Big Data Technologies and Architectures

@Tekna --- May 25-26

Marc Gallofré Ocaña

marcgallofre@gmail.com


/marcgallofre

Ga11u 


Overview of NoSQL databases and their applications in handling big data for AI

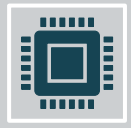
Agenda

- Overview of NoSQL
- Types of NoSQL Databases
- Use Cases and Implementation
- Operations and Optimisation

What are NoSQL Databases?

NoSQL databases provide flexible and/or scalable solutions for handling big data.

Some have a schema-less structure, allowing for agile development and adaptation to evolving data models.



Many NoSQL databases offer horisontal scalability to handle increasing data volumes and high traffic loads.

Support diverse data types enabling efficient storage and processing of unstructured and semi-structured data.

What are NoSQL Databases?

NoSQL databases offer high availability and fault tolerance through data replication.

They can excel in handling big data for AI, IoT, and real-time data processing.

NoSQL databases provide flexibility for accommodating data growth and handling complex structures without predefined schemas.

Their scalability and flexibility can be crucial for effectively managing and analysing large volumes of data.

Comparison: SQL vs. NoSQL

- SQL databases use structured, tabular models with predefined schemas
- NoSQL databases can be non-relational DB or non-SQL.

- SQL databases utilise SQL for querying and managing data
- NoSQL databases utilise various query languages or APIs.

- SQL databases prioritise data consistency.
- NoSQL databases prioritise scalability and availability.

Comparison: SQL vs. NoSQL

- SQL databases have a mature ecosystem
- NoSQL databases offer specialised solutions for specific use cases.

Choosing between SQL and NoSQL depends on factors like data structure, scalability, complexity, and application requirements.

Understanding the strengths and limitations of each approach is crucial for selecting the appropriate database solution.

Big Data and Unstructured Data

- SQL databases face challenges with big data due to rigid schemas and structures.
- Unstructured or semi-structured data doesn't fit well into SQL's tabular structure.
- SQL databases may struggle with scalability and performance in handling large volumes and high-speed data.

Big Data and Unstructured Data

- NoSQL databases can provide horizontal scalability for distributed storage and processing of big data workloads.
- NoSQL databases can be good in high-speed data ingestion and horizontal scalability to accommodate growing data volumes.

Types of NoSQL Databases

Document databases (e.g., MongoDB):	• Flexible JSON-like document storage
Key-value stores (e.g., Redis):	• Fast retrieval and caching with key-value pairs
Column-family databases (e.g., Cassandra):	• Efficient storage and retrieval of large datasets
Time-series databases (e.g., InfluxDB):	• Specialised for time-stamped data analysis
Knowledge graphs (e.g., Neo4j):	• Modelling complex interconnected data with nodes and relationships
Vector databases (e.g., Vald)	• Storing vector representations

Document Databases

Store data in flexible and self-describing documents (e.g.,JSON, BSON).

Advantages of document databases:

- Flexible schema to accommodate varying structures and fields.
- Scalability to handle large amounts of unstructured or semi-structured data.
- The data model is intuitive and closer to the way developers work.
- Documents can map to objects like in object-oriented programing.

Document Databases

Examples of popular document databases:

- MongoDB, known for its flexibility and scalability.
- Couchbase, offering high-performance and distributed architecture.
- Amazon DocumentDB, a fully managed document database service.

Key-Value Stores

- Key-value stores store data as key-value pairs.
- Advantages of key-value stores:
  - High-performance data retrieval and storage.
  - Scalability for large data volumes and high traffic.
  - Flexibility to store any type of data without a fixed schema.
- Key-value stores are suitable for:
  - Caching and high-speed data access.
  - Session management and user profiles.
  - Message queues and task management.
  - Distributed systems and distributed caching.

Key-Value Stores

- Examples of popular key-value stores:
  - Amazon DynamoDB: Fully managed NoSQL database service.
  - Memcached: Widely used caching system for web applications.
  - Riak: Distributed and highly available key-value storage.

Column-Family Databases

Column-family databases organise data in columns and column families.

Advantages of column-family databases:

Efficient storage and retrieval of large datasets.	Scalability for high write throughput and massive data volumes.	**Flexibility in adding or modifying columns without impacting other data.**

Column-family databases are suitable for:

Time-series data, event logging, and sensor data.	Social media analytics and recommendation systems.	Data warehousing and business intelligence.	Internet of Things (IoT) data storage and analysis.

Column-Family Databases

Examples of popular column-family databases:			
Apache Kudu: Build for the Hadoop ecosystem.	ClickHouse: designed for OLAP operations	Apache Druid: Designed large ingestion and analysis.	SAP Hanna

Time-Series Databases

Time-series databases efficiently store, analyse, and retrieve time-stamped data.

Advantages of time-series databases:

- Optimised for large volumes of time-series data.
- Fast querying with specialised storage and indexing mechanisms.
- Support for advanced time-based analytics and real-time monitoring.

Time-series databases are suitable for:

- IoT sensor data monitoring and analysis.
- Financial market data analysis and forecasting.
- Log and event data analysis.
- Performance monitoring and anomaly detection.

Time-Series Databases

Examples of popular time-series databases:

- InfluxDB: Scalable with high write and query performance.
- TimescaleDB: Open-source, built on PostgreSQL.
- Prometheus: Monitoring and alerting toolkit with a time-series database.
- Graphite: Scalable time-series database and visualisation tool.
- MongoDB

Knowledge Graphs

Knowledge graphs represent information as interconnected entities and relationships.

Advantages of knowledge graphs:

- Capture complex relationships and semantic context.
- Enable powerful semantic querying and inference.
- Facilitate knowledge discovery and reasoning.

Knowledge graphs are used in various domains:

- Semantic search and recommendation systems.
- Data integration and knowledge management.
- Biomedical research and drug discovery.

