DStreams

Spark Streaming

Let's look at some code for a Spark Streaming application

DStreams

The objective:

Spark Streaming

Monitor all text data that arrives at a certain port

DStreams

Spark Streaming

The objective:

Monitor all text data that arrives at a certain port

Filter any lines that contain the word "ERROR"

DStreams

Spark Streaming

The objective:

Listen to all text data that arrives at a certain port

Filter any lines that contain the word "ERROR"

Print a count to screen

Spark Streaming

```scala
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
val conf= new SparkConf().setMaster(“yarn-client").
                          setAppName("My App")

val sc= new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(1))

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

We’ll need to set up the SparkContext and a StreamingContext

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
val conf= new SparkConf().setMaster(“yarn-client").
                          setAppName("My App")

val sc= new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(1))


    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

A StreamingContext is used to create DStreams

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
val conf= new SparkConf().setMaster(“yarn-client").
                          setAppName("My App")

val sc= new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(1))

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

The StreamingContext is where you set the batch interval

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
val conf= new SparkConf().setMaster(“yarn-client").
                          setAppName("My App")

val sc= new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(1))

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

The batch interval is set to 1 second

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
val conf= new SparkConf().setMaster(“yarn-client").
                          setAppName("My App")

val sc= new SparkContext(conf)
val ssc = new StreamingContext(sc, Seconds(1))

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

This will be the batch interval for all DStreams we create using this StreamingContext

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

This is some additional setup we need for fault-tolerance

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val
        new StreamingContext(sc, Seconds(2))

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")

        val lines = ssc.socketTextStream(args(0), args(1).toInt)

    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

It sets a specified location where a backup of the data is saved at a certain frequency

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error"))
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

Create the DStream

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

A DStream that reads text data which arrives at a specified port

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

The hostname where the data will arrive

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey( + )
    counts.print()

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

The port where the data arrives

Spark Streaming

```
> spark-submit --class <className>
<JAR path> localhost 9999
```

```
object Streaming {

  def main (args: Array[String]){
    val conf = new SparkConf().setMaster("yarn-cluster").setAppName("New App")
    val sc= new SparkContext(conf)
    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
     val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+
    counts.print()

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

Both these arguments will be passed at the commandline when we submit this script

Spark Streaming

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
    reduceByKey(_+_)
```

Apply any transformations and actions on the DStream

Spark Streaming

```
val counts = lines.filter(_.contains("Error")).
flatMap(_.split(" ")).map( word => (word, 1)).
reduceByKey(_+_)
```

This looks as if the operations are applied on the entire DStream

Internally, the operations are applied to each individual RDD in the DStream

Spark Streaming

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
    reduceByKey(_+_)
```

Filter for lines that have the substring "ERROR"

Spark Streaming

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
    reduceByKey(_+_)
```

flatMap converts an RDD with lines into an RDD with words

Spark Streaming

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
    reduceByKey(_+_)
```

Each word is mapped to (word, 1)

Spark Streaming

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
    reduceByKey(_+_)
```

Returns (word, count)

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).

      reduceByKey(_+_)
```

```
counts.print()
```

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

Prints the count to screen

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).

      reduceByKey(_+_)

counts.print()

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

This method is called for each individual RDD in the DStream

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).

      reduceByKey(_+_)
```

```
counts.print()
```

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

Since batch interval = 1 s, an RDD is created every second

```
def main (args: Array[String]){
  val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
  val sc= new SparkContext(conf)

  val ssc = new StreamingContext(sc, Seconds(1))
  ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
  val lines = ssc.socketTextStream(args(0), args(1).toInt)
  val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
    reduceByKey(_+_)
  counts.print()
```

Spark Streaming

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

Finally, we have to actually tell the StreamingContext to start listening for Streaming Data

Streaming.scala

Spark Streaming

```scala
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

To run this code, first start a stream at your localhost using the netcat utility

```
> nc -lk 9999
```

