SPARK STREAMING

Say you work for a payment service

Say you work for a payment service

The number of failed payments/ second

is a mission critical metric

The number of
failed payments/second

Typically business reports are published at a certain frequency

Quarterly,
Monthly,
Weekly,
Daily,
Hourly

The number of
failed payments/second

Quarterly,
Monthly,
Weekly,
Daily,
Hourly

These reports require batch processing

The number of
failed payments/second

Quarterly,
Monthly,
Weekly,
Daily,
Hourly

batch processing

Data is collected and stored in a database / HDFS

The number of
failed payments/second

Quarterly,
Monthly,
Weekly,
Daily,
Hourly

batch processing

Processing tasks are run at the required frequency

The number of
failed payments/second

Quarterly,
Monthly,
Weekly,
Daily,
Hourly

batch processing

Each processing task runs on a 'batch' of data

To monitor

The number of
failed payments/second

batch processing
simply isn't enough!

The number of
failed payments/second

If this metric
shows a spike

You need to know
immediately

The number of
failed payments/second

You need to react within seconds or minutes

The number of
failed payments/second

Otherwise, it could mean very
bad things

Loss of reputation, customer
trust, revenue

Being slow to react could be a
death blow to your business!

The number of
failed payments/second

Mission critical metrics need to be monitored in Real Time

The number of
failed payments/second

Real Time

Payment logs are created
by your payment service

These logs need to be
processed as they are
created

The number of
failed payments/second

Real Time

Payment logs are created
by your payment service

Streaming data

These logs need to
be processed as
they are created

Stream Processing

The number of
failed payments/second

Stream Processing

Typically, distributed computing systems like MapReduce and even Spark

Process data from stable storage (database, HDFS)

The number of
failed payments/second

Stream Processing

Since they read data
from disk

There would be too much lag in
the processing of Streaming data

The number of
failed payments/second

Stream Processing

Usually, folks end up
using separate systems

For batch processing
and stream processing

The number of
failed payments/second

Stream Processing

Apache Storm for
example

is a standalone Stream
processing application

The number of
failed payments/second

Stream Processing

Spark is envisioned as a
general purpose engine to solve
all data processing problems

The number of
failed payments/second

Stream Processing

Spark Streaming provides Stream processing capability within Spark

The number of
failed payments/second

Stream Processing

Spark Streaming means that you can use a single engine (Spark) for both batch and stream processing

Spark Streaming

Just like typical Spark applications involve the use of RDDs

Spark Streaming applications involve the use of

DStreams

DStreams

Spark Streaming

Let's say you have a stream of payment logs

Typically, a stream processing application will process each individual log

DStreams

Spark Streaming

Stream processing typically

DStreams

Spark Streaming

In a Spark Streaming app, the entire stream of logs is represented as a DStream

DStreams

Spark Streaming

The DStream has a property called batch interval (measured in seconds)

