APACHE Spark™

SPARK BASICS SIMPLY EXPLAINED

PART 2: SPARKS MAIN ARCHITECTURE CONCEPTS!

Data with Nikk the Greek

- ***Deep dive architecture by example***
- ***Lazy Evaluation and Catalyst Optimizer***
- ***Actions and Transformations***
- ***Jobs, Stages, Tasks***

Data with
Nikk the Greek

APACHE Spark architecture when using filter and write

Transformation

Action

Filter and write

Create dataframe

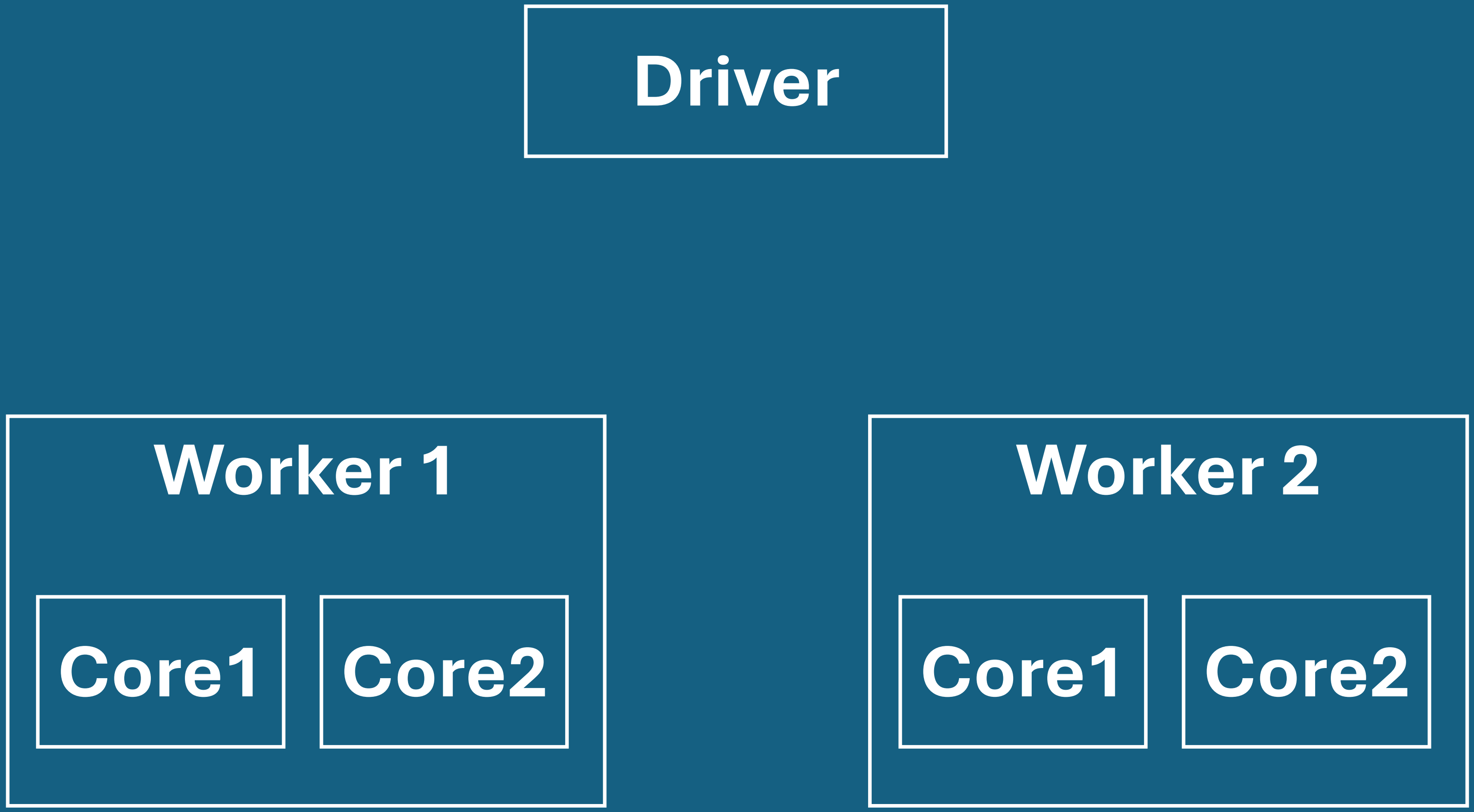
col1	col2
1	A
2	B
3	A
4	B
5	B
6	B
7	A
8	A
9	B
10	B
11	A
12	A

