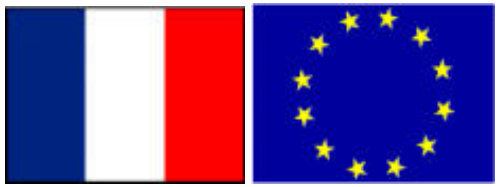


Big Data

Spark application deployment

Stéphane Vialle **&** **Gianluca Quercini**

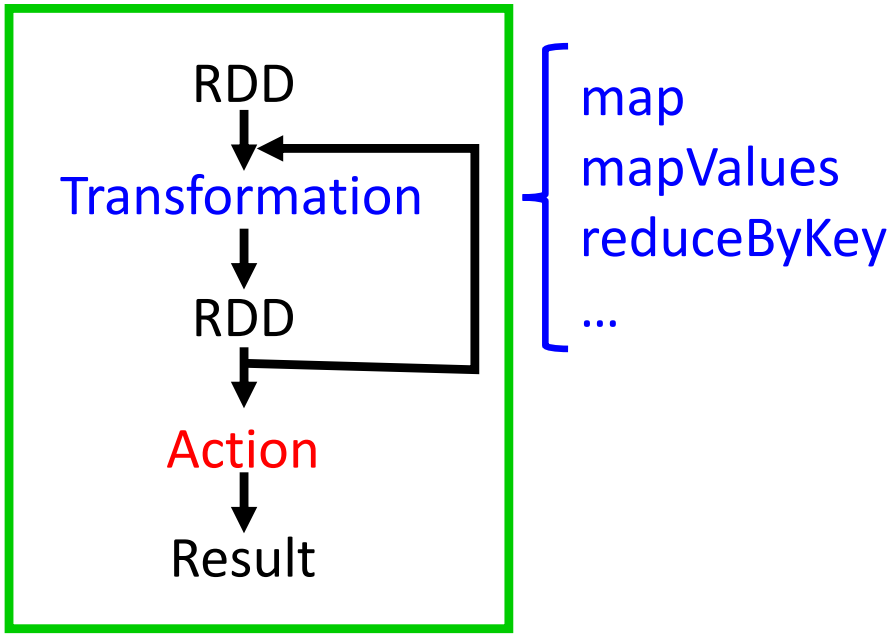
Spark application deployment

1. **Task DAG execution**
2. Spark execution on clusters

Task DAG execution

- *A RDD is a dataset distributed among the Spark compute nodes*
- *Transformations* are **lazy** operations: saved and executed further
- *Actions* **trigger** the execution of the sequence of transformations

A *job* is a sequence of RDD transformations, ended by an action



A ***Spark application*** is ***a set of jobs*** to run sequentially or in parallel
→ A DAG of tasks

Task DAG execution

The *Spark application driver* controls the application run

- It creates the Spark context
- It analyses the Spark program

- It **creates** a **DAG of tasks** for each job
- It **optimizes** the DAG
  - pipelining narrow transformations
  - identifying the tasks that can be run in parallel

- It **schedules** the DAG of tasks on the available worker nodes (the *Spark Executors*) **in order to maximize parallelism** (and to reduce the execution time)

Task DAG execution

Spark job trace: on 10 Spark executors, with 3GB input file

DAGScheduler: **Submitting 24 missing tasks** from ShuffleMapStage 0 ...
TaskSchedulerImpl: **Adding task set 0.0 with 24 tasks**
...

TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, 172.20.10.14, executor 0, partition 1, ...)
TaskSetManager: Starting task 2.0 in stage 0.0 (TID 2, 172.20.10.11, executor 7, partition 2, ...)
...
TaskSetManager: Starting task 10.0 in stage 0.0 (TID 10, 172.20.10.11, executor 7, partition 10, ...)

TaskSetManager: Finished task 2.0 in stage 0.0 (TID 2) in 18274 ms … (executor 7) (1/24)
TaskSetManager: Starting task 11.0 in stage 0.0 (TID 11, 172.20.10.7, executor 8, partition 11, ...)
TaskSetManager: Finished task 8.0 in stage 0.0 (TID 8) in 18459 ms … (executor 8) (2/24)
...

TaskSchedulerImpl: **Removed TaskSet 0.0, whose tasks have all completed**, from pool
...

