Hadoop Fundamentals

Hadoop

- Is an open source framework for storing, and processing extremely large data in a distributed computing environment.
- Not a database

Hadoop Components

- Distributed Computing
    - HDFS
- Parallel Processing
    - MapReduce

Advantages of Hadoop

- Low cost
  - Hadoop can run on average performing commodity
  - Help in controlling cost and achieve scalability and performance.
  - Adding or removing nodes from the cluster is simple
- Storage flexibility
  - Hadoop can store data in raw format in a distributed environment.
  - Hadoop can process the unstructured data and semi-structured data better than most of the available technologies.

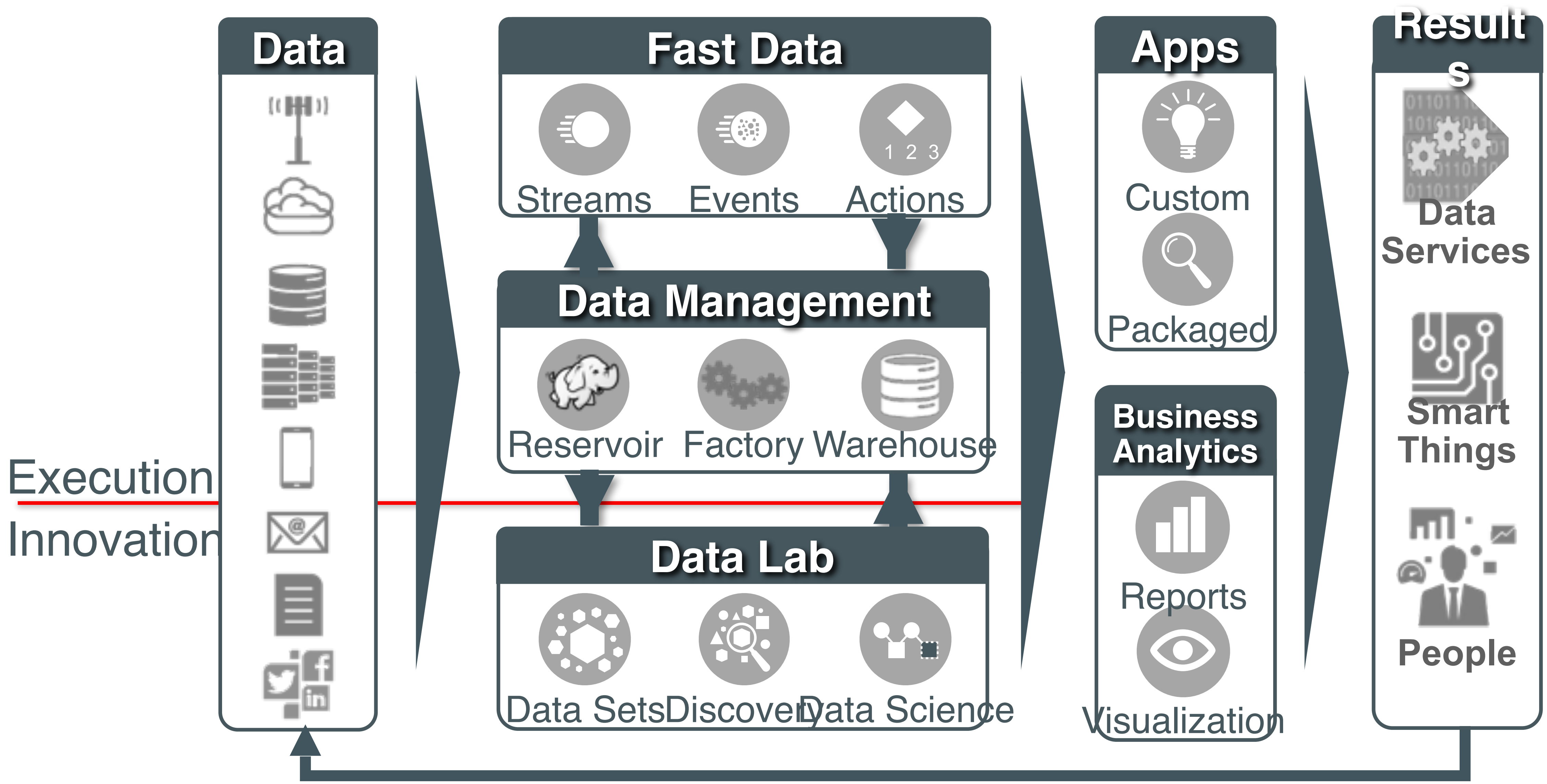
Advantages of Hadoop

- Open source community
  - Hadoop is open source and supported by many contributors with a growing network of developers worldwide.
  - Many organizations such as Yahoo, Facebook, Hortonworks, and others have contributed immensely toward the progress of Hadoop and other related sub-projects.
- Fault tolerant
  - Hadoop is massively scalable and fault tolerant.
  - Hadoop is reliable in terms of data availability, and even if some nodes go down, Hadoop can recover the data.
  - Hadoop architecture assumes that nodes can go down and the system should be able to process the data.

Use Cases for Hadoop

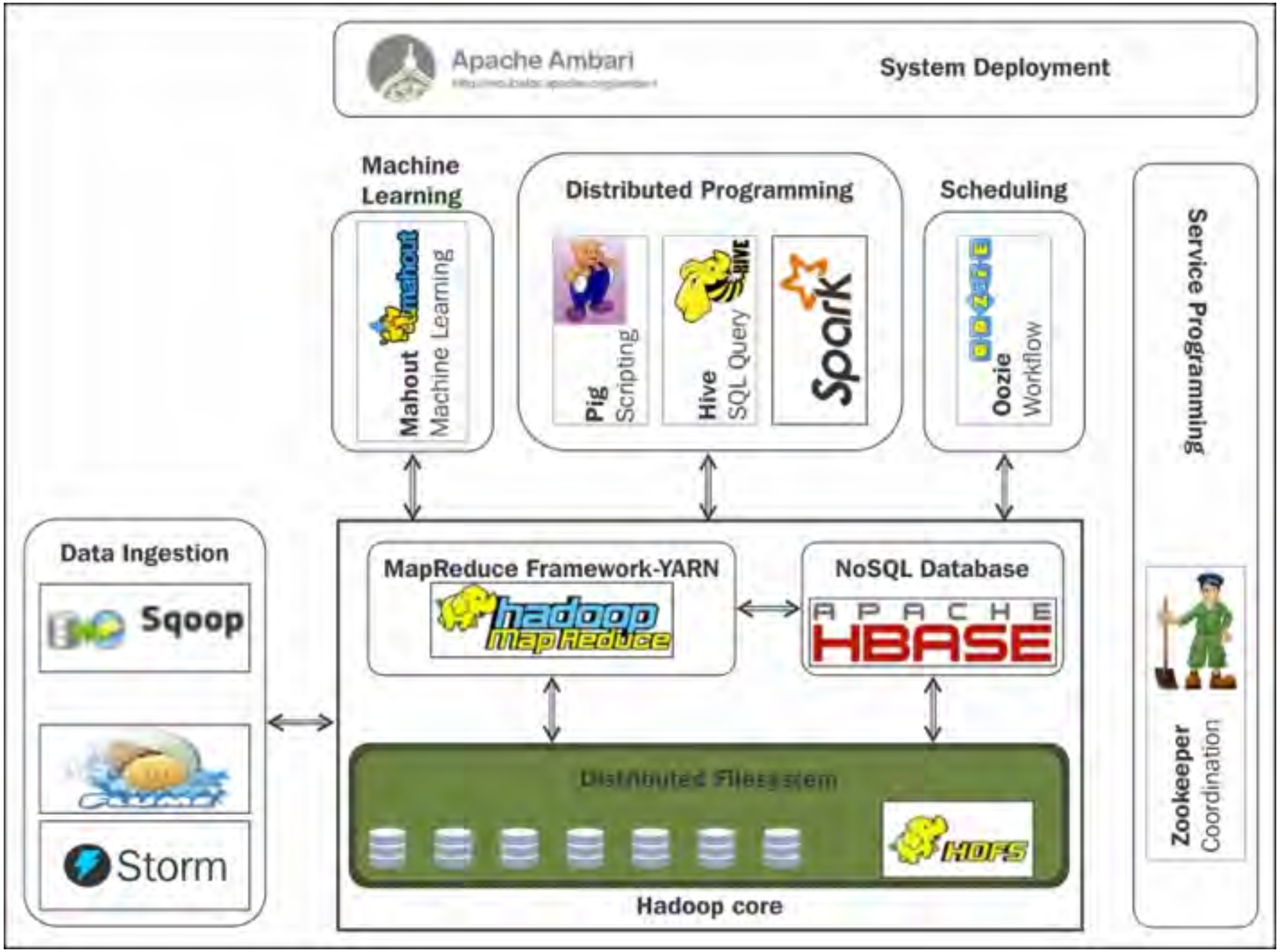
- Searching/text mining
- Log processing
- Recommendation systems
- Business intelligence/data warehousing
- Video and image analysis
- Archiving
- Graph creation and analysis
- Pattern recognition
- Risk assessment
- Sentiment analysis

