Database Applications (15-415)

Hadoop
Lecture 24, April 23, 2014

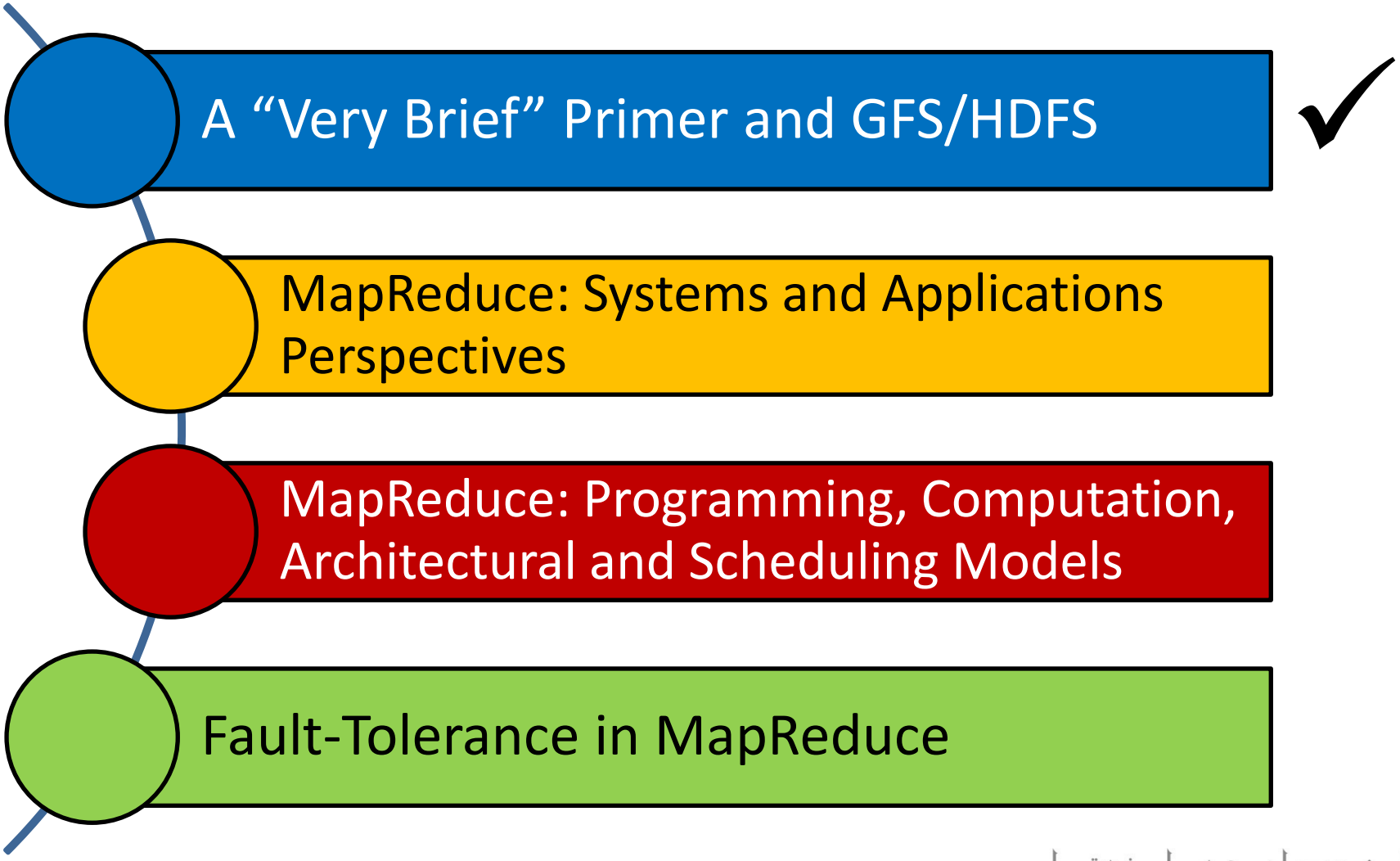
Mohammad Hammoud

جامعة كارنيجي ميلون في قطر
Carnegie Mellon University Qatar

Today…

- Last Session:
  - NoSQL databases
- Today's Session:
  - Hadoop = HDFS + MapReduce
- Announcements:
  - Final Exam is on Sunday April 27th, at 9:00AM in room 2051 (*all materials are included- open book, open notes*)
  - We will hold a "review session" (for the final exam) tomorrow during the recitation
  - PS4 grades are out
  - PS5 (the "last" assignment) is due tomorrow, by midnight

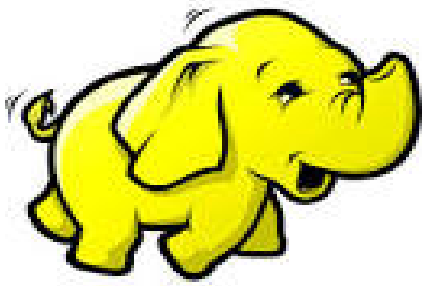
Outline



Hadoop MapReduce

- MapReduce is one of the most successful realizations of large-scale “data-parallel” distributed analytics engines

