Big Data

- Volume
- Variety
- Velocity
- Veracity
- Volume

Need for a new technology stack ?

Monolithic vs Distributed

One powerful system

Cluster of systems (commodity machines)

Compute, Memory, Storage

2x resources ≠ 2x performance

Vertical Scaling

Horizontal Scaling (True Scaling)

Design a good Big Data System ? Consider

- Storage (distributed storage)
- Process (distributed processing)
- Scalability (↑ number of nodes)

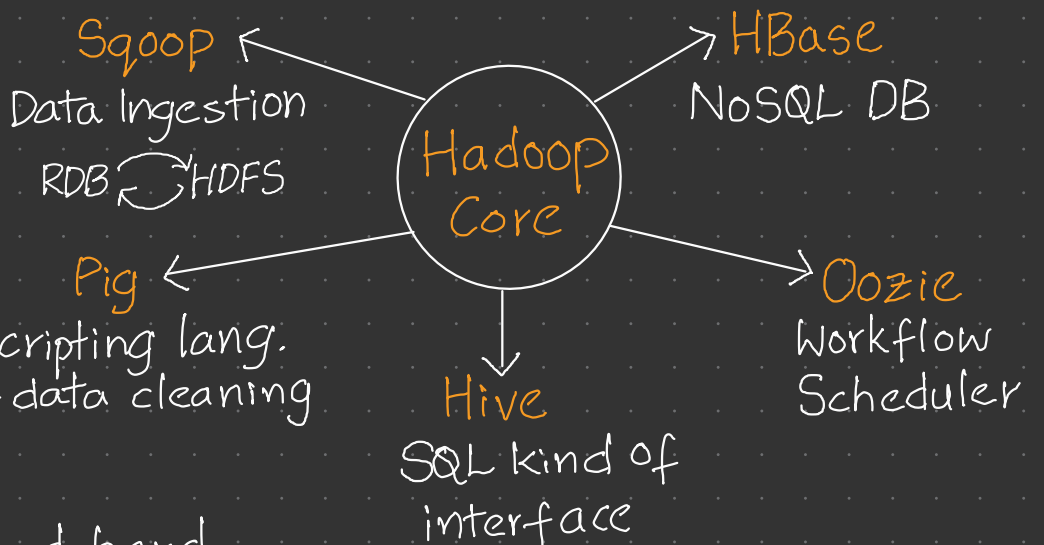
Hadoop → first framework designed to solve Big Data problems

2006 → 2012 → Now
1.0 → 2.0 → 3.0

Core Components

- HDFS
- Map Reduce
- YARN (added in 2.0)

Hadoop Ecosystem



Challenges

- MR is very slow and hard
- Need to learn different components for different tasks

Advantages of Cloud

- Scalable
- Minimum OpEx
- Agility
- Geo Distribution
- Disaster Recovery

On-Premise Vs Cloud

On-Premise	Cloud
- Buy the needed Infra, like Space to hold the servers	- Infra is taken care by Cloud providers
- Buy servers	- No need to buy servers
- Setup a cooling system	- Setup the cluster with a click of a button
- Hire a technical team to install and maintain S/W	- Low maintenance cost
- Huge upfront cost / Capex & Opex	- No upfront cost / Capex
- Not very scalable	- Highly Scalable

Cloud Types

- Public : AWS, Azure, GCP
- Private : when the data is very confidential
- Hybrid