Submitting the 10 first tasks on the 10 Spark executor processes

Submitting a new task when a previous one has finished

End of task graph execution

Task DAG execution

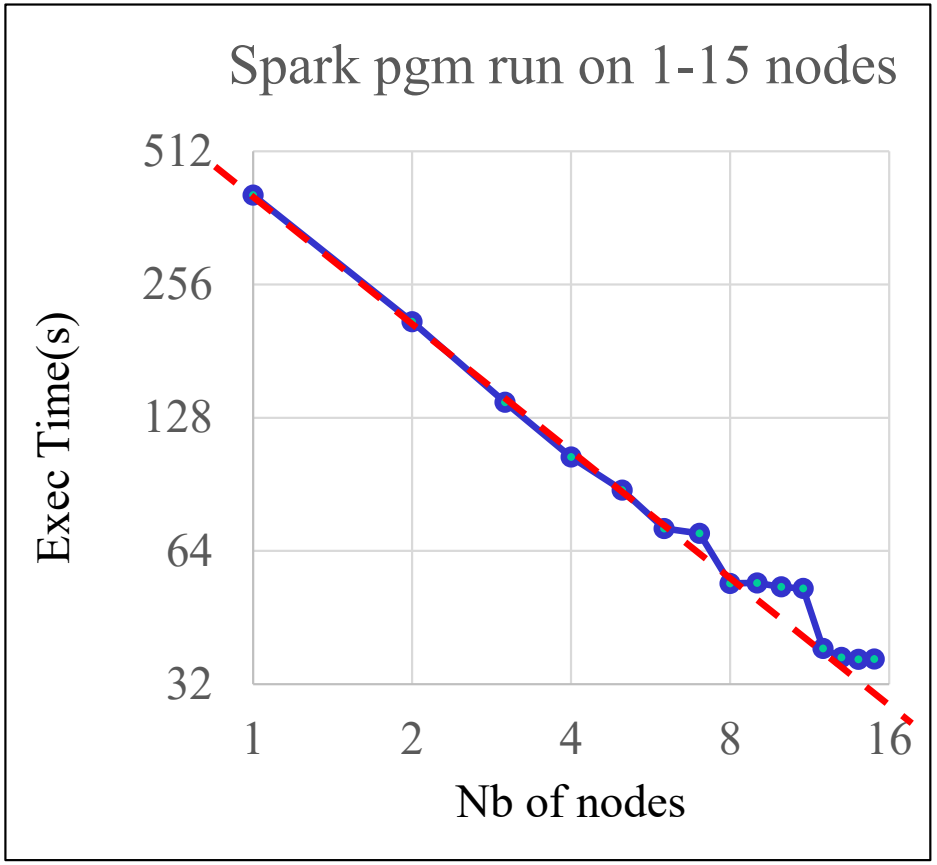
Execution time as a function of the number of Spark executors

Ex. of Spark application run:

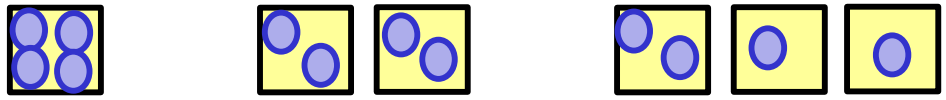
- from 1 up to 15 executors
- with 1 executor per node

Good overall decrease but plateaus appear !

