BIGDATA TEAM

Megafon

# Real-Time Big Data

**Vybornov Artyom**, avybornov@bigdatateam.org
Big Data Instructor, http://bigdatateam.org/
Head of Big Data Dev Team, Rambler Group
https://www.linkedin.com/in/artvybor/

2019-08-12, Moscow

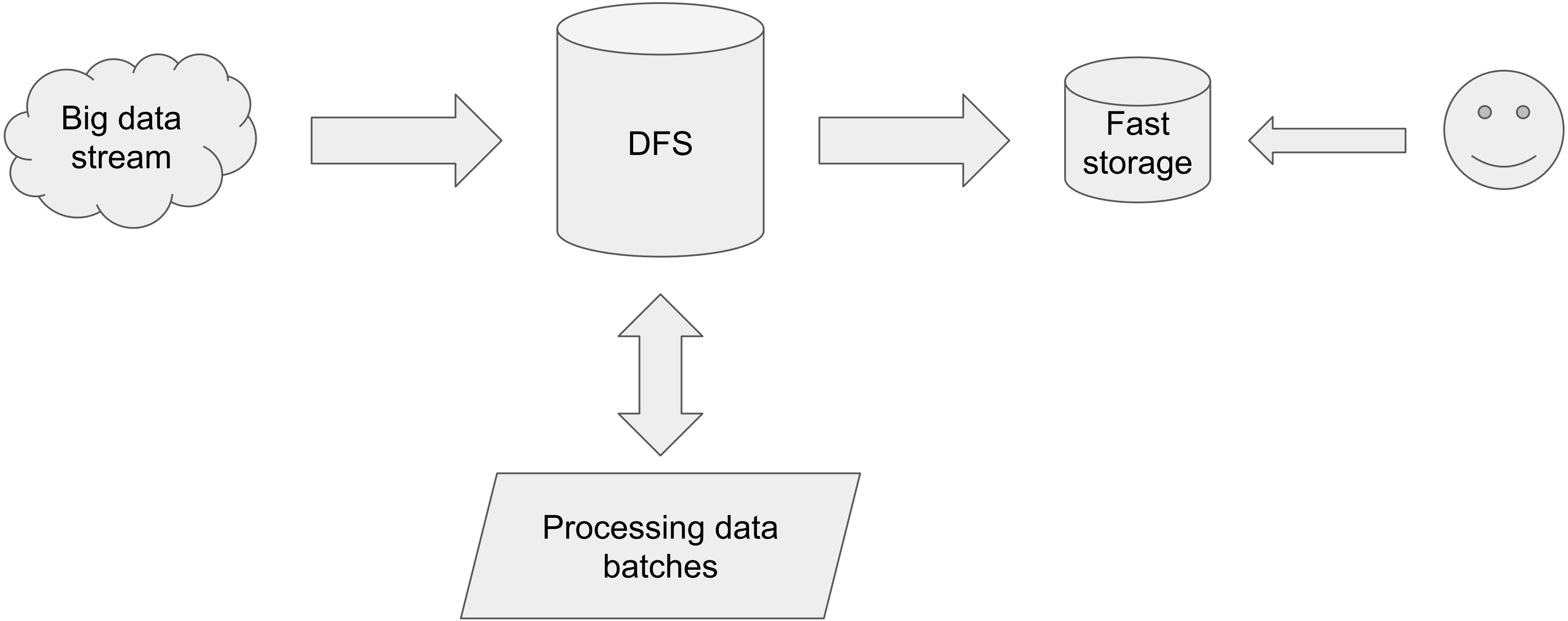
# Real-Time Big Data

- RT Intro
  - Batch -> RT
  - Approaches to RT data processing
- Spark Streaming
  - Application example
  - Hints
- Spark Streaming practice
- Kafka
  - Internal
  - CLI

# Real-Time Big Data

- RT Intro
  - **Batch -> RT**
  - Approaches to RT data processing
- Spark Streaming
  - Application example
  - Hints
- Spark Streaming practice
- Kafka
  - Internal
  - CLI

# Batch approach

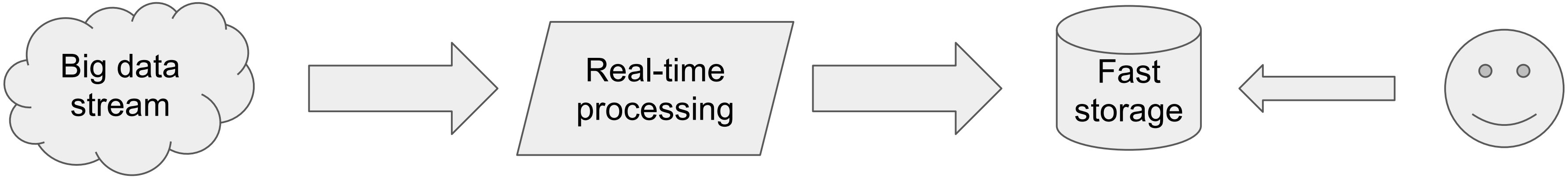


# The main cons of batch approach

- In practice a batch is a big time interval like an hour or day
- The size of batch is a minimal value for a lag
- Lag (delay) is the time between the event and its recording in the results of work
- For many purposes the following rule works: less lag => more valuable data

# Real-time big data

- Real-time big data - is a set of technologies to process big data with minimal lag
- Without DFS
- Work with a stream of data instead of a batch



# Real-time big data - lag in minutes

- Build custom recommendations (Linkedin, Facebook)
  - Billions of events per day
  - Millions of events every second
  - Minutes to build fresh recommendation

# Real-time big data - lag in seconds

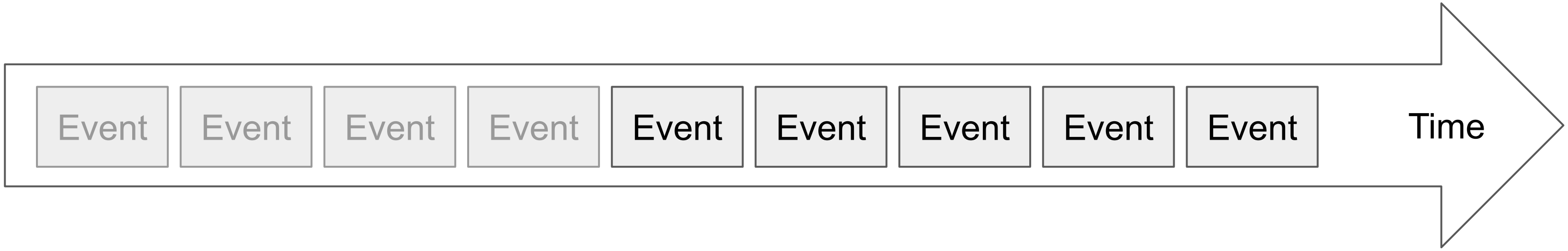
- Programmatic advertising (Google, Facebook)
  - Billions of events per day
  - You liked the article and saw the relevant ads already on the next page
- Credit card real-time fraud detection
  - It takes a few seconds to detect a card with suspicious activity and block it

# Real-time big data - lag in ms

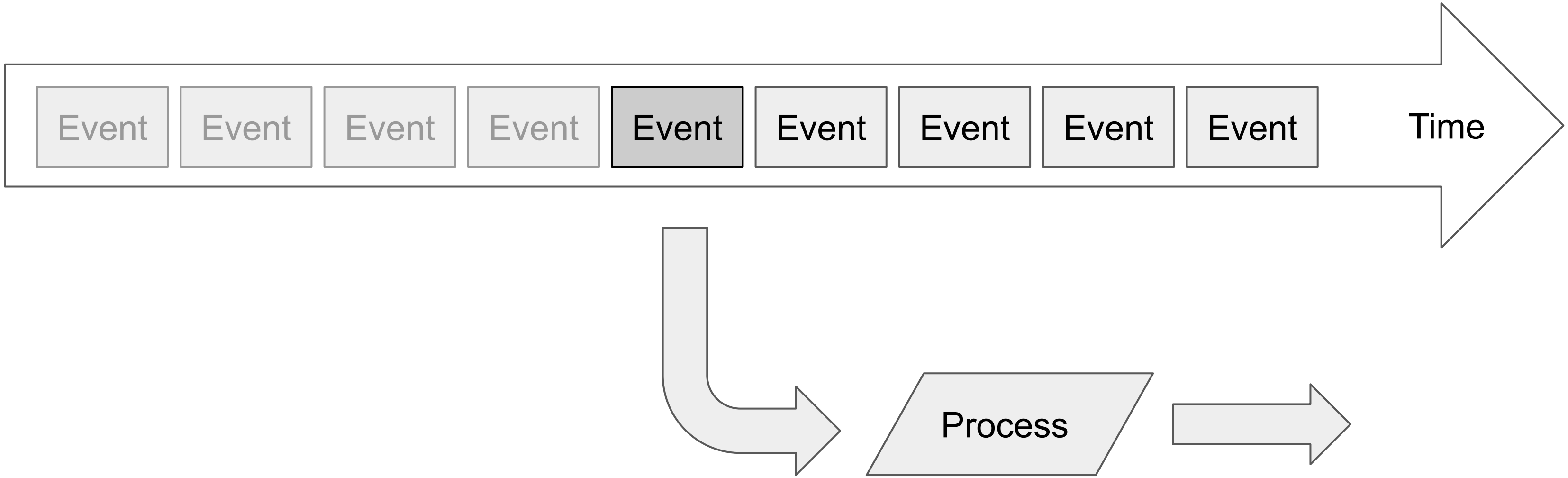
- Calls billing (mobile network operator)
  - Hundred millions of users
  - Less than a second to debit funds
  - Zero fault tolerance
- High-frequency trading (HFT)
  - The reaction to the change of quotations of the exchange with a delay of 10-100 milliseconds

- RT Intro
  - Batch -> RT
  - **Approaches to RT data processing**
- Spark Streaming
  - Application example
  - Hints
- Spark Streaming practice
- Kafka
  - Internal
  - CLI

# Event-based approach

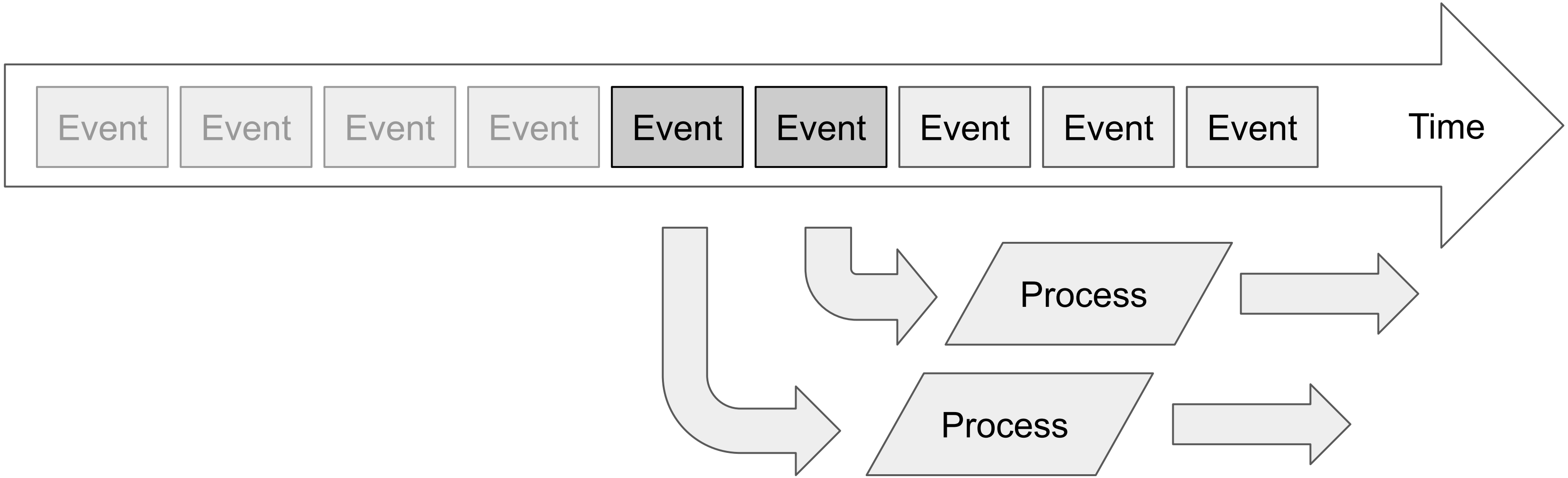


# Event-based approach



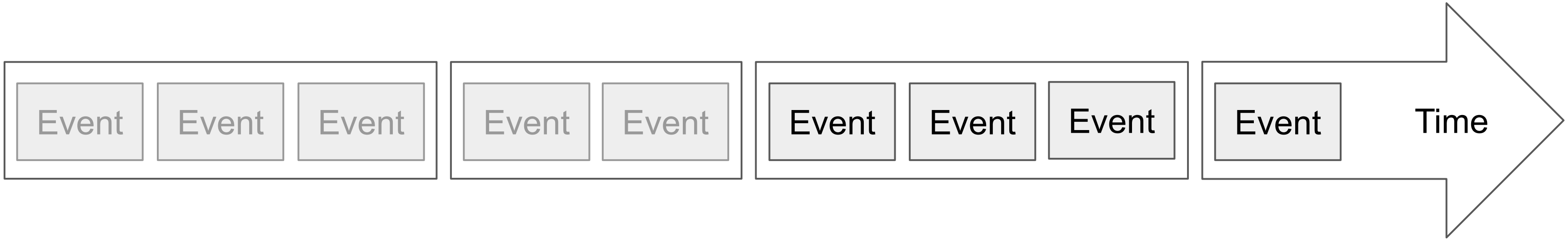

▶️ Handling one event at a time

# Event-based approach



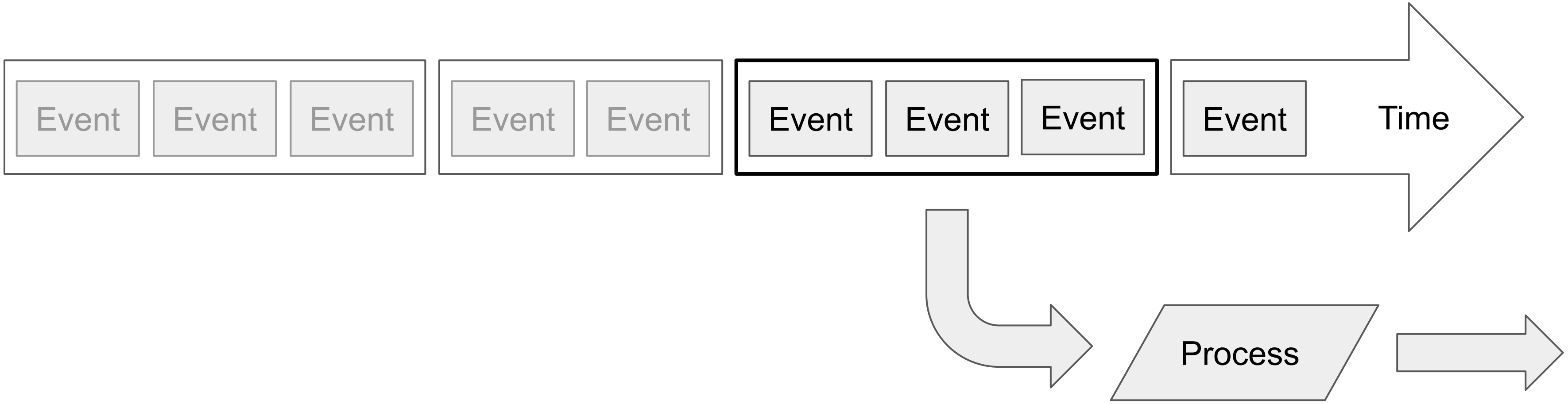

- Handling one event at a time
- Events are processed in parallel, but completely independently
- Latency ~10ms

