

Database System Internals

Intro to Parallel DBMSs

Paul G. Allen School of Computer Science and Engineering
 University of Washington, Seattle

Announcements

- Lab 3 due date extended by 2 days
 - Due tonight without late days, Friday if using late days
 - Make sure to look at Lab 4 even if taking extra time on Lab 3!

What We Have Already Learned

- Phase 1: Query Execution
 - Data Storage and Indexing
 - Buffer management
 - Query evaluation and operator algorithms
 - Query optimization
- Phase 2: Transaction Processing
 - Concurrency control: pessimistic and optimistic
 - Transaction recovery: undo, redo, and undo/redo
- Phase 3: Parallel Processing & Distributed Transactions

Where We Are Headed Next

- Scaling the execution of a query
 - Parallel DBMS
 - MapReduce
 - Spark

- Scaling transactions
 - Distributed transactions
 - Replication

DATA & AI LANDSCAPE 2019

INFRASTRUCTURE

HADOOP ON-PREMISE
cloudera, Hortonworks, MAPR, Pivotal, IBM InfoSphere, jethro

HADOOP IN THE CLOUD
AWS, Microsoft Azure, Google Cloud, SAP Cloud Platform, IBM InfoSphere BigInsights, Qlik, Oracle, CAZENA

STREAMING / IN-MEMORY
Amazon Kinesis, databricks, SAP Cloud Platform, Oracle, confluent, streamio, hazelcast, GridGain, GIGASPACE, Wallaroo, FASTDATA, Ix

ANALYTICS & MACHINE INTELLIGENCE

DATA ANALYST PLATFORMS
Microsoft, pentaho, alteryx, Digital Reasoning, GUAVUS, AYASDI, ATTIVO, Datameer, Incorta, interana, MODE, ENDOR, sisu, switchboard, Starburst

DATA SCIENCE PLATFORMS
IBM, databricks, dataiku, DOMINO, rapidminer, TIBCO, ANACONDA, SAS, KNIIME, MathWorks

APPLICATIONS - ENTERPRISE

SALES
Salesforce, CHORUS, INSIDESALES.COM, people.ai, conversica, clari, aviso, tact.ai, PROOPS, fuse/machines, Clearbit

MARKETING - B2B
RADIUS, App Annie, EVERSTING, Lattice, HINTIGO, sense, tubular, JE N G A I O, KNOTCH, mrp

MARKETING - B2C
Zeta, bloomreach, SendGrid, braze, ACTIONIQ, BLUECORE, CONTENTGOLD, TEALUM, importio, Amploro, amplicity, QUANTIFIND, Simon, Jiffys, PERSADO, rmesh

CUSTOMER EXPERIENCE / SERVICE
qualtrics, MEDALLIA, SurveyMonkey, CLARABRIDGE, Zendesk, Customer, INTERCOM, Orbit, LIVEPERSON, Gainight, pendo, HEAD, Amplitude, Watson Assistant, DigitalGenius, AS APP, ada, AUTOMAT, ahnti, CallDesk, metomi, tSME, Frame.io, talla, Kasisto

ENTERPRISE PRODUCTIVITY
SLACK, ORACLE, GURU, lumiatra, DFFBOT, clara, Kasisto

APPLICATIONS - INDUSTRY

ADVERTISING
AppNexus, Criteo, Oracle, theTradeDesk, distillery, TapR, TapR, TapR

EDUCATION
Knewton, Clever, Clarica, kidaptive, MINDRAMA, knowre, gradescope

REAL ESTATE
REDFIN, Opendoor, VTS, CREDIFY, GEOPHY, STREETCREDITS, COMPSTAK, SPACEMARKER

GOVT
OPENGOV, mark43, FIS, LiveStories, Passport, SmartProcure, STREETCREDITS, QuantSet

INTELLIGENCE
Palantir, Dataminr, Quid, PRIMER, FORGE

FINANCE - INVESTING
KENSIG, Quantopian, ADDRAR, NUBISA, ISENTUM, ALGORO, FlarePack, APAGATA

FINANCE - LENDING
oneck, affirm, JIANPUAI, Kreedtech, AVANT, TALA, CLEARBANC, Upstart, 100credit, Wells, Wacash, cire, cignifi

INSURANCE
Insomnis, Lemonade, Hippo, Snft Technology, ROOT, TRAVELER, CAPE

CROSS-INFRASTRUCTURE/ANALYTICS

aws, Google Cloud, Microsoft, Pivotal, IBM, SAP, Oracle, NetApp, synsort, MAPR, cloudera

OPEN SOURCE

FRAMEWORKS
Spark, Flink, YARN, TEZ, Mesos, Hadoop, CDAP, Redhat, Helix

QUERY / DATA FLOW
Spark, SQL, Presto, SLAMDATA, GraphQL

DATA ACCESS & DATABASES
Cassandra, mongoDB, redis, Cockroach Labs, druid, Riik, Aerospike, Oracle, SciDB, Riik, Aerospike, Oracle, SciDB, Riik, Aerospike, Oracle, SciDB

ORCHESTRATION & MGMT
talend, Apache Airflow, Mesos, etcd, Kong

STREAMING & MESSAGING
Spark, nifi, kafka, beam, storm, Apache RocketMQ

STAT TOOLS & LANGUAGES
Scala, Python, R, Julia, Studio, SciPy

AI OPS & INFRA
miflow, Kubeflow, Deep, DVC, SELDON, Polysyn

AI / MACHINE LEARNING / DEEP LEARNING
TensorFlow, Keras, PyTorch, OpenAI, DM, TK, theano, H2O, MAJIB, Caffe, Microsoft Cognitive Toolkit, DIMSUM, FeatureFu

SEARCH
elasticsearch, Solr

LOGGING & MONITORING
elasticsearch, kibana, logstash, fluentbit, fluentd, Grafana, Vector

VISUALIZATION
matplotlib, TensorBoard, seaborn, D3.js, ANACONDA

COLLABORATION
BeakerX, KNOX, Apache Ranger, Sentry, OCCURFULO

DATA SOURCES & APIs

HEALTH
Apple, VALIDIC, practicefusion, fitbit, GARMIN, kinso, IMMIC

IOT
GE Digital, UPTAKE, thingworx, helium, samsara, estimate

FINANCIAL & ECONOMIC DATA
Bloomberg, THOMSON REUTERS, DOW JONES, SEP CAPITAL IQ, CBINSIGHTS, FLAID, SECOND MEASURE, INVESTMENT VOICELINE, Gestimio, PREMISE, Quant, Eagle Alpha, Stocktwits, xignite, Thinknum, earnest, predata

AIR / SPACE / SEA
Orbital Insight, planet, AIRBOTICS, spire, kespri, UNDERSTORY, telluslabs, WINDWARD, DroneDeploy, MarineTraffic

PEOPLE / ENTITIES
ackion, Experian, EPISON, InsideView, Crismon Hexagon, BASIS, Quantcast, SAFEGRAPH

LOCATION INTELLIGENCE
FOURSQUARE, sense360, PlaceIQ, esri, Mopiliary, streets, cuebiq, Radar, OpenStreetMap

OTHER
DATA.GOV, IMAGENET, CRUX, ugrafruit.io

DATA RESOURCES

DATA SERVICES
OPERA, fractal, kaggle, DataKind, innodcxus

INCUBATORS & SCHOOLS
PLURALSIGHT, DataCamp, DataElite, INSIGHT, The Data Incubator, METIS

RESEARCH
OpenAI, facebook research, MIRI, VECTOR INSTITUTE, ALLEN INSTITUTE FOR ARTIFICIAL INTELLIGENCE

July 16, 2019 - FINAL 2019 VERSION

© Matt Turck (@mattturck), Lisa Xu (@lisaxu92), & FirstMark (@firstmarkcap) mattturck.com/data2019

FIRSTMARK
EARLY STAGE VENTURE CAPITAL

How to Scale the DBMS?

- Can easily replicate the web servers and the application servers
- We cannot so easily replicate the database servers, because the database is unique
- We need to design ways to **scale up the DBMS**

Building Our Parallel DBMS

Data model?

Relational
(SimpleDB!)

Building Our Parallel DBMS

Data model?

Relational
(SimpleDB!)

Scaleup goal?

Scaling Transactions Per Second

- OLTP: Transactions per second
“Online Transaction Processing”
- Amazon
- Facebook
- Twitter
- ... your favorite Internet application...
- Goal is to increase transaction throughput
- We will get back to this next week

Scaling Single Query Response Time

- OLAP: Query response time
“Online Analytical Processing”
- Entire parallel system answers one query
- Goal is to improve query runtime
- Use case is analysis of massive datasets

Volume alone is not an issue

- Relational databases *do* parallelize easily; techniques available from the 80's
 - Data partitioning
 - Parallel query processing
- SQL is *embarrassingly parallel*
 - We will learn how to do this!

New **workloads** are an issue

- Big volumes, small analytics
 - OLAP queries: join + group-by + aggregate
 - Can be handled by today's RDBMSs
- Big volumes, big analytics
 - More complex Machine Learning, e.g. click prediction, topic modeling, SVM, k-means
 - Requires innovation – Active research area

Building Our Parallel DBMS

Data model?

Relational

Scaleup goal?

OLAP

Building Our Parallel DBMS

Data model?

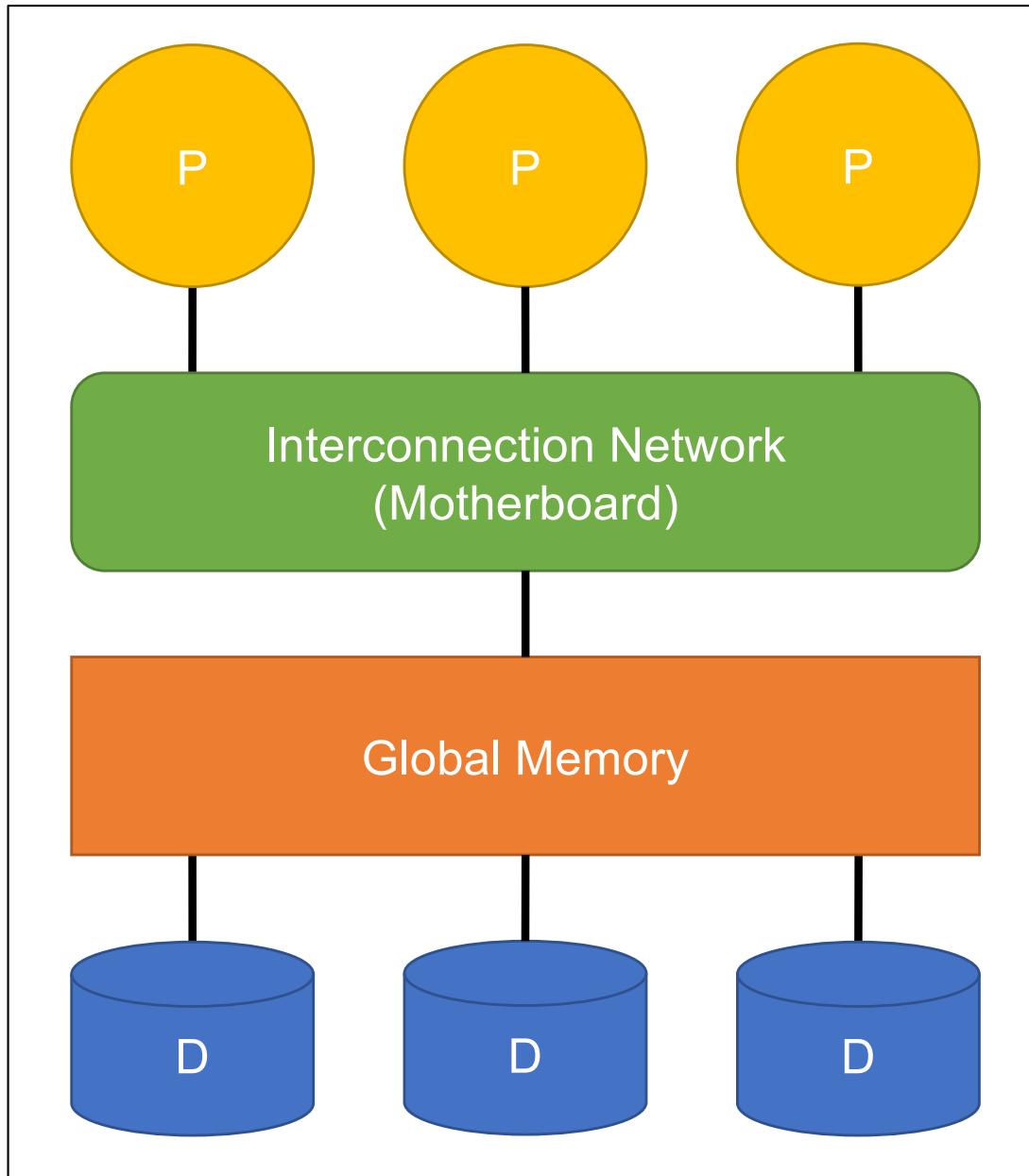
Relational

Scaleup goal?

OLAP

Architecture?

