

极客大学 Java 进阶训练营

第 25 课

分布式消息——Kafka消息中间件



KimmKing

Apache Dubbo/ShardingSphere PMC

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

目录

1. Kafka概念和入门
2. Kafka的简单使用*
3. Kafka的集群配置*
4. Kafka的高级特性*
5. 总结回顾与作业实践

第 25 课 1. Kafka概念与入门

什么是Kafka

Kafka 是一个消息系统，由 LinkedIn 于2011年设计开发，用作 LinkedIn 的活动流（Activity Stream）和运营数据处理管道（Pipeline）的基础。

Kafka 是一种分布式的，基于发布 / 订阅的消息系统。主要设计目标如下：

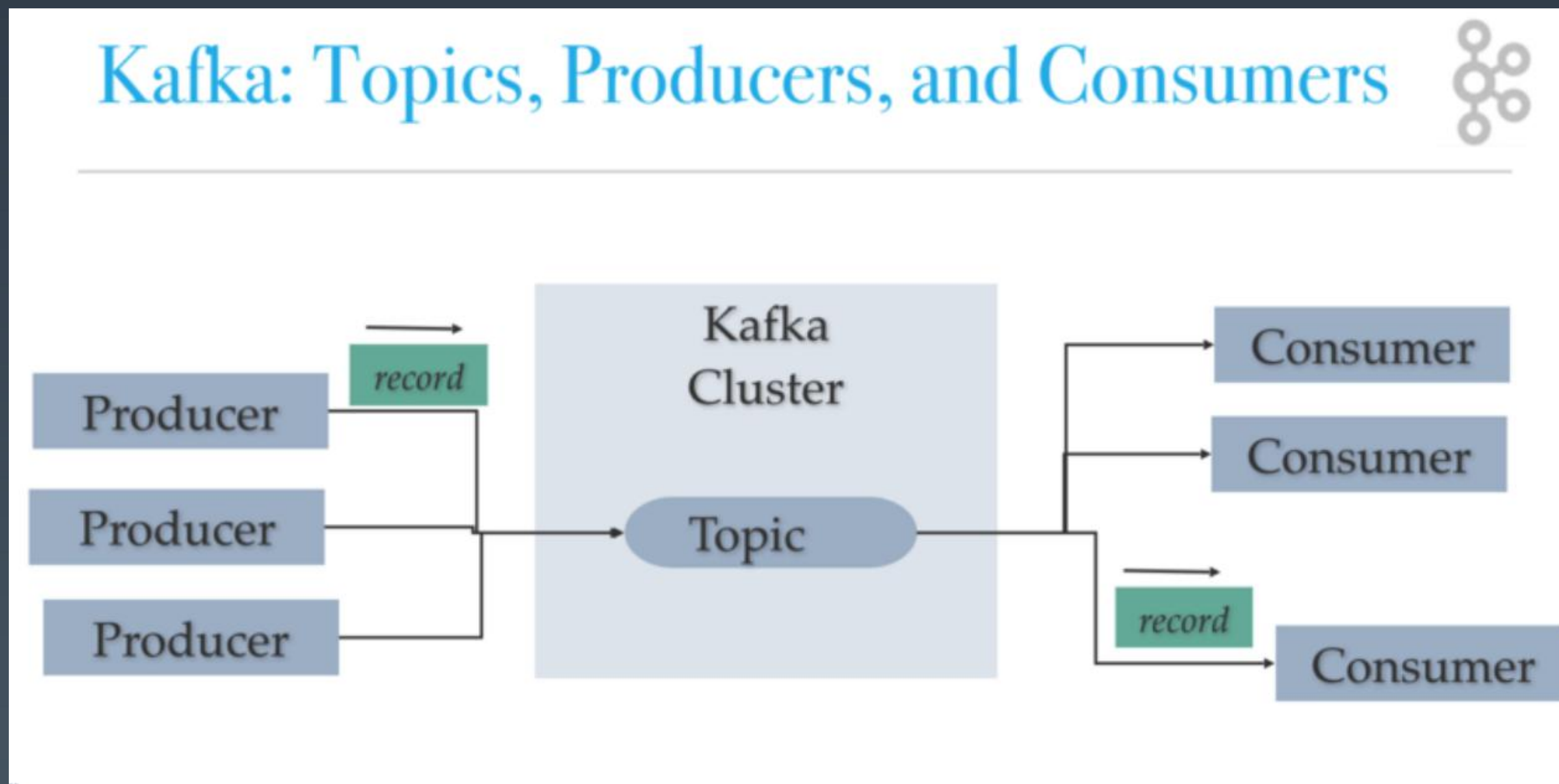
1. 以时间复杂度为 $O(1)$ 的方式提供消息持久化能力，即使对 TB 级以上数据也能保证常数时间复杂度的访问性能。
2. 高吞吐率。即使在非常廉价的商用机器上也能做到单机支持每秒 100K 条以上消息的传输。
3. 支持 Kafka Server 间的消息分区，及分布式消费，同时保证每个 Partition 内的消息顺序传输。
4. 同时支持离线数据处理和实时数据处理。
5. Scale out：支持在线水平扩展。

