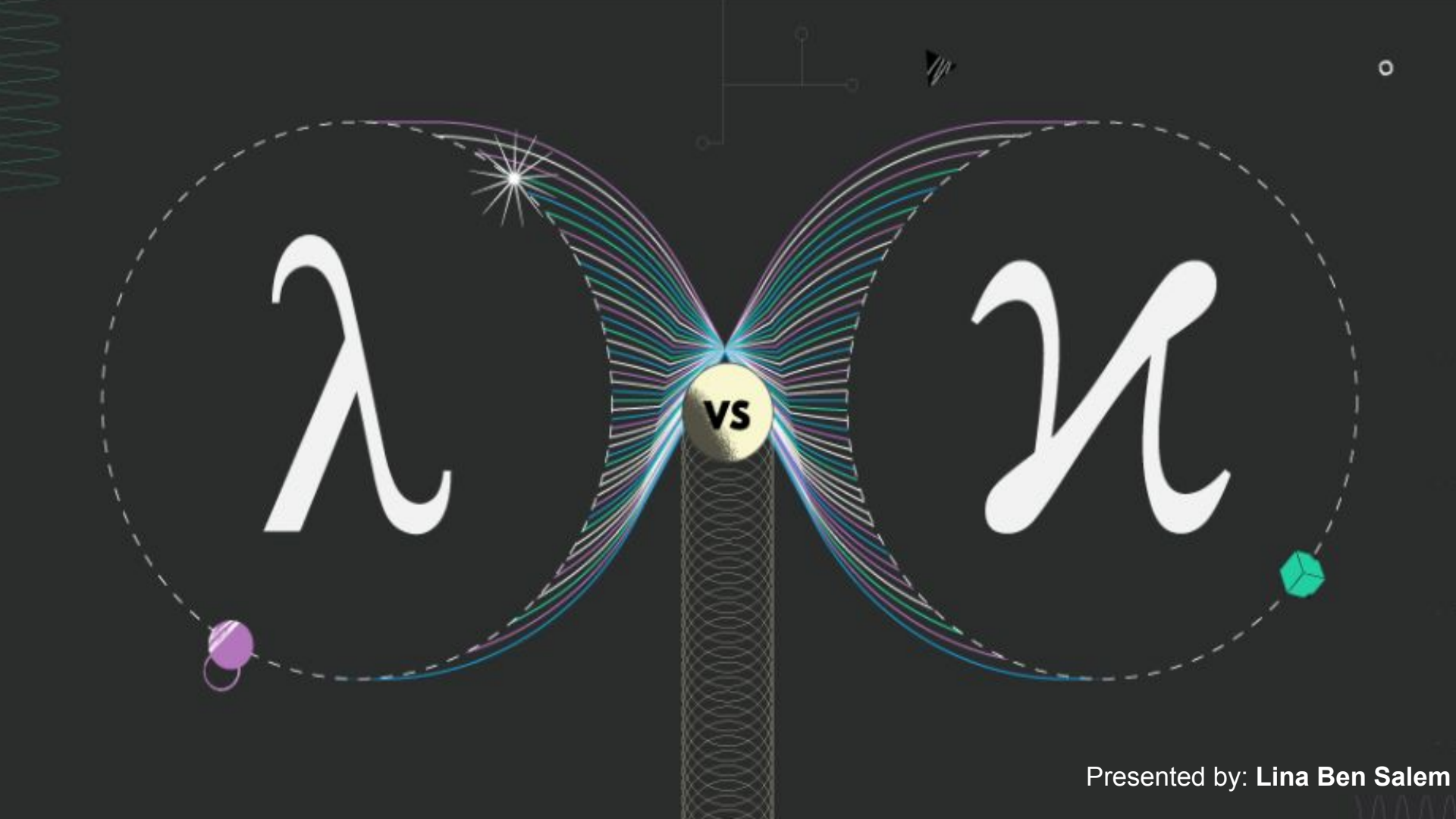


Presented by: **Lina Ben Salem**

Plan

1. Introduction
2. λ Architecture
3. K Architecture
4. λ vs K Architecture
5. Conclusion

1. Introduction

- **Data processing :**

  is the conversion of raw data into **meaningful information**

  is fundamental to unlocking the **value** of data and realizing its full potential to drive business success, enhance operational efficiency, improve customer experiences, manage risks effectively, and fuel growth and innovation

