Kafka pours and Spark resolves!

Alexey Zinovyev, Java/BigData Trainer in EPAM

With IT since 2007

With Java since 2009

With Hadoop since 2012

With Spark since 2014

With EPAM since 2015

About

Contacts

E-mail : Alexey_Zinovyev@epam.com

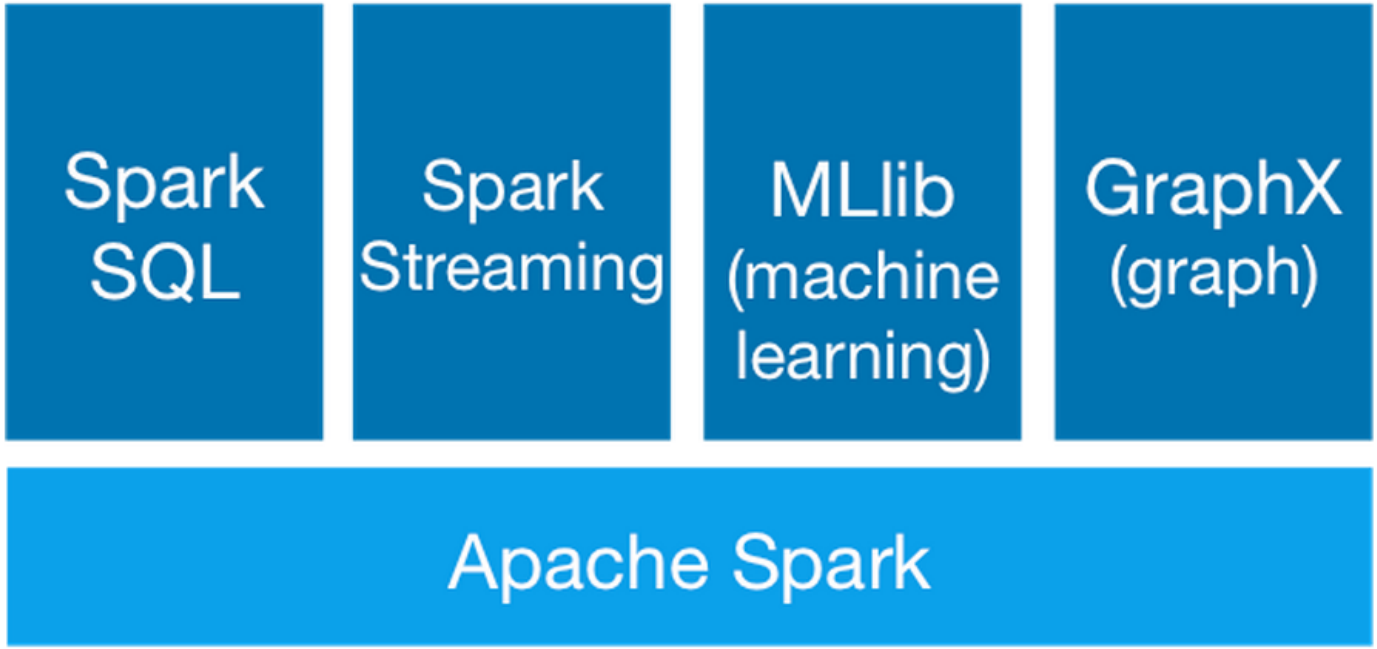
Twitter : @zaleslaw @BigDataRussia

Facebook: https://www.facebook.com/zaleslaw

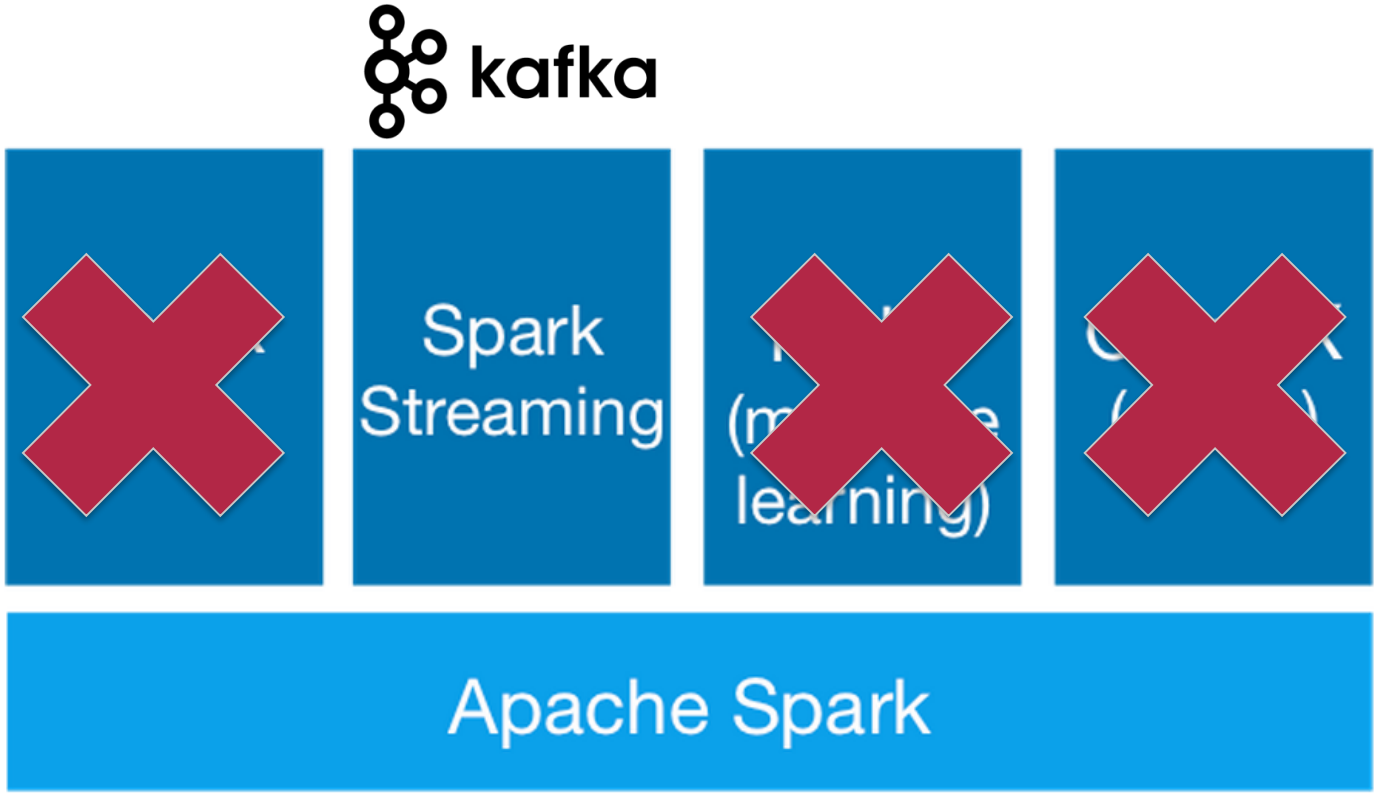
vk.com/big_data_russia **Big Data Russia**

vk.com/java_jvm **Java & JVM langs**

Spark Family



Spark Family



Pre-summary

- Before RealTime
- Spark + Cassandra
- Sending messages with Kafka
- DStream Kafka Consumer
- Structured Streaming in Spark 2.1
- Kafka Writer in Spark 2.2

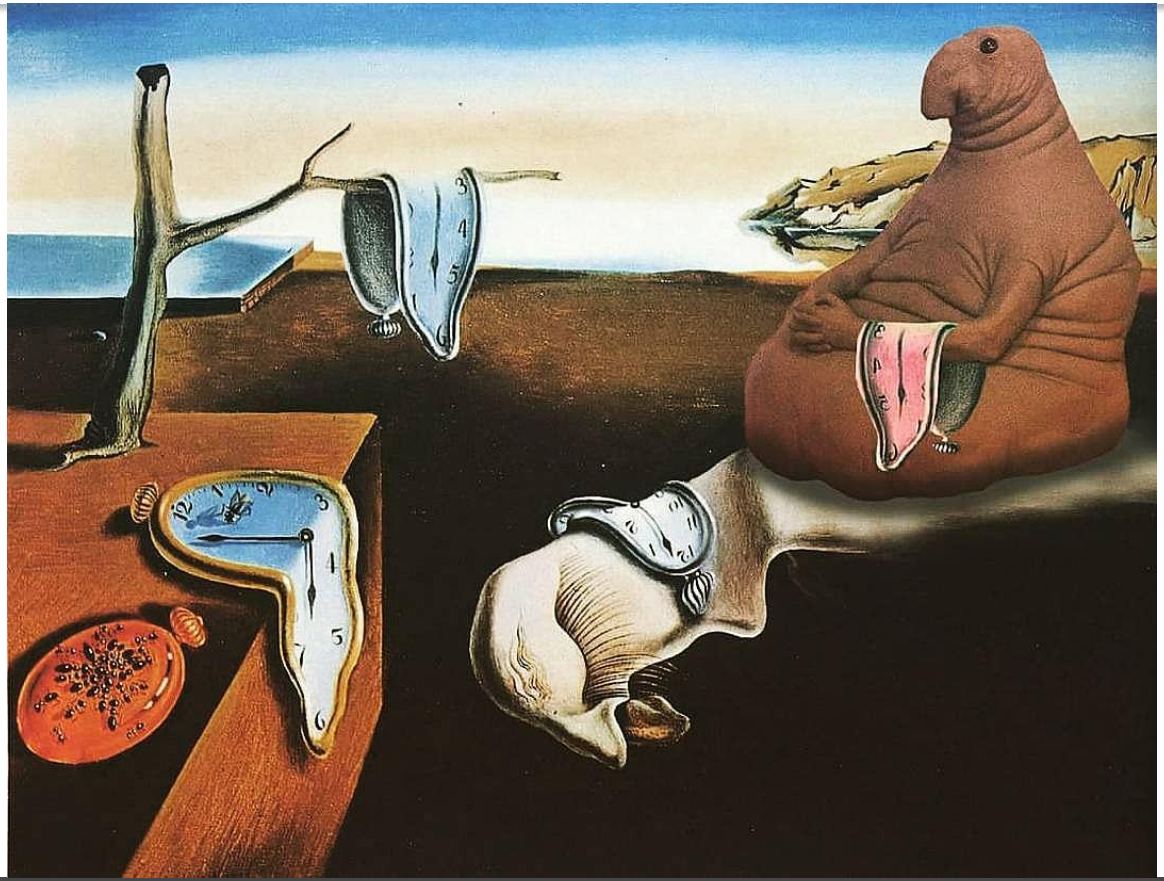
<epam>

< REAL-TIME


Big Data in 2014

Batch jobs produce reports. More and more..

