COSC 460 Lecture 21: Map Reduce

Instructor: Michael Hay

Fall 2018

Credits: Slides adapted from Franklin, Miklau, and Kot

Recap: Programming Model

- Borrows from functional programming (note: ideas from Haskell, etc. but not implemented in Haskell)
- Users implement two functions ([…] denotes list)

map (k, v) → [(k', v')]

  - map( ) takes *single* input key-value pair and produces one or more *intermediate* results: (output key, value) pairs
  - after map phase over, system combines all the intermediate values for a given output key together into a list.

reduce (k', [v']) → [v'']

  - reduce( ) combines intermediate values into one or more final values for that output key

let's look at python version

Exercises

- Input: a relation of web logs
  - Key: tuple_id, Value: (ipaddr, url, category, timestamp)
- Tasks
  1. Urls that have at least V visits (entries in log)
  2. Categories that have at least S *distinct* urls
  3. Categories that have at least S urls with at least V visits each (hint: may require multiple rounds of map-reduce)

Exercises

- Input: a friends relation *Friend(user, friend)*
  - Key: tuple_id, Value: tuple *(u,f)*
- Tasks
  1. For each user, number of friends
  2. Set of pairs *(u, fof)* where u is a user and fof is a friend of a friend
  3. For each (u,f) pair, the number of mutual friends (hint: may require multiple rounds of map-reduce)

Exercises

- Input: a relation of numbers R(*x)*
  - Key: tuple_id, Value: x
- Tasks
  1. Largest number
  2. select AVG(x) from R
  3. select x, COUNT(x) from R group by x
  4. select count(distinct x) from R

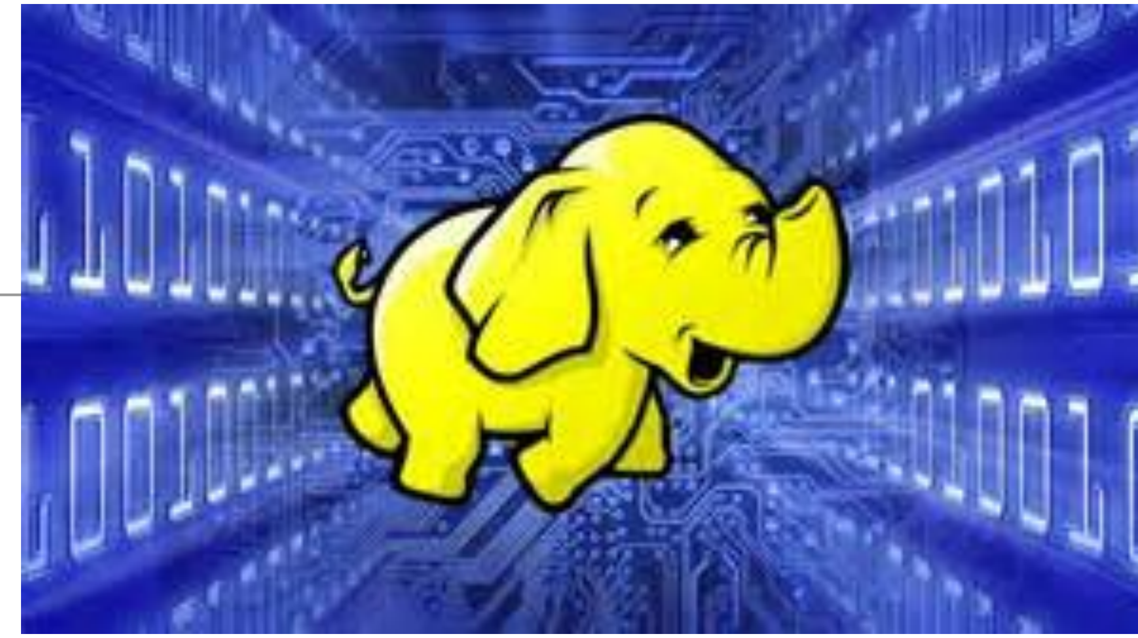
Map Reduce Implementation

- System setup
  - Data is stored using a distributed file system.
  - Computations parallelized over many machines.
- Key concerns
  - Coordination
  - Fault-tolerance
  - Data distribution, especially “shuffling” data from map to reduce
  - Load balancing

Hadoop

- An open-source implementation in Java
- Uses HDFS for stable storage
- Download: http://lucene.apache.org/hadoop/

