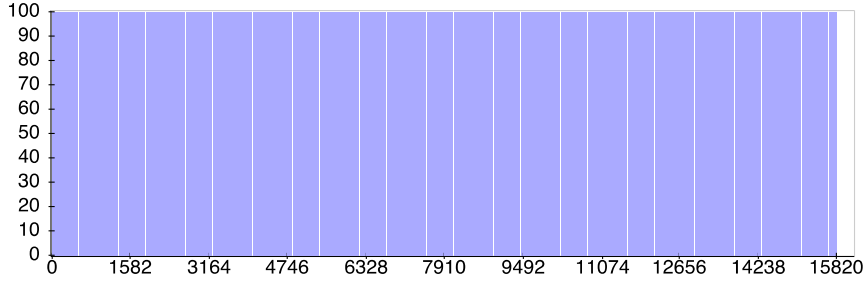


3h 50min

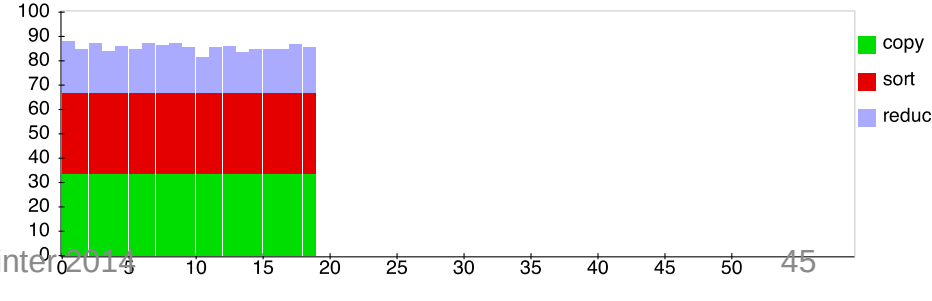
Speculative Execution

Completed. Sorting, and the rest of Reduce may proceed now

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	15816	0	0	15816	0	0 / 18
reduce	32.42%	50	31	19	0	0	0 / 0



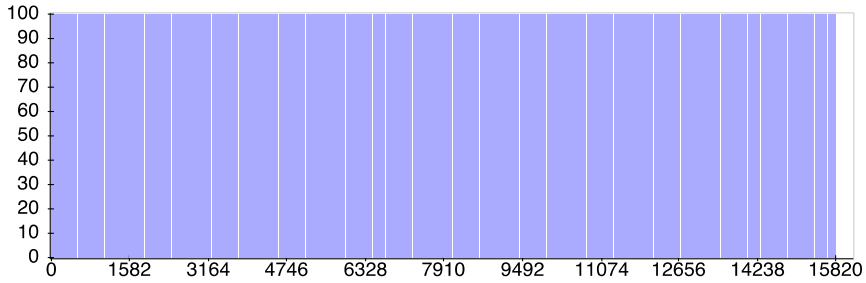
uce Completion Graph - close



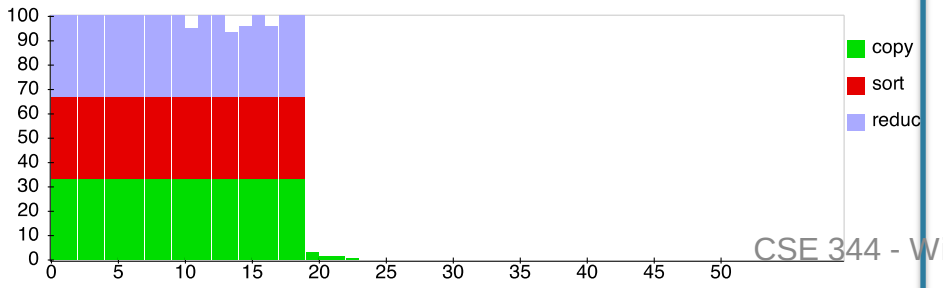
3h 51min

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	15816	0	0	15816	0	0 / 18
reduce	37.72%	50	19	22	9	0	0 / 0

ɔ Completion Graph - close



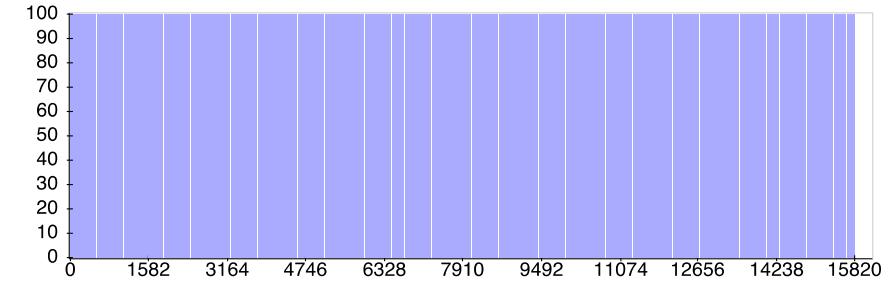
duce Completion Graph - close



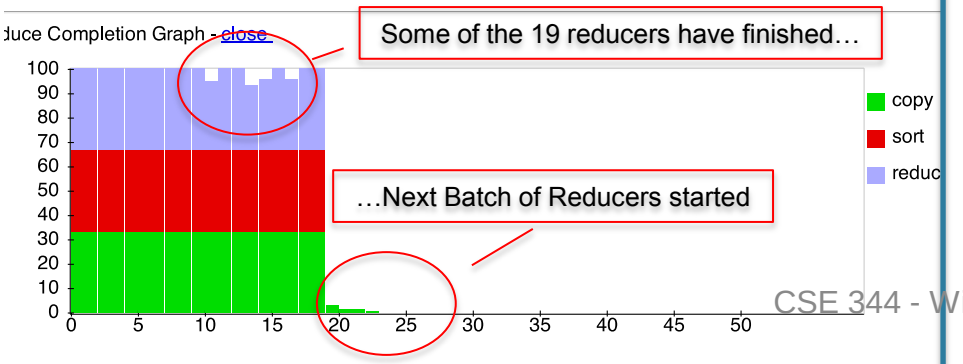
3h 51min

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	15816	0	0	15816	0	0 / 18
reduce	37.72%	50	19	22	9	0	0 / 0