- Hadoop is an open source implementation of MapReduce



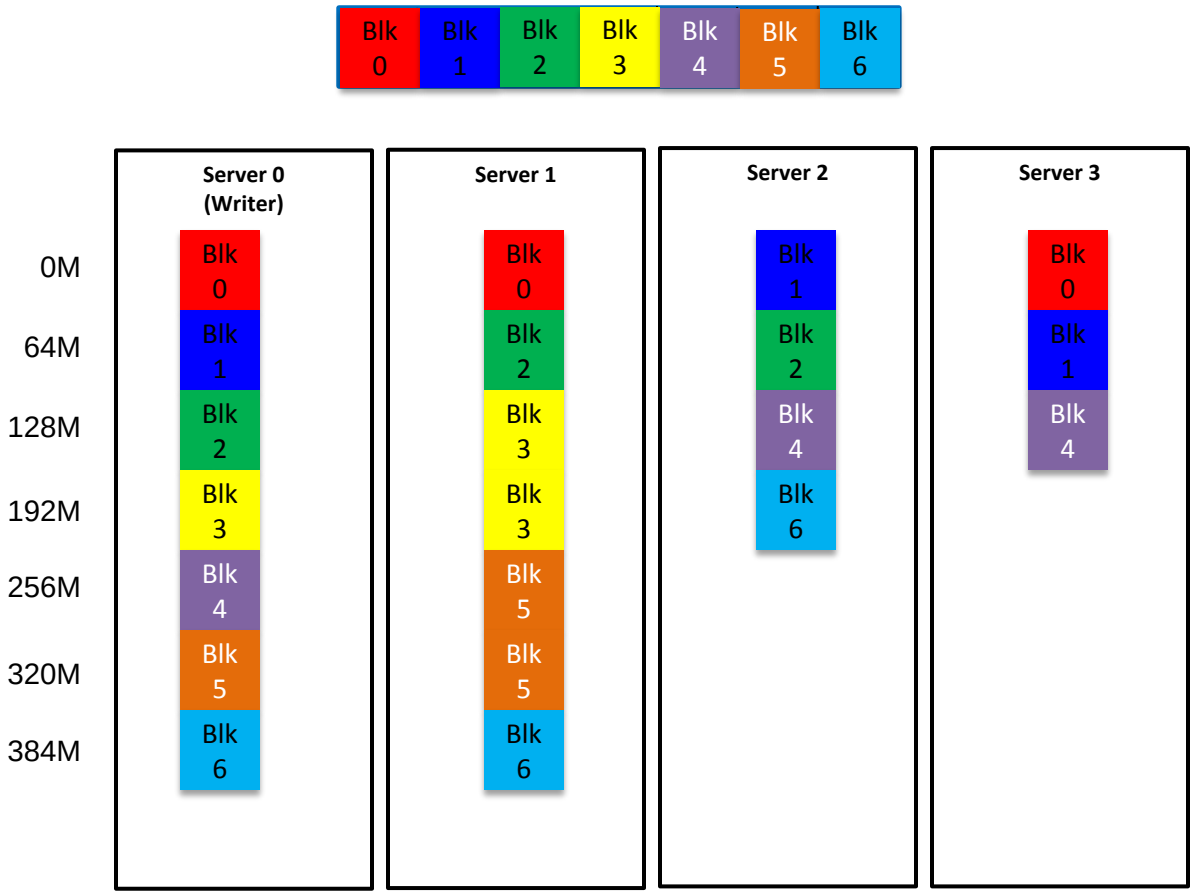
- Hadoop MapReduce uses Hadoop Distributed File System (HDFS) as a distributed storage layer

- HDFS is an open source implementation of GFS

GFS Data Distribution Policy

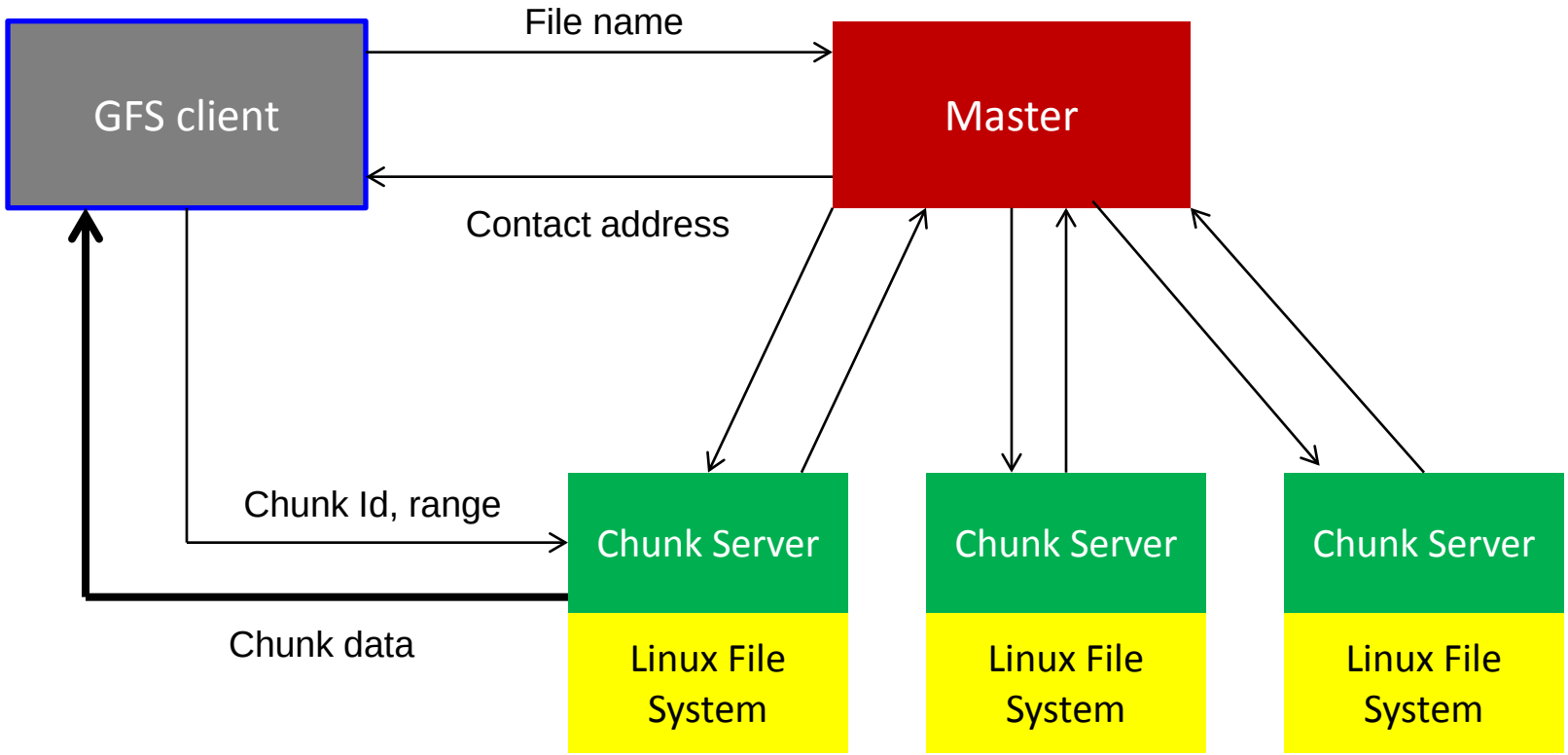
- The Google File System (GFS) is a scalable DFS for data-intensive applications
- GFS divides large files into multiple pieces called chunks or blocks (by default 64MB) and stores them on different data servers
  - This design is referred to as block-based design
- Each GFS chunk has a unique 64-bit identifier and is stored as a file in the lower-layer local file system on the data server
- GFS distributes chunks across cluster data servers using a random distribution policy

