"If you torture the data long enough, it will confess"

Ronald Coase, Professor at UChicago, 1981

Lecture C1
Batch Data Processing

CS205: Computing Foundations for Computational Science
Dr. Ignacio M. Llorente
Spring Term 2021

HARVARD
School of Engineering and Applied Sciences

IACS
INSTITUTE FOR APPLIED COMPUTATIONAL SCIENCE
AT HARVARD UNIVERSITY

Before We Start

Where We Are

Computing Foundations for Computational and Data Science

How to use modern computing platforms in solving scientific problems

Intro: Large-Scale Computational and Data Science

A. Parallel Processing Fundamentals

B. Parallel Computing

C. Parallel Data Processing

- **C1. Batch Data Processing**
- C2. Dataflow Processing
- C3. Stream Data Processing

Wrap-Up: Advanced Topics

CS205: Contents

APPLICATION SOFTWARE

Application Parallelism | Program Design

Application Software

BIG COMPUTE

OpenACC | Optimization

OpenMP | MPI

Spark

Map-Reduce

Programming Model

BIG DATA

Slurm

Platform

Hadoop

Architecture

amazon web services

ODYSSEY HARVARD FAS RESEARCH COMPUTING

FAS RC

Cloud Computing | Computing Cluster

Before We Start

Where We Are

Concepts → Programming → Platform

Batch Data Processing => MapReduce

3/18 **Lecture C1** Batch Data Processing (Quiz & Reading)	**3/23** Hands-on H4 MapReduce Programming	**Lab** Lab I8 MapReduce Hadoop Cluster

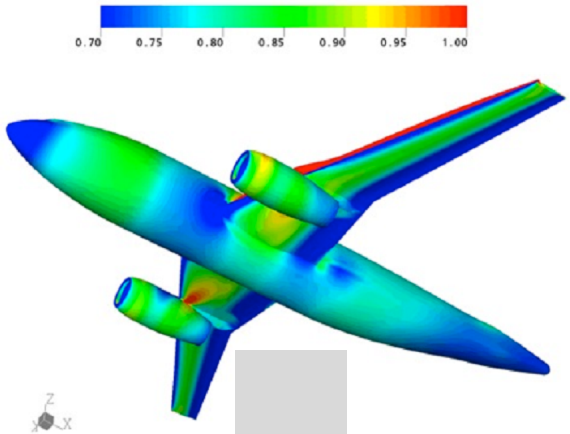
Dataflow Processing => Spark

3/25 Lecture C2 Dataflow Processing (Quiz & Reading)	**3/30** Hands-on H5 Spark Programming	**Lab** Lab I9 Spark Single Node	**Lab** Lab I10 Spark Cluster

