SpringOne Platform

Reactive Kafka

Rajini Sivaram

Pivotal

Safe Harbor Statement

- *The following is intended to outline the general direction of Pivotal's offerings. It is intended for information purposes only and may not be incorporated into any contract. Any information regarding pre-release of Pivotal offerings, future updates or other planned modifications is subject to ongoing evaluation by Pivotal and is subject to change. This information is provided without warranty or any kind, express or implied, and is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions regarding Pivotal's offerings. These purchasing decisions should only be based on features currently available. The development, release, and timing of any features or functionality described for Pivotal's offerings in this presentation remain at the sole discretion of Pivotal. Pivotal has no obligation to update forward looking information in this presentation.*

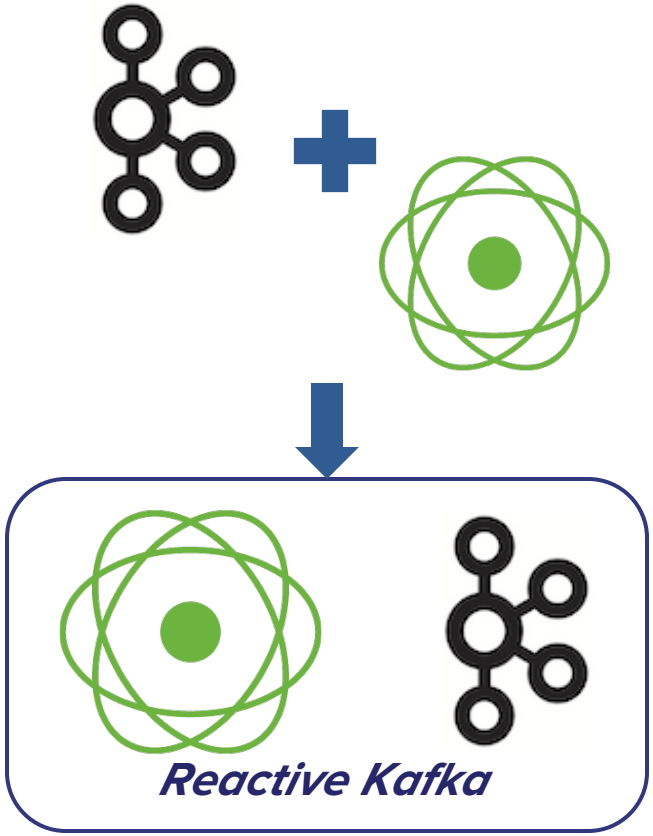
About me

Pivotal

- *Principal Software Engineer at Pivotal UK*
- *Previously at IBM*
  - *Message Hub developer – Kafka-as-a-Service on Bluemix*
- *Contributor to Apache Kafka*

Agenda

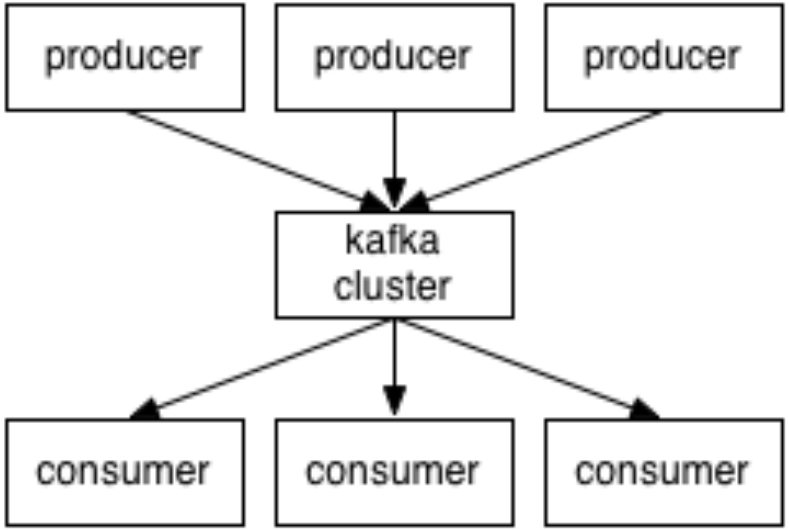
- *Introduction to Apache Kafka*
  - *Clients and tools for Kafka*
  - *Non-reactive Java clients*
- *Introduction to Reactive Streams*
- *Reactive Kafka*
  - *Reactive API for Kafka*
  - *Performance evaluation*
- *Summary*



What is Kafka?

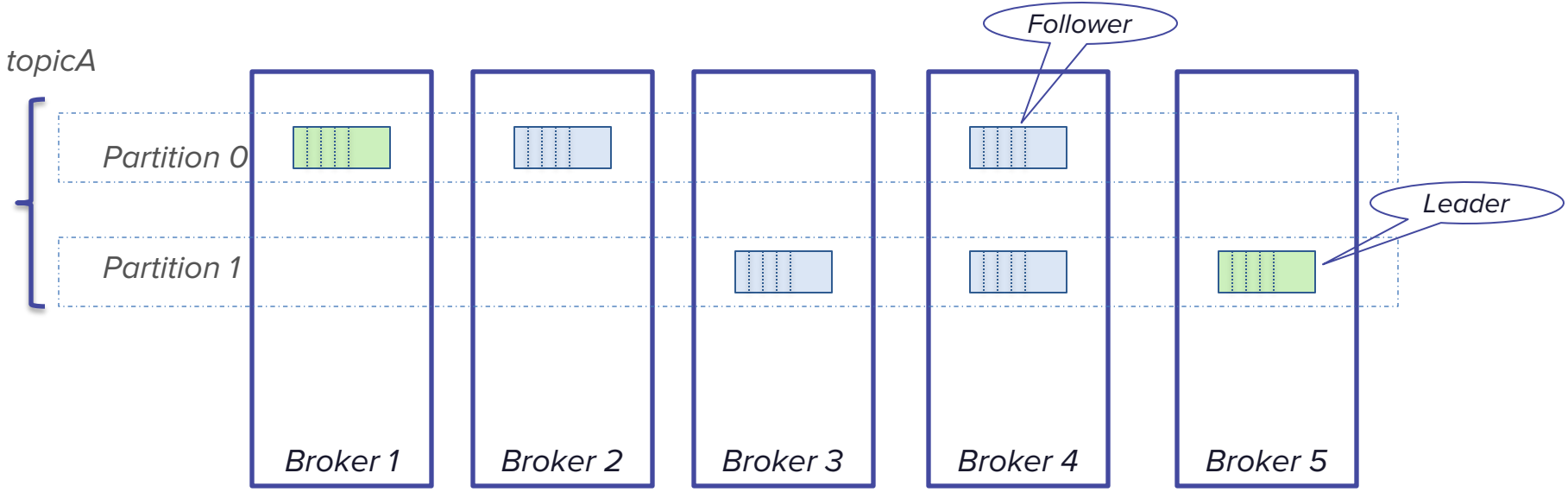
- *Message Bus*
  - *Publish/Subscribe*
  - *Fast*
  - *Scalable*
  - *Durable*
  - *Highly available*
- *Commit Log Service*
  - *Distributed*
  - *Replicated*
  - *Partitioned*

Kafka as a Message Bus



- *Decouples producers and consumers*
- *Messages are stored for a configurable retention interval*
  - *Consumers can replay message stream*

Kafka as a Commit Log Service



- *Partition*
  - *Immutable, Ordered Stream*
  - *Enables scalability and parallelism*
- *Distributed*
- *Replicated*

Message Format

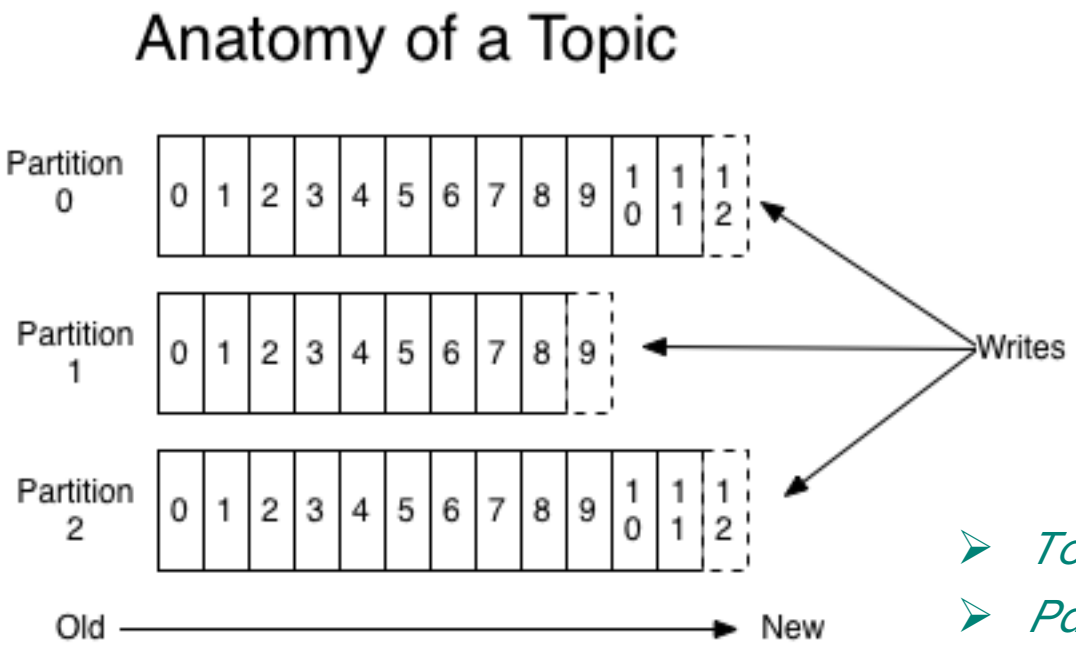
Message (v1) => Crc MagicByte Attributes Key Value

- *Crc => int32*
- *MagicByte => int8*
- *Attributes => int8*
- *Timestamp => int64*

Header

- *Key => bytes* → *Typically used to choose partition*
- *Value => bytes* → *Opaque array of bytes*

Topics and Logs

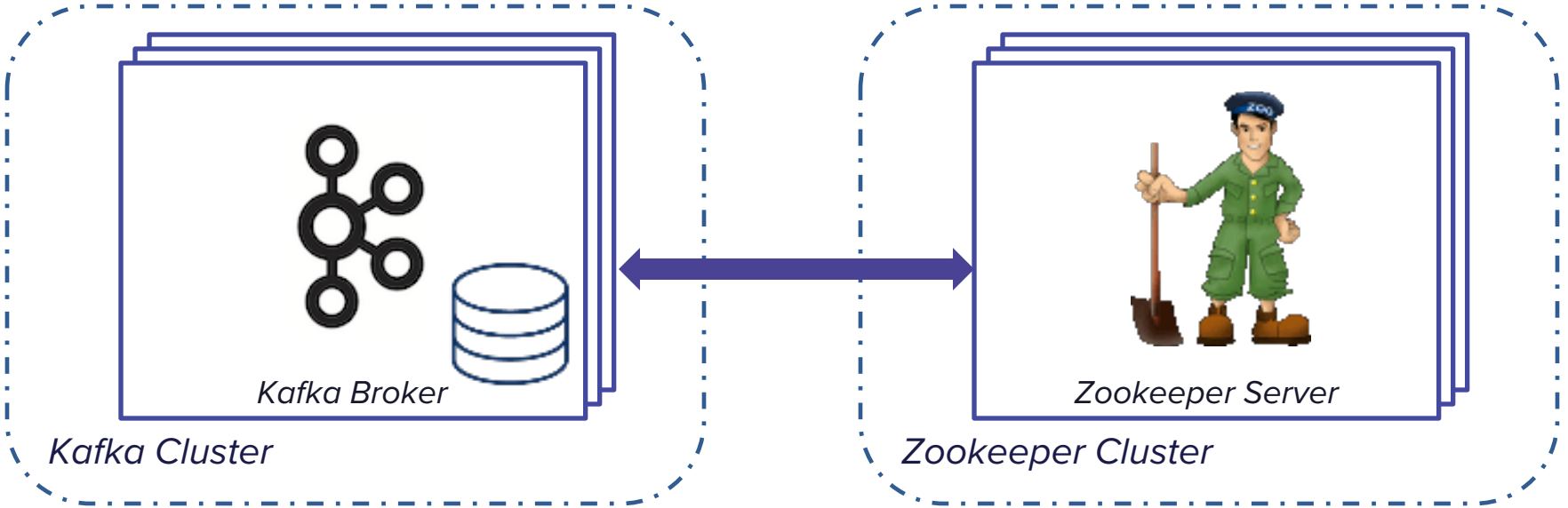


On-disk:

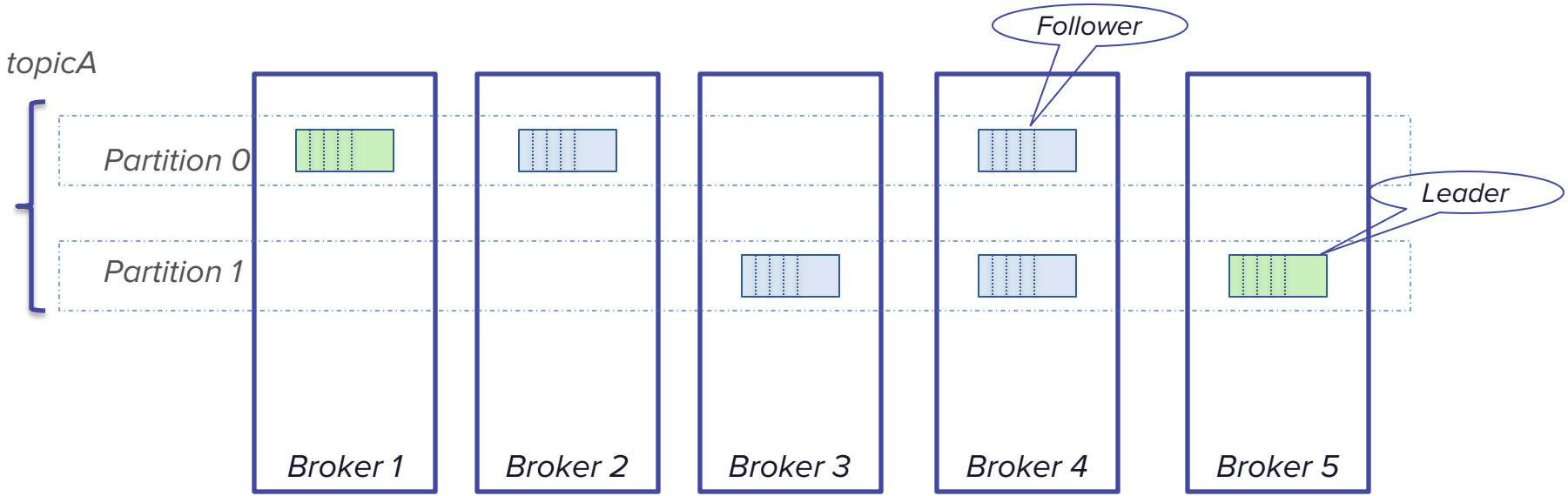
- */kafka-logs*
  - *topicA-0*
    - *00000000000000000000.index*
    - *00000000000000000000.log*
  - *topicA-1*
    - *00000000000000000000.index*
    - *00000000000000000000.log*
    - *00000000000008012974.index*
    - *00000000000008012974.log*
  - *recovery-point-offset-checkpoint*
  - *replication-offset-checkpoint*
  - *cleaner-offset-checkpoint*

- *Topics*
- *Partitions*
- *Offsets*
- *Logs*
- *Log segments*
- *Log compaction*

Kafka Cluster

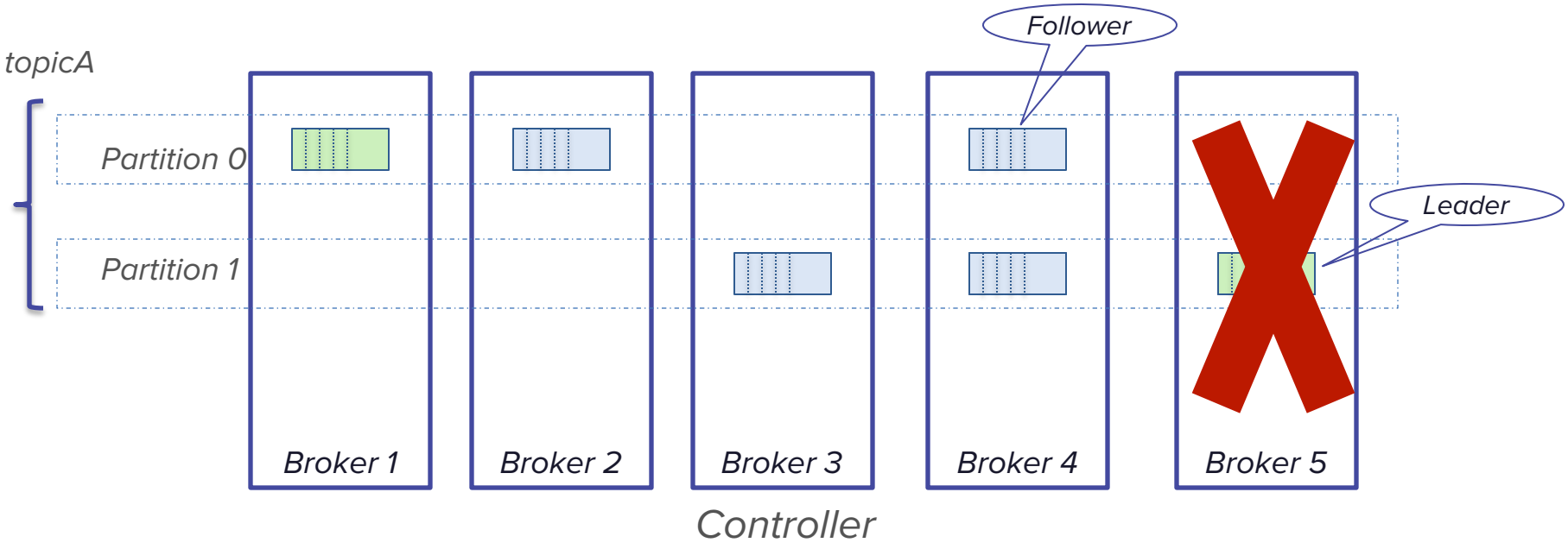


Message transmission



- *Replication*
  - *Async persistence to disk*
  - *In-sync replicas*
- *Zero-copy writes*

High Availability



- **Rolling upgrades**

Zookeeper

Messaging guarantees

- *Message ordering*
  - *Ordered partitions*
  - *Messages sent by a producer to a partition are appended to log in that order*
  - *Consumer instances see messages in the order stored in the log*
- *Delivery semantics*
  - *At least once*
  - *At most once*
  - *Exactly once – not currently supported within Kafka*

Broker configuration

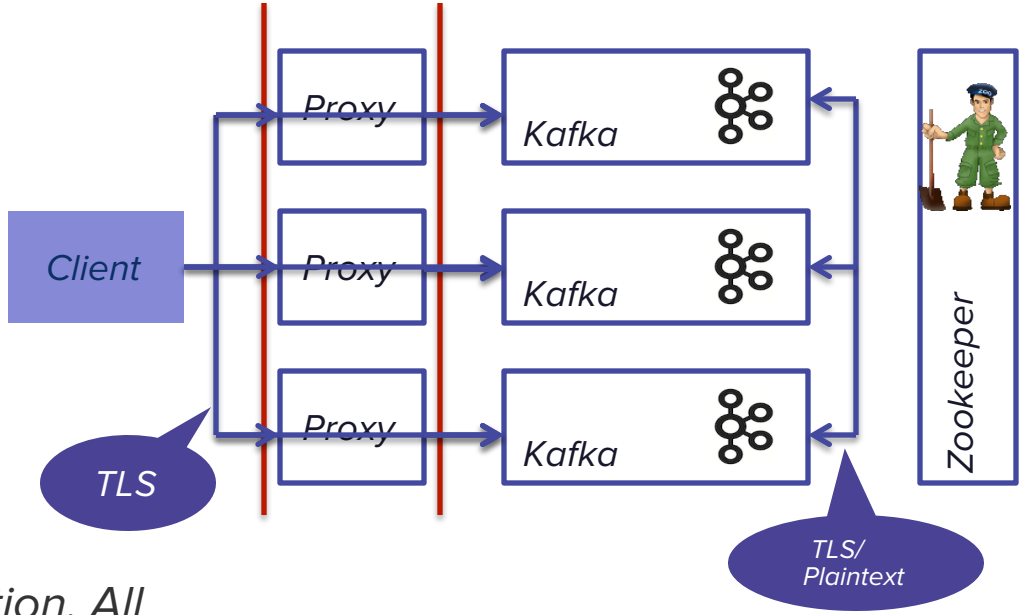
Configuration	Description
advertised.listeners	Listeners published to ZK for clients to use
auto.leader.rebalance.enable	Background thread periodically checks and triggers rebalance if required
replica.lag.time.max.ms	If follower hasn't fetched or consumed up to log end for this period, follower is removed from ISR
unclean.leader.election.enable	Indicates whether to enable replicas not in the ISR set to be elected as leader as a last resort, even though doing so may result in data loss
broker.rack	Used for rack-aware replica assignment for improved fault tolerance
etc. etc.	

Topic configuration

Configuration	Description
cleanup.policy	Compact or delete
retention.bytes	Maximum log size
retention.ms	Maximum time logs are retained (7 days by default)
max.message.bytes	Largest message size allowed for topic
segment.bytes	Maximum log segment size
min.insync.replicas	For producer acks=all, minimum number of replicas for a successful write
etc. etc.	

Security

- *TLS*
  - *Encryption*
  - *Client authentication*
- *SASL Authentication*
  - *GSSAPI/Kerberos*
  - *PLAIN*
- *Access Control*
  - *Operations: CRUD, ClusterAction, All*
  - *ACL Resources: Cluster, Topic, Consumer groups*
- *Quotas*



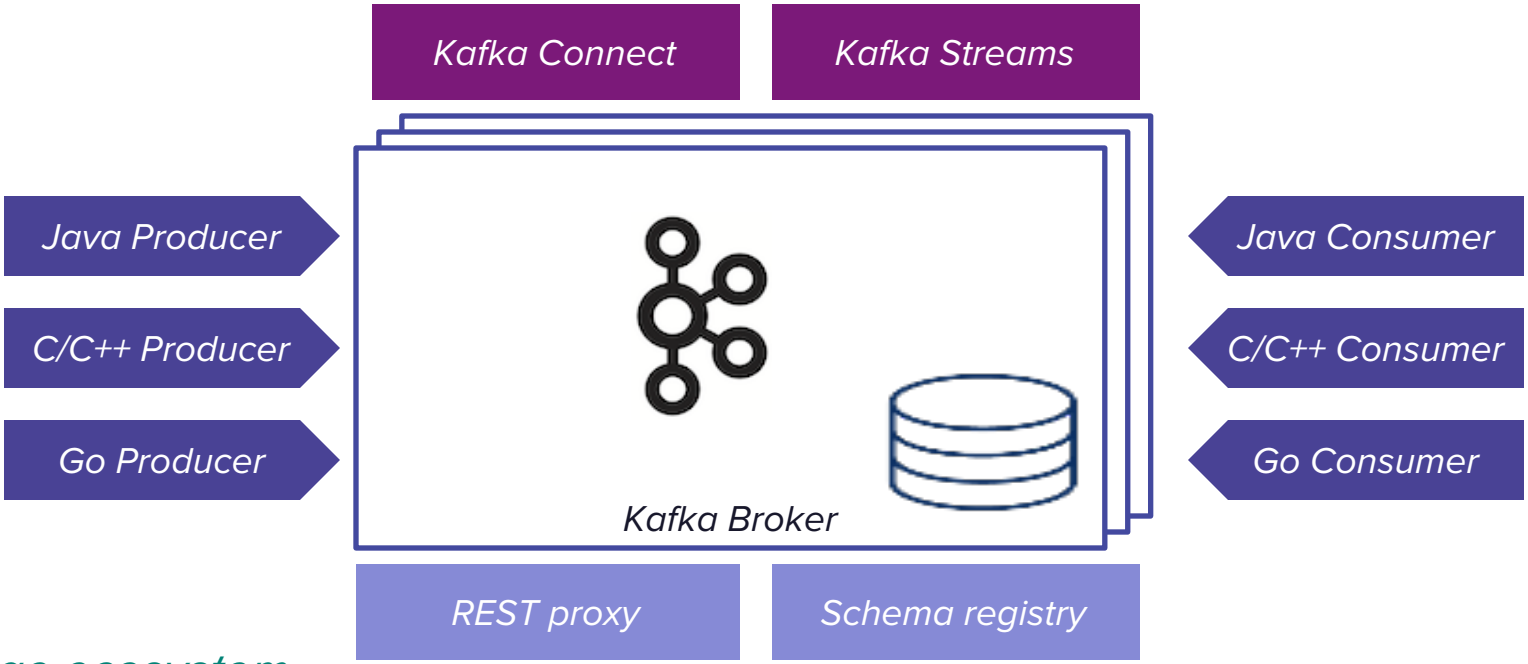
Kafka Use cases

- *Messaging*
- *Micro-service architecture*
- *Commit Log*
- *Stream processing*
- *Event sourcing*
- *See http://kafka.apache.org/documentation.html#uses for more examples*

Agenda

- *Introduction to Apache Kafka*
  - *Clients and tools for Kafka*
  - *Non-reactive Java clients*
- *Introduction to Reactive Streams*
- *Reactive Kafka*
  - *Reactive API for Kafka*
  - *Performance evaluation*
- *Summary*