DStreams

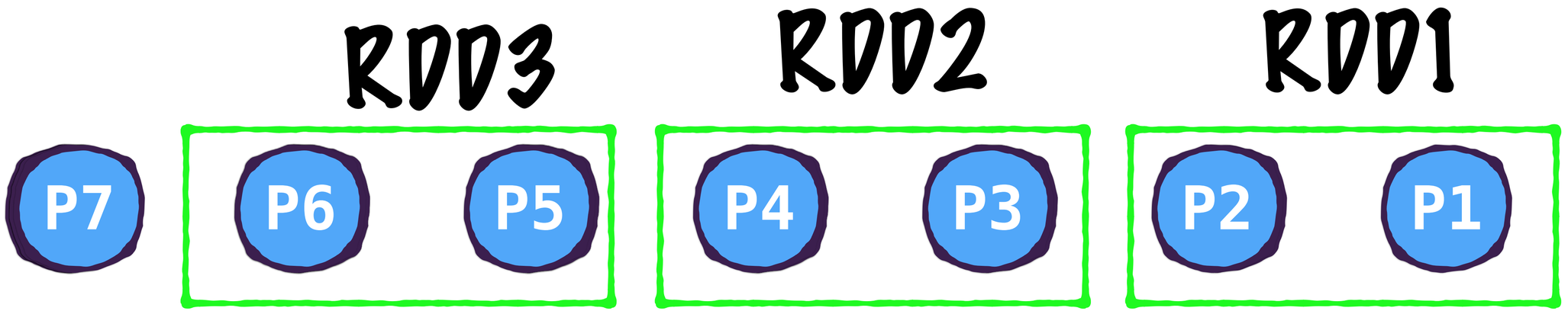
Spark Streaming

batch interval

All data arriving within this interval is grouped into an RDD

DStreams

Spark Streaming



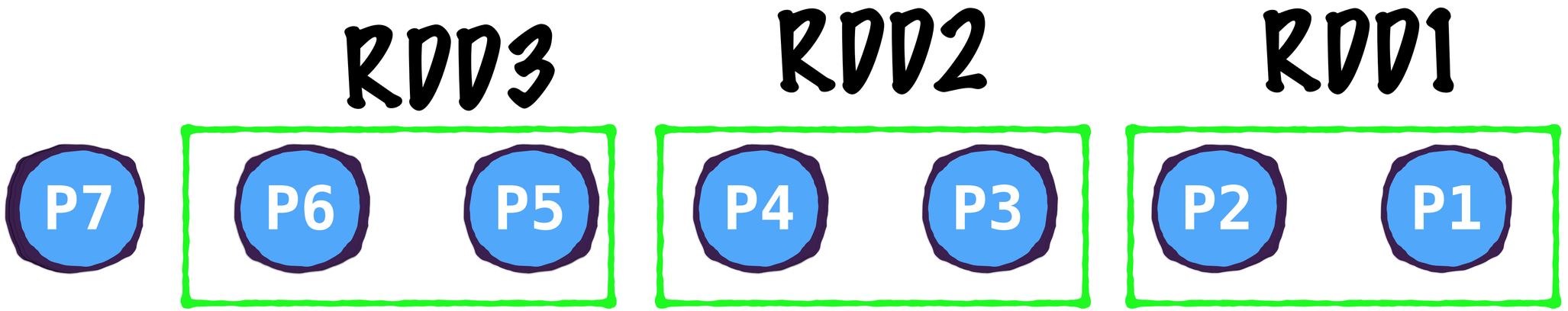
batch interval

Let's say the batch interval is 1 sec

The stream consists of 2 logs per second

DStreams

Spark Streaming

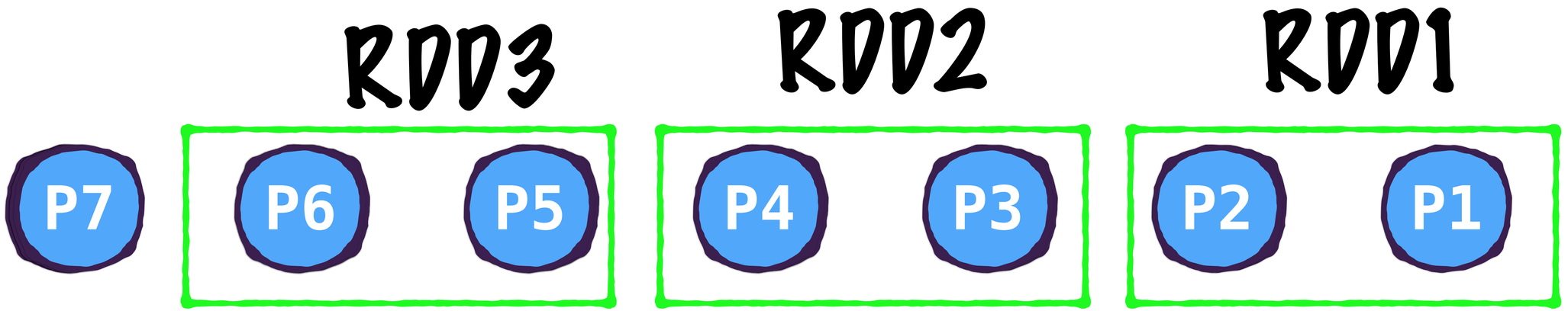


The DStream is a sequence of RDDs

1 RDD every batch interval

DStreams

Spark Streaming

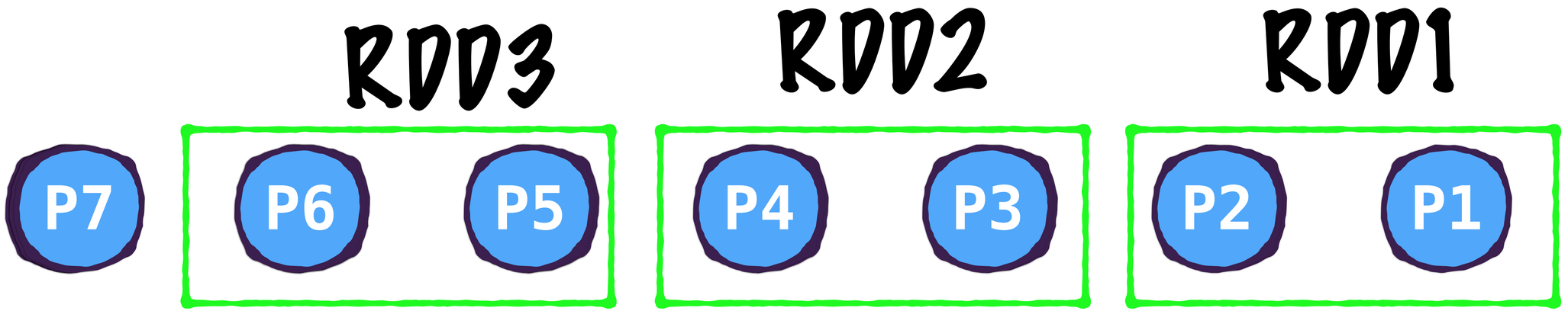


You can apply transformations and actions on DStreams

They will be applied to each of these RDDs as they are created

DStreams

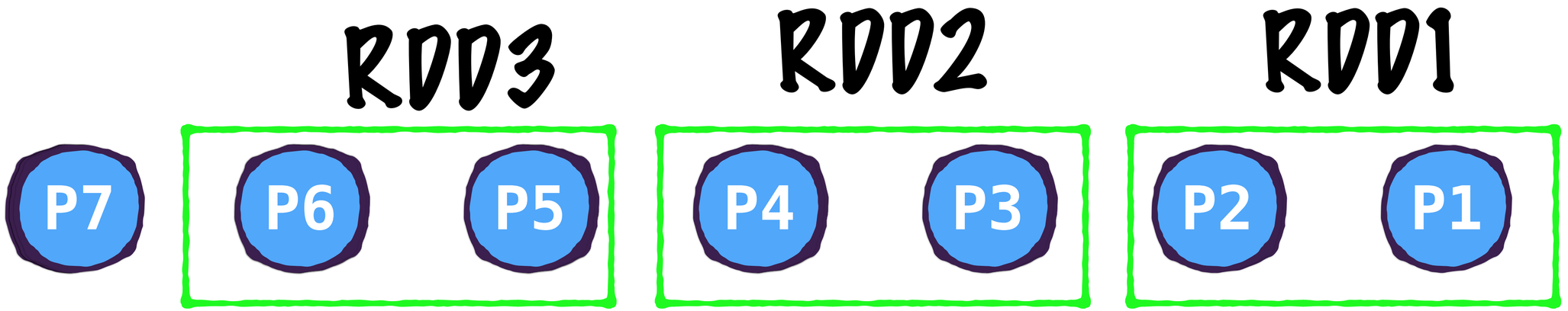
Spark Streaming



With DStreams, Spark still does batch processing

DStreams

Spark Streaming



With DStreams, Spark still does batch processing

On individual RDDs in this stream

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

```
import sys

from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":

    sc = SparkContext(appName="StreamingErrorCount")
    ssc = StreamingContext(sc, 1)

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
    counts = lines.flatMap(lambda line: line.split(" "))\
                  .filter(lambda word:"ERROR" in word)\
                  .map(lambda word: (word, 1))\
                  .reduceByKey(lambda a, b: a+b)

    counts.pprint()
    ssc.start()
    ssc.awaitTermination()
```

Spark Streaming

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

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

Spark Streaming

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

A StreamingContext is used to create DStreams

Spark Streaming

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

Import SparkContext and StreamingContext

Spark Streaming

```
sc = SparkContext(appName="StreamingErrorCount")
ssc = StreamingContext(sc, 1)
```

Set up the SparkContext and StreamingContext

Spark Streaming

```
sc = SparkContext(appName="StreamingErrorCount")
ssc = StreamingContext(sc, 1)
```

The StreamingContext is where you set the batch interval

Spark Streaming

```
sc = SparkContext(appName="StreamingErrorCount")
ssc = StreamingContext(sc, 1)
```

