Spark RDD

DS 5110/CS 5501: Big Data Systems

Spring 2024

Lecture 5a

Yue Cheng

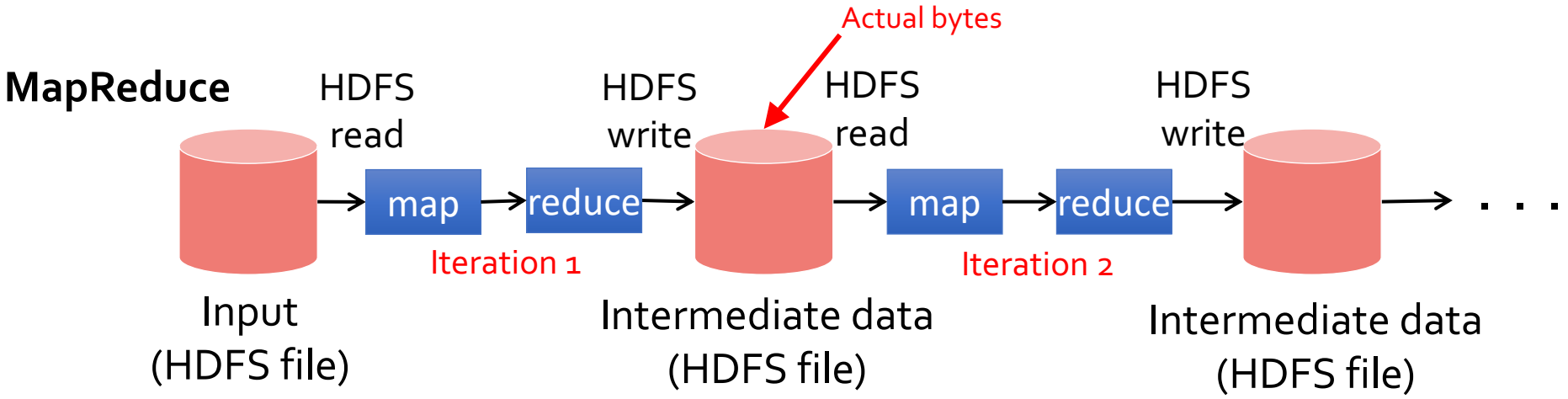
Some material taken/derived from:

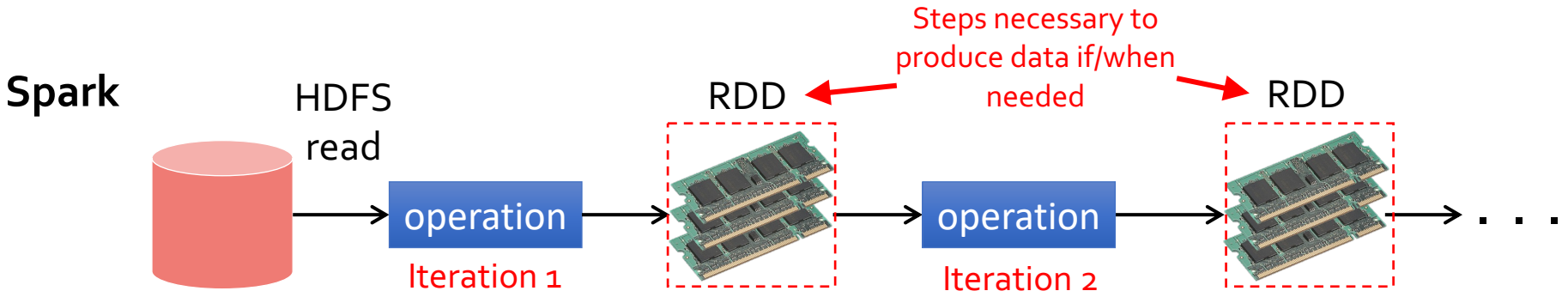
- Wisconsin CS 320 by Tyler Caraza-Harter.

Learning objectives

- The motivation of Spark RDD
- The difference between RDD transformations and actions
- The benefits of the RDD abstraction

Intermediate data: MapReduce vs. Spark





Resilient Distributed Datasets (RDD)

- Data lineage: Record series of operations on other data necessary to obtain results
- Lazy evaluation: Computation only done when results needed (to write file, make plot, etc.)
- Immutability: You can't change an RDD, but you can define a new one in terms of another

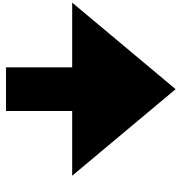
Data lineage: Transformations & Actions

```
data = [
    (“A”, 1),
    (“B”, 2),
    (“A”, 3),
    (“B”, 4)
]
```

```
def mult2(row):
    return (row[0], row[1]*2)

def onlyA(row):
    return row[0] == “A”
```

Goal: Get 2 times the second column wherever the first column is “A”

```
table = sc.parallelize(data)
double = table.map(mult2)
doubleA = double.filter(onlyA)
doubleA.collect()
```



```
[(‘A’, 2),
 (‘A’, 6)]
```

The computation is a sequence of 4 operations. Operations come in two types:

- Transformation: Create a new RDD (lazy, so no execution yet). Here: `parallelize`, `map`, and `filter`.
- Action: Perform all operations in the graph to get an actual result. Here: `collect`.

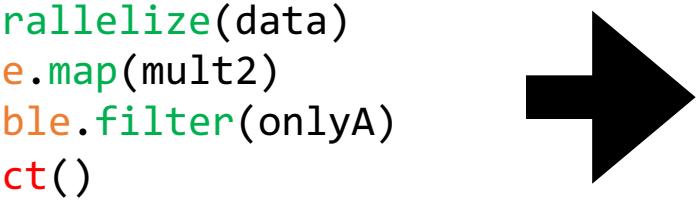
Data lineage: Transformations & Actions

```
data = [
    (“A”, 1),
    (“B”, 2),
    (“A”, 3),
    (“B”, 4)
]
```

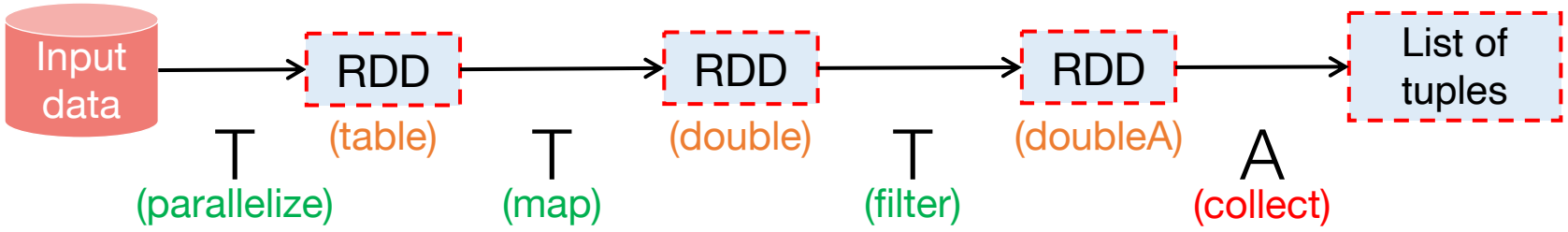
```
def mult2(row):
    return (row[0], row[1]*2)

def onlyA(row):
    return row[0] == “A”
```

Goal: Get 2 times the second column wherever the first column is “A”

```
table = sc.parallelize(data)
double = table.map(mult2)
doubleA = double.filter(onlyA)
doubleA.collect()
```



```
[(‘A’, 2),
 (‘A’, 6)]
```



Q: Are there alternative paths you could create from the start to end node?

