Do this while we start

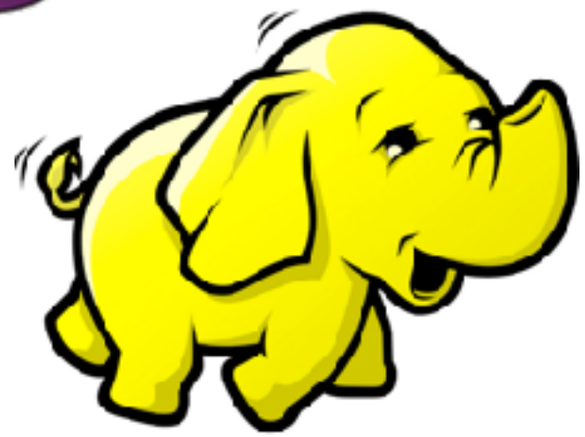
- Open a Web Browser
- Visit http://10.10.27.49/workshop.ova
- Temp: http://169.254.233.110/workshop.ova
- Open VirtualBox
- Import the file that was downloaded as an Appliance
    - In the File menu, select Import Appliance. Select the Downloaded File
    - If you can allot more RAM to the appliance. Go ahead and do so
    - If not. I am sorry!

HDFS, MapReduce and Spark

Hitesh Dharamdasani
Informant Networks


HIVE


mahout

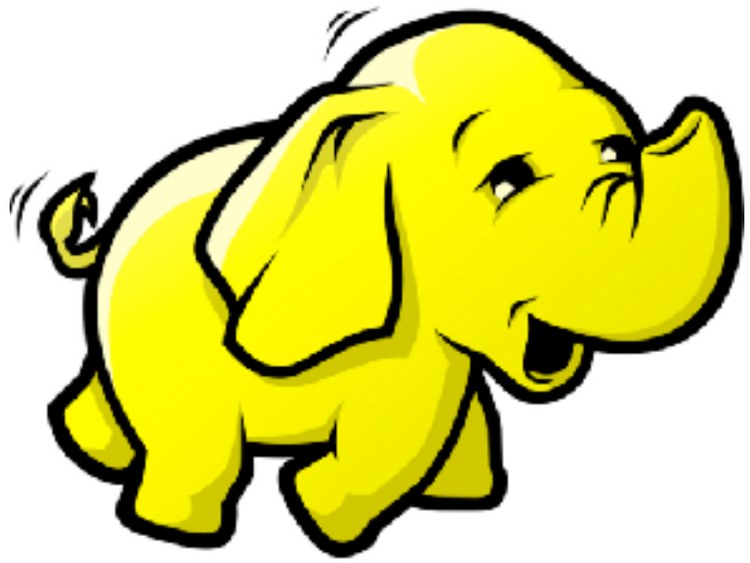



Scalding


Spark


APACHE
HBASE

Today we will play with



What the hell does “Distributed” even mean?

- Distributed does not mean “Parallel Computation”
- Distributed does not mean 4000 Clients and 1 Server
- Distributed does not mean “Grid Computing”
- If the words “Tightly Coupled” are used, Its *NOT* distributed
- Good examples: git-scm, Internet, MapReduce
- Bad Examples: CERN, GIT Network, etc…

What is BigData?

- Very broad term
- Usually means that computation code has become minuscule compared to error handling and synchronization
- Or if data is heterogeneous
- 100 GB is not yet BigData. 10+ TB. Probably?
- MapReduce won sorting challenge. 102 TB in 72 mins with 2100 nodes in 2014
- **Data Processing using Distributed Systems**