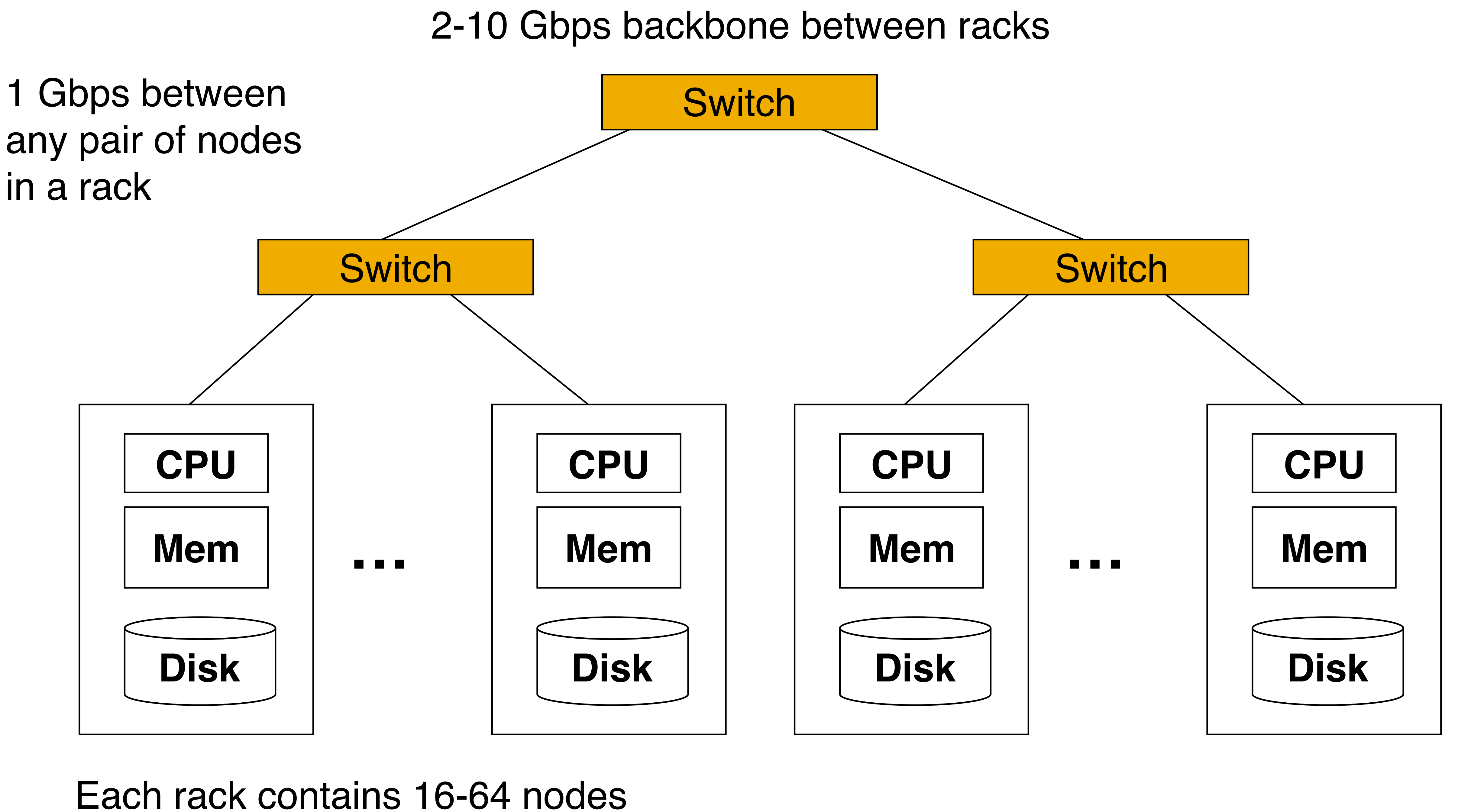
Hadoop (2005?…)



- Open-source project initiated by Cutting and Cafarella
- In 2010 Facebook claimed that they had the largest Hadoop cluster in the world with 21 PB of storage. (1 PB = 1000 TB)
- On July 27, 2011 announced growth to 30 PB.
- On June 13, 2012 announced growth to 100 PB.
- On November 8, 2012 announced warehouse grows by roughly half a PB per day.

adapted from Mike Franklin, Berkeley

Cluster Architecture



In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

adapted from J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Storage Infrastructure

