CompSci 516
Database Systems

Lecture 20

Parallel DBMS

Instructor: Sudeepa Roy

Announcements

- HW3 due on Monday, Nov 20, 11:55 pm (in 2 weeks)
  - See some clarifications on Piazza

Reading Material

- [RG]
  - Parallel DBMS: Chapter 22.1-22.5

- [GUW]
  - Parallel DBMS and map-reduce: Chapter 20.1-20.2

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

Reading Material

- [RG]
  - Parallel DBMS: Chapter 22.1-22.5
  - Distributed DBMS: Chapter 22.6 – 22.14

- [GUW]
  - Parallel DBMS and map-reduce: Chapter 20.1-20.2
  - Distributed DBMS: Chapter 20.3, 20.4.1-20.4.2, 20.5-20.6

- Recommended readings:
  - Chapter 2 (Sections 1,2,3) of Mining of Massive Datasets, by Rajaraman and Ullman: http://i.stanford.edu/~ullman/mmds.html
  - Original Google MR paper by Jeff Dean and Sanjay Ghemawat, OSDI' 04: http://research.google.com/archive/mapreduce.html

Acknowledgement:
The following slides have been created adapting the instructor material of the [RG] book provided by the authors Dr. Ramakrishnan and Dr. Gehrke.

Parallel and Distributed Data Processing

- Recall from Lecture 18!
- data and operation distribution if we have multiple machines

- Parallelism
  - performance

- Data distribution
  - increased availability, e.g. when a site goes down
  - distributed local access to data (e.g. an organization may have branches in several cities)
  - analysis of distributed data

Parallel vs. Distributed DBMS

Parallel DBMS

- Parallelization of various operations
  - e.g. loading data, building indexes, evaluating queries
- Data may or may not be distributed initially
- Distribution is governed by performance consideration

Distributed DBMS

Lecture 18

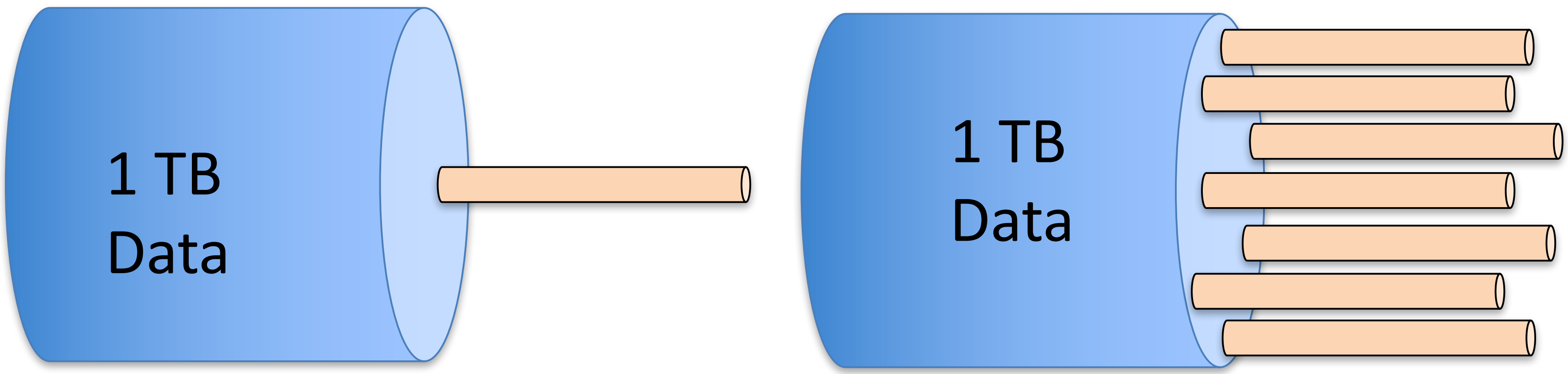
- Data is physically stored across different sites
  - Each site is typically managed by an independent DBMS
- Location of data and autonomy of sites have an impact on Query opt., Conc. Control and recovery
- Also governed by other factors:
  - increased availability for system crash
  - local ownership and access

Parallel DBMS

Why Parallel Access To Data?

At 10 MB/s
1.2 days to scan

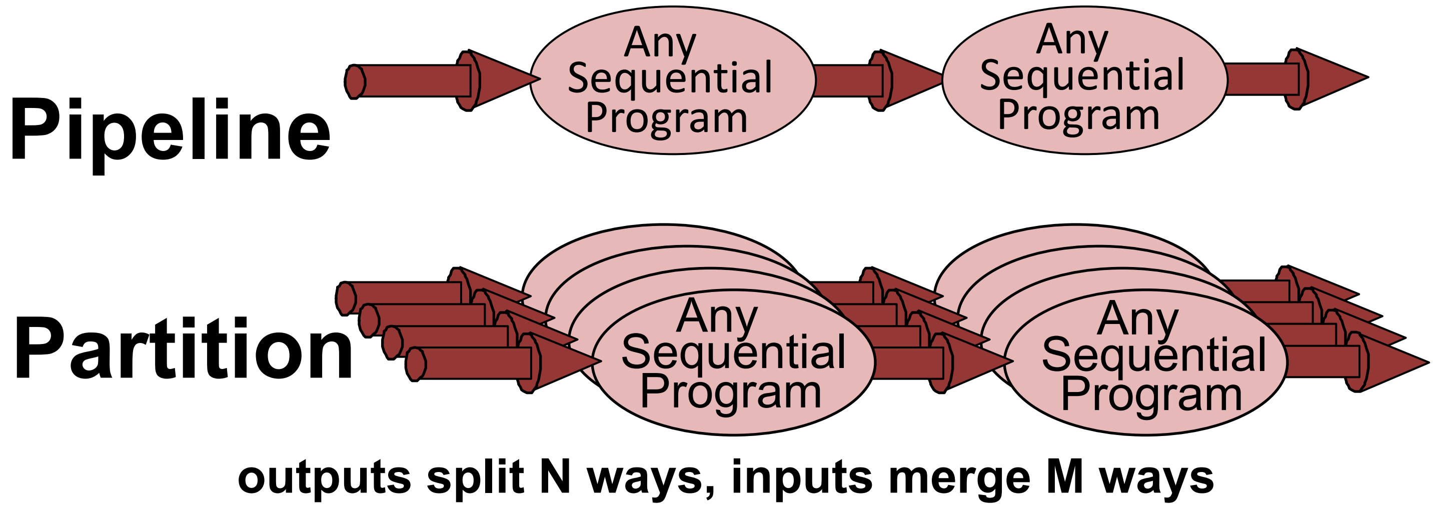
1,000 x parallel
1.5 minute to scan.



Parallelism:
divide a big problem
into many smaller ones
to be solved in parallel.

Parallel DBMS

- Parallelism is natural to DBMS processing
  - Pipeline parallelism: many machines each doing one step in a multi-step process.
  - Data-partitioned parallelism: many machines doing the same thing to different pieces of data.
  - Both are natural in DBMS!



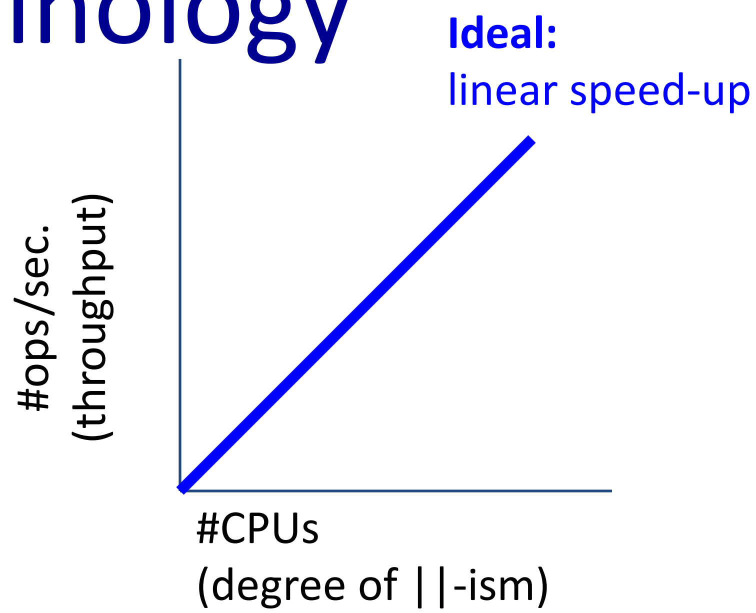
DBMS: The parallel Success Story

- DBMSs are the most successful application of parallelism
  - Teradata (1979), Tandem (1974, later acquired by HP),..
  - Every major DBMS vendor has some parallel server
- Reasons for success:
  - Bulk-processing (= partition parallelism)
  - Natural pipelining
  - Inexpensive hardware can do the trick
  - Users/app-programmers don’t need to think in parallel

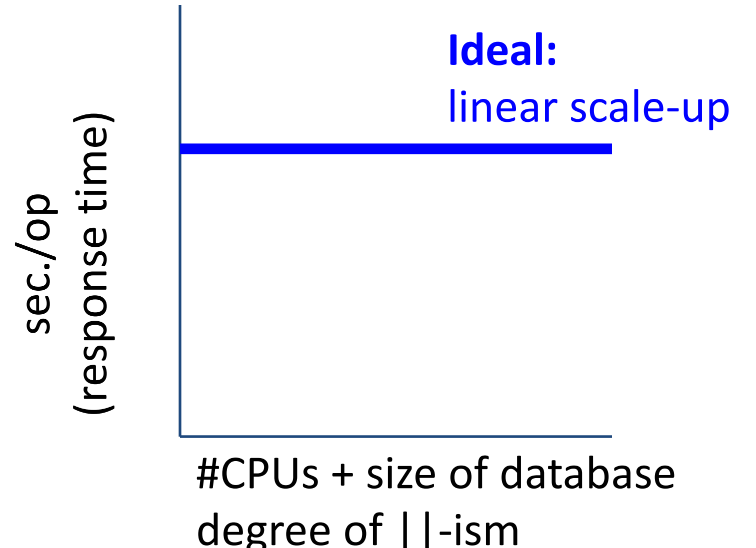
Some || Terminology

Ideal graphs

- Speed-Up
  - More resources means proportionally less time for given amount of data.



- Scale-Up
  - If resources increased in proportion to increase in data size, time is constant.



