Memory Management & Optimisation for Spark

Tao Ruangyam, ING Analytics - Frankfurt Hub

Amsterdam, 03.02.2020

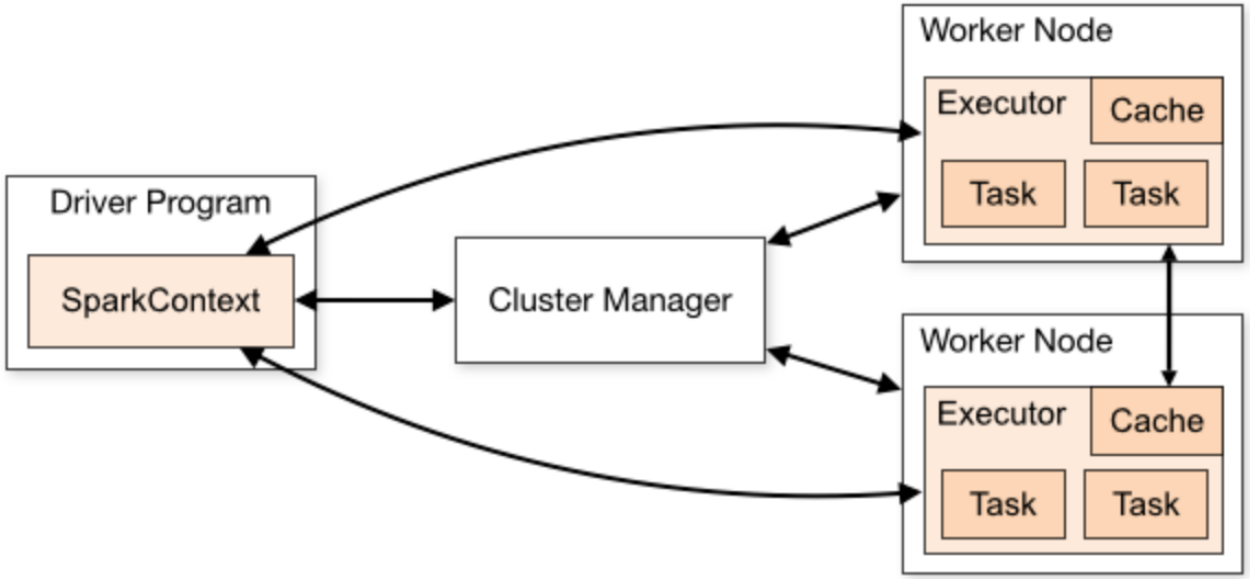
ING

Quick Facts about Spark

- Runs on **JVM**.
- Has **Garbage Collection**.
- Uses **Tungsten** for representing objects.
- **Lazy evaluation**, and highly optimisable.
- Spark is slower on small data.

ING

How does Spark Architecture look like?

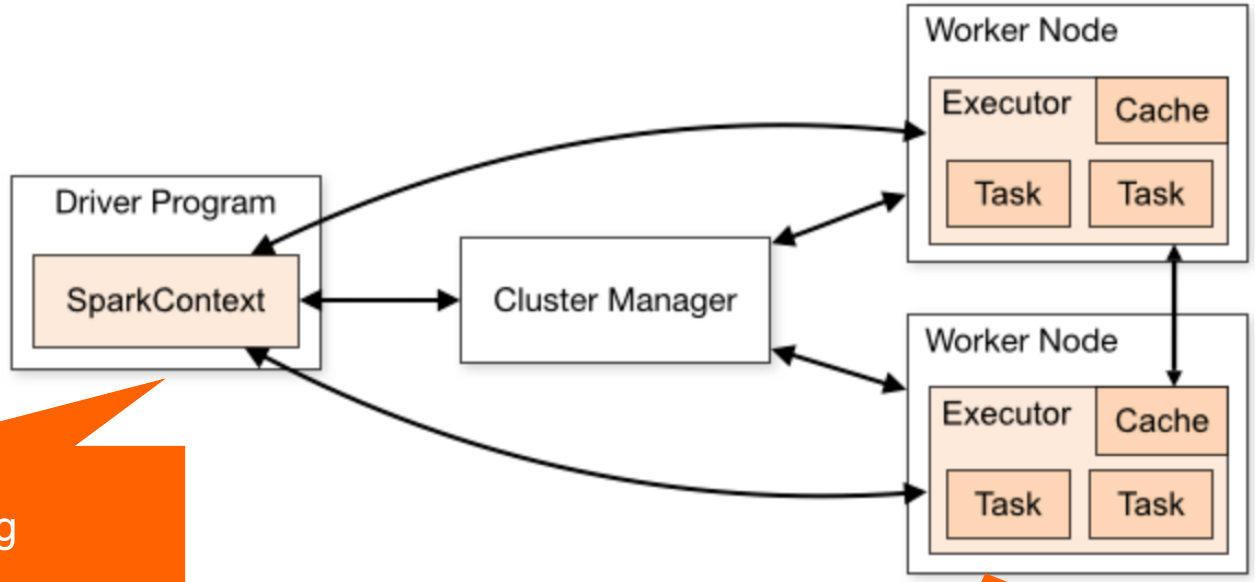


1 Driver process per Application

Multiple worker processes

ING

Configurable resources

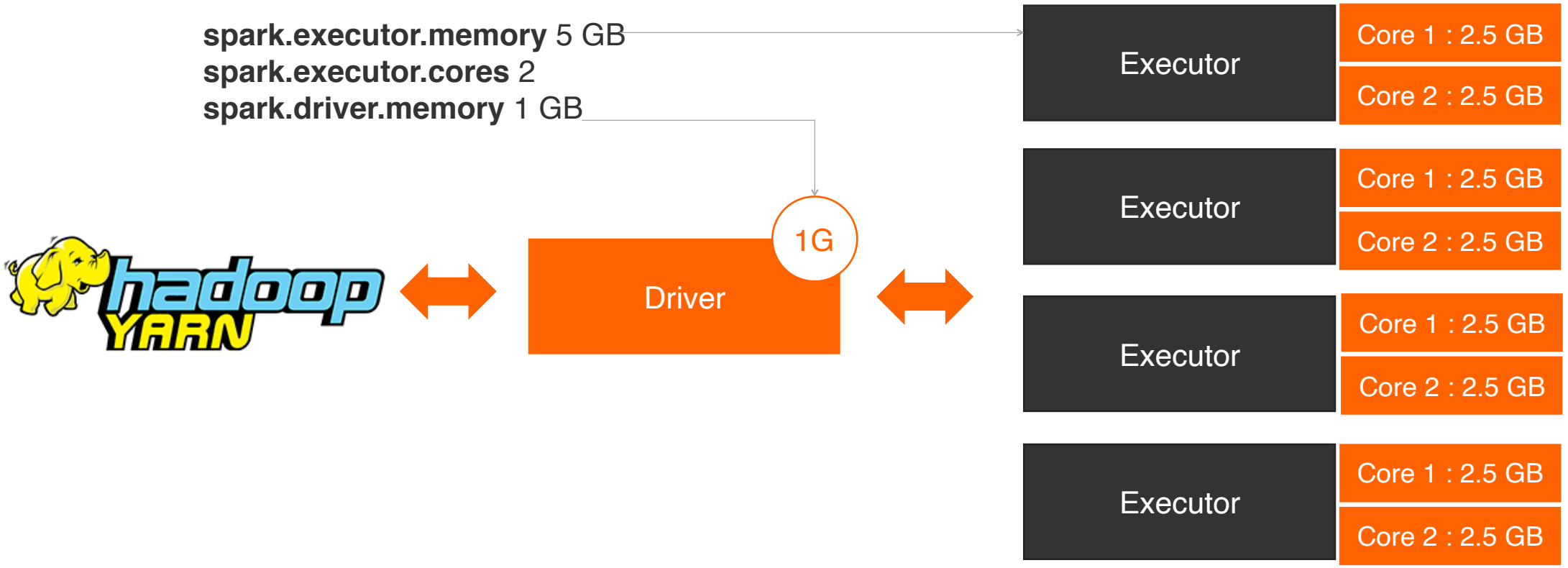


spark.driver.memory = 1g
spark.driver.cores = 1
spark.driver.maxResultSize = 1g

spark.executor.memory = 1g
spark.executor.cores = 1
spark.dynamicAllocation.enabled = false