1. Introduction

- Big data **blooming** because of the rapid growth of social media applications, cloud based systems, Internet of things and an unending spree of innovations

  => It became essential to take well calculated decisions while launching, upgrading or troubleshooting an enterprise application and to find the **best manners to deal with it**

1. Introduction

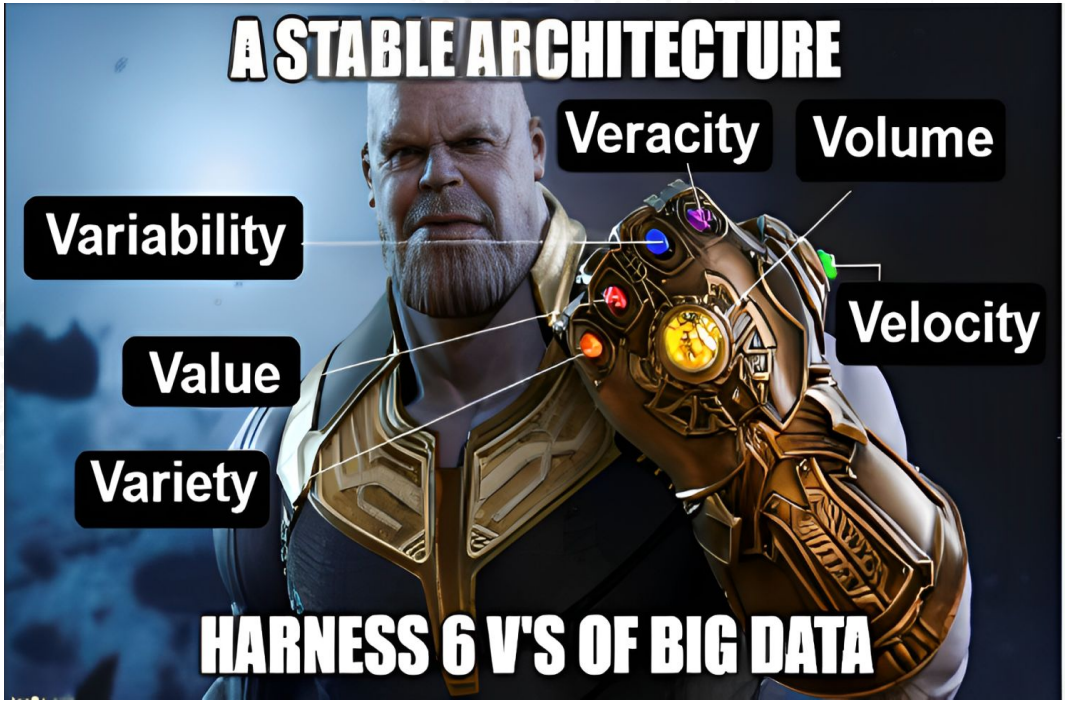
- The old ways of processing data :

  1. Batch processing
  2. Data warehousing
  3. ETL
  4. Message queueing system

Traditional approaches to data processing were often **limited** in their ability to **handle the volume, velocity, and variety of big data in real-time or near-real-time**

1. Introduction

We need systems that are being built on a well-designed big data **architecture** to handle the the **6 V's** of Big Data **Volume, Velocity, Variety, Veracity, Validity, and Volatility** while keeping the gain of new insights and make better business decision