Context

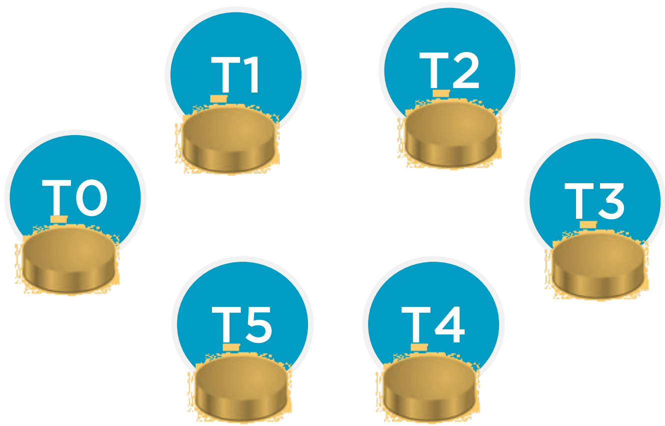
Big Compute vs Big data

"Big" Compute



Compute-intensive

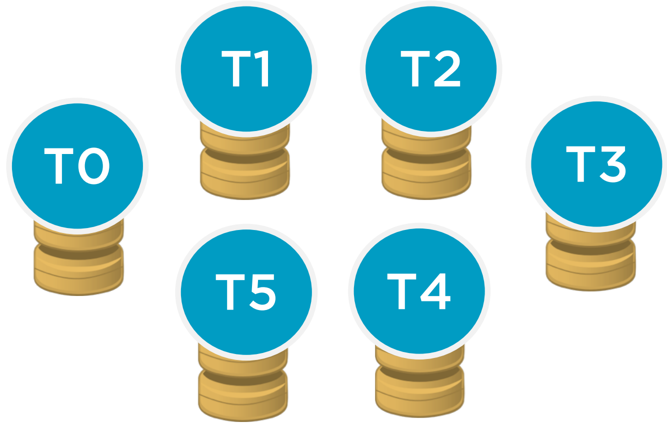
Bringing data to compute



Big Data

Data-intensive

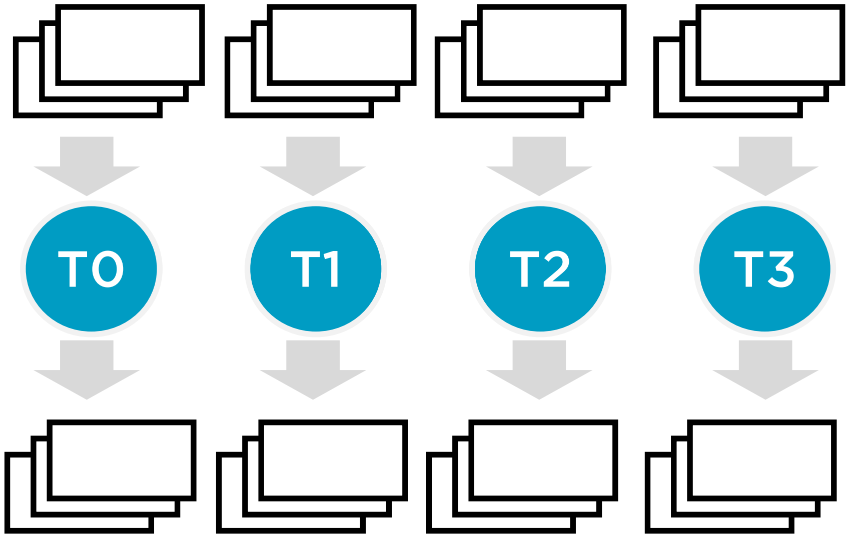
Bringing compute to data



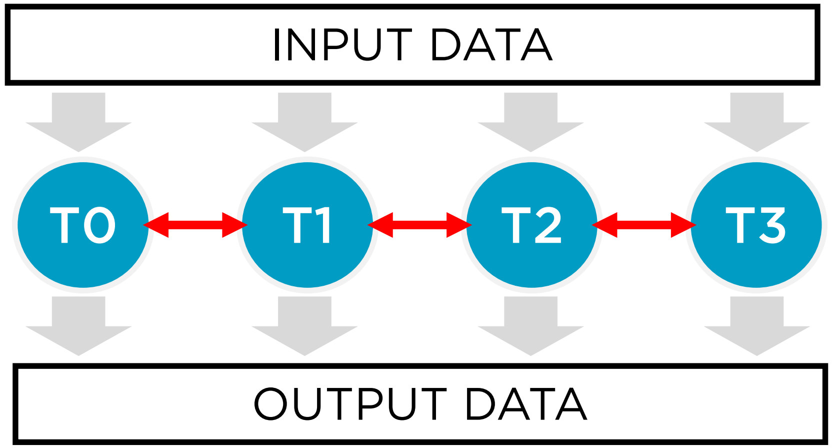
Context

Big Compute

Paradigm	Independent parallel tasks that are performed simultaneously to address a particular part of the problem
Challenge	Decompose the application into tasks and define their communication and synchronization
Bottleneck	CPU
Input data	Gigabyte-scale to describe initial conditions
Programming	OpenMP, OpenACC and MPI



HTC

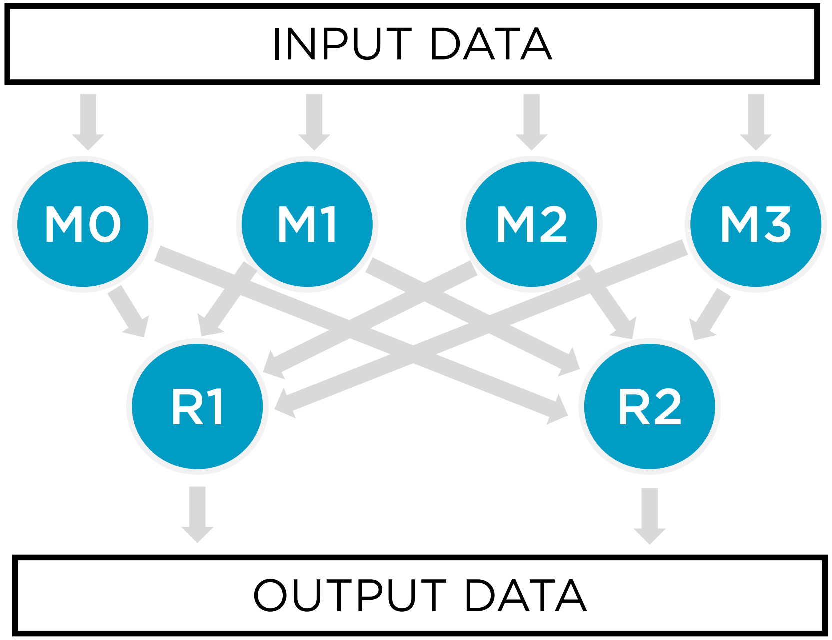


HPC

Context

Big Data

Paradigm	Same task is applied to large volumes of data
Challenge	Partition the data into multiple segments and the subsequent combination of the intermediate results in multiple stages
Bottleneck	Storage
Input data	Far beyond gigabyte-scale: datasets are commonly on the order of tens, hundreds, or thousands of terabytes
Programming	MapReduce, Spark



Hands-on Examples

Requirements

1. Unix-like shell (Linux, Mac OS or Windows/Cygwin)

2. Python installed

3. Download example python codes

https://harvard-iacs.github.io/2020-CS205/lectures/C1/

Roadmap

Batch Data Processing

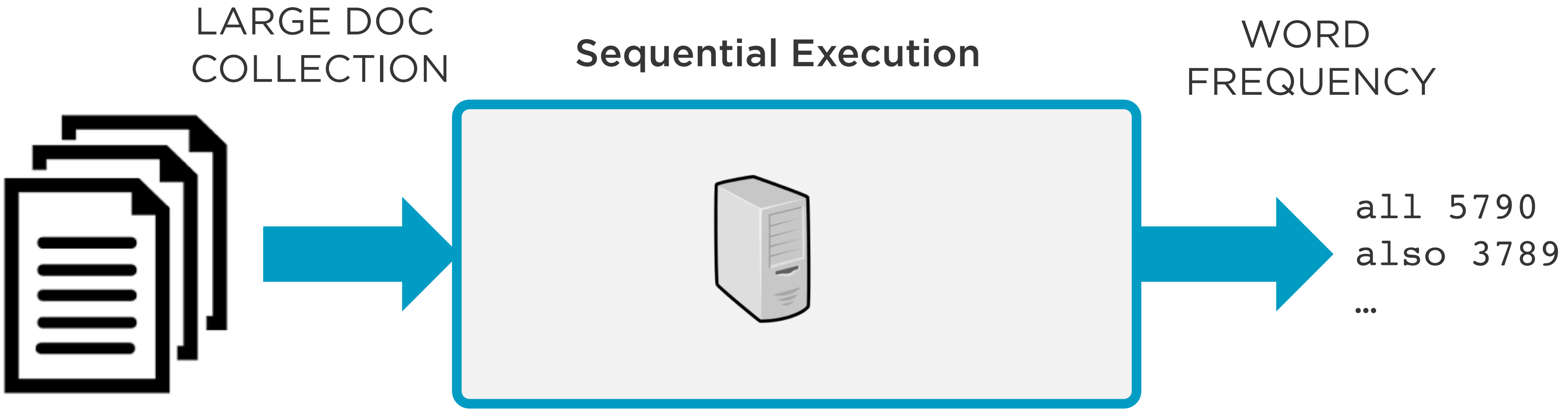
Why Is Big Data Processing Different?

The MapReduce Programming Model

The Hadoop Processing Framework

Why Is Big Data Processing Different?

