Aalto University
School of Science

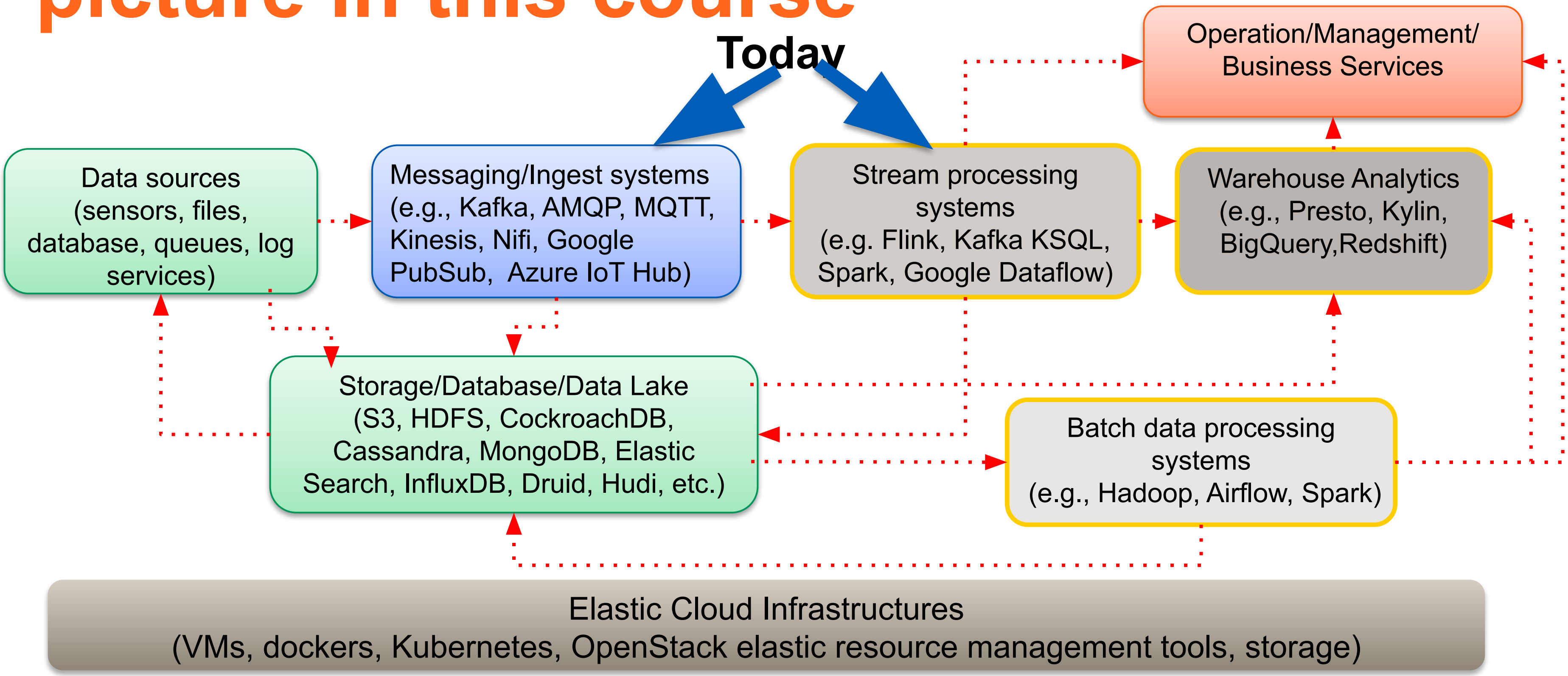
Stream Processing and Big Data Platforms

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



Today

Operation/Management/ Business Services

Data sources (sensors, files, database, queues, log services)

Messaging/Ingest systems (e.g., Kafka, AMQP, MQTT, Kinesis, Nifi, Google PubSub, Azure IoT Hub)

Stream processing systems (e.g. Flink, Kafka KSQL, Spark, Google Dataflow)

Warehouse Analytics (e.g., Presto, Kylin, BigQuery,Redshift)

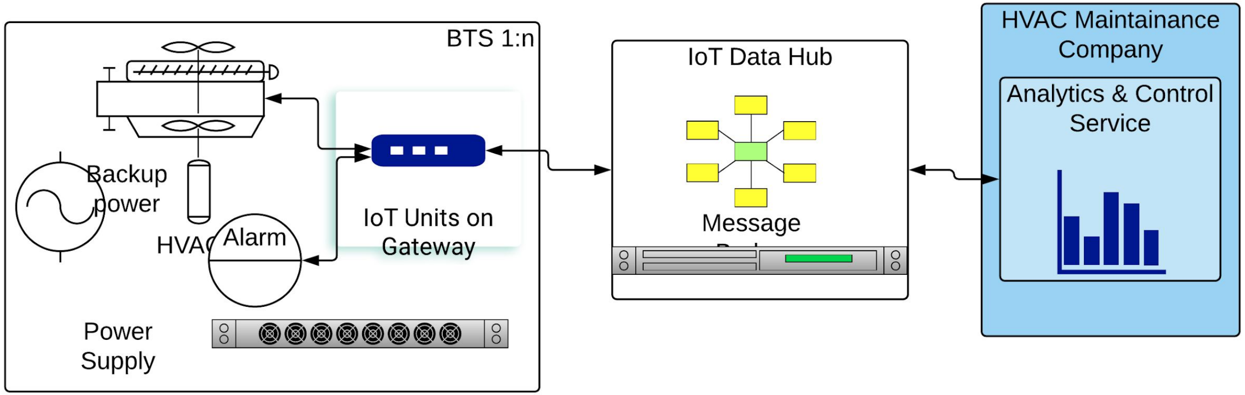
Storage/Database/Data Lake (S3, HDFS, CockroachDB, Cassandra, MongoDB, Elastic Search, InfluxDB, Druid, Hudi, etc.)

Batch data processing systems (e.g., Hadoop, Airflow, Spark)

Elastic Cloud Infrastructures (VMs, dockers, Kubernetes, OpenStack elastic resource management tools, storage)