Error Amplification Problem

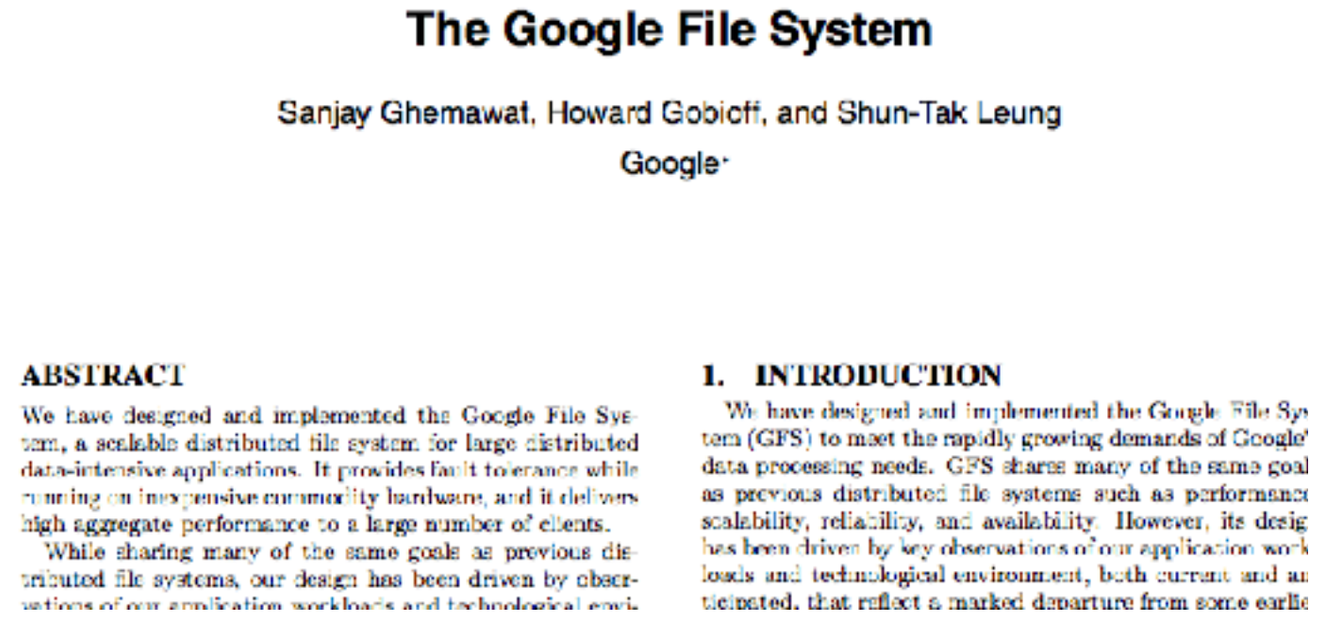
- 5 commodity machines cheaper than one large machine
- If availability of large machine is 99.9% it will fail once day in a year.
- 5 machines will fail 5 days in a year.
- Systems need to incorporate this downtime
- Imagine no Google for 5 days a year.

The problem of Scale

- “Anything that can go wrong will go wrong” - Murphy
- At 10000+ machines, If one fails everyday, A lot of failures will happen time and again
- At large scale. Small error rates manifest themselves.
- Leads to Error Amplification problem

The Origins
GFS and MapReduce

- Produced by Jeff Dean, Sanjay Ghemawat and others
- GFS - Google File System
- Yahoo! open sourced version - Hadoop
- Hadoop = HDFS(based on GFS) + MapReduce
- Comes under the Apache License
- **Fun Fact:** Hadoop was the name of Doug Cutting(Hadoop Inventor) kids elephant doll



The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by obser-

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier

Proceedings of the 19th ACM
symposium on Operating systems principles
Over 5000 citations

