Spark

Basics

Why Map reduce introduced

Tradition system were failing

Centralized server to store and process data which creates bottleneck

Google introduced Map Reduce

Divide the task into small parts and assign them to many computers

Challenges with Map Reduce

Data-sharing abstraction

Concurrent data access to memory across the cluster

Inefficient use of resources

Poor memory utilization by spilling to disk after each job

Apache Spark

Big Data Tool

“Spark is an open source unified analytics engine for large-scale data processing.”

1

Fast and efficient

Supports structured, semi-structured or unstructured.

2

In-memory engine

100 times faster than Hadoop

3

Highly scalable architecture

Process terabytes of data in parallel

4

Unifies variety of use cases

Batch processing, streaming ingestion, machine learning and advanced analytics

Hadoop

HDFS

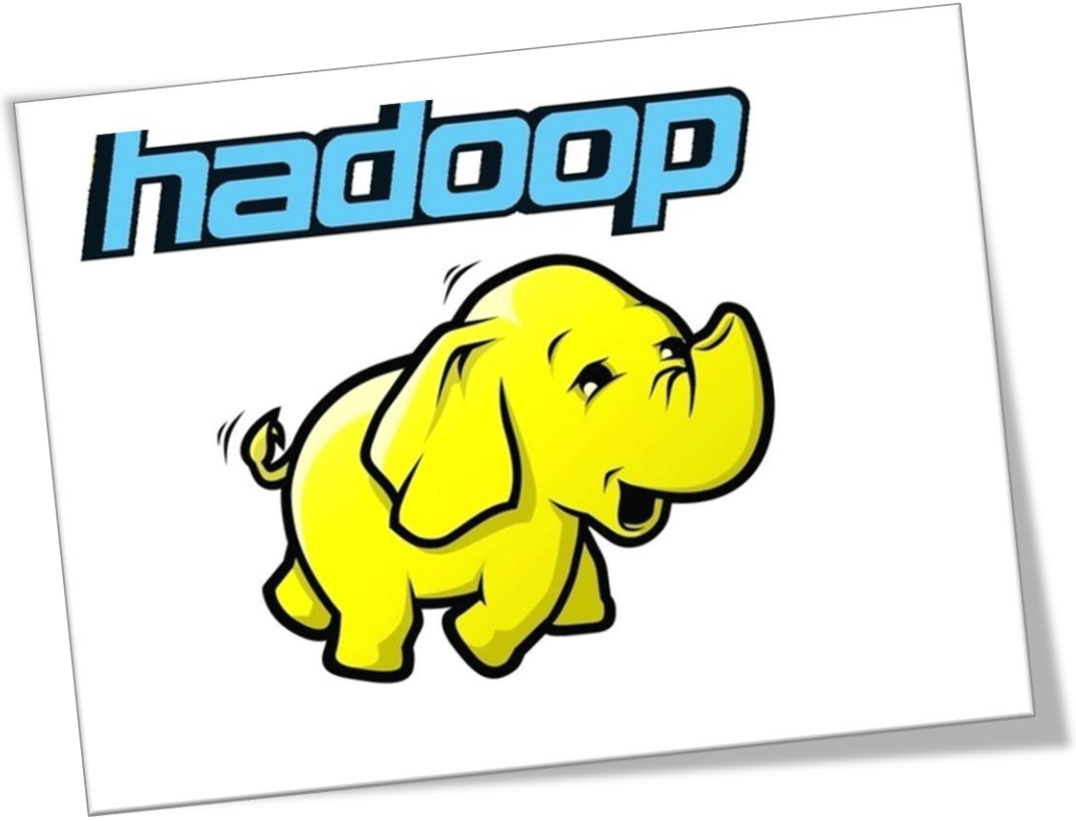
A file system to manage the storage of data

