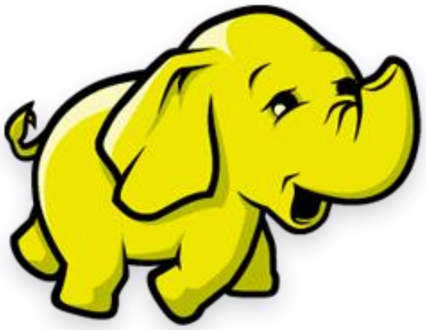
Apache Hadoop and MapReduce

COMP9313: Big Data Management

Hadoop



source

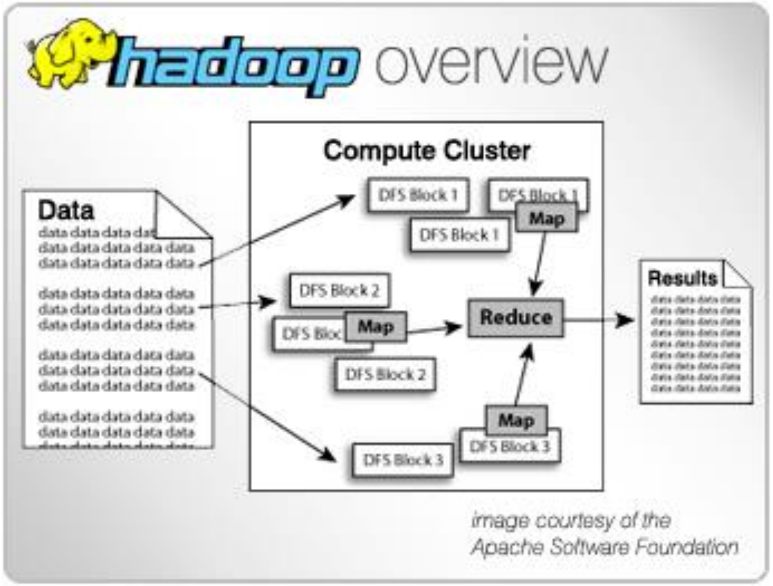
- Open-source data storage and processing platform
- Before the advent of Hadoop, storage and processing of big data was a big challenge
- Massively scalable, automatically parallelizable
  - ➢ Based on work from Google
    - Google: GFS + MapReduce + BigTable (Not open)
    - Hadoop: HDFS + Hadoop MapReduce + Hbase (opensource)
- Named by Doug Cutting in 2006 (worked at Yahoo! at that time), after his son's toy elephant

What’s offered by Hadoop?

- Redundant, fault-tolerant data storage
- Parallel computation framework
- Job coordination
- Programmers do not need to worry about:
  - Where are files located?
  - How to handle failures and data loss?
  - How to distribute computation?
  - How to program for scaling?

Why use Hadoop?

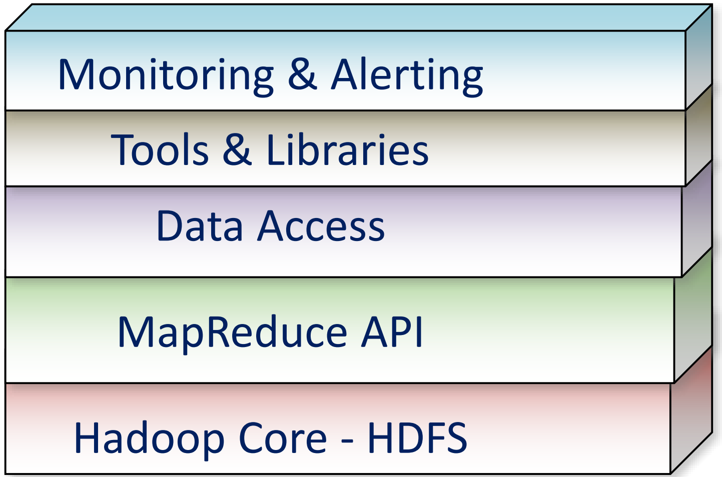
- Cheap
  - Scales to Petabytes or more easily
- Fast
  - Parallel data processing
- Suitable
  - ... for particular types of big data problems



