Build a Time Series Application with Spark and HBase

Tugdual Grall
Technical Evangelist
@tgrall
tug@mapr.com
MapR

MapR Converged Data Platform

Processing

Open Source Engines & Tools

- hadoop
- Spark
- APACHE DRILL
- Search and Others

Commercial Engines & Applications

- Cloud and Managed Services
- VERTICA, SAP, MySQL
- Custom Apps

Data

HDFS API | POSIX, NFS | HBase API | JSON API | Kafka API

Web-Scale Storage
MapR-FS

Database
MapR-DB

Event Streaming
MapR Streams

High Availability | Real Time | Unified Security | Multi-tenancy | Disaster Recovery | Global Namespace

Enterprise-Grade Platform Services

Unified Management and Monitoring

Agenda

- Time Series
- Apache Spark & Spark Streaming
- Apache HBase
- Apache Kafka & MapR Streams
- Lab

About the Lab

- Use Spark & HBase in MapR Cluster
  - Option 1: Use a SandBox (Virtual Box VM located on USB Key)
  - Option 2: Use Cloud Instance (SSH/SCP only)

- Content:
  - Option 1: spark-streaming-hbase-workshop.zip on USB
  - Option 2: download zip from https://github.com/tgrall/mapr-streams-spark-hbase-workshop

Time Series

What is a Time Series?

- Stuff with timestamps
  - sensor measurements
  - system stats
  - log files
  - ….

Got Some Examples?


3587.52 -06.10 -33.83
B 3476.96 +03.76 +75.74
P 2753.08 -00.58 -03.61 S
0245.96 +04.79 +48.72
S 9937.21 +09.80 +37.90
-06.14 -80.78 P


MSC


THALYS
4305


Emirates
Kimi
at&t
SIEMENS
SAP
Mobil 1
Emirates
Emirates
NewCarWallpaper.co


ZOE

What do we need to do?

- Acquire
    - Measurement, transmission, reception
- Store
    - Individually, or grouped for some amount of time
- Retrieve
    - Ad hoc, flexible, correlate and aggregate
- Analyze and visualize
    - We facilitate this via retrieval

Acquisition

Not usually our problem

- Sensors
- Data collection – agents, raspberry pi
- Transmission – via LAN/Wan, Mobile Network, Satellites
- Receipt into system – listening daemon or queue, or depending on use case writing directly to the database

Storage Choice

- **Flat files**
    - Great for rapid ingest with massive data
    - Handles essentially any data type
    - Less good for data requiring frequent updates
    - Harder to find specific ranges
- **Traditional** RDBMS
    - Ingests up to ~10,000/ sec; prefers well structured (numerical) data; expensive
- **NoSQL** (such as MapR-DB or HBase)
    - Easily handle 10,000 rows / sec / node – True linear scaling
    - Handles wide variety of data
    - Good for frequent updates
    - Easily scanned in a range

Specific Example

Consider oil drilling rigs

- When drilling wells, there are *lots* of moving parts
- Typically a drilling rig makes about **10K samples/s**
- Temperatures, pressures, magnetics, machine vibration levels, salinity, voltage, currents, many others
- Typical project has 100 rigs

General Outline

10K samples / second / rig

x 100 rigs

= **1M samples / second**

- But wait, there’s more
  - Suppose you want to test your system
  - Perhaps with a year of data
  - And you want to load that data in << 1 year
- 100x real-time = 100M samples / second

Data Storage

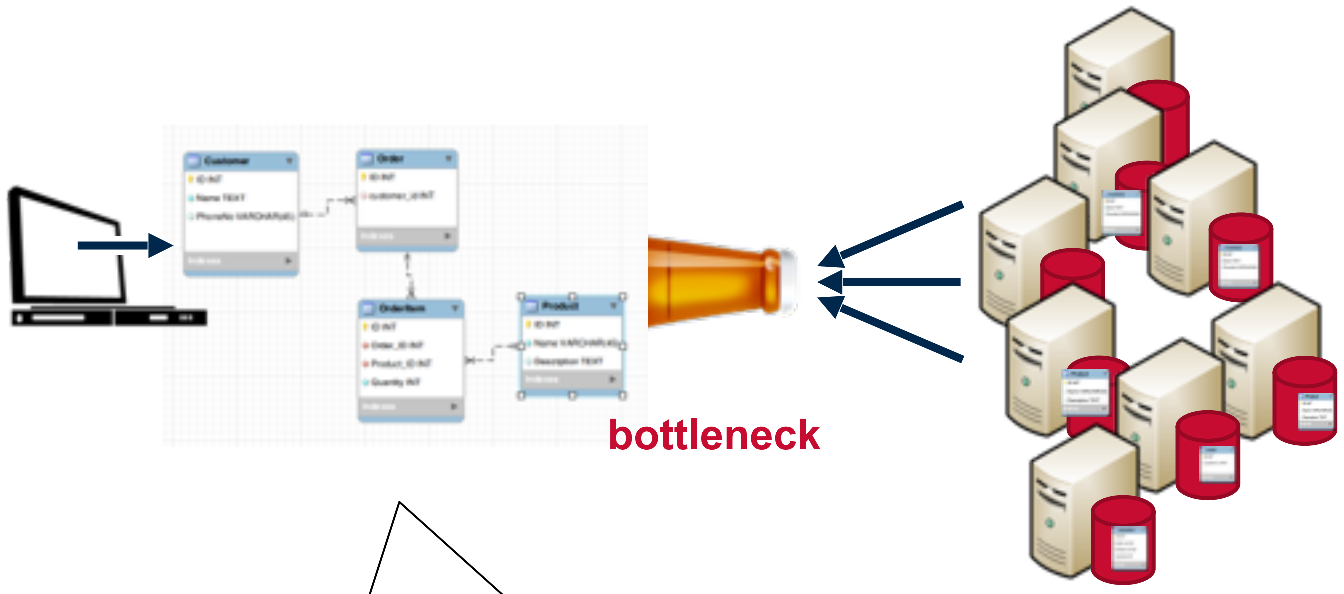
Key	13	43	73	103	…
…					
series-uid.time-window	*4.5*	*5.2*	*6.1*	*4.9*	
…					

- Typical time window is one hour
- Column names are offsets in time window
- Find series-uid in separate table