Probable **load balancing problem**…



Ex: a graph of 4 parallel tasks



on 1 node: T

on 2 nodes: T/2

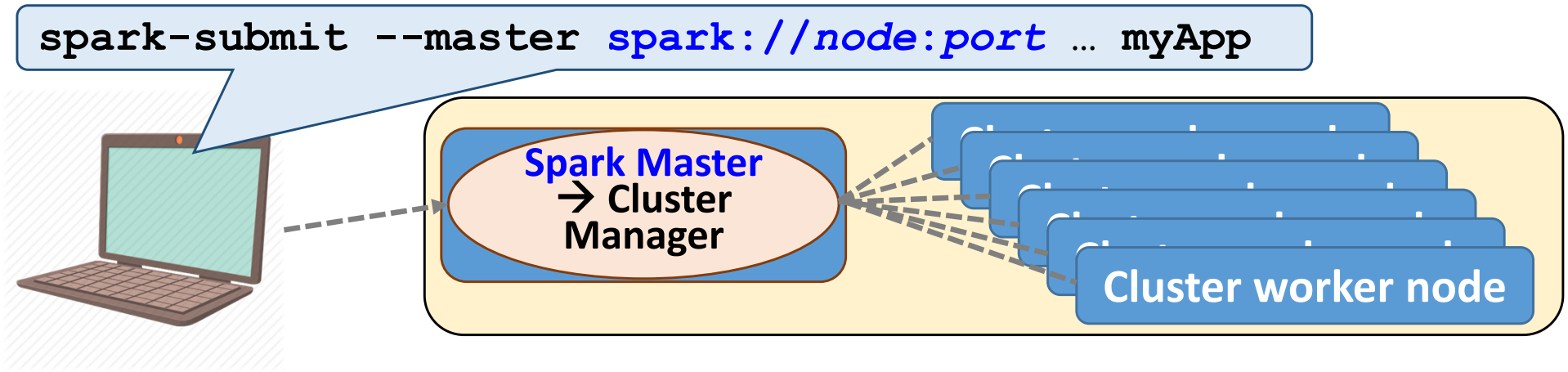
on 3 nodes: T/**2**

→ A plateau appears

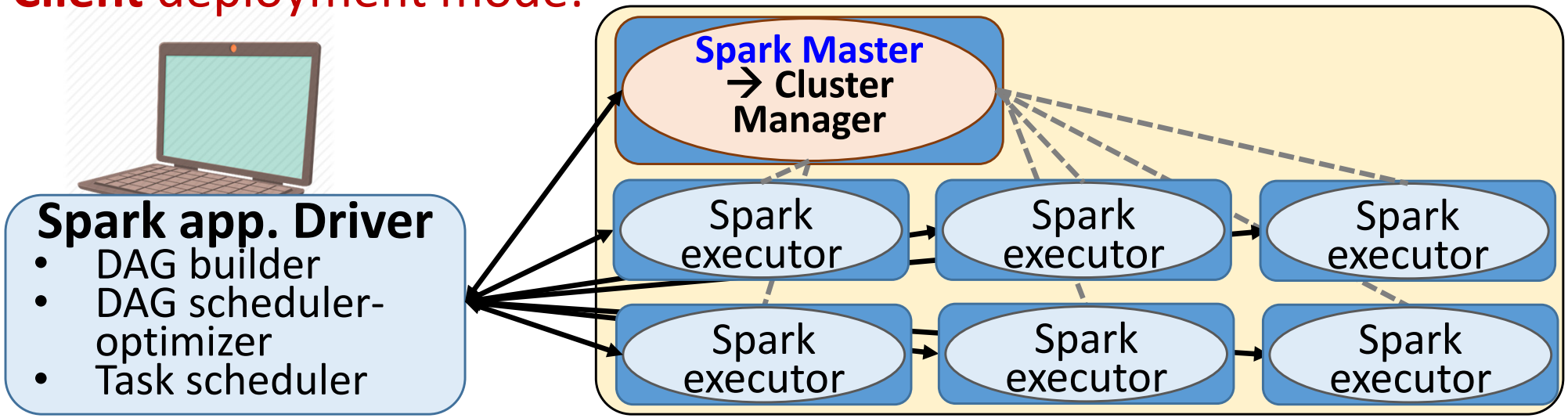
Spark application deployment

1. Task DAG exécution
2. **Spark execution on cluster**
   (standalone mode)

Spark exécution on cluster (standalone mode)