Filter col1 < 10

col1	col2
1	A
2	B
3	A
4	B
5	B
6	B
7	A
8	A
9	B

Write data

Filter and write



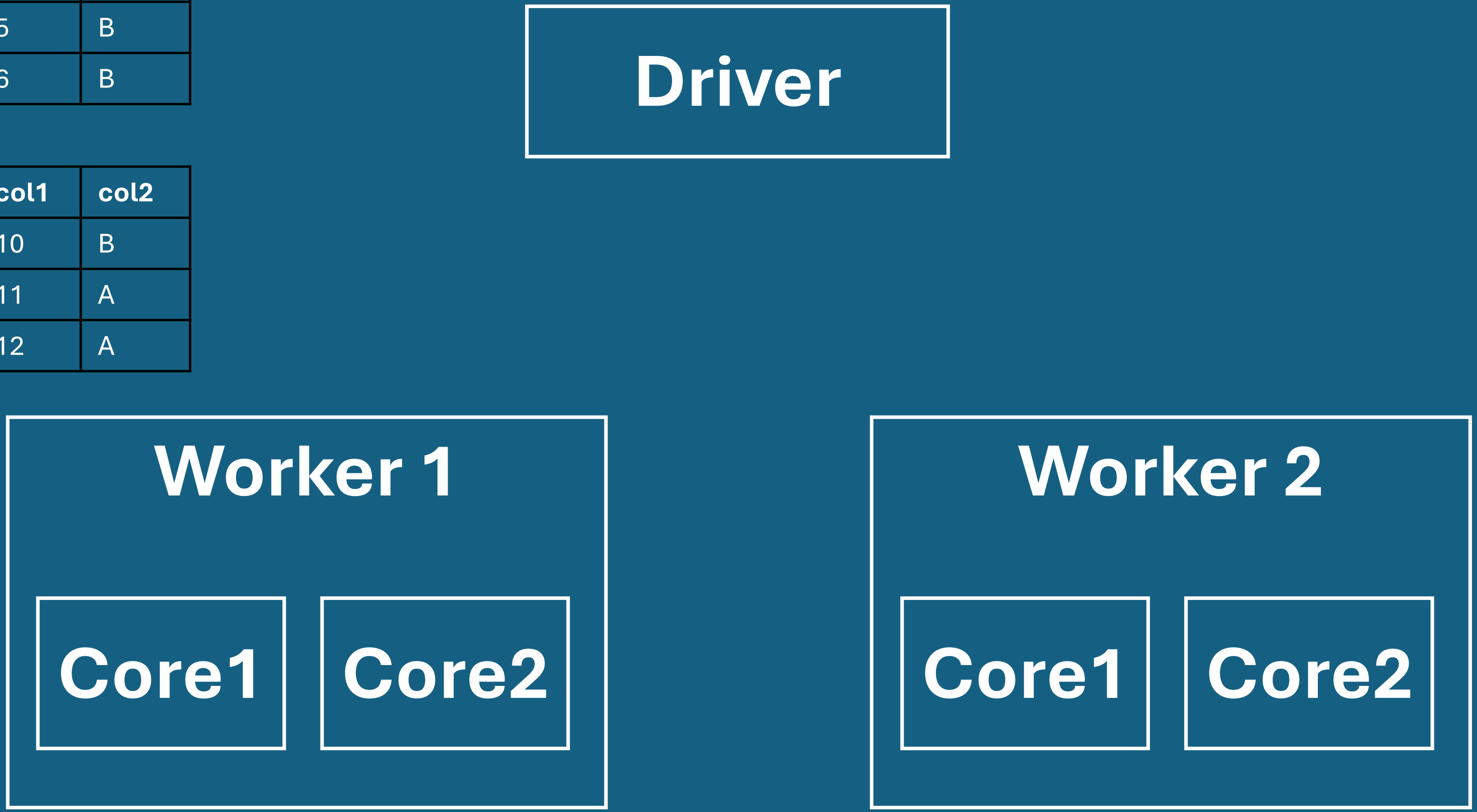
Filter and write

col1	col2
1	A
2	B
3	A

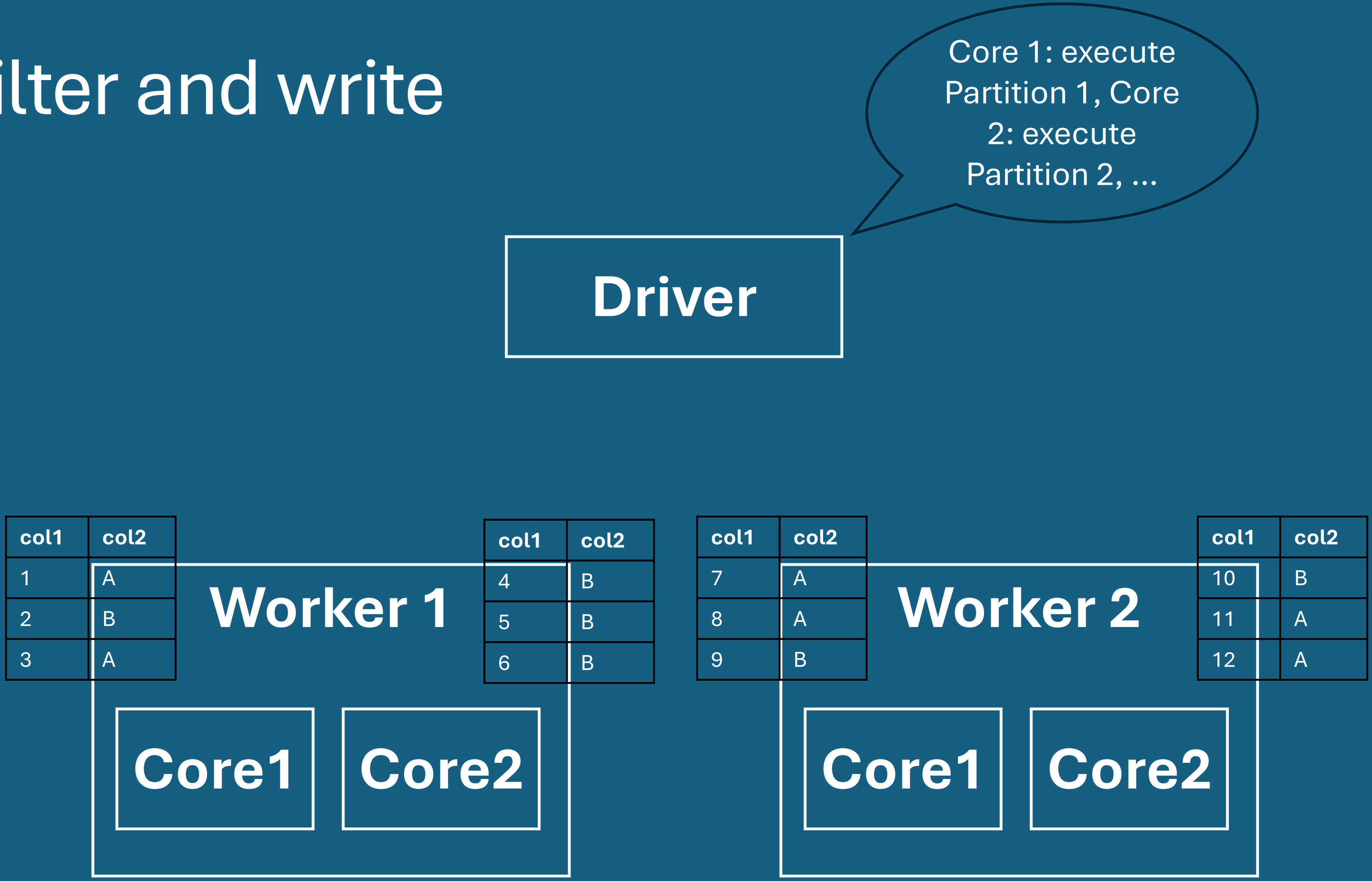
col1	col2
4	B
5	B
6	B

col1	col2
7	A
8	A
9	B

col1	col2
10	B
11	A
12	A



Filter and write



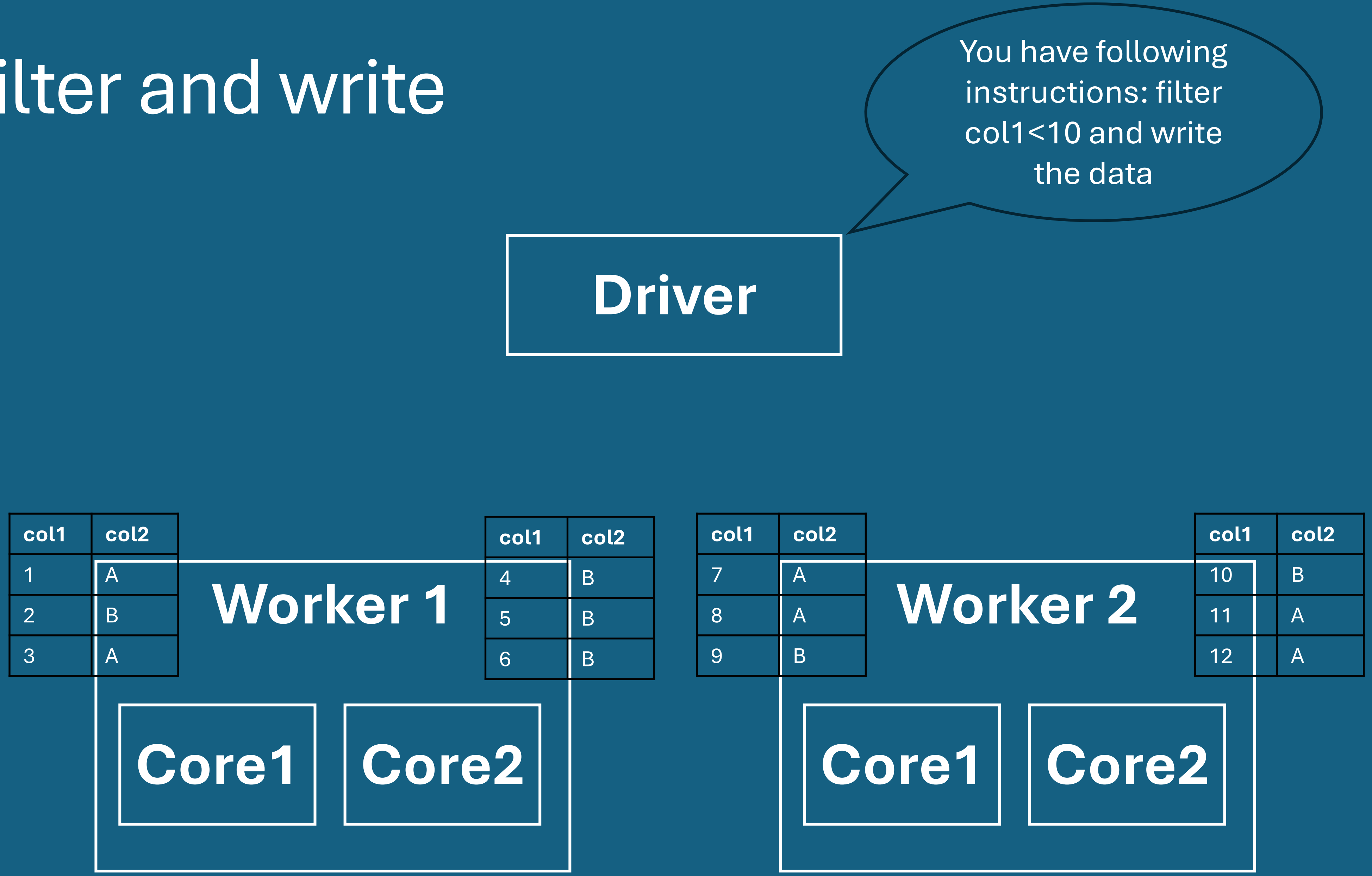
col1	col2
1	A
2	B
3	A

col1	col2
4	B
5	B
6	B

col1	col2
7	A
8	A
9	B

col1	col2
10	B
11	A
12	A

Filter and write

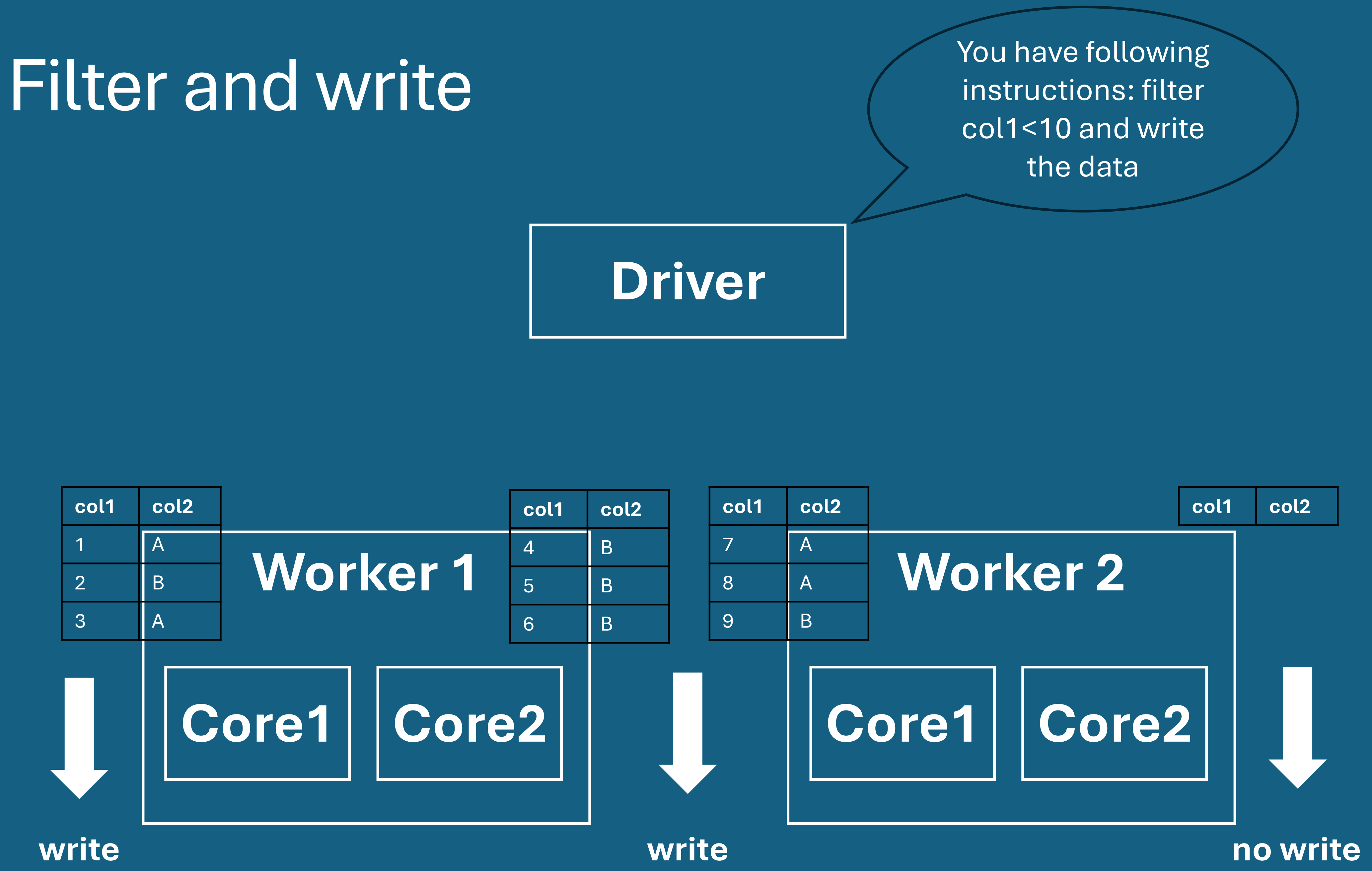


col1	col2
1	A
2	B
3	A

col1	col2
4	B
5	B
6	B

col1	col2
7	A
8	A
9	B

col1	col2
10	B
11	A
12	A


Filter and write
You have following instructions: filter col1<10 and write the data
Driver
Worker 1
Core1
Core2
Worker 2
Core1
Core2
col1 col2
1 A
2 B
3 A
col1 col2
4 B
5 B
6 B
col1 col2
7 A
8 A
9 B
col1 col2
write
write
no write

Take away

- Spark distributes data based on their partitioning
- The driver coordinates what the Executors/Workers are doing
- The cores of each worke process one partition. This is called a task
- One executor share the same disk, memory, network

