Apache Spark
when things go wrong

Apache Spark - when things go wrong

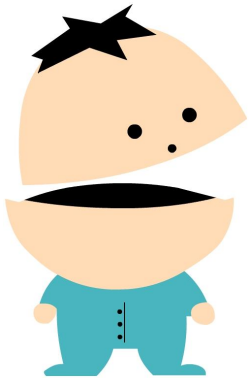
```
val sc      = new SparkContext(conf)
sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
    .take(300)
    .foreach(println)
```

```
1 2015-05-09T01:11 UserInitialized Brad Smith
1 2015-05-08T02:12 LoggedIn
1 2015-05-07T03:13 LoggedIn
```

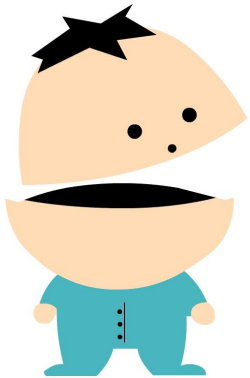
*If above seems **odd**, this talk is rather **not** for you.*

Target for this talk

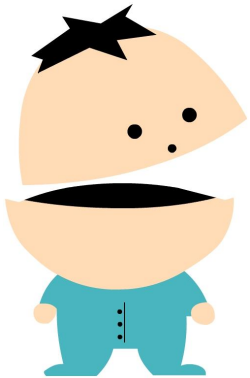
Target for this talk

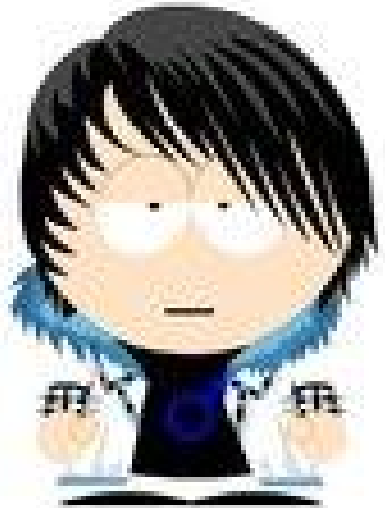


Target for this talk

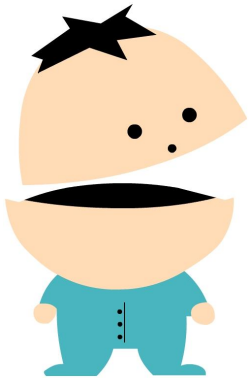


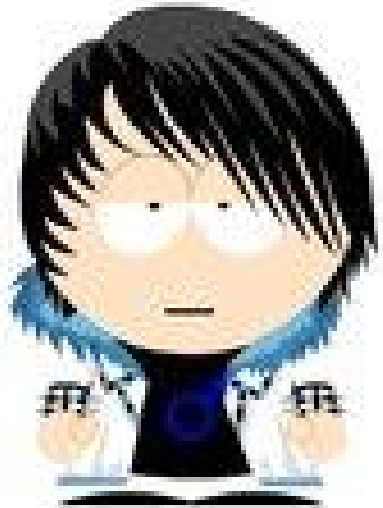
Target for this talk



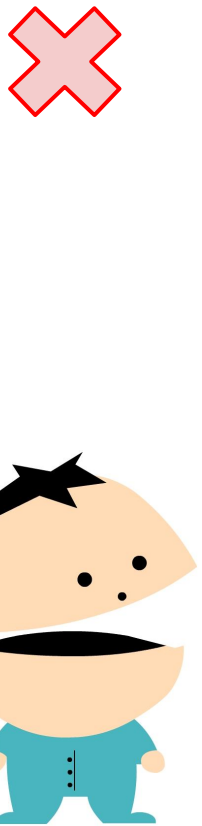


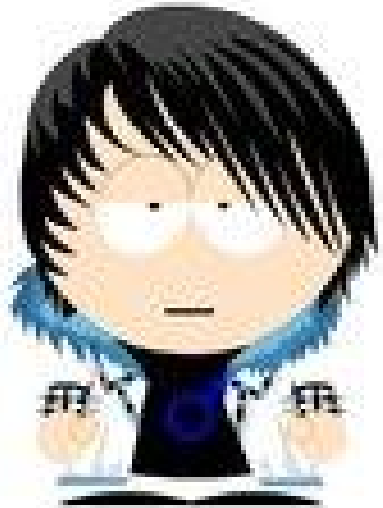
Target for this talk



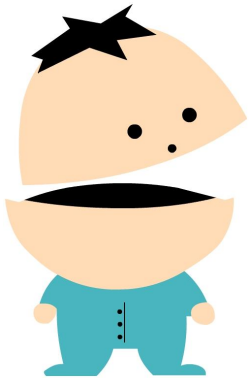


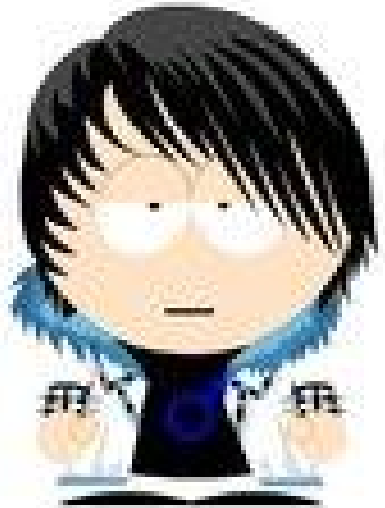
Target for this talk



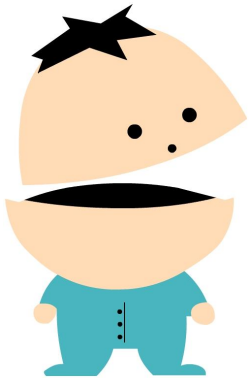


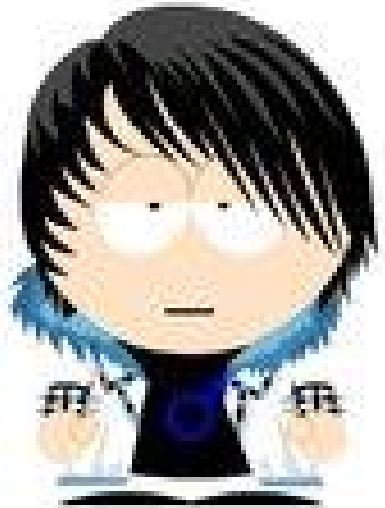
Target for this talk



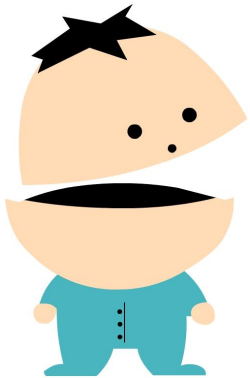


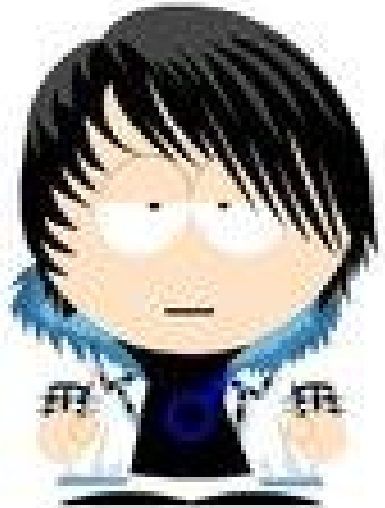
Target for this talk





Target for this talk





4 things for take-away

4 things for take-away

1. Knowledge of how Apache Spark works internally

4 things for take-away

1. Knowledge of how Apache Spark works internally
2. Courage to look at Spark's implementation (code)

4 things for take-away

1. Knowledge of how Apache Spark works internally
2. Courage to look at Spark's implementation (code)
3. Notion of how to write efficient Spark programs

4 things for take-away

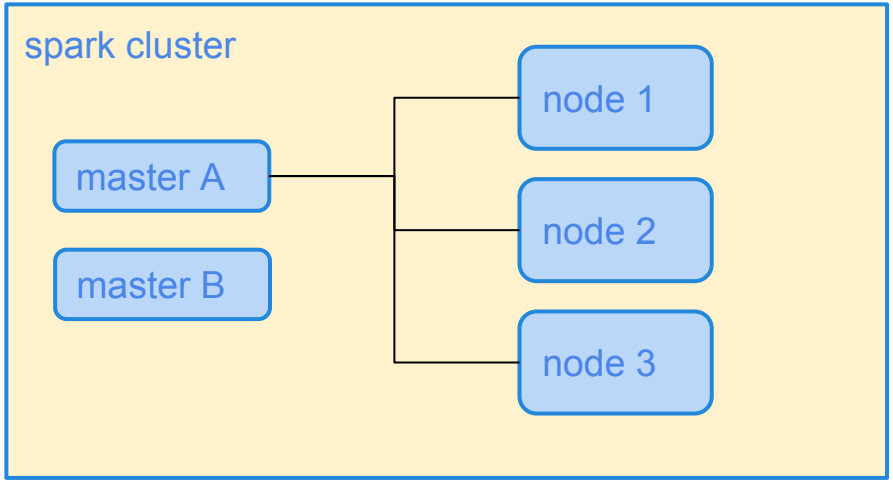
1. Knowledge of how Apache Spark works internally
2. Courage to look at Spark's implementation (code)
3. Notion of how to write efficient Spark programs
4. Basic ability to monitor Spark when things are starting to act weird

Two words about examples used

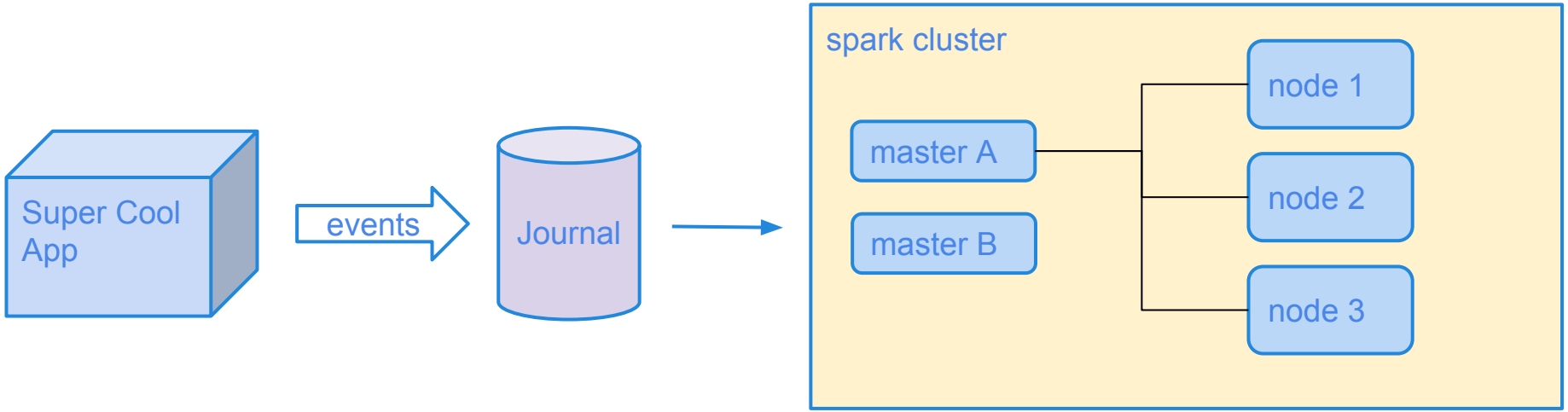
Two words about examples used

Two words about examples used

Two words about examples used

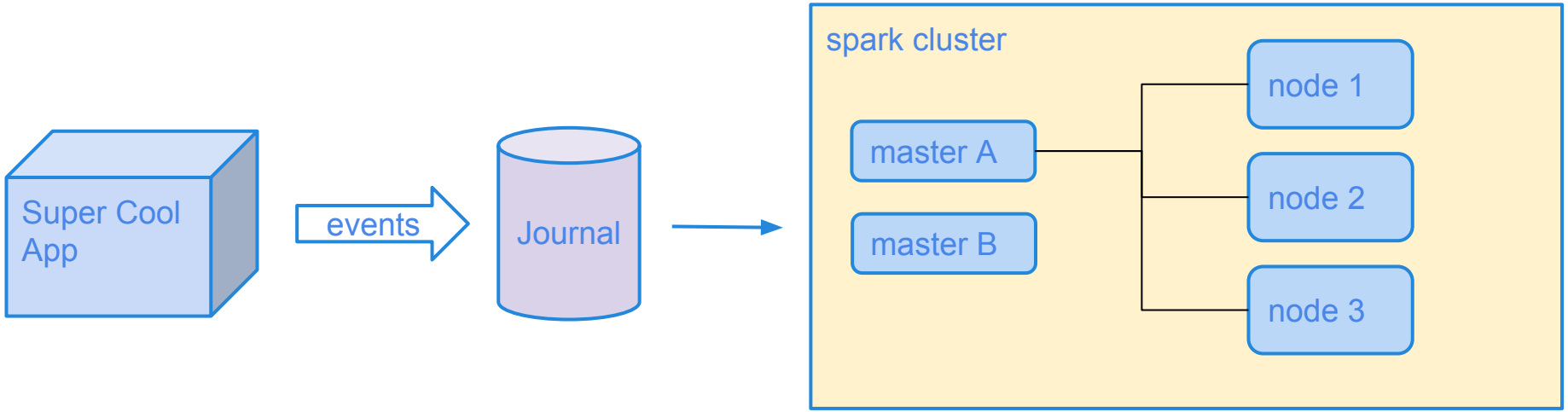


Two words about examples used



Two words about examples used

```
1 2015-05-09T01:11 UserInitialized Brad Smith
1 2015-05-08T02:12 LoggedIn
1 2015-05-07T03:13 LoggedIn
```



Two words about examples used

```
1 2015-05-09T01:11 UserInitialized Brad Smith
1 2015-05-08T02:12 LoggedIn
1 2015-05-07T03:13 LoggedIn
```

Two words about examples used

```
sc.textFile("hdfs://journal/*")
```

```
1 2015-05-09T01:11 UserInitialized Brad Smith
1 2015-05-08T02:12 LoggedIn
1 2015-05-07T03:13 LoggedIn
```

Two words about examples used

```
sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
```

1 2015-05-09T01:11 UserInitialized Brad Smith
1 2015-05-08T02:12 LoggedIn
1 2015-05-07T03:13 LoggedIn

Two words about examples used

```
1 2015-05-09T01:11 UserInitialized Brad Smith
1 2015-05-08T02:12 LoggedIn
1 2015-05-07T03:13 LoggedIn
```

```
sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
```

Two words about examples used

```
1 2015-05-09T01:11 UserInitialized Brad Smith
1 2015-05-08T02:12 LoggedIn
1 2015-05-07T03:13 LoggedIn
```

```
sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
```

Two words about examples used

```
1 2015-05-09T01:11 UserInitialized Brad Smith
1 2015-05-08T02:12 LoggedIn
1 2015-05-07T03:13 LoggedIn
```

```
sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
    .take(300)
```

Two words about examples used

```
1 2015-05-09T01:11 UserInitialized Brad Smith
1 2015-05-08T02:12 LoggedIn
1 2015-05-07T03:13 LoggedIn
```

```
sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
    .take(300)
    .foreach(println)
```

What is a RDD?

What is a RDD?

Resilient Distributed Dataset

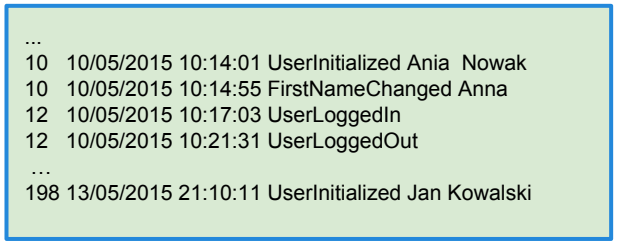
What is a RDD?

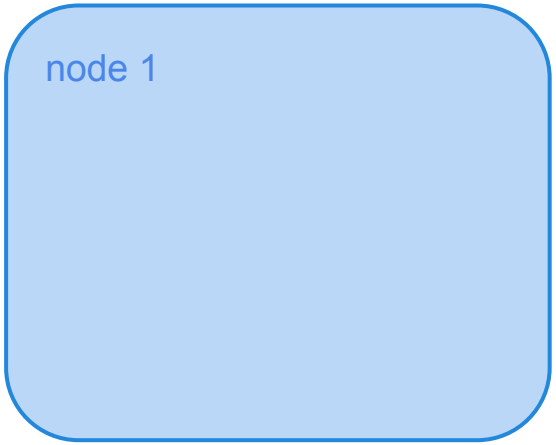
Resilient **Distributed Dataset**

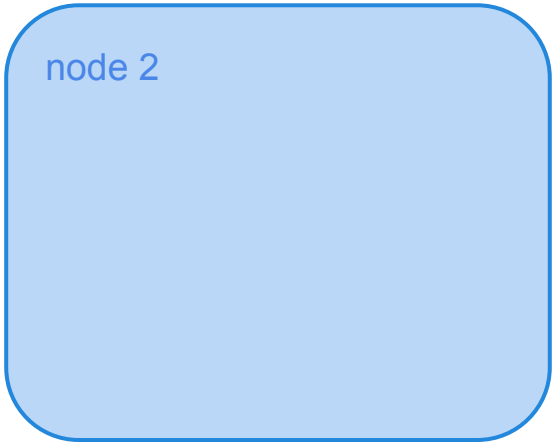
What is a RDD?

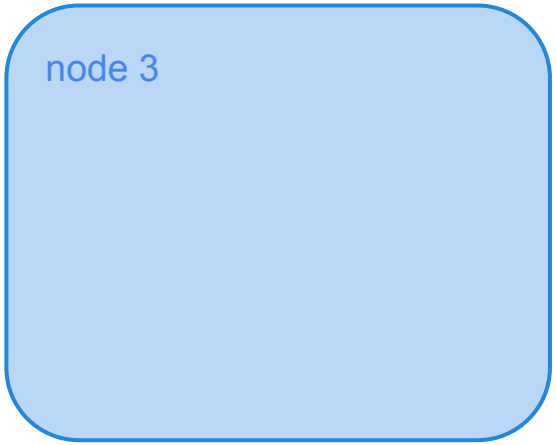
```
...
10   10/05/2015 10:14:01 UserInitialized Ania  Nowak
10   10/05/2015 10:14:55 FirstNameChanged Anna
12   10/05/2015 10:17:03 UserLoggedIn
12   10/05/2015 10:21:31 UserLoggedOut
 …
198 13/05/2015 21:10:11 UserInitialized Jan Kowalski
```

What is a RDD?

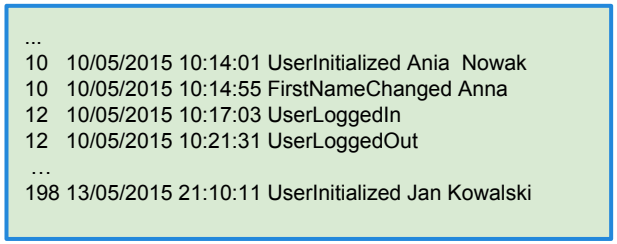

...
10 10/05/2015 10:14:01 UserInitialized Ania Nowak
10 10/05/2015 10:14:55 FirstNameChanged Anna
12 10/05/2015 10:17:03 UserLoggedIn
12 10/05/2015 10:21:31 UserLoggedOut
…
198 13/05/2015 21:10:11 UserInitialized Jan Kowalski

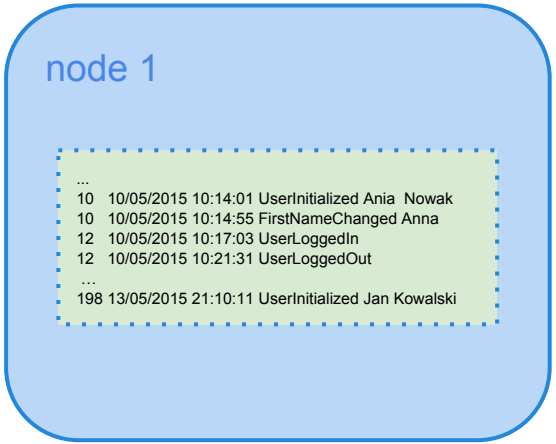

node 1

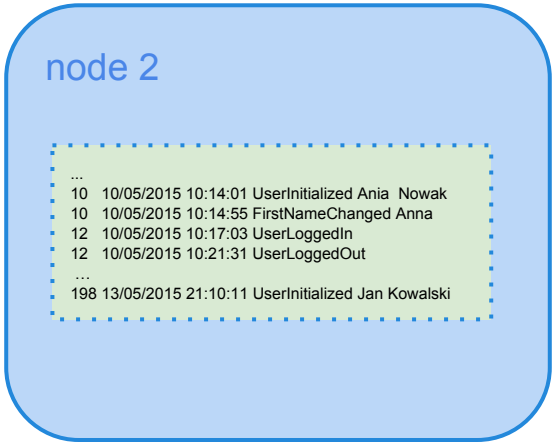

node 2

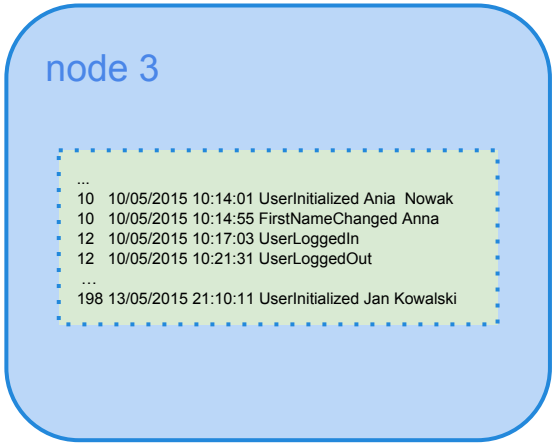

node 3

What is a RDD?


