Aalto University
School of Science

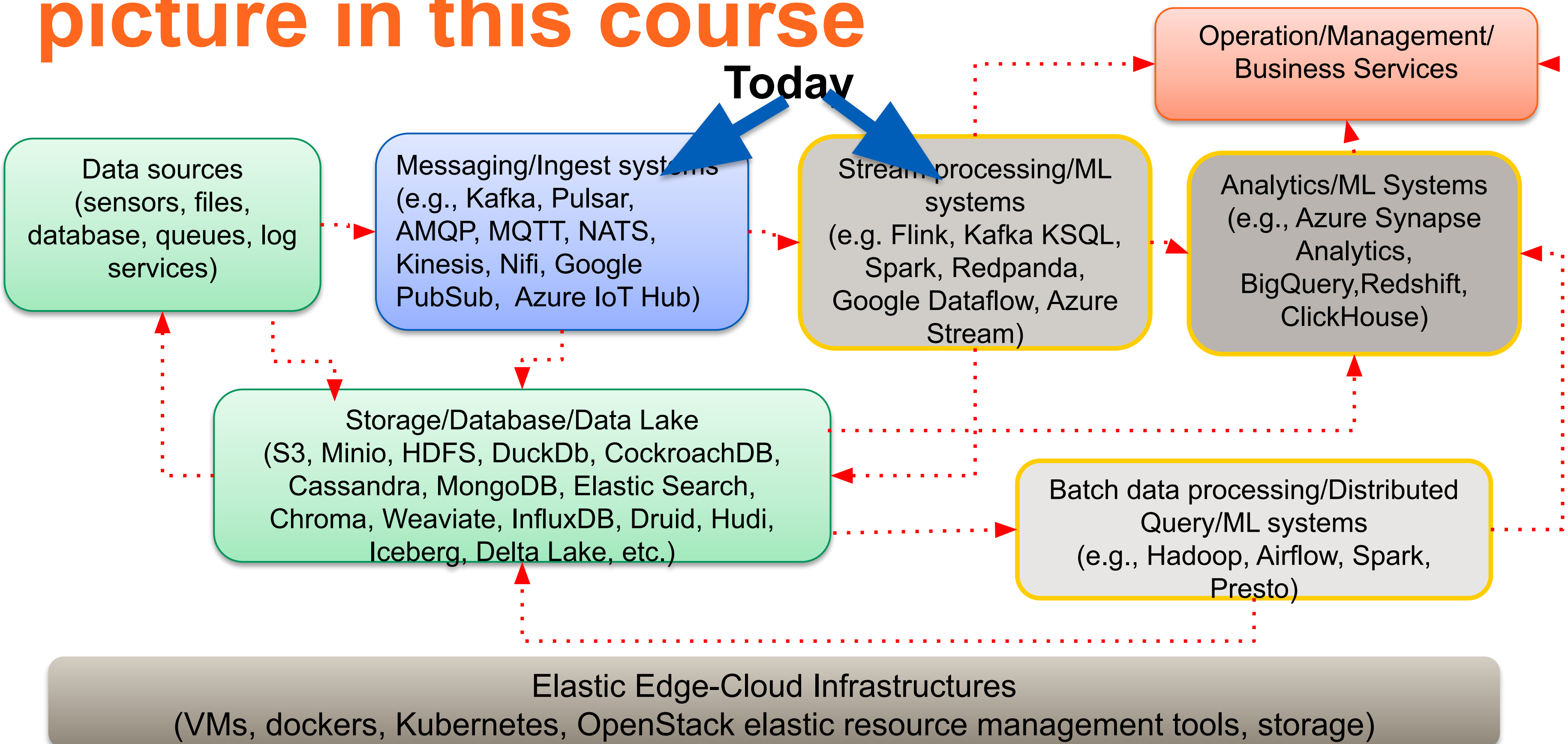
Stream Processing and Big Data Platforms

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 streaming 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



