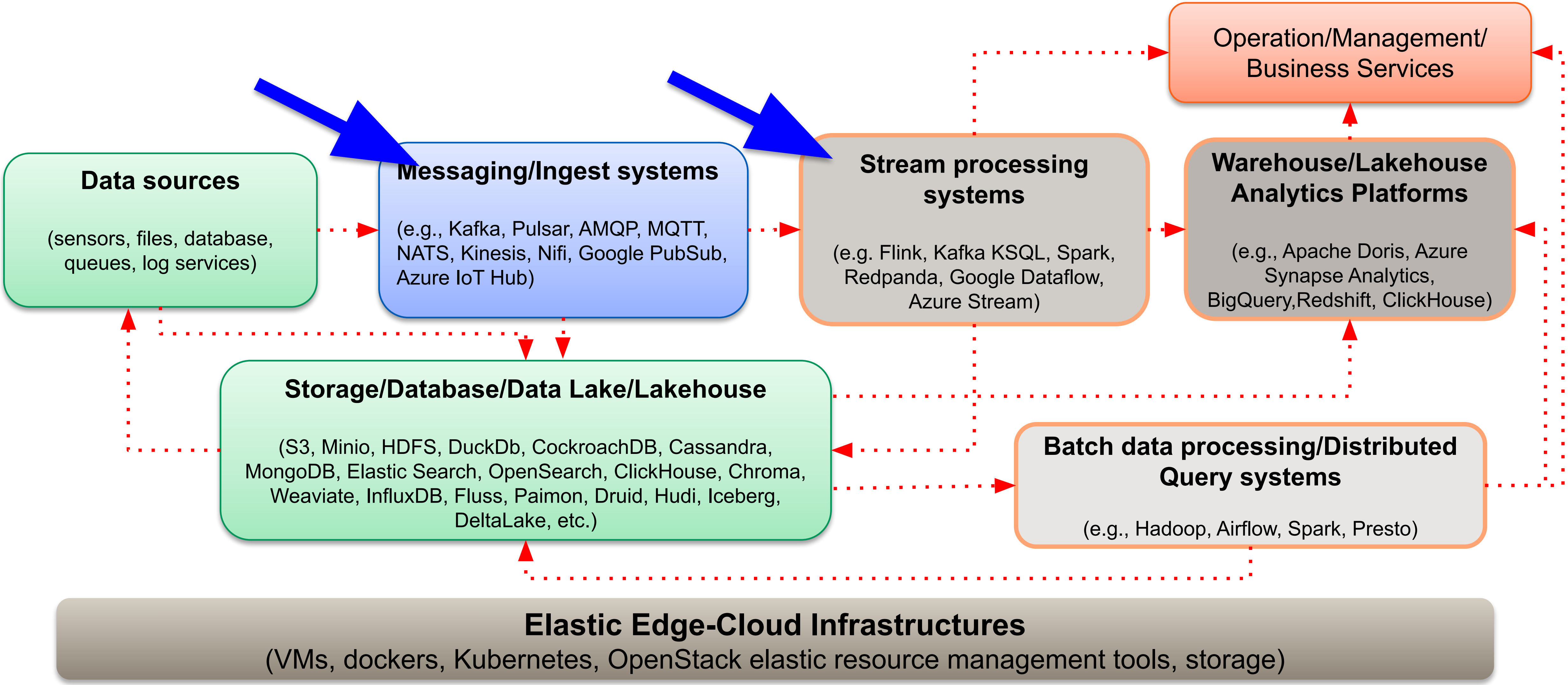
Aalto University
School of Science

Apache Kafka for Streaming Data Ingestion - the Core

Hong-Linh Truong
Department of Computer Science
linh.truong@aalto.fi

23.1.2026

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



Data sources

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

Messaging/Ingest systems

(e.g., Kafka, Pulsar, AMQP, MQTT, NATS, Kinesis, Nifi, Google PubSub, Azure IoT Hub)

Stream processing systems

(e.g. Flink, Kafka KSQL, Spark, Redpanda, Google Dataflow, Azure Stream)

Warehouse/Lakehouse Analytics Platforms

(e.g., Apache Doris, Azure Synapse Analytics, BigQuery,Redshift, ClickHouse)

Operation/Management/ Business Services

Storage/Database/Data Lake/Lakehouse

(S3, Minio, HDFS, DuckDb, CockroachDB, Cassandra, MongoDB, Elastic Search, OpenSearch, ClickHouse, Chroma, Weaviate, InfluxDB, Fluss, Paimon, Druid, Hudi, Iceberg, DeltaLake, etc.)