Optimization

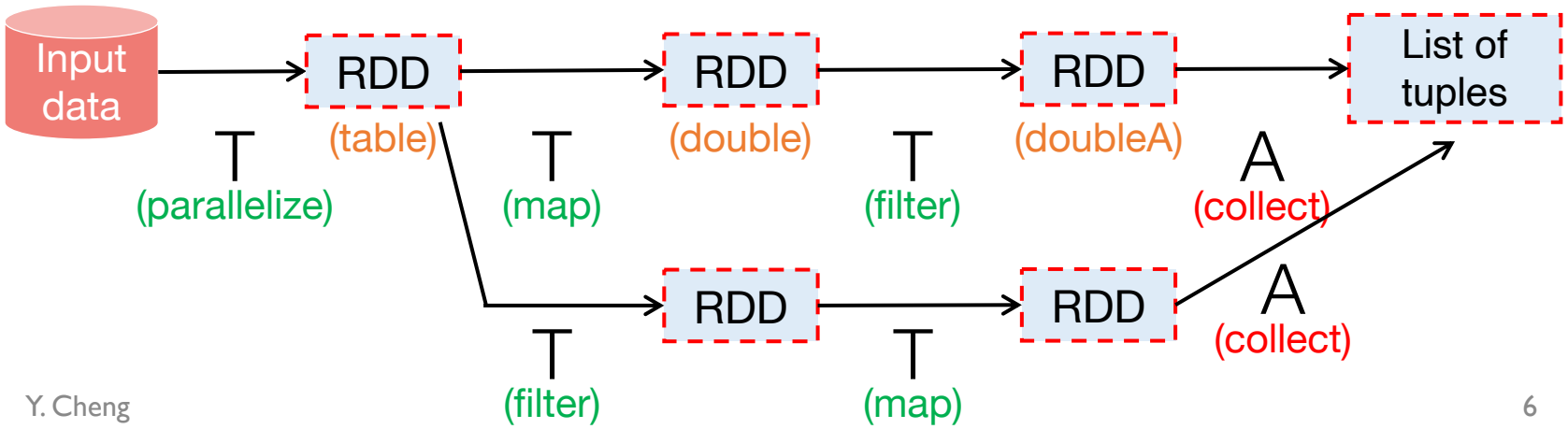
Transformation vs. action:

- Transformation: intermediate results (means to an end)
- Action: Final results we care about
- This distinction creates opportunities for **optimize** (choosing a more efficient sequence of transformations to get the same result + pipelining the compute)

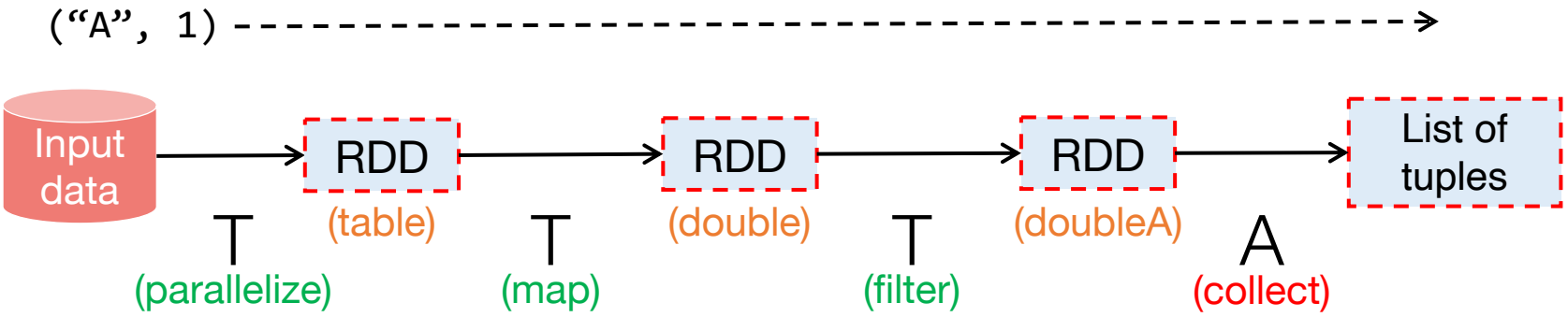
Goal: Get 2 times the second column wherever the first column is "A"

```
table = sc.parallelize(data)
double = table.map(mult2)
doubleA = double.filter(onlyA)
doubleA.collect()
```

```
[('A', 2),
 ('A', 6)]
```



Partitions



At what granularity should data flow through the transformation?

- Whole dataset: It could all proceed through, one transformation at a time, but might not fit in memory
- Row: In this pipeline, nothing prevents each row from passing through independently, but probably slower than computing in bulk
- Partition: Spark users can specify the number of partitions for an RDD

```
sc.parallelize(data, 1)
    data = [
        ("A", 1),
        ("B", 2),
        ("A", 3),
        ("B", 4)
    ]
```

partition

```
sc.parallelize(data, 2)
    data = [
        ("A", 1),
        ("B", 2),
        ("A", 3),
        ("B", 4)
    ]
```

partition

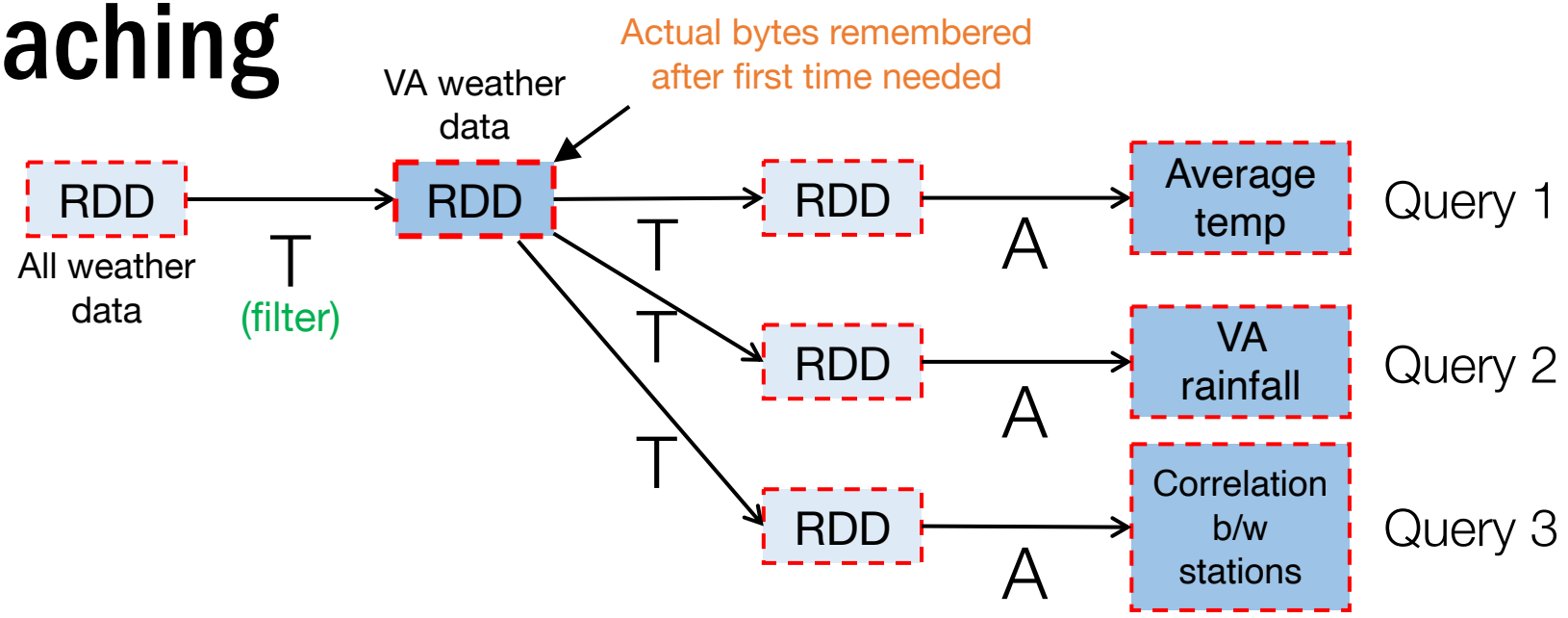
partition

Tasks

- Spark work
  - Spark code is converted to jobs, which consist of stages, which consist of tasks
  - Tasks:
    - Run on a single CPU core
    - Operate on a single partition, which is loaded entirely to memory
- Choosing a partition count directly affects the number of tasks necessary to do a job.

