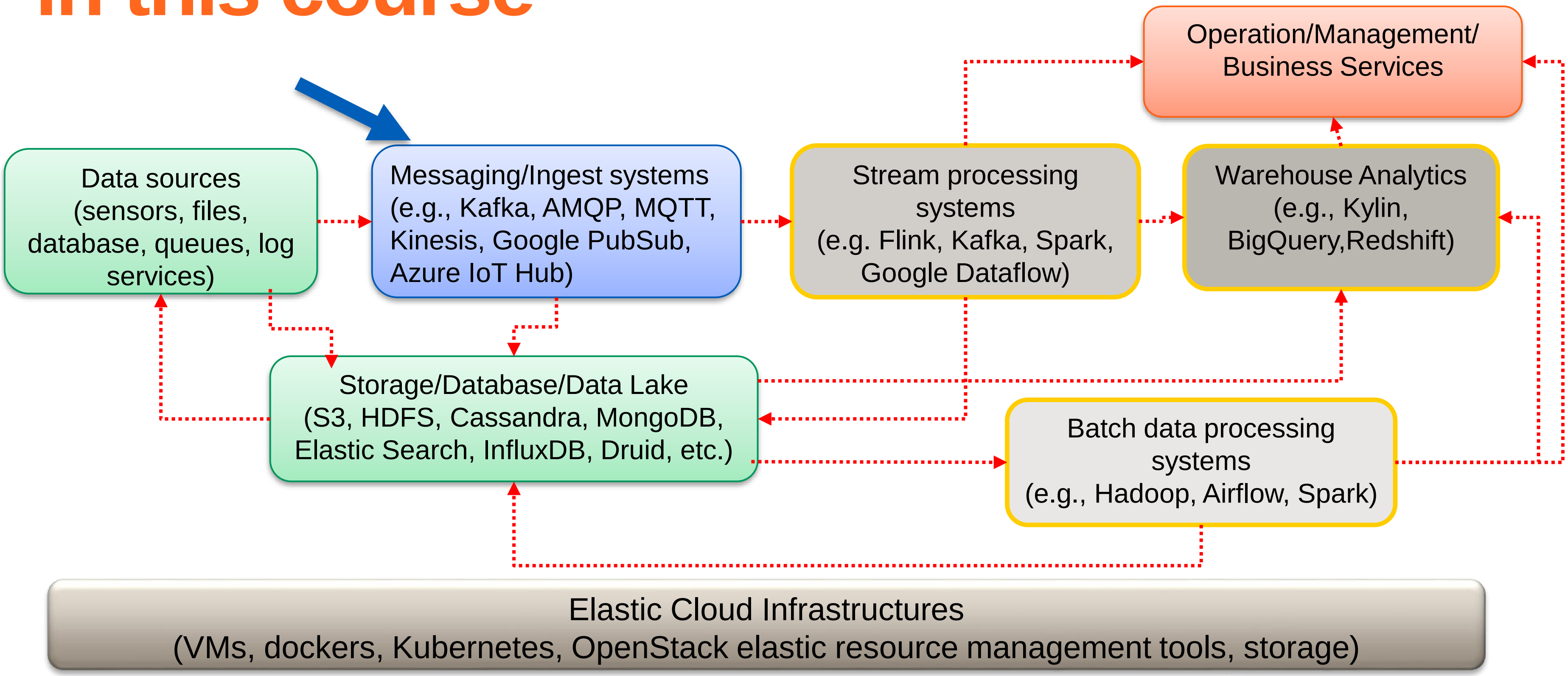
Aalto University
School of Science

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


Operation/Management/ Business Services
Data sources (sensors, files, database, queues, log services)
Messaging/Ingest systems (e.g., Kafka, AMQP, MQTT, Kinesis, Google PubSub, Azure IoT Hub)
Stream processing systems (e.g. Flink, Kafka, Spark, Google Dataflow)
Warehouse Analytics (e.g., Kylin, BigQuery,Redshift)
Storage/Database/Data Lake (S3, HDFS, Cassandra, MongoDB, Elastic Search, InfluxDB, Druid, etc.)
Batch data processing systems (e.g., Hadoop, Airflow, Spark)
Elastic Cloud Infrastructures (VMs, dockers, Kubernetes, OpenStack elastic resource management tools, storage)

