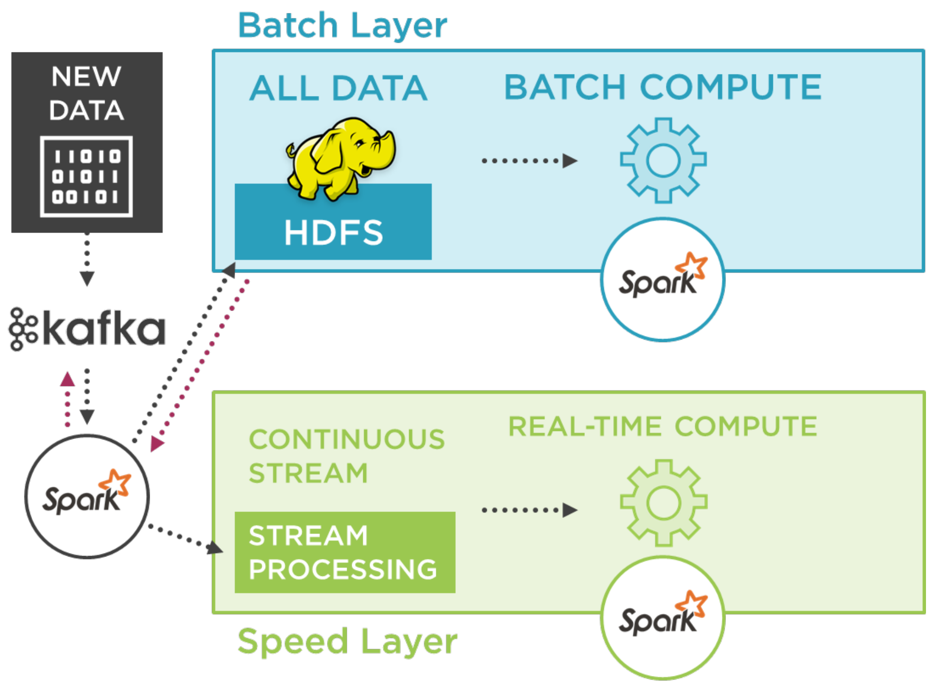
Streaming Ingest with Kafka and Spark Streaming

Ahmad Alkilani
DATA ARCHITECT

@akizl

Streaming Ingest with Kafka and Spark Streaming

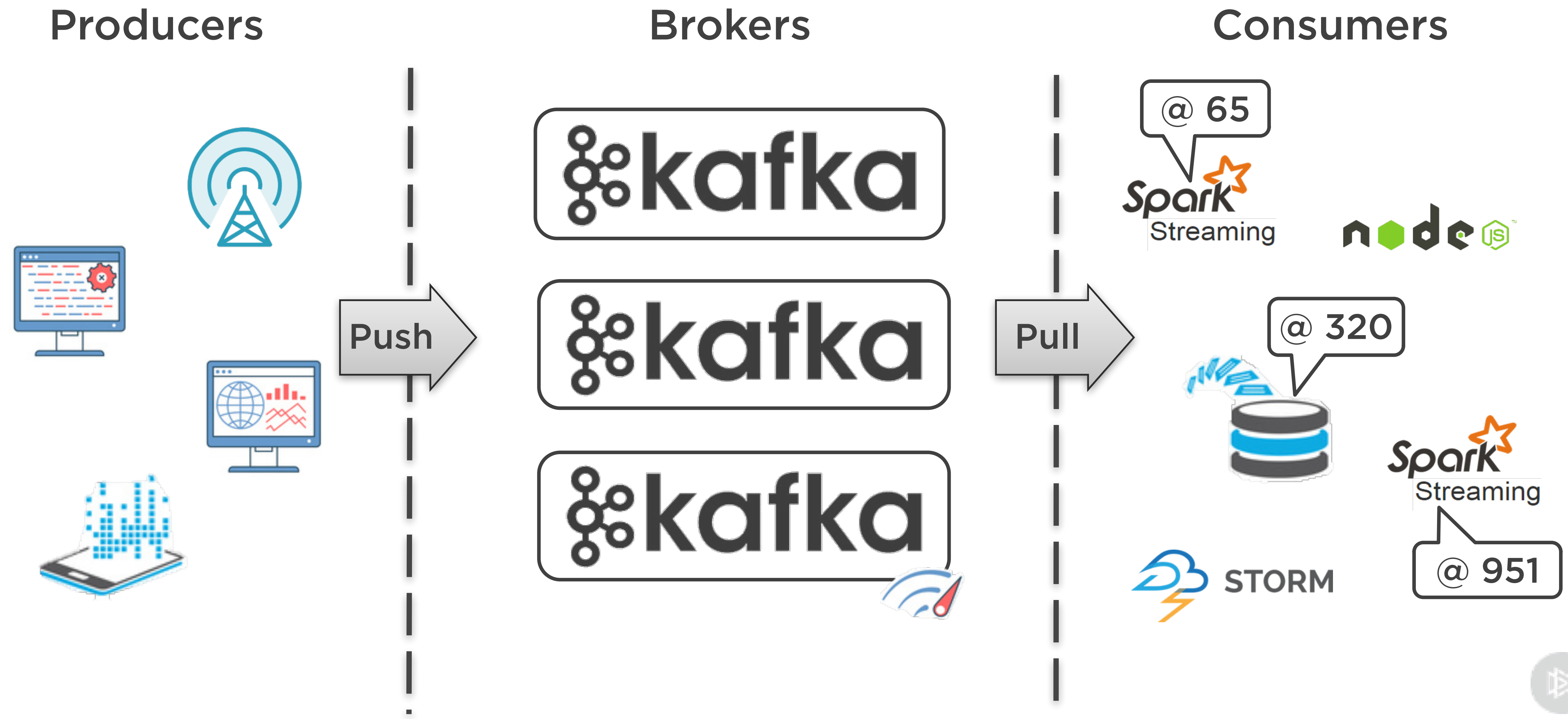


- **Introduction to Kafka**
  - Architecture
  - Producers and Consumers
- **Create a Kafka Producer**
- **Spark Streaming Integration with Kafka**
- **Integrate Batch and Streaming**