Some || Terminology

In practice

- Due to overhead in parallel processing
- Start-up cost

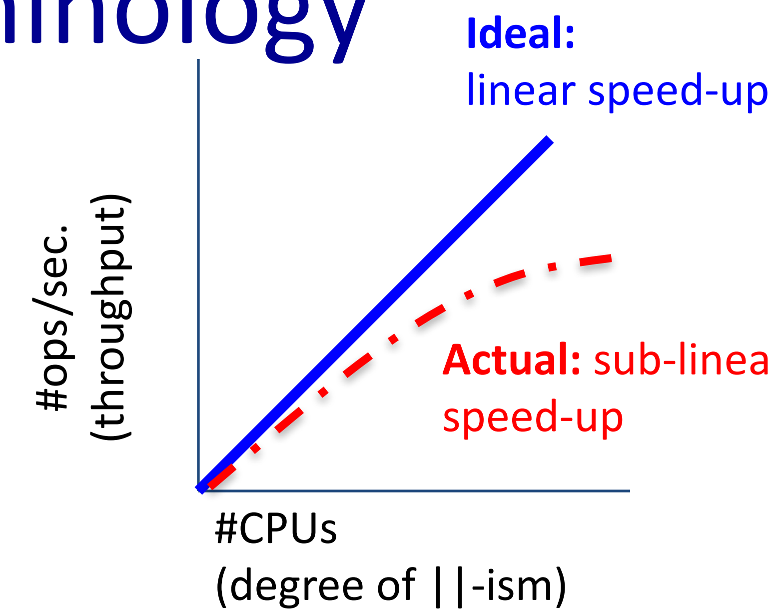
Starting the operation on many processor, might need to distribute data

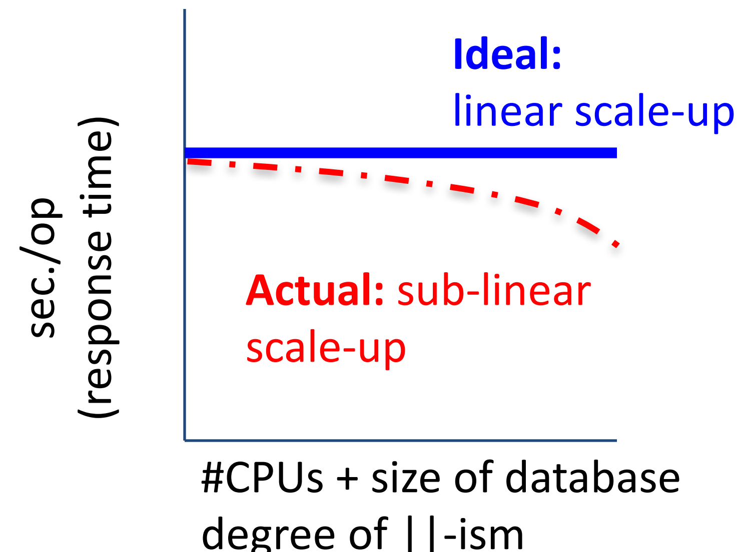
- Interference

Different processors may compete for the same resources

- Skew

The slowest processor (e.g. with a huge fraction of data) may become the bottleneck





Architecture for Parallel DBMS

- Among different computing units
  - Whether memory is shared
  - Whether disk is shared

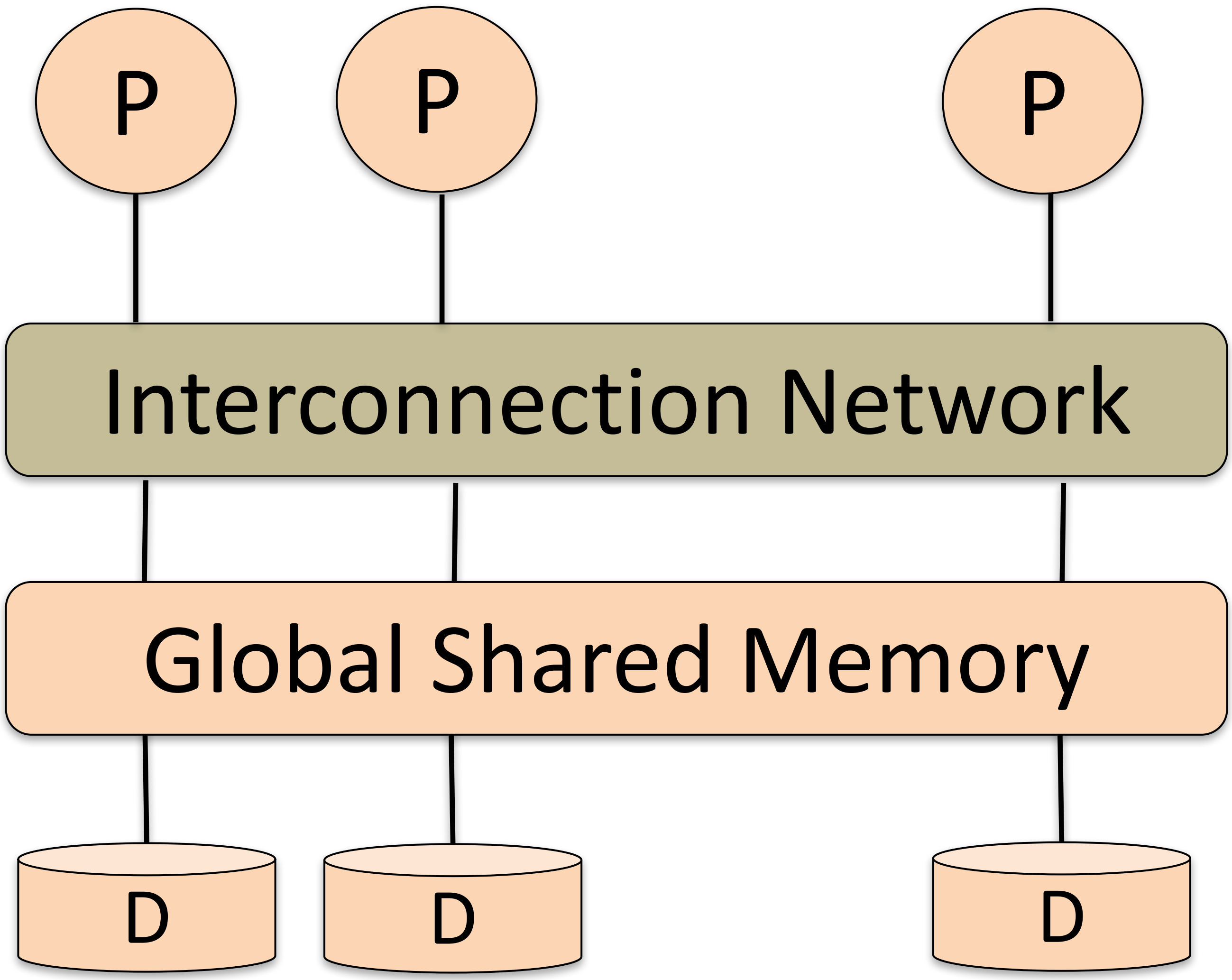
Basics of Parallelism

- Units: a collection of processors
  - assume always have local cache
  - may or may not have local memory or disk (next)

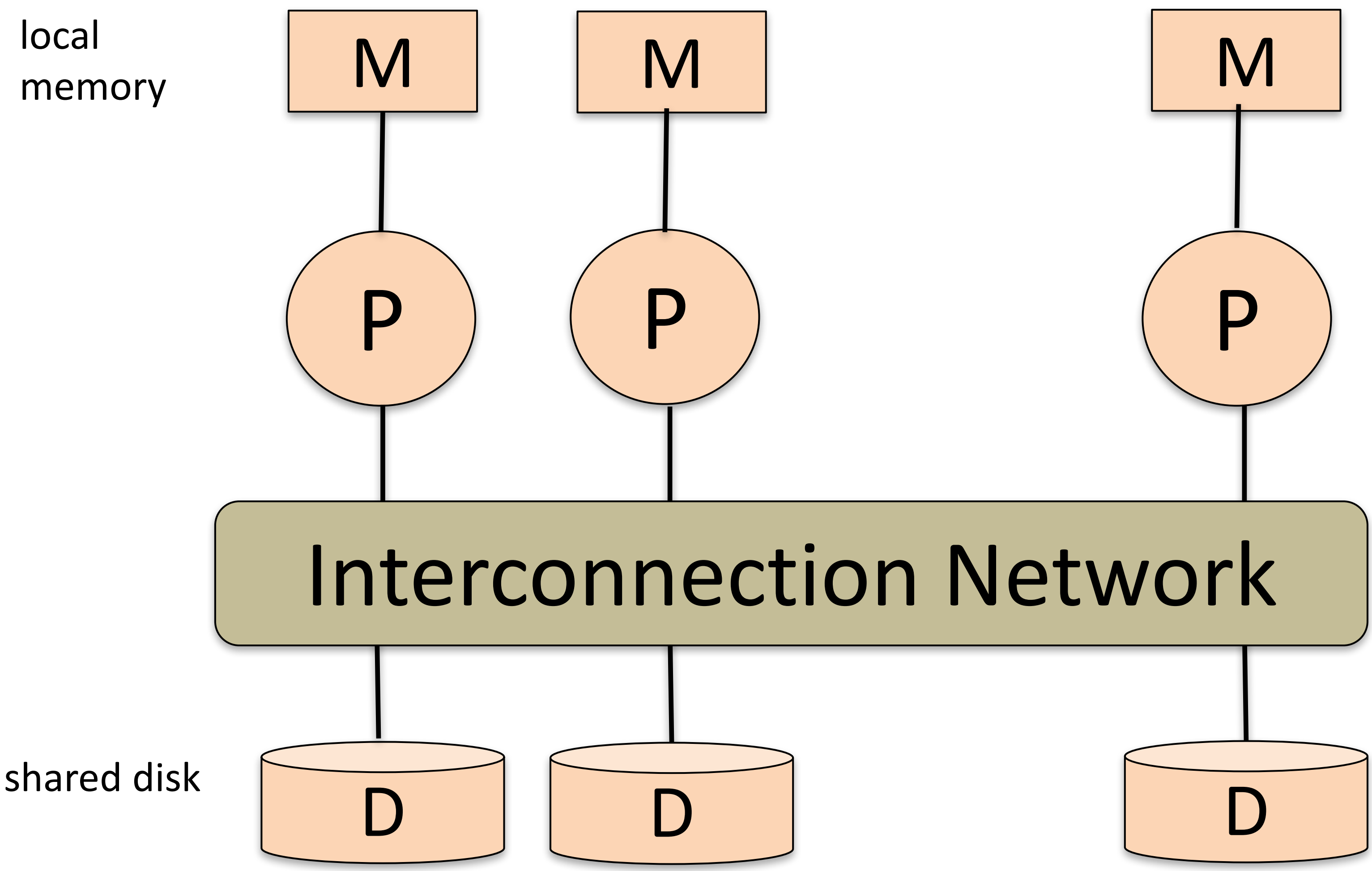
- A communication facility to pass information among processors
  - a shared bus or a switch

