Hadoop:
Distributed Processing of Big Data

Lecture 01

Marilson Campos

UCSC Extension - 2022

About Me

– Marilson Campos

- Principal Data Architect with 15+ years of experience.
- Background in Machine Learning and Search engines
- Designing Big Data systems since 2007

- Some relevant presentations
  - Hadoop Summit 2013
  - Apache Impala User Group - 2015
  - Strata 2016 – Use case @ Cloudera Booth
  - Impala spotlight – Cloudera 2016

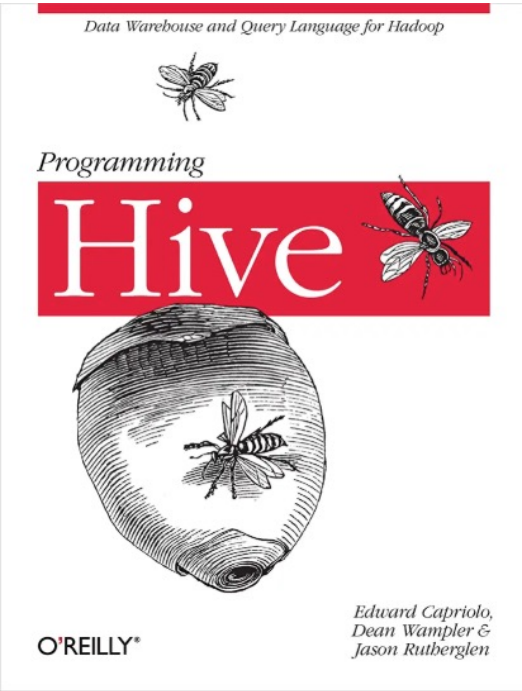
About the class

What we will cover

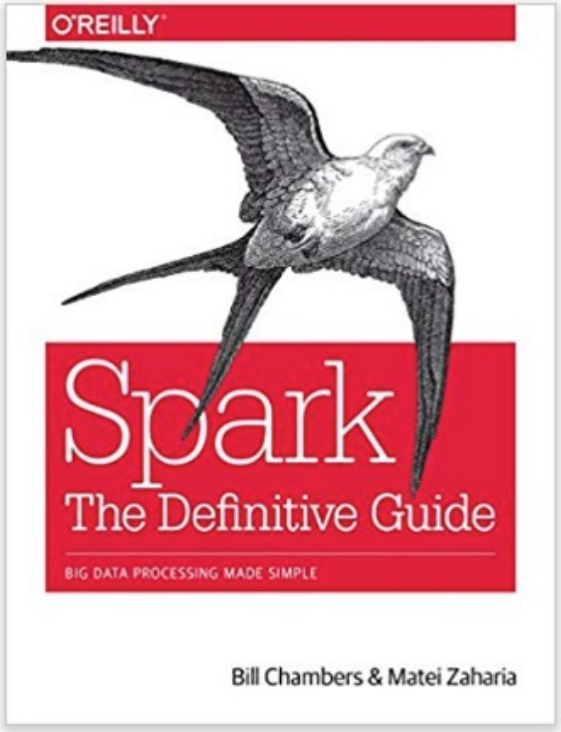
- Big Data architecture and its components
- Hadoop Map/Reduce model
- Spark programing.
- Big Data using Hive (SQL)
- Extending Hive
- Kafka & HBase
- Data pipeline design

Recommended Books

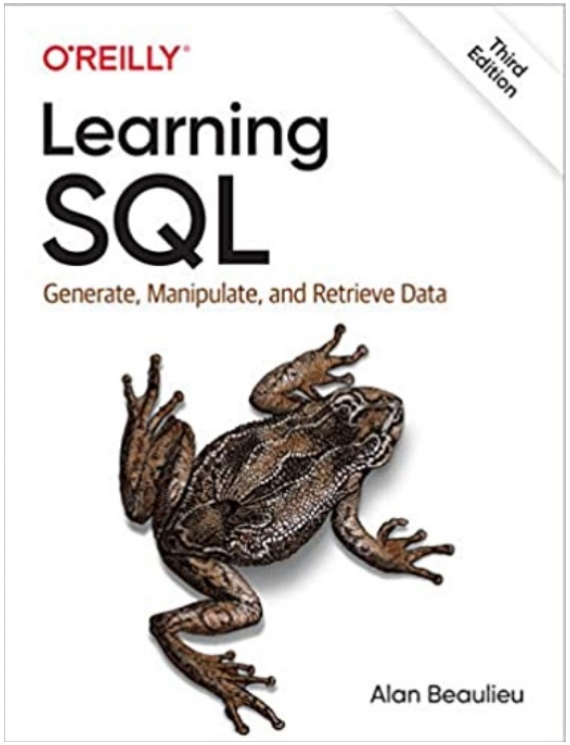
Hive



Spark



SQL



About the class

At the end of this class, I expect you to…

- Have an intuition about Big Data system design.
- Be able to write Big data programs.
- Know how to design a “data pipeline”.
- Continue to learn using other sources.

Class Material vs The Test

- We are covering some non-trivial concepts.
  - If you miss few advanced details, don't worry too much.
    Important: If you are lost, **please stop me!** And ask for clarification.
  - I want to make the class as interactive as possible.

- Problems in the labs are a lot harder than the ones in the test.
  - This is by design, so advanced students can also benefit from the class.

- To succeed in the test, you need to master the foundational knowledge about a Hadoop cluster, Hive, and Spark.

Tests

Test 1 – After 5th Lecture (Take home)

- 8 questions worth 5 points each.
  - 7 multiple choice questions
  - 1 question with one paragraph answer.

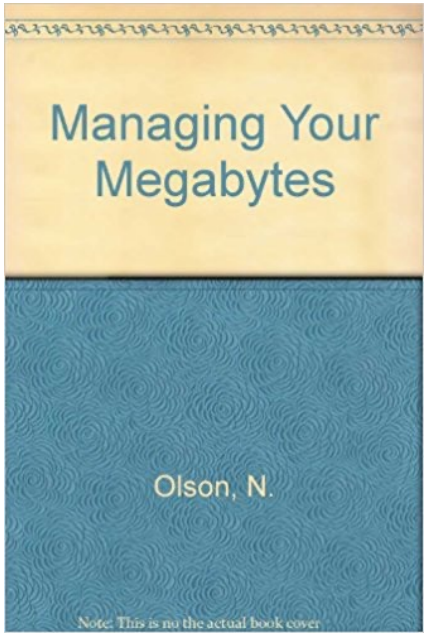
Test 2 – At 8th Lecture - Open book.

- 4 questions worth 15 points each.
  - Test your knowledge on perform operations like:
    - Load file into a Hadoop cluster
    - Create a table mapping to files loaded.
    - Perform transformations on datasets.
    - Query and extract results from cluster.

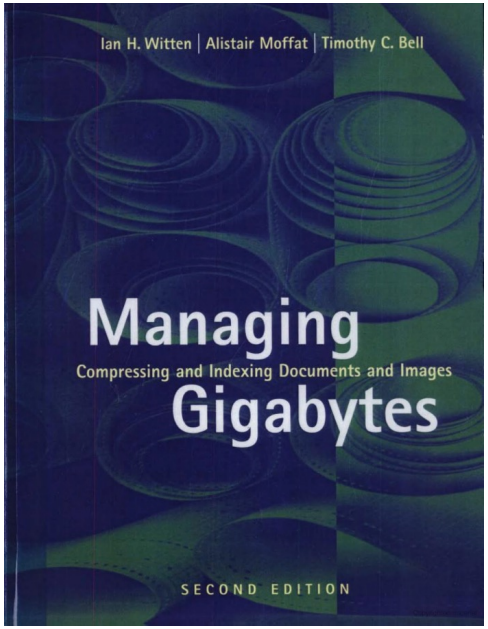
Big Data

Why Big Data?