Why do we need NoSQL / HBase?
Relational Model

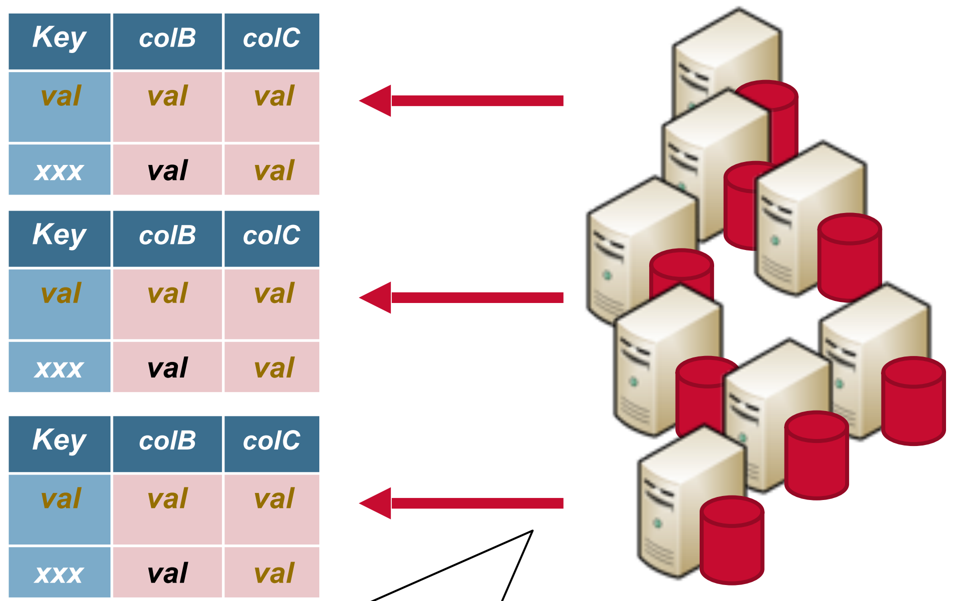
Storage Model

RDBMS



Distributed Joins, Transactions **do not scale**

HBase



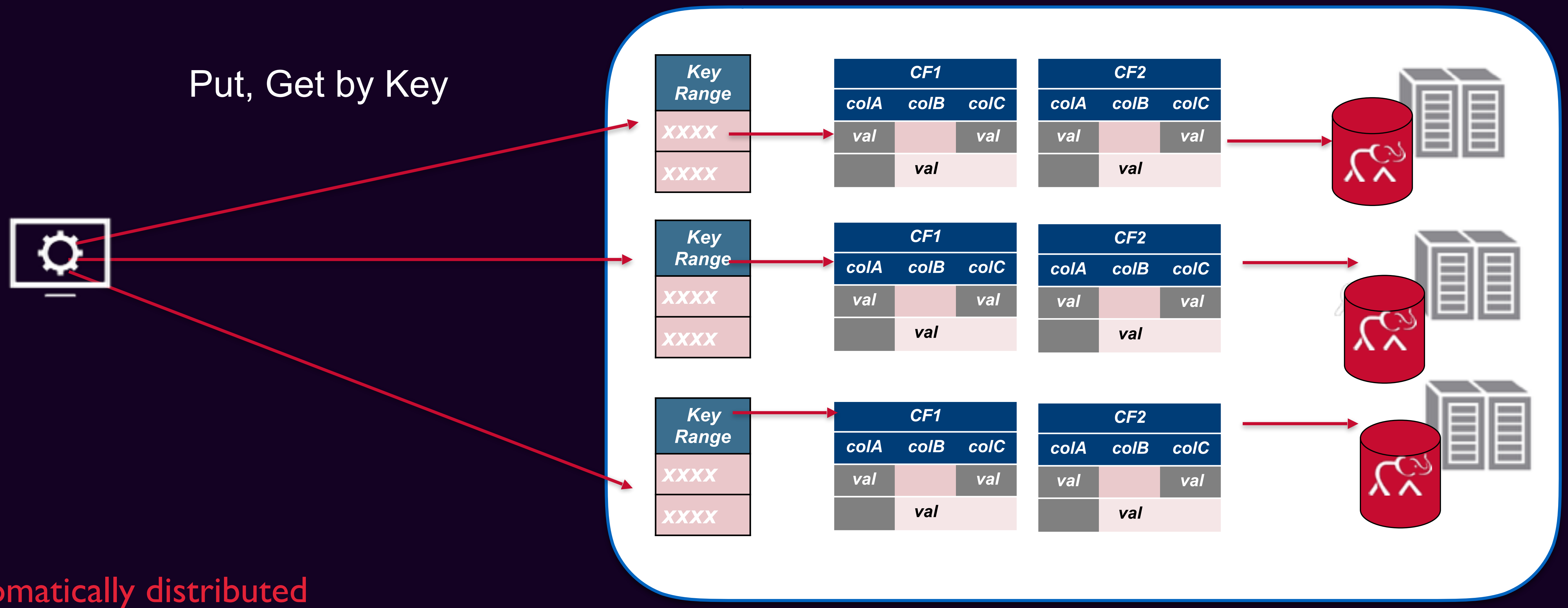
Data that is accessed together is stored together

HBase is a ColumnFamily oriented Database

Customer id	Raw Data			Stats		
	CF_DATA			CF_STATS		
RowKey	*colA*	*colB*	*colC*	*colA*	*colB*	*colC*
series-abc.time-window	*Val*		*val*	*val*		*val*
series-efg.time-window		*val*			*val*	

- Data is accessed and stored together:
  - RowKey is the primary index
  - Column Families group similar data by row key

HBase is a Distributed Database



Data is automatically distributed across the cluster

- **Key range** is used for **horizontal partitioning**

Basic Table Operations

- Create Table, define Column Families before data is imported
  - but not the rows keys or number/names of columns
- Low level API, technically more demanding
- Basic data access operations (CRUD):

put	Inserts data into rows (both create and update)
get	Accesses data from one row
scan	Accesses data from a range of rows
delete	Delete a row or a range of rows or columns

Learn More

- Free Online Training: http://learn.mapr.com
  - DEV 320 - Apache HBase Data Model and Architecture
  - DEV 325 - Apache HBase Schema Design
  - DEV 330 - Developing Apache HBase Applications: Basics
  - DEV 335 - Developing Apache HBase Applications: Advanced

Spark

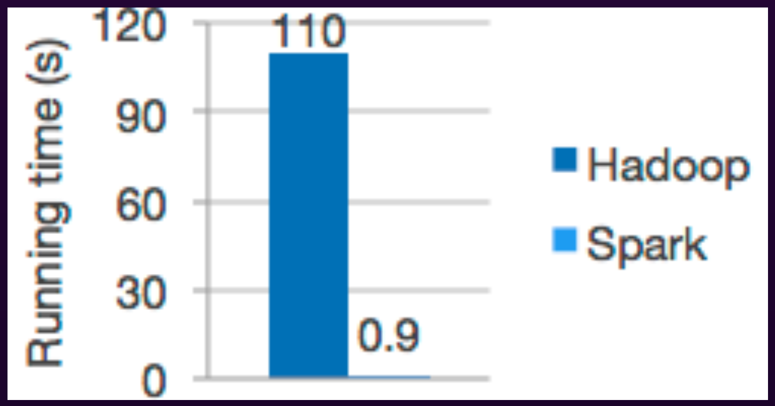
What is Spark?

- Cluster Computing Platform
- Extends “MapReduce” with extensions
    - Streaming
    - Interactive Analytics
- Run in Memory

What is Spark?

Fast

- 100x faster than M/R



Logistic regression in Hadoop and Spark

What is Spark?

Ease of Development

- Write programs quickly
- More Operators
- Interactive Shell
- Less Code

What is Spark?

Multi Language Support

- Scala
- Python
- Java
- SparkR

What is Spark?

Deployment Flexibility

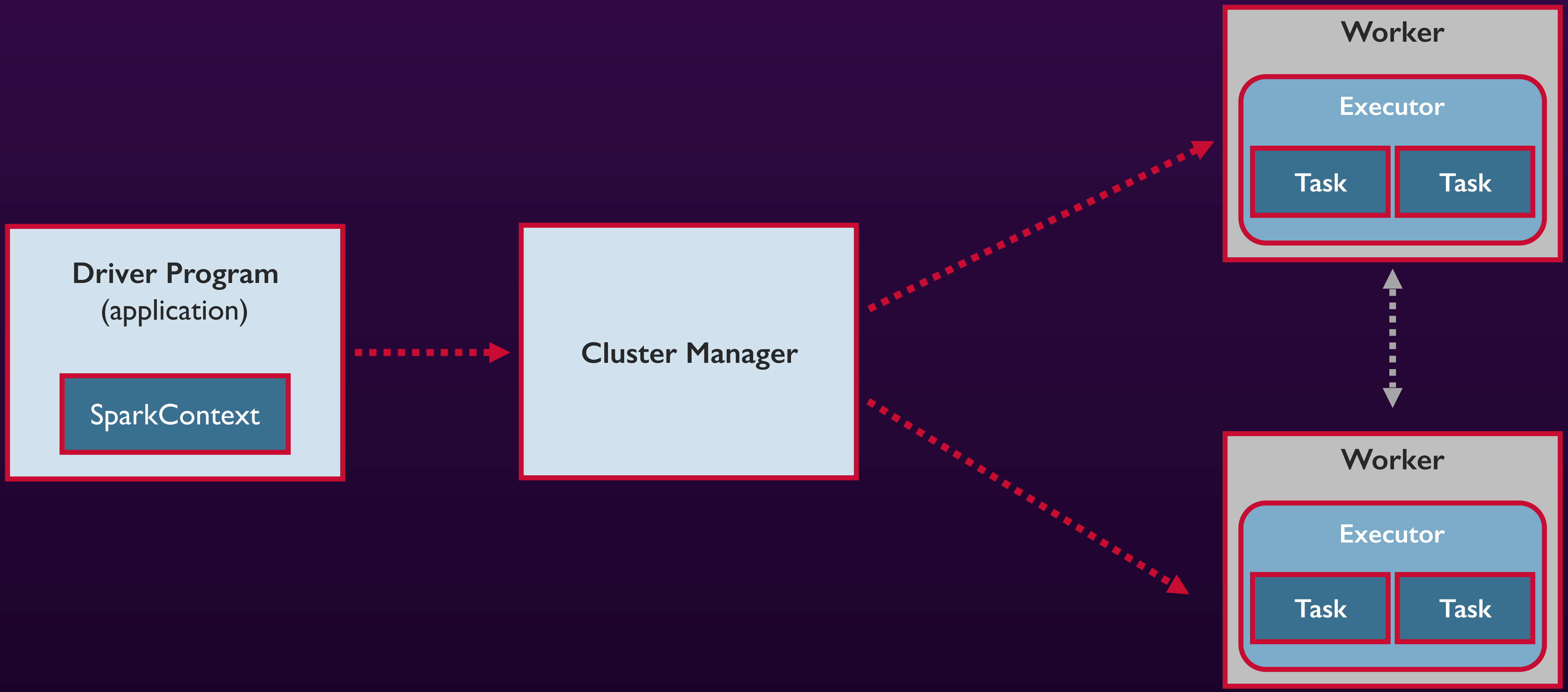
- Deployment
  - Local
  - Standalone
  - YARN
  - Mesos
- Storage
  - HDFS
  - MapR-FS
  - S3
  - Cassandra

