Introduction to MapReduce

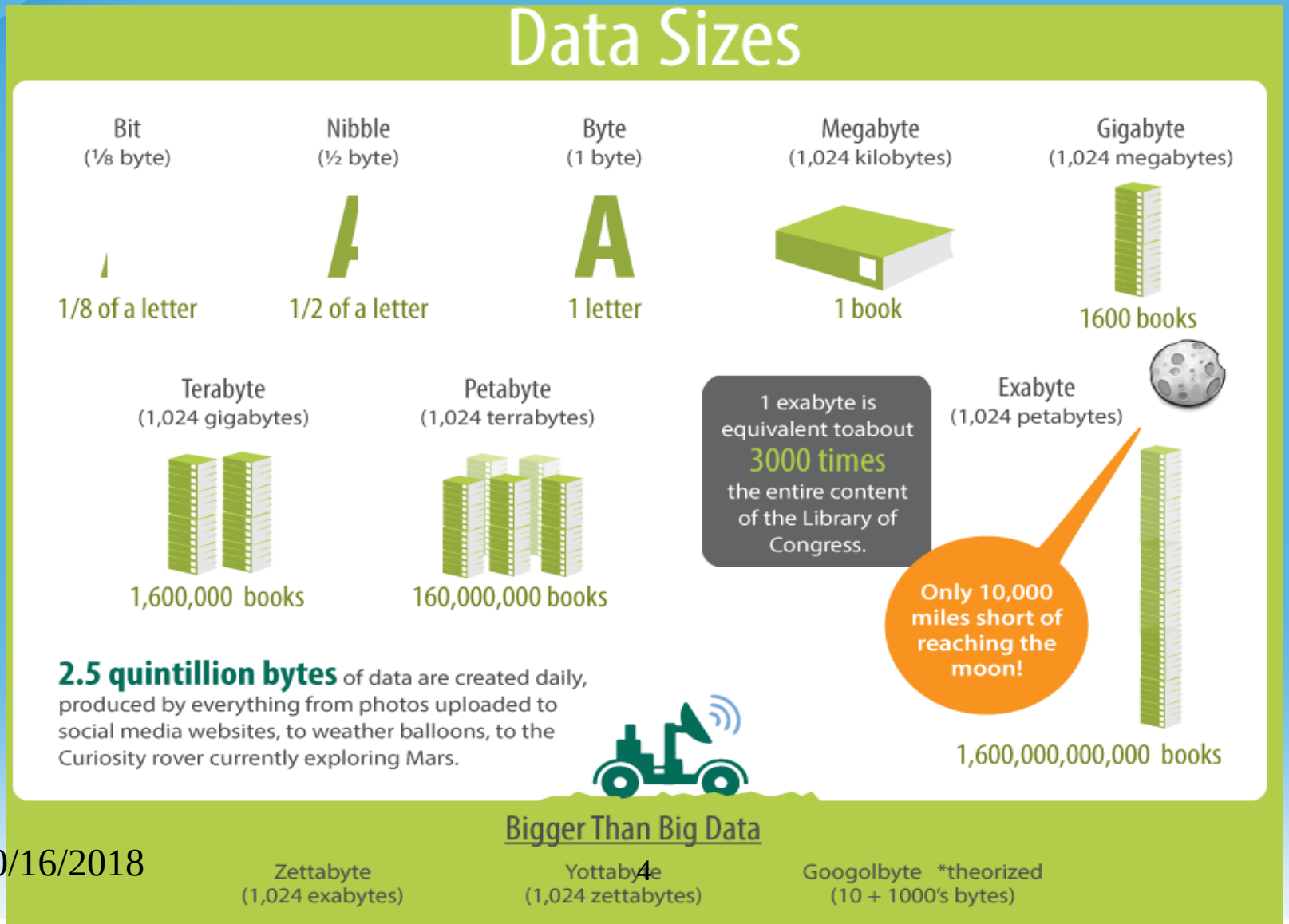
Outline

- We will cover:
  - Basics
  - Data Flow in MapReduce
  - Scheduling in MapReduce
  - Next Generation MapReduce

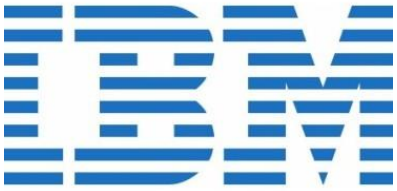
Big Data

- Big data includes data sets with sizes beyond the ability of commonly used software to process within a tolerable elapsed time.

Big Data


Data Sizes
Bit
(⅛ byte)
1/8 of a letter
Nibble
(½ byte)
1/2 of a letter
Byte
(1 byte)
1 letter
Megabyte
(1,024 kilobytes)
1 book
Gigabyte
(1,024 megabytes)
1600 books
Terabyte
(1,024 gigabytes)
1,600,000 books
Petabyte
(1,024 terrabytes)
160,000,000 books
1 exabyte is equivalent toabout 3000 times the entire content of the Library of Congress.
Exabyte
(1,024 petabytes)
Only 10,000 miles short of reaching the moon!
1,600,000,000,000 books
2.5 quintillion bytes of data are created daily, produced by everything from photos uploaded to social media websites, to weather balloons, to the Curiosity rover currently exploring Mars.
Bigger Than Big Data
Zettabyte
(1,024 exabytes)
Yottabyte
(1,024 zettabytes)
Googolbyte *theorized
(10 + 1000's bytes)

Why BigData?



