ORACLE Certified Professional

Oracle Certified Professional Java Programmer

Microsoft Certified Professional

sohailimran@yahoo.com

Real-Time Data Processing

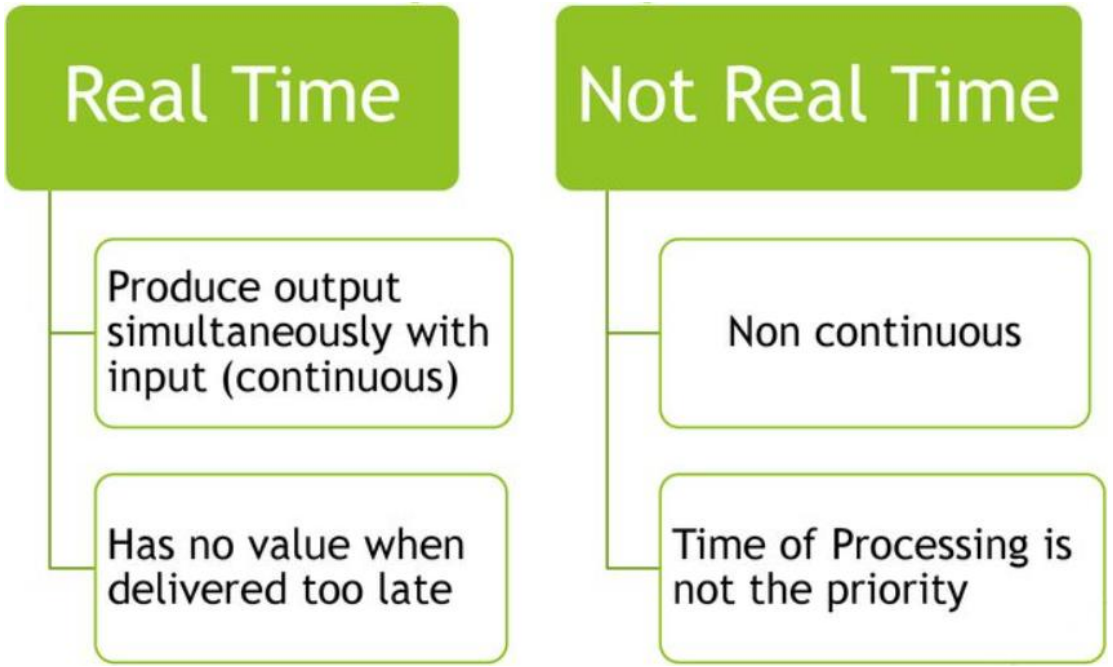
Sohail IMRAN سهیل عمران

Introduction

Ref: Web

intro

Real-time stream processing is the process of taking action on data at the time the data is generated or published.
Historically, real-time processing simply meant data was “processed as frequently as necessary for a particular use case.”
But as stream processing technologies and frameworks are becoming ubiquitous, real-time stream processing now means what it says.



RT vs Streaming

Real-time data typically refers to data that is immediately available without delay from a source system or process for some follow-up action.
For example, day traders may require real-time stock ticker data on which they run algorithms (or processes) in order to trigger a buy, no-buy, or sell action.

Streaming data is data that is continuously generated and delivered rather than processed in batches or micro-batches.
This is often referred to as “event data,” since each data point describes something that occurred at a given time.

Again, stock ticker data is a good example. If day traders collect and process a batch of data every 15 minutes, or when they complete some other interval or cross some threshold, they may miss many opportunities to buy and sell for a profit.

RT Data Streaming Tools & Technologies

Traditional data tools were built around disk-based processing and batch data pipelines, making them insufficient for streaming real-time data as required in the use cases above. While a variety of real-time data streaming tools and technologies have entered the market in the last few years, there are some commonalities.