Kafka的基本概念

1. Broker: Kafka 集群包含一个或多个服务器，这种服务器被称为 broker。
2. Topic: 每条发布到 Kafka 集群的消息都有一个类别，这个类别被称为 Topic。
(物理上不同 Topic 的消息分开存储，逻辑上一个 Topic 的消息虽然保存于一个或多个 broker 上，但用户只需指定消息的 Topic 即可生产或消费数据而不必关心数据存于何处)。
3. Partition: Partition 是物理上的概念，每个 Topic 包含一个或多个 Partition。
4. Producer: 负责发布消息到 Kafka broker。
5. Consumer: 消息消费者，向 Kafka broker 读取消息的客户端。
6. Consumer Group: 每个 Consumer 属于一个特定的 Consumer Group (可为每个 Consumer 指定 group name, 若不指定 group name 则属于默认的 group)。

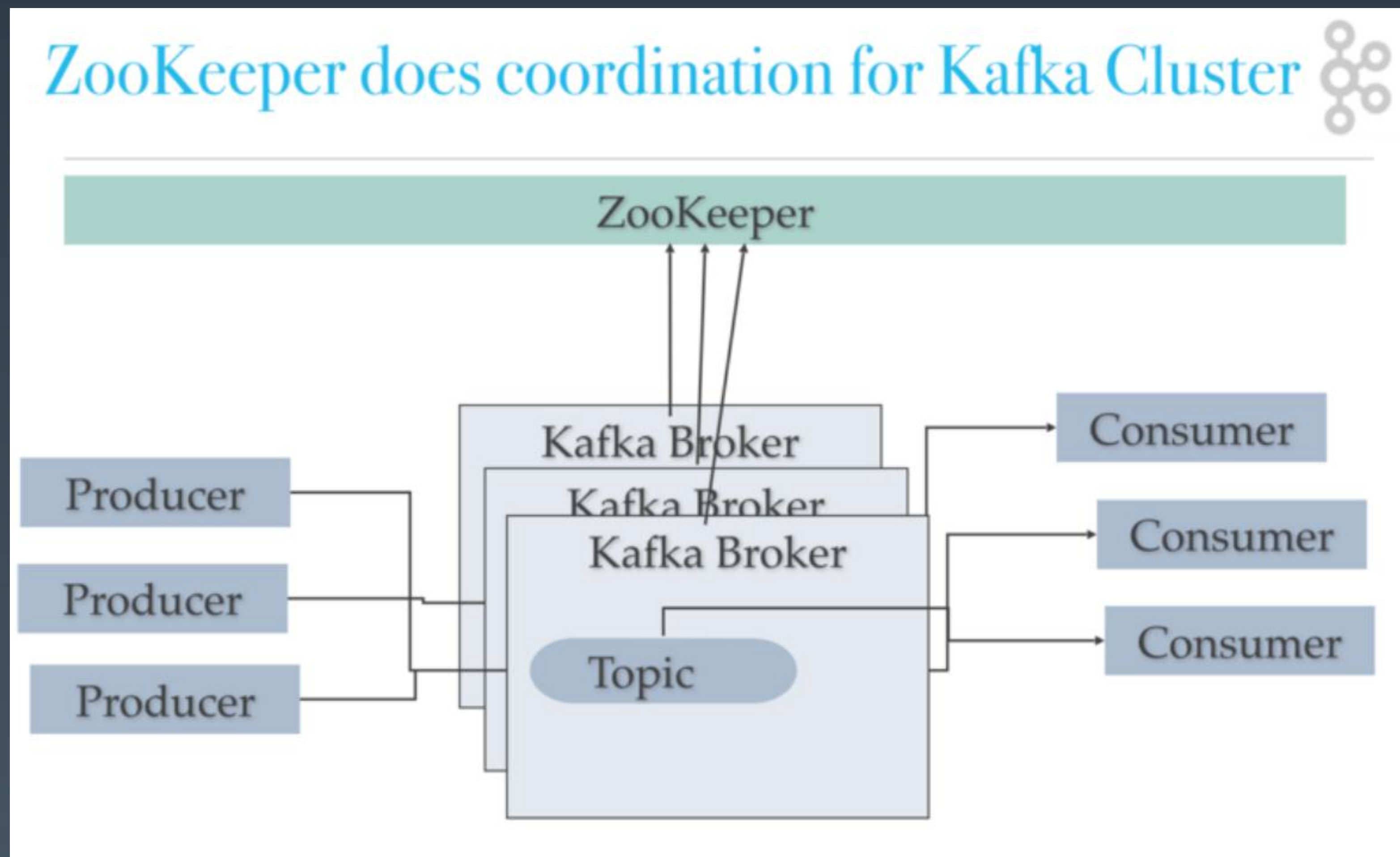
单机部署结构

Kafka 单机消息处理



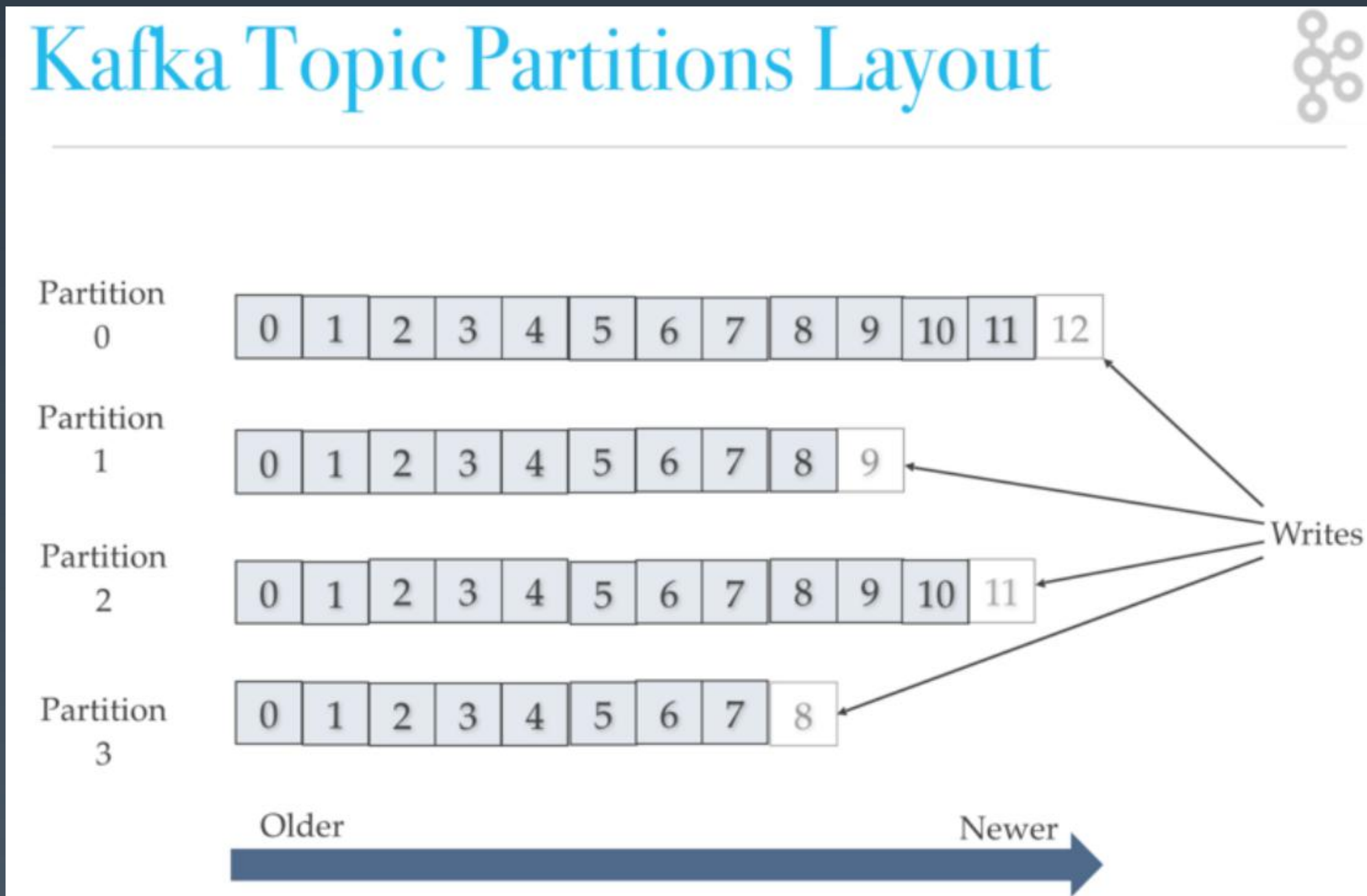
集群部署结构

Kafka 集群消息处理



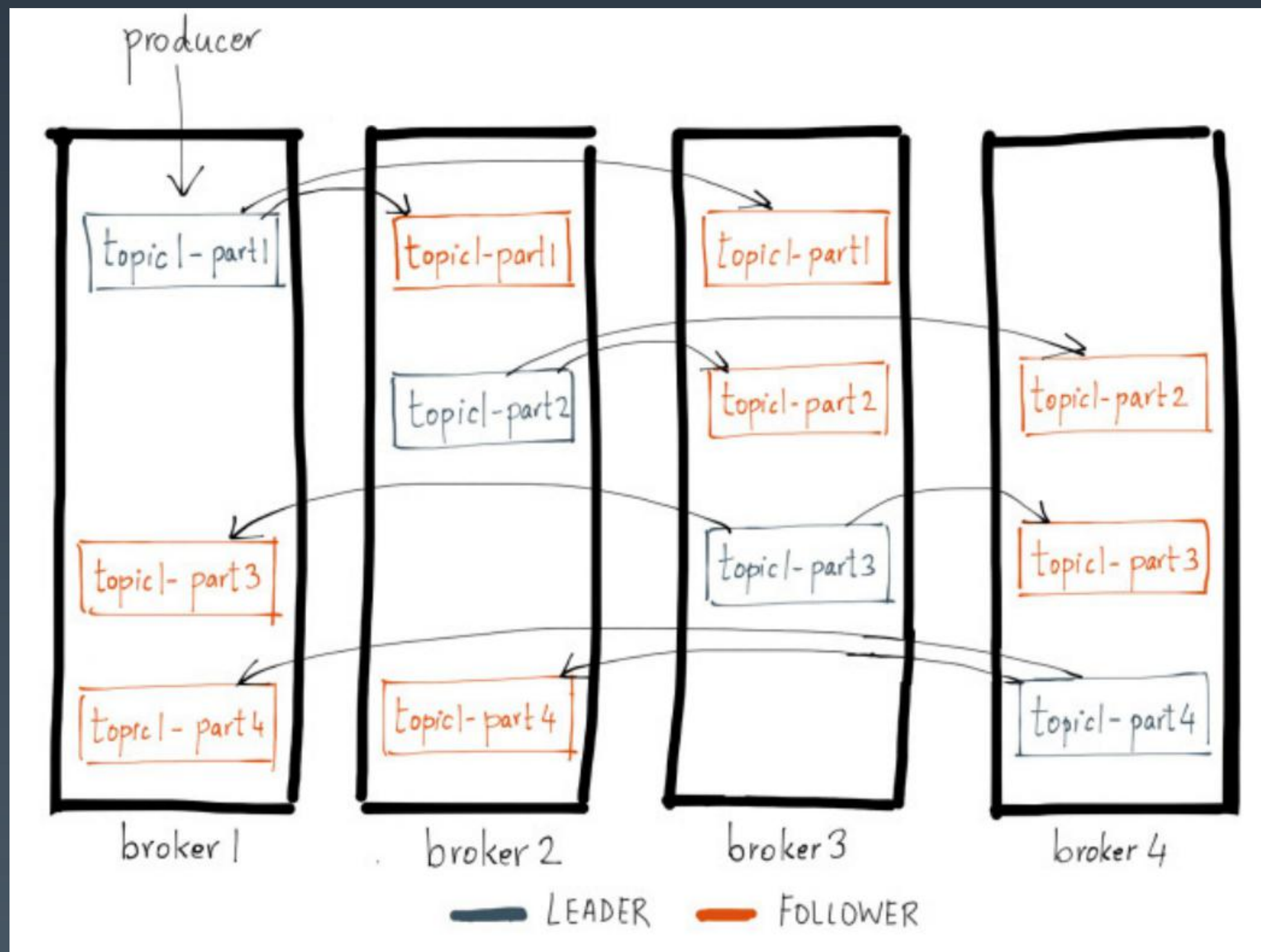
Topic和Partition

多Partition支持水平扩展和并行处理，顺序写入提升吞吐性能



Partition和Replica

每个partition可以通过副本因子添加多个副本



Topic特性

1. 通过partition增加可扩展性
2. 通过顺序写入达到高吞吐
3. 多副本增加容错性

第 25 课 2. Kafka的简单使用

单机安装部署

1、kafka安装

<http://kafka.apache.org/downloads>

下载2.6.0或者2.7.0，解压。

2、启动kafka：

命令行下进入kafka目录

修改配置文件 `vim config/server.properties`

打开 `listeners=PLAINTEXT://localhost:9092`

`bin/zookeeper-server-start.sh config/zookeeper.properties`

`bin/kafka-server-start.sh config/server.properties`

单机部署测试

3、命令行操作Kafka

```
bin/kafka-topics.sh --zookeeper localhost:2181 --list
```

```
bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic testk --partitions 4 --replication-factor 1
```

```
bin/kafka-topics.sh --zookeeper localhost:2181 --describe --topic testk
```

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --from-beginning --topic testk
```

