



A Deeper Understanding of Spark's Internals

Aaron Davidson
07/01/2014

This Talk

- Goal: Understanding how Spark runs, focus on performance
- Major core components:
 - Execution Model
 - The Shuffle
 - Caching

This Talk

- Goal: Understanding how Spark runs, focus on performance
- Major core components:
 - Execution Model
 - The Shuffle
 - ~~– Caching~~

Why understand internals?

Goal: Find number of distinct names per “first letter”

```
sc.textFile("hdfs:/names")  
  
  .map(name => (name.charAt(0), name))  
  
  .groupByKey()  
  
  .mapValues(names => names.toSet.size)  
  
  .collect()
```

Why understand internals?

Goal: Find number of distinct names per “first letter”

```
sc.textFile("hdfs:/names")
```

Ahir

Pat

Andy

```
.map(name => (name.charAt(0), name))
```

```
.groupByKey()
```

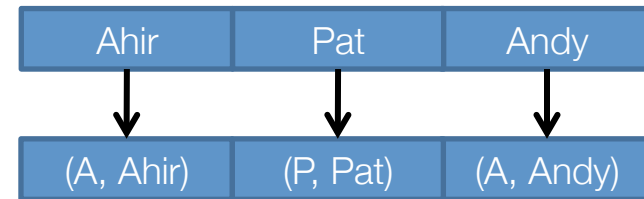
```
.mapValues(names => names.toSet.size)
```

```
.collect()
```

Why understand internals?

Goal: Find number of distinct names per “first letter”

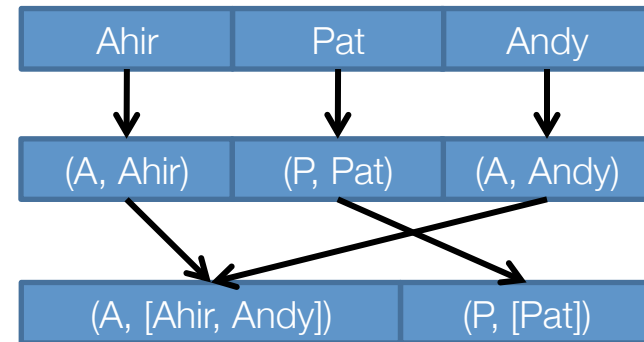
```
sc.textFile("hdfs:/names")  
  
  .map(name => (name.charAt(0), name))  
  
  .groupByKey()  
  
  .mapValues(names => names.toSet.size)  
  
  .collect()
```



Why understand internals?

Goal: Find number of distinct names per “first letter”

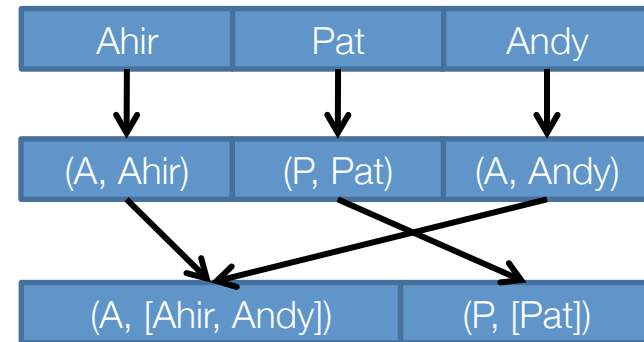
```
sc.textFile("hdfs:/names")  
  
  .map(name => (name.charAt(0), name))  
  
  .groupByKey()  
  
  .mapValues(names => names.toSet.size)  
  
  .collect()
```



Why understand internals?

Goal: Find number of distinct names per “first letter”

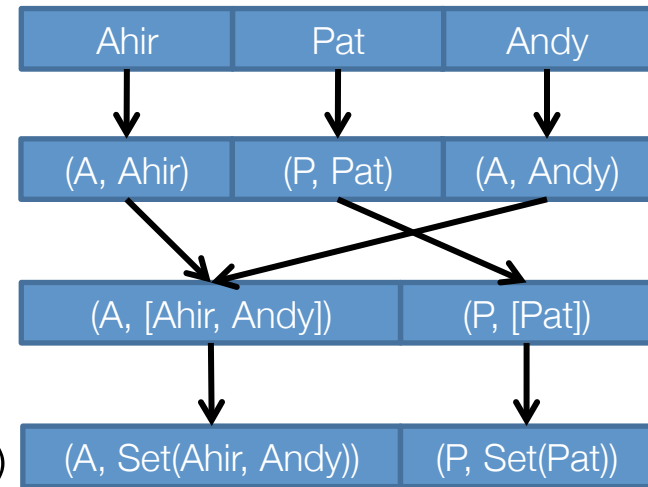
```
sc.textFile("hdfs:/names")  
  
  .map(name => (name.charAt(0), name))  
  
  .groupByKey()  
  
  .mapValues(names => names.toSet.size)  
  
  .collect()
```



Why understand internals?

Goal: Find number of distinct names per “first letter”

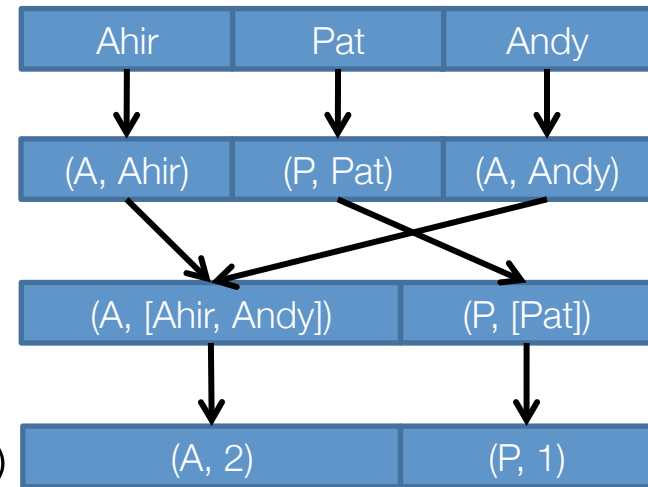
```
sc.textFile("hdfs:/names")  
  
  .map(name => (name.charAt(0), name))  
  
  .groupByKey()  
  
  .mapValues(names => names.toSet.size)  
  
  .collect()
```



Why understand internals?

Goal: Find number of distinct names per “first letter”

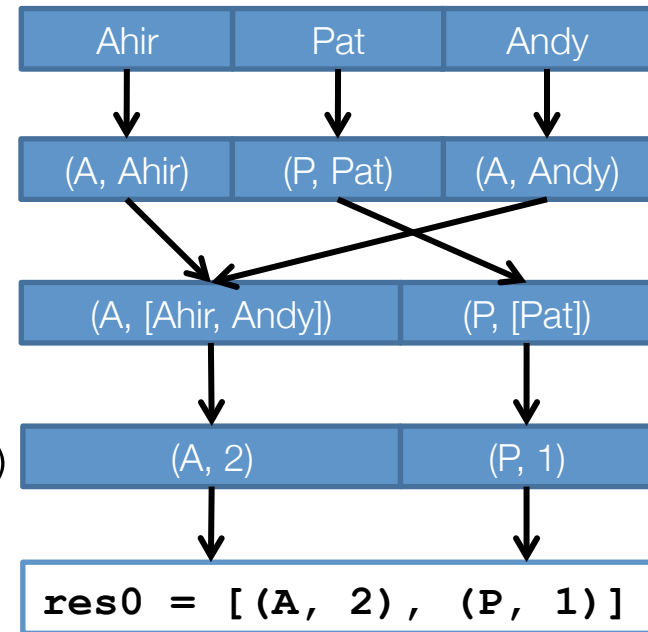
```
sc.textFile("hdfs:/names")  
  
  .map(name => (name.charAt(0), name))  
  
  .groupByKey()  
  
  .mapValues(names => names.toSet.size)  
  
  .collect()
```



Why understand internals?

