Spark RDD Persistence

Chapter 15

201509

Course Chapters

Chapter	Title	Section
1	Introduction	Course Introduction
2	Introduction to Hadoop and the Hadoop Ecosystem	Introduction to Hadoop
3	Hadoop Architecture and HDFS	Introduction to Hadoop
4	Importing Relational Data with Apache Sqoop	Importing and Modeling Structured Data
5	Introduction to Impala and Hive	Importing and Modeling Structured Data
6	Modeling and Managing Data with Impala and Hive	Importing and Modeling Structured Data
7	Data Formats	Importing and Modeling Structured Data
8	Data File Partitioning	Importing and Modeling Structured Data
9	Capturing Data with Apache Flume	Ingesting Streaming Data
10	Spark Basics	**Distributed Data Processing with Spark**
11	Working with RDDs in Spark	**Distributed Data Processing with Spark**
12	Aggregating Data with Pair RDDs	**Distributed Data Processing with Spark**
13	Writing and Deploying Spark Applications	**Distributed Data Processing with Spark**
14	Parallel Processing in Spark	**Distributed Data Processing with Spark**
15	**Spark RDD Persistence**	**Distributed Data Processing with Spark**
16	Common Patterns in Spark Data Processing	**Distributed Data Processing with Spark**
17	Spark SQL and DataFrames	**Distributed Data Processing with Spark**
18	Conclusion	Course Conclusion

Spark RDD Persistence

In this chapter you will learn

- **How Spark uses an RDD's lineage in operations**
- **How to persist RDDs to improve performance**

Chapter Topics

Spark RDD Persistence	Distributed Data Processing with Spark

- **RDD Lineage**
- RDD Persistence Overview
- Distributed Persistence
- Conclusion
- Homework: Persist an RDD

Lineage Example (1)