...
10 10/05/2015 10:14:01 UserInitialized Ania Nowak
10 10/05/2015 10:14:55 FirstNameChanged Anna
12 10/05/2015 10:17:03 UserLoggedIn
12 10/05/2015 10:21:31 UserLoggedOut
…
198 13/05/2015 21:10:11 UserInitialized Jan Kowalski


node 1
...
10 10/05/2015 10:14:01 UserInitialized Ania Nowak
10 10/05/2015 10:14:55 FirstNameChanged Anna
12 10/05/2015 10:17:03 UserLoggedIn
12 10/05/2015 10:21:31 UserLoggedOut
…
198 13/05/2015 21:10:11 UserInitialized Jan Kowalski


node 2
...
10 10/05/2015 10:14:01 UserInitialized Ania Nowak
10 10/05/2015 10:14:55 FirstNameChanged Anna
12 10/05/2015 10:17:03 UserLoggedIn
12 10/05/2015 10:21:31 UserLoggedOut
…
198 13/05/2015 21:10:11 UserInitialized Jan Kowalski


node 3
...
10 10/05/2015 10:14:01 UserInitialized Ania Nowak
10 10/05/2015 10:14:55 FirstNameChanged Anna
12 10/05/2015 10:17:03 UserLoggedIn
12 10/05/2015 10:21:31 UserLoggedOut
…
198 13/05/2015 21:10:11 UserInitialized Jan Kowalski

What is a RDD?

What is a RDD?

RDD needs to hold 3 chunks of information in order to do its work:

What is a RDD?

RDD needs to hold 3 chunks of information in order to do its work:

1. pointer to his parent

What is a RDD?

RDD needs to hold 3 chunks of information in order to do its work:

1. pointer to his parent
2. how its internal data is partitioned

What is a RDD?

RDD needs to hold 3 chunks of information in order to do its work:

1. pointer to his parent
2. how its internal data is partitioned
3. how evaluate its internal data

What is a RDD?

RDD needs to hold 3 chunks of information in order to do its work:

1. pointer to his parent
2. how its internal data is partitioned
3. how evaluate its internal data

What is a partition?

What is a partition?

A partition represents subset of data within your distributed collection.

What is a partition?

A partition represents subset of data within your distributed collection.

How this subset is defined depends on type of the RDD

example: HadoopRDD

```
val journal = sc.textFile(“hdfs://journal/*”)
```

example: HadoopRDD

```
val journal = sc.textFile(“hdfs://journal/*”)
```

How HadoopRDD is partitioned?

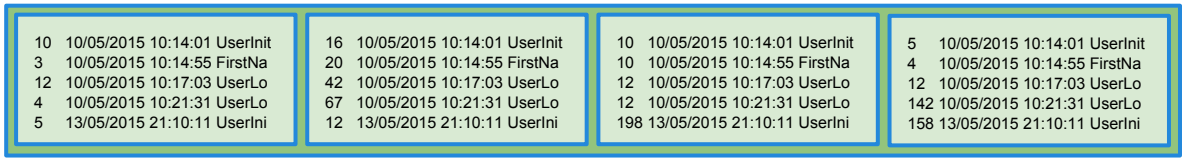
example: HadoopRDD

```
val journal = sc.textFile(“hdfs://journal/*”)
```

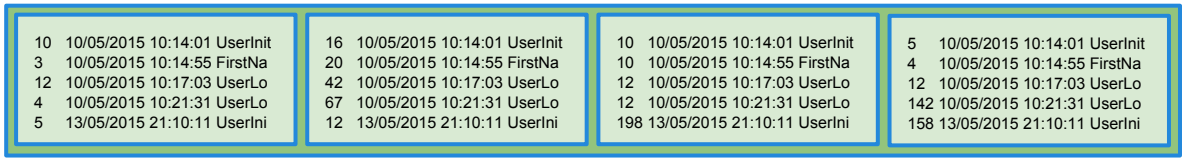
How HadoopRDD is partitioned?

In HadoopRDD partition is exactly the same as file chunks in HDFS

example: HadoopRDD



example: HadoopRDD

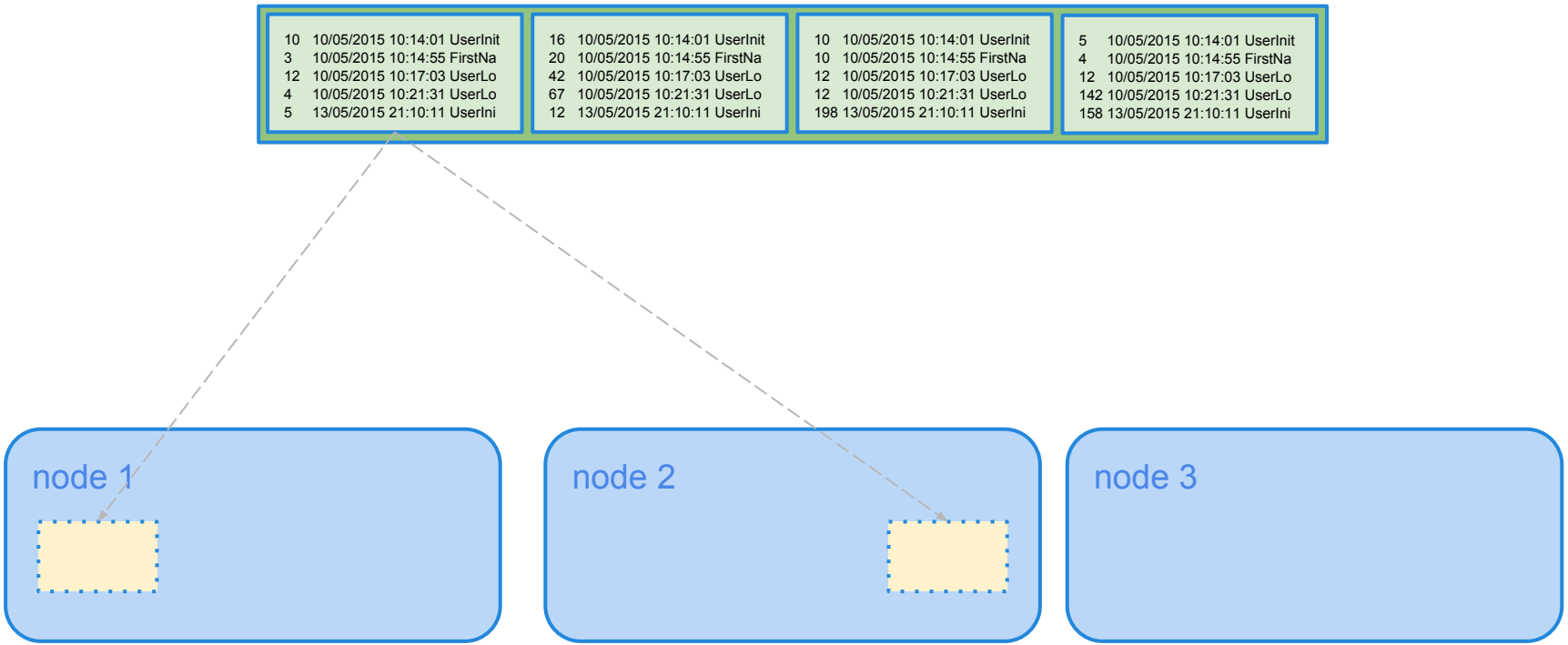


node 1

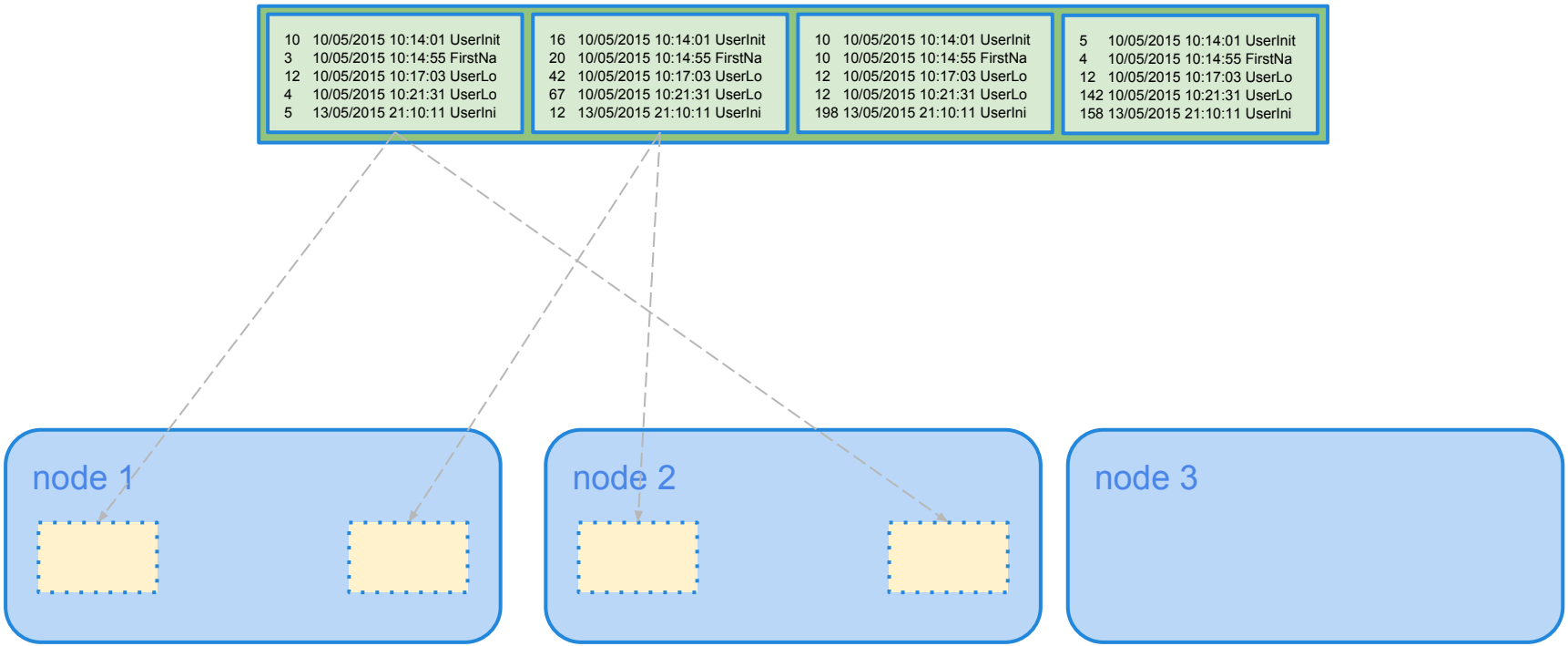
node 2

node 3

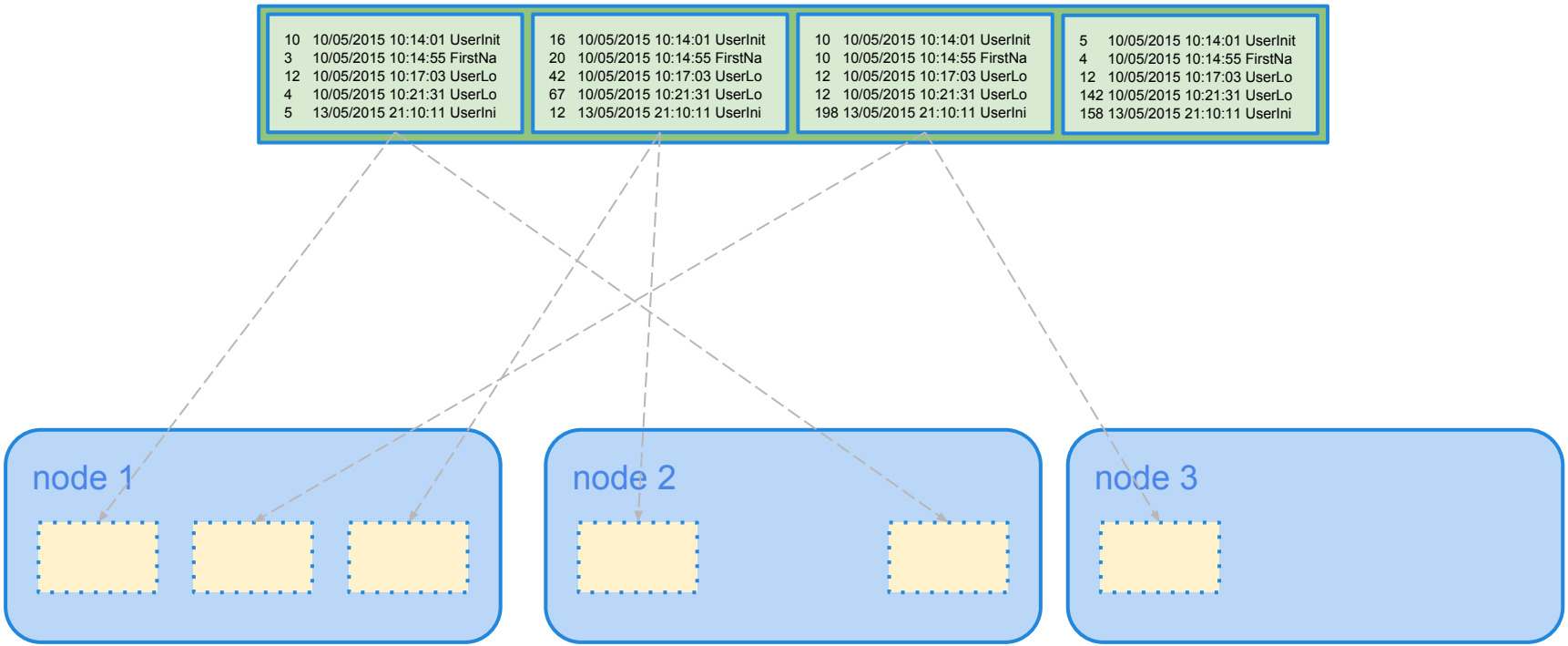
example: HadoopRDD



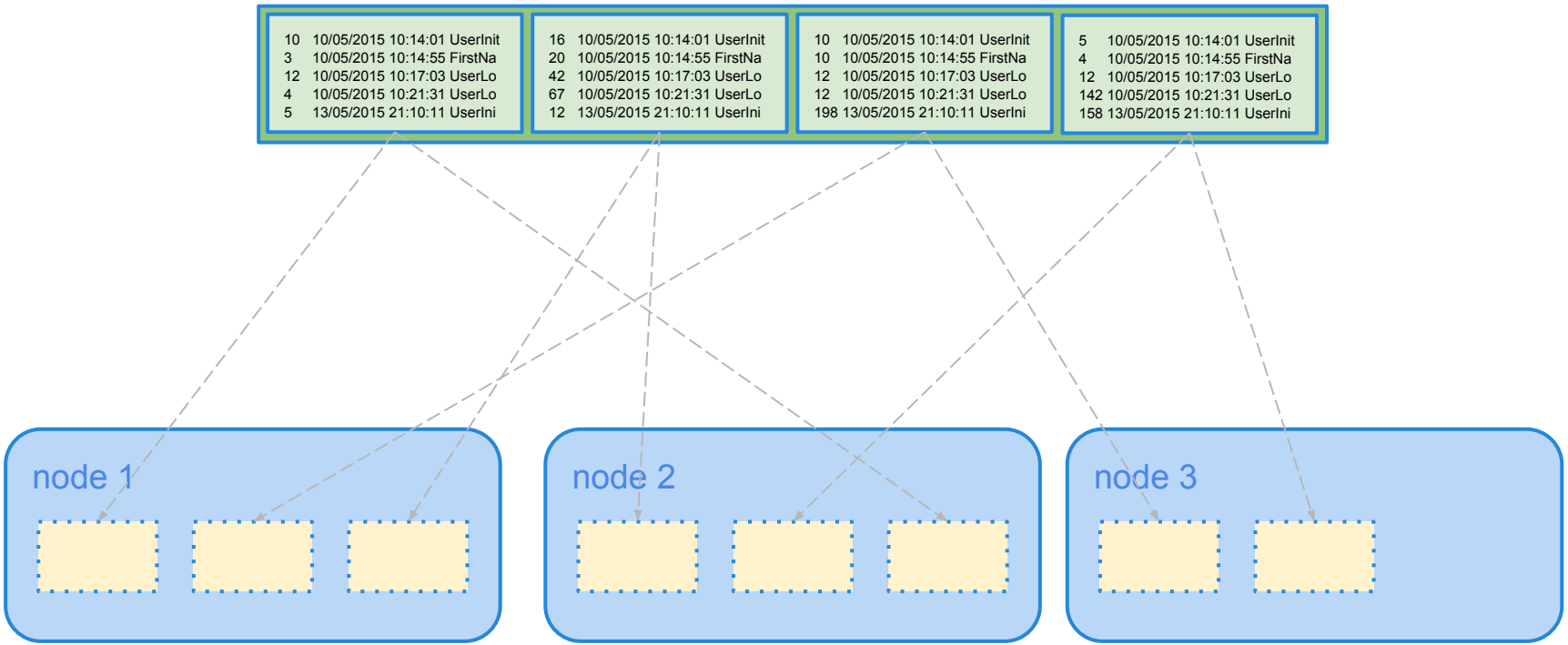
example: HadoopRDD



example: HadoopRDD



example: HadoopRDD



example: HadoopRDD

```
class HadoopRDD[K, V](...) extends RDD[(K, V)](sc, Nil) with Logging {
...
override def getPartitions: Array[Partition] = {
        val jobConf = getJobConf()
        SparkHadoopUtil.get.addCredentials(jobConf)
        val inputFormat = getInputFormat(jobConf)
        if (inputFormat.isInstanceOf[Configurable]) {
                inputFormat.asInstanceOf[Configurable].setConf(jobConf)
        }
        val inputSplits = inputFormat.getSplits(jobConf, minPartitions)
        val array = new Array[Partition](inputSplits.size)
        for (i <- 0 until inputSplits.size) { array(i) = new HadoopPartition(id, i, inputSplits(i))  }
        array
}
```

example: HadoopRDD

```
class HadoopRDD[K, V](...) extends RDD[(K, V)](sc, Nil) with Logging {
...
override def getPartitions: Array[Partition] = {
        val jobConf = getJobConf()
        SparkHadoopUtil.get.addCredentials(jobConf)
        val inputFormat = getInputFormat(jobConf)
        if (inputFormat.isInstanceOf[Configurable]) {
                inputFormat.asInstanceOf[Configurable].setConf(jobConf)
        }
        val inputSplits = inputFormat.getSplits(jobConf, minPartitions)
        val array = new Array[Partition](inputSplits.size)
        for (i <- 0 until inputSplits.size) { array(i) = new HadoopPartition(id, i, inputSplits(i))  }
        array
}
```

example: HadoopRDD

```
class HadoopRDD[K, V](...) extends RDD[(K, V)](sc, Nil) with Logging {
...
override def getPartitions: Array[Partition] = {
        val jobConf = getJobConf()
        SparkHadoopUtil.get.addCredentials(jobConf)
        val inputFormat = getInputFormat(jobConf)
        if (inputFormat.isInstanceOf[Configurable]) {
                inputFormat.asInstanceOf[Configurable].setConf(jobConf)
        }
        val inputSplits = inputFormat.getSplits(jobConf, minPartitions)
        val array = new Array[Partition](inputSplits.size)
        for (i <- 0 until inputSplits.size) { array(i) = new HadoopPartition(id, i, inputSplits(i))  }
        array
}
```

example: MapPartitionsRDD

```
val journal = sc.textFile(“hdfs://journal/*”)
val fromMarch = journal.filter {
     case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
```

example: MapPartitionsRDD

```
val journal = sc.textFile(“hdfs://journal/*”)
val fromMarch = journal.filter {
     case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
```

How MapPartitionsRDD is partitioned?

example: MapPartitionsRDD

```
val journal = sc.textFile(“hdfs://journal/*”)
val fromMarch = journal.filter {
     case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
```

How MapPartitionsRDD is partitioned?

MapPartitionsRDD inherits partition information from its parent RDD

example: MapPartitionsRDD

```
class MapPartitionsRDD[U: ClassTag, T: ClassTag](...)  extends RDD[U](prev) {
...
  override def getPartitions: Array[Partition] = firstParent[T].partitions
```

What is a RDD?

RDD needs to hold 3 chunks of information in order to do its work:

1. pointer to his parent
2. how its internal data is partitioned
3. how evaluate its internal data

What is a RDD?

