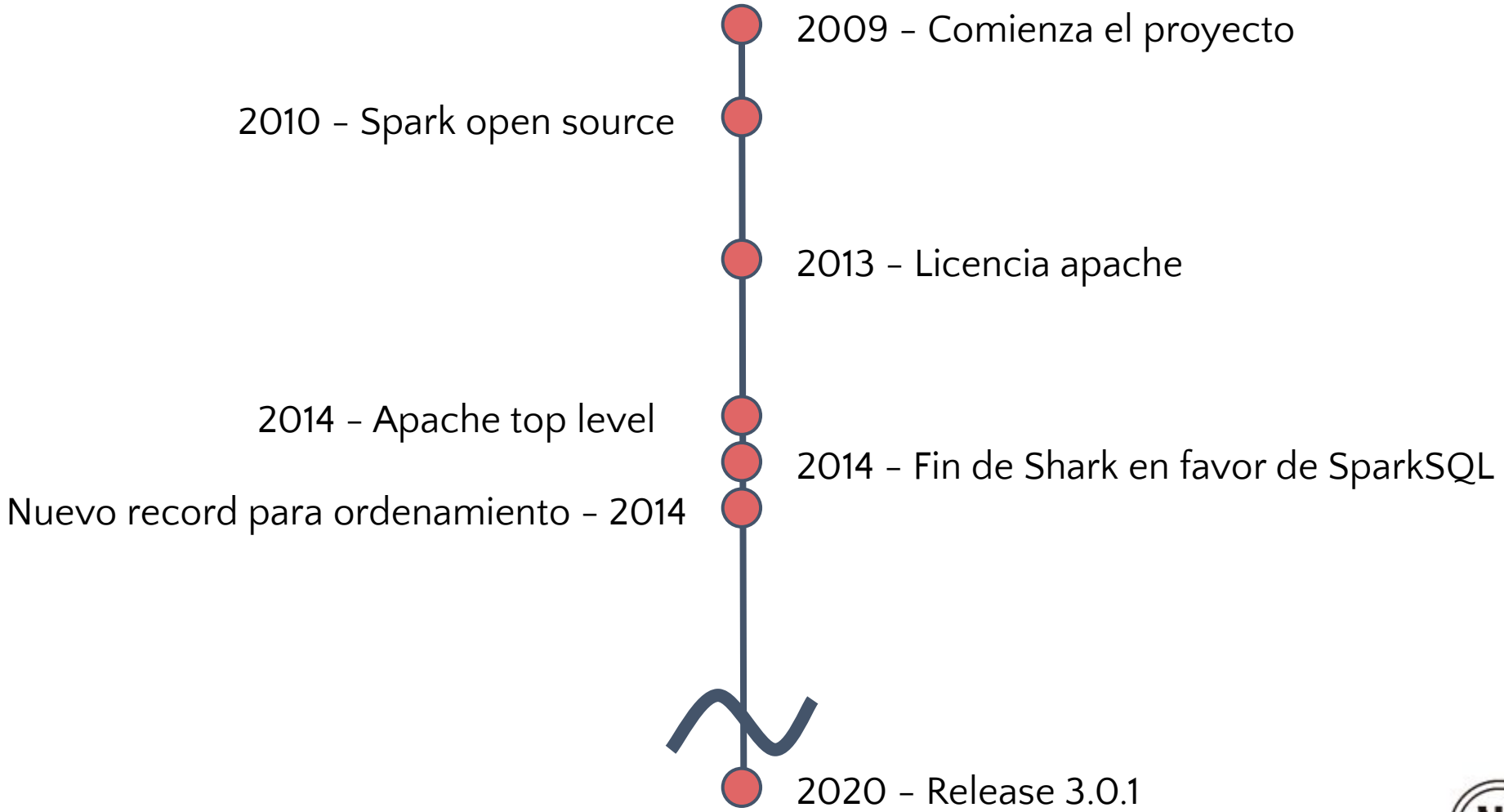
Spark

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

¿Qué es Spark?

- Un **framework** de procesamiento **distribuido** para el análisis de Big Data.
- **OpenSource** originalmente desarrollado en la Universidad de Berkeley en California.
- Provee **análisis de datos en memoria.**
- Diseñado para ejecutar algoritmos **iterativos** y análisis **predictivos**.
- **Altamente compatible** con los medios de almacenamientos en Hadoop.
- Actualmente es el proyecto **más activo de apache** con más de 1000 contribuyentes

