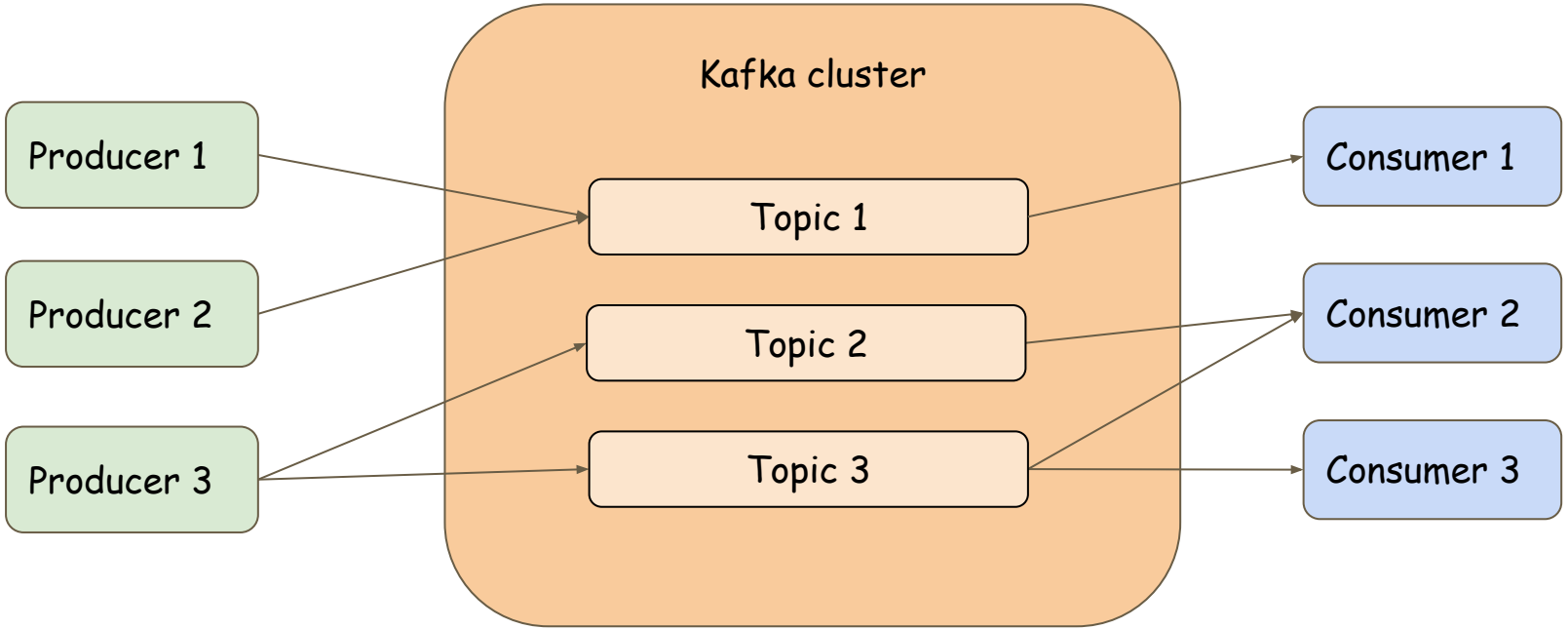
JavaCro'18

Using KafkaStreams API to deal with challenges of developing distributed applications

