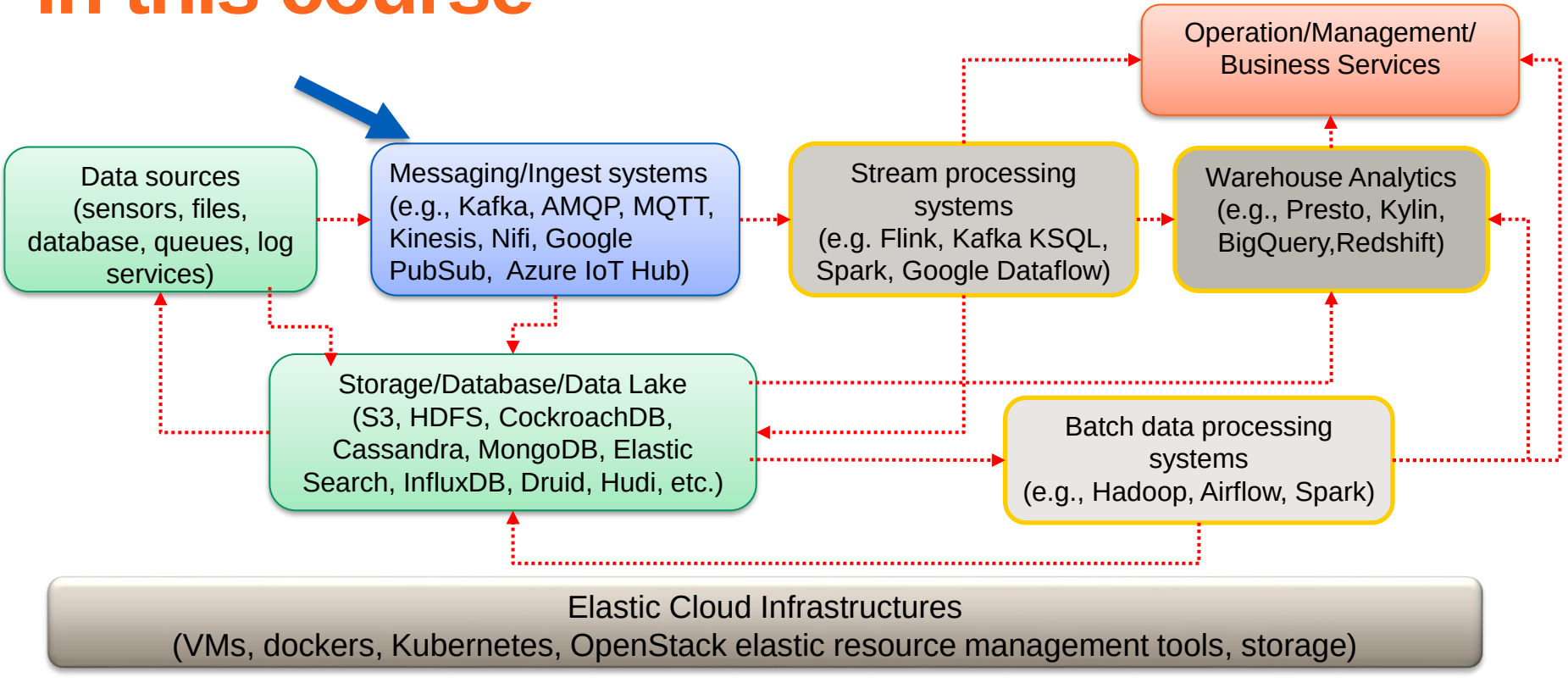
Aalto University
School of Science

Streaming Data Ingestion with Apache Kafka

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

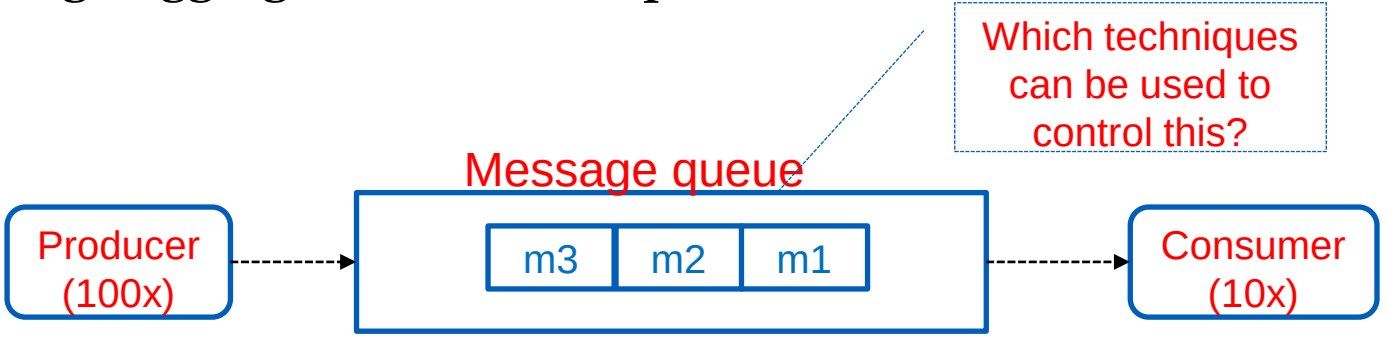
Data stream: a sequence/flow of *data units*

