CSE 424

Big Data

Big Data Stack and Storage Concept

Slides 2

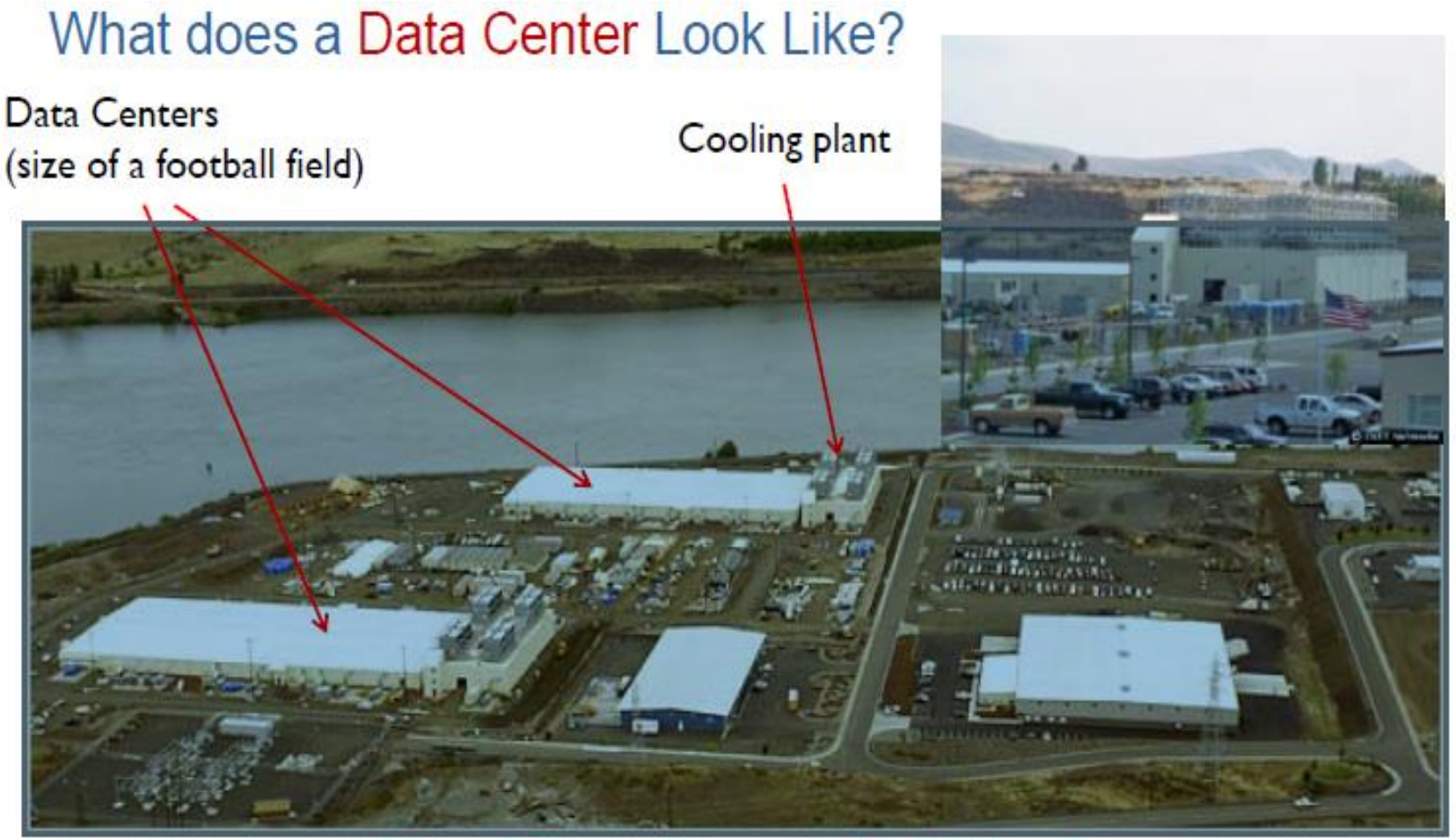
Instructor: Asst. Prof. Dr. Hüseyin ABACI

Outline

- Big Data Stack
  - Raw Data Sources
  - Data Access Connectors
  - Data Storage
  - Batch Analytics
  - Serving Databases, Web & Visualization Frameworks
- Mapping Analytics Flow to Big Data Stack
- Scaling Approaches
- Clusters
- Hadoop Distributed File Systems (HDFS)
- NoSQL
  - Sharding
  - Join
- CAP Theorem
- ACID
- BASE

Thomas Erl, Wajid Khattak, and Paul Buhler, Big Data Fundamentals Concepts: Drivers & Techniques, 2016
Bahga, Arshdeep, and Vijay Madisetti. Big data science & analytics: A hands-on approach. VPT, 2016.

Bigdata can be Analysed on Massive Datacenters



Google Data Center in The Dalles, Oregon

- **A single Data Center** can easily contain **10,000 racks** with 100 cores in each rack (1,000,000 cores total)

What is in Datacenters

What is in Datacenters

- A single-site ***cloud (aka "Datacenter")*** consists of
- Compute nodes (grouped into racks).
- Switches, connecting the racks.
- A network topology, e.g., hierarchical.
- Storage (backend) nodes connected to the network
- Front-end for submitting jobs.
- Software services.

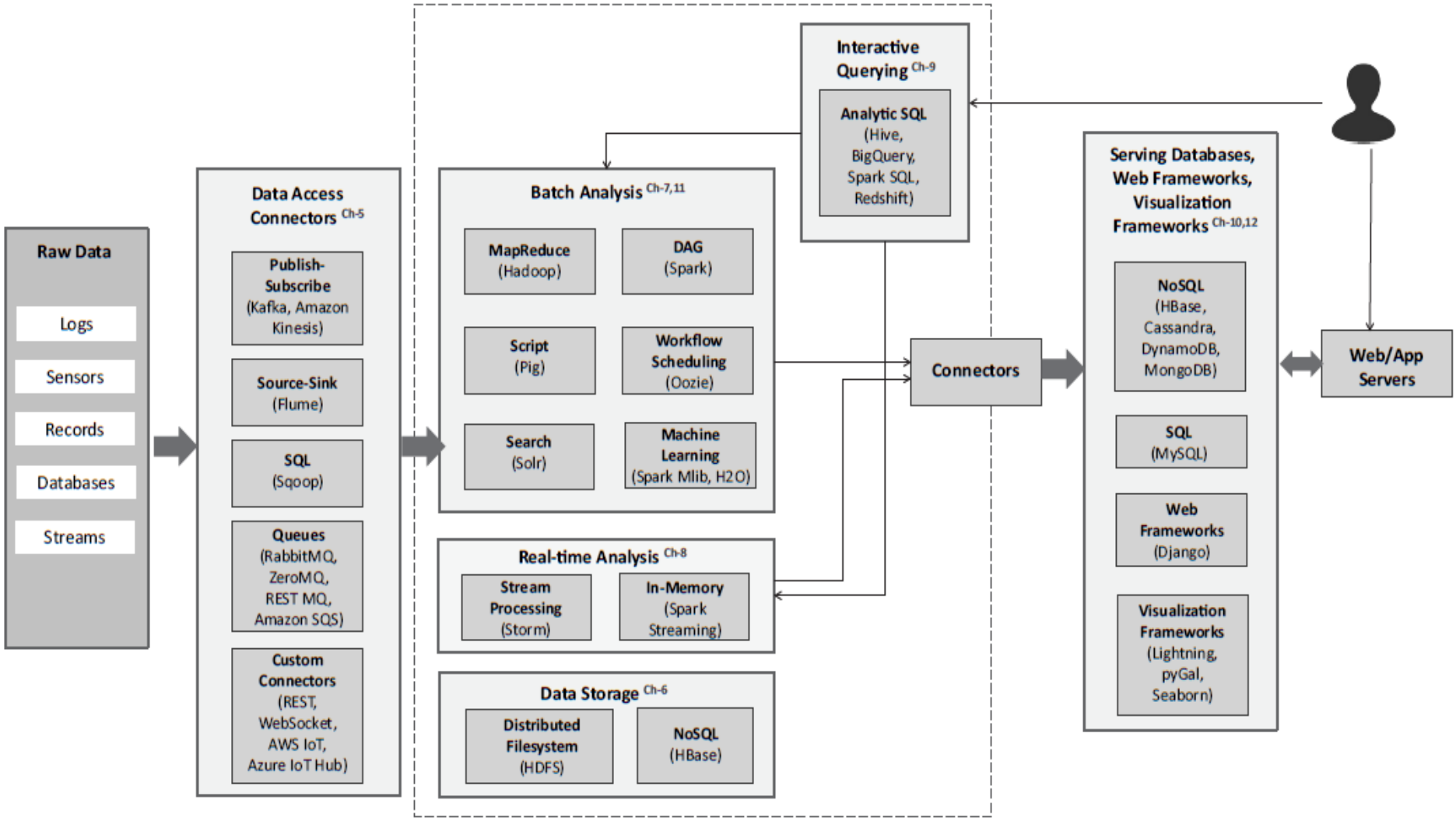
- A geographically distributed cloud consists of multiple such sites.