- More information being stored digitally.
- Need to process larger datasets and unstructured data.
- Cannot process in traditional databases systems.



Published 1986



Published 1994

A New Architecture is needed

Data Transfer versus Storage Size

- 1990
  - 1Gb Storage
  - 4Mb per second data transfer
  - Time to read all data = **approx. 4 minutes**

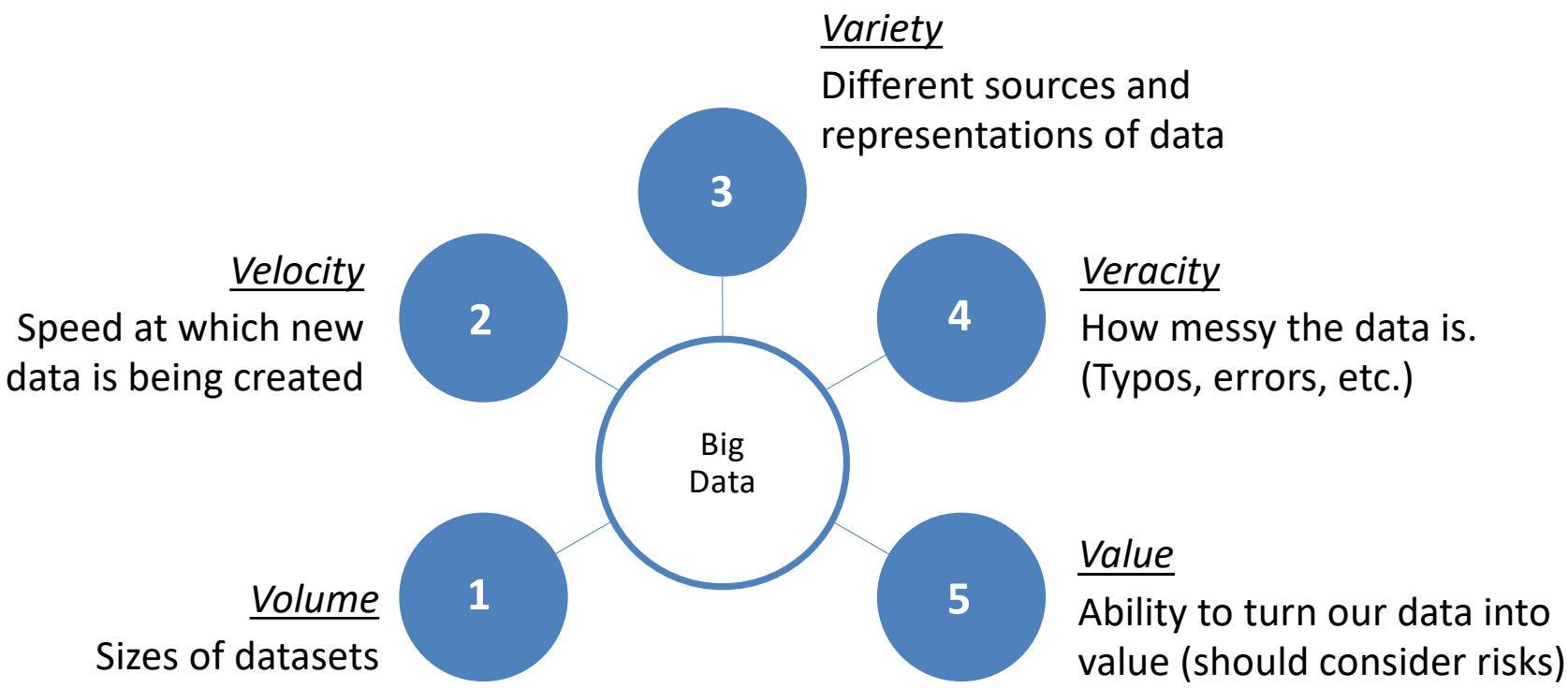
- 2019
  - 4,000Gb Storage (~4Tb)
  - 100Mb per second data transfer
  - Time to read all data = **approx. 11 hours.**

1Mb data transfer

Operation	X Speed factor
Processor Read	1000 x
Memory Read	120 x
SSD Read	20 x
Disk Read	**1**
Disk Seek	½ x
1Gpbs Net	½ x

Big Data

The 5 V's of Big Data



A New Architecture is Needed

High level abstraction

- Inverse of Virtualization
  - Virtualization = one machine appears to be many.
  - Big Data = many machines appear to be one.

Forces

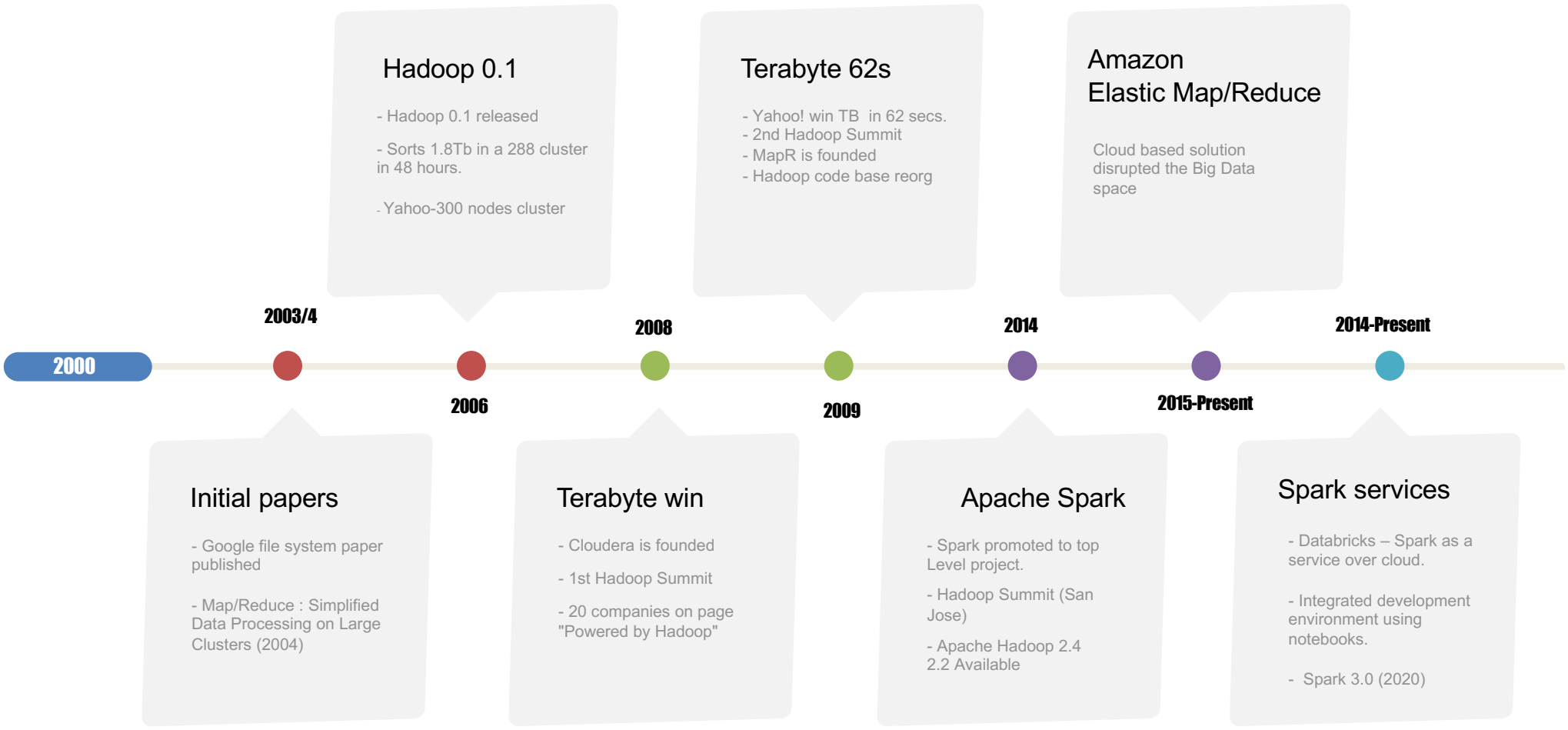
- Represent large datasets
  - Distributed
  - Memory, disk or combination.
- Process datasets
  - Use cluster resources to execute in parallel.
- Handle failure
  - Automatically recover from failure.