Unified Platform

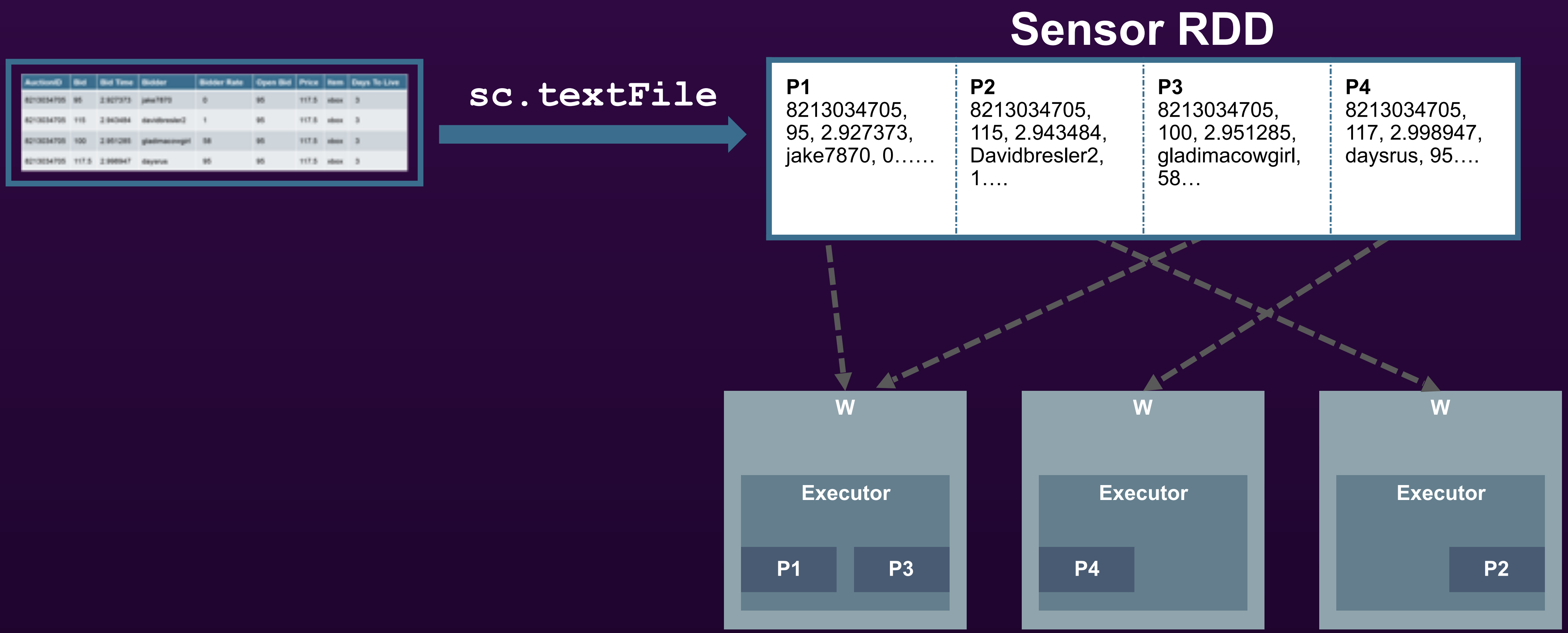
Spark *SQL*	**Spark Streaming** *(Streaming)*	**MLlib** *(Machine Learning)*	**GraphX** *(Graph Computation)*

Spark Core *(General execution engine)*

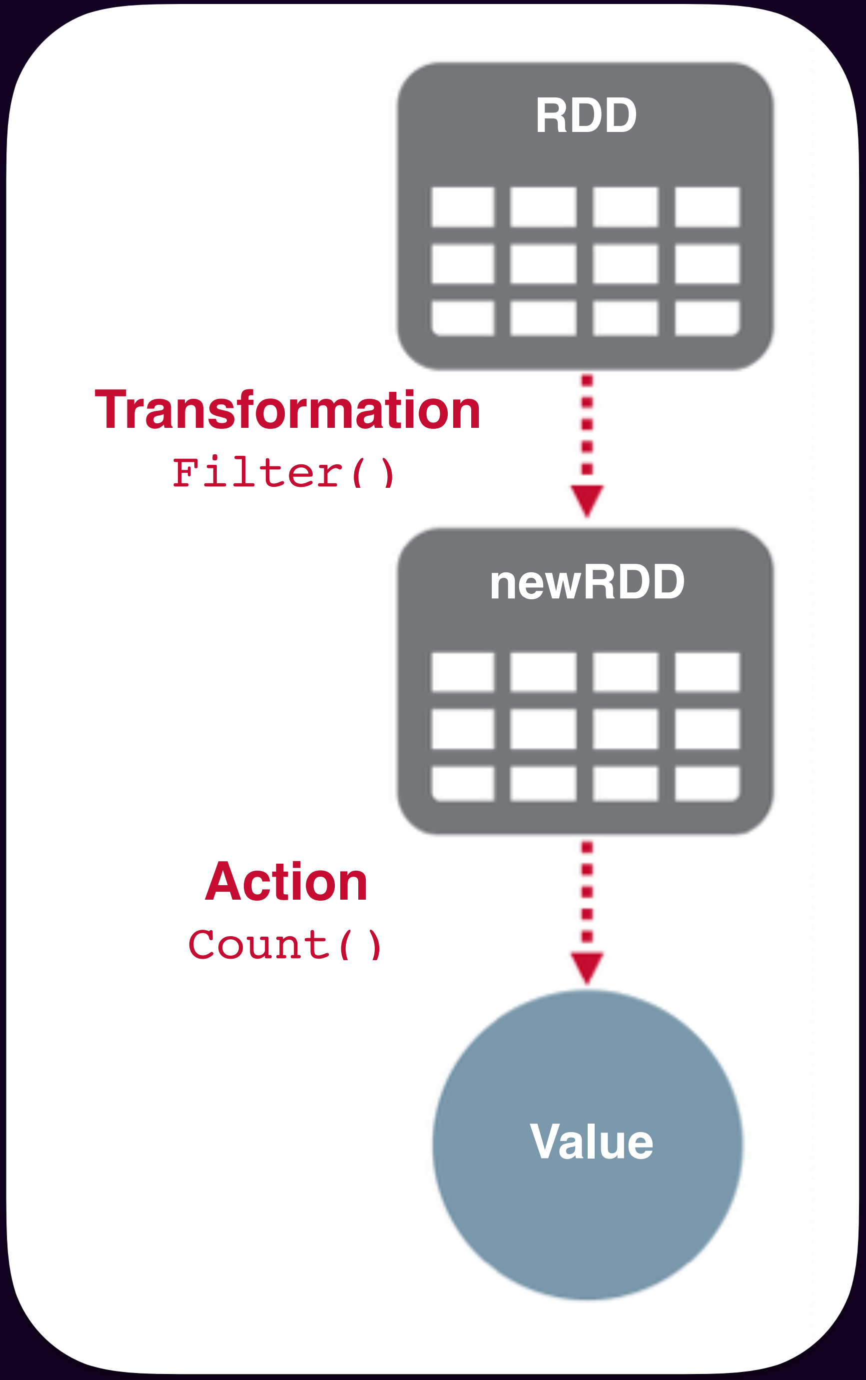
Spark Components



Spark Resilient Distributed Datasets



Spark Resilient Distributed Datasets



Spark Streaming

Spark *SQL*	Spark Streaming *(Streaming)*	MLlib *(Machine Learning)*	GraphX *(Graph Computation)*