What is in Datacenters

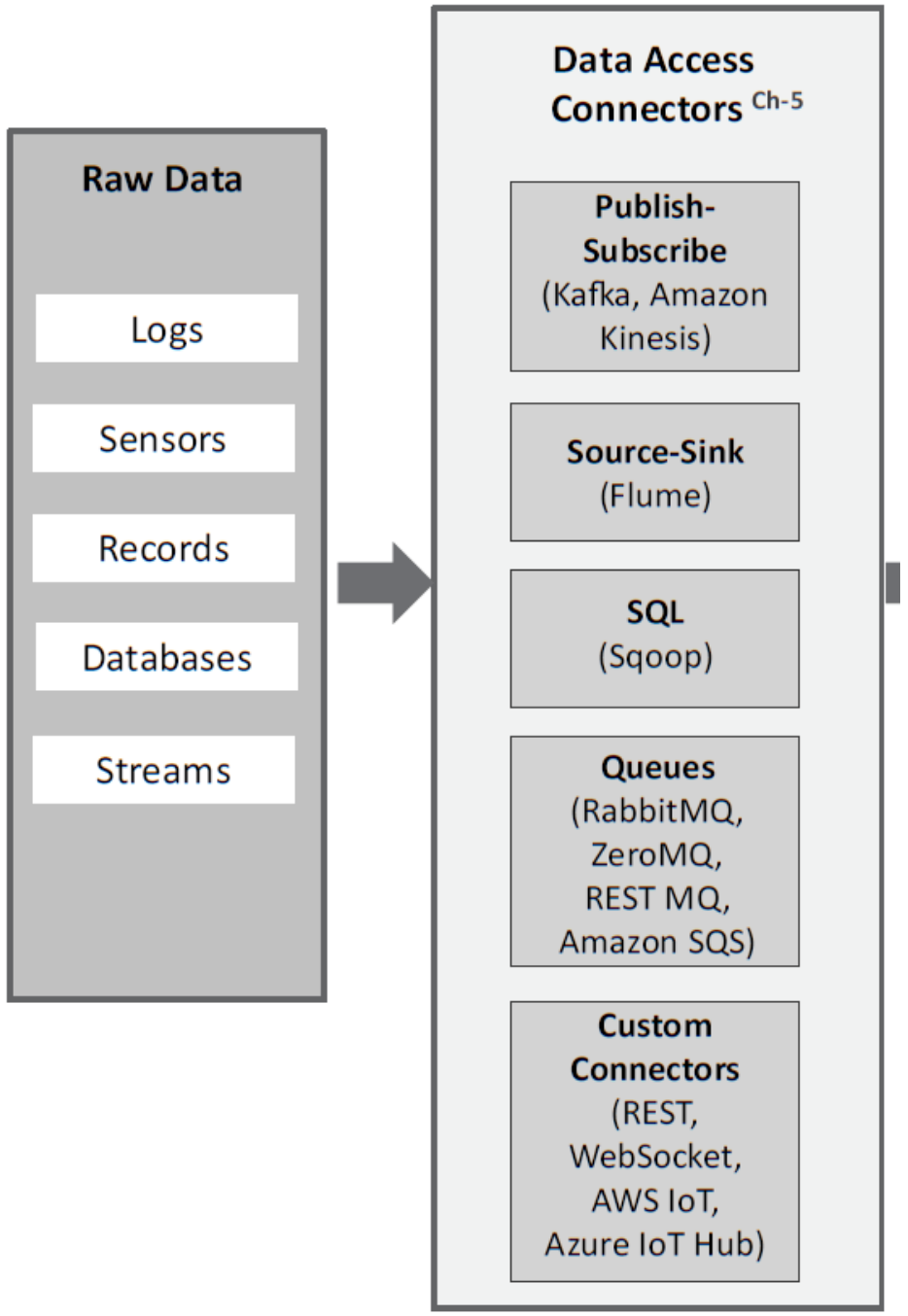
Big Data Stack

- While the Hadoop framework has been one of the most popular frameworks for big data analytics, there are several types of computational tasks for which Hadoop does not work well.
- Hadoop is an open source framework for distributed batch processing of massive scale data using the MapReduce programming model.
- The MapReduce programming model is useful for applications in which the data involved is so massive that it would not fit on a single machine.
- In such applications, the data is typically stored on a distributed file system (such as Hadoop Distributed File System - HDFS).
- MapReduce programs take advantage of locality of data and the data processing takes place on the nodes where the data resides.
- In other words, the computation is moved to where the data resides, as opposed the traditional way of moving the data from where it resides to where the computation is done.