Abstraction of Data Streams

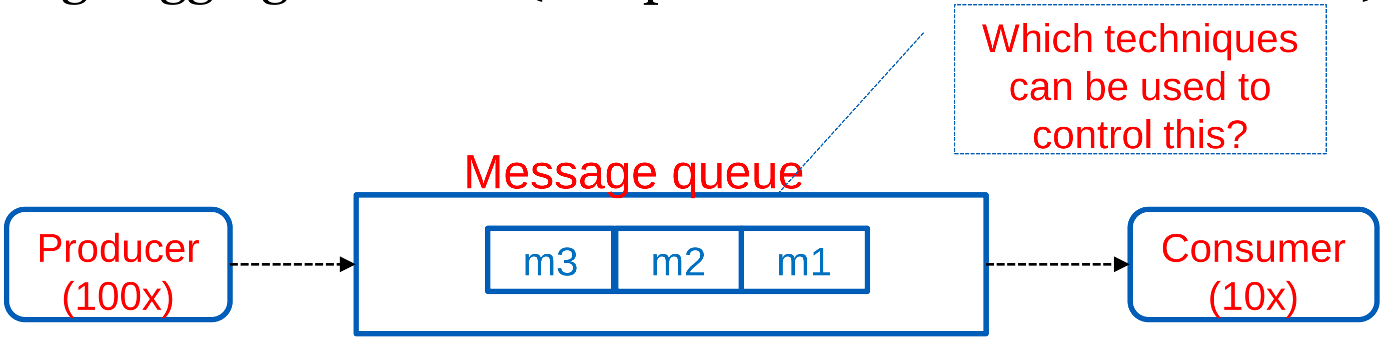
Data stream: a sequence/flow of *data units*

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

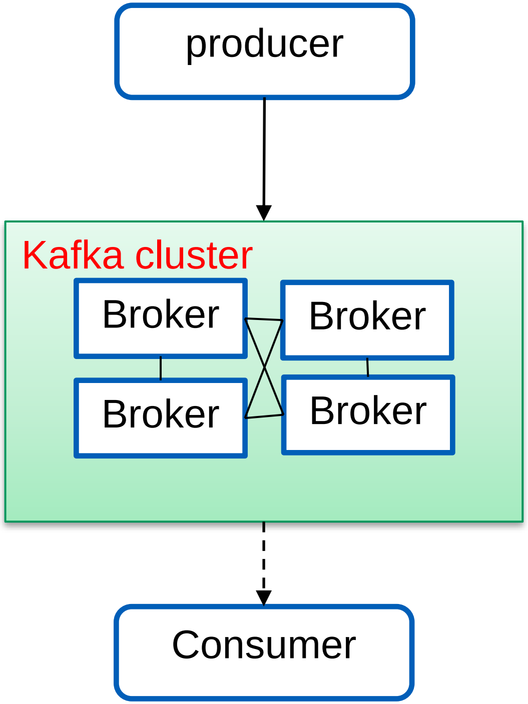
- **http://kafka.apache.org/**
  - originally from LinkedIn, not a protocol!
- **Some components are commercialized by Confluent**
  - https://www.confluent.io/
- **Widely used for big data use cases, including message processing in large-scale enterprise service platforms**
  - data messages (e.g.,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