- **Each *transformation* operation creates a new *child* RDD**

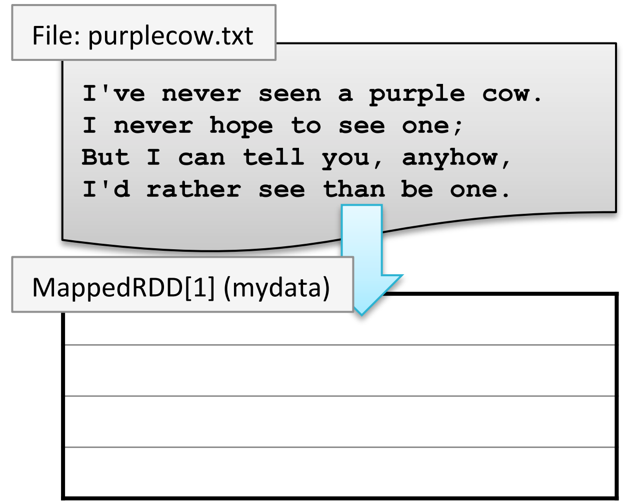
File: purplecow.txt

```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```


Lineage Example (2)

- **Each *transformation* operation creates a new *child* RDD**

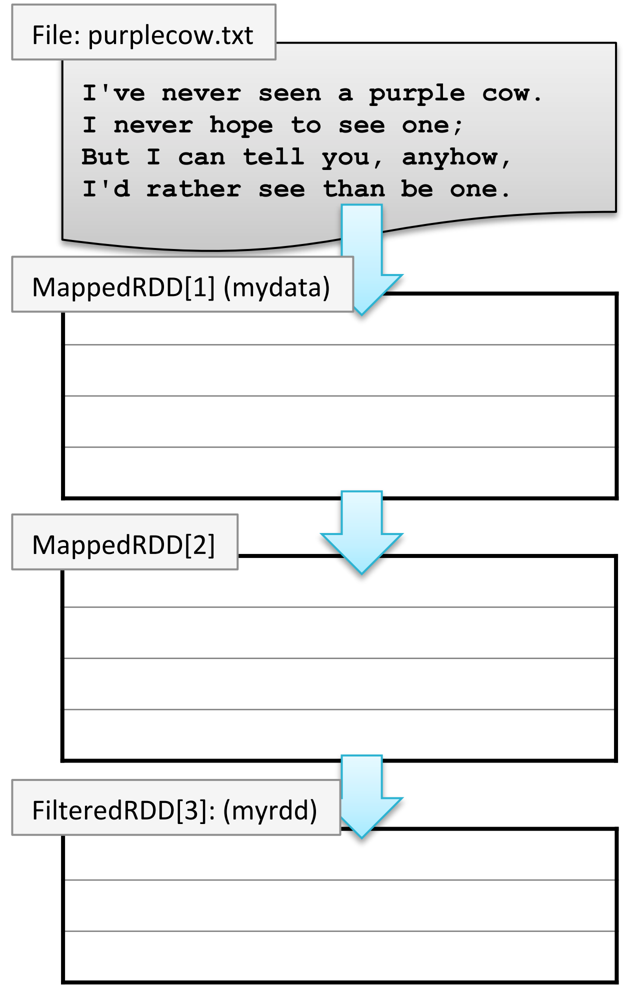
```
> mydata = sc.textFile("purplecow.txt")
```



Lineage Example (3)

- **Each *transformation* operation creates a new *child* RDD**

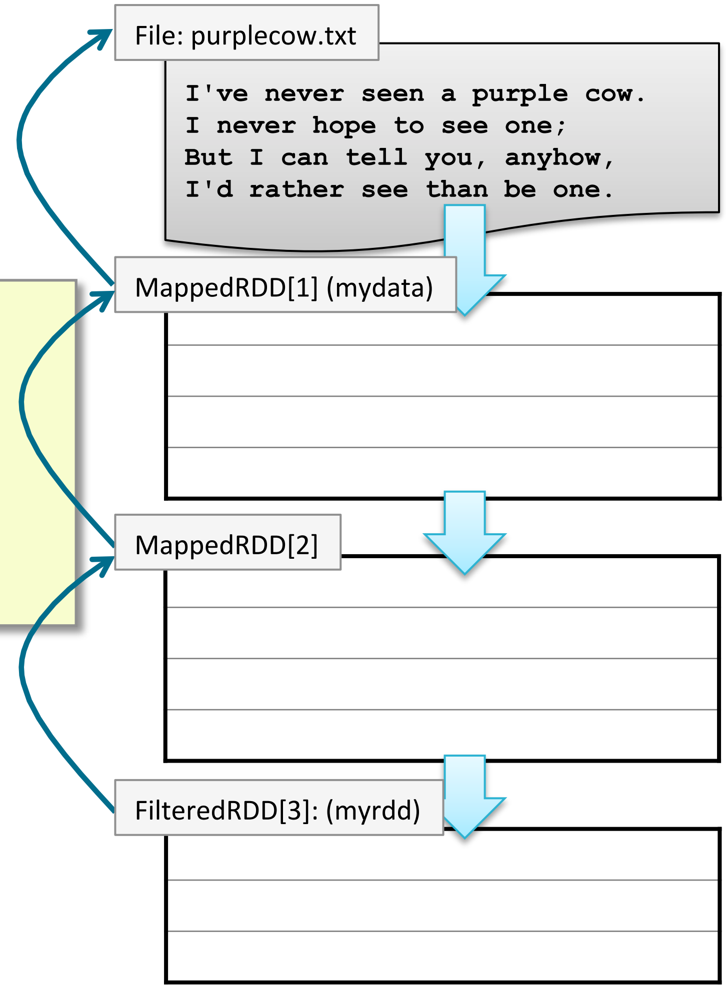
```
> mydata = sc.textFile("purplecow.txt")
> myrdd = mydata.map(lambda s: s.upper())\
  .filter(lambda s:s.startswith('I'))
```



Lineage Example (4)

- **Spark keeps track of the *parent* RDD for each new RDD**
- **Child RDDs *depend on* their parents**

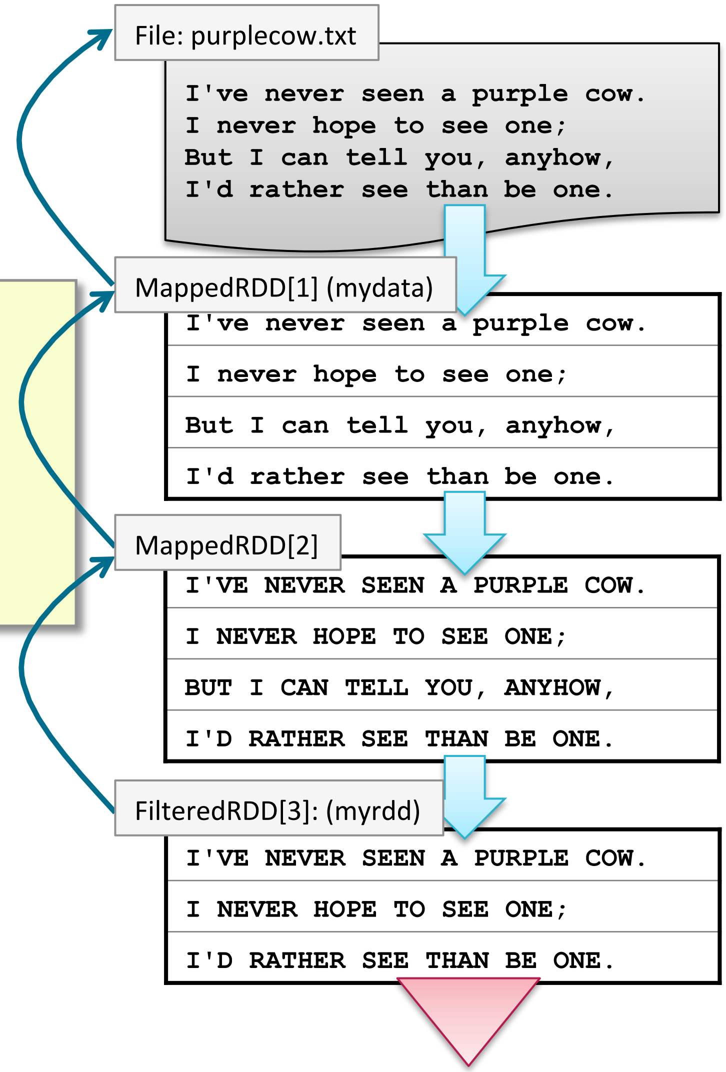
```
> mydata = sc.textFile("purplecow.txt")