- Advantages of large partitions
  - Less overhead in starting tasks
- Disadvantages of large partitions
  - Might not expose enough parallelism to use all cores available
  - Harder to balance work evenly
  - Uses more memory

Caching



Some RDDs might be used repeatedly

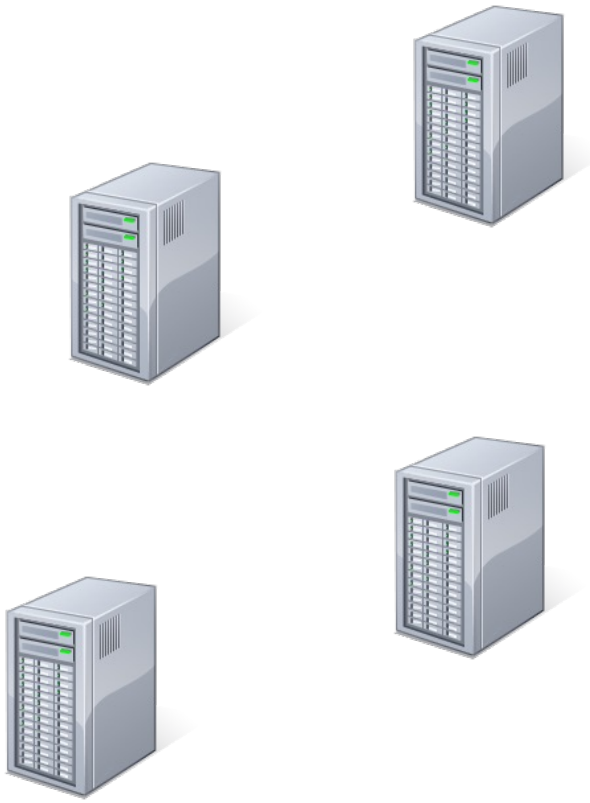
- Spark might cache a copy of the computed results
- OR we can tell it to

```
all_weather = ...
va_weather = all_weather.filter(...)
va_weather.cache()
...
va_weather.unpersist()   # stop caching
```

Putting it all together…

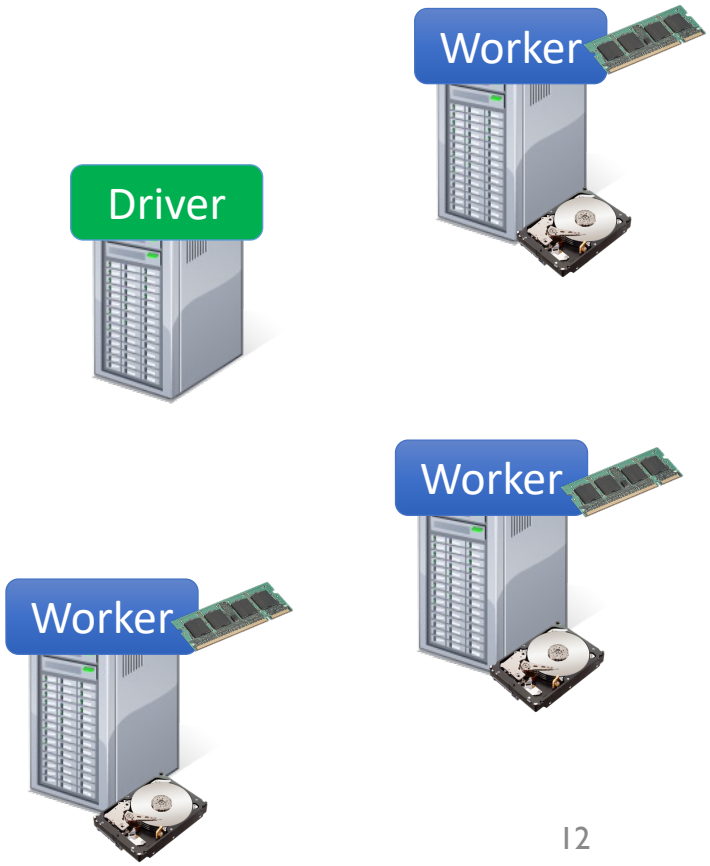
Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns



Interactive debugging (control & data flow)

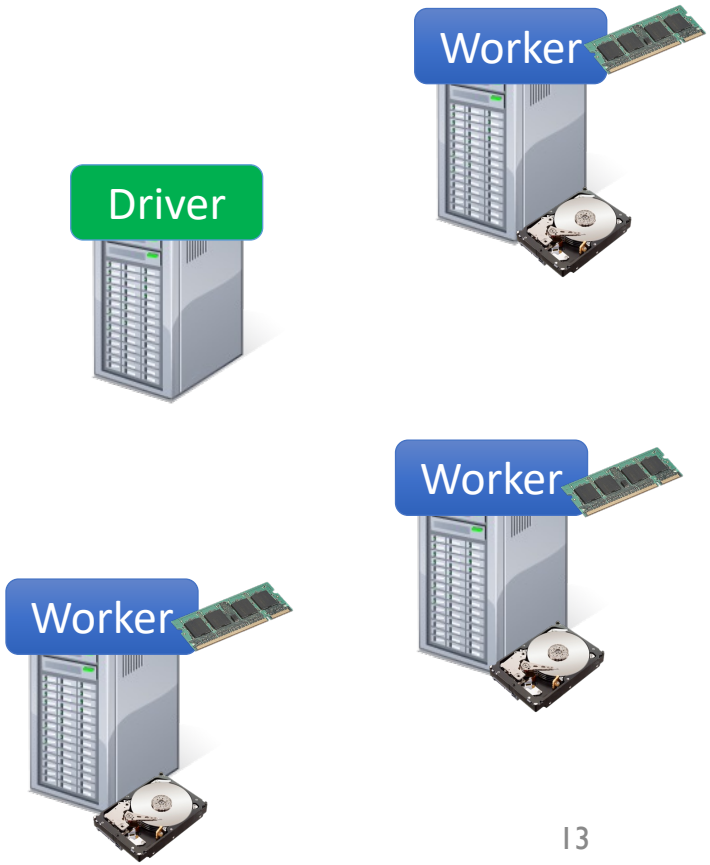
Load input data from an HDFS file into memory, then interactively search for various patterns



Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
```

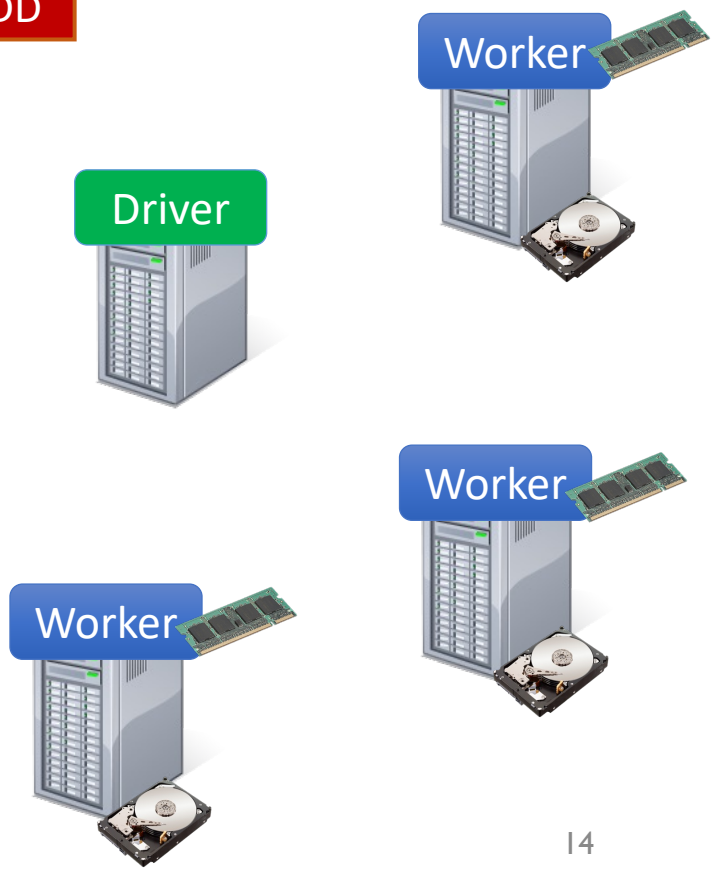


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
```