•**In-memory processing.** RAM is critical to **stream processing**. Disk-based technologies simply aren't fast enough to process streams in real-time, even in massively parallel architectures where the resources of many computers are joined together. The decreasing price for RAM has made **in-memory computing platforms** the de facto standard for building applications to support real-time data streaming.
•**Open source.** The open source community has been responsible for laying the groundwork for many real-time data streaming tools and technologies. Technology providers will often build features and components to make real-time stream processing technologies enterprise-ready, but there is quite often an open source element underneath.
•**Cloud.** Many of the applications that generate data for real-time stream processing are cloud-based. And many businesses are shifting their IT budgets to the cloud. Because of this convergence, real-time data streaming technologies are often cloud-native.

Infrastructure & Architecture

Infrastructure: Components of a System (e.g., server machines of IT department)

Architecture: Design which uses the components and shows interactions

Growth of Big Datasets

- Internet/Online Data
  - Clicks
  - Searches
  - Server requests
  - Web logs
  - Cell phone logs
  - Mobile GPS locations
  - User generated content
  - Entertainment (YouTube, Netflix, Spotify, …)
- Healthcare and Scientific Computations
  - Genomics, medical images, healthcare data, billing data
- Graph data
  - Telecommunications network
  - Social networks (Facebook, Twitter, LinkedIn, …)
  - Computer networks
- Internet of Things
  - RFID
  - Sensors
- Financial data

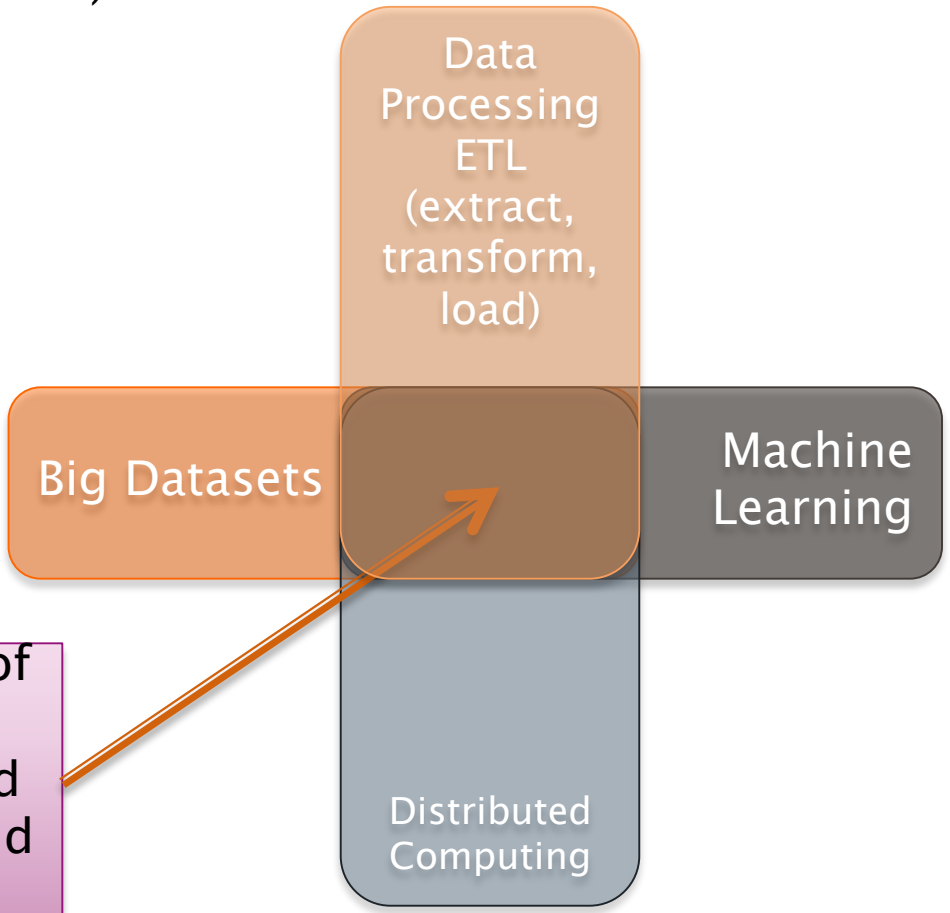
Cloud and Distributed Computing

- Pervasiveness of cloud based storage and computational resources
  - For processing of big datasets
- Cloud characteristics
  - Provide a scalable standard environment
  - On-demand computing
  - Pay as you need
  - Dynamically scalable
  - Cheaper

Data Processing & Machine Learning Methods

- Data processing
  - Traditional ETL (extract, transform, load)
  - Data Stores (HBase, ……..)
  - Tools for processing of streaming, multimedia & batch data

- Machine Learning (fourth trend)
  - Classification
  - Regression
  - Clustering
  - Collaborative filtering



Working at the Intersection of these four trends is very exciting and challenging and require new ways to store and process **Big Data**

Need for a framework …

- … for building such complex stream processing applications

- But what are the requirements from such a framework?