Shared Memory

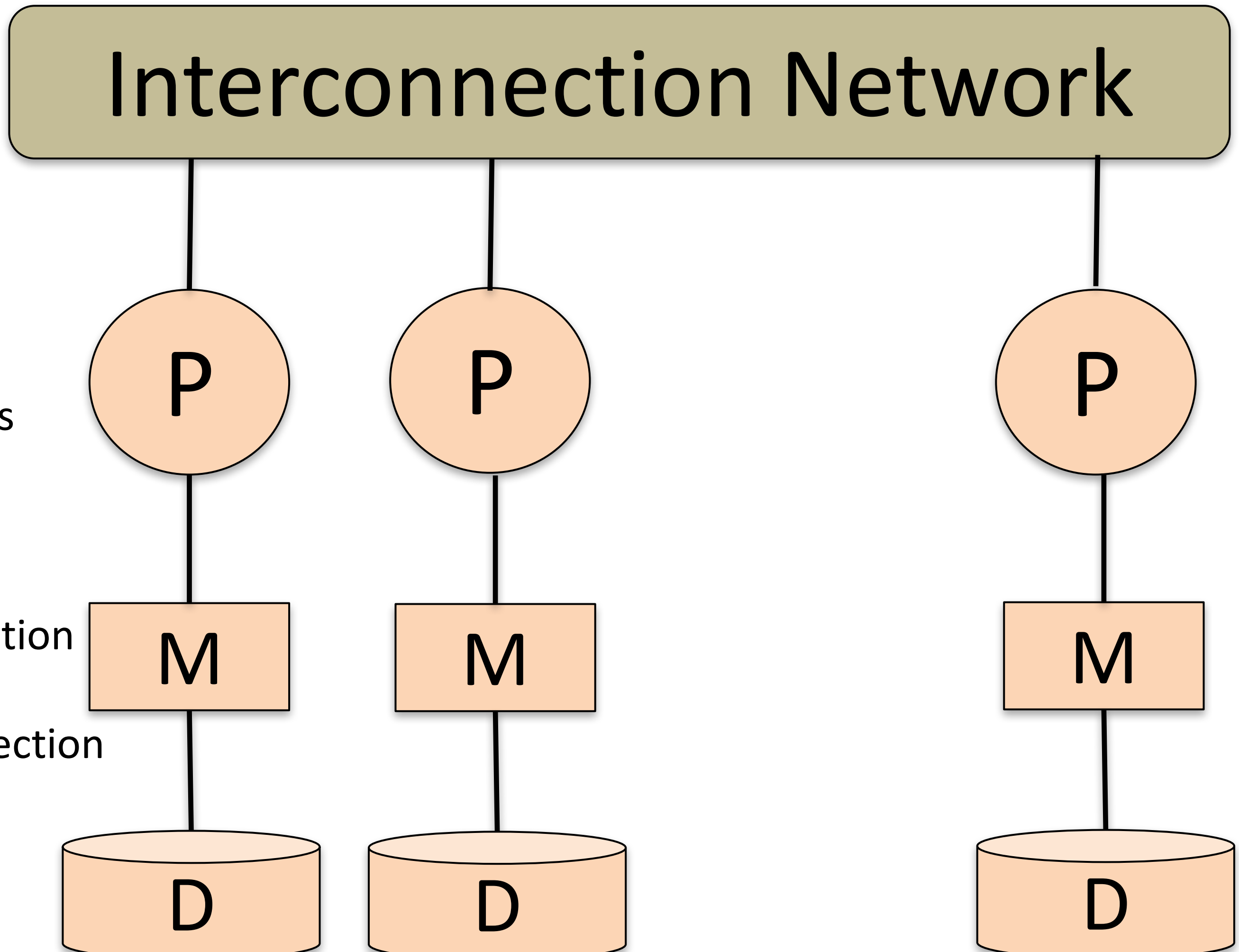
shared
memory



Shared Disk


local
memory
M
M
M
P
P
P
Interconnection Network
shared disk
D
D
D

Shared Nothing



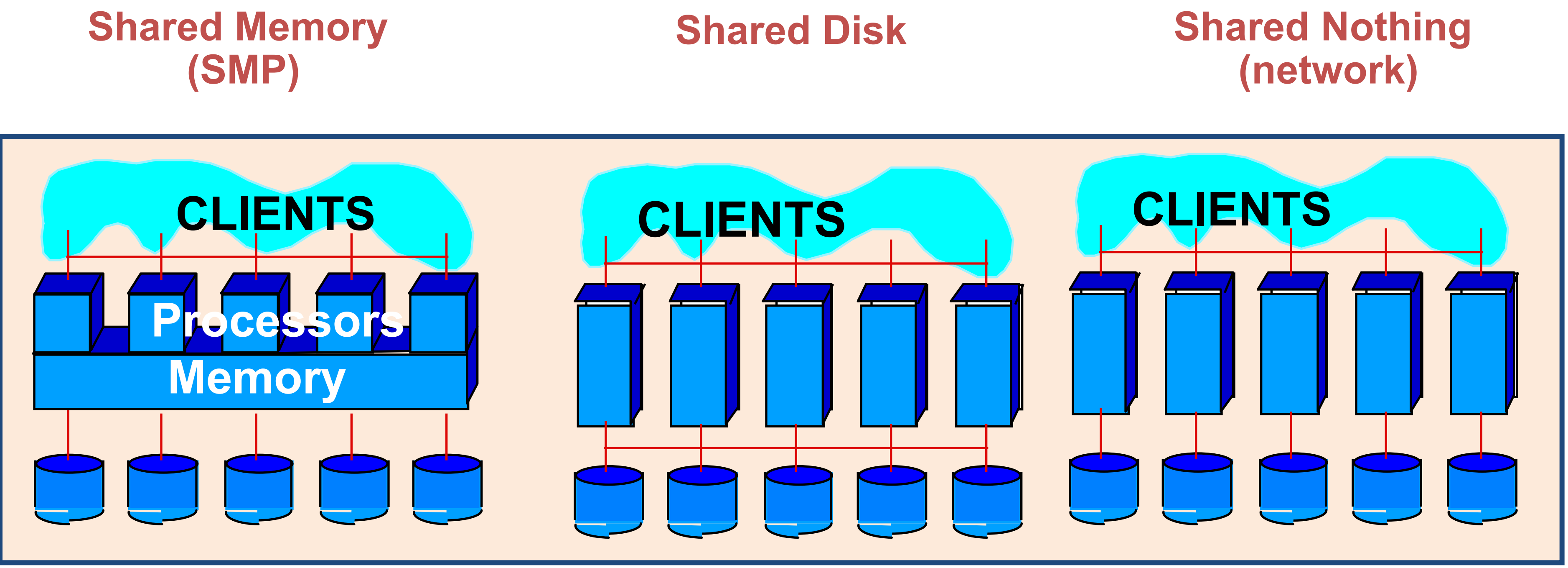
local
memory
and disk

no two
CPU can access
the same
storage area

all communication
through a
network connection

Architecture: At A Glance

we will assume shared nothing



- Easy to program
- Expensive to build
- Low communication overhead: shared mem.
- Difficult to scaleup (memory contention)

Sequent, SGI, Sun

- Trade-off but still interference like shared-memory (contention of memory and nw bandwidth)

VMScluster, Sysplex

- Hard to program and design parallel algos
- Cheap to build
- Easy to scaleup and speedup
- Considered to be the best architecture

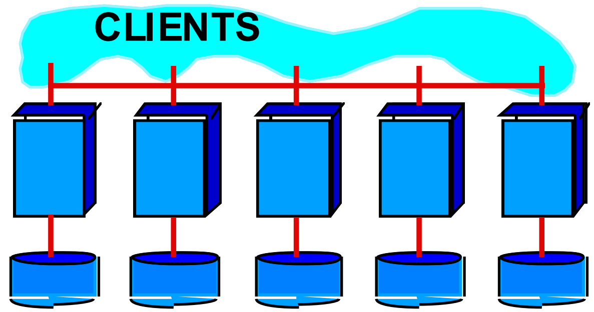
Tandem, Teradata, SP2

What Systems Worked This Way

NOTE: (as of 9/1995)!

