Basics to Advanced: Azure Synapse Analytics Hands-On Project

Author: Shanmukh Sattiraju

Azure Synapse Analytics

Author: Shanmukh Sattiraju

Pre-requisites

- No experience needed for Azure Synapse Analytics, we will start from Scratch
- Basic knowledge on Python
- Basic knowledge on SQL language
- Basic Azure cloud knowledge would be a plus

What you'll get from this course?

- More than 18.5 hours of updated learning content
- 50+ most commonly used PySpark transformations
- 45+ PySpark notebooks
- Practical understanding on Delta lake
- Understand Spark Optimization techniques
- Lifetime access to this Course
- Certificate of completion at end of the course

1 - Selection and Filtering

2 - Handling Nulls, Duplicates and aggregations

3 - Data Transformation and Manipulation

4 - MSSpark Utilities

5 - Spark SQL

6 - Join Transformation

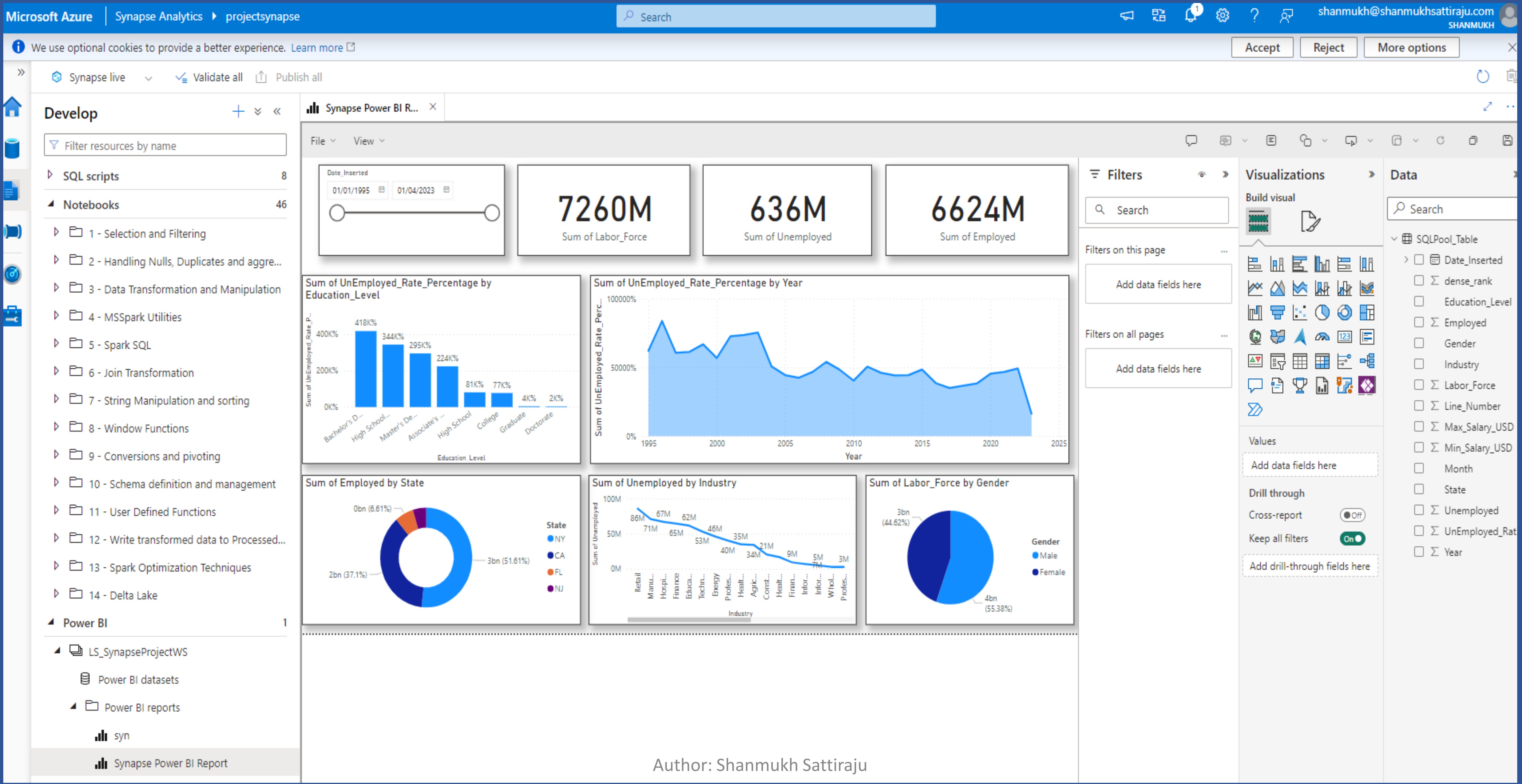
7 - String Manipulation and sorting

8 - Window Functions

9 - Conversions and pivoting

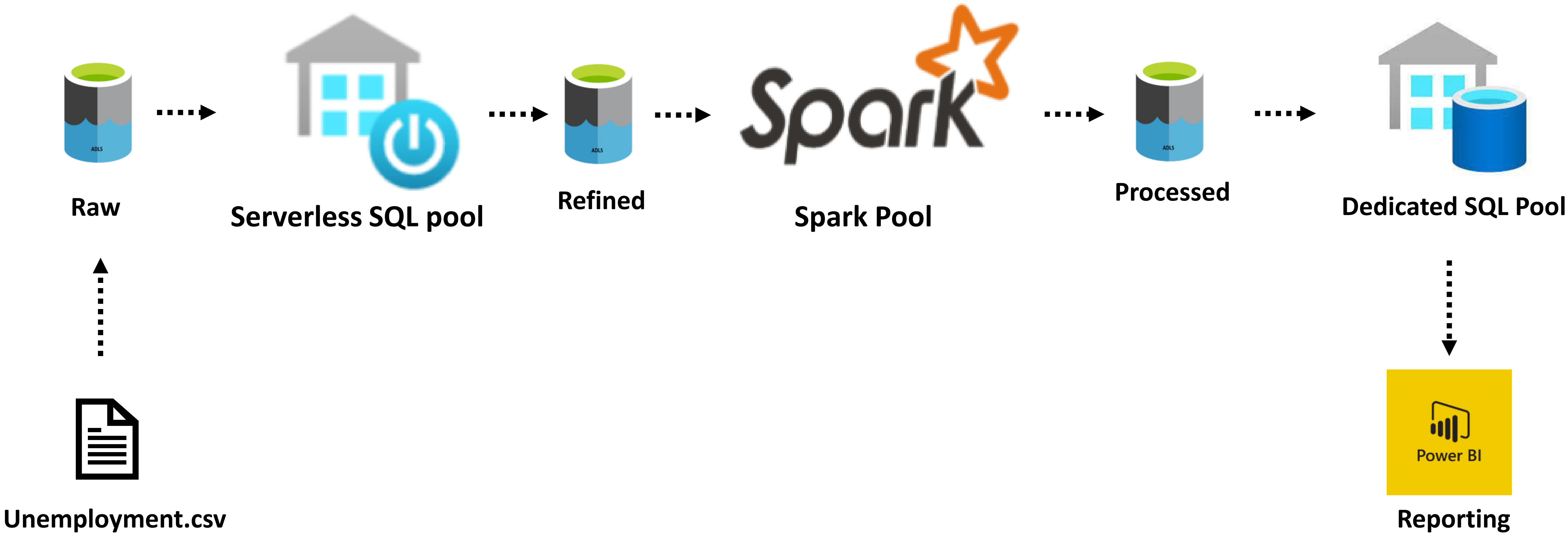
10 - Schema definition and management

11 - User Defined Functions



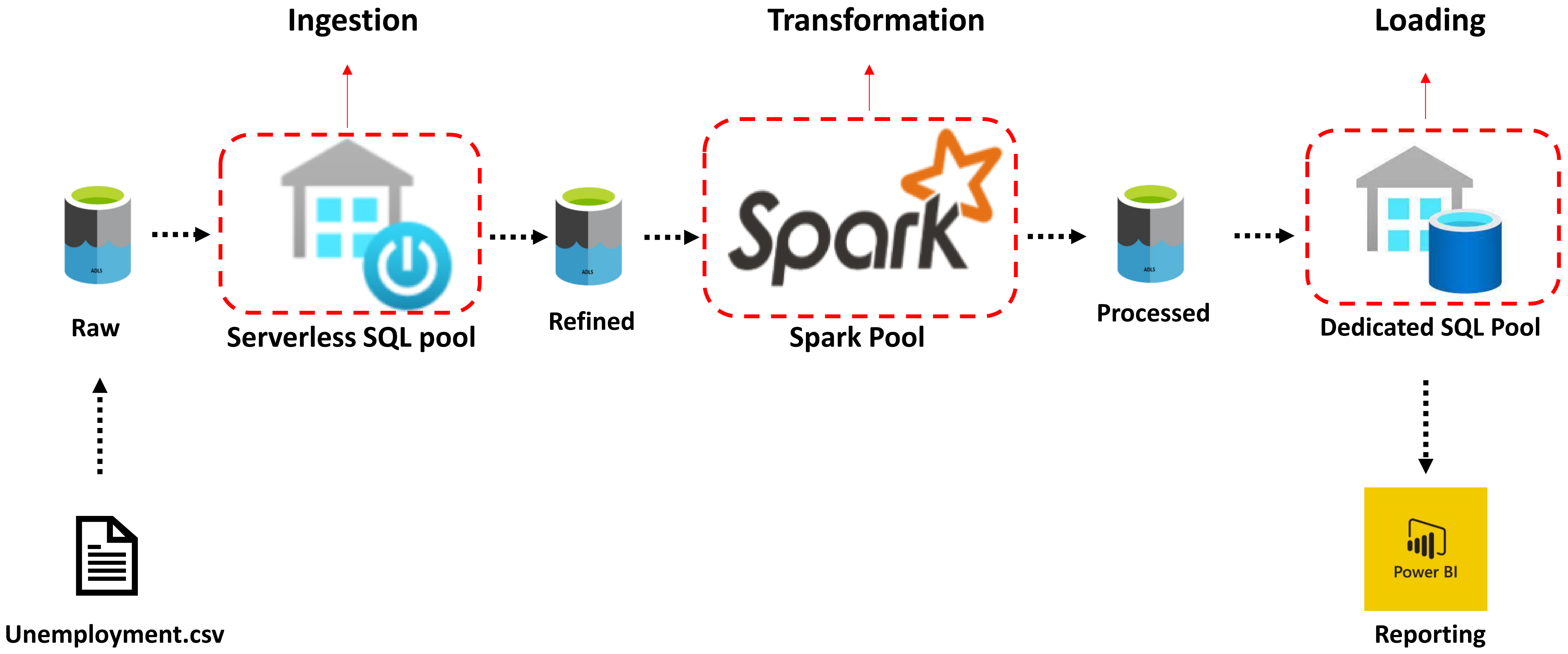
Author: Shanmukh Sattiraju

Project Architecture



Author: Shanmukh Sattiraju

Project Architecture



Author: Shanmukh Sattiraju

Along with Hands-on project:

+

Origin of Azure Synapse Analytics

Author: Shanmukh Sattiraju

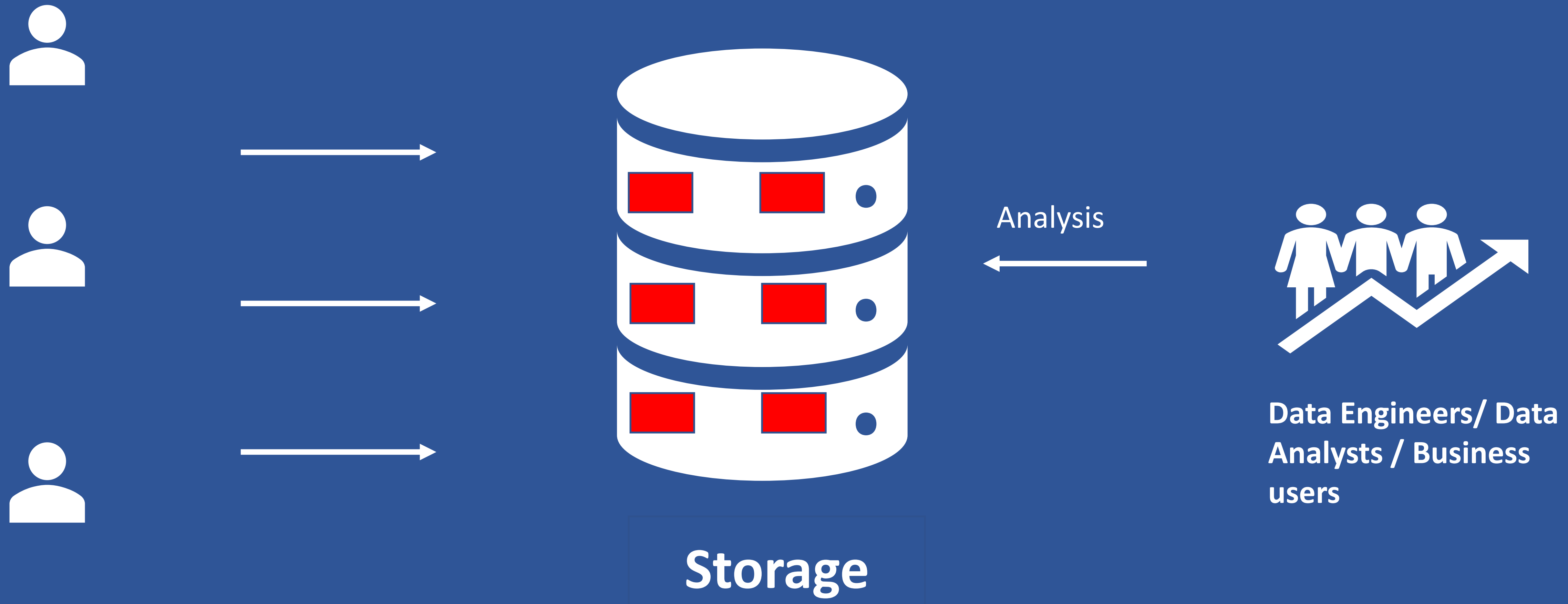
Rise of Data Warehouse

- All started with a need of a separate Transactional system and an Analytical System

Example of a Transactional System

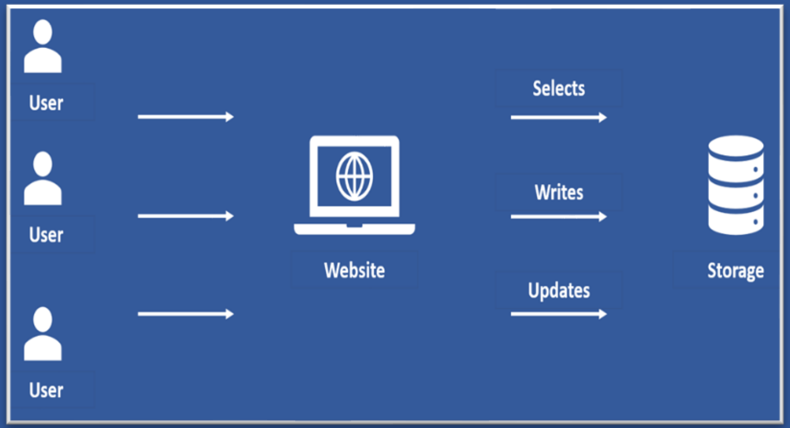
Author: Shanmukh Sattiraju

Performing Analysis on Data



Author: Shanmukh Sattiraju

OLTP vs OLAP



Data Cleansing/ Transformation

ETL

OLTP (Online Transactional Processing)

Can contain NULL values,
Columns that we don't need
to perform analysis

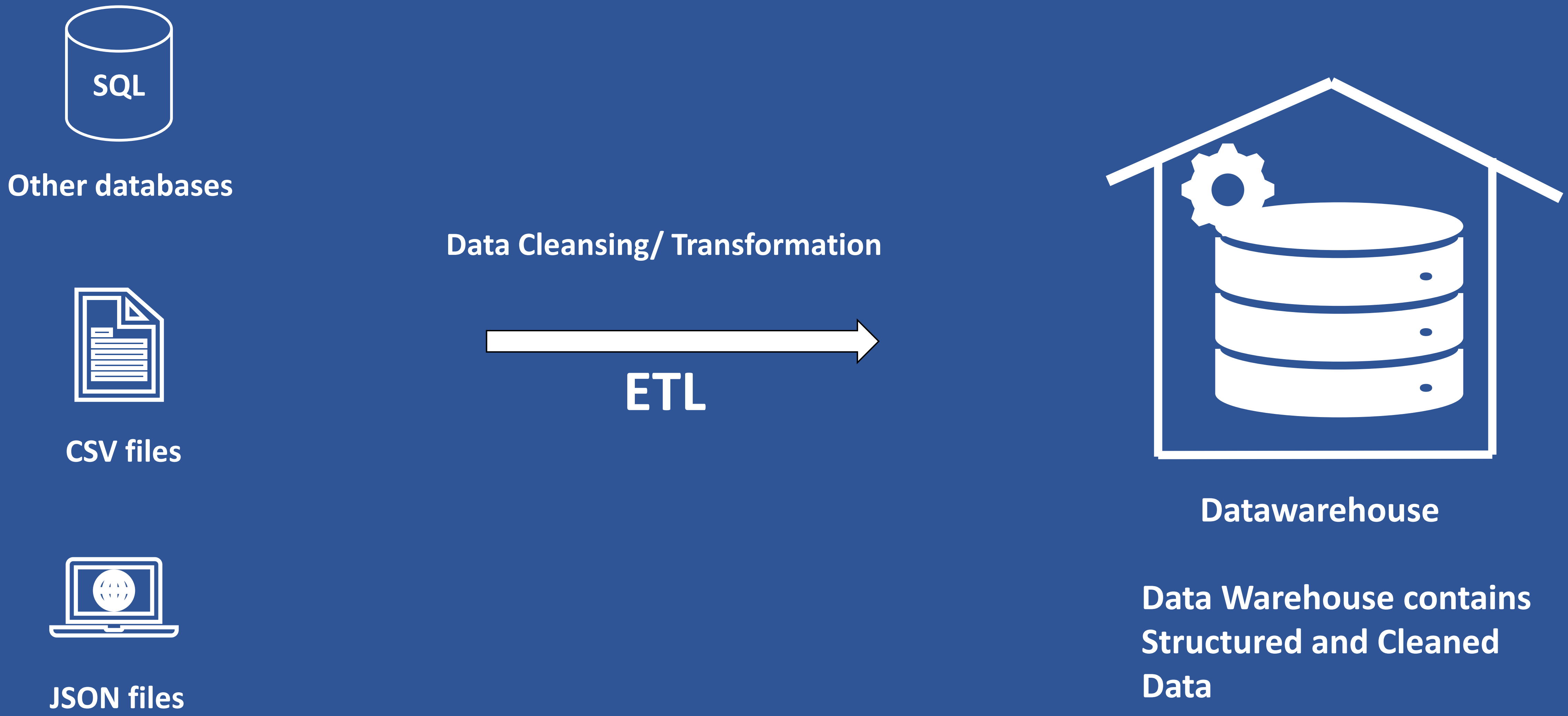
OLAP (Online Analytical Processing)

Data Warehouse contains
Structured and Cleaned
Data

Summary

- OLTP (Online Transactional processing system) is suited for current data which required high reads and writes
- OLAP (Online Analytical processing System) will contains all the historical data
- OLAP is dedicated for performing the analytics on the data which brings us the need to have a **data warehouse**

A typical Datawarehouse

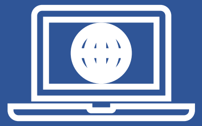


Author: Shanmukh Sattiraju

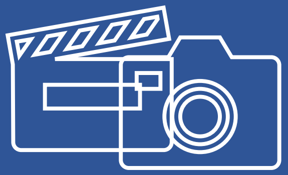
Data lake

Other databases

CSV files



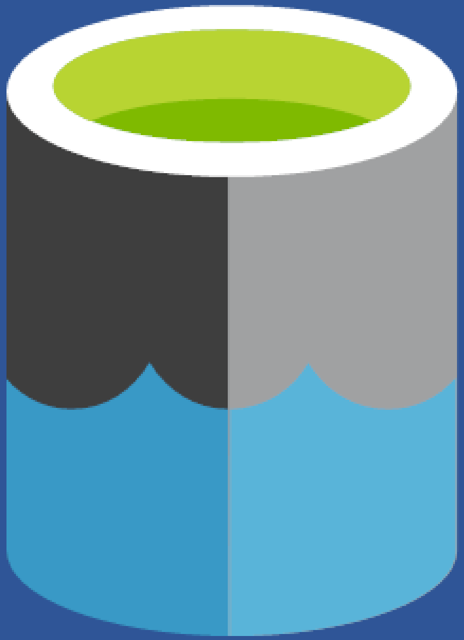
JSON files



Image/ video

Ingestion

ETL

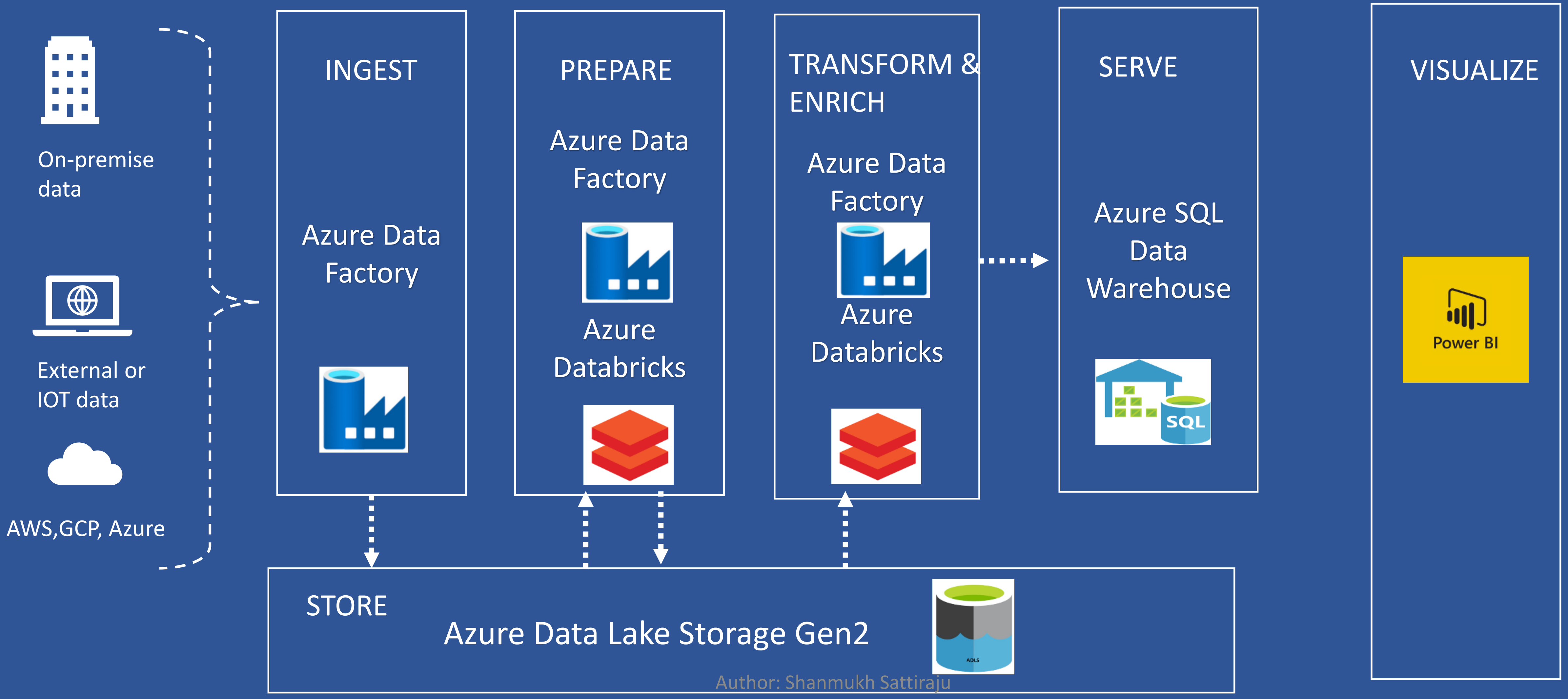


Data Lake

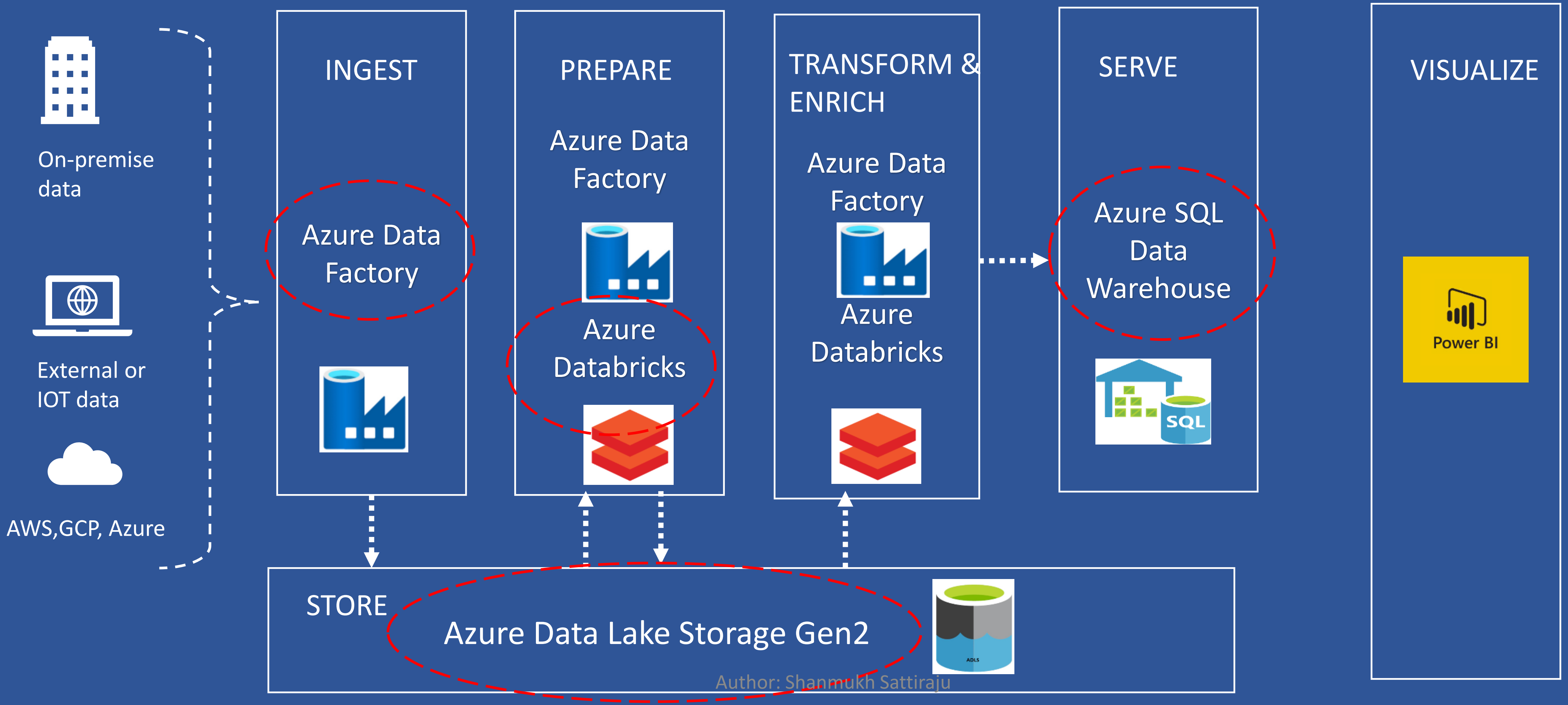
Can store structured, Semi-structed and un-structured data

Author: Shanmukh Sattiraju

Modern Data Warehouse



Problem with Modern Data Warehouse



The Solution – Azure Synapse Analytics

On-premise data

External or IOT data

AWS,GCP, Azure

Azure Synapse Analytics

VISUALIZE

Power BI

STORE

Azure Data Lake Storage Gen2

ADLS

Author: Shanmukh Sattiraju

Components of Azure Synapse Analytics

On-premise data

External or IOT data

AWS,GCP, Azure

Studio

Data Integration | Management | Monitoring | Security

Analytics Pools

SQL Pools | Apache Spark Pools | Data Explorer Pool

Serverless SQL Pool | Dedicated SQL Pool | Spark Pool | Data Explorer Pool

STORE

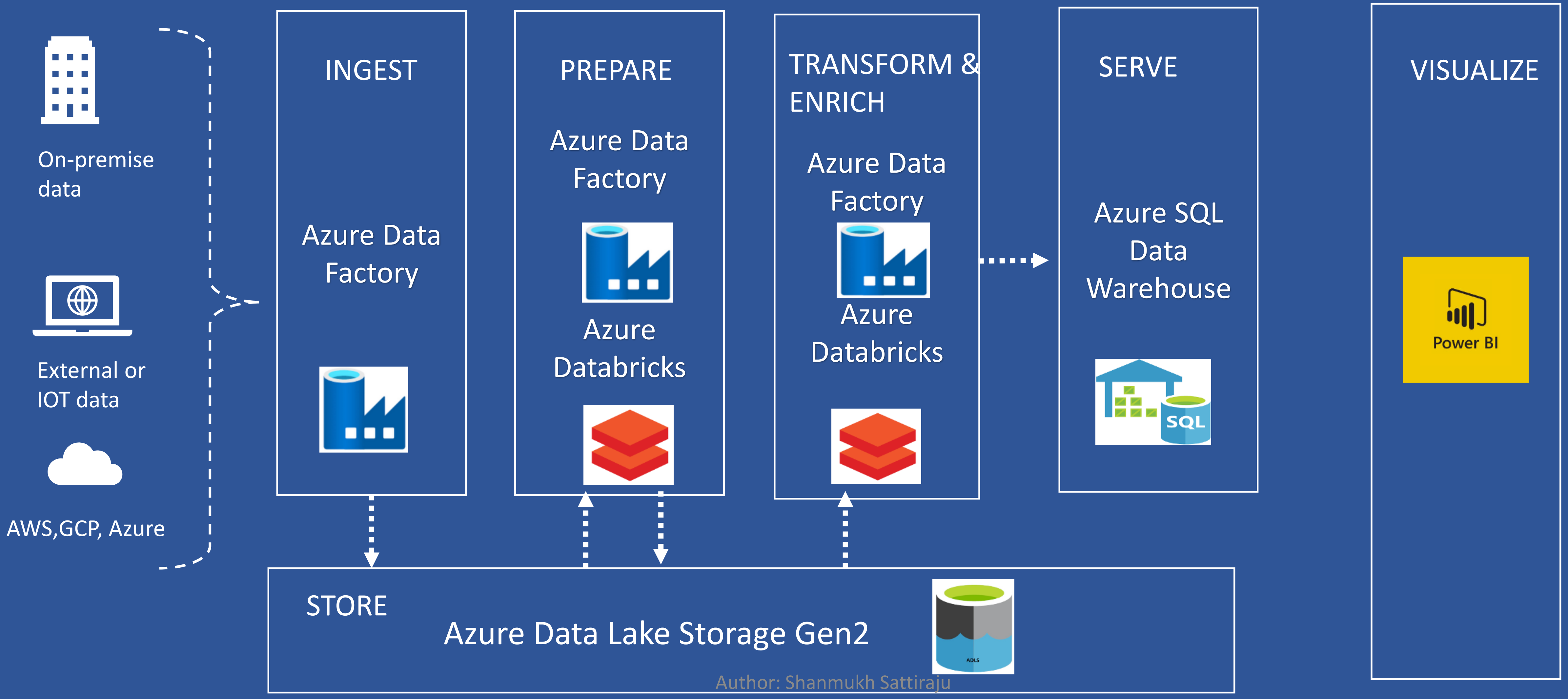
Azure Data Lake Storage Gen2

ADLS

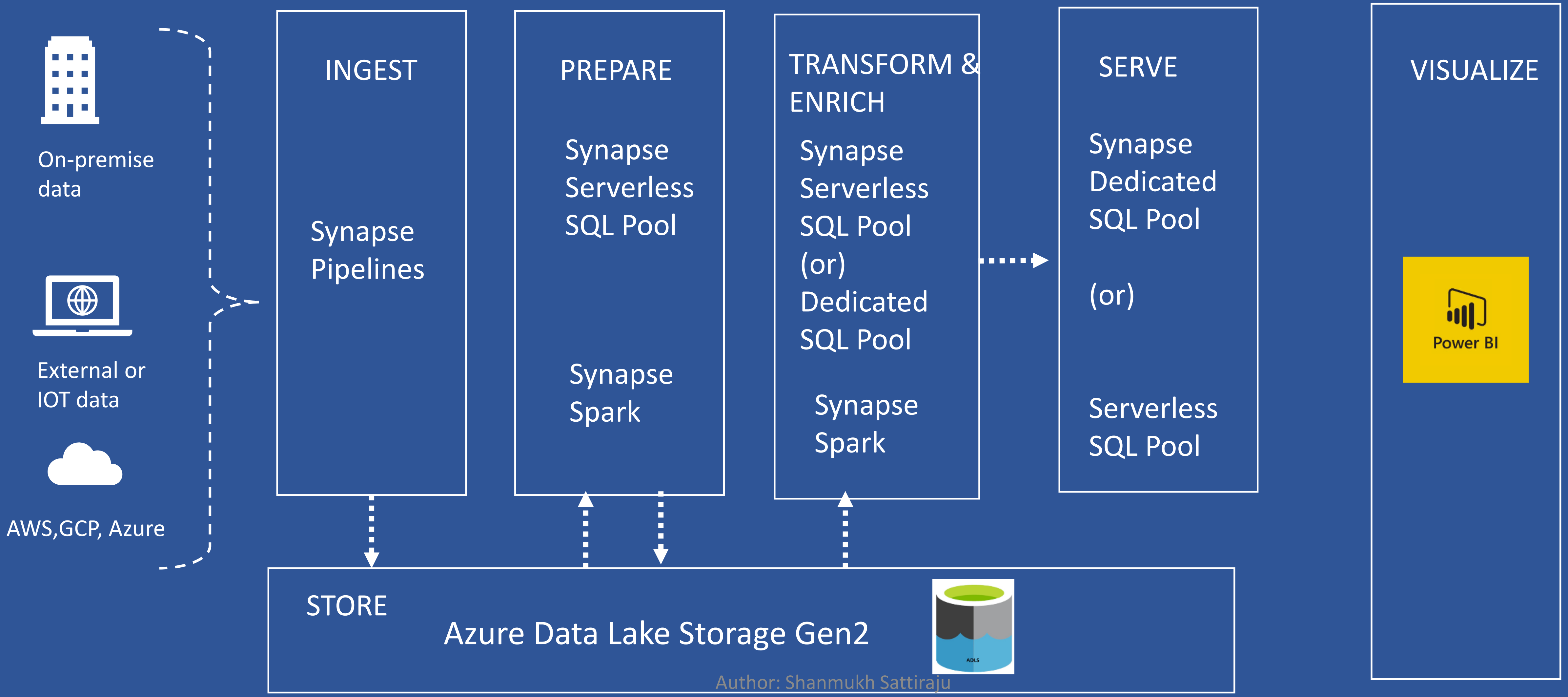
VISUALIZE

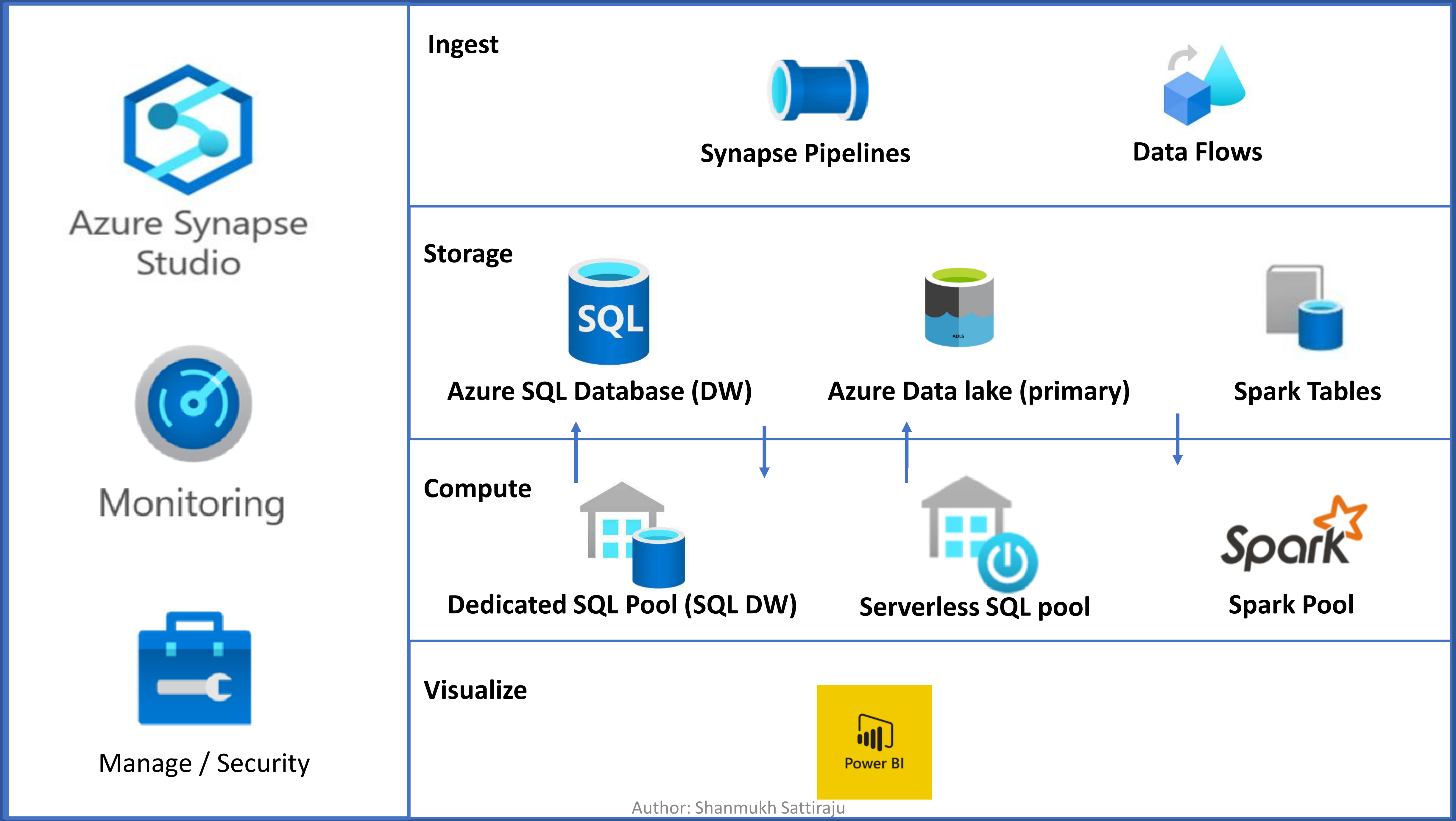
Power BI

Replacing the Modern Data Warehouse



Replacing the Modern Data Warehouse

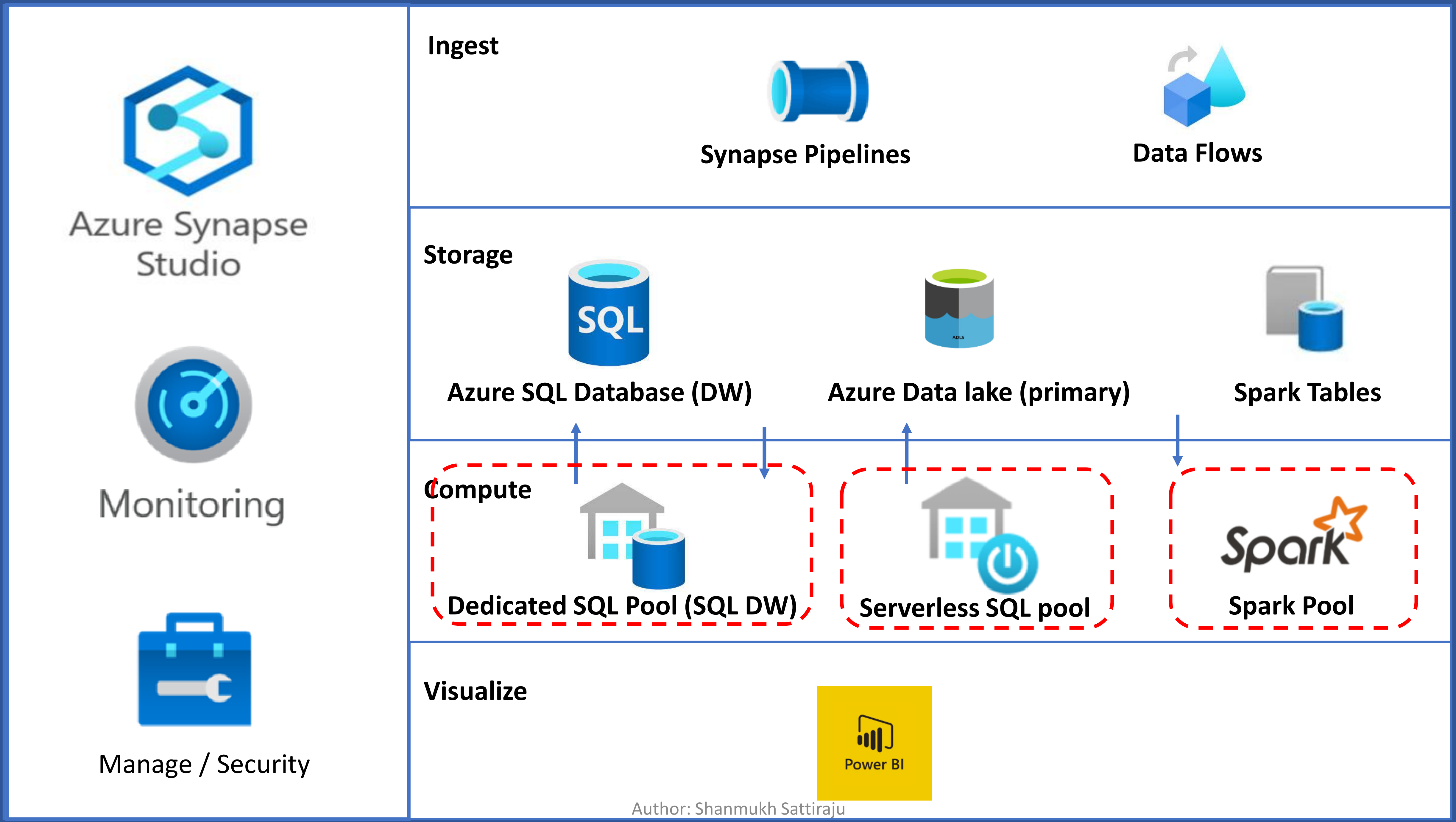



Azure Synapse Studio
Monitoring
Manage / Security
Ingest
Synapse Pipelines
Data Flows
Storage
SQL
Azure SQL Database (DW)
Azure Data lake (primary)
Spark Tables
Compute
Dedicated SQL Pool (SQL DW)
Serverless SQL pool
Spark
Spark Pool
Visualize
Power BI
Author: Shanmukh Sattiraju

Azure Synapse Analytics

Microsoft's Definition:

Azure Synapse is a limitless analytics service that brings **together enterprise data warehousing and Big Data analytics**. It gives you the freedom to query data on your terms, using either **serverless or dedicated resources**—at scale.