> myrdd = mydata.map(lambda s: s.upper())\
  .filter(lambda s:s.startswith('I'))
```



Lineage Example (5)

- ***Action*** **operations execute the parent transformations**

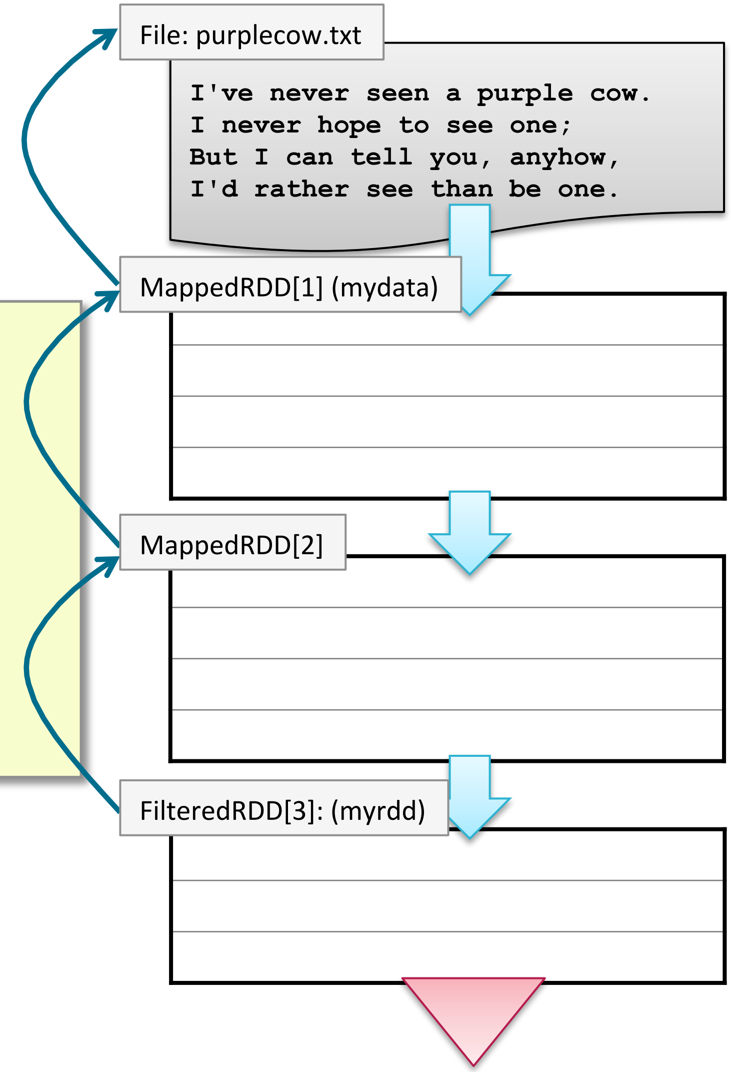
```
> mydata = sc.textFile("purplecow.txt")
> myrdd = mydata.map(lambda s: s.upper())\
  .filter(lambda s:s.startswith('I'))
> myrdd.count()
3
```



Lineage Example (6)

- **Each action re-executes the lineage transformations starting with the base**
  - By default

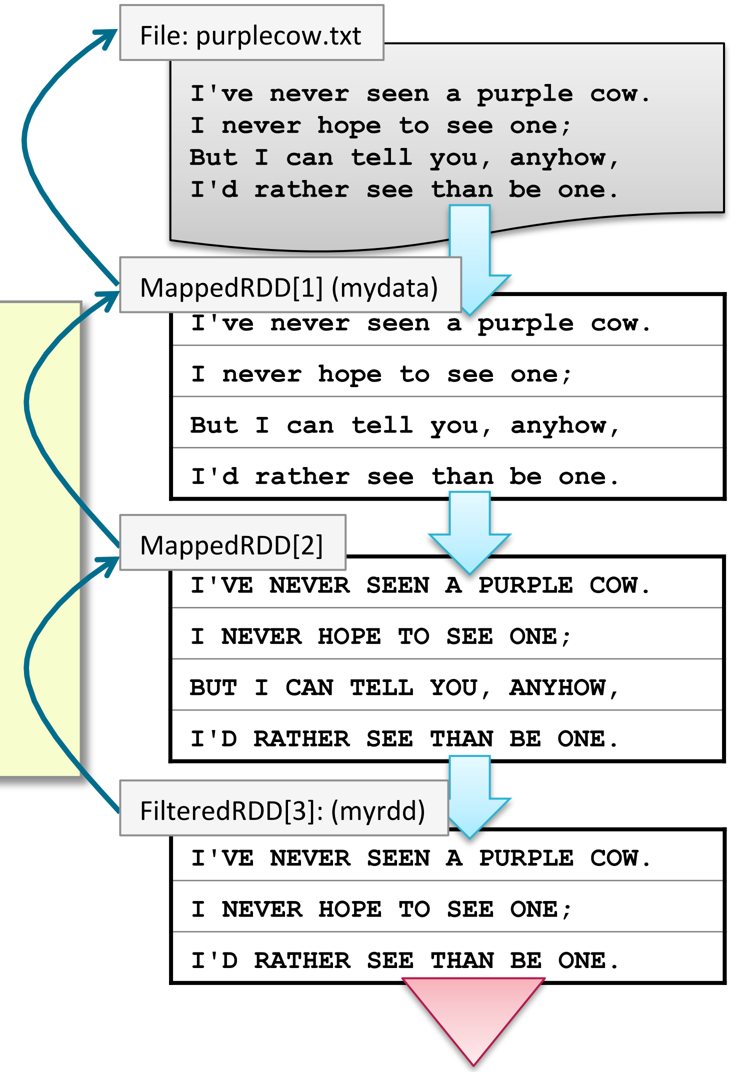
```
> mydata = sc.textFile("purplecow.txt")
> myrdd = mydata.map(lambda s: s.upper())\
  .filter(lambda s:s.startswith('I'))
> myrdd.count()
3
> myrdd.count()
```



Lineage Example (7)

- **Each action re-executes the lineage transformations starting with the base**
  - By default

