Producing Messages with Kafka Producers

Ryan Plant
COURSE AUTHOR

@ryan_plant blog.ryanplant.com

Demo

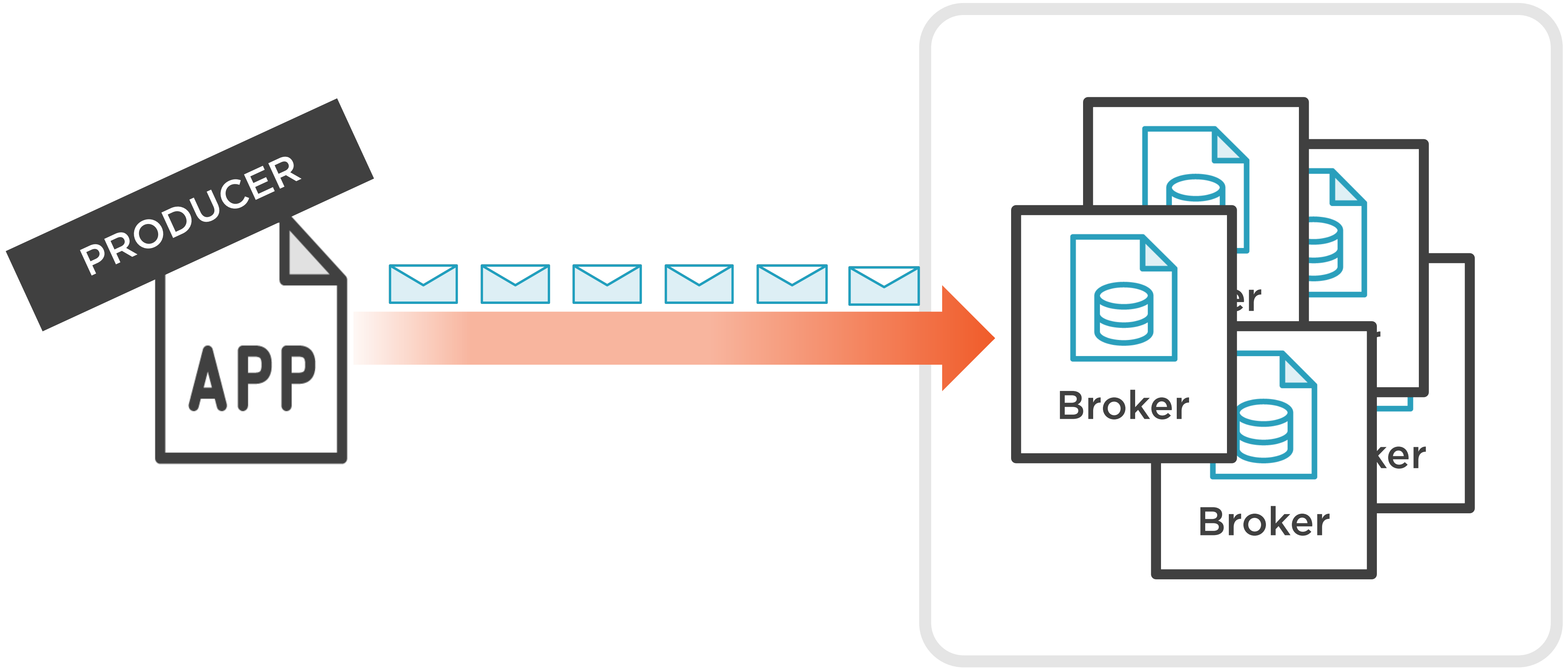
Development Environment Setup

- Adding Kafka Dependencies
- Browsing the API

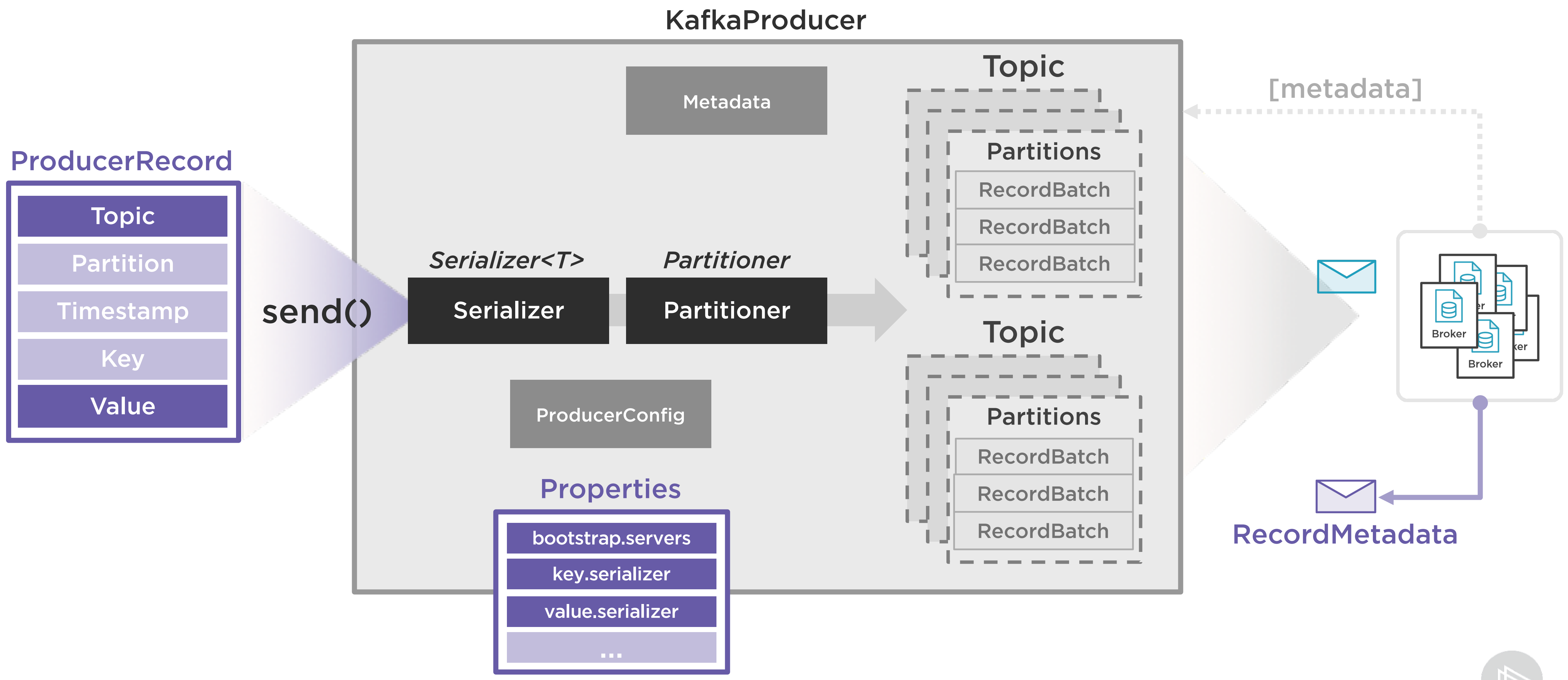
Prerequisites

- Integrated Development Environment
- Java 8 JDK
- Maven
- Access to a test Kafka cluster

