PySpark on Jupyter Notebook

- Install PySpark on Windows
- Test and Demo using Jupyter Notebook

PySpark: Parallelize()

Distribute the data across multiple nodes

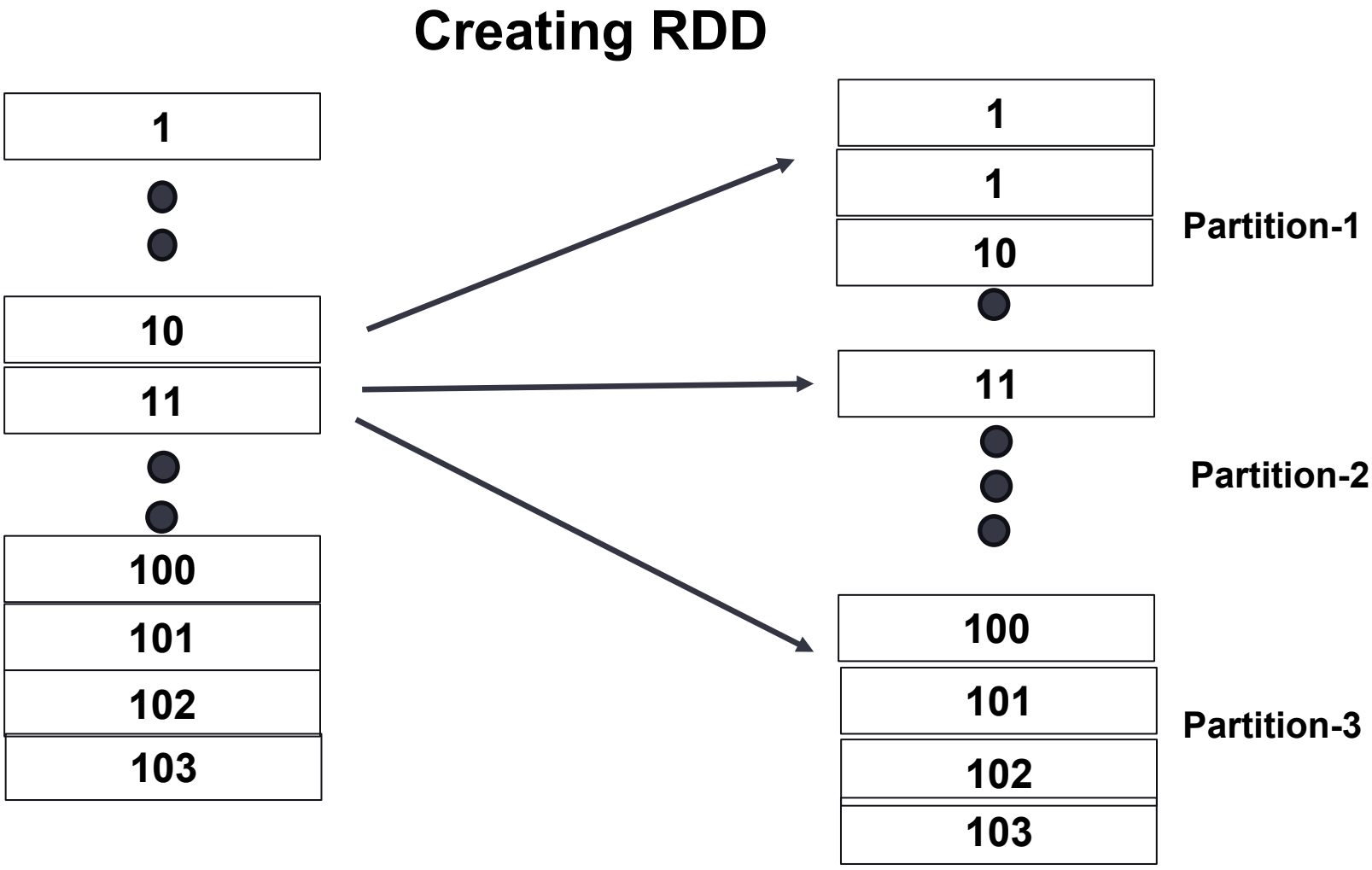
- Parallelize() is a function in SparkContext
- Creates Resilient Distributed Datasets (RDD) from a list collection.
- RDD is a data structure.
- Each dataset in RDD is divided into logical partitions, each partition may be computed on a different node of the cluster.

```
nums = list(range(0,100000,1)
sc = sparkContext.parallelize(nums)
```

Creates an RDD and distributes “nums” across workers in Hadoop environment

Let’s see how this works!

Resilient Distributed Data: Parallelize()

