Kafka Consumers

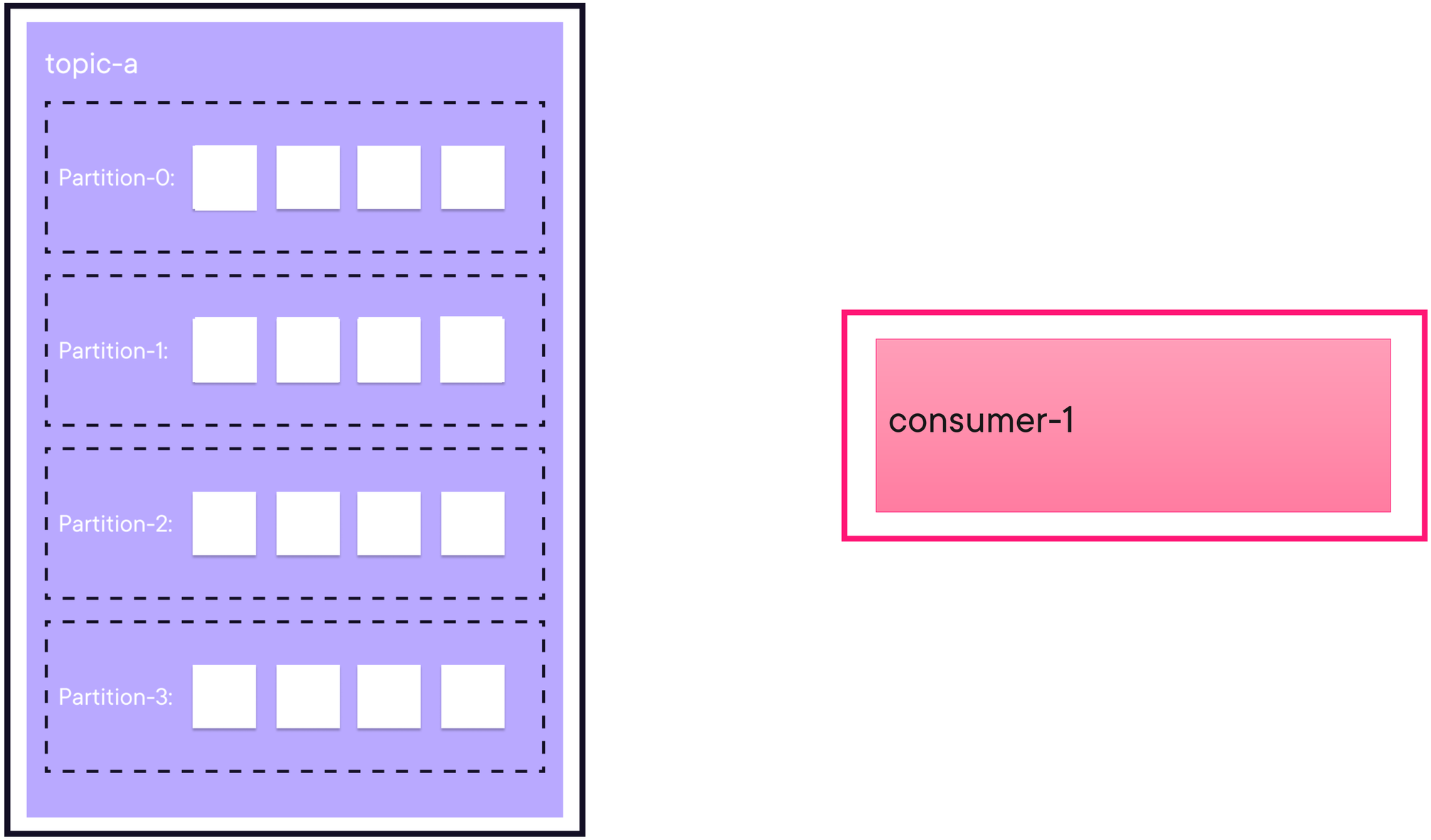
Axel Sirota

AI and Cloud Consultant

@AxelSirota

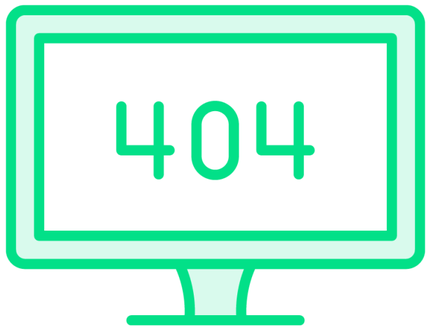
Kafka Consumer Group

Consumers are typically done as a group


topic-a
Partition-0:
Partition-1:
Partition-2:
Partition-3:
consumer-1

Kafka Consumer Group

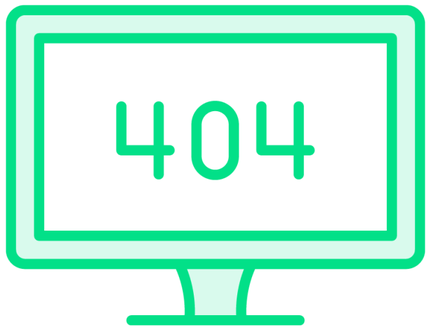
Consumers are tipically done as a group



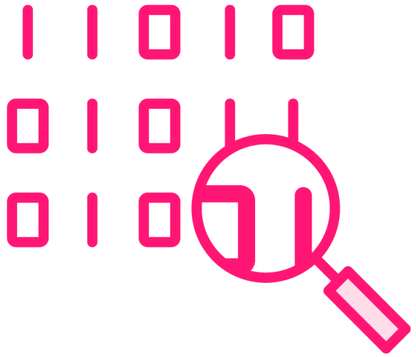
A single consumer will end up inefficient with large amounts of data