The WordCount ~~HelloWorld~~ Example



`$ counter.py < input.txt`

counter.py

```python
#!/usr/bin/python

import sys
import re

sums = {}

for line in sys.stdin:
    line = re.sub( r'^\W+|\W+$', '', line )
    words = re.split(r'\W+', line)

    for word in words:
        word = word.lower()
        sums[word] = sums.get( word, 0 ) + 1

print sums
```

Implementation

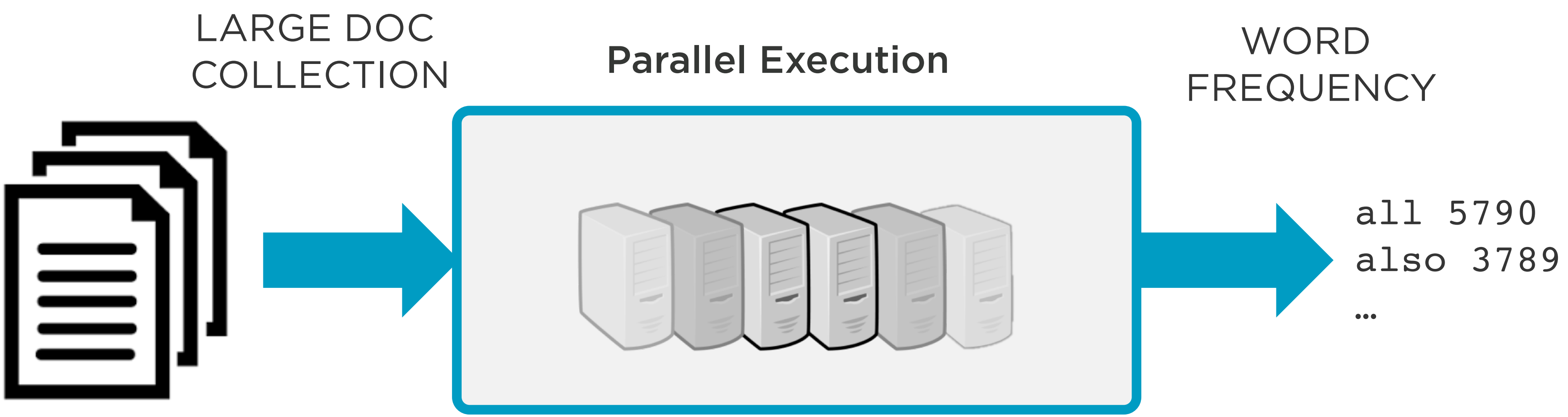
- Centralized **key-value data structure**, hash table (dictionary `sums`) to keep track of counts

Scalability Limitations

- Compute-bound: Limited by the speed of the system
- Memory-bound: Limited by the memory size of the system

Why Is Big Data Processing Different?

The WordCount Example



Is the `counter` application limited by the CPU?

How would you develop a parallel version of the `counter` application?

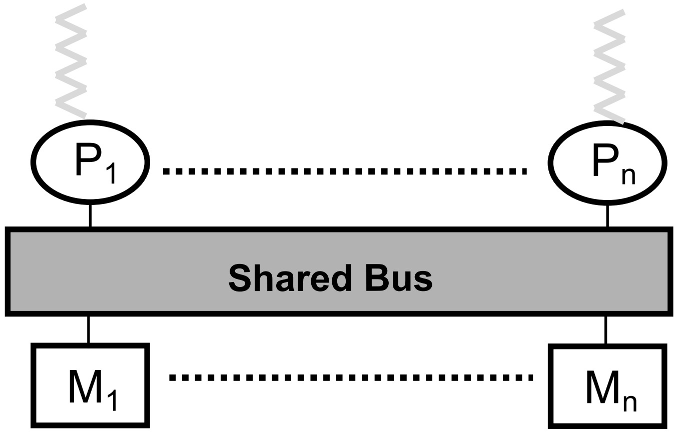
Why Is Big Data Processing Different?

The WordCount Example

Shared Memory (OpenMP)

```
#!/usr/bin/python

import sys
import re

sums = {}

for line in sys.stdin:
    line = re.sub( r'^\W+|\W+$', '', line )
    words = re.split(r'\W+', line)

    for word in words:
        word = word.lower()
        sums[word] = sums.get( word, 0 ) + 1

print sums
```

OMP Parallel



Implementation

- Each thread processes a part of each doc
- Single parallel instance of the `counter` code and **shared data structure between threads**

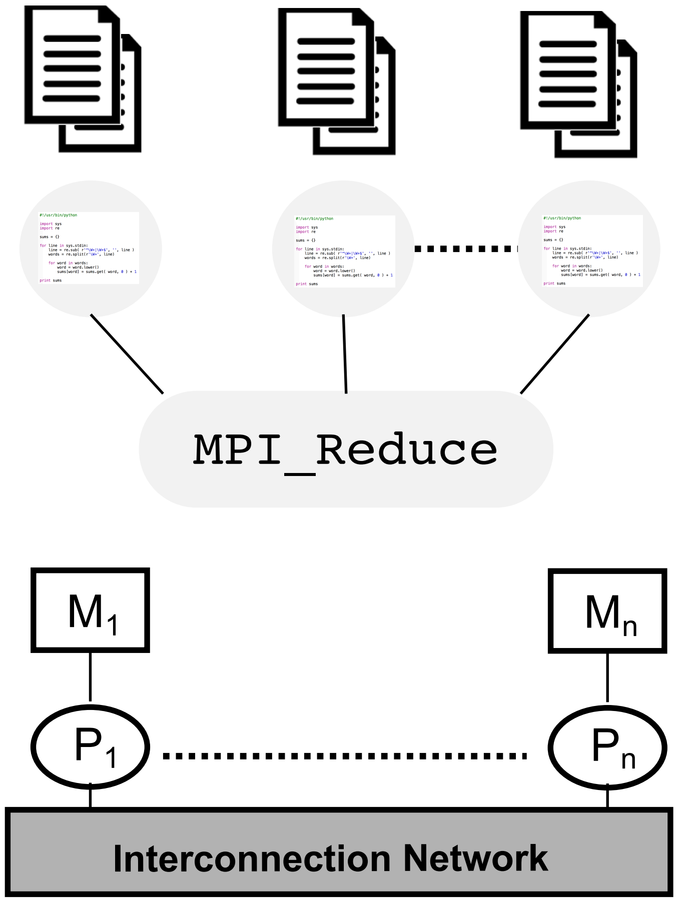
Scalability Limitations

- Memory access synchronization to shared data structure
- Shared memory architecture (bus bottleneck)

Why Is Big Data Processing Different?

The WordCount Example

Distributed Memory (MPI)



Implementation

- Each node processes a subset in parallel
- Each node executes a sequential instance of the `counter` code and keeps its own local data structure
- Big final reduction operation for the complete data structure

Scalability Limitations

- Communication-bound: Cost of final aggregation with reduction of all the data structure
- Memory-bound: Limited by the memory size of each node

Why Is Big Data Processing Different?