Spark → general-purpose in-memory compute engine

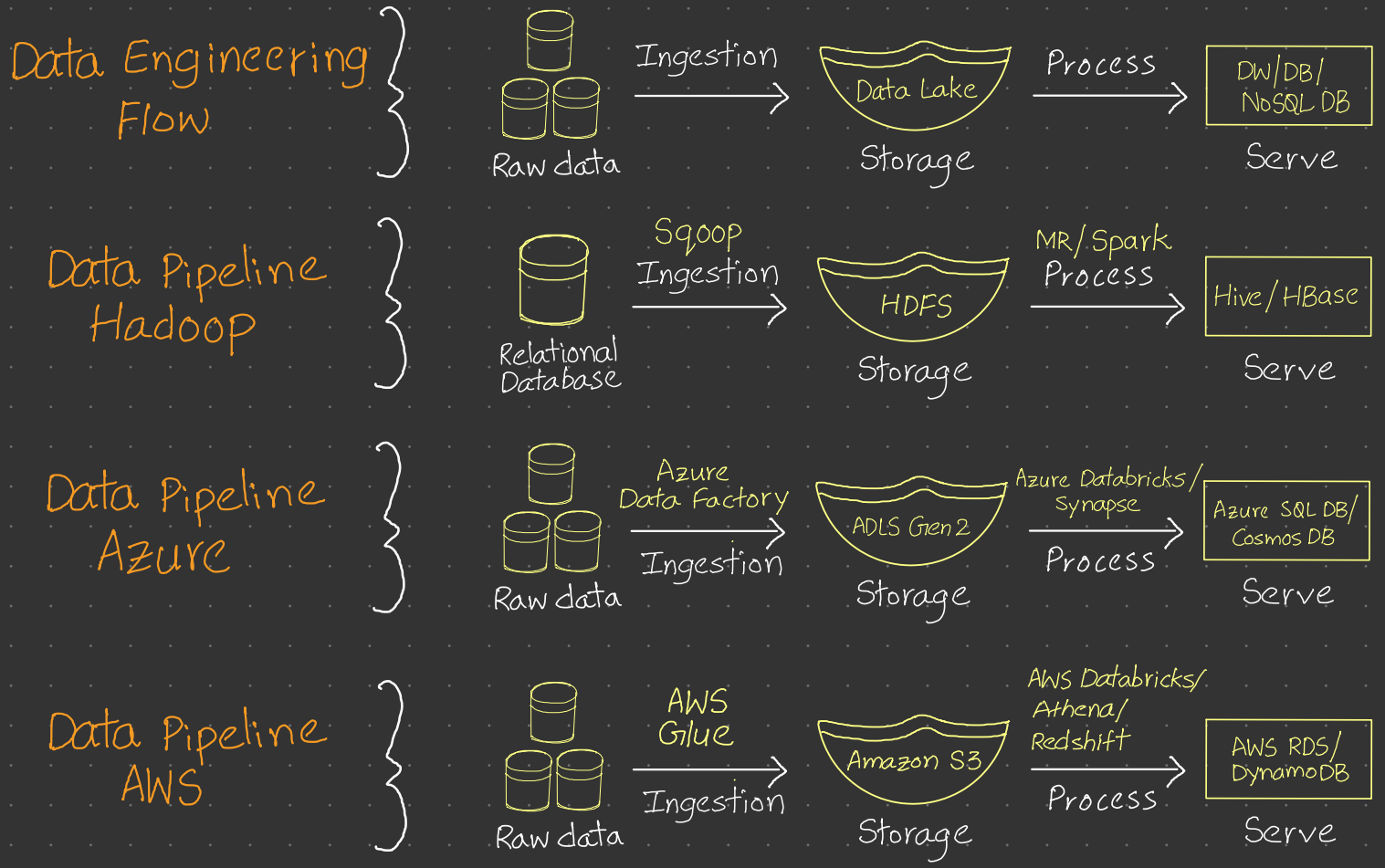
- Spark can act as a replacement for MapReduce (not for Hadoop)
- It is a plug and play compute engine
- Needs storage and resource manager to function.
  - not bound to HDFS, and YARN, only.
    - Amazon S3, ADLS Gen 2, GIC Storage, Local Storage
    - Mesos, Kubernetes

- Spark is 10x - 100x faster than MR

Polyglot

- Provides high-level APIs in Java, Scala, Python and R
- Spark is written in Scala

	Database	Data Warehouse	Data Lake
Workloads	- OLTP	- OLAP	- OLAP
Data type	- Structured or Semi-Structured	- Structured and/or Semi-Structured	- Structured, Semi-Structured and/or unstructured
Schema	- Schema-on-write	- Schema-on-write	- Schema-on-read
Freshness	- operational current data	- current and historical data	- current and historical data
Prior Processing	- ETL	- ETL	- ELT
Cost	- Higher	- High	- Low
Examples	- PostgreSQL, Oracle, MySQL	- Teradata, AWS Redshift	- Amazon S3, ADLS Gen 2
Users	- Application developers	- Business Analysts	- Data Scientists


Data Engineering Flow
Raw data
Ingestion
Data Lake
Storage
Process
DW/DB/ NoSQL DB
Serve
Data Pipeline Hadoop
Relational Database
Sqoop Ingestion
HDFS
Storage
MR/Spark Process
Hive/HBase
Serve
Data Pipeline Azure
Raw data
Azure Data Factory
Ingestion
ADLS Gen2
Storage
Azure Databricks/ Synapse
Process
Azure SQL DB/ Cosmos DB
Serve
Data Pipeline AWS
Raw data
AWS Glue
Ingestion
Amazon S3
Storage
AWS Databricks/ Athena/ Redshift
Process
AWS RDS/ DynamoDB
Serve

Serverless Computing

- Resources are not dedicated
  ↓
  No guaranteed performance
- Less expensive
- Good for scheduled jobs
  ↓
  that are not particular about the time in which they get finished
- Eg:- Athena, Synapse serverless

Serverful Computing

- Resources are dedicated
  ↓
  guaranteed performance
- More expensive
- Good for ad-hoc jobs
  ↓
  that require immediate execution
- Eg:- Redshift, Synapse dedicated pool

HDFS Architecture

- Usually each block is of 128 MB
- the blocks of files gets replicated for fault tolerance (default RF is 3)

file 1

Block 1
Block 2
Block 3
Block 4

Powerful machine

NameNode → Metadata

file 1	B1	DN 1
file 1	B2	DN2
file 1	B3	DN3
file 1	B4	DN4

Secondary NameNode → backup for NameNode

heartbeat

- Every DN sends a heartbeat to NN every 3 seconds indicating that it's alive
- If the NN does not receive a heartbeat for 30 seconds, then NN assumes DN is dead

Client

DataNode 1 [B1 Disk]

DataNode 2 [B2 Disk]

DataNode 3 [B3 Disk]

DataNode 4 [B4 Disk]

Commodity machines

Data locality

↓

Move the computation close to where the actual data resides on the node