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.

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

Some use cases

- **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)



- Rich and diverse types of events
- Dealing with cases when consumers might be on and off (fault tolerance support)

Key log-based messaging systems

- **Apache Kafka**
  - https://kafka.apache.org/
- **Apache Pulsar**
  - https://pulsar.apache.org/
- **LogDevice (Facebook)**
  - https://logdevice.io/

Apache Kafka

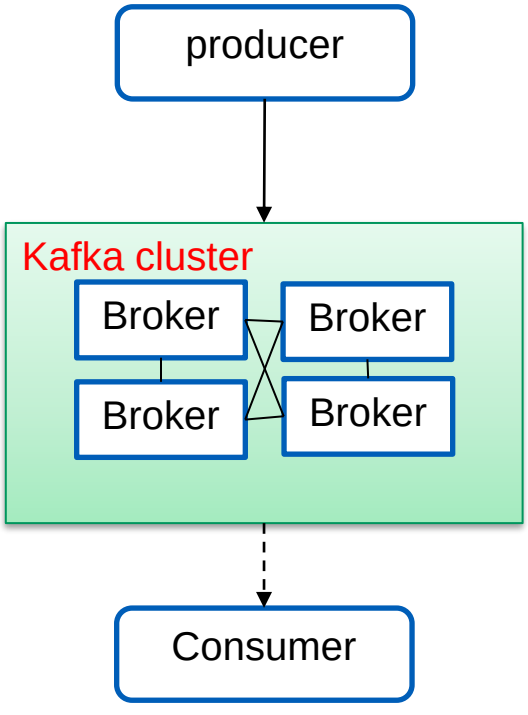
- **http://kafka.apache.org/**
  - originally from LinkedIn, not a protocol!
- **Some components are commercialized by Confluent**
  - https://www.confluent.io/
- **Widely used for big data use cases, including message processing in large-scale enterprise service platforms**
  - data messages (e.g., 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)

More than a message broker

- **In Apache Kafka: the basic data element is <Key,Value> tuple**
- **Messaging features**
  - for transferring messages
    - *Other frameworks in the ecosystem: RabbitMQ, Mostquitto*
- **Streaming processing**
  - streaming applications handle data from streams
  - read and write data back to Kafka messaging brokers
  - other frameworks in the ecosystem: Apache Flink and Apache Spark
- **High-level SQL-style: KSQL**
  - other possibilities: SQL-liked + Java in Apache Flink

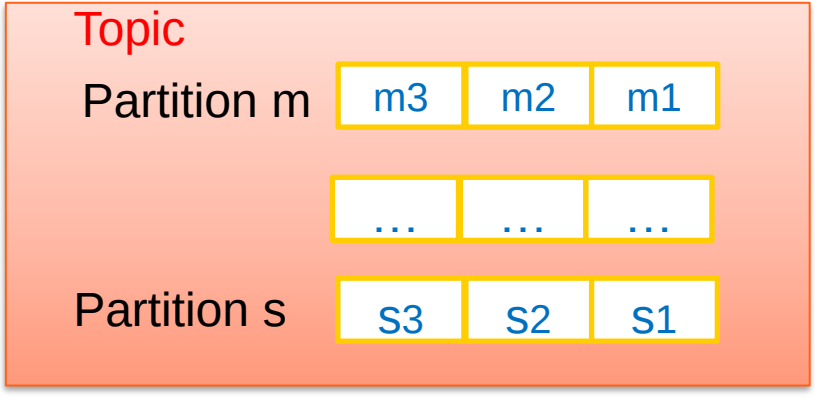
In the context of big data: we examine Apache Kafka for transferring, ingesting and processing messages of data

Kafka messaging design



- **Use a cluster of brokers to deliver messages**
  - usually within single data center, with high-speed networks, for a single tenant
- **Durable messages, ordered delivery via partitions**
- **Online/offline consumers**
- **Using filesystem heavily for message storage and caching**