Azure Synapse Studio
Monitoring
Manage / Security
Ingest
Synapse Pipelines
Data Flows
Storage
SQL
Azure SQL Database (DW)
Azure Data lake (primary)
Spark Tables
Compute
Dedicated SQL Pool (SQL DW)
Serverless SQL pool
Spark
Spark Pool
Visualize
Power BI
Author: Shanmukh Sattiraju

Environment Setup

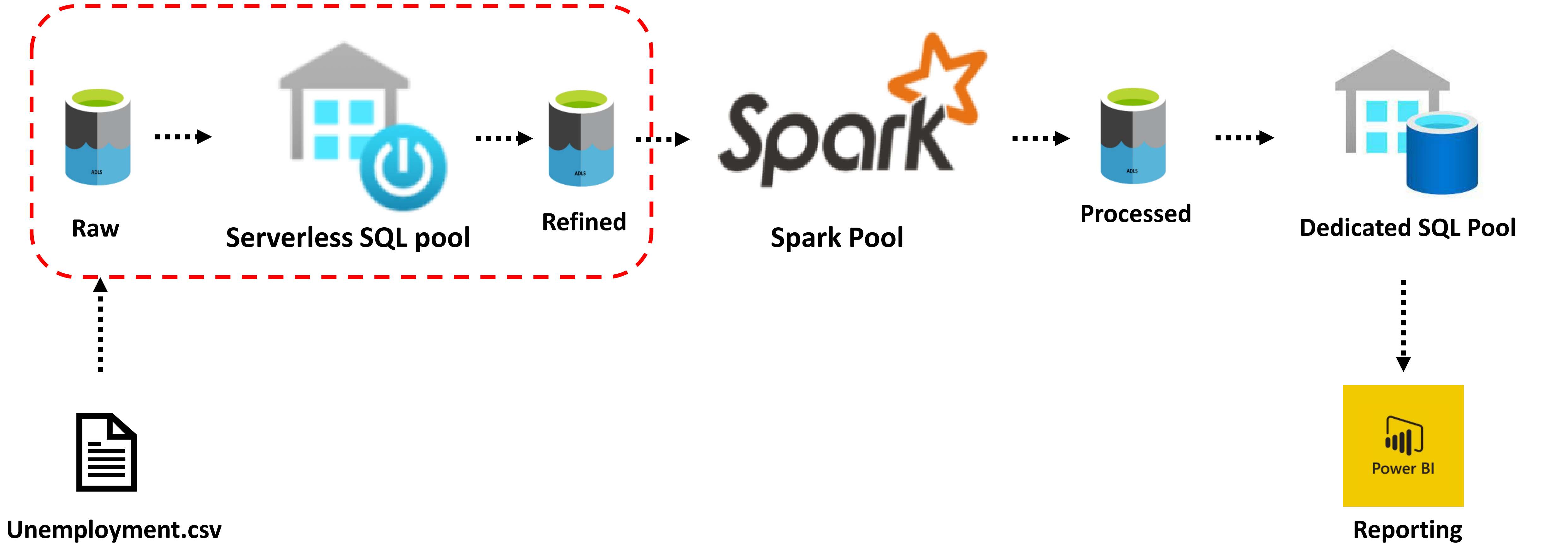
Understanding dataset

Unemployment dataset

Column Name	Description
Line Number	Contains unique number for each line
Year	Year number
Month	Name of the month
State	2 Digit state code
Labor Force	Total population of the state
Employed	Total Employed citizens
Unemployed	Total Unemployed citizens
Unemployment Rate	Contains the value of Unemployment Rate
Industry	What industry the Citizens belong to
Gender	Gender of Citizen
Education Level	Education level of the Citizens
Date Inserted	Date the record got inserted
Other metadata columns	..

Serverless SQL Pool

Project Architecture



Author: Shanmukh Sattiraju

On-demand Serverless SQL pool

Synapse Serverless SQL Pool

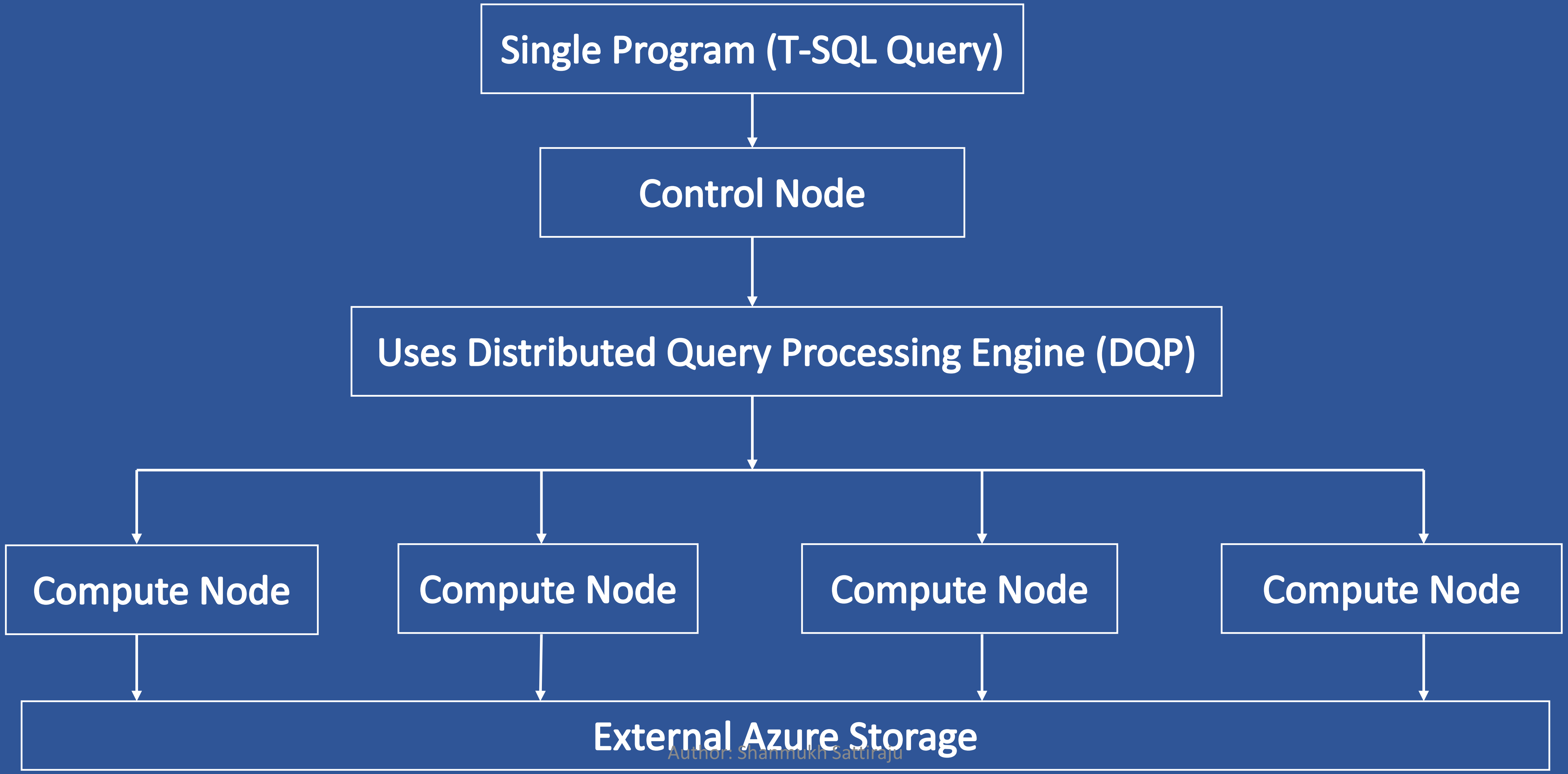
T-SQL Queries

STORE

Azure Data Lake Storage Gen2

Author: Shanmukh Sattiraju

Serverless SQL Pool – Architecture - DQP



Benefits of a Serverless SQL Pool

- You are charged based on how much data in processed by query, better for data exploration
- No underlying Infrastructure
- You can use T-SQL queries to work with your like (same in Dedicated SQL pool)
- You cannot create Tables in Serverless SQL pool because this is not managing any storage on its own
- You can only create external tables or views to work with.

Analysing data with Serverless SQL Pool

- Mostly used for data exploration
- T-SQL to query data
- Pricing

Querying data with Serverless SQL Pool

- OPENROWSET() Function to query data
- We can use OPENROWSET() function with or without DATA_SOURCE parameter
- We will use OPENROWSET() after FROM Clause in T-SQL query
- The OPENROWSET function is not supported in dedicated SQL pool.

```
SELECT
        *
FROM
        OPENROWSET()
```

Mandatory parameters for OPENROWSET()

```
SELECT
    TOP 100 *
FROM
    OPENROWSET(
        *BULK '<storage-path>',
        *FORMAT = 'CSV or PARQUET or DELTA',
        *PARSER_VERSION = '1.0 or 2.0'
    ) AS [result]
```

URL formats for BULK parameter

External Data Source	Prefix	Storage account path
Azure Blob Storage	http[s]	<storage_account>.blob.core.windows.net/path/file
Azure Blob Storage	wasb[s]	<container>@<storage_account>.blob.core.windows.net/path/file
Azure Data Lake Store Gen2	http[s]	<storage_account>.dfs.core.windows.net /path/file
Azure Data Lake Store Gen2	abfs[s]	<container>@<storage_account>.dfs.core.windows.net/path/file

External Data Source	Full path
Azure Blob Storage with https	https://<storage_account>.blob.core.windows.net/path/file
Azure Blob Storage with wasbs	wasbs://<container>@<storage_account>.blob.core.windows.net/path/file
Azure Data Lake Store Gen2	https://<storage_account>.dfs.core.windows.net /path/file
Azure Data Lake Store Gen2	abfss://<container>@<storage_account>.dfs.core.windows.net/path/file

Creating external data source

Data in ADLS Gen2

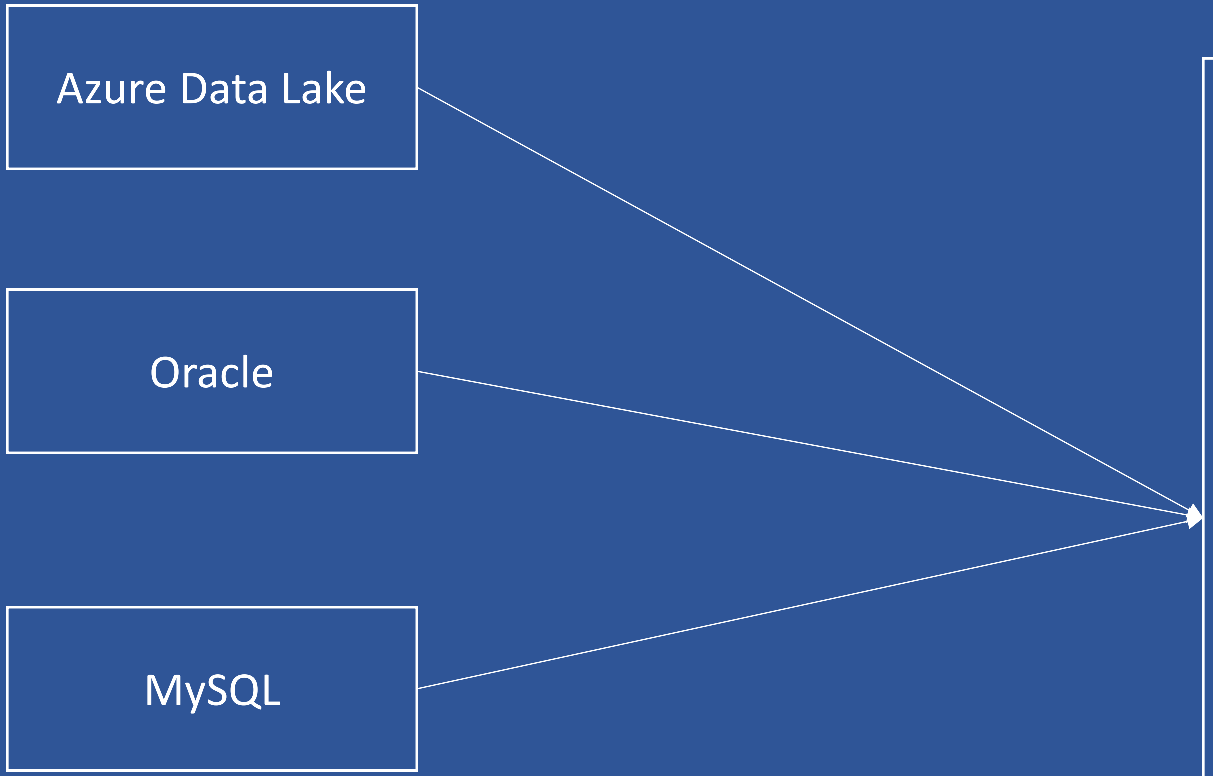
External Data Source
=‘abfss://<container>@<storage_account>.dfs.core.windows.net/’

Serverless SQL DB
(Stores Metadata)

```
SELECT
   TOP 10 *
FROM
   OPENROWSET(
      BULK ‘folder/file',
      DATA_SOURCE =‘<name>',
      FORMAT = 'CSV',
      PARSER_VERSION = '2.0',
      FIRSTROW = 2
   )
```

SQL Script

External Data Source, Credential

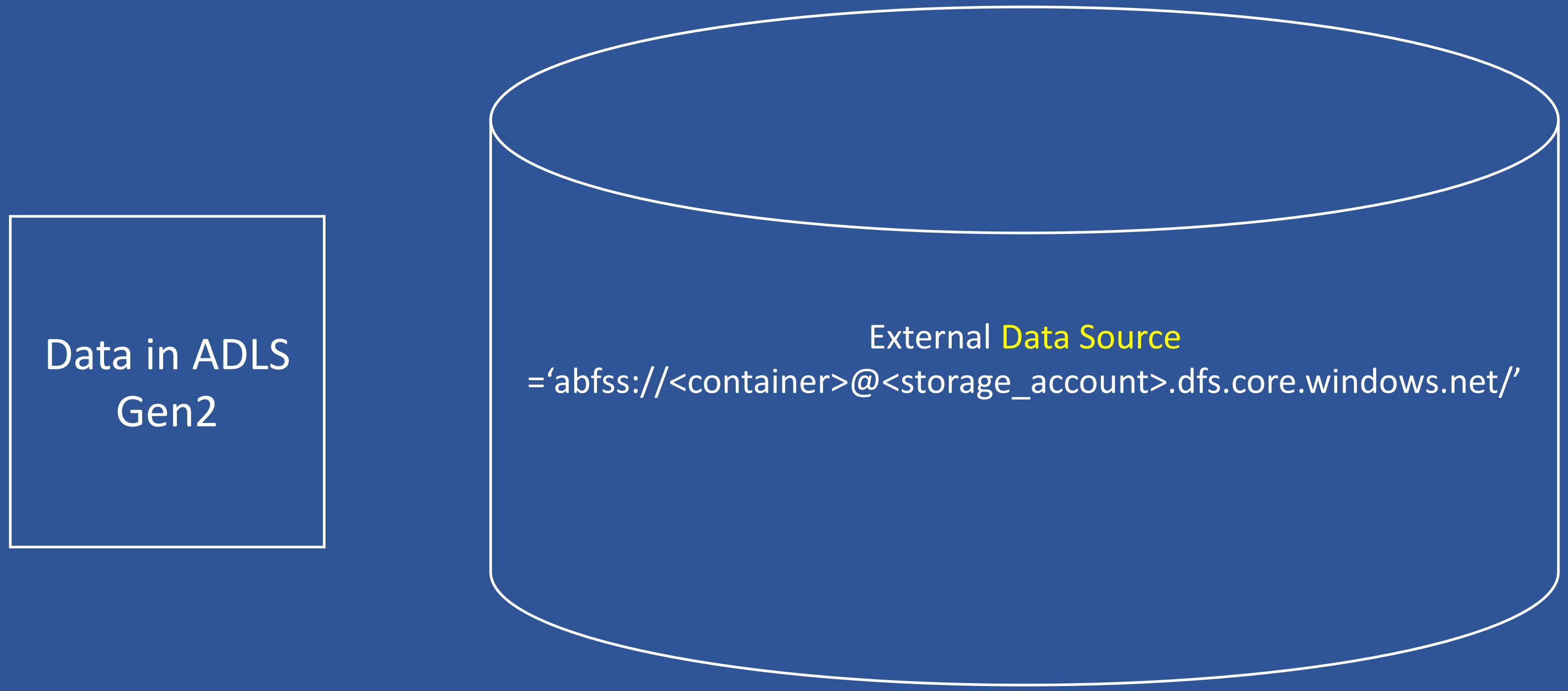



```
CREATE MASTER KEY ENCRYPTION BY PASSWORD
= '<pass>';

CREATE DATABASE SCOPED CREDENTIAL <cred>
WITH IDENTITY = <Type>;

CREATE EXTERNAL DATA SOURCE <Sourcename>
WITH (
    LOCATION = '<Path>',
    CREDENTIAL = <cred>
)
```

Author: Shanmukh Sattiraju

Credential

- A database scoped credential is a record that contains the authentication information that is required to connect to a resource outside SQL Server
- Before creating a database scoped credential, the database must have a master key to protect the credential



Serverless SQL DB
(Stores Metadata)

```
CREATE MASTER KEY ENCRYPTION BY
PASSWORD = <password>';
GO
CREATE DATABASE SCOPED CREDENTIAL
<cred>
WITH IDENTITY = ‘<Identity name>';
GO

CREATE EXTERNAL DATA SOURCE <s_name>
WITH (
   LOCATION =
'abfss://test@storagenew212.dfs.core.window
s.net/',
   CREDENTIAL = <cred>
)
```

SQL Script

Author: Shanmukh Sattiraju

CREATE EXTERNAL FILE FORMAT

- It defines what FILE FORMAT does a file will hold when creating EXTERNAL TABLE (Will be discussed)
- Creating an external file format is a prerequisite for creating an External Table
- The data that is created from EXTERNAL TABLE uses this EXTERNAL FILE FORMAT.
- In short, EXTERNAL FILE FORMAT will specify the actual layout of the data referred by an external table.

Currently supported File Formats:

- DELIMITEDTEXT
- PARQUET
- DELTA (Applies to only Serverless SQL Pools)

CREATE EXTERNAL FILE FORMAT

```
-- Create an external file format for DELIMITED (CSV/TSV) files.
CREATE EXTERNAL FILE FORMAT file_format_name
WITH (
        FORMAT_TYPE = DELIMITEDTEXT
    [ , FORMAT_OPTIONS ( <format_options> [ ,...n  ] ) ]
    [ , DATA_COMPRESSION = {
           'org.apache.hadoop.io.compress.GzipCodec'
        }
     ]);

<format_options> ::=
{
    FIELD_TERMINATOR = field_terminator
    | STRING_DELIMITER = string_delimiter
    | FIRST_ROW = integer -- ONLY AVAILABLE FOR AZURE SYNAPSE ANALYTICS
    | DATE_FORMAT = datetime_format
    | USE_TYPE_DEFAULT = { TRUE | FALSE }
    | ENCODING = {'UTF8' | 'UTF16'}
    | PARSER_VERSION = {'parser_version'}
}
```

CREATE EXTERNAL FILE FORMAT

```
--Create an external file format for PARQUET files.

CREATE EXTERNAL FILE FORMAT file_format_name
WITH (
        FORMAT_TYPE = PARQUET,
        DATA_COMPRESSION = {
       'org.apache.hadoop.io.compress.SnappyCodec'
      | 'org.apache.hadoop.io.compress.GzipCodec' }
         );
```

```
-- Create an external file format for delta table files (serverless SQL pools in Synapse analytics and SQL Server 2022).

CREATE EXTERNAL FILE FORMAT file_format_name
WITH (
        FORMAT_TYPE = DELTA
      );
```

Create External Table As Select (CETAS)

CREATE EXTERNAL TABLE ext_table

WITH (

LOCATION = ‘test/extfile/’

DATA_SOURCE =

FILE_FORMAT =

) AS

```
SELECT
    TOP 10 [data].Year, [data].State
    FROM
      OPENROWSET(
          BULK
  'abfss://raw@datalake11212.dfs.core.windows.net/Unemployment.csv',
          FORMAT = 'CSV',
          PARSER_VERSION = '2.0',
          HEADER_ROW = TRUE
              ) AS [data]
```

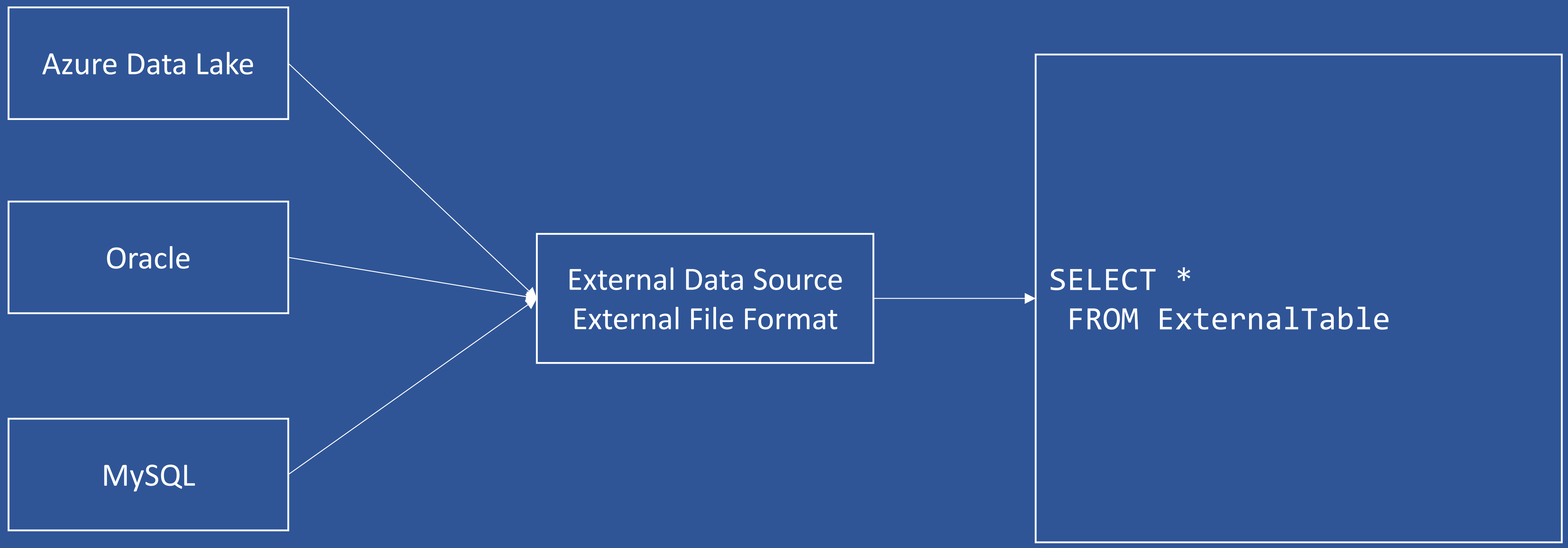
Initial Transformation

EXTERNAL DATA SOURCE = @Refined

EXTERNAL FILE FORMAT = .parquet

EXTERNAL TABLE

External Table



CREATE EXTERNAL TABLE

- With **Synapse** SQL, you can **use external tables** to read external data using dedicated SQL pool or serverless SQL pool.
- External tables is a table-like object in azure synapse that represents structure and schema of data stored in external data sources.
- External tables acts as a reference point to data that is stored externally in storage (Azure data lake storage or Azure blob storage)
- This helps to control the access to external data.
- In parallel to creating external table the data will be created in the storage that you wish.
- Options needed to create an external table:
  - LOCATION = Where you want to store the data that is created by external table
  - DATA_SOURCE = Holds the path of the storage
  - FILE_FORMAT = Format that you wish to have for the data that is created by External table

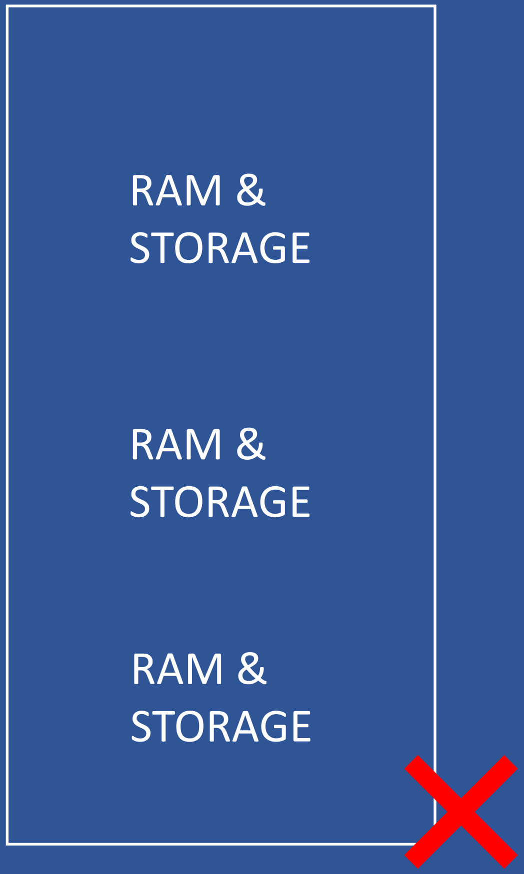
Serverless SQL pool initial Transformation

Author: Shanmukh Sattiraju

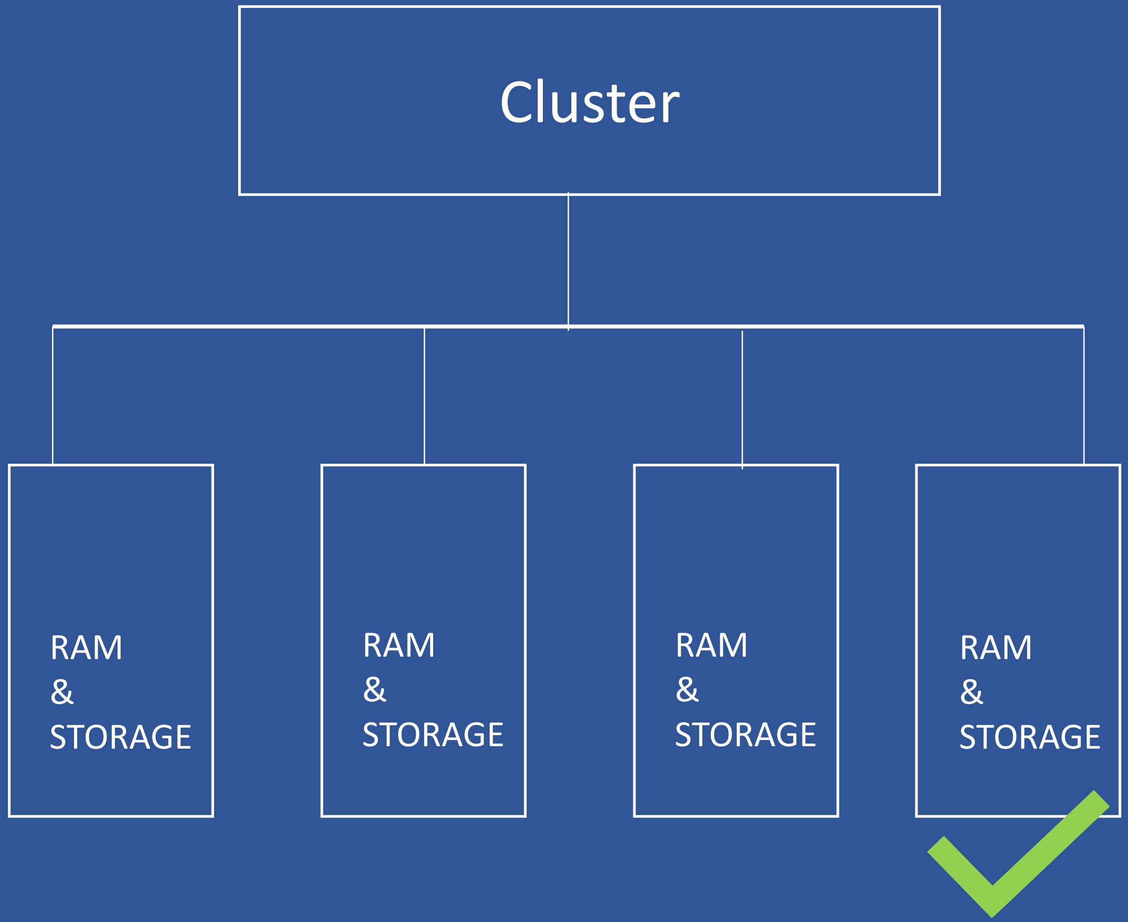
History and Data processing before Spark

Author: Shanmukh Sattiraju

Big data approach

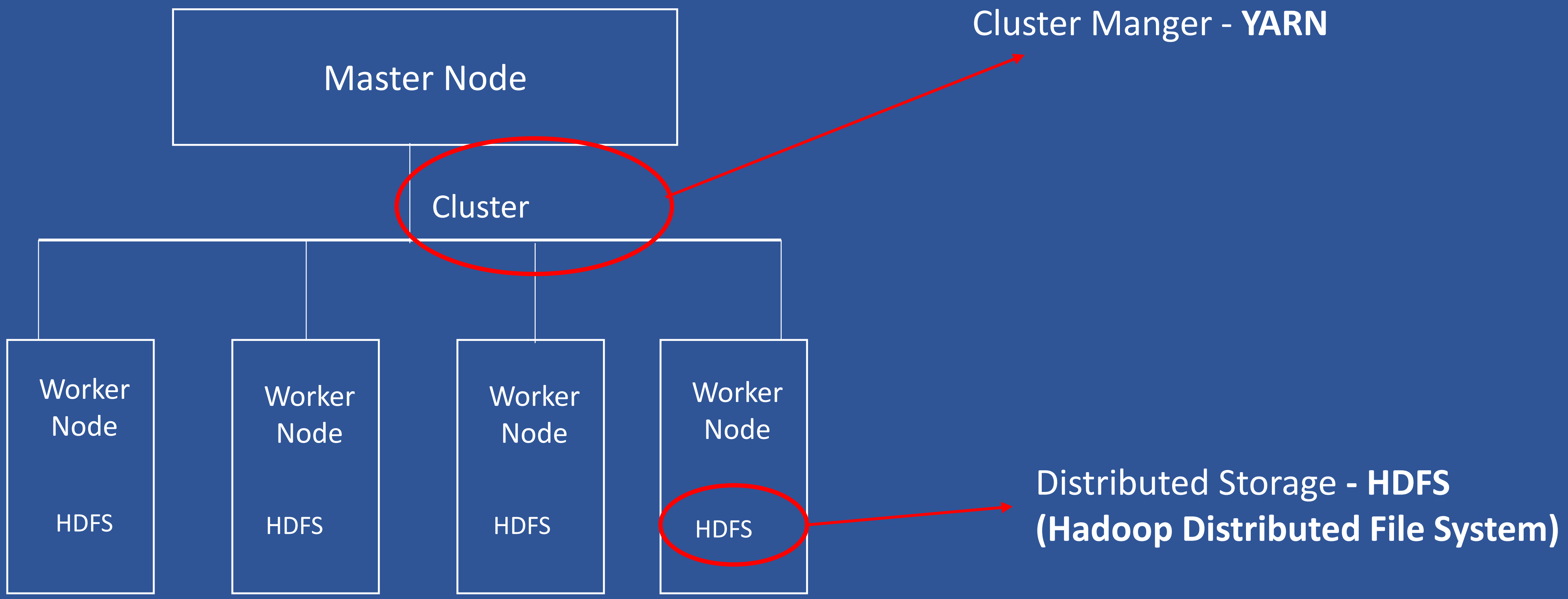


Single Computer for Data Storage and Processing (Monolithic)



Distributed Approach (adding multiple machines to achieve parallel processing)

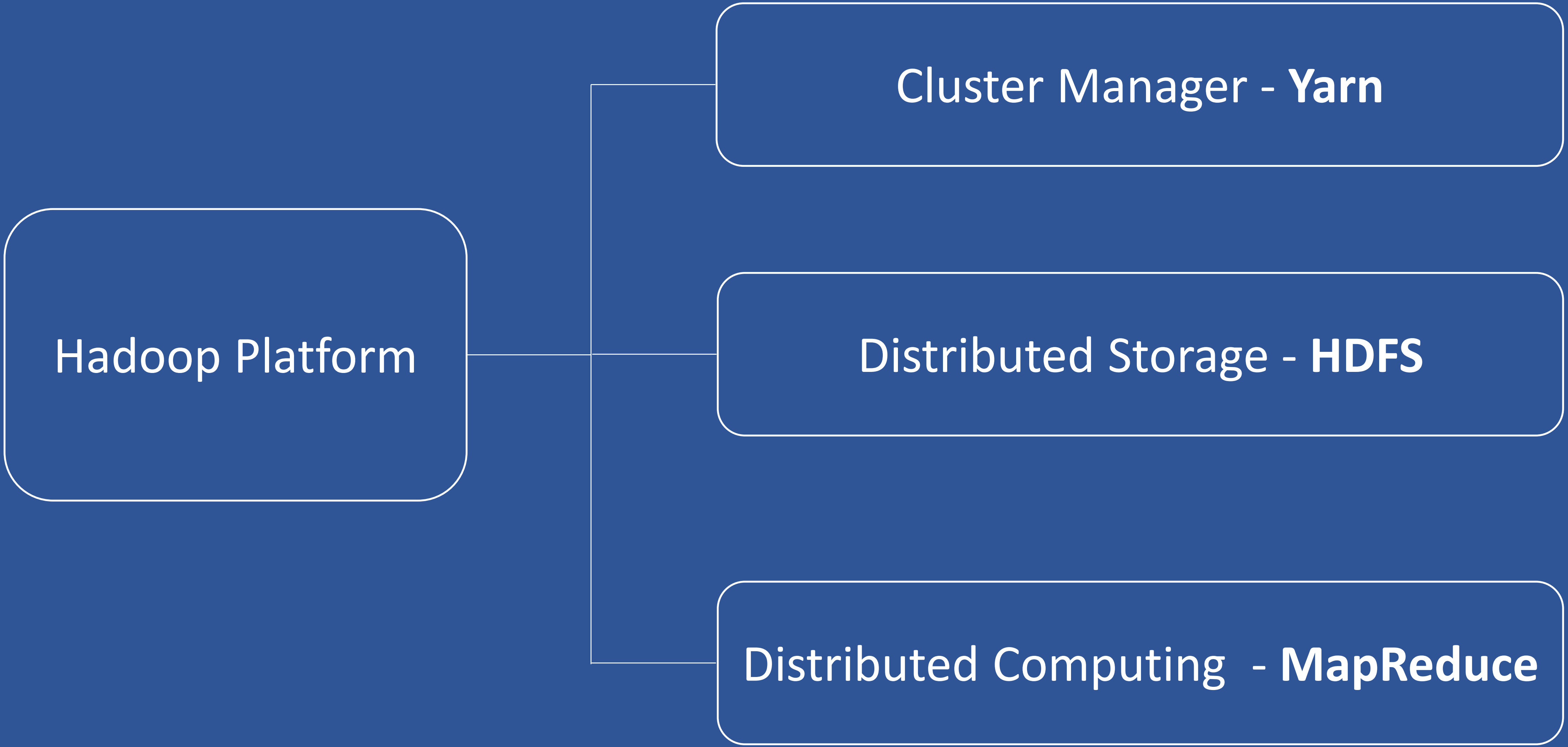
Hadoop Platform



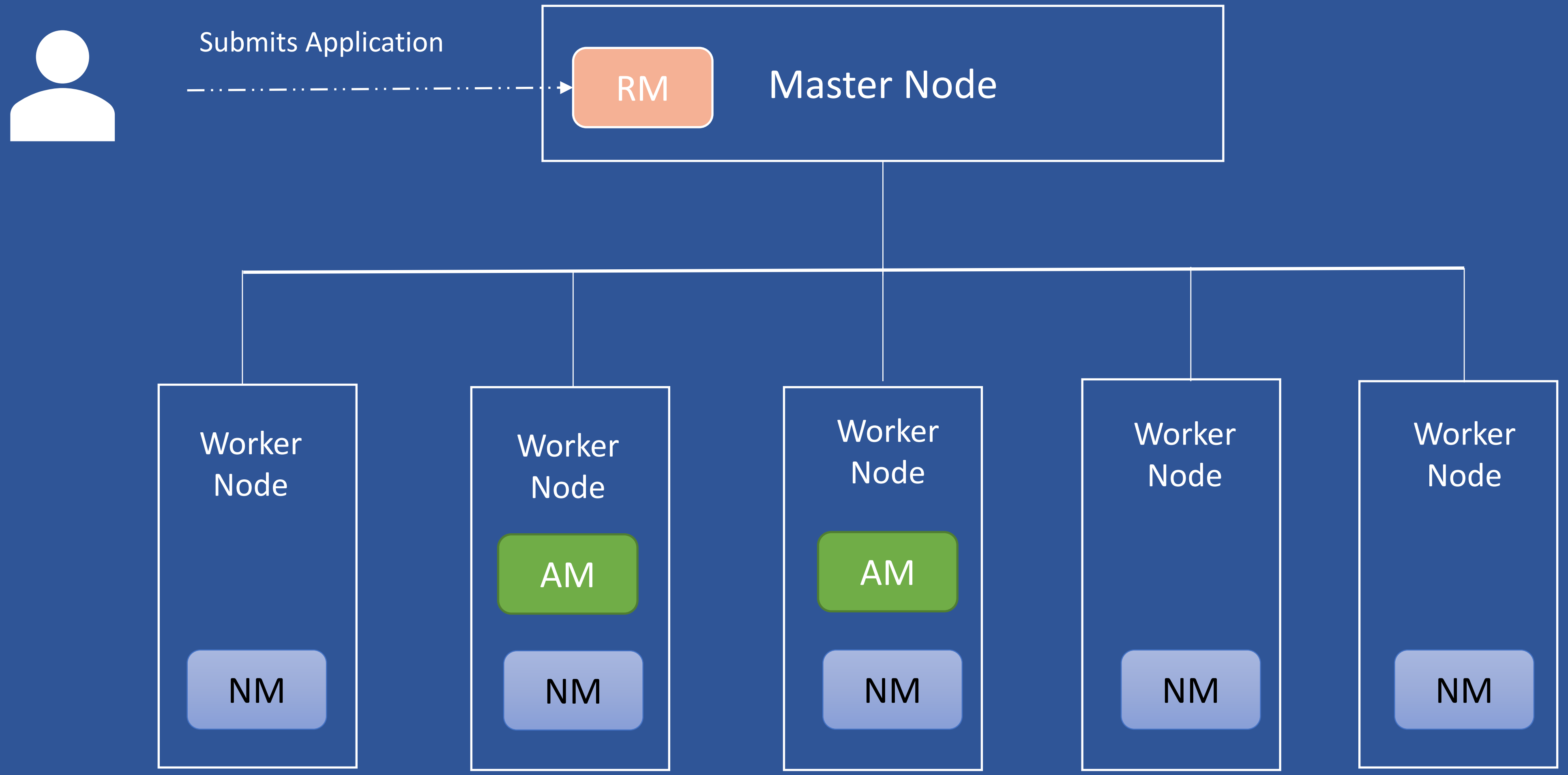
Distributed Approach (adding multiple machines to achieve parallel processing)