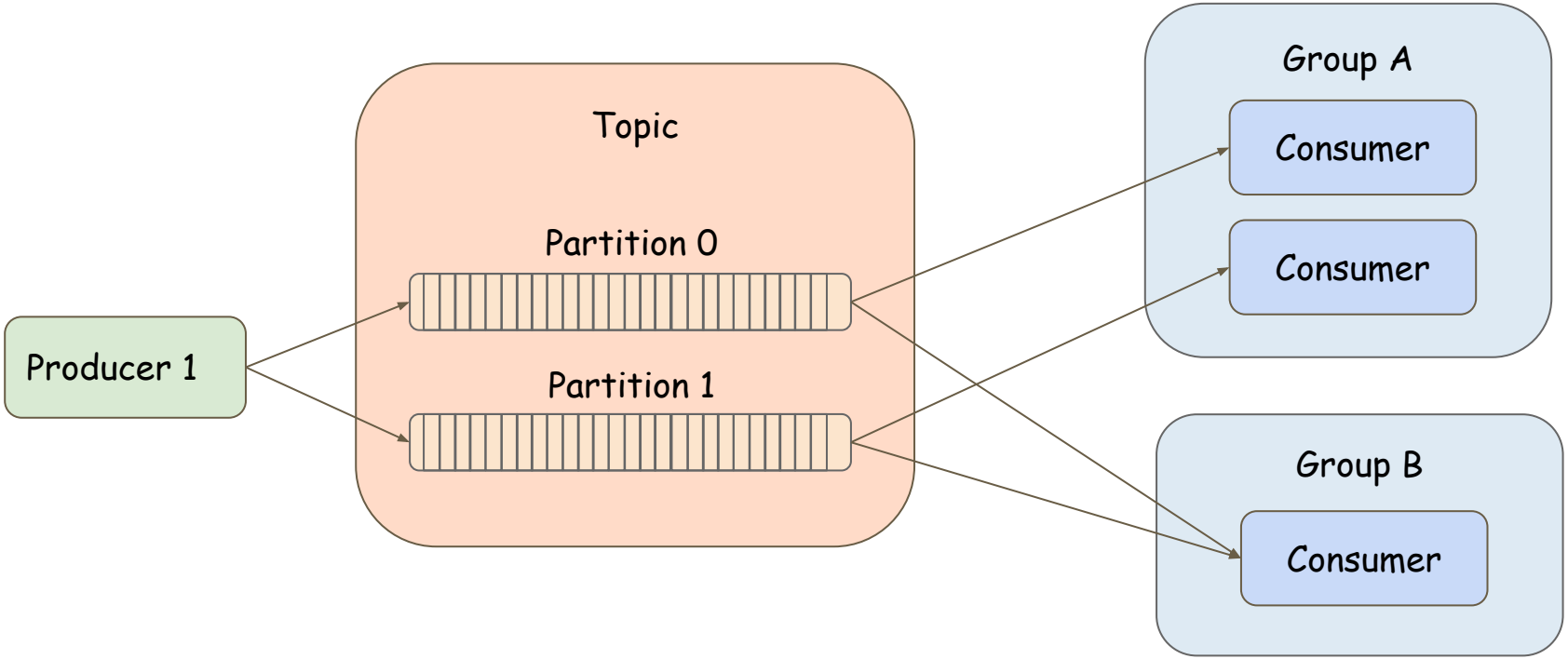
Igor Buzatović

INOVATREND
Inovativni trendovi d.o.o.