Goal: Find number of distinct names per “first letter”

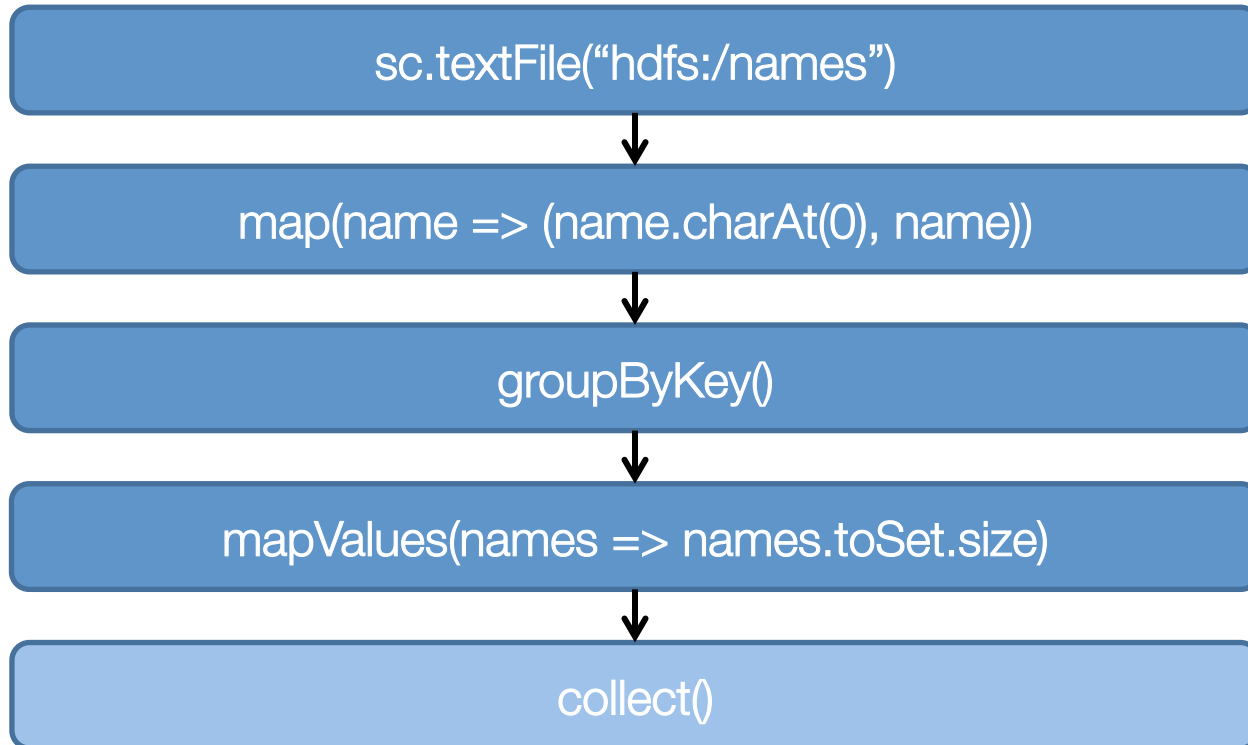
```
sc.textFile("hdfs:/names")  
  
  .map(name => (name.charAt(0), name))  
  
  .groupByKey()  
  
  .mapValues(names => names.toSet.size)  
  
  .collect()
```



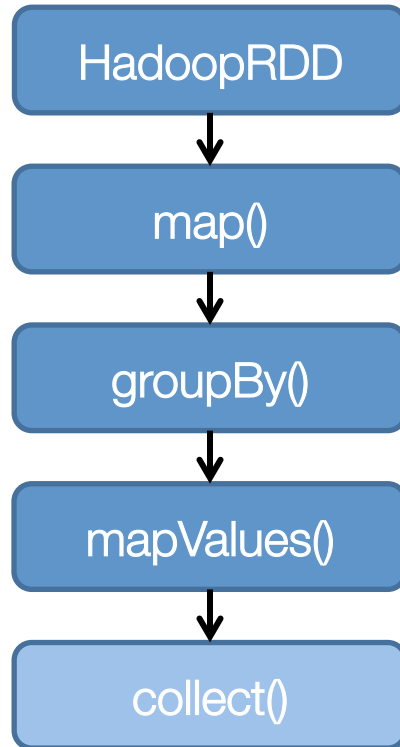
Spark Execution Model

1. Create DAG of RDDs to represent computation
2. Create logical execution plan for DAG
3. Schedule and execute individual tasks

Step 1: Create RDDs



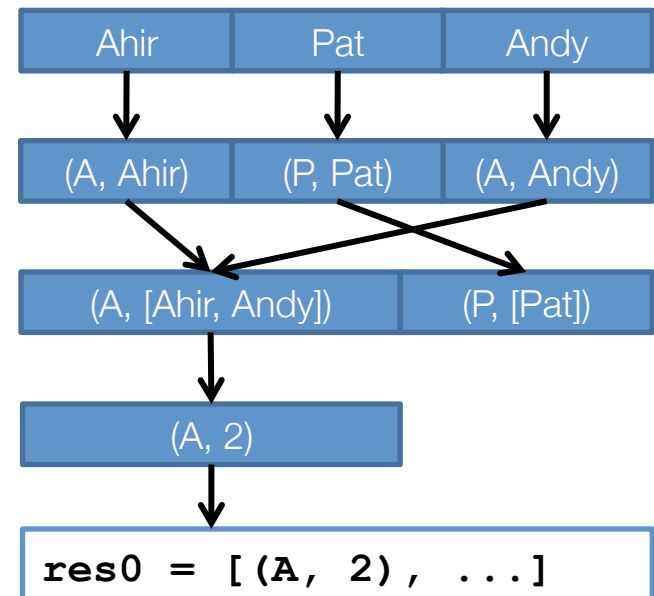
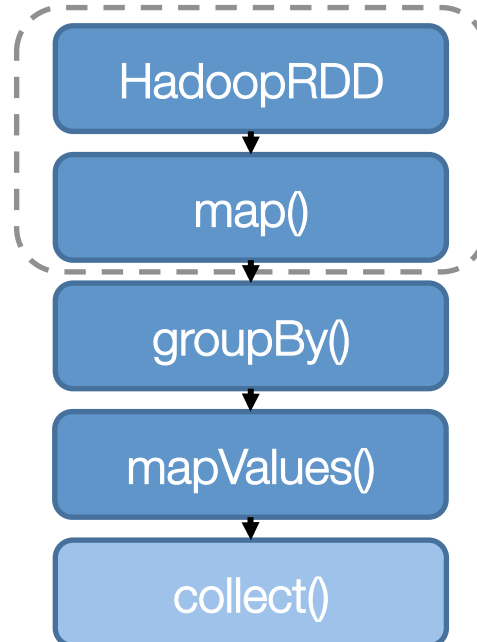
Step 1: Create RDDs



Step 2: Create execution plan

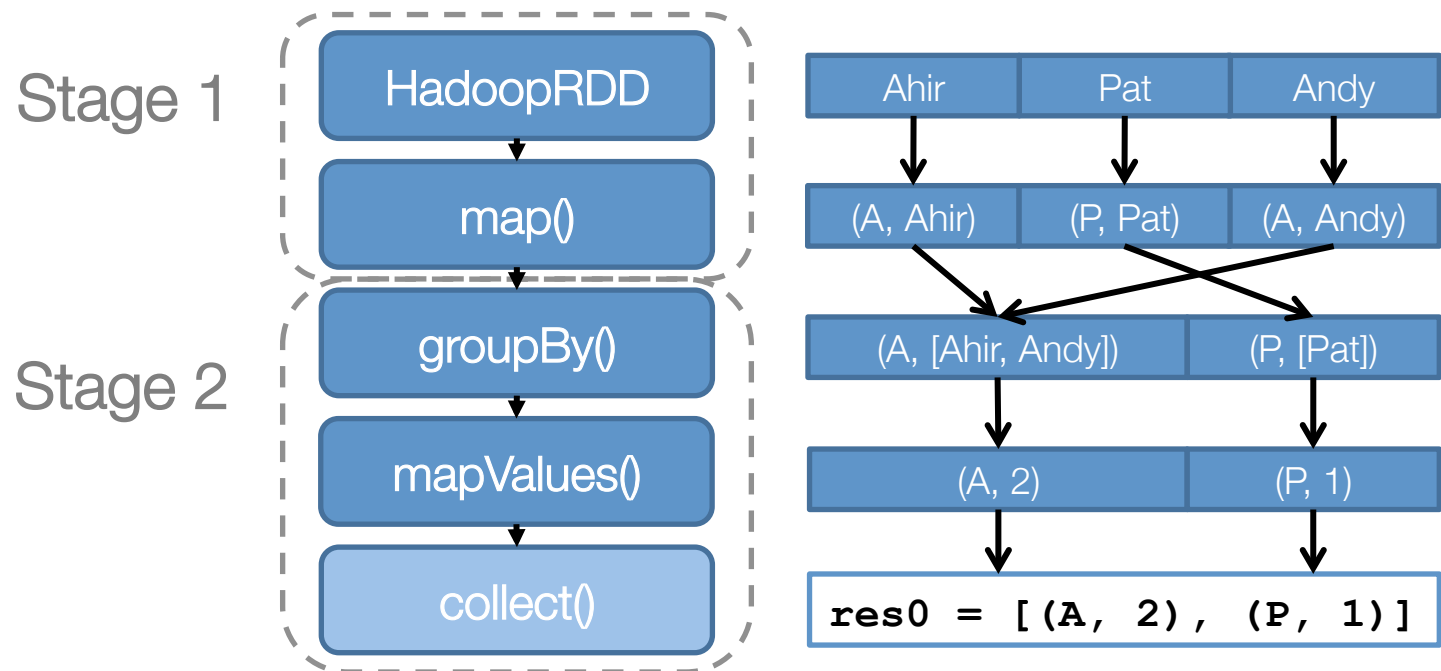
- Pipeline as much as possible
- Split into “**stages**” based on need to reorganize data