Author: Shanmukh Sattiraju

Hadoop Platform

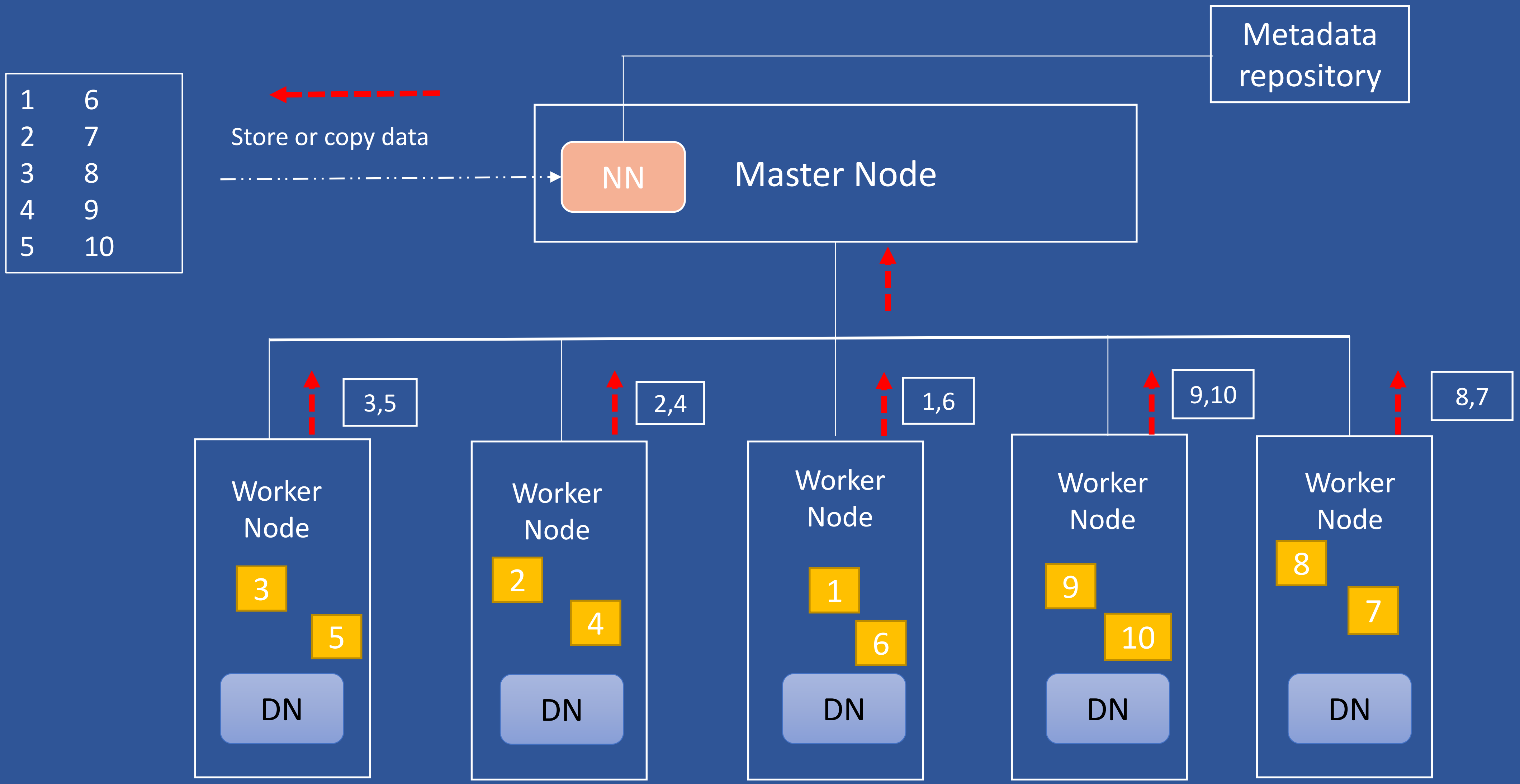


YARN – Yet Another Resource Negotiator



Author: Shanmukh Sattiraju

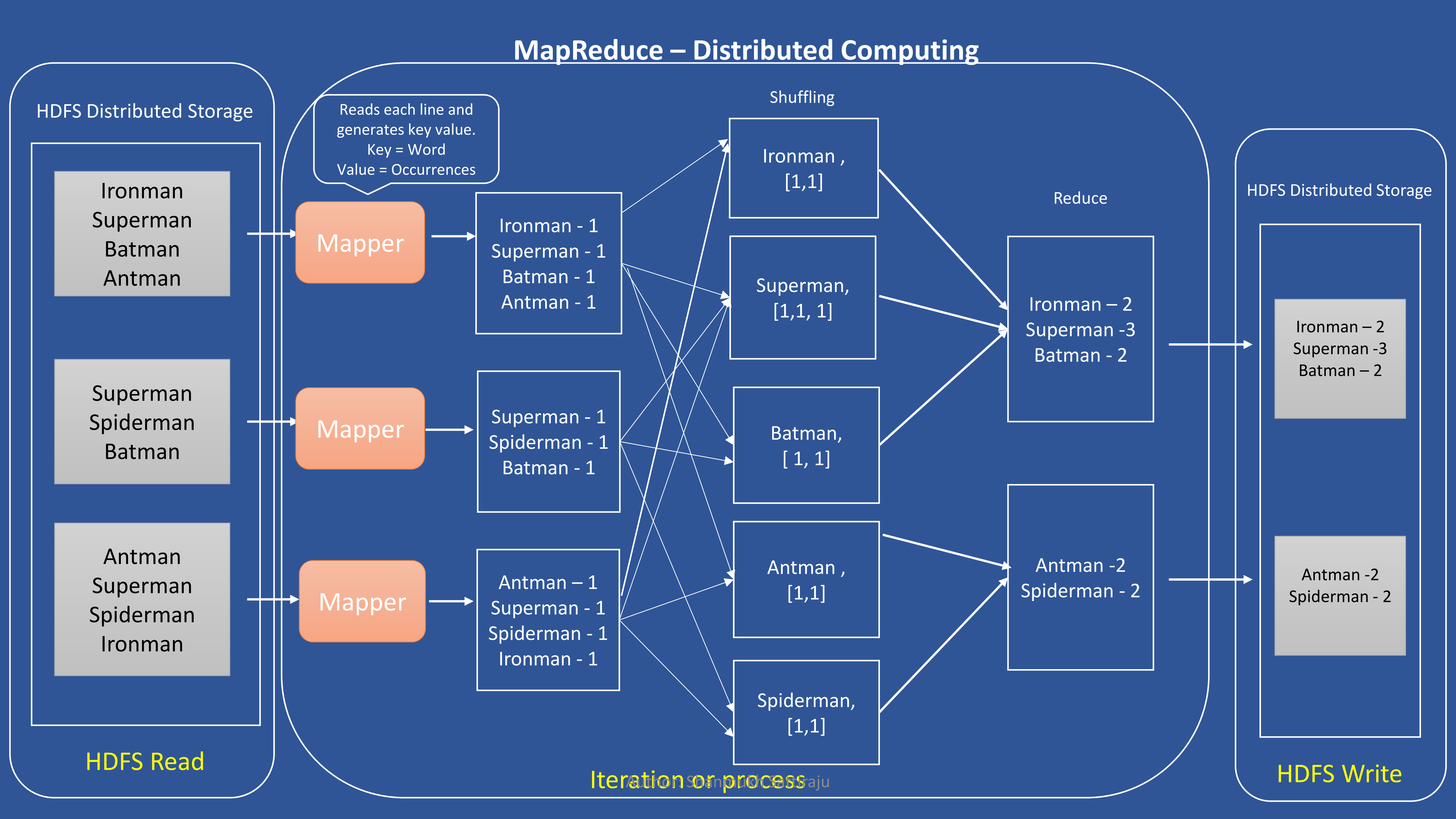
HDFS – Distributed Storage



Author: Shanmukh Sattiraju

Map/Reduce – Distributed Computing

Author: Shanmukh Sattiraju



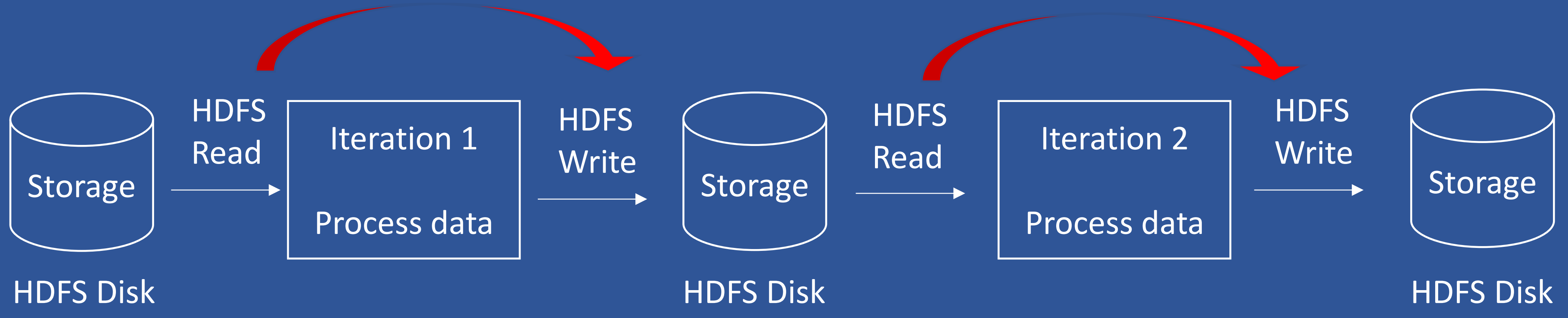
MapReduce – Distributed Computing
HDFS Distributed Storage
Ironman
Superman
Batman
Antman
Superman
Spiderman
Batman
Antman
Superman
Spiderman
Ironman
HDFS Read
Reads each line and generates key value.
Key = Word
Value = Occurrences
Mapper
Mapper
Mapper
Ironman - 1
Superman - 1
Batman - 1
Antman - 1
Superman - 1
Spiderman - 1
Batman - 1
Antman – 1
Superman - 1
Spiderman - 1
Ironman - 1
Shuffling
Ironman , [1,1]
Superman, [1,1, 1]
Batman, [1, 1]
Antman , [1,1]
Spiderman, [1,1]
Reduce
Ironman – 2
Superman -3
Batman - 2
Antman -2
Spiderman - 2
HDFS Distributed Storage
Ironman – 2
Superman -3
Batman – 2
Antman -2
Spiderman - 2
HDFS Write
Iteration or process

Emergence of Spark

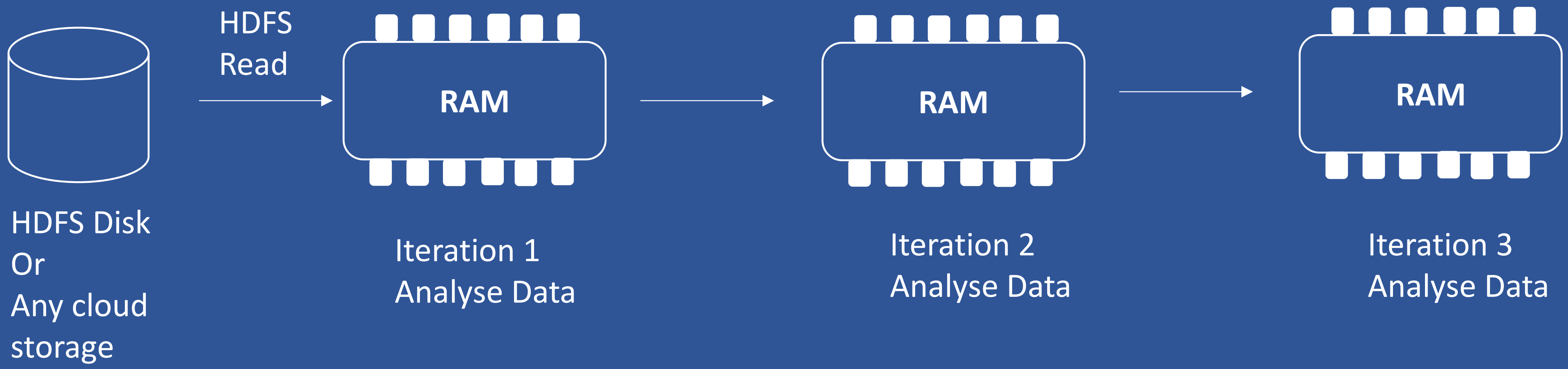
Author: Shanmukh Sattiraju

Drawbacks of MapReduce

Traditional Hadoop MapReduce processing



Emergence of spark



Apache Spark

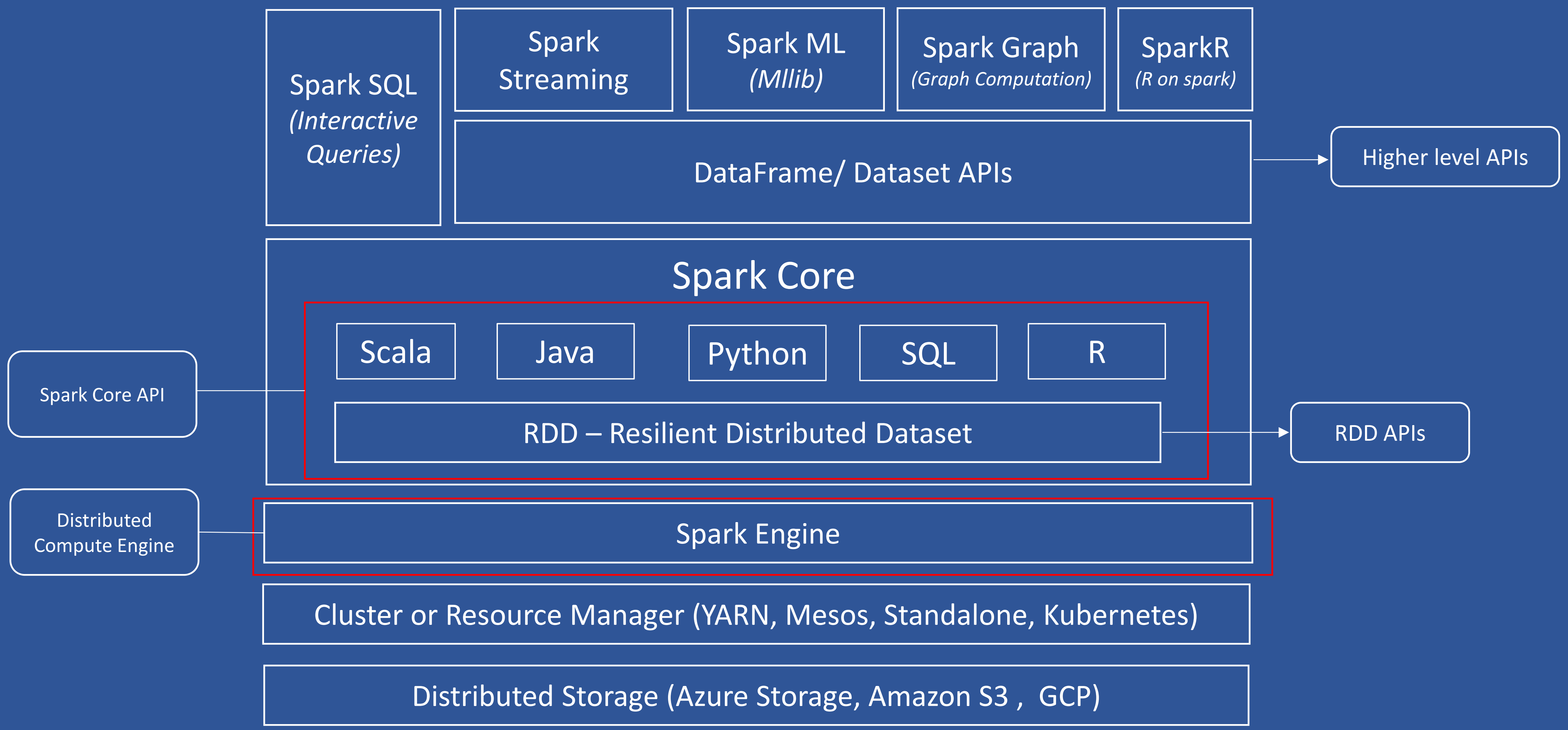
Apache Spark is an **open** source **in-memory** application framework for **distributed data processing** and iterative analysis on massive data volumes

In simple terms, Spark is a

- Compute Engine
- Unified data processing System

Spark Core Concepts

Apache Spark Ecosystem

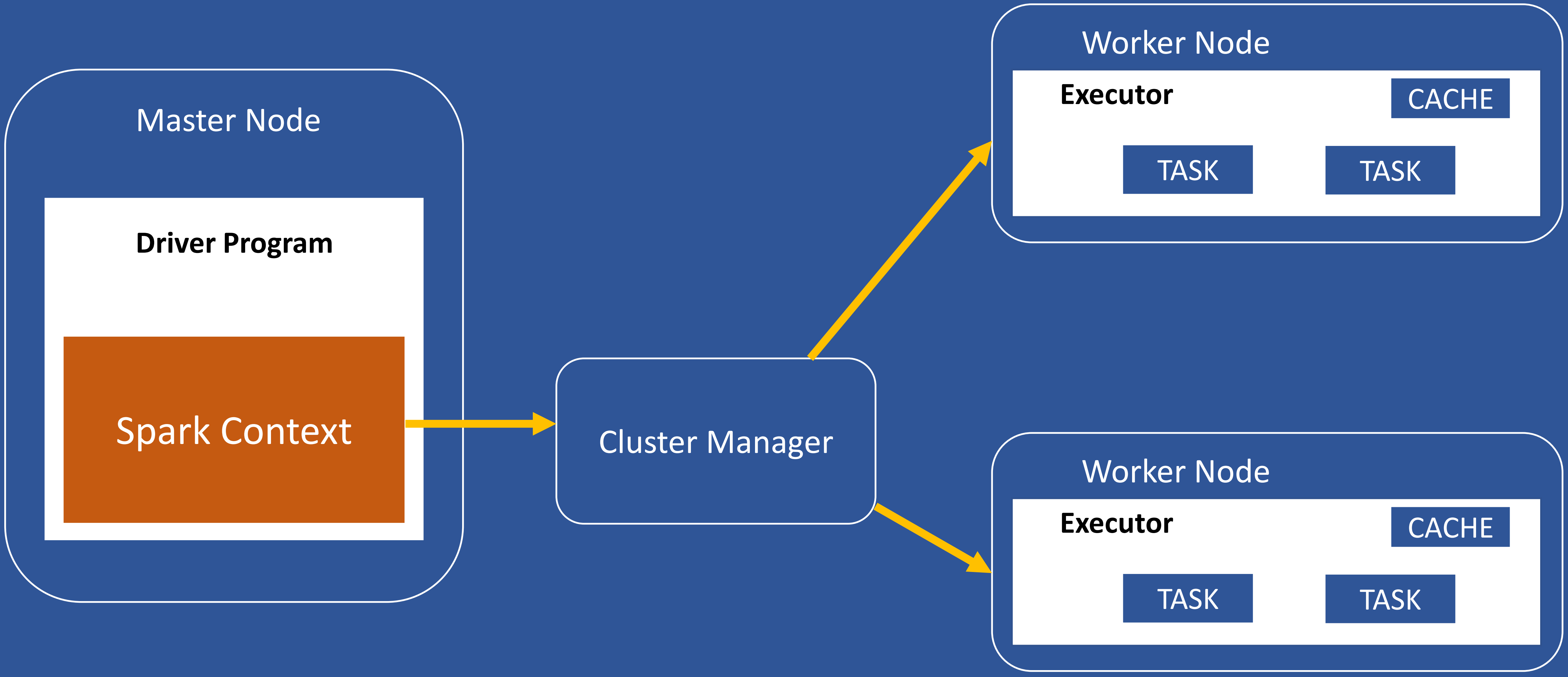


Author: Shanmukh Sattiraju

Limitations with Hadoop

Metrics	Hadoop	Apache Spark
Performance	Dependency on disks for read and write operations. Slower disk I/O	In-memory processing. 10-100x faster than Hadoop
Development	Need to develop Map and Reduce Code which is complex	Use native SQL to develop by making use of Spark SQL and composable APIs
Language	Java	Java, Scala, Python and R
Storage	HDFS	HDFS and Cloud Storages(Azure Storage, Amazon S3, etc.)
Resource Management	YARN	YARN, Measos, Standalone, Kubernetes
Data processing	Batch Processing	Batch Processing, Streaming, Machine Learning

Apache Spark Architecture



Author: Shanmukh Sattiraju

Benefits of Spark pool

- Speed and efficiency
- Ease of Creation
- Support for ADLS Gen2
- Scalability

RDD – Resilient Distributed Dataset

- RDD Stands for Resilient Distributed Dataset.
- It is fundamental data structure of Apache Spark
- It is immutable collection of objects
- RDD is divided into logical partitions, which may be computed on different nodes of the cluster

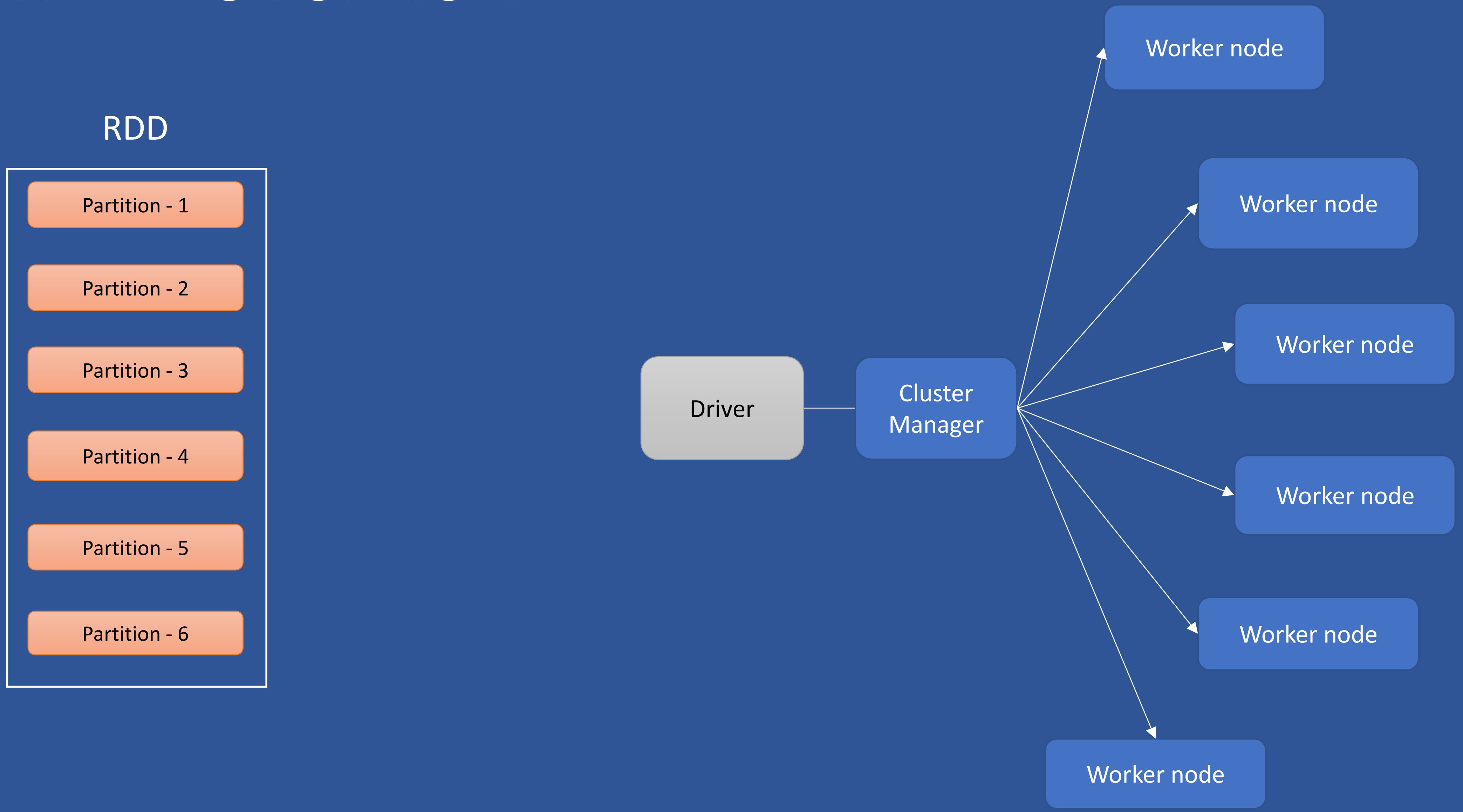
Let’s breakdown and see RDD

Resilient = It is fault tolerant

Distributed = data is spread across partitions in the clusters

Dataset = Set of data that have rows and columns

RDD -Overview



Author: Shanmukh Sattiraju

Lambda(), Map() , filter()

Lambda:
Anonymous functions (i.e. functions defined without a name)
Syntax:
lambda (< value> : <expression>)

Eg:

a = lambda (x : x+10)

print(a (10))

Result is 20

Map:
Return a new distributed dataset formed by passing each element of the source through a <function>

Syntax:
Map (<function>)

Eg:
If RDD have values [1,2,3]
rdd.map(lambda (x : x+10)

Result [11,12,13]

Map adds 10 on reach value in given data

Filter:
Return a new dataset formed by selecting those elements of the source on which <function> returns true

Syntax:
Filter (<function>)

Eg:
If RDD have values [11,12,13]
rdd.Filter(lambda x: x%2==0)

Result [12]

Filter will apply that on each element if that condition is true then it will take as new dataset

RDD Operations

Transformations

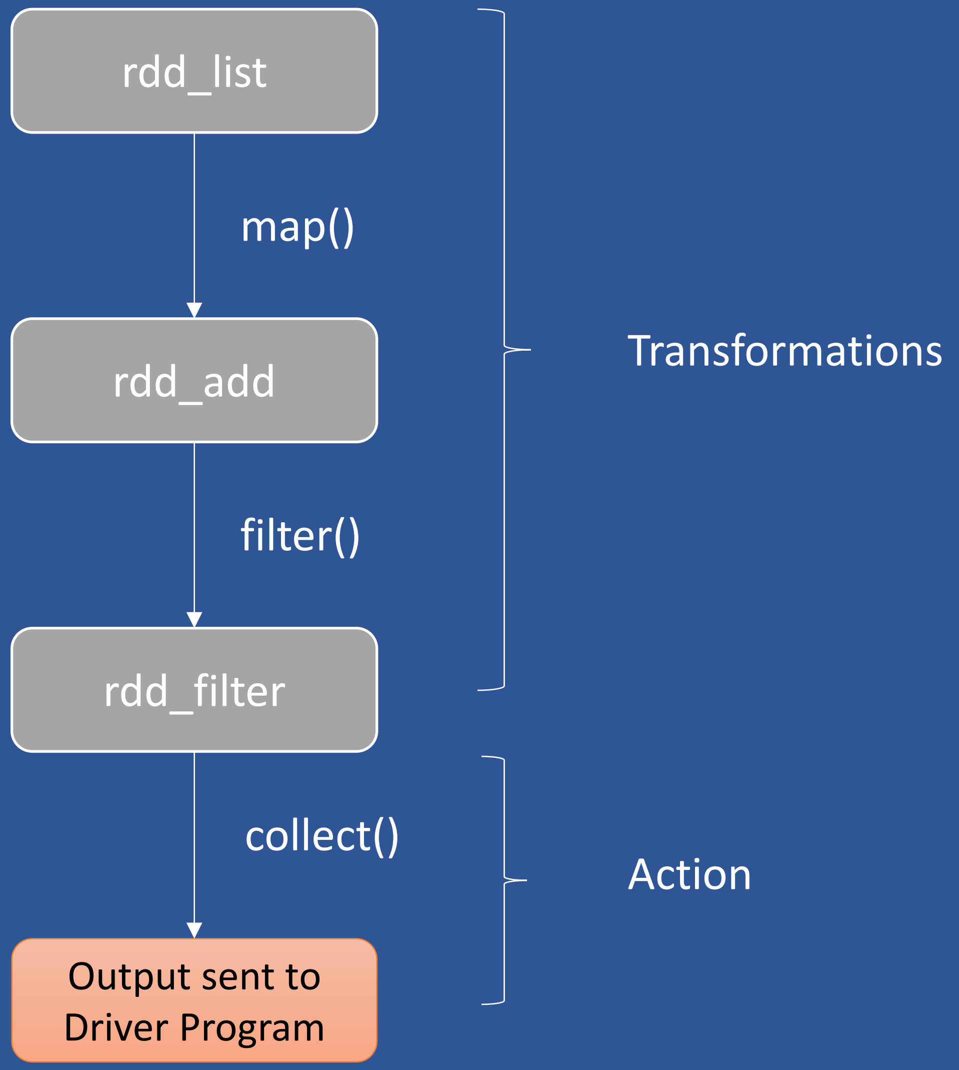
- Any operation that leads to some change in form of data is transformation.
- These take an RDD as the input and produces one or many RDDs as output
- After executing a transformation, the result RDD(s) can be smaller, bigger or sometimes same size as Parent RDDs
- Transformations are Lazy, they are not executed immediately. Transformations can execute only when actions are called (also called Lazy Evaluation)
- Transformations don't change the input RDD as RDDs are immutable
- E.g.: Filter(), Map(), FlatMap(), etc.

Actions

- Any operation that leads returns a value or data back to driver program is an Action
- It brings laziness of RDD into motion.

E.g.: count(), collect(), take()

RDD operations



Author: Shanmukh Sattiraju

Lineage

Dictionary

Definitions from Oxford Languages · Learn more

English

lineage

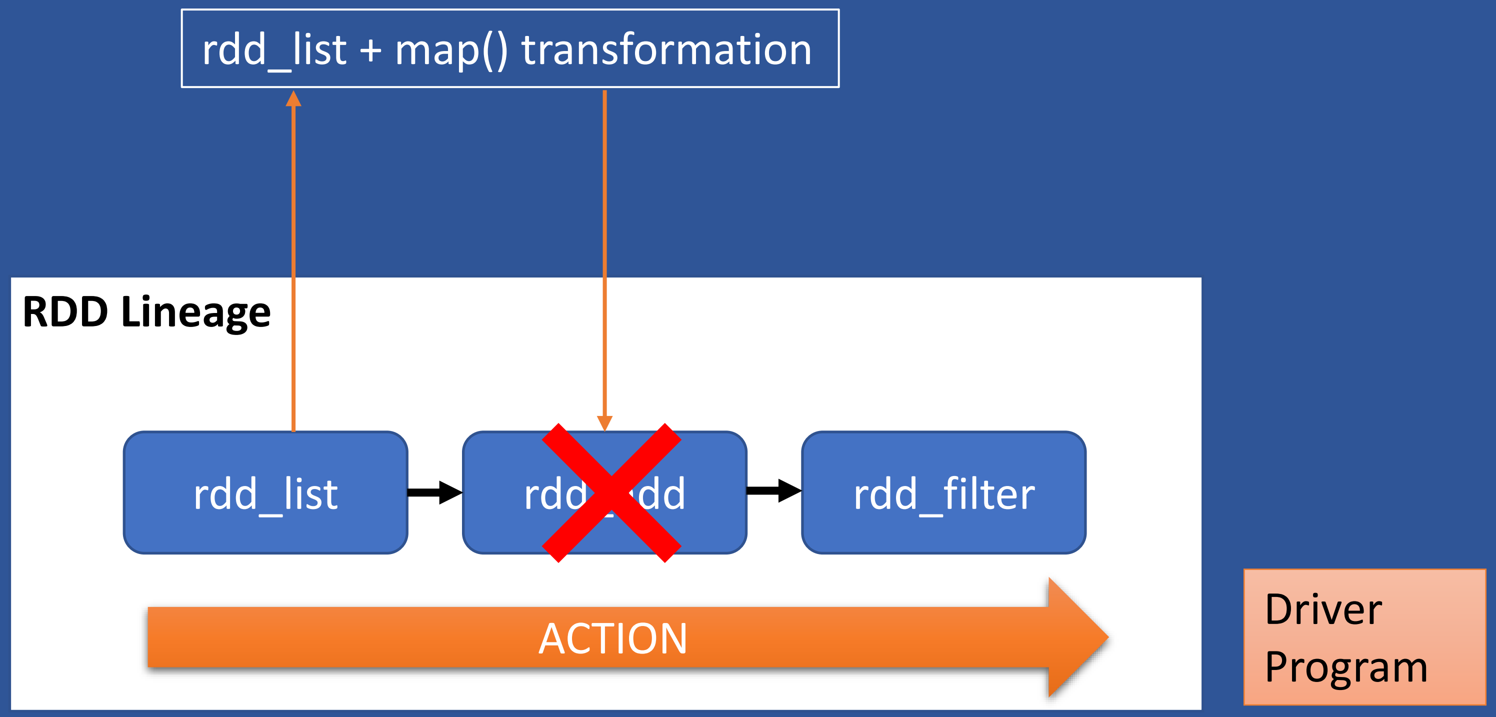
noun

1. direct descent from an ancestor; ancestry or pedigree.
   "a Dutch nobleman of ancient lineage"
   Similar: ancestry | family | parentage | birth | descent | line | extraction
2. BIOLOGY
   a sequence of species each of which is considered to have evolved from its predecessor.
   "the chimpanzee and gorilla lineages"

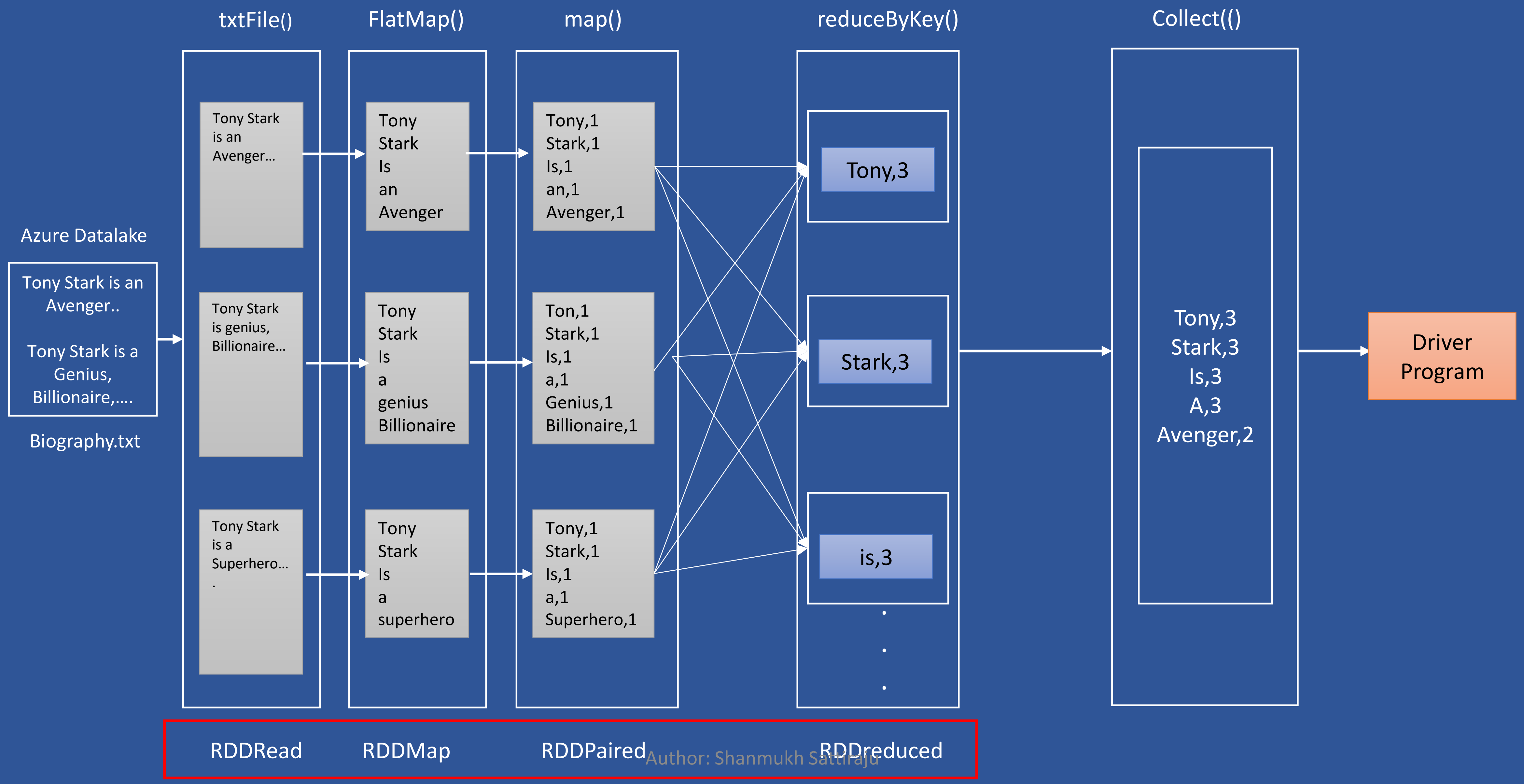
Feedback

RDD Lineage

```
>> rdd_list = sc.parallelize(list)

>> rdd_add = rdd_list.map(<func>)

>> rdd_filter = rdd_add.filter(<func>)

>> rdd.filter.collect()
```



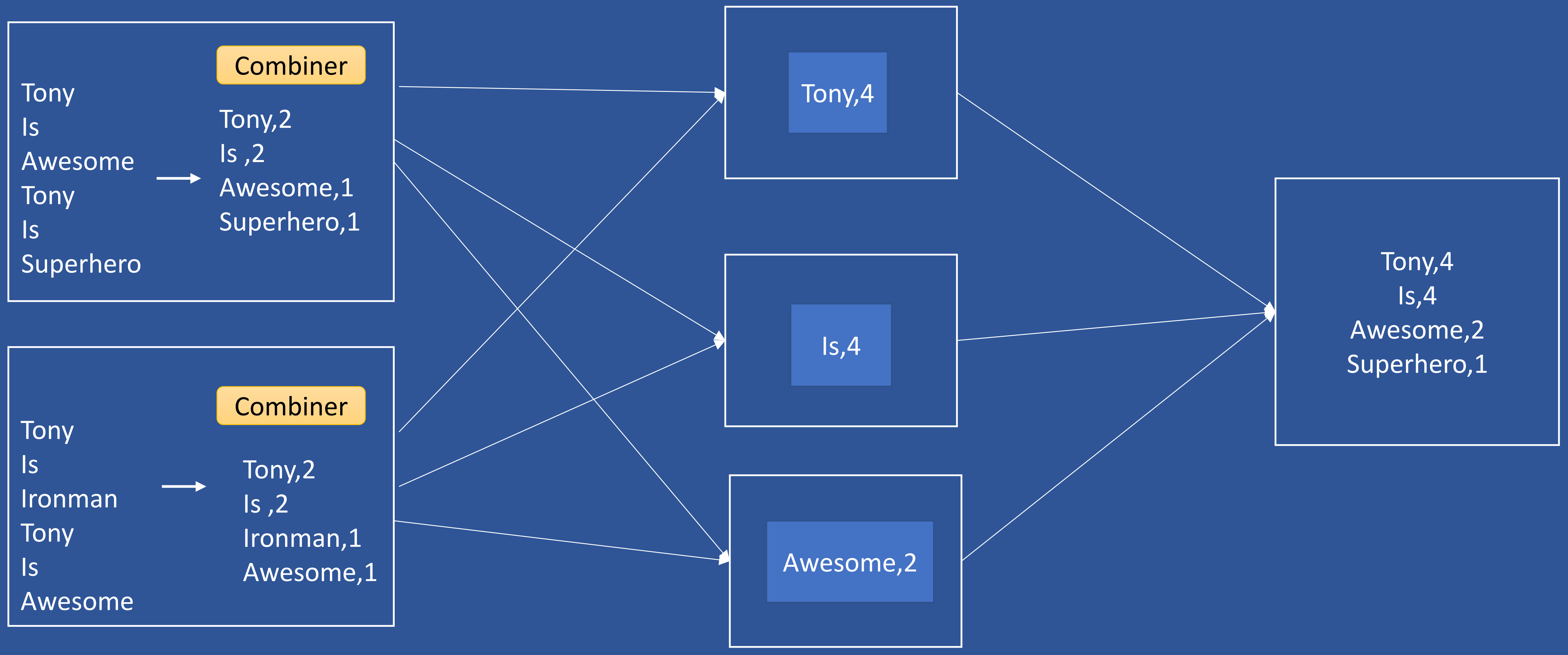
Word count



txtFile()
FlatMap()
map()
reduceByKey()
Collect(()
Azure Datalake
Tony Stark is an Avenger..
Tony Stark is a Genius, Billionaire,....
Biography.txt
Tony Stark is an Avenger...
Tony Stark is genius, Billionaire...
Tony Stark is a Superhero... .
Tony Stark Is an Avenger
Tony Stark Is a genius Billionaire
Tony Stark Is a superhero
Tony,1 Stark,1 Is,1 an,1 Avenger,1
Ton,1 Stark,1 Is,1 a,1 Genius,1 Billionaire,1
Tony,1 Stark,1 Is,1 a,1 Superhero,1
Tony,3
Stark,3
is,3
Tony,3 Stark,3 Is,3 A,3 Avenger,2
Driver Program
RDDRead
RDDMap
RDDPaired
RDDreduced
Author: Shanmukh Sattiraju

ReduceByKey() vs GroupByKey()

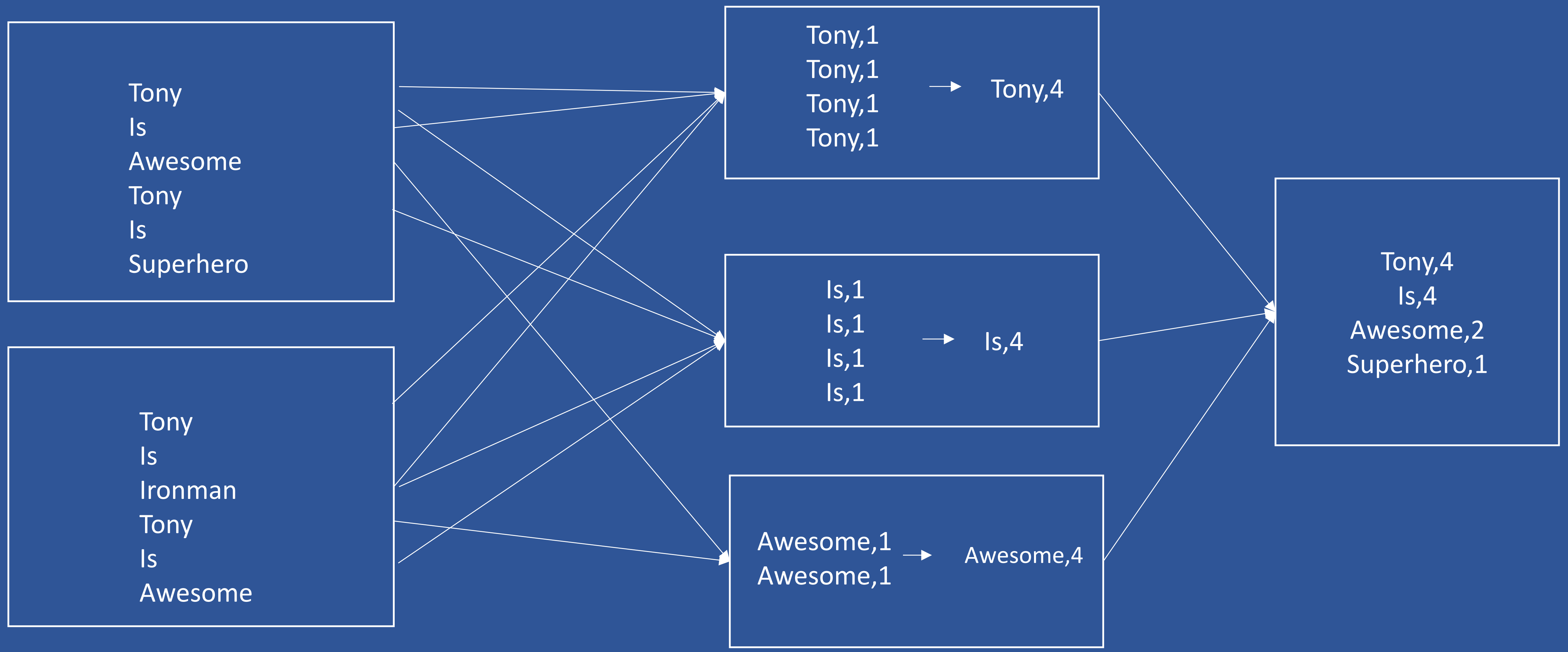
ReduceByKey()