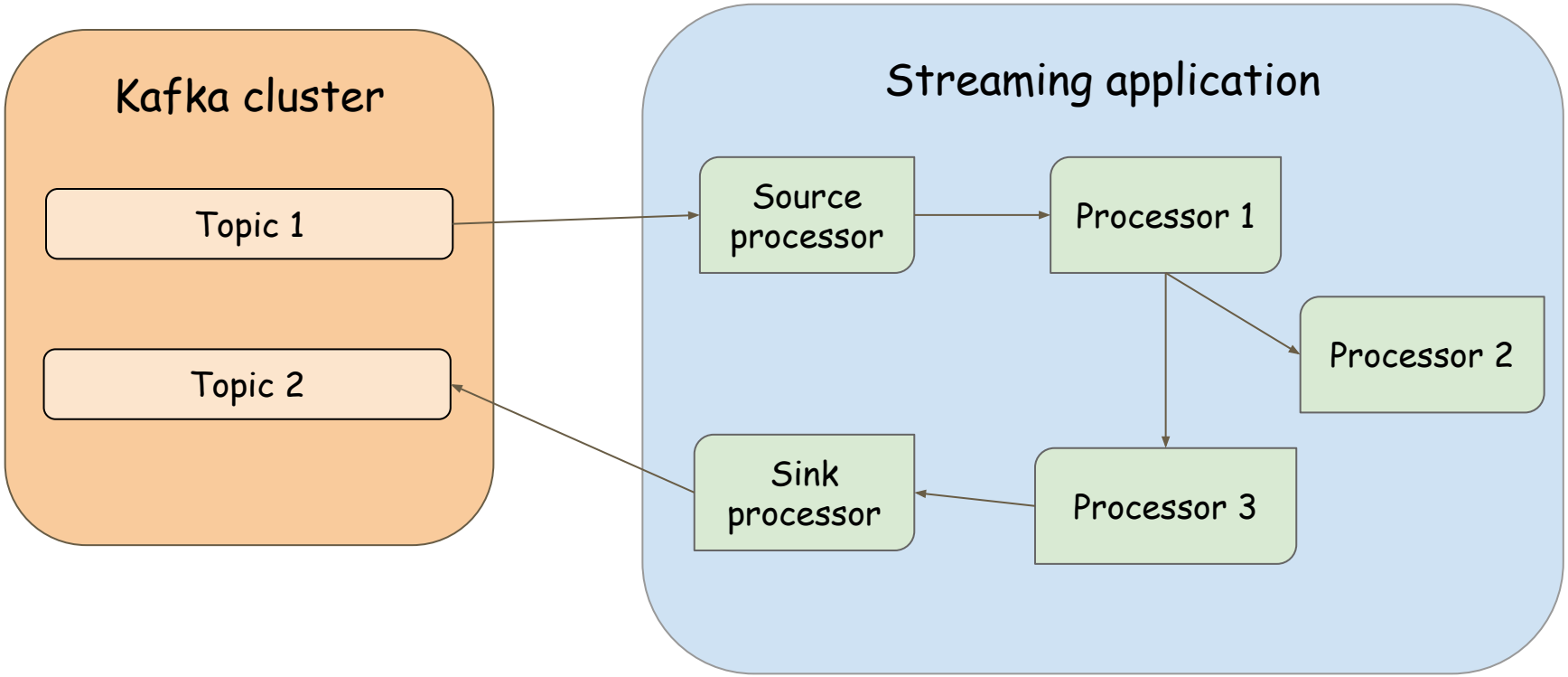
Kafka basics - Brokers, Producers, Consumers