GFS Random Distribution Policy

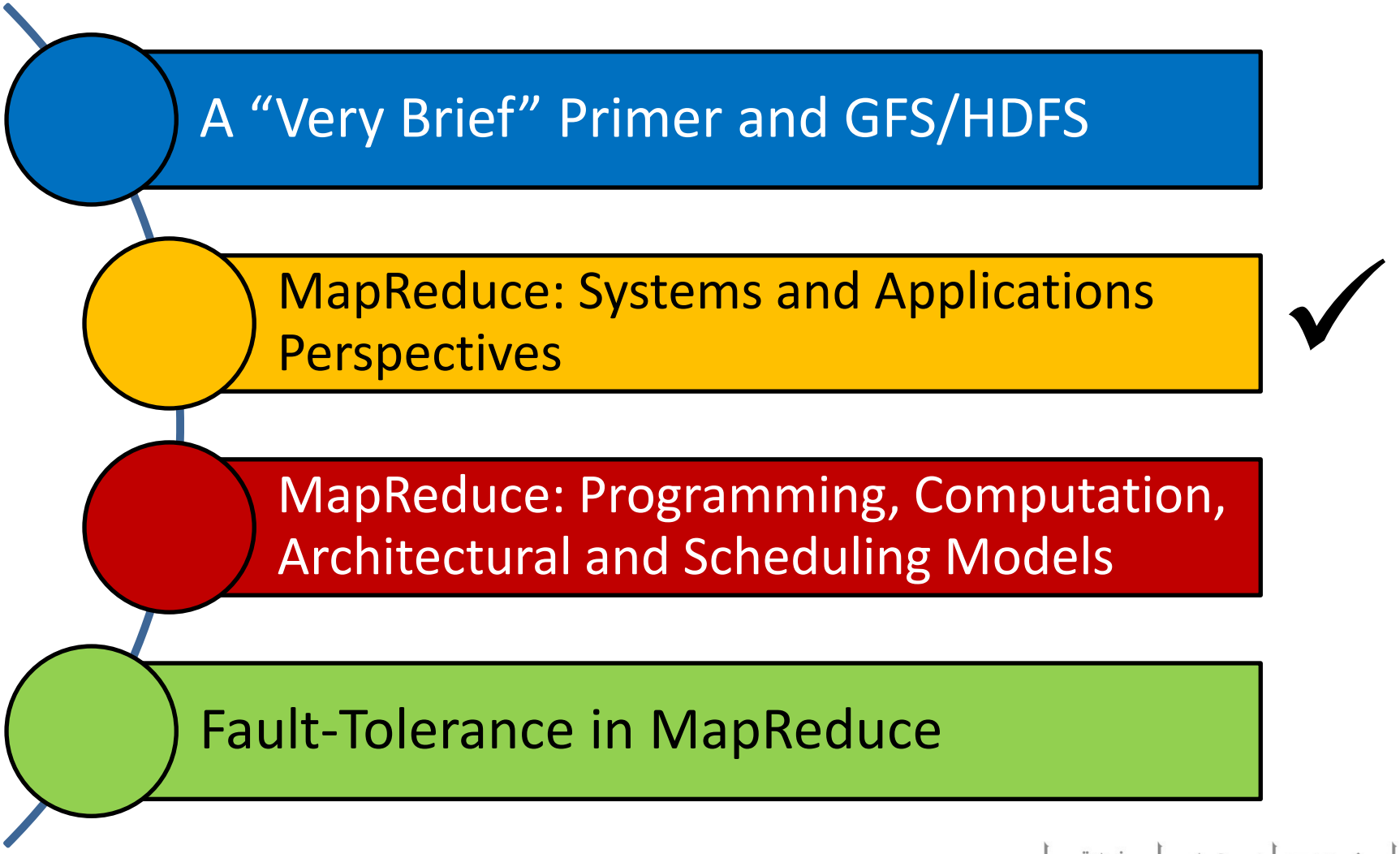


GFS Architecture

- GFS adopts a master-slave architecture



Outline



The Problem Scope

- Hadoop MapReduce is used for powerful and efficient analytics over *Big Data*

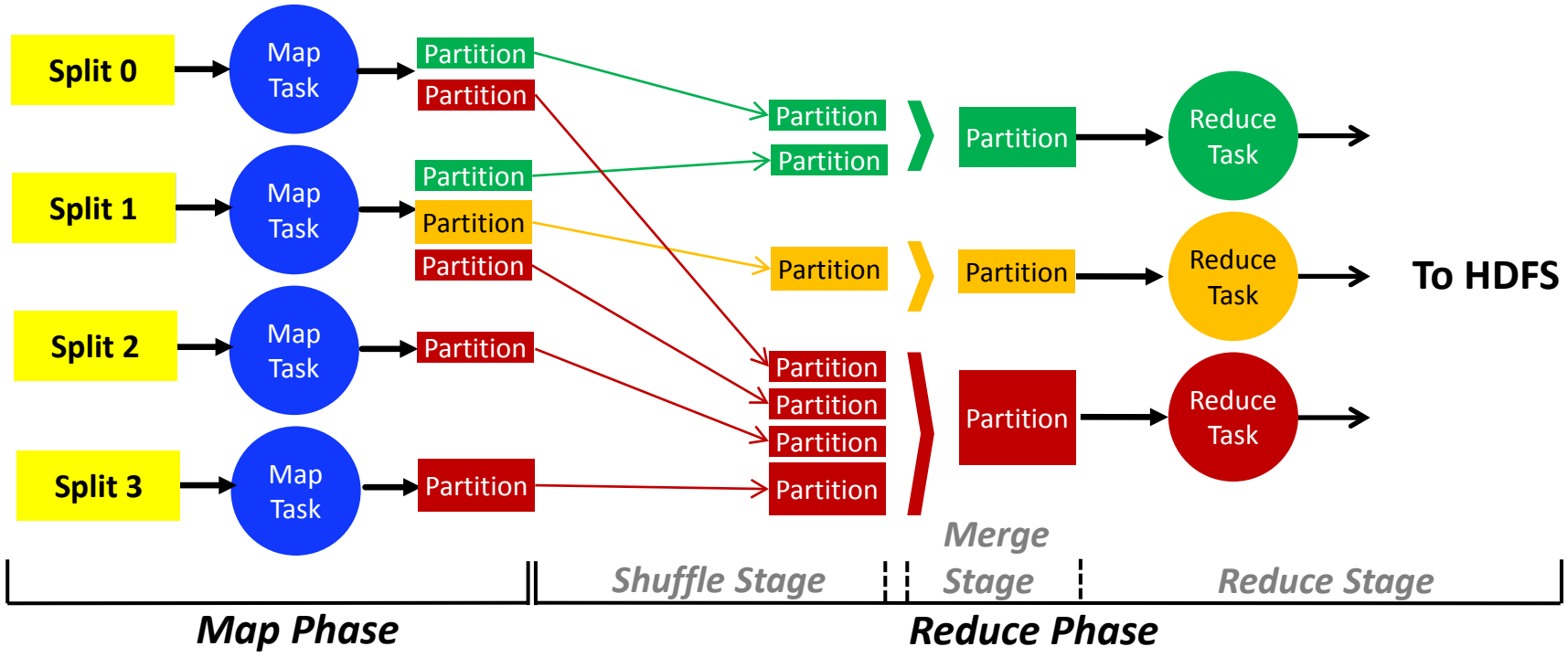
- The power of MapReduce lies in its ability to *scale* to 100s and even 1000s of machines

- What amount of work can MapReduce handle?
  - Big Data in the order of 100s of GBs, TBs or PBs

  - It is unlikely that datasets of such sizes can fit on a single machine
    - Hence, a storage layer like HDFS is required!

