Introduction to Stream Processing

Guido Schmutz
Frankfurt - 21.2.2019

FRANKFURTER ENTWICKLERTAG

@gschmutz guidoschmutz.wordpress.com

Agenda

1. Motivation for Stream Processing?
2. Capabilities for Stream Processing
3. Implementing Stream Processing Solutions
4. Demo
5. Summary

Guido Schmutz

Working at Trivadis for more than 22 years

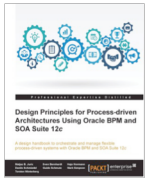
Oracle Groundbreaker Ambassador & Oracle ACE Director

Consultant, Trainer Software Architect for Java, Oracle, SOA and Big Data / Fast Data

Head of Trivadis Architecture Board

Technology Manager @ Trivadis

More than 30 years of software development experience



Contact: guido.schmutz@trivadis.com

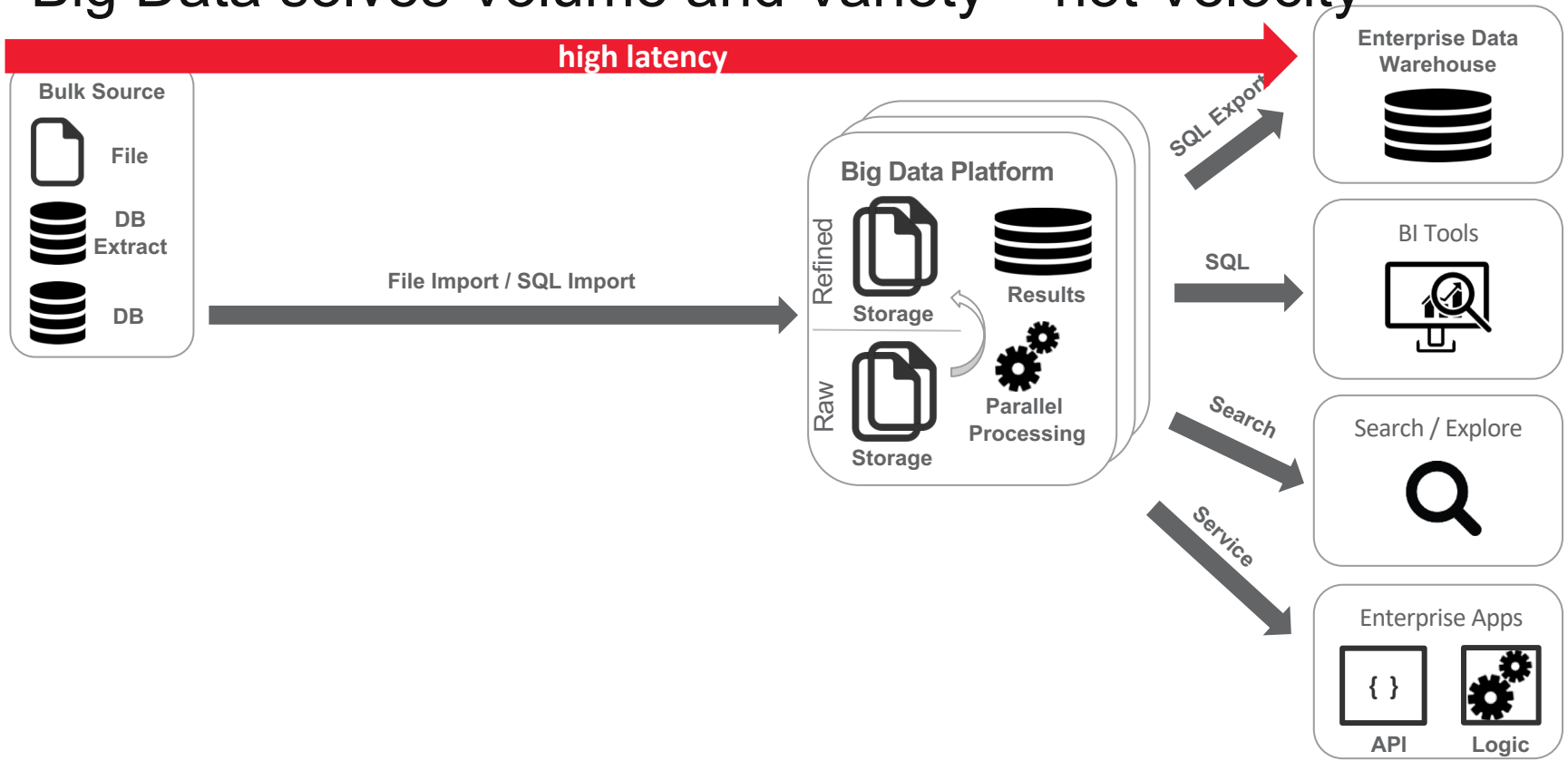
Blog: http://guidoschmutz.wordpress.com

Slideshare: http://www.slideshare.net/gschmutz

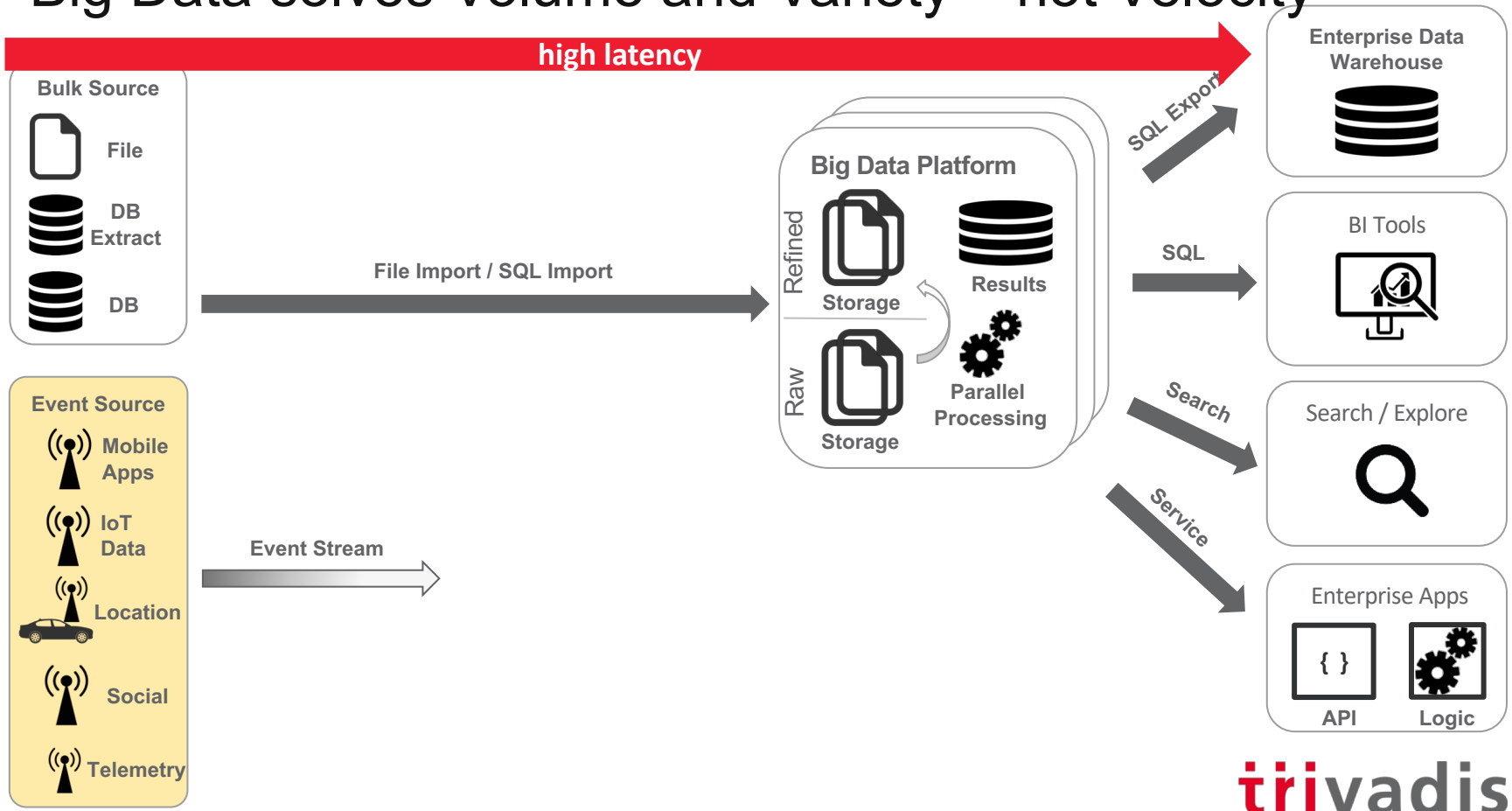
Twitter: gschmutz

Motivation for Stream Processing?

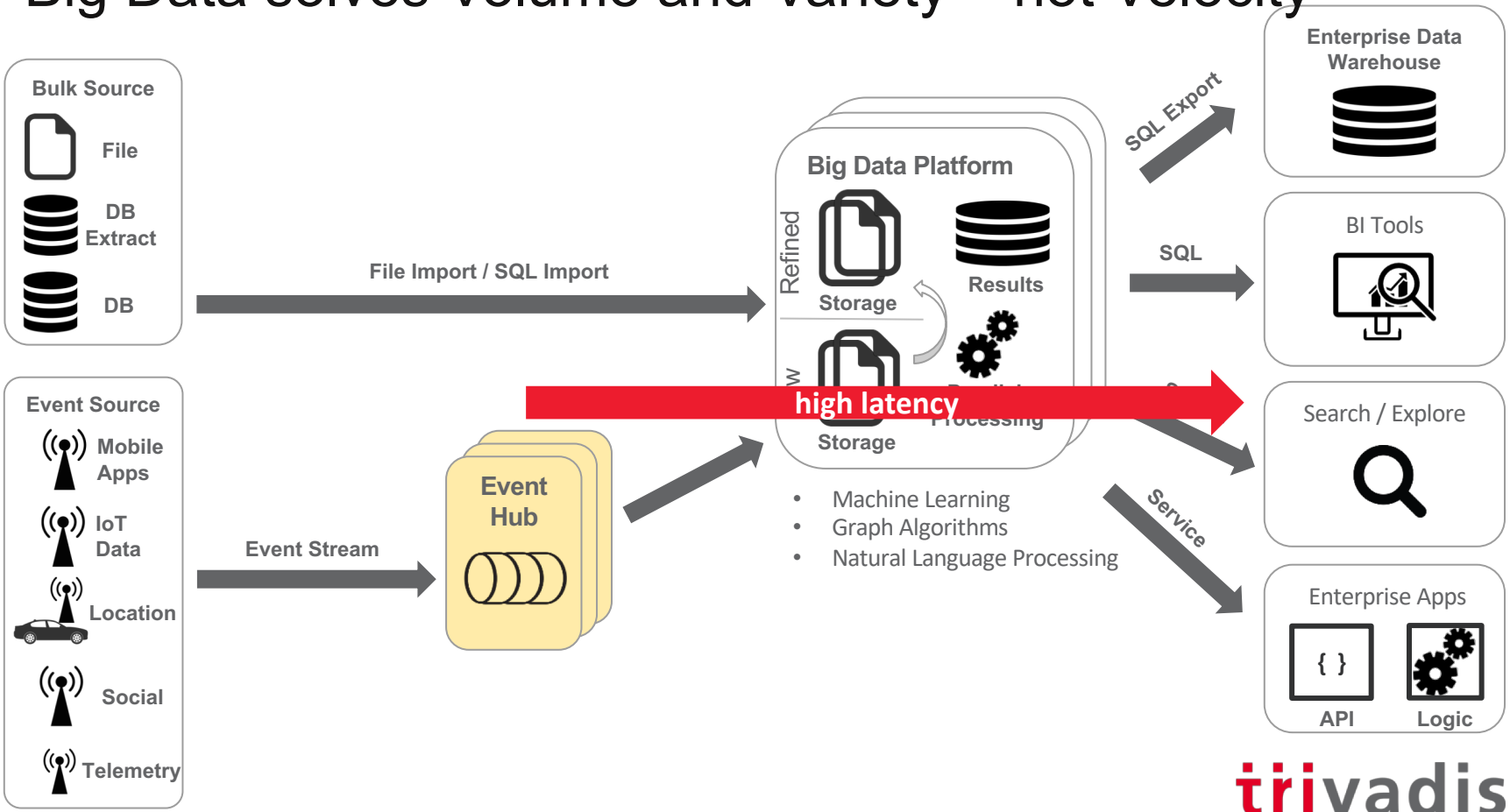
Big Data solves Volume and Variety – not Velocity



