Map Reduce II

Big Data Management

Knowledge objectives

1. Enumerate the different kind of processes in Hadoop MapReduce
2. Draw the hierarchy of Hadoop MapReduce objects
3. Explain the information kept in the Hadoop MapReduce coordinator node
4. Explain how to decide the number of mappers and reducers
5. Explain the fault tolerance mechanisms in Hadoop MapReduce in case of
    a) Worker failure
    b) Master failure
6. Identify query shipping and data shipping in MapReduce
7. Explain the effect of using the combine function in MapReduce
8. Identify the synchronization barriers of MapReduce
9. Explain the main problems and limitations of Hadoop MapReduce

Single-stage MapReduce jobs – Example (1)

```
order_id:1001
customer: Ann

line items:
```

puerh	8	$3.25	$26
genmaicha	4	$3	$12
dragonwell	8	$2.25	$18

```
shipping address: …

payment details:

card: Amex
cc number: 12345
expiry: 04/28
```

- We have an object that stores `orders`
- Each `order` has `line items`
- Each line item has a `product id`, `quantity`, and the `price` charged

- Sales analysis people want to see a product and its total revenue for the last seven days

Single-stage MapReduce jobs – Example (1)

order_id:1001

customer: Ann

line items:

puerh	8	$3.25	$26
genmaicha	4	$3	$12
dragonwell	8	$2.25	$18

shipping address: …

payment details:

card: Amex
cc number: 12345
expiry: 04/28

map

puerh:

price	$26
quantity	8

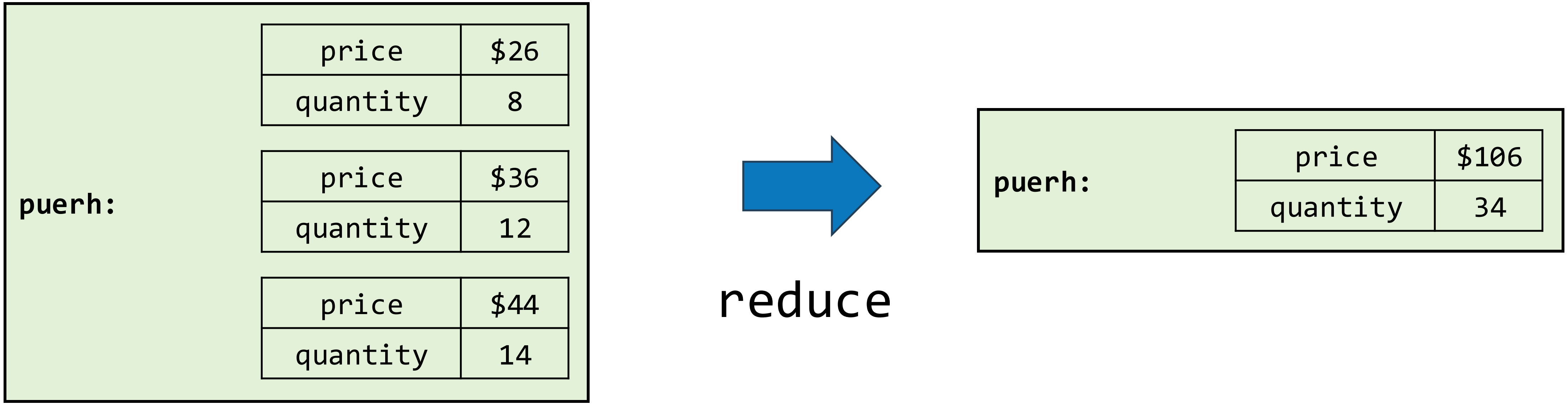
genmaicha:

price	$12
quantity	4

dragonwell:

price	$18
quantity	8

Single-stage MapReduce jobs – Example (1)



Multi-stage MapReduce jobs – Example (1)

order_id:1001

customer: Ann

line items:

puerh	8	$3.25	$26
genmaicha	4	$3	$12
dragonwell	8	$2.25	$18

shipping address: …

payment details:

card: Amex
cc number: 12345
expiry: 04/28

- We have an object that stores `orders`
- Each `order` has `line items`
- Each line item has a `product id`, `quantity`, and the `price` charged

- Sales analysis people want to see and compare the sales of products for each month in 2011 to the prior year
  - Product X in `Dec 2011` sold Y times, representing a Z% increase compared to `Dec 2010`

Multi-stage MapReduce jobs – Example (2)

order_id:1001

customer: Ann

line items:

puerh	8	$3.25	$26
genmaicha	4	$3	$12
dragonwell	8	$2.25	$18

shipping address: …

payment details:

card: Amex
cc number: 12345
expiry: 04/28

map

2011:12:puerh:

product	puerh
year	2011
month	12
quantity	26

2011:12:puerh:

product	puerh
year	2011
month	12
quantity	44

2011:12:n:

product	…
year	…
month	…
quantity	…

Multi-stage MapReduce jobs – Example (3)

2011:12: puerh:

product	puerh
year	2011
month	12
quantity	26

2011:12: puerh:

product	puerh
year	2011
month	12
quantity	44

…

2011:12: n:

product	…
year	…
month	…
quantity	…

reduce

2011:12: puerh:

product	puerh
year	2011
month	12
quantity	1200

2010:12: puerh:

product	puerh
year	2011
month	12
quantity	1000

2009:12: puerh:

product	puerh
year	2009
month	12
quantity	950

Multi-stage MapReduce jobs – Example (3)

2011:12:puerh:

product	puerh
year	2011
month	12
quantity	1200

2010:12:puerh:

product	puerh
year	2011
month	12
quantity	1000

2009:12:puerh:

product	puerh
year	2009
month	12
quantity	950

→ map2

12:puerh:

product	puerh
year	2011
month	12
quantity	1200
prior_yr	0

12:puerh:

product	puerh
year	2011
month	12
quantity	0
prior_yr	1000

Multi-stage MapReduce jobs – Example (3)

12: puerh:

product	puerh
year	2011
month	12
quantity	1200
prior_yr	0

12: puerh:

product	puerh
year	2011
month	12
quantity	0
prior_yr	1000

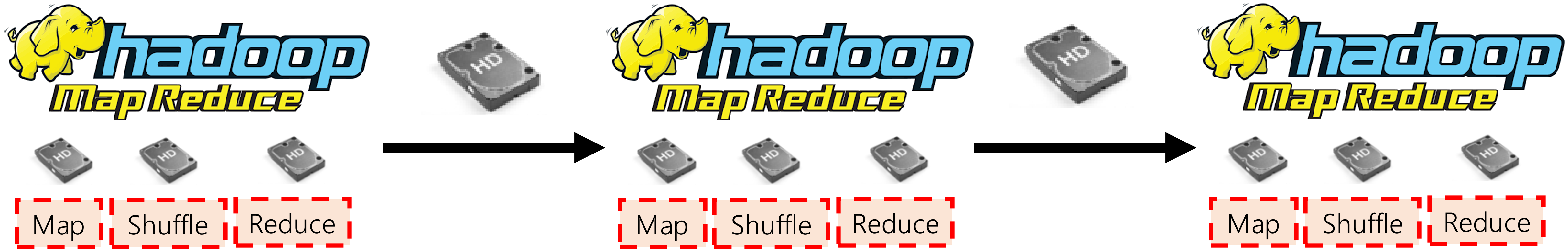
→ reduce2

12: puerh:

product	puerh
year	2011
month	12
quantity	1200
prior_yr	1000
increase	20%

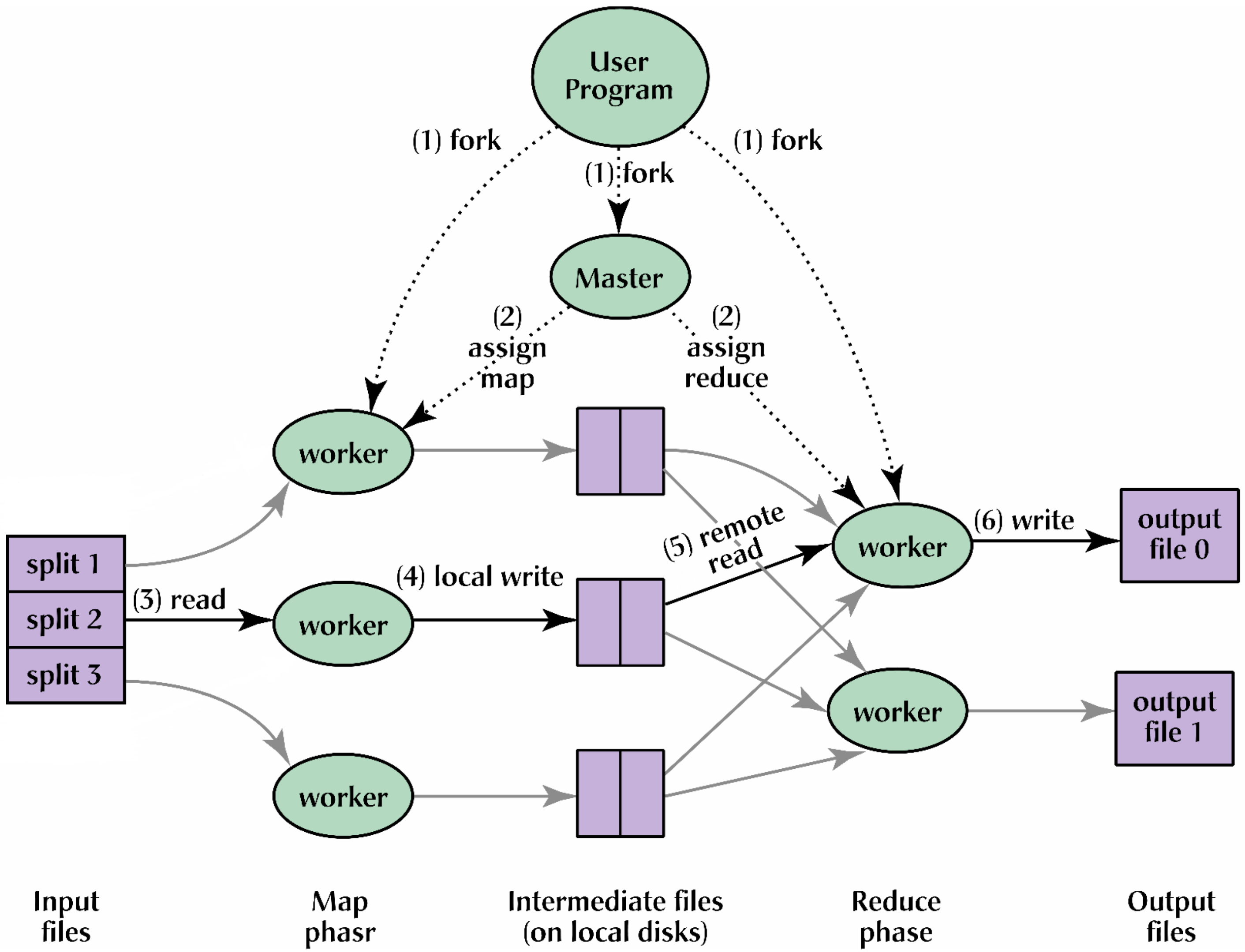
Multi-stage MapReduce

- Coordination between phases using DFS
  - Map, Shuffle, Reduce
- Coordination between jobs using DFS
  - Count, rank, aggregate, …



Architecture

Processes



J. Dean et al.

Architectural decisions

- Users submit jobs to a coordinator scheduling system
  - There is one coordinator and many workers
  - Jobs are decomposed into a set of tasks
  - Tasks are assigned to available workers within the cluster/cloud by the coordinator
    - `O(M+R)` scheduling decisions
    - Try to benefit from locality
    - As computation comes close to completion, coordinator assigns the same task to multiple workers
- The coordinator keeps all relevant information
  a) Map and Reduce tasks
    - Worker state (i.e., idle, in-progress, or completed)
    - Identity of the worker machine
  b) Intermediate file regions
    - Location and size of each intermediate file region produced by each map task
      - Stores `O(M*R)` states in memory

Design decisions

- Number of Mappers (M)
  - One per split in the input (default one chunk)
    - To exploit data parallelism: `10*N < M < 100*N`
  - Mappers should take at least a minute to execute
    - Split size depends on the time to process data
- Number of Reducers (R)
  - Many can produce an explosion of intermediate files
    - For immediate launch: `0.95*N*MaxTasks`
    - For load balancing: `1.75*N*MaxTasks`

`N` is the number of nodes (a.k.a. machines) in the cluster.