```
bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic testk
```

4、简单性能测试

```
bin/kafka-producer-perf-test.sh --topic testk --num-records 100000 --record-size 1000 --throughput 2000 --producer-props bootstrap.servers=localhost:9092
```

```
bin/kafka-consumer-perf-test.sh --bootstrap-server localhost:9092 --topic testk --fetch-size 1048576 --messages 100000 --threads 1
```


Java中使用kafka发送接收消息

基于Kafka Client发送和接收消息——极简生产者

```
//kafka producer 配置
Properties props = new Properties();
props.setProperty("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.setProperty("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.setProperty("bootstrap.servers", "localhost:9092");

//创建kafka producer
KafkaProducer producer = new KafkaProducer(props);
for (long i = 0; i < 10; i++) {
    Order data = new Order();
    data.setAmount(new BigDecimal(1));
    data.setId(i);
    data.setType(1);
    //构造record
    ProducerRecord record = new ProducerRecord(topic: "demo-source", JSON.toJSONString(data));
    //发送record
    producer.send(record);
}
//关闭producer
producer.close();
```

Java中使用kafka发送接收消息

基于Kafka Client发送和接收消息——极简消费者

```
//kafka 配置
Properties props = new Properties();
props.setProperty("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.setProperty("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.setProperty("bootstrap.servers", "localhost:9092");
//kafka consumer group 配置
props.setProperty("group.id", "group1");

//构建Kafka consumer
