Big Data

(Streaming Analytics / Stream Processing)

Instructor: Thanh Binh Nguyen

September 1st, 2019

S3Lab
Smart Software System Laboratory


“Big data is at the foundation of all the megatrends that are happening today, from social to mobile to cloud to gaming.”
– Chris Lynch, Vertica Systems

Real-time Data

Real-time Data

Introduction

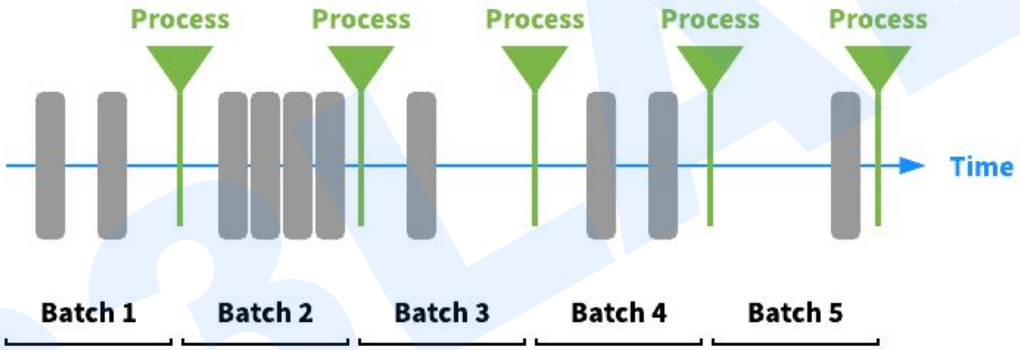
- We want to react to data faster, rather than storing them in a disk and periodically processing and acting on the data -> Real-time analytics
- Real-time analytic has three flavors
  - Real-time **Interactive / Ad-hoc Analytics:** Allows users to explore a large data set by issuing ad-hoc queries. Queries should respond within 10 seconds, which is considered the upper bound for acceptable human interaction.
  - Real-time **Streaming Analytics / Stream Processing:** users issue static queries once and they do not change, and the system process data as they come in without storing them and react to those data very fast, often within few milliseconds.
  - **Stream Data Integration**: primarily focuses on the ingestion and processing of data sources targeting realtime extract-transform-load (ETL) and data integration use cases. Filter and enrich the data. Optionally calculate time-windowed aggregations before storing the results in a database or file system.

Batch Processing, MicroBatch, Stream Processing

- **In batch processing**, newly arriving data elements are collected into a group. The whole group is then processed at a future time. When each group is processed can be determined by scheduling or triggered condition.
- The term "**microbatch**" is frequently used to describe scenarios where batches are small and/or processed at small intervals. Even though processing may happen as often as once every few minutes, data is still processed a batch at a time. Spark Streaming is an example of a system that supports micro-batch processing.

Batch Processing, MicroBatch, Stream Processing

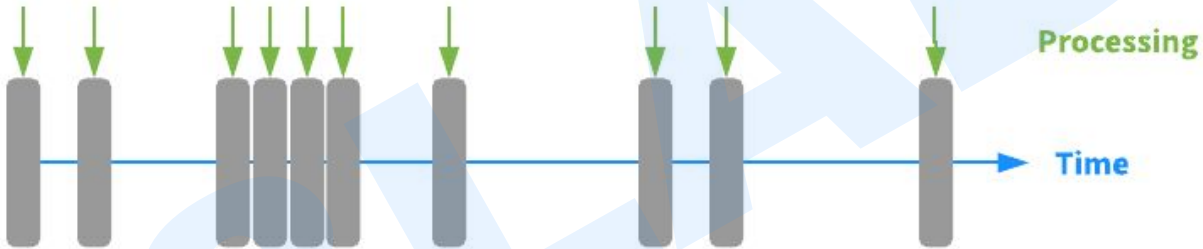
Batch Processing and MicroBatch



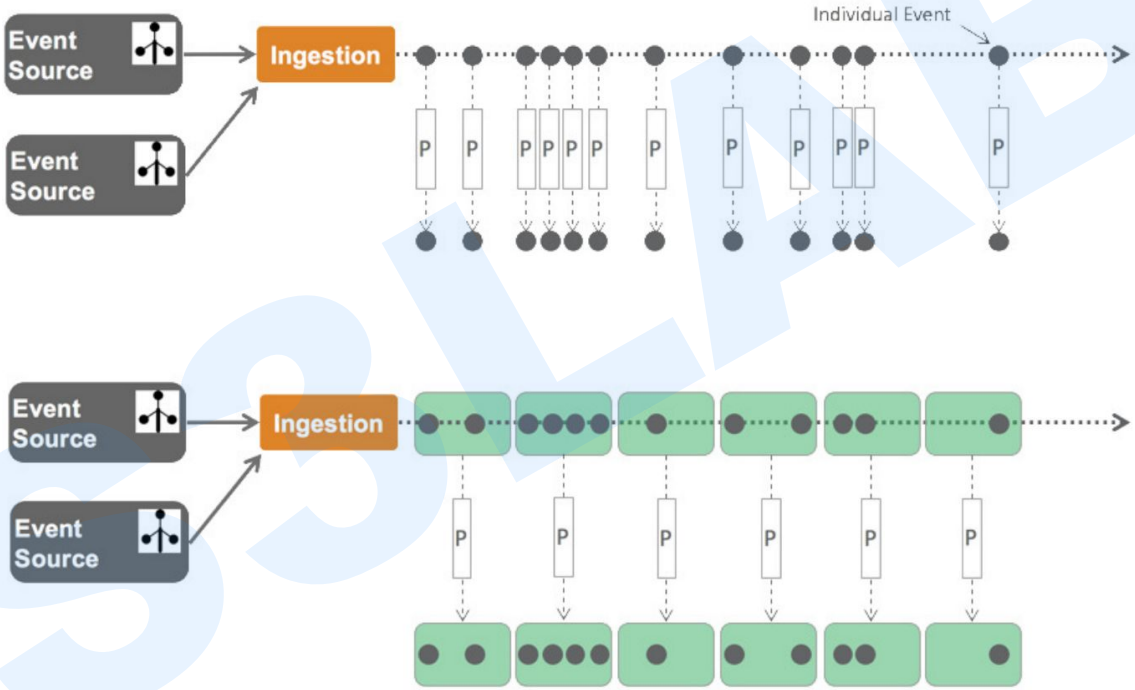
Batch Processing, MicroBatch, Stream Processing

- In **stream processing**, each new piece of data is processed when it arrives. Unlike batch processing, there is no waiting until the next batch processing interval and data is processed as individual pieces rather than being processed a batch at a time. Although each new piece of data is processed individually, many stream processing systems do also support "**window**" operations that allow processing to also reference data that arrives within a specified interval before and/or after the current data arrived.

Batch Processing, MicroBatch, Stream Processing

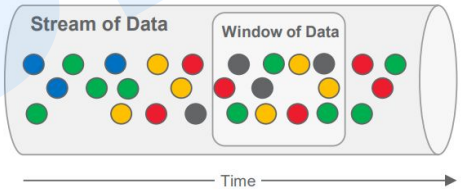


Batch Processing, MicroBatch, Stream Processing



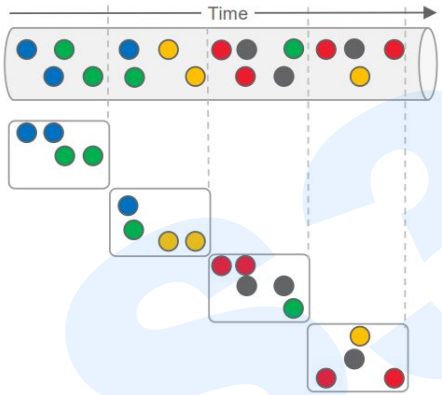
Windowing

- Computations over events done using windows of data
- Due to size and never-ending nature of it, it's not feasible to keep entire stream of data in memory
- A window of data represents a certain amount of data where we can perform computations on
- Windows give the power to keep a working memory and look back at recent data efficiently

