ARIZONA STATE UNIVERSITY

CSE 512: Distributed and Parallel Data Systems

Lecture 3

Instructor: Mohamed Sarwat

Distributed Query Processing (cont.)

- If the data is not fragmented
  - process whatever operations you can locally (select, project)
- Query can involve joins between data at multiple sites
- Must transfer data over network
- Transfer of data highest cost
  - Algorithms to minimize amount of data transferred (NP-hard)

Joins on multiple sites

- Assume R1 |X| R2 → Site 3
- with R1 at S1 and R2 at S2
- Can do:

a) R1 → S3 and R2 → S3, do join at S3

b) R1 → S2, execute join at S2, result to S3

c) R2 → S1, execute join at S1, result to S3

Semi-Join

- Si X|$_{a=b}$ Sj
  - – π join attribute from Si, select tuples at Sj whose join attributes match
  - – Send join attribute from Si to Sj, Select tuples at Sj who match keys sent, send those tuples back to Si
- |X|
  - – Last step is to perform join at Si

Query: $\pi_{\text{dname, lname}}$ Department $|X|_{\text{mgr=ssn}}$ Employee

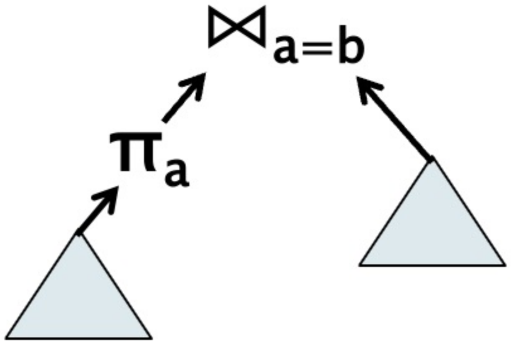
Steps 1) and 2) are equivalent to the semi-join
Department $|X_{\text{mgrssn=ssn}}$ Employee

1) At S2: E' ← (π_{ssn} Employee) //project join attribute
Send to S1

2) At S1: E'' ← ($\pi_{\text{mgrssn, dname}}$ (Department $|X|_{\text{mgrssn=ssn}}$ E'))
join with department and send only needed attributes
Send to S2

3) At S2: $\pi_{\text{dname, lname}}$ (E'' $|X|_{\text{mgrssn=ssn}}$ Employee)
join with employee and send final attributes needed
Send to S3

Two-way semijoin



- Take R, project join key attributes
- Ship to S
- Fetch all matching S tuples
- Ship to destination
- Join R tuples with S tuples

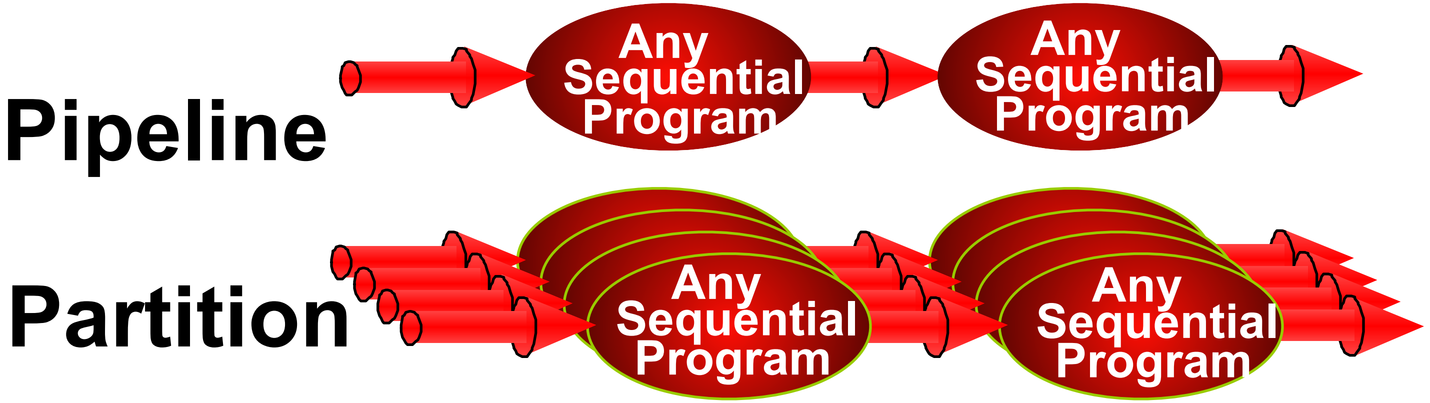
Bloomjoin

- Take R, build *Bloom filter*
  - Build a large bit vector
  - Take each value of R.a, hash with multiple functions $h_i(a)$
  - Set bits for each $h_i(a)$
- Ship to S
- Fetch all matching S tuples
  - For each S.b, ensure a match to each $h_i(b)$ in Bloom filter
- Ship to destination
- Join R tuples with S tuples

