Apache Spark

Scalable Data Processing

gradiant
Connectivity · Intelligence · Security
for your business

What's big data?

- extremely large datasets that are hard to deal using relational databases
- requires different aproaches: techniques, tools and architectures
- big data generates values from storage and processing of very large quantities of digital information that cannot be analyzed with traditional techniques

What's big data, again?

- Traditionally, computation has been processor bound
- but as data increase becomes the bottleneck

Early solution: bigger computers

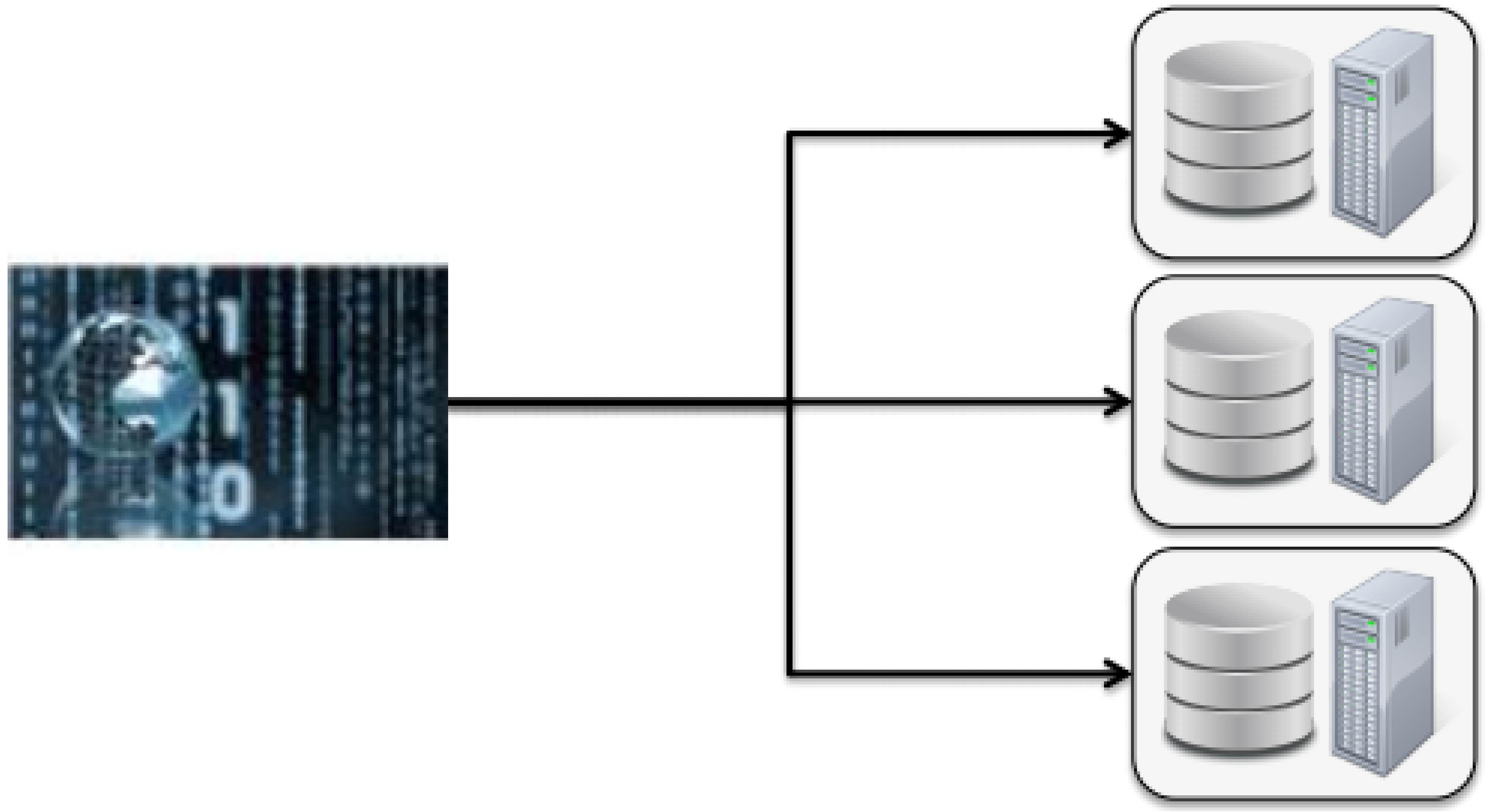
- faster processor, more memory
- but even this couldn't keep up

Distributed systems

The better solution, more machines

Examples:

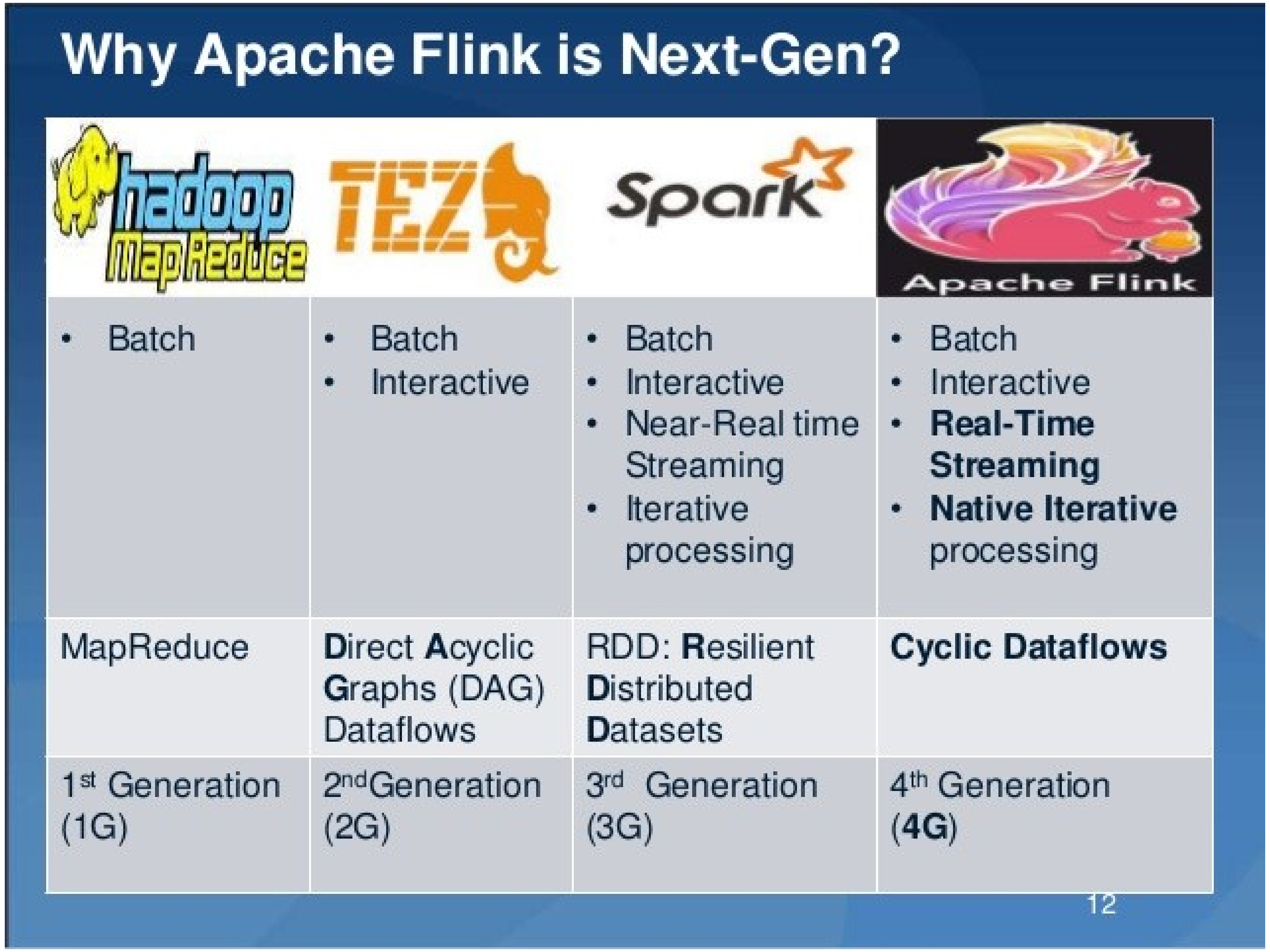
- google
- hadoop
- tez
- spark
- flink



Challenges with distributed systems

- Programming complexity
  - Keeping data and processes in sync
- Finite bandwidth
- Partial failures

Distributed systems: evolution



Why Apache Flink is Next-Gen?

hadoop Map Reduce	TEZ	Spark	Apache Flink
• Batch	• Batch • Interactive	• Batch • Interactive • Near-Real time Streaming • Iterative processing	• Batch • Interactive • **Real-Time Streaming** • **Native Iterative** processing
MapReduce	**D**irect **A**cyclic **G**raphs (DAG) Dataflows	RDD: **R**esilient **D**istributed **D**atasets	**Cyclic Dataflows**
1st Generation (1G)	2ndGeneration (2G)	3rd Generation (3G)	4th Generation (**4G**)

Apache spark

Apache Spark is a fast, general engine for large-scale data processing on a cluster

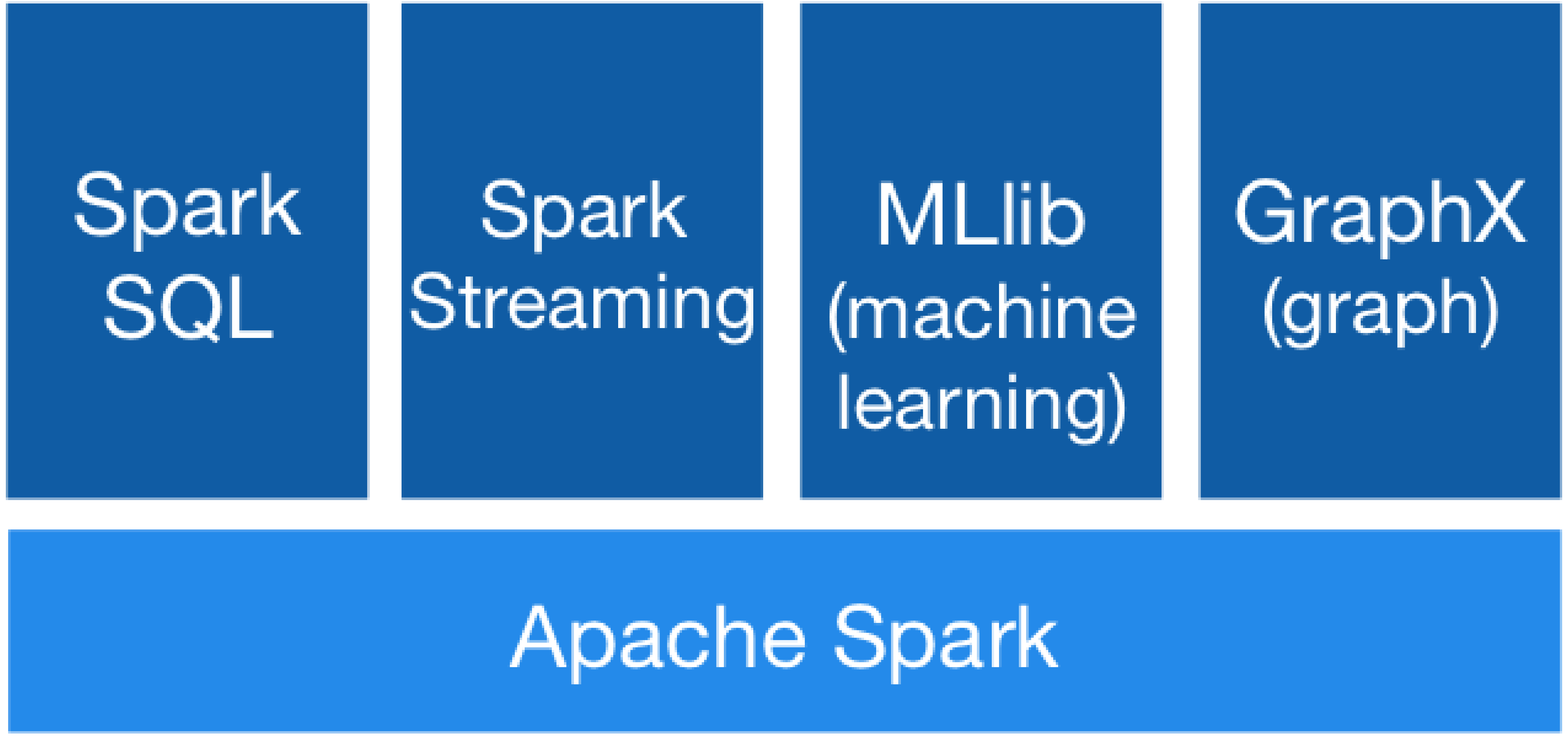
- Originally developed in Berkeley
- The creators founded Databricks to commercialize Spark
- Open source Apache project
  - committer for Yahoo, Databricks, Berkeley, Intel, Cloudera…
  - one of the most active and fastest-growing Apache projects