Kafka basics - Consumer groups



Kafka Streams - Processor topology



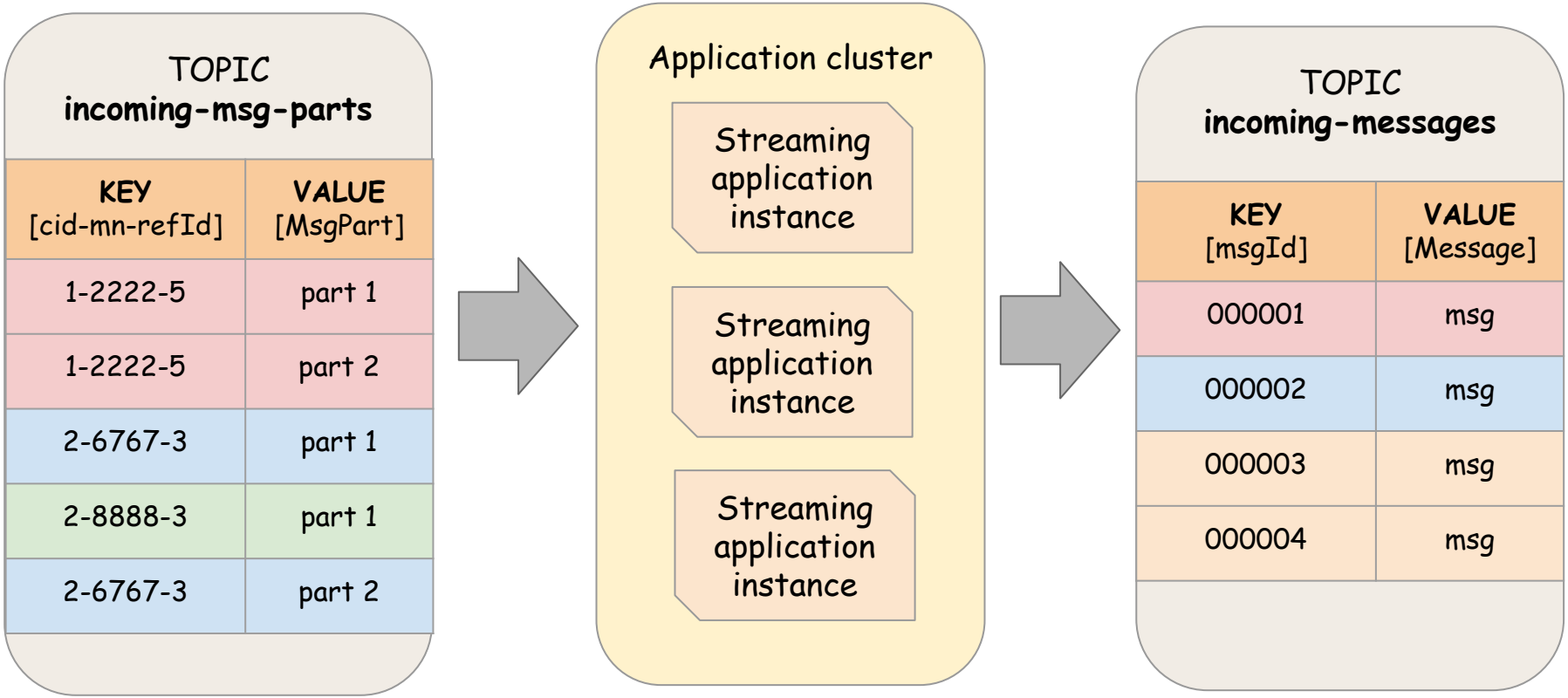
KafkaStreams - Low level API

```
Topology topology = new Topology();

topology.addSource("INPUT", "input_topic");

topology.addProcessor (
    "WORKER", () -> new WorkerProcessor(), "INPUT"
);

topology.addSink("OUTPUT", "output_topic", "WORKER")
```

KafkaStreams - High level API

```
StreamsBuilder builder = new StreamsBuilder();
KStream<String, String> inputStream = builder.stream("input-topic");
inputStream
        .filter ( (key, value) -> value.length() > 10 )
        .groupByKey()
        .count()
        .toStream()
        .to("output-topic");
```

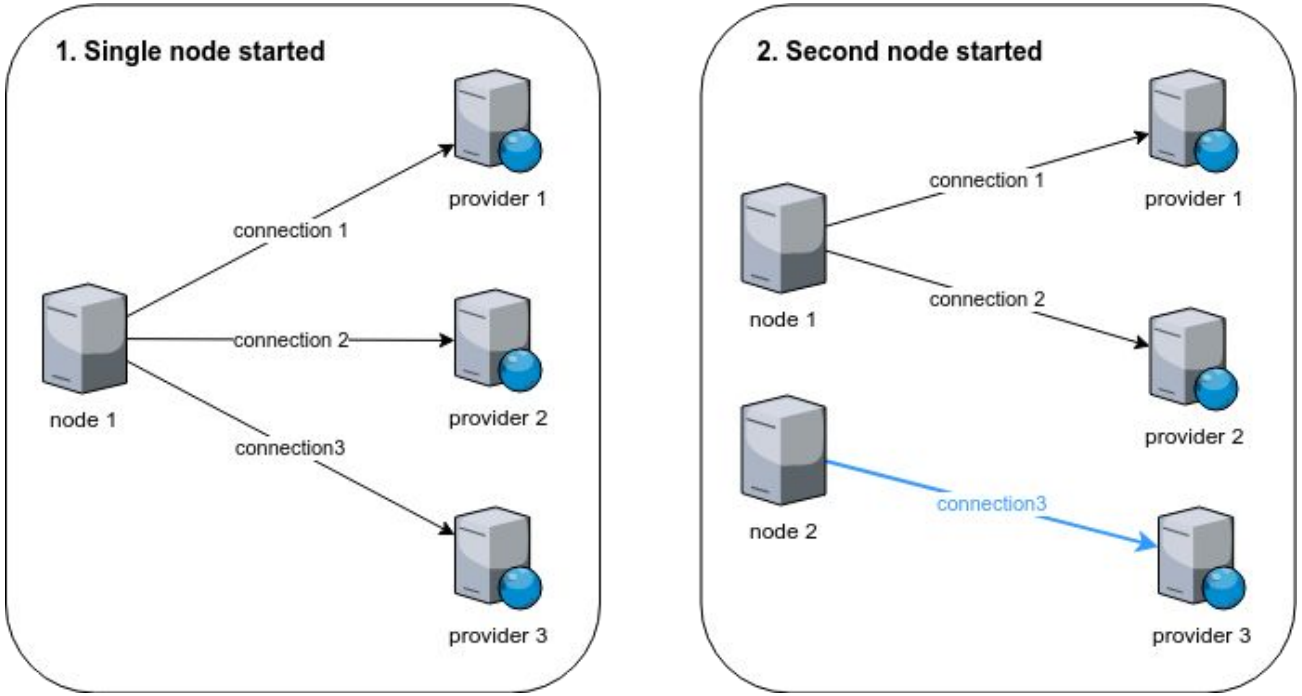
Use case 1 - Aggregate SMS message parts