Aalto University
School of Science

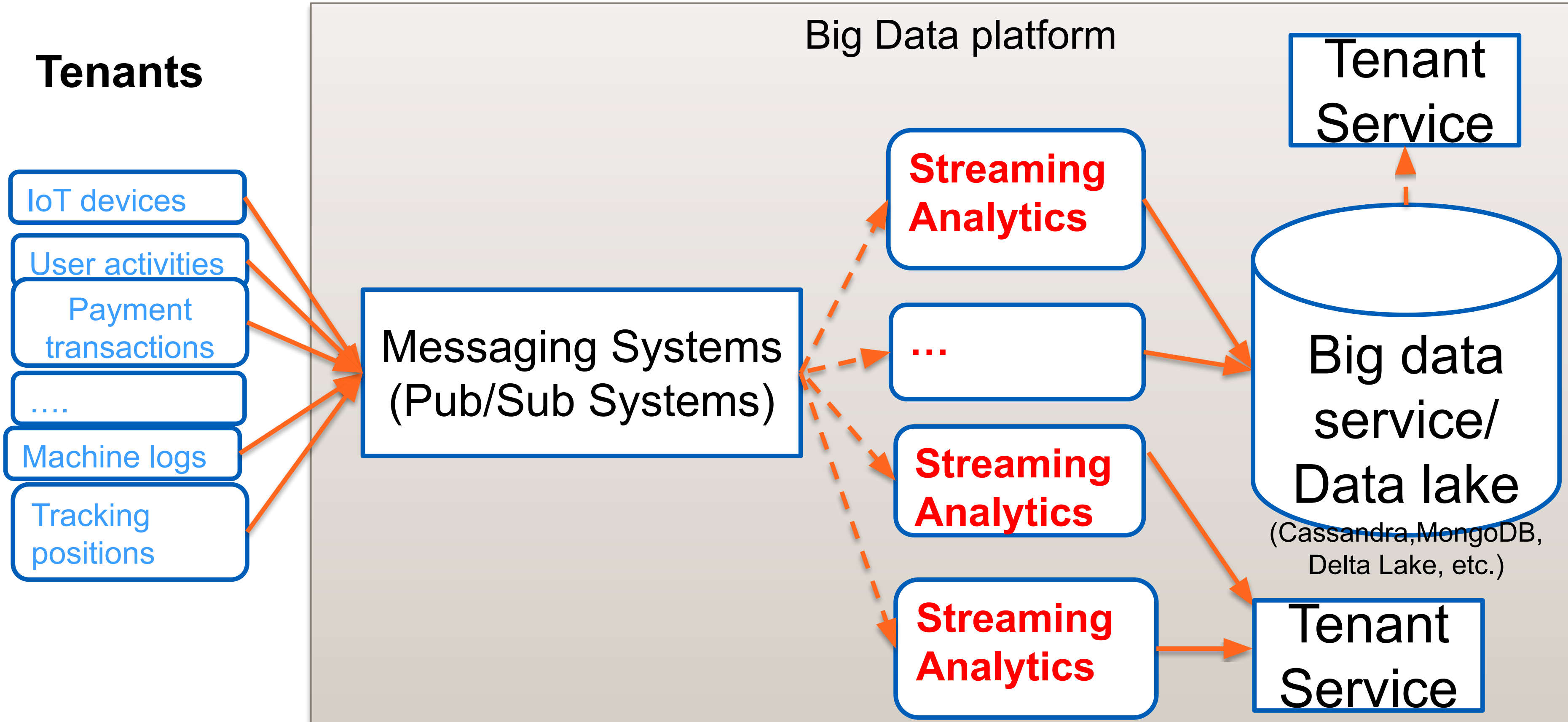
Stream analytics for data in motion

CS-E4640 Big Data Platforms, Spring 2025, Hong-Linh Truong
05/03/2025

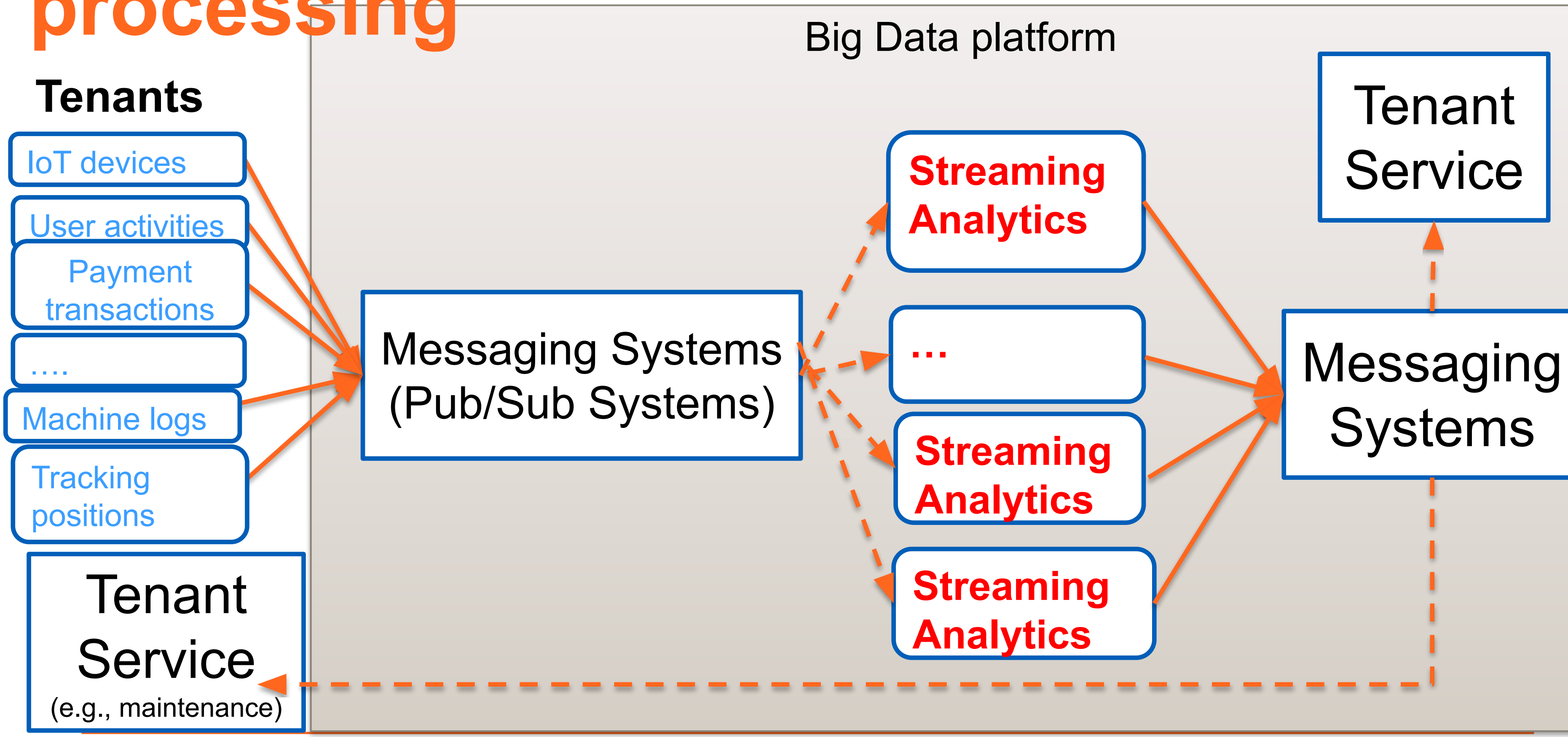
Stream processing in big data

- **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 processing time expected
  - not just *"take a record and store it into a database"*
- **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 $\Rightarrow$ distributed, parallel processing tasks
  - *stateful analytics:* processing needs state information across multiple data records and time

Near real-time streaming data processing



Near real-time streaming data processing



Example in the cloud – Azure stream analytics for stream processing and big data platforms

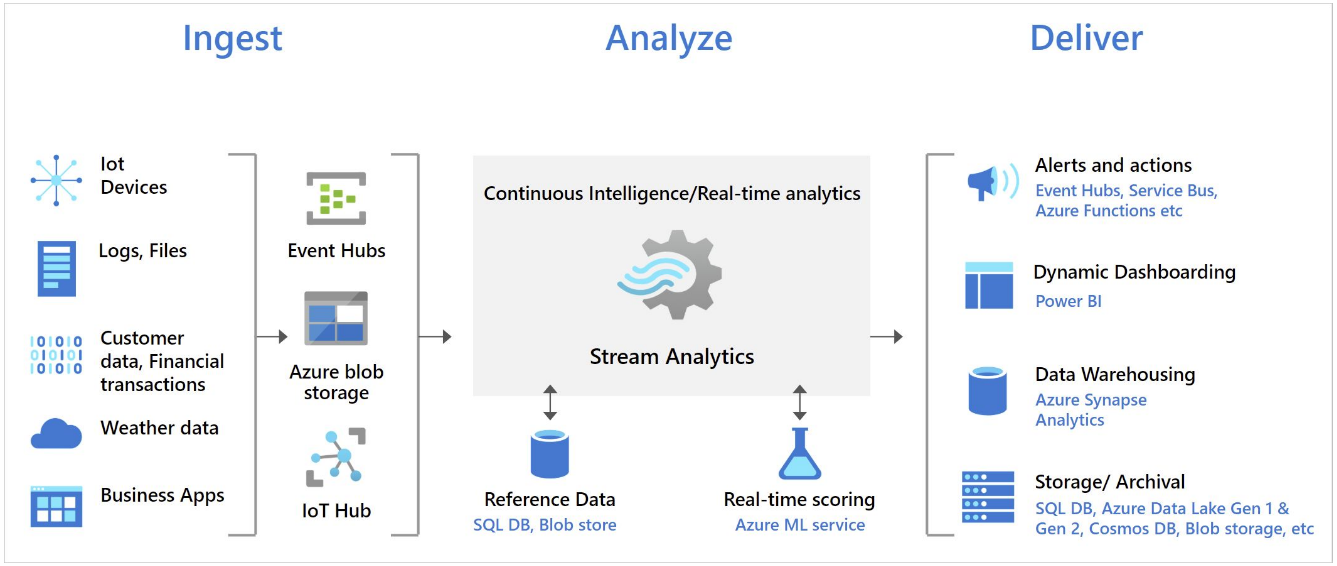


Figure source: https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction

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

From complex event processing (CEP) in the age of enterprise computing

Siddhi
Cloud Native Stream Processor

Esper CEP

APACHE STORM™

Apache Apex™

samza

Our practices focus on modern technologies like: Apache Flink, Apache Spark and Arroyo, which are used intensively in business systems and big cloud platforms

Stream processing and big data platforms

- **Stream processing services as a component of data platforms**
  - a big data technology for pre-processing, ingestion and high-level analytics, including near-real time machine learning
- **Stream processing services as data platforms**
  - a big data platform offers mainly stream processing services for streaming analytics
  - analytics on the fly as the first class feature
  - e.g., IoT analytics, e-commerce user activities, fraud detection, real time AI/ML