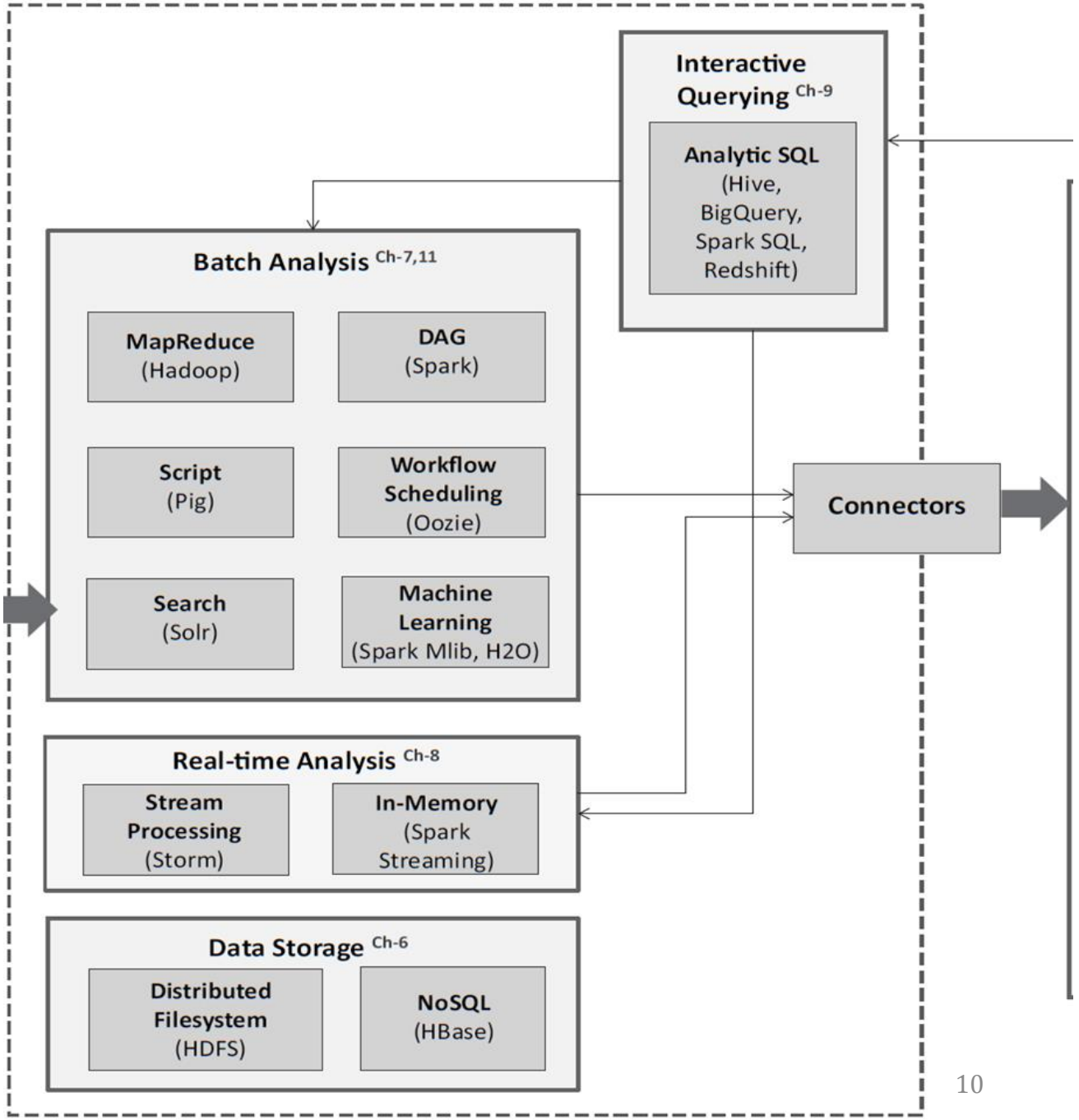
Big Data Stack



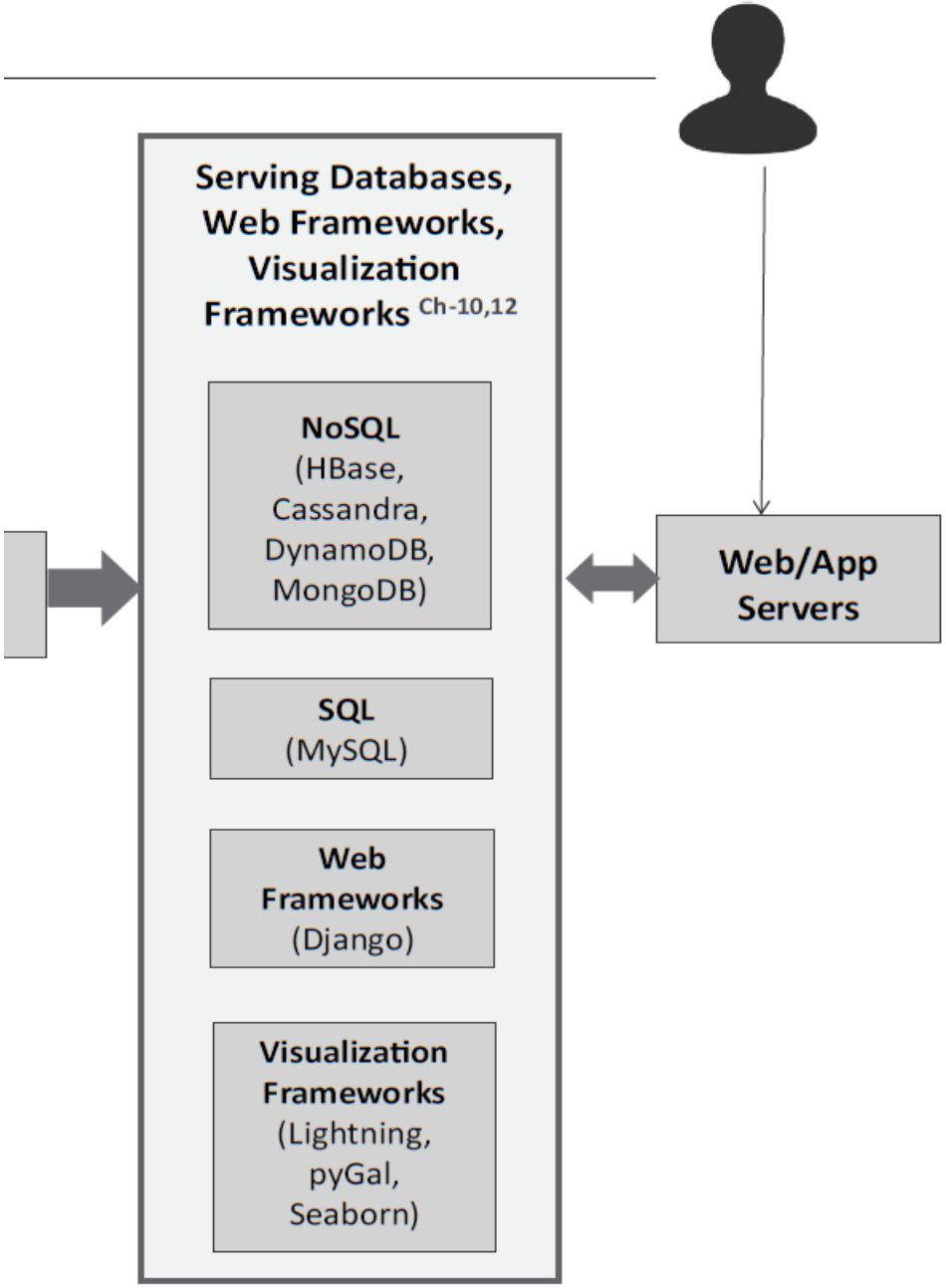
Big Data Stack



Big Data Stack



Big Data Stack



Big Data Stack- Raw Data Sources

- In any big data analytics application or platform, before the data is processed and analyzed, it must be captured from the raw data sources into the big data systems and frameworks. Some of the examples of raw big data sources include:
  - **Logs:** Logs generated by web applications and servers which can be used for performance monitoring
  - **Transactional Data:** Transactional data generated by applications such as eCommerce, Banking and Financial
  - **Social Media:** Data generated by social media platforms
  - **Databases:** Structured data residing in relational databases
  - **Sensor Data:** Sensor data generated by Internet of Things (IoT) systems
  - **Clickstream Data:** Clickstream data generated by web applications which can be used to analyze browsing patterns of the users
  - **Surveillance Data:** Sensor, image and video data generated by surveillance systems
  - **Healthcare Data:** Healthcare data generated by Electronic Health Record (EHR) and other healthcare applications
  - **Network Data:** Network data generated by network devices such as routers and firewalls

Big Data Stack- Data Access Connectors

- The Data Access Connectors includes tools and frameworks for collecting and ingesting data from various sources into the big data storage and analytics frameworks. The choice of the data connector is driven by the type of the data source.
  - **Publish-Subscribe Messaging:** Publish-Subscribe is a communication model that involves publishers, brokers and consumers. Publishers are the source of data. Publishers send the data to the topics which are managed by the broker. Publish-subscribe messaging frameworks such as Apache Kafka and Amazon Kinesis.
  - **Source-Sink Connectors:** Source-Sink connectors allow efficiently collecting, aggregating and moving data from various sources (such as server logs, databases, social media, streaming sensor data from Internet of Things devices and other sources) into a centralized data store (such as a distributed file system). Apache Flume, which is a framework for aggregating data from different sources. Flume uses a data flow model that comprises sources, channels and sinks.
  - **Database Connectors:** Database connectors can be used for importing data from relational database management systems into big data storage and analytics frameworks for analysis. Apache Sqoop, which is a tool that allows importing data from relational databases.

Big Data Stack- Data Access Connectors (cont'd)

- **Messaging Queues:** Messaging queues are useful for push-pull messaging where the producers push data to the queues and the consumers pull the data from the queues. The producers and consumers do not need to be aware of each other. Examples are RabbitMQ, ZeroMQ, RestMQ and Amazon SQS.
- **Custom Connectors:** Custom connectors can be built based on the source of the data and the data collection requirements. Some examples of custom connectors include: custom connectors for collecting data from social networks, custom connectors for NoSQL databases and connectors for Internet of Things (IoT). Examples are REST (REpresentational State Transfer), WebSocket and MQTT (Message Queue Telemetry Transport). IoT connectors such as CoAP (Constrained Application Protocol), AWS IoT and Azure IoT Hub are also available.

Big Data Stack- Data Storage

- The data storage block in the big data stack includes distributed file systems and non-relational (NoSQL) databases, which store the data collected from the raw data sources using the data access connectors.
- The Hadoop Distributed File System (HDFS), a distributed file system that runs on large clusters and provides high-throughput access to data. With the data stored in HDFS, it can be analysed with various big data analytics frameworks built on top of HDFS.
- For certain analytics applications, it is preferable to store data in a NoSQL database such as HBase. HBase is a scalable, non-relational, distributed, column-oriented database that provides structured data storage for large tables.

Big Data Stack- Batch Analytics

- The batch analytics block in the big data stack includes various frameworks which allow analysis of data in batches. These include the following:
  - **Hadoop-MapReduce:** Hadoop is a framework for distributed batch processing of big data. The MapReduce programming model is used to develop batch analysis jobs which are executed in Hadoop clusters.
  - **Pig:** Pig is a high-level data processing language (in between SQL and programing language) which makes it easy for developers to write data analysis scripts which are translated into MapReduce programs by the Pig compiler.
  - **Oozie:** Oozie is a workflow scheduler system that allows managing Hadoop jobs. With Oozie, you can create workflows which are a collection of actions (such as MapReduce jobs) arranged as Direct Acyclic Graphs (DAG).
  - **Spark:** Apache Spark is an open source cluster computing framework for data analytics. Spark includes various high-level tools for data analysis such as Spark Streaming for streaming jobs, Spark SQL for analysis of structured data, Mllib machine learning library for Spark, and GraphX for graph processing.

Big Data Stack- Batch Analytics (cont'd)

- **Solr:** Apache Solr is a scalable and open-source framework for searching data. Used for indexing documents.
- **Machine Learning**: Spark MLlib is the Spark's machine learning library which provides implementations of various machine learning algorithms. H2O is an open source predictive analytics framework which provides implementations of various machine learning algorithms.

Big Data Stack- Real-time Analytics

- The real-time analytics block includes the Apache Storm and Spark Streaming frameworks.
- Apache Storm is a framework for distributed and fault-tolerant real-time computation. Storm can be used for real-time processing of streams of data. Storm can consume data from a variety of sources such as publish-subscribe messaging frameworks (such as Kafka or Kinesis), messaging queues (such as RabbitMQ or ZeroMQ) and other custom connectors.
- Spark Streaming is a component of Spark which allows analysis of streaming data such as sensor data, click stream data, web server logs, for instance. The streaming data is ingested and analyzed in micro-batches. Spark Streaming enables scalable, high throughput and fault-tolerant stream processing.

Big Data Stack- Serving Databases, Web & Visualization Frameworks

- While the various analytics blocks process and analyze the data, the results are stored in serving databases for subsequent tasks of presentation and visualization. These serving databases allow the analyzed data to be queried and presented in the web applications.
- **MySQL**: MySQL is one of the most widely used Relational Database Management System (RDBMS) and is a good choice to be used as a serving database for data analytics applications where the data is structured.
- **Amazon DynamoDB**: Amazon DynamoDB is a fully-managed, scalable, high-performance NoSQL database service from Amazon. DynamoDB is an excellent choice for a serving database for data analytics applications as it allows storing and retrieving any amount of data and the ability to scale up or down the provisioned throughput.
- **Cassandra:** Cassandra is a scalable, highly available, fault tolerant open source non-relational database system.
- **MongoDB:** MongoDB is a document oriented non-relational database system. MongoDB is powerful, flexible and highly scalable database designed for web applications and is a good choice for a serving database for data analytics applications.

Mapping Analytics Flow to Big Data Stack

- Some guidelines in mapping the analytics flow to the big data stack.
- If the data is to ingested in bulk (such as log files), then a source-sink such as Apache Flume can be used.
- If high-velocity data is to be ingested at real-time, then a distributed publish-subscribe messaging framework such as Apache Kafka or Amazon Kinesis can be used.
- For ingesting data from relational databases, a framework such as Apache Sqoop can be used.
- If IoT devices generating sensor data may be resource and power constrained, in which case a light-weight communication protocol such as MQTT or CoAP may be chosen and a custom MQTT-based connector can be used.

Mapping Analytics Flow to Big Data Stack (cont'd)

Data Collection

Analysis Type	Framework (Mode)
Publish-Subscribe	Kafka, Kinesis
Source-Sink	Flume
SQL	Sqoop
Queues	SQS, RabbitMQ, ZeroMQ, RESTMQ
Custom Connectors	REST, WebSocket, MQTT

Data Preparation

Analysis Type	Framework
Data Cleaning	Open Refine
Data Wrangling	Open Refine DataWrangler
De-Duplication	Open Refine, Pig, Hive, Spark SQL
Normalization Sampling, Filtering	MapReduce, Pig, Hive, Spark SQL

Mapping Analytics Flow to Big Data Stack (cont'd)

Basic Statistics

Analysis Type	Framework (Mode)
Counts, Max, Min, Mean, Top-N, Distinct	Hadoop-MapReduce (Batch), Pig (Batch), Spark (Batch), Spark Streaming (Realtime), Spark SQL (Interactive), Hive (Integrative), Storm (Real-time)
Correlations	Hadoop-MapReduce (Batch), Spark Mlib (Batch)

Clustering

Analysis Type	Framework (Mode)
K-Means	Hadoop-MapReduce (Batch), Spark Mlib (Batch & Real-time) H2O (Batch)
DBSCAN	Spark (Batch)
Gaussian Mixture	Spark Mlib (Batch)
PIC	Spark Mlib (Batch)
LDA	Spark Mlib (Batch)

Mapping Analytics Flow to Big Data Stack (cont'd)

Classification

Analysis Type	Framework (Mode)
KNN	Spark Mlib (Batch , Realtime)
Decision Trees	Spark Mlib (Batch, Realtime)
Random Forest	Spark Mlib (Batch , Realtime), H2O (Batch)
SVM	Spark Mlib (Batch , Realtime)
Naïve Bayes	Spark Mlib (Batch, Realtime), H2O (Batch)
Deep Learning	H2O (Batch)

Regression

Analysis Type	Framework (Mode)
Linear Least Squares	Spark Mlib (Batch, Realtime)
Generalized Linear Model	H2O (Batch)
Stochastic Gradient Descent	Spark Mlib (Batch, Realtime)
Isotonic Regression	Spark Mlib (Batch, Realtime)

Mapping Analytics Flow to Big Data Stack (cont'd)

Dimensionality Reduction

Analysis Type	Framework (Mode)
SVD	Spark Mlib (Batch)
PCA	Spark Mlib (Batch), H2O (Batch)