Initial approach

- Big Data distributed System
  - Many Computers processing in parallel.
  - Many disks available.
  - No transaction beyond files.
  - Single computers/disks should be able to fail with no impact.

Hadoop is born

The Hadoop Timeline



Hadoop is born

Big Data Distributions

- **In premises (Datacenter)**
  - **Cloudera**
    - Most popular in premises
    - Hybrid cloud offerings
  - **Hortonworks**
    - Most portable (Compatible with Windows)
    - Merged with Cloudera
  - **MapR**
    - Based on custom Linux filesystem
    - Acquired by HP.
- **Cloud Based**
  - **Amazon Elastic Map/Reduce (EMR)**
    - Cloud based simple to use
    - Current cloud-based leader.
  - **Microsoft**
    - Azure Big data
    - Part of Microsoft cloud offerings.
  - **Google**
    - Google compute engine
    - Part of Google other cloud offerings.

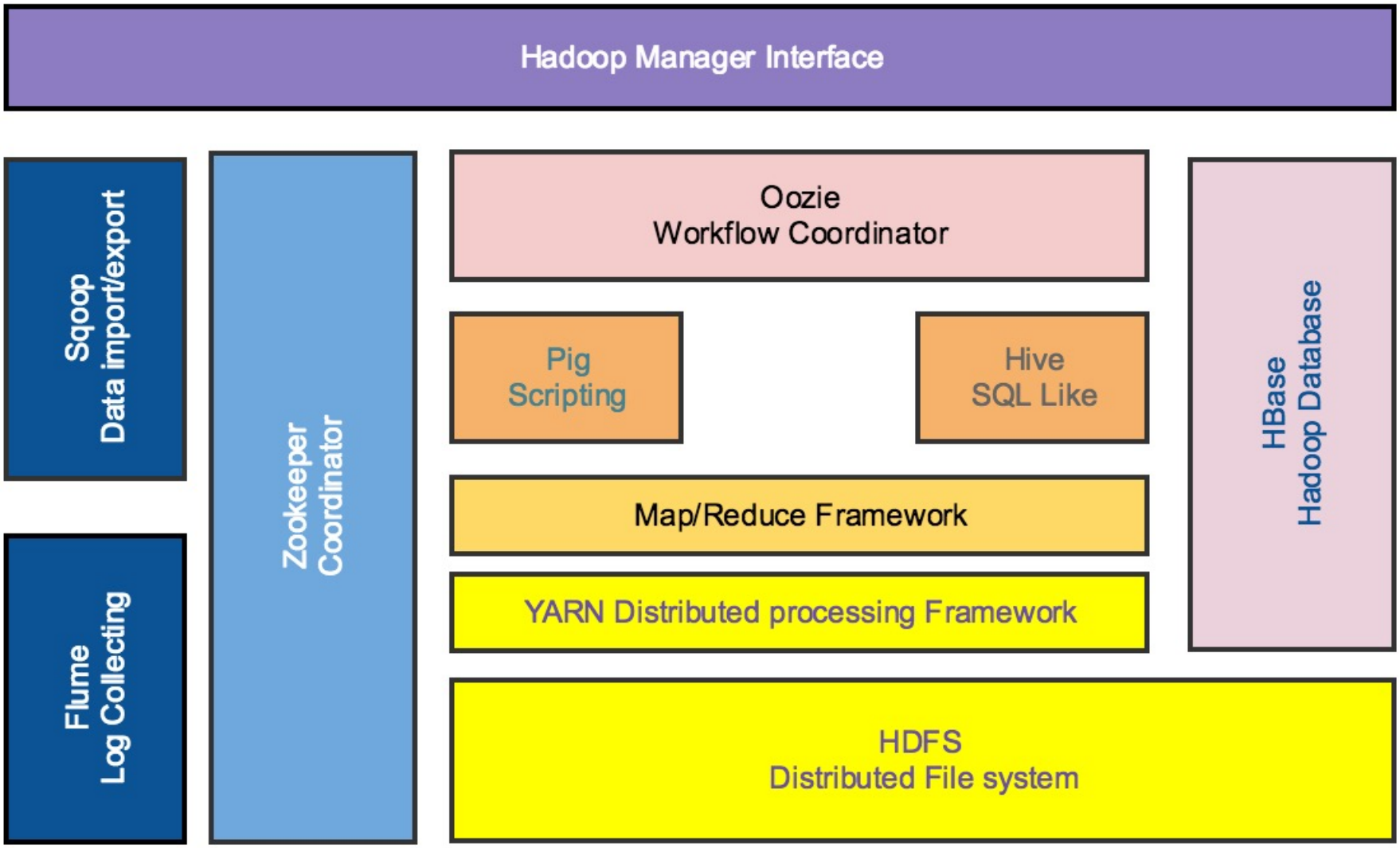
Hadoop is born

The Hadoop Cluster

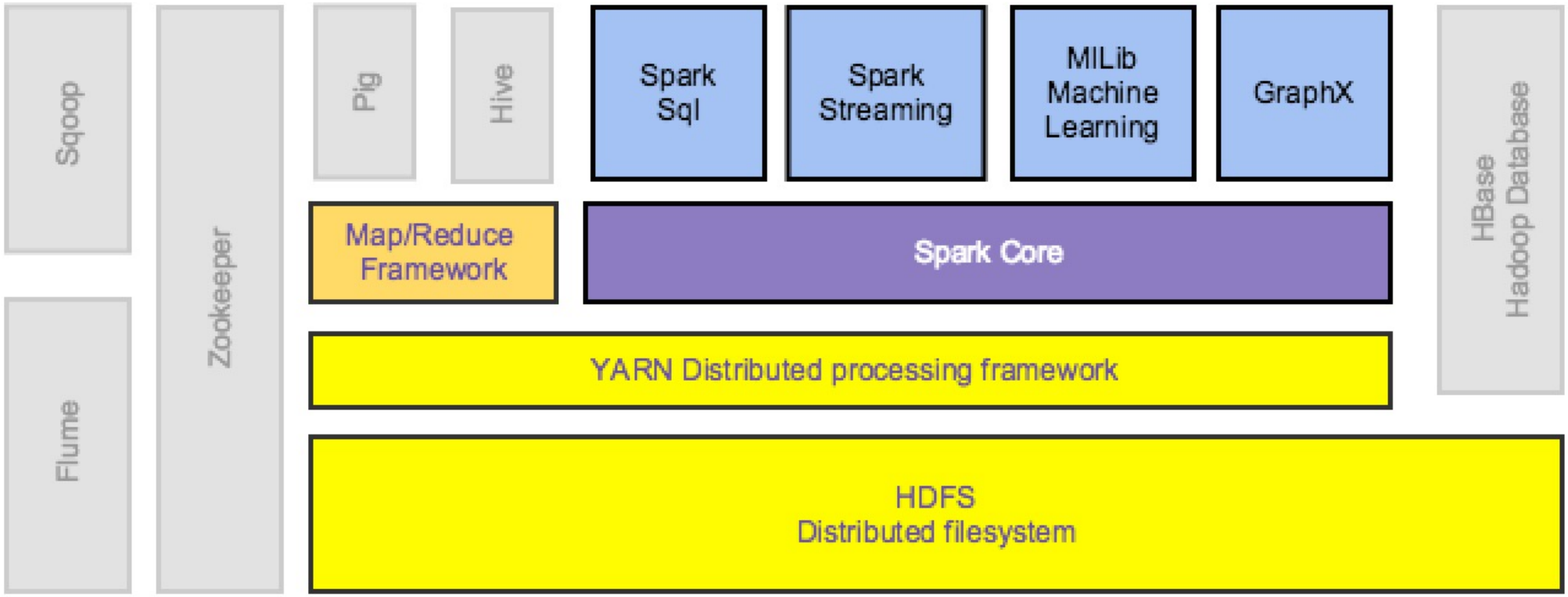
- 1000's of servers running
  - Each with dozens of disks.
  - High chance of single failure.
- Automated recovery
  - In case of a failure, the system needs to detect it and recover.
  - Cluster programs should not fail in this case.