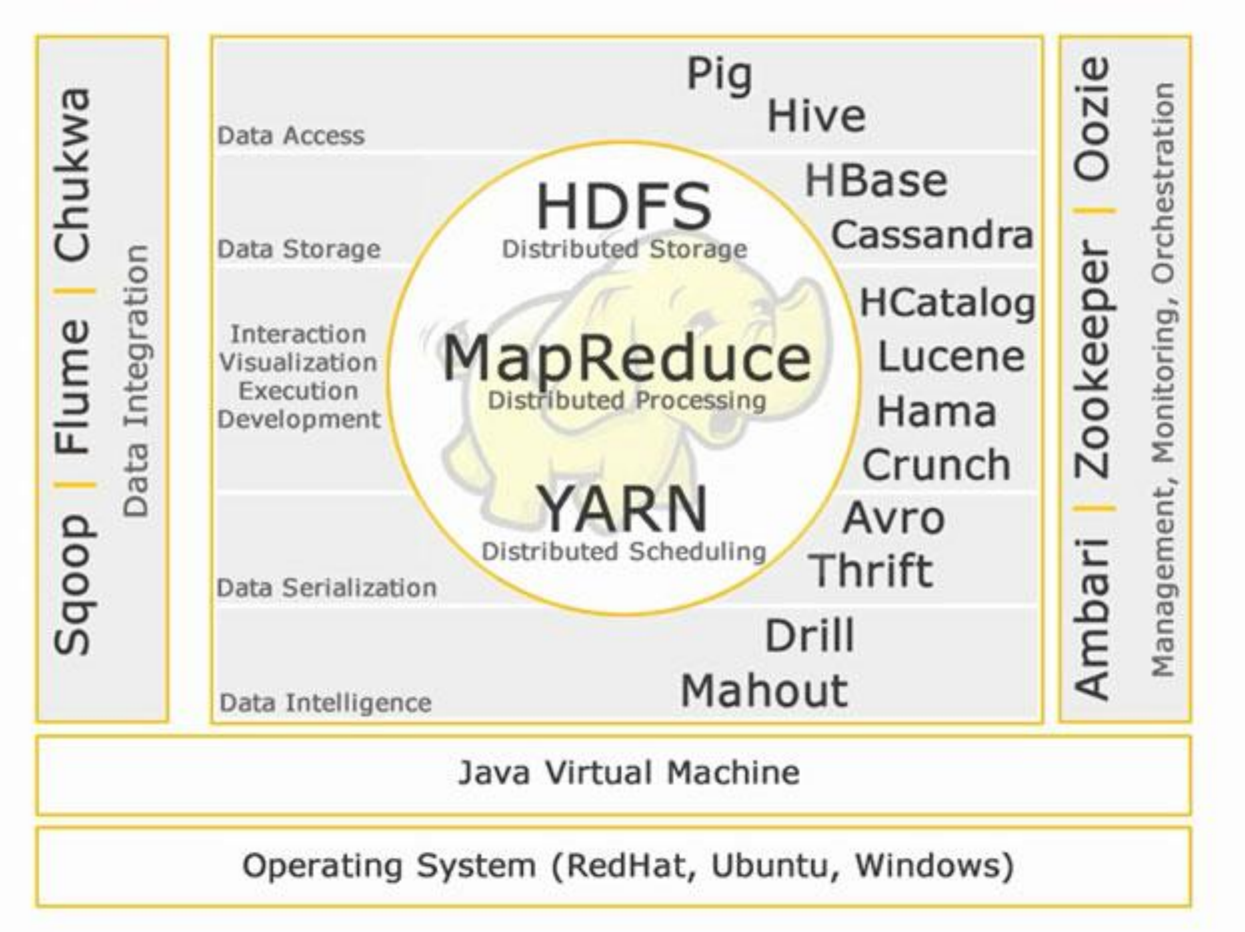
source

Hadoop is a set of Apache Frameworks…

- Data storage (HDFS)
  - Runs on commodity hardware
  - Horizontally scalable
- Processing (MapReduce)
  - Parallelized (scalable) processing
  - Fault Tolerant
- Yarn (Yet Another Resource Negotiator)
- Other Tools / Frameworks
  - Data Access
    - Hbase (column store), Hive (Data warehousing), Pig (high-level language on top of Hadoop), Mahout (library for ML / Data Analytics)
  - Tools
    - Hue (SQL Cloud editor), Sqoop (data transfer Hadoop / Structured stores)

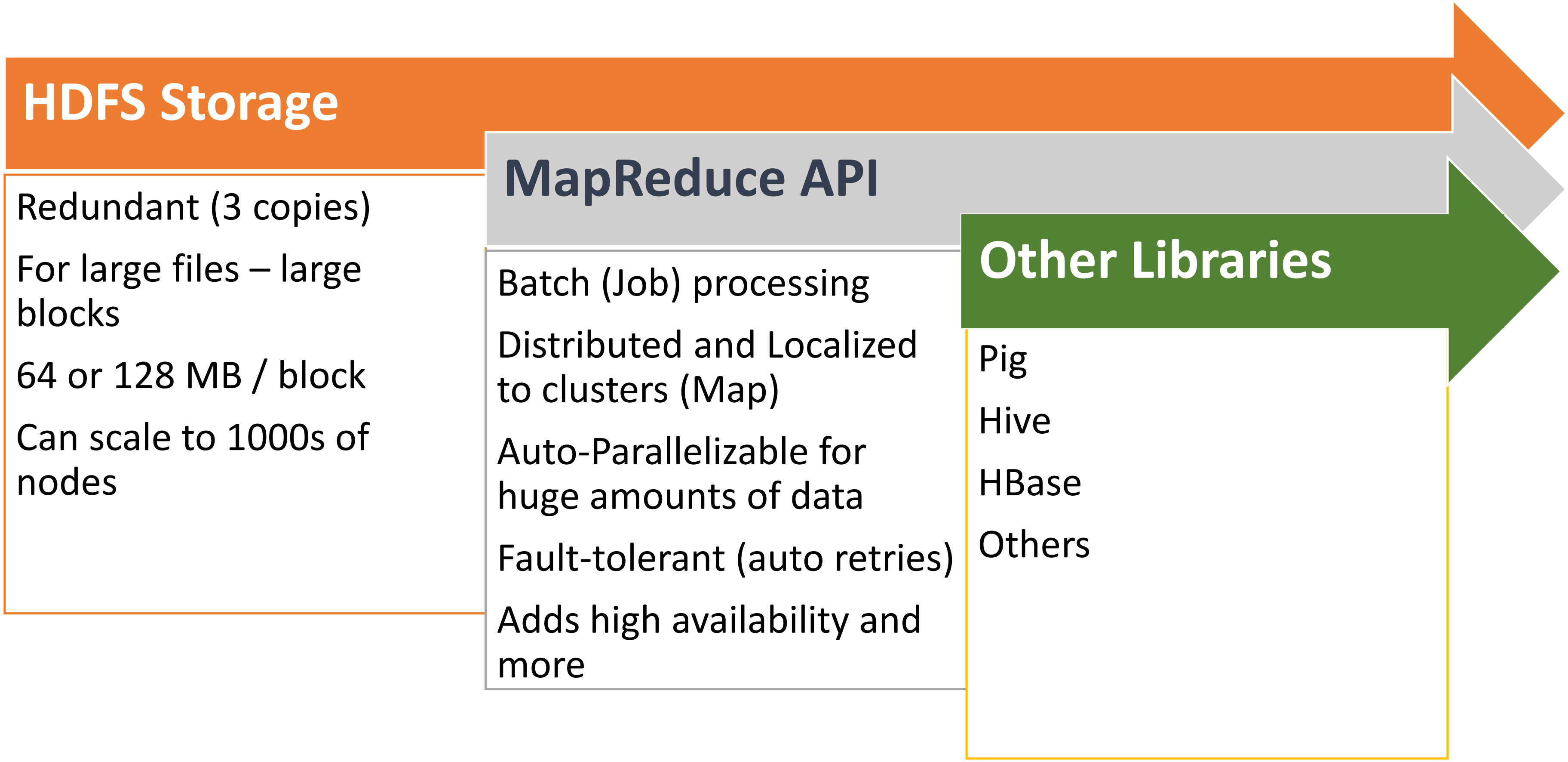


Hadoop ecosystem



source

Core Parts of a Hadoop Distribution



HDFS Storage

Redundant (3 copies)

For large files – large blocks

64 or 128 MB / block

Can scale to 1000s of nodes

MapReduce API

Batch (Job) processing

Distributed and Localized to clusters (Map)

Auto-Parallelizable for huge amounts of data

Fault-tolerant (auto retries)