Kafka design



- **A topic consists of different partitions**
- **Partitions**
  - enable parallel processing → performance
  - fault-tolerance via replication
- **Durable messages, ordered delivery via partitions**

Messages, topics and partitions

- Ordered, immutable sequence of messages
- Messages are kept in a period (regardless of consumers or not)
- Support total order for messages within a partition
- Partitions are distributed among server

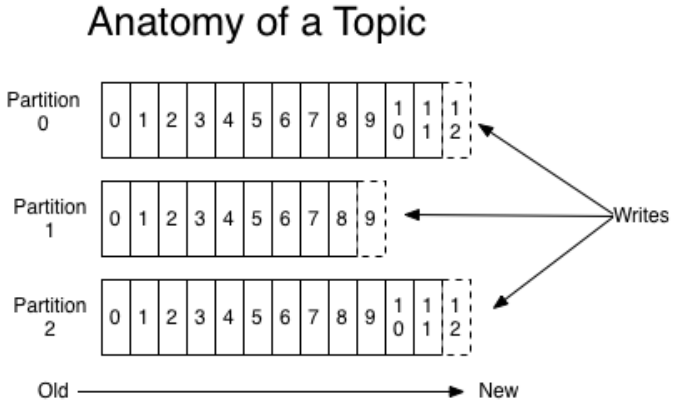


Figure source: http://kafka.apache.org/documentation.html

Consumers

- **Consumer pulls the data**
- **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://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/blob/master/tutorials/basickafka/code/simple_kafka_producer.py

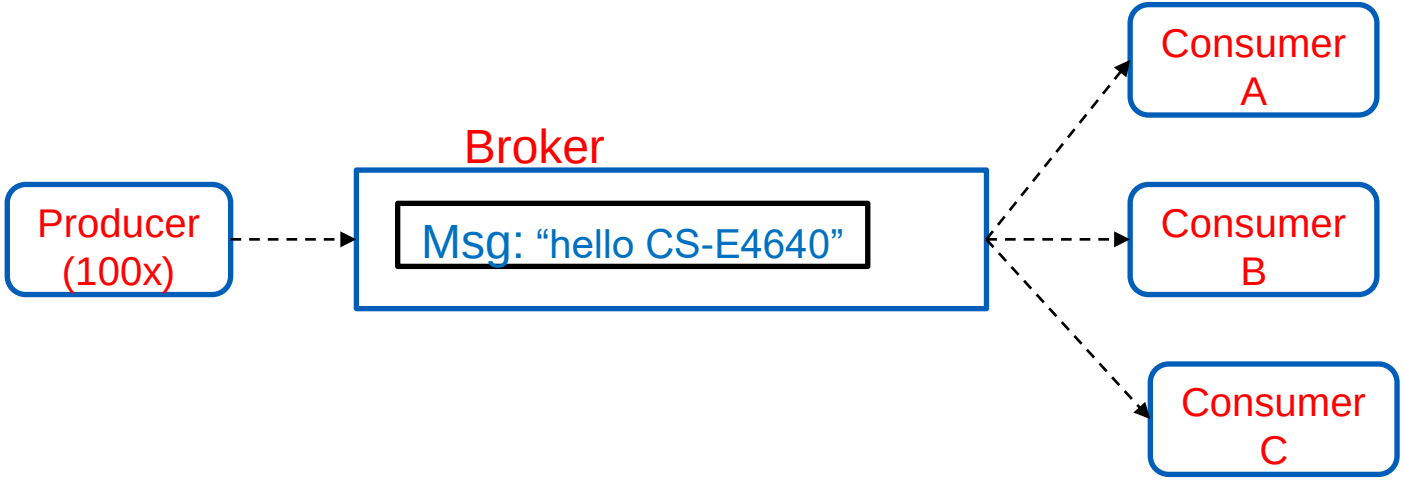
Example of a consumer

https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/blob/master/tutorials/basickafka/code/simple_kafka_consumer.py