Parallel Database Systems

Parallel DBMS: Intro

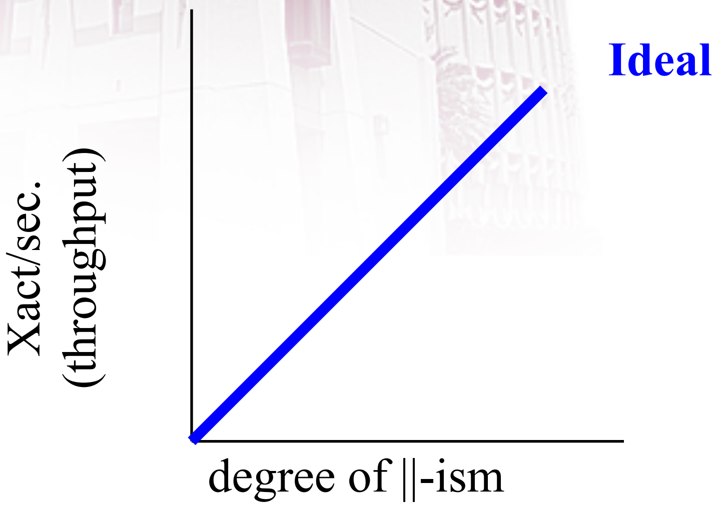
- Parallelism is natural to DBMS processing
  - *Pipeline parallelism:* many machines each doing one step in a multi-step process.
  - *Partition parallelism:* many machines doing the same thing to different pieces of data.
  - **Both are natural in DBMS!**



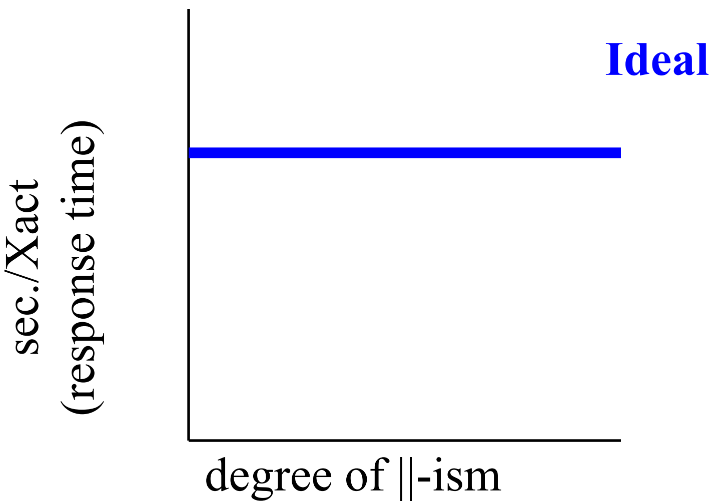
outputs split N ways, inputs merge M ways

Some || Terminology

- 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.

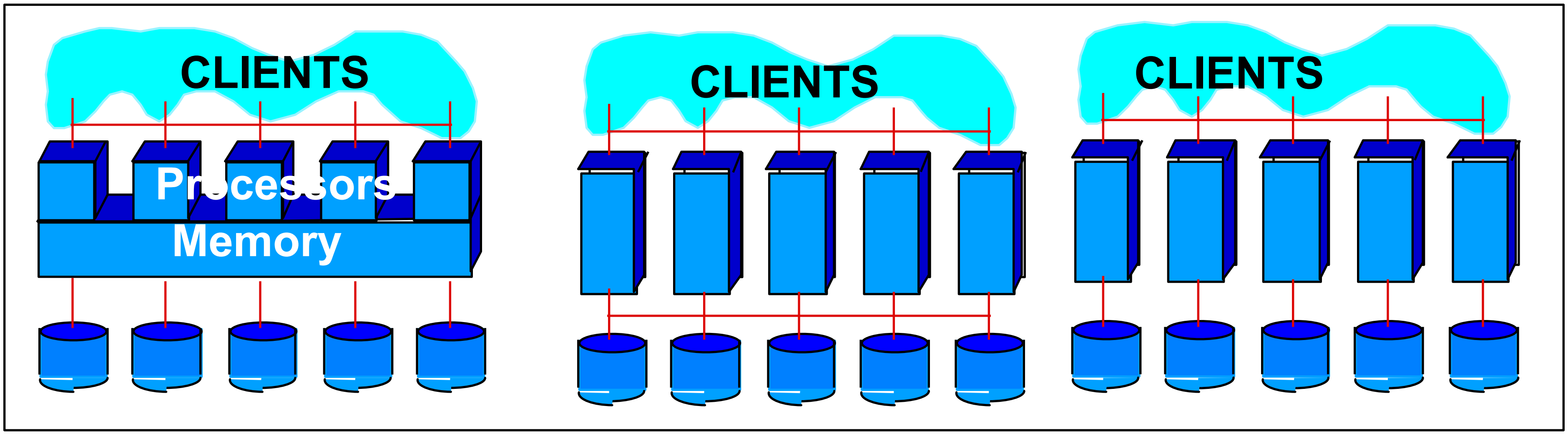


Architecture Issue: Shared What?

Shared Memory (SMP)