ɔ Completion Graph - close



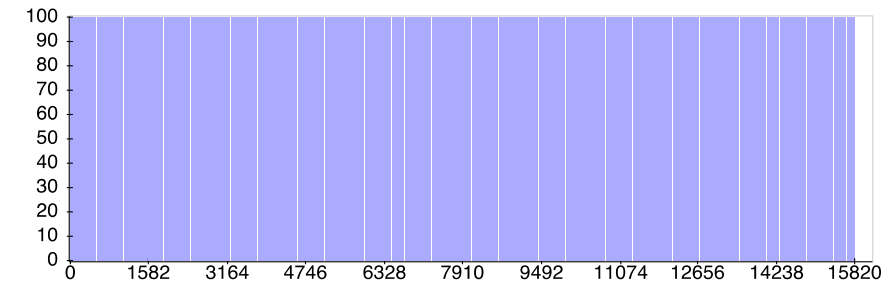
duce Completion Graph - close



3h 51min

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	15816	0	0	15816	0	0 / 18
reduce	37.72%	50	19	22	9	0	0 / 0

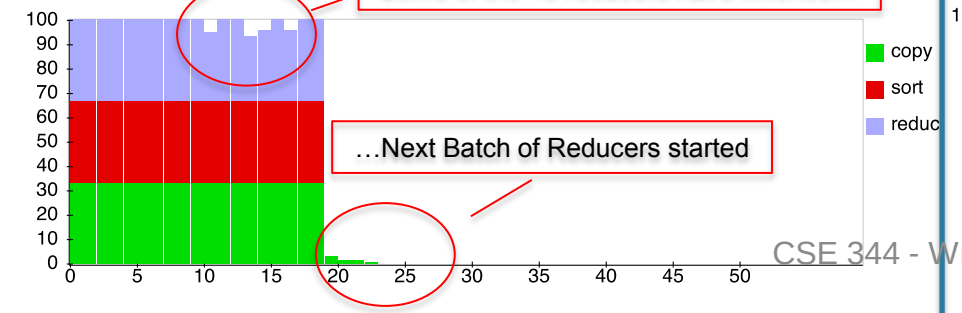
ɔ Completion Graph - close



duce Completion Graph - close

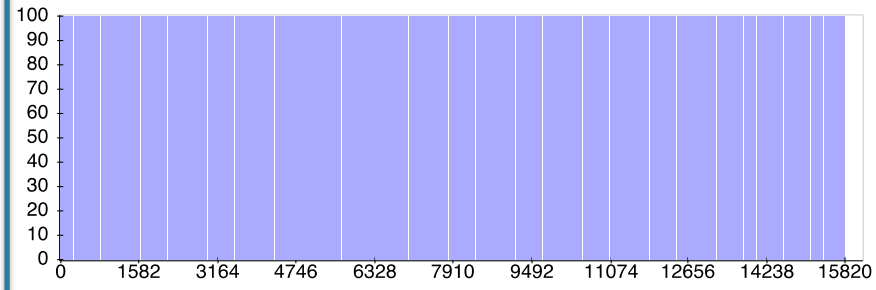
Some of the 19 reducers have finished…

…Next Batch of Reducers started



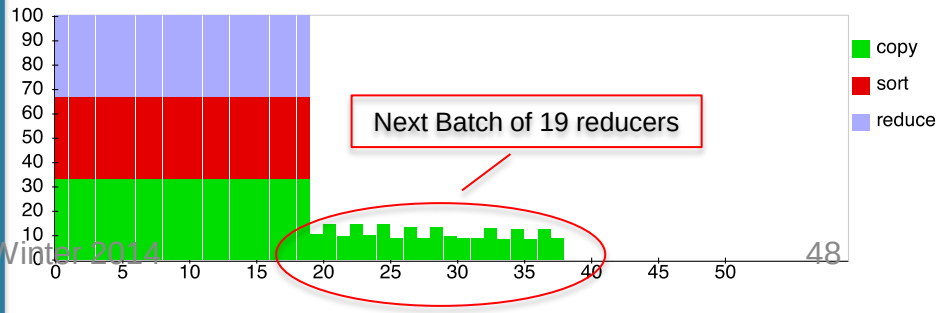
3h 52min

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	15816	0	0	15816	0	0 / 18
reduce	42.35%	50	11	20	19	0	0 / 0



ce Completion Graph - close

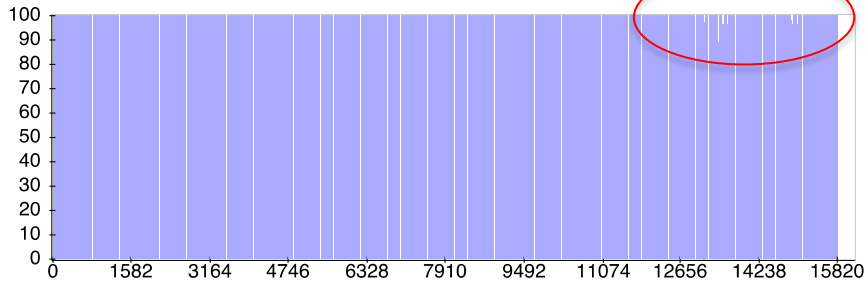
Next Batch of 19 reducers



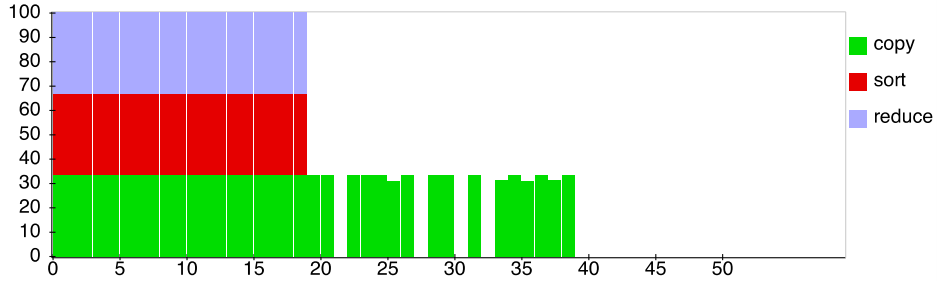
4h 18min

Several servers failed: “fetch error”. Their map tasks need to be rerun. All reducers are waiting….