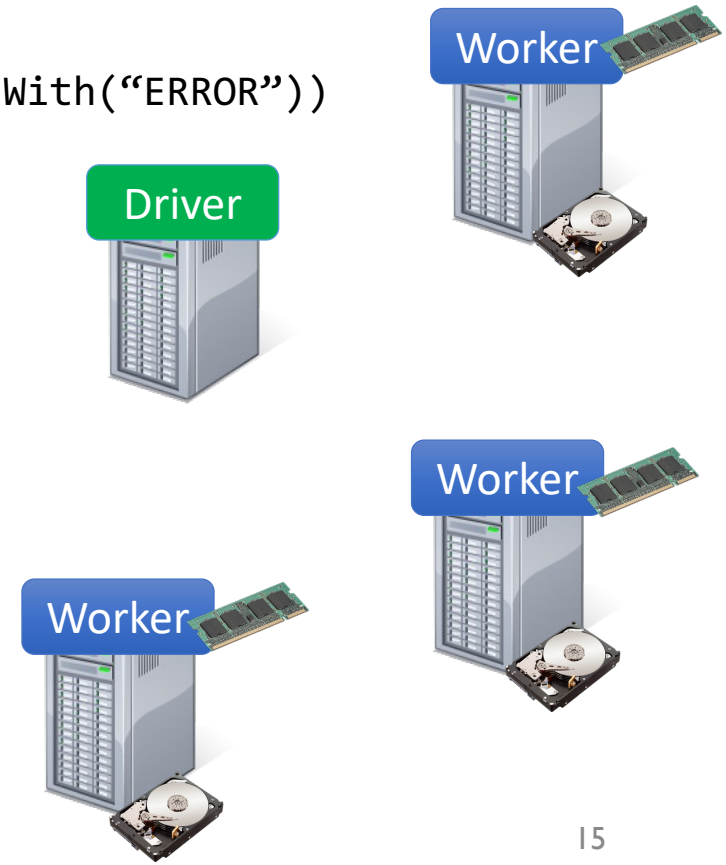
Base RDD



Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

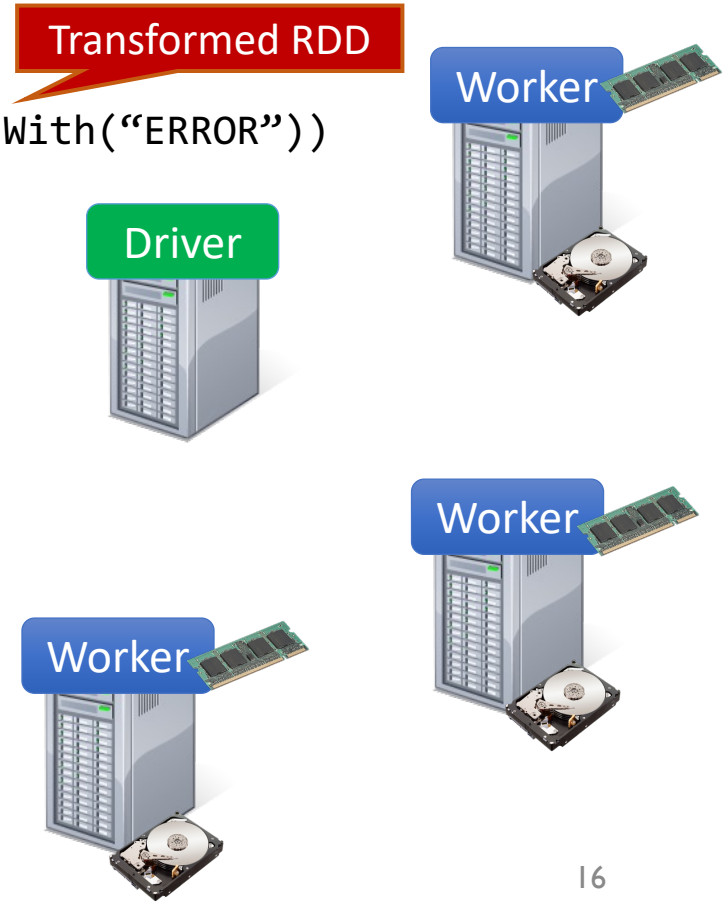
```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
```



Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

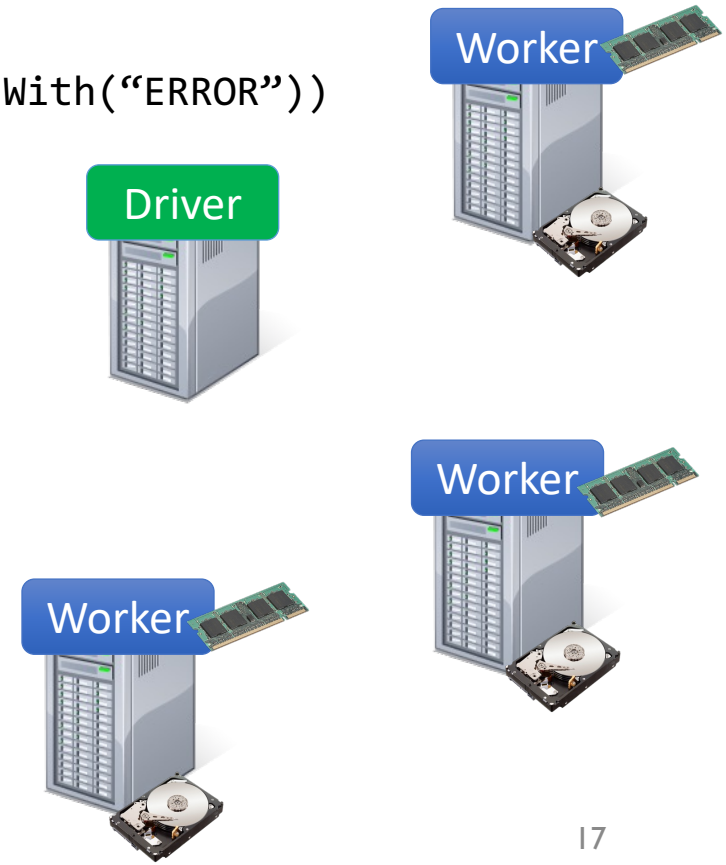
```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
```



Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

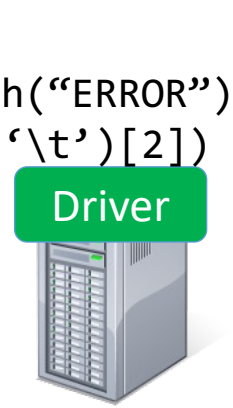
```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
```

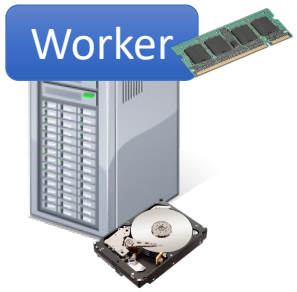


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
```



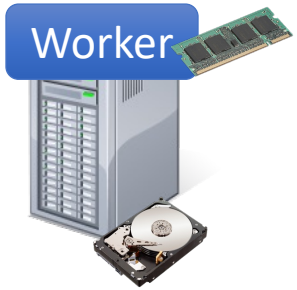


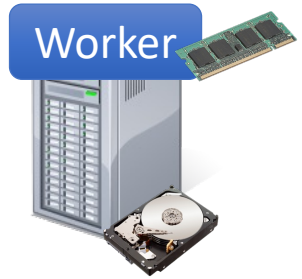
Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
```

Another Transformed RDD

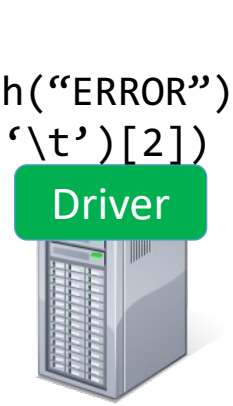


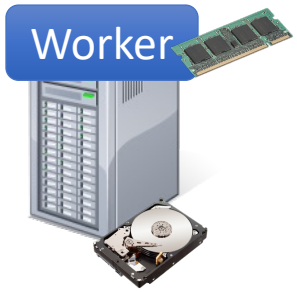


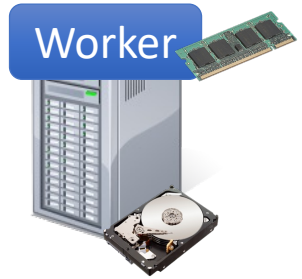
Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```