RDD needs to hold 3 chunks of information in order to do its work:

1. pointer to his parent
2. how its internal data is partitioned
3. how evaluate its internal data

RDD parent

```
sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
    .take(300)
    .foreach(println)
```

RDD parent

```
sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
    .take(300)
    .foreach(println)
```

RDD parent

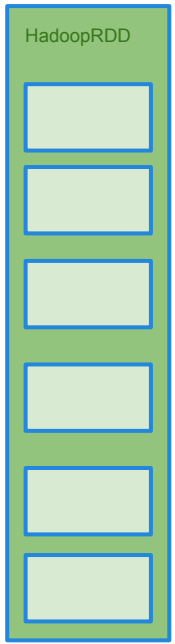
```
sc.textFile()
    .groupBy()
    .map { }
    .filter {

    }
    .take()
    .foreach()
```

Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```

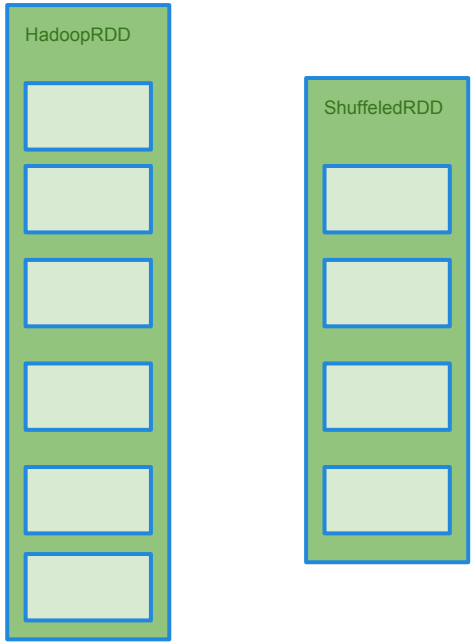
Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```



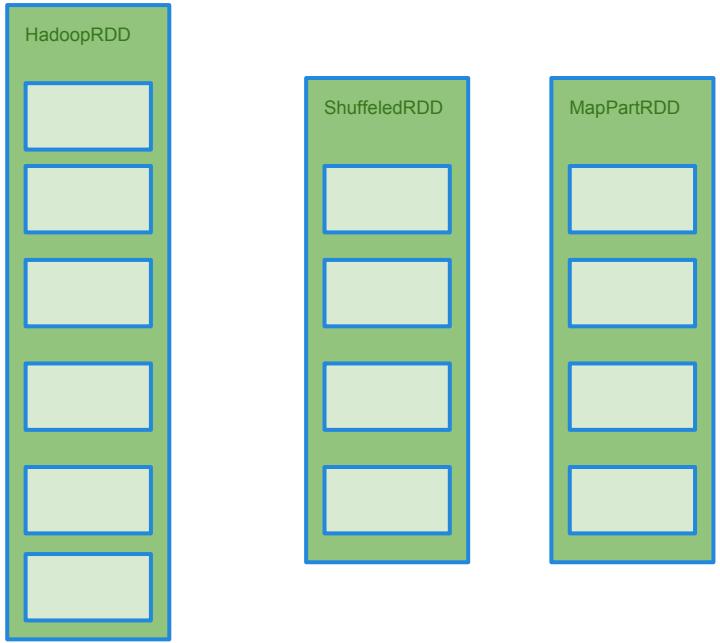
Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```



Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```

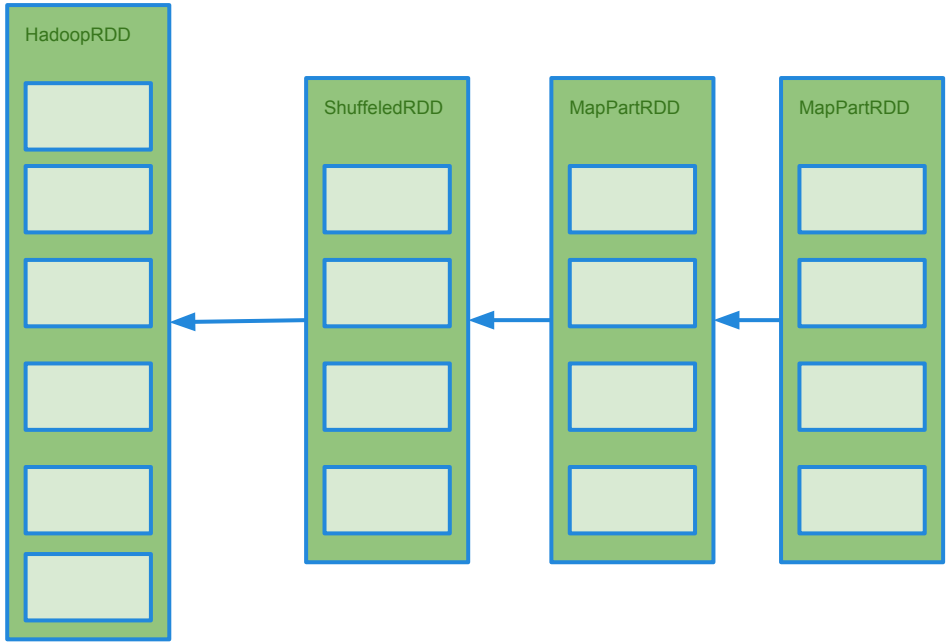


Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```

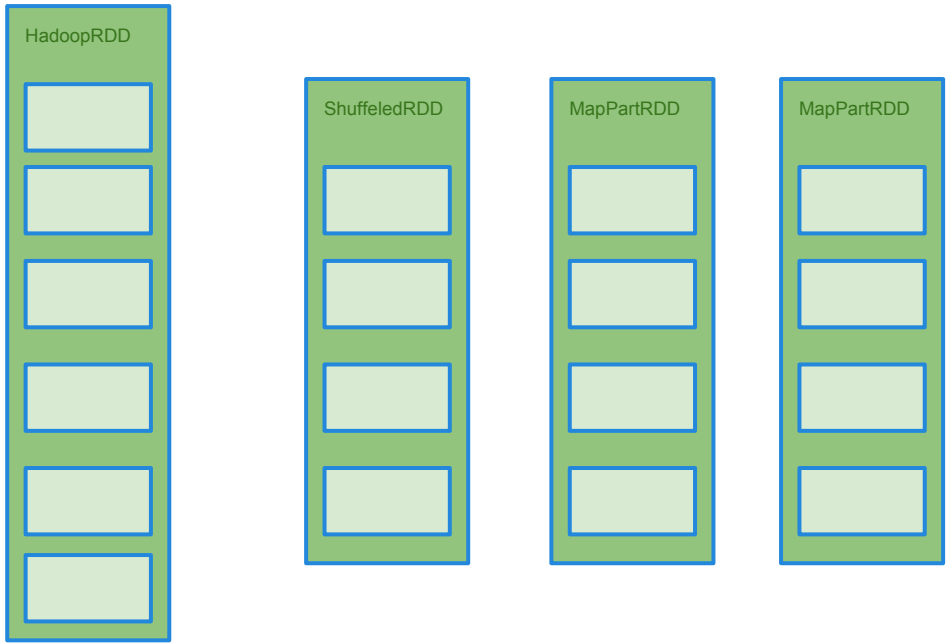
Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```



Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```

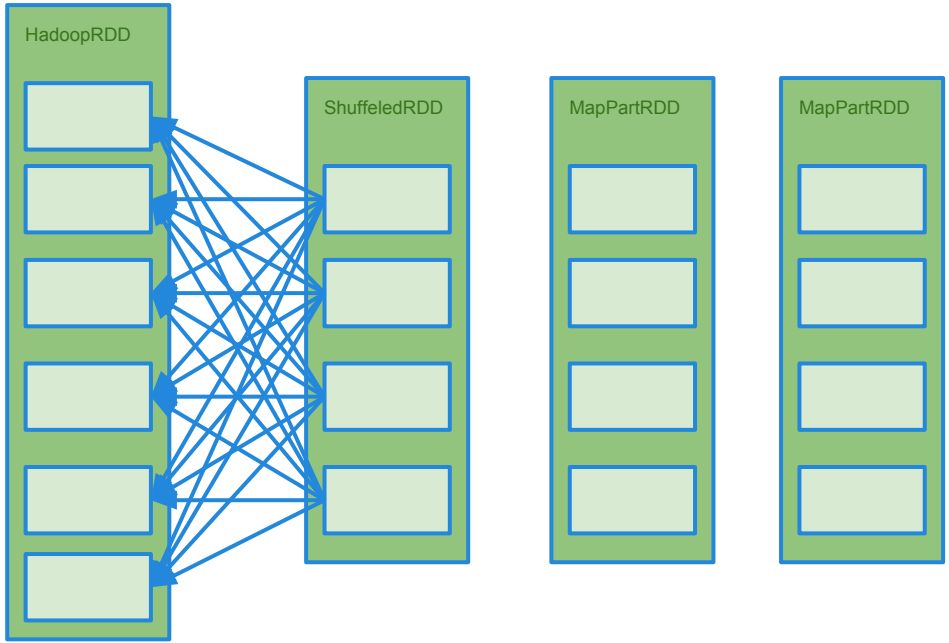


Two types of parent dependencies:

1. narrow dependency
2. wider dependency

Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```

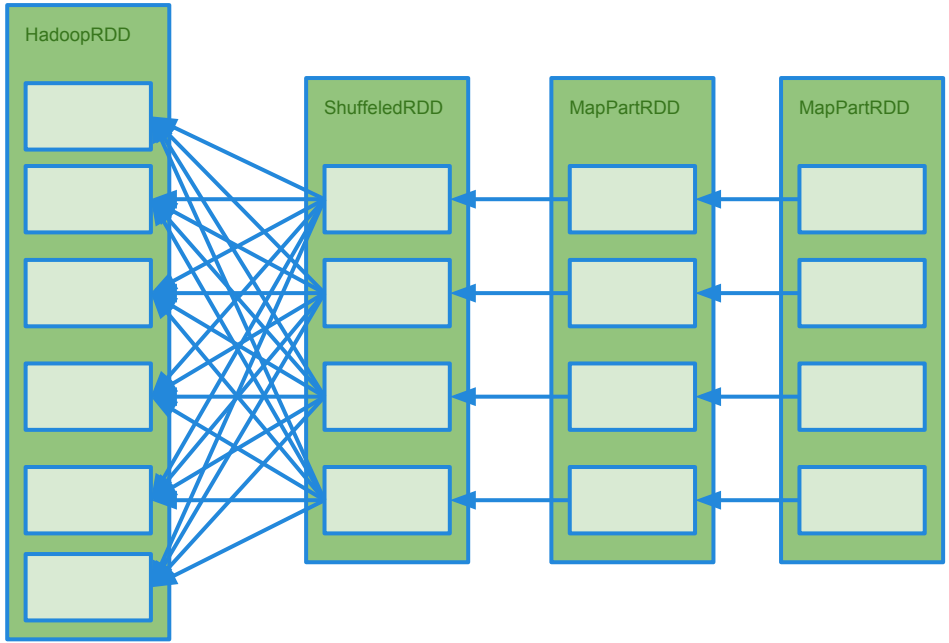


Two types of parent dependencies:

1. narrow dependency
2. wider dependency

Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```

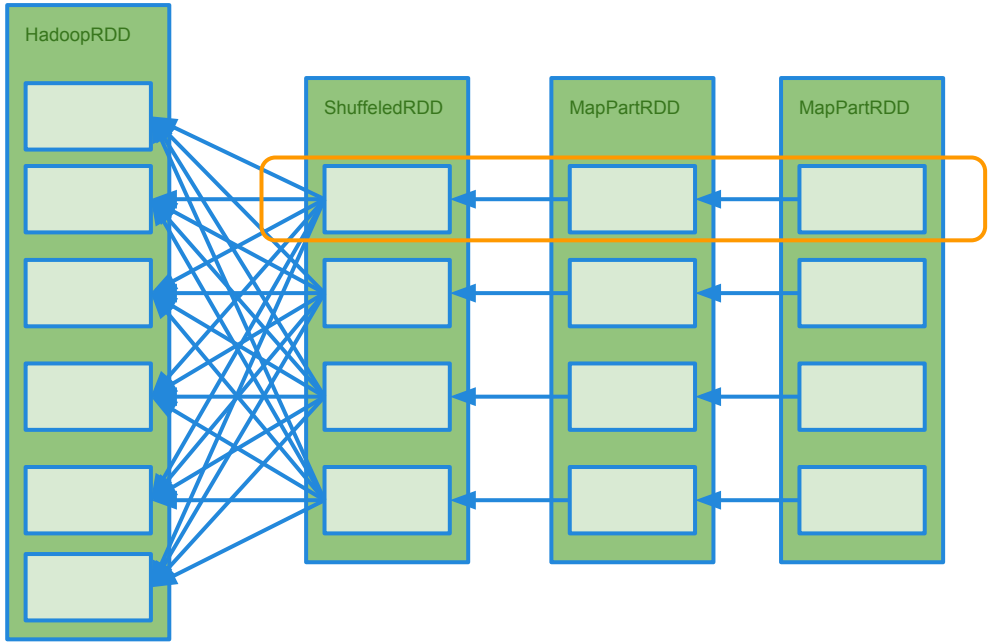


Two types of parent dependencies:

1. narrow dependency
2. wider dependency

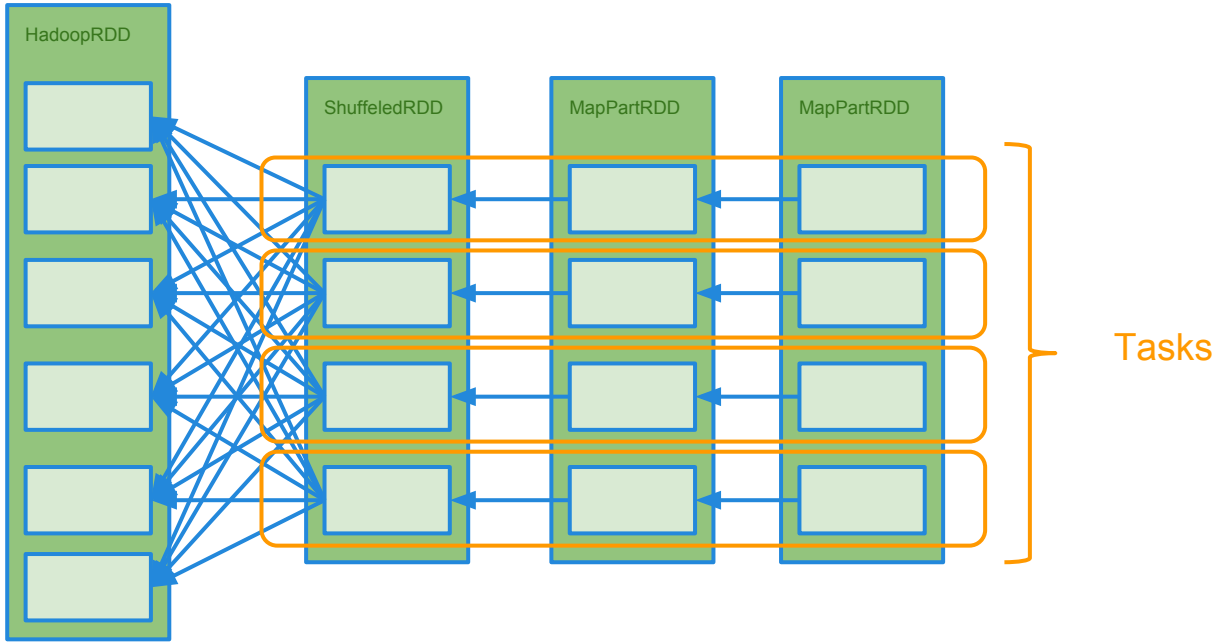
Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```



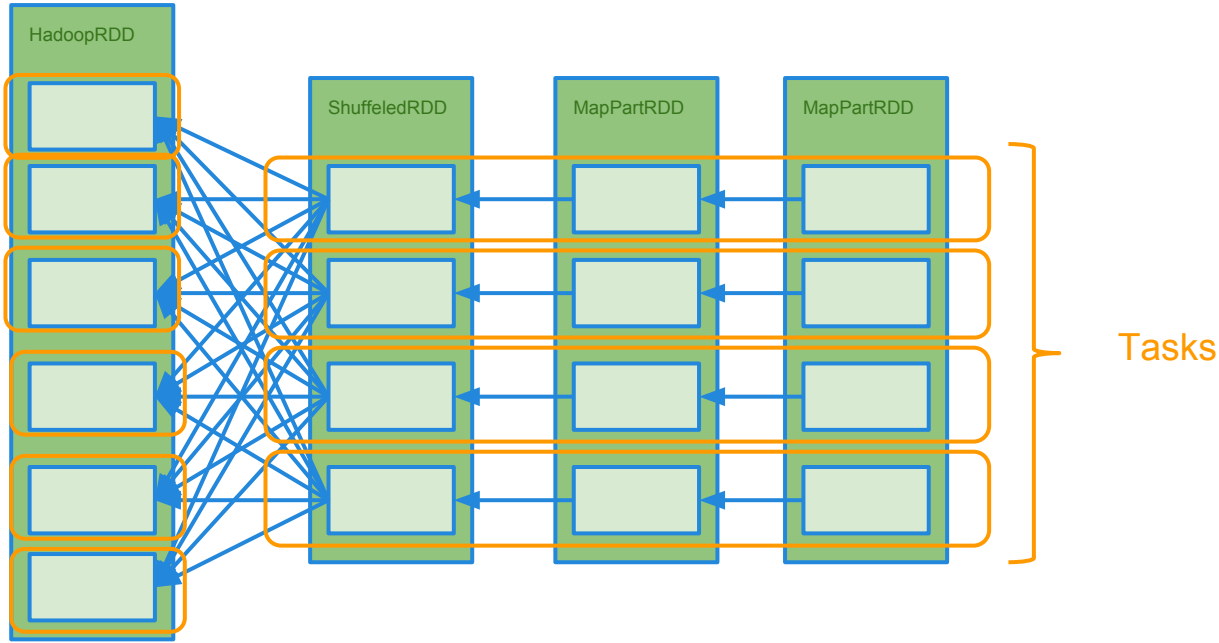
Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```



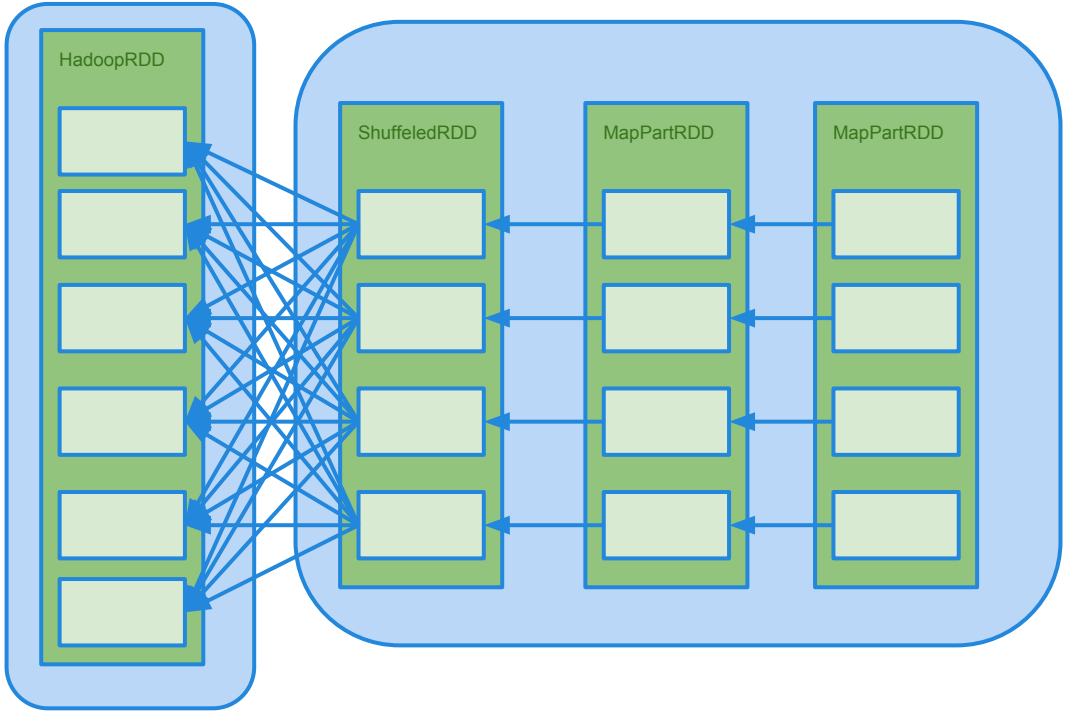
Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```



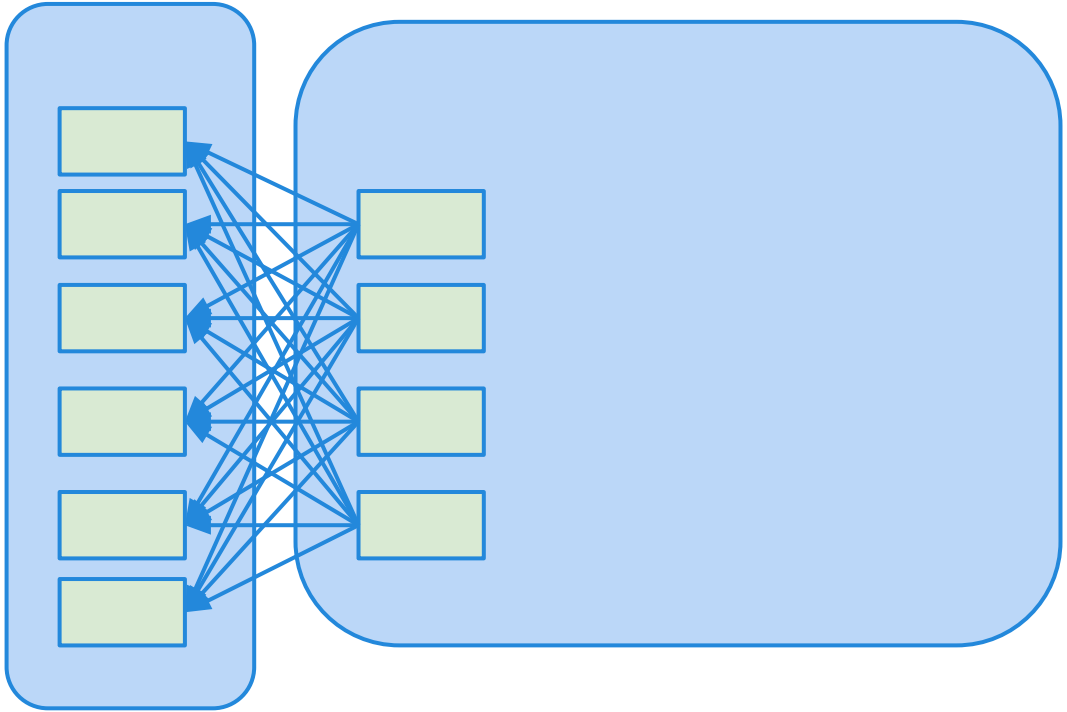
Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```



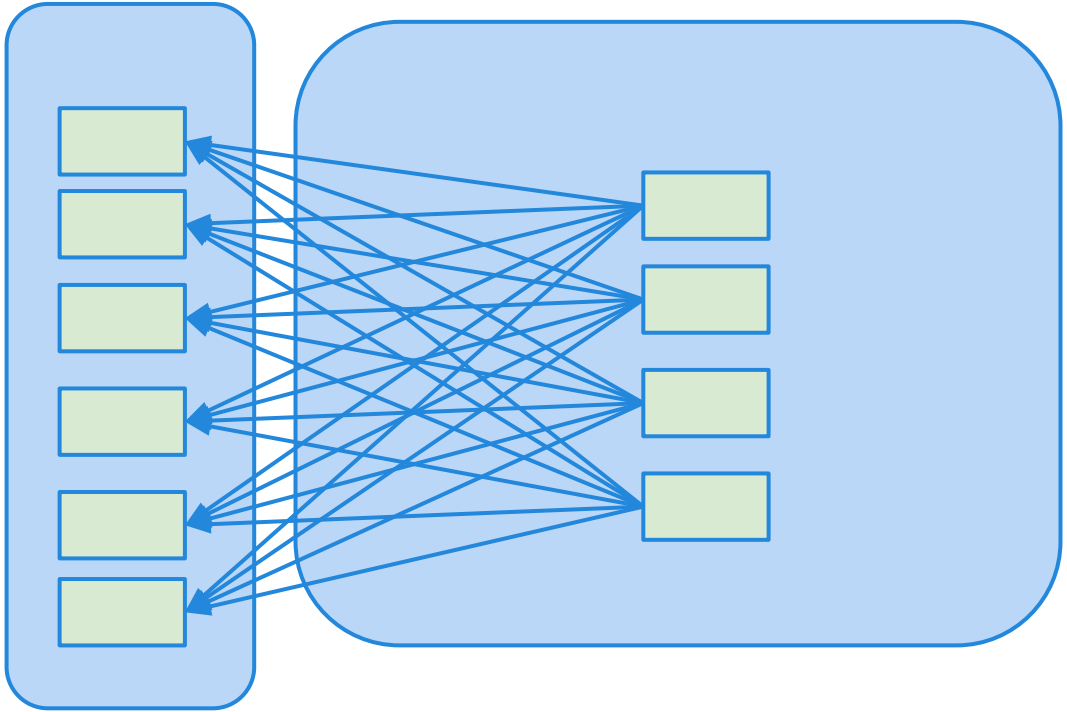
Directed acyclic graph

```
sc.textFile()     .groupBy()     .map { }     .filter { }     .take()     .foreach()
```