Motivating examples

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



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

Aalto University
School of Science

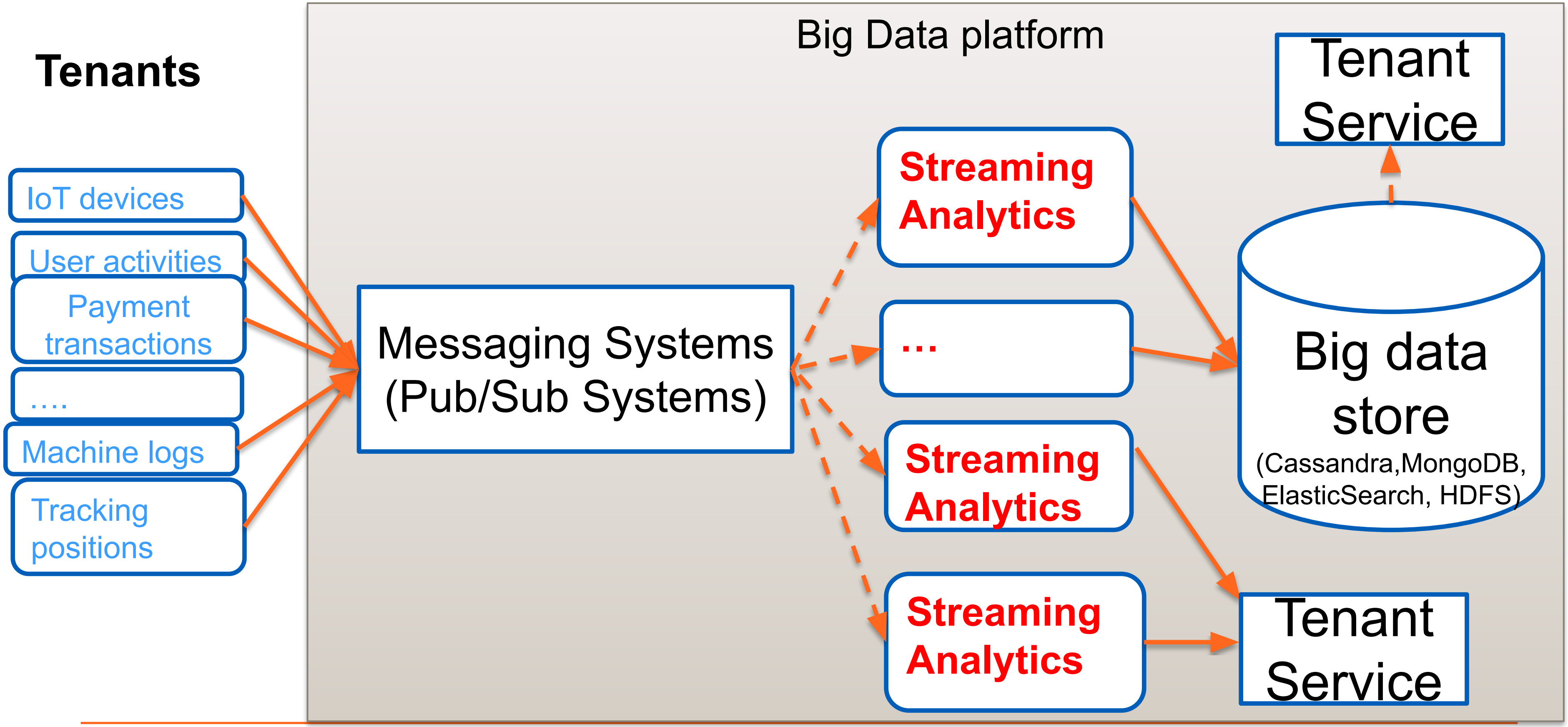
Stream analytics for data in motion

CS-E4640 Big Data Platforms, Spring 2023, Hong-Linh Truong
08/03/2023

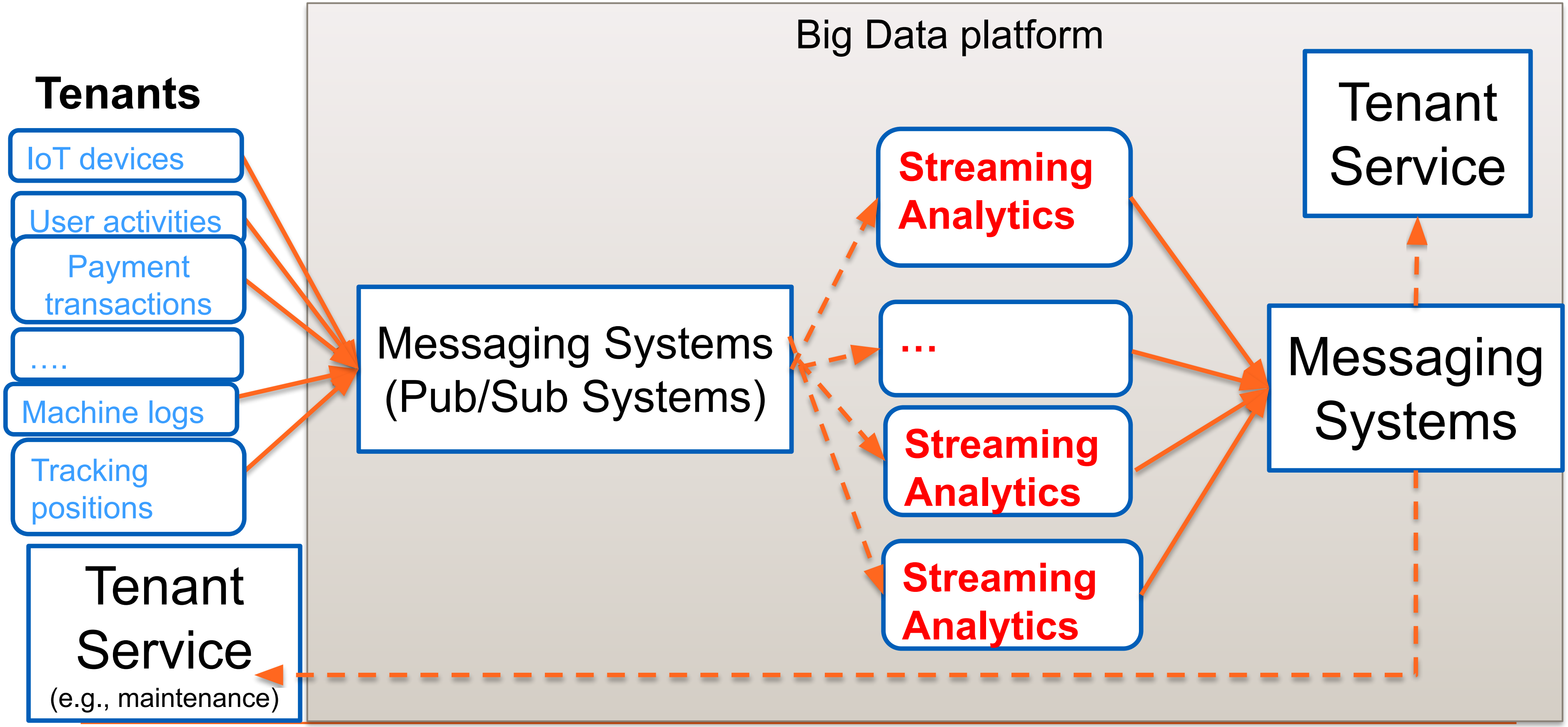
Stream processing in big data

- **Processing big data coming from streams at near real time**
  - the data element/unit may be “small” but voluminous and delivered in a near-real time manner
  - high and volatile throughput, but low service latency expected
- **Require large-scale computing infrastructures and many other platform services**
  - task parallelism: multiple tasks for processing data
  - data parallelism: data is partitioned into concurrent/parallel data streams □ distributed, parallel processing tasks

Near realtime streaming data processing



Near realtime streaming data processing



Example in the cloud – stream processing and big data platforms

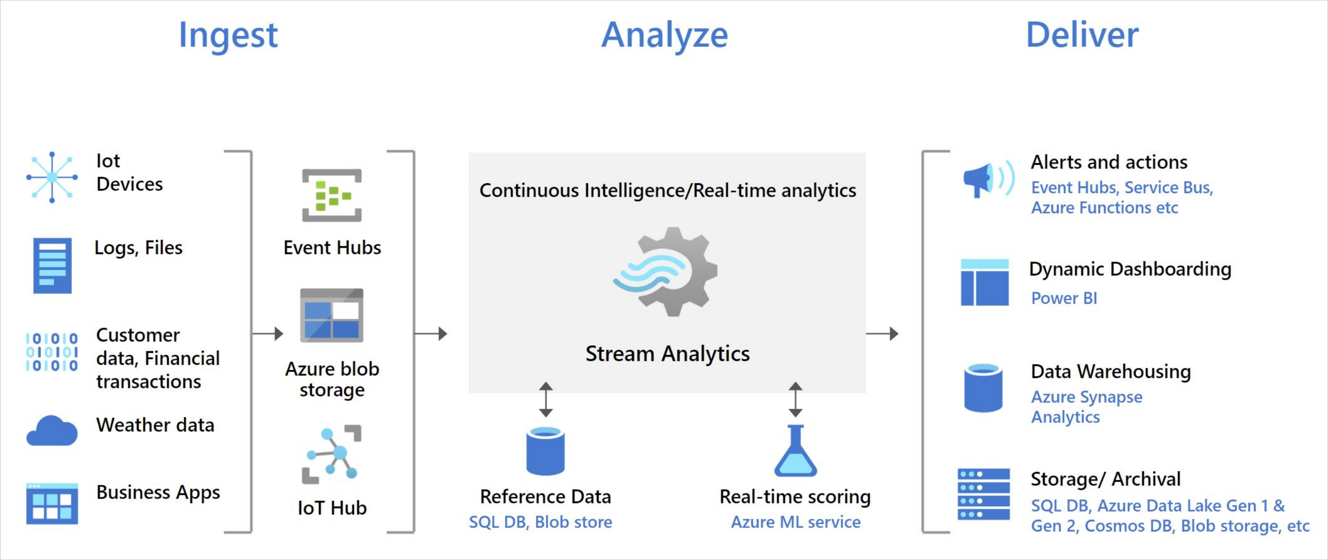


Figure source: https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction

Known public cloud services: Amazon Kinesis, Google Dataflow, Alibaba Cloud DataHub

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 modern technologies like: Apache Flink, Apache Kafka and Apache Spark, which are used intensively in business systems and big cloud platforms

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, including near-real time machine learning
- **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, real time AI/ML

Stream Processing – key concepts

Common concepts

- **The way to connect data to streams and obtain messages from the streams**
  - focusing very much on *connector concepts* and well-defined event structures
  - the data can be pulled/pushed via connectors
- **The way to specify/program the "analytics" logic**
  - *analytics functions, statements* and how they are glued together to process flows of messages
  - 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 the result to external components (databases, new streams, files)**

Data stream programming

Data stream: a sequence/flow of data units

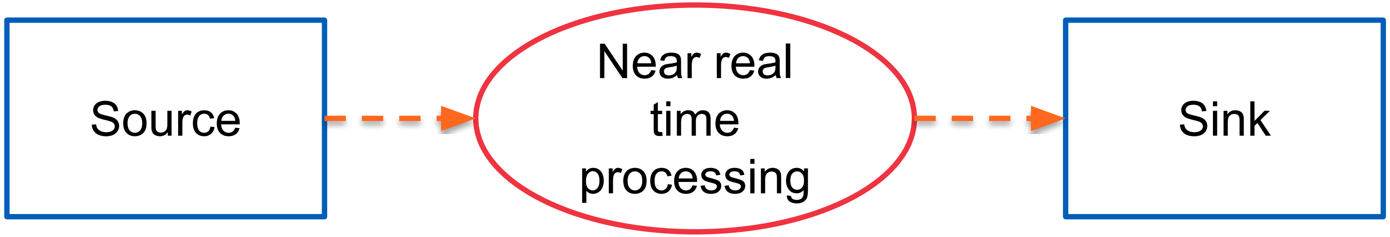
Data units are defined by applications: a data unit can be data described by a primitive data type or by a complex data type, a serializable object, etc.

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, log records)

Messages of events/data records

- **messages encapsulating real-world events, data records and other types of data**
- **data to be sent/processed can be in a simple or complex structure**



Split data based on keys or not? One message vs batch of messages

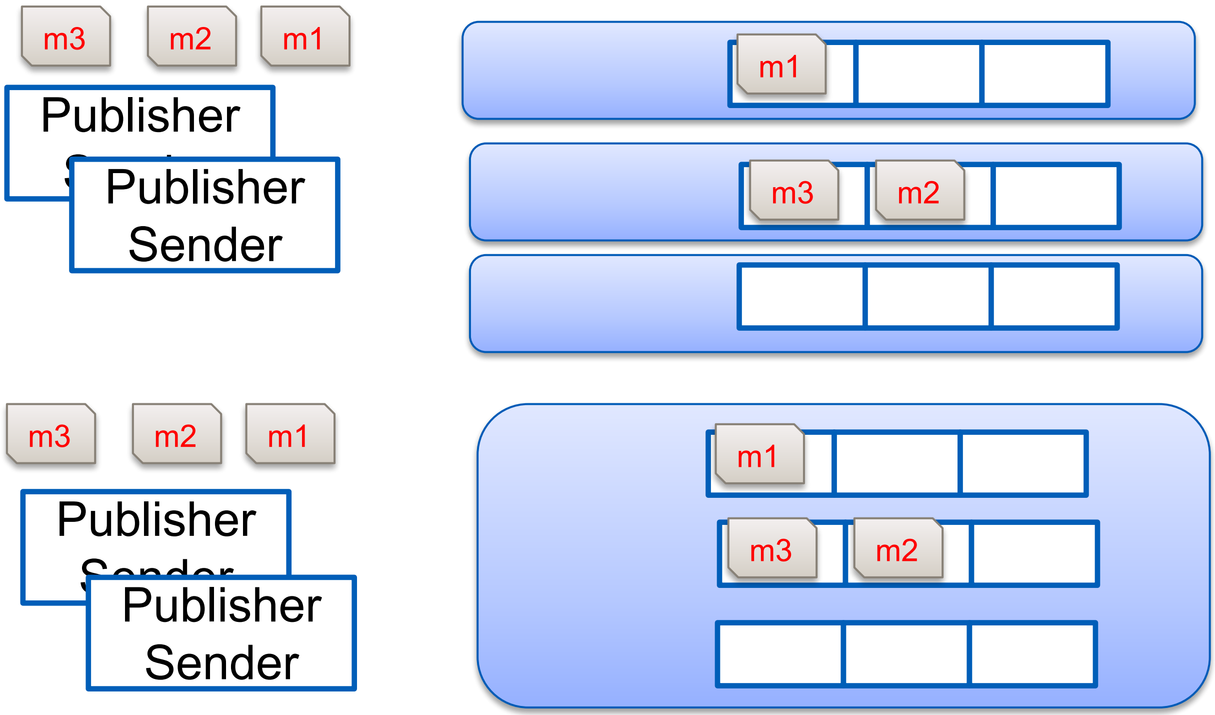
We focus on unbounded discrete messages of data

Message representations and streams

- **Data Sources:**
  - via message brokers, database, websocket, different IO adapters/connectors, etc.
- **Data Sinks**
  - messaging systems, databases, files, etc.
- **Data representations**
  - text, POJO (Plain Old Java Object), CSV, JSON, Arvo format, etc.
  - serialization and deserialization (short name: SerDe) are required
  - data format validation
  - data schema registry