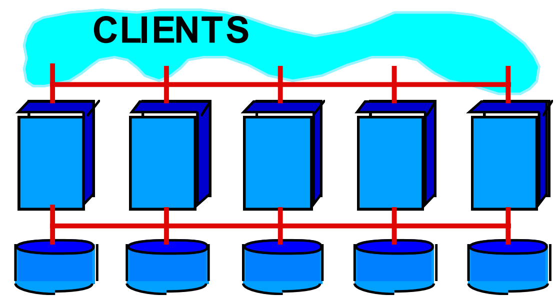
Shared Nothing

Teradata: 400 nodes
Tandem: 110 nodes
IBM / SP2 / DB2: 128 nodes
Informix/SP2 48 nodes
ATT & Sybase ? nodes



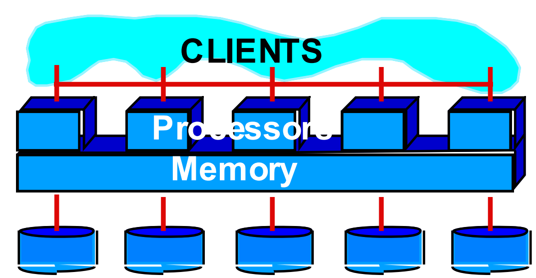
Shared Disk

Oracle 170 nodes
DEC Rdb 24 nodes



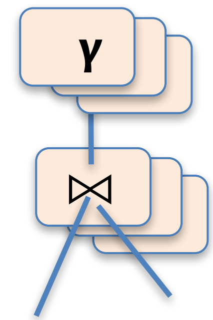
Shared Memory

Informix 9 nodes
RedBrick ? nodes

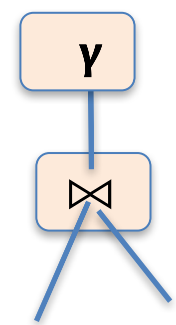


Different Types of DBMS Parallelism

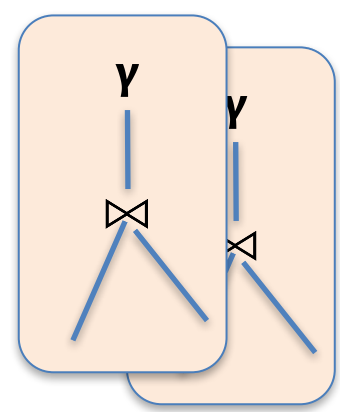
- Intra-operator parallelism
  - get all machines working to compute a given operation (scan, sort, join)
  - OLAP (decision support)



- Inter-operator parallelism
  - each operator may run concurrently on a different site (exploits pipelining)
  - For both OLAP and OLTP



- Inter-query parallelism
  - different queries run on different sites
  - For OLTP



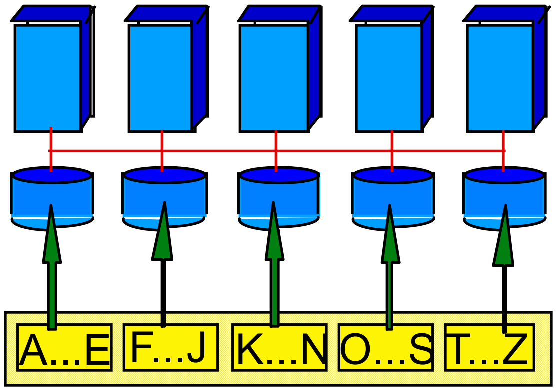
- We'll focus on intra-operator parallelism

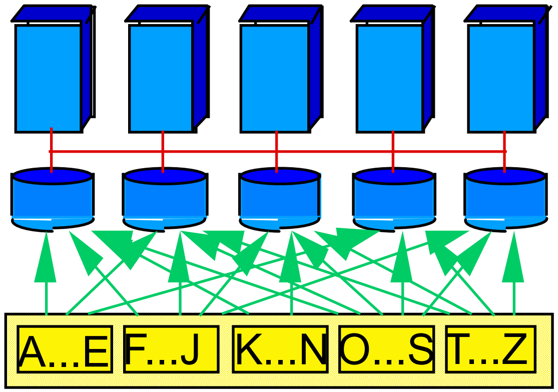
Ack:
Slide by Prof. Dan Suciu

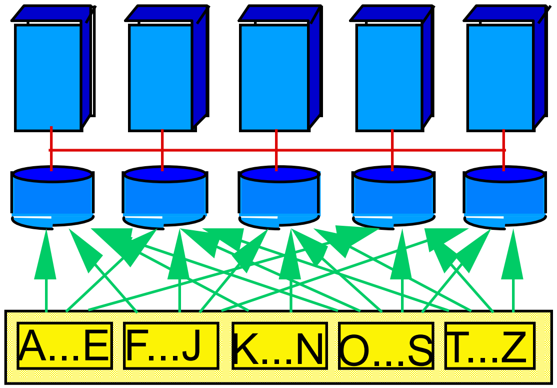
Data Partitioning

Horizontally Partitioning a table (why horizontal?):

Range-partition | Hash-partition | Block-partition or Round Robin







Range-partition
- Good for equijoins, range queries, group-by
- Can lead to data skew

Hash-partition
- Good for equijoins
- But only if hashed on that attribute
- Can lead to data skew

Block-partition or Round Robin
- Send i-th tuple to i-mod-n processor
- Good to spread load
- Good when the entire relation is accessed

Shared disk and memory less sensitive to partitioning,
Shared nothing benefits from "good" partitioning

Example

- R(Key, A, B)

- Can Block-partition be skewed?
  – no, uniform

- Can Hash-partition be skewed?
  – on the key: uniform with a good hash function
  – on A: may be skewed,
    - e.g. when all tuples have the same A-value

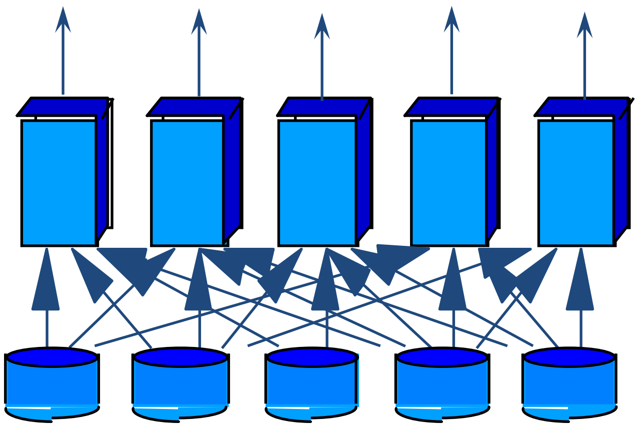
Parallelizing Sequential Evaluation Code

- “Streams” from different disks or the output of other operators
  - are “merged” as needed as input to some operator
  - are “split” as needed for subsequent parallel processing
- Different Split and merge operations appear in addition to relational operators
- No fixed formula for conversion
- Next: parallelizing individual operations

Parallel Scans

- Scan in parallel, and merge.
- Selection may not require all sites for range or hash partitioning
  - but may lead to skew
  - Suppose $\sigma_{A = 10}R$ and partitioned according to A
  - Then all tuples in the same partition/processor
- Indexes can be built at each partition

Parallel Sorting



Idea:

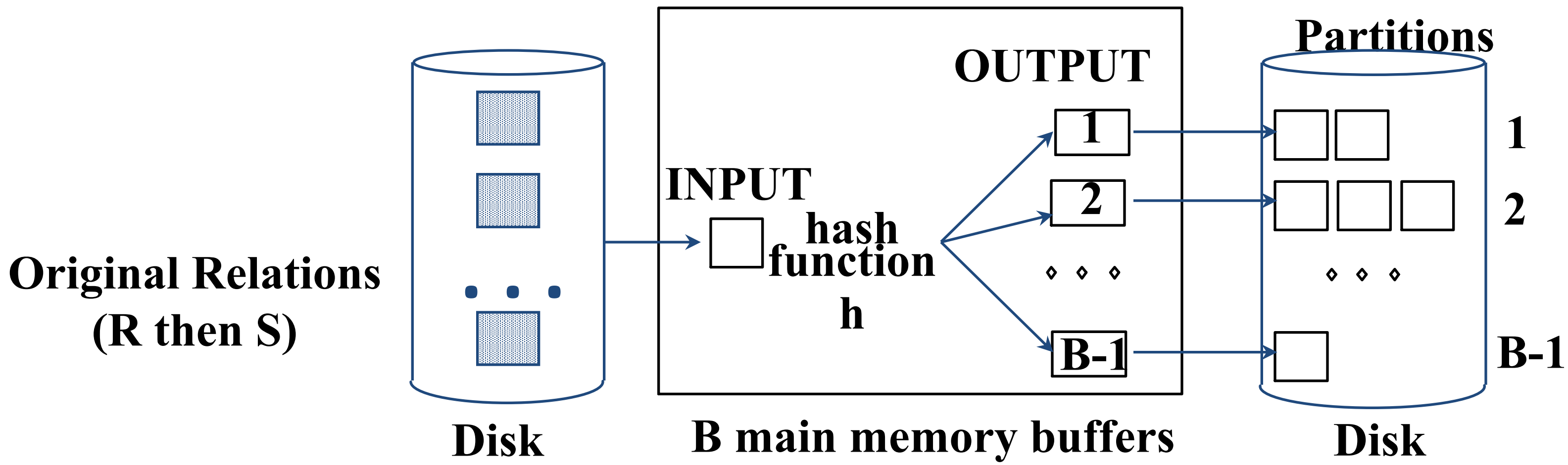
- Scan in parallel, and range-partition as you go
  - e.g. salary between 10 to 210, #processors = 20
  - salary in first processor: 10-20, second: 21-30, third: 31-40, ...
- As tuples come in, begin "local" sorting on each
- Resulting data is sorted, and range-partitioned
- Visit the processors in order to get a full sorted order
- Problem: skew!
- Solution: "sample" the data at start to determine partition points.

Parallel Joins

- Need to send the tuples that will join to the same machine
  - also for GROUP-BY
- Nested loop:
  - Each outer tuple must be compared with each inner tuple that might join
  - Easy for range partitioning on join cols, hard otherwise
- Sort-Merge:
  - Sorting gives range-partitioning
  - Merging partitioned tables is local

