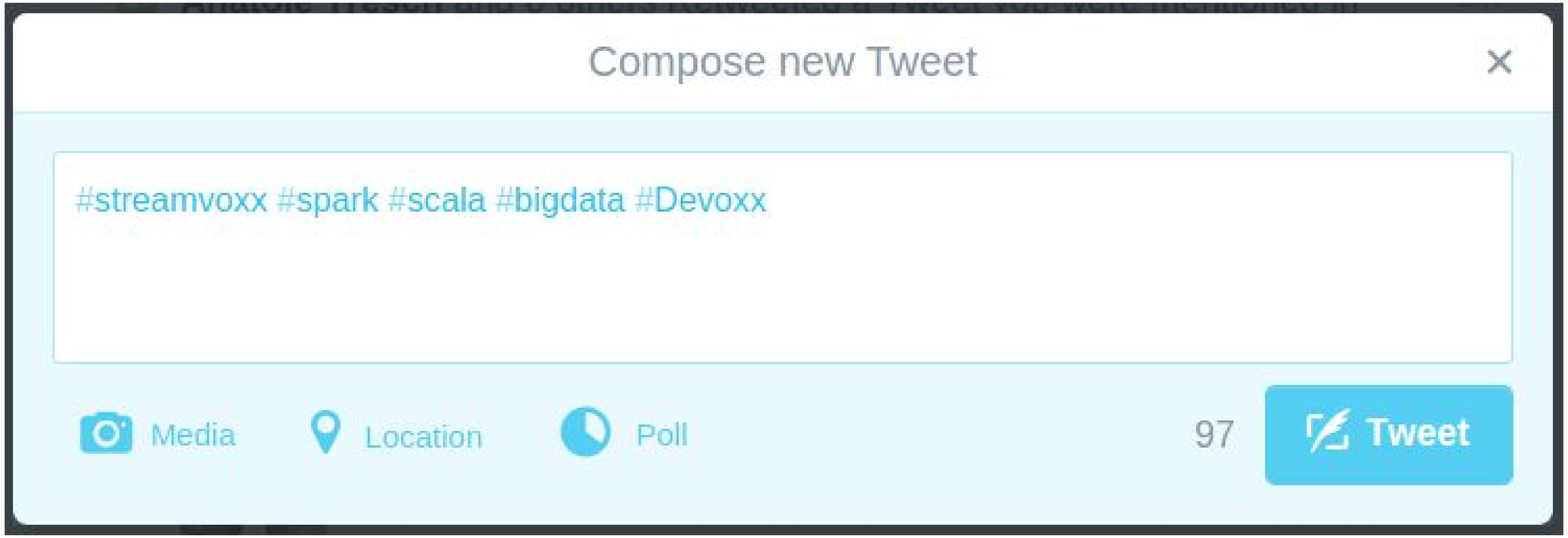
Dive into Spark Streaming by

Gerard Maas
Data Processing Team Lead
gerard.maas@virdata.com @maasg
virdata

Tweet few keywords about your interests and experience.

Use hashtag "#streamvoxx"



#streamvoxx #spark #scala #bigdata #Devoxx

100TB

5MB

100TB

5MB/second

Agenda

Lightning fast Spark recap

What is Spark Streaming?

Programming Model

Demo 1

Execution Model

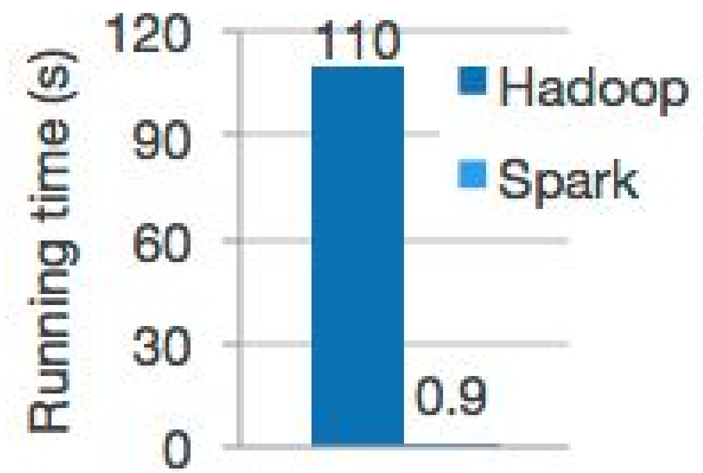
Demo 2

Resources

Q/A

Apache Spark

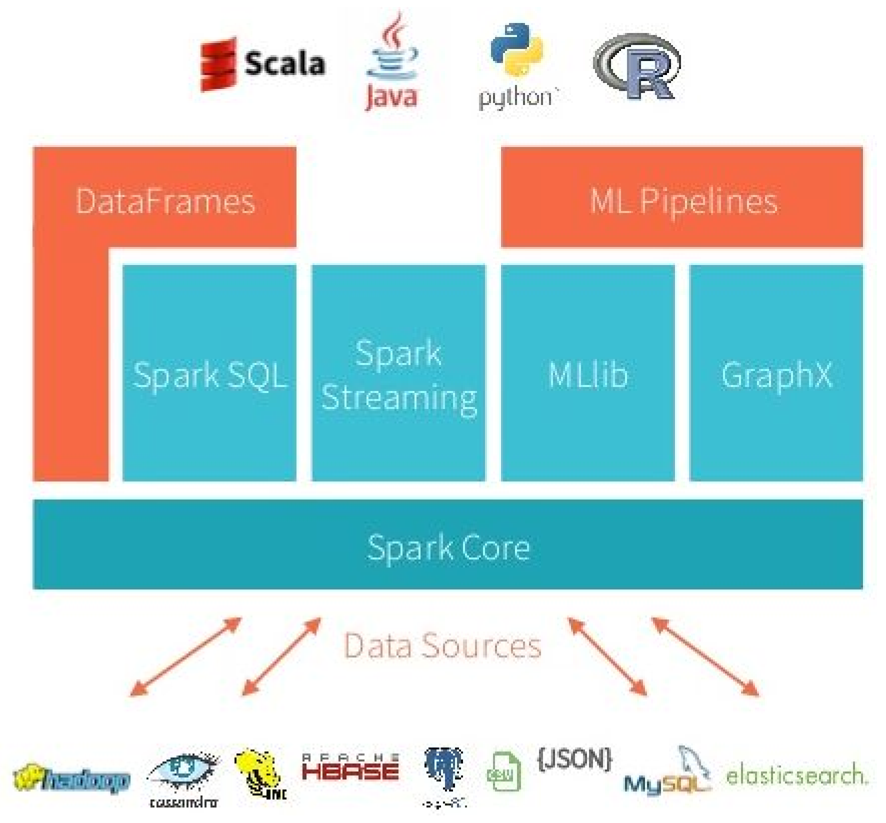
Spark is a fast and general engine for large-scale distributed data processing.



Fast

```
val file = spark.textFile("hdfs://...")

val counts = file.flatMap(line =>  line.
split(" "))
.map(word => (word, 1))
.reduceByKey(_ + _)

counts.saveAsTextFile("hdfs://...")
```

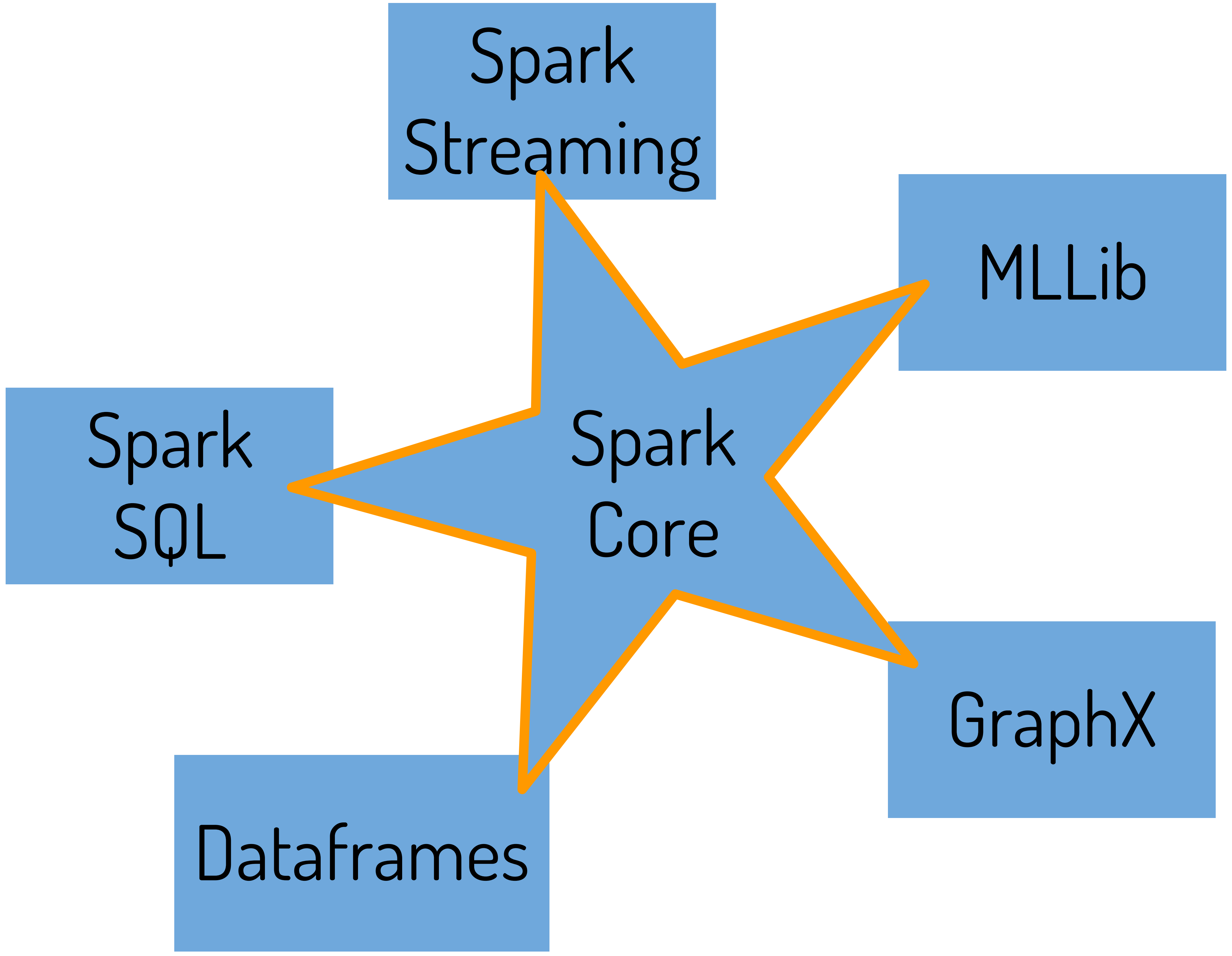
Functional



Growing Ecosystem

#devoxx #streamvoxx

@maasg virdata


Spark Streaming
MLLib
Spark SQL
Spark Core
GraphX
Dataframes

RDDs

Express computations in terms of transformations and actions on a distributed data set.

Spark Core Concept: RDD => **R**esilient **D**istributed **D**ataset

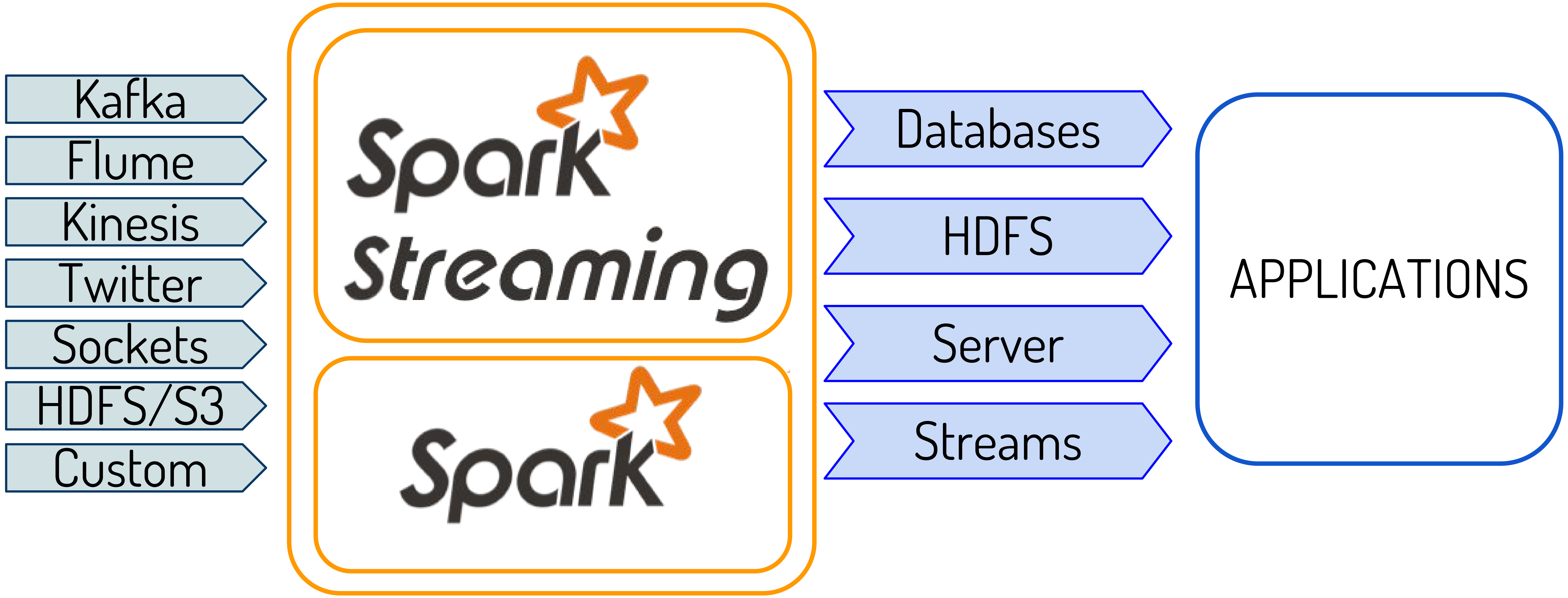
Think of an **RDD** as an immutable, distributed collection of objects

- **R**esilient => Can be reconstructed in case of failure
- **D**istributed => Transformations are parallelizable operations
- **D**ataset => Data loaded and partitioned across cluster nodes (executors)

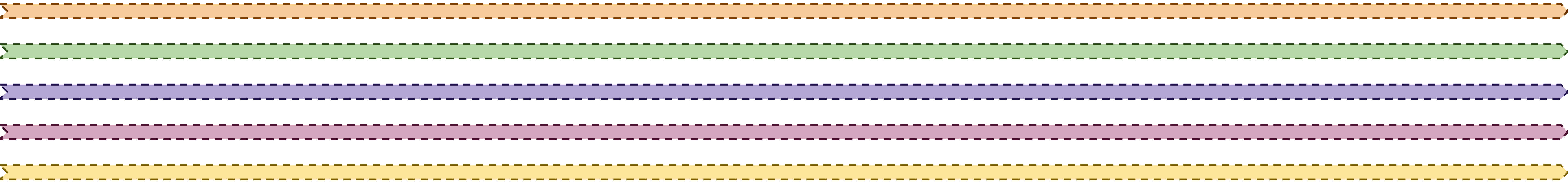
RDDs are memory-intensive. **Caching** behavior is controllable.