Batch data processing/Distributed Query systems

(e.g., Hadoop, Airflow, Spark, Presto)

Elastic Edge-Cloud Infrastructures

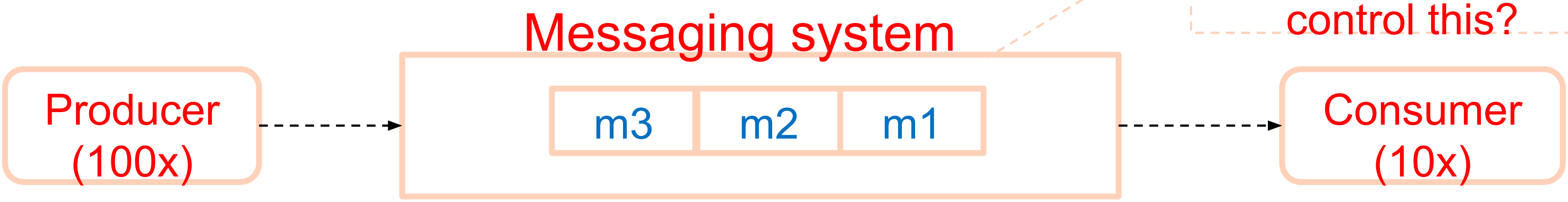
(VMs, dockers, Kubernetes, OpenStack elastic resource management tools, storage)

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!

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



Which techniques can be used to control this?

- 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

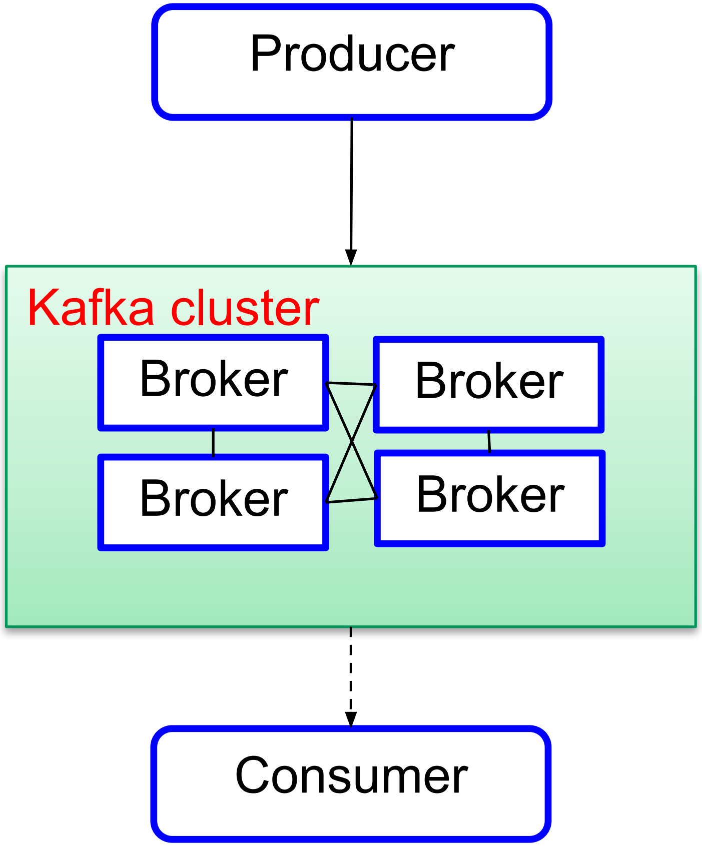
Examples of log-based/event-based messaging systems

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

Apache Kafka

- https://kafka.apache.org/
  - originally from LinkedIn, not a protocol!
- Some components are commercialized by Confluent (bought by IBM)
  - 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)
    - in the focus of big data platforms
  - request/command messages (e.g., payment/database update)
  - event messages (e.g., notification of a payment due)