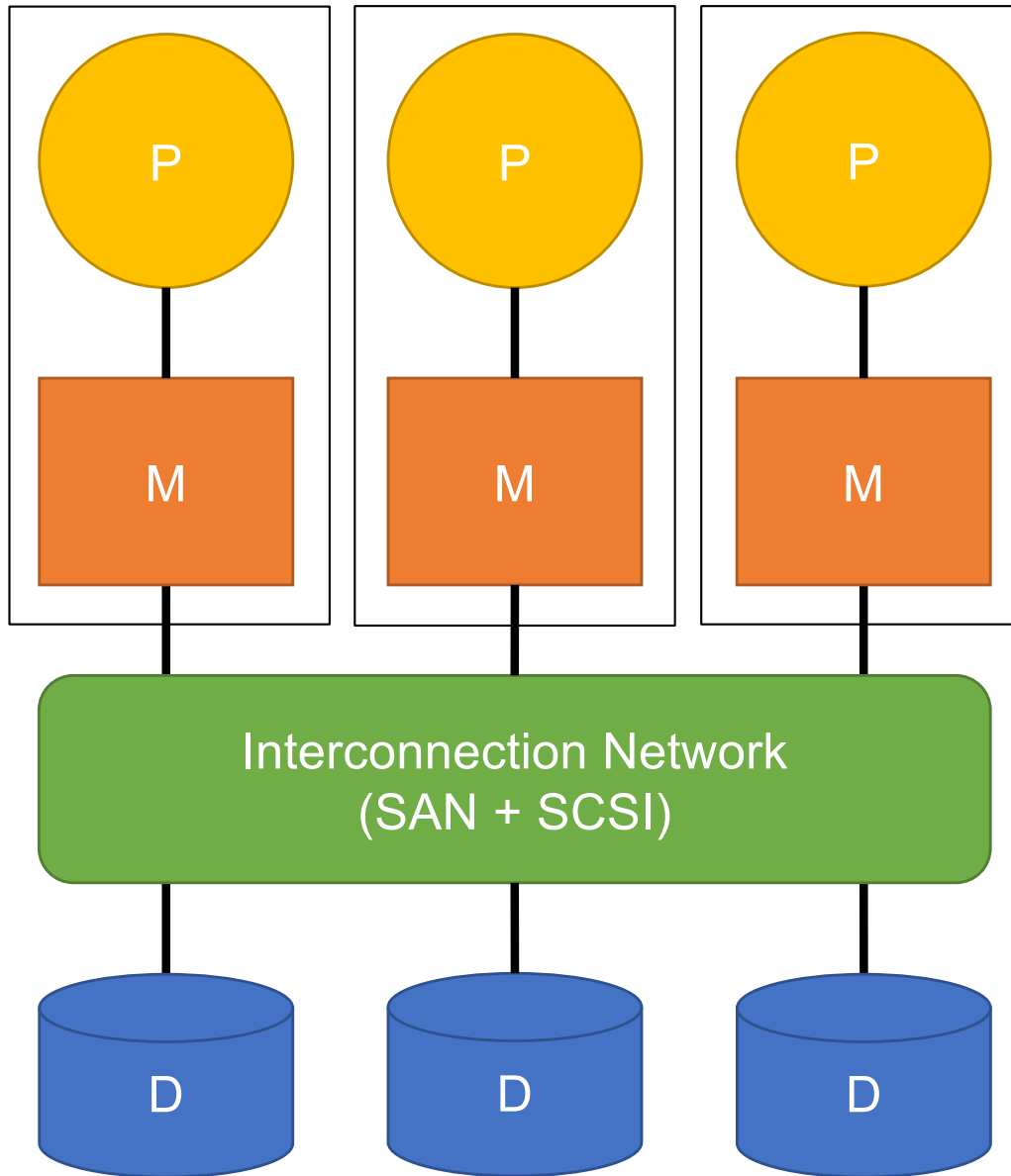
Shared-Memory Architecture



- Shared main memory and disks
- Your laptop or desktop uses this architecture
- **Expensive to scale**
- **Easiest to implement on**



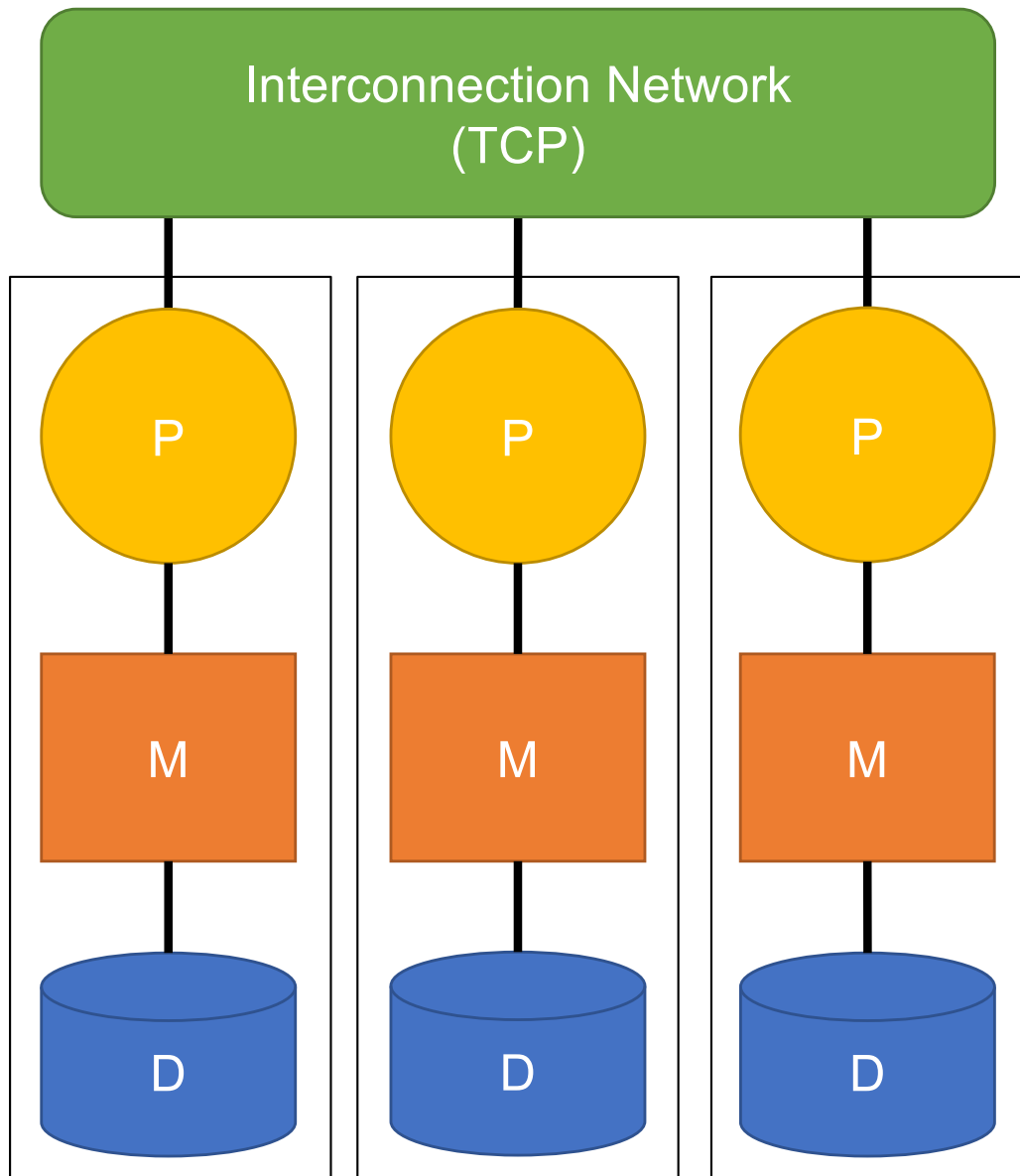
Shared-Disk Architecture



- Only shared disks
- No contention for memory and high availability
- Typically 1-10 machines

ORACLE[®]
DATABASE

Shared-Nothing Architecture



- Uses cheap, commodity hardware
- No contention for memory and high availability
- Theoretically can **scale infinitely**
- Hardest to implement on

teradata.

APACHE
Spark[™]

MySQL[™] Cluster

Building Our Parallel DBMS

Data model?

Relational

Scaleup goal?

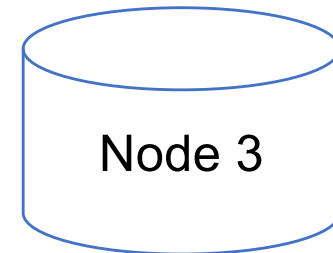
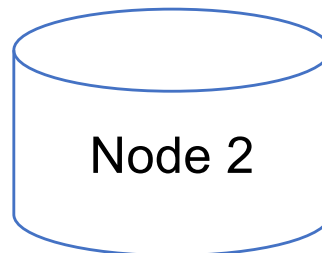
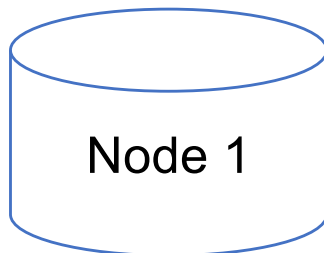
OLAP

Architecture?

Shared-Nothing

Shared-Nothing Execution Basics

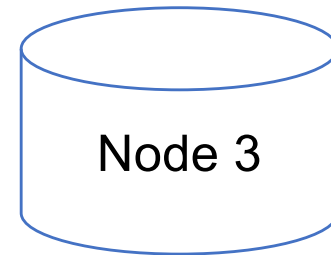
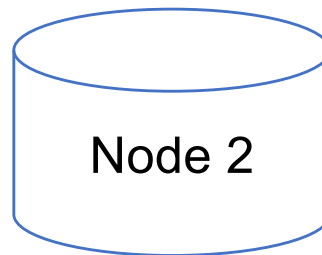
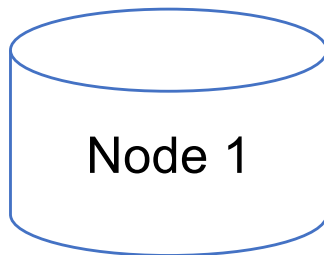
- Multiple DBMS instances (= processes) also called “nodes” execute on machines in a cluster
 - One node plays role of the coordinator
 - Other nodes play role of workers
- Workers execute queries
 - Typically **all workers execute the same plan**
 - Workers can execute multiple queries at the same time



Shared-Nothing Database

We will assume a system that consists of multiple commodity machines on a common network

New problem: **Where does the data go?**

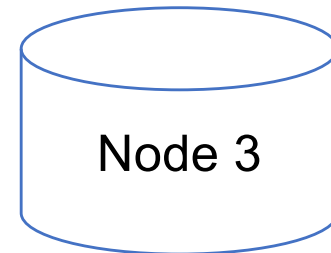
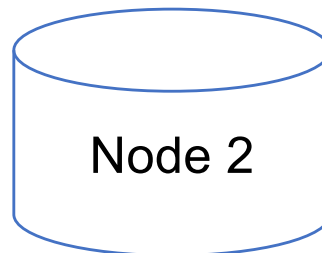
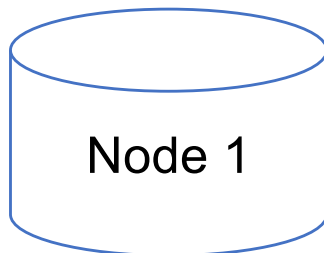


Shared-Nothing Database

We will assume a system that consists of multiple commodity machines on a common network

New problem: **Where does the data go?**

The answer will influence our execution techniques

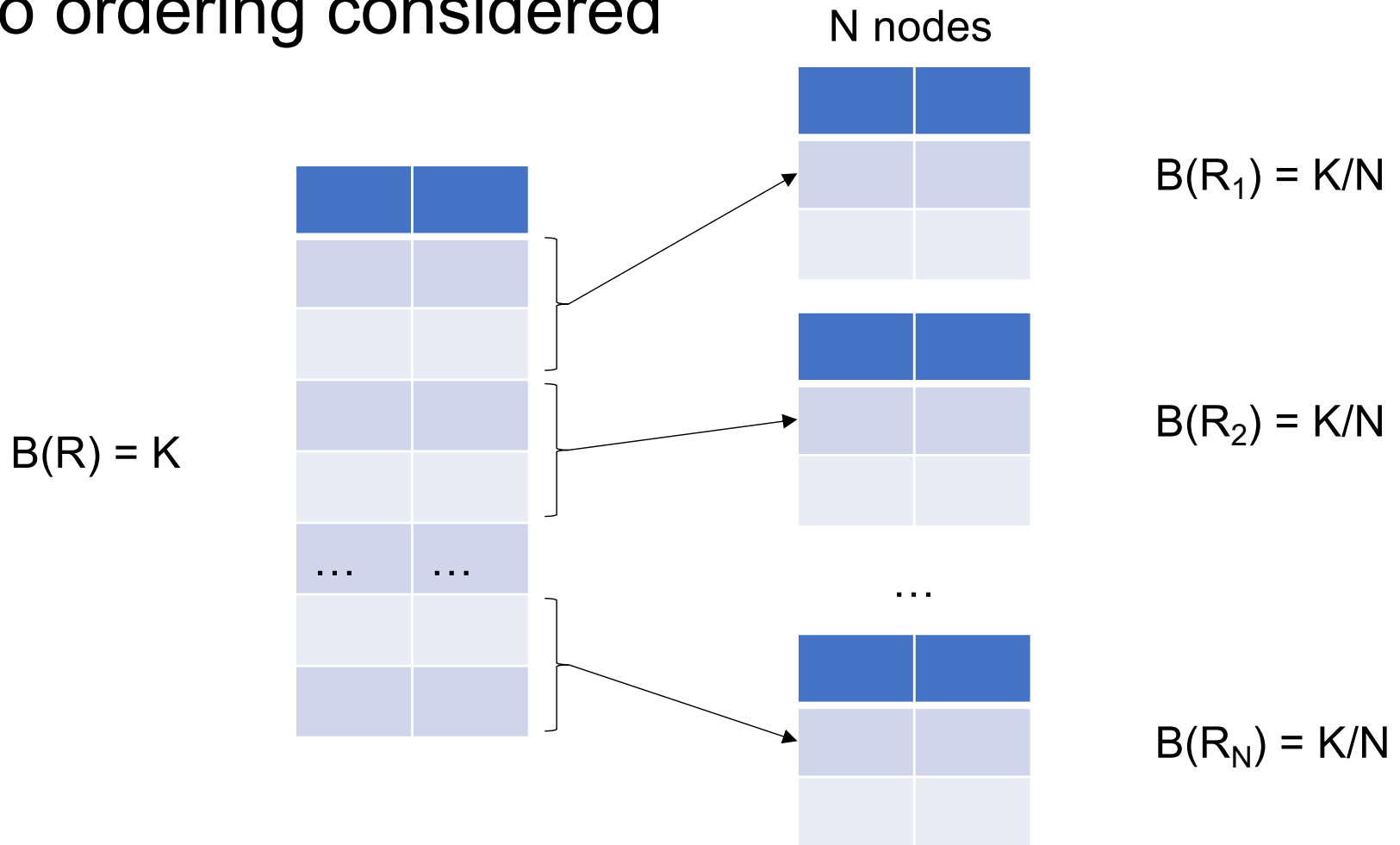


Option 1: Unpartitioned Table

- Entire table on just one node in the system
- Will bottleneck any query we need to run in parallel
- We choose partitioning scheme to divide rows among machines