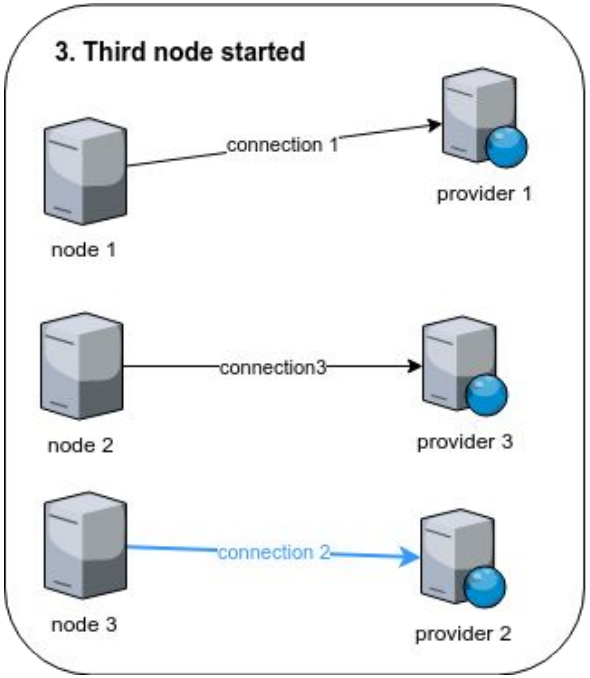
Use case 1 - Aggregate SMS message parts

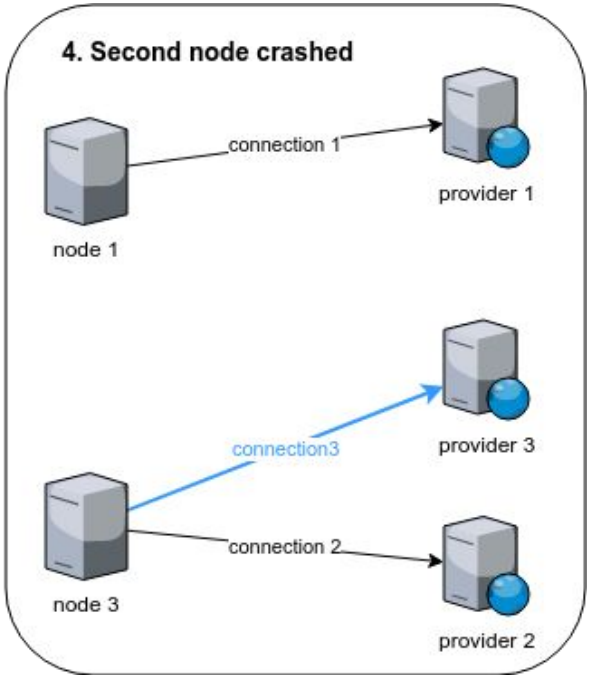
```
config.put(StreamsConfig.APPLICATION_ID_CONFIG, "incoming-msg-parts-processor" )
. . .
val inputStream = builder.stream<String, MessagePart>("incoming-msg-parts")
inputStream
        .groupByKey()
        .aggregate (
                Initializer { Message() },
                Aggregator { key, part, message ->
                        message.addPart(part)
                }
        )
       .filter { key, message -> message.parts.size == message.totalPartsCount }
       .to( "incoming-messages")
```

