Aalto University
School of Science

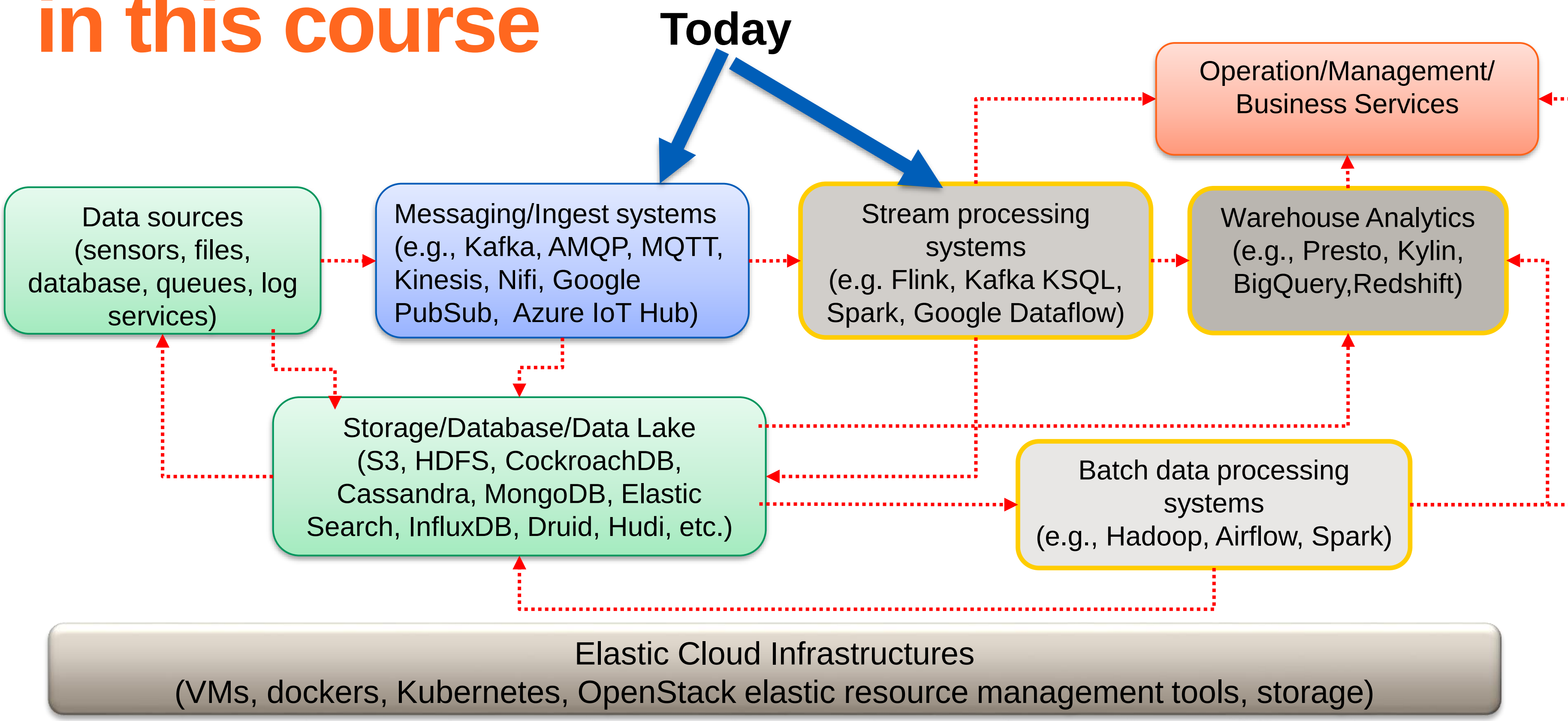
Stream Processing and Big Data Platforms

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi, https://rdsea.github.io

Learning objectives

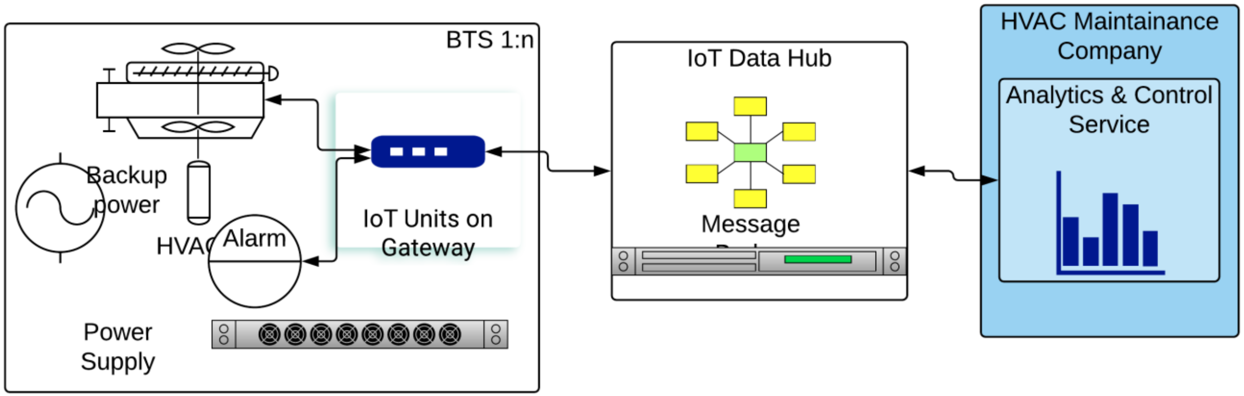
- Understand fundamental concepts and techniques in stream processing in big data
- Able to design stream processing analytics in big data platforms and applications
- Able to select and use common stream processing frameworks

Big data at large-scale: the big picture in this course



Motivating examples

Near real-time monitoring and anomaly detection for equipment and sites: what if you have 200K of Base Stations (BTS)



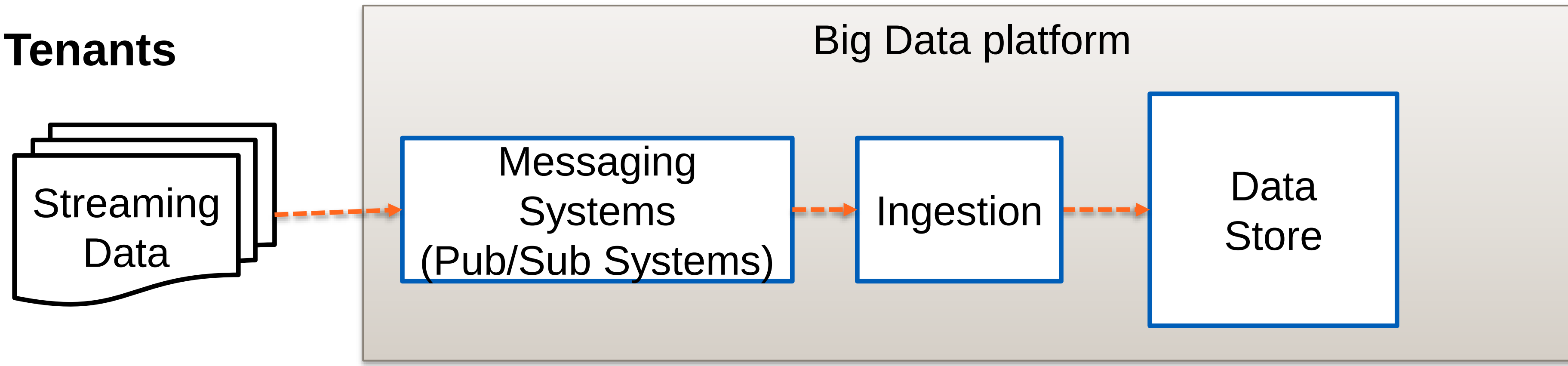
We need to analyze streaming data in a near-real time manner

Many other scenarios: fraud detection in online payment, stock market monitoring, traffic detection, etc.

Aalto University
School of Science

Basic stream analytics for data in motion

Recall: near-real time streaming data ingestion



- **Ingestion**
  - Mostly we ingest raw data without/little processing, e.g., IoT data
- **Data is unbounded from different places in different orders!**

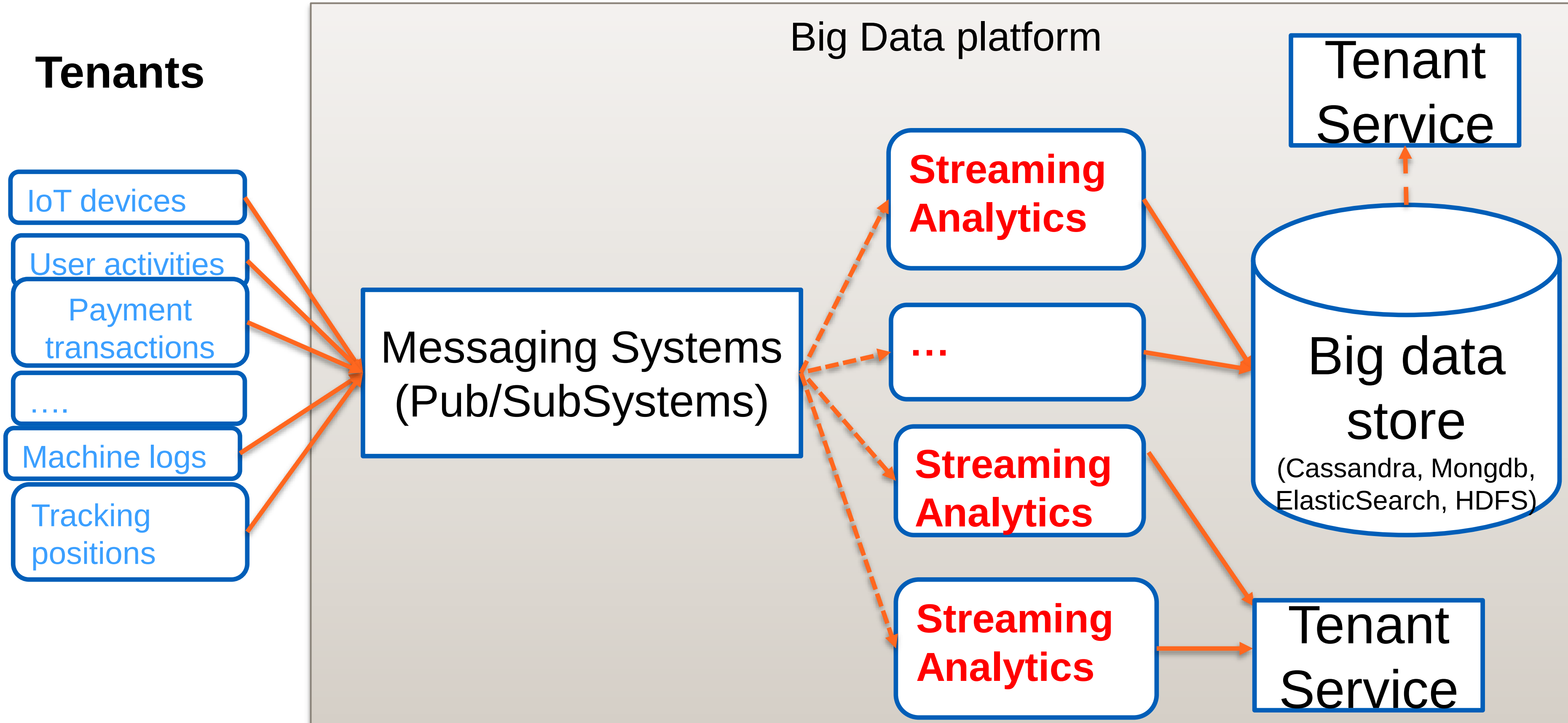
Stream processing and big data platforms