Spark Streaming

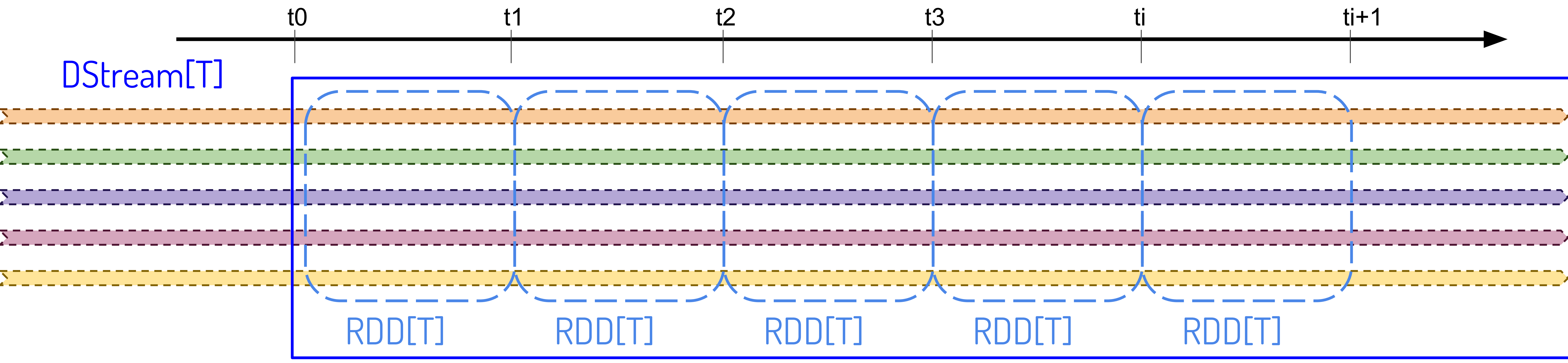
Scalable, fault-tolerant stream processing system



