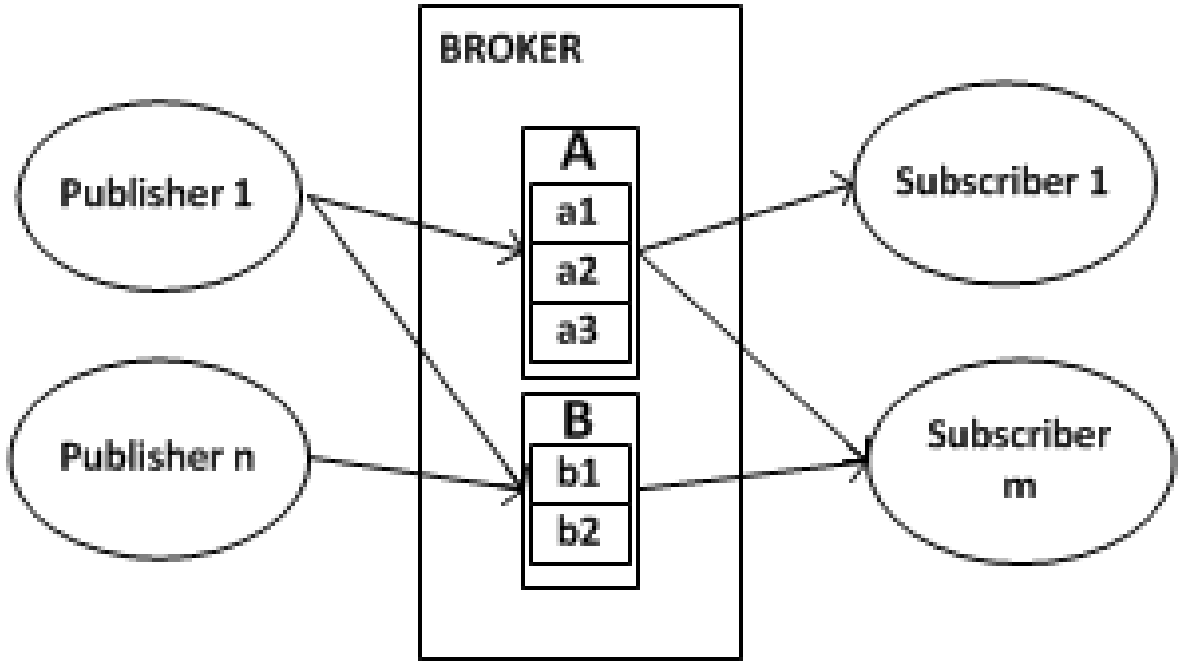
KAFKA

Óscar Pereira (omp@ua.pt)
University of Aveiro, Portugal

Goal

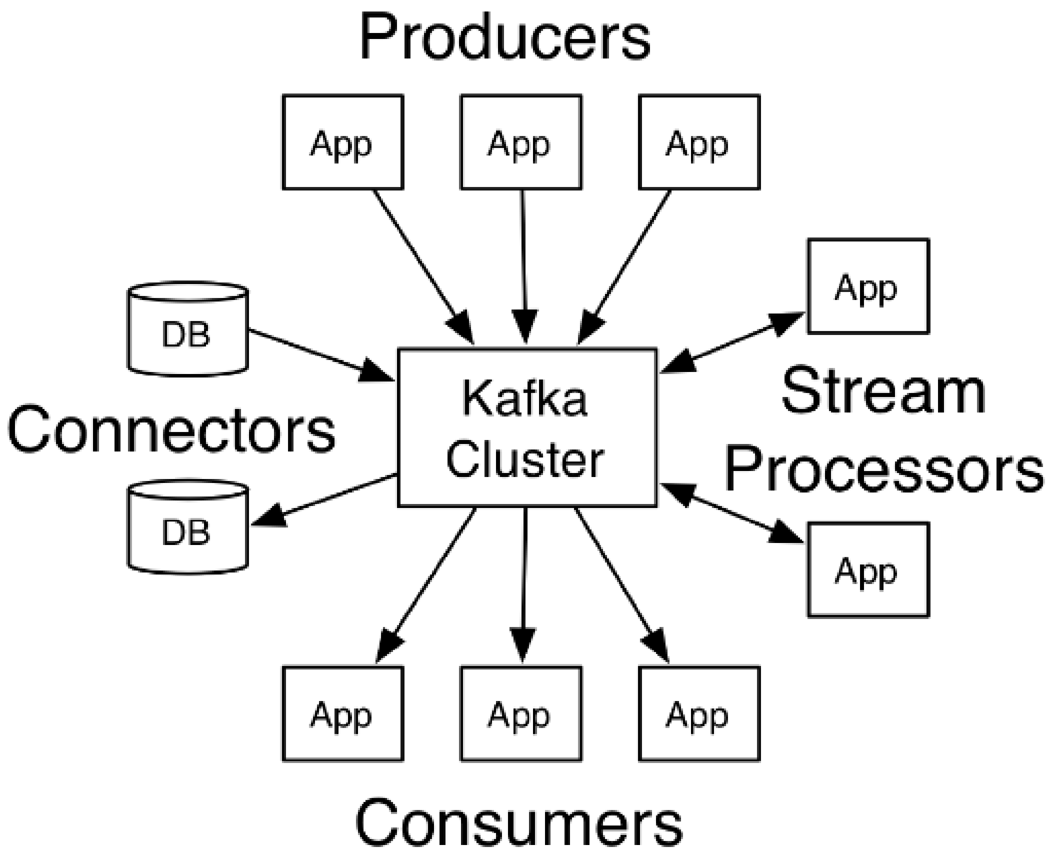
- Realize that Quality Attributes are the most important architectural entities
- Develop a project with KAFKA

What is Pub/Subs?



Basic API

- Producer API
- Consumer API
- Streams API
- Connector API



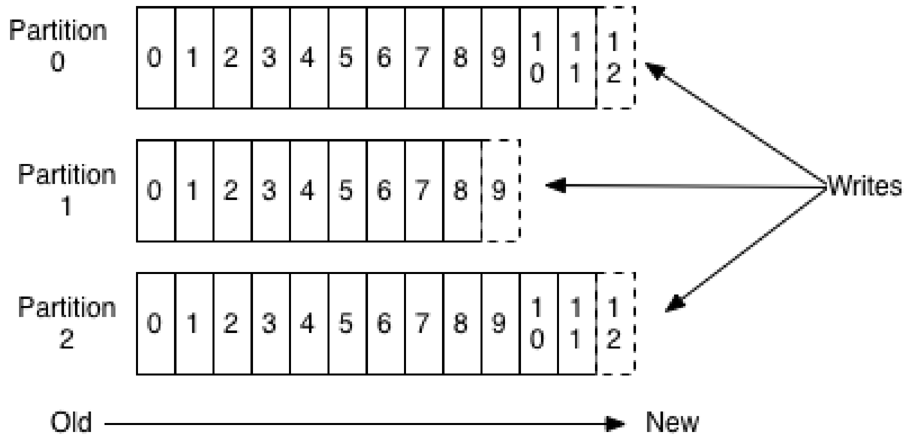
Concepts

- Producer: produces records
- Consumer: consumes records
- Broker: KAFKA server (running process)
- Cluster: group of coordinated brokers (zookeeper)
- Topic: name for KAKFA streams (customer orders)
- Partitions: where subsets of records of Topics are split
- Offset: seq number of associated with records
- Consumer Groups: to share work from Topics
- A record: key, a value and a timestamp (byte array)

Topics

- Divided in partitions deployed in brokers
- Immutable sequence
- Offset: write, read and commit
- Multi-subscriber
- Retention policy

(compacted topics)



Fault-Tolerance

- Definition: system to continue operating properly even if some components fail
- Topics split in one or more partitions
- Partitions are deployed in Brokers
- What if a broker goes down?

Fault-Tolerance

- HOW:
  - KAFKA can replicate partitions
  - Defined at the Topic level
  - One replica is the leader, others are followers
  - Leader:
    - where consumers and producers act
    - transparent for consumers and producers
  - Followers: read records from the leader
  - Consumers can be reallocated to partitions (consumer group)