Kafka Consumer Group

Consumers are tipically done as a group

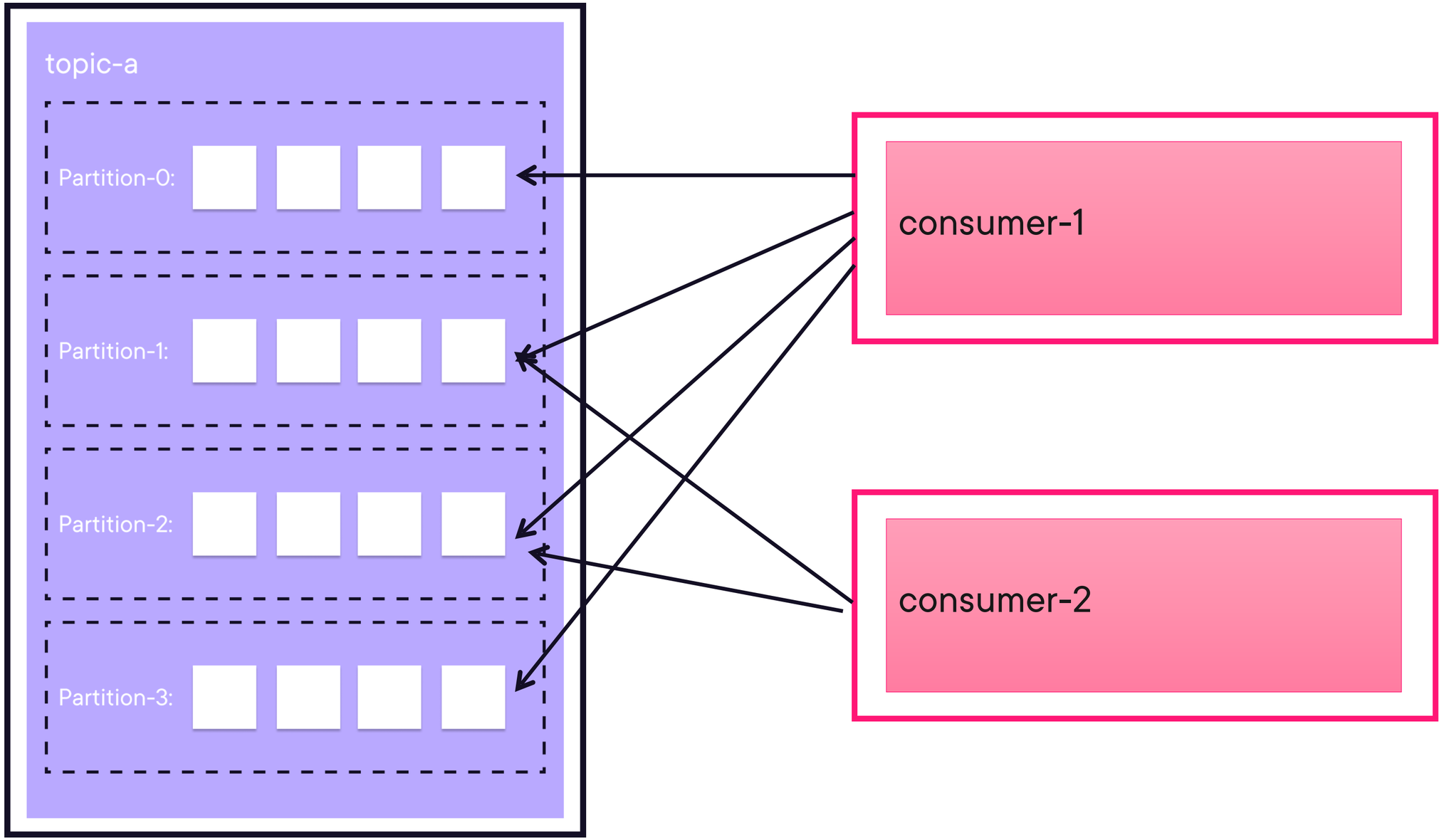


