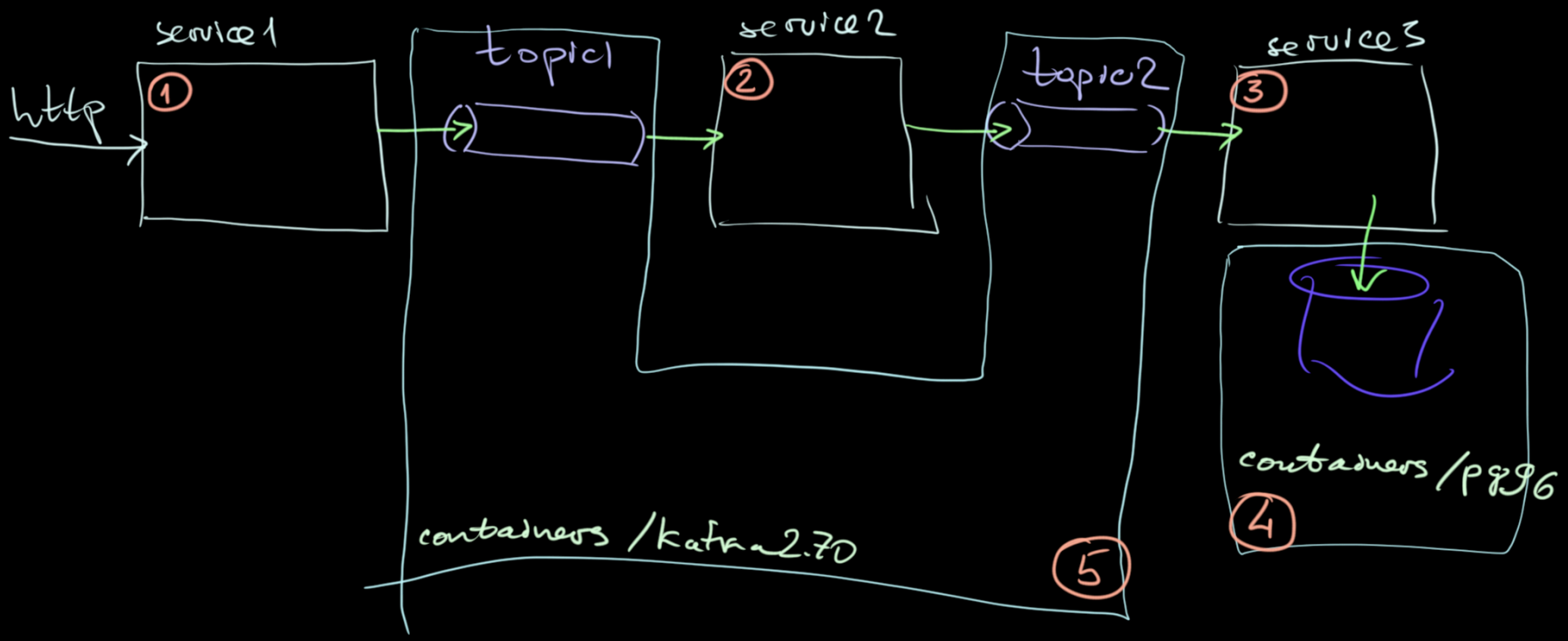
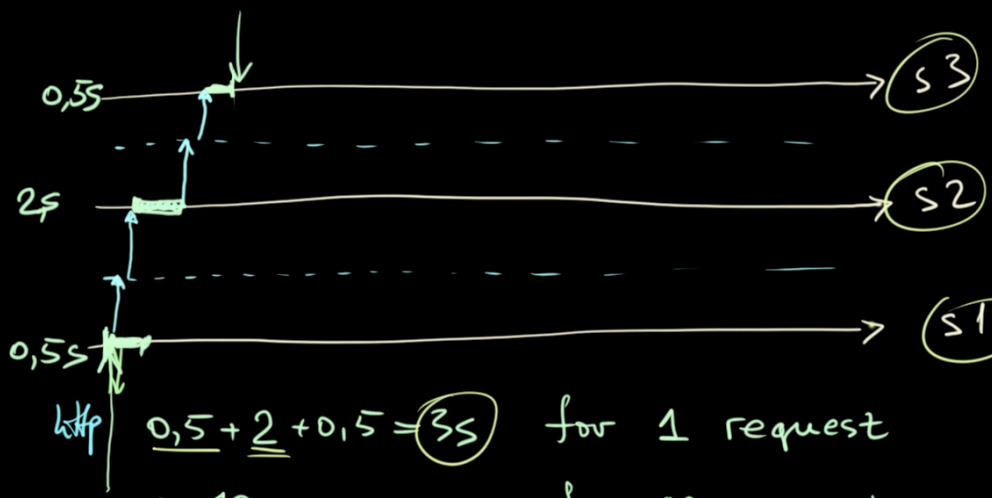
You are screen sharing Stop Share