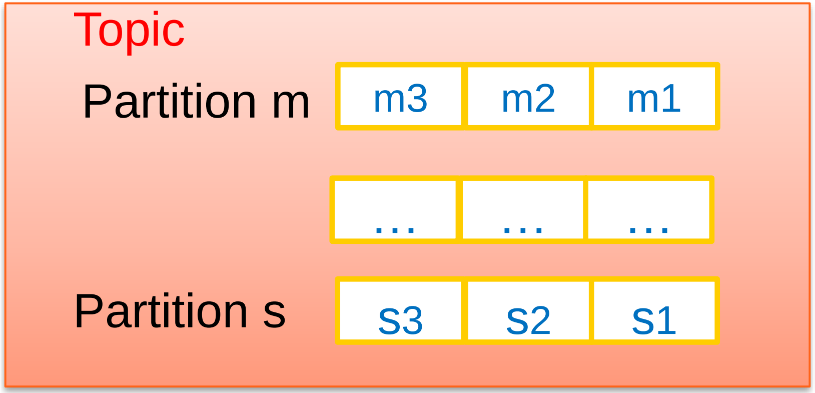
In the context of big data: we examine Apache Kafka for transferring, ingesting and processing messages of data

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

Anatomy of a Topic

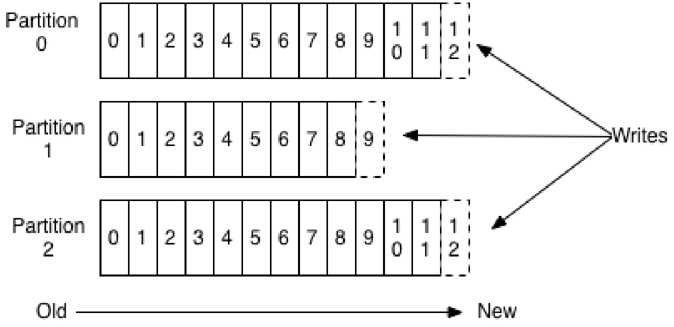


Figure source: http://kafka.apache.org/documentation.html