MapReduce

A framework to define a data processing task

YARN

A framework to run the data processing task



Co-ordination between Hadoop Blocks

1. Step One

User defines map and reduce tasks using the MapReduce API

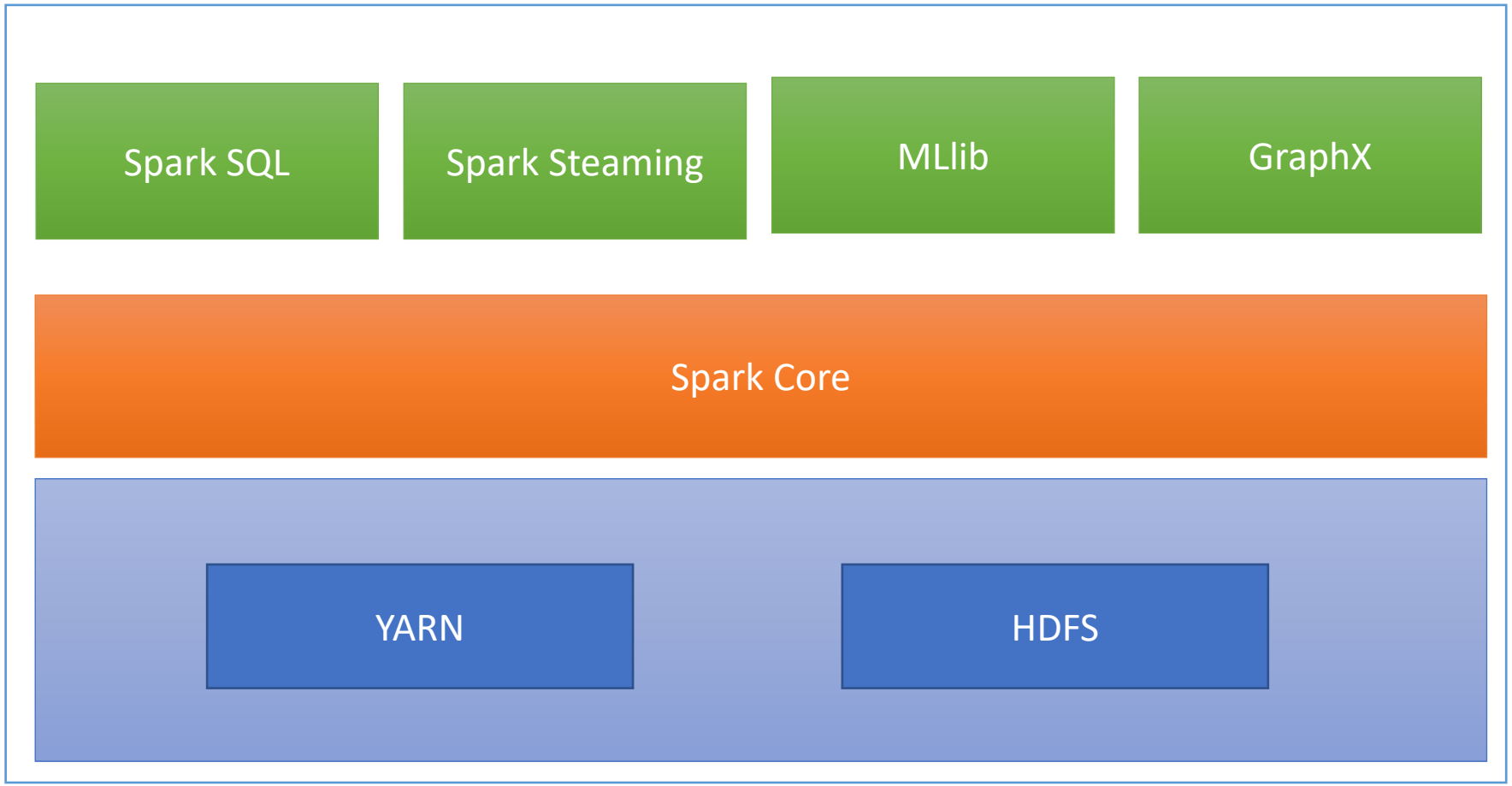
2. Step Two

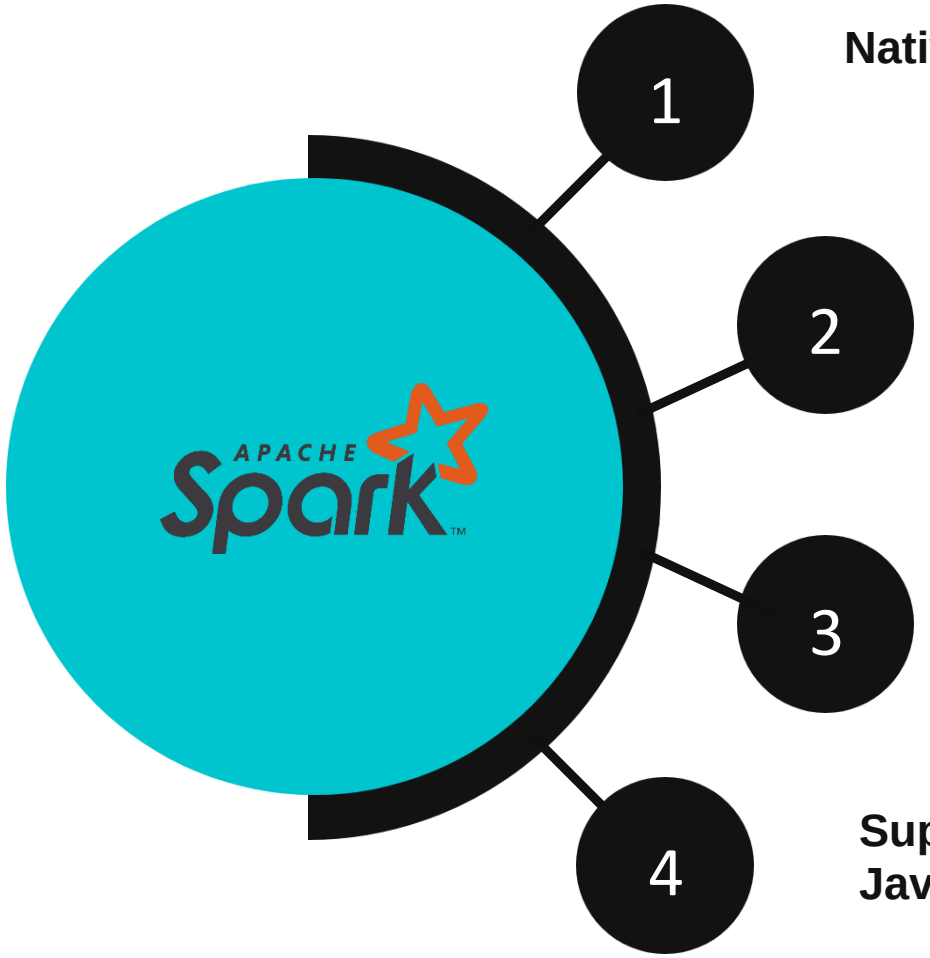
YARN takes care of resource allocation and figures out where and how to runt he job

3. Step Three

YARN stores the result in HDFS

Spark Ecosystem





Native support

Real-time as well as batch processing

Interactive REPL environment

Support for Phthon, Java, Scala and R

RDDs: Basic Building Blocks of Spark

"

RDD are still the fundamental building blocks of Spark

An RDD is a collection of entities – rows, records

RDDs: Basic Building Blocks of Spark

"

All operations in Spark are performed on in-memory objects

Two types of methods:

- Action - Return value
- Apply Transformation

Characteristics of RDDs

01

Partitioned

Split across data nodes in cluster

02

Immutable

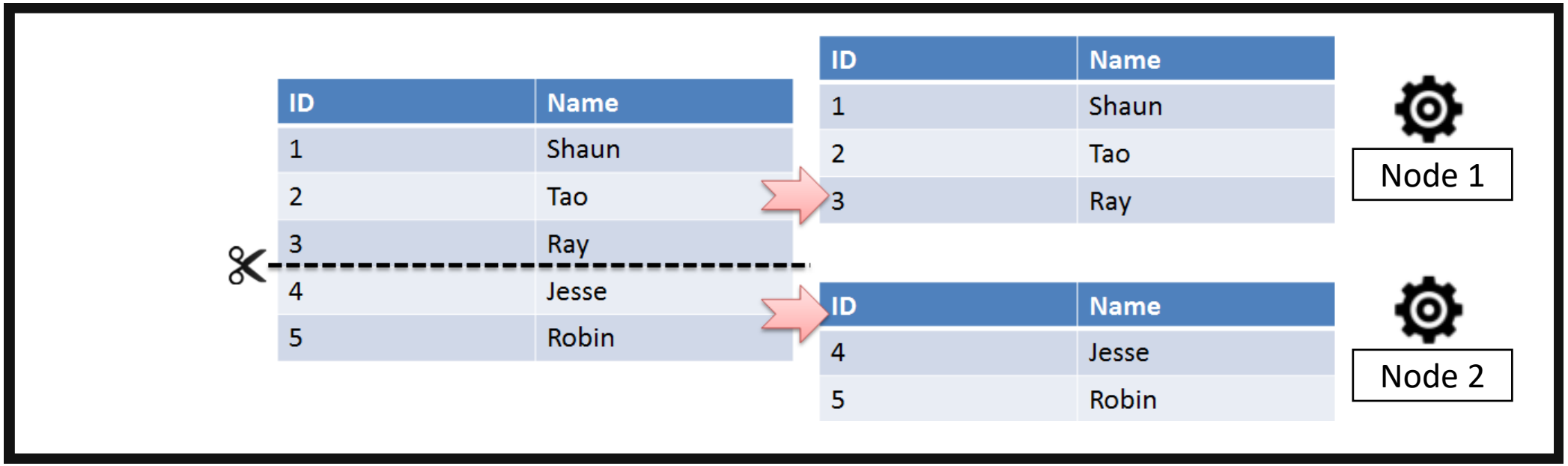
RDDs, once created, cannot be changed

03

Resilient

Can be reconstructed even if a node crashes

Partitioned



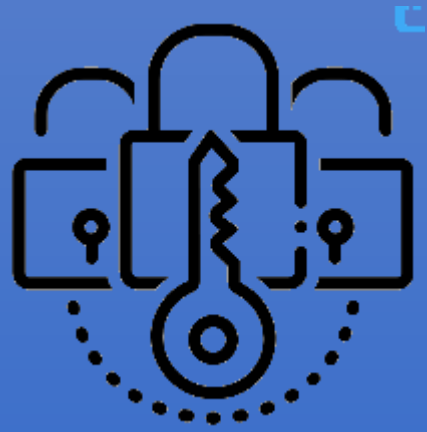
Parallel processing

The main reason for the splitting of data across multiple nodes is parallelization. Data can be processed in parallel on all of these individual nodes.

Data stored in memory for each node in cluster

The most important thing about Spark is that the contents of this RDD are kept entirely in memory across multiple cluster nodes.

- An RDD cannot be mutated
- Only **two types of operations** are permitted on RDD
  - **Transformation**: Transform in to another RDD
  - **Action**: Request a result



RDD: Immutable

RDD: Immutable

- A data set loaded in to RDD
- The user may define a chain of transformations on the dataset.

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

Example: Transformation

1. Load Data
2. Pick only the 3rd column
3. Sort the values

Transformation is executed only when a result is requested

Example: Action

1. The first 10 rows
2. A count
3. A Sum

Request a result using an action

RDD: Lazy Evaluation

- Spark keeps a record of the series of transformations requested by the user.
- It groups the transformations in an efficient way when an Action is requested

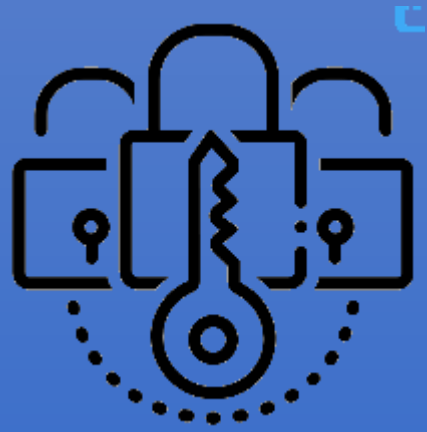
Lazy Evaluation

Spark keeps a record of the series of transformations requested by the user.

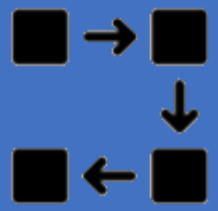
It groups the transformations in an efficient way when an Action is requested.