gitegithub.com.alexr007/spring-kafka-postgres-docker.git

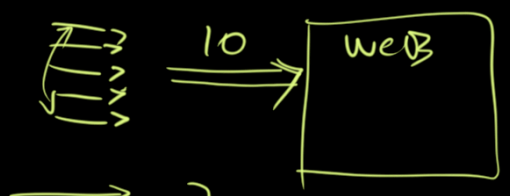




0,5 + 2 + 0,5 = 3s for 1 request

3 x 10 = 30s for 10 requests

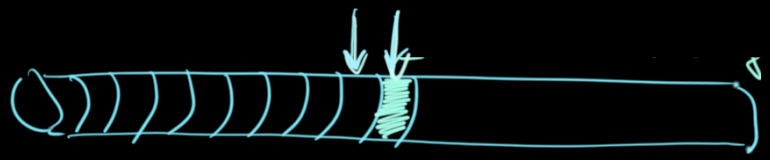
we can't process items in queue in parallel



we don't have any relation and ordering

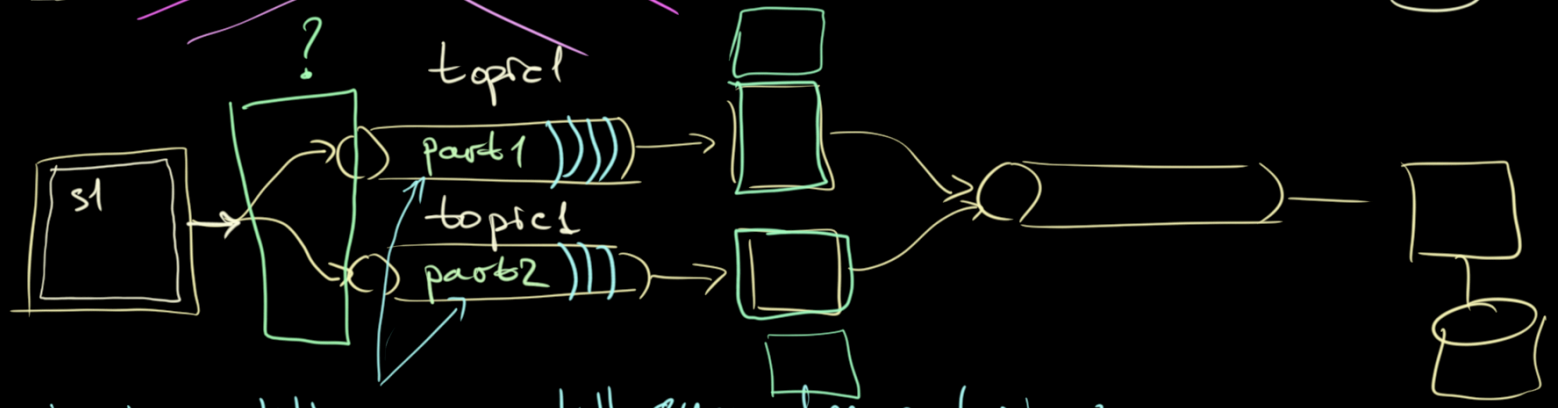
here is the problem we need to deal with

Kafka partitions



topic has a property --> # partitions
once topic is partitioned (>1)

1. Kafka guarantees ordering
2. How to do in parallel ?



inside partition we still guarantee ordering

2 partitions for 1 topic

1.
- partition 0 → S1
- partition 1 → S2

ok

2.
- partition 0 → S1
- partition 1 → S1

ok

3.
- partition 0 → S1
- partition 1 → S2
- → S3 (X)

Wrong

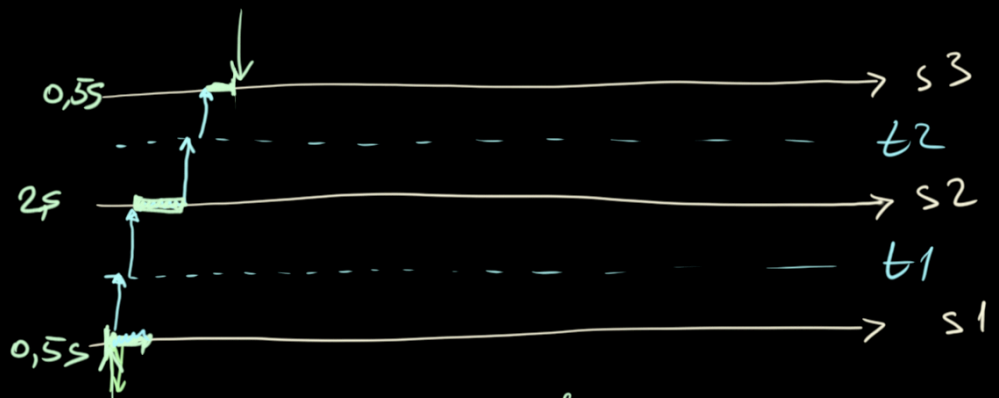
the # of partitions should be greater than # of listeners