Recommendation

Analysis Type	Framework (Mode)
Item-bases Recommendation	Spark Mlib (Batch)
Collaborative Filtering	Spark Mlib (Batch)

Mapping Analytics Flow to Big Data Stack (cont'd)

Text Analysis

Analysis Type	Framework (Mode)
Categorization	Hadoop-MapReduce (Batch), Storm (Realtime), Spark (Batch, Realtime)
Summarization	Spark (Batch)
Sentiment Analysis	Storm (Realtime), Spark (Batch, Realtime)
Text Mining	Storm (Realtime), Spark (Batch, Realtime)

Visualization

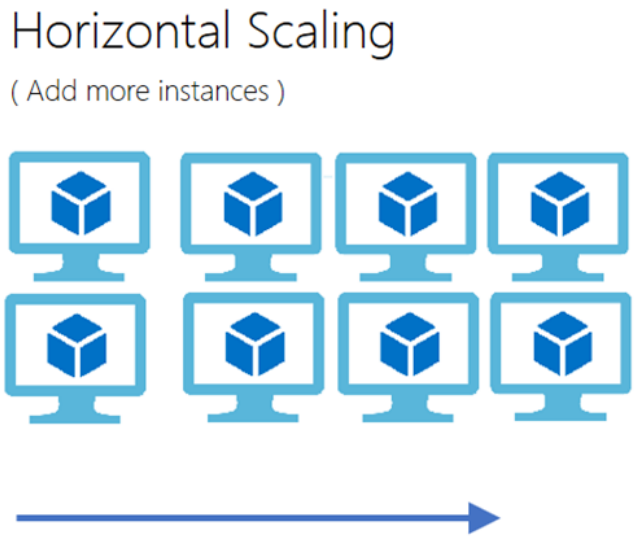
Analysis Type	Framework (Mode)
Web Frameworks	Django, Flask
SQL Databases	MySQL
NoSQL Databases	Hbase, DynamoDB, Cassandra, MongoDB
Visualization Frameworks	Lightning, pyGal, Seaborn

Scaling Approaches

- **Vertical Scaling/Scaling up:**
- Involves upgrading the hardware resources (adding additional computing, memory, storage or network resources to the same server).
- Can be expensive.
- Has a performance limit.

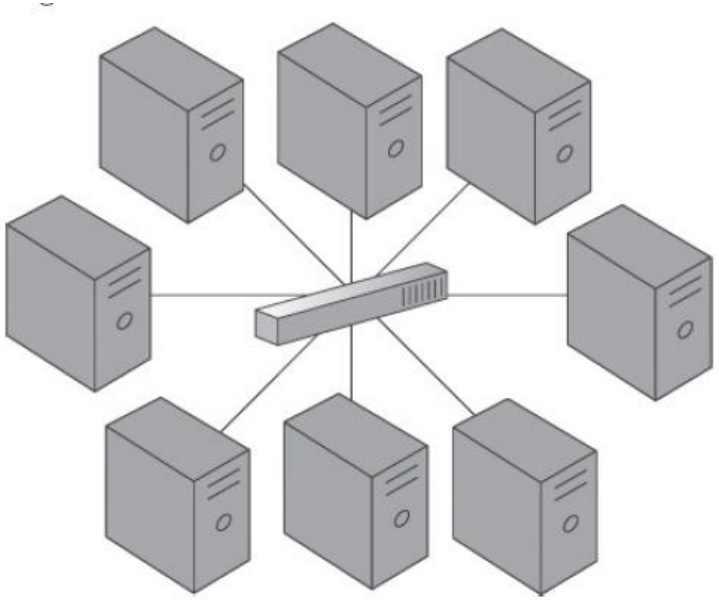
- **Horizontal Scaling/Scaling out:**
- Involves addition of more resources of the same type.
- More than one server works together (i.e. distributed systems).
- Integrates some overhead for coordination.

www.abhijitkakade.com



Clusters (Distributed Systems)

- In computing, a cluster is a tightly coupled collection of servers, or nodes. These servers usually have the same hardware specifications and are connected together via a network to work as a single unit.
- Each node in the cluster has its own dedicated resources, such as memory, a processor, and a hard drive.
- A cluster can execute a task by splitting it into small pieces and distributing their execution onto different computers that belong to the cluster.



Hadoop Distributed File Systems (HDFS)

- HDFS is a distributed file system (DFS) that runs on large clusters and provides high-throughput access to data and is designed to work with commodity hardware.
- HDFS is a highly fault-tolerant system HDFS stores each file as a sequence of blocks. The blocks of each file are replicated on multiple machines in a cluster to provide fault tolerance.
- **Scalable Storage for Large Files:** HDFS has been designed to store large files (typically from gigabytes to terabytes in size). Large files are broken into chunks or blocks and each block is replicated across multiple machines in the cluster. HDFS has been designed to scale to clusters comprising of thousands of nodes.
- **Replication:** HDFS replicates data blocks to multiple machines in a cluster which makes the system reliable and fault-tolerant. The default block size used is 128MB (64MB in some systems) and the default replication factor is 3.
- **Streaming Data Access:** HDFS has been designed for streaming data access patterns and provides high throughput streaming reads and writes. This design choice has been made to meet the requirements of applications that involve write-once, read many times data access patterns. HDFS is not suited for applications that require low-latency access to data. Instead, HDFS provides high throughput data access.
- **File Appends:** HDFS was originally designed to have immutable files. Files once written to HDFS could not be modified by writing at arbitrary locations in the file or appending to the file. Recent versions of HDFS have introduced the append capability.

Some HDFS Command Line Tools

```
■ #Copy file to HDFS
#Format of command:
hdfs dfs -put <local source> <destination on HDFS>

#Example:
hdfs dfs -put file /user/hadoop/file
```

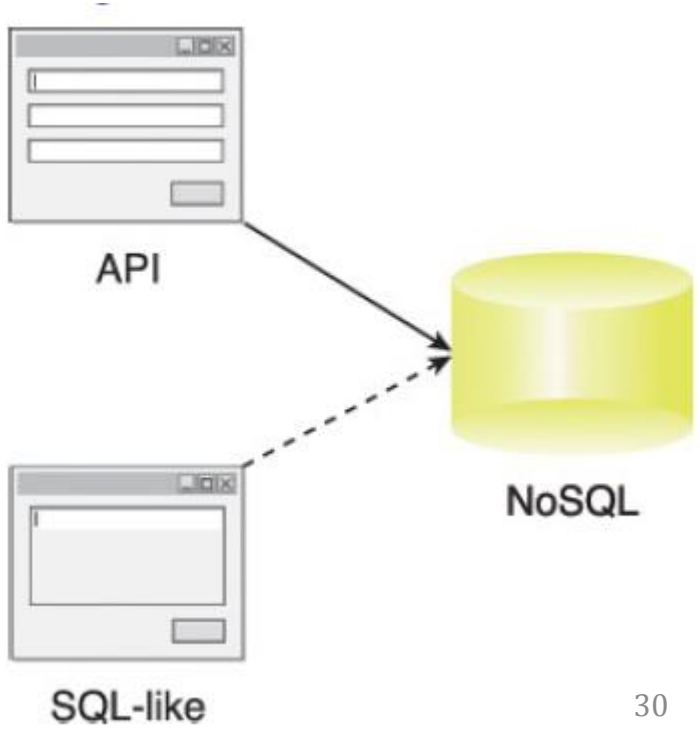
```
■ #Get file from HDFS
#Format of command:
hdfs dfs -get <source on hdfs> <local destination>

#Example:
hdfs dfs -get /user/hadoop/file file
```

```
■ #List files on HDFS
#Format of command:
hdfs dfs -ls <args>

#Example:
hdfs dfs -ls /user/hadoop/
```

```
■ #Show contents of a file on HDFS
#Format of command:
hdfs dfs -cat <HDFS Path>

#Example:
hdfs dfs -cat /user/hadoop/file
```

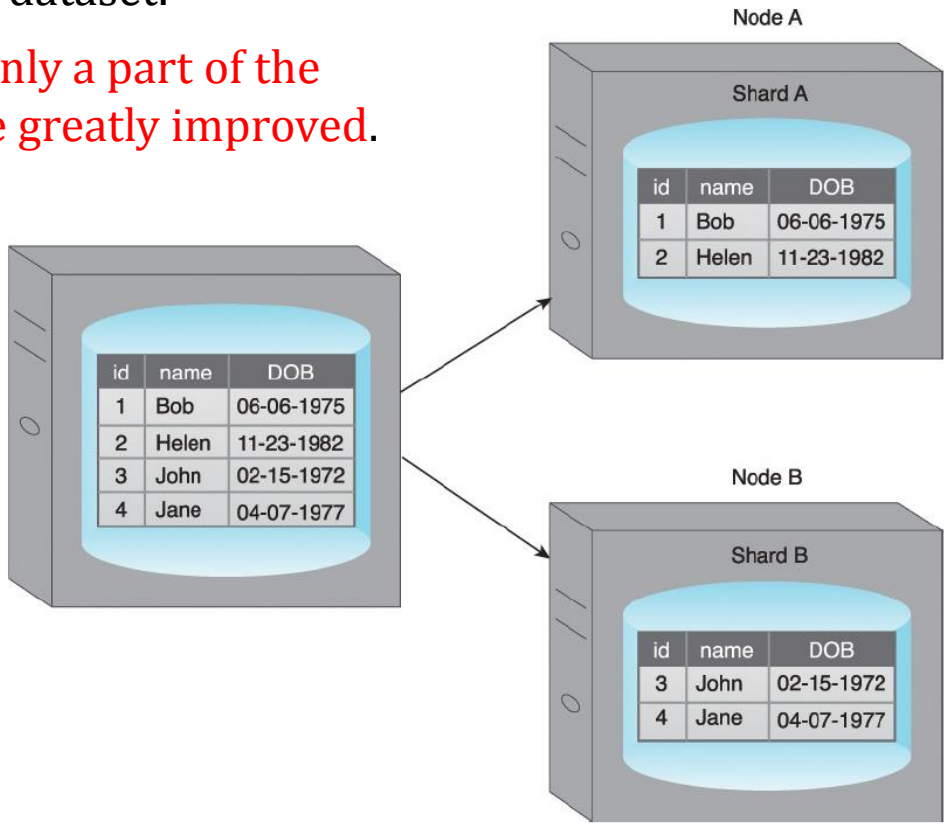
NoSQL

- A Not-only SQL (NoSQL) database is a non-relational database that is highly scalable, fault-tolerant and specifically designed to house semi-structured and unstructured data.
- A NoSQL database often provides an API-based (code functions) query interface that can be called from within an application.
- NoSQL databases are popular for applications that real-time performance is more important than consistency.
- There are some NoSQL databases that also provide an SQL-like query interface.