Advantages of spark

- High level programming framework
- Cluster computing
- Distributed storage
- Data in memory
- Provides fault tolerance
- Adding nodes adds capacity proportionally

The spark stack

- Core Spark provides the fundamental Spark abstraction: Resilient Distributed Datasets (RDDs)
- Spark SQL works with structured data
- MLlib supports scalable machine learning
- Spark Streaming applications process data in real time
- GraphX works with graphs and graph-parallel computation



Spark SQL

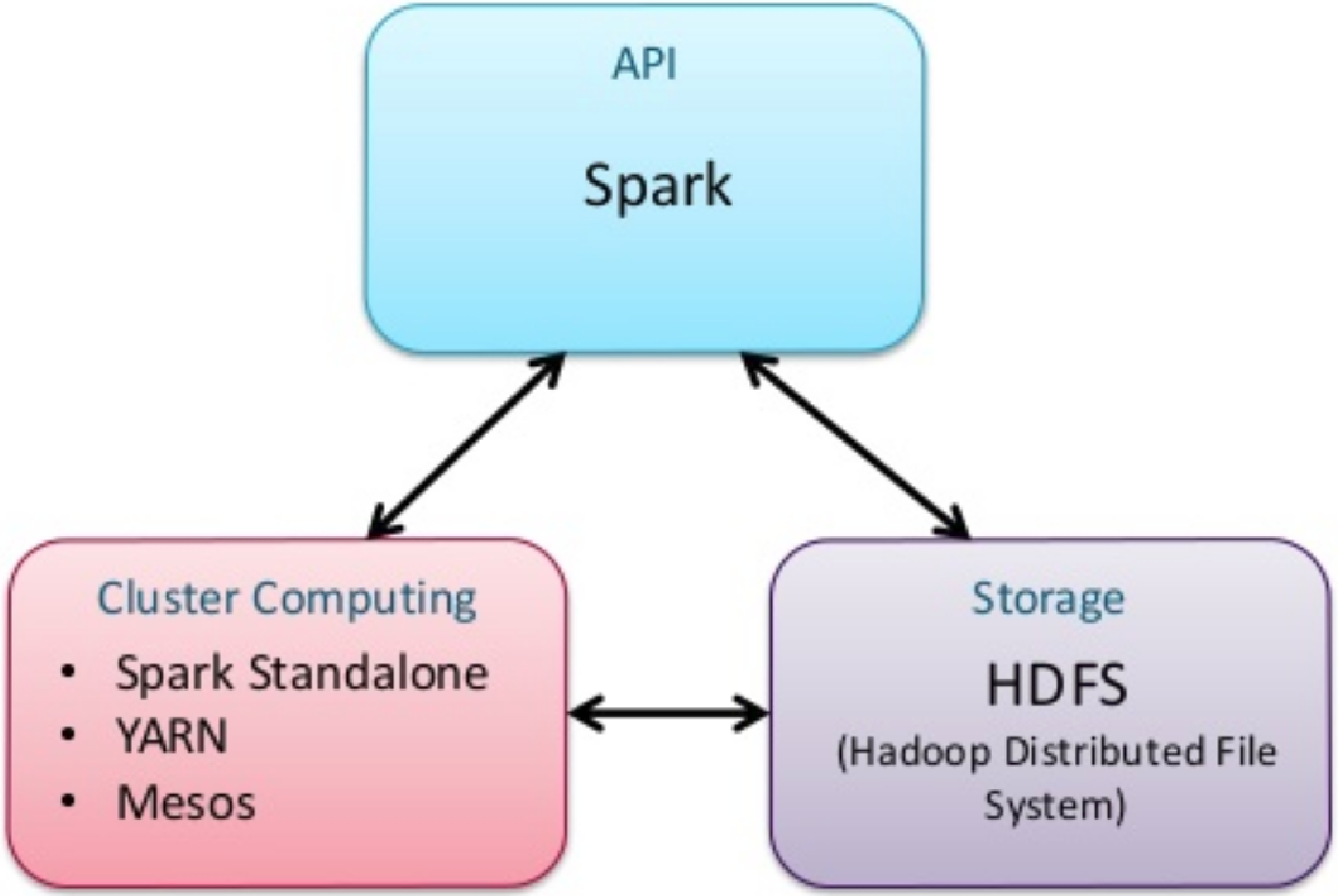
Spark Streaming

MLlib (machine learning)

GraphX (graph)

Apache Spark

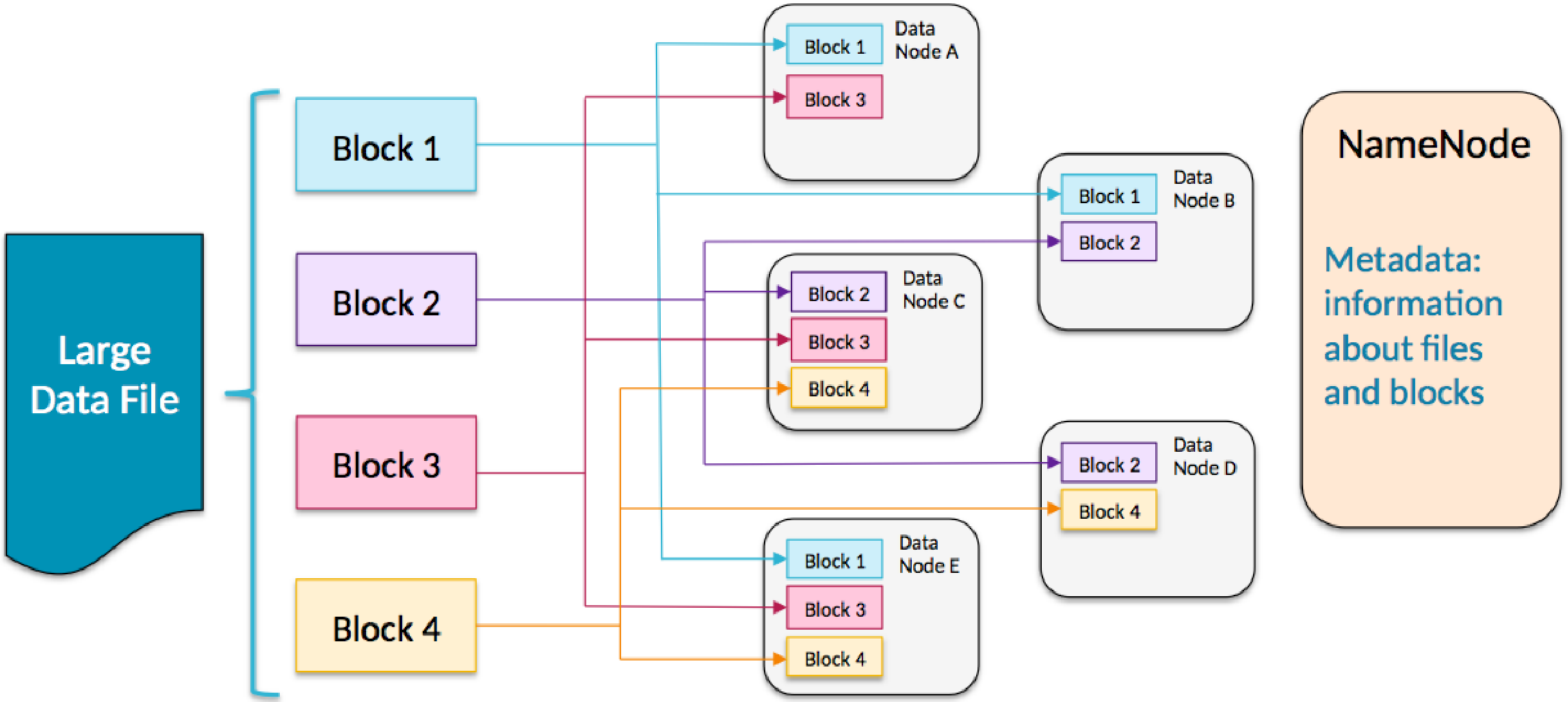
Distributed Processing with the Spark Framework



Architecture HDFS (1)

- HDFS is a file system written in Java. Based on Google File System
- Provides redundant storage for massive amounts of data
- Data files are split into blocks (default 128MB)
- The actual blocks are stored on cluster worker nodes (DataNodes)
- Each block is replicated
- A cluster master node runs the HDFS Name Node service (NameNode)

Architecture HDFS (2)