Transformation: Transform the RDD to create another RDD

Action: Read data from an RDD



RDD: Immutable



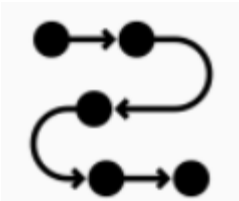
Resilient

Spark keeps a record of the series of transformations requested by the user

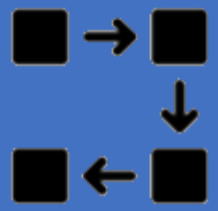
RDD can be created in 2 ways

- Reading a file
- Transforming another RDD

Every RDD keeps track of where it came from



This is RDD's lineage

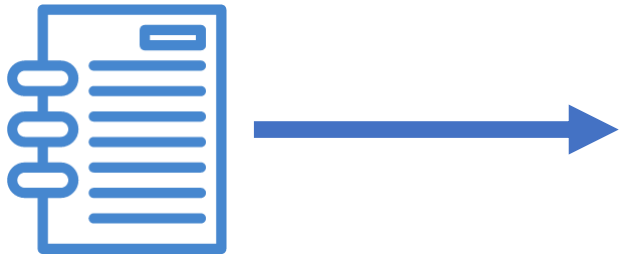


Resilient

Allows RDDs to be reconstructed when nodes crash

Allows RDDs to be Lazily instantiated (materialized) when accessing the results

DataFrame: Data in Rows and Columns

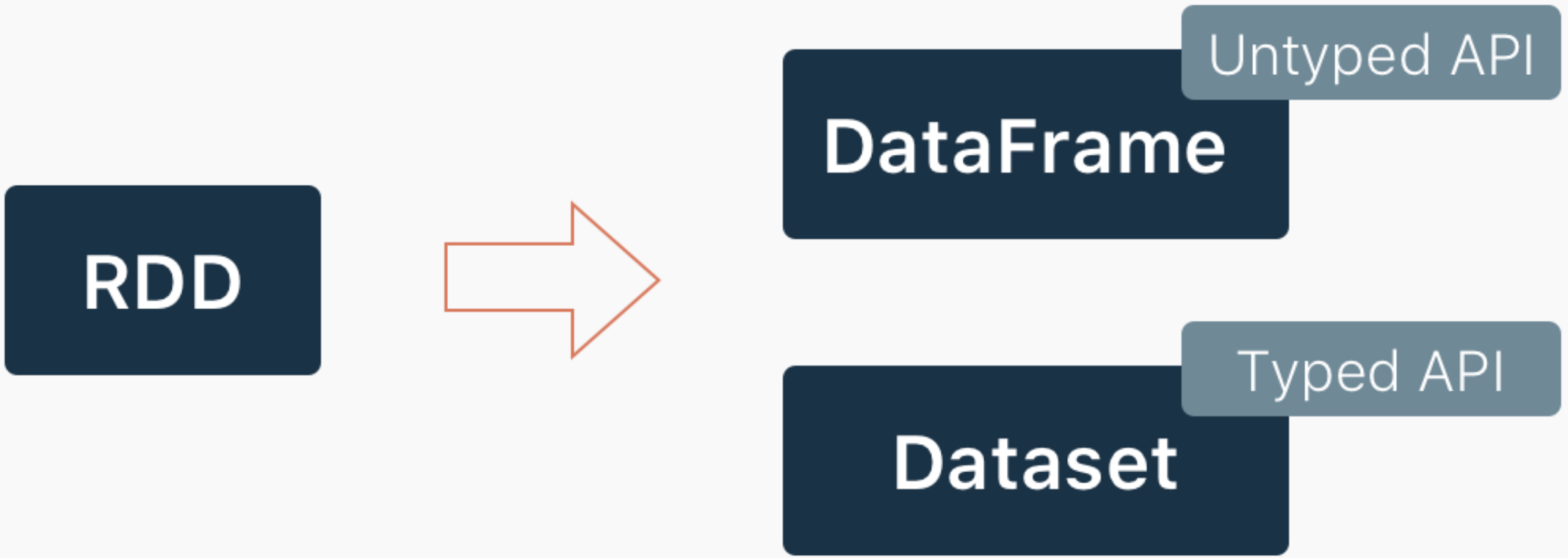


File

First Name	Last Name	Address	City	Age
Mickey	Mouse	123 Fantasy Way	Anaheim	73
Bat	Man	321 Cavern Ave	Gotham	54
Wonder	Woman	987 Truth Way	Paradise	39
Donald	Duck	555 Quack Street	Mallard	65
Bugs	Bunny	567 Carrot Street	Rascal	58
Wiley	Coyote	999 Acme Way	Canyon	61
Cat	Woman	234 Purrfect Street	Hairball	32
Tweety	Bird	543	Itotltaw	28