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	99.88%	15816	2638	30	13148	0	15 / 3337
reduce	48.42%	50	15	16	19	0	0 / 0



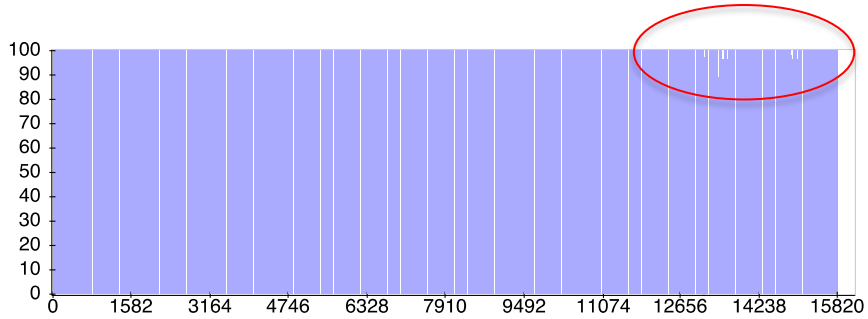
uce Completion Graph - close



4h 18min

Several servers failed: “fetch error”. Their map tasks need to be rerun. All reducers are waiting….

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	99.88%	15816	2638	30	13148	0	15 / 3337
reduce	48.42%	50	15	16	19	0	0 / 0

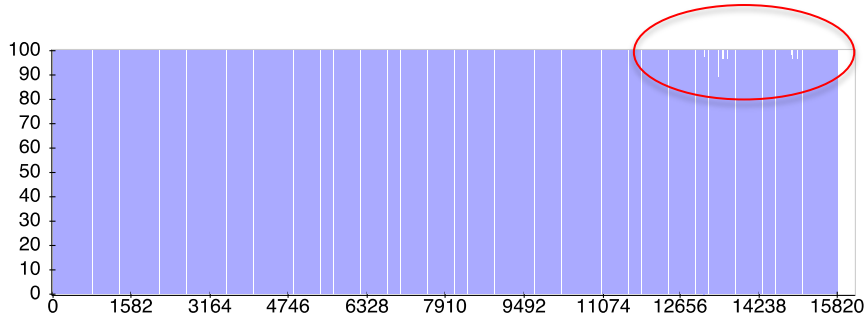


uce Completion Graph - close

4h 18min

Several servers failed: “fetch error”. Their map tasks need to be rerun. All reducers are waiting….

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	99.88%	15816	2638	30	13148	0	15 / 3337
reduce	48.42%	50	15	16	19	0	0 / 0

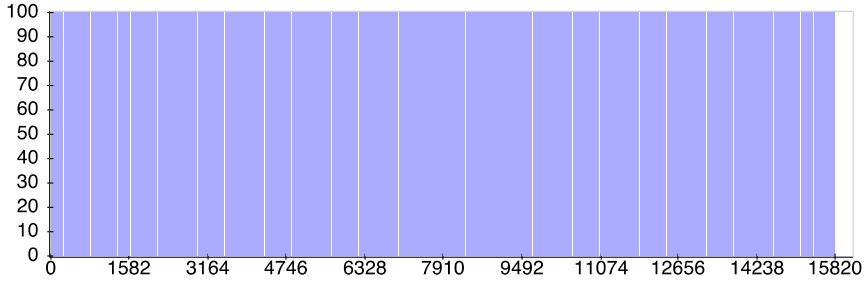


uce Completion Graph - close

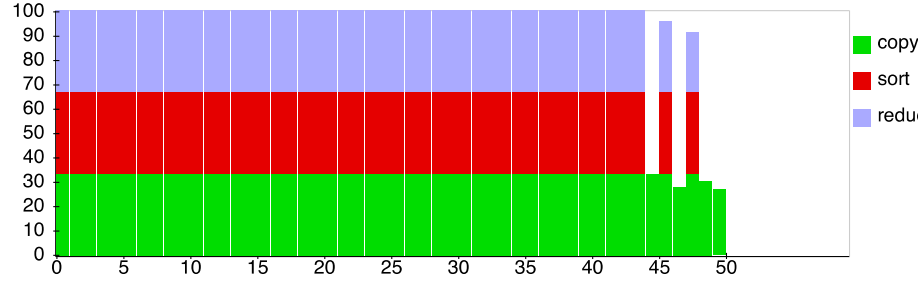
7h 10min

Mappers finished, reducers resumed.

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	15816	0	0	15816	0	26 / 5968
reduce	94.15%	50	0	6	44	0	0 / 8



uce Completion Graph - close



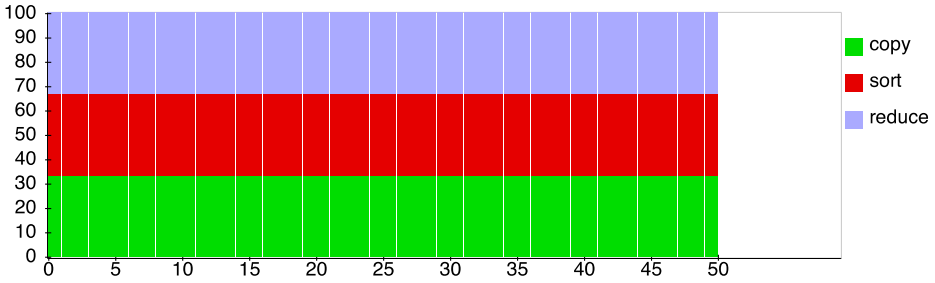
7h 20min

Success! 7hrs, 20mins.

Hadoop job_201203041905_0001 on ip-10-203-30-146

User: hadoop
Job Name: PigLatin:DefaultJobName
Job File: hdfs://10.203.30.146:9000/mnt/var/lib/hadoop/tmp/mapred/staging/hadoop/.staging/job_201203041905_0001/job.xml
Submit Host: ip-10-203-30-146.ec2.internal
Submit Host Address: 10.203.30.146
Job-ACLs: All users are allowed
Job Setup: Successful
Status: Succeeded
Started at: Sun Mar 04 19:08:29 UTC 2012
Finished at: Mon Mar 05 02:28:39 UTC 2012
Finished in: 7hrs, 20mins, 10sec
Job Cleanup: Successful
Black-listed TaskTrackers: 3

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00%	15816	0	0	15816	0	26 / 5968
reduce	100.00%	50	0	0	50	0	0 / 14



Parallel DBs v.s. MapReduce

Parallel DB

- Plusses
- Minuses

MapReduce

- Minuses
- Plusses

Parallel DBs v.s. MapReduce

Parallel DB

- Plusses
  - Efficient 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

Review: 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).