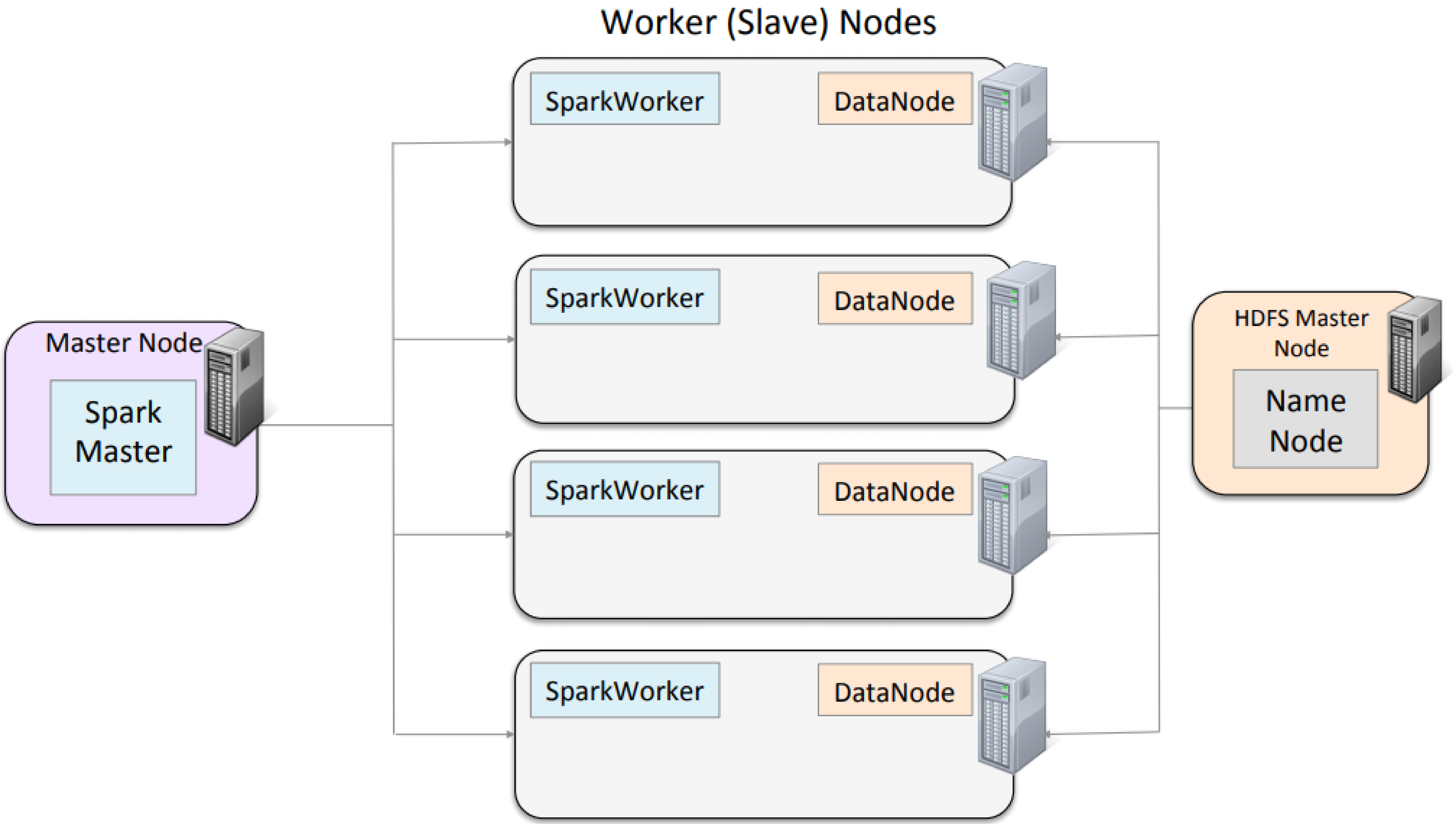


Architecture Spark Standalone Cluster (1)

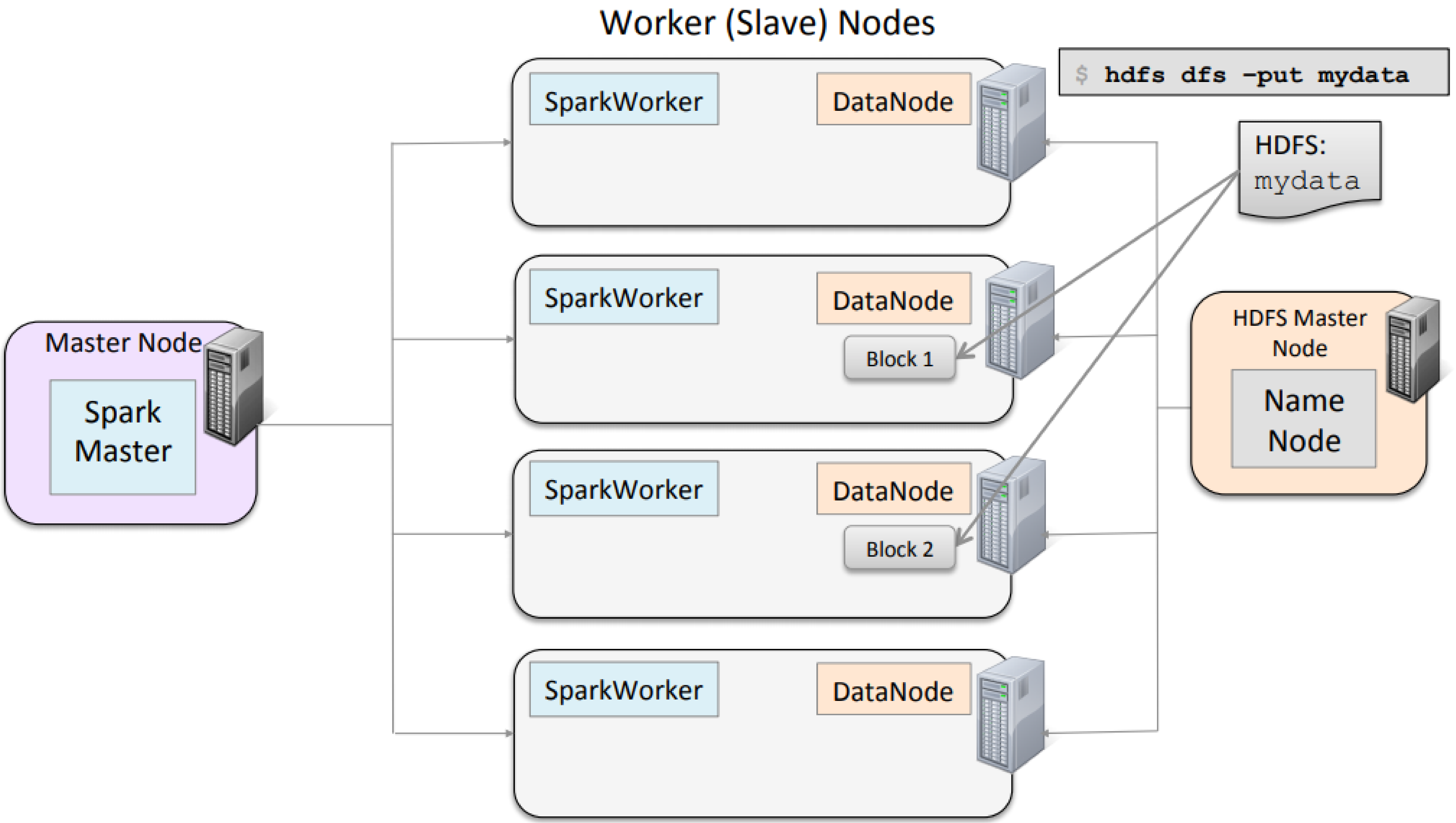
Spark Standalone daemons

- Spark Master: One per cluster. Manages applications, distributes individual tasks to Spark Workers
- Spark Worker: One per worker node. Starts and monitors Executors for applications

Architecture Spark Standalone Cluster (2)

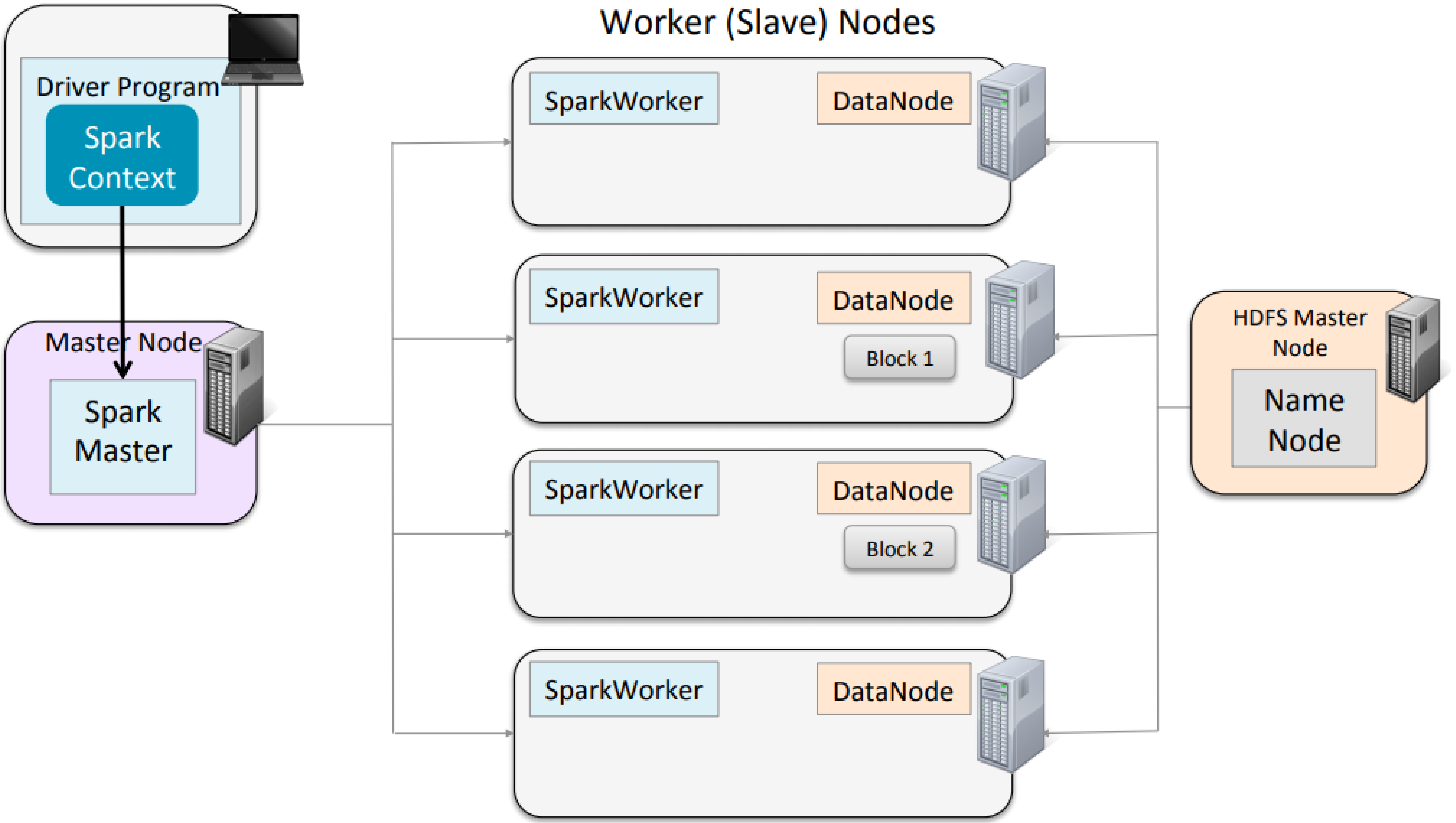


Architecture Spark Standalone Cluster (3)