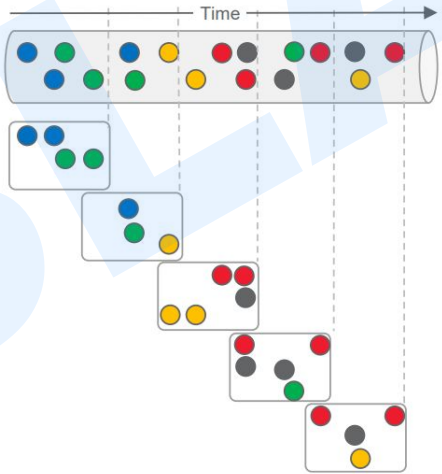


Windowing

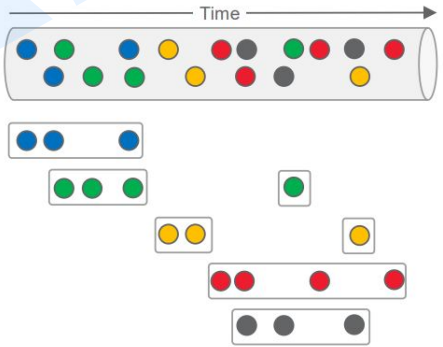
Fixed Window (aka Tumbling Window) - eviction policy always based on the window being full and trigger policy based on either the count of items in the window or time



Sliding Window (aka Hopping Window) - uses eviction and trigger policies that are based on time: *window length* and *sliding interval length*



Session Window – composed of sequences of temporarily related events terminated by a gap of inactivity greater than some timeout

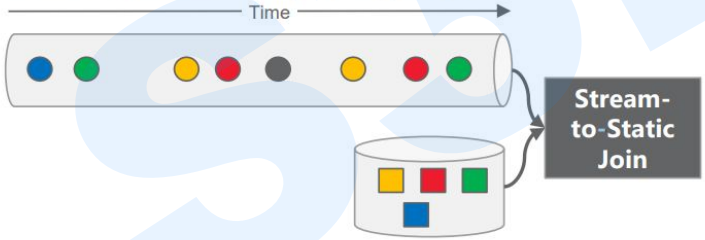


Joining

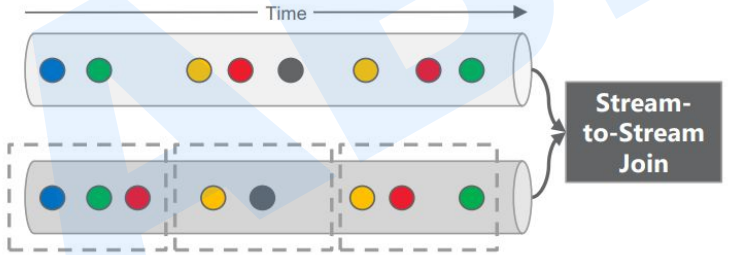
Challenges of joining streams

1. Data streams need to be aligned as they come because they have different timestamps
2. since streams are never-ending, the joins must be limited; otherwise join will never end
3. join needs to produce results continuously as there is no end to the data

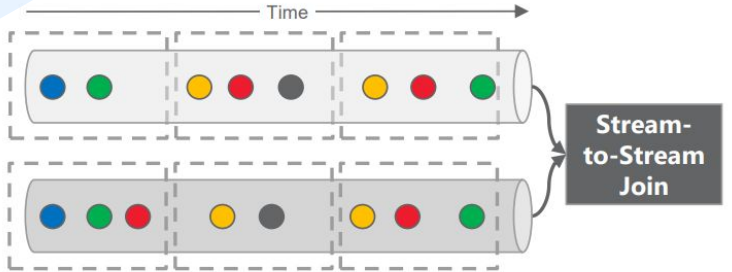
Stream to Static (Table) Join



Stream to Stream Join (one window join)



Stream to Stream Join (two window join)



Stateless vs. Stateful Stream Processing

- **Stateless stream:**
  - Each event is handled is completely independent from the preceding events. Given an event, the stream processor will treat it exactly the same way every time, no matter what data arrived beforehand.
  - Easy to scale up
- **Stateful stream**:
  - A "state" is shared between events and therefore past events can influence the way current events are processed. This state can usually be queried from outside the stream processing system as well.
  - Much more difficult to scale up

Stateless vs. Stateful Stream Processing

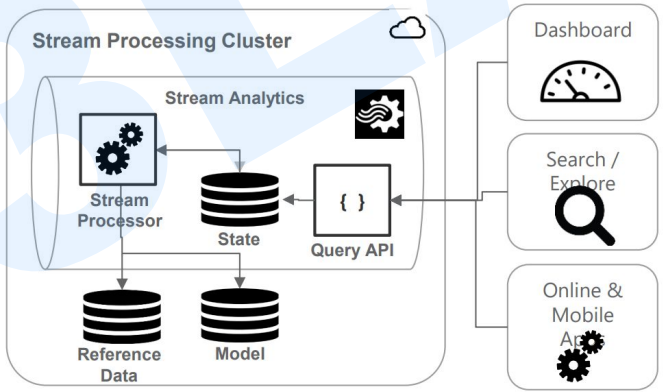
Stateful Stream - state management

- **Windowing**, **Joining** and **Pattern Detection** use State Management behind the scenes
- State Management services can be made available for custom state handling logic
- State needs to be managed as close to the stream processor as possible

Stateless vs. Stateful Stream Processing

Queryable State (aka. Interactive Queries)

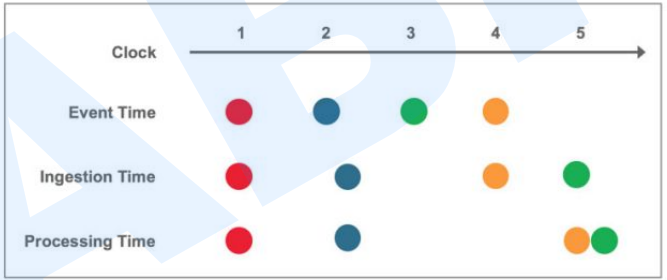
- Exposes the state managed by the Stream Analytics solution to the outside world. Allows an application to query the managed state, i.e. to visualize it. For some scenarios, Queryable State can eliminate the need for an external database to keep results



Event Time vs. Ingestion Time vs. Processing Time

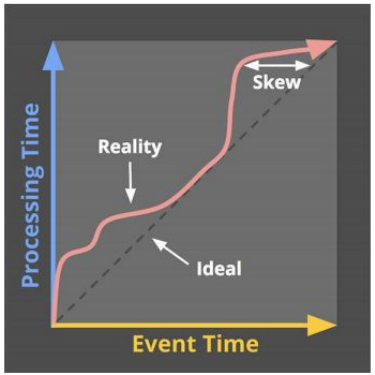
Event time
- the time at which events actually ***occurred***

Ingestion time / Processing Time
- the time at which events are ***ingested*** / ***observed*** in the system

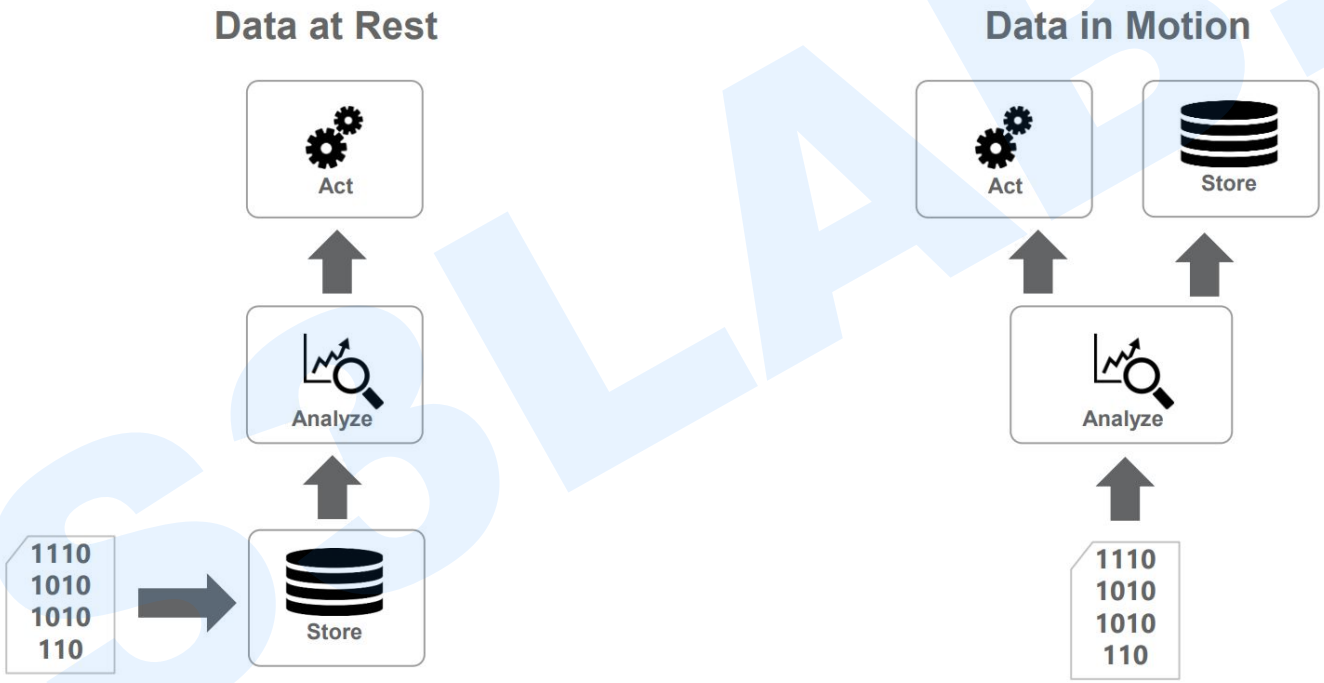


Not all use cases care about event times but many do

Examples
- characterizing user behavior over time
- most billing applications
- anomaly detection