# Micro-batch approach



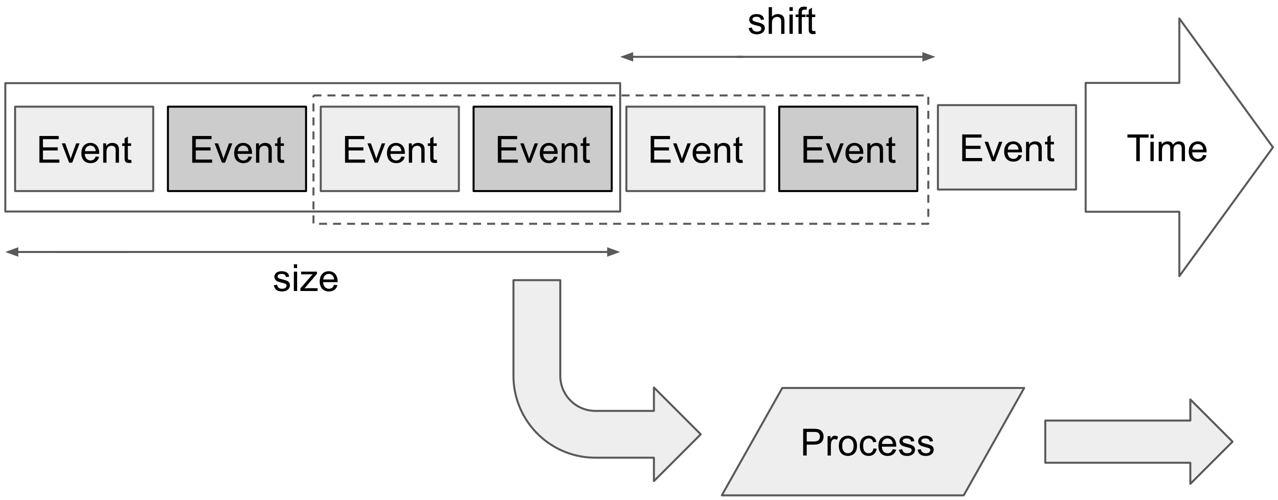

- Stream is cut to batch by time (for example 10 seconds batch)

# Micro-batch approach




- Stream is cut to batch by time (for example 10 seconds batch)
- Batches are processed sequentially
- Latency >> 1s

# Windowed approach




- Batch as a sliding window
- Similar to micro-batch approach

# Event-based vs micro-batch

- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources
  by reducing the common parts
  of each event handling/processing

# Event-based vs micro-batch

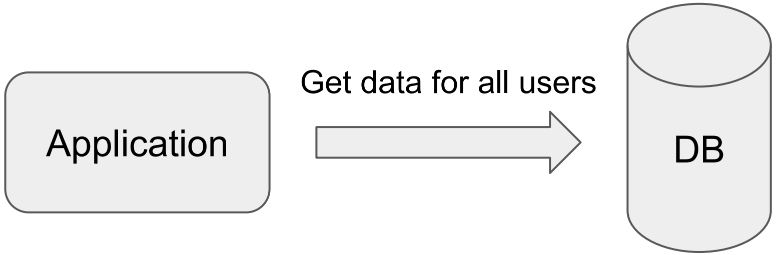
- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources
  by reducing the common parts
  of each event handling/processing

# Event-based vs micro-batch

- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources by reducing the common parts of each event handling/processing

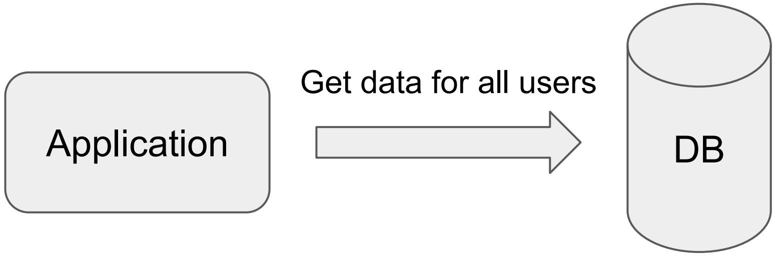
# Event-based vs micro-batch

- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources by reducing the common parts of each event handling/processing

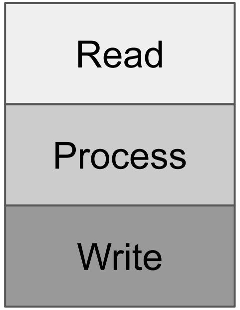
# Event-based vs micro-batch

- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources by reducing the common parts of each event handling/processing

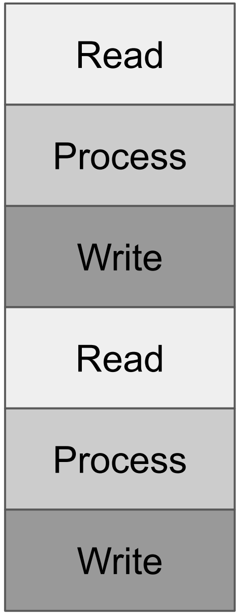
# Event-based vs micro-batch

- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources by reducing the common parts of each event handling/processing



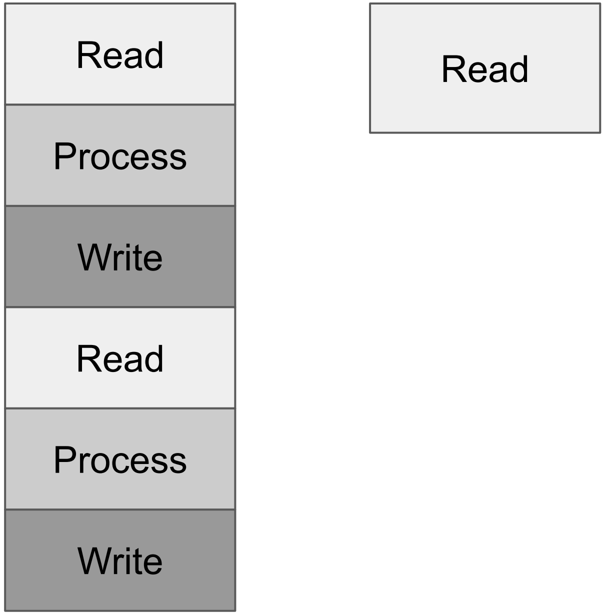
# Event-based vs micro-batch

- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources by reducing the common parts of each event handling/processing



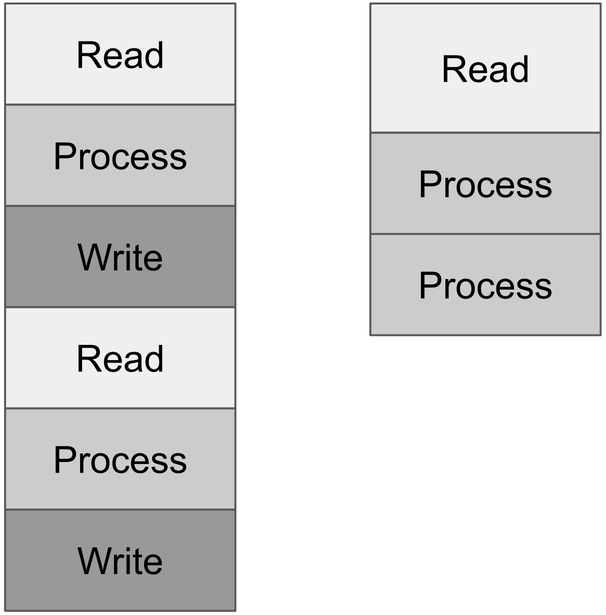
# Event-based vs micro-batch

- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources by reducing the common parts of each event handling/processing



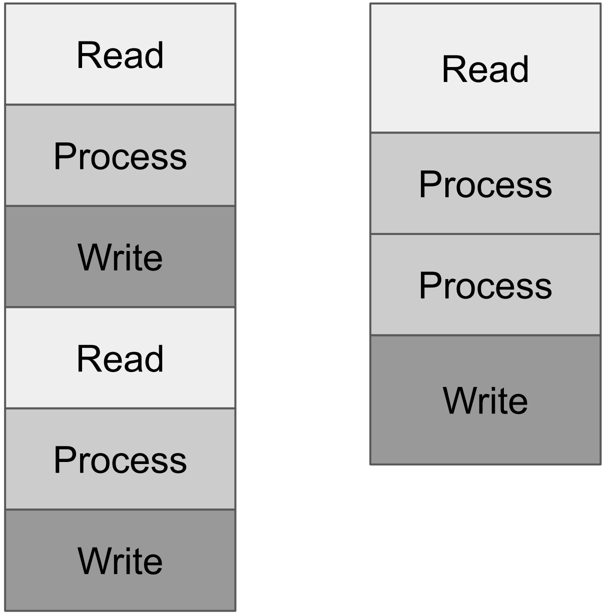
# Event-based vs micro-batch

- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources by reducing the common parts of each event handling/processing



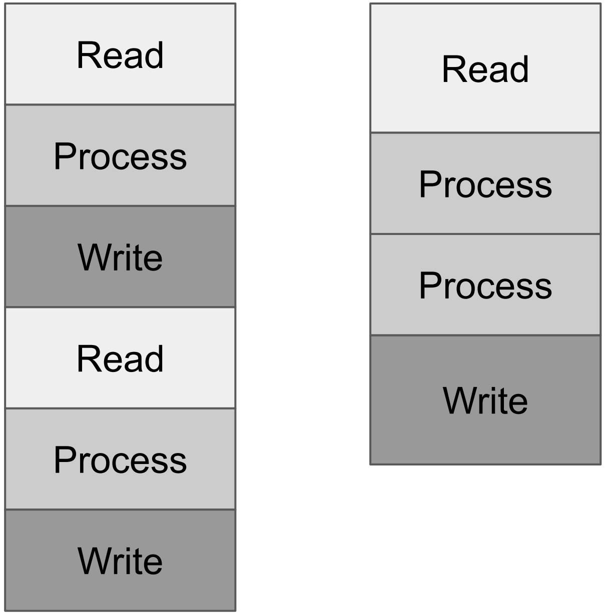
# Event-based vs micro-batch

- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources by reducing the common parts of each event handling/processing



# Event-based vs micro-batch

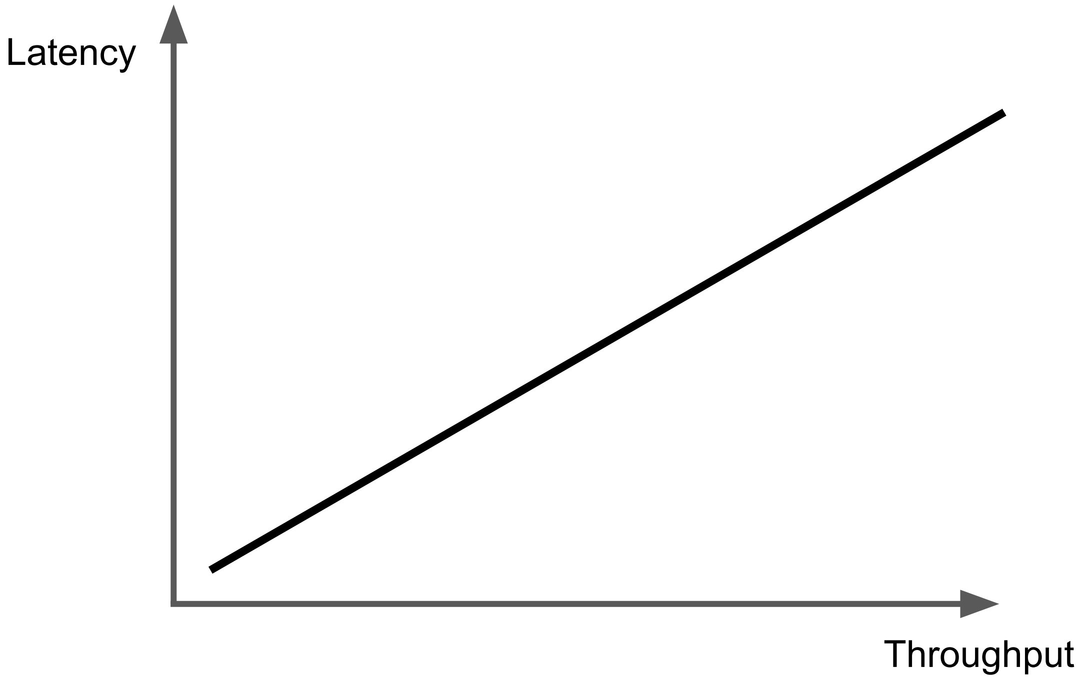
- Event-based allows you to achieve less lag
- Micro-batch allows you to save resources by reducing the common parts of each event handling/processing
- Micro-batch allows you to process more data on the same hardware