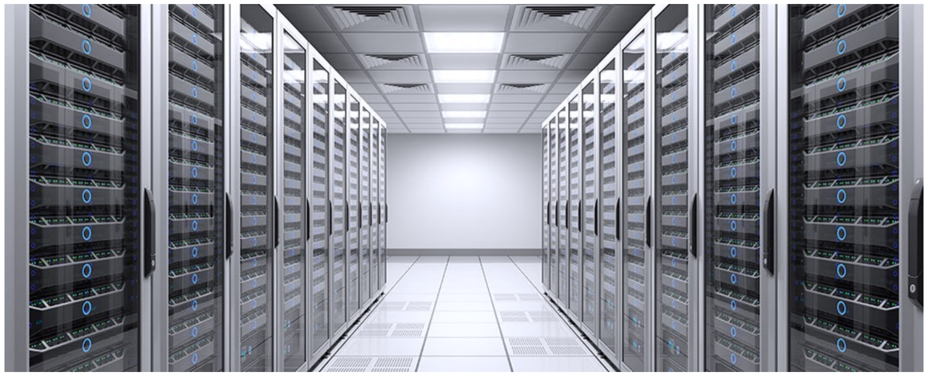
Data-Intensive Applications: Bring Compute to the Data

We want to avoid

- Centralized resources that are likely bottlenecks
- Replication of data structures across nodes
- Communication of too much intermediate data

We need a programming model with data locality

- Same computation to be applied to large volumes of data
- Assign tasks to machines that already have the input data
- Efficient combination of intermediate results from multiple processors
- Highly distributed and scale-out



The MapReduce Programming Model

Core Idea and Benefits

MapReduce is a programming model for processing big data sets with a parallel, distributed algorithm on a cluster

The core idea behind MapReduce is **mapping** your data set into a collection of <key, value> pairs, and then **reducing** over all pairs with the same key

The concept is quite **powerful** because almost all data can be mapped into <key, value> pairs somehow, and keys and values can be of any type (strings, integers, user-defined…)

The concept is very **simple** because developers are required to only write simple map and reduce functions, while distribution and parallelism are handled by the MapReduce framework

The concept is very **efficient** because computation operations are performed on data local to the computing node, data transfer over the network is reduced to a minimum

The MapReduce Programming Model

Not So New

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

OSDI'04, San Francisco, CA, December, 2004
https://www.usenix.org/legacy/event/osdi04/tech/full_papers/dean/dean.pdf

The MapReduce Programming Model

Assign Compute to Machines that already Have the Data

The programmer essentially only specifies two (sequential) functions

STEP 1. MAP: *map(k1,v1) → list(**k2**,v2)*

- Inputs each record consisting of key of type k1 and value of type v1
- Outputs a set of intermediate key-value pairs, each of type k2 and v2
- Types can be simple or complex user-defined objects
- Each map call is independent

STEP 2. SUFFLING: Internal grouping of all intermediate pairs with same key together and passes them to the workers executing reduce

STEP 3. REDUCE: *reduce(**k2**,list(v2)) → list(k3,v3)*

- Combines information across records that share this same **intermediate key**

This is too abstract!

The MapReduce Programming Model

WordCount Example on a Single System

STEP 1. MAP: *map(k1,v1) → list(**k2**,v2)*

mapper.py

```python
#!/usr/bin/python

import sys
import re

for line in sys.stdin:
    line = re.sub( r'^\W+|\W+$', '', line )
    words = re.split(r"\W+", line)

    for word in words:
        print( word.lower() + "\t1" )
```

- Parse input text lines
- Extract words
- For each word writes the “word” as output key and “1” as value

```
$ mapper.py < input.txt
…
email 1
newsletter 1
to 1
hear 1
about 1
new 1
```

The MapReduce Programming Model

WordCount Example on a Single System

STEP 2. SUFFLING

```
$ mapper.py < input.txt | sort
…
zodiac 1
zodiac 1
zogranda 1
zone 1
zone 1
zone 1
zone 1
zone 1
zoned 1
zoned 1
zones 1
zones 1
zones 1
zoology 1
zoology 1
zoroaster 1
```

The MapReduce Programming Model

WordCount Example on a Single System

STEP 3. REDUCE: *reduce(**k2**,list(v2)) → list(k3,v3)*

reducer.py

```python
#!/usr/bin/python

import sys

previous = None
sum = 0

for line in sys.stdin:
    key, value = line.split( '\t' )

    if key != previous:
        if previous is not None:
            print str( sum ) + '\t' + previous
        previous = key
        sum = 0

    sum = sum + int( value )

print str( sum ) + '\t' + previous
```

- Count the number of times each key occurs by summing values as long as they have the same key
- Publish the result once the key changes

```
$ mapper.py < input.txt | sort | reducer.py
...
3 zones
2 zoology
1 zoroaster
```

The MapReduce Programming Model

Prototyping and Debugging – Hadoop Streaming

Both the mapper and the reducer should be python executable scripts that read the input from stdin (line by line) and emit the output to stdout

```
$ cat files | mapper.py| sort | reducer.py
```

1. Copy files to HDFS

```
bin/hadoop dfs -copyFromLocal /tmp/gutenberg /user/hduser/gutenberg
```

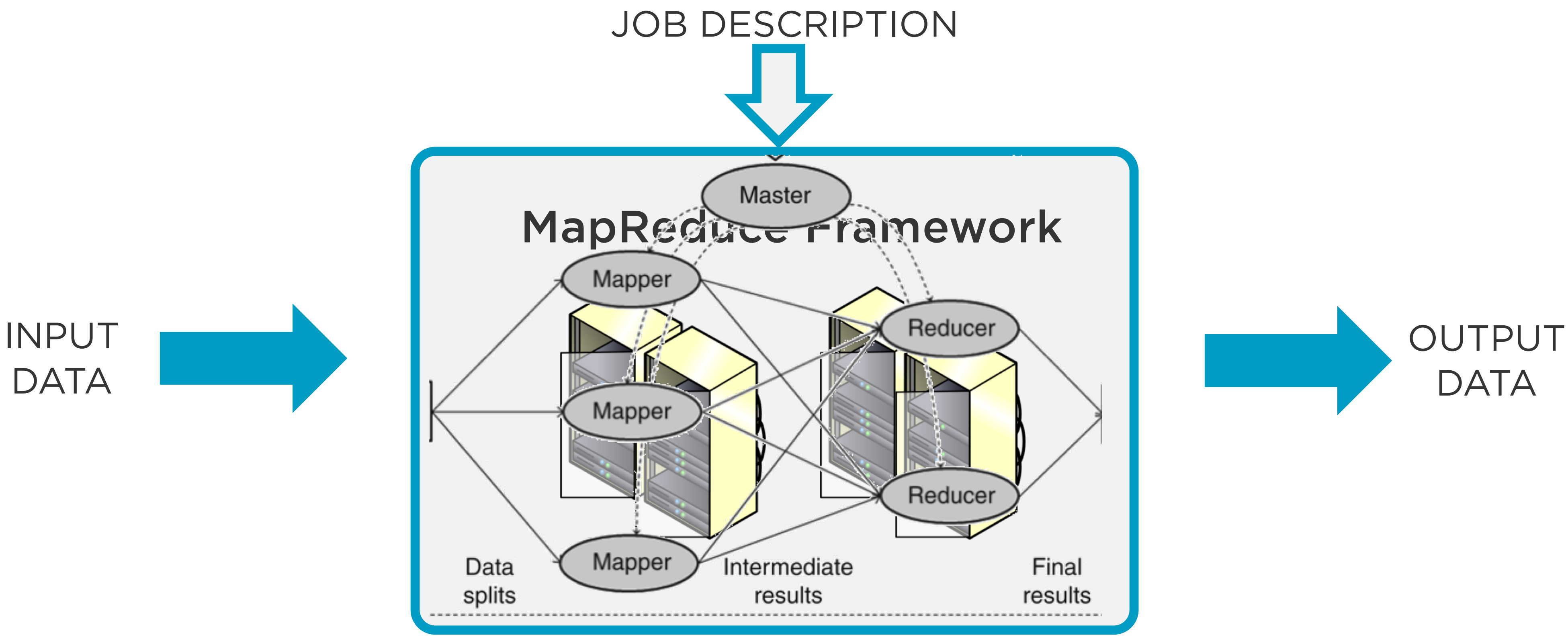
2. Execute Hadoop command

```
$ bin/hadoop jar contrib/streaming/hadoop-*streaming*.jar \ -file
/home/hduser/mapper.py -mapper /home/hduser/mapper.py \ -file
/home/hduser/reducer.py -reducer /home/hduser/reducer.py \ -input
/user/hduser/input/* -output /user/hduser/g-output
```