- **Stream processing is a component of big data platforms**
  - A big data technology for pre-processing, ingestion and high-level analytics
- **Stream processing services as big data platforms**
  - a big data platform offers mainly stream processing services for streaming analytics
  - Analytics on the fly as the first class
    - *Historical analytics results as the second class*
  - E.g., IoT analytics, e-commerce user activities, fraud detection

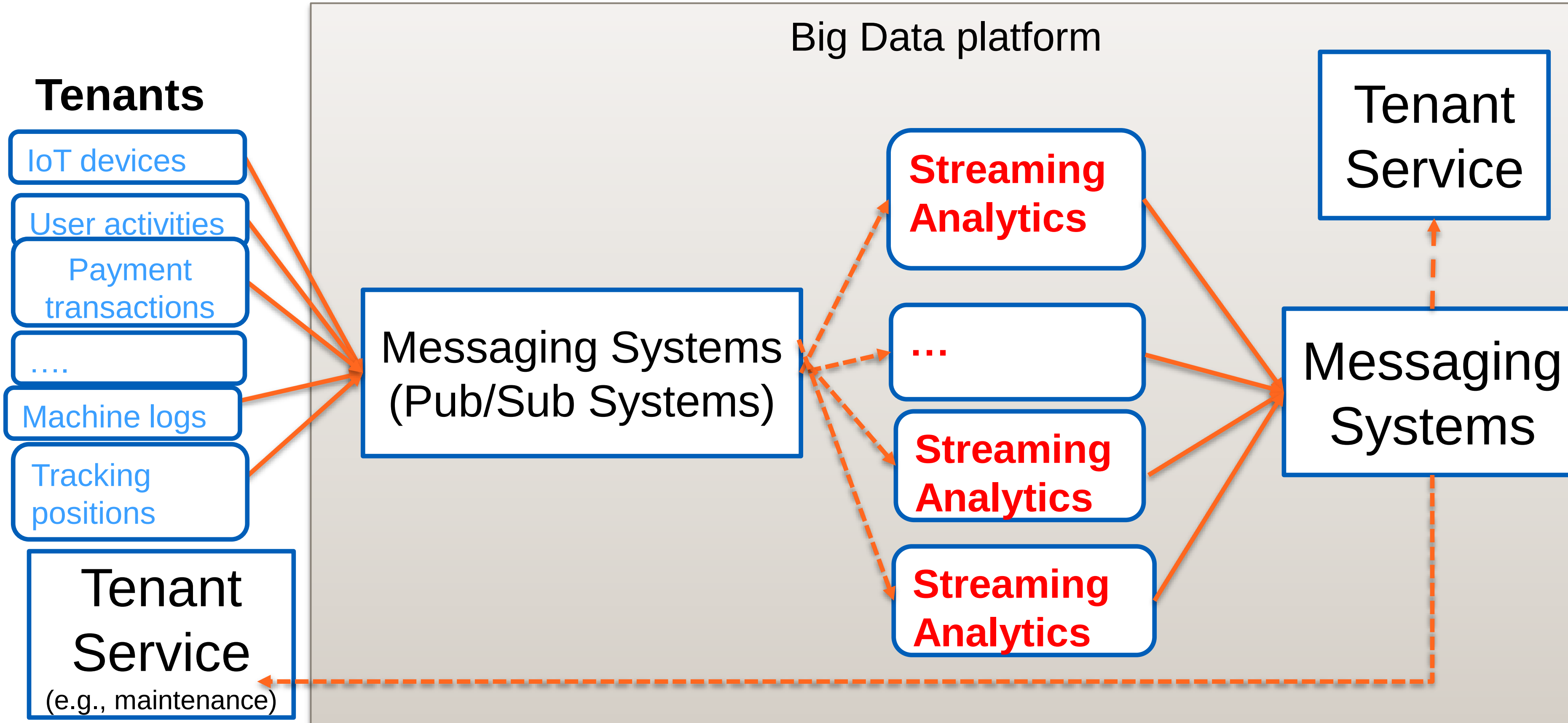
Stream processing in big data

- **Processing big data coming from streams at near real time**
  - The data element is "small" but voluminous and delivered in a near-real time manner
  - High and volatile throughput, low service latency
- **Require large-scale computing infrastructures and many other platform services**
  - Task parallelism: multiple tasks processing data
  - Data parallelism: data is partitioned into concurrent/parallel data streams → not like in data at rest

Near realtime streaming data processing



Near realtime streaming data processing



Example in the cloud – stream processing and big data platforms

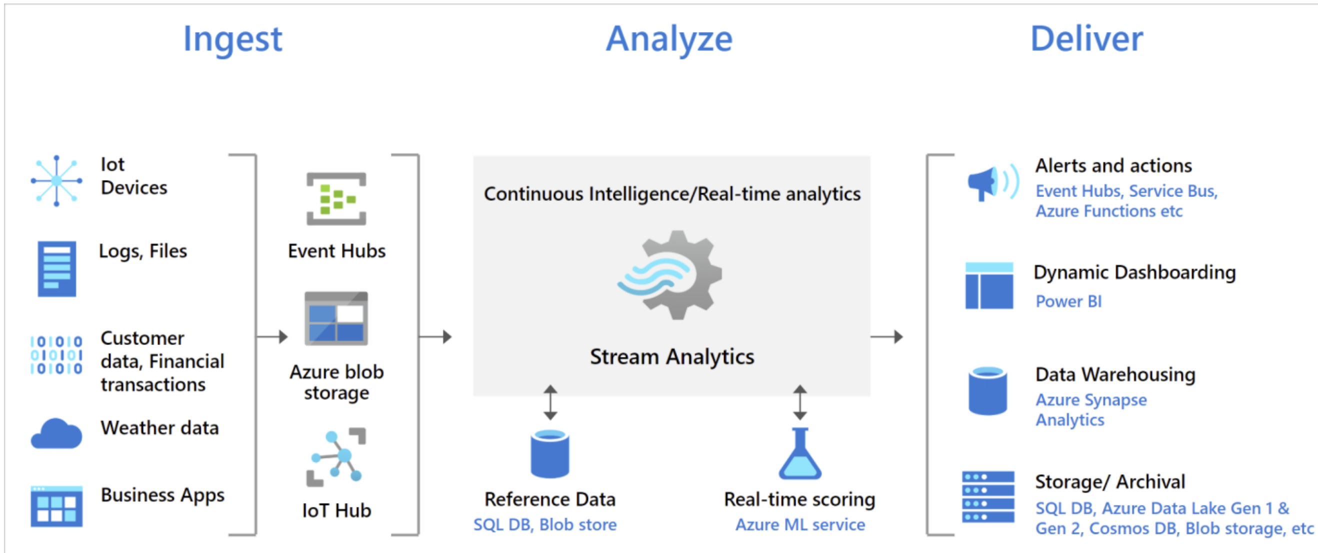


Figure source:
https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction

Long history, e.g., complex event processing (CEP) from enterprise computing

Siddhi
Cloud Native Stream Processor

Esper CEP

APACHE STORM™

Apache Apex™

samza

Our practices focus on Apache Flink, Apache Kafka and Apache Spark which are used intensively in business systems and big cloud platforms

Stream Processing – Key concepts

Common concepts

- **The way to connect data streams and obtain events/messages/records**
  - Focusing very much on *connector concepts* and well-defined event structures
  - The data can be pulled/pushed via connectors
- **The way to specify/programming the “analytics” logic**
  - Analytics functions, statements and how they are glued together to process flows of events
  - High-level, easy to use
- **The engine to process analytics tasks/operators**
  - Centralized in the view of the user → so the user does not have to program complex distributed applications
- **The way to push results to external components**

Data stream programming

Data stream: a sequence/flow of data units

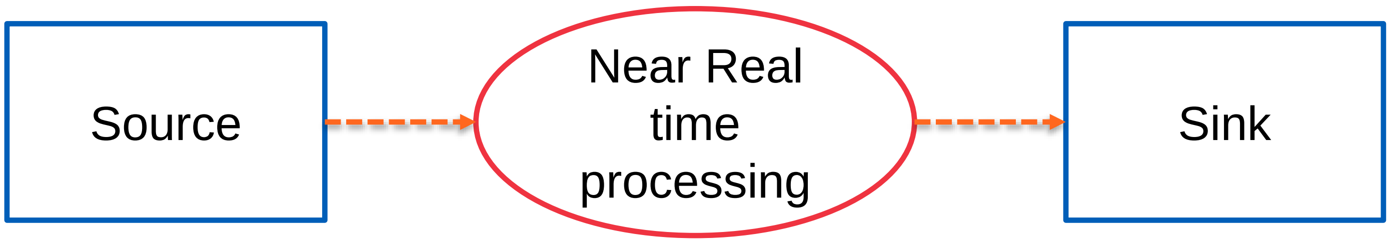
Data units are defined by applications: a data unit can be data described by a primitive data type or by a complex data type, a serializable object, etc.