Knowledge Graphs

Examples of popular knowledge graph databases:

- Neo4j: High-performance, graph-based database.
- Stardog: RDF-based knowledge graph platform.
- Amazon Neptune: Fully managed graph database service.
- Virtuoso: Hybrid database for relational and graph data.
- Apache Jena: RDF-store

Vector Databases

- Vector databases are designed specifically for storing and retrieving machine learning vector data efficiently.
- They offer fast storage and retrieval capabilities for high-dimensional vectors.
- Examples of vector databases include Vald, Milvus, and Weaviate.
- Vector databases are used in various applications such as image recognition, natural language processing, and recommendation systems.

Use Cases of NoSQL Databases

Real-world examples of NoSQL databases for AI and big data:

- Netflix: Personalised recommendations and processing users' data (Apache Cassandra)
- Uber: Real-time data processing for route optimisation and pricing (MongoDB)
- Twitter: Real-time analytics and sentiment analysis (Apache Cassandra)

Benefits and outcomes of NoSQL databases in AI:

- Enhanced user experiences and real-time decision-making.
- Improved operational efficiency and scalability.
- Accelerated data analysis and valuable insights.
- Efficient handling of diverse unstructured data types.
- Seamless integration of AI models and algorithms.

Security Considerations for NoSQL Databases

Security considerations for NoSQL databases:	Authentication:	Authorisation:
• Authentication • Authorisation • Encryption • Auditing	• User authentication mechanisms • Secure access control	• Role-based access control • Fine-grained access permissions

Discussion: Use Cases and Challenges

- In groups, share uses of NoSQL DBs in your organisation or based on your experience.
- Think where you could use any of the NoSQL DBs
- Share

Hands-on tutorial on NoSQL databases

Introduction to Apache Cassandra

- Highly scalable and distributed NoSQL database
- Handles massive amounts of data with high availability and fault tolerance
- Open-source distributed NoSQL database
- Designed for handling large-scale data across multiple commodity servers

Key Features of Apache Cassandra

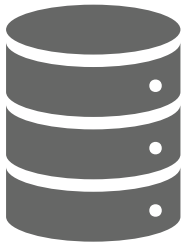
- Distributed architecture for horizontal scalability
- No single point of failure
- High availability and fault tolerance
- Linear scalability with added nodes
- Tunable consistency levels
- Peer-to-peer node communication for fault tolerance
- Built-in replication and data distribution across multiple data centres

Data Model in Apache Cassandra

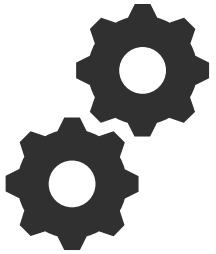
- Row-based column-partitioned data model
- Tables organised by partition keys and clustering columns
- Query-oriented design
- Optimised for write-heavy workloads

Data Replication in Apache Cassandra

Replication factor determines the number of copies of data stored



Replication strategy defines how data is distributed across nodes



Ensures data availability and fault tolerance in case of node failures

Apache Cassandra Architecture

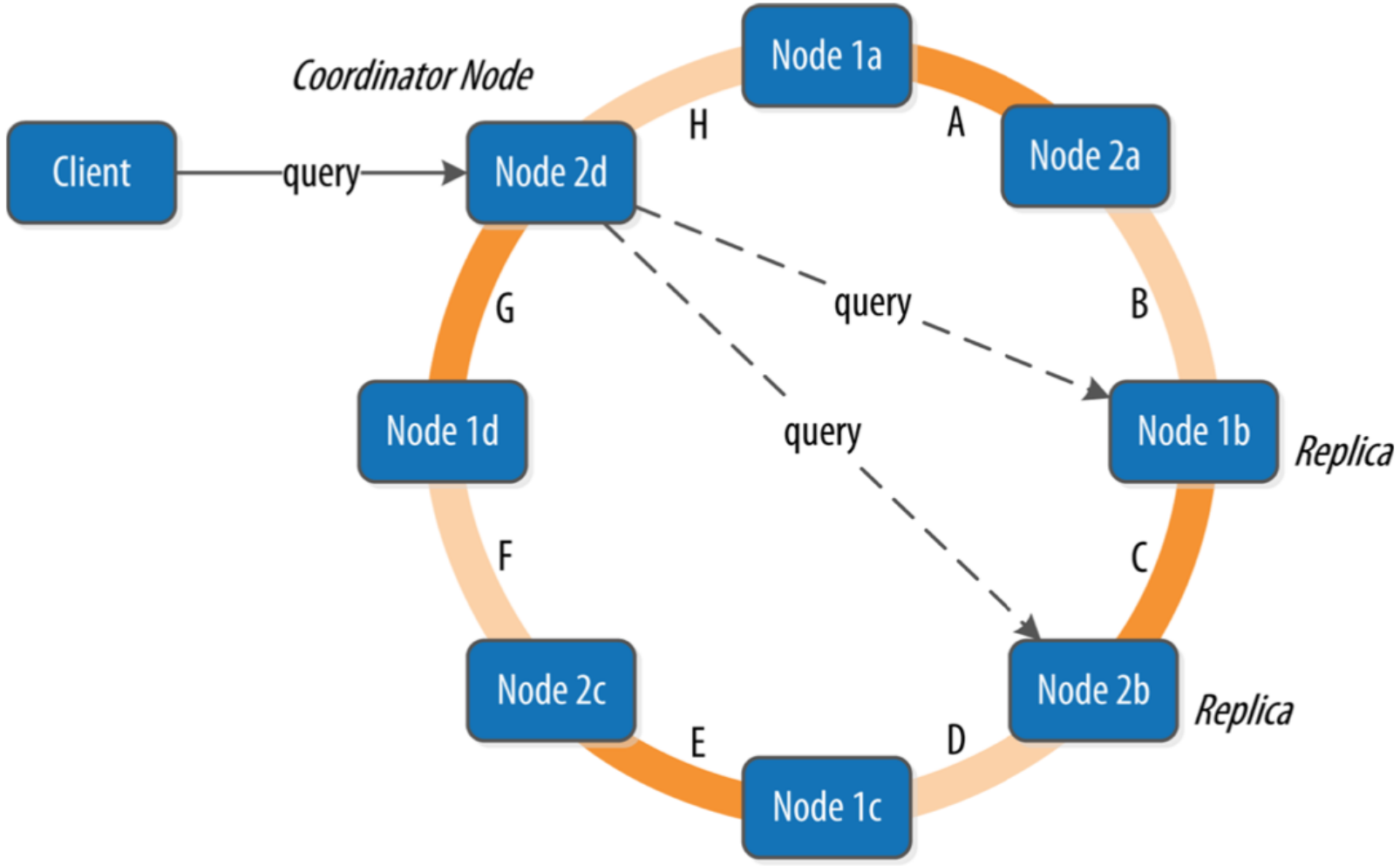
Peer-to-peer architecture with no single point of failure