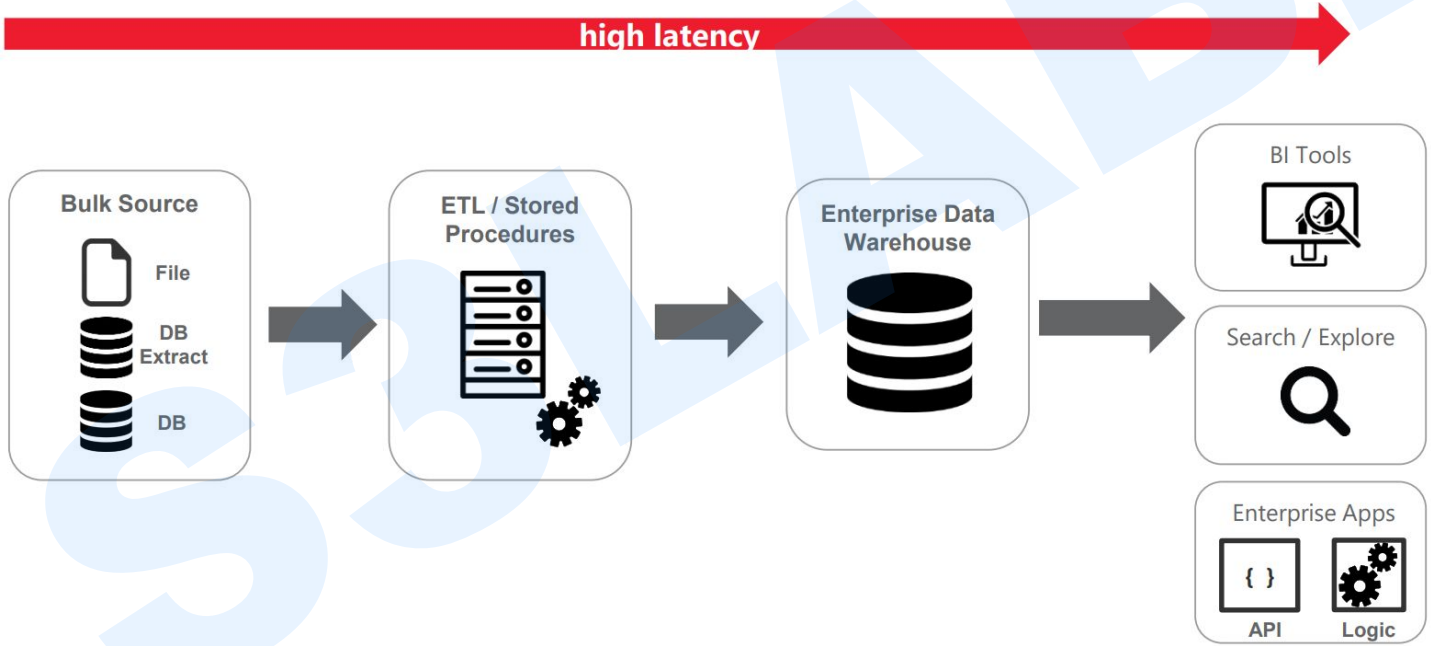


Data “at Rest” vs. “in Motion”