KafkaConsumer consumer = new KafkaConsumer(props);

//订阅topic
consumer.subscribe(Arrays.asList("demo-source"));

while (true) {
    //拉取数据
    ConsumerRecords poll = consumer.poll(Duration.ofMillis(100));
    poll.forEach(o -> {
        ConsumerRecord<String, String> record = (ConsumerRecord) o;
        Order order = JSON.parseObject(record.value(), Order.class);
        System.out.println("order = " + order);
    });
}
```

第 25 课 3. Kafka的集群配置

集群安装部署01

1、现在我们来部署3个节点的集群，首先准备3个配置文件(kafka900x.properties)

```
1 broker.id=1  ## 三个文件分别改为1, 2, 3
2 num.network.threads=3
3 num.io.threads=8
4 socket.send.buffer.bytes=102400
5 socket.receive.buffer.bytes=102400
6 socket.request.max.bytes=104857600
7 log.dirs=/tmp/kafka/kafka-logs1  ## 三个文件分别改为logs1, logs2, logs3
8 num.partitions=1
9 num.recovery.threads.per.data.dir=1
10 offsets.topic.replication.factor=1
11 transaction.state.log.replication.factor=1
12 transaction.state.log.min.isr=1
13 log.retention.hours=168
14 log.segment.bytes=1073741824
15 log.retention.check.interval.ms=300000
16 zookeeper.connection.timeout.ms=6000000
17 delete.topic.enable=true
18 group.initial.rebalance.delay.ms=0
19 message.max.bytes=5000000
20 replica.fetch.max.bytes=5000000
21 listeners=PLAINTEXT://localhost:9001  ## 三个文件分别改为9001, 9002, 9003
22 broker.list=localhost:9001,localhost:9002,localhost:9003
23 zookeeper.connect=localhost:2181
```

集群安装部署02

2、清理掉zk上的所有数据，可以删除zk的本地文件或者用ZooInspector操作

3、启动3个kafka：

三个命令行下进入kafka目录，分别执行