Spark architecture when using count

Action

Count

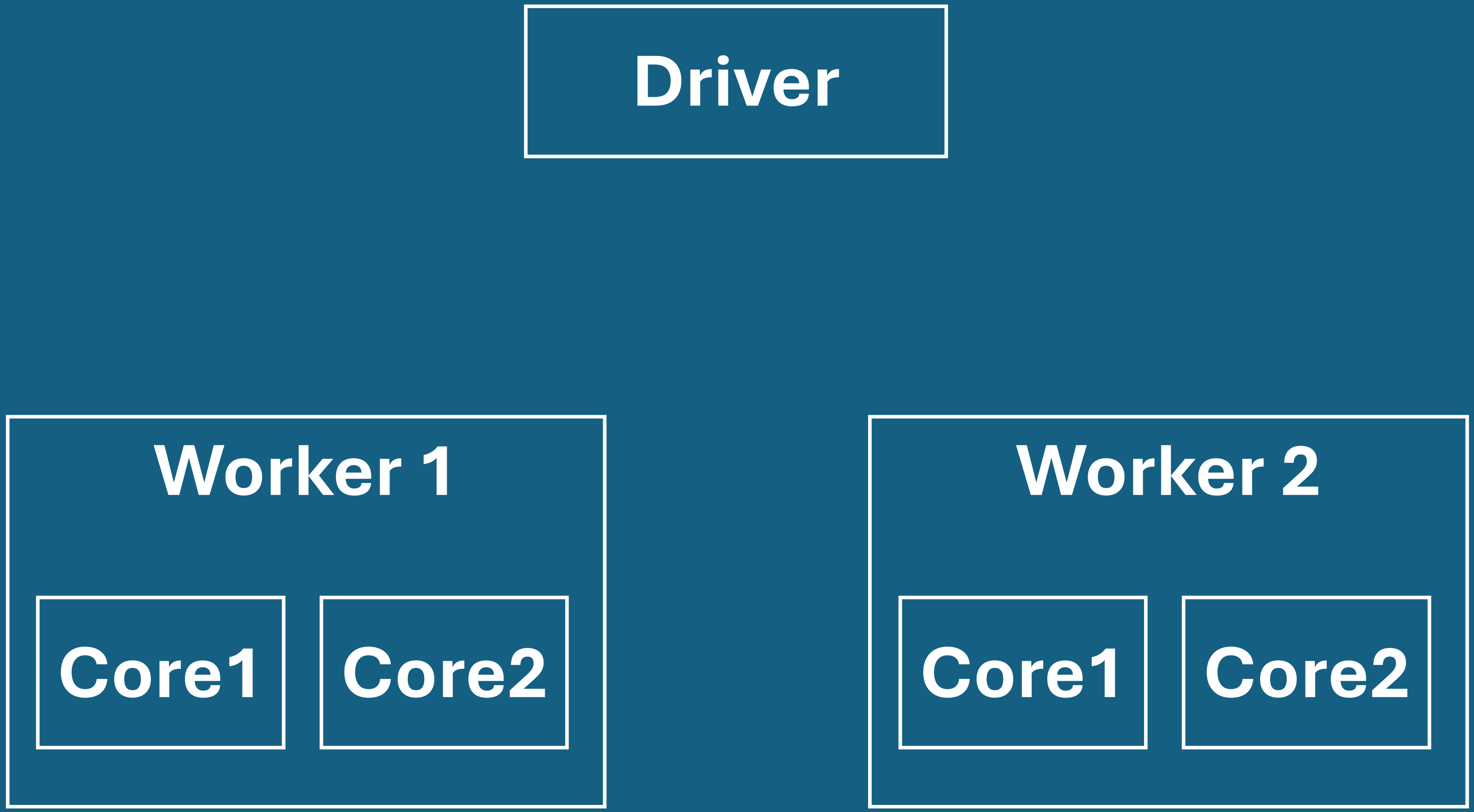
Create dataframe

col1	col2
1	A
2	B
3	A
4	B
5	B
6	B
7	A
8	A
9	B
10	B
11	A
12	A

Count rows

12

Count



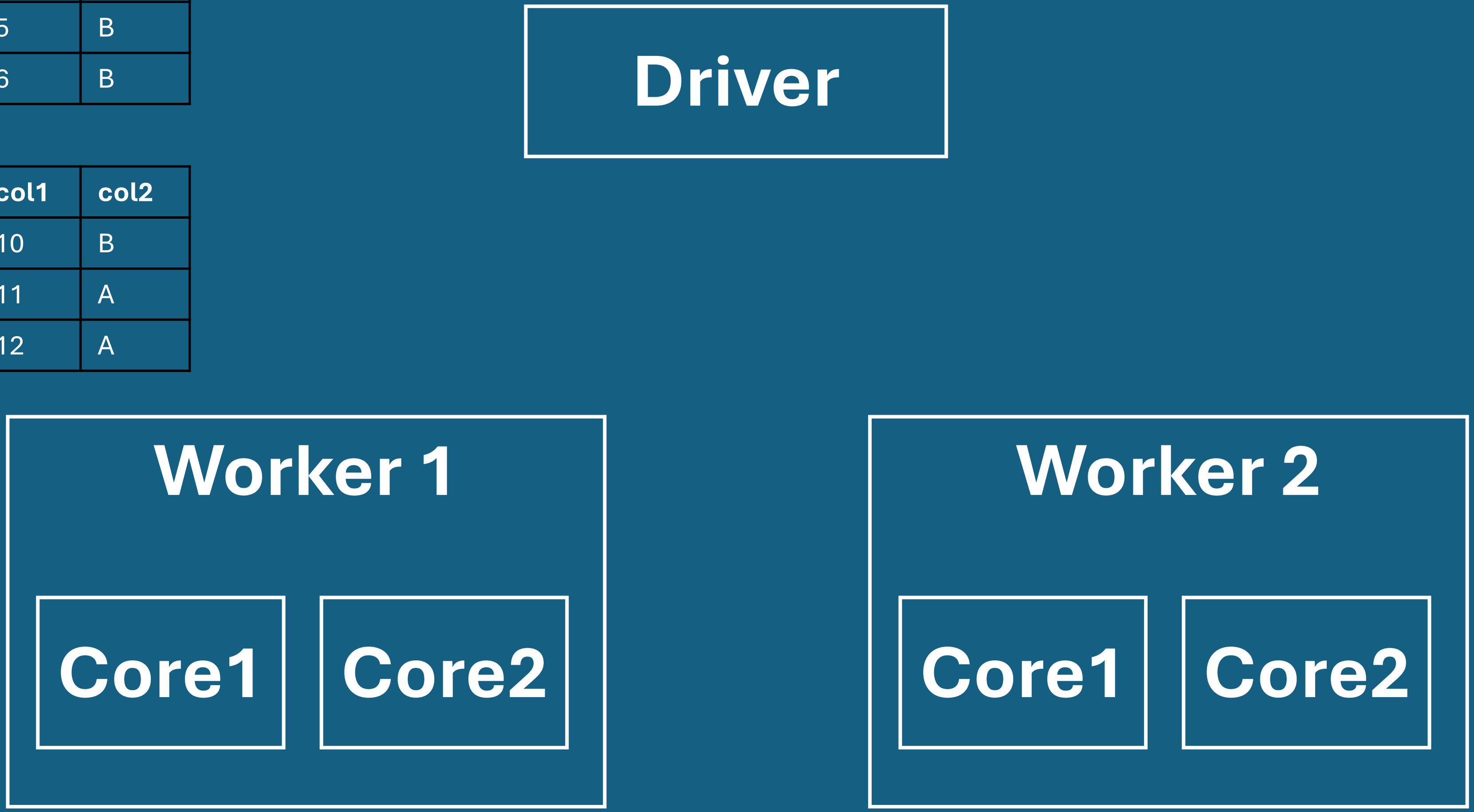
Count

col1	col2
1	A
2	B
3	A

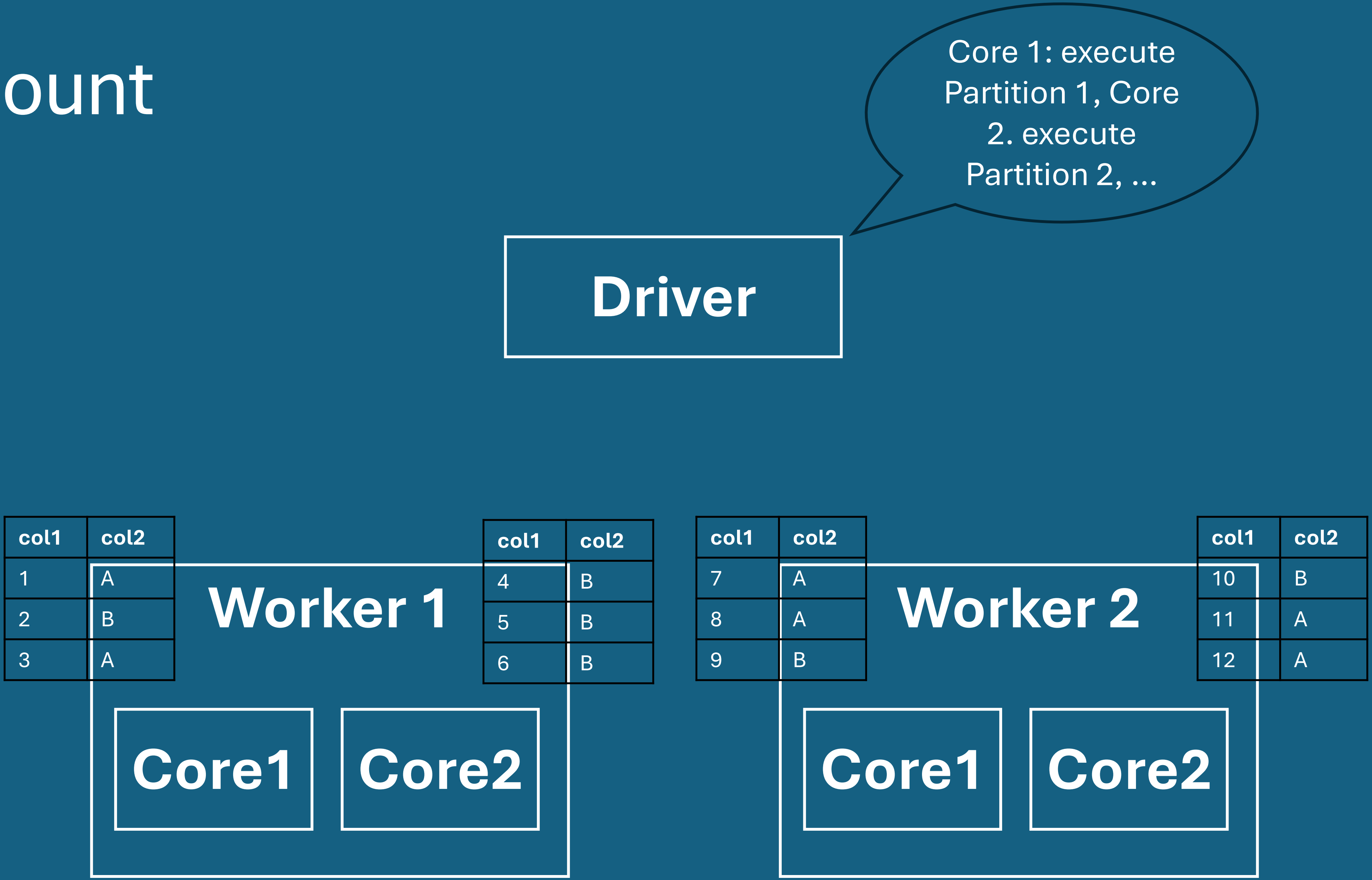
col1	col2