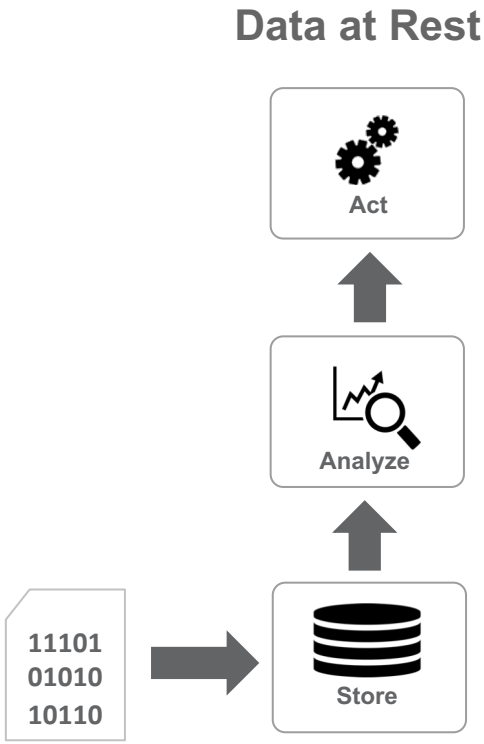
Big Data solves Volume and Variety – not Velocity

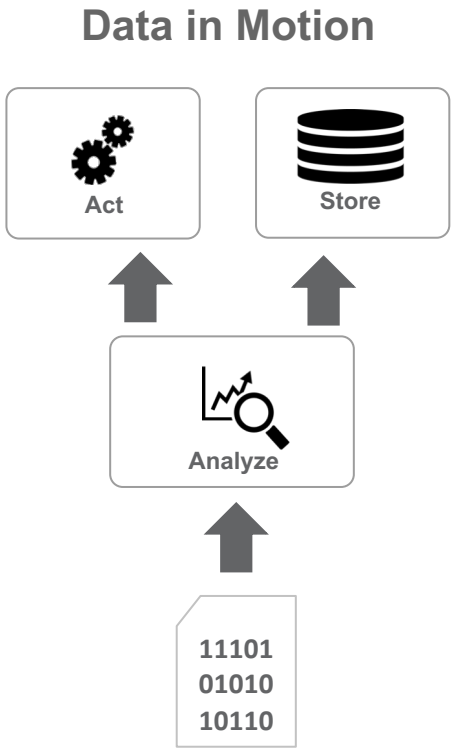


Big Data solves Volume and Variety – not Velocity

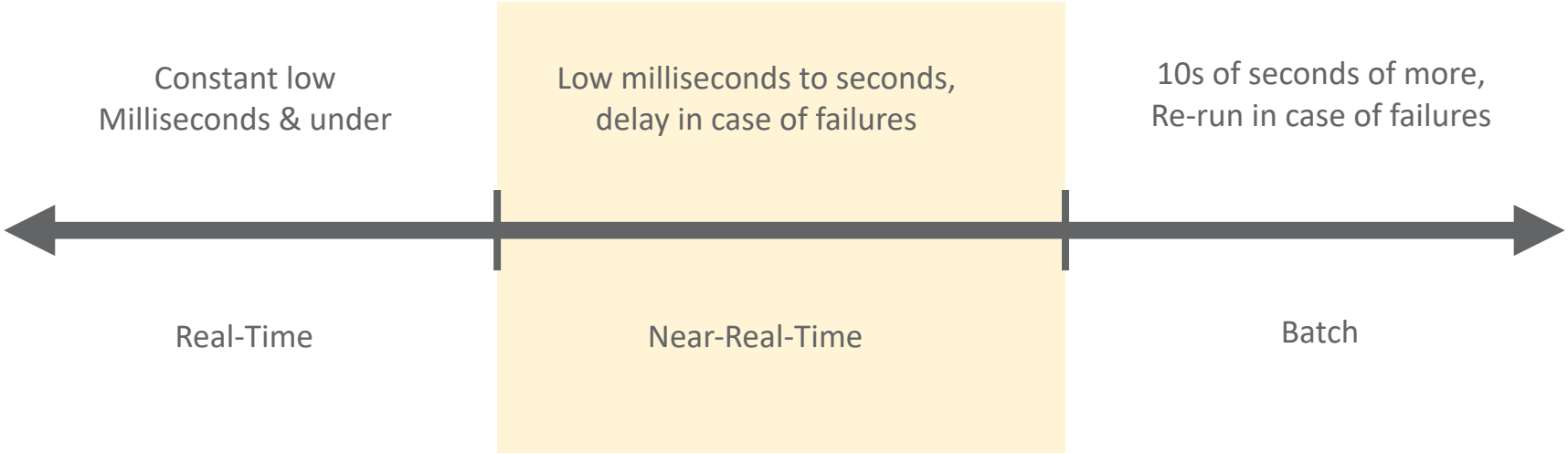


"Data at Rest" vs. "Data in Motion"



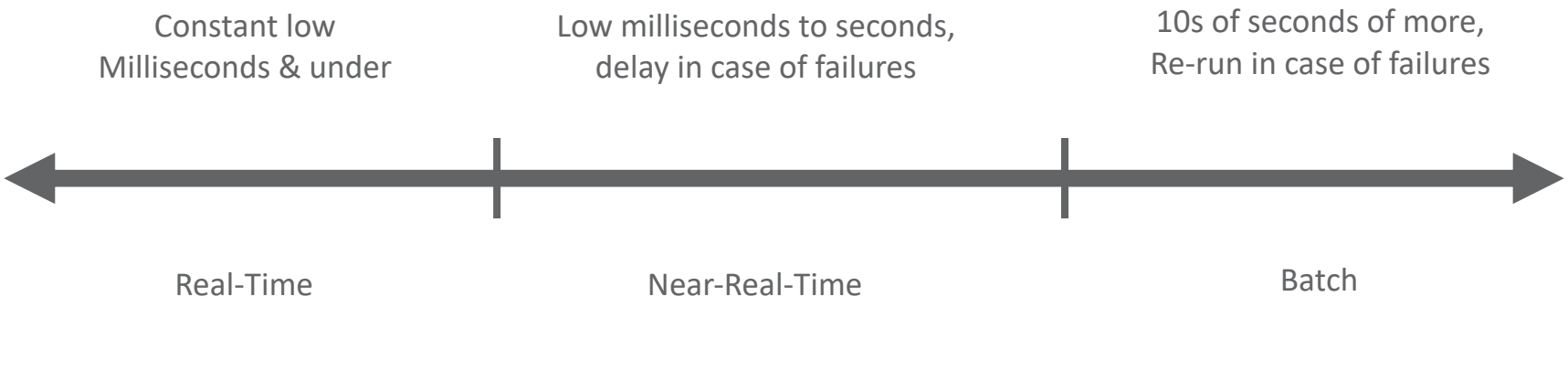


When to Stream / When not?