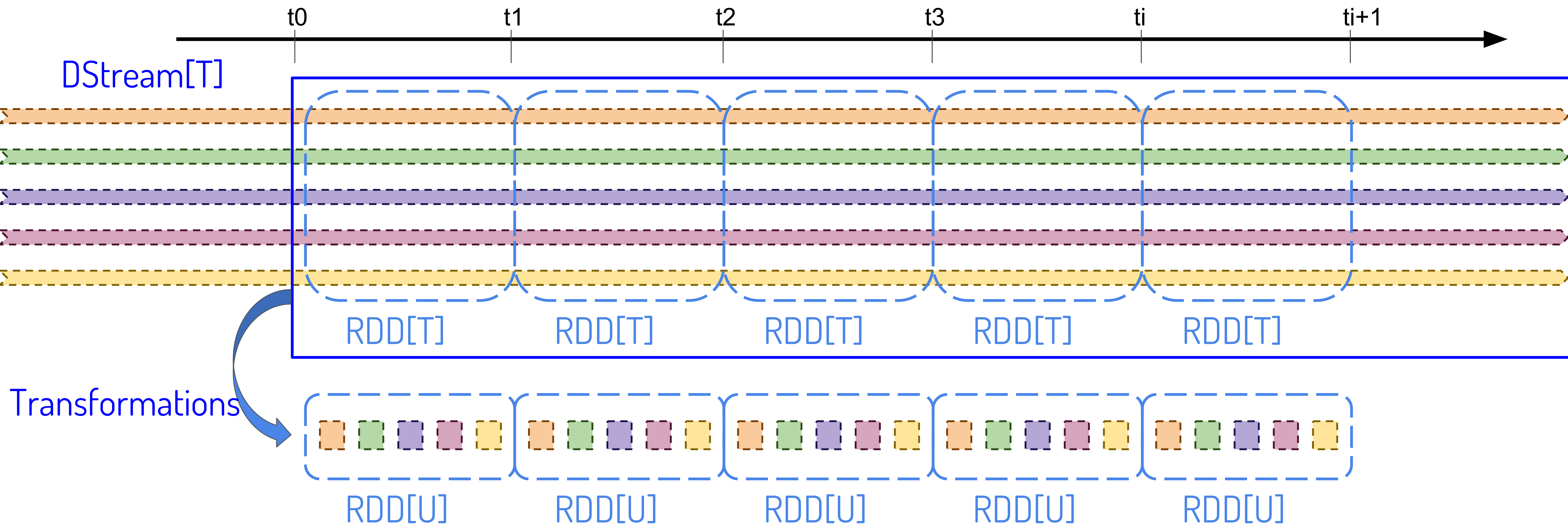
Spark Streaming



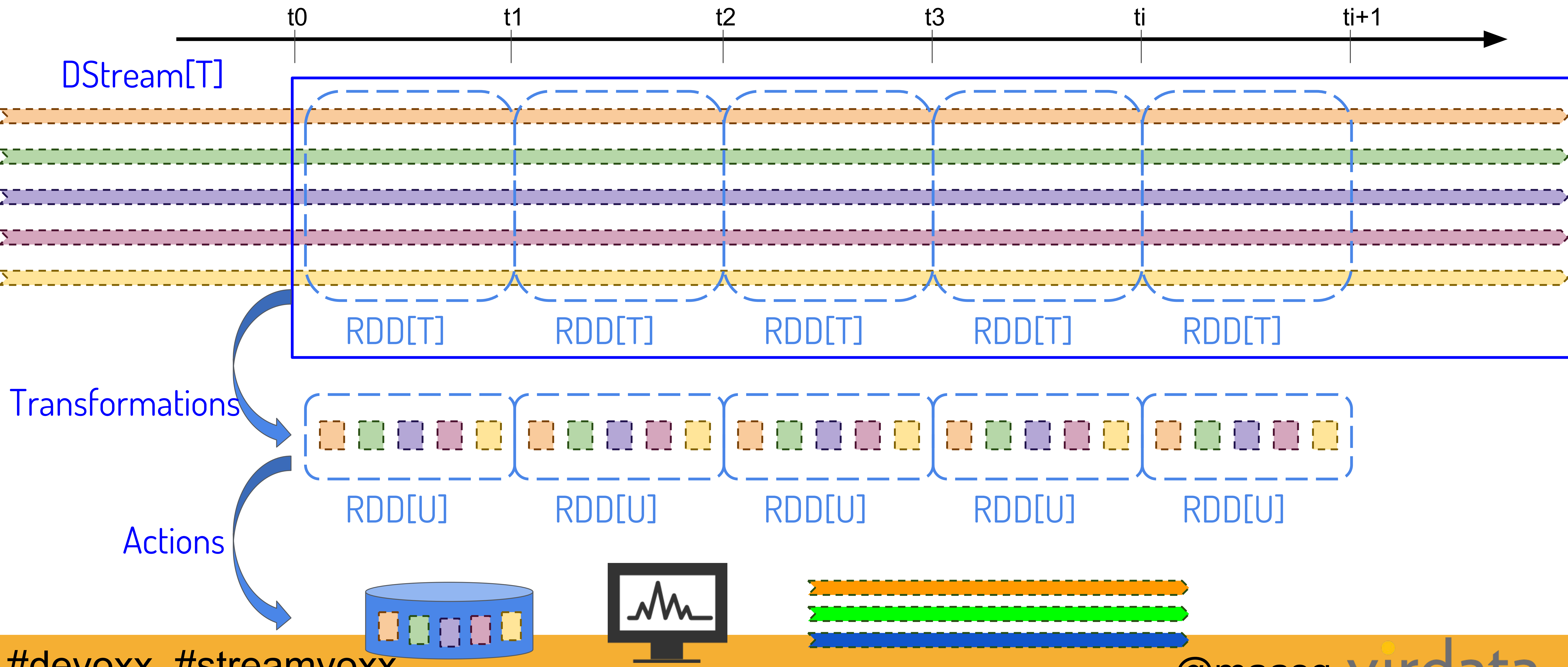
Spark Streaming



Spark Streaming

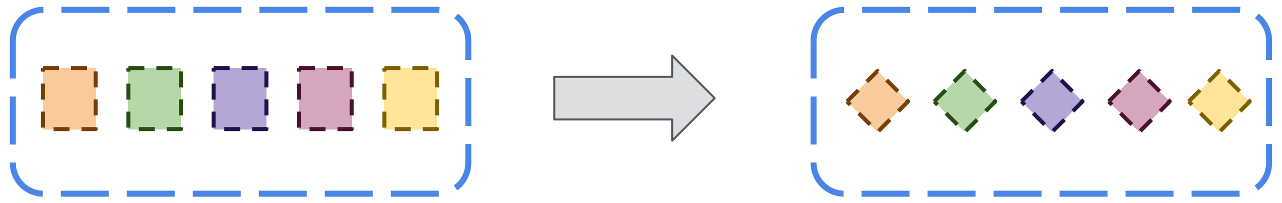


Spark Streaming

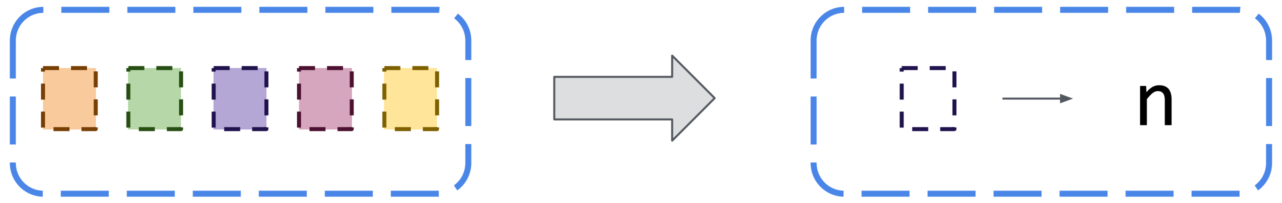


Transformations

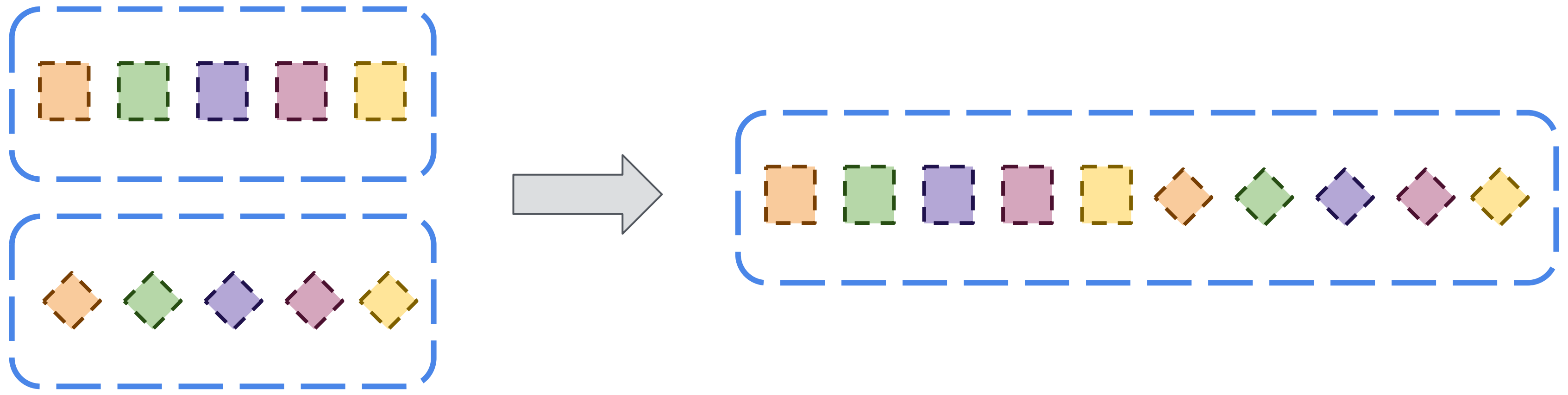
map,
flatmap,
filter



count,
reduce,
countByValue,
reduceByKey

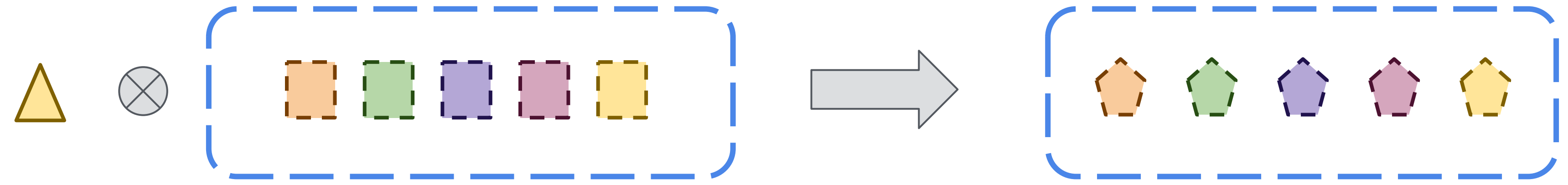


union,
join
cogroup



Transformations

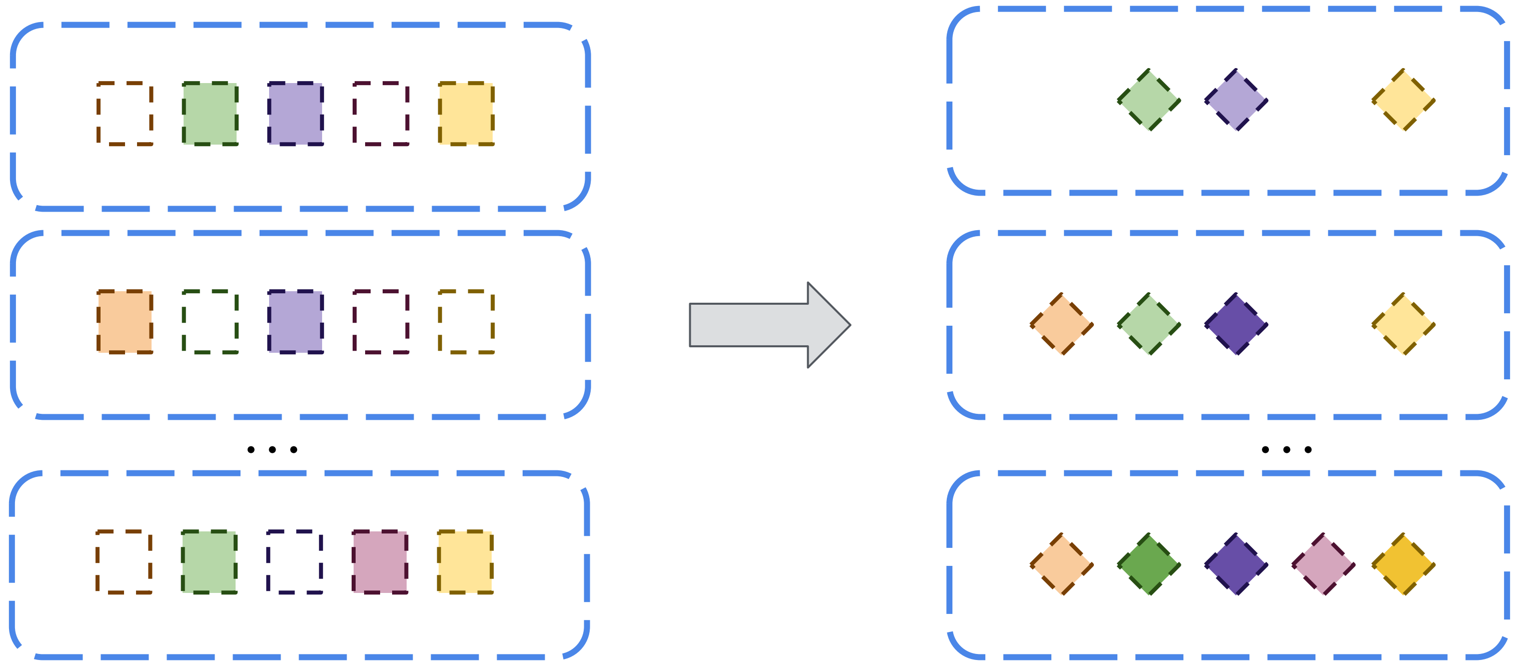
transform



```
val iotDstream = MQTTUtils.createStream(...)
val devicePriority = sparkContext.cassandraTable(...)
val prioritizedDStream = iotDstream.transform{rdd =>
   rdd.map(d => (d.id, d)).join(devicePriority)
}
```

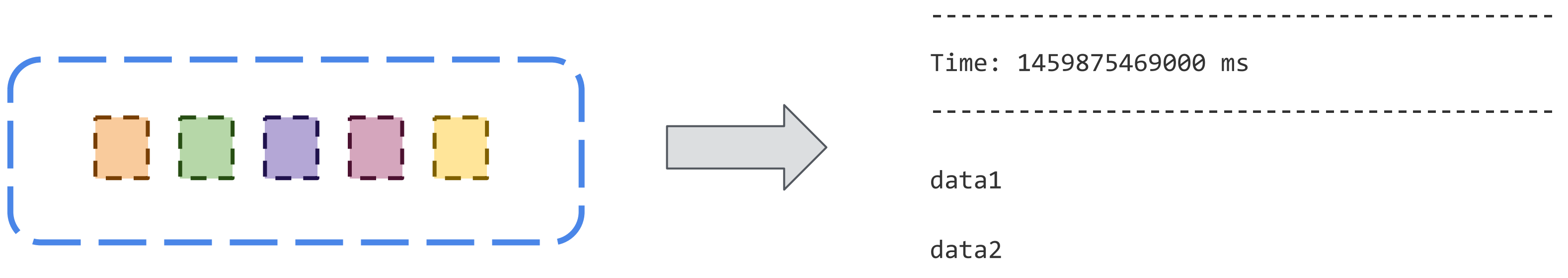
Transformations

updateStateByKey



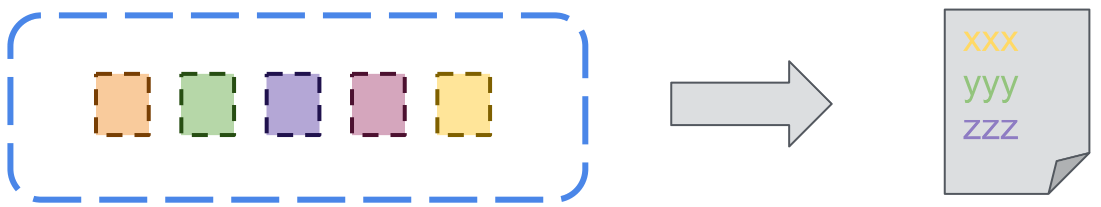
Actions

print

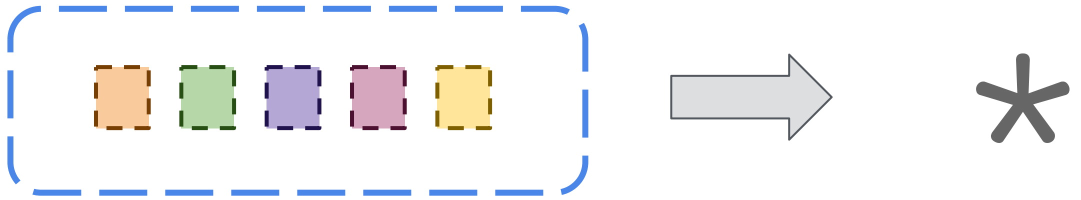


```
-------------------------------------------
Time: 1459875469000 ms
-------------------------------------------
data1
data2
```

saveAsTextFiles,
saveAsObjectFiles,
saveAsHadoopFiles



foreachRDD



Actions - foreachRDD

```
dstream.foreachRDD{rdd =>
```

Spark SQL

MLLib

Dataframes

GraphX

Databases

...

```
}
```

Actions - foreachRDD Usage

```
dstream.foreachRDD{rdd =>
      rdd.cache()
      val alternatives = restServer.get(“/v1/alternatives”).toSet
      alternatives.foreach{alternative =>
          val byAlternative = rdd.filter(element => element.kind == alternative)
          val asRecords = byAlternative.map(element => asRecord(element))
          val conn = DB.connect(server)
          asRecords.foreachPartition{partition =>
              partition.foreach(element => conn.insert(element)
          }
      }
      rdd.unpersist(true)
}
```

Actions - foreachRDD Usage

Executes on the Driver

```
dstream.foreachRDD{rdd =>
      rdd.cache()
      val alternatives = restServer.get("/v1/alternatives").toSet
      alternatives.foreach{alternative =>
          val byAlternative = rdd.filter(element => element.kind == alternative)
          val asRecords = byAlternative.map(element => asRecord(element))
          val conn = DB.connect(server)
          asRecords.foreachPartition{partition =>
              partition.foreach(element => conn.insert(element))
          }
      }
      rdd.unpersist(true)
}
```

Executes on the Workers

Actions - foreachRDD Usage

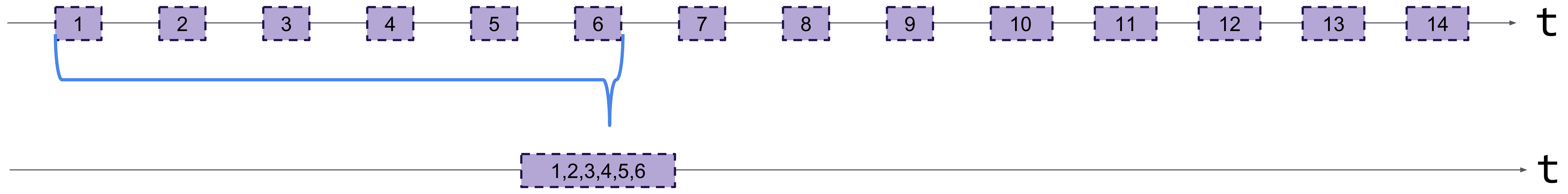
Executes on the Driver

```
dstream.foreachRDD{rdd =>
      rdd.cache()
      val alternatives = restServer.get(“/v1/alternatives”).toSet
      alternatives.foreach{alternative =>
          val byAlternative = rdd.filter(element => element.kind == alternative)
          val asRecords = byAlternative.map(element => asRecord(element))
          asRecords.foreachPartition{partition =>
              val conn = DB.connect(server)
              partition.foreach(element => conn.insert(element)
          }
      }
      rdd.unpersist(true)
}
```

Executes on the Workers

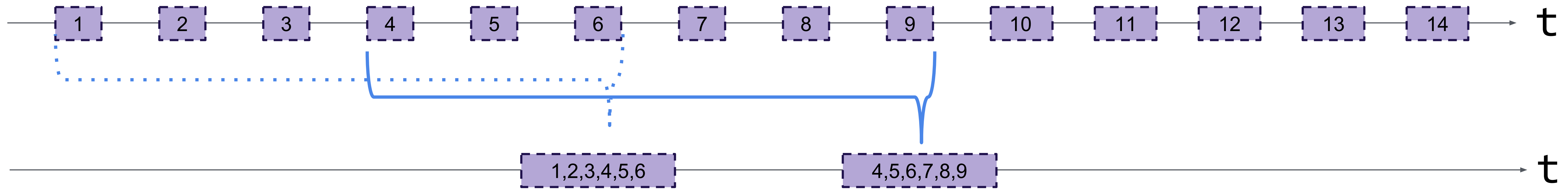
Windows - Sliding

```
dstream.window(windowLength = 6, slideInterval = 3)
```



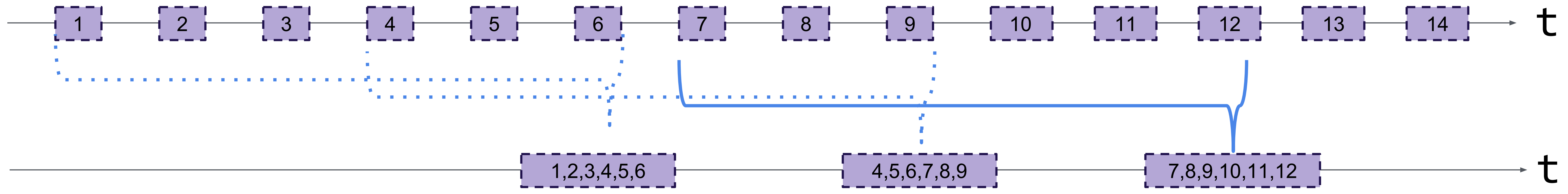
Windows - Sliding

```
dstream.window(windowLength = 6, slideInterval = 3)
```



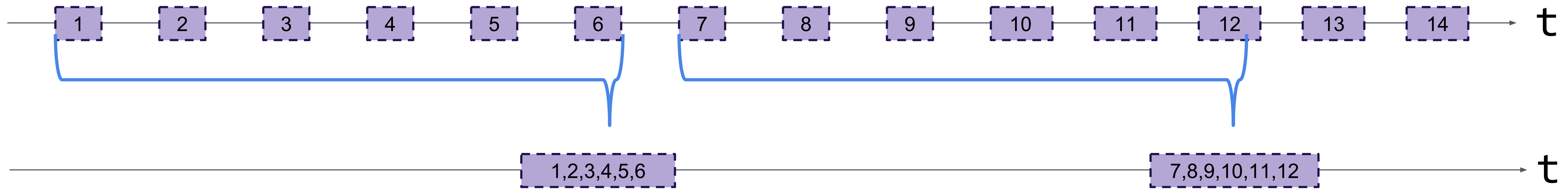
Windows - Sliding

```
dstream.window(windowLength = 6, slideInterval = 3)
```



Windows - Non-Overlapping

```
dstream.window(windowLength = 6, slideInterval = 6)
```