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?**

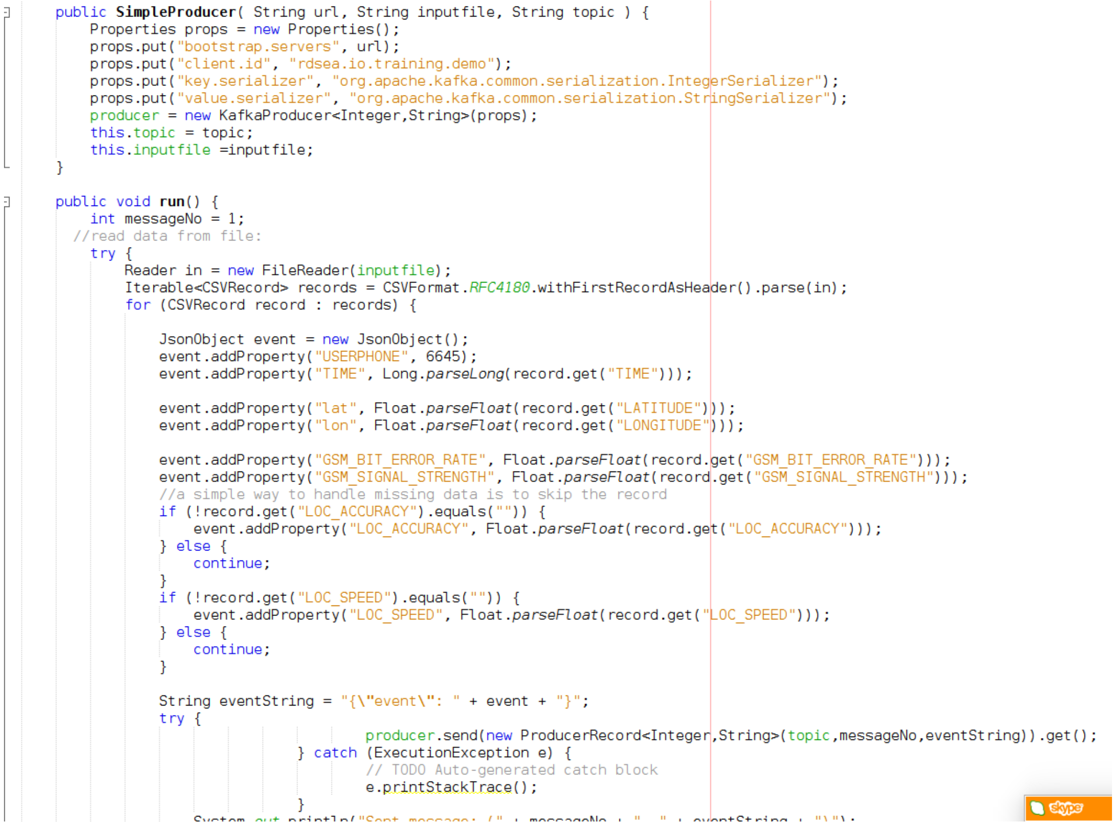
→ allow customers to design their speed

→ support/optimize batching data

→ easy to implement total order over message

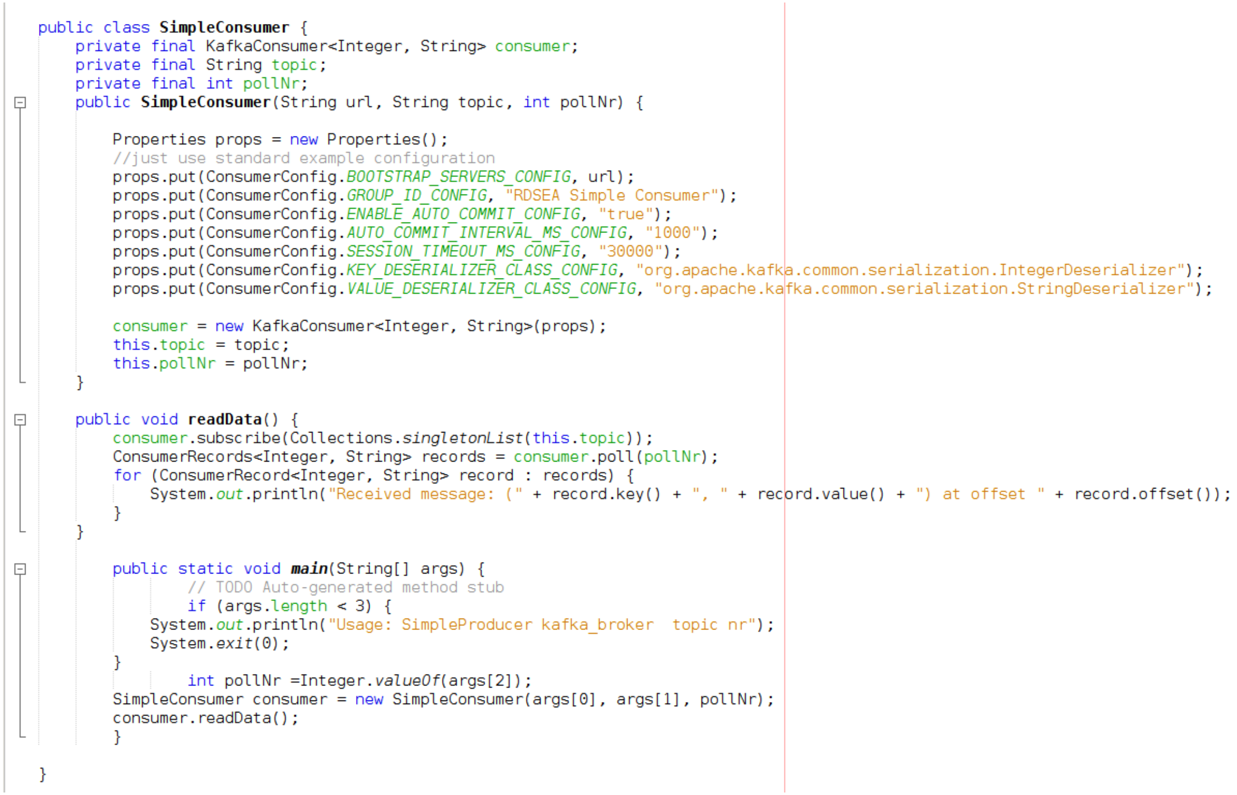
→ easy to implement reliable message/fault tolerance