Mapping between part and consumers

1. automatic
2. manual

```
@KafkaListener(id = "group1", topics = "topic1",
```

```
@KafkaListener(
    topicPartitions = @TopicPartition(topic = "topic1",
        partitionOffsets = {
            @PartitionOffset(partition = "0", initialOffset = "0"),
        }
    ),
```



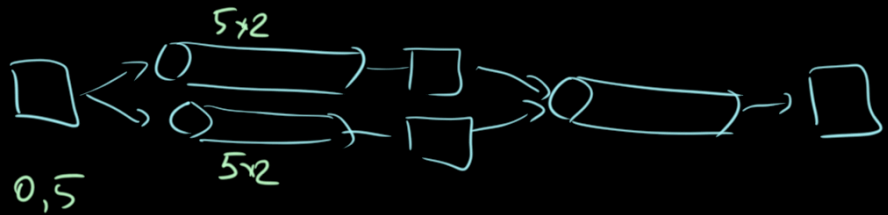
0,5 + 2 + 0,5 = 3s for 1 request

~~3 x 10 = 30s~~ for 10 requests 1 partition topic1 +1 instance for service 2

0,5 + 10 x 2 + 10 x 0,5 = 25.5 s

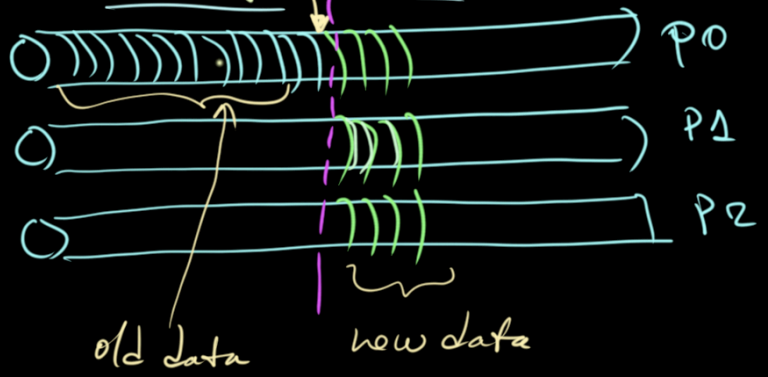
0,5 + 5 x 2 + 10 x 0,5 = 15,5 s

2 part topic 1 +2 instancy for service 2



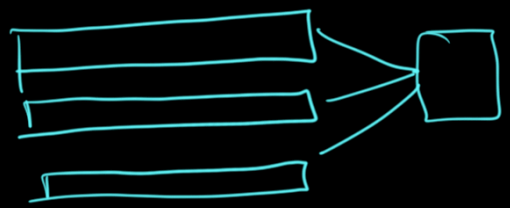
part = 1

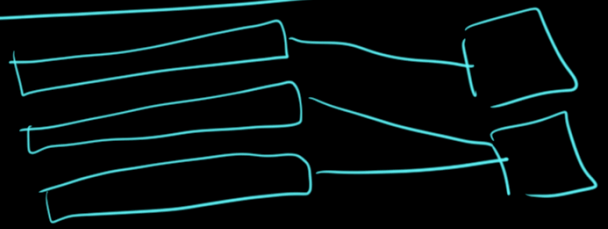
topic modify # parts=3



the rule of thumb

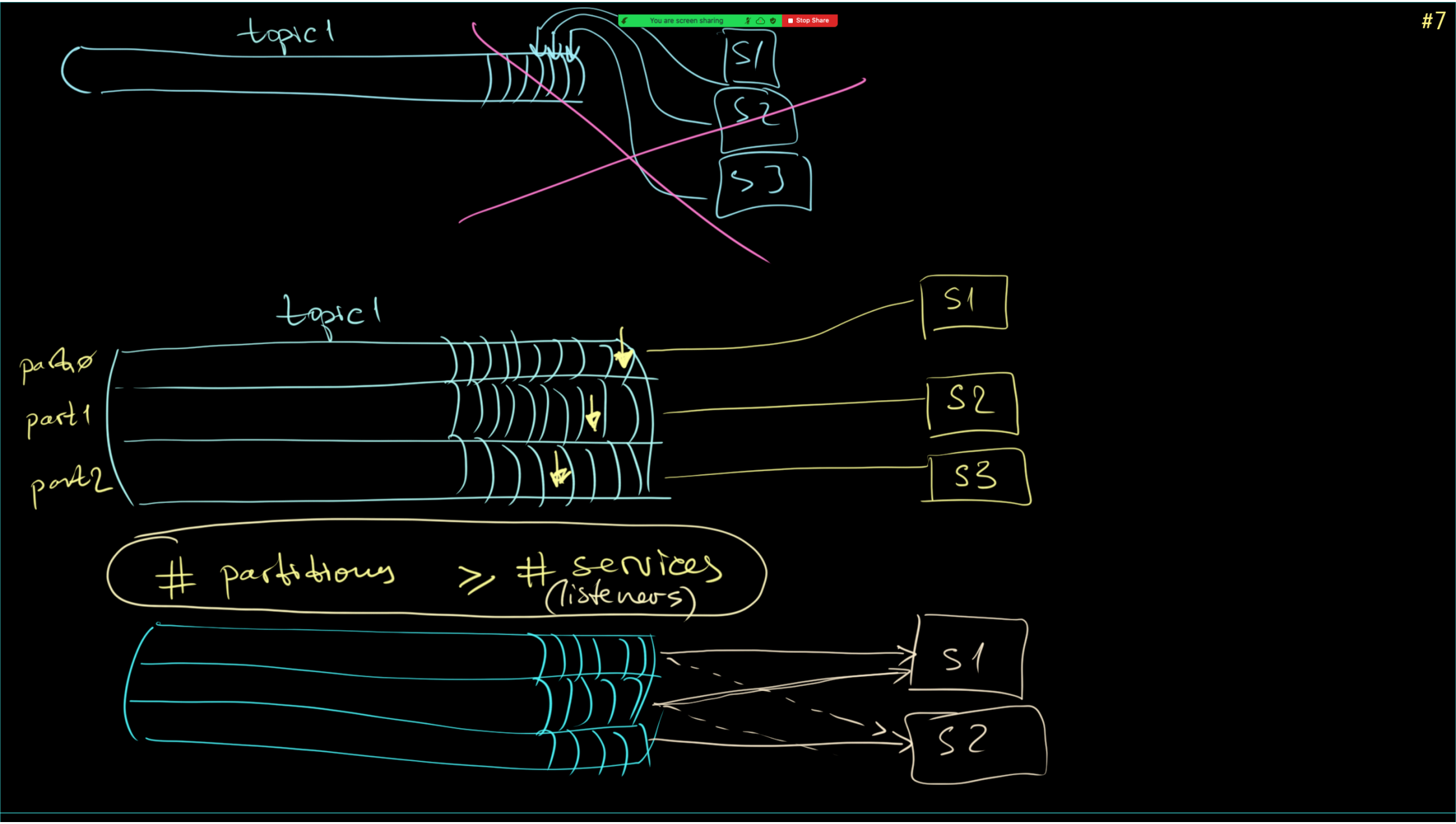
create new topic with at least (3) partitions