Hadoop is born

The Hadoop Stack

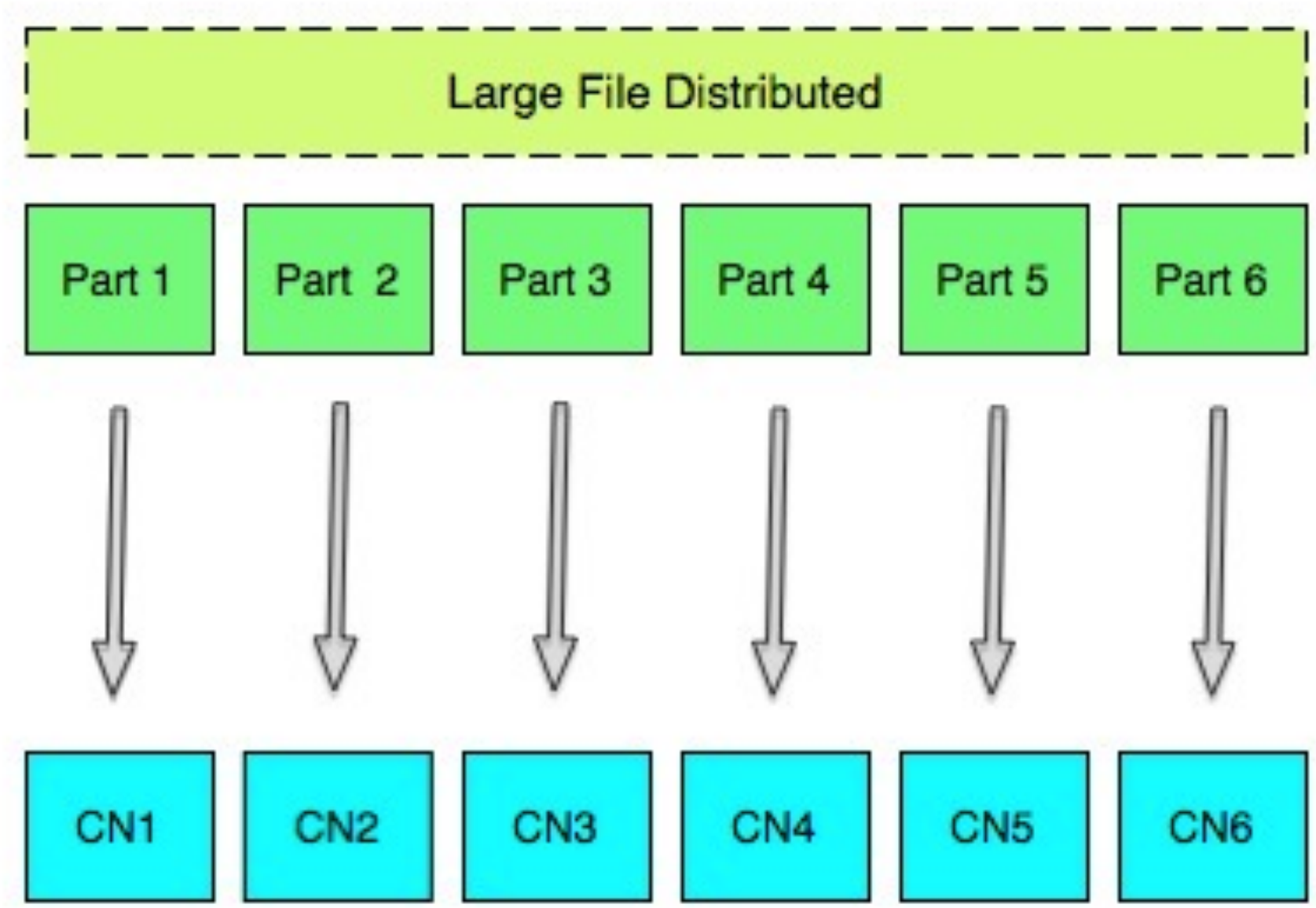


The Hadoop/Spark Stack

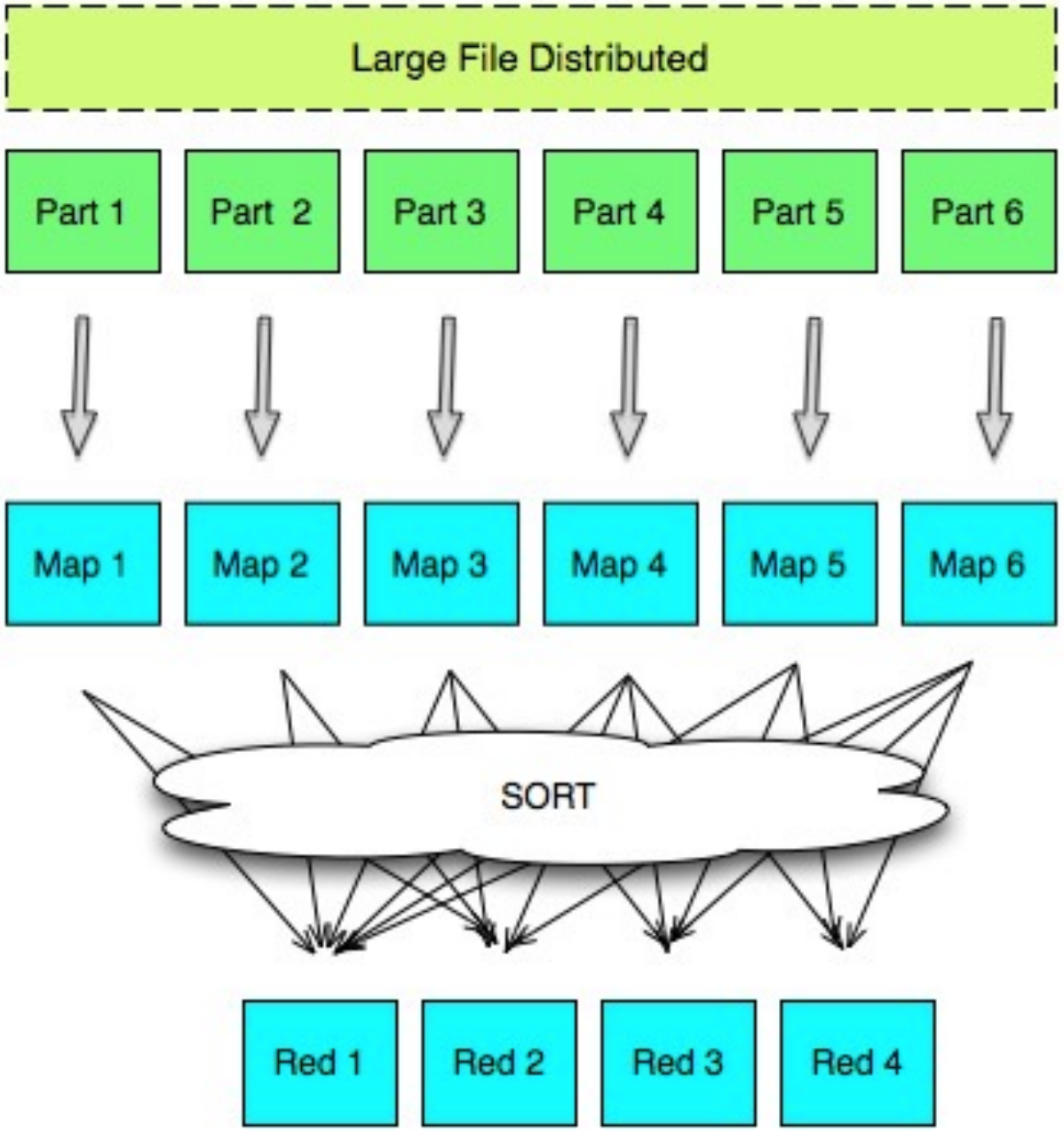


- Spark uses Hadoop core to execute jobs.
- Spark uses HDFS for storage.

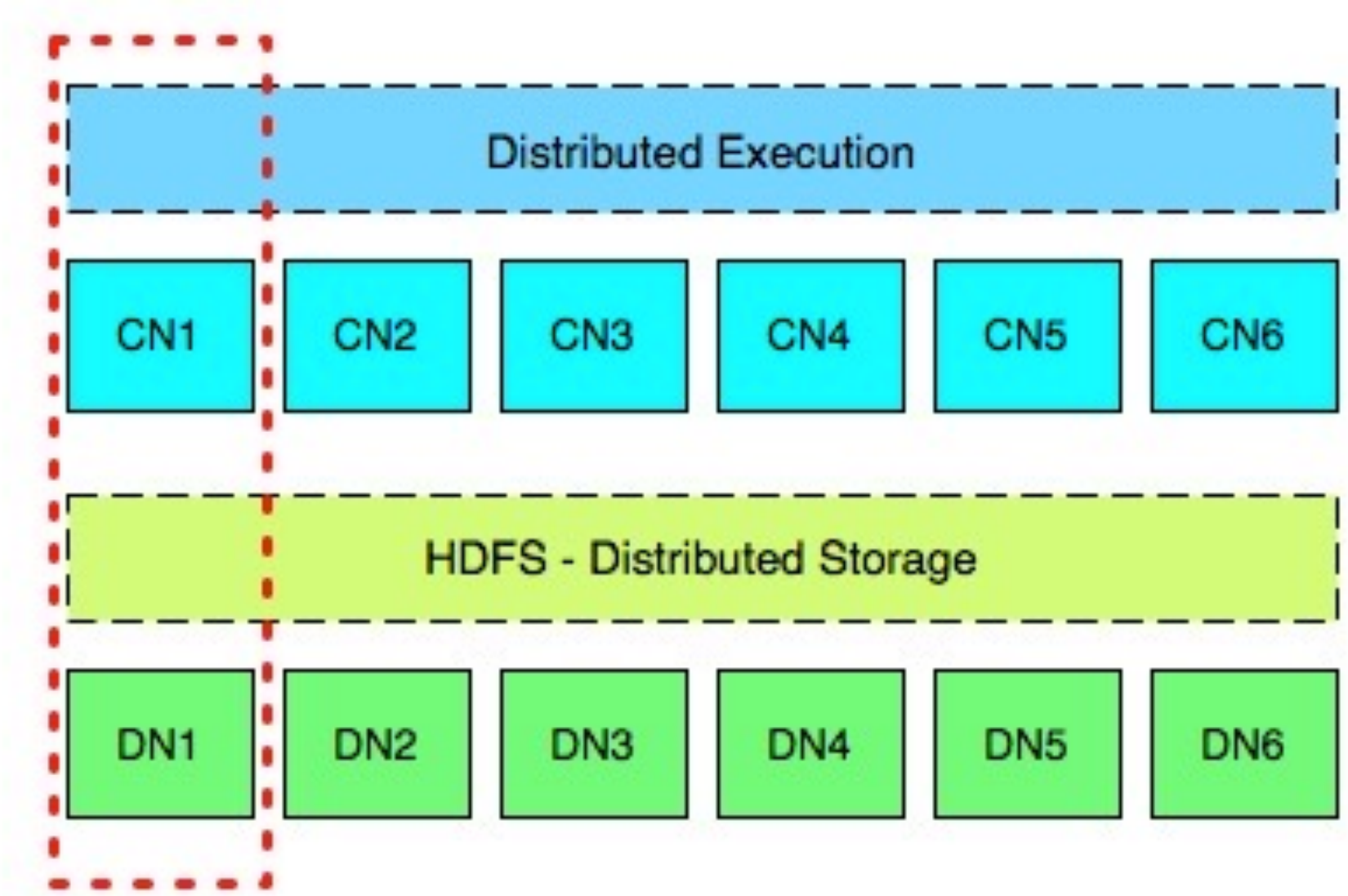
Distributed execution v0.1 – pure map