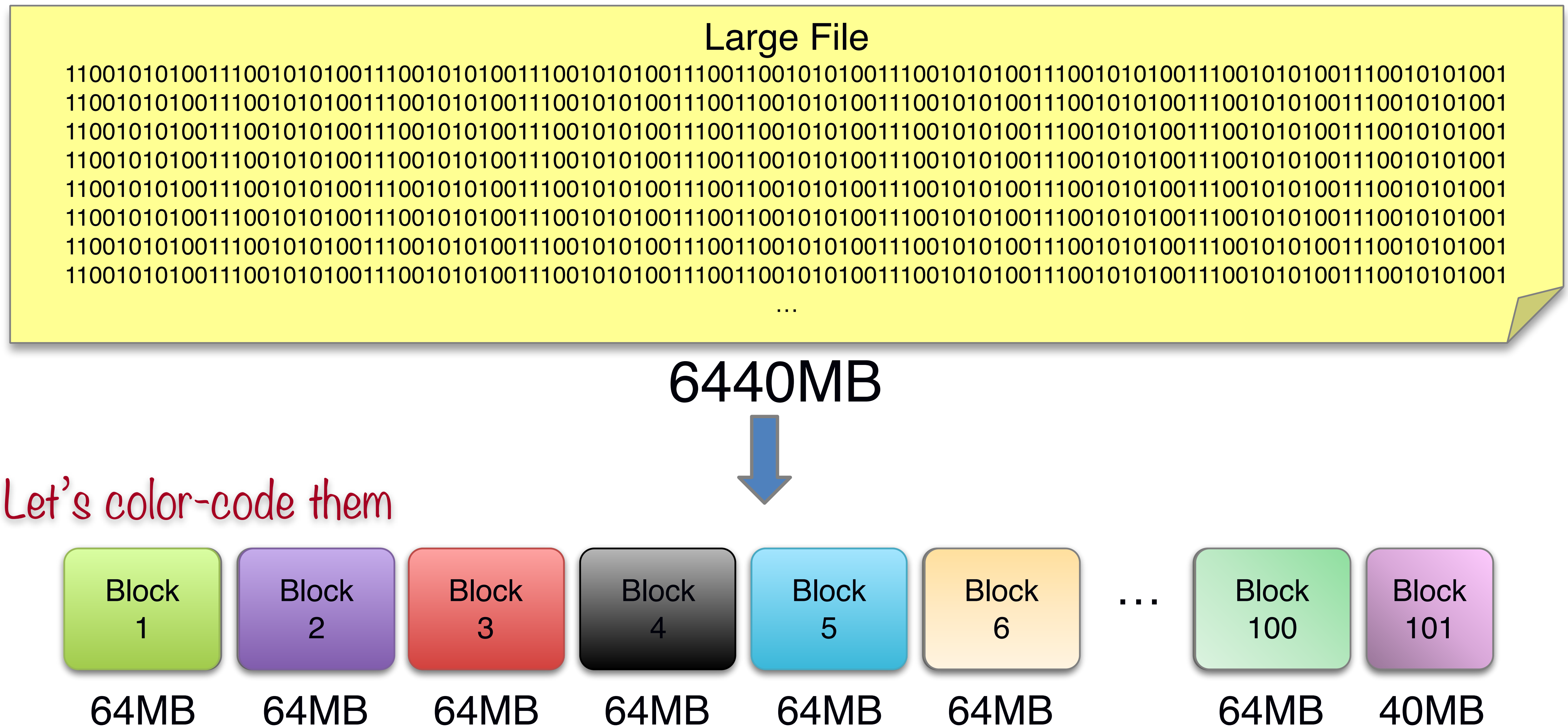
- Problem:
  - If nodes fail, how to store data persistently?
- Answer: Distributed File System:
  - Provides global file namespace
  - Google GFS; Hadoop HDFS;

adapted from J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Hadoop Distributed File System

- Underpinnings of the entire Hadoop ecosystem
- Traditional hierarchical file organization: directories and files
- Highly portable
- HDFS properties:
  - Scalable to 1000s of nodes
  - Assume failures (hardware and software) are common
  - Can store *very large* files
  - Append only workloads: Write once, read multiple times