```
>  mydata = sc.textFile("purplecow.txt")
>  myrdd = mydata.map(lambda s: s.upper())\
  .filter(lambda s:s.startswith('I'))
>  myrdd.count()
3
>  myrdd.count()
3
```



Chapter Topics

Spark RDD Persistence	Distributed Data Processing with Spark

- RDD Lineage
- **RDD Persistence Overview**
- Distributed Persistence
- Conclusion
- Homework: Persist an RDD

RDD Persistence

- **Persisting an RDD saves the data (by default in memory)**

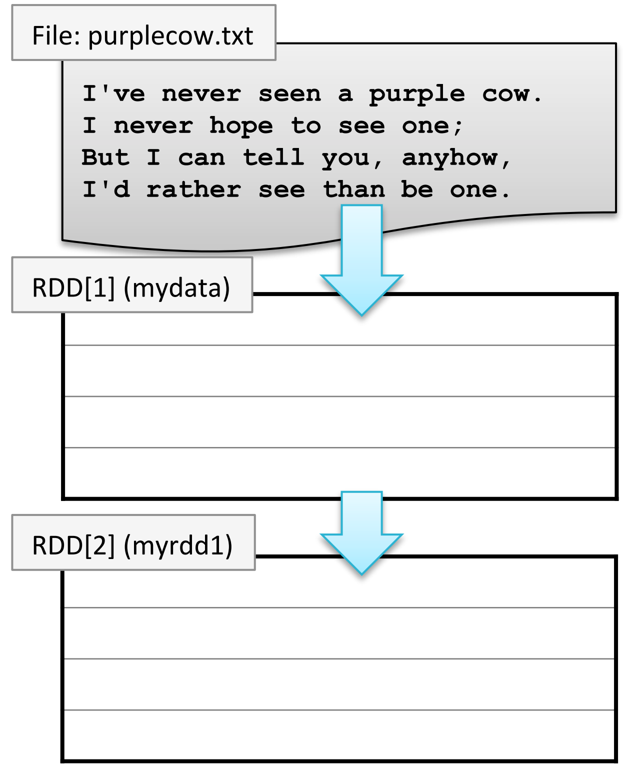
File: purplecow.txt