Streaming.scala

Spark Streaming

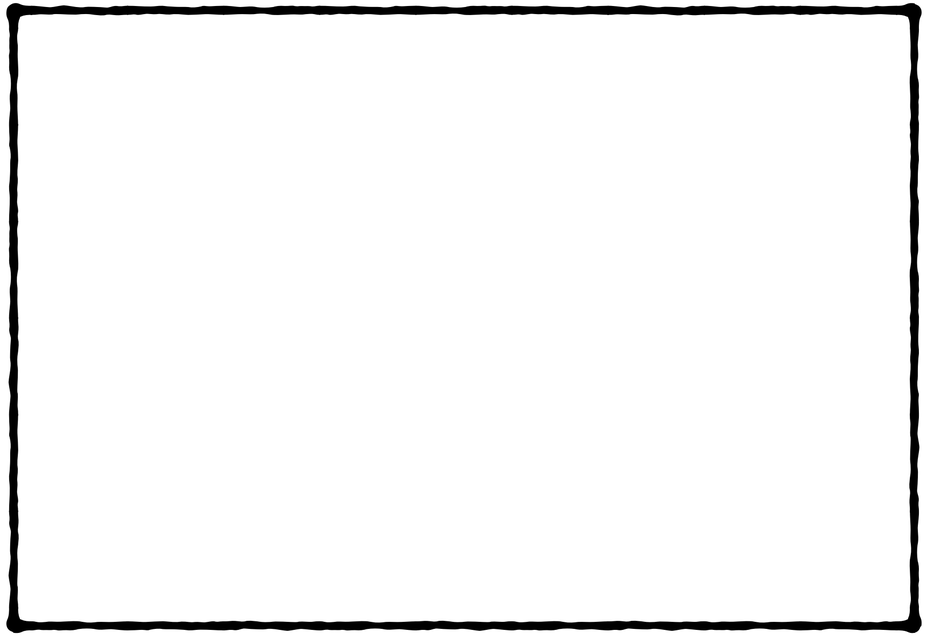
```scala
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

```
> nc -lk 9999
```



In a separate terminal submit your script

```
> spark-submit --class <className>
<JAR path> localhost 9999
```

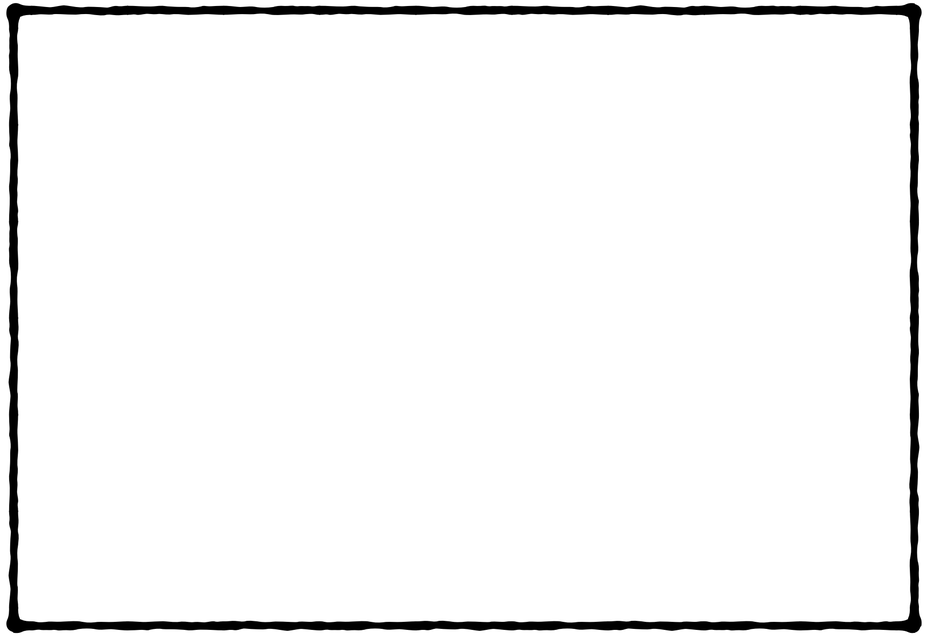
Streaming.scala

Spark Streaming