Kafka messaging design



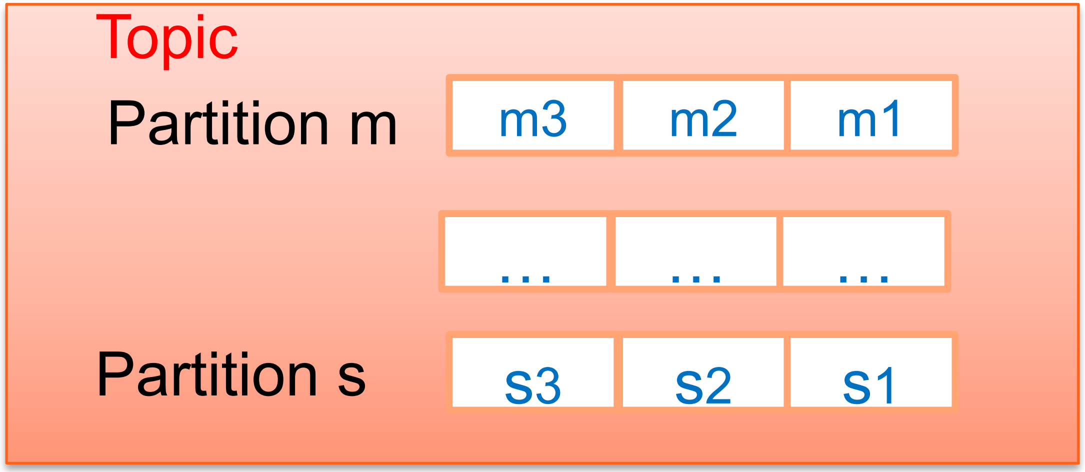
- Use a cluster of brokers to deliver messages
  - e.g., within single data center or on-premise
- Durable messages, ordered delivery via partitions
- Online/offline consumers
- Using underlying file systems heavily for message storage and caching

More than a message broker

- In Apache Kafka: the basic data element is <Key,Value> tuple
  - also with timestamp and metadata
  - called “Kafka record”
- Data streaming features
  - for near real-time 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/ksqlDB
  - 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