But customer can wait forever (ok, 1h)



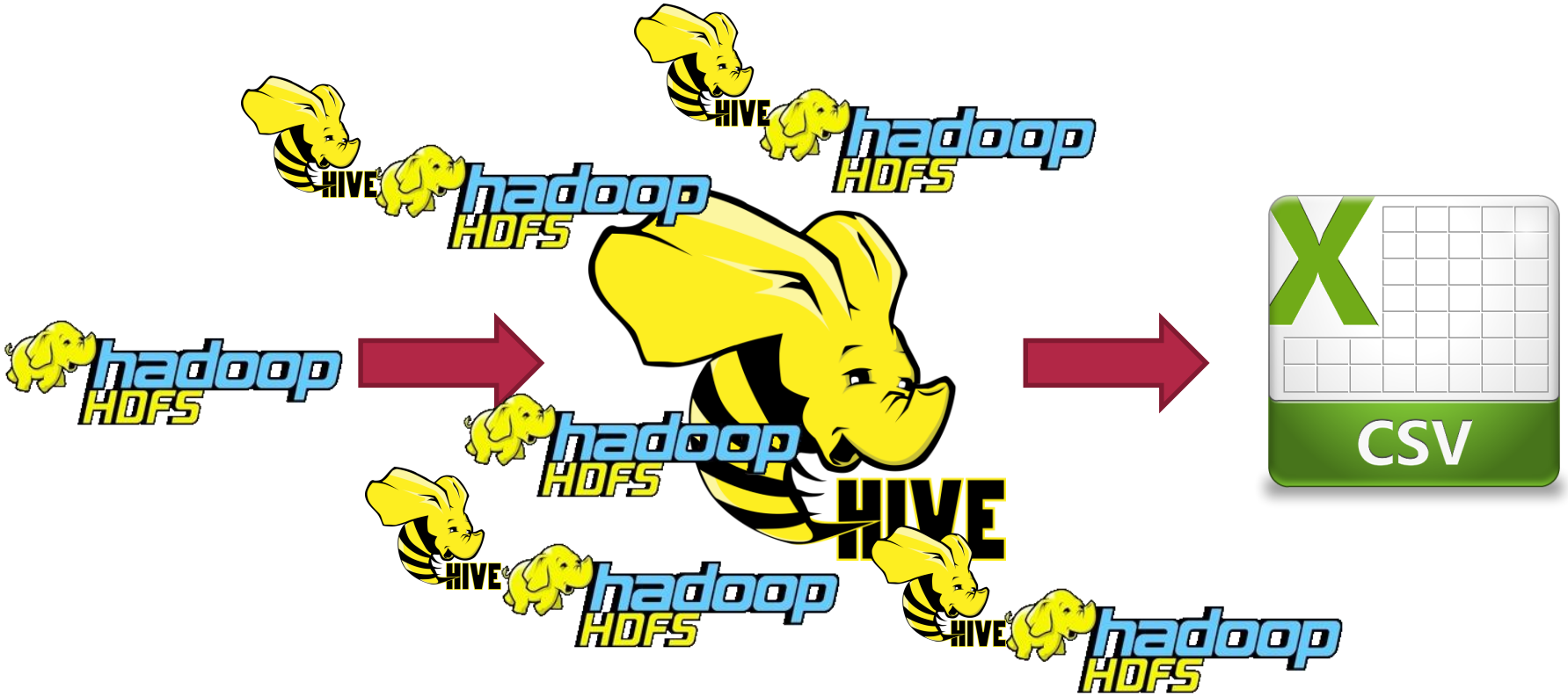

Hello, Smart Home
Big Data in 2017


Machine Learning EVERYWHERE

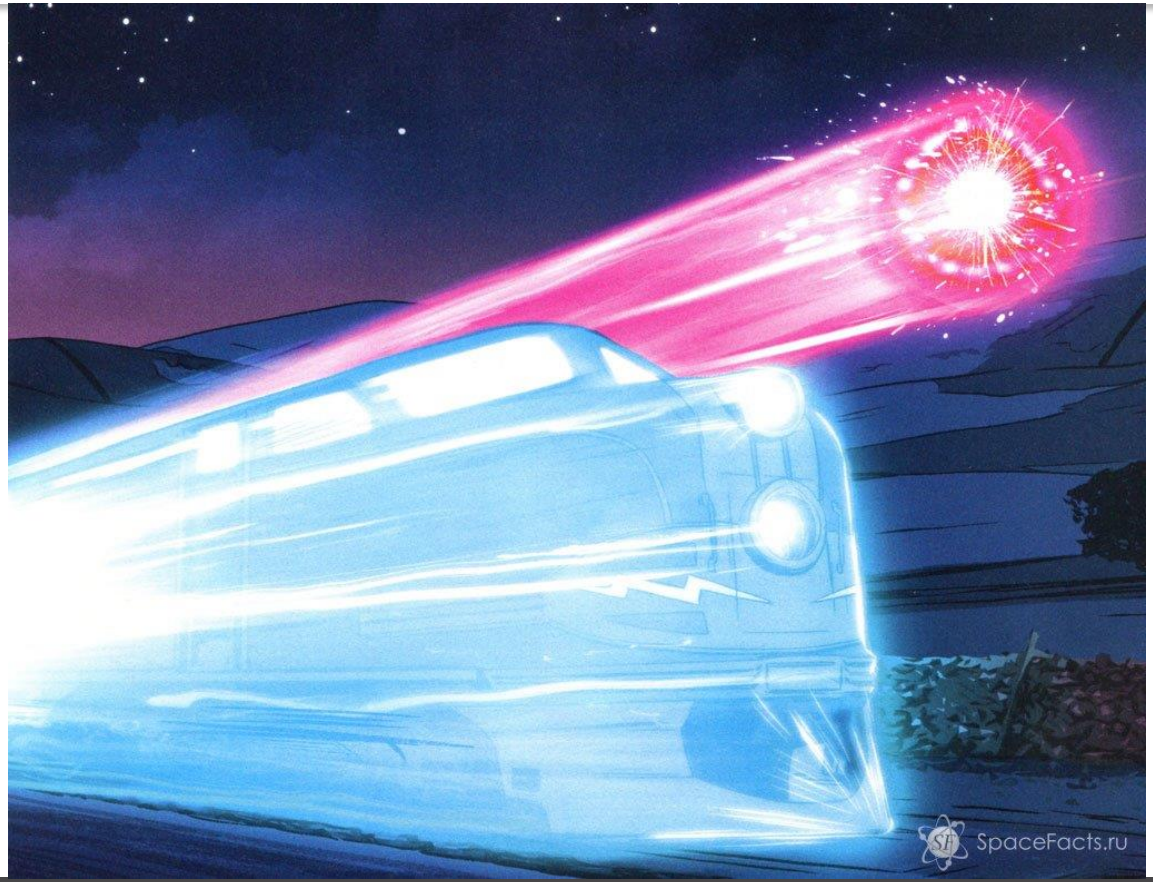
Data Lake in promotional brochure

Data Lake in production

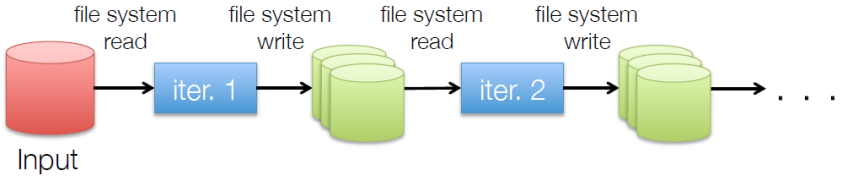
Simple Flow in Reporting/BI systems

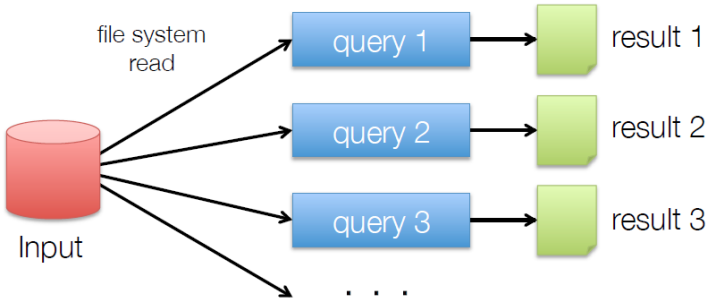


Let’s use Spark. It’s fast!