Publisher view: how messages are published

Messaging system

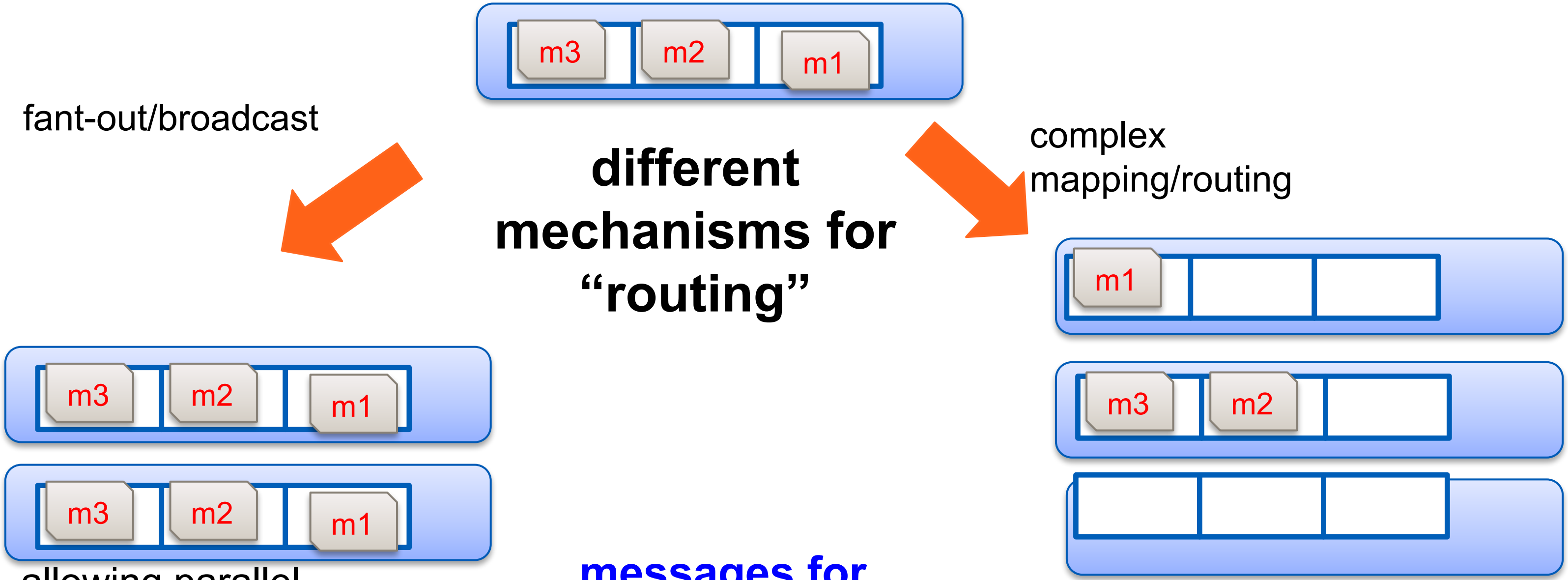


topic=queue;
no partition

topic = n
partitions = n
queues

Topic and topic partitions

How messages are handled for consumption



Messages in systems

fant-out/broadcast

different mechanisms for “routing”

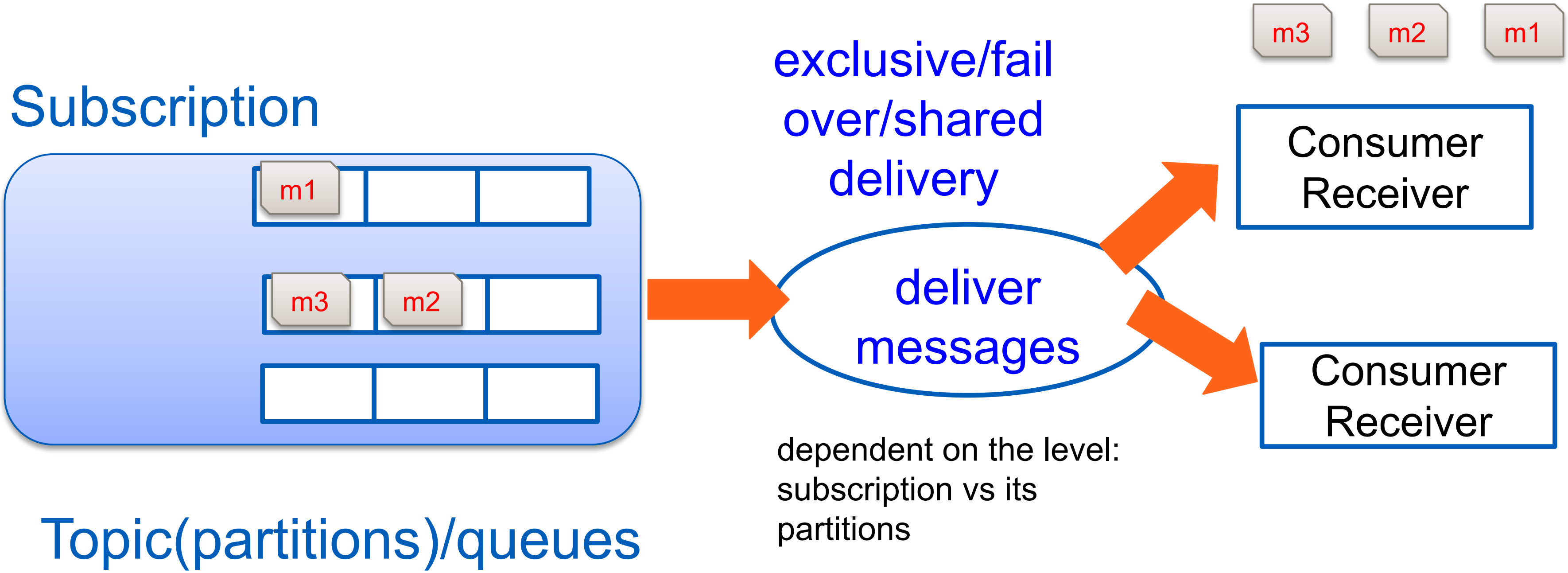
complex mapping/routing

allowing parallel processing of the same messages

messages for consumption

parallel processing of different messages

Consumer view in accessing messages: subscription and delivery

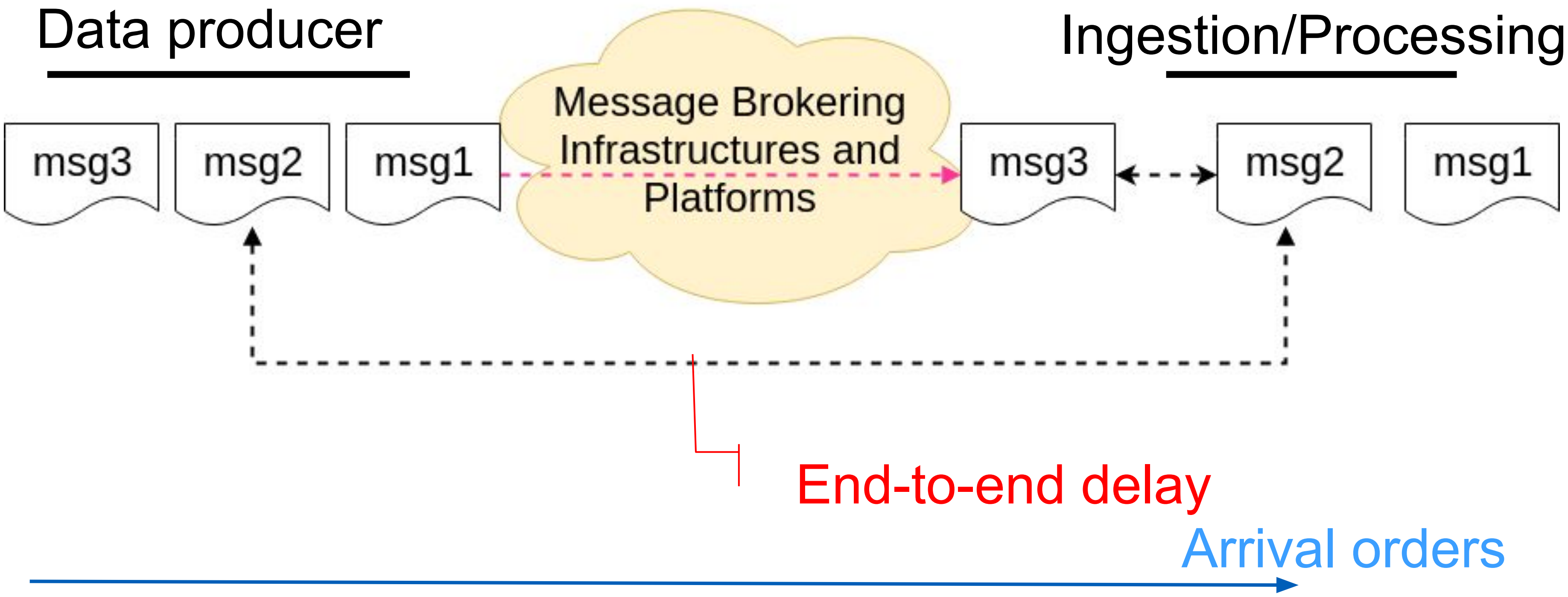


dependent on the level: subscription vs its partitions

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

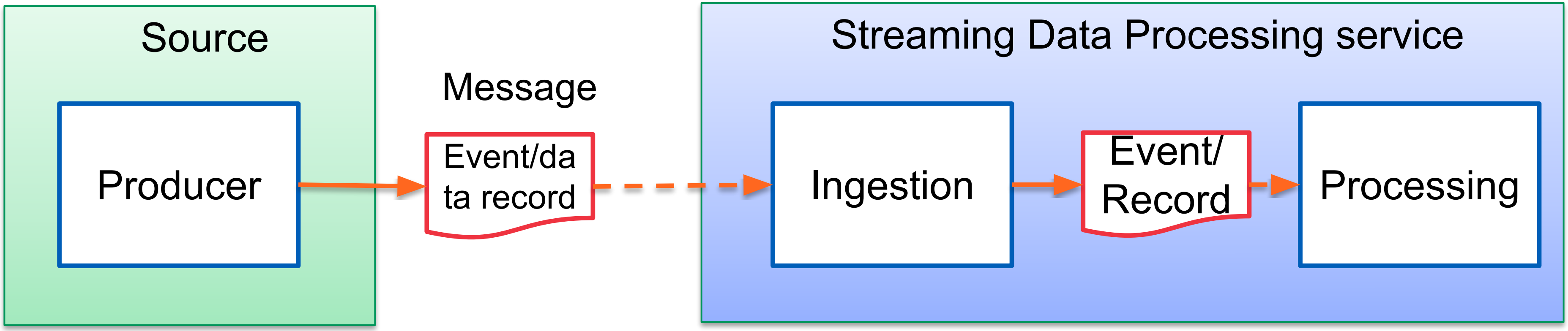


Without event/record time, do we know the delay or out of order?