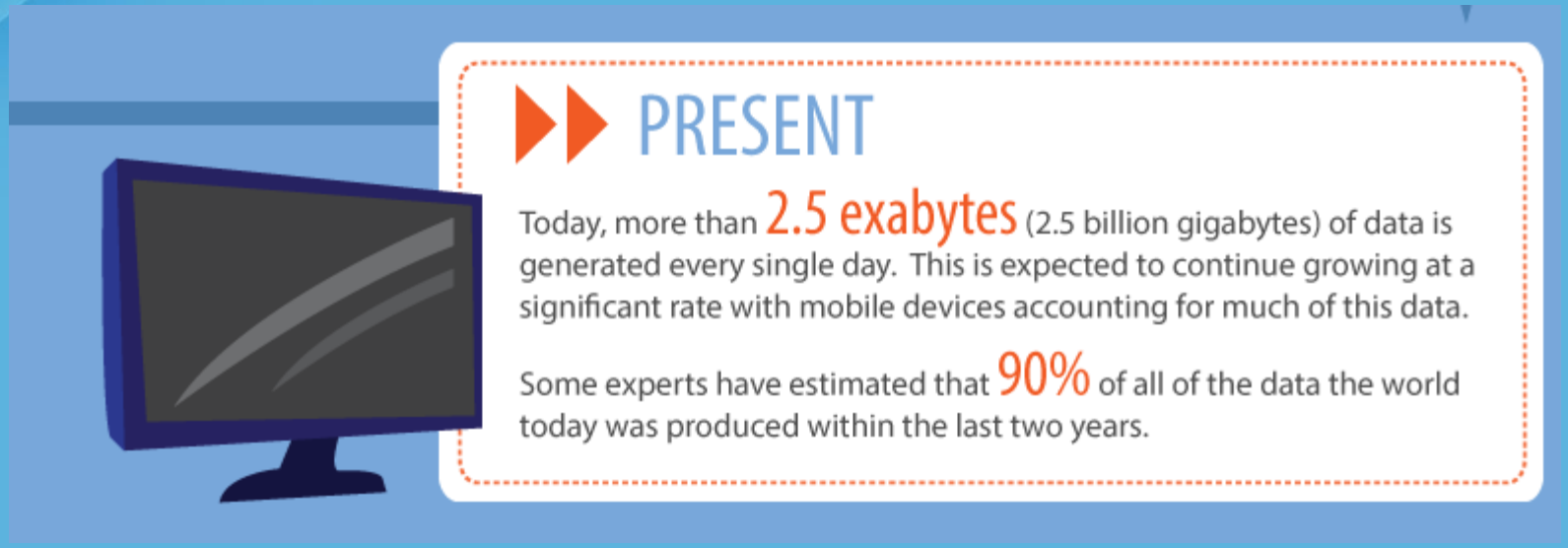
2.7 Zetabytes of data exist in the digital universe today

100 terabytes of data uploaded daily to Facebook

Walmart handles more than 1 million customer transactions every hour, more than 2.5 petabytes of data

YouTube users upload 48 hours of new video every minute of the day

Why Big Data??


PRESENT
Today, more than 2.5 exabytes (2.5 billion gigabytes) of data is generated every single day. This is expected to continue growing at a significant rate with mobile devices accounting for much of this data.
Some experts have estimated that 90% of all of the data the world today was produced within the last two years.


THE FUTURE
NEXT EXIT
of Cloud Technologies
It is estimated that 40 zettabytes will be created by 2020.

Motivation

- Process lots of data
  - Google processed about 24 petabytes of data per day in 2009
- A single machine cannot serve all the data
  - You need a distributed system to store and process in parallel
- Parallel programming?
  - Threading is hard!
  - How do you facilitate communication between nodes?
  - How do you scale to more machines?
  - How do you handle machine failures?

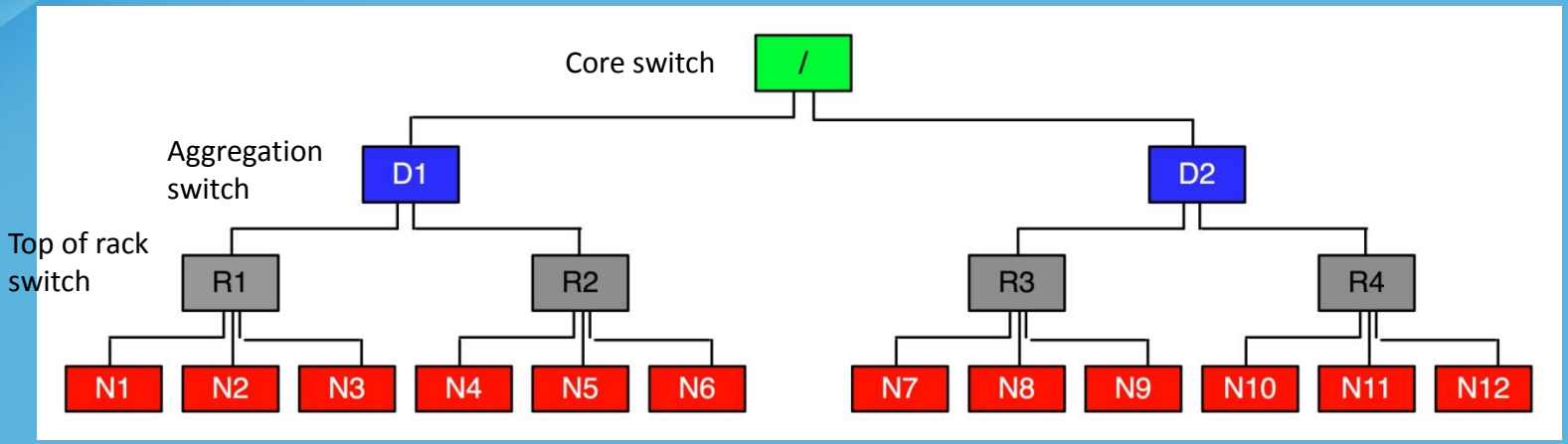
MapReduce

- *MapReduce* is a programming model for data processing
- The power of MapReduce lies in its ability to scale to 100s or 1000s of computers, each with several processor cores
- MapReduce divides the workload into multiple *independent tasks* and schedule them across cluster nodes
- A work performed by each task is done *in isolation* from one another

Key Features

- MapReduce provides
  - Automatic parallelization, distribution
  - I/O scheduling
    - Load balancing
    - Network and data transfer optimization
  - Fault tolerance
    - Handling machine failures
- Need more power: Scale out, not up!
    - A large number of commodity servers as opposed to some high end specialized servers