Example of a producer



```
public SimpleProducer( String url, String inputfile, String topic ) {
    Properties props = new Properties();
    props.put("bootstrap.servers", url);
    props.put("client.id", "rdsea.io.training.demo");
    props.put("key.serializer", "org.apache.kafka.common.serialization.IntegerSerializer");
    props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
    producer = new KafkaProducer<Integer,String>(props);
    this.topic = topic;
    this.inputfile =inputfile;
}

public void run() {
    int messageNo = 1;
  //read data from file:
    try {
        Reader in = new FileReader(inputfile);
        Iterable<CSVRecord> records = CSVFormat.RFC4180.withFirstRecordAsHeader().parse(in);
        for (CSVRecord record : records) {

            JsonObject event = new JsonObject();
            event.addProperty("USERPHONE", 6645);
            event.addProperty("TIME", Long.parseLong(record.get("TIME")));

            event.addProperty("lat", Float.parseFloat(record.get("LATITUDE")));
            event.addProperty("lon", Float.parseFloat(record.get("LONGITUDE")));

            event.addProperty("GSM_BIT_ERROR_RATE", Float.parseFloat(record.get("GSM_BIT_ERROR_RATE")));
            event.addProperty("GSM_SIGNAL_STRENGTH", Float.parseFloat(record.get("GSM_SIGNAL_STRENGTH")));
            //a simple way to handle missing data is to skip the record
            if (!record.get("LOC_ACCURACY").equals("")) {
                event.addProperty("LOC_ACCURACY", Float.parseFloat(record.get("LOC_ACCURACY")));
            } else {
                continue;
            }
            if (!record.get("LOC_SPEED").equals("")) {
                event.addProperty("LOC_SPEED", Float.parseFloat(record.get("LOC_SPEED")));
            } else {
                continue;
            }

            String eventString = "{\"event\": " + event + "}";
            try {
                                    producer.send(new ProducerRecord<Integer,String>(topic,messageNo,eventString)).get();
                        } catch (ExecutionException e) {
                                    // TODO Auto-generated catch block
                                    e.printStackTrace();
                        }
                System.out.println("Sent message: (" + messageNo + ", " + eventString + ")");
```