DataFrame

Apache Spark API



RDDs to Dataset

RDDs

Primary abstraction since initial versions

Immutable and distributed

Strong typing, use of Lambda

No optimized execution

Available in all languages

Datasets

Added to Spark in 1.6

Also immutable and distributed

Also support strong typing, lambdas

Leverage optimizers in recent versions

Present in Scala and Java, not python or R

Datasets to DataFrames

Datasets

Added to Spark in 1.6

Immutable and distributed

No named columns

Extension of DataFrames – OOP interface

Compile time type safety

Present in Scala, Java, not Python, R

DataFrames

Added to Spark in 1.3

Also immutable and distributed

Named columns, like Pandas or R

Conceptually equal to a table in an RDBMS

No type safety at compile time

Available in all languages

Starting Spark 2.0
APIs for Datasets and DataFrames heave merged

Datasets to DataFrames

Datasets

Scala and Java

Datasets of the Row() object in Scala/Java

often called DataFrames

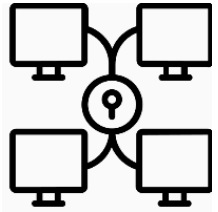
DataFrames

Python, R, Scala, Java

Equivalent to Dataset<Row> in Java or

Dataset[Row] in Scala

What makes Apache Spark difficult to use?

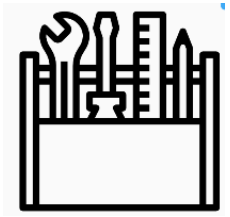


Infrastructure Management

Upgrade Challenge

User Interface

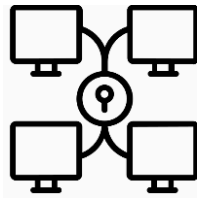
Manual Configuration



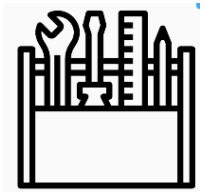
Tooling & Integration Complexity

Difficult to Collaborate on Projects

databricks®



Efficient and Interactive Platform



Tools are available

Integrated and Interactive workspace

User Interface to manage Infrastructure (Scalability, failure recovery, upgrades)

Distributed processing of data

In-memory

Language support

- Scala, Python, SQL, R & Java

Use cases

- Batch & Stream processing
- Machine learning
- Advanced Analytics

An Apache Spark based Unified Analytics Platform, optimized for the cloud

1

Infrastructure Management

Setup and scale an optimized Apache Spark environment

2

Collaboration

Workspace for Data Engineers, Data Scientists & Business Analysts

3

Automation

Run data pipelines and workflows on demand or an schedule

4

Security

Built in access control and enterprise grade security

Microsoft Azure Databricks

A fast, easy, and collaborative Apache Spark™ based analytics platform optimized for Azure