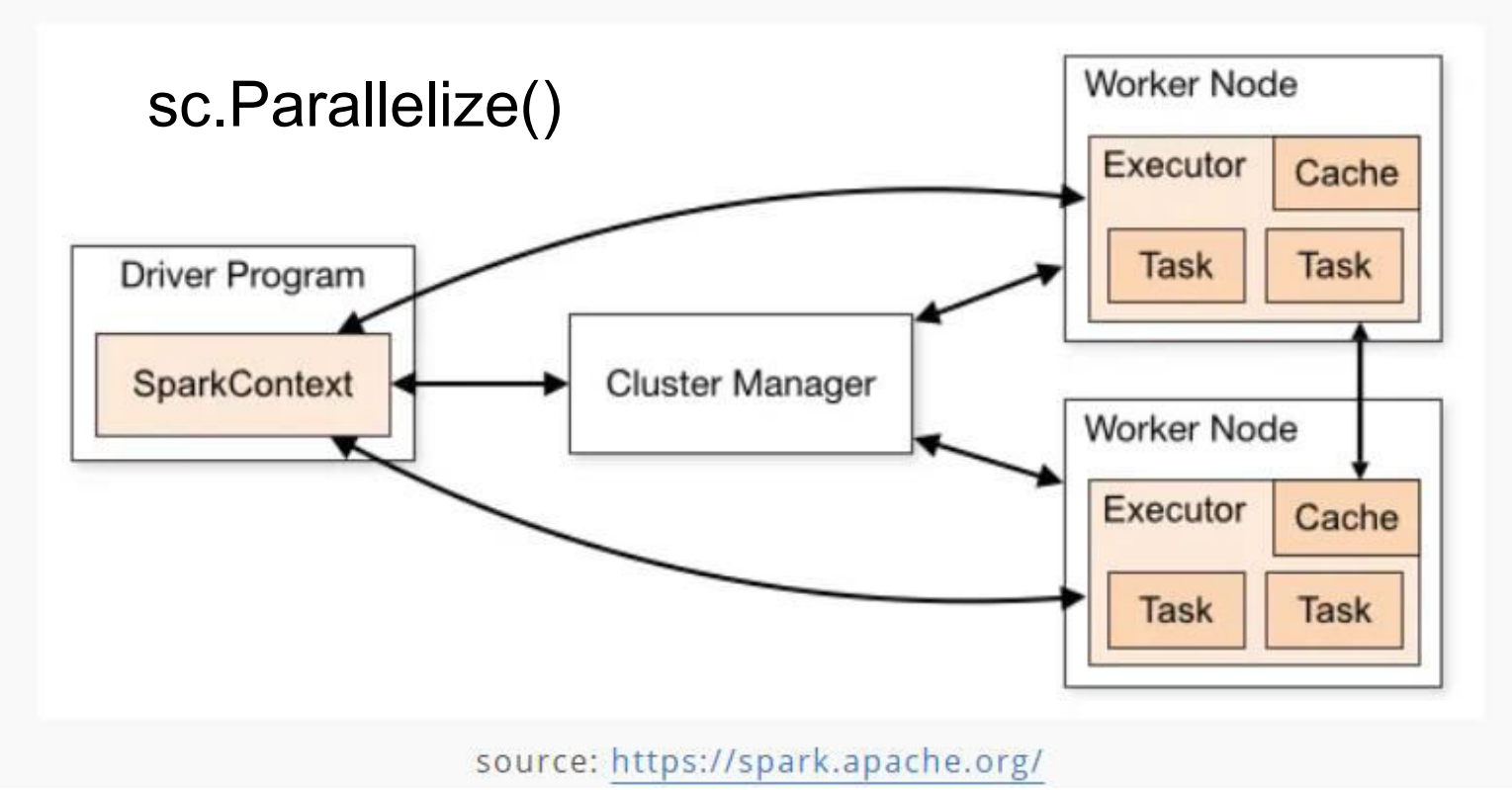


PySpark

- Big Data in Realtime
- In-memory computing
- **Spark Context (SC)**
- Linux

Features

- Speed Caching
- PolyGlot (supports several languages)
- Widely used across verticals
- Resilient Distributed Datasets (RDD)



PySpark with Jupyter Notebook

```
!pip3 install pyspark
import pyspark
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Org').getOrCreate()
sc  = spark.sparkContext

Testing
numbers = list(range(0,100000,1))

#create an RDD and distribute the list across available nodes
numbers_rdd = sc.parallelize(numbers)

#list the numbers using methods
numbers_rdd.collect()
numbers_rdd.take(5)
```

Additional Materials and Tutorials

How to set up PySpark for your Jupyter notebook | Opensource.com

PySpark parallelize() - Create RDD from a list data — SparkByExamples

PySpark

Outcomes:

- Introduction to Hadoop Architecture
- PySpark for Data Frames

Advantages of Spark DataFrame

- Scales from desktop to PetaFLOPs
- Seamless integration with various Big Data tools and Storage Systems
- APIs for Python, Scala, Java, and R

DataFrames are …

- Preferred abstraction of data in Spark
- Distributed (by design) across the Hadoop cluster
- Built on Resilient Distributed Datasets (RDD). At least three copies across the cluster
- Immutable once constructed

Benefits:

- Compute in parallel

How to Build a DF?

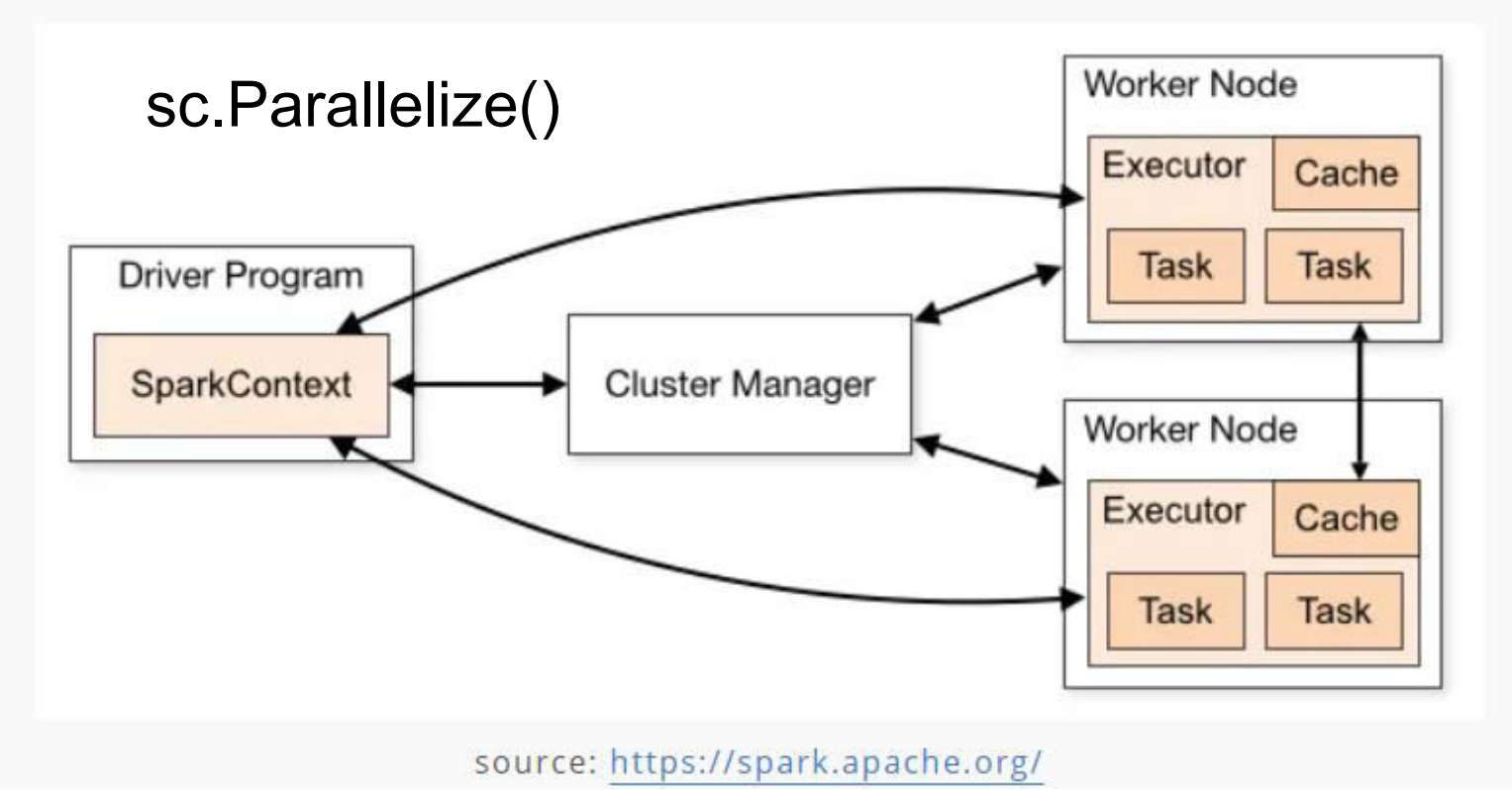
- Convert Pandas DF
- Transform an existing DF
- From files in HDFS or any other storage system such as Parquet

PySpark

- Big Data in Realtime
- In-memory computing
- **Spark Context (SC)**
- Linux

Features

- Speed Caching
- PolyGlot (supports several languages)
- Widely used across verticals
- Resilient Distributed Datasets (RDD)



PySpark: Parallelize()