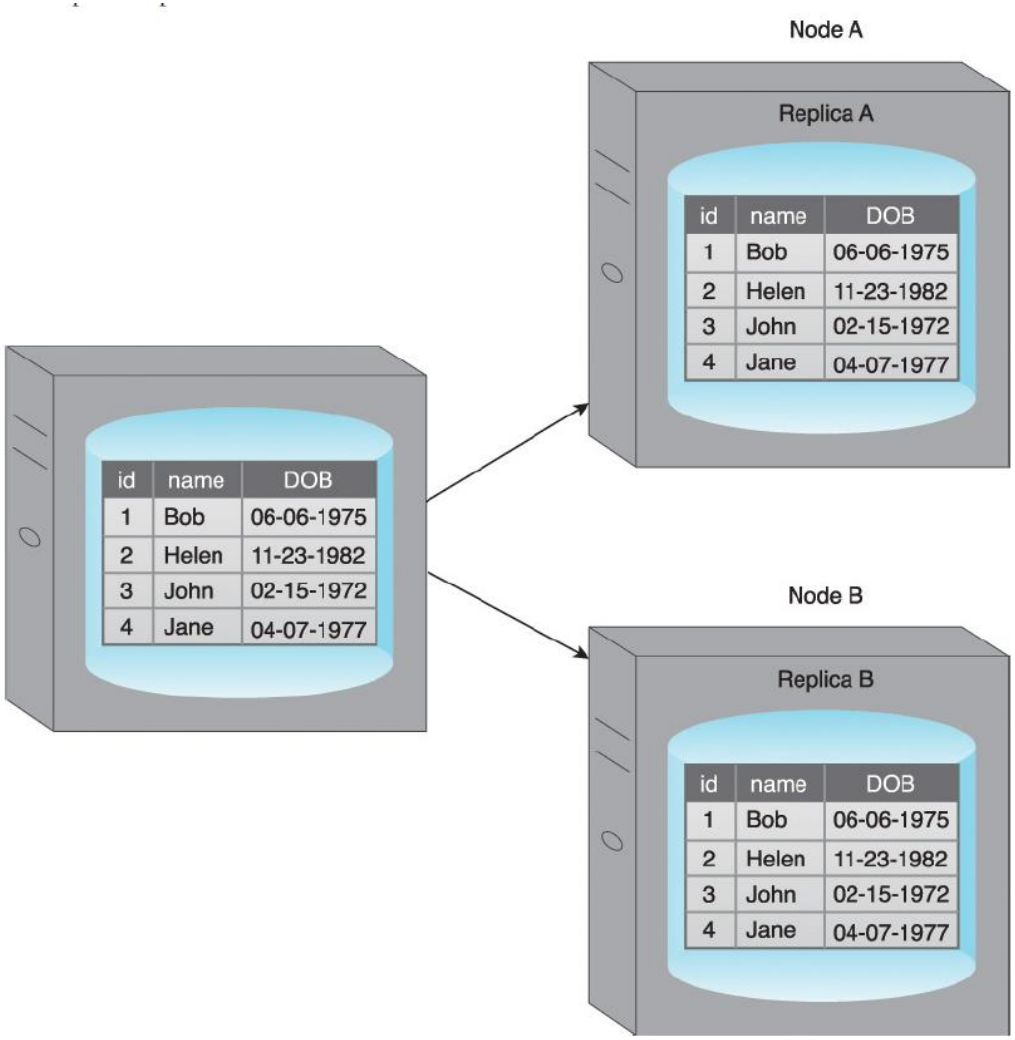
Sharding

- Sharding is the process of horizontally partitioning a large dataset into a collection of smaller, more manageable datasets called *shards*. The shards are distributed across multiple nodes, where a node is a server or a machine.
- Each shard is stored on a separate node and each node is responsible for only the data stored on it. Each shard shares the same schema, and all shards collectively represent the complete dataset.
- Since each node is responsible for only a part of the whole dataset, read/write times are greatly improved.



Replication

- Replication stores multiple copies of a dataset, known as replicas, on multiple nodes.
- Replication provides scalability and availability due to the fact that the same data is replicated on various nodes. Fault tolerance is also achieved since data redundancy ensures that data is not lost when an individual node fails.
- There are two different methods that are used to implement replication
  - master-slave
  - peer-to-peer



CAP Theorem

- The Consistency, Availability, and Partition tolerance (CAP) theorem, also known as Brewer's theorem, expresses a triple constraint related to distributed database systems.

- It states that a distributed database system, running on a cluster, can only provide two of the following three properties:

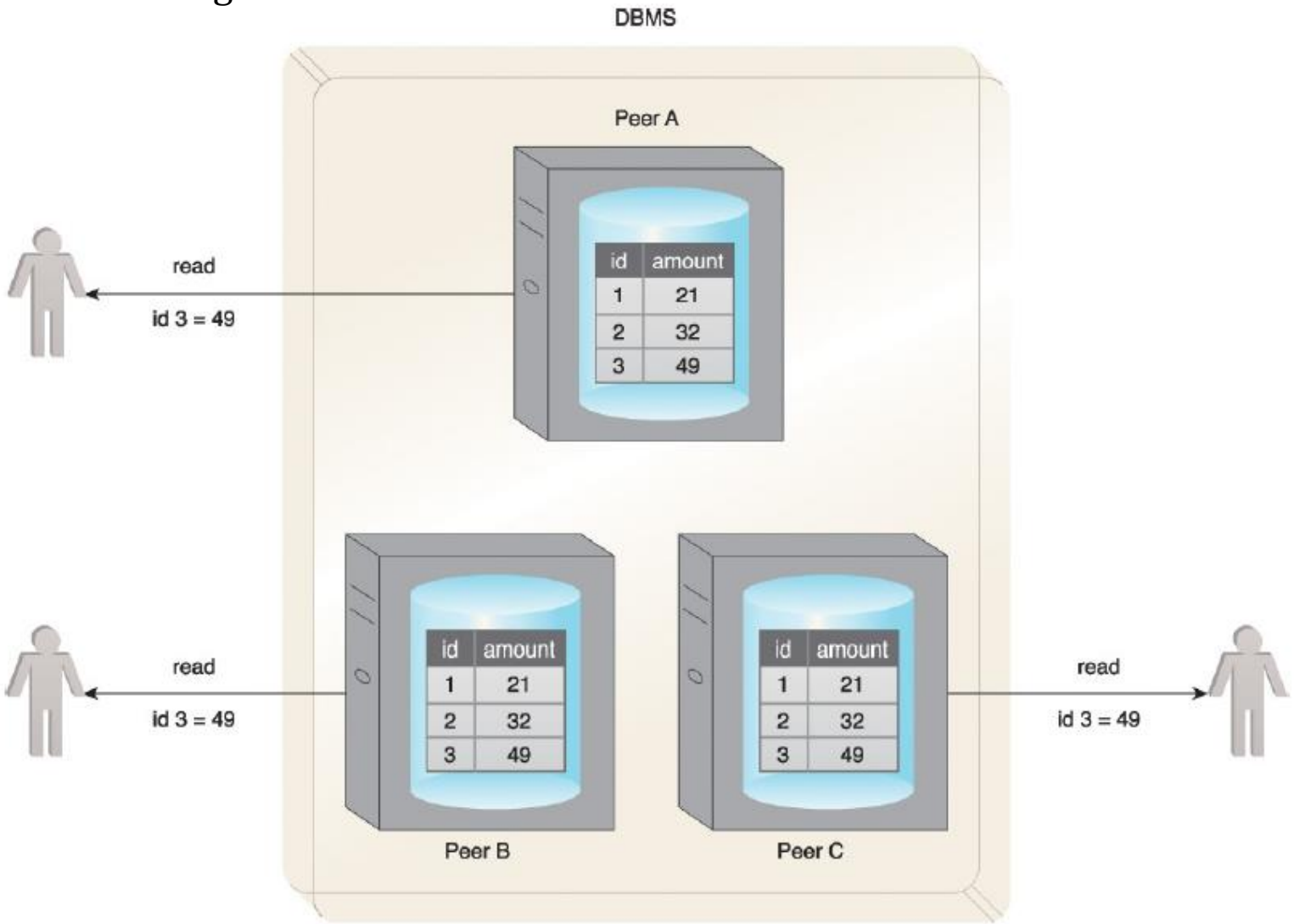
- Consistency

- Availability

- Partition tolerance

CAP - Consistency

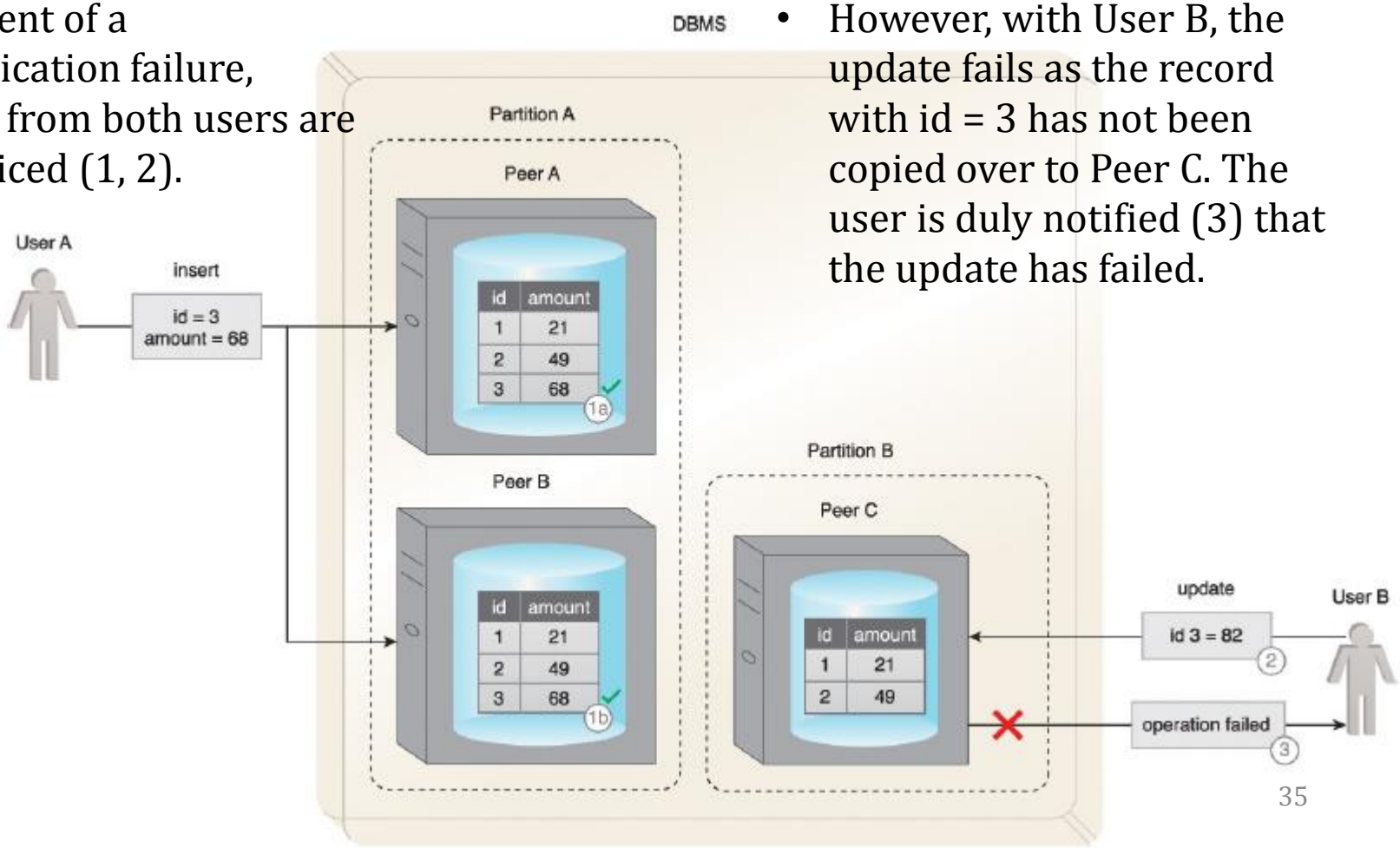
- ***Consistency*** – A read from any node results in the same data across multiple nodes.
- All three users get the same value for the amount column even though three different nodes are serving the record.



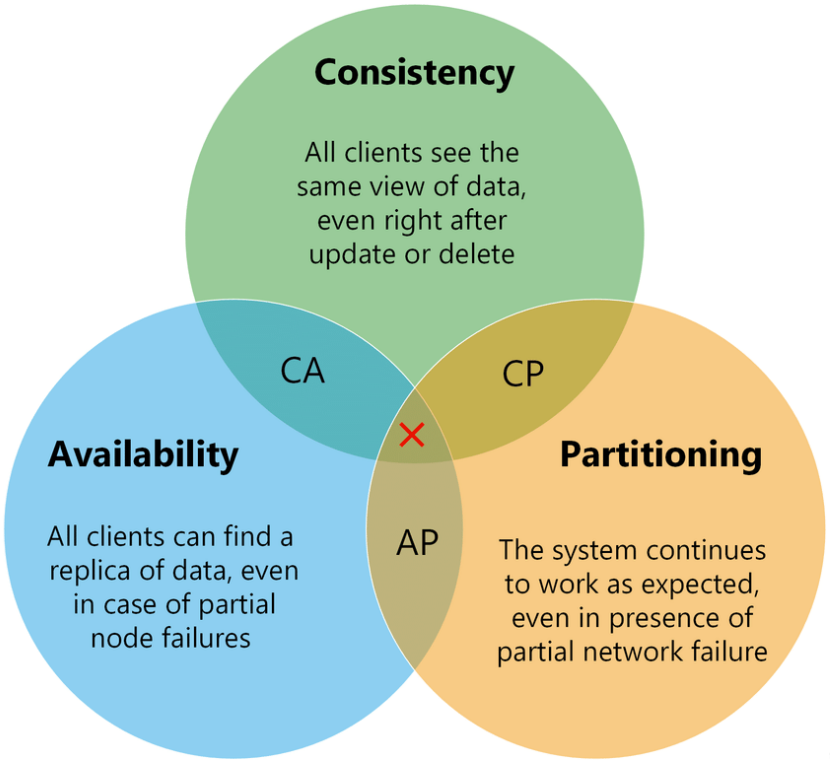
CAP – Availability and Partition Tolerance

- ***Availability*** – A read/write request will always be acknowledged in the form of a success or a failure.
- ***Partition tolerance*** – The database system can tolerate communication outages that split the cluster into multiple silos and can still service read/write requests

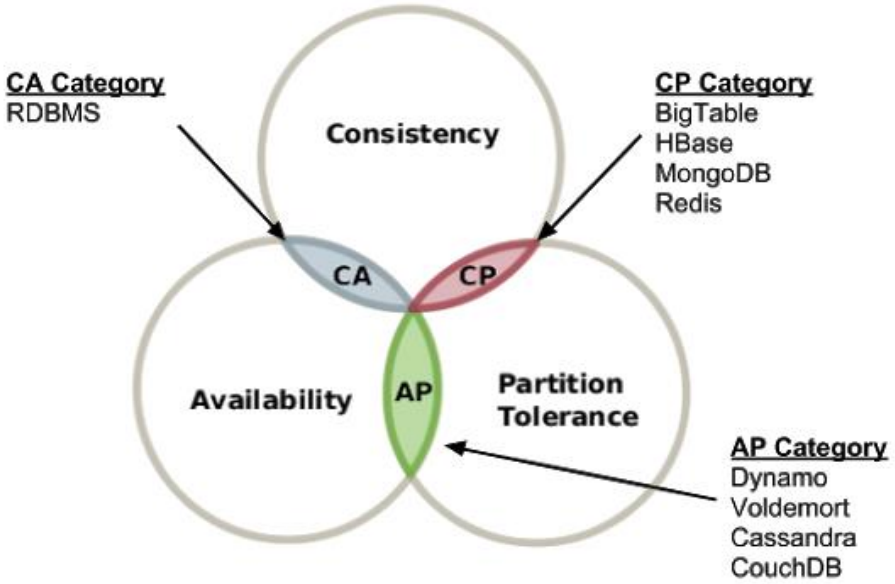
- In the event of a communication failure, requests from both users are still serviced (1, 2).
- However, with User B, the update fails as the record with id = 3 has not been copied over to Peer C. The user is duly notified (3) that the update has failed.



CAP – Two of the Three Properties

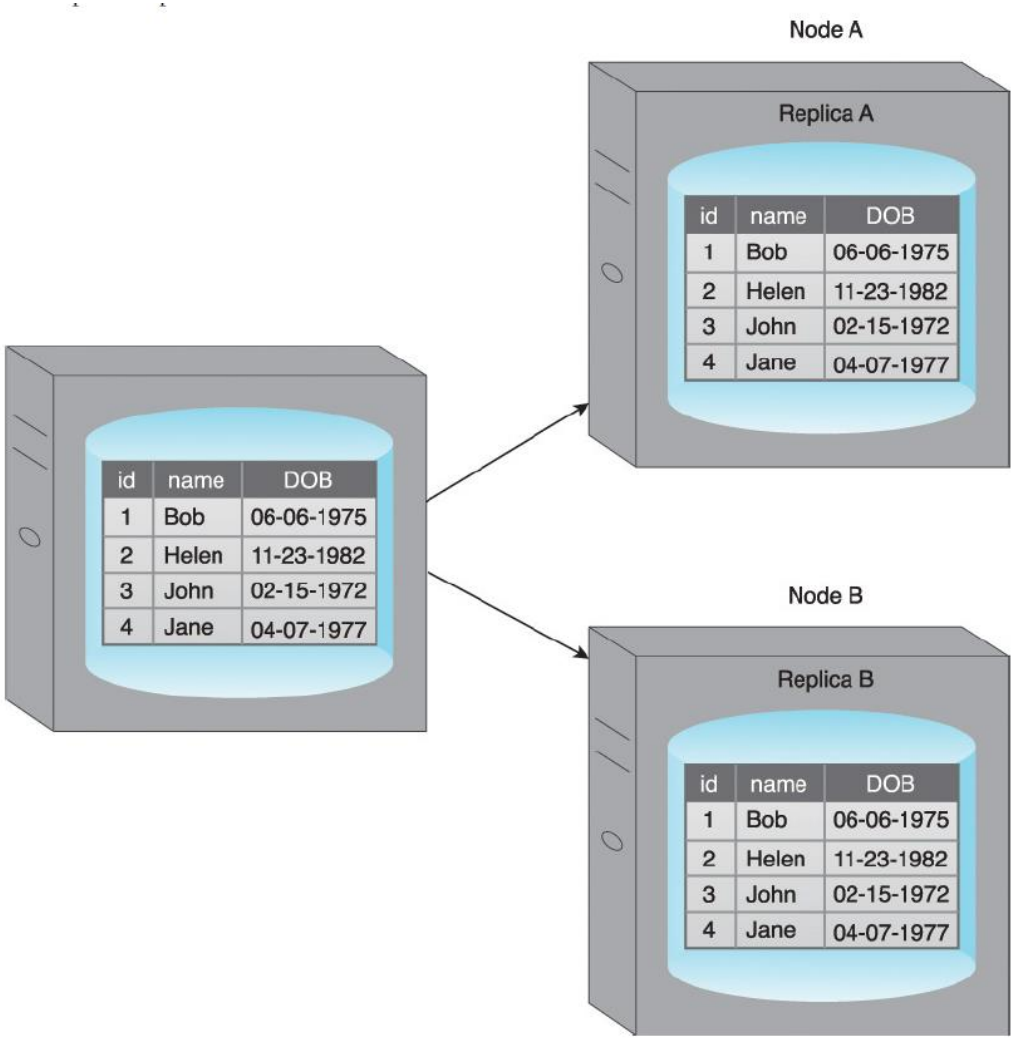


- It states that a distributed database system, running on a cluster, can only provide two of the following three properties.
- Although communication outages are rare and temporary, partition tolerance (P) must always be supported by a **distributed** database; therefore, CAP is generally a choice between choosing either C+P or A+P.
- The requirements of the system will dictate which is chosen.