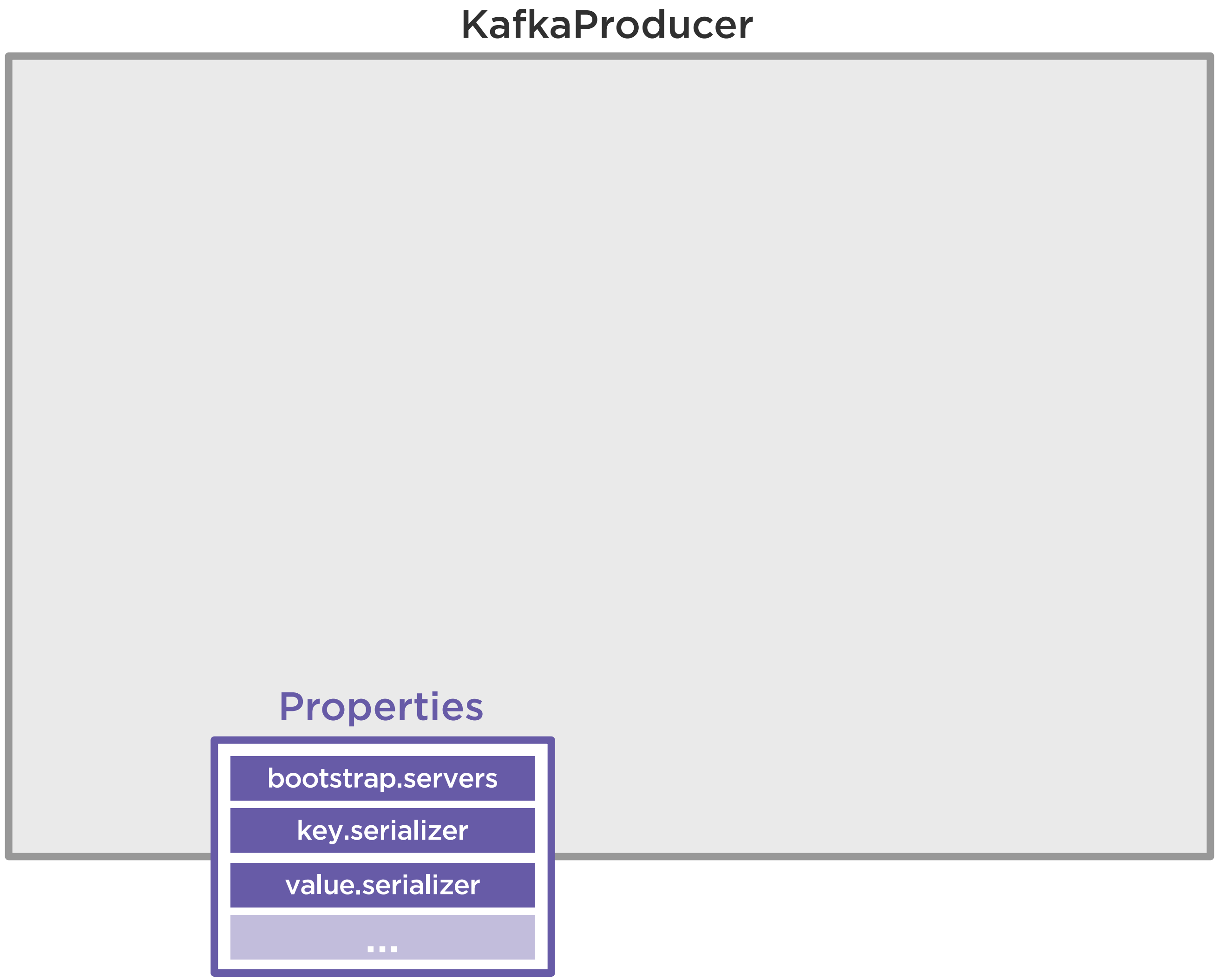
Kafka Producer Externals



Kafka Producer Internals



Creating a Kafka Producer



http://kafka.apache.org/documentation.html#producerconfigs

```
Properties props = new Properties();

props.put(“bootstrap.servers”, “BROKER-1:9092, BROKER-2:9093”);

props.put(“key.serializer”, “org.apache.kafka.common.serialization.StringSerializer”);

props.put(“value.serializer”, “org.apache.kafka.common.serialization.StringSerializer”);
```

Kafka Producer: Required Properties

bootstrap.servers

- Cluster membership: partition leaders, etc.