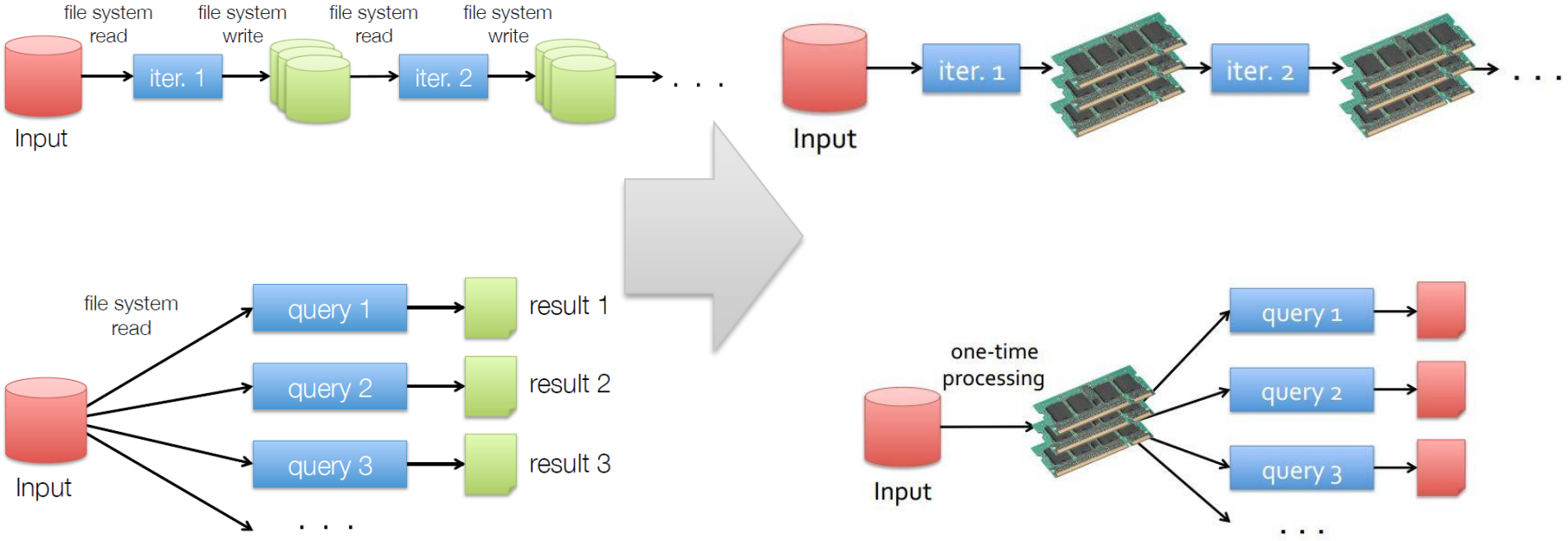


MapReduce vs Spark





MapReduce vs Spark

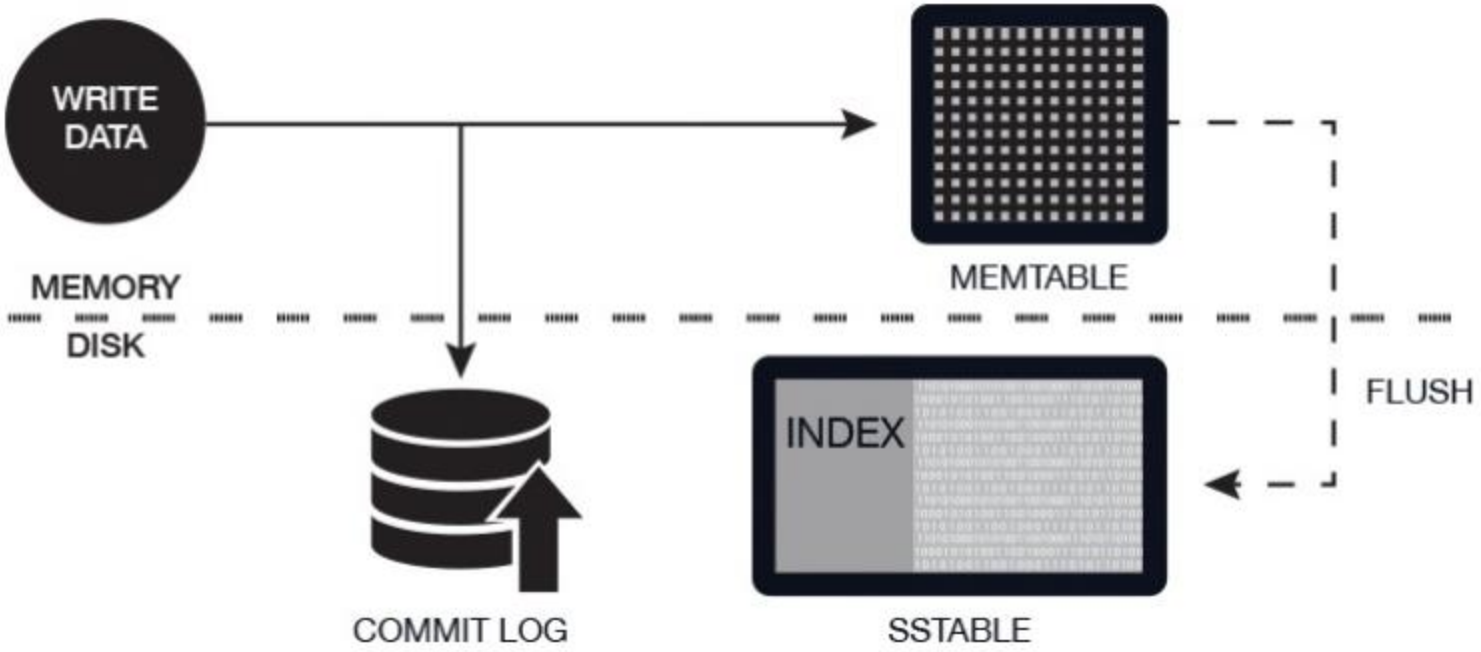


Simple Flow in Reporting/BI systems with Spark

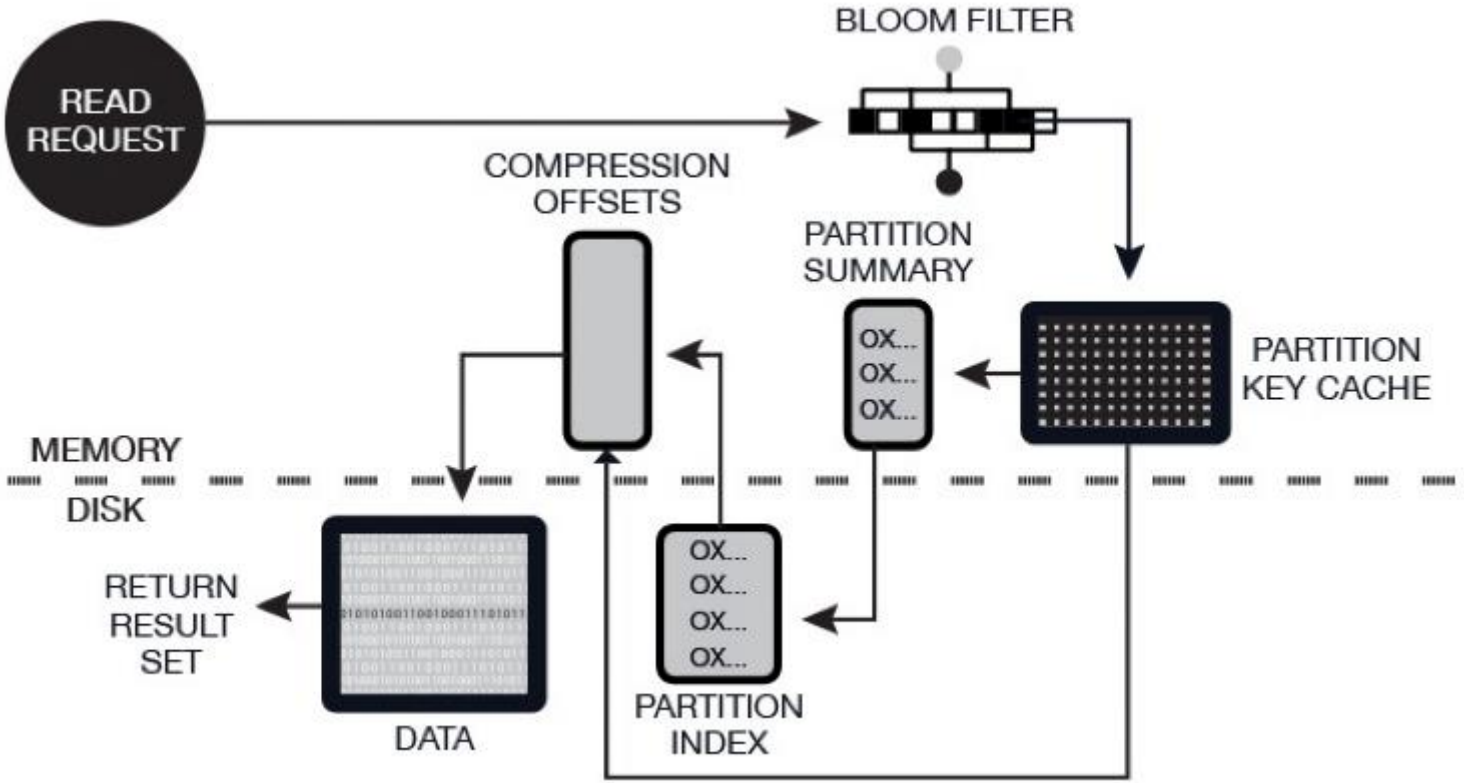
Spark handles last year logs with ease


Where can we store events?

Let’s use Cassandra to store events!



Let’s use Cassandra to read events!



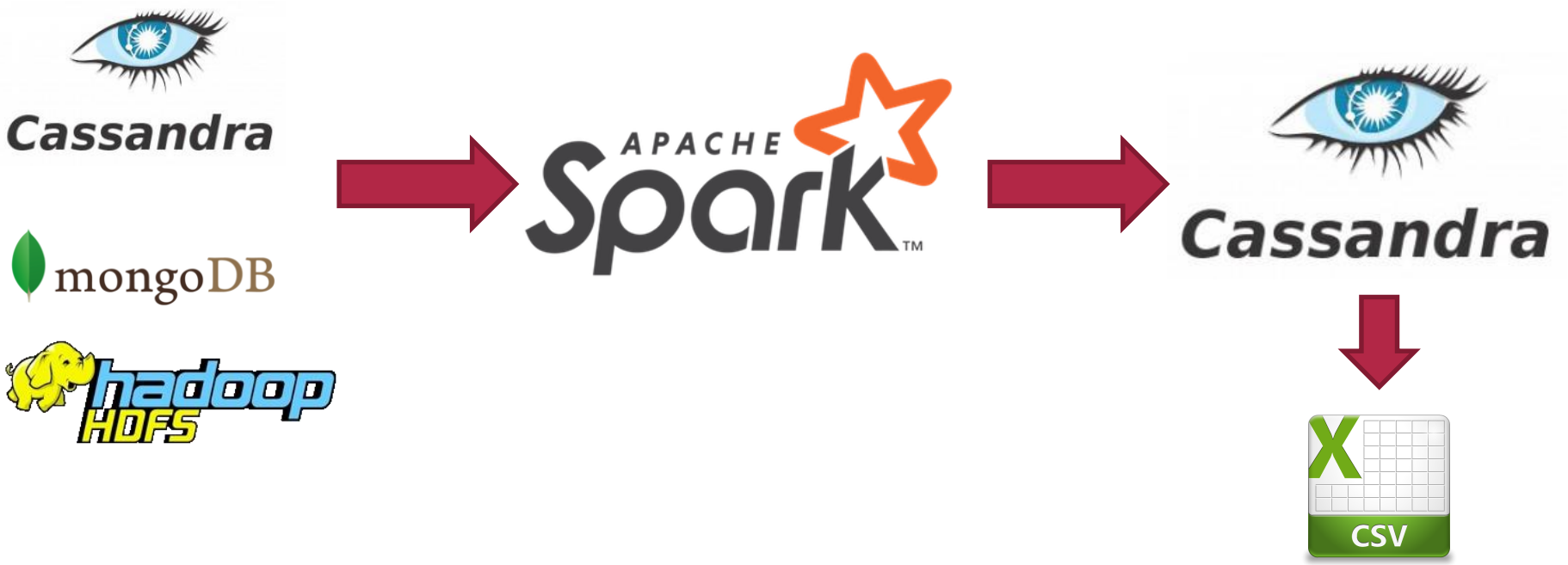
Cassandra

```
CREATE KEYSPACE mySpace WITH replication = {'class':
'SimpleStrategy', 'replication_factor': 1 };

USE test;

CREATE TABLE logs
( application TEXT,
  time TIMESTAMP,
  message TEXT,
  PRIMARY KEY (application, time));
```

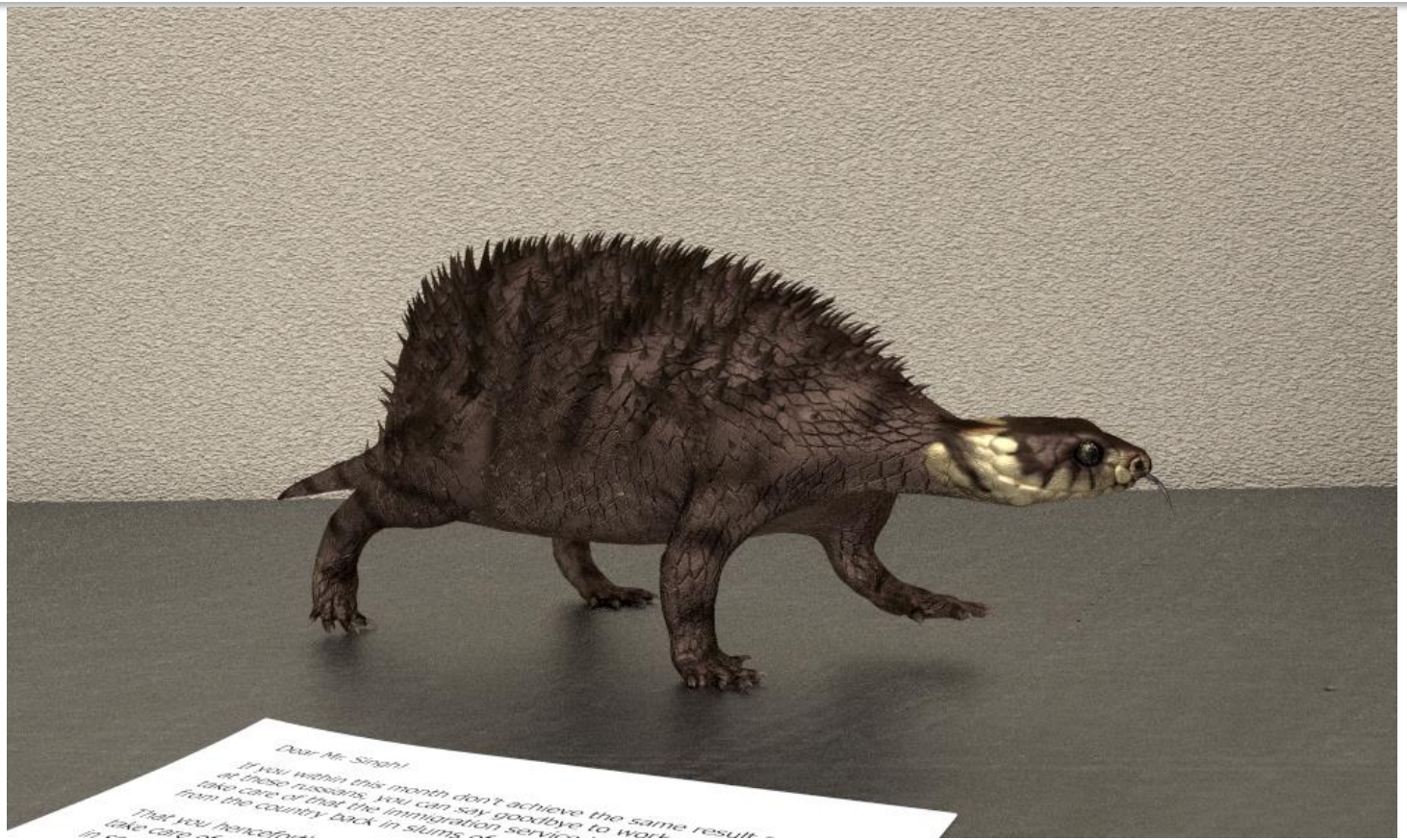
Cassandra to Spark

```
val dataSet = sqlContext
.read
.format("org.apache.spark.sql.cassandra")
.options(Map( "table" -> "logs", "keyspace" -> "mySpace"
))
.load()

dataSet
.filter("message = 'Log message'")
.show()
```

Simple Flow in Pre-Real-Time systems



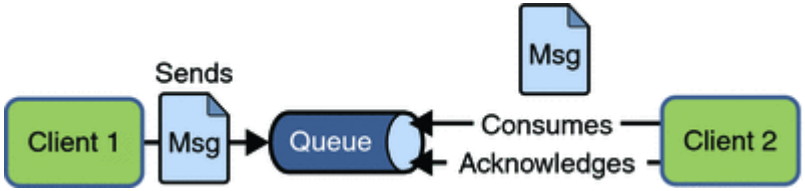
Spark cluster over Cassandra Cluster



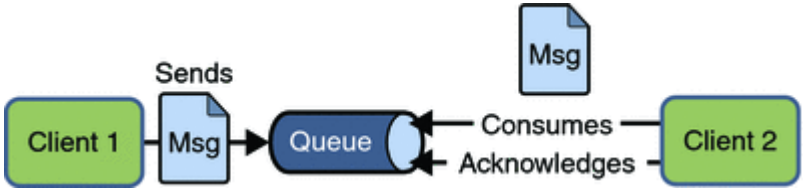
More events every second!