```
./bin/kafka-server-start.sh kafka9001.properties
```

```
./bin/kafka-server-start.sh kafka9002.properties
```

```
./bin/kafka-server-start.sh kafka9003.properties
```

完成启动操作。

集群安装部署03

4、执行操作测试

创建带有副本的topic:

```
bin/kafka-topics.sh --zookeeper localhost:2181 --create --topic test32 --partitions 3 --replication-factor 2
```

```
bin/kafka-console-producer.sh --bootstrap-server localhost:9003 --topic test32
```

```
bin/kafka-console-consumer.sh --bootstrap-server localhost:9001 --topic test32 --from-beginning
```

执行性能测试:

```
bin/kafka-producer-perf-test.sh --topic test32 --num-records 100000 --record-size 1000 --throughput 2000 --producer-props bootstrap.servers=localhost:9002
```

```
bin/kafka-consumer-perf-test.sh --bootstrap-server localhost:9002 --topic test32 --fetch-size 1048576 --messages 100000 --threads 1
```


集群与多副本的说明

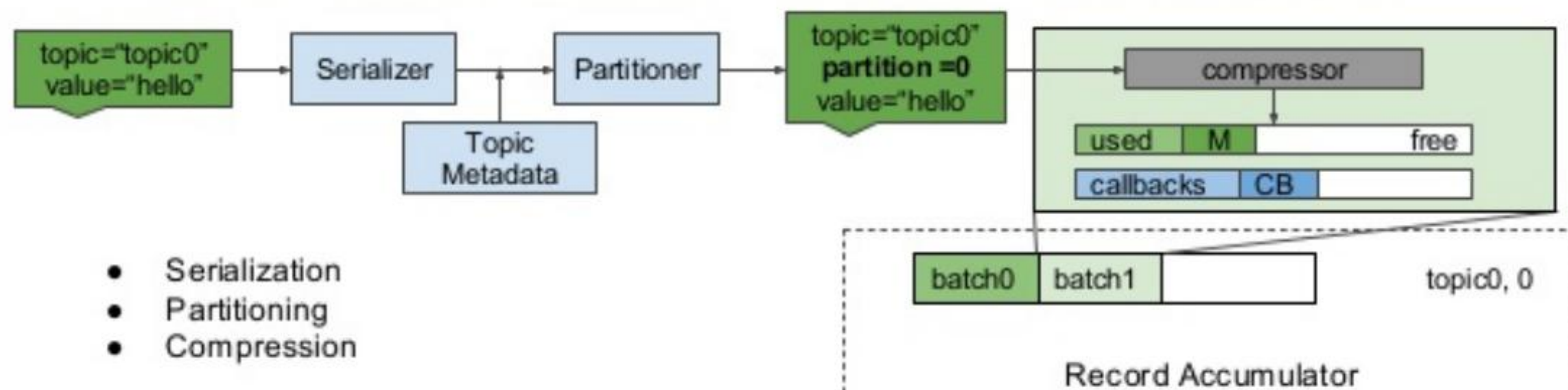
- 1、ISR: In-Sync Replica
- 2、Rebalance: broker和consumer group的rebalance
- 3、热点分区: 需要重新平衡

第 25 课 4. Kafka的高级特性

生产者-执行步骤

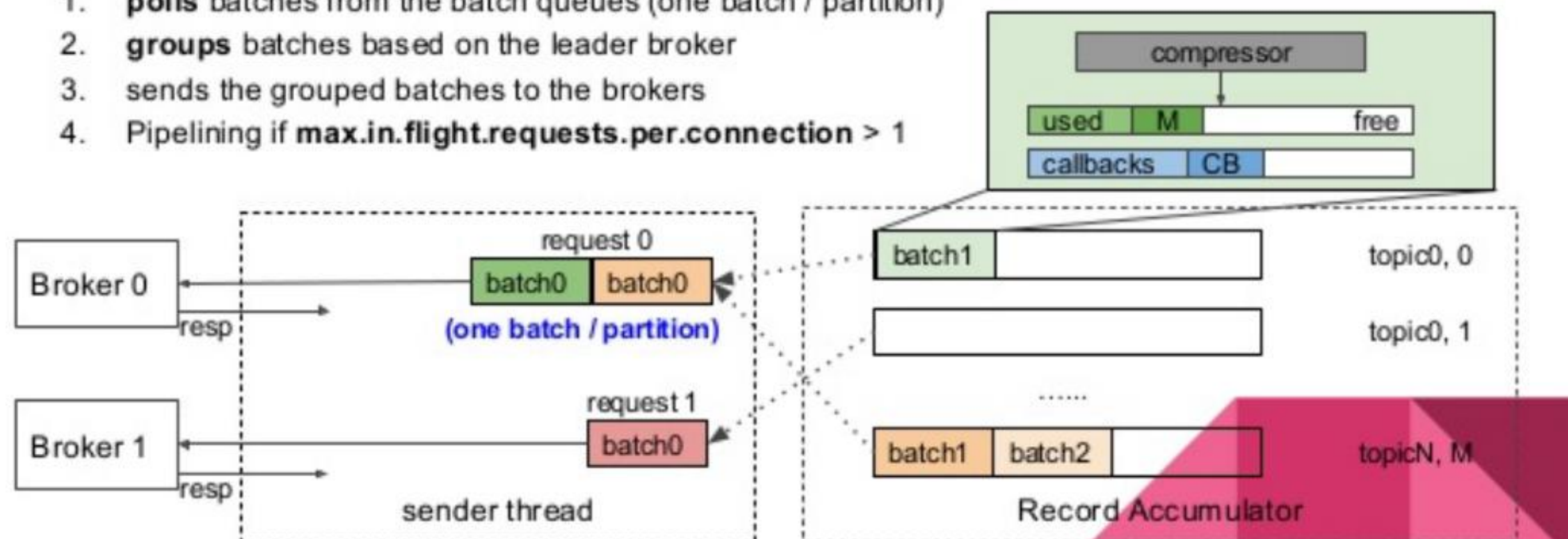
客户端实现序列化，分区，压缩操作

User: `producer.send(new ProducerRecord("topic0", "hello"), callback);`



Sender:

1. **polls** batches from the batch queues (one batch / partition)
2. **groups** batches based on the leader broker
3. sends the grouped batches to the brokers
4. Pipelining if `max.in.flight.requests.per.connection > 1`

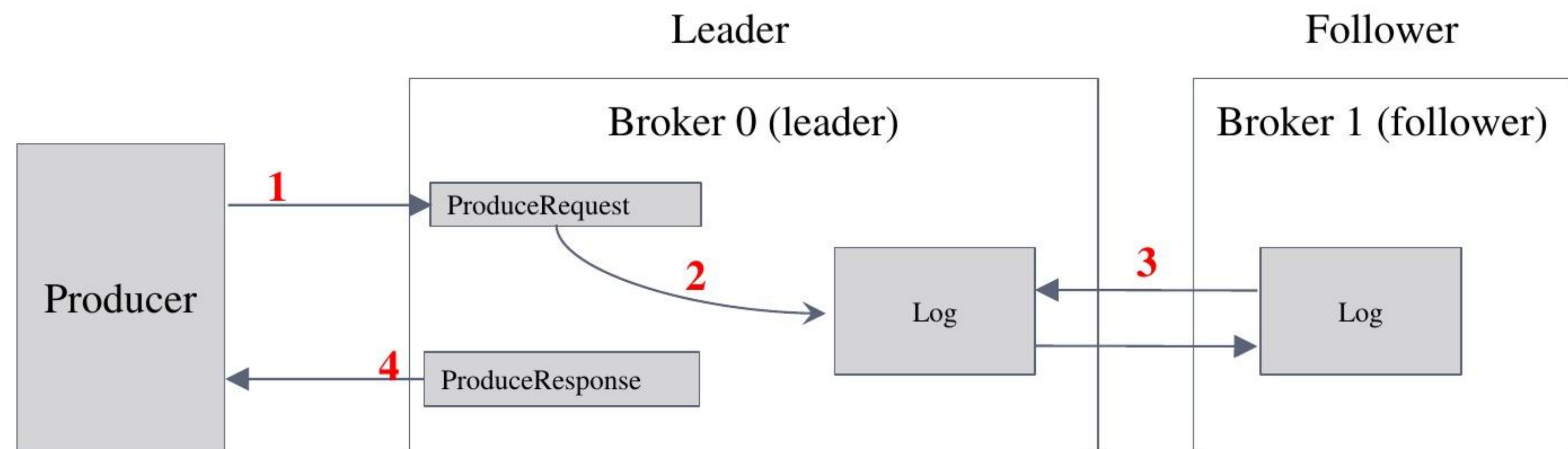


生产者-确认模式

ack=0 : 只发送不管有没有写入到broker

ack=1: 写入到leader就认为成功

ack=-1/all: 写入到最小的副本数则认为成功



1. [Network] Send ProduceRequest
2. [Broker] Append messages to the leader's log
- 3. [Broker] Replication (before sending the response)**
4. [Broker] ProduceResponse

生产者特性-同步发送

同步发送