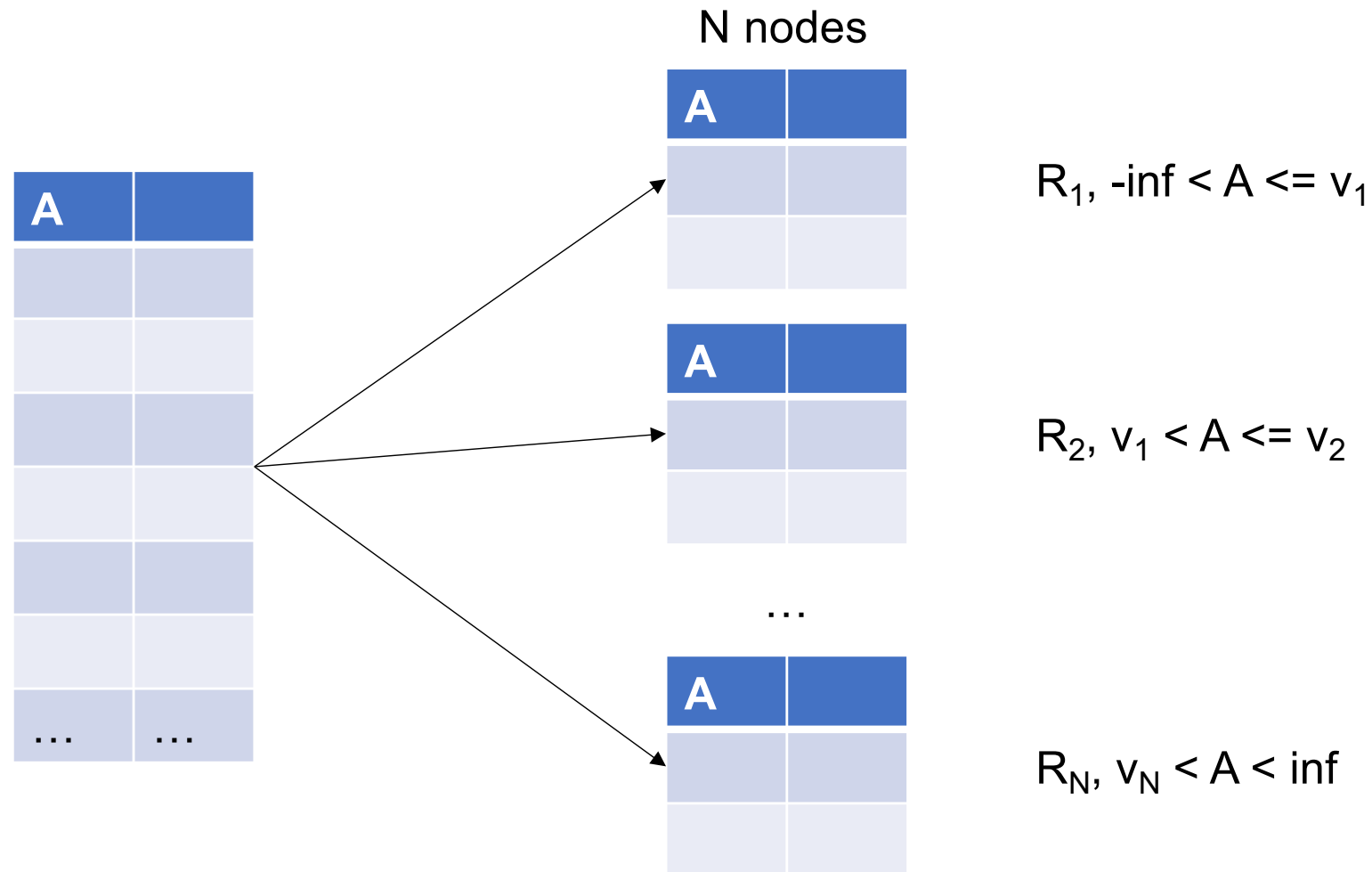
Option 2: Block Partitioning

Tuples are horizontally (row) partitioned by raw size with no ordering considered



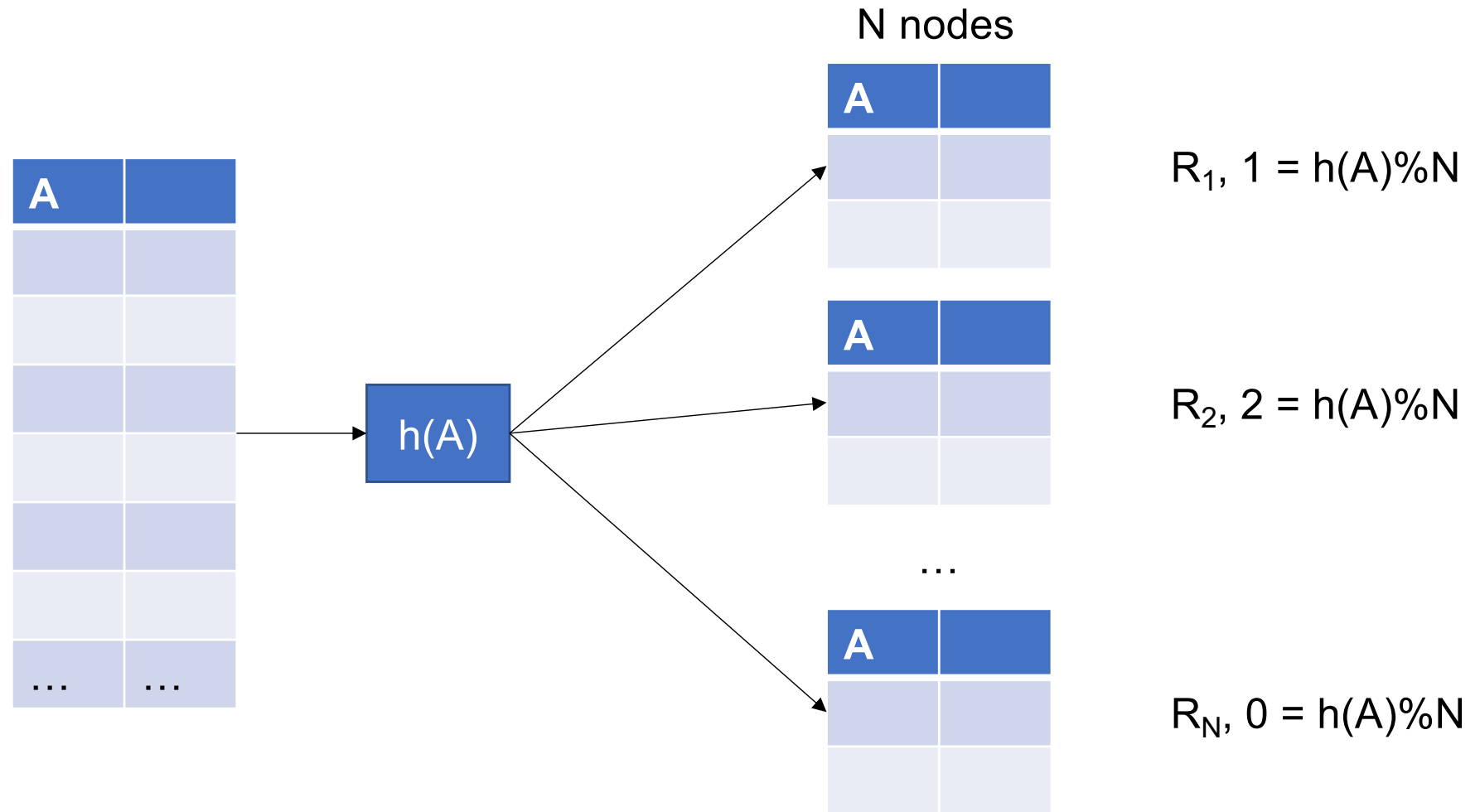
Option 3: Range Partitioning

Node contains tuples in chosen attribute ranges



Option 4: Hash Partitioning

Node contains tuples with chosen attribute hashes



Skew: The Justin Bieber Effect

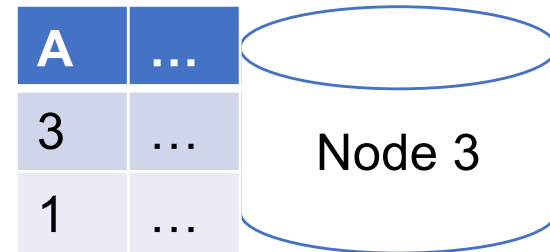
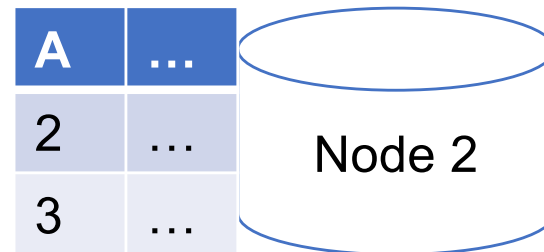
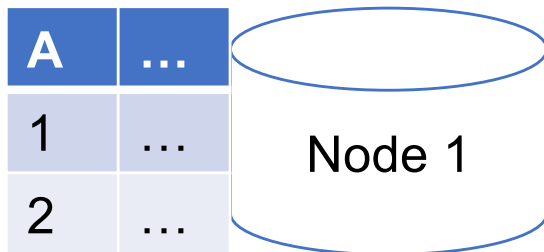
- Hashing data to nodes is very good when the attribute chosen better approximates a uniform distribution
- Keep in mind: Certain nodes will become bottlenecks if a poorly chosen attribute is hashed

Parallel Selection

Assume:

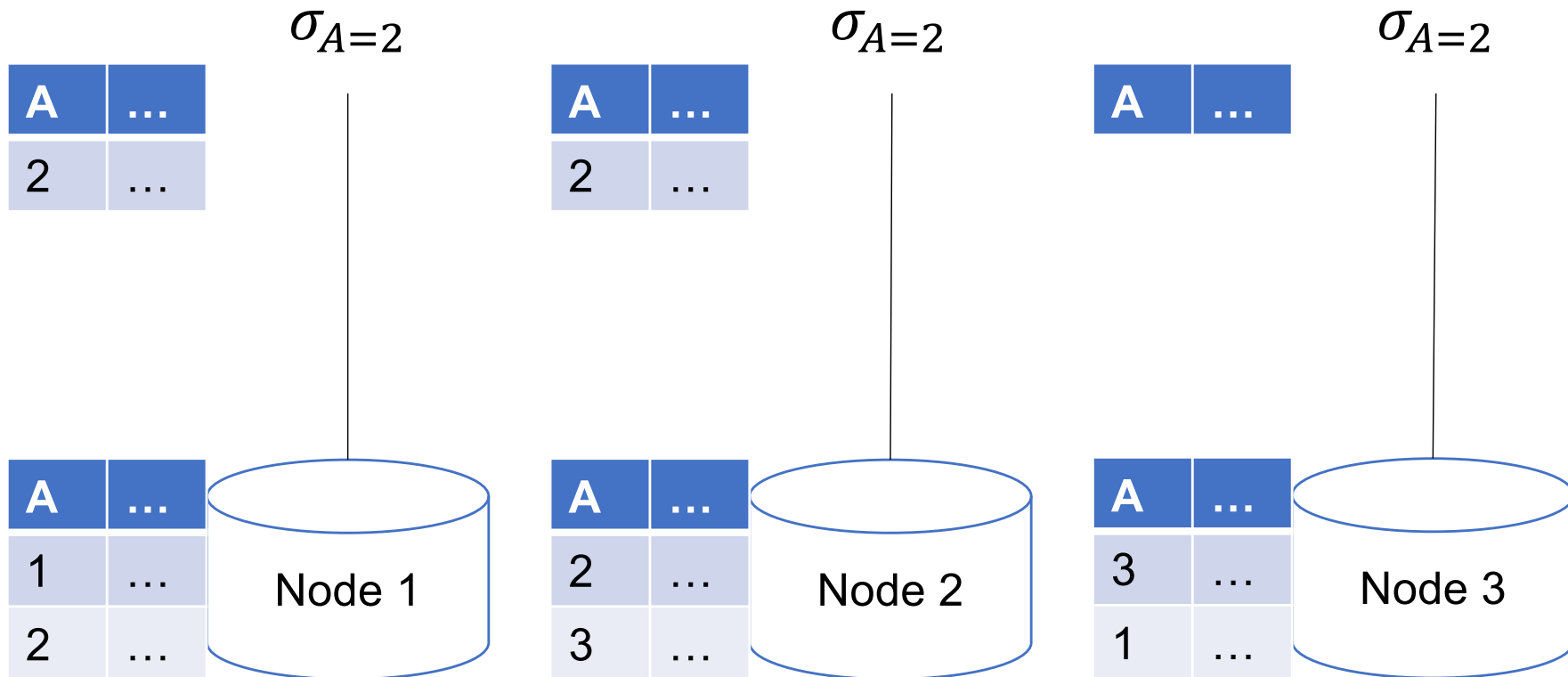
R is block partitioned

```
SELECT *  
  FROM R  
  WHERE A = 2
```