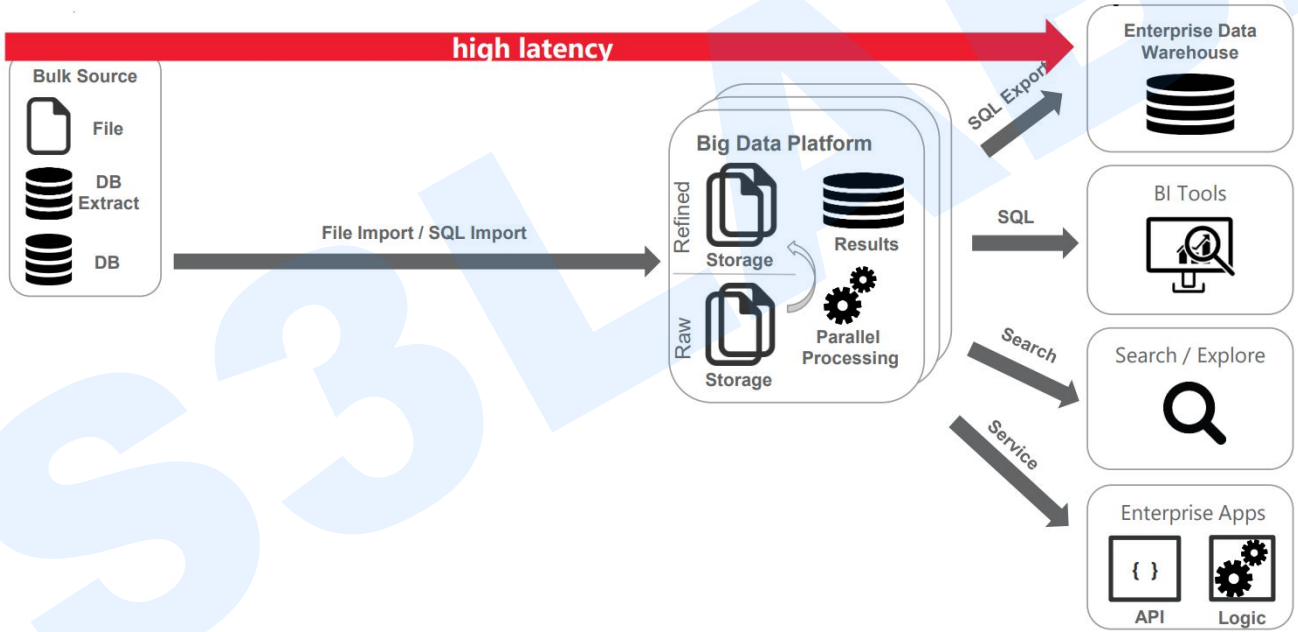
Motivation

Traditional BI infrastructures



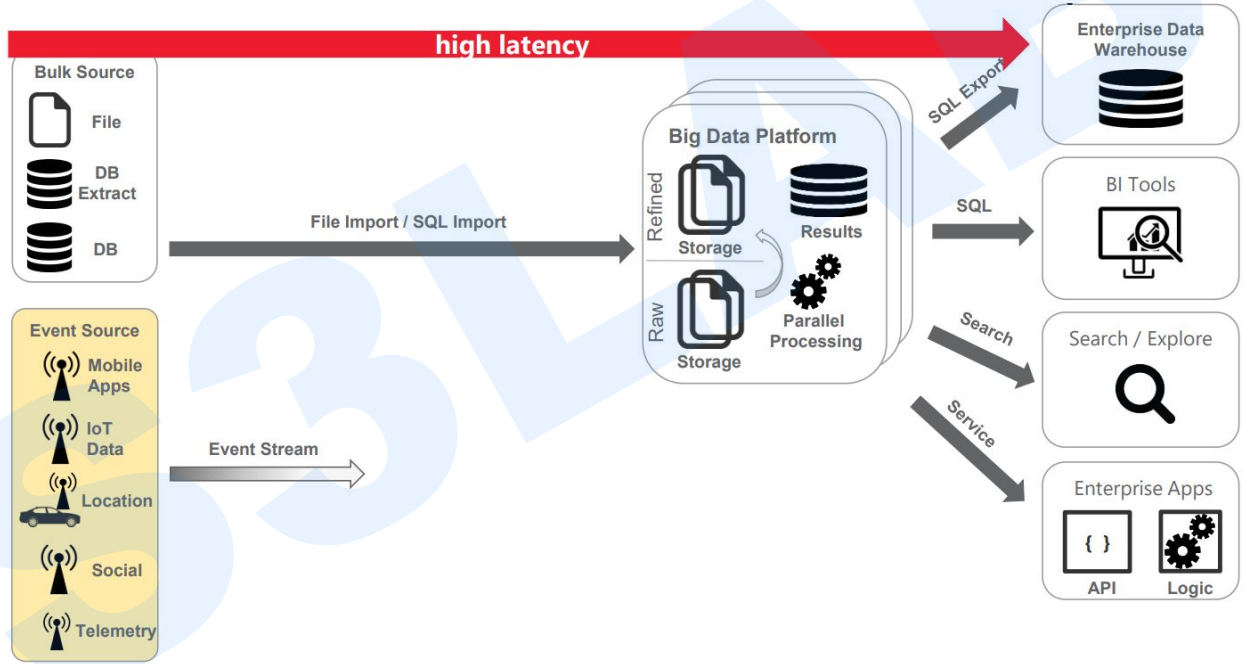
Motivation

Big Data solves Volume, Variety - But not Velocity



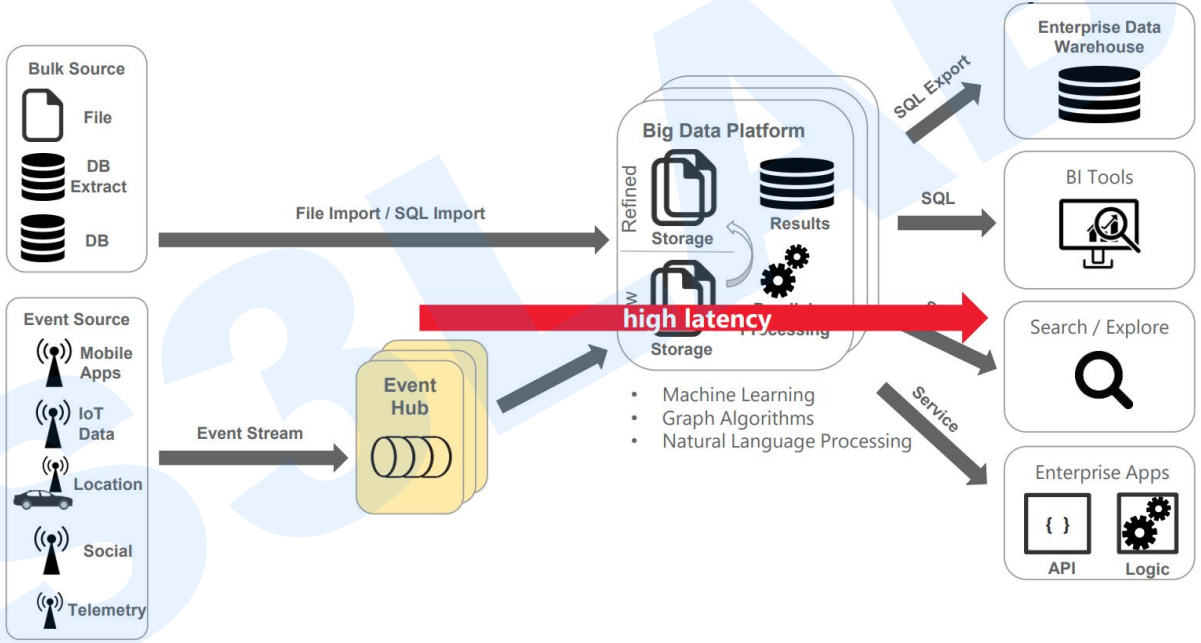
Motivation

Big Data solves Volume, Variety - But not Velocity

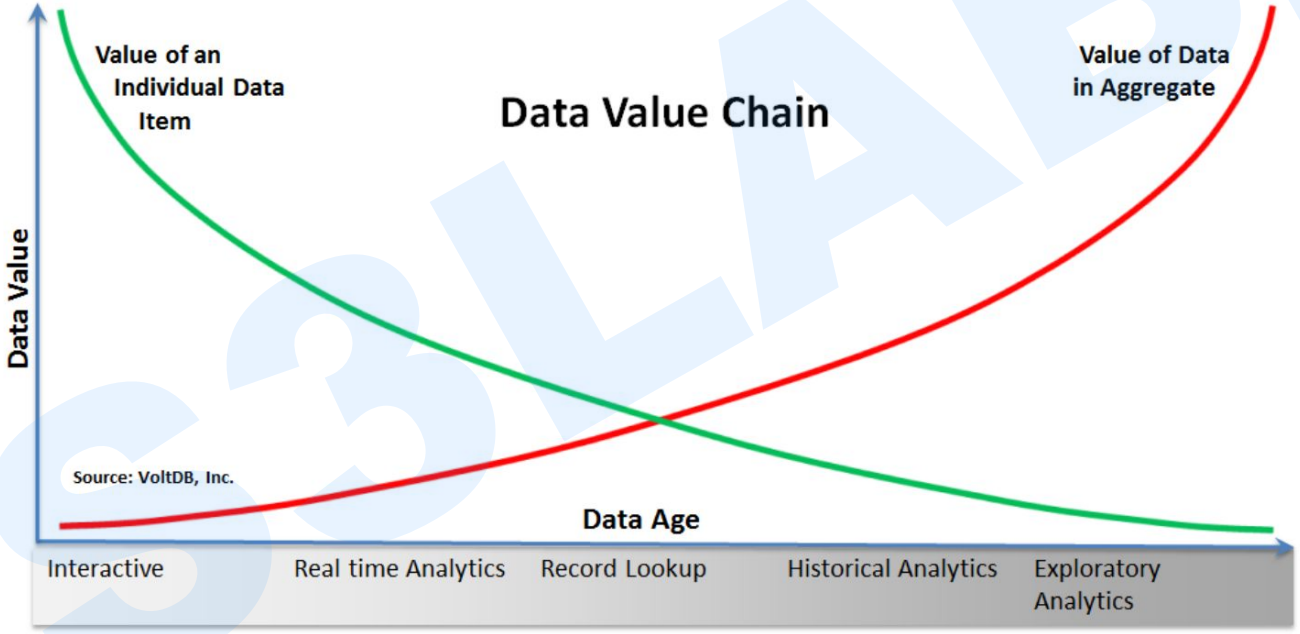


Motivation

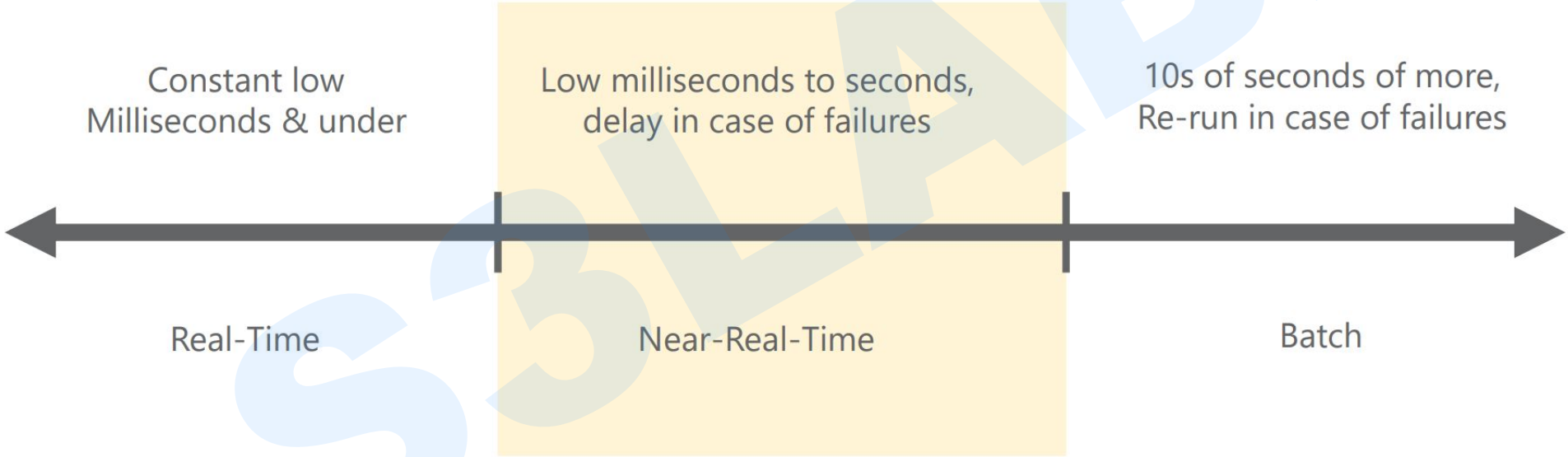
Big Data solves Volume, Variety - But not Velocity