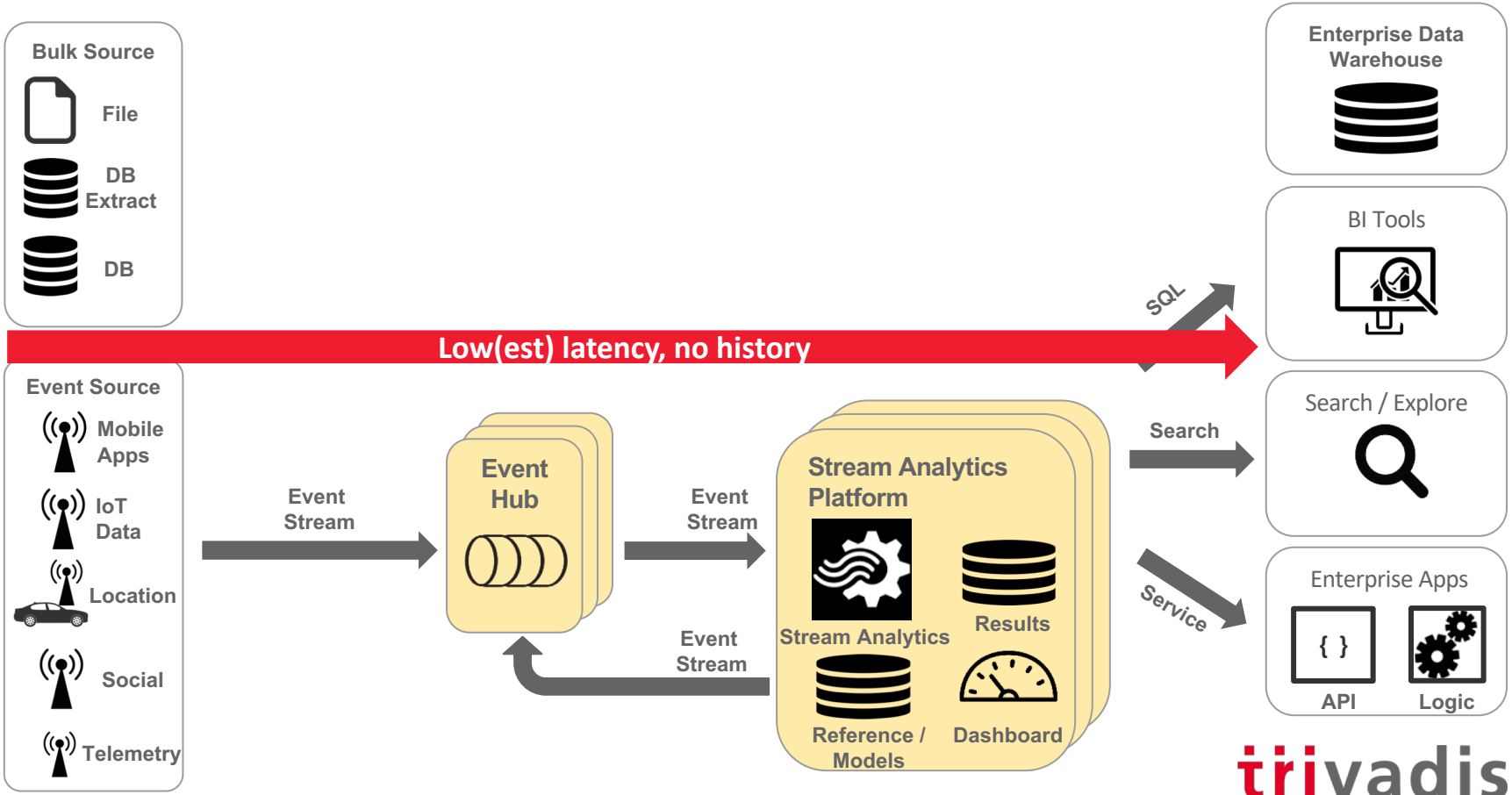


Source: adapted from Cloudera

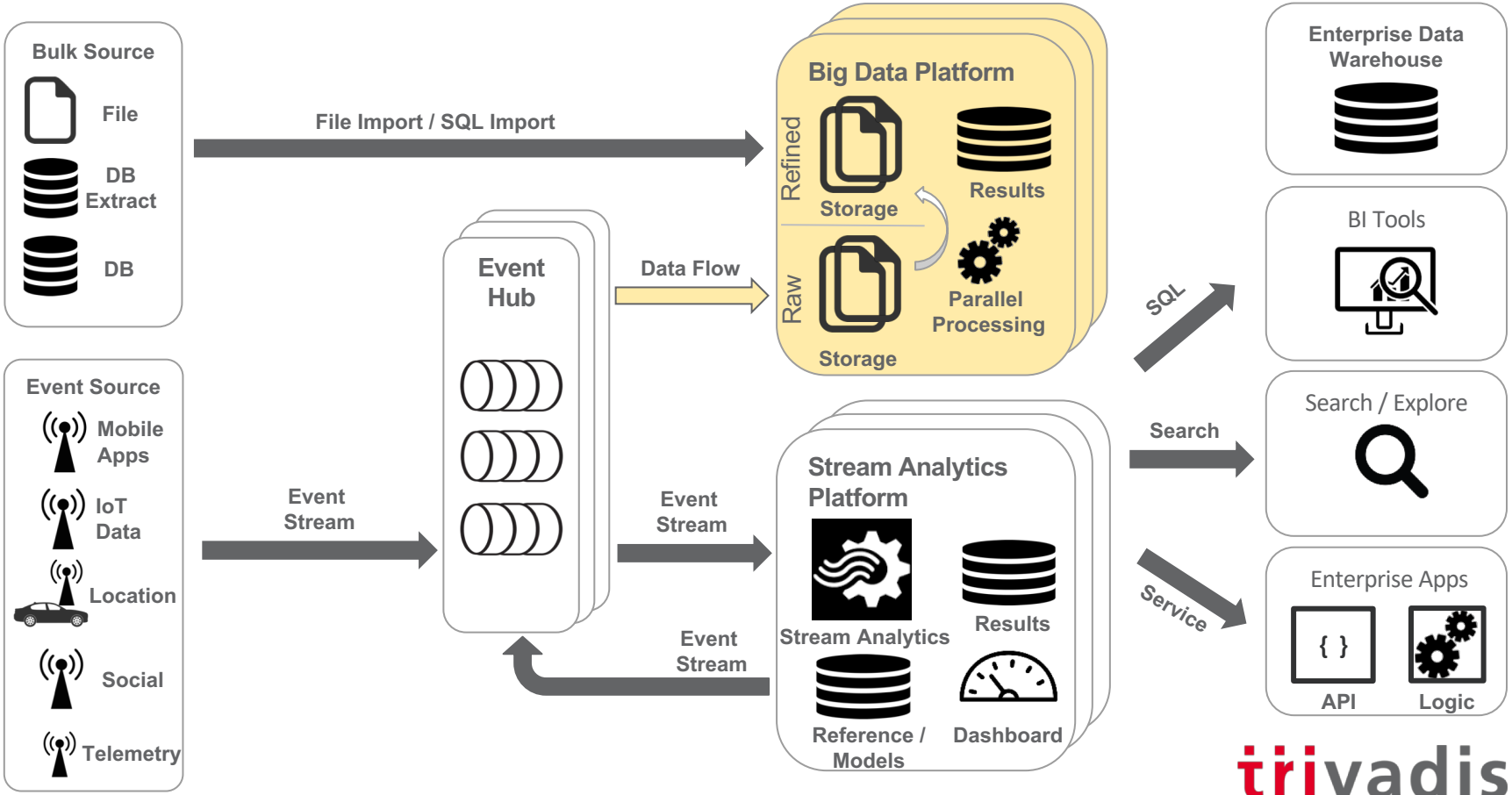
"No free lunch"



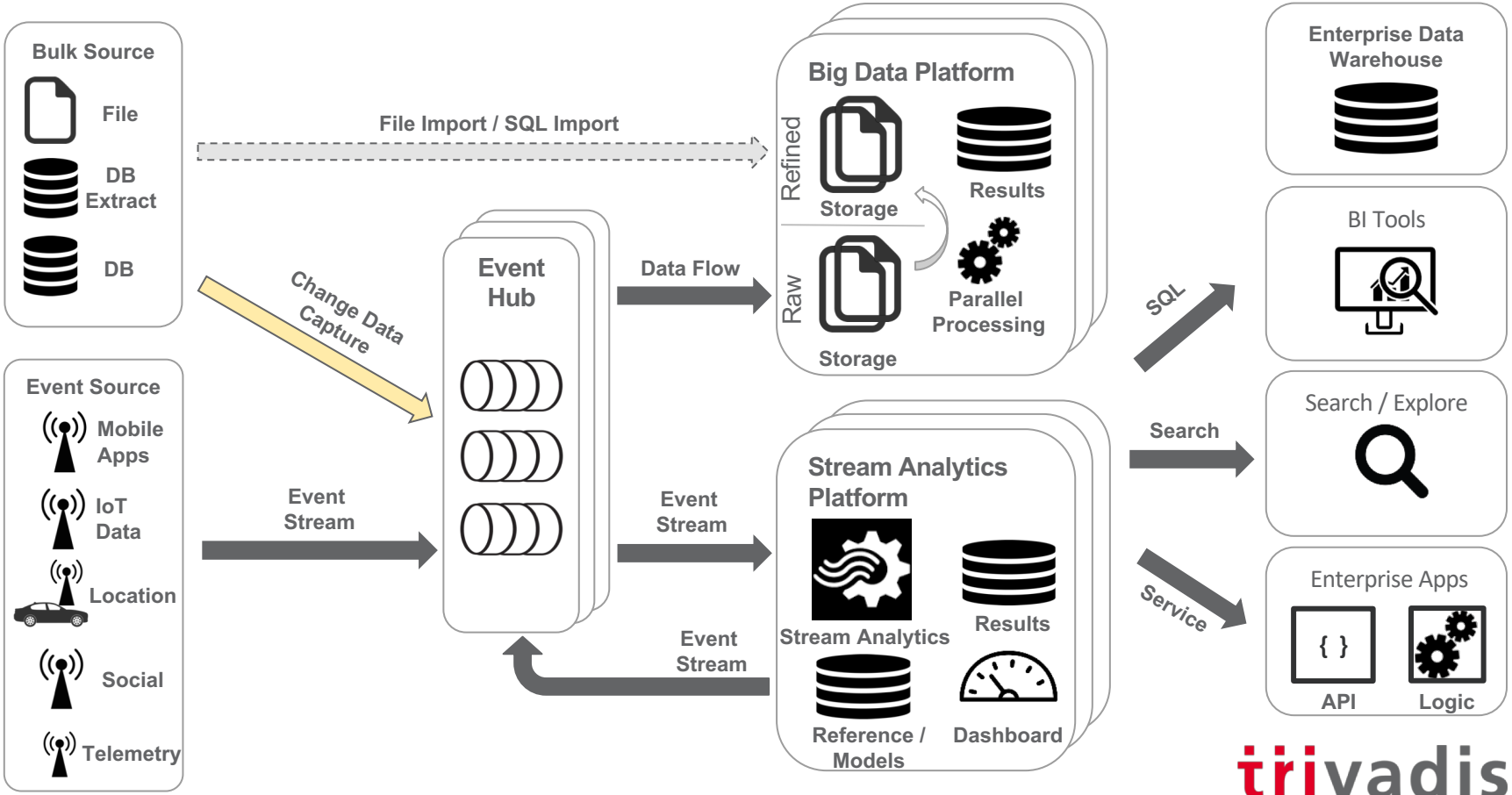
Stream Processing Architecture solves Velocity