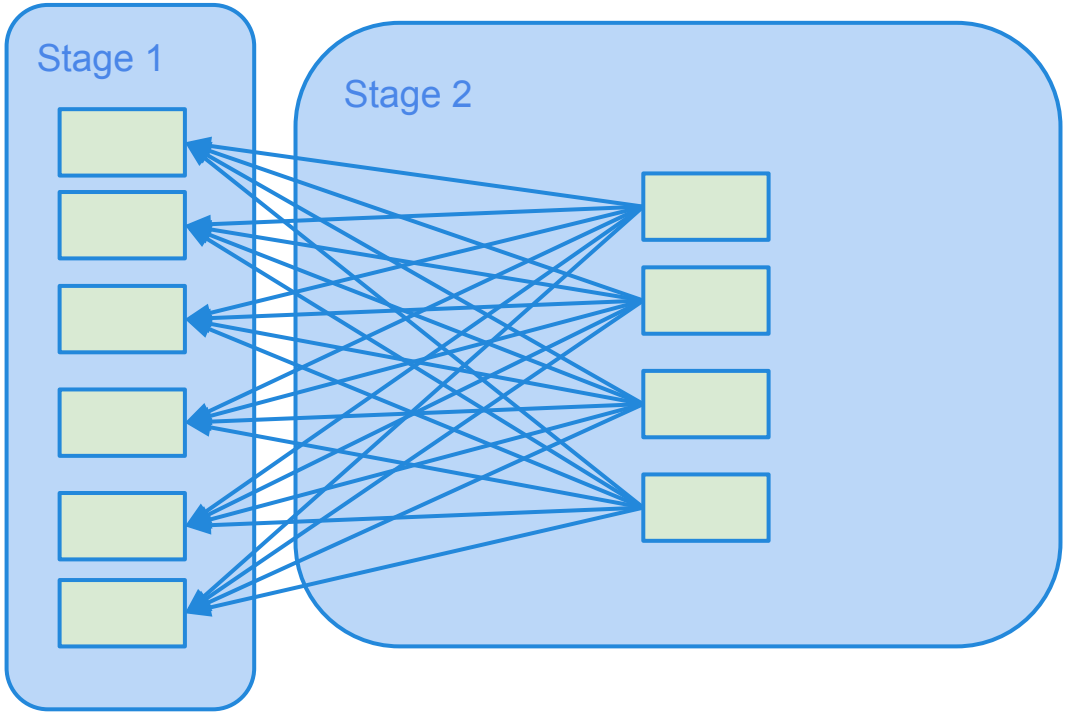
Directed acyclic graph

```
sc.textFile()     .groupBy()     .map { }     .filter { }     .take()     .foreach()
```



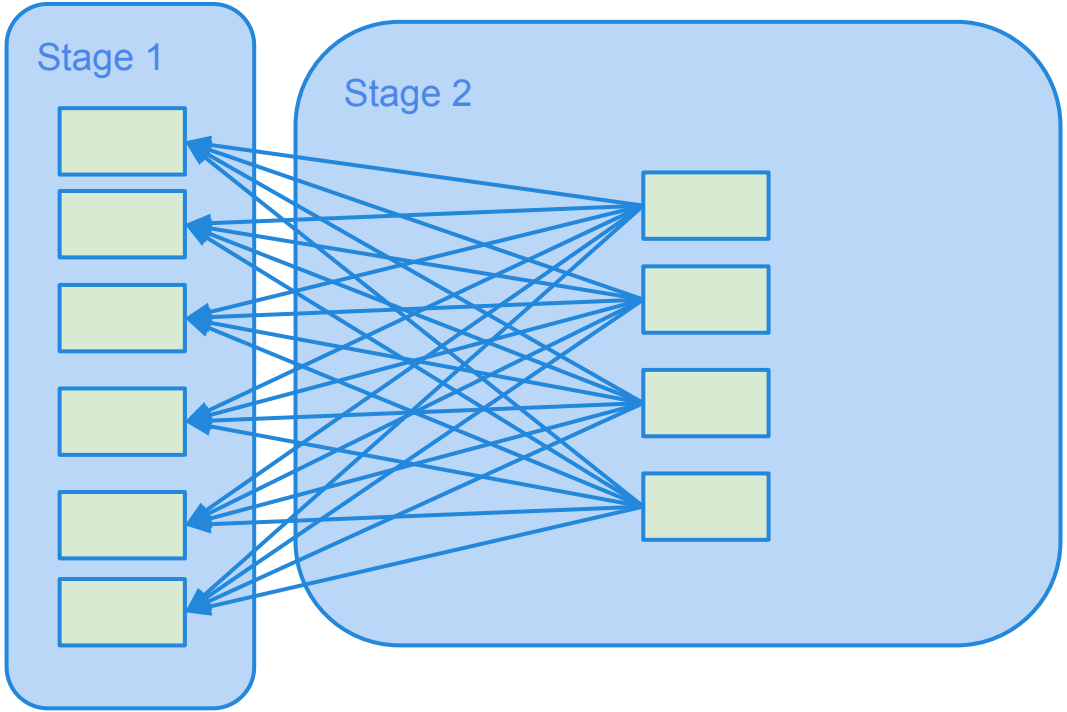
Directed acyclic graph

```
sc.textFile()     .groupBy()     .map { }     .filter { }     .take()     .foreach()
```



Directed acyclic graph

```
sc.textFile()    .groupBy()    .map { }    .filter { }    .take()    .foreach()
```



Two important concepts:

1. shuffle write
2. shuffle read

toDebugString

```
val events = sc.textFile(“hdfs://journal/*”)
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
```

toDebugString

```
val events = sc.textFile(“hdfs://journal/*”)
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }

scala> events.toDebugString
```

toDebugString

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }

scala> events.toDebugString
res5: String =
(4) MapPartitionsRDD[22] at filter at <console>:50 []
 |   MapPartitionsRDD[21] at map at <console>:49 []
 |   ShuffledRDD[20] at groupBy at <console>:48 []
 +-(6)  HadoopRDD[17] at textFile at <console>:47 []
```

What is a RDD?

RDD needs to hold 3 chunks of information in order to do its work:

1. pointer to his parent
2. how its internal data is partitioned
3. how evaluate its internal data

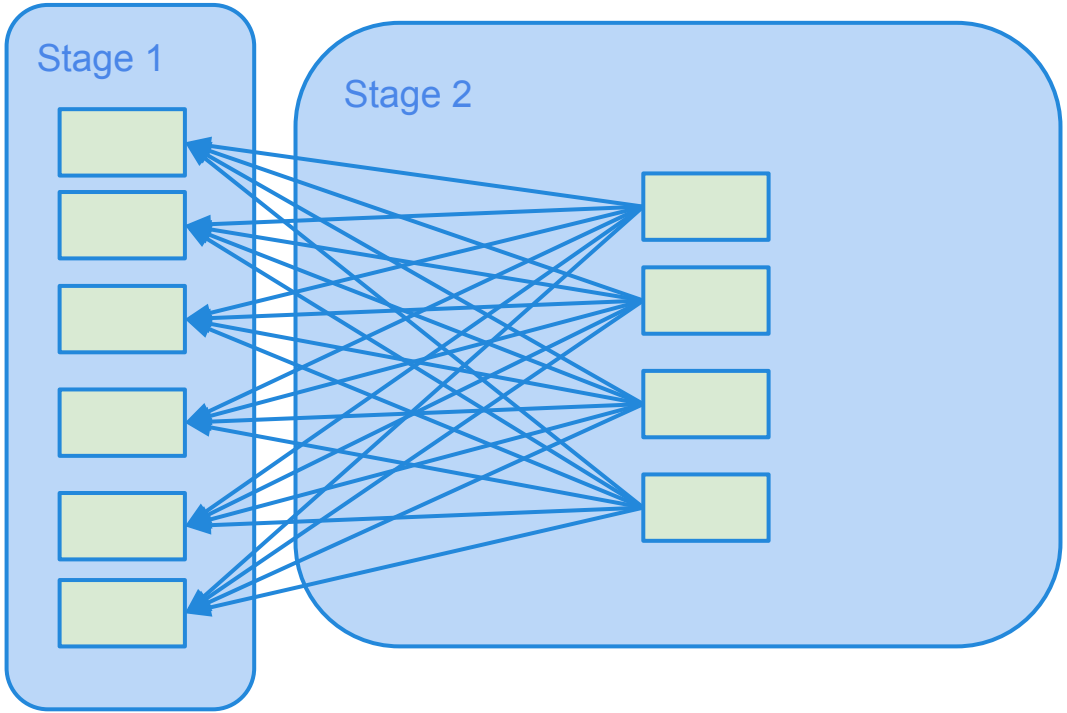
What is a RDD?

RDD needs to hold 3 chunks of information in order to do its work:

1. pointer to his parent
2. how its internal data is partitioned
3. how evaluate its internal data

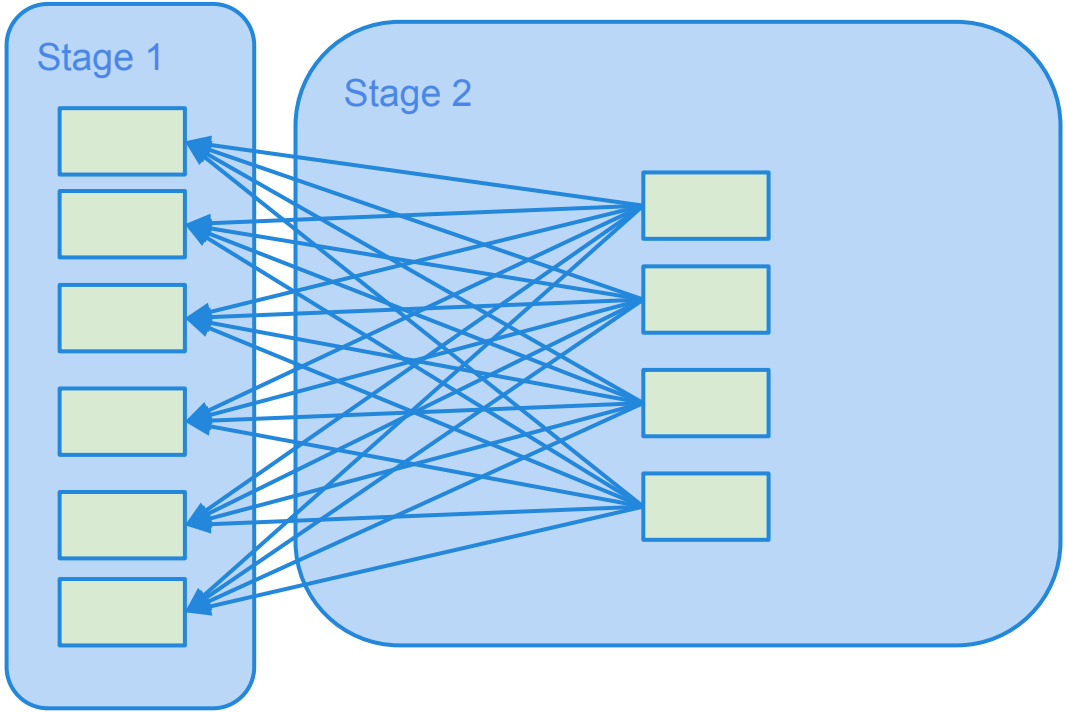
Running Job aka materializing DAG

```
sc.textFile()     .groupBy()     .map { }     .filter { }
```

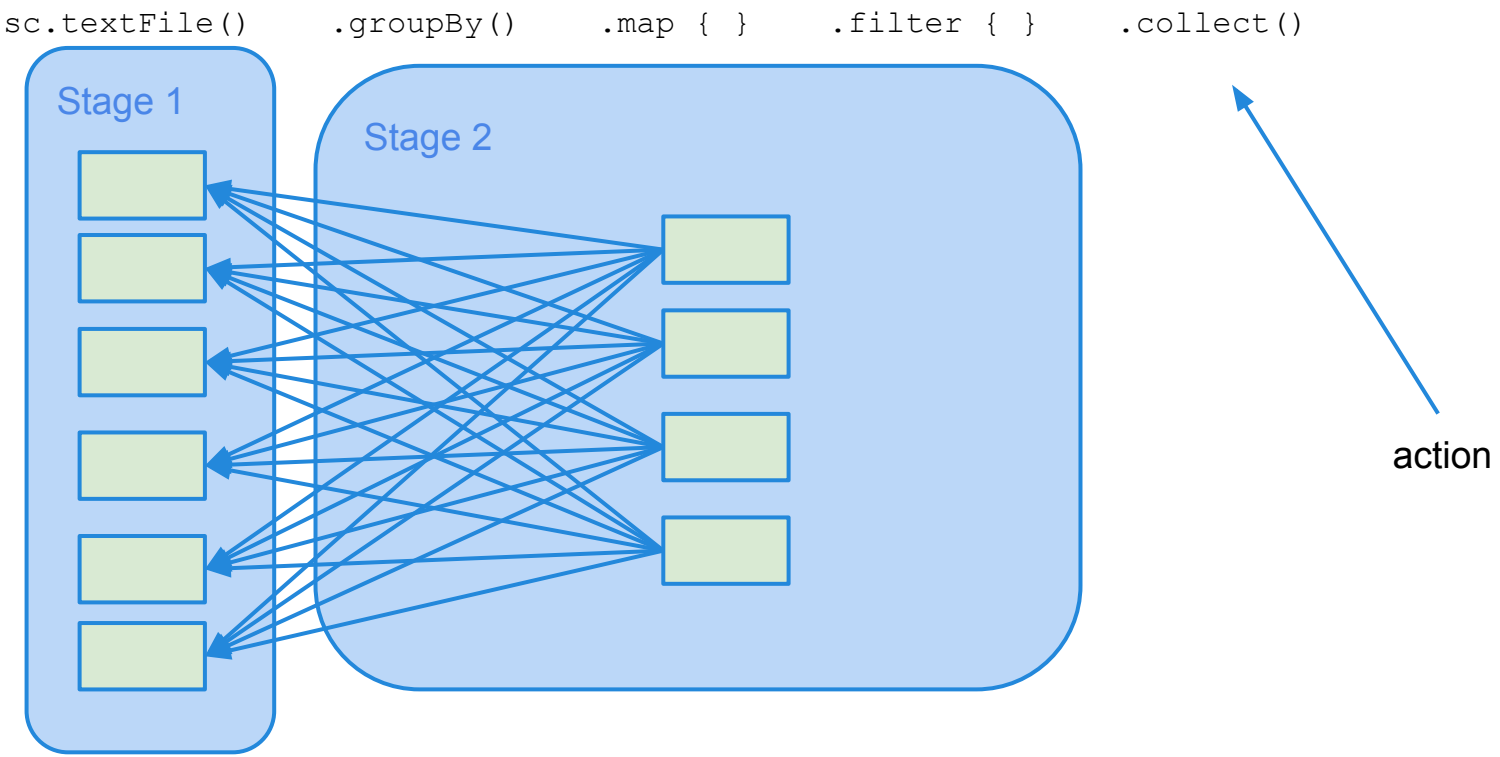


Running Job aka materializing DAG

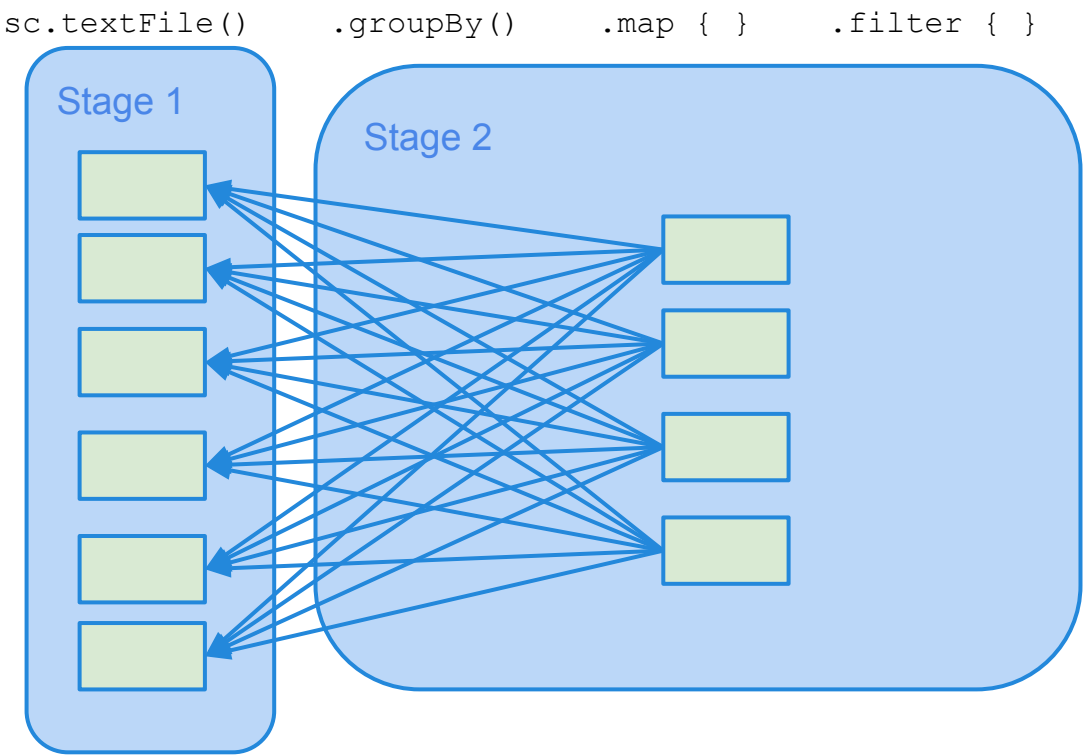
```
sc.textFile()     .groupBy()     .map { }     .filter { }     .collect()
```



Running Job aka materializing DAG



Running Job aka materializing DAG



action

Actions are implemented using **sc.runJob** method

Running Job aka materializing DAG

```
/**
* Run a function on a given set of partitions in an RDD and return the results as an array.
*/
def runJob[T, U](



  ): Array[U]
```

Running Job aka materializing DAG

```
/**
* Run a function on a given set of partitions in an RDD and return the results as an array.
*/
def runJob[T, U](
  rdd: RDD[T],


  ): Array[U]
```

Running Job aka materializing DAG

```
/**
* Run a function on a given set of partitions in an RDD and return the results as an array.
*/
def runJob[T, U](
  rdd: RDD[T],
  partitions: Seq[Int],

  ): Array[U]
```

Running Job aka materializing DAG

```
/**
* Run a function on a given set of partitions in an RDD and return the results as an array.
*/
def runJob[T, U](
  rdd: RDD[T],
  partitions: Seq[Int],
  func: Iterator[T] => U,
  ): Array[U]
```

Running Job aka materializing DAG

```scala
/**
* Return an array that contains all of the elements in this RDD.
*/
def collect(): Array[T] = {
    val results = sc.runJob(this, (iter: Iterator[T]) => iter.toArray)
    Array.concat(results: _*)
}
```

Multiple jobs for single action

```
/**
* Take the first num elements of the RDD. It works by first scanning one partition, and use the  results from that partition to estimate the number of additional partitions needed to satisfy the limit.
*/
def take(num: Int): Array[T] = {
    (….)
     val left = num - buf.size
     val res = sc.runJob(this, (it: Iterator[T]) => it.take(left).toArray, p, allowLocal = true)
    (….)
     res.foreach(buf ++= _.take(num - buf.size))
     partsScanned += numPartsToTry
     (….)
     buf.toArray
}
```

Lets test what we've learned

Towards efficiency

```
val events = sc.textFile(“hdfs://journal/*”)
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }
```

Towards efficiency

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }

scala> events.toDebugString
(4) MapPartitionsRDD[22] at filter at <console>:50 []
 |   MapPartitionsRDD[21] at map at <console>:49 []
 |   ShuffledRDD[20] at groupBy at <console>:48 []
 +-(6)  HadoopRDD[17] at textFile at <console>:47 []
```