# Throughput vs latency

- In real world resources are restricted
- Big data needs a huge throughput => in most cases we a choosing micro-batch
- There is no right answer - you should choose the approach by task

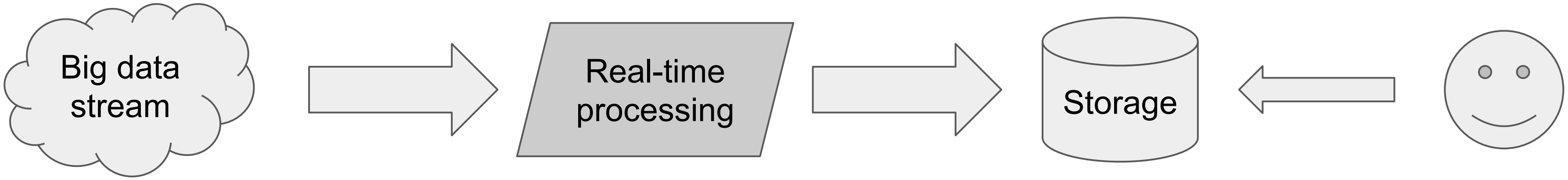
# Throughput vs latency



# Real-Time Big Data

- RT Intro
  - Batch -> RT
  - Approaches to RT data processing
- **Spark Streaming**
  - Application example
  - Hints
- Spark Streaming practice
- Kafka
  - Internal
  - CLI

# RT processing



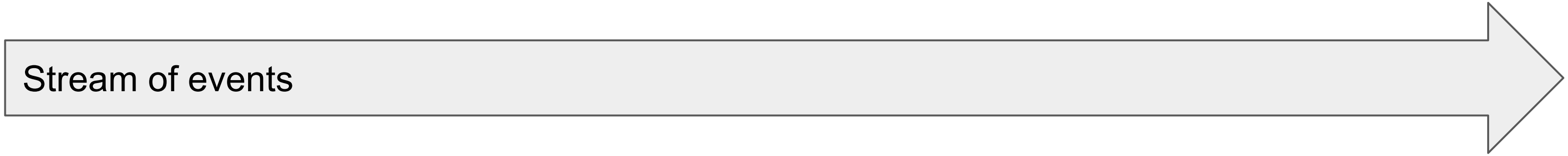

- Apache Spark Streaming - classic
- Apache Spark Structured Streaming - new wave (:

# Spark Streaming

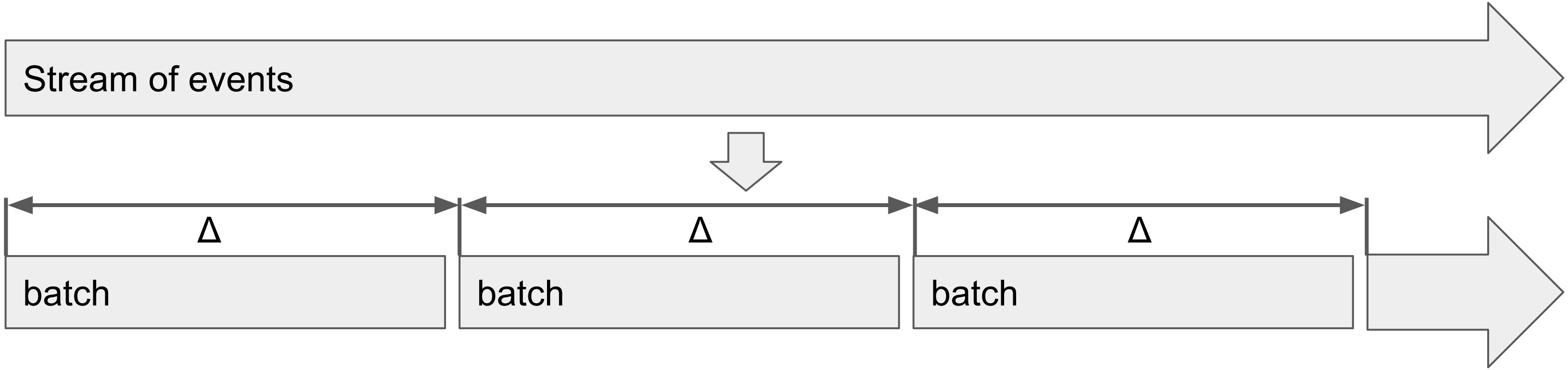
- Spark Streaming is an extension of Spark API core that enables real-time stream processing using micro-batch approach