Author: Shanmukh Sattiraju

ReduceByKey() vs GroupByKey()

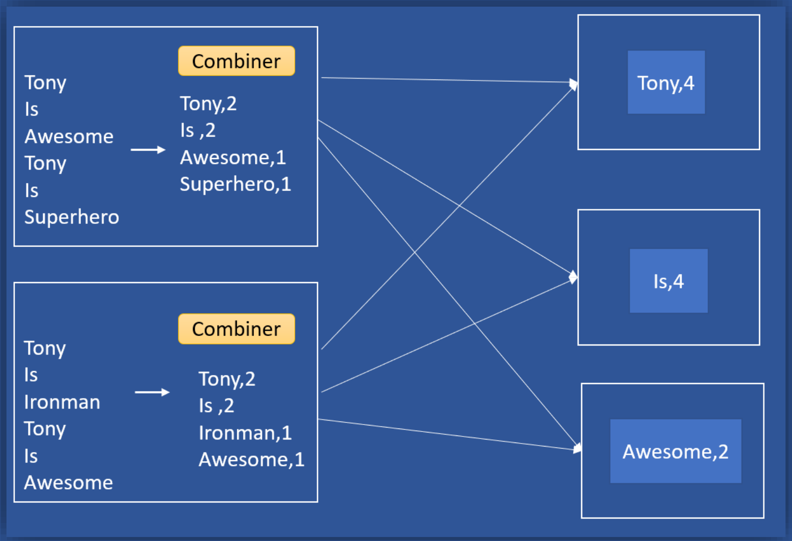
GroupByKey()



Author: Shanmukh Sattiraju

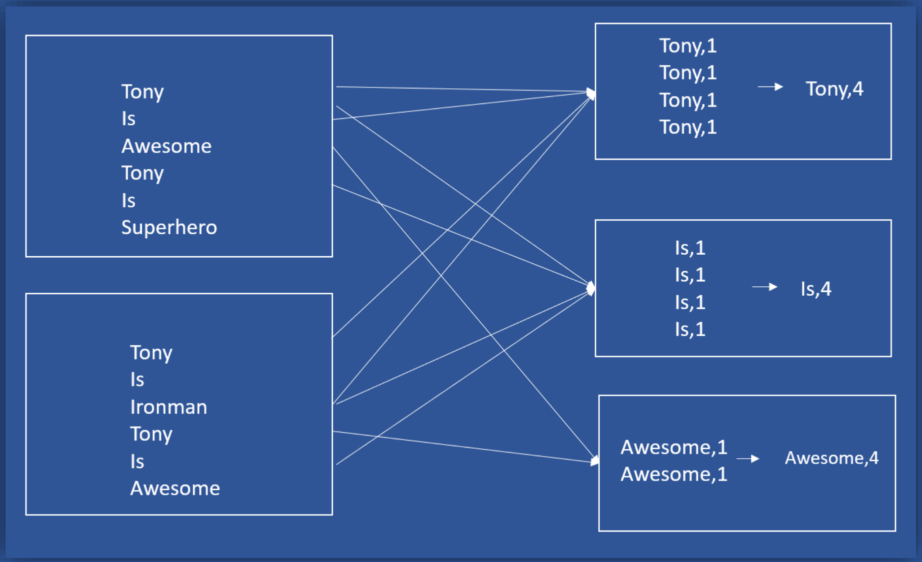
ReduceByKey()

- Wide Transformation
- Data is combined or aggregated at partition itself before getting shuffled
- Less shuffling – as data already combined
- More Efficient



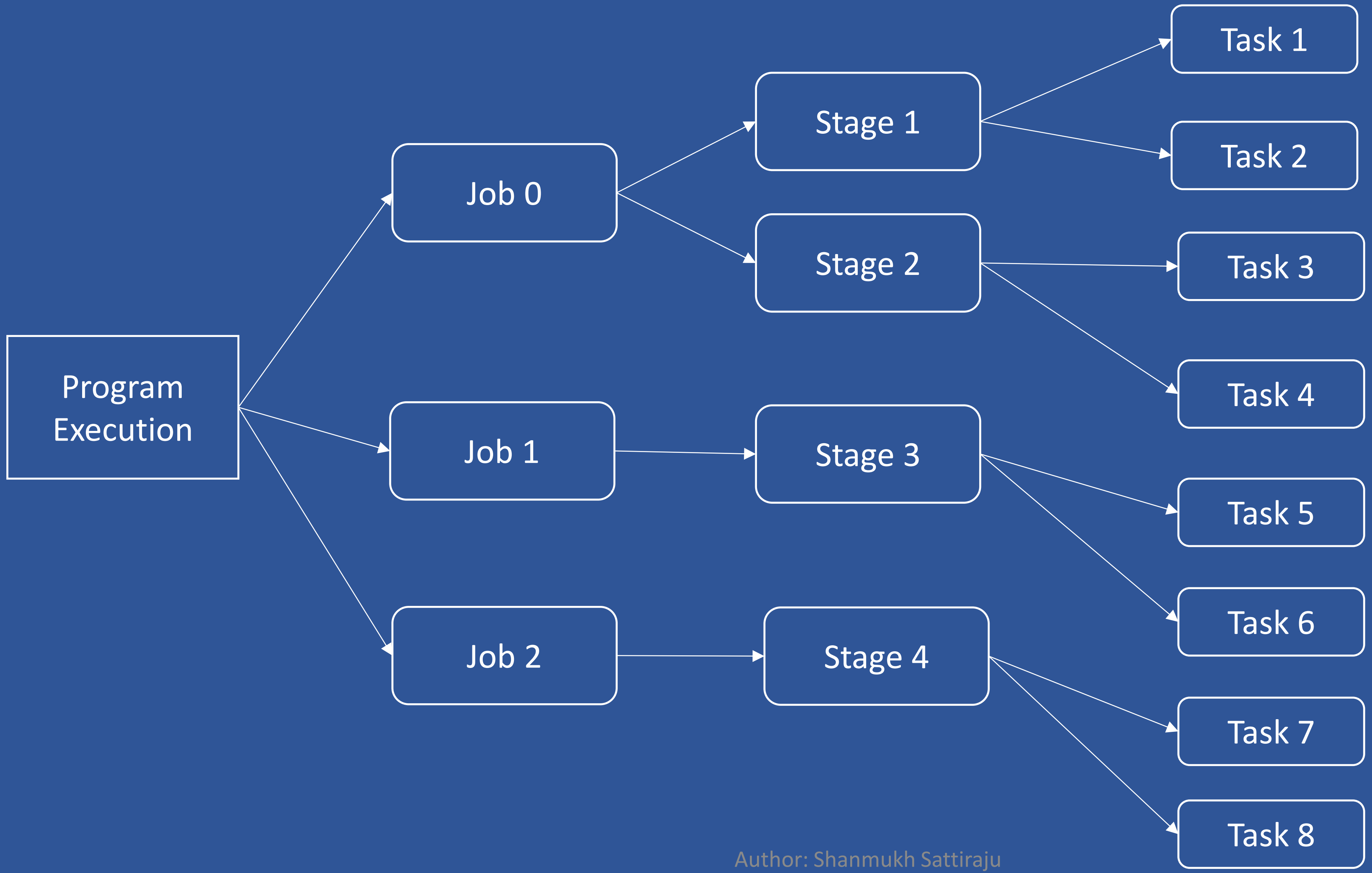
GroupByKey()

- Wide Transformation
- Data is combined or aggregated at after shuffling
- More shuffling – as need to collect data
- Less Efficient



Author: Shanmukh Sattiraju

Execution plan



Author: Shanmukh Sattiraju

Jobs

Number of Jobs = Number of Actions

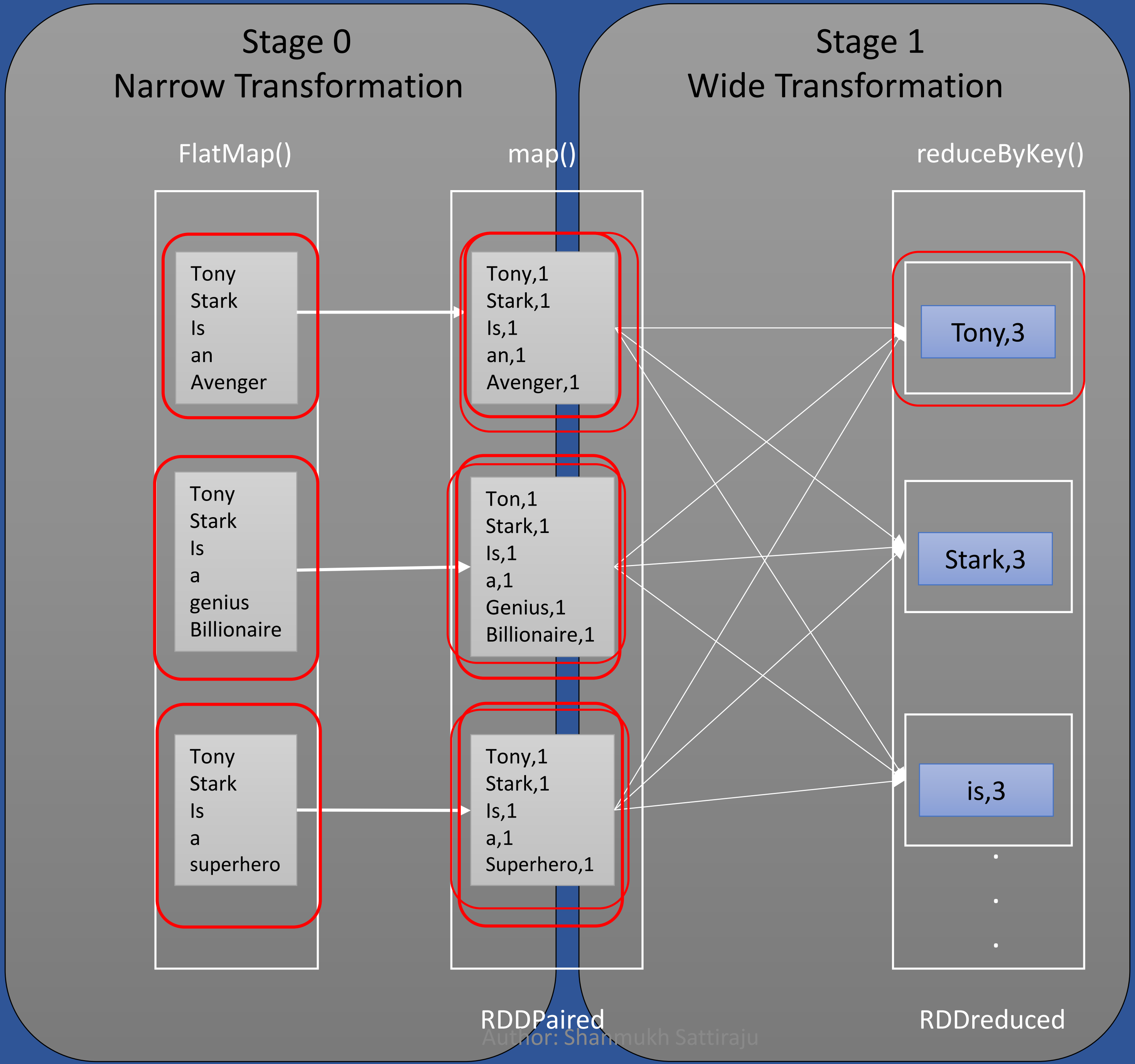
Transformations

Narrow Transformations

- In Narrow transformation, all the elements that are required to compute the records in single partition, live in the single partition of parent RDD
- E.g. map(), filter()

Wide Transformations

- In wide transformation, all the elements that are required to compute the records in the single partition, may live in many partitions of parent RDD.
- E.g. groupbyKey() and reducebyKey()


Stage 0
Narrow Transformation
Stage 1
Wide Transformation
FlatMap()
map()
reduceByKey()
Tony
Stark
Is
an
Avenger
Tony,1
Stark,1
Is,1
an,1
Avenger,1
Tony,3
Tony
Stark
Is
a
genius
Billionaire
Ton,1
Stark,1
Is,1
a,1
Genius,1
Billionaire,1
Stark,3
Tony
Stark
Is
a
superhero
Tony,1
Stark,1
Is,1
a,1
Superhero,1
is,3
RDDPaired
RDDreduced
Author: Shanmukh Sattiraju

Stages

Number of Stages = Number of Wide Transformations applied +1

Tasks

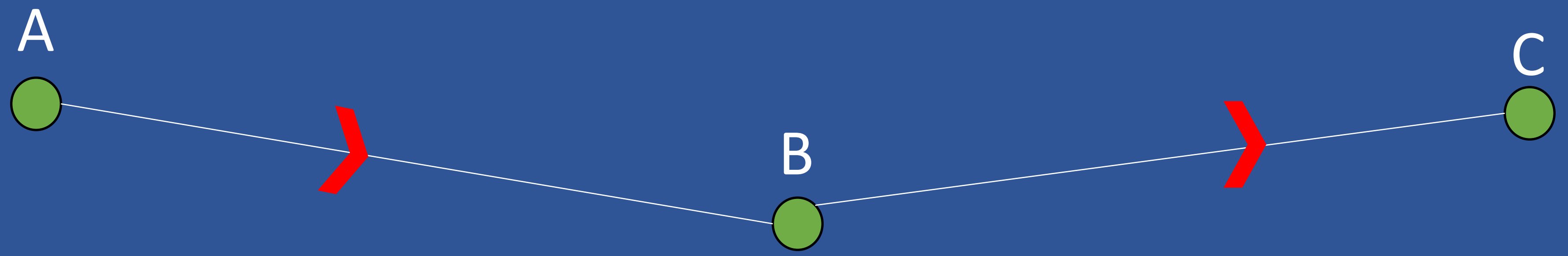
Number of Tasks = Number of Partitions

Summary

Number of Jobs	=	Number of Actions
Number of Stages	=	Number of Wide Transformations + 1
Number of Tasks	=	Number of Paritions

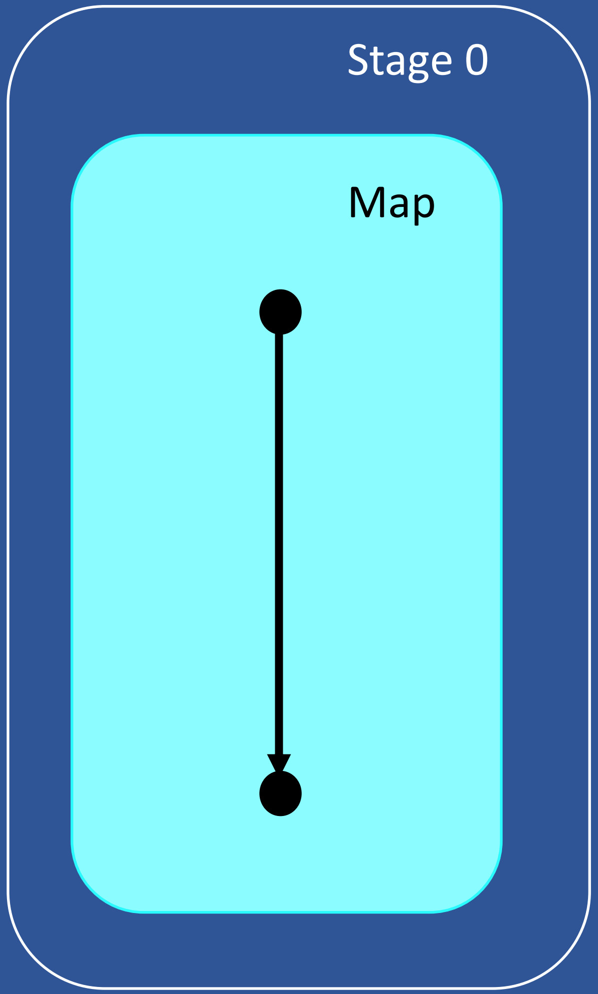
DAG

Directed Acyclic Graph



Author: Shanmukh Sattiraju

DAG



Author: Shanmukh Sattiraju

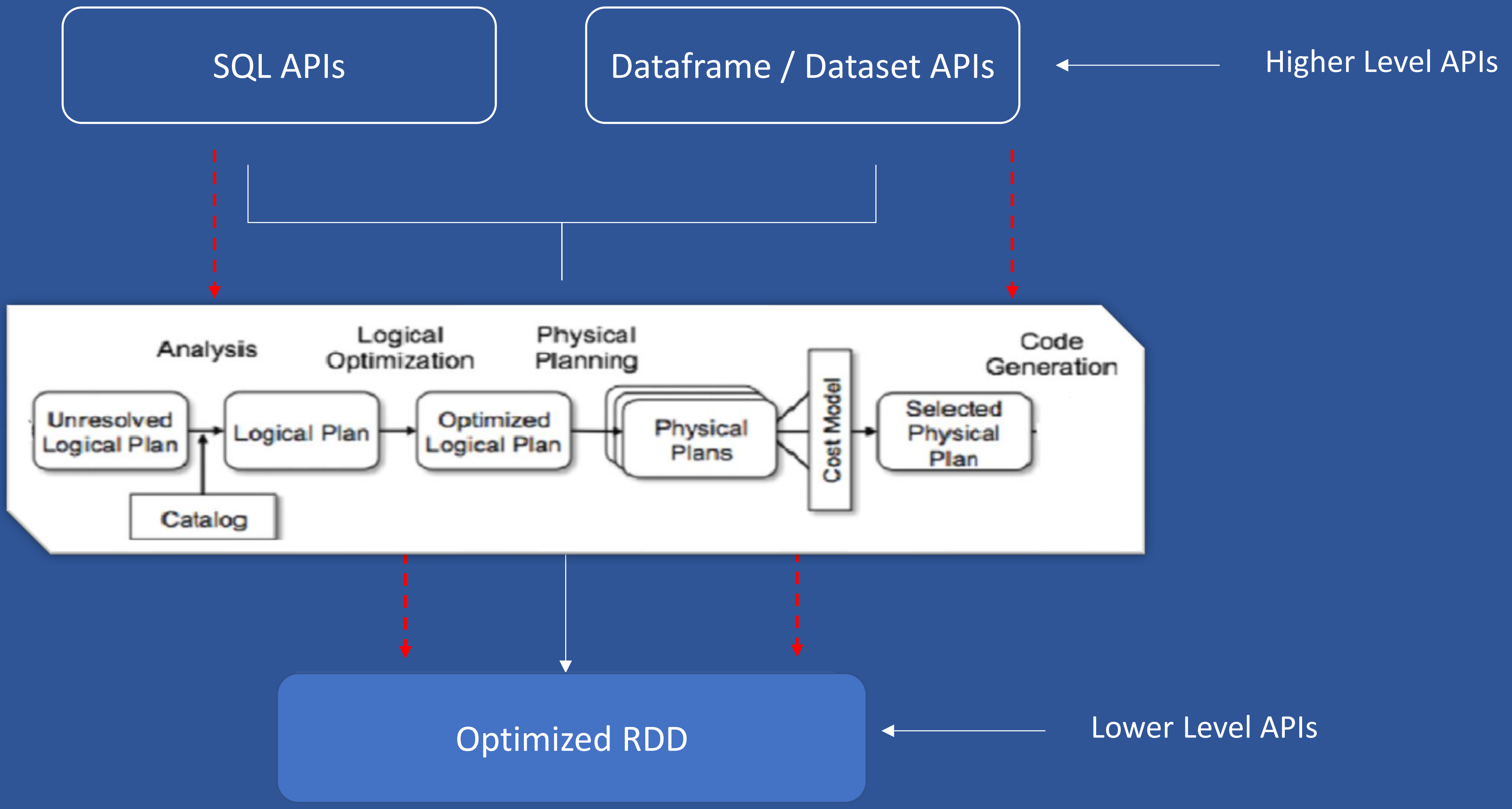
RDD lineage vs DAG

RDD Lineage

- Formed when a RDD/Data frame is created or after each transformation is applied
- Each RDD points to one or more parent RDD which forms a lineage
- It is Logical plan
- Its like a portion of DAG

DAG

- Forms when an action is called
- It's a physical plan built by DAG scheduler after an action is called
- It's like a combination of many RDD and its transformations



Author: Shanmukh Sattiraju

DataFrames

- Dataframes are built on top of Spark RDD APIs
- DataFrame APIs is more efficient, it can optimize operations using underlying catalyst

To read DataFrame :

DataframeReader

Supports:

JSON

CSV

PARQUET

AVRO

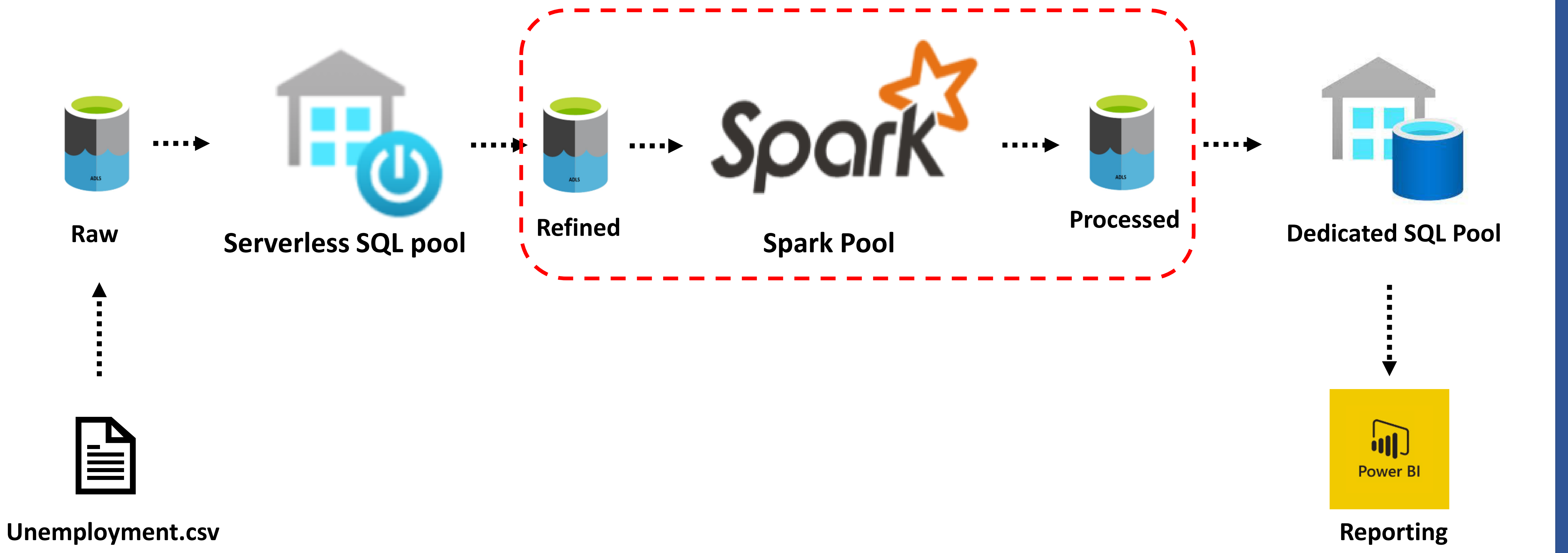
ORC

TEXT

PySpark Transformations

Author: Shanmukh Sattiraju

Project Architecture



Author: Shanmukh Sattiraju

Selection and filtering

We are making use of below functions and understand their usage

1. display()
2. Select()
3. selectExpr()
4. Filter()
5. Where()

Handling NULLs/Missing values and Grouping Aggregation

We are making use of below functions and understand their usage

1. fillna()
2. na.fill()
3. Groupby()
4. Agg()
5. dropna()
6. na.drop()

Handling NULLs and aggregation

Columns

Line Number
Year
Month
State
Labor Force
Employed
Unemployed
Unemployment Rate
Industry
Gender
Education Level
Date Inserted
Aggregation Level
Data Accuracy

Transformation

Identify NULLs
Filter()

Replace NULLs
fillna() or na.fill()

Drop NULLs
Dropna() or na.drop()

Drop Duplicate rows
dropDuplicates()

Aggregation
groupBy() / agg()

Data Transformation and Manipulation:

We are making use of below functions and understand their usage

1. withColumn()
2. Distinct()
3. Drop()
4. withColumnRenamed()

Data Transformation and Manipulation:

Before transformation

Line Number
Year
Month
State
Labor Force
Employed
Unemployed
Unemployment Rate
Industry
Gender
Education Level
Date Inserted
Aggregation Level
Data Accuracy

After transformation

Line_Number
Year
Month
State
Labor Force
Employed
Unemployed
Industry
Gender
Education Level
Date Inserted
Aggregation Level
Data Accuracy
UnEmployed Rate Percentage

Transformation

Add column
withColumn()

Update column
withColumn()

Update column based on Condition
withColumn(when..)

Delete column
Drop()

Rename column
withColumnRenamed()

Author: Shanmukh Sattiraju

MSSparkUtils

- Microsoft Spark Utilities (MSSparkUtils) is a builtin package to help you easily perform common tasks.
- In databricks it is dbutils
- You can use MSSparkUtils to work with file systems, to get environment variables, to chain notebooks together, and to work with secrets
- MSSparkUtils are available in PySpark (Python), Scala, .NET Spark (C#), and R (Preview) notebooks and Synapse pipelines.
- We can work on storages like we do in local file system using file system in MSSparkUtils

MSSpark Utilities

This module provides various utilities for users to interact with the rest of Synapse notebook.

- **fs**: Utility for filesystem operations in Synapse
- **notebook**: Utility for notebook operations (e.g, chaining Synapse notebooks together)
- **credentials**: Utility for obtaining credentials (tokens and keys) for Synapse resources
- **env**: Utility for obtaining environment metadata (e.g, userName, clusterId etc)

Env utilities

- getUserName(): returns user name
- getUserId(): returns unique user id
- getJobId(): returns job id
- getWorkspaceName(): returns workspace name
- getPoolName(): returns Spark pool name
- getClusterId(): returns cluster id

Filesystem Utilities

- **Cp** -> Copies a file or directory, possibly across File Systems
- **Mv** -> Moves a file or directory, possibly across File Systems
- **ls** -> Lists the contents of a directory
- **mkdirs** -> Creates the given directory if it does not exist, also creating any necessary parent directories
- **put** -> Writes the given String out to a file, encoded in UTF-8
- **head** -> Returns up to the first 'maxBytes' bytes of the given file as a String encoded in UTF-8
- **append** -> Append the content to a file
- **rm** -> Removes a file or directory
- **exists** -> Check if a file or directory exists
- **mount** -> Mounts the given remote storage directory at the given mount point
- **Unmount** -> Deletes a mount point
- **mounts** -> Show information about what is mounted
- **getMountPath** -> Gets the local path of the mount point

Mounting Storage

Notebook

```
Spark.read.format('csv')\
        .option(('header','true')\
        .load('synfs:/'+JobId + '/lake/transformed/nulls.csv'
```

Mount point

/lake

Azure Data Lake

Container : refined

Folder: transformed

File: nulls.csv

Mounting storage to Spark pool

Mounting = attaching

Once we attached our storage to a **mount point** , we can access the storage account without using full path name

Syntax to mount: (Using linked service authentication - Recommended)

```
mssparkutils.fs.mount(
    “< full_path>",
    “<Mount_point_name>",
    {"LinkedService":“<Linked_Service_name>"}
)
```

To increase quota

Workspace level

- Every Azure Synapse workspace comes with a default quota of vCores that can be used for Spark. The quota is split between the user quota and the dataflow quota so that neither usage pattern uses up all the vCores in the workspace. The quota is different depending on the type of your subscription but is symmetrical between user and dataflow. However if you request more vCores than are remaining in the workspace, then you'll get the following error:

> Failed to start session: [User] MAXIMUM_WORKSPACE_CAPACITY_EXCEEDED
> Your Spark job requested 12 vCores.
> However, the workspace only has xxx vCores available out of quota of yyy vCores.
> Try reducing the numbers of vCores requested or increasing your vCore quota. Click here for more information - https://go.microsoft.com/fwlink/?linkid=213499

- The following article describes how to request an increase in workspace vCore quota.
- Select "Azure Synapse Analytics" as the service type.
- In the Quota details window, select Apache Spark (vCore) per workspace

Notebook utilities

- **exit** -> This method lets you exit a notebook with a value.
- **run** -> This method runs a notebook and returns its exit value.

- Similar to run() method as above we also have a magic command %%run <notebook>

- Using %%run followed by notebook name will also runs the notebook from another book

Magic commands

- You can use multiple languages in one notebook
- You need to specify language magic command at the beginning of a cell.
- By default, the entire notebook will work on the language that you choose at the top

Magic command	Language	Description
%%pyspark	Python	Execute a Python query against Spark Context.
%%spark	Scala	Execute a Scala query against Spark Context.
%%sql	Spark SQL	Execute a SparkSQL query against Spark Context.
%%csharp	.NET for Spark C#	Execute a .NET for Spark C# query against Spark Context.
%%sparkr	R	Execute a R query against Spark Context.

Access mount point from other notebook

- The purpose of mount of is to reduce our development effort
- Ideally once we mount the mount point in synapse, they should be accessing for other notebooks

Notebook 1

```
mssparkutils.fs.mount(
    "abfss://raw@datalake11212.dfs.co
re.windows.net/",
 "/lake",
{"LinkedService":"synapse1121-
WorkspaceDefaultStorage"}
```

Notebook 2

```
mssparkutils.fs.mount()
```

Returns an empty array

SQL Temp Views

- You cannot reference data or variables directly across different languages in a Synapse notebook.
- In Spark, a temporary views can be referenced across languages.
- The lifetime of this temporary table is tied to the SparkSession
- SQL script only works on either a table or a view not on data frames

```
%%sql

SELECT * FROM df
```

```
df = spark.read.format(‘csv’)\
                    .option(‘header’,’true’)\
                    .load(<path>)
```

```
df.createTempView(‘<ViewName’>)
```

```
%%sql

SELECT * FROM ViewName
```

Temporary Views

All the temporary views are active for Session Only

- createTempView()
  - This will throw error if another view is created with same name in that session
- createOrReplaceTempView()
  - This is used when you want to automate running you notebook and use the same name again

- We have 2 more views
  - createGlobalTempView()
  - createOrReplaceGlobalTempView()

These Global views makes the view available for another notebook to access them if they are attached to same cluster .

But synapse analytics is not like databricks , we will not have any cluster. Hence the Global Temp views are not like we used in databricks

Workspace data

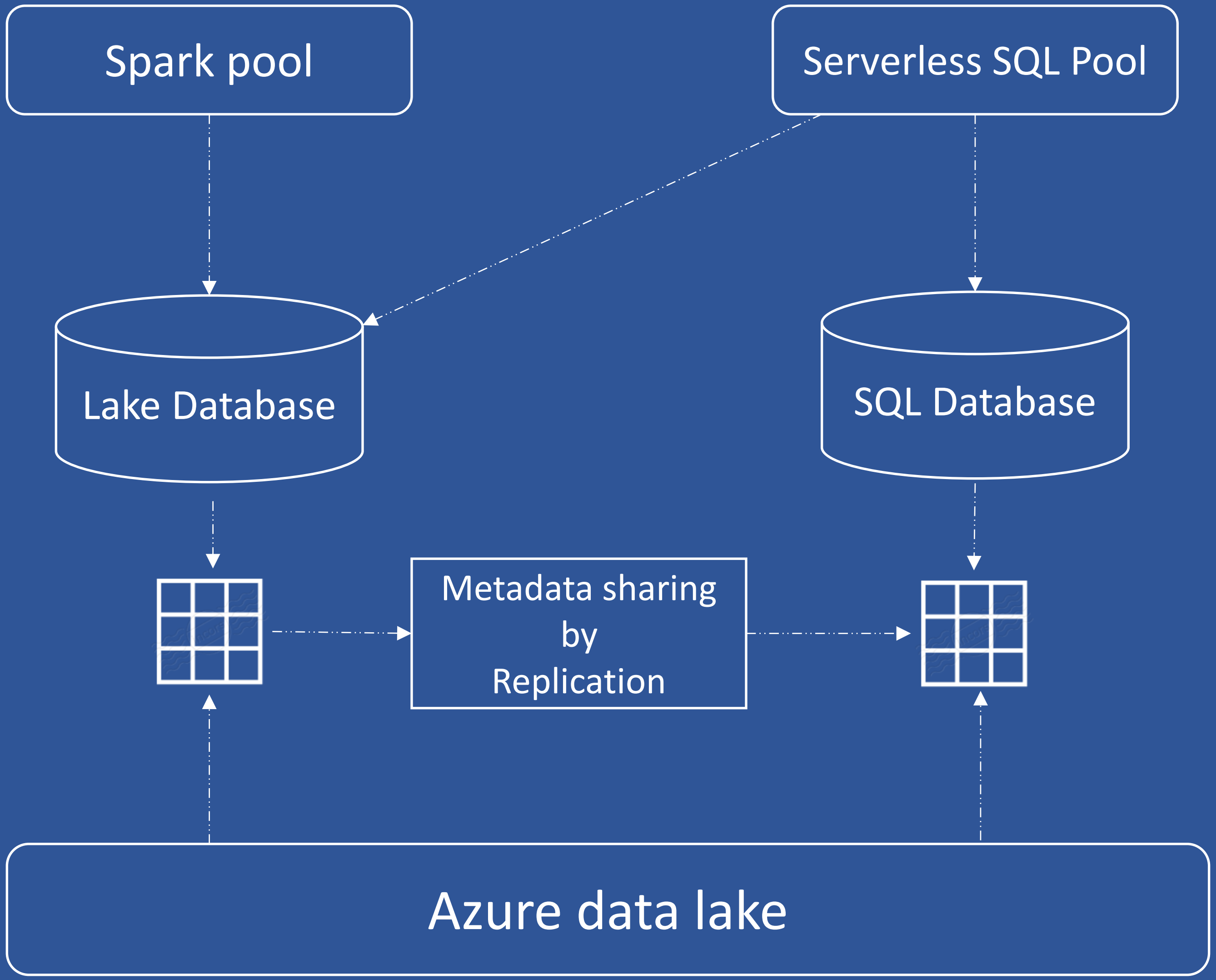
- **Lake databases**
  - You can define tables on top of lake data using Apache Spark notebooks
  - You can refer this tables for querying using T-SQL (Transact-SQL) language using the serverless SQL pool
- **SQL Databases**
  - You can define your own databases and tables directly using the serverless SQL pools
  - You can use T-SQL CREATE DATABASE, CREATE EXTERNAL TABLE to define the objects

Spark Managed vs External Tables

- Managed Tables
  - These can be defined without a specified location
  - The data files are stored within the storage used by the metastore
  - Dropping the table not only removes its metadata from the catalog, but also deletes the folder in which its data files are stored.

- External Tables
  - These can be defined for a custom file location, where the data for the table is stored.
  - The metadata for the table is defined in the Spark catalog.
  - Dropping the table deletes the metadata from the catalog, but doesn't affect the data files.

Metadata sharing



Author: Shanmukh Sattiraju

Joins and combining data

We are making use of below functions and understand their usage

1. join()
   I. Inner join
   II. Left join
   III. Right join
   IV. Outer Join
   V. Left Semi Join
   VI. Left Anti Join
   VII. Cross Join
2. Union()

Join Transformations

Before transformation

Line_Number
Year
Month
State
Labor Force
Employed
Unemployed
Industry
Gender
Education Level
Date Inserted
Aggregation Level
Data Accuracy
UnEmployed Rate Percentage

Education Level
Expected Salary Range in USD

After transformation

Line_Number
Year
Month
State
Labor Force
Employed
Unemployed
Unemployment Rate
Industry
Gender
Education Level
Date Inserted
Aggregation Level
Data Accuracy
UnEmployed Rate Percentage
Expected Salary Range in USD

Transformation

Joining data
.join()

Author: Shanmukh Sattiraju

String Manipulation and sorting

We are making use of below functions and understand their usage

1. replace()
2. Split()
3. Concat()
4. OrderBy()
5. Sort()

String Manipulation and sorting

Before transformation

Line_Number
Year
Month
State
Labor Force
Employed
Unemployed
Industry
Gender
Education Level
Date Inserted
Aggregation Level
Data Accuracy
UnEmployed Rate Percentage
Expected Salary Range in USD

After transformation

Line_Number
Year
Month
State
Labor_Force
Employed
Unemployed
Industry
Gender
Education_Level
Date_Inserted
Aggregation_Level
Data_Accuracy
UnEmployed_Rate_Percentage
Min_Salary_USD
Max_Salary_USD

Transformation

Add underscores in columns
.replace()

Created 2 columns from
Expected salary range
.split()

Combine month Year Columns
.concatenate()

Orderby() / Sort()

Window functions

We are making use of below functions and understand their usage

1. row_number()
2. rank()
3. dense_rank()

String Manipulation and sorting

Before transformation

Line_Number
Year
Month
State
Labor_Force
Employed
Unemployed
Industry
Gender
Education_Level
Date_Inserted
Aggregation_Level
Data_Accuracy
UnEmployed_Rate_Percentage
Min_Salary_USD
Max_Salary_USD

After transformation

Line_Number
Year
Month
State
Labor_Force
Employed
Unemployed
Industry
Gender
Education_Level
Date_Inserted
Aggregation_Level
Data_Accuracy
UnEmployed_Rate_Percentage
Min_Salary_USD
Max_Salary_USD
dense_rank

Transformation

Assigning ranks based on Unemployment rate
.dense_rank()

We understood how the below will work
.row_number()
.rank()

Pivoting and conversions

We are making use of below functions and understand their usage

1. cast()
2. pivot()
3. Stack()
4. to_date()

Schema definition and Management

StructType and StructField

- StructType & StructField classes are used to programmatically specify the schema to the DataFrame
- **StructType**:
  - Represents the schema or structure of a DataFrame.
  - It is a collection of StructField objects.
  - Defines the columns and their data types in a DataFrame.
  - Created by passing a list of StructField objects.
- **StructField:**
  - Represents a single field or column in a DataFrame schema.
  - Defines the name, data type, and other attributes of a column.
  - Used as elements within a StructType object.
  - Syntax: StructField(name, datatype, nullable=True)
    - name: Name or identifier of the column.
    - dataType: Data type of the column.
    - nullable: Specifies whether the column can contain null values.

UDFs

- In PySpark, UDF stands for User-Defined Function.
- UDFs allow you to define custom functions to operate on Spark DataFrames or RDDs (Resilient Distributed Datasets).
- These functions can be used to perform complex computations
- These can be used when transformations on the data that are not available through built-in Spark functions.

Why UDFs?