4	B
5	B
6	B

col1	col2
7	A
8	A
9	B

col1	col2
10	B
11	A
12	A

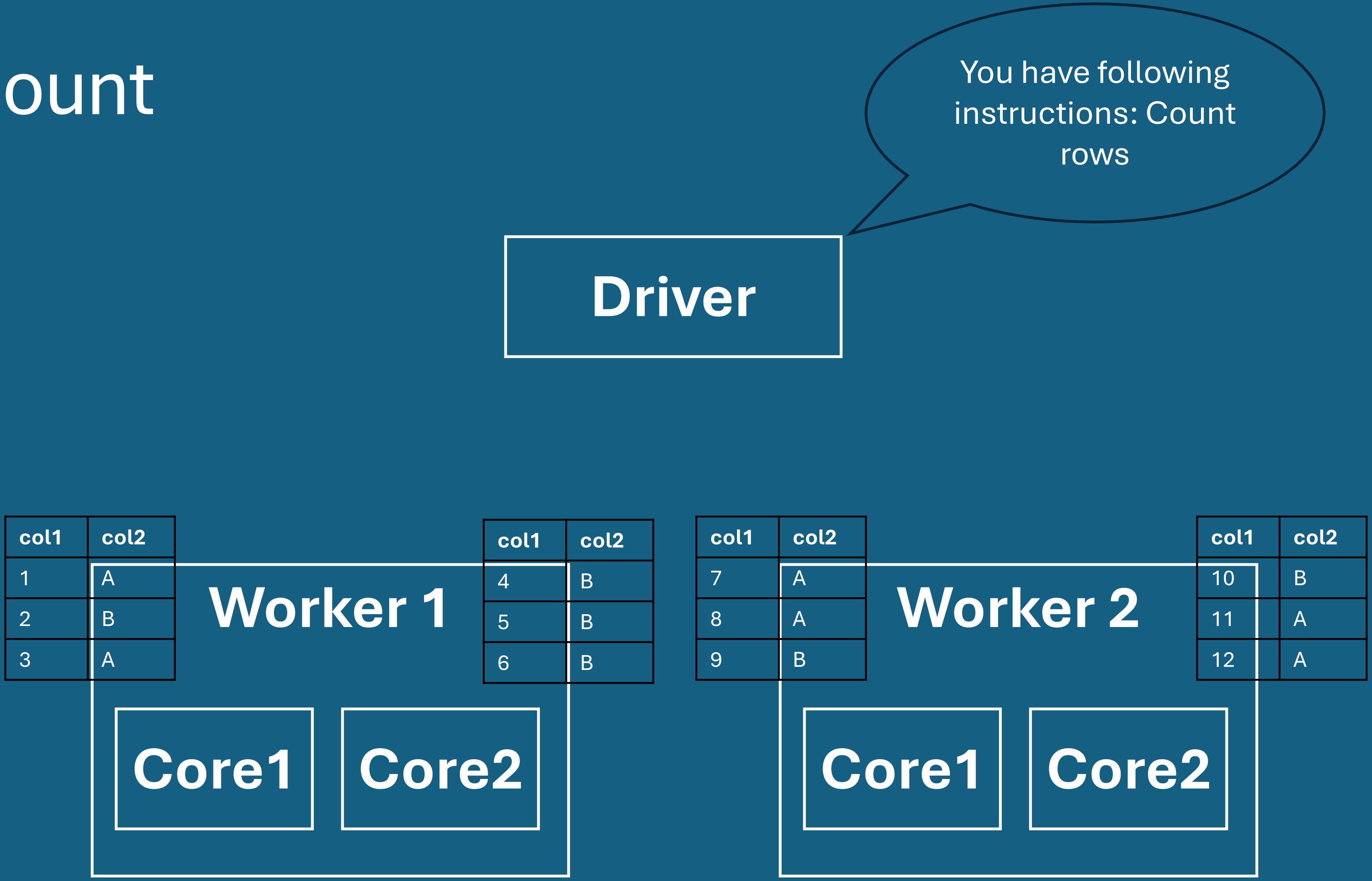


Count



Core 1: execute Partition 1, Core 2. execute Partition 2, ...
Driver
col1 col2
1 A
2 B
3 A
Worker 1
col1 col2
4 B
5 B
6 B
Core1
Core2
col1 col2
7 A
8 A
9 B
Worker 2
col1 col2
10 B
11 A
12 A
Core1
Core2