Stream Processing – key concepts

Common building blocks

- **The way to connect data to streams and obtain data records (messages) from the streams**
  - focusing very much on *connector concepts* and well-defined message structures (JSON, Avro, customized binary format, etc.)
  - connectors implement complex data handling mechanisms (low level session management, message retainment, delivery quality of service)
- **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 distributed engine to process analytics tasks**
  - handle complex task processing atop multiple compute nodes
- **The way to push the result to external components (sink databases, new streams, files)**

Data stream programming

Data stream: a sequence/flow of data units

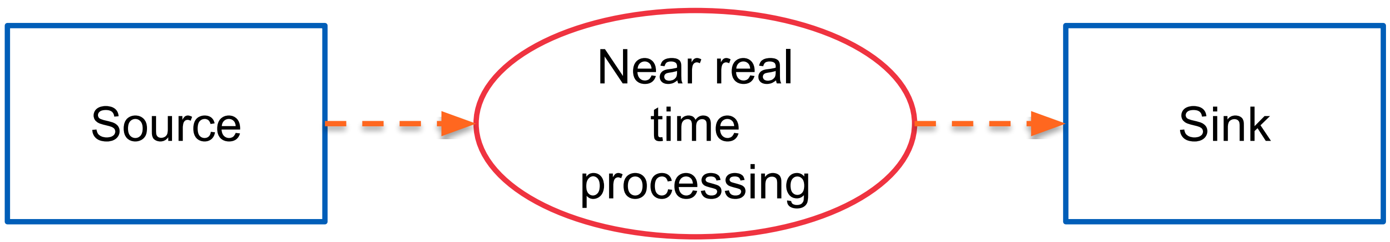
Data units are defined by applications: a data unit can be data described by a primitive data type or by a complex data type, a serializable object, etc.

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**



We focus on unbounded discrete messages of data

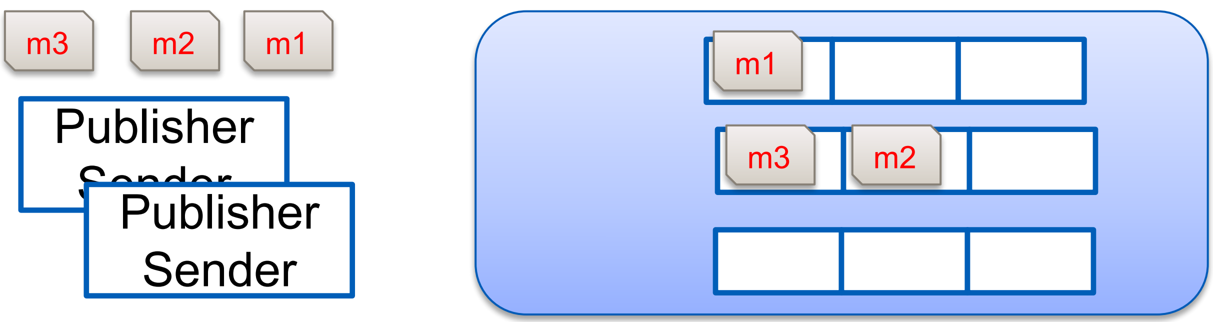
Message representations and streams

- **Data Sources**
  - via message brokers, databases, websocket, different IO adapters/connectors, etc.
- **Data Sinks**
  - messaging systems, databases, file storage/systems (S3, HDFS), etc.
- **Data representations**
  - text/CSV, JSON, Arvo format, etc.
  - serialization and deserialization (short name: SerDe) are required
  - data format validation
  - data schema registry for registered schemas