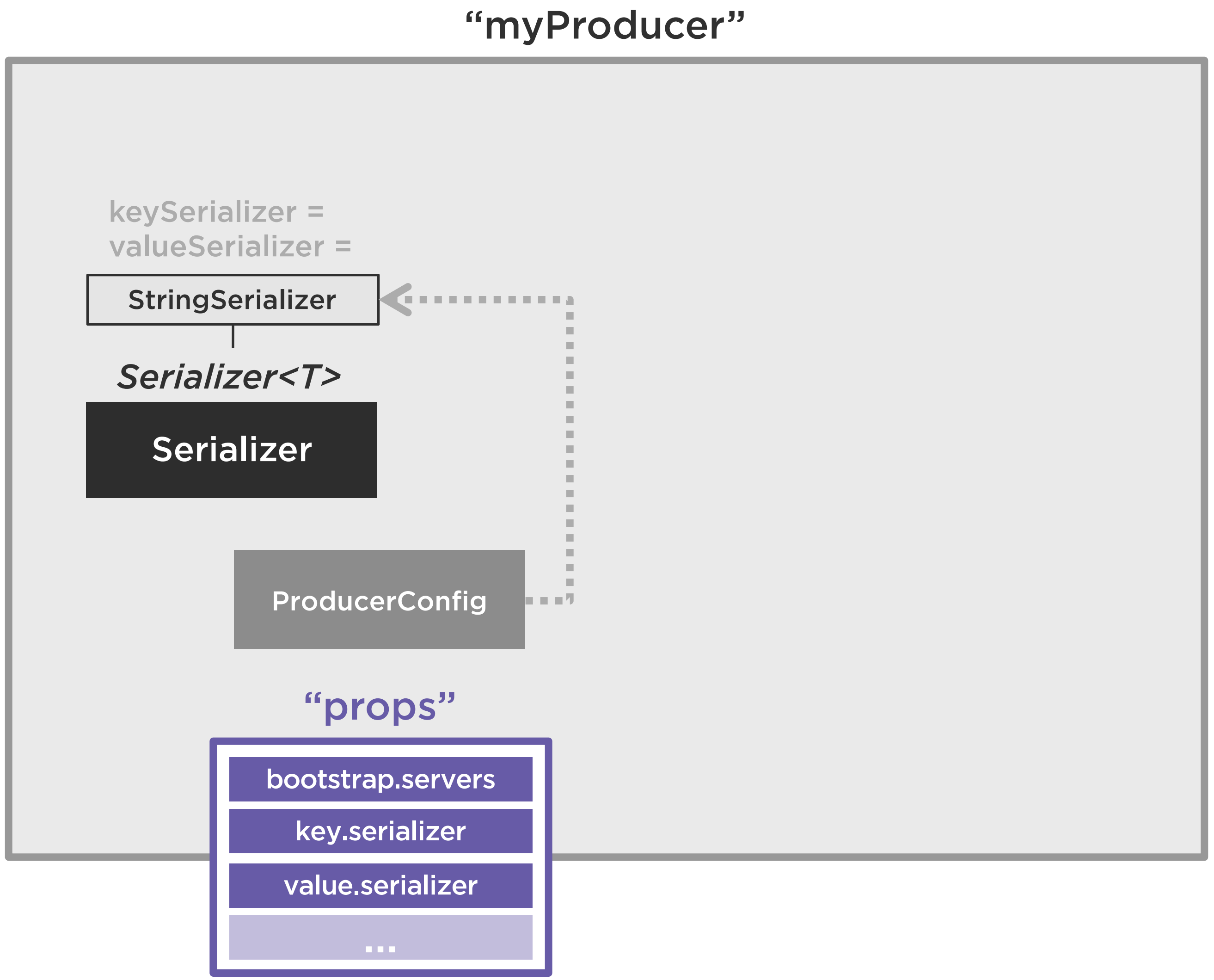
key and value serializers

- Classes used for message serialization and deserialization

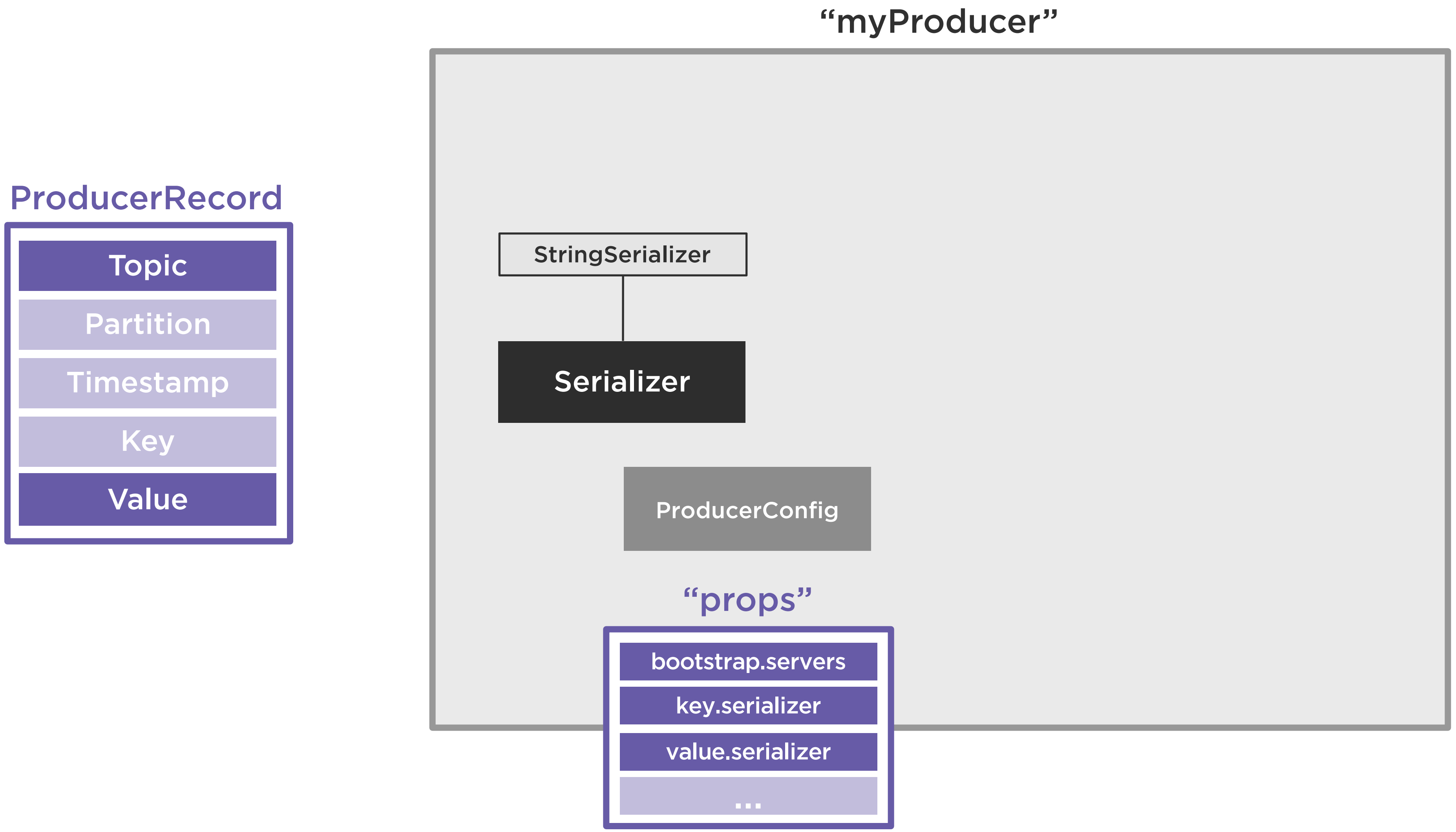
Creating a Kafka Producer

```
public class KafkaProducerApp {

    public static void main(String[] args){

        Properties props = new Properties();

        props.put(“bootstrap.servers”, “BROKER-1:9092, BROKER-2:9093”);

        props.put(“key.serializer”, “org.apache.kafka.common.serialization.StringSerializer”);

        props.put(“value.serializer”, “org.apache.kafka.common.serialization.StringSerializer”);


        KafkaProducer myProducer = new KafkaProducer(props);

    }

}
```

Basic Kafka Producer



Kafka Producer ~~Messages~~ Records



```
props.put(“value.serializer”, “org.apache.kafka.common.serialization.StringSerializer”)

ProducerRecord myMessage = new ProducerRecord(“my_topic”,“My Message 1”);
```

ProducerRecord: Required Properties

topic

- Topic to which the ProducerRecord will be sent

value

- The message content (matching the serializer type for value)

Kafka Producer Shell Program

```
~$ bin/kafka-console-producer.sh \
> --broker-list localhost:9092, localhost:9093 \
> --topic my_topic

MyMessage 1
MyMessage 2
```

```
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer")

ProducerRecord myMessage = new ProducerRecord("my_topic","My Message 1");


ProducerRecord myMessage = new ProducerRecord("my_topic",3.14159);

SerializationException: Can't convert value of class ...
```

ProducerRecord: Required Properties

topic

- Topic to which the ProducerRecord will be sent

value

- The message content (matching the serializer type for value)

KafkaProducer instances can only send ProducerRecords that match the key and value serializers types it is configured with.

```
ProducerRecord(String topic, Integer partition, Long timestamp, K key, V value);

// Example:

ProducerRecord(“my_topic”, 1, 124535353325, “Course-001”,“My Message 1”);

// Defined in server.properties:

log.message.timestamp.type = [CreateTime, LogAppendTime]

// CreateTime: producer-set timestamp used.

// LogAppendTime: broker-set timestamp used when message is appended to commit log.
```