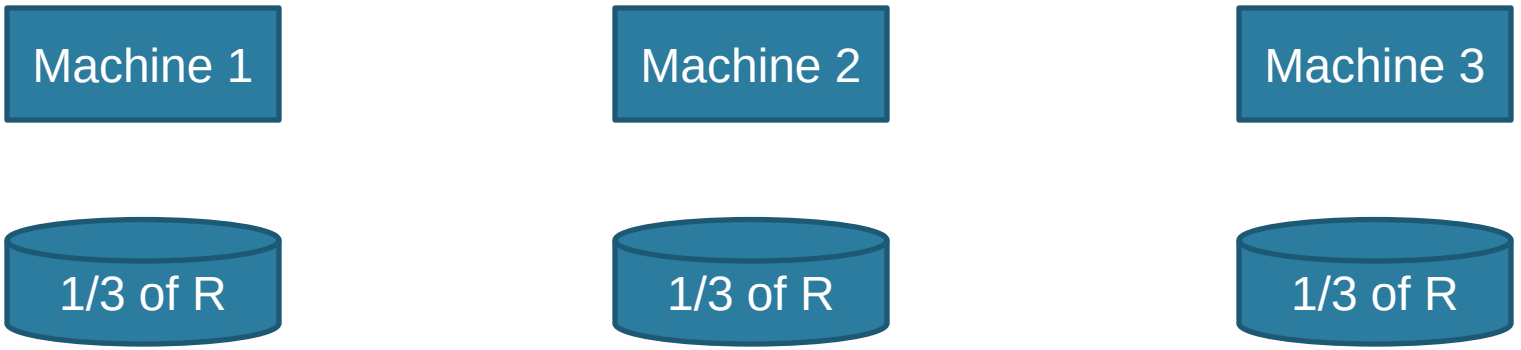
Show a RA plan for this query and how it will be executed across the N = 3 machines.

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)

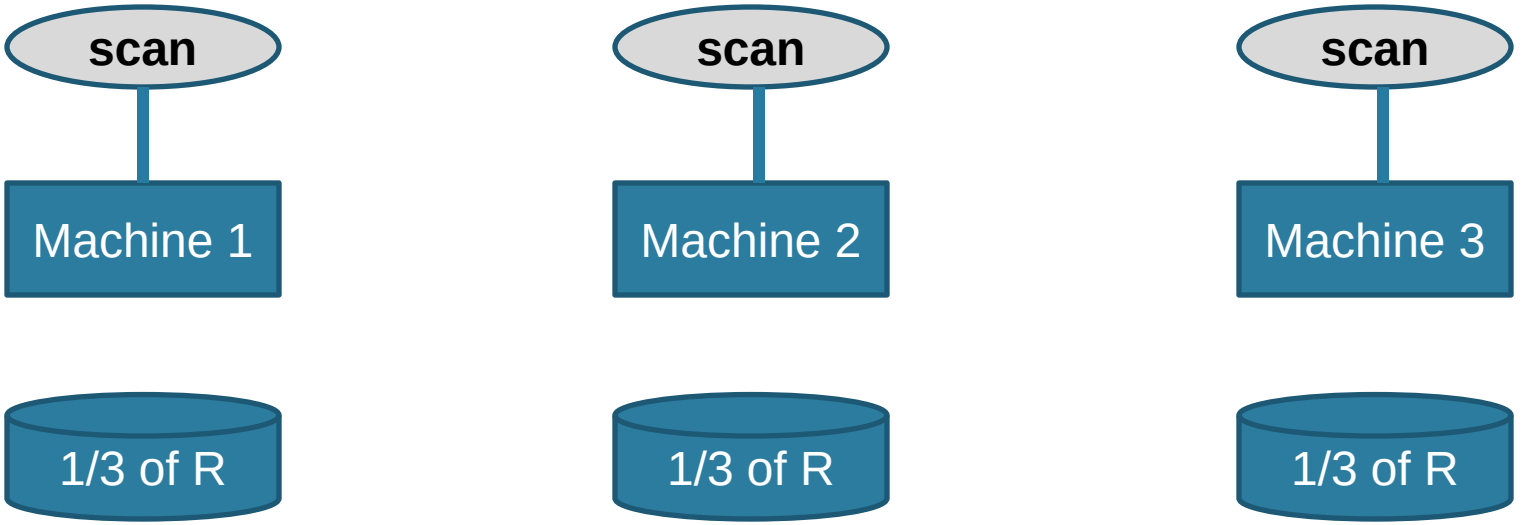
```
SELECT a, max(b) as
topb
FROM R
WHERE a > 0
GROUP BY a
```



R(a, b)

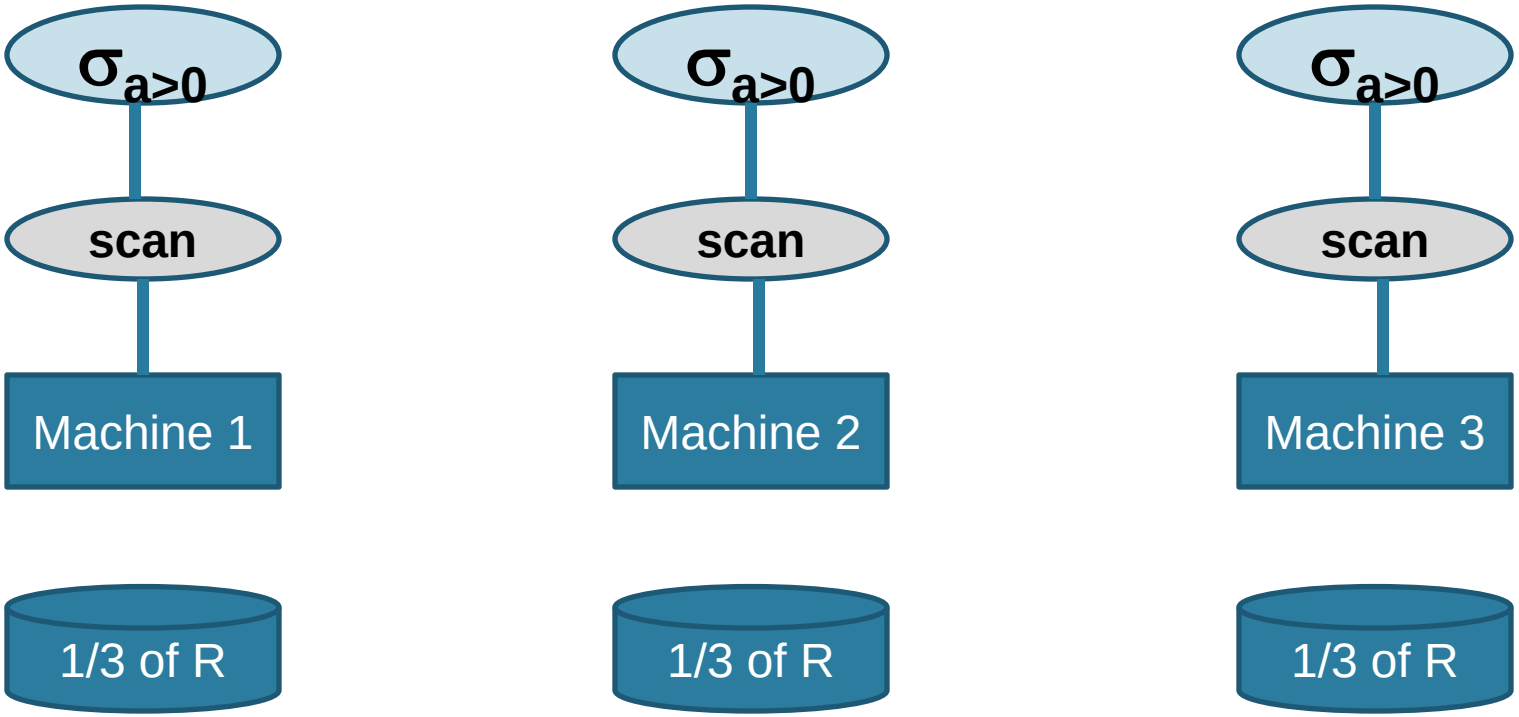
```
SELECT a, max(b) as
topb
FROM R
WHERE a > 0
GROUP BY a
```