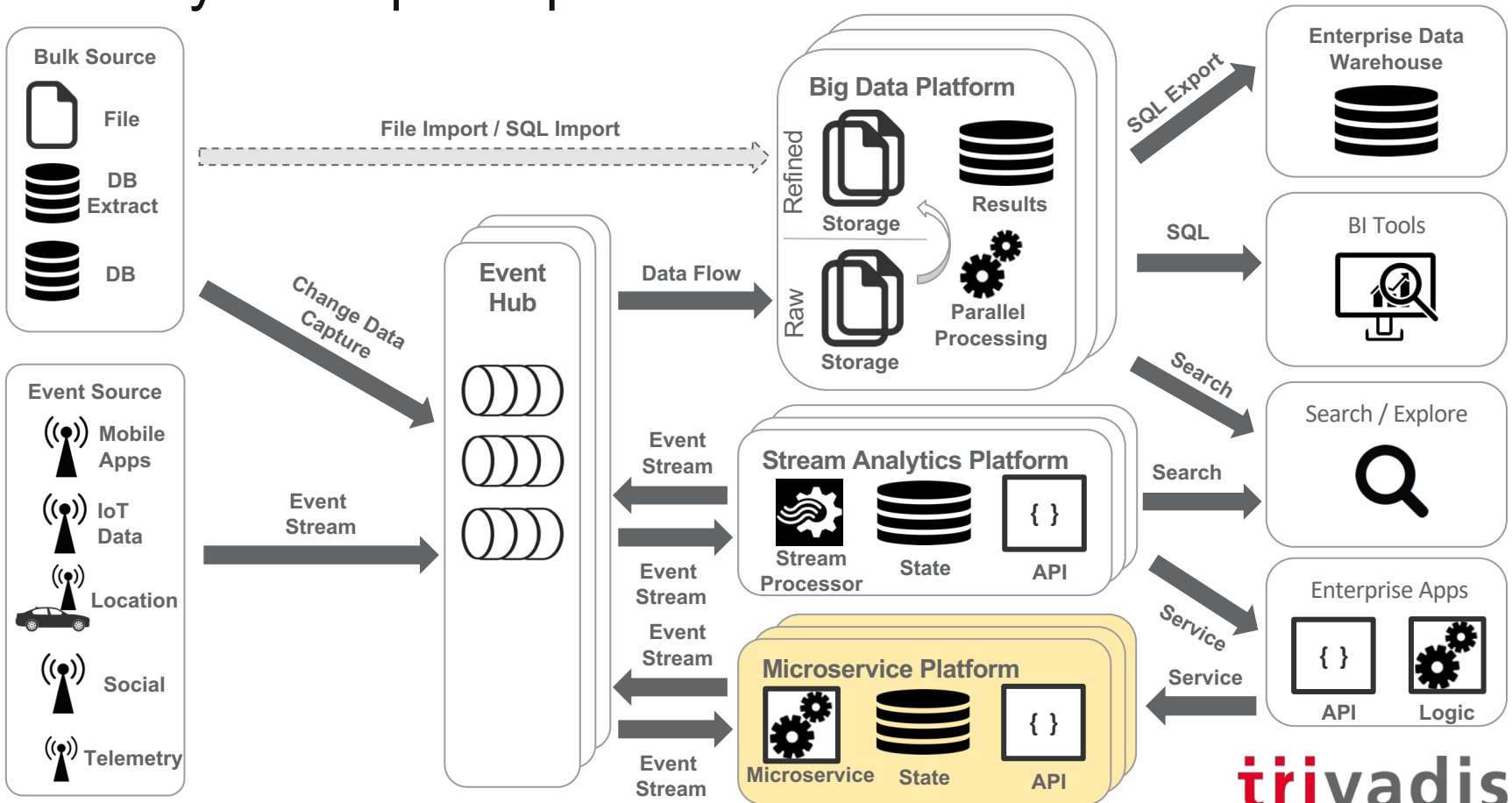
Big Data for all historical data analysis



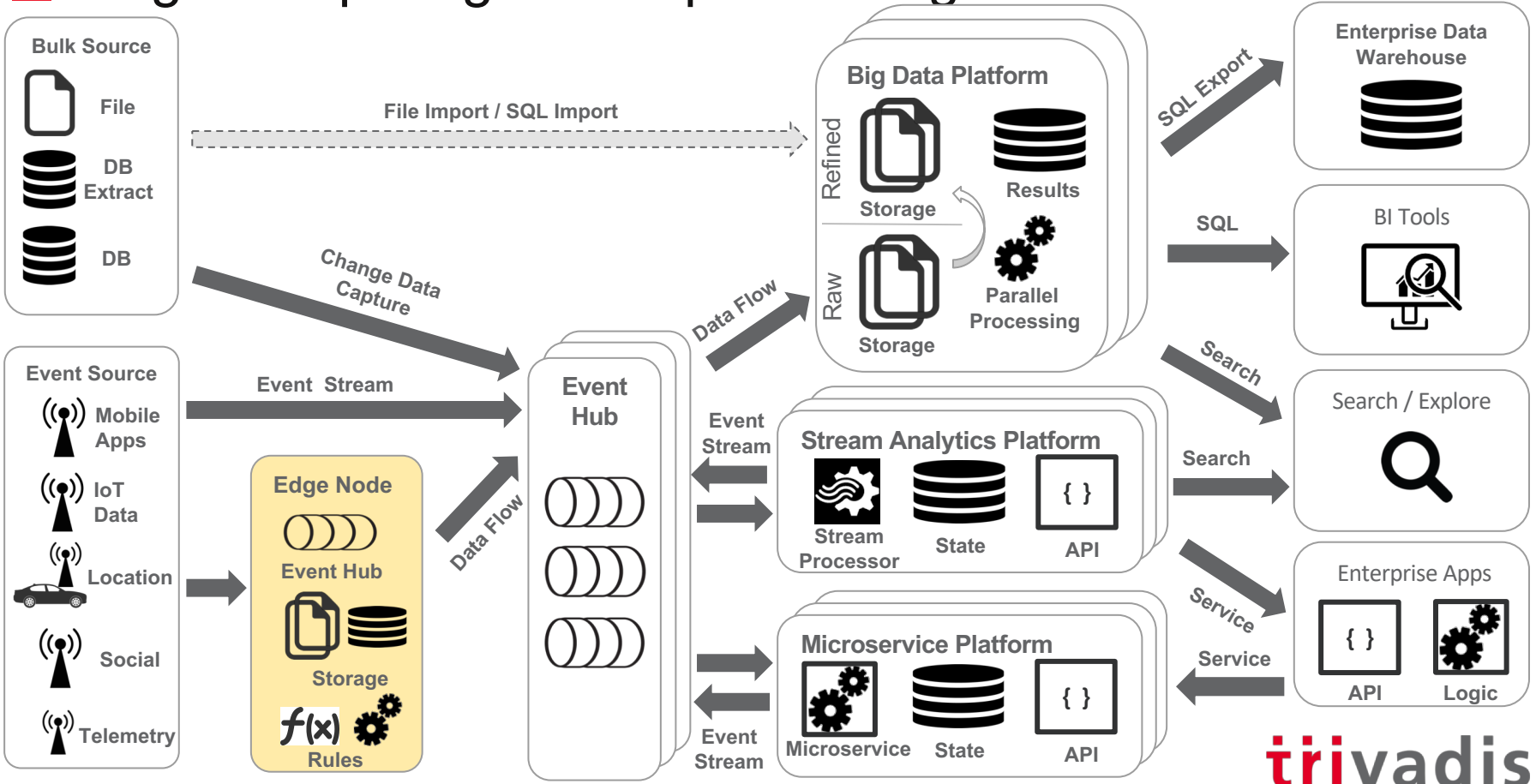
Integrate existing systems with lower latency through CDC



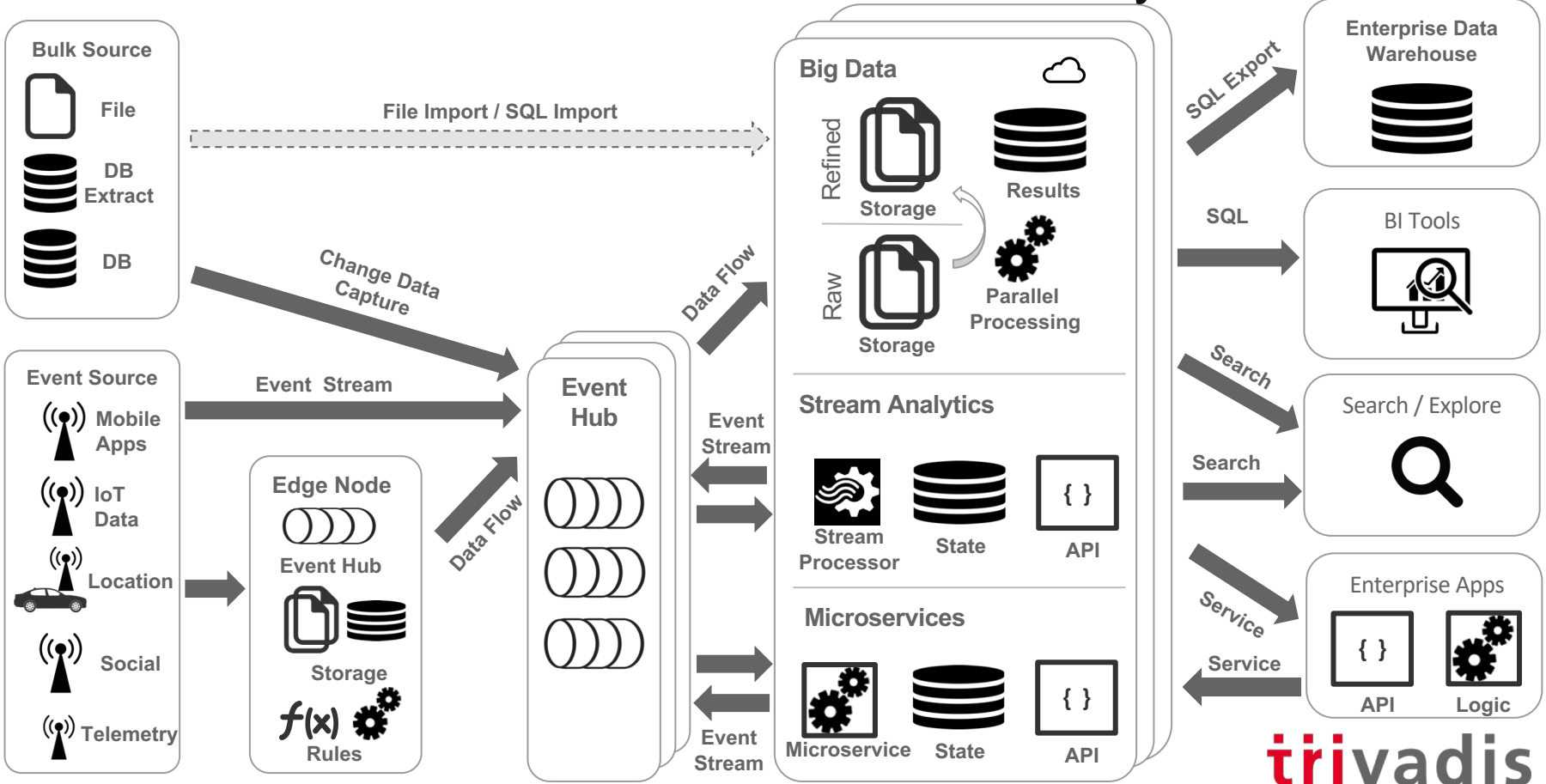
New systems participate in event-oriented fashion



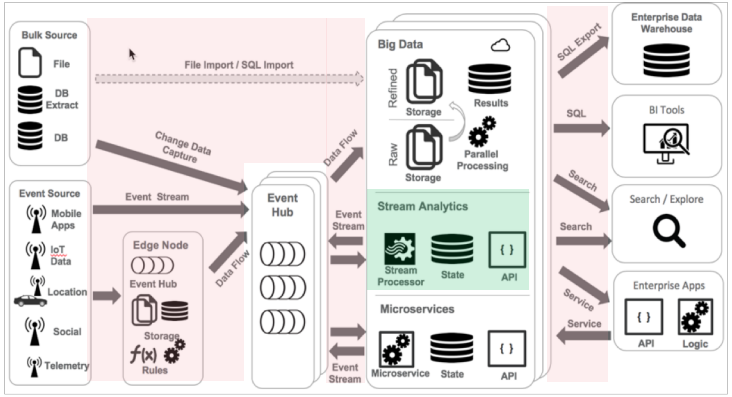
Edge computing allows processing close to data sources