The batch interval is set to 1 second

Spark Streaming

```
sc = SparkContext(appName="StreamingErrorCount")
ssc = StreamingContext(sc, 1)
```

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

Spark Streaming

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

This is some additional setup we need for fault-tolerance

Spark Streaming

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

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

Spark Streaming

```
lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
```

Create the DStream

Spark Streaming

```
lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
```

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

Spark Streaming

```
lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
```

The hostname where the data will arrive

Spark
Streaming

```
lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
```

The port where the data arrives

Spark Streaming

```
lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
```

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

Spark Streaming

```
lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
```

```
> spark-submit Streaming.py localhost 9999
```

Spark Streaming

```
counts = lines.flatMap(lambda line: line.split(" "))\
              .filter(lambda word:"ERROR" in word)\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a+b)
```

Apply any transformations and actions on the DStream

Spark Streaming

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

```
counts = lines.flatMap(lambda line: line.split(" "))\
              .filter(lambda word:"ERROR" in word)\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a+b)
```

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

Spark Streaming

flatMap converts an RDD with lines into an RDD with words

```
counts = lines.flatMap(lambda line: line.split(" "))\
              .filter(lambda word:"ERROR" in word)\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a+b)
```

Spark Streaming

```
counts = lines.flatMap(lambda line: line.split(" "))\
              .filter(lambda word:"ERROR" in word)\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a+b)
```

Filter for words that have the substring "ERROR"