Network Topology in MapReduce

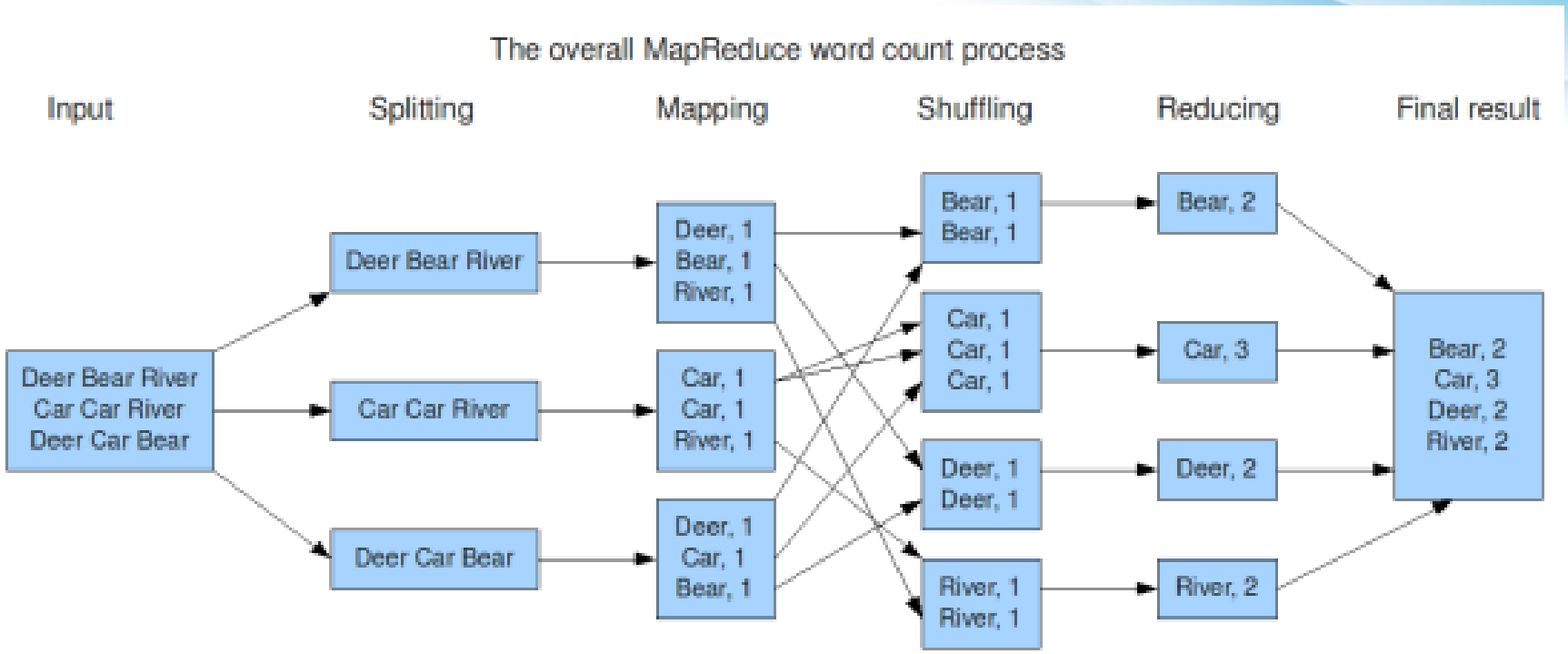


- MapReduce assumes a tree style network topology
- Nodes are spread over different racks embraced in one or many data centers
- A salient point is that the bandwidth between two nodes is dependent on their relative locations in the network topology
- For example, nodes that are on the same rack will have higher bandwidth between them as opposed to nodes that are off-rack

Typical Cluster

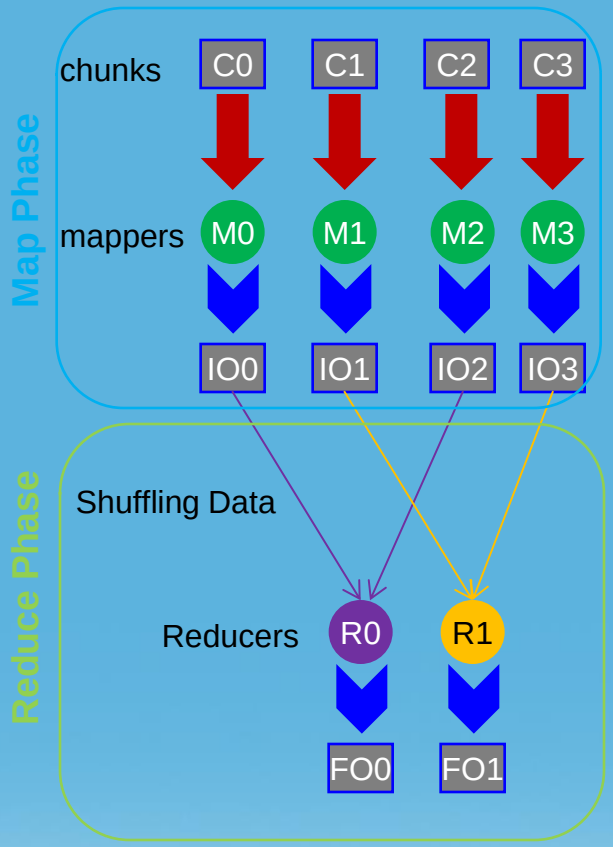
Image from http://wiki.apache.org/hadoop-data/attachments/HadoopPresentations/attachments/aw-apachecon-eu-2009.pdf