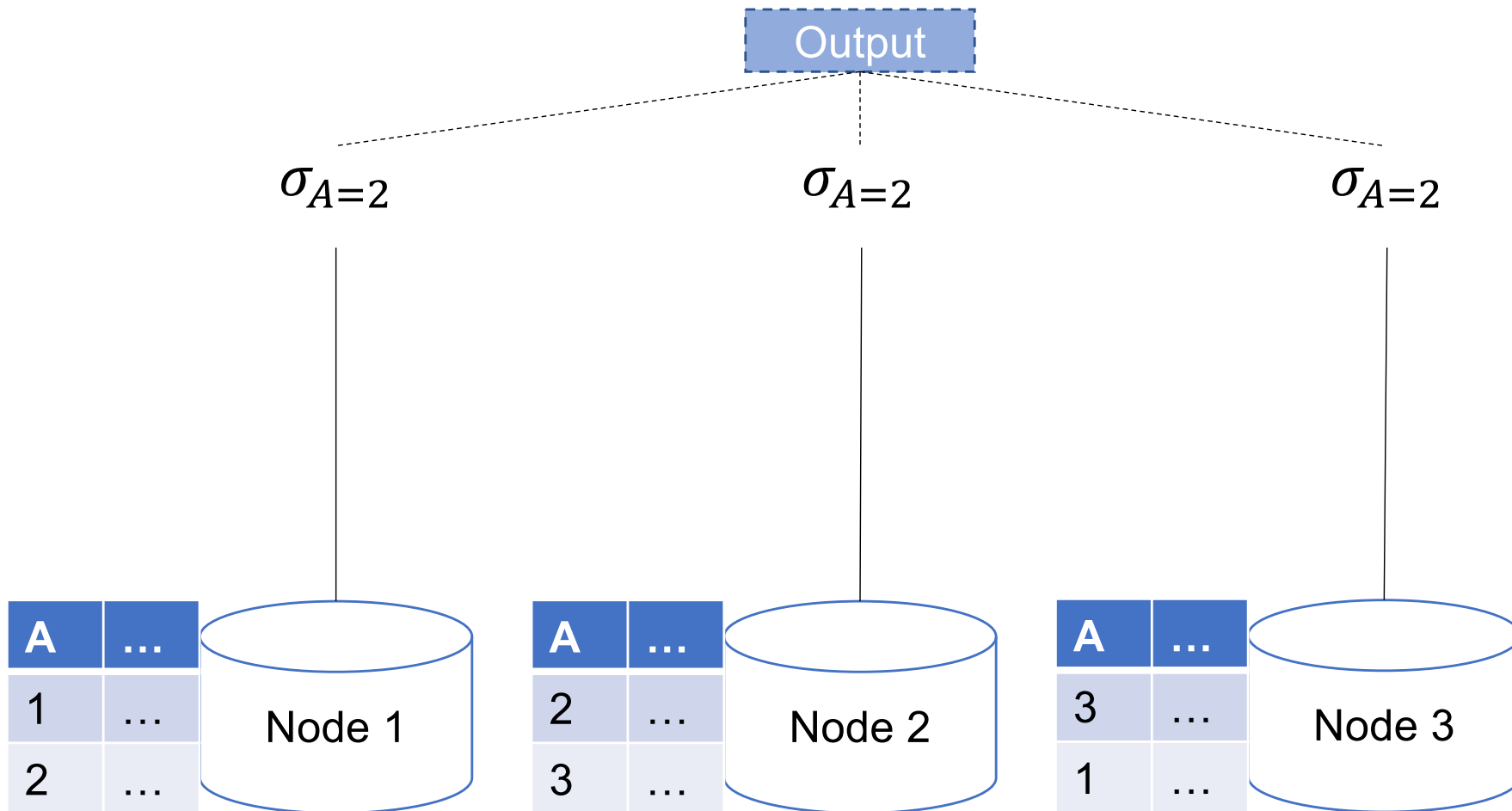
Parallel Selection

```
SELECT *  
FROM R  
WHERE A = 2
```



Implicit Union

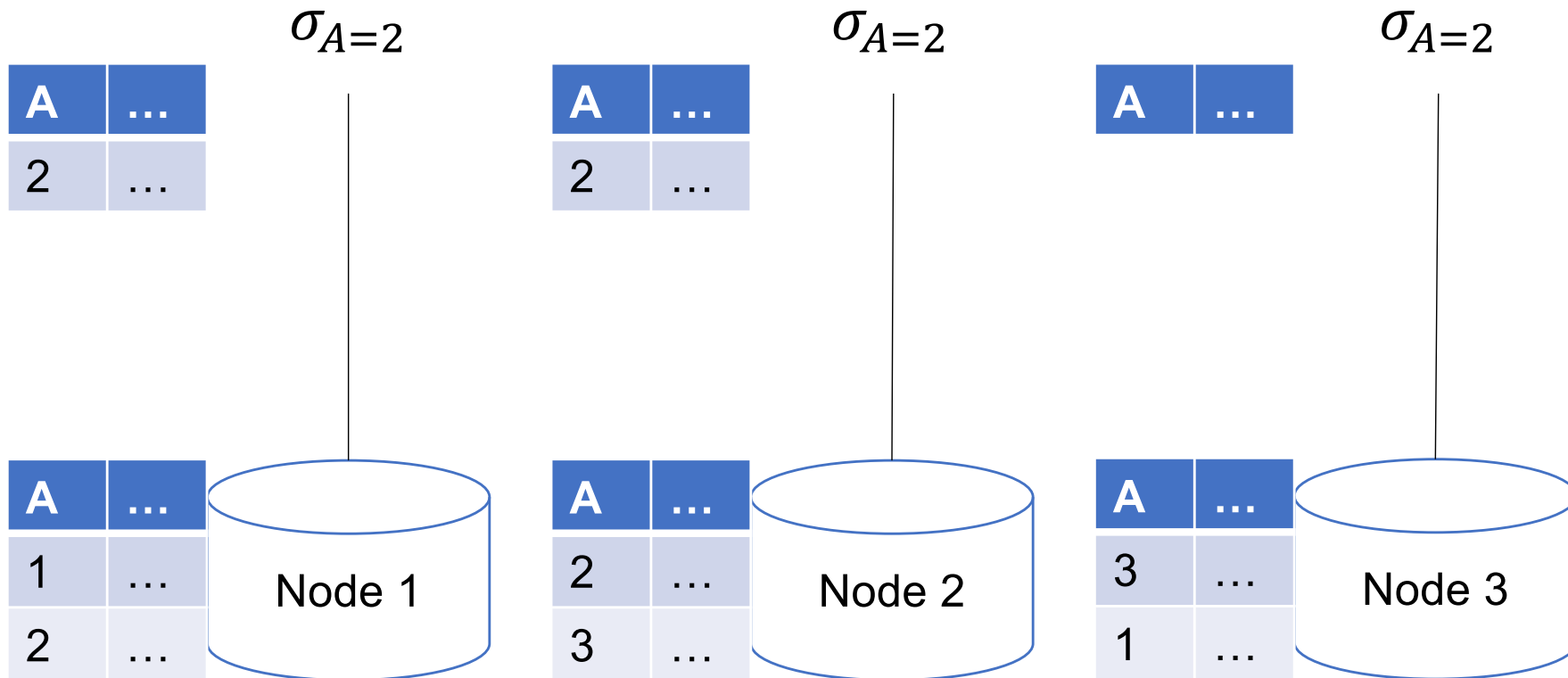
Parallel query plans implicitly union at the end



Parallel Selection

Data-parallel!

```
SELECT *  
FROM R  
WHERE A = 2
```



Parallel Selection

Compute $\sigma_{A=v}(R)$, or $\sigma_{v1 < A < v2}(R)$

- On a conventional database: cost = **B(R)**

Q: What is the cost on each node for a database with N nodes ?

A:

Parallel Selection

Compute $\sigma_{A=v}(R)$, or $\sigma_{v1 < A < v2}(R)$

- On a conventional database: cost = **B(R)**

Q: What is the cost on each node for a database with N nodes ?

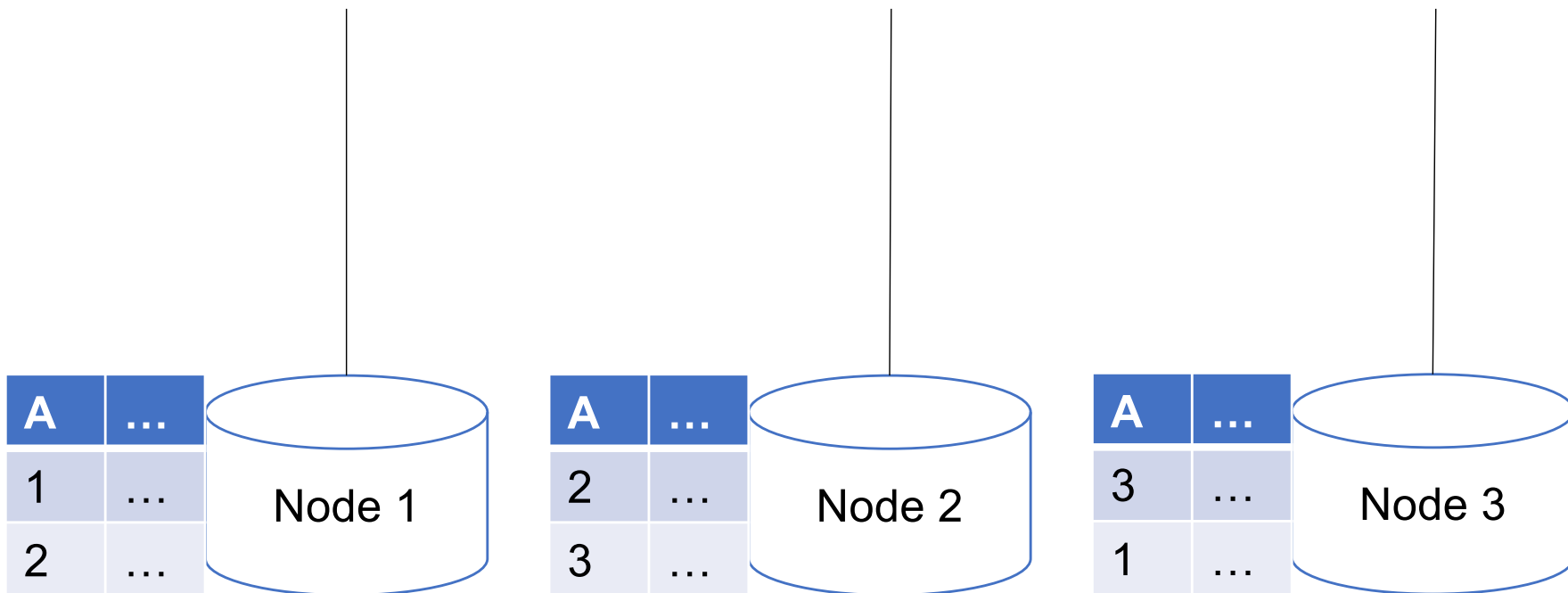
A: $B(R) / N$ block reads on each node

Parallel Selection

What if this query
is not data-parallel?

Assume:
R is block partitioned

```
SELECT *  
FROM R  
.....
```



Partitioned Aggregation

Assume:

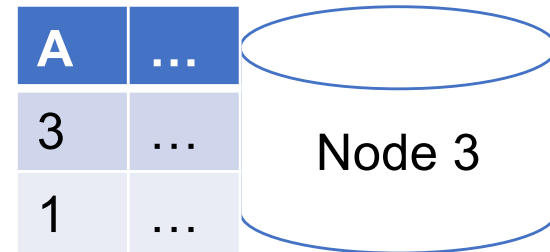
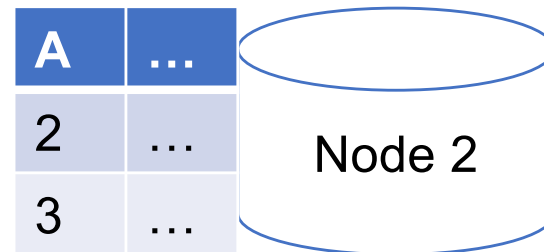
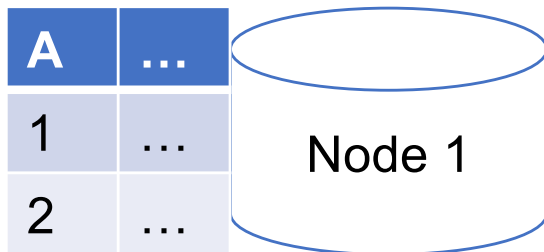
R is block partitioned

```
SELECT *  
FROM R  
GROUP BY R.A
```

$\gamma_{R.A}$

$\gamma_{R.A}$

$\gamma_{R.A}$



Partitioned Aggregation

Assume:

R is block partitioned

```
SELECT *  
FROM R  
GROUP BY R.A
```

A	...
1	...
1	...

$\gamma_{R.A}$

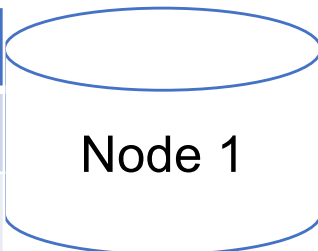
A	...
2	...
2	...

$\gamma_{R.A}$

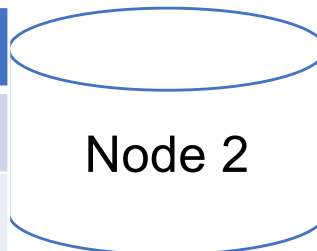
A	...
3	...
3	...