Example of a consumer



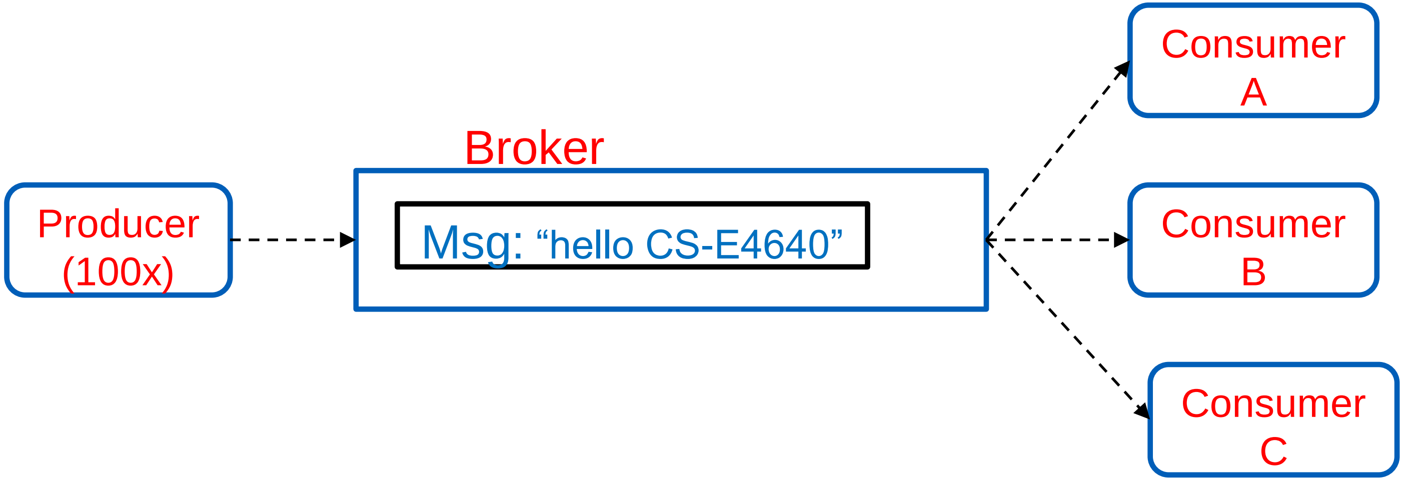
```
public class SimpleConsumer {
    private final KafkaConsumer<Integer, String> consumer;
    private final String topic;
    private final int pollNr;
    public SimpleConsumer(String url, String topic, int pollNr) {

        Properties props = new Properties();
        //just use standard example configuration
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, url);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "RDSEA Simple Consumer");
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
        props.put(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
        props.put(ConsumerConfig.SESSION_TIMEOUT_MS_CONFIG, "30000");
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.IntegerDeserializer");
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

        consumer = new KafkaConsumer<Integer, String>(props);
        this.topic = topic;
        this.pollNr = pollNr;
    }

    public void readData() {
        consumer.subscribe(Collections.singletonList(this.topic));
        ConsumerRecords<Integer, String> records = consumer.poll(pollNr);
        for (ConsumerRecord<Integer, String> record : records) {
            System.out.println("Received message: (" + record.key() + ", " + record.value() + ") at offset " + record.offset());
        }
    }

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                if (args.length < 3) {
            System.out.println("Usage: SimpleProducer kafka_broker  topic nr");
            System.exit(0);
        }
                int pollNr =Integer.valueOf(args[2]);
        SimpleConsumer consumer = new SimpleConsumer(args[0], args[1], pollNr);
        consumer.readData();
        }

}
```

Aalto University
School of Science

Message delivery

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

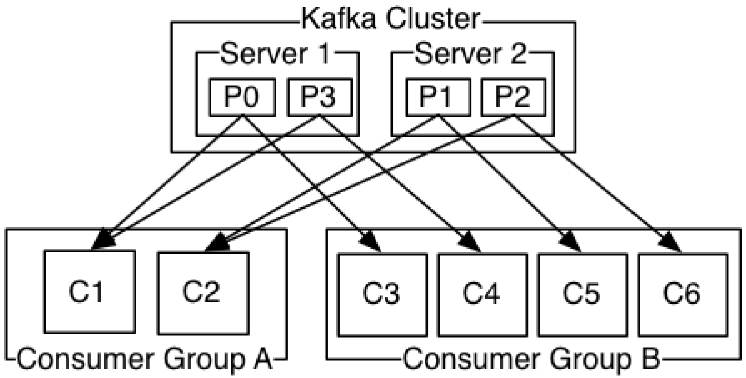
What does it mean exactly one?