3. Read all output files (one per reducer)

The MapReduce Programming Model

It Is All about the Framework for Parallel Processing



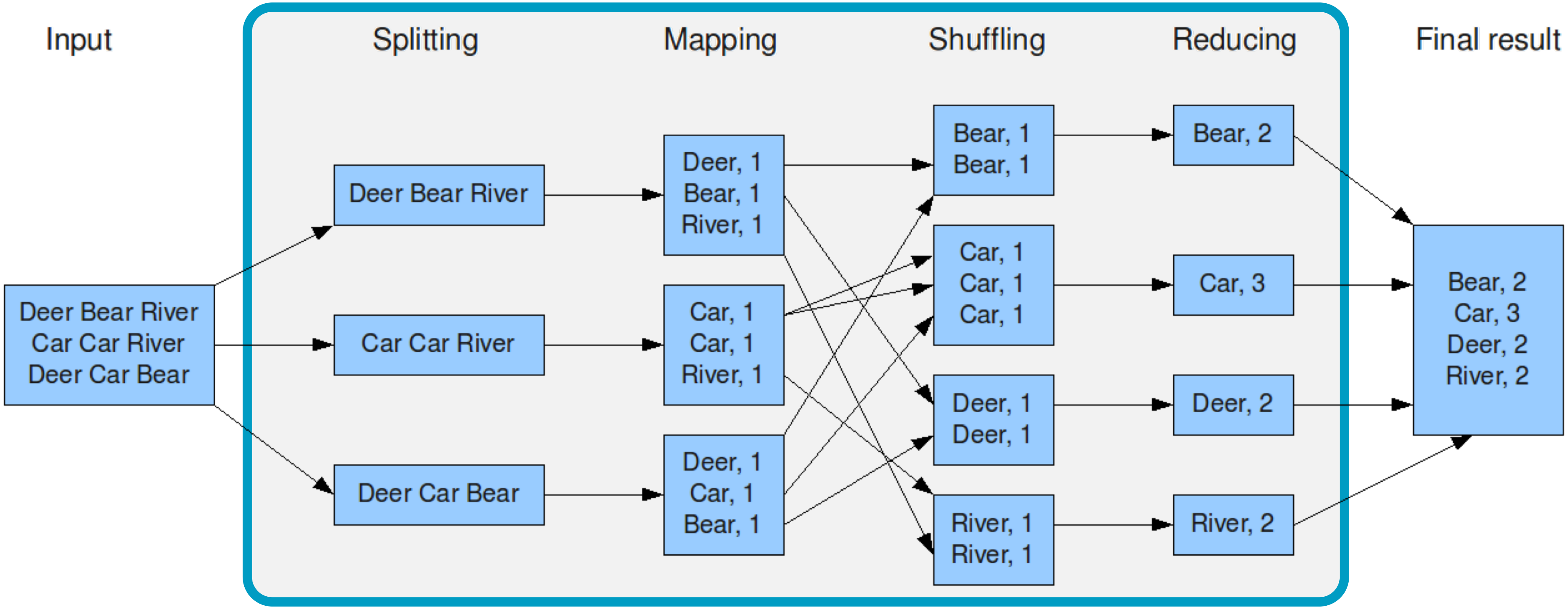
Programmer focus on the algorithm while the framework takes care of:

- Parallelizing program execution
- Partitioning input data
- Delivering data chunks to the different worker machines
- Scheduling the map/reduce tasks for execution on the worker machines
- Handling machine failures and slow responses