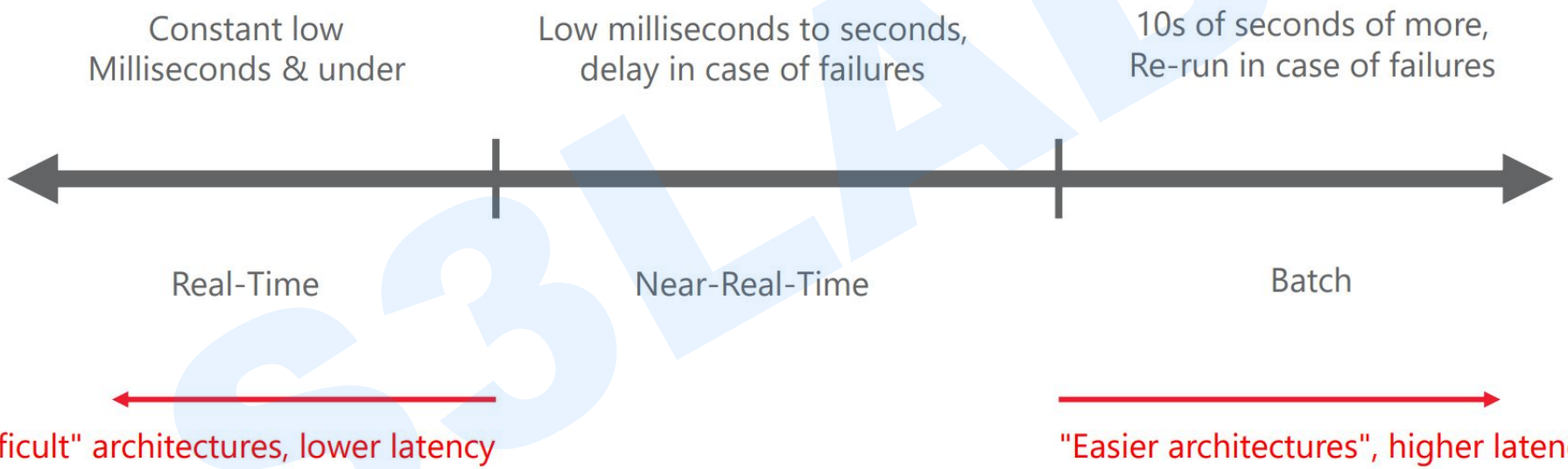
Data Value Chain



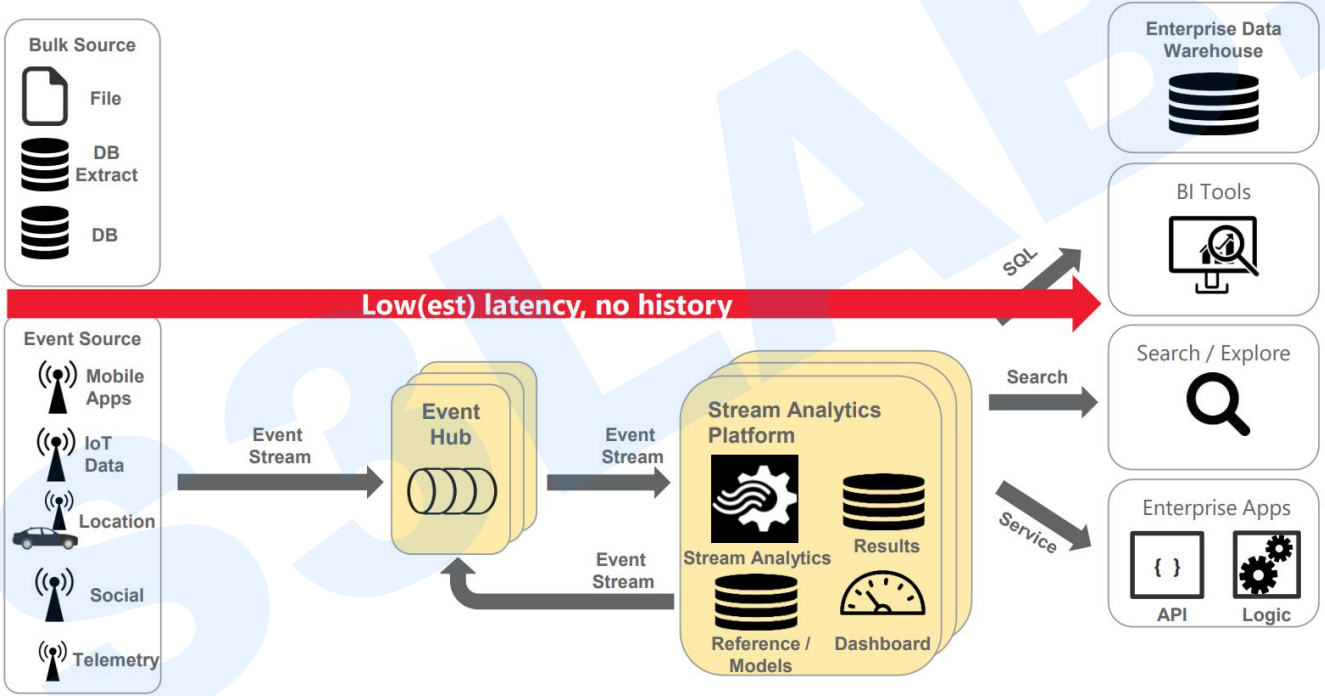
When to Stream



When to Stream

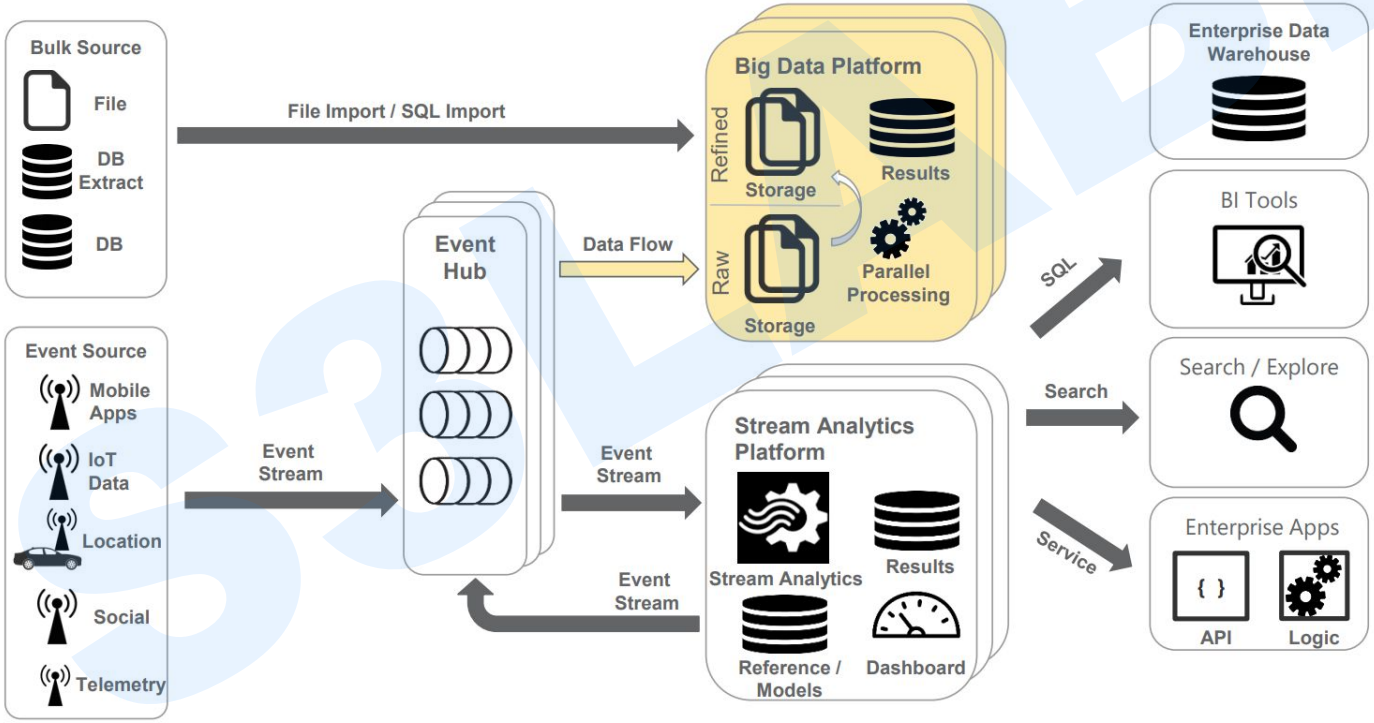


Stream Processing Architecture



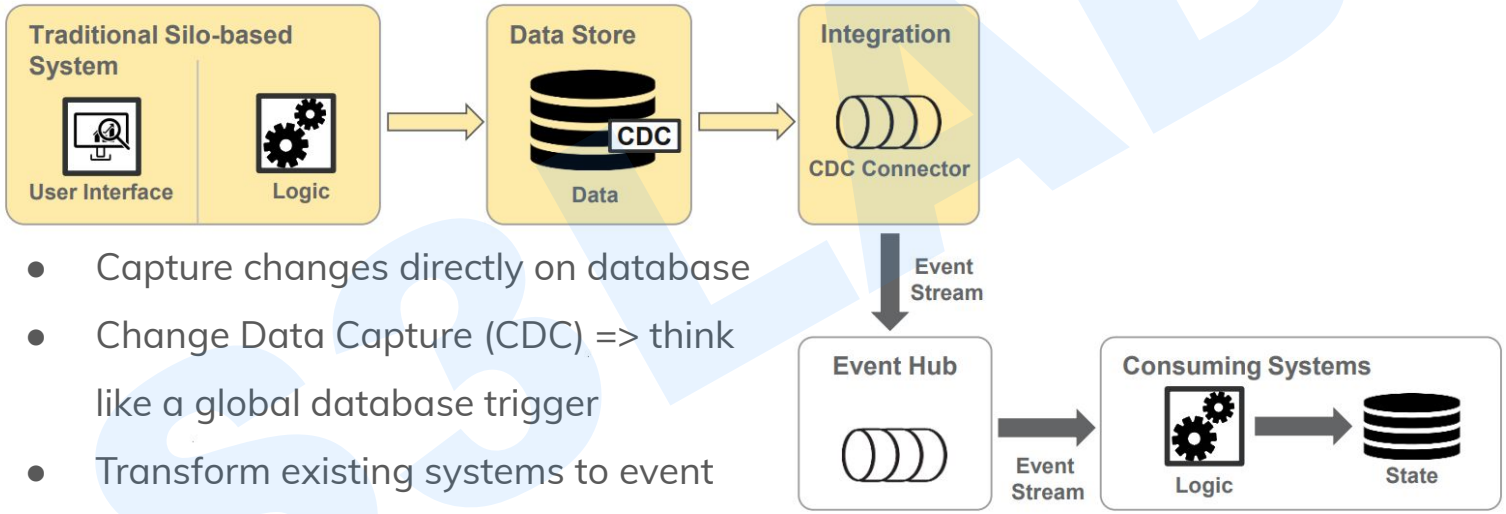
Stream Processing Architecture