Streaming data: produced by (near)realtime data sources as well as (big) static data sources → unbounded and bounded

- Examples of data streams
  - Continuous media (e.g., video for video analytics)
  - Discrete media (e.g., stock market events, twitter events, system monitoring events, comments, notifications)

Events/Records

In many applications: data is generated continuously and needs to be processed in near real-time



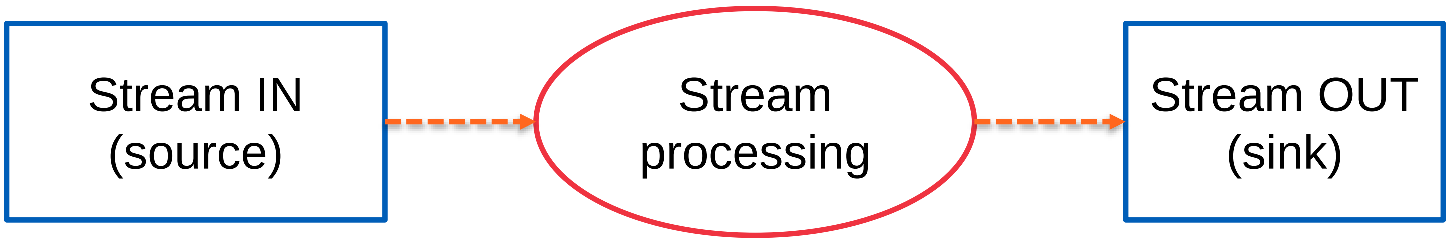
We focus on unbounded discrete events/records/messages

Event/record representation and streams

- **Data Sources:**
  - via message brokers, database, websocket, different IO adapters/connectors, etc.
- **Data Sink**
  - Message systems, databases, files, etc.
- **Data Representation & views**
  - POJO (Plain Old Java Object), CSV, JSON, Arvo format, etc.
  - SQL-alike tables

Stream processing

High level view

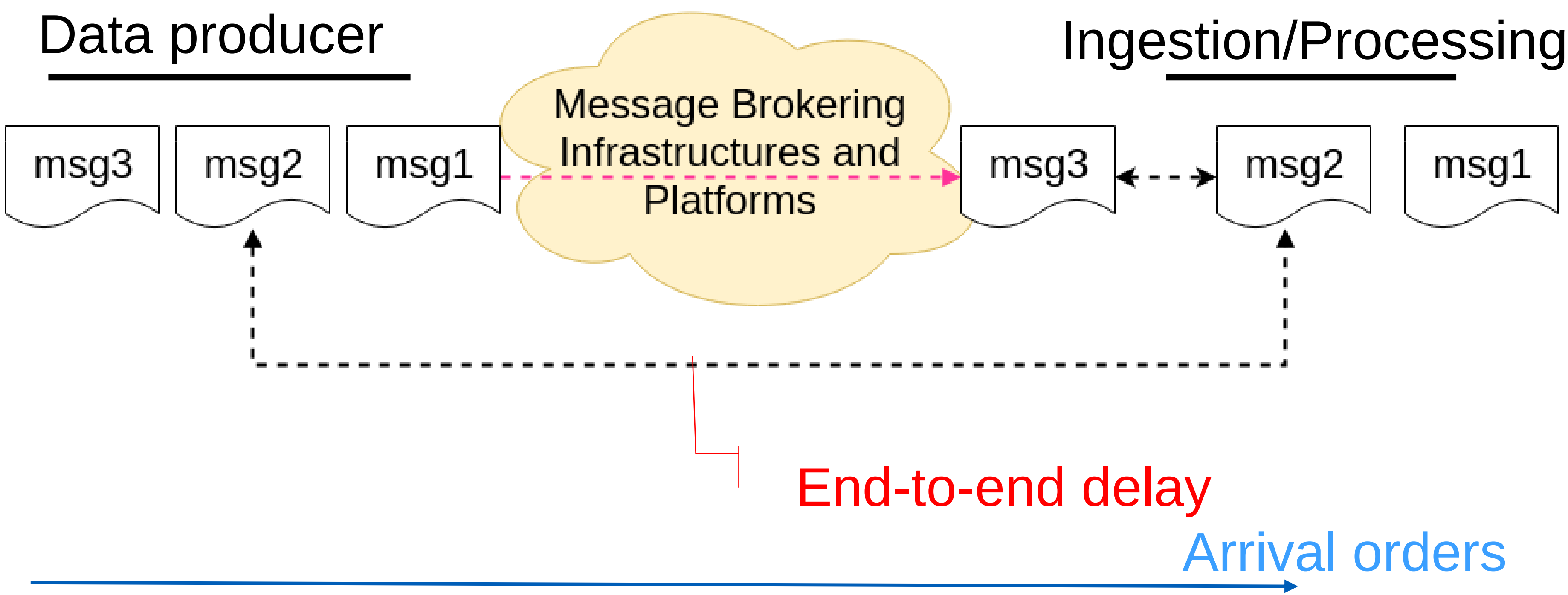


Multiple streams, a set of events

Some key issues

- **Data order & delivery**
  - Late data, out of order data
- **Times associated with events and processing**
- **Data parallelism**
  - Key-based data processing
- **Task parallelism**
  - Stateful vs stateless processing

Key issues in streaming data: delay and out of order

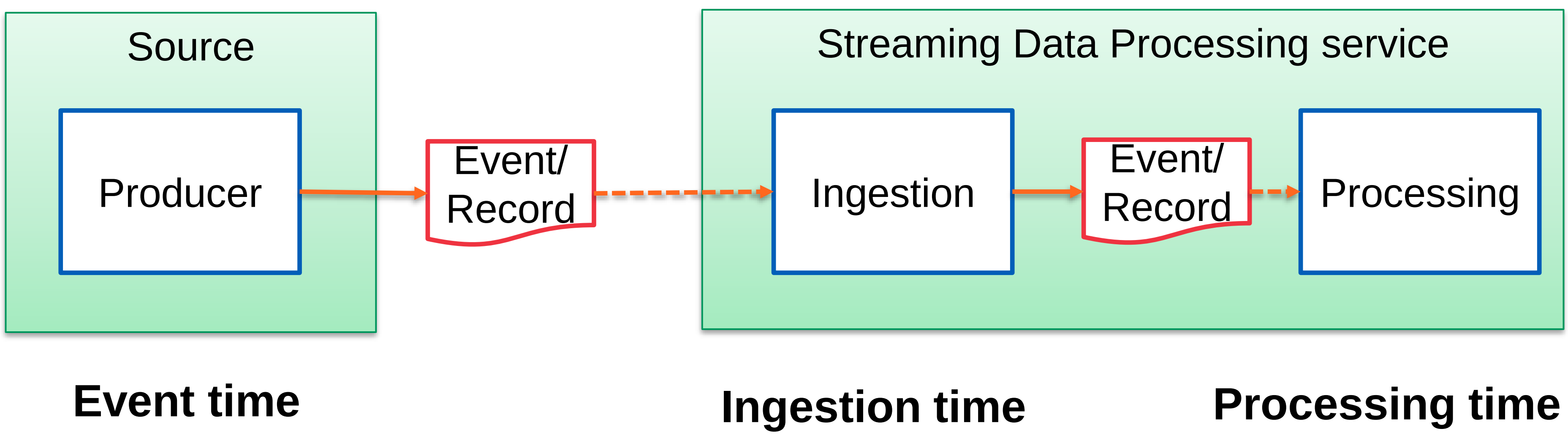


Without event/record time, do we know the delay or out of order?

What is the consequence of delay/out of order for processing?

Key issues in streaming data: the notion of times

Times associated with data and processing

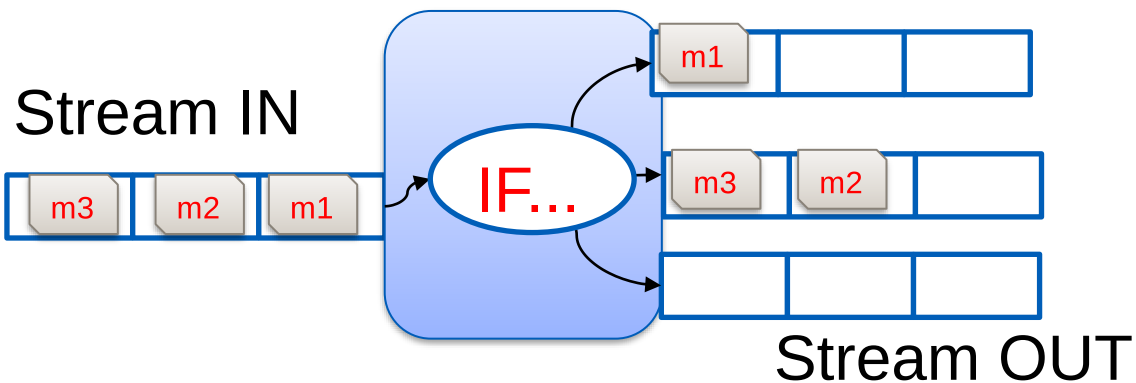


Which time is important for analytics (from business viewpoint)?

Data parallelism: partition stream data based on some keys for analytics

Split based on keys?

One-record vs batch of records



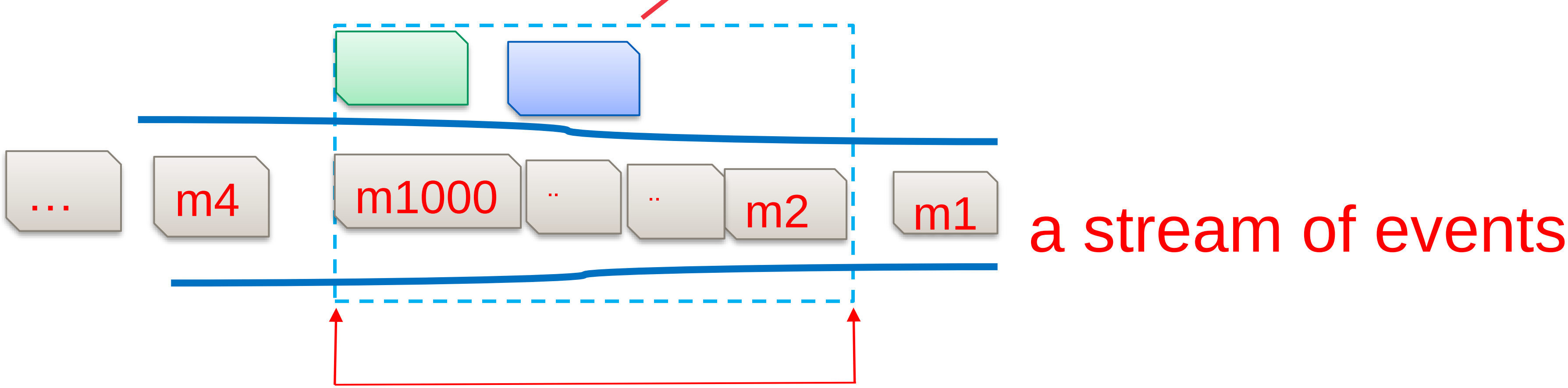
Can be processed in parallel using multiple nodes

With keyed data: enable parallel processing based on the keys

Windows of data

Window is used to group data for processing:

Which constraints are used to determine a window?