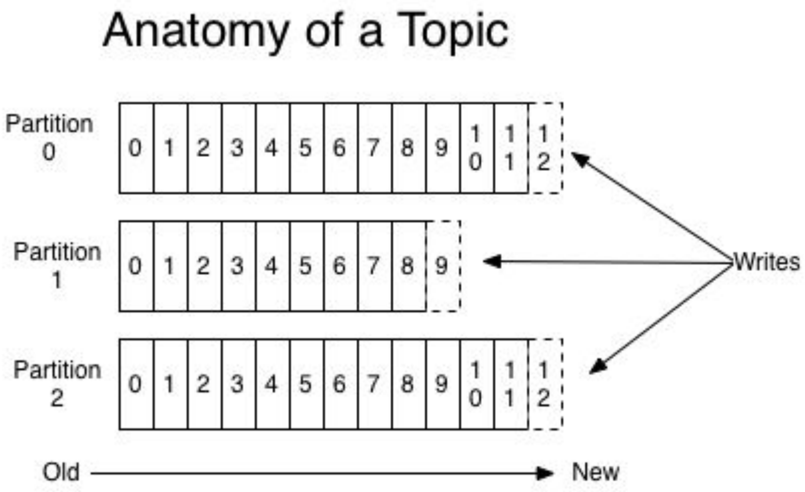


Figure source: https://kafka.apache.org/23/getting-started/introduction/

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

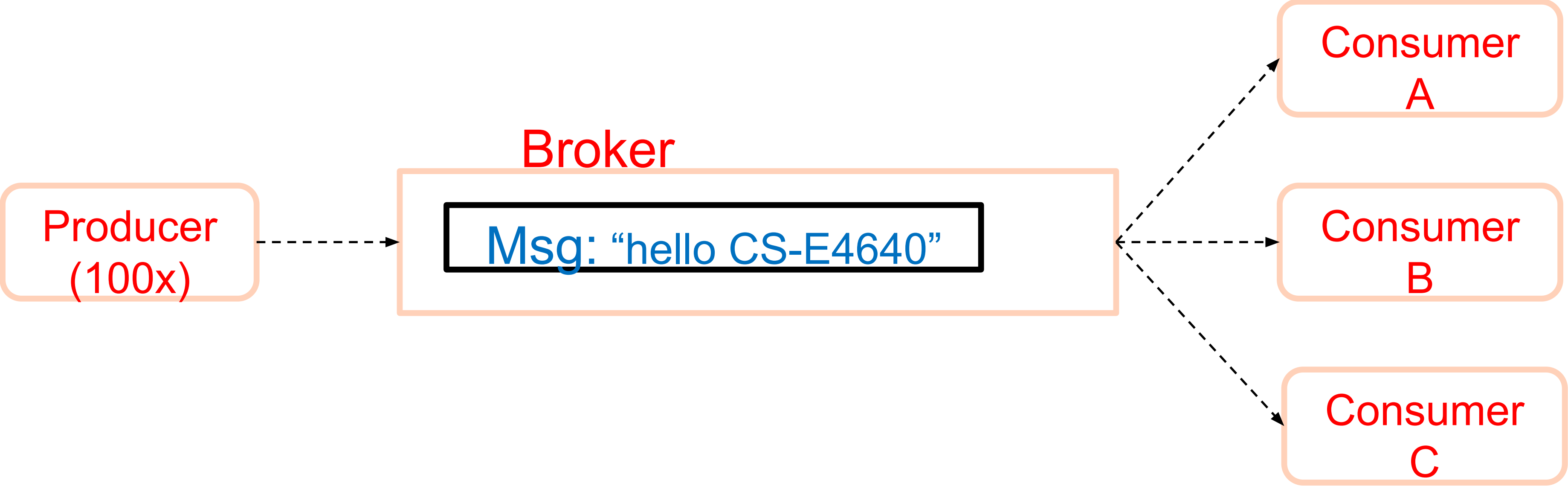
Example of a consumer

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

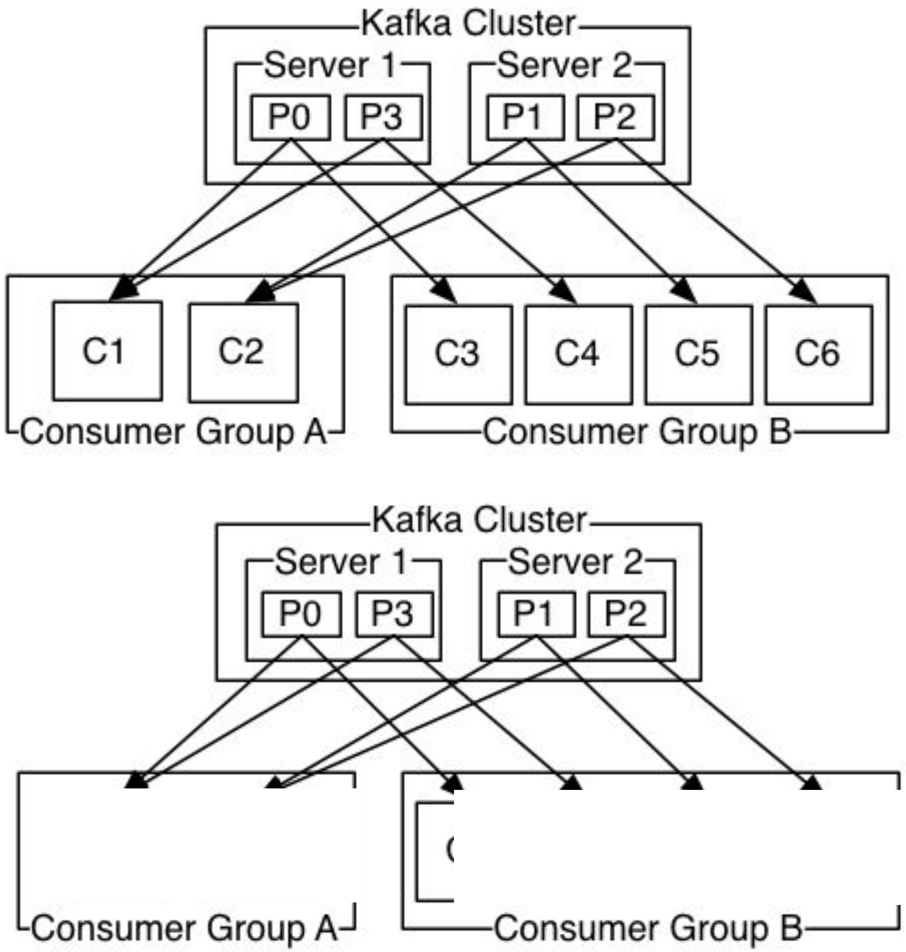
What does it mean exactly one?



- Producer: idempotent delivery ⇒ no duplicate entry in the log
- Transaction-like semantics:
  - within a transaction, EITHER messages 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: https://kafka.apache.org/23/getting-started/introduction/