MapReduce Application Example: WordCount

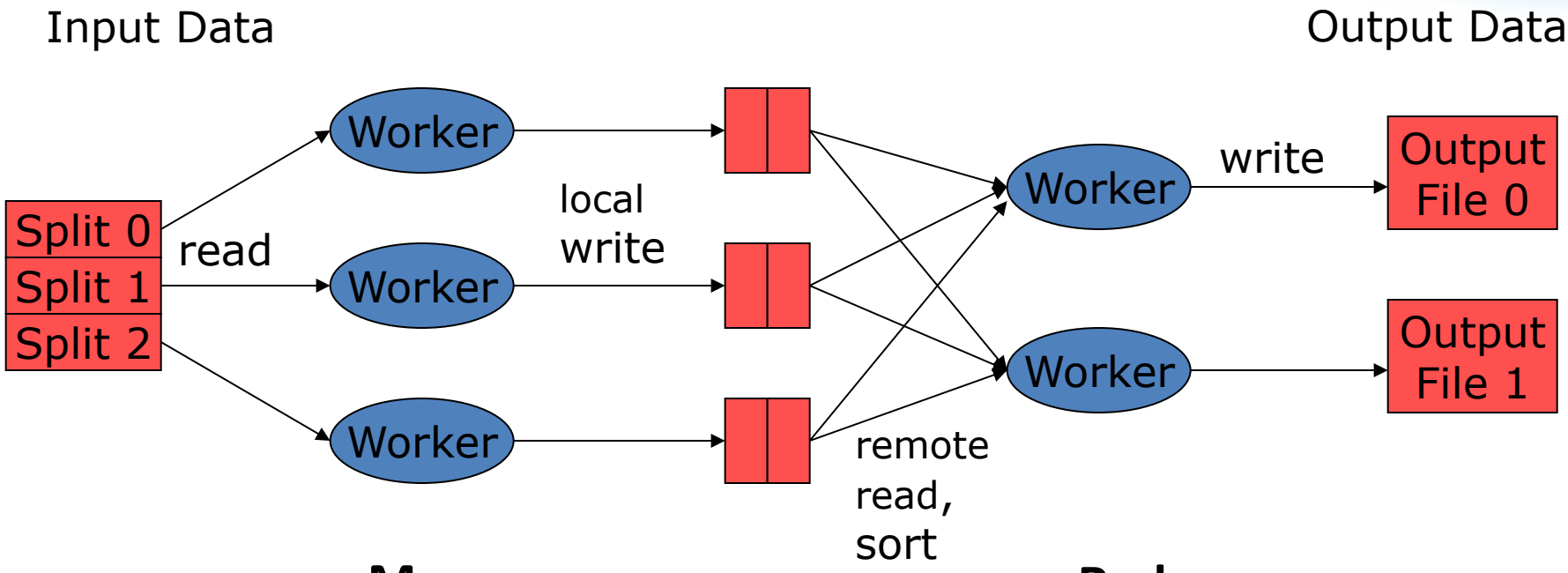


MapReduce: A Bird's-Eye View

- In MapReduce, chunks are processed in isolation by tasks called *Mappers*
- The outputs from the mappers are denoted as intermediate outputs (IOs) and are brought into a second set of tasks called *Reducers*
- The process of bringing together IOs into a set of Reducers is known as *shuffling process*
- The Reducers produce the final outputs (FOs)
- Overall, MapReduce breaks the data flow into two phases, *map phase* and *reduce phase*



MapReduce Workflow



Map

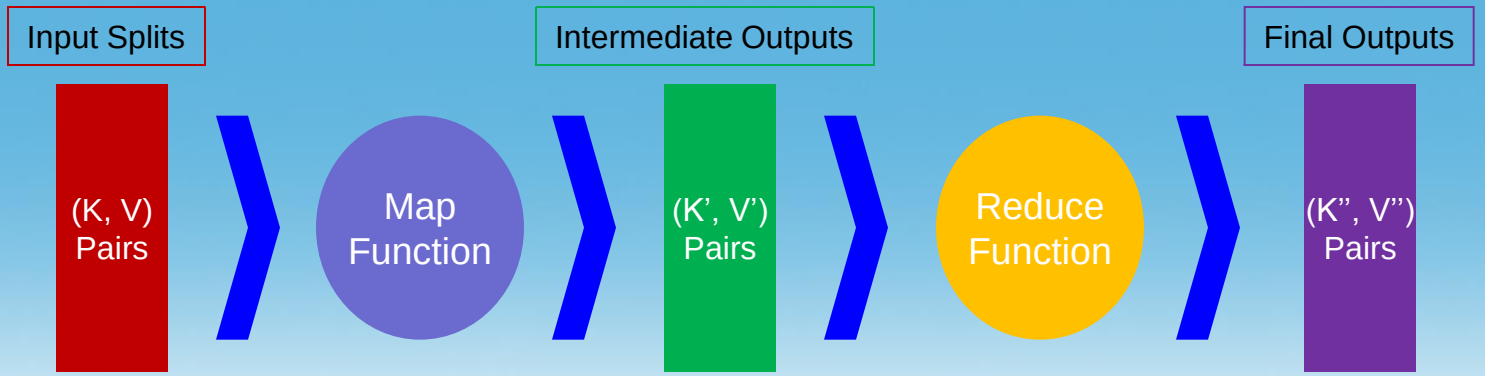
extract something you care about from each record

Reduce

aggregate, summarize, filter, or transform

Keys and Values

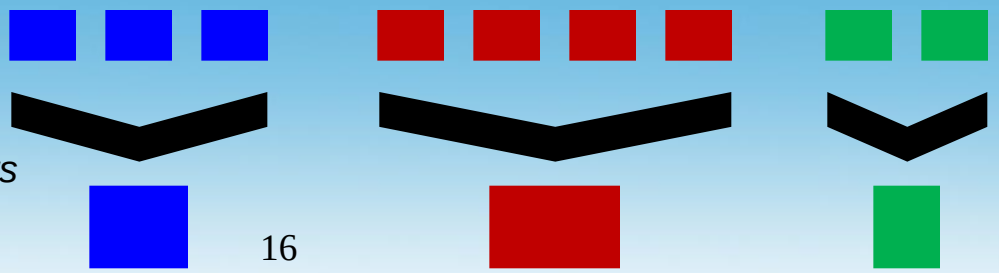
- The programmer in MapReduce has to specify two functions, the *map function* and the *reduce function* that implement the Mapper and the Reducer in a MapReduce program
- In MapReduce, data elements are always structured as key-value (i.e., (K, V)) pairs
- The map and reduce functions receive and *emit* (K, V) pairs



Partitions

- In MapReduce, intermediate output values are not usually reduced together
- *All values with the same key are presented to a single Reducer together*
- More specifically, a different subset of intermediate key space is assigned to each Reducer
- These subsets are known as *partitions*

Different colors represent different keys (potentially) from different Mappers



Partitions are the input to Reducers

Hadoop

- Since its debut on the computing stage, MapReduce has frequently been associated with *Hadoop*
- Hadoop is an open source implementation of MapReduce and is currently enjoying wide popularity
- Hadoop presents MapReduce as an analytics engine and under the hood uses a distributed storage layer referred to as Hadoop Distributed File System (*HDFS*)
- HDFS mimics Google File System (*GFS*)