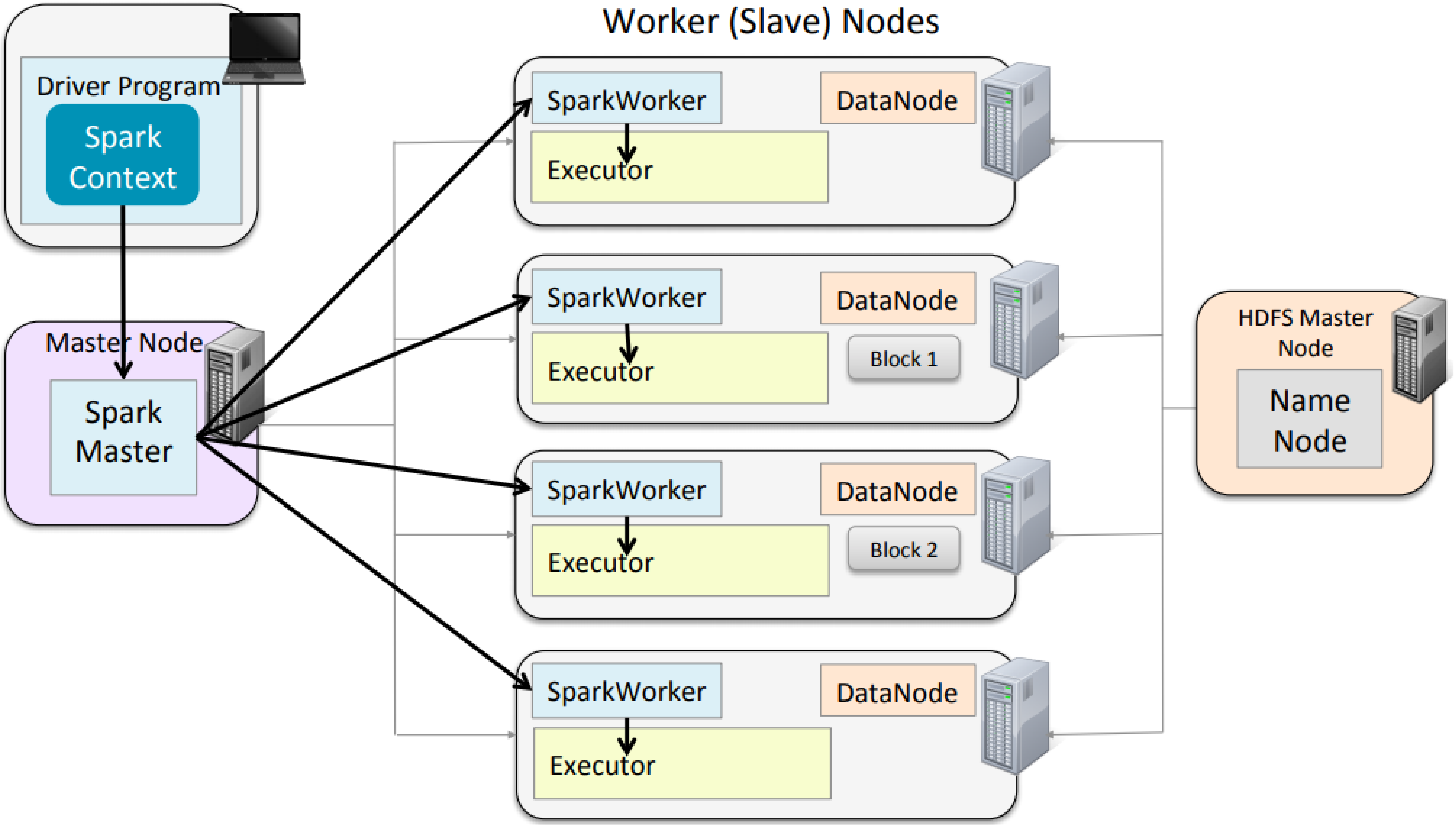


```
$ hdfs dfs -put mydata
```

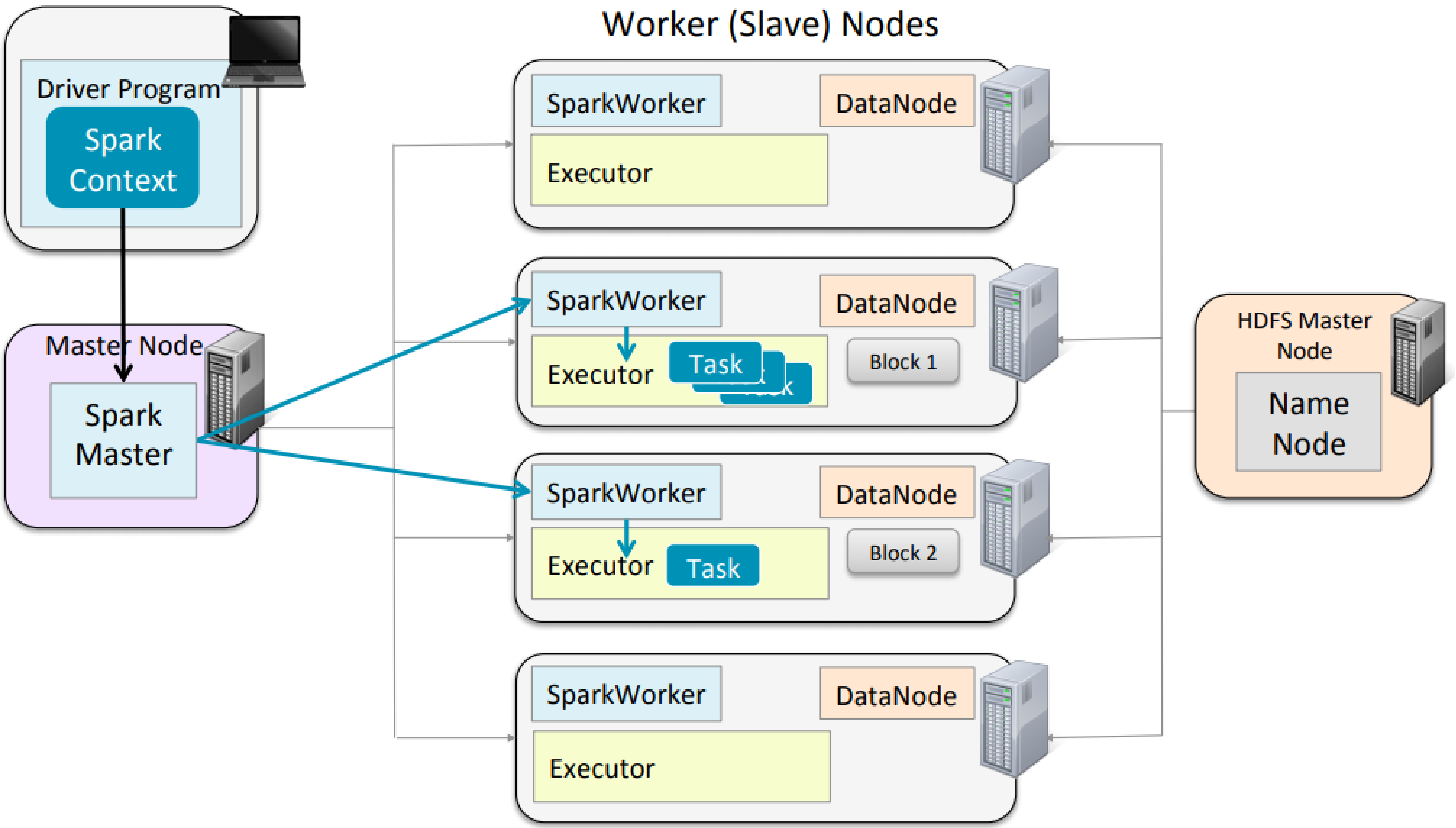
Architecture Spark Standalone Cluster (4)



Architecture Spark Standalone Cluster (5)



Architecture Spark Standalone Cluster (6)



Architecture Yarn Cluster (1)

YARN Daemons

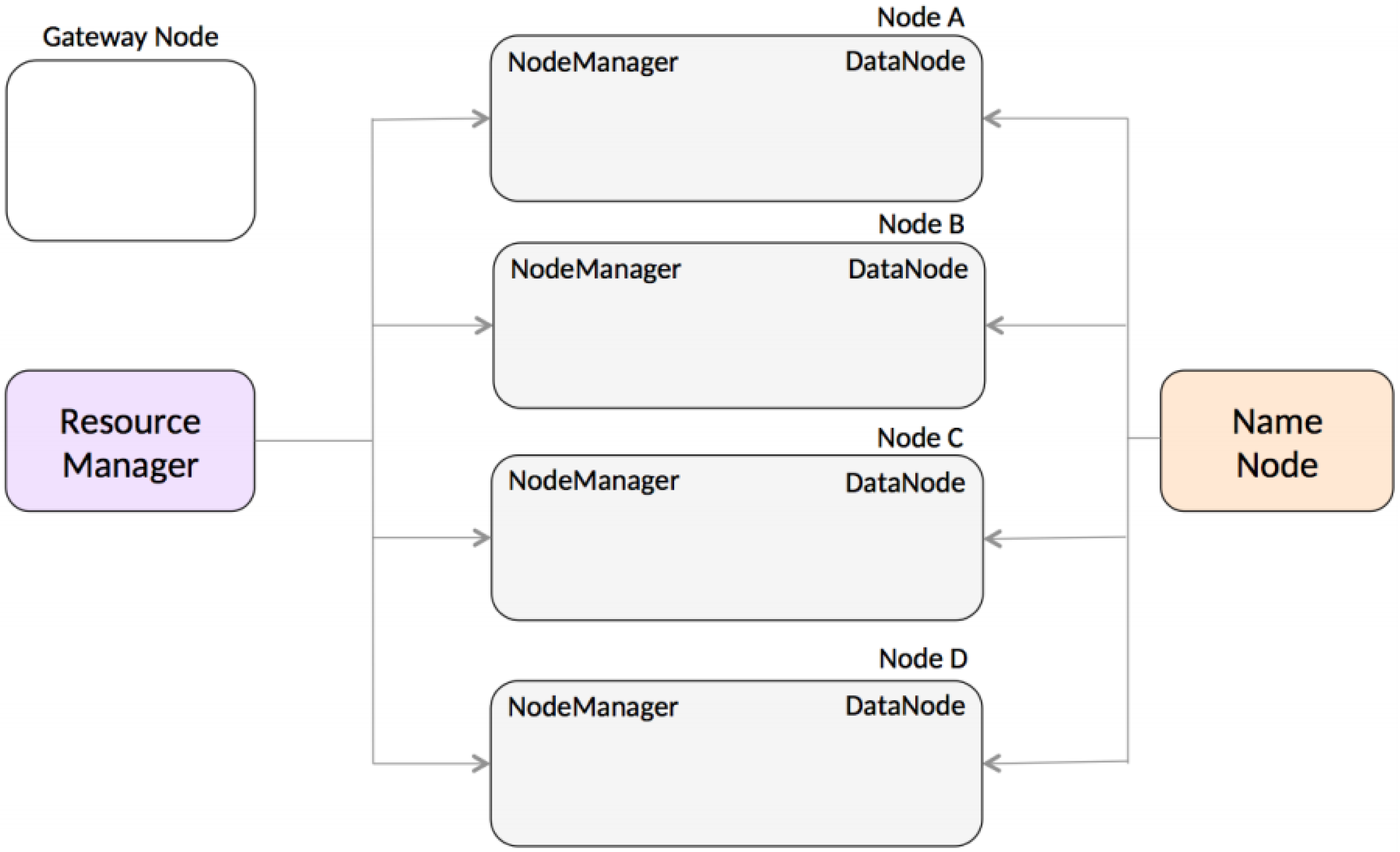
- ResourceManager (RM)
  - Runs on master node
  - Global resource scheduler
  - Arbitrates system resources between competing applications
  - Has a pluggable scheduler to support different algorithms (such as Capacity or Fair Scheduler)
- NodeManager (NM)
  - Runs on worker nodes
  - Communicates with RM
  - Manages node resources
  - Launches containers

Architecture Yarn Cluster (2)