Stage 1



Step 2: Create execution plan

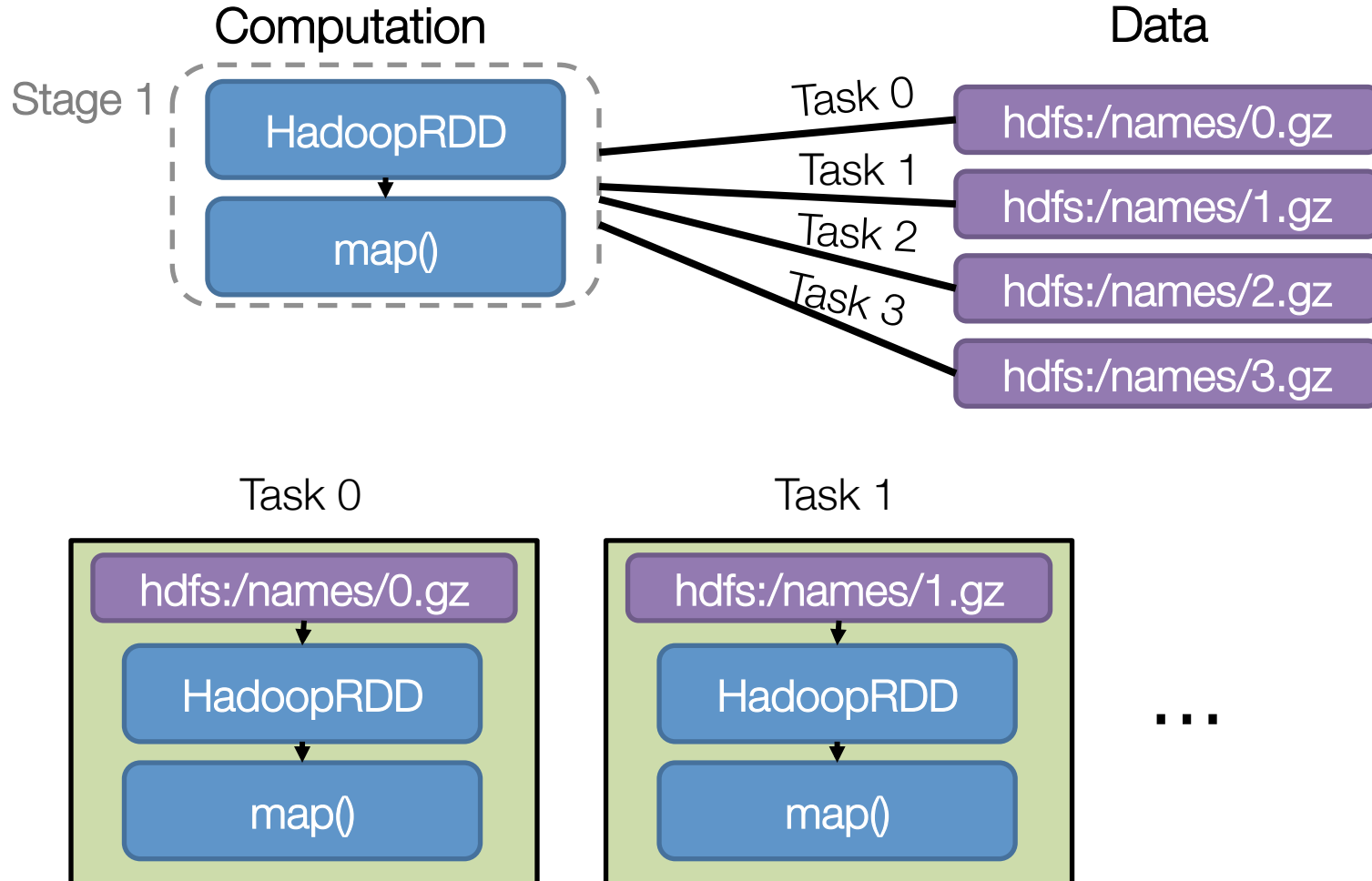
- Pipeline as much as possible
- Split into “**stages**” based on need to reorganize data



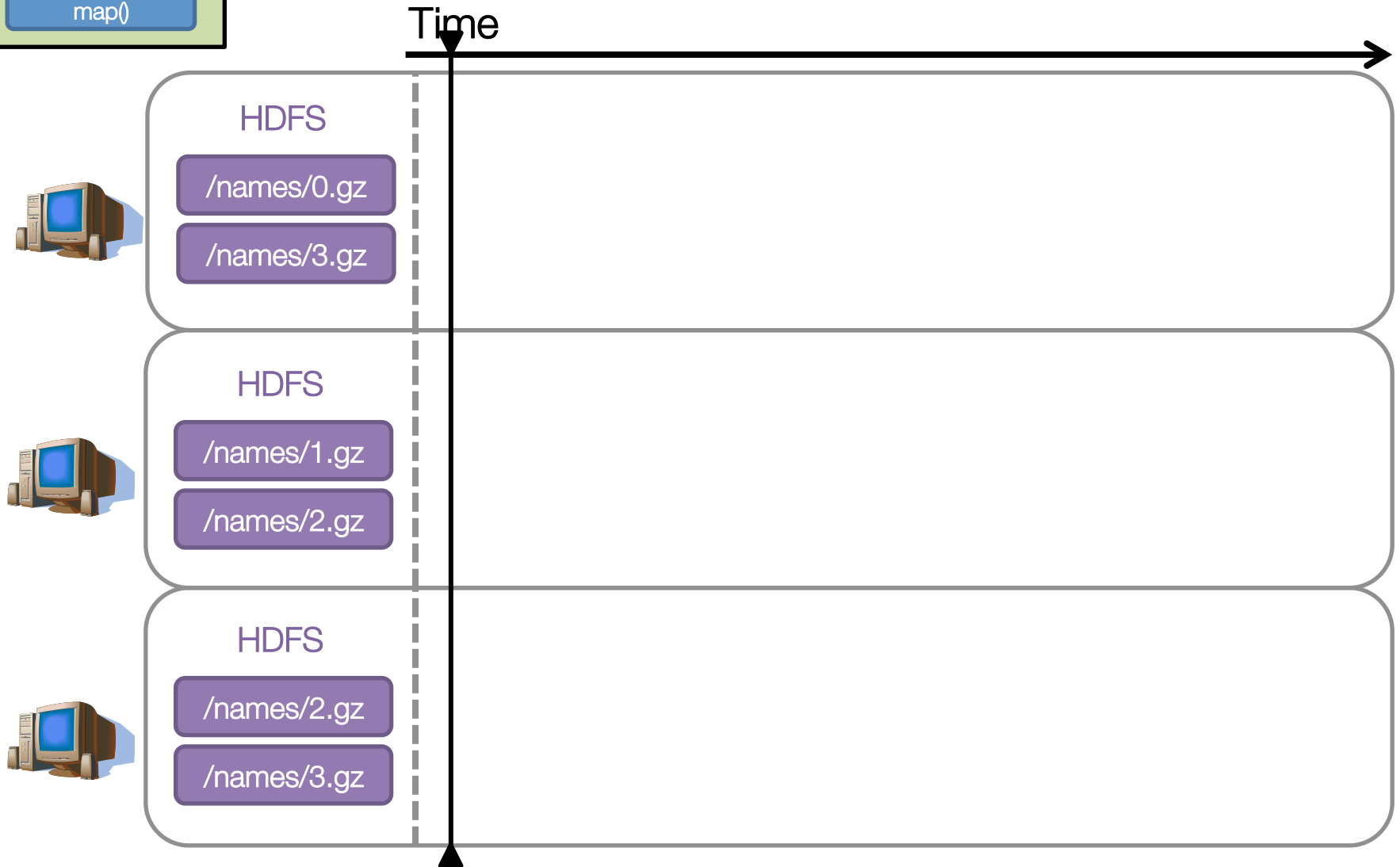
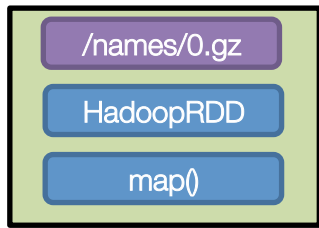
Step 3: Schedule tasks

- Split each stage into **tasks**
- A task is data + computation
- Execute all tasks within a stage before moving on