- In SQL or PySpark, you cannot directly use python functions on them.
- To use the custom functions on dataframes or Spark SQL you need UDFs

Methods to create UDF

Method 1:

1. Create a function in Python syntax
2. Register that function as udf() to use it on dataframe or Spark SQL

Method 2:

Create a function in a Python syntax and wrap it with UDF Annotation

Steps to create UDF – Method 1

1. Defining python function using def

   Syntax:

   ```
   def <Function_name>(<args>):
           <Function_definition>
           return <return_Type>
   ```

2. Registering the function as UDF

   Syntax (to use on Dataframe):

   ```
   from pyspark.sql.functions import udf
   my_udf = udf(Function_name, returnType)
   ```

   Syntax (to use on Spark SQL):

   ```
   spark.udf.register(“<UDF_name”>, <Function_Name>)
   ```

Steps to create UDF – Method 2

1. Use annotation to wrap the UDF around the function for applying on DF

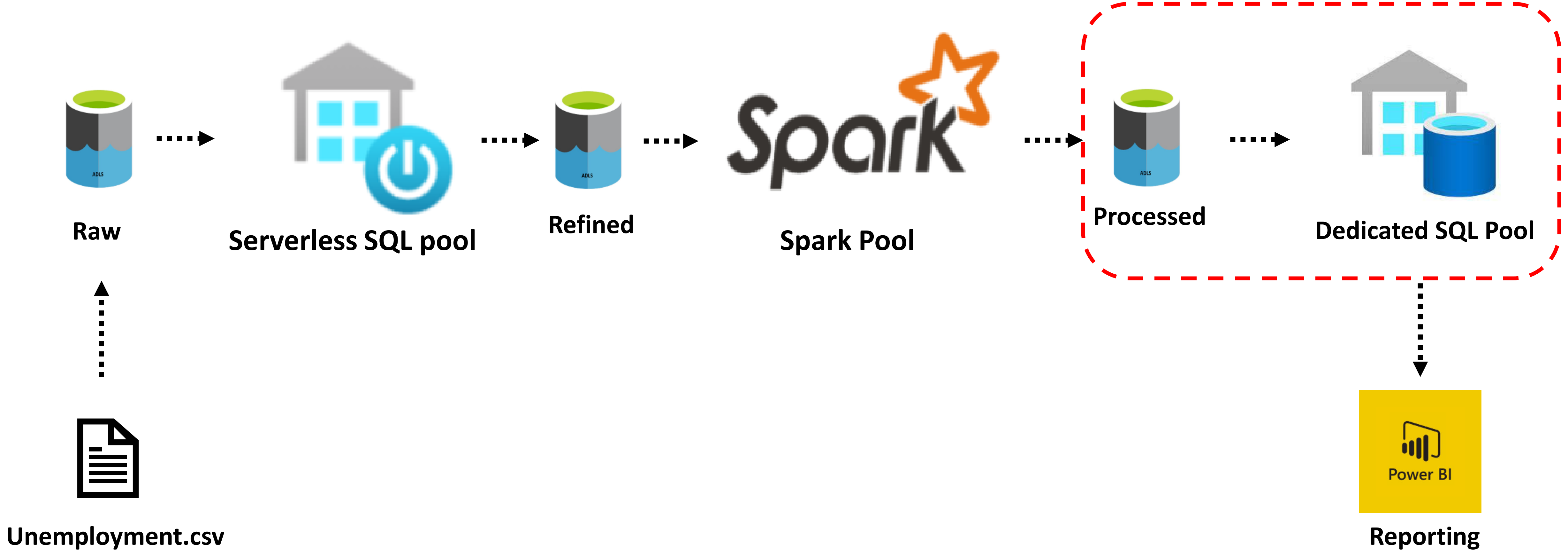
Syntax:

```
@udf(return_Type)
def <Function_name>(<args>):
        <Function_definition>
        return <return_Type>
```

Dedicated SQL Pool

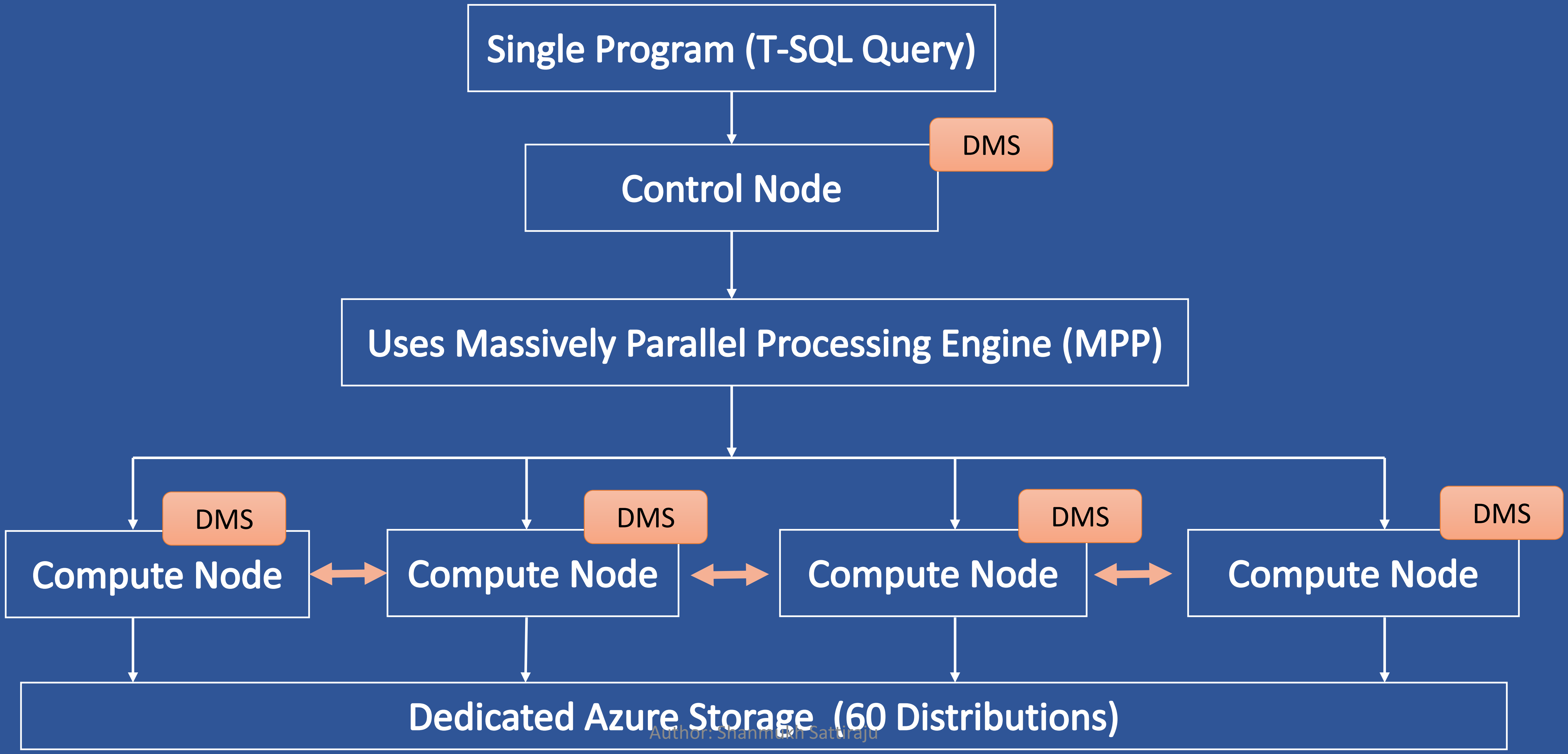
- Previously known as SQL Data Warehouse.
- This stores data in a relational table with Columnar storage
- Dedicated SQL pool is just a traditional Data Warehouse with MPP architecture
- You will have an internal storage specific to Dedicated SQL Pool
- The size of the Dedicated SQL pool depends on DWU (Data Warehousing Units) that we choose while creating it.

Project Architecture

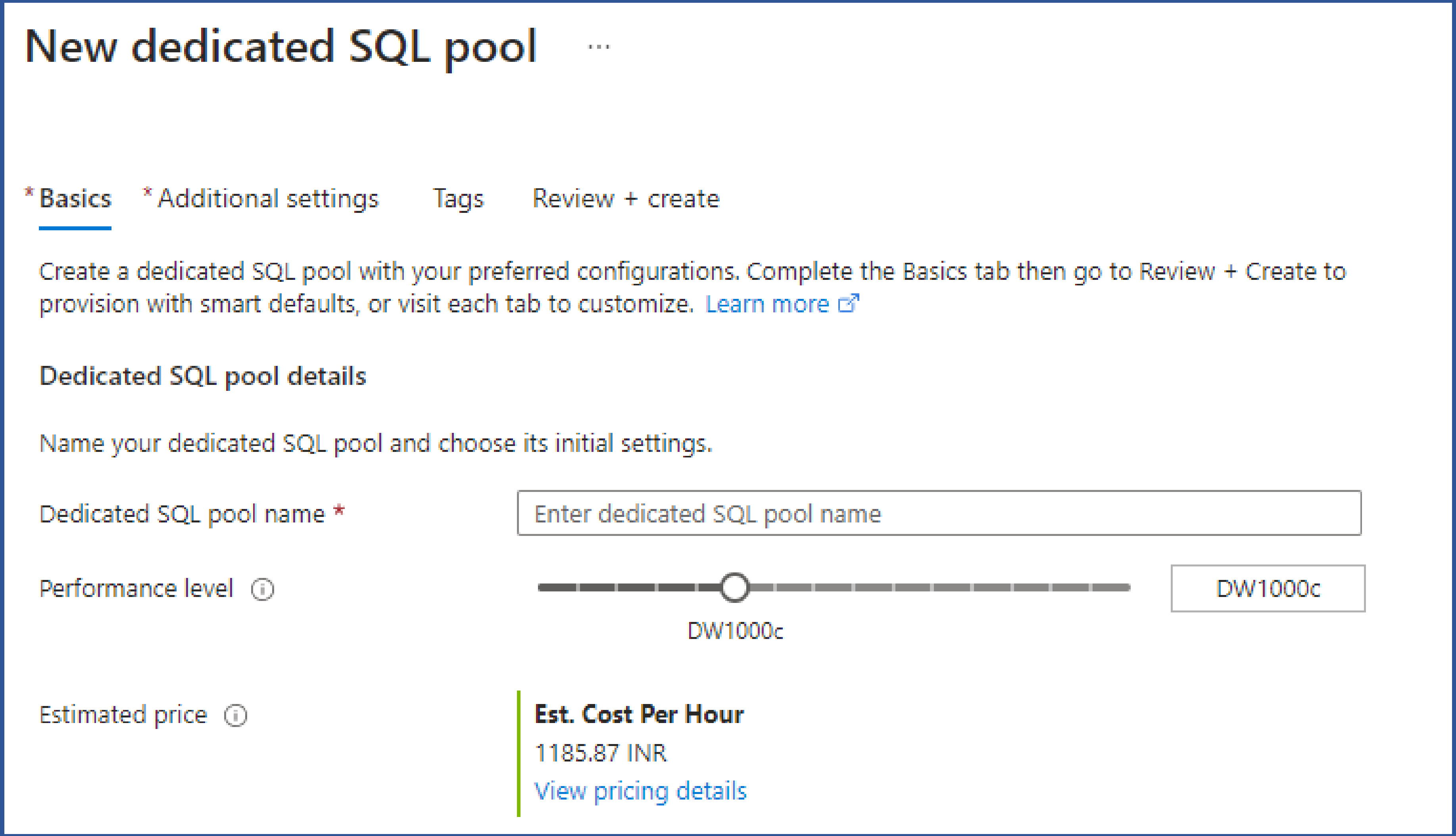


Author: Shanmukh Sattiraju

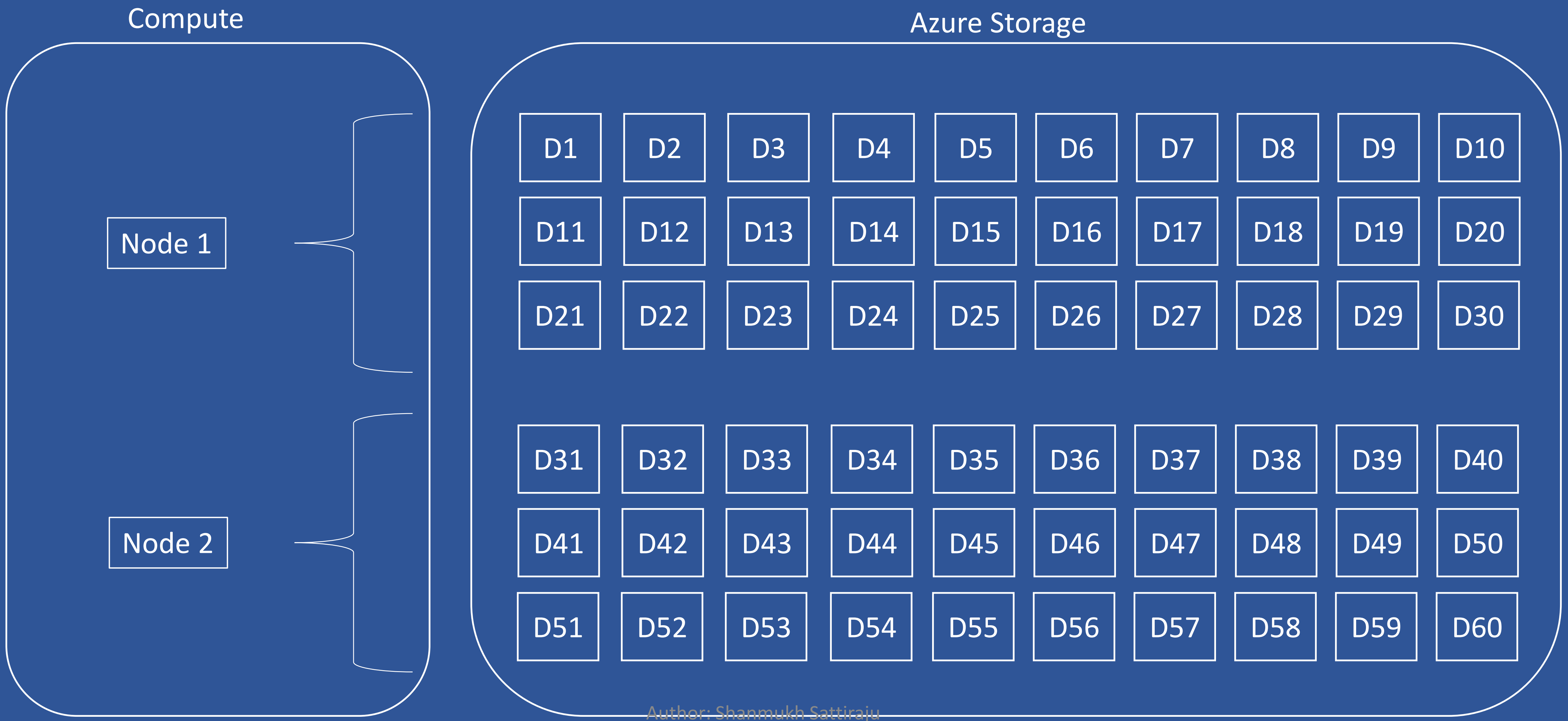
Synapse Dedicated SQL Architecture – MPP



Performance Level



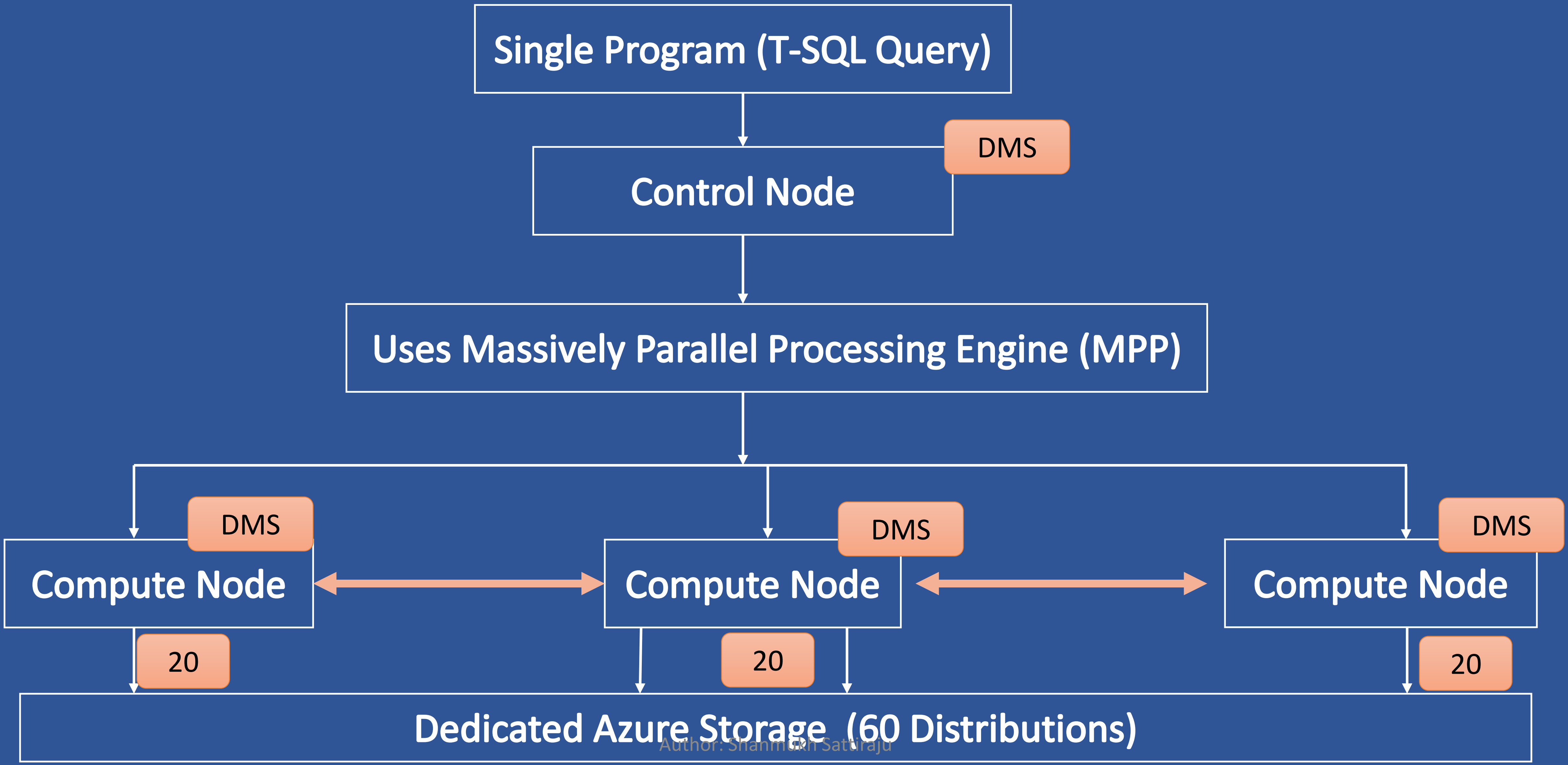
For DW1000c


Compute
Azure Storage
Node 1
Node 2
D1 D2 D3 D4 D5 D6 D7 D8 D9 D10
D11 D12 D13 D14 D15 D16 D17 D18 D19 D20
D21 D22 D23 D24 D25 D26 D27 D28 D29 D30
D31 D32 D33 D34 D35 D36 D37 D38 D39 D40
D41 D42 D43 D44 D45 D46 D47 D48 D49 D50
D51 D52 D53 D54 D55 D56 D57 D58 D59 D60

Scaling Compute with DWU

Data warehouse units	# of compute nodes	# of distributions per node
DW100c	1	60
DW200c	1	60
DW300c	1	60
DW400c	1	60
DW500c	1	60
DW1000c	2	30
DW1500c	3	20
DW2000c	4	15
DW2500c	5	12
DW3000c	6	10
DW5000c	10	6
DW6000c	12	5
DW7500c	15	4
DW10000c	20	3
DW15000c	30	2
DW30000c	60	1

DW1500c


Single Program (T-SQL Query)
DMS
Control Node
Uses Massively Parallel Processing Engine (MPP)
DMS
Compute Node
DMS
Compute Node
DMS
Compute Node
20
20
20
Dedicated Azure Storage (60 Distributions)

When to consider Dedicated SQL Pool?

- When data size more than a 1 TB
- When we have more than a billion Rows
- When we need high concurrency
- When you want predicted workloads

Copying data into Dedicated SQL pool

- You can copy data to Dedicated SQL pool in multiple ways.
- For now lets see the below ways
  - Using Copy command
  - Using BULK Load feature
  - Using pipeline to copy data

Using copy command

1. CREATE A TABLE

```
CREATE TABLE [schema].[TableName]
(
    <Column_Name> <DataType>,
    <Column_Name> <DataType>,
    <Column_Name> <DataType>,
)
WITH
(
   DISTRIBUTION = ROUND_ROBIN,
   CLUSTERED COLUMNSTORE INDEX
);
```

2. COPY data to table

```
COPY INTO [schema].[TableName]
FROM '<HTTPS://ExternalFilePath>'

WITH
(
FILE_TYPE='parquet'
)
```

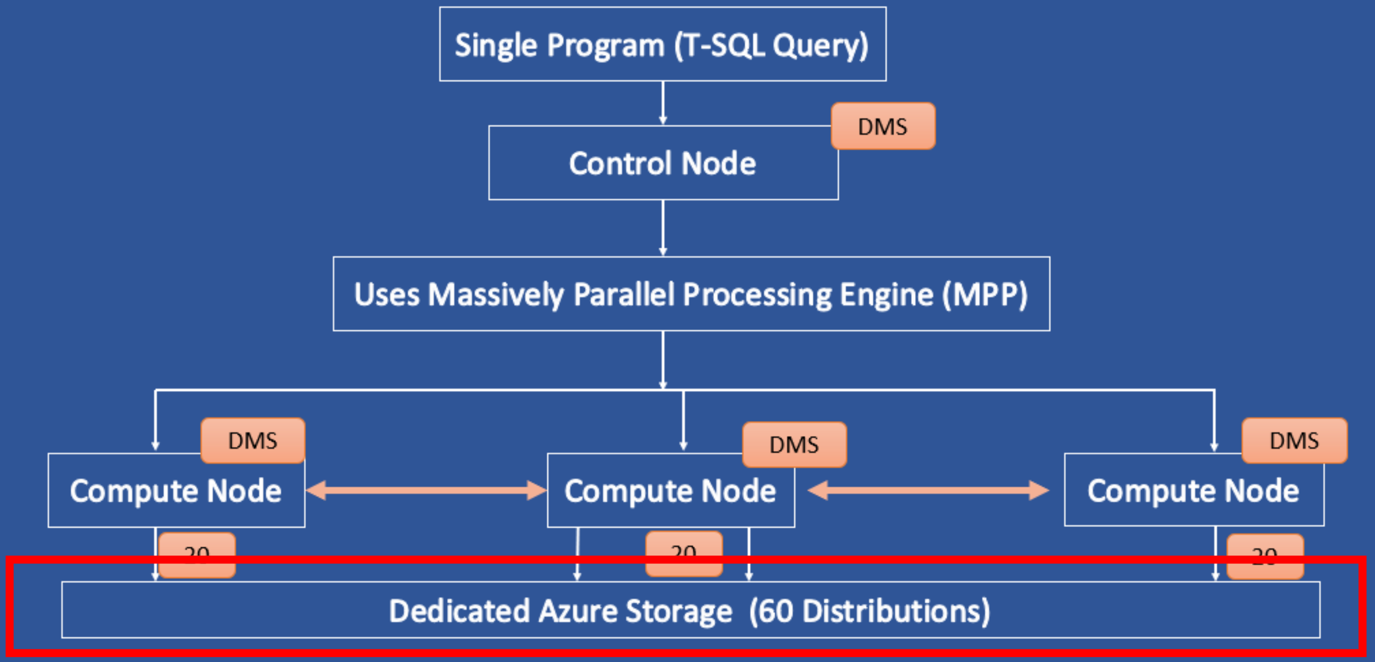
Author: Shanmukh Sattiraju

Clustered column store index

101,2013,Male, Vijay
102,2013,Male, Stark
103,2013,Female, Andrea
104,2014,Male,Steve

Author: Shanmukh Sattiraju

Sharding Pattern



60 distributions

These sharding patterns are:

- Hash
- Round Robin
- Replicate

Round Robin Distribution

CREATE TABLE StudenDetails
WITH (DISTRIBUTION = ROUND_ROBIN)
AS..

Student ID	Subject
101	Networking
102	Linux
103	Java
101	Azure

Hash Distribution

```
CREATE TABLE StudenDetails
WITH (DISTRIBUTION = HASH(StudentID)
AS..
```

Student ID	Subject
101	Networking
102	Linux
103	Java
101	Azure

Author: Shanmukh Sattiraju

Replicated Distribution

```
CREATE TABLE StudenDetails
WITH (DISTRIBUTION = REPLICATE)
AS..
```

Student ID	Subject
101	Networking
102	Linux
103	Java
101	Azure

Student ID	Subject
101	Networking
102	Linux
103	Java
101	Azure

Dist_1

Student ID	Subject
101	Networking
102	Linux
103	Java
101	Azure

Dist_2

Student ID	Subject
101	Networking
102	Linux
103	Java
101	Azure

Dist_3

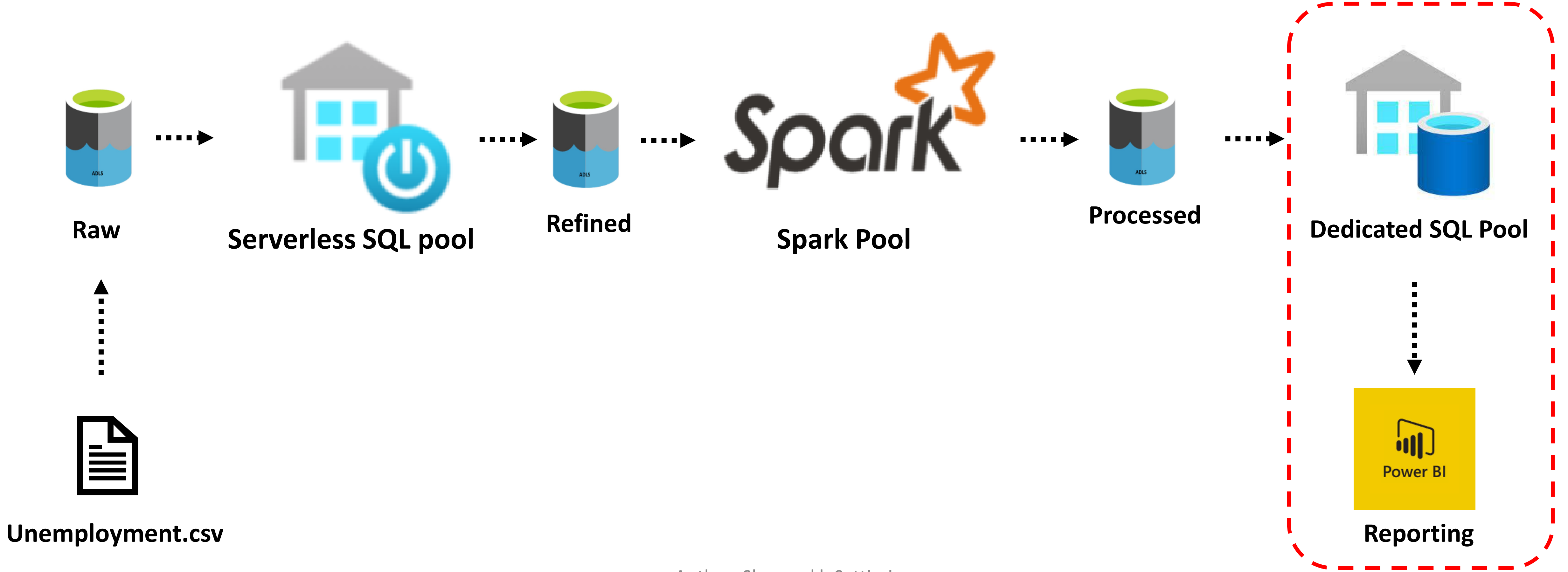
Sharding Patterns

Distribution	Description	Performance	Suited for
Round Robin	Distributes randomly row evenly across nodes, No logic on how data is distributed	Performance is not optimized	Staging Tables
Hash	Rows are distributed across nodes based on the hash column that we defined. 1 node = 1 hash value	Maximum query performance	Fact Tables
Replicated	Keeps copy of entire table in every node , 60 distributions makes 60 copies	Good performance if its used for small tables	Dimension Tables

Reporting data with Power BI

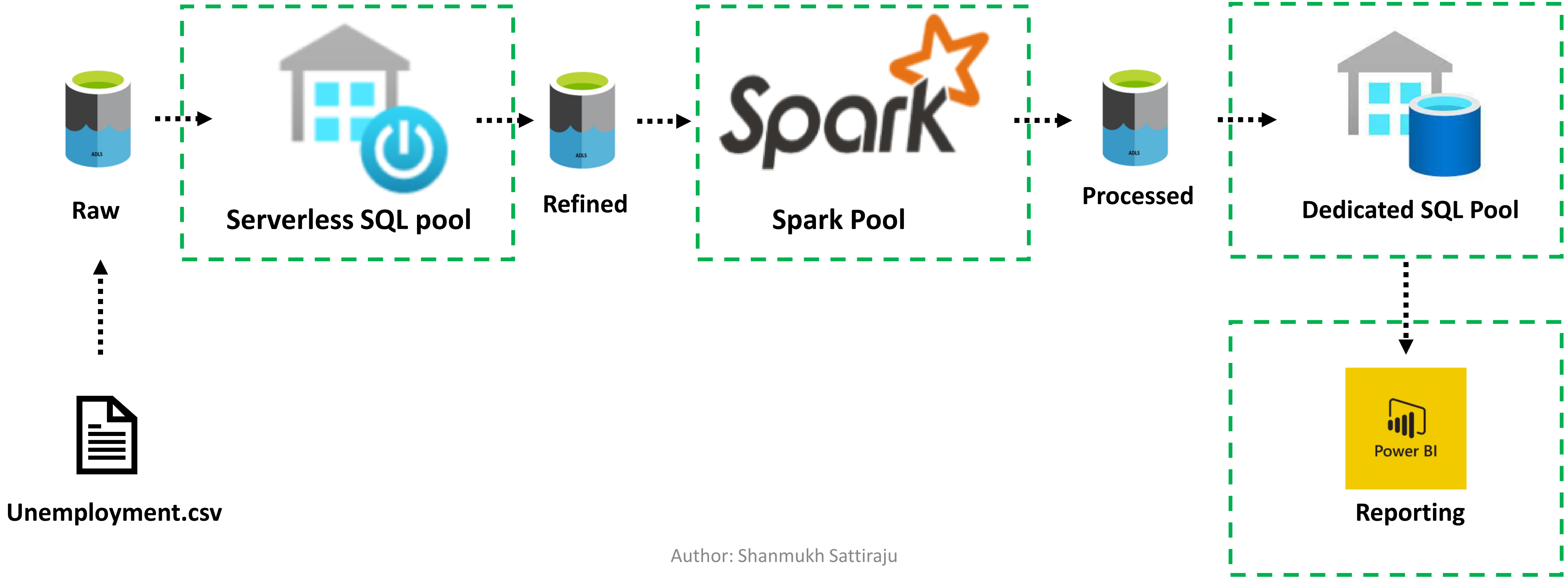
Author: Shanmukh Sattiraju

Project Architecture



Author: Shanmukh Sattiraju

Project Architecture



Author: Shanmukh Sattiraju

Spark Optimization Techniques

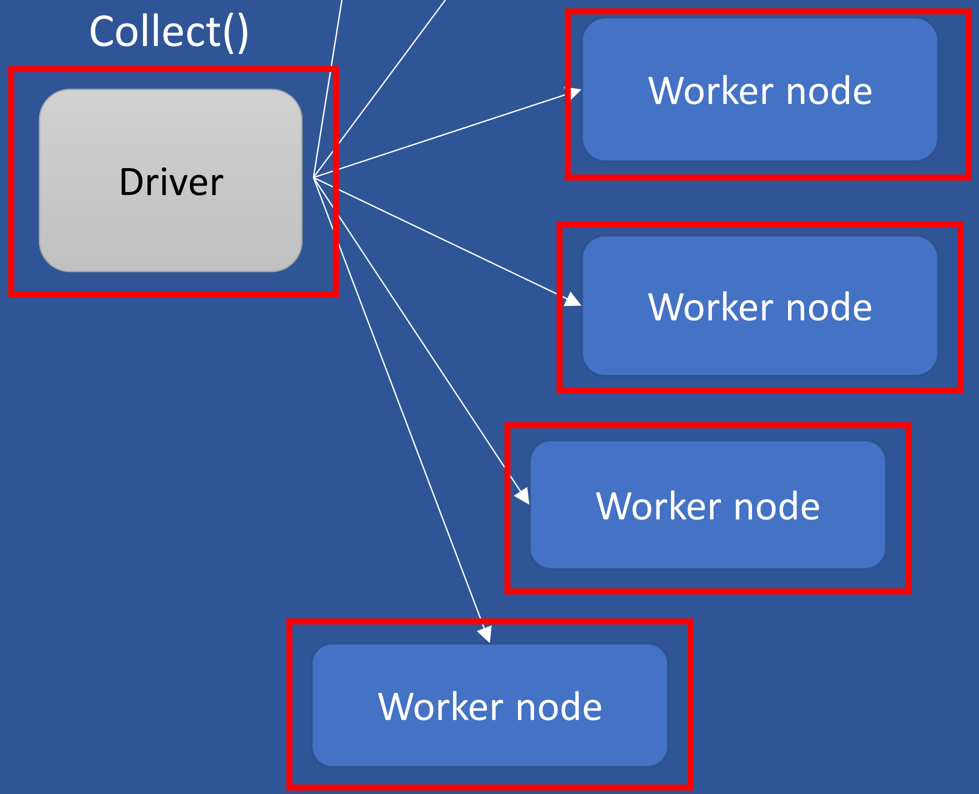
Author: Shanmukh Sattiraju

Spark can be optimized at 2 levels

1. Spark pool Optimization (For Synapse)
    - Choosing right node size
        - Number of vCores and Memory
    - Auto-scaling enabled
    - Number of nodes
2. Application or Code Level Optimization
    - Writing code to make efficient use of available resources

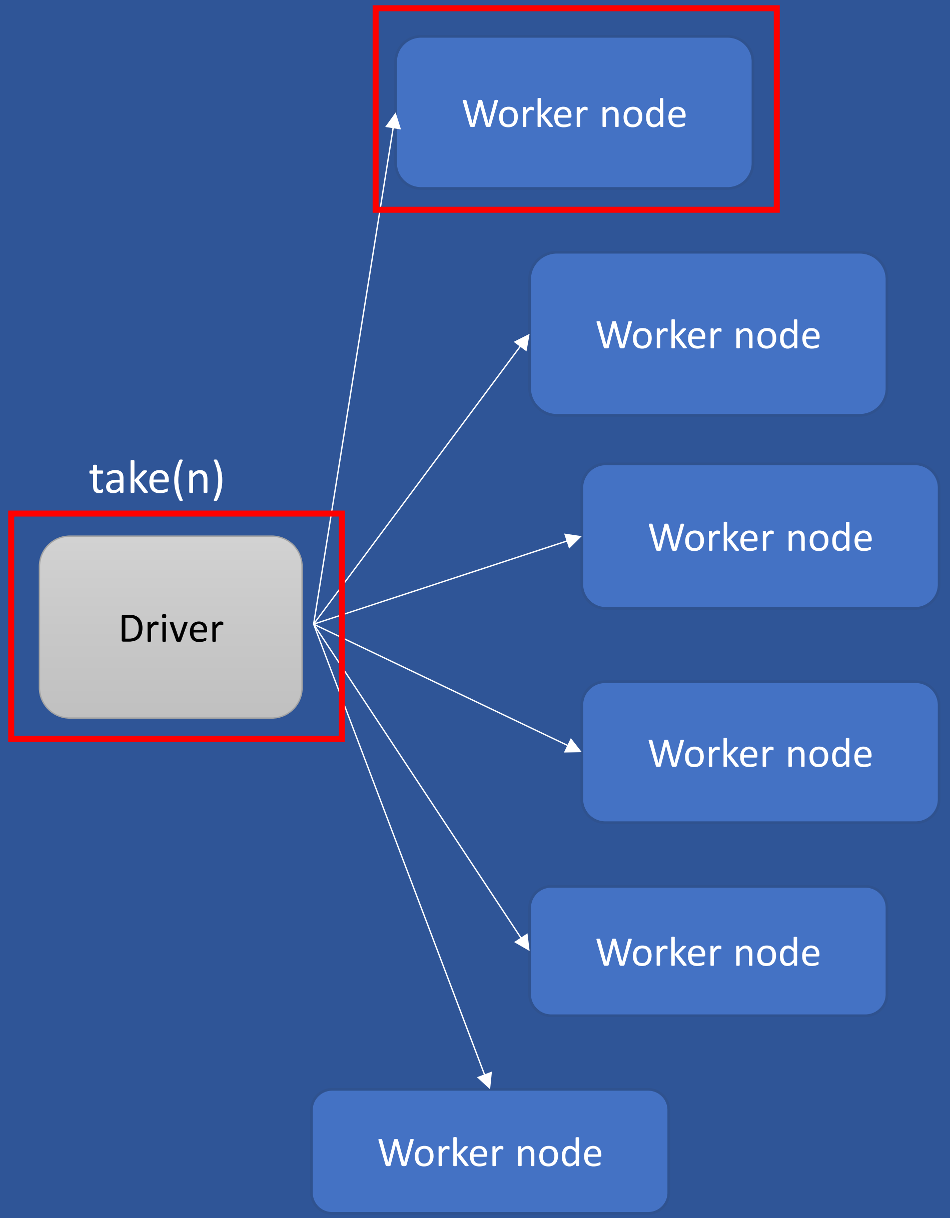
Avoid using collect()

Out of memory



Instead use take()

Returns first n elements



Avoid using InferSchema

Using inferSchema will:

- Invokes spark job and reads all the columns
- Takes lot time to load due to that.
- Will not provide accurate data types .e.g. date columns
- Not recommended for production notebooks

```
df = spark.read.format('csv')
        .option('header','true')\
        .option('inferSchema','true')\
        .load('<storage_path>')
```

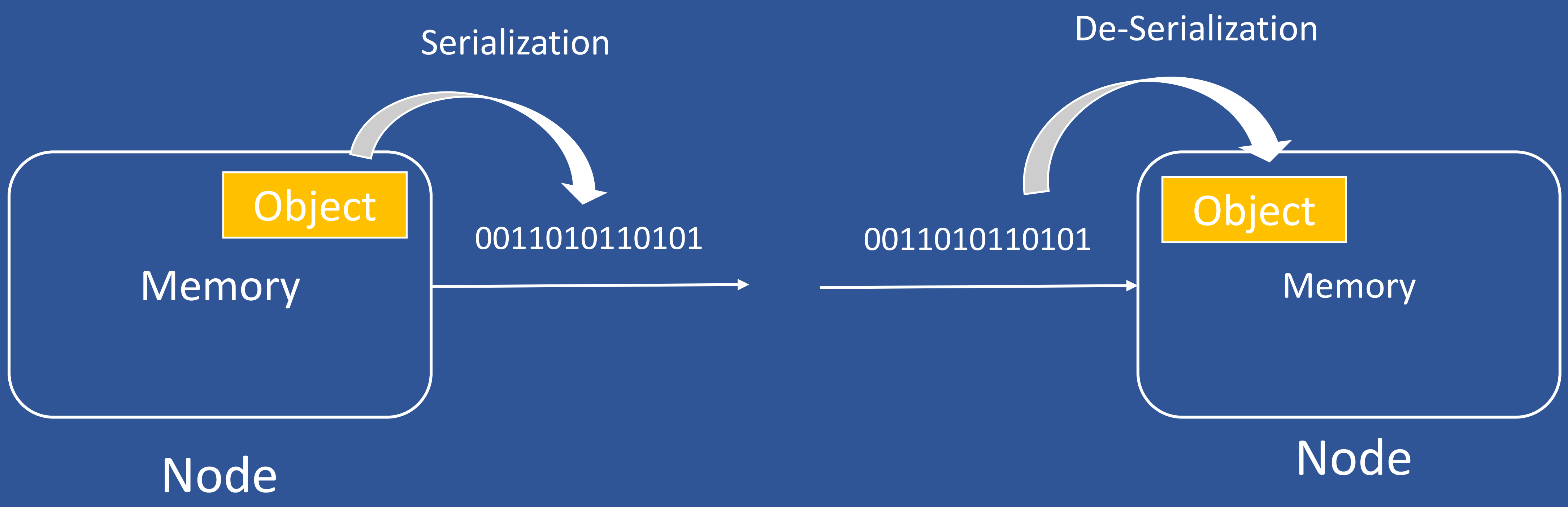
Best practice:

- Use StructType/StructField to enforce schema to columns

Data Serialization

Author: Shanmukh Sattiraju

Data Serialization



Author: Shanmukh Sattiraju

Cache and persist

- They allow you to store intermediate data in memory
- The stored data will be reused in subsequent actions which can significantly improve the performance of your Spark applications.
- Both caching and persisting are used to save
- Cache() saves data only in memory
- Persist() can save data with multiple storage levels (will see shortly)

How cache() and persist() works

Without cache() or persist()

```
df = spark.read.format('csv')\
      .option('header','true')\
      .load('abfss://raw@da..')
```

```
df_transform = df.withColumn()
```

```
df_dropped = df_transform\
                  .drop(..)
```

```
df_converted = df_dropped\
                 .withColumn(..)
```

```
df_converted.select()
```

```
df_converted.Filter()
```

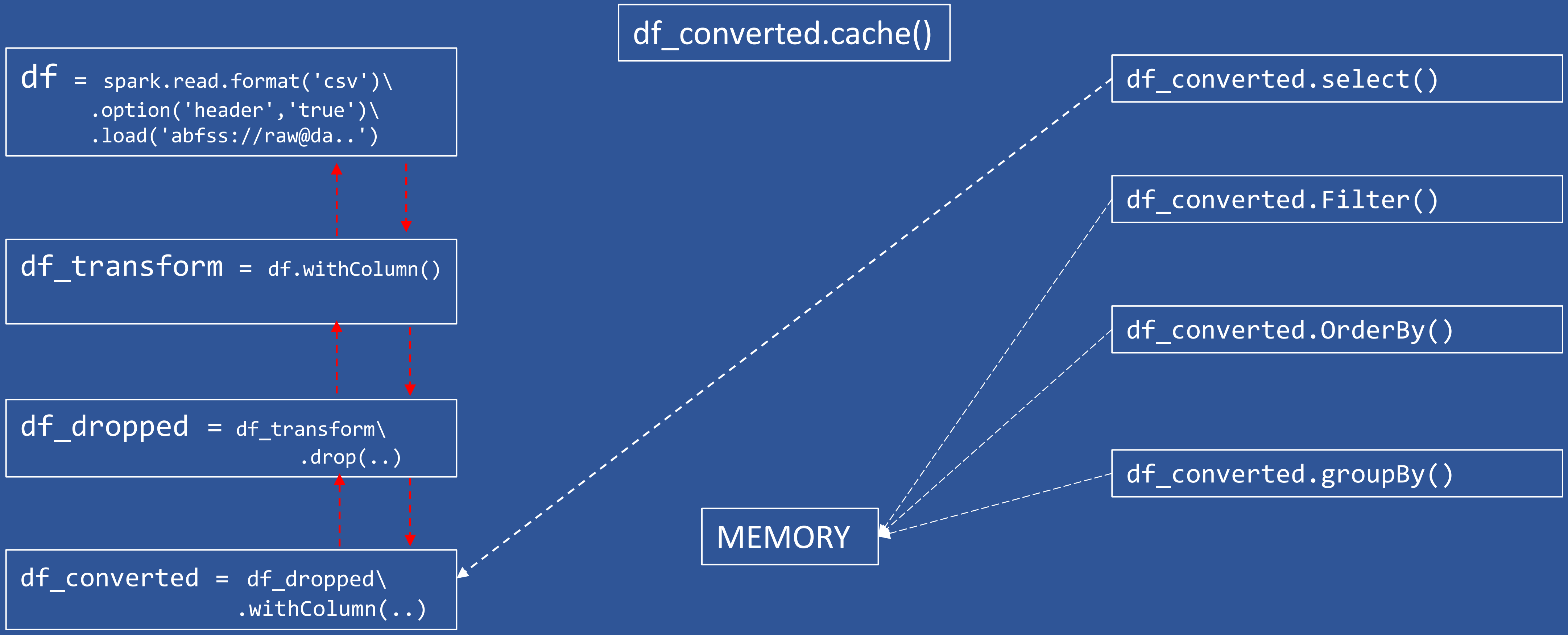
```
df_converted.OrderBy()
```

```
df_converted.groupBy()
```

Author: Shanmukh Sattiraju

How cache() and persist() works

With cache() or persist()



Author: Shanmukh Sattiraju

How cache() and persist() works

With cache() or persist()

Initially stored in memory and it will be re-used for **subsequent** actions
Why subsequent actions?