Message delivery

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

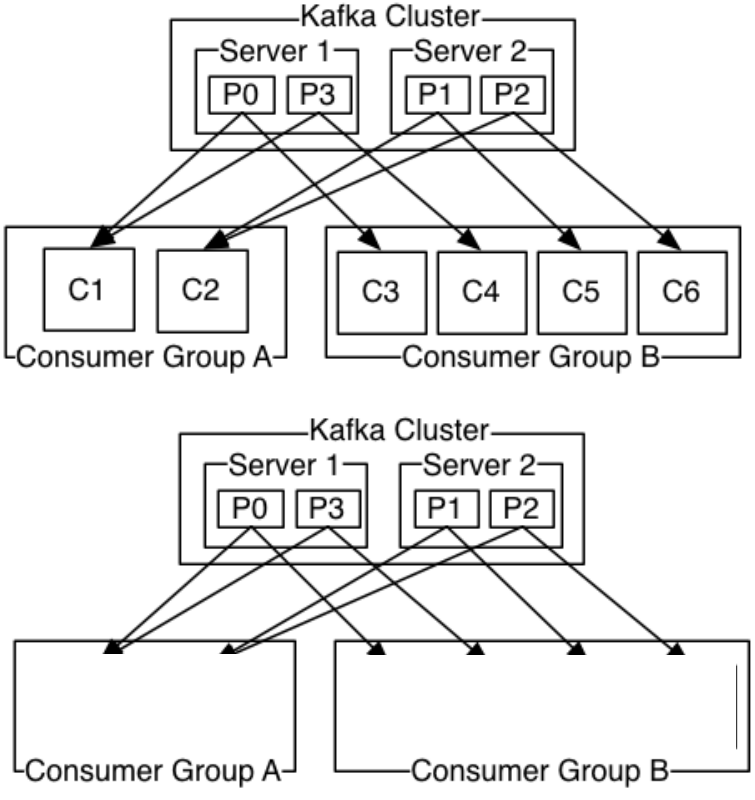
What does it mean exactly one?



- **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

- Partitions are distributed and replicated among broker servers
- Consumers are organized into groups
- Each message is delivered to a 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

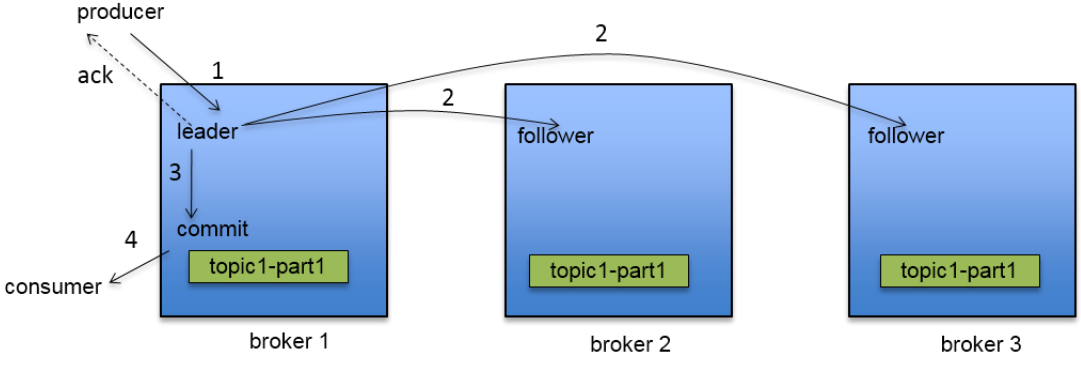


Figure source: http://de.slideshare.net/junrao/kafka-replication-apachecon2013

Consumer group

- **Consumer group: a set of consumers**
  - is 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