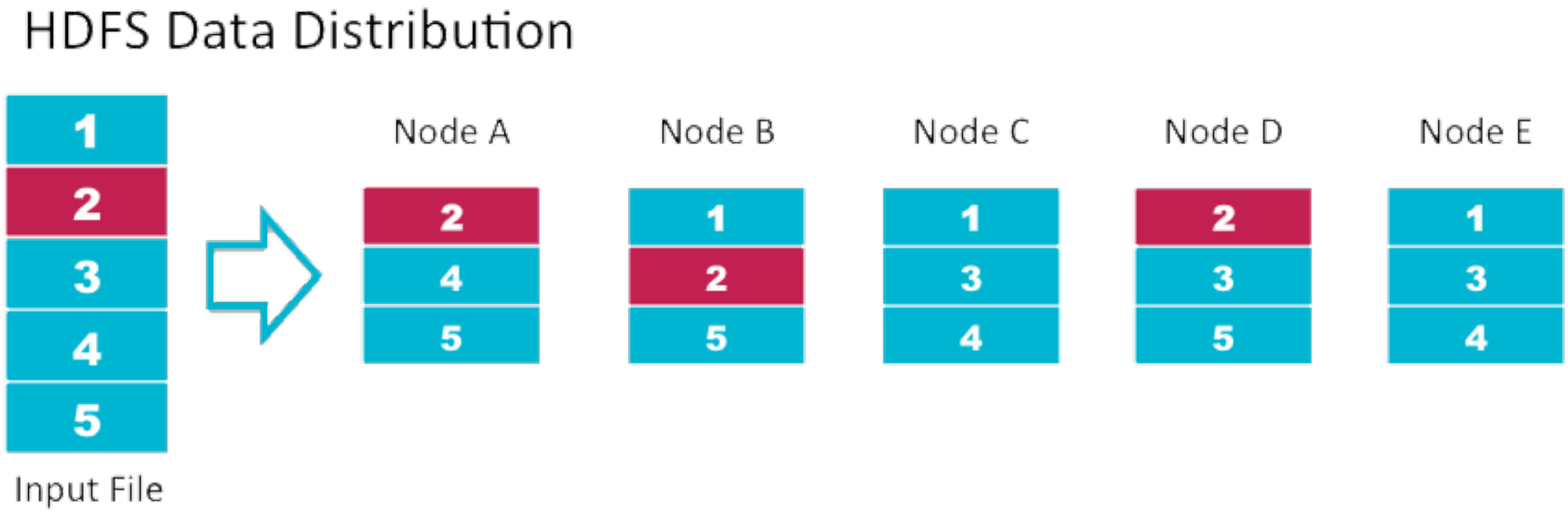
Hadoop Distributed File System (HDFS)

- Just a way of storing files. Like FAT, NTFS, ext4…
- Does not do computation at all
- Stores data with high degree of redundancy. Minimises loss due to hardware failure
- Built for Throughput, not for speed!

Key ideas

- Files are stored as chunks
    - Fixed size(64 MB).
- Reliability through replication.
    - Each chunk is replicated across N+ chunk servers (N >= 1)
- Single master to co ordinate access, keep metadata (Not Distributed Yet)
    - Simple centralized management.
- No data caching
    - Little benefit due to large datasets, streaming reads.

In a nutshell



More

- Replication is rack aware
- Fault tolerance is extremely high
- Supports scale of 10,000+ machines
- Allows for building higher degrees of abstraction

Map Reduce

- Hopes for a fault tolerant file system underneath (i.e. HDFS)
- Applies computation to the data at the node
- No need to bring data to one place
- Send the computation to where the data resides
- Exploits the associative property of Mathematics