Distributed execution v1 – map/reduce



Distributed execution – deployment



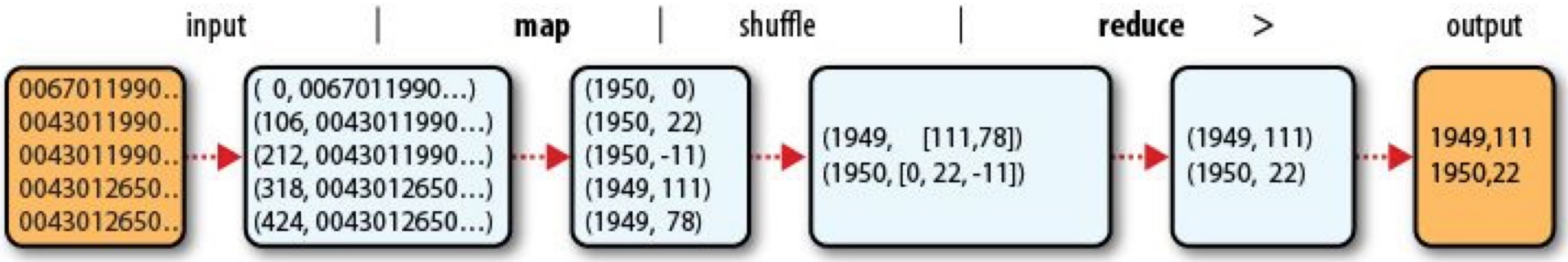
Hadoop is born

Hadoop Map/Reduce x Spark

- Low level API
- Designed for batch processing (slower starting time)
- Long running process
- Reads and writes data to and from disk many times.