# Spark Streaming

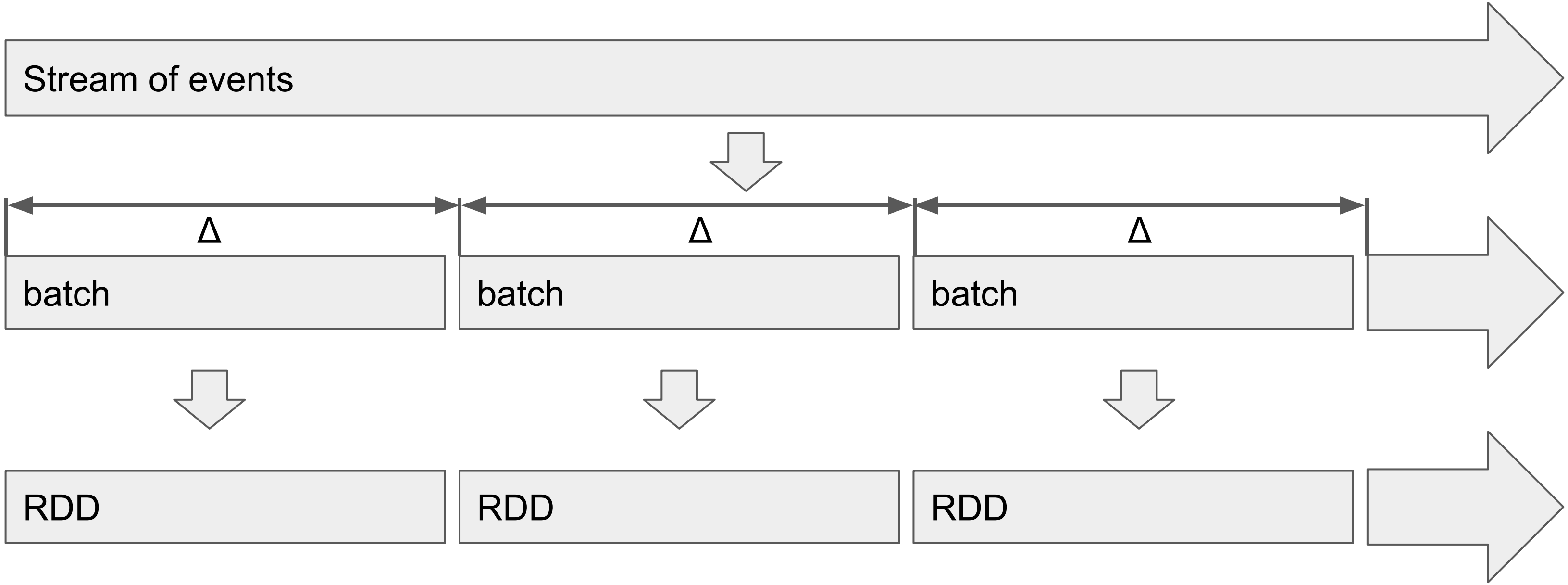
# Spark Streaming - DStream



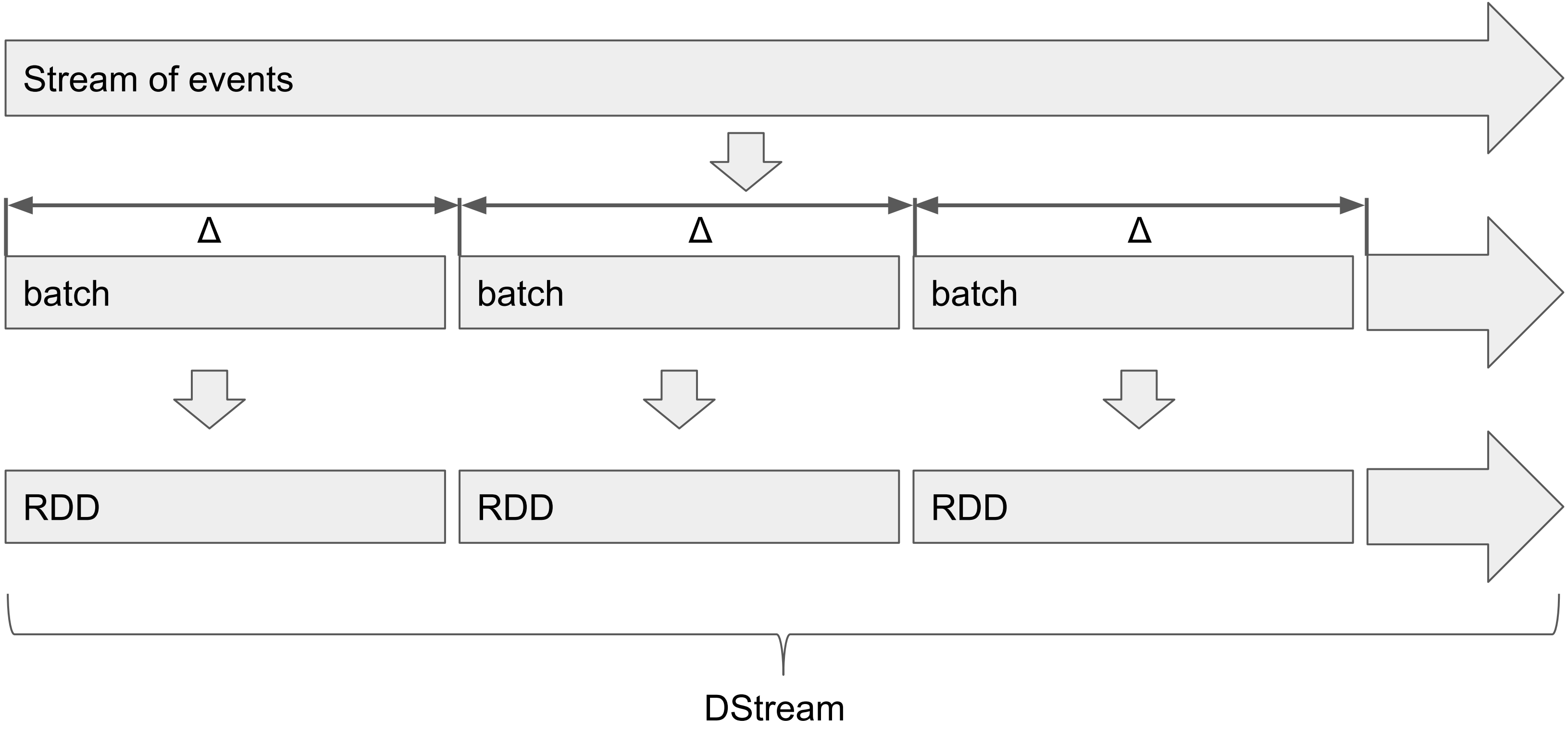
# Spark Streaming - DStream



# Spark Streaming - DStream

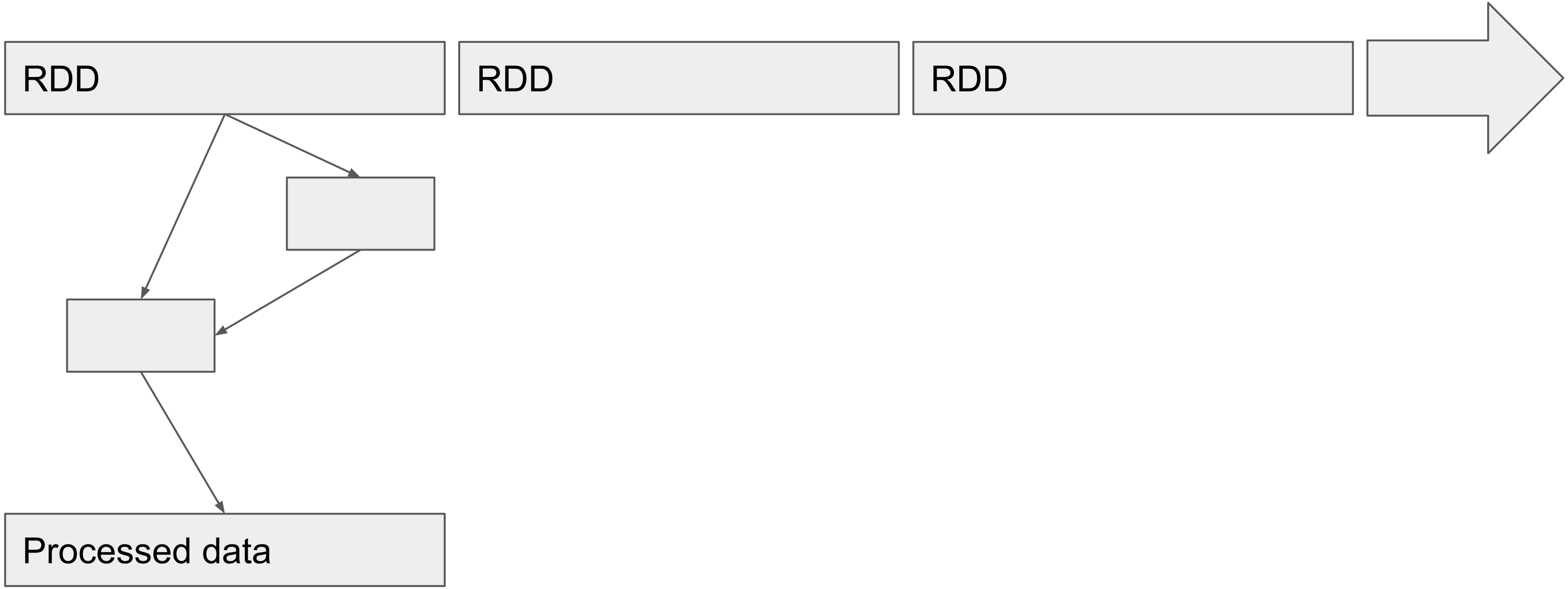


# Spark Streaming - DStream

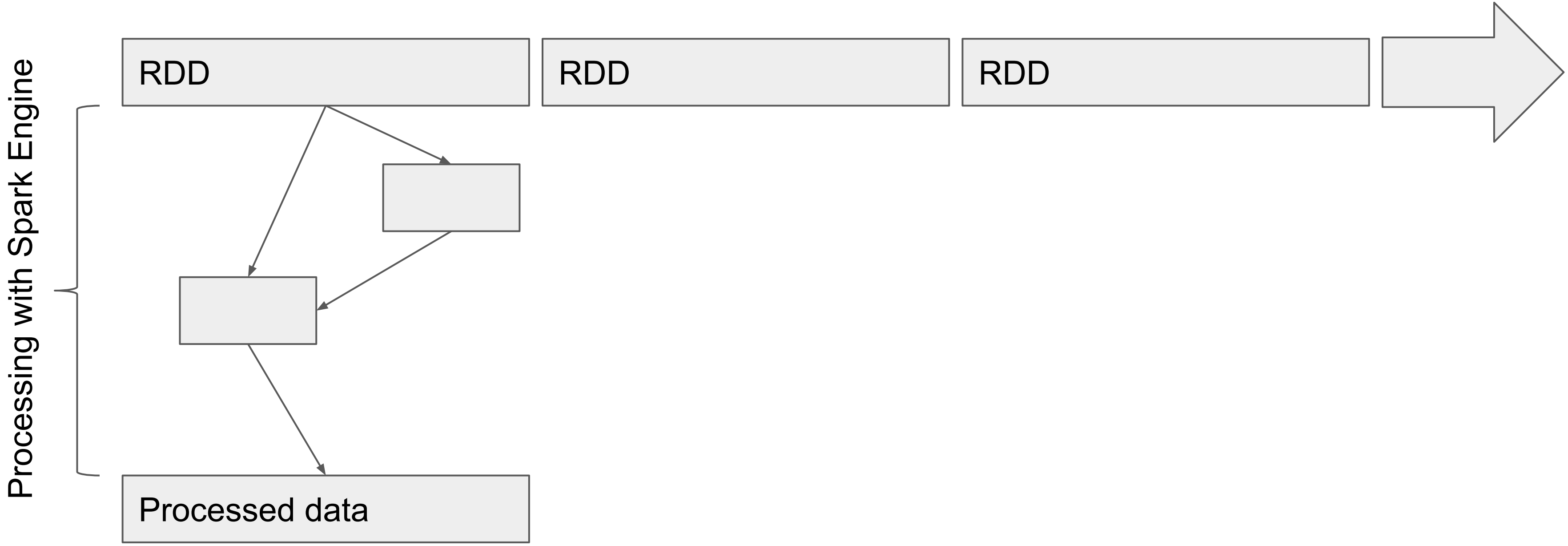


# Spark Streaming - processing

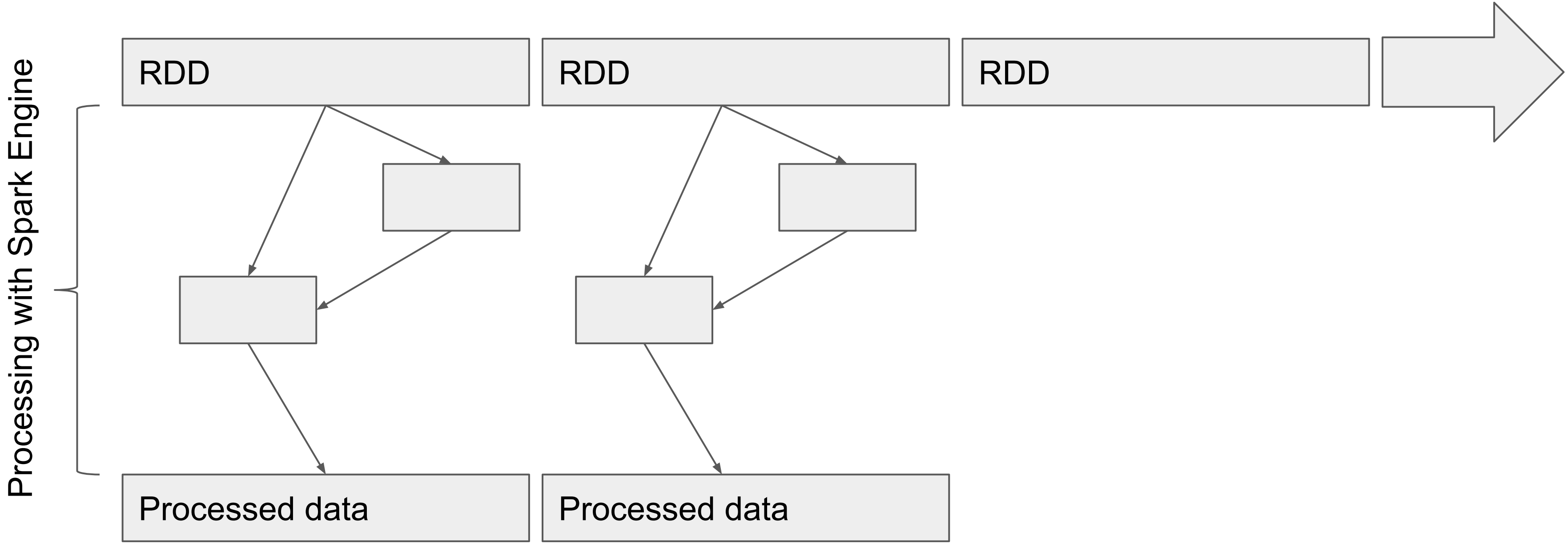
# Spark Streaming - processing



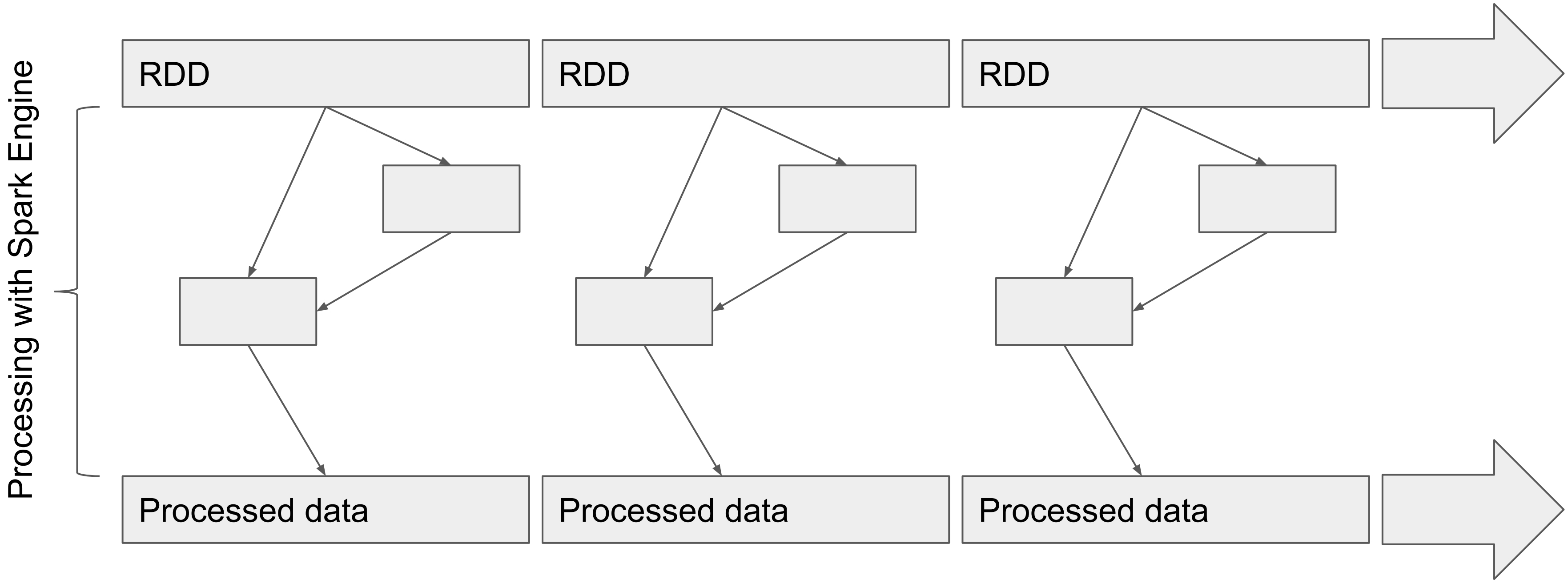
# Spark Streaming - processing



# Spark Streaming - processing



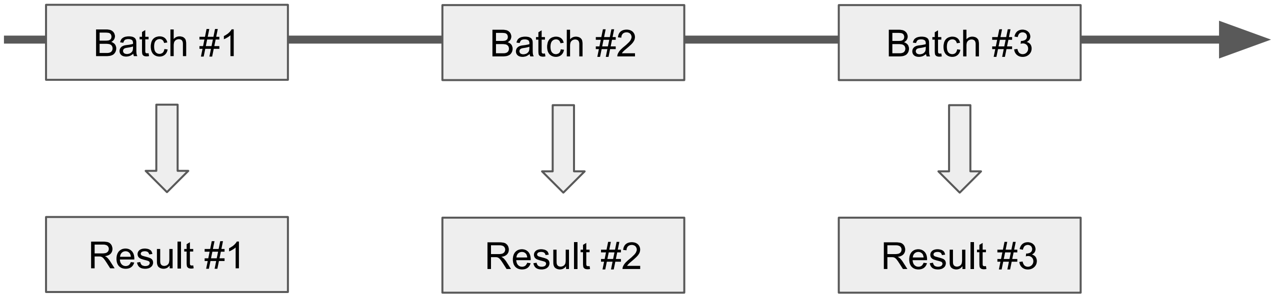
# Spark Streaming - processing



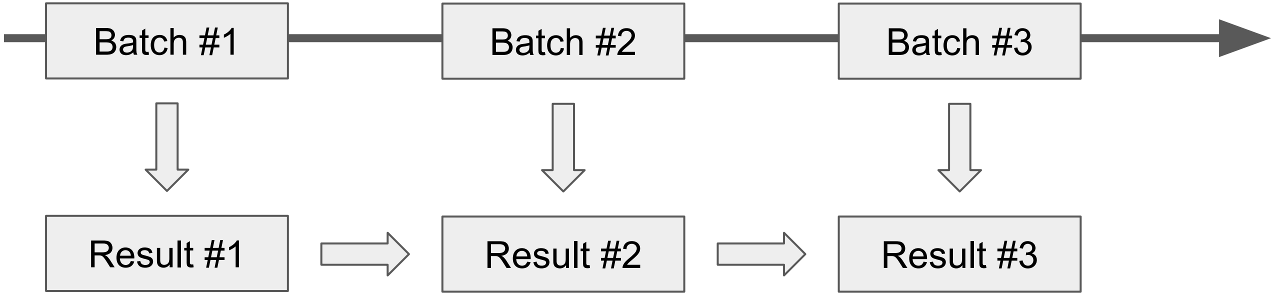
# Spark Streaming vs Spark

- Dstream has the same transformations and actions as RDD
  - *map, filter, repartition, join ...*
- Some new operations on DStream
  - *updateStateByKey(...)* - a way to make Spark Streaming stateful
  - *window(...)* - an implementation of windowed approach
  - *checkpoint(...)* - provides recovering after failures
- Spark Streaming RDD is just Spark RDD