Historia

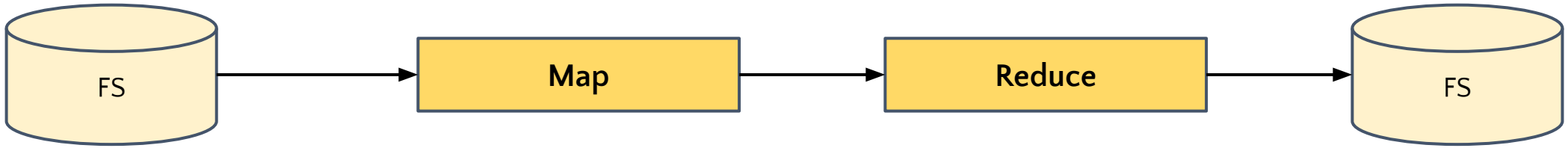


Spark vs MapReduce

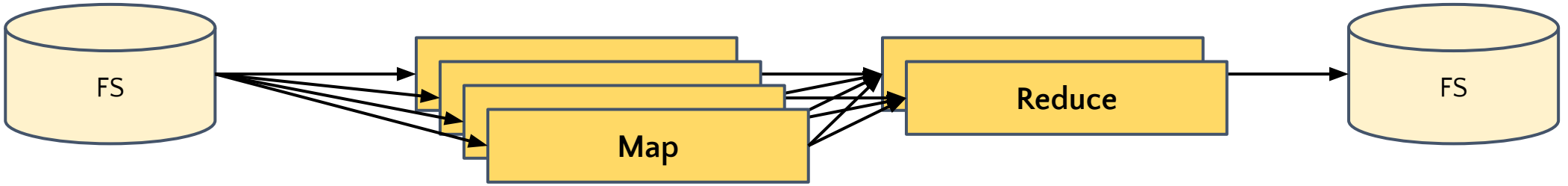
- **Procesamiento rápido de datos**: Procesar en memoria hace a Spark más rápido que MapReduce. 100 veces si los datos están en RAM y 10 veces para datos en disco.
- **Procesamiento Iterativo**: Si el task debe procesar y reprocesar los datos, Spark le gana a MapReduce. Los RDDs permiten que múltiples tareas map se ejecuten en memoria, mientras que MapReduce debe escribir los resultados intermedios a disco.
- **Procesamiento near real-time**: Gracias al rápido procesamiento en memoria.
- **Procesamiento de grafos**: El modelo de Spark es bueno para el procesamiento iterativo, que es típico en el procesamiento de grafos.
- **Machine learning**: Spark provee la librería Mlib, mientras que MapReduce necesita herramientas de terceros.

Spark vs MapReduce

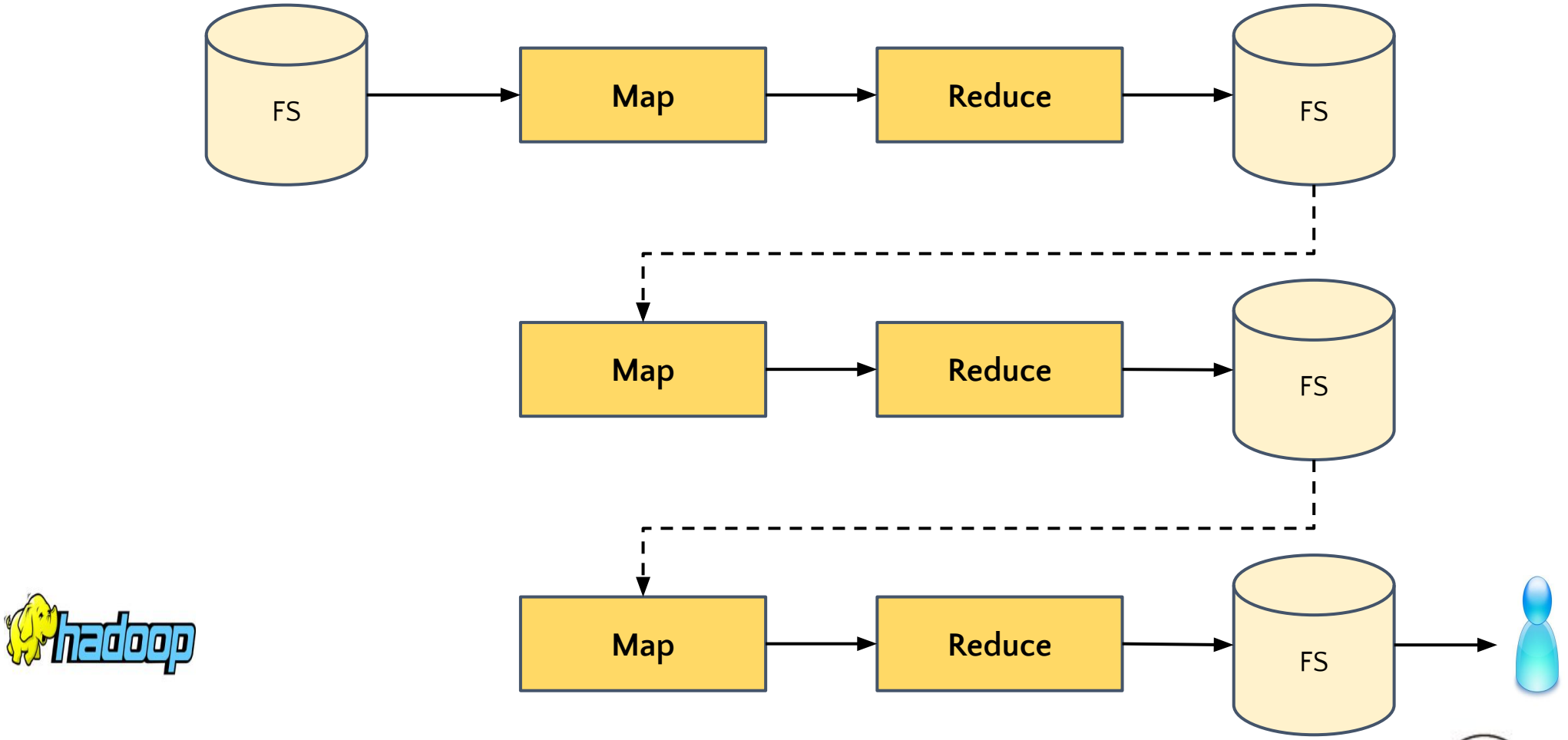
Logicamente:



Fisicamente:



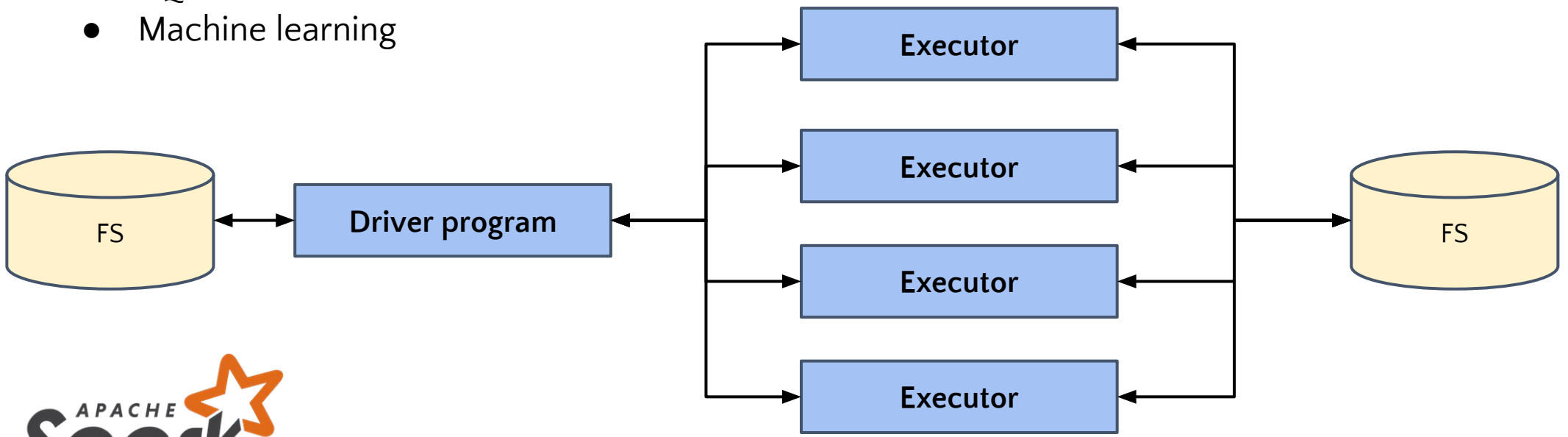
Spark vs MapReduce



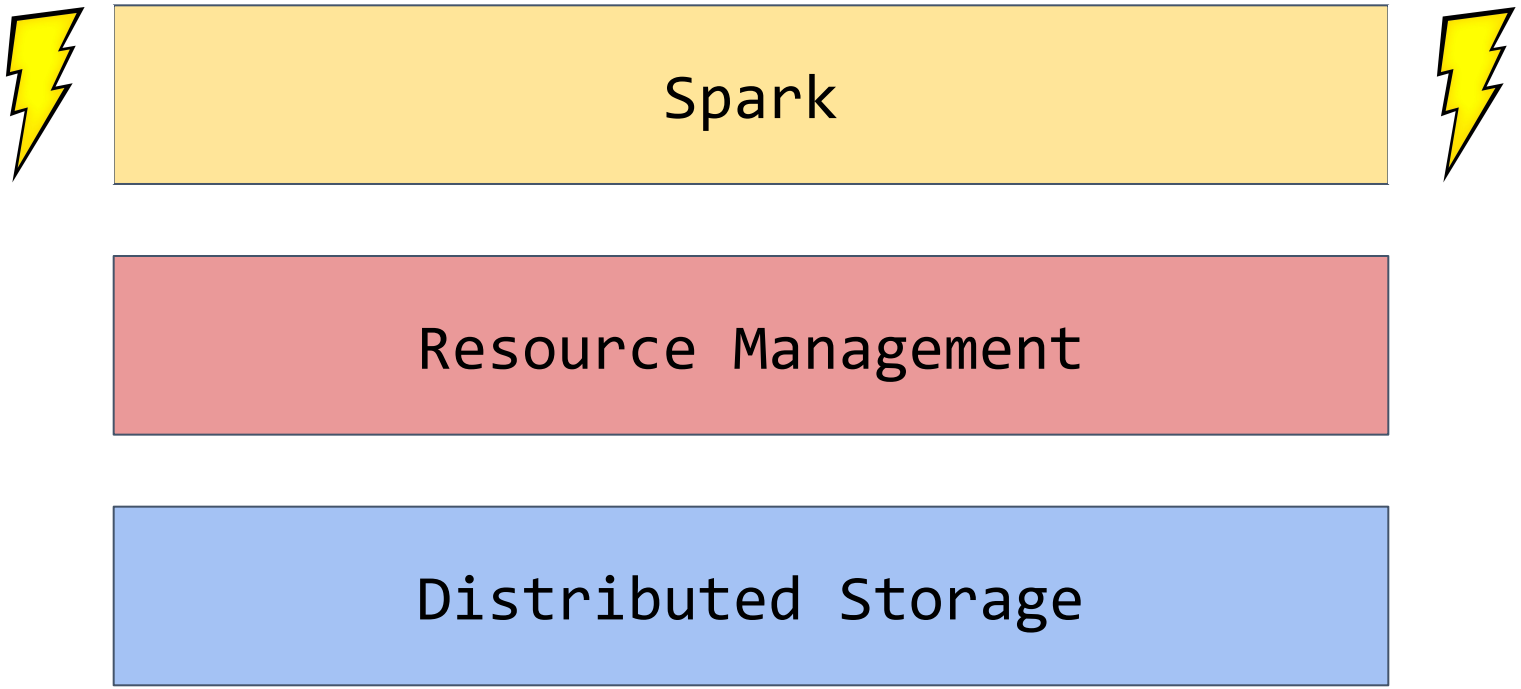
Spark vs MapReduce

Procesamiento distribuido de propósito general:

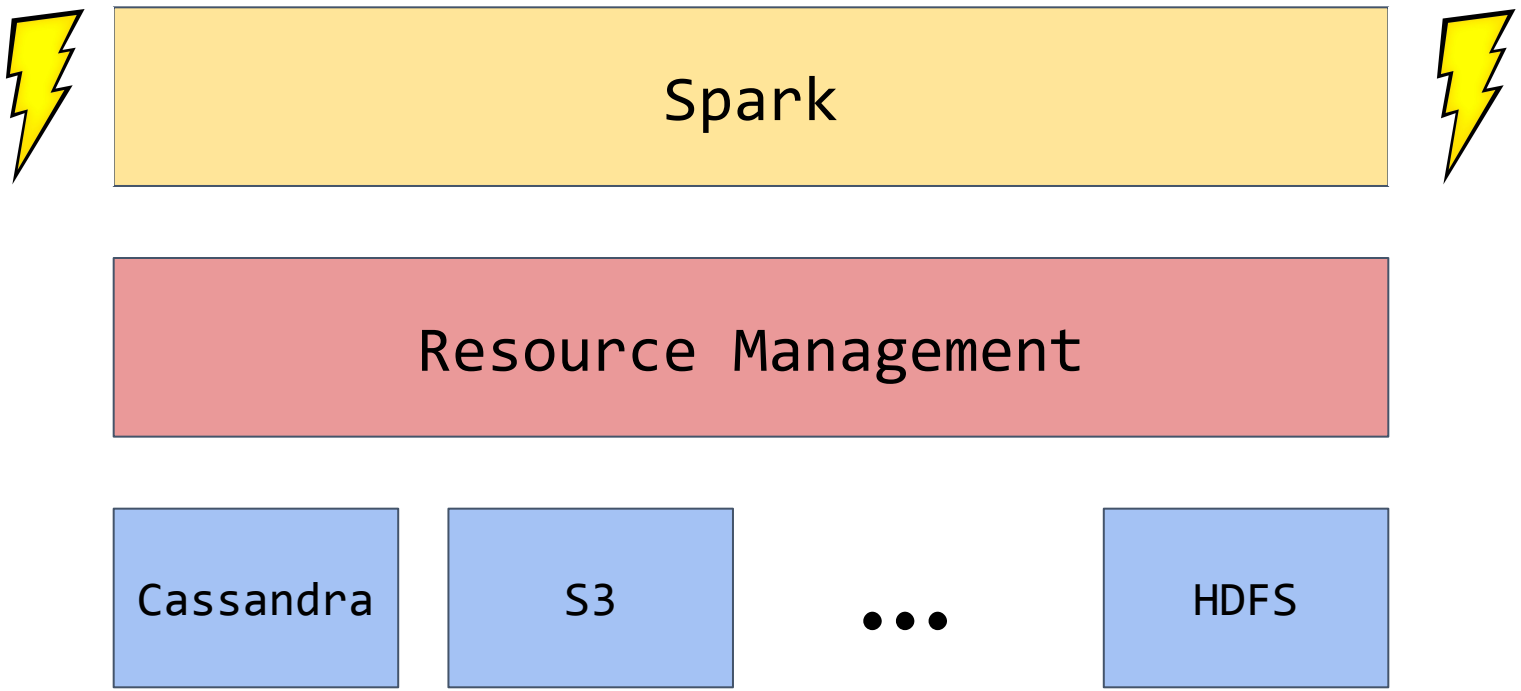
- Map reduce
- Métricas
- SQL
- Machine learning