$\gamma_{R.A}$

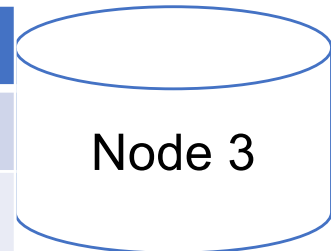
A	...
1	...
2	...



A	...
2	...
3	...



A	...
3	...
1	...



Partitioned Aggregation

1. Hash shuffle tuples

Assume:

R is block partitioned

```
SELECT *  
FROM R  
GROUP BY R.A
```

A	...
1	...
1	...

$\gamma_{R.A}$

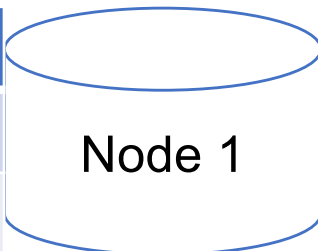
A	...
2	...
2	...

$\gamma_{R.A}$

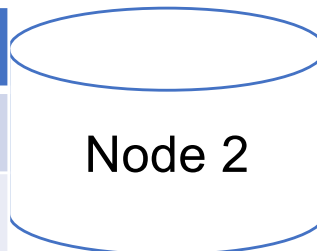
A	...
3	...
3	...

$\gamma_{R.A}$

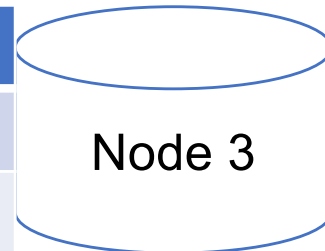
A	...
1	...
2	...



A	...
2	...
3	...



A	...
3	...
1	...

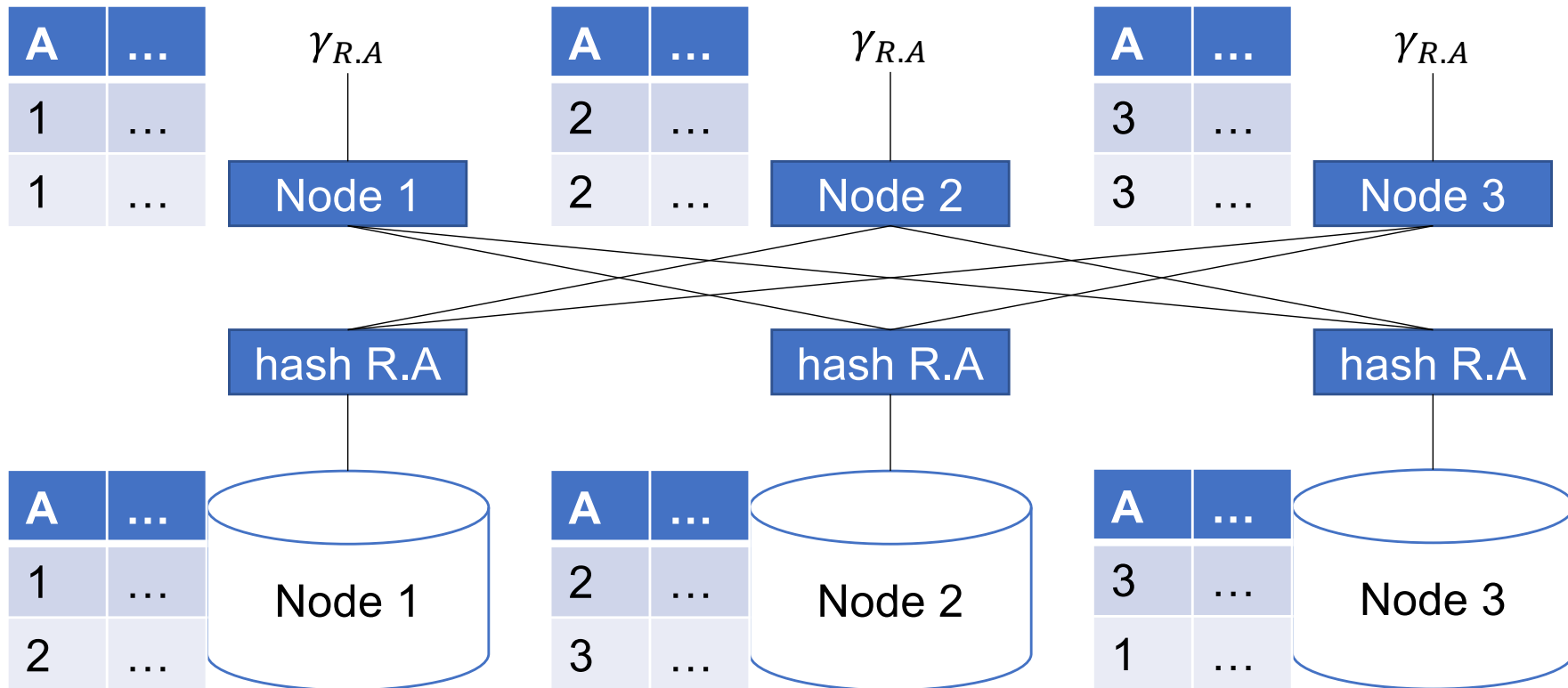


Partitioned Aggregation

1. Hash shuffle tuples

Assume:
R is block partitioned

```
SELECT *  
FROM R  
GROUP BY R.A
```



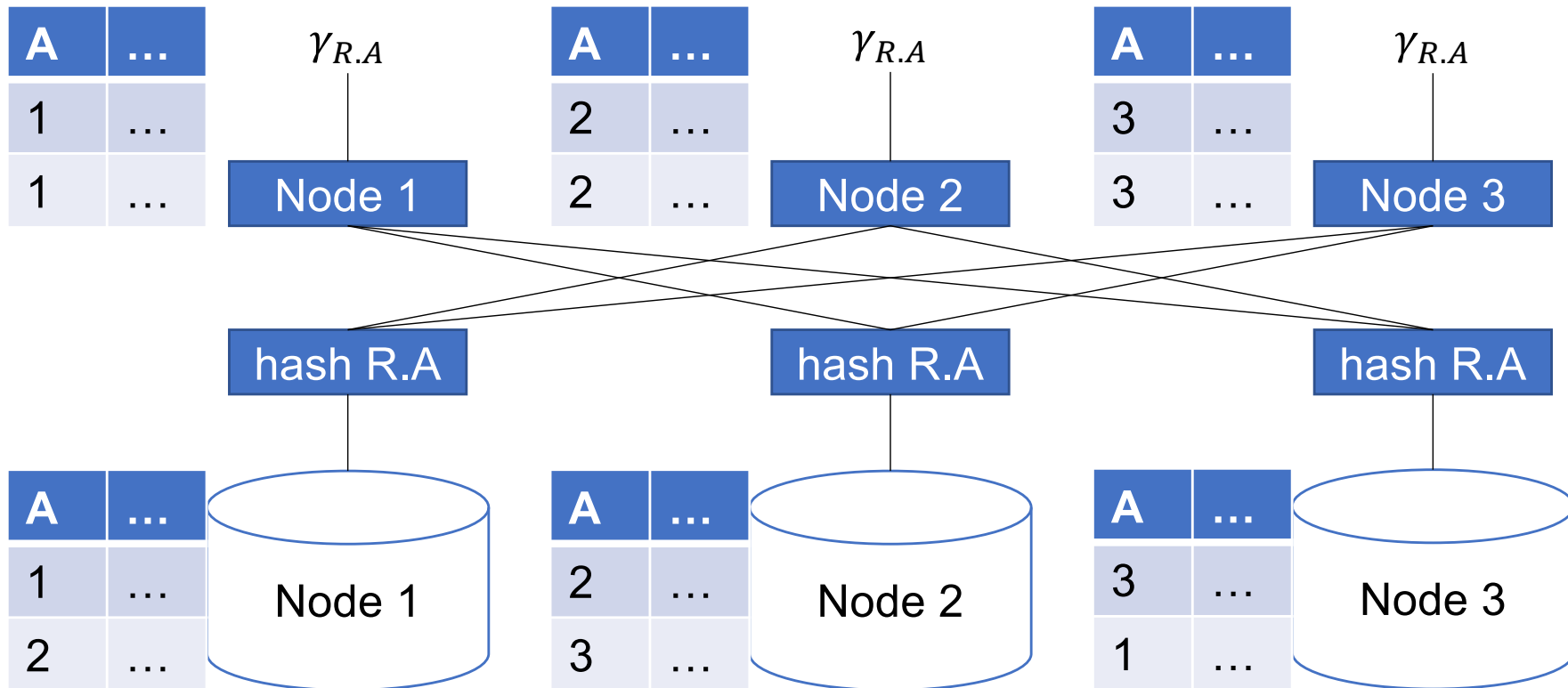
Partitioned Aggregation

1. Hash shuffle tuples
2. Local aggregation

Assume:

R is block partitioned

```
SELECT *  
FROM R  
GROUP BY R.A
```



Partition Aggregation: Summary

Select A, sum(B) from R group by A

- **Case 1:** R is partitioned on A
 - Do the group-by locally; done.

Partition Aggregation: Summary

Select A, sum(B) from R group by A

- **Case 1:** R is partitioned on A
 - Do the group-by locally; done.
- **Case 2:** R is partitioned on something else
 - Naïve: reshuffle on A, then do as in case 1

Partition Aggregation: Summary

Select A, sum(B) from R group by A

- **Case 1:** R is partitioned on A
 - Do the group-by locally; done.
- **Case 2:** R is partitioned on something else
 - Naïve: reshuffle on A, then do as in case 1
 - Better: do a *local* group-by-sum (reduces size), then reshuffle on A and do a second group-by

Partition Aggregation: Summary

Select A, sum(B) from R group by A

- **Case 1:** R is partitioned on A
 - Do the group-by locally; done.
- **Case 2:** R is partitioned on something else
 - Naïve: reshuffle on A, then do as in case 1
 - Better: do a *local* group-by-sum (reduces size), then reshuffle on A and do a second group-by

$$\begin{aligned} & \gamma_{A, \text{sum}(B)}(R_1 \cup R_2 \cup \dots \cup R_N) \\ &= \gamma_{A, \text{sum}(B)}(\gamma_{A, \text{sum}(B)}(R_1) \cup \dots \cup \gamma_{A, \text{sum}(B)}(R_N)) \end{aligned}$$

Partition Aggregation: Summary

Select A, sum(B) from R group by A

- **Case 1:** R is partitioned on A
 - Do the group-by locally; done.
- **Case 2:** R is partitioned on something else
 - Naïve: reshuffle on A, then do as in case 1
 - Better: do a *local* group-by-sum (reduces size), then reshuffle on A and do a second group-by

“Combiners”
in MapReduce

$$\begin{aligned} & \gamma_{A, \text{sum}(B)}(R_1 \cup R_2 \cup \dots \cup R_N) \\ &= \gamma_{A, \text{sum}(B)}(\gamma_{A, \text{sum}(B)}(R_1) \cup \dots \cup \gamma_{A, \text{sum}(B)}(R_N)) \end{aligned}$$

Basic Parallel GroupBy

Can we do better?

- Sum?
- Count?
- Avg?
- Max?
- Median?

Basic Parallel GroupBy

Can we do better?

- Sum?
- Count?
- Avg?
- Max?
- Median?

Distributive	Algebraic	Holistic
$\text{sum}(a_1+a_2+\dots+a_9)=$ $\text{sum}(\text{sum}(a_1+a_2+a_3)+$ $\text{sum}(a_4+a_5+a_6)+$ $\text{sum}(a_7+a_8+a_9))$	$\text{avg}(B) =$ $\text{sum}(B)/\text{count}(B)$	$\text{median}(B)$

Basic Parallel GroupBy

Can we do better?

- Sum?
- Count?
- Avg?
- Max?
- Median?

Distributive	Algebraic	Holistic
$\text{sum}(a_1+a_2+\dots+a_9)=$ $\text{sum}(\text{sum}(a_1+a_2+a_3)+$ $\text{sum}(a_4+a_5+a_6)+$ $\text{sum}(a_7+a_8+a_9))$	$\text{avg}(B) =$ $\text{sum}(B)/\text{count}(B)$	$\text{median}(B)$

YES

- Compute partial aggregates before shuffling

Basic Parallel GroupBy

Can we do better?

- Sum?
- Count?
- Avg?
- Max?
- Median?