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



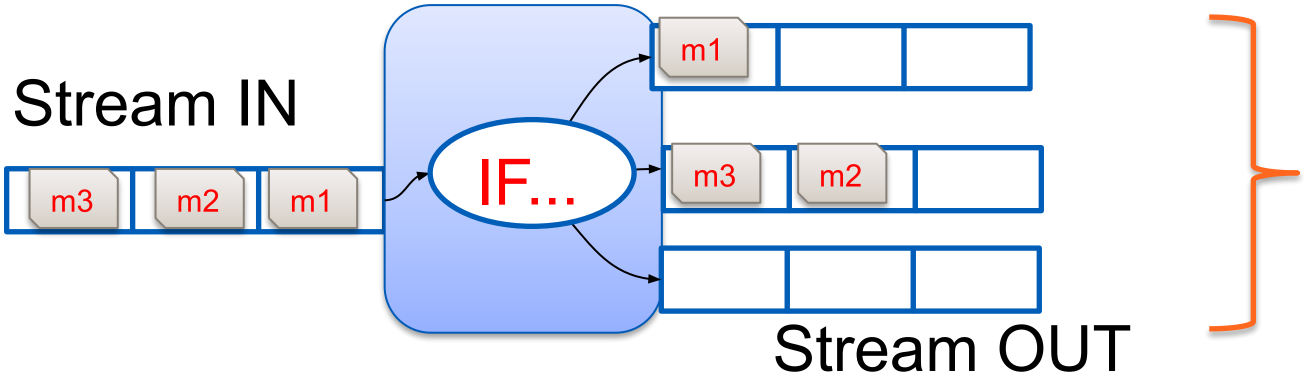
Event time

Ingestion time
(when the message is entered into the system)

Processing time

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

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



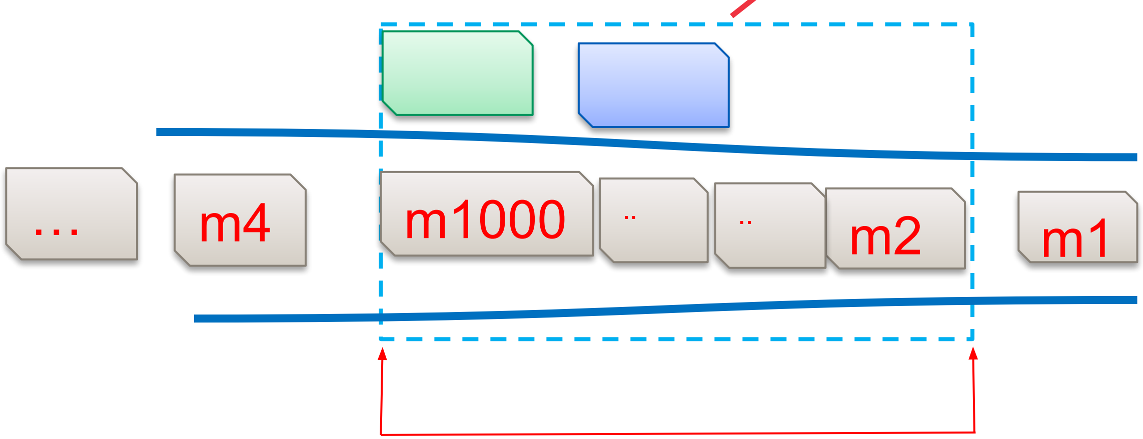
Can be processed in parallel using multiple nodes

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?



a stream of events

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

Functions

- **Can be simple or complex!**
- **Core for analytics and ML**
- **Examples**
  - individual threshold/alarm based alerting, atypical events monitoring
  - anomaly detection based on statistical functions, like quantile/T-digest, ...
  - real time AI/machine learning

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

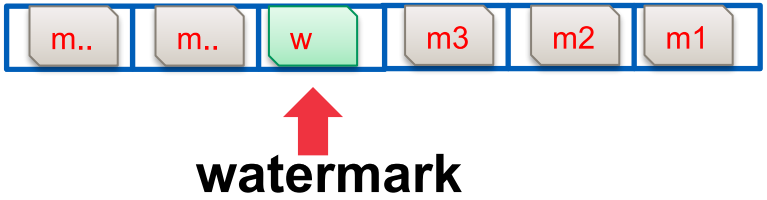
What if events/records come late into the windows?

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

correctness and completeness issues

Support lateness

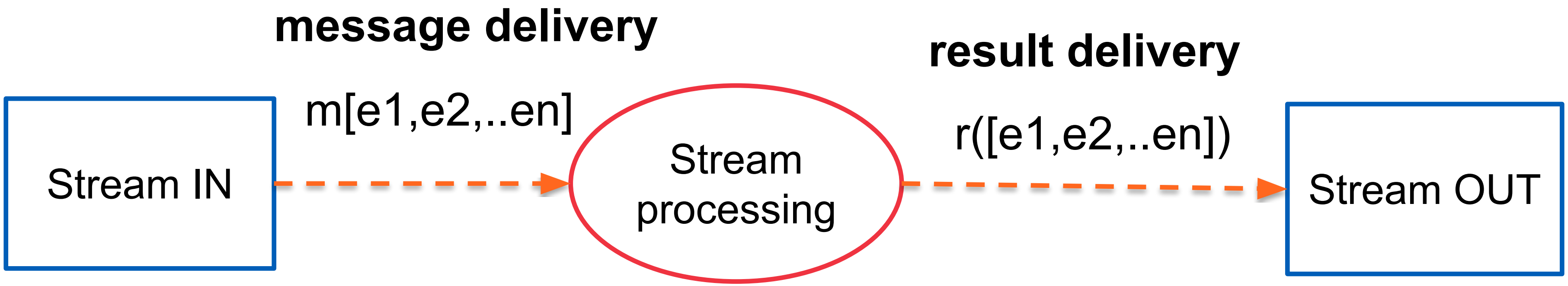
- **Identify timestamp of events/data records**
- **Identify watermark in streams**
  - a watermark is a timestamp
  - a watermark indicates that no events which are older that the watermark should be processed
  - enable the delay of processing functions to wait for late events
- **Using watermark to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



Delivery guarantees

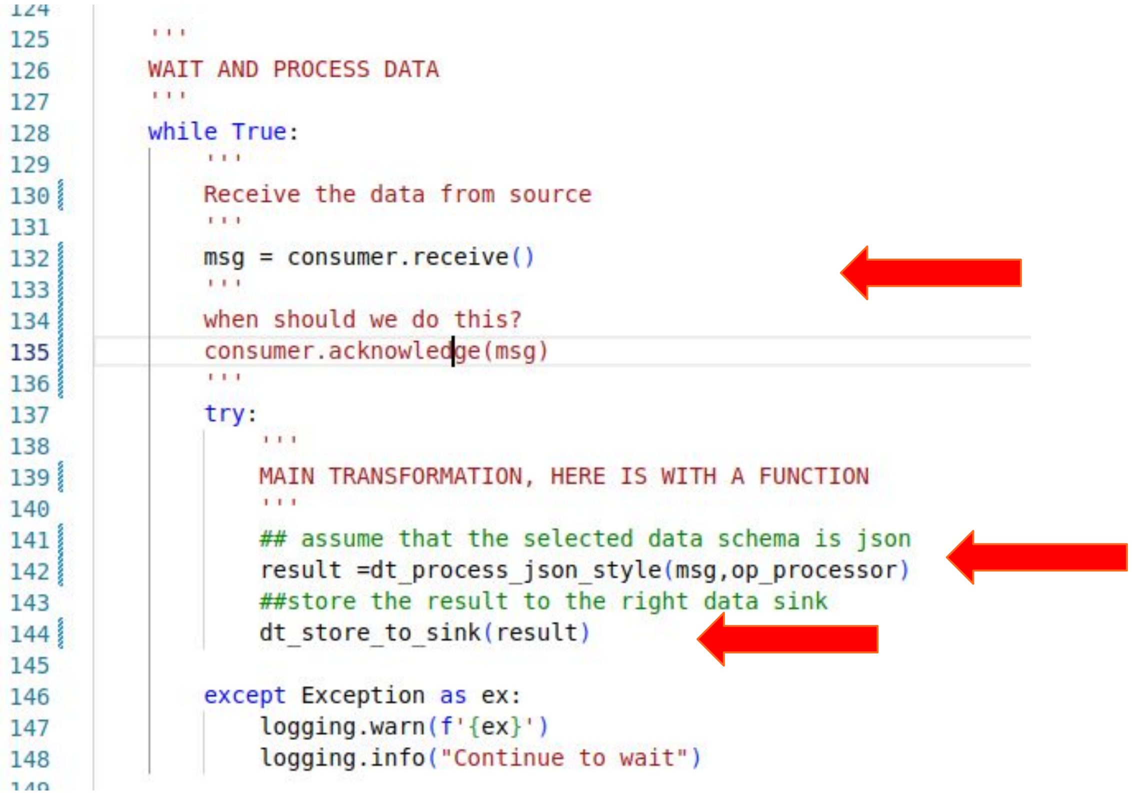
Exactly once? at least once? or at-most-once

End-to-end?



What if the stream processing fails and restarts

Examine a simple example



```
    '''
    WAIT AND PROCESS DATA
    '''
    while True:
        '''
        Receive the data from source
        '''
        msg = consumer.receive()
        '''
        when should we do this?
        consumer.acknowledge(msg)
        '''
        try:
            '''
            MAIN TRANSFORMATION, HERE IS WITH A FUNCTION
            '''
            ## assume that the selected data schema is json
            result =dt_process_json_style(msg,op_processor)
            ##store the result to the right data sink
            dt_store_to_sink(result)

        except Exception as ex:
            logging.warn(f'{ex}')
            logging.info("Continue to wait")
```

How to handle possible errors

Note: Example with a Pulsar consumer for data transformation

Message and processing guarantees

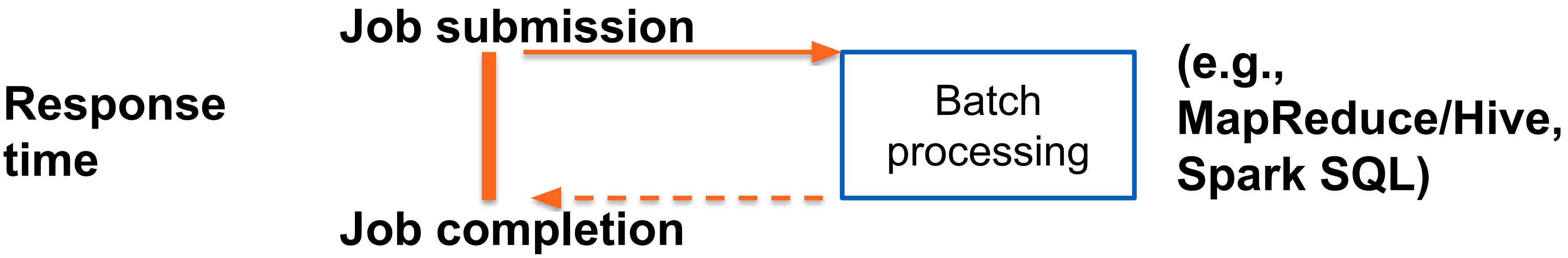
- Message guarantees are the job of the broker/messaging system
- Processing guarantees are the job of the stream processing frameworks
- They are highly connected if messaging systems and processing frameworks are tightly coupled (e.g., Kafka case)