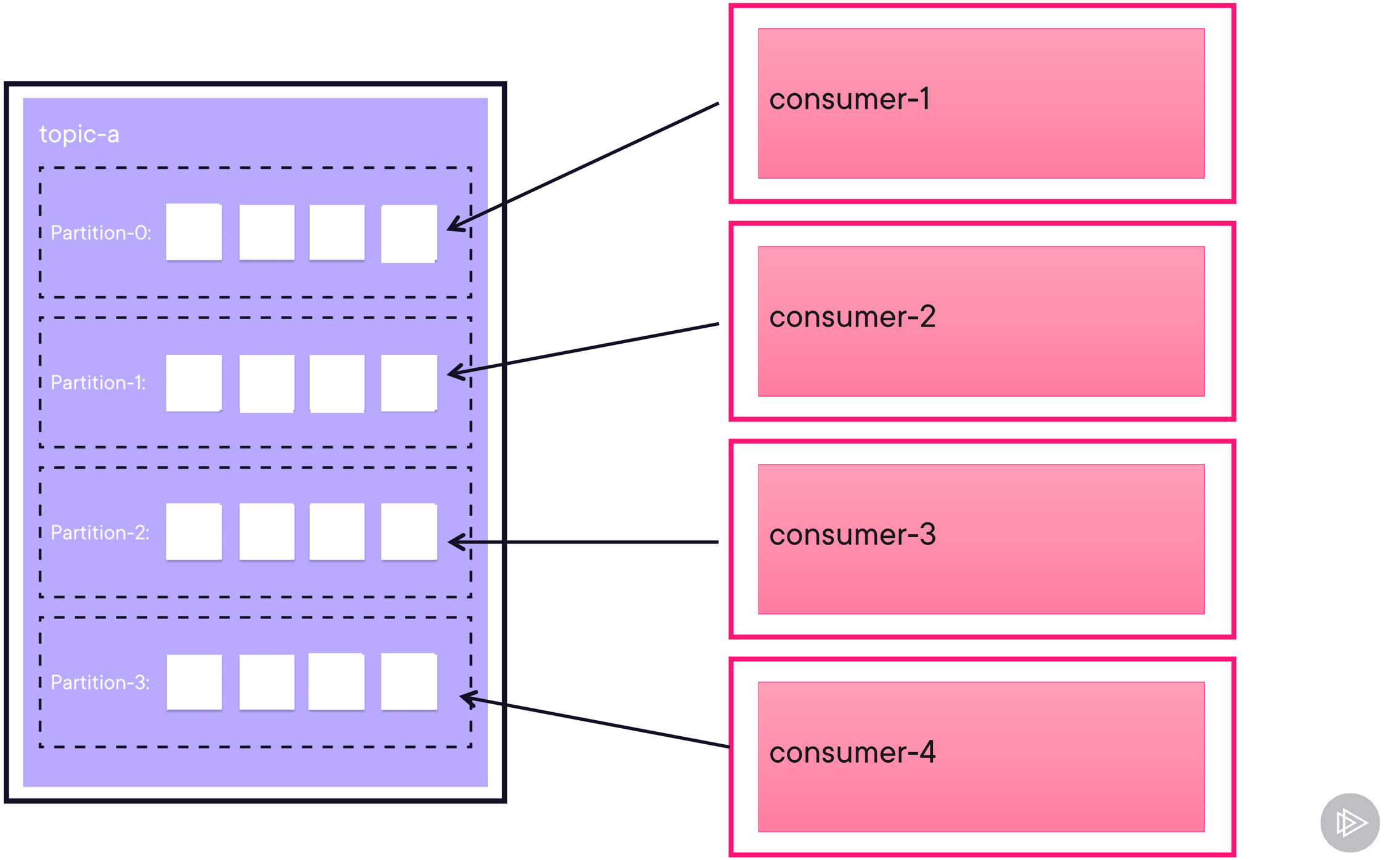
A single consumer will end up inefficient with large amounts of data