Node coordination through the gossip protocol

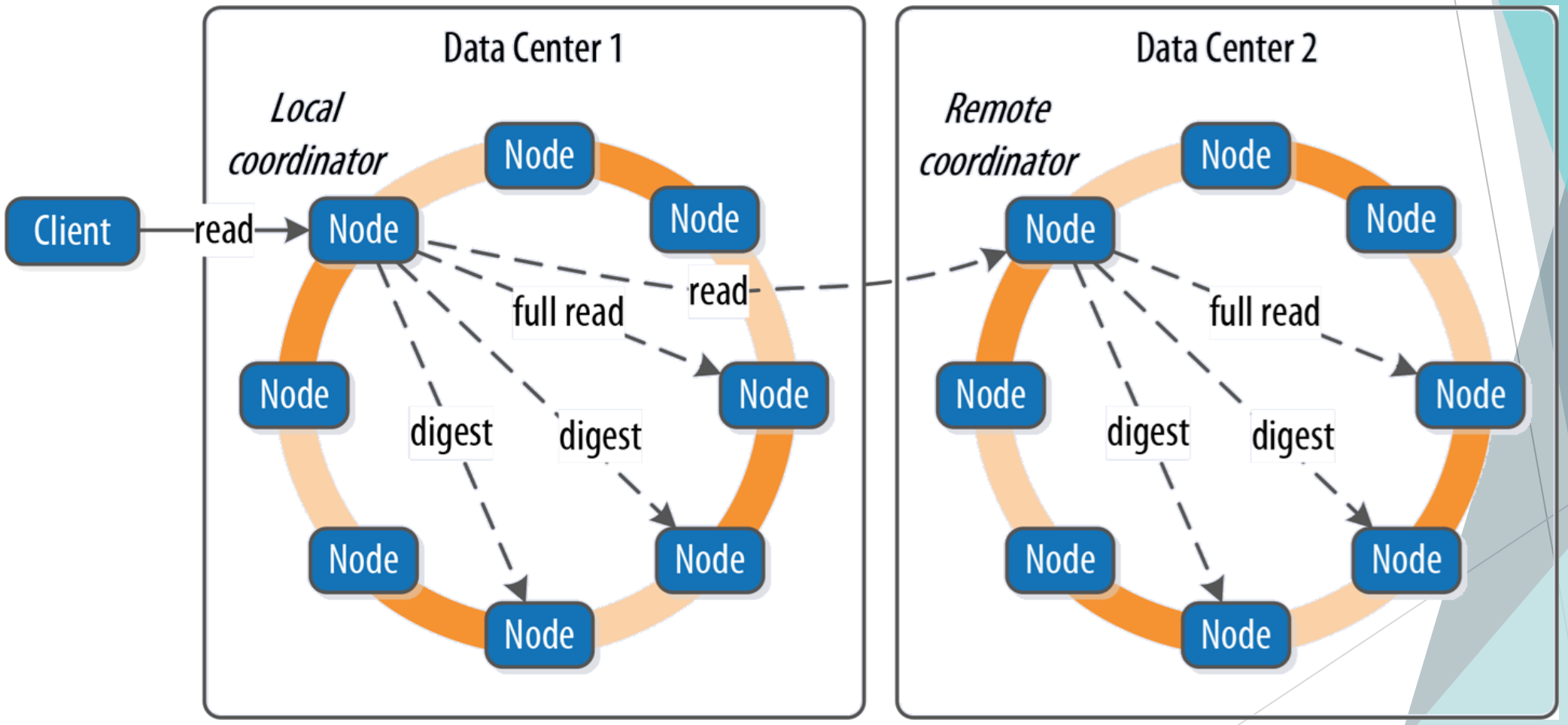
Data distribution using consistent hashing

Replication and data consistency mechanisms

Apache Cassandra Architecture

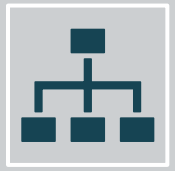


Apache Cassandra Architecture



Consistency Levels

Cassandra offers consistency levels for read and write operations



Consistency levels:

ONE, QUORUM, ALL, LOCAL_QUORUM, etc.

Cassandra Query Language (CQL)

SQL-like language for interacting with Cassandra

Supports create, read, update, and delete operations

Schema definition, data querying, and indexing capabilities

Use Cases for Apache Cassandra

TIME SERIES DATA:

IOT SENSOR DATA, FINANCIAL MARKET DATA

E-COMMERCE:

INVENTORY MANGEMENT

CUSTOMER DATA MANAGEMENT:

USER PROFILES, PREFERENCES, AND INTERACTIONS

ANALYTICS:

CAPTURING AND ANALYSING LARGE VOLUMES OF DATA IN REAL-TIME

Case Studies and Success Stories

Netflix: managing massive amounts of streaming data

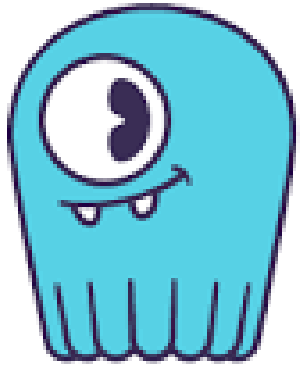
Instagram: handling user-generated content and social interactions