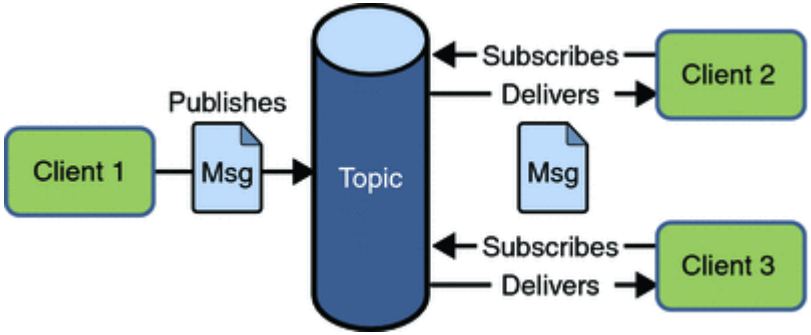
SENDING MESSAGES

Your Father's Messaging System

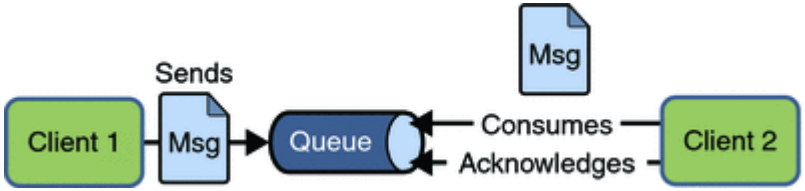


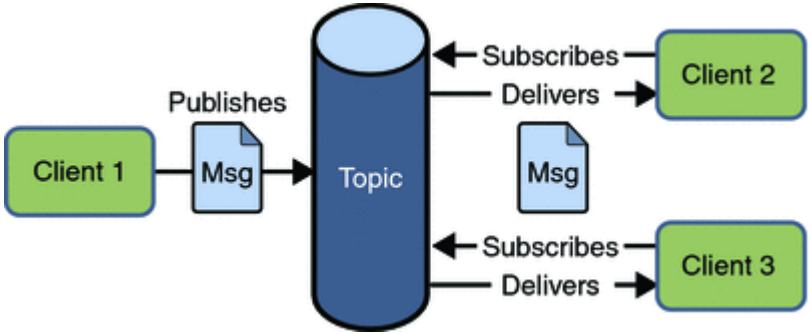
Your Father's Messaging System





Your Father's Messaging System






```
InitialContext ctx = new InitialContext();
QueueConnectionFactory f =
(QueueConnectionFactory)ctx.lookup("qCFactory");  QueueConnection con =
f.createQueueConnection();
con.start();
```

KAFKA

Kafka

- messaging system

Kafka

- messaging system
- distributed

Kafka

- messaging system
- distributed
- supports Publish-Subscribe model

Kafka

- messaging system
- distributed
- supports Publish-Subscribe model
- persists messages on disk

Kafka

- messaging system
- distributed
- supports Publish-Subscribe model
- persists messages on disk
- replicates within the cluster (integrated with Zookeeper)

The main benefits of Kafka

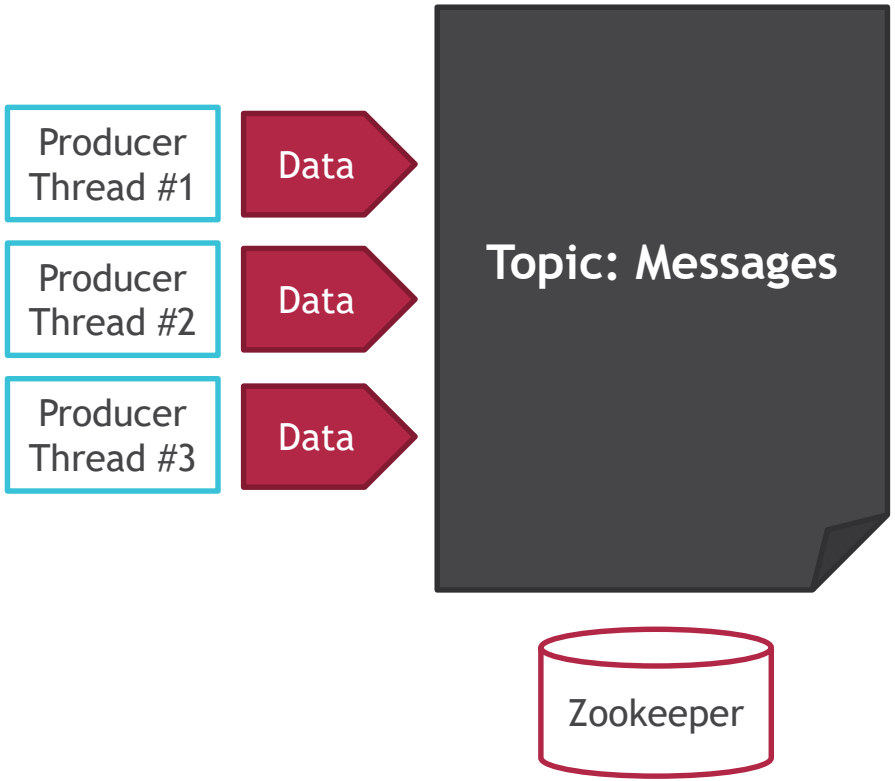
Scalability with *zero down time*

Zero data loss due to replication

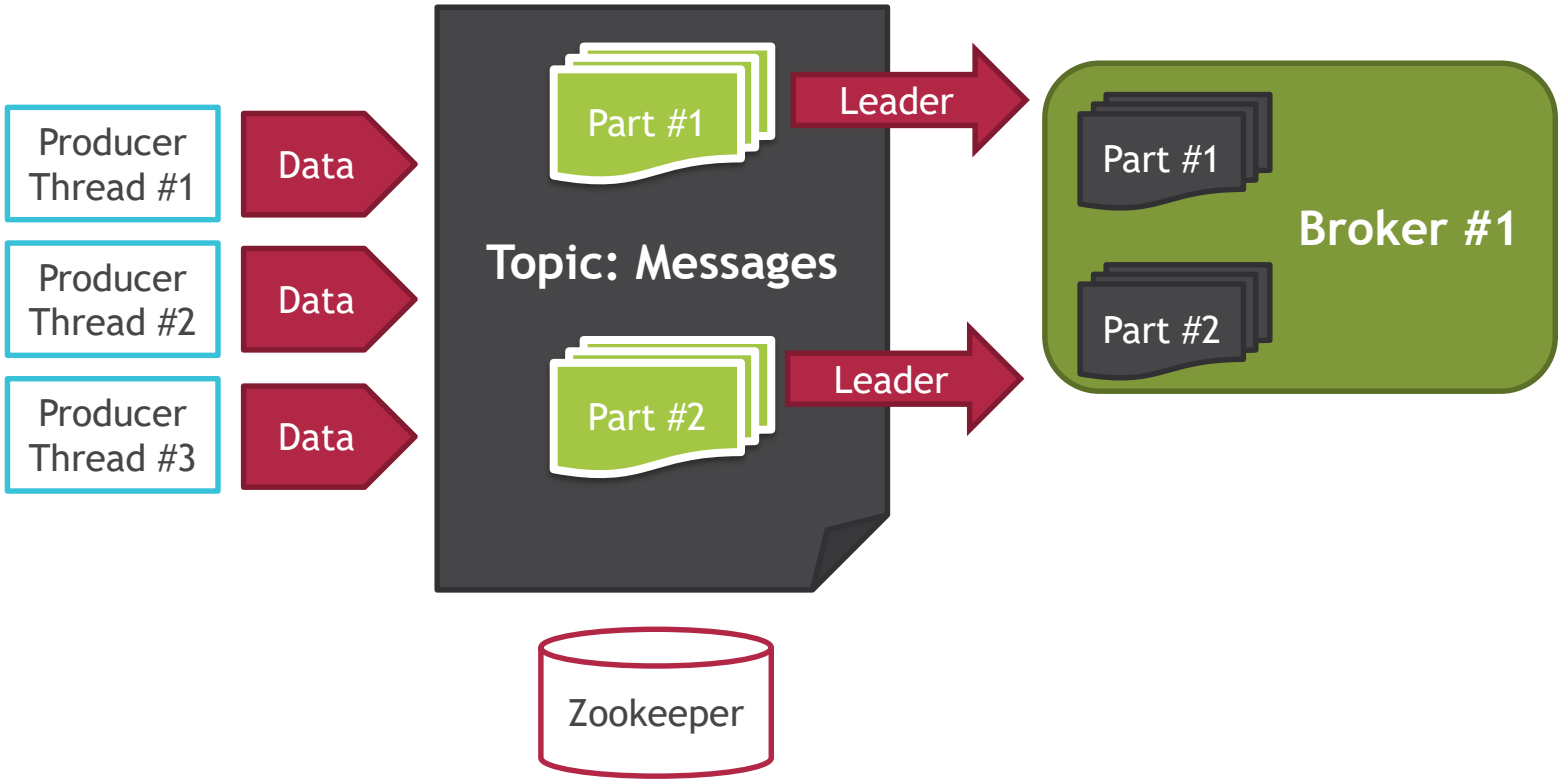
Kafka Cluster consists of …

- brokers (leader or follower)
- topics (>= 1 partition)
- partitions
- partition offsets
- replicas of partition
- producers/consumers

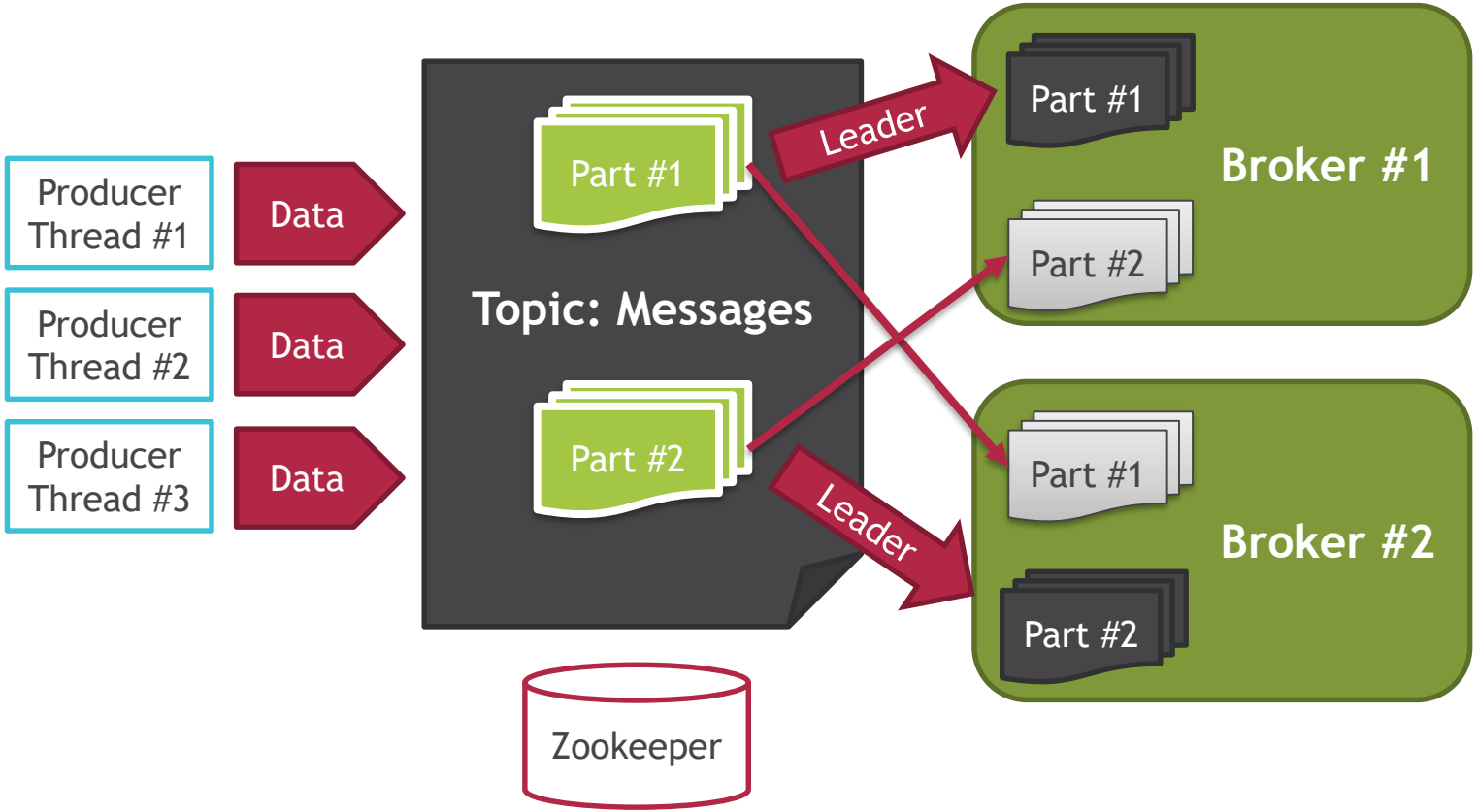
Kafka Components with topic “messages” #1