```
I've never seen a purple cow.
I never hope to see one;
But I can tell you, anyhow,
I'd rather see than be one.
```

RDD Persistence

- **Persisting an RDD saves the data (by default in memory)**

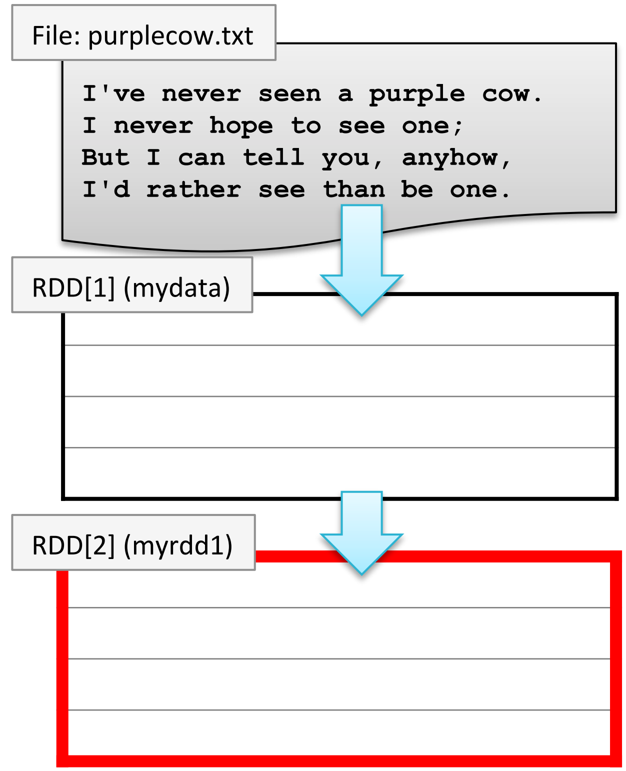
```
> mydata = sc.textFile("purplecow.txt")
> myrdd1 = mydata.map(lambda s:
  s.upper())
```



RDD Persistence

- **Persisting an RDD saves the data (by default in memory)**

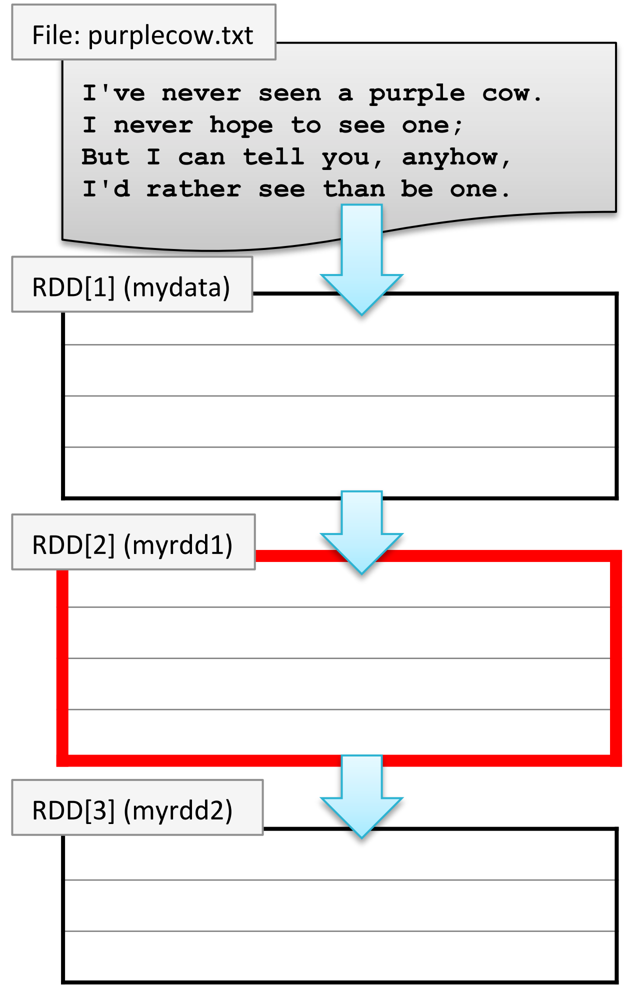
```
> mydata = sc.textFile("purplecow.txt")
> myrdd1 = mydata.map(lambda s:
    s.upper())
> myrdd1.persist()
```



RDD Persistence

- **Persisting an RDD saves the data (by default in memory)**

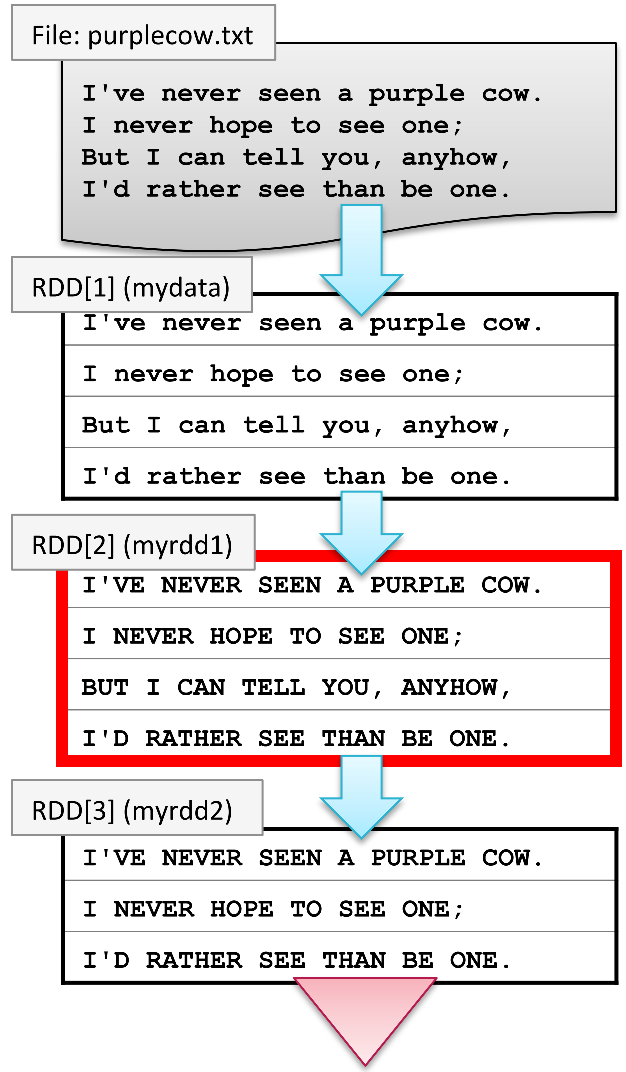
```
> mydata = sc.textFile("purplecow.txt")
> myrdd1 = mydata.map(lambda s:
  s.upper())
> myrdd1.persist()
> myrdd2 = myrdd1.filter(lambda \
  s:s.startswith('I'))
```



RDD Persistence

- **Persisting an RDD saves the data (by default in memory)**

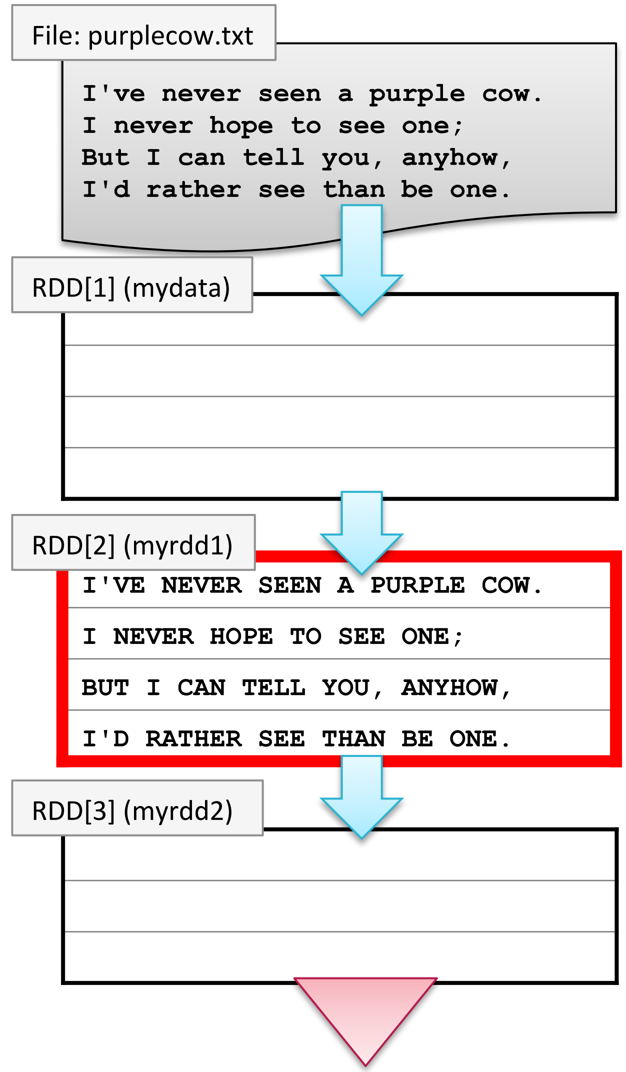
```
> mydata = sc.textFile("purplecow.txt")
> myrdd1 = mydata.map(lambda s:
  s.upper())
> myrdd1.persist()
> myrdd2 = myrdd1.filter(lambda \
  s:s.startswith('I'))
> myrdd2.count()
3
```



RDD Persistence

- **Subsequent operations use saved data**

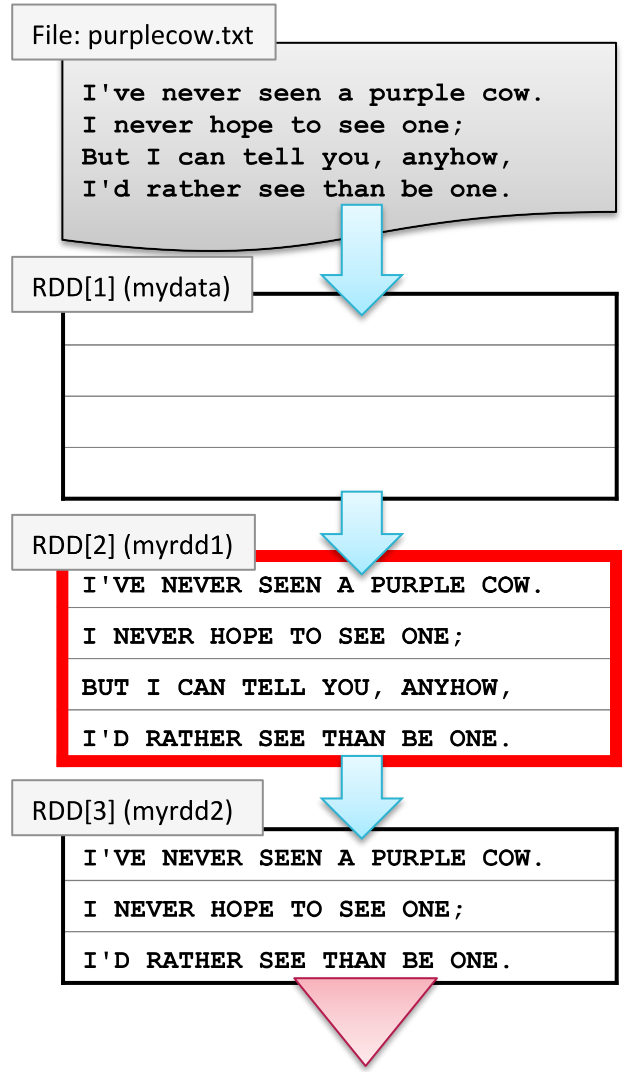
```
> mydata = sc.textFile("purplecow.txt")
> myrdd1 = mydata.map(lambda s:
  s.upper())
> myrdd1.persist()
> myrdd2 = myrdd1.filter(lambda \
  s:s.startswith('I'))
> myrdd2.count()
3
> myrdd2.count()
```



RDD Persistence

- **Subsequent operations use saved data**