Apple: utilising Cassandra for distributed data storage

Spotify: powering personalised music recommendations and playlists

ScyllaDB

A C++ reimplementation of Apache Cassandra



SCYLLA

Best Practices for Apache Cassandra

Careful data modelling to optimise performance and scalability

Proper hardware selection and configuration for optimal performance

Regular monitoring and maintenance of cluster health

Consistent backups and disaster recovery planning

Cassandra Ecosystem and Integration

- Integration with popular frameworks like Apache Spark and Apache Kafka
- Support for different programming languages through drivers
- Rich ecosystem of tools for monitoring, management, and data visualisation

Cassandra Data Modelling Best Practices

Denormalization and duplication of data for efficient queries

Understanding data access patterns for optimal table design

Proper use of primary keys, partition keys, and clustering columns

Hands-on Cassandra

Introduction to MongoDB

Overview of MongoDB as a NoSQL document database

Benefits of using MongoDB for flexible and scalable data storage

Use cases where MongoDB excels, such as real-time analytics and content management systems

MongoDB Architecture

Document-oriented data model

Replica sets for high availability and fault tolerance

Query and indexing mechanisms for efficient data retrieval

MongoDB Document Model

BSON (Binary JSON) format for document storage

Documents composed of fields and values

Support for nested and dynamic schema

Rich data model for storing complex structures and relationships

CRUD Operations in MongoDB

- Create, Read, Update, and Delete operations
- Inserting documents into collections
- Querying documents using the MongoDB Query Language (MQL)
- Updating and deleting documents

Indexing in MongoDB

- Types of indexes: single-field, compound, multi-key
- Index creation and management
- Query optimization through index usage

Aggregation Framework

- Pipeline stages for filtering, transforming, grouping, and analyzing data

Data Replication and High Availability

- Replica sets for data redundancy and fault tolerance
- Primary and secondary nodes in a replica set
- Automatic failover and election of a new primary

Data Distribution and Horisontal scaling

- Data shards are distributed along nodes in the reply set.
- A router to coordinate the data access (mongos)
- A configuration server to coordinate metadata about the cluster
- Data is partitioned into chunks and distributed along the replica nodes. Each chunk has an inclusive lower and exclusive upper range based on the shard key.

MongoDB in Big Data Processing

- MongoDB's role in handling unstructured and semi-structured data:
  - Supports flexible data model without predefined schemas
  - Handles diverse data types (text, JSON, binary, etc.)
  - Enables easy storage and retrieval of unstructured and semi-structured data
- Scalability and flexibility of MongoDB for big data use cases:
  - Horizontal scalability
  - High-performance data processing and parallel querying
  - Seamless integration with big data frameworks and analytics tools

Hands-on MongoDB

Overview of big data architectures and their applications in handling big data for AI

Introduction to Big Data Architectures for AI

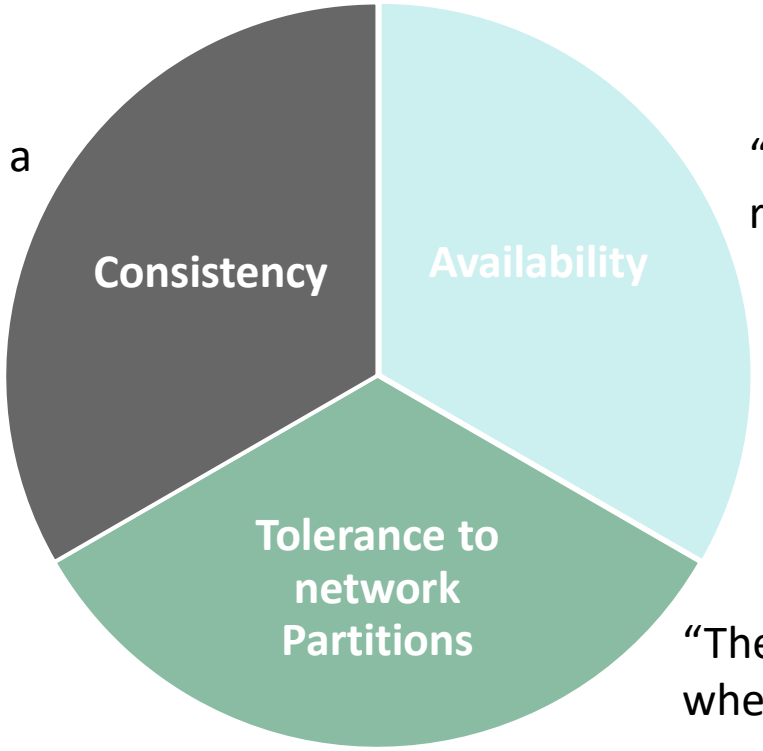
- Big Data Architectures refer to the combination of technologies and frameworks designed to handle and process large volumes of data.
- In the context of AI, Big Data Architectures play a crucial role in enabling the effective handling of data required for training and powering AI models.
- These architectures provide scalable, distributed, and parallel processing capabilities to tackle the challenges posed by big data in AI applications.

So far, we know about big data technologies and how to split our systems and the consequences.

But how do we process the data?

CAP theorem (a tradeoff for distributed systems)

"Every response sent to a client is correct"



"Every request eventually receives a response"

"The system can respond even when a partition is down"

Brewer, E. A. (2000, July). Towards robust distributed systems. In *PODC* (Vol. 7, No. 10.1145, pp. 343477-343502).

S. Gilbert and N. Lynch, "Perspectives on the CAP Theorem," in *Computer*, vol. 45, no. 2, pp. 30-36, Feb. 2012, doi: 10.1109/MC.2011.389.

