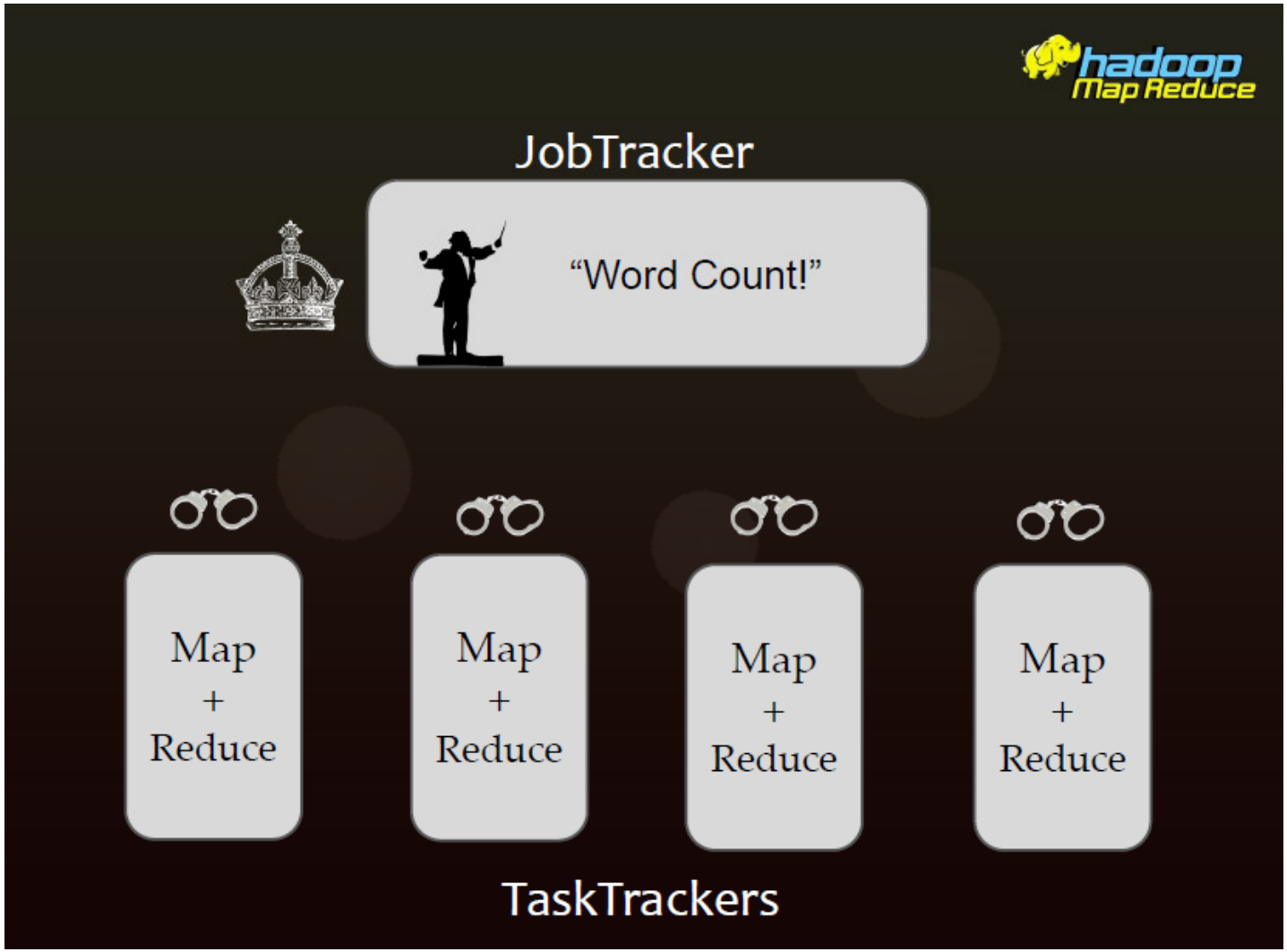
Session 4

MapReduce v1