2. λ Architecture

Design overview

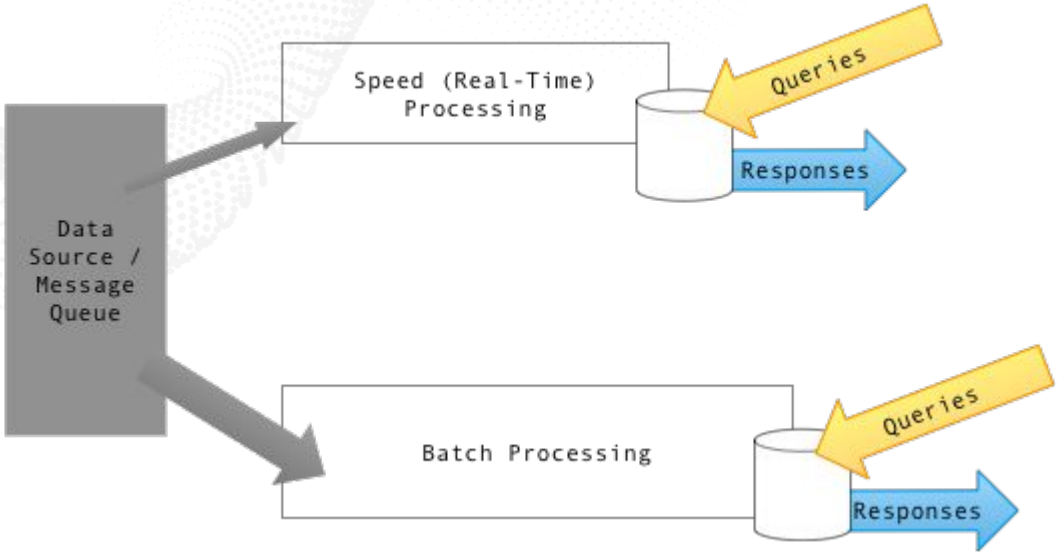
Lambda architecture is a data processing architecture that aims to provide a scalable, fault-tolerant, and flexible system for processing large amounts of data.

It was developed by Nathan Marz in 2011 as a solution to the challenges of **processing data in real time**

2. λ Architecture

It uses two separate data processing systems to handle different types of data processing workloads:

- The first system is a **batch processing system**, which processes data in large batches and stores the results in a centralized data store, such as a data warehouse or a distributed file system
- The second system is a **stream processing system**, which processes data in real-time as it arrives and stores the results in a distributed data store



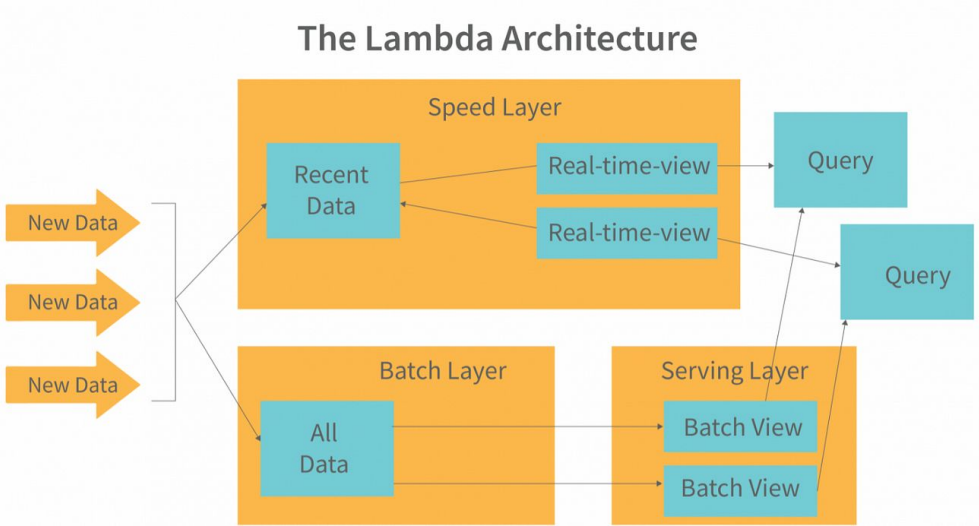
2. λ Architecture

Components:

Batch Layer: This layer manages the historical data and performs batch processing on it. It stores immutable, raw data and computes batch views on top of this data. Technologies like Apache Hadoop, Spark, or Flink are often used in this layer.

Speed Layer: This layer deals with real-time data processing. It processes incoming data in real-time and provides real-time views. Technologies like Apache Storm, Apache Samza, or Apache Kafka Streams are commonly used in this layer.