Lambda architecture

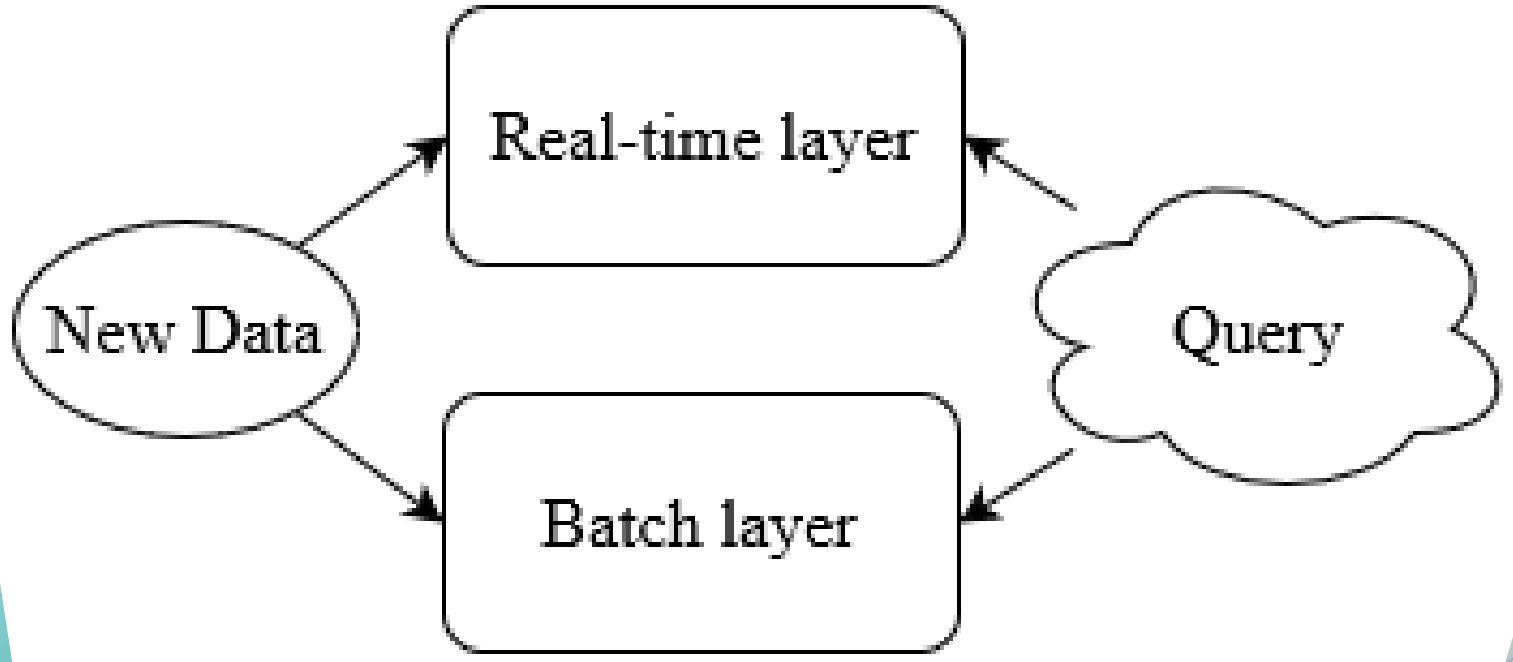
- Lambda architecture is a data processing architecture designed to handle massive volumes of data in a robust and scalable manner.
- It combines batch processing and real-time/streaming processing to provide a comprehensive solution for big data analytics.
- It enables organizations to process and analyze both historical and real-time data, facilitating timely insights and decision-making.

Components of Lambda Architecture

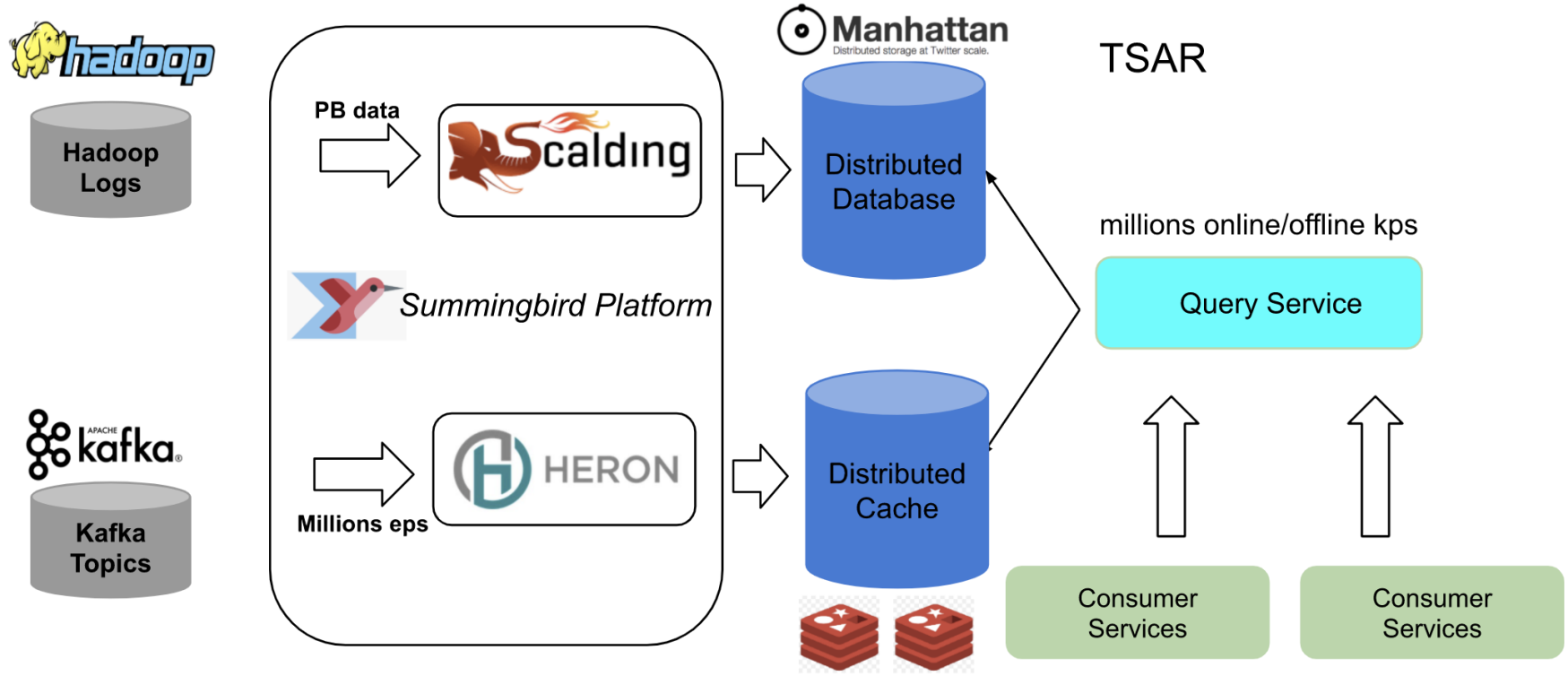
- Batch Layer: The batch layer handles the large-scale processing of historical data, providing comprehensive and accurate results. It reprocesses the whole data every time.
- Speed Layer: The speed layer processes real-time data streams, enabling low-latency and up-to-date insights.
- Serving Layer: The serving layer merges the results from the batch and speed layers, providing a unified view of the data for querying and analysis