Big Data Flow

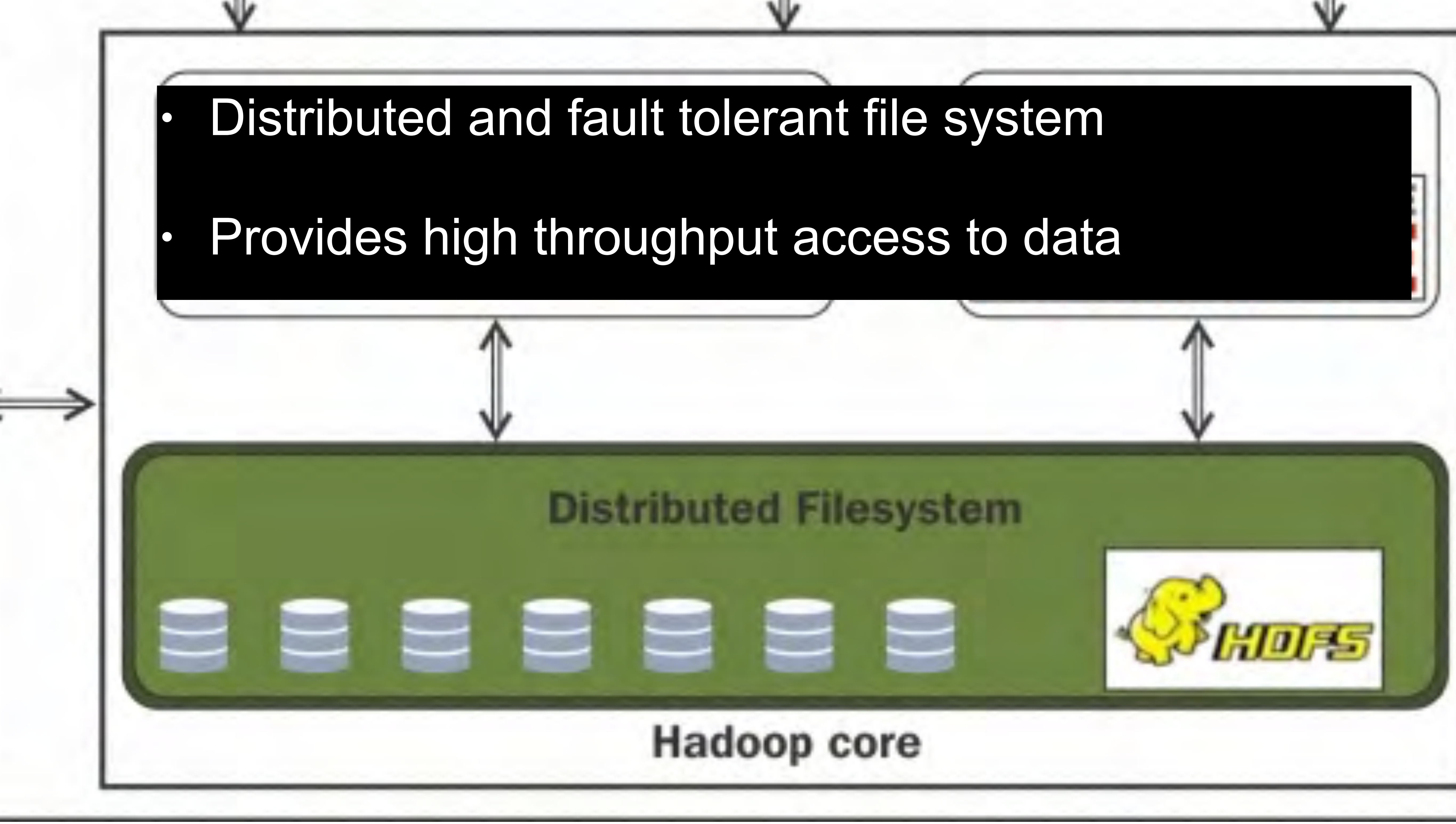


Sohail Rafiqi

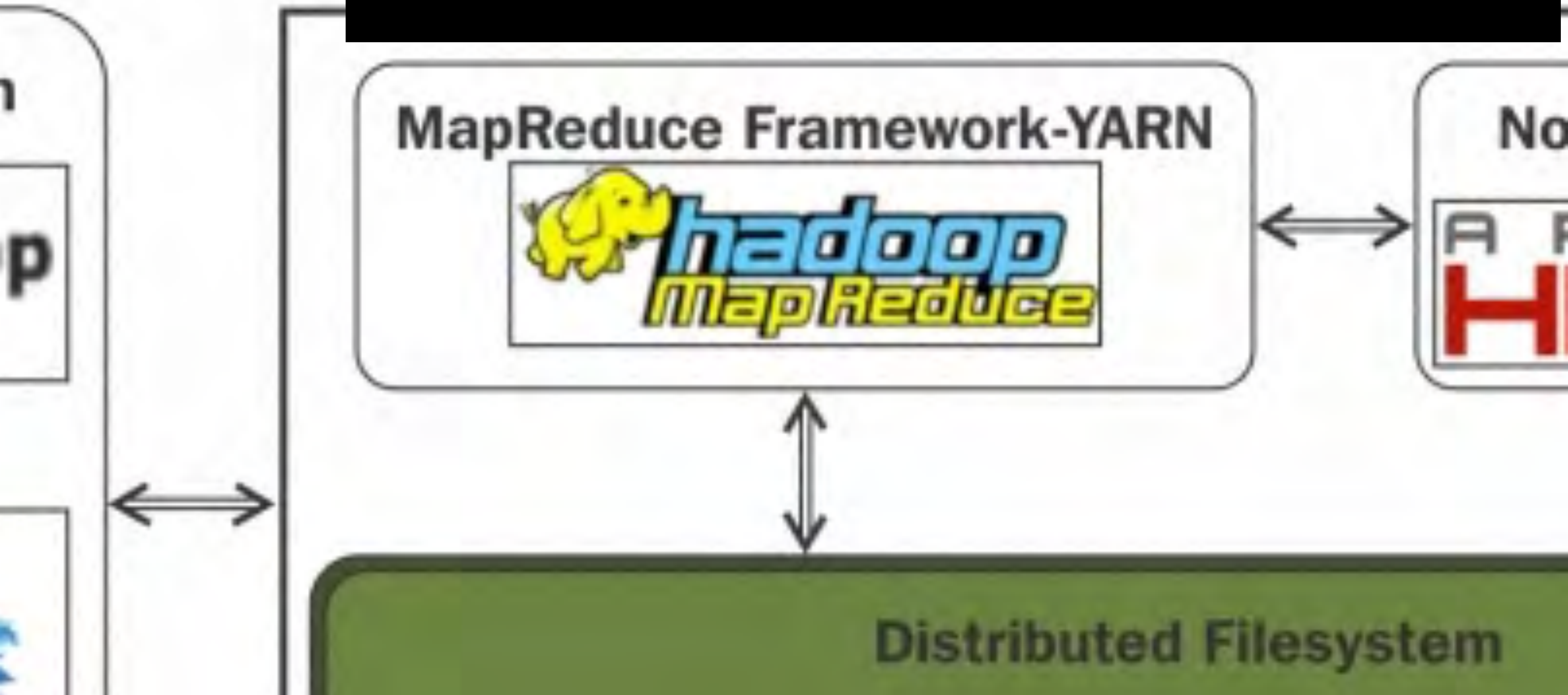
Hadoop Ecosystem

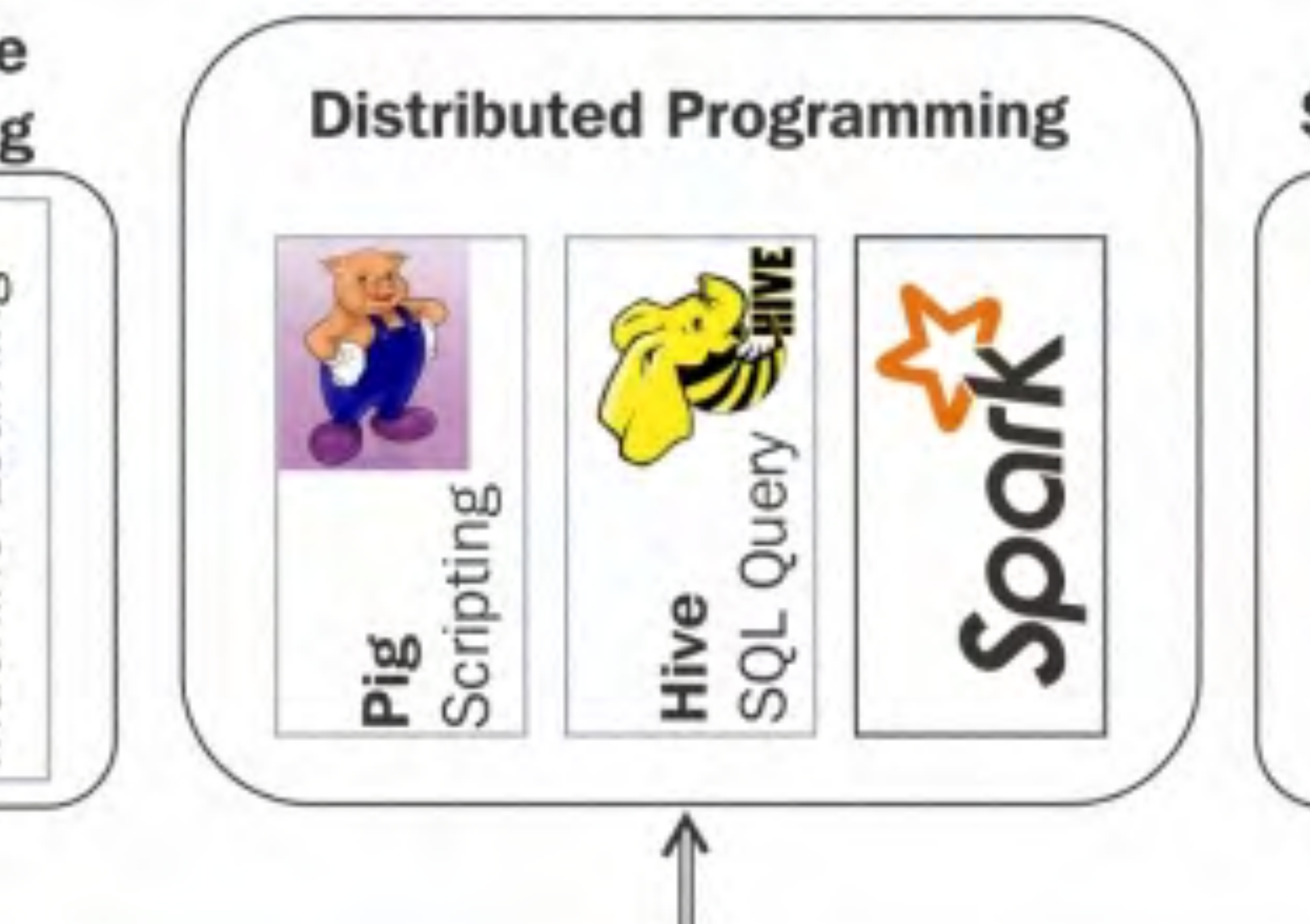


- Distributed and fault tolerant file system
- Provides high throughput access to data