```
> spark-submit --class <className> <JAR path> localhost 9999
```

Spark will start listening for the streaming data at the 9999 port

```
> nc -lk 9999
```



Streaming.scala

Spark Streaming

```
> spark-submit --class <className> <JAR path> localhost 9999
```

```
-------------------------------------------
Time: 2016-06-24 01:27:15
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:27:16
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:27:17
-------------------------------------------
```

The batch interval is 1 second

```
> nc -lk 9999
```

Streaming.scala

Spark Streaming

```
> spark-submit --class <className> <JAR path> localhost 9999
```

```
-------------------------------------------
Time: 2016-06-24 01:27:15
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:27:16
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:27:17
-------------------------------------------
```

```
> nc -lk 9999
```

Anything typed here will be processed by Spark

Streaming.scala

Spark Streaming

```
> spark-submit --class <className> <JAR path> localhost 9999
```

```
-------------------------------------------
Time: 2016-06-24 01:27:15
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:27:16
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:27:17
-------------------------------------------


-------------------------------------------
Time: 2016-06-24 01:37:00
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:37:01
-------------------------------------------
(u'ERROR', 1)
```

```
> nc -lk 9999
```

```
This line won't be printed

This line has ERROR
```

Streaming.scala

Spark Streaming

```
> spark-submit --class <className> <JAR path> localhost 9999
```

```
-------------------------------------------
Time: 2016-06-24 01:37:00
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:37:01
-------------------------------------------
(u'ERROR', 1)

-------------------------------------------
Time: 2016-06-24 01:37:06
-------------------------------------------
(u'ERROR', 1)
```

```
> nc -lk 9999
```

```
This line won't be printed

This line has ERROR

This line has ERROR too
```

We typed each sentence after a couple of seconds

Streaming.scala

Spark Streaming

```
> spark-submit --class <className> <JAR path> localhost 9999
```

```
-------------------------------------------
Time: 2016-06-24 01:37:00
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:37:01
-------------------------------------------
(u'ERROR', 1)
-------------------------------------------
Time: 2016-06-24 01:37:06
-------------------------------------------
(u'ERROR', 1)
```

```
> nc -lk 9999
```

```
This line won't be printed
This line has ERROR
This line has ERROR too
```

Each line went into a separate RDD

Streaming.scala

Spark Streaming

```
> spark-submit --class <className> <JAR path> localhost 9999
```

```
-------------------------------------------
Time: 2016-06-24 01:37:00
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:37:01
-------------------------------------------
(u'ERROR', 1)
-------------------------------------------
Time: 2016-06-24 01:37:06
-------------------------------------------
(u'ERROR', 1)
```

```
> nc -lk 9999
```

```
This line won't be printed

This line has ERROR

This line has ERROR too
```

Each line got processed individually

Streaming.scala