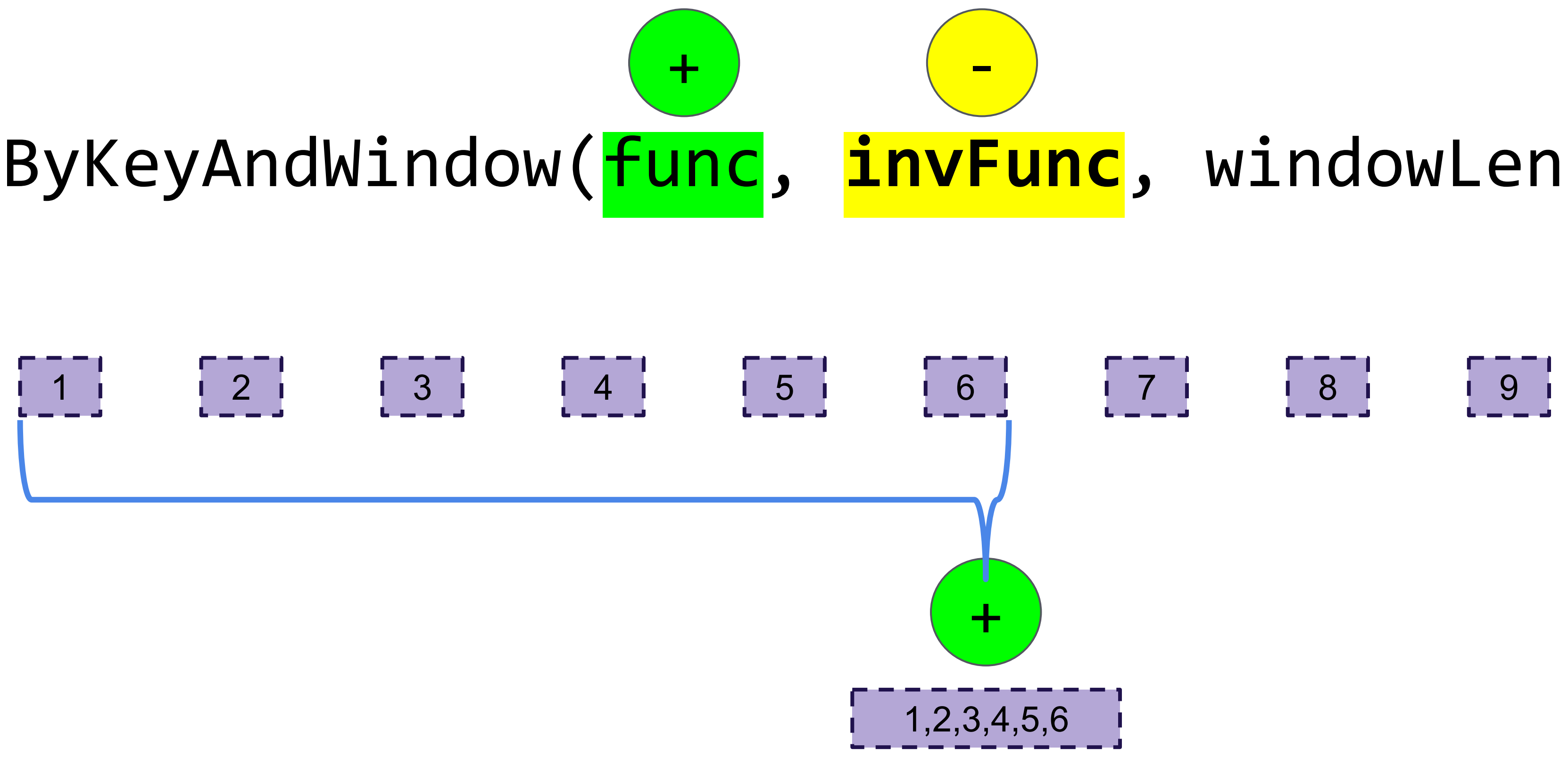


Windows - Operations

window,
countByWindow,
reduceByWindow,
reduceByKeyAndWindow,
countByValueAndWindow

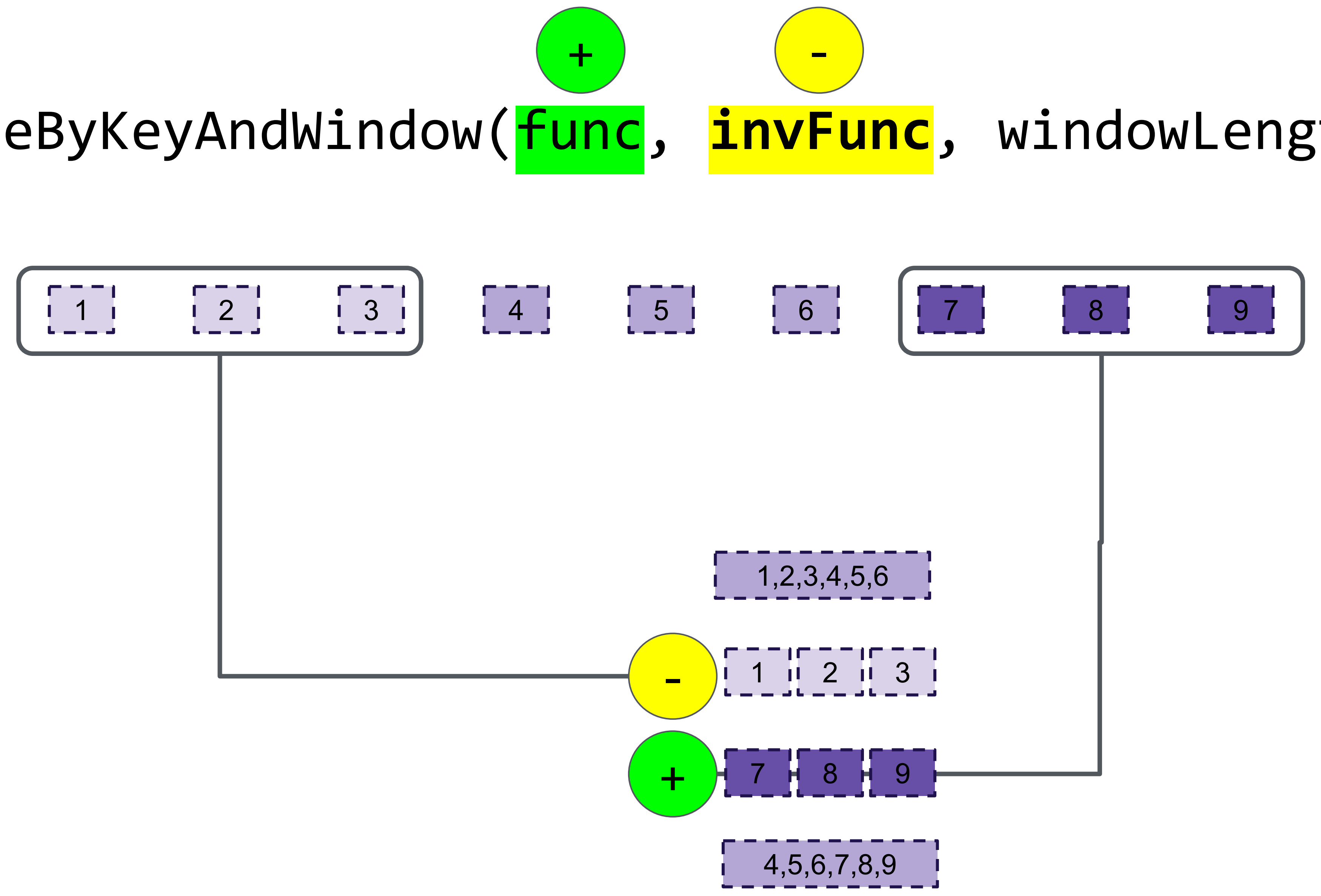
Windows - Inverse Function Optimization

```
reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval,[numTasks])
```



Windows- Inverse Function Optimization

```
reduceByKeyAndWindow(func, invFunc, windowLength, slideInterval,[numTasks])
```



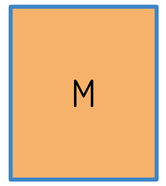
Demo 1

Anatomy of an Spark Streaming Application

Ready to dive in?

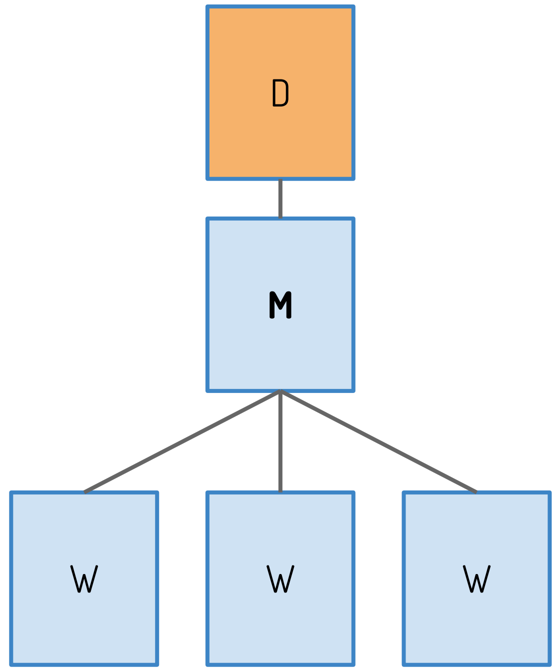
Deployment Options

Local



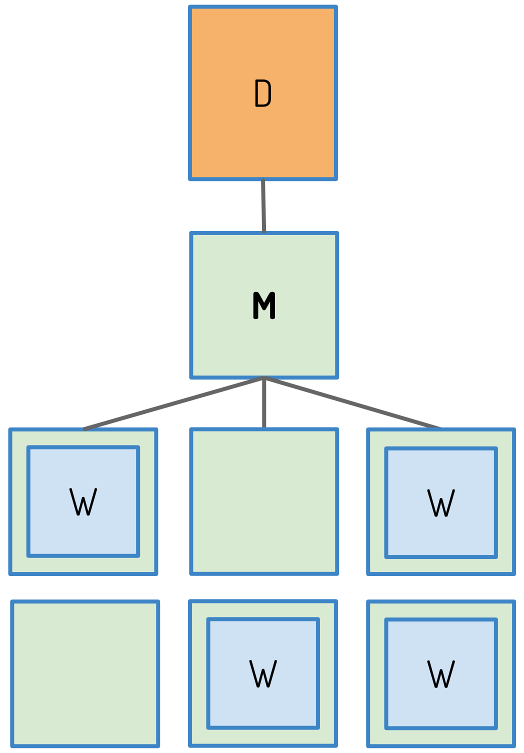
spark.master=local[*]