Unified Architecture for Modern Data Analytics Solutions



Two Types of Stream Processing (by Gartner)



Stream Data Integration

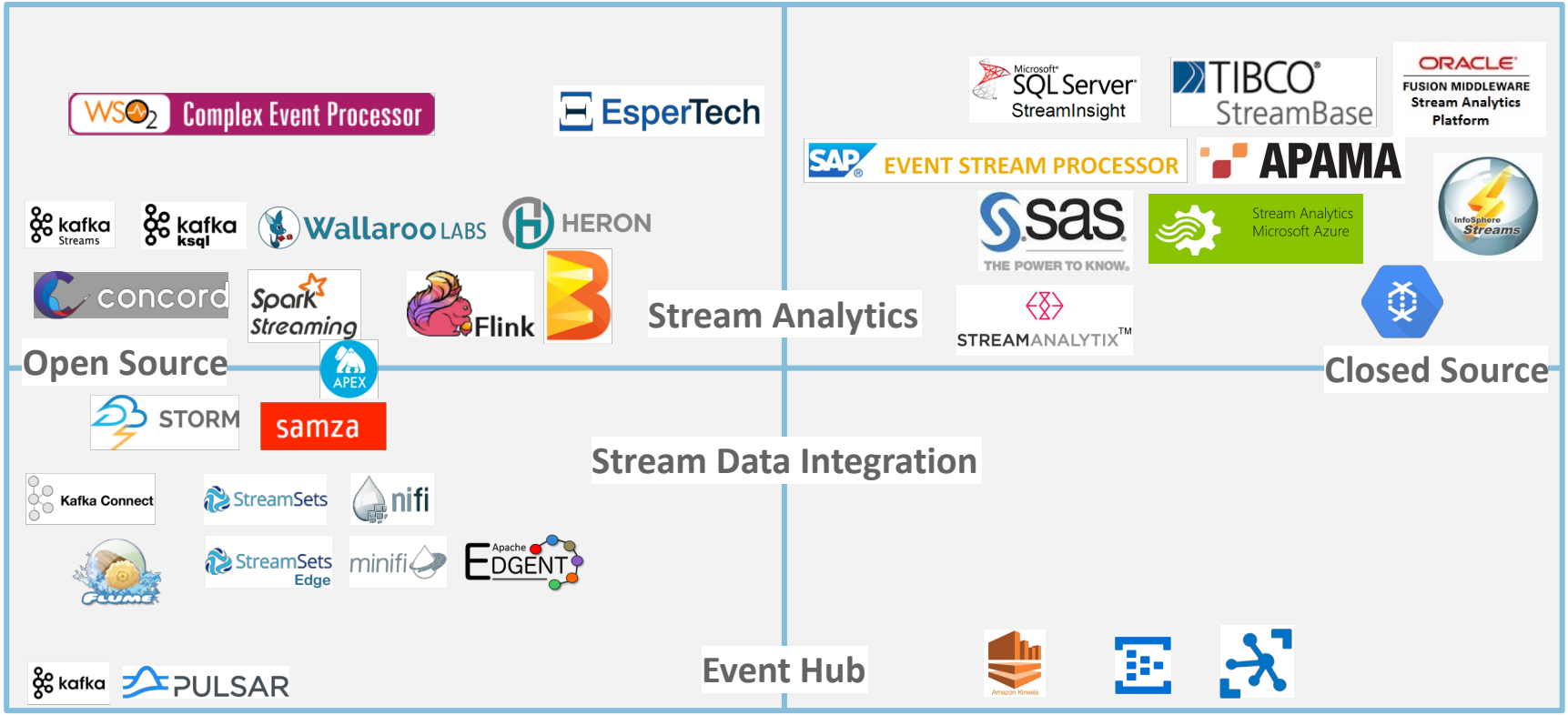
- focuses on the ingestion and processing of data sources targeting real-time extract-transform-load (ETL) and data integration use cases
- filter and enrich the data

Stream Analytics

- targets analytics use cases
- calculating aggregates and detecting patterns to generate higher-level, more relevant summary information (complex events)
- Complex events may signify threats or opportunities that require a response from the business

Gartner: Market Guide for Event Stream Processing, Nick Heudecker, W. Roy Schulte

Stream Processing & Analytics Ecosystem



Source: adapted from Tibco

Stream vs. Table / Static

Stream

"History"

an unbounded sequence of structured data ("facts")

Facts in a stream are **immutable**

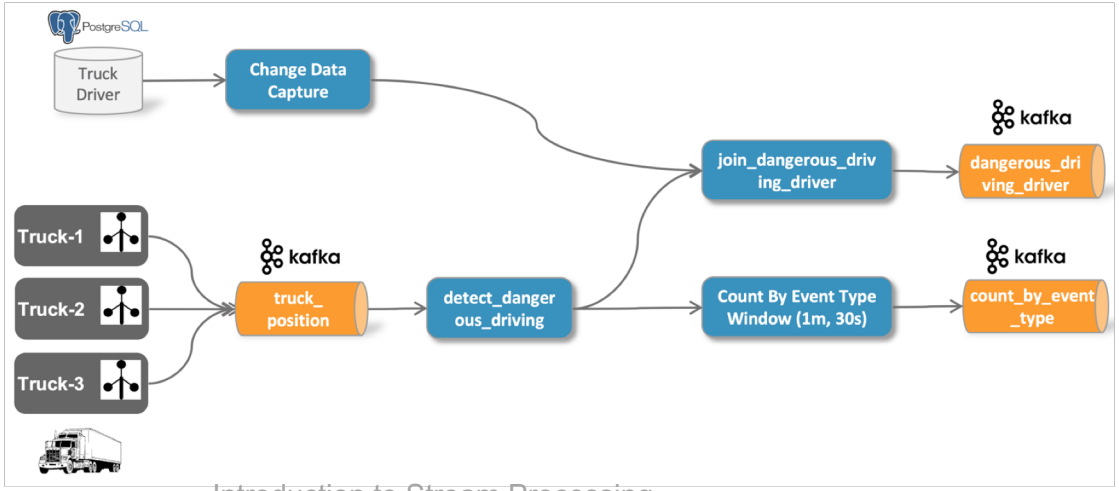
Table / Static

"State"

a view of a stream, or another table, and represents a collection of evolving facts

Latest value for each key in a stream

Facts in a table are **mutable**

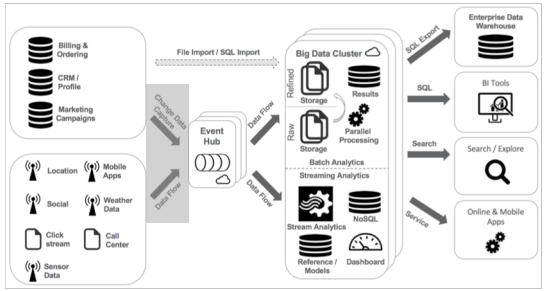


Important Capabilities for Stream Processing

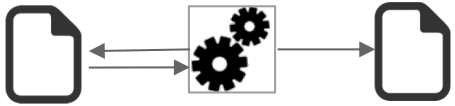
Capabilities: Stream Data Integration vs. Stream Analytics

	Stream Data Integration	Stream Analytics
Support for Various Data Sources	yes	partial
Streaming ETL (Transformation/Format Translation, Routing, Validation)	yes	partial
Micro-Batching	yes	partial
Event-at-a-time	yes	yes
Delivery Guarantees	yes	yes
API : GUI-Based API / Declarative API / Programmatic	yes	yes
API: Streaming SQL	-	yes
Event Time vs. Ingestion / Processing Time	-	yes
Windowing	-	yes
Stream-to-Static Joins (Lookup/Enrichment)	partial	yes
Stream-to-Stream Joins	-	yes
State Management	-	yes
Queryable State (aka Interactive Queries)	-	yes
Event Pattern Detection	-	Yes

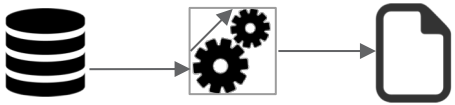
Integrating Data Sources



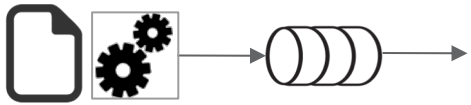
File Polling



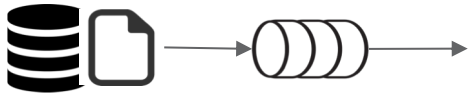
SQL Polling



File Stream (File Tailing)

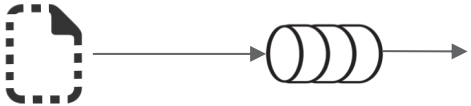


Change Data Capture (CDC)



Sensor Stream

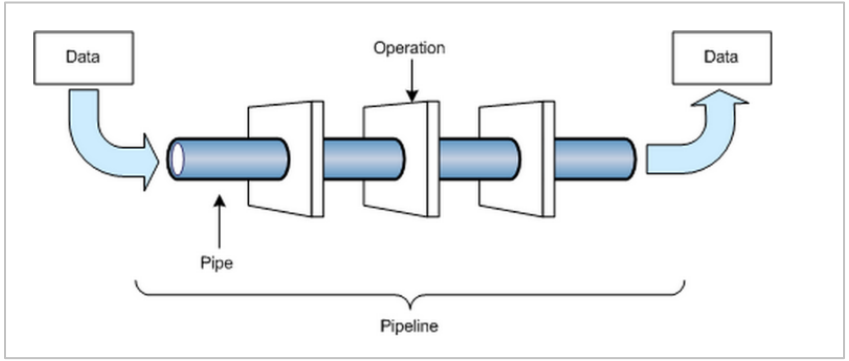
File Stream (Appender)



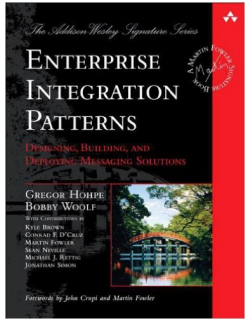
Streaming ETL

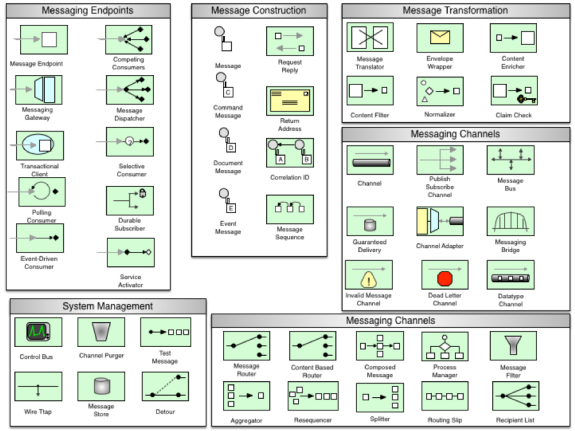
- Flow-based ”programming”
- Ingest Data from various sources
- **E**xtract – **T**ransform – **L**oad
- High-Throughput, straight-through data flows
- Data Lineage
- Batch- or Stream-Processing
- Visual coding with flow editor