Hadoop MapReduce: A System's View

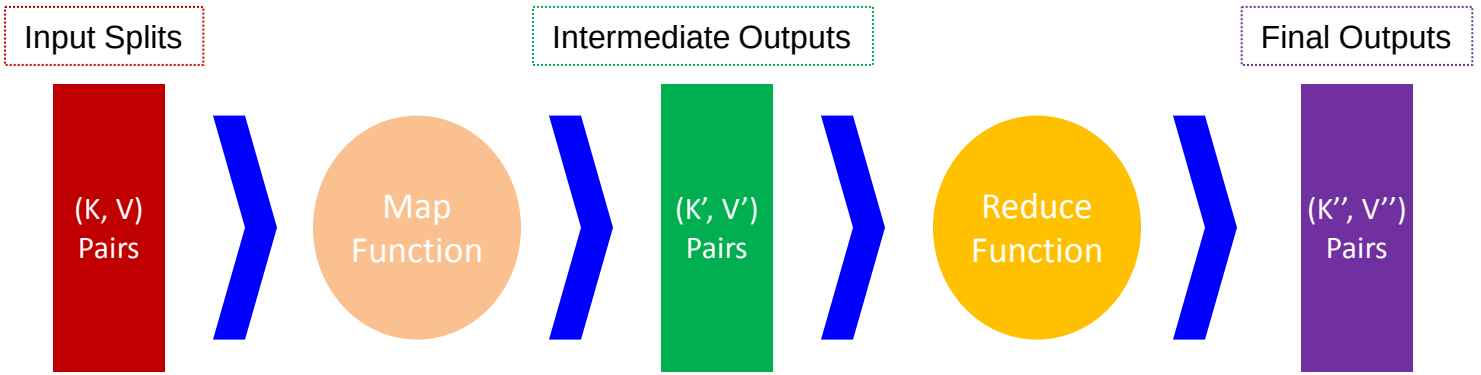
- Hadoop MapReduce incorporates two phases, Map and Reduce phases, which encompass multiple Map and Reduce tasks



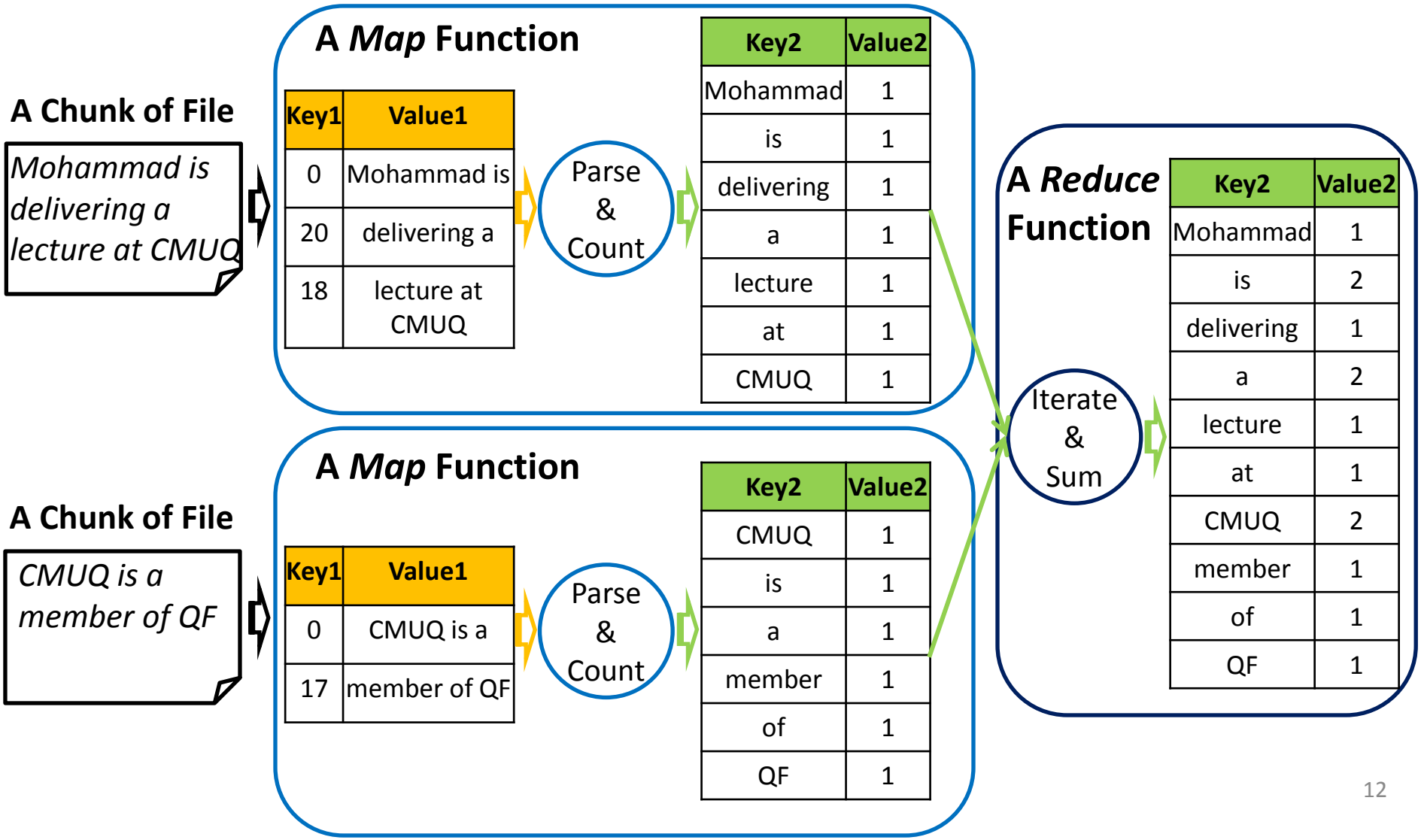
Data Structure: Keys and Values

- The MapReduce programmer has to specify only two "sequential" functions, the Map and the Reduce functions
  - These functions will be translated "automatically" into *multiple* Map and Reduce tasks