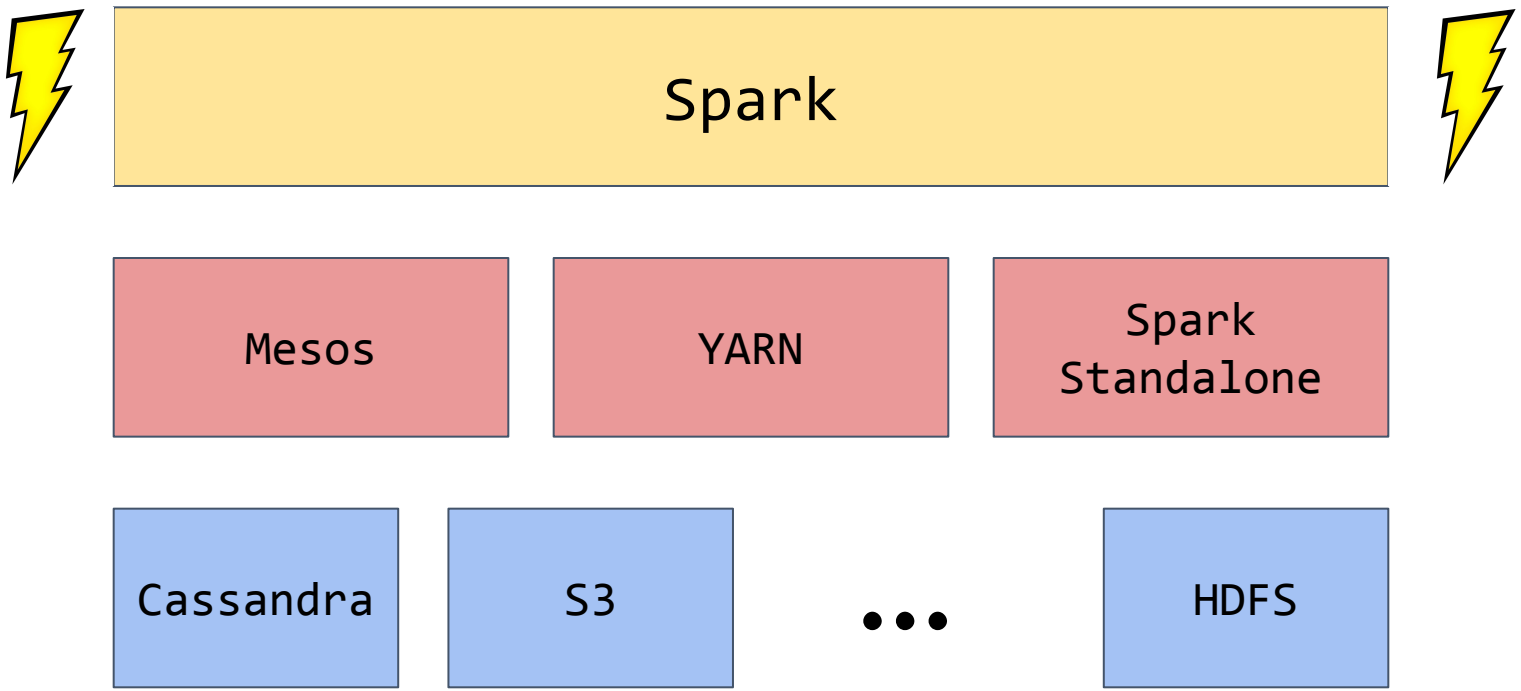
Spark Stack



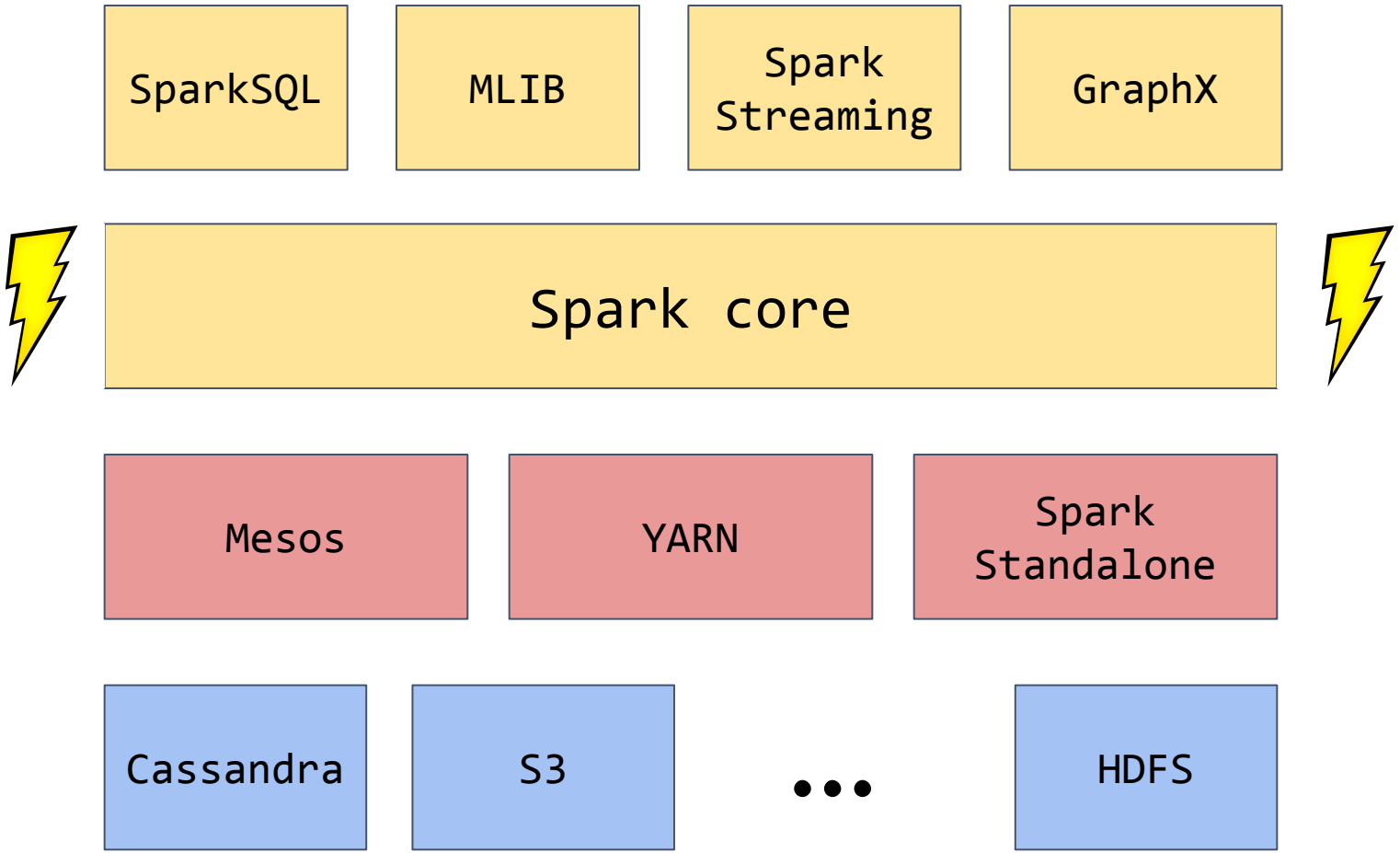
Spark Stack



Spark Stack



Spark Stack



Spark Stack

SparkSQL

Paquete de spark diseñado para trabajar con **información estructurada**. Utiliza una **variante de HQL** (Hive Query Language) y permite la conexión con **múltiples fuentes y formatos** (Hive, NoSQL, RDBMS, Parquet, Avro, Json). A su vez **puede combinarse** con las primitivas de **Spark-core** para combinar SQL con manipulaciones programáticas de datos.

MLib

Funcionalidad y **modelos de machine learning**. Incluye modelos que poseen capacidad de **aprendizaje distribuido** (clasificación, regresión, clustering, filtros colaborativos). También posee algoritmos distribuidos para optimización por descenso de gradiente y reducción dimensional.

Spark Stack

Spark Streaming

Componente que permite **procesar streams de información** en **tiempo real**. Implementa un mecanismo de **micro batches** y permite **interoperar** con **spark-core, sparksql y mlib** para procesamiento brindando las mismas **características** de **tolerancia a fallas**, **escalabilidad** y **distribución** que spark-core.