Kafka Components with topic “messages” #2



Kafka Components with topic “messages” #3

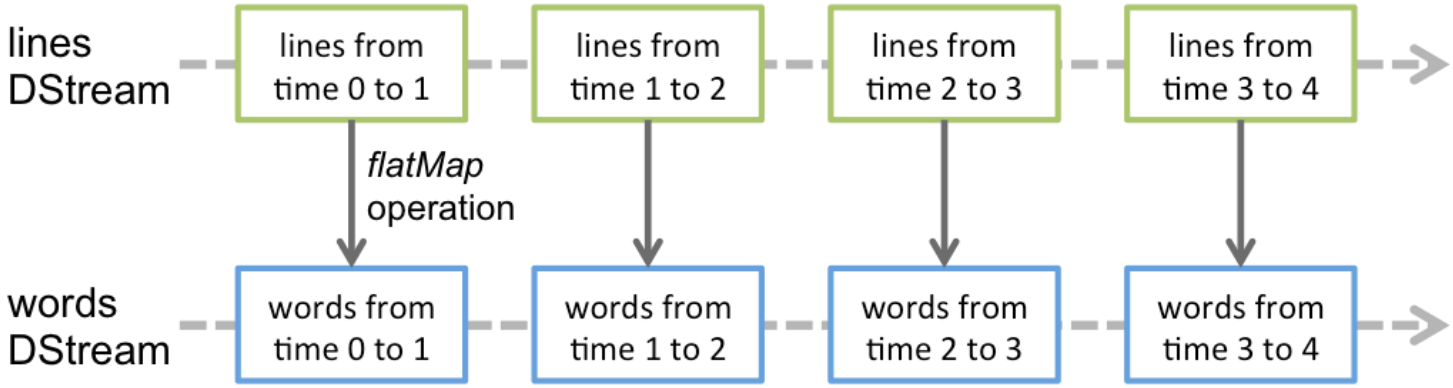



Why do we need Zookeeper?

Kafka Demo

REAL TIME WITH DSTREAMS

RDD Factory☺


lines
DStream
lines from
time 0 to 1
lines from
time 1 to 2
lines from
time 2 to 3
lines from
time 3 to 4
flatMap
operation
words
DStream
words from
time 0 to 1
words from
time 1 to 2
words from
time 2 to 3
words from
time 3 to 4

From socket to console with DStreams

DStream

```
val conf = new SparkConf().setMaster("local[2]")
.setAppName("NetworkWordCount")
```

DStream

```
val conf = new SparkConf().setMaster("local[2]")
.setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))




ssc.start()
ssc.awaitTermination()
```

DStream

```
val conf = new SparkConf().setMaster("local[2]")
.setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)




ssc.start()
ssc.awaitTermination()
```

DStream

```
val conf = new SparkConf().setMaster("local[2]")
.setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))



ssc.start()
ssc.awaitTermination()
```

DStream

```
val conf = new SparkConf().setMaster("local[2]")
.setAppName("NetworkWordCount")
val ssc = new StreamingContext(conf, Seconds(1))
val lines = ssc.socketTextStream("localhost", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)
wordCounts.print()

ssc.start()
ssc.awaitTermination()
```

Kafka as a main entry point for Spark

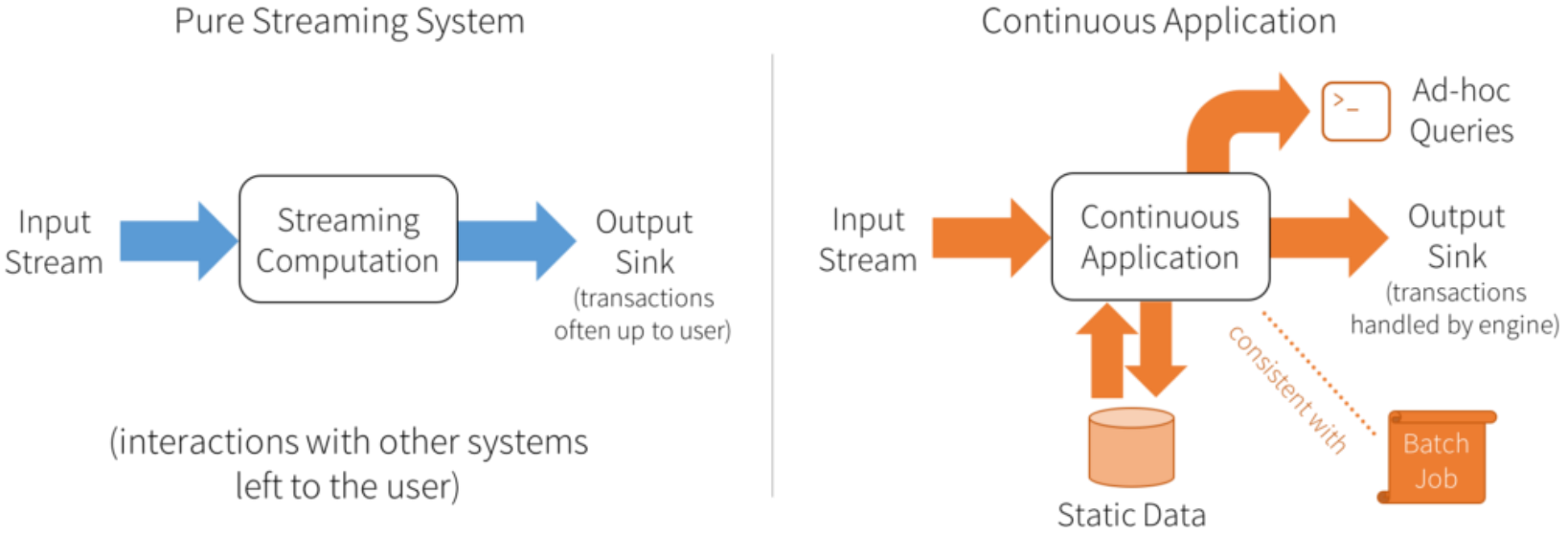
DStreams Demo


How to avoid DStreams with RDD-like API?

<epam>

SPARK 2.2 DISCUSSION

Continuous Applications



Continuous Applications cases

- Updating data that will be served in real time
- Extract, transform and load (ETL)
- Creating a real-time version of an existing batch job
- Online machine learning

The main concept of Structured Streaming

You can express your streaming computation the same way you would express a batch computation on static data.

Batch Spark 2.2

```
// Read JSON once from S3
logsDF = spark.read.json("s3://logs")

// Transform with DataFrame API and save
logsDF.select("user", "url", "date")
      .write.parquet("s3://out")
```

Real Time Spark 2.2

```
// Read JSON continuously from S3
logsDF = spark.readStream.json("s3://logs")

// Transform with DataFrame API and save
logsDF.select("user", "url", "date")
      .writeStream.parquet("s3://out")
      .start()
```

WordCount from Socket

```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()
```

WordCount from Socket

```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()

val words = lines.as[String].flatMap(_.split(" "))
```

WordCount from Socket

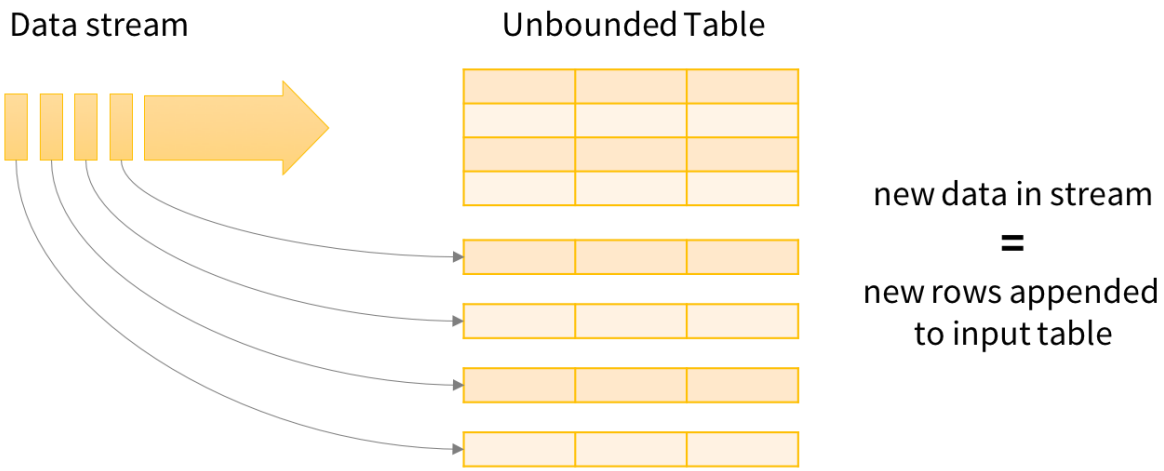
```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()

val words = lines.as[String].flatMap(_.split(" "))

val wordCounts = words.groupBy("value").count()
```

WordCount from Socket

```
val lines = spark.readStream
  .format("socket")
  .option("host", "localhost")
  .option("port", 9999)
  .load()

val words                          ing].flatMap(_.split(" "))

val wordCounts = words.groupBy("value").count()
```

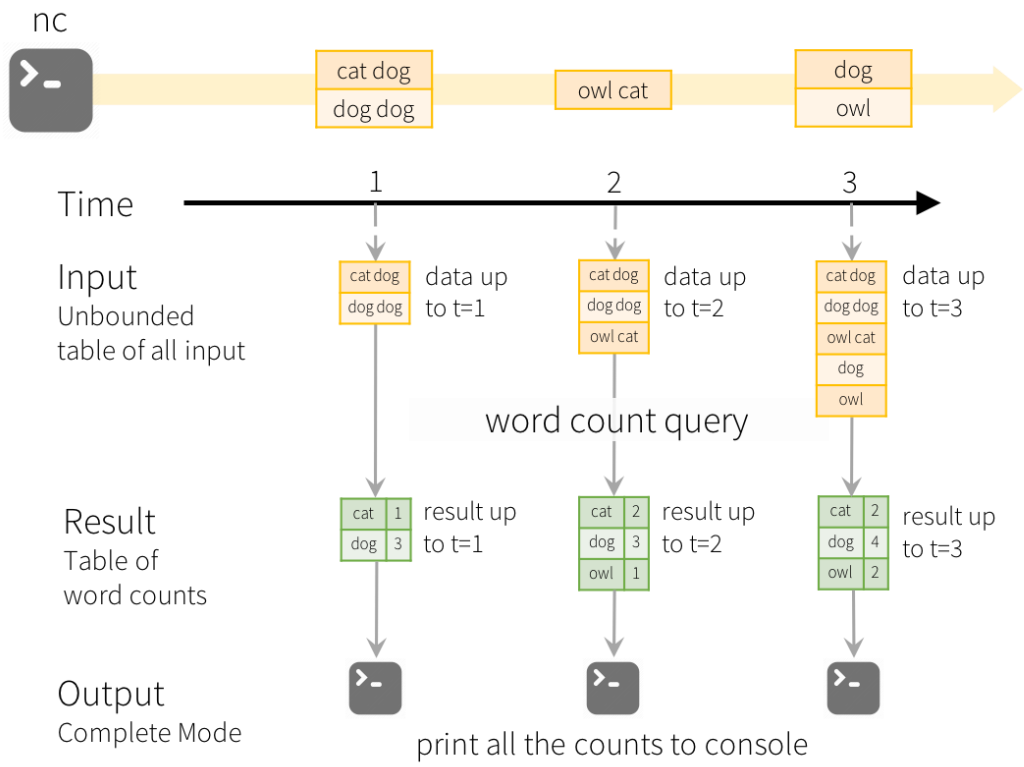
Don’t forget to start Streaming

Unlimited Table



Data stream as an unbounded Input Table

WordCount with Structured Streaming [Complete Mode]



Kafka -> Structured Streaming -> Console

Kafka To Console Demo

OPERATIONS

You can …

- filter
- sort
- aggregate
- join
- foreach
- explain

Operators Demo


How it works?