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



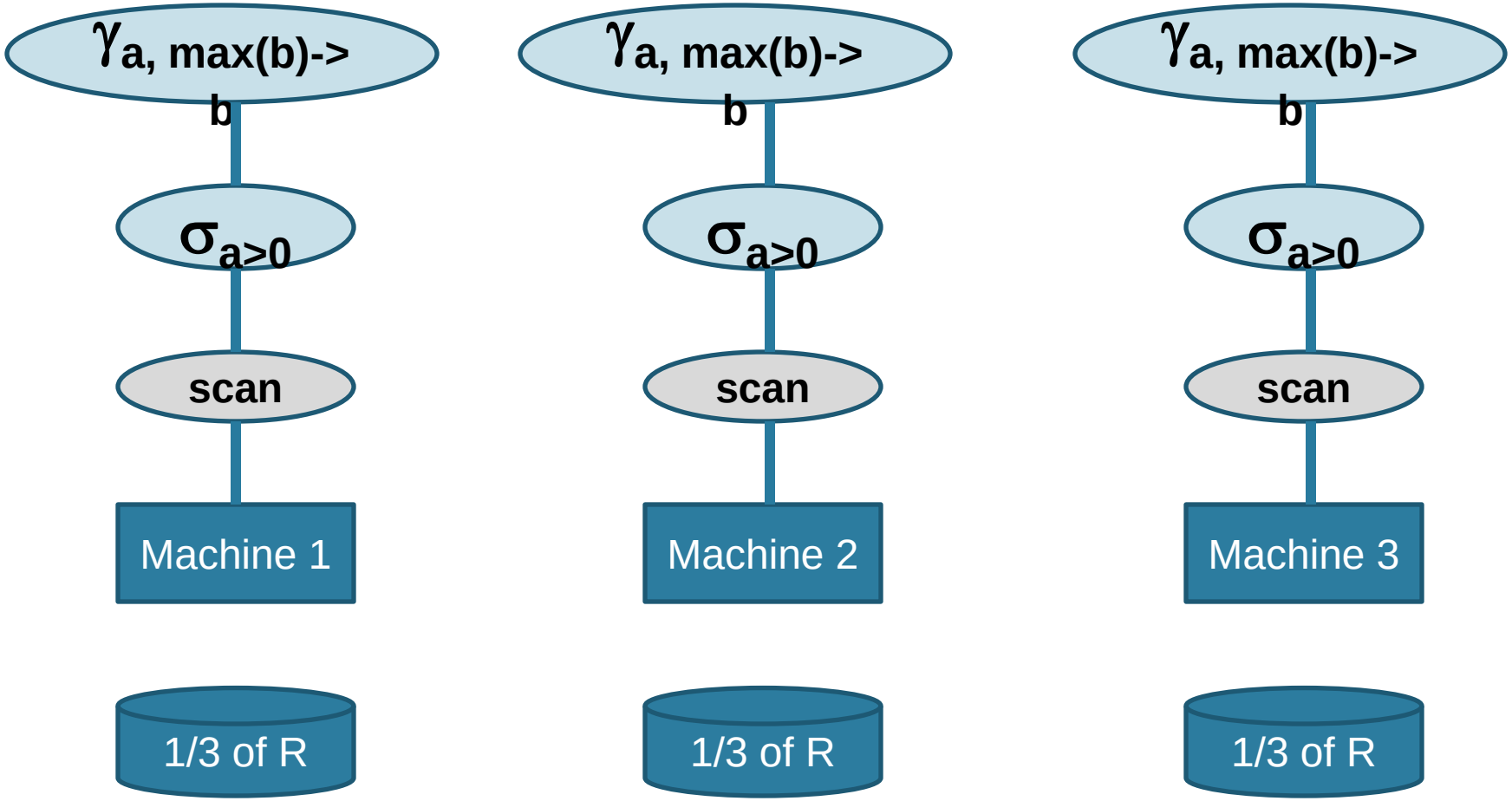
R(a, b)