Count



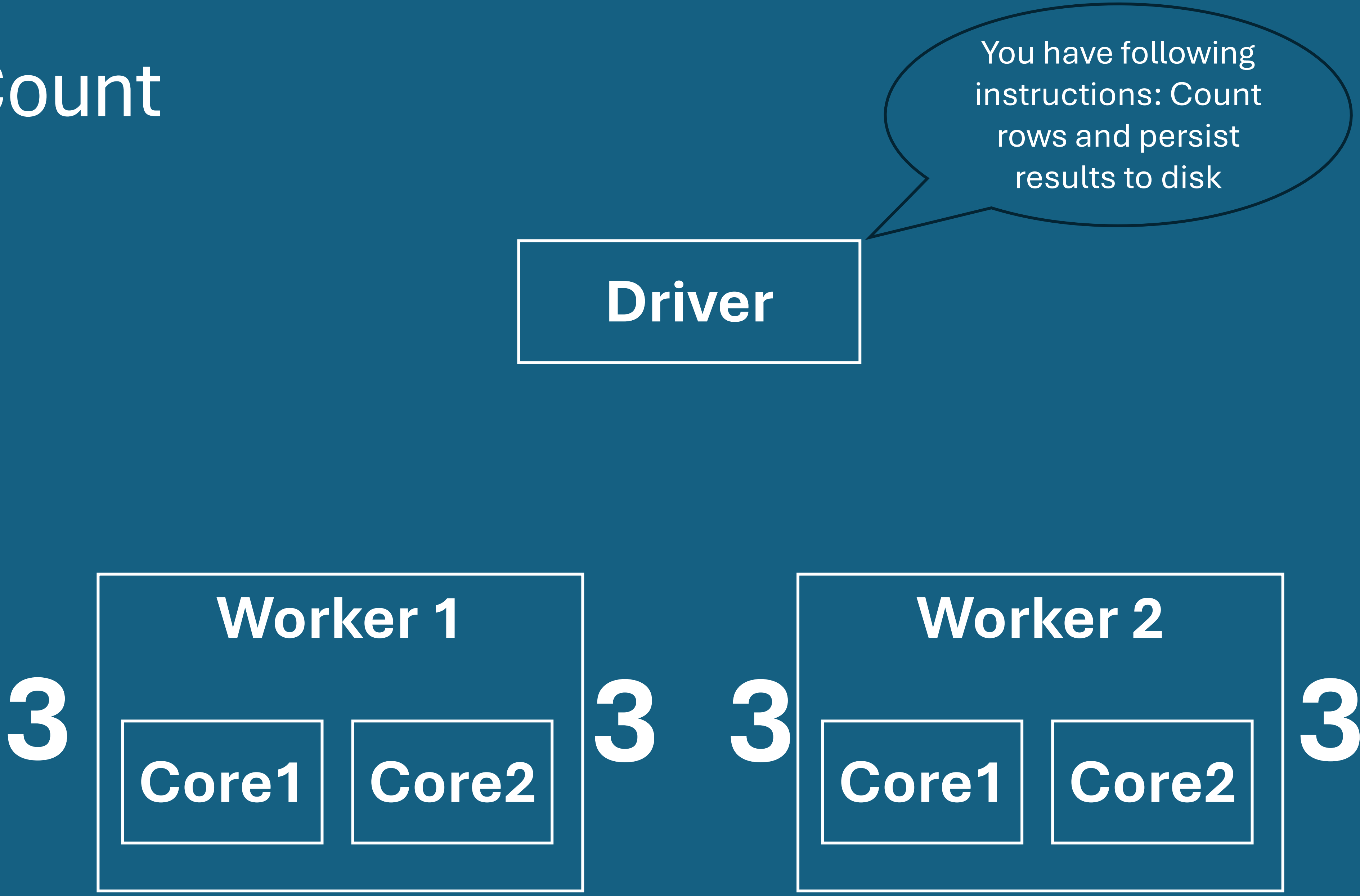
col1	col2
1	A
2	B
3	A

col1	col2
4	B
5	B
6	B

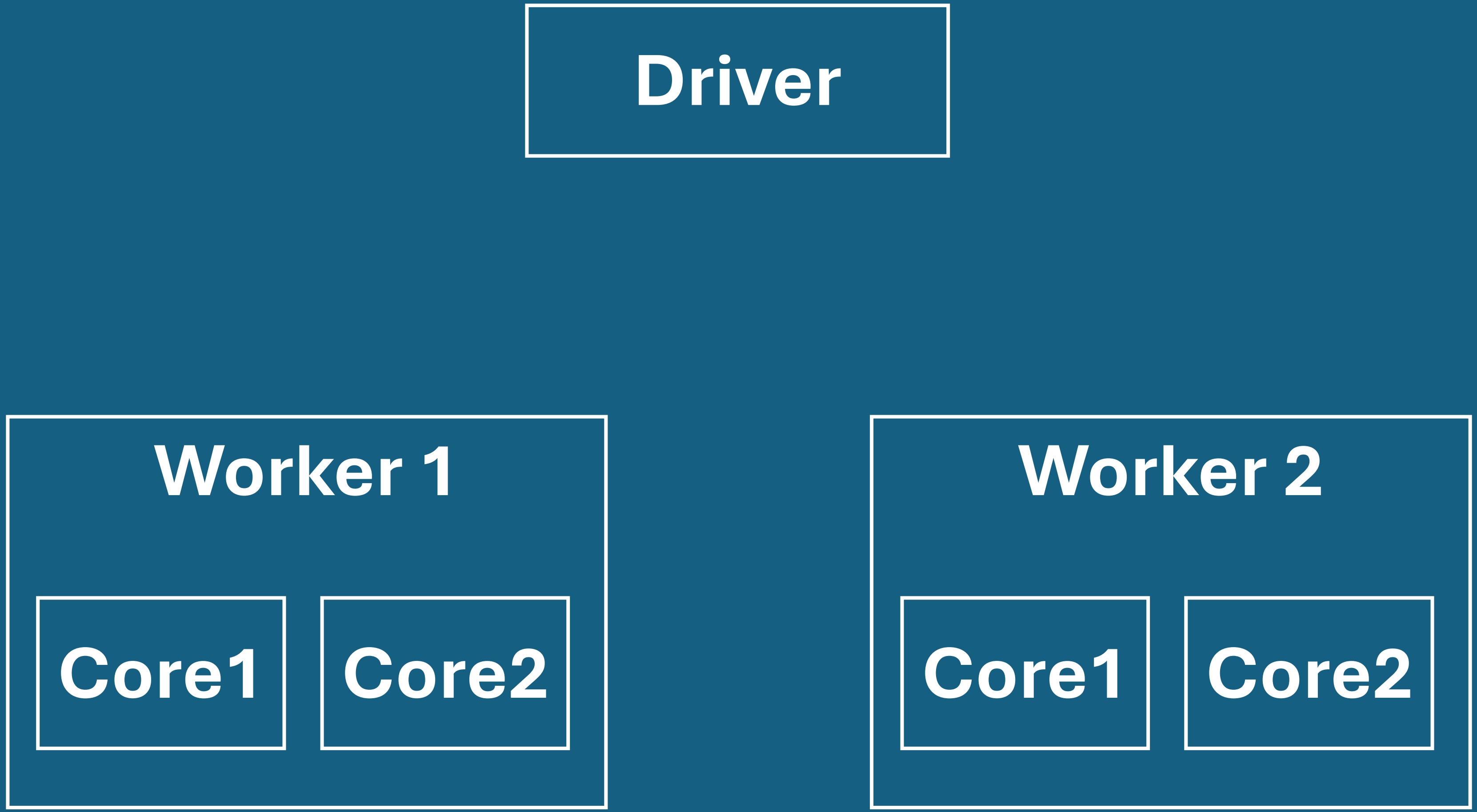
col1	col2
7	A
8	A
9	B

col1	col2
10	B
11	A
12	A

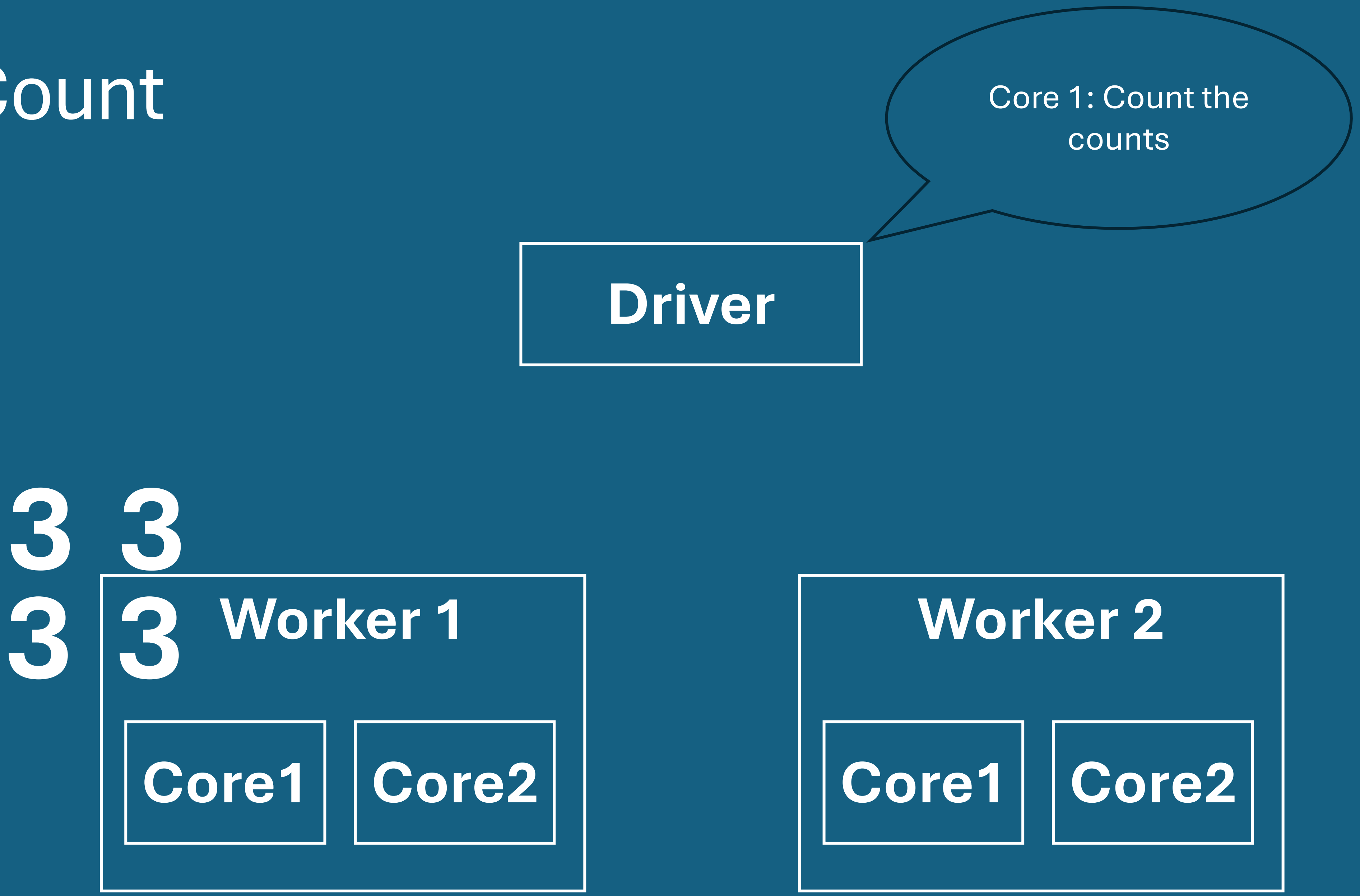
Count


You have following instructions: Count rows and persist results to disk
Driver
Worker 1
Worker 2
3
Core1
Core2
3
3
Core1
Core2
3