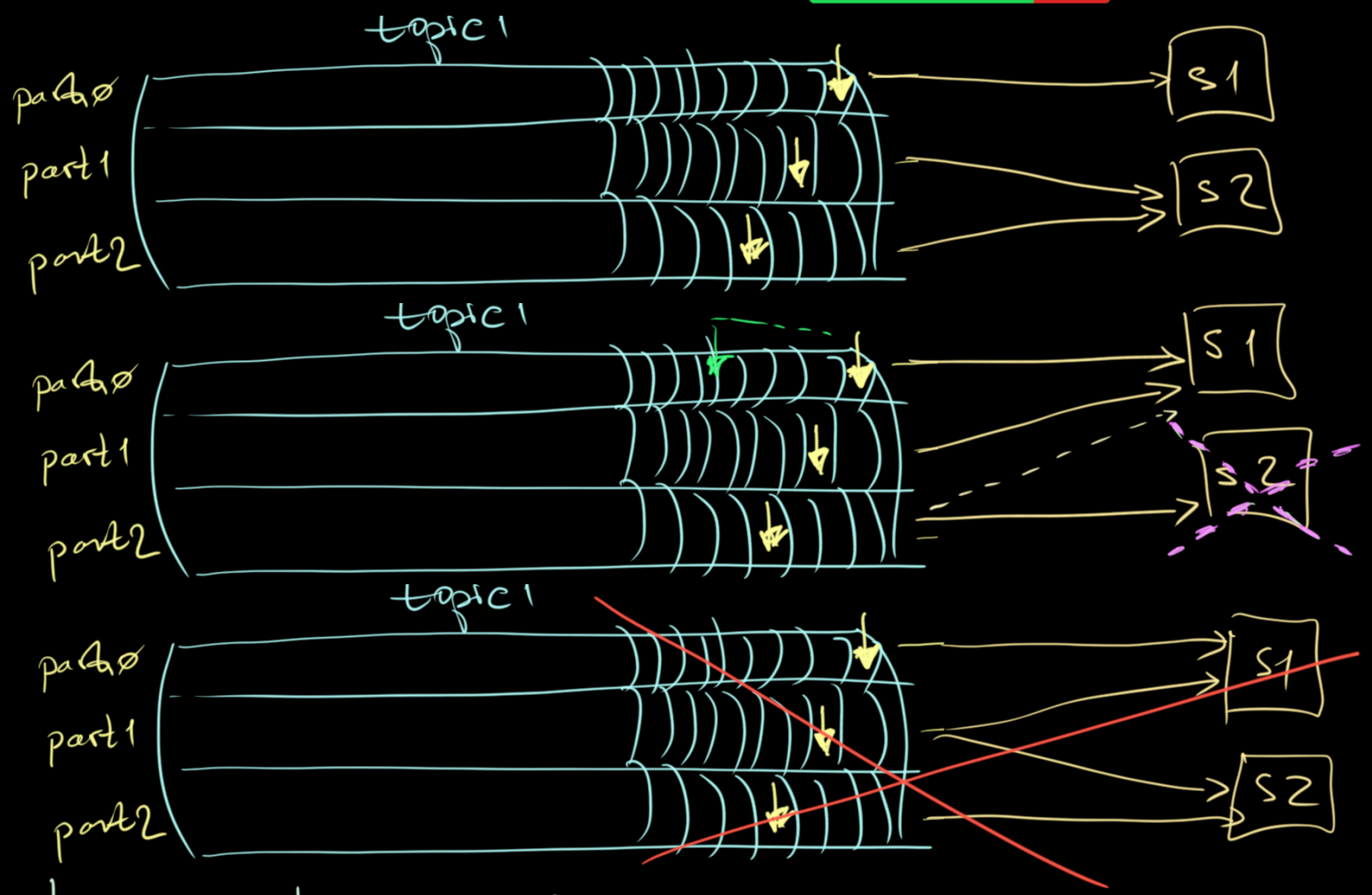




n → n/2

n/3 → n/4


topic 1
S1
S2
S3
topic 1
part 0
part 1
part 2
S1
S2
S3
partitions ≥ # services (listeners)
S1
S2



topic ~ shop / market
partition ~ queue to the cash desk
services (consumers / listeners) ~ cash desk

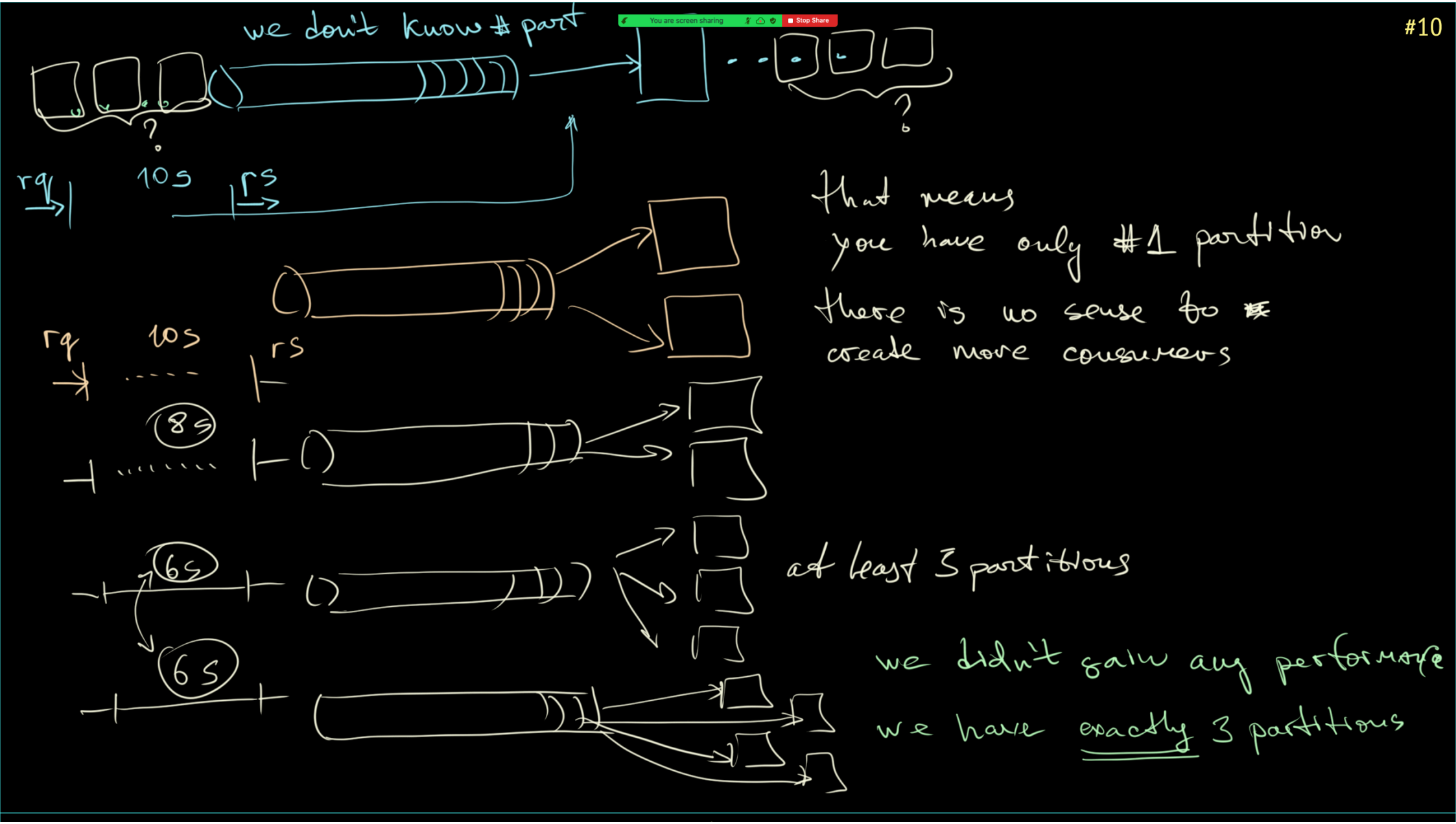
```java
@KafkaListener(id = "group1", topics = "topic1",containerFactory = "kafkaListenerContainerFactory123")
public void handleTopic1p0(ConsumerRecord<String, String> cr) throws InterruptedException {
  String key = cr.key();
  String value = cr.value();
  log.info( message: "Got Kafka Record: key:{}, value:{}", key, value);
  log.info("doing business...");

  Thread.sleep( millis: 2000);

  String valueProcessed = value.toUpperCase();

  log.info("... done business");
  publisher.send( topic: "topic2", key, valueProcessed);
}
```