Adds high availability and more

Other Libraries

Pig

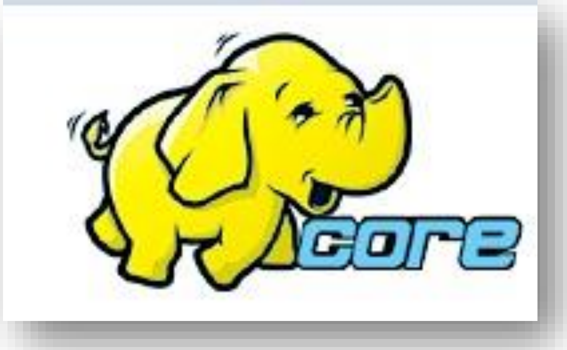
Hive

HBase

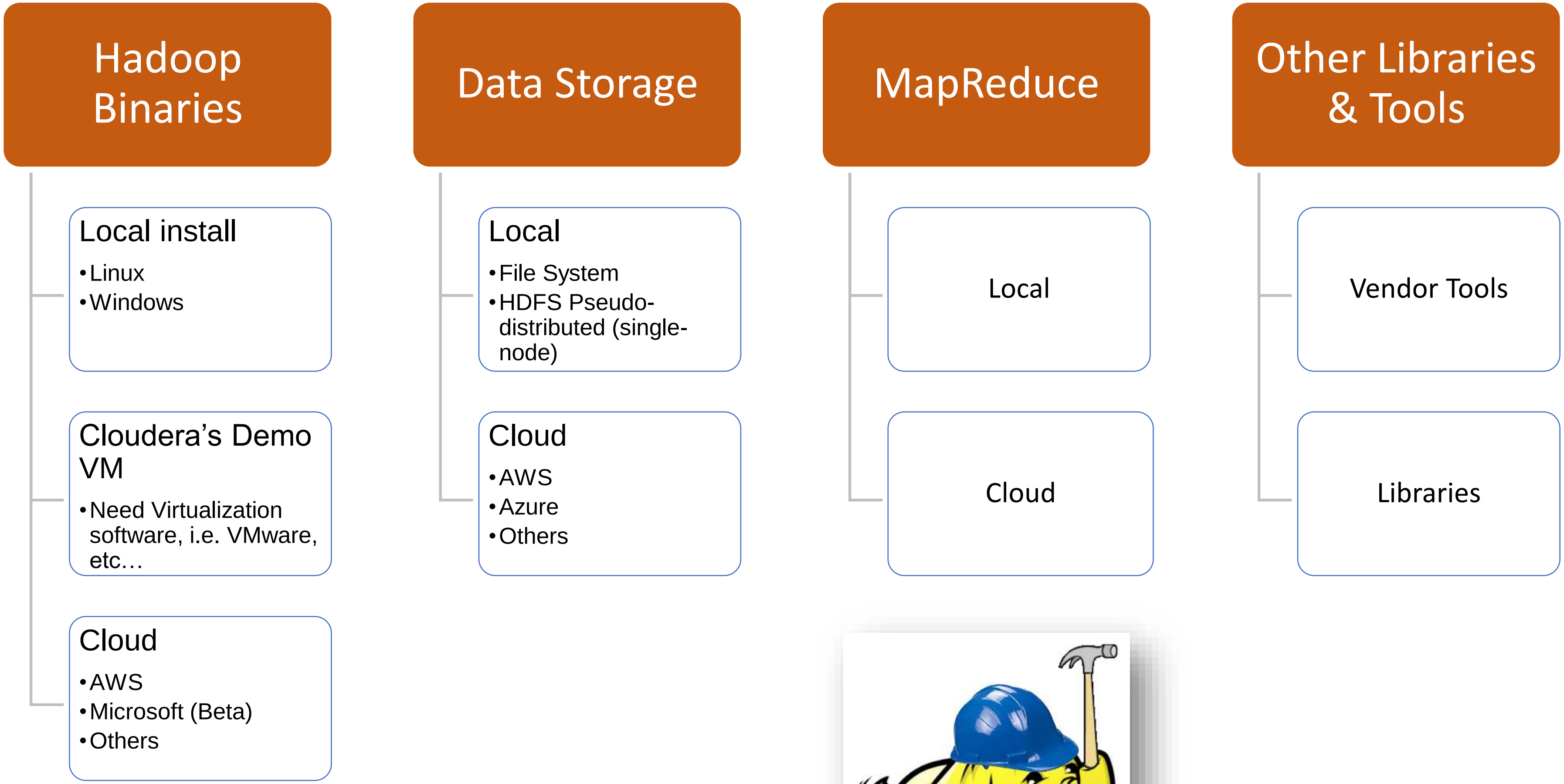
Others

Common Hadoop Distributions

- Open Source
  - Apache
- Commercial
  - Cloudera
  - Hortonworks
  - MapR
  - AWS MapReduce
  - Microsoft Azure HDInsight (Beta)



Setting up for Hadoop Development



Hadoop Binaries

- Local install
  - Linux
  - Windows
- Cloudera’s Demo VM
  - Need Virtualization software, i.e. VMware, etc…
- Cloud
  - AWS
  - Microsoft (Beta)
  - Others

Data Storage

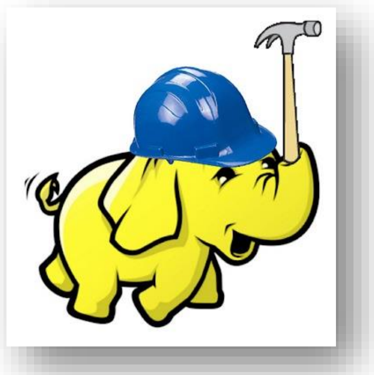
- Local
  - File System
  - HDFS Pseudo-distributed (single-node)
- Cloud
  - AWS
  - Azure
  - Others