Standalone Cluster



spark.master=spark://host:port

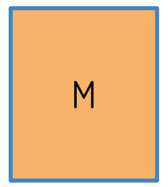
Using a Cluster Manager



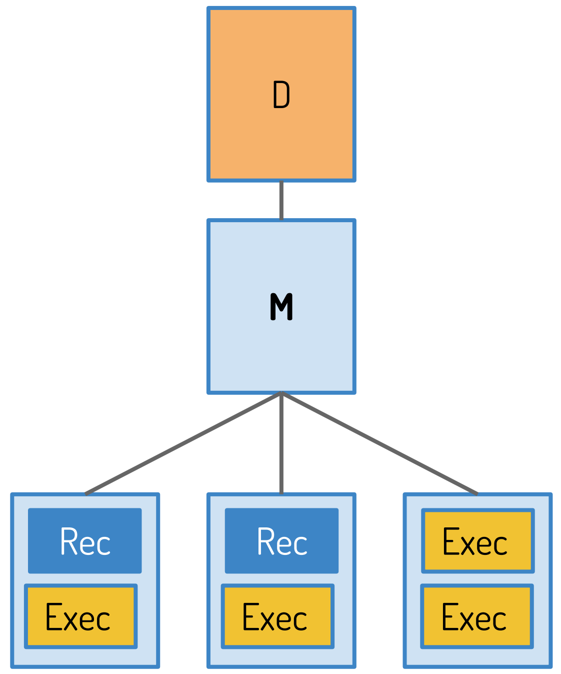
spark.master=mesos://host:port

Deployment Options

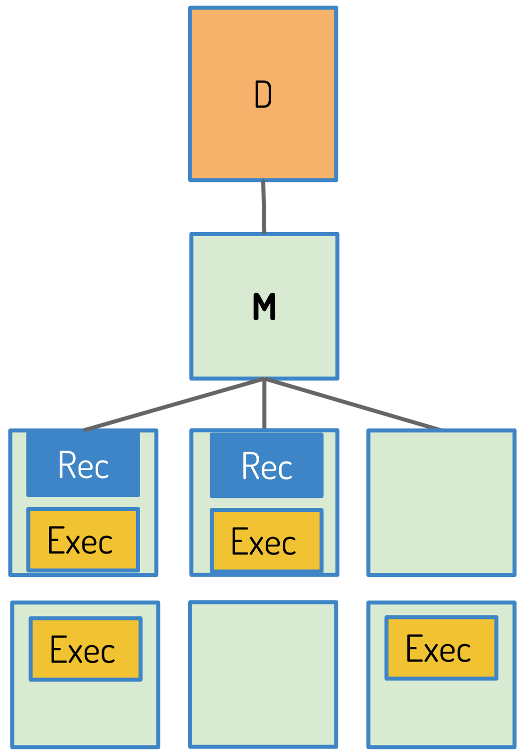
Local



Standalone Cluster



Using a Cluster Manager

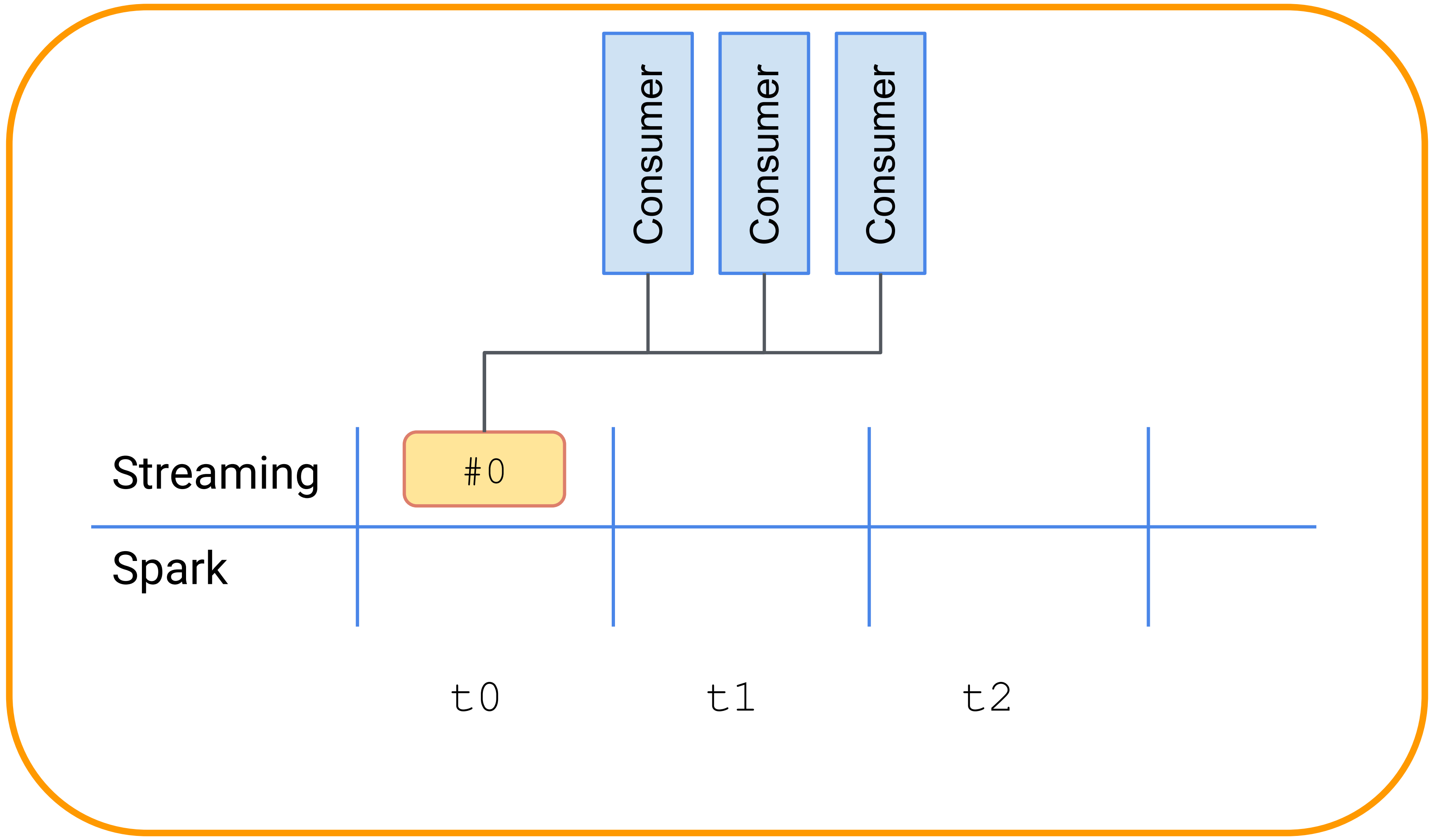


```
spark.master=local[*]
```

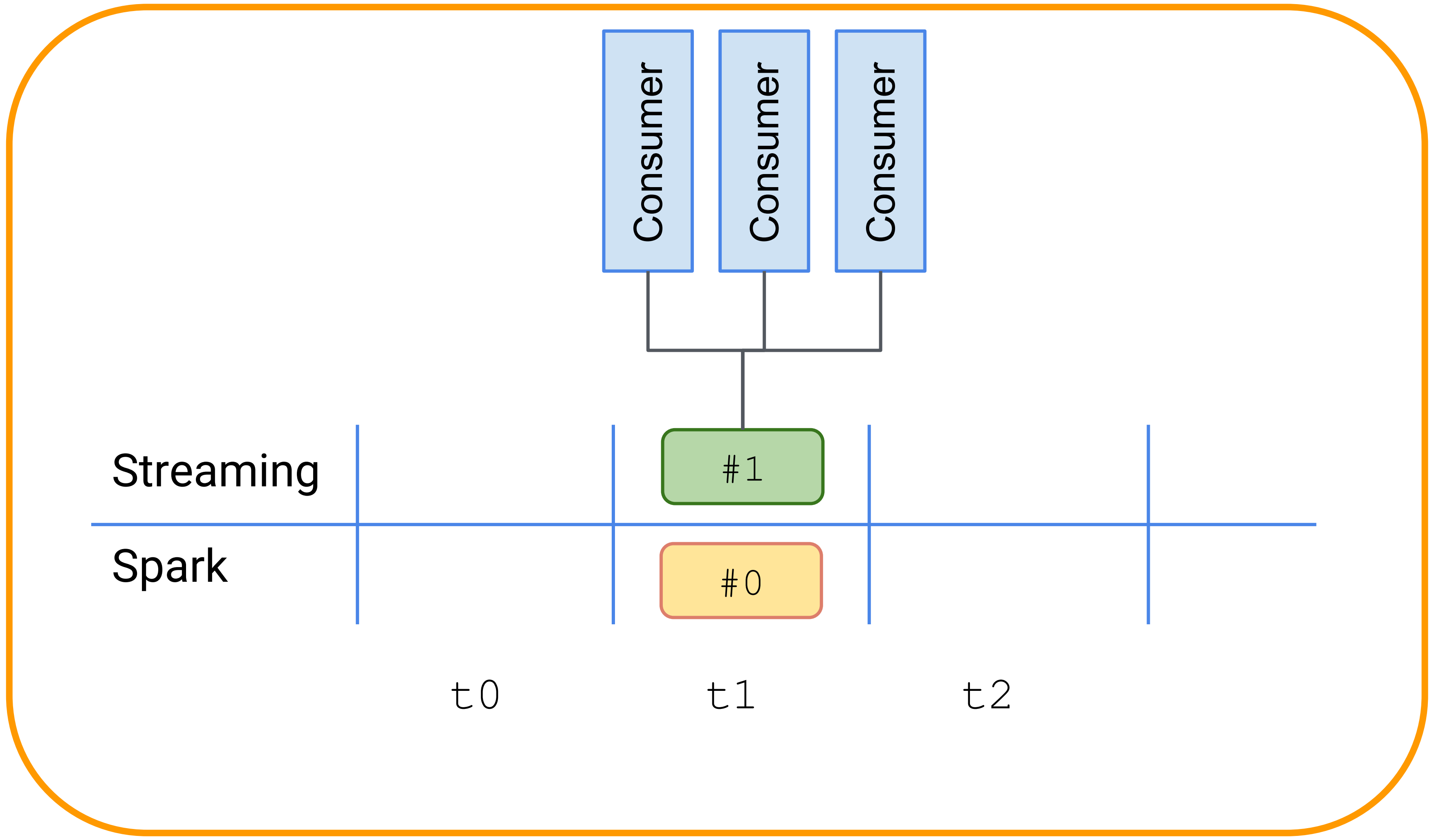
```
spark.master=spark://host:port
```

```
spark.master=mesos://host:port
```