Kafka clients



- *Large ecosystem of clients, tools, connectors*

Spark APACHE STORM™ Flink FLUME

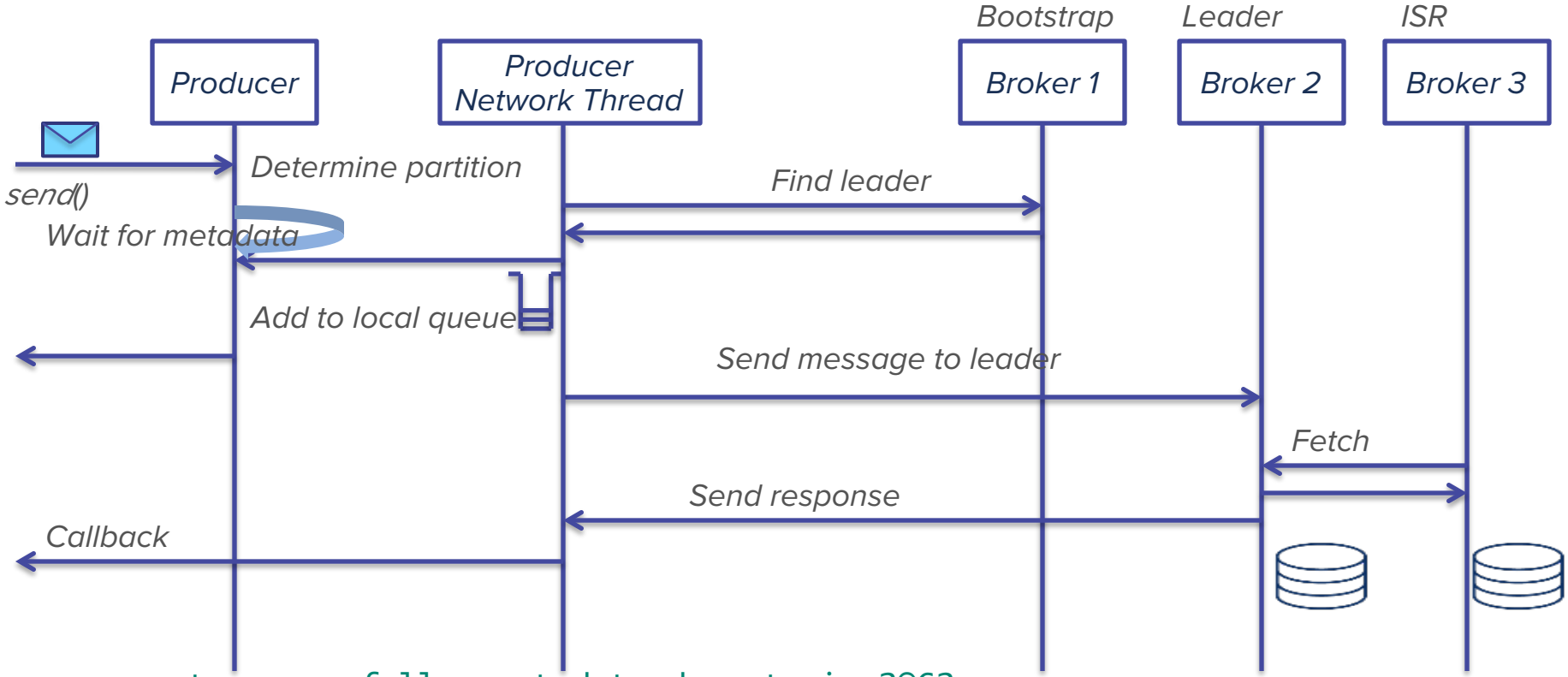
Kafka producer: non-reactive Java example

```
producer = new KafkaProducer<>(producerProps);
for (int i = 0; i < count; i++) {
  String message = "Message_" + i;
   producer.send(new ProducerRecord<>(topic, i, message), new Callback() {
     public void onCompletion(RecordMetadata metadata, Exception exception) {
       if (exception == null)
         System.out.println("Message sent successfully, metadata=" + metadata);
       else
         log.error("Send failed", exception);
     }
   });
}
```

Kafka producer: non-reactive Java 8

```
producer = new KafkaProducer<>(producerProps);
for (int i = 0; i < count; i++) {
  String message = "Message_" + i;
   producer.send(new ProducerRecord<>(topic, i, message),
        (metadata, exception) -> {
              if (exception == null)
                System.out.println("Message sent successfully, metadata=" +
metadata);
              else
                log.error("Send failed", exception);
         });}
```

Producer sequence diagram



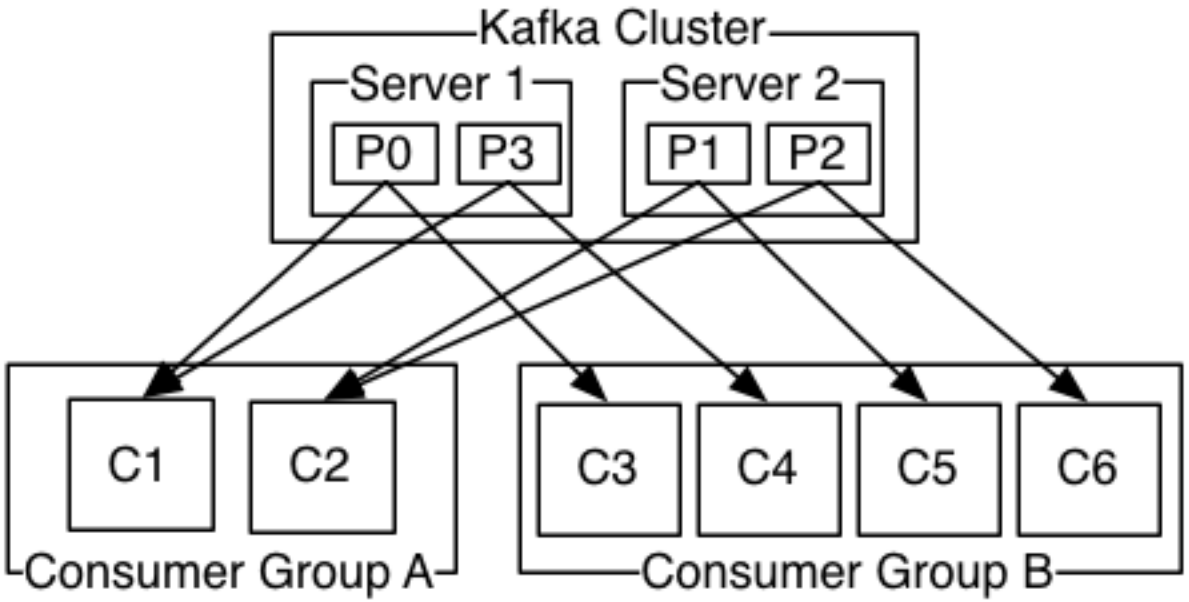

```
Message sent successfully, metadata=demo-topic-2@62
```

Producer configuration

Configuration	Description
acks	0, 1, all
buffer.memory	Total memory for local buffer
max.block.ms	Max block time to wait for metadata or buffer space
retries	Automatic retries
max.in.flight.requests.per.connection	Max number of unacked requests on a connection
batch.size	Max size for batch of records sent to broker
linger.ms	Time to wait for batch
etc. etc.	

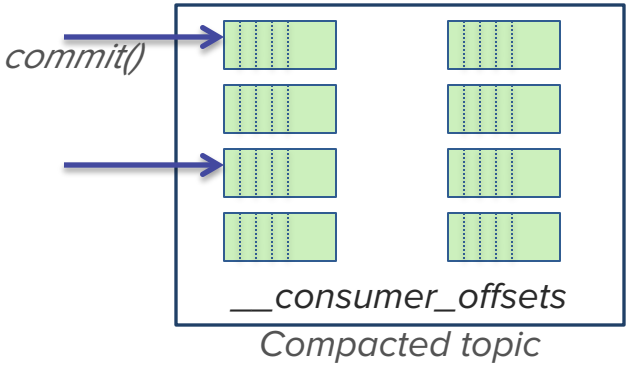
Kafka Consumer

- *Messaging models*
  - *Publish-subscribe*
  - *Point-to-point*



Consuming from Kafka

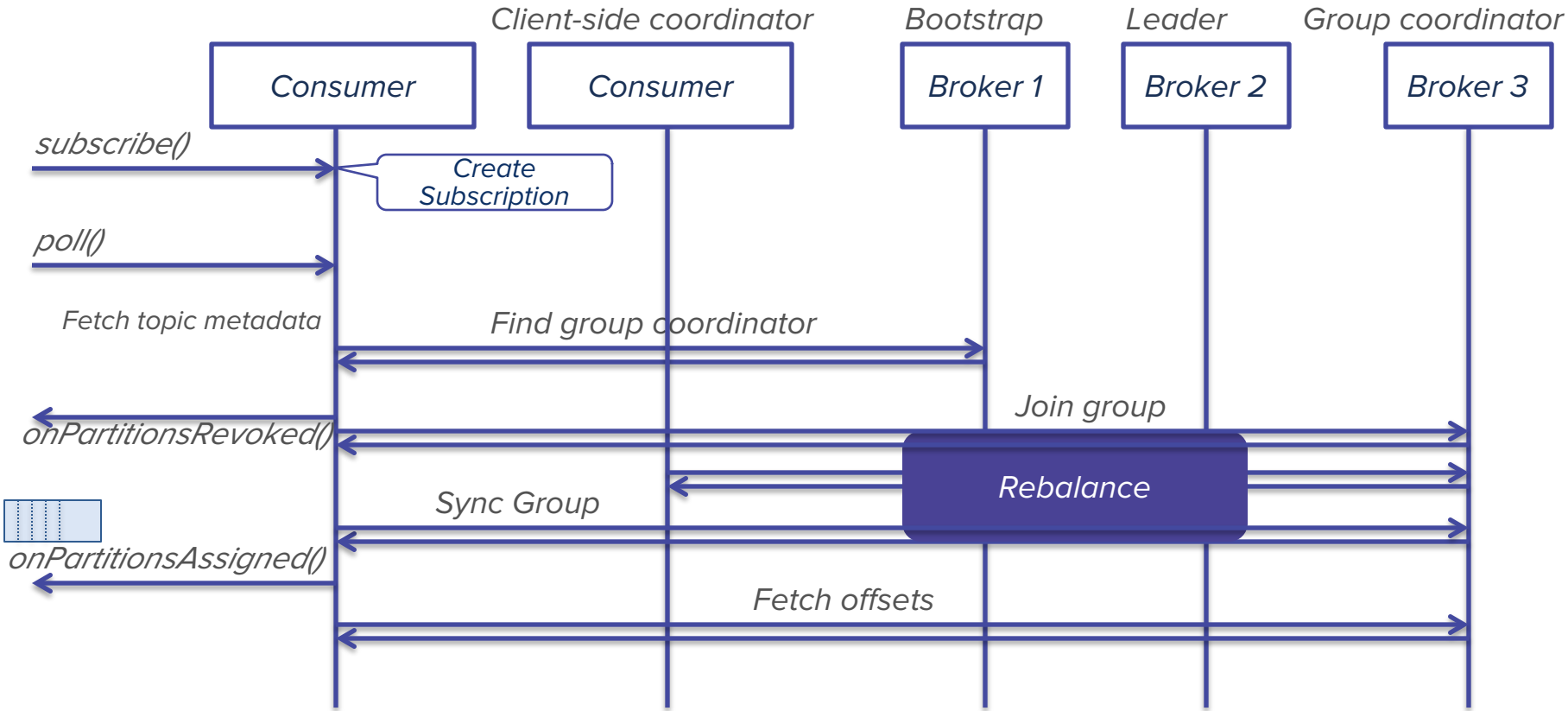
- *Push vs Pull*
  - *Push*
  - *Pull*
- *Kafka*
  - *Producers push messages, consumers pull messages*
  - *Long poll*
- *Offset management*
  - *Auto-commit*
  - *Manual commit*
  - *External offset management*



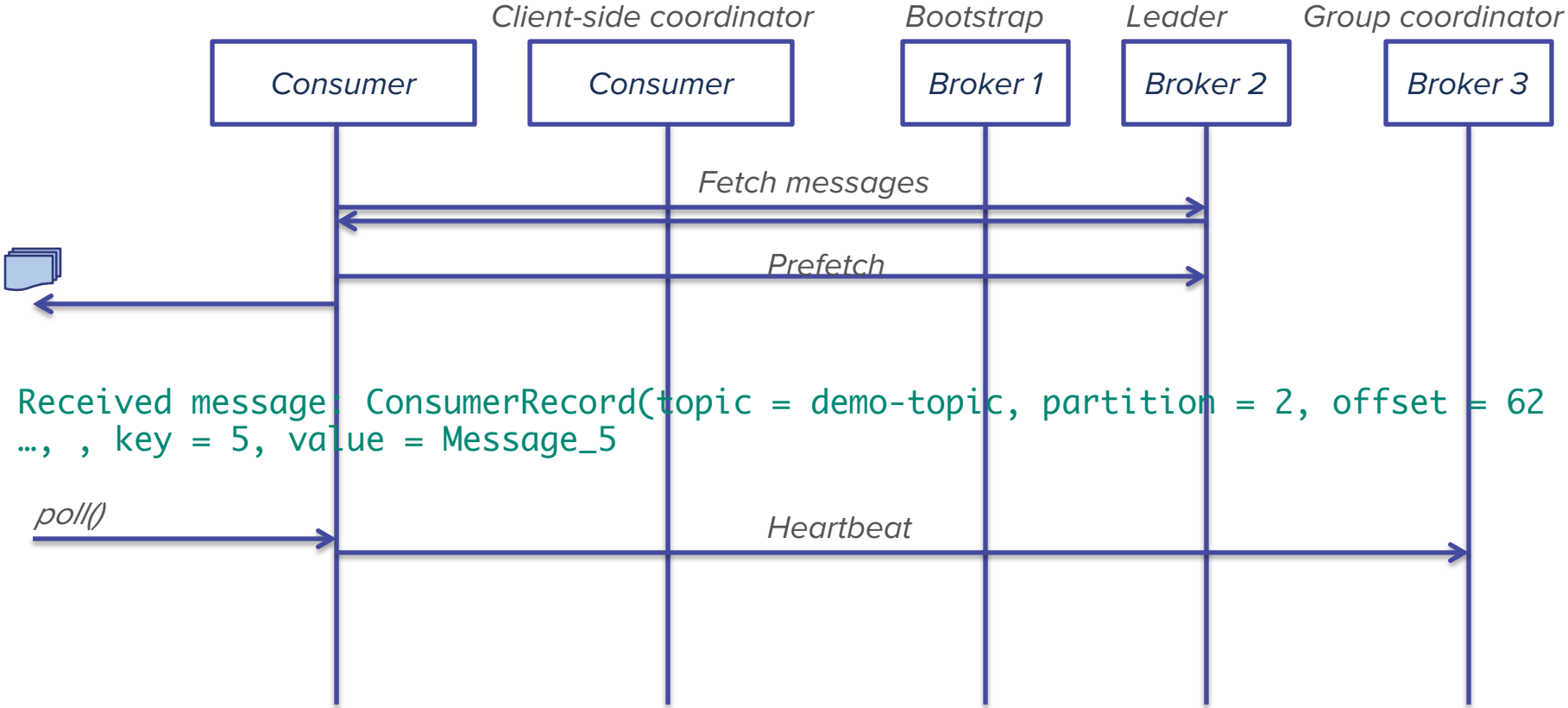
Kafka consumer: non-reactive Java example

```
consumer.subscribe(topics, new ConsumerRebalanceListener() {
        public void onPartitionsAssigned(Collection<TopicPartition> partitions) {}
        public void onPartitionsRevoked(Collection<TopicPartition> partitions) {}
    });
while (received < count) {
    ConsumerRecords<Integer, String> records = consumer.poll(1000); // 1 second
    for (ConsumerRecord<Integer, String> record : records) {
        System.out.println("Received message: " + record);
        received++;
    }
}
```