MapReduce

- Local
- Cloud

Other Libraries & Tools

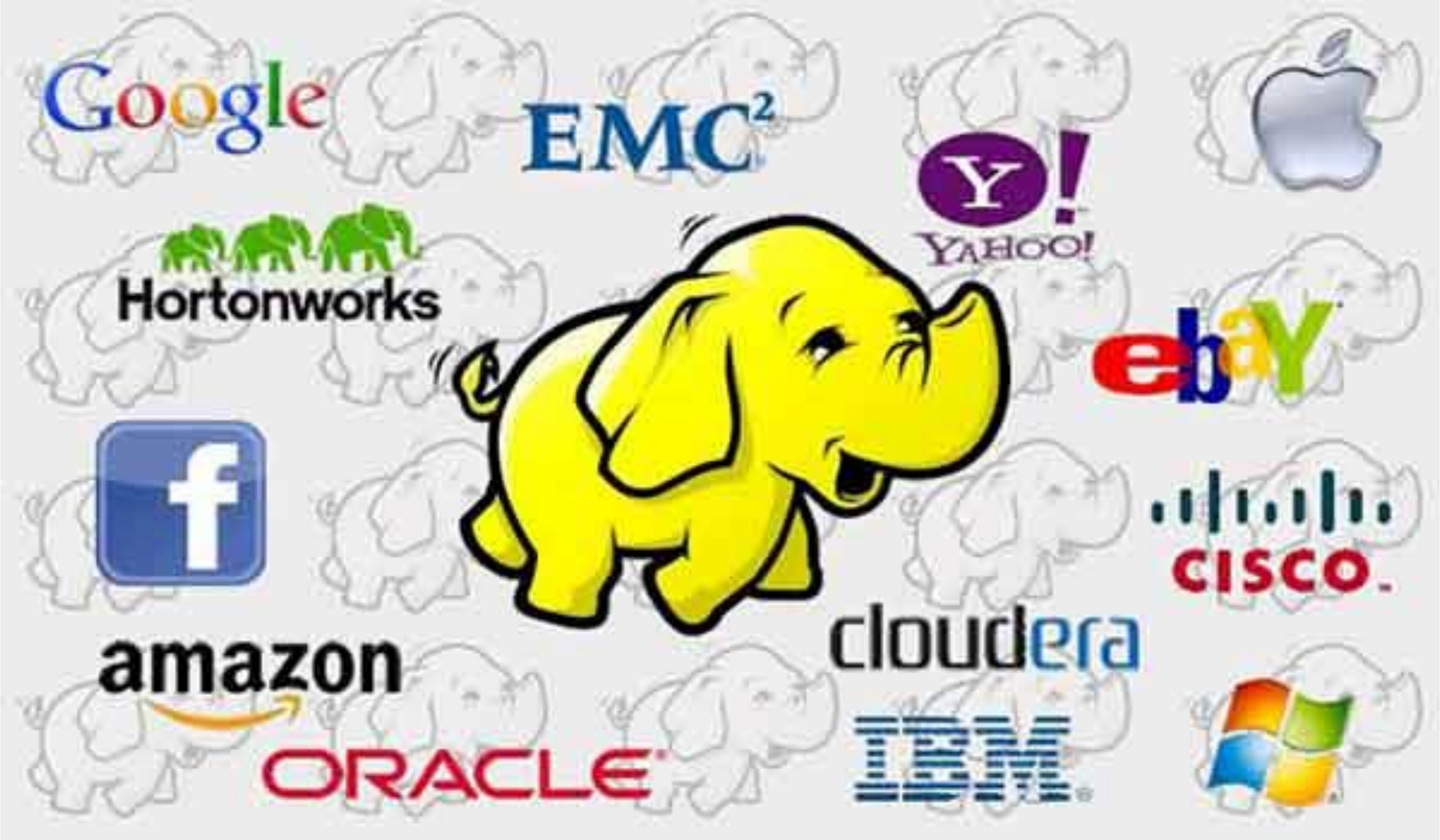
- Vendor Tools
- Libraries



RDMS vs Hadoop

Feature	RDBMS	Hadoop
Data variety	Mainly structured data	Structured, semi-structured and unstructured data
Data storage	Average size data (GBs)	Used for large datasets (TBs, PTs)
Querying	SQL	HQL (Hive Query Language)
Schema	Required on write (static schema)	Required on read (dynamic schema)
Speed	Read are fast	Both read and write are fast
Use case	OLTP (Online Transaction Processing)	Analytics (audio, video, logs), data discovery
Data objects	Works on relational tables	Works on key/value pairs
Scalability	Vertical	Horizontal
Hardware profile	High-end servers	Commodity / Utility hardware
Integrity	ACID	Low

Companies using Hadoop



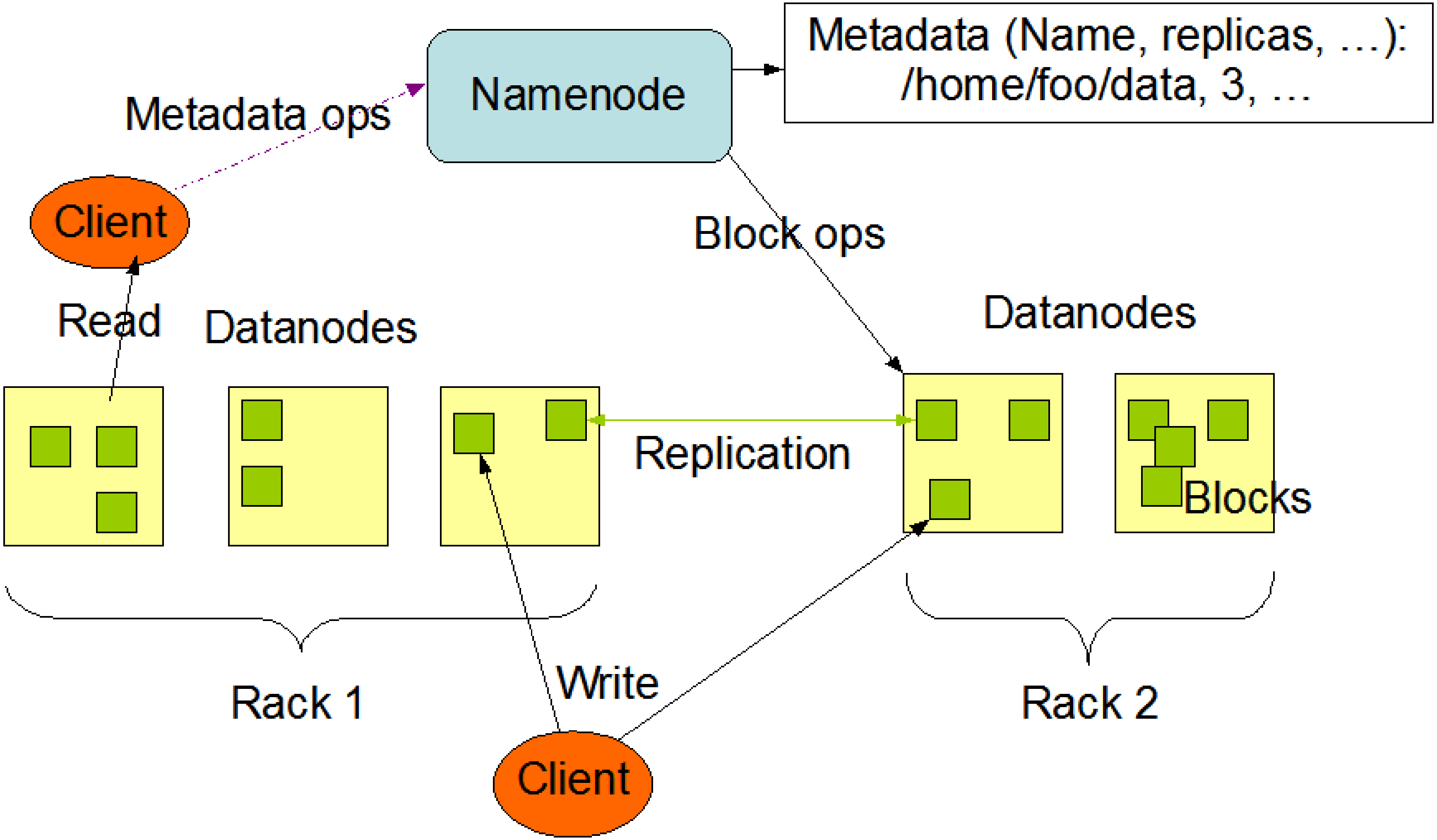
source

HDFS
Hadoop Distributed File System

What is HDFS?

- It provides data storage for Hadoop
- HDFS splits the data unit into smaller units called blocks and stores them in a distributed manner
- HDFS has a **Master-slave architecture**
- It has got two daemons running:
  - master node – NameNode
  - and slave nodes – DataNode.