ProducerRecord: Optional Properties

partition

- specific partition within the topic to send ProducerRecord

timestamp

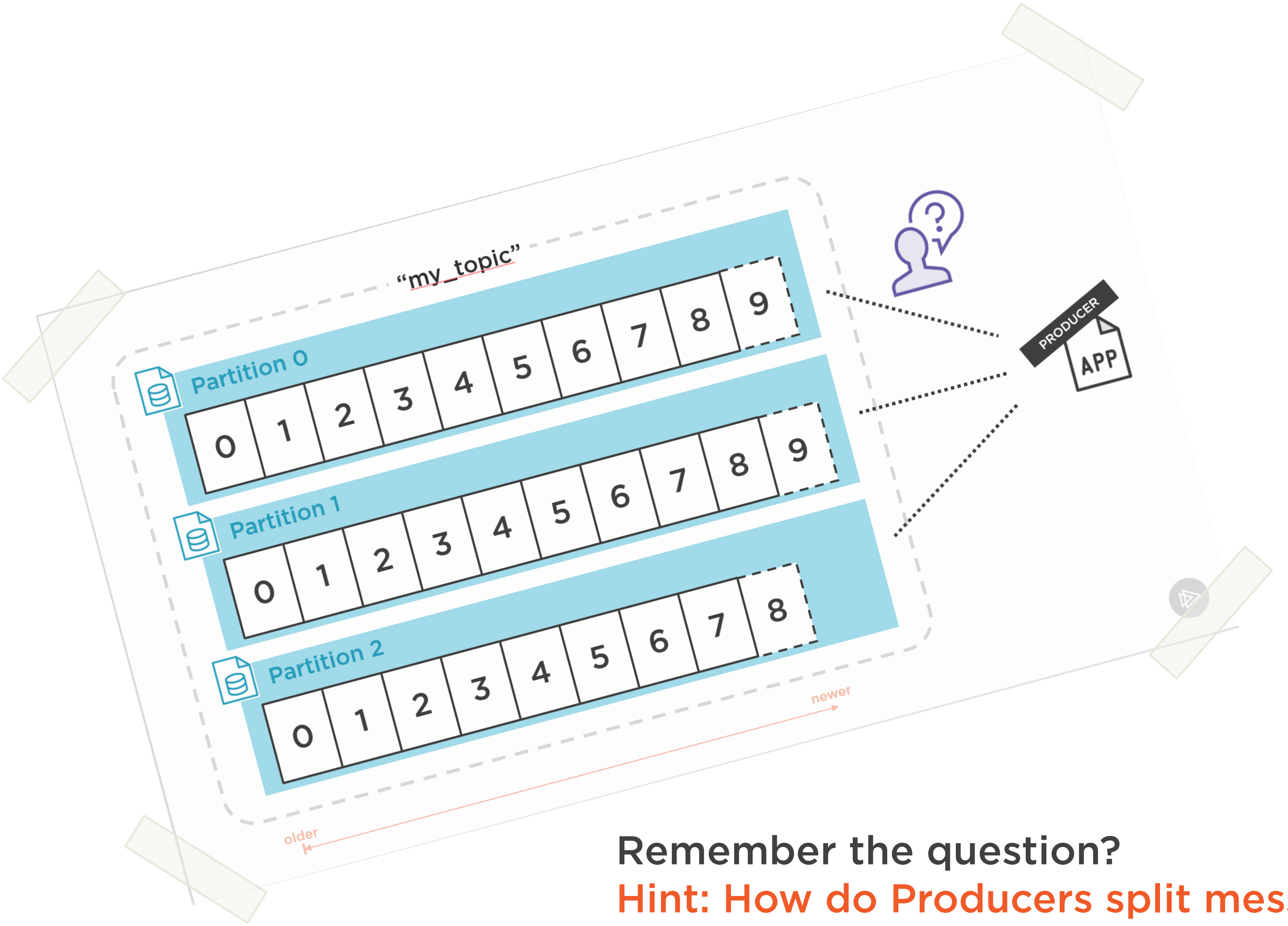
- the Unix timestamp applied to the record

```
ProducerRecord(String topic, Integer partition, Long timestamp, K key, V value);

// Example:

ProducerRecord(“my_topic”, 1, 124535353325, “Course-001”,“My Message 1”);
```

ProducerRecord: Optional Properties

key

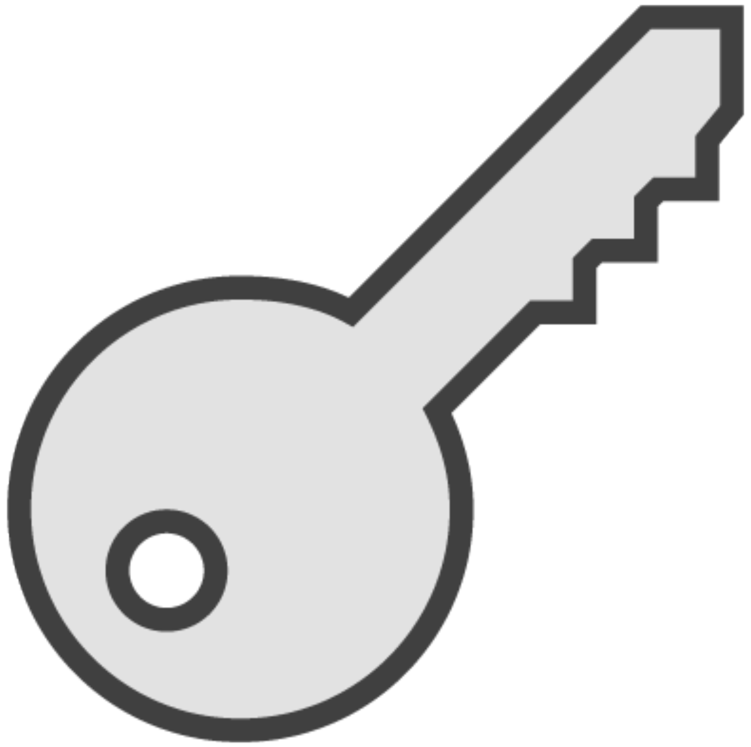
- a value to be used as the basis of determining the partitioning strategy to be employed by the Kafka Producer



Remember the question?

Hint: How do Producers split messages to partitions?

Best Practice: Define a Key



Two useful purposes:

- Additional information in the message
- Can determine what partitions the message will be written to