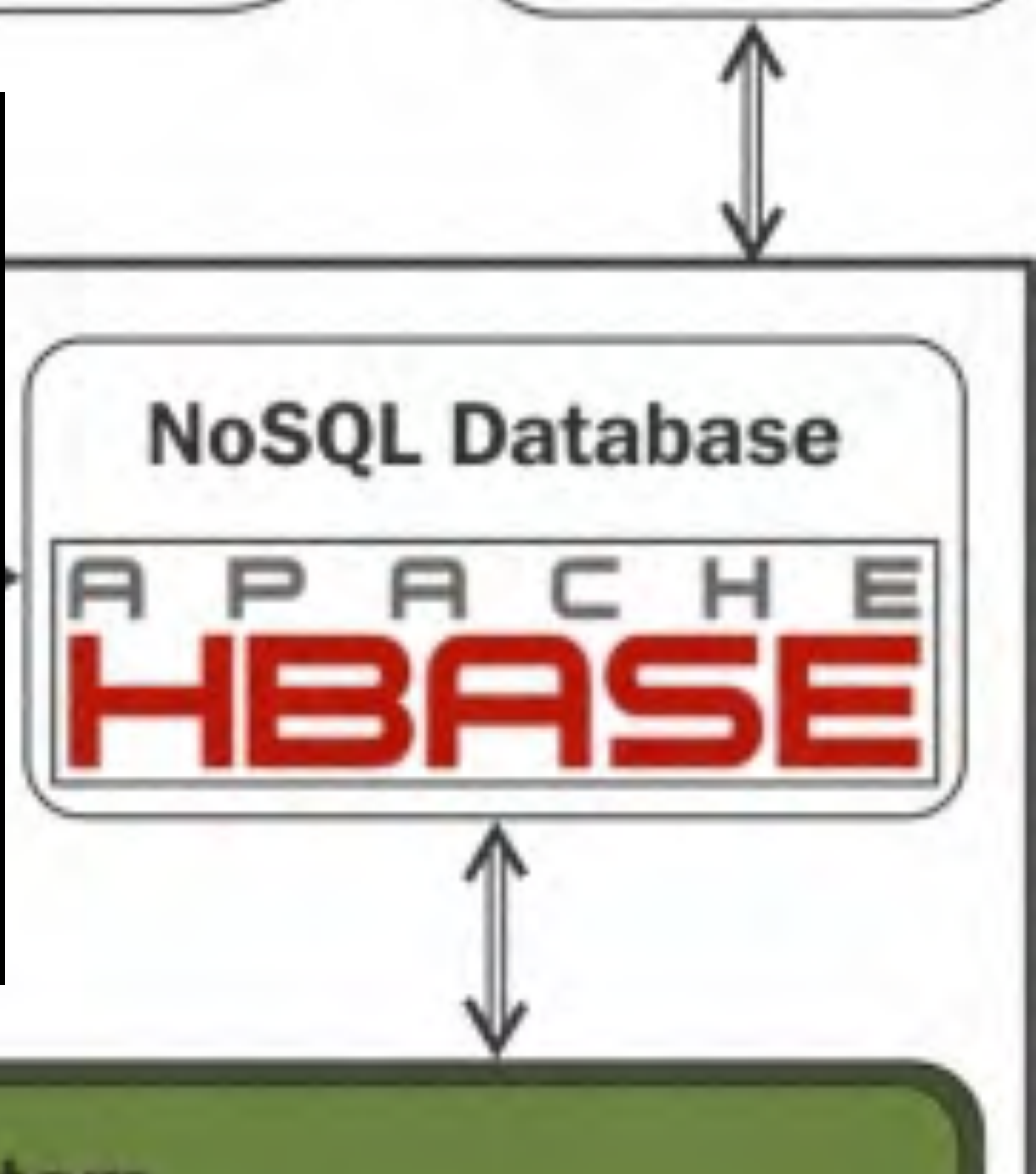


- Parallel Processing Framework
- Processing is moved to the data




Distributed Programming
Pig
Scripting
HIVE
Hive
SQL Query
Spark

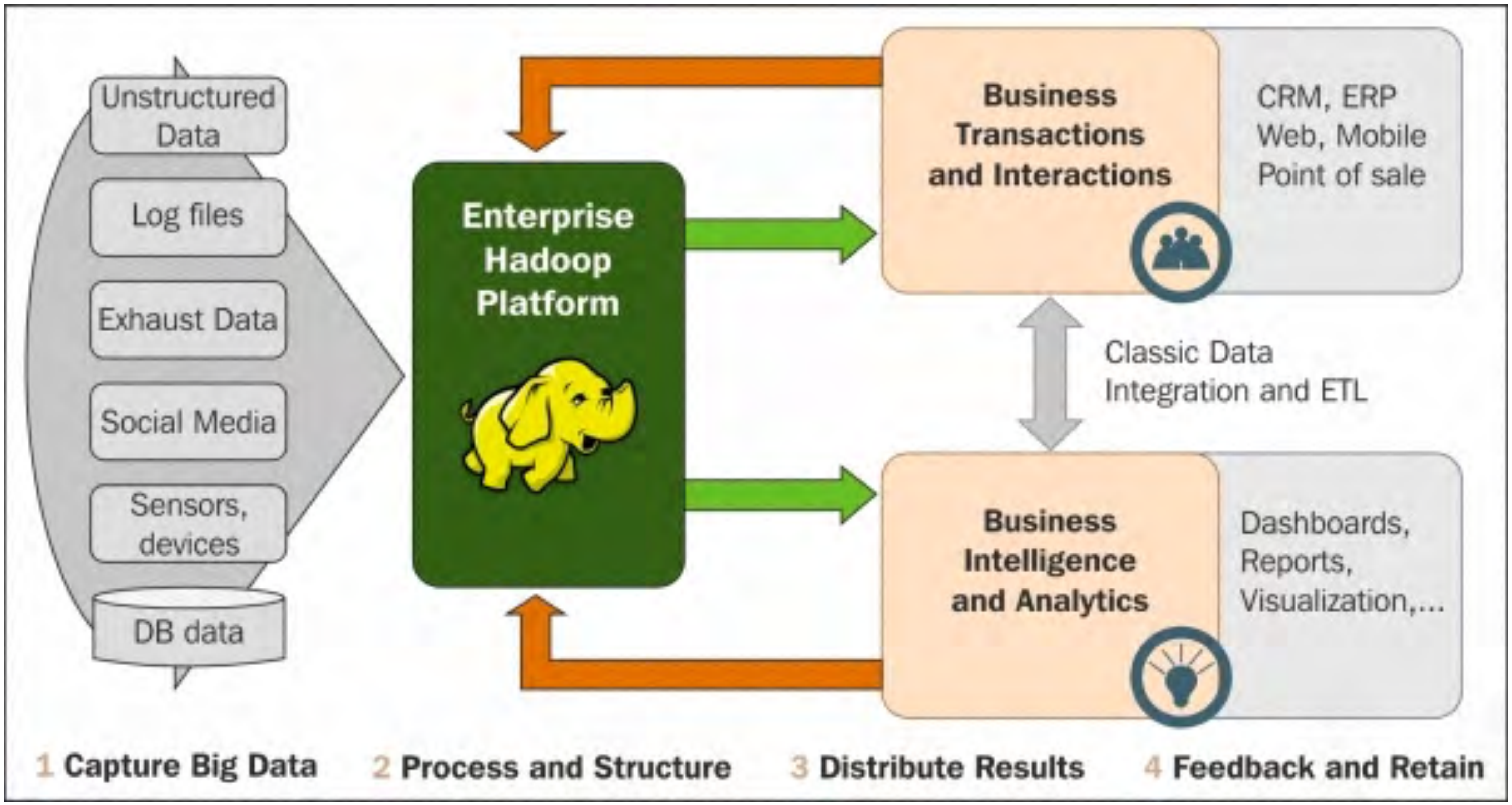
- NoSQL
- Column oriented DB and a key-Value store
- Works on top of HDFS

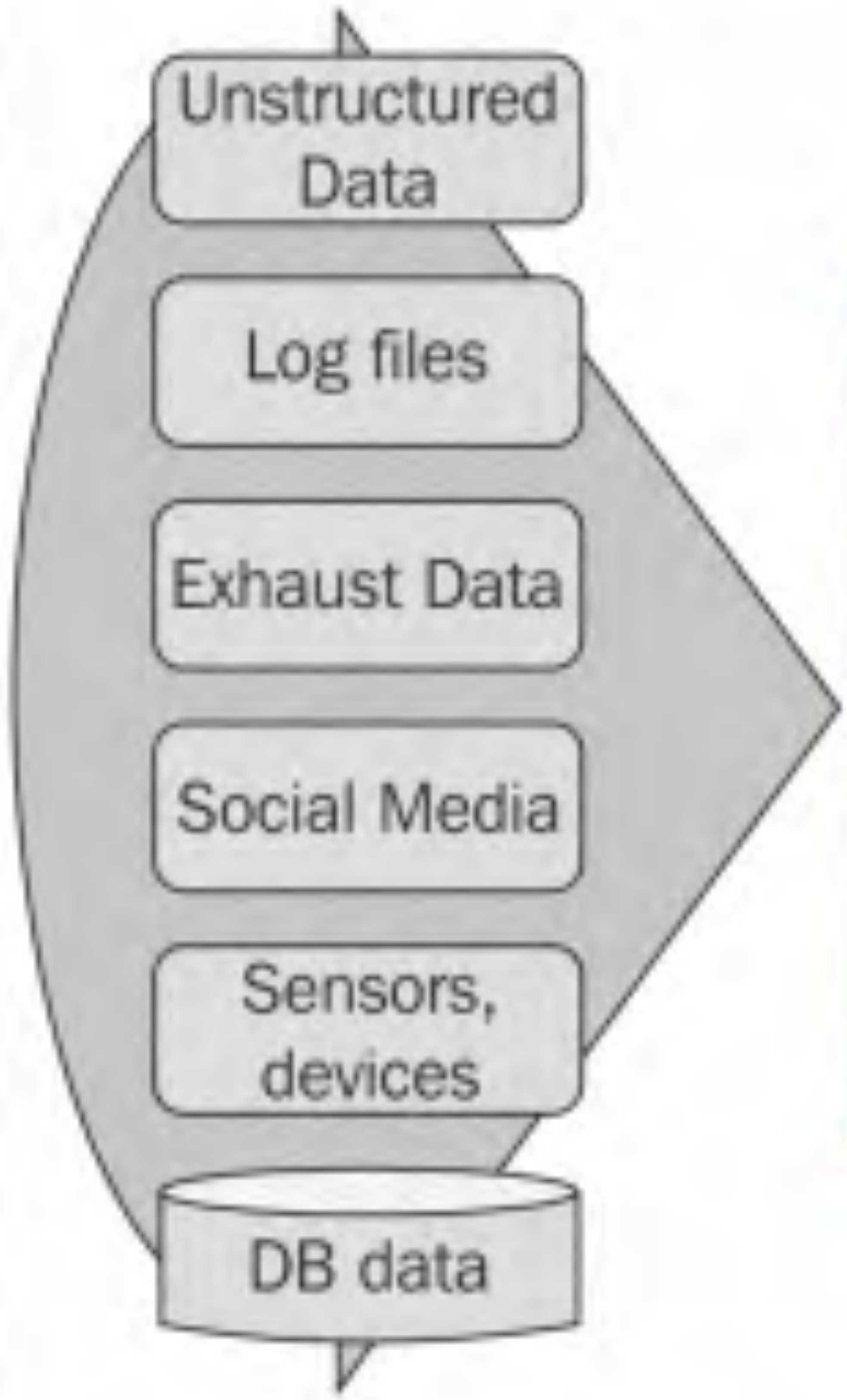


Hadoop

- Sqoop — Manage data between Hadoop and Relational DBs, enterprise DWs, and NoSQL
- Flume — Used for collecting, aggregating and moving log data from different sources
- Storm — Provides a solution for processing distributed data in real-time

Hadoop's data flow





1 **Capture Big Data** 2 P

1. Capture Big Data

- Sources can be extensive lists:
  - Structured, semi-structured, and unstructured,
  - some streaming, real-time data sources,
  - sensors, machine-captured data, and many other sources.