HDFS Architecture

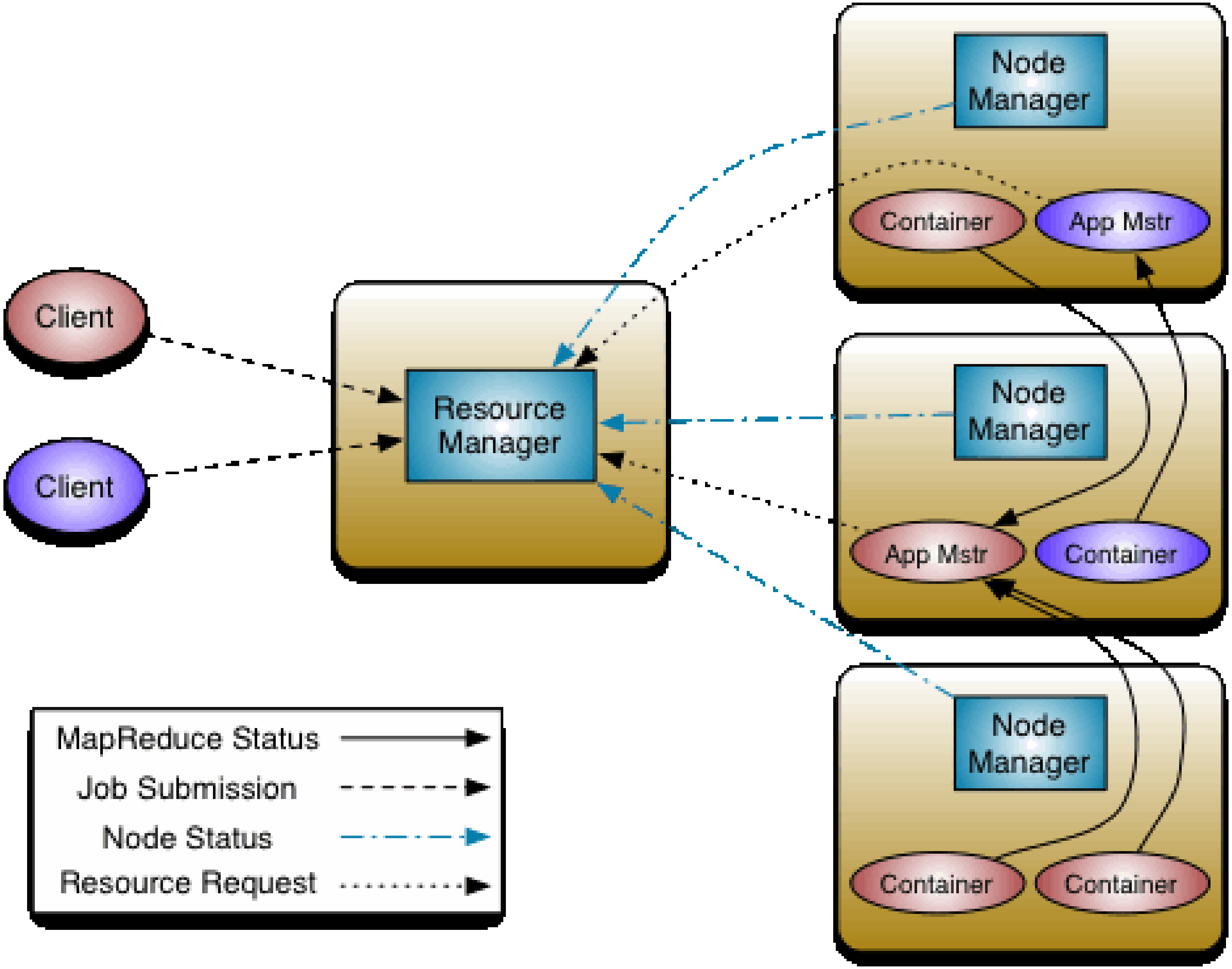


https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

YARN
Yet Another Resource Negotiator

What is YARN?

- The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring into separate daemons
- Yarn was introduced in Hadoop 2.x.
- Yarn allows different data processing engines like graph processing, interactive processing, stream processing as well as batch processing to run and process data stored in HDFS
- **Components of YARN**
  - **Client:** For submitting MapReduce jobs.
  - **Resource Manager:** To manage the use of resources across the cluster
  - **Node Manager:**For launching and monitoring the computer containers on machines in the cluster.
  - **Application Master:** Checks tasks running the MapReduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager, and managed by the node managers.


Node Manager
Container
App Mstr
Client
Resource Manager
Client
Node Manager
App Mstr
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

MapReduce

What is MapReduce?

- Originated from Google, [OSDI’04]
  - MapReduce: Simplified Data Processing on Large Clusters
  - Jeffrey Dean and Sanjay Ghemawat (paper)
- Programming model for parallel data processing
- Hadoop can run MapReduce programs written in various languages:
  e.g. Java, Ruby, Python, C++
- For large-scale data processing
  - Exploits large set of commodity computers
  - Executes process in distributed manner
  - Offers high availability

Motivation for MapReduce (1)

- Typical big data problem challenges:
  - How do we break up a large problem into smaller tasks that can be executed in parallel?
  - How do we assign tasks to workers distributed across a potentially large number of machines?
  - How do we ensure that the workers get the data they need?
  - How do we coordinate synchronization among the different workers?
  - How do we share partial results from one worker that is needed by another?
  - How do we accomplish all of the above in the face of software errors and hardware faults?

Motivation for MapReduce (2)

- There was need for an abstraction that hides many system-level details from the programmer.
- MapReduce addresses this challenge by providing a simple abstraction for the developer, transparently handling most of the details behind the scenes in a **scalable**, **robust**, and **efficient** manner.
- MapReduce separates the **what** from the **how**