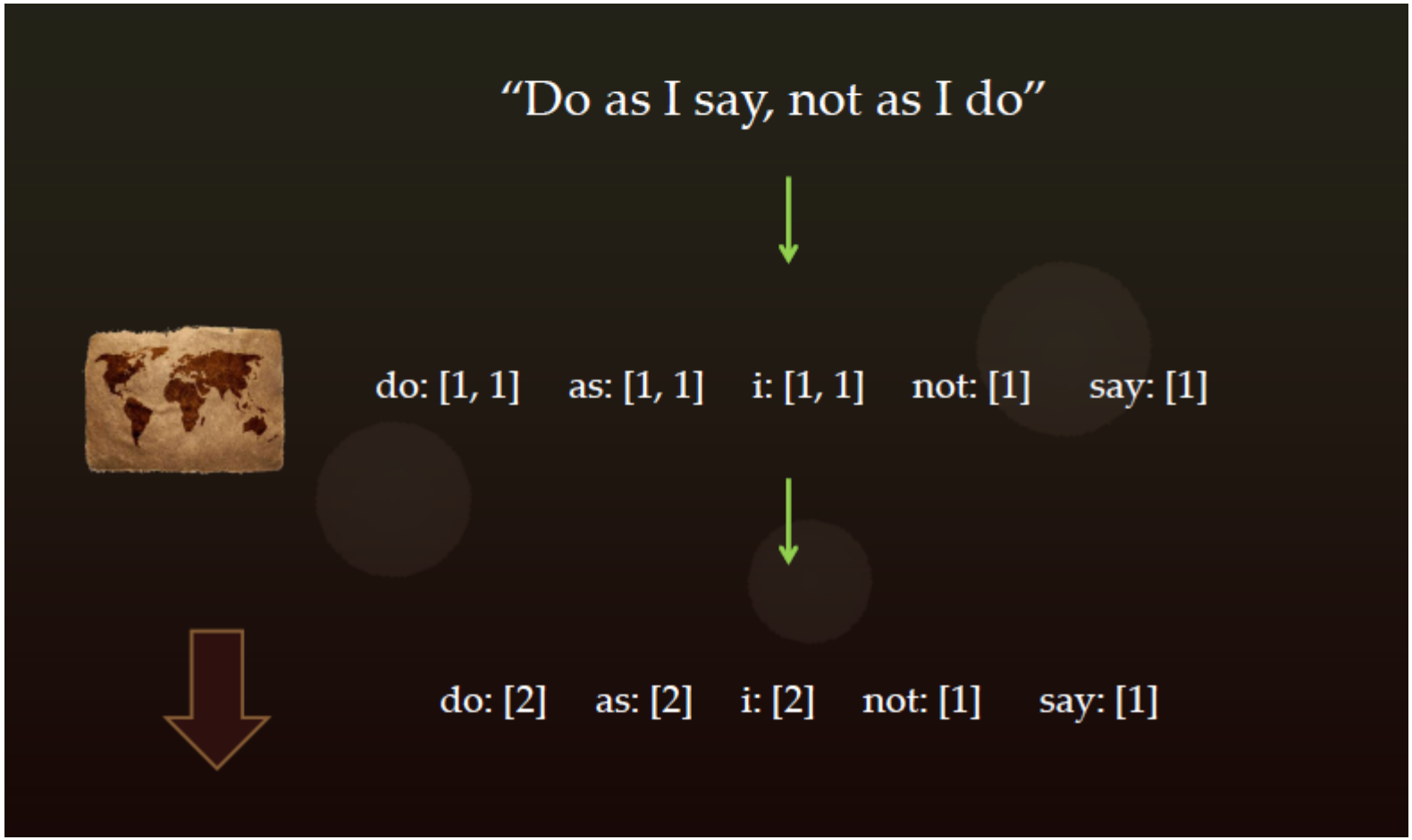
MapReduce – What??