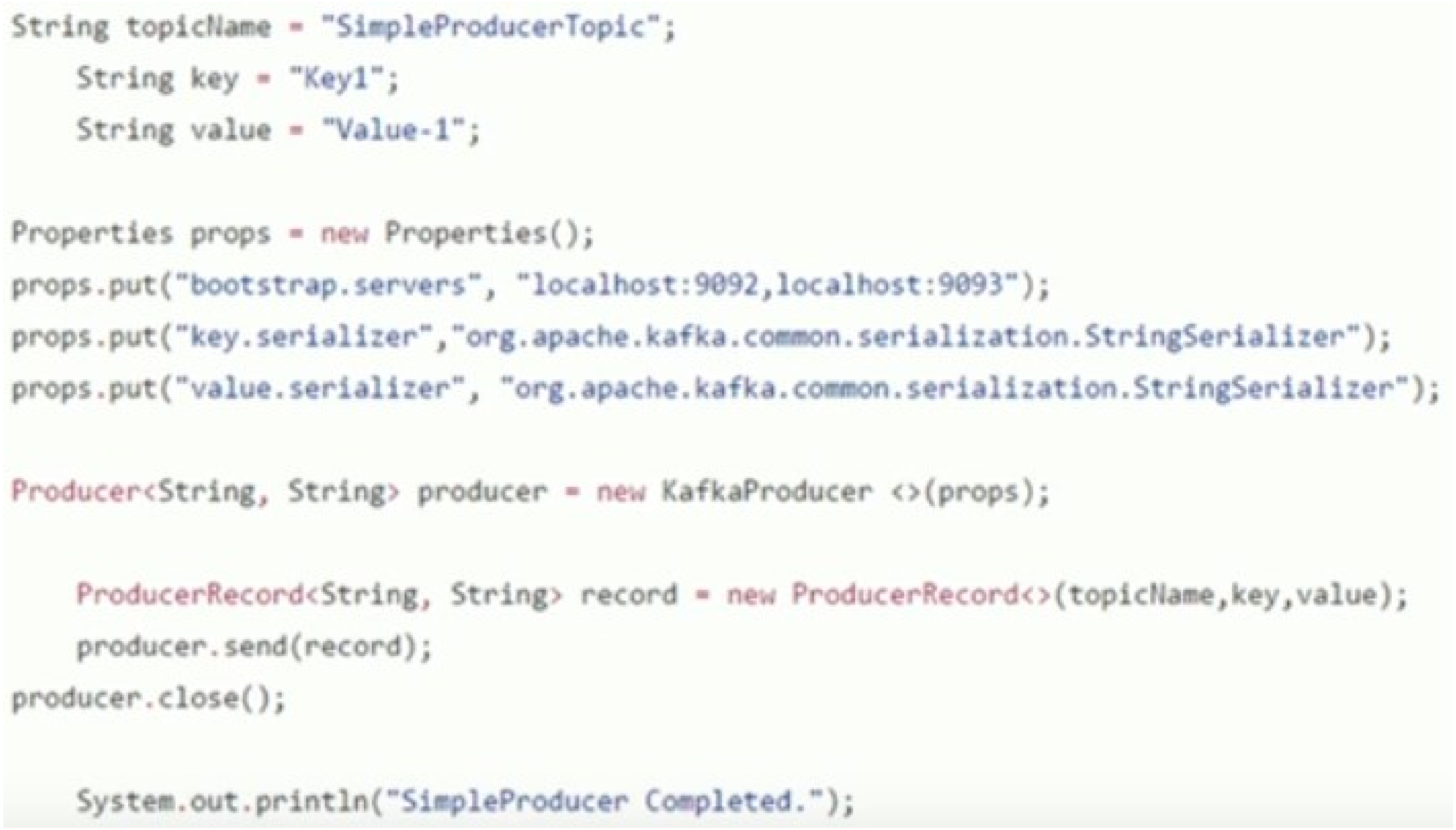
Broker Configurations

- zookeeper.connect: localhost:2181
- broker.id
- port
- log.dirs
- delete.topic.enable (false)
- auto.create.topics.enable (true)
- default.replication.factor (1)

Broker Configurations

- num.partitions (1)
- log.retention.hours … (7 days)
- log.retention.bytes (user specified)
- Many more ….. you must be aware about them

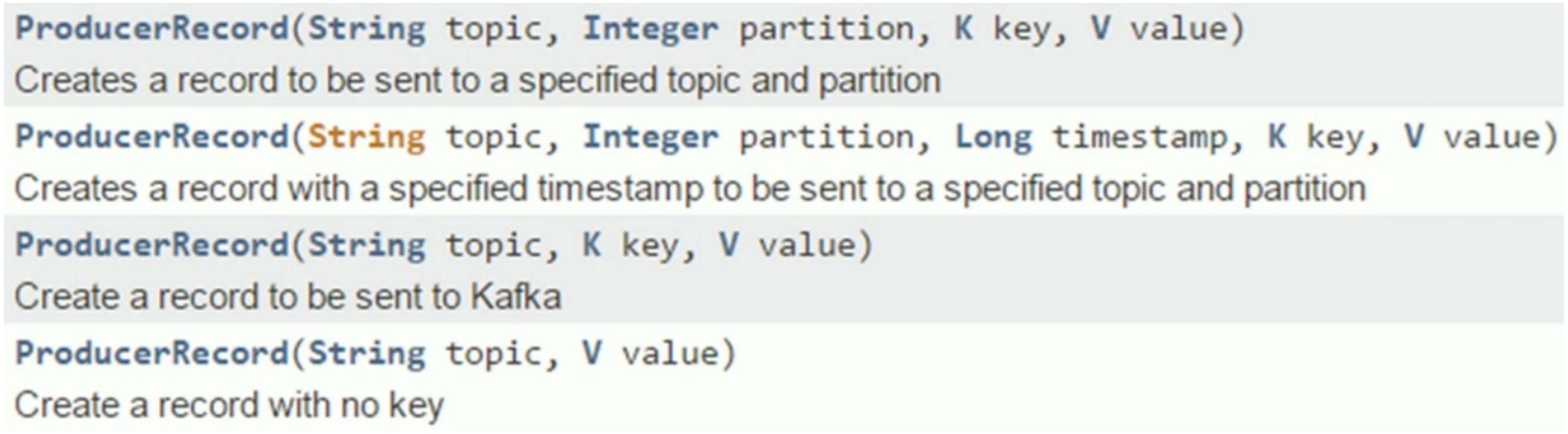
Producer API



```
String topicName = "SimpleProducerTopic";
    String key = "Key1";
    String value = "Value-1";

Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092,localhost:9093");
props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

Producer<String, String> producer = new KafkaProducer <>(props);

    ProducerRecord<String, String> record = new ProducerRecord<>(topicName,key,value);
    producer.send(record);
producer.close();

    System.out.println("SimpleProducer Completed.");
```

ProducerRecord Object

Some Constructors:



`ProducerRecord(String topic, Integer partition, K key, V value)`

Creates a record to be sent to a specified topic and partition

`ProducerRecord(String topic, Integer partition, Long timestamp, K key, V value)`

Creates a record with a specified timestamp to be sent to a specified topic and partition

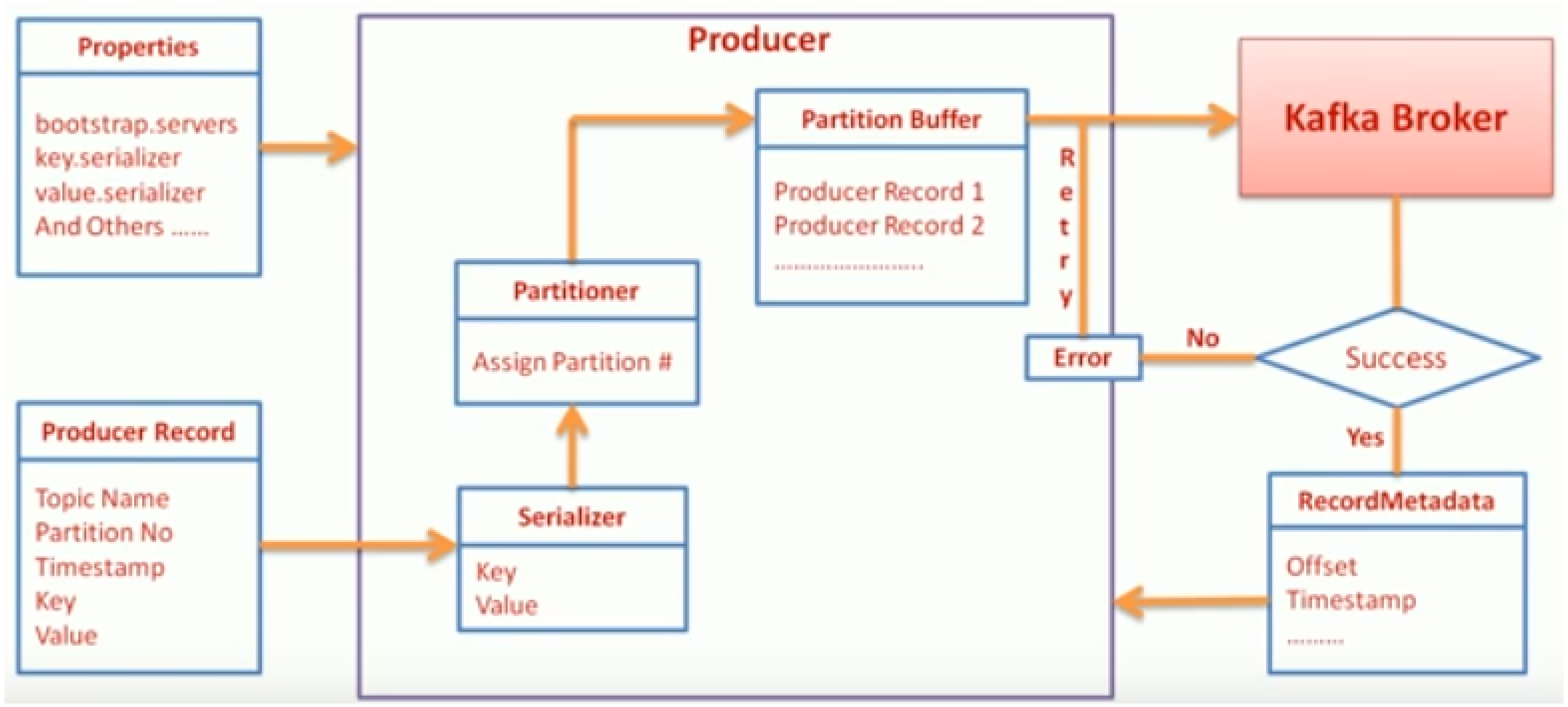
`ProducerRecord(String topic, K key, V value)`