Shared Disk

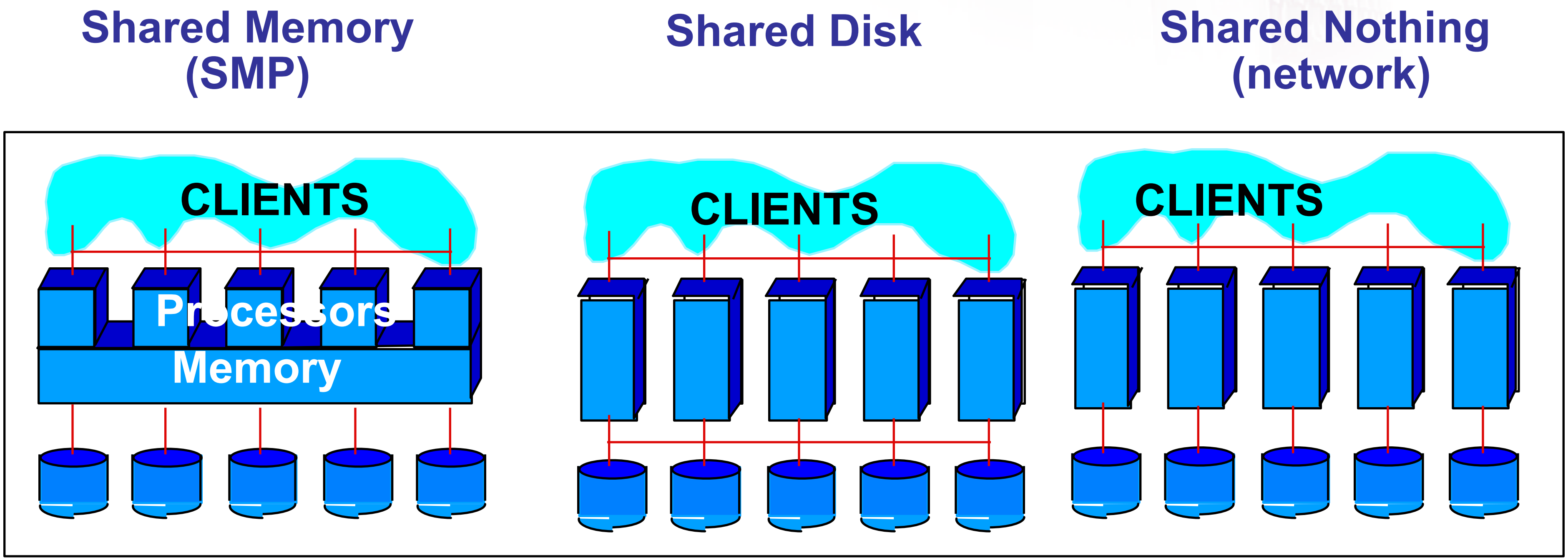
Shared Nothing (network)



Shared Memory Vs. Shared Nothing?

ARIZONA STATE UNIVERSITY

Architecture Issue: Shared What?



Easy to program
Expensive to build
Difficult to scaleup

Hard to program
Cheap to build
Easy to scaleup

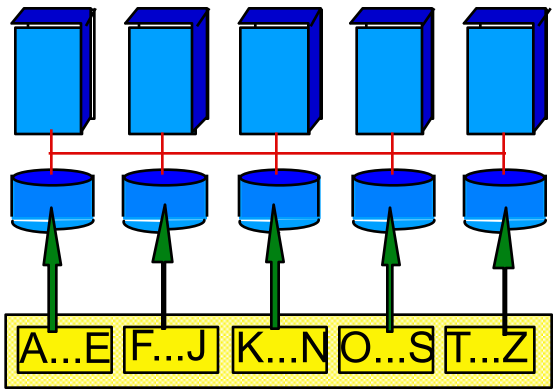
Different Types of DBMS II-ism

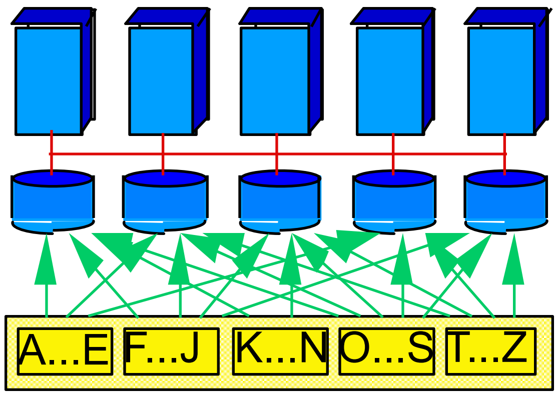
- Intra-operator parallelism
  - get all machines working to compute a given operation (scan, sort, join)
- Inter-operator parallelism
  - each operator may run concurrently on a different site (exploits pipelining)
- Inter-query parallelism
  - different queries run on different sites

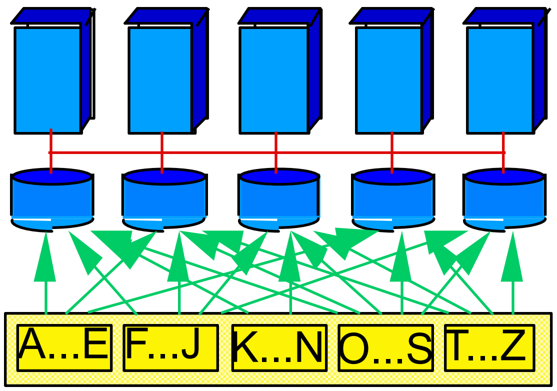
Automatic Data Partitioning

Partitioning a table:

Range **Hash** **Round Robin**



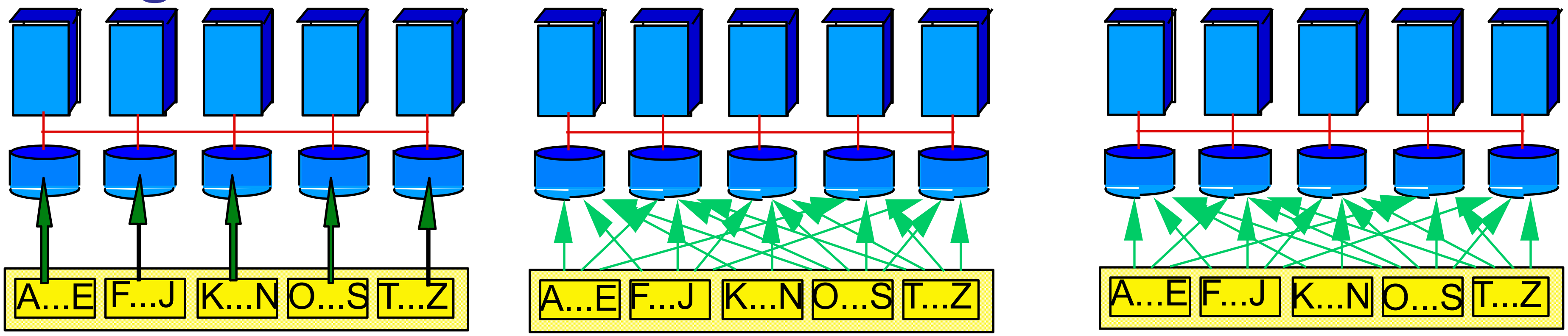




Automatic Data Partitioning

Partitioning a table:

Range | **Hash** | **Round Robin**



Good for equijoins, range queries group-by

Good for equijoins

Good to spread load

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

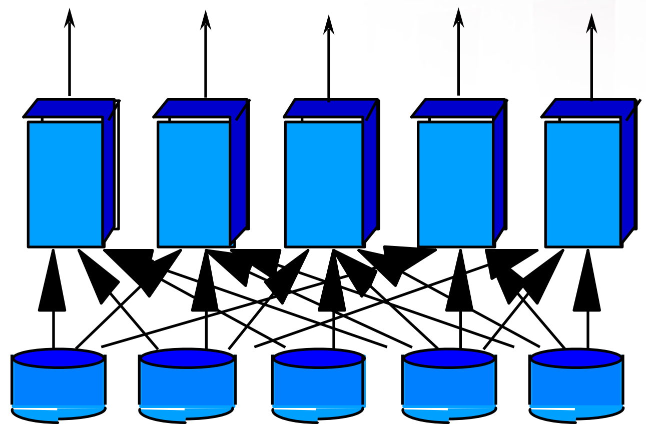
Parallel Scans

- Scan in parallel, and merge.
- Selection may not require all sites for range or hash partitioning.
- Indexes can be built at each partition.

Parallel Sorting

e.g., Sort all Employees based on their age?

Parallel Sorting



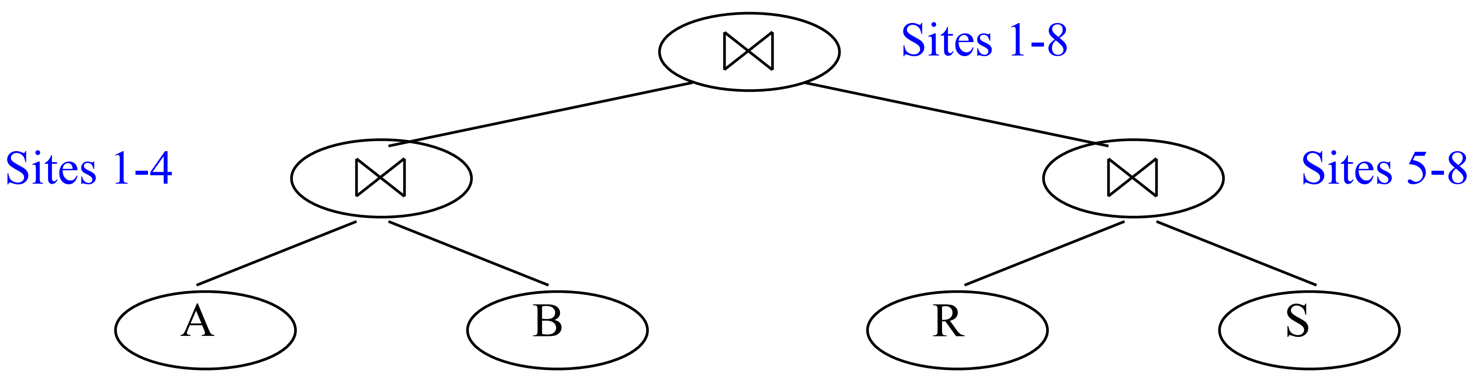
- Idea:
  - Scan in parallel, and range-partition as you go.
  - As tuples come in, begin "local" sorting on each
  - Resulting data is sorted, and range-partitioned.
  - Problem: ***skew!***
  - Solution: "sample" the data at start to determine partition points.

Parallel Joins

- 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 (or plain Merge-Join):
  - Sorting gives range-partitioning.
  - Merging partitioned tables is local.