```
> my data =
  sc.textFile("purplecow.txt")
> myrdd1 = mydata.map(lambda s:
  s.upper())
> myrdd1.persist()
> myrdd2 = myrdd1.filter(lambda \
  s:s.startswith('I'))
> myrdd2.count()
3
> myrdd2.count()
3
```



cloudera

Memory Persistence

- **In-memory persistence is a *suggestion* to Spark**
  - If not enough memory is available, persisted partitions will be cleared from memory
    - Least recently used partitions cleared first
  - Transformations will be re-executed using the lineage when needed

Chapter Topics

Spark RDD Persistence	Distributed Data Processing with Spark

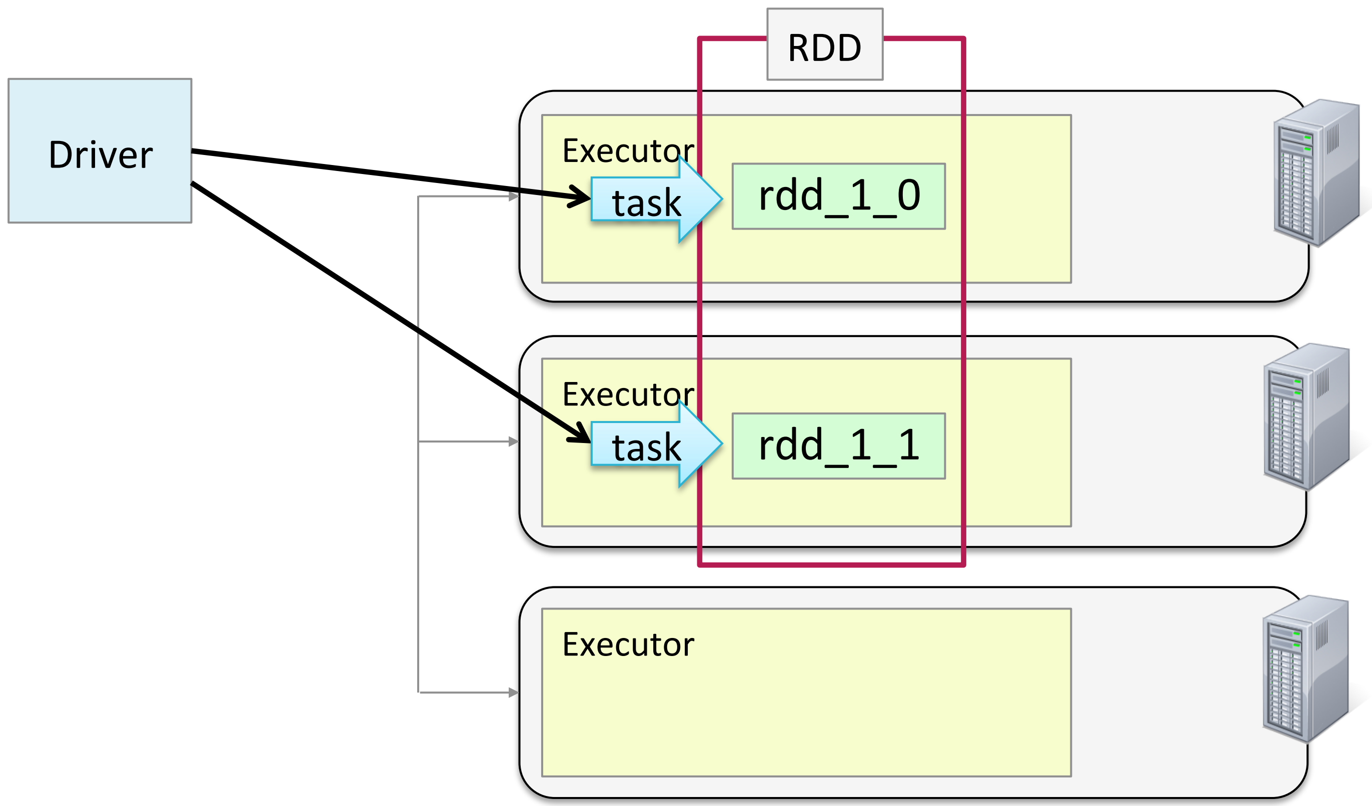
- RDD Lineage
- RDD Persistence Overview
- **Distributed Persistence**
- Conclusion
- Homework: Persist an RDD

Persistence and Fault-Tolerance

- **RDD = *Resilient* Distributed Dataset**
  - Resiliency is a product of tracking lineage
  - RDDs can always be recomputed from their base if needed

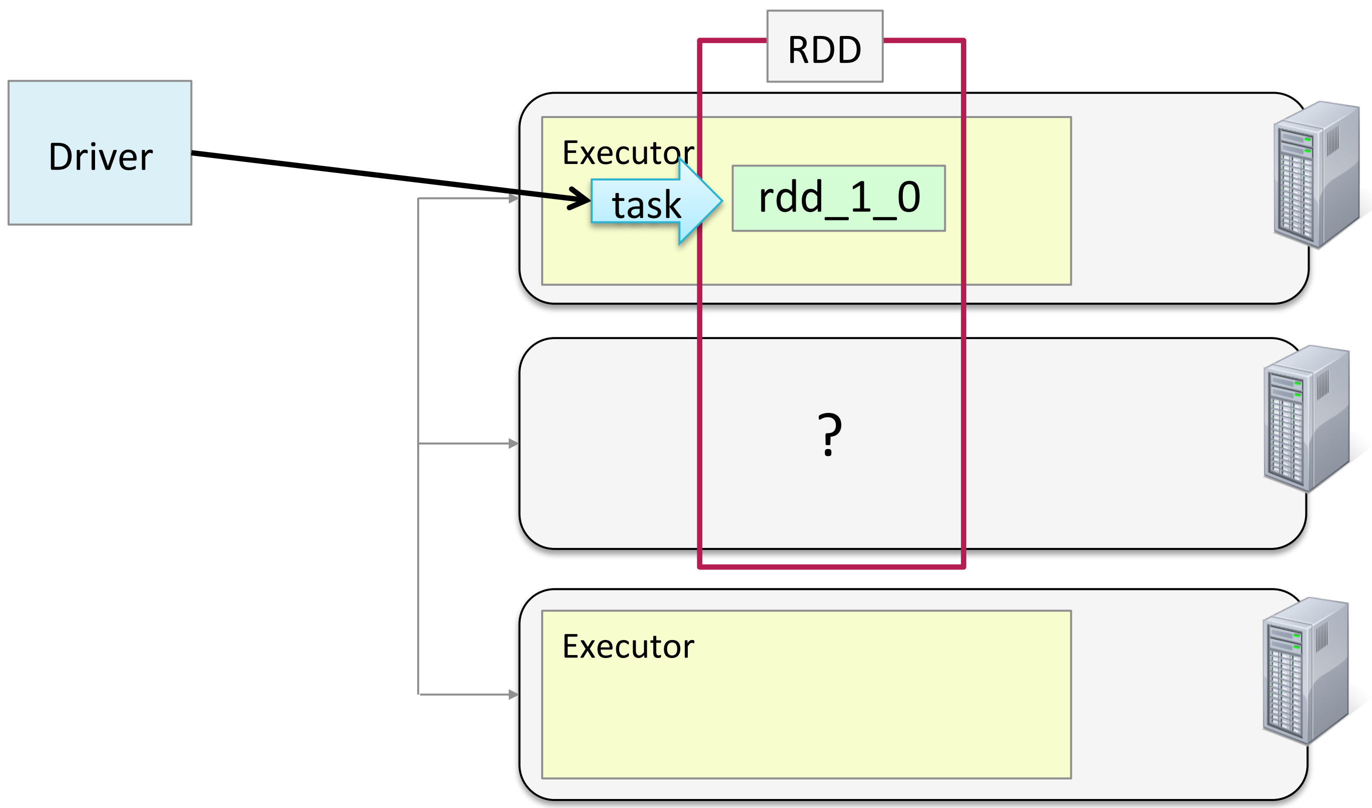
Distributed Persistence

- **RDD partitions are distributed across a cluster**
- **By default, partitions are persisted in memory in Executor JVMs**



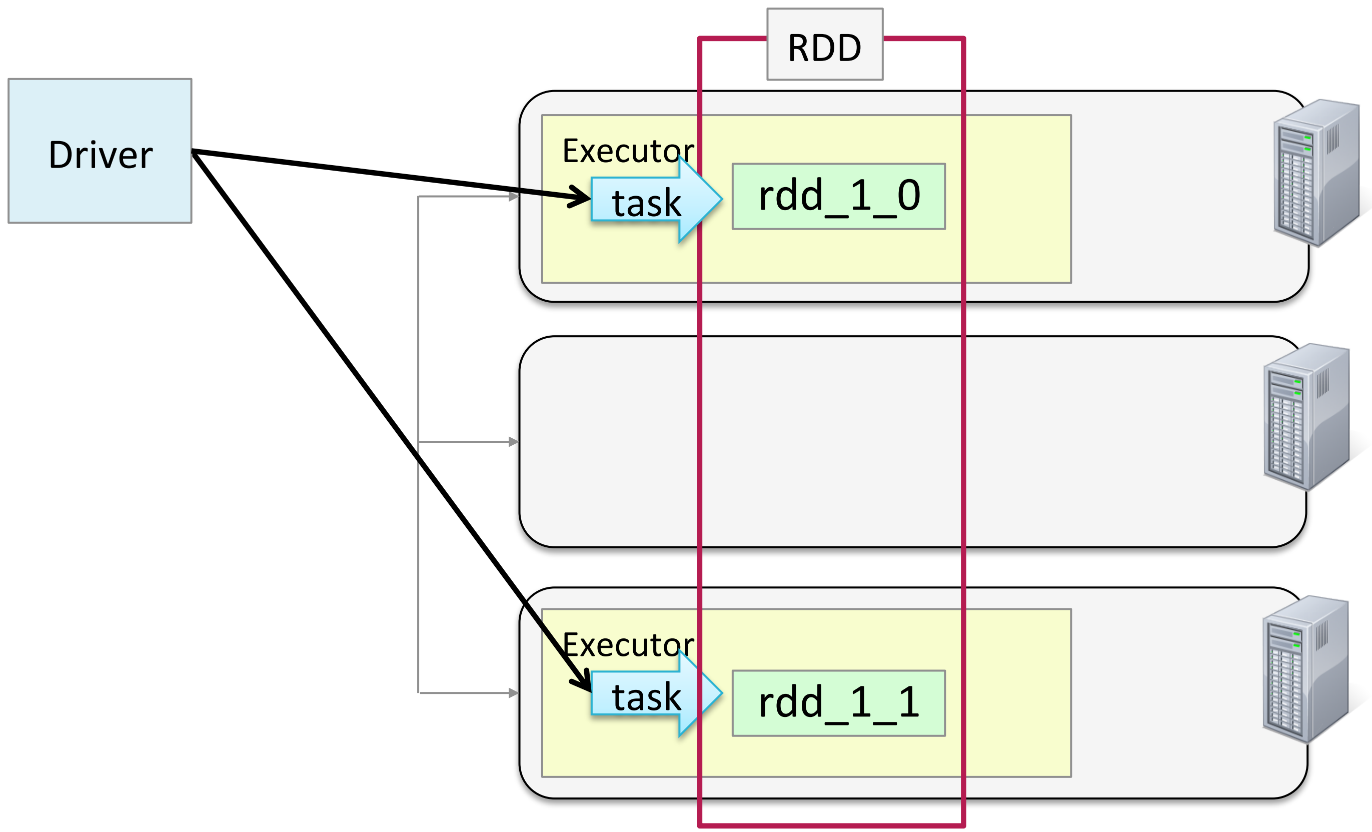
RDD Fault-Tolerance (1)

- **What happens if a partition persisted in memory becomes unavailable?**



RDD Fault-Tolerance (2)

- **The driver starts a new task to recompute the partition on a different node**
- **Lineage is preserved, data is never lost**



Persistence Levels

- **By default, the `persist` method stores data in memory only**
  - The **`cache`** method is a synonym for default (memory) persist
- **The `persist` method offers other options called Storage Levels**
- **Storage Levels let you control**
  - Storage location
  - Format in memory
  - Partition replication

Persistence Levels: Storage Location

- **Storage location – where is the data stored?**
  - **`MEMORY_ONLY`** (default) – same as **`cache`**
  - **`MEMORY_AND_DISK`** – Store partitions on disk if they do not fit in memory
    - Called *spilling*
  - **`DISK_ONLY`** – Store all partitions on disk

Python