Publisher view: how messages are published

Messaging system

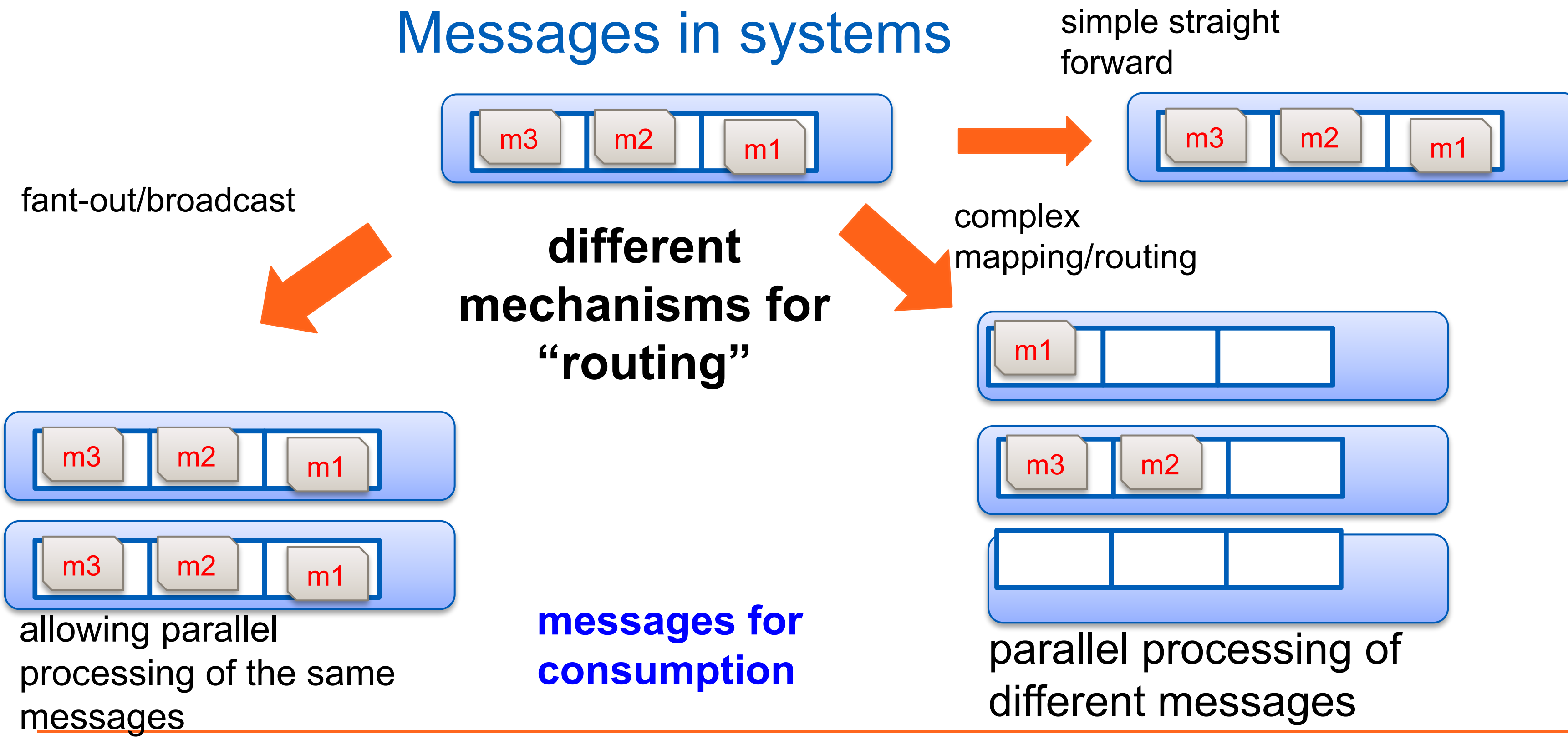
topic=queue; no partition



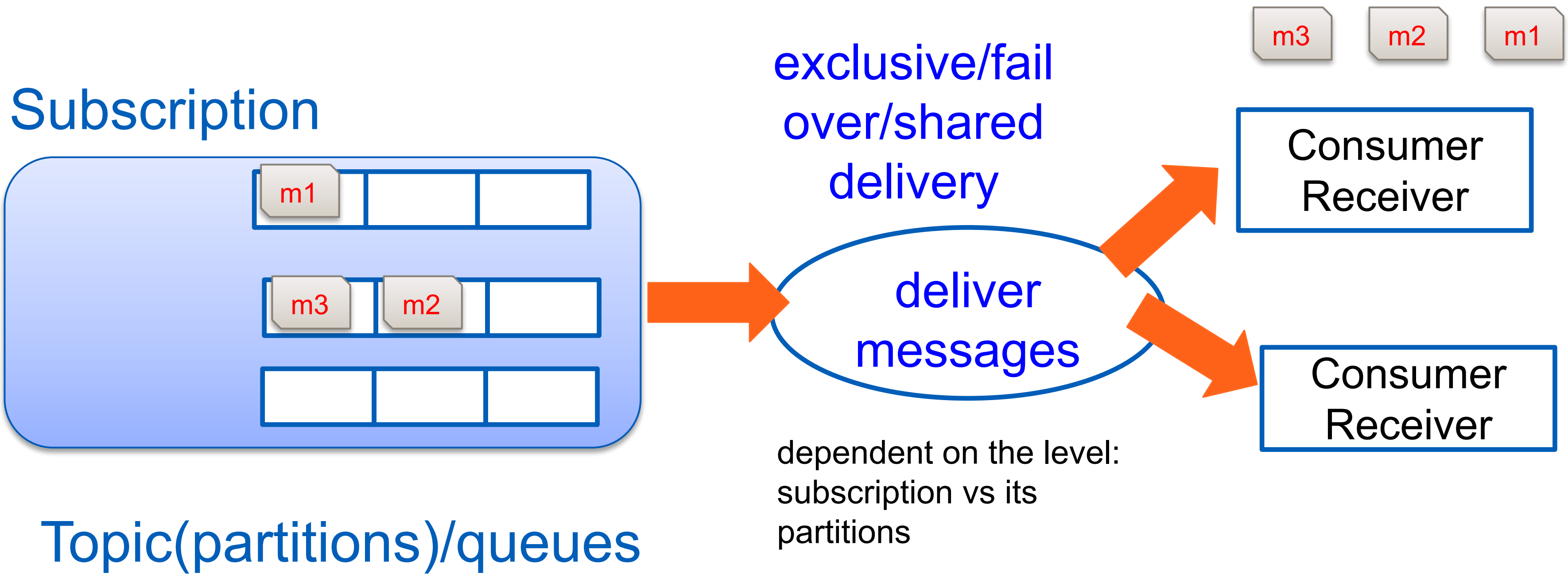
Topic and topic partitions

topic = n partitions = n queues

Handling messages for consumption (processing)



Consumer view in accessing messages: subscription and delivery

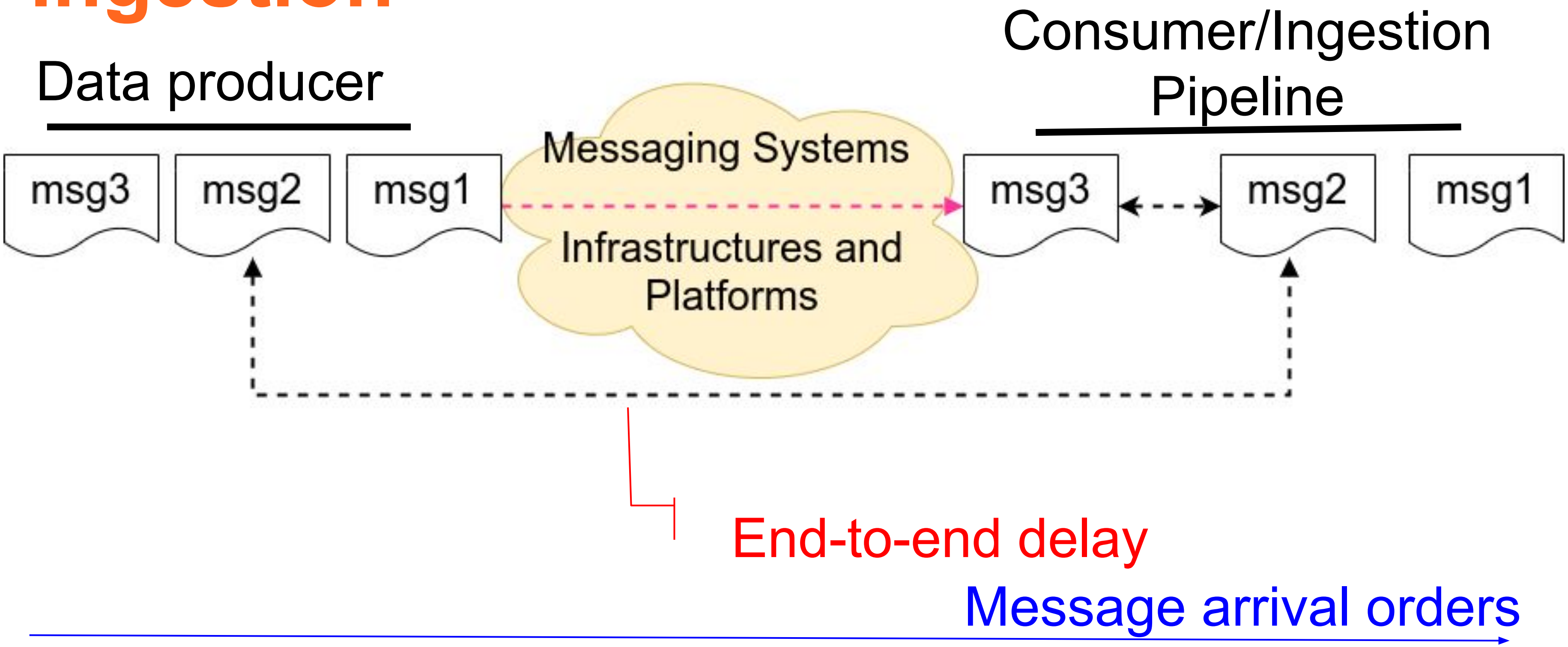


dependent on the level: subscription vs its partitions

Some key issues

- **Data order & delivery**
  - late data, out of order data
- **Times associated with messages and processing**
- **Data parallelism**
  - key-based data processing
- **Task parallelism**
  - stateful vs stateless processing

Key issues in streaming data ingestion

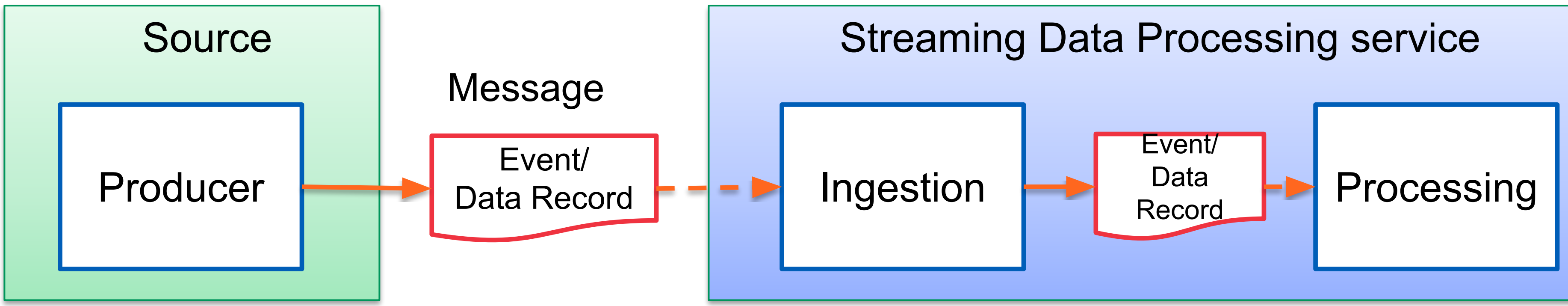


Without a timestamp associated to a message, do we know the delay or out of order?