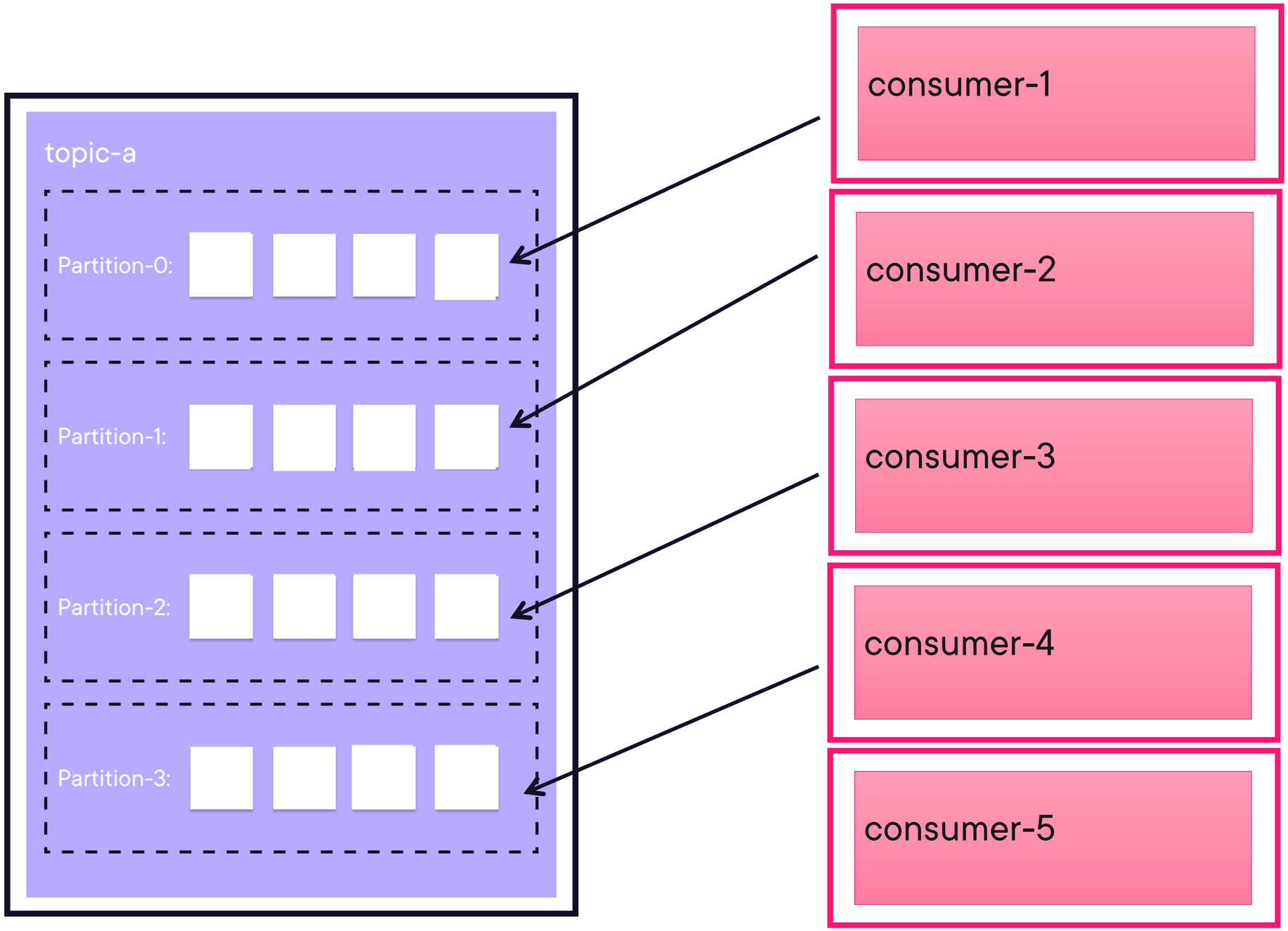


A consumer may never catch up

The consumer group id is key so Kafka knows that messages should be distributed to both consumers without duplicating

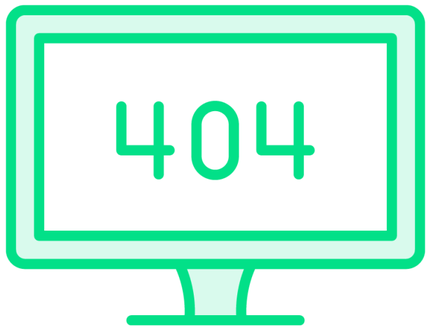

topic-a
Partition-0:
Partition-1:
Partition-2:
Partition-3:
consumer-1
consumer-2


topic-a
Partition-0:
Partition-1:
Partition-2:
Partition-3:
consumer-1
consumer-2
consumer-3
consumer-4

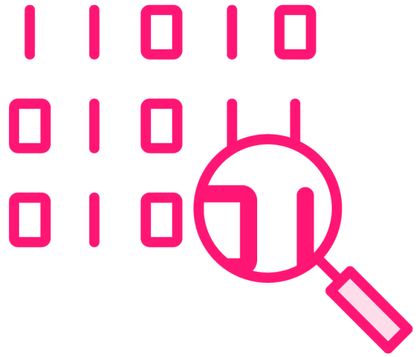

topic-a
Partition-0:
Partition-1:
Partition-2:
Partition-3:
consumer-1
consumer-2
consumer-3
consumer-4
consumer-5

Kafka Consumer Group

Consumers are typically done as a group



A single consumer will end up inefficient with large amounts of data



A consumer may never catch up

Every consumer should be on it’s own machine, instance, pod

Consuming from Kafka

Deserializers and Consumer Configuration

Construct a java.util.Properties object

Deserializers and Consumer Configuration

```
Properties properties = new Properties();

properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, “localhost:9092“);

properties.put(ConsumerConfig.GROUP_ID_CONFIG, "my_group");

properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,"org.apache.kafka
.common.serialization.StringDeserializer");

properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,"org.apache.kaf
ka.common.serialization.IntegerDeserializer");

properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest“);

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);
```

Deserializers and Consumer Configuration

Provide two or more locations where the Bootstrap servers are located

```
Properties properties = new Properties();

properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, “localhost:9092“);

properties.put(ConsumerConfig.GROUP_ID_CONFIG, "my_group");

properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,"org.apache.kafka
.common.serialization.StringDeserializer");

properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,"org.apache.kaf
ka.common.serialization.IntegerDeserializer");

properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest“);

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);
```

Deserializers and Consumer Configuration

Provide a “Team Name”, officially called a group.id

```
Properties properties = new Properties();

properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, “localhost:9092“);

properties.put(ConsumerConfig.GROUP_ID_CONFIG, "my_group");

properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,"org.apache.kafka
.common.serialization.StringDeserializer");

properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,"org.apache.kaf
ka.common.serialization.IntegerDeserializer");

properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest“);

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);
```

Deserializers and Consumer Configuration

Provide a DeSerializer for the key

```
Properties properties = new Properties();

properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, “localhost:9092“);

properties.put(ConsumerConfig.GROUP_ID_CONFIG, "my_group");

properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,"org.apache.kafka
.common.serialization.StringDeserializer");

properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,"org.apache.kaf
ka.common.serialization.IntegerDeserializer");

properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest“);

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);
```