Create a record to be sent to Kafka

`ProducerRecord(String topic, V value)`

Create a record with no key

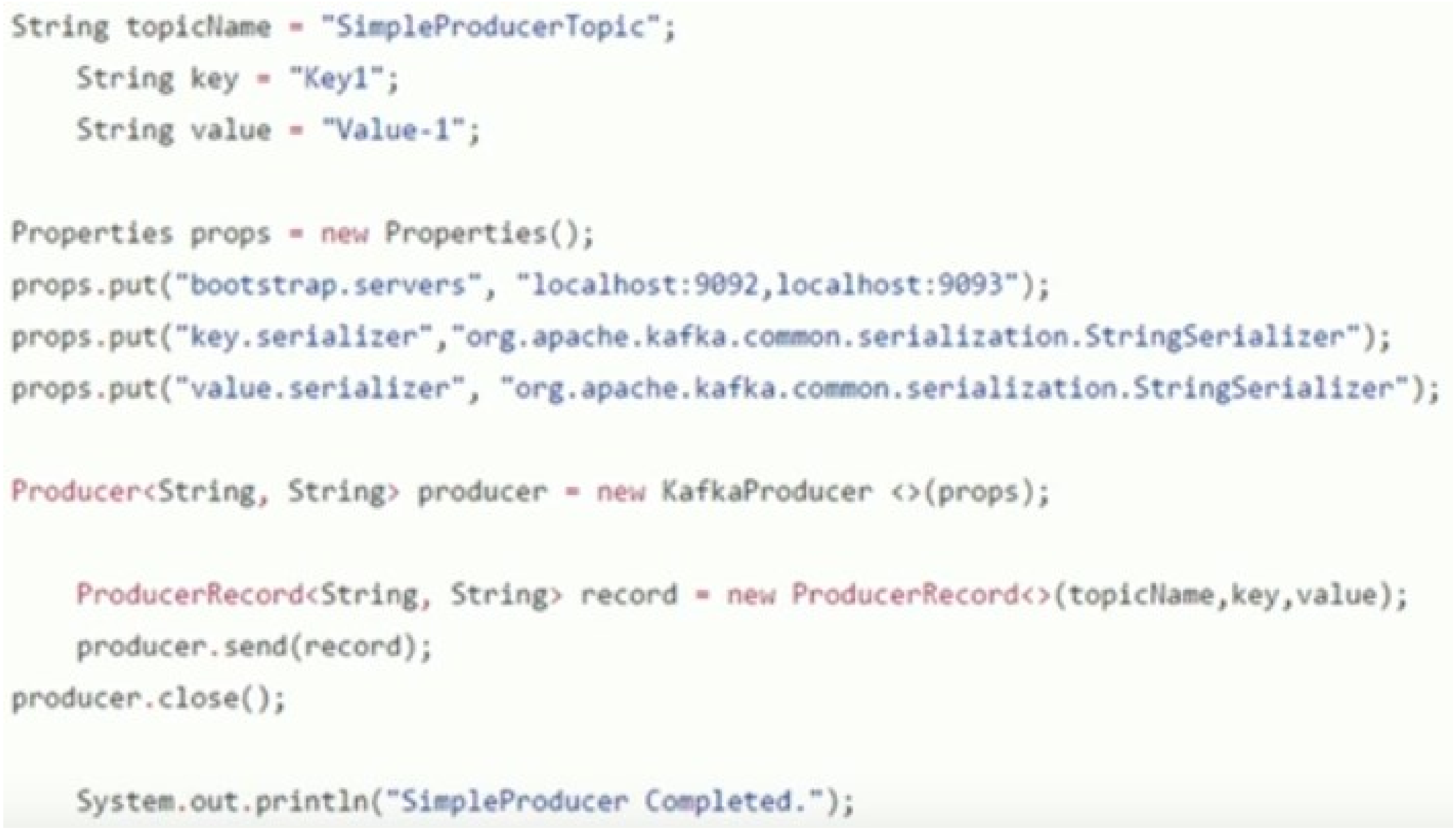
Producers


Properties
bootstrap.servers
key.serializer
value.serializer
And Others ......
Producer Record
Topic Name
Partition No
Timestamp
Key
Value
Producer
Partitioner
Assign Partition #
Serializer
Key
Value
Partition Buffer
Producer Record 1
Producer Record 2
....
Retry
Error
No
Kafka Broker
Success
Yes
RecordMetadata
Offset
Timestamp
....

Producers

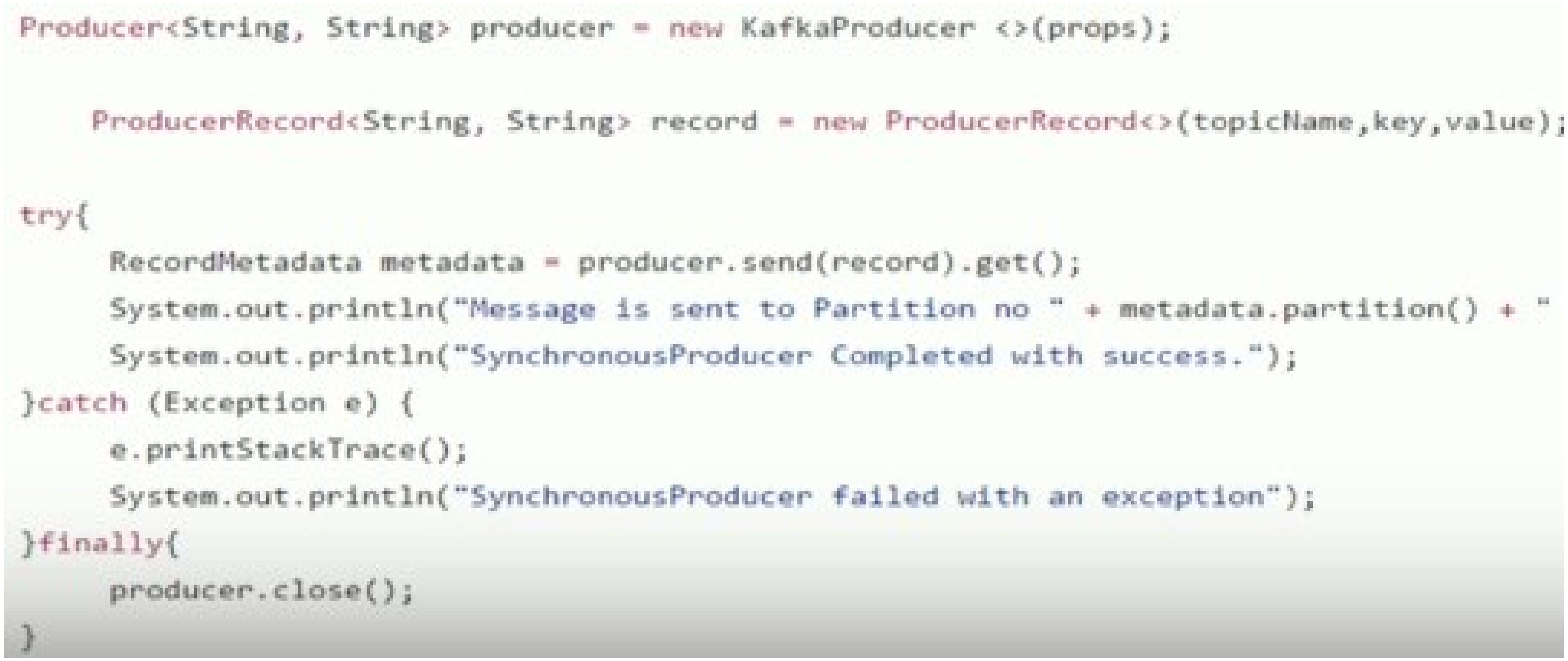
- Dealing with throughput/fault-tolerance:
  - Fire and forget: adv and disadv (already shown)
  - Synchronous send (each record): success or failure
  - Asynchronous send - (max.in.flight.requests.per.connection - 5)