Spark Streaming

Each word is mapped to (word, 1)

```
counts = lines.flatMap(lambda line: line.split(" "))\
              .filter(lambda word:"ERROR" in word)\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a+b)
```

Spark Streaming

```
counts = lines.flatMap(lambda line: line.split(" "))\
              .filter(lambda word:"ERROR" in word)\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a+b)
```

Returns (word, count)

Spark Streaming

Prints the count to screen

```
counts.pprint()
```

This method is called for each individual RDD in the DStream

```
counts.pprint()
```

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

```
counts.pprint()
```

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

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

Spark Streaming

```
import sys

from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":

    sc = SparkContext(appName="StreamingErrorCount")
    ssc = StreamingContext(sc, 1)

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
    counts = lines.flatMap(lambda line: line.split(" "))\
                  .filter(lambda word:"ERROR" in word)\
                  .map(lambda word: (word, 1))\
                  .reduceByKey(lambda a, b: a+b)

    counts.pprint()
    ssc.start()
    ssc.awaitTermination()
```

Streaming.py

```python
import sys

from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":

    sc = SparkContext(appName="StreamingErrorCount")
    ssc = StreamingContext(sc, 1)

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
    counts = lines.flatMap(lambda line: line.split(" "))\
                  .filter(lambda word:"ERROR" in word)\
                  .map(lambda word: (word, 1))\
                  .reduceByKey(lambda a, b: a+b)

    counts.pprint()
    ssc.start()
    ssc.awaitTermination()
```

Spark Streaming

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

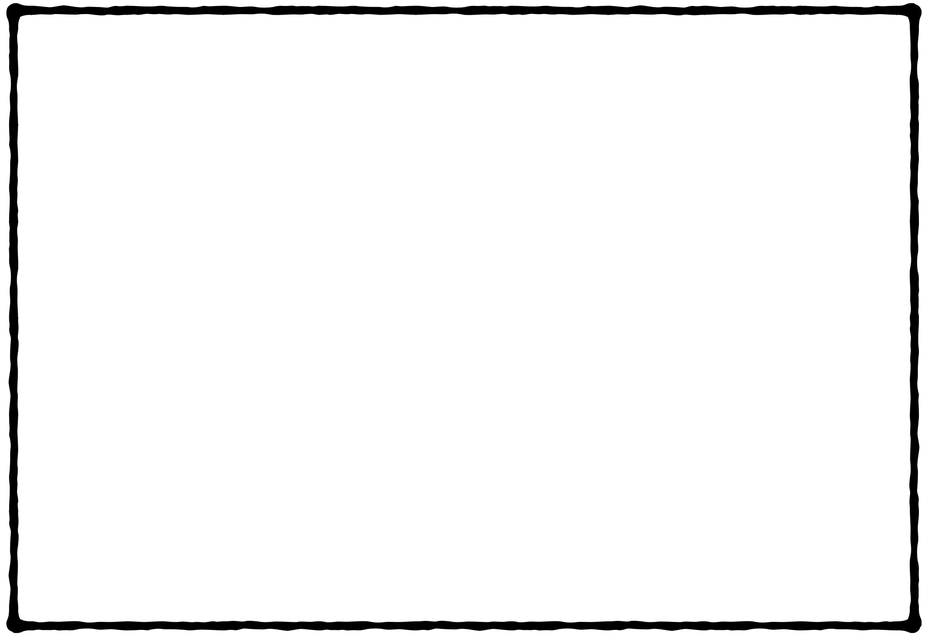
```
> nc -lk 9999
```

Streaming.py

```python
import sys

from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":

    sc = SparkContext(appName="StreamingErrorCount")
    ssc = StreamingContext(sc, 1)

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
    counts = lines.flatMap(lambda line: line.split(" "))\
                  .filter(lambda word:"ERROR" in word)\
                  .map(lambda word: (word, 1))\
                  .reduceByKey(lambda a, b: a+b)

    counts.pprint()
    ssc.start()
    ssc.awaitTermination()
```

Spark Streaming

```
> nc -lk 9999
```



In a separate terminal submit your script

```
> spark-submit Streaming.py localhost 9999
```

Streaming.py

Spark Streaming

```
> spark-submit Streaming.py localhost 9999
```

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

```
> nc -lk 9999
```

Streaming.py

Spark Streaming

```
> spark-submit Streaming.py 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.py

Spark Streaming

```
> spark-submit Streaming.py 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.py

Spark Streaming

```
> spark-submit Streaming.py 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.py

Spark Streaming

```
> spark-submit Streaming.py 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.py

Spark Streaming

```
> spark-submit Streaming.py 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.py

Spark Streaming