Serving Layer: Also known as the Query Layer, this layer serves queries by combining results from the batch and speed layers. It ensures that both historical and real-time data are integrated seamlessly.
Technologies: Apache HBase or Apache Cassandra are often used in this layer.



2. λ Architecture

Advantages:

Scalability:
Lambda architecture is designed to handle large volumes of data and scale horizontally to meet the needs of the business

Fault-tolerance:
Lambda architecture is designed to be fault-tolerant, with multiple layers and systems working together to ensure that data is processed and stored reliably

Flexibility:
Lambda architecture is flexible and can handle a wide range of data processing workloads, from historical batch processing to streaming architecture

2. λ Architecture

Disadvantages

Complexity:
Lambda architecture is a complex system that uses multiple layers and systems to process and store data. It can be challenging to set up and maintain

Errors and data discrepancies:
With doubled implementations of different workflows ,you may run into a problem of different results from batch and stream processing engines. Hard to find, hard to debug.

Architecture lock-in:
It may be super hard to reorganize or migrate existing data stored in the Lambda architecture.

2. λ Architecture

Use Cases:

Lambda architecture is useful for handling **large volumes of data and providing low-latency query results**

- **Real-time analytics applications**: such as dashboards and reporting
- **Batch processing tasks**: such as data cleansing, transformation, and aggregation
- **Stream processing tasks:** such as event processing, machine learning models, anomaly detection, and fraud detection
- **Build data lakes:** which are centralized repositories that store structured and unstructured data at rest
- **Handling the high-volume data streams** generated by IoT devices

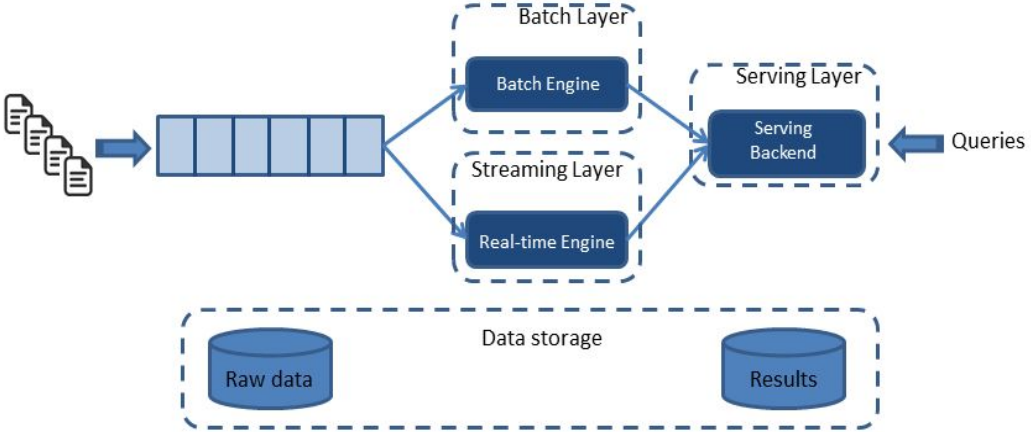
3. K Architecture

Design overview

Kappa architecture is a data processing architecture that is designed to provide a scalable, fault-tolerant, and flexible system for processing large amounts of data in real time.

It was developed as an alternative to Lambda architecture to handle different types of data processing workloads by using a single data processing system to handle both batch processing and stream processing workloads, as it treats everything as streams.

This allows it to provide a more streamlined and simplified data processing pipeline while still providing fast and reliable access to query results.