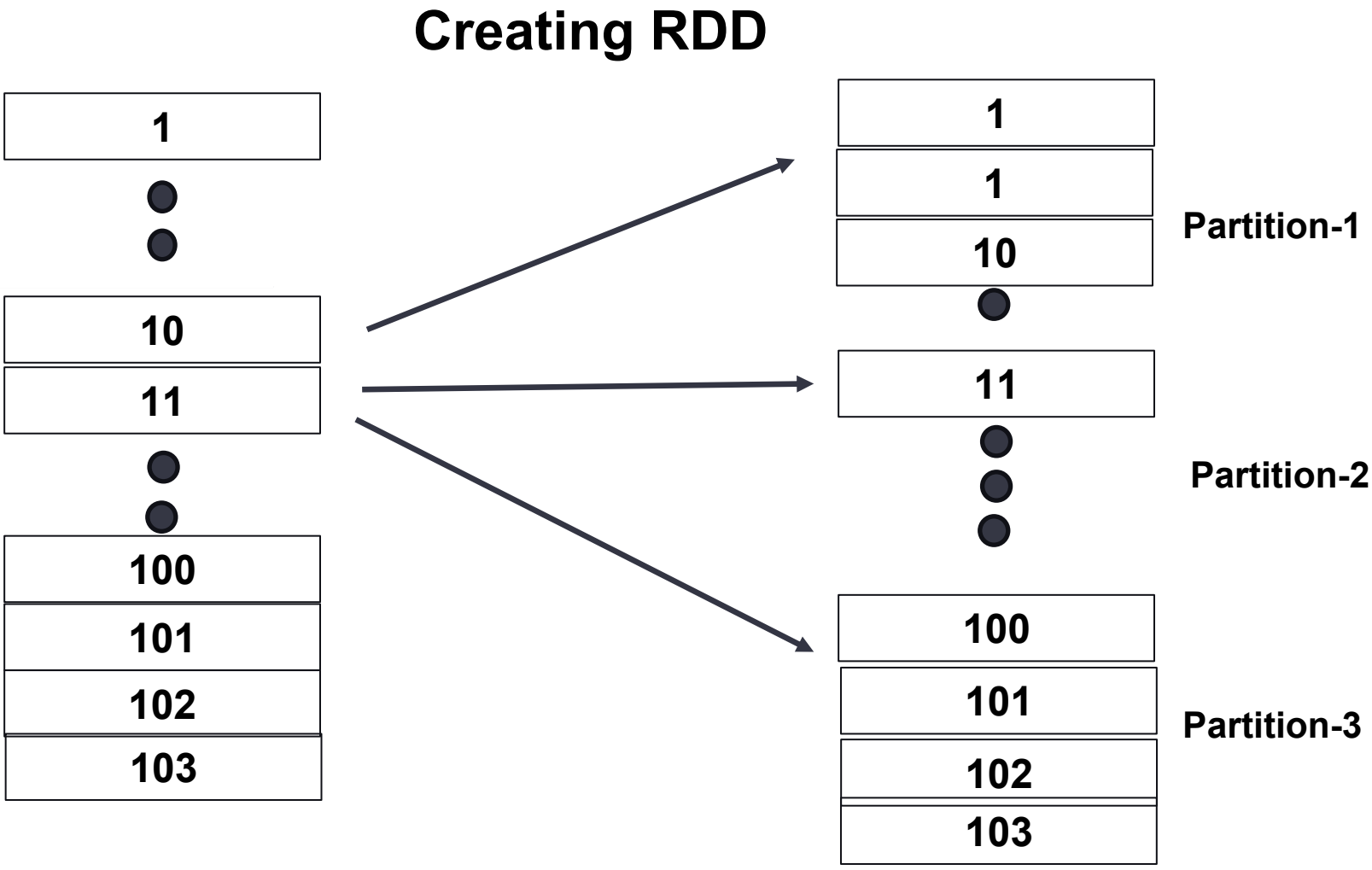
Distribute the data across multiple nodes

- Parallelize() is a function in SparkContext
- Creates Resilient Distributed Datasets (RDD) from a list collection.
- RDD is a data structure.
- Each dataset in RDD is divided into logical partitions, each partition may be computed on a different node of the cluster.

```
nums = list(range(0,100000,1)
sc = sparkContext.parallelize(nums)
```

Creates an RDD and distributes “nums” across workers in Hadoop environment

Resilient Distributed Data: Parallelize()



DataFrames vs. RDD vs. Dataset

- DataFrames
  - API support for programming languages such as Scala, Java, and R
  - Easy to program compared to using RDD
  - Code optimization provides improved performance of the DataFrame
- Resilient Distributed Data
  - Immutable once created
  - Divided into logical partitions across nodes on a cluster
  - Provides OOP style with compile-time safety (e.g., if you access a column that doesn't exist in the table: RDD detects sooner. DataFrame APIs detects the attribute error at runtime only!)
- Datasets
  - Allows conversion of existing RDD and DataFrame into Datasets
  - Provides OOP style with compile-time safety (e.g., DataFrame APIs do not support compile-time error. It detects attribute error at runtime only!)

Overhead: Garbage Collection

- RDD. Immutable.
  - Each time an object (row/column element) is modified
  - Create/destroy individual objects
- DataFrame
  - No garbage collection
  - Avoids cost of creating/destroying individual objects for each row in the dataset
- Dataset
  - No garbage collection

PySpark With Jupyter Notebook

#Create an environment (if you like to keep this install separate)

conda activate fall23-5300

pip install pyspark

Luanch Jupyter notebook

#change kernel to fall23-5300

import pyspark

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('Org').getOrCreate()

sc = spark.sparkContext

import numpy as np

Testing

numbers = list(np.arange(0,100000,1)

#create an RDD and distribute the list across available nodes

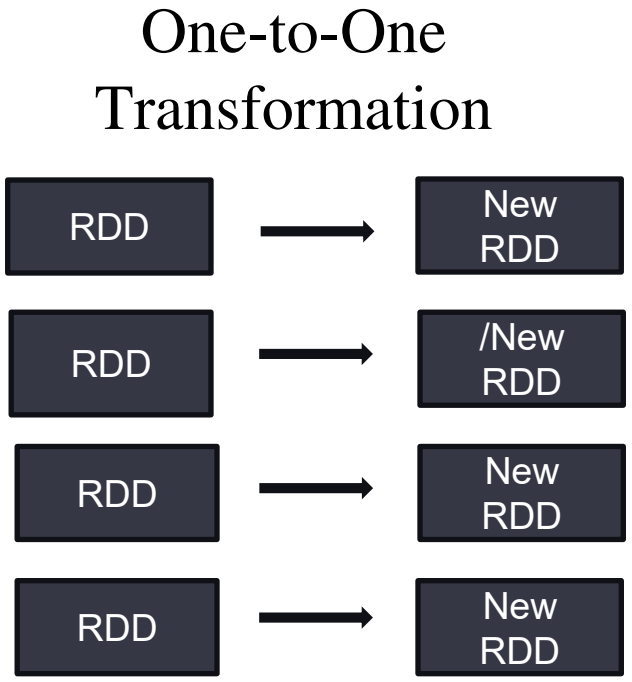
numbers_rdd = sc.parallelize(numbers)

#list the numbers using methods

numbers_rdd.collect()

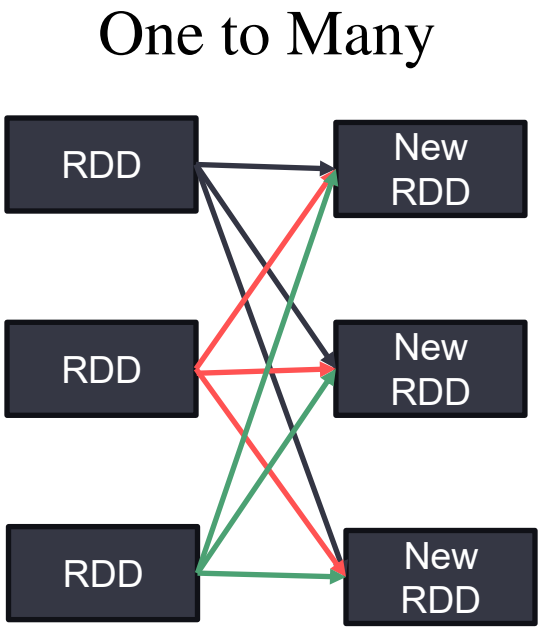
Narrow Transformation: Spark Job Optimization

- Transformations and Actions:
- Each transformation produces a new RDD because RDD is immutable
  - Lazy transformation.
  - Executes when called but, not immediately!
- DataFrames Vs Datasets



Wide Transformation: Spark Job Optimization

- Parallel transformations
- Partitioning: All jobs are not created equally!
- Repartitioning
- Serialization: Data (into RDD), and closure (introduced into computation)
- Memory Management: application driver executor memory (for data movement, swap, etc.)
- Cluster resources (load, hardware, accelerators, etc.)