Complex Parallel Query Plans

- Complex Queries: Inter-Operator parallelism
  - Pipelining between operators:
    - note that sort block the pipeline!!
  - Bushy Trees

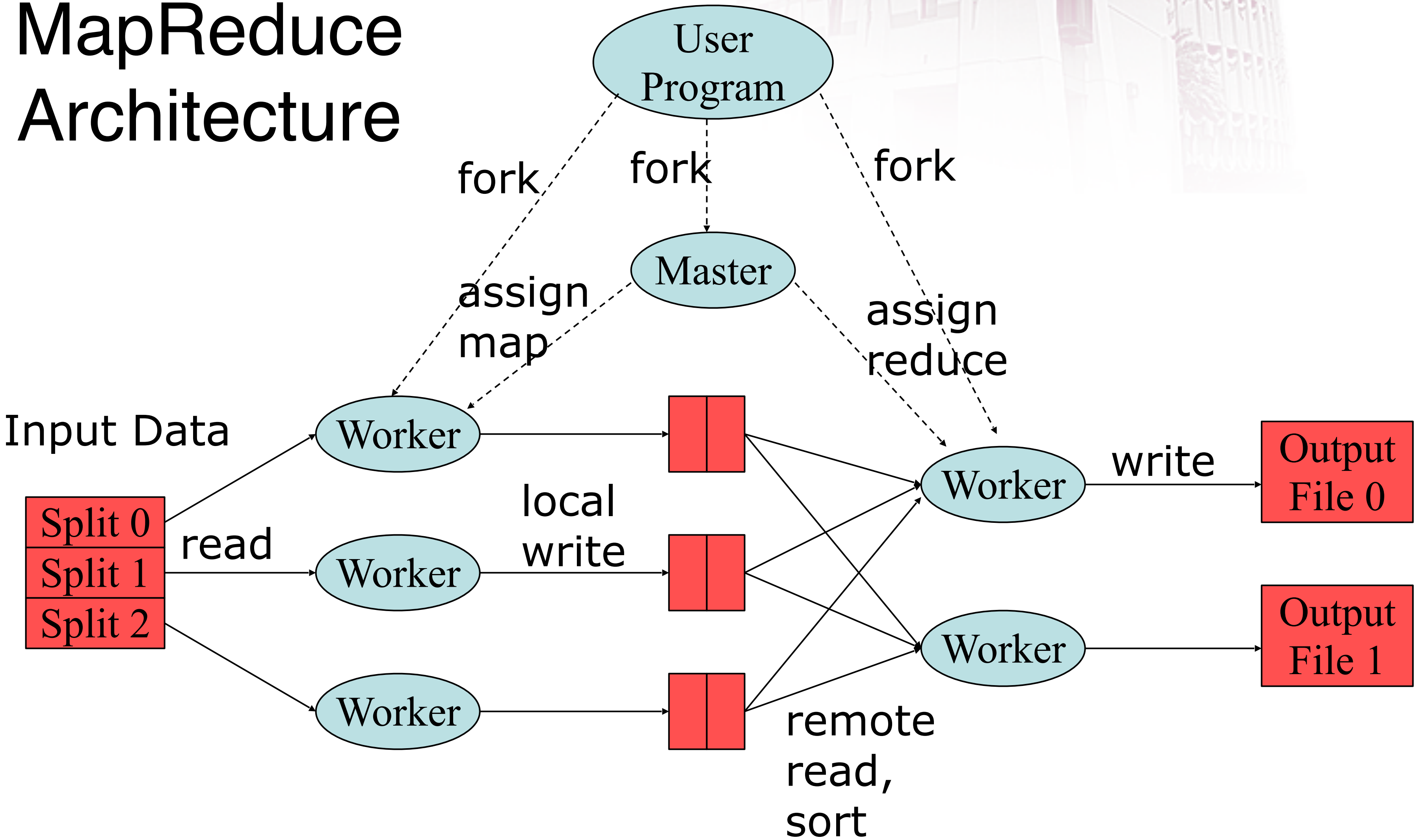


ARIZONA STATE UNIVERSITY

Questions

Data Management in MapReduce Systems

MapReduce Architecture



How It Works

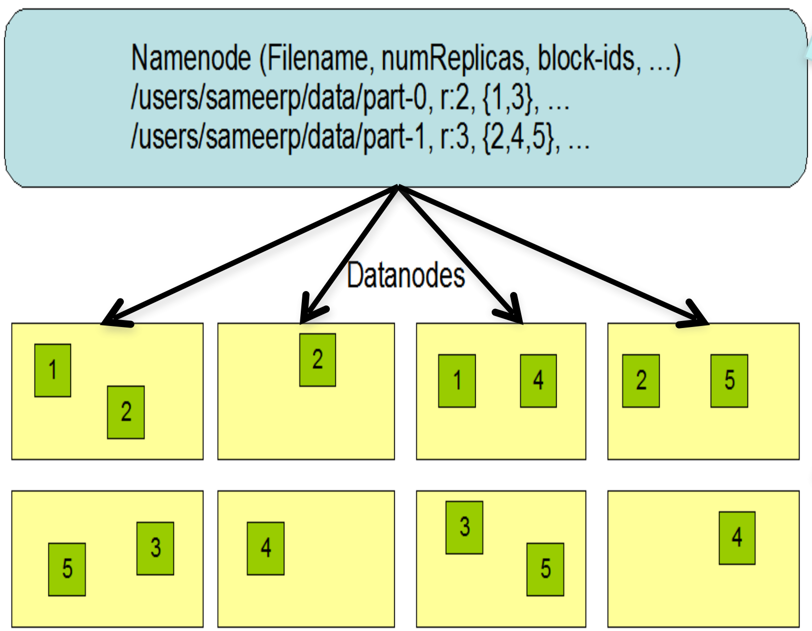
- **Distributed file system (HDFS)**
  - Single namespace for entire cluster
  - Replicates data 3x for fault-tolerance
- **MapReduce framework**
  - Executes user jobs specified as “map” and “reduce” functions
  - Manages work distribution & fault-tolerance