Typical Big Data Problem

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

Map

Reduce

Key idea:
Provide a **functional abstraction** for these two operations

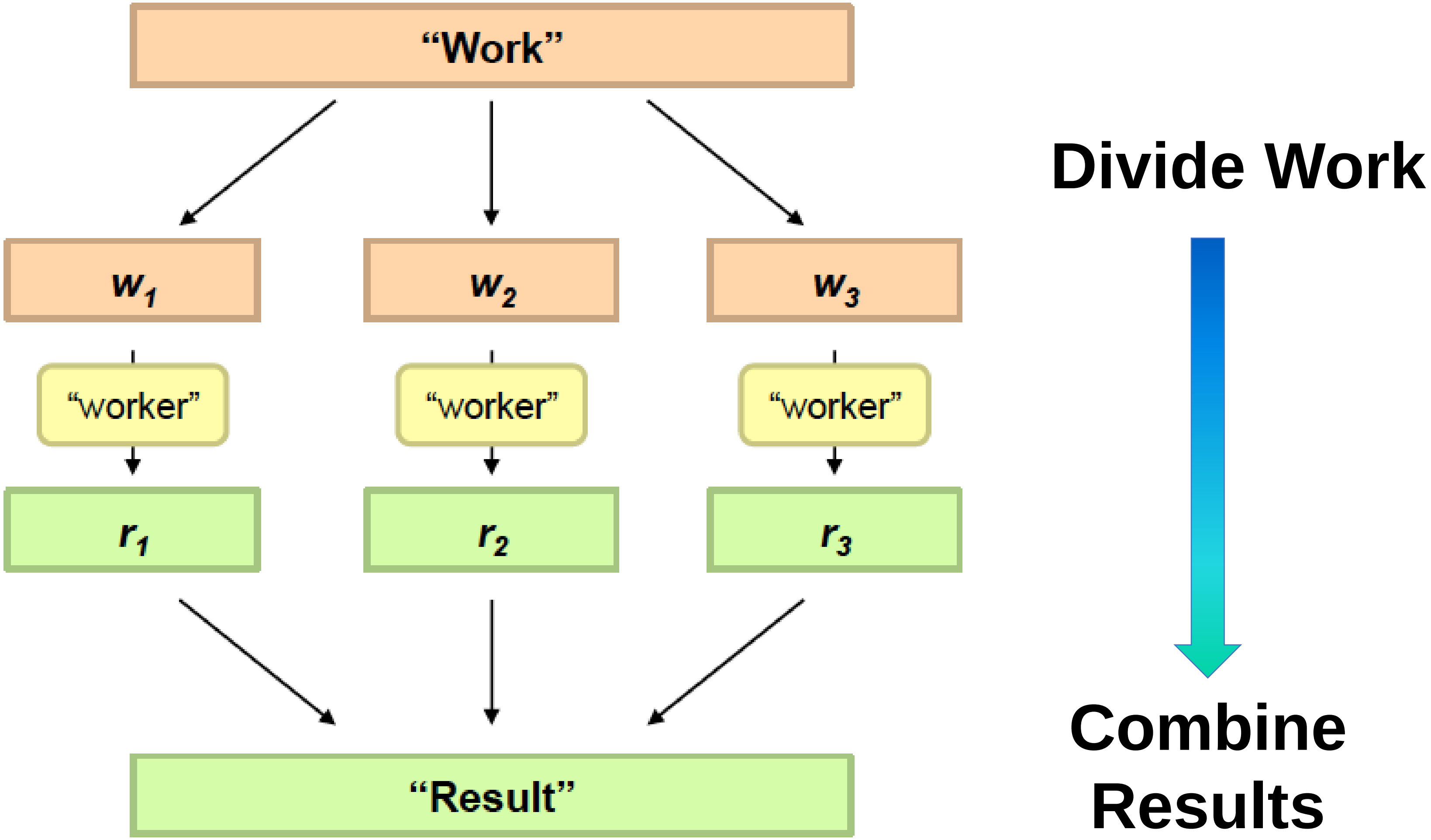
The Idea of MapReduce (1)

- Inspired by the map and reduce functions in functional programming
- We can view map as a transformation over a dataset
  - This transformation is specified by the function *f*
  - Each functional application happens in isolation
  - The application of *f* to each element of a dataset can be parallelized in a straightforward manner
- We can view reduce as an aggregation operation
  - The aggregation is defined by the function *g*
  - Data locality: elements in the list must be “brought together”
  - If we can group elements of the list, also the reduce phase can proceed in parallel
- The framework coordinates the map and reduce phases:
  - Grouping intermediate results happens in parallel

The Idea of MapReduce (2)

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed file system (HDFS)
- You don’t know:
  - Where mappers and reducers run
  - When a mapper or reducer begins or finishes
  - Which input a particular mapper is processing
  - Which intermediate key a particular reducer is processing

Philosophy to Scale for Big Data Processing



“Work”

w_1 w_2 w_3

“worker” “worker” “worker”

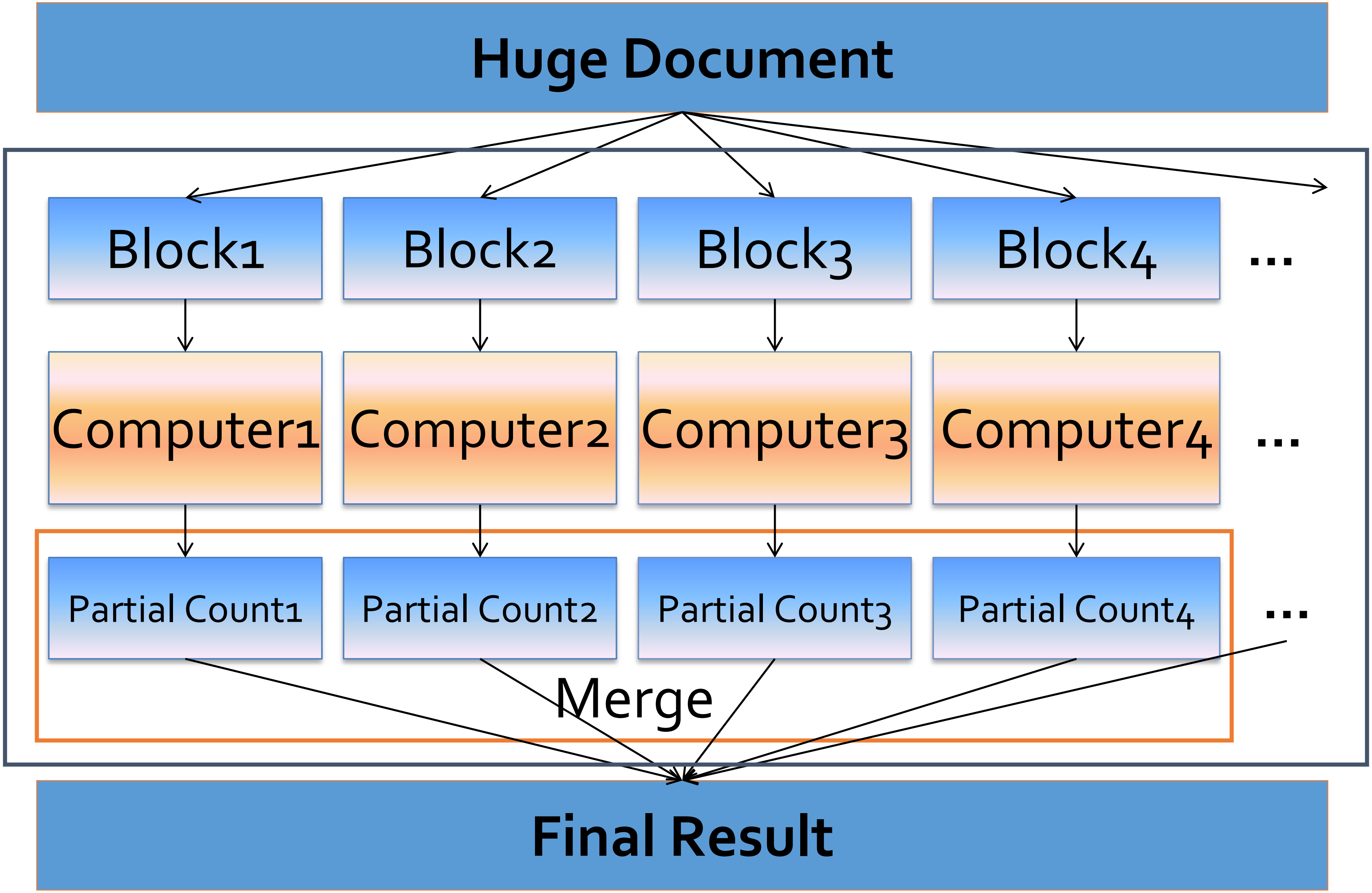
r_1 r_2 r_3

“Result”