Towards efficiency

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {
        case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1)
    }

scala> events.toDebugString
(4) MapPartitionsRDD[22] at filter at <console>:50 []
 |   MapPartitionsRDD[21] at map at <console>:49 []
 |   ShuffledRDD[20] at groupBy at <console>:48 []
 +-(6)  HadoopRDD[17] at textFile at <console>:47 []

events.count
```

Stage 1

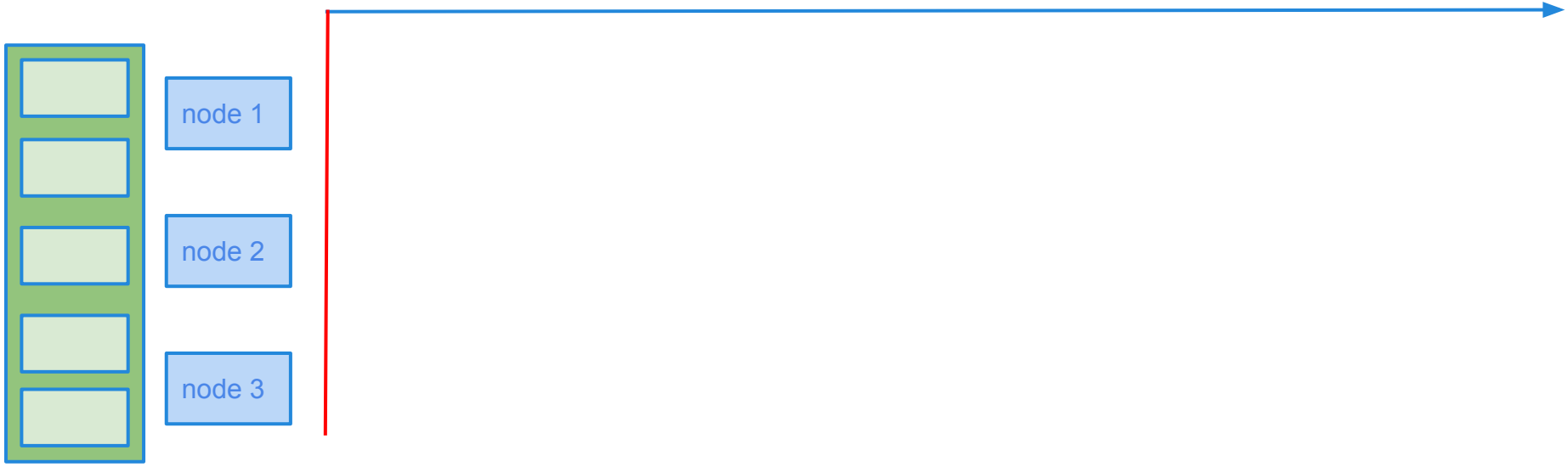
```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```

Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```

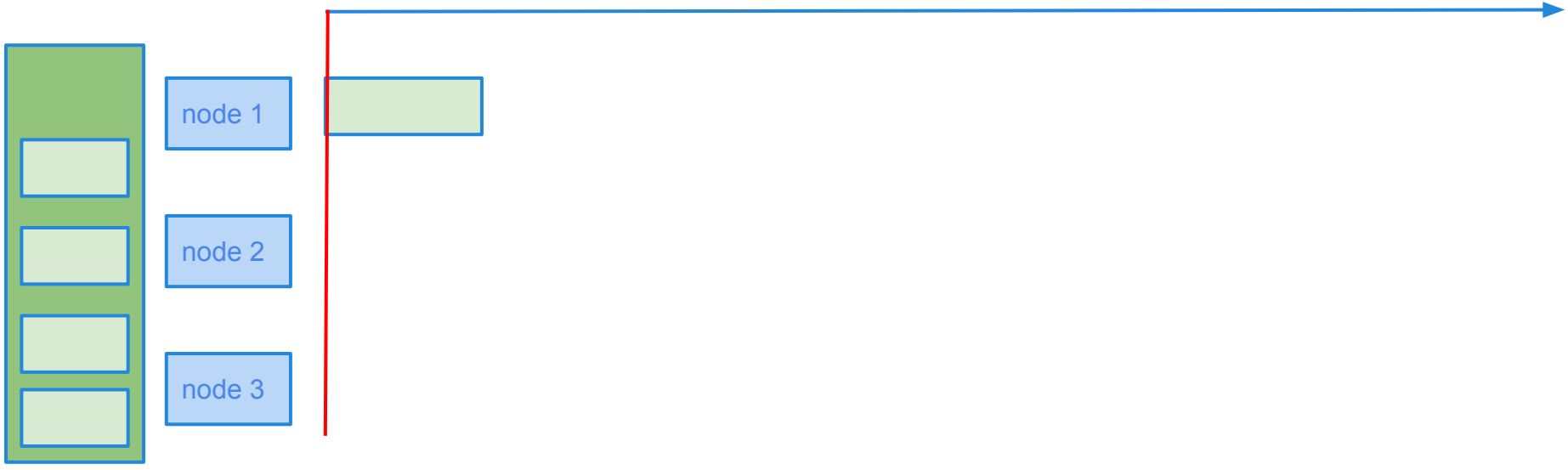
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



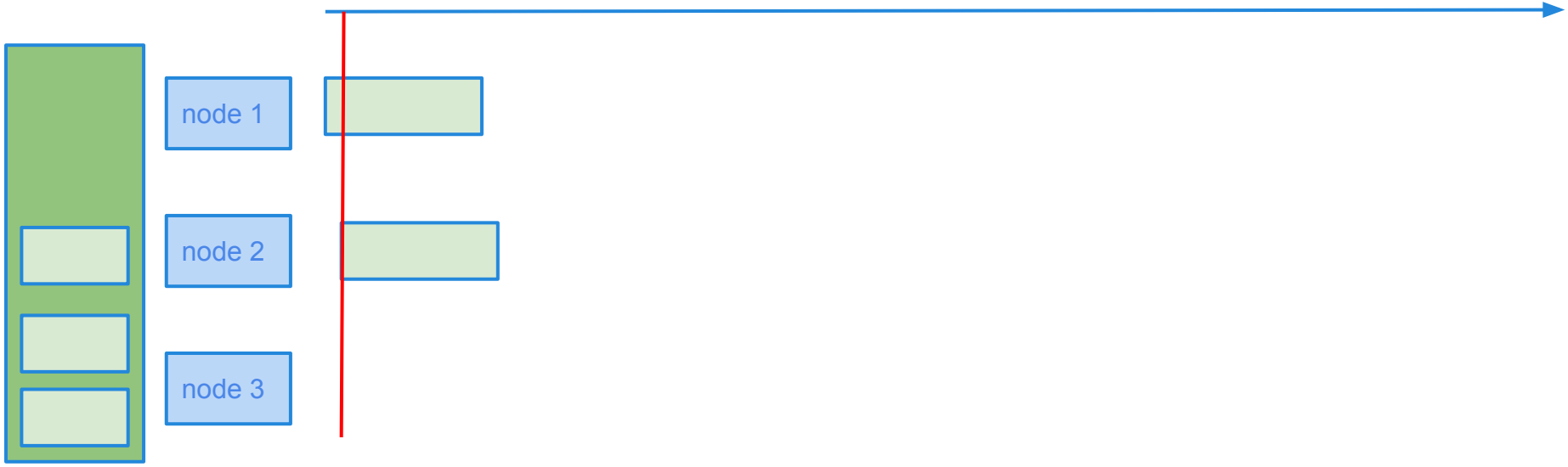
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



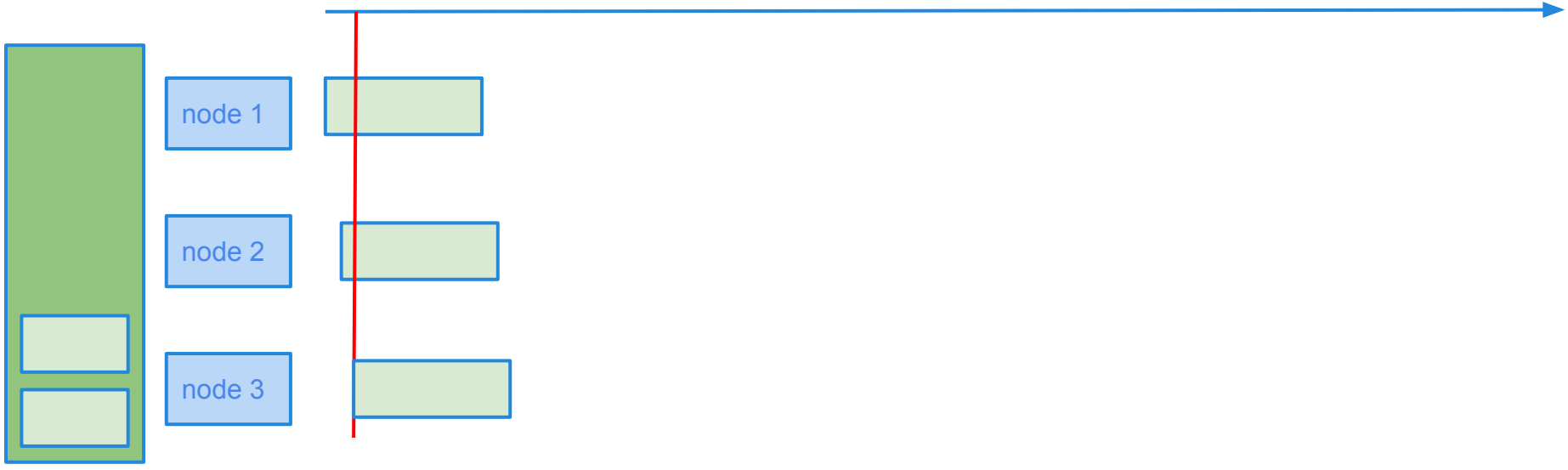
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



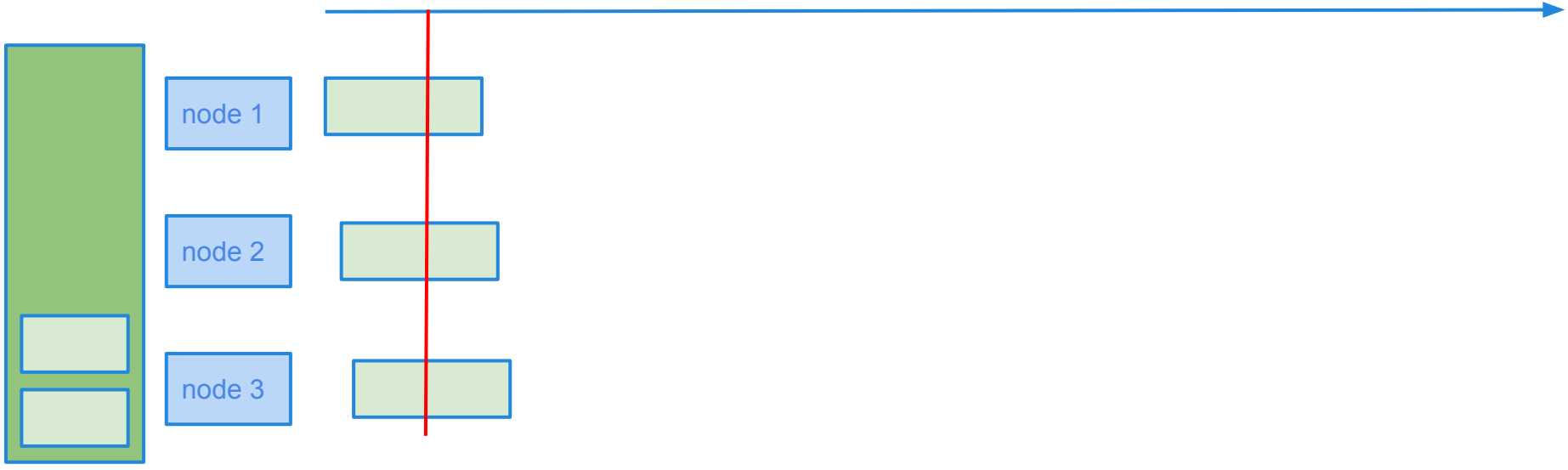
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



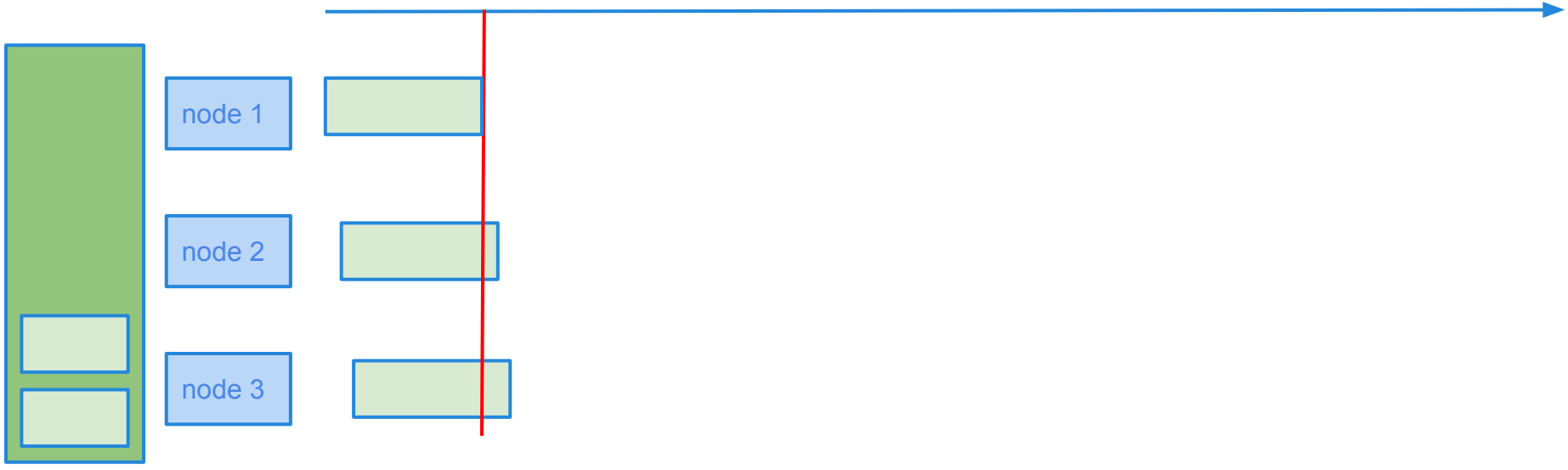
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



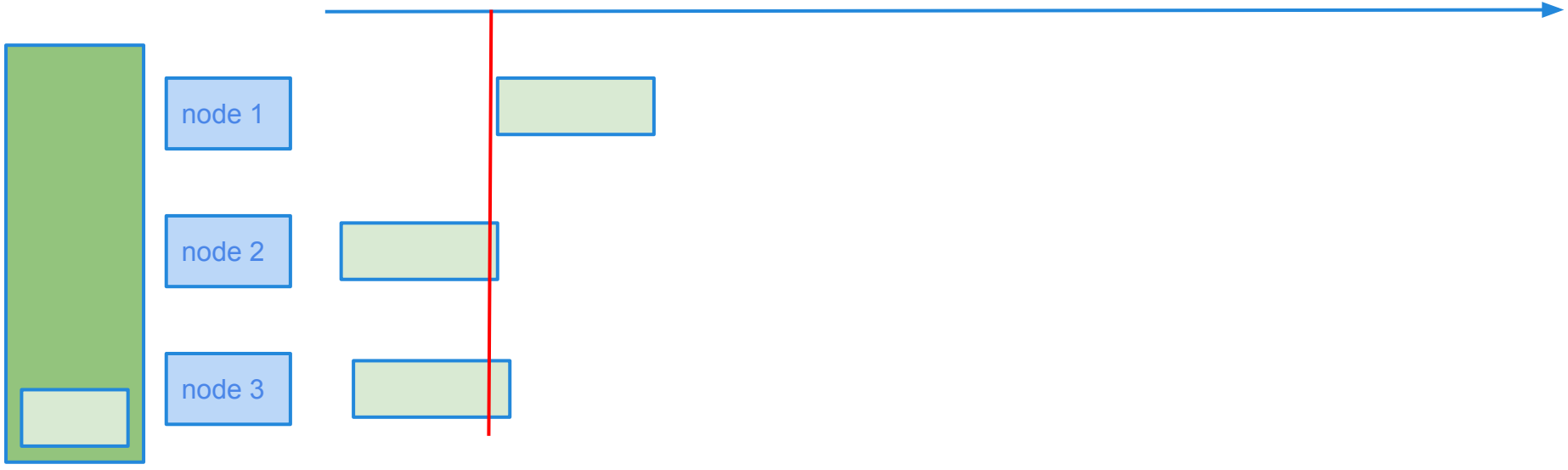
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



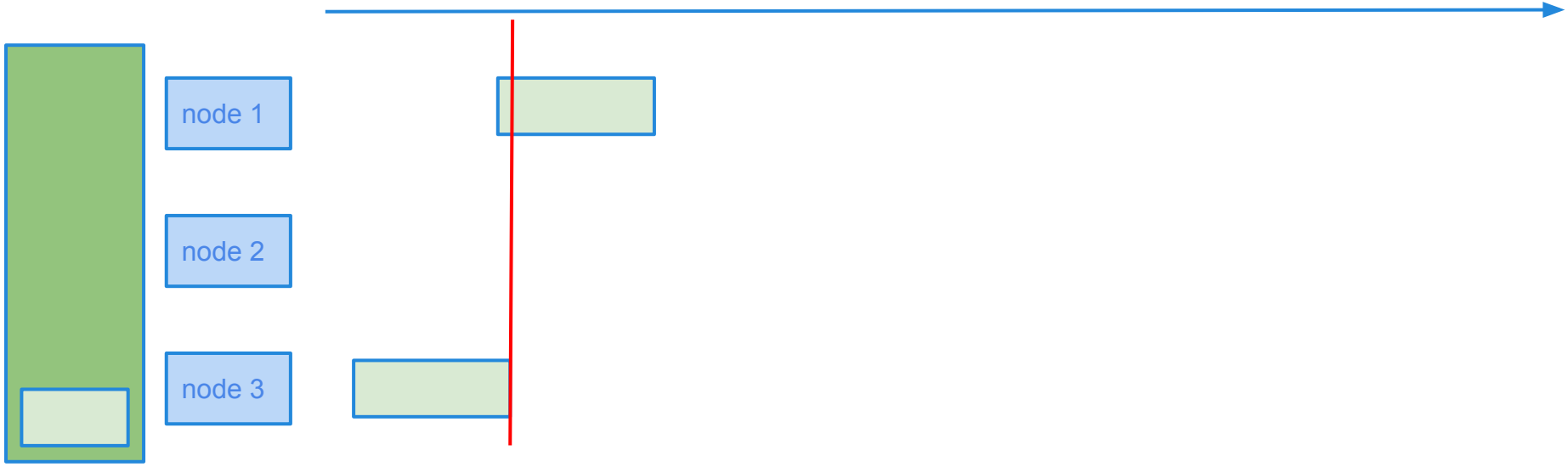
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



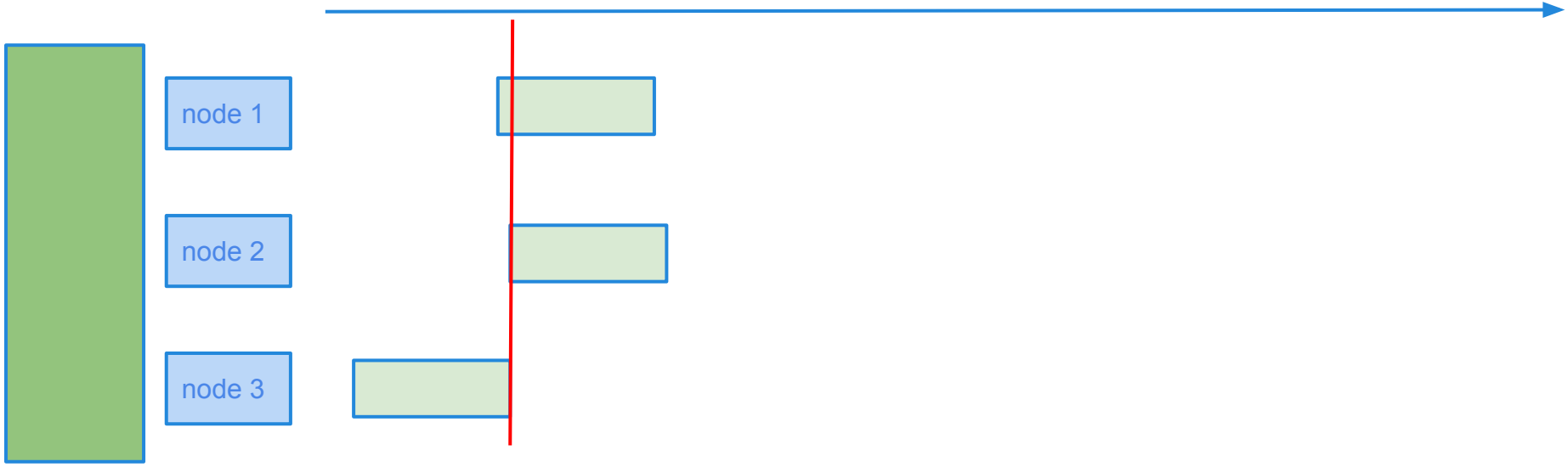
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



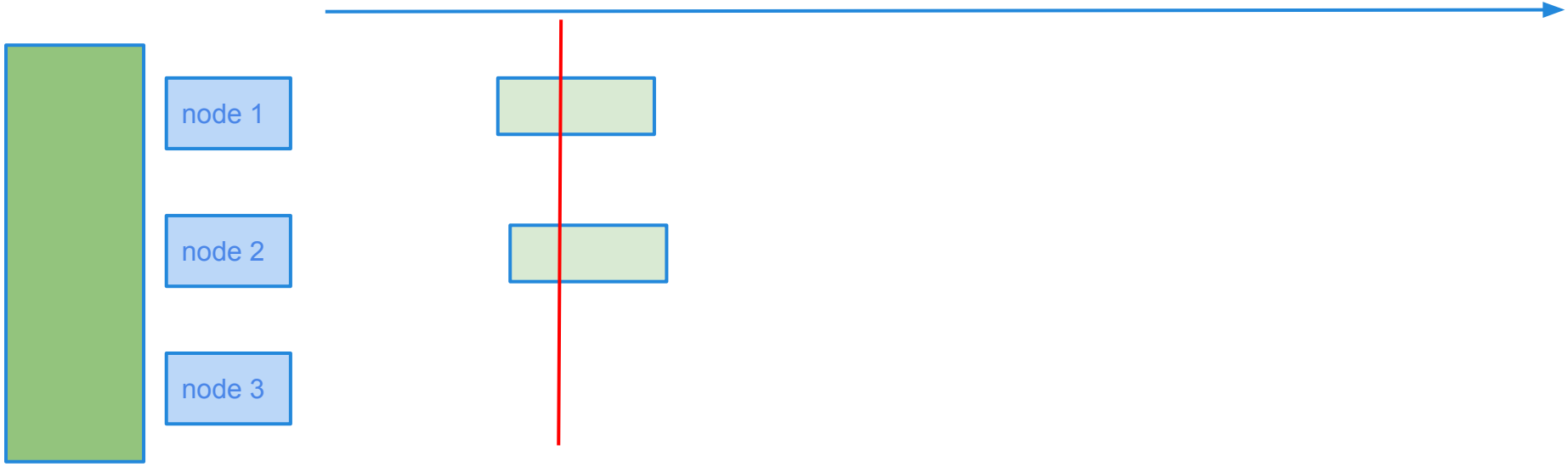
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