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

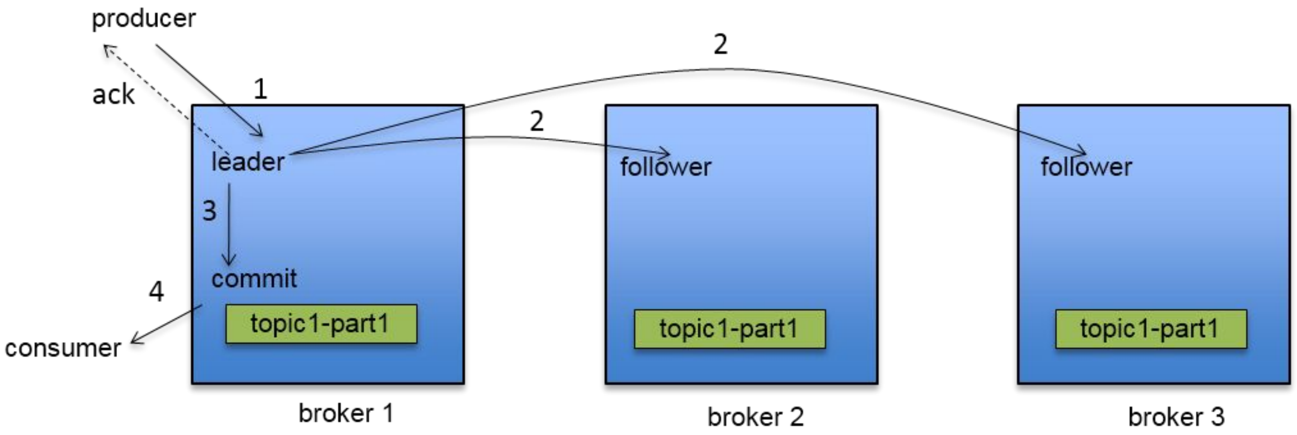


Figure source: http://de.slideshare.net/junrao/kafka-replication-apachecon2013

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

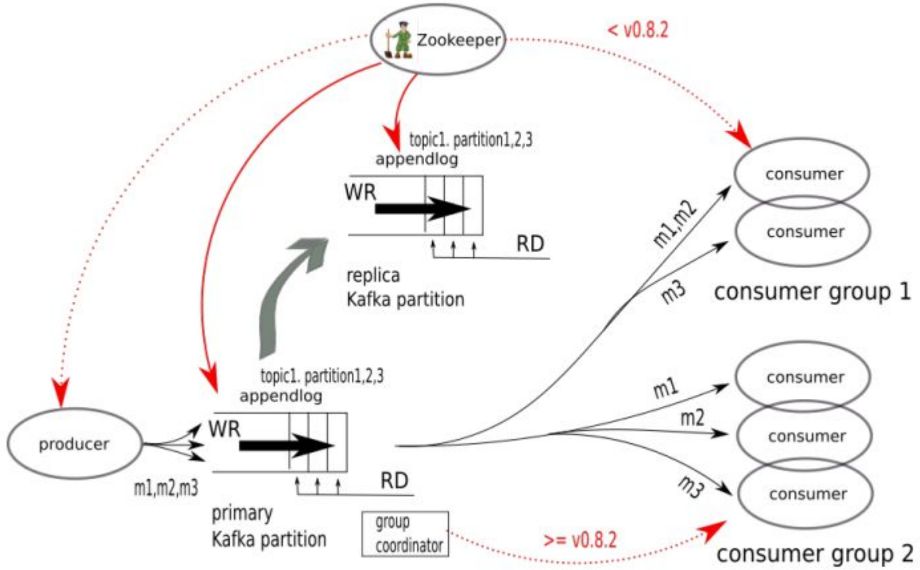


Figure 1: Kafka Architecture

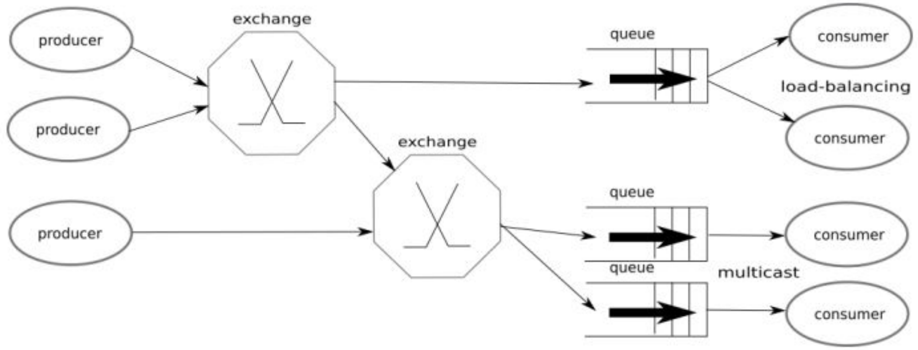


Figure 2: RabbitMQ (AMQP) Architecture

Hands-on

- Understanding the message broker systems and message delivery are key for stream processing

- Check our tutorial:
  - https://github.com/rdsea/bigdataplatforms/tree/master/tutorials/basickafka

Thanks!

Hong-Linh Truong
Department of Computer Science

rdsea.github.io

Kiitos
aalto.fi