Installing Spark on Oracle VM

STOP! You need Oracle VM or another Working Linux Environment!

Install java

$sudo apt update
$sudo apt install openjdk-11-jdk
$sudo apt install openjdk-11-jre
$export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
$export JAVA_JRE==/usr/lib/jvm/java-11-openjdk-amd64/jre

Install python

$sudo apt install python3-pip

Install Jupyter

$pip3 install jupyter

Install scala

$sudo apt-get install scala

Install py4j

$pip3 install py4j

Installing Spark on Ubuntu VM

Download spark package

$wget https://dlcdn.apache.org/spark/spark-3.3.2/spark-3.3.2-bin-hadoop3.tgz

$sudo tar -xzvf spark-3.3.2-bin-hadoop3.tgz

$ln -s spark-3.3.2-bin-hadoop3 spark

Add following lines at the end of the file ".bashrc"

$nano fun~/.bashrc

export SPARK_HOME=/home/username/spark

export PYTHONPATH=$SPARK_HOME/python:$PYTHONPATH

export PYSPARK_DRIVER_PYTHON="jupyter"

export PYSPARK_DRIVER_PYTHON_OPTS="notebook"

Run .bashrc

$source ~/.bashrc

Test Spark Installation

$cd spark/bin

$./spark-shell --version

```
tirtha@tirtha-VirtualBox:~/Spark/spark-2.3.1-bin-hadoop2.7/bin$ spark-shell --version
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.3.1
      /_/

Using Scala version 2.11.8, Java HotSpot(TM) 64-Bit Server VM, 1.8.0_181
Branch
Compiled by user vanzin on 2018-06-01T20:37:04Z
Revision
Url
Type --help for more information.
```

Install Spark on Mac M1

Install HomeBrew package manager
$/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)“

#Add Java into PATH
$(echo; echo `eval “$(/opt/homebrew/bin/brew shellenv)” `) >> /Users/username/.zprofile
$eval “$(/opt/homebrew/bin/brew shellenv)”

#install OpenJDK
$brew install openjdk@11

#install scala
$brew install scala

#install python
$brew install python

#install spark
$brew install apache-spark

#Run PySpark
$pyspark

#Test the installation
#create a DF in pyspark shell
>>>data = [(“Java”, “2000”), (“Scala”, “10000”)]
>>>df = spark.createDataFrame(data)
>>>df.show()

_1	_2
Java	Scala
2000	10000

How to Install PySpark on Mac (in 2022) - Spark By {Examples} (sparkbyexamples.com)

Tutorials

PySpark Tutorial For Beginners | Python Examples — Spark by {Examples} (sparkbyexamples.com)

Tutorials on using PySpark with Jupyter Notebook.
PySpark Dataframe Tutorial | Introduction to Dataframes | Edureka

Google Cloud Project (look for instructions)
How to set up PySpark in Jupyter notebook | Opensource.com

PySpark parallelize() - Create RDD from a list data — SparkByExamples

DataFrame

Testing the Installation in Jupyter Notebook

Run within Jupyter Notebook
$cd $HOME
$mkdir pyspark-tests
$cd pyspark-tests
$Jupyter notebook

```
from pyspark.sql import SparkSession
```

Test in Jupyter Notebook. If the command completes with no errors, the installation is successful.

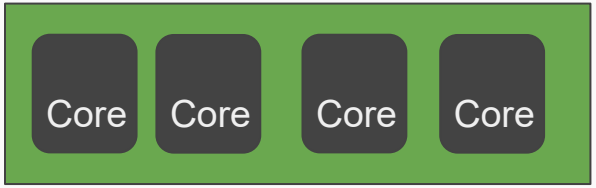
Hadoop Distributed Computing

- Local versus Distributed Systems
- Explanation of Hadoop, MapReduce, and Spark

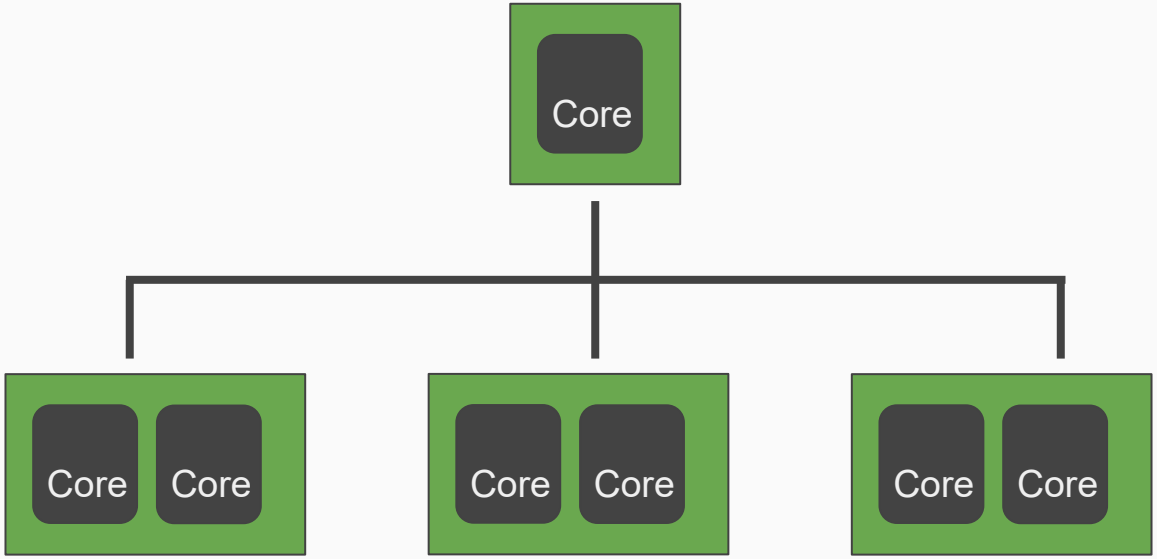
Big Data: What if Data Exceeds RAM

- We've worked with data that can fit in to RAM of a local computer.
- What can we do if we have a larger set of data?
  - Try using a SQL database to move storage onto hard drive instead of RAM
  - Or use a **distributed computing environment**, that distributes the data to multiple machines/Nodes.

Local versus Distributed



Local



Distributed

Local or Distributed Processing?

- A local **process** will use the computational resources of a single machine.
- A **distributed process** has access to the computational resources across a number of machines connected through a network.

Which computing architecture is better?