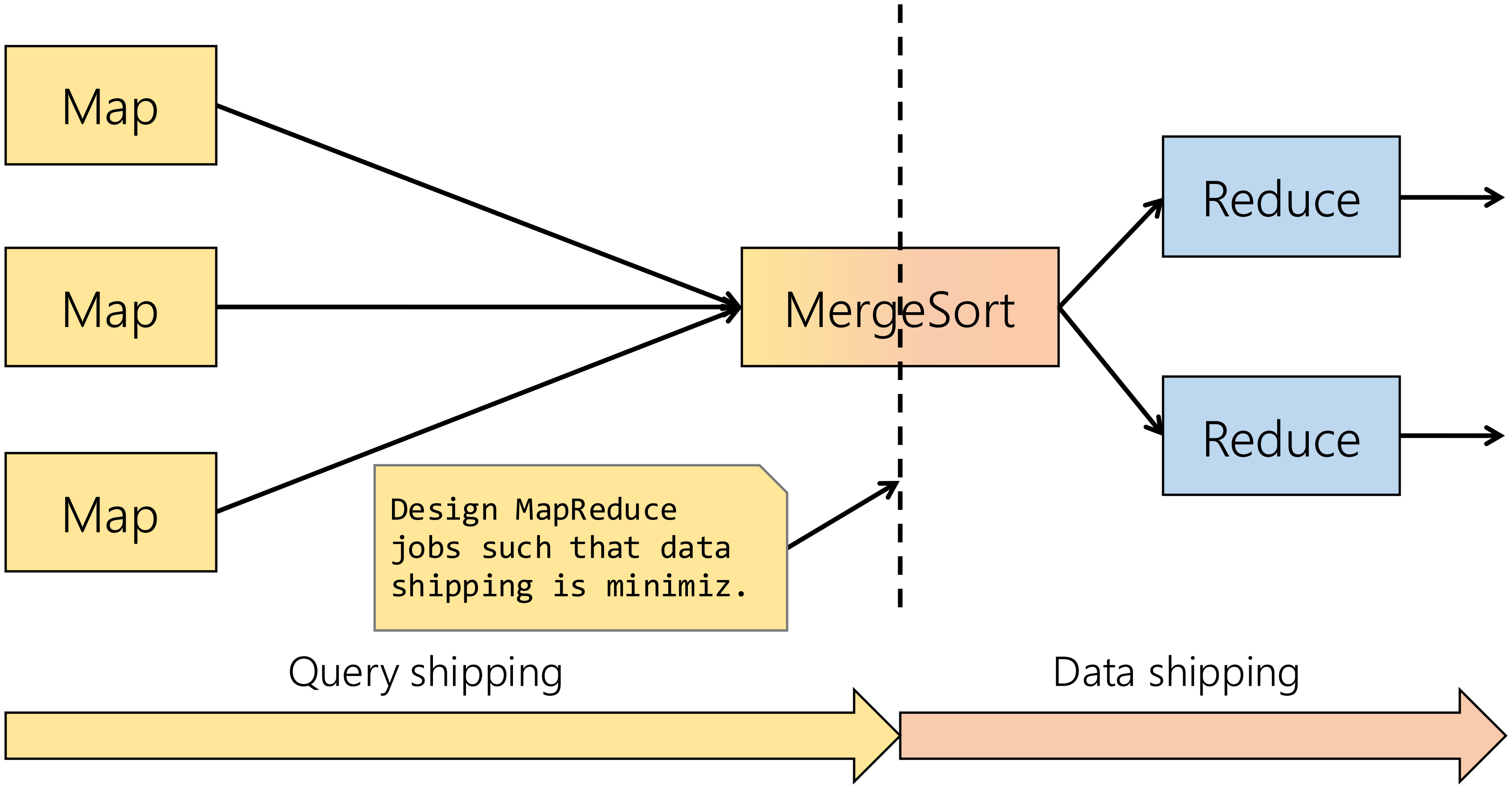
http://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html#Payload

Fault-tolerance mechanisms

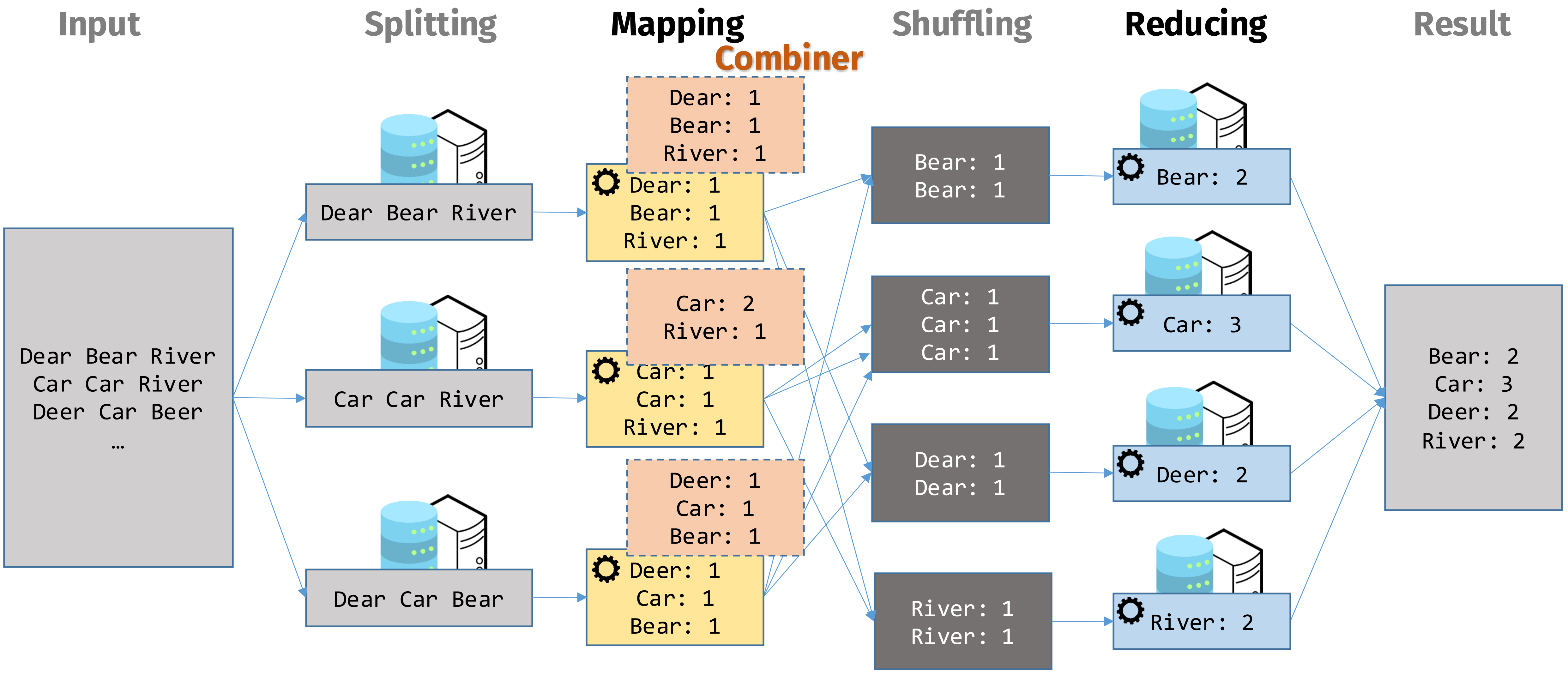
- Worker failure
  - Workers ping the coordinator periodically (heartbeat)
    - Coordinator assumes failure if not happening
  - Completed/in-progress map and in-progress reduce tasks on failed worker are rescheduled on a different worker node
    - Use chunk replicas
- Coordinator failure
  - Since there is only one, it is less likely it fails
    - Keep checkpoints of data structure

Internal algorithm

Query shipping vs. data shipping (I)