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



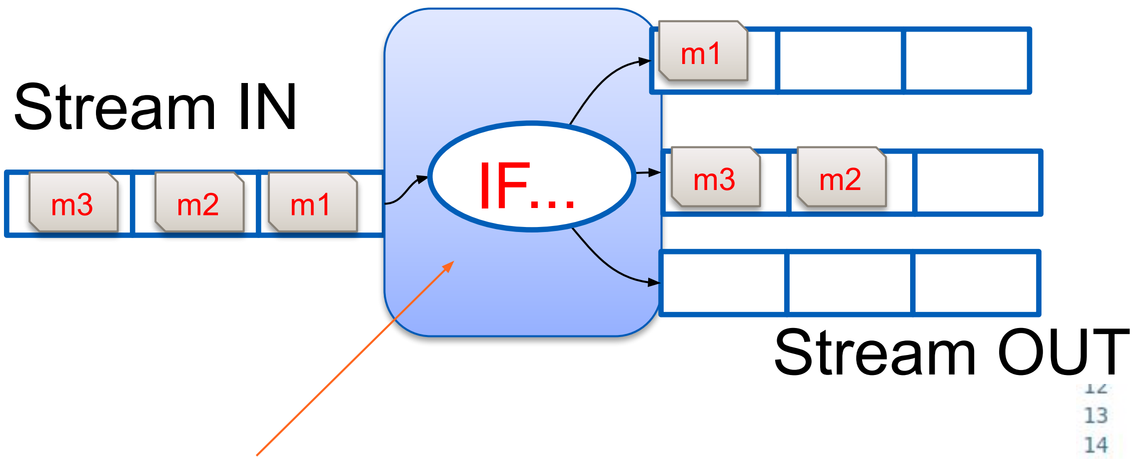
Event time

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

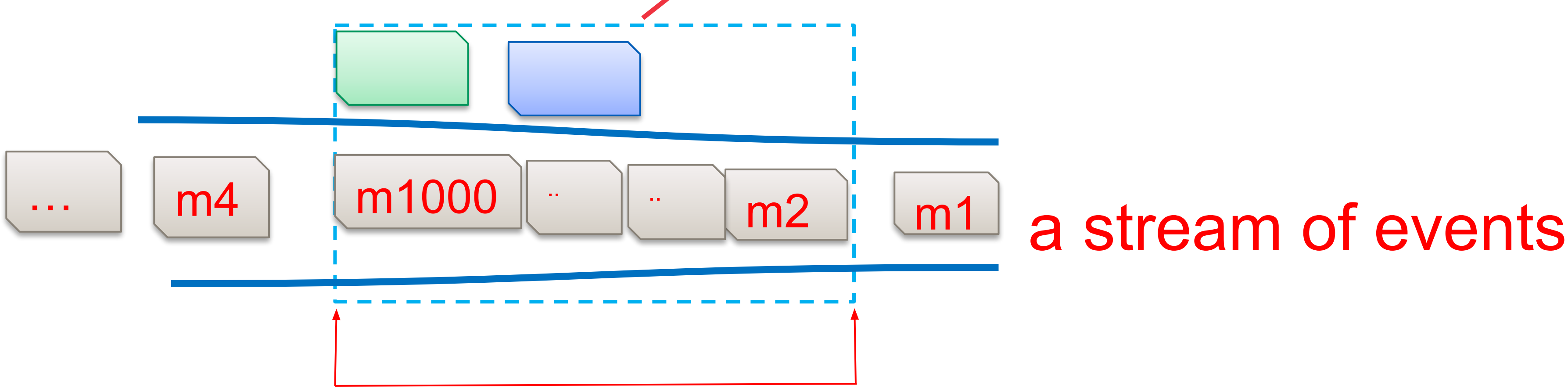
```java
*/
public class BTSAlarmEvent {
    public String station_id;
    public String datapoint_id;
    public String alarm_id;
    public Date event_time;
    public float value;
    public float valueThreshold;
    BTSAlarmEvent() {
```

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
- **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 ⇒ windows will be created according to the conditions to store message in corresponding windows

then we can

- Apply functions to messages 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!**
  - built-in and user-defined functions
- **Core for analytics and ML**
- **Examples**
  - individual threshold/alarm based alerting, atypical events monitoring
  - data rollup
  - 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 real-time):

- 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

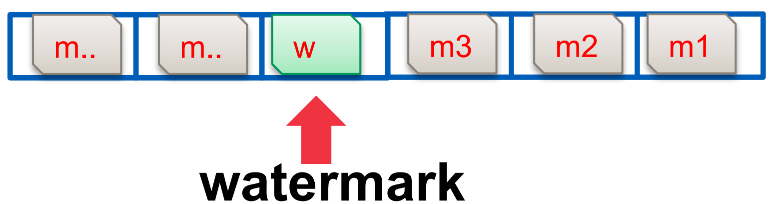
What if events/records come late into the windows?

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

correctness and completeness issues

Support lateness

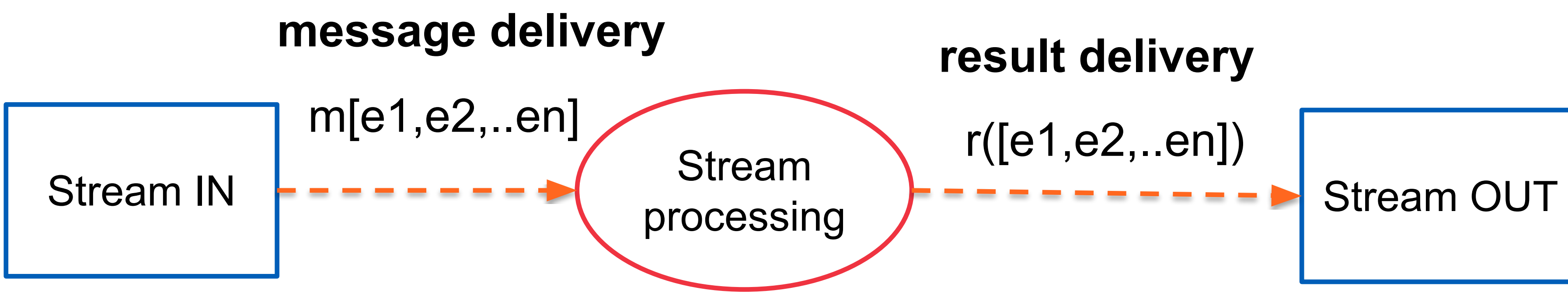
- **Identify timestamp of events/data records**
- **Identify watermark in streams**
  - a watermark is a timestamp
  - a watermark indicates that no events which are older that the watermark should be processed
  - enable the delay of processing functions to wait for late events
- **Using watermark to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

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)