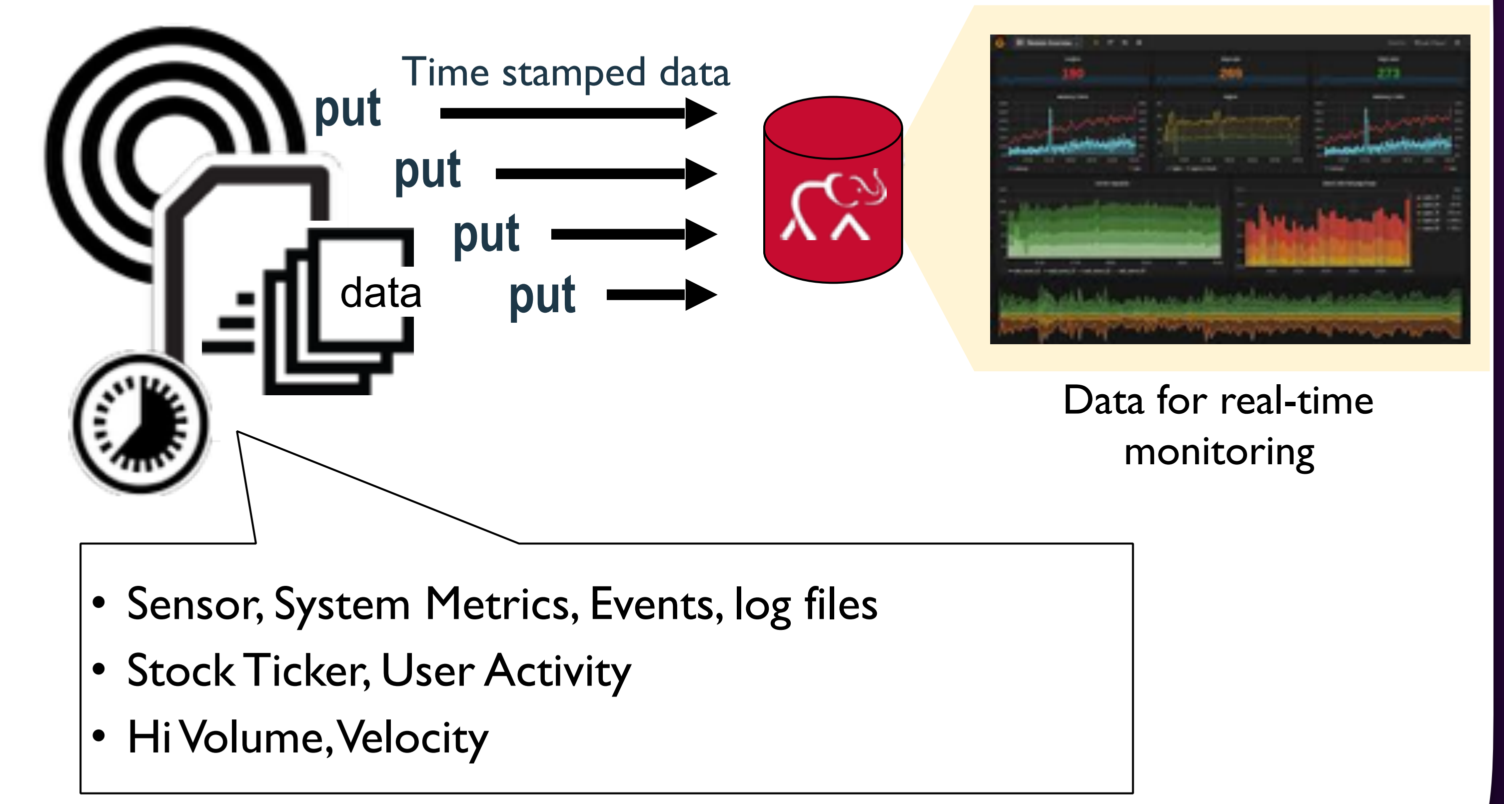
Spark Core *(General execution engine)*

What is Streaming?

- Data Stream:
  - Unbounded sequence of data arriving continuously
- Stream processing:
  - Low latency processing, querying, and analyzing of real time streaming data

Why Spark Streaming

- Many applications must process streaming data
- With the following Requirements:
    - Results in near-real-time
    - Handle large workloads
    - latencies of few seconds
- Use Cases
    - Website statistics, monitoring
    - IoT
    - Fraud detection
    - Social network trends
    - Advertising click monetization

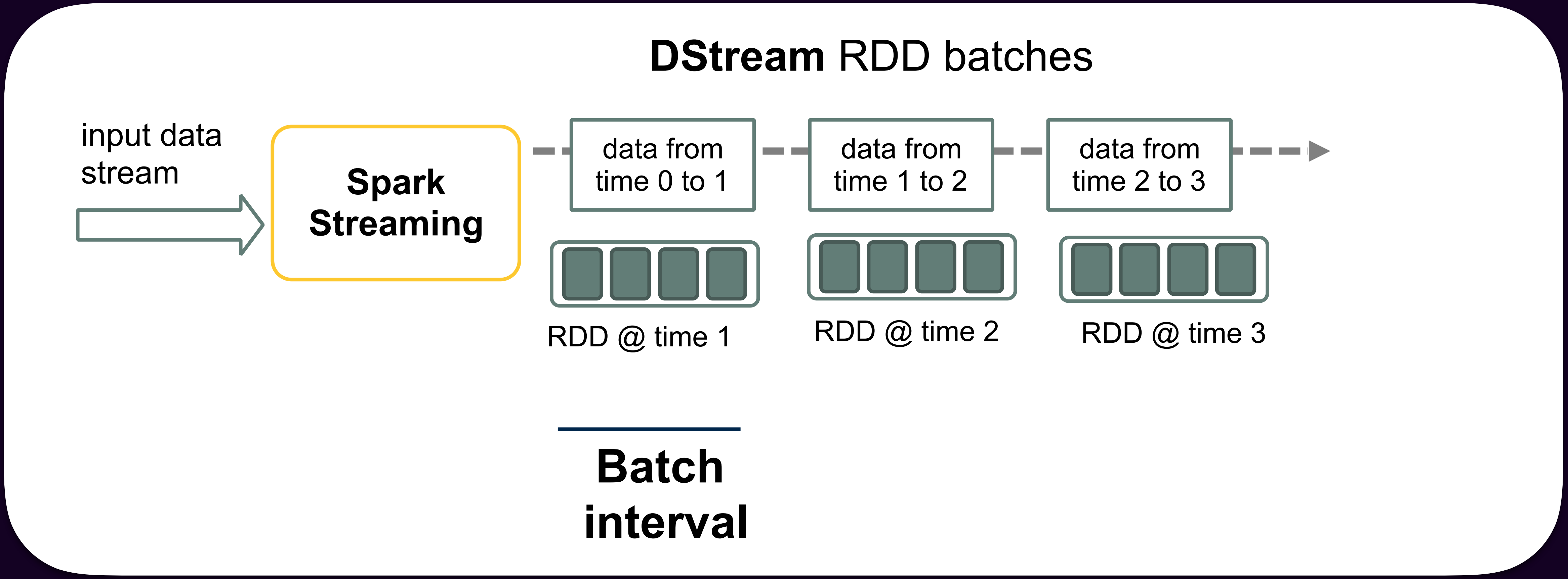


What is Spark Streaming?

- Enables scalable, high-throughput, fault-tolerant stream processing of live data
- Extension of the core Spark

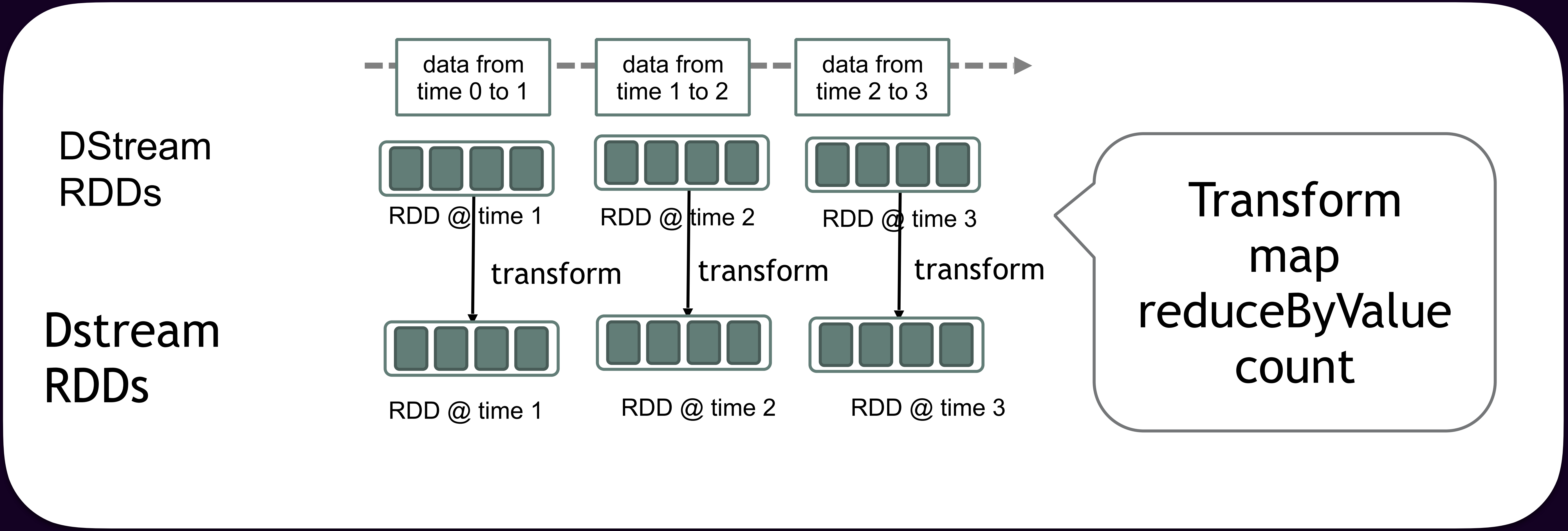
Spark Streaming Architecture

- Divide data stream into batches of X seconds
    - Called DStream = sequence of RDDs



Process DStream

- Process using transformations
  - creates new RDDs



kafka

What is Kafka?

- http://kafka.apache.org/
- Created at LinkedIn, open sourced in 2011
- Implemented in Scala / Java
- Distributed messaging system built to scale

What for?