GraphX

Librería para **manipulación de grafos** de manera **distribuida**. Incluye **algoritmos** comunes como **PageRank**, **conteo de triángulos** y componentes conexas

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark

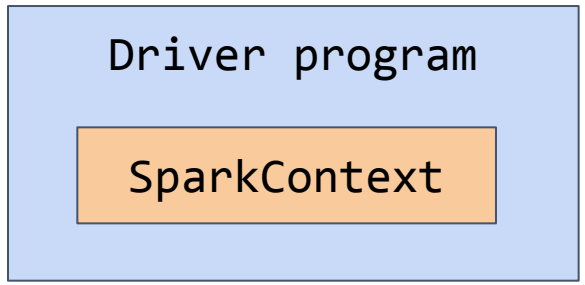
Cuidado!

Trabajar con spark implica comprometerse a conocer y entender los detalles de funcionamiento.

- API y operadores
- Esquema de deploy
- Modelo de memoria
- Tuning, tuning, y tuning

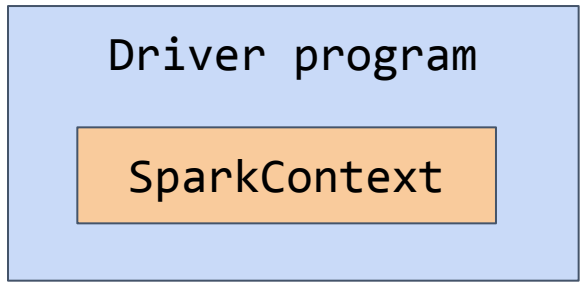
Spark – Application

Toda aplicación de spark comienza con un **driver program**



Spark – Application

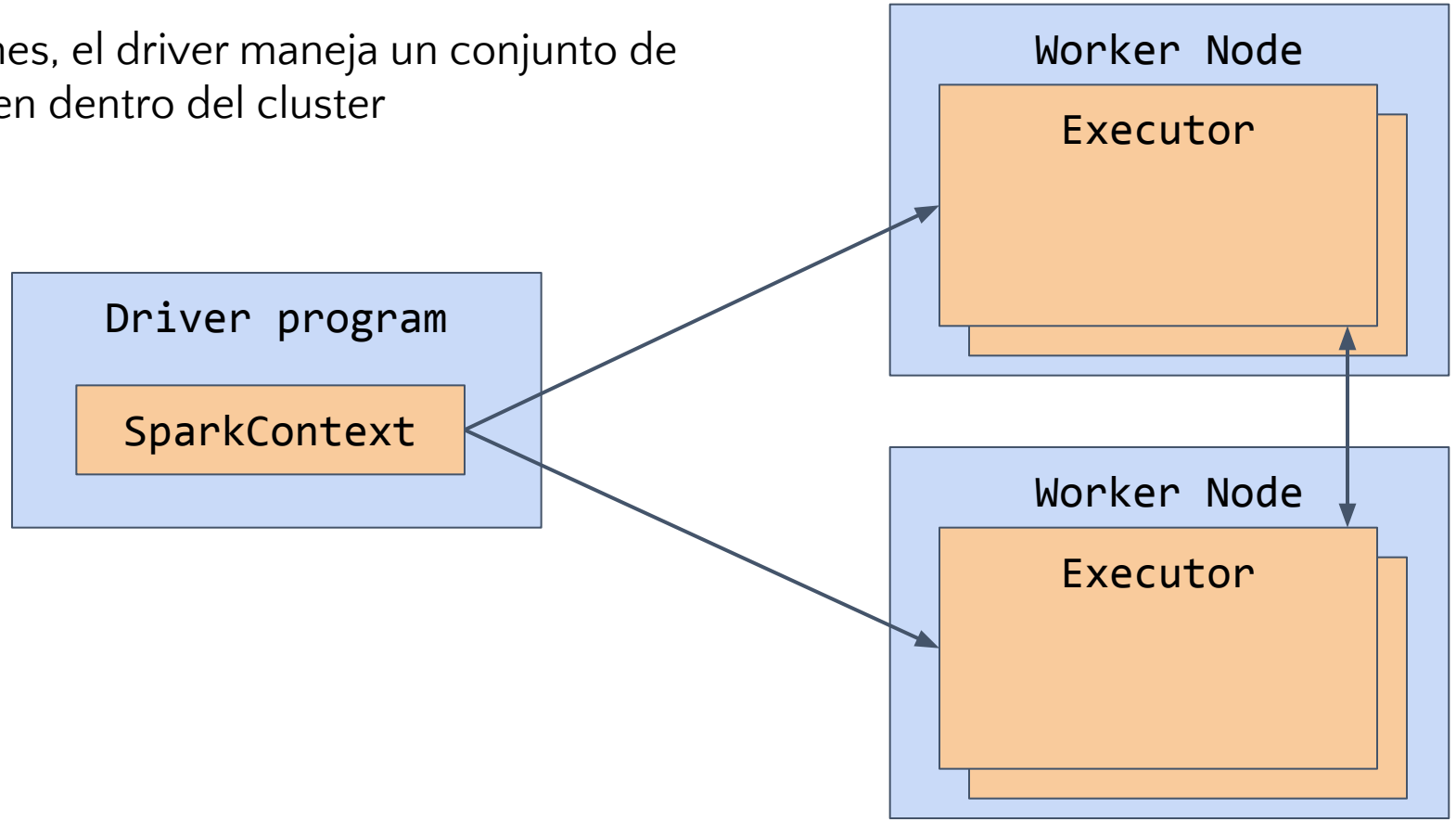
Toda aplicación de spark comienza con un **driver program**



- Contiene la función **main** de nuestra aplicación
- Se comunica con el cluster a través del **SparkContext**
- Lanza todas las operaciones, acciones y transformaciones sobre nuestros datos