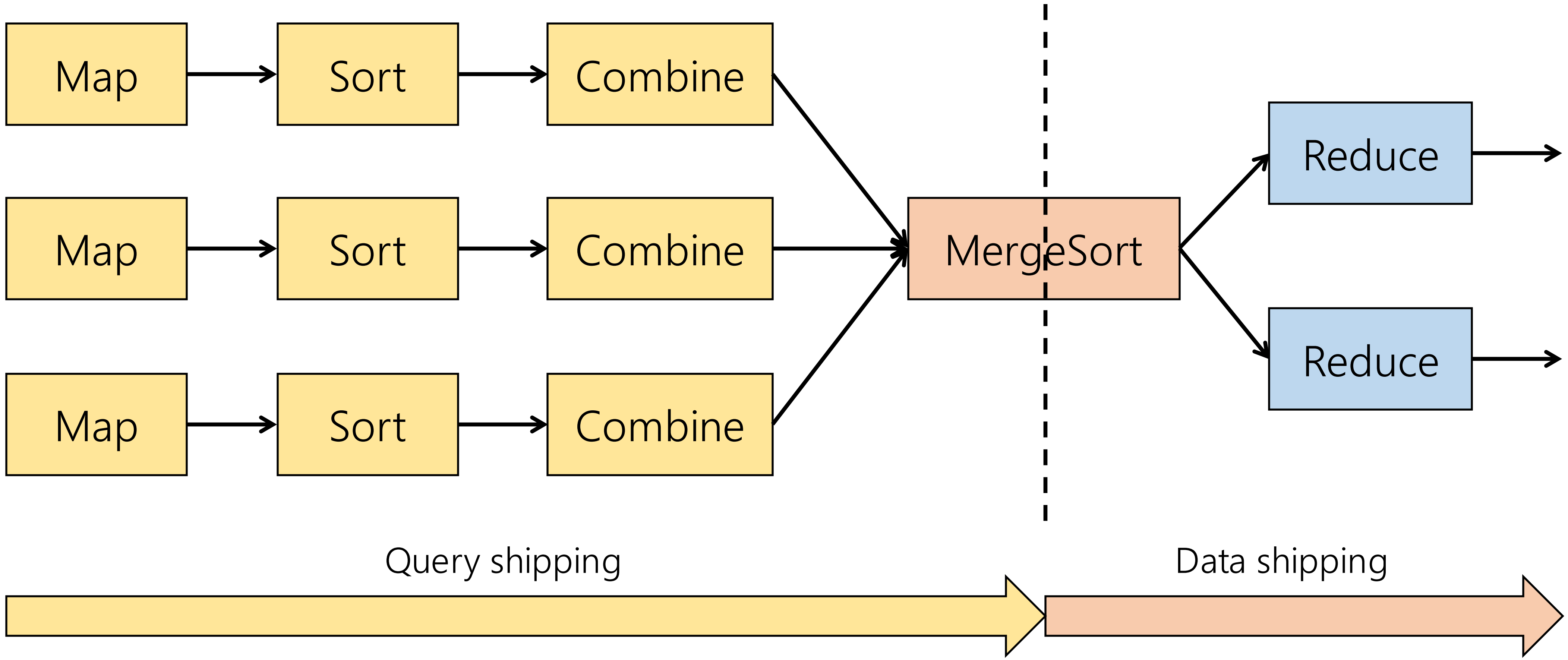
Word Count


Input
Splitting
Mapping
Combiner
Shuffling
Reducing
Result
Dear Bear River
Car Car River
Deer Car Beer
…
Dear Bear River
Car Car River
Dear Car Bear
Dear: 1
Bear: 1
River: 1
Dear: 1
Bear: 1
River: 1
Car: 2
River: 1
Car: 1
Car: 1
River: 1
Deer: 1
Car: 1
Bear: 1
Deer: 1
Car: 1
Bear: 1
Bear: 1
Bear: 1
Car: 1
Car: 1
Car: 1
Dear: 1
Dear: 1
River: 1
River: 1
Bear: 2
Car: 3
Deer: 2
River: 2
Bear: 2
Car: 3
Deer: 2
River: 2

Combiner

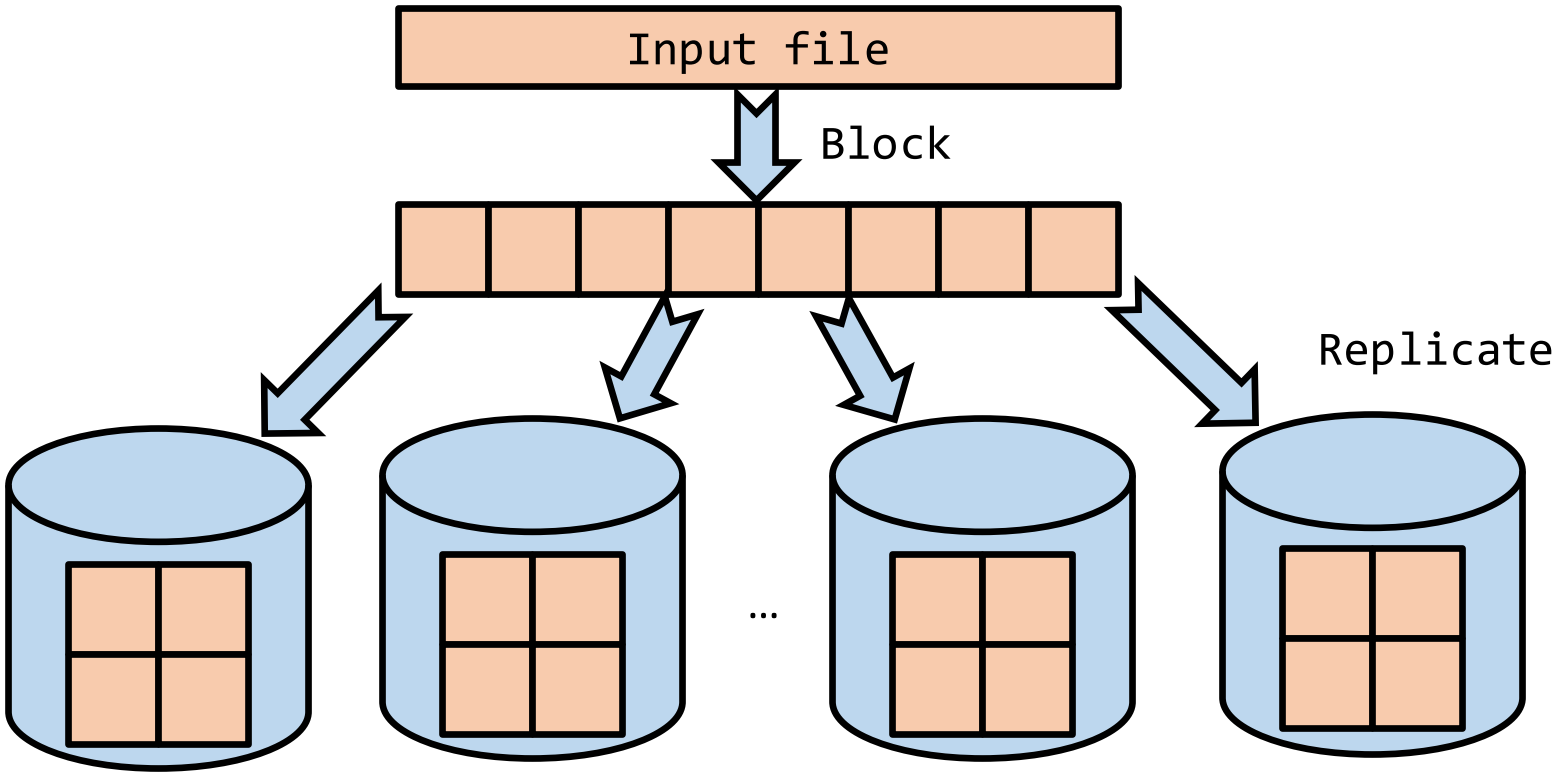
- Coincides with reducer function when it is:
  - Commutative
  - Associative
- Exploits data locality at the Mapper level
  - Data transfer diminished since Mapper outputs are reduced
    - Saving both network and storing intermediate results costs
- Only makes sense if `|I|/|O|>>#CPU`
  - Skewed distribution of input data improves early reduction of data

Query shipping vs. data shipping (II)