```
KafkaProducer kafkaProducer = new KafkaProducer(pro);  
ProducerRecord record = new ProducerRecord("topic", "key", "value");  
Future future = kafkaProducer.send(record);  
//同步发送方法1  
Object o = future.get();  
  
//同步发送方法2  
kafkaProducer.flush();
```


生产者特性-异步发送

异步发送

```
pro.put("linger.ms", "1");
```

```
pro.put("batch.size", "10240");
```

```
KafkaProducer kafkaProducer = new KafkaProducer(pro);
```

```
ProducerRecord record = new ProducerRecord("topic", "key", "value");
```

```
Future future = kafkaProducer.send(record);
```

```
//异步发送方法1
```

```
kafkaProducer.send(record, (metadata, exception) -> {
```

```
    if (exception == null) System.out.println("record = " + record);
```

```
});
```

```
//异步发送方法2
```

```
kafkaProducer.send(record);
```


生产者特性-顺序保证

顺序保证

```
pro.put("max.in.flight.requests.per.connection", "1");
```

```
KafkaProducer kafkaProducer = new KafkaProducer(pro);
```

```
ProducerRecord record = new ProducerRecord("topic", "key", "value");
```

```
Future future = kafkaProducer.send(record);
```

//同步发送

```
kafkaProducer.send(record);
```

```
kafkaProducer.flush();
```

生产者特性-消息可靠性传递

```
pro.put("enable.idempotence","true"); // 此时就会默认把acks设置为all
```

```
pro.put("transaction.id","tx0001"); //思考一下，什么是消息的事务？？？
```

```
try {
```

```
    kafkaProducer.beginTransaction();
```

```
    ProducerRecord record = new ProducerRecord("topic", "key", "value");
```

```
    for (int i = 0; i < 100; i++) {
```

```
        kafkaProducer.send(record, (metadata, exception) -> {
```

```
            if (exception != null) {
```

```
                kafkaProducer.abortTransaction();
```

```
                throw new KafkaException(exception.getMessage() + " , data: " + record);
```