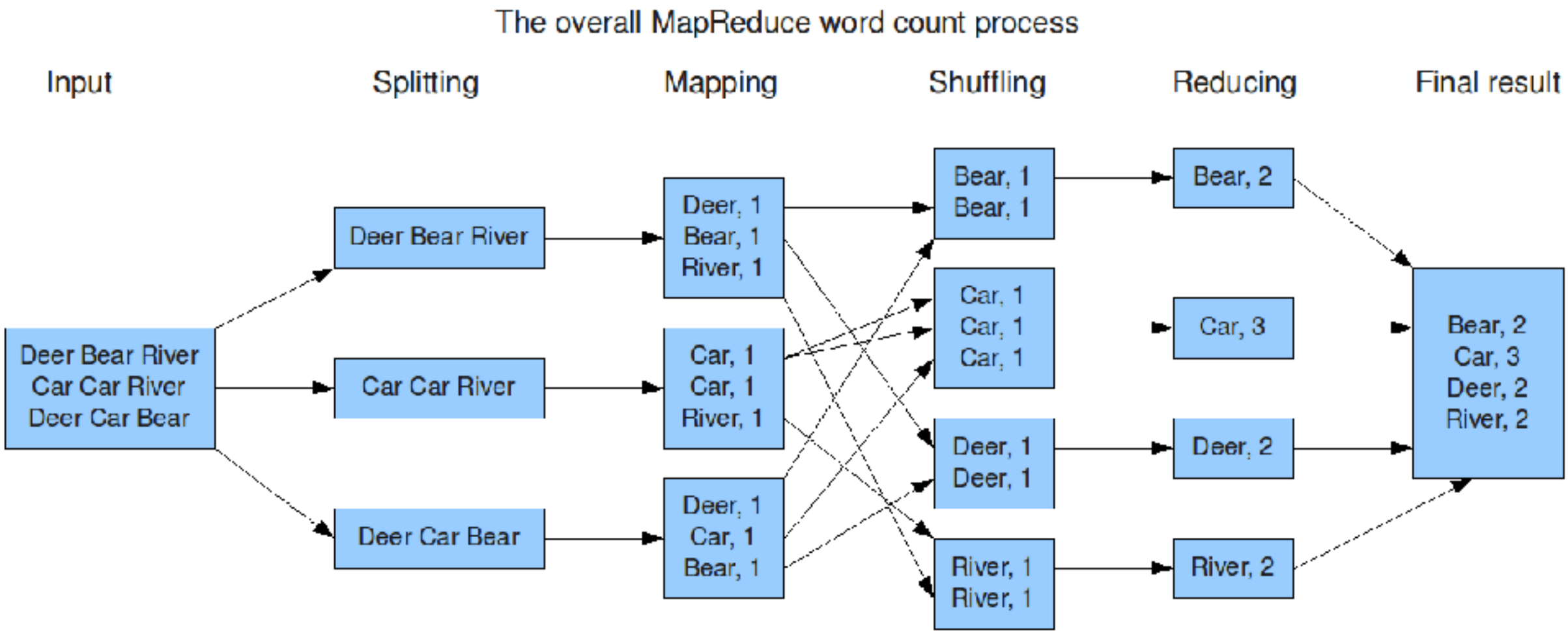
Map Reduce

- Map is a one-to-one operation
- For some ‘x’, Apply a function, f(x) = y

- Reduce collects all y’s to give a result
- count(y) = n

- All maps are done in parallel, reduce is done in parallel over similar data items.

Illustration



What happens when things break?

- Computation on nodes that have failed get restarted where other copies of data is present

- Slow moving computations are started on other nodes. The one that finishes first is taken into effect

- When a hard drive on a node fails. Just replace it. Data will be put back automatically