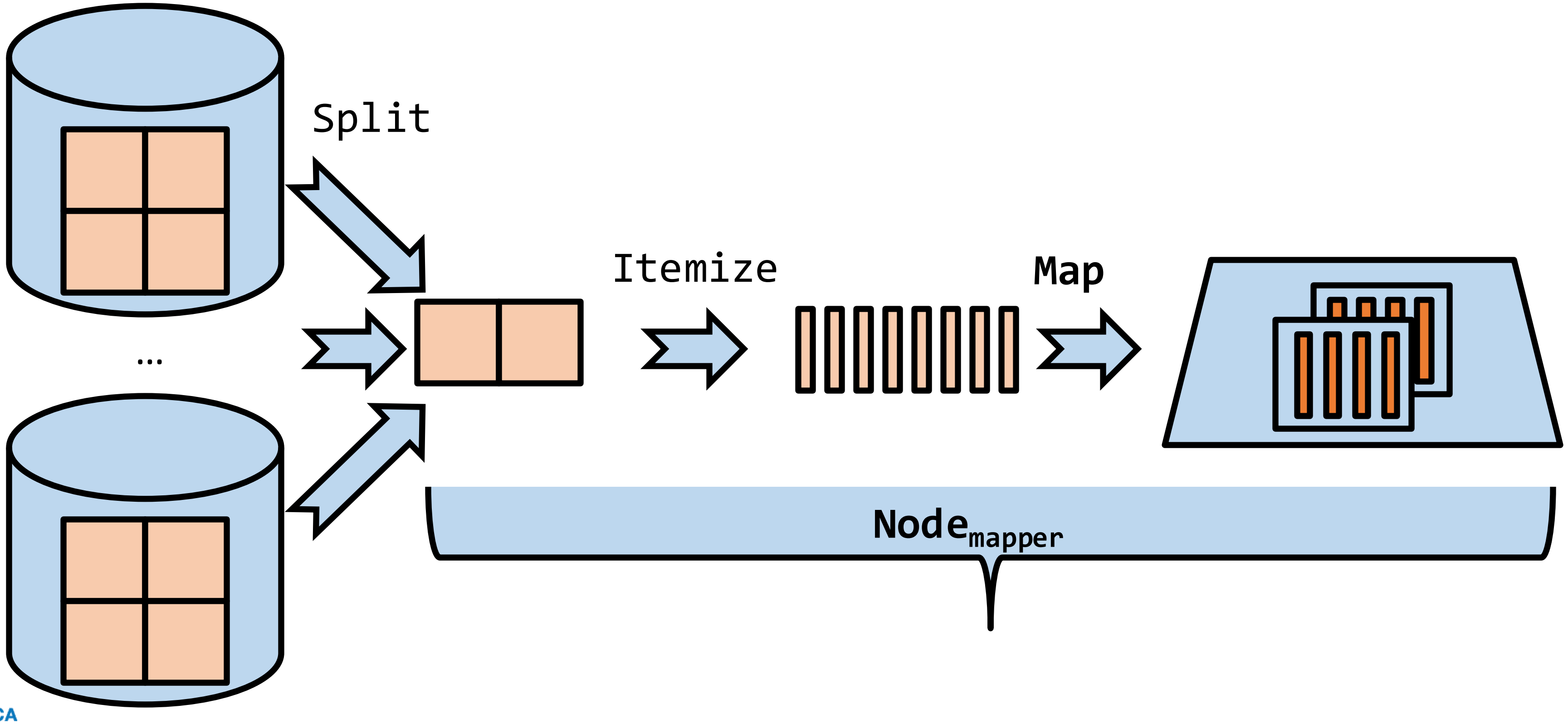
Algorithm: Data Load

1. Upload the data to the Cloud
   - Partition them into blocks
   - Using HDFS or any other storage (e.g., HBase, MongoDB, Cassandra, CouchDB, etc.)
2. Replicate them in different nodes



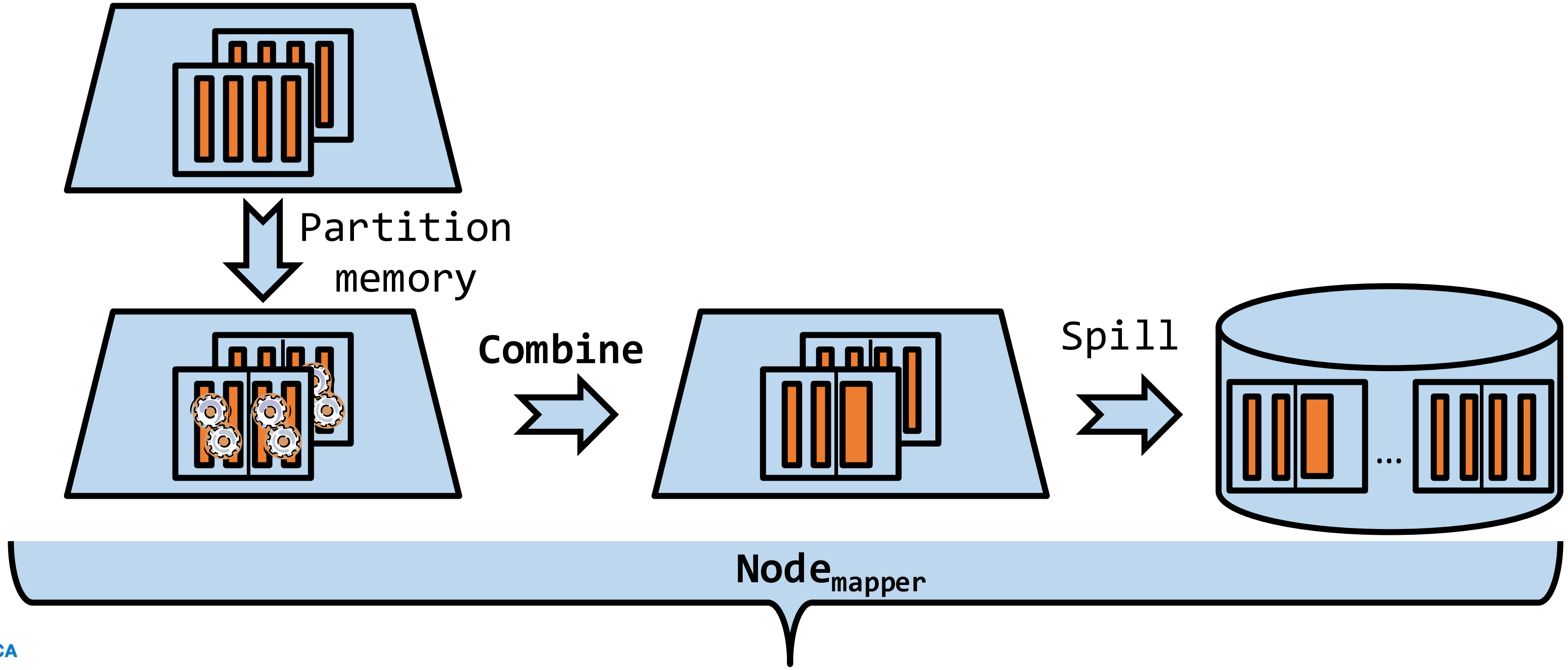
Algorithm: Map Phase (I)

3. Each mapper (i.e., JVM) reads a subset of blocks/chunks (i.e., split)
4. Divide each split into records
5. Execute the map function for each record and keep its results in memory
    - JVM heap used as a circular buffer