Producer: fire and forget



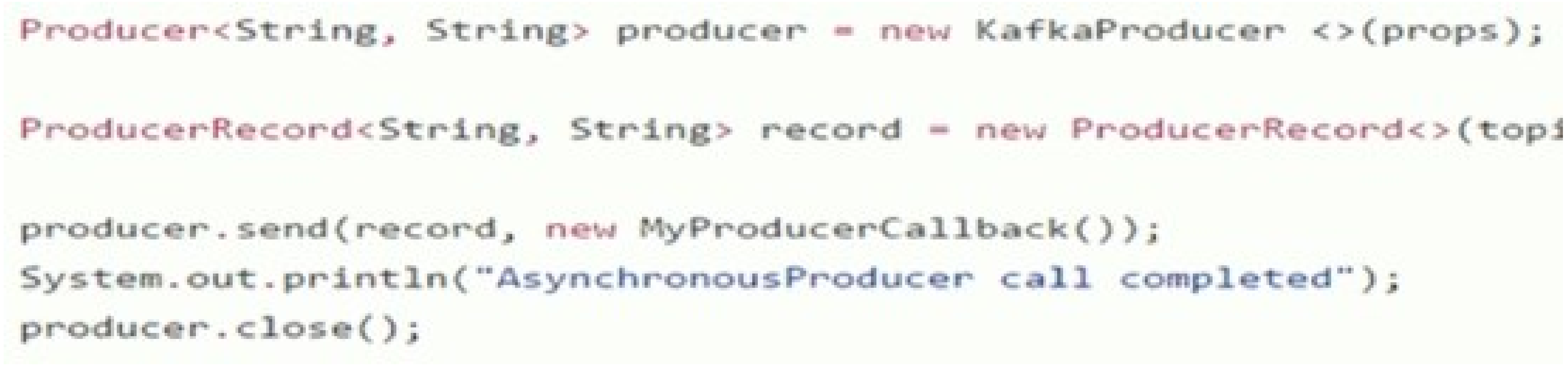
```
String topicName = "SimpleProducerTopic";
    String key = "Key1";
    String value = "Value-1";

Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092,localhost:9093");
props.put("key.serializer","org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

Producer<String, String> producer = new KafkaProducer <>(props);

    ProducerRecord<String, String> record = new ProducerRecord<>(topicName,key,value);
    producer.send(record);
producer.close();

    System.out.println("SimpleProducer Completed.");
```

Producer: synchronous send

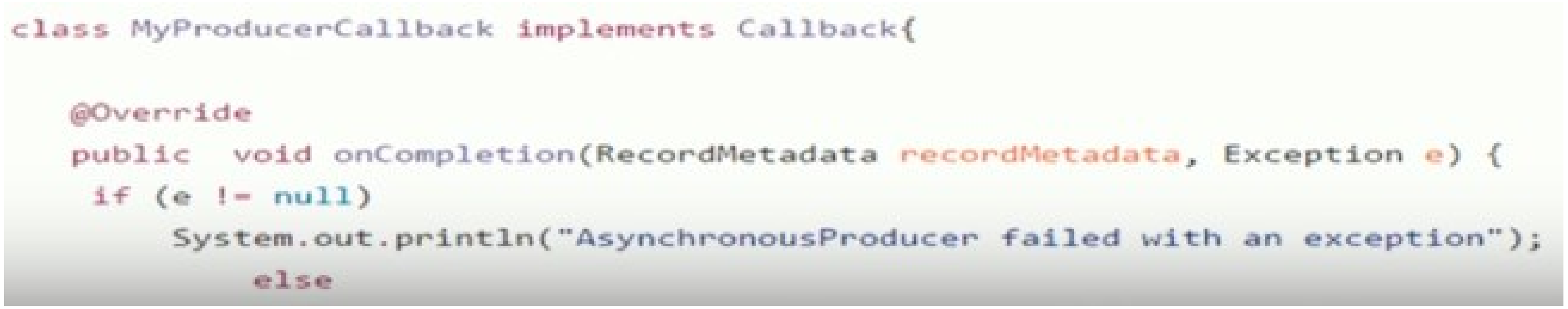


```
Producer<String, String> producer = new KafkaProducer <>(props);

    ProducerRecord<String, String> record = new ProducerRecord<>(topicName,key,value);

try{
     RecordMetadata metadata = producer.send(record).get();
     System.out.println("Message is sent to Partition no " + metadata.partition() + "
     System.out.println("SynchronousProducer Completed with success.");
}catch (Exception e) {
     e.printStackTrace();
     System.out.println("SynchronousProducer failed with an exception");
}finally{
     producer.close();
}
```

Producer: asynchronous send



```
Producer<String, String> producer = new KafkaProducer <>(props);

ProducerRecord<String, String> record = new ProducerRecord<>(topi

producer.send(record, new MyProducerCallback());
System.out.println("AsynchronousProducer call completed");
producer.close();
```



```
class MyProducerCallback implements Callback{

   @Override
   public  void onCompletion(RecordMetadata recordMetadata, Exception e) {
    if (e != null)
        System.out.println("AsynchronousProducer failed with an exception");
            else
```

Producers: some properties

- bootstrap.servers
  key.serializer, value.serializer, partitioner.class
- acks =
  - 0: no ack, high throughput, no retries
    - High-throughput
    - Possible loss of messages
    - No retries
  - 1: leader ack after write on leader
  - -1: leader ack after write on all in-sync replicas

Producers: some properties

- retries:
  - number of retries
- max.in.flight.requests.per.connection
  - High value:
    - High throughput
    - Increased probability of losing order of batches
  - Critical if order of batches is important
- **Many more ….. you should be aware about them**

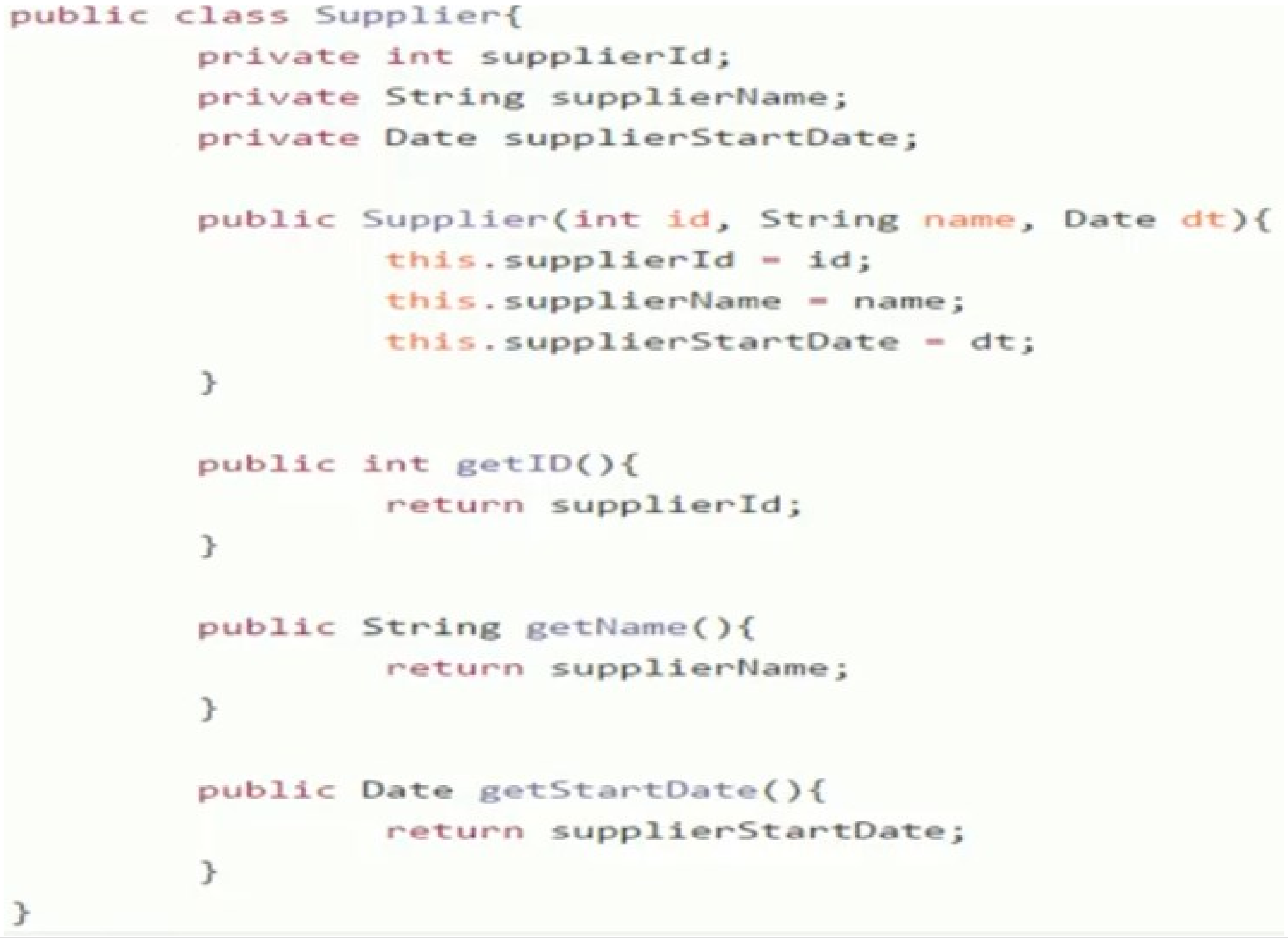
How to Partition?