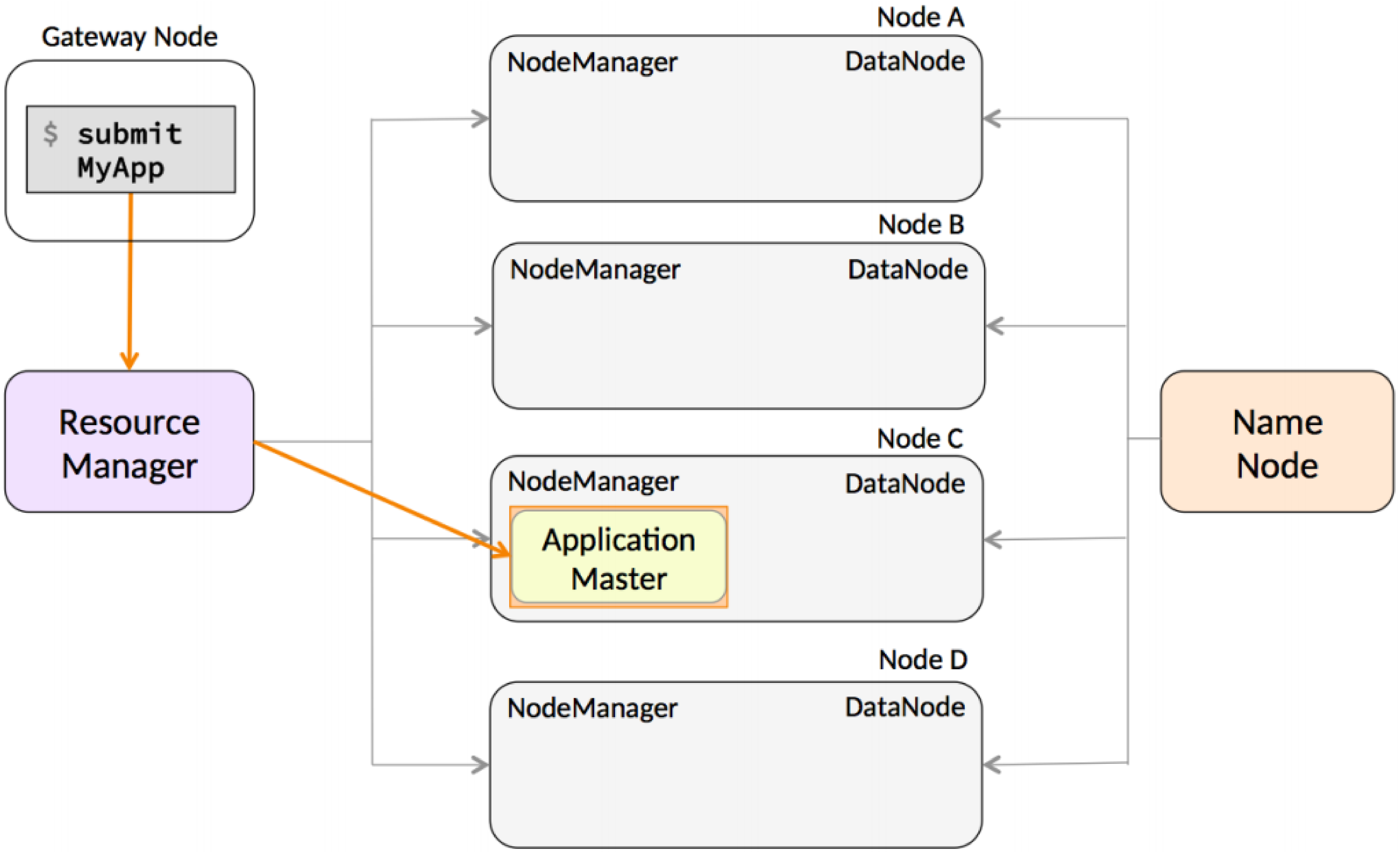
- Containers
  - Containers allocate a certain amount of resources (memory, CPU cores) on a worker node
  - Applications run in one or more containers
  - Applications request containers from RM

- ApplicationMaster (AM)
  - One per application
  - Framework/application specific
  - Runs in a container
  - Requests more containers to run application tasks

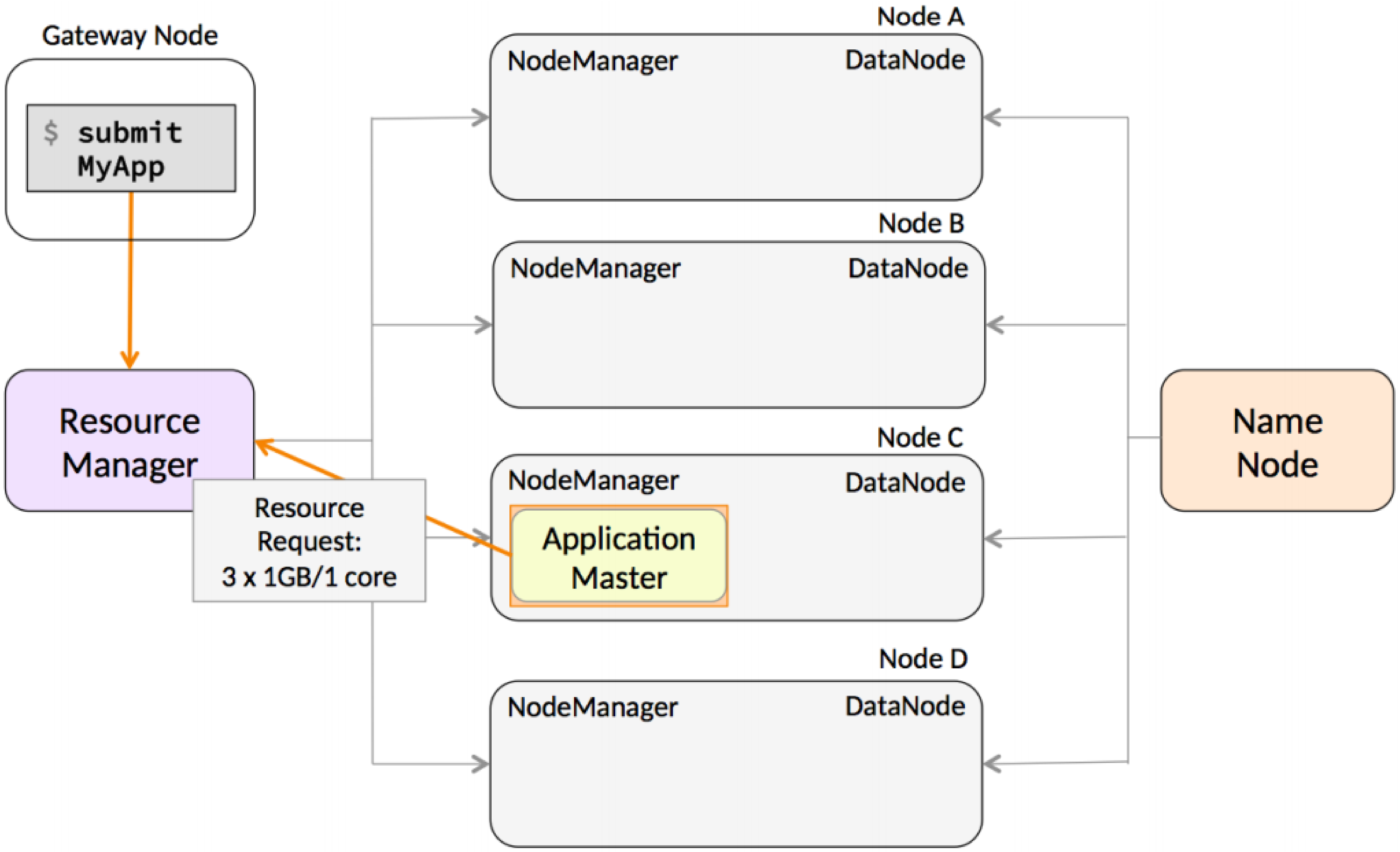
Architecture Yarn Cluster (2)



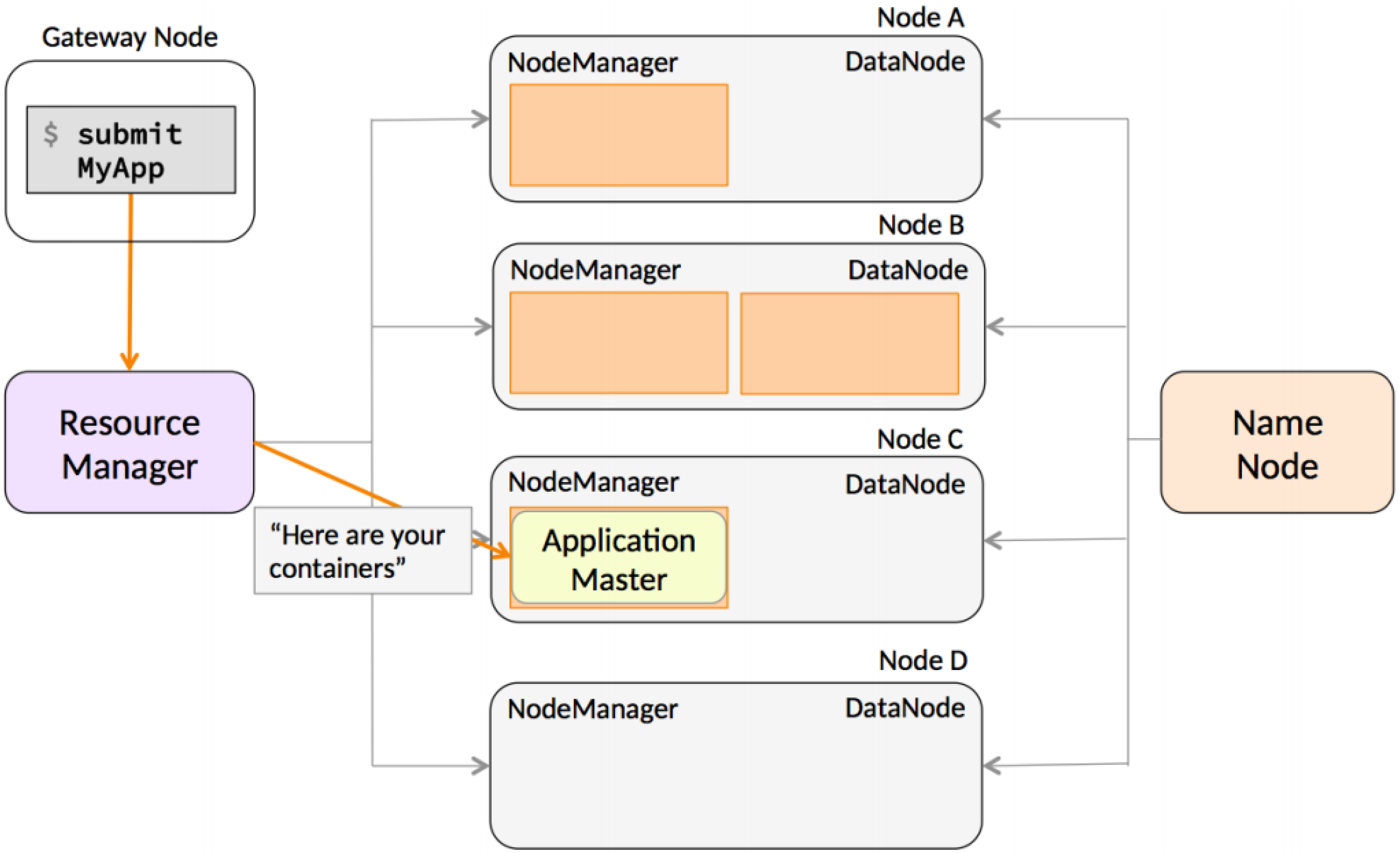
Architecture Yarn Cluster (3)