Sliding/Tumble window size: period of time or number of events/records

Arrival order

Windowing

- **Windows size: time or number of records (not popular)**
- **Tumbling window:**
  - identified by size, no gap between windows
- **Sliding window:**
  - identified by size and a sliding internal
- **Session Window:**
  - identified by "gap" between windows (e.g., the gap of events is used to mark "sessions")

Functions applied to Windows of data

If we

- specify a set of conditions for the window and events within the window

then we can

- Apply functions to events in the window that match these conditions

Task parallelism: we can have a lot of such functions executed in parallel in multiple compute nodes

Example

Monitoring working hours of (taxi/truck) drivers (assume events about pickup/drop captured at near realtime):

- **Windows: 12 hours**
- **Partitioning data/Keyed streams: licenseID**
- **Function: determine working and break times and check with the law/regulation**

Source: https://www.infoworld.com/article/3293426/how-to-build-stateful-streaming-applications-with-apache-flink.html

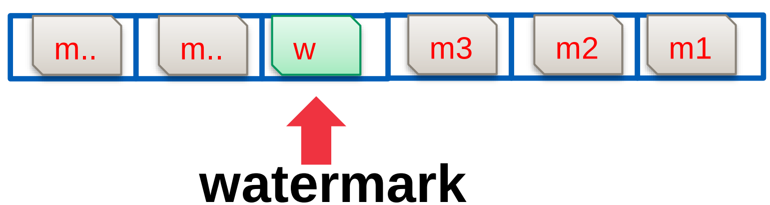
What if events/records come late into the windows?

Do we need to deal with late, out of order events/records?

correctness and completeness issues

Support lateness

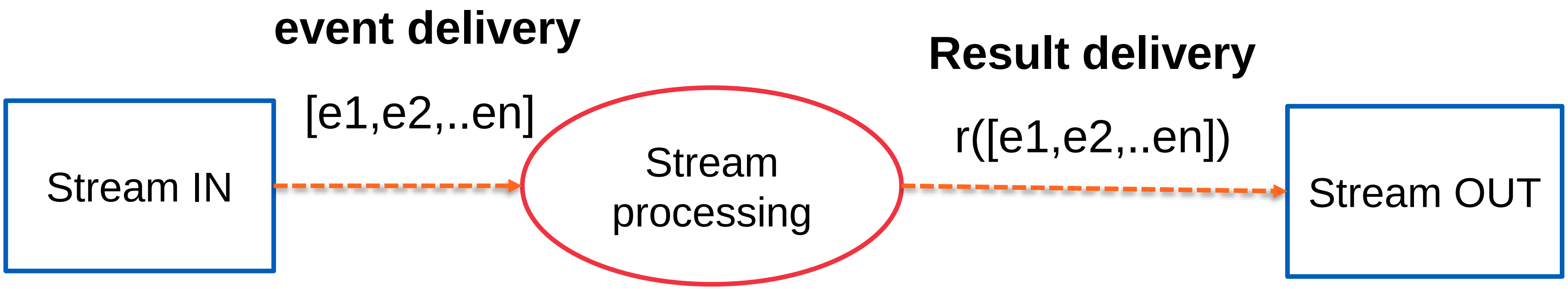
- **Identify timestamp of events/records**
- **Identify watermark in streams**
  - A watermark is a timestamp
  - A watermark indicates that no events which are older that the watermark should be processed
  - Enable the delay of processing functions to wait for late events
- **Using watermark to ignore late data → computing under "incompleteness assumption"**



Delivery guarantees

Exactly once? at least once? or at-most-once

End-to-end?



What if the stream processing fails and restarts

Message and processing guarantees

- **Message guarantees are the job of the broker/messaging system**
- **Processing guarantees are the job of the stream processing frameworks**
- **They are highly connected if messaging systems and processing frameworks are tightly coupled (e.g., Kafka case)**

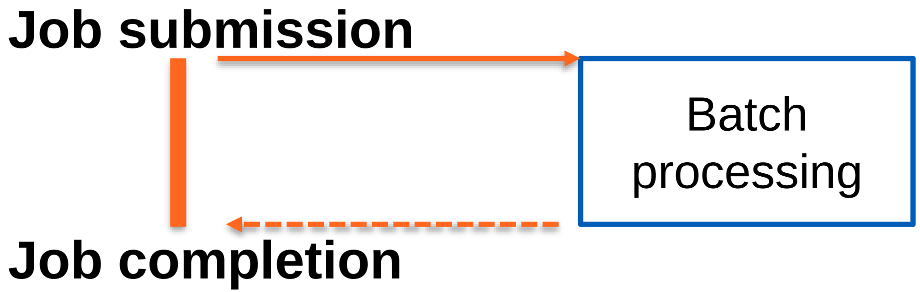
End-to-end exactly once

- **Exactly once for processing is not enough**
- **Messaging systems must support**
  - redeliver messages/data, recoverable data
- **Sink and output must support exactly one**
  - idempotent results, roll back
- **Coordination among various components**

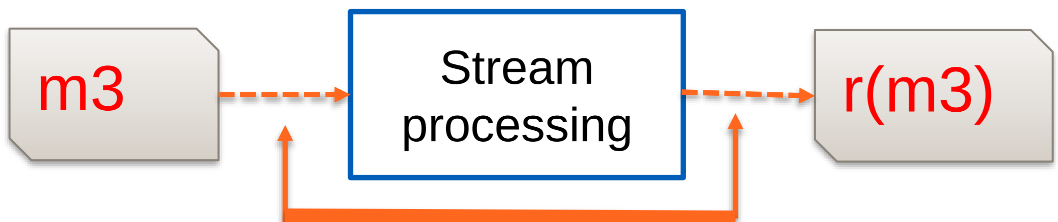
Further reading:
https://flink.apache.org/features/2018/03/01/end-to-end-exactly-once-apache-flink.html
https://www.confluent.io/blog/simplified-robust-exactly-one-semantics-in-kafka-2-5/
https://docs.microsoft.com/en-us/azure/hdinsight/spark/apache-spark-streaming-exactly-once

Performance metrics

Response time



(e.g., MapReduce/Hive, Spark SQL)



Service time/latency

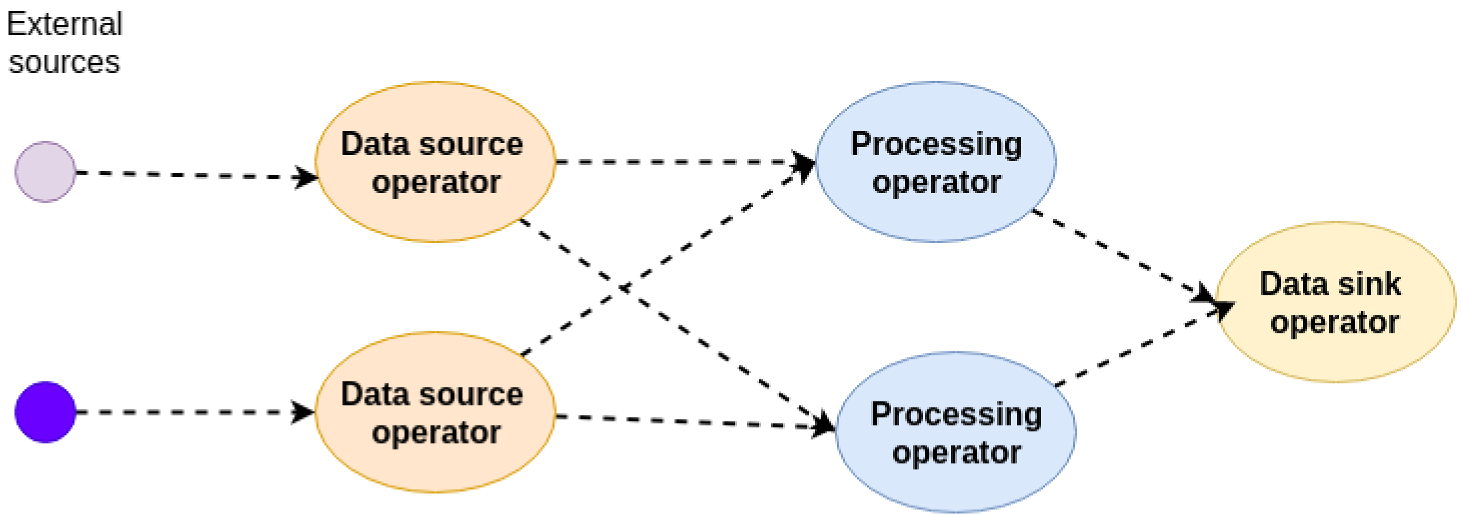
Latency and Throughput

- **Service latency**
  - Subseconds! E.g., milliseconds
  - Max, min or percentile? → up to application requirements
- **Throughput**
  - How many events can be processed per second?
- **Goal: low latency and high throughput!**

Structure of streaming data processing programs (1)

- **We have multiple streams of data, different functions for processing data, multiple computing nodes**
- **Data exchange between tasks**
  - Links in task graphs reflect data flows
- **Stream processing**
  - Centralized or distributed (in terms of computing resources)

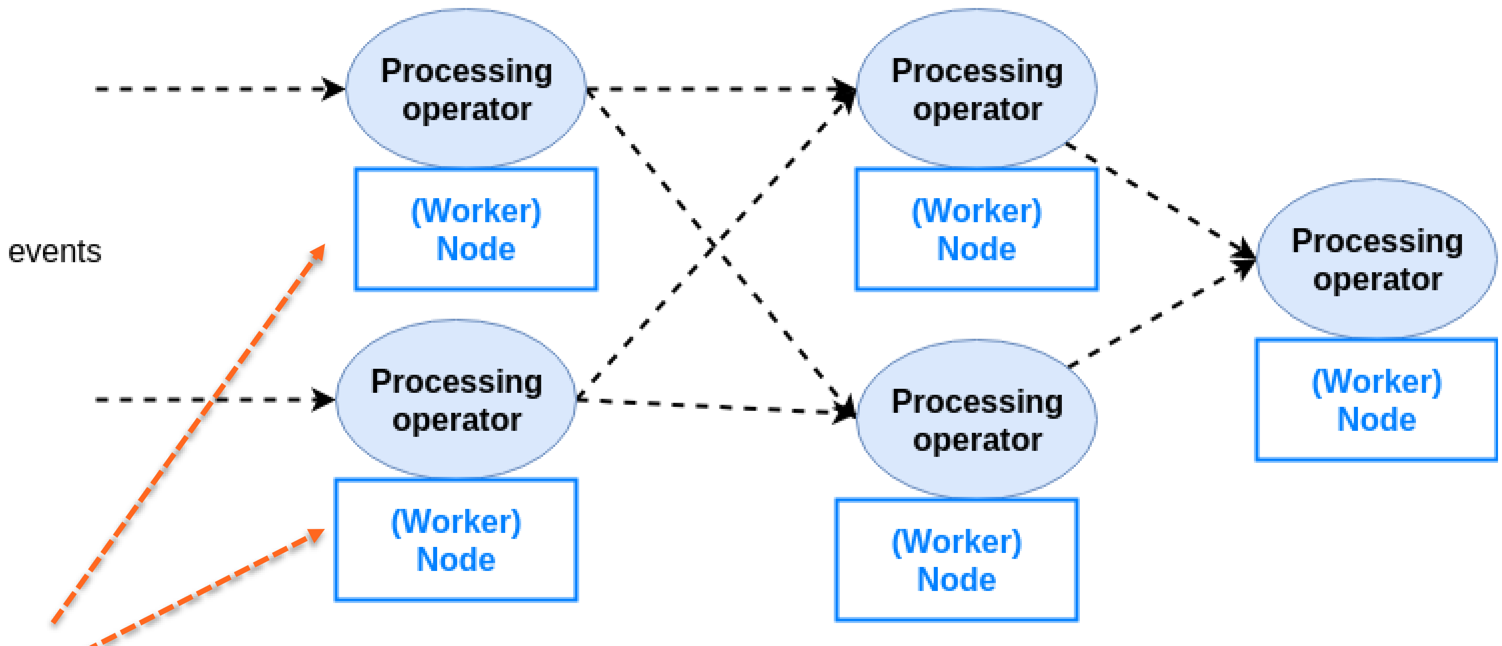
Structure of streaming data processing programs (2)