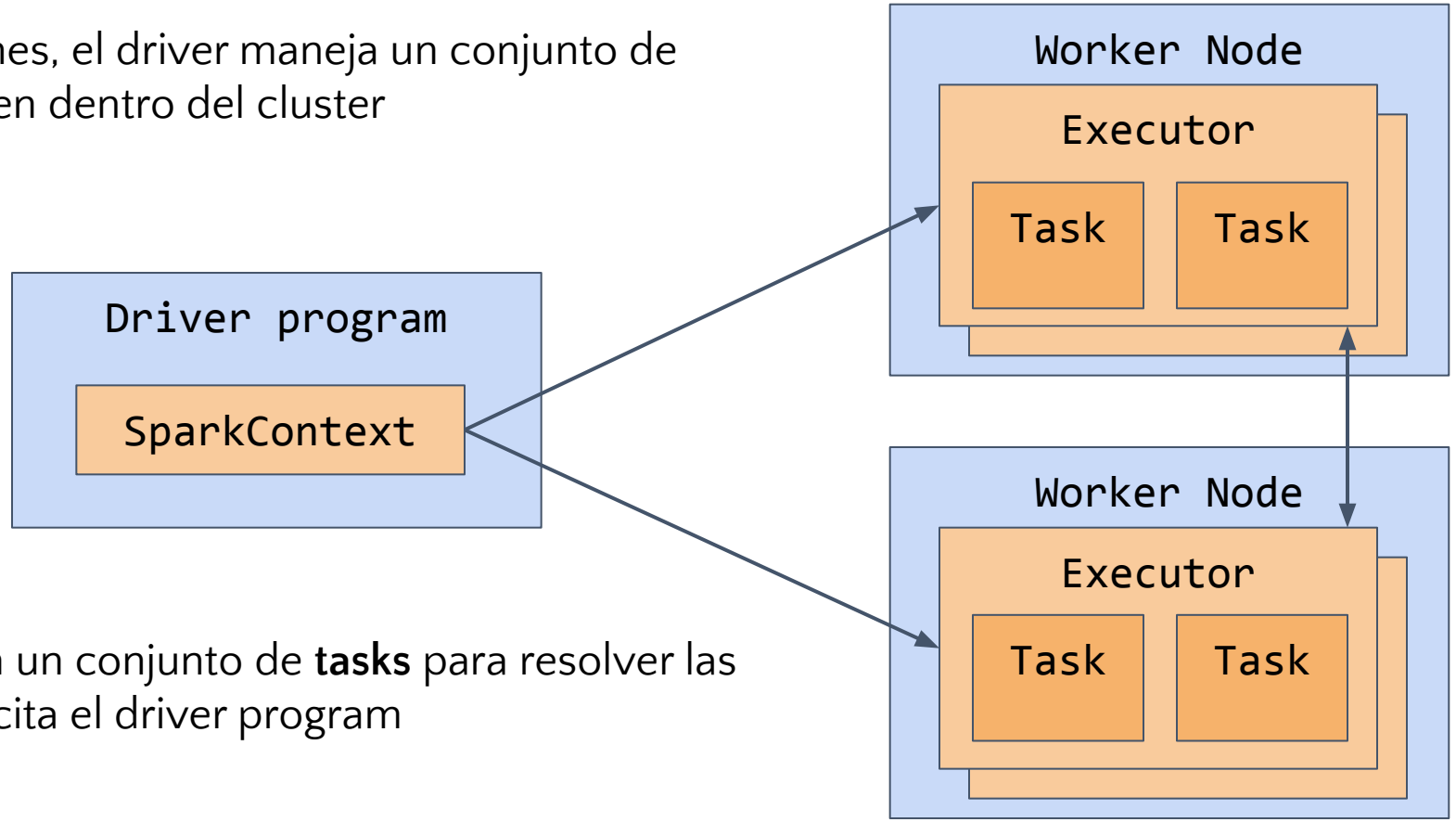
Spark – Application

Para lanzar operaciones, el driver maneja un conjunto de **ejecutores**, que existen dentro del cluster