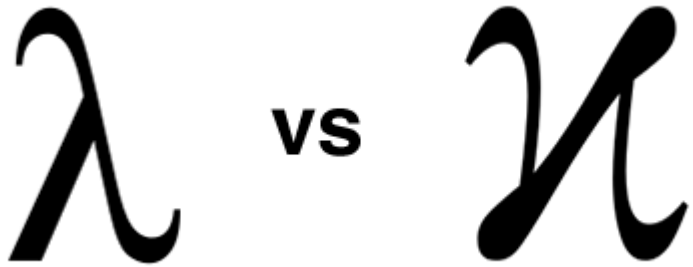
Requirements

- **Scalable** to large clusters
- **Second-scale** latencies
- **Simple** programming model
- **Integrated** with batch & interactive processing
- **Efficient fault-tolerance** in stateful computations

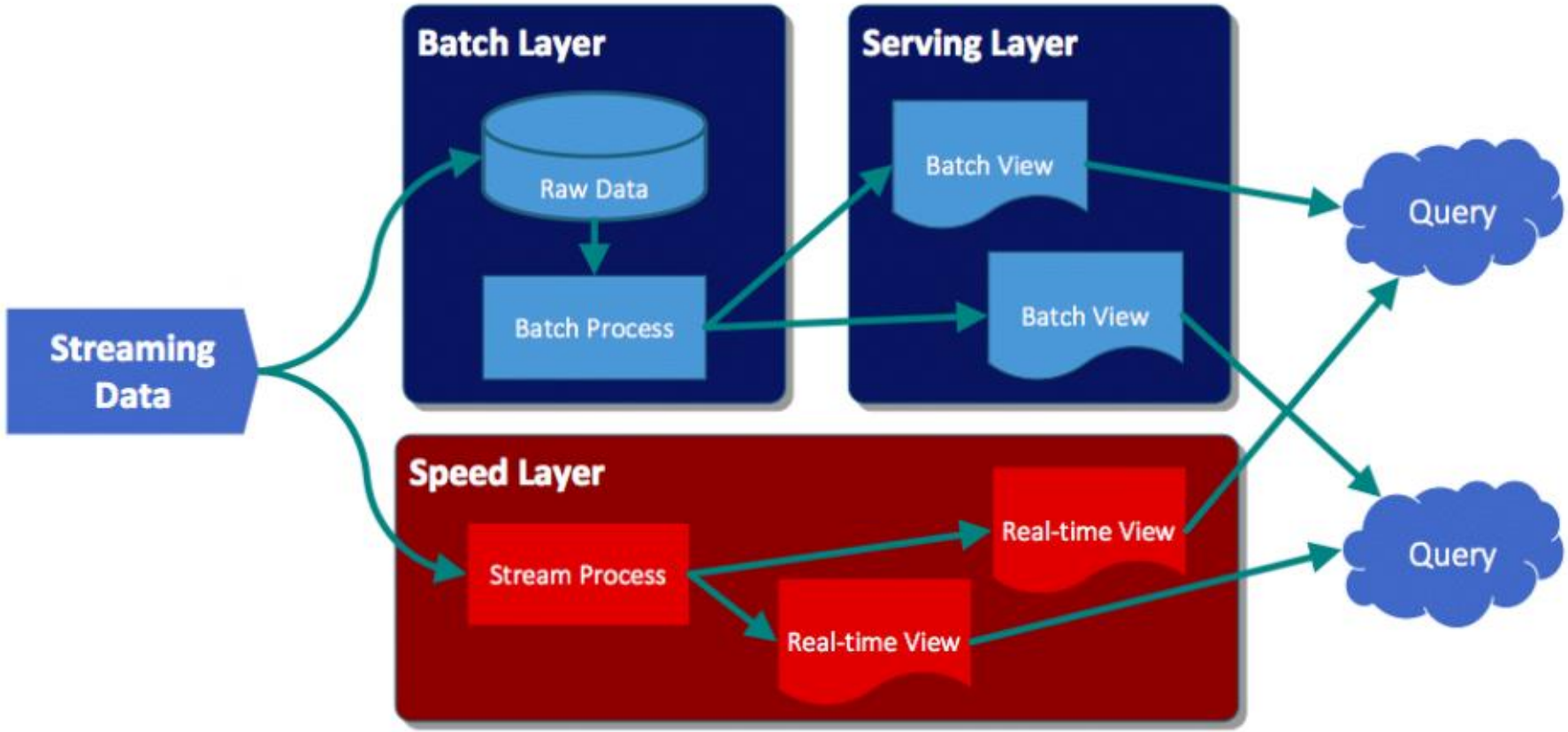
Real-Time Big Data Architectures