kafka

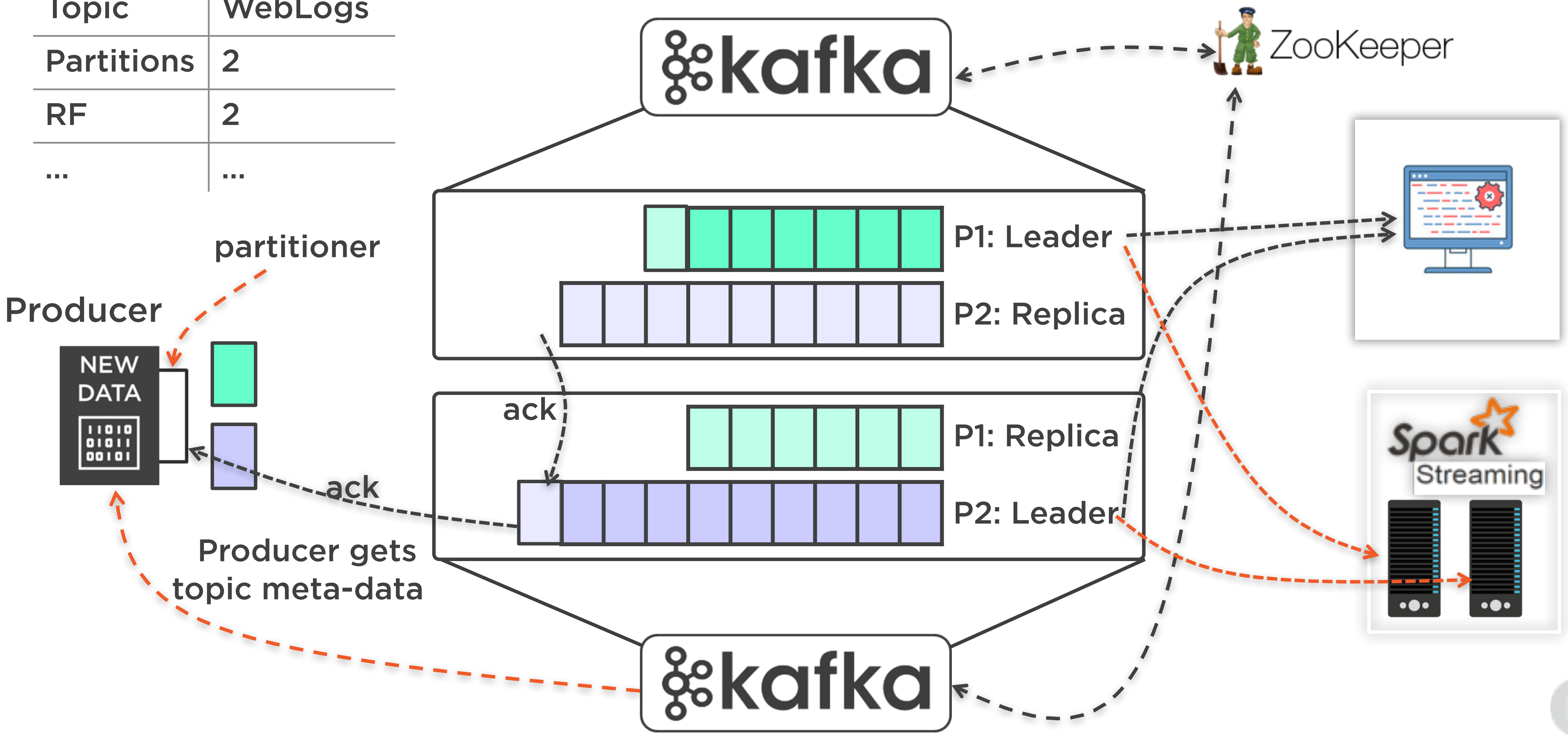
Distributed publish-subscribe messaging system

Introduction to Kafka



The Kafka Broker

Topic	WebLogs
Partitions	2
RF	2
...	...