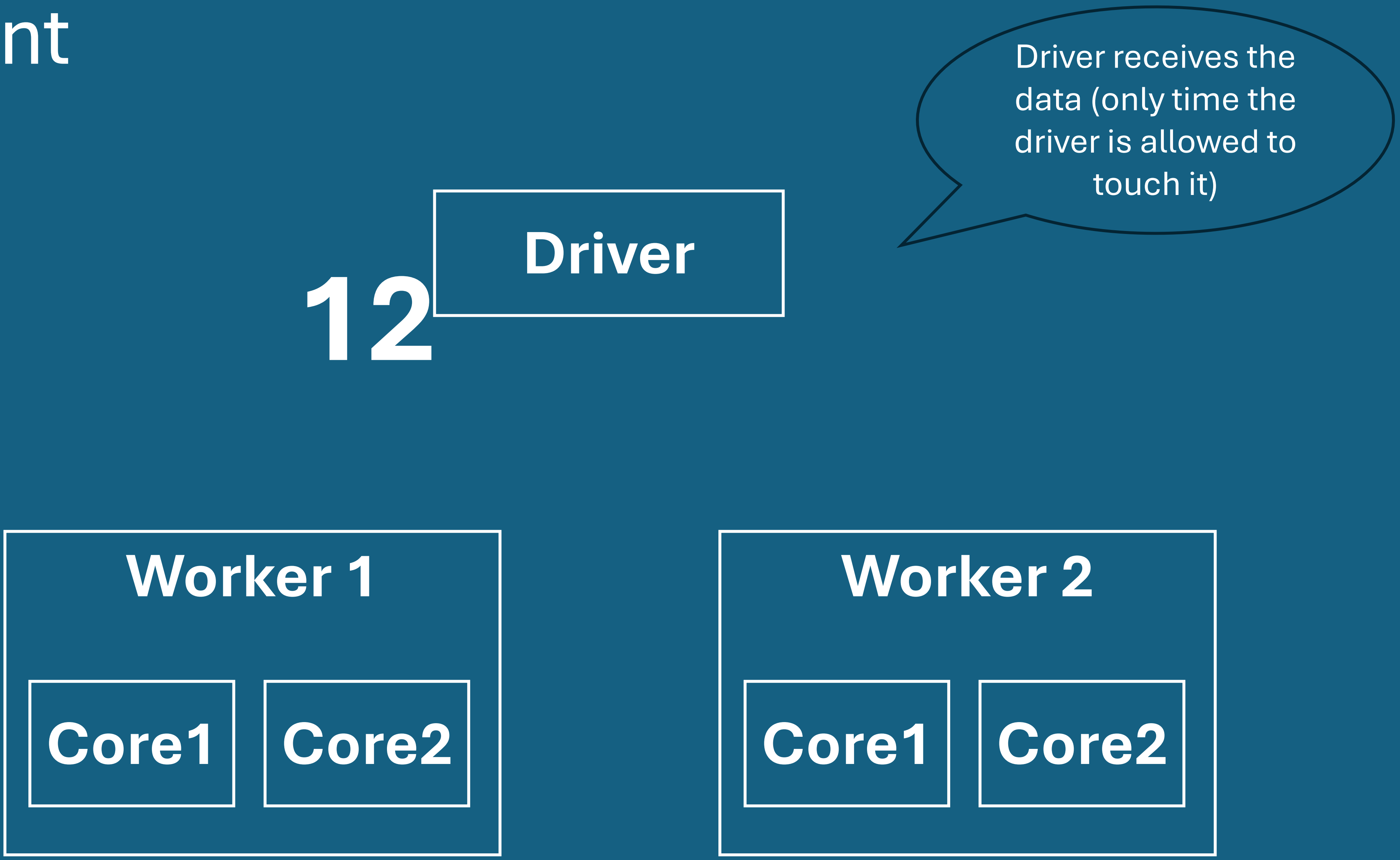
Count



Count


Core 1: Count the counts
Driver
3 3
3 3
Worker 1
Core1
Core2
Worker 2
Core1
Core2

Count


Driver receives the data (only time the driver is allowed to touch it)
Driver
12
Worker 1
Core1
Core2
Worker 2
Core1
Core2

Take away

- Spark creates two rounds of executions
- First we have a local count where results are saved on disk
- Then we have a second stage performing the final count
- The driver is never allowed to touch the data except of when taking the result

Lazy evaluation and catalyst optimizer

Python vs. PySpark – Lazy Evaluation

```
# Load parquet data from a defined path
pdf = pd.read_parquet(path)

# Filter data
pdf = pdf[pdf["id"] > 10000]

# Count
pdf.shape[0]
```

- Python loads all data in the first line into Memory
- After this the filtering and count is performedc
- For small datasets Python might be faster

```
# Load parquet data from a defined path
sdf = spark.read.parquet(path)

# Filter data
sdf = sdf.filter(f.col("id") > 10000)

# Count
sdf.count()
```

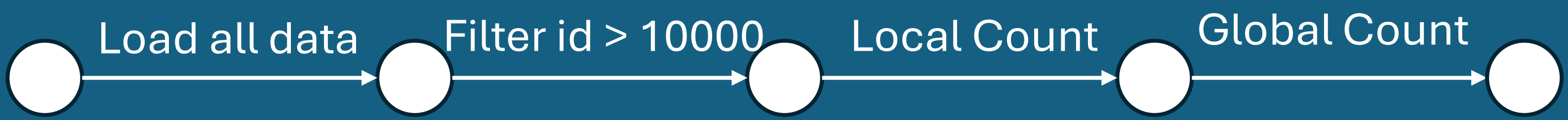
Transformation

Action

- Spark is loading, filtering and counting data „lazily“ only with the count (called action)
- Based on all identfied steps the „Catalyst Optimizer“ finds the most efficient execution plan

DAG – Directed Acyclic Graph

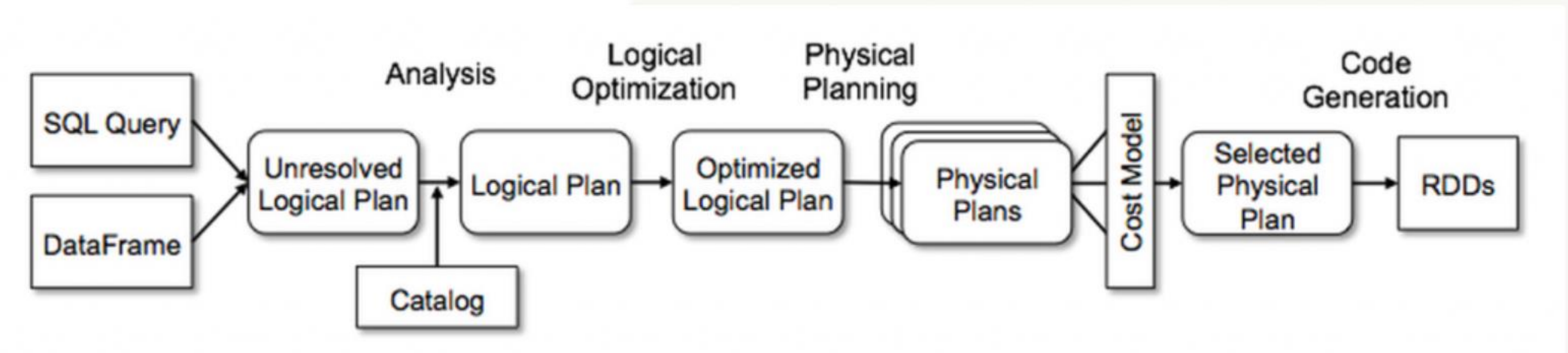
- Created once an action ocures and represents operations to be performed
- Every edge represents an operation of an RDD – resilient distributed dataset which represents the data frame in Spark core
- The DAG is further devided into so called Stages
- The DAG also supports fault tolerance if a Task failes



- The DAG is then used in the Catalyst Optimizer. The DAG just says what has to be done. Spark optimizer the instructions to execute them in the most efficient way.

Catalyst Optimizer

- A rule based engine which rewrites the logical plan into an optimal physical plan.



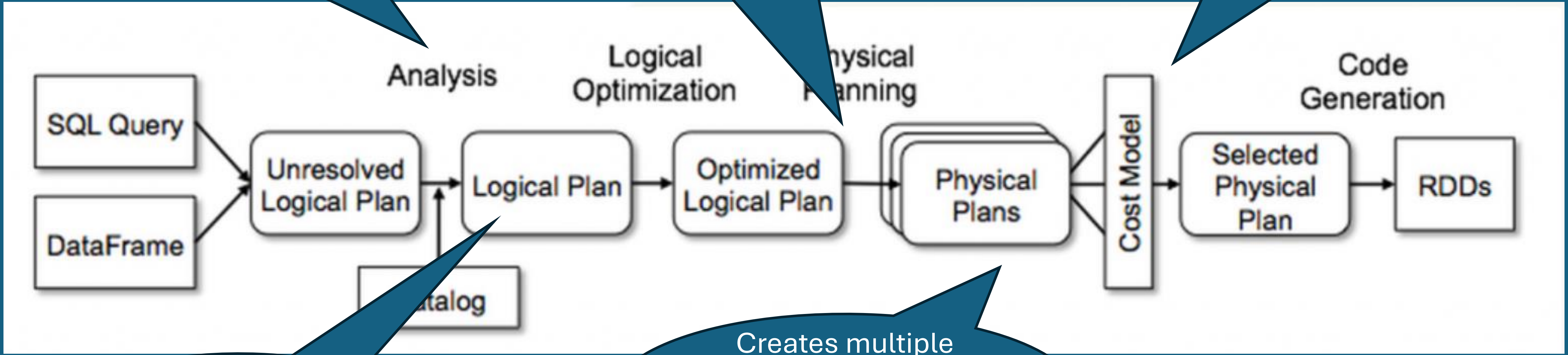
Reference: Deep Dive into Spark SQL's Catalyst Optimizer | Databricks Blog

Catalyst Optimizer

Meta data about columns and tables are received and checked

It basically rewrites rule based your code like pushing filter down or combine filters together