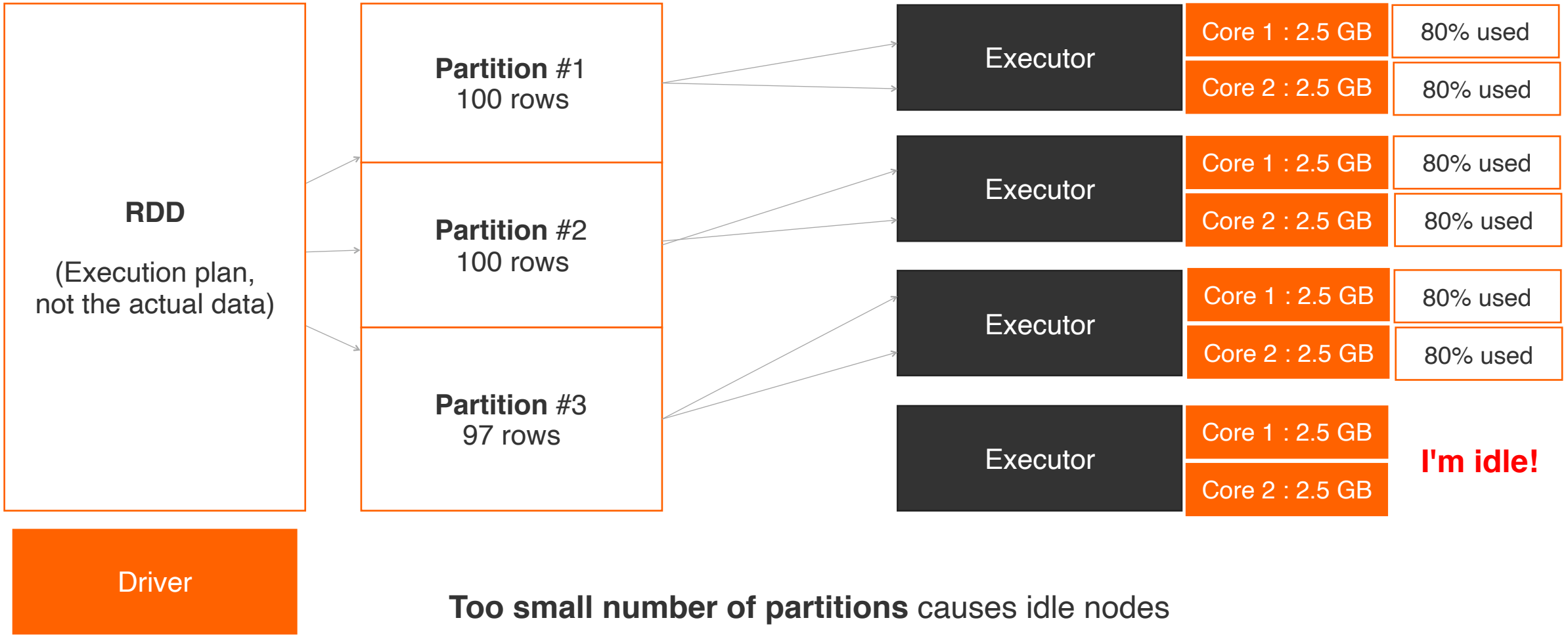
ING

How memory is allocated on Spark



ING

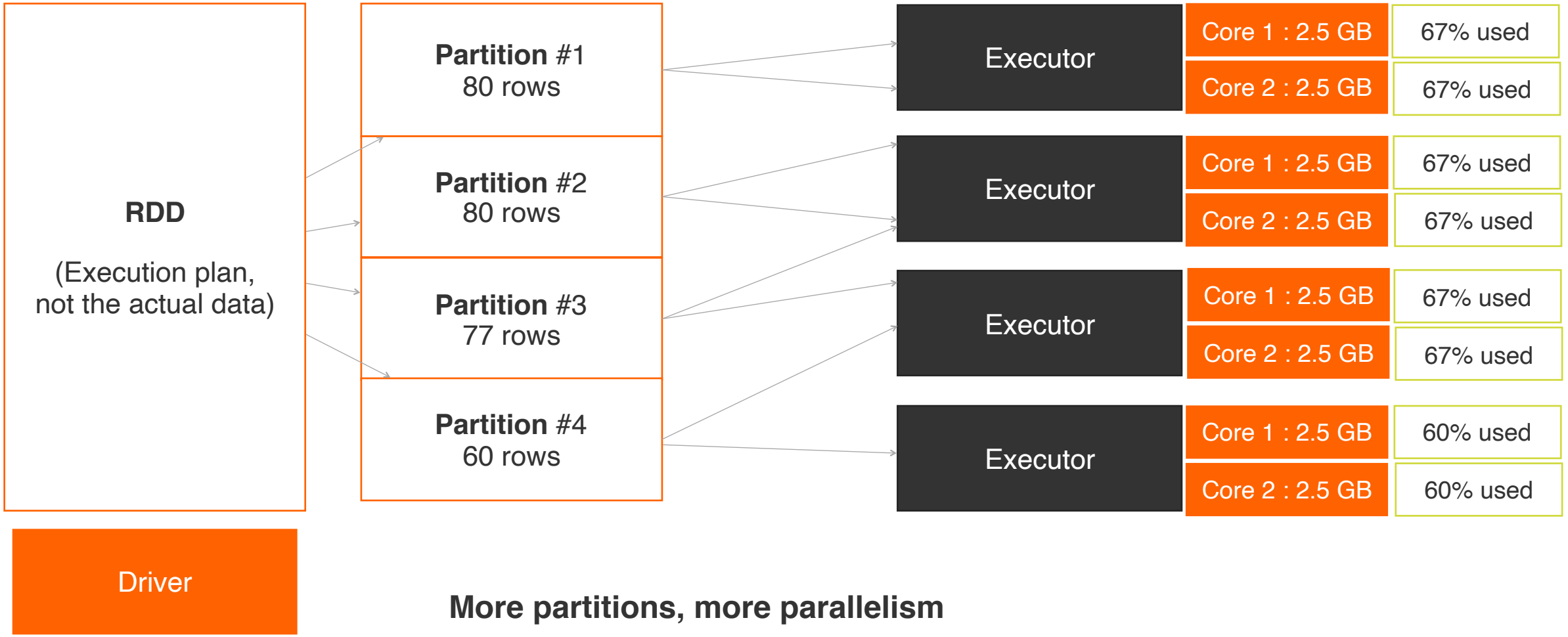
How RDD is stored on each executor



Too small number of partitions causes idle nodes

ING

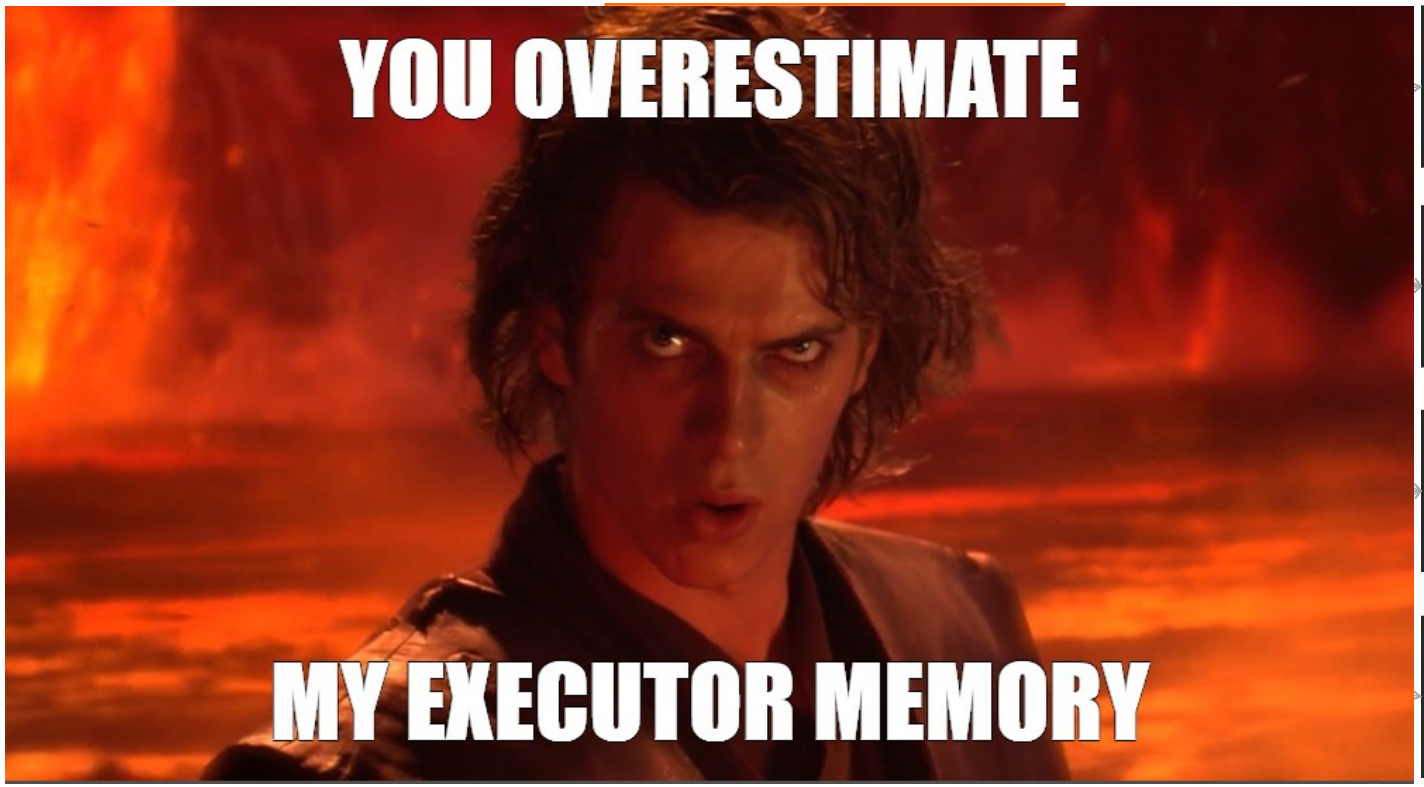
Find the right number of partitions



More partitions, more parallelism

ING

What if an executor is memory overloaded?



Executor	Core 1 : 2.5 GB	70% used
	Core 2 : 2.5 GB	80% used
Executor	Core 1 : 2.5 GB	75% used
	Core 2 : 2.5 GB	80% used
Executor	Core 1 : 2.5 GB	70% used
	Core 2 : 2.5 GB	75% used
Executor	Core 1 : 2.5 GB	150% used
	Core 2 : 2.5 GB	140% used

Overloading excessive memory on an executor can bring it down.