HDFS (1/2)

- Files split into 128MB blocks
- Blocks replicated across several DataNodes (usually 3)
- NameNode stores metadata (file names, location)
- Optimized for large files, sequential reads
- Files are append-only

HDFS (2/2)

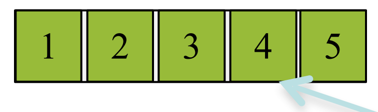
Block Replication



Centralized namenode

-Maintains metadata info about files

File *F*



Blocks (64 MB)

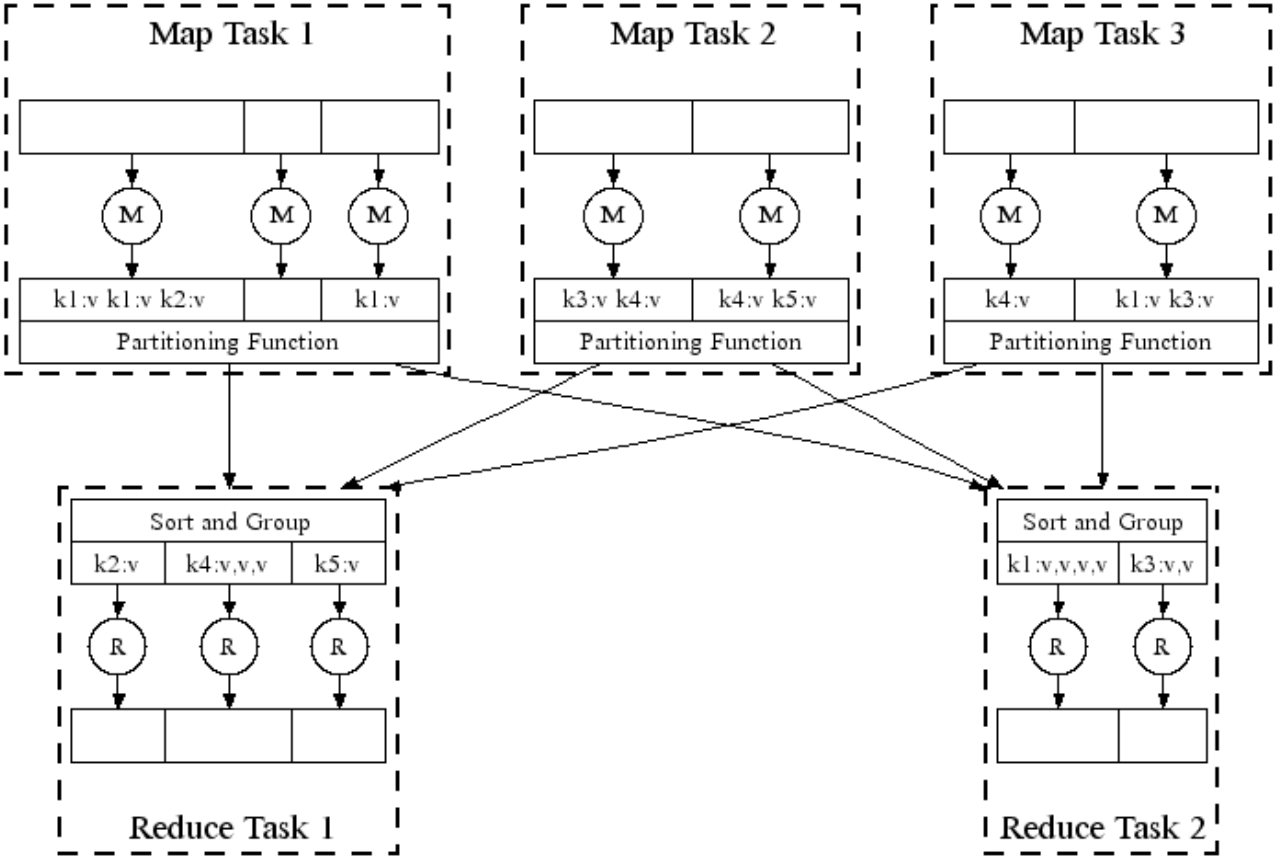
Many datanode (1000s)

- Store the actual data
- Files are divided into blocks
- Each block is replicated *N* times (Default = 3)