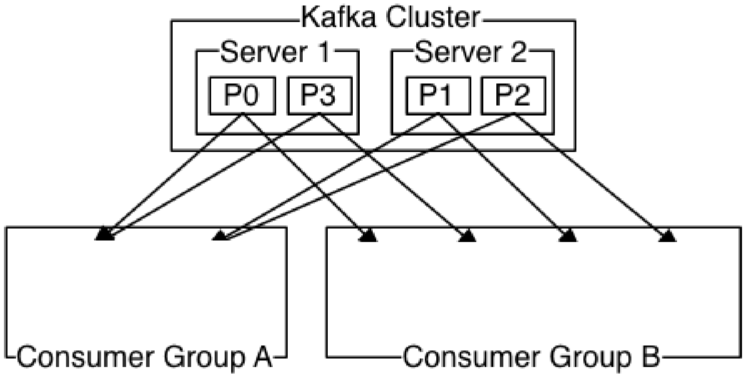


- **Producer: idempotent delivery → no duplicate entry in the log**
- **Transaction-like semantics: either message to ALL partition topics or not at all**
- **Consumer behavior management**

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

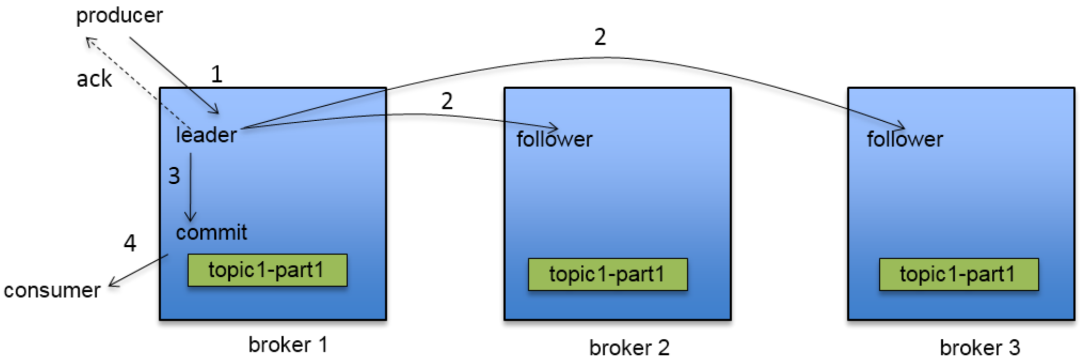


Figure source: http://de.slideshare.net/junrao/kafka-replication-apachecon2013

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