```
-------------------------------------------
Time: 2016-06-24 01:37:00
-------------------------------------------
-------------------------------------------
Time: 2016-06-24 01:37:01
-------------------------------------------
(u'ERROR', 1)
-------------------------------------------
Time: 2016-06-24 01:37:06
-------------------------------------------
(u'ERROR', 1)
```

The count is from a single RDD, **not** accumulated across RDDs

DStreams have 2 types of transformations

Stateless — Regular transformations like map, reduceBykey etc

Stateful — reduceByWindow, reduceByKeyAndWindow etc

Stateless

Regular transformations like map, reduceBykey etc

These transformations apply on each individual RDD in the DStream

Stateless

Regular transformations

Map, reduceBykey etc

These are used to accumulate results across the RDDs in the DStream

Stateful

reduceByWindow,
reduceByKeyAndWindow
etc

Stateful transformations depend upon a Sliding Window

Stateful

reduceByWindow,
reduceByKeyAndWindow
etc

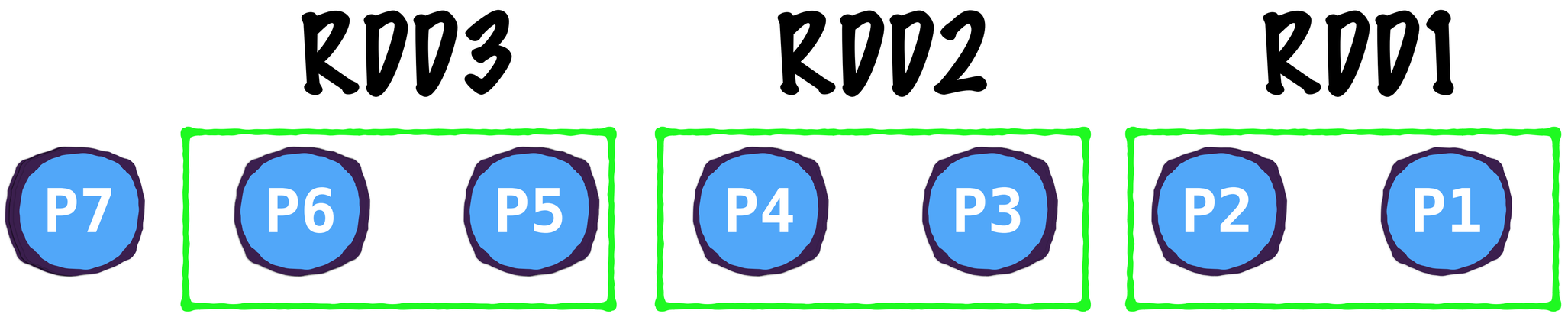
A Sliding Window consists of multiple RDDs in the DStream

Stateful

reduceByWindow,
reduceByKeyAndWindow