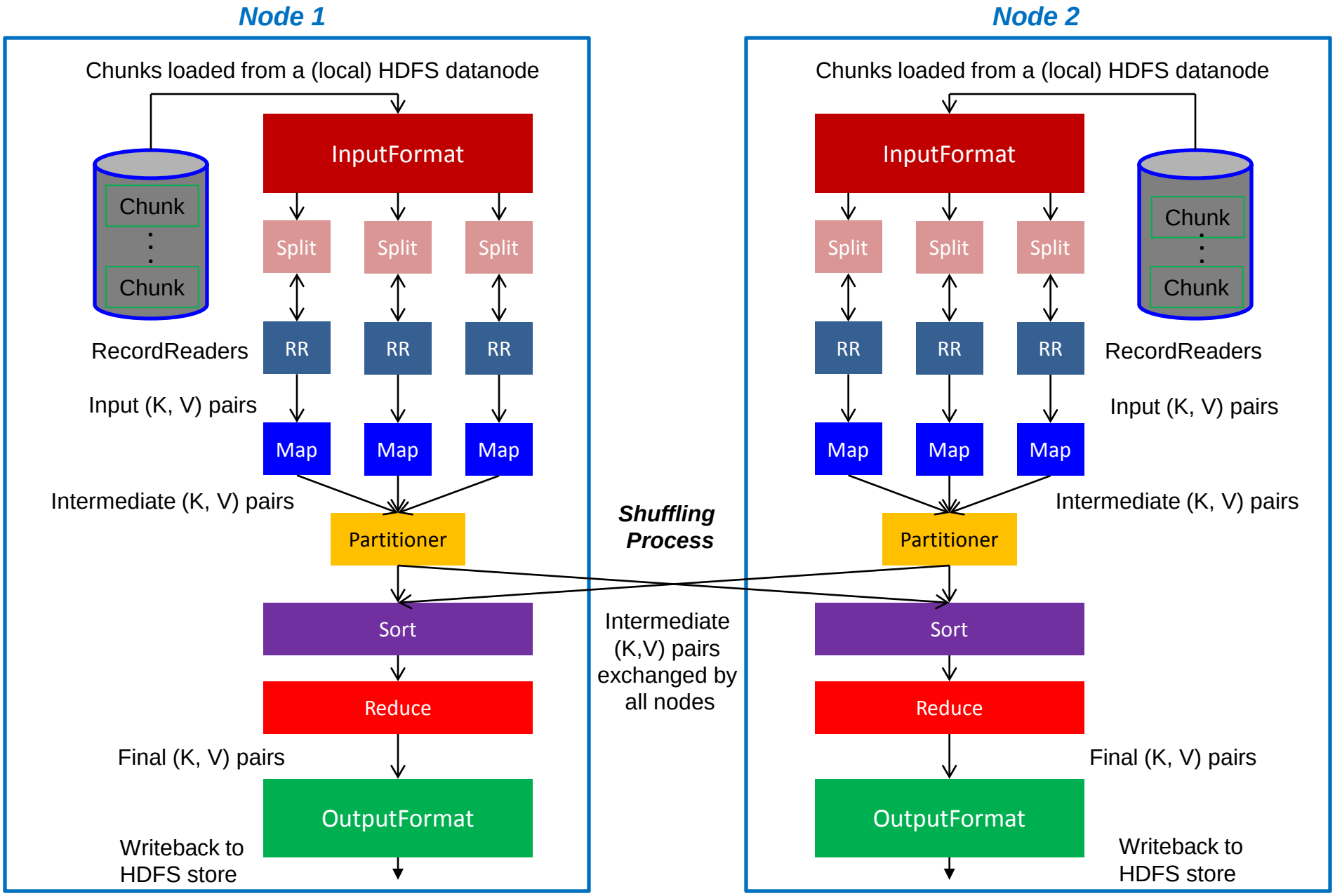
- In MapReduce, data elements are always structured as key-value (i.e., (K, V)) pairs
  - In particular, the Map and Reduce functions receive and *emit* (K, V) pairs