```
> from pyspark import StorageLevel
> myrdd.persist(StorageLevel.DISK_ONLY)
```

Scala

```
> import org.apache.spark.storage.StorageLevel
> myrdd.persist(StorageLevel.DISK_ONLY)
```

Persistence Levels: Memory Format

- **Serialization – you can choose to serialize the data in memory**
  - `MEMORY_ONLY_SER` and `MEMORY_AND_DISK_SER`
  - Much more space efficient
  - Less time efficient
    - If using Java or Scala, choose a fast serialization library (e.g. Kryo)

Persistence Levels: Partition Replication

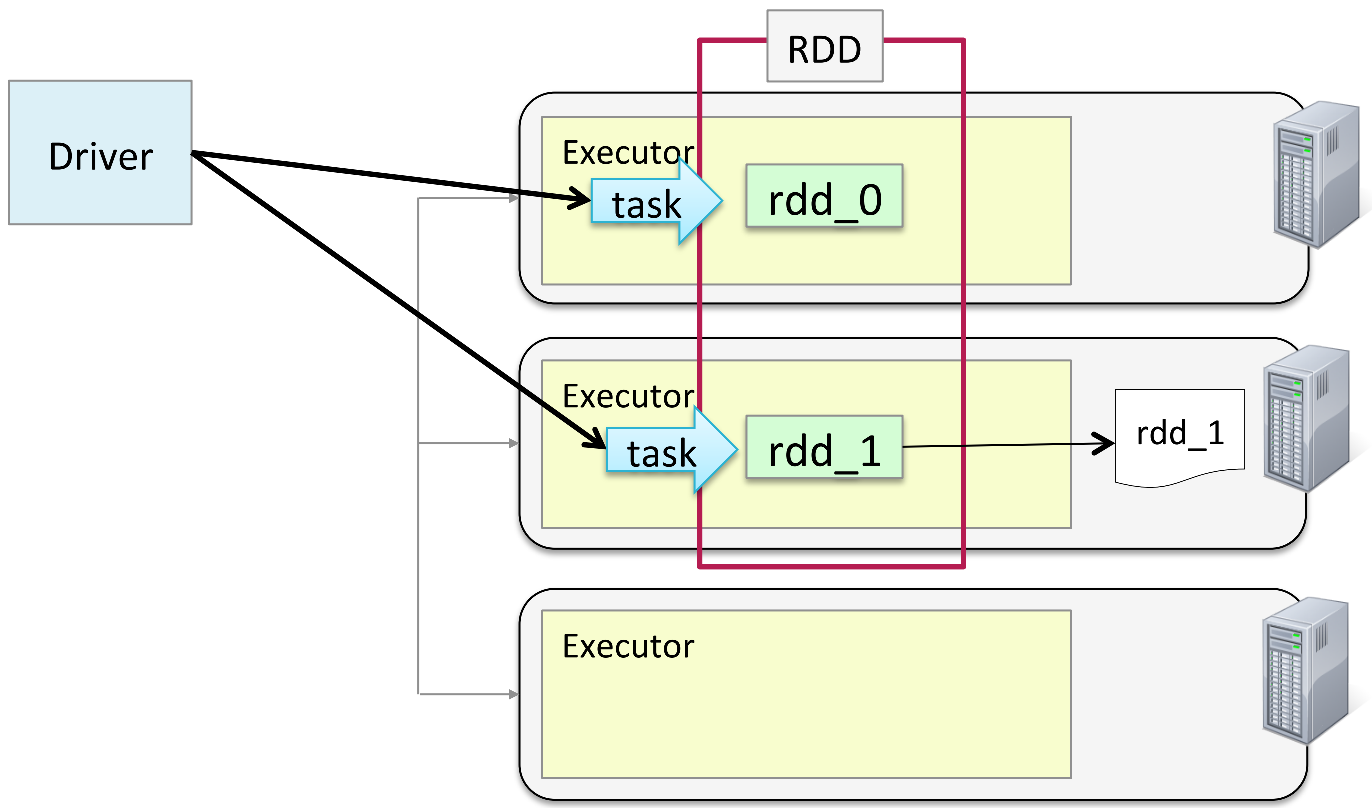
- **Replication – store partitions on two nodes**
  - **`MEMORY_ONLY_2`**
  - **`MEMORY_AND_DISK_2`**
  - **`DISK_ONLY_2`**
  - **`MEMORY_AND_DISK_SER_2`**
  - **`DISK_ONLY_2`**
  - You can also define custom storage levels

Changing Persistence Options

- **To stop persisting and remove from memory and disk**
  - `rdd.unpersist()`
- **To change an RDD to a different persistence level**
  - Unpersist first

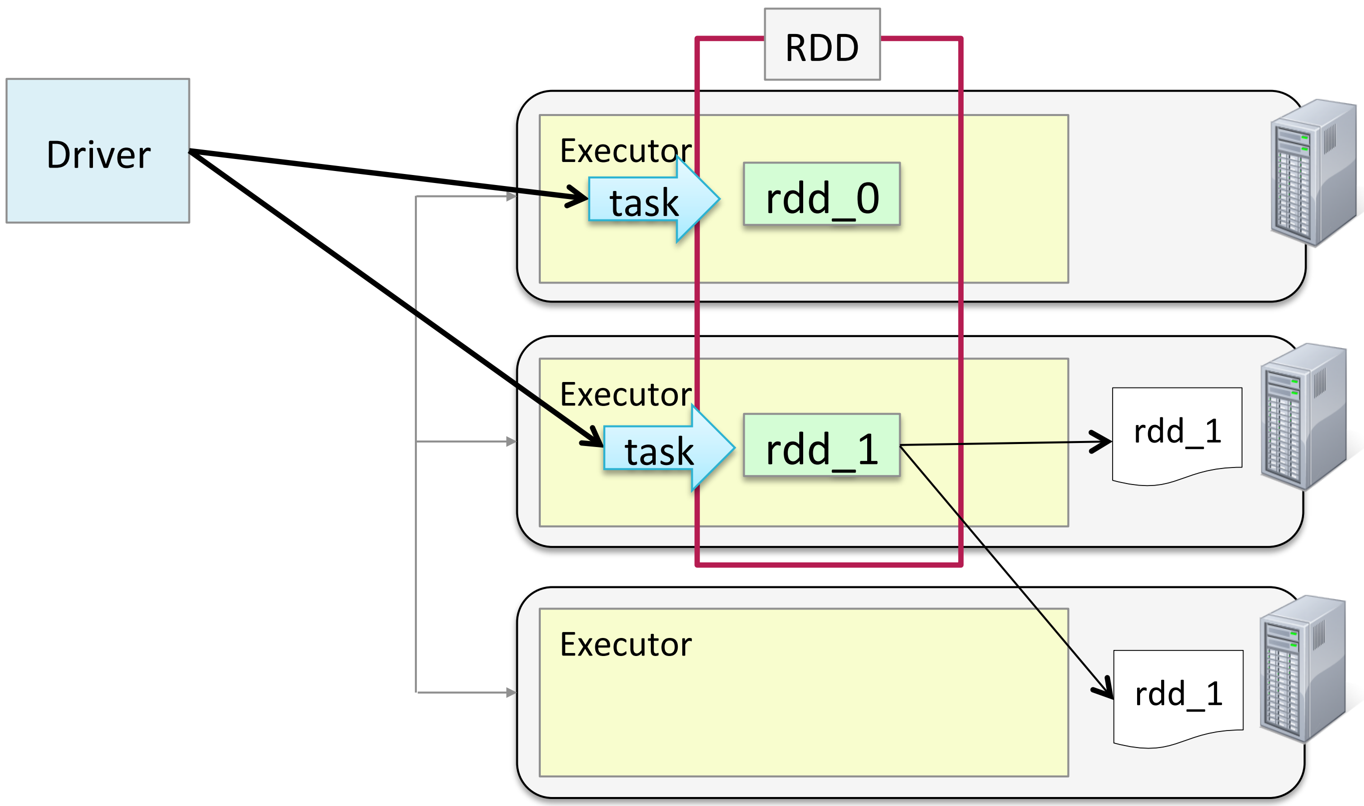
Disk Persistence

- **Disk-persisted partitions are stored in local files**



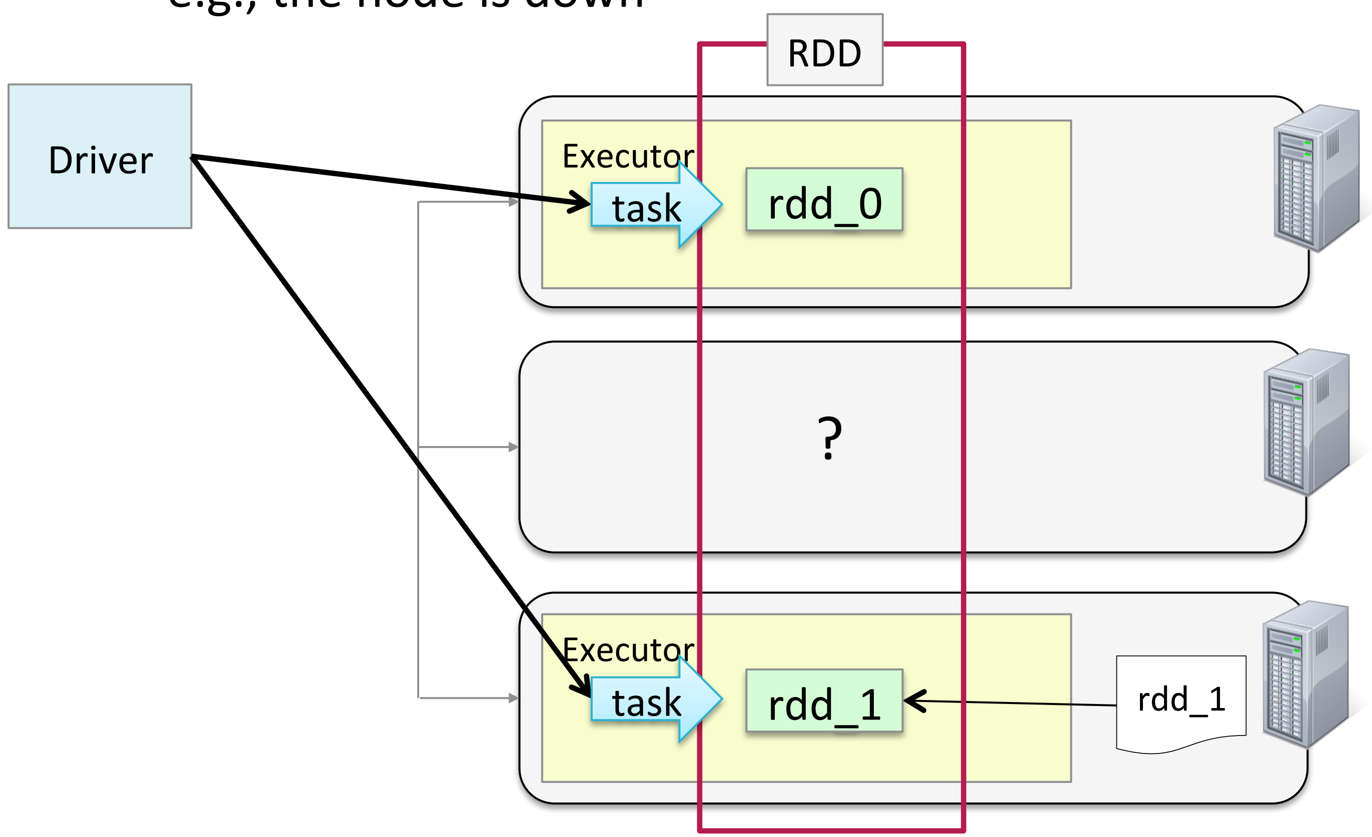
Disk Persistence with Replication (1)

- **Persistence replication makes recomputation less likely to be necessary**



Disk Persistence with Replication (2)

- **Replicated data on disk will be used to recreate the partition if possible**
  - Will be recomputed if the data is unavailable
    - e.g., the node is down



When and Where to Persist

- **When should you persist a dataset?**
  - When a dataset is likely to be re-used