The MapReduce Programming Model

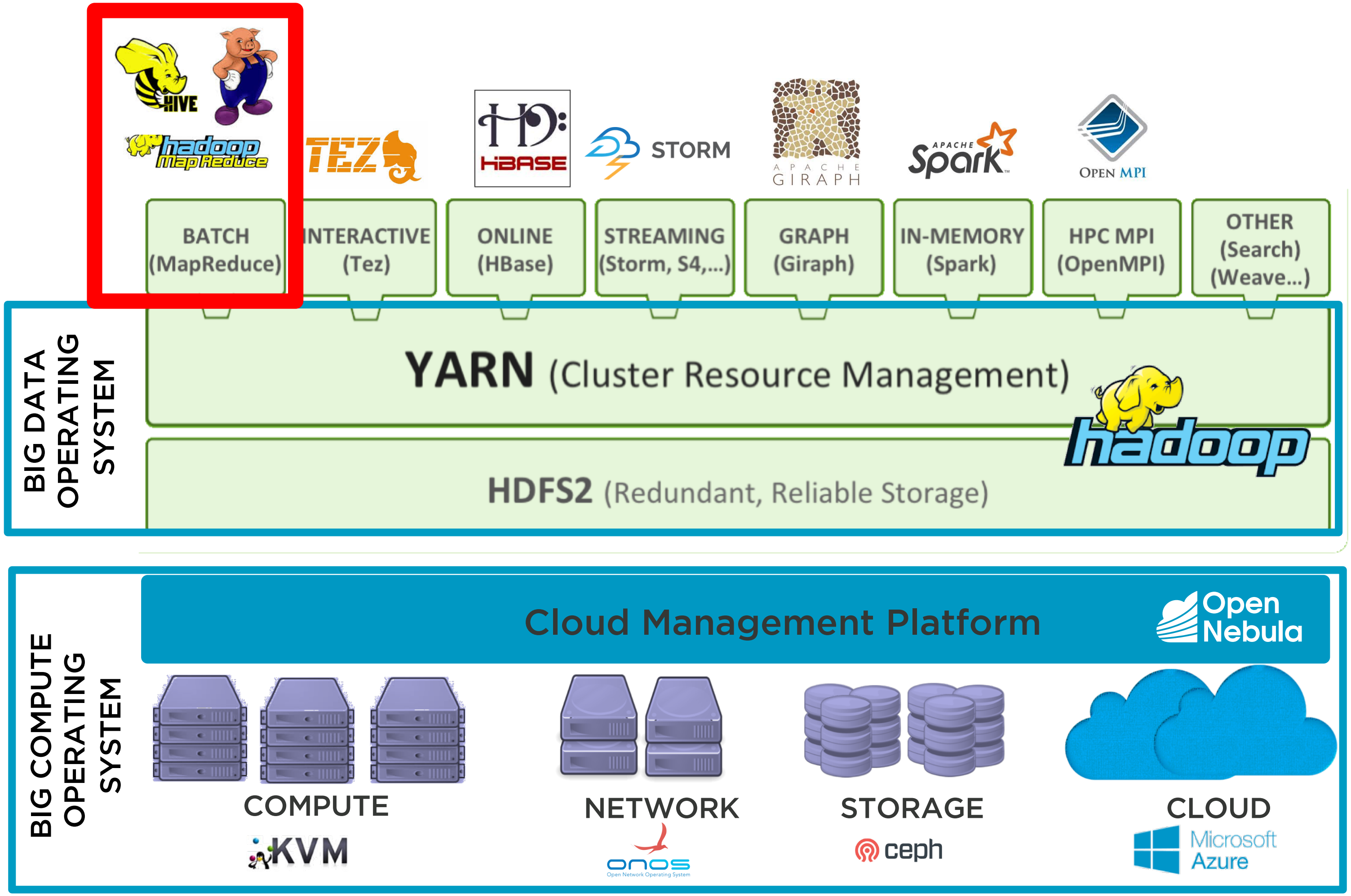
WordCount Example on a Parallel System

The overall MapReduce word count process



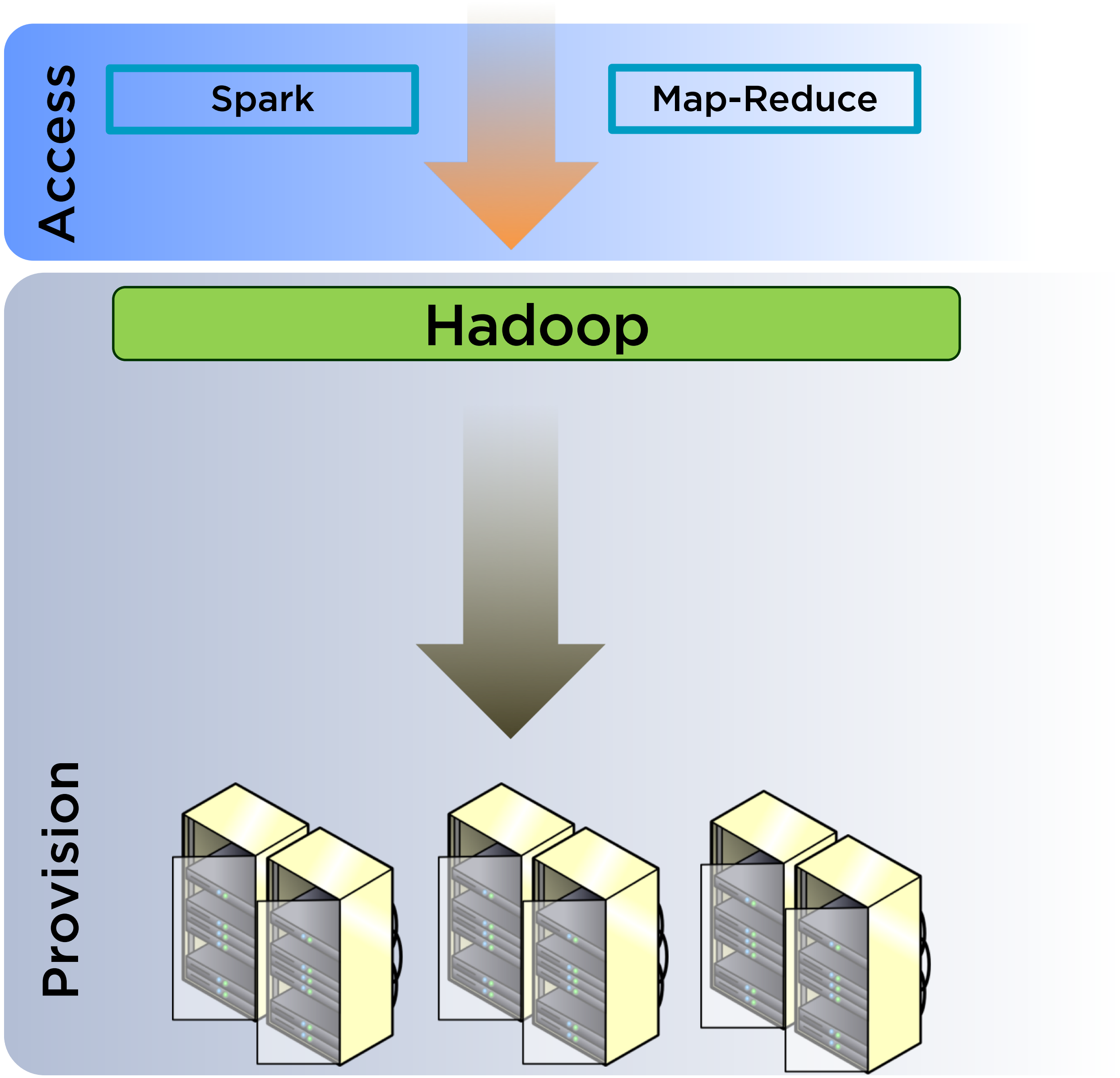
The Hadoop Processing Framework

Apache Hadoop and Alternatives



The Hadoop Processing Framework

Bare-metal Deployment (On-premises)