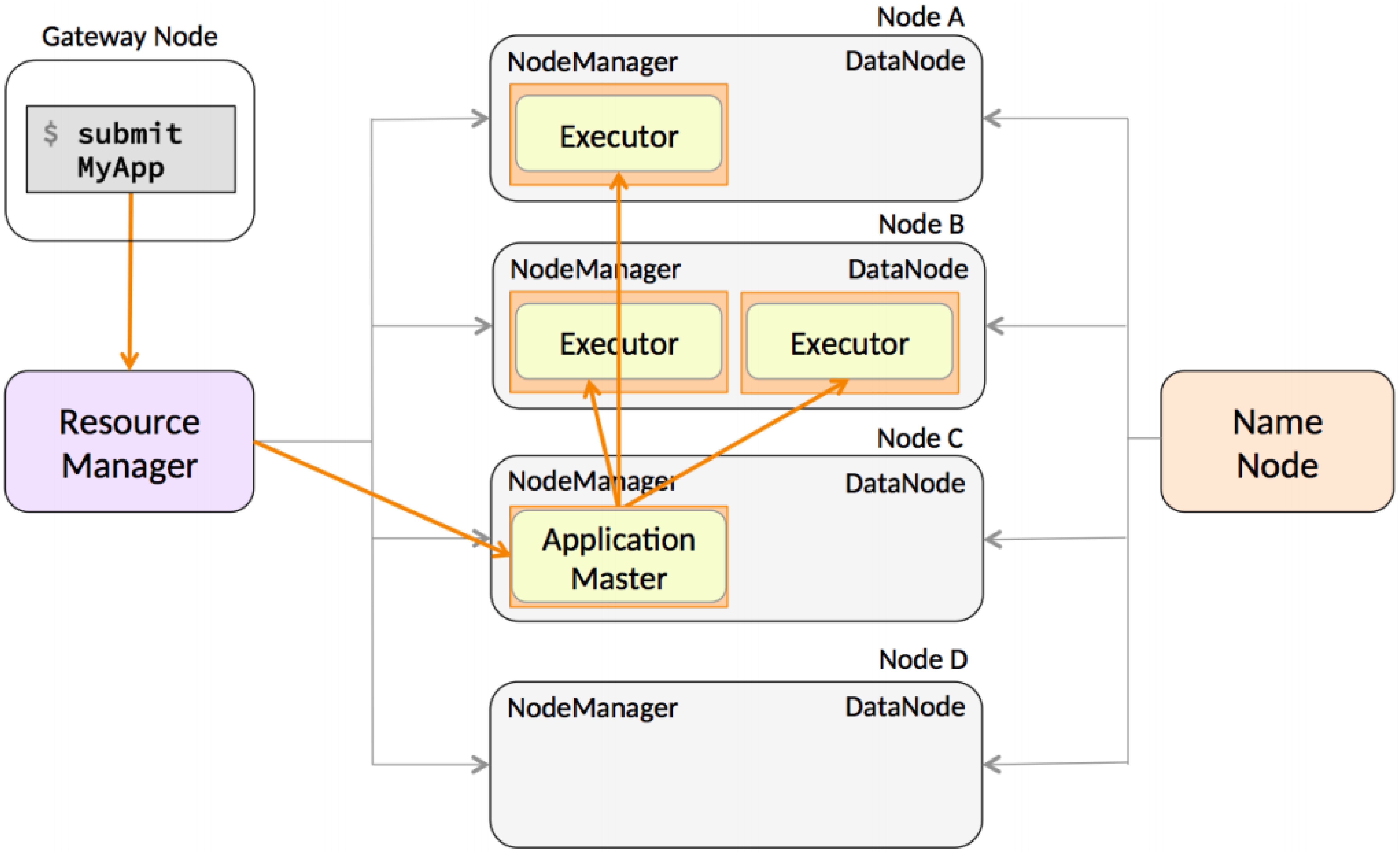
Architecture Yarn Cluster (4)



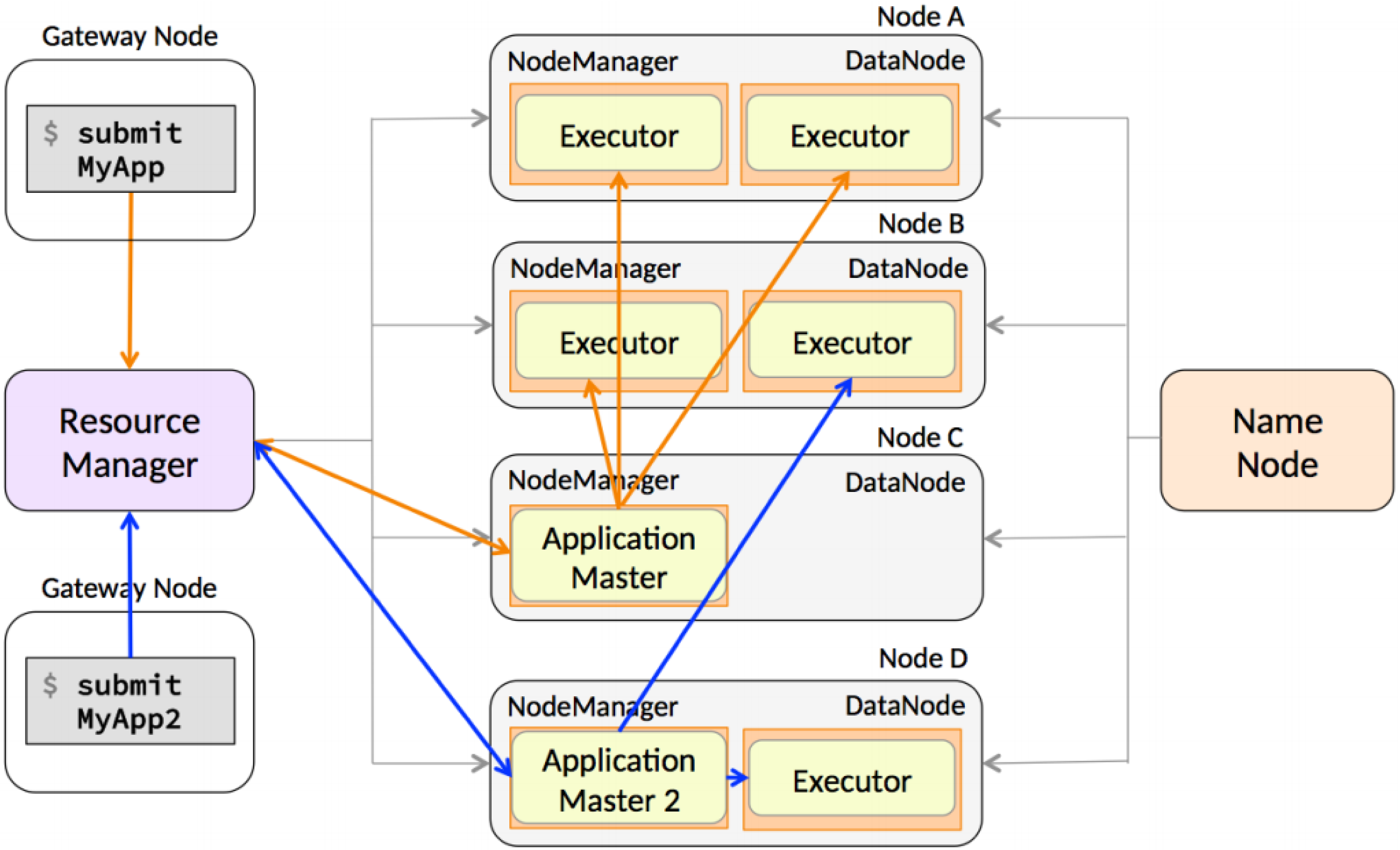
Architecture Yarn Cluster (5)



Architecture Yarn Cluster (6)



Architecture Yarn Cluster (7)

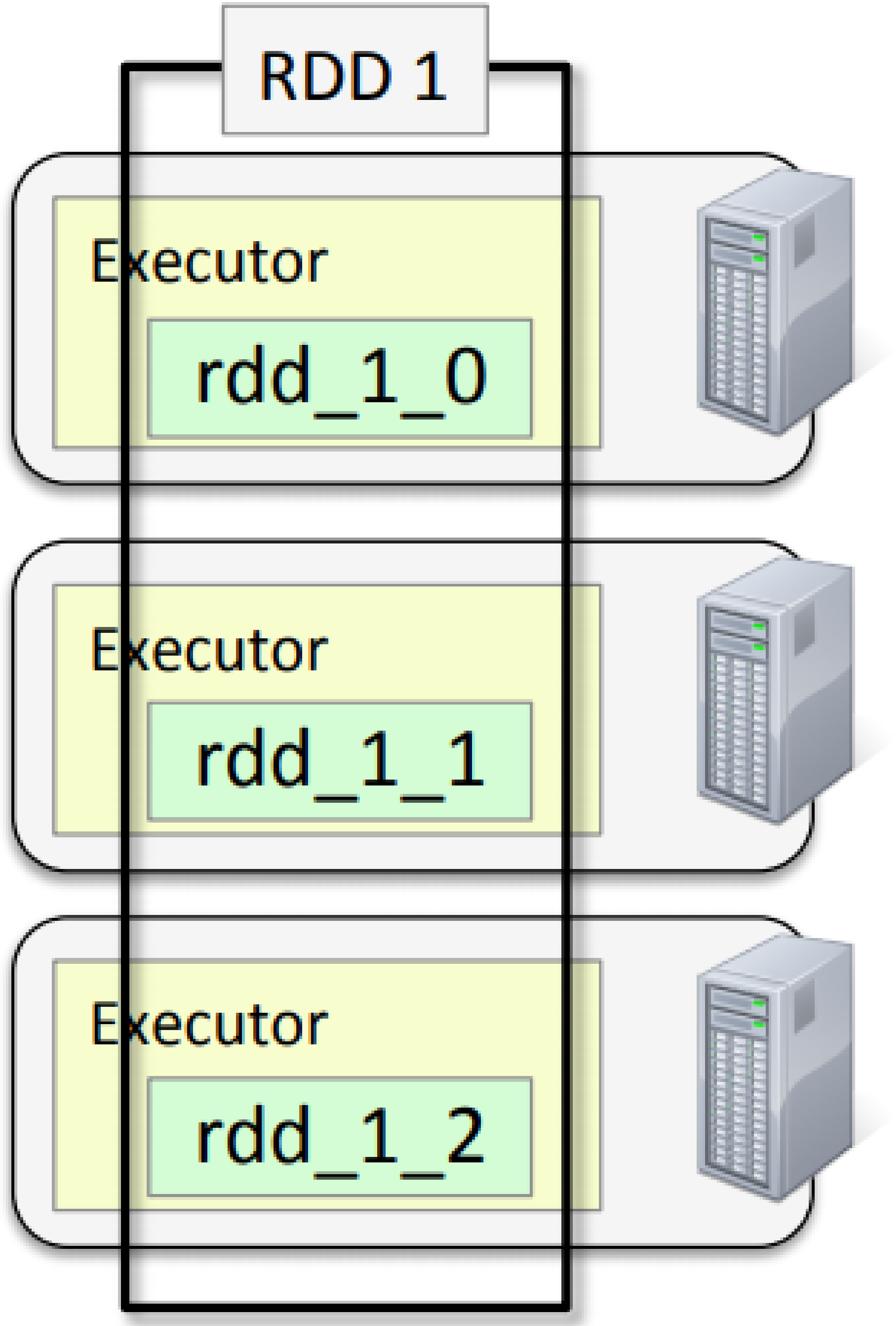


Data model: Resilient Distributed Datasets (RDDs)(1)

- RDDs are part of core Spark
- Resilient Distributed Dataset (RDD)
  - Resilient: If data in memory is lost, it can be recreated
  - Distributed: Processed across the cluster
  - Dataset: Initial data can come from a source such as a file, or it can be created programmatically
- RDDs are unstructured
  - No schema defining columns and rows
  - Not table-like; cannot be queried using SQL-like transformations such as where and select
  - RDD transformations use lambda functions

Data model: Resilient Distributed Datasets (RDDs)(1)

- Resilient Distributed Datasets
  - Data is partitioned across worker nodes
- Partitioning is done automatically by Spark
  - Optionally you can control how many partitions are created



Data model: DataFrame

- DataFrames represent structured data in a tabular form
    - DataFrames model data similar to tables in an RDBMS
    - DataFrames consist of a collection of loosely typed Row objects
    - Rows are organized into columns described by a schema
- DataFrames contain an ordered collection of Row objects
    - Rows contain an ordered collection of values
    - Row values can be basic types (such as integers, strings, and floats) or collections of those types (such as arrays and lists)
    - A schema maps column names and types to the values in a row