Real-time data processing often requires qualities such as scalability, fault-tolerance, predictability, resiliency against stream imperfections, and extensibility.

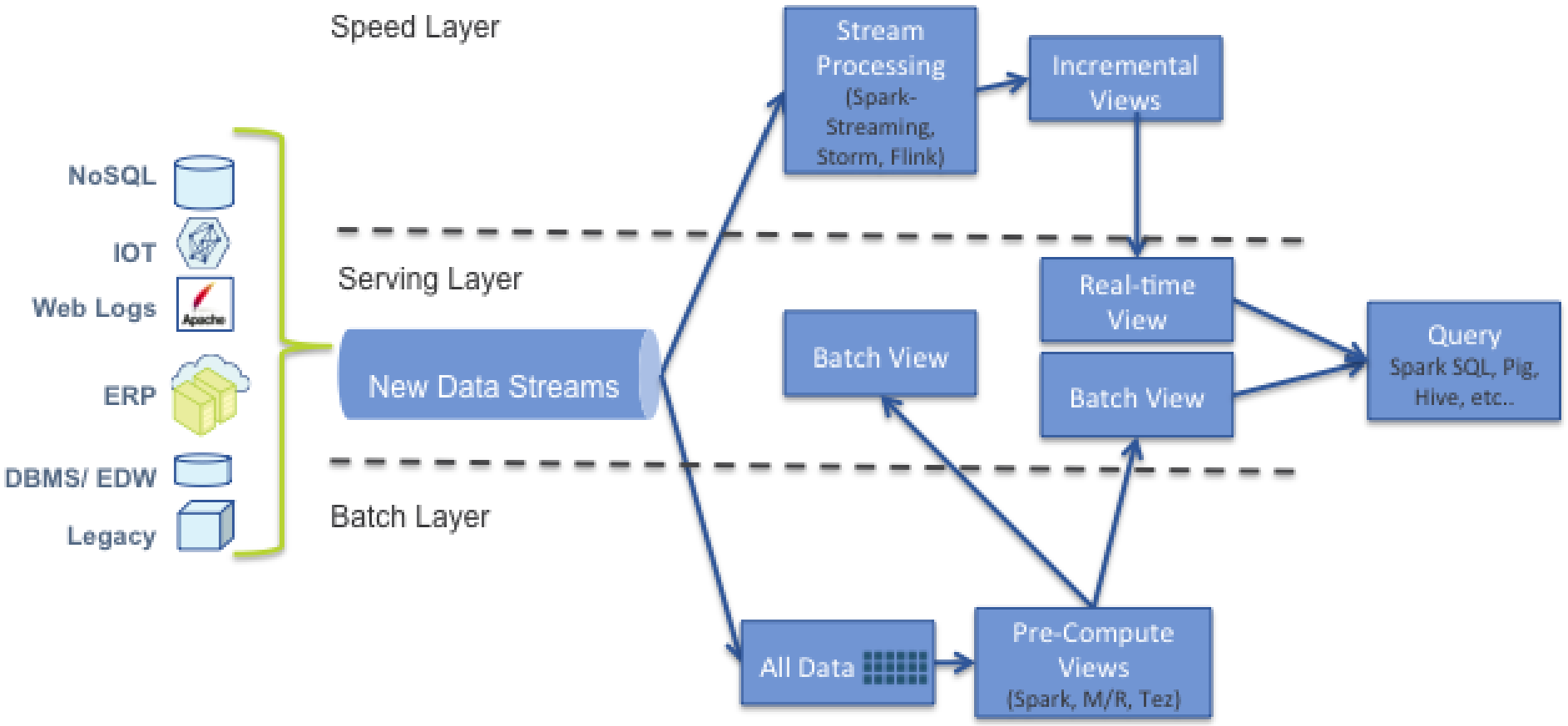
The number of choices you have for real-time big data architectures is just plain overwhelming.