Partition Assignment & Consumers

Kafka Consumers

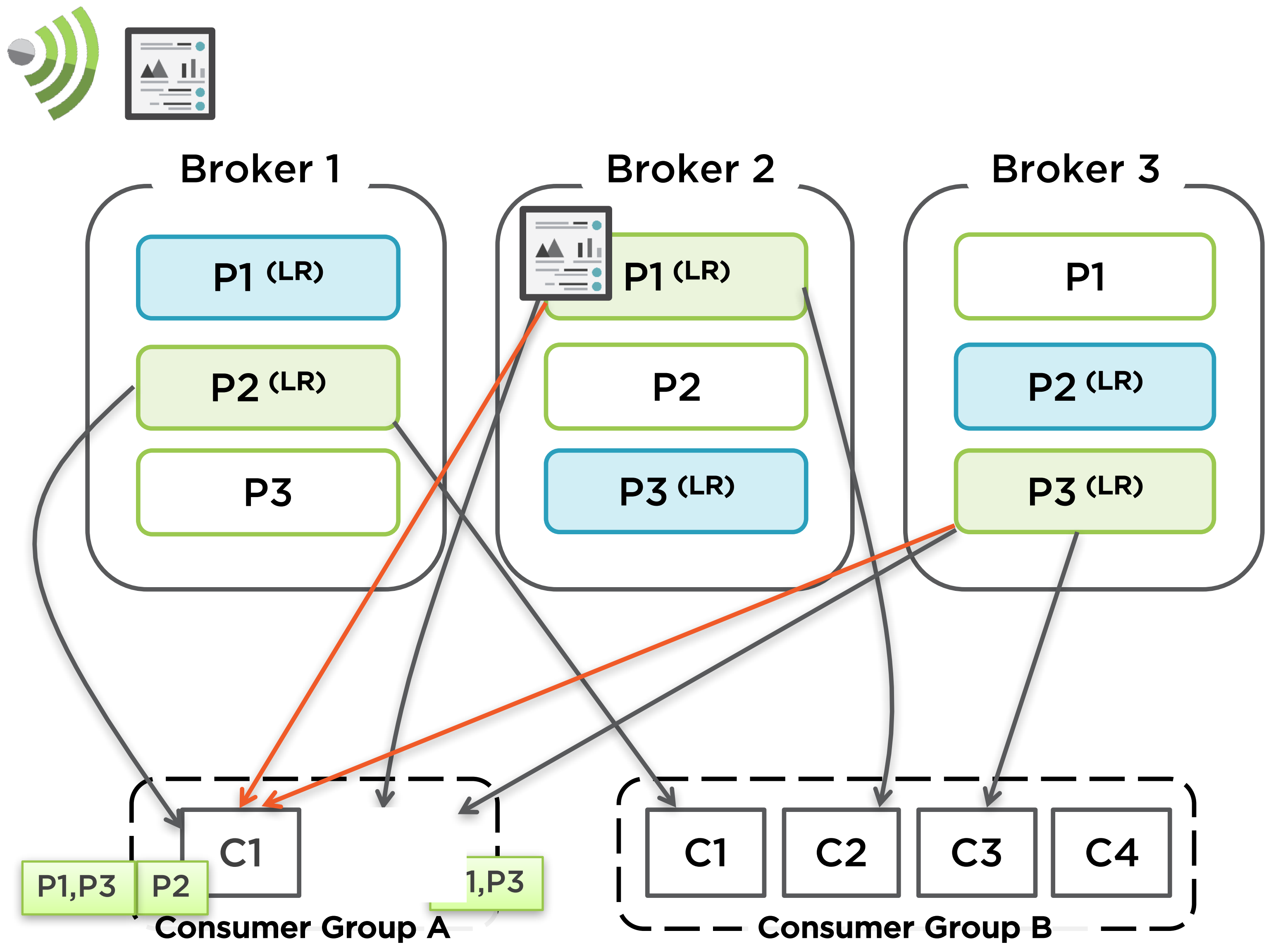
Topics

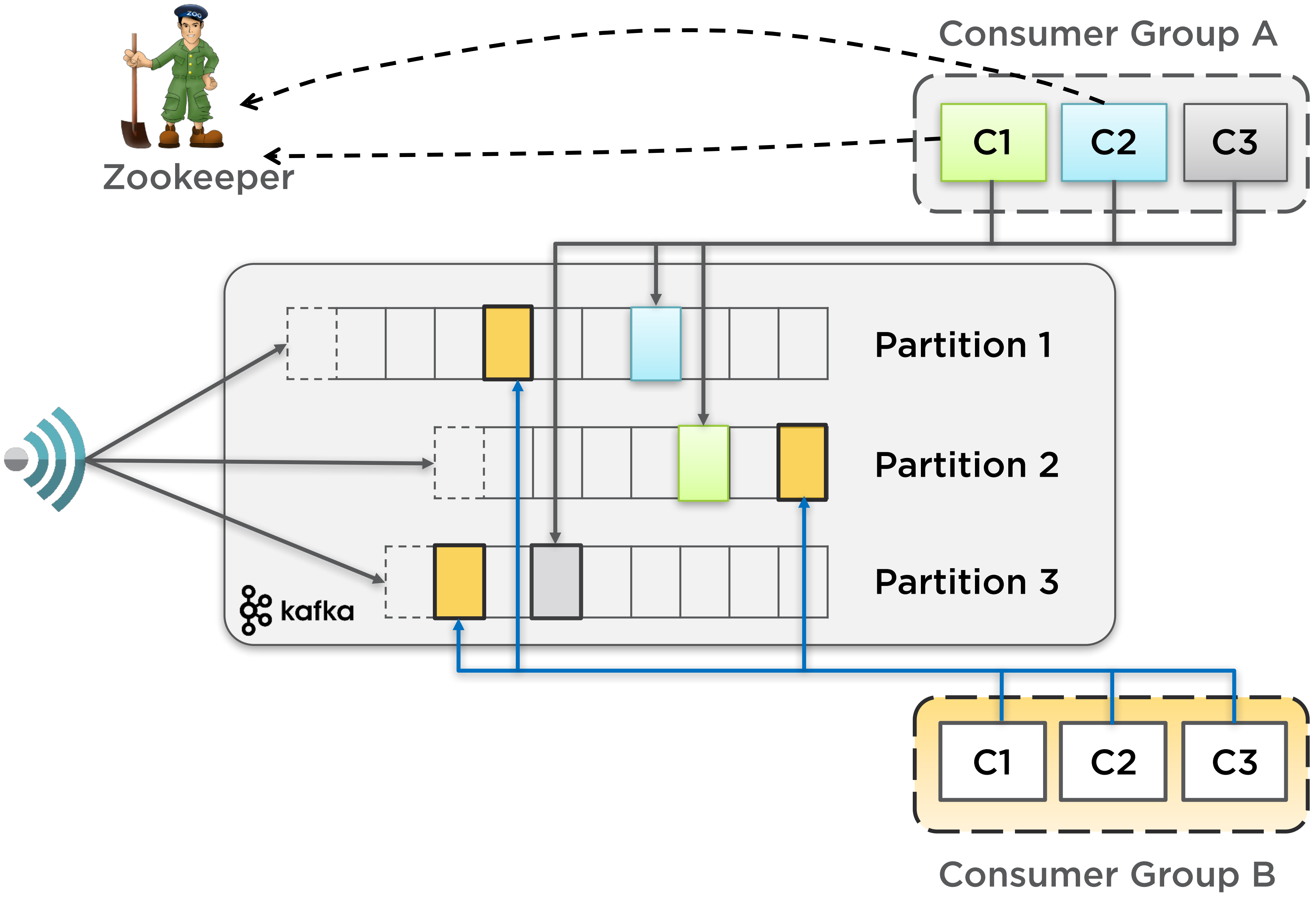
🟦 weblogs

Partitions	3
RF	1