- **Data source operator: represents a source of streams**
- **Processing operators: represents processing functions**
- ***Native* versus *micro-batching***

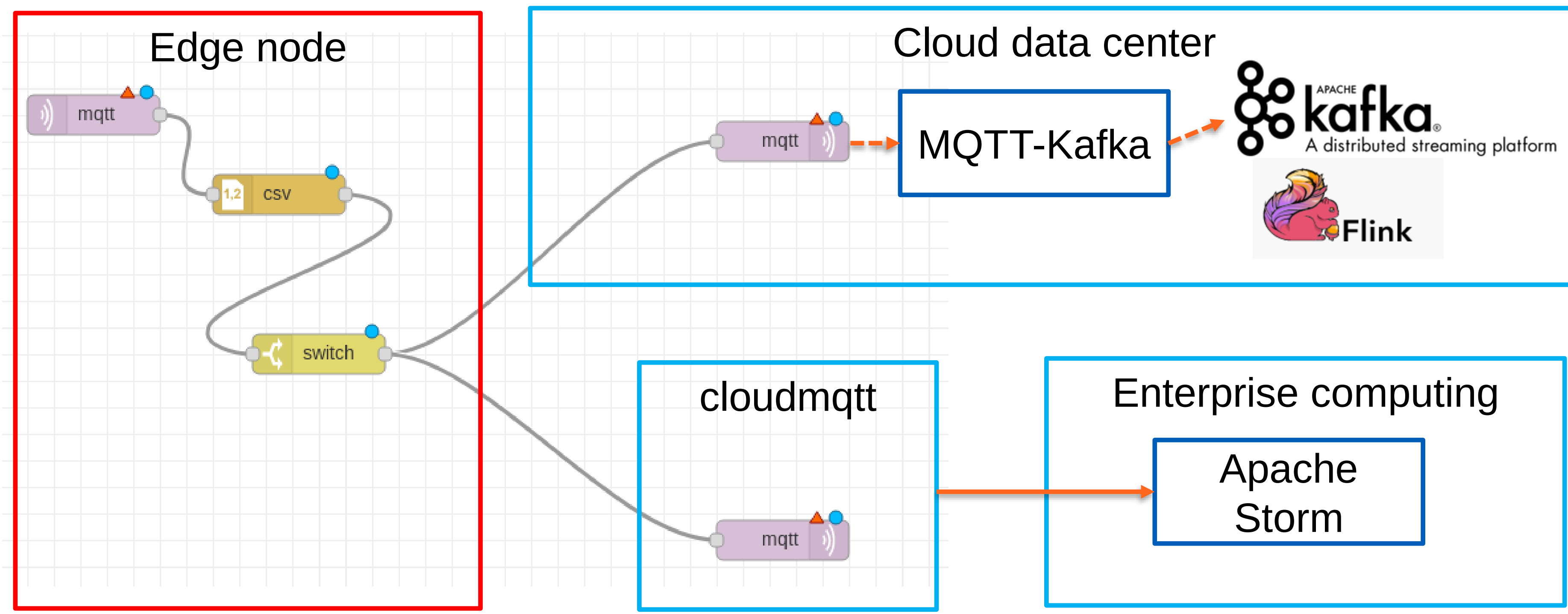
Distributed processing topology in a cluster

A graph of tasks (running operators); all tasks are running



Nodes of a cluster (VMs, containers, Kubernetes)

Distributed processing topology in cross distributed sites



Common concepts in existing frameworks - programming level

- **How to write streaming program?**
- **With programming languages**
  - Low level APIs
  - DSL
  - Java, Scala, Python (Spark, Flink, Kafka)
- **High-level data models**
  - KSQL
- **Flow/pipeline description**
  - Node-RED/GUI-based flow editors

Common concepts in existing frameworks - key common concepts

- **Abstraction of streams**
- **Connector library for data sources/sinks**
  - Very important for application domains
- **Runtime elasticity**
  - Add/remove (new) operators
  - Add/remove underlying computing nodes
- **Fault tolerance**

Why are the richness and the diversity of connectors important?

Where do you find most of concepts that we have discussed

- **Apache Storm**
  - *https://storm.apache.org/*
- **Apache Spark (Structured Streaming)**
  - *https://spark.apache.org/*
- **Apache Kafka Streams and KSQL**
  - strongly bounded to Kafka messaging
- **Apache Flink (Stream Analytics)**
  - native, clustered, better data sources/sinks
- **Apache Beam (https://beam.apache.org/)**
  - Unifying programming models for batch and stream processing

Practical learning paths

- **Path 1: if you don’t have a preference and need challenges**
  - Apache Flink Stream API (e.g., with RabbitMQ/Kafka connectors)
- **Path 2: many of you have worked with Kafka**
  - Kafka Streams DSL (everything can be done with Kafka)
- **Path 3: for those of you who are working with Spark (and Python is the main programming language)**
  - Apache Spark Structured Streaming
- **Path 4: for those who deal with MQTT brokers**
  - Apache Storm (but also Kafka, …): Spout and Bolt API or Stream API

Examples of Apache Flink

Apache Flink

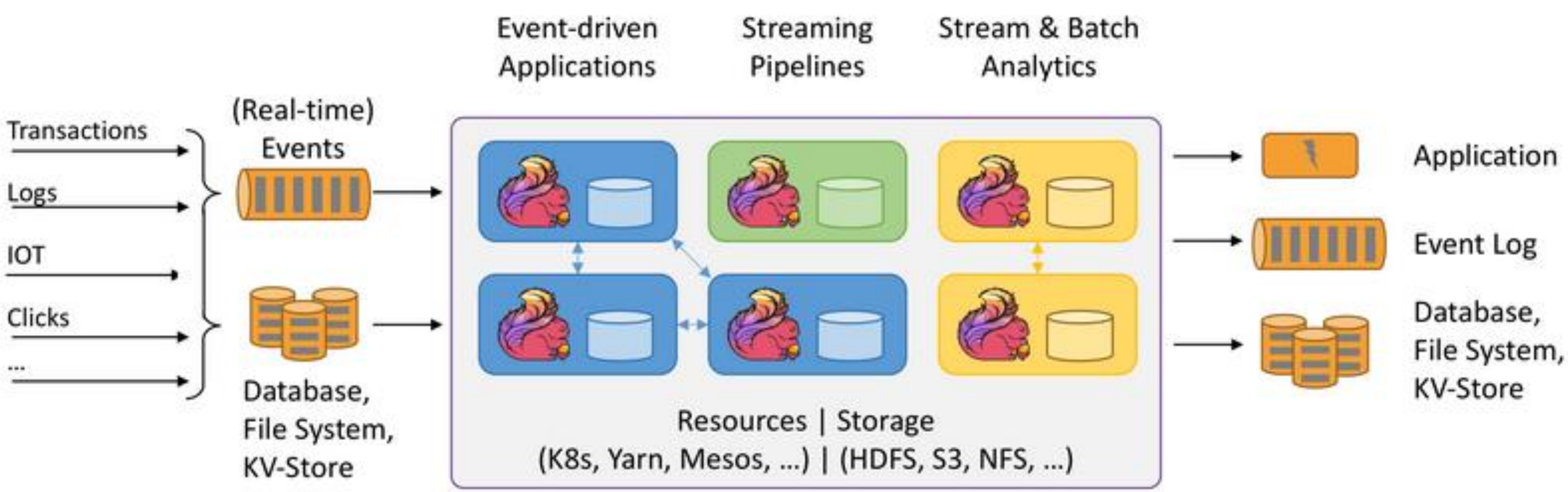
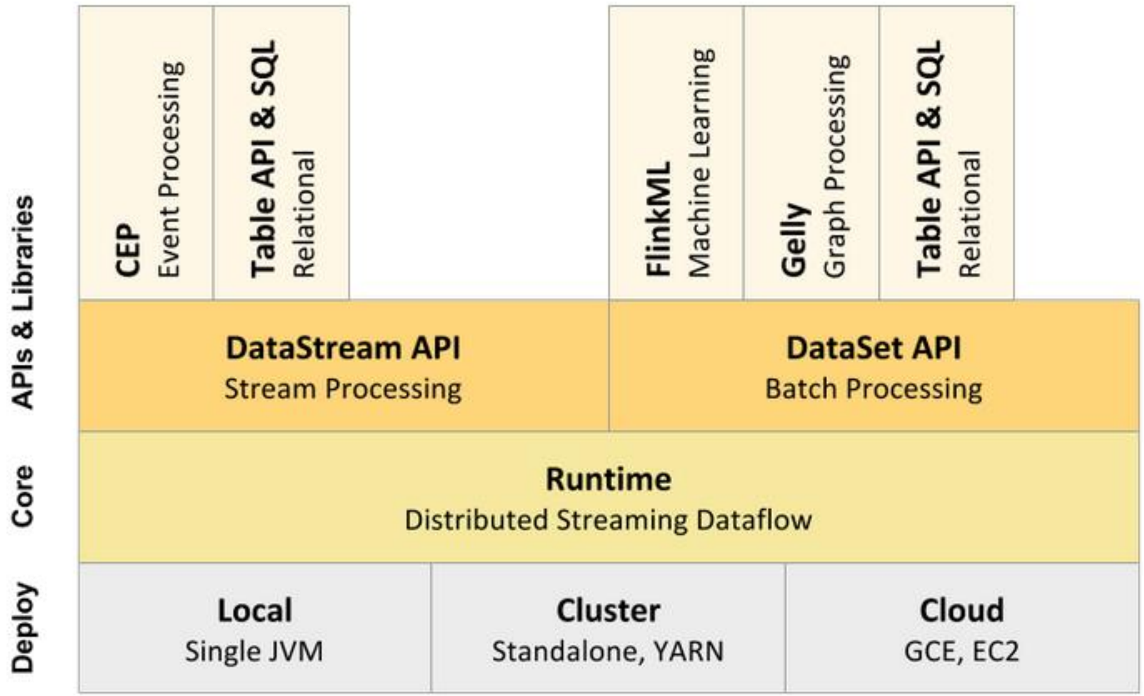