- Higher level API
- Tries to use and keep data in memory.
- Supports batch and near real-time.
  (faster starting time)

Map/Reduce processing



Data flow:

- Input
  - Map (transformation)
    - Shuffle (sort)
      - Reduce
        - Output

The Mapper

```
public class MaxTemperatureMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

The Reducer

```
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
  extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values,
      Context context)
      throws IOException, InterruptedException {

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

The Driver

```
public static void main(String[] args) throws Exception {
  if (args.length != 2) {
    System.err.println("Usage: MaxTemperature <input path> <output path>");
    System.exit(-1);
  }

  Job job = new Job();
  job.setJarByClass(MaxTemperature.class);
  job.setJobName("Max temperature");

  FileInputFormat.addInputPath(job, new Path(args[0]));
  FileOutputFormat.setOutputPath(job, new Path(args[1]));

  job.setMapperClass(MaxTemperatureMapper.class);
  job.setReducerClass(MaxTemperatureReducer.class);

  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);

  System.exit(job.waitForCompletion(true) ? 0 : 1);
 }
}
```

Hadoop Map/Reduce - Run output 1/2

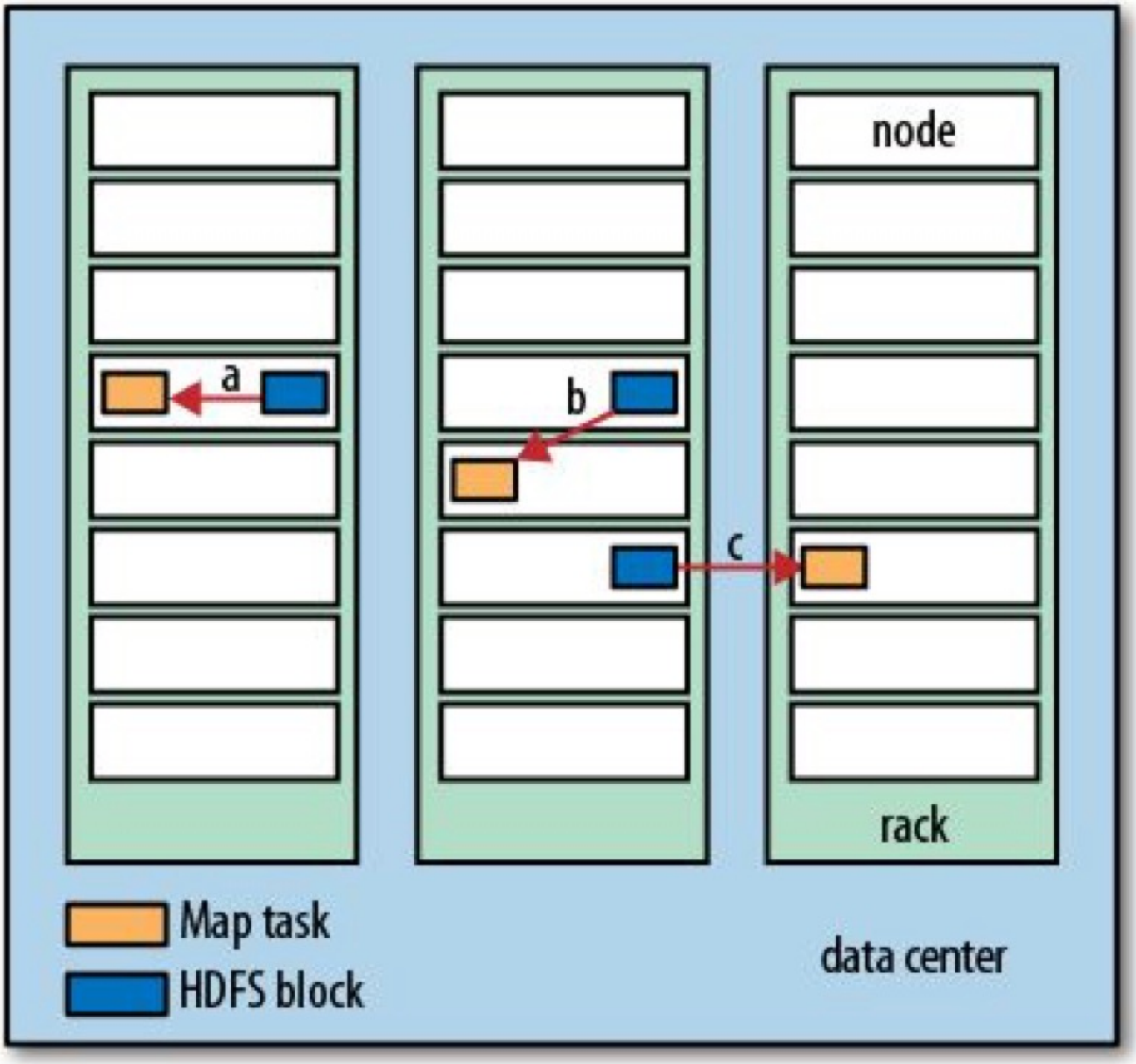
```
12/02/04 11:50:41 WARN util.NativeCodeLoader: Unable to load native-hadoop library
for your platform... using builtin-java classes where applicable
12/02/04 11:50:41 WARN mapred.JobClient: Use GenericOptionsParser for parsing the
arguments. Applications should implement Tool for the same.
12/02/04 11:50:41 INFO input.FileInputFormat: Total input paths to process : 1
12/02/04 11:50:41 INFO mapred.JobClient: Running job: job_local_0001
12/02/04 11:50:41 INFO mapred.Task:  Using ResourceCalculatorPlugin : null
12/02/04 11:50:41 INFO mapred.MapTask: io.sort.mb = 100
12/02/04 11:50:42 INFO mapred.MapTask: data buffer = 79691776/99614720
12/02/04 11:50:42 INFO mapred.MapTask: record buffer = 262144/327680
12/02/04 11:50:42 INFO mapred.MapTask: Starting flush of map output
12/02/04 11:50:42 INFO mapred.MapTask: Finished spill 0
12/02/04 11:50:42 INFO mapred.Task: Task:attempt_local_0001_m_000000_0 is done. And i
s in the process of commiting
12/02/04 11:50:42 INFO mapred.JobClient:  map 0% reduce 0%
12/02/04 11:50:44 INFO mapred.LocalJobRunner:
12/02/04 11:50:44 INFO mapred.Task: Task 'attempt_local_0001_m_000000_0' done.
12/02/04 11:50:44 INFO mapred.Task:  Using ResourceCalculatorPlugin : null
12/02/04 11:50:44 INFO mapred.LocalJobRunner:
12/02/04 11:50:44 INFO mapred.Merger: Merging 1 sorted segments
12/02/04 11:50:44 INFO mapred.Merger: Down to the last merge-pass, with 1 segments
left of total size: 57 bytes
12/02/04 11:50:44 INFO mapred.LocalJobRunner:
12/02/04 11:50:45 INFO mapred.Task: Task:attempt_local_0001_r_000000_0 is done. And
```