http://nathanmarz.com/blog/how-to-beat-the-cap-theorem.html

LAMBDA architecture

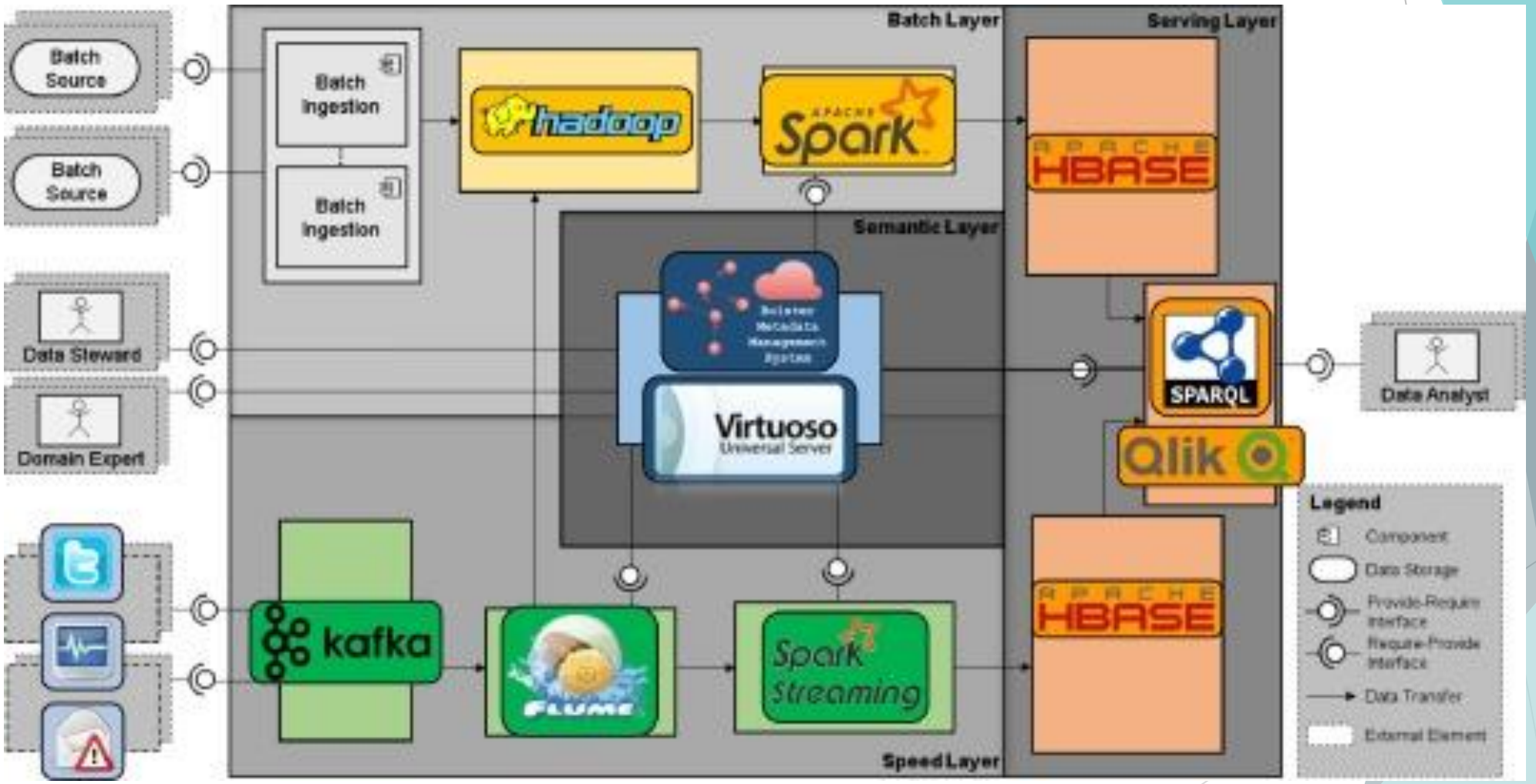


Lambda on Twitter



https://blog.twitter.com/engineering/en_us/topics/infrastructure/2021/processing-billions-of-events-in-real-time-at-twitter-

Another example



https://doi.org/10.1016/j.infsof.2017.06.001

Benefits and Advantages of Lambda Architecture

- Easily scale data processing and analytics capabilities to handle increasing data volumes.
- Combine batch and real-time processing for both historical and up-to-the-minute insights.
- Support iterative refinement and improvement of data models and analytics.
- Accommodate advanced analytics techniques, including machine learning algorithms.
- Cover both historical and real-time data processing for a complete view.
- Oriented towards immutable data and analytical solutions