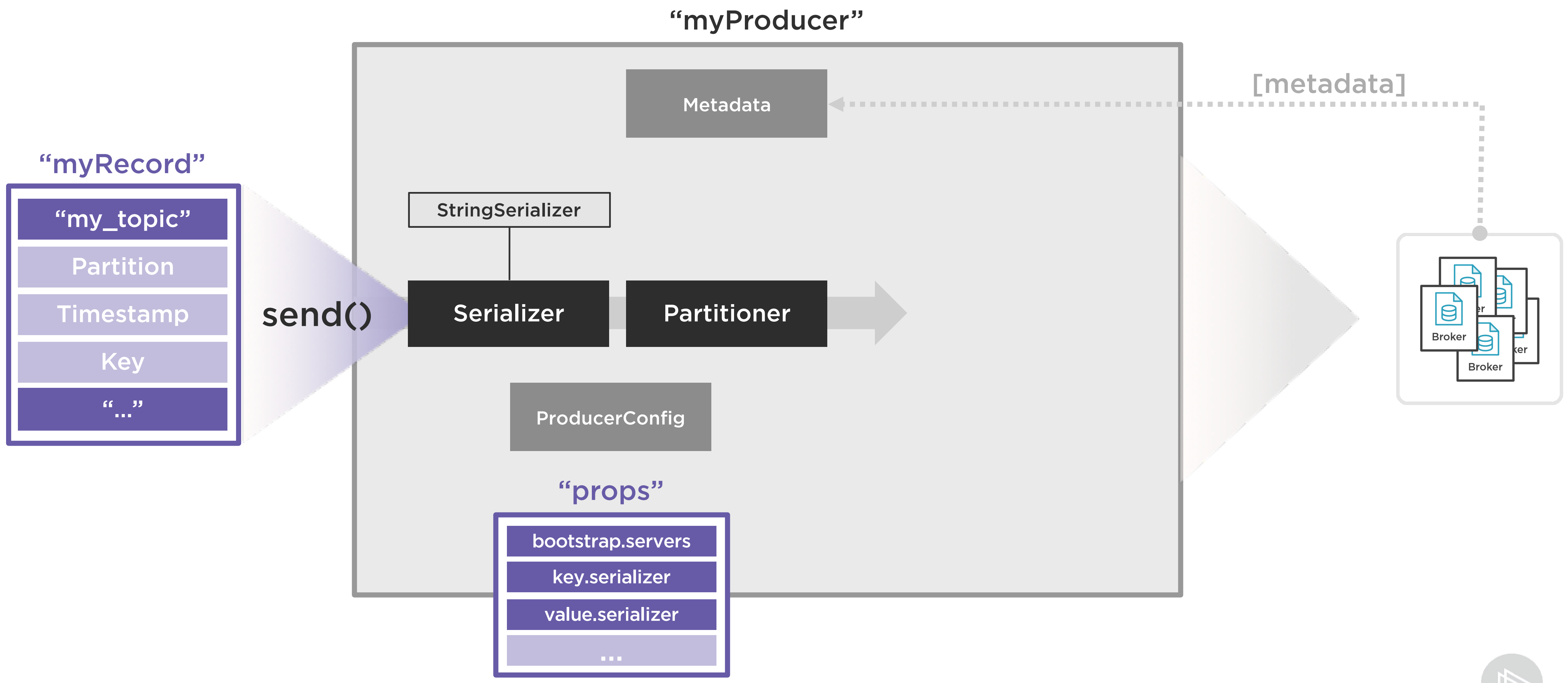
Downside:

- Additional overhead
- Depends on the serializer type used

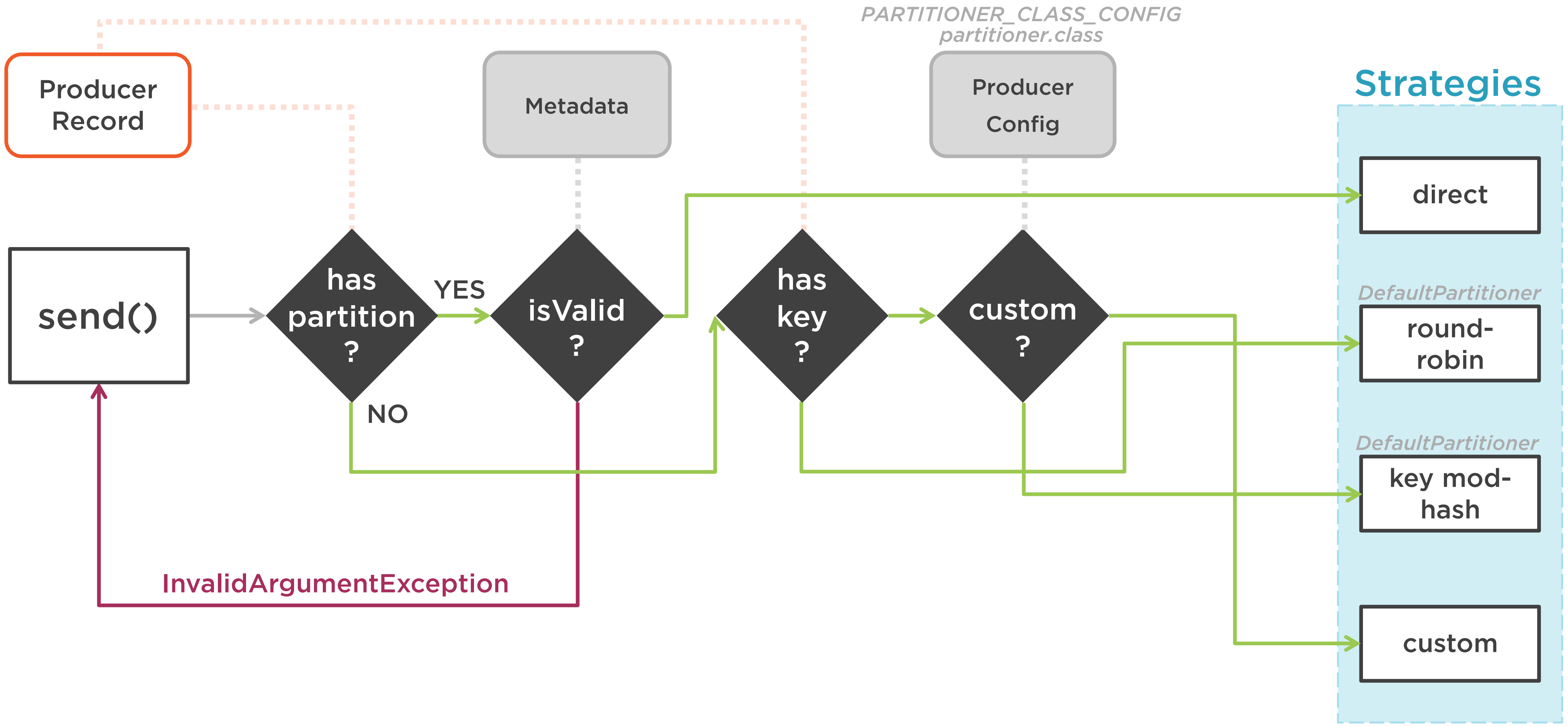
Making Messaging Magic

```
public class KafkaProducerApp {

    public static void main(String[] args){

        Properties props = new Properties();

        props.put(“bootstrap.servers”, “BROKER-1:9092, BROKER-2:9093”);

        props.put(“key.serializer”, “org.apache.kafka.common.serialization.StringSerializer”);

        props.put(“value.serializer”, “org.apache.kafka.common.serialization.StringSerializer”);


        KafkaProducer myProducer = new KafkaProducer(props);

        ProducerRecord myRecord = new ProducerRecord(“my_topic”, “Course-001”, “My Message 1”);

        myProducer.send(myRecord); // Best practice: try..catch

     }

}
```