```
> spark-submit Streaming.py 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.py

```
-------------------------------------------
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

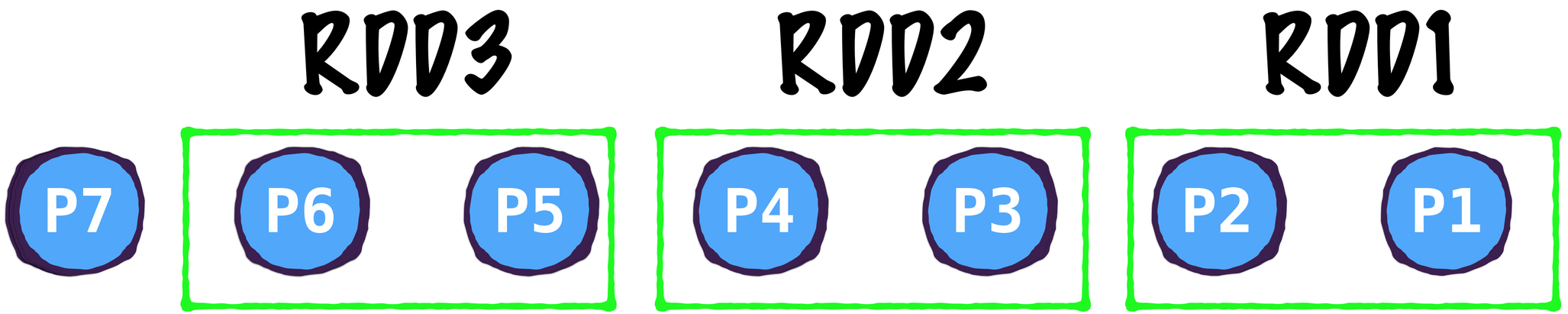
A Sliding Window consists of multiple RDDs in the DStream

Stateful

reduceByWindow,
reduceByKeyAndWindow
etc

Sliding Window

Spark Streaming