- Event Stream Processing (ESP) **but not** Complex Event Processing (CEP)



Source: Confluent





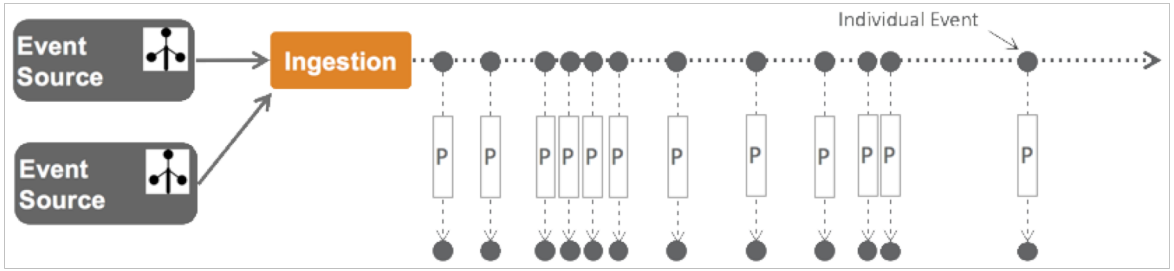
@gschmutz

trivadis makes IT easier.

Event-at-a-time vs. Micro Batch

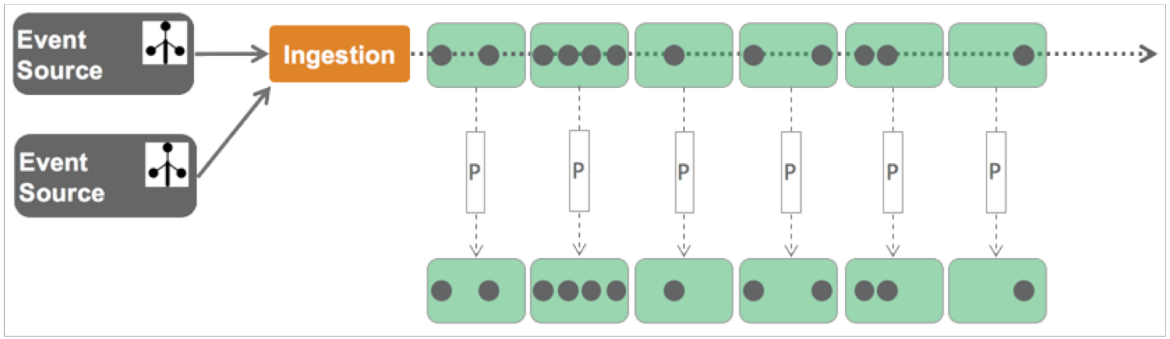
Event-at-a-time Processing

- Events processed as they arrive
- low-latency
- fault tolerance expensive



Micro-Batch Processing

- Splits incoming stream in small batches
- Fault tolerance easier
- Better throughput



Delivery Guarantees

At most once (fire-and-forget)

message is sent, but the sender **doesn't care if it's received or lost**. it is *the easiest and most performant* behavior to support.

At least once

retransmission of a message will occur until an acknowledgment is received. Since a delayed acknowledgment from a receiver could be in flight when the sender retransmits, the message may be received **one or more times**.

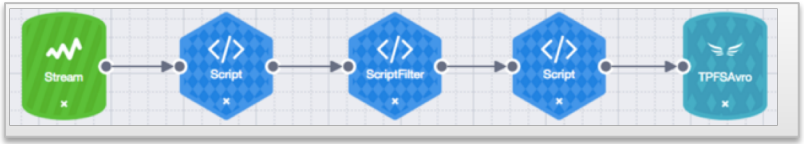
Exactly once

ensures that a message is received ***once and only once***, and is never lost and never repeated. The system must implement whatever mechanisms are required to ensure that a message is received and processed just once

API

GUI-based / Drag-and-Drop

- A graphical way of designing a plipeline
- Often web-based to improve usability



Declarative

- An streaming engine which can be configured in a declarative way
- JSON, YML

```
 "config": {
    "connector.class":
"io.confluent.connect.mqtt.MqttSourceConnector",
    "tasks.max": "1",
    "mqtt.server.uri": "tcp://mosquitto-1:1883",
    "mqtt.topics": "truck/+/position",
    "kafka.topic":"truck_position",
    ...
```

Programmatic

- Low-level (class) or high-level fluent API
- Higher order function as operators (filter, mapWithState …)

```
val filteredDf = truckPosDf.
   where("eventType !='Normal'")
```

Streaming SQL

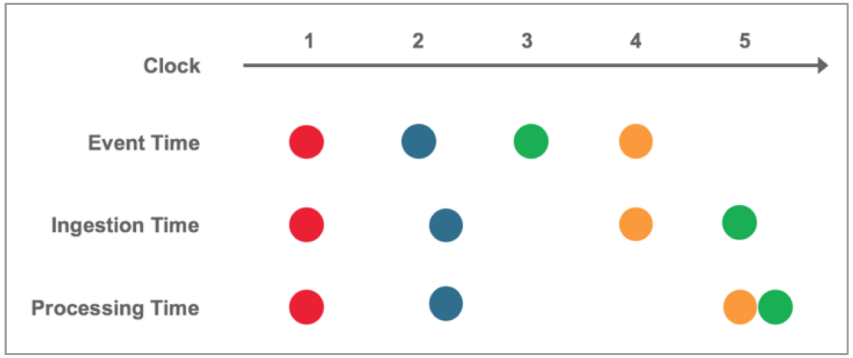
- use a stream in a FROM clause
- Extensions supporting windowing, pattern matching, spatial, …. Operators

```
SELECT * FROM truck_position_s
WHERE eventType != 'Normal'
```

Event Time vs. Ingestion / Processing Time

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

Ingestion time / Processing Time
the time at which events are ***ingested into*** / ***processed by*** the system



Not all use cases care about event times but many do

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

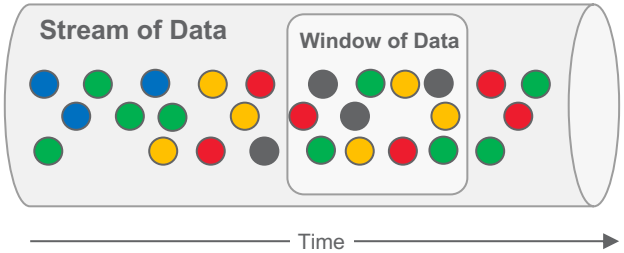
Windowing

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

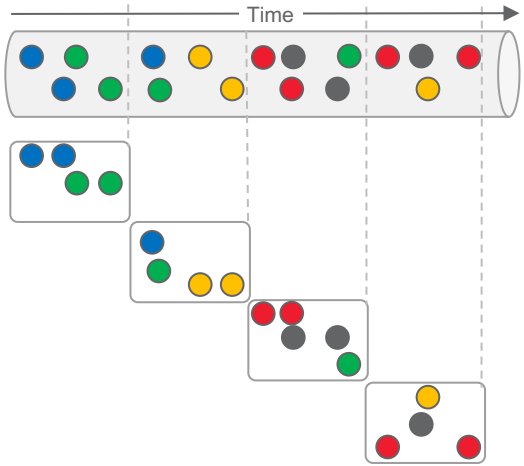
A window of data represents a certain amount of data where we can perform computations on

Windows give the power to keep a working memory and look back at recent data efficiently

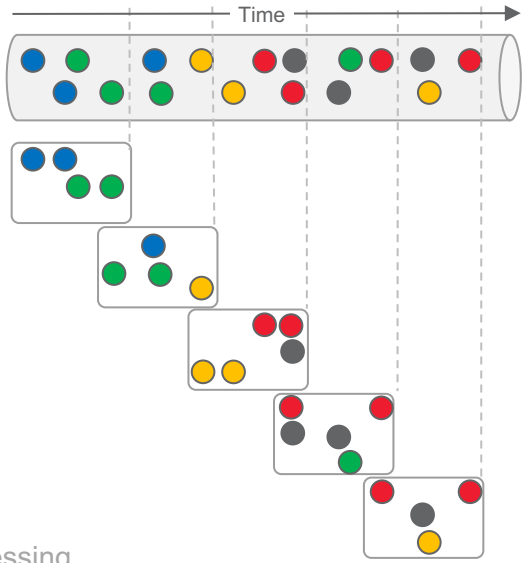


Windowing

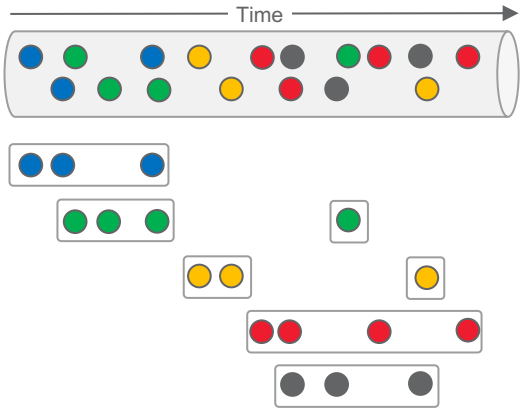
Fixed Window (aka Tumbling Window) - eviction policy always based on the window being full and trigger policy based on either the count of items in the window or time



