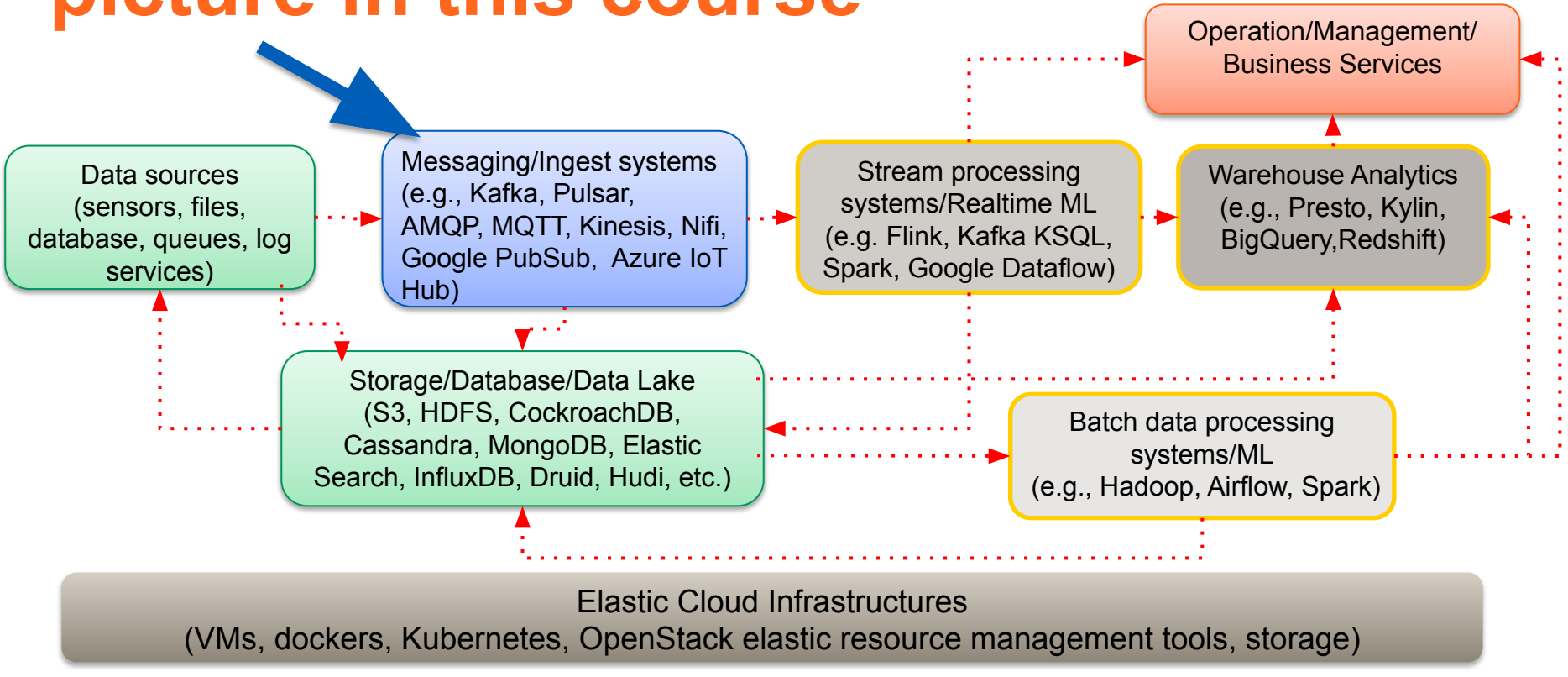
Aalto University
School of Science

Apache Kafka for Streaming Data Ingestion - the core

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

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



Abstraction of Data Streams

Data stream: a sequence/flow of *data units*

- continuously, unbounded or bounded data

Data units are defined by applications:

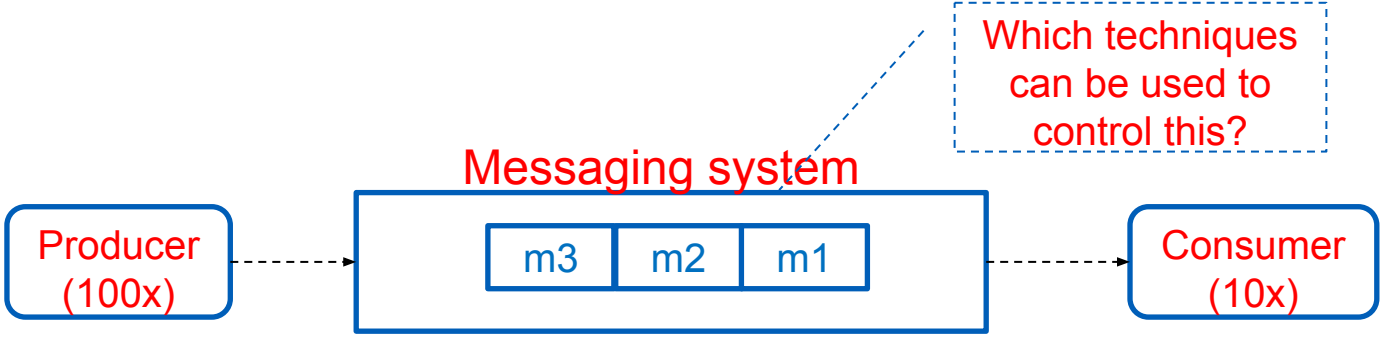
- a data unit can be data described by a primitive data type or by a complex data type
- “small” or “big” w.r.t. size
- events, messages (requests/responses, information), small documents, etc.

Usually we encapsulate a data unit in a record/message of data

- record in the application view != record in the system view

Some use cases - the diversity!

- **Producers generate a lot of near real-time events**
- **Producers and consumers have different processing speeds**
  - E.g. logging activities (fast producers but slow consumers)



- Diverse types of data to be produced and consumed
- Dealing with cases when consumers might be on and off (fault tolerance support)
- Asynchronous producing and consuming data

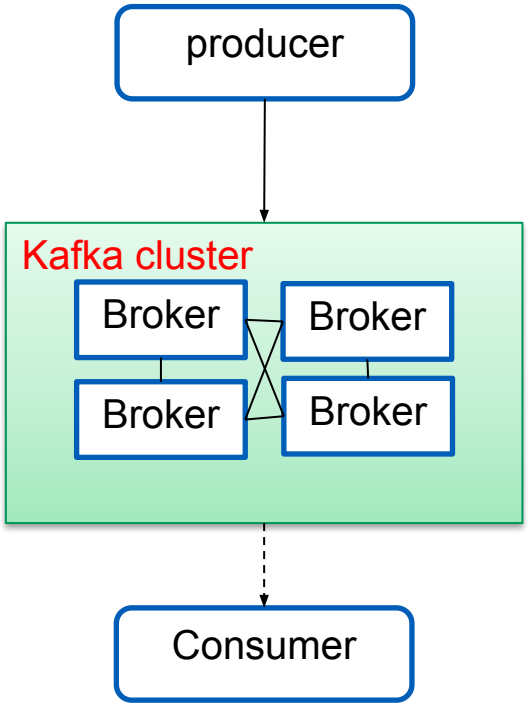
Examples of log-based/event-based messaging systems

- **Apache Kafka**
  - https://kafka.apache.org/
- **Apache Pulsar**
  - https://pulsar.apache.org/
- **RedPanda**
  - https://redpanda.com/

Apache Kafka

- **http://kafka.apache.org/**
  - originally from LinkedIn, not a protocol!
- **Some components are commercialized by Confluent**
  - https://www.confluent.io/
- **Widely used for big data use cases, including message processing in large-scale enterprise service platforms**
  - data messages (e.g., logs, records, historical events)
    - *It is our focus on big data platforms*
  - request/command messages (e.g., payment/database update)
  - event messages (e.g., notification of a payment due)

Kafka messaging design



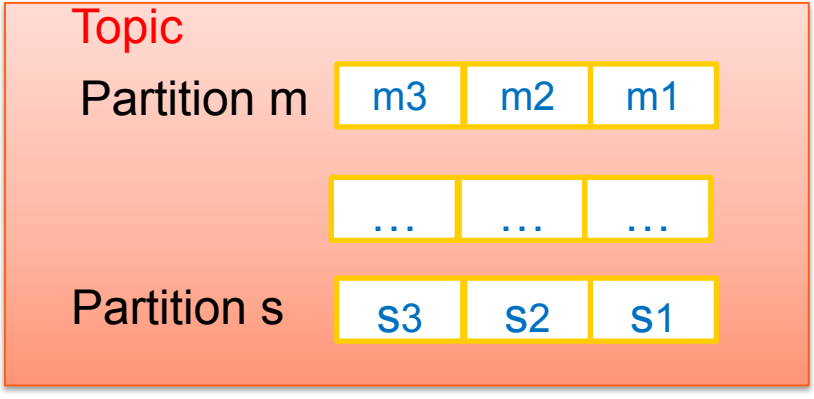
- **Use a cluster of brokers to deliver messages**
  - e.g., within single data center, on-premise
- **Durable messages, ordered delivery via partitions**
- **Online/offline consumers**
- **Using underlying file systems heavily for message storage and caching**