🟩 telemetry

Partitions	3
RF	2

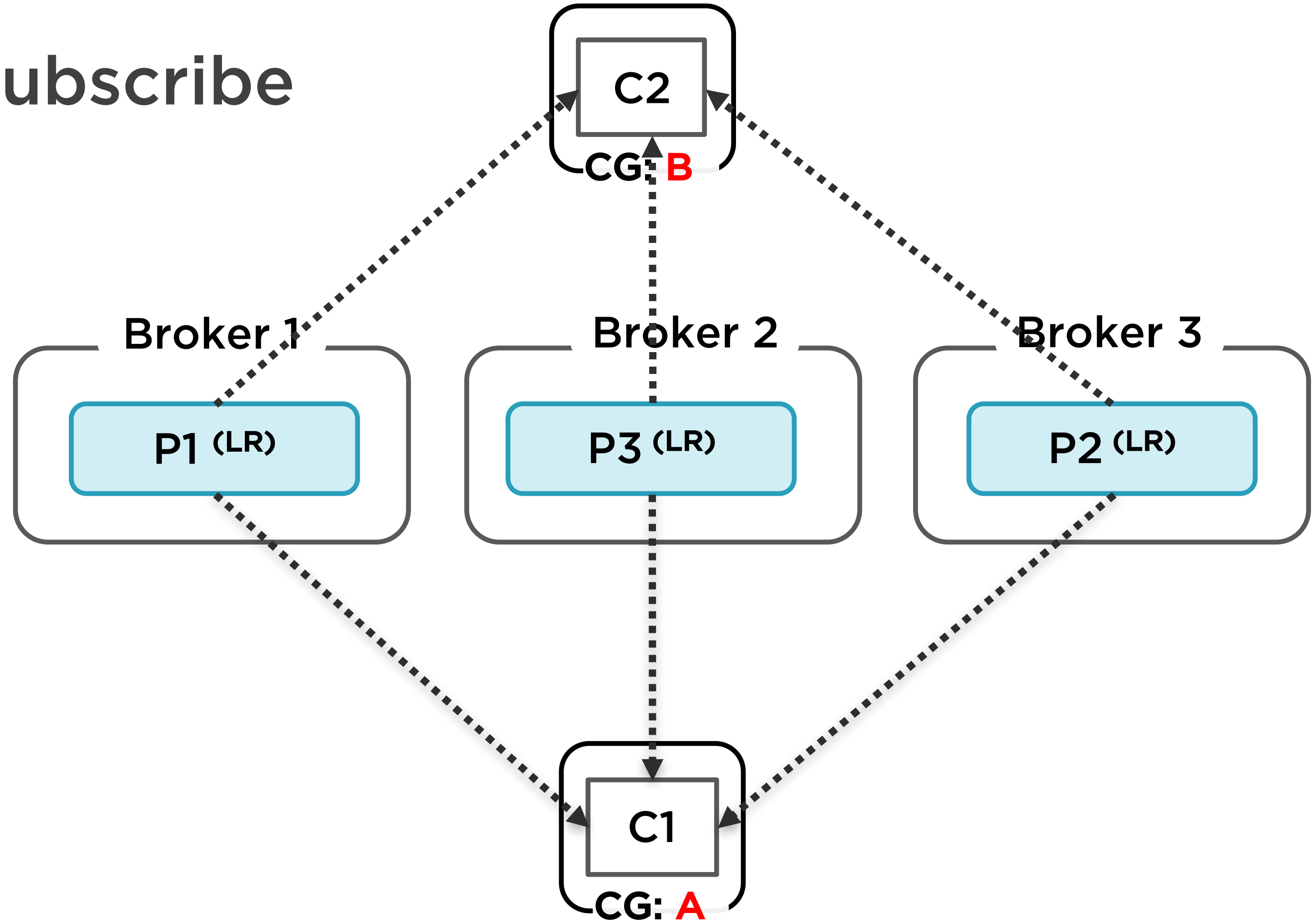



Consumer Group A
C1
C2
C3
Zookeeper
Partition 1
Partition 2
Partition 3
kafka
C1
C2
C3
Consumer Group B