ING

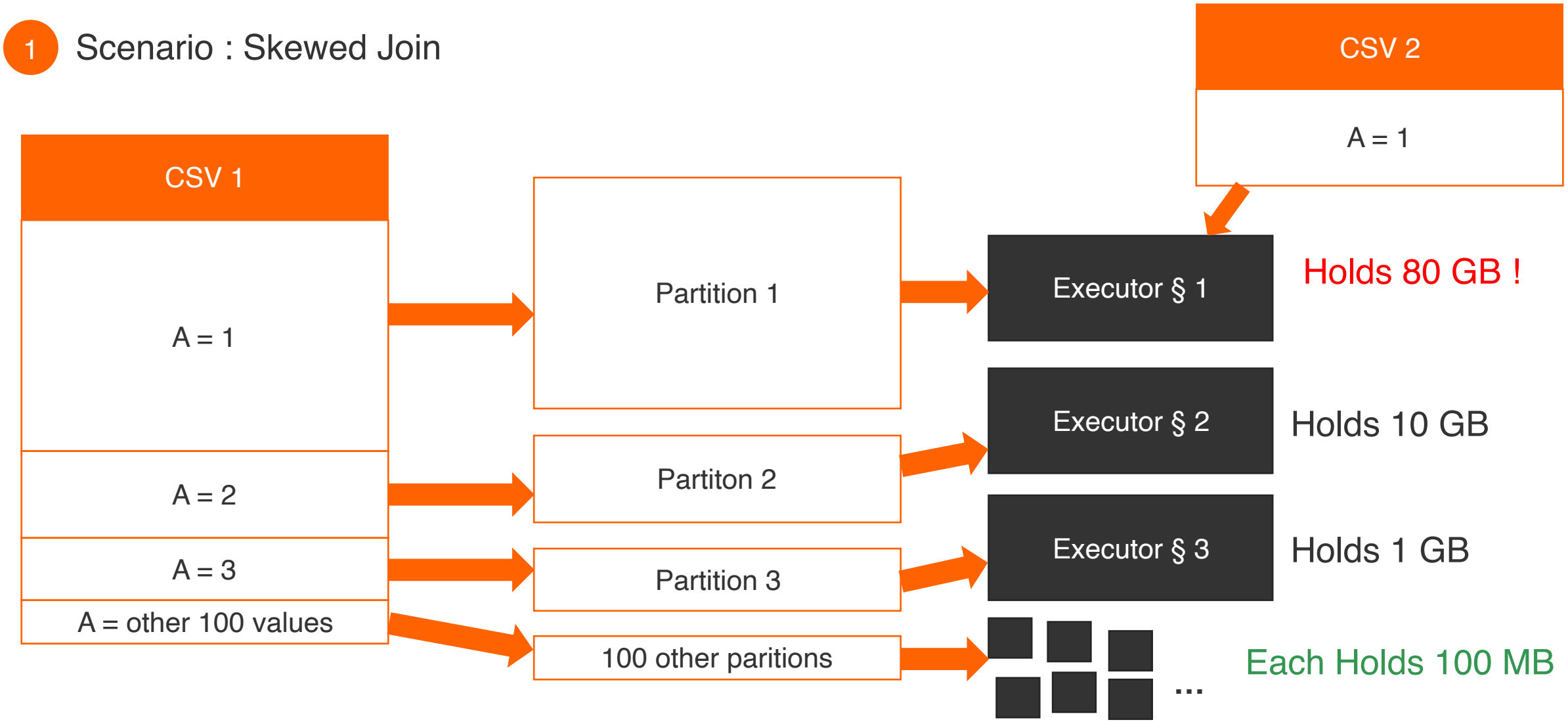
Root causes of executor memory overload

1 Scenario : Skewed Join

ING

Root causes of executor memory overload

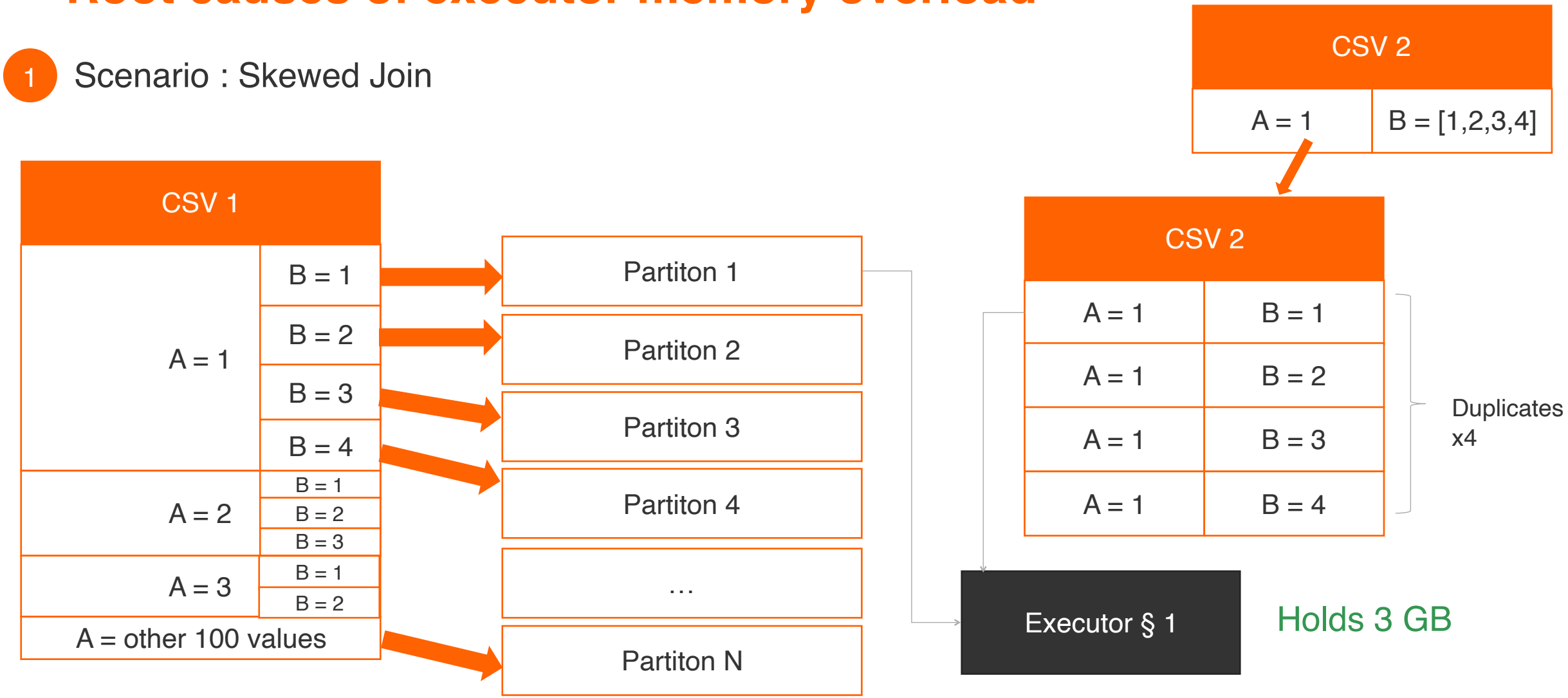
1 Scenario : Skewed Join



ING

Root causes of executor memory overload

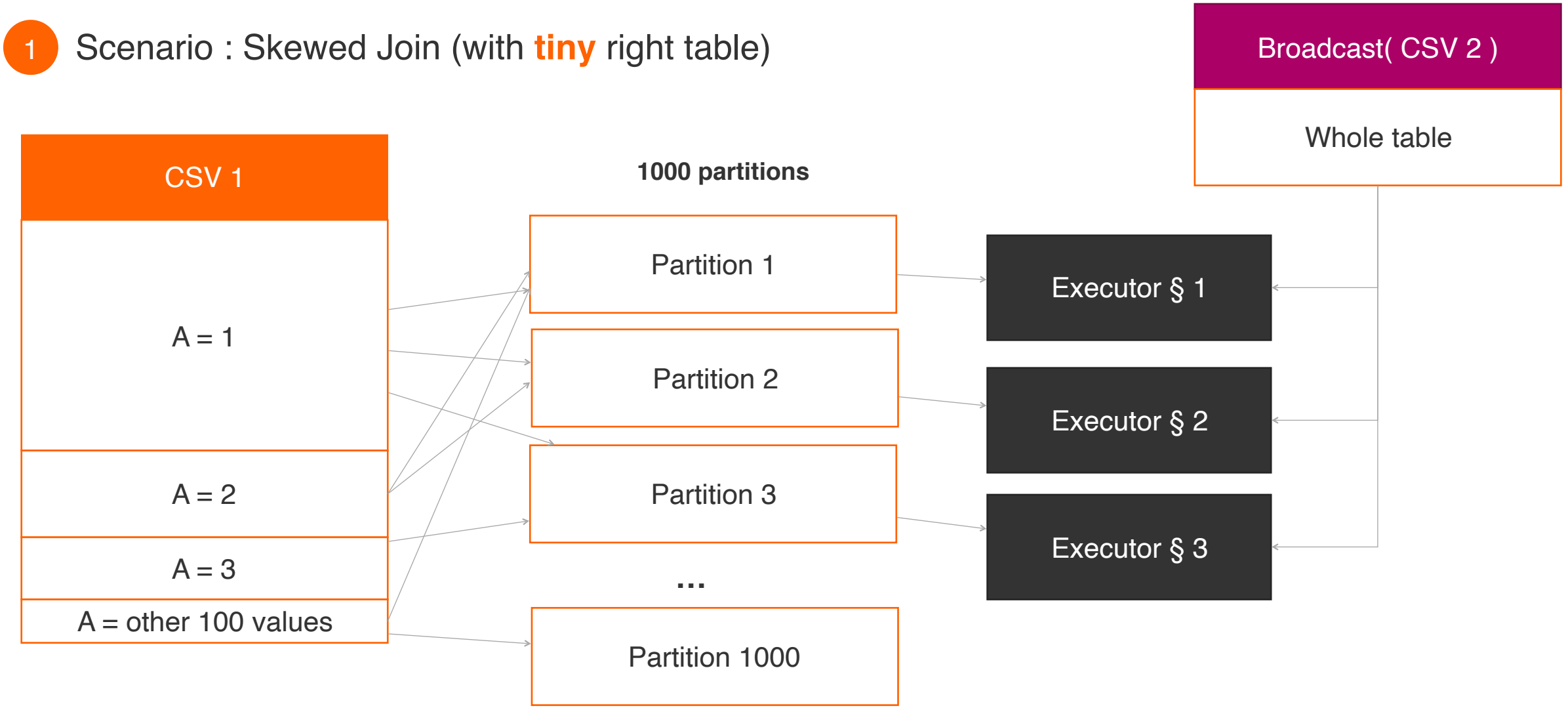
1 Scenario : Skewed Join



ING

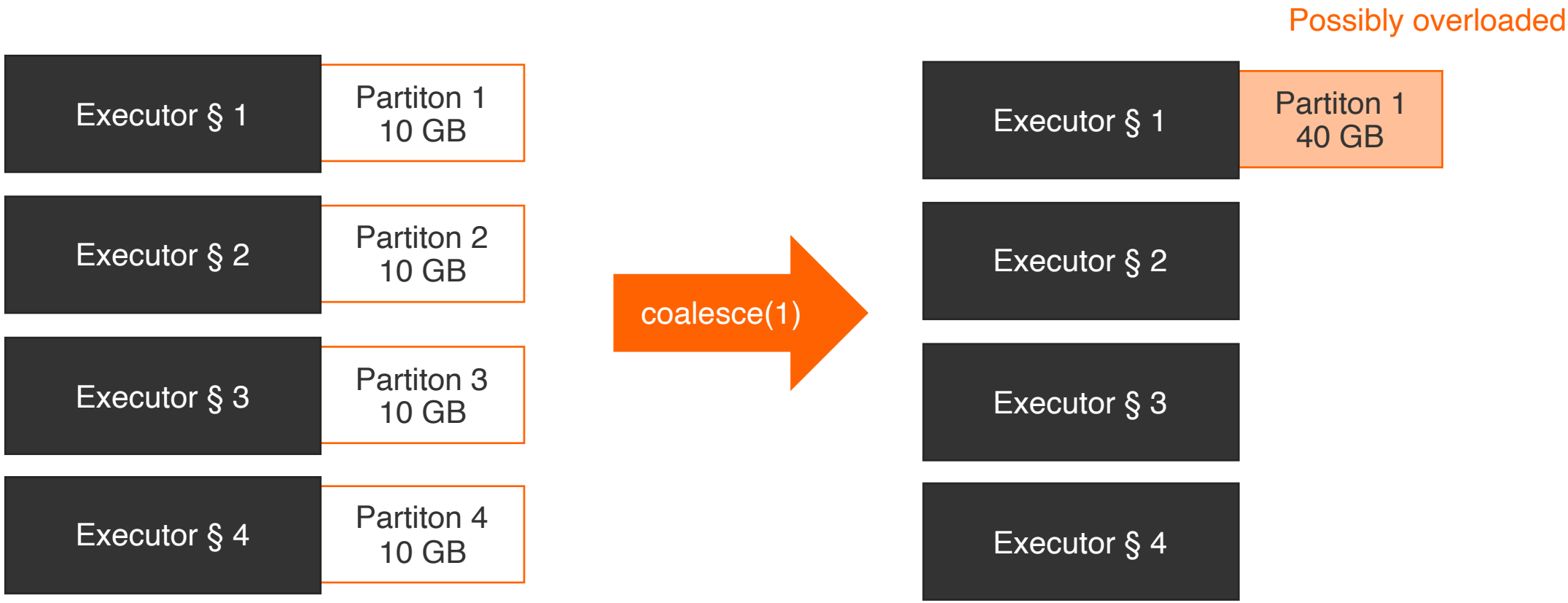
Root causes of executor memory overload

1 Scenario : Skewed Join (with **tiny** right table)



Root causes of executor memory overload

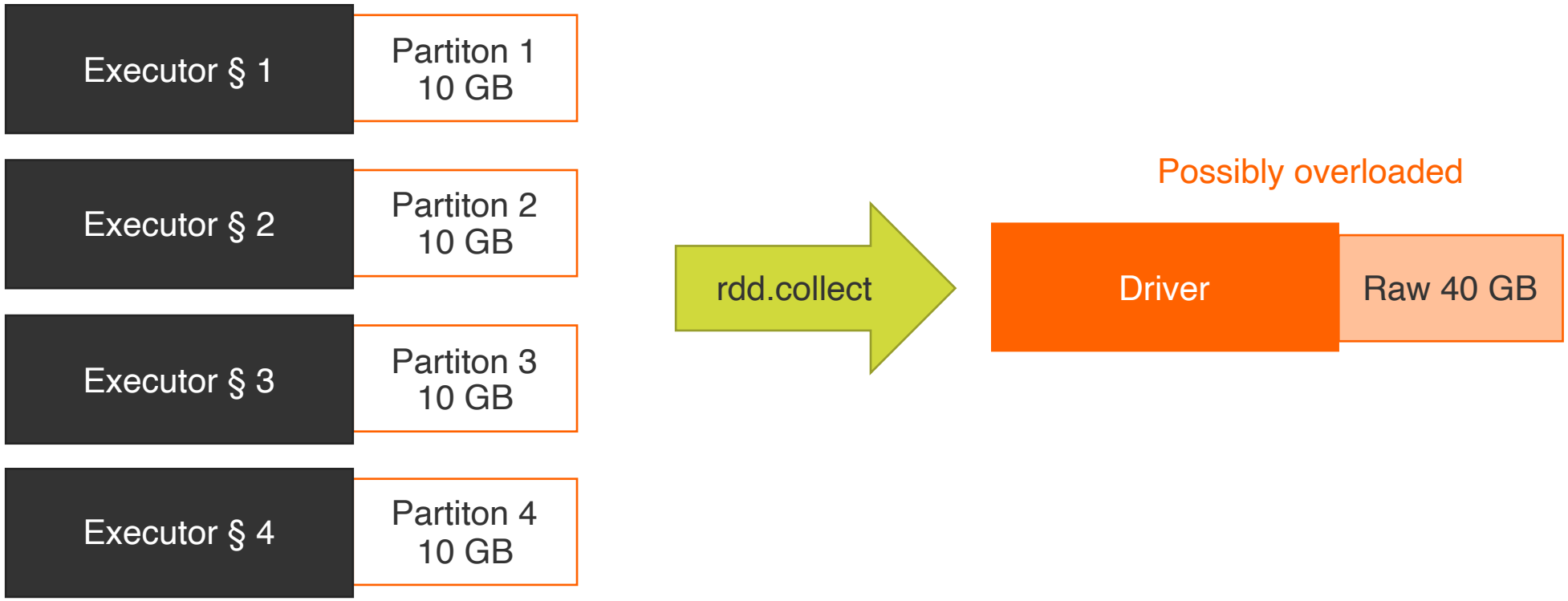
2 Scenario : Coalesce large data



ING

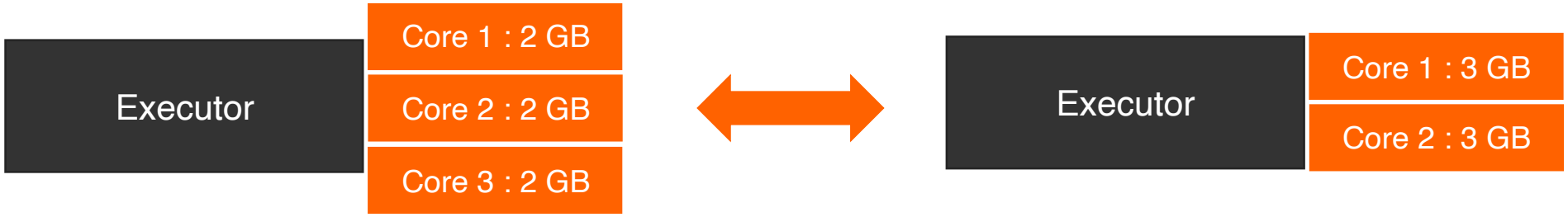