- MapReduce is a programming model for efficient distributed computing
- It works like a Unix pipeline
  - cat input | grep | sort | uniq -c | cat > output
  - **Input | Map |** Shuffle & Sort **| Reduce | Output**
- Efficiency from
  - Streaming through data, reducing seeks
  - Pipelining
- A good fit for a lot of applications
  - Log processing
  - Web index building

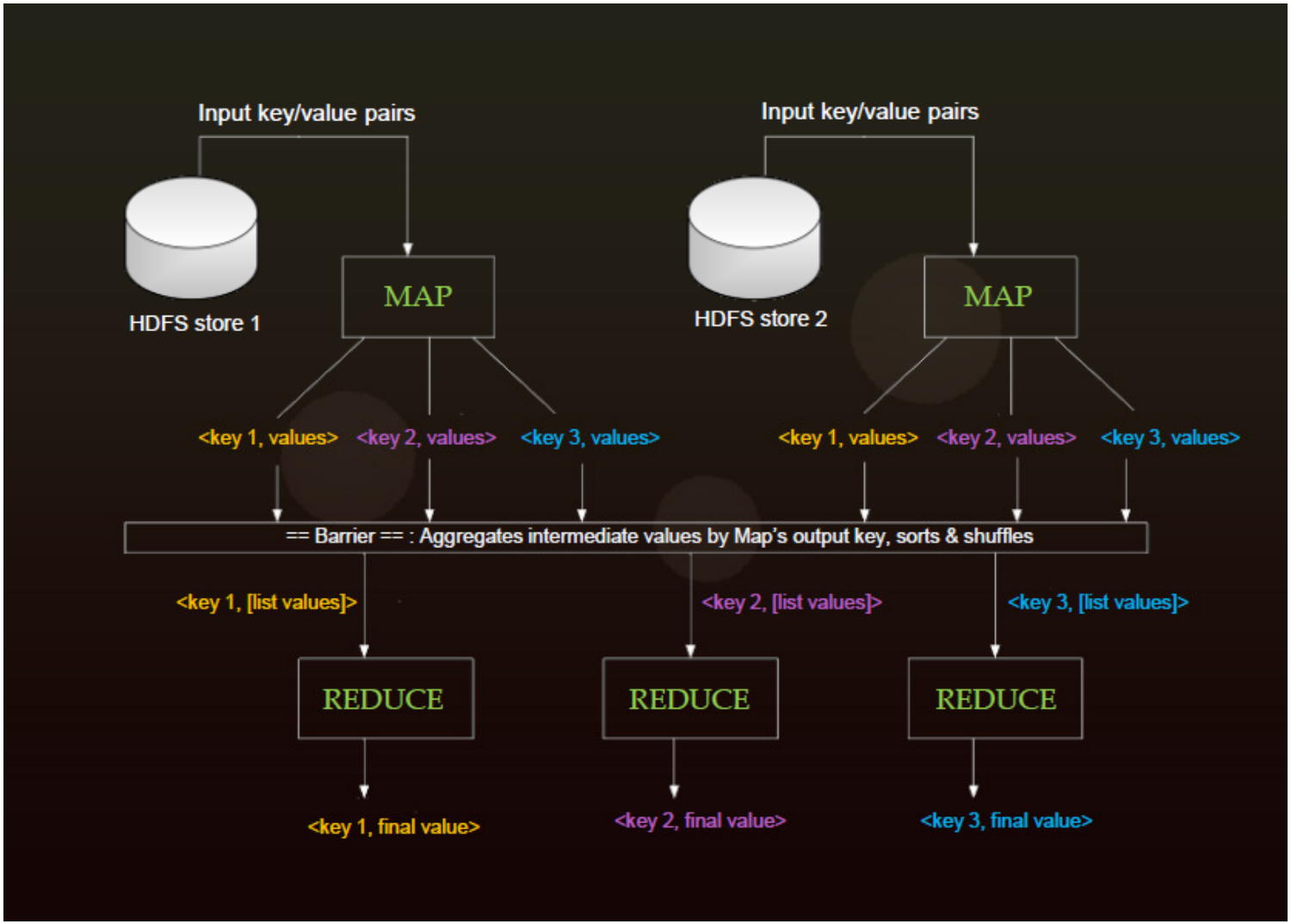
MapReduce Daemons



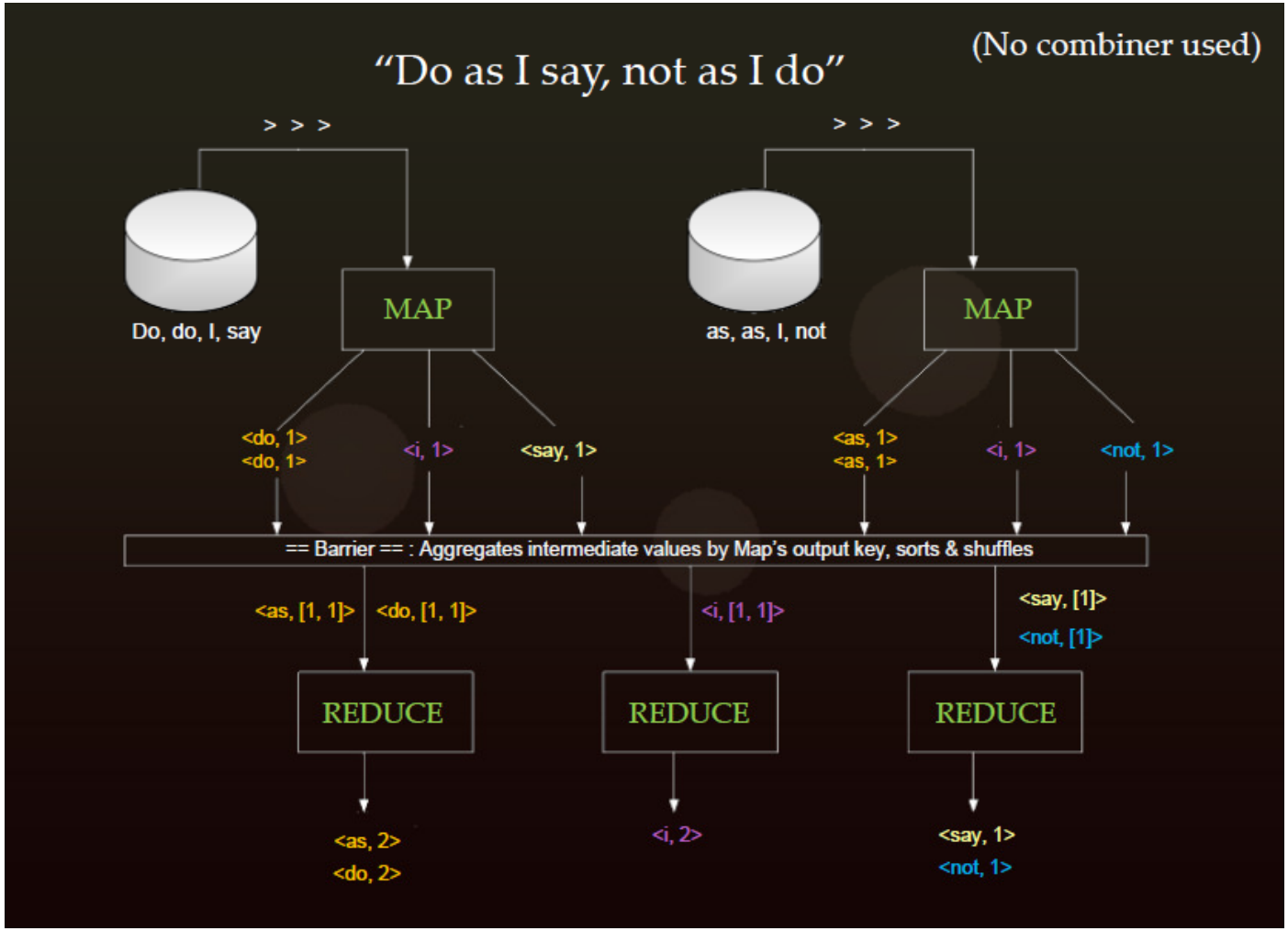
Map Reduce


"Do as I say, not as I do"
do: [1, 1] as: [1, 1] i: [1, 1] not: [1] say: [1]
do: [2] as: [2] i: [2] not: [1] say: [1]

Algorithm


Input key/value pairs
Input key/value pairs
HDFS store 1
HDFS store 2
MAP
MAP
<key 1, values>
<key 2, values>
<key 3, values>
<key 1, values>
<key 2, values>
<key 3, values>
== Barrier == : Aggregates intermediate values by Map's output key, sorts & shuffles
<key 1, [list values]>
<key 2, [list values]>
<key 3, [list values]>
REDUCE
REDUCE
REDUCE
<key 1, final value>
<key 2, final value>
<key 3, final value>

Word Count !!!


"Do as I say, not as I do"
(No combiner used)
> > >
> > >
Do, do, I, say
as, as, I, not
MAP
MAP
<do, 1>
<do, 1>
<i, 1>
<say, 1>
<as, 1>
<as, 1>
<i, 1>
<not, 1>
== Barrier == : Aggregates intermediate values by Map's output key, sorts & shuffles
<as, [1, 1]>
<do, [1, 1]>
<i, [1, 1]>
<say, [1]>
<not, [1]>
REDUCE
REDUCE
REDUCE
<as, 2>
<do, 2>
<i, 2>
<say, 1>
<not, 1>