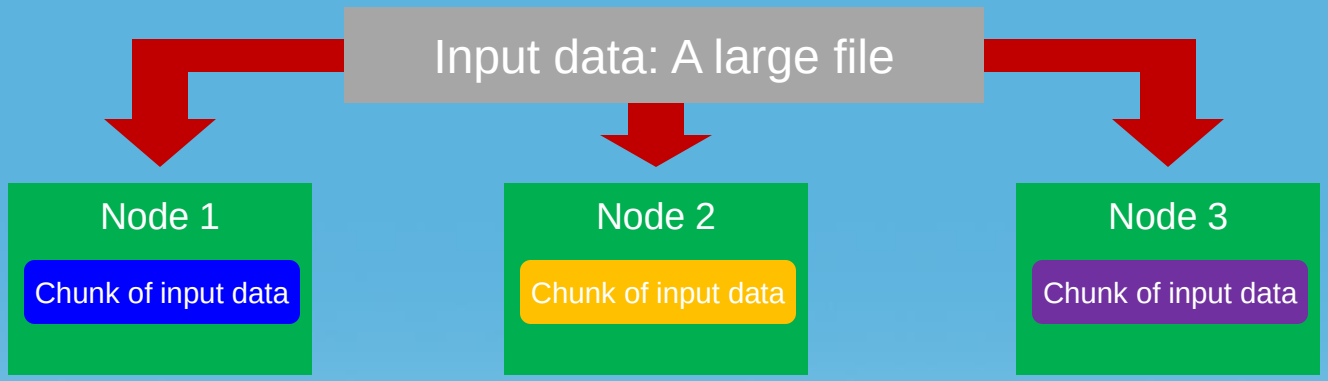
Hadoop Components

- **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

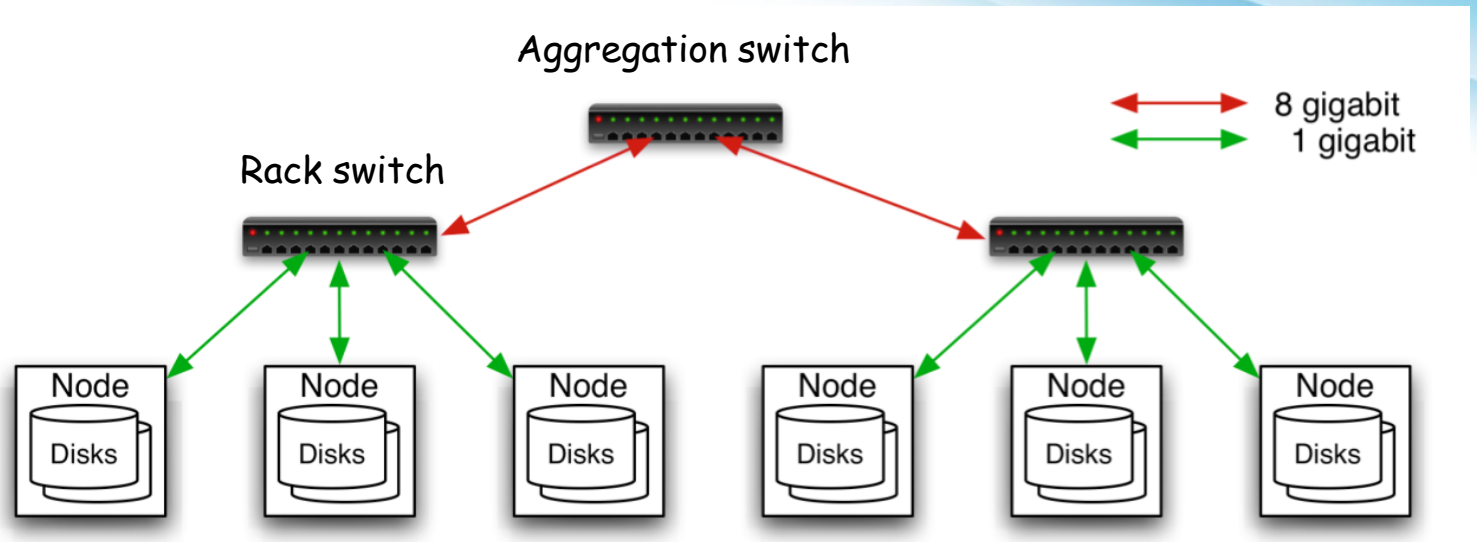
Data Distribution

- In a MapReduce cluster, data is distributed to all the nodes of the cluster as it is being loaded in
- An underlying distributed file systems (e.g., GFS) splits large data files into chunks which are managed by different nodes in the cluster



- Even though the file chunks are distributed across several machines, they form *a single namesapce*