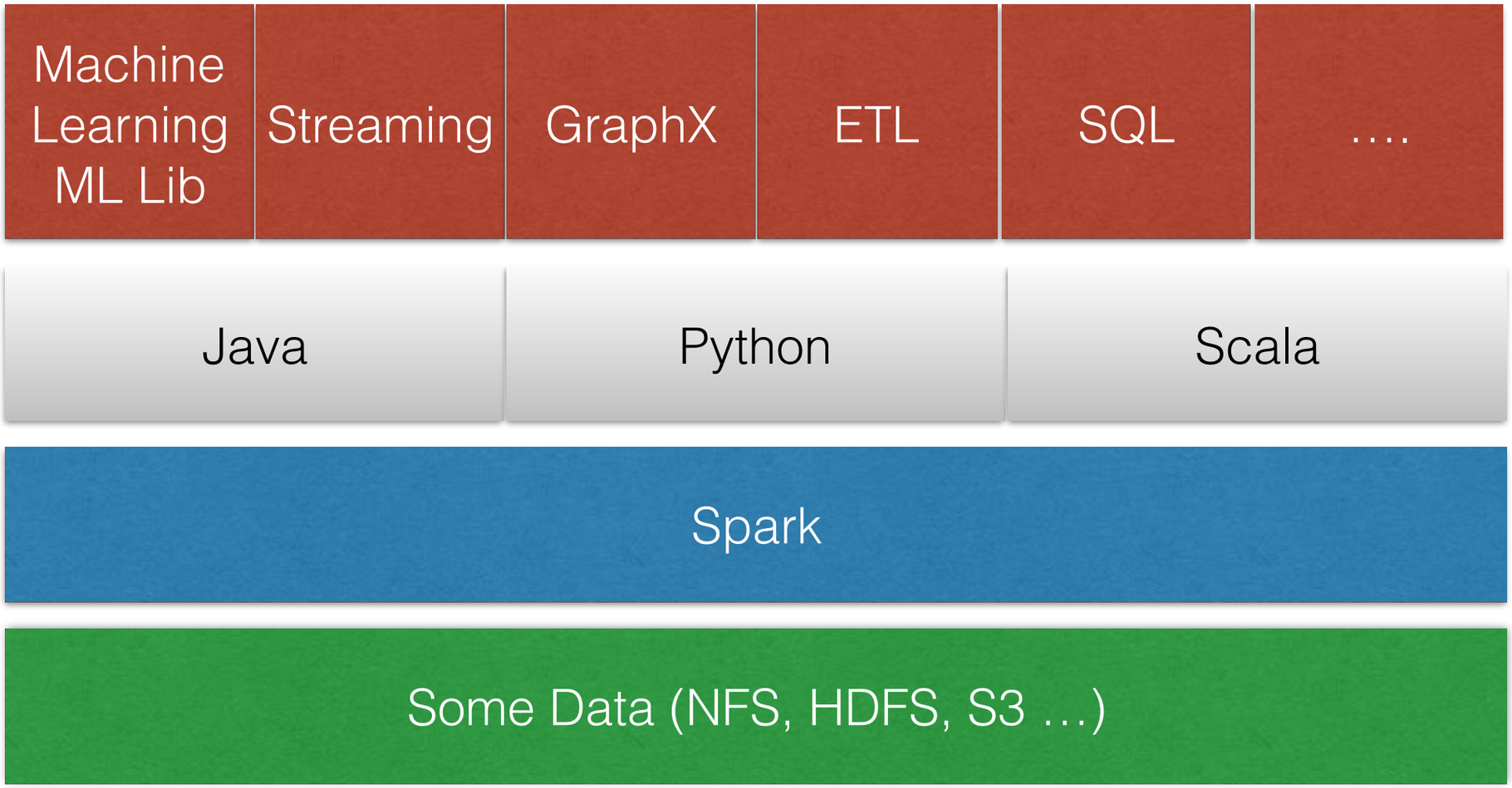
Limitations of MapReduce

- Multiple Stages still write to disk
- Stage Failures are not handled correctly
- Disk is the Major Bottleneck right now
- Good for ETL, But not for ML or Streaming Jobs