File Splits



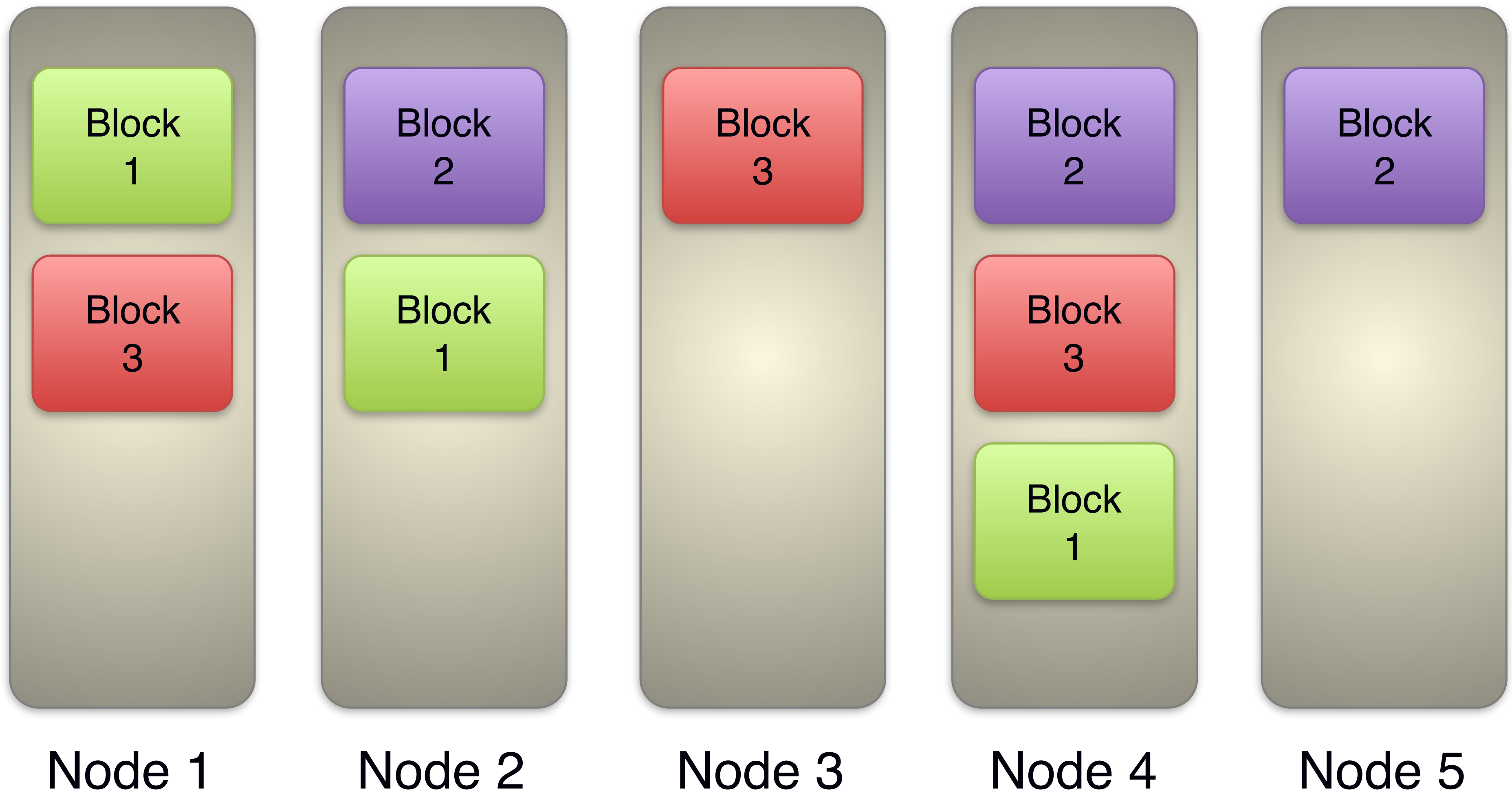
e.g., Block Size = 64MB

Files are composed of set of blocks

- Typically 64MB in size
- Each block is stored as a separate file in the local file system of a node