The Lambda Architecture



Layers of λ


Speed Layer
Stream Processing (Spark-Streaming, Storm, Flink)
Incremental Views
NoSQL
IOT
Web Logs
ERP
DBMS/ EDW
Legacy
Serving Layer
New Data Streams
Batch View
Real-time View
Batch View
Query
Spark SQL, Pig, Hive, etc..
Batch Layer
All Data
Pre-Compute Views
(Spark, M/R, Tez)

batch layer

The **batch layer** stores the raw data as it arrives, and computes the batch views for consumption.
Naturally, batch processes will occur on some interval and will be long-lived.
The scope of data is anywhere from hours to years.
The batch views may be processed with more complex or expensive rules and may have better data quality and less skew.

speed layer (stream layer)

The **speed layer** is used to compute the real-time views to compliment the batch views.
The real-time views give you up to the moment access to the latest possible data.
As time goes on, real-time data expires and is replaced with data in the batch views.

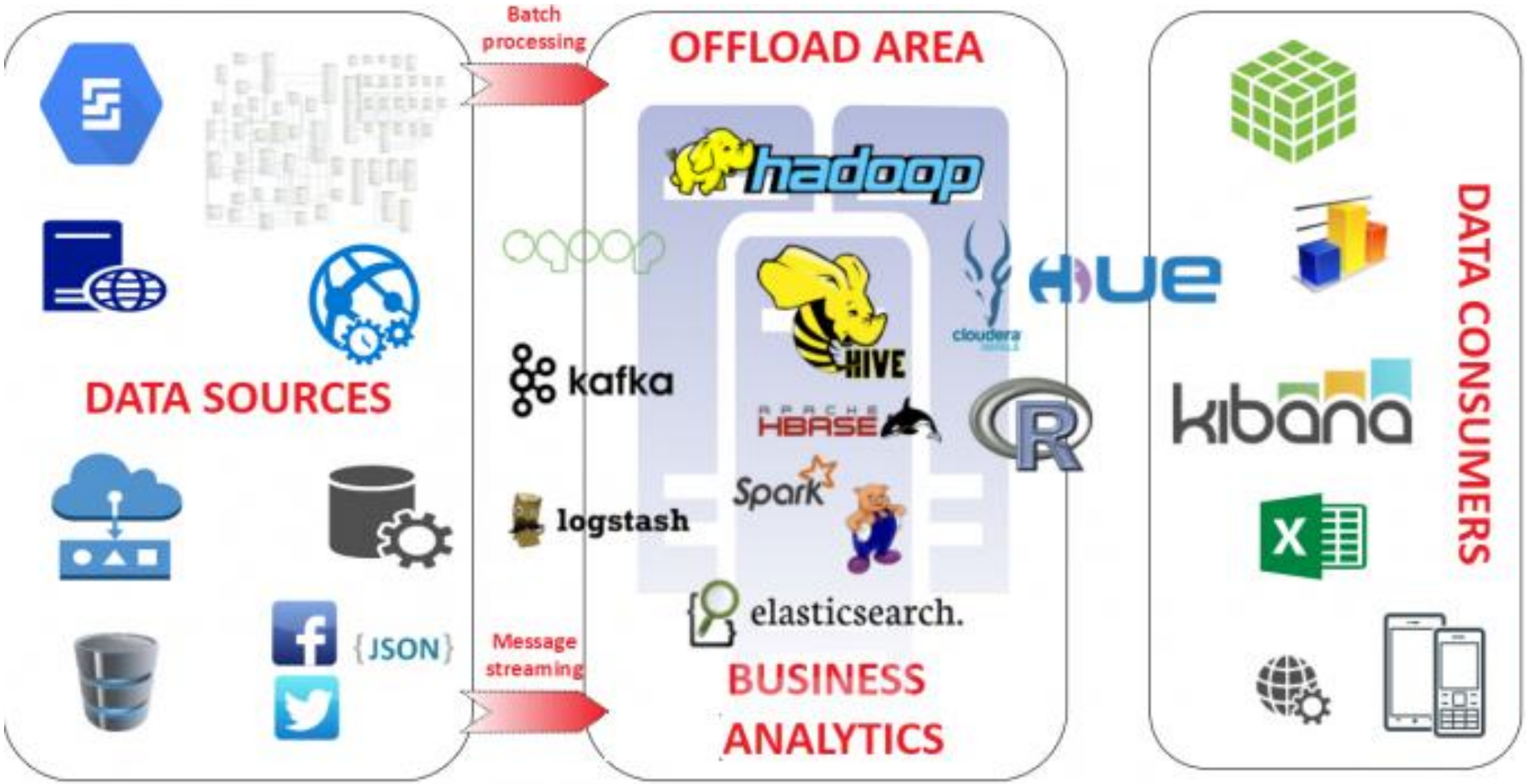
serving layer

The outputs from batch layer in the form of batch views and from speed layer in the form of near-real time views are forwarded to the serving layer which uses this data to cater the pending queries on ad-hoc basis.

Query = λ (Complete data) = λ (live streaming data) * λ (Stored data)

Any query may get a complete picture by retrieving data from both the batch views and the real-time views.
The queries will get the best of both worlds.

Lambda Architecture - Technologies



Pros n Cons

Pros

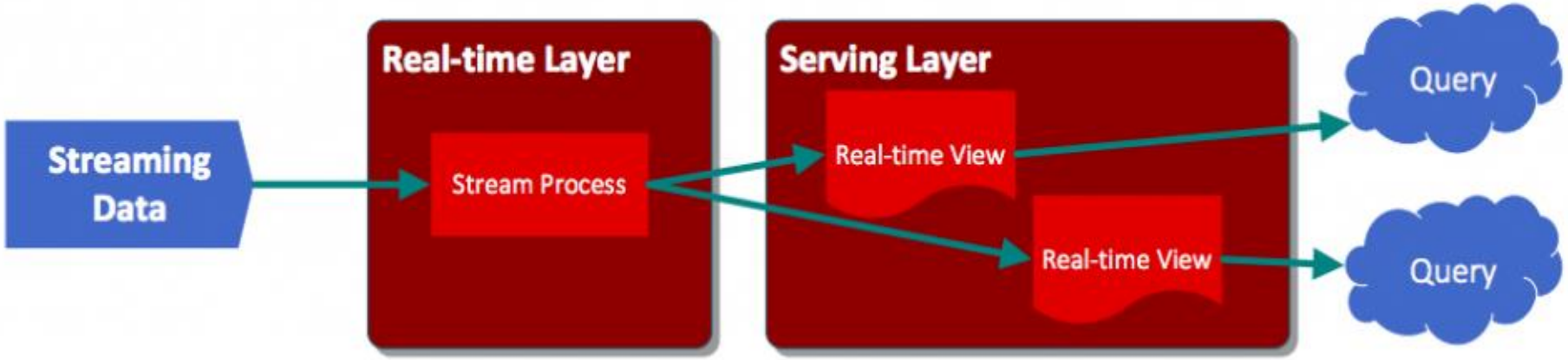
- Batch layer of Lambda architecture manages historical data with the fault tolerant distributed storage which ensures low possibility of errors even if the system crashes.
- It is a good balance of speed and reliability.
- Fault tolerant and scalable architecture for data processing.