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



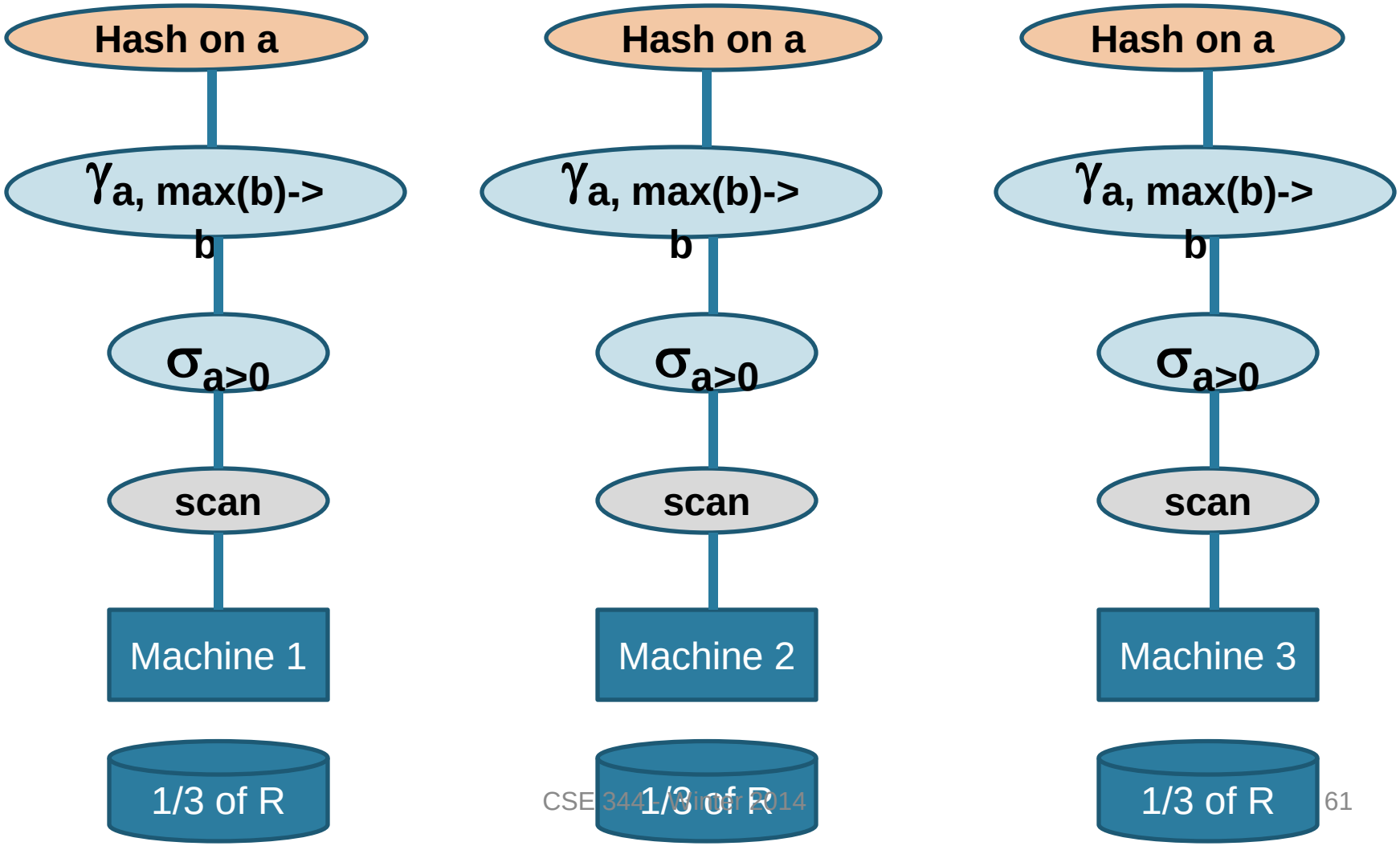
R(a, b)

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



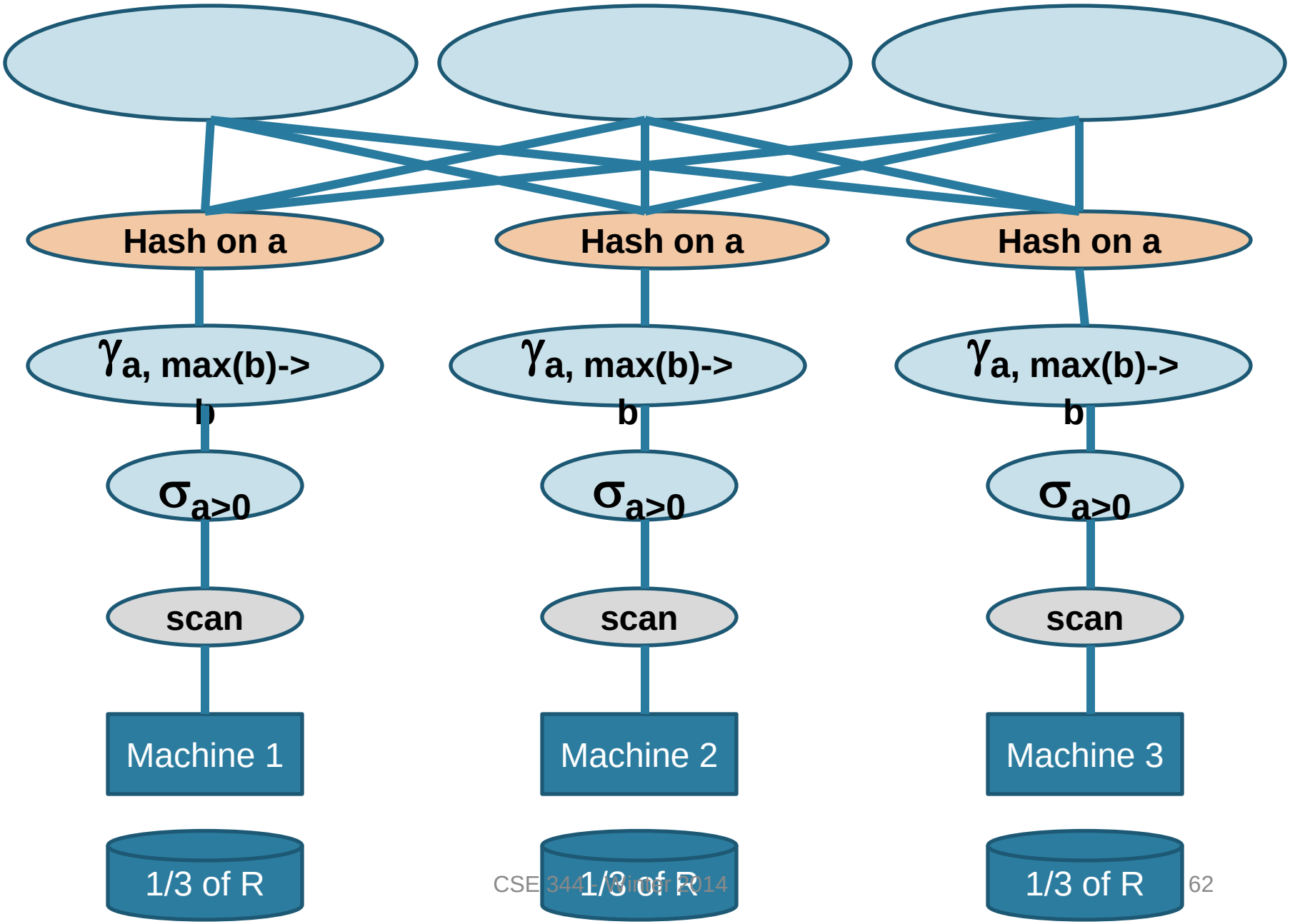
R(a, b)