Kappa architecture

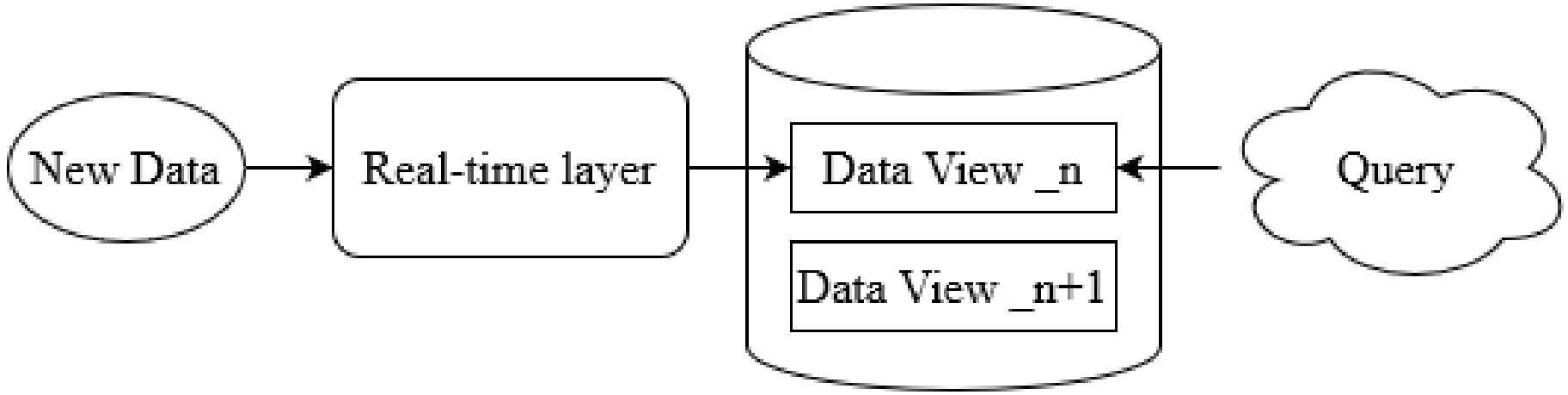
- Kappa architecture is a data processing architecture designed for handling real-time streaming data.
- It simplifies the architecture by eliminating the need for separate batch and stream processing systems.
- In Kappa architecture, data is ingested, processed, and served in a continuous and unified stream.

Components of Kappa Architecture

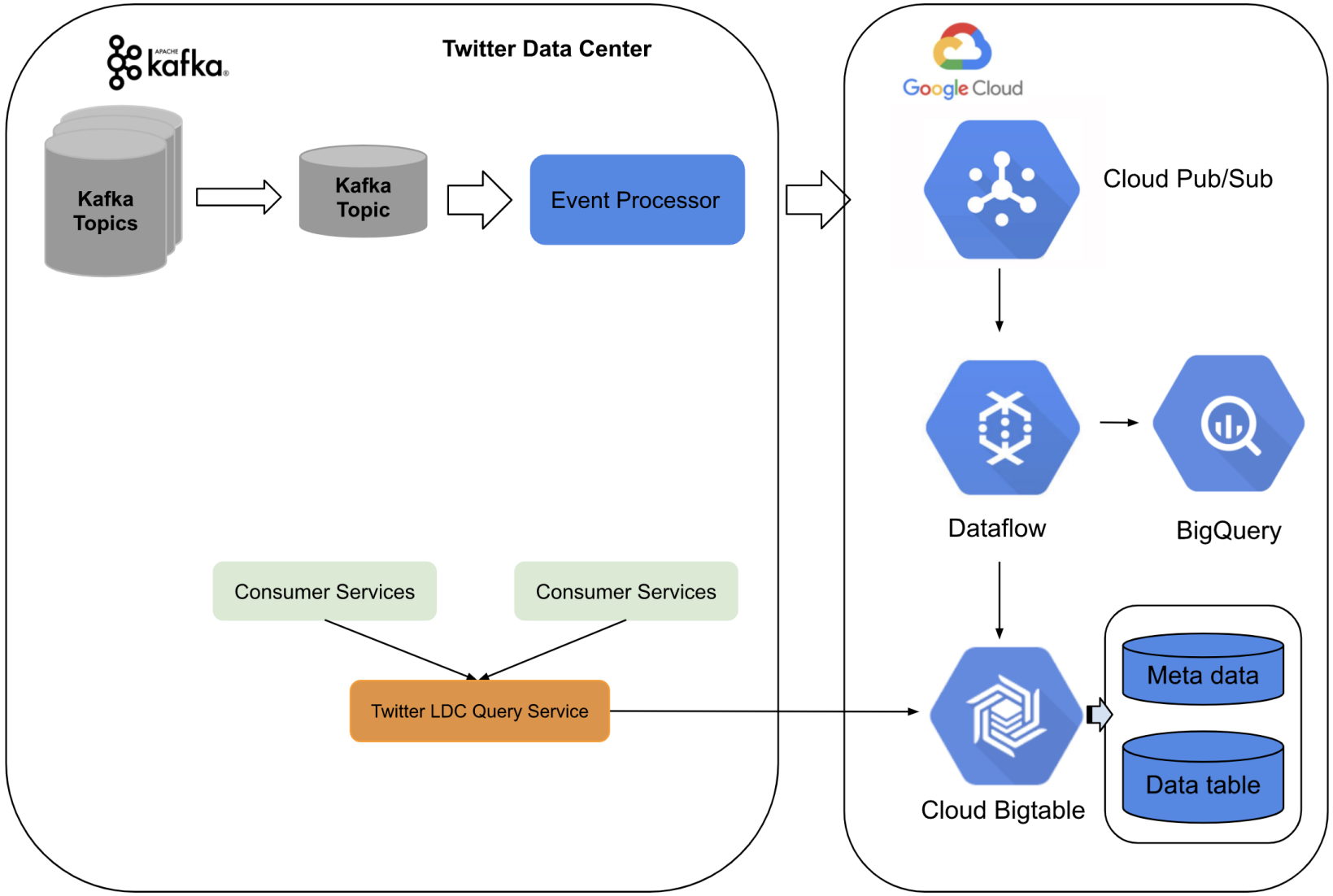
- Real-time layer: Real-time data is ingested from various sources into a streaming platform, such as Apache Kafka.
- Serving Layer: Processed data is made available for consumption by applications, analytics, or downstream systems.
- If the analysis needs to be reprocessed, it is done in parallel with the current one, until it reaches the same point. Then, the serving layer changes.