Messaging Models

Messaging Models

Publish-Subscribe



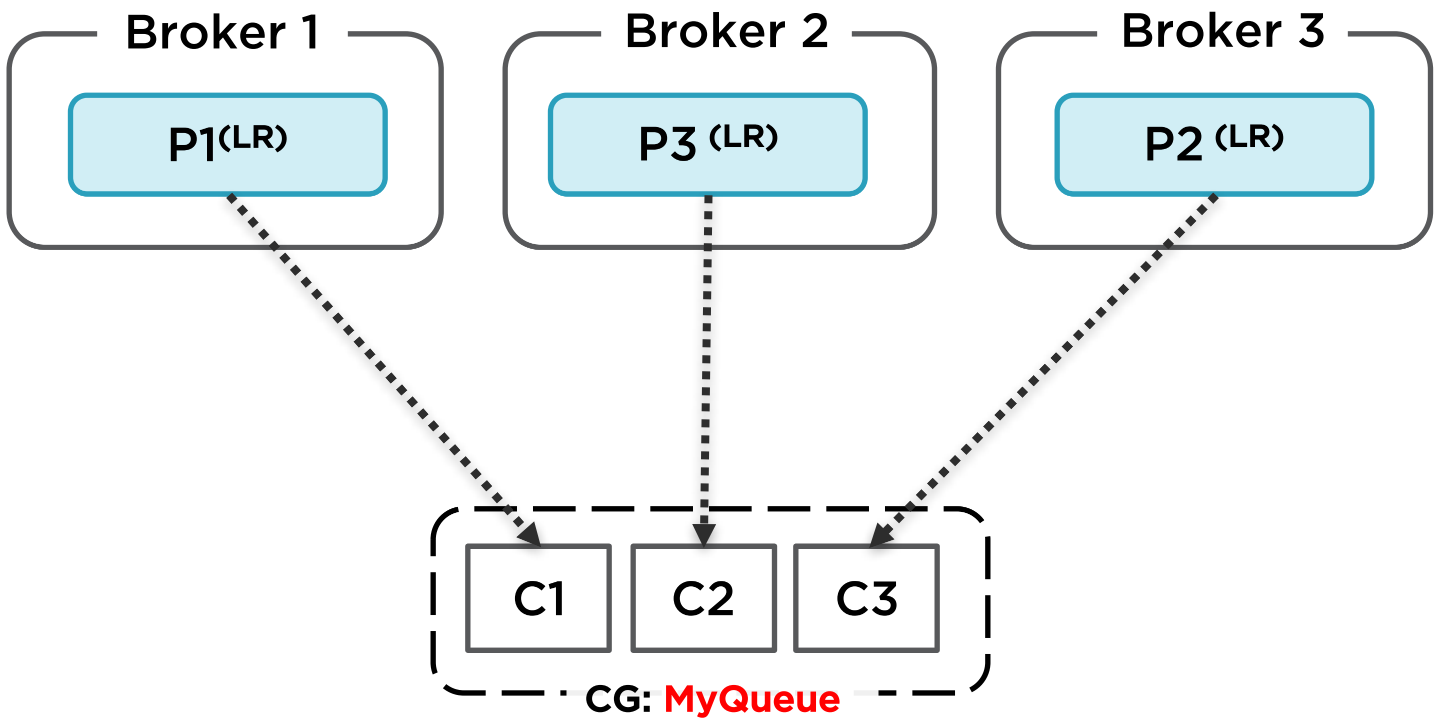
Topics

🟦 weblogs

Partitions	3
RF	1

Messaging Models

Queue Semantics

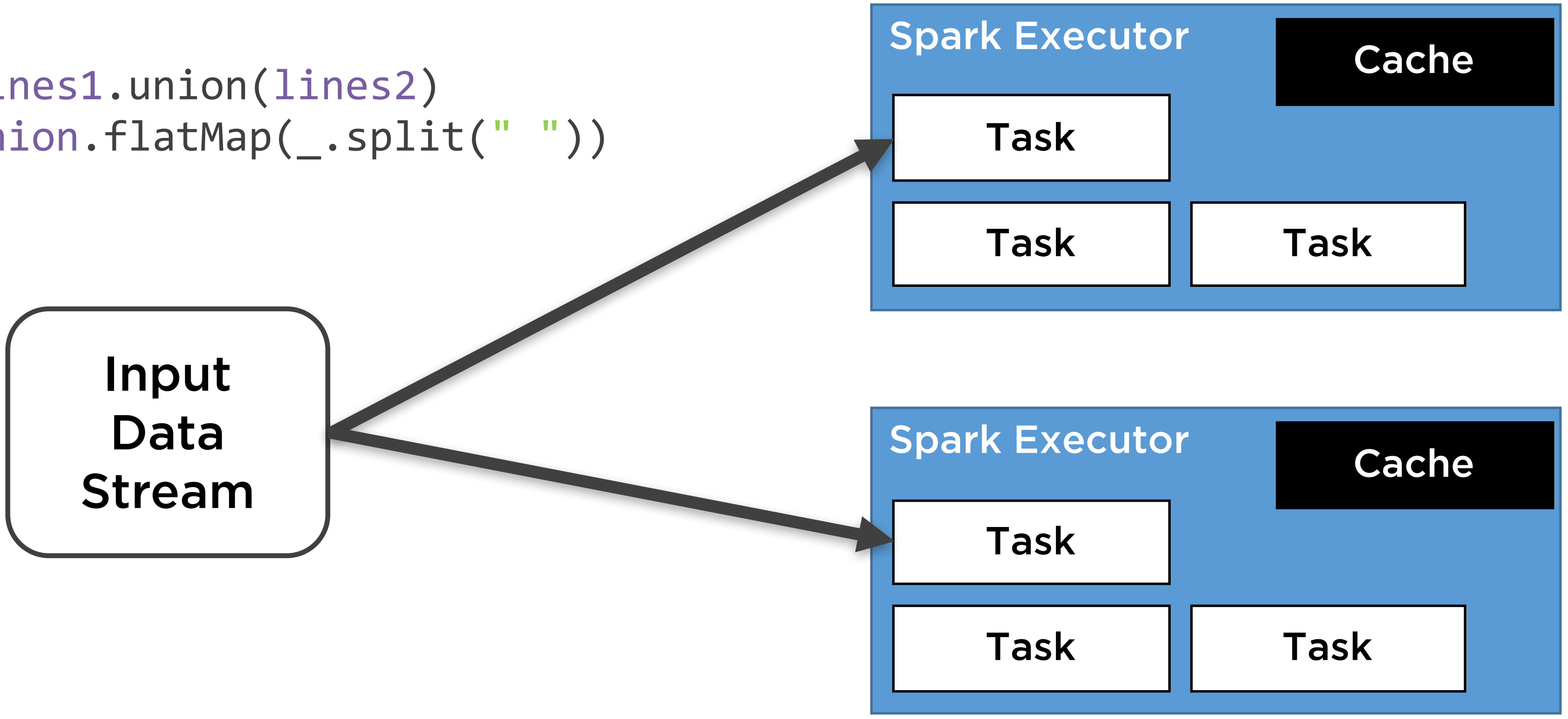


Topics

🟦	weblogs
Partitions	3
RF	1

Receiver Model

```
val lines1 = ssc.socketTextStream("localhost", 9999)
val lines2 = ssc.socketTextStream("localhost", 9998)

val linesUnion = lines1.union(lines2)
val words = linesUnion.flatMap(_.split(" "))
```



Spark Kafka Integration

Spark Streaming Kafka Integration

Kafka Consumer APIs

High-Level API

Receiver Approach

- Receivers to receive data
- Data stored in Spark executors
- Zero-data loss requires write-ahead log
- Allows for at-least-once semantics