Cons

- It can result in coding overhead due to involvement of comprehensive processing.
- Re-processes every batch cycle which is not beneficial in certain scenarios.
- A data modeled with Lambda architecture is difficult to migrate or reorganize.

The biggest detraction to this architecture has been the need to maintain two distinct (and possibly complex) systems to generate both batch and speed layers. Luckily with **Spark Streaming** (abstraction layer), this has become far less of an issue... although the operational burden still exists.

Kappa Architecture

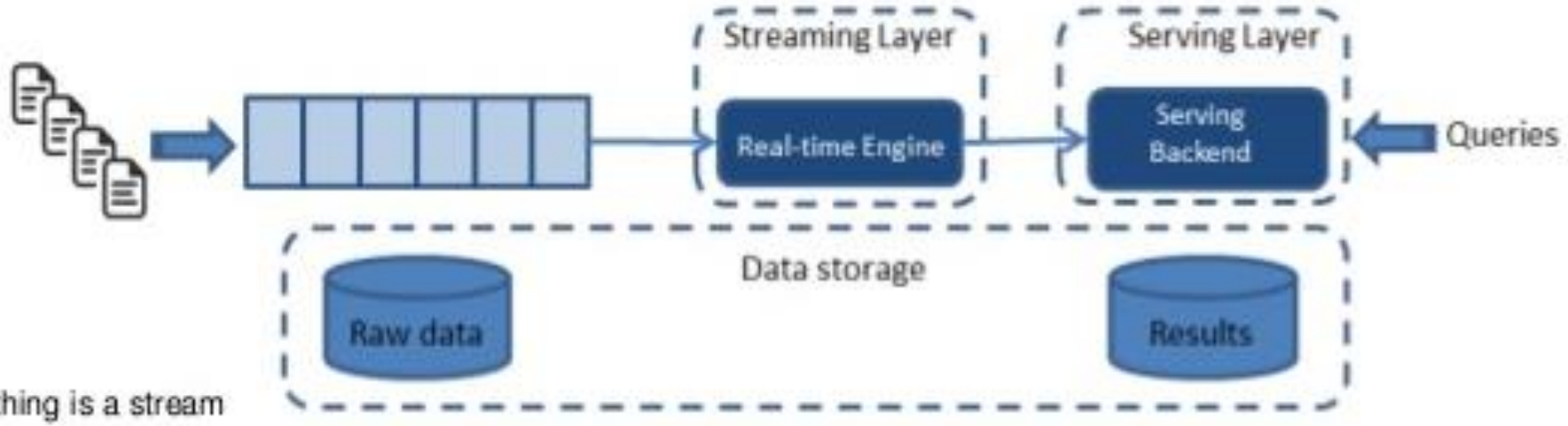


- Architecture is composed of only two layers:
  - The stream processing layer runs the stream processing jobs.
    - Normally, a single stream processing job is run to enable real-time data processing.
    - Data reprocessing is only done when some code of the stream processing job needs to be modified.
      - This is achieved by running another modified stream processing job and replying all previous data.
  - The serving layer is used to query the results (like the Lambda architecture).

Kappa Architecture

The **Kappa** Architecture focuses on only processing data as a stream.
It is not a replacement for the Lambda Architecture, except for where your use case fits.
For this architecture, incoming data is streamed through a real-time layer and the results of which are placed in the serving layer for queries.

Stream Processing with Scalable Storages



- Everything is a stream
- Immutable unstructured data sources
- Single analytics framework
- Windows on Streaming Layer
- Linearly scalable Serving Layer
- Interactive querying

Pros n Cons

Pros

- Kappa architecture can be used to develop data systems that are online learners and therefore don’t need the batch layer.
- Re-processing is required only when the code changes.
- It can be deployed with fixed memory.
- It can be used for horizontally scalable systems.
- Fewer resources are required as the machine learning is being done on the real time basis.

Cons

- Absence of batch layer might result in errors during data processing or while updating the database that requires having an exception manager to reprocess the data or reconciliation.