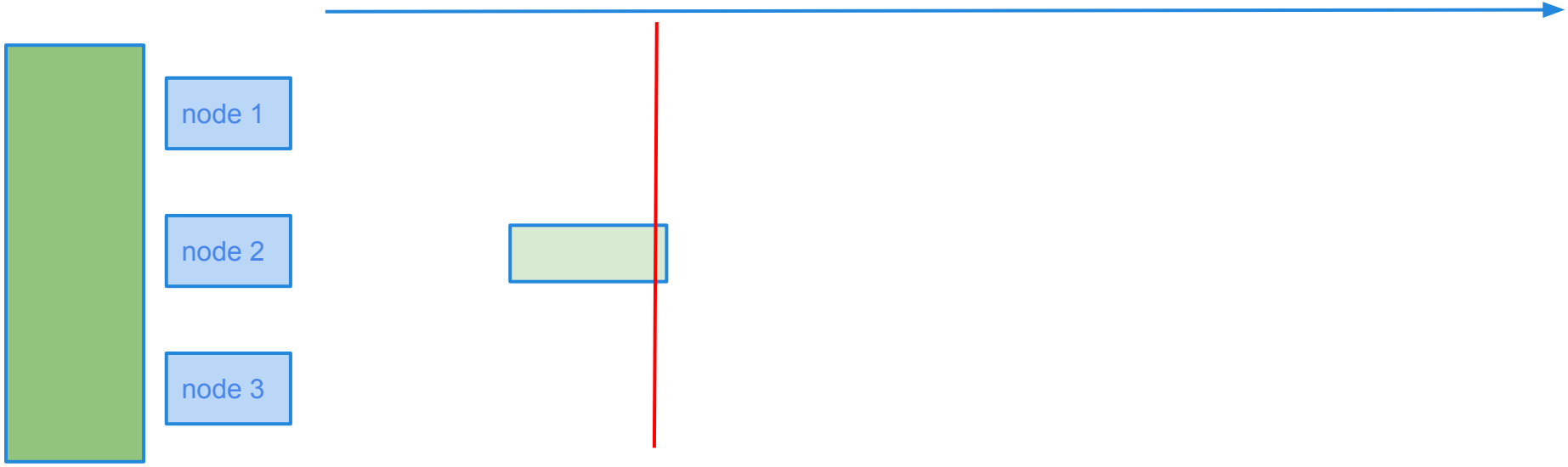
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