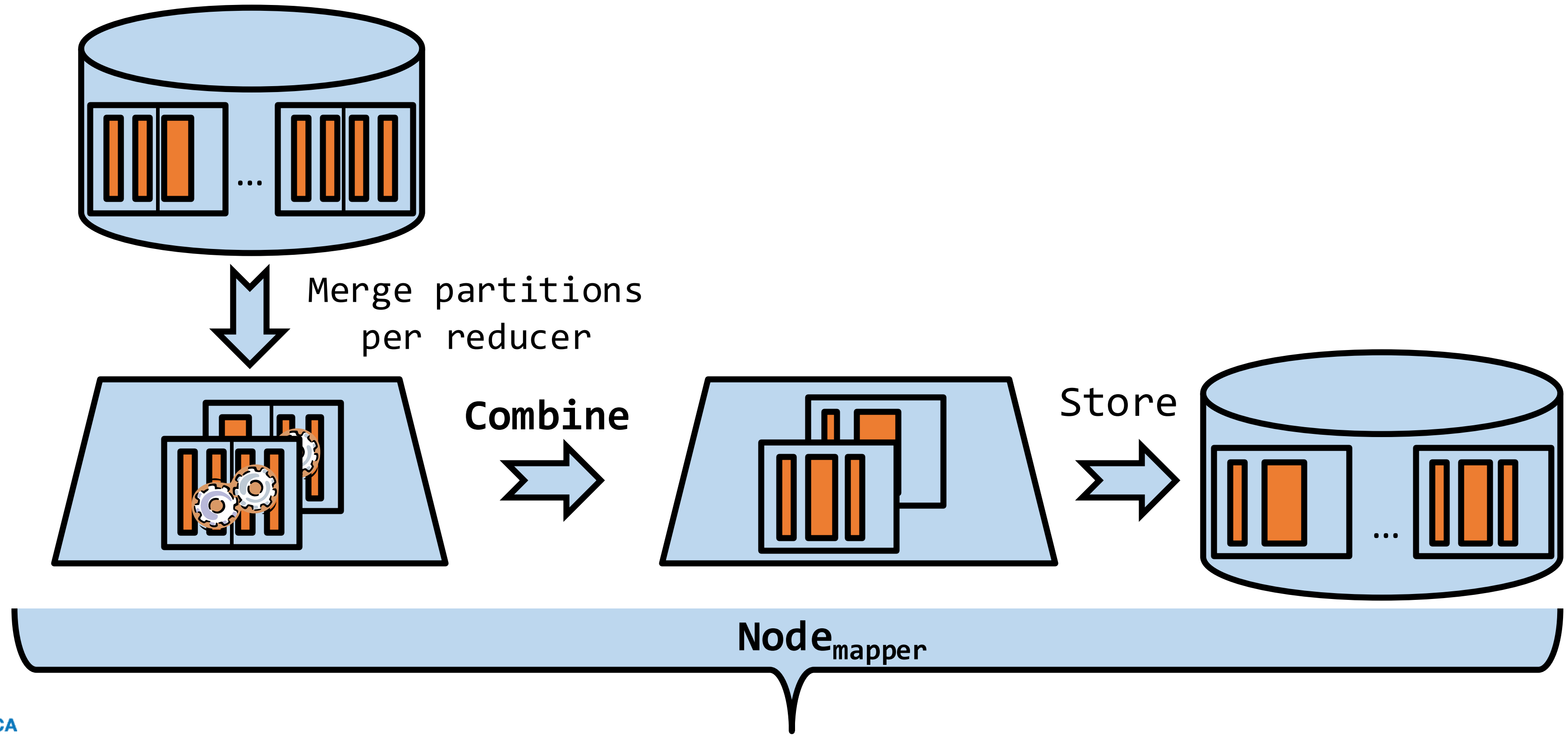
Algorithm: Map Phase (II)

6. Each time memory becomes full
   a) The memory is then partitioned per reducers
      - Using a hash function f over the new key
   b) Each memory partition is sorted independently
      - If a combine is defined, it is executed locally during sorting
   c) Spill partitions into disk (massive writing)



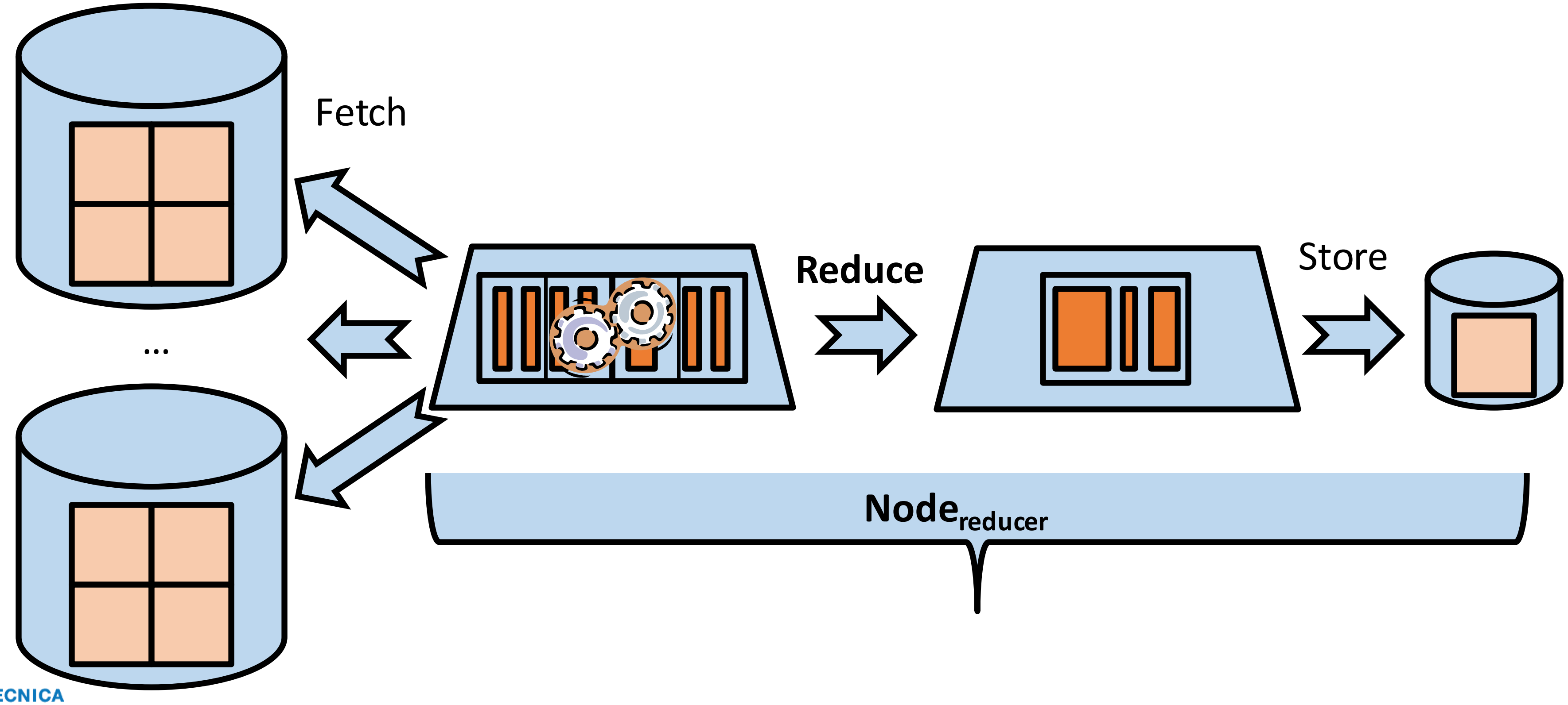
Algorithm: Map phase (III)