- Message Queue (!= ESB)
- Realtime Streaming
- Event Sourcing
- Logs
- Change Data Capture

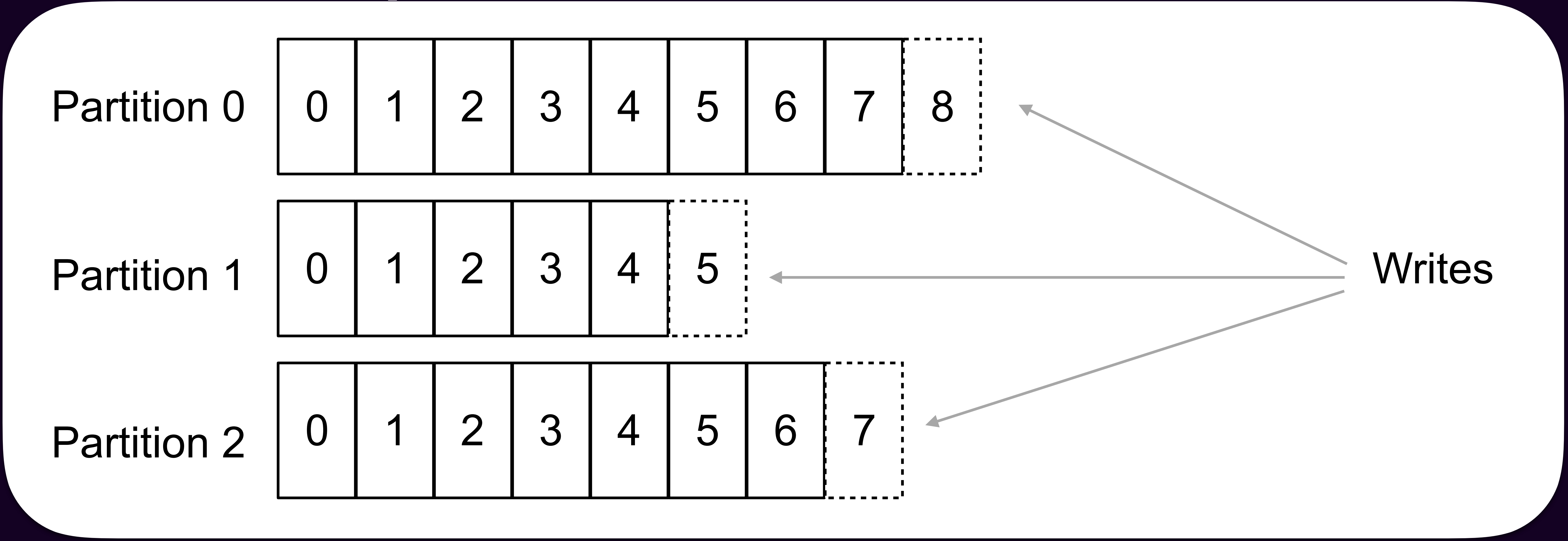
Key Concepts

- Feeds of messages are organised in **<u>topics</u>**
- Processes that publish messages are called **<u>producers</u>**
- Processes that subscribed to topic and process messages are **<u>consumers</u>**
- A Kafka cluster is made of one or more **<u>brokers</u>** (== node)

Key Features

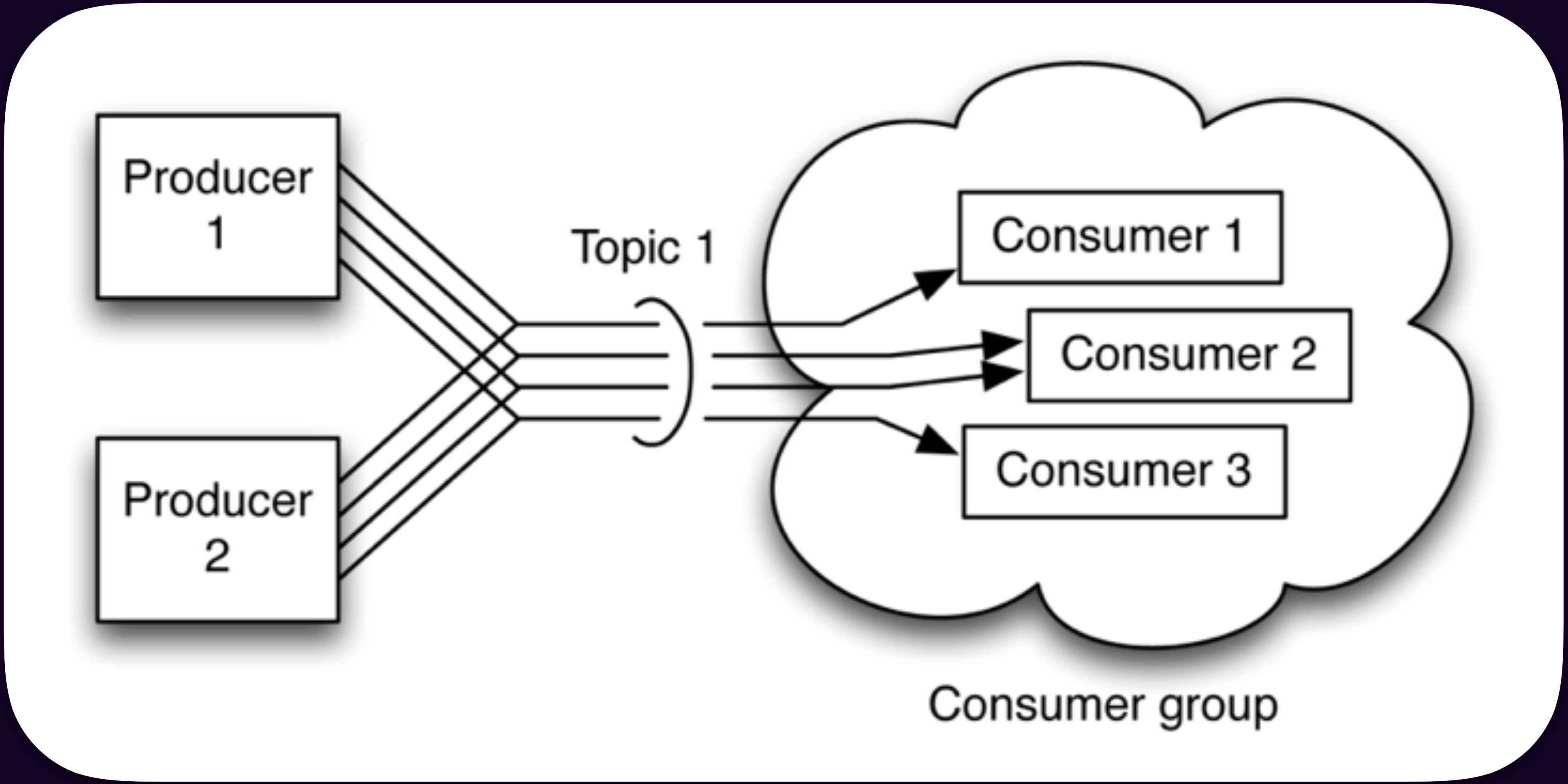
- Durable
- Scalable
  - Distributed
  - Stateless
- Fast
- At least once or at most once
  - You need to deal with it!

Topics and Partitions



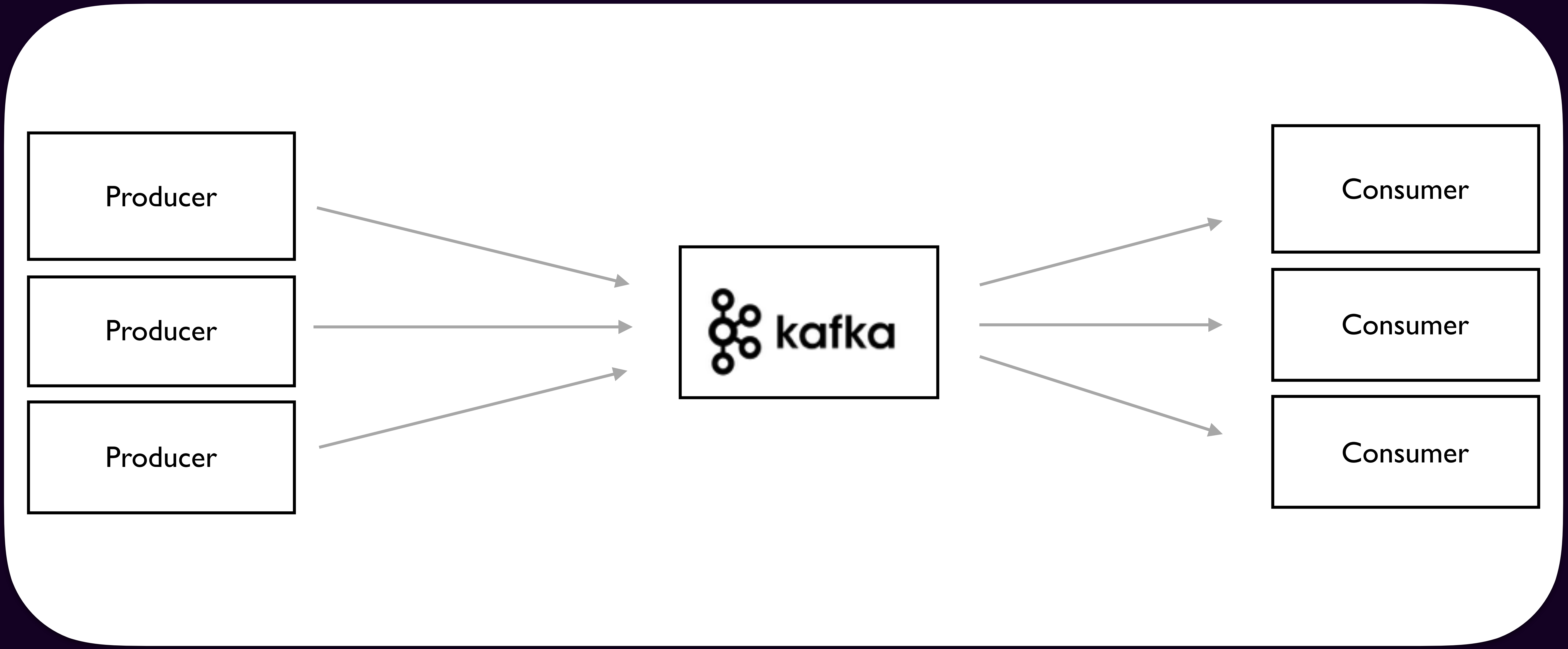
Split topics into partitions for scalability