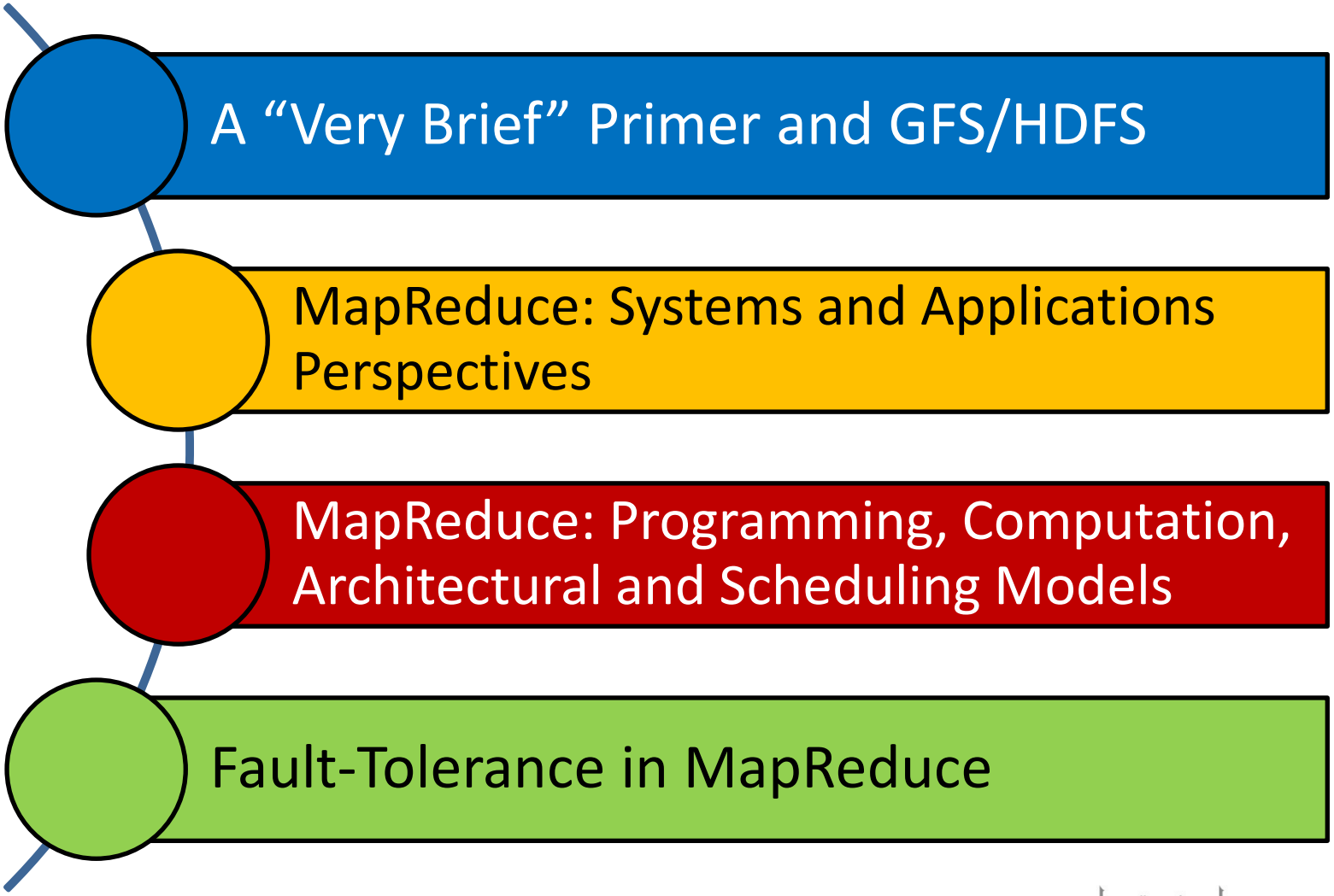
WordCount: An Application View



Hadoop MapReduce: A Closer Look



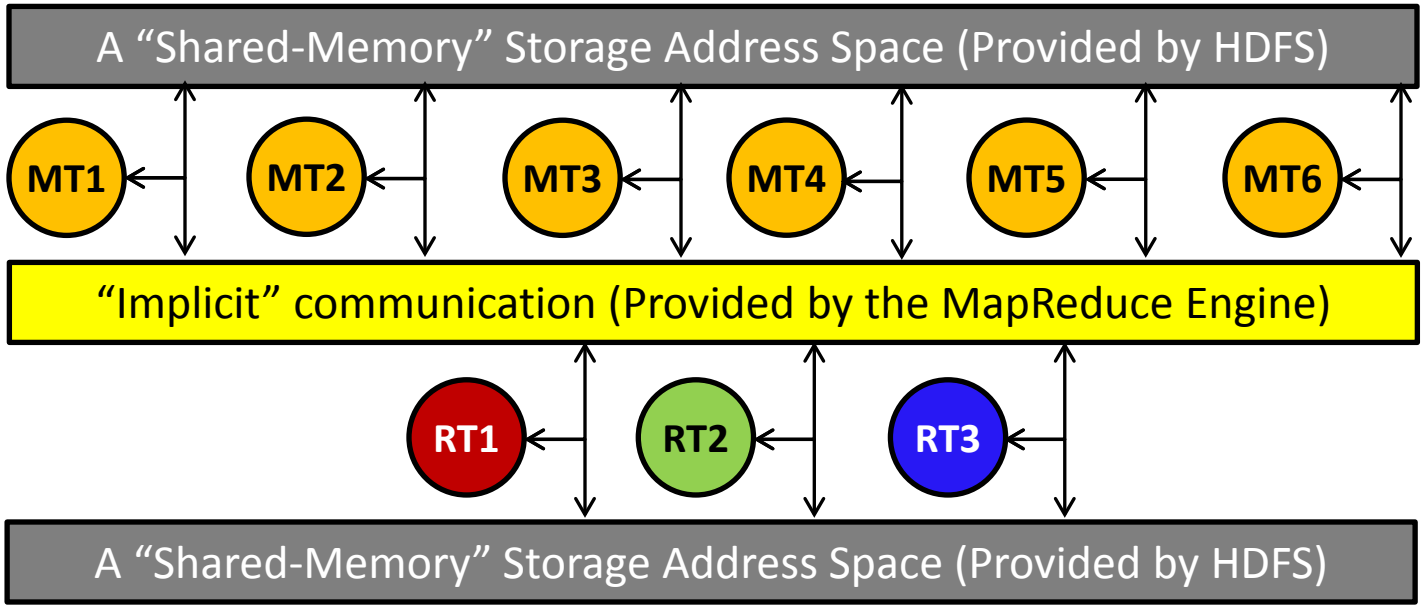
Outline



The Programming Model

- Hadoop MapReduce employs a shared-memory programming model

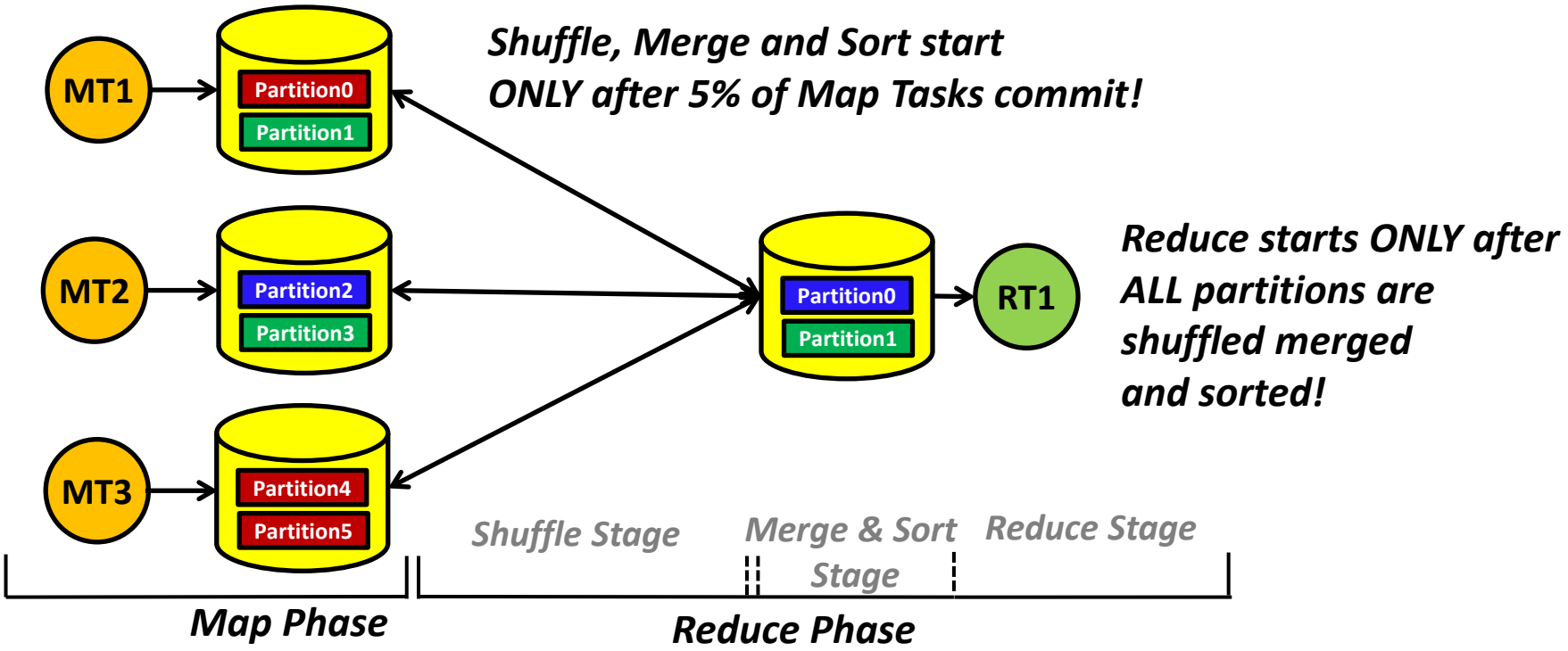
- This entails two main issues:
  - Developers need not "explicitly" encode functions that send/receive messages within their MapReduce programs
  - HDFS provides a shared abstraction to all tasks