Azure Databricks

Managed 1st Party Azure Service

Native integration with Azure & Its services;
Azure SLA and support

Transparency

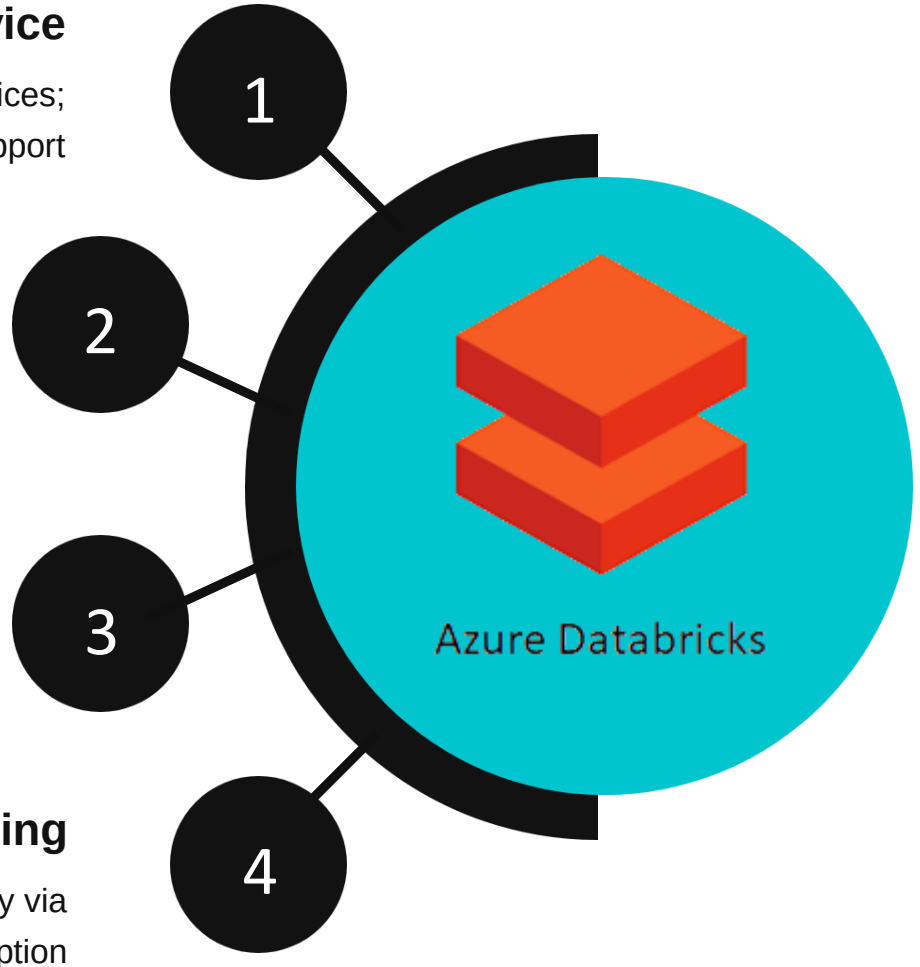
Deploys Databricks workspace and
clusters in customer subscription

Security

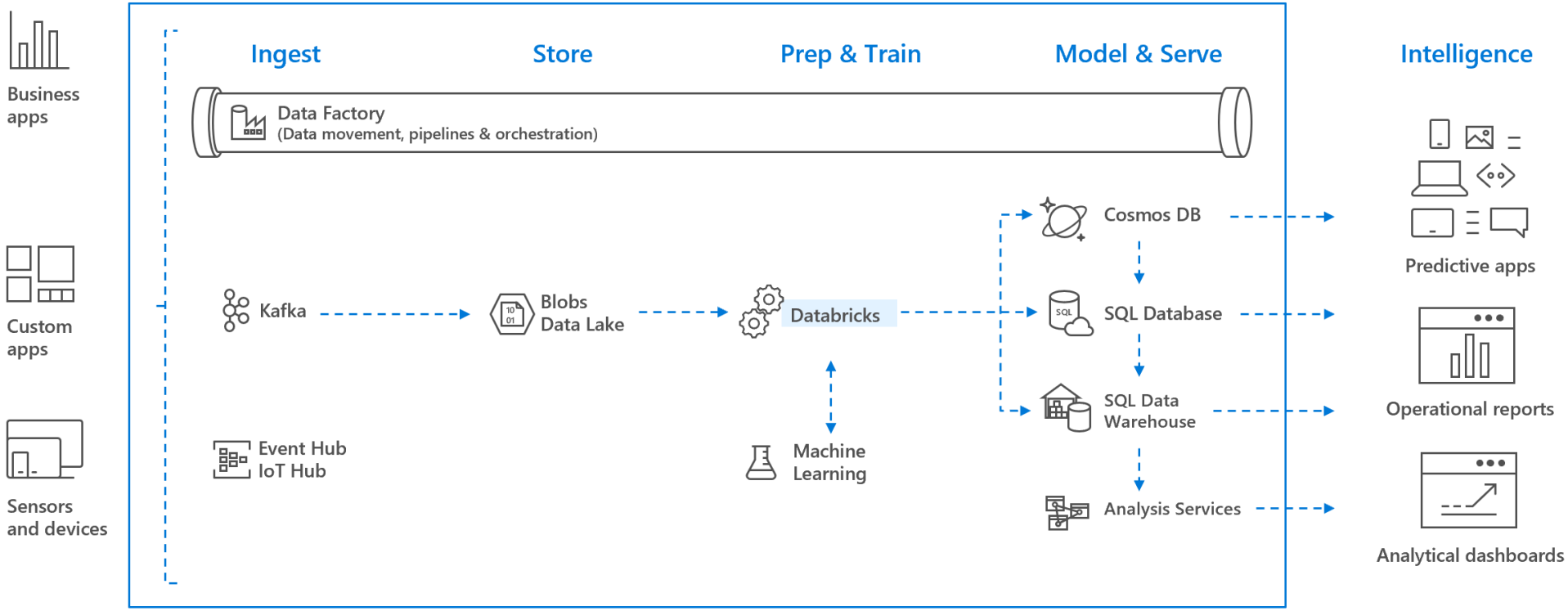
Natively integrates with Azure
Active Directory & Providers RBAC