    - e.g., iterative algorithms, machine learning
- **How to choose a persistence level**
  - Memory only – when possible, best performance
    - Save space by saving as serialized objects in memory if necessary
  - Disk – choose when recomputation is more expensive than disk read
    - e.g., expensive functions or filtering large datasets
  - Replication – choose when recomputation is more expensive than memory

Chapter Topics

Spark RDD Persistence	Distributed Data Processing with Spark

- RDD Lineage
- RDD Persistence Overview
- Distributed Persistence
- **Conclusion**
- Homework: Persist an RDD

Essential Points

- **Spark keeps track of each RDD's lineage**
  - Provides fault tolerance
- **By default, every RDD operation executes the entire lineage**
- **If an RDD will be used multiple times, persist it to avoid re-computation**
- **Persistence options**
  - Location – memory only, memory and disk , disk only
  - Format – in-memory data can be serialized to save memory (but at the cost of performance)
  - Replication – saves data on multiple nodes in case a node goes down, for job recovery without recomputation

Chapter Topics

Spark RDD Persistence	Distributed Data Processing with Spark

- RDD Lineage
- RDD Persistence Overview
- Distributed Persistence
- Conclusion
- **Homework: Persist an RDD**

Homework: Persist an RDD

- **In this homework assignment you will**
  - Persist an RDD before reusing it
  - Use the Spark Application UI to see how an RDD is persisted
- **Please refer to the Homework description**