adapted from Mike Franklin, Berkeley

Block Placement

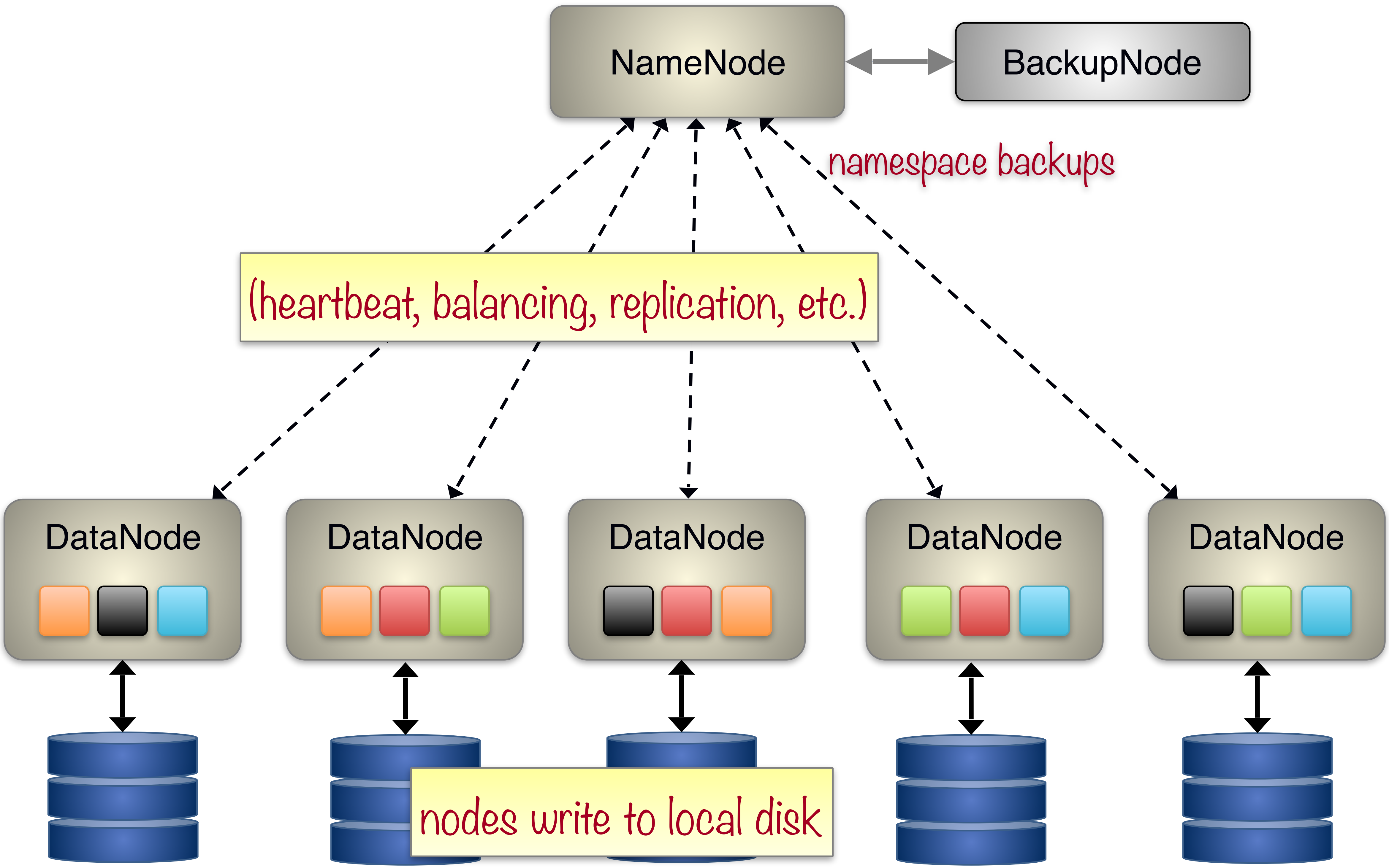


- **<u>Default placement policy:</u>** ... factor = 3
  - **First copy** is written to the node creating the ...
  - **Second copy** i... ...within the **same rack** (to minimize cross-rack
  - **Third copy** is written to a data node in a **different rack** (to tolerate switch failures)

<u>Objectives</u>: load balancing, fast access, fault tolerance

adapted from Mike Franklin, Berkeley

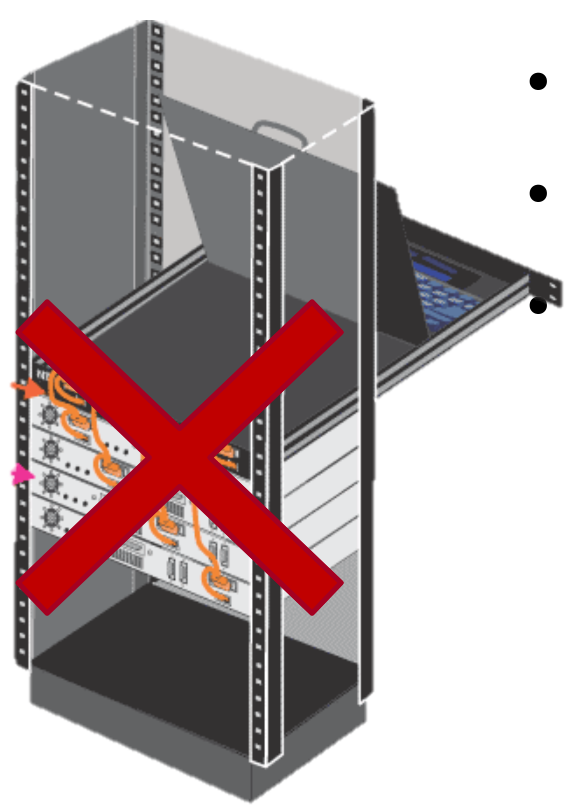
HDFS Architecture



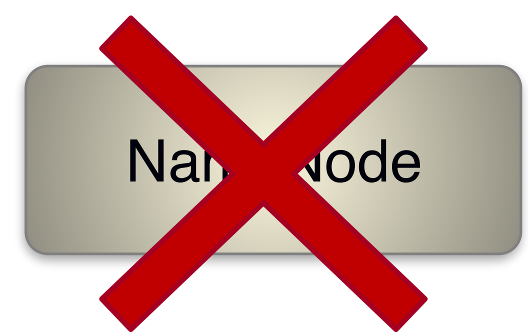
adapted from Mike Franklin, Berkeley

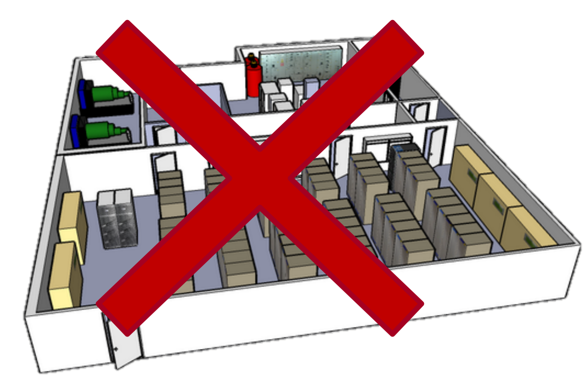
Failures, Failures, Failures

- HDFS was designed with the expectation that failures (both hardware and software) would occur frequently