United Billing

Pay for what you use only via
Azure subscription



Azure Databricks Architecture



Cluster Types

Interactive Cluster

Multiple users interactively analyze the data together

Job Cluster

Created and terminated for running automated jobs

Cluster Types

Interactive Cluster	Job Cluster
Interactively analyze the data	Run automated jobs
Created by users	Auto created when job starts
Manually terminate	Terminates when the job ends
Option to auto terminate, if inactive	Option to auto terminate not applicable
Low execution time	High throughput
Auto scale on demand	Auto scale on demand
Comparatively costly	Comparatively cheaper

Cluster Types

Standard Mode	High Concurrency Mode
Single user	Multiple users
No fault isolation	Fault isolation
No task preemption	Task preemption – fair resource sharing
Each user require separate cluster	Maximum cluster utilization
Supports Scala, Python, SQL, R % Java	Only supports Python, SQL & R

Cluster

There are two types of nodes

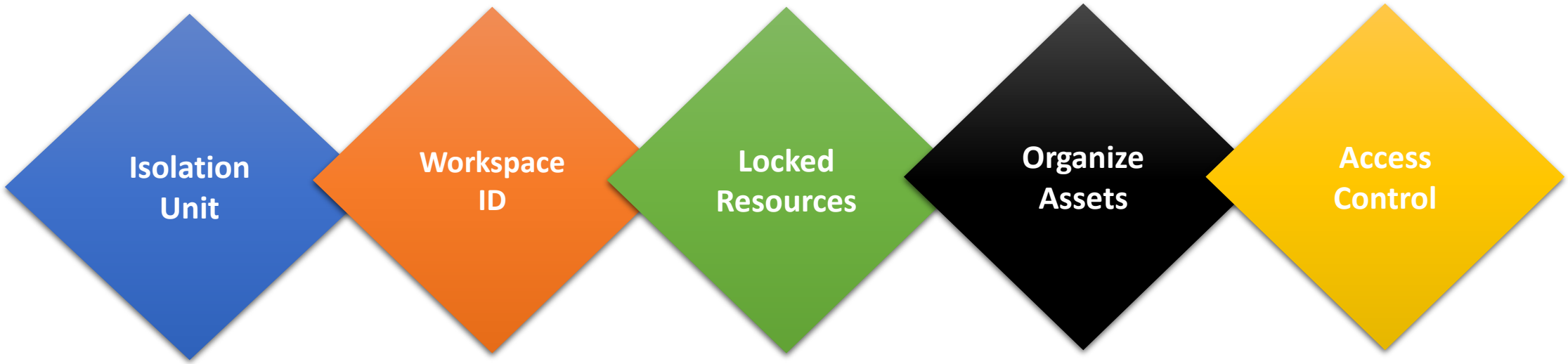
Worker Nodes

Multiple nodes perform data processing task

Driver Node

Distributes task to workers and coordinates execution

Workspace



Each workspace is isolated from others

Each workspace has an identifier

Deployed in control plane and data plane

Notebooks, Libraries, Dashboard etc.

Define access control on all assets

Notebooks

Code in any Spark supported Languages

Invoke notebook from others & pass data

Run directly on clusters or visa jobs

Turn data into graphs or build dashboards

Multiple users can edit and share comments

.

Jobs

- Execution of a notebook or JAR
- It can run immediately or on schedule
- Create job clusters to run jobs
- Each job can have different cluster configuration
- Monitor job runs and setup alerts

- Install 3rd party libraries
- Can be in any supported language
- Import the library into notebook to work
- Scoped at:
  - Cluster
  - Notebook

Libraries

- Create databases and tables inside them
- Table:
  - Collection of structured data
  - Equivalent to DataFrame – perform same operations on table
  - Created using files lying on storage
  - Directly query or write to tables

Database & Tables