Consumer Groups



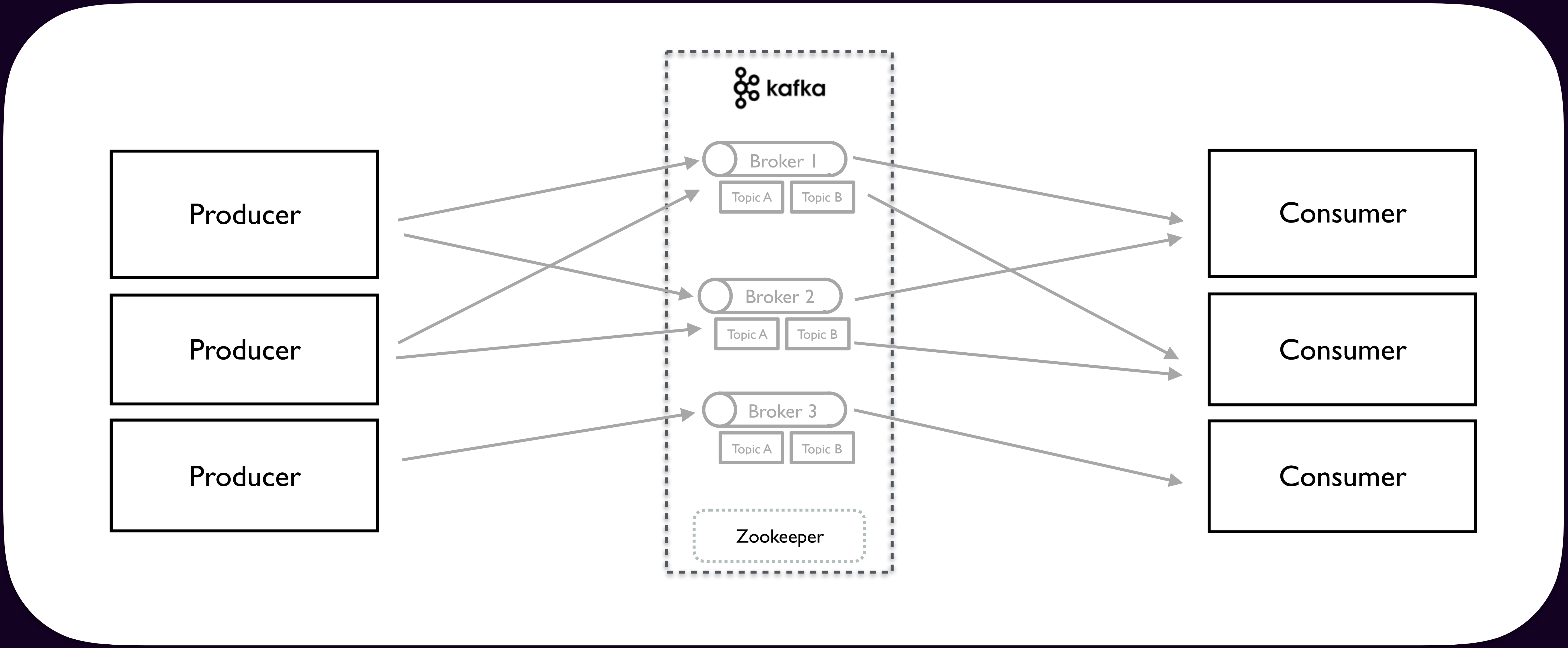
- Single consumer abstraction for scalability
- Max 1 consumer per partition
- Any number of consumer groups