one listener
we don't specify part

manual consuming by different listeners

```java
@KafkaListener(
    topicPartitions = @TopicPartition(topic = "topic1", partitions = "0"),
    containerFactory = "kafkaListenerContainerFactory123"
)
public void handleTopic1p0(ConsumerRecord<String, String> cr) throws InterruptedException {
  String key = cr.key();
  String value = cr.value();
  log.info( message: "Got Kafka Record: key:{}, value:{}", key, value);
  log.info("doing business...");

  Thread.sleep( millis: 2000);

  String valueProcessed = value.toUpperCase();

  log.info("... done business");
  publisher.send( topic: "topic2", key, valueProcessed);
}
```

```java
@KafkaListener(
    topicPartitions = @TopicPartition(topic = "topic1", partitions = "0"),
    containerFactory = "kafkaListenerContainerFactory123"
)
public void handleTopic1p0(ConsumerRecord<String, String> cr) throws InterruptedException {
  String key = cr.key();
  String value = cr.value();
  log.info( message: "Got Kafka Record: key:{}, value:{}", key, value);
  log.info("doing business...");

  Thread.sleep( millis: 2000);

  String valueProcessed = value.toUpperCase();

  log.info("... done business");
  publisher.send( topic: "topic2", key, valueProcessed);
}
```