Combiner

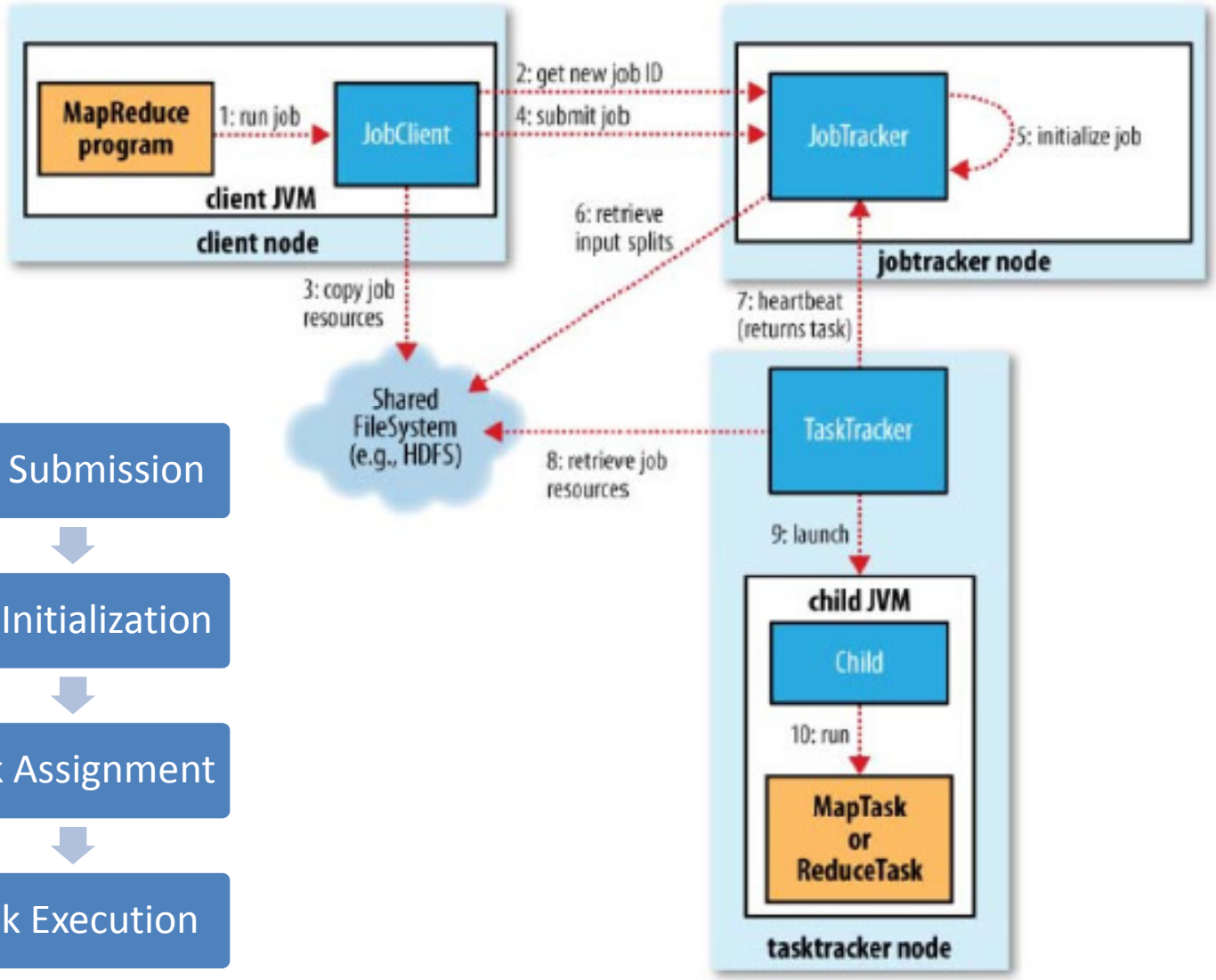
- ✓ Helper for the reducer
- ✓ Reduces the # of key/value pairs sent to a Reducer
- ✓ Can be applied 0, 1 or many times
- ✓ Does not affect final Reducer output
- ✓ Usually the same code as a Reducer
- ✓ Don’t use if every K/V pair emitted from Mapper is unique
- ✓ Map -> Combiner -> Partitioner -> Sort -> Reduce

Partitioner

- ✓ Don’t use one reducer for sorting all the data
- ✓ Multiple reducers require a partitioner to direct k/v pairs data flow
- ✓ A subset of data (partition) are inputs to reduce tasks
- ✓ Each Map may emit k/v pair for any partition
- ✓ Default: **HashPartitioner**

Task Tracker Heartbeats

Anatomy of a Map Reduce Job Run



- Job Submission
- Jon Initialization
- Task Assignment
- Task Execution

Job Submission

- **JobClient class**
  - ✓ The runJob() method creates a new instance of a **JobClient**
  - ✓ Then it calls the submitJob() on this class

- **Simple verifications on the Job**
  - ✓ Is there an output directory?
  - ✓ Are there any input splits?
  - ✓ Can I copy the JAR of the job to HDFS?

NOTE: The JAR of the job is replicated 10 times

Job Initialization

- **The JobTracker is responsible for:**
  - ✓ Create an object for the job
  - ✓ Encapsulate its tasks
  - ✓ Bookkeeping with the tasks' status and progress

- **This is where the scheduling happens**
  - ✓ JobTracker performs scheduling by maintaining a queue
  - ✓ Queueing disciplines are pluggable

- **Compute mappers and reducers**
  - ✓ JobTracker retrieves input splits (computed by JobClient)
  - ✓ Determines the number of Mappers based on the number of input
  - ✓ splits
  - ✓ Reads the configuration file to set the number of Reducers

Task Assignment

- **Hearbeat-based mechanism**
  - ✓ TaskTrackers periodically send hearbeats to the JobTracker
  - ✓ TaskTracker is alive
  - ✓ Heartbeat contains also information on availability of the TaskTrackers to execute a task
  - ✓ JobTracker piggybacks a task if TaskTracker is available

- **Selecting a task**
  - ✓ JobTracker first needs to select a job (i.e. scheduling)
  - ✓ TaskTrackers have a fixed number of slots for map and reduce tasks
  - ✓ JobTracker gives priority to map tasks (WHY?)