- Failure types:
  - Disk errors and failures
  - DataNode failures
  - Switch/Rack failures
  - NameNode failures
  - Datacenter failures





Map-Reduce: Environment

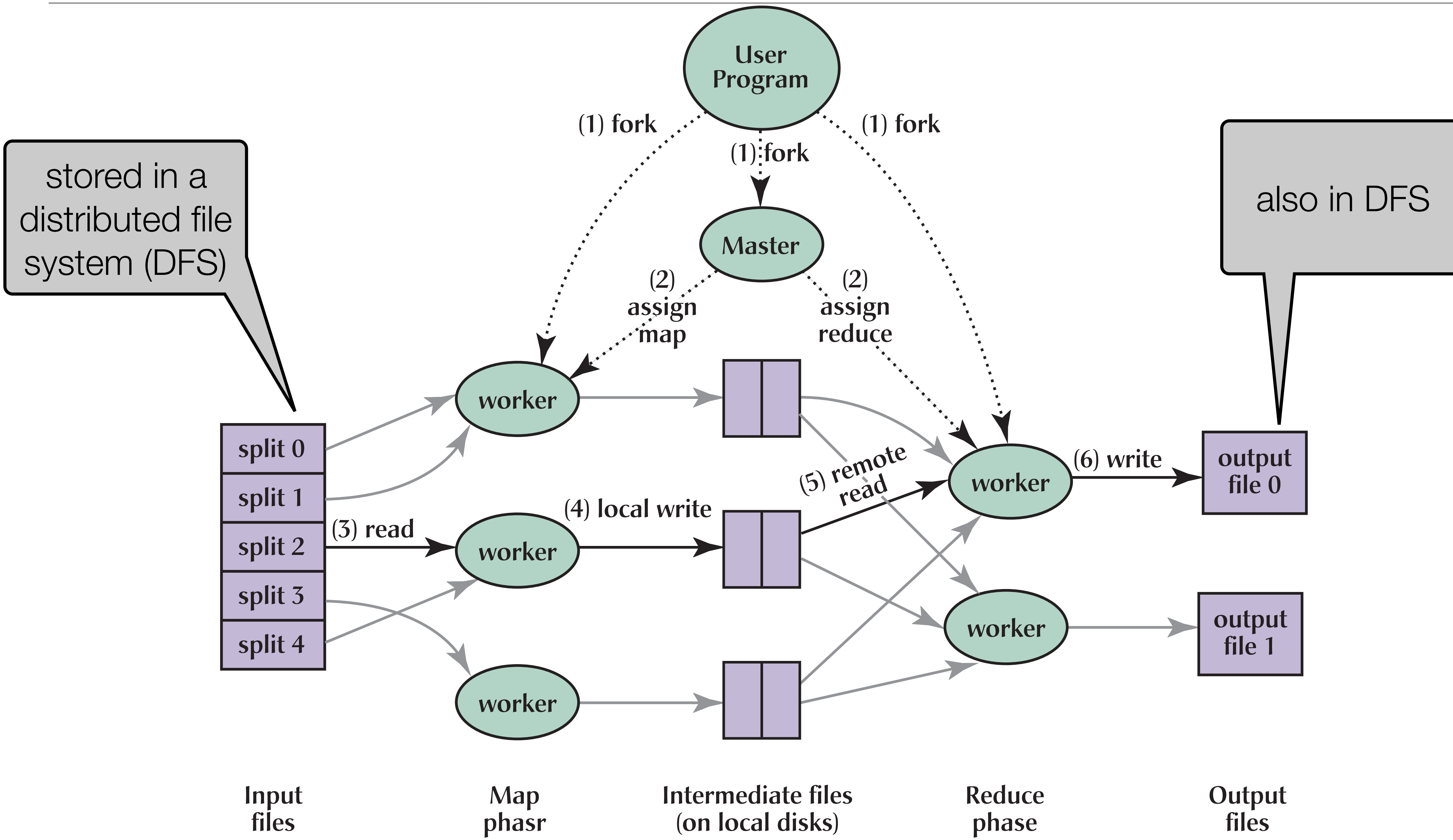
Map-Reduce environment takes care of:

- Partitioning the input data across machines (DFS)
- Scheduling the program's execution across a set of machines (tasks and workers)
- Performing the group by key step (using "shuffle" and sort)
- Handling machine failures
- Managing required inter-machine communication

adapted from J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Execution overview

W workers
M map tasks
R reduce tasks



J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. Communications of the ACM, 51(1):107–113, 2008.

Intermediate results

Instructions: ~1 minute to think/answer on your own; then discuss with neighbors; then I will call on one of you

Suppose a given map reduce job has M map tasks and R reduce tasks and there are W workers available. How many intermediate files are created?