The Computation Model

- Hadoop MapReduce adopts a synchronous computation model

- A distributed program is said to be synchronous if and only if the tasks operate in a *lock-step mode*

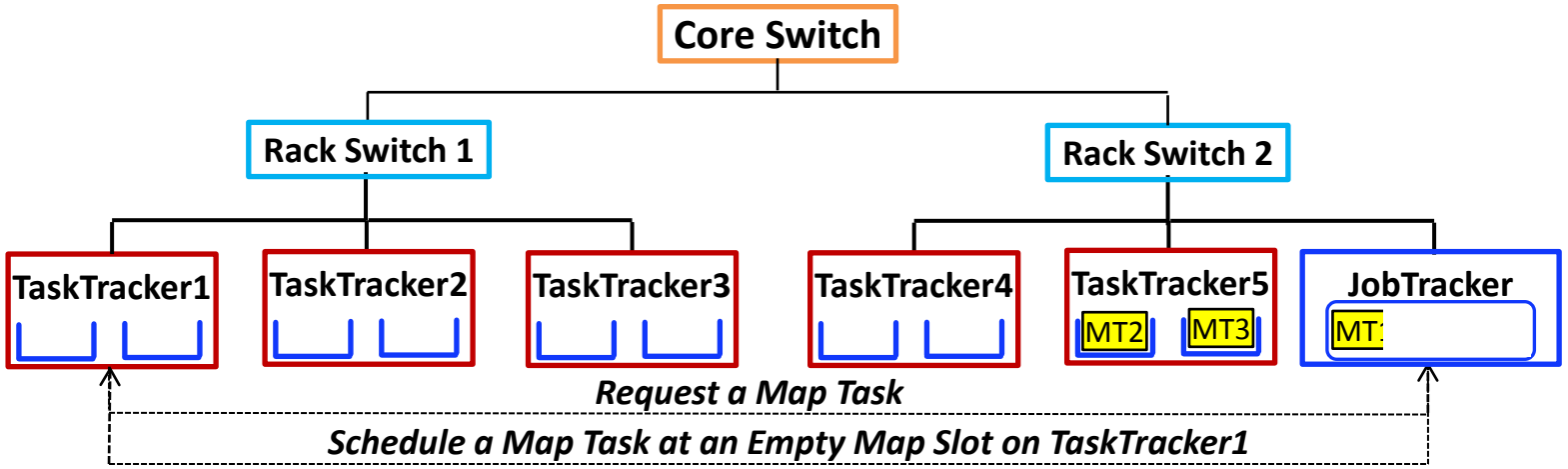


The Architectural and Scheduling Models

- Hadoop MapReduce employs a master-slave architecture

The Architectural and Scheduling Models

- Hadoop MapReduce employs a master-slave architecture



- A pull-based "task" scheduling strategy is used, whereby:
  - Map tasks are scheduled nearby HDFS blocks
  - Reduce tasks are scheduled *anywhere*