```
            } }); }
```

```
    kafkaProducer.commitTransaction();
```

```
} catch (Throwable e) {
```

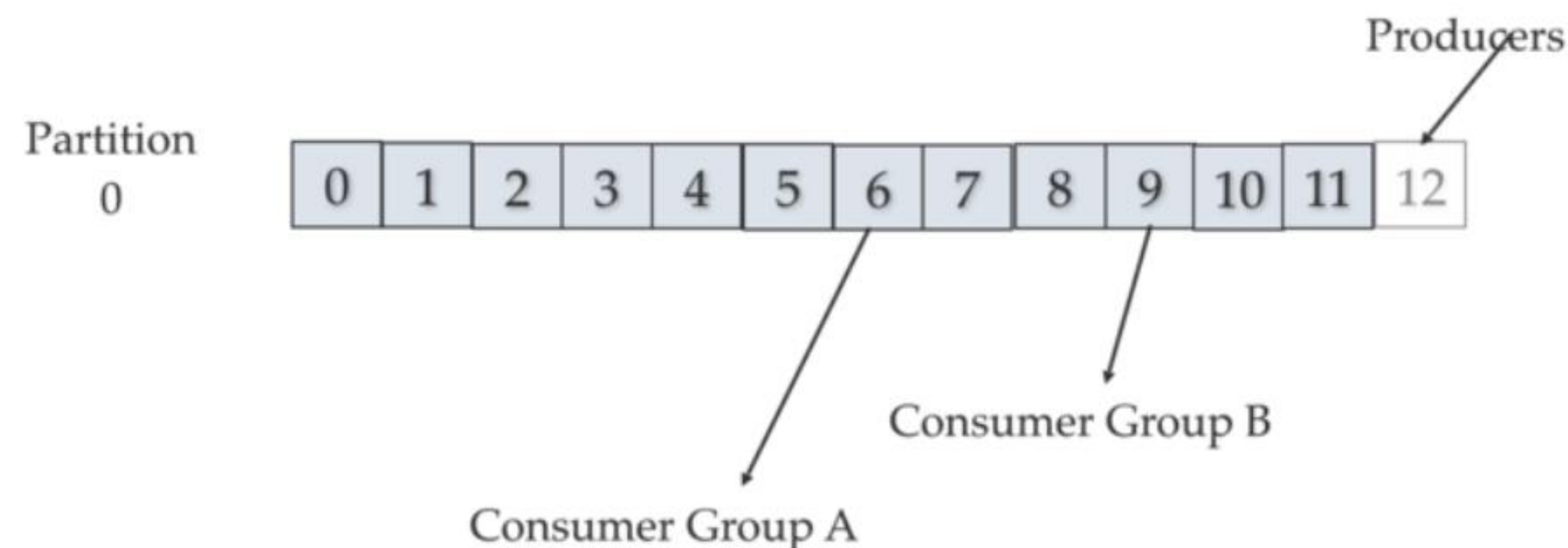
```
    kafkaProducer.abortTransaction();
```

```
}
```

消费者-Consumer Group

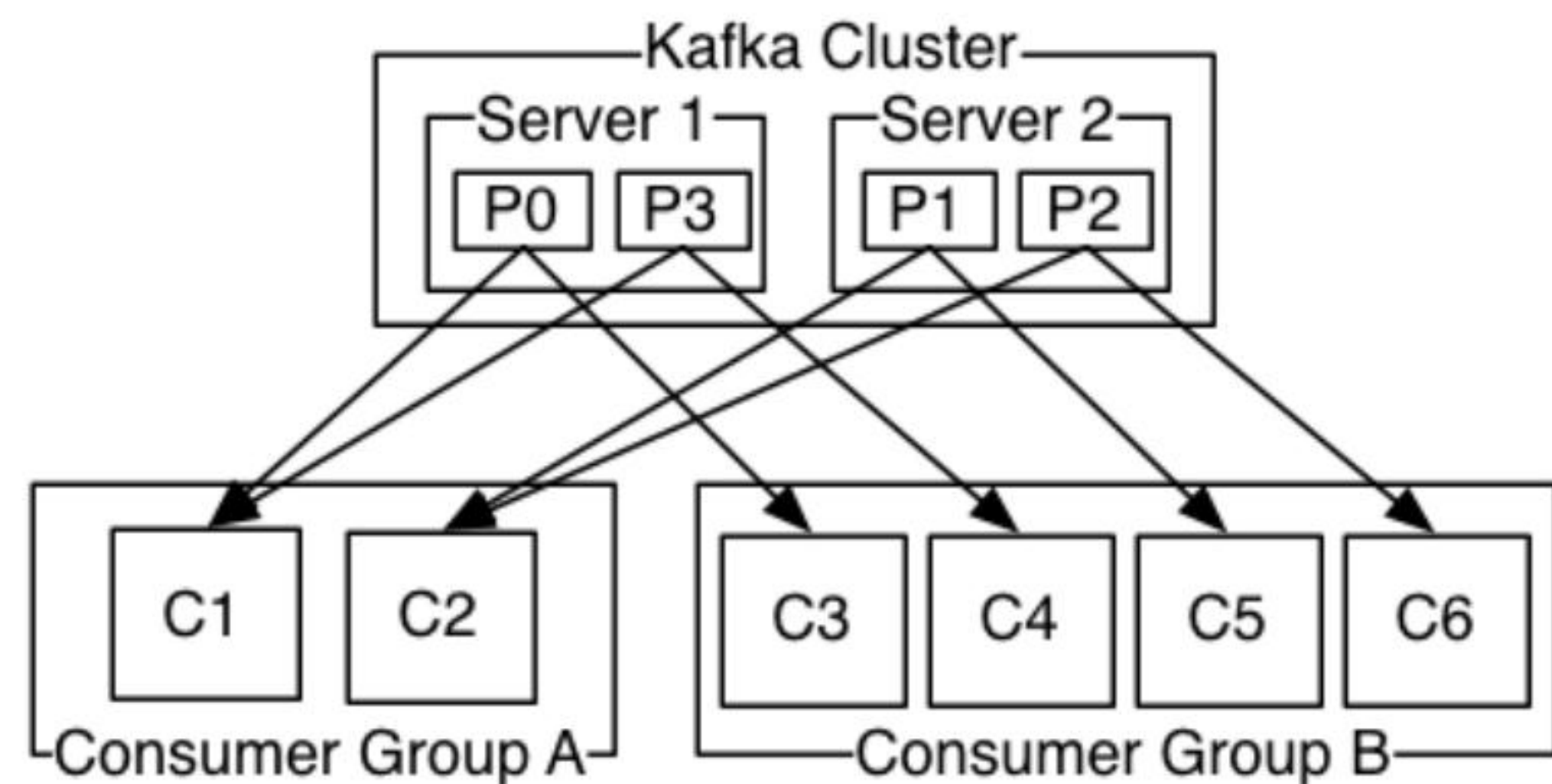
消费者与Partition对应关系，如果4个partition，3个消费者怎么办？5个呢？

Kafka Consumer Groups



Consumers remember offset where they left off.

Consumers groups each have their own offset per partition.



消费者特性-Offset同步提交