Consumer sequence diagram



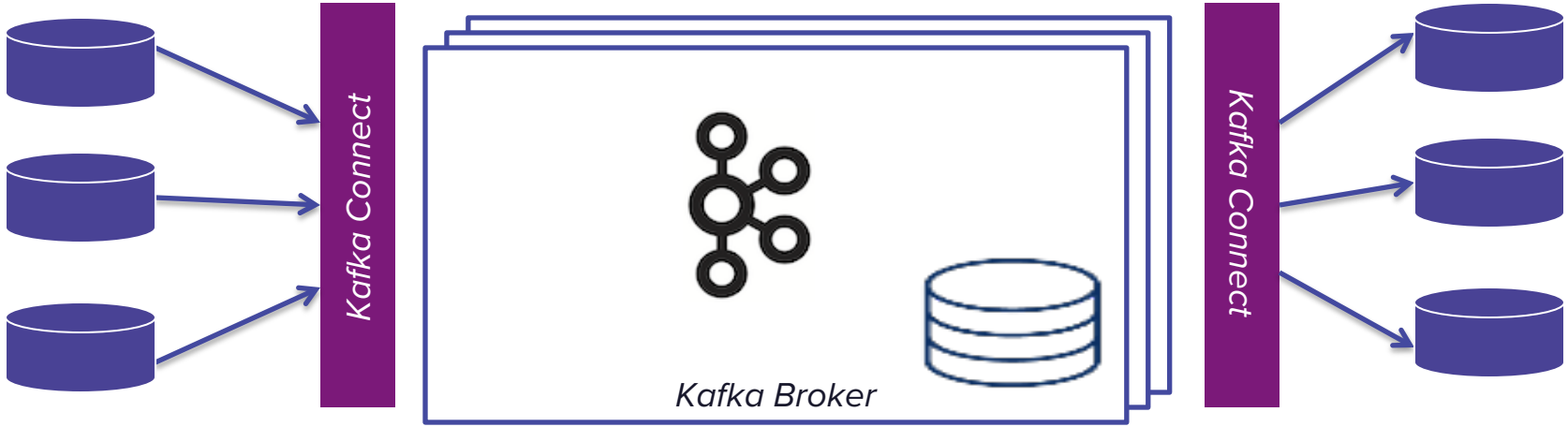
Consumer sequence diagram



Consumer configuration

Configuration	Description
group.id	Consumer group
fetch.min.bytes	Min bytes to wait for per fetch
max.partition.fetch.bytes	Max bytes per partition per fetch
fetch.max.wait.ms	Max time server blocks before sending response
max.poll.records	Max messages per poll
heartbeat.interval.ms	Keeps session active and facilitates group rebalancing
session.timeout.ms	Timeout to detect consumer instance failure
etc. etc.	

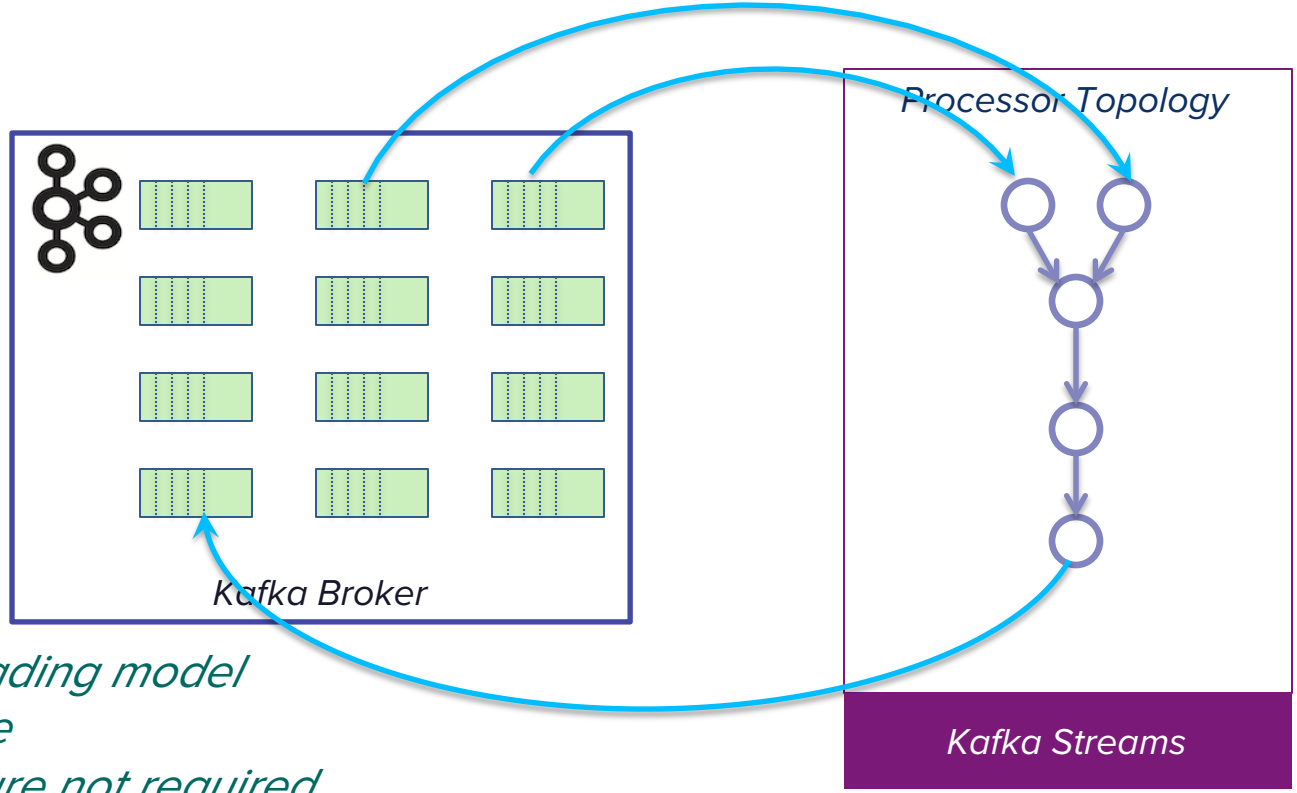
Kafka Connect



External
Data Sources

External
Data Sinks

Kafka Streams



- *Simple threading model*
- *Non-reactive*
- *Back-pressure not required*

Agenda

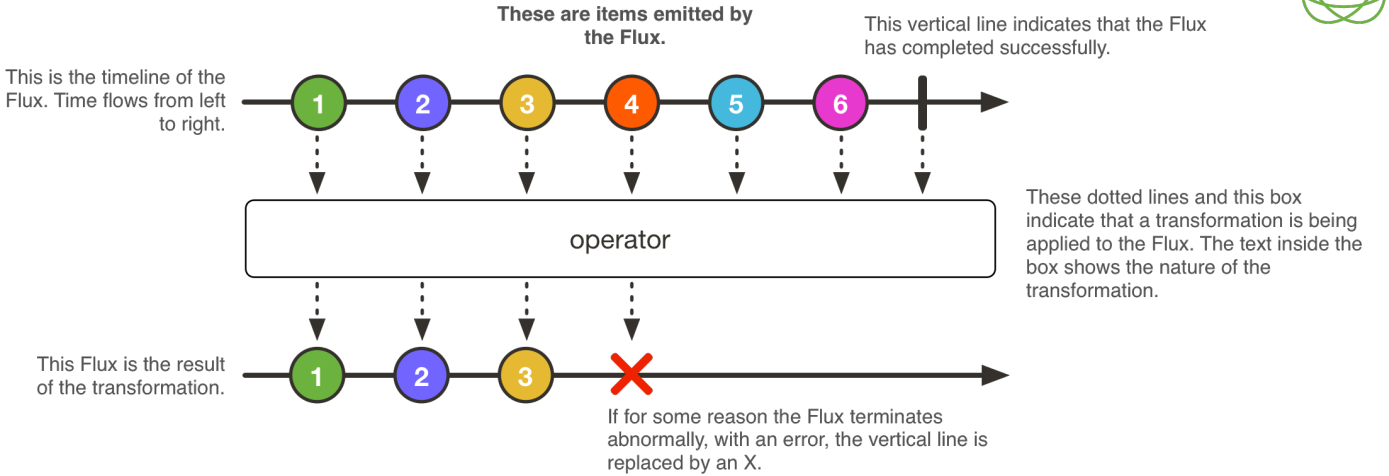
- *Introduction to Apache Kafka*
  - *Clients and tools for Kafka*
  - *Non-reactive Java clients*
- *Introduction to Reactive Streams*
- *Reactive Kafka*
  - *Reactive API for Kafka*
  - *Performance evaluation*
- *Summary*

Reactive Streams

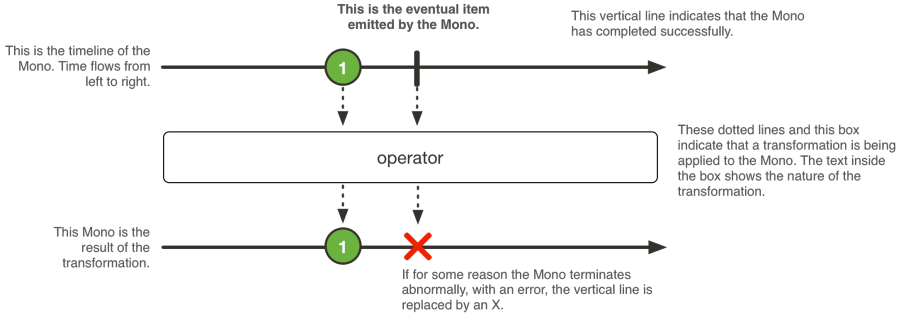
- *Fundamental shift from imperative-style to asynchronous, non-blocking functional-style code*
  - *Serve more heterogeneous requests concurrently*
  - *Handle remote operations more efficiently*
  - *Composable*
- *Back-pressure*
  - *Consumer requests data from emitter*
  - *Start receiving data only when consumer is ready to process data*
  - *Control the amount of inflight data*