Distributive	Algebraic	Holistic
$\text{sum}(a_1+a_2+\dots+a_9)=$ $\text{sum}(\text{sum}(a_1+a_2+a_3)+$ $\text{sum}(a_4+a_5+a_6)+$ $\text{sum}(a_7+a_8+a_9))$	$\text{avg}(B) =$ $\text{sum}(B)/\text{count}(B)$	$\text{median}(B)$

YES

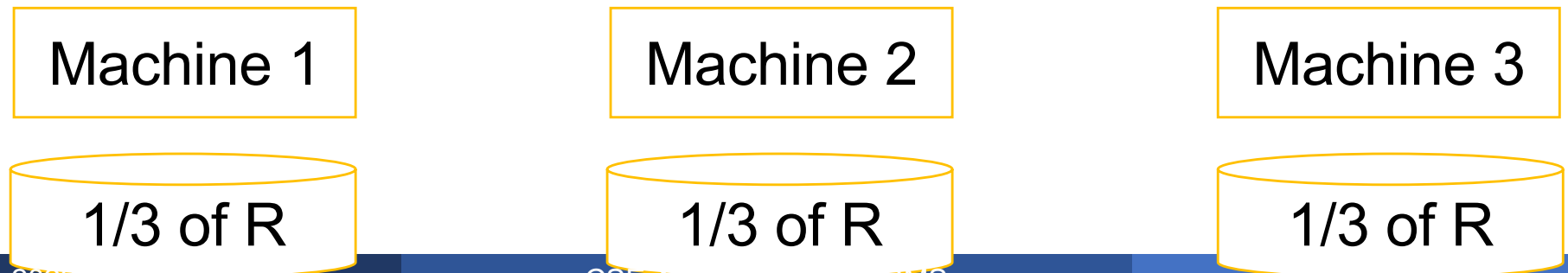
- Compute partial aggregates before shuffling

MapReduce implements this as “Combiners”

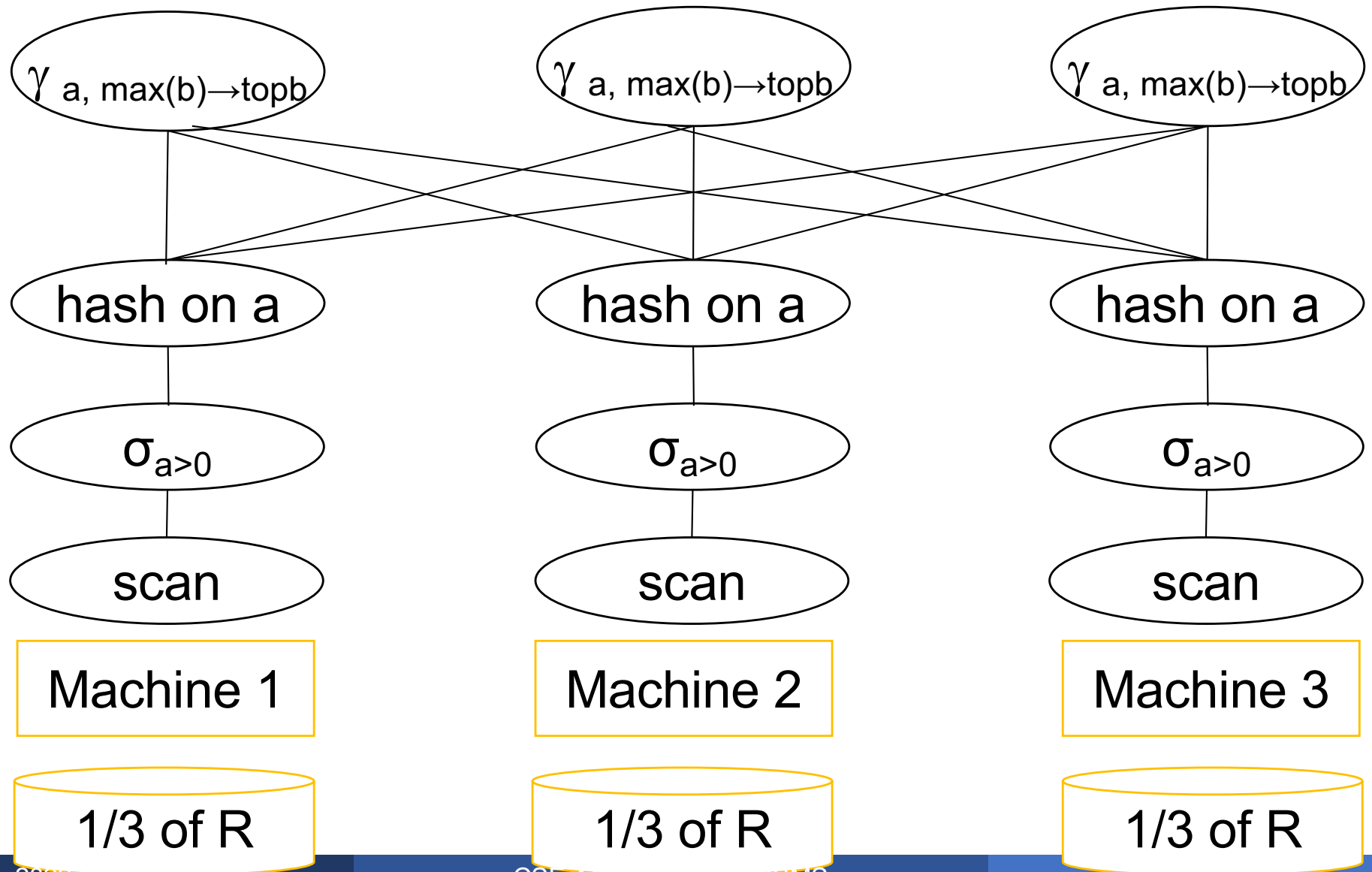
Exercise (www.draw.io is fast!)

Example Query with Group By

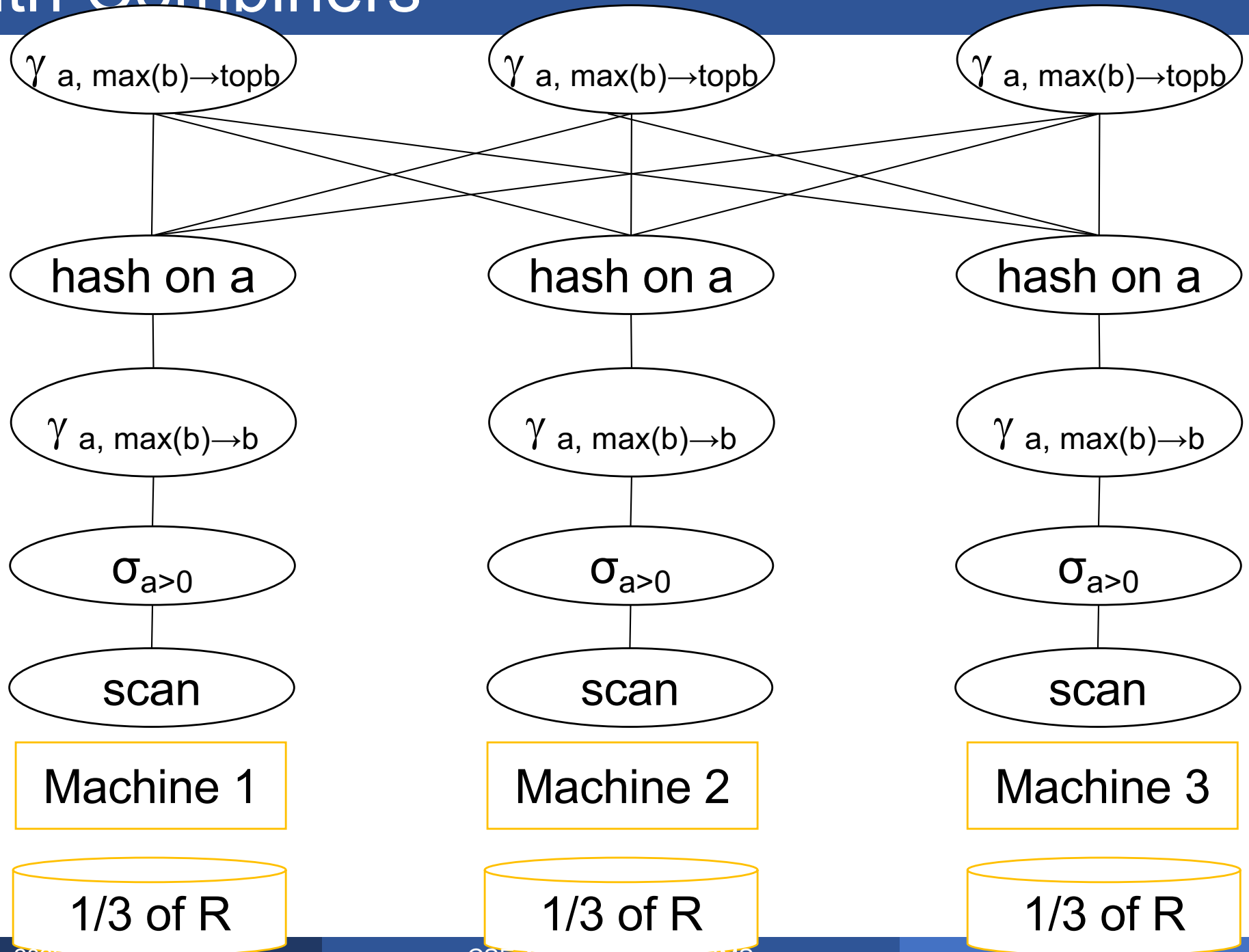
```
SELECT a, max(b) as topb  
FROM R WHERE a > 0  
GROUP BY a
```