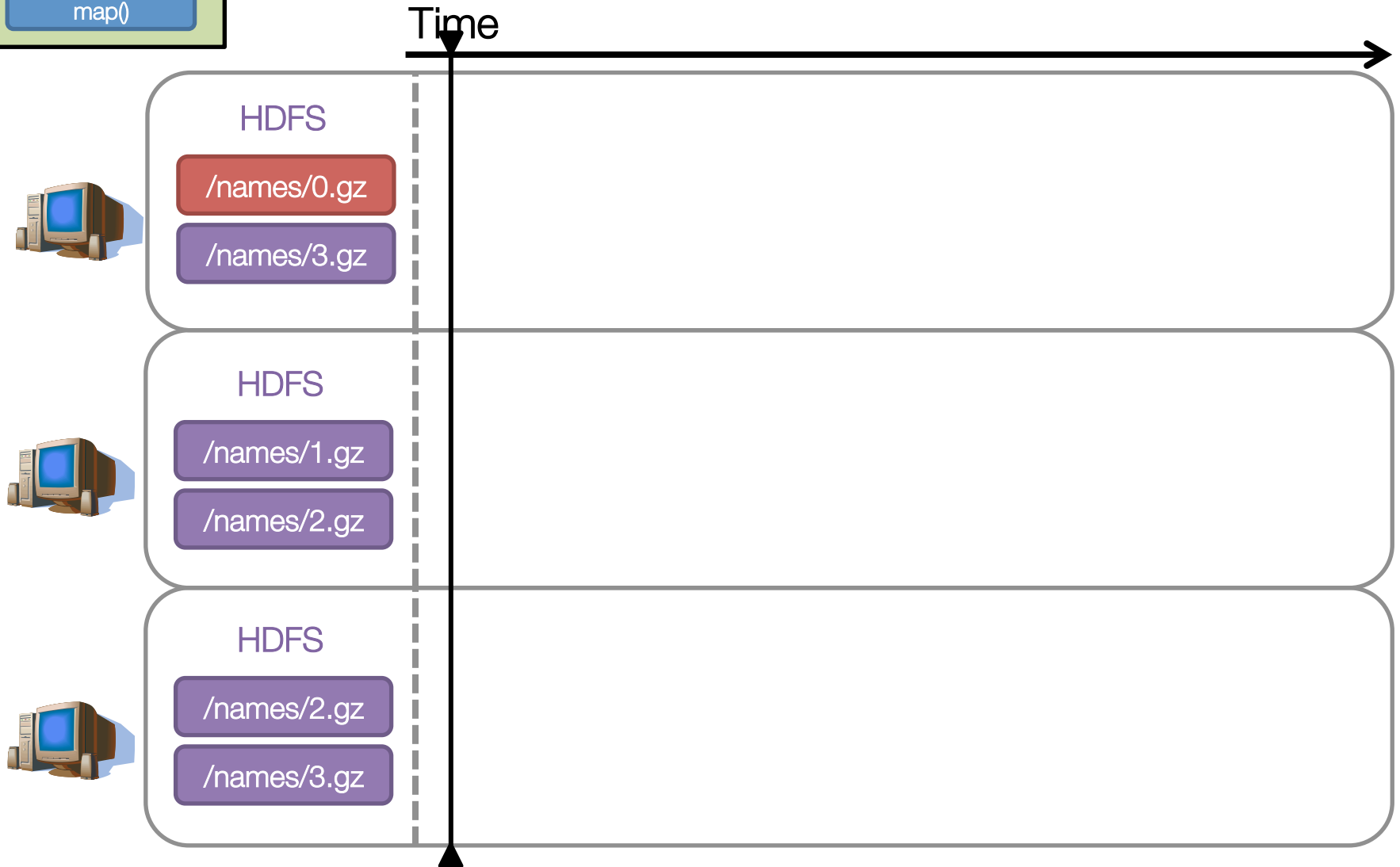
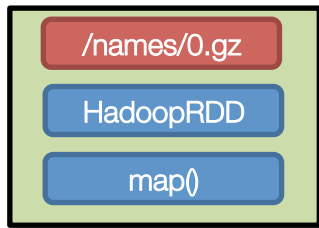
Step 3: Schedule tasks