What information does master need to keep track of?

Key concerns

- Coordination
- Fault-tolerance
  - one or more machines may fail *during* computation
- Data distribution
  - especially “shuffling” data from map to reduce
- Load balancing

Coordination: Master

- **Master node takes care of coordination:**
    - **Task status:** (idle, in-progress, completed)
    - **Idle tasks** get scheduled as workers become available
    - When a map task completes, it sends the master the location and sizes of its *R* intermediate files, one for each reducer
    - Master pushes this info to reducers

- Master pings workers periodically to detect failures

adapted from J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Dealing with Failures

- Map worker failure*
  - Map tasks reset to idle if in-progress or completed (why?)
  - Reduce workers are notified when map task is executed by another worker (which they can ignore in some cases — see “stragglers”).
- Reduce worker failure
  - Only in-progress tasks are reset to idle (why not complete?)
  - Reduce task is restarted
- Master failure
  - MapReduce task is aborted and client is notified

* failure = master not getting timely response, worker may still be working!
system must handle workers that “come back from the dead”

adapted from J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

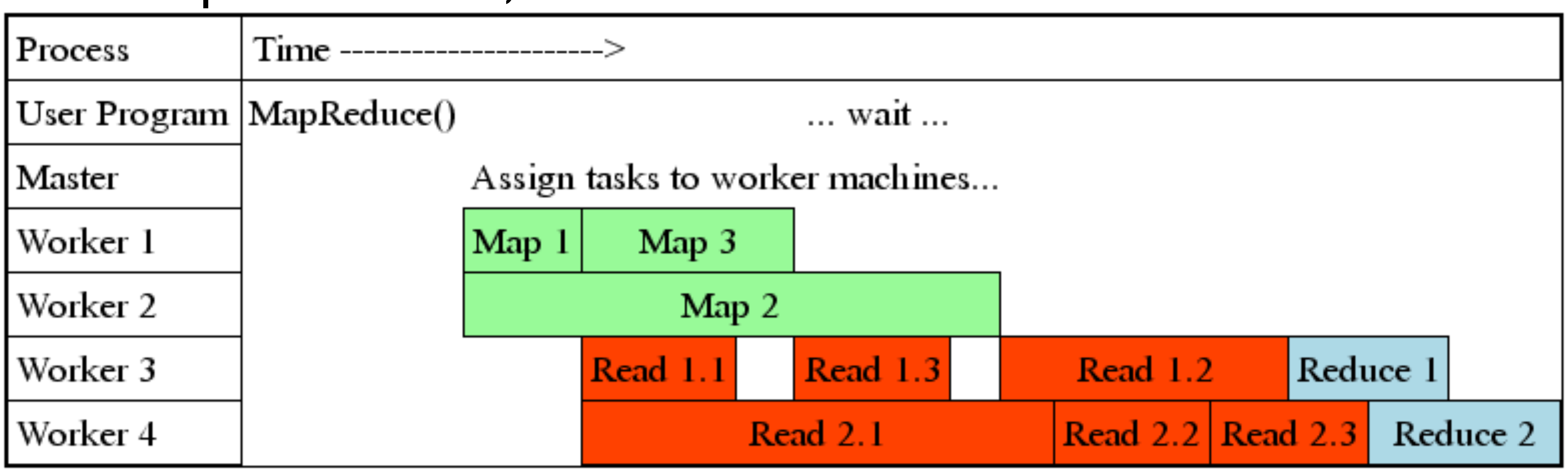
How many Map and Reduce jobs?

- M map tasks, R reduce tasks, W workers
- Rules of thumb:
  - Make M much larger than the number of workers
    - One DFS chunk per map is common
    - Improves dynamic load balancing and speeds up recovery from worker failures
  - Make R small multiple of W
    - Final output is spread across R files
- Common numbers at Google: M=200,000, R=5,000 using 2,000 worker machines.

Task Granularity & Pipelining

- Fine granularity tasks: map tasks >> machines
  - Minimizes time for fault recovery
  - Can do pipeline shuffling with map execution
  - Better dynamic load balancing

Example: M=3, R=2



adapted from J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Fault tolerance: “Stragglers”