Data model: DataFrame

- DataSet is distributed collection of strongly-typed objects
    - Primitive types such as Int or String
    - Complex types such as arrays and lists containing supported types
- Mapped to a relational schema
  - The schema is defined by an encoder
  - The schema maps object properties to typed columns
- In Scala, DataFrame is an alias for a Dataset containing Row objects
- DataFrames and Datasets represent different types of data
  - DataFrames (Datasets of Row objects) represent tabular data
  - – Datasets represent typed, object-oriented data

Data model: recap

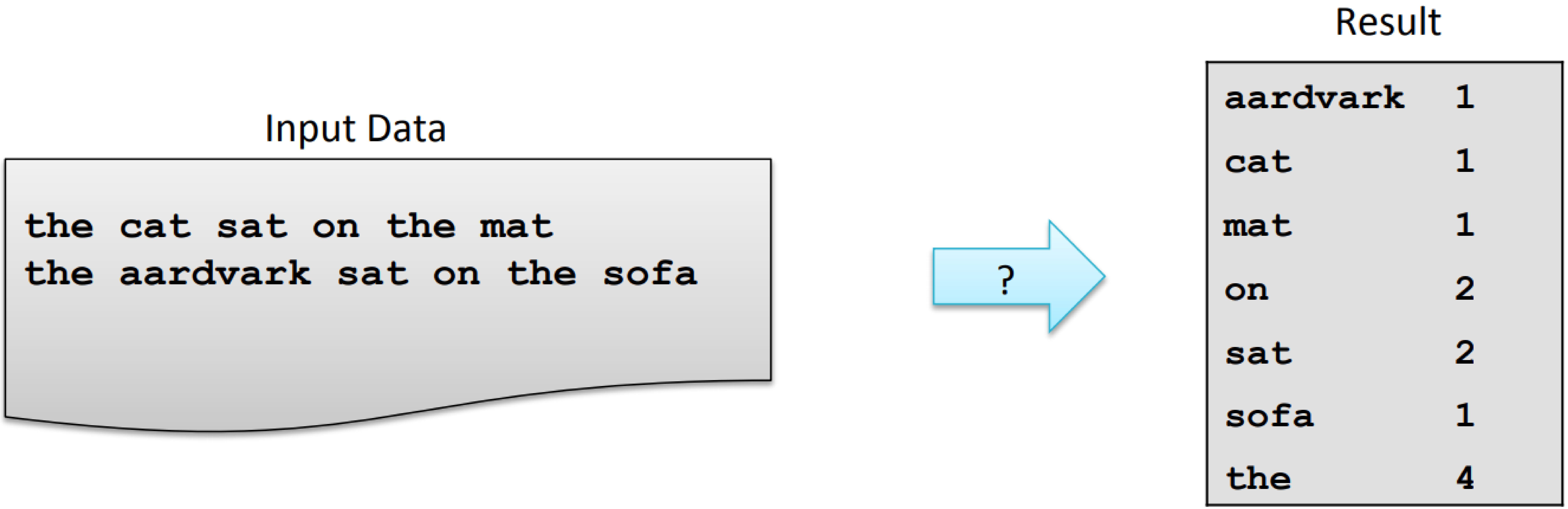
- RDD Is building block of spark. No matter which abstraction Dataframe or Dataset we use, internally final computation is done on RDDs
- offers huge performance improvement over RDDs
  - Custom Memory management
  - Optimized Execution Plans
  - but Lack of Type Safety
- DataSet is is an extension to Dataframe API
  - comes with OOPs style and developer friendly compile time safety

Data model: case uses

- RDD
  - low-level transformation
  - unstructured data

- DataFrame or DataSets
  - want rich semantics or high-level abstractions
  - need high-level expressións
  - want higher degree of type-safety at compile time

Example: Wordcount(1)



Input Data

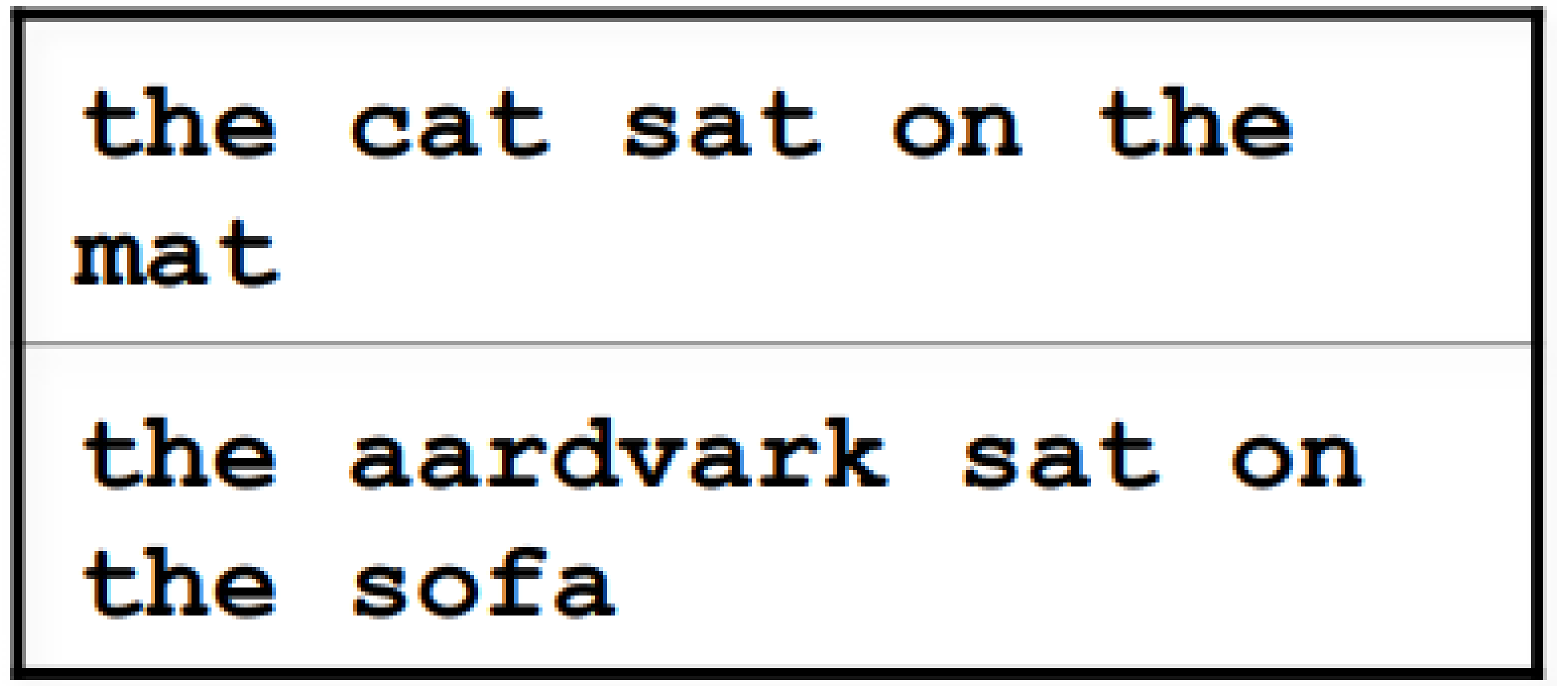
```
the cat sat on the mat
the aardvark sat on the sofa
```

Result

aardvark	1
cat	1
mat	1
on	2
sat	2
sofa	1
the	4

Example: Wordcount(2)

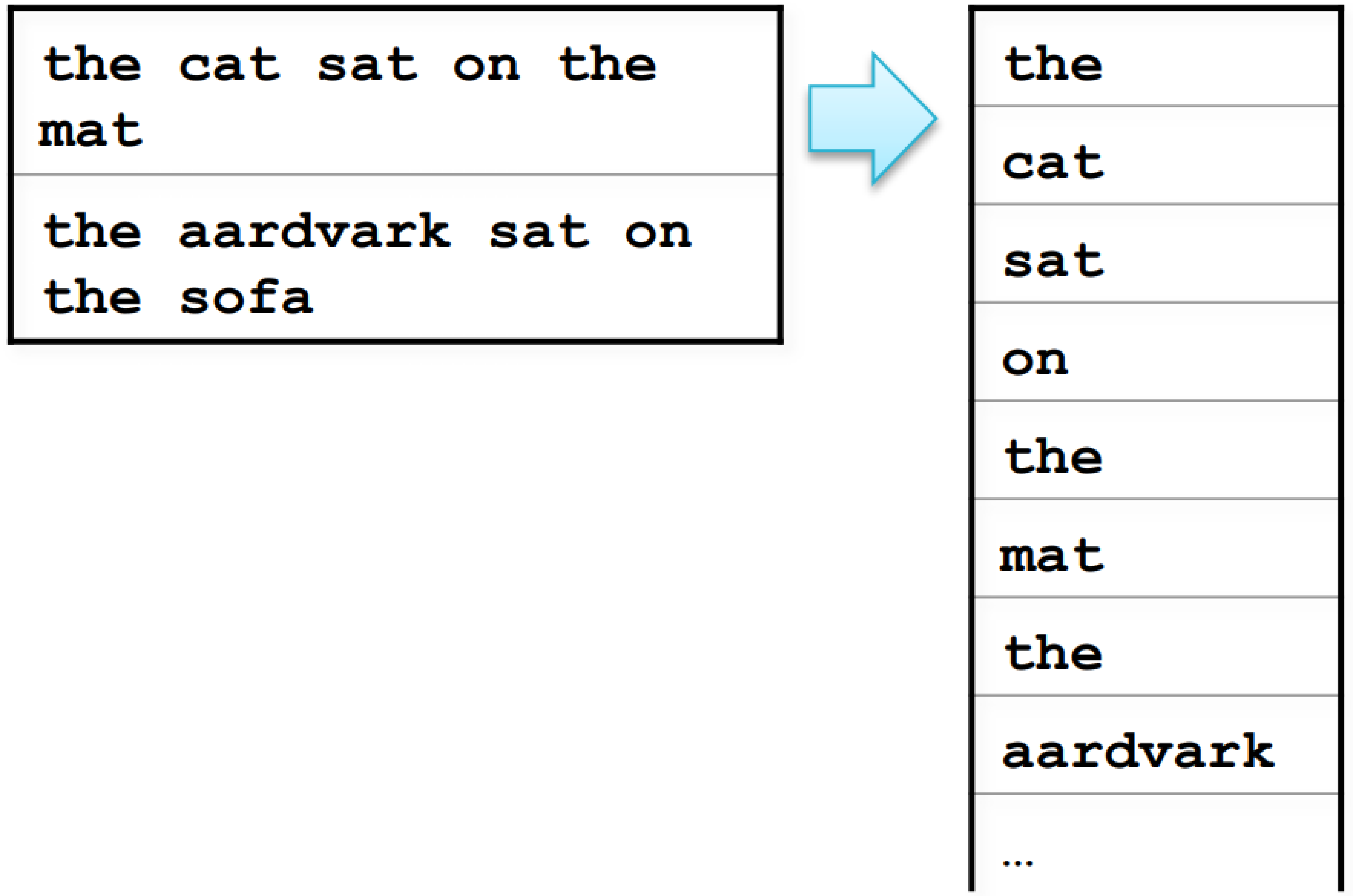
```
text_file = sc.textFile("hdfs://...")
```



the cat sat on the mat
the aardvark sat on the sofa

Example: Wordcount(3)