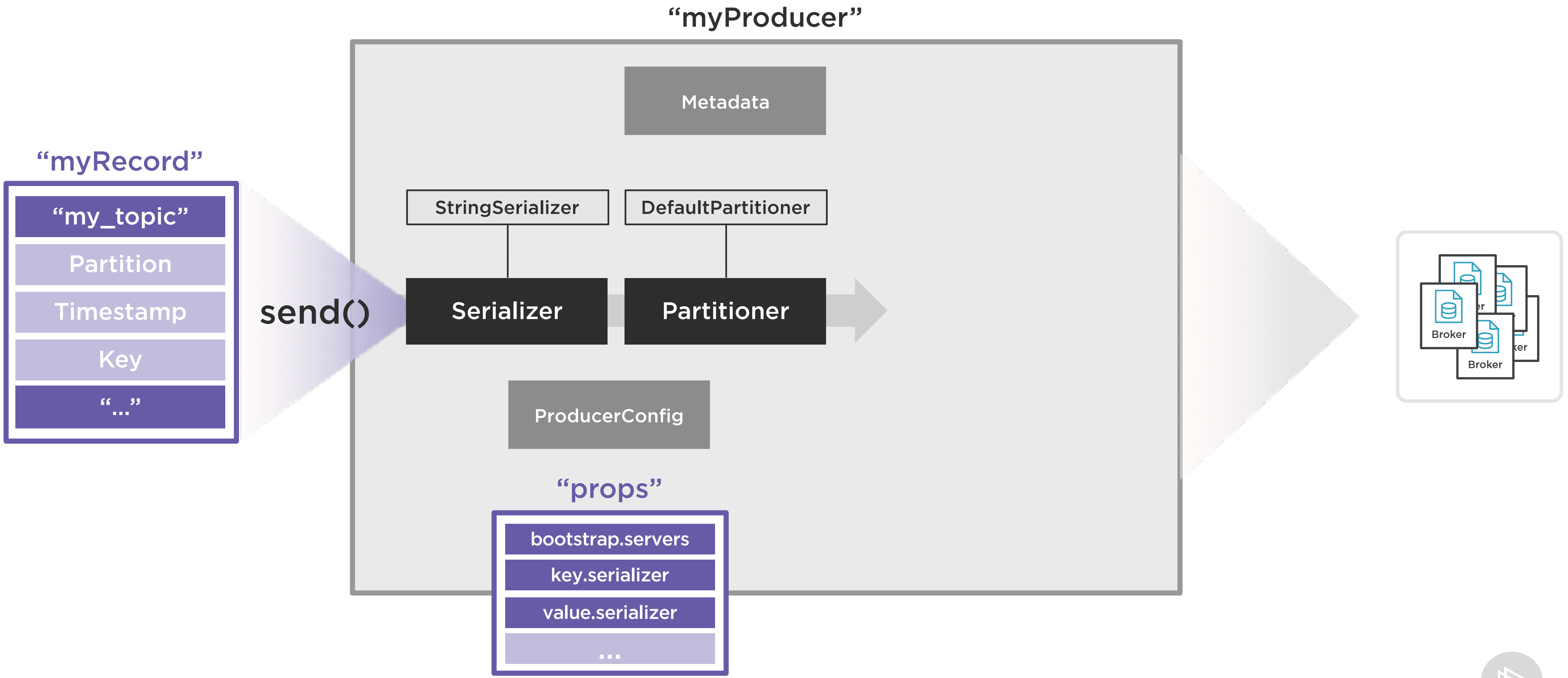
Sending the Message, Part 1



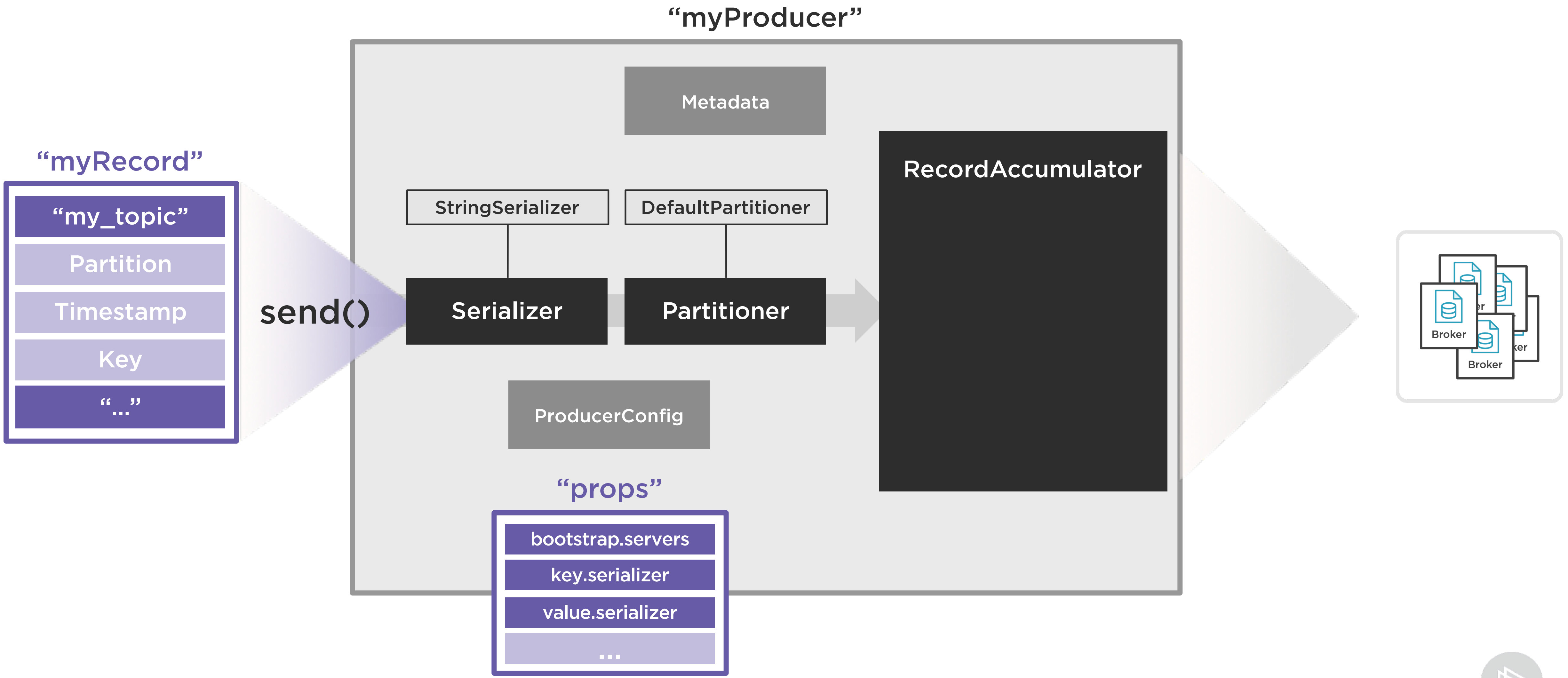
Kafka Producer Partitioning Strategy



Sending the Message, Part 1



Sending the Message, Part 2



Micro-batching in Apache Kafka

At scale, efficiency is everything.