Root causes of driver memory overload

3 Scenario : Collect large data



ING

Trade-offs : cores & memory



spark.executor.memory = 6 GB
spark.executor.cores = 3

spark.executor.memory = 6 GB
spark.executor.cores = 2

More parallelsim

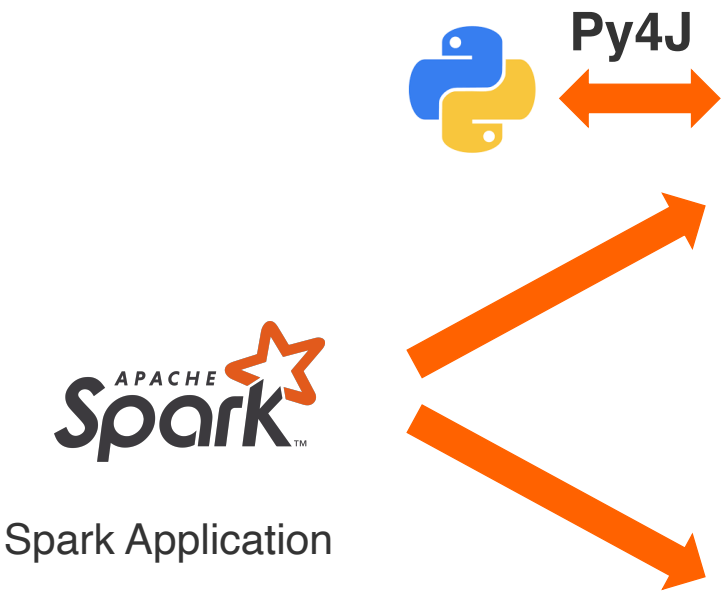
Less parallelsim

ING

Spark RDD, Dataset API

	What's in each row	Memory usage	Serialiser	Compatibility
RDD[object]	Java Object	Larger (JVM)	Java (Kryo)	Scala, Java
DataFrame	"Row" object	Smaller (Off-JVM)	Tungsten	Scala, Java, Python, R
Dataset[Object]	Binary object	Smaller (Off-JVM)	Java & Tungsten	Scala, Java

JVM v Off-JVM ?



JVM

- Execution code
- Functions, UDF
- Classes
- Cache