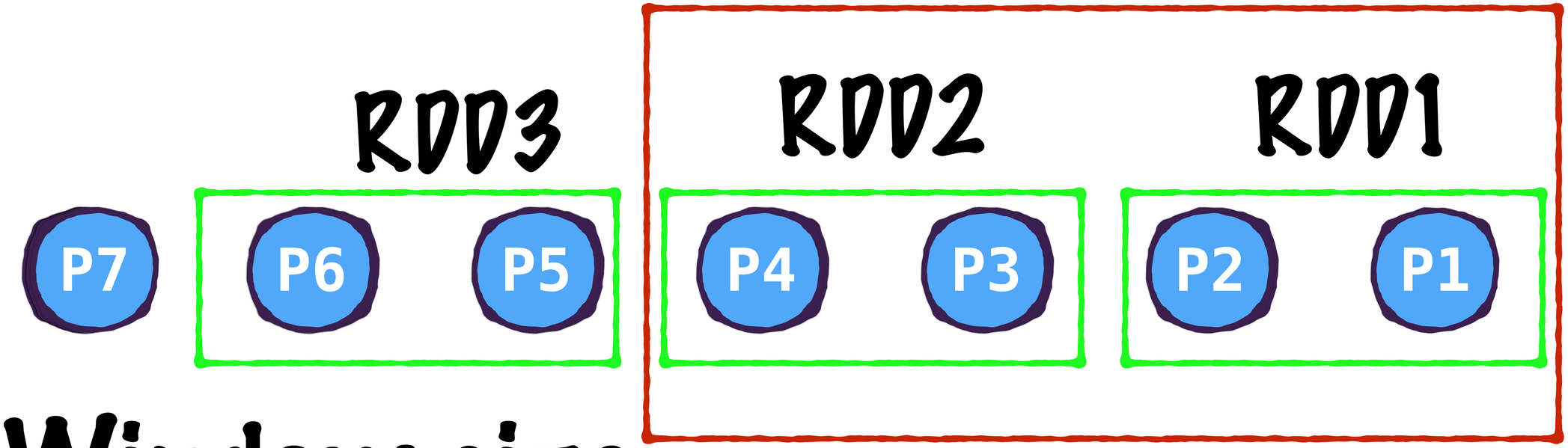


Let's say the batch interval is 1 sec

The stream consists of 2 logs per second

Sliding Window

Spark Streaming

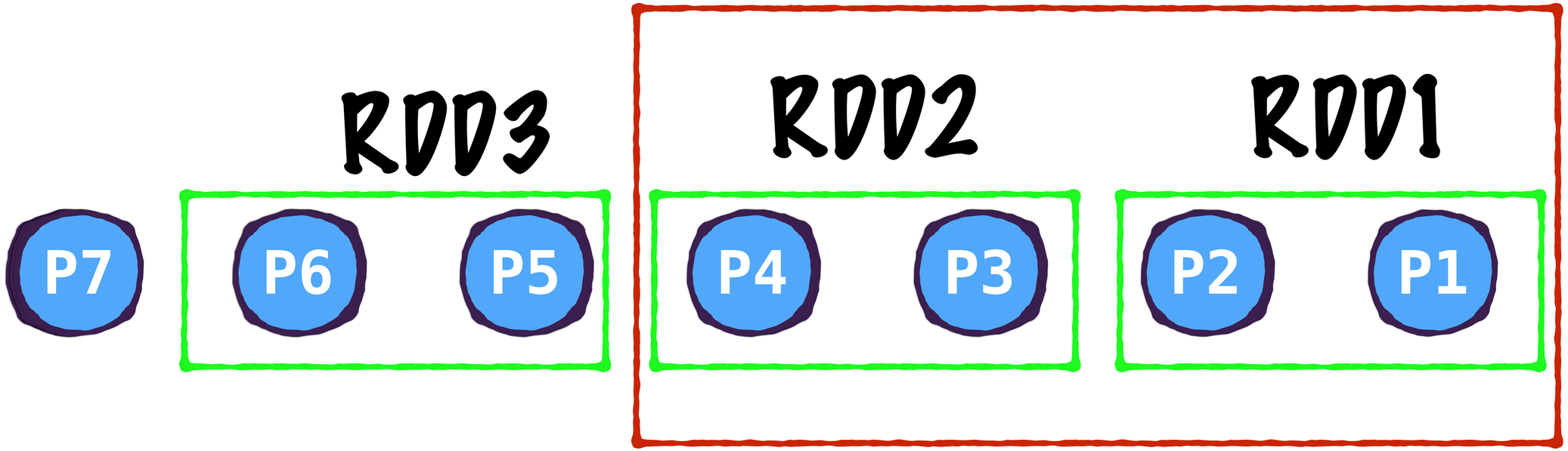


Window size is 2 seconds

Sliding interval is 1 second

Sliding Window

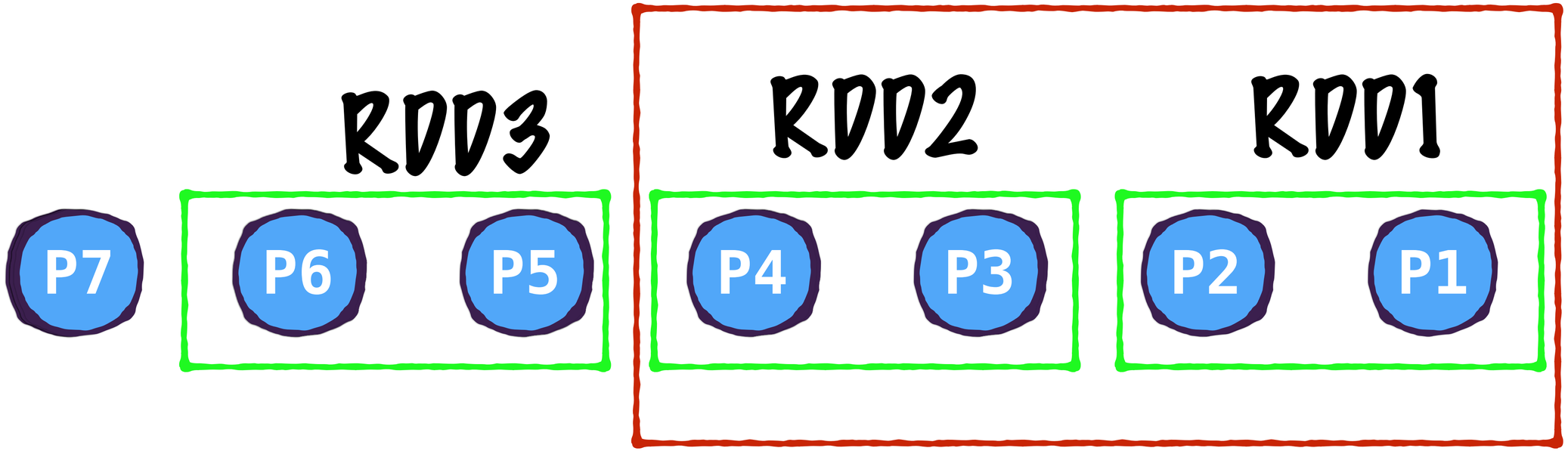
Spark Streaming



Window at t = 2s

Sliding Window

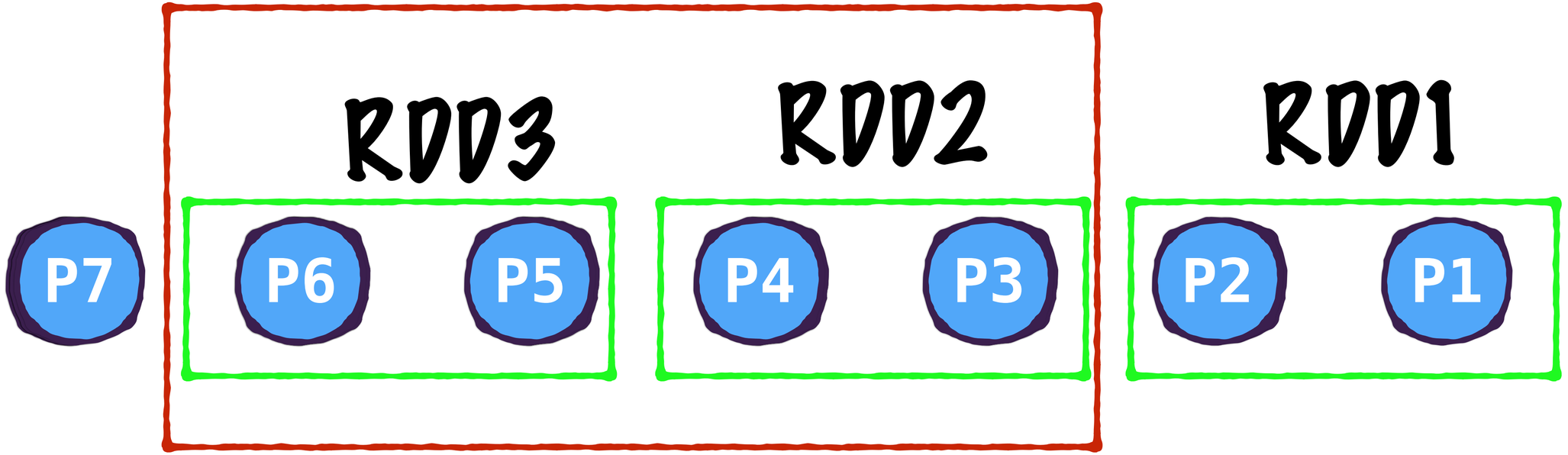