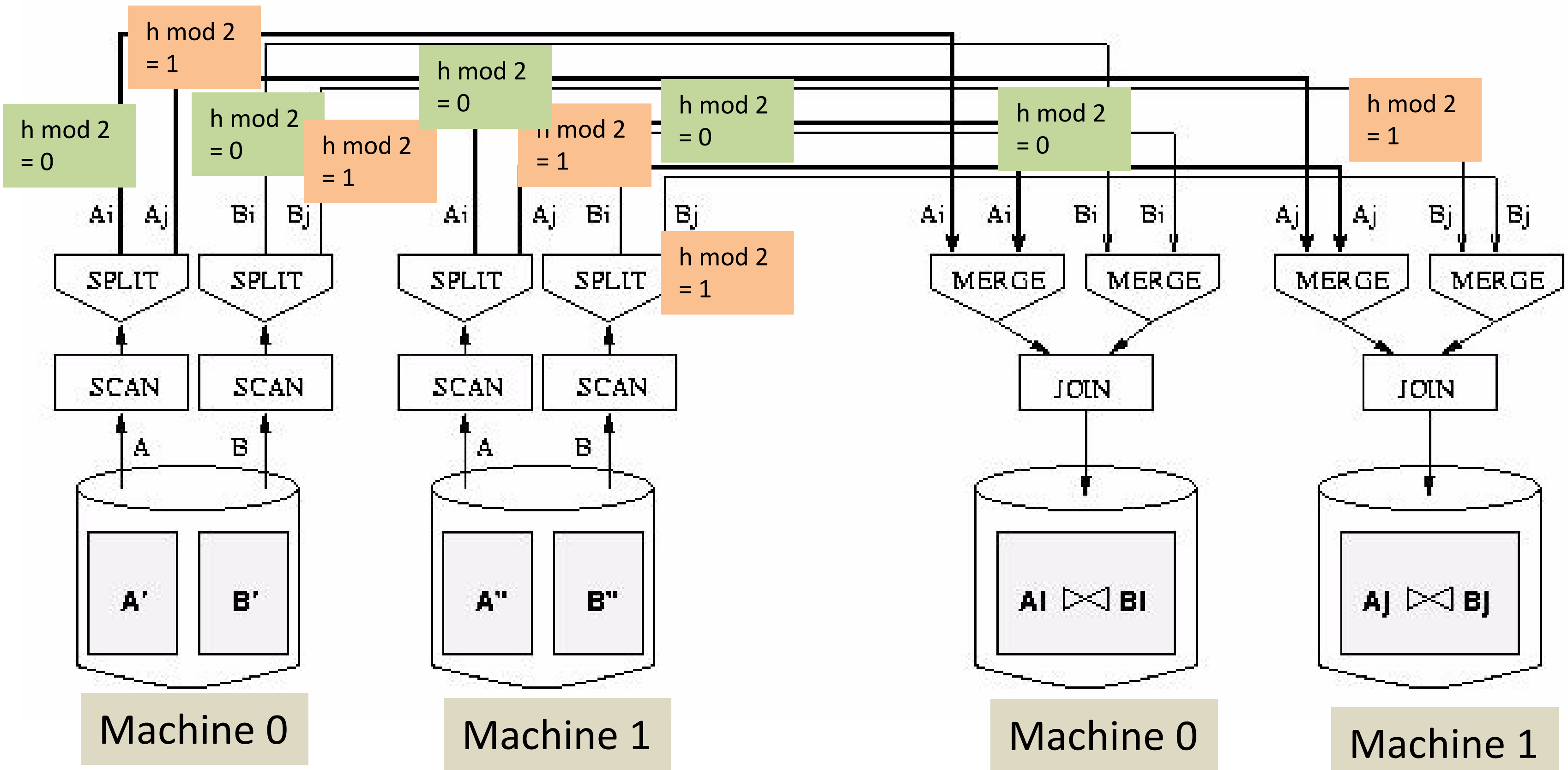
Parallel Hash Join

Phase 1



- In first phase, partitions get distributed to different sites:
  - A good hash function *automatically* distributes work evenly
- Do second phase at each site.
- Almost always the winner for equi-join

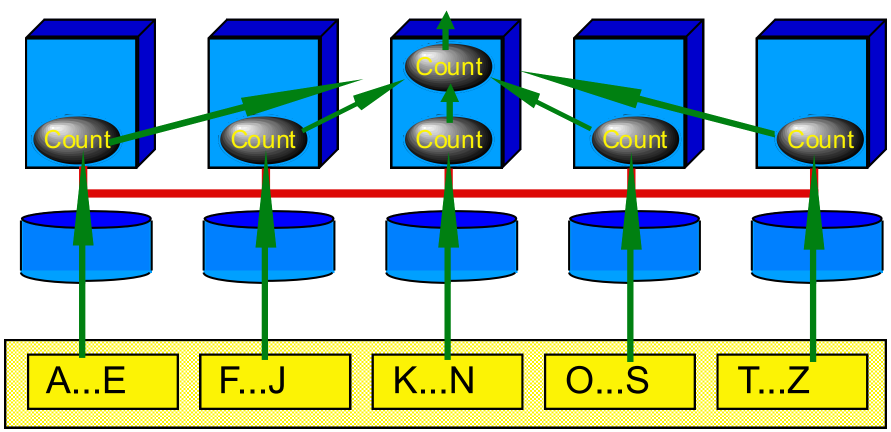
Dataflow Network for parallel Join



- Good use of split/merge makes it easier to build parallel versions of sequential join code.

Parallel Aggregates

- For each aggregate function, need a decomposition:
  - count(S) = Σ count(s(i)), ditto for sum()
  - avg(S) = (Σ sum(s(i))) / Σ count(s(i))
  - and so on...
- For group-by:
  - Sub-aggregate groups close to the source.
  - Pass each sub-aggregate to its group's site.
    - Chosen via a hash fn.

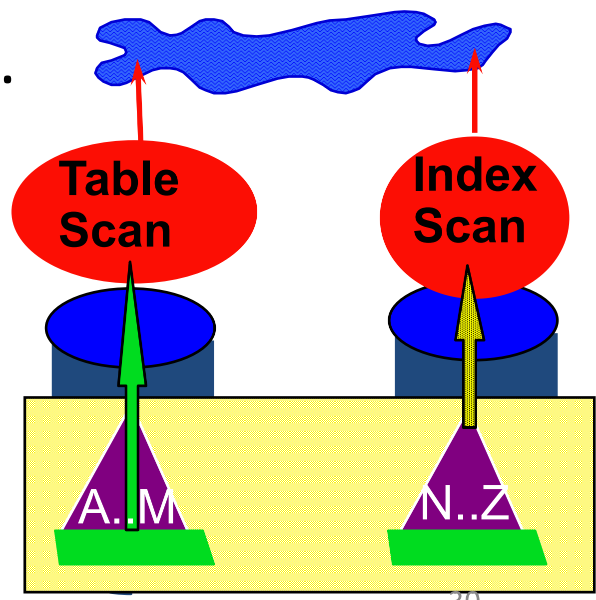


Which SQL aggregate operators are not good for parallel execution?

Jim Gray & Gordon Bell: VLDB 95 Parallel Database Systems Survey

Best serial plan may not be best ||

- Why?
- Trivial counter-example:
  - Table partitioned with local secondary index at two nodes
  - Range query: all of node 1 and 1% of node 2.
  - Node 1 should do a scan of its partition.
  - Node 2 should use secondary index.



Examples

Example problem: 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 <u>block partitioned</u> across machines).

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

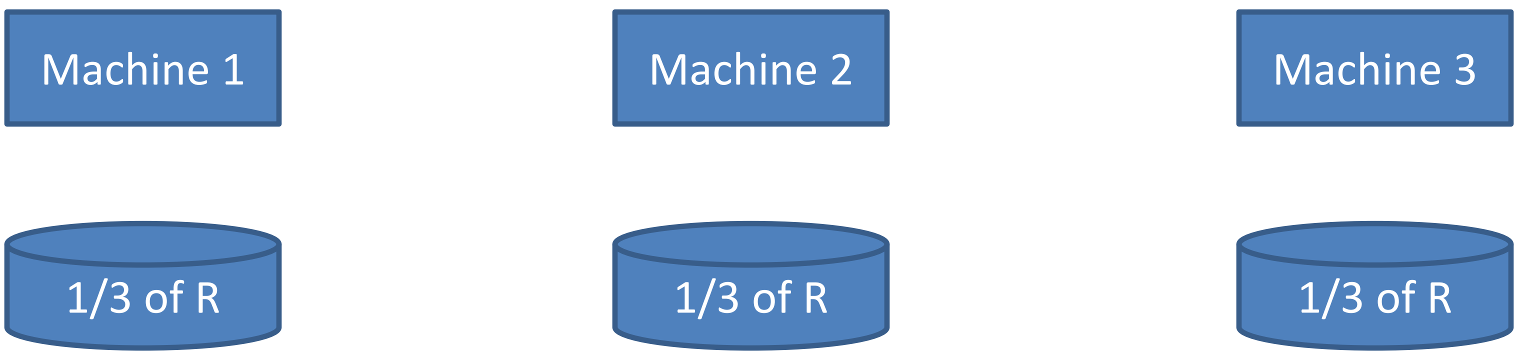
Pick an efficient plan that leverages the parallelism as much as possible.

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

We did this example
for Map-Reduce in Lecture 12!

R(a, b)