A single node with several processor cores? OR
Multiple nodes each with smaller set of cores?

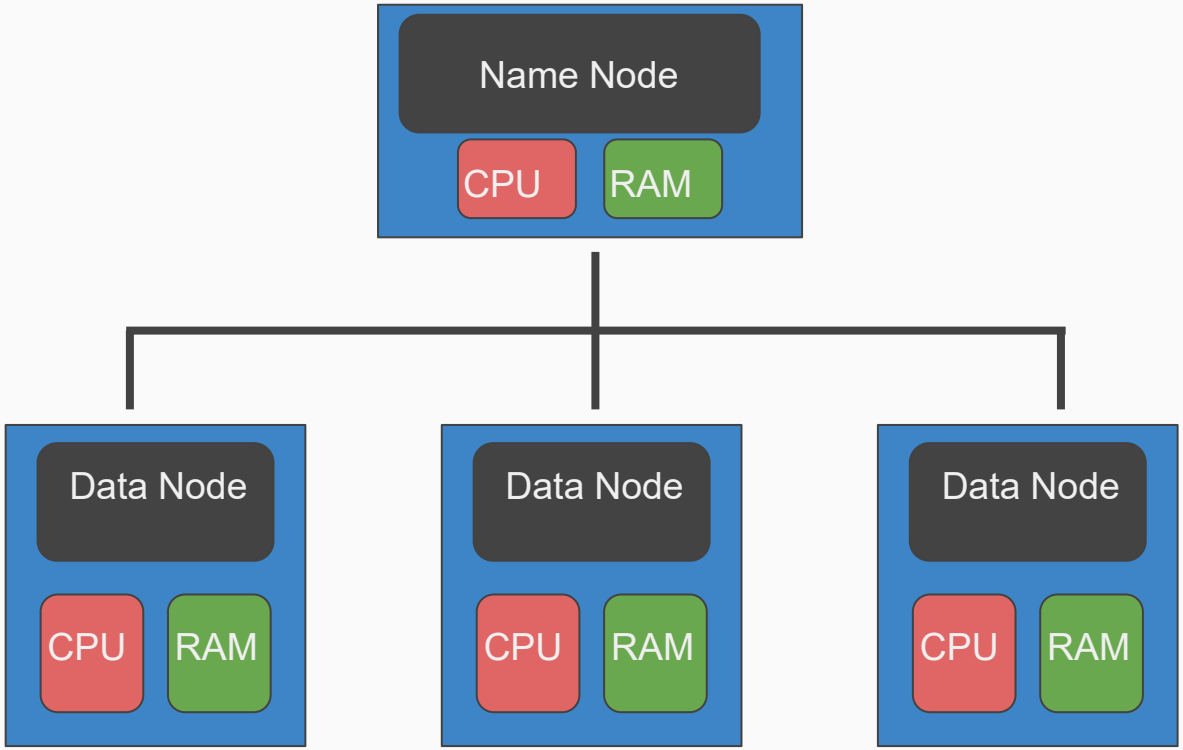
Scalability of Local Computing

- After a certain point, it is easier to scale out to Nodes (each may have a small number of CPU-cores), than to trying to scale up to a single machine with a high number of CPU-cores.
- Distributed machines also have the advantage of easily scaling, you can just add more machines.

Hadoop Environment

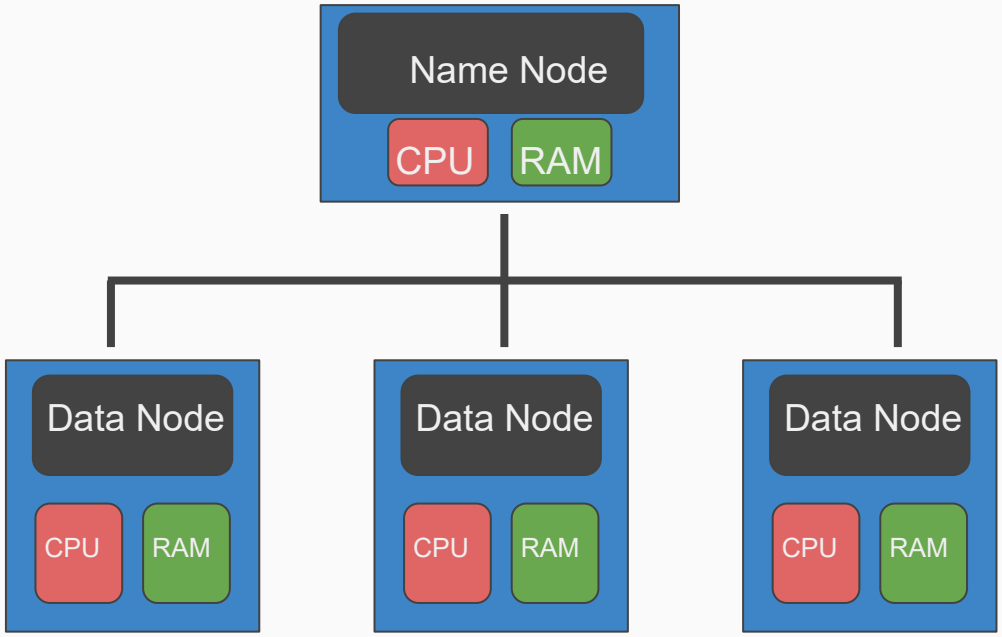
- Hadoop is a way to distribute very large files across multiple machines.
- It uses the Hadoop Distributed File System (HDFS)
- HDFS allows a user to work with large data sets
- HDFS also duplicates blocks of data across nodes for fault tolerance
- Hadoop computing on is based on MapReduce Algorithm and distributed data via client/server or master/slave model.

Distributed Storage - HDFS



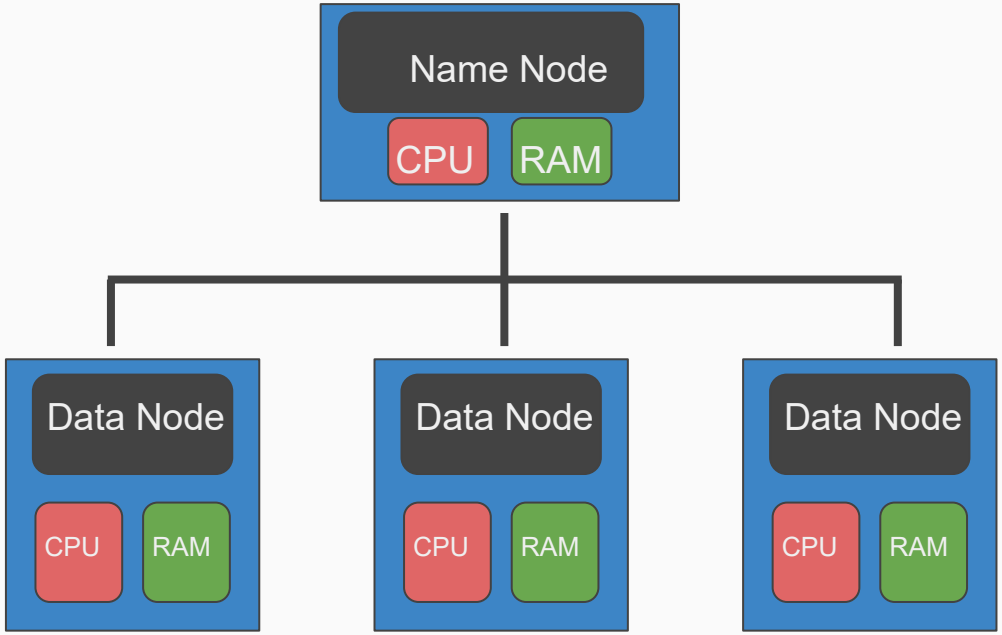
Redundant Distributed Storage

- HDFS will use blocks of data, with a size of 128 MB by default
- Each of these blocks is replicated 3 times
- The blocks are distributed in a way to support fault tolerance