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



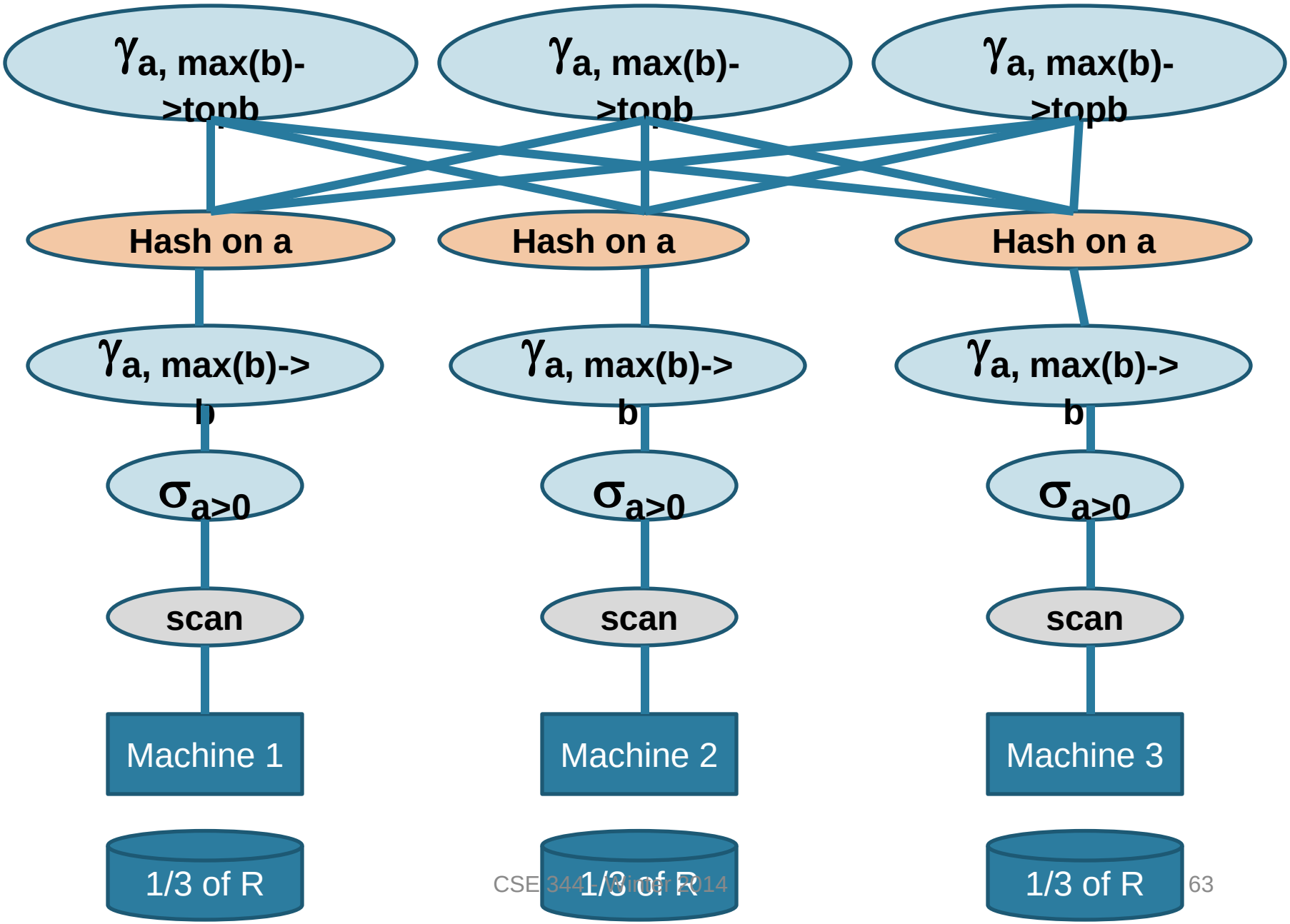
R(a, b)