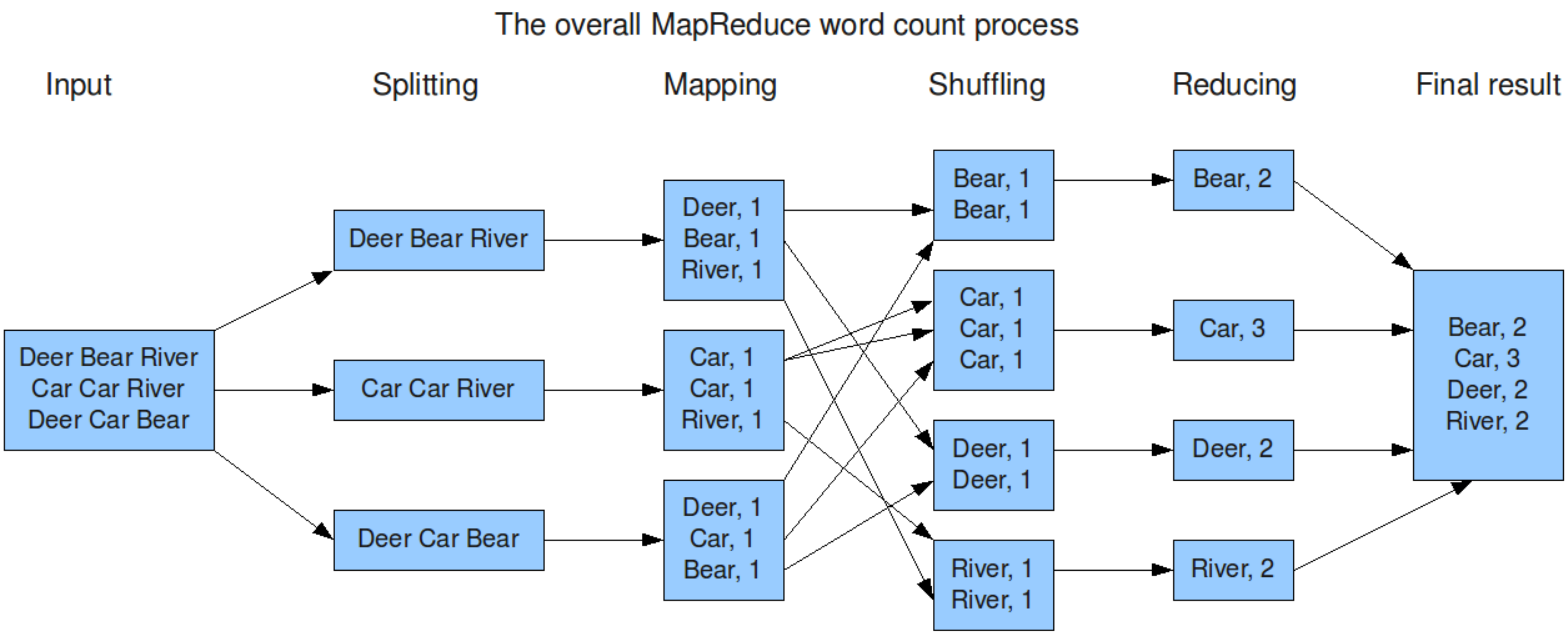
MapReduce Programming Model

- Data type: key-value records
- Map function:
  - (Key1, Value1) ➔ list(Key2, Value2)
- Reduce function:
  - (Key2, list(Value2)) ➔ list(Key3, Value3)

Parallel Execution



Example: Word Count



MapReduce Execution Details

- Mappers preferentially placed on same node or same rack as their input block
  - Push computation to data, minimize network use
- Mappers save outputs to local disk before serving to reducers
  - Allows having more reducers than nodes
  - Allows recovery if a reducer crashes

Fault Tolerance in Hadoop

- If a task crashes:
  - Retry on another node
  - OK for a map because it had no dependencies
  - OK for reduce because map outputs are on disk
- If the same task repeatedly fails, fail the job
- If a node crashes:
  - Re-launch its current tasks on other nodes
  - Re-launch any maps the node previously ran
    - Necessary because their output files were lost along with the crashed node

Fault Tolerance in Hadoop

- If a task is going slowly (straggler):
  - Launch second copy of task on another node
  - Take the output of whichever copy finishes first, and kill the other one
- Critical for performance in large clusters ("everything that can go wrong will")