Use case 2 - Distribute connections across nodes

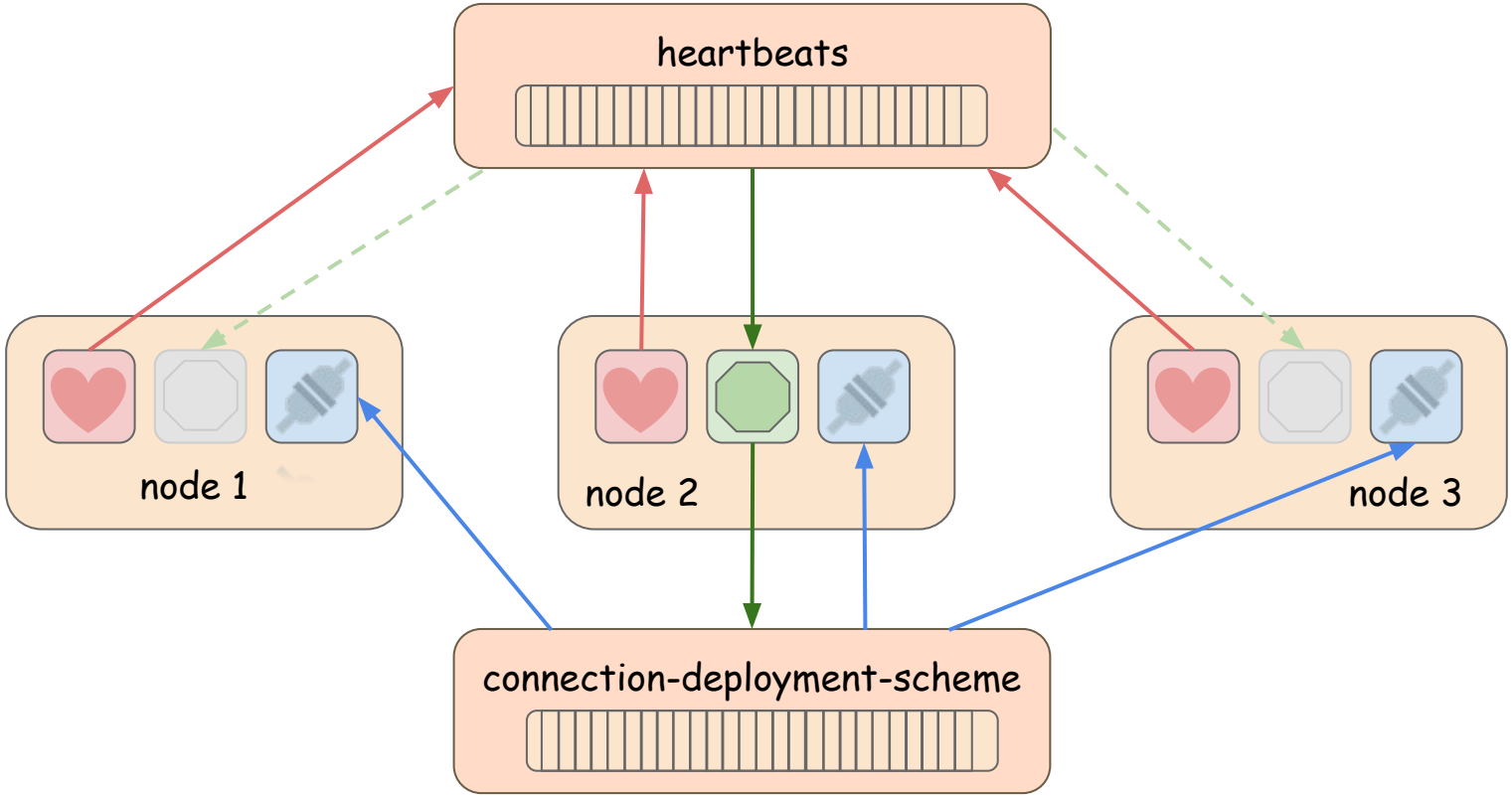


Use case 2 - Distribute connections across nodes





Use case 2 - Distribute connections across nodes



Use case 2 - Heartbeat sender

```
producer = KafkaProducer ( producerConfig )

scheduler.scheduleAtFixedRate (

   Runnable {
         val now = System.currentTimeMillis()
         val connectionsInfo = connectionManager.currentDeployments
         val nodeInfo = HeartBeat ( nodeId, connectionsInfo, now )
         producer.send ( ProducerRecord ( "heartbeats", "tick", nodeInfo ) )
      },

      0, 1, TimeUnit.SECONDS
)
```

Use case 2 - Rebalancer