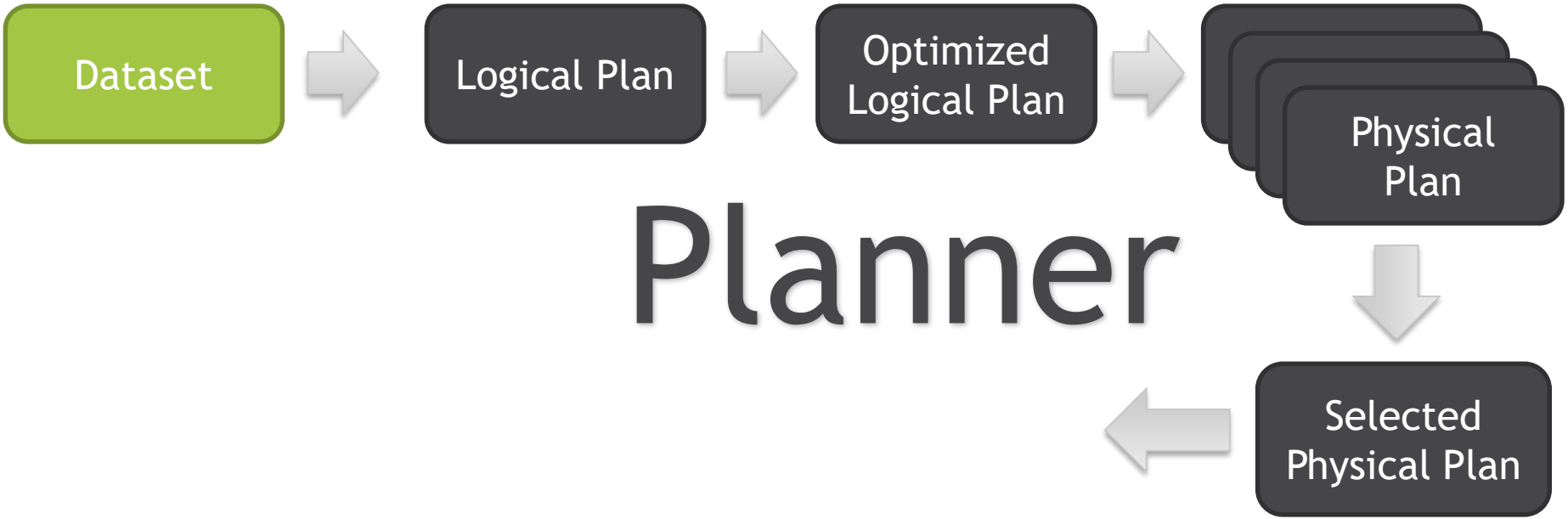
Deep Diving in Spark Internals

Deep Diving in Spark Internals

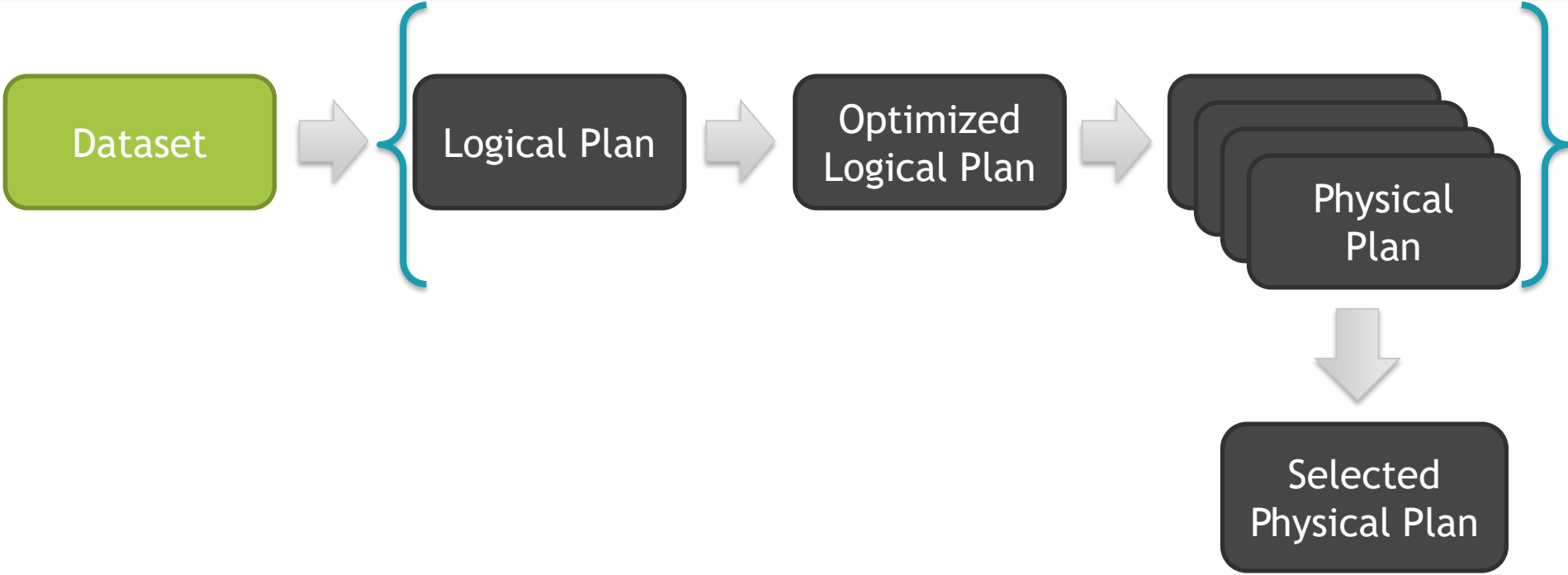
Deep Diving in Spark Internals

Deep Diving in Spark Internals

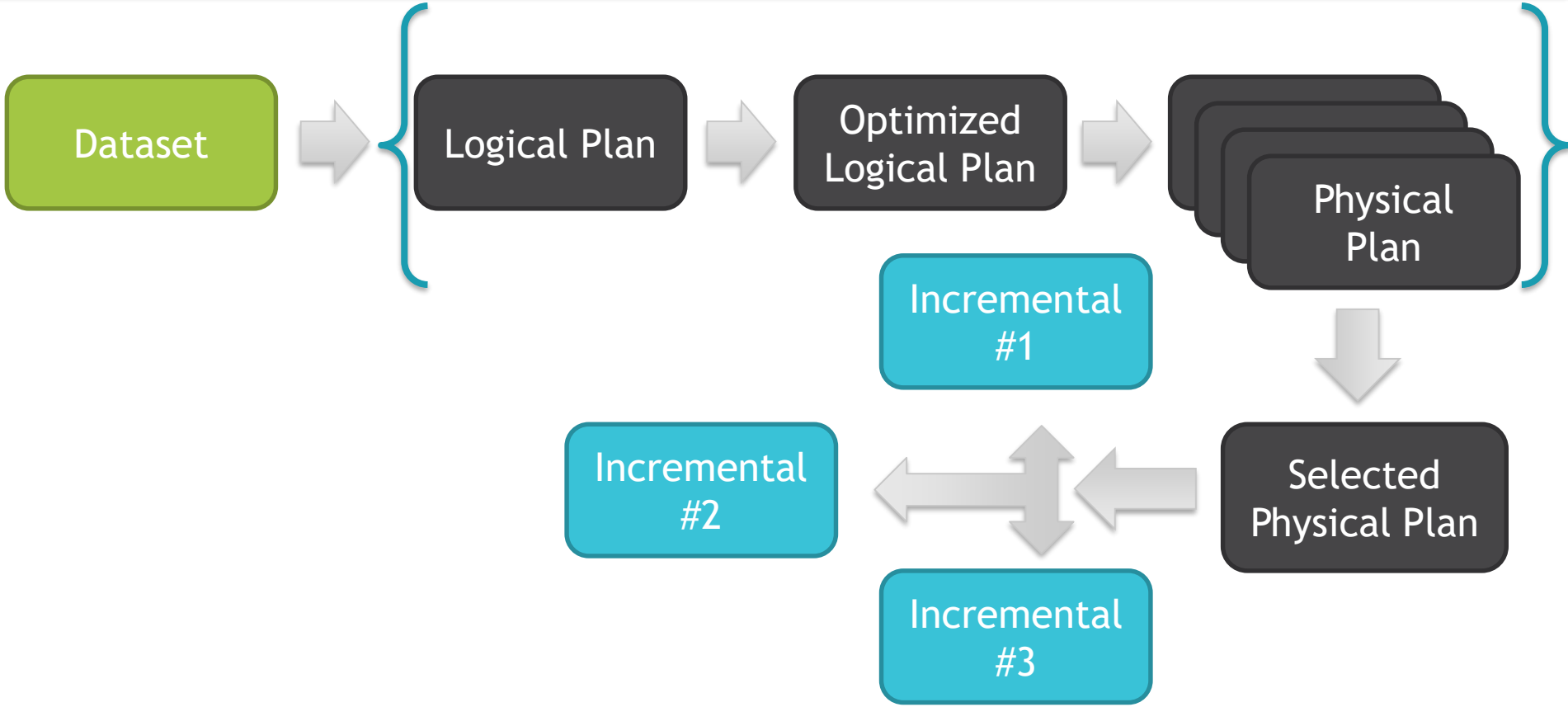
Deep Diving in Spark Internals



Deep Diving in Spark Internals



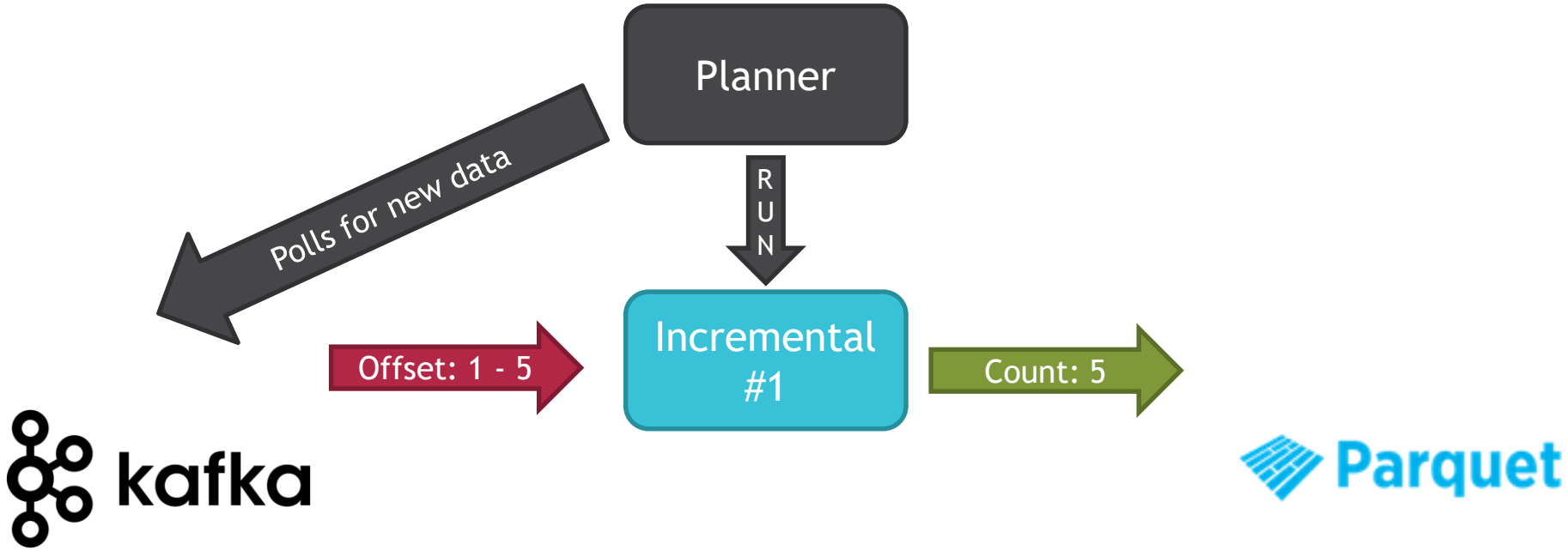
Deep Diving in Spark Internals



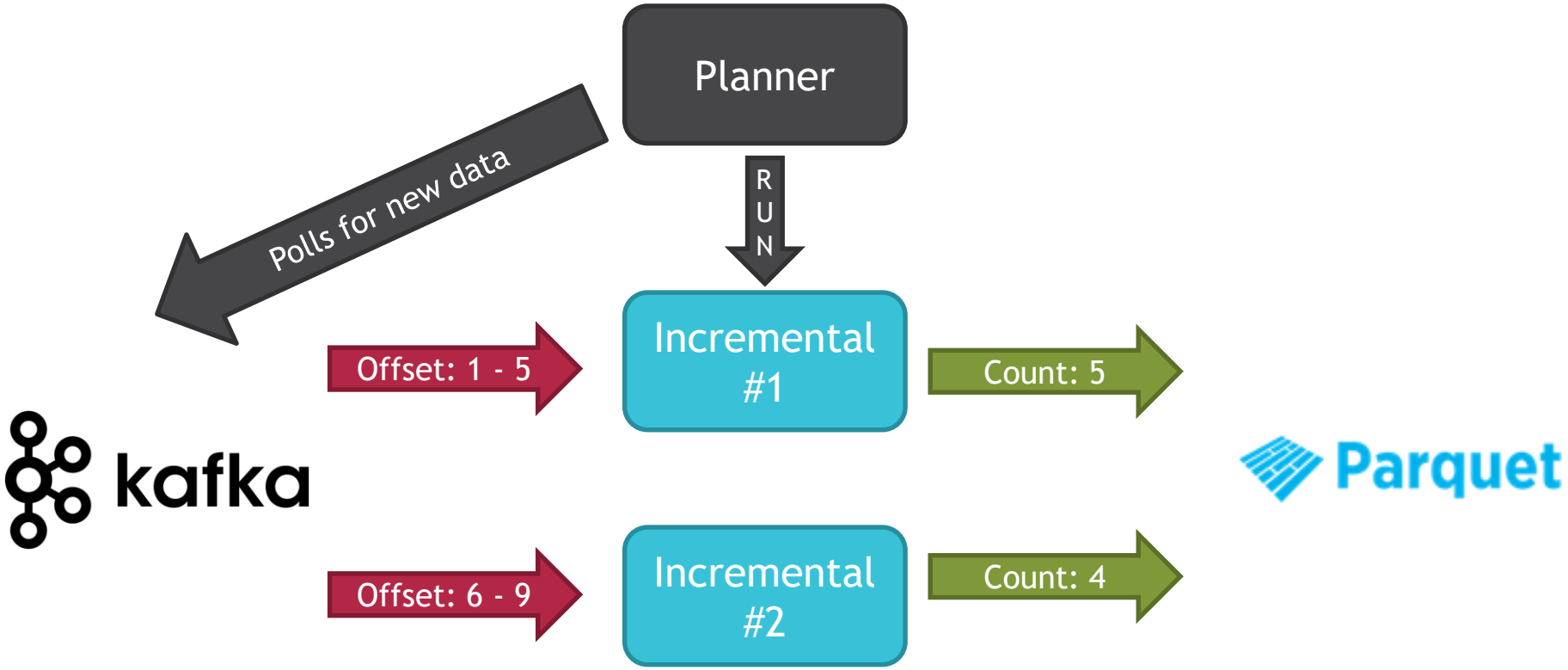
Incremental Execution: Planner polls

kafka

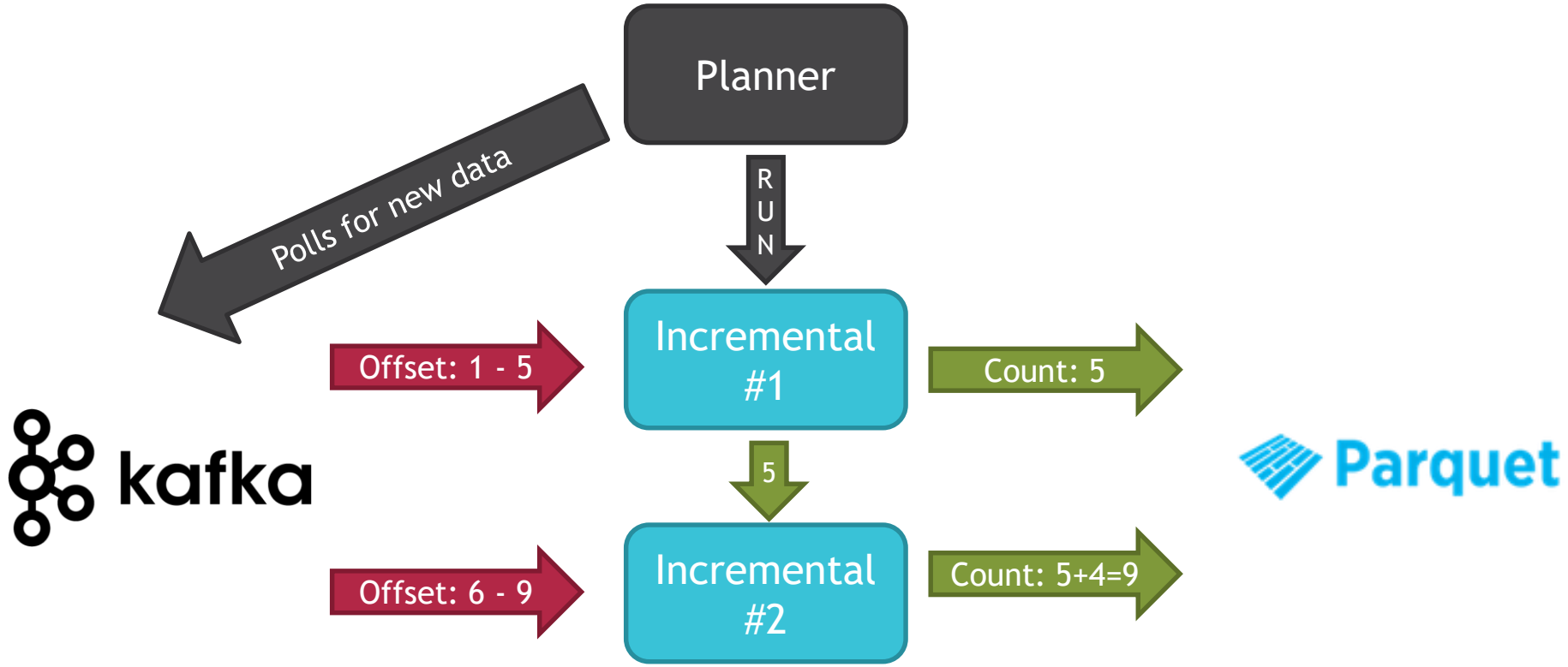
Incremental Execution: Planner runs



Incremental Execution: Planner runs #2

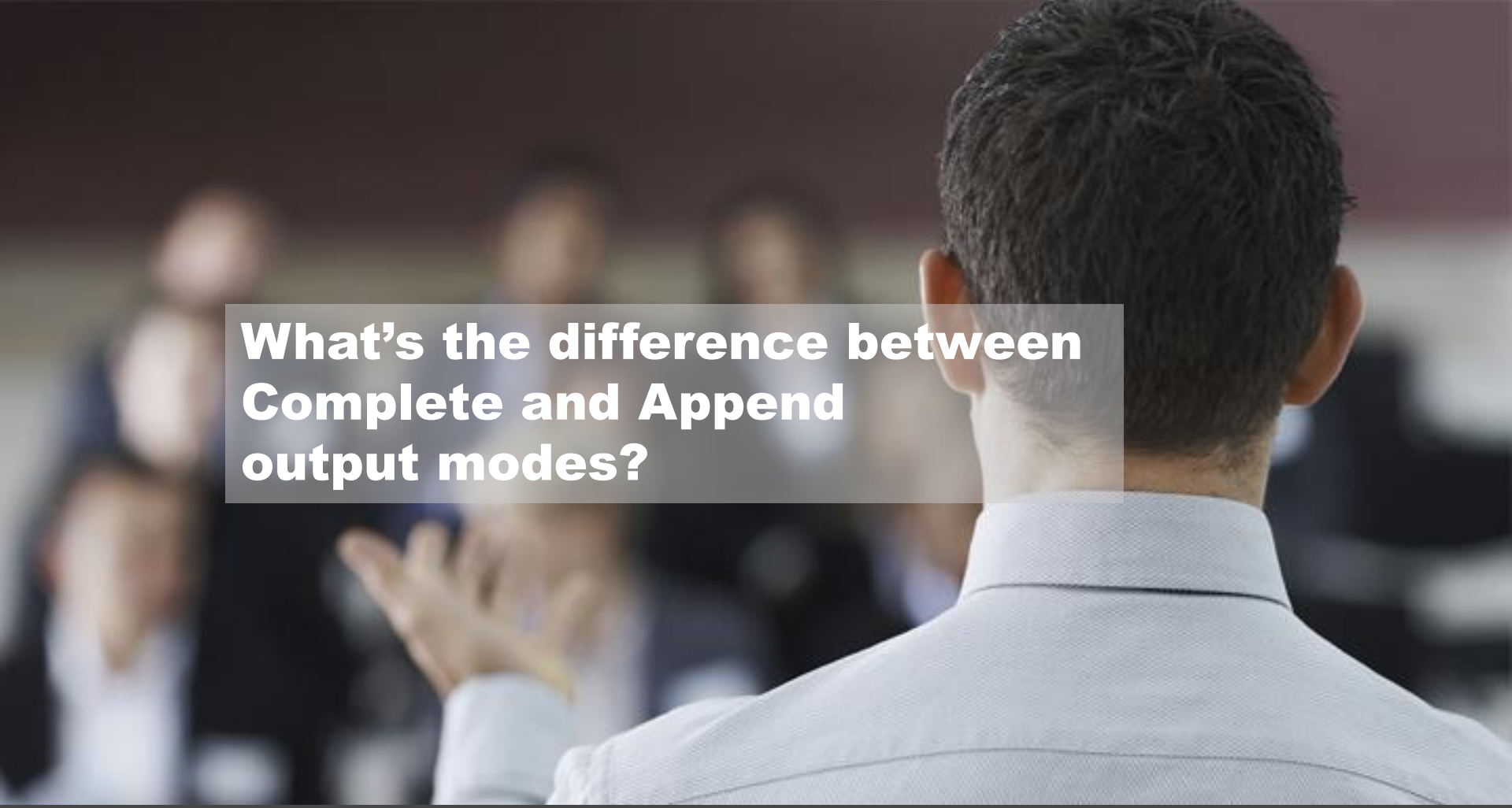


Aggregation with State



DataSet.explain()

```
== Physical Plan ==
Project [avg(price)#43,carat#45]
+- SortMergeJoin [color#21], [color#47]
   :- Sort [color#21 ASC], false, 0
   :  +- TungstenExchange hashpartitioning(color#21,200), None
   :     +- Project [avg(price)#43,color#21]
   :        +- TungstenAggregate(key=[cut#20,color#21], functions=[(avg(cast(price#25 as
bigint)),mode=Final,isDistinct=false)], output=[color#21,avg(price)#43])
   :           +- TungstenExchange hashpartitioning(cut#20,color#21,200), None
   :              +- TungstenAggregate(key=[cut#20,color#21],
functions=[(avg(cast(price#25 as bigint)),mode=Partial,isDistinct=false)],
output=[cut#20,color#21,sum#58,count#59L])
   :                 +- Scan CsvRelation(-----)
   +- Sort [color#47 ASC], false, 0
      +- TungstenExchange hashpartitioning(color#47,200), None
         +- ConvertToUnsafe
            +- Scan CsvRelation(----)
```


What’s the difference between Complete and Append output modes?