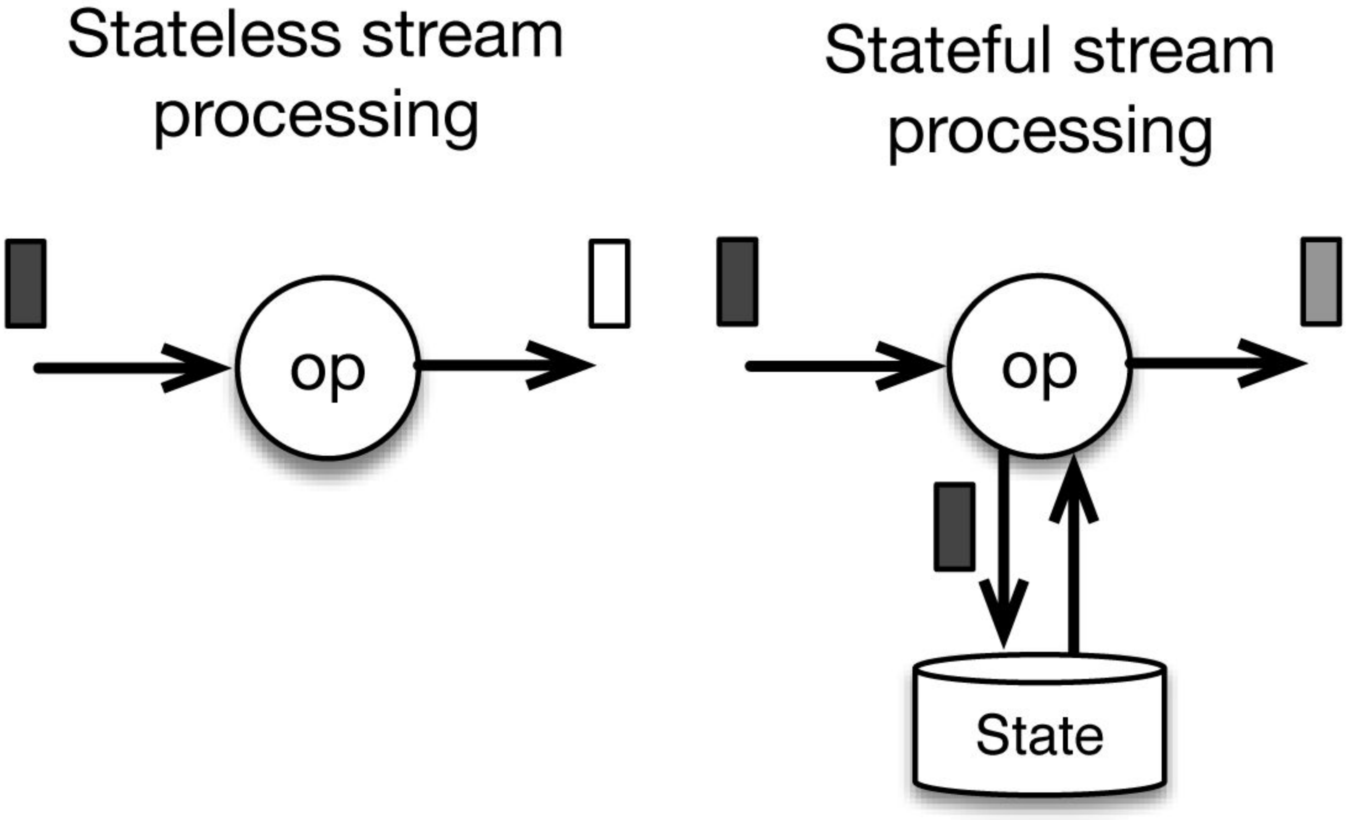
# Stateless processing



# Stateful processing



# Stateful vs Stateless



# Features of Spark Streaming

+ High level abstraction
+ Processing data with Spark
+ Rich spark ecosystem (SparkSQL, SparkML, ...)
▶️ Micro-batch
  + High throughput
  - High latency

- RT Intro
  - Batch -> RT
  - Approaches to RT data processing
- Spark Streaming
  - **Application example**
  - Hints
- Spark Streaming practice
- Kafka
  - Internal
  - CLI

# Spark Streaming context

- Spark Streaming context <- func(Spark context)

```python
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext(master='local[4]')
ssc = StreamingContext(sc, batchDuration=10)
```

# Input sources for Spark Streaming

- DStream <- func(Spark streaming context)
- Basic sources
  - *ssc.fileStream(...)* - folder with files
  - *ssc.socketTextStream(...)* - network socket
  - *ssc.queueStream(...)* -  queue of RDDs for testing purposes
- Advanced sources: Kafka, Flume, Kinesis
- Custom sources (write yourself)

# Input sources examples

- Create DStream over network socket

```
dstream = ssc.socketTextStream(hostname='localhost', port=9999)
```

- Create DStream over Kafka topic

```
from pyspark.streaming.kafka import KafkaUtils

dstream = KafkaUtils.createDirectStream(
    ssc,
    topics=['test_topic'],
    kafkaParams={'metadata.broker.list':
                 'kafka00.example.org:9092,kafka01.example.org:9092'})
```

# Output sources for Spark Streaming

- *dstream.pprint()* - prints the result to stdout
- *dstream.saveAsTextFiles(...)* - saves data to external storage (e.g. HDFS)
- *dstream.foreachRDD(...)* - saves data manually (e.g. external key-value storage)
  - The most common option
  - Providing the required semantics is entirely in the developer's jurisdiction

# Output sources examples

- Save data to HDFS

```
dstream.saveAsTextFiles('hdfs://cluster/path/to/result/')
```

- Save data manually

```
dstream.foreachRDD(lambda rdd: write_result_to_db(rdd))
```

# Sample Spark Streaming application

- Create context
- Create DStream
- Process DStream on Spark
- Write the result
- Start streaming
- Wait for exit (only for CLI)

```python
from pyspark import SparkContext
from pyspark.streaming import StreamingContext

sc = SparkContext(master='local[4]')
ssc = StreamingContext(sc, batchDuration=10)

dstream = ssc.socketTextStream(hostname='localhost', port=9999)

result = dstream \
    .filter(bool) \
    .count()

result.pprint()

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

# How to run Spark Streaming application from CLI

- Run the netcat, that can send data into socket

```
$ nc -lk 9999
```

- Create *spark_streaming_example.py* file and put the source code in it
- Run python file with Spark

```
$ spark-submit spark_streaming_example.py
```

- Send some data through netcat
- You can exit by pressing *Ctrl-C*

# How to run Spark Streaming application from iPython

- Run the netcat, that can send data into socket

  `$ nc -lk 9999`
- Run application code (first 15 rows) in iPython
- Send some data through netcat
- You can exit by calling method

```
ssc.stop()
```

# Real-Time Big Data

- RT Intro
  - Batch -> RT
  - Approaches to RT data processing
- Spark Streaming
  - Application example
  - **Hints**
- Spark Streaming practice
- Kafka
  - Internal
  - CLI

# Repartition

- *dstream.repartition(num_of_partititions)* - distributes the data across the specified number of executors in the cluster before further processing
- Allows you to get linear scalability and data shuffle
- An example of code

```
dstream.repartition(30)
```

# Broadcast

- Broadcast - creates a local copy of a read-only variable on each server
- Useful for sharing config or complex data structures between Spark Streaming stages
- An example of code

```
multiplier = config.get_multiplier()  # will be run once
broadcast_multiplier = sc.broadcast(multiplier)

result = dstream\
    .map(lambda x: float(x) * broadcast_multiplier.value)
```

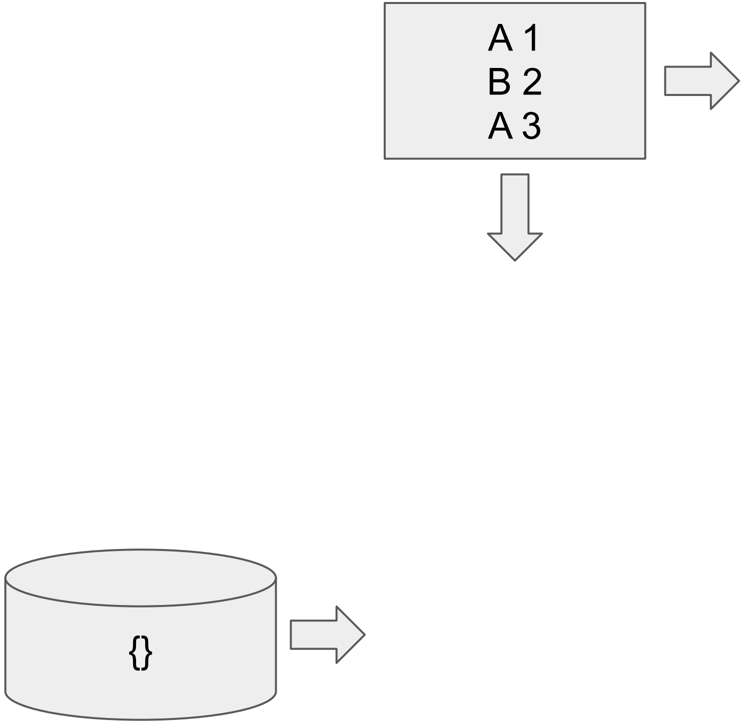
# UpdateStateByKey

- *dstream.updateStateByKey(update_func, ...)* - allows you to store key-value pairs and update the values with every batch of data
- It gives the ability to obtain stateful calculations in Spark Streaming
- To use updateStateByKey:
  - Set the checkpoint directory
  - Specify the function how to update the state

# Example of updateStateByKey usage

Input stream

State by key



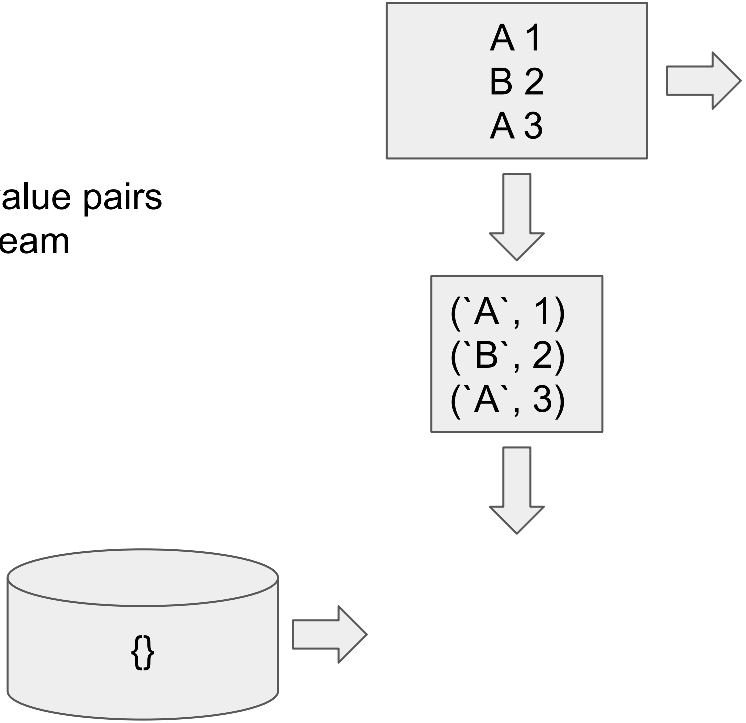
# Example of updateStateByKey usage

Input stream

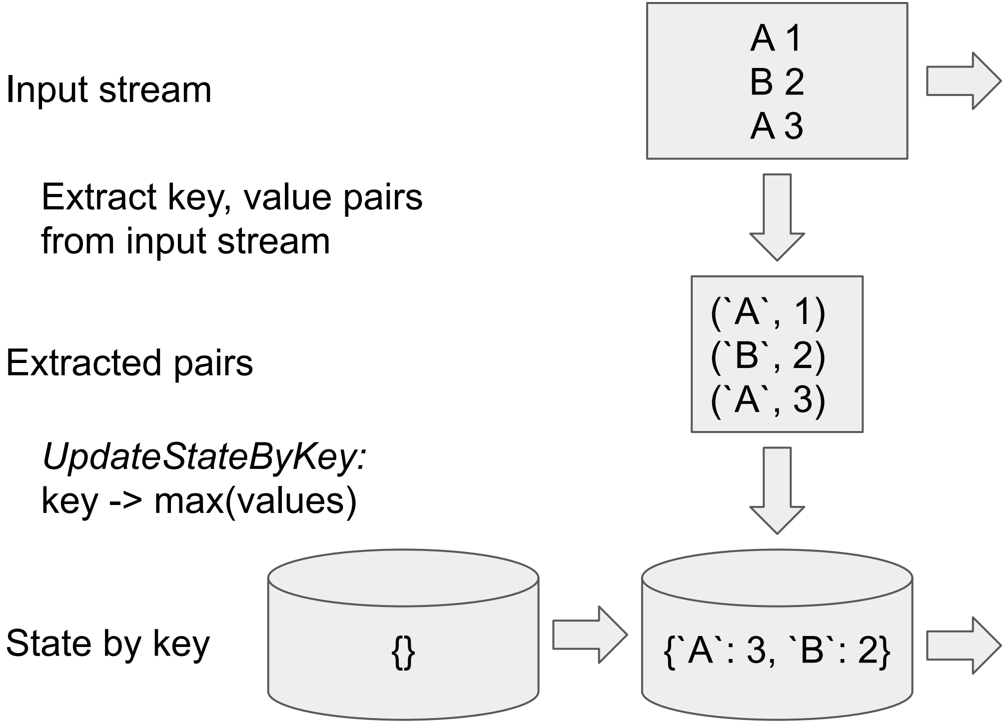
Extract key, value pairs
from input stream

Extracted pairs

State by key



# Example of updateStateByKey usage



# Example of updateStateByKey usage

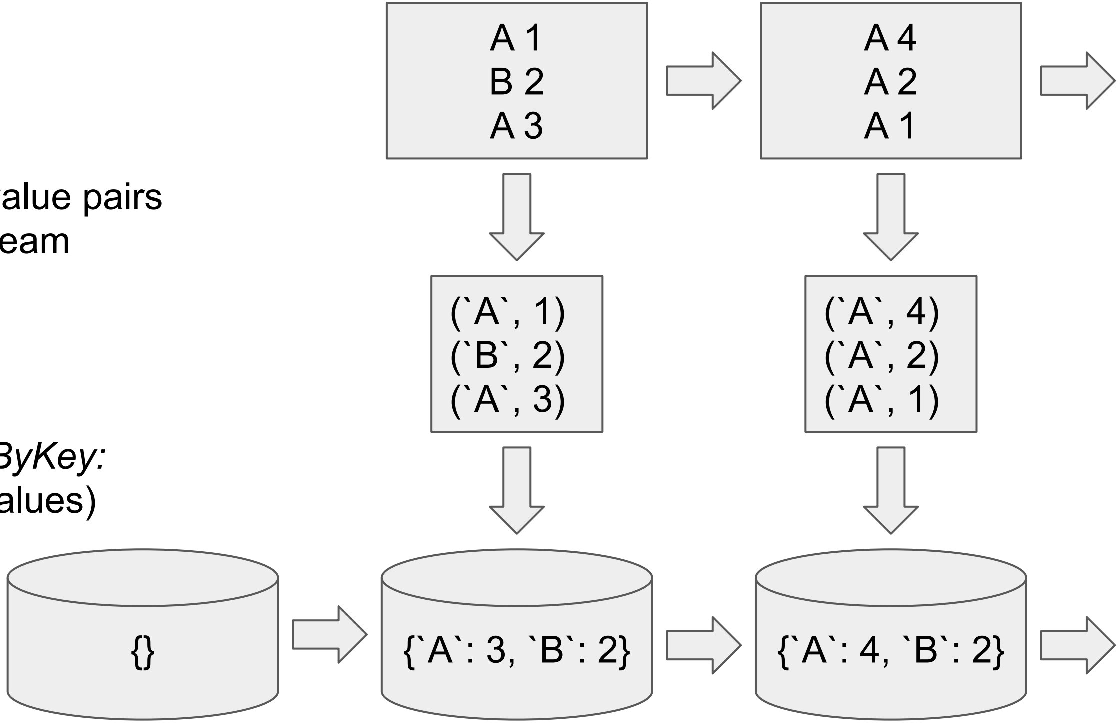
Input stream

Extract key, value pairs
from input stream

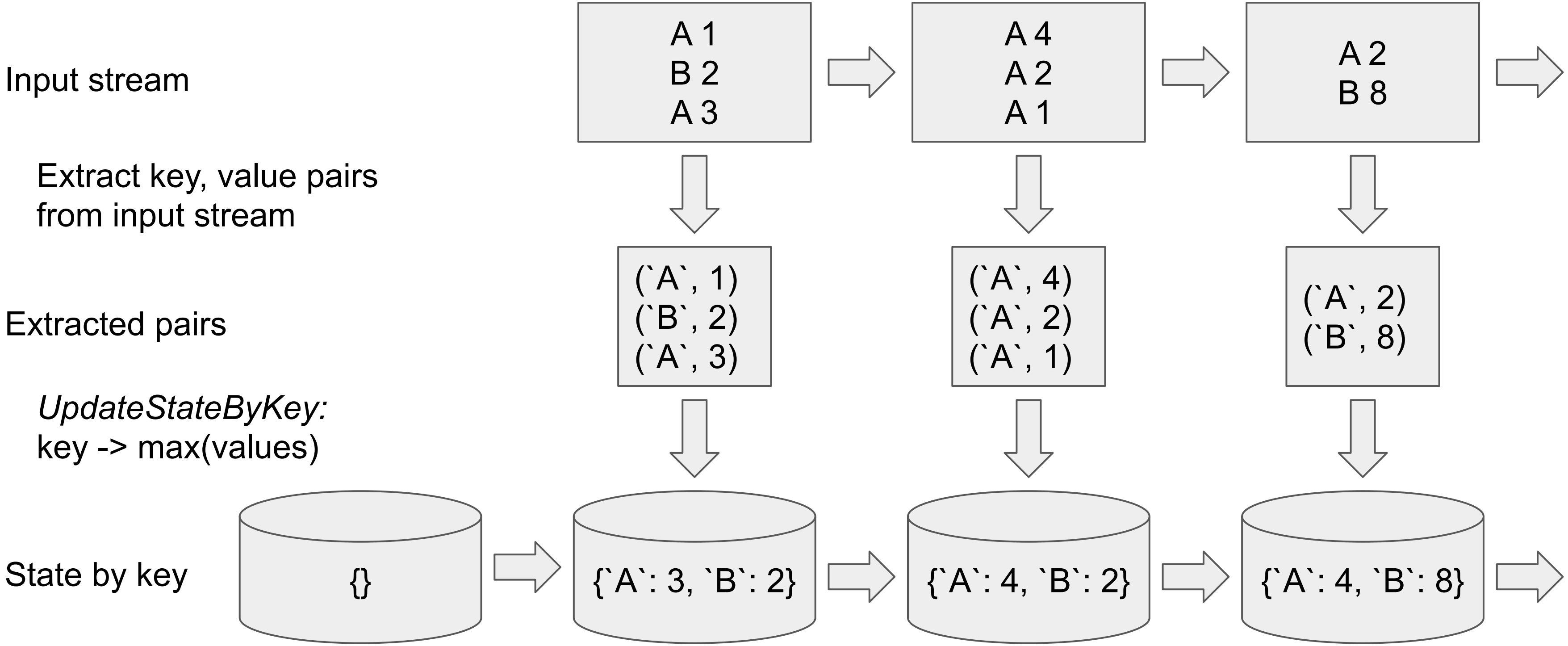
Extracted pairs

*UpdateStateByKey:*
key -> max(values)

State by key



# Example of updateStateByKey usage



# Example of updateStateByKey usage

- Find the total maximum for each key