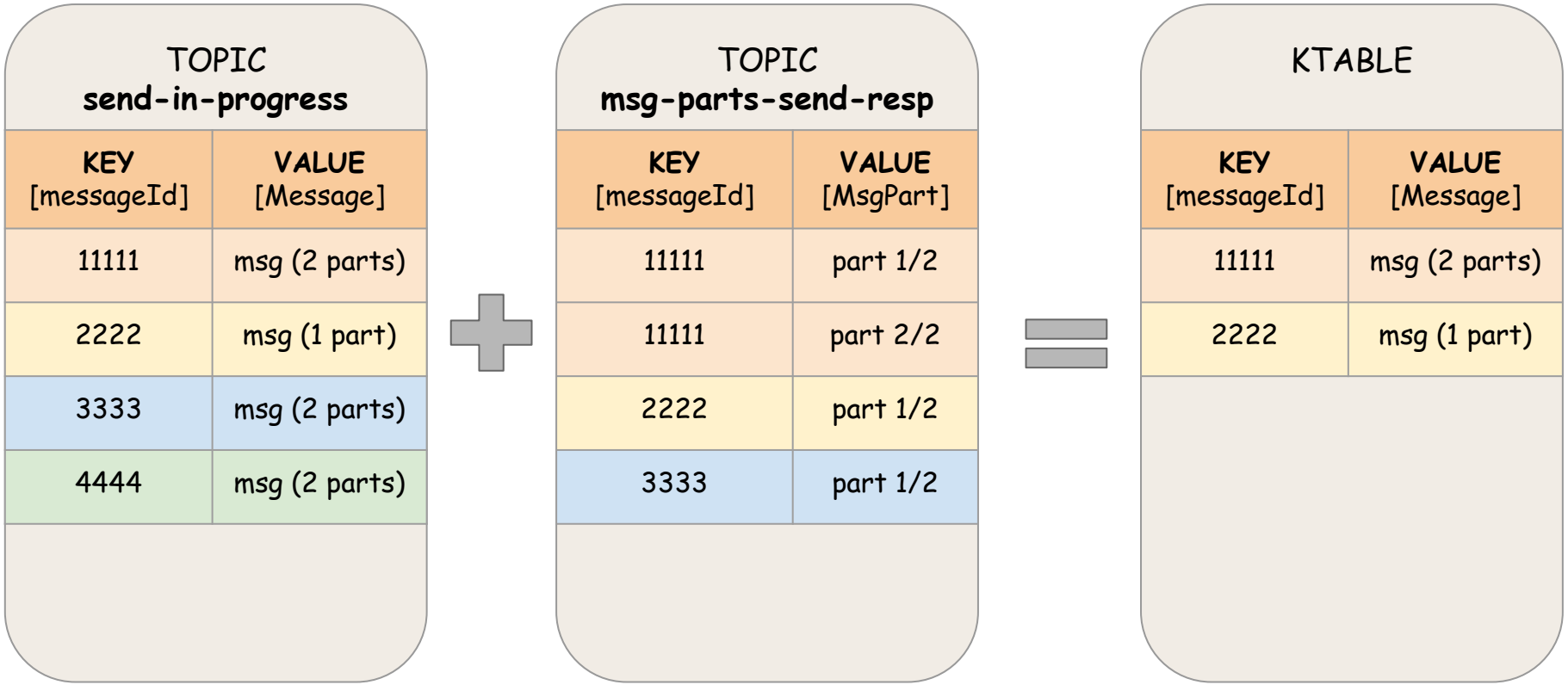
JavaCro'18

```
val inputStream = builder.stream<String, HeartBeat>("heartbeats")
inputStream.groupByKey()
    .windowedBy (TimeWindows.of( 5000 ) )
    .aggregate (
        Initializer { HeartBeatsList() },
        Aggregator { key, heartbeat, heartBeatsList ->
            heartBeatsList.addHeartBeat(heartbeat)
        }
    )
    .filter( { key, value -> isCurrentWindow ( key.window() ) } )
    .foreach { key, heartBeatsList ->
        if ( isNextWindow ( key.window() ) )
            recalulateConnectionDeployment ( currentHeartBeatsList )
        this.currentWindow = key.window()
        this.currentHeartBeatsList = heartBeatsList
    }
```