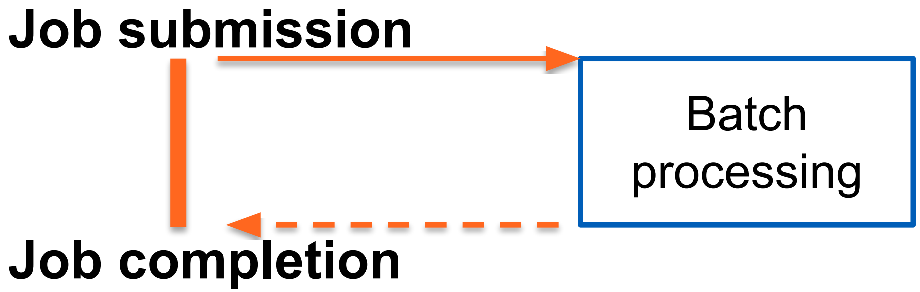
End-to-end exactly once

- **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**

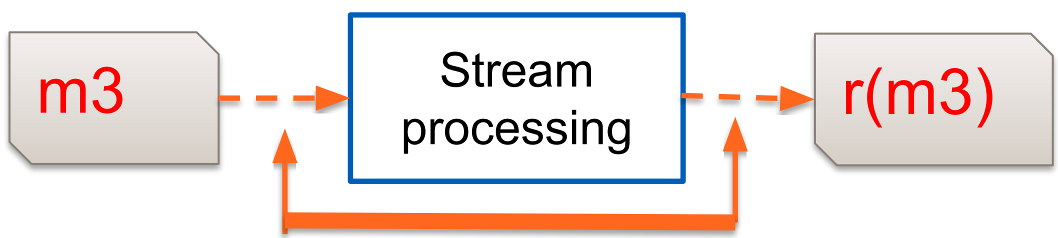
Further reading:
https://flink.apache.org/features/2018/03/01/end-to-end-exactly-once-apache-flink.html
https://www.confluent.io/blog/simplified-robust-exactly-one-semantics-in-kafka-2-5/
https://docs.microsoft.com/en-us/azure/hdinsight/spark/apache-spark-streaming-exactly-once

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? $\Rightarrow$ 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)
  - simple functions vs complex ones

Structure of streaming data processing programs (2) - examine a simple example

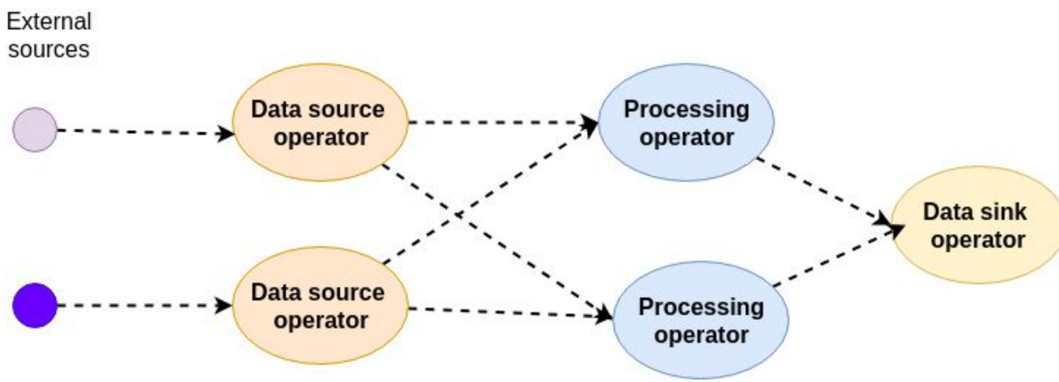
```python
WAIT AND PROCESS DATA
'''
while True:
    '''
    Receive the data from source
    '''
    msg = consumer.receive()
    '''
    when should we do this?
    consumer.acknowledge(msg)
    '''
    try:
        '''
        MAIN TRANSFORMATION, HERE IS WITH A FUNCTION
        '''
        ## assume 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 an external Pulsar consumer for data transformation

Structure of streaming data processing programs (3)

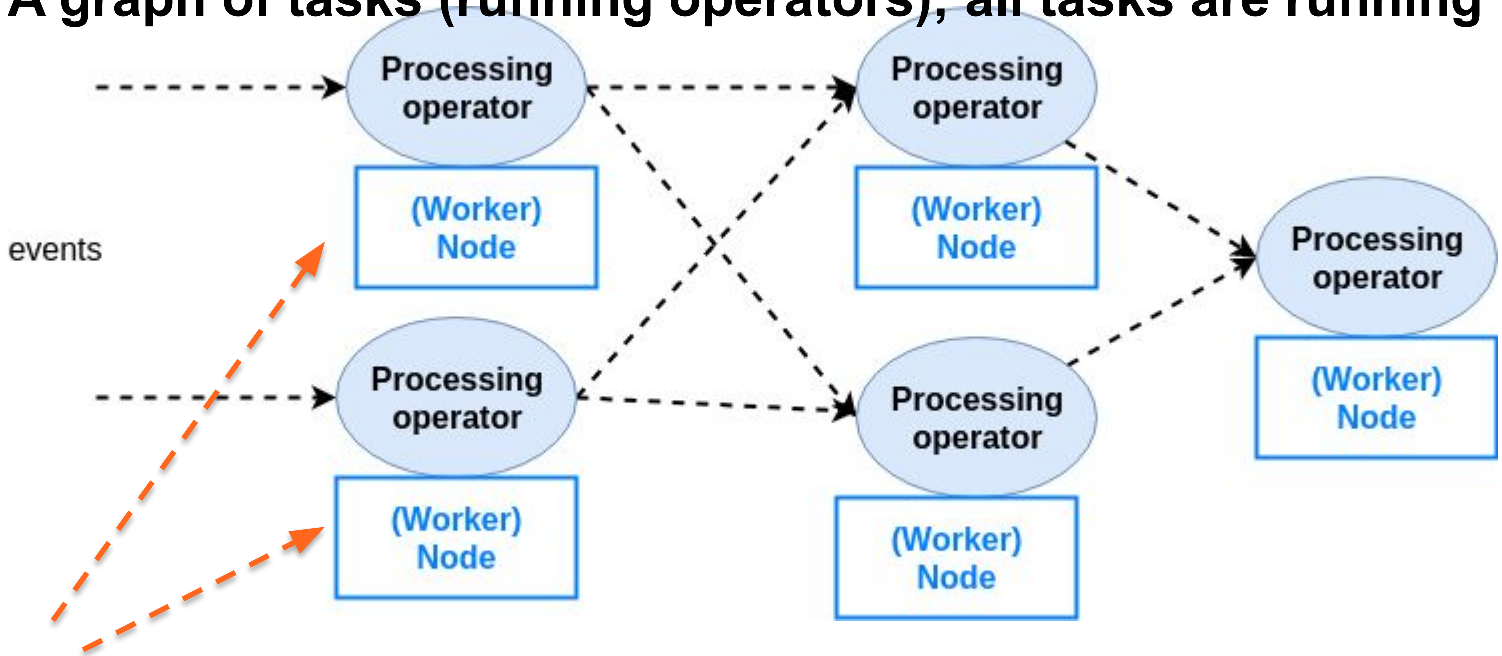