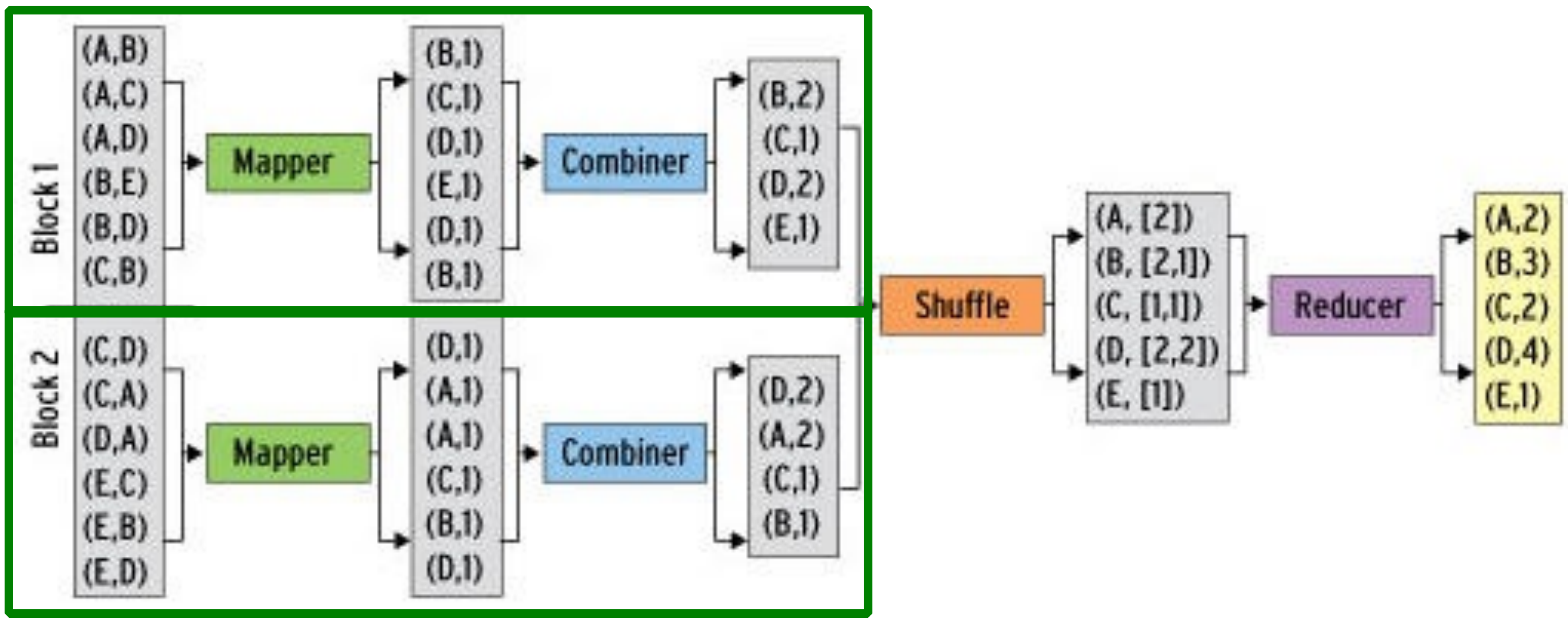
- Problem: Slow workers significantly lengthen the job completion time:
- Causes for slowness:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- Solution
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first “wins” (idempotence!)
- Effect
  - Dramatically shortens job completion time

Issues

- Synchronization barrier
  - Reduce function cannot be applied until *all* map tasks have finished (why?)
  - Other systems allow asynchronous computation
- Data skew
  - When some keys appear many many times (word count: “the”)
  - Load unevenly distributed across reduce workers

Refinement: Combiners

- Combiner: combines the values of all keys of a single mapper (single machine). Combiner often same as reducer function. Back to our word counting example:



- Much less data needs to be copied and shuffled!
- Works when reduce function is *associative* and *commutative*
- Improves load balancing (somewhat) for reduce workers

adapted from J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

Data skew

Instructions: ~1 minute to think/answer on your own; then discuss with neighbors; then I will call on one of you

Suppose we execute word count program on large collection of documents (WWW) with M map tasks and R reduce tasks. (Recall how data is sent from map tasks to reduce tasks.)

1. Suppose no combiner used and R=10,000. Do you expect significant skew?

2. Suppose no combiner used and R=10. Do you expect significant skew?

3. Suppose we use combiner and R=10,000. Do you expect significant skew?

Dealing with Data skew

- Combiners
    - Map worker combines all values for given key.
- Hashing
    - Recall that map worker hashes intermediate results
    - Reduce worker takes one hash bucket (contains many keys)
    - While *key distribution* may be skewed, *bucket size distribution* may be closer to uniform
- Set R larger than W (# workers)
    - Avg. tasks per worker: R/W
    - Worker with skew may do 1 task, others may do > R/W tasks

Resources

- Hadoop Wiki
  - Introduction
    - http://wiki.apache.org/lucene-hadoop/
  - Getting Started
    - http://wiki.apache.org/lucene-hadoop/GettingStartedWithHadoop
  - Map/Reduce Overview
    - http://wiki.apache.org/lucene-hadoop/HadoopMapReduce
    - http://wiki.apache.org/lucene-hadoop/HadoopMapRedClasses
  - Eclipse Environment
    - http://wiki.apache.org/lucene-hadoop/EclipseEnvironment
- Javadoc
  - http://lucene.apache.org/hadoop/docs/api/

jsmapreduce.com