```
props.put("enable.auto.commit", "false");

while (true) {
    //拉取数据
    ConsumerRecords poll = consumer.poll(Duration.ofMillis(100));
    poll.forEach(o -> {
        ConsumerRecord<String, String> record = (ConsumerRecord) o;
        Order order = JSON.parseObject(record.value(), Order.class);
        System.out.println("order = " + order);
    });
    consumer.commitSync();
}
```

消费者特性-Offset异步提交

```
props.put("enable.auto.commit", "false");

while (true) {
    //拉取数据
    ConsumerRecords poll = consumer.poll(Duration.ofMillis(100));
    poll.forEach(o -> {
        ConsumerRecord<String, String> record = (ConsumerRecord) o;
        Order order = JSON.parseObject(record.value(), Order.class);
        System.out.println("order = " + order);
    });
    consumer.commitAsync();
}
```

消费者特性-Offset自动提交

```
props.put("enable.auto.commit", "true");  
props.put("auto.commit.interval.ms", "5000");  
  
while (true) {  
    //拉取数据  
    ConsumerRecords poll = consumer.poll(Duration.ofMillis(100));  
    poll.forEach(o -> {  
        ConsumerRecord<String, String> record = (ConsumerRecord) o;  
        Order order = JSON.parseObject(record.value(), Order.class);  
        System.out.println("order = " + order);  
    });  
}
```


消费者特性-Offset Seek

```
props.put("enable.auto.commit", "true");  
//订阅topic  
consumer.subscribe(Arrays.asList("demo-source"), new ConsumerRebalanceListener() {  
    @Override  
    public void onPartitionsRevoked(Collection<TopicPartition> partitions) {  
        commitOffsetToDB();  
    }  
    @Override  
    public void onPartitionsAssigned(Collection<TopicPartition> partitions) {  
        partitions.forEach(topicPartition -> consumer.seek(topicPartition,  
getOffsetFromDB(topicPartition)));  
    }  
});
```

消费者特性-Offset Seek

```
while (true) {  
    //拉取数据  
    ConsumerRecords poll = consumer.poll(Duration.ofMillis(100));  
    poll.forEach(o -> {  
        ConsumerRecord<String, String> record = (ConsumerRecord) o;  
        processRecord(record);  
        saveRecordAndOffsetInDB(record, record.offset());  
    });  
}
```

第 25 课总结回顾

Kafka入门

Kafka简单使用

Kafka集群部署

Kafka高级特性

第 25 课作业实践

- 1、（**必做**）搭建一个3节点Kafka集群，测试功能和性能；实现spring kafka下对kafka集群的操作，将代码提交到github。
- 2、（选做）安装kafka-manager工具，监控kafka集群状态。
- 3、（挑战☆）演练本课提及的各种生产者和消费者特性。
- 4、（挑战☆☆☆）Kafka金融领域实战：在证券或者外汇、数字货币类金融核心交易系统里，对于订单的处理，大概可以分为收单、定序、撮合、清算等步骤。其中我们一般可以用mq来实现订单定序，然后将订单发送给撮合模块。
 - 1) 收单：请实现一个订单的rest接口，能够接收一个订单Order对象；
 - 2) 定序：将Order对象写入到kafka集群的order.usd2cny队列，要求数据有序并且不丢失；
 - 3) 撮合：模拟撮合程序（不需要实现撮合逻辑），从kafka获取order数据，并打印订单信息，要求可重放，顺序消费，消息仅处理一次。

THANKS! |  极客大学