- RDD

KryoSerializer

- Optimised for **Storage**
- **JVM** specific
- **Garbage Collected**

Off-Heap

- Rows as Binary
- Non-executable data
- DataFrame
- SQL

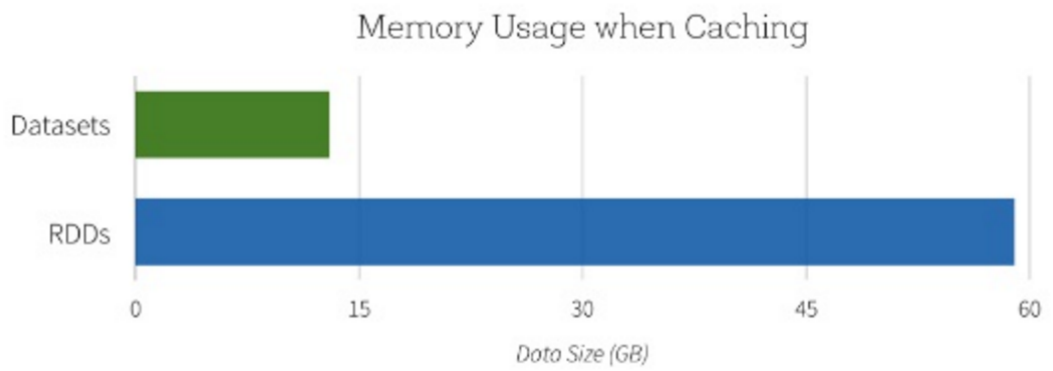
Tunsten Serializer

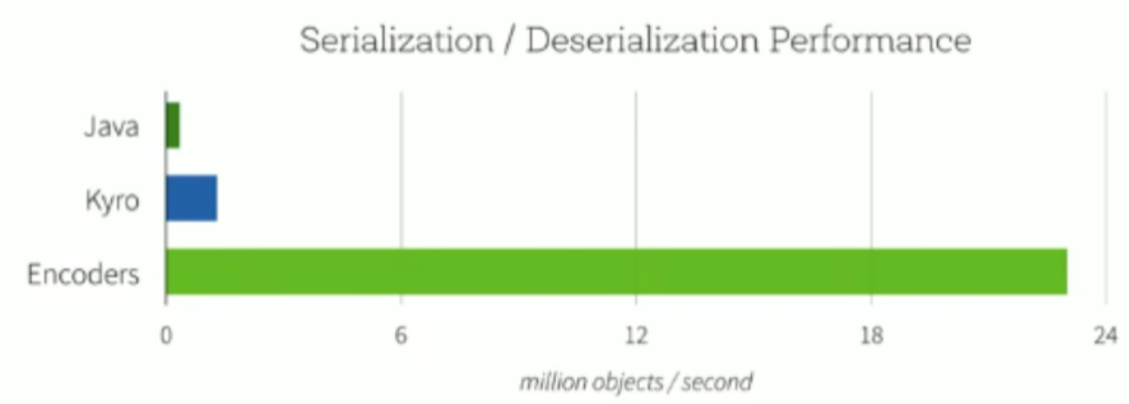
- Optimised for **Memory**
- **Universal**
- **Optimised for CPU**

ING

Tungstens as an object encoder

Space Efficiency





More info: https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html
And https://github.com/spoddutur/spark-notes/blob/master/deep_dive_into_storage_formats.md

ING

Tungsten binary format – vs - Python

Row Schema

Key	Name	Feature
Int	String	Array[Double]