Simple API

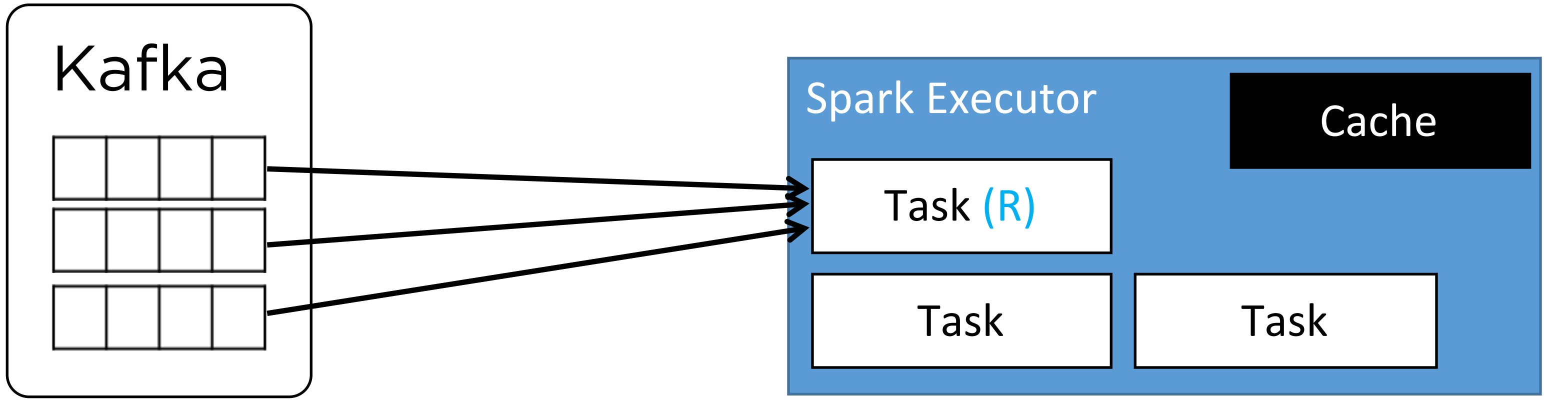
Direct Approach

- No receivers. Queries Kafka each batch for offset range
- Simplifies parallelism at the expense of latency
- Zero-data loss without write-ahead log; relies on Kafka's retention to replay messages. Better at processing larger datasets
- Allows for exactly-once semantics

Receiver-based Approach

Option 1: Create a single Kafka stream

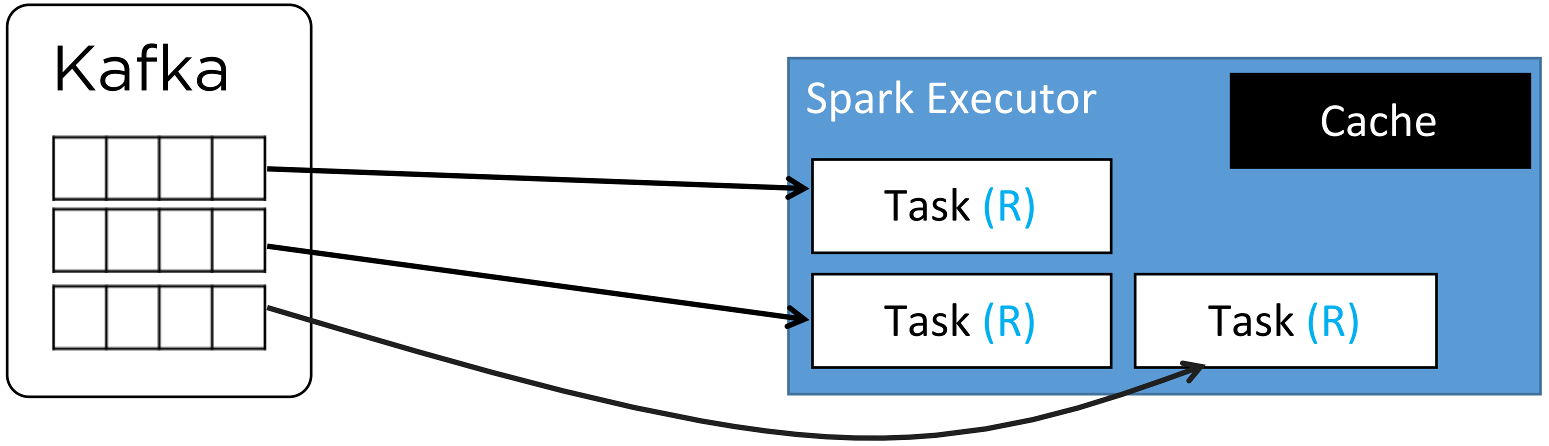
```
val kafkaStream = KafkaUtils.createStream[String, String, StringDecoder, StringDecoder](
  ssc, kafkaParams, Map(topic -> 1), StorageLevel.MEMORY_AND_DISK)
  .map(_._2)
```



Receiver-based Approach

Option 2: Create a Kafka stream per topic-partition

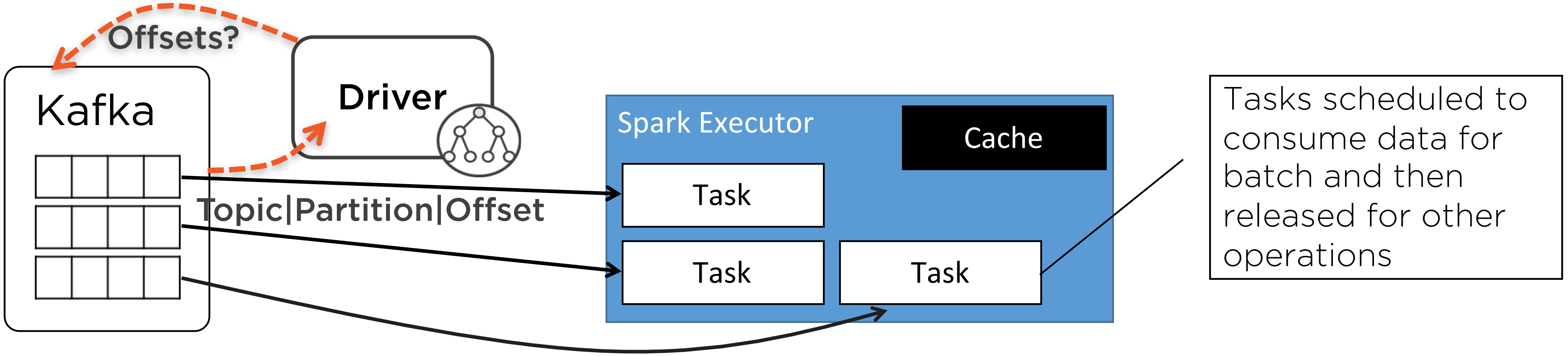
```
val receiverCount = 3
val kafkaStreams = (1 to receiverCount).map { _ =>
  KafkaUtils.createStream[String, String, StringDecoder, StringDecoder](
    ssc, kafkaParams, Map(topic -> 1), StorageLevel.MEMORY_AND_DISK)
}
val kafkaStream = ssc.union(kafkaStreams)
  .map(_._2)
```