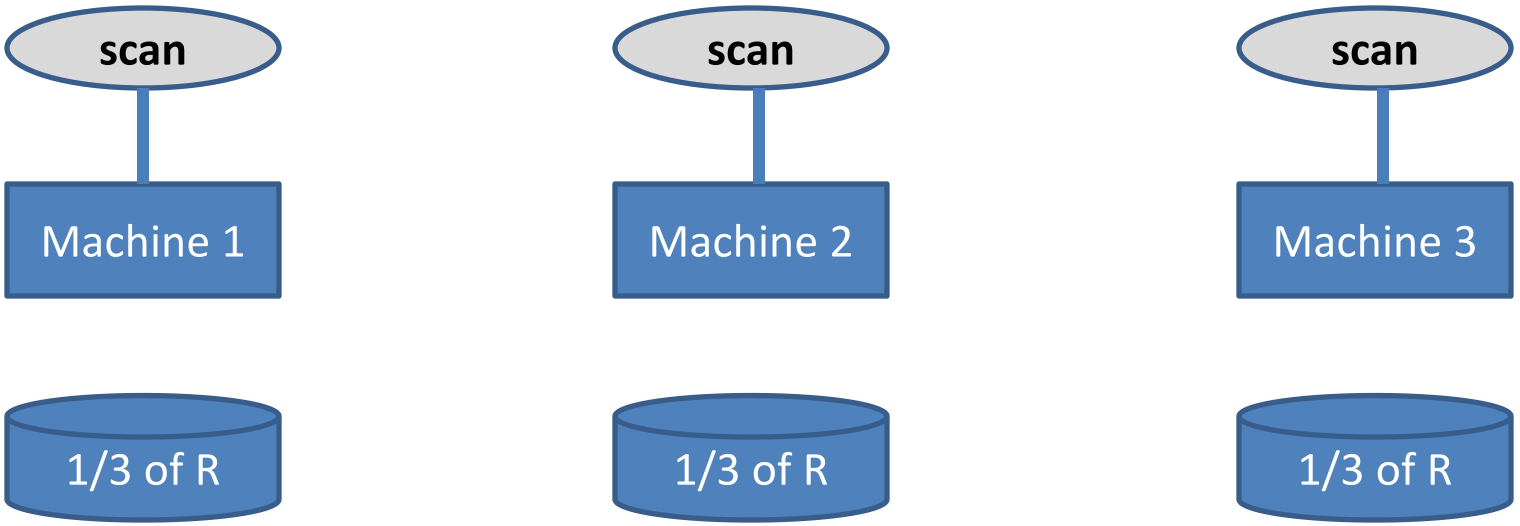
```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```



R(a, b)

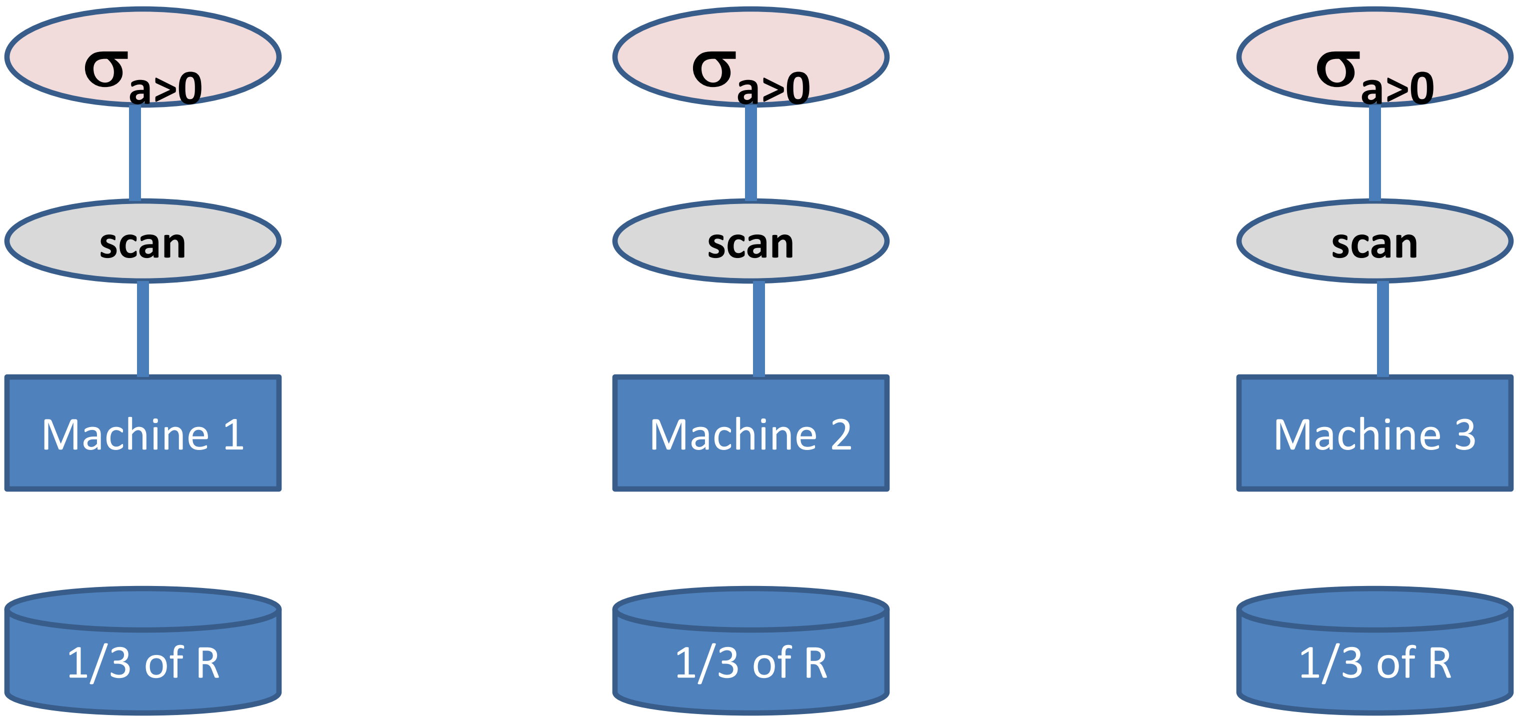
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a

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



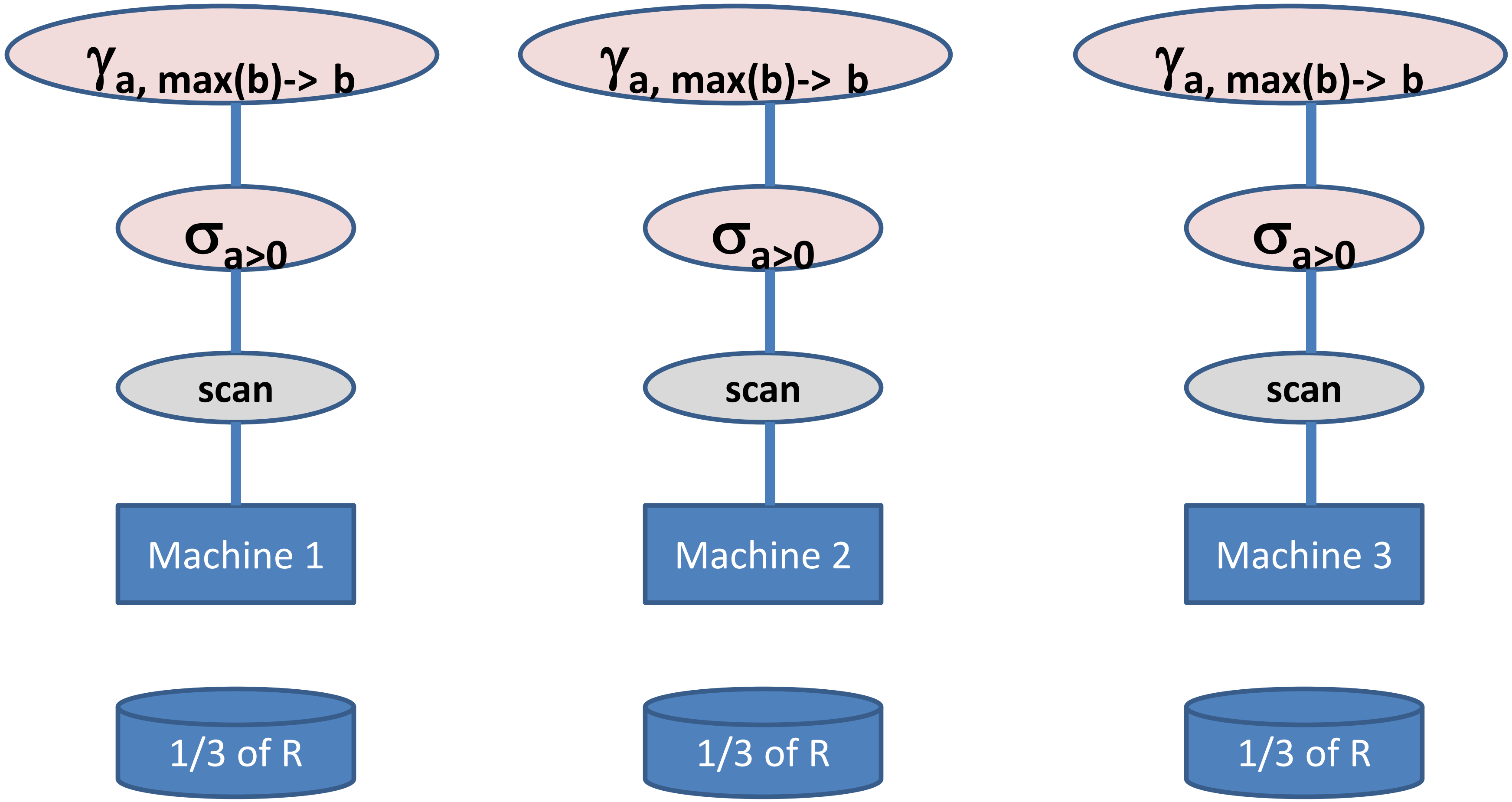
R(a, b)

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



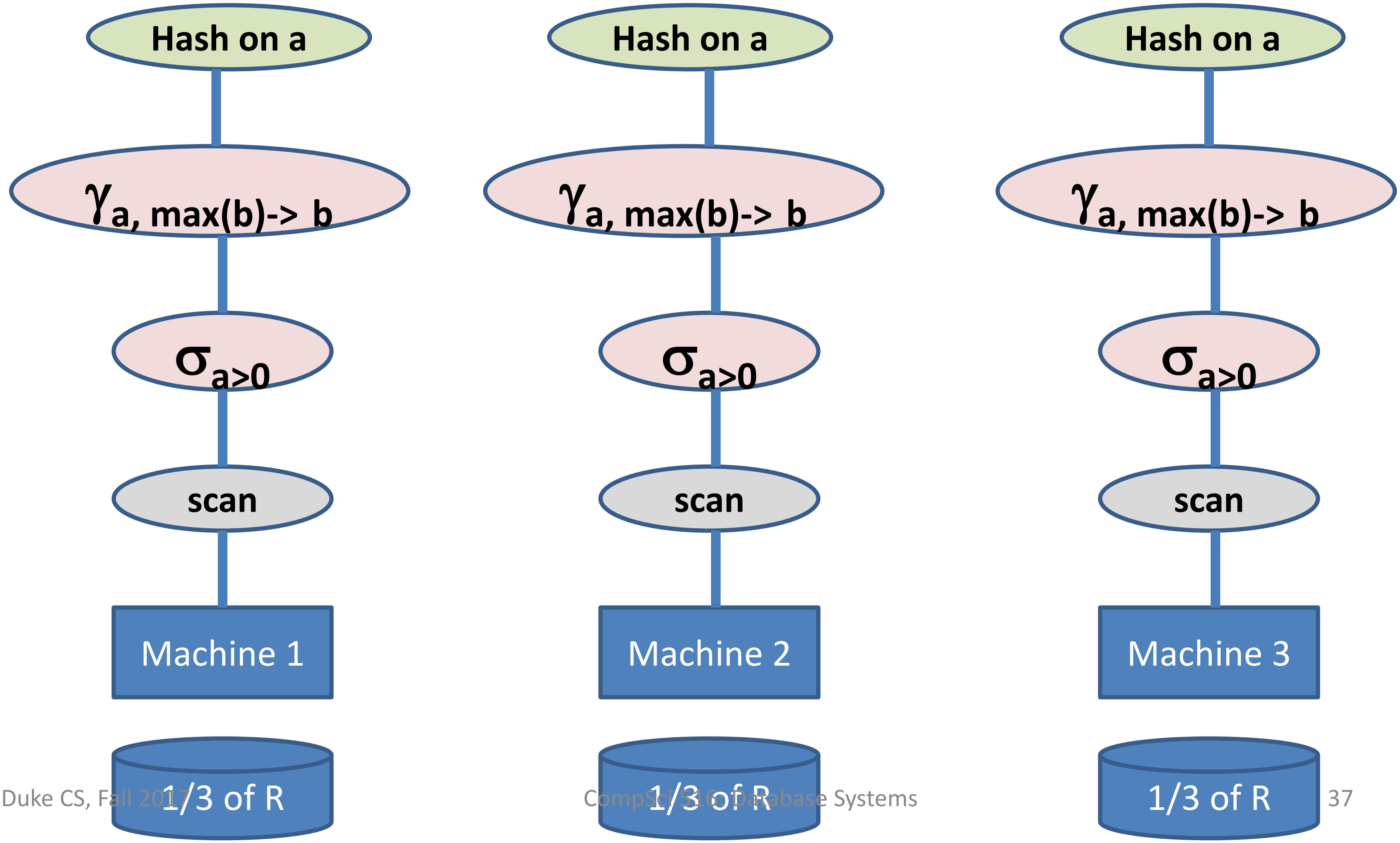
R(a, b)

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



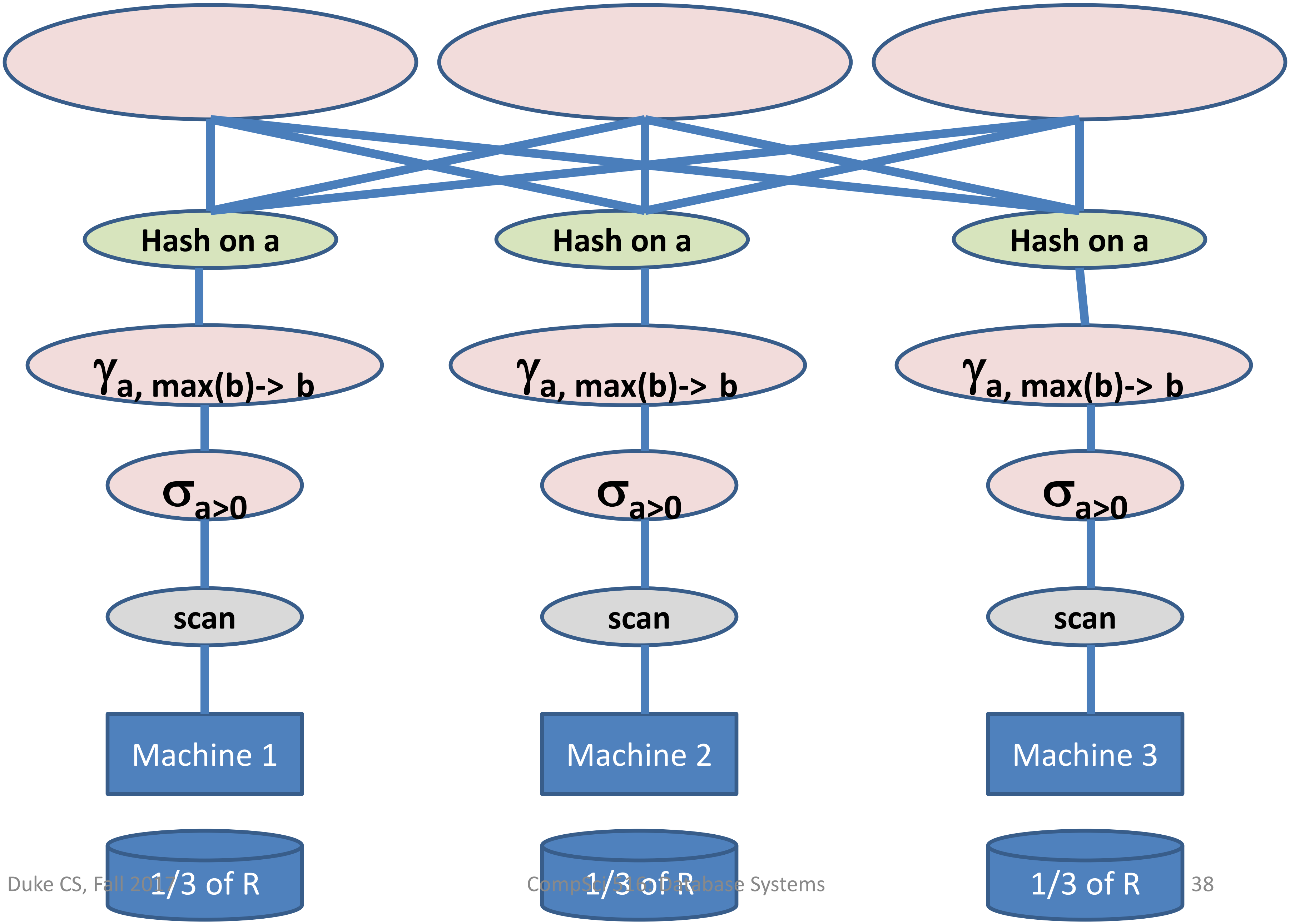
R(a, b)

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



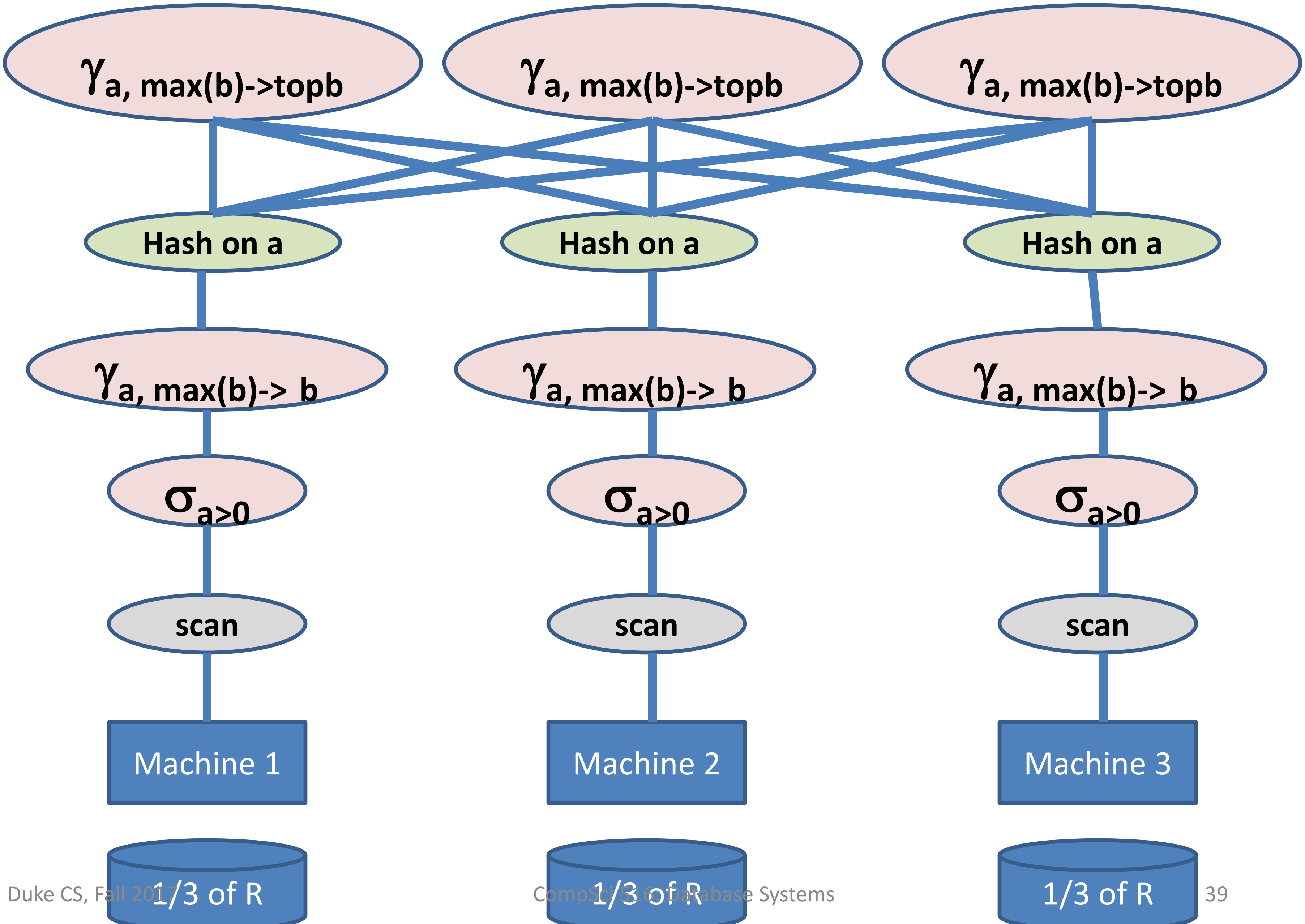
R(a, b)

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



Benefit of hash-partitioning

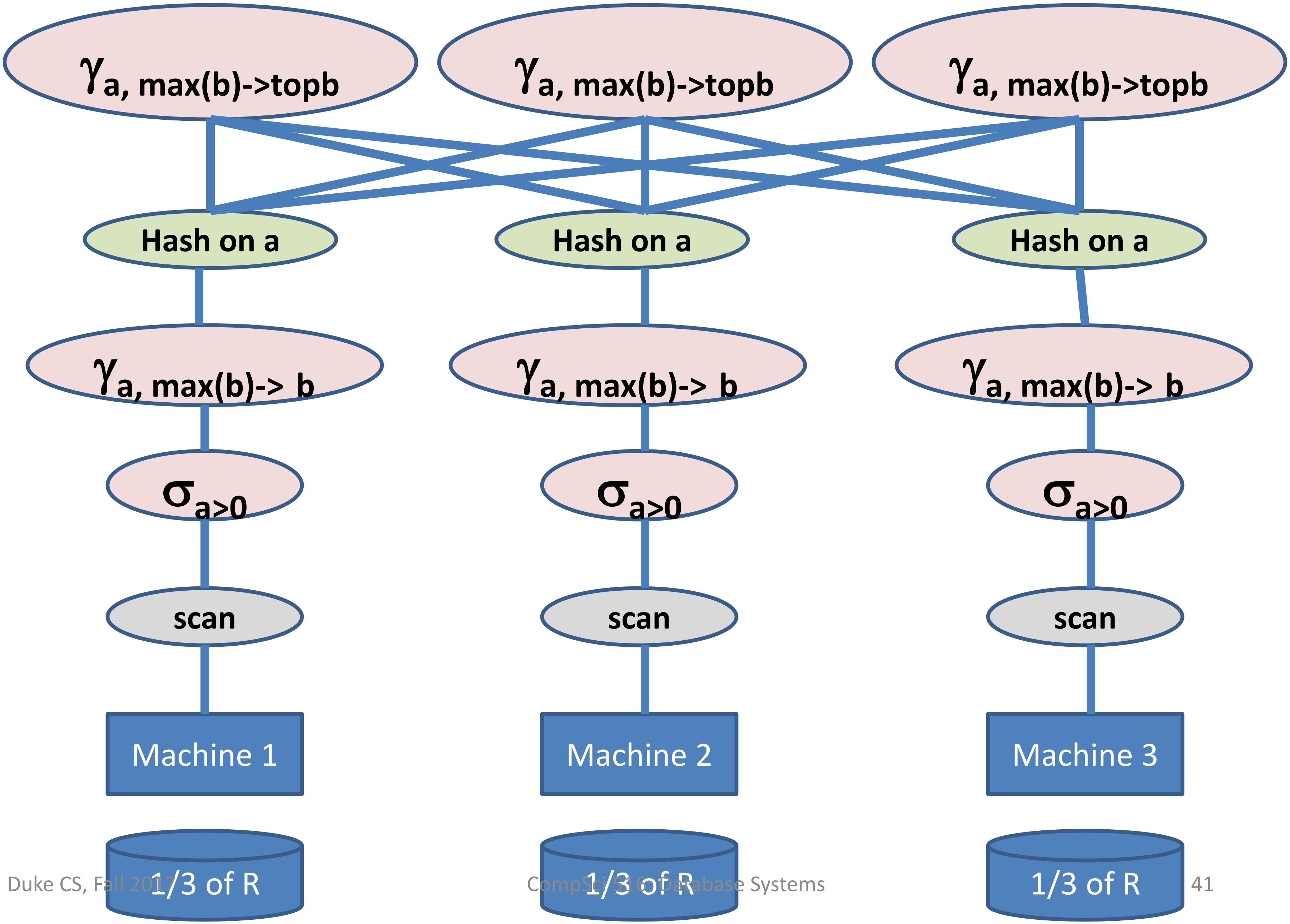
```
SELECT a, max(b) as topb
FROM R
WHERE a > 0
GROUP BY a
```

- What would change if we hash-partitioned R on R.a before executing the same query on the previous parallel DBMS and MR

- First Parallel DBMS

Prev: block-partition

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



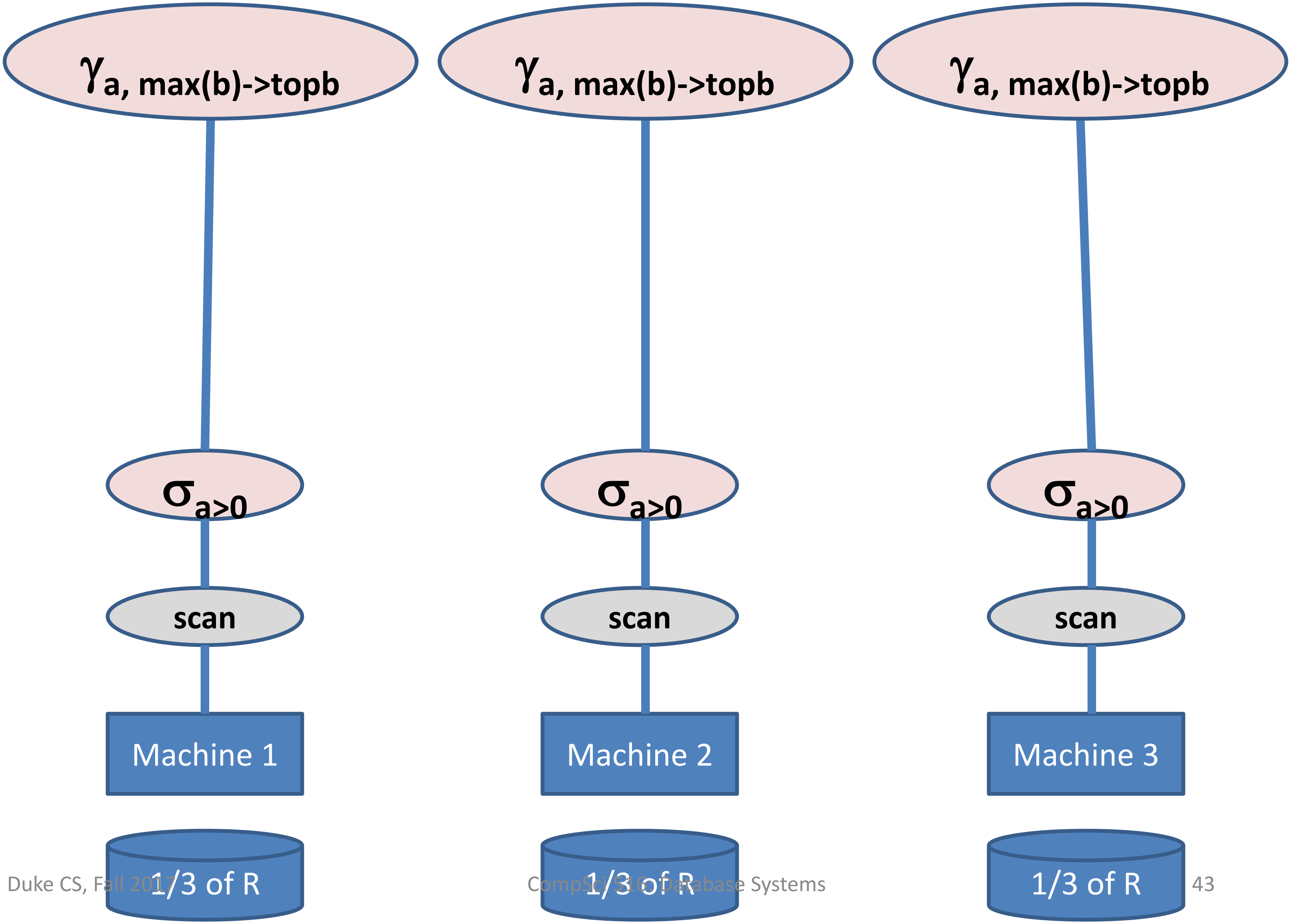
Hash-partition on a for R(a, b)