- Partitioner policy:
  - a partition can be explicitly specified
  - If not and a key is present, KAFKA hashes the key
  - If none, KAKFA uses round-robin
  - Custom partitioner

Custom Serializers

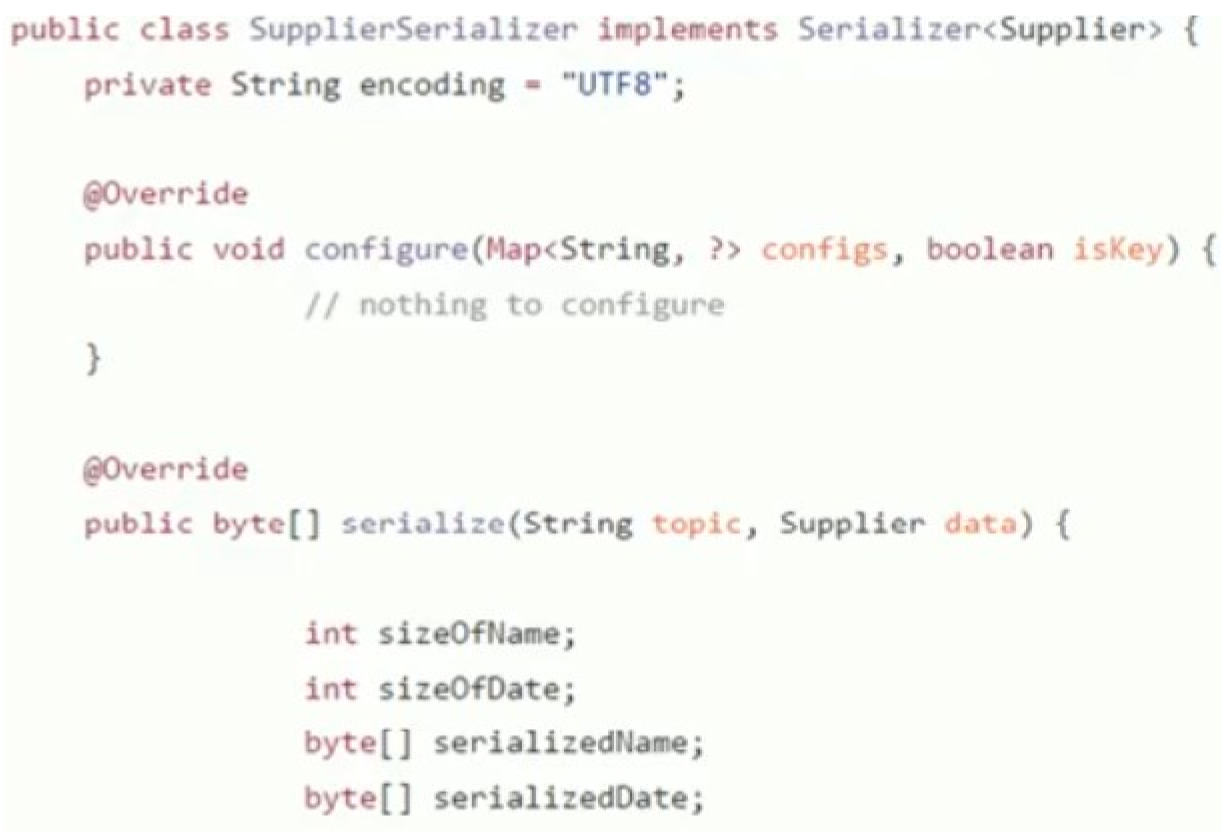
- Default serializers for java data types
- other data types:
  - KAFKA:
    - Serializer kafka class:
      - Class for the record schema with getter methods
      - Class implements Serializer<class>
    - Deserializer: opposite to serializer
  - AVRO: schema evolution

Custom Serializers: object to serialize



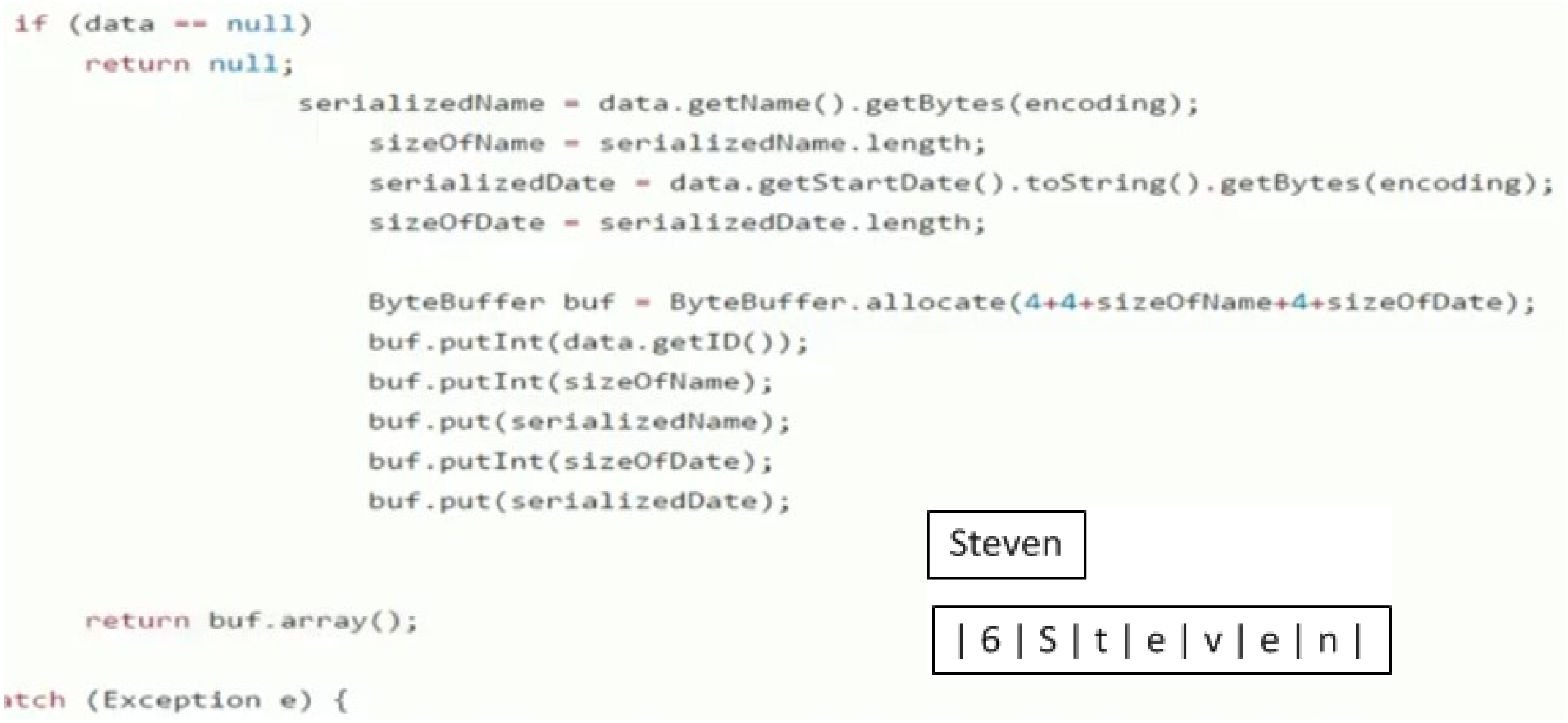
```
public class Supplier{
        private int supplierId;
        private String supplierName;
        private Date supplierStartDate;

        public Supplier(int id, String name, Date dt){
                this.supplierId = id;
                this.supplierName = name;
                this.supplierStartDate = dt;
        }

        public int getID(){
                return supplierId;
        }

        public String getName(){
                return supplierName;
        }

        public Date getStartDate(){
                return supplierStartDate;
        }
}
```

Custom Serializer: serializer class



```
public class SupplierSerializer implements Serializer<Supplier> {
    private String encoding = "UTF8";

    @Override
    public void configure(Map<String, ?> configs, boolean isKey) {
                // nothing to configure
    }

    @Override
    public byte[] serialize(String topic, Supplier data) {

                int sizeOfName;
                int sizeOfDate;
                byte[] serializedName;
                byte[] serializedDate;
```