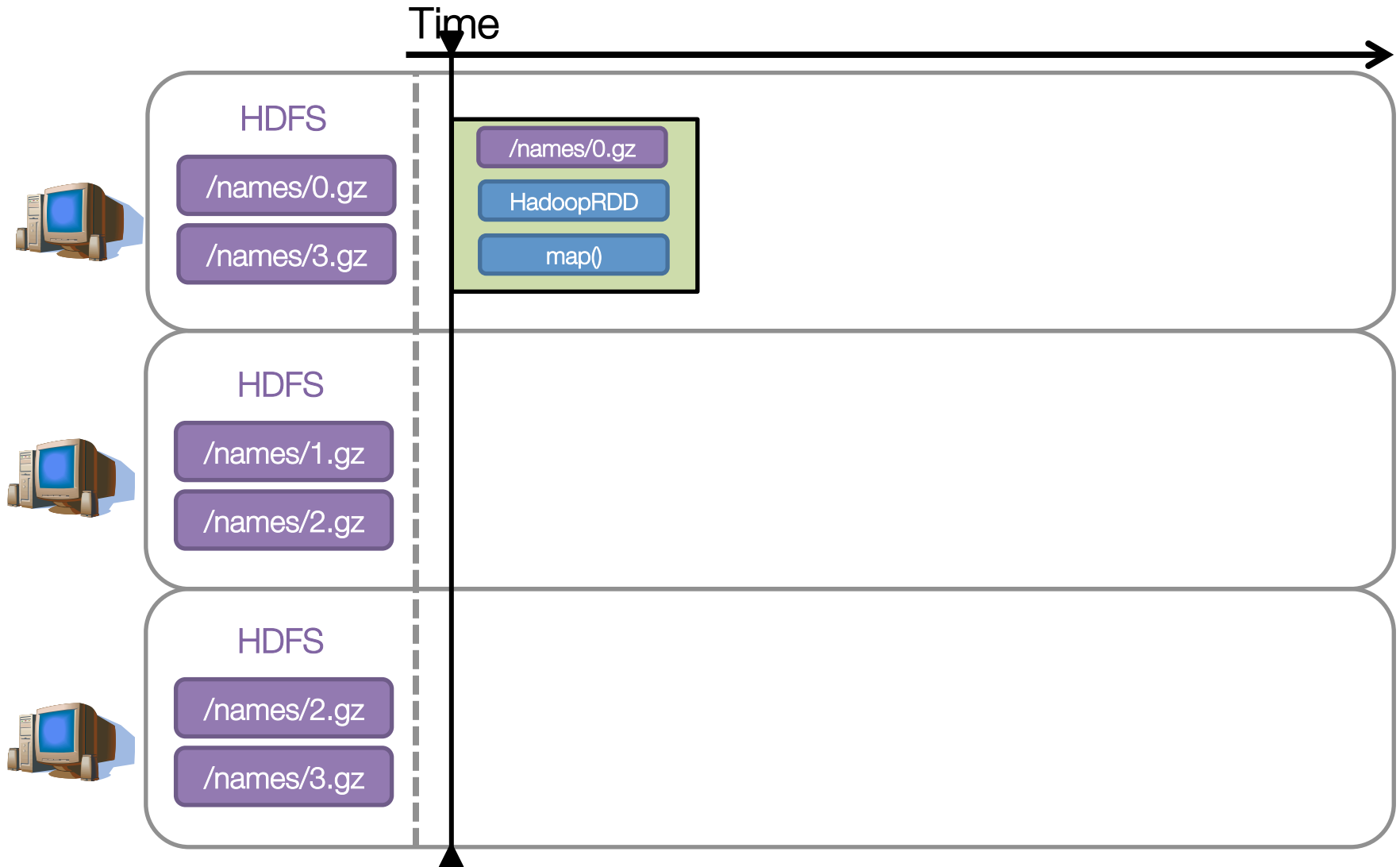
Step 3: Schedule tasks



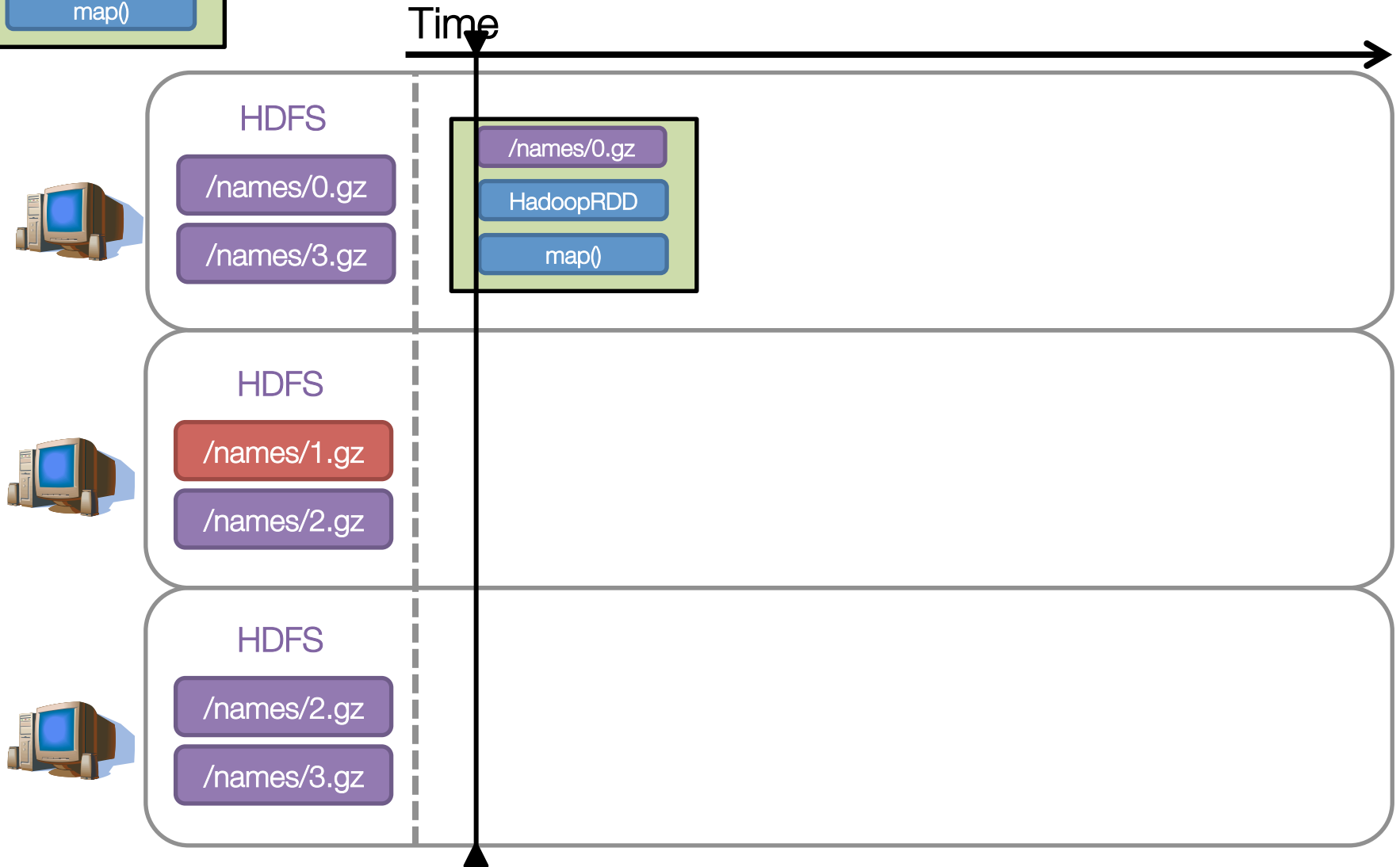
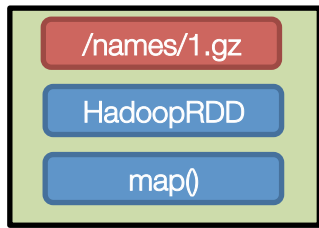
Step 3: Schedule tasks