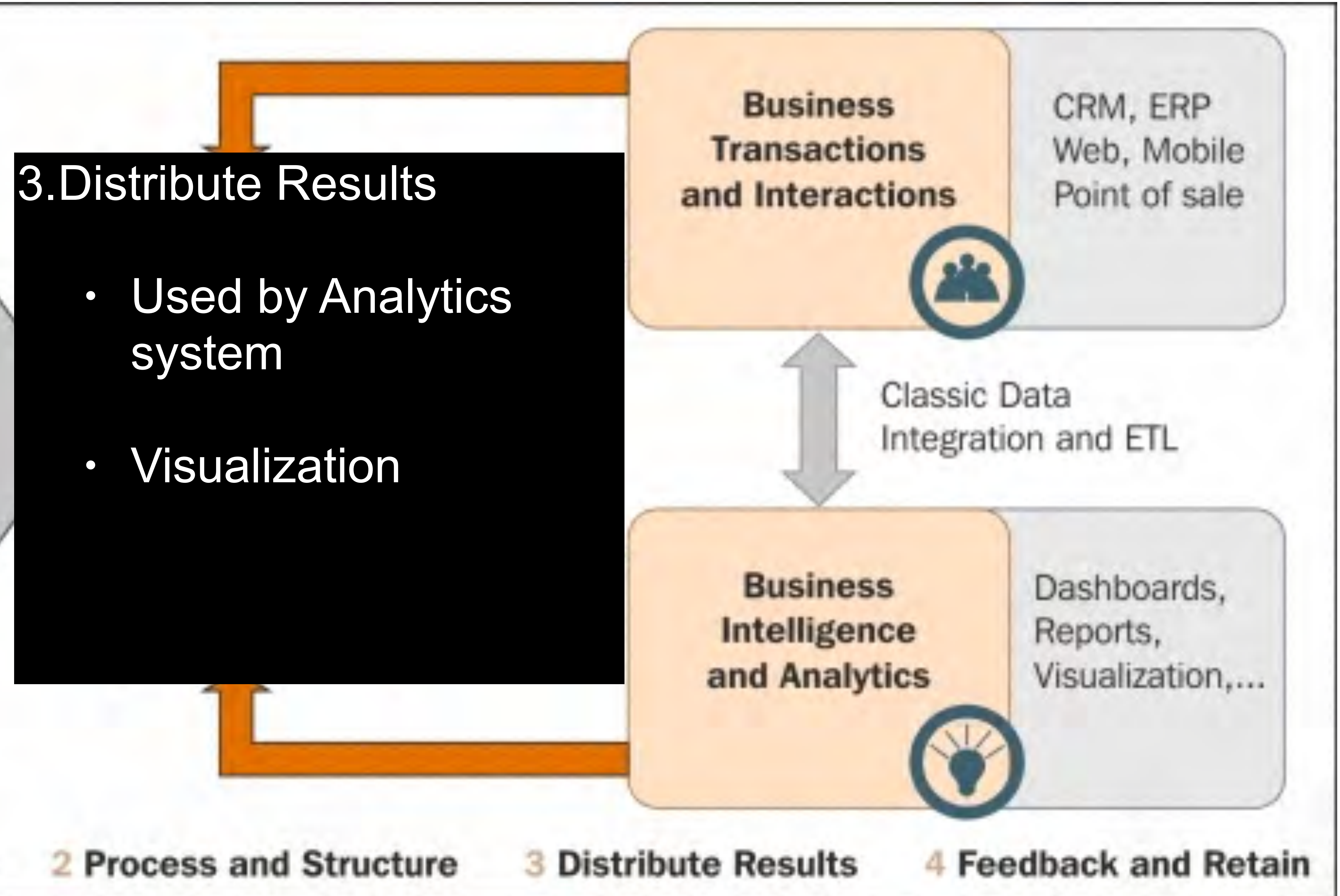
2. Process & Structure

- Cleanse
- Filter
- Transform
- Process

Sol

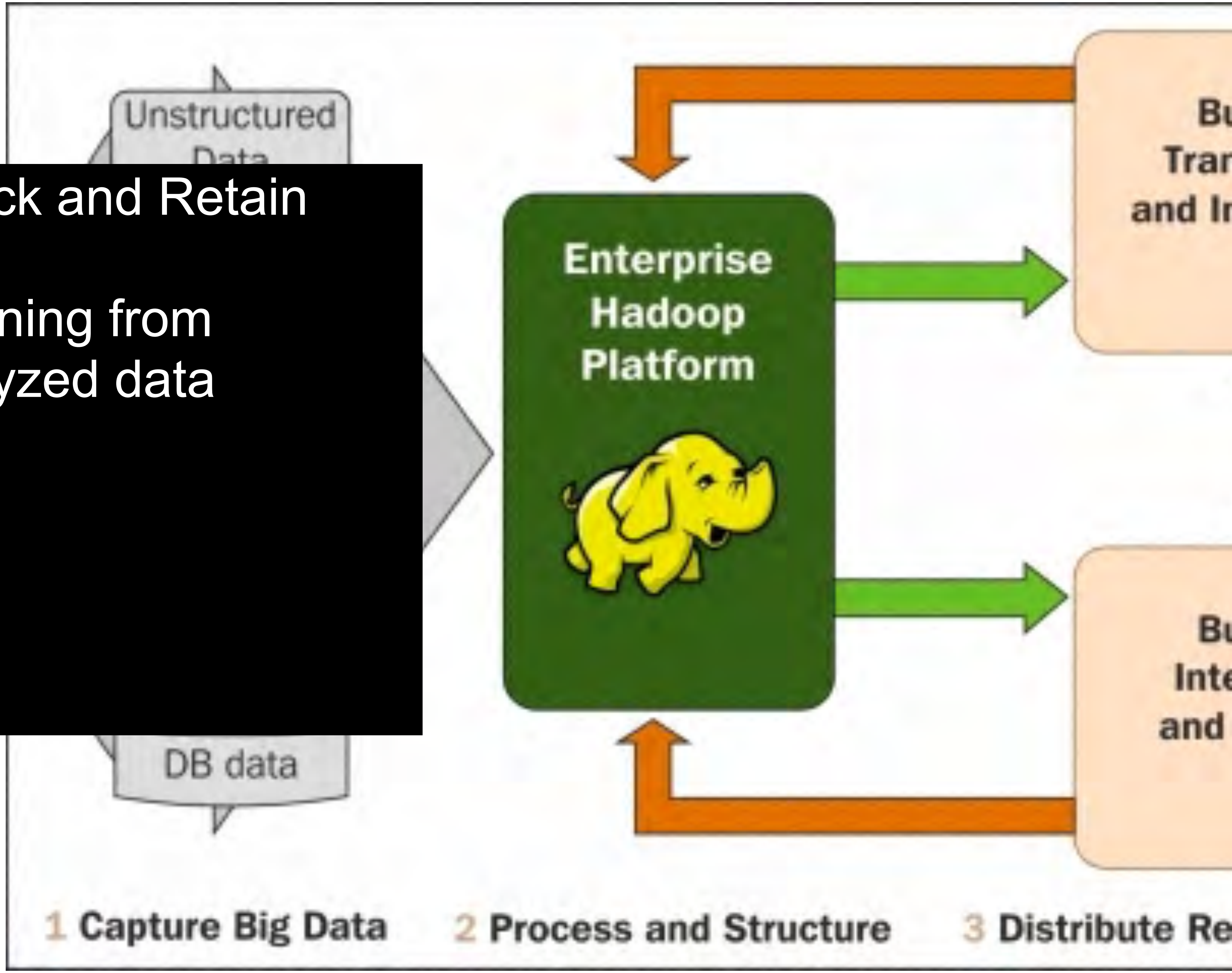
3.Distribute Results

- Used by Analytics system
- Visualization



3.Feedback and Retain

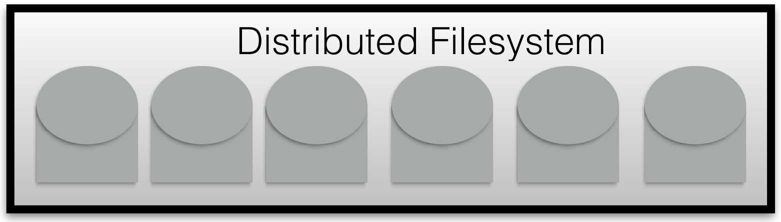
- Learning from analyzed data



Sohail Rafiqi

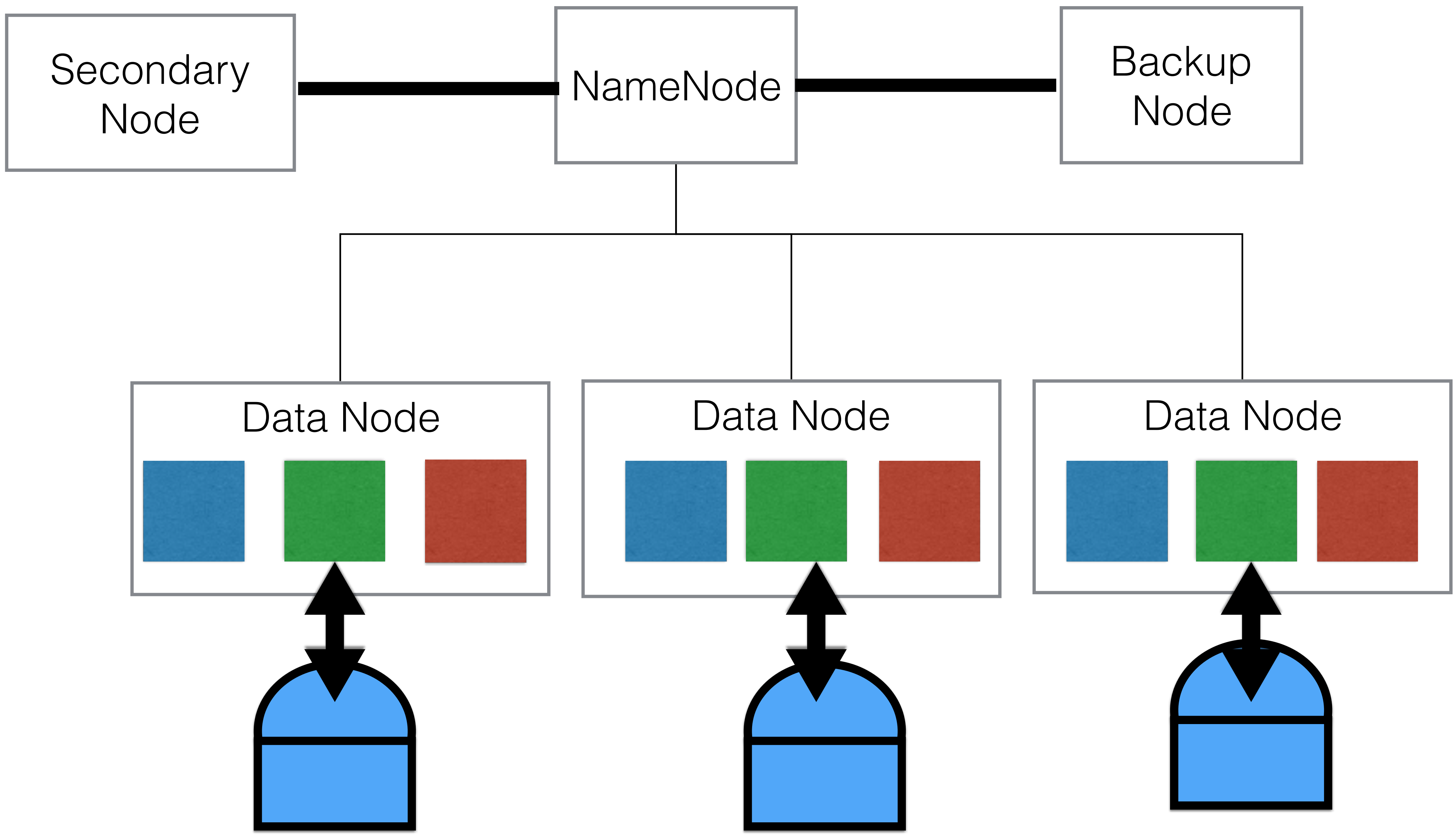
HDFS

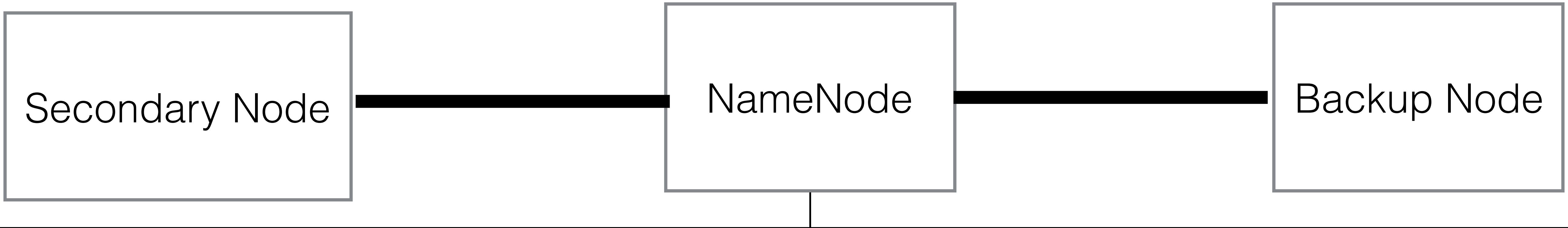
Features of HDFS



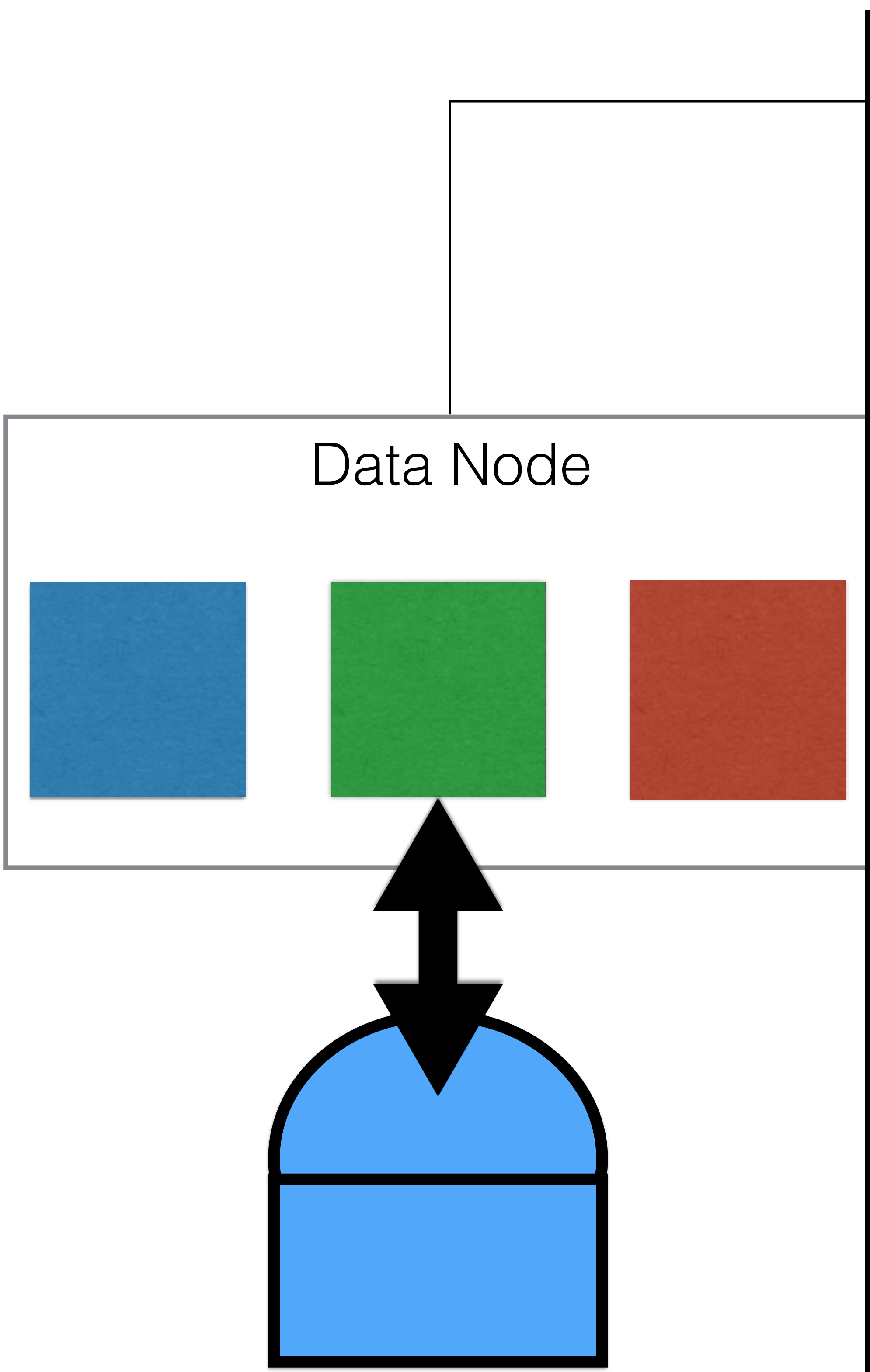
- Scalability
  - Ability to seamlessly add/delete nodes
- Reliability and fault tolerance
  - Replicates data to a configurable number of nodes
    - Data is stored in multiple nodes — system available in case of node failure
    - Replication Factor
      - ✓ Specify how many nodes you want to replicate a file
- Portability
  - Portable on different hardware and software
  - Built using the Java language — can be deployed on any machine supporting Java
- Processing closer to the data
  - Move query processing where data is rather than moving the data

HDFS Architecture