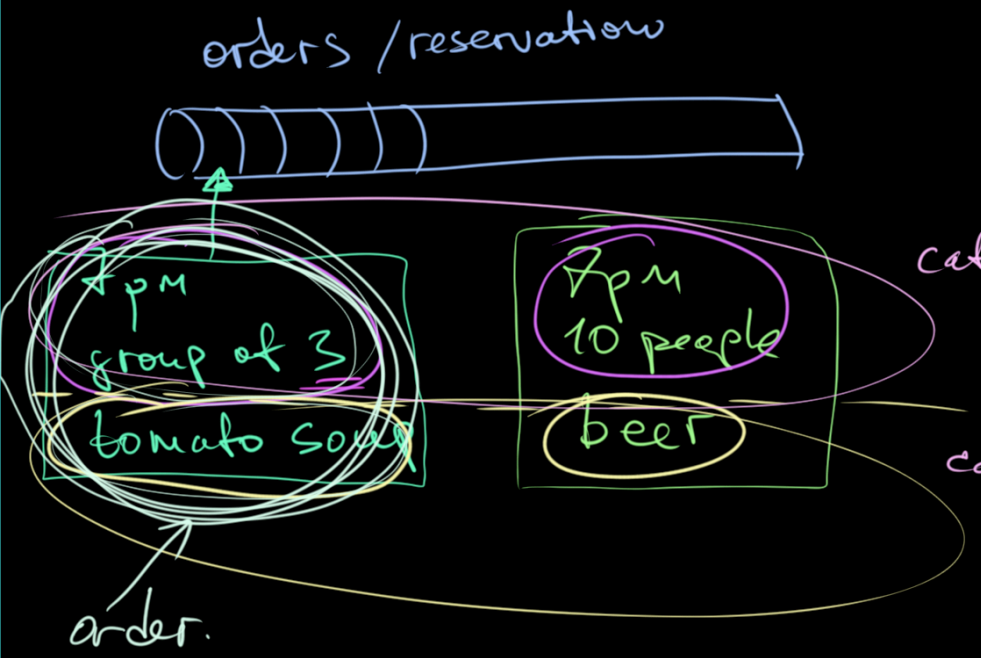


that means
you have only #1 partition
there is no sense to
create more consumers

at least 3 partitions

we didn't gain any performance
we have exactly 3 partitions

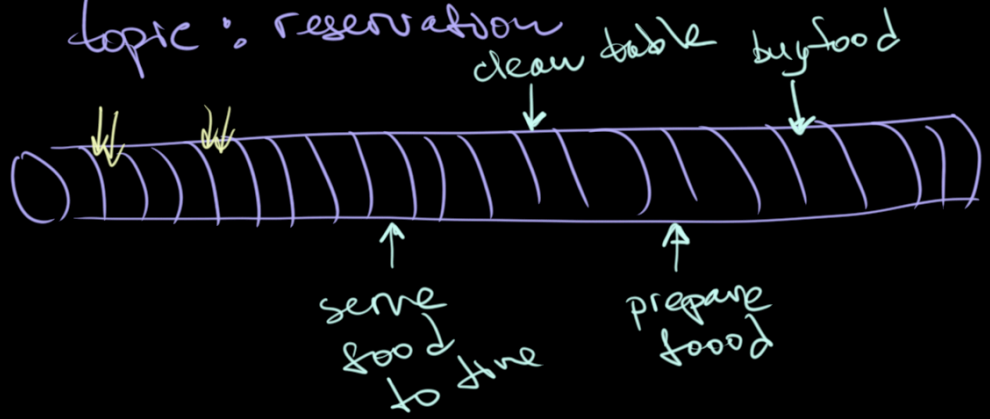
Restaurant

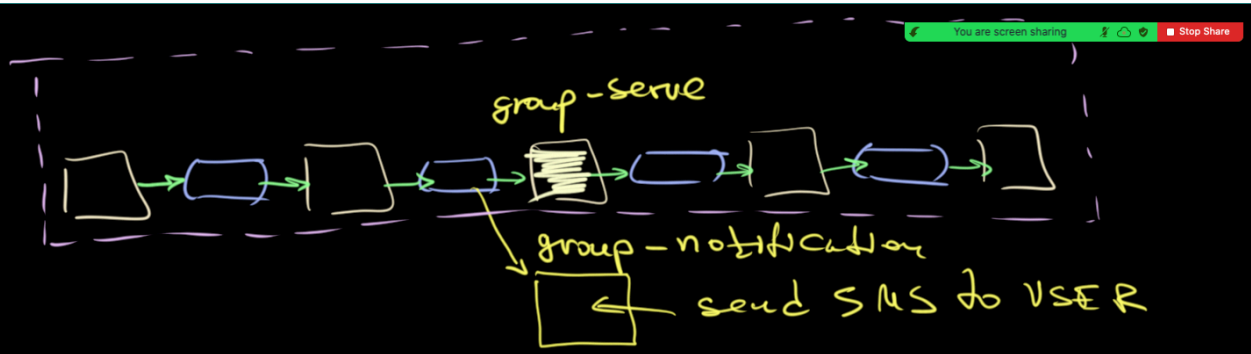


consumer group - reservation

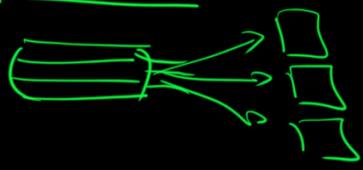
consumer group - kitchen

- as many consumer groups as we want
- different ports for different consumer groups





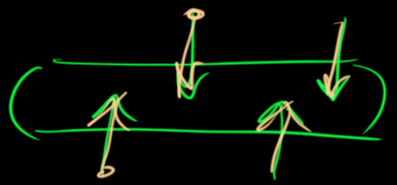
partitions → for similar parallel processing