Figure source: https://flink.apache.org/

Apache Flink



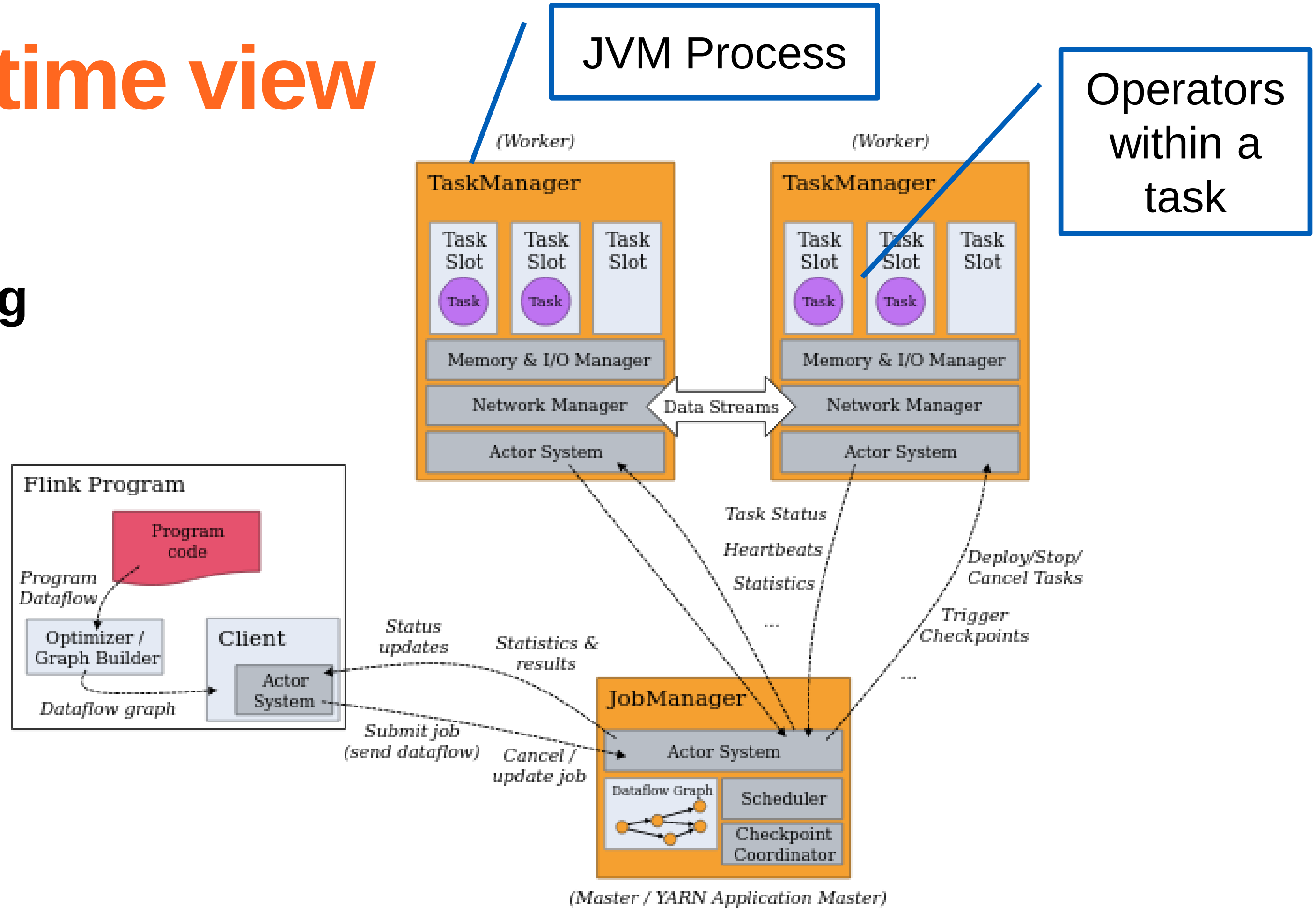
We focus only on DataStream API for understanding studied concepts

Source: https://ci.apache.org/projects/flink/flink-docs-release-1.9/internals/components.html

Flink runtime view

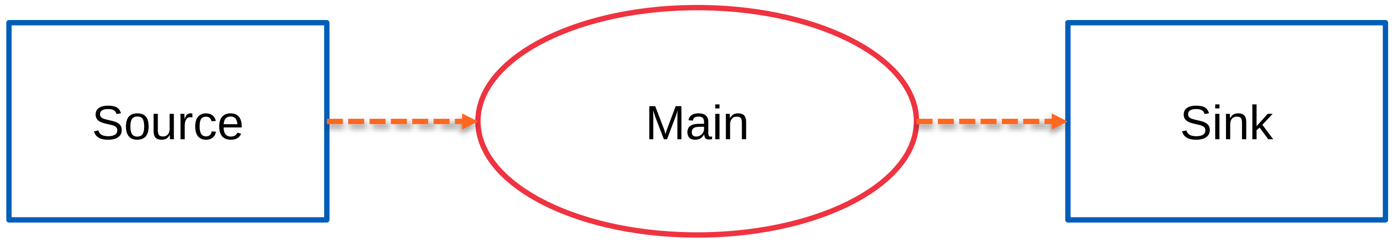
- Parallelism
- Checkpointing
- Monitoring

Remember 24/7



Sourcehttps://nightlies.apache.org/flink/flink-docs-release-1.14/docs/concepts/flink-architecture/

Main elements in Flink applications



- **Rich set of sources and sinks via many connectors**

Connectors

- **Major systems in big data**
- **We have used many of them in our study**
  - Apache Kafka
  - Apache Cassandra
  - Elasticsearch (sink)
  - Hadoop FileSystem
  - RabbitMQ
  - Apache NiFi
  - Google PubSub

Main

- **Setting environments**
- **Handling inputs and outputs via data streams**
- **Key functions for processing data**
- **Stream processing flows**

Bounded and unbounded streams

Stream processing flows

Split streaming data into different windows with a key for analytics purposes

Keyed Windows

```
stream
       .keyBy(...)               <-  keyed versus non-keyed windows
       .window(...)              <-  required: "assigner"
      [.trigger(...)]            <-  optional: "trigger" (else default trigger)
      [.evictor(...)]            <-  optional: "evictor" (else no evictor)
      [.allowedLateness(...)]    <-  optional: "lateness" (else zero)
      [.sideOutputLateData(...)] <-  optional: "output tag" (else no side output for late data)
       .reduce/aggregate/fold/apply()      <-  required: "function"
      [.getSideOutput(...)]      <-  optional: "output tag"
```

Source: https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/stream/operators/windows.html

Stream processing flows

Handling streaming data without a key for analytics purposes

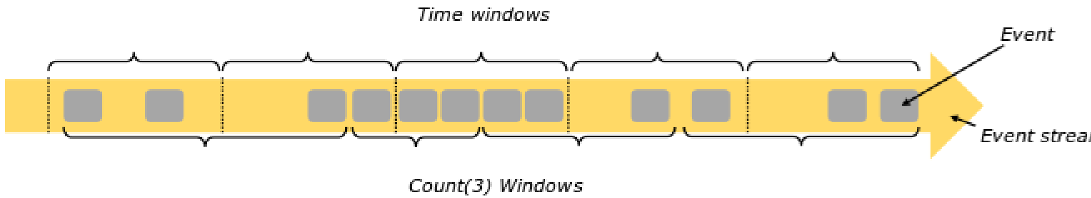
Non-Keyed Windows

```
stream
       .windowAll(...)           <-  required: "assigner"
      [.trigger(...)]            <-  optional: "trigger" (else default trigger)
      [.evictor(...)]            <-  optional: "evictor" (else no evictor)
      [.allowedLateness(...)]    <-  optional: "lateness" (else zero)
      [.sideOutputLateData(...)] <-  optional: "output tag" (else no side output for late data)
       .reduce/aggregate/fold/apply()      <-  required: "function"
      [.getSideOutput(...)]      <-  optional: "output tag"
```

Source: https://ci.apache.org/projects/flink/flink-docs-release-1.9/dev/stream/operators/windows.html