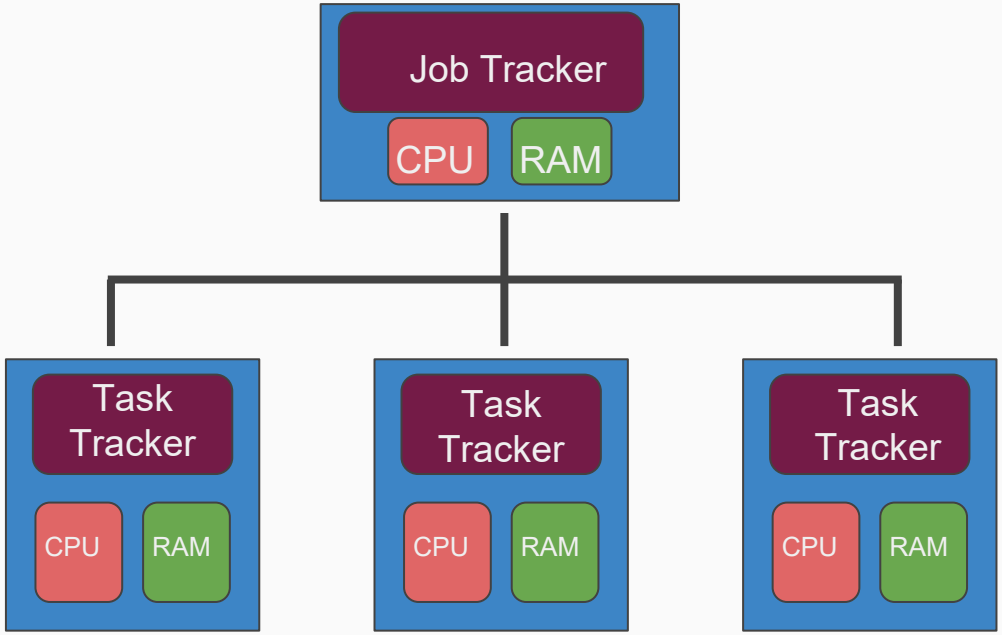
HDFS is Fault Tolerant

- Smaller blocks provide more parallelization during data processing.
- Multiple copies of a block prevents loss of data due to a failure of a node.



MapReduce Algorithm

- MapReduce is a way of splitting a computation task to a distributed set of files (such as HDFS)
- It consists of a Job Tracker and multiple Task Trackers
- The Job Tracker sends code to run on the Task Trackers
- The Task trackers allocate CPU and memory for the tasks and monitor the tasks on the worker nodes



Covered So far: Hadoop Computing

- What we covered can be thought of in two distinct parts:
  - Hadoop uses HDFS to distribute large data sets and multiple copies for fault tolarence.
  - Uses MapReduce and master/slave algorithm for comptuation on distributed data
- Spark: What does it DO?

What is Spark?

- You can think of Spark as a flexible alternative to MapReduce
- Spark can use data stored in a variety of formats
  - Cassandra
  - AWS S3
  - HDFS
  - And more

Spark vs MapReduce

- MapReduce requires files to be stored in HDFS, Spark does not!
- Spark also can perform operations up to 100x faster than MapReduce
- So how does it achieve this speed?

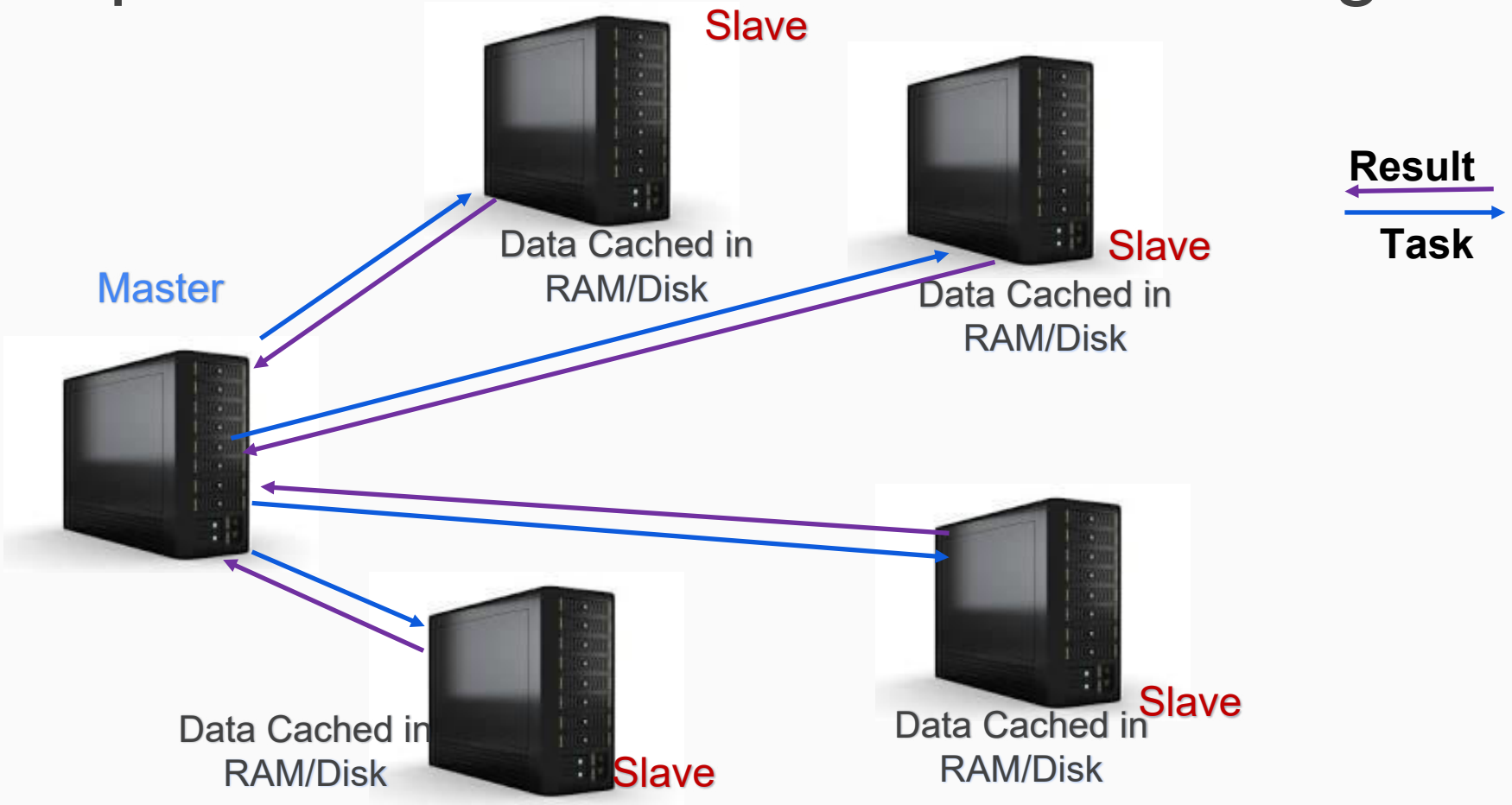
Spark: In-Memory Computing

- MapReduce writes most data to disk after each map and reduce operation
- Spark keeps most of the data in memory after each transformation
- Spark can spill over to disk if the memory is filled

Spark RDDs

- At the core of Spark is the idea of a Resilient Distributed Dataset (RDD)
- RDD supports FOUR main features:
  - Distributed Collection of Data
  - Fault-tolerant
  - Parallel operation – partition data, and
  - Ability to use many data sources

Spark RDD Communication Paradigm



Spark DataFrames

- Since the release of Spark 2.0, Spark is moving towards a DataFrame based syntax.
- RDD is still the native data structure for distributed data within Spark.
- Spark DataFrames are also now the standard way of using Spark's Machine Learning Capabilities.