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" "))
```

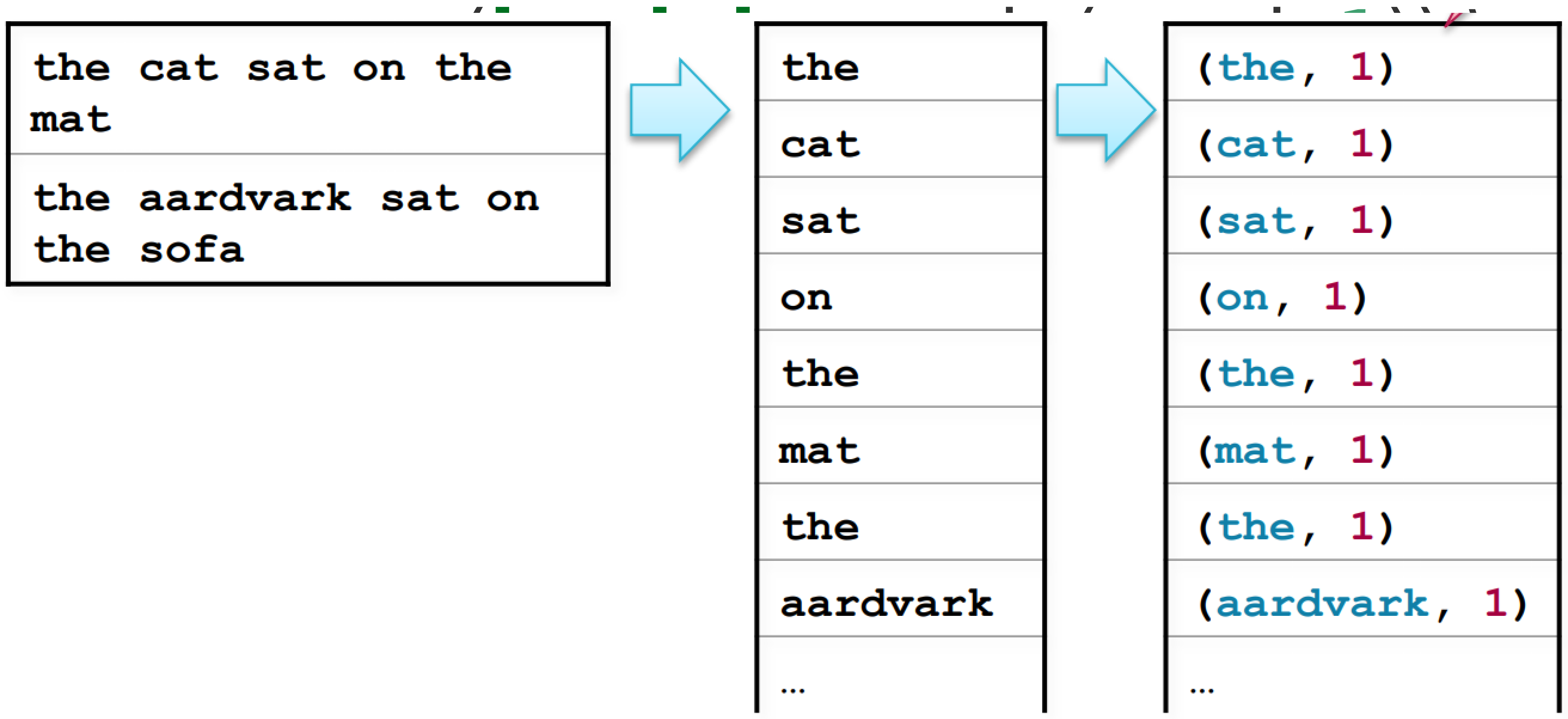


the cat sat on the mat
the aardvark sat on the sofa

the
cat
sat
on
the
mat
the
aardvark
...

Example: Wordcount(4)

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" "))
\
```



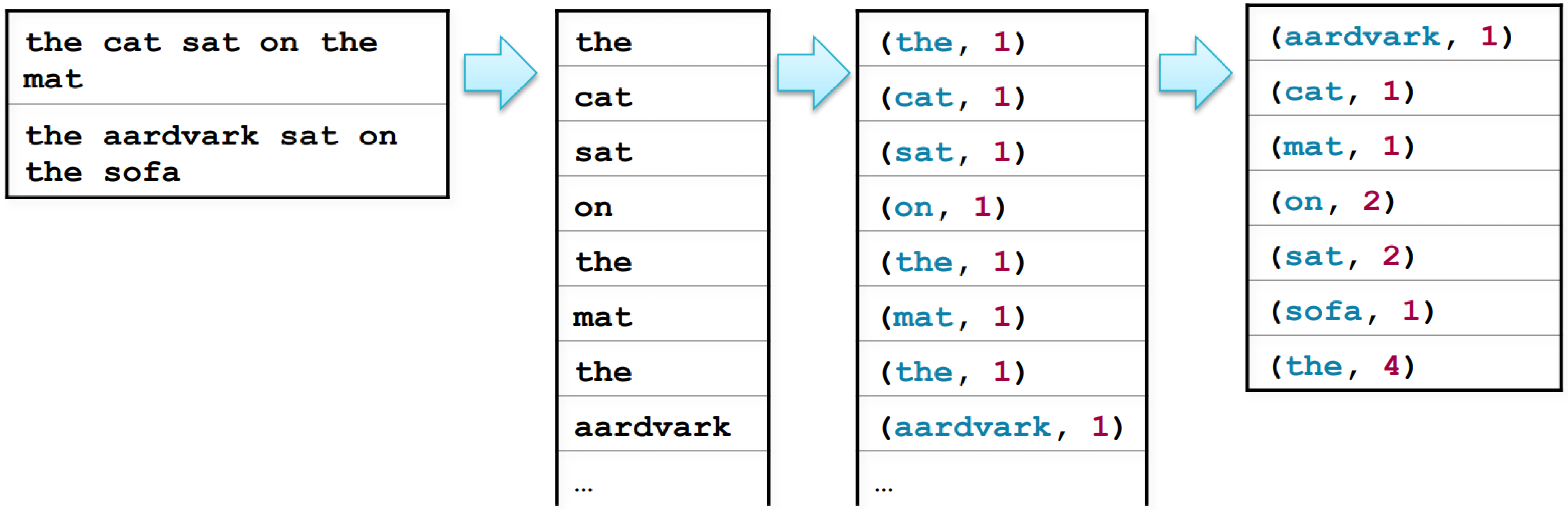
```
the cat sat on the
mat
the aardvark sat on
the sofa
```

```
the
cat
sat
on
the
mat
the
aardvark
…
```

```
(the, 1)
(cat, 1)
(sat, 1)
(on, 1)
(the, 1)
(mat, 1)
(the, 1)
(aardvark, 1)
…
```

Example: Wordcount(5)

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
         .map(lambda word: (word, 1)) \
         .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```



```
the cat sat on the
mat

the aardvark sat on
the sofa
```

```
the
cat
sat
on
the
mat
the
aardvark
...
```

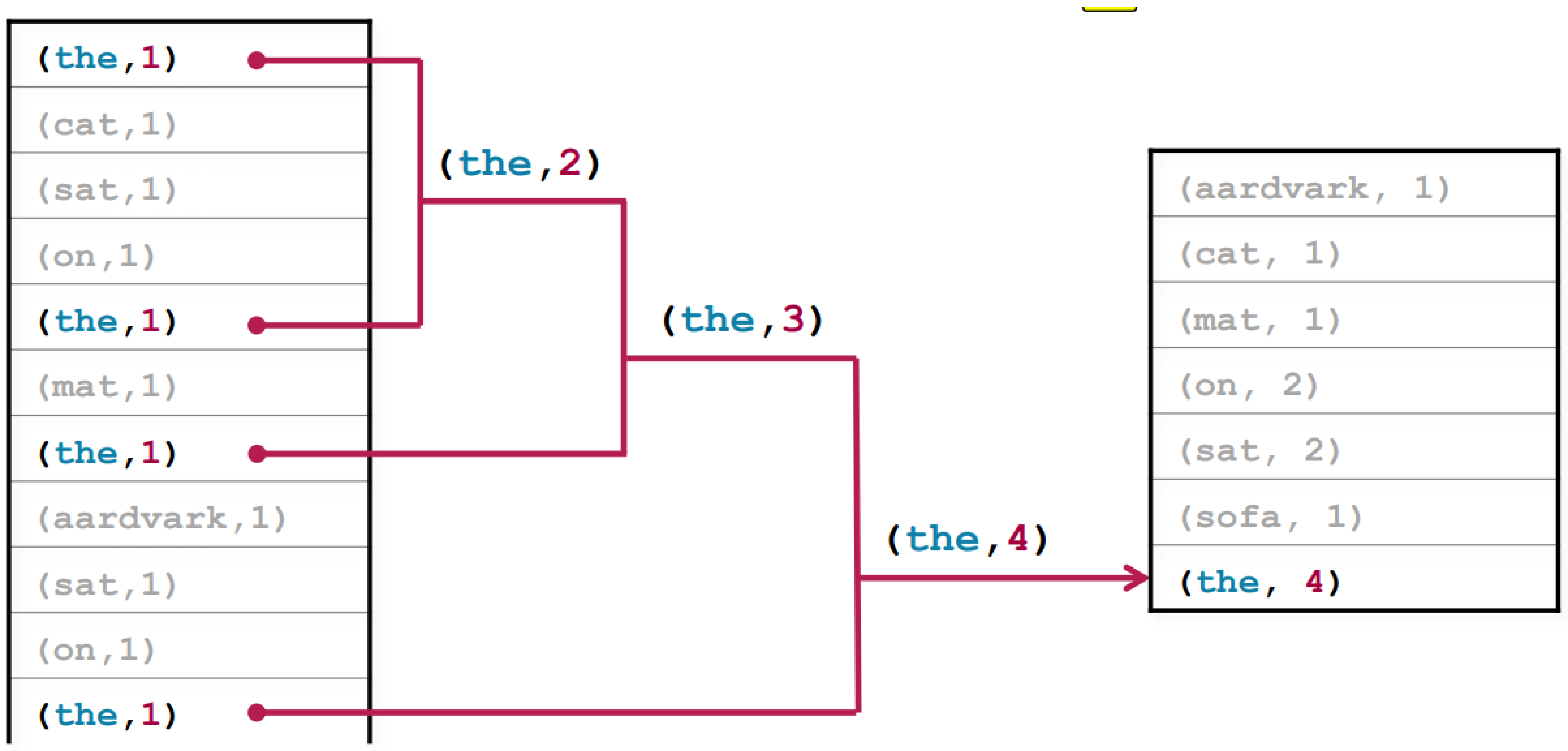
```
(the, 1)
(cat, 1)
(sat, 1)
(on, 1)
(the, 1)
(mat, 1)
(the, 1)
(aardvark, 1)
...
```

```
(aardvark, 1)
(cat, 1)
(mat, 1)
(on, 2)
(sat, 2)
(sofa, 1)
(the, 4)
```

Example: Wordcount(5)

ReduceByKey functions must be

- Binary–combines values from two keys
- Commutative – x+y =y+x
- Associative – (x+y)+z=x+(y+z)



(the,1)
(cat,1)
(sat,1)
(on,1)
(the,1)
(mat,1)
(the,1)
(aardvark,1)
(sat,1)
(on,1)
(the,1)

(the,2)
(the,3)
(the,4)

(aardvark, 1)
(cat, 1)
(mat, 1)
(on, 2)
(sat, 2)
(sofa, 1)
(the, 4)

Spark: use cases

- CERN: predict dataset popularity
- Verizon: feature extraction
- Uber: recommender system
- BBVA: tagging text in money transfers
- Toyota: power customer platforms
- Airbnb: predict demand