For 1st action = It will be computed once and stored in memory

2nd action = Instead of re-computing, retrieved from memory
.
.
.
6th Action = Instead of re-computing, retrieved from memory

Cache() vs persist()

- Cache() when used will store the data in MEMORY_ONLY
- Persist() when used will store data in different persistent levels or storage levels

Here persistent level means
– **Where** (memory / disk) and **how** (serialized or de-serialized) the data will be stored

Various persistent levels are:

MEMORY_ONLY

MEMORY_AND_DISK

MEMORY_ONLY_SER (Java, Scala)

MEMORY_AND_DISK_SER (Java, Scala)

DISK_ONLY

OFF_HEAP

Usage:

```
df.persist(StorageLevel.MEMORY_ONLY)
df.persist(StorageLevel. MEMORY_AND_DISK
.
.
.
.
.
.
df.persist(StorageLevel.OFF_HEAP)
```

Author: Shanmukh Sattiraju

As per Spark documentation

In Python, stored objects will always be serialized with the Pickle library, so it does not matter whether you choose a serialized level.

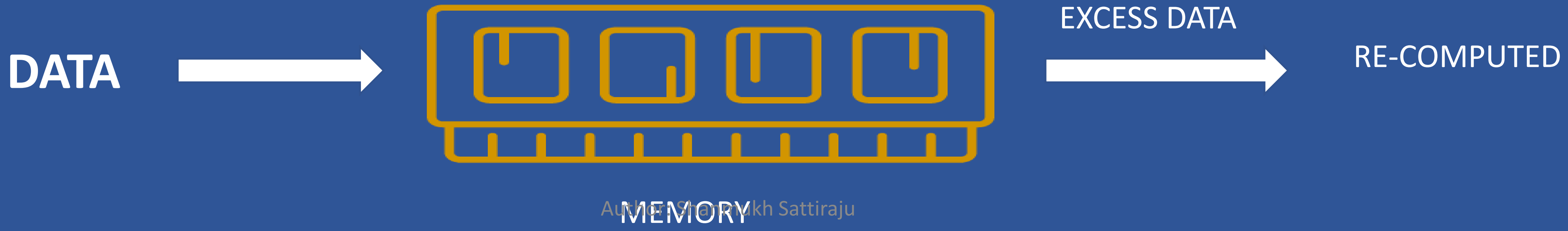
The available storage levels in Python (PySpark) include

- MEMORY_ONLY
- MEMORY_ONLY_2
- MEMORY_AND_DISK
- MEMORY_AND_DISK_2
- DISK_ONLY
- DISK_ONLY_2
- DISK_ONLY_3.

Persistant Levels

MEMORY_ONLY

- In memory it is stored as **de-serialized** objects (Java / Scala)
- In memory it is stored as **serialized** objects (PySpark)
- Any excess data the doesn’t fit into memory is re-computed
- When MEMORY_ONLY is used it is same as using cache() (in functionality)
    - Using cache() from PySpark will store them in de-serialized format.
- **Usage**: df.persist(StorageLevel.MEMORY_ONLY))
- **Best suitable use case** : Interactive Data Exploration



Understanding StorageLevel

StorageLevel (<useDisk>,<useMemory>,<useOffHeap>,<de-serialized>,<replication>)

For MEMORY_ONLY

StorageLevel (*false* , *true* , *false* , *false* ,*1*)

useDisk

useMemory

useOffHeap

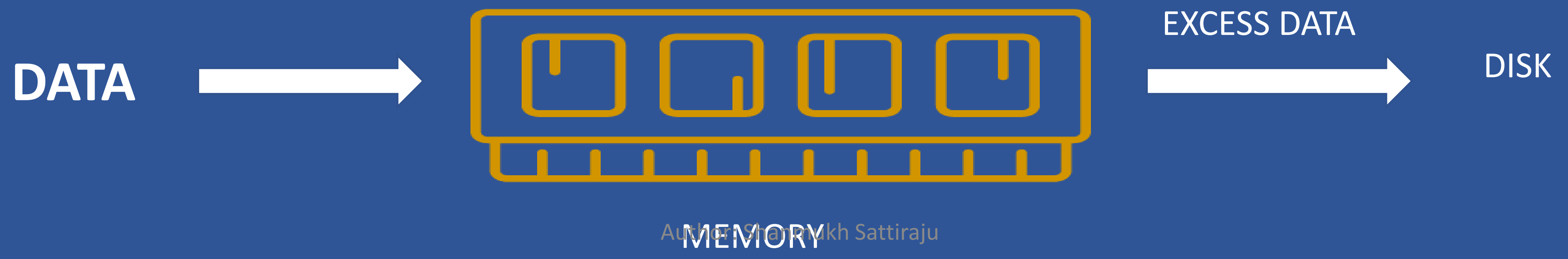
deserialized

replication

Persistant Levels

MEMORY_AND_DISK

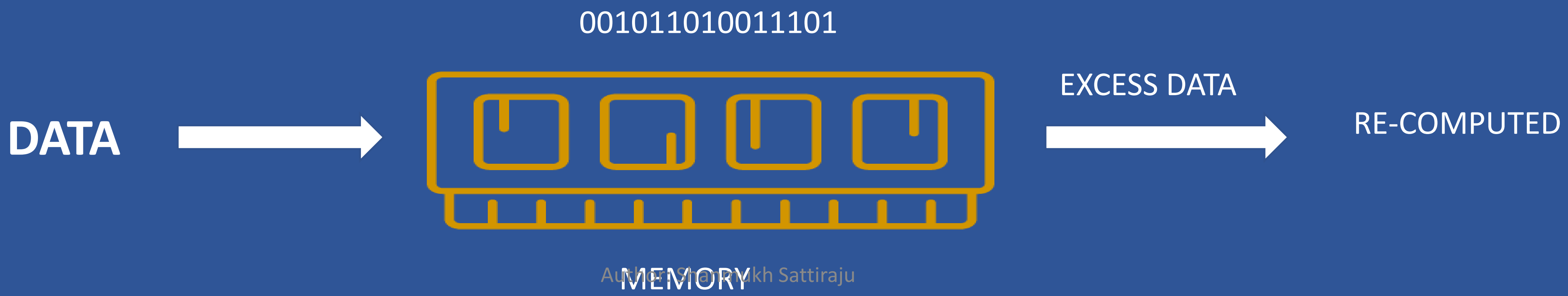
- First the data will be stored in memory as :
    - **de-serialized** objects (Java or Scala)
    - **Serialized** Object (PySpark)
- Any excess data the doesn’t fit into memory is sent to Disk (nothing but storage)
- **Usage**: df.persist(StorageLevel.MEMORY_AND_DISK))
- **Best suitable use case:** Machine Learning Training



Persistant Levels

MEMORY_ONLY_SER

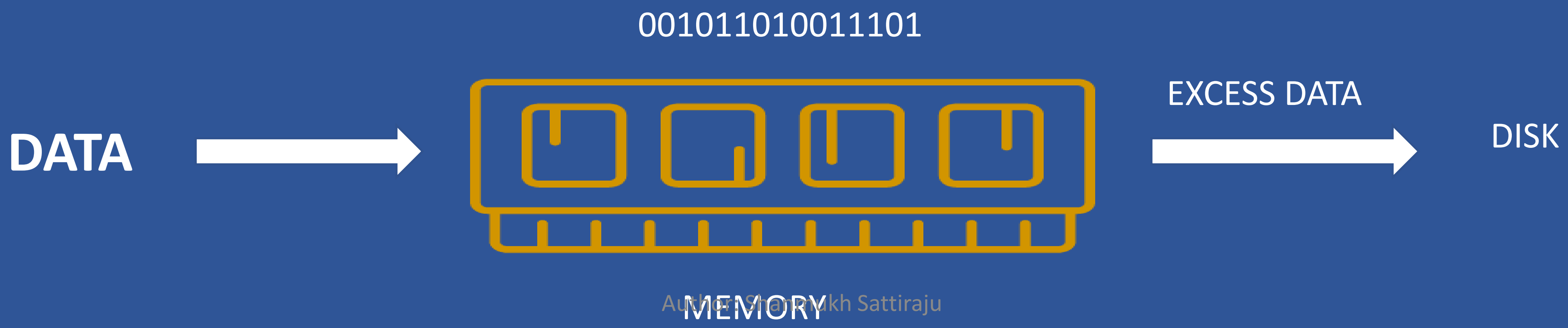
- Similar to MEMORY_ONLY using PySpark
- But data here will be stored as ‘**serialized** object’ (Java or Scala)
- This is not available in PySpark because it is already serialized by using Python
- Any excess data the doesn’t fit into memory is recomputed
- **Usage**: df.persist(StorageLevel.MEMORY_ONLY_SER))
- **Best suitable use case:** Serialize data for memory optimization



Persistant Levels

MEMORY_AND_DISK_SER

- Similar to MEMORY_AND_DISK in PySpark
- But data here will be stored as ‘**serialized** object’ (Java or Scala)
- This is not available in PySpark because it is already serialized by using Python
- Any excess data the doesn’t fit into memory is sent to disk (storage)
- **Usage**: df.persist(StorageLevel.MEMORY_AND_DISK_SER))
- **Best suitable use case:** When you want to reduce memory usage by storing data to disk



Persistant Levels

DISK_ONLY

- Stores only on disk
    - Serialized object in both Scala and PySpark
- **Usage**: df.persist(StorageLevel.DISK_ONLY))
- **Best suitable Use Case:** When you have large datasets that doesn't fit into memory

DATA → **DISK**

Persistant Levels

OFF_HEAP

- Stores only on OFF_HEAP
- **Usage**: df.persist(StorageLevel.OFF_HEAP))
- **Best suitable use case:** Off-Heap Storage for Extremely Large Datasets

DATA → **OFF HEAP MEMORY**

Remaining Persistent Levels of PySpark

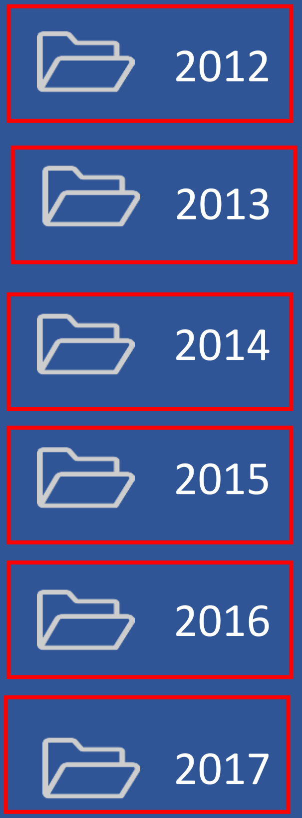
- MEMORY_ONLY_2
- MEMORY_AND_DISK_2
- DISK_ONLY_2
- DISK_ONLY_3.

Partitioning

- Partitioning is a way to split data into separate folders based on one or multiple columns.
- Each of the partitions is saved into a separate folder when partitioned.
- Optimizes the queries by skipping reading parts of the data that are not required.

Year	Month	Unemployed
2012	Jan	211741
2013	Jan	451751
2014	Jan	51652
2015	Jan	5174
2016	Jan	21657
2017	Jan	45868
2018	Jan	87474

Partition on Year column



Author: Shanmukh Sattiraju

Partitioning

Which column to choose of partitioning?

- A column that is used frequently in filtering
- No of distinct values in partitioned column = No of partitions = No of Folders.
- A column which have less distinct values (low cardinality)

Which column to avoid for partitioning?

- A column that is having many distinct values
- This will make too many partitions and make it less efficient in querying

Repartition and coalesce

Repartition()

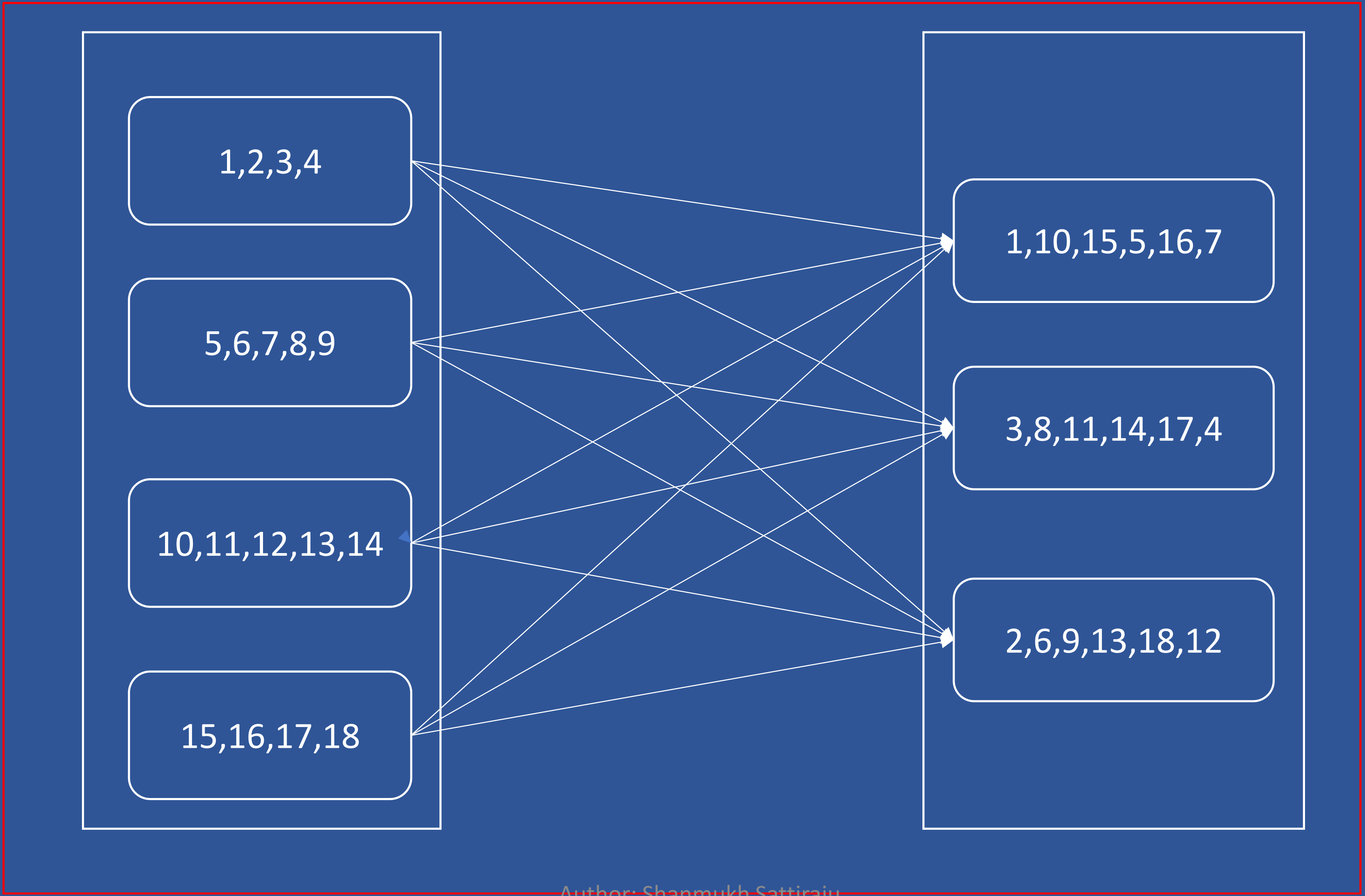
- Repartition is a transformation API that can be used to increase or decrease the number of partitions in a dataframe/RDD.

Coalesce()

- Coalesce is a transformation API that can be used to decrease the number of partitions in a dataframe/RDD.

Repartition

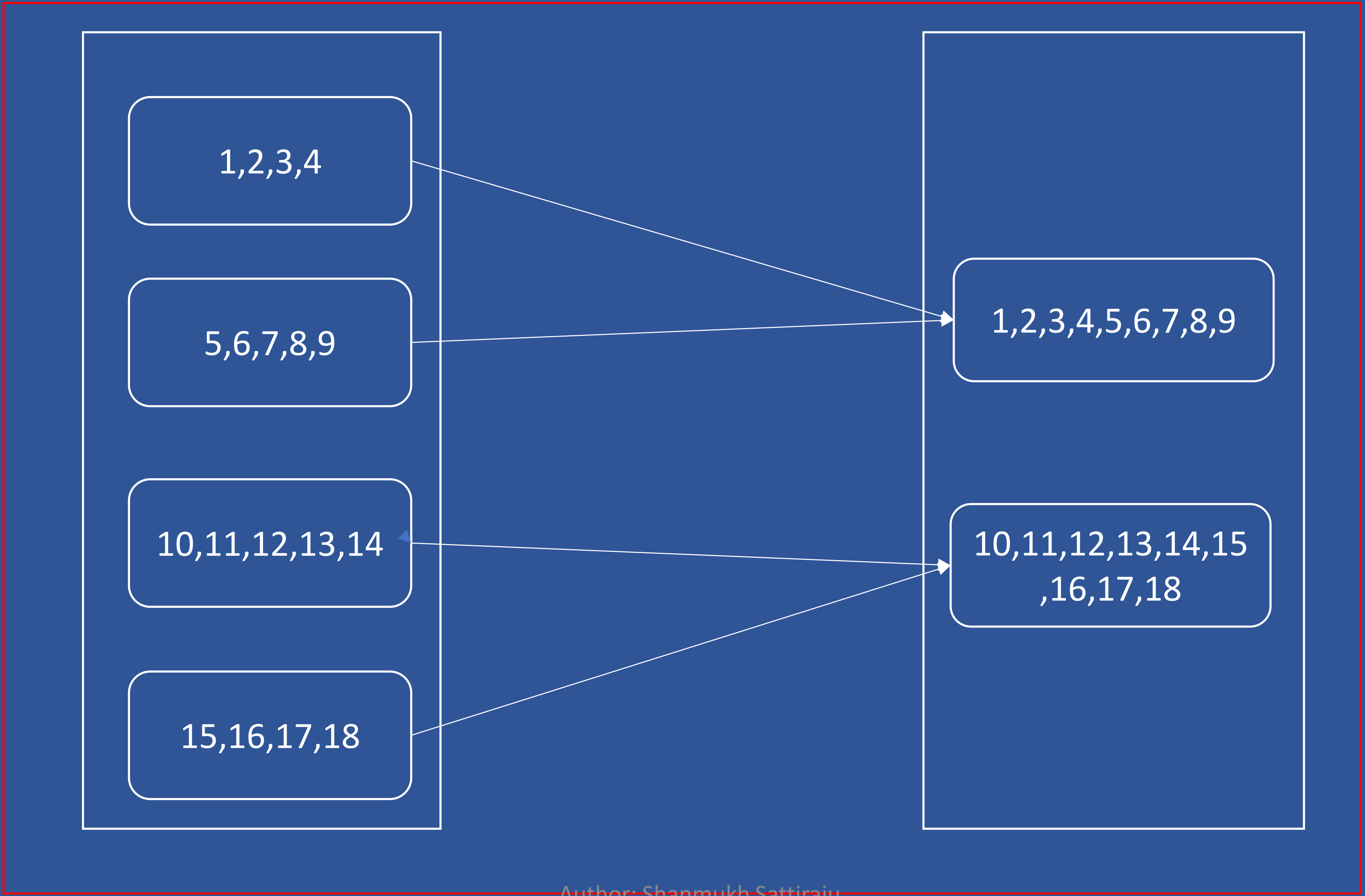
Wide Transformation



Author: Shanmukh Sattiraju

coalesce

Narrow Transformation



Author: Shanmukh Sattiraju

	Repartition	Coalesce
Purpose	Reduce or increase partition numbers by performing a full shuffle.	Only reduces number of partitions on dataframe and avoids shuffling
Shuffle	Performs a full shuffle, which can be an expensive operation.	Does not perform a full shuffle.
Number of Partitions	Can increase or decrease the number of partitions in a DataFrame.	Only decreases the number of partitions
Data Movement	Moves data across the network to create the new partitioning scheme.	Tries to minimize data movement and avoid shuffling whenever possible.
Performance	Generally slower compared to coalesce due to the full shuffle operation.	Generally faster compared to repartition since it avoids shuffling whenever possible.

Broadcast variables

CA - California
NY - New York

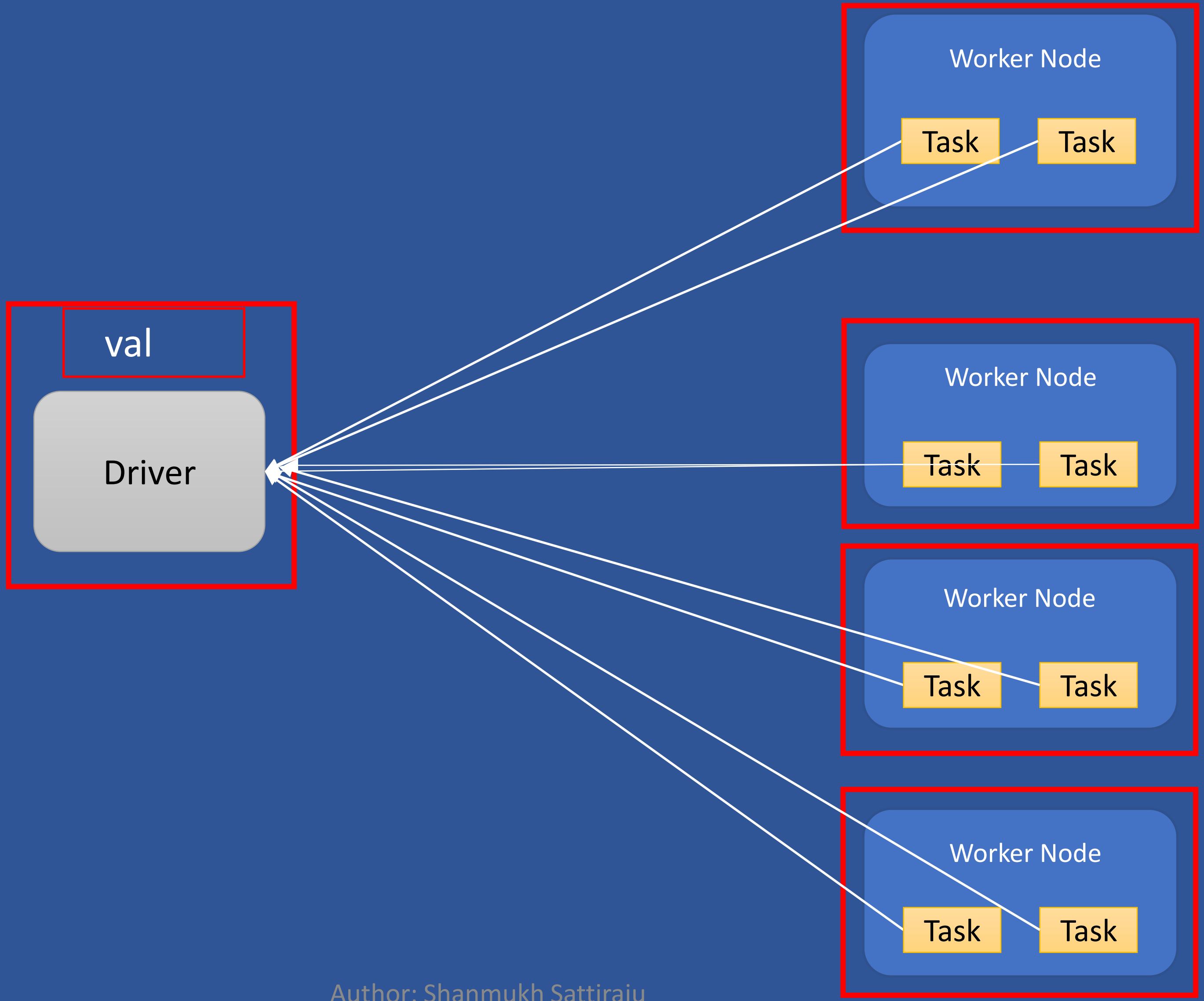
State	State_Name
CA	California
NY	New York

Author: Shanmukh Sattiraju

Broadcast variables

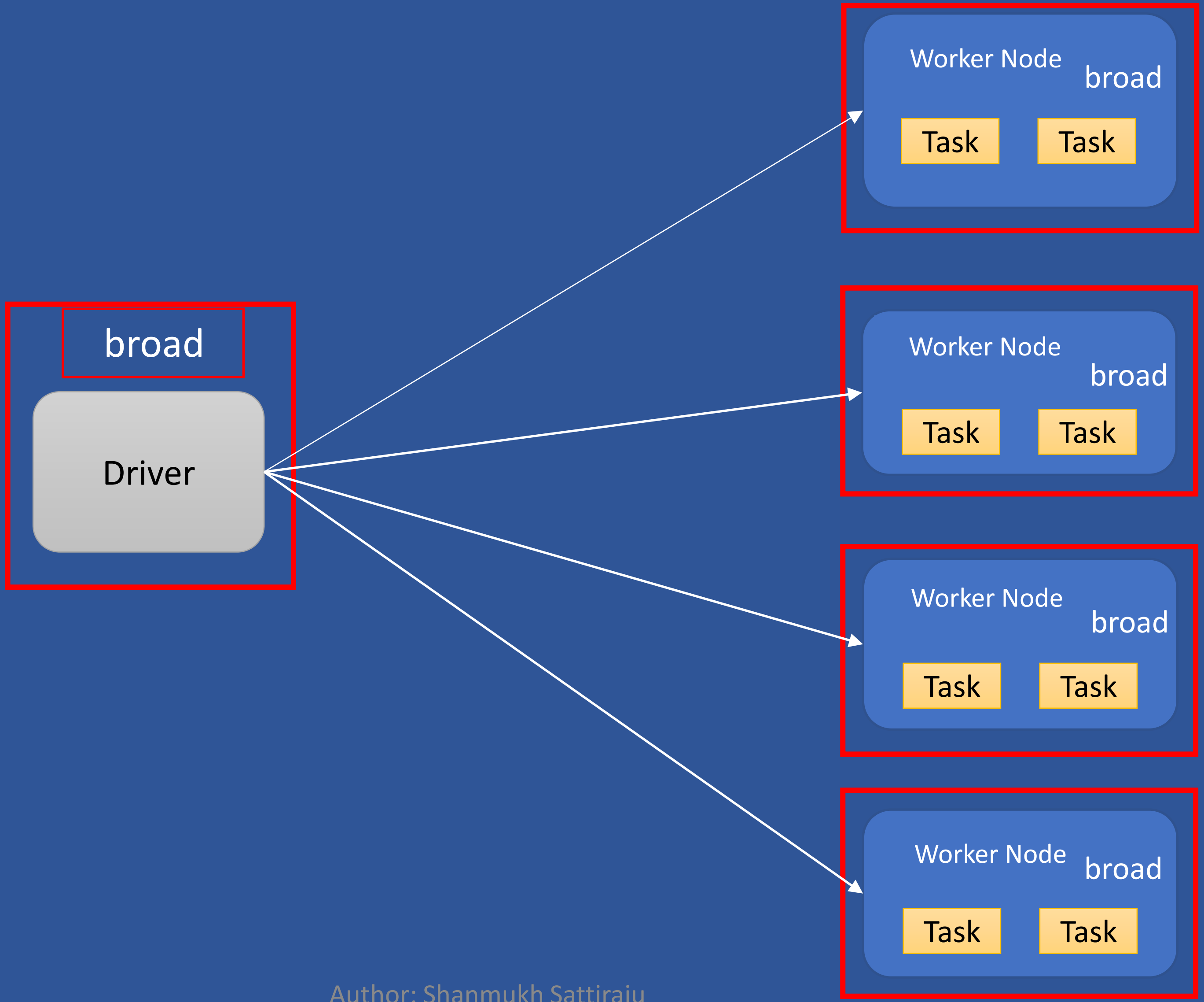
Without broadcast

val



Broadcast variables

broad



Author: Shanmukh Sattiraju

Broadcast variables

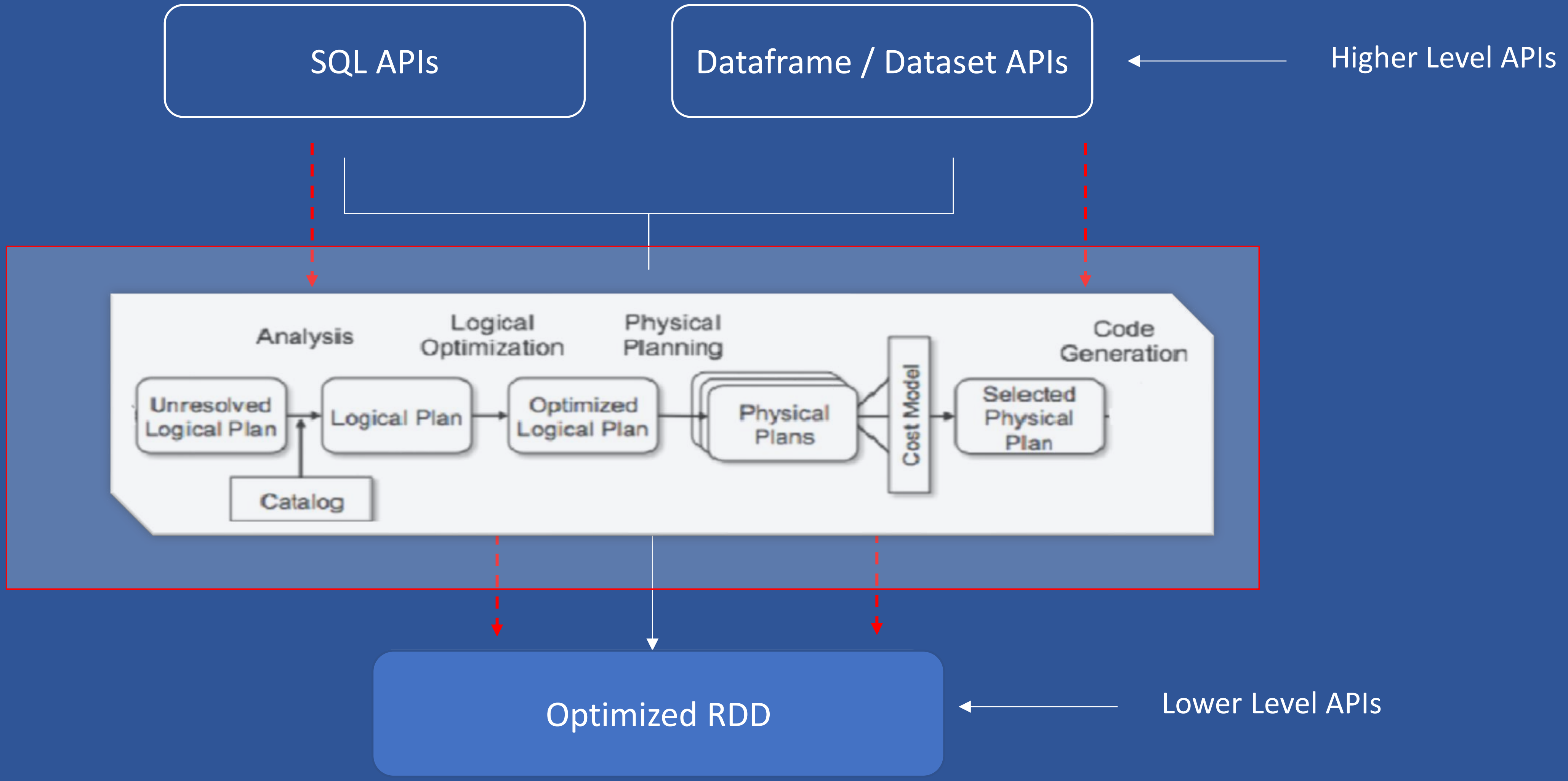
- Read-only variables cached on each machine
- Access the value in them using .value[]
- Cached on each worker node in serialized form
- Useful when a dataset needs to be shared across all nodes
- Reduces the data transfer

Serialization Types

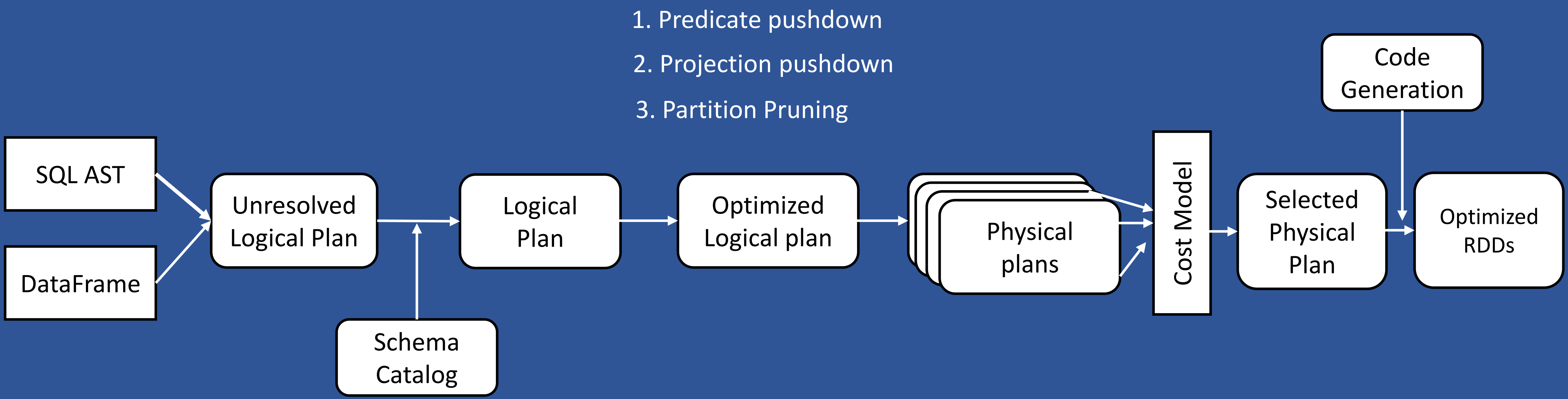
- **Java Serialization**
  - Default serialization technique used by spark
  - Less performative than Kryo
  - Not efficient in space-utilization

- **Kryo Serialization**
  - Faster and more efficient
  - Takes less time to convert object to byte stream hence faster
  - Since Spark 2.0, the framework had used Kryo for all internal shuffling of RDDs, DataFrame with simple types, arrays of simple types, and so on.
  - Spark also provides configurations to enhance the Kryo Serializer as per our application requirement.

Catalyst Optimizer



Catalyst Optimizer



Join

A	B	C	D

A	E	F

A	B	C	D	E	F

Author: Shanmukh Sattiraju

Join Strategies

1. Shuffle Joins

   A. Sort-merge Join

   B. Shuffle hash join

2. Broadcast hash join

3. Cartesian Join

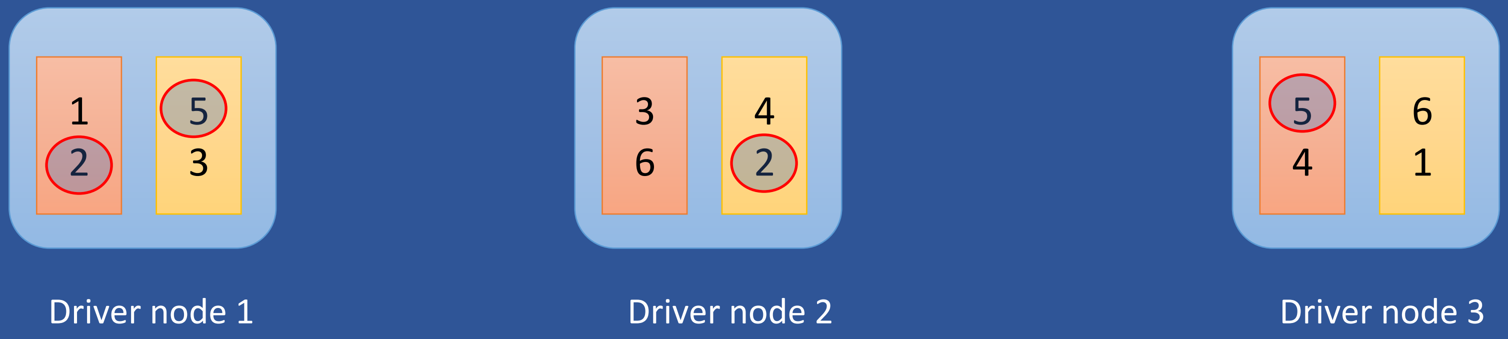
Sort-merge Join

df1

1	Jack
2	Ram
3	Ted
6	Raj
5	Ben
4	Nick

df2

4	1000
2	750
1	940
3	490
6	649
5	749



Author: Shanmukh Sattiraju

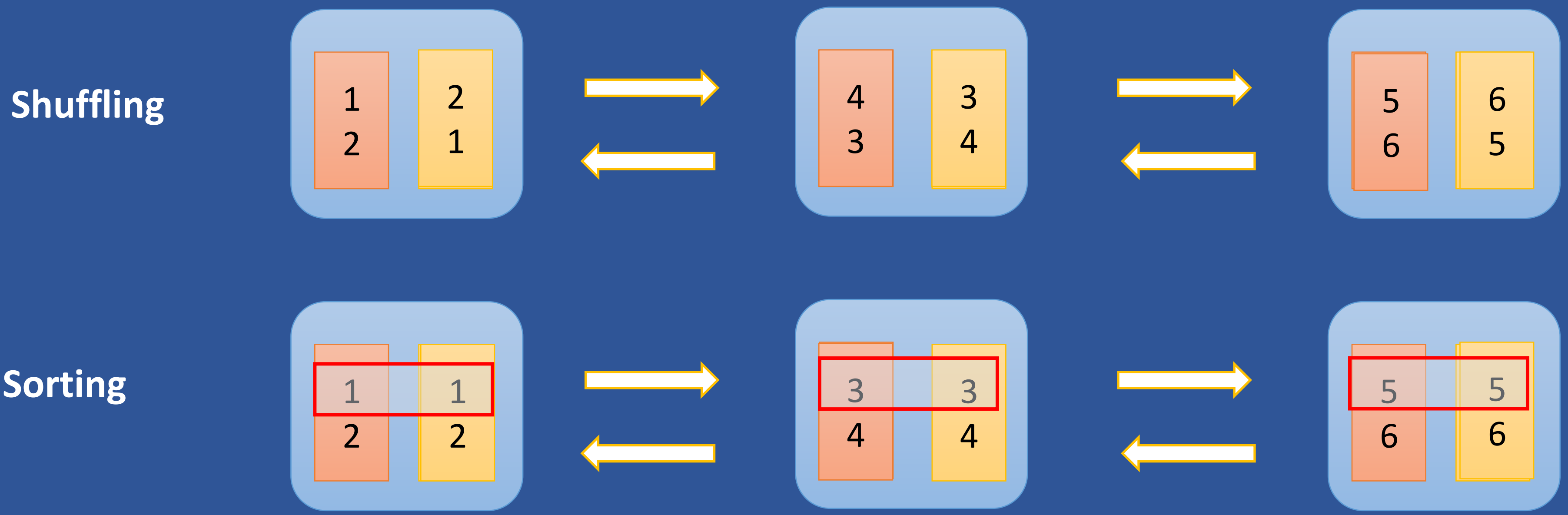
Sort-merge Join

df1

1	Jack
2	Ram
3	Ted
6	Raj
5	Ben
4	Nick

df2

4	1000
2	750
1	940
3	490
6	649
5	749



Author: Shanmukh Sattiraju

Sort-merge Join

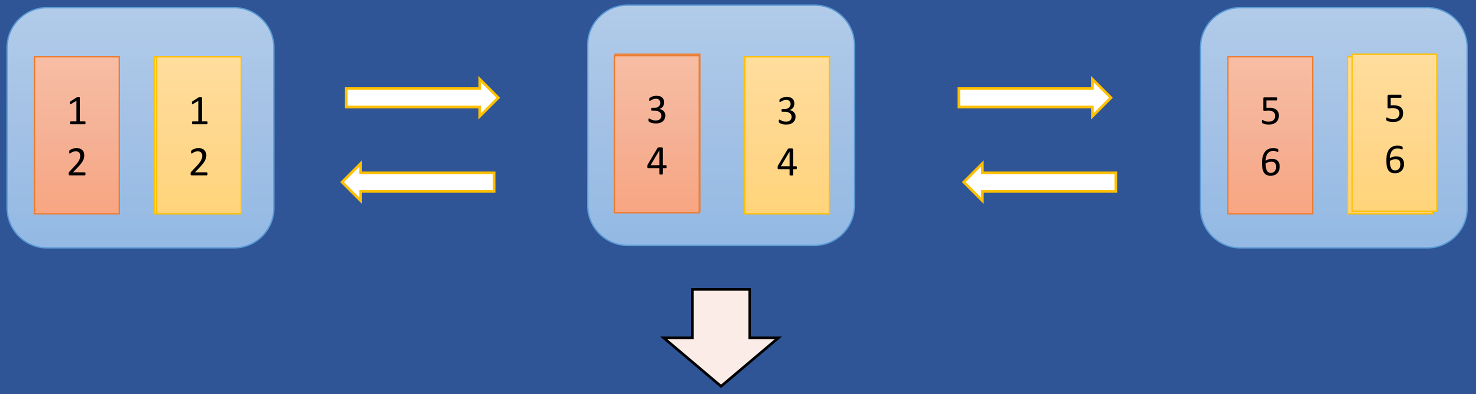
df1

1	Jack
2	Ram
3	Ted
6	Raj
5	Ben
4	Nick

df2

4	1000
2	750
1	940
3	490
6	649
5	749

Sorted data



Merge

1	Jack	940
2	Ram	750
3	Ted	490
4	Nick	1000
5	Ben	749
6	Raj	649

Sort-Merge Join

- Shuffles both the dataframe to bring records of same key into same paritition
- Sorts them in order
- Once sorted , merges data to join the dataframes

Pros:

- Can handle Large datasets on both sides

Cons:

- Need to shuffle and sort data which is time taking
- May result in data skewness