More than a message broker

- **In Apache Kafka: the basic data element is <Key,Value> tuple**
  - also timestamp and metadata
  - called “Kafka record”
- **Data streaming features**
  - for near-realtime transferring data
- **Stream processing**
  - streaming applications handle data from streams
  - read and write data back to Kafka message brokers
  - other important frameworks in the ecosystem:
    - Apache Flink and Apache Spark
- **High-level SQL-style: KSQL**
  - other possibilities: SQL-liked + Java in Apache Flink

In the context of big data: we examine Apache Kafka for transferring, ingesting and processing messages of (event) data

Kafka design



- **A topic consists of different partitions**
- **Partitions**
  - enable parallel processing ⇒ performance
  - fault-tolerance via replication
- **Durable messages, ordered delivery via partitions**
- **Messages with the same partition key will go to the same partition**

Messages, topics and partitions

- Ordered, immutable sequence of messages
- Messages are kept in a period (regardless of the consumed state)
- Support total order for messages within a partition
- Partitions are distributed among server

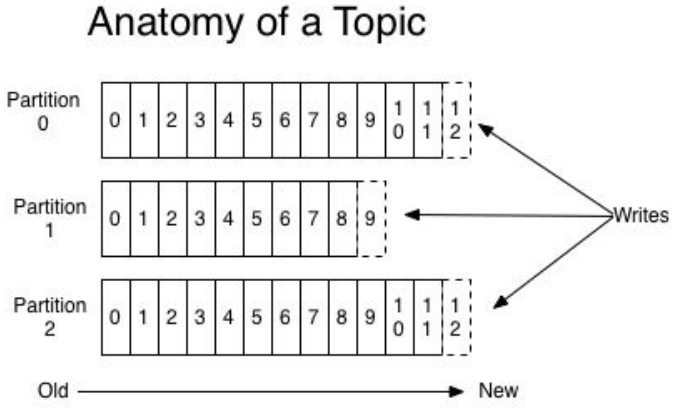


Figure source:
http://kafka.apache.org/documentation.html

Consumers

- Consumer pulls the data by sending requests to the broker
- The consumer keeps a single pointer indicating the position in a partition to keep track the offset of the next message being consumed
- Why?
  - ⇒ allow customers to design their speed
  - ⇒ support/optimize batching data
  - ⇒ easy to implement total order over message
  - ⇒ easy to implement reliable message/fault tolerance

Example of a producer

https://github.com/rdsea/bigdataplatforms/blob/master/tutorials/basickafka/code/simple_kafka_producer.py