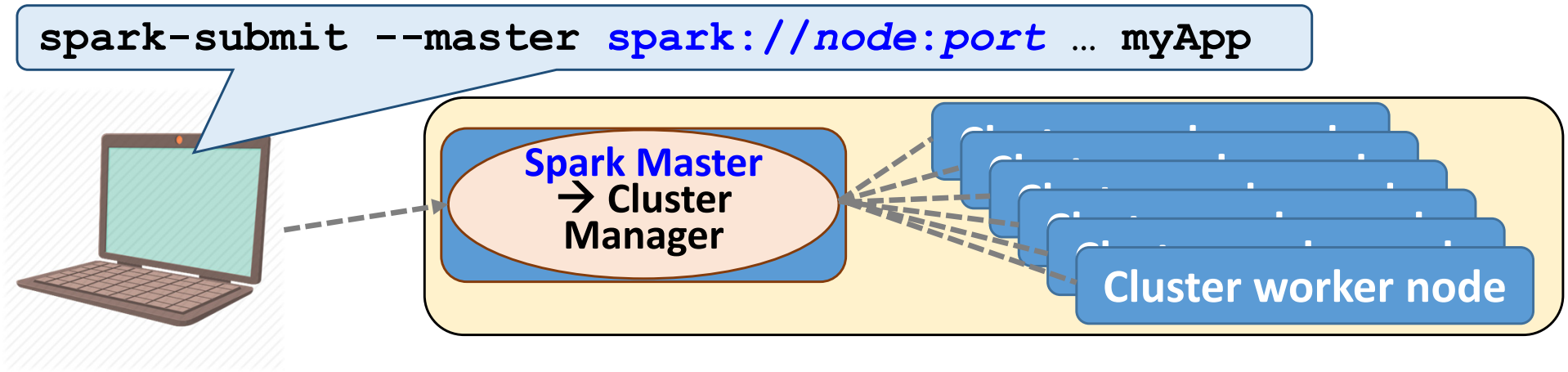


Client deployment mode:

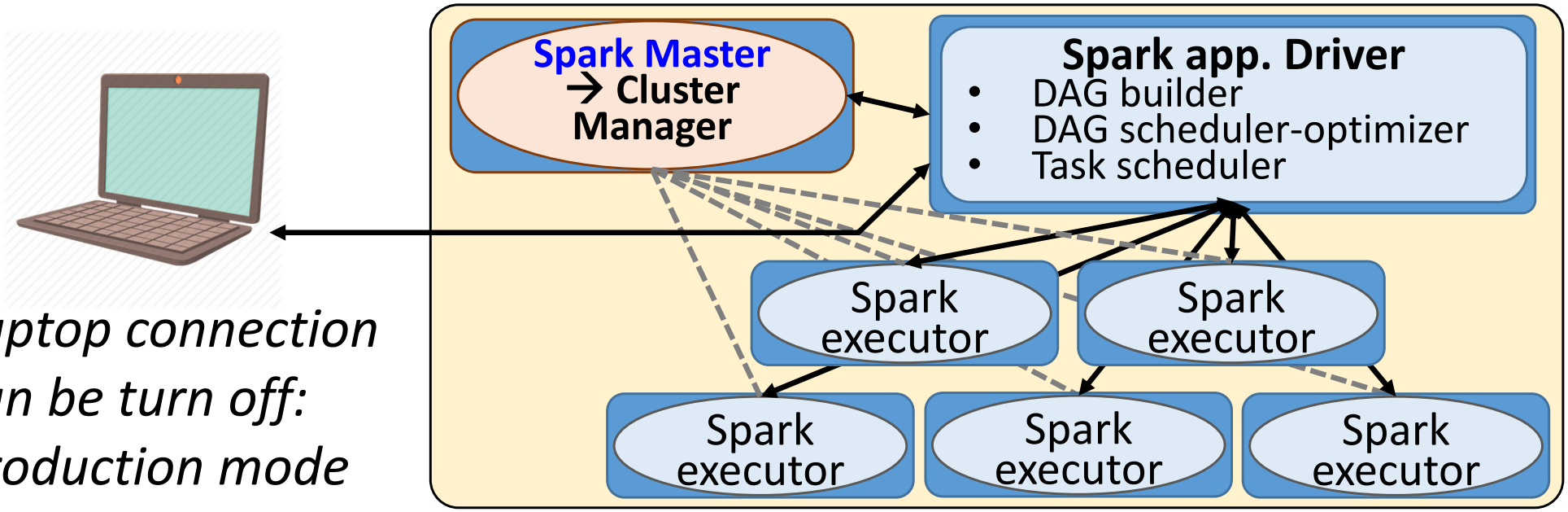


Interactive control of the application: development mode

Spark exécution on cluster (standalone mode)