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

```
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’

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

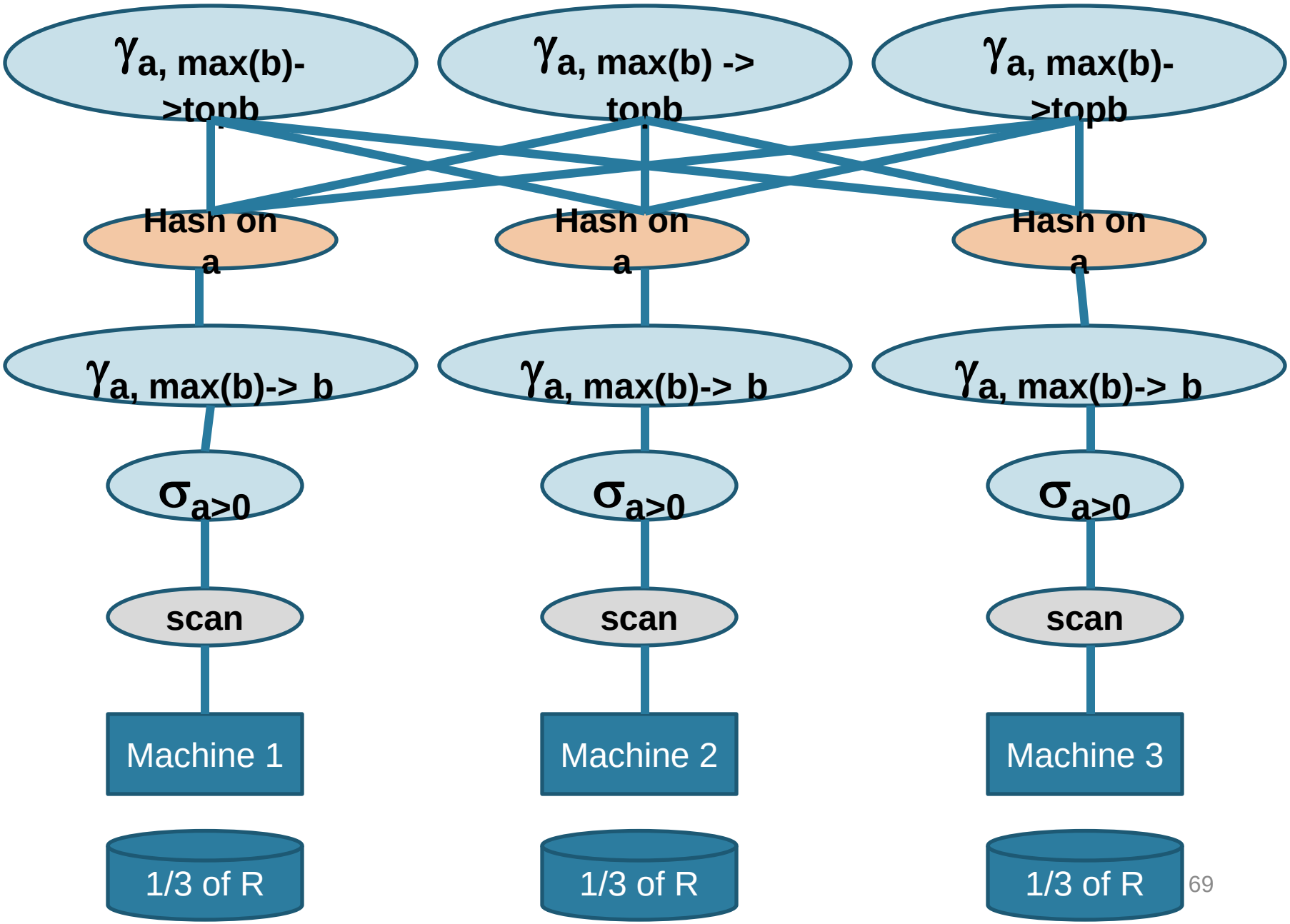
```
SELECT a, max(b) as
topb
FROM R
WHERE a > 0
GROUP BY a
```

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



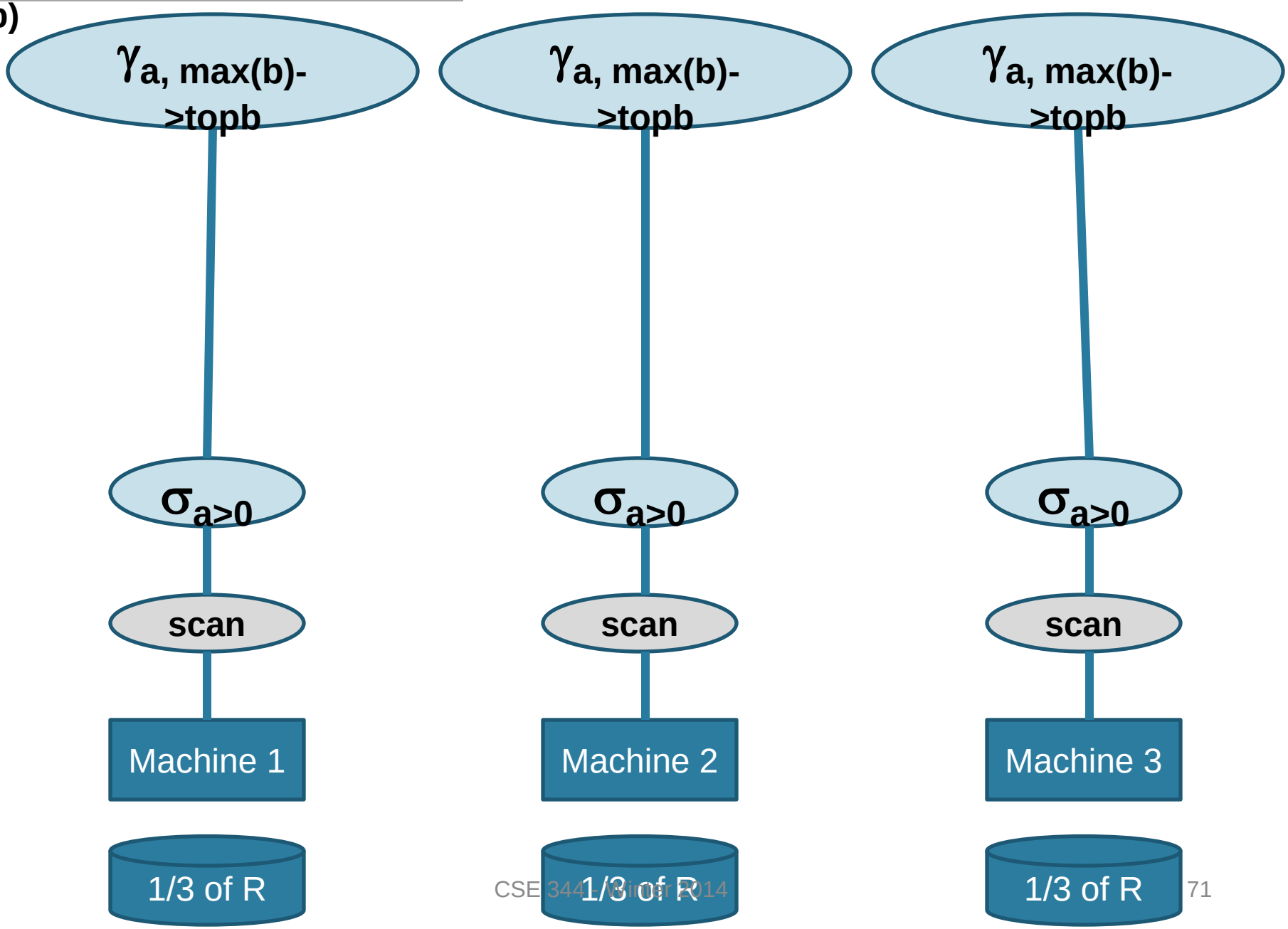
```
SELECT a, max(b) as
topb
FROM R
WHERE a > 0
GROUP BY a
```

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 > 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 not have to do anything (if one map task/node) or less computation (multiple map tasks/node)