Additional Resources

- Follow our tutorials:
  - Spark DataFrame Basics
  - Spark DataFrame Basic Operations
  - GroupBy and Aggregate Functions
  - Missing Data
  - Dates and Timestamps

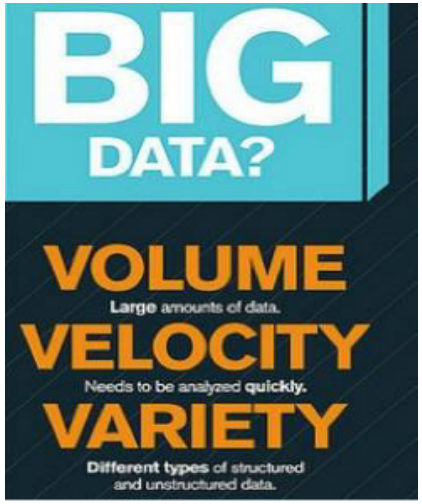
Introduction to Big Data

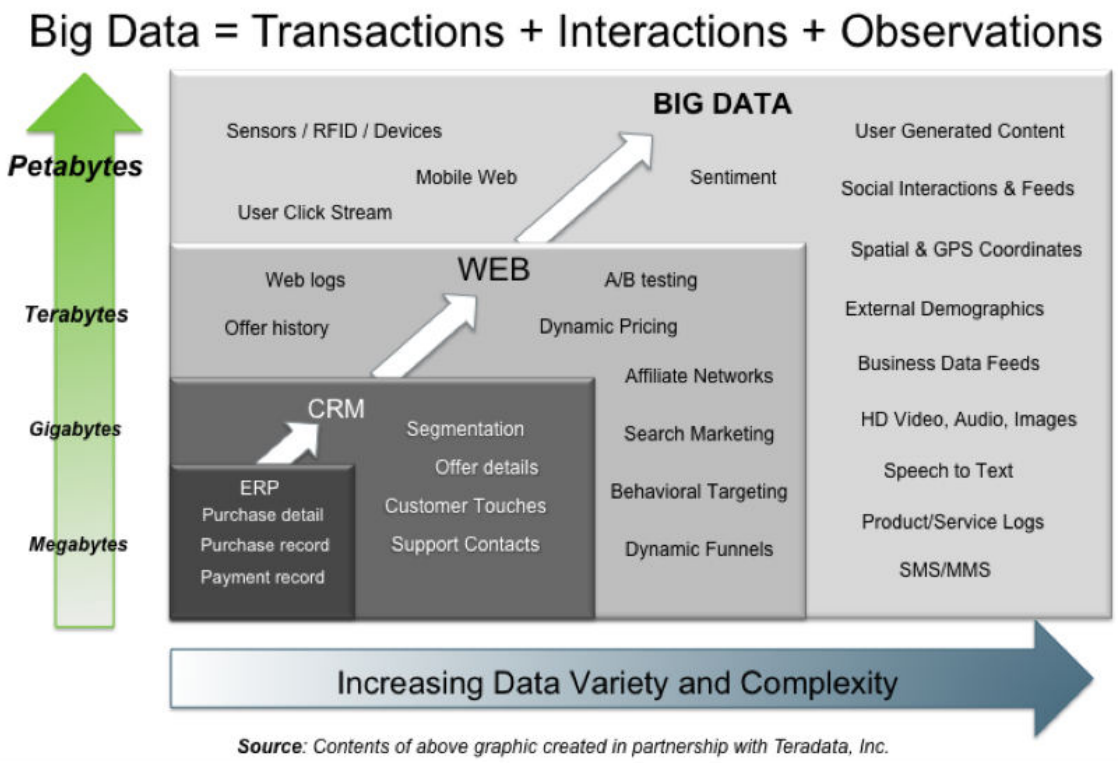
What's Big Data?

No single definition; here is from Wikipedia:

- **Big data** is the term for a collection of data sets so large and complex that it becomes difficult to process using on-hand database management tools or traditional data processing applications.
- Data Challenges include capture, curation, storage, search, share, transfer, analyze, and visualization.

Big Data: 3V's





[Courtesy Dr. Jin]

Variety (Complexity)

- **Relational Data**: Tables/Transaction/Legacy Data
- **Text Data**: Web, smart devices, speech, logs, etc.
- **unstructured Data**: XML, audio, images, etc.
- **Graph Data**
  - Social Network, Semantic Web (RDF), …
- **Streaming Data**
  - You can only scan the data once
- Public Data (online, weather, finance, policy, regulatory, legacy, etc.)

Big Data → (Data Science) → Knowledge

Velocity (Speed)