Tungsten Row (DF)

Memory block representing a row

Python Memory Map

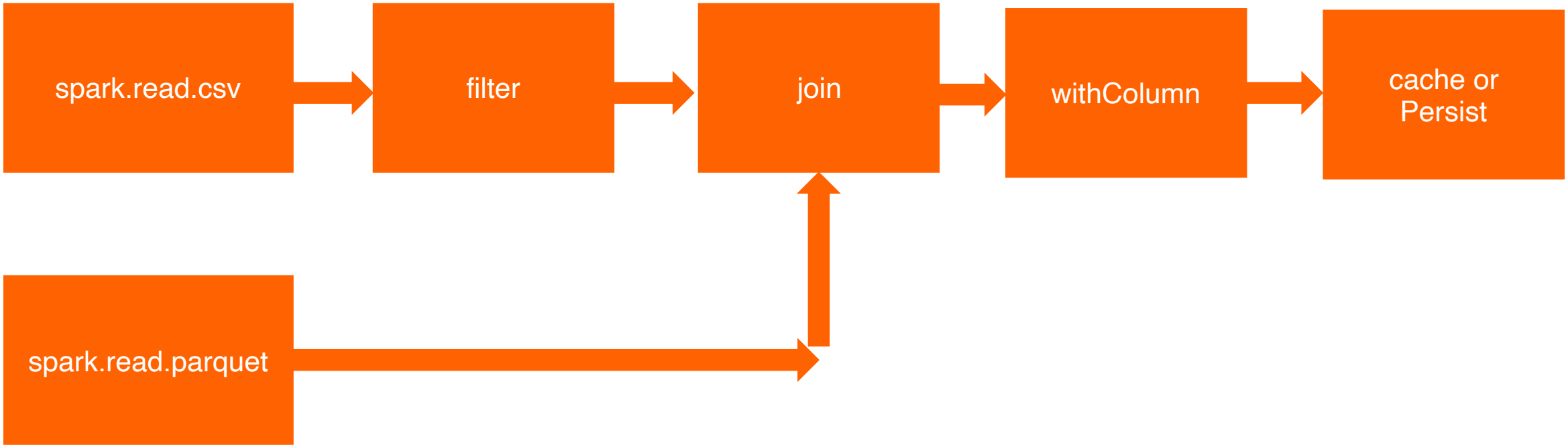
Key	Name	Feature
24 bytes	49 bytes overhead + **X bytes** per character	64 bytes overhead + **Y bytes** per element

Memory alignment in Python, always larger than original

ING

Note that Spark is lazy evaluated

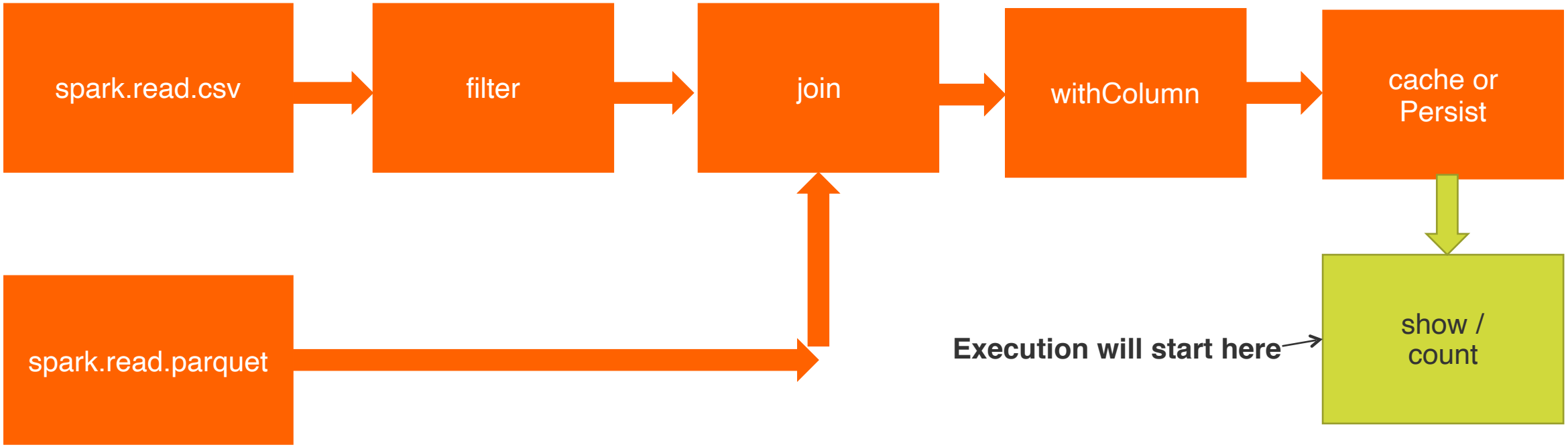
Following DAG **will not be executed rightway** after declaration.



ING

Note that Spark is lazy evaluated

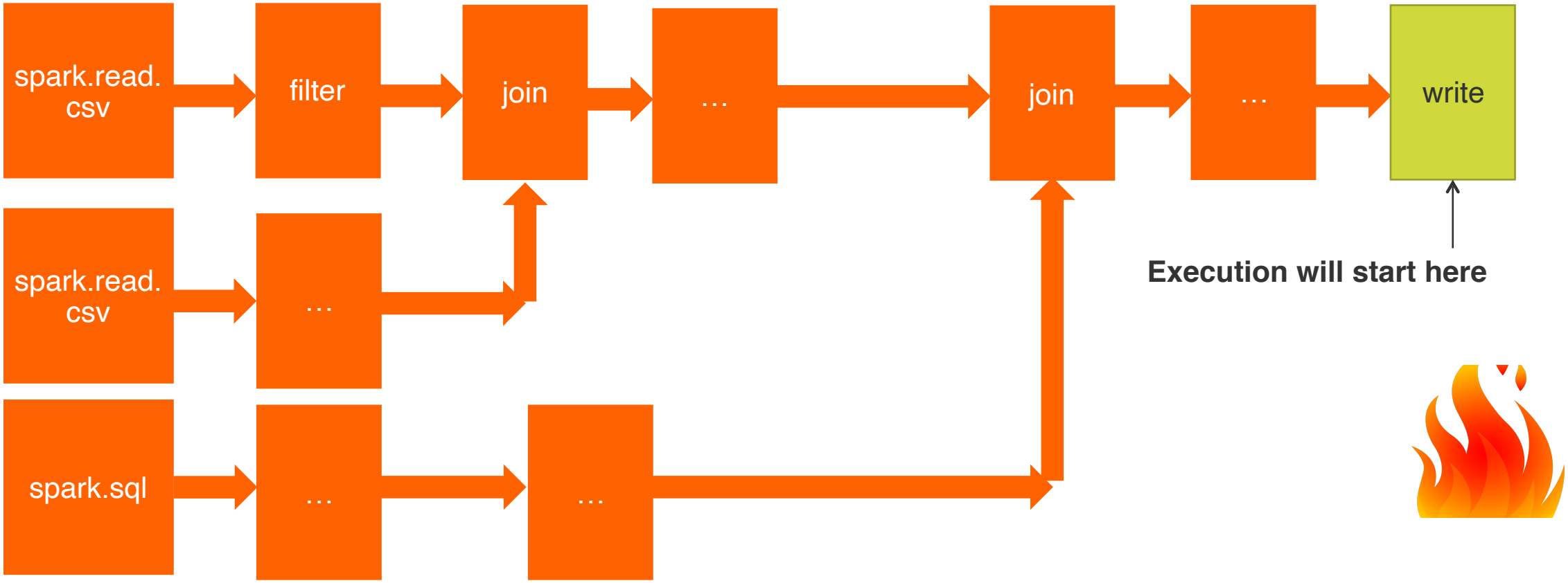
Materialisation happens when the result is **needed**.