Spark Streaming



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

Sliding Window

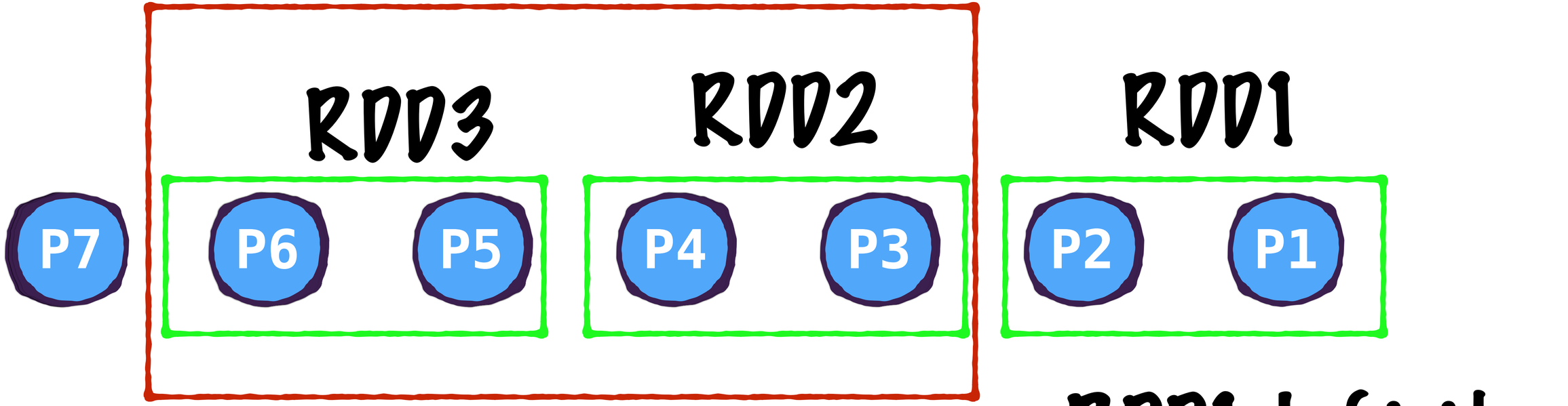
Spark Streaming



Window at t = 3s

Sliding Window

Spark Streaming

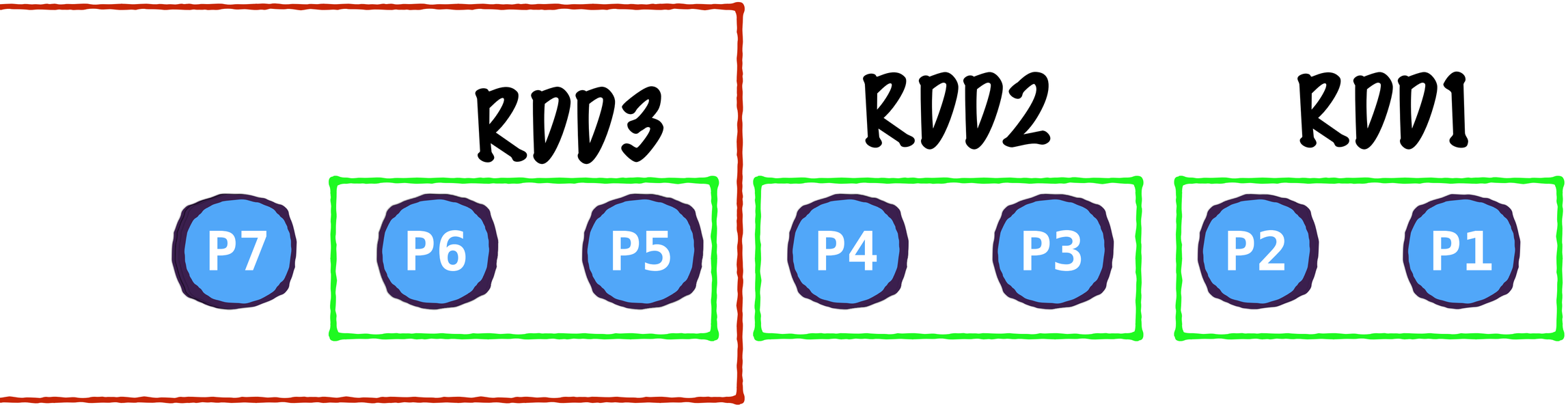


RDD3 entered the window

RDD1 left the window

Sliding Window

Spark Streaming



Window at t = 4s

Here is our old code

Spark Streaming