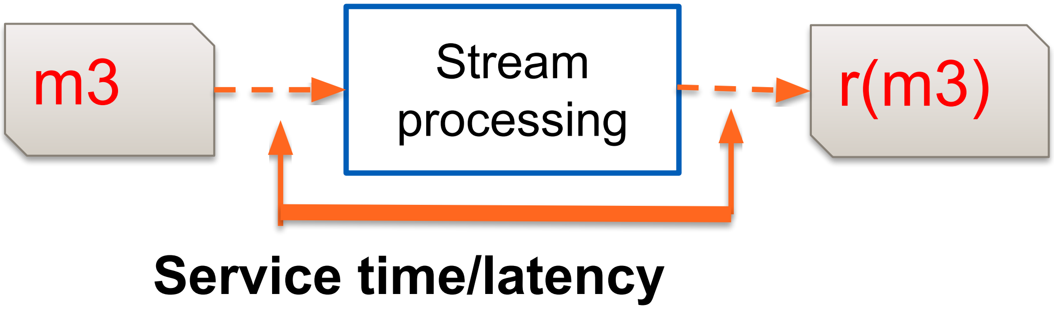
End-to-end exactly once

- **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





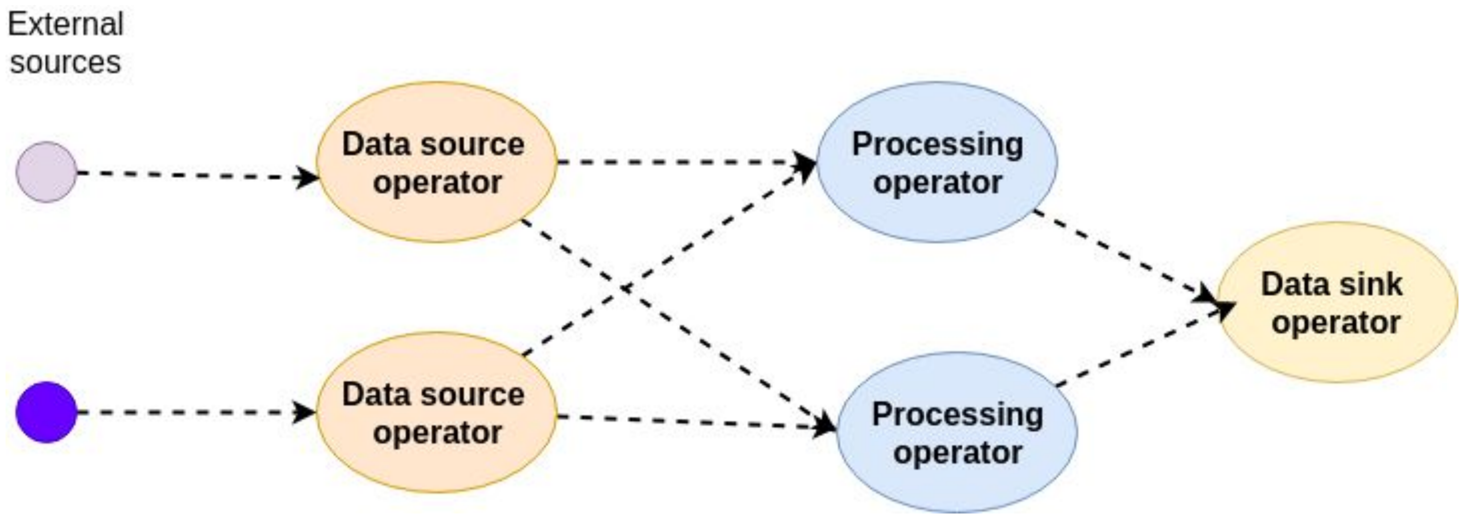
Latency and Throughput

- **Service latency**
  - subseconds! e.g., milliseconds
  - max, min or percentile? ⇒ up to application requirements
- **Throughput**
  - how many messages 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)

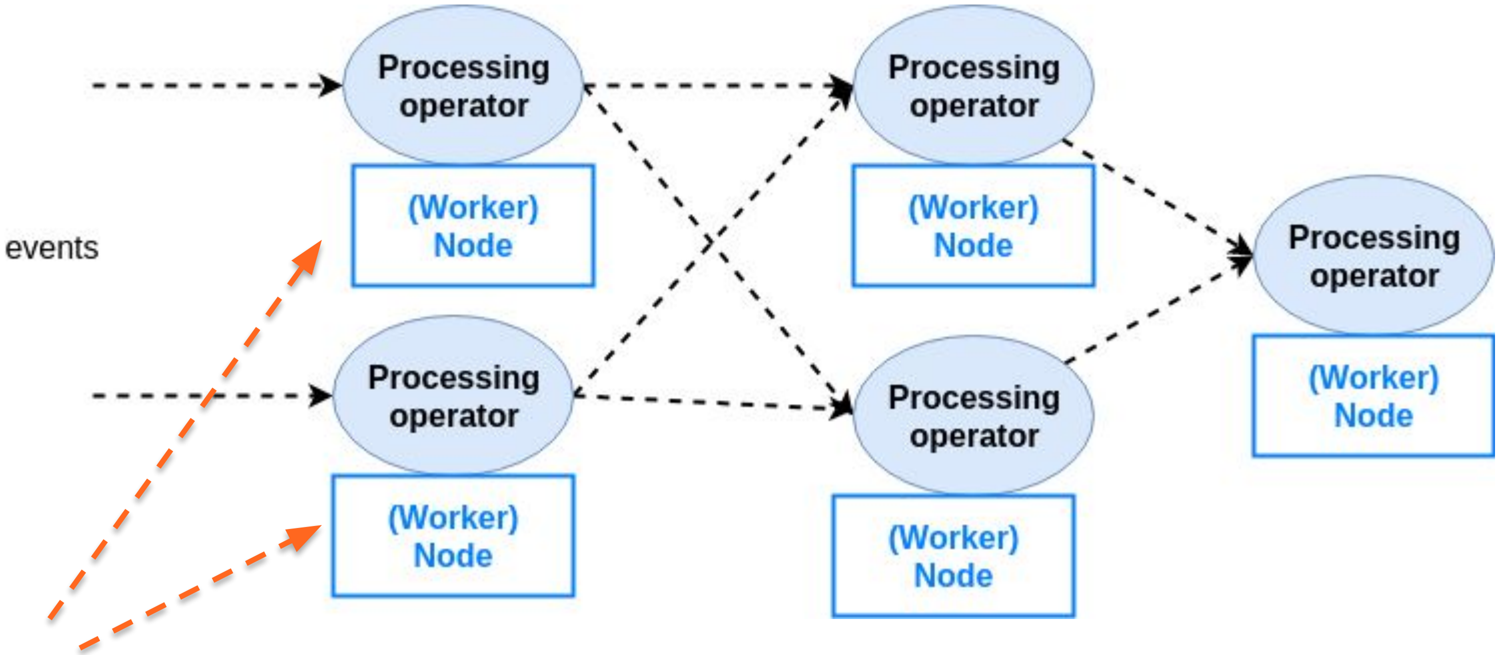
Structure of streaming data processing programs (2)



- Data source operators: represent sources of streams
- Processing operators: represent processing functions

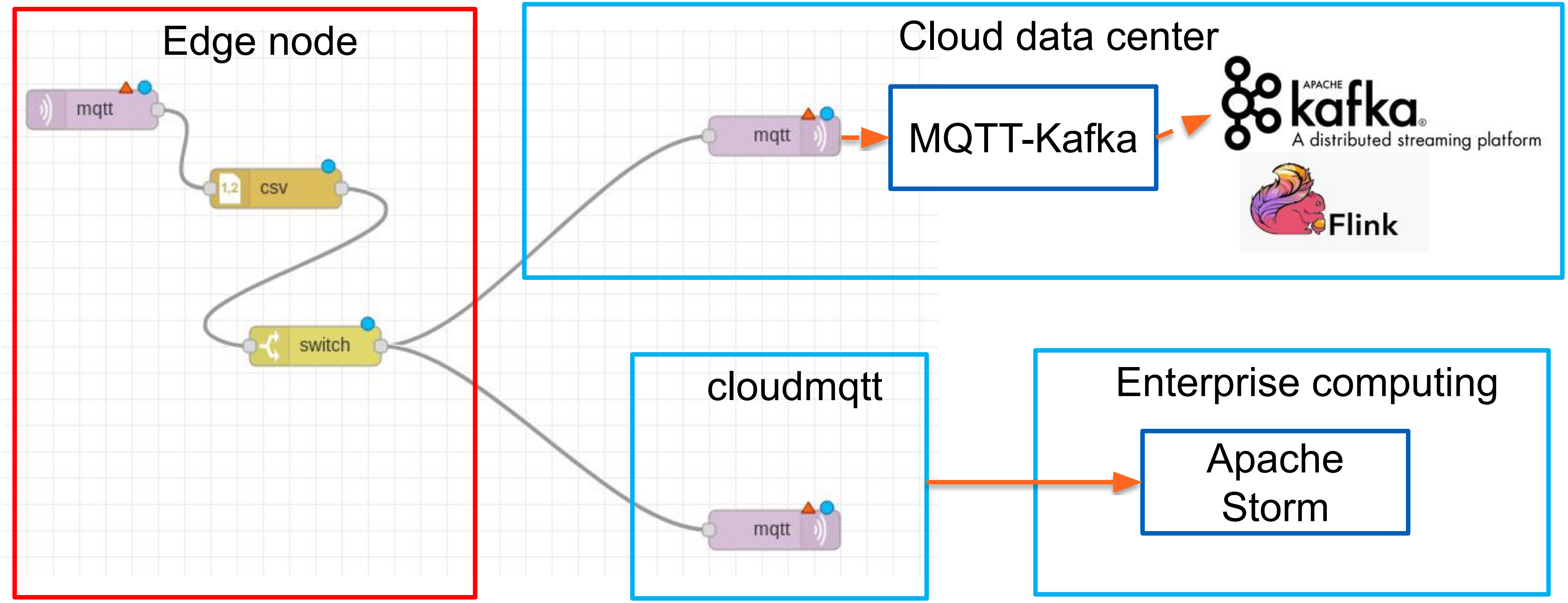
Distributed processing topology in a cluster