Windows and Times

Windows



Times

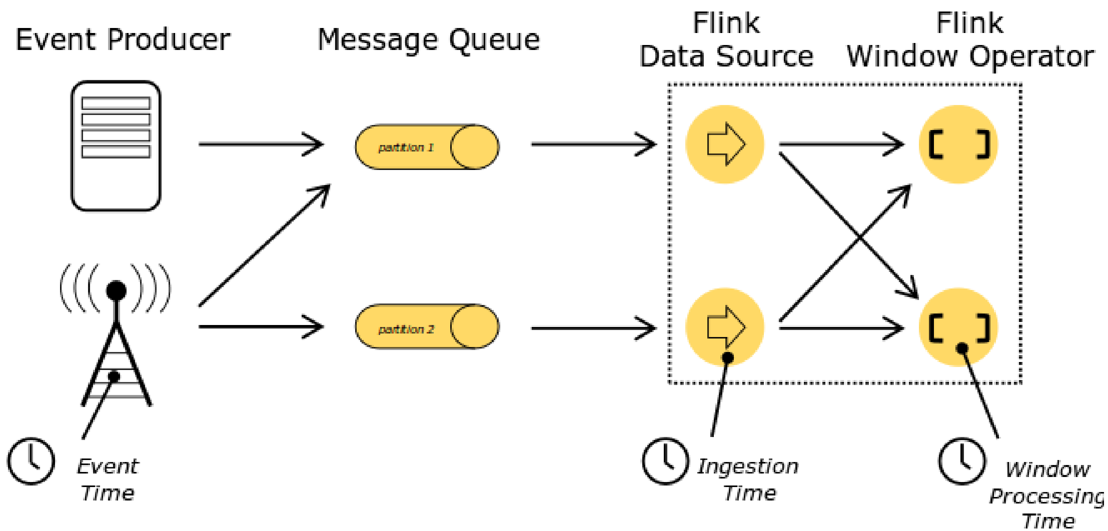
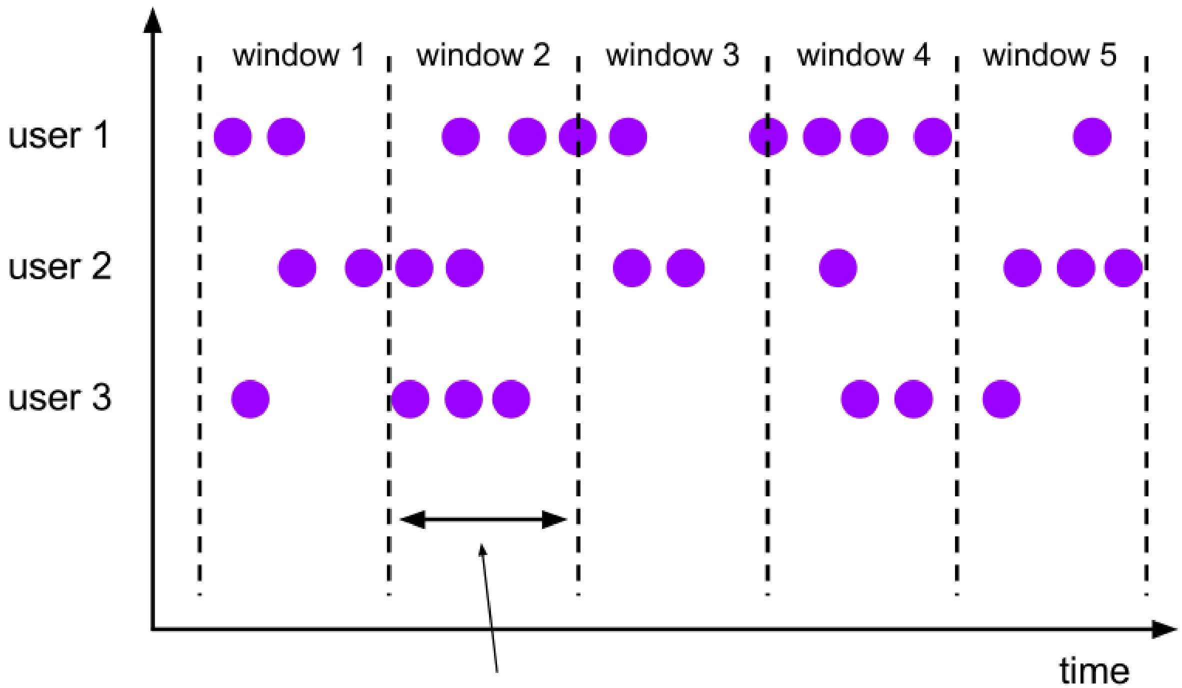


Figure source: https://nightlies.apache.org/flink/flink-docs-release-1.14/docs/concepts/time/

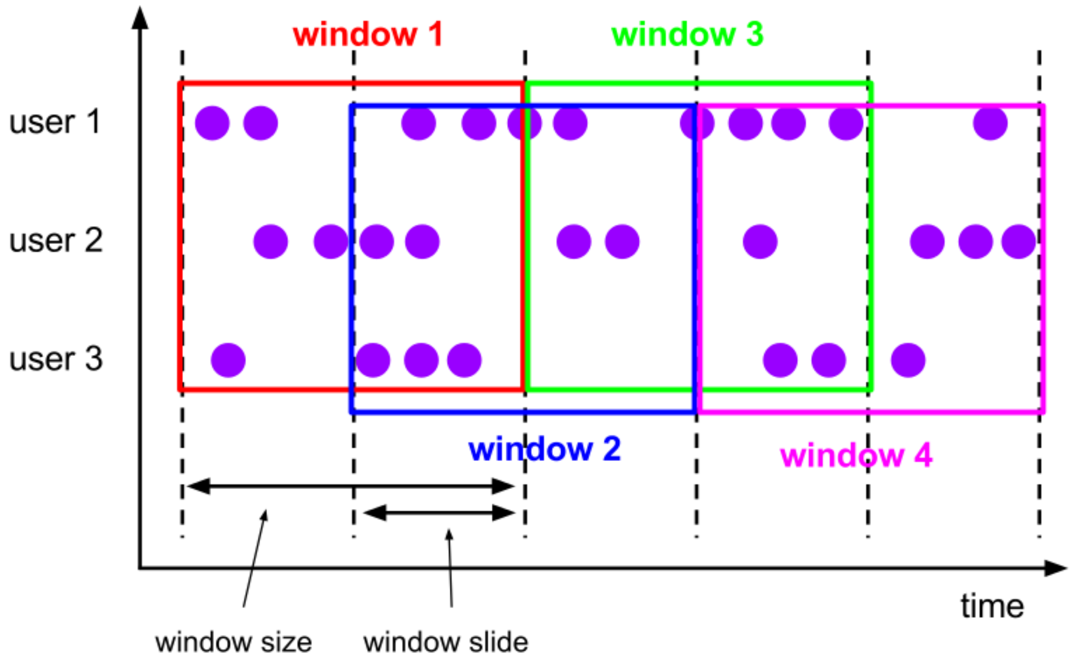
Batch/Tumbling Windows



Use cases: Period computation (e.g. stock, temperature, IoT data)

Figure source: https://nightlies.apache.org/flink/flink-docs-release-1.14/docs/dev/datastream/operators/windows/

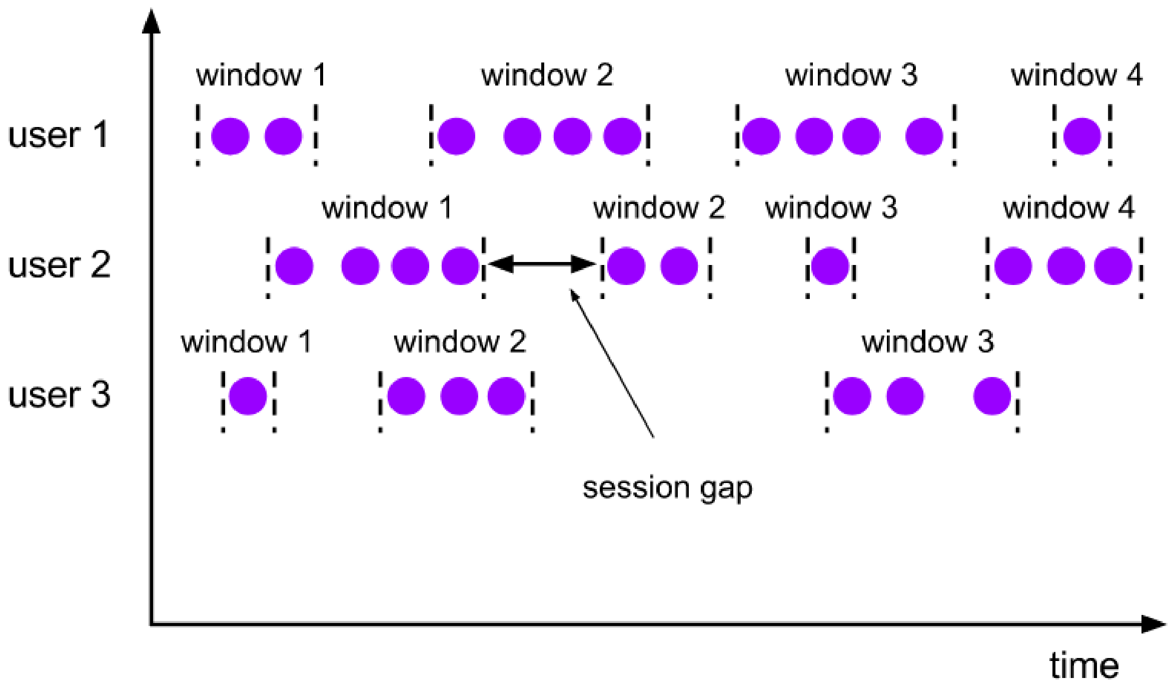
Sliding windows



Use cases:
Moving average

Figure source: https://nightlies.apache.org/flink/flink-docs-release-1.14/docs/dev/datastream/operators/windows/

Session Windows



Use cases: Web/user activities clicks

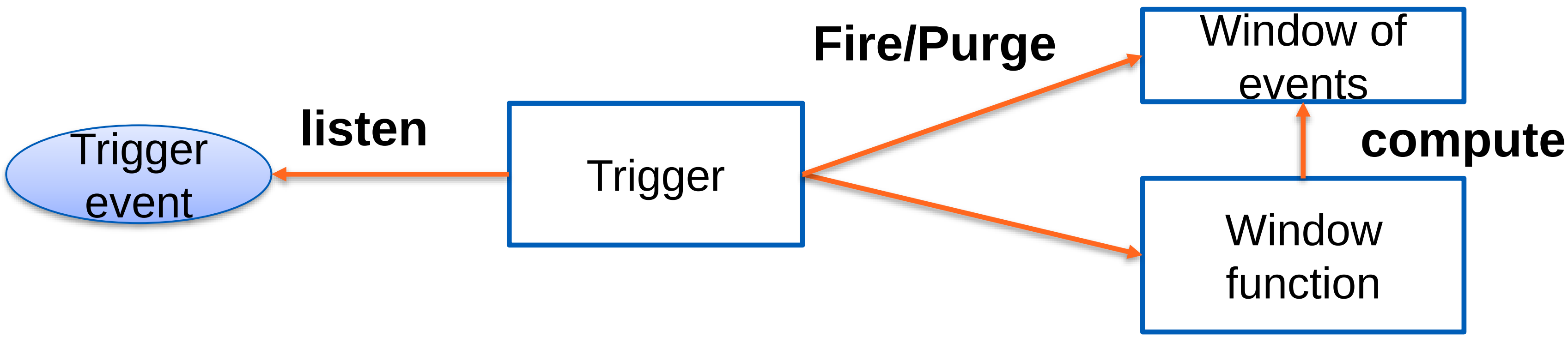
Figure source: https://nightlies.apache.org/flink/flink-docs-release-1.14/docs/dev/datastream/operators/windows/

Window Functions

- **Reduce Function**
  - Reduce two inputs
- **Aggregate Function**
  - Add an input into an accumulator
- **Fold Function**
  - Combine input with an output
- **ProcessWindow Function**
  - Get all elements of the windows and many other information so that you can do many tasks

Triggers & Evictor

- **Trigger: determine if a window is ready for window functions**



Evictor: actions after the trigger fires and before and/or after the windows function is called