Finally, we choose the best performing plan and generate Java byte code

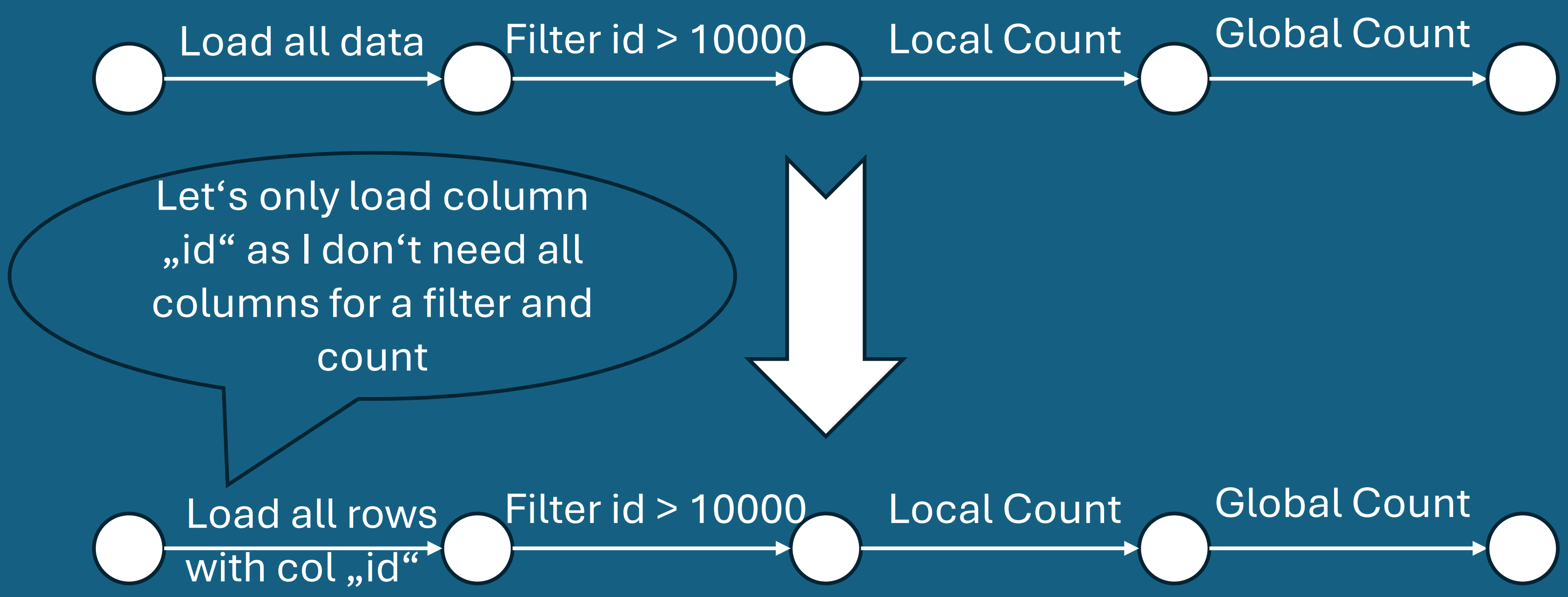


Approves legit tables and columns and might re-order commands order

Creates multiple physical plans that can be used to execute the query

Reference: Deep Dive into Spark SQL's Catalyst Optimizer | Databricks Blog

Catalyst Optimizer



Take away

- Lazy Evaluation in Sparks means Spark will not start the execution of the process until an Action is called. Once an Action is called, Spark starts looking at all the transformations. By that it can optimize the whole execution plan.
- The DAG – Directed Acyclic Graph represents sequentially the planned operations, helps with fault tolerance and is the input for the Catalyst Optimizer.
- The Catalyst Optimizer optimizes your written code, e.g. to reduce data input and perform operations in Memory more efficient.

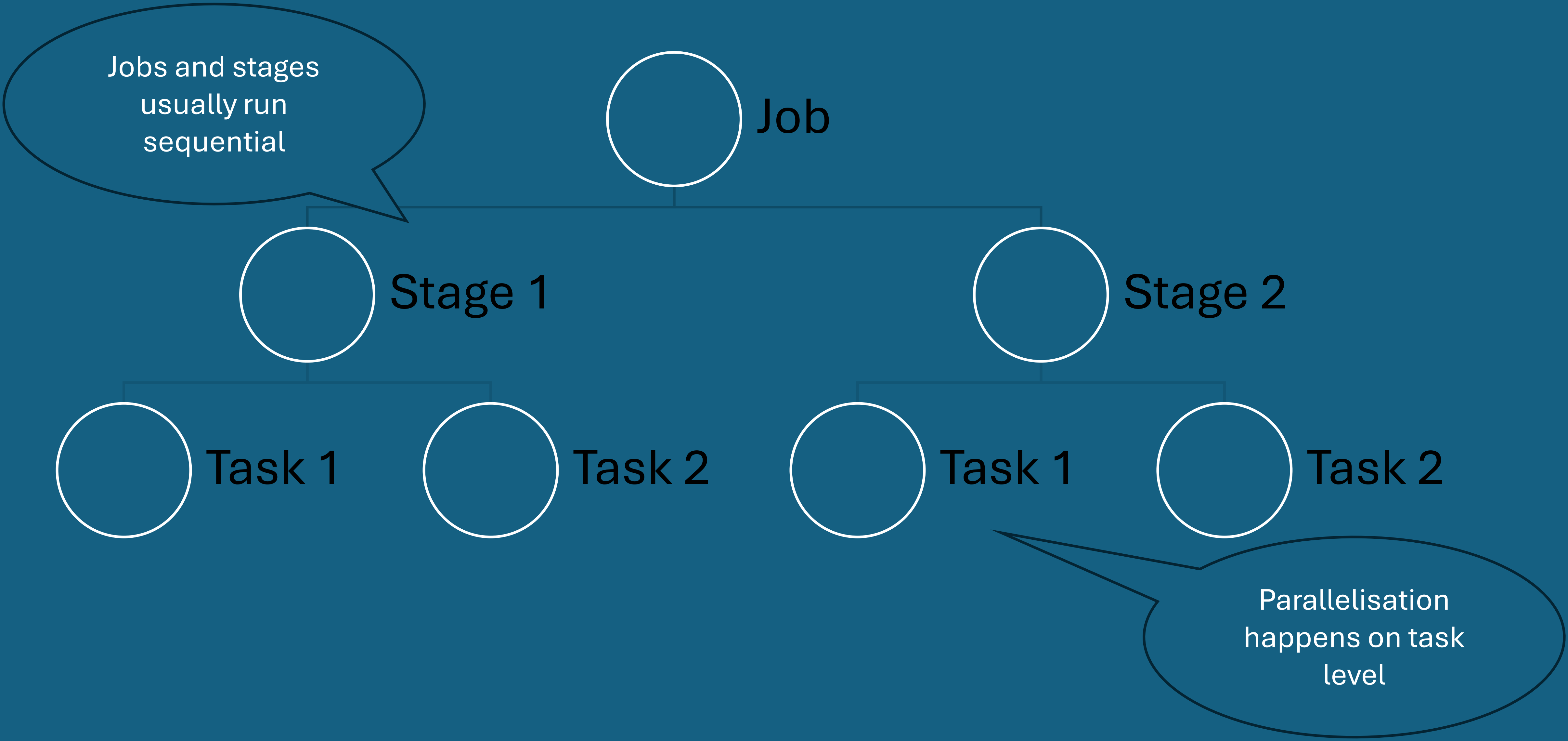
Actions and Transformations

Actions and transformations

- Actions:
  - Actions are methods to access the actual data available in a Dataframe. Action executes all the related transformations to get the required data.
  - Functions such as collect(), show(), count(), first(), take(n) are examples of actions.
- Transformations: Transformations when executed results in a single or multiple new RDD’s.
  - Narrow: Transformations that do not result in data movement between partitions are called Narrow transformations. Examples: select(), union(), filter(), …
  - Wide: Transformations that involves data movement between partitions are called Wide transformations or shuffle transformations. Examples: groupBy(), aggregate(), join(), repartition(), …

Jobs, Stages, Tasks

Jobs, Stages, Tasks



Jobs, stages, tasks

- Jobs: The highes level in the hierachy consisting of stages and tasks which are distributred across the available cores. One or more jobs are initiated by an action.
- Stages: Jobs are devided into multiple stages. Usually this depends on which operations can be performed in serial or in paralle. Our count example is one where we need two stages. Local and global count. Wide transformations also result into multiple stages
- Tasks: Tasks are the lowest level of work. Each tasks is federated across a core within a worker. Each task executes only one partition. That‘s where the parallisation is happening. If a cluster (driver + worker) has 16 cores then 16 tasks can be executed simultaniasly

Summary

- Spark distributes data partitions across multiple cores within the workers using tasks
- Jobs have multiple stages and stages multiple tasks.
- We learned about transformations and actions
- Spark is lazy and executes after an action and creates a DAG – Directed Acyclic Graph
- By that Spark can optimize the whole execution plan to increase efficiency of an ETL workflow