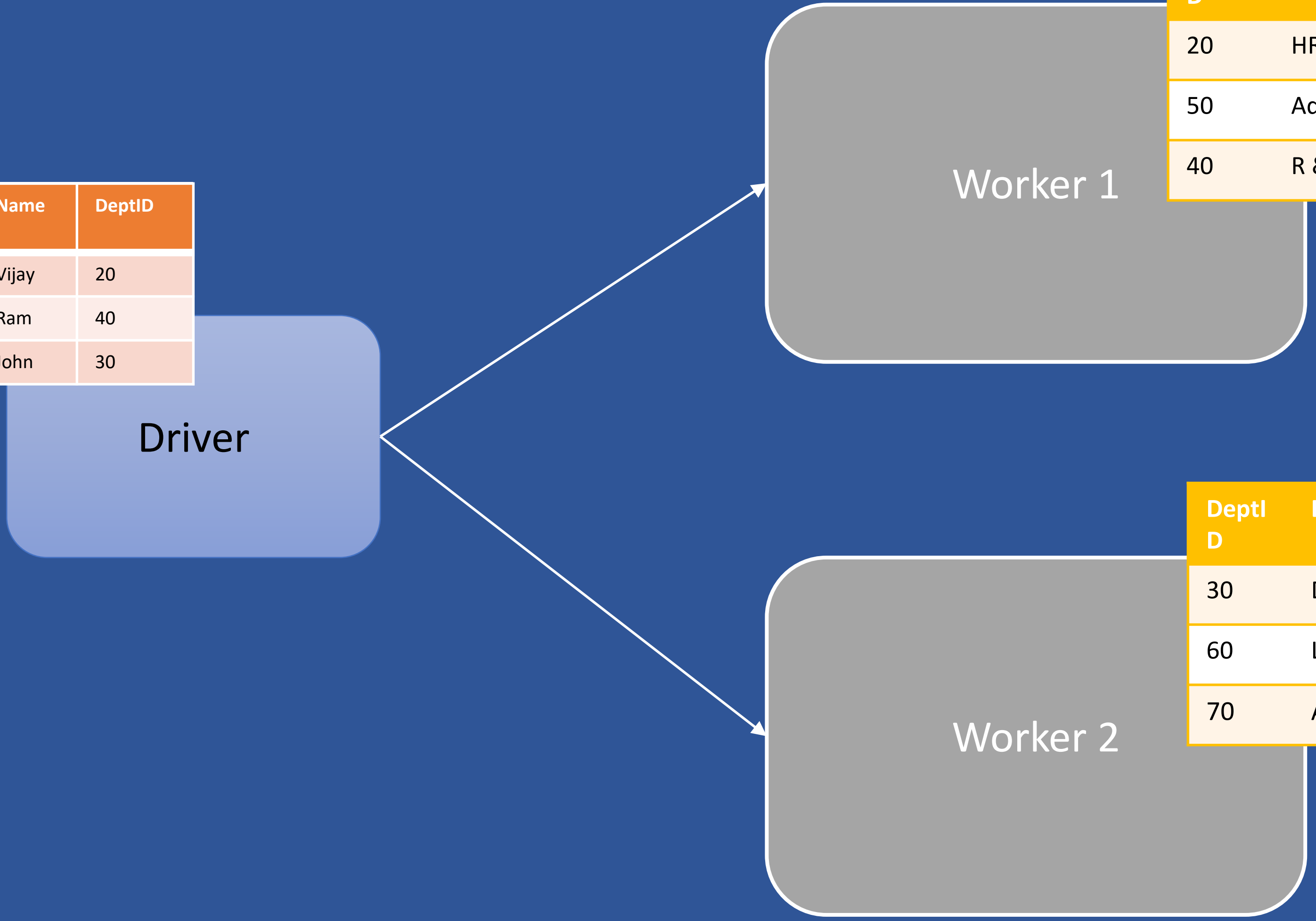
Broadcast Join

ID	Name	DeptID
1	Vijay	20
2	Ram	40
3	John	30

DeptID	DeptName	Location	Established	Contact
20	HR	Chennai	20/01/2022	+918750458
50	Admin	Mexico	05/06/2000	+915471256
40	R & D	Dallas	08/11/1998	+918740574
30	D & A	Chicago	15/04/2021	+919498724
60	Library	China	14/05/2009	+914578215
70	Accounts	Russia	05/09/2014	+917452475

Broadcast Join

ID	Name	DeptID
1	Vijay	20
2	Ram	40
3	John	30



DeptID	DeptName	Location	Established	Contact
20	HR	Chennai	20/01/2022	+918750458
50	Admin	Mexico	05/06/2000	+915471256
40	R & D	Dallas	08/11/1998	+918740574

DeptID	DeptName	Location	Established	Contact
30	D & A	Chicago	15/04/2021	+919498724
60	Library	China	14/05/2009	+914578215
70	Accounts	Russia	05/09/2014	+917452475

Author: Shanmukh Sattiraju

Broadcast Join

ID	Name	DeptID
1	Vijay	20
2	Ram	40

Hashing

DeptID	DeptName	Location	Established	Contact
20	HR	Chennai	20/01/2022	+918750458
		xico	05/06/2000	+915471256
		las	08/11/1998	+918740574

ID	Name	DeptID	DeptName	Location	Established	Contact
1	Vijay	20	HR	Chennai	20/01/2022	+918750458
2	Ram	40	R & D	Dallas	08/11/1998	+918740574
3	John	30	D & A	Chicago	15/04/2021	+919498724

Driver

ID	Name	DeptID
1	Vijay	20
2	Ram	40
3	John	30

Hashing

DeptID	DeptName	Location	Established	Contact
30	D & A	Chicago	15/04/2021	+919498724
60	Library	China	14/05/2009	+914578215
70	Accounts	Russia	05/09/2014	+917452475

Worker 2

spark.sql.autoBroadcastJoinThreshold = 10 MB default

Broadcast Hash Join

- Broadcast the smaller datasets to all the worker nodes
- When you only want to perform an equi-join, to combine two data sets based on matching keys

Pros:

- No shuffling or sorting in required
- No data skewness

Cons:

- Out Of Memory Issue (OOM)
- No support for Full Outer join

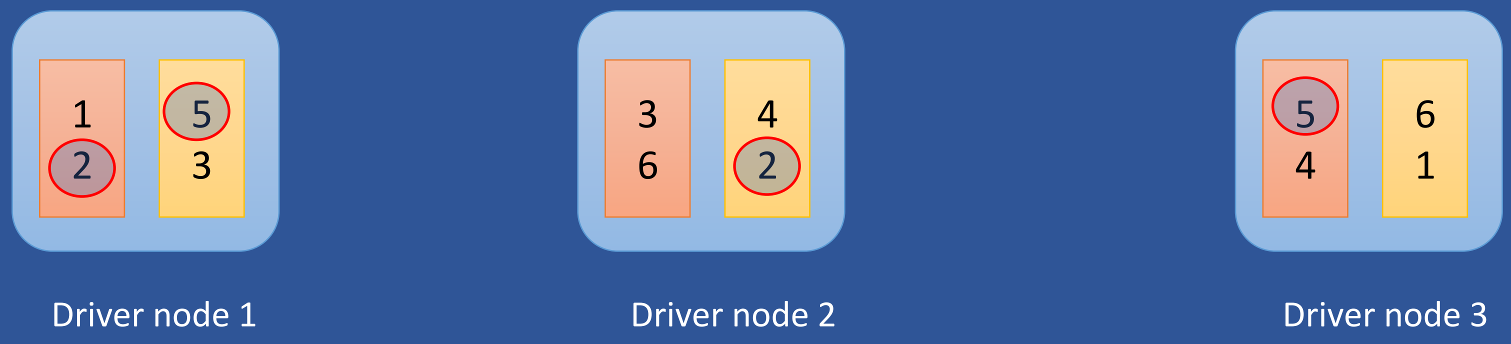
Shuffle Hash Join

df1

1	Jack
2	Ram
3	Ted
6	Raj
5	Ben
4	Nick

df2

4	1000
2	750
1	940
3	490
6	649
5	749



Author: Shanmukh Sattiraju

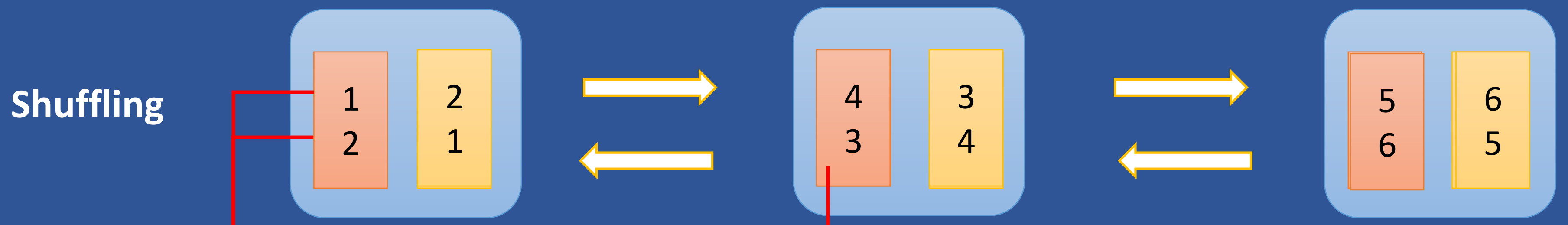
Shuffle hash join

df1

1	Jack
2	Ram
3	Ted
6	Raj
5	Ben
4	Nick

df2

4	1000
2	750
1	940
3	490
6	649
5	749



Key	Hash value
1	2141
2	54127
3	21557

In-memory

Shuffle hash join

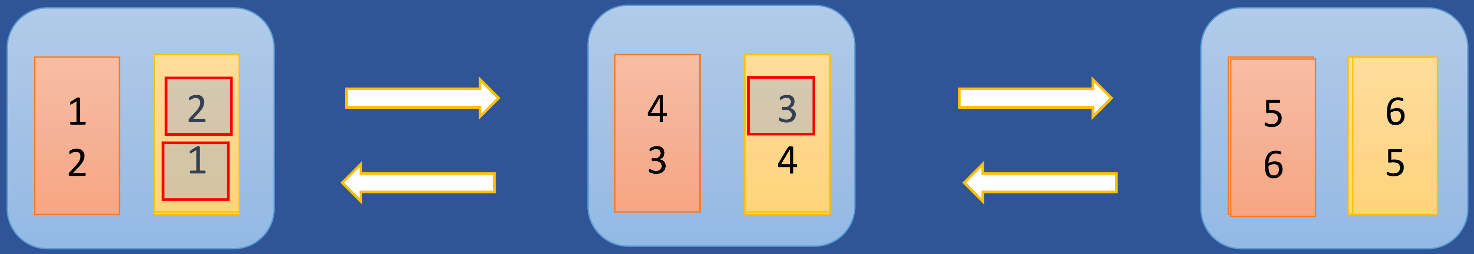
df1

1	Jack
2	Ram
3	Ted
6	Raj
5	Ben
4	Nick

df2

4	1000
2	750
1	940
3	490
6	649
5	749

Shuffling



Key	Hash value
1	2141
2	54127
3	21557

2	Ram	750
1	Jack	940
3	Ted	490
4	Nick	1000
5	Ben	749
6	Raj	649

Shuffle Hash Join

- Shuffles both the dataframe to bring records of same key into same paritition
- Created a hash table based on one dataset
- Joins data based on hash key

Pros:

- Can handle Large datasets
- Prevents data skewness

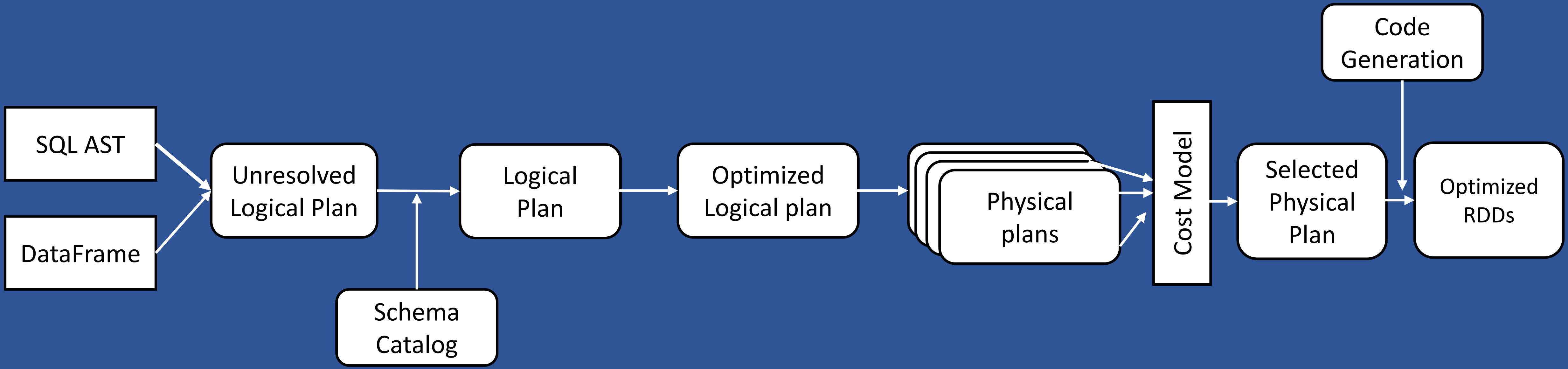
Cons:

- Need to shuffle
- OOM issue

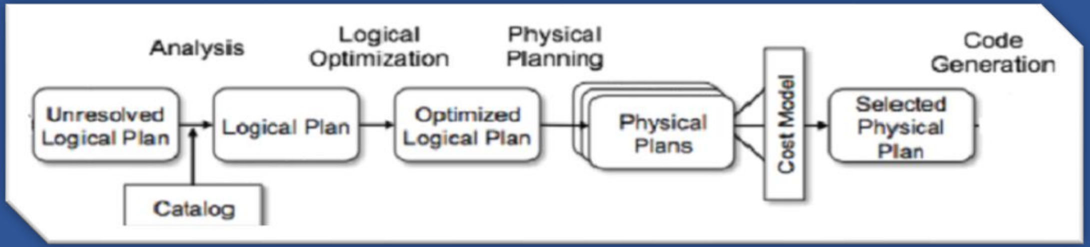
How to decide the strategy?

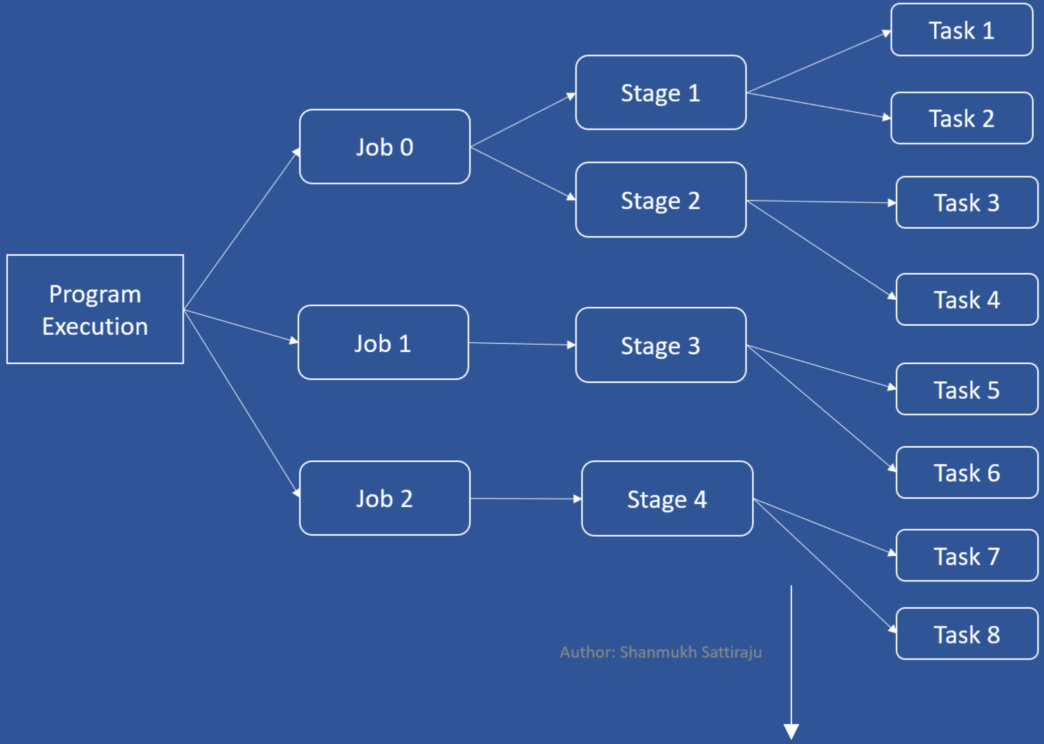
- **Shuffle hash join:**
  - Should be an equi-join
  - When there is data skew
  - Large datasets on both sides
- **Broadcast hash join:**
  - When one dataset is small and can fit into memory
  - Skew free data
- **Shuffle sort-merge join:**
  - Data already sorted
  - Comparable Key Ranges

Adaptive Query Execution (AQE)



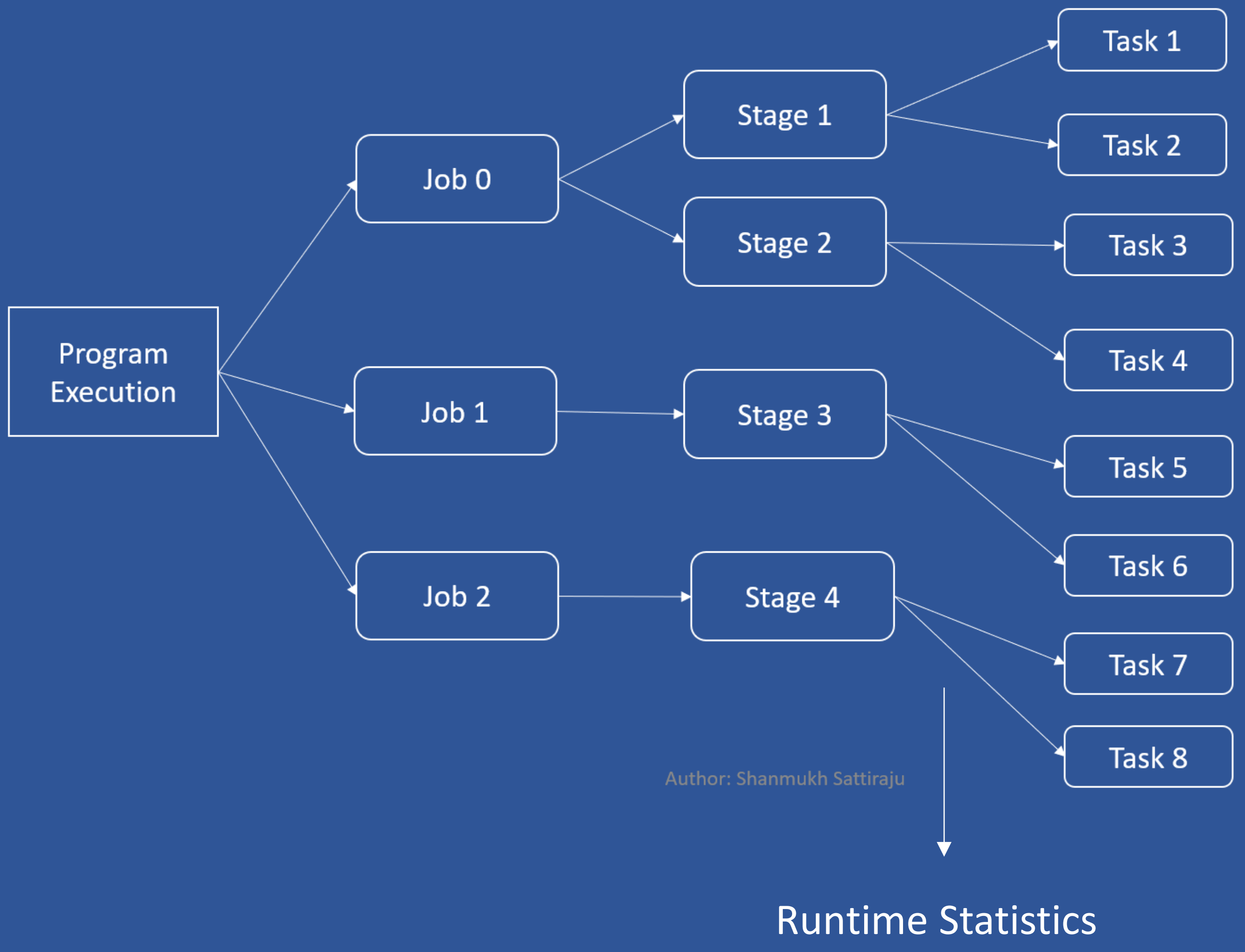
Adaptive Query Execution





Runtime Statistics

Adaptive Query Execution



Adaptive Query Execution

AQE aims to do following Optimizations during runtime:

- Dynamically coalesce shuffle partitions
- Dynamically switch join strategies
- Dynamically optimize data skewness

Dynamically coalesce Shuffle partitions

Sales

Sale_date	Product_ID	Quantity
10/01/2024	100	2
17/02/2024	102	4
10/01/2024	101	1
06/02/2024	100	9
05/04/2024	103	4
10/01/2024	100	7
12/04/2024	101	5
04/09/2023	102	2
17/02/2024	101	7

Author: Shanmukh Sattiraju

Dynamically coalesce Shuffle partitions

SELECT **sale_date**, SUM(sales_quantity)
AS Total_Quantity
FROM sales
GROUP BY **sale_date**

Sales

Sale_date	Product_ID	Quantity
10/01/2024	100	2
17/02/2024	102	4
10/01/2024	101	1
06/02/2024	100	5
05/04/2024	103	5
10/01/2024	100	7
17/02/2024	101	2
05/04/2024	102	2
17/02/2024	101	6
11/02/2024	103	7

Sale_date	Total_Quantity
10/01/2024	10
17/02/2024	12
06/02/2024	5
05/04/2024	7
04/09/2023	2
11/02/2024	7

Dynamically coalesce Shuffle partitions

Sales

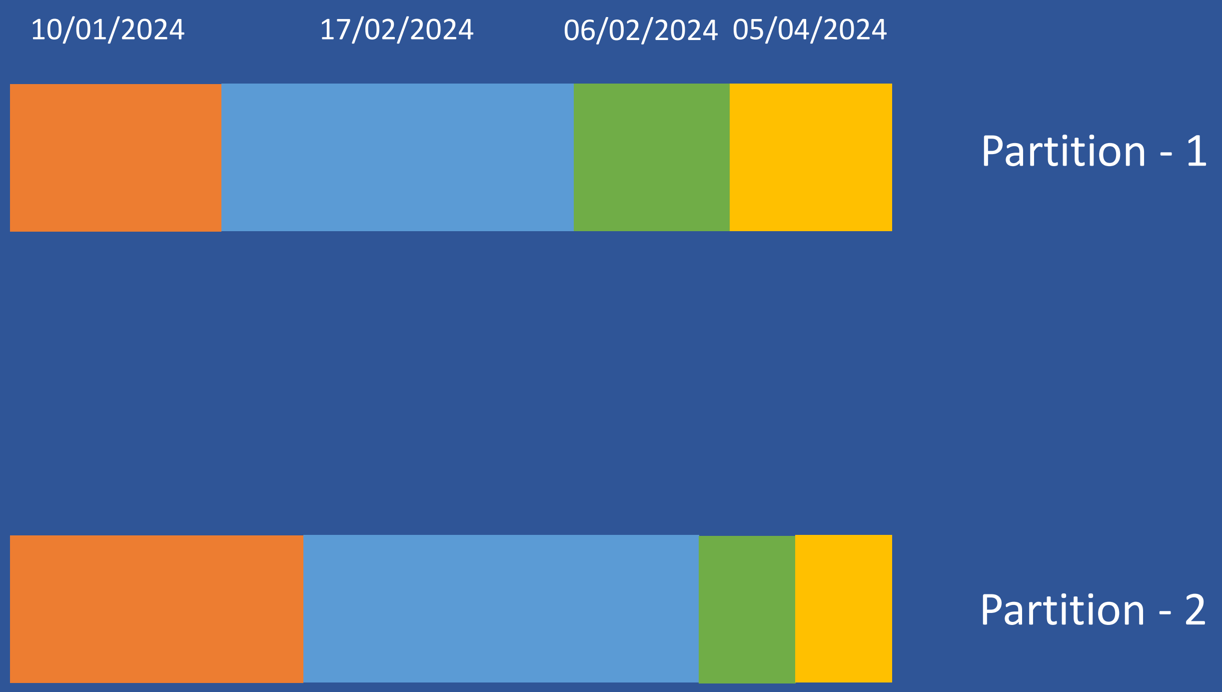
Sale_date	Product_ID	Quantity
10/01/2024	100	2
17/02/2024	102	4
10/01/2024	101	1
06/03/2024	100	5
05/04/2024	103	5
05/04/2024	103	5

Sale_date	Product_ID	Quantity
10/01/2024	100	7
17/02/2024	101	2
05/04/2024	102	2
13/02/2024	101	6
11/02/2024	103	7
11/02/2024	103	7

Dynamically coalesce Shuffle partitions

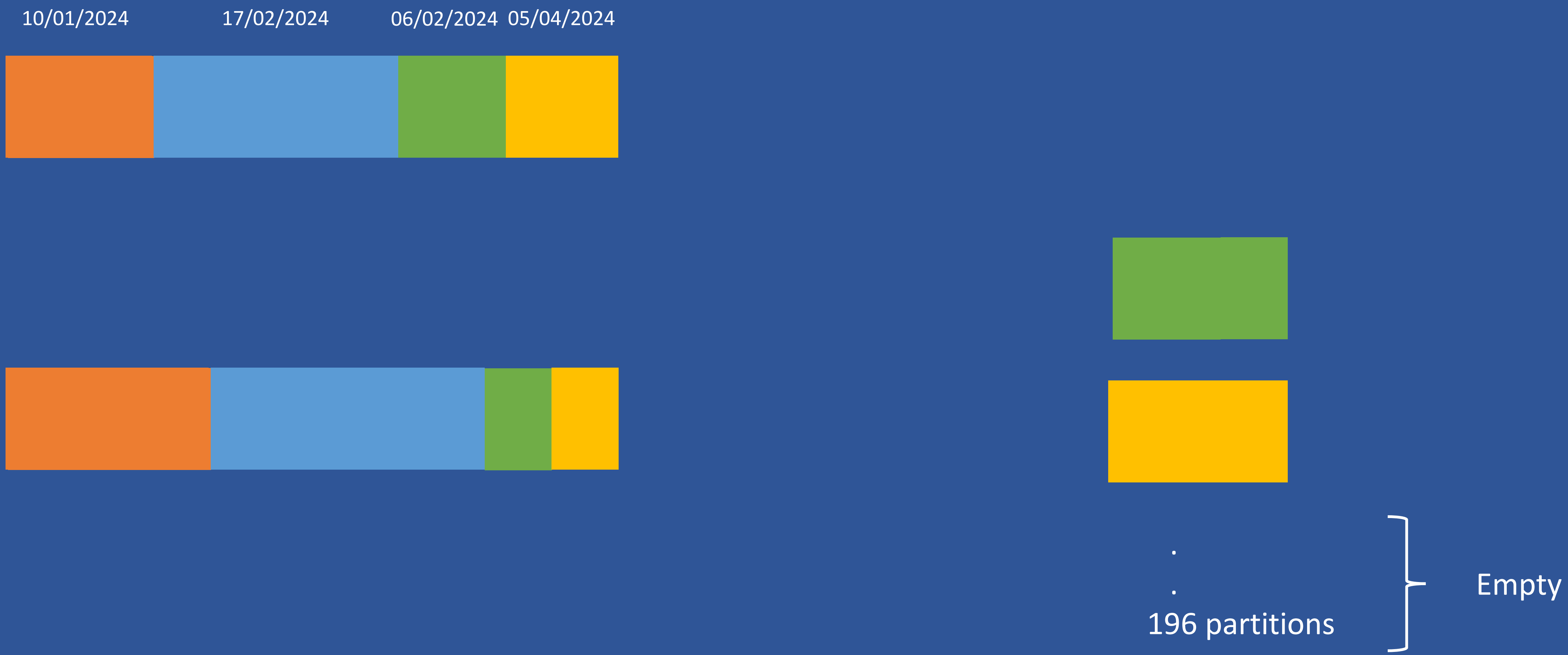
Sale_date	Product_ID	Quantity
10/01/2024	100	2
17/02/2024	102	4
10/01/2024	101	1
06/02/2024	100	5
05/04/2024	103	5

Sale_date	Product_ID	Quantity
10/01/2024	100	7
17/02/2024	101	2
05/04/2024	102	2
17/02/2024	101	6
11/02/2024	103	7



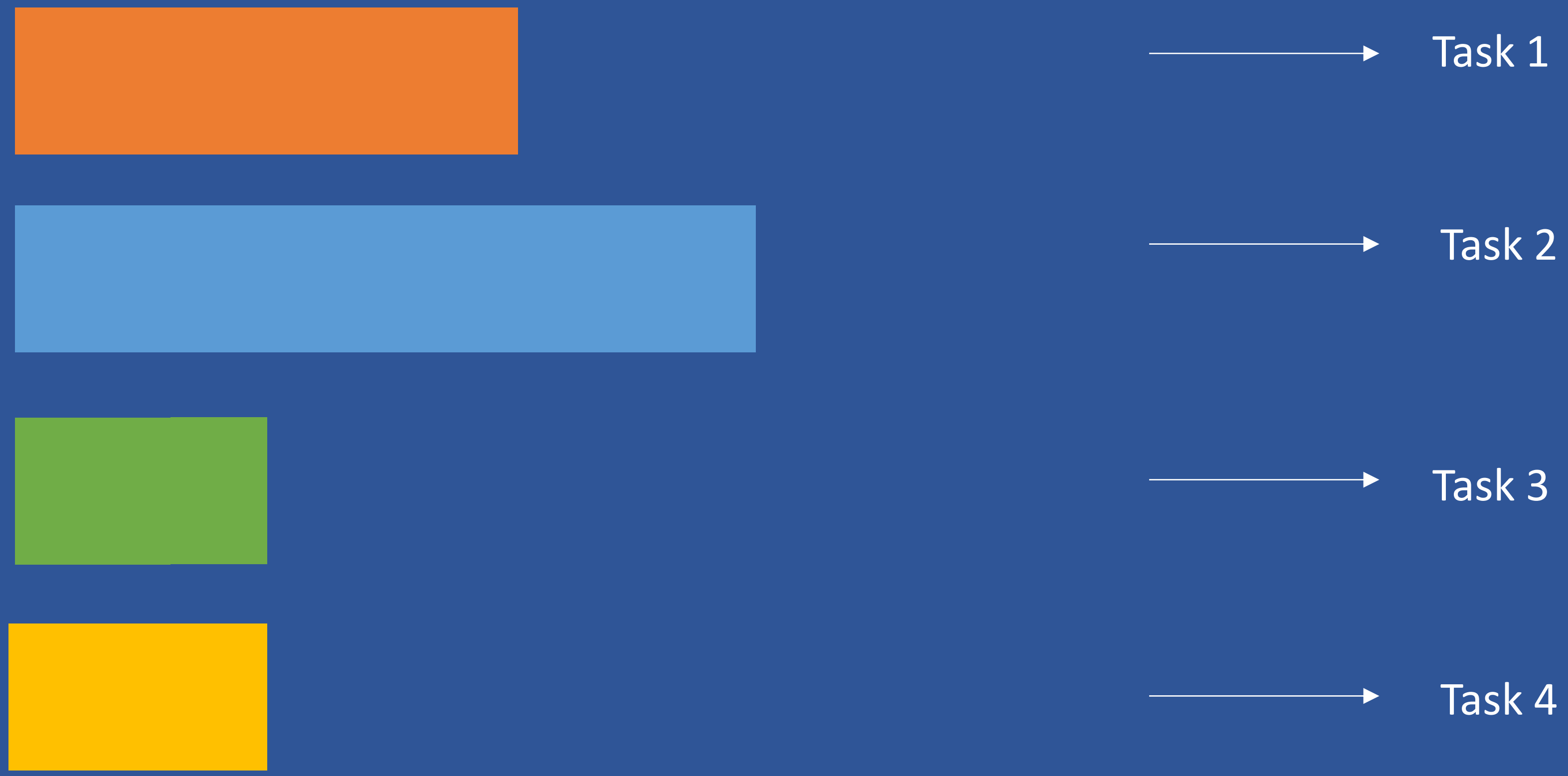
Author: Shanmukh Sattiraju

Dynamically coalesce shuffle partitions



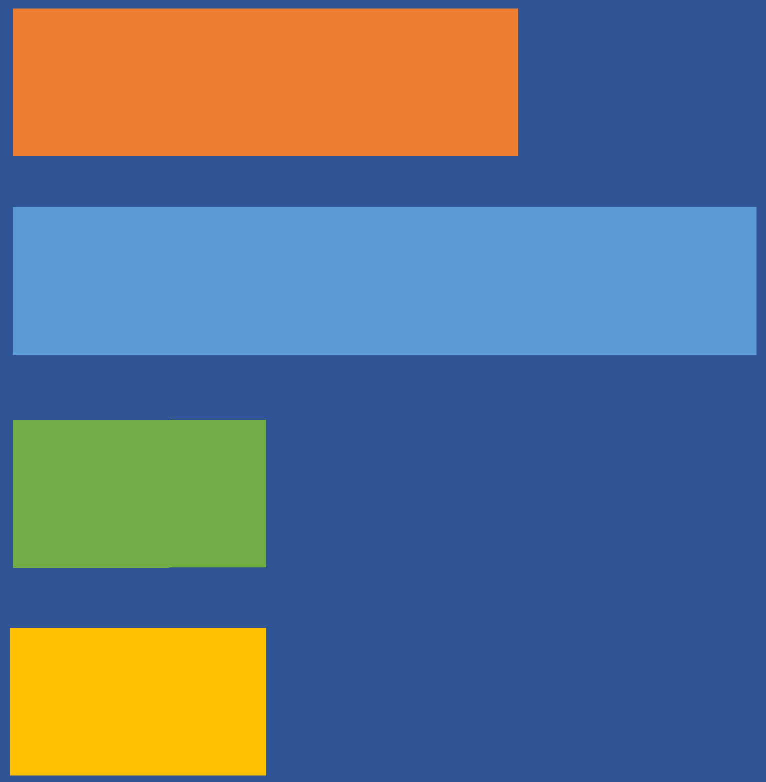
Author: Shanmukh Sattiraju

Dynamically coalesce shuffle partitions



Author: Shanmukh Sattiraju

Dynamically coalesce shuffle partitions



Author: Shanmukh Sattiraju

Dynamically switch join strategy

Products

Product_ID	Price
100	21.05
101	14.7
102	480
103	955
105	1000
106	599
107	749
108	477
109	522

Sales

Sale_date	Product_ID	Quantity
10/01/2024	100	2
17/02/2024	102	4
10/01/2024	101	1
06/02/2024	100	9
05/04/2024	103	4
10/01/2024	100	7
12/04/2024	101	5
04/09/2023	102	2
17/02/2024	101	7

Dynamically switch join strategy

```
SELECT sale_date, sum(sales_quantity * product_price) AS
total_sales
FROM sales S
JOIN products P ON S.product_id = P.product_id
WHERE product_price < 10
GROUP BY sale_date
```

Products

Product_ID	Price
100	21.05
101	14.7
102	480
103	955
105	1000
106	599
107	749
108	477
109	522

Sales

Sale_date	Product_ID	Quantity
10/01/2024	100	2
17/02/2024	102	4
10/01/2024	101	1
06/02/2024	100	9
05/04/2024	103	4
10/01/2024	100	7
12/04/2024	101	5
04/09/2023	102	2
17/02/2024	101	7

Dynamically optimize skewed data

Products

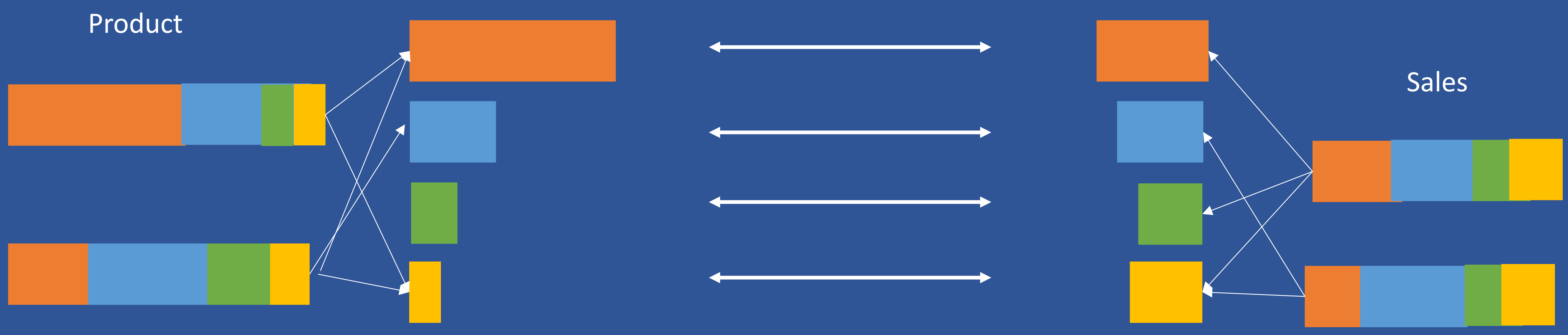
Product_ID	Price
100	21.05
101	14.7
102	480
103	955
105	1000
106	599
107	749
108	477
109	522

Sales

Sale_date	Product_ID	Quantity
10/01/2024	100	2
17/02/2024	102	4
10/01/2024	101	1
06/02/2024	100	9
05/04/2024	103	4
10/01/2024	100	7
12/04/2024	100	5
04/09/2023	100	2
17/02/2024	101	7

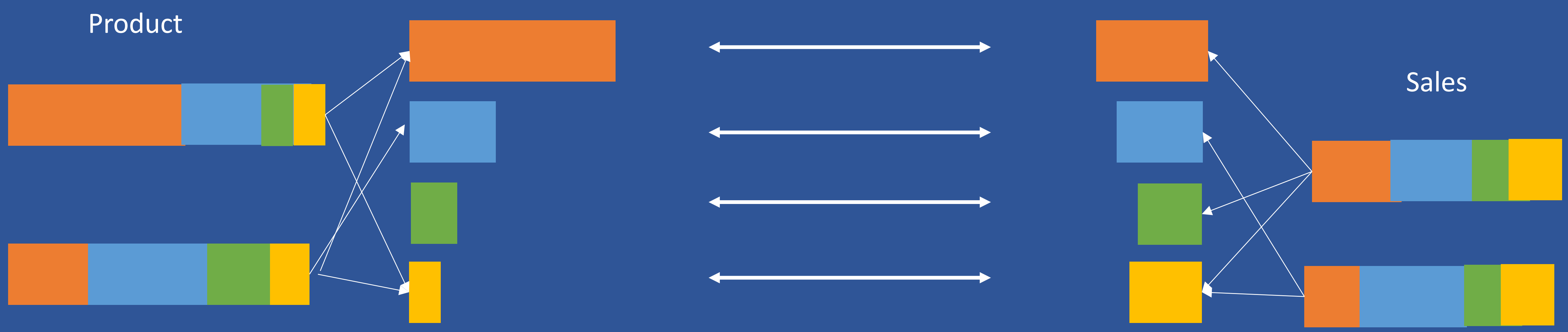
Author: Shanmukh Sattiraju

Dynamically optimize skewed data



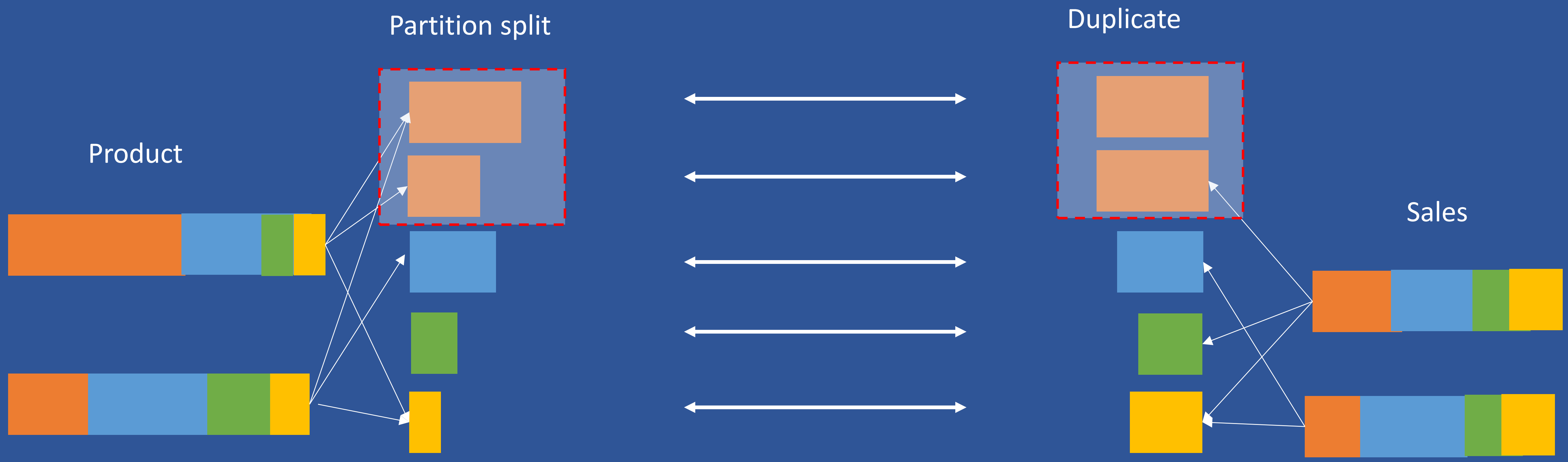
Author: Shanmukh Sattiraju

Dynamically optimize skewed data



Author: Shanmukh Sattiraju

Dynamically optimize skewed data

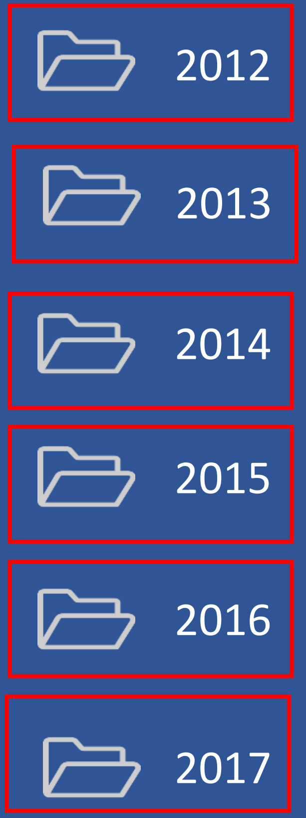


Author: Shanmukh Sattiraju

Partitioning vs Bucketing

Year	Month	Unemployed
2012	Jan	211741
2013	Jan	451751
2014	Jan	51652
2015	Jan	5174
2016	Jan	21657
2017	Jan	45868
2018	Jan	87474