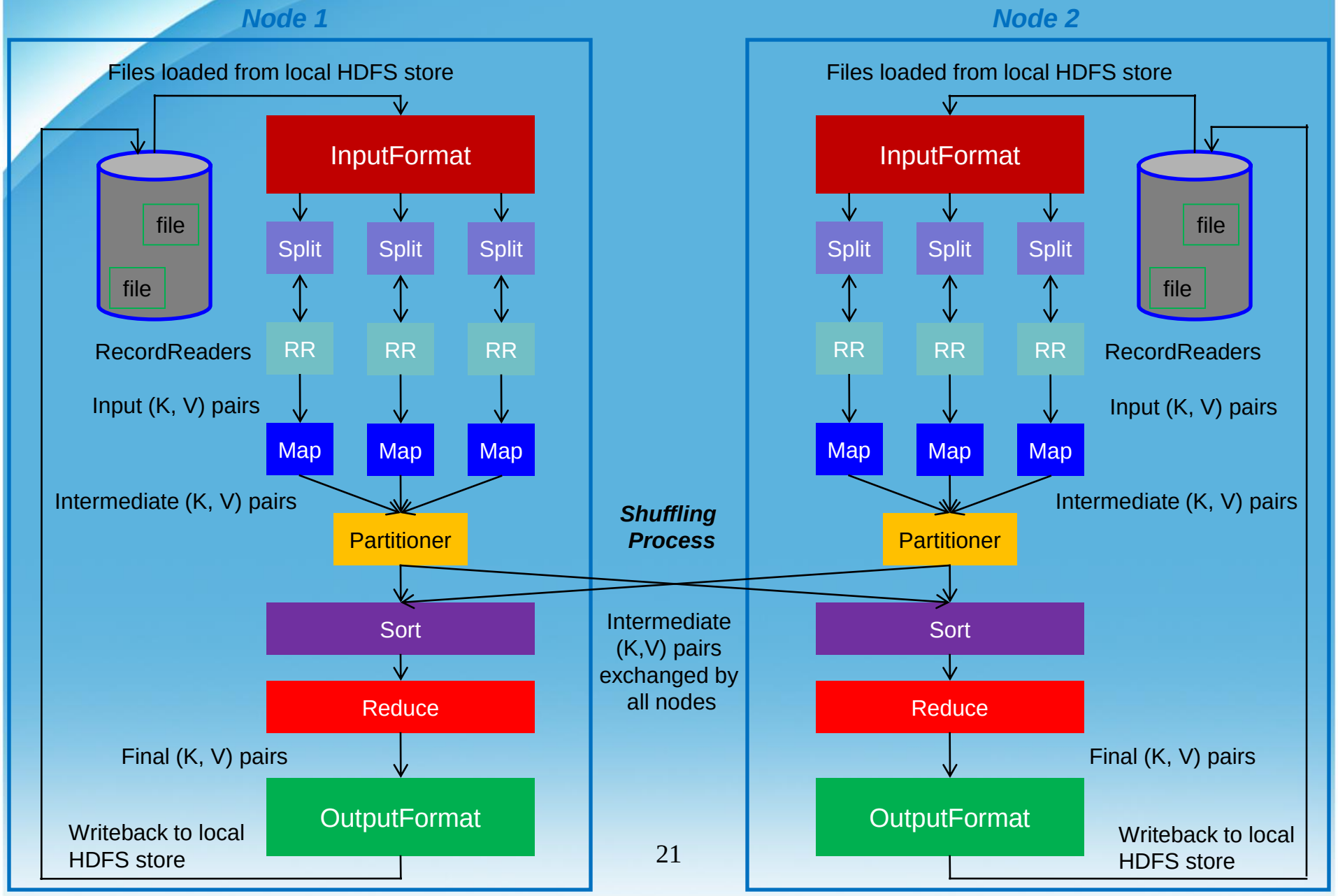
Typical Hadoop Cluster



- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth within rack, 8 Gbps out of rack
- Node specs (Yahoo terasort):
  8 x 2GHz cores, 8 GB RAM, 4 disks (= 4 TB)

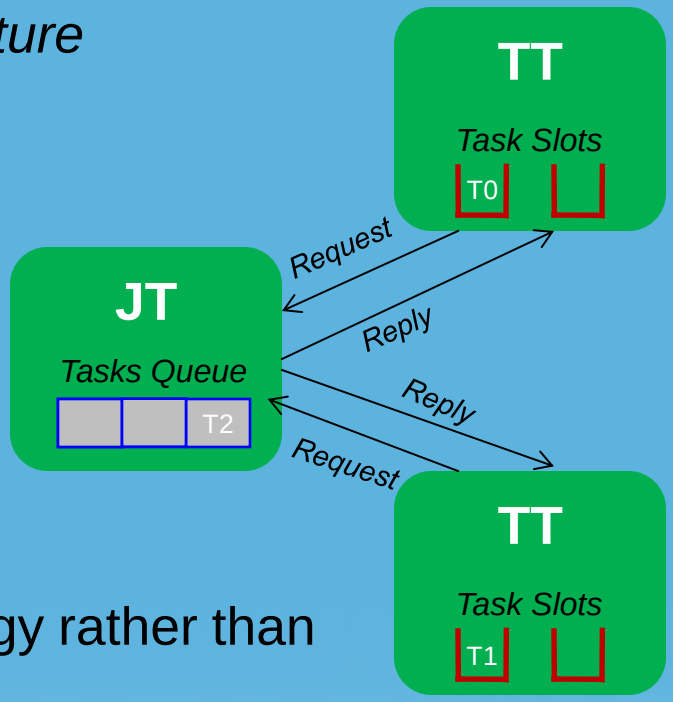
Image from http://wiki.apache.org/hadoop-data/attachments/HadoopPresentations/attachments/YahooHadoopIntro-apachecon-us-2008.pdf

Hadoop MapReduce: A Closer Look



Task Scheduling in MapReduce

- MapReduce adopts a *master-slave architecture*
- The master node in MapReduce is referred to as *Job Tracker* (JT)
- Each slave node in MapReduce is referred to as *Task Tracker* (TT)
- MapReduce adopts a *pull scheduling* strategy rather than a *push one*
  - I.e., JT does not push map and reduce tasks to TTs but rather TTs pull them by making pertaining requests



Map and Reduce Task Scheduling

- Every TT sends a *heartbeat message* periodically to JT encompassing a request for a map or a reduce task to run

I. Map Task Scheduling:

  - JT satisfies requests for map tasks via attempting to schedule mappers in the *vicinity* of their input splits (i.e., it considers locality)

II. Reduce Task Scheduling:

  - However, JT simply assigns the next yet-to-run reduce task to a requesting TT regardless of TT's network location and its implied effect on the reducer's shuffle time (i.e., it does not consider locality)

Job Scheduling in MapReduce

- In MapReduce, an application is represented as a *job*
- A job encompasses multiple map and reduce tasks
- MapReduce in Hadoop comes with a choice of schedulers:
  - The default is the *FIFO scheduler* which schedules jobs in order of submission
  - There is also a multi-user scheduler called the *Fair scheduler* which aims to give every user a fair share of the cluster capacity over time

Fault Tolerance in MapReduce

1. If a task crashes:
   - Retry on another node
     - OK for a map because it has no dependencies
     - OK for reduce because map outputs are on disk
   - If the same task fails repeatedly, fail the job or ignore that input block (user-controlled)

➢ Note: For these fault tolerance features to work, *your map and reduce tasks must be side-effect-free*

Fault Tolerance in MapReduce

2. If a node crashes:
   - Re-launch its current tasks on other nodes
   - Re-run any maps the node previously ran
     - Necessary because their output files were lost along with the crashed node