ING

Note that Spark is lazy evaluated

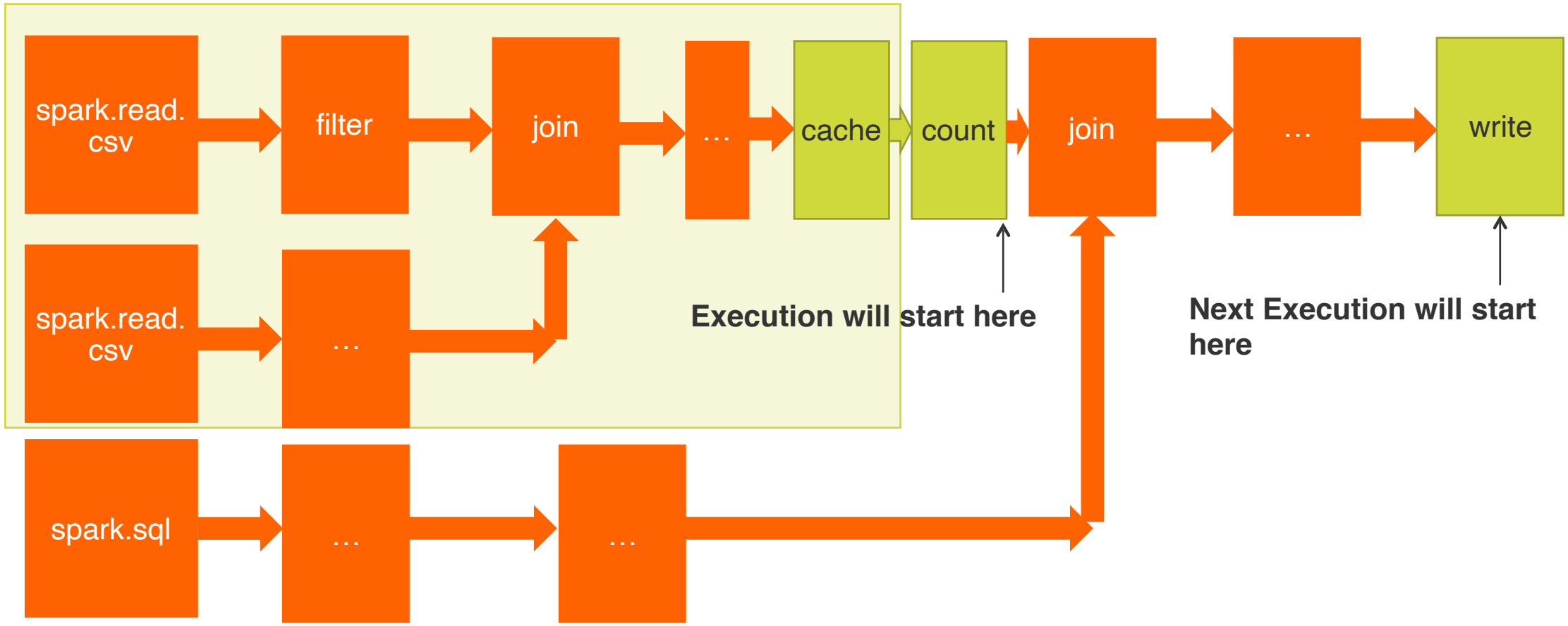
WARNING: Large DAG causes high overhead and less reliablity



ING

Note that Spark is lazy evaluated

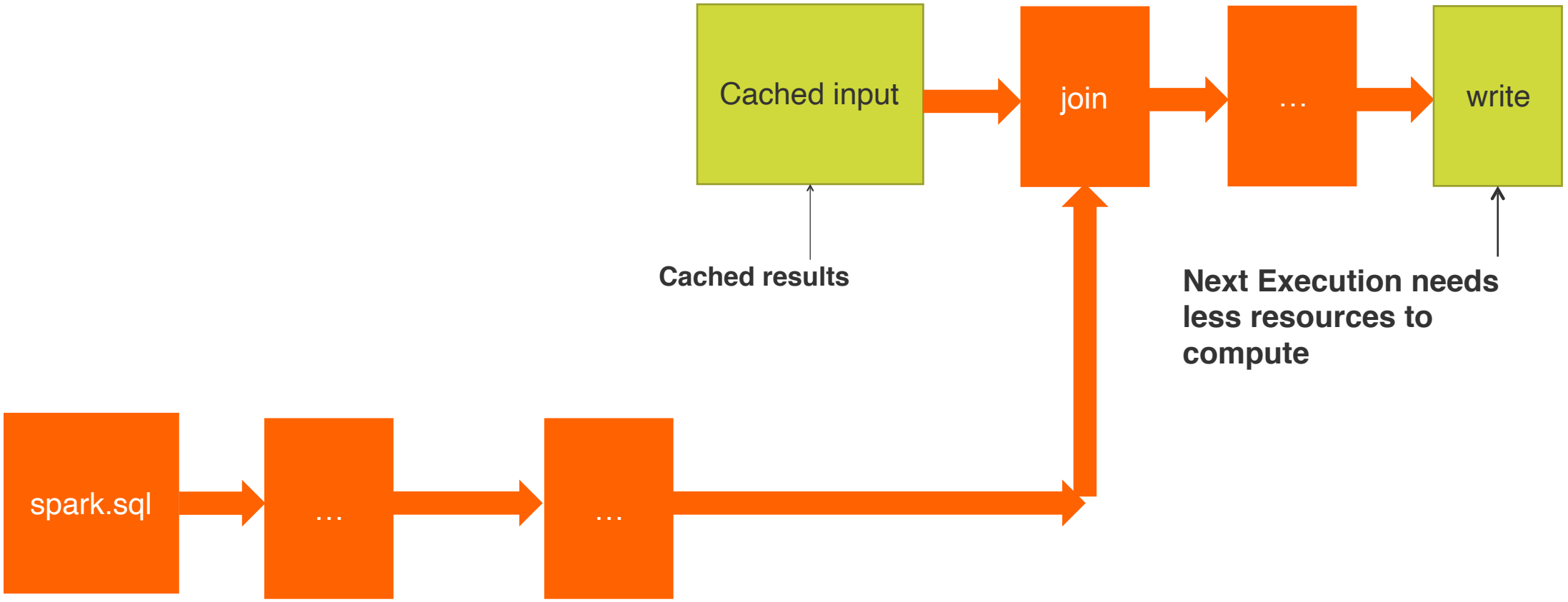
TIPS: Split DAG for more efficiency, by adding materiasliation enforcement