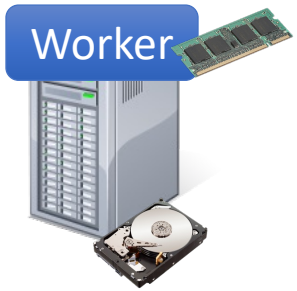


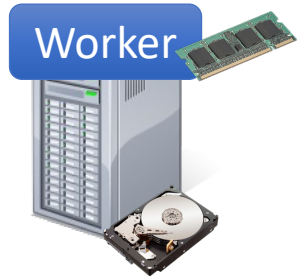
Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

Cache it to memory for reuse



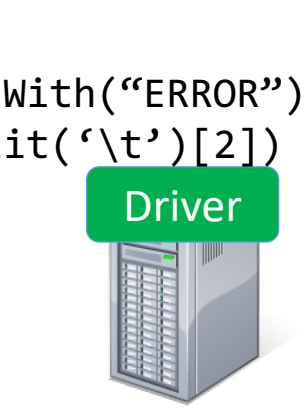


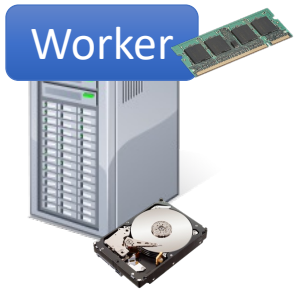
Interactive debugging (control & data flow)

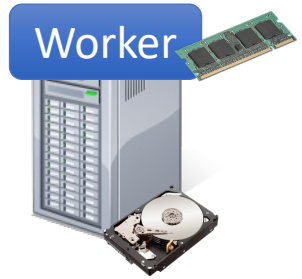
Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                .count()
```







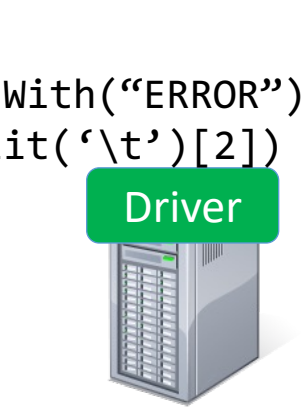
Interactive debugging (control & data flow)

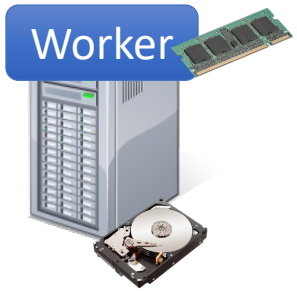
Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                .count()
```

Action





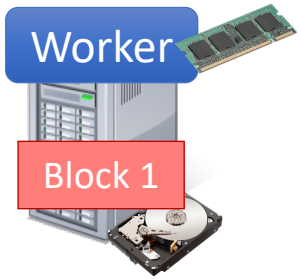
Interactive debugging (control & data flow)

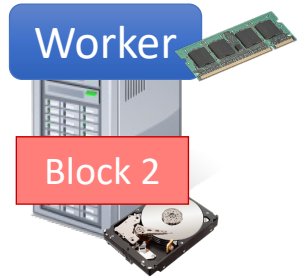
Load input data from an HDFS file into memory, then interactively search for various patterns

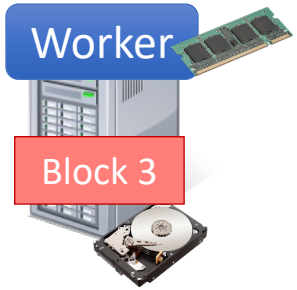
```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                .count()
```

Action





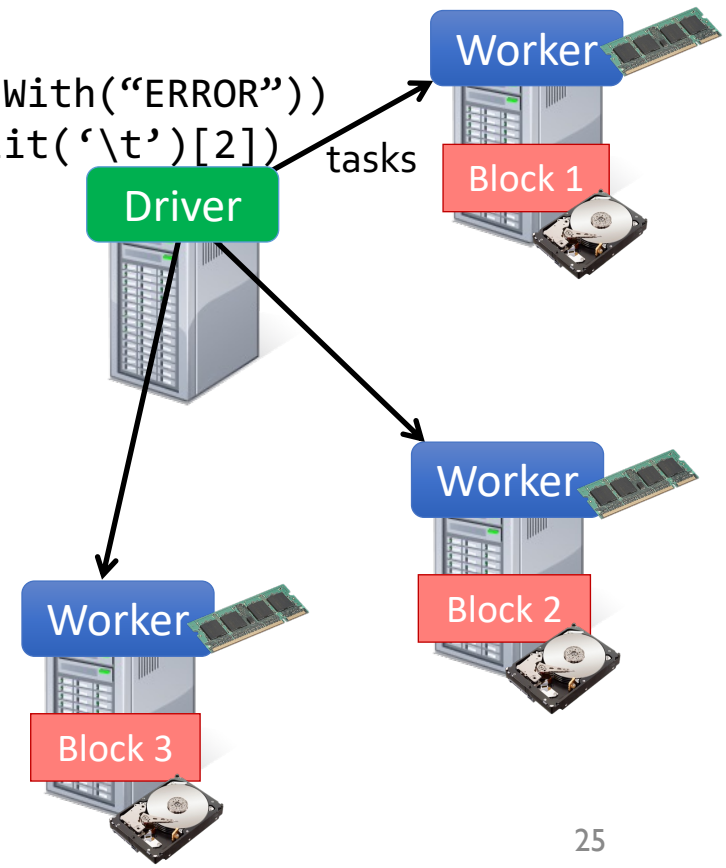


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()


messages.filter(lambda line: “MySQL” in line)
                    .count()
```

Action

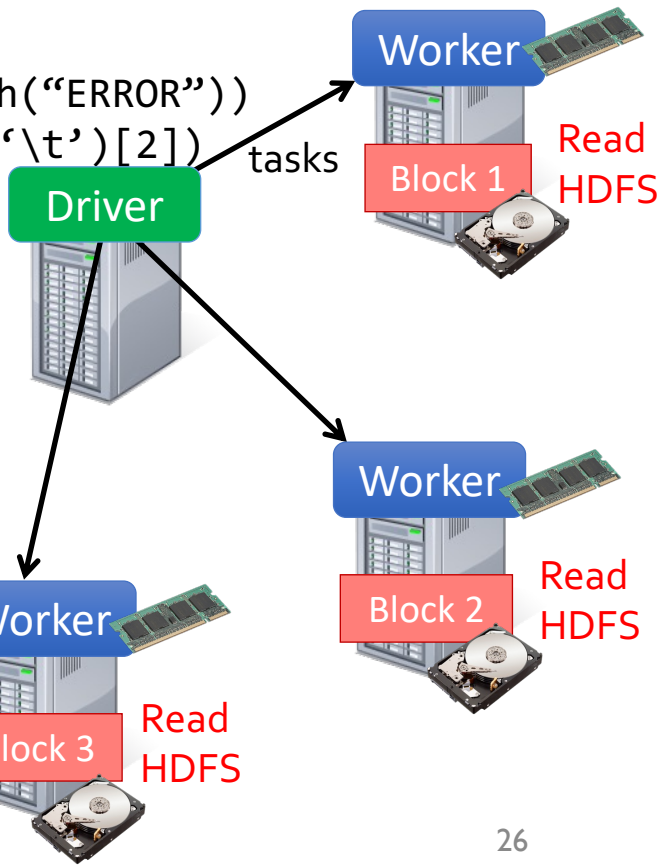


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()


messages.filter(lambda line: “MySQL” in line)
                    .count()
```