- Coordinates all storage operations including read/write in HDFS
- Manages filesystem namespace
- Holds Metadata about which nodes has what data block
    - Fsimage file — Holds entire filesystem namespace including mappings of blocks to file and filesystem
    - Editlog file — Holds ever change that occurs to the filesystem metadata
- Secondary NameNode is configured for HA
    - Keeps checkpoint of FsImage and EditLog merged and available in case of a failure to NameNode
    - Requires similar configuration as the NameNode
- Backup Node
    - Similar to Secondary NameNode but keeps the FsImage in the memory
    - Always sychrnozed with the NameNode
    - Has same RAM requirements as NameNode

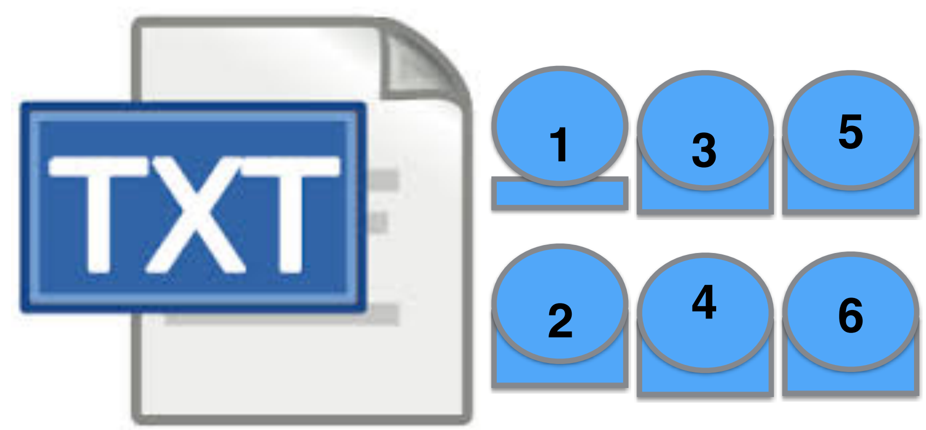


- Holds actual data
- Create, delete and replicate data blocks as assigned by NameNode
- DataNode sends heartbeat messages (3 sec) to the NameNode
    - No heartbeat — NameNode declares DataNode out of service
- Every 10th heartbeat is a block report
- NameNode updates metadata from block report