ING

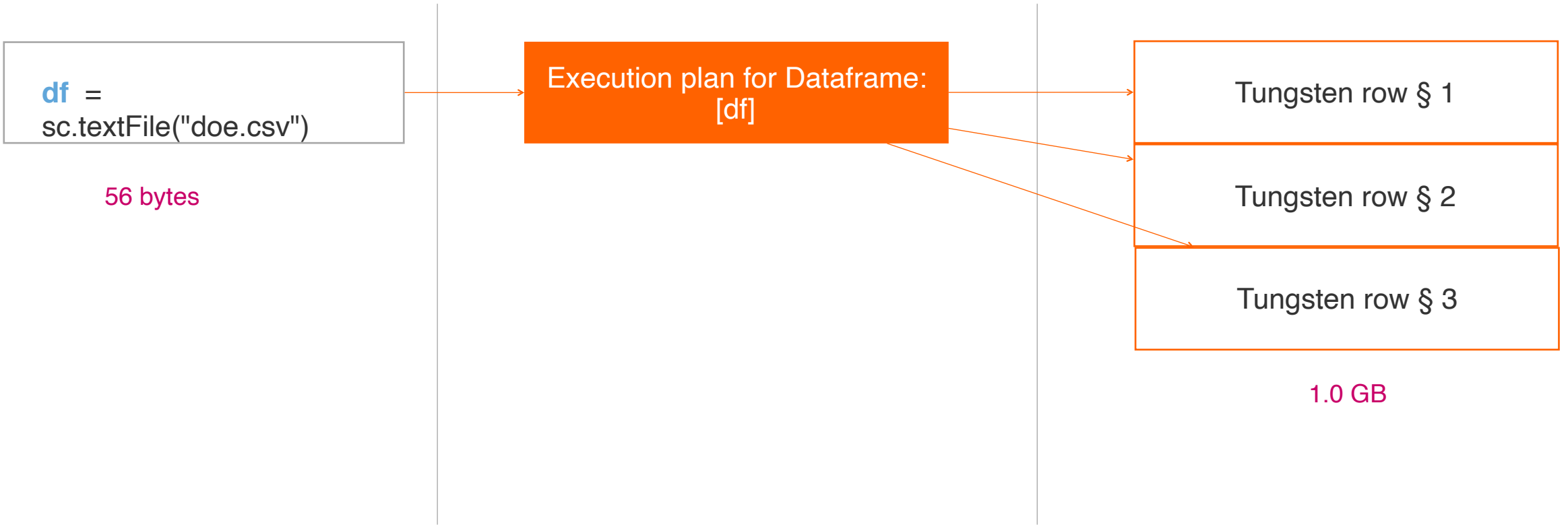
Note that Spark is lazy evaluated

TIPS: Split DAG for more efficiency, by adding materiasliation enforcement

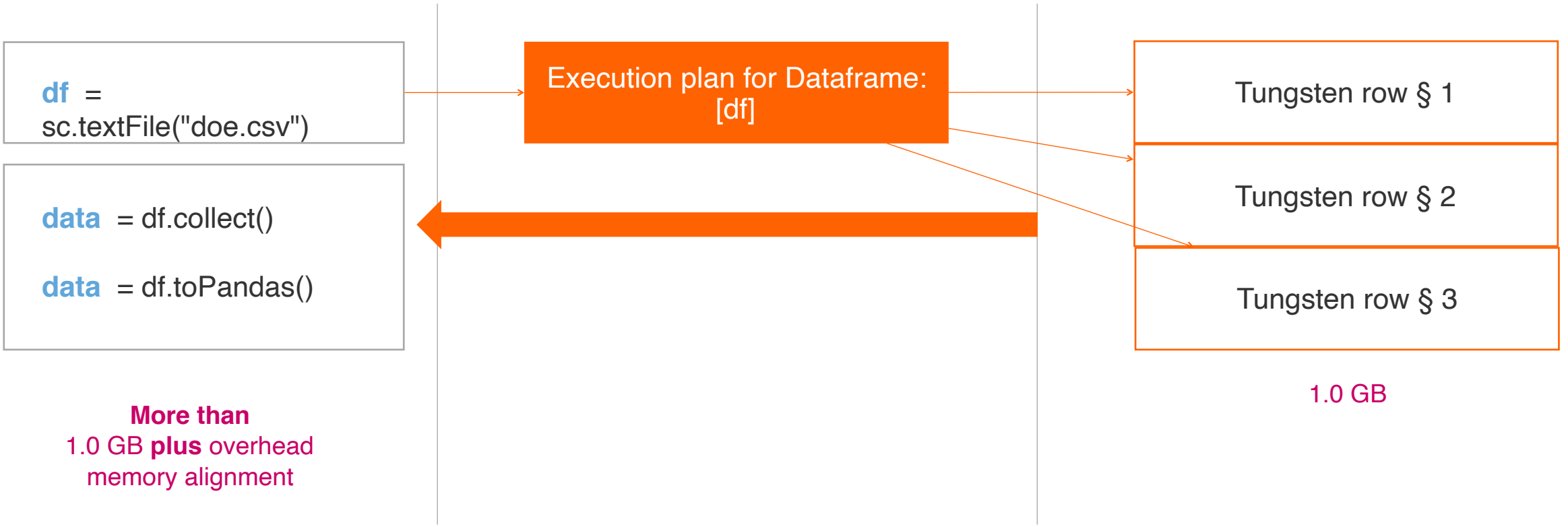


ING

PySpark : Where the memory is stored?



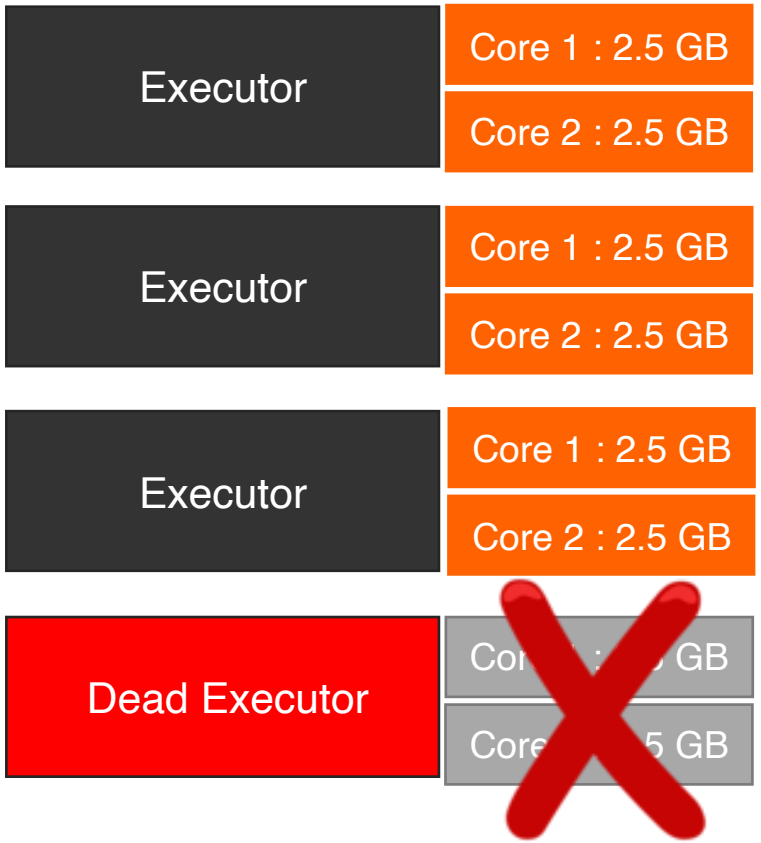
Collecting data from Spark to Python



Common Spark Failures due to memory

Case: Executor dies of Out of memory
- Unable to connect to executor machine
- Executor raising OOM
- Timeout while waiting for a task to finish

Possible cause:
- Low number of partition
- Too many cores
- Skewness of joining data (Big DF vs Tiny DF)
- Too large DAG
- Unstable cluster



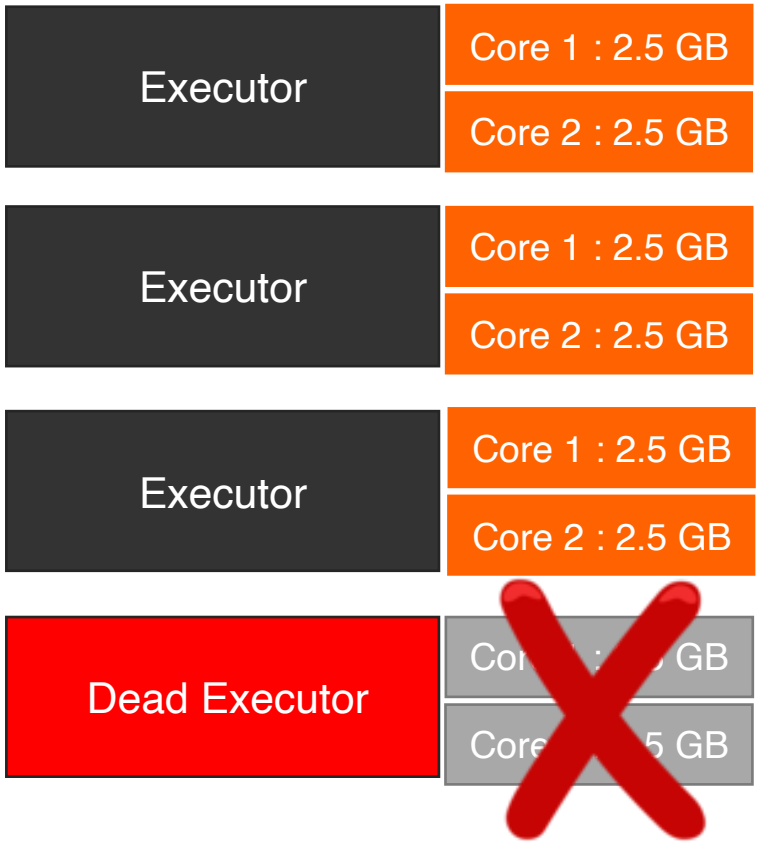
Common Spark Failures due to memory

Case: Executor dies of Out of memory

- Unable to connect to executor machine
- Executor raising OOM
- Timeout while waiting for a task to finish

Possible cause:

- Low number of partition : Increase number of partitions
- Too many cores : Decrease number of core per executor
- Skewness of joining data (Big DF vs Tiny DF) : Broadcast
- Too large DAG : Cut by forcing materialisation after cache
- Unstable cluster : Infrastructure solution



More useful resources

Spark Partitioning	https://medium.com/@adrianchang/apache-spark-partitioning-e9faab369d14
Spark Cluster overview	https://spark.apache.org/docs/latest/cluster-overview.html
Spark Configurations	https://spark.apache.org/docs/latest/configuration.html
Tungsten overview	https://jaceklaskowski.gitbooks.io/mastering-spark-sql/spark-sql-tungsten.html
RDD v DataFrame v Dataset	https://databricks.com/blog/2016/07/14/a-tale-of-three-apache-spark-apis-rdds-dataframes-and-datasets.html
RDD v DataFrame v Dataset (more advanced)	https://data-flair.training/blogs/apache-spark-rdd-vs-dataframe-vs-dataset/
KryoSerialiser Github	https://github.com/EsotericSoftware/kryo
Spark Memory Management (v.2.x)	https://databricks.com/session/deep-dive-apache-spark-memory-management

ING