Scheduling

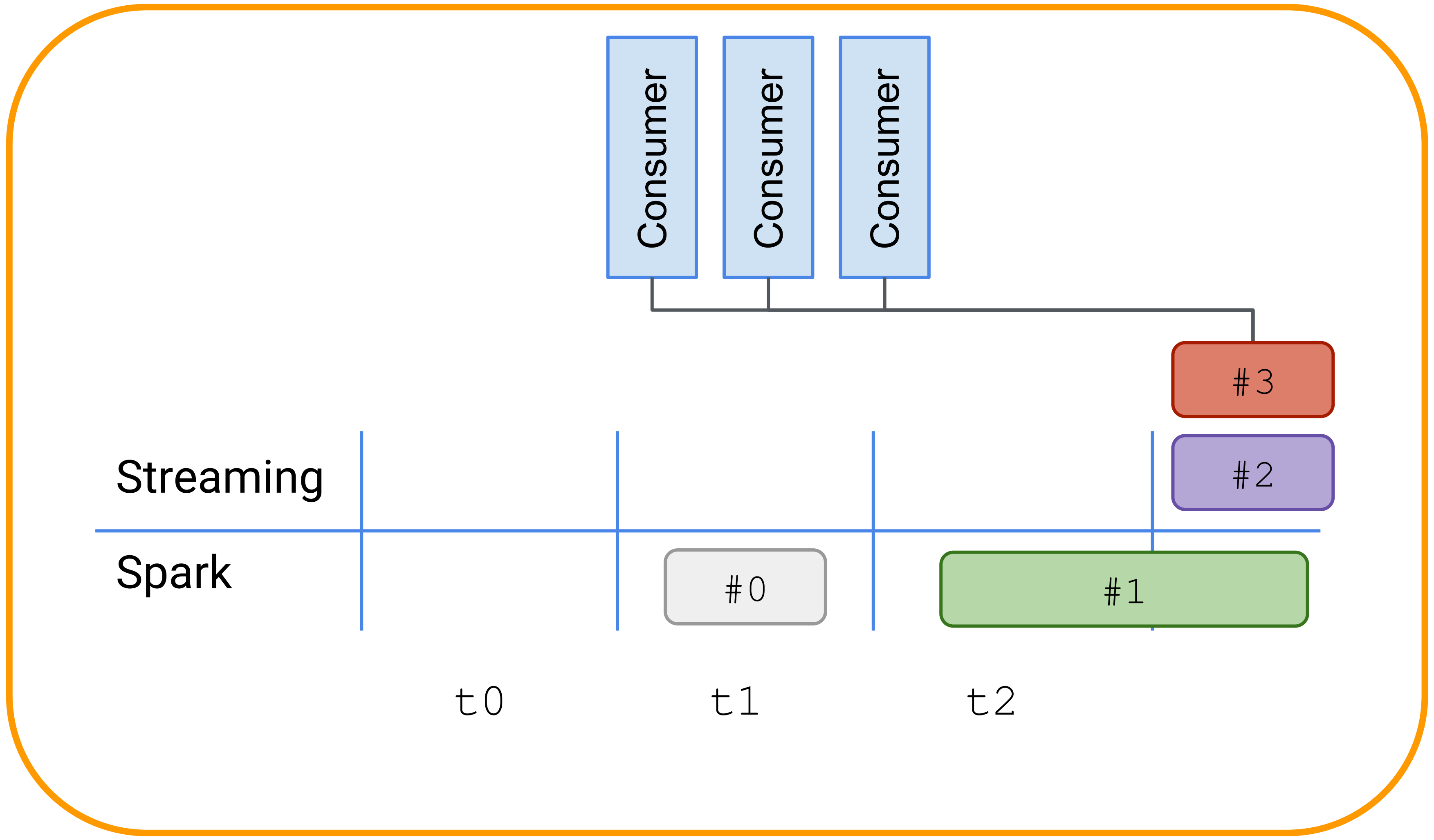


Scheduling



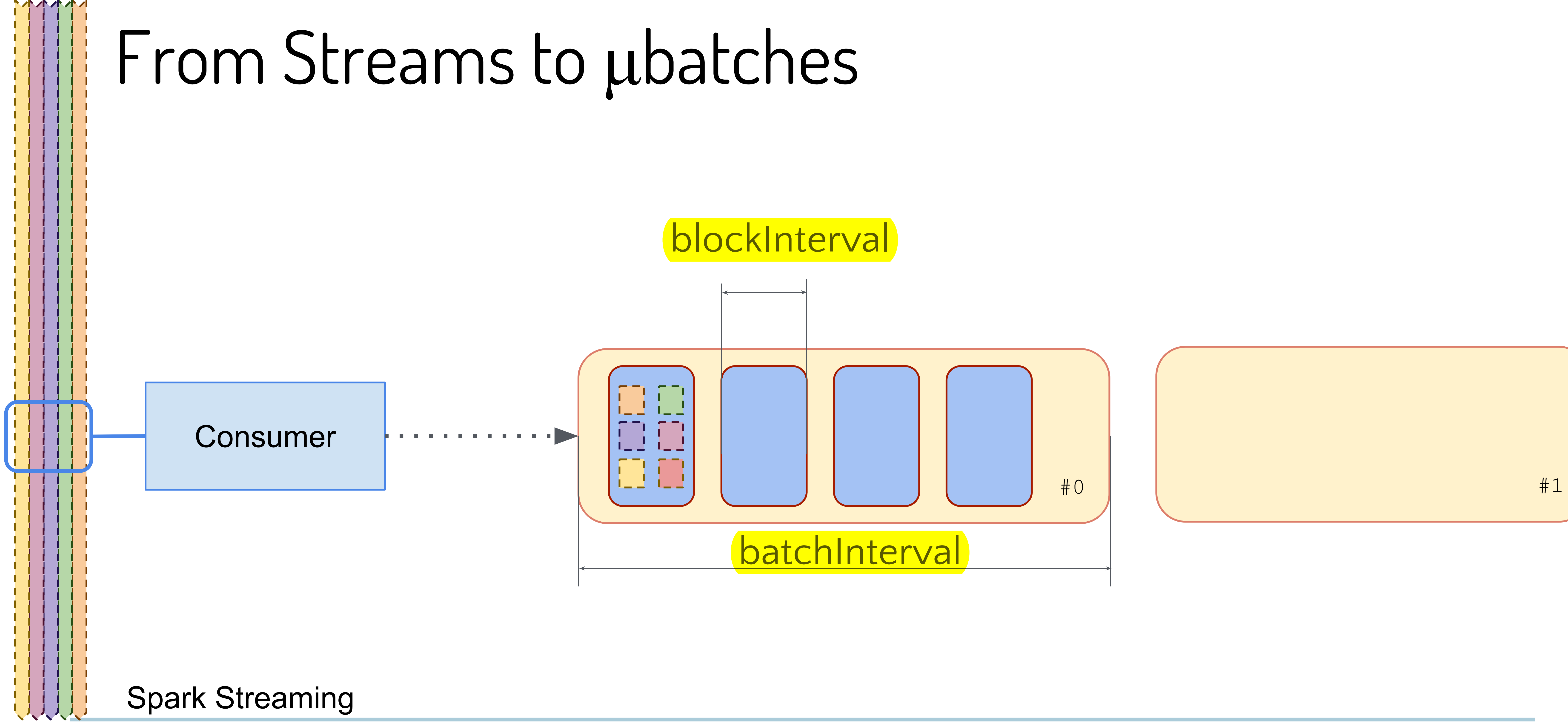
`Process Time < Batch Interval`

Scheduling



Scheduling Delay

From Streams to µbatches

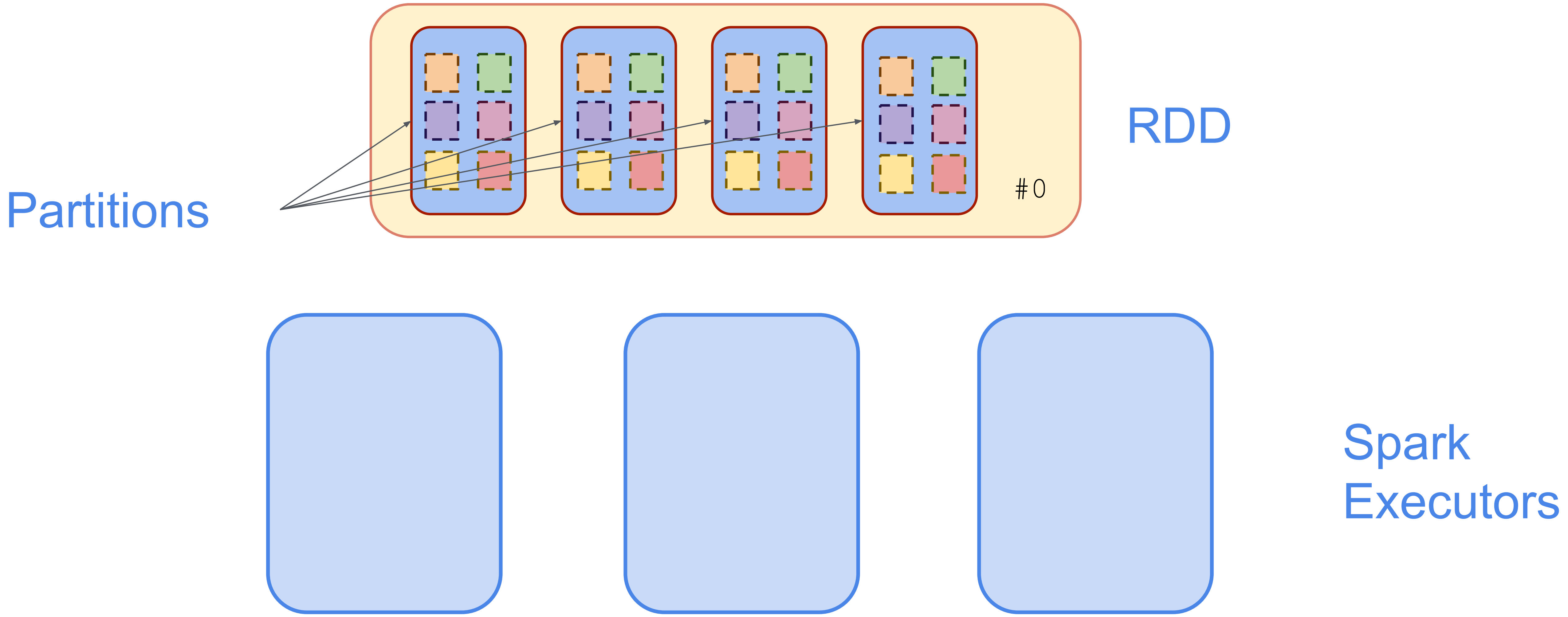


#partitions = receivers x batchInterval / blockInterval

From Streams to µbatches

Spark Streaming

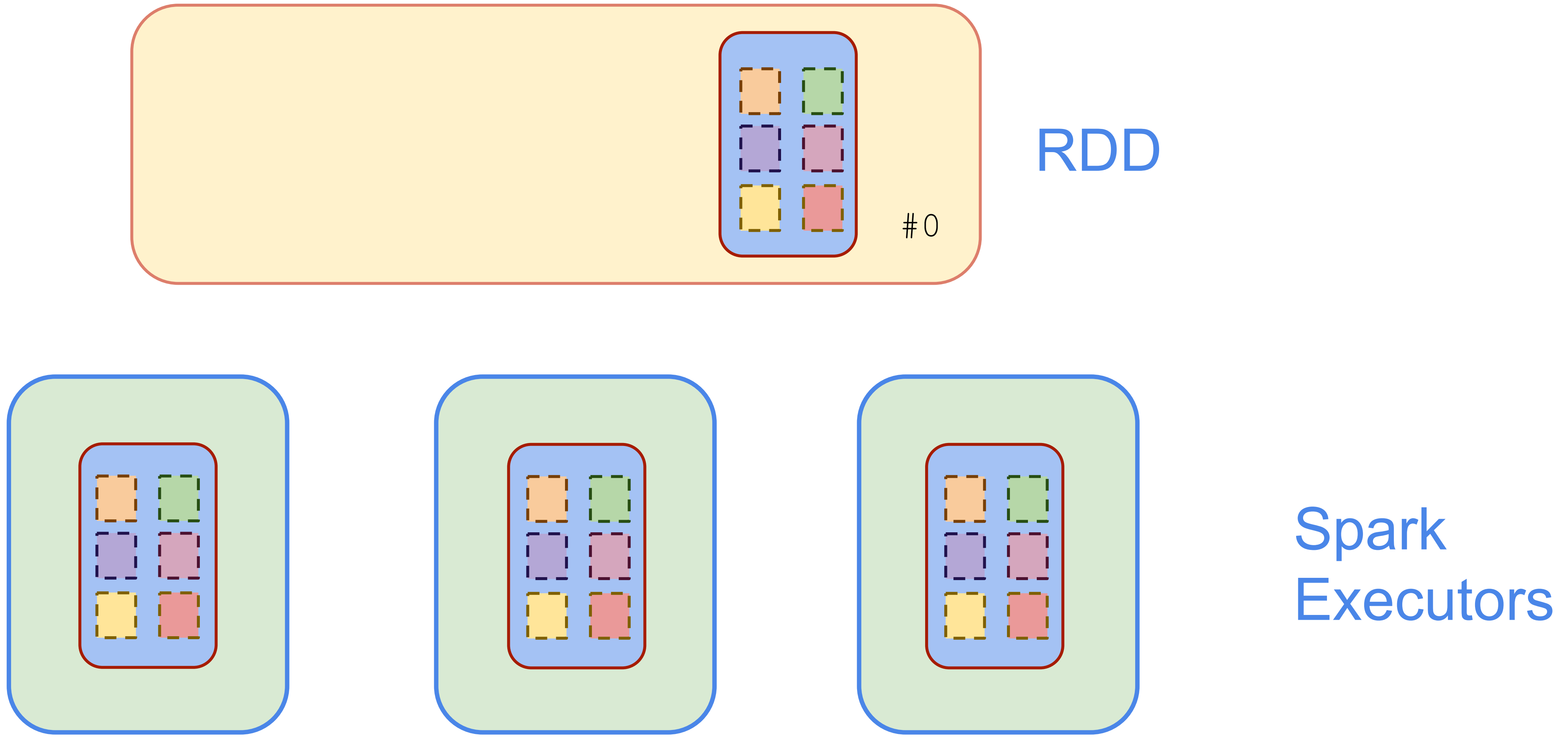
Spark



From Streams to µbatches

Spark Streaming

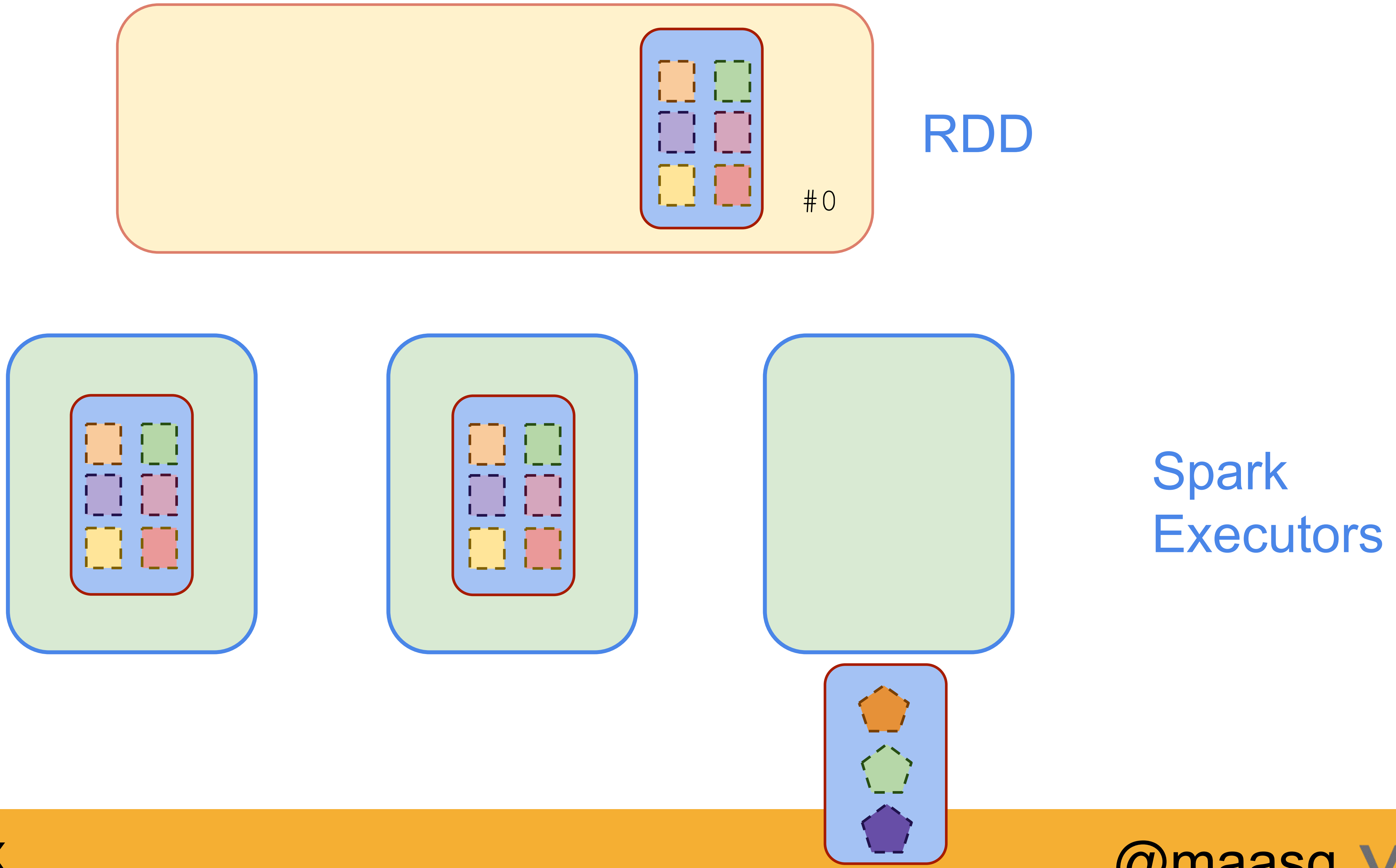
Spark



From Streams to µbatches

Spark Streaming

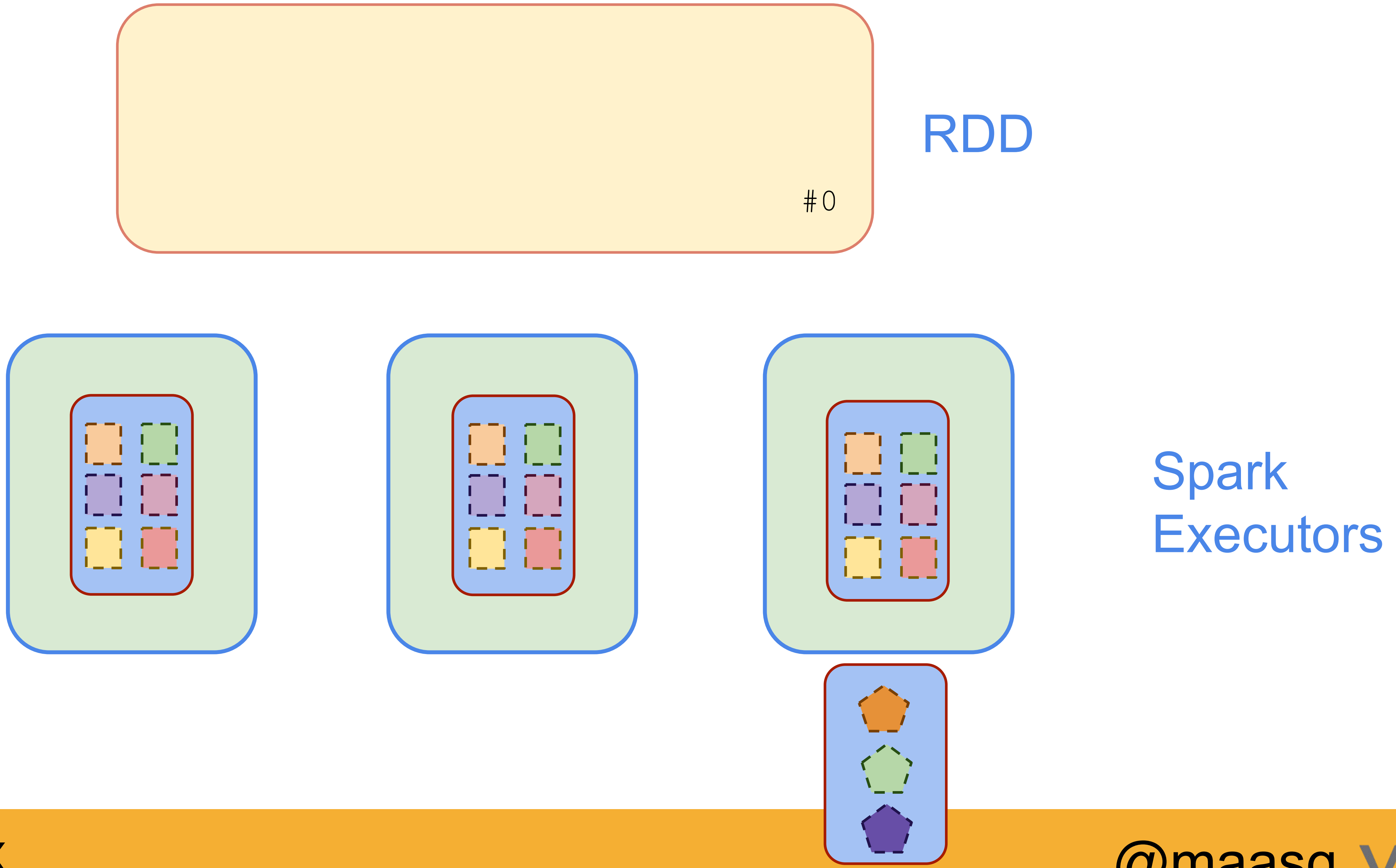
Spark



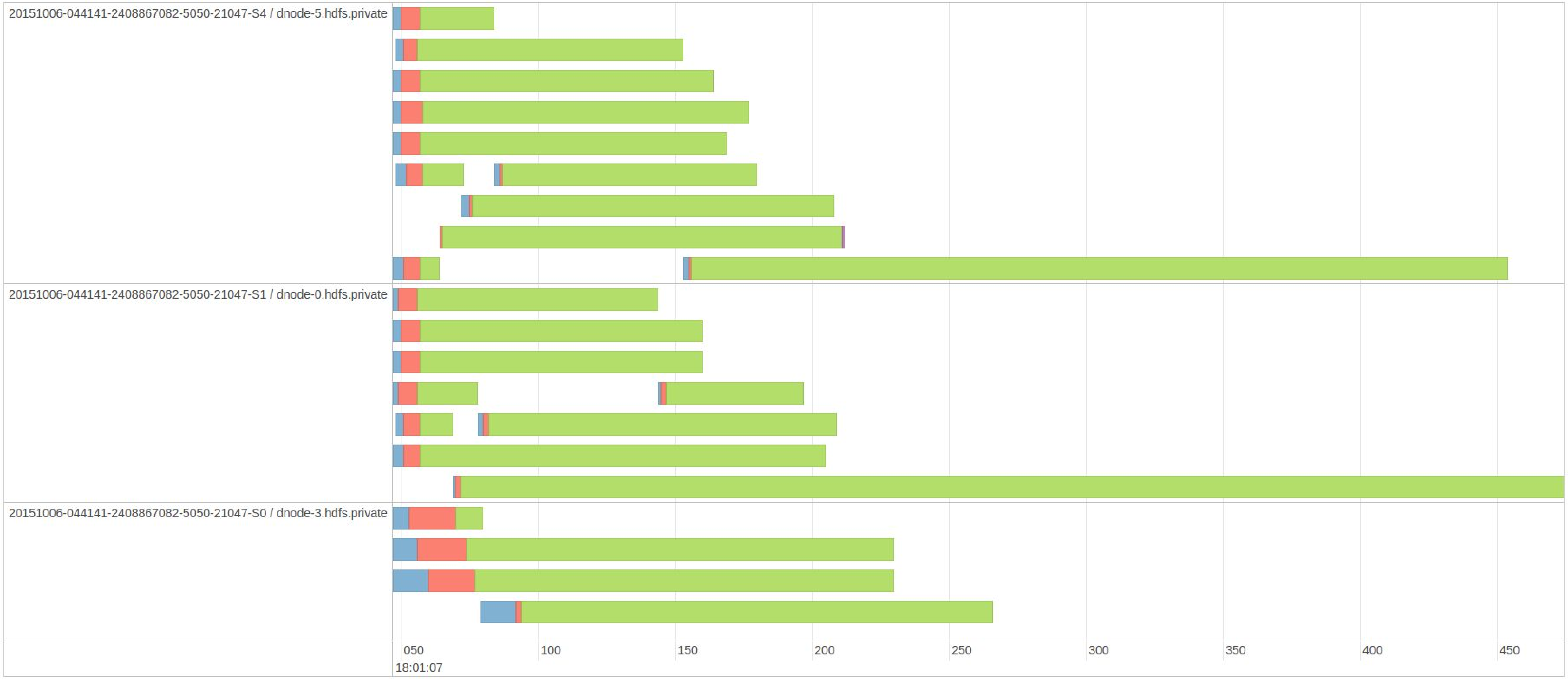
From Streams to μbatches

Spark Streaming

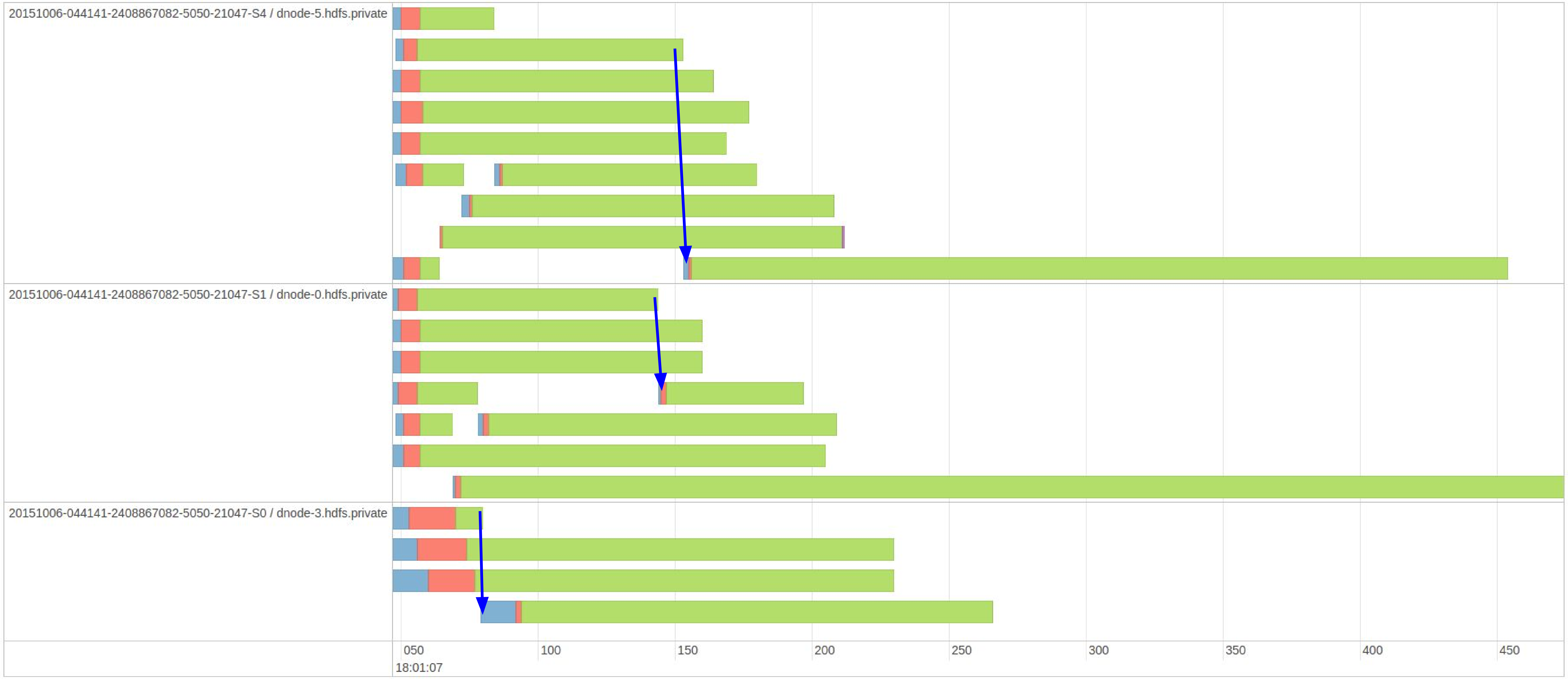
Spark



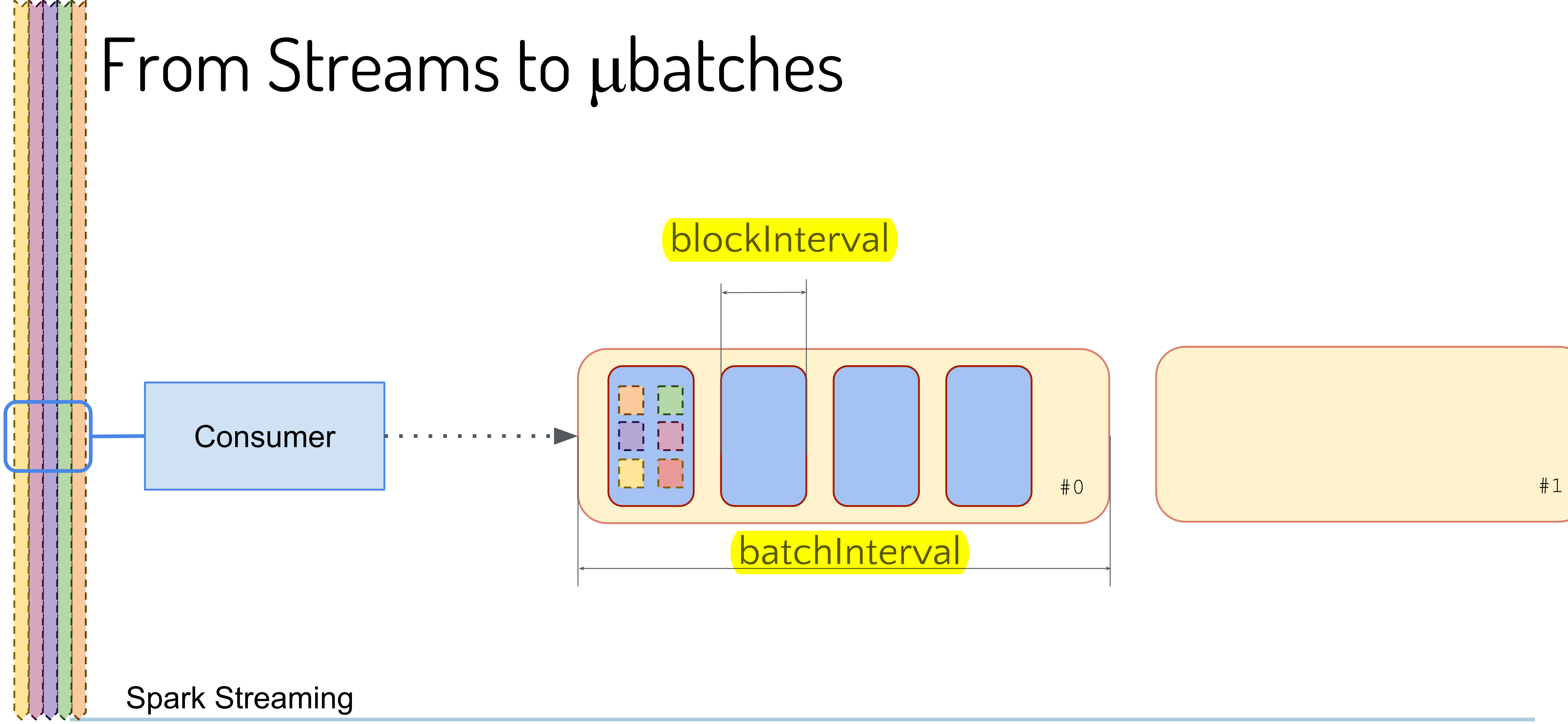
From Streams to µbatches



From Streams to µbatches

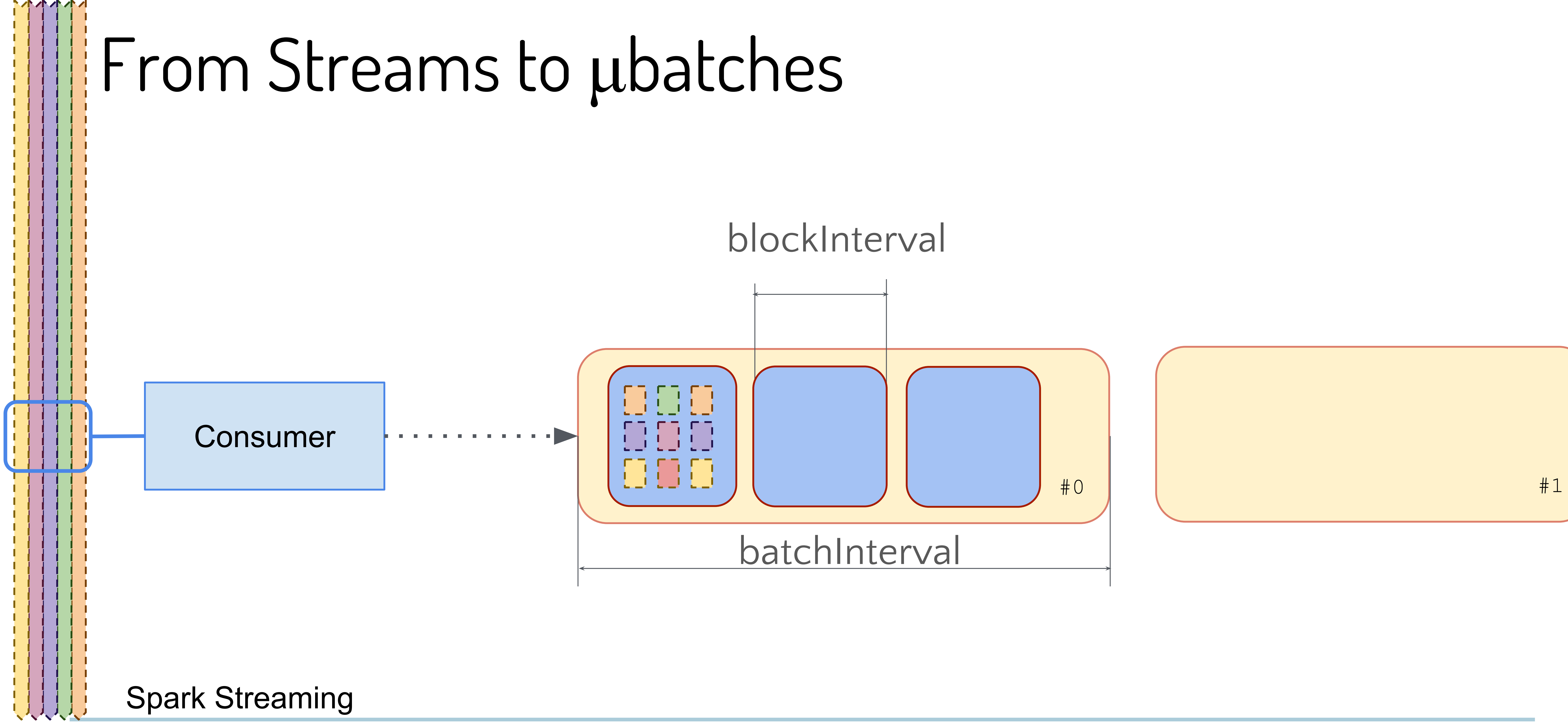


From Streams to µbatches



`#partitions = receivers x batchInterval / blockInterval`

From Streams to µbatches



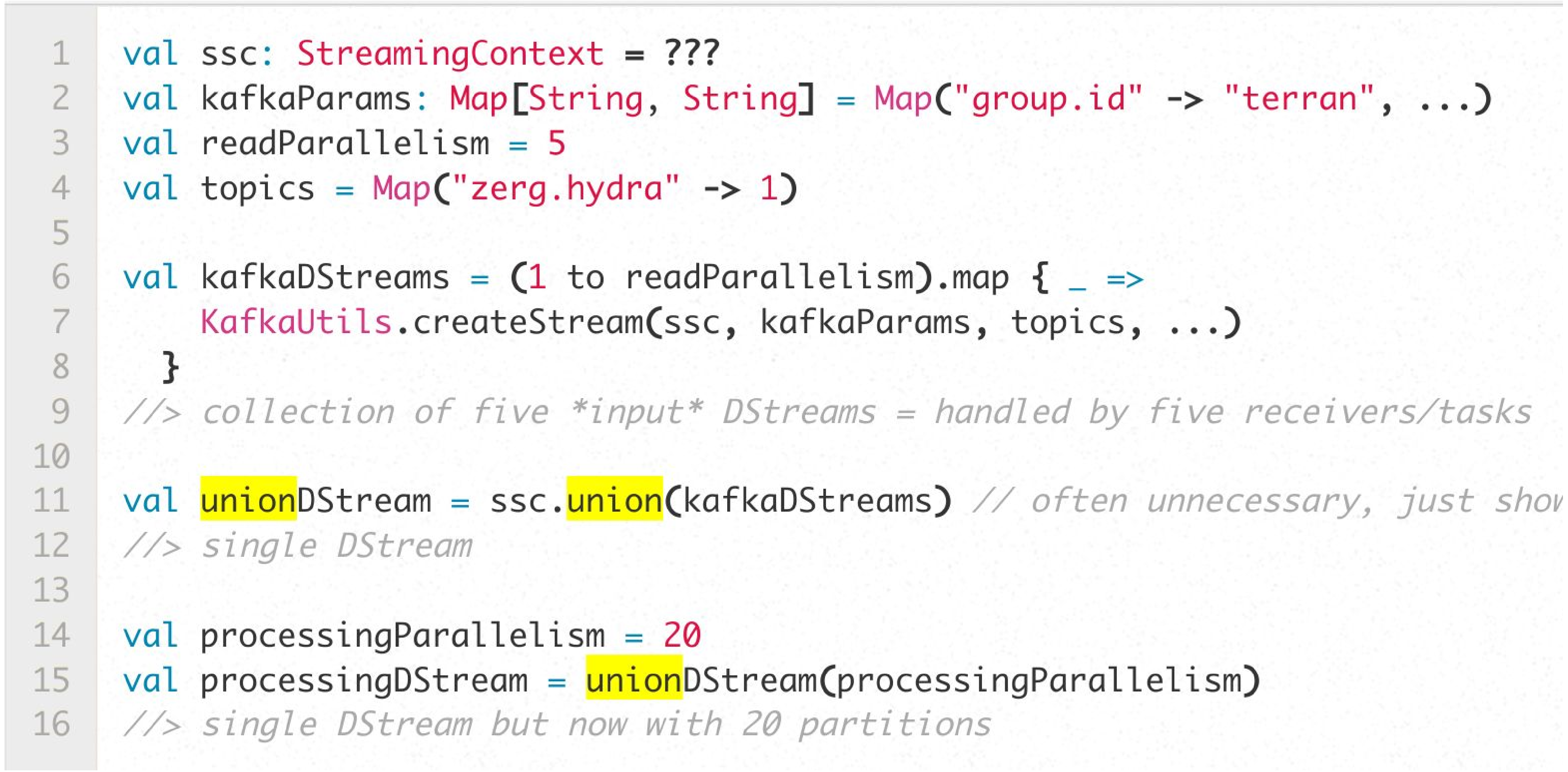
Spark

```
spark.streaming.blockInterval = batchInterval x
  receivers / (partitionFactor x sparkCores)
```

The Importance of Caching