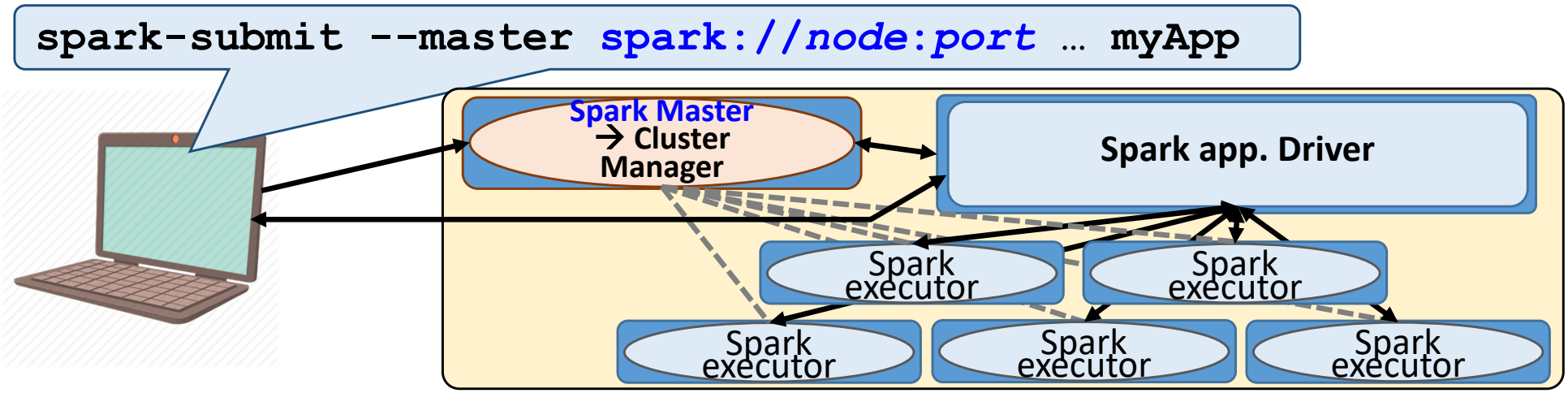


Cluster deployment mode:



Laptop connection can be turn off: production mode

Spark exécution on cluster (standalone mode)



Spark cluster configuration:

- Add the list of cluster worker nodes in the Spark Master config.
- Specify the maximum amount of memory per Spark Executor

```
spark-submit  --executor-memory XX …
```

default: **1GB**

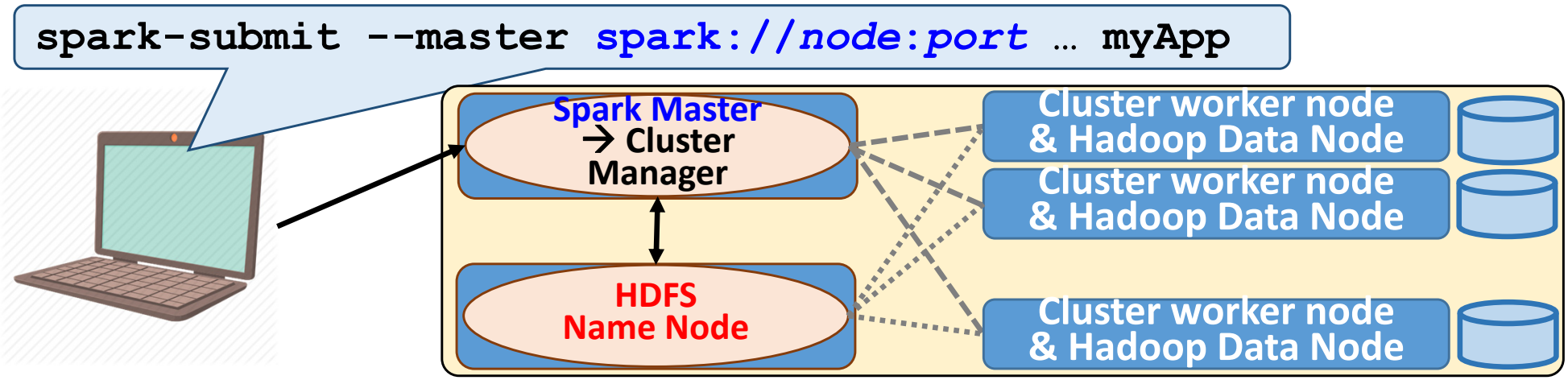
- Specify the total amount of CPU cores to use per Spark Application

```
spark-submit  --total-executor-cores YY …
```

default: nbWorkerNodes × nbCoresPerNode

Flooding strategy!

Spark exécution on cluster (standalone mode)



Spark cluster configuration:

To get a global *data-computations locality*:

- *The Spark Worker nodes* are connected to the *Data nodes* through a (very) fast network