Deserializers and Consumer Configuration

Provide a DeSerializer for the value

```
Properties properties = new Properties();

properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, “localhost:9092“);

properties.put(ConsumerConfig.GROUP_ID_CONFIG, "my_group");

properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,"org.apache.kafka
.common.serialization.StringDeserializer");

properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,"org.apache.kaf
ka.common.serialization.IntegerDeserializer");

properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest“);

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);
```

Consumer Object

Construct a org.apache.kafka.clients.consumer.KafkaConsumer object

```
KafkaConsumer consumer = new KafkaConsumer<>(properties);
```

Processing

Use the consumer that you have constructed, and call poll with pulse time.

```
while (!done.get()) {
      ConsumerRecords<String, String> records =
      consumer.poll(Duration.of(500, ChronoUnit.MILLIS));
  for (ConsumerRecord<String, String> record : records) {
      System.out.format("offset: %d\n", record.offset());
      System.out.format("partition: %d\n", record.partition());
      System.out.format("timestamp: %d\n", record.timestamp());
      System.out.format("timeStampType: %s\n",
record.timestampType());
       System.out.format("topic: %s\n", record.topic());
       System.out.format("key: %s\n", record.key());
       System.out.format("value: %s\n", record.value());
   }
}
```

Be a Good Citizen

Processing

```
consumer.close();
```

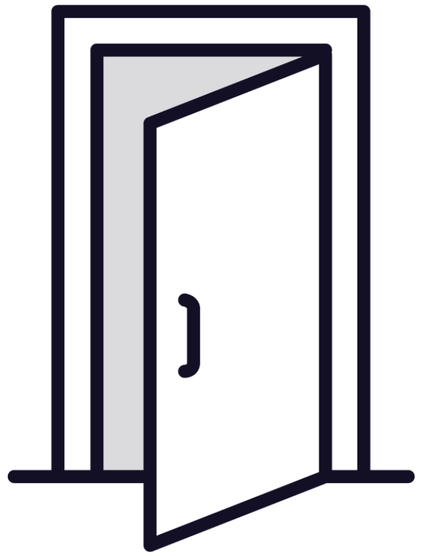
Speaking of Rebalances

Regardless of what a consumer is doing, at regular intervals, which are configurable, they send a heartbeat to Kafka

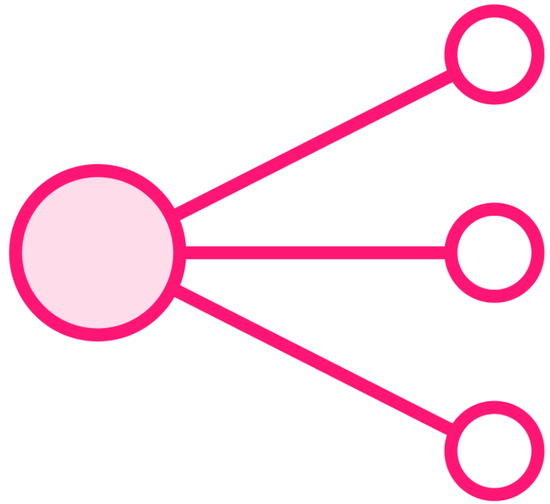
Specifically to the coordinator inside the broker, basically letting them know they are alive