```
dstream.foreachRDD { rdd  =>
    rdd.cache() // cache the RDD before iterating!


     keys.foreach{ key =>
      rdd.filter(elem=> key(elem) == key).saveAsFooBar(...)
    }


    rdd.unpersist()

}
```

The Receiver model

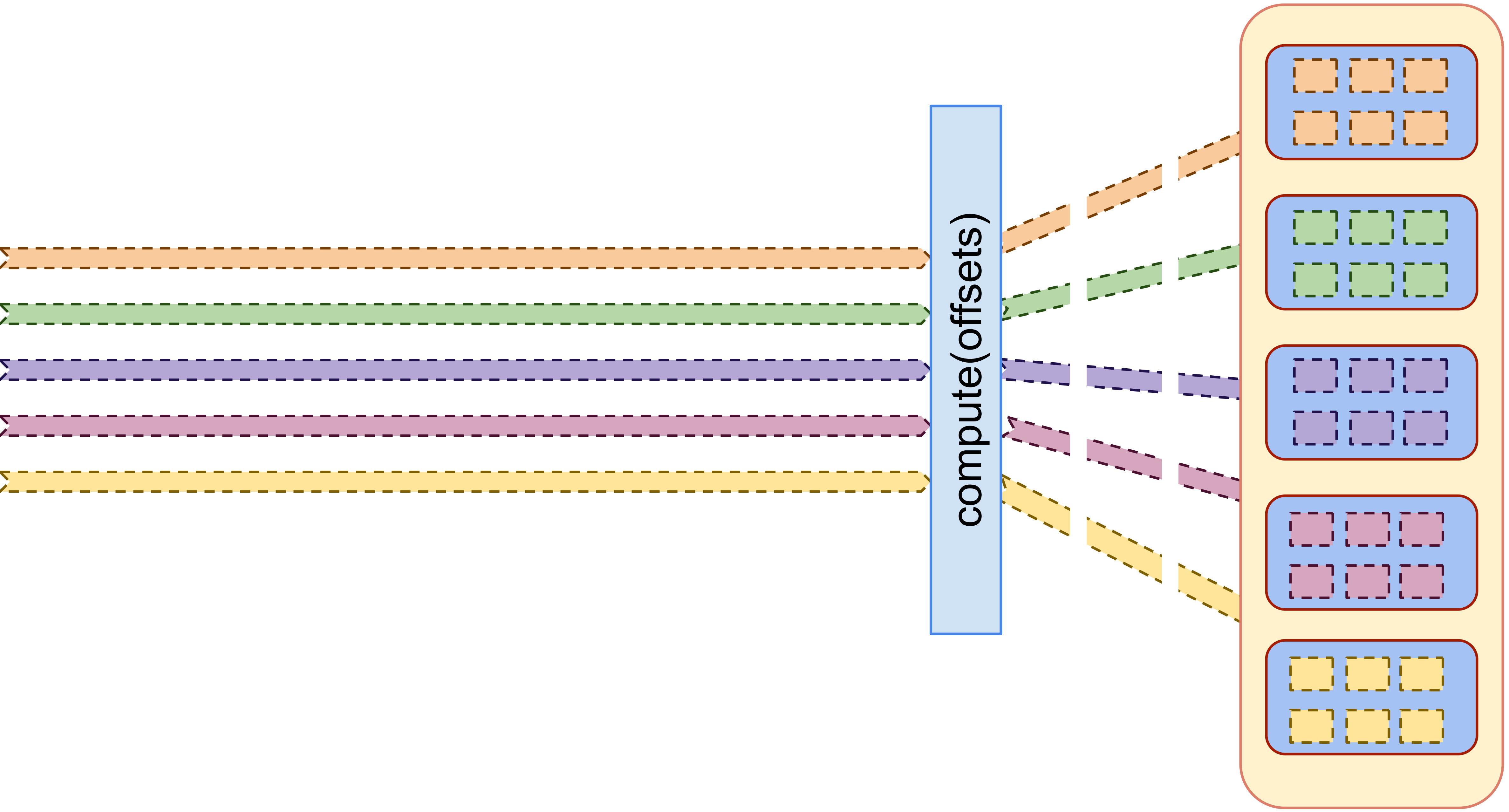


```scala
val ssc: StreamingContext = ???
val kafkaParams: Map[String, String] = Map("group.id" -> "terran", ...)
val readParallelism = 5
val topics = Map("zerg.hydra" -> 1)

val kafkaDStreams = (1 to readParallelism).map { _ =>
    KafkaUtils.createStream(ssc, kafkaParams, topics, ...)
  }
//> collection of five *input* DStreams = handled by five receivers/tasks

val unionDStream = ssc.union(kafkaDStreams) // often unnecessary, just show
//> single DStream

val processingParallelism = 20
val processingDStream = unionDStream(processingParallelism)
//> single DStream but now with 20 partitions
```

`spark.streaming.receiver.maxRate`

Fault tolerance ? WAL

Direct Kafka Stream



Kafka:The Receiver-less model

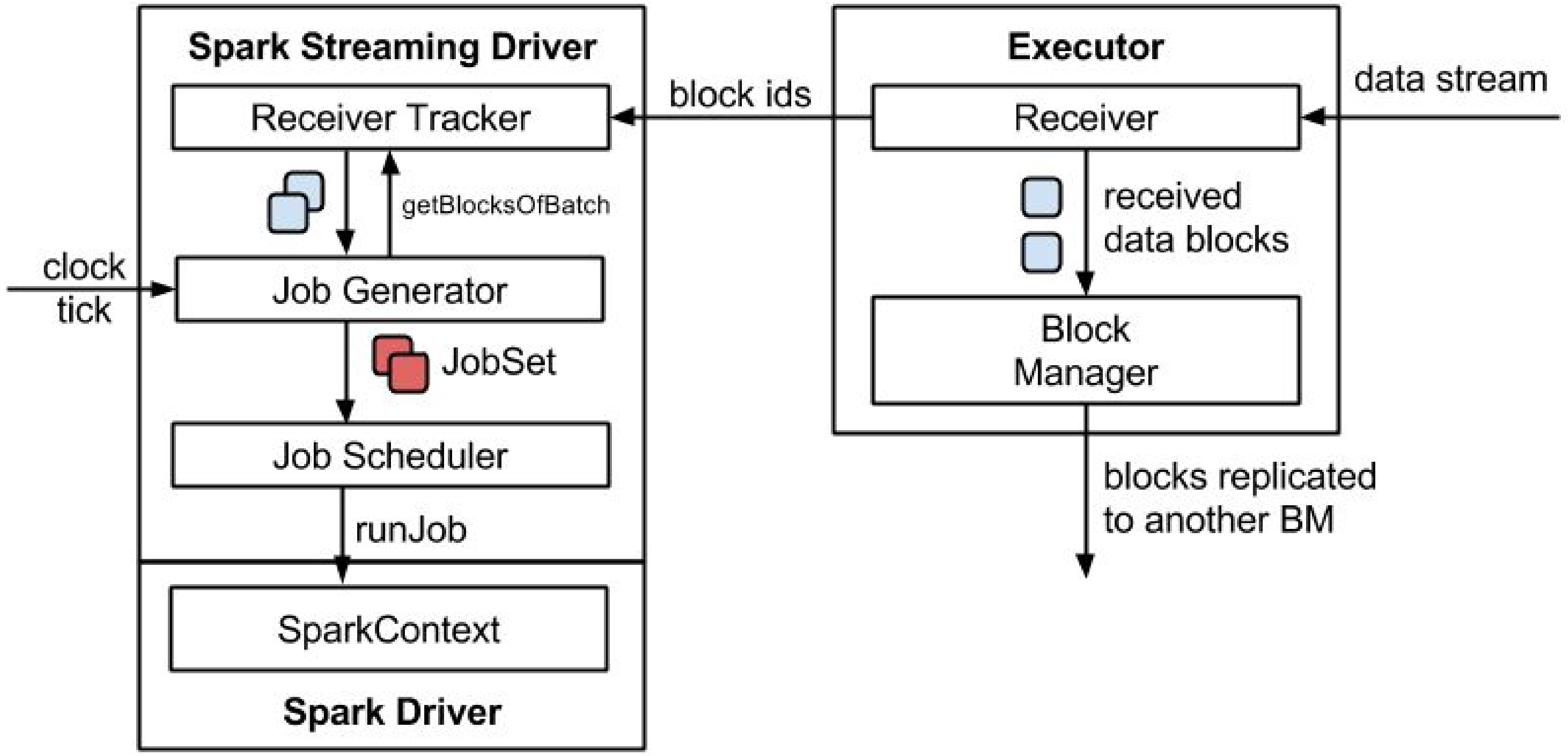
- *Simplified Parallelism*
- *Efficiency*
- *Exactly-once semantics*
- Less degrees of freedom

```
val directKafkaStream = KafkaUtils.
createDirectStream[
    [key class],
    [value class],
    [key decoder class],
    [value decoder class] ](
  streamingContext, [map of Kafka parameters], [set
of topics to consume]
)
```

`spark.streaming.kafka.maxRatePerPartition`

Spark Streaming (v1.5) made Reactive

Backpressure support



proportional-integral-derivative controller (PID controller)

Demo 2

Spark Streaming Performance

Resources

Spark Streaming Official Programming Guide:
http://spark.apache.org/docs/latest/streaming-programming-guide.html

Backpressure in Spark Streaming:
http://blog.garillot.net/post/121183250481/a-quick-update-on-spark-streaming-work-since-i

The Virdata's Spark Streaming tuning guide:
http://www.virdata.com/tuning-spark/

Spark Summit Presentations:
https://spark-summit.org/

Diving into Spark Streaming Execution Model:
https://databricks.com/blog/2015/07/30/diving-into-spark-streamings-execution-model.html

Kafka direct approach:
https://github.com/koeninger/kafka-exactly-once/blob/master/blogpost.md

Questions?

Thanks!

Gerard Maas

@maasg

www.virdata.com

– we're hiring –