Custom Serializers

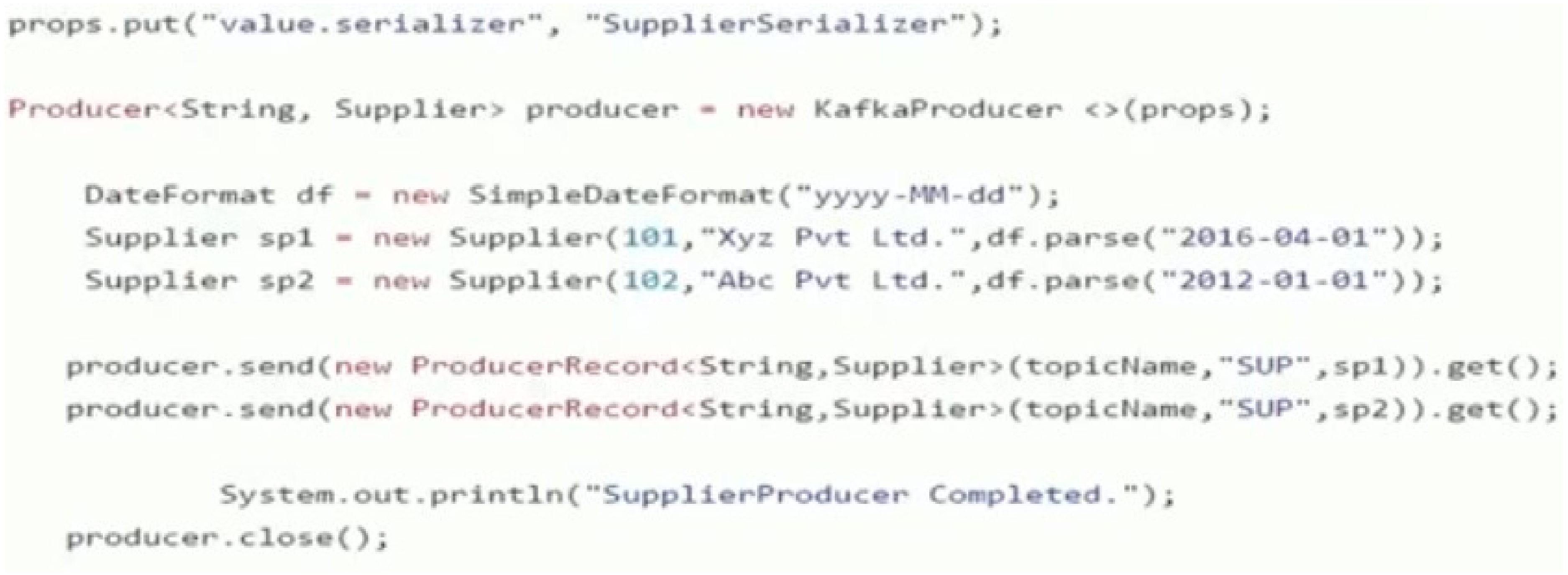


```
if (data == null)
    return null;
                serializedName = data.getName().getBytes(encoding);
                    sizeOfName = serializedName.length;
                    serializedDate = data.getStartDate().toString().getBytes(encoding);
                    sizeOfDate = serializedDate.length;

                    ByteBuffer buf = ByteBuffer.allocate(4+4+sizeOfName+4+sizeOfDate);
                    buf.putInt(data.getID());
                    buf.putInt(sizeOfName);
                    buf.put(serializedName);
                    buf.putInt(sizeOfDate);
                    buf.put(serializedDate);


    return buf.array();

atch (Exception e) {
```

Steven

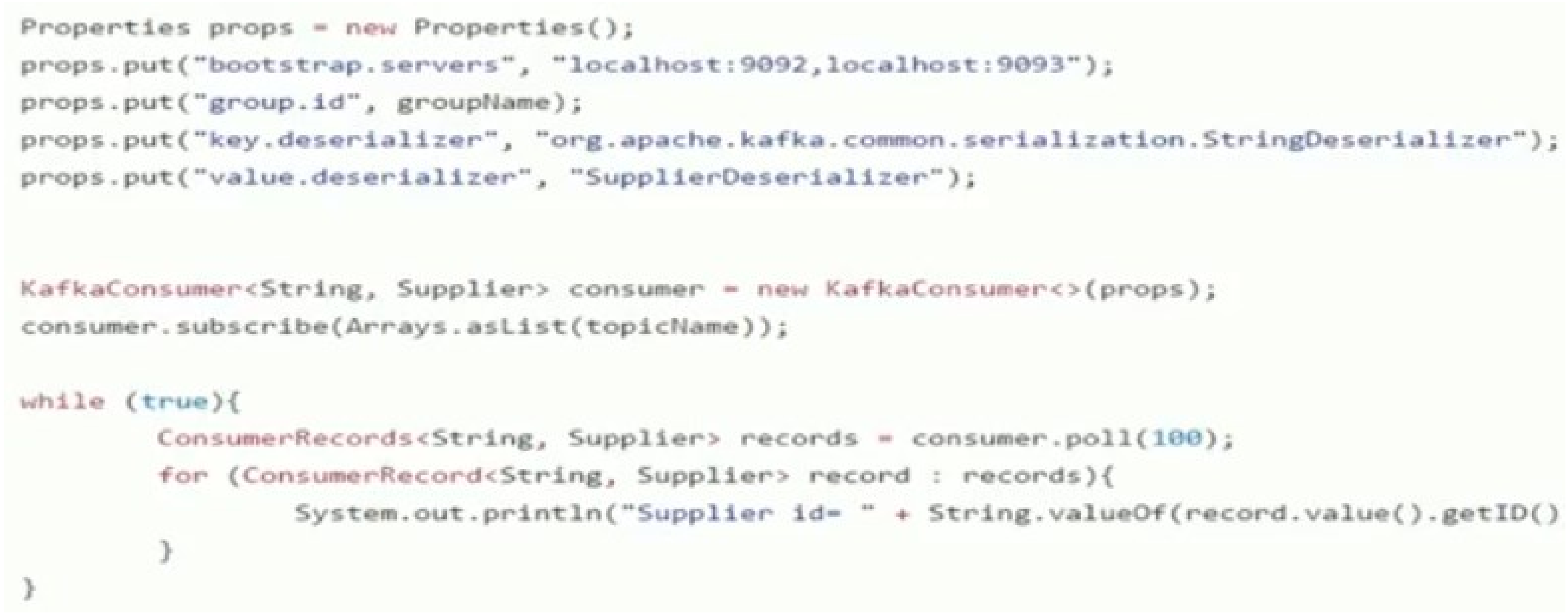
| 6 | S | t | e | v | e | n |

Custom Serializers: producers



```
props.put("value.serializer", "SupplierSerializer");

Producer<String, Supplier> producer = new KafkaProducer <>(props);

    DateFormat df = new SimpleDateFormat("yyyy-MM-dd");
    Supplier sp1 = new Supplier(101,"Xyz Pvt Ltd.",df.parse("2016-04-01"));
    Supplier sp2 = new Supplier(102,"Abc Pvt Ltd.",df.parse("2012-01-01"));

   producer.send(new ProducerRecord<String,Supplier>(topicName,"SUP",sp1)).get();
   producer.send(new ProducerRecord<String,Supplier>(topicName,"SUP",sp2)).get();

           System.out.println("SupplierProducer Completed.");
   producer.close();
```

Custom Serializer: consumers

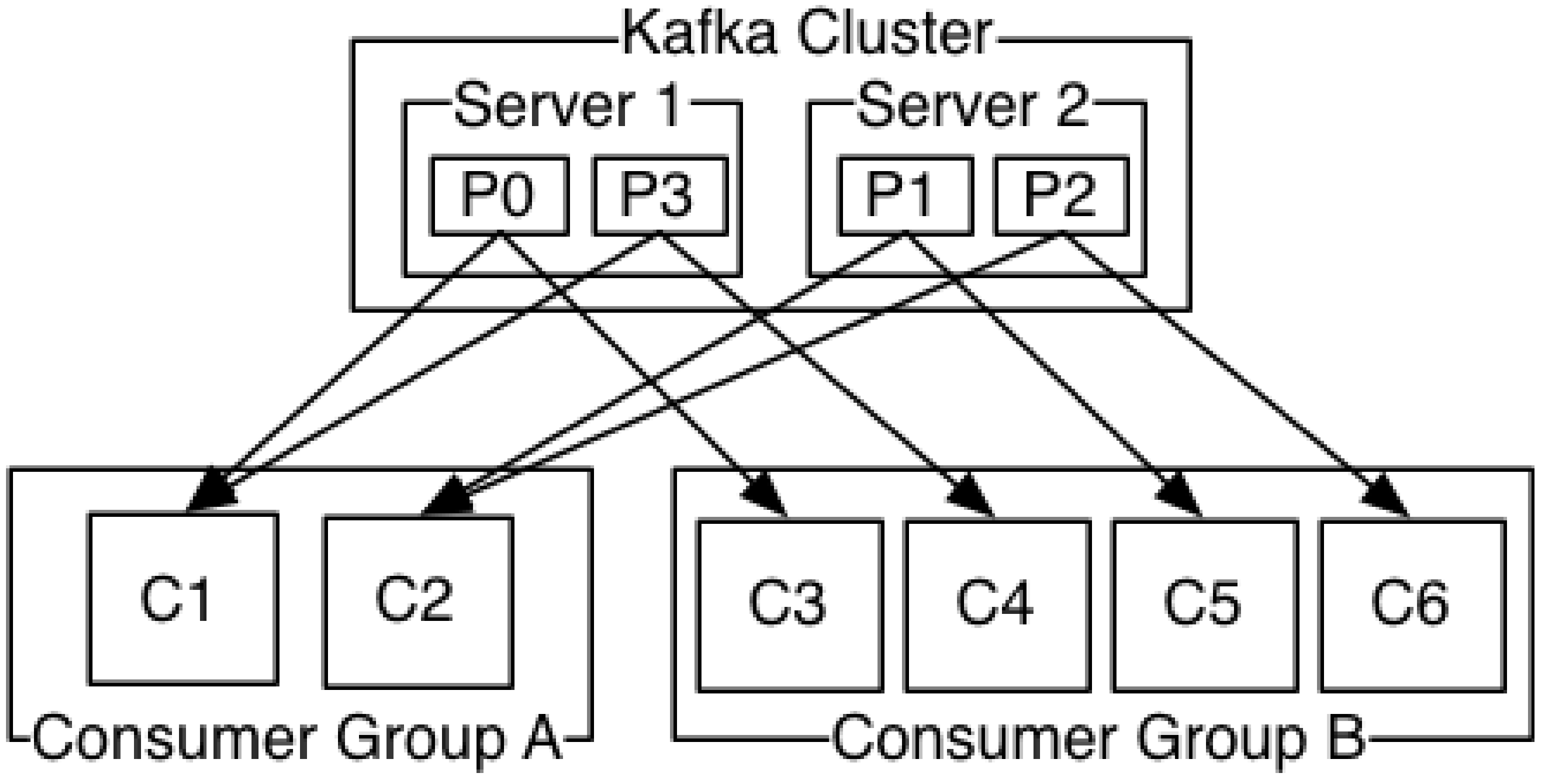


```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092,localhost:9093");
props.put("group.id", groupName);
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "SupplierDeserializer");


KafkaConsumer<String, Supplier> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList(topicName));

while (true){
        ConsumerRecords<String, Supplier> records = consumer.poll(100);
        for (ConsumerRecord<String, Supplier> record : records){
                System.out.println("Supplier id= " + String.valueOf(record.value().getID()
        }
}
```