etc

Sliding Window

Spark Streaming

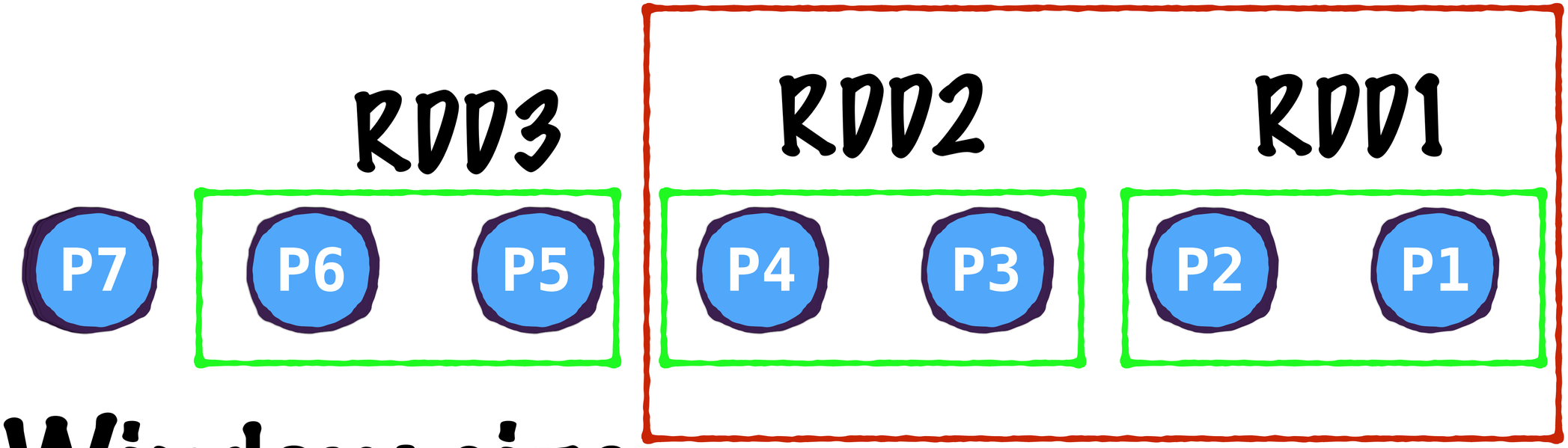


Let's say the batch interval is 1 sec

The stream consists of 2 logs per second

Sliding Window

Spark Streaming

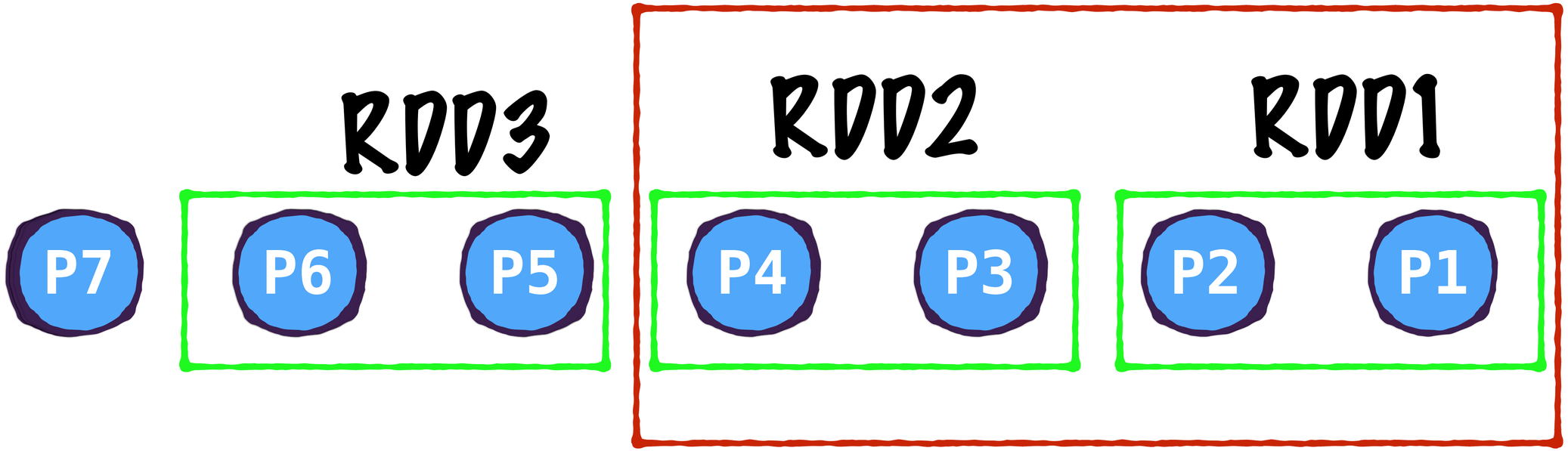


Window size is 2 seconds

Sliding interval is 1 second

Sliding Window

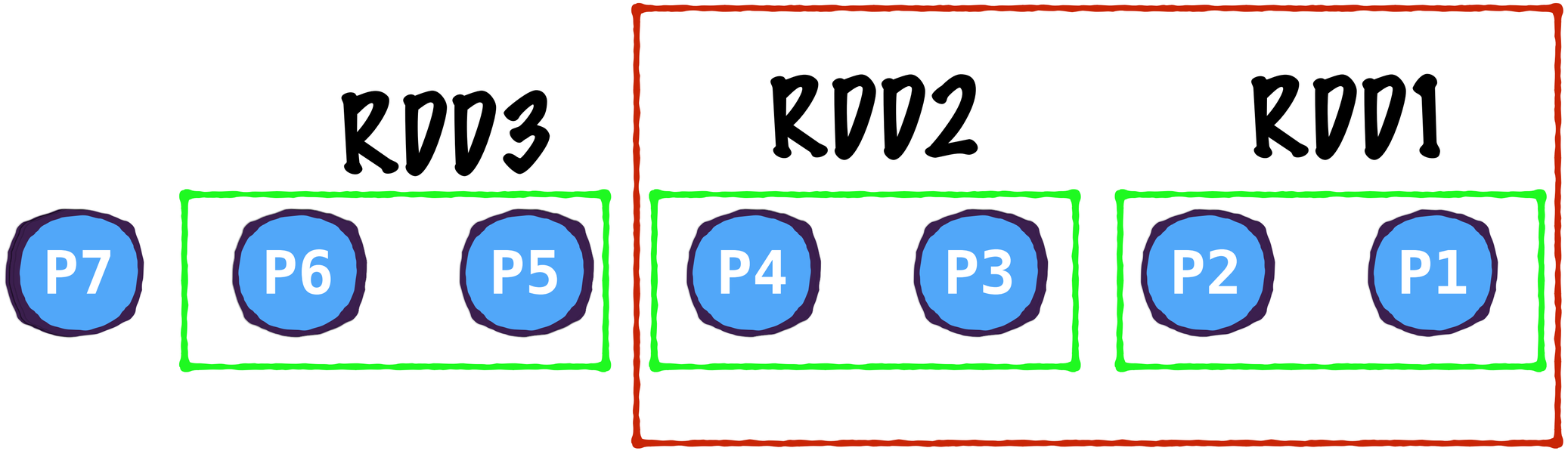
Spark Streaming



Window at t = 2s

Sliding Window

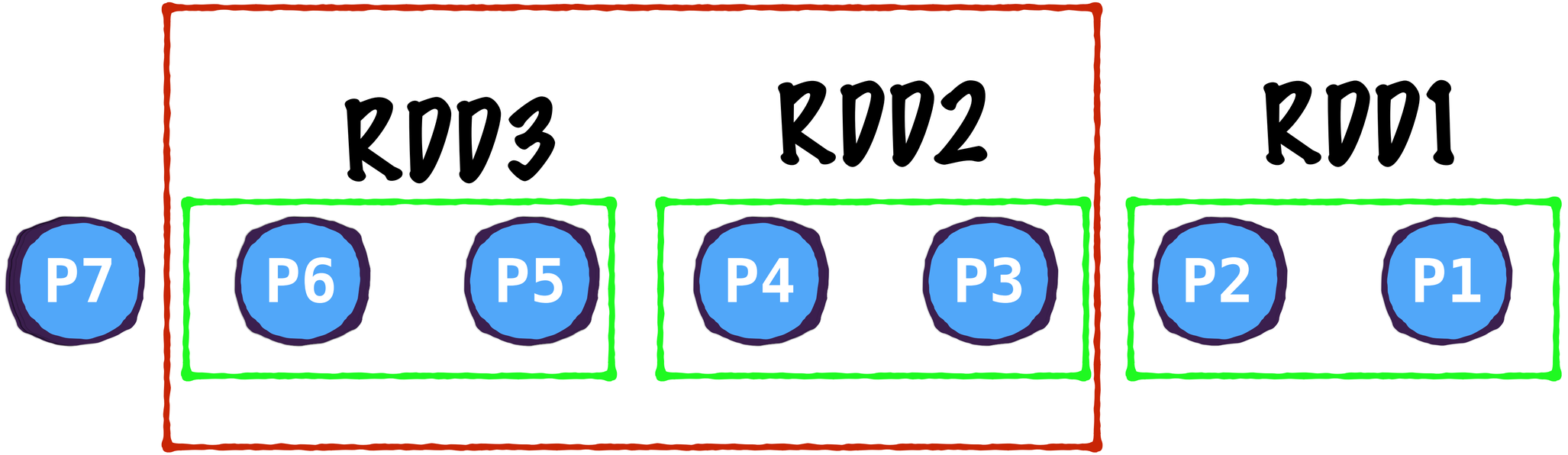
Spark Streaming



A transformation on this window will treat both RDDs as a combined RDD

Sliding Window