```java
//      parse the data, determine alert and return the alert in a json string
//      Apply function on data stream
        DataStream<String> alerts = btsdatastream
//              .flatMap(new BTSParser()
                .flatMap(new BTS_Trend_Parser()
                 /*
                    Another example is to have:
                    new FlatMapFunction<String, BTSAlarmEvent>() {
                        @Override
                        public void flatMap(String valueString, Collector<BTSAlarmEvent> out) {
                                String[] record = valueString.split(",");
                                ....
                            out.collect(...);
                          }
                 })
                */
                )
                //.setParallelism(5)    // uncomment this line to scale the Parser stream and set the valu
                .keyBy(new AlarmKeySelector()
                    /* another way is to have:
                    new KeySelector<BTSAlarmEvent, String>() {
                      public String getKey(BTSAlarmEvent btsalarm) { return btsalarm.station_id; }
                    }
                */
                )
                .window(SlidingProcessingTimeWindows.of(Time.seconds(60), Time.seconds(5)))
//              .window(SlidingEventTimeWindows.of(Time.minutes(5), Time.seconds(5))) // set the window si
//              .process(new MyProcessWindowFunction()).setParallelism(1);
                .process(new TrendDetection()).setParallelism(1);
        //.setParallelism(5);  // uncomment this line to scale the stream processing and set the value for
```

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

Distributed processing topology in a cluster

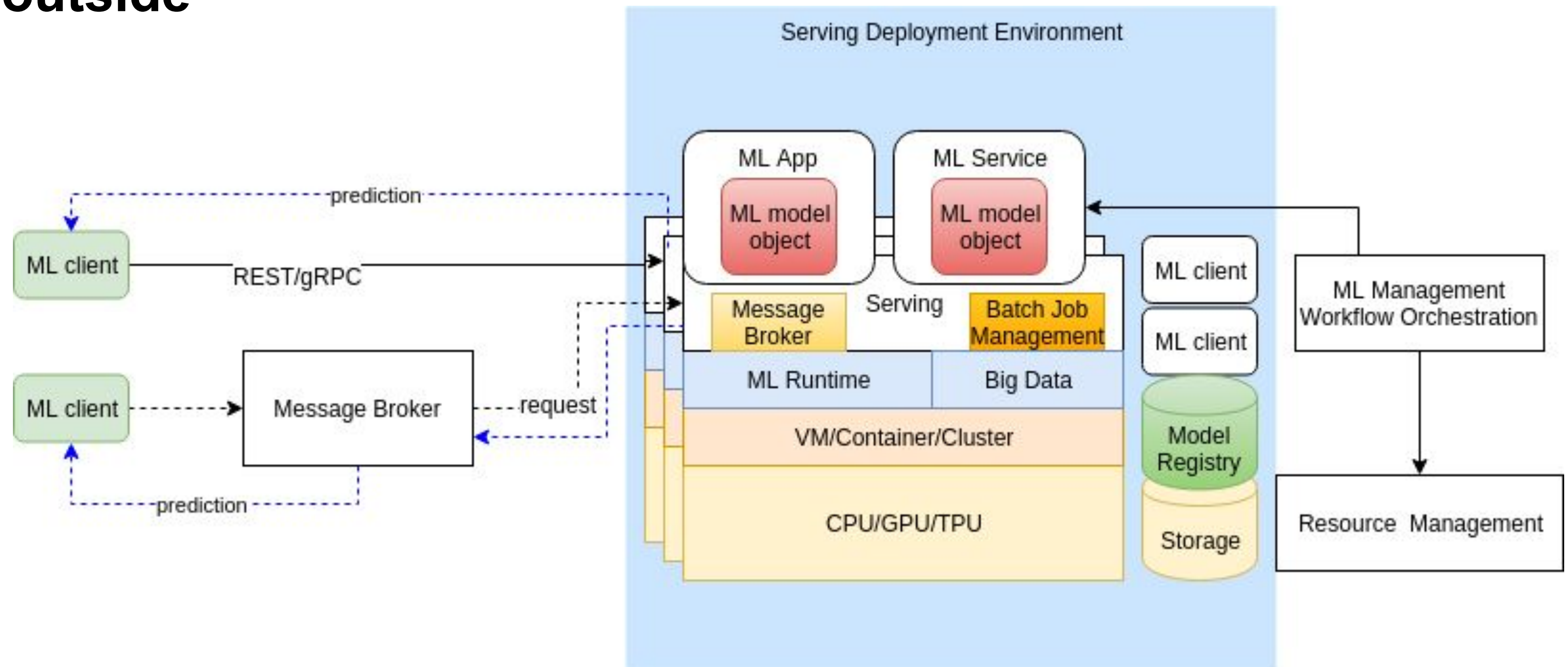
A graph of tasks (running operators); all tasks are running



Nodes of a cluster (VMs, containers, Kubernetes)

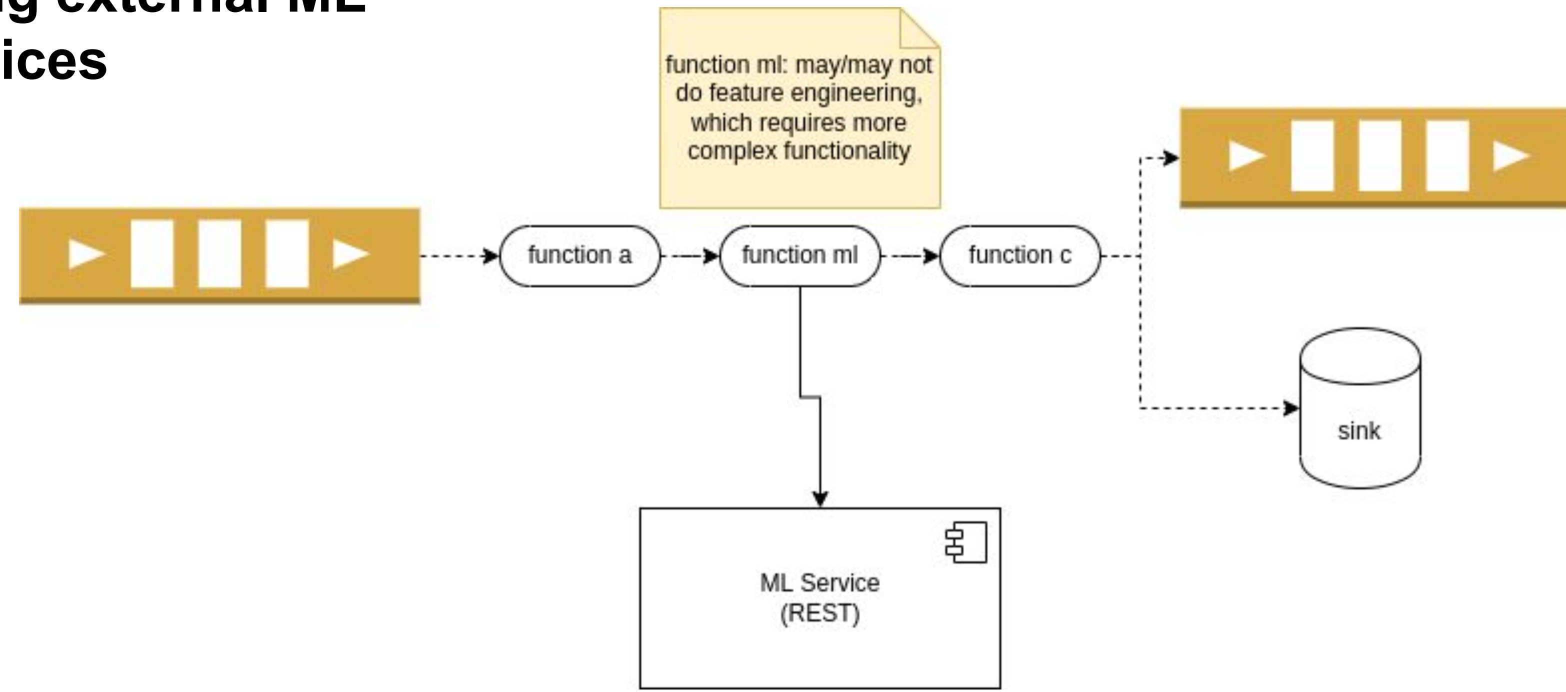
Integration with ML

From outside



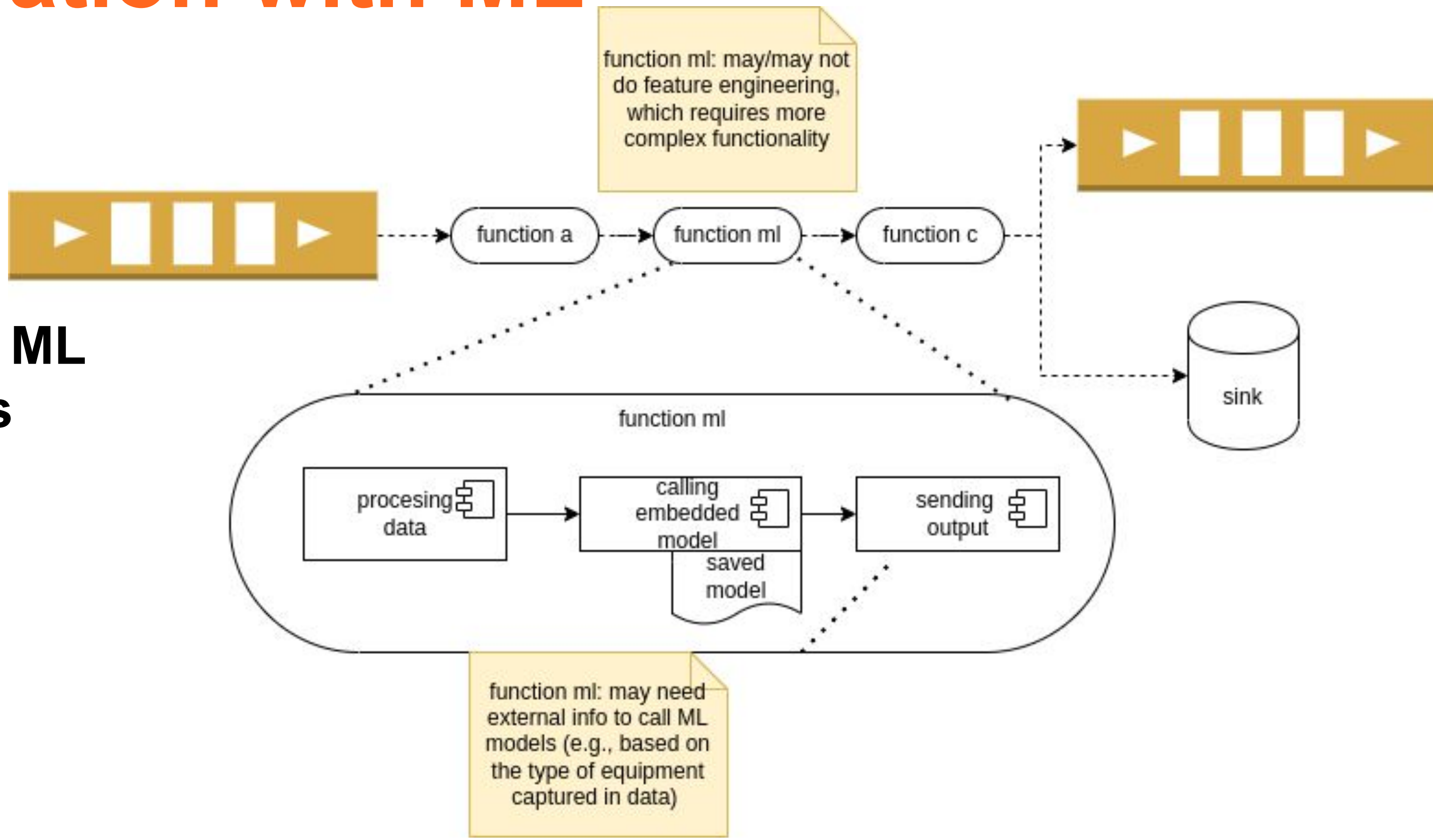
Integration with ML

Using external ML services

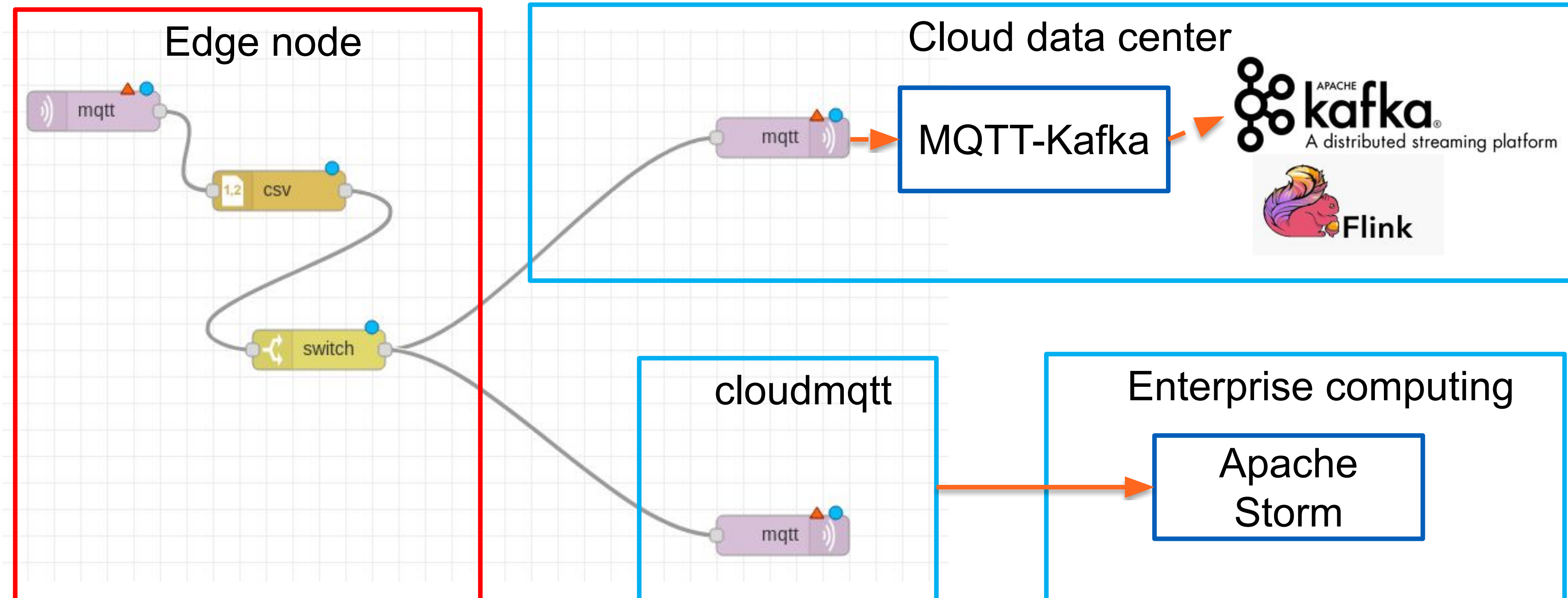


Integration with ML

Embedded ML capabilities



Distributed, composable processing topologies 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**

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

Summary

- **Focus:**
  - Practical programming with one of the stacks:
    - *Apache Flink Stream API (with different connectors)*
    - *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