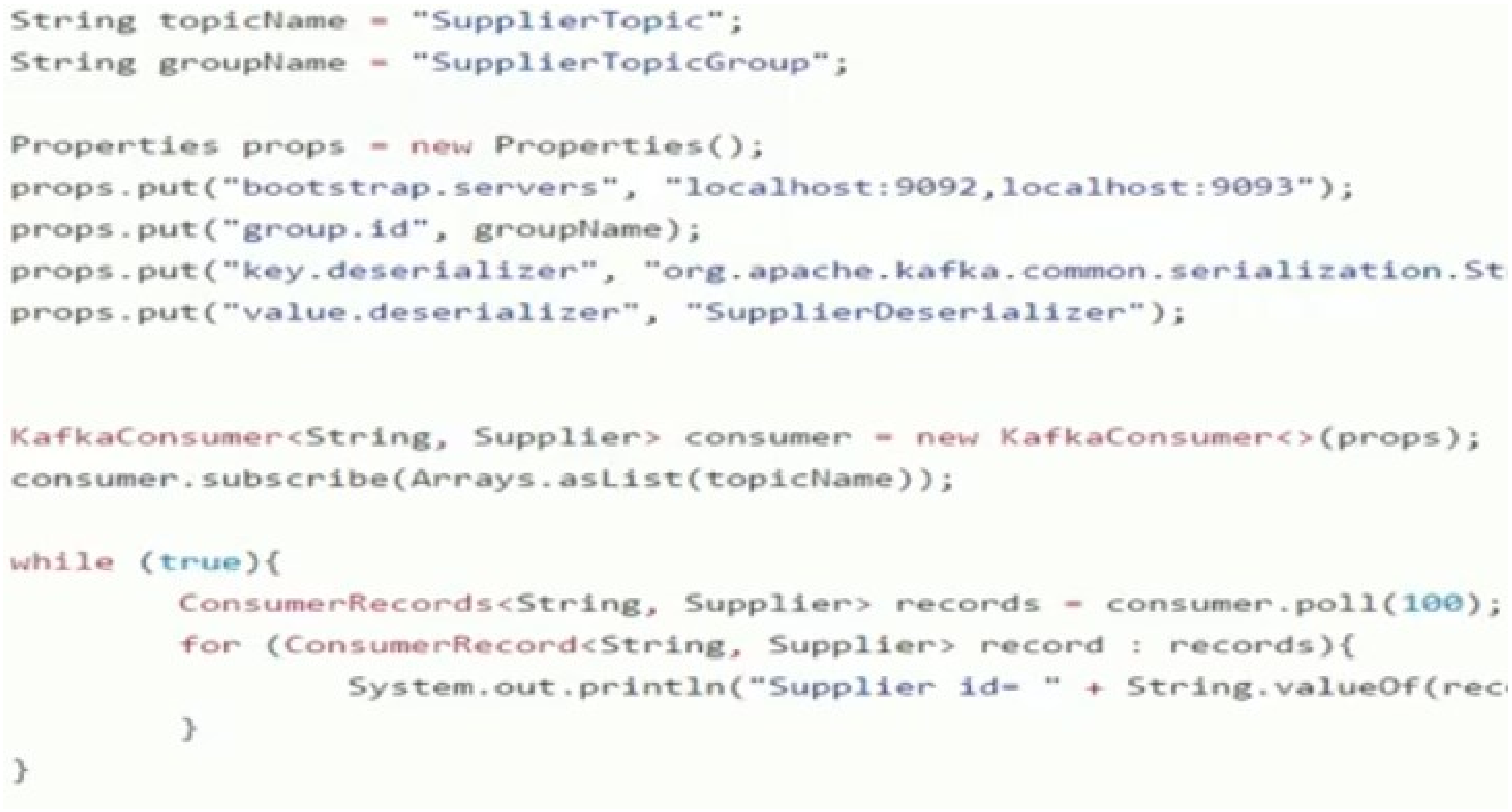
Consumers

- Consumer: subscribes to one or more topics
- Consumer group
- One consumer group at one topic:
  - at any point in time, each consumer owns exclusively a balanced subset of all partitions
  - (n consumers) – (n partitions) = (idle consumers)
- Record of a topic sent to:
  - one consumer only of each consumer group
- Consumers can enter and leave cons groups

Consumers


Kafka Cluster
Server 1
Server 2
P0
P3
P1
P2
C1
C2
C3
C4
C5
C6
Consumer Group A
Consumer Group B

Consumers



```
String topicName = "SupplierTopic";
String groupName = "SupplierTopicGroup";

Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092,localhost:9093");
props.put("group.id", groupName);
props.put("key.deserializer", "org.apache.kafka.common.serialization.St
props.put("value.deserializer", "SupplierDeserializer");


KafkaConsumer<String, Supplier> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList(topicName));

while (true){
        ConsumerRecords<String, Supplier> records = consumer.poll(100);
        for (ConsumerRecord<String, Supplier> record : records){
                System.out.println("Supplier id= " + String.valueOf(rec
        }
}
```

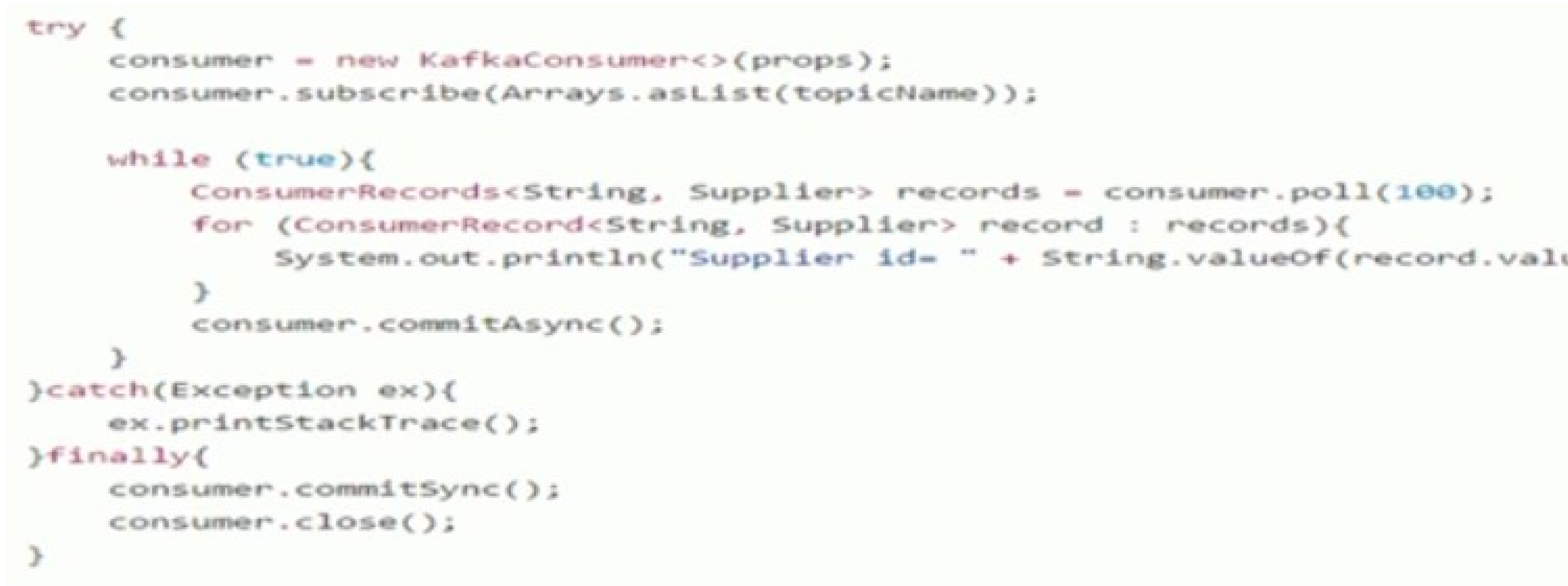
KAFKA Offsets

- Current offset:
  - next record to be read by a consumer
  - Used by brokers to send records
- Committed offset:
  - Processed records by a consumer
  - Critical in partitioning rebalancing
  - Used to set processed records
  - Manual and auto commit
  - Properties:
    - enable.auto.commit (true)
    - auto.commit.interval.ms (5 sec) – be careful -> reprocessing

KAFKA Offsets

- Manual:
  - Commit sync: wait for ack
  - Commit async:
    - no wait for ack
    - Can be defined in a callback