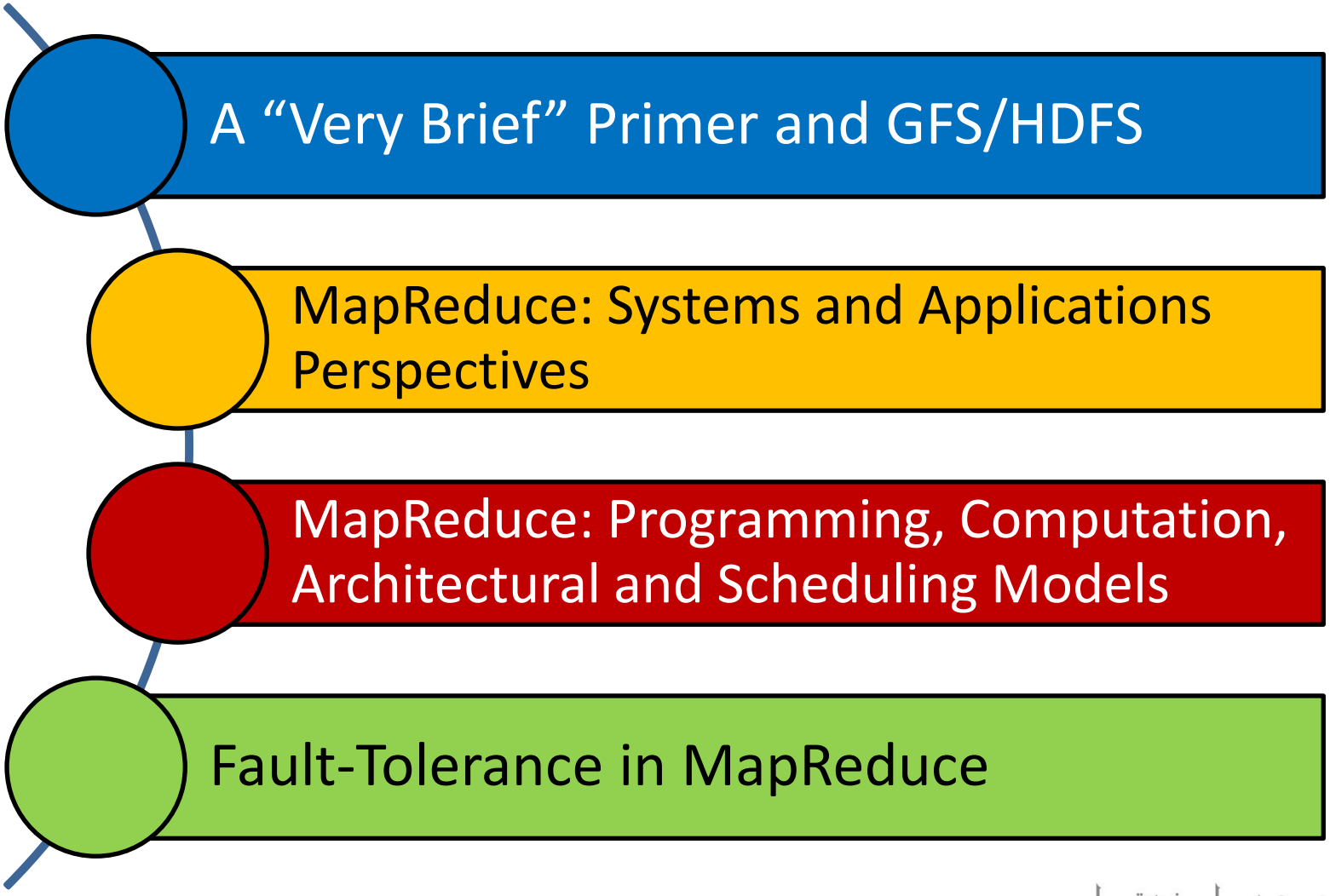
Job Scheduling in MapReduce

- An application is represented by one or many jobs
- A job consists of one or many Map and Reduce tasks
- Hadoop MapReduce comes with various choices of job schedulers:
  - FIFO Scheduler: schedules jobs in order of submission
  - Fair Scheduler: aims at giving every user a “fair” share of the cluster capacity over time
  - Capacity Scheduler: Similar to Fair Scheduler but does not apply job preemption

Summary

Aspect	Hadoop MapReduce
Parallelism Model	Data-Parallel
Programming Model	Shared-Memory
Computation Model	Synchronous
Architectural Model	Master-Slave
Scheduling Model	Pull-Based
Application Suitability	Loosely-Connected/Embarrassingly-Parallel Applications

Outline



Fault Tolerance in Hadoop: Node Failures

- MapReduce can guide jobs toward a successful completion even when jobs are run on large clusters (*where probability of failures increases*)

- Hadoop MapReduce achieves fault-tolerance through restarting tasks

- If a TT fails to communicate with JT for a period of time (by default, 1 minute), JT will assume that TT in question has crashed
  - If the job is still in the Map phase, JT asks another TT to re-execute *all Map tasks that previously ran at the failed TT*

  - If the job is in the Reduce phase, JT asks another TT to re-execute *all Reduce tasks that were in-progress on the failed TT*

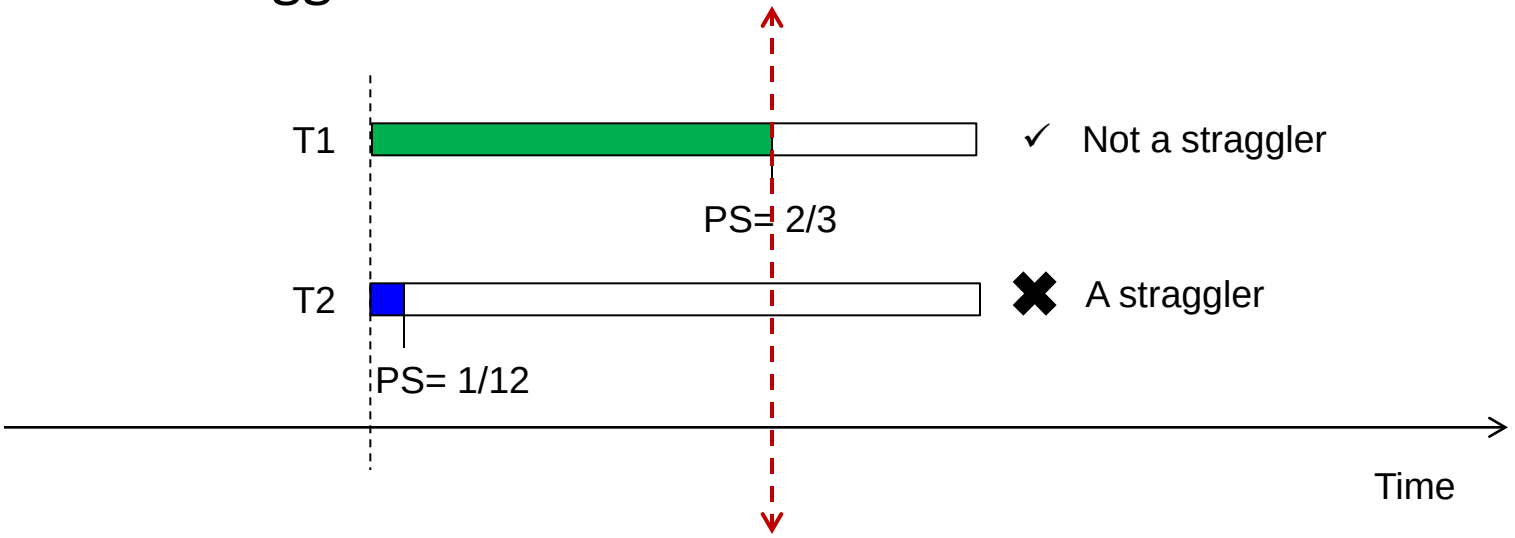
Fault Tolerance in Hadoop: Speculative Execution

- A MapReduce job is dominated by the slowest task
- MapReduce attempts to locate slow tasks (or *stragglers*) and run *replicated* (or *speculative*) tasks that will optimistically commit before the stragglers
- In general, this strategy is known as *task resiliency* or *task replication* (as opposed to data replication), but in Hadoop it is referred to as speculative execution
- Only one copy of a straggler is allowed to be replicated
- Whichever copy (among the two copies) of a task commits first, it becomes the definitive copy, and the other one is killed by JT

But, How to Locate Stragglers?

- Hadoop monitors each task progress using a *progress score* between 0 and 1

- If a task's progress score ***is less than*** (average – 0.2), and the task has run for at least 1 minute, it is marked as a straggler



Next Class

A Review Session