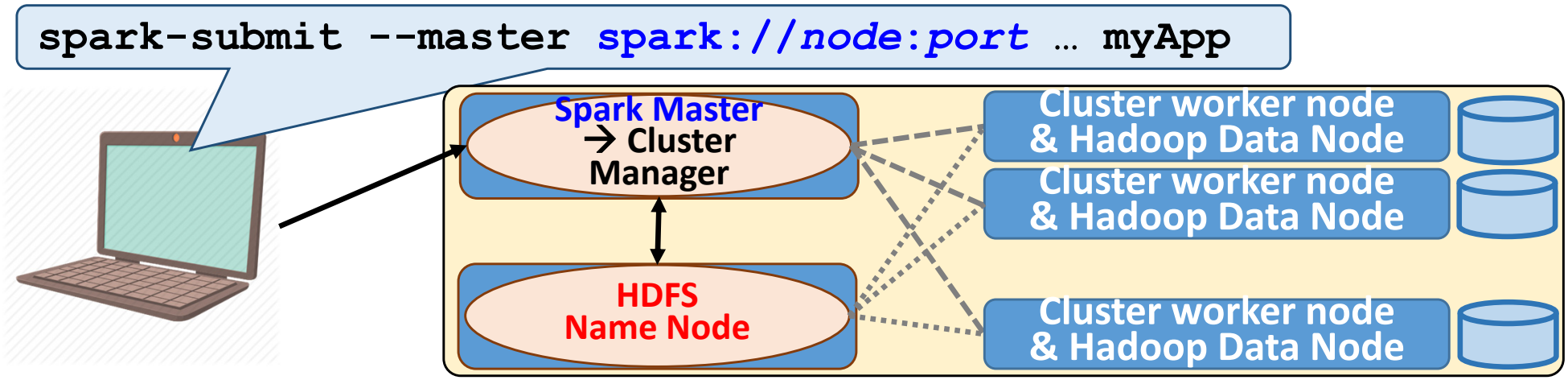
or:

- The *Spark Worker nodes* are the *Data nodes*

On the Spark + HDFS cluster of the DCE:

the *Hadoop Data nodes* host the *Spark Executors*

Spark exécution on cluster (standalone mode)



Spark cluster configuration:

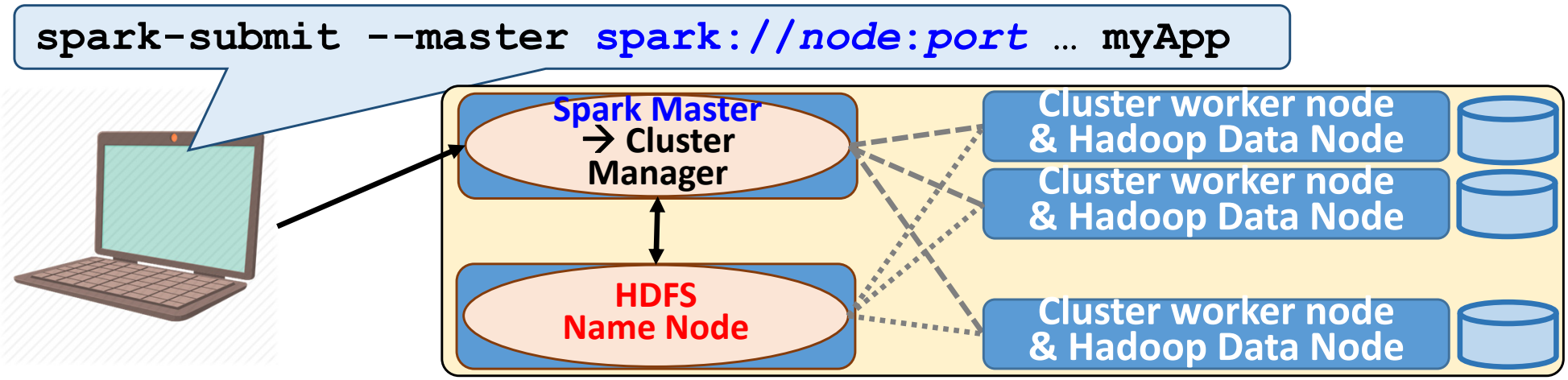
To get a global *data-computations locality*:

- *The Spark Worker nodes* are connected to the *Data nodes* through a (very) fast network …
- …or the *Spark Worker nodes* are the *Data nodes*

On a Spark + HDFS cluster with Spark standalone mode:

- *Spark Executors can be launched on the Hadoop Data nodes*
- But not precisely on the *Data nodes* storing the data used!

Spark exécution on cluster (standalone mode)



Strenght and weakness of standalone mode:

- Nothing more to install (all is included in Spark)
- Easy to configure
- Can interface and share the nodes with HDFS
- Can run different jobs concurrently
- Can not share the cluster with non-Spark applications
- Can not launch Spark Executors only on the right data nodes
- Limited scheduling mechanism (unique queue)

Spark applicaion deployment