ARIZONA STATE UNIVERSITY

Selection Operator in MapReduce

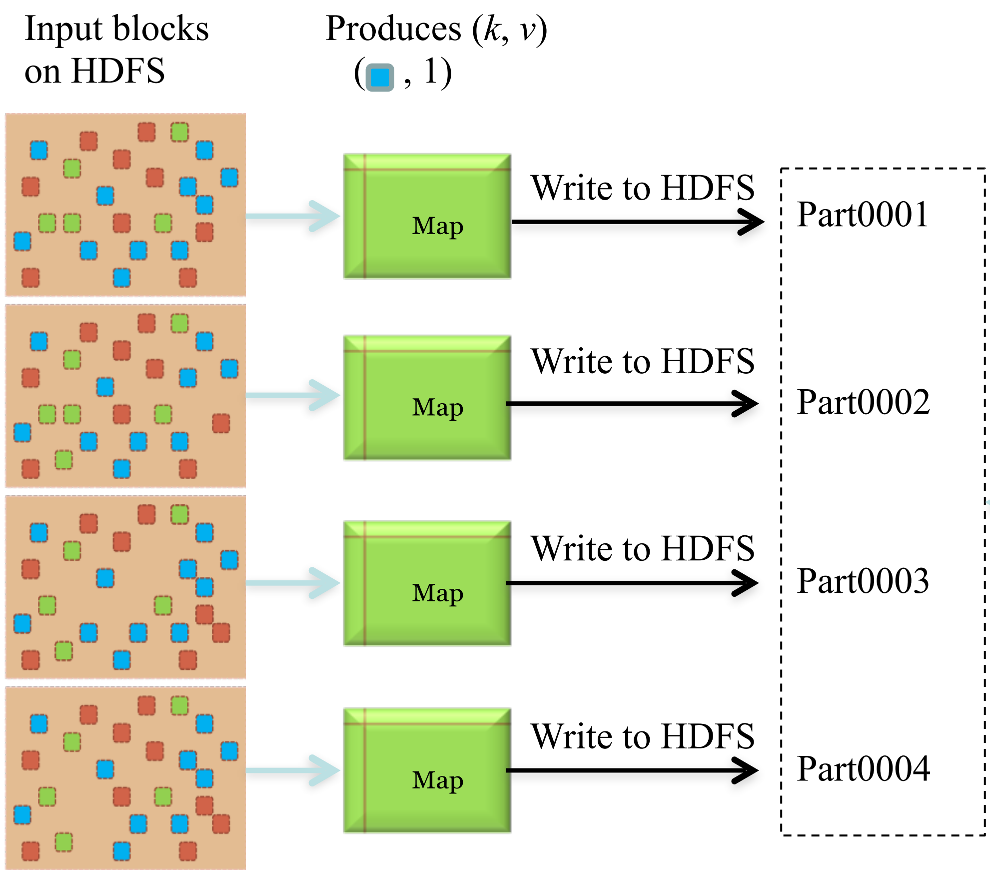
$$\sigma_{rating>8}(S2)$$

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Example 3: Color Filter

Job: Select only the blue and the green colors



- Each map task will select only the blue or green colors
- No need for reduce phase

That’s the output file, it has 4 parts on probably 4 different machines

Projection Operator in MapReduce

$\pi_{sname,rating}(S2)$

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

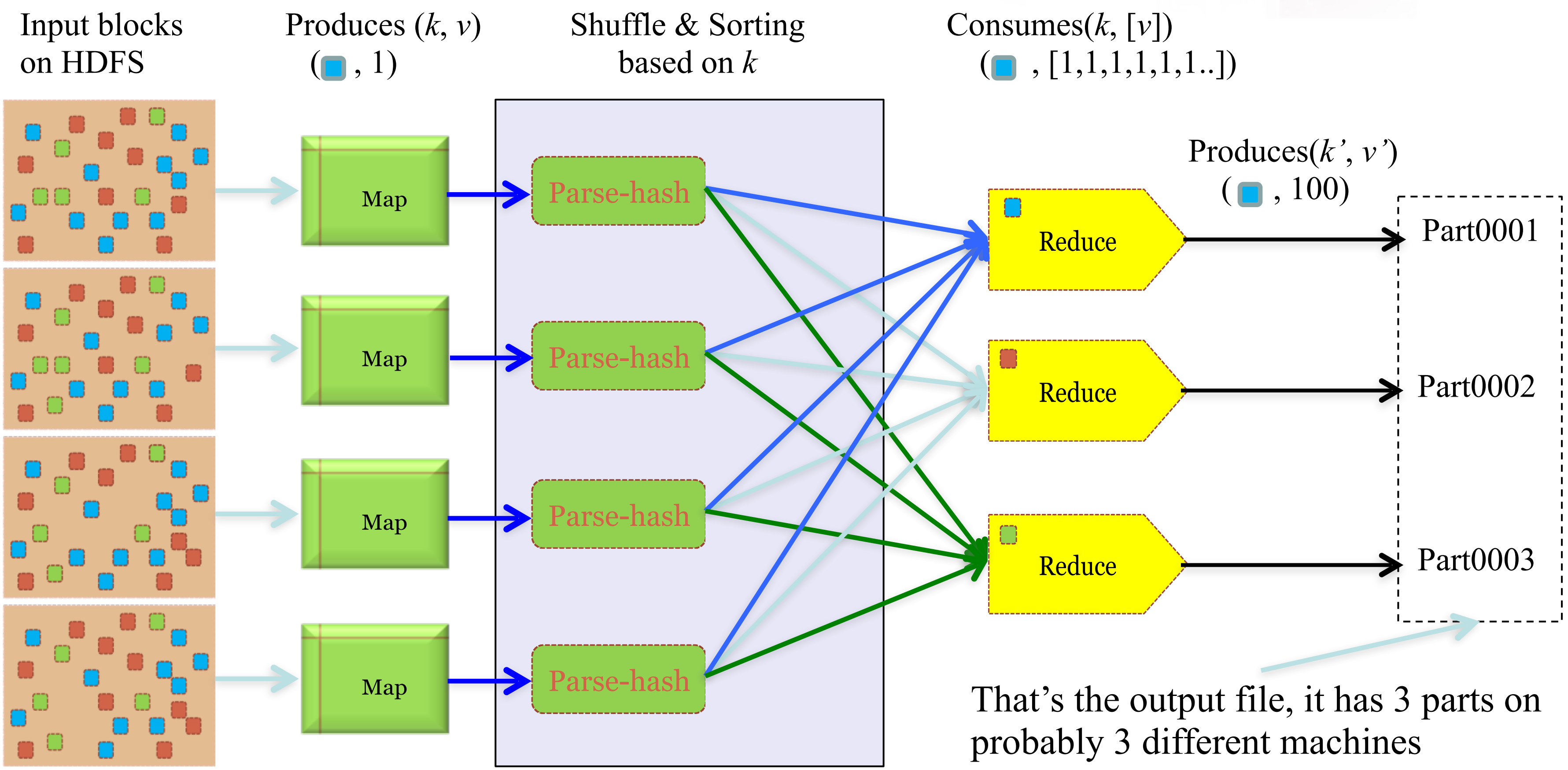
S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Group By in MapReduce

Example 2: Color Count

Job: Count the number of each color in a data set



That's the output file, it has 3 parts on probably 3 different machines

Join Operator in MapReduce

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Questions