Action

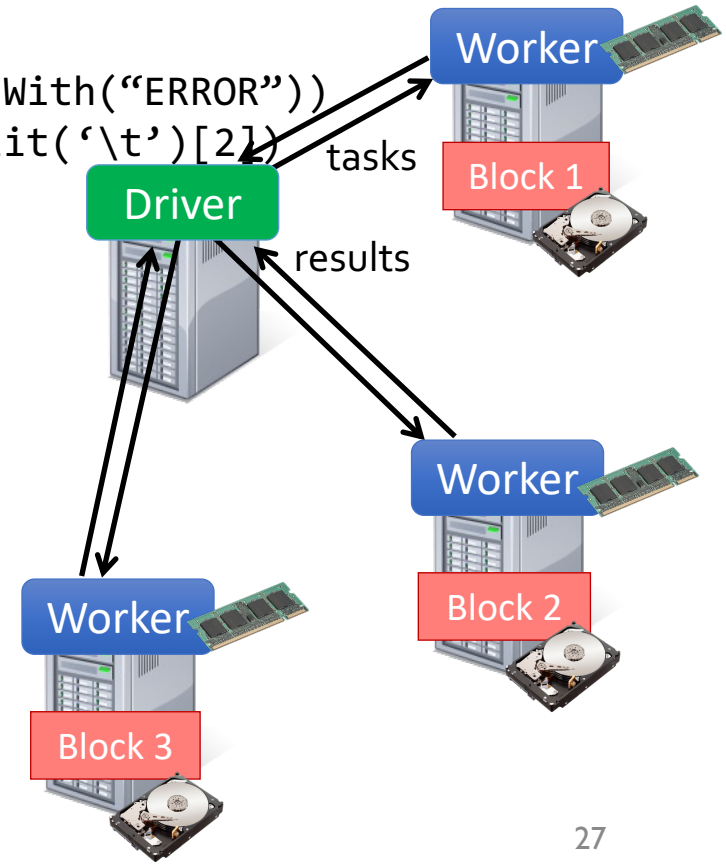


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()


messages.filter(lambda line: “MySQL” in line)
                    .count()
```

Action

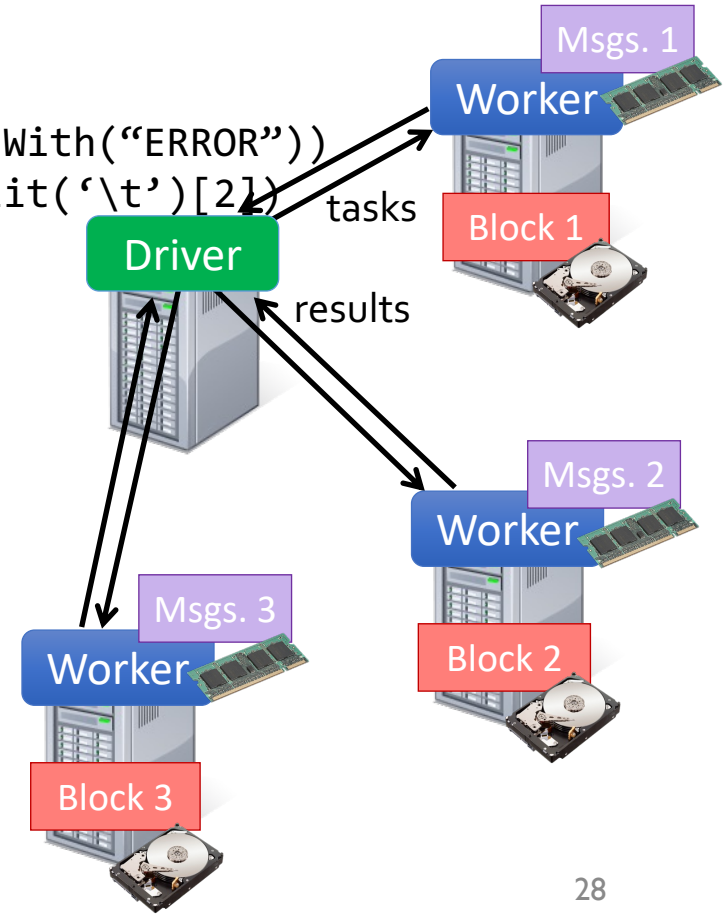


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()


messages.filter(lambda line: “MySQL” in line)
                    .count()
```

Action

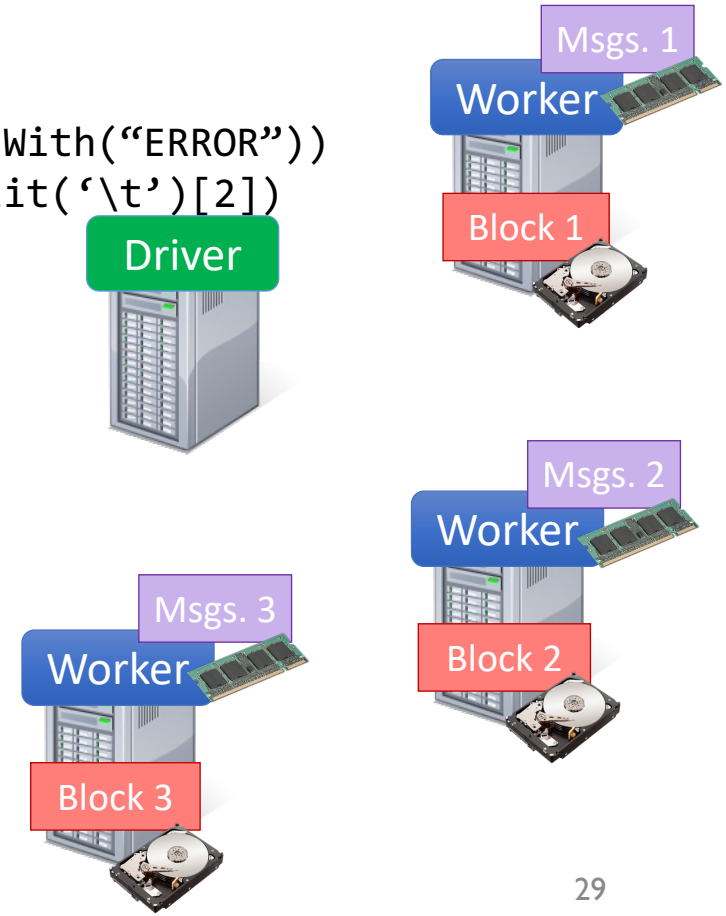


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                    .count()
```

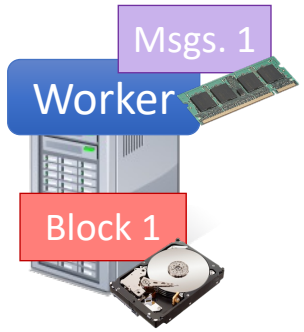


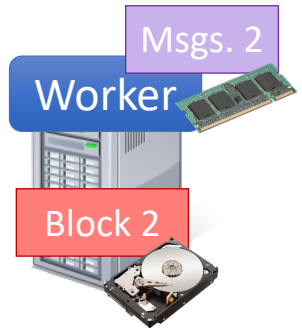
Interactive debugging (control & data flow)

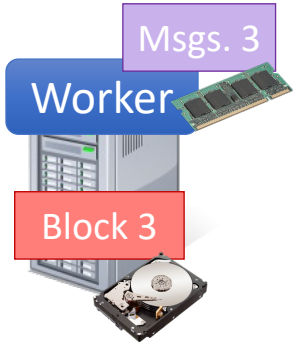
Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                    .count()
messages.filter(lambda line: “HDFS” in line)
                    .count()
```