```
import sys

from pyspark import SparkContext
from pyspark.streaming import StreamingContext

if __name__ == "__main__":

    sc = SparkContext(appName="StreamingErrorCount")
    ssc = StreamingContext(sc, 1)

    ssc.checkpoint("hdfs:///user/swethakolalapudi/streaming")
    lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
    counts = lines.flatMap(lambda line: line.split(" "))\
                  .filter(lambda word:"ERROR" in word)\
                  .map(lambda word: (word, 1))\
                  .reduceByKey(lambda a, b: a+b)

    counts.pprint()
    ssc.start()
    ssc.awaitTermination()
```

Spark Streaming

Let's update the code to add a stateful transformation

```
counts = lines.flatMap(lambda line: line.split(" "))\
              .filter(lambda word:"ERROR" in word)\
              .map(lambda word: (word, 1))\
              .reduceByKey(lambda a, b: a+b)
```

Spark Streaming

```
counts = lines.flatMap(lambda line: line.split(" "))\
              .filter(lambda line:"ERROR" in line)\
              .map(lambda word: (word, 1))\
              .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
```

We changed reduceByKey to reduceByKeyAndWindow

Spark Streaming

```
= lines.flatMap(lambda line: line.split(" "))\
        .filter(lambda line:"ERROR" in line)\
        .map(lambda word: (word, 1))\
        .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y
```

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

Spark Streaming

```
es.flatMap(lambda line: line.split(" "))\
  .filter(lambda line:"ERROR" in line)\
  .map(lambda word: (word, 1))\
  .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
```

This will sum values with the same key

(word,1) (word,count)

Spark Streaming

```
es.flatMap(lambda line: line.split(" "))\
  .filter(lambda line:"ERROR" in line)\
  .map(lambda word: (word, 1))\
  .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
```

The window size is 30 seconds

Sliding interval is 10 seconds

Batch interval is 1 second (set in StreamingContext)

Spark Streaming

```
es.flatMap(lambda line: line.split(" "))\
  .filter(lambda line:"ERROR" in line)\
  .map(lambda word: (word, 1))\
  .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
```

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

A new RDD is created every second

Spark Streaming

```
es.flatMap(lambda line: line.split(" "))\
  .filter(lambda line:"ERROR" in line)\
  .map(lambda word: (word, 1))\
  .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
```

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

A new RDD is created every second

Spark Streaming

```
es.flatMap(lambda line: line.split(" "))\
  .filter(lambda line:"ERROR" in line)\
  .map(lambda word: (word, 1))\
  .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
```

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

A window consists of 30 RDDs

Spark Streaming

```
es.flatMap(lambda line: line.split(" "))\
  .filter(lambda line:"ERROR" in line)\
  .map(lambda word: (word, 1))\
  .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
```

Window = 30 s

Slide = 10 s

Batch = 1s

Every 10s, a new window is created

Spark Streaming

```
es.flatMap(lambda line: line.split(" "))\
  .filter(lambda line:"ERROR" in line)\
  .map(lambda word: (word, 1))\
  .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 10)
```

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

```
es.flatMap(lambda line: line.split(" "))\
  .filter(lambda line:"ERROR" in line)\
  .map(lambda word: (word, 1))\
  .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 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

```
es.flatMap(lambda line: line.split(" "))\
  .filter(lambda line:"ERROR" in line)\
  .map(lambda word: (word, 1))\
  .reduceByKeyAndWindow(lambda x, y: x + y, lambda x, y: x - y, 30, 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!