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

- 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



Benefit of hash-partitioning for Map-Reduce

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

- **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

Column Store
(slides from Lecture 19)

Row vs. Column Store

- Row store
  - store all attributes of a tuple together
  - storage like "row-major order" in a matrix
- Column store
  - store all rows for an attribute (column) together
  - storage like "column-major order" in a matrix
- e.g.
  - MonetDB, Vertica (earlier, C-store), SAP/Sybase IQ, Google Bigtable (with column groups)

What is a column-store?

+ easy to add/modify a record

- might read in unnecessary data

+ only need to read in relevant data

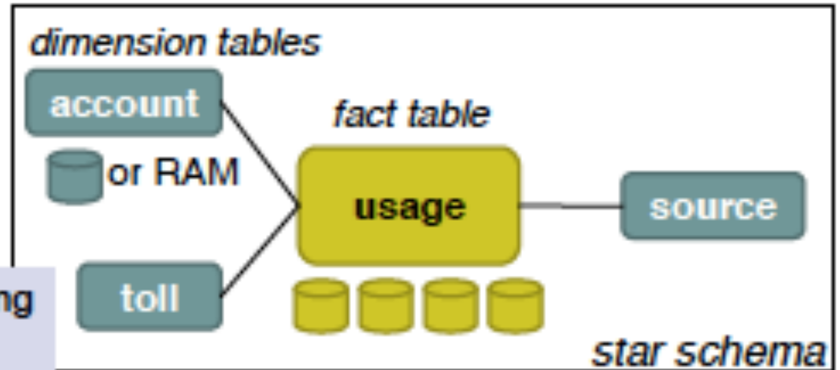
- tuple writes require multiple accesses

=> suitable for read-mostly, read-intensive, large data repositories

Ack: Slide from VLDB 2009 tutorial on Column store

Telco Data Warehousing example

- Typical DW installation
- Real-world example



"One Size Fits All? - Part 2: Benchmarking Results" Stonebraker et al. CIDR 2007

QUERY 2