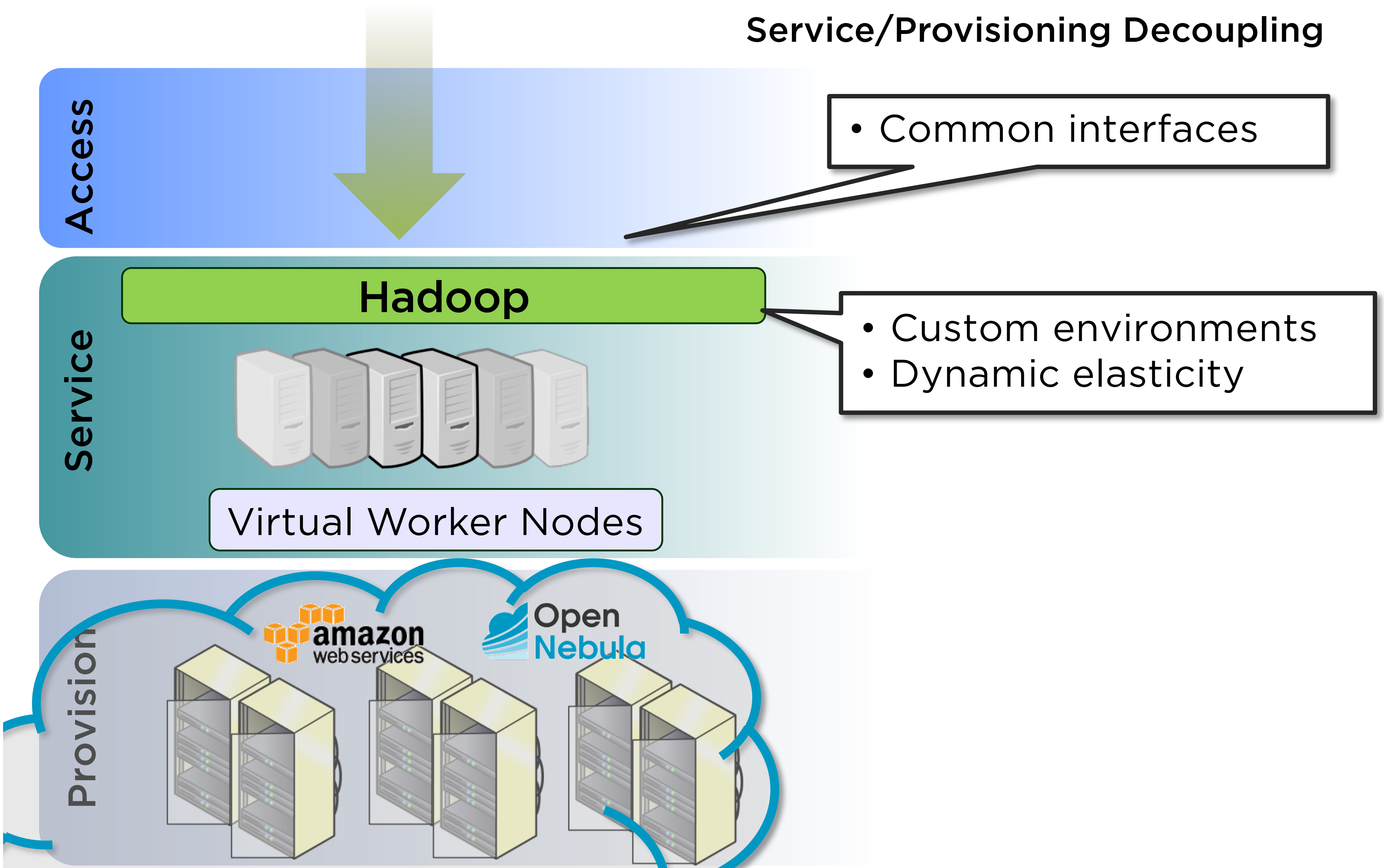


BIG DATA

The Hadoop Processing Framework

Cloud Deployment

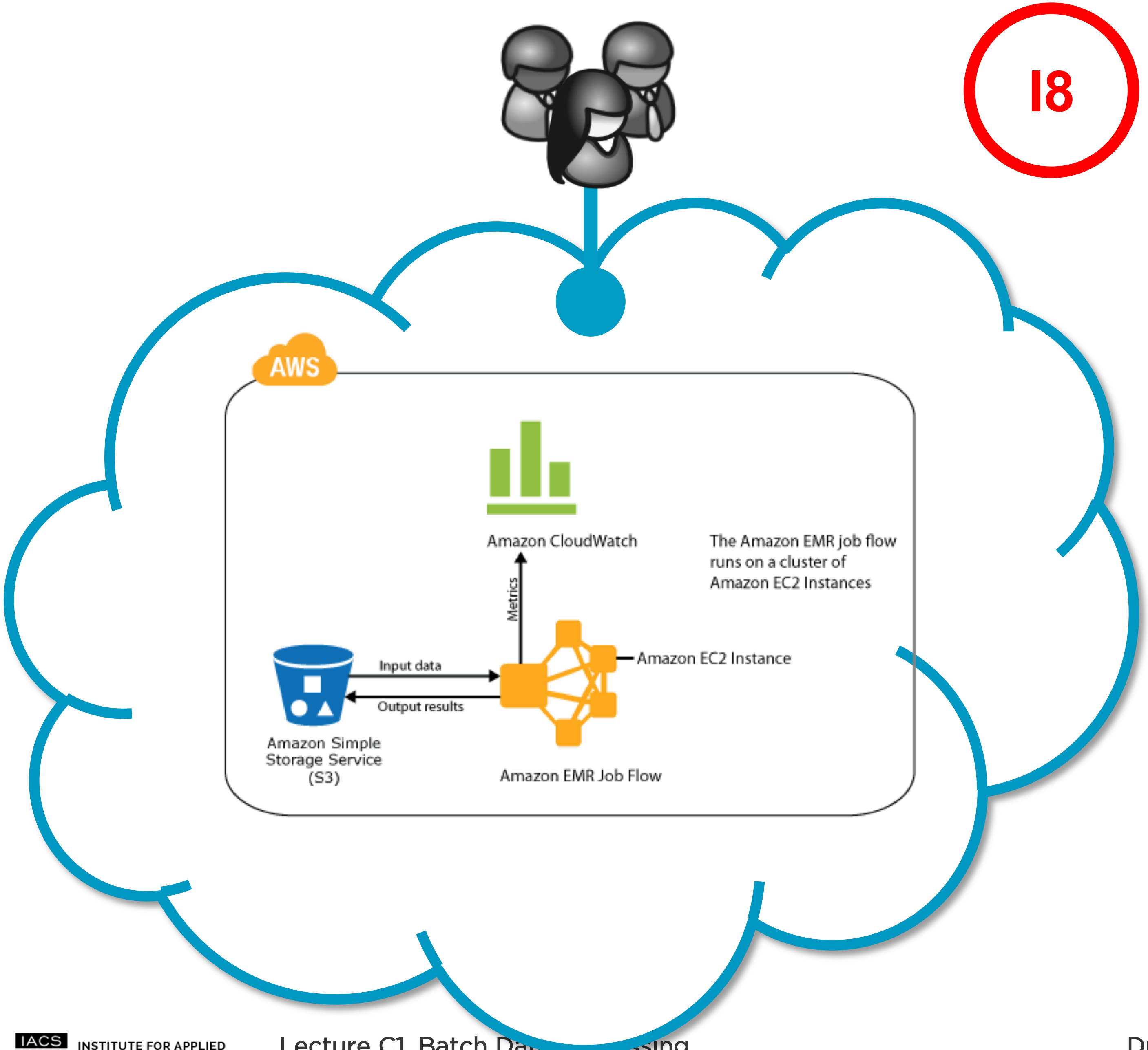
Service/Provisioning Decoupling



The Hadoop Processing Framework

Elastic Map Reduce - AWS

18

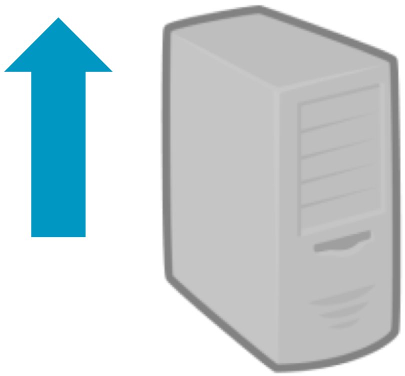


The Hadoop Processing Framework

Scale Horizontally!

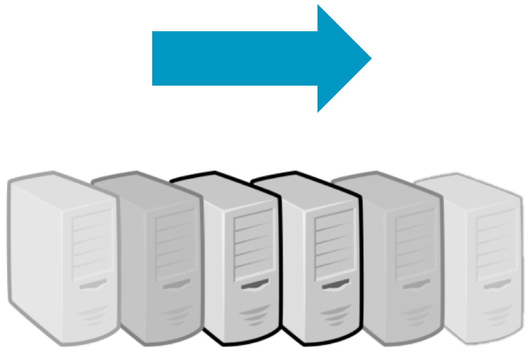
Scale up

fewer, larger servers



Scale out

More, smaller servers



The Hadoop Processing Framework

Elastic Map Reduce - AWS

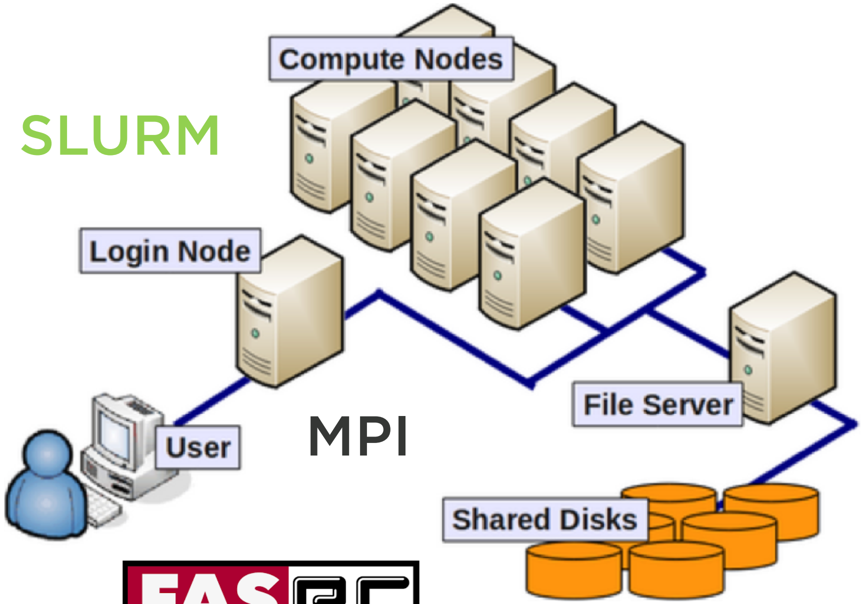
Filter: Filter instance groups ... 2 instance groups (all loaded)

ID	Status	Node type & name	Instance type	Instances	Purchasing option	Auto Scaling