```python
def update_func(new_values, state):
    max_new = max(new_values) if new_values else None
    return max(max_new, state)

result = dstream \
    .map(lambda line: line.split()) \
    .map(lambda (x,y): (x, int(y))) \
    .updateStateByKey(update_func)
```

# Window

- *dstream.window(...)* - provides a windowed approach, which allows you to apply transformations over a sliding window of data
  - Size - window duration (should be multiple of the slide duration)
  - Shift - sliding duration (should be multiple of the batch duration)
- The windowed approach is suitable to determine batches with intersection or holes between them

# Window example

- Monitoring response time of the server
- The response time comes in Spark Streaming as a stream of numbers from a network socket. It is required to determine median response time for the last minute with seconds precision

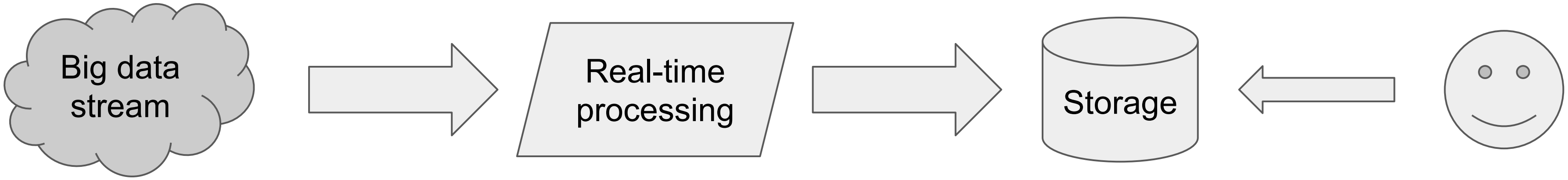
```python
ssc = StreamingContext(sc, batchDuration=1)

dstream = ssc.socketTextStream(hostname='localhost', port=9999)