For all historical data analysis



Stream Processing Architecture

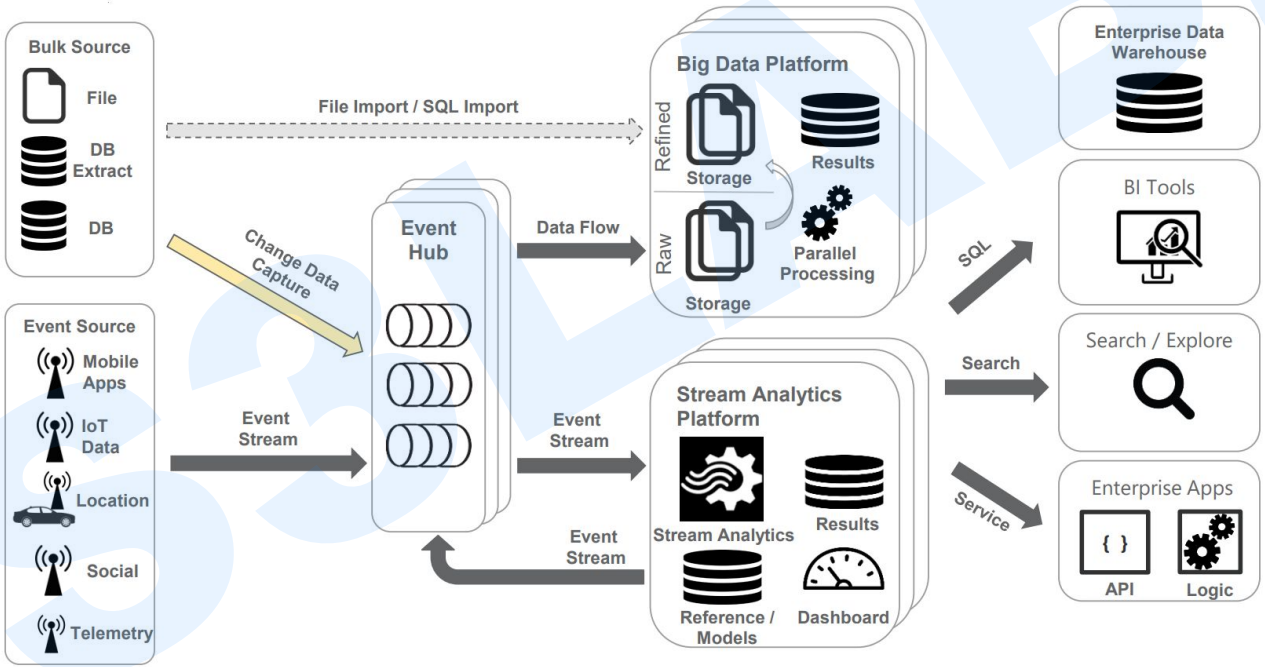
Integrate existing systems through CDC



- Capture changes directly on database
- Change Data Capture (CDC) => think like a global database trigger
- Transform existing systems to event producer