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

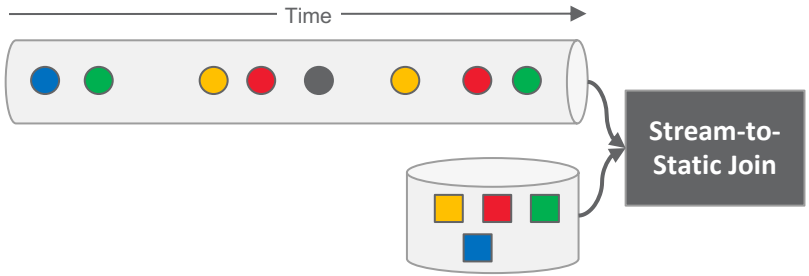


Joining – Stream-to-Static and Stream-to-Stream

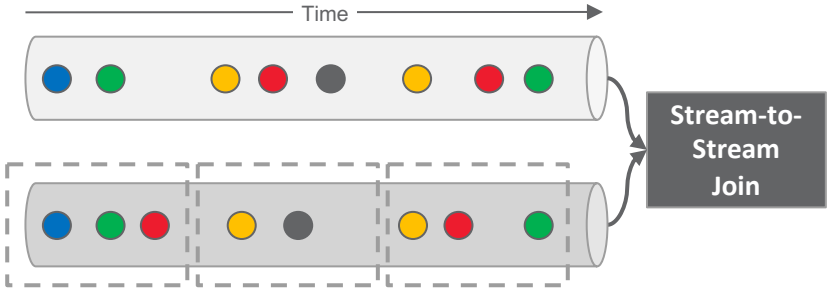
Challenges of joining streams

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

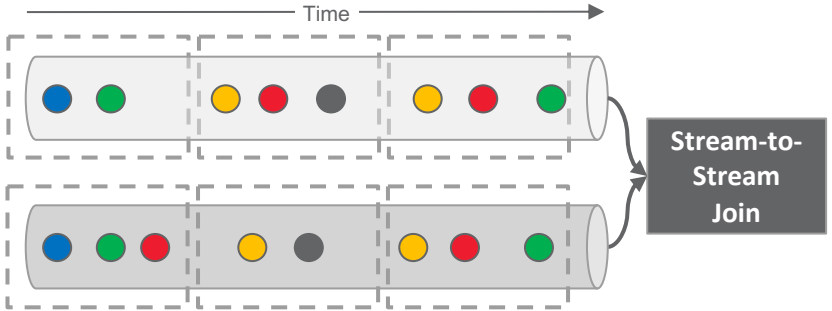
Stream-to-Static (Table) Join



Stream-to-Stream Join (one window join)



Stream-to-Stream Join (two window join)



State Management

Necessary if stream processing use case is dependent on previously seen data or external data

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

Options for State Management

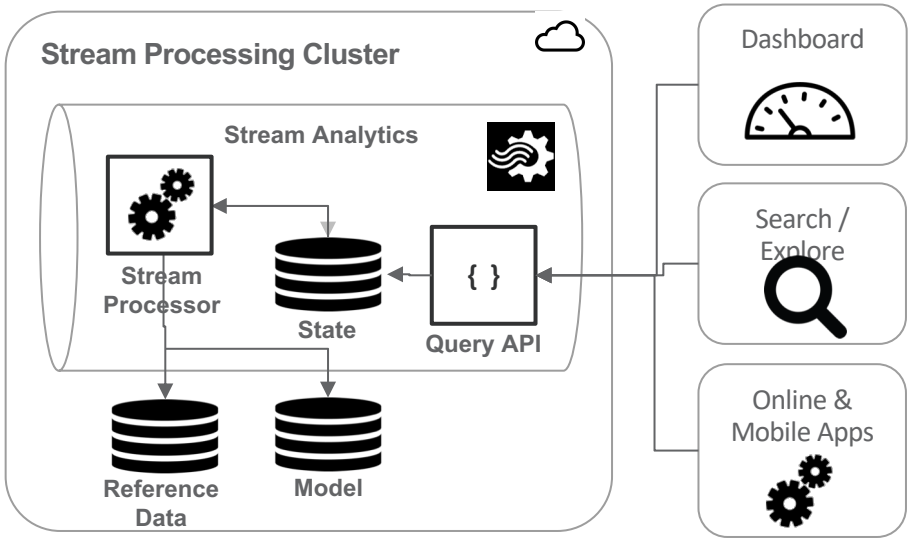
How does it handle failures? If a machine crashes and the/some state is lost?

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 Pattern Detection

- Streaming Data often contains interesting patterns that only emerge as new streaming data arrives, e.g.
  - Absence Pattern: event A not followed by event B within time window
  - Sequence Pattern: event A followed by event B followed by event C
  - Increasing Pattern: up trend of a value of a certain attribute
  - Decreasing Pattern: down trend of a value of a certain attribute
  - …
- Pattern operators allow developers to define complex relationships between streaming events

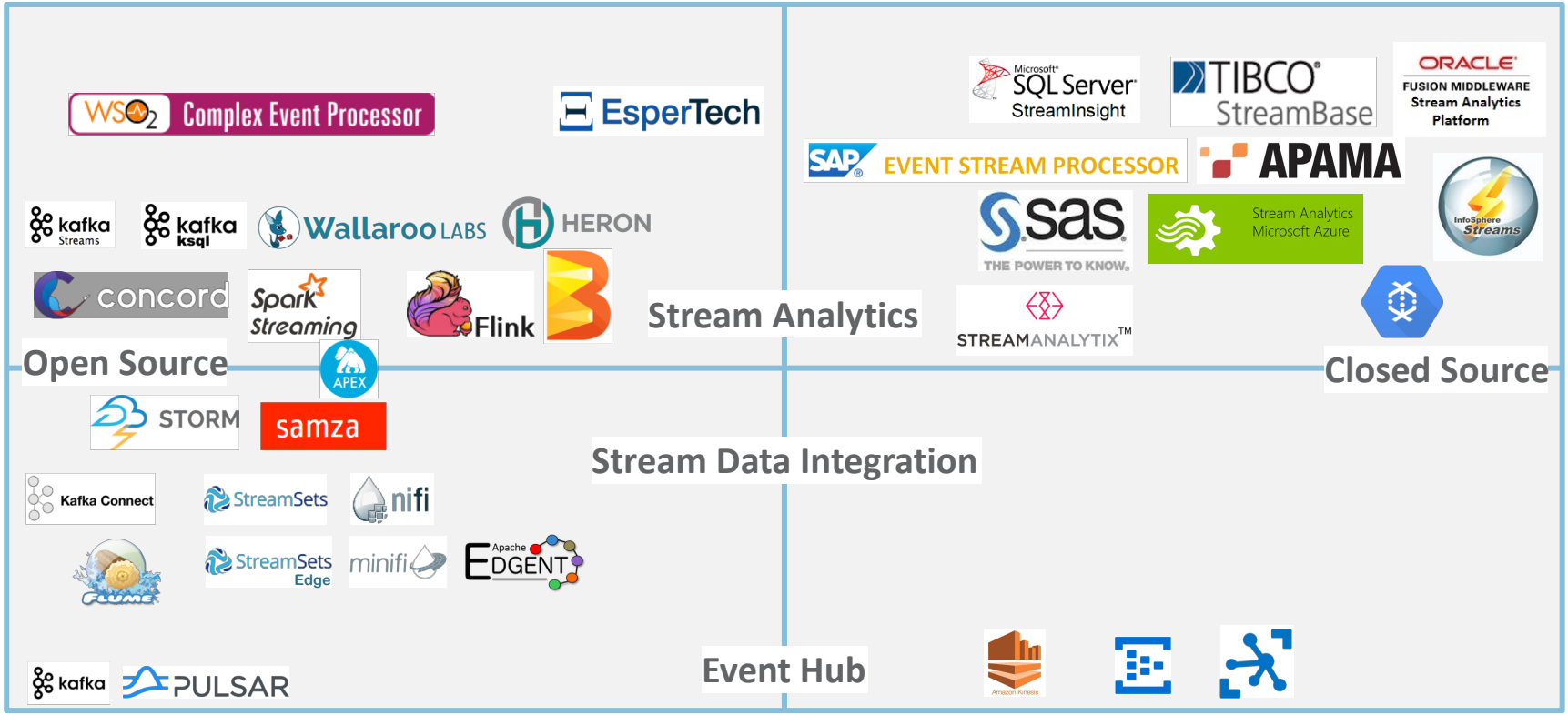
Capabilities: Stream Data Integration vs. Stream Analytics

	Stream Data Integration	Stream Analytics
Support for Various Data Sources	yes	-
Streaming ETL (Transformation/Format Translation, Routing, Validation)	yes	partial
Micro-Batching	yes	partial
Event-at-a-time	yes	yes
Delivery Guarantees	yes	yes
API : GUI-Based API / Declarative API / Programmatic	yes	yes
API: Streaming SQL	-	yes
Event Time vs. Ingestion / Processing Time	-	yes
Windowing	-	yes
Stream-to-Static Joins (Lookup/Enrichment)	partial	yes
Stream-to-Stream Joins	-	yes
State Management	-	yes
Queryable State (aka Interactive Queries)	-	yes
Event Pattern Detection	-	Yes

Implementing Stream Processing Solutions

@gschmutz

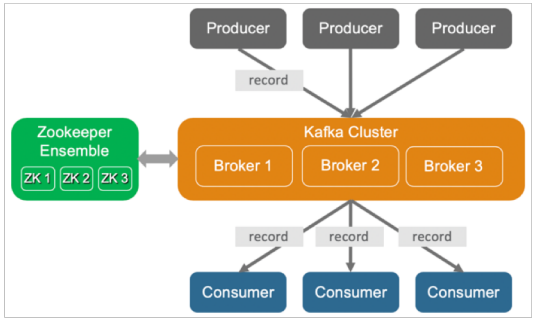
Stream Processing & Analytics Ecosystem



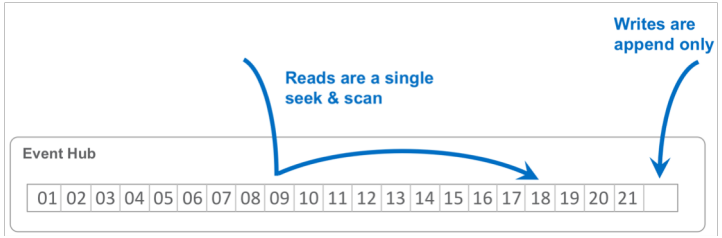
Source: adapted from Tibco

Event Hub: Apache Kafka

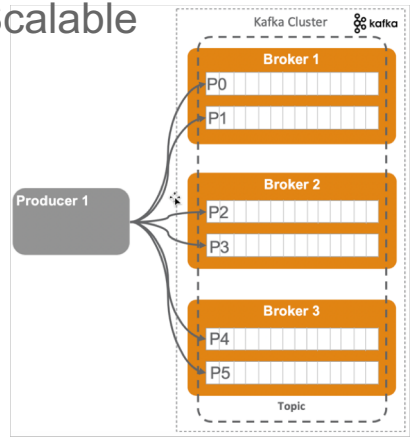
Highly available, Pub/Sub infrastructure