Fault tolerance

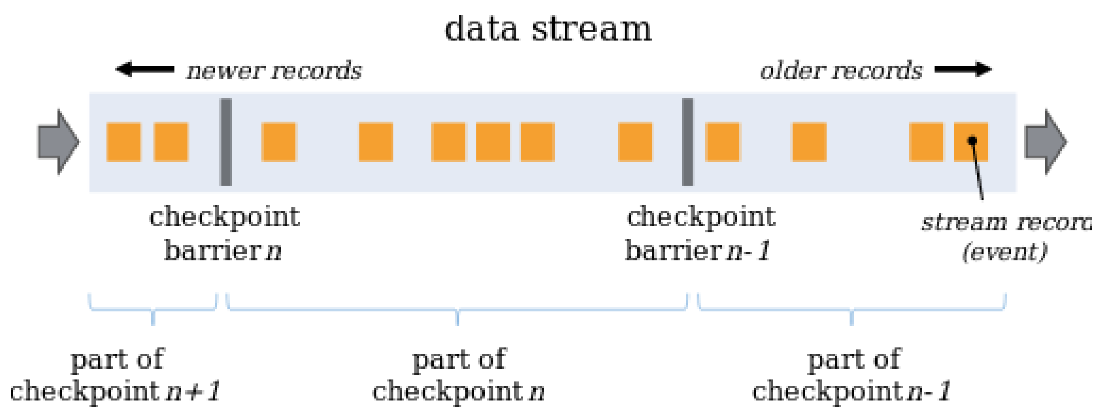


Figure source: https://nightlies.apache.org/flink/flink-docs-master/docs/learn-flink/fault_tolerance/

- **Principles: checkpointing, restarts operators from the latest successful checkpoints**
- **Need support from data stream sources/sinks w.r.t. (end-to-end) exactly once message receiving and result delivery**

Example with Base Transceiver Station

Data in our git

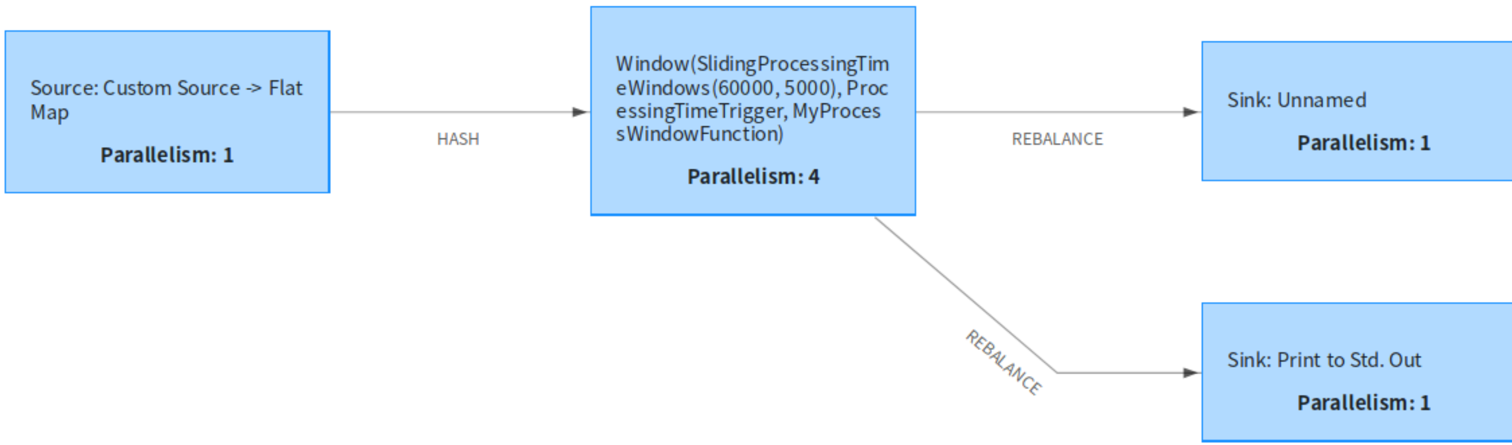
station_id,datapoint_id,alarm_id,event_time,value,valueThreshold,isActive,storedtime
1161115016,121,308,2017-02-18 18:28:05 UTC,240,240,false,
1161114050,143,312,2017-02-18 18:56:20 UTC,28.5,28,true,
1161115040,141,312,2017-02-18 18:22:03 UTC,56.5,56,true,
1161114008,121,308,2017-02-18 18:34:09 UTC,240,240,false,
1161115040,141,312,2017-02-18 18:20:49 UTC,56,56,false,

Simple example

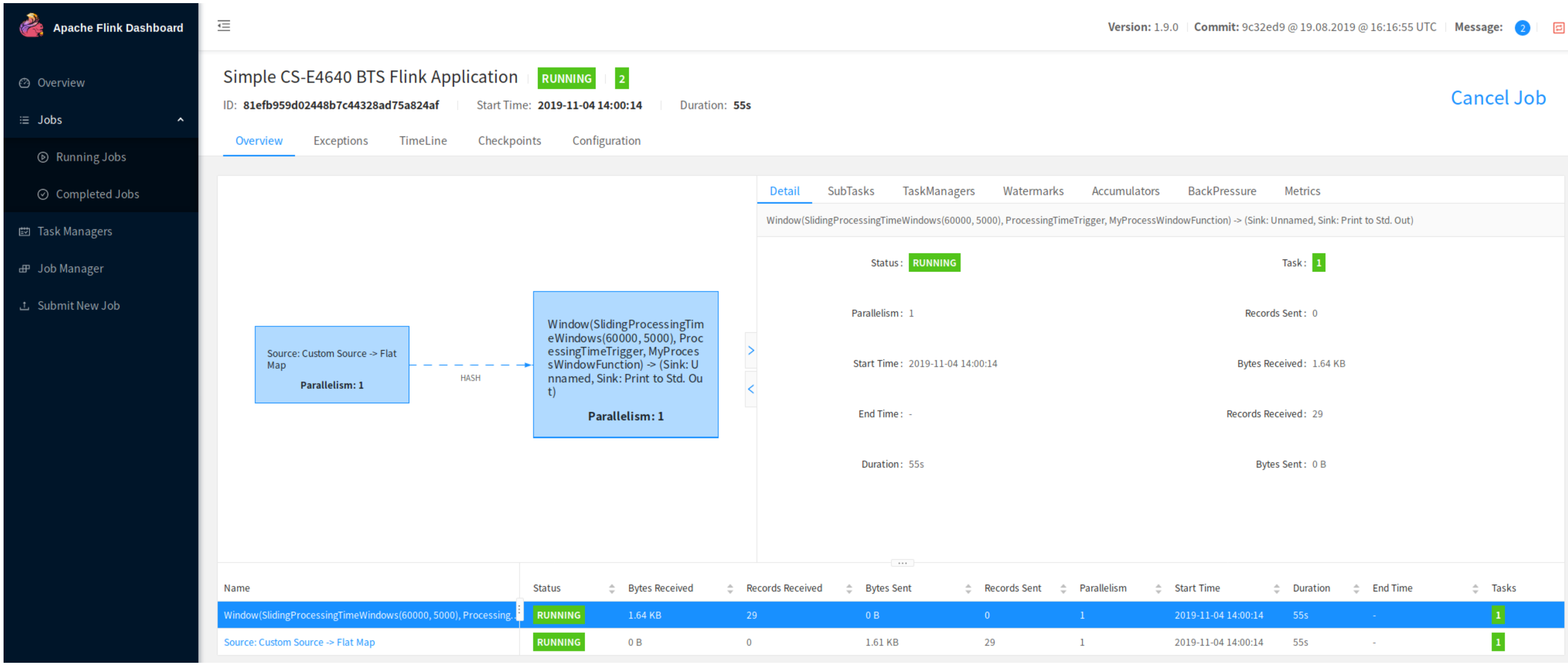
See the code in our git:

https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/blob/master/tutorials/streamingwithflink/

Simple example



Monitoring


Apache Flink Dashboard
Overview
Jobs
Running Jobs
Completed Jobs
Task Managers
Job Manager
Submit New Job
Version: 1.9.0 Commit: 9c32ed9 @ 19.08.2019 @ 16:16:55 UTC Message: 2
Simple CS-E4640 BTS Flink Application RUNNING 2
ID: 81efb959d02448b7c44328ad75a824af Start Time: 2019-11-04 14:00:14 Duration: 55s
Cancel Job
Overview Exceptions TimeLine Checkpoints Configuration
Detail SubTasks TaskManagers Watermarks Accumulators BackPressure Metrics
Window(SlidingProcessingTimeWindows(60000, 5000), ProcessingTimeTrigger, MyProcessWindowFunction) -> (Sink: Unnamed, Sink: Print to Std. Out)
Status: RUNNING Task: 1
Parallelism: 1 Records Sent: 0
Start Time: 2019-11-04 14:00:14 Bytes Received: 1.64 KB
End Time: - Records Received: 29
Duration: 55s Bytes Sent: 0 B
Source: Custom Source -> Flat Map
Parallelism: 1
HASH
Window(SlidingProcessingTimeWindows(60000, 5000), ProcessingTimeTrigger, MyProcessWindowFunction) -> (Sink: Unnamed, Sink: Print to Std. Out)
Parallelism: 1
Name | Status | Bytes Received | Records Received | Bytes Sent | Records Sent | Parallelism | Start Time | Duration | End Time | Tasks
Window(SlidingProcessingTimeWindows(60000, 5000), Processing... | RUNNING | 1.64 KB | 29 | 0 B | 0 | 1 | 2019-11-04 14:00:14 | 55s | - | 1
Source: Custom Source -> Flat Map | RUNNING | 0 B | 0 | 1.61 KB | 29 | 1 | 2019-11-04 14:00:14 | 55s | - | 1

One of the successful projects from Europe: originally from TU Berlin

Alibaba cloud:

"Flink can process over 472 million transactions per second during business peaks, which is truly astronomical"

Source: https://www.alibabacloud.com/blog/why-did-alibaba-choose-apache-flink-anyway_595190

Four Billion Records per Second! What is Behind Alibaba Double 11 - Flink Stream-Batch Unification Practice during Double 11 for the Very First Time

Alibaba Clouder December 2, 2020 2,607 0

Source: https://bit.ly/3sUTwqT

"Amazon Kinesis Data Analytics is the easiest way to transform and analyze streaming data in real time with Apache Flink" (From https://aws.amazon.com/kinesis/data-analytics/)

Interested in a use case?

Check:

https://www.alibabacloud.com/blog/application-of-real-time-compute-for-apache-flink-in-weibo_597955

Summary

- **Focus:**
  - Practical programming with one of the stacks:
    - *Apache Flink Stream API (with different connectors)*
    - *Apache Spark*
    - *Kafka Streams*
  - Check the common concepts in other tools/systems
- **Action:**
  - Work on use cases where you can use stream analytics (as a user/developer) → there are many interesting analytics
  - Provision services for stream processing (as a platform)

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io