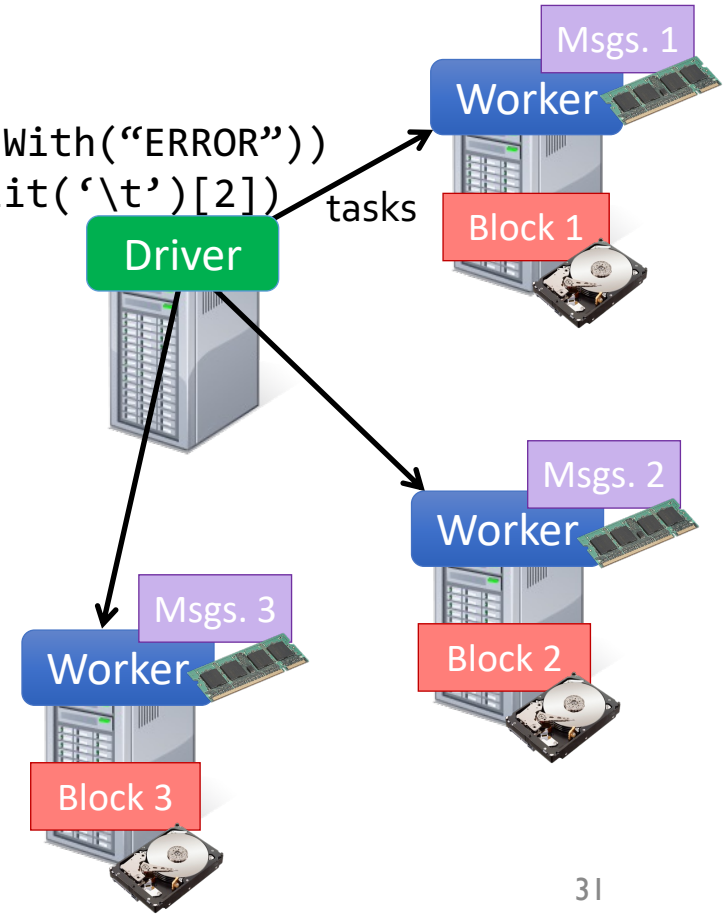


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                    .count()
messages.filter(lambda line: “HDFS” in line)
                    .count()
```

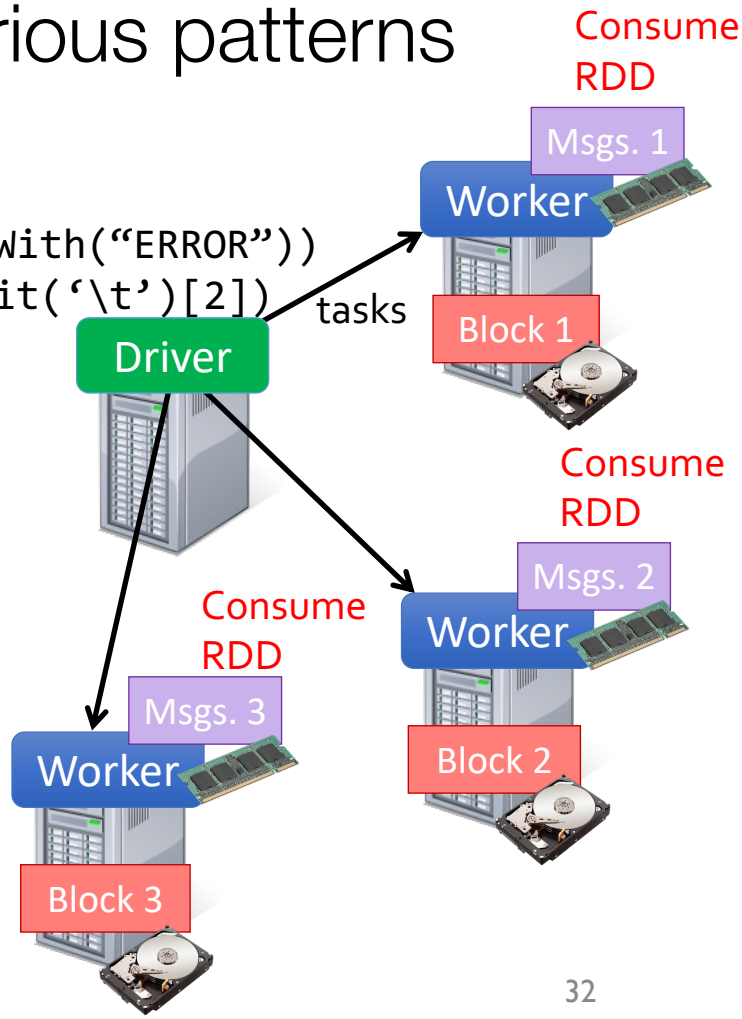


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                    .count()
messages.filter(lambda line: “HDFS” in line)
                    .count()
```

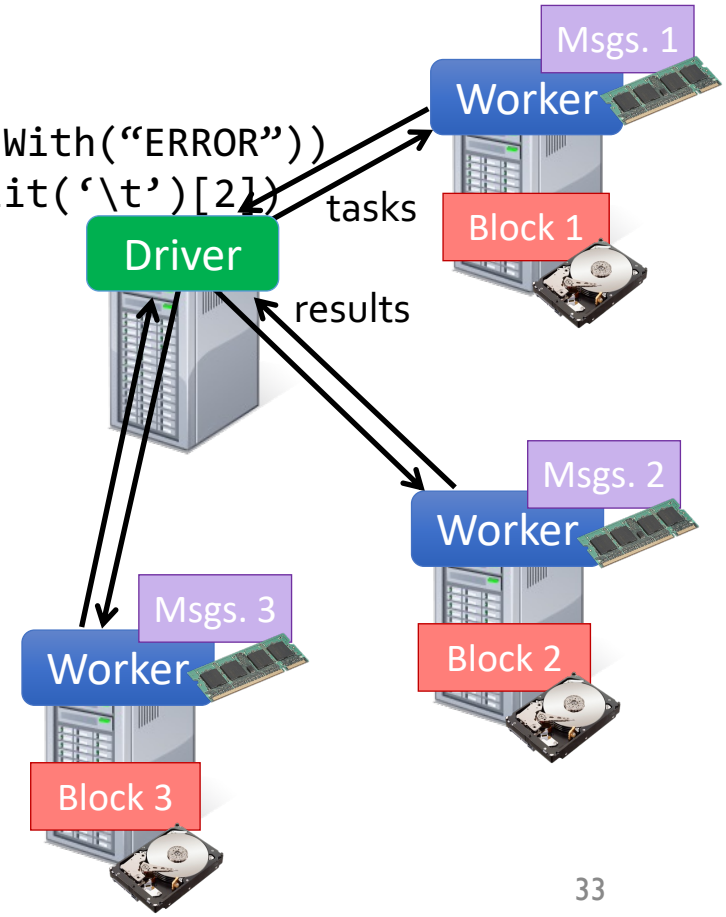


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                    .count()
messages.filter(lambda line: “HDFS” in line)
                    .count()
```

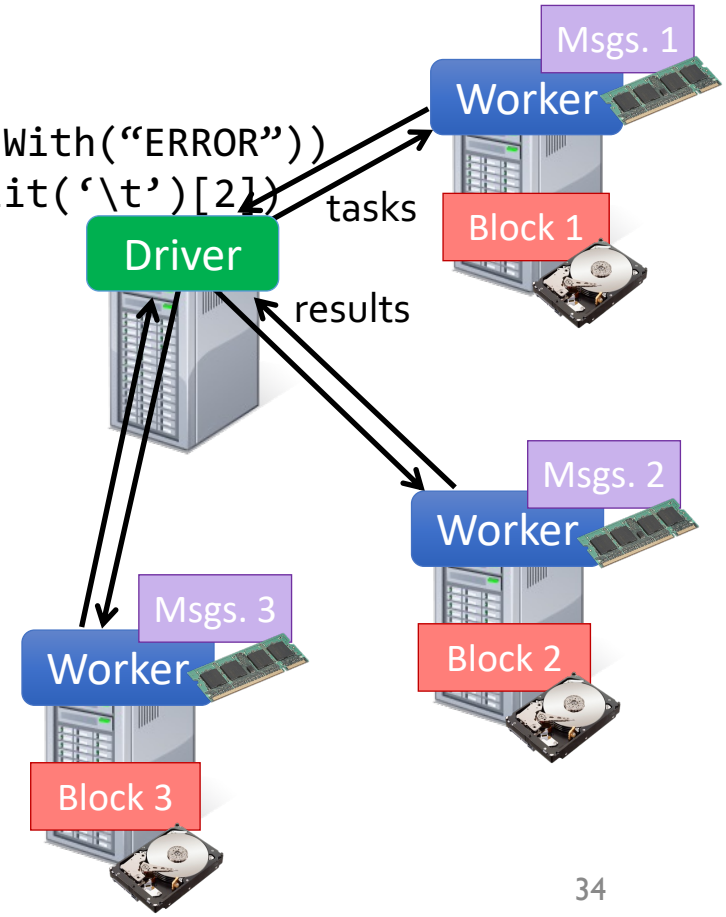


Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                .count()
messages.filter(lambda line: “HDFS” in line)
                .count()
```