Next Generation

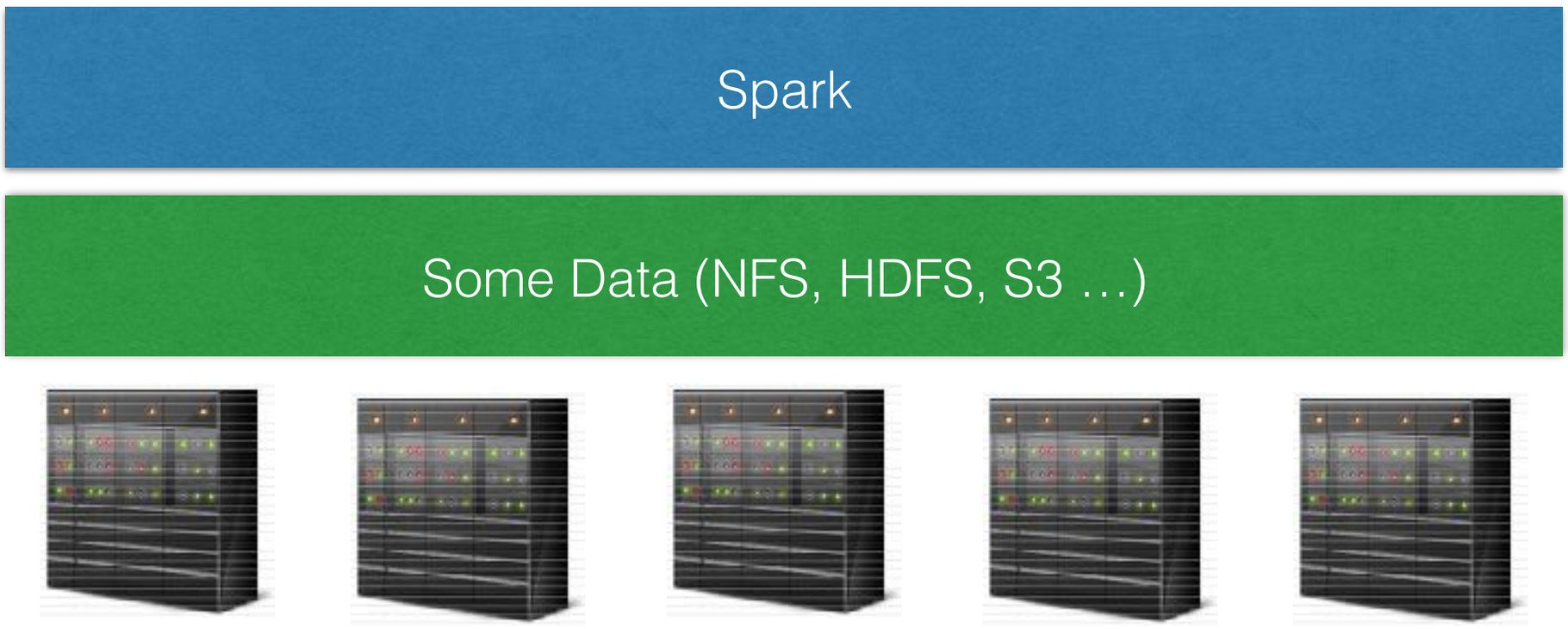
- 100's of frameworks built on top of HDFS
- Cloudera offers a one-click solution for $$$$
- Spark is the most upcoming project in the Apache group of projects
- Remember MapReduce had sorted 102 TB in 72 minutes using 2100 machines?

What is Spark?



What is Spark?

- Inherently distributed
  - Computation happens where the data resides



What is different from MapReduce

- Uses main memory for caching
- Dataset is partitioned and stored in RAM/Disk for iterative queries
- Large speedups for iterative operations

Spark Internals

The Init

- Creating a SparkContext
- It is Sparks' gateway to access the cluster
- In interactive mode. SparkContext is created as 'sc'

Lets begin by running `pyspark` on the command line

Spark Internals

The Key Idea - Resilient Distributed Datasets

- Basic abstraction in Spark
- Immutable collection of parallelised data
- Stored in RAM or Disk

Two Types of Operations on RDDs

Transformations	**Actions**
Take a RDD and produce another RDD	Compute the sequence of transformations to give back results

Spark Internals

Resilient Distributed Datasets

```
data_rdd = sc.textFile("file://...")
```

`sc` is SparkContext. A gateway for us to interact with the cluster

Spark Internals

Also applicable to Apache Pig

- Nothing actually happens when you perform transformations
- Spark just remembers what has to be done
- Everything gets evaluated only when you ask for it
- Lazy Evaluation

Spark Internals

Transformation Operations on RDDs

Map

```
def map_func(x):
    return x+1

rdd_2 =
rdd_1.map(func)
```

Filter

```
def fil_func(x):
    if x % 2 == 0:
      return True
    else:
      return False

rdd_2 =
rdd_1.filter(func)
```

Spark Internals

Transformation Operations on RDDs

- `map`
- `filter`
- `flatMap`
- `mapPartitions`
- `mapPartitionsWithIndex`
- `sample`
- `union`
- `intersection`
- `distinct`
- `groupByKey`

Why use Spark?

- Need to have chained operations. Not possible with MapReduce
- Gain time with in-memory querying
- Far easier to learn top down than bottom up

Where things are going

	Hadoop MR Record	**Spark Record**	**Spark 1 PB**
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**