Fault Tolerance in MapReduce

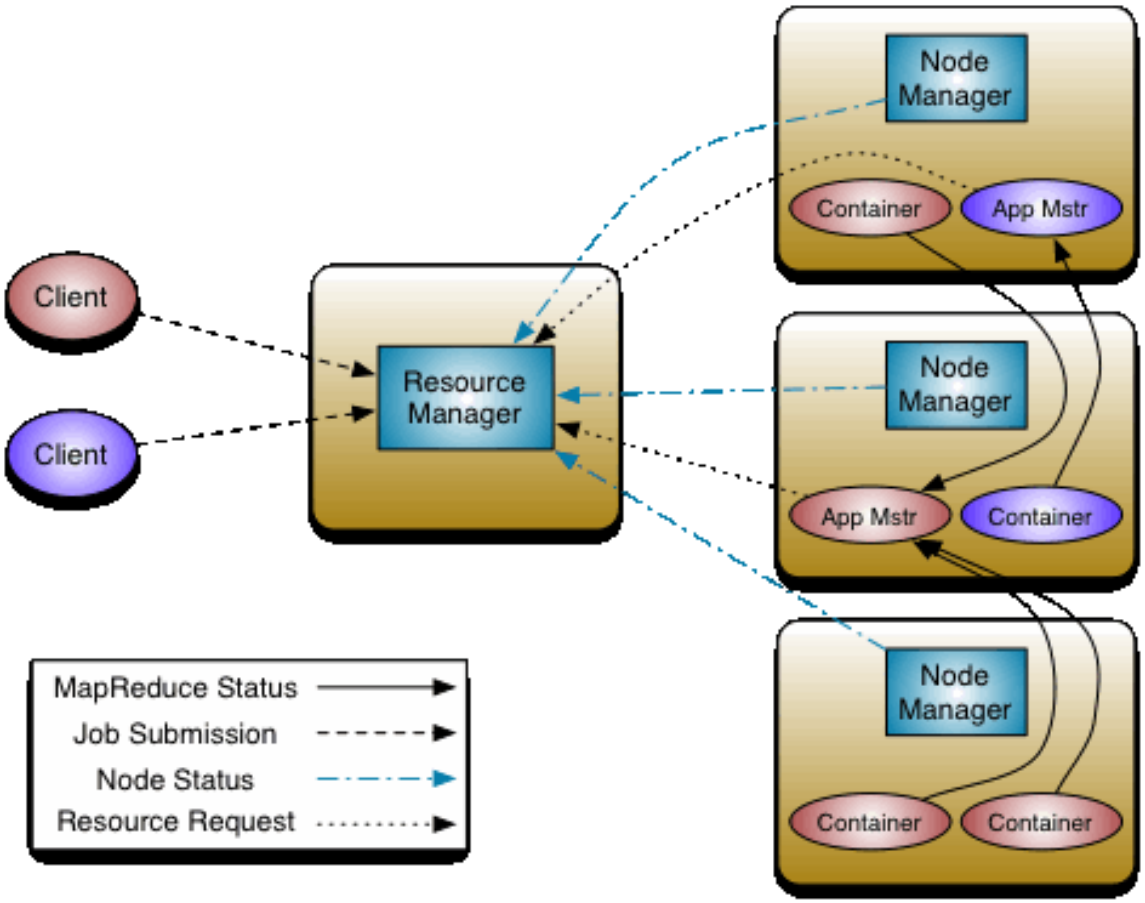
3. If a task is going slowly (straggler):
   - Launch second copy of task on another node ("speculative execution")
   - Take the output of whichever copy finishes first, and kill the other

- Surprisingly important in large clusters
  - Stragglers occur frequently due to failing hardware, software bugs, misconfiguration, etc
  - Single straggler may noticeably slow down a job