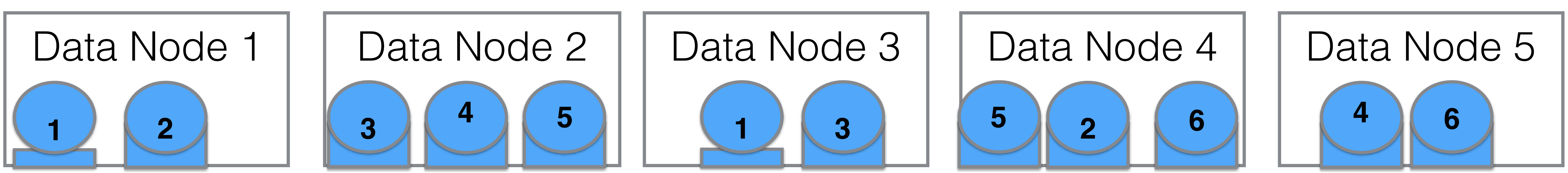
Sohail Rafiqi

Data Storage

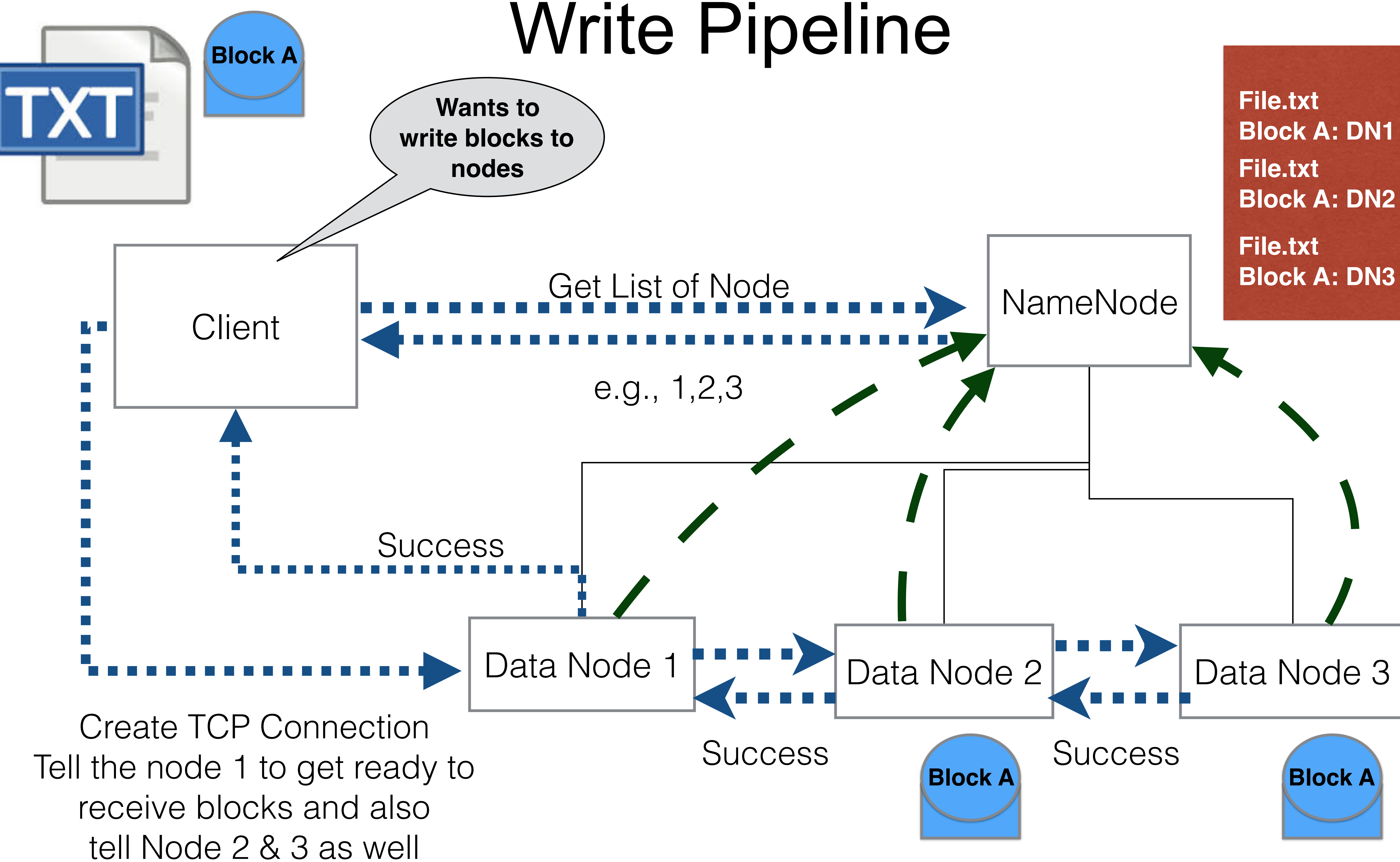
- Files are divided into blocks — Stored in multiple DataNodes
  - Blocks are configurable parameter in HDFS — default block size 128MB
- Replication Factor
  - Number of DataNodes a block is replicated
    - Default value is 3



Replication Factor —> 2



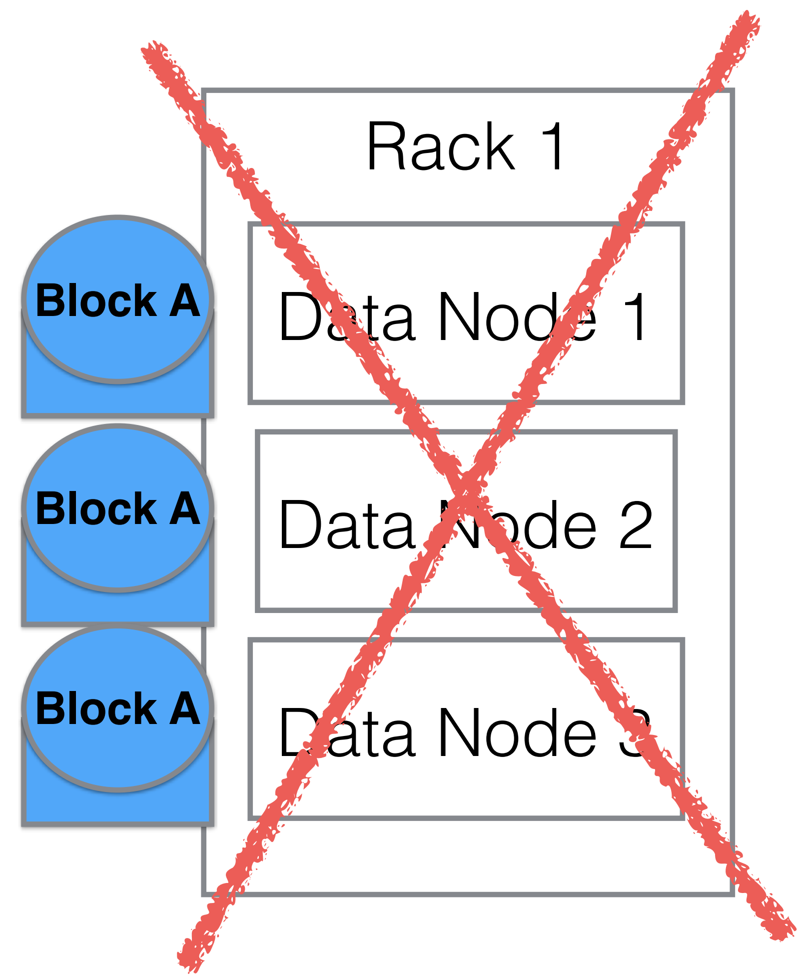
Write Pipeline



Sohail Rafiqi

Rack Awareness

- In practical deployment nodes are deployed in racks
- Each block of data is replicated to multiple nodes to eliminate single point of failure
- Hadoop is rack aware:
    - Data loss prevention
        - What happens if all of the blocks are in the same racks?
        - What happens if racks goes down?



Read Pipeline

- Similar to the write pipeline
- Clients gets a list of data nodes for each block
- Establishes TCP connection with the first node
- Reads all blocks sequentially

Re-replication

- If the Name Node stops receiving heartbeats from a Data Node
  - Presumes it to be dead and any data it had to be gone as well.
- Based on the block reports it had been receiving from the dead node
  - Name Node knows which copies of blocks died along with the node
  - Make the decision to re-replicate those blocks to other Data Nodes.
  - Also consult the Rack Awareness data in order to maintain at least one copy in another rack
- If entire rack of servers falls off the network,
  - The NameNode tells remaining nodes in the cluster to re-replicate all of the data blocks lost in that rack.
  - Large amount of data could potentially traverse the network

Re-replication

- HDFS scan blocks for corruption.