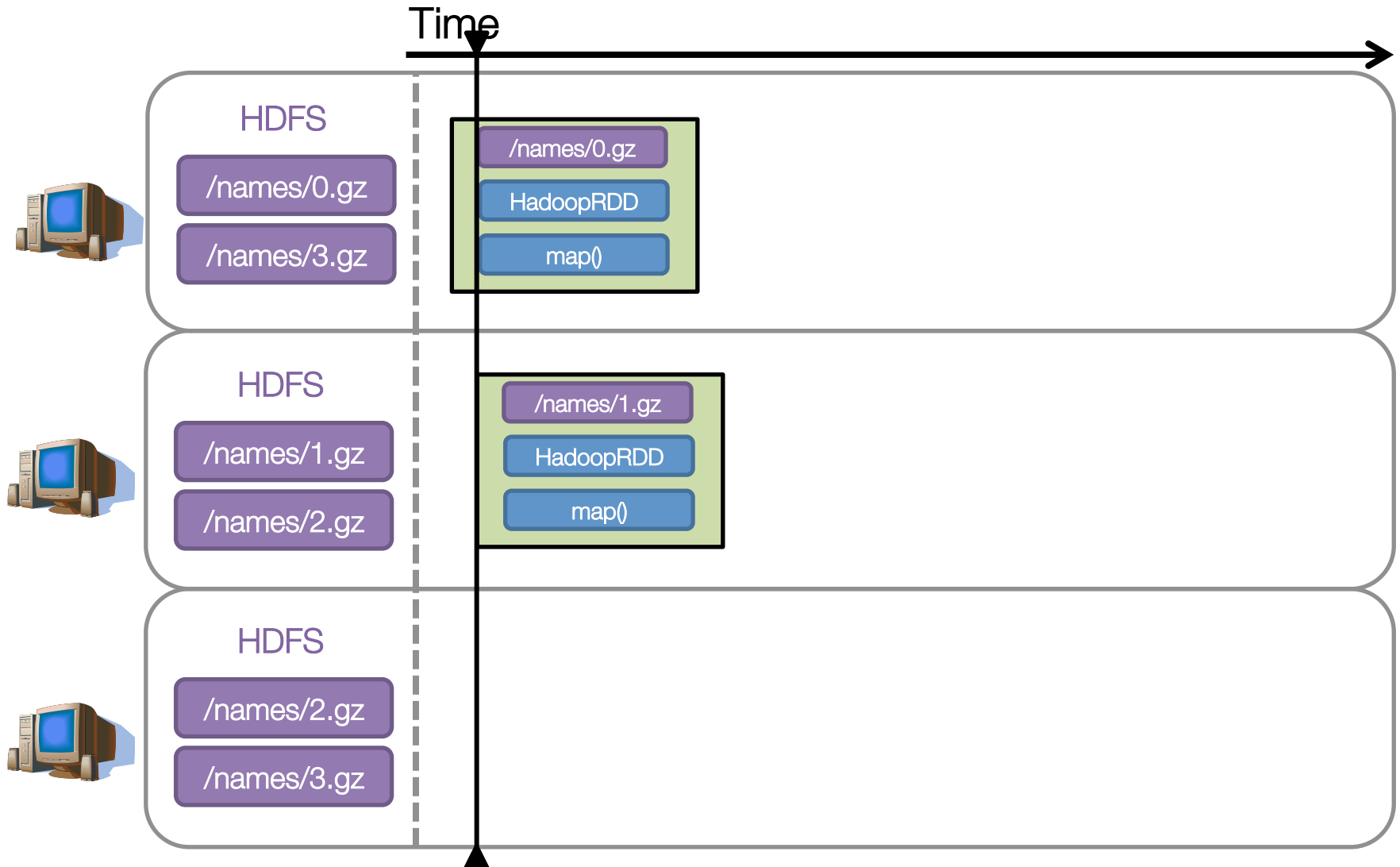
Step 3: Schedule tasks



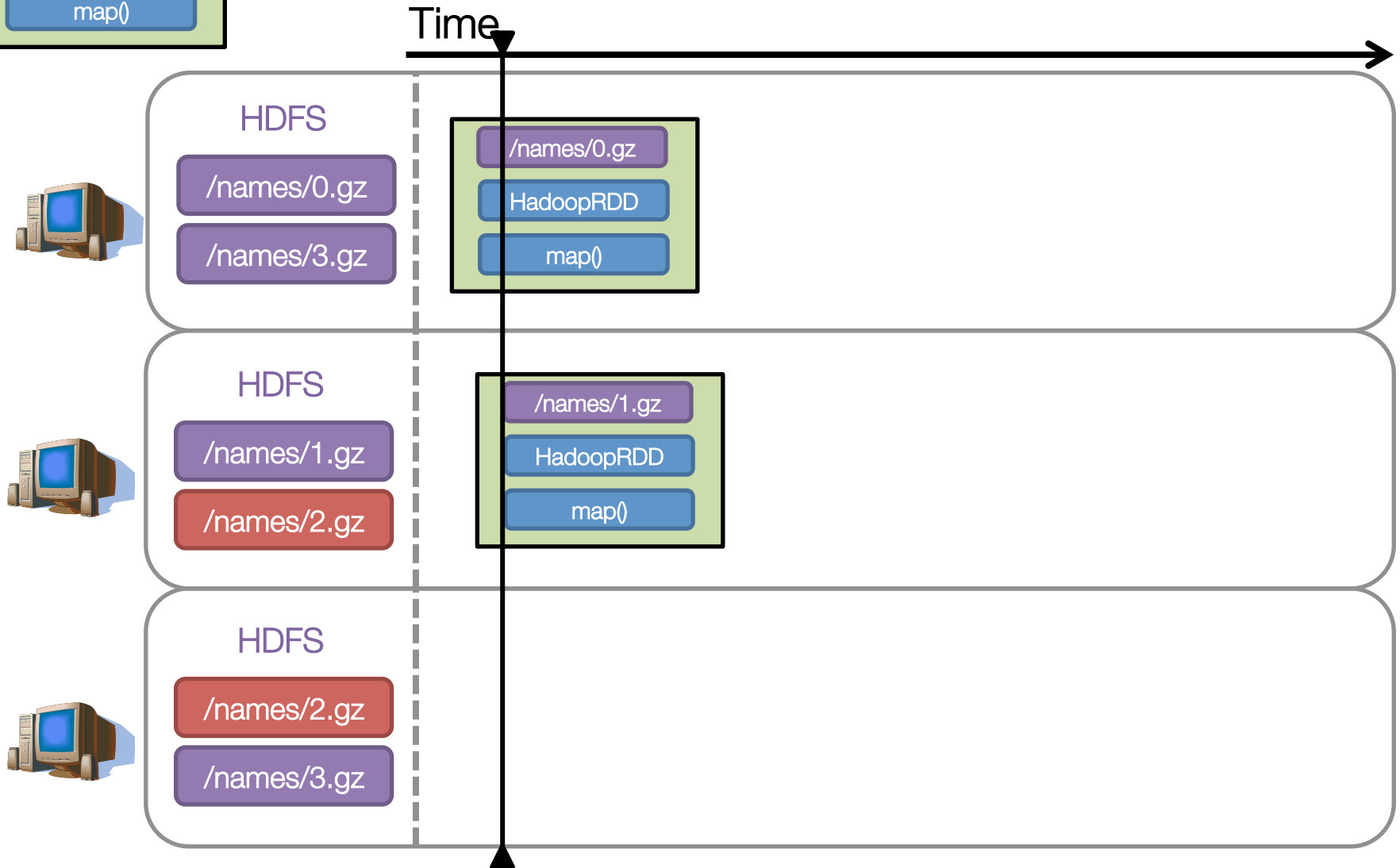
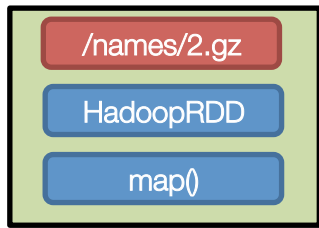
Step 3: Schedule tasks