ig-RIN3SNG19O5S	Running	**CORE** Core Instance Group	**m3.xlarge** 8 vCore, 15 GiB memory, 80 SSD GB storage EBS Storage: none	2 Instances Resize	**On-demand**	Not enabled
ig-2MTQ4AOPQC6MD	Running	**MASTER** Master Instance Group	**m3.xlarge** 8 vCore, 15 GiB memory, 80 SSD GB storage EBS Storage: none	1 Instances	**On-demand**	Not available for Master

The Hadoop Processing Framework

Clustered Architectures

SLURM

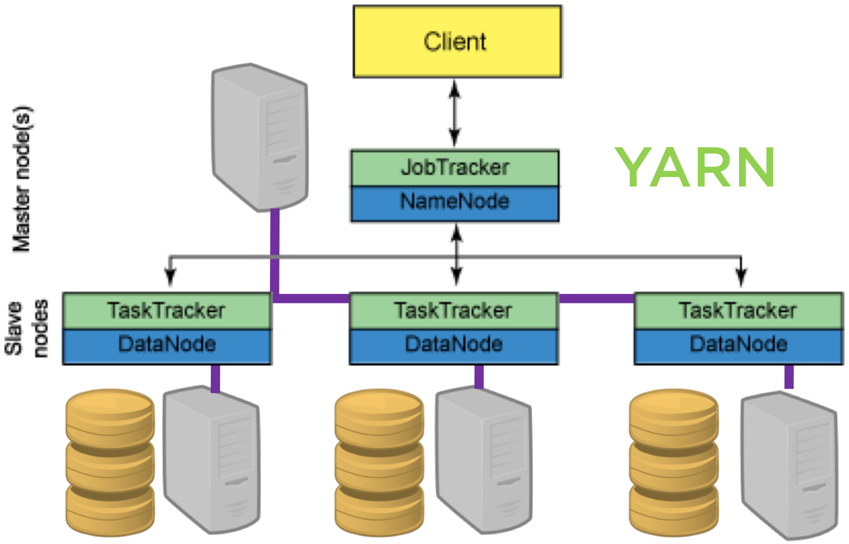


MPI

LUSTRE

Compute-centric

MapReduce



YARN

HDFS

Data-centric

Next Steps

- Quiz today!
- Get ready for next **lab:**
  I8. Hadoop Cluster on
- Get ready for next **hands-on:**
  H4. MapReduce Design Patterns (Tuesday 3/23)

Questions

Batch Data Processing

http://piazza.com/harvard/spring2021/cs205/home