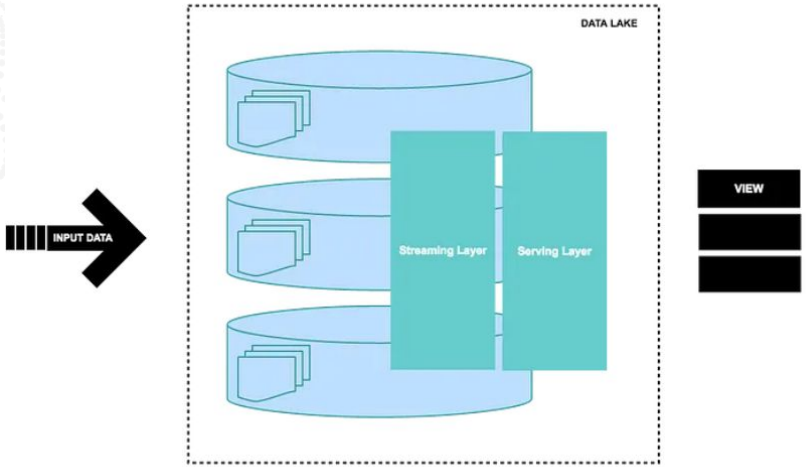
1

3. K Architecture

Components:

Stream Layer: This layer processes data as an unbounded stream, handling both real-time and batch processing tasks. It eliminates the need for separate batch and real-time processing paths.

Serving Layer: Similar to Lambda Architecture, the serving layer provides the interface for querying the processed data.



Kappa Architecture

3. K Architecture

Advantages:

Simplicity and streamlined pipeline:
Kappa architecture uses a single data processing system to handle both batch processing and stream processing workloads, which makes it simpler to set up and maintain compared to Lambda architecture. This can make it easier to manage and optimize the data processing pipeline by reducing the coding overhead.

Enables high-throughput big data processing of historical data:
Kappa architecture can support this, enabling reprocessing directly from our stream processing job.

Ease of migrations and reorganizations:
As there is only stream processing pipeline, you can perform migrations and reorganizations with new data streams created from the canonical data store.

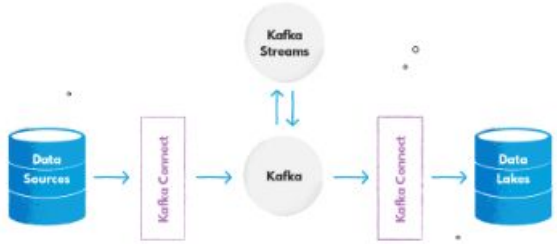
Tiered storage:
Tiered storage is a method of storing data in different storage tiers, based on the access patterns and performance requirements of the data. The idea behind tiered storage is to optimize storage costs and performance by storing different types of data on the most appropriate storage tier. It makes it a cost-efficient and elastic data processing technique without the need for a traditional data lake.

3. K Architecture

Disadvantages:

Complexity:
While Kappa architecture is simpler than Lambda, it can still be complex to set up and maintain, especially for businesses that are not familiar with stream processing frameworks (review common challenges in stream processing)

Costly infrastructure with scalability issues (when not set properly):
Storing big data in an event streaming platform can be costly. To make it more cost-efficient you may want to use data lake approach from your cloud provider (like AWS S3 or GCP Google Cloud Storage).
Another common approach for big data architecture is building a “streaming data lake” with Apache Kafka as a streaming layer and object storage to enable long-term data storage.



Streaming architecture based on Kafka for Kappa approach - Kafka message flow through components

3. K Architecture

Use Cases:

Kappa architecture is a data processing architecture that is designed to provide a flexible, fault-tolerant, and scalable architecture for processing large amounts of data in real-time. It is well-suited for a wide range of data processing workloads, including continuous data pipelines, real time data processing, machine learning models and real-time data analytics, IoT systems, and many other use cases with a single technology stack.

4. λ vs K Architecture

Lambda Architecture	Kappa Architecture
Designed to handle low-latency reads and updates in a linear scalable and fault-tolerant way	Single stream processing engine handles both real-time data processing and continuous reprocessing
3 layer architecture - Batch,Speed and Severing layer	2 layer architecture - Real-time layer and Serving layer
The input process can be re-triggered in case of any code/metric changes	Re-triggered from the latest real-time engine and replace the data stored in the serving layers
Different code bases for Batch and Speed layers	Single code base
Batch layer - Hadoop MapReduce Speed layer- Apache Storm, Apache Samza, and Spark Streaming Apache Spark,Apache Flink- can be used across Batch and Speed layers	Apache Storm, Apache Samza, and Spark Streaming

5. Conclusion

Zeta (ζ) Architecture

Thank you!