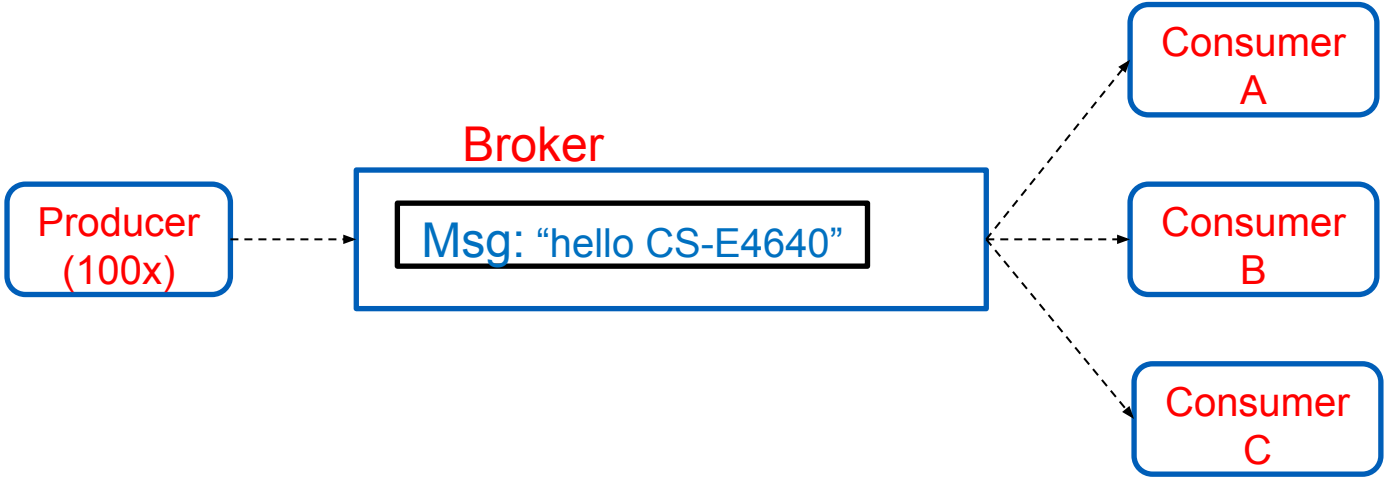
Example of a consumer

https://github.com/rdsea/bigdataplatforms/blob/master/tutorials/basickafka/code/simple_kafka_consumer.py

Message delivery

- **Message delivery guarantees are important for different use cases/requirements**
- **Some delivery models**
  - At most once
  - At least once
  - Exactly once

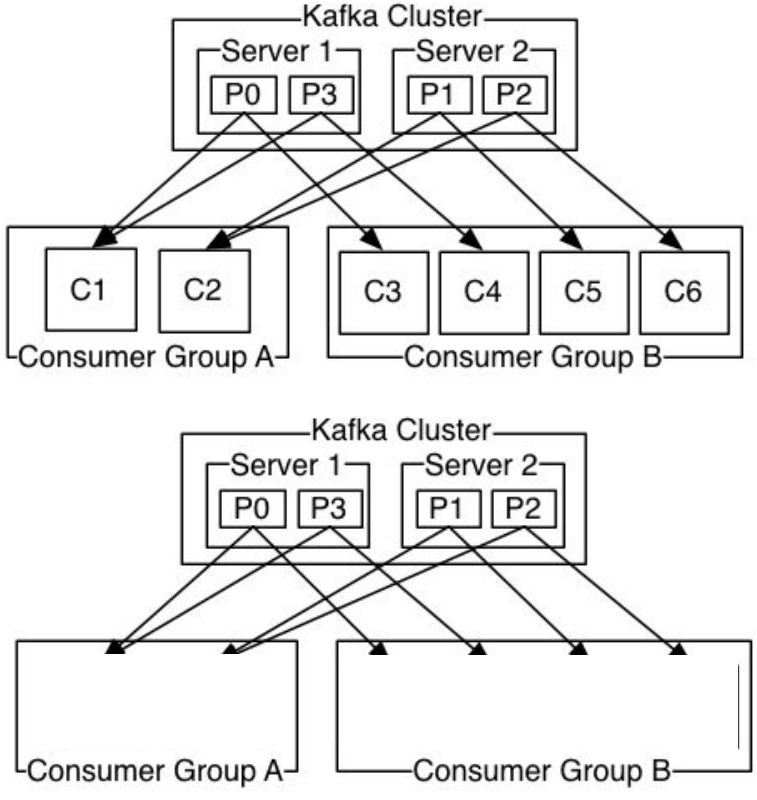
What does it mean exactly one?



- **Producer: idempotent delivery ⇒ no duplicate entry in the log**
- **Transaction-like semantics: either message to ALL partition topics or not at all**
- **Consumer behavior management**

Scalability and fault tolerance

- Topic can be replicated
- Partitions are distributed and replicated among broker servers
- Consumers are organized into groups
- Each message is delivered to only one consumer instance in a group
- One partition is assigned to one consumer



Figures source: http://kafka.apache.org/documentation.html#majordesignelements

Partitions and partition replication

- **Why partitions?**
  - support scalability
    - *enable arbitrary data types and sizes for a topic*
    - *enable parallelism in producing and consuming data*

- **But partitions are replicated, why?**
  - for fault tolerance

Partition replication

Replication model: the leader-follower (primary-secondary) model!