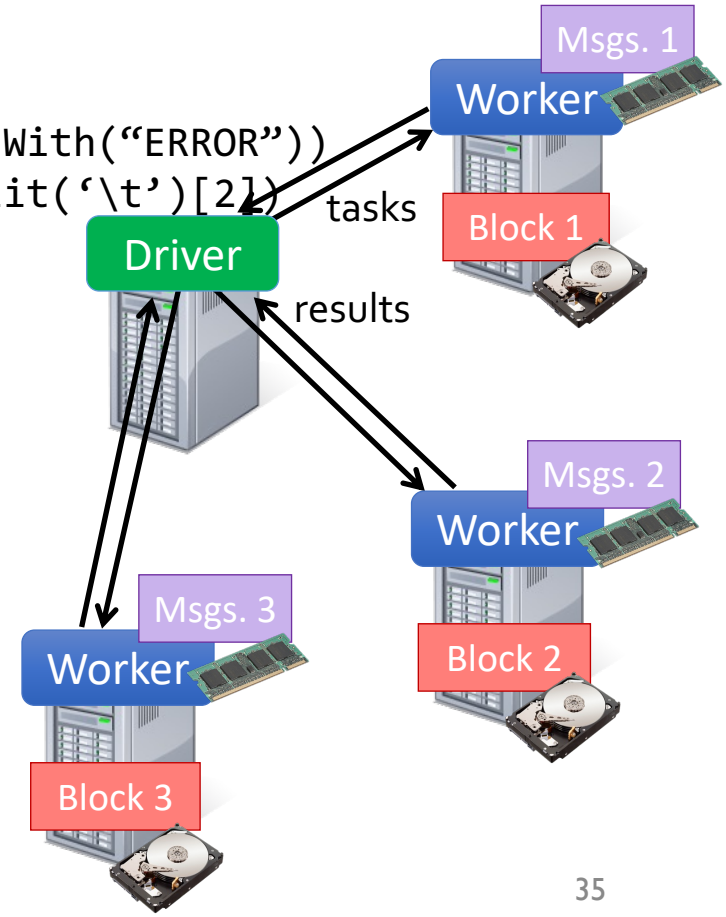
Result: full-text search of Wikipedia in <1 sec (vs. 20 sec for on-disk data)

Interactive debugging (control & data flow)

Load input data from an HDFS file into memory, then interactively search for various patterns

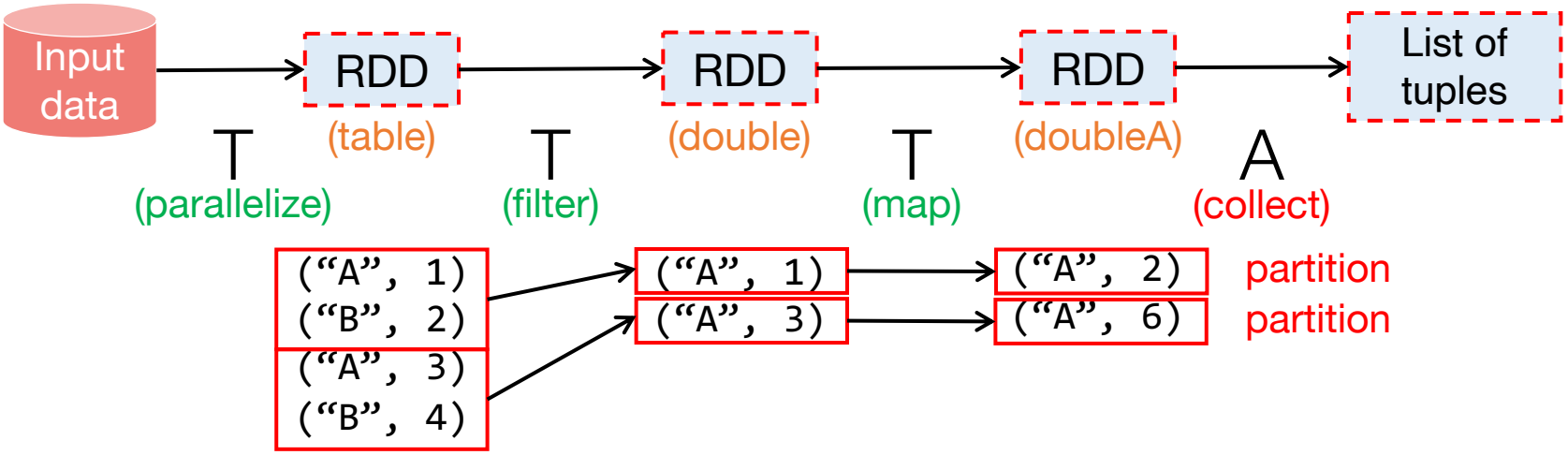
```
lines = sc.textFile(“hdfs://...”)
errors = lines.filter(lambda line: line.startWith(“ERROR”))
messages = errors.map(lambda error: error.split(‘\t’)[2])
messages.cache()
```

```
messages.filter(lambda line: “MySQL” in line)
                .count()
messages.filter(lambda line: “HDFS” in line)
                .count()
```



Result: scaled to 1 TB data in 5-7 sec
(vs. 170 sec for on-disk data)

Repartitioning



Many operations (like `filter` and `map`) output the same number of partitions as they receive

- If data is growing/shrinking a lot after transformation, you might want to change the partition count
- `rdd.getNumPartitions()` `# check how many`
- `rdd2 = rdd.repartition(10)` `# change how many`

Examples:

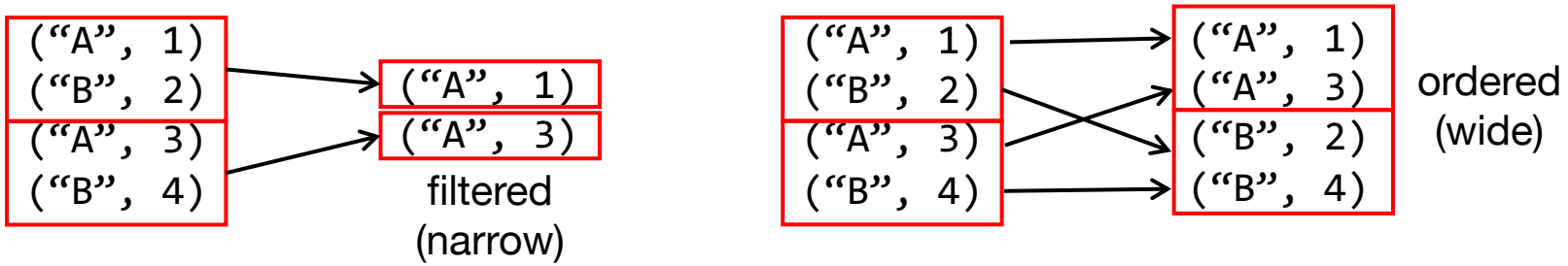
```
table.filter(onlyA).map(mult2).collect()
table.filter(onlyA).repartition(1).map(mult2).collect()
```

Transformations: Narrow vs. Wide

- Any transformation where a single output partition can be computed from a single input partition is a narrow transformation.

- Others are wide transformations.

```
data = [("A", 1), ("B", 2), ("A", 3), ("B", 4),]
table = sc.parallelize(data, 2)
filtered = table.filter(lambda row: row[0] == "A")
ordered = table.sortBy(lambda row: row[0])
```



- Wide transformations often require network resources. Unless all input partitions are on the same machine, some will need to be transferred.

DataFrames: Pandas vs. Spark

```
pandas_df = pd.DataFrame({“x”: [1,2,3]})
```

	x
0	1
1	2
2	3

```
# pandas DFs are mutable
pandas_df[“y”] = pandas_df[“x”] ** 2
```

	x	y
0	1	1
1	2	4
2	3	9

```
spark_df = spark.createDataFrame(pandas_df)

# could convert back:
# spark_df2.toPandas()

# cannot add column to immutable Spark DF
# can only create a new DF RDD
spark_df2 = spark_df.withColumn(“y”, col(“x”) ** 2)
```