def print_median(rdd):
    print rdd.mean() if not rdd.isEmpty() else 0

dstream\
    .window(windowDuration=60, slideDuration=1)\
    .map(float)\
    .foreachRDD(print_median)
```

- RT Intro
  - Batch -> RT
  - Approaches to RT data processing
- Spark Streaming
  - Application example
  - Hints
- **Spark Streaming practice**
- Kafka
  - Internal
  - CLI

# Real-Time Big Data

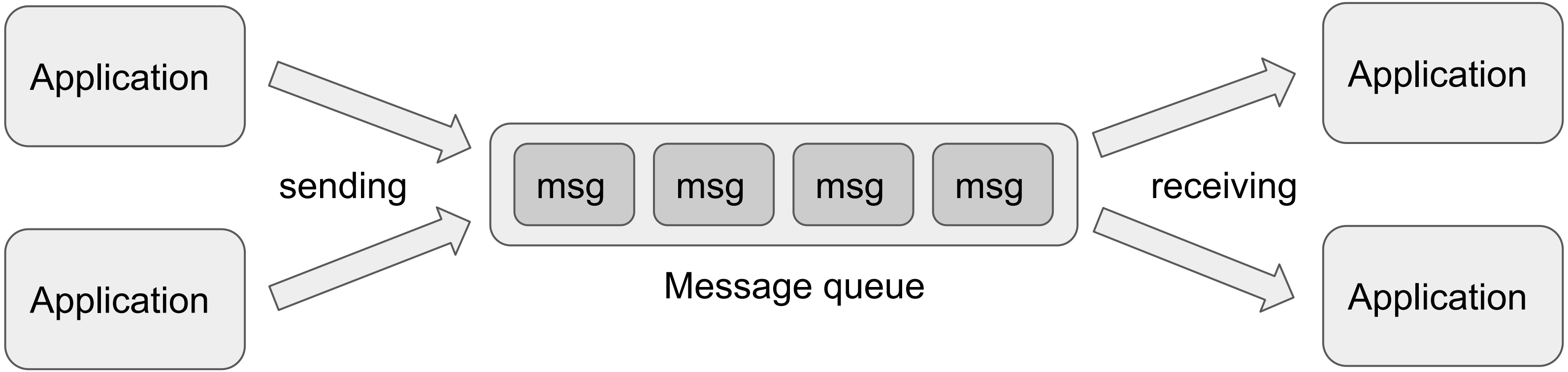
- RT Intro
  - Batch -> RT
  - Approaches to RT data processing
- Spark Streaming
  - Application example
  - Hints
- Spark Streaming practice
- **Kafka**
  - Internal
  - CLI

# Requirements of storage for input data




- Event stream
- Big throughput (hundreds of thousands message per second)
- Small latency (less than 1s)

# Stream handling - message queue




- Message queue provides an asynchronous communications protocol between applications or between processes/threads inside a single application

# Key features of classical message queue

- Complex schemes of message delivery
  - Reduces the throughput
- Per-message state
  - Reduces the throughput
- Stores the data in RAM
  - Not persistent storage
  - Strongly limits the amount of stored data

# Storage of events in big data world

- Simplify the message delivery scheme
- Per message state not tracked by queue
- High throughput
- Persistent storage for a big amount of data

APACHE kafka®

Google Cloud Pub/Sub

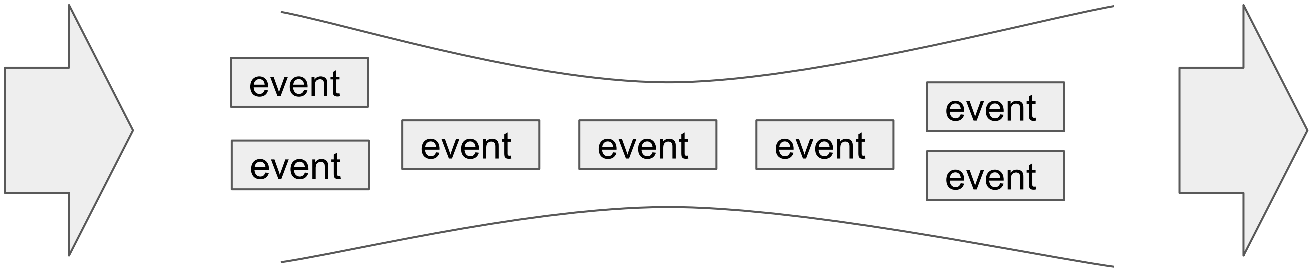
PULSAR

Amazon Kinesis

# Kafka

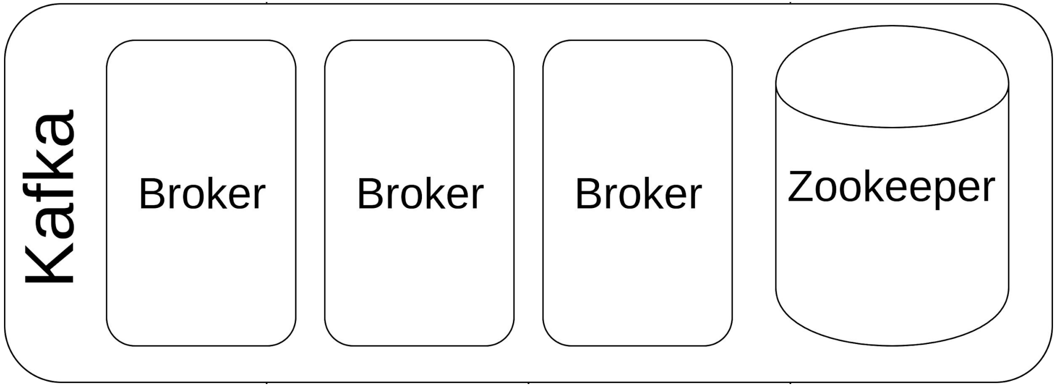
- Kafka is a unified, high-throughput, low-latency platform for handling real-time data feeds
- Kafka is a data bus for big data
- Kafka is an input events storage for real-time processing
- Kafka is an event-based real-time processing engine (Kafka Streams)

# Kafka

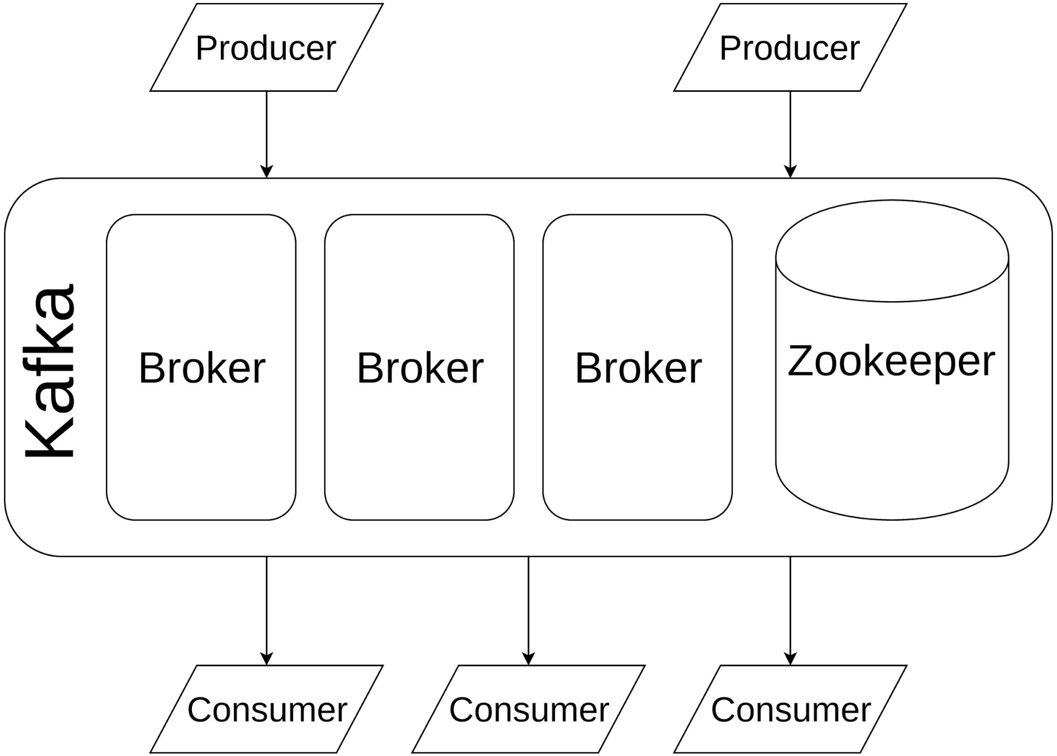


- RT Intro
  - Batch -> RT
  - Approaches to RT data processing
- Spark Streaming
  - Application example
  - Hints
- Spark Streaming practice
- Kafka
  - **Internal**
  - CLI

# Kafka architecture



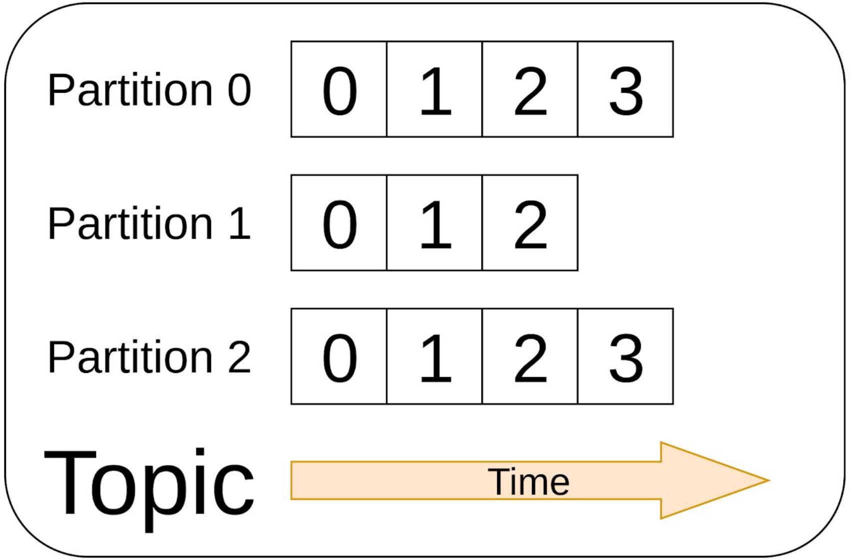
# Kafka architecture



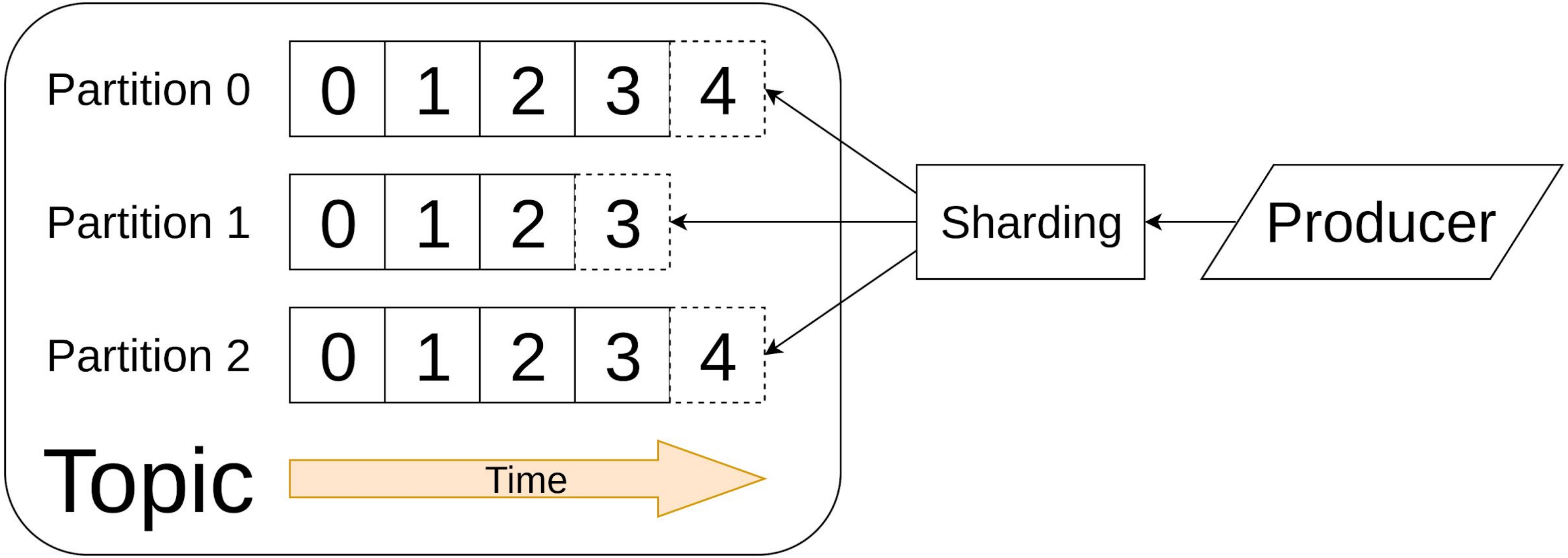
# Topic

Topic