7. Partitions of different spills are merged
    - Each merge is sorted independently
    - Combine is applied again
8. Store the result into disk

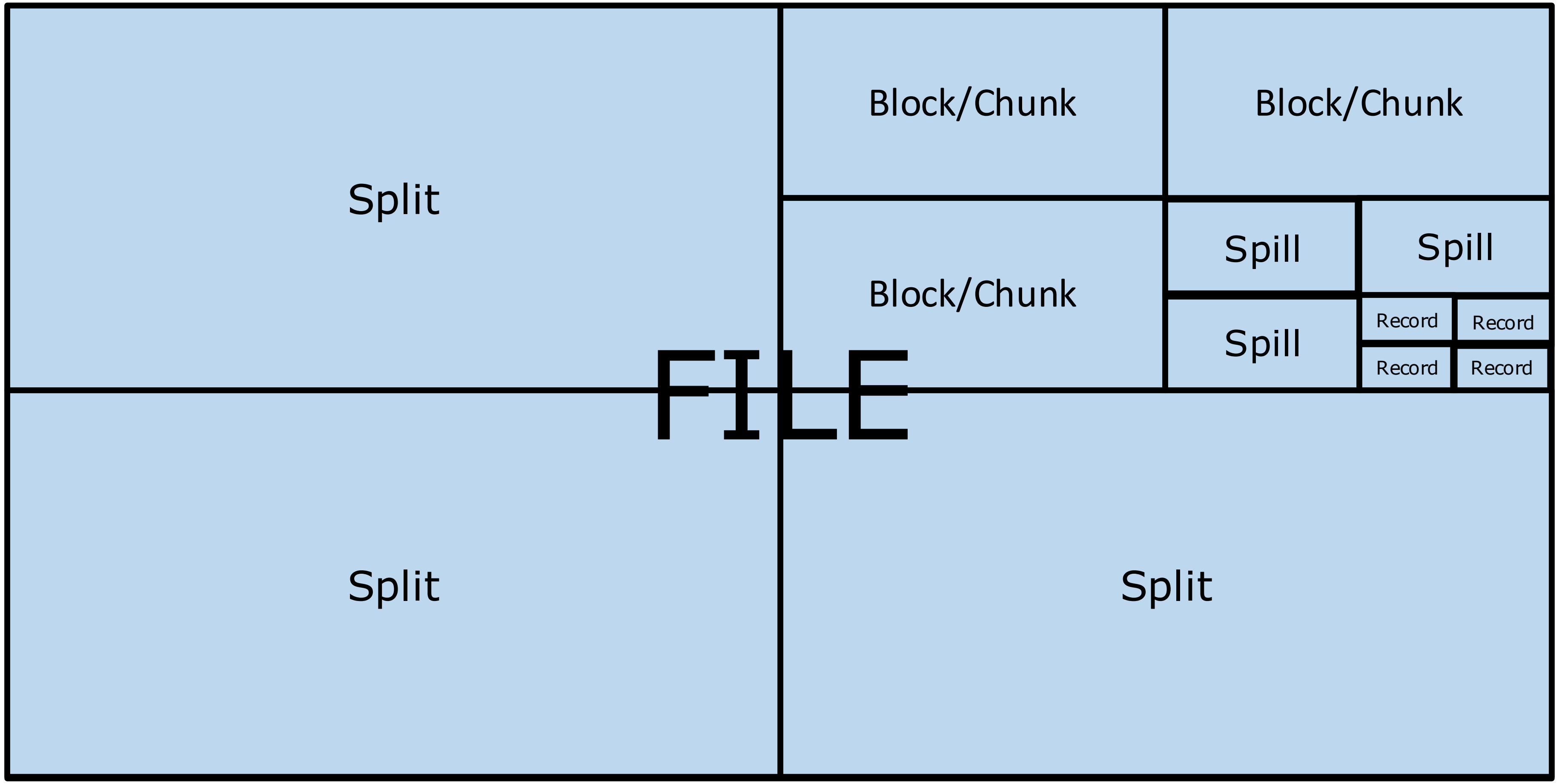


Algorithm: Shuffle and Reduce

9. Reducers fetch data through the network (massive data transfer)
10. Key-Value pairs are sorted and merged
11. Reduce function is executed per key
12. Store the result into disk