Replication

- Replication stores multiple copies of a dataset, known as replicas, on multiple nodes.
- Replication provides scalability and availability due to the fact that the same data is replicated on various nodes. Fault tolerance is also achieved since data redundancy ensures that data is not lost when an individual node fails.
- There are two different methods that are used to implement replication
  - master-slave
  - peer-to-peer



ACID (RDBMS)

- ACID is a database design principle related to transaction management. It is an acronym that stands for:

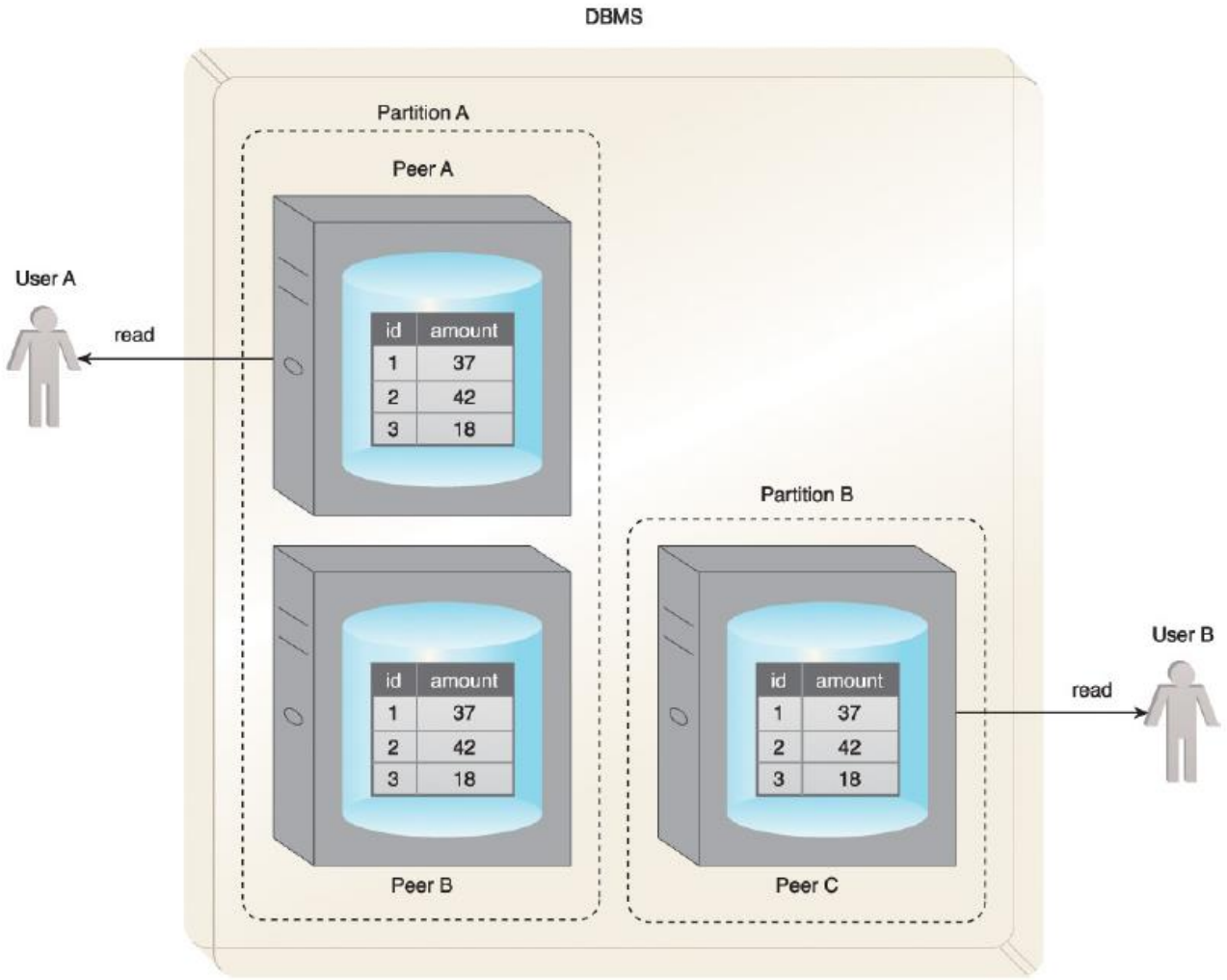
  - **Atomicity**: Atomicity ensures that all operations will always succeed or fail completely. In other words, there are no partial transactions.
  - **Consistency**: Consistency ensures that the database will always remain in a consistent state by ensuring that only data that conforms to the constraints of the database schema can be written to the database. Thus a database that is in a consistent state will remain in a consistent state following a successful transaction.
  - **Isolation**: Isolation ensures that the results of a transaction are not visible to other operations until it is complete.
  - **Durability**: Durability ensures that the results of an operation are permanent. In other words, once a transaction has been committed, it cannot be rolled back. This is irrespective of any system failure.

BASE
(Basically Available, Soft state, Eventual consistency)

- BASE is a database design principle based on the CAP theorem and leveraged by No-SQL database systems that use distributed technology. BASE stands for:
  - basically available
  - soft state
  - eventual consistency
- When a database supports BASE, it favours **availability** over **consistency**. In other words, the database is A+P from a CAP perspective.
- In essence, BASE leverages optimistic concurrency by relaxing the strong consistency constraints mandated by the ACID properties.
- Therefore, BASE-compliant databases are not useful for transactional systems where lack of consistency is a concern.

BASE - Basically Available

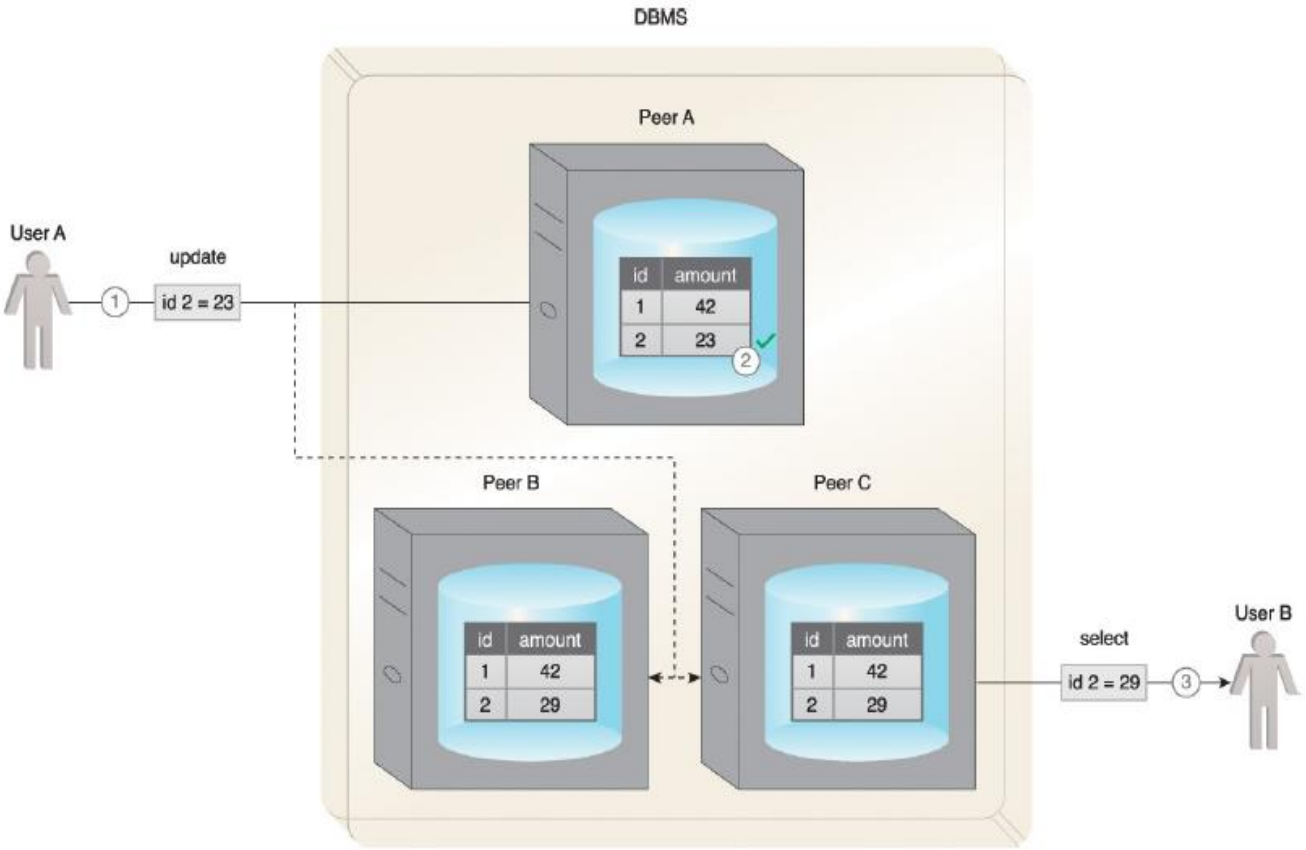
- If a database is "basically available," that database will always acknowledge a client's request (high availability, cluster-based technology), either in the form of the requested data or a success/failure notification.



User A and User B receive data despite the database being partitioned by a network failure.