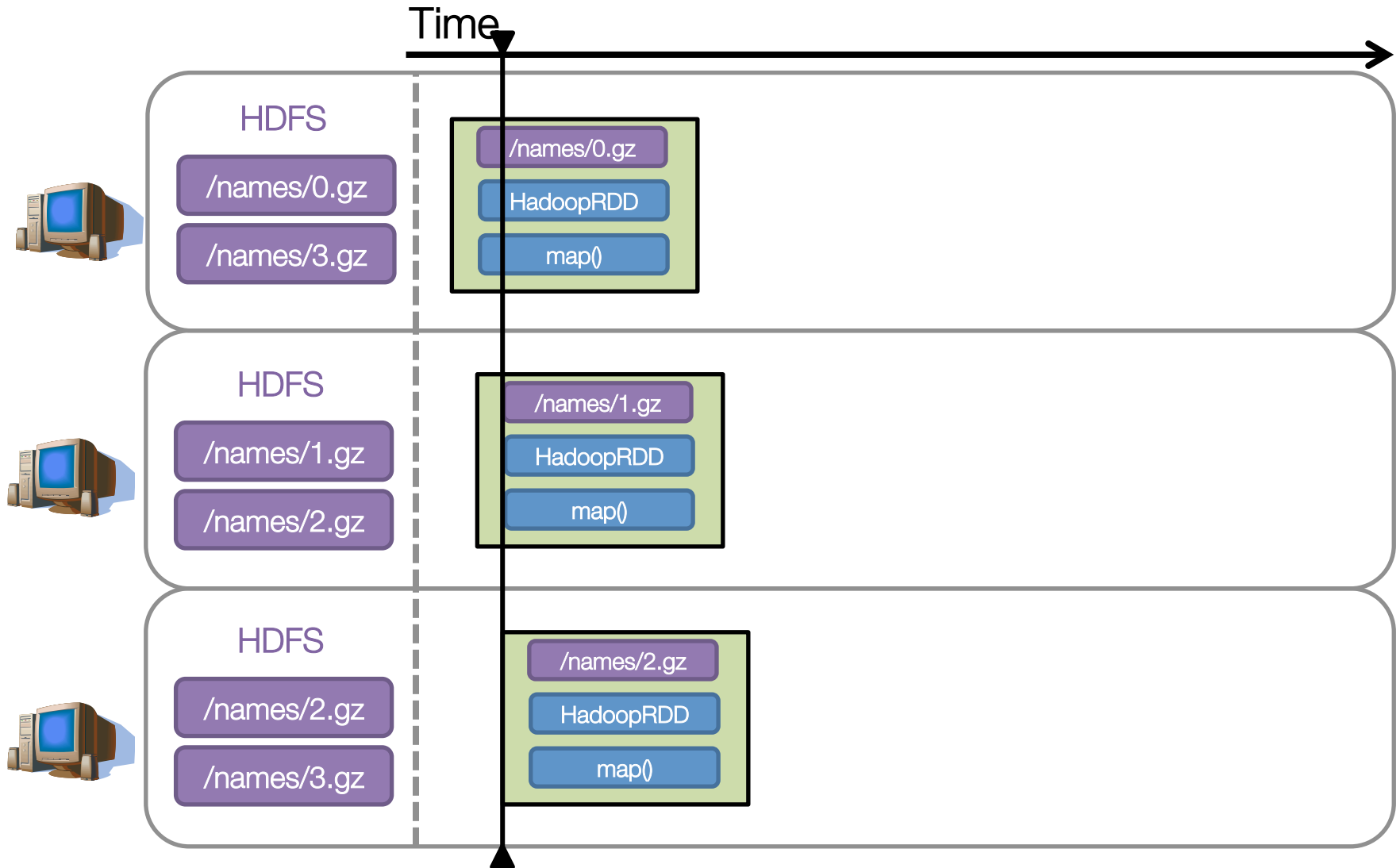
Step 3: Schedule tasks



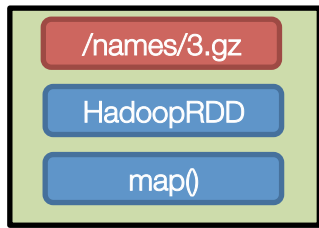
Step 3: Schedule tasks



Step 3: Schedule tasks



Step 3: Schedule tasks



Time



HDFS

`/names/0.gz`

`/names/3.gz`

`/names/0.gz`

`HadoopRDD`

`map()`



HDFS

`/names/1.gz`

`/names/2.gz`

`/names/1.gz`

`HadoopRDD`

`map()`



HDFS

`/names/2.gz`

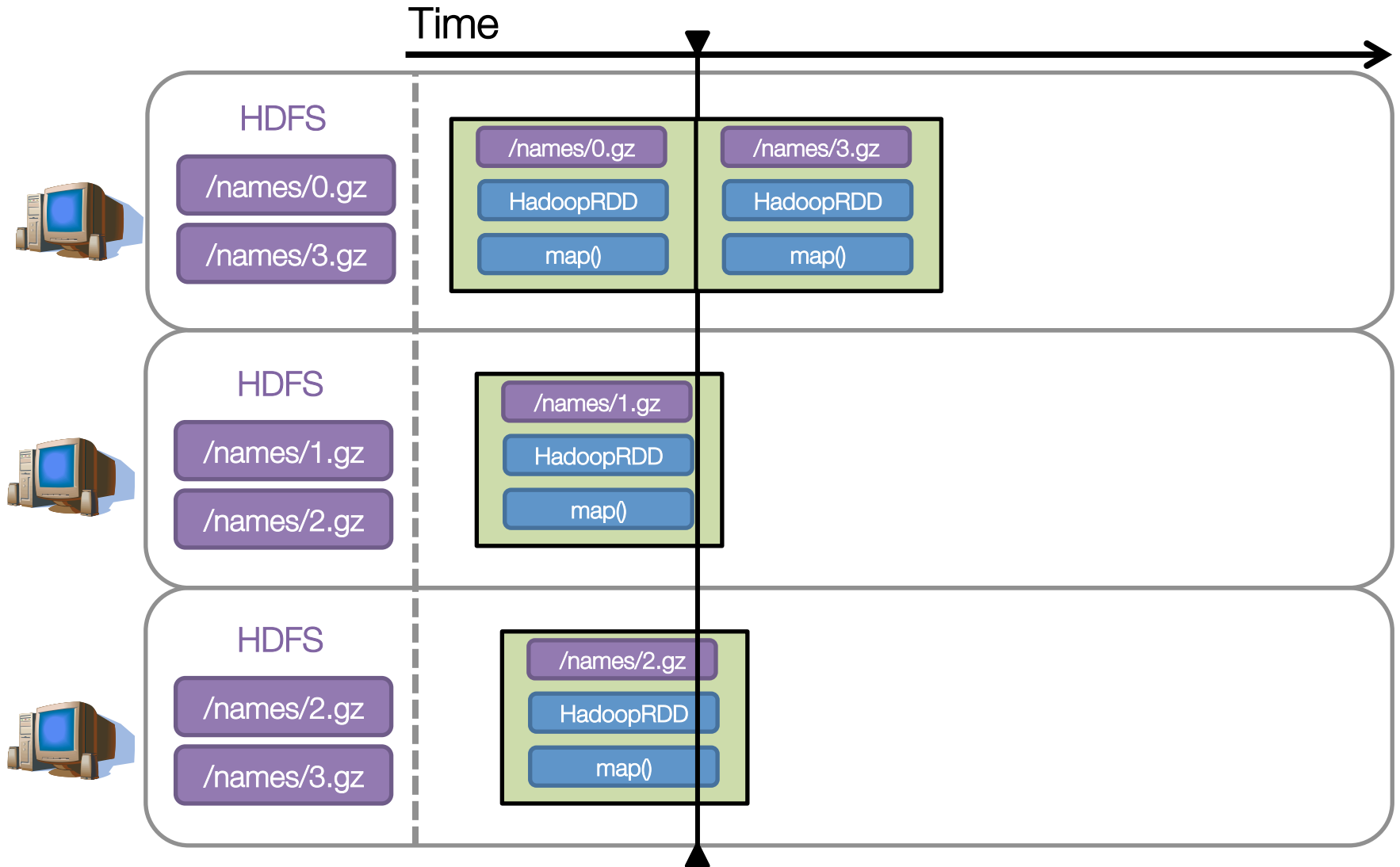
`/names/3.gz`

`/names/2.gz`

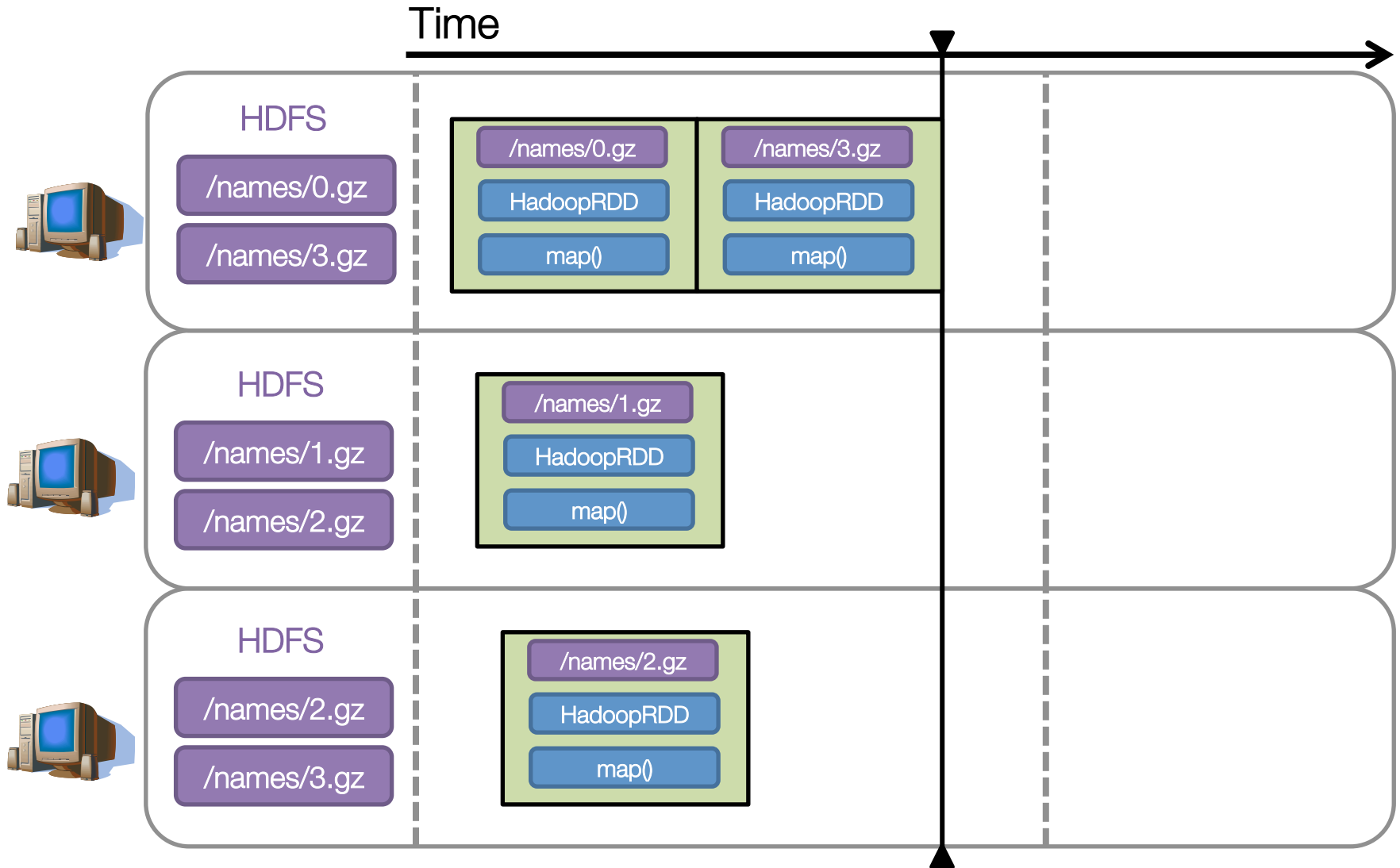
`HadoopRDD`

`map()`

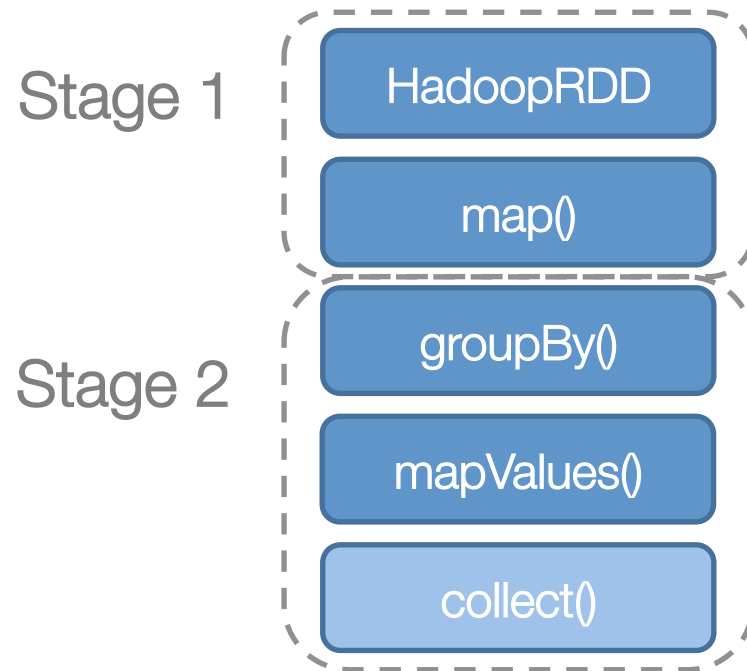
Step 3: Schedule tasks



Step 3: Schedule tasks

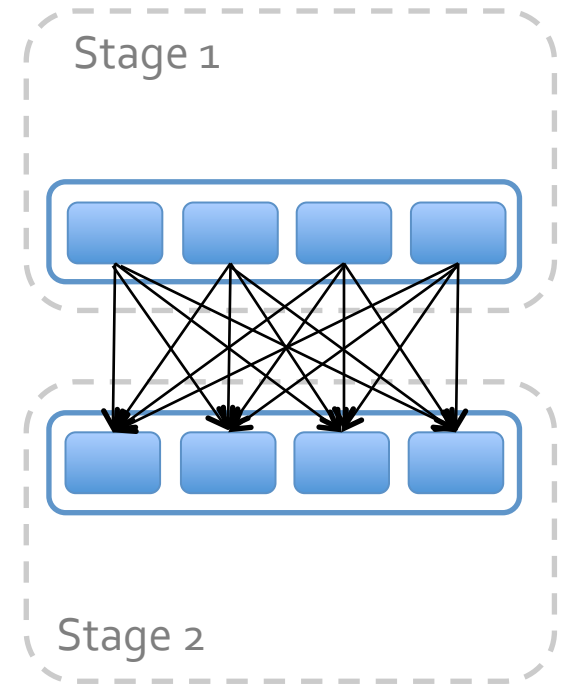


The Shuffle

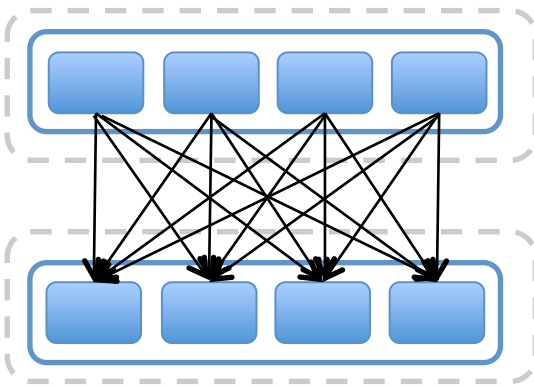


The Shuffle

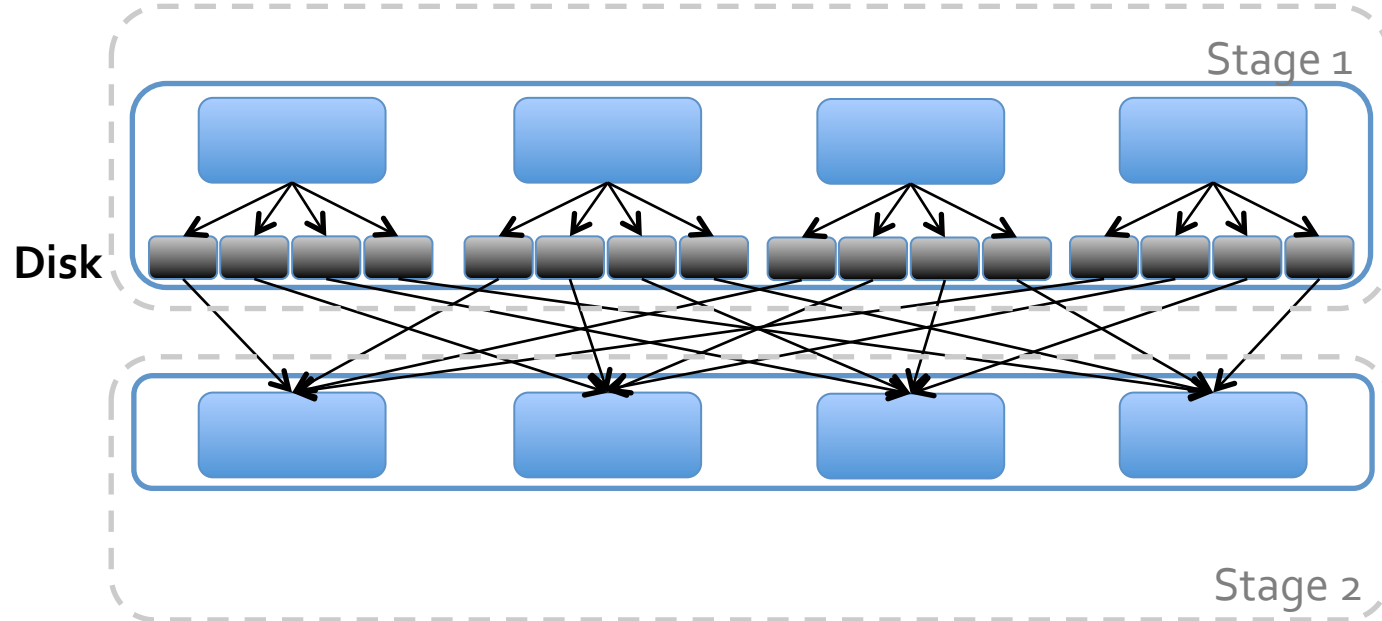
- Redistributes data among partitions
- Hash keys into buckets
- Optimizations:
 - Avoided when possible, if data is already properly partitioned
 - Partial aggregation reduces data movement



The Shuffle



- Pull-based, not push-based
- Write intermediate files to disk



Execution of a groupBy()

- Build hash map within each partition

A => [Arsalan, Aaron, Andrew, Andrew, Andy, Ahir, Ali, ...],

E => [Erin, Earl, Ed, ...]

...

- Note: Can spill across keys, but a single key-value pair must fit in memory

Done!

Stage 1

HadoopRDD

map()

Stage 2

groupBy()

mapValues()

collect()

What went wrong?

- Too few partitions to get good concurrency
- Large per-key groupBy()
- Shipped all data across the cluster

Common issue checklist

1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of sorting and large keys in groupBys)
3. Minimize amount of data shuffled
4. Know the standard library

1 & 2 are about tuning number of partitions!

Importance of Partition Tuning

- Main issue: too few partitions
 - Less concurrency
 - More susceptible to data skew
 - Increased memory pressure for groupBy, reduceByKey, sortByKey, etc.
- Secondary issue: too many partitions
- Need “reasonable number” of partitions
 - Commonly between 100 and 10,000 partitions
 - Lower bound: At least $\sim 2x$ number of cores in cluster
 - Upper bound: Ensure tasks take at least 100ms

Memory Problems

- Symptoms:
 - Inexplicably bad performance
 - Inexplicable executor/machine failures
(can indicate too many shuffle files too)
- Diagnosis:
 - Set `spark.executor.extraJavaOptions` to include
 - `-XX:+PrintGCDetails`
 - `-XX:+HeapDumpOnOutOfMemoryError`
 - Check `dmesg` for oom-killer logs
- Resolution:
 - Increase `spark.executor.memory`