Small, fast batches of messages:

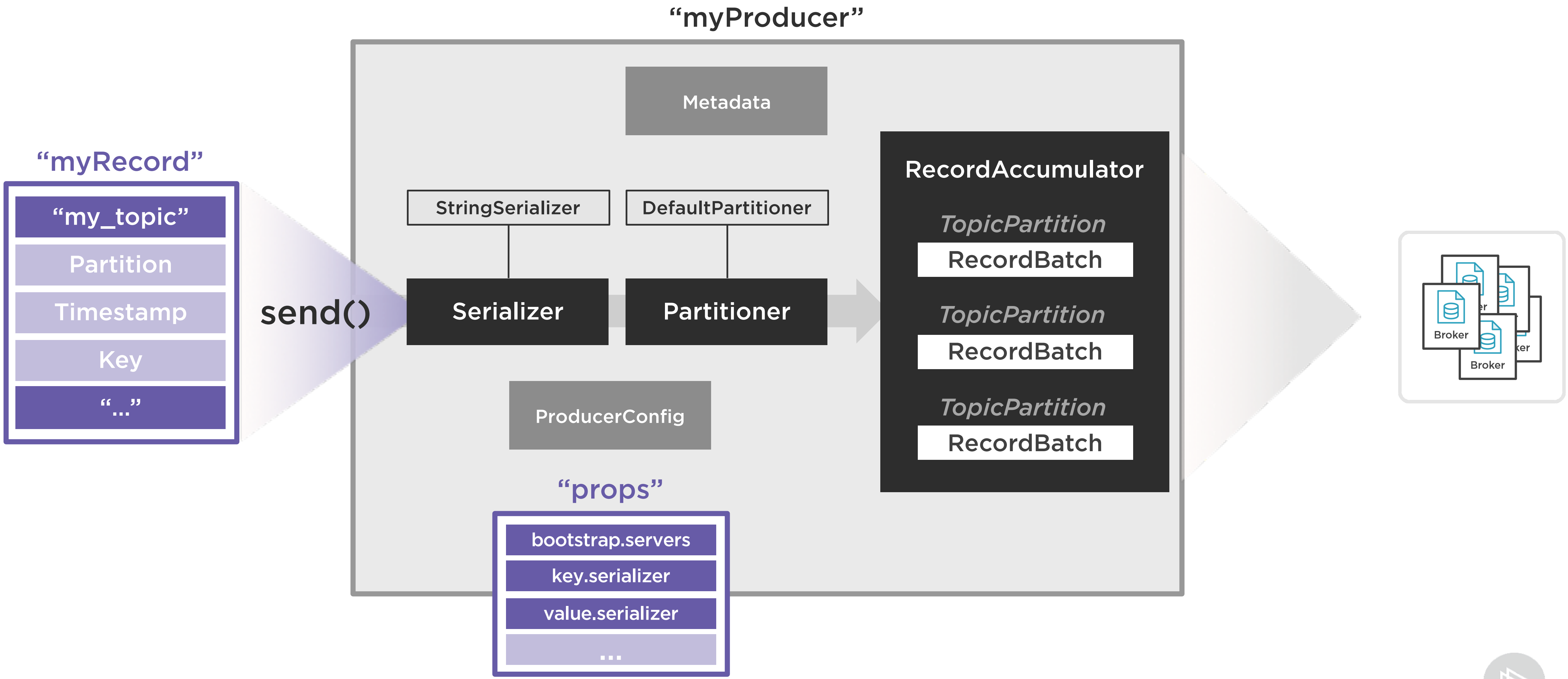
- Sending (Producer)
- Writing (Broker)
- Reading (Consumer)

Modern operating system functions:

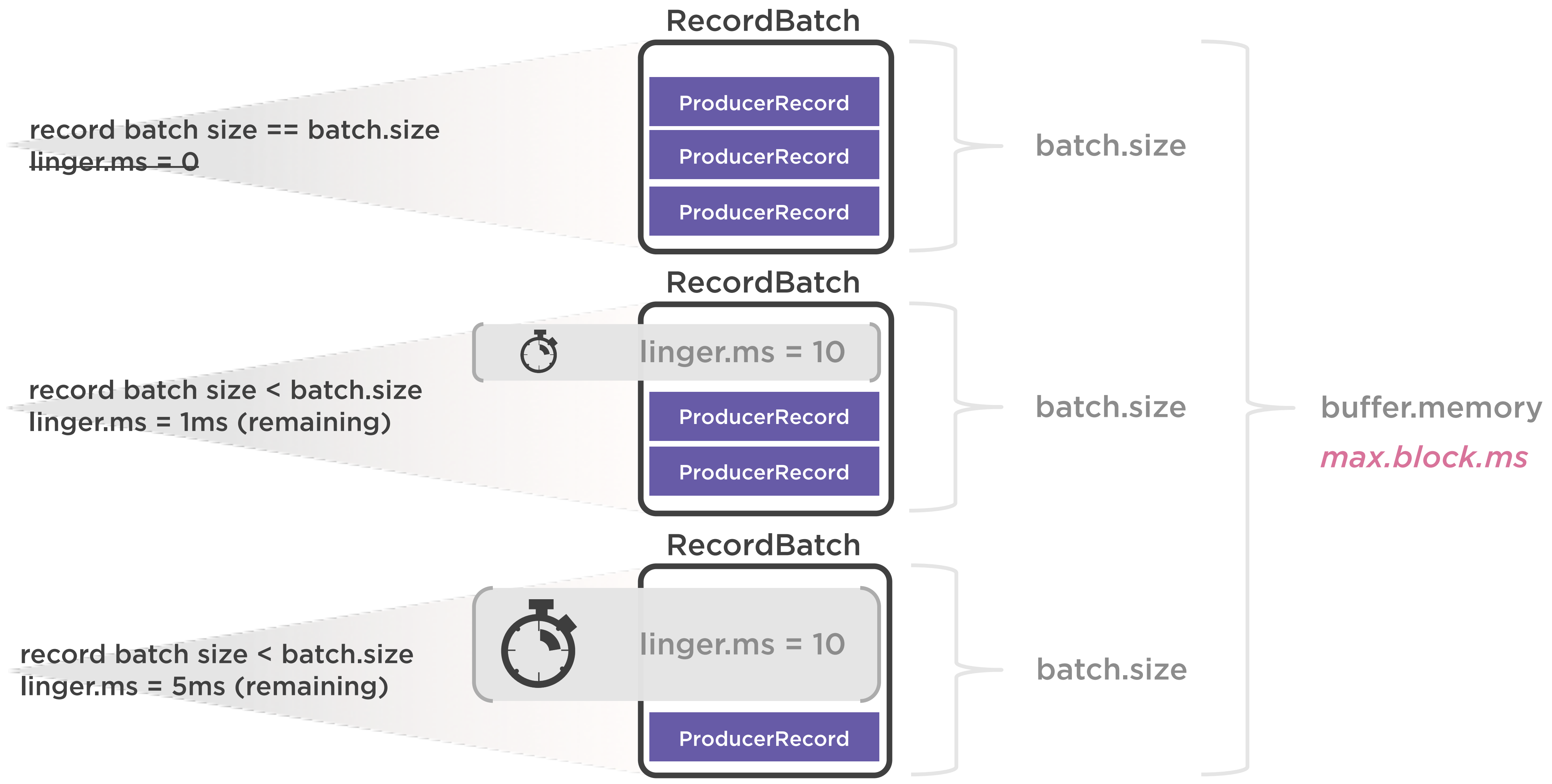
- Pagecache
- Linux sendfile() system call (kernel)