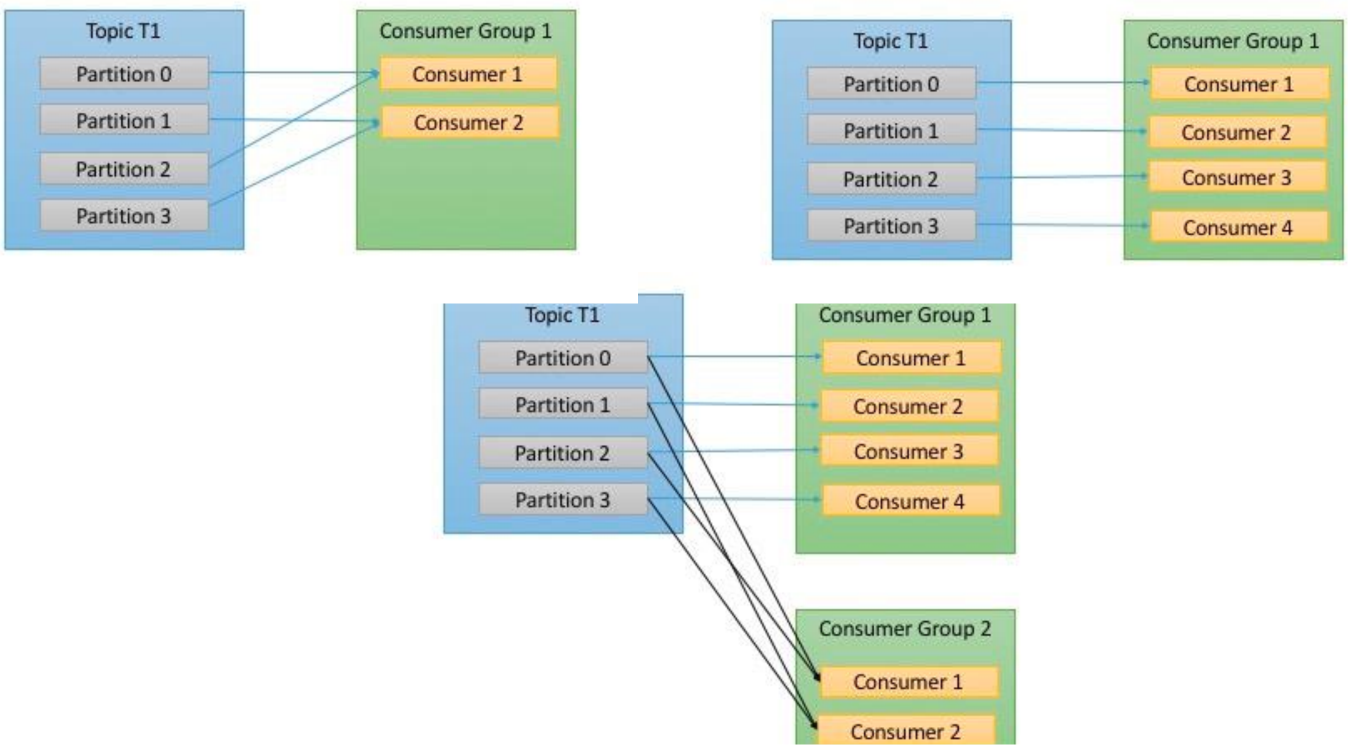


Figure source: https://www.safaribooksonline.com/library/view/kafka-the-definitive/9781491936153/ch04.html

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

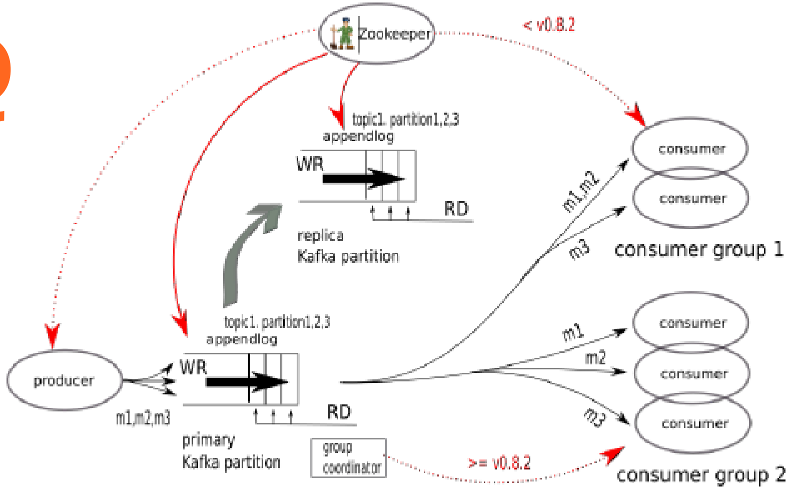


Figure 1: Kafka Architecture

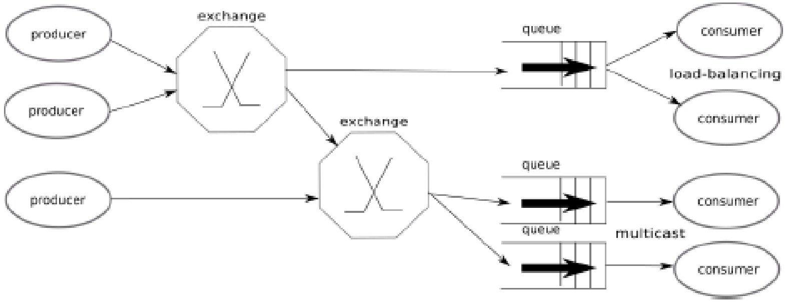


Figure 2: RabbitMQ (AMQP) Architecture

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