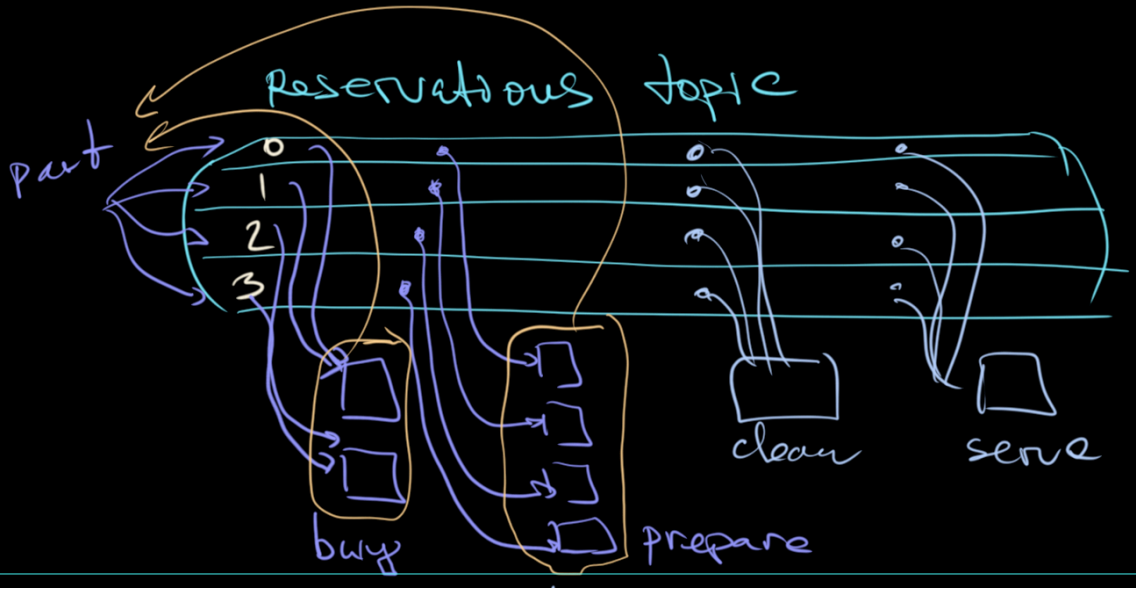
publisher
- topic, key, value

listener
- topic
- group-id

groups → for non-similar non-parallel processing



different publishers in one topic



groups
- buy food x2
- prepare food x4
- clean table x1
- serve table x1

rabbit is pushing to service

service is pulling from kafka

we can lose our data

we can't lose the data

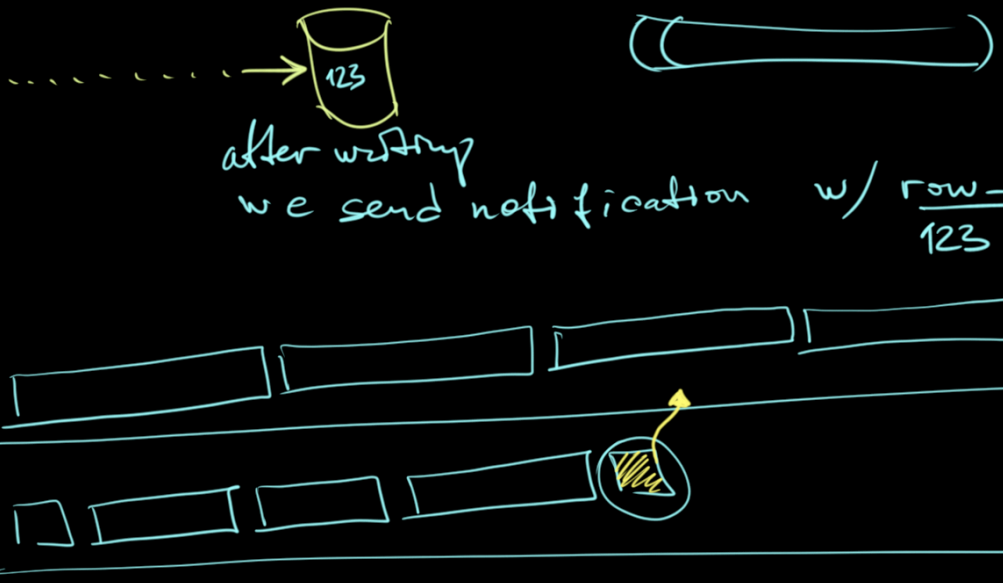
2 stages
1. pull data
2. commit offset

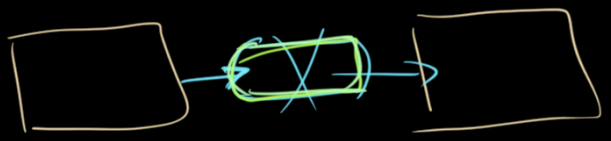
the most common use cases for the different groups

- another listener to put the data to different datasource (backup)
- notification (sms, email, push notif ....)
- tracking
- aggregation → total # of orders today / month
  → min / max / average / sum

TTL Time To Live

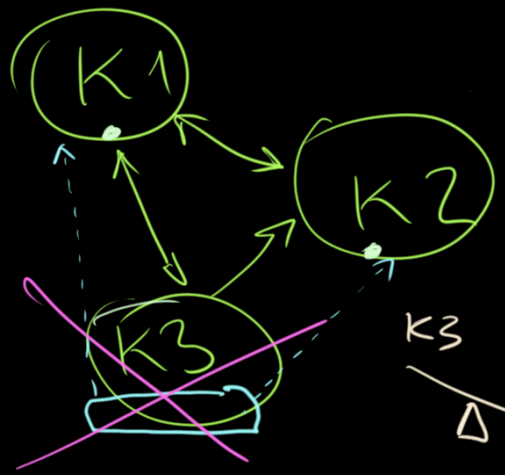
you can specify how long your data will live is ... ∞

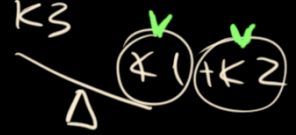


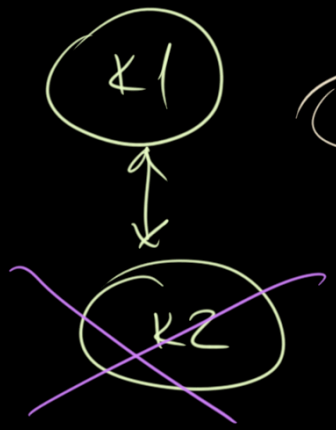


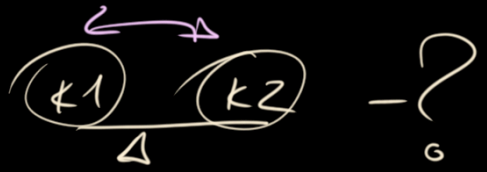
what if our Kafka down
=> we have broken everything

Kafka out of the box is clustered





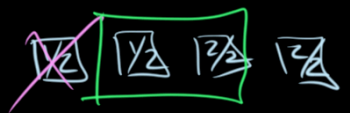




transaction
replica=3

notification
replica=2

raid 0
raid 1 => twice faster



replica =1