Amortization of the constant cost

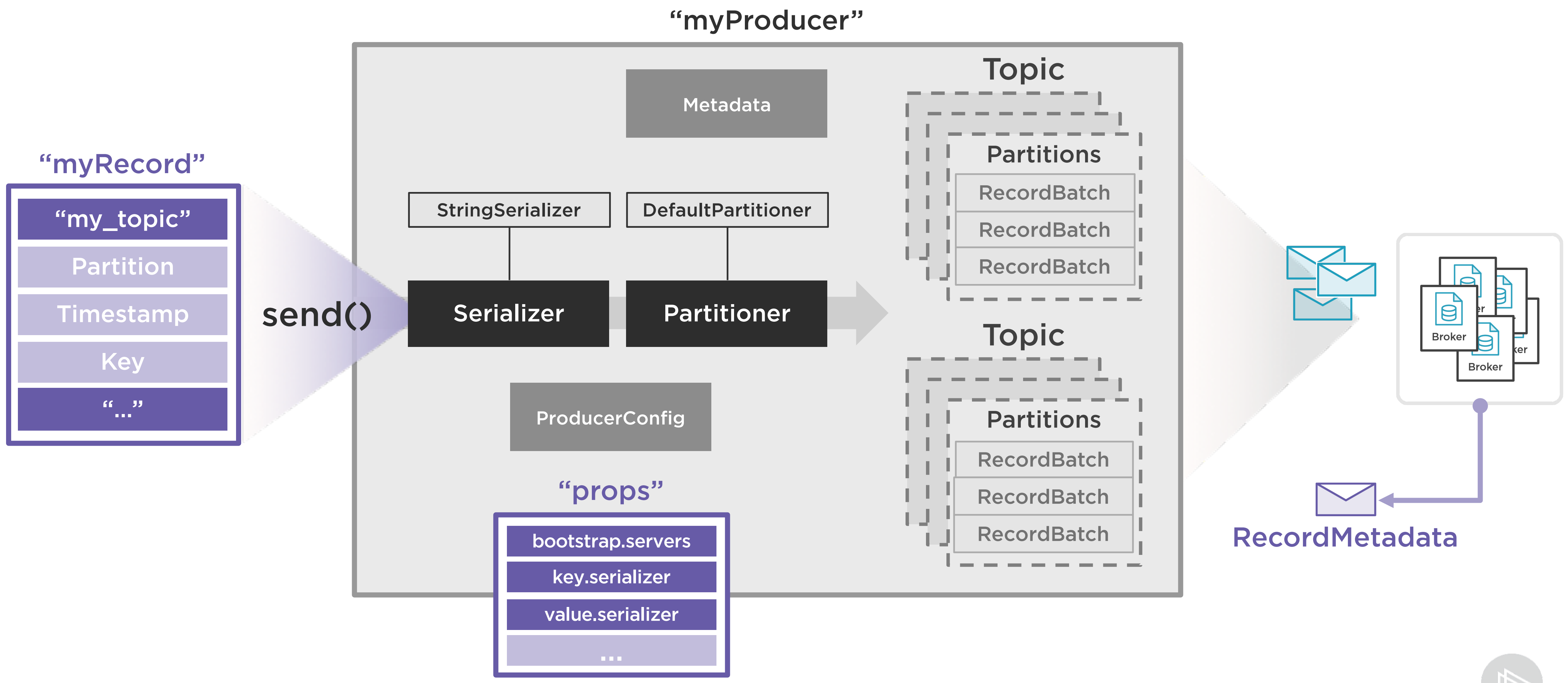
Sending the Message, Part 2



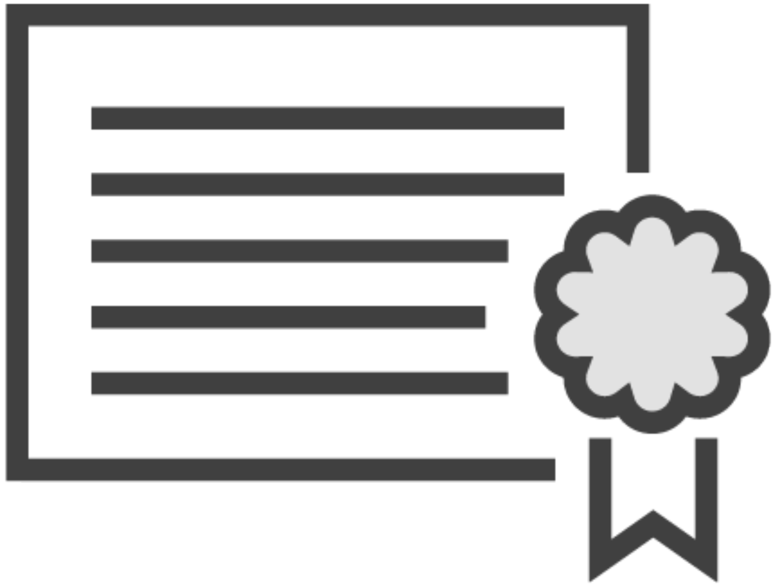
Message Buffering



Sending the Message, Part 2



Delivery Guarantees



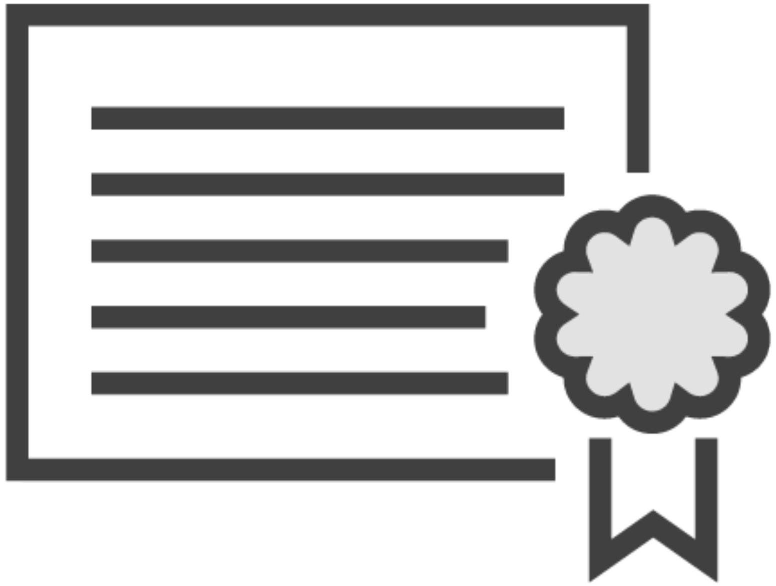
Broker acknowledgement (“acks”)

- 0: fire and forget
- 1: leader acknowledged
- 2: replication quorum acknowledged

Broker responds with error

- “retries”
- “retry.backoff.ms”

Ordering Guarantees



Message order by partition

- No global order across partitions

Can get complicated with errors

- retries, retry.backoff.ms
- max.in.flight.request.per.connection

Delivery semantics

- At-most-once, at-least-once, only-once

Demo

Custom Producer application in Java

- Summary of what we’ve covered
- Similar to shell program

Basic Producer configuration

Cluster setup:

- Three partitions
- Three brokers
- Replication factor: 3

Look for:

- Default partitioner
- No global order (across partitions)

Advanced Topics Not Covered

Custom Serializers

Custom Partitioners

Asynchronous Send

Compression

Advanced Settings

Summary

Kafka Producer Internals

- Properties -> ProducerConfig
- Message -> ProducerRecord
- Processing Pipeline: Serializers and Partitioners
- Micro-batching -> Record Accumulator and RecordBuffer

Delivery and Ordering Guarantees

Java-based Producer