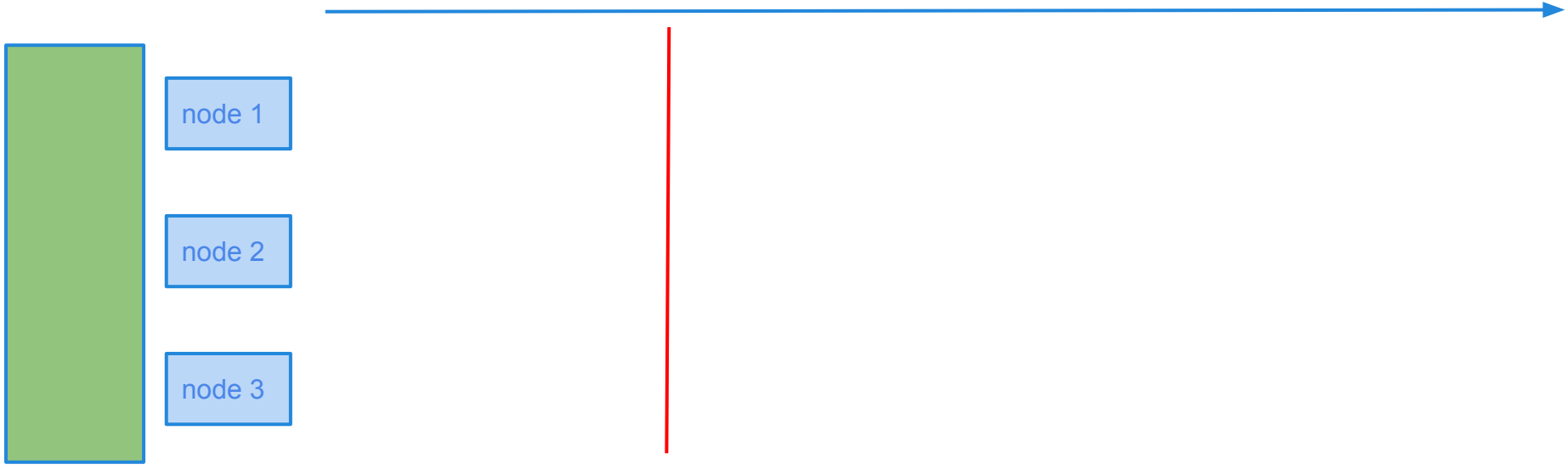
Stage 1

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



Stage 1

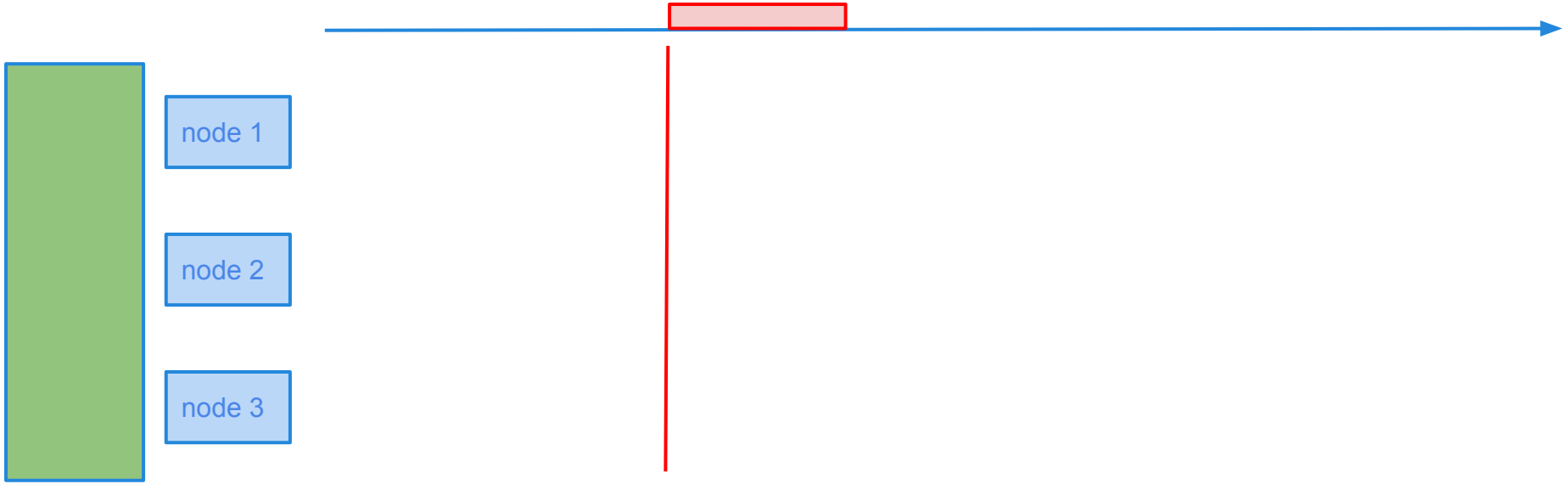
```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```




Everyday I'm Shuffling

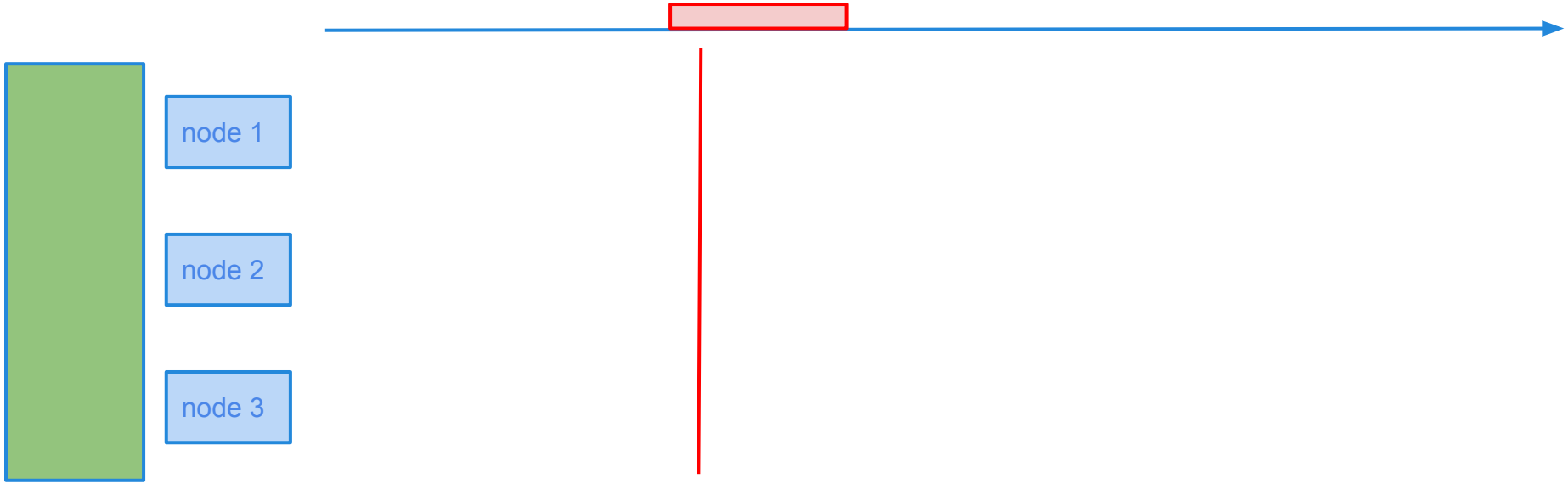
Everyday I'm Shuffling

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



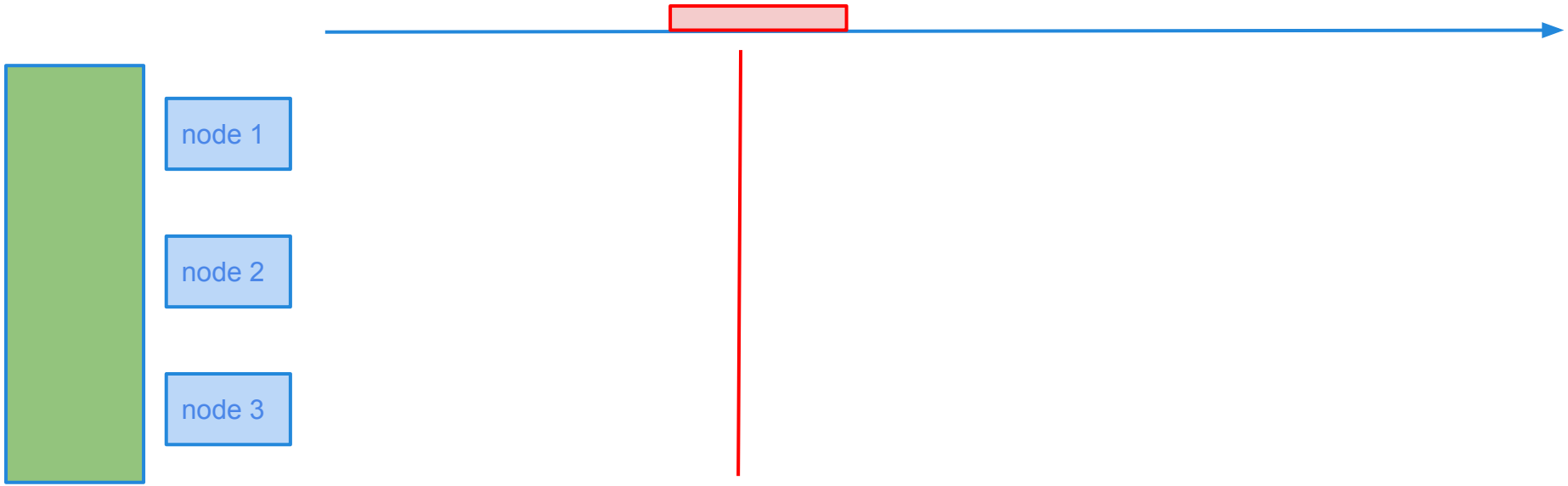
Everyday I'm Shuffling

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



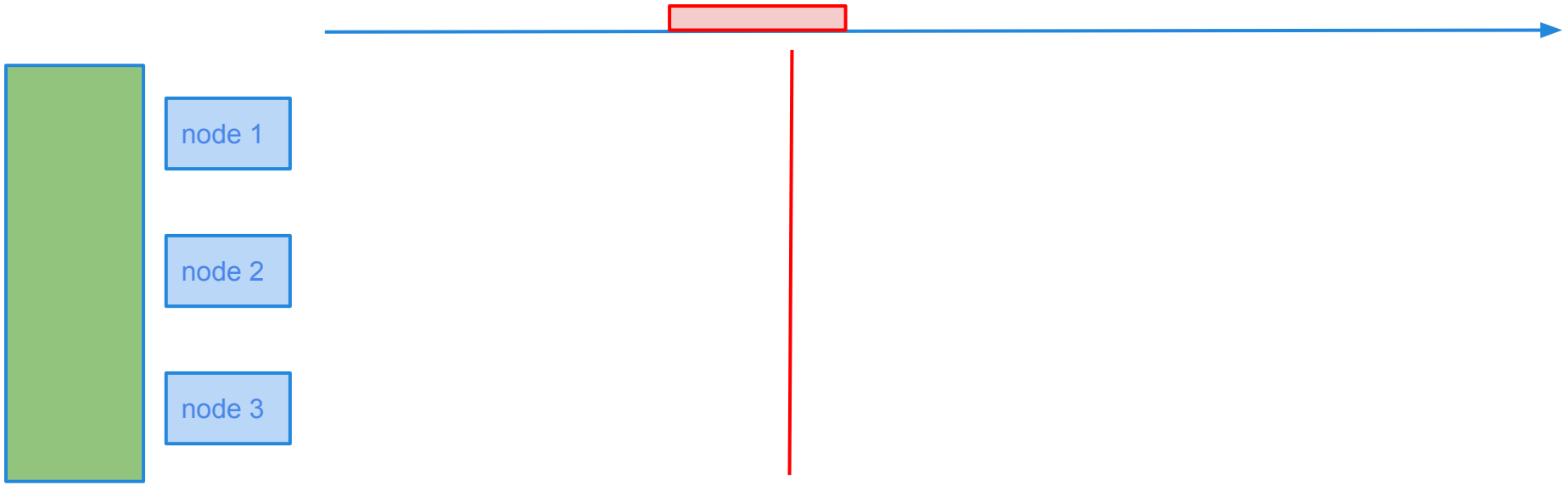
Everyday I'm Shuffling

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



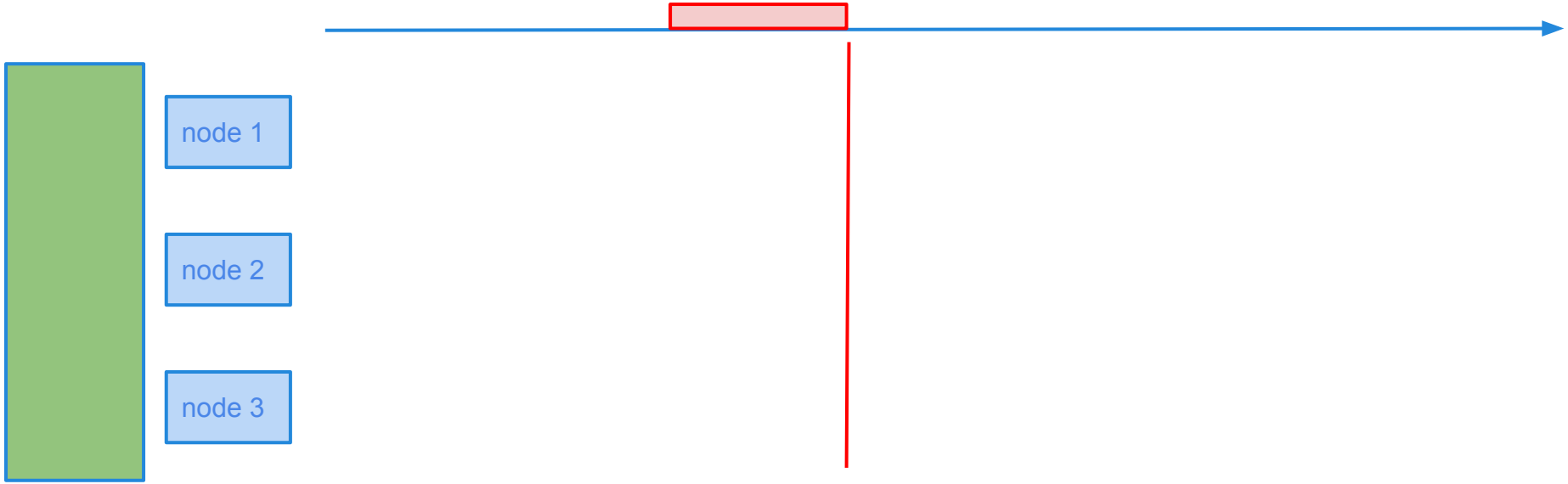
Everyday I'm Shuffling

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



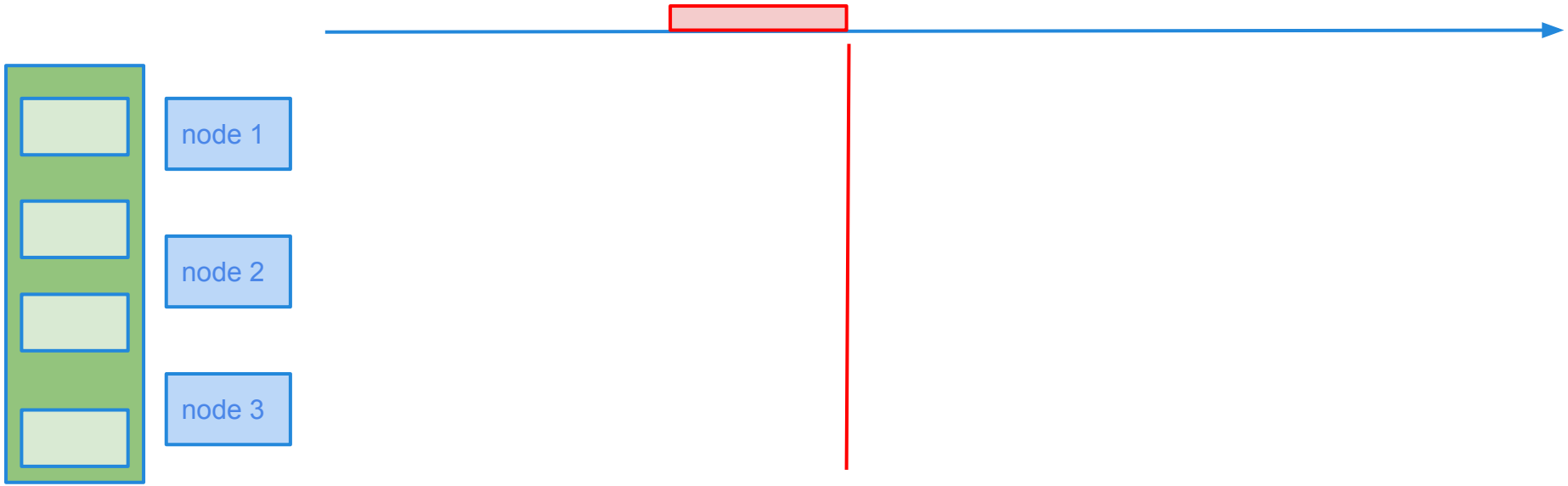
Everyday I'm Shuffling

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



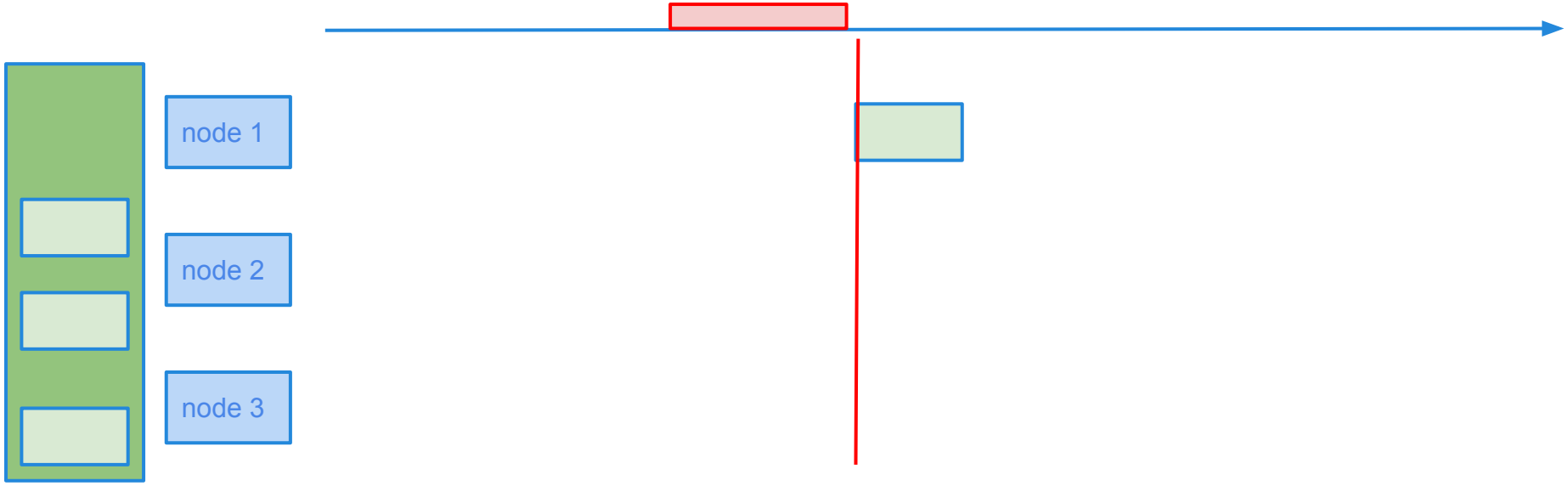
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



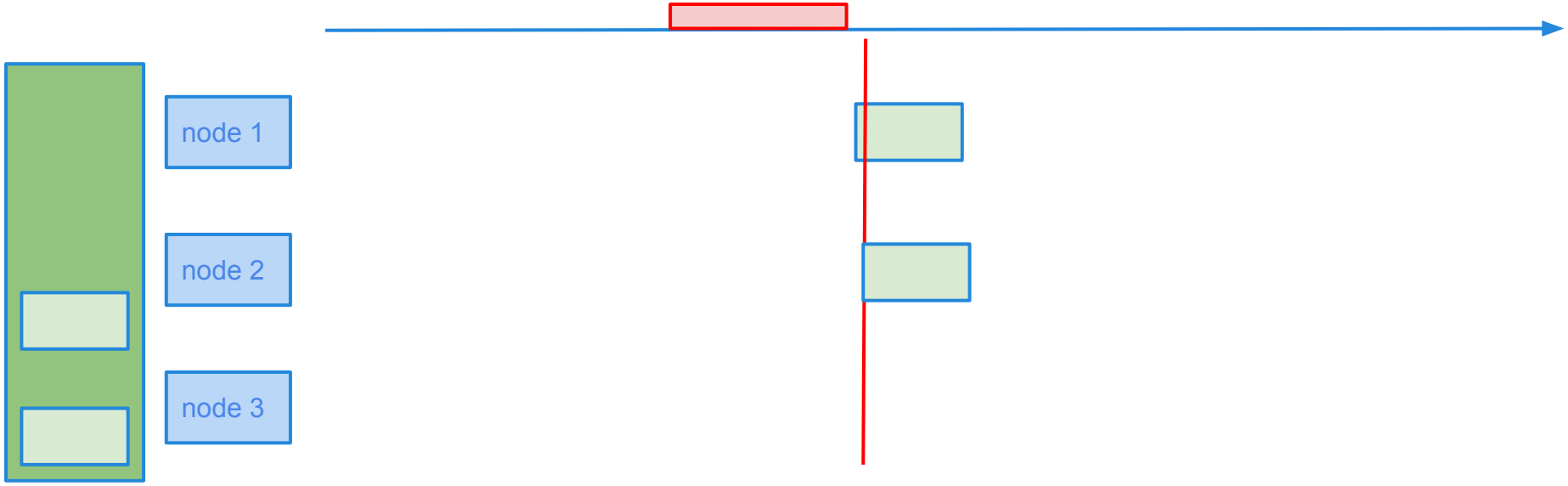
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



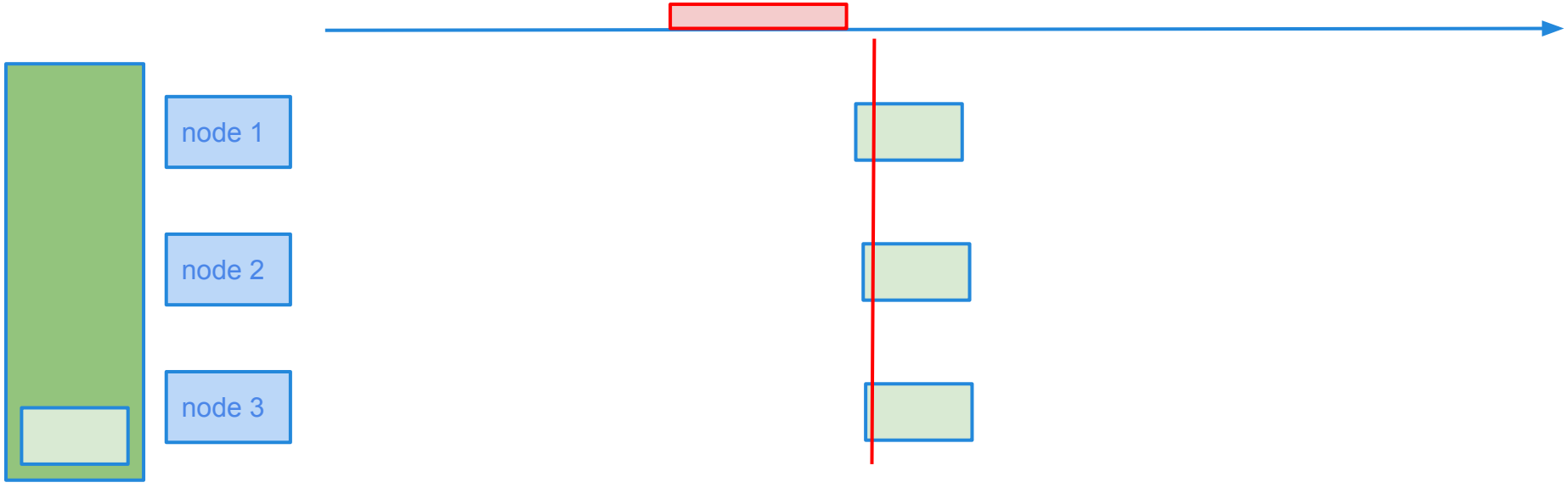
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



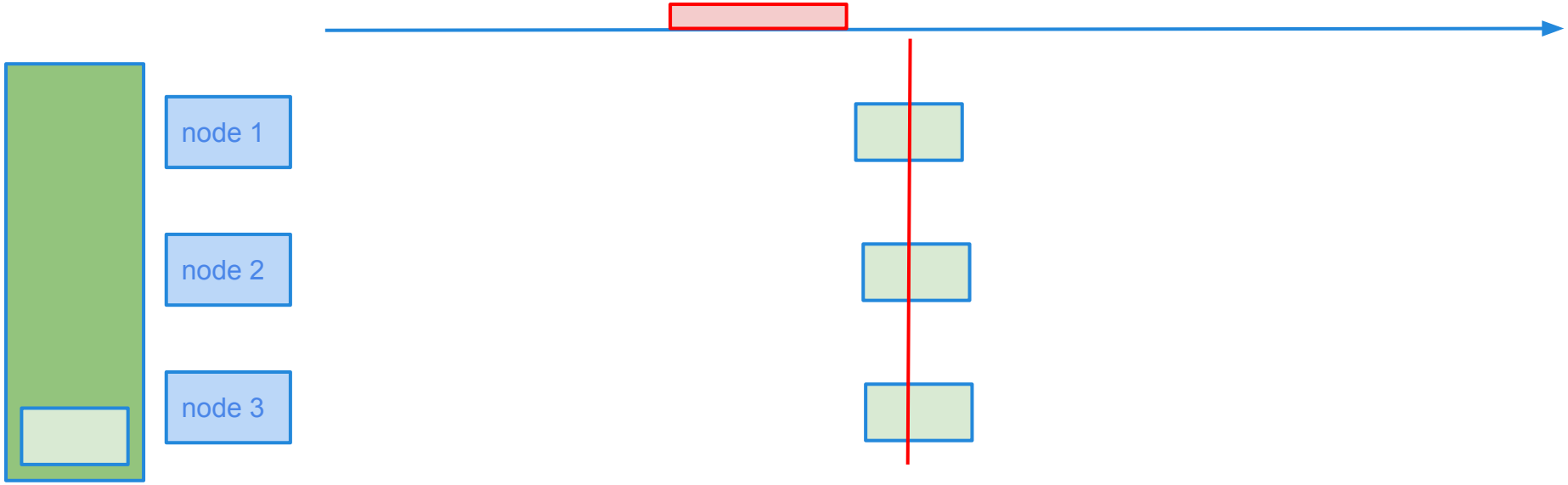
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



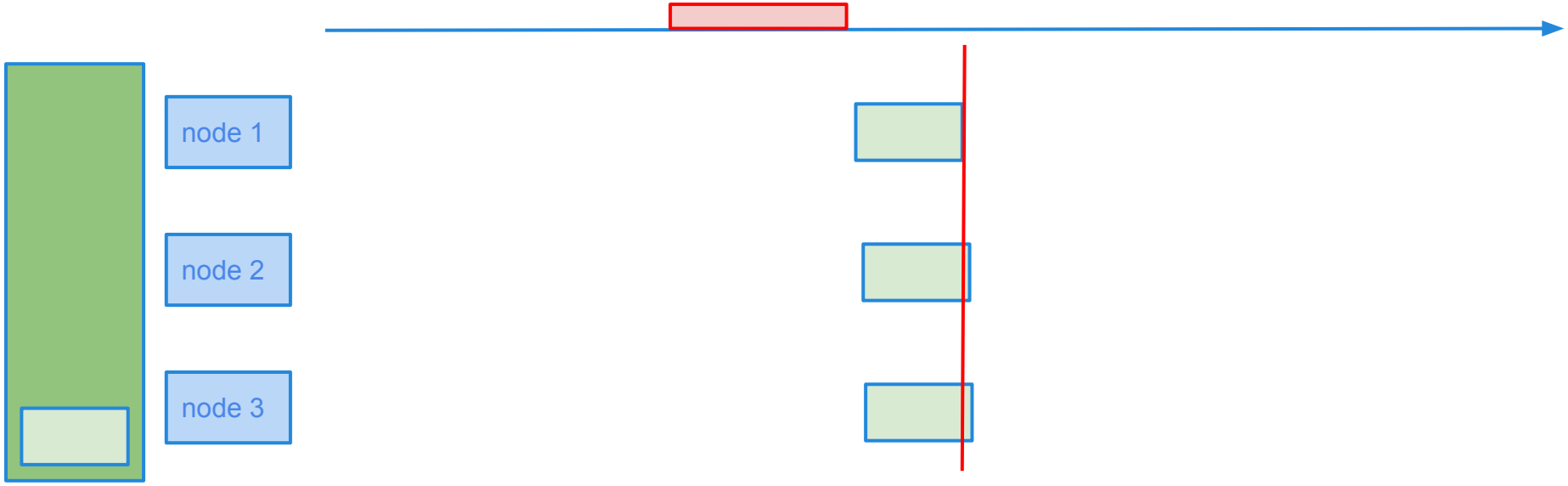
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



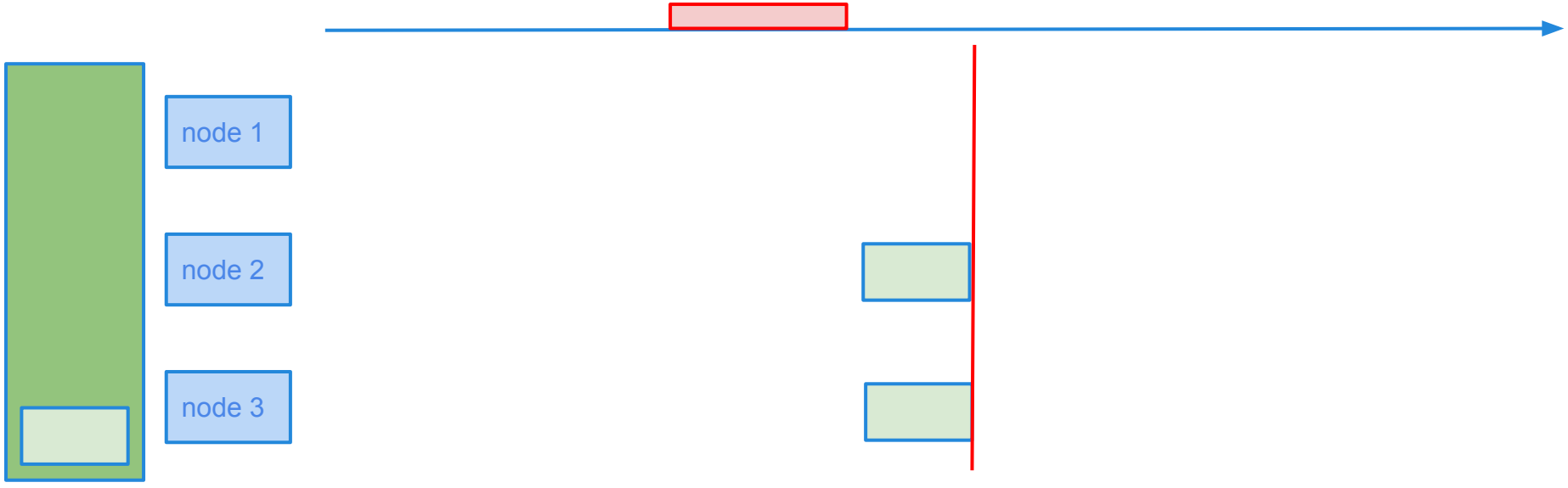
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



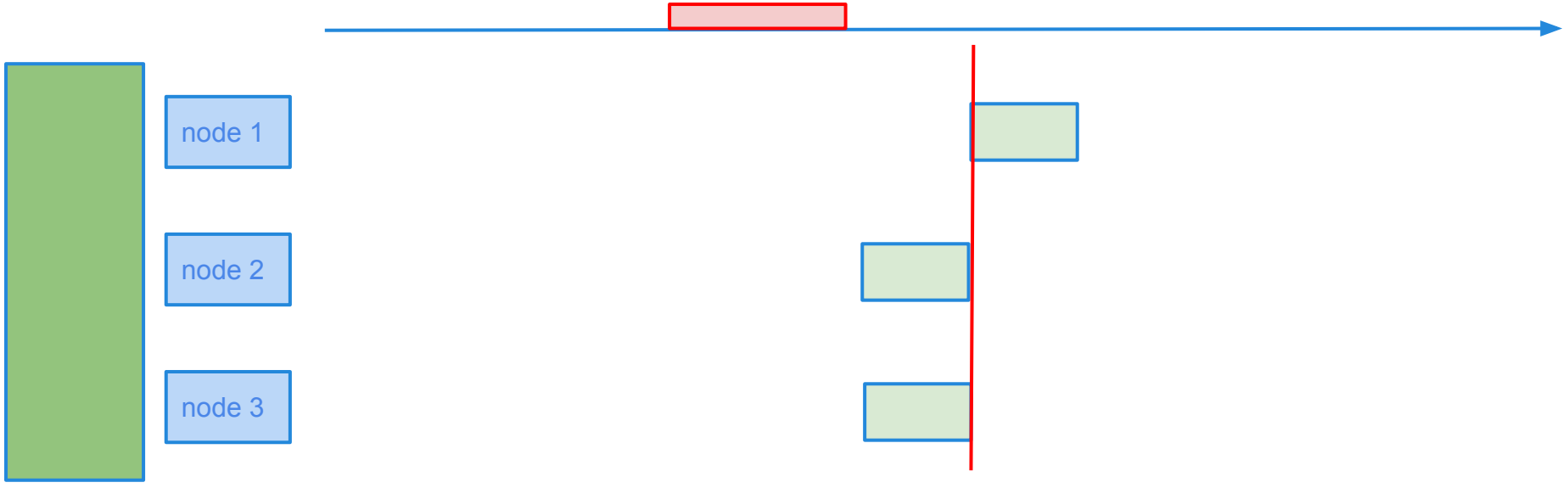
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



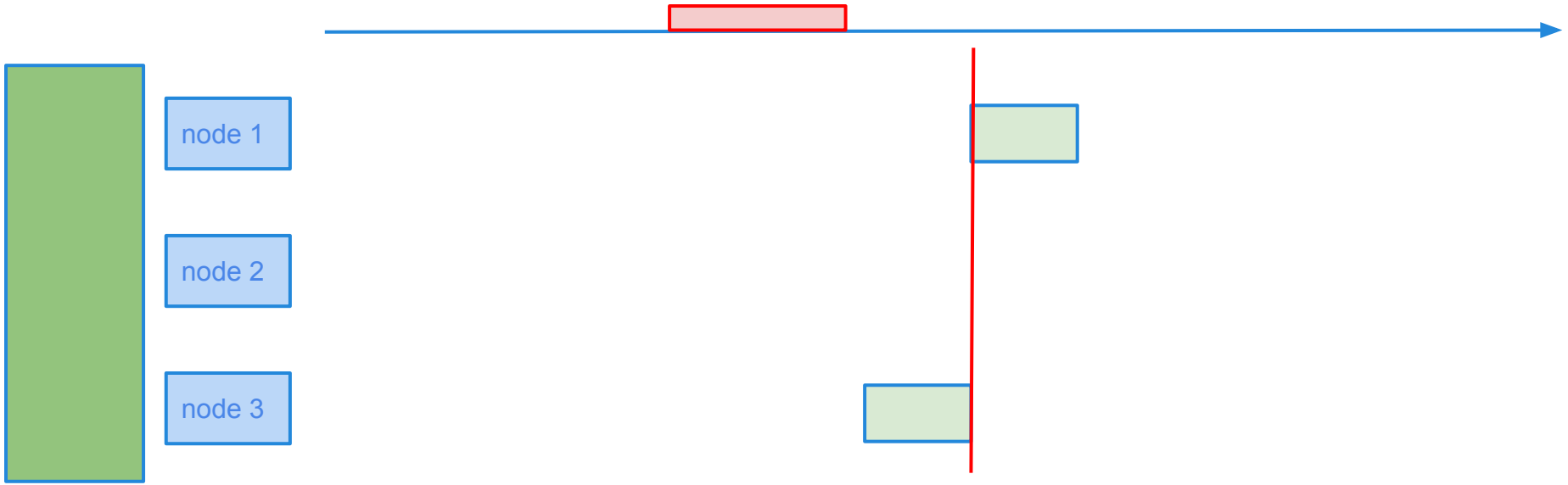
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



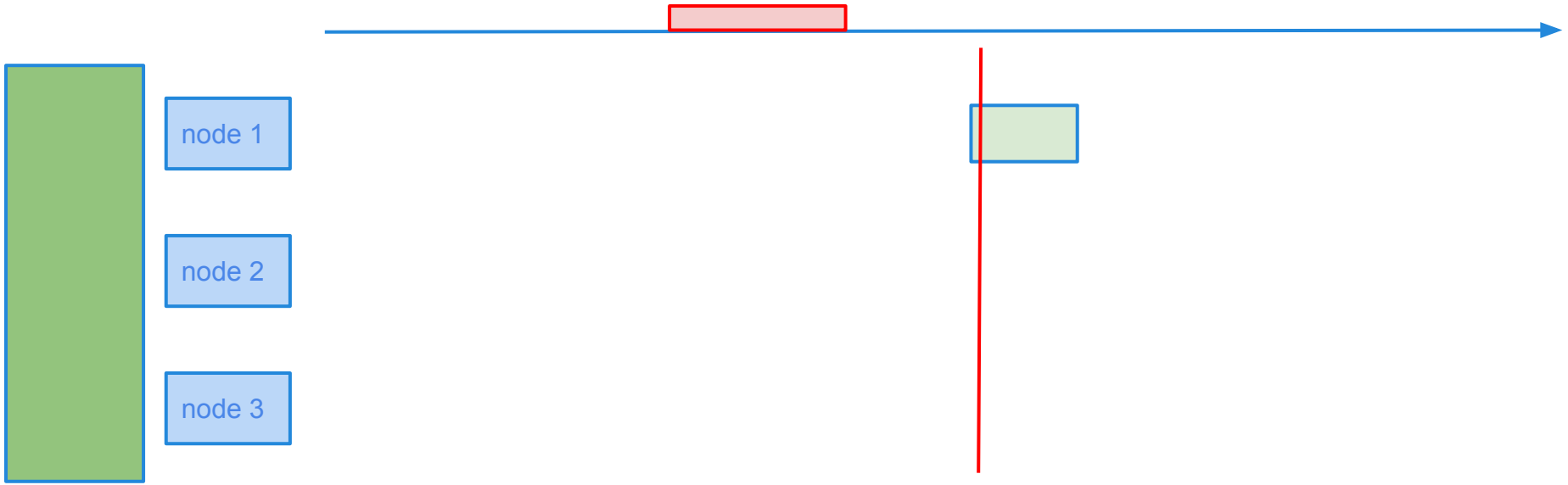
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



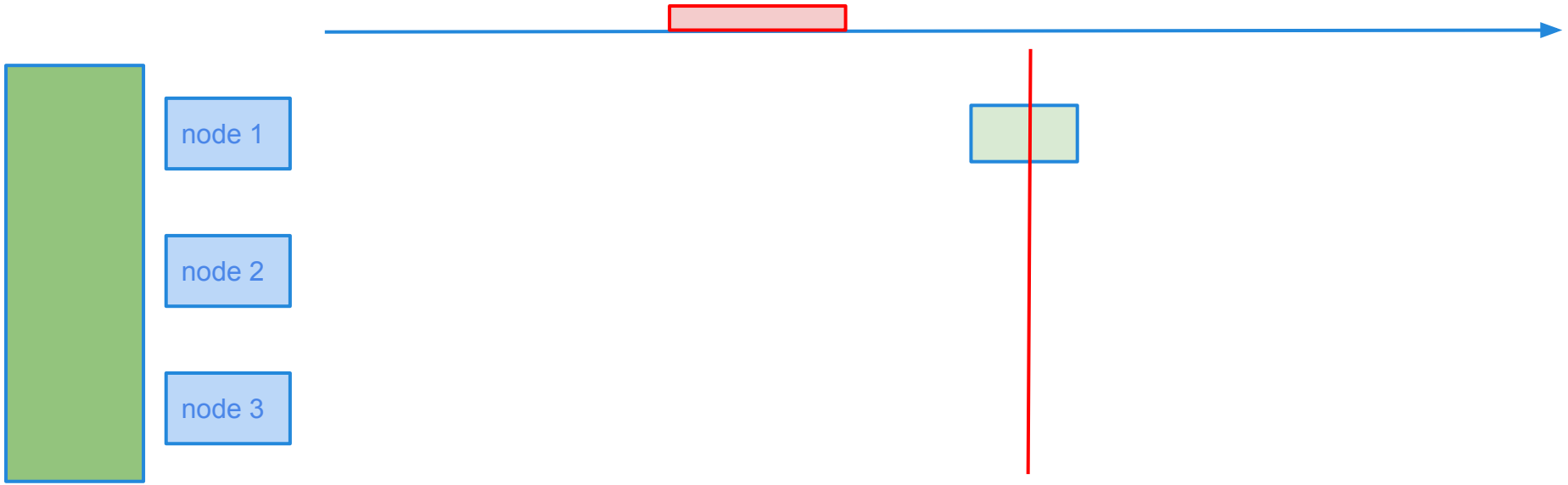
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



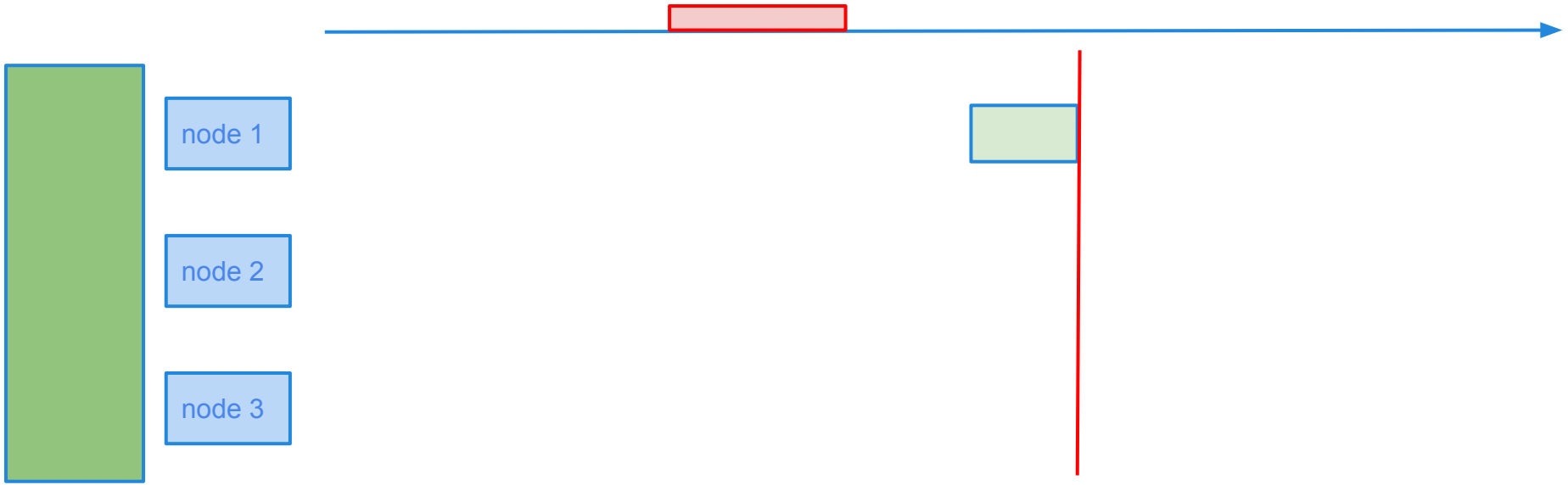
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



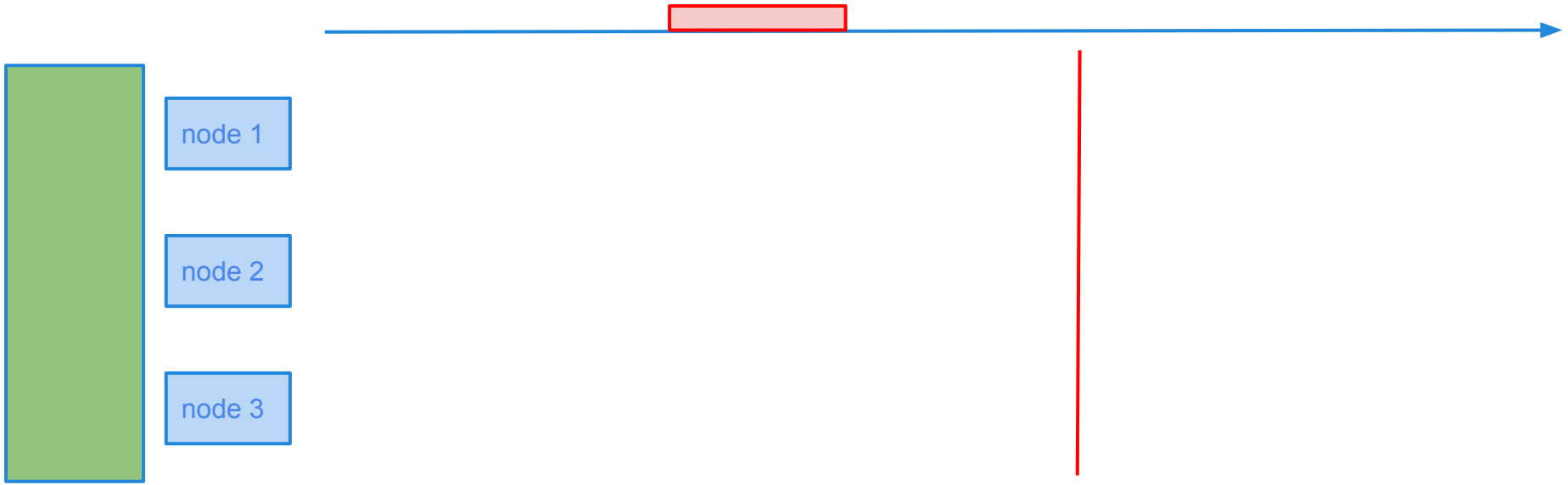
Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



Stage 2

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```



Let's refactor