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



Nodes of a cluster (VMs, containers, Kubernetes)

Distributed, composable processing topologies in cross distributed sites



Aalto University
School of Science

CS-E4640 Big Data Platforms, Spring 2023, Hong-Linh Truong
08/03/2023

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**

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

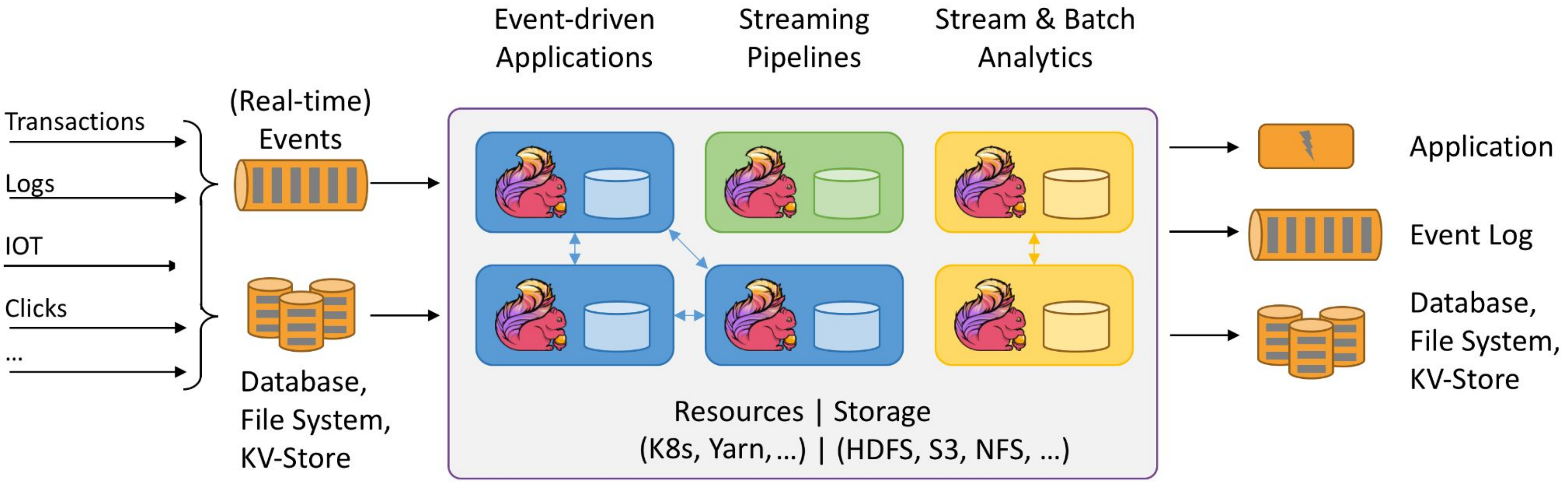
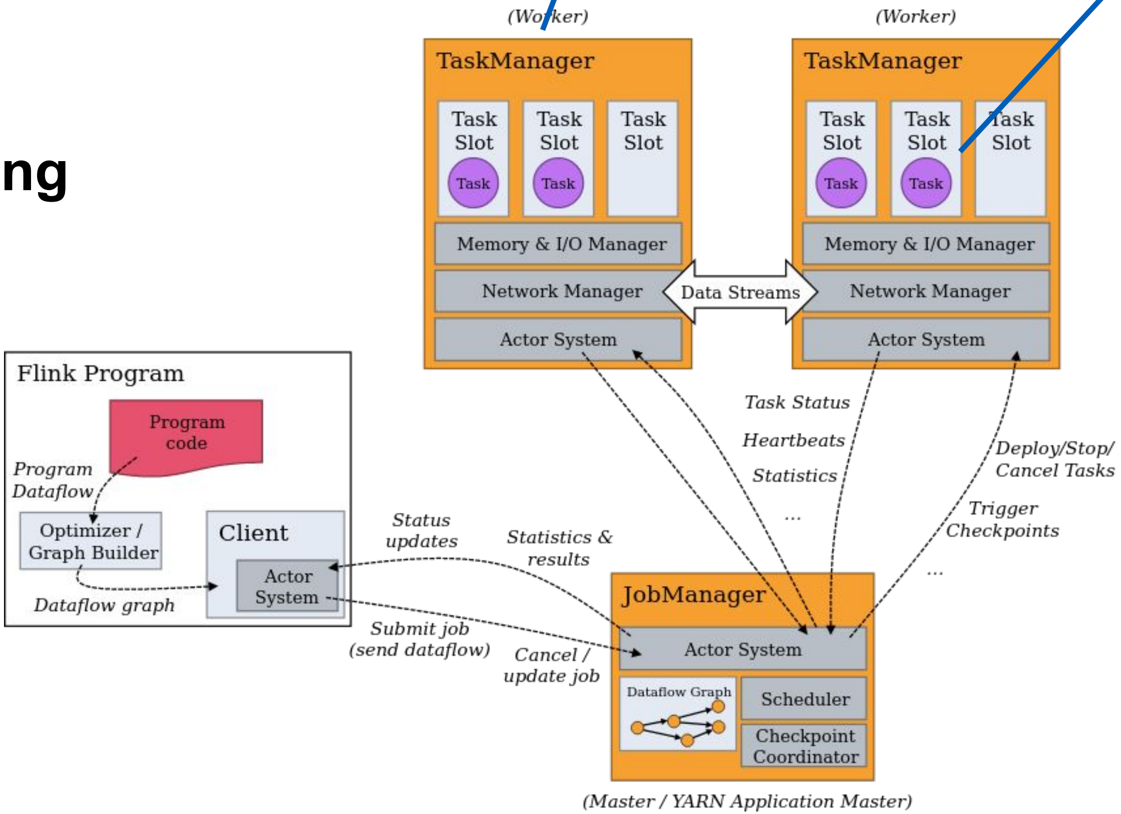


Figure source: https://flink.apache.org/

Flink runtime view

- **Parallelism**
- **Checkpointing**
- **Monitoring**

Remember 24/7

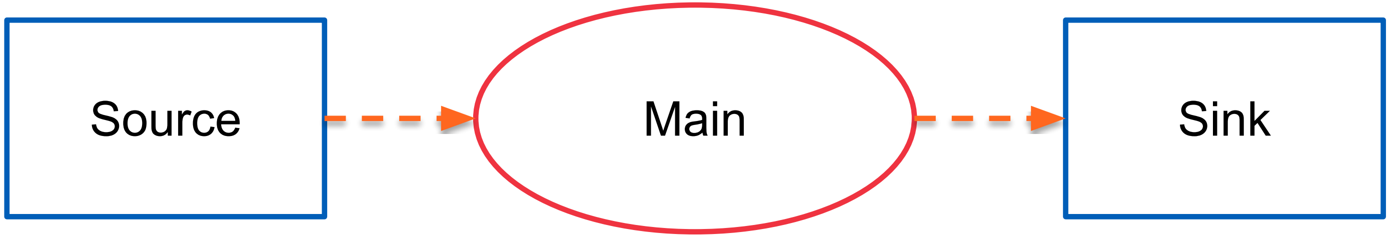


JVM Process

Operators within a task

Figure source: https://nightlies.apache.org/flink/flink-docs-release-1.16/docs/concepts/flink-architecture/

Main elements in Flink applications



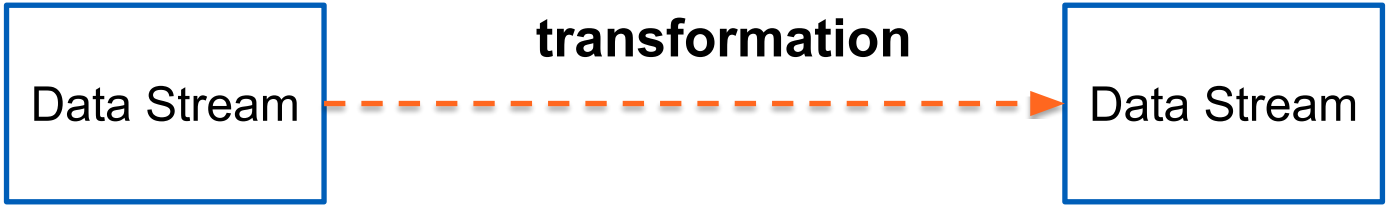
- **Rich set of sources and sinks via many connectors**