 - Increase number of partitions
 - Re-evaluate program structure (!)

Fixing our mistakes

```
sc.textFile("hdfs:/names")  
  .map(name => (name.charAt(0), name))  
  .groupByKey()  
  .mapValues { names => names.toSet.size }  
  .collect()
```

1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of large groupBys and sorting)
3. Minimize data shuffle
4. Know the standard library

Fixing our mistakes

```
sc.textFile("hdfs:/names")  
  .repartition(6)  
  .map(name => (name.charAt(0), name))  
  .groupByKey()  
  .mapValues { names => names.toSet.size }  
  .collect()
```

1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of large groupBys and sorting)
3. Minimize data shuffle
4. Know the standard library

Fixing our mistakes

```
sc.textFile("hdfs:/names")  
  .repartition(6)  
  .distinct()  
  .map(name => (name.charAt(0), name))  
  .groupByKey()  
  .mapValues { names => names.toSet.size }  
  .collect()
```

1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of large groupBys and sorting)
3. Minimize data shuffle
4. Know the standard library

Fixing our mistakes

```
sc.textFile("hdfs:/names")  
  .repartition(6)  
  .distinct()  
  .map(name => (name.charAt(0), name))  
  .groupByKey()  
  .mapValues { names => names.size }  
  .collect()
```

1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of large groupBys and sorting)
3. Minimize data shuffle
4. Know the standard library

Fixing our mistakes

```
sc.textFile("hdfs:/names")  
  .distinct(numPartitions = 6)  
  .map(name => (name.charAt(0), name))  
  .groupByKey()  
  .mapValues { names => names.size }  
  .collect()
```

1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of large groupBys and sorting)
3. Minimize data shuffle
4. Know the standard library

Fixing our mistakes

```
sc.textFile("hdfs:/names")  
  .distinct(numPartitions = 6)  
  .map(name => (name.charAt(0), 1))  
  .reduceByKey(_ + _)  
  .collect()
```

1. Ensure enough partitions for concurrency
2. Minimize memory consumption (esp. of large groupBys and sorting)
3. Minimize data shuffle
4. Know the standard library

Fixing our mistakes

```
sc.textFile("hdfs:/names")  
  .distinct(numPartitions = 6)  
  .map(name => (name.charAt(0), 1))  
  .reduceByKey(_ + _)  
  .collect()
```

Original:

```
sc.textFile("hdfs:/names")  
  .map(name => (name.charAt(0), name))  
  .groupByKey()  
  .mapValues { names => names.toSet.size }  
  .collect()
```



DATABRICKS

Questions?