Also when we poll for records that lets Kafka know that consumer is alive

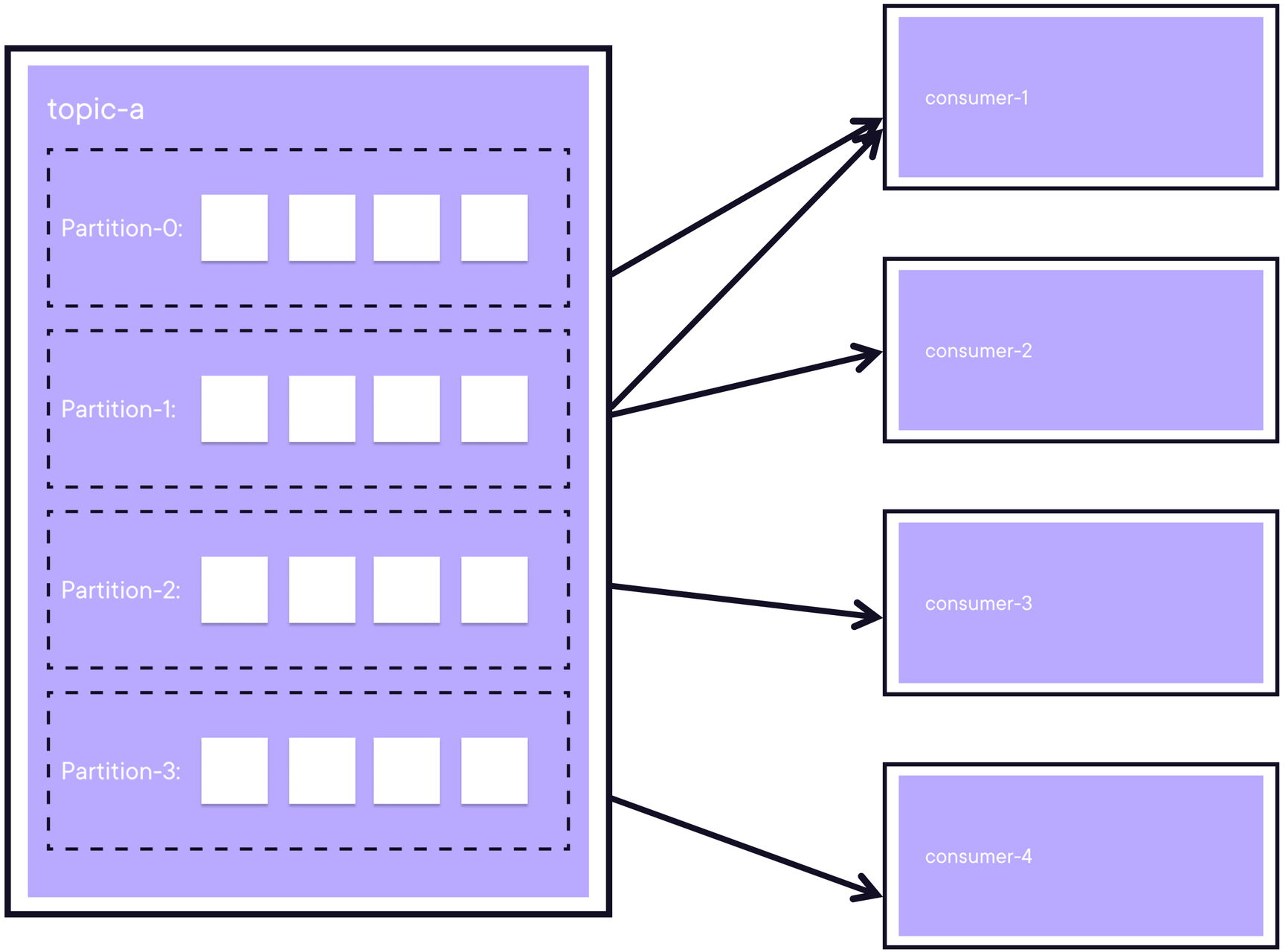
Speaking of Rebalances



And as we said when you close a consumer that close method lets Kafka know they are not alive anymore



Which triggers a rebalance


topic-a
Partition-0:
Partition-1:
Partition-2:
Partition-3:
consumer-1
consumer-2
consumer-3
consumer-4

Consuming Messages with Java

Key, Takeaways, and Tips

Takeaways

Consumers act as a group via the Consumer group ID

Each consumer in the group gets assigned a partition, and a partition is not shared by two members of a group

A rebalance is triggered when a member leaves the group or they haven’t sent a heartbeat in a long time

A rebalance is a stop the world event that ensures all partitions are attended by some consumer in the group

Keys

Be sure to configure your Consumer to ensure you are not duplicating messages

Try to create a consumer loop with two consumers and verify the partition assignament playing with keys in the producer

See what happens in the logs of the broker when a consumer joins or leaves the group

Up Next:

Kafka Streams