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

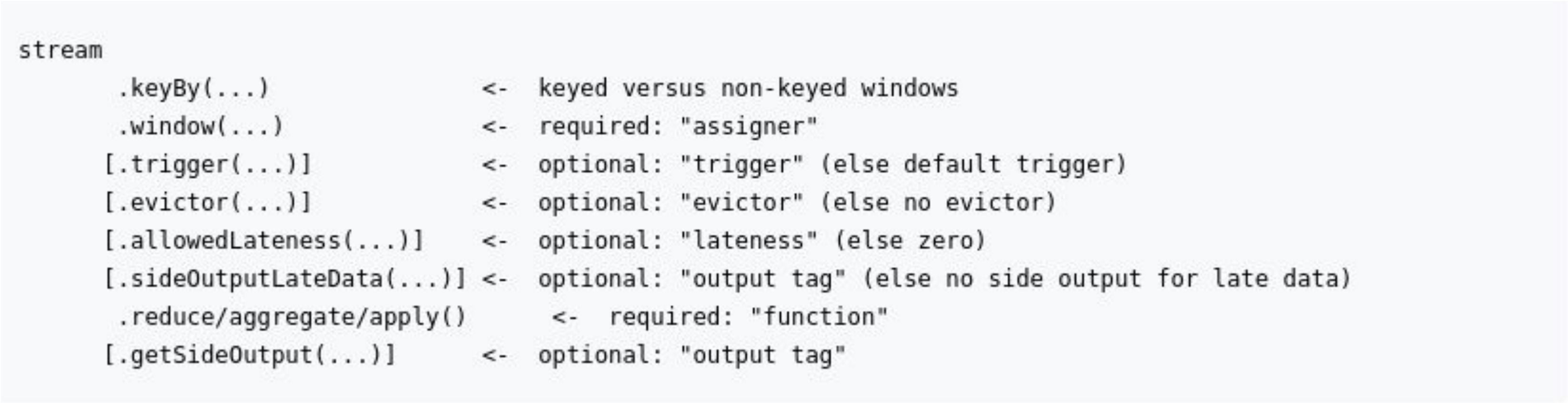


Bounded and unbounded streams

Stream processing flows

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

Keyed data/Keyed window: if we can separate data via keys

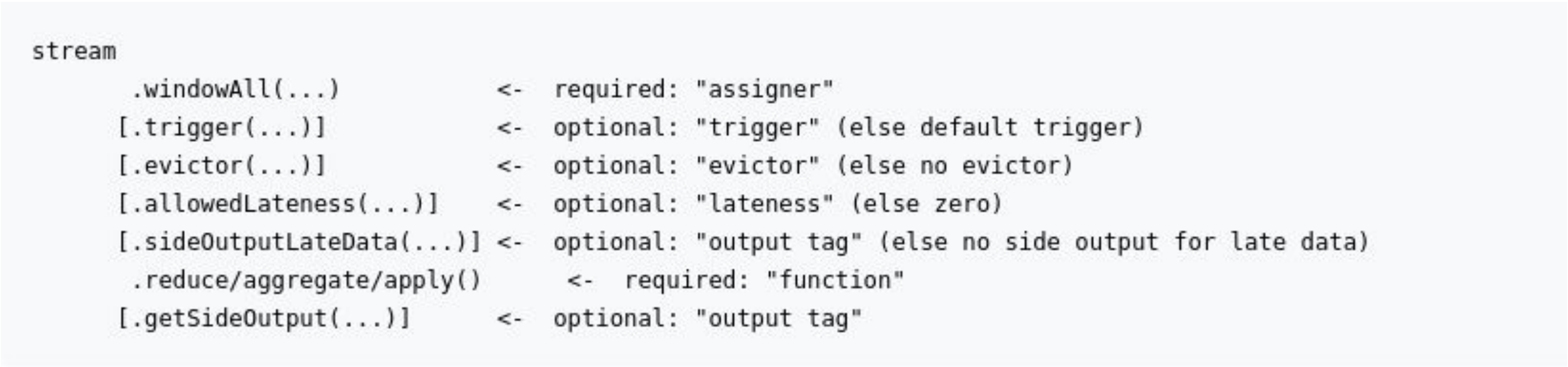


```
stream
       .keyBy(...)               <-  keyed versus non-keyed windows
       .window(...)              <-  required: "assigner"
      [.trigger(...)]            <-  optional: "trigger" (else default trigger)
      [.evictor(...)]            <-  optional: "evictor" (else no evictor)
      [.allowedLateness(...)]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

Source: https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/operators/windows/

Stream processing flows

Handling streaming data without a key for analytics purposes

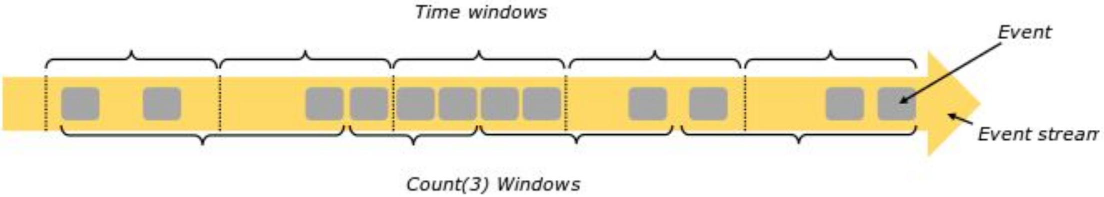


```
stream
       .windowAll(...)           <-  required: "assigner"
      [.trigger(...)]            <-  optional: "trigger" (else default trigger)
      [.evictor(...)]            <-  optional: "evictor" (else no evictor)
      [.allowedLateness(...)]    <-  optional: "lateness" (else zero)
      [.sideOutputLateData(...)]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

Source: https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/operators/windows/

Windows and Times

Windows



Times

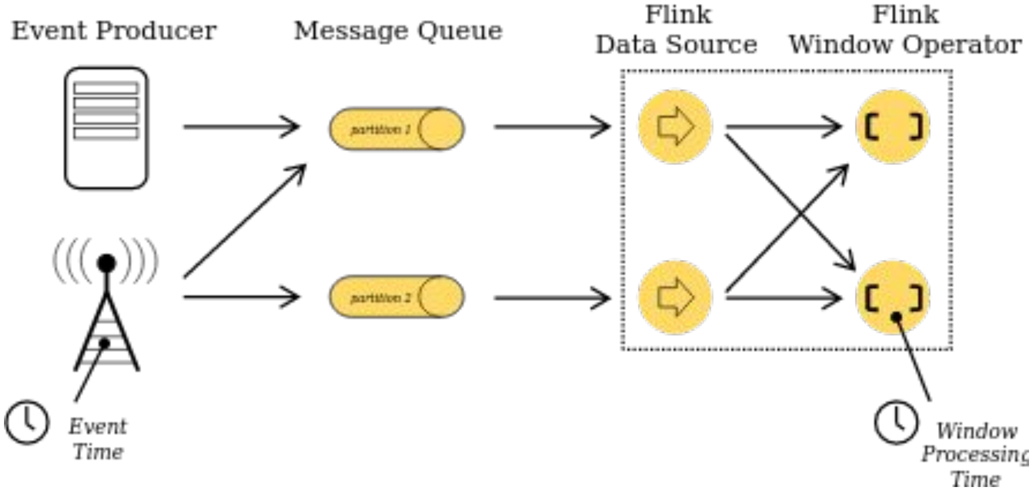
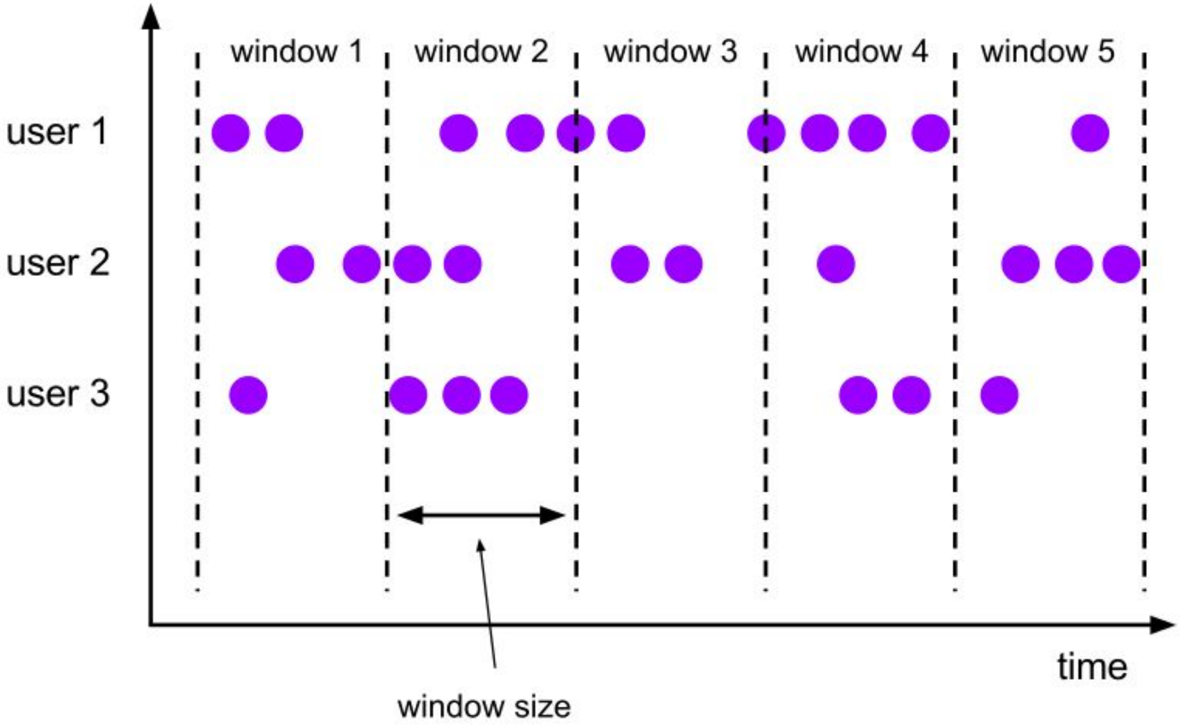


Figure source: https://nightlies.apache.org/flink/flink-docs-master/docs/concepts/time/