Use case 2 - ConnectionManager

```
val consumer = KafkaConsumer <String, ConnectionDeploymentScheme> ( consumerConfig )
consumer.subscribe( listOf ( "connection-deployment-scheme" ) )
consumer.seekToEnd ( listOf () )

thread {
    val records = consumer.poll(1000)
    val lastRecord = records.lastOrNull()
    if (lastRecord != null)
      handleRecord( lastRecord.value() )
}
```

Use case 3 - handling send responses



Use case 3 - handling send responses

```
val sentMessages = builder.table<String, Message>("send-in-progress")
val inputStream = builder.stream<String, MessagePart>("msg-parts-send-resp")
   inputStream .groupByKey()
   .aggregate (
      Initializer { Message() },
      Aggregator { msgId, part, message -> message.addPart(part) }
   )
   .filter { msgId, message -> message.parts.size == message.totalPartsCount }
   .join (
       sentMessages,
       { msgId,  message ->
          message.updateStatus(MessageStatus.SENT)  // returns updated message
       }
   )
   .to("mt-messages")
```

Use case 3 - handling delivery reports

TOPIC
send-in-progress

KEY [messageId]	VALUE [Message]
11111	msg
2222	msg
3333	msg
4444	msg

TOPIC
msg-parts-send-resp

KEY [msgId]	VALUE [MsgPart]
11111	part 1/2
11111	part 2/2
2222	part 1/1
3333	part 1/2

TOPIC
msg-parts-delivery

KEY [spId-outId]	VALUE [DLR]
1-001	DLR
1-002	DLR
1-003	DLR
1-004	DLR

Use case 3 - handling delivery reports

```
val sentMessages  = builder.table<String, Message>("send-in-progress")
val msgPartSendResponses = builder.stream<String, MessagePart>("msg-parts-send-resp")
val msgPartDLRs = builder.stream<String, DeliveryReport>("msg-parts-delivery")

val msgPartSendRespTable =
      msgPartSendResponses
           .groupBy( { msgId, part -> "${part.recpId}-${part.outId}" } )
           .reduce({ value1, value2 -> value1 })

val msgPartsWithDeliveryReports = msgPartDLRs
     .join (
           msgPartSendResponseTable,
           { dr, part -> part }
      )
```

Use case 3 - handling delivery reports

```
msgPartsWithDeliveryReports
        .groupBy { _, part -> part.messageId }
        .aggregate (
                Initializer { Message() },
                Aggregator { msgId, part, message -> message.addPart ( part ) }
        )
       .filter { msgId, message -> message.parts.size == message.totalPartsCount }
       .join(
             sentMessages,
             { parts, message -> message.updateStatus(MessageStatus.DELIVERED) }
       )
       .toStream()
       .to("mt-messages")
```

Questions
&
Discussion

JavaCro'18

Thank you!

Email: igor.buzatovic@inovatrend.com

Blog: http://blog.inovatrend.com

Linkedin: https://www.linkedin.com/in/bigor

Twitter: @buzzgor