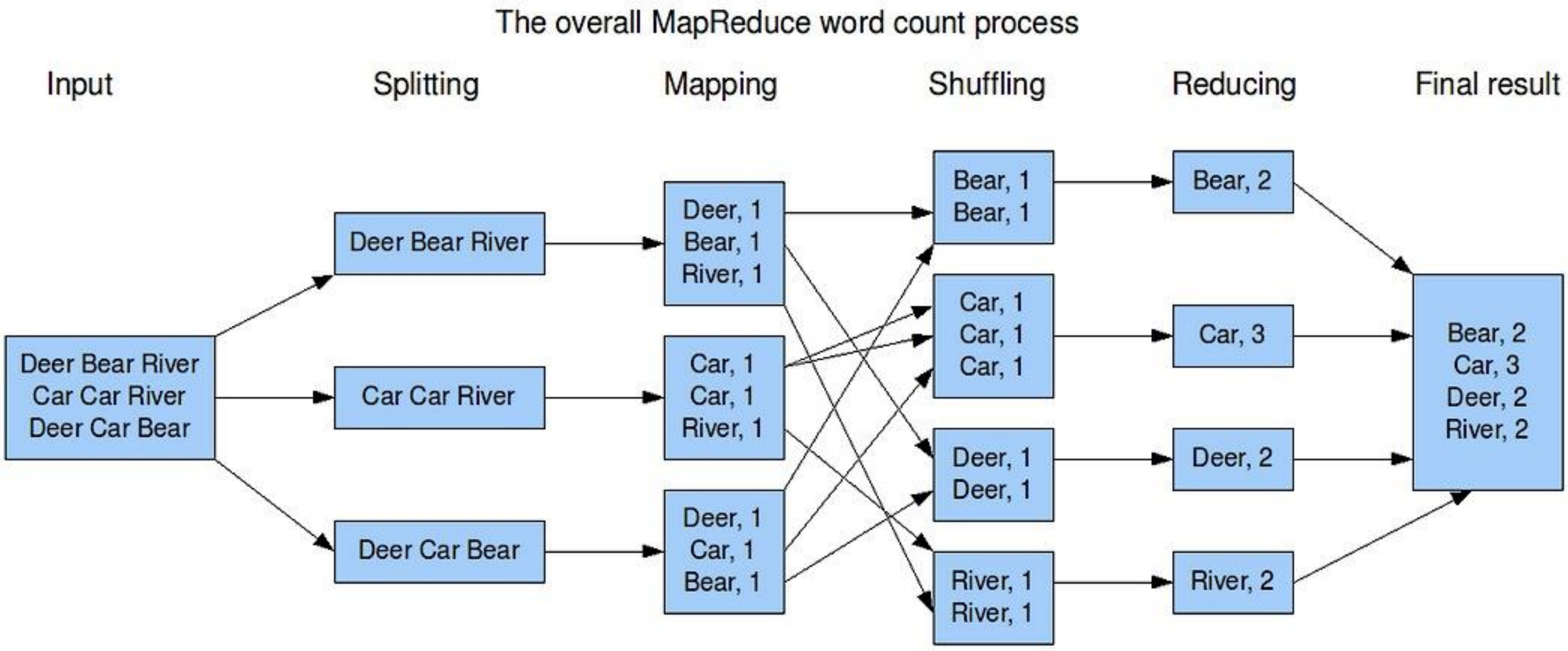
Divide Work

Combine Results

Distributed Word Count



MapReduce Example - WordCount



- Hadoop MapReduce is an implementation of MapReduce
  - MapReduce is a computing paradigm (Google)
  - Hadoop MapReduce is an open-source software

Data Structures in MapReduce

- Key-value pairs are the basic data structure in MapReduce
  - Keys and values can be: integers, float, strings, raw bytes, etc.
  - They can also be arbitrary data structures
- The design of MapReduce algorithms involves:
  - Imposing the key-value structure on arbitrary datasets
    - E.g.: for a collection of Web pages, input keys may be URLs and values may be the HTML content
  - In some algorithms, input keys uniquely identify a record
  - Keys can be combined in complex ways to design various algorithms

Map and Reduce Functions

- Programmers specify two functions:
  - **map** (k_1, v_1) → list [<k_2, v_2>]
    - Map transforms the input into key-value pairs to process
  - **reduce** (k_2, list [v_2]) → [<k_3, v_3>]
    - Reduce aggregates the list of values for each key
    - All values with the same key are sent to the same reducer
  - list [<k_2, v_2>] will be grouped according to key k_2 as (k_2, list [v_2])

- The MapReduce environment takes in charge of everything else…

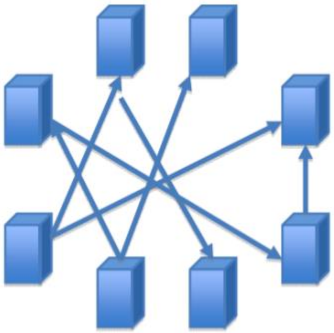
- A complex program can be decomposed as a succession of Map and Reduce tasks

Understanding MapReduce

□ Map>>

- (K1, V1) →
  - Info in
  - Input Split
- list (K2, V2)
  - Key / Value out (intermediate values)
  - One list per local node
  - Can implement local Reducer (or Combiner)

□ Shuffle/Sort>>



□ Reduce

- (K2, list(V2)) →
  - Shuffle / Sort phase precedes Reduce phase
  - Combines Map output into a list
- list (K3, V3)
  - Usually aggregates intermediate values

(input) <k1, v1> → **map** → <k2, v2> → **shuffle/sort** → <k2, list(V2)> → **reduce** → <k3, v3> *(output)*

WordCount - Mapper

- Let’s count number of each word in documents (e.g., Tweets/Blogs)
  - Reads input pair <k1,v1>
    - The input to the mapper is in format of <docID, docText>:

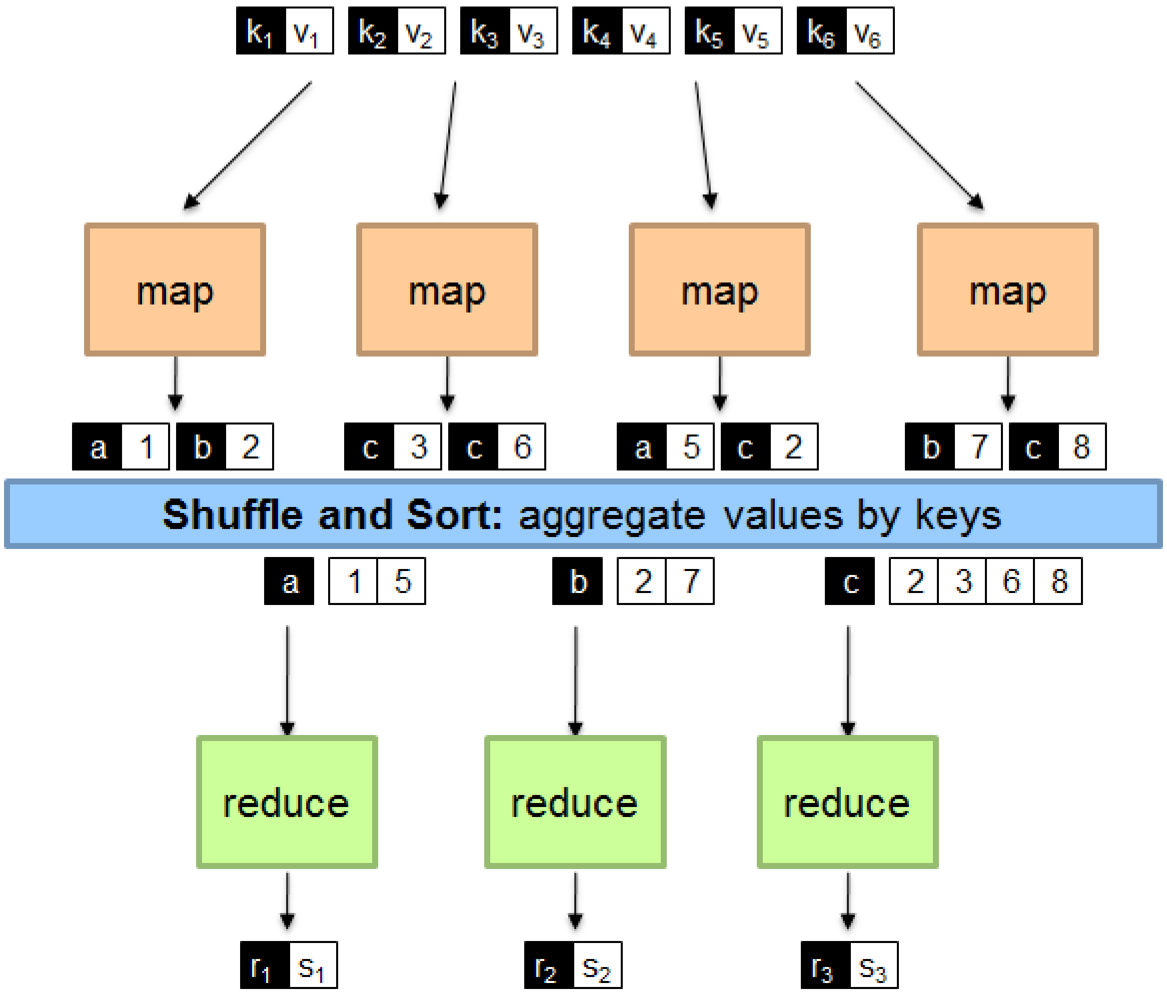
      <D1,“Hello World” >,<D2,“Hello Hadoop Bye Hadoop” >
  - Outputs pairs <k2, v2>
    - The output of the mapper is in format of <term, 1>:

  <Hello, 1><World, 1><Hello, 1><Hadoop, 1><Bye, 1><Hadoop, 1>
  - After shuffling and sort, reducer receives <k2, list(v2)>

  <Hello, {1, 1}><World, {1}><Hadoop, {1, 1}><Bye, {1}>
  - The output is in format of <k3, v3>:

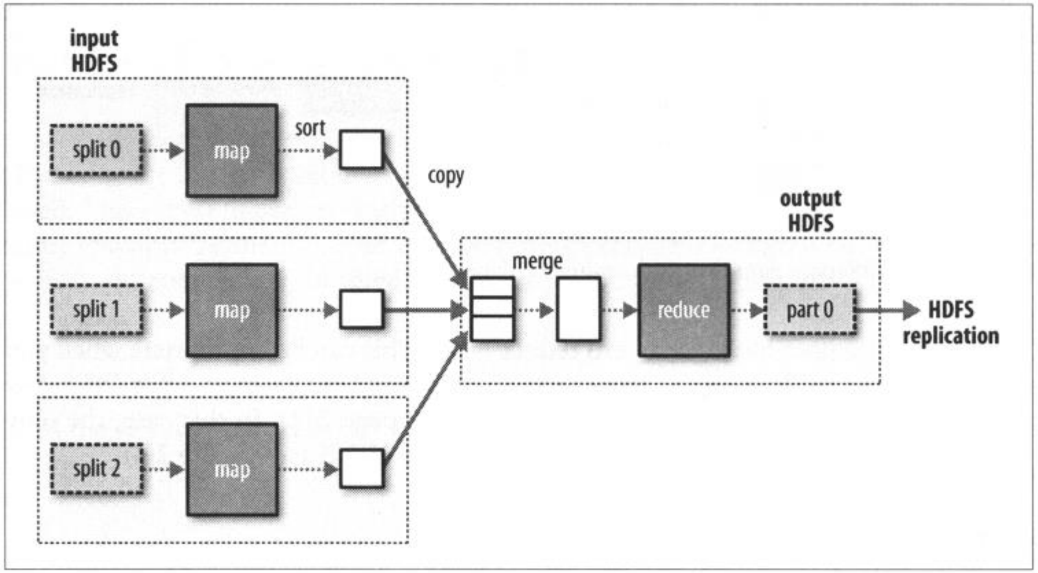
  <Hello, 2><World, 1><Hadoop, 2><Bye, 1>

WordCount - Mapper



Shuffle and Sort

- Shuffle
  - Input to the Reducer is the sorted output of the mappers. In this phase the framework fetches the relevant partition of the output of all the mappers, via HTTP.
- Sort
  - The framework groups Reducer inputs by keys (since different Mappers may have output the same key) in this stage.
- Hadoop framework handles the Shuffle and Sort step .



Useful Resources

- https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
- https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html
- https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Questions?

Notes

- Thursday lecture time we’ll have a hands-on activity
  - Bring Laptop
  - We’ll run a step by step activity
  - We can’t expect for everyone to complete in time that is why we have Labs (starting week4)7yju