Project Reactor

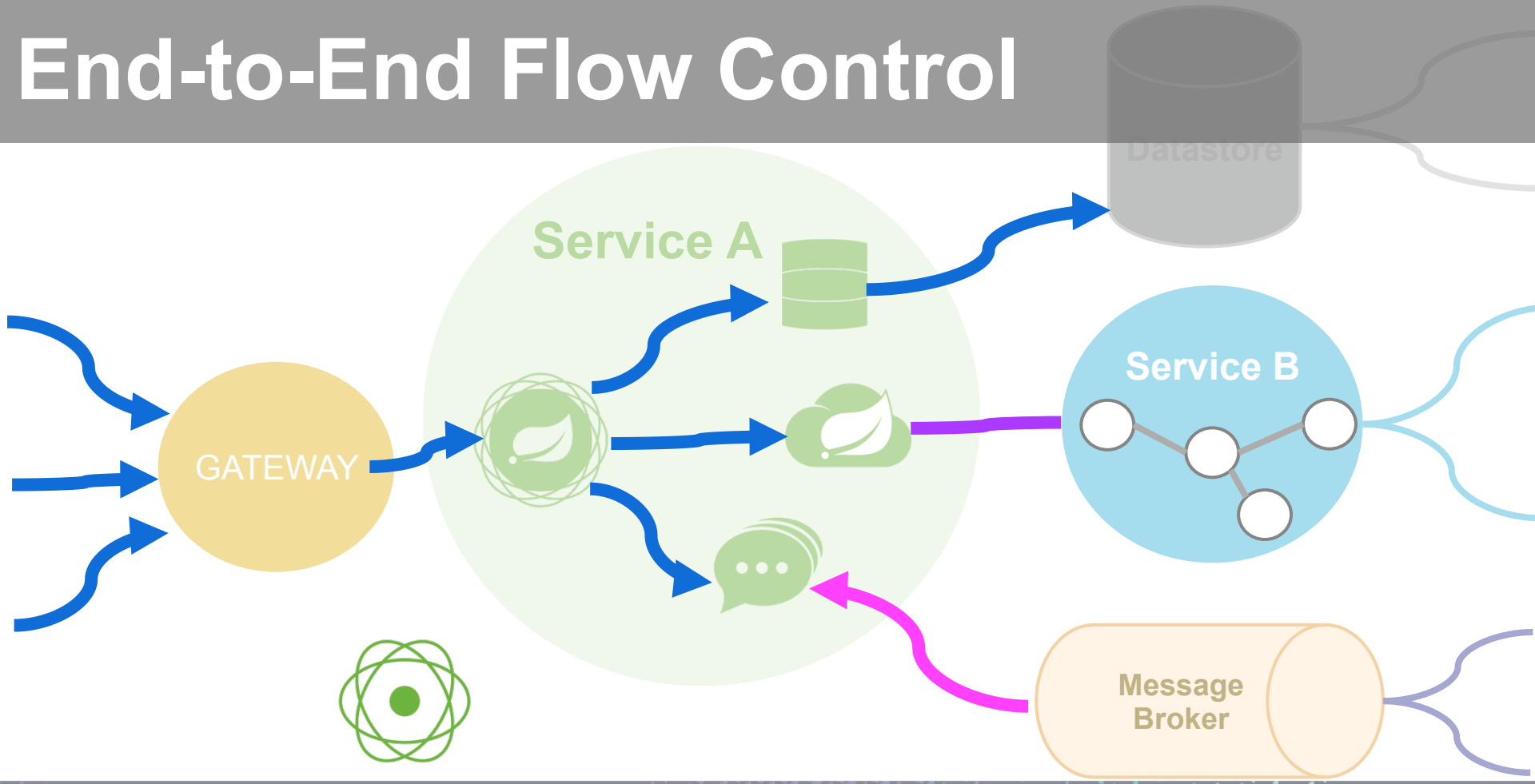
Flux



Mono

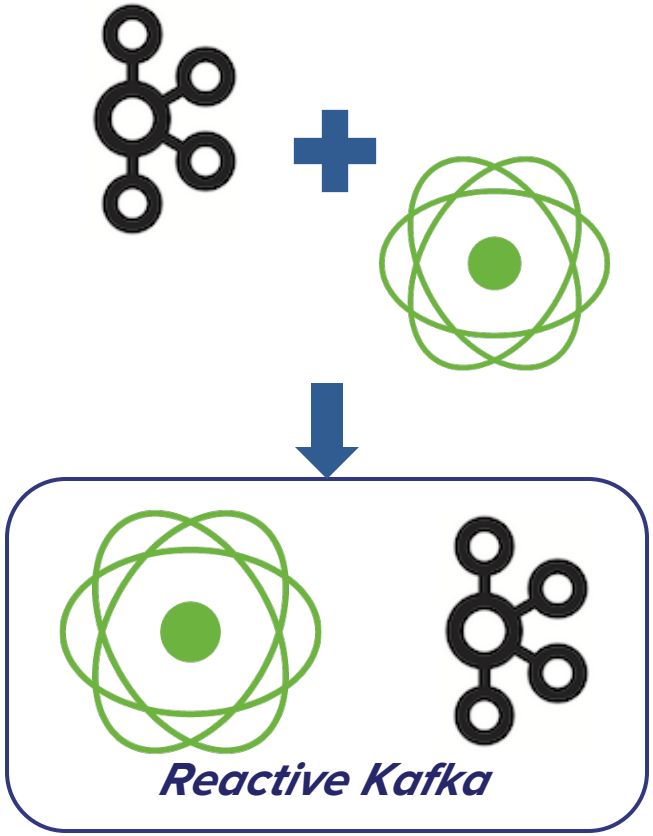


End-to-End Flow Control



Agenda

- *Introduction to Apache Kafka*
  - *Clients and tools for Kafka*
  - *Non-reactive Java clients*
- *Introduction to Reactive Streams*
- *Reactive Kafka*
  - *Reactive API for Kafka*
  - *Performance evaluation*
- *Summary*



Reactive Kafka

- *Bringing reactive streams and Kafka together*
  - *Functional-style reactive API for Kafka*
  - *Back-pressure*
  - *Composable API*
- *Implementation*
  - *As a shim over non-reactive Java API for Kafka*

Reactive Kafka API

KafkaSender:

```
public static <K, V> KafkaSender<K, V> create(SenderConfig<K, V> config) {}
public Mono<RecordMetadata> send(ProducerRecord<K, V> record) {}
```

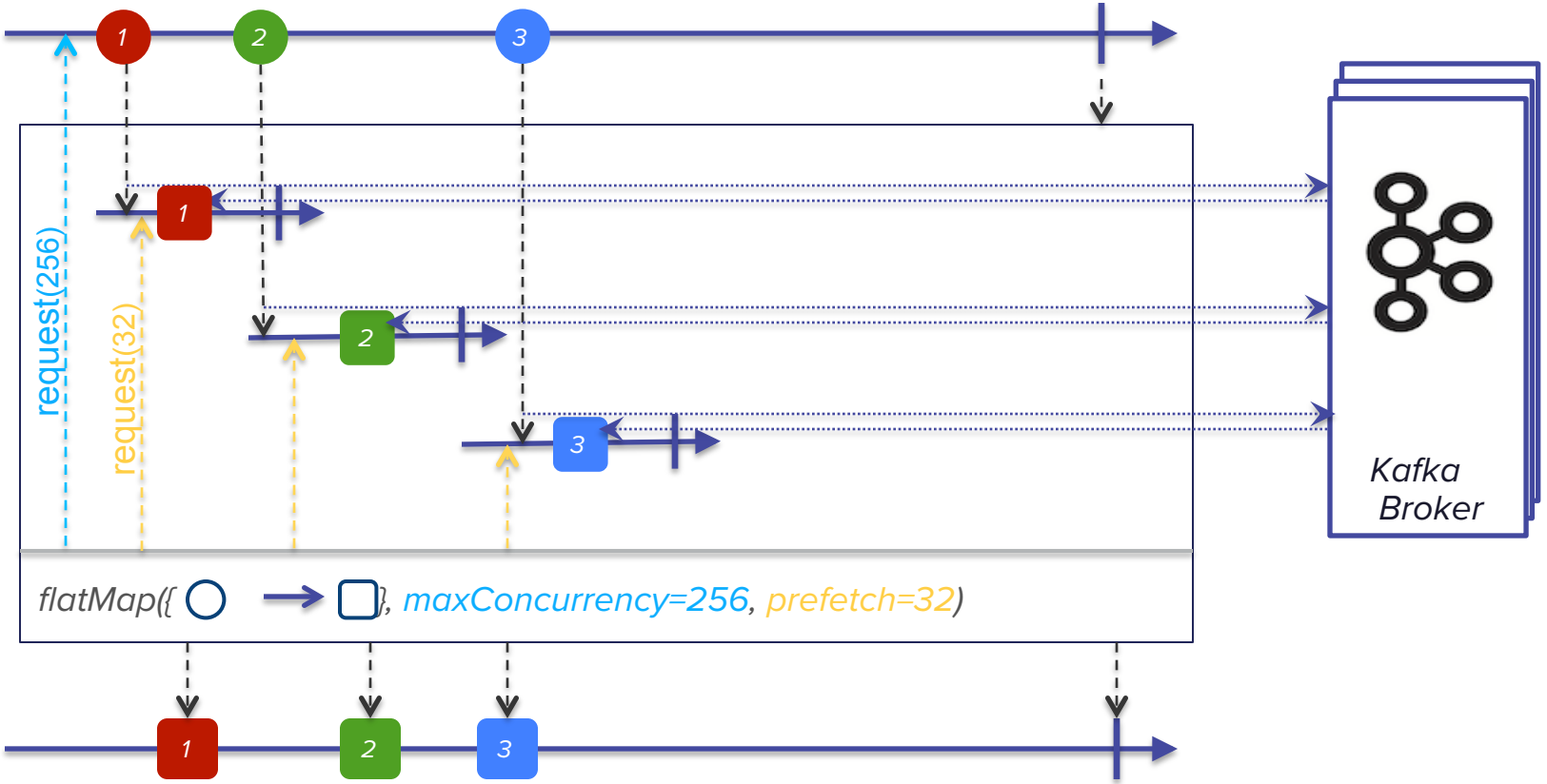
KafkaFlux:

```
public static <K, V> KafkaFlux<K, V>
        listenOn(FluxConfig<K, V> config, Collection<String> topics) {}
public enum AckMode {
      AUTO_ACK, ATMOST_ONCE, MANUAL_ACK, MANUAL_COMMIT
 }
```

Reactive Kafka Producer

```
KafkaSender<Integer, String> sender = KafkaSender.create(senderConfig);
Flux<RecordMetadata> flux = Flux.range(1,  count)
      .flatMap(i -> sender.send(new ProducerRecord<>(topic, i, "Message_" + i)));
      .doOnNext(metadata -> {
          log.info("Message sent successfully, metadata=" + metadata);
          latch.countDown();
       })
     .doOnError(e-> log.error("Send failed", e))
…
flux.subscribe();
```

Reactive KafkaSender sequence



Producer sample log (edited for readability)

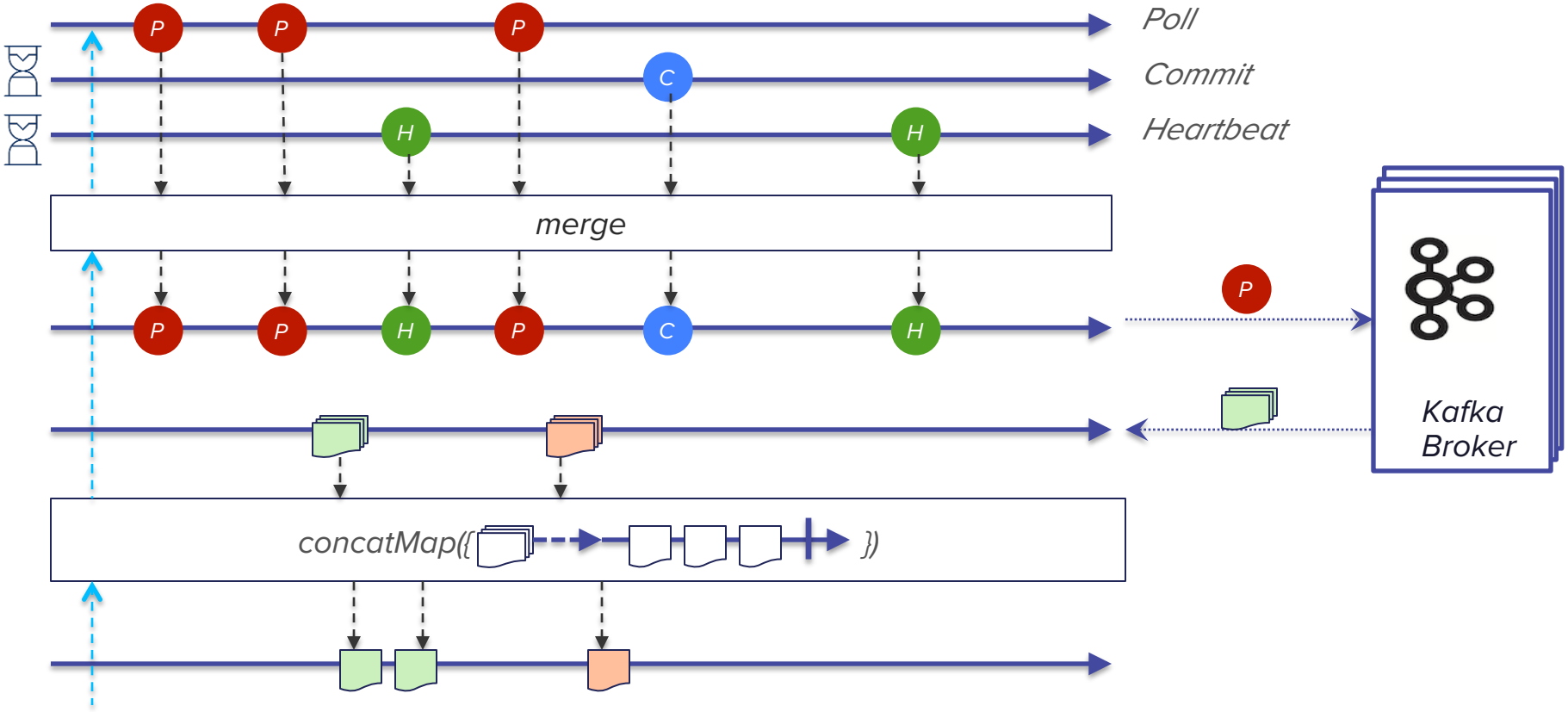
```
[main]  onSubscribe(r.c.publisher.FluxRange$RangeSubscription)
[main]  onSubscribe(r.c.publisher.FluxFlatMap$FlatMapMain)
[main]  request(unbounded)
[main]  request(256)
[main]  onNext(1)
[main]  ProducerConfig values:
[main]  onNext(2) ....
[single-1] onNext(demo-topic-2@64)
[single-1] Message sent successfully, metadata=demo-topic-2@64
```

Kafka producer created

Reactive Kafka Consumer

```
KafkaFlux<Integer, String> kafkaFlux =
    KafkaFlux.listenOn(fluxConfig, Collections.singleton(topic))
        .doOnPartitionsAssigned(partitions -> log.debug("assigned {}", partitions))
        .doOnPartitionsRevoked(partitions -> log.debug("revoked {}", partitions));
kafkaFlux.subscribe(message -> {
        ConsumerOffset offset = message.consumerOffset();
        ConsumerRecord<Integer, String> record = message.consumerRecord();
        log.info("Received message: {} commitOffset: {}", record.value(), offset);
        latch.countDown();
    });
```