Direct Approach

Driver determines offsets since last batch

```
val params = Map(
  "metadata.broker.list" -> "localhost:9092",
  "group.id" -> "lambda",
  "auto.offset.reset" -> "smallest"
)

KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](ssc, params, Set(topic))
  .map(_._2)
```

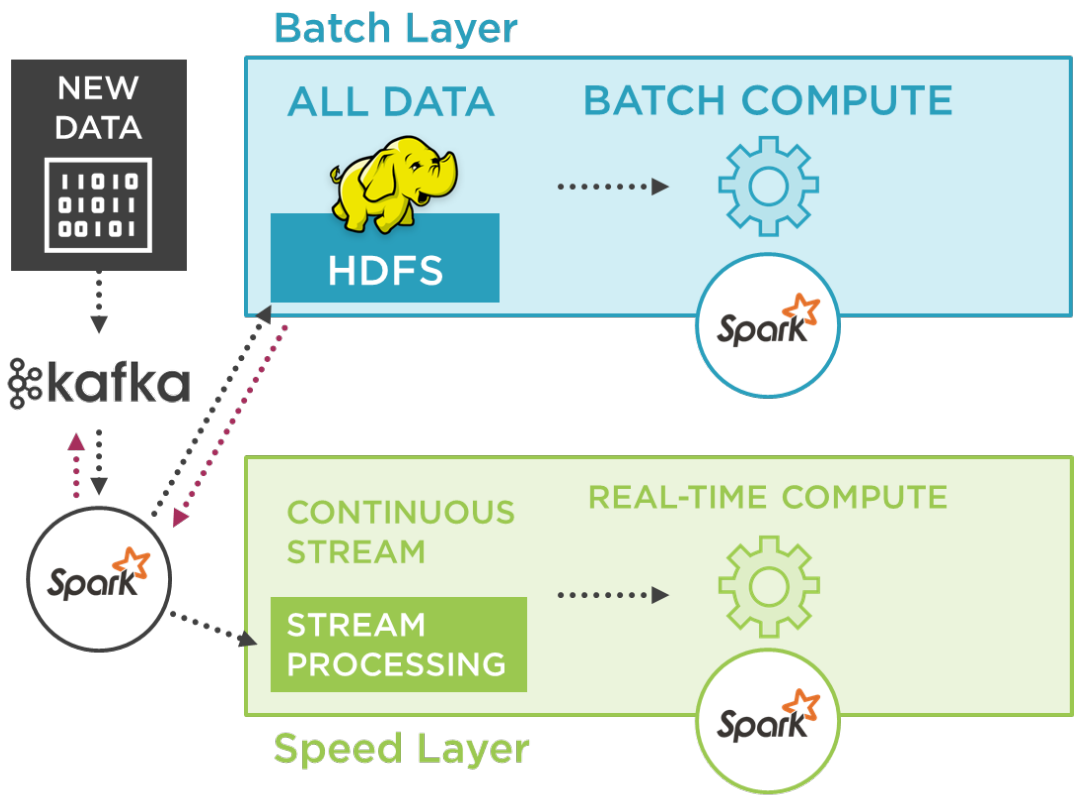

Offsets?
Kafka
Driver
Topic|Partition|Offset
Spark Executor
Cache
Task
Task
Task
Tasks scheduled to consume data for batch and then released for other operations