Big Picture



Producer
Producer
Producer
kafka
Consumer
Consumer
Consumer

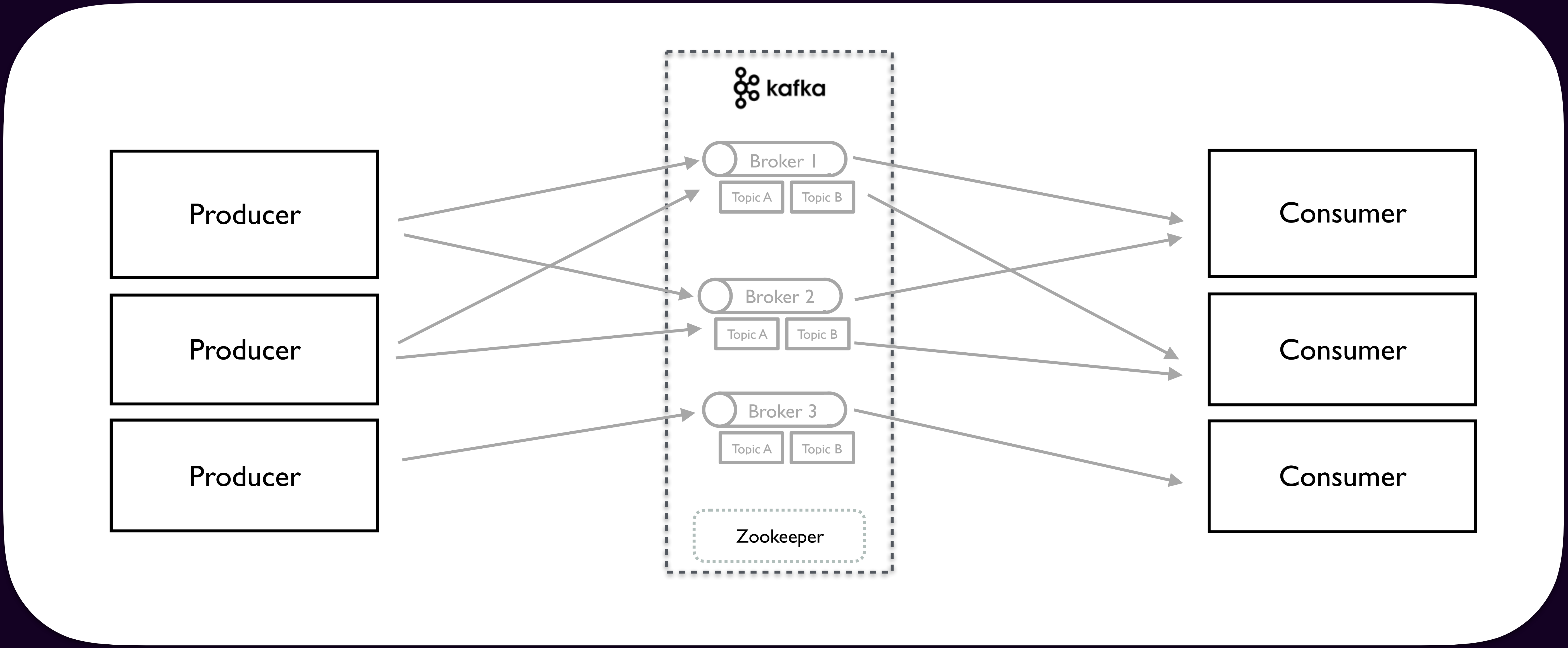
More real life Kafka …



MapR Streams

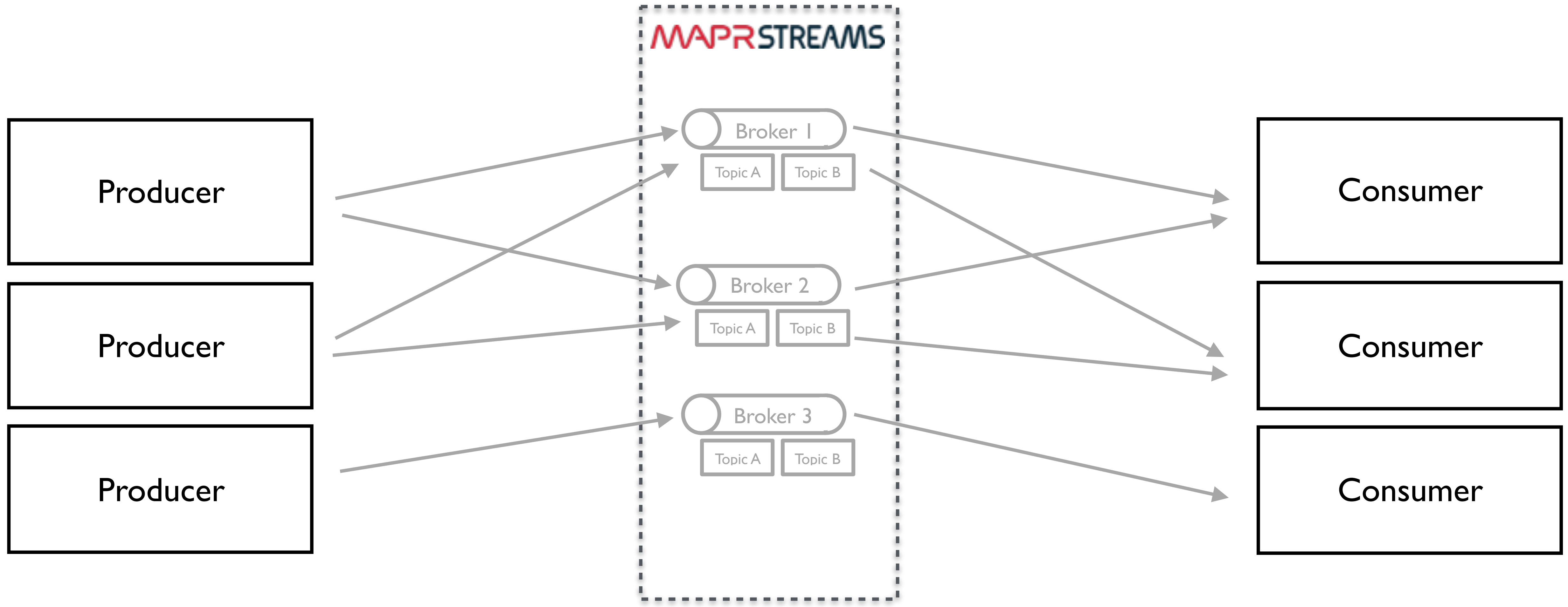
- Distributed messaging system built to scale
- Use Apache Kafka API 0.9.0
  - No code change
- Does not use the same “broker” architecture
  - Log stored in MapR Storage *(Scalable, Secured, Fast, Multi DC)*
  - No Zookeeper

Kafka

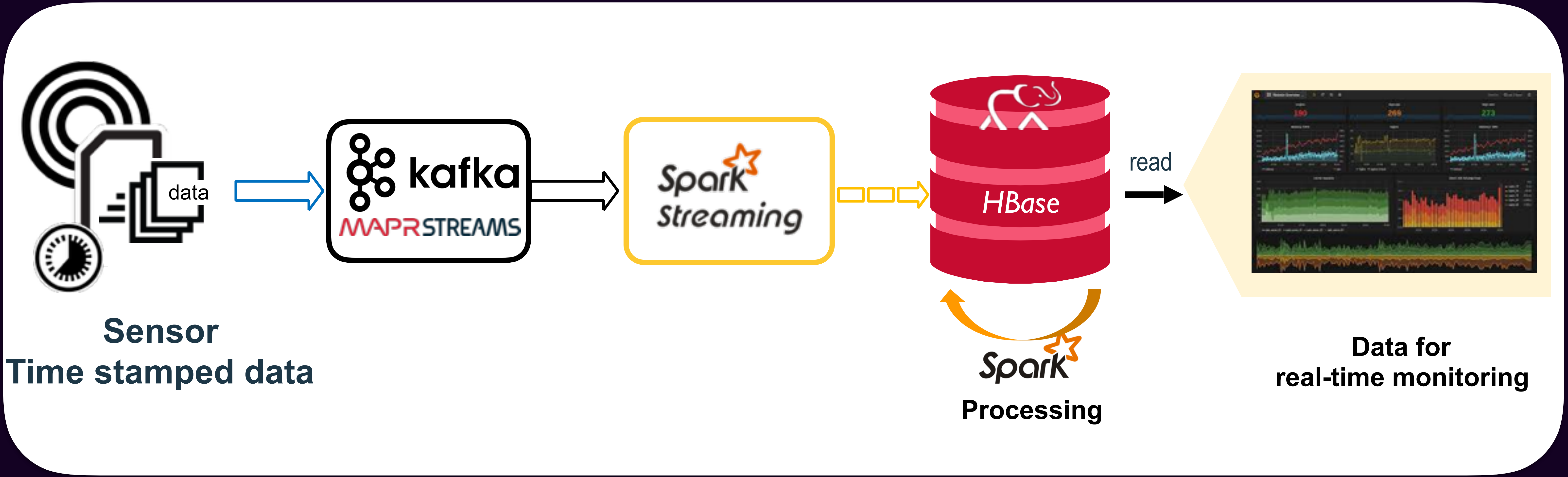


kafka
Producer
Producer
Producer
Broker 1
Topic A
Topic B
Broker 2
Topic A
Topic B
Broker 3
Topic A
Topic B
Zookeeper
Consumer
Consumer
Consumer

MapR Streams



Time Series


data
kafka
MAPR STREAMS
Spark
Streaming
HBase
read
Spark
Sensor
Time stamped data
Processing
Data for
real-time monitoring

Lab “flow”

Convert Line of CSV data to Sensor Object

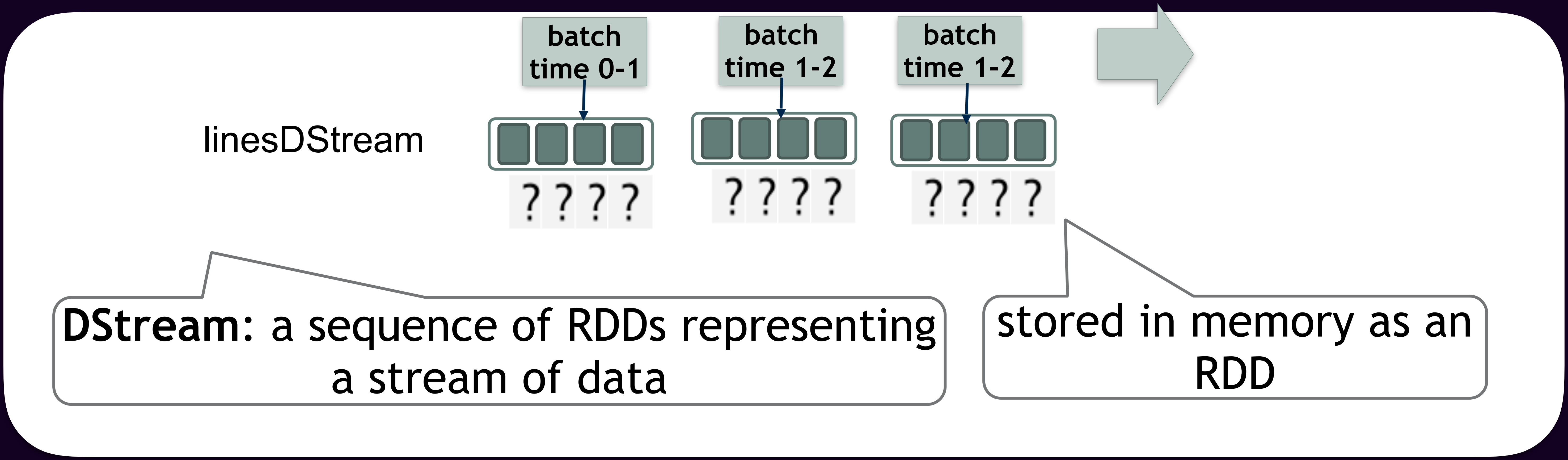
sensordata.csv

```
COHUTTA,3/10/14,1:01,10.27,1.73,881,1.56,85,1.94
COHUTTA,3/10/14,1:02,9.67,1.731,882,0.52,87,1.79
COHUTTA,3/10/14,1:03,10.47,1.732,882,1.7,92,0.66
COHUTTA,3/10/14,1:05,9.56,1.734,883,1.35,99,0.68
COHUTTA,3/10/14,1:06,9.74,1.736,884,1.27,92,0.73
COHUTTA,3/10/14,1:08,10.44,1.737,885,1.34,93,1.54
COHUTTA,3/10/14,1:09,9.83,1.738,885,0.06,76,1.44
COHUTTA,3/10/14,1:11,10.49,1.739,886,1.51,81,1.83
COHUTTA,3/10/14,1:12,9.79,1.739,886,1.74,82,1.91
COHUTTA,3/10/14,1:13,10.02,1.739,886,1.24,86,1.79
```

```
case class Sensor(resid: String, date: String, time: String,
   hz: Double, disp: Double, flo: Double, sedPPM: Double,
   psi: Double, chlPPM: Double)

def parseSensor(str: String): Sensor = {
   val p = str.split(",")
   Sensor(p(0), p(1), p(2), p(3).toDouble, p(4).toDouble, p(5).toDouble,
       p(6).toDouble, p(7).toDouble, p(8).toDouble)
}
```