Reactive KafkaFlux sequence



Consumer sample log (edited for readability)

```
[main]  onSubscribe(r.c.publisher.EmitterProcessor$EmitterSubscriber)
[main]  onSubscribe(r.c.publisher.FluxFlatMap$FlatMapMain)
[main]  request(256)
[main]  onNext(reactor.kafka.internals.FluxManager$InitEvent)
[main]  request(unbounded)
[main]  onNext(r.k.internals.FluxManager$PollEvent)
[sample-group-1] ConsumerConfig values:
[sample-group-1] Discovered coordinator … group sample-group.
[sample-group-1] (Re-)joining group sample-group
[sample-group-1] onPartitionsAssigned [demo-topic-2, demo-topic-1]
[parallel-1]  onNext(org.apache.kafka.clients.consumer.ConsumerRecords)
[parallel-1] ConsumerRecord(topic = demo-topic, partition = 2, offset = 64,
…, key = 5, value = Message_5)
[parallel-1] Received message: Message_5 commitOffset: demo-topic-2@65
```

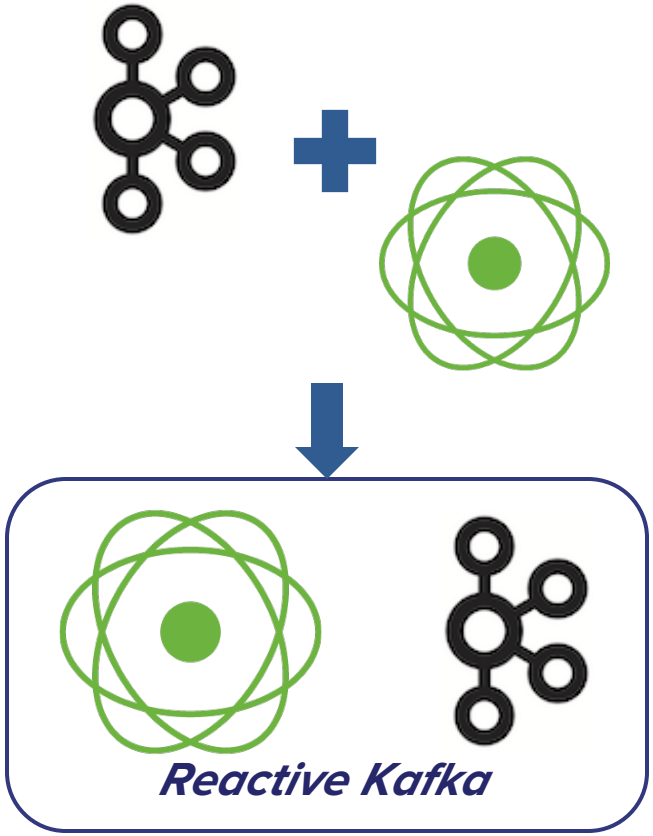
Reactive consumer configuration

- *All non-reactive producer and properties configurable for reactive clients*

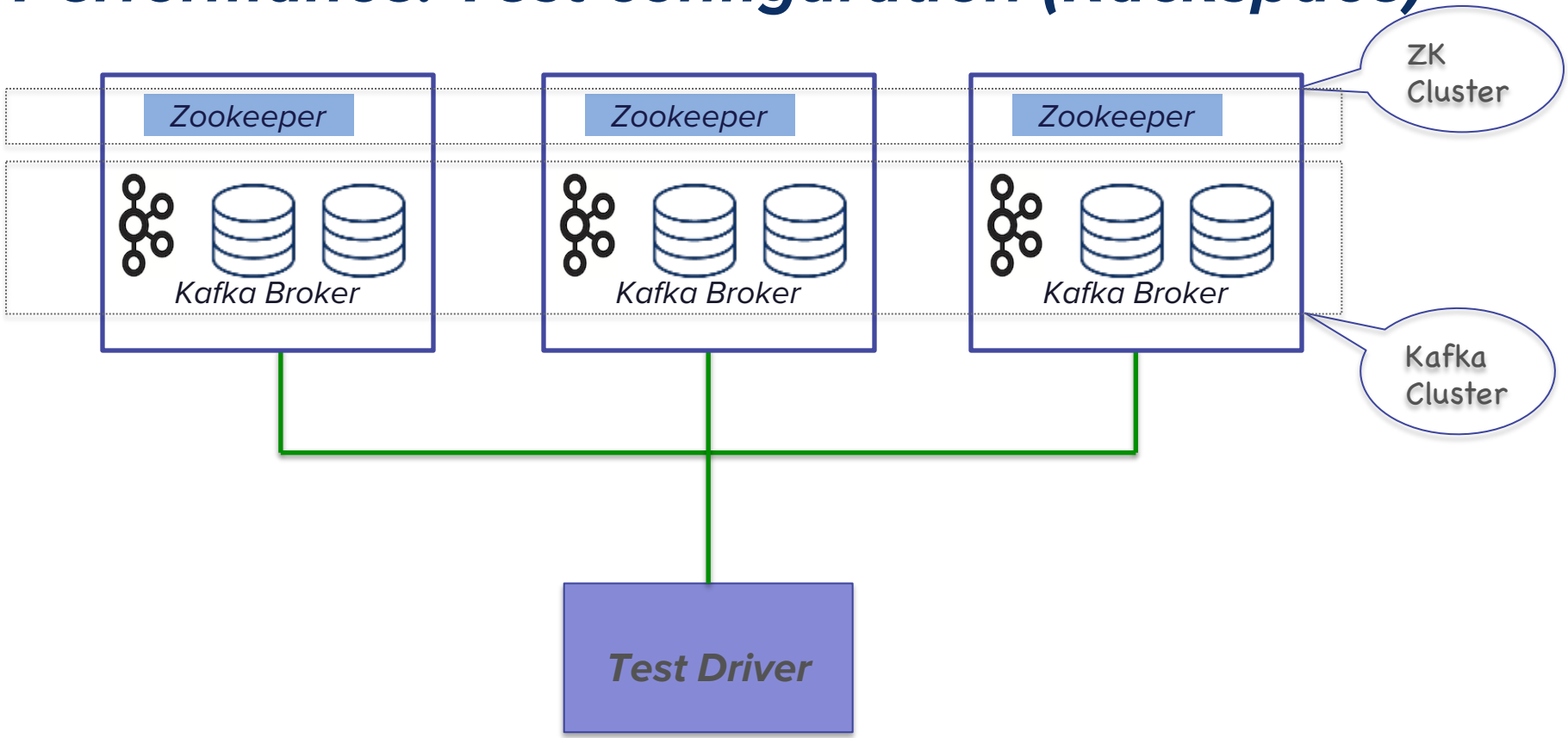
Configuration	Description
`pollTimeout`	*Timeout specified in each consumer.poll()*
`commitInterval`	*Duration between auto commits*
`commitBatchSize`	*Maximum batch size before auto commit is triggered*
`maxAutoCommitAttempts`	*Number of consecutive attempts to auto commit before raising an error*
`closeTimeout`	*Timeout for graceful close (for both producer and consumer)*

Agenda

- Introduction to Apache Kafka
  - Clients and tools for Kafka
  - Non-reactive Java clients
- Introduction to Reactive Streams
- Reactive Kafka
  - Reactive API for Kafka
  - Performance evaluation
- Summary



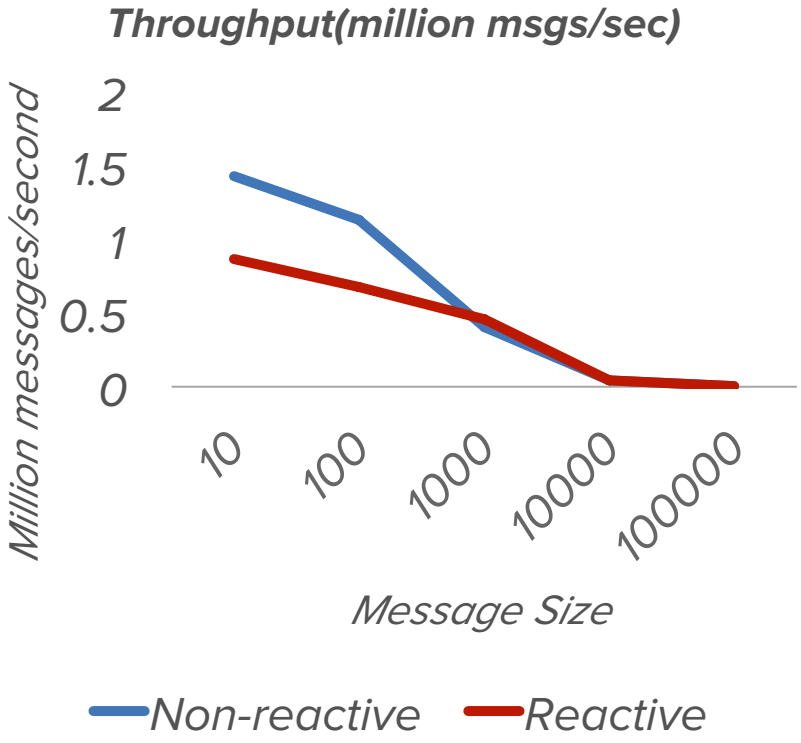
Performance: Test configuration (Rackspace)

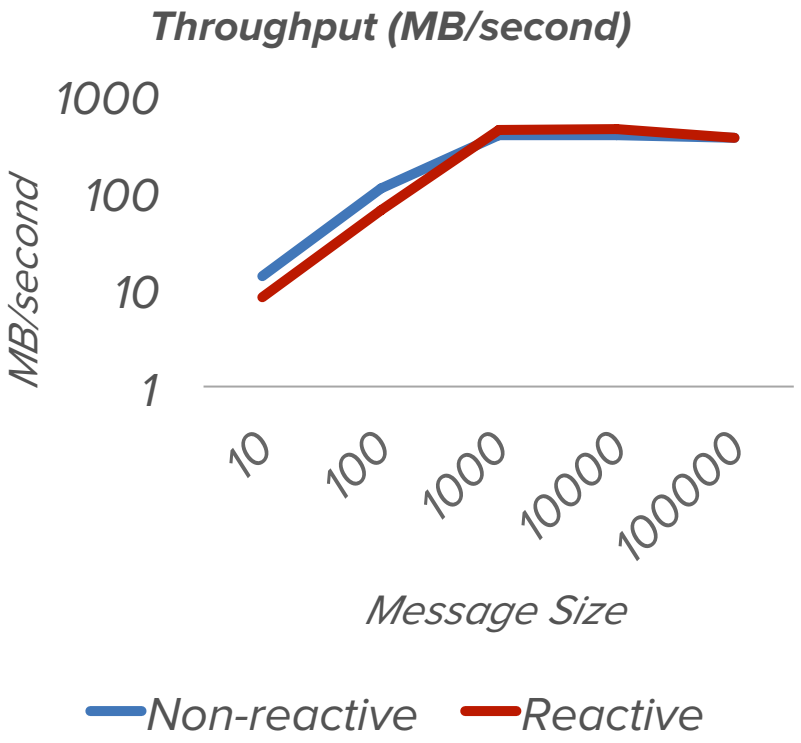


Rackspace machine configuration

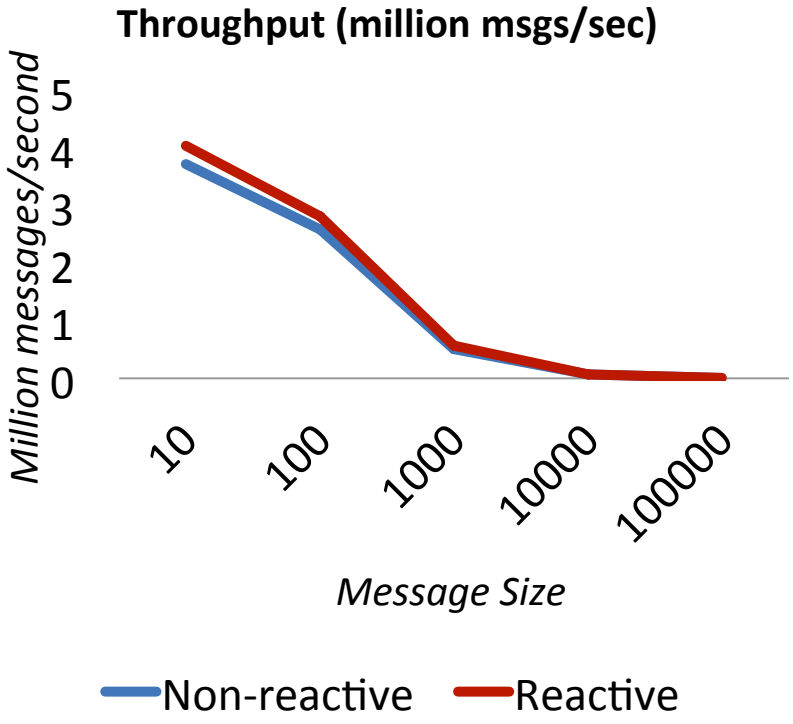
- *OnMetal I/O v2 instances for Kafka*
  - *CPU: Dual 2.6 GHz, 10 core Intel® Xeon® E5-2660 v3*
  - *RAM: 128 GB*
  - *Storage: Dual 1.6 TB PCIe flash cards*
  - *Network: Redundant 10 Gb / s connections in a high availability bond*
- *Bare Metal Compute instances for Test driver and HTTP proxy*
  - *CPU: Dual 2.4 Ghz, 6 core Intel® Xeon® E5-2620 v3*
  - *RAM: 64 GB*
  - *Network: Redundant 10 Gb / s connections in a high availability bond*