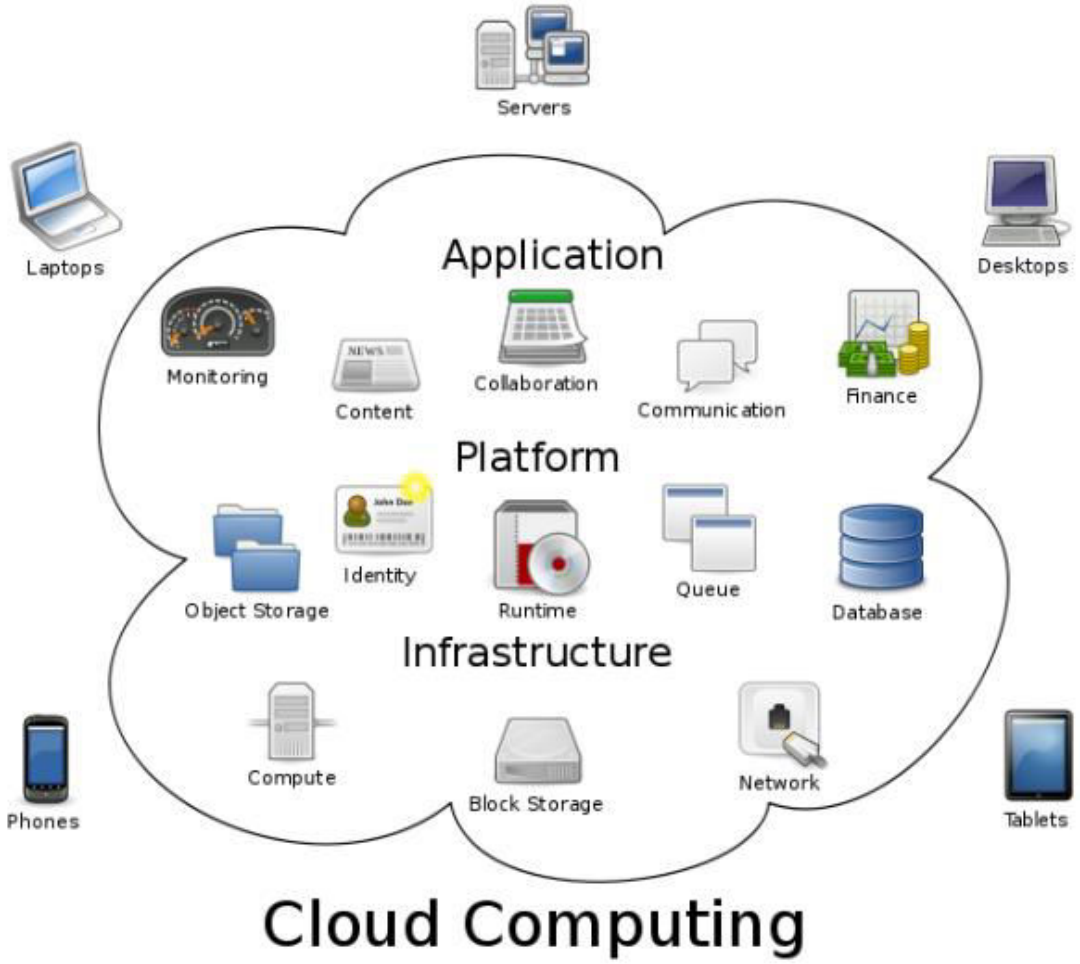
- Data is generated fast and needs to be processed fast
- Online Data Analytics
- Late decisions ➔ missing opportunities
- **Examples**
  - **E-Promotions:** Based on your current location, your purchase history, predict your likes/dislikes ➔ send promotions right now for store next to you
  - **Peer-pressure campaigns** (recommend products from what others with similar interests brought?)
  - **Precision Health:** Sensors monitoring your activities and body ➔ report results to care provider for immediate action

Cloud Computing

Cloud Computing is an on-demand delivery of IT resources in a PAYG model.

- IT resources provided as a service
  - Compute, storage, databases, visualization
- Leverages economies of scale of commodity hardware
  - Cheap storage, high bandwidth networks & multicore processors
  - Geographically distributed data centers
- Popular providers
  - Microsoft Azure, Amazon AWS, Google Cloud Project

Cloud Computing Architecture



Wikipedia: Cloud Computing

Benefits of Cloud Computing

- Cost & management
  - Economies of scale, “out-sourced” resource management
- Reduced Time to deployment
  - Ease of assembly, works “out of the box”
- Scaling
  - On demand provisioning, co-locate data and compute
- Reliability
  - Massive, redundant, shared resources
- Sustainability
  - Hardware not owned

Types of Cloud Computing

- **Public Cloud**: Computing infrastructure is hosted at the vendor's premises.
- **Private Cloud**: Dedicated to the customer and is not shared with other organizations.
- **Hybrid Cloud**: A combination of both private and public clouds. Example: Private cloud hosts critical, proprietary, business sensitive, and secure applications in private clouds. Public cloud for general purpose usage.
  - **Cloud bursting**: Use local/own infrastructure for normal usage, but cloud is used for peak workloads.

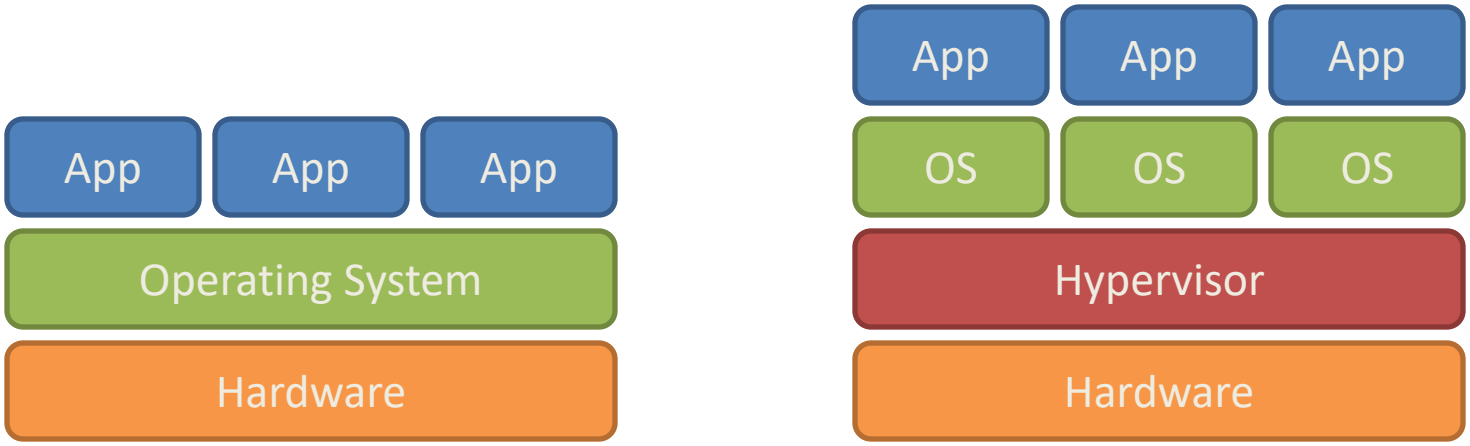
Cloud Service Oriented Architecture

- Infrastructure as a service (IaaS)
  - Offering hardware related services using the principles of cloud computing. These could include storage services (database or disk storage) or virtual servers.
  - Amazon EC2, Amazon S3, Rackspace Cloud Servers and Flexiscale.
- Platform as a Service (PaaS)
  - Offering a development platform on the cloud.
  - Google's Application Engine, Microsofts Azure, Salesforce.com's force.com .
- Software as a service (SaaS)
  - Including a complete software offering in the cloud. Users can access a software application hosted by the cloud vendor on PAYG
  - Salesforce.com offering in the online Customer Relationship Management (CRM) space, gmail, hotmail, Google docs.

Key Ingredients in Cloud Computing

- Service-Oriented Architecture (SOA)
- Utility Computing (on demand)
- Virtualization (P2P Network)
- SaaS (Software as a Service)
- PaaS (Platform as a Service)
- IaaS (Infrastructure aS a Service)
- Web Services in the Cloud

Enabling Technology: Virtualization



Everything as a Service

- Utility computing = Infrastructure as a Service (IaaS)
  - Why buy machines when you can rent cycles?
  - Examples: Amazon's EC2, Rackspace
- Platform as a Service (PaaS)
  - Give me nice API and take care of the maintenance, upgrades, …
  - Example: Google App Engine
- Software as a Service (SaaS)
  - Just run it for me!
  - Example: Gmail, Salesforce

References:

- No Official Textbooks
- Hadoop: The Definitive Guide by Tom White, O'Reilly
- Hadoop In Action by Chuck Lam, Manning
- Data-Intensive Text Processing with MapReduce by Jimmy Lin and Chris Dyer (*www.umiacs.umd.edu/~jimmylin/**MapReduce**-book-final.pdf*)
- Data Mining: Concepts and Techniques, Third Edition by Jiawei Han et al.
- Many Online Tutorials, Reports, and Papers