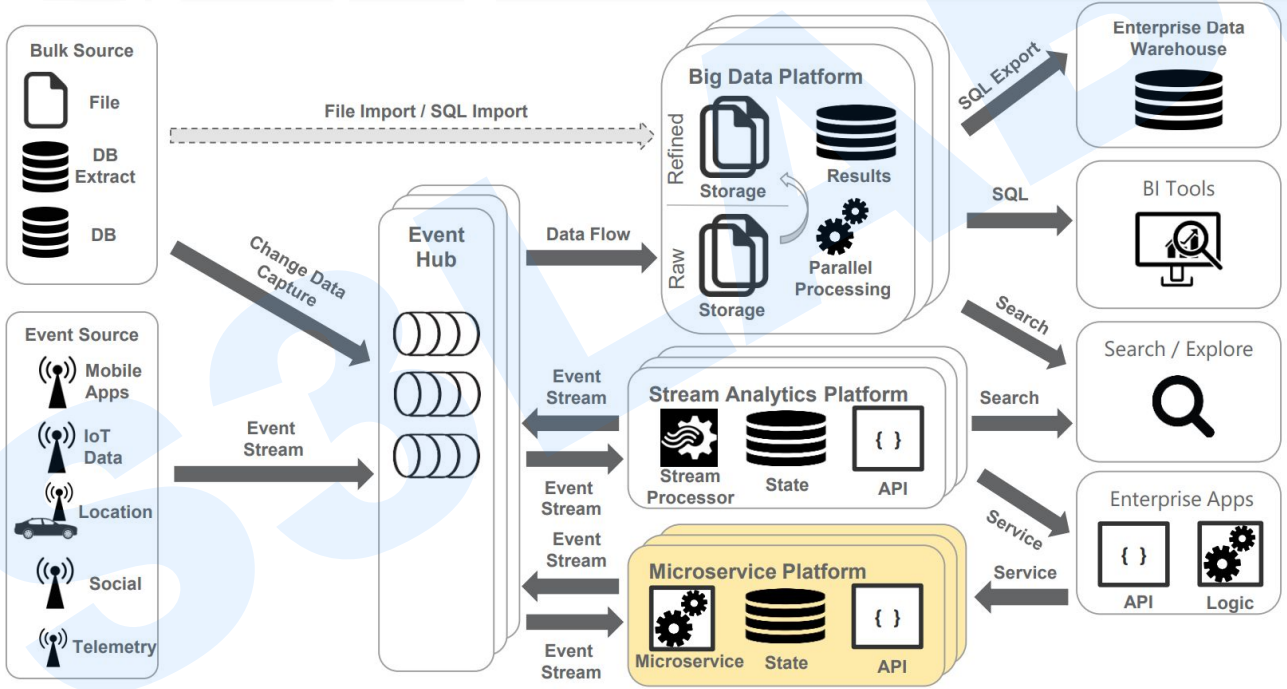
Stream Processing Architecture

Integrate existing systems through CDC



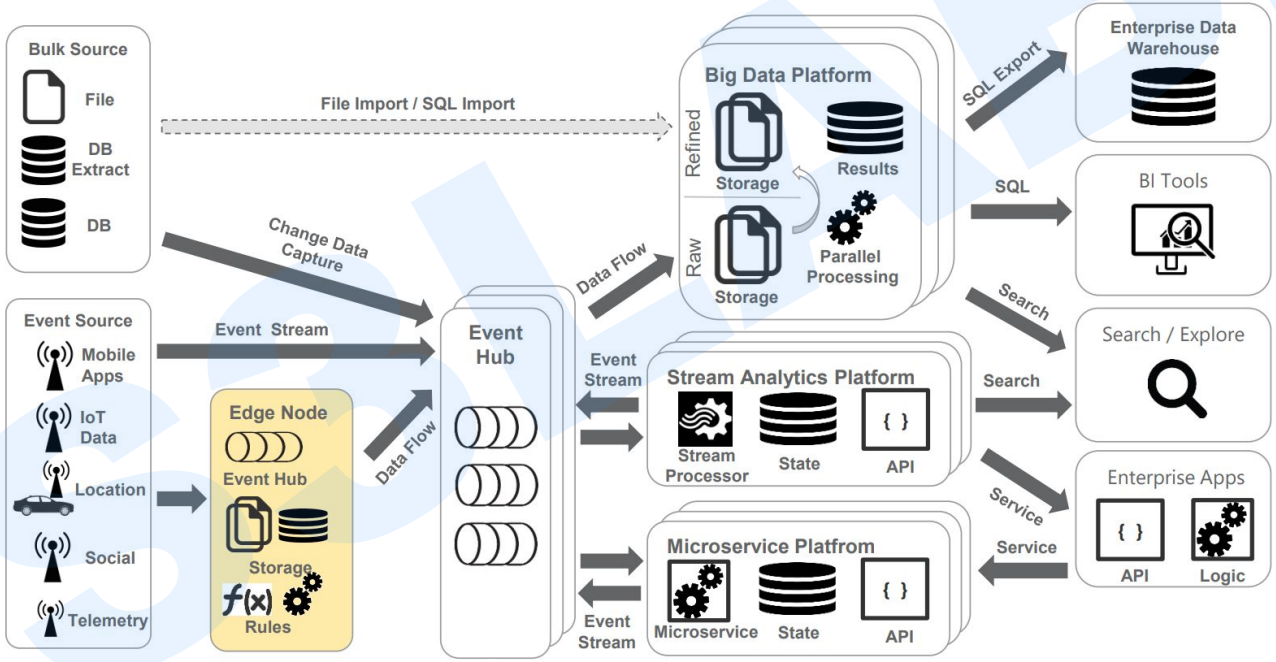
Stream Processing Architecture

Event-oriented fashion



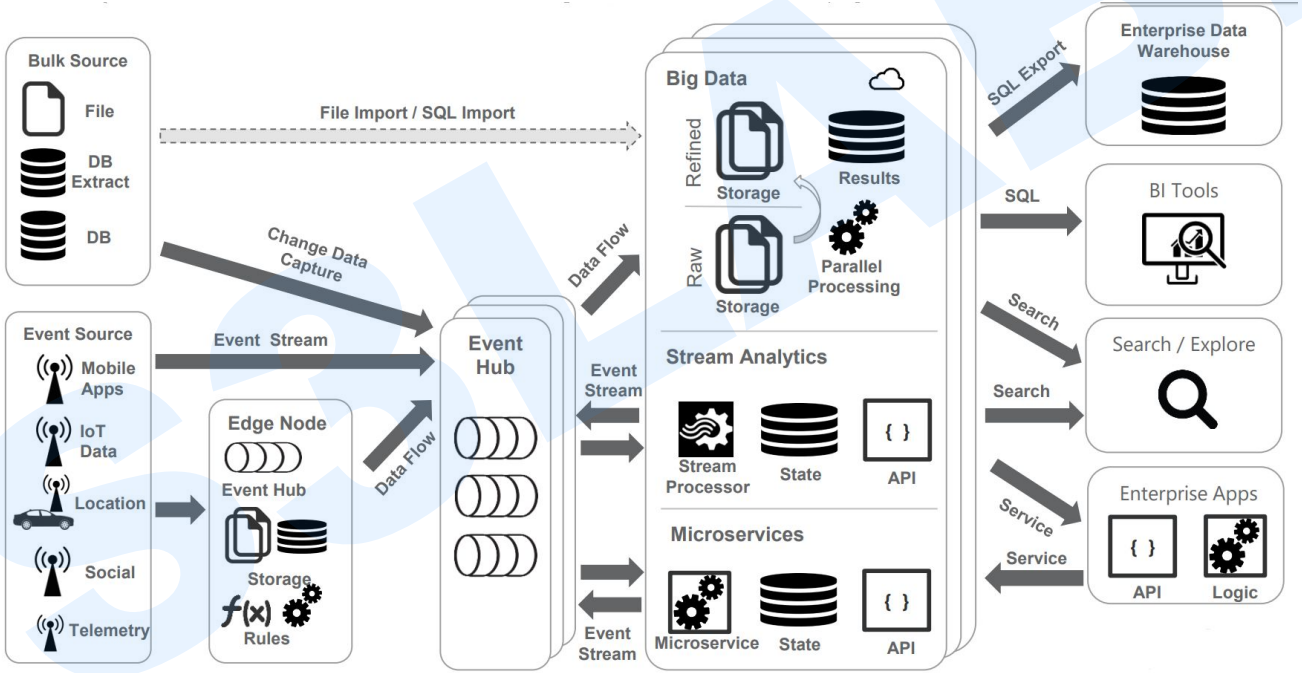
Stream Processing Architecture

Edge computing allows processing close to data sources



Stream Processing Architecture

Unified Architecture for Modern Data Analytics Solutions



Stream Analytics Patterns

Pattern 1: Preprocessing

- **Preprocessing** is often done as a projection from one data stream to the other or through filtering. Potential operations include
  - Filtering and removing some events
  - Reshaping a stream by removing, renaming, or adding new attributes to a stream
  - Splitting and combining attributes in a stream
  - Transforming attributes
- **Example**: from a twitter data stream, we might choose to extract the fields: author, timestamp, location, and then filter them based on the location of the author.

Stream Analytics Patterns

Pattern 2: Alerts and Thresholds

- Detects a condition and generates alerts based on a condition. (e.g. Alarm on high temperature). These alerts can be based on a simple value or more complex conditions such as rate of increase etc.
- Examples:
  - in TFL (Transport for London), we trigger a speed alert when the bus has exceed a given speed limit.
  - We can generate alerts for scenarios such as the server room temperature is continually increasing for last 5 mins.

Stream Analytics Patterns

Pattern 3: Simple Counting and Counting with Windows

- This pattern includes aggregate functions like Min, Max, Percentiles etc, and they can be counted without storing any data. (e.g. counting the number of failed transactions).
- Counts are often used with a **time window** attached to it. (e.g. failure count last hour).

Stream Analytics Patterns

Pattern 3: Simple Counting and Counting with Windows

- There are four main variations:
  - **Time, Sliding window**: keeps each event for the given time window, produce an output whenever a new event has added or removed.
  - **Time, Batch window**: also called tumbling windows, they only produce output at the end of the time window
  - **Length, Sliding**: same as the **time, sliding window**, but keeps a window of n events instead of selecting them by time.
  - **Length, Batch window**: same as the time, batch window, but keeps a window of n events instead of selecting them by time

Stream Analytics Patterns

Pattern 4: Joining Event Streams

- Match up multiple data streams and create a new event stream
- Example:
  - Let's assume we play a football game with both the players and the ball having sensors that emit events with current location and acceleration. We can use "joins" to detect when a player has kicked the ball. To that end, we can join the ball location stream and the player stream on the condition that they are close to each other by one meter and the ball's acceleration has increased by more than 55m/s^2.