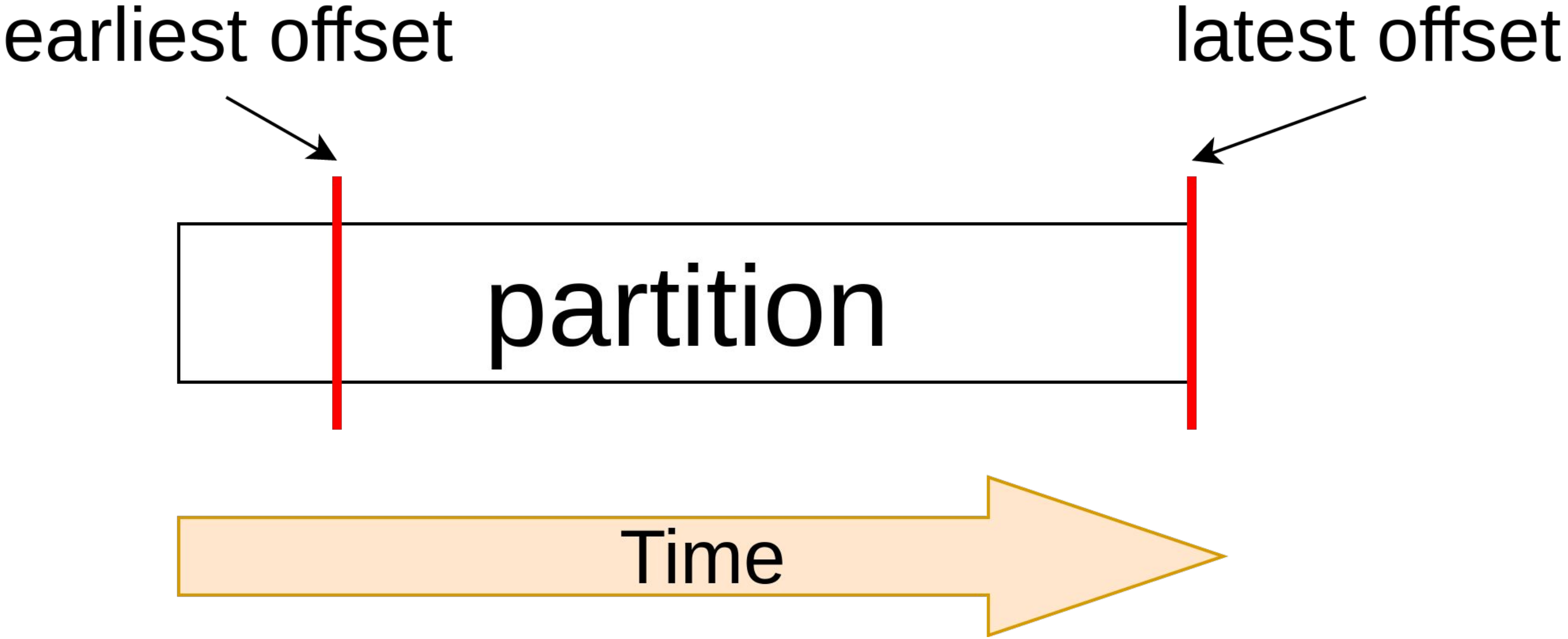
# Topic


Partition 0
0
1
2
3
Partition 1
0
1
2
Partition 2
0
1
2
3
Topic
Time

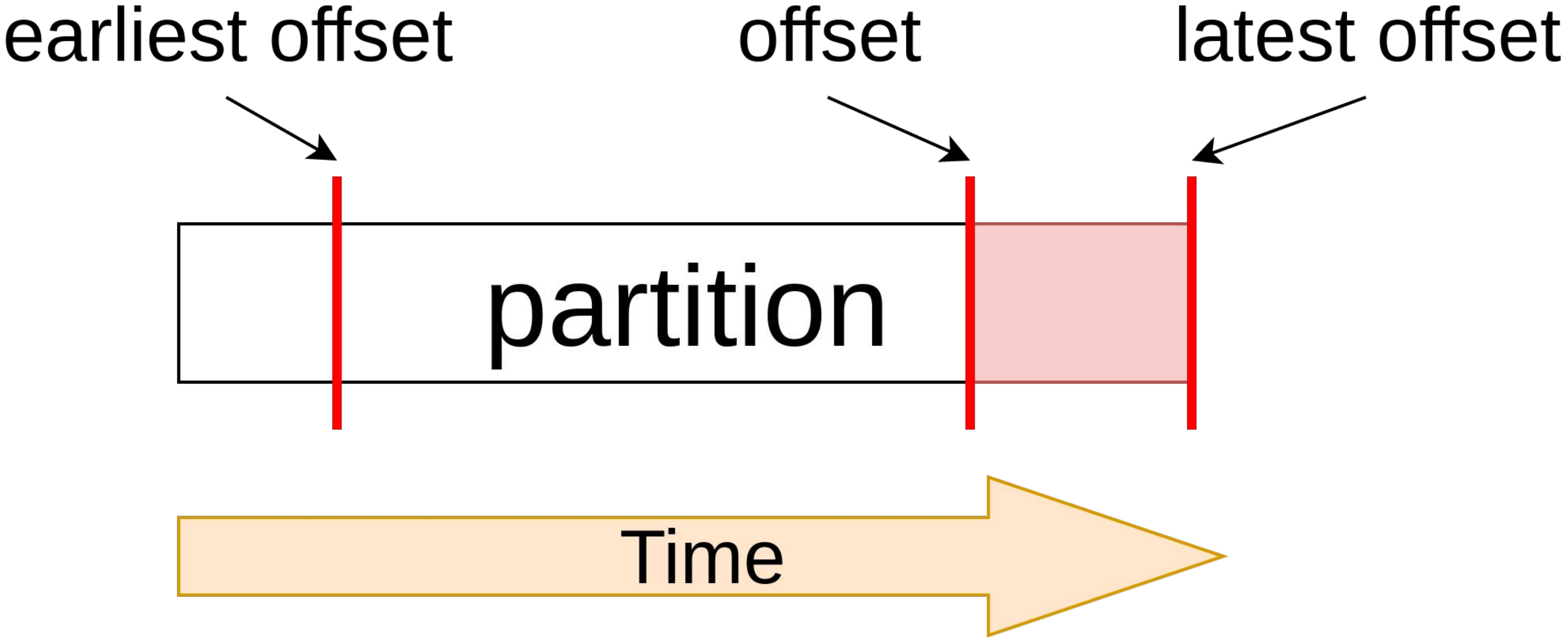
# Topic & Write


Partition 0
0
1
2
3
4
Partition 1
0
1
2
3
Partition 2
0
1
2
3
4
Topic
Time
Sharding
Producer

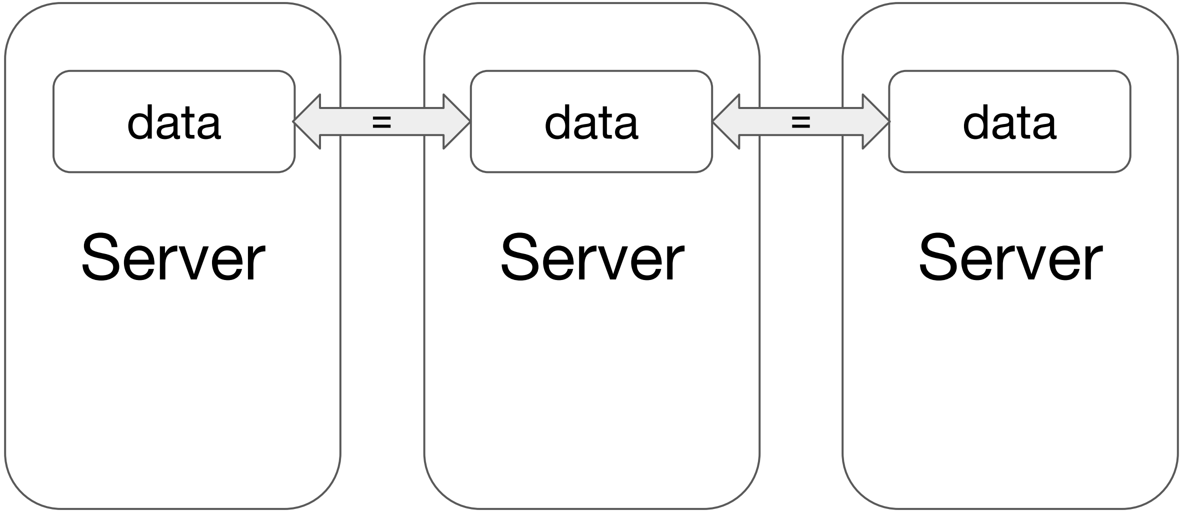
# Offsets



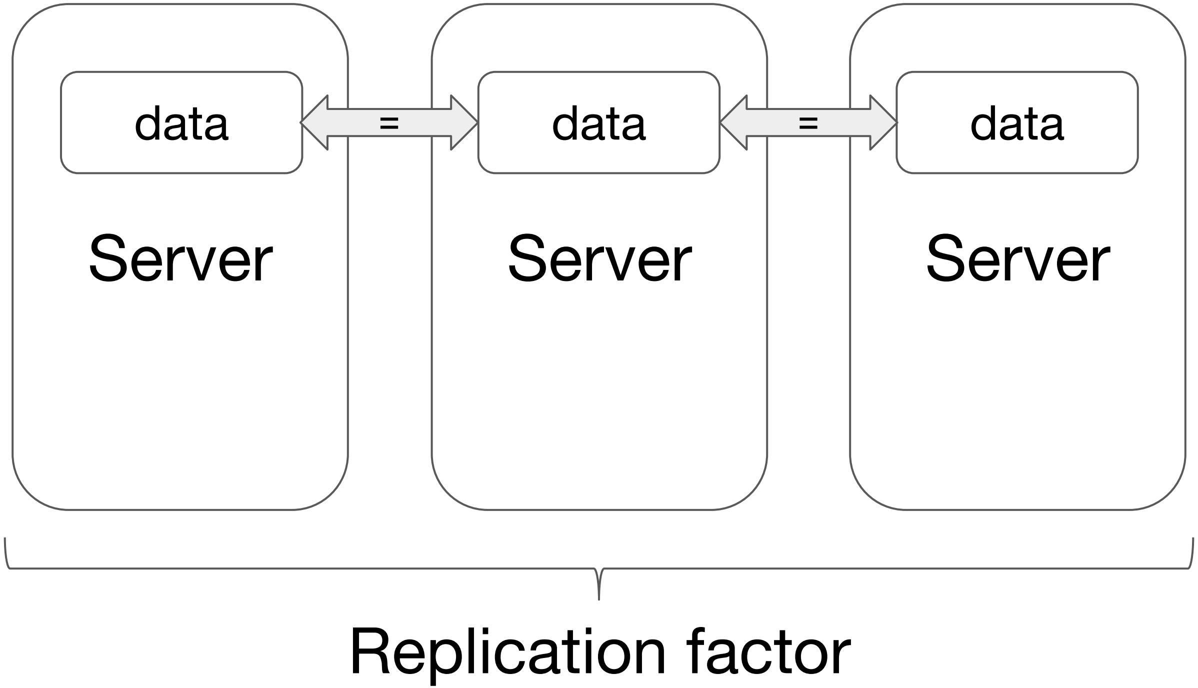
# Offsets & Read



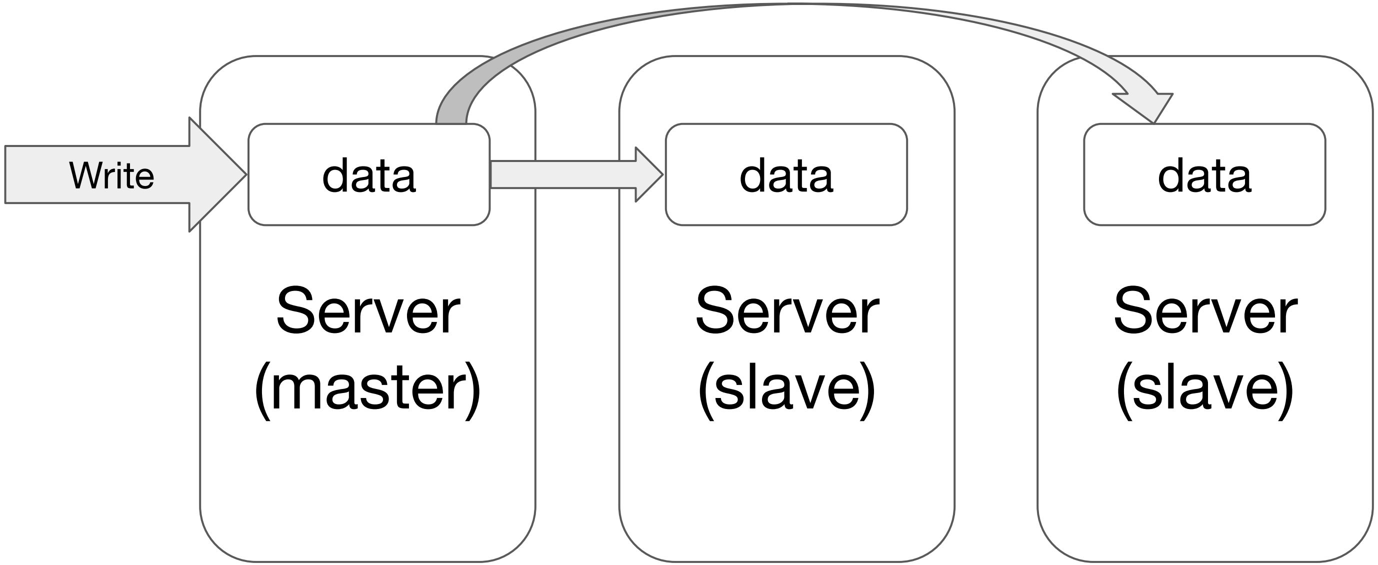
# Replication



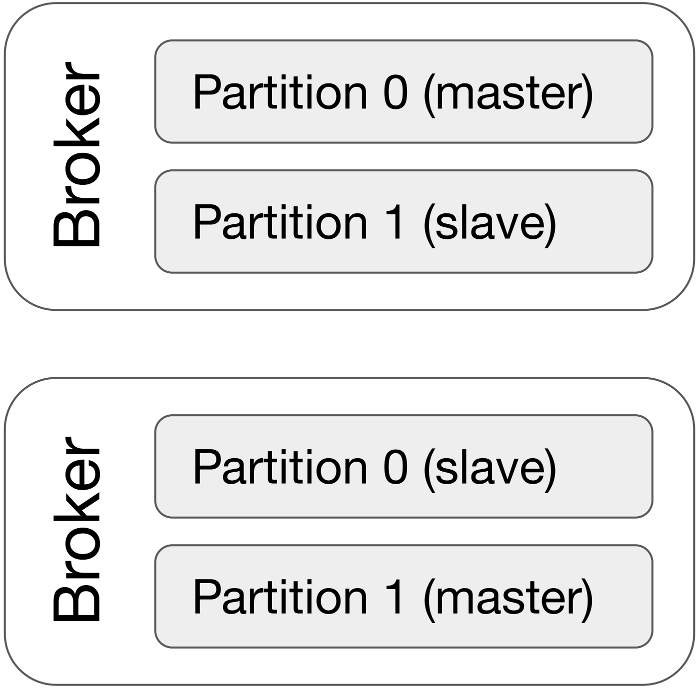
# Replication



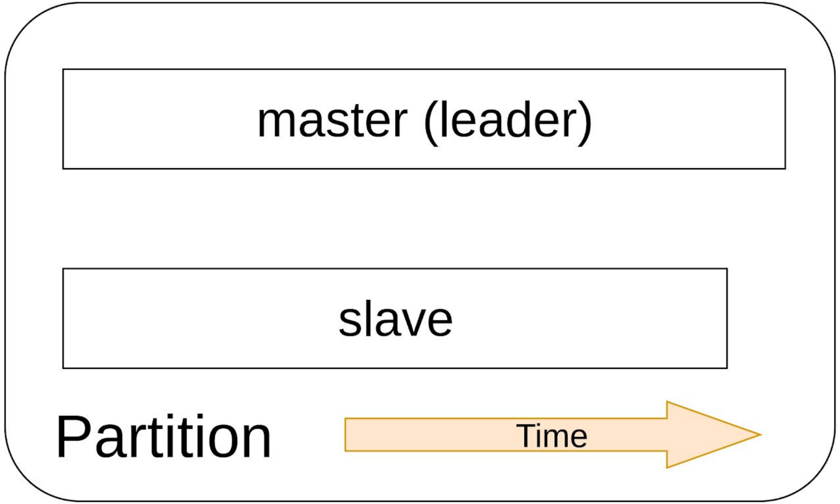
# Replication



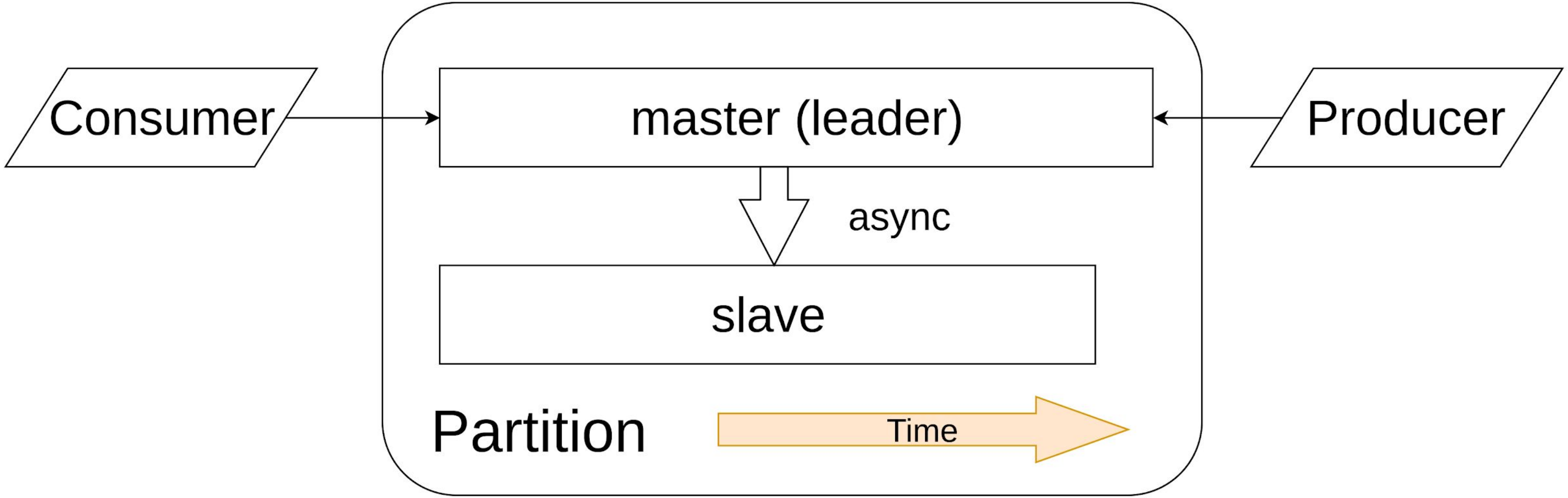
# Kafka replication



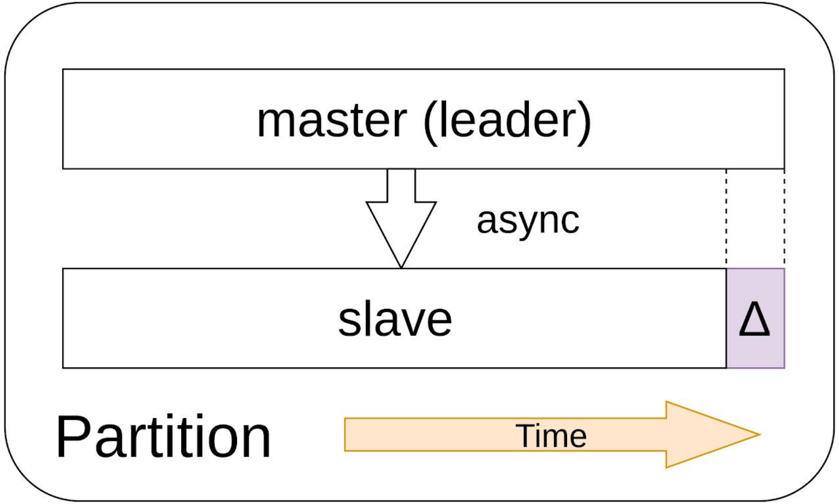
# Kafka replication



# Kafka replication



# Kafka replication delay



- RT Intro
  - Batch -> RT
  - Approaches to RT data processing
- Spark Streaming
  - Application example
  - Hints
- ~~Spark Streaming practice~~
- ~~Kafka~~
  - Internal
  - **CLI**

# Kafka CLI: kafka-topics

- It is the utility to create, delete, describe or change the topic
- `kafka-topics --zookeeper $ZOOKEEPERS --create --topic test_topic --partitions 3 --replication-factor 2`
- `kafka-topics --zookeeper $ZOOKEEPERS --describe --topic test_topic`
- `kafka-topics --zookeeper $ZOOKEEPERS --list`

# Kafka CLI: kafka-console-producer

- It is utility to send data from standard input and to Kafka topic
- `kafka-console-producer --broker-list $BROKERS --topic test_topic`

# Kafka CLI: kafka-console-consumer

- It is utility to read data from Kafka topic
- `kafka-console-consumer --zookeeper $ZOOKEEPERS --topic test_topic --from-beginning`

# Kafka CLI: kafka-run-class

- It is entry point to run any class in the Kafka environment
- `kafka-run-class kafka.tools.GetOffsetShell --broker-list $BROKERS --topic test_topic --time -1`

- RT Intro
  - Batch -> RT
  - Approaches to RT data processing
- Spark Streaming
  - Application example
  - Hints
- Spark Streaming practice
- Kafka
  - Internal
  - CLI

# Thank you! Questions?

**Vybornov Artyom**, avybornov@bigdatateam.org
Big Data Instructor, http://bigdatateam.org/
Head of Big Data Dev Team, Rambler Group
https://www.linkedin.com/in/artvybor/