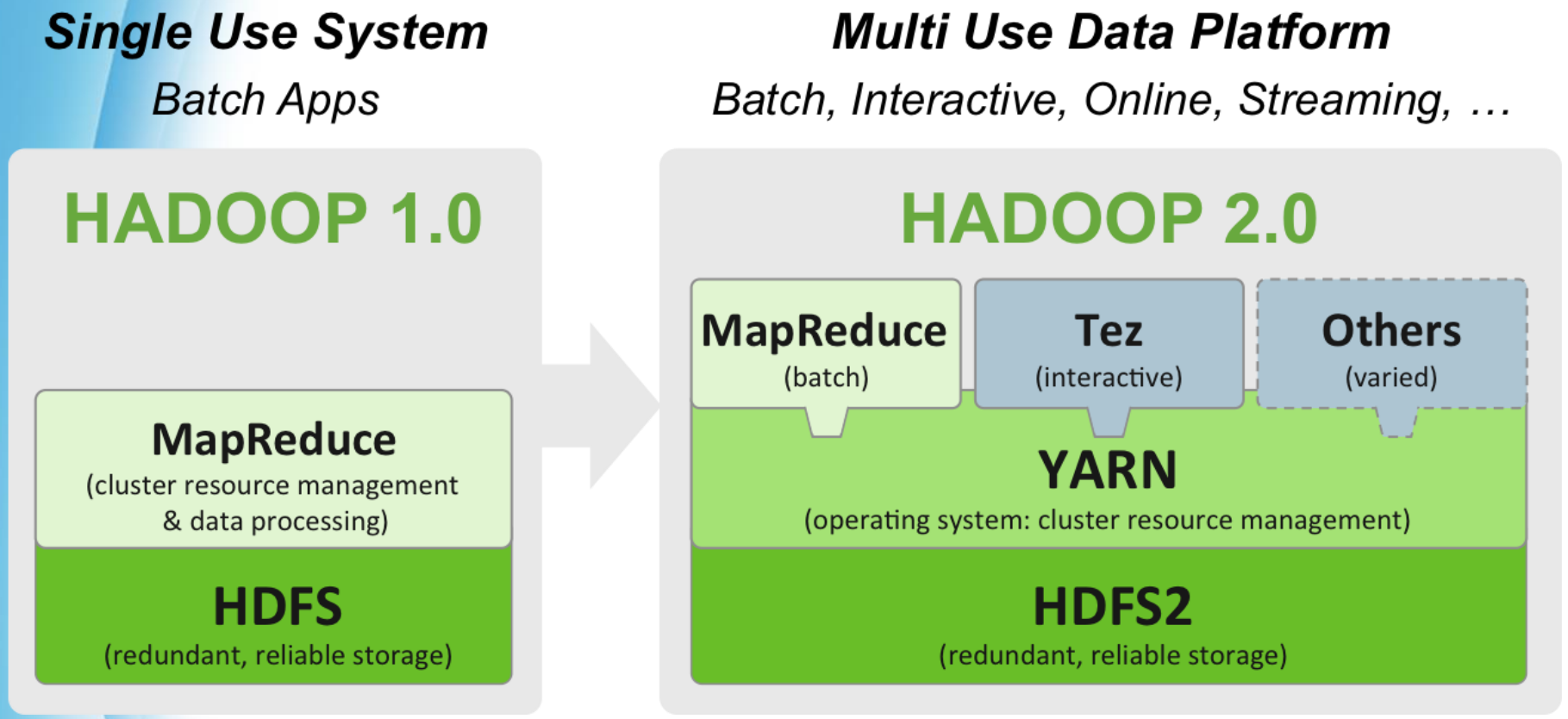
Hadoop 2: Big data's big leap forward

- The new Hadoop is the Apache Foundation's attempt to create a whole new general framework for the way big data can be stored, mined, and processed.
- The biggest constraint on scale has been Hadoop's job handling. All jobs in Hadoop are run as batch processes through **a single daemon** called JobTracker, which creates a scalability and processing-speed bottleneck.
- Hadoop 2 uses an entirely new job-processing framework built using **two daemons**: **ResourceManager**, which governs all jobs in the system, and **NodeManager**, which runs on each Hadoop node and keeps the ResourceManager informed about what's happening on that node.

Hadoop 2.0 – YARN (Yet Another Resource Negotiator)



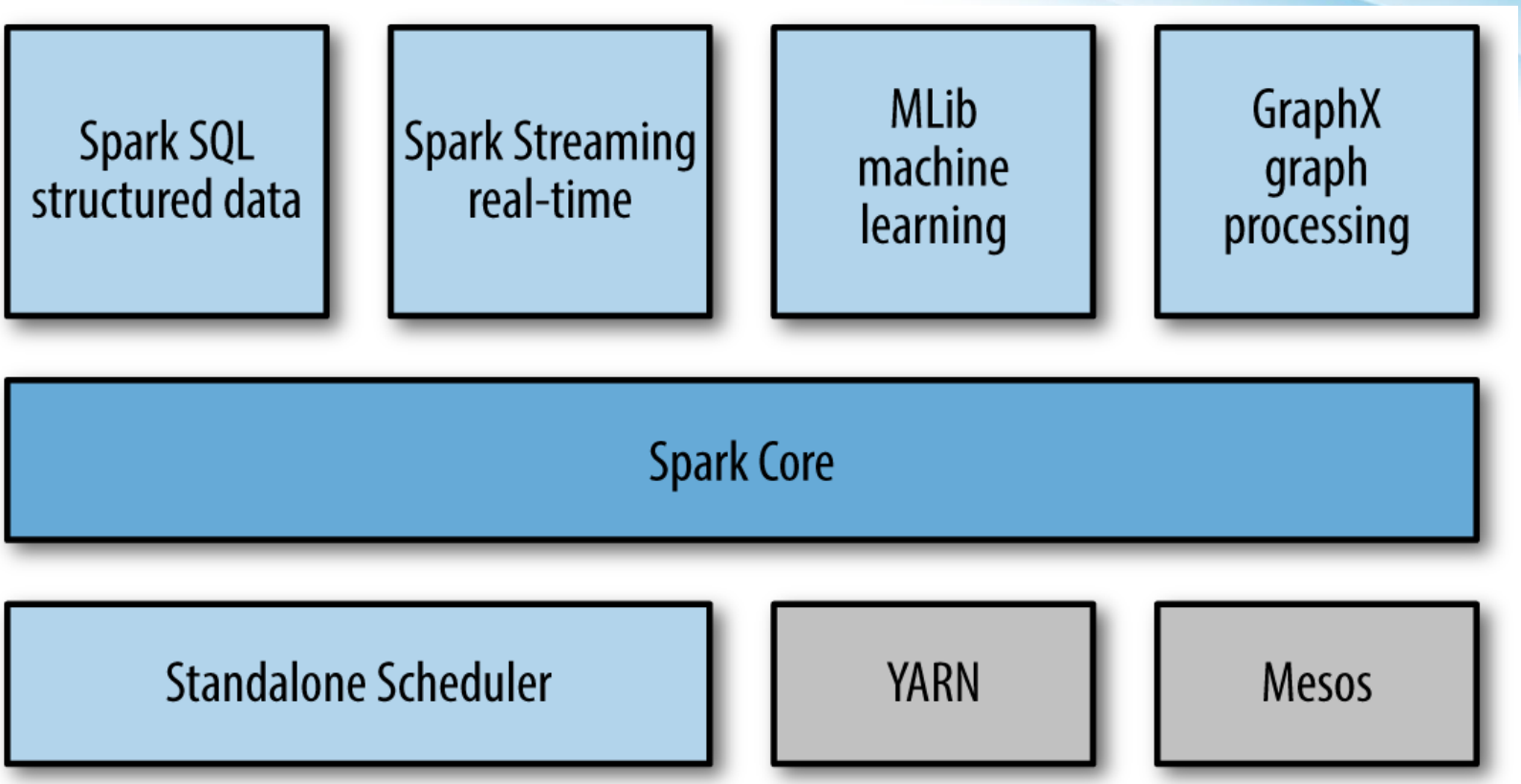
Comparison of Two Generations of Hadoop



Apache Spark

- An open-source cluster computing framework originally developed in the AMPLab at UC Berkeley (Dr. Matei Zaharia). In contrast to Hadoop's two-stage disk-based MapReduce paradigm, Spark's in-memory primitives provide performance up to 100 times faster for certain applications.
- Spark requires a cluster manager and a distributed storage system. For cluster manager, Spark supports standalone (native Spark cluster), Hadoop YARN, or Apache Mesos. For distributed storage, Spark can interface with a wide variety, including HDFS, Cassandra, Openstack Swift, and Amazon S3.
- In February 2014, Spark became an Apache Top-Level Project. Spark has over 1000 contributors in 2016.

Apache Spark



Summary

- MapReduce
  - Programming paradigm for data-intensive computing
  - Distributed & parallel execution model
  - Simple to program
    - The framework automates many tedious tasks (machine selection, failure handling, etc.)

- Microsoft Datacenter Tour (short version)

https://www.youtube.com/watch?v=zXsoygN_v7A

  - The world's largest data center

https://www.youtube.com/watch?v=4e97g7_qSxA

Thank You!