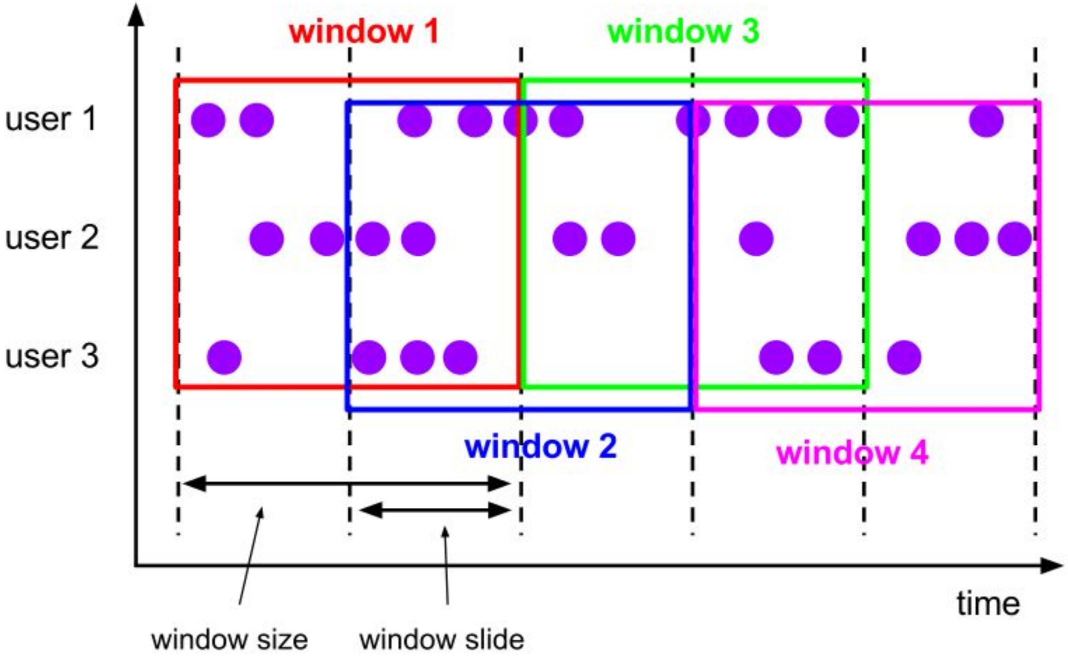
Batch/Tumbling Windows



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

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

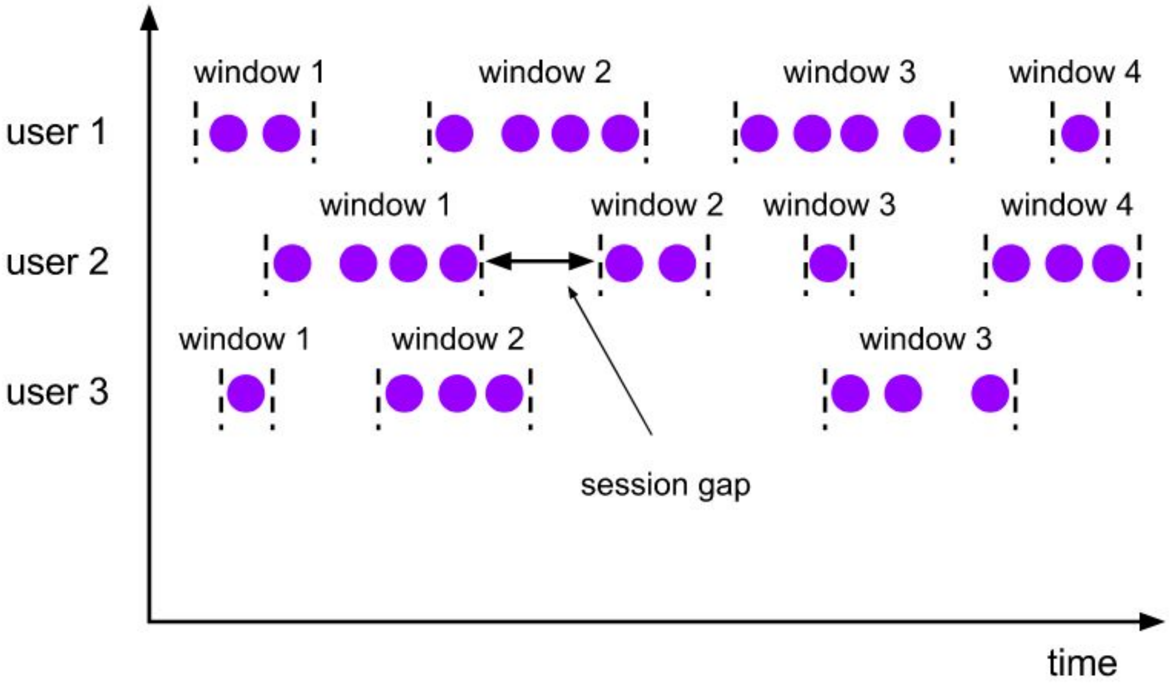
Sliding windows



Use cases:
Moving average

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

Session Windows



Use cases: Web/user activities clicks

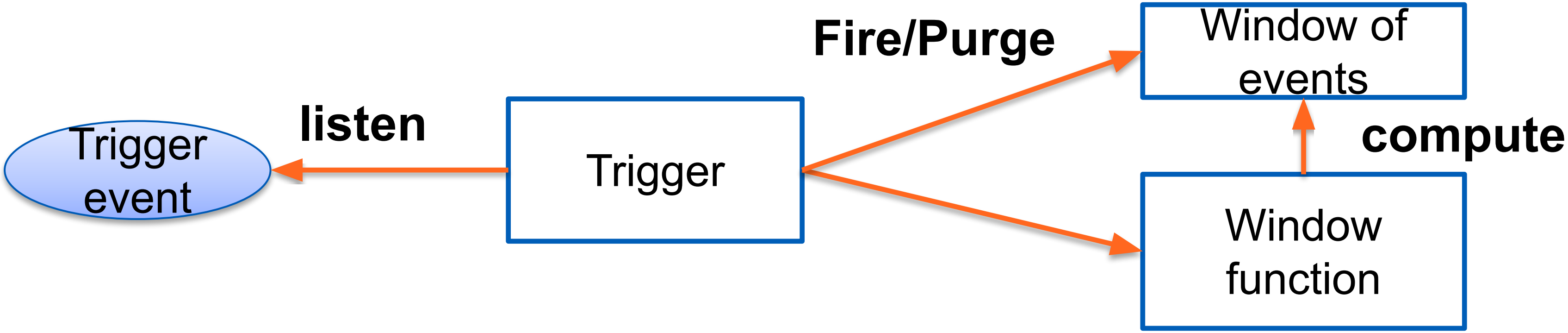
Figure source: https://nightlies.apache.org/flink/flink-docs-master/docs/dev/datastream/operators/windows/

Window Functions

- **Reduce Function**
  - Reduce through the combination of two inputs
- **Aggregate Function**
  - Add an input into an accumulator
- **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

- **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**

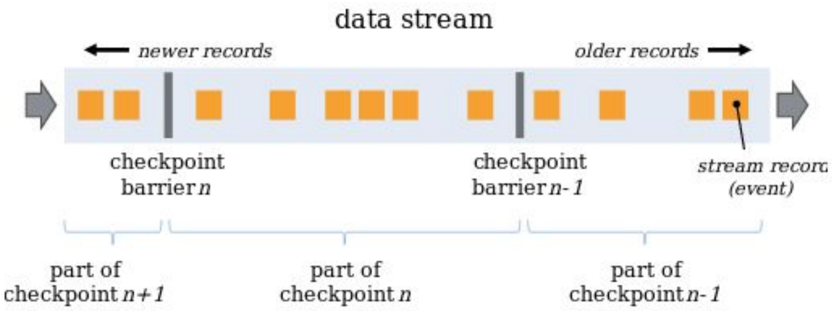


Figure source:
https://nightlies.apache.org/flink/flink-docs-release-1.16/docs/learn-flink/fault_tolerance/

Example with Base Transceiver Station

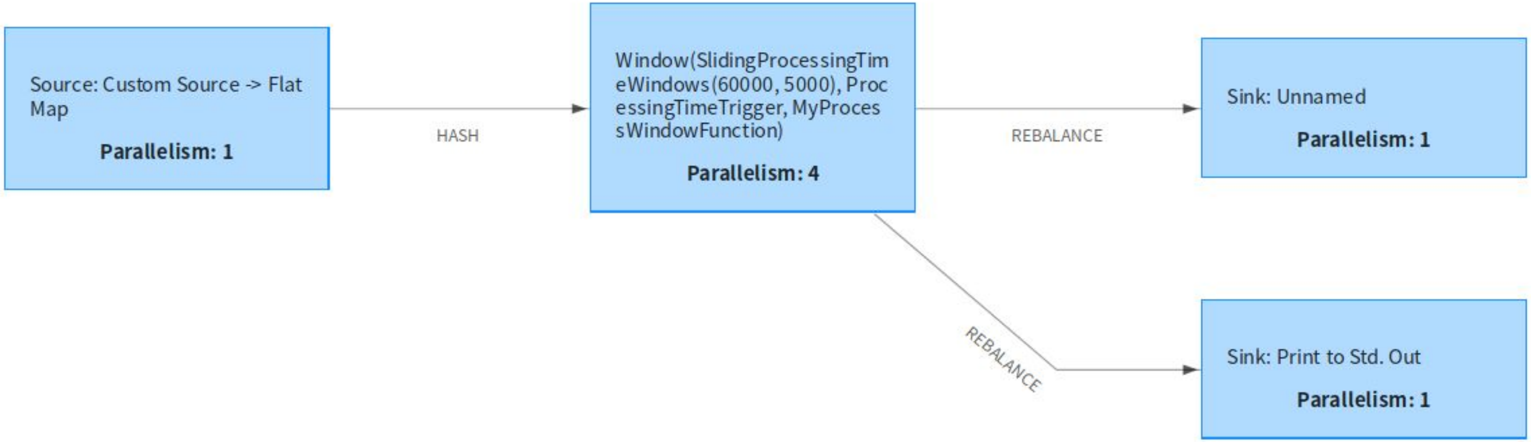
Data in our git

```
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,
```

See the code in our git:

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

Simple example



Source: Custom Source -> Flat Map

Parallelism: 1

Window(SlidingProcessingTimeWindows(60000, 5000), ProcessingTimeTrigger, MyProcessWindowFunction)

Parallelism: 4

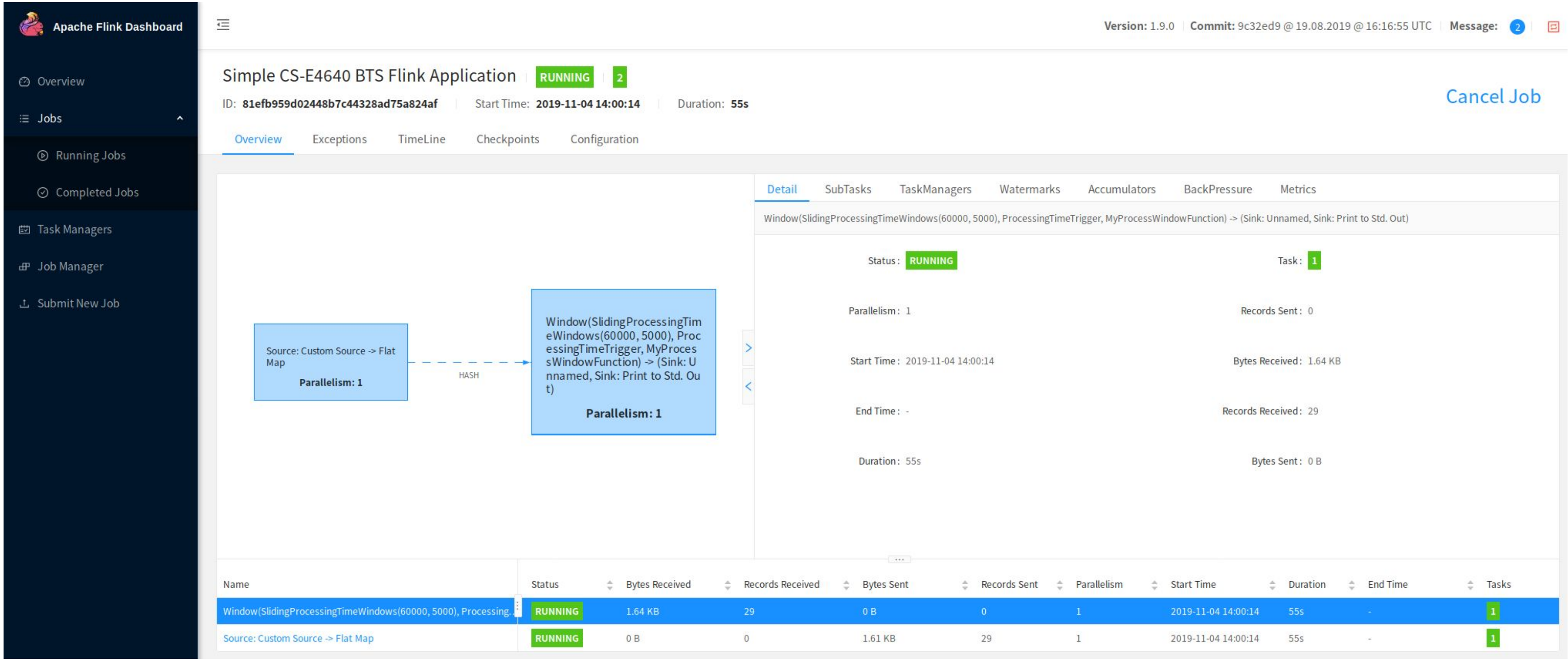
Sink: Unnamed

Parallelism: 1

Sink: Print to Std. Out

Parallelism: 1

Monitoring



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**

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

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

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