Hadoop Map/Reduce - Run output 2/2

```
is in the process of commiting
12/02/04 11:50:45 INFO mapred.LocalJobRunner:
12/02/04 11:50:45 INFO mapred.Task: Task attempt_local_0001_r_000000_0 is allowed to
commit now
12/02/04 11:50:45 INFO output.FileOutputCommitter: Saved output of task 'attempt_local
_0001_r_000000_0' to output
12/02/04 11:50:45 INFO mapred.JobClient:  map 100% reduce 0%
12/02/04 11:50:47 INFO mapred.LocalJobRunner: reduce > reduce
12/02/04 11:50:47 INFO mapred.Task: Task 'attempt_local_0001_r_000000_0' done.
12/02/04 11:50:48 INFO mapred.JobClient:  map 100% reduce 100%
12/02/04 11:50:48 INFO mapred.JobClient: Job complete: job_local_0001
12/02/04 11:50:48 INFO mapred.JobClient: Counters: 17
12/02/04 11:50:48 INFO mapred.JobClient:   File Output Format Counters
12/02/04 11:50:48 INFO mapred.JobClient:     Bytes Written=29
12/02/04 11:50:48 INFO mapred.JobClient:   FileSystemCounters
12/02/04 11:50:48 INFO mapred.JobClient:     FILE_BYTES_READ=357503
12/02/04 11:50:48 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=425817
12/02/04 11:50:48 INFO mapred.JobClient:   File Input Format Counters
12/02/04 11:50:48 INFO mapred.JobClient:     Bytes Read=529
12/02/04 11:50:48 INFO mapred.JobClient:   Map-Reduce Framework
12/02/04 11:50:48 INFO mapred.JobClient:     Map output materialized bytes=61
12/02/04 11:50:48 INFO mapred.JobClient:     Map input records=5
12/02/04 11:50:48 INFO mapred.JobClient:     Reduce shuffle bytes=0
12/02/04 11:50:48 INFO mapred.JobClient:     Spilled Records=10
12/02/04 11:50:48 INFO mapred.JobClient:     Map output bytes=45
12/02/04 11:50:48 INFO mapred.JobClient:     Total committed heap usage (bytes)=36923
8016
12/02/04 11:50:48 INFO mapred.JobClient:     SPLIT_RAW_BYTES=129
12/02/04 11:50:48 INFO mapred.JobClient:     Combine input records=0
12/02/04 11:50:48 INFO mapred.JobClient:     Reduce input records=5
12/02/04 11:50:48 INFO mapred.JobClient:     Reduce input groups=2
12/02/04 11:50:48 INFO mapred.JobClient:     Combine output records=0
12/02/04 11:50:48 INFO mapred.JobClient:     Reduce output records=2
12/02/04 11:50:48 INFO mapred.JobClient:     Map output records=5
```

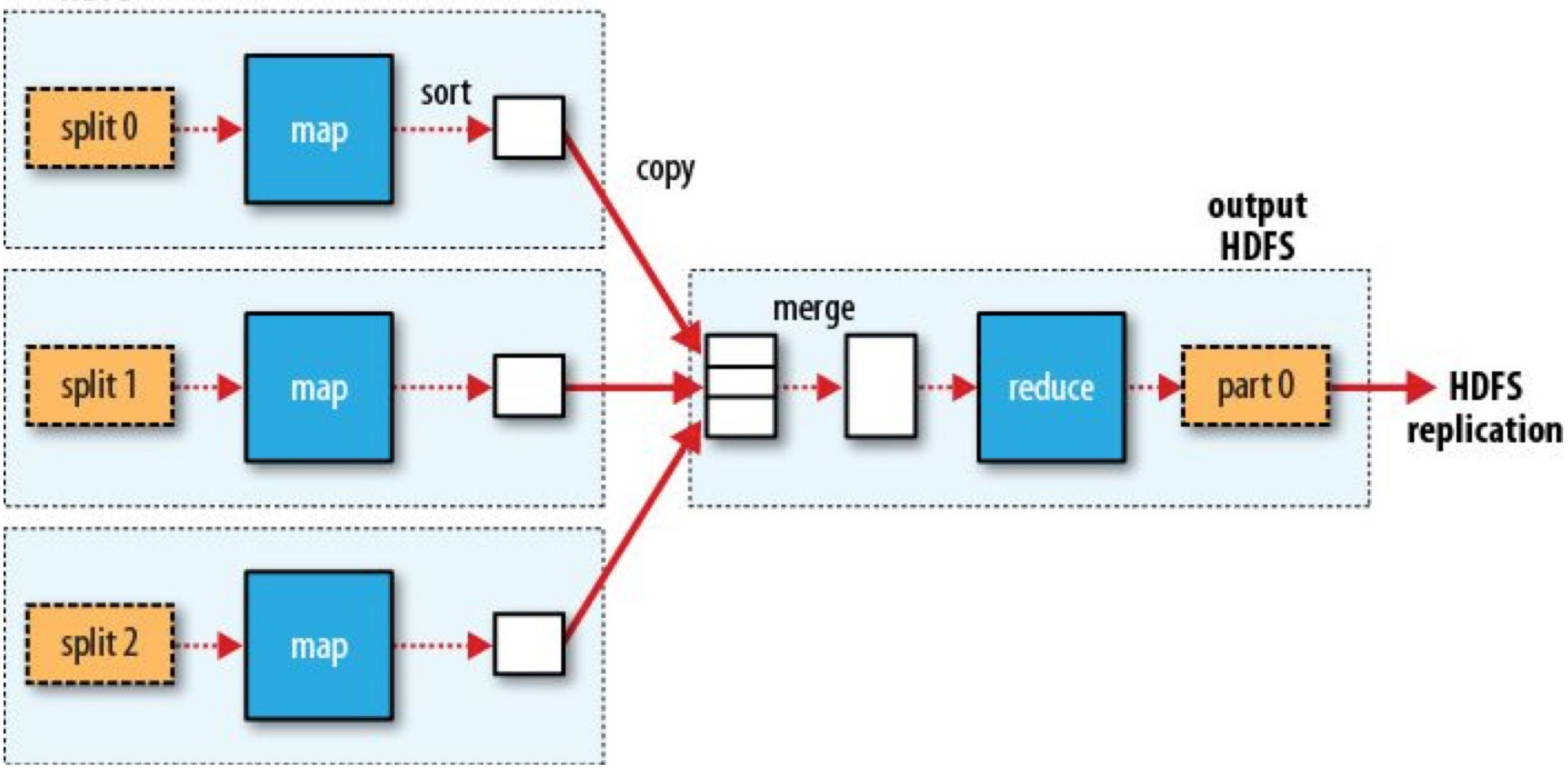
Hadoop Map/Reduce

Send the code to the data!