Stream Analytics Patterns

Pattern 5: Data Correlation, Missing Events, and Erroneous Data

- Match up multiple streams and correlate the data within the same stream. This is because different data sensors can send events at different rates, and many use cases require this fundamental operator.
- Scenarios:
  - Matching up two data streams that send events at different speeds
  - Detecting a missing event in a data stream (e.g. detect a customer request that has not been responded within 1 hour of its reception.)
  - Detecting erroneous data (e.g. Detect failed sensors using a set of sensors that monitor overlapping regions and using those redundant data to find erroneous sensors and removing their data from further processing)

Stream Analytics Patterns

Pattern 6: Interacting with Databases

- Often we need to combine the real-time data against the historical data stored in a disk.
- Examples:
  - When a transaction happened, look up the age using the customer ID from customer database to be used for Fraud detection (enrichment)
  - Checking a transaction against blacklists and whitelists in the database
  - Receive an input from the user (e.g. Daily discount amount may be updated in the database, and then the query will pick it automatically without human intervention.)

Stream Analytics Patterns

Pattern 7: Detecting Temporal Event Sequence Patterns

- Using regular expressions with strings, we detect a pattern of characters from a sequence of characters. Similarly, given a sequence of events, we can write a regular expression to detect a temporal sequence of events arranged on time where each event or condition about the event is parallel to a character in a string in the above example.
- Examples:
  - Detect the ball possession, the time period a specific player controlled the ball. A player possessed the ball from the time he hits the ball until someone else hits the ball. This condition can be written as a regular expression: a hit by me, followed by any number of hits by me, followed by a hit by someone else.

Stream Analytics Patterns

Pattern 8: Tracking

- Tracks something over space and time and detects given conditions.
- Examples:
  - Tracking a fleet of vehicles, making sure that they adhere to speed limits, routes, and geo-fences.
  - Tracking wildlife, making sure they are alive (they will not move if they are dead) and making sure they will not go out of the reservation.
  - Tracking airline luggage and making sure they are not been sent to wrong destinations
  - Tracking a logistic network and figure out bottlenecks and unexpected conditions.

Stream Analytics Patterns

Pattern 9: Detecting Trends

- We often encounter time series data. Detecting patterns from time series data and bringing them into operator attention are common use cases.
- Some Type of Trends:
  - Rise, Fall, Turn (switch from a rise to a fall), Outliers, Complex trends like triple bottom etc.
- Examples:
  - Stock markets and Algorithmic trading
  - Enforcing SLA (Service Level Agreement), Auto Scaling, and Load Balancing
  - Predictive maintenance (e.g. guessing the Hard Disk will fill within next week).

Stream Analytics Patterns

Pattern 10: Running the same Query in Batch and Realtime Pipelines

- It is often used to fill the gap left in the data due to batch processing. For example, if batch processing takes 15 minutes, results would lack the data for last 15 minutes.
- The idea of this pattern, which is sometimes called “Lambda Architecture” is to use real-time analytics to fill the gap.

Stream Analytics Patterns

Pattern 11: Detecting and switching to Detailed Analysis

- Detect a condition that suggests some anomaly, and further analyze it using historical data. This pattern is used with the use cases where we cannot analyze all the data with full detail. Instead, we analyze anomalous cases in full detail.
- Examples:
  - Use basic rules to detect Fraud (e.g. large transaction), then pull out all transactions done against that credit card for a larger time period (e.g. 3 months data) from a batch pipeline and run a detailed analysis
  - While monitoring weather, detect conditions like high temperature or low pressure in a given region and then start a high resolution localized forecast on that region.
  - Detect good customers, for example through the expenditure of more than $1000 within a month, and then run a detailed model to decide the potential of offering a deal.

Stream Analytics Patterns

Pattern 12: Using model

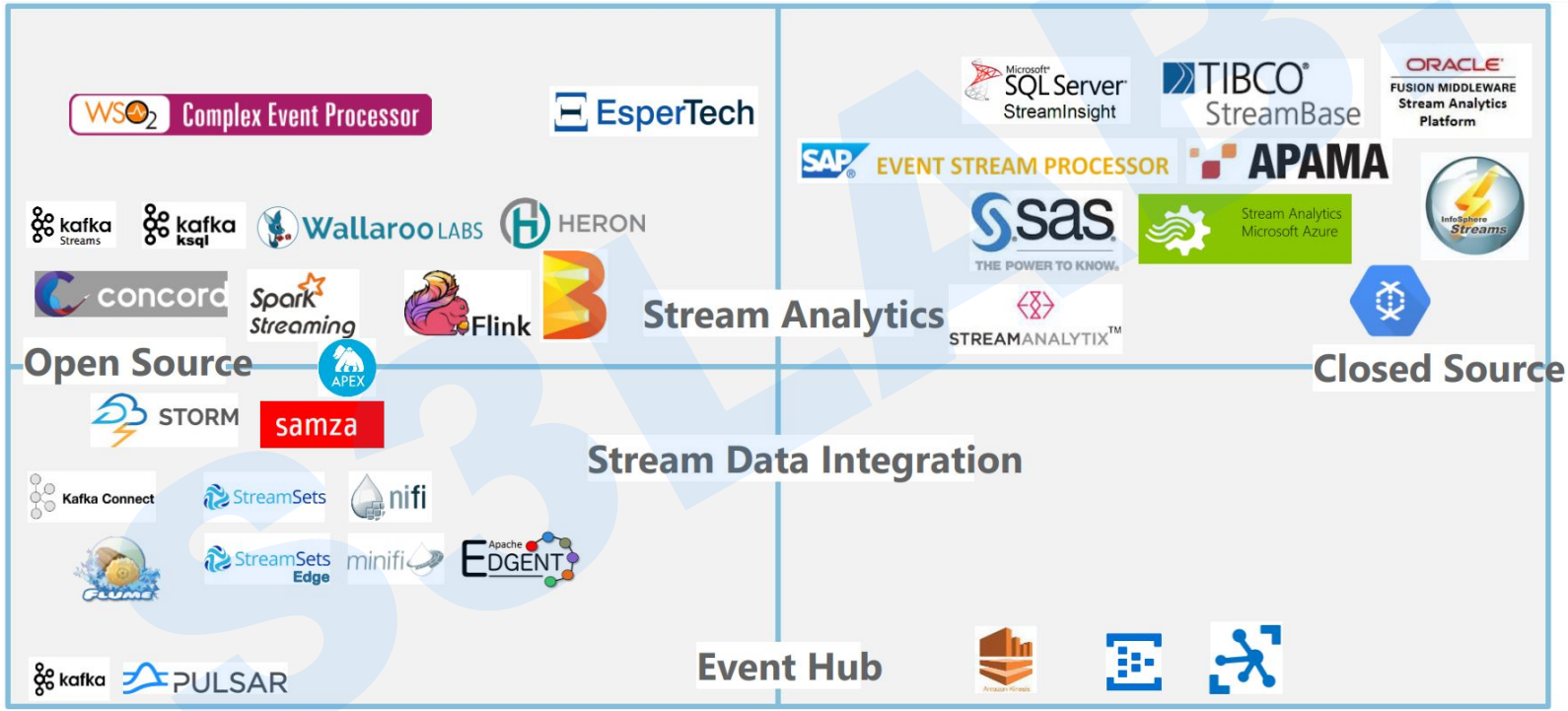
- The idea is to train a model (often a Machine Learning model), and then use it with the Real-time pipeline to make decisions. For example, you can build a model using R, export it as PMML (Predictive Model Markup Language) and use it within your realtime pipeline.
- Examples:
  - Fraud Detections, Segmentation, Predict next value, Predict Churn

Stream Analytics Patterns

Pattern 13: Online Control

- There are many use cases where we need to control something online. The classical use cases are the autopilot, self-driving, and robotics. These would involve problems like current situation awareness, predicting next value(s), and deciding on corrective actions.

Stream Processing & Analytics Ecosystem



Q & A

Cảm ơn đã theo dõi

Chúng tôi hy vọng cùng nhau đi đến thành công.