COMPLETE, APPEND & UPDATE

There are two main modes and one in future

- append (default)

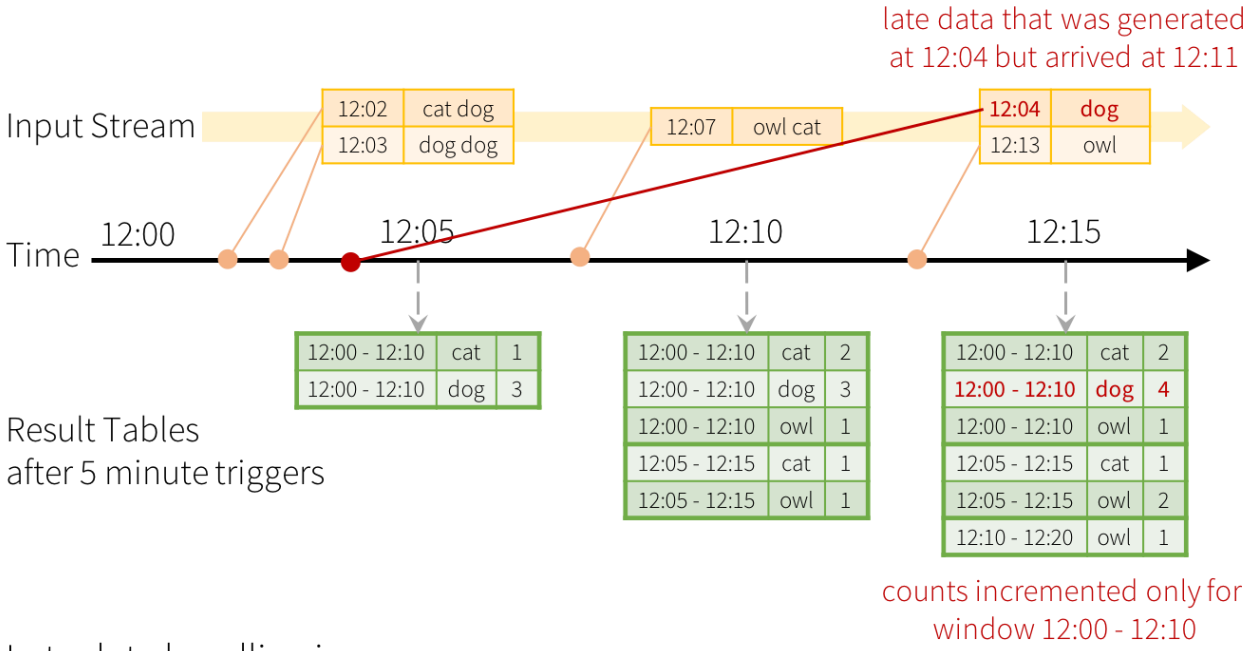
There are two main modes and one in future

- append (default)
- complete

There are two main modes and one in future

- append (default)
- complete
- update [in dreams]

Aggregation with watermarks

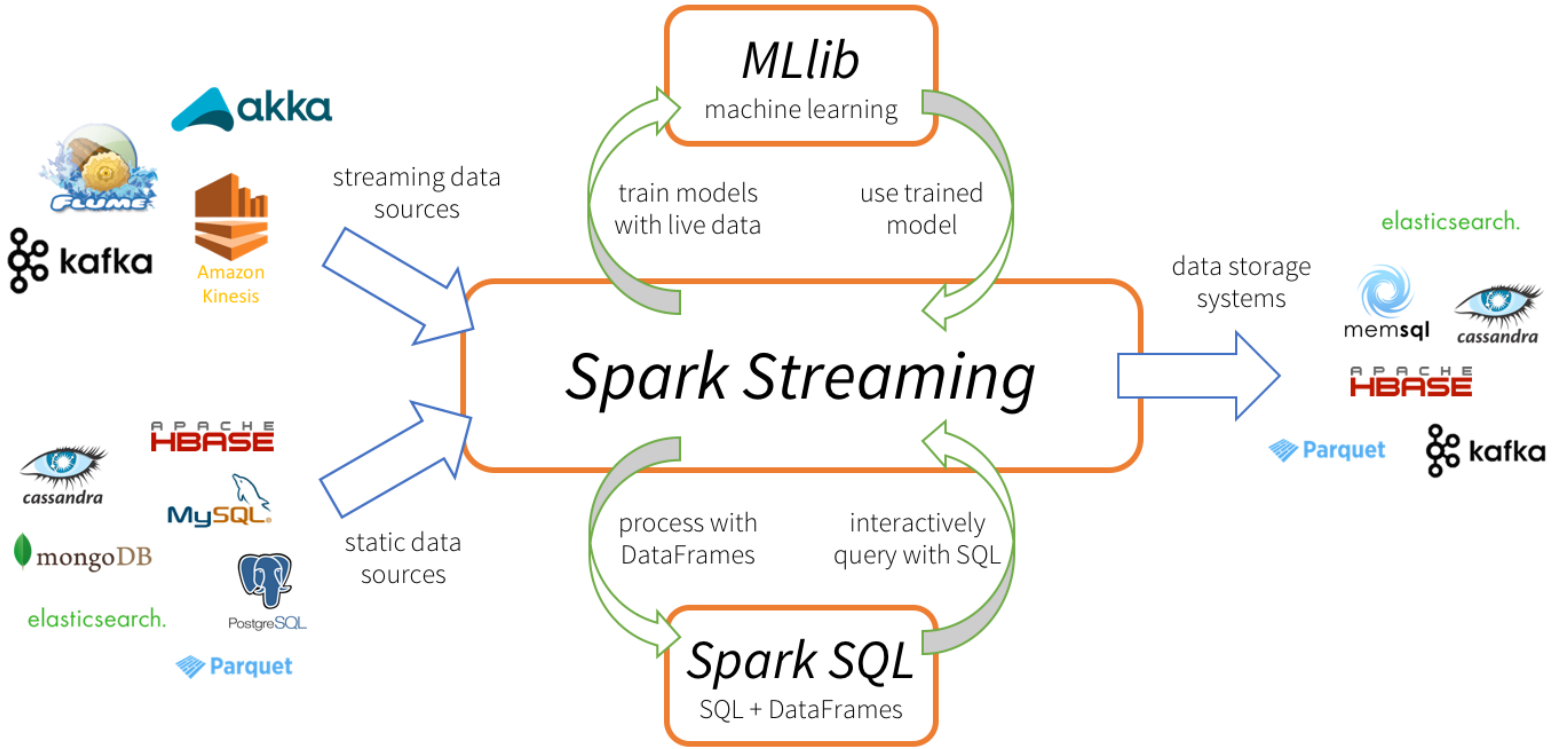


Late data handling in
Windowed Grouped Aggregation

<epam>

SOURCES & SINKS

Spark Streaming is a brick in the Big Data Wall



Let's save to Parquet files

Let's save to Parquet files

Let's save to Parquet files

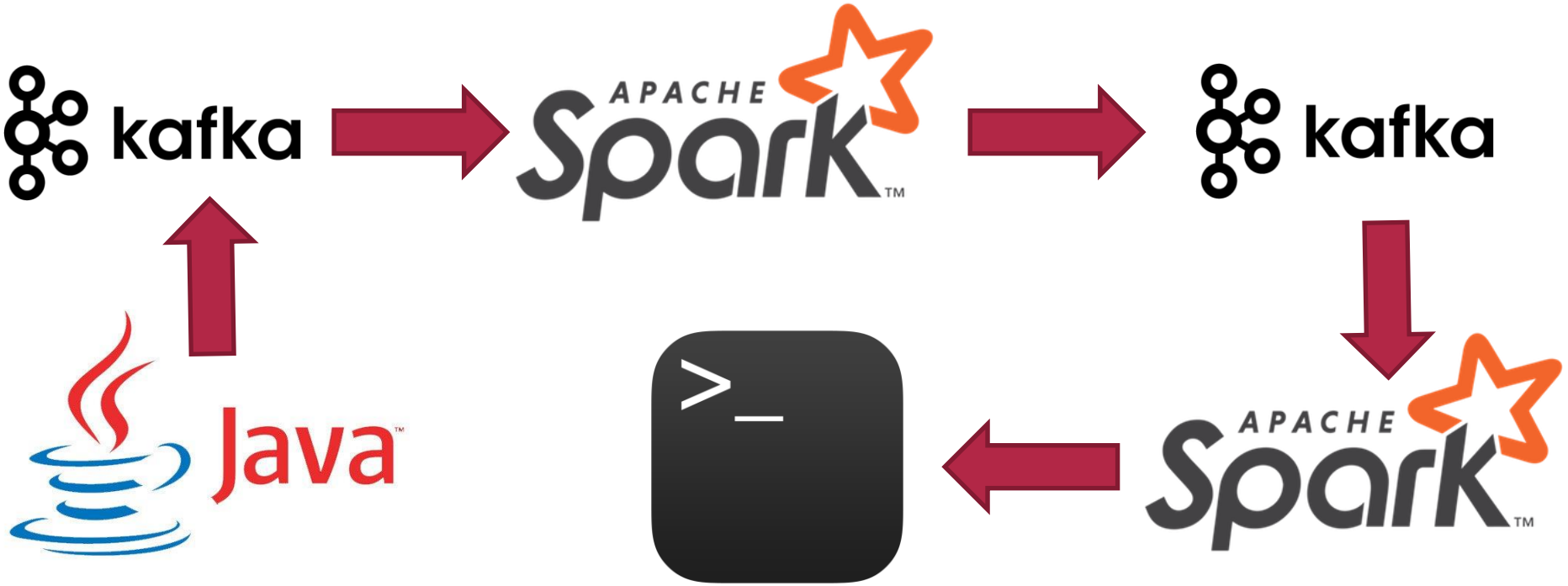
File to Memory


Can we write to Kafka?

Nightly Build

Kafka-to-Kafka

J-K-S-K-S-C



Pipeline Demo


I didn’t find sink/source for XXX…

Console Foreach Sink

```
import org.apache.spark.sql.ForeachWriter

val customWriter = new ForeachWriter[String] {
  override def open(partitionId: Long, version: Long) = true
  override def process(value: String) = println(value)
  override def close(errorOrNull: Throwable) = {}
}

stream.writeStream
  .queryName("ForeachOnConsole")
  .foreach(customWriter)
  .start
```

Pinch of wisdom

- check checkpointLocation
- don’t use MemoryStream
- think about GC pauses
- be careful about nighty builds
- use .groupBy.count() instead count()
- use console sink instead .show() function

We have no ability…

- join two streams
- work with update mode
- make full outer join
- take first N rows
- sort without pre-aggregation

Roadmap 2.2

- Support other data sources (not only S3 + HDFS)
- Transactional updates
- Dataset is one DSL for all operations
- GraphFrames + Structured MLLib
- KafkaWriter
- TensorFrames

<epam>

IN CONCLUSION

Final point

Scalable Fault-Tolerant **Real-Time** Pipeline with

Spark & Kafka

is ready for usage

A few papers about Spark Streaming and Kafka

Introduction in Spark + Kafka

http://bit.ly/2mJjE4i

Contacts

E-mail : Alexey_Zinovyev@epam.com

Twitter : @zaleslaw @BigDataRussia

Facebook: https://www.facebook.com/zaleslaw

vk.com/big_data_russia **Big Data Russia**

vk.com/java_jvm **Java & JVM langs**

Any questions?