- **Data locality**
  - ✓ JobTracker is topology aware
  - ✓ Useful for map tasks
  - ✓ Unused for reduce tasks

Task Execution

- **Task Assignement is done, now TaskTrackers can execute**
  - ✓ Copy the JAR from the HDFS
  - ✓ Create a local working directory
  - ✓ Create an instance of TaskRunner
- **TaskRunner launches a child JVM**
  - ✓ This prevents bugs from stalling the TaskTracker
  - ✓ A new child JVM is created per InputSplit
  - ✓ Can be overriden by specifying JVM Reuse option, which is very useful for custom, in-memory, combiners
- **Streaming and Pipes**
  - ✓ User-defined map and reduce methods need not to be in Java
  - ✓ Streaming and Pipes allow C++ or python mappers and reducers

Speculative Execution

- ✓ On by default
- ✓ Re-launches slow tasks
- ✓ If a node is slow b/c of software/hardware misconfiguration, Speculative Execution kicks in
- ✓ Hadoop doesn't try to diagnose or fix slow tasks
- ✓ NOT a race condition
- ✓ This is an optimization, not a reliability feature
- ✓ A bug can cause both cloned tasks to hang or crash

Distributed Cache

- ✓ Distributes a file to all nodes so it is available while M/R job is running
- ✓ Typically a helper library for code
- ✓ Used when joining a big data source to a much smaller source
- ✓ Example: phone company's customers data joined to call log
- ✓ Reduces network I/O b/c files only need to be downloaded once
- ✓ Files are read only
- ✓ Dist. Cache files are stored in the Linux Filesystem

Failures

Child Task Failure:

- ✓ Fails when it throws an uncaught exception, exists with non-zero code or fails to report progress to TT within a specified time.
- ✓ TT detects failure and reports it to the JT via next heartbeat. JT can retry that task up to 4 times by default.
- ✓ Job-level and global blacklists (24h) exist.

TT / worker node Failure:

- ✓ JT detects this when heartbeats stop coming from a TT. Client is not aware of these failures and just sees the job slow down.

JT Failure:

- ✓ Single Point of Failure for MR. The internal state about currently executing jobs is lost and all running tasks will eventually fail

Zero Reducers

- Frequently, we only need to run a filter on the input data
  - No sorting or shuffling required by the job
  - Set the number of reduces to 0
  - Output from maps will go directly to OutputFormat and disk

How many Maps and Reduces

- Maps
  - Usually as many as the number of HDFS blocks being processed, this is the default
  - Else the number of maps can be specified as a hint
  - The number of maps can also be controlled by specifying the *minimum split size*
- Reduces
  - Unless the amount of data being processed is small
    - *0.95*num_nodes*mapred.tasktracker.reduce.tasks.maximum*

Hadoop Streaming

- Mapper and Reducer receive data from stdin and output to stdout

- Hadoop takes care of the transmission of data between the map/reduce tasks
  - ✓ It is still the programmer's responsibility to set the correct key/value
  - ✓ Default format: "key \t value\n"

- Simple and powerful interface for programming
  - ✓ Application developers do not need to learn hadoop java APIs
  - ✓ Good for simple, adhoc tasks

MapReduce Lab

- **Objectives**
  - ✓ Run a Java word count example with a bundled wordcount example
  - ✓ Run custom Java Map Reduce code for finding max temp and take a first glance at the Java code
  - ✓ Navigating MapReduce Web UI
  - ✓ MapReduce Counters
  - ✓ Change cluster Map Task Capacity (mapred.tasktracker.map.tasks.maximum)

Thank You