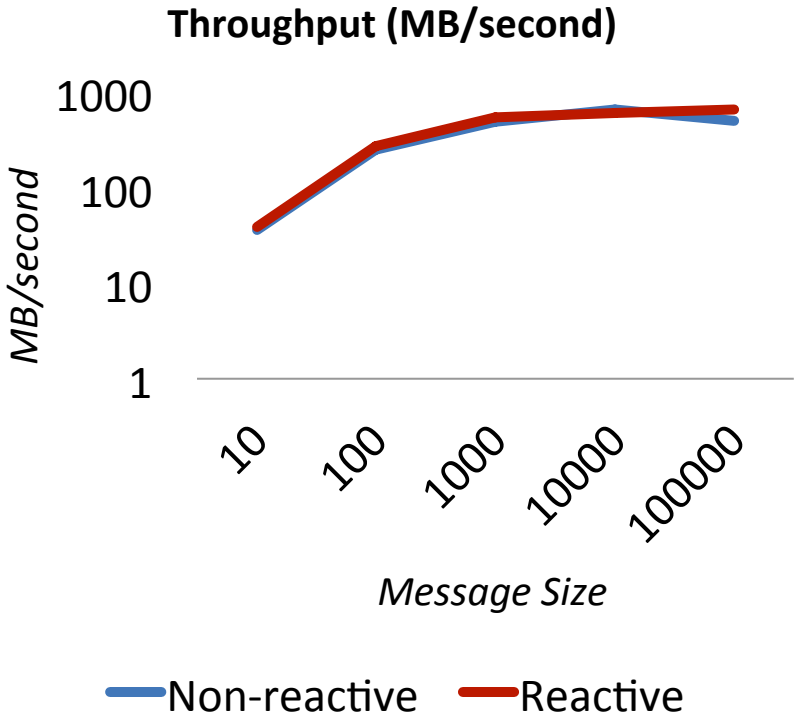
Producer performance



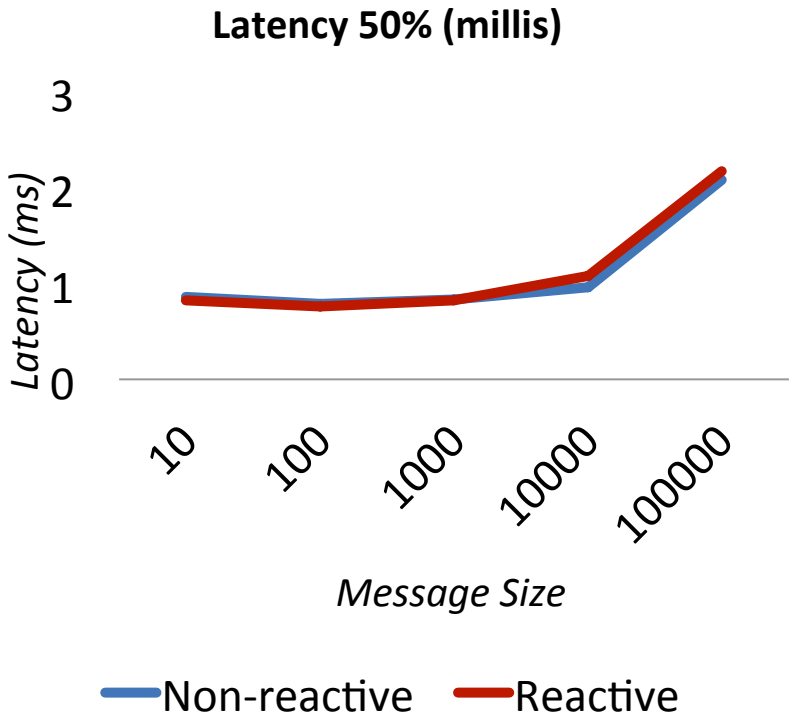


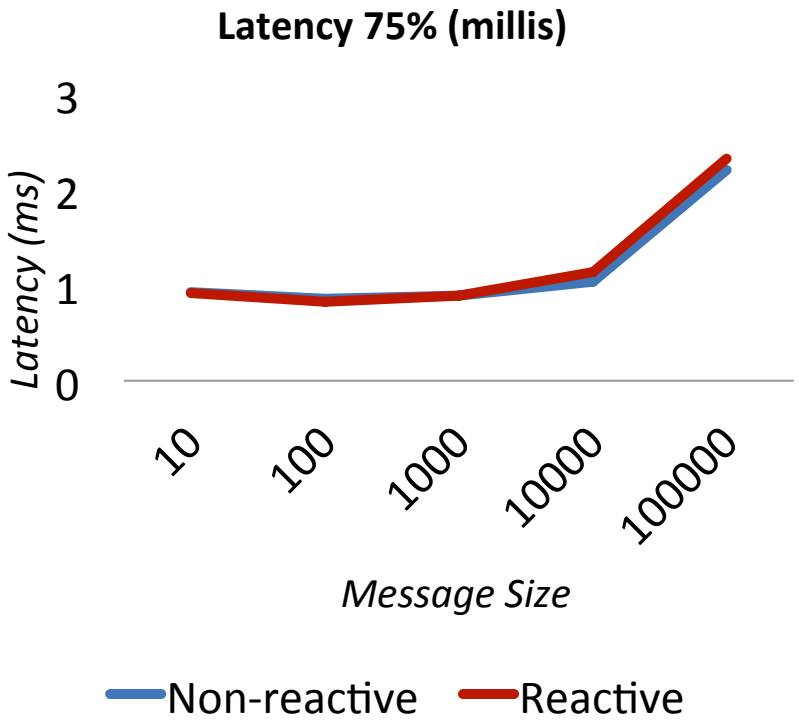
Consumer Performance



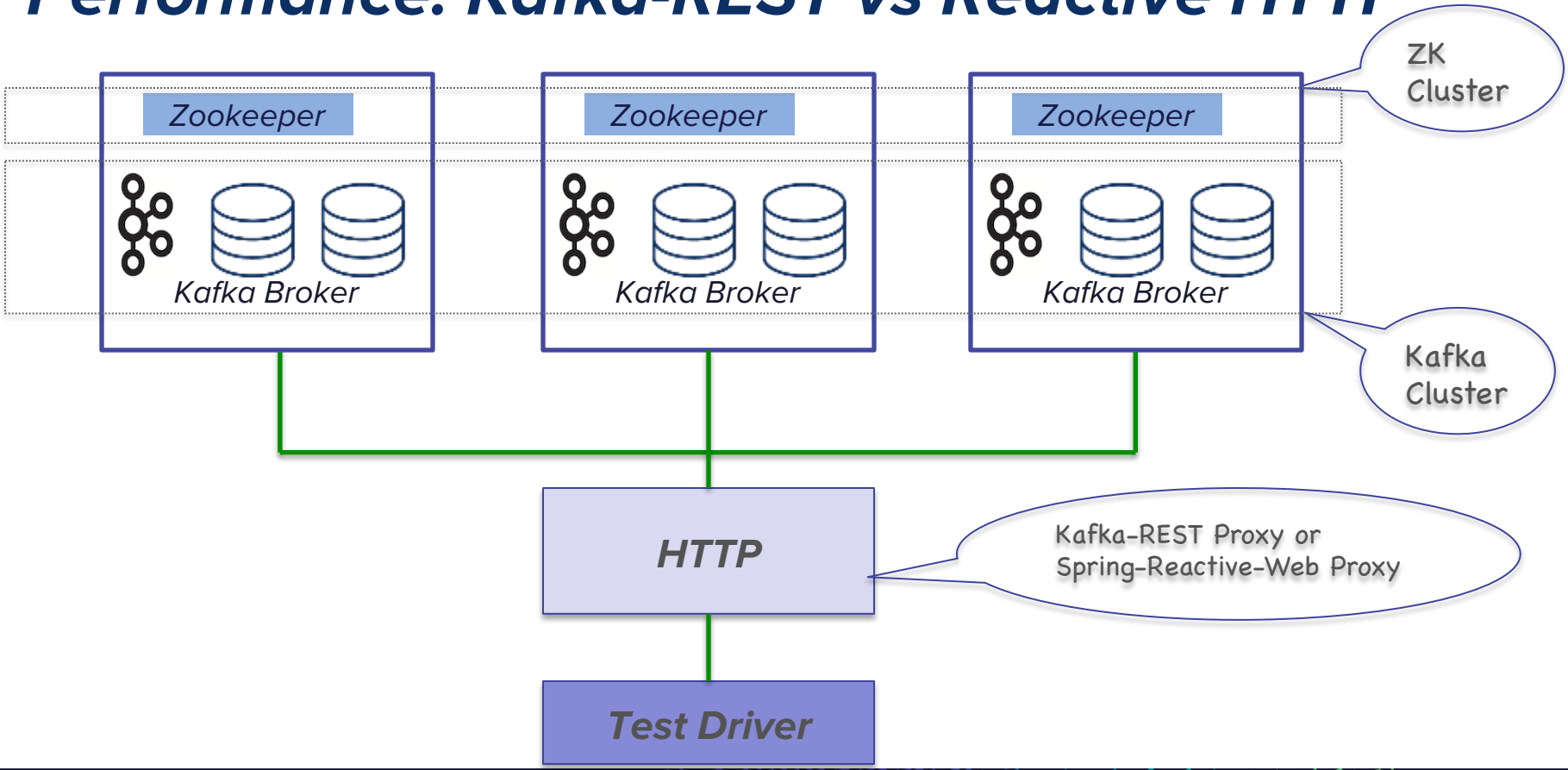


End-to-End Latency





Performance: Kafka-REST vs Reactive HTTP



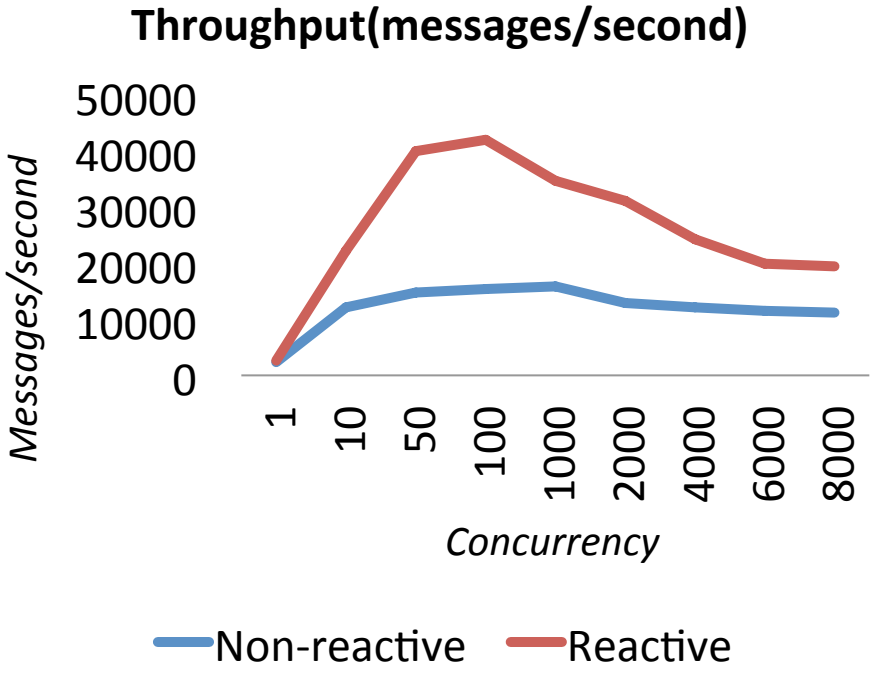
Spring Reactive Web Framework HTTP->Kafka

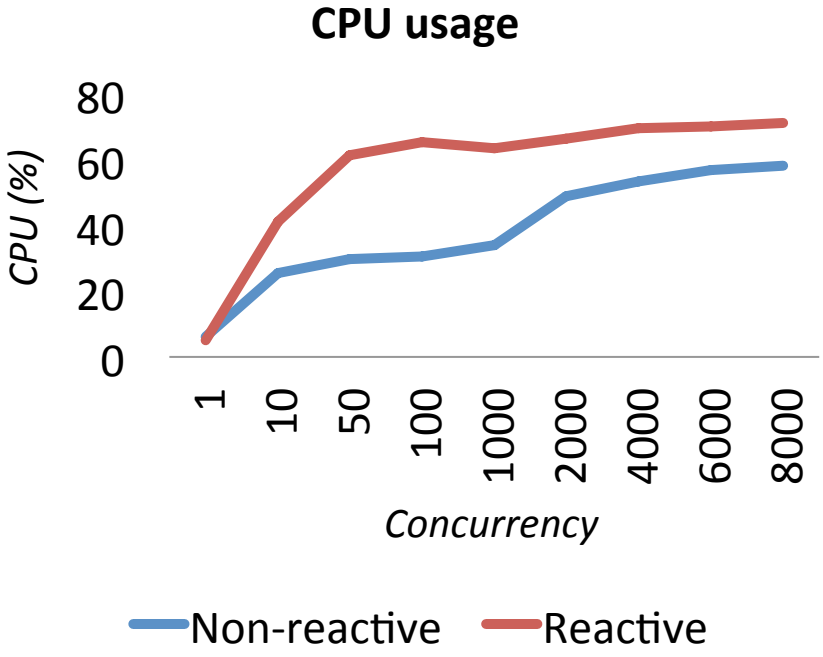
```
@Profile("kafkahttp")
@RestController
public class KafkaHttpController {
private final ReactiveKafkaHttpSender sender;
public KafkaHttpController(ReactiveKafkaHttpSender sender) {
  this.sender = sender;
}
@RequestMapping(path= "/kafkahttp/{topic}", method = RequestMethod.POST)
public Flux<SendResponse> sendToKafka(@PathVariable String topic,
@RequestBody Flux<Records> binaryStream) {
  return this.sender.sendToKafka(topic, binaryStream)
              .map(metadata -> new SendResponse(metadata)); }
```