Demo

Save Data from Kafka to HDFS

Build Resiliency into the Application

- Recover from complete failures
- Allow for application updates



Kafka Direct Stream to HDFS

HDFS

../KafkaTopic/KafkaPartition

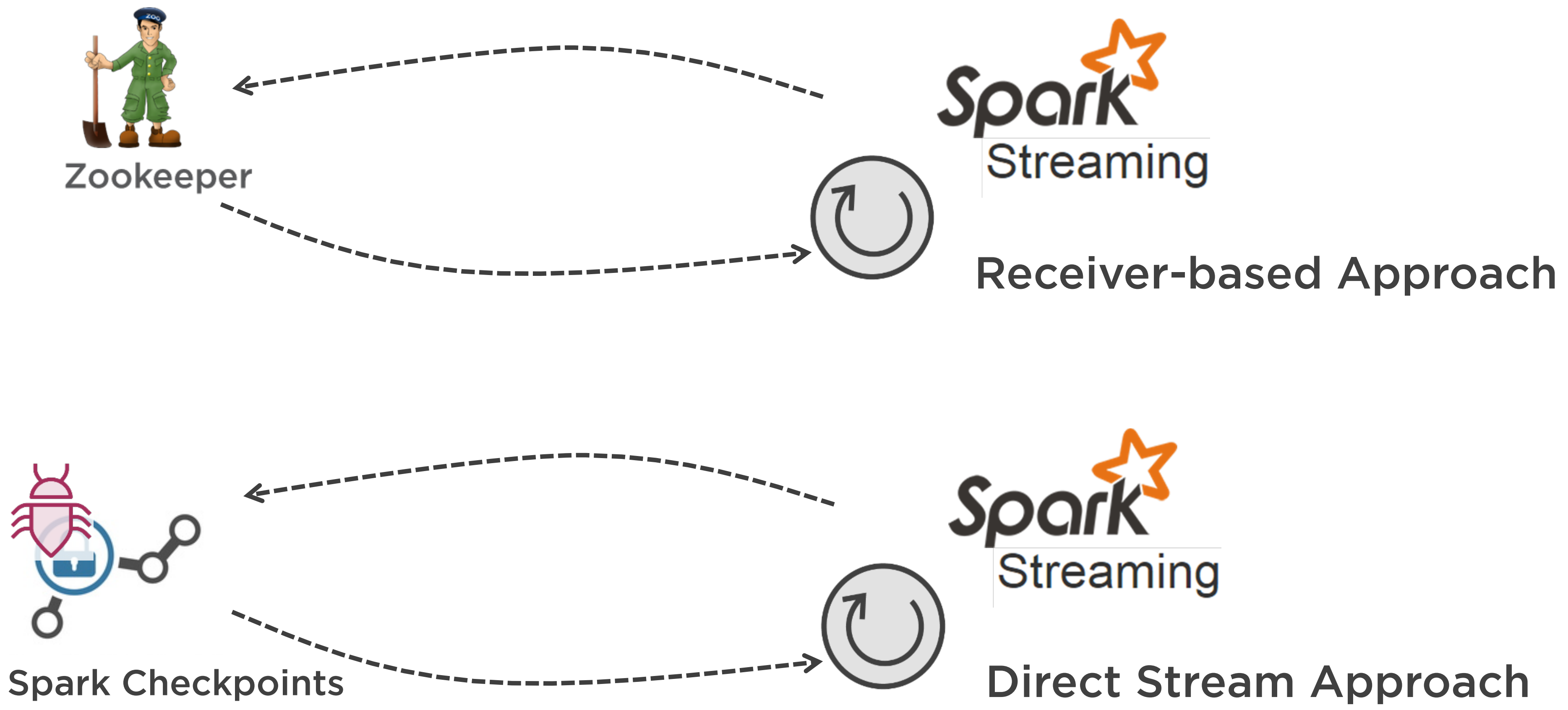
↳ data, fromOffset, untilOffset

> Direct Kafka stream means there's a 1-1 mapping between Kafka partition and Spark partition

HasOffsetRanges
RDD

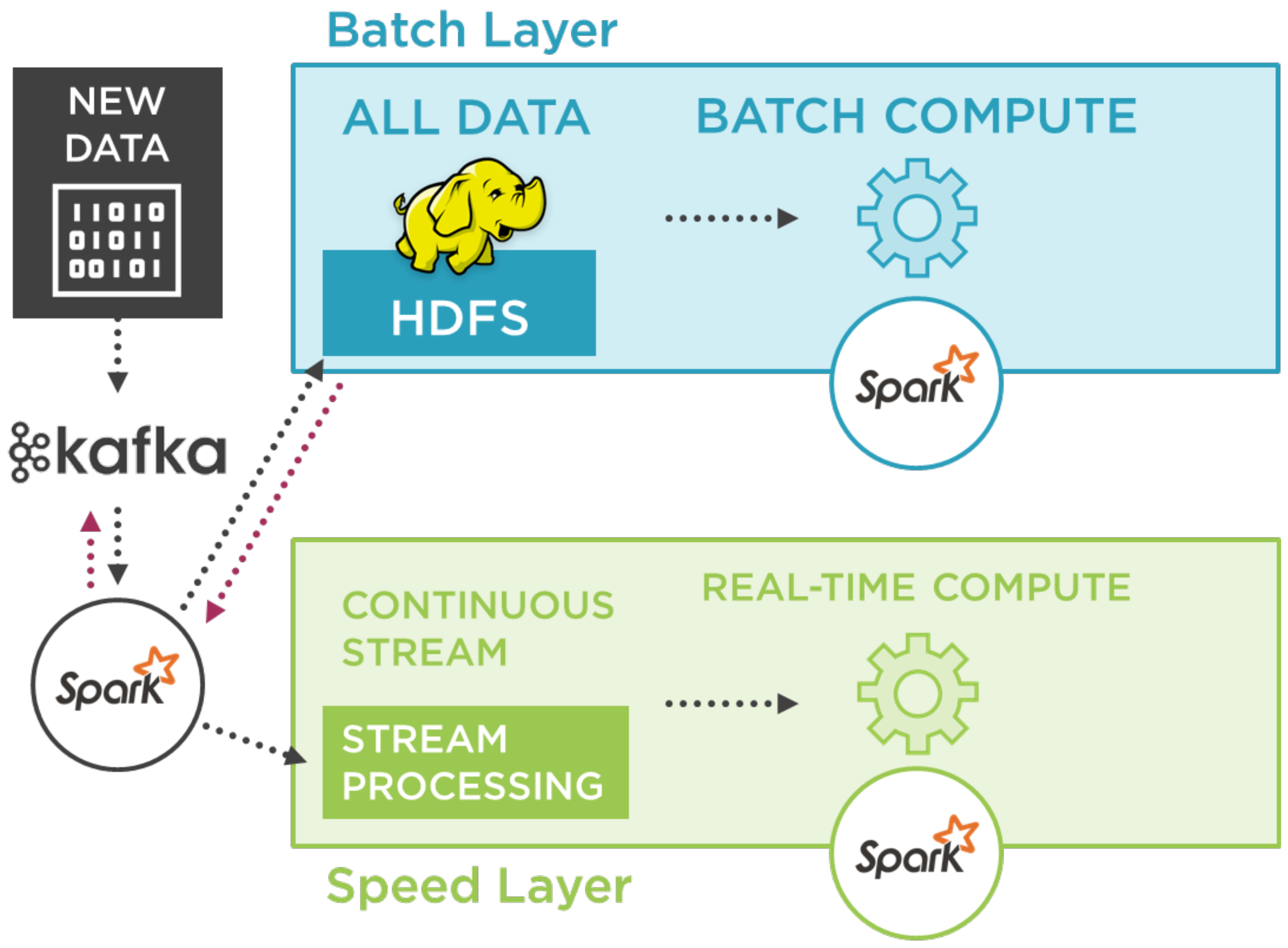
```
val offsetRanges = rdd.asInstanceOf[HasOffsetRanges].offsetRanges

offsetRanges(partitionNumber)
          .topic
          .partition
          .fromOffset
          .untilOffset
```

Streaming Resiliency



Summary

- **Apache Kafka**
  - Broker
  - Producer
  - Consumers and Partitions
- **Spark Streaming**
  - Receiver-based
  - Direct Stream
- **Resiliency**
  - Direct Stream Offsets
  - Recover from Upgrades
- **HDFS and Batch Layer Integration**



Batch Layer
NEW DATA
11010
01011
00101
kafka
Spark
ALL DATA
HDFS
BATCH COMPUTE
Spark
CONTINUOUS STREAM
STREAM PROCESSING
REAL-TIME COMPUTE
Spark
Speed Layer