Partition on Year column



Bucketing

Year	Month	Unemployed
2012	Jan	211741
2013	Jan	451751
2014	Jan	51652
2015	Jan	5174
2016	Jan	21657
2017	Jan	45868
2018	Jan	87474

BucketBy(4, "Year")

2012
2017

2013
2016

2014
2018

2015
2019

Bucketing

Traffic

Record_ID	Direction	hour	Region_id	Road_name	Road_Category_ID
28376	N	15	11	A1	3
80658	W	11	6	A1	4
28356	N	7	11	A1	3
28363	S	14	11	A1	3
456	S	7	7	A1	1
26557	S	16	8	A1	3
82139	N	15	7	A1	3

Roads

Road_category_id	Road_category	Region_name
1	TM	Scotland
2	PM	Wales
3	TA	Yorkshire
4	PA	Landon
5	m	Mexico

Bucketing

Traffic

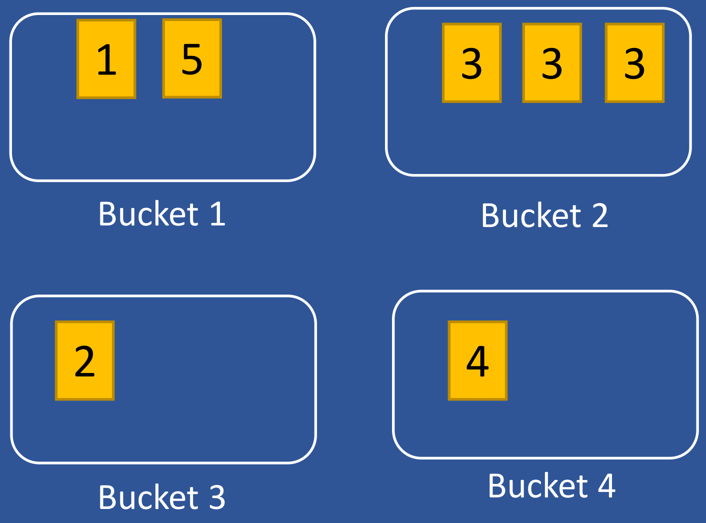
Record_ID	Direction	hour	Region_id	Road_name	Road_Category_ID
28376	N	15	11	A1	3
80658	W	11	6	A1	4
28356	N	7	11	A1	2
28363	S	14	11	A1	3
456	S	7	7	A1	1
26557	S	16	8	A1	3
82139	N	15	7	A1	5

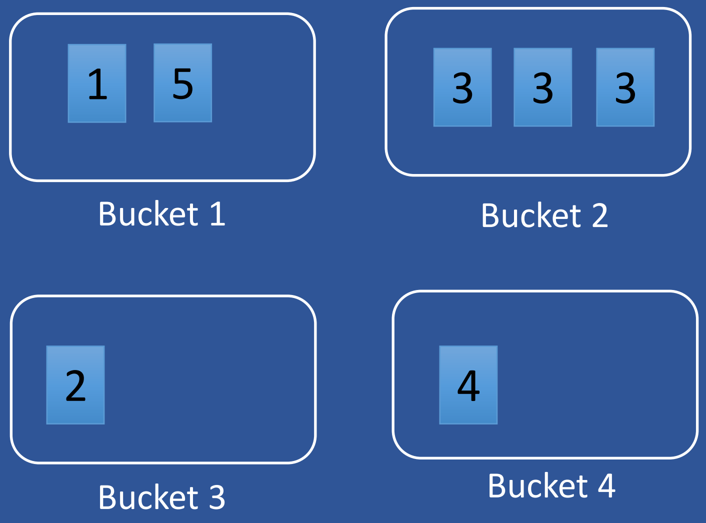
bucketBy(4,’ Road_Category_ID’)

Roads

Road_category_id	Road_category	Region_name
1	TM	Scotland
2	PM	Wales
3	TA	Yorkshire
4	PA	Landon
5	m	Mexico

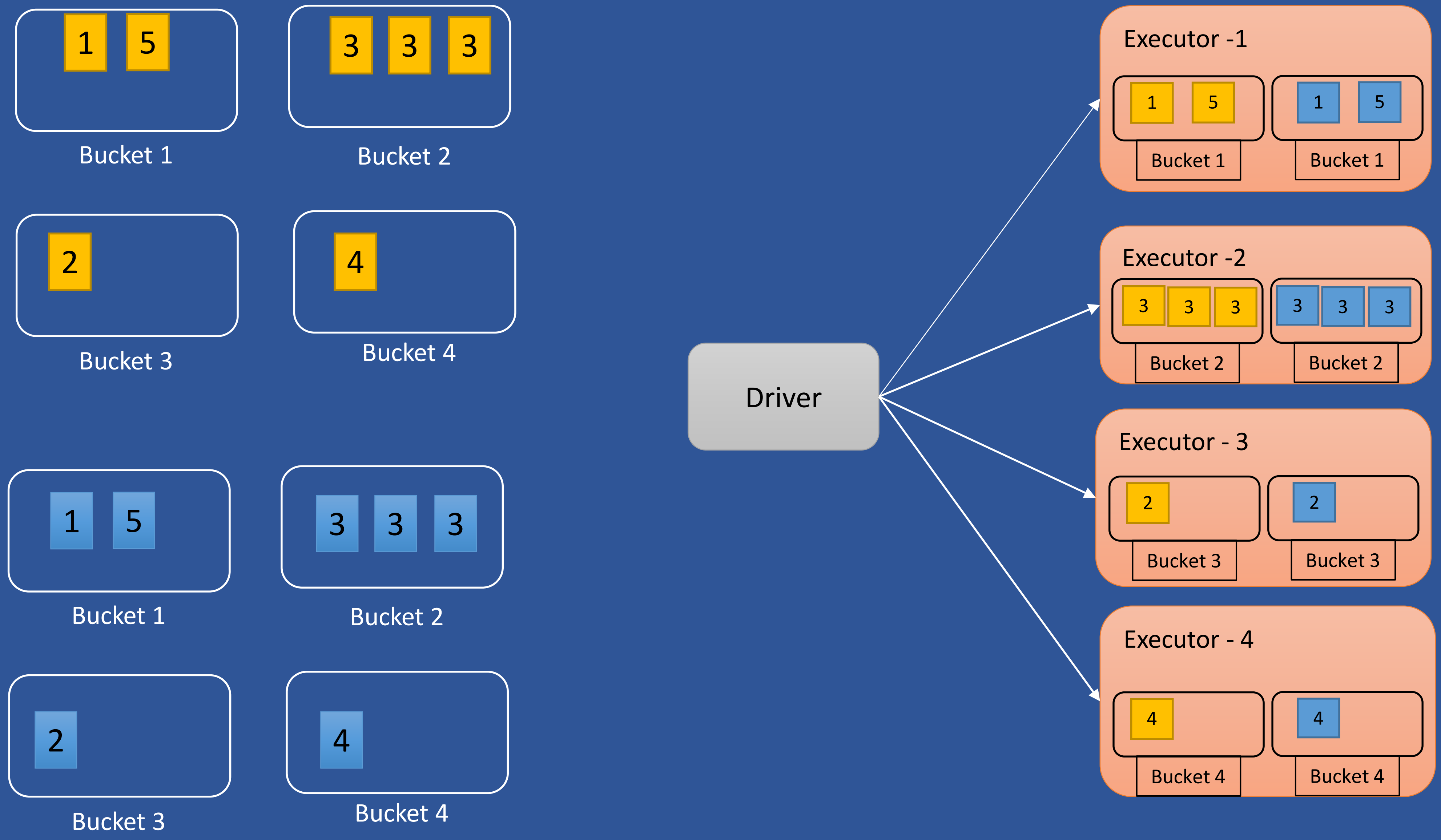
bucketBy(4,’ Road_Category_ID’)





Author: Shanmukh Sattiraju

Bucketing



Author: Shanmukh Sattiraju

Salting

Traffic

Record_ID	Direction	hour	Region_id	Road_name	Road_Category_ID
28376	N	15	11	A1	3
80658	W	11	6	A1	4
32215	N	7	11	A1	3
14415	S	14	11	A1	3
456	S	7	7	A1	1
82139	N	15	7	A1	3

Roads

Road_category_id	Road_category	Region_name
3	TA	Scotland
3	TA	Wales
2	PM	Yorkshire
2	PM	Landon
3	TA	Mexico
1	AM	US

Record_ID	Direction	hour	Region_id	Road_name	Road_Category_ID	Road_category_id	Road_category	Region_name
28376	N	15	11	A1	3	3	TA	Scotland
28376	N	15	11	A1	3	3	TA	Wales
28376	N	15	11	A1	3	3	TA	Mexico
32215	N	7	11	A1	3	3	TA	Scotland
32215	N	7	11	A1	3	3	TA	Wales
32215	N	7	11	A1	3	3	TA	Mexico
14415	S	14	11	A1	3	3	TA	Scotland
14415	S	14	11	A1	3	3	TA	Wales
14415	S	14	11	A1	3	3	TA	Mexico
82139	N	15	7	A1	3	3	TA	Scotland
82139	N	15	7	A1	3	3	TA	Wales
82139	N	15	7	A1	3	3	TA	Mexico
456	S	7	7	A1	1	1	AM	US

Salting

Traffic

Record_ID	Direction	hour	Region_id	Road_name	Road_Category_D	Salt_ID
28376	N	15	11	A1	3	3_0
80658	W	11	6	A1	4	4_1
28356	N	7	11	A1	3	3_1
28363	S	14	11	A1	3	3_2
456	S	7	7	A1	1	1_2
26557	S	16	8	A1	3	3_1
82139	N	15	7	A1	3	3_1

Roads

Salt_ID	Road_category_id	Road_category	Region_name
3_0	3	TA	Scotland
3_1	3	TA	Wales
2_0	2	TA	Yorkshire
2_1	2	TA	Landon
3_1	3	TA	Mexico

Salt_ID	Road_category_id	Road_category	Region_name
3_0	3	TA	Scotland
3_1	3	TA	Scotland
3_2	3	TA	Scotland
3_0	3	TA	Wales
3_1	3	TA	Wales
3_2	3	TA	Wales

Author: Shanmukh Sattiraju

Salting

Traffic

Record_ID	Direction	hour	Region_id	Road_name	Road_Category_ID	Salt_ID
28376	N	15	11	A1	3	3_0
80658	W	11	6	A1	4	4_1
28356	N	7	11	A1	3	3_1
28363	S	14	11	A1	3	3_2
456	S	7	7	A1	1	1_2
26557	S	16	8	A1	3	3_1
82139	N	15	7	A1	3	3_1

Roads

Salt_ID	Road_category_id	Road_category	Region_name
3_0	3	TA	Scotland
3_1	3	TA	Scotland
3_2	3	TA	Scotland
3_0	3	TA	Wales
3_1	3	TA	Wales
3_2	3	TA	Wales
3_0	3	TA	Mexico
3_1	3	TA	Mexico
3_2	3	TA	Mexico

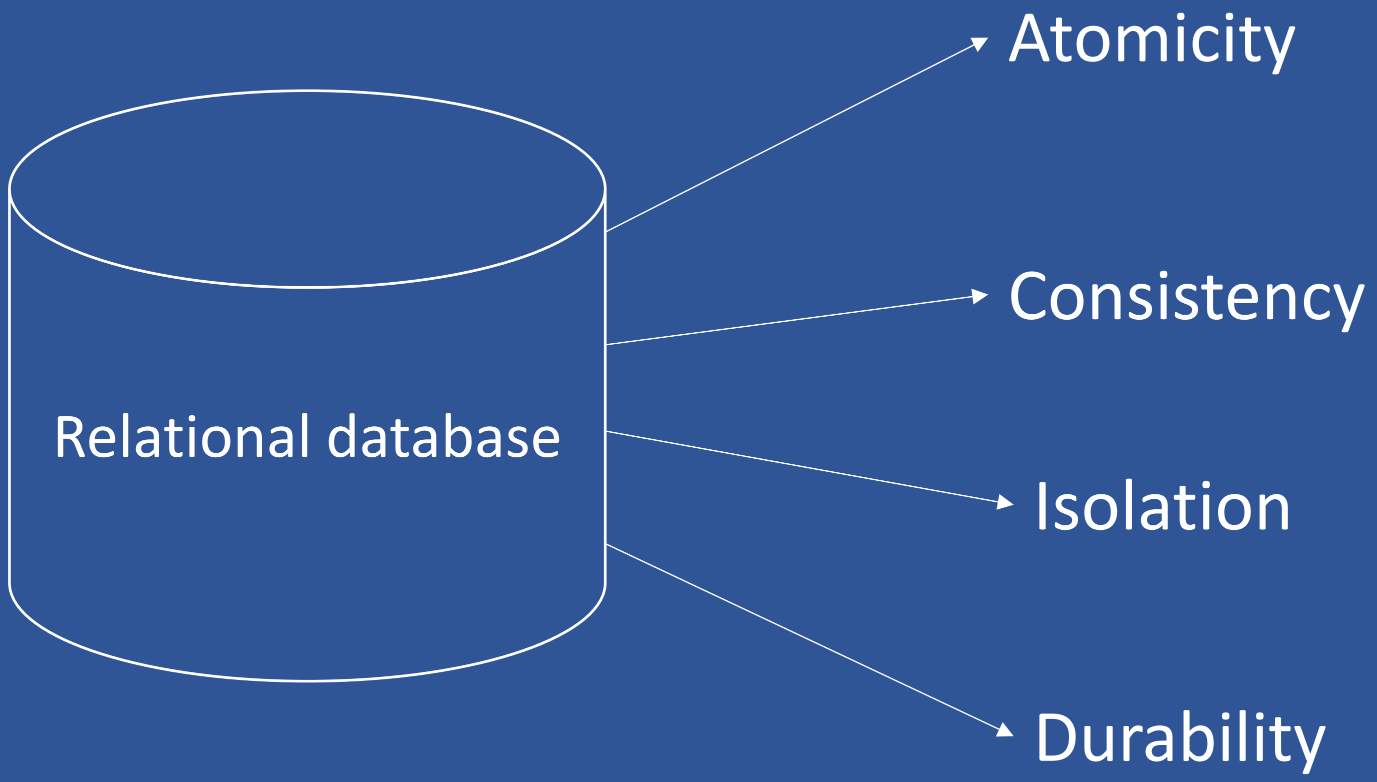
Record_ID	Direction	hour	Region_id	Road_name	Road_Category_ID	Salt_ID	Salt_ID	Road_category_id	Road_category	Region_name
28376	N	15	11	A1	3	3_0	3_0	3	TA	Scotland
28376	N	15	11	A1	3	3_0	3_0	3	TA	Wales
28376	N	15	11	A1	3	3_0	3_0	3	TA	Mexico
28356	N	7	11	A1	3	3_1	3_1	3	TA	Scotland
28356	N	7	11	A1	3	3_1	3_1	3	TA	Wales
28356	N	7	11	A1	3	3_1	3_1	3	TA	Mexico

Delta Lake

Author: Shanmukh Sattiraju

Drawbacks of ADLS

ADLS != Database



Drawbacks of ADLS

- No ACID properties
- Job failures lead to inconsistent data
- Simultaneous writes on same folder brings incorrect results
- No schema enforcement
- No support for updates
- No support for versioning

What is delta lake

- Open-source storage framework that brings reliability to data lakes
- Brings transaction capabilities to data lakes
- Runs on top of your existing datalake and supports parquet
- Enables Lakehouse architecture

Lakehouse Architecture

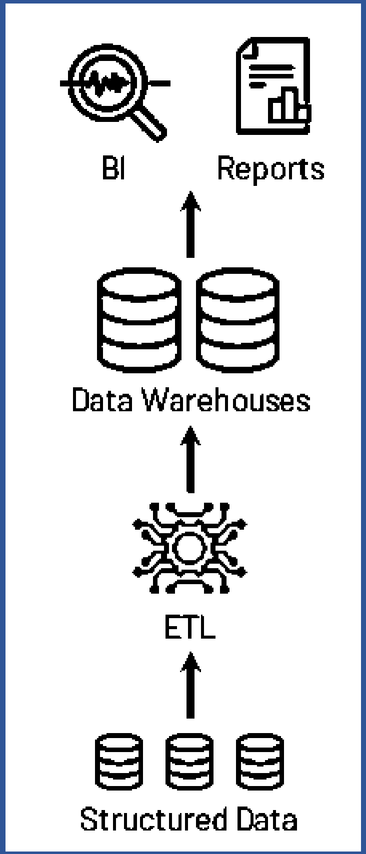
Best elements of
Data lake

Best elements of
Data warehouse

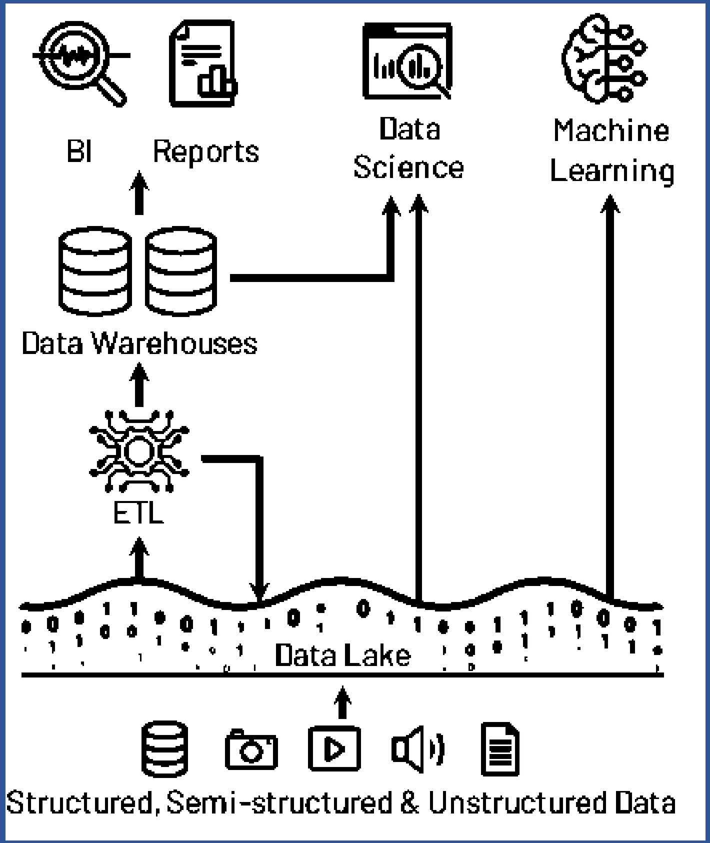
Lakehouse

Author: Shanmukh Sattiraju

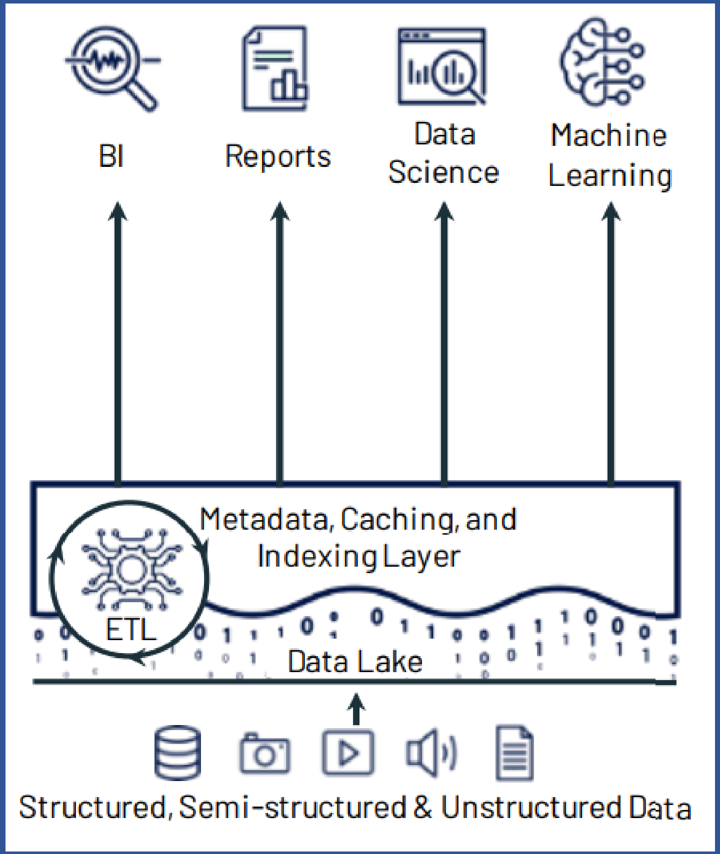
Lakehouse Architecture



Datawarehouse

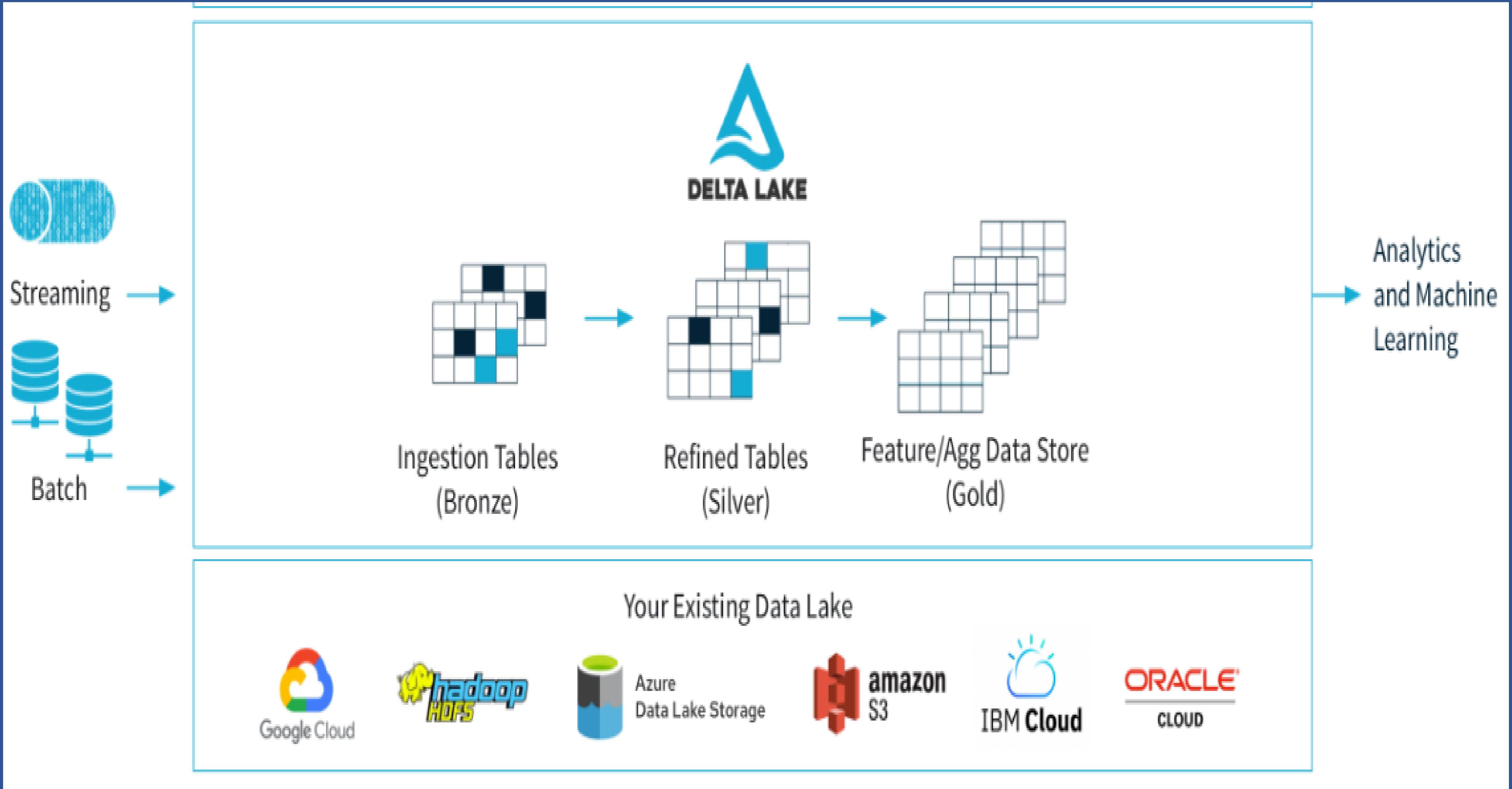


Modern Datawarehouse (uses Datalake)



Lakehouse Architecture

Author: Shanmukh Sattiraju



DELTA LAKE
Streaming
Batch
Ingestion Tables
(Bronze)
Refined Tables
(Silver)
Feature/Agg Data Store
(Gold)
Analytics
and Machine
Learning
Your Existing Data Lake
Google Cloud
hadoop
HDFS
Azure
Data Lake Storage
amazon
S3
IBM Cloud
ORACLE
CLOUD

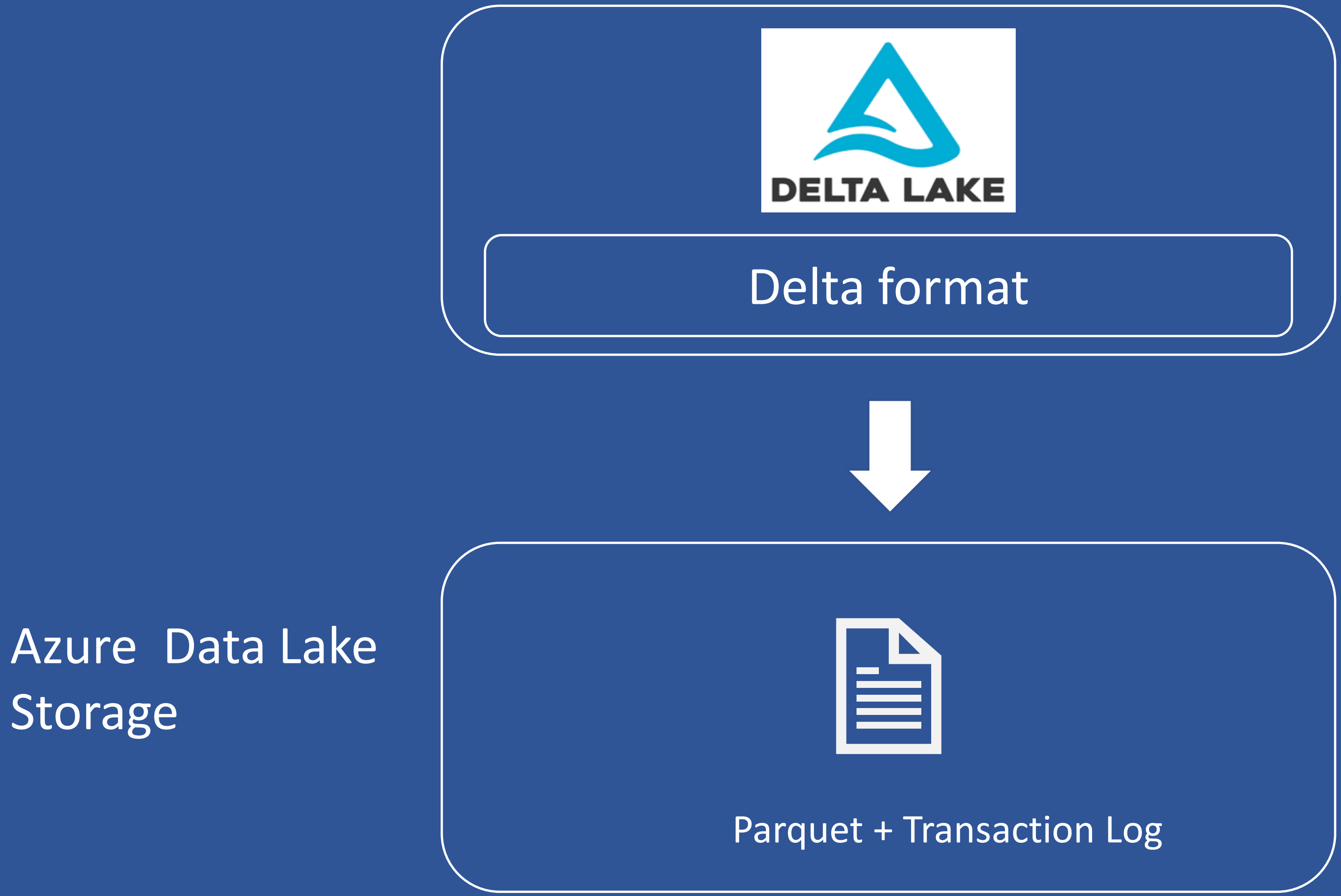
How to create delta lake?

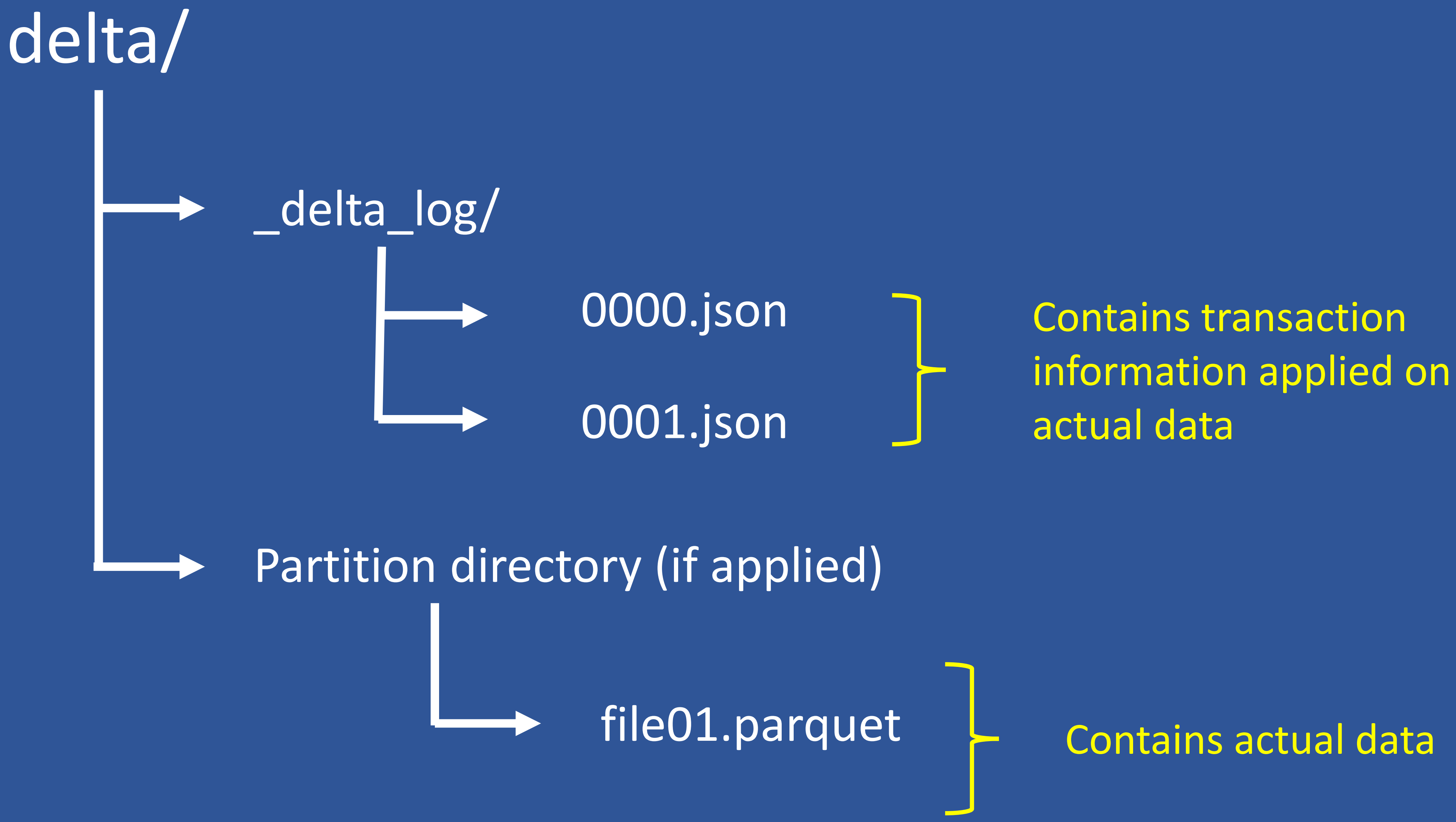
Instead of parquet..

```
dataframe.
        write\
        .format("parquet")\
        .save("/data/")
```

Replace with delta..

```
dataframe.
        write\
        .format("delta")\
        .save("/data/")
```


DELTA LAKE
Delta format
Azure Data Lake Storage
Parquet + Transaction Log



Author: Shanmukh Sattiraju

Understanding Transaction log file (Delta Log)

- Contains records of every transaction performed on the delta table
- Files under _delta_log will be stored in JSON format
- Single source of truth

Transaction log contents

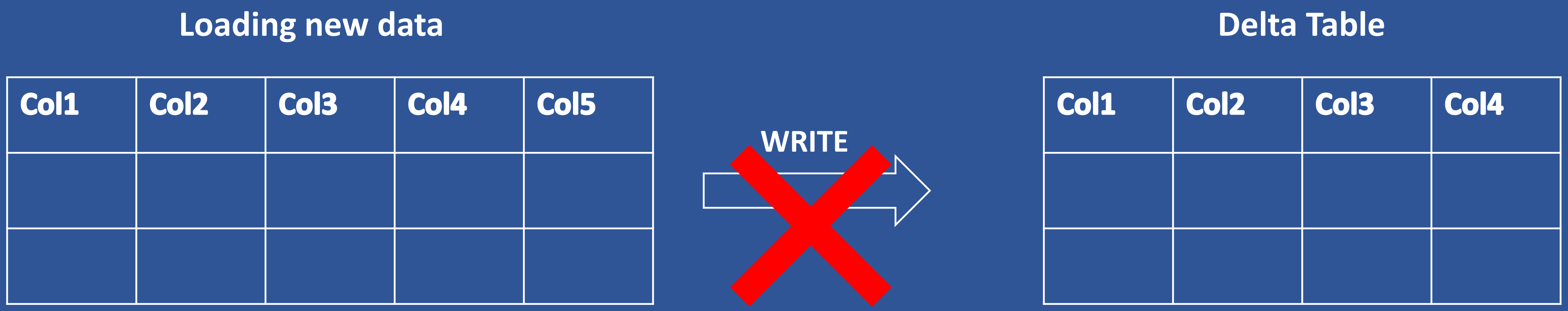
JSON File = result of set of actions

- **metadata** – Table's name, schema, partitioning ,etc
- **Add –** info of added file (with optional statistics)
- **Remove –** info of removed file
- **Set Transaction –** contains record of transaction id
- **Change protocol –** Contains the version that is used
- **Commit info –** Contains what operation was performed on this

Delta lake key features

- **Open Source**: Stored in form of **parquet** files in ADLS
- **ACID Transactions:** Ensures data quality
- **Schema Enforcement :** Restricts unexpected schema changes
- **Schema Evolution:** Accepts any required schema changes.
- **Audit History:** Logs all the change details happened on table
- **Time Travel:** Helps to get previous versions using version or timestamp
- **DML Operations:** Enables us to use UPDATE, DELETE and MERGE
- **Unified batch /Streaming:** Follows same approach for batch and streaming flows

Schema Enforcement



Author: Shanmukh Sattiraju

How does schema enforcement works?

Delta lake uses Schema validation on **“writes”** .

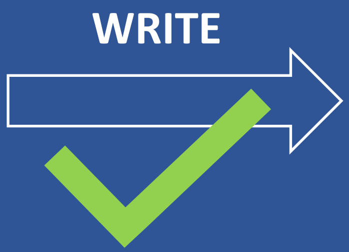
Schema Enforcement Rules:

1. Cannot contain any additional columns that are not present in the target table's schema
2. Cannot have column data types that differ from the column data types in the target table.

Schema Evolution

Loading new data

Col1	Col2	Col3	Col4	Col5



Delta Table

Col1	Col2	Col3	Col4

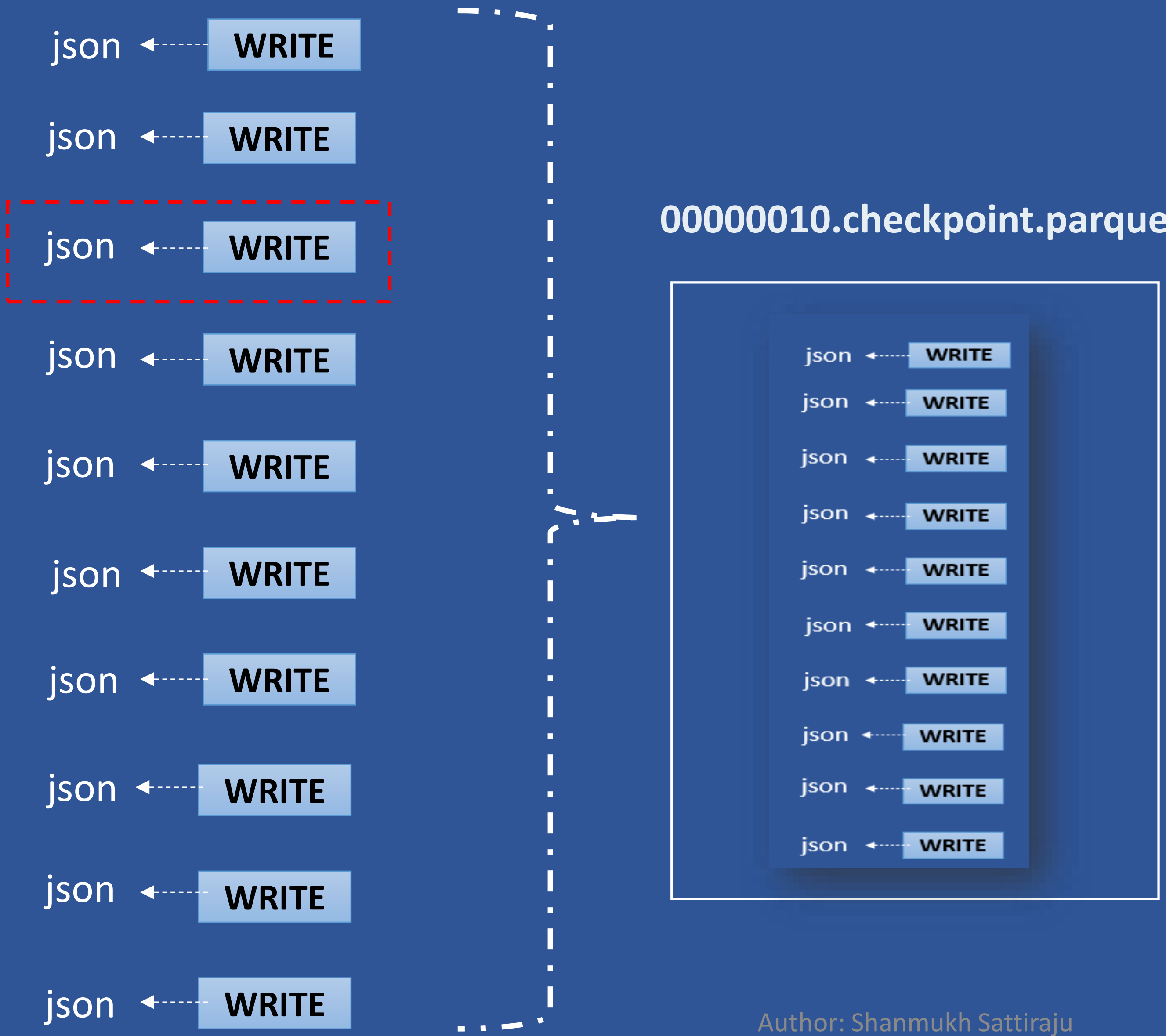
Audit Data Changes & Time Travel

- Delta automatically versions every operation that you perform
- You can time travel to historical versions
- This versioning makes it easy to audit data changes, roll back data in case of accidental bad writes or deletes, and reproduce experiments and reports.

Vacuum in Delta lake

- Vacuum helps to remove parquet files which are not in latest state in transaction log
- It will skip the files that are starting with _ (underscore) that includes _delta_log
- It deletes the files that are older then retention threshold
- Default retention threshold in 7 days
- If you run VACUUM on a Delta table, you lose the ability to time travel back to a version older than the specified data retention period.

Checkpoints in Delta lake



- Serves as starting point of compute
- Contains replay of all actions performed
- Reduce reading of JSON files
- By default, checkpoint is created for every 10 commits

Author: Shanmukh Sattiraju

Optimize in Delta lake

Operation	parquet files	_delta_log	Line number Column	State
CREATE TABLE		000.json		
WRITE	aabb.parquet	001.json	100	**Active**
WRITE	ccdd.parquet	002.json	101	Inactive
WRITE	eeff.parquet	003.json	102	Inactive
DELETE 101	gghh.parquet (empty)	004.json		Inactive
UPDATE 102	iijj.parquet	005.json	99	**Active**

Author: Shanmukh Sattiraju

UPSERT (Merge) in delta lake

- We can UPSERT (UPDATE + INSERT) data using MERGE command.
- If any matching rows found, it will update them
- If no matching rows found, this will insert that as new row

```
MERGE INTO <Destination_Table>
USING <Source_Table>
        ON <Dest>.Col2 = <Source>.Col2
WHEN MATCHED
THEN UPDATE SET
        <Dest>.Col1 = <Source>.Col1,
        <Dest>.Col2 = <Source>.Col2
WHEN NOT MATCHED
THEN INSERT
        VALUES(Source.Col1, Source.Col2)
```

Author: Shanmukh Sattiraju

End of the course