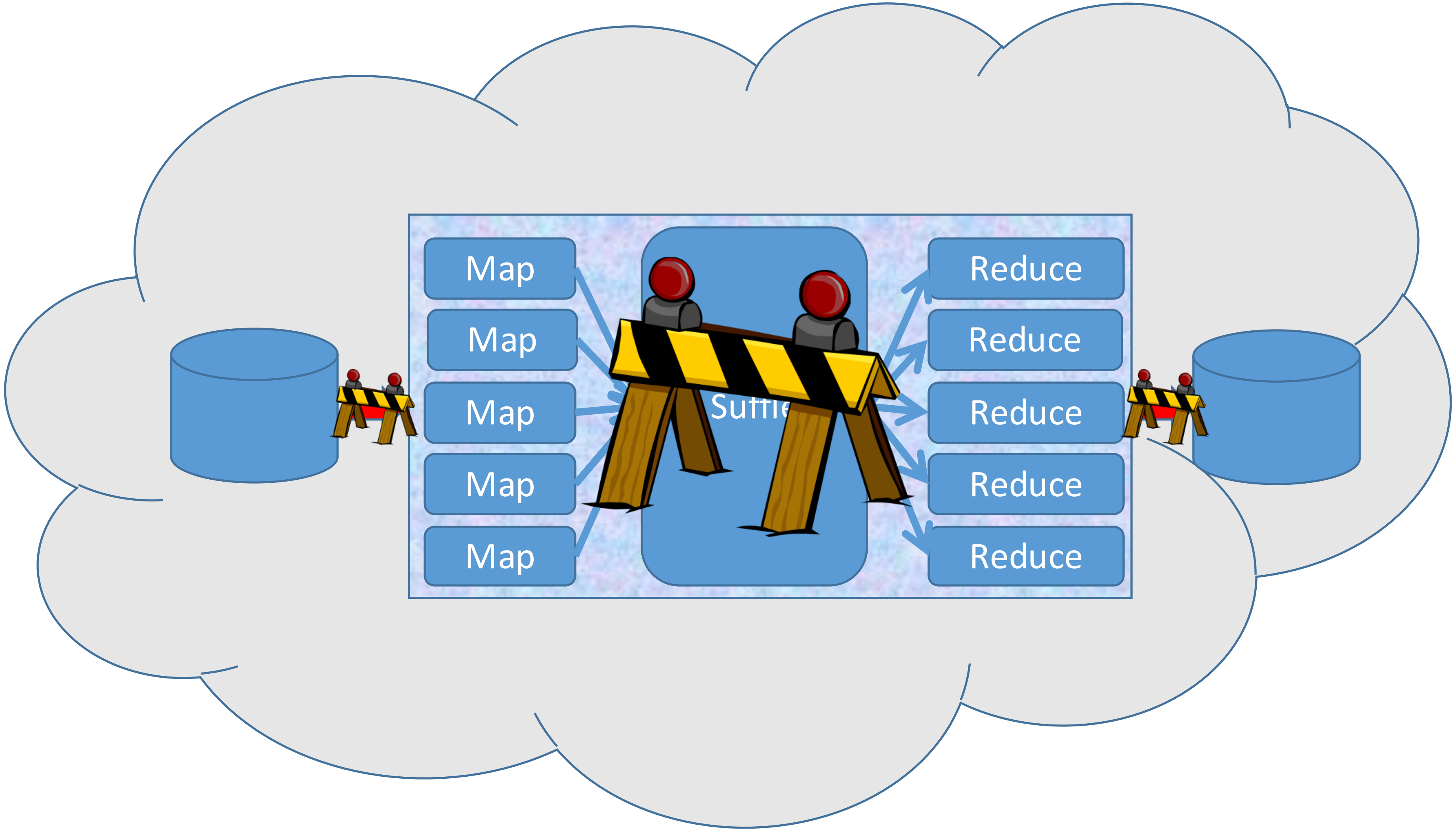
MapReduce objects



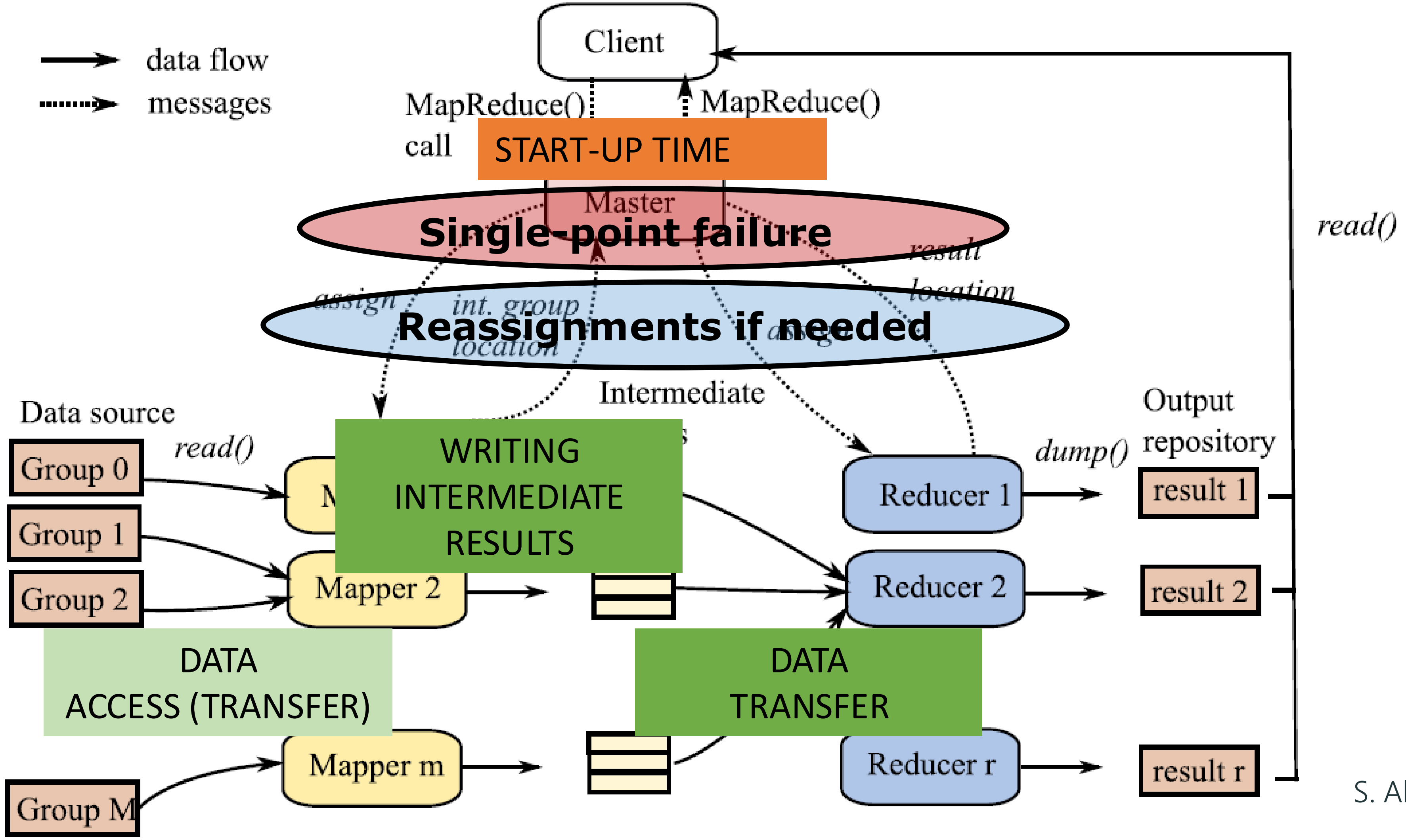
Record=Key-Value pair

Bottlenecks

Synchronization barriers



Tasks and Data Flows



S. Abiteboul et al.

Limitations

- Writes intermediate results to disk
  - Reduce tasks pull intermediate data
    - Improves fault tolerance
- Defines the execution plan on the fly
  - Schedules one block at a time
    - Adapts to workload and performance imbalance
- Does not provide transactions
  - Read-only system
    - Performs analytical tasks
- Cannot process data without decompressing them

Exercise

Executing a MapReduce job step by step

Activity: MapReduce

- **Objective: Understand/apply the algorithm underneath MapReduce**
- **Tasks:**
  1. **(40’) Reproduce step by step the MapReduce execution**
     - Consider the following data set:
       - `Block0: “a b b a c | c d c a e”`
       - `Block1: “a b d d a | b b c c f”`
     - Simulate the execution of the MapReduce code given the following configuration:
       - The map and reduce functions are those of the wordcount
         - The combine function shares the implementation of the reduce
       - There is one block per split
       - The “`|`” divides the records inside each block
         - We have two records per block
       - We can keep four pairs `[key,value]` per spill
       - We have two mappers and two reducers
         - `Machine0`, contains `block0`, runs `mapper0` and `reducer0`
         - `Machine1`, contains `block1`, runs `mapper1` and `reducer1`
       - The hash function used to shuffle data to the reducers uses the correspondence:
         - `{b,d,f}->0`
         - `{a,c,e}->1`

Closing

Summary

- MapReduce architecture
  - Processes
  - Fault-tolerance mechanisms
  - Bottlenecks
    - Synchronization barriers
- MapReduce detailed algorithm
  - Query shipping
  - Data shipping
- MapReduce limitations

References

- J. Dean et al. MapReduce: Simplified Data Processing on Large Clusters. OSDI'04
- A. Pavlo et al. A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD, 2009
- J. Dittrich et al. Hadoop++: Making a Yellow Elephant Run Like a Cheetah (Without It Even Noticing). Proc. VLDB Endow. 3(1-2), 2010
- M. Stonebraker et al. MapReduce and parallel DBMSs: friends or foes? Communication of ACM 53(1), 2010
- S. Abiteboul et al. Web data management. Cambridge University Press, 2011
- A. Rajaraman et al. Mining massive data sets. Cambridge University Press, 2012
- P. Sadagale and M. Fowler. NoSQL distilled. Addison-Wesley, 2013