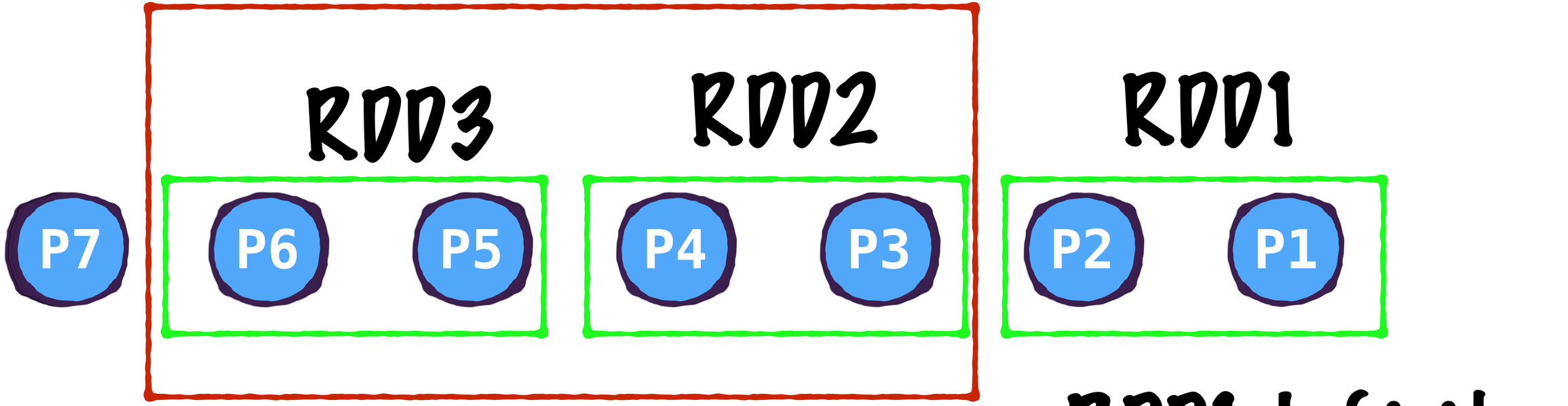
Spark Streaming



Window at t = 3s

Sliding Window

Spark Streaming

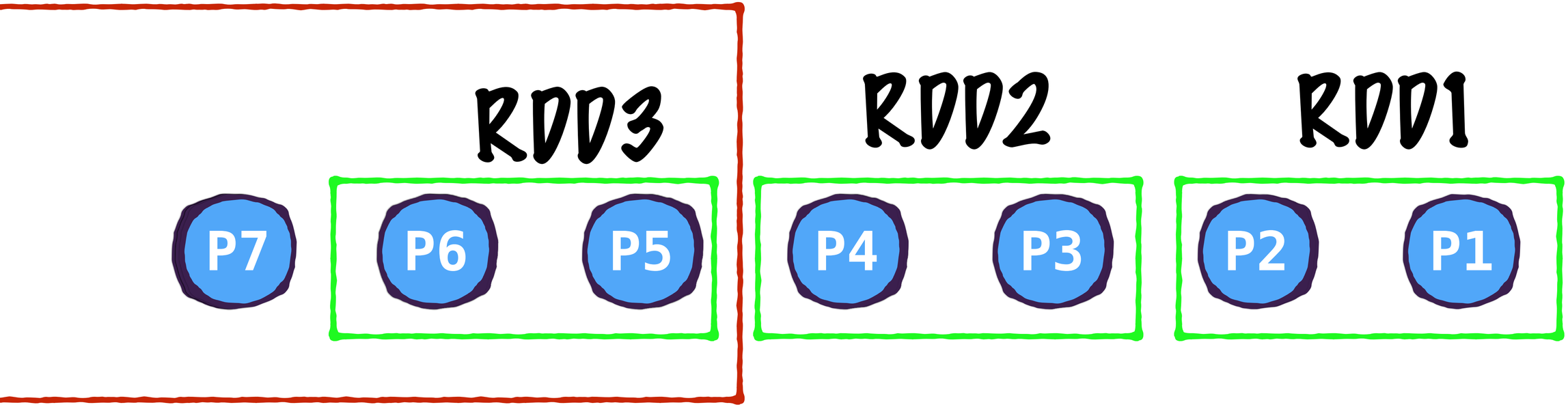


RDD3 entered the window

RDD1 left the window

Sliding Window

Spark Streaming



Window at t = 4s

Here is our old code

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)


    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)
    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)
    counts.print()

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

Spark Streaming

Let's update the code to add a stateful transformation