```
SELECT account.account_number,
sum (usage.toll_airtime),
sum (usage.toll_price)
FROM usage, toll, source, account
WHERE usage.toll_id = toll.toll_id
AND usage.source_id = source.source_id
AND usage.account_id = account.account_id
AND toll.type_ind in ('AE'. 'AA')
AND usage.toll_price > 0
AND source.type != 'CIBER'
AND toll.rating_method = 'IS'
AND usage.invoice_date = 20051013
GROUP BY account.account_number
```

	Column-store	*Row-store*
Query 1	*2.06*	*300*
Query 2	*2.20*	*300*
Query 3	*0.09*	*300*
Query 4	*5.24*	*300*
Query 5	*2.88*	*300*

Why? Three main factors (next slides)

Ack: Slide from VLDB 2009 tutorial on Column store

Telco example explained (1/3): *read efficiency*

row store | **column store**

read pages containing entire rows

one row = 212 columns!

is this typical? (it depends)

What about vertical partitioning? (it does not work with ad-hoc queries)

read only columns needed

in this example: 7 columns

caveats:
- "select * " not any faster
- clever disk prefetching
- clever tuple reconstruction

Ack: Slide from VLDB 2009 tutorial on Column store

Telco example explained (2/3): *compression efficiency*

- Columns compress better than rows
  - Typical row-store compression ratio 1 : 3
  - Column-store 1 : 10
- Why?
  - Rows contain values from different domains
    => more entropy, difficult to dense-pack
  - Columns exhibit significantly less entropy
  - Examples:

Male, Female, Female, Female, Male
1998, 1998, 1999, 1999, 1999, 2000

  - Caveat: CPU cost (use lightweight compression)

Ack: Slide from VLDB 2009 tutorial on Column store

Telco example explained (3/3): *sorting & indexing efficiency*

- Compression and dense-packing free up space
  - Use multiple overlapping column collections
  - Sorted columns compress better
  - Range queries are faster
  - Use sparse clustered indexes

Ack: Slide from VLDB 2009 tutorial on Column store