- Every DataNode checks the block present in it

  - Verifies with the stored checksum, which is generated during the block creation.

  - In case block corruption is identified, NameNode is informed

    - NameNode marks the block in the DataNode as corrupt and initiates a re-replication of the block.

HDFS Federation

- NameNode has a single namespace and tightly coupled with DataNodes
  - All requests have to coordinate with NameNode to get blocks' location
    - Can easily become a bottleneck and limiting factor
- HDFS Federation enables multiple namespaces
  - Responsibility of NameNode is shared across multiple namespaces
  - Load balancer is used to route requests
  - Federated NameNode works in multi-tenant environment

HDFS Commands

- Very similar to unix
  - hadoop fs <args>
- hadoop fs -mkdir <paths>
- Example:
  - hadoop fs -mkdir /usr/srafiqi/CloudComputing
  - hadoop fs -ls /usr/srafiqi/CloudComputing
  - Put/Get:
    - hadoop fs -put <localsrc> <HDFS destination path>

HBase Overview

HBase

- Column-oriented NoSQL, Distributed Database designed for queries of massive data sets
- HBase can run in distributed and pseudo-distributed modes
- Data is stored in tables
- In pseudo-distributed mode
    - All HBase daemons runs on a single node.
    - Can run on local file system or an instance of HDFS
- HBase support auto-swarding
    - Tables are dynamically split and distributed by the database when they become too large

HBase

- In HBase, each key has a structure
  - Row key, column family and timestamp
  - Actual key-value pair mapping in HBase
    - (row key, column family, column timestamp)

Flexible Data Model

- HBase is a wide-column data store based on BigTable Concept
- Basic unit of storage in HBase is a table
  - Consists of one or more column families
- Data is stored in rows
  - Collection of key/value pairs
- Each row is uniquely identified by a row key
  - Row keys are created when table data is added
  - Row keys are used to determine the sort order and for data sharding

Flexible Data Model

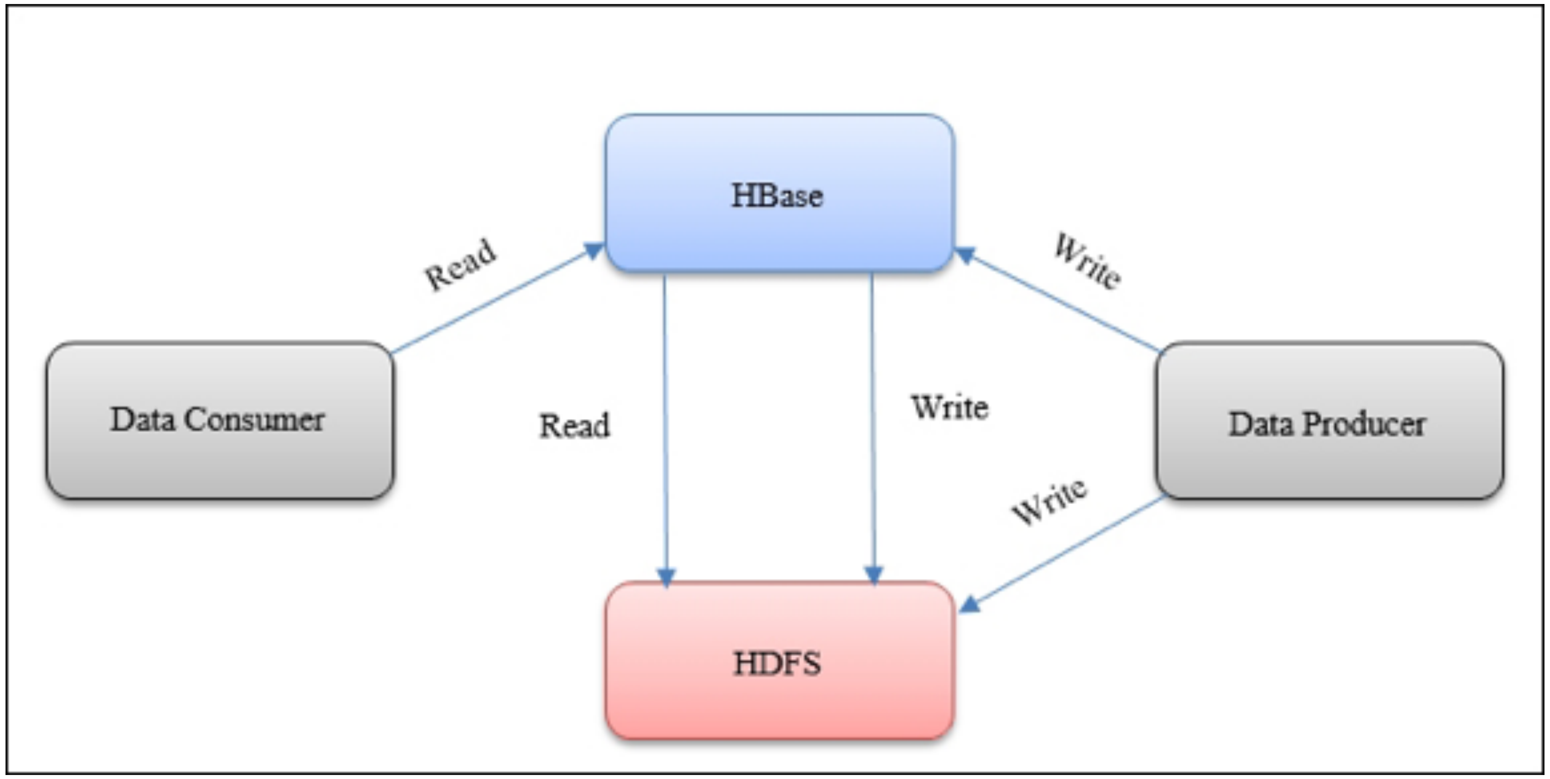
- Columns may be added to a table column family as required without predefining columns
- Only the table and column families are required to be defined in advance
- No two rows in a table are required to have the same columns
- All columns in the column family are stored closer
- HBase is strongly consistent (not eventual)
    - Data is always consistent — but potentially increase latency
- Does not have the notion of data types
    - All data is stored as an array of bytes

Scalable

- Basic unit of horizontal scalability is region
- Rows are shared by region
  - Region is a sorted set consisting of a range of adjacent rows stored together
- Table’s data can be stored in one or more regions
- When a region becomes too large
  - Split into two at the middle row key into equal region

Transactional Capabilities

- HBase adds transactional capabilities to Hadoop
- Allows user to conduct CRUD operation in the DB
- Provides real-time random read/write access to data stored in HDFS



Main Features of HBase

- Linear and modular scalability
- Fault-tolerant storage for a huge amount of data
- Flexible data model
- Support for atomic operations
- Atomic sharding
- Easy-to-use Java API for client access
- Replication across the data center
- Strictly consistent read and writes
- Block cache and bloom filters for real-time queries

HDFS and HBase

HDFS	HBase
HDFS is a distributed file system well suited for the storage of large files. It is not a general-purpose filesystem and does not provide fast individual record lookups in files.	HBase is built on top of HDFS and provides fast record lookups for large tables. It internally puts the data in indexed StoreFiles available in HDFS for fast lookups.
It is suitable for high latency operations batch processing.	It is built for low latency operations.
The data here is mainly accessed through MapReduce.	It provides access to single rows from billions of records.
It has been designed for batch processing, so it does not have the concept of random read and writes.	It supports random read and writes.

Data Model

- **Table** — All data is stored there — Logical collection of rows
- **Rows** — Data is stored in rows, rowKey (unique identifier) but has no data type
- **Column Family** — Columns are grouped in column families.
  - All columns of column family has same prefix (c1)
  - All the columns in column family are stored in the File. Required to declare column families upfront
- **Column Qualifier** — In each column family, data is addressed in a column identifier
- Timestamp — Version in HBase. Whenever a value is written, it is stored with a version number
  - Version number is a timestamp of when cell was written

Table X

rowkey	Column Family	Column Qualifier	Timestamp	Value
x	C1	“foo”	“1223455634”	5
			“1245564567”	“Hadoop”
		“test”	“1235674323”	34
			“1256348945”	564
			“3467893423”	“Guide”
	C2	“1.0002”	“1235576347”	“Yet another value”
		“2-12-2014”	“1235752574”	“Completed Task”
y	C2	“CValue”	“1235573212”	3576
		“qualifier”	“1245322456”	“My Data”

Accessing data

- Four primary data operations that we can execute to access data in HBase:
  - Get, Put, Scan, and Delete
- Get — Returns an attribute for a specified row.
- Put — Used to either add a new row in the table or update an existing row.
- Scan — Allows to iterate over multiple rows for a defined attribute.
  - Using scan, user can match columns with filters and other attributes, such as the start and end time.
- Delete — This is used to remove a row from a table.