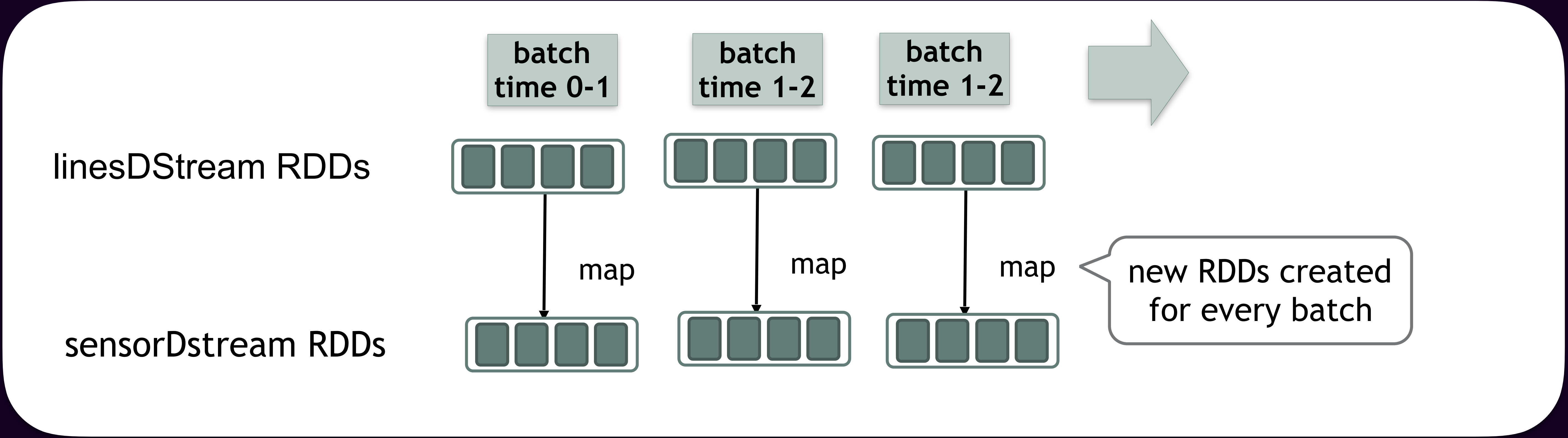
Create a DStream

```
val ssc = new StreamingContext(sparkConf, Seconds(2))
val messages = KafkaUtils.createDirectStream[String, String]
                                    (ssc, kafkaParams, topicsSet)
```



Process DStream

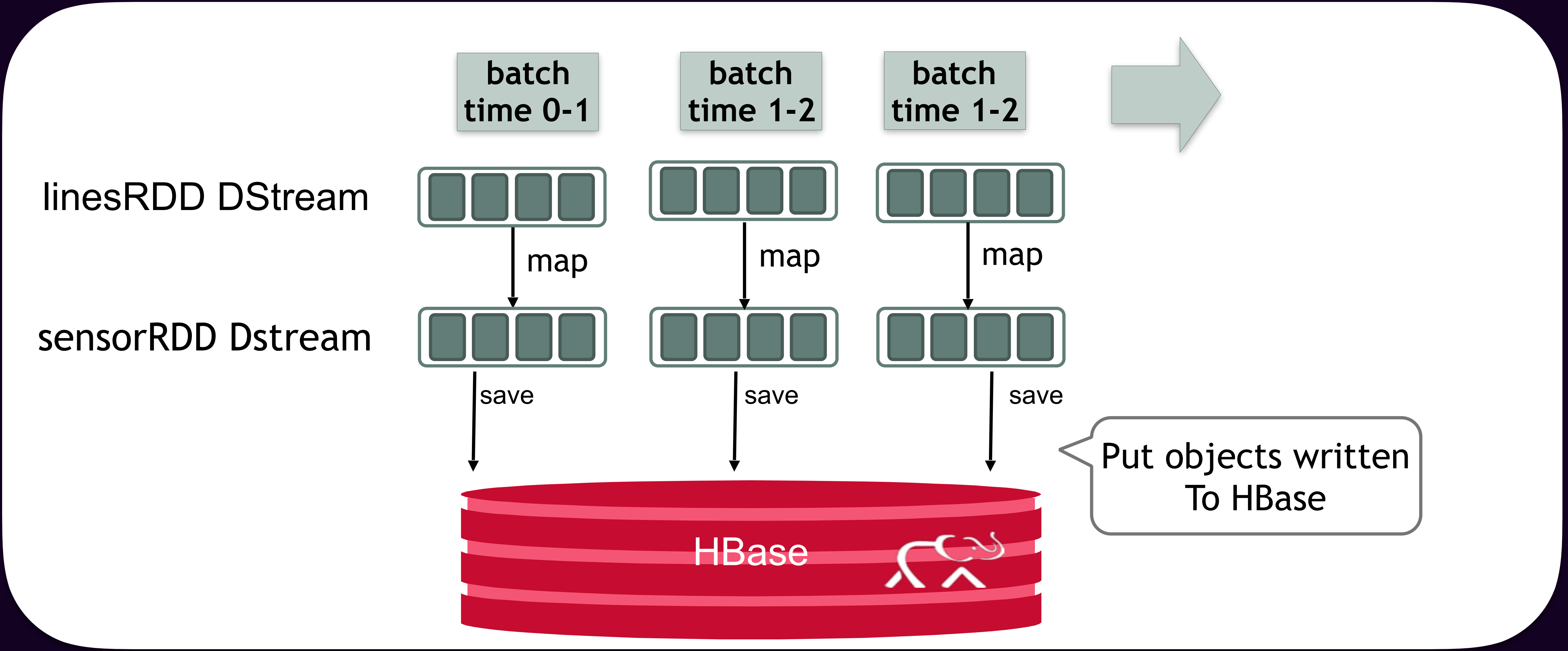
```
val messages = KafkaUtils.createDirectStream[String, String]
                                    (ssc, kafkaParams, topicsSet)

val sensorDStream = messages.map(_._2).map(Sensor.parseSensor)
```



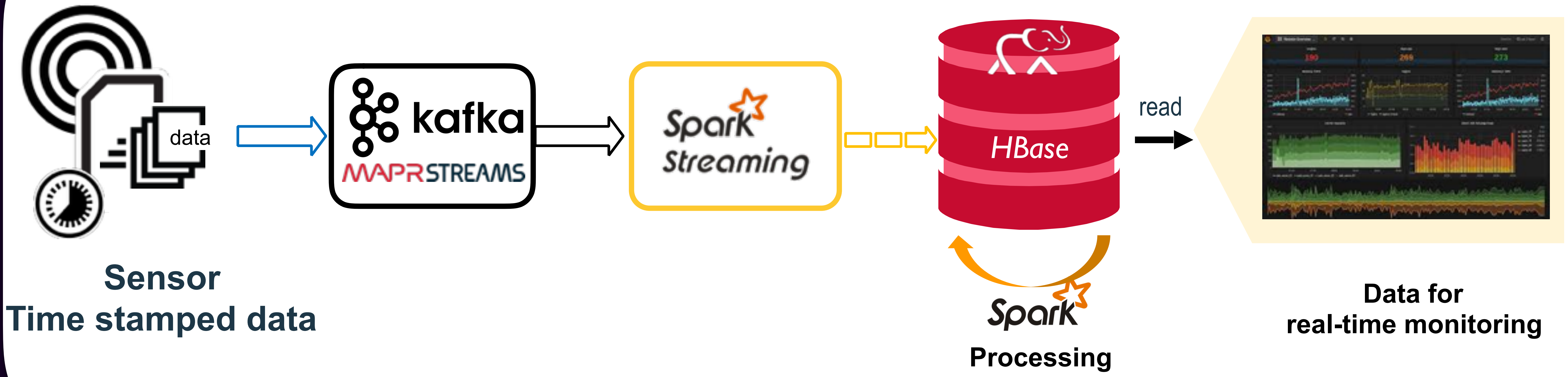
Save to HBase

```
rdd.map(Sensor.convertToPut).saveAsHadoopDataset(jobConfig)
```

output operation: persist data to external storage



Time Series



data
kafka
MAPR STREAMS
Spark Streaming
HBase
read
Spark
Sensor
Time stamped data
Processing
Data for
real-time monitoring

Go !