BASE - Soft State

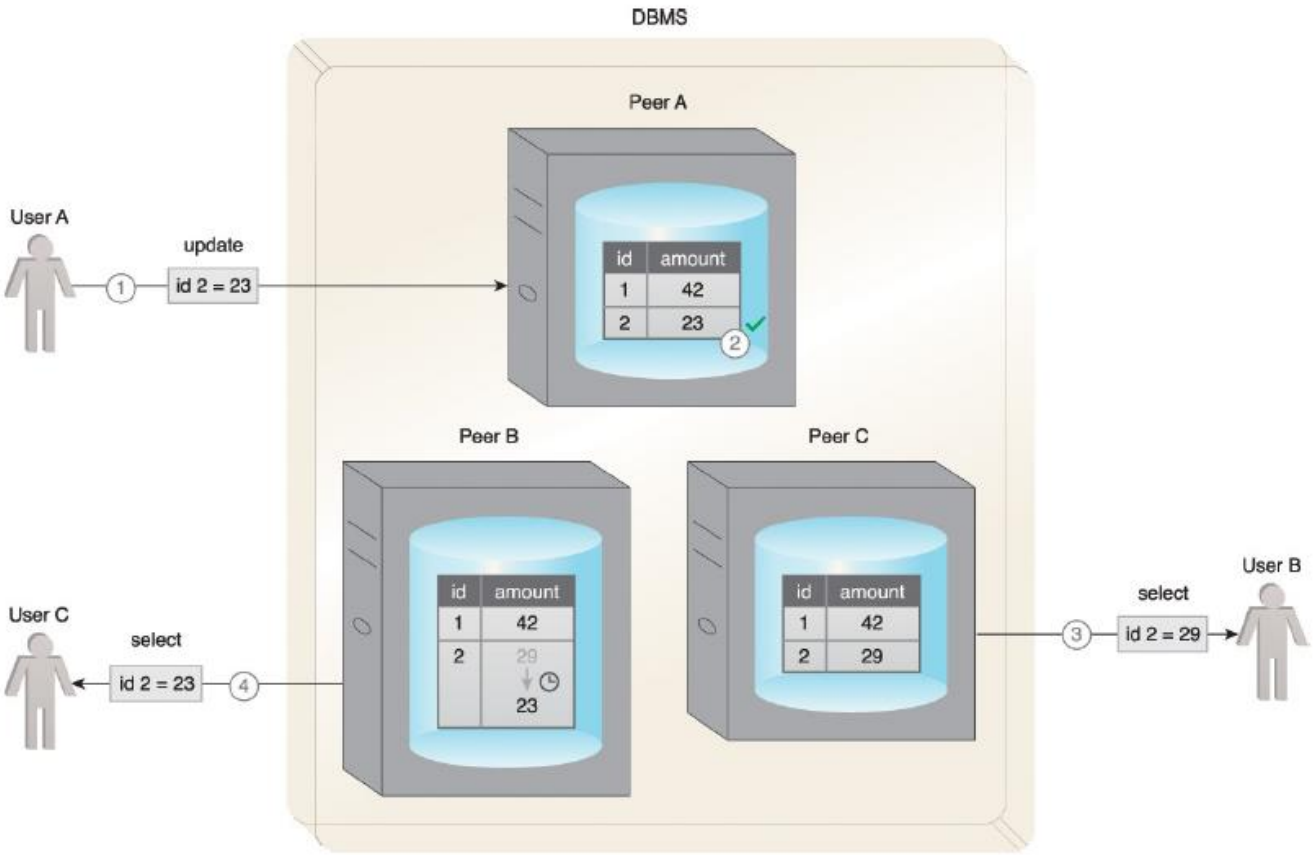
- Soft state means that a database may be in an inconsistent state when data is read; thus, the results may change if the same data is requested again.
- This is because the data could be updated for consistency, even though no user has written to the database between the two reads.
- This property is closely related to eventual consistency.



1. User A updates a record on Peer A.
2. Before the other peers are updated, User B requests the same record from Peer C.
3. The database is now in a soft state, and stale data is returned to User B.

BASE - Eventual Consistency

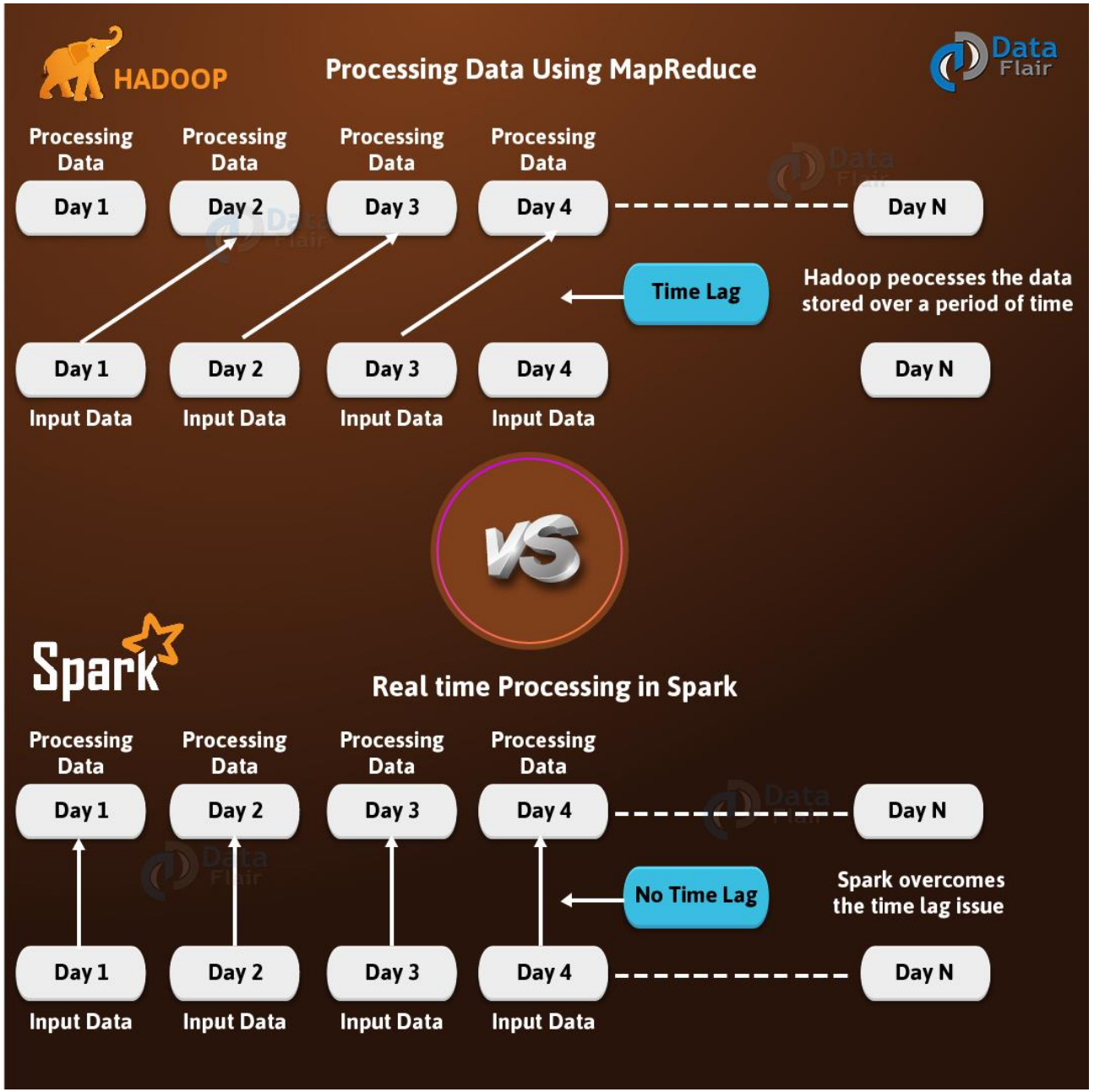
- Eventual consistency is the state in which reads by different clients, immediately following a write to the database, may not return consistent results.
- The database only attains consistency once the changes have been propagated to all nodes.
- While the database is in the process of attaining the state of eventual consistency, it will be in a soft state.



1. User A updates a record.
2. The record only gets updated at Peer A, but before the other peers can be updated, User B requests the same record.
3. The database is now in a soft state. Stale data is returned to User B from Peer C.
4. However, the consistency is eventually attained, and User C gets the correct value

Backup Slides (Optional)

Batch Process vs Real-Time



Batch Process vs Real-Time

- Batch Processing
  - An efficient way of processing high/large volumes of data is what you call Batch Processing. It is processed, especially where a group of transactions is collected over a period of time. In this process, At first, data is collected, entered and processed. Afterward, it produces batch results. For input, process, and output, batch processing requires separate programs. Payroll and billing systems are beautiful examples of batch processing.
  - Batch processing access to all data.
  - It might compute something big and complex.
  - Generally, it is very concerned with throughput. Rather than the latency of individual components of the computation.
  - Batch processing has latency measured in minutes or more.

Batch Process vs Real-Time

- Advantages of Batch Processing
  - Batch Processing is Ideal for processing large volumes of data/transaction. It also increases efficiency rather than processing each individually.
  - Here, we can do processing independently. Even during less-busy times or at a desired designated time.
  - For the organization by carrying out the process, it also offers cost efficiency.
  - Also, allows a good audit trail.

- Disadvantages of Batch Processing
  - The time delay between the collection of data and getting the result after the batch process.
  - In the batch processing master file is not always kept up to date.
  - Here, a one-time process can be very slow.

Batch Process vs Real-Time

- Real-time
  - Real-Time Processing involves continuous input, process, and output of data. Hence, it processes in a short period of time. There are some programs which use such data processing type. For example, bank ATMs, customer services, radar systems, and Point of Sale (POS) Systems.
  - Real-Time processing helps to compute a function of one data element. Also, can say it computes a smallish window of recent data.
  - Real-Time processing computes something relatively simple
  - While we need to compute in near-real-time, only seconds at most, we go for real-time processing.
  - In real-time processing, computations are generally independent.
  - They are asynchronous in nature. It means a source of data doesn't interact with the stream processing directly.

Batch Process vs Real-Time

- Advantages of Real-Time Processing
  - While performing real-time processing, there is no significant delay in response.
  - In real-time processing, information is always up to date. Hence, it makes the organization able to take immediate action. Also, when responding to an event, issue or scenario in the shortest possible span of time.
  - It also makes the organization able to gain insights from the updated data. Even helps to detect patterns of possible identification of either opportunities or threats.

- Disadvantages of Real-Time Processing
  - Real-Time processing is very complex as well as expensive processing.
  - Also turns out to be very difficult for auditing.
  - Real-Time processing is a bit tedious processing.

Sharding – Join (an example)

- The following steps are shown in Figure :

1. A user requests multiple records (id = 1, 3) and the application logic is used to determine which shards need to be read.
2. It is determined by the application logic that both Shard A and Shard B need to be read.
3. The data is read and joined by the application.
4. Finally, the data is returned to the user.