Group rebalancing

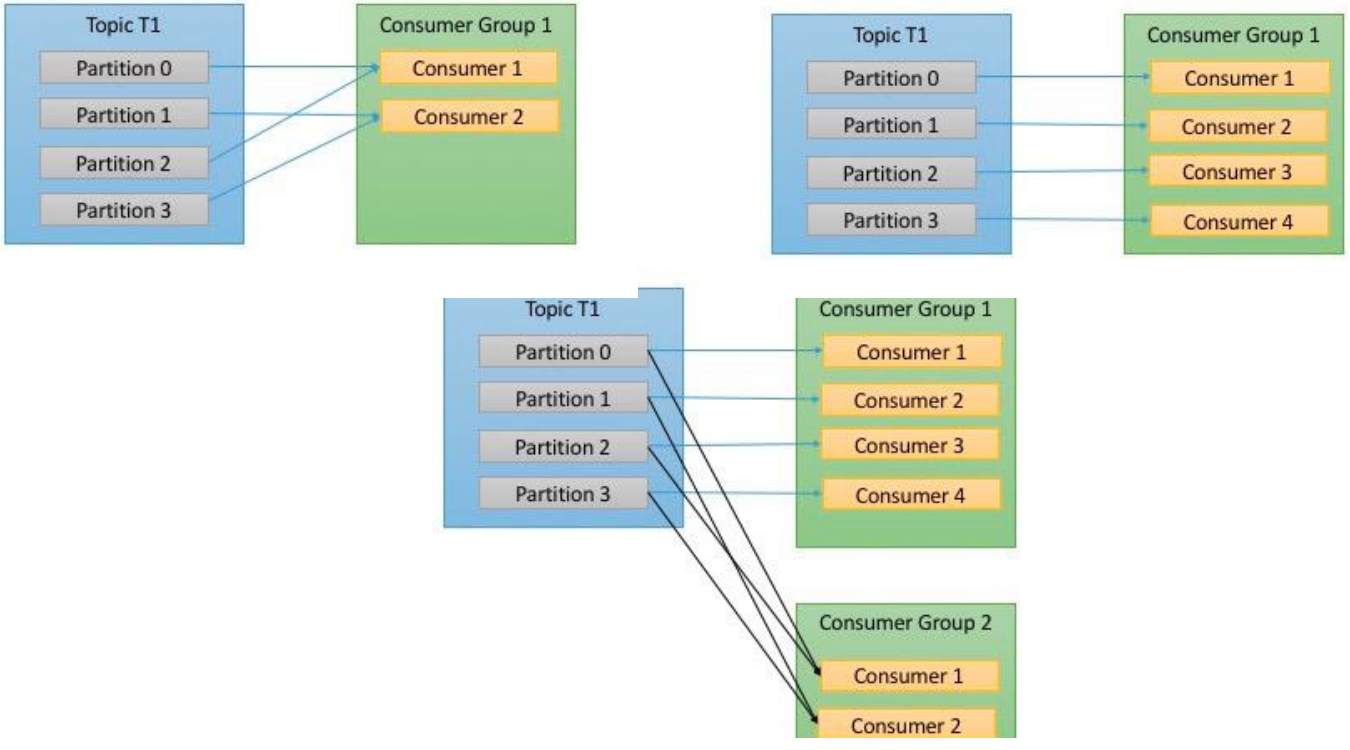


Figure source: https://www.safaribooksonline.com/library/view/kafka-the-definitive/9781491936153/ch04.html

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

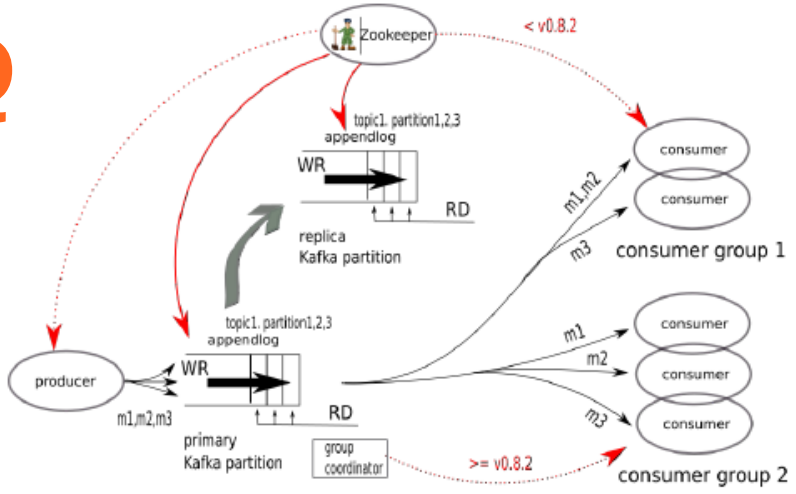


Figure 1: Kafka Architecture

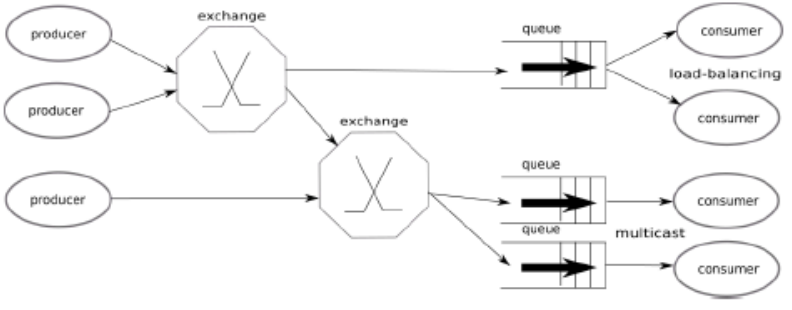


Figure 2: RabbitMQ (AMQP) Architecture

Figure 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

Hands-on

- **Understanding the message broker systems and message delivery are key for streaming processing**
- **Check our tutorial:**
  - https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/basickafka
  - https://version.aalto.fi/gitlab/bigdataplatforms/cs-e4640/-/tree/master/tutorials/cloud-data-pipeline