https://www.oreilly.com/radar/questioning-the-lambda-architecture/

KAPPA architecture



Kappa architecture in Twitter



Twitter numbers

Metrics		New Architecture	Old Architecture	
		Real Time Pipelines on Kafka and Dataflow	Real Time Pipelines on Heron	Batch Pipelines on Hadoop
Processing	Events Processed	~ 4 million events/second	~ 4 million events/second	PB data scale daily
	Real Time Compute Throughput	~600MB/s - ~1GB/s	~30MB/s - ~100MB/s *	N/A
	Batch Compute Cost	N/A	N/A	~300,000,000 PB Millis
Latency		~10s	~10s - 10 min	~1 day
Aggregation Correctness	Events Loss When Restarting	No	Yes	No
	Late Events Handling	Yes	No	No

* Heron compressed data internally for streaming

Benefits and Advantages of Kappa Architecture

- Kappa architecture eliminates the complexity of maintaining separate batch and stream processing systems.
- Enables real-time analytics and faster decision-making by processing data in near real-time.
- Easily scale the system horizontally to handle increasing data volumes and varying workloads.
- Minimizes data latency by processing data as it arrives, eliminating the need for batch processing intervals.
- Offers a streamlined data pipeline with fewer components, reducing operational complexity.
- Allows easy experimentation and modification of processing logic without impacting the entire pipeline.

Liquid architecture

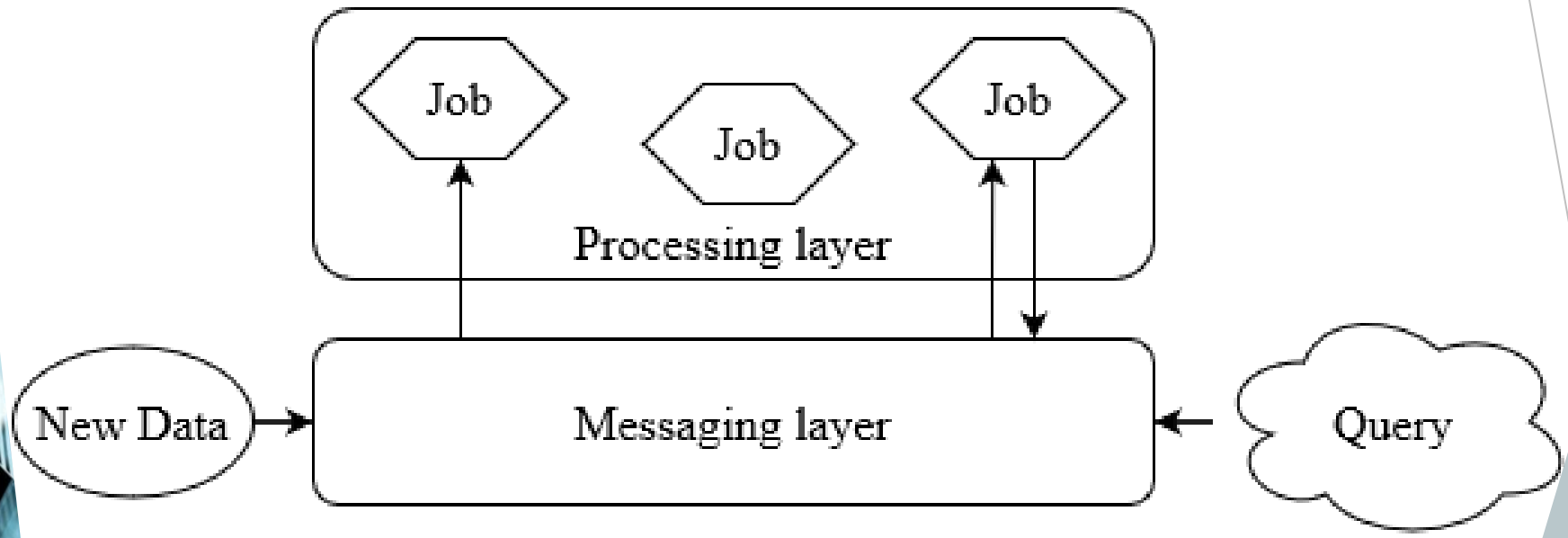
- It combines batch, stream, and interactive processing capabilities to provide a unified and flexible approach to data processing.

Components of Liquid Architecture:

- Messaging layer: Distributes the messages around jobs.
- Processing layer: where stateful and stateless jobs can process the data.

https://www.cidrdb.org/cidr2015/Papers/CIDR15_Paper25u.pdf

Liquid architecture



Benefits and Advantages of Liquid Architecture

- Hybrid Processing Paradigm: Liquid architecture combines the strengths of batch, stream, and interactive processing, allowing for versatile data processing.
- Real-time Insights and Responsiveness: Enables real-time analytics and immediate insights from streaming data, empowering faster decision-making.
- Scalability and Elasticity: Easily scales the processing infrastructure based on data volumes, processing requirements, and resource needs.
- Flexibility and Agility: Allows for dynamic and adaptable data pipelines, accommodating changing business needs and evolving data sources.

Applications of Big Data Architectures in Handling Big Data for AI

- Storage and Processing: Storage and processing of massive amounts of data, enabling AI systems to access and analyse large datasets efficiently.
- Real-time Data Streaming: Apache Kafka, enable the ingestion and processing of real-time data streams, facilitating immediate insights and decision-making for AI applications.
- Scalability and Parallel Processing: Big Data technologies like Apache Hadoop and Apache Spark offer scalability and parallel processing capabilities, allowing AI workflows to efficiently handle large-scale data processing and analytics.

SWOT Brainstorming

In groups. Think about:

- Strengths
- Weaknesses
- Opportunities
- Threats

For Big Data, the technologies and architectures when they are applied within an organization with similar requirements like yours.