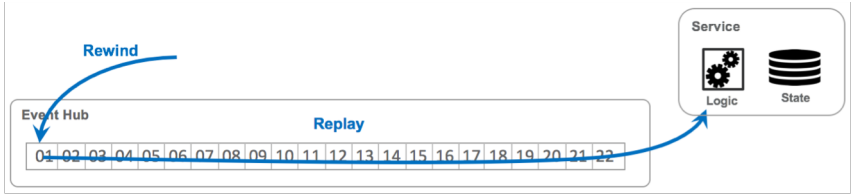
Distributed Log at the Core



Highly Scalable



Logs do not (necessarily) forget

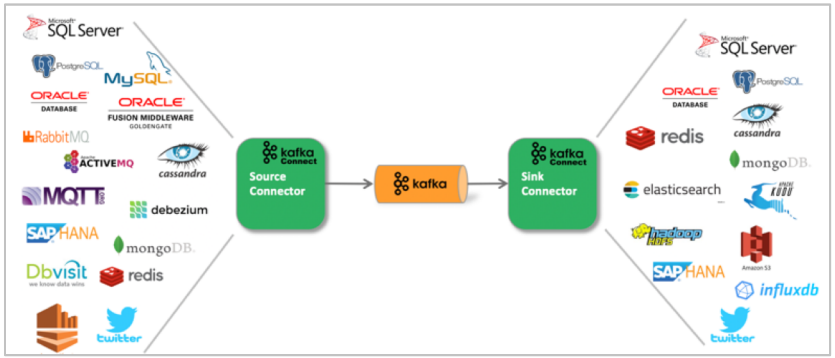


- Never
- Time (TTL) or Size-based
- Log-Compacted based

Stream Data Integration: Kafka Connect

declarative style data flows
simplicity - “simple things done simple”
very well integrated with Kafka – framework is part of Kafka
Single Message Transforms (SMT)

Many connectors available
Implement custom connectors using Java
Supported by Confluent




```
#!/bin/bash
curl -X "POST"
     "http://192.168.69.138:8083/connectors" \
     -H "Content-Type: application/json" \
     -d $'{
"name": "mqtt-source",
"config": {
  "connector.class": ”...MqttSourceConnector",
  "tasks.max": "1",
  "name": "mqtt-source",
  "mqtt.server.uri": "tcp://mosquitto:1883",
  "mqtt.topics": "truck/+/position",
  "kafka.topic":"truck_position",
  }
}'
```

Stream Data Integration: StreamSets

StreamSets

Continuous open source, intent-driven, big data ingest

Visible, record-oriented approach fixes combinatorial explosion

Both stream and batch processing

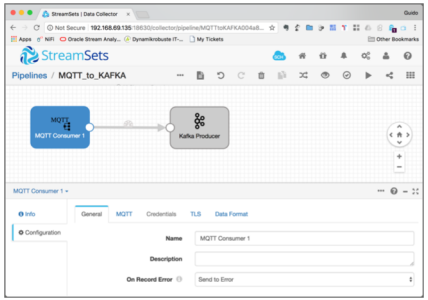
- Standalone, Spark cluster, MapReduce cluster

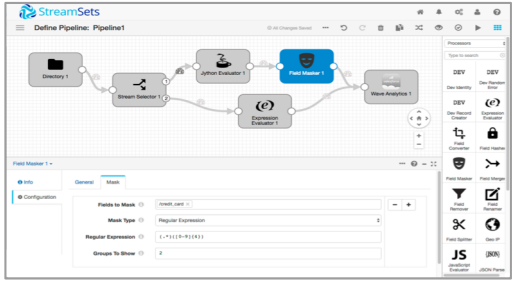
IDE for pipeline development by ‘civilians’

special option for Edge computing

custom sources, sinks, processors

Supported by StreamSets





Streaming Analytics: Kafka Streams

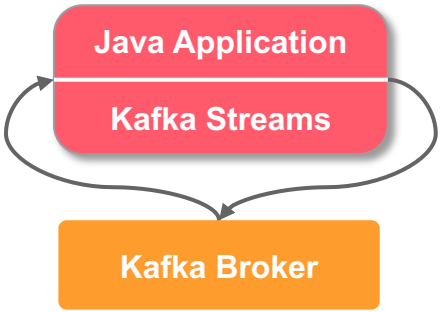
kafka Streams

Designed as a simple and lightweight library in Apache Kafka

no other dependencies than Kafka

Supports fault-tolerant local state

Supports Windowing (Fixed, Sliding and Session) and Stream-Stream / Stream-Table Joins



Millisecond processing latency, no micro-batching

At-least-once and exactly-once processing guarantees

```
KTable<Integer, Customer> customers =
   builder.stream("customer");

KStream<Integer, Order> orders =
   builder.stream("order");

KStream<Integer, String> enriched =
   orders.leftJoin(customers, …);

joined.to("orderEnriched");
```

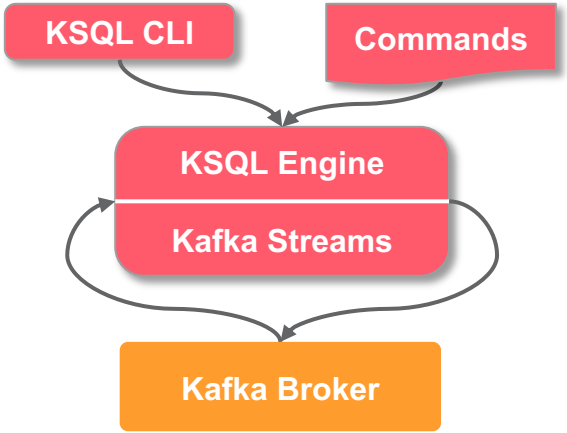
Streaming Analytics: KSQL

kafka ksql

Stream Processing with zero coding using SQL-like language

Built on top of Kafka Streams

Interactive (CLI) and headless (command file)



STREAM and TABLE as first-class citizens

- STREAM = data in motion
- TABLE = collected state of a stream

```
ksql> CREATE STREAM order_s \
      WITH (kafka_topic='order', \
        value_format='AVRO');

 Message
----------------
 Stream created

ksql> SELECT * FROM order_s \
WHERE address->country = 'Switzerland';
...
```

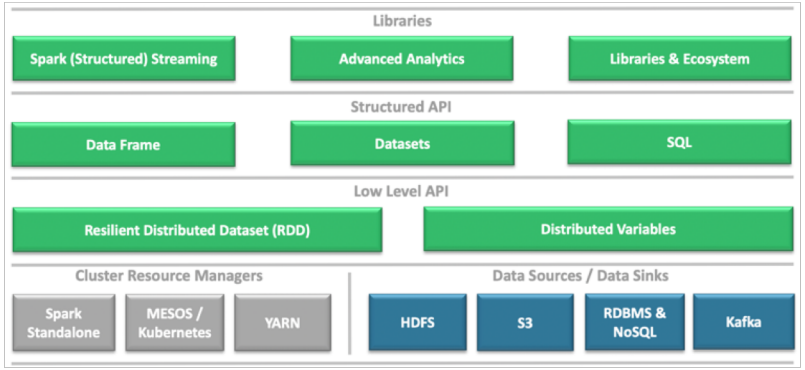
Streaming Analytics: Spark Structured Streaming

Spark Structured Streaming

2nd generation (1st to be Spark Streaming)

Structured API through DataFrames / Datasets rather than RDDs

Easier code reuse between batch and streaming



marked production ready in Spark 2.2.0

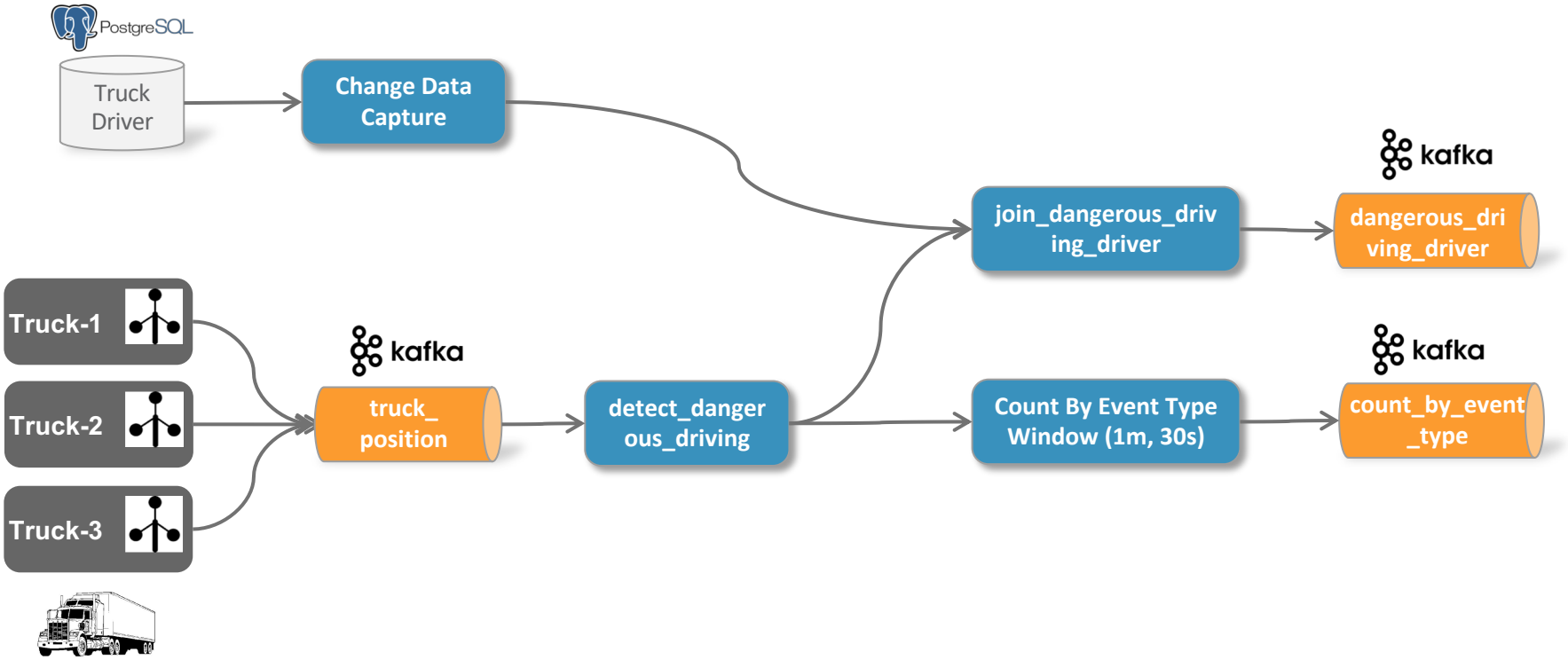
Support for Java, Scala, Python, R and SQL

```
val oderDf = spark
  .readStream
  .format("kafka")
  .option("kafka.bootstrap.servers",
"broker-1:9092")
  .option("subscribe", "order")
  .load()

val orderFilteredDf =
   orderDf.where(
        "address.county = 'Switzerland'")
```

Demo

Sample Use Case



Summary

Summary

Stream Processing is the solution for low-latency

Event Hub, **Stream Data Integration** and **Stream Analytics** are the main building blocks in your architecture

Kafka is currently the de-facto standard for Event Hub

Various options exists for Stream Data Integration and Stream Analytics

SQL becomes a valid option for implementing Stream Analytics

Still room for improvements (SQL, Event Pattern Detection, Streaming Machine Learning)

Technology on its own won't help you.

You need to know how to use it properly.