HTTP->Kafka

```
@Profile("kafkahttp”)
public class ReactiveKafkaHttpSender {
    private final KafkaSender<String, byte[]> kafkaSender;
    public ReactiveKafkaHttpSender(KafkaSender<String, byte[]> kafkaSender) {
        this.kafkaSender = kafkaSender;
    }
    public Flux<RecordMetadata> sendToKafka(String topic, Publisher<Records>
recordStream) {
        return Flux.from(recordStream)
                .concatMap(records -> Flux.fromArray(records.getRecords()))
                .concatMap(r -> kafkaSender.send(new ProducerRecord<>(topic,
r.getValue())));    }
```

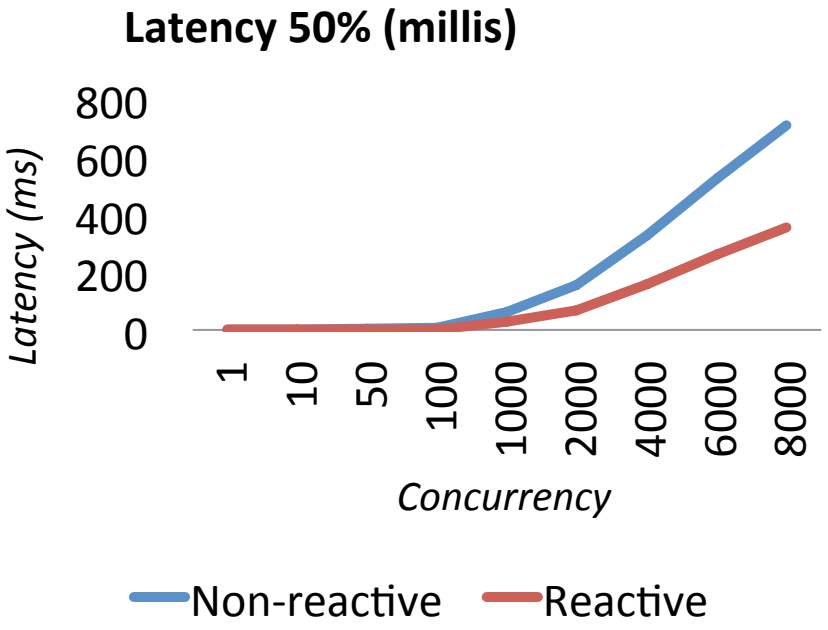
HTTP->Kafka: Kafka-REST vs Reactive

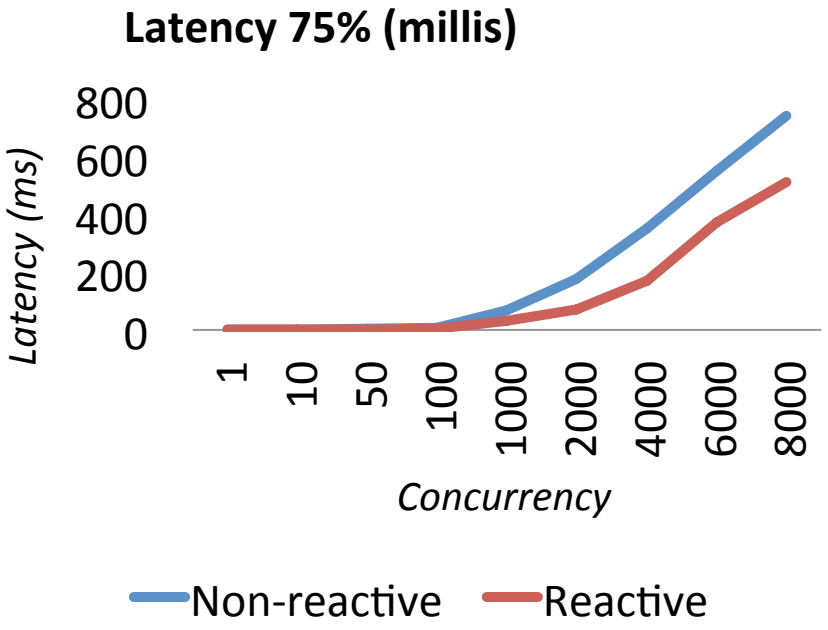




- *Higher throughput, better CPU utilization with reactive*

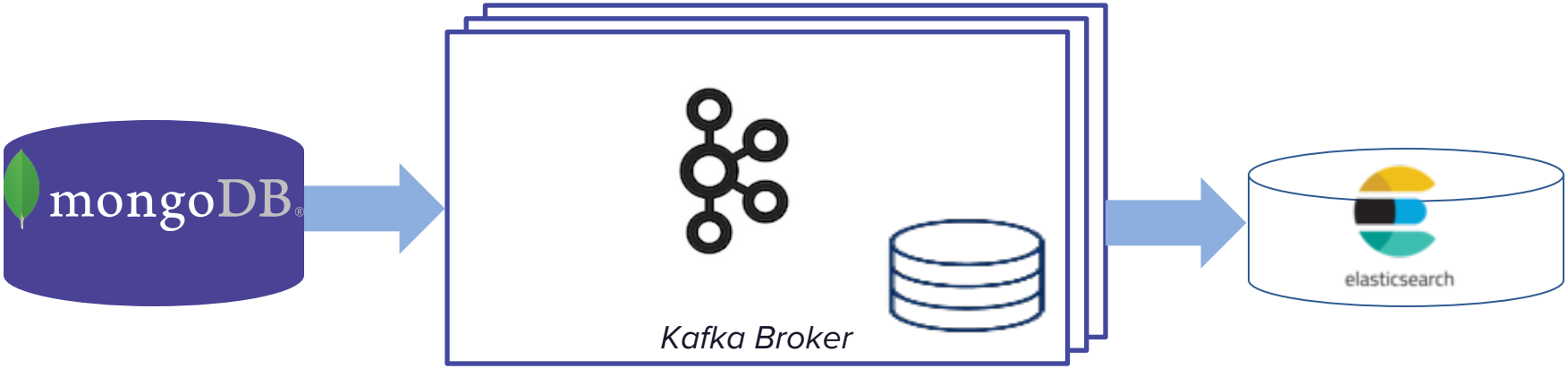
HTTP->Kafka: Kafka-REST vs Reactive





- *Lower latency at high concurrency with reactive*

ETL: MongoDB => Kafka => ElasticSearch



- *ETL pipeline: Extract, Transform, Load*
- *Can be done with Kafka Connect*
  - *Reactive API useful when pipeline includes multiple remote components*

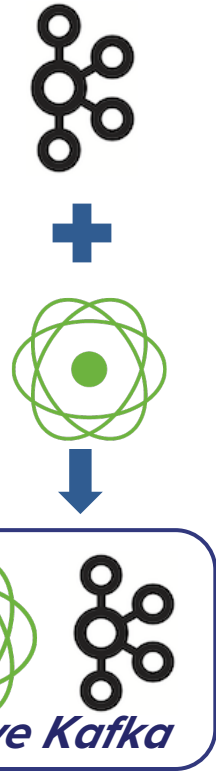
MongoDB => Kafka => ElasticSearch

```
kafkaConnector.createFlux()
    .doOnSubscribe(s ->
          mongoSource.findAll()                          // Extract from MongoDB source
                  .flatMap(person -> kafkaConnector.store(person))
                  .subscribe()
           )
    .map(k-> new PersonRecord(k).transform())  // Transform
    .window(sinkBufferSize, sinkBufferTimeout)
    .flatMap(list -> elasticSearchSink.put(list));     // Load into ElasticSearch
```

Composition

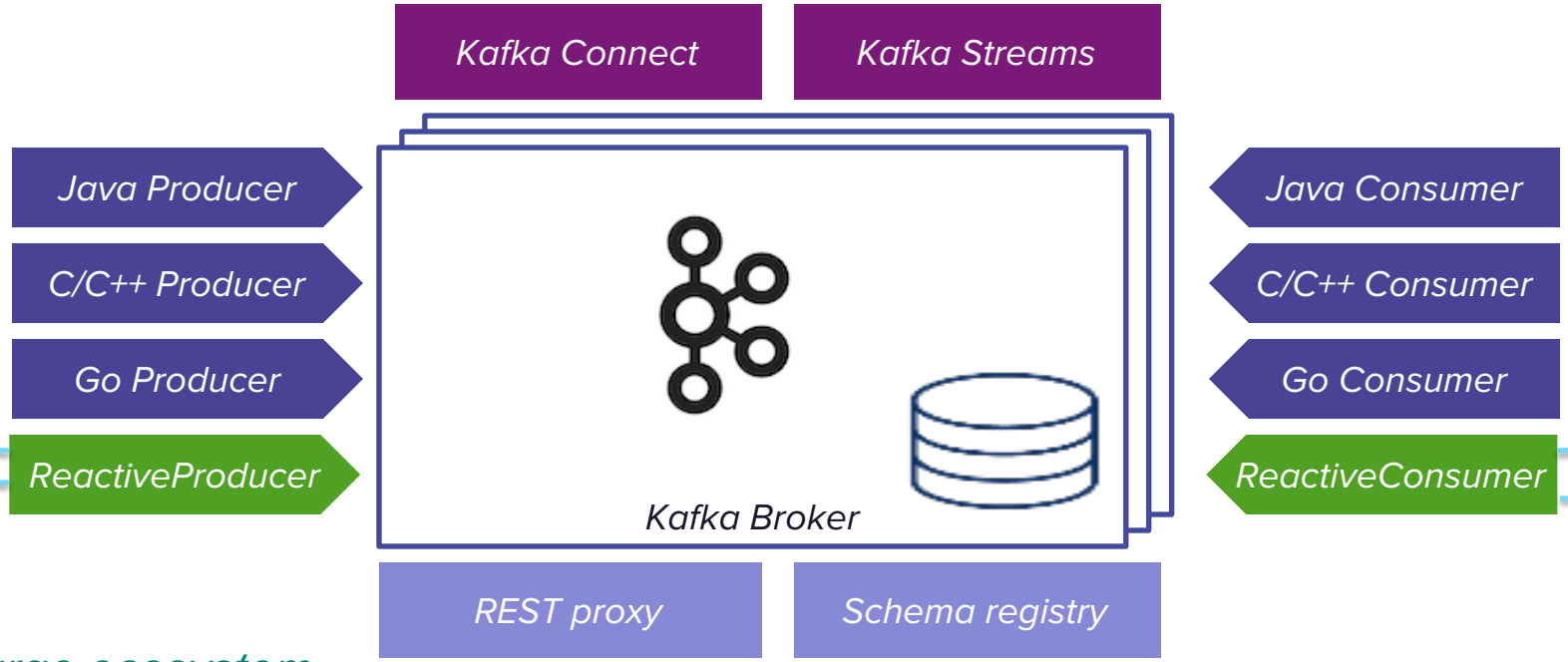
```
mongoSource
      .findAll()                                                    // Extract
      .flatMap(person -> kafkaConnector.store(person))              // Store in Kafka
      .map(person -> personalDataSource.fetch(person))              // Remote operation
      .window(sinkBufferSize, sinkBufferTimeout)
      .flatMap(list -> elasticSearchSink.put(list));                // Load
```

Summary



- *Apache Kafka*
  - *Distributed, highly available, scalable, high throughput, low-latency*
  - *Vast ecosystem of clients and tools*
    - *Works well without the need for back-pressure for Kafka interactions*
- *Reactive Streams*
  - *Functional-style, Back-pressure*
- *Reactive Kafka*
  - *Full functionality of non-reactive clients*
  - *May see performance drop in some scenarios*
  - *Reactive pipeline with Kafka as well as non-Kafka components*
    - *Benefits from non-blocking back-pressure*
    - *Better concurrency, more efficient use of resources, better CPU utilization*

Kafka clients - revisited



- *Large ecosystem of clients, tools, connectors*

Spark APACHE STORM™ Flink FLUME

Want to find out more?

- *Apache Kafka*
  - *http://kafka.apache.org/documentation.html*
  - *https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying*
  - *https://cwiki.apache.org/confluence/display/KAFKA/Clients*
- *Project Reactor*
  - *https://projectreactor.io/docs/*
- *Reactive Kafka*
  - *https://github.com/reactor/reactor-kafka*

SpringOne Platform

Learn More. Stay Connected.

rsivaram@pivotal.io

Related Sessions:

Reactor 3.0, a JVM Foundation for Java 8 and Reactive Streams
Designing, Implementing, and Using Reactive APIs
From Imperative To Reactive Web Apps
Spring for Apache Kafka


@springcentral
spring.io/blog


@pivotal
pivotal.io/blog


@pivotalcf
http://engineering.pivotal.io

SpringOne Platform

Thank you for listening.

Questions?

 *@springcentral*
spring.io/blog

 *@pivotal*
pivotal.io/blog

 *@pivotalcf*
http://engineering.pivotal.io