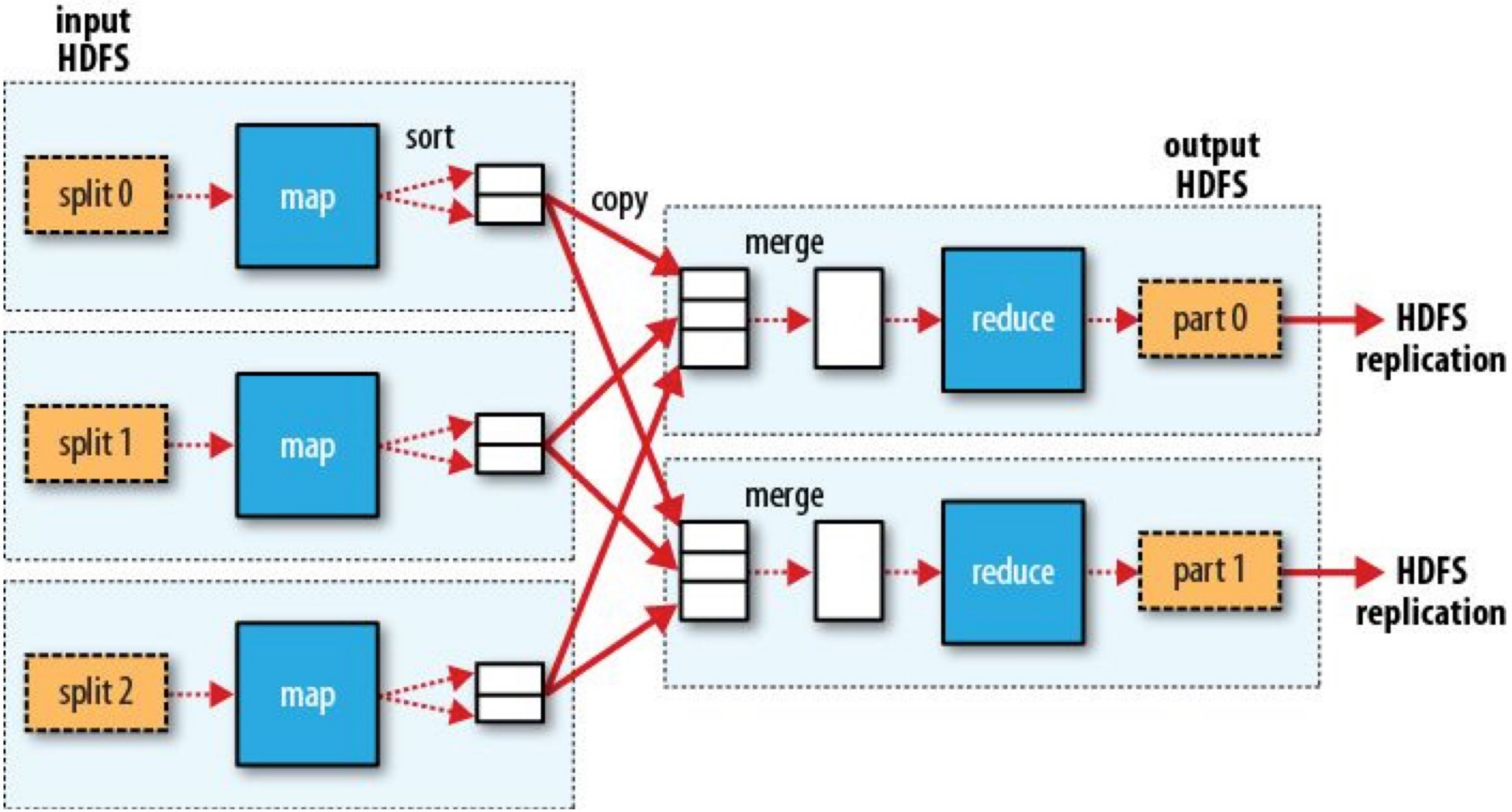
Hadoop Map/Reduce

MR Splits and dataflow 1 reducer

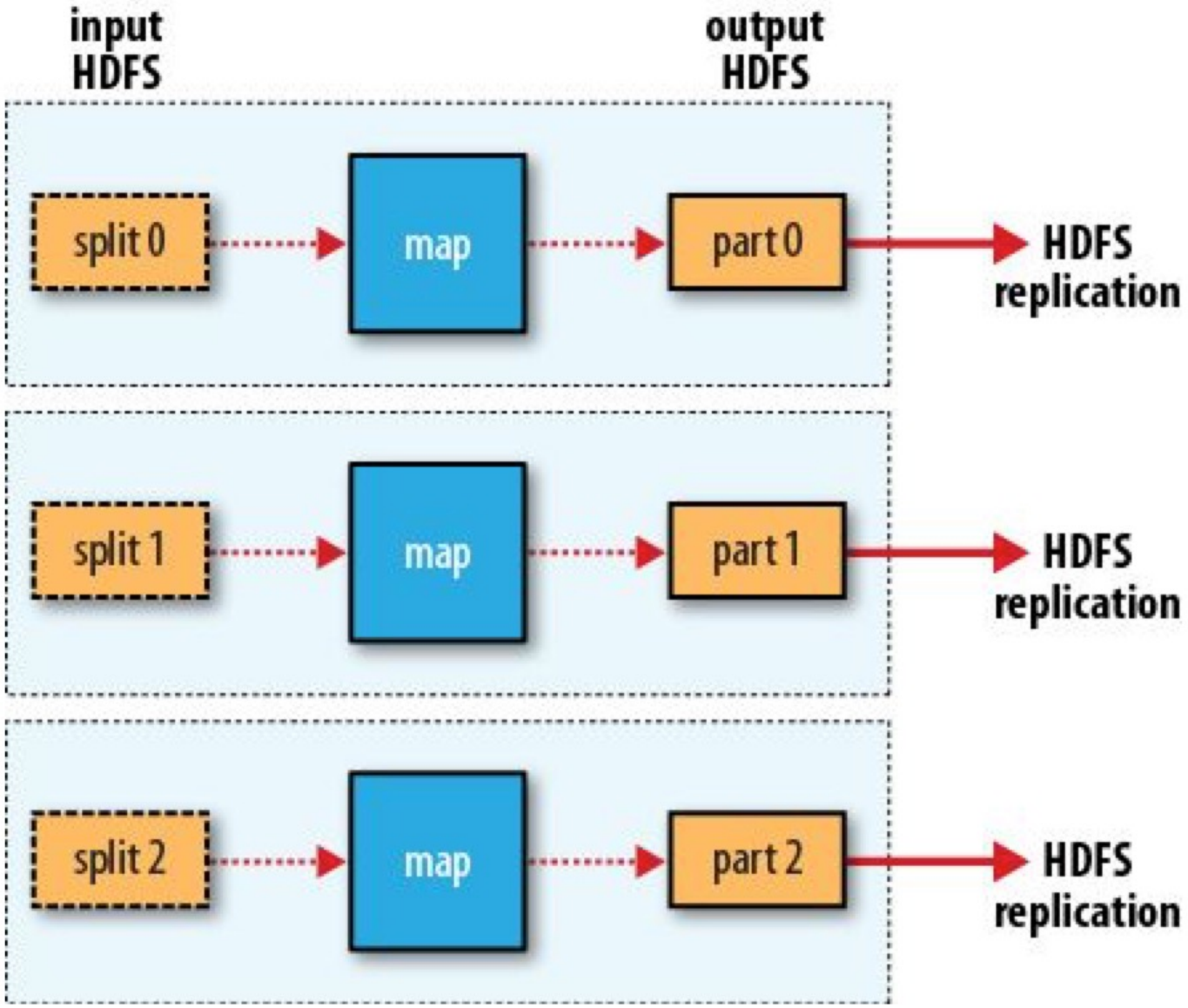


Hadoop Map/Reduce

MR Splits and dataflow multiple reducers



MR Splits and dataflow Zero reducers



The Combiner

- Calculates Intermediary results

- No guarantees if it will run or even how many times it will runs.

- Optimization feature

The Driver with Combiner

```
public class MaxTemperatureWithCombiner {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperatureWithCombiner <input path> " +
          "<output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperatureWithCombiner.class);
    job.setJobName("Max temperature");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setCombinerClass(MaxTemperatureReducer.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

Hadoop Map/Reduce

Different integrations

- Map/Reduce Api -> JVM language (Java, Scala)

- Hadoop M/R Streaming -> Unix Stdin/Stdout.

- Hadoop pipes -> Unix Sockets

HDFS – Hadoop File Systems

Big Data Storage

- Unix like filesystem

- Access rights identical to Linux

- No support to updates

- Self healing capabilities.

HDFS – Hadoop File Systems

Big Data Storage

- Very large file support

- Optimized for continuous ”reads”.

- High throughput at the expense of latency

- Single writer, no support to multiple writers.

HDFS – Hadoop File Systems

HDFS Concepts

- Blocks
  - Atomic unit of storage
- Namenode
  - the master daemon for storage
  - Stores the metadata
  - 1 Active
  - 1 Standby
- Journal Nodes
  - Record changes on filesystem
  - Responsible for metadata updates on standby namenode.
- Datanodes
  - Responsive for reading/writing data.

HDFS – Big Data Storage

- Namenodes

- Journal Nodes.

- Datanodes

Spark

Spark lab Preparation

Signup for Databricks

Search for ‘Databricks Community Edition: Login’ and signup for the community edition.

Note: Do not create a trial account as it will expire and will require for you to pay for it.

Q & A