The leader handles all read and write requests

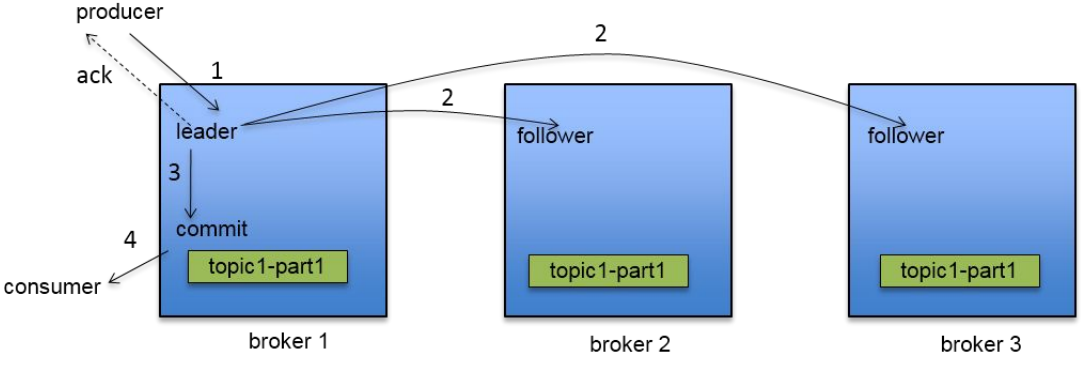


Figure source: http://de.slideshare.net/junrao/kafka-replication-apachecon2013

Consumer group

- **Consumer group: a set of consumers**
  - used to support scalability and fault tolerance
  - allows multiple consumers to read a topic
- **In one group: each partition is consumed by only consumer instance**
  - combine „queuing“ model and „publish/subscribe“ model
- **Enable different applications receive data from the same topic.**
  - different consumers in different groups can retrieve the same data

Key questions/thoughts

- Why do we need partitions per topic?
  - ⇒ arbitrary data handling, ordering guarantees, load balancing
- How to deal with high volume of near real-time messages for online and offline consumers?
  - ⇒ partition, cluster, message storage, batch retrieval, etc.
- Queuing or publish-subscribe model?
  - ⇒ check how Kafka delivers messages to consumer instances/groups

Kafka vs RabbitMQ

Figures source: Philippe Dobbelaere and Kyumars Sheykh Esmaili. 2017. Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper. In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems (DEBS '17). ACM, New York, NY, USA, 227-238. DOI: https://doi.org/10.1145/3093742.3093908

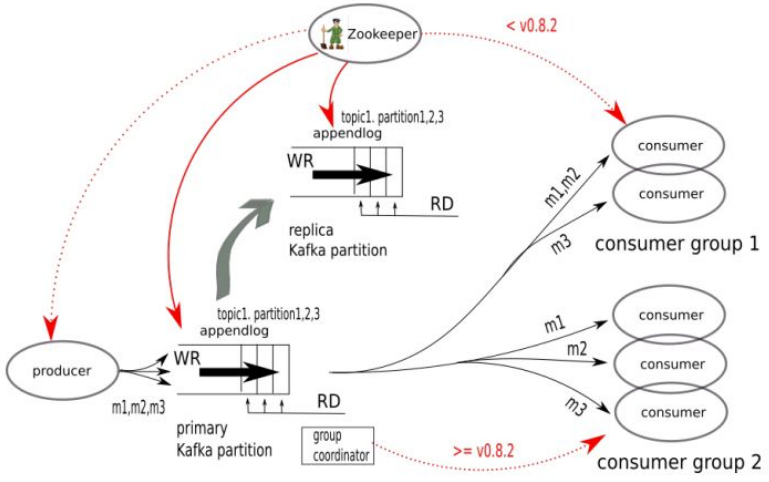


Figure 1: Kafka Architecture

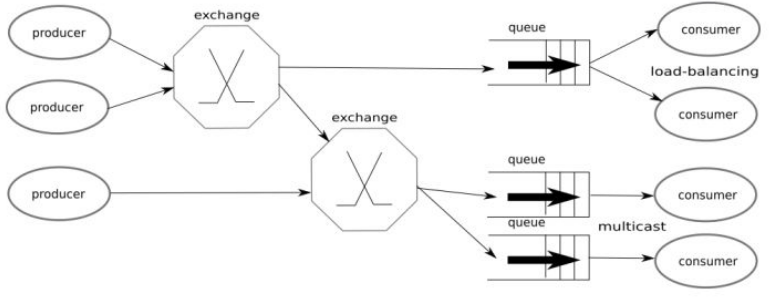


Figure 2: RabbitMQ (AMQP) Architecture

Hands-on

- **Understanding the message broker systems and message delivery are key for stream processing**
- **Check our tutorials:**
  - https://github.com/rdsea/bigdataplatforms/tree/master/tutorials/basickafka
  - https://github.com/rdsea/bigdataplatforms/tree/master/tutorials/cloud-data-pipeline