Without Combiners



With Combiners

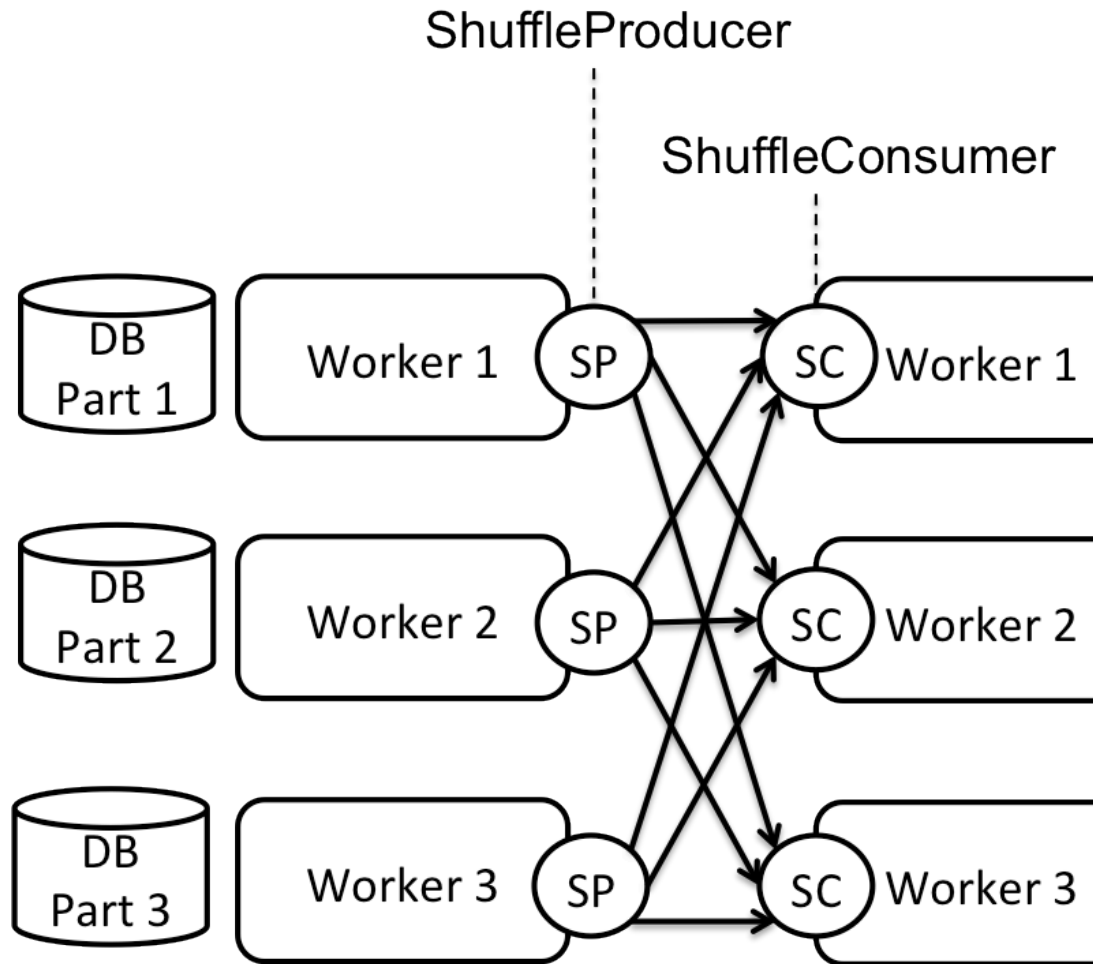


Parallel Query Evaluation

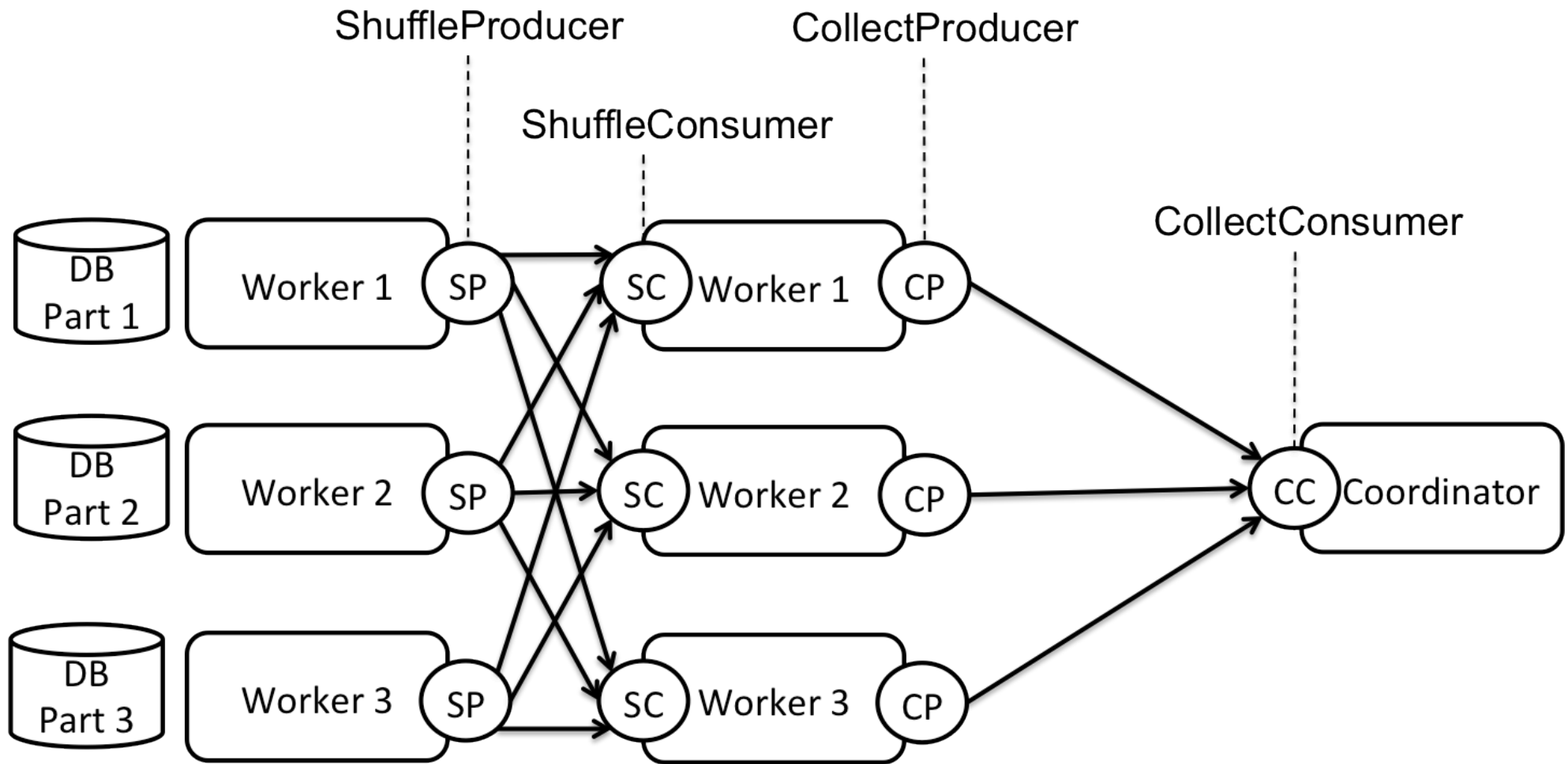
New operator: **Shuffle**

- Serves to re-shuffle data between processes
 - Handles data routing, buffering, and flow control
- Two parts: **ShuffleProducer** and **ShuffleConsumer**
- Producer:
 - Pulls data from child operator and sends to n consumers
 - Producer acts as driver for operators below it in query plan
- Consumer:
 - Buffers input data from n producers and makes it available to operator through getNext() interface

Parallel Query Execution



Parallel Query Execution



Partitioned Hash Equijoin Algorithm

1. Hash shuffle tuples on join attributes

Assume:

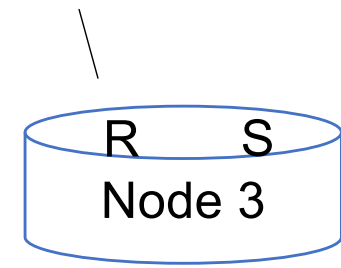
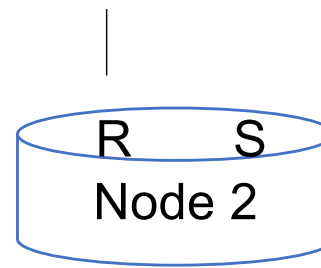
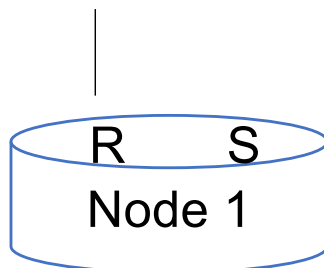
R and S are block partitioned

```
SELECT *  
FROM R, S  
WHERE R.A = S.A
```

$\bowtie_{R.A=S.A}$

$\bowtie_{R.A=S.A}$

$\bowtie_{R.A=S.A}$



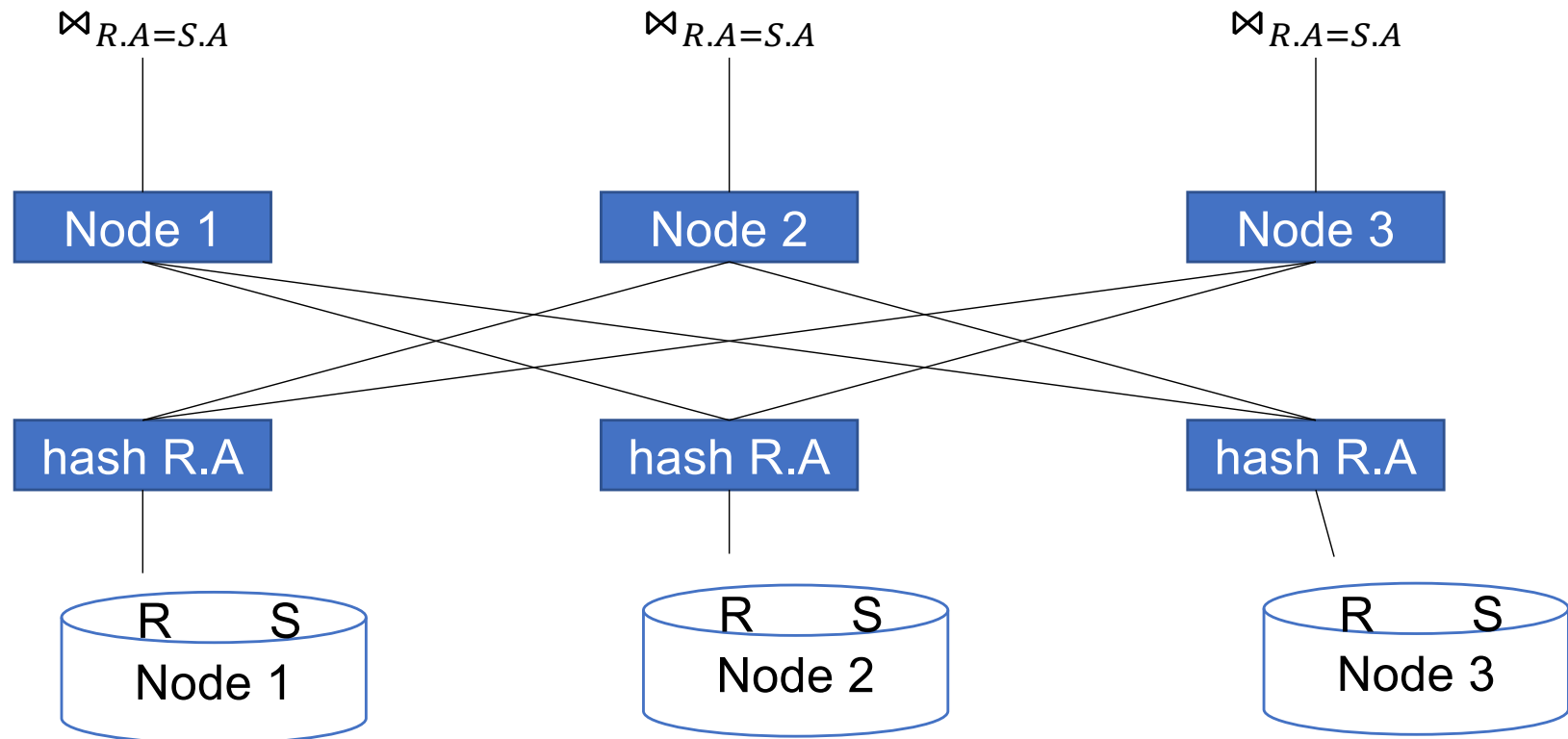
Partitioned Hash Equijoin Algorithm

1. Hash shuffle tuples on join attributes

Assume:

R and S are block partitioned

```
SELECT *  
FROM R, S  
WHERE R.A = S.A
```



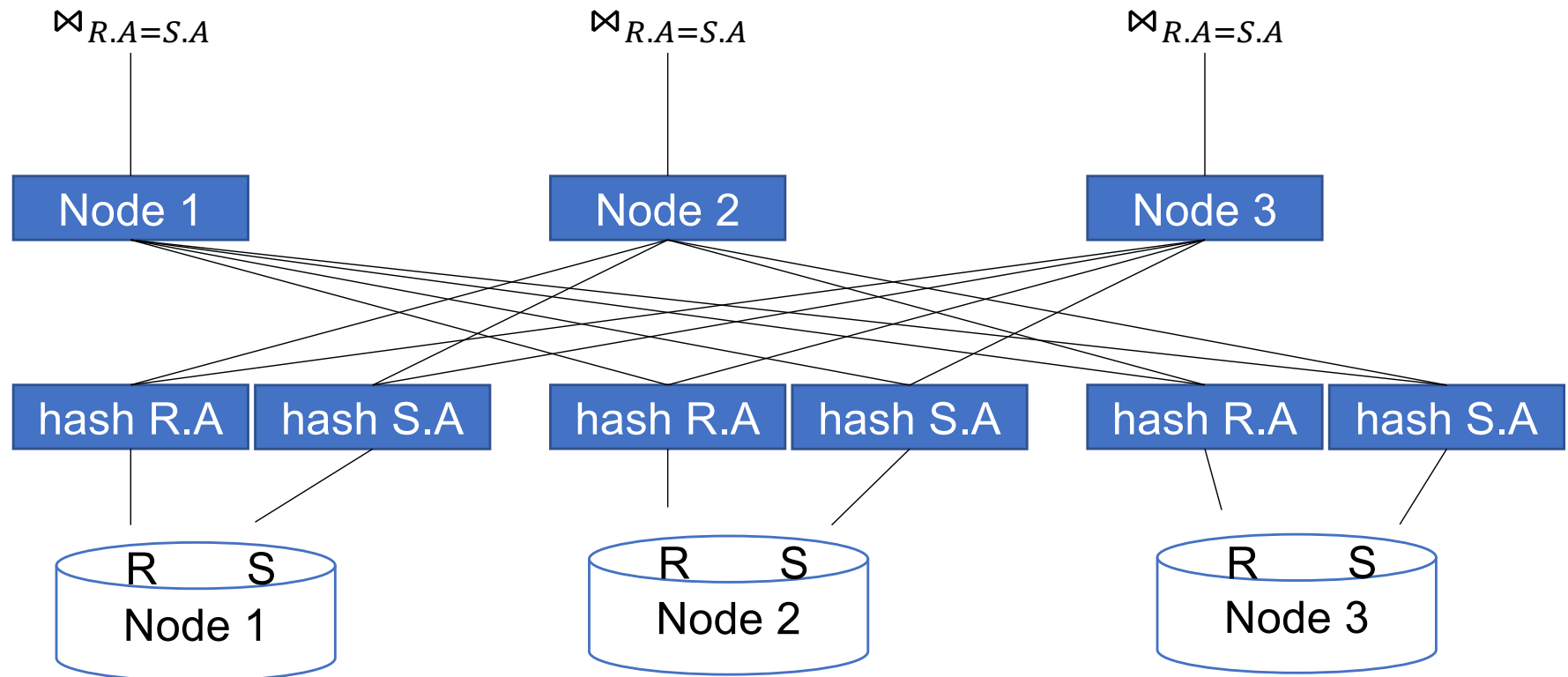
Partitioned Hash Equijoin Algorithm

1. Hash shuffle tuples on join attributes

Assume:

R and S are block partitioned

```
SELECT *  
FROM R, S  
WHERE R.A = S.A
```



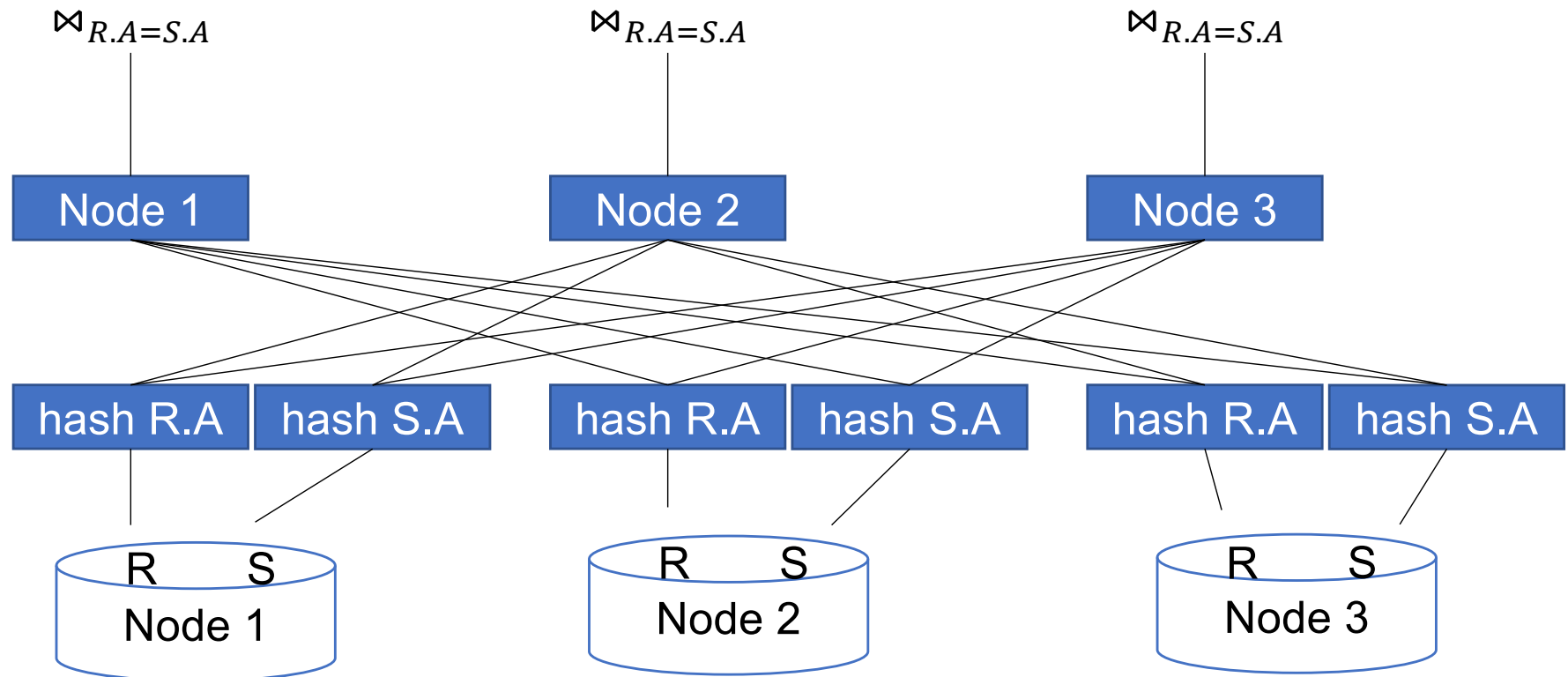
Partitioned Hash Equijoin Algorithm

1. Hash shuffle tuples on join attributes
2. Local join

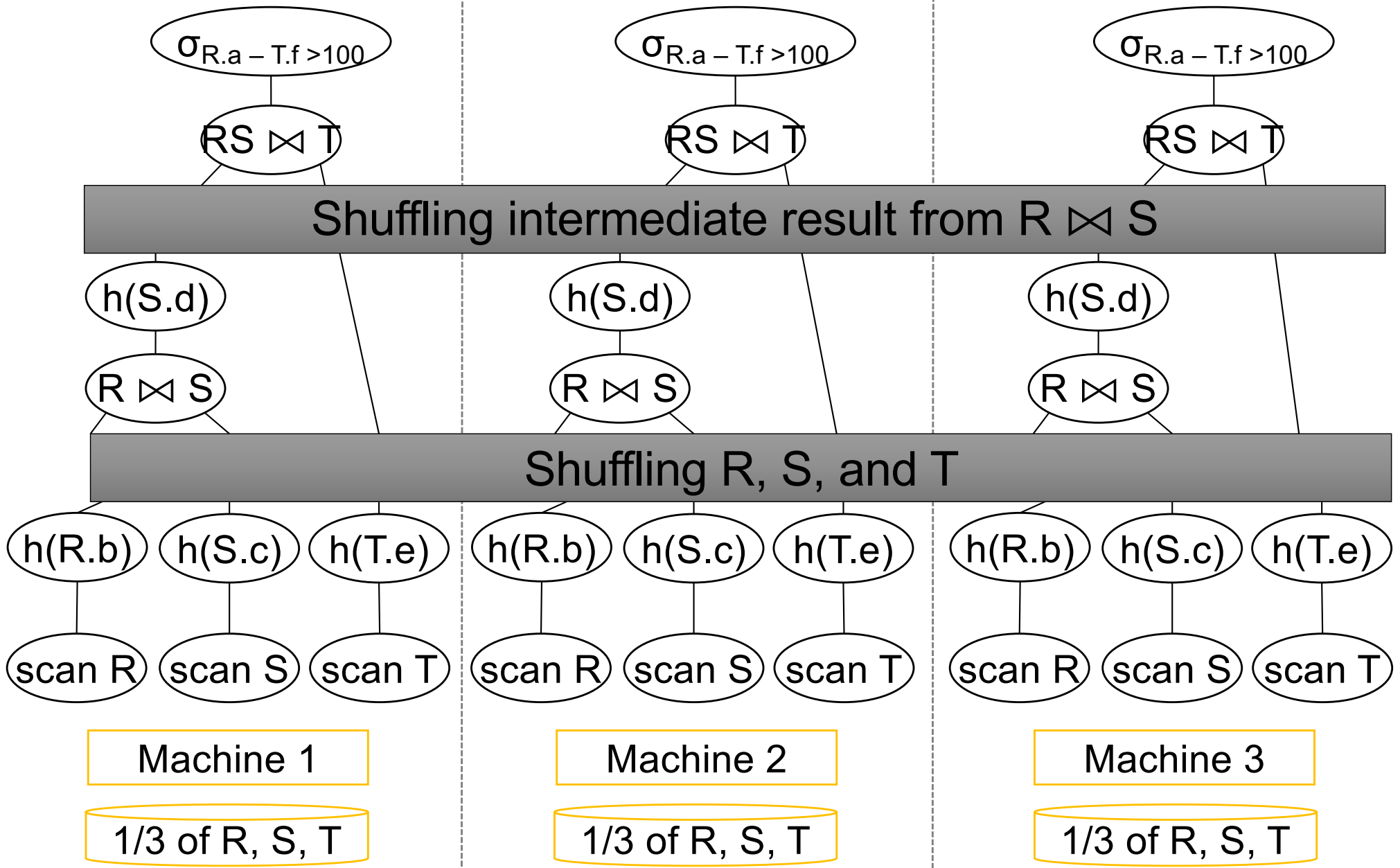
Assume:

R and S are block partitioned

```
SELECT *  
FROM R, S  
WHERE R.A = S.A
```



Multiple Shuffles



Summary

- With one new operator, we've made SimpleDB an OLAP-ready parallel DBMS!
- Next lecture:
 - Skew handling
 - Algorithm refinements

Speedup and Scaleup

- Consider:
 - Query: $\gamma_{A,\text{sum}(C)}(R)$
 - Runtime: dominated by reading chunks from disk
- If we double the number of nodes P , what is the new running time?

- If we double both P and the size of R , what is the new running time?

Speedup and Scaleup

- Consider:
 - Query: $\gamma_{A, \text{sum}(C)}(R)$
 - Runtime: dominated by reading chunks from disk
- If we double the number of nodes P , what is the new running time?
 - **Half** (each server holds $\frac{1}{2}$ as many chunks)
- If we double both P and the size of R , what is the new running time?

Speedup and Scaleup

- Consider:
 - Query: $\gamma_{A, \text{sum}(C)}(R)$
 - Runtime: dominated by reading chunks from disk
- If we double the number of nodes P , what is the new running time?
 - **Half** (each server holds $\frac{1}{2}$ as many chunks)
- If we double both P and the size of R , what is the new running time?
 - **Same** (each server holds the same # of chunks)

Parallel Join: $R \bowtie_{A=B} S$

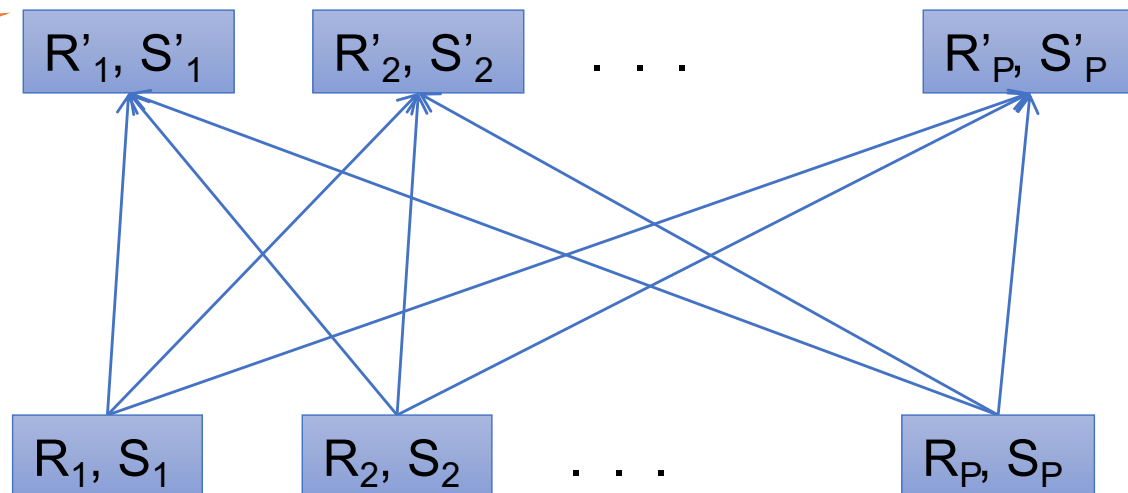
- **Data:** $R(\underline{K1}, A, C), S(\underline{K2}, B, D)$
- **Query:** $R(\underline{K1}, A, C) \bowtie S(\underline{K2}, B, D)$

Parallel Join: $R \bowtie_{A=B} S$

- **Data:** $R(\underline{K1}, A, C)$, $S(\underline{K2}, B, D)$
- **Query:** $R(\underline{K1}, A, C) \bowtie S(\underline{K2}, B, D)$

Each server computes the join locally

Reshuffle R on R.A and S on S.B



Initially, both R and S are horizontally partitioned on K1 and K2

Parallel Join: $R \bowtie_{A=B} S$

■ Step 1

- Every server holding any chunk of R partitions its chunk using a hash function $h(t.A) \bmod P$
- Every server holding any chunk of S partitions its chunk using a hash function $h(t.B) \bmod P$

■ Step 2:

- Each server computes the join of its local fragment of R with its local fragment of S

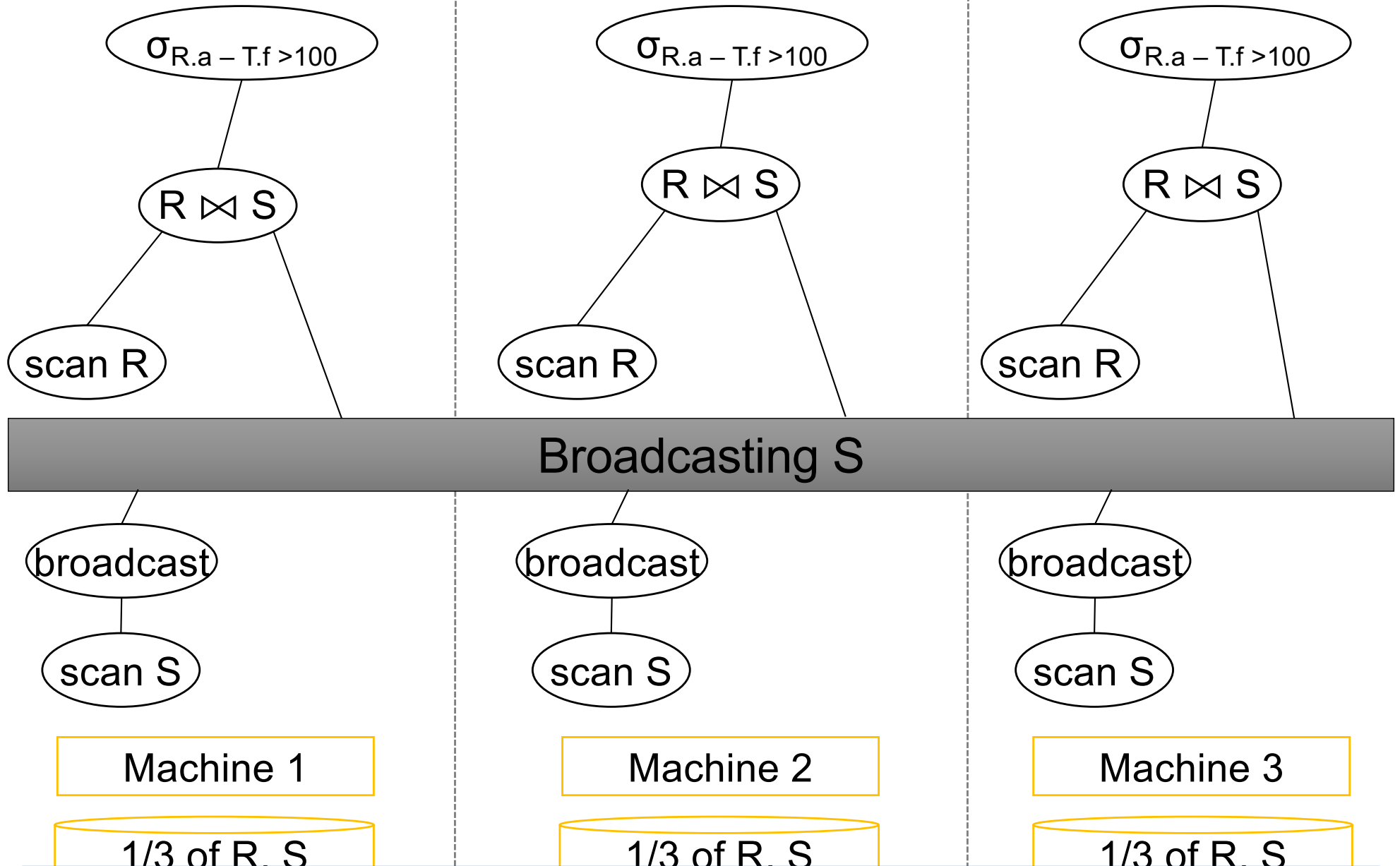
Optimization for Small Relations

When joining R and S

- If $|R| \gg |S|$
 - Leave R where it is
 - Replicate entire S relation across nodes

- Also called a **small join** or a **broadcast join**

Broadcast Join Example



Can save huge network costs!

Justin Biebers Re-visited

Skew:

- Some partitions get more **input** tuples than others

Reasons:

- Range-partition instead of hash
 - Some values are very popular: “heavy hitters”
 - Selection before join with different selectivities
-
- Some partitions generate more **output** tuples than others

Some Skew Handling Techniques

If using range partition:

- Ensure each range gets same number of tuples
- E.g.: $\{1, 1, 1, 2, 3, 4, 5, 6\} \rightarrow [1,2]$ and $[3,6]$
- Eq-depth v.s. eq-width histograms

Some Skew Handling Techniques

Create more partitions than nodes

- And be smart about scheduling the partitions
 - E.g. One node ONLY does Justin Biebers
- Note: MapReduce uses this technique

Some Skew Handling Techniques

Use subset-replicate (a.k.a. “skewedJoin”)

- Given $R \bowtie_{A=B} S$
- Given a heavy hitter value $R.A = 'v'$ (i.e. $'v'$ occurs very many times in R)
- Partition R tuples with value $'v'$ across all nodes e.g. block-partition, or hash on other attributes
- Replicate S tuples with value $'v'$ to all nodes
- R = the build relation
- S = the probe relation