KAFKA Offsets

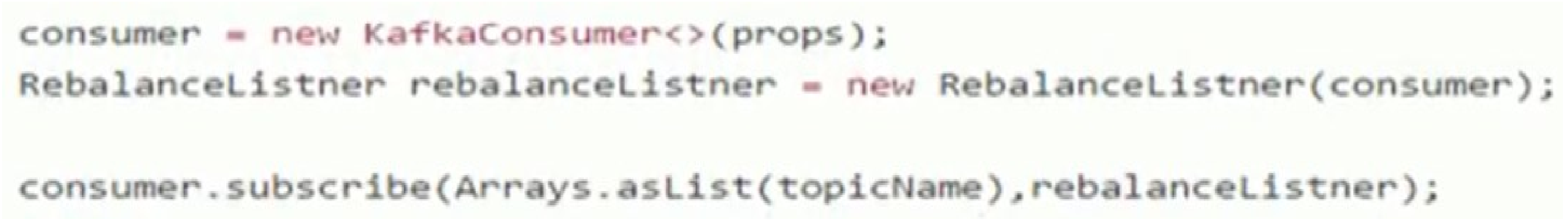


```
try {
    consumer = new KafkaConsumer<>(props);
    consumer.subscribe(Arrays.asList(topicName));

    while (true){
        ConsumerRecords<String, Supplier> records = consumer.poll(100);
        for (ConsumerRecord<String, Supplier> record : records){
            System.out.println("Supplier id= " + String.valueOf(record.valu
        }
        consumer.commitAsync();
    }
}catch(Exception ex){
    ex.printStackTrace();
}finally{
    consumer.commitSync();
    consumer.close();
}
```

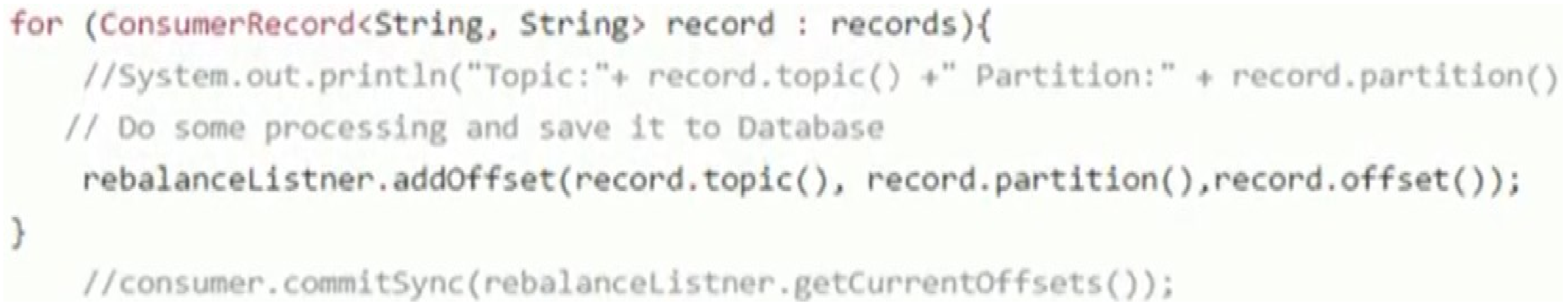
Consumer Group Rebalance

- And if processing records take too long?
  - Are you dead?: a rebalance of consumers may take place
  - Two things to know:
    - How to commit a particular offset?
    - How to know that a rebalance is triggered?
  - ConsumerRebalanceListener interface
    - addOffset
    - onPartitionsRevoked: … do commit
    - onPartitionsAssigned

KAFKA Offsets

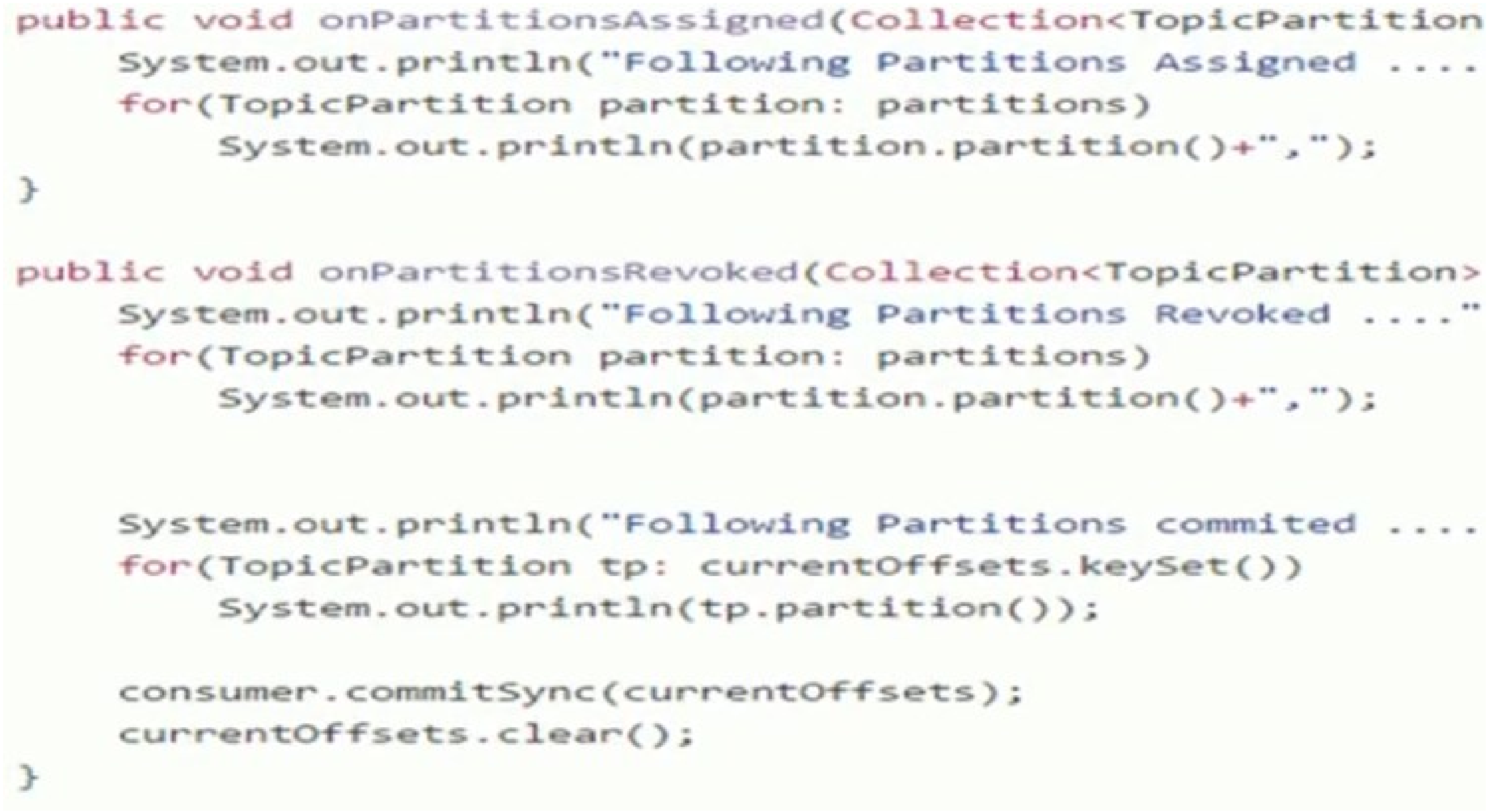


```
consumer = new KafkaConsumer<>(props);
RebalanceListner rebalanceListner = new RebalanceListner(consumer);

consumer.subscribe(Arrays.asList(topicName),rebalanceListner);
```



```
for (ConsumerRecord<String, String> record : records){
    //System.out.println("Topic:"+ record.topic() +" Partition:" + record.partition()
   // Do some processing and save it to Database
    rebalanceListner.addOffset(record.topic(), record.partition(),record.offset());
}
    //consumer.commitSync(rebalanceListner.getCurrentOffsets());
```

KAFKA Offsets: Listener



```
public void onPartitionsAssigned(Collection<TopicPartition
    System.out.println("Following Partitions Assigned ....
    for(TopicPartition partition: partitions)
        System.out.println(partition.partition()+",");
}

public void onPartitionsRevoked(Collection<TopicPartition>
    System.out.println("Following Partitions Revoked ...."
    for(TopicPartition partition: partitions)
        System.out.println(partition.partition()+",");


    System.out.println("Following Partitions commited ....
    for(TopicPartition tp: currentOffsets.keySet())
        System.out.println(tp.partition());

    consumer.commitSync(currentOffsets);
    currentOffsets.clear();
}
```

The End

Practical

- Tutorial (steps 1-7): https://kafka.apache.org/quickstart

Bibliography

https://kafka.apache.org/intro

https://www.youtube.com/watch?v=gg-VwXSRnmg&list=PLkz1SCf5iB4enAR00Z46JwY9GGkaS2NON:

- Videos: Inaugural till … Rebalance Listener

Kafka: The Definitive Guide. Neha Narkhede, Gwen Shapira & Todd Palino. O'Reilly. 2017.

https://www.youtube.com/watch?v=-DyWhcX3Dpc&list=PLa7VYi0yPIH2PelhRHoFR5iQgflg-y6JA&index=1