```
object Streaming {
  def main(args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)

    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)

    val counts = lines.filter(_.contains("Error")).
      flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKey(_+_)

    counts.print()

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

Spark Streaming

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
    reduceByKeyAndWindow(
    {(x,y) => x+y},
    {(x,y) => x-y},
    Seconds(30),Seconds(10))
```

We changed reduceByKey to reduceByKeyAndWindow

Spark Streaming

```
object Streaming {

  def main (args: Array[String]){
    val conf= new SparkConf().setMaster("yarn-client").setAppName("My App")
    val sc= new SparkContext(conf)

    val ssc = new StreamingContext(sc, Seconds(1))
    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    val lines = ssc.socketTextStream(args(0), args(1).toInt)

    val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
      reduceByKeyAndWindow(
      {(x,y) => x+y},
      {(x,y) => x-y},
      Seconds(30),Seconds(10))

    counts.print()

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

This part extracts words from the text and creates pairs (word,1)

Spark Streaming

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
    reduceByKeyAndWindow(
    {(x,y) => x+y},
    {(x,y) => x-y},
    Seconds(30),Seconds(10))
```

This will sum values with the same key

(word,1) (word,count)

```
val counts = lines.filter(_.contains("Error")).
flatMap(_.split(" ")).map( word => (word, 1)).
 reduceByKeyAndWindow(
 {(x,y) => x+y},
 {(x,y) => x-y},
 Seconds(30),Seconds(10))
```

Spark Streaming

The window size is 30 seconds

Sliding interval is 10 seconds

Batch interval is 1 second (set in StreamingContext)

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
     reduceByKeyAndWindow(
     {(x,y) => x+y},
     {(x,y) => x-y},
     Seconds(30),Seconds(10))
```

Spark Streaming

Window = 30 s
Slide = 10 s
Batch = 1s

A new RDD is created every second

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
     reduceByKeyAndWindow(
     {(x,y) => x+y},
     {(x,y) => x-y},
     Seconds(30),Seconds(10))
```

Spark Streaming

Window = 30 s

Slide = 10 s

Batch = 1s

A new RDD is created every second

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
     reduceByKeyAndWindow(
     {(x,y) => x+y},
     {(x,y) => x-y},
     Seconds(30),Seconds(10))
```

Spark Streaming

Window = 30 s

Slide = 10 s

Batch = 1s

A window consists of 30 RDDs

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
     reduceByKeyAndWindow(
     {(x,y) => x+y},
     {(x,y) => x-y},
     Seconds(30),Seconds(10))
```

Spark Streaming

Window = 30 s

Slide = 10 s

Batch = 1s

Every 10s, a new window is created

```
val counts = lines.filter(_.contains("Error")).
    flatMap(_.split(" ")).map( word => (word, 1)).
     reduceByKeyAndWindow(
     {(x,y) => x+y},
     {(x,y) => x-y},
     Seconds(30),Seconds(10))
```

Spark Streaming

Window = 30 s

Slide = 10 s

Batch = 1s

10 RDDs leave the window,
10 RDDs enter the window

10 RDDs leave the window,
10 RDDs enter the window

Spark Streaming

```
reduceByKeyAndWindow(
{(x,y) => x+y},
{(x,y) => x-y},
Seconds(30),Seconds(10))
```

When RDDs enter the window, add the values in the RDDs to the word count

10 RDDs leave the window,
10 RDDs enter the window

Spark Streaming

```
reduceByKeyAndWindow(
{(x,y) => x+y},
{(x,y) => x-y},
Seconds(30),Seconds(10))
```

When RDDs leave the window, subtract the values in the RDDs from the word count

Spark Streaming

Sliding windows are useful to observe trends

Visualizing such output in chart form will allow us to see spikes, such as when sudden payment errors occur!