```
val events = sc.textFile(“hdfs://journal/*”)
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```

Let's refactor

```
val events = sc.textFile(“hdfs://journal/*”)
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```

Let's refactor

```
val events = sc.textFile("hdfs://journal/*")
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
    .filter {   case (date, size) => LocalDate.parse(date) isAfter LocalDate.of(2015,3,1) }
```

Let's refactor

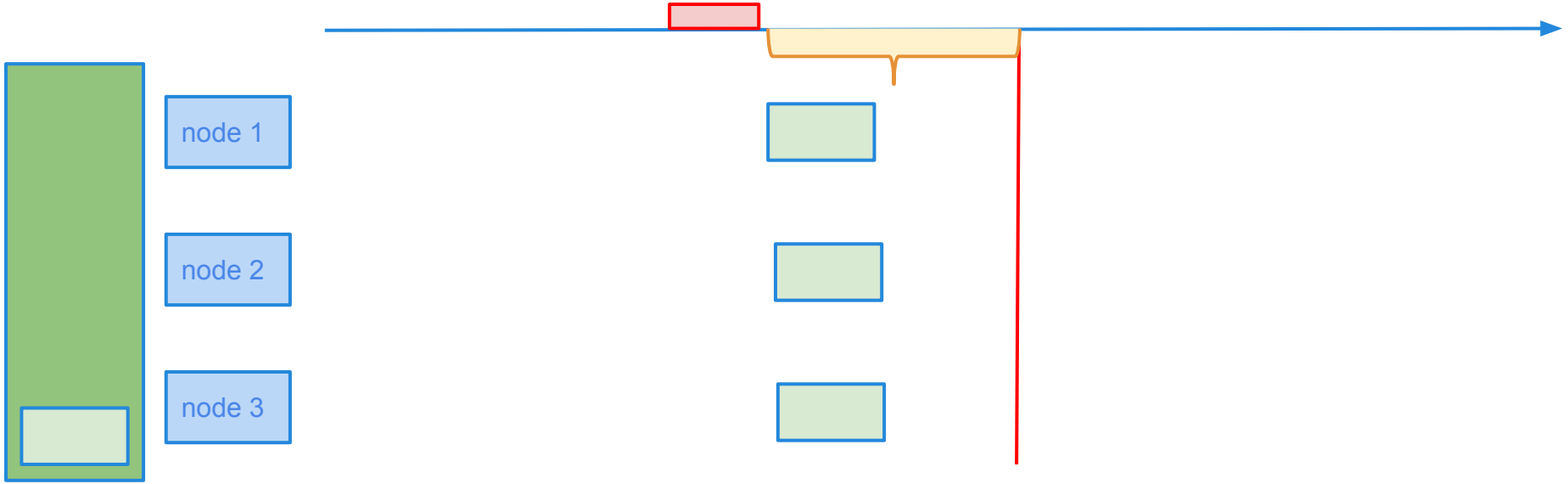
```
val events = sc.textFile(“hdfs://journal/*”)
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
```

Let's refactor

```
val events = sc.textFile(“hdfs://journal/*”)
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
```

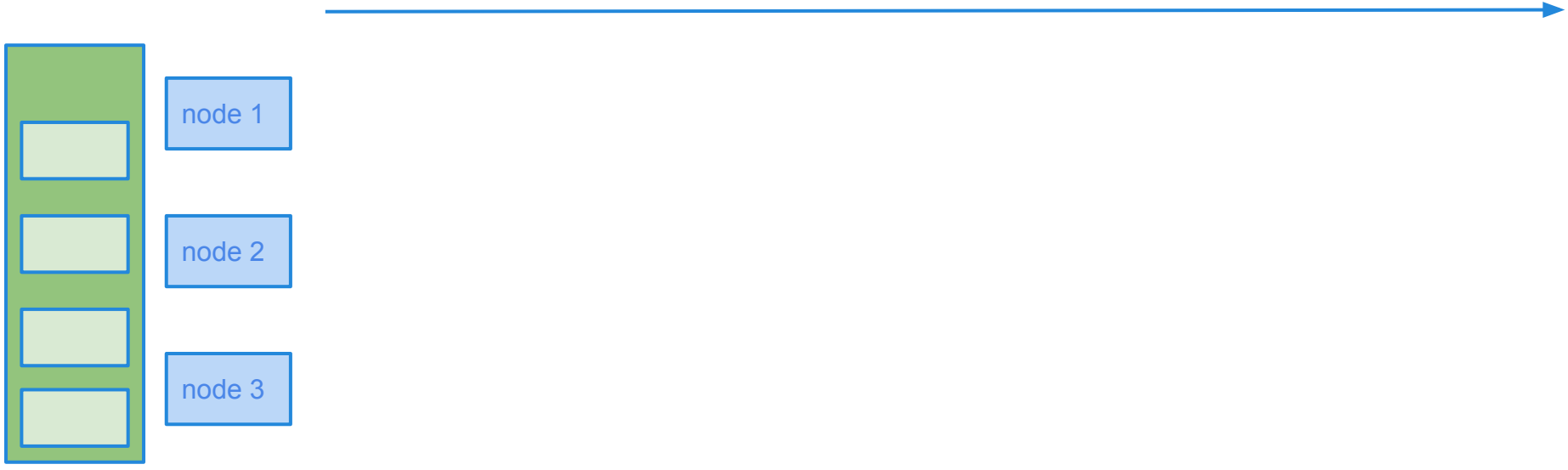
Let's refactor

```
val events = sc.textFile("hdfs://journal/*")
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
```



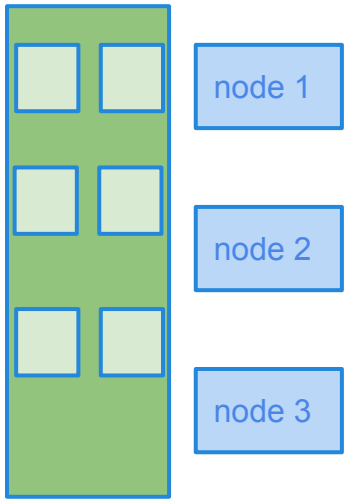
Let's refactor

```
val events = sc.textFile("hdfs://journal/*")
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .groupBy(extractDate _)
    .map { case (date, events) => (date, events.size) }
```



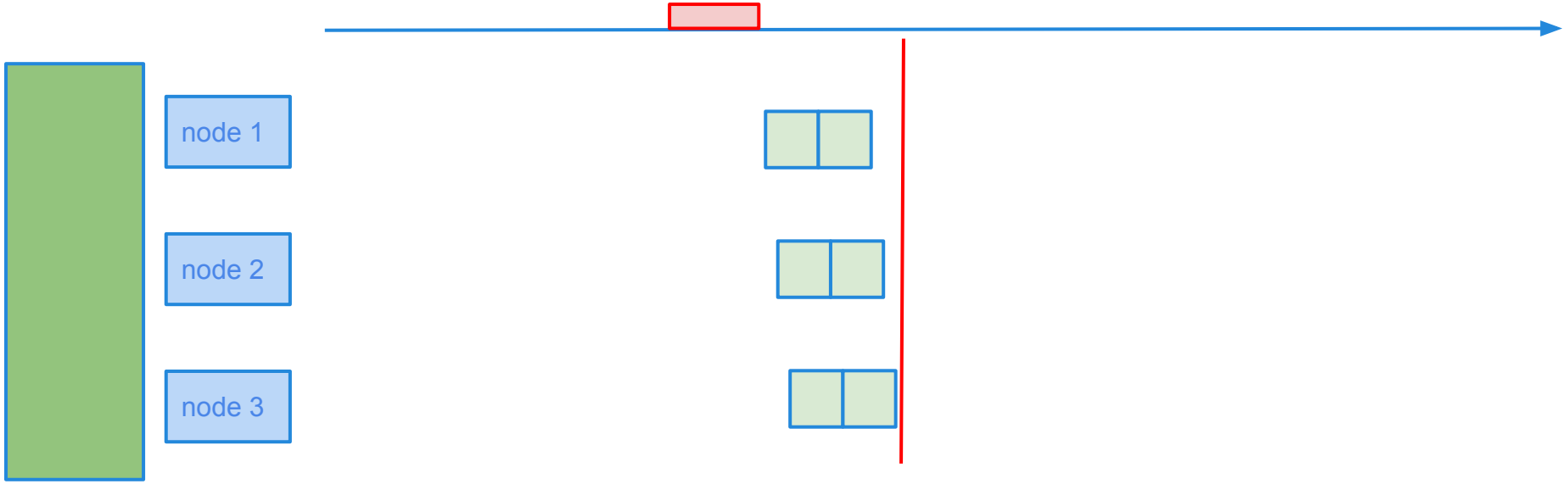
Let's refactor

```
val events = sc.textFile("hdfs://journal/*")
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .groupBy(extractDate _,  6)
    .map { case (date, events) => (date, events.size) }
```



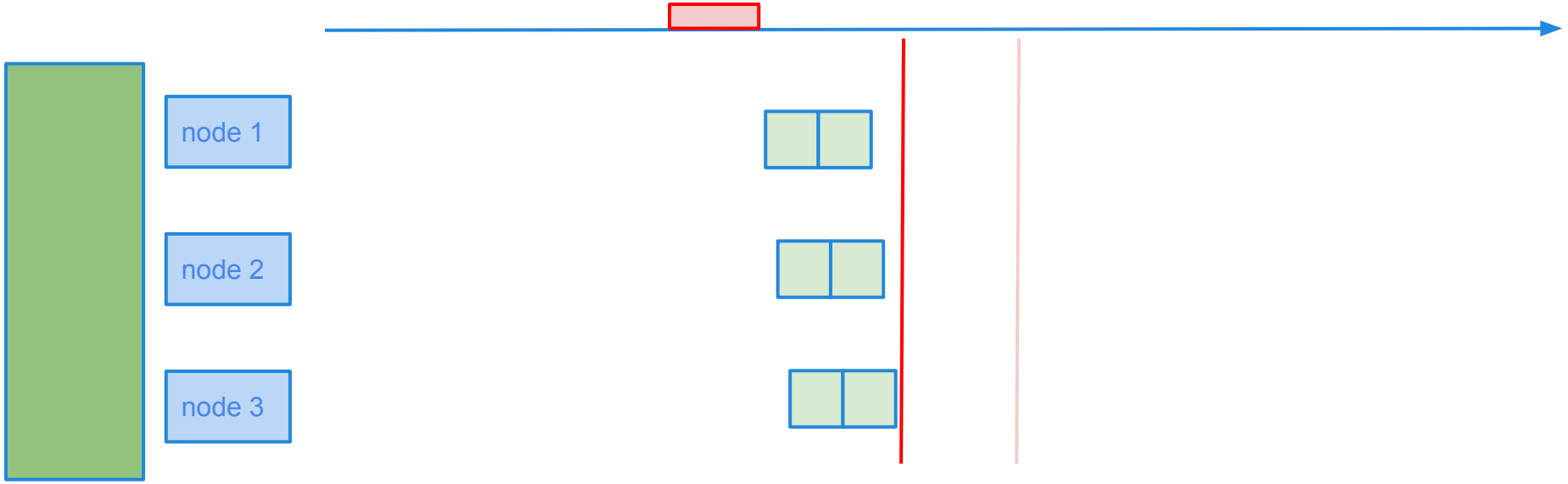
Let's refactor

```
val events = sc.textFile("hdfs://journal/*")
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .groupBy(extractDate _,  6)
    .map { case (date, events) => (date, events.size) }
```



Let's refactor

```
val events = sc.textFile("hdfs://journal/*")
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .groupBy(extractDate _,  6)
    .map { case (date, events) => (date, events.size) }
```



Let's refactor

```
val events = sc.textFile(“hdfs://journal/*”)
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .groupBy(extractDate _,  6)
    .map { case (date, events) => (date, events.size) }
```

Let's refactor

```
val events = sc.textFile(“hdfs://journal/*”)
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .map( e => (extractDate(e), e))
    .combineByKey( e => 1, _ + 1, _ + _)
    .groupBy(extractDate _,  6)
    .map { case (date, events) => (date, events.size) }
```

Let's refactor

```
val events = sc.textFile(“hdfs://journal/*”)
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .map( e => (extractDate(e), e))
    .combineByKey( e => 1, _ + 1, _ + _)
```

A bit more about partitions

```
val events = sc.textFile(“hdfs://journal/*”)  // here small number of partitions, let’s say 4
    .map( e => (extractDate(e), e))
```

A bit more about partitions

```
val events = sc.textFile("hdfs://journal/*")   // here small number of partitions, let's say 4
      .repartition(256)                        // note, this will cause a shuffle
      .map( e => (extractDate(e), e))
```

A bit more about partitions

```
val events = sc.textFile(“hdfs://journal/*”) // here a lot of partitions, let’s say 1024
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .map( e => (extractDate(e), e))
```

A bit more about partitions

```
val events = sc.textFile("hdfs://journal/*") // here a lot of partitions, let's say 1024
    .filter {  LocalDate.parse(extractDate _) isAfter LocalDate.of(2015,3,1) }
    .coalesce(64)    // this will NOT shuffle
    .map( e => (extractDate(e), e))
```

Few optimization tricks

Few optimization tricks

1. Serialization issues (e.g KryoSerializer)

Few optimization tricks

1. Serialization issues (e.g KryoSerializer)
2. Turn on speculations if on shared cluster

Few optimization tricks

1. Serialization issues (e.g KryoSerializer)
2. Turn on speculations if on shared cluster
3. Experiment with compression codecs

Few optimization tricks

1. Serialization issues (e.g KryoSerializer)
2. Turn on speculations if on shared cluster
3. Experiment with compression codecs
4. Monitor GC aka “immutability is not your friend in distributed environment”

Few optimization tricks

1. Serialization issues (e.g KryoSerializer)
2. Turn on speculations if on shared cluster
3. Experiment with compression codecs
4. Monitor GC aka “immutability is not your friend in distributed environment”
5. Use profiler to monitor behaviour of running executors

Few optimization tricks

1. Serialization issues (e.g KryoSerializer)
2. Turn on speculations if on shared cluster
3. Experiment with compression codecs
4. Monitor GC aka “immutability is not your friend in distributed environment”
5. Use profiler to monitor behaviour of running executors
6. Consider Spark History Server (spark.eventlog.enabled)

Few optimization tricks

1. Serialization issues (e.g KryoSerializer)
2. Turn on speculations if on shared cluster
3. Experiment with compression codecs
4. Monitor GC aka “immutability is not your friend in distributed environment”
5. Use profiler to monitor behaviour of running executors
6. Consider Spark History Server (spark.eventlog.enabled)
7. DataFrames

When things go really wrong

When things go really wrong

1. Monitor (SparkUI, Profiler)

When things go really wrong

1. Monitor (SparkUI, Profiler)
2. Learn API (e.g groupBy can destroy your computation)

When things go really wrong

1. Monitor (SparkUI, Profiler)
2. Learn API (e.g groupBy can destroy your computation)
3. Understand basics & internals to omit common mistakes

When things go really wrong

1. Monitor (SparkUI, Profiler)
2. Learn API (e.g groupBy can destroy your computation)
3. Understand basics & internals to omit common mistakes

Example: Usage of RDD within other RDD. Invalid since only the driver can call operations on RDDS (not other workers)

When things go really wrong

1. Monitor (SparkUI, Profiler)
2. Learn API (e.g groupBy can destroy your computation)
3. Understand basics & internals to omit common mistakes

Example: Usage of RDD within other RDD. Invalid since only the driver can call operations on RDDS (not other workers)

rdd.map((k,v) => otherRDD.get(k)) -> rdd.join(otherRdd)

rdd.map(e => otherRdd.map {}) -> rdd.cartesian(otherRdd)

Resources & further read

- Spark Summit Talks @ Youtube

Resources & further read

- Spark Summit Talks @ Youtube
- “Learn Spark” book, published by O’Reilly

Resources & further read

- Spark Summit Talks @ Youtube
- “Learn Spark” book, published by O’Reilly
- Apache Spark Documentation
  - https://spark.apache.org/docs/latest/monitoring.html
  - https://spark.apache.org/docs/latest/tuning.html

Resources & further read

- Spark Summit Talks @ Youtube
- “Learn Spark” book, published by O’Reilly
- Apache Spark Documentation
  - https://spark.apache.org/docs/latest/monitoring.html
  - https://spark.apache.org/docs/latest/tuning.html
- Mailing List aka solution-to-your-problem-is-probably-already-there

Resources & further read

- Spark Summit Talks @ Youtube
- “Learn Spark” book, published by O’Reilly
- Apache Spark Documentation
  - https://spark.apache.org/docs/latest/monitoring.html
  - https://spark.apache.org/docs/latest/tuning.html
- Mailing List aka solution-to-your-problem-is-probably-already-there
- Pretty soon my blog & github :)

Paweł Szulc

Paweł Szulc

blog: http://rabbitonweb.com

Paweł Szulc

blog: http://rabbitonweb.com
twitter: @rabbitonweb

Paweł Szulc

blog: http://rabbitonweb.com
twitter: @rabbitonweb
github: https://github.com/rabbitonweb