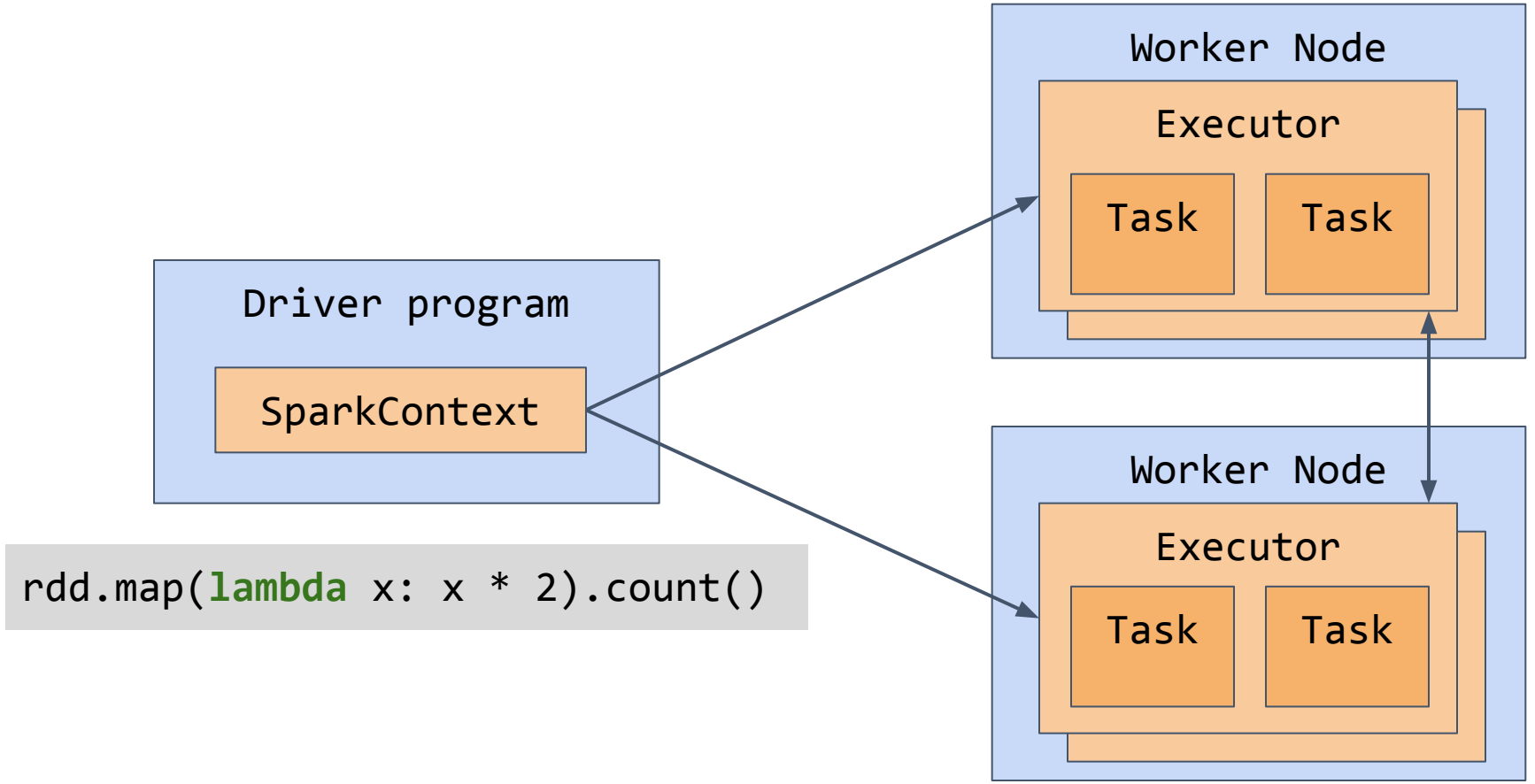
Spark – Application

Para lanzar operaciones, el driver maneja un conjunto de **ejecutores**, que existen dentro del cluster



Los **ejecutores** lanzan un conjunto de **tasks** para resolver las **operaciones** que solicita el driver program

Spark – Application




```
rdd.map(lambda x: x * 2).count()
```

Spark – Resource Manager.

Cuando decidimos usar spark, podemos optar por 3 opciones de resource manager

- YARN
- Mesos
- Spark standalone

Spark – Resource Manager.

Spark – Resource Manager.

Para la práctica vamos a usar **Spark Standalone** (Facilidad de uso, documentación, comunidad)

Spark – Resource Manager.

Para la práctica vamos a usar **Spark Standalone** (Facilidad de uso, documentación, comunidad)

Spark Master

- Responsable de dar visión global de los recursos del cluster
- Punto de contacto para solicitar recursos

Spark Worker

- Responsable por informar la disponibilidad de recursos locales
- Responsable de controlar el uso de recursos

Spark – Resource Manager.

Para la práctica vamos a usar **Spark Standalone** (Facilidad de uso, documentación, comunidad)

Spark Master

- Responsable de dar visión global de los recursos del cluster
- Punto de contacto para solicitar recursos

Spark Worker

- Responsable por informar la disponibilidad de recursos locales
- Responsable de controlar el uso de recursos

HistoryServer

Spark – Ejemplo aplicación

```
$SPARK_HOME/bin/pyspark --master local[1]
```

```
text = spark.sparkContext.textFile("/barney_el_dinosaurio.txt")

words = text.flatMap(lambda line: line.split(" "))

word_counter = words.map(lambda word: (word, 1))

counts = word_counter.reduceByKey(lambda prev,new: prev + new)

counts.sortBy(lambda tuple: tuple[1], False).take(100)
```

Spark - Core concepts

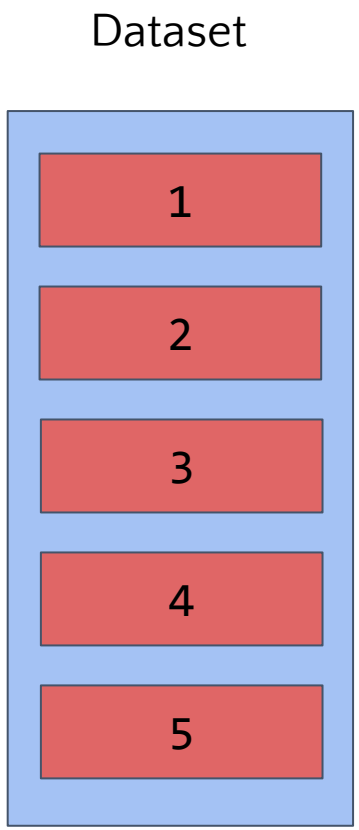
Spark – RDD

La estructura de datos básica que maneja spark es el **RDD** (Resilient Distributed Dataset)

Un **RDD** es:

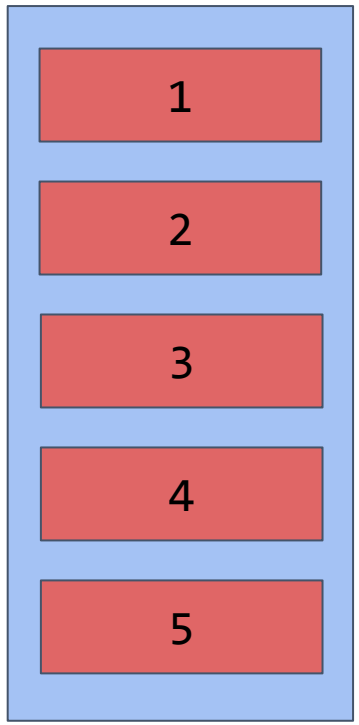
- Una colección distribuida de elementos
  - Cada RDD internamente se divide en un conjunto de **particiones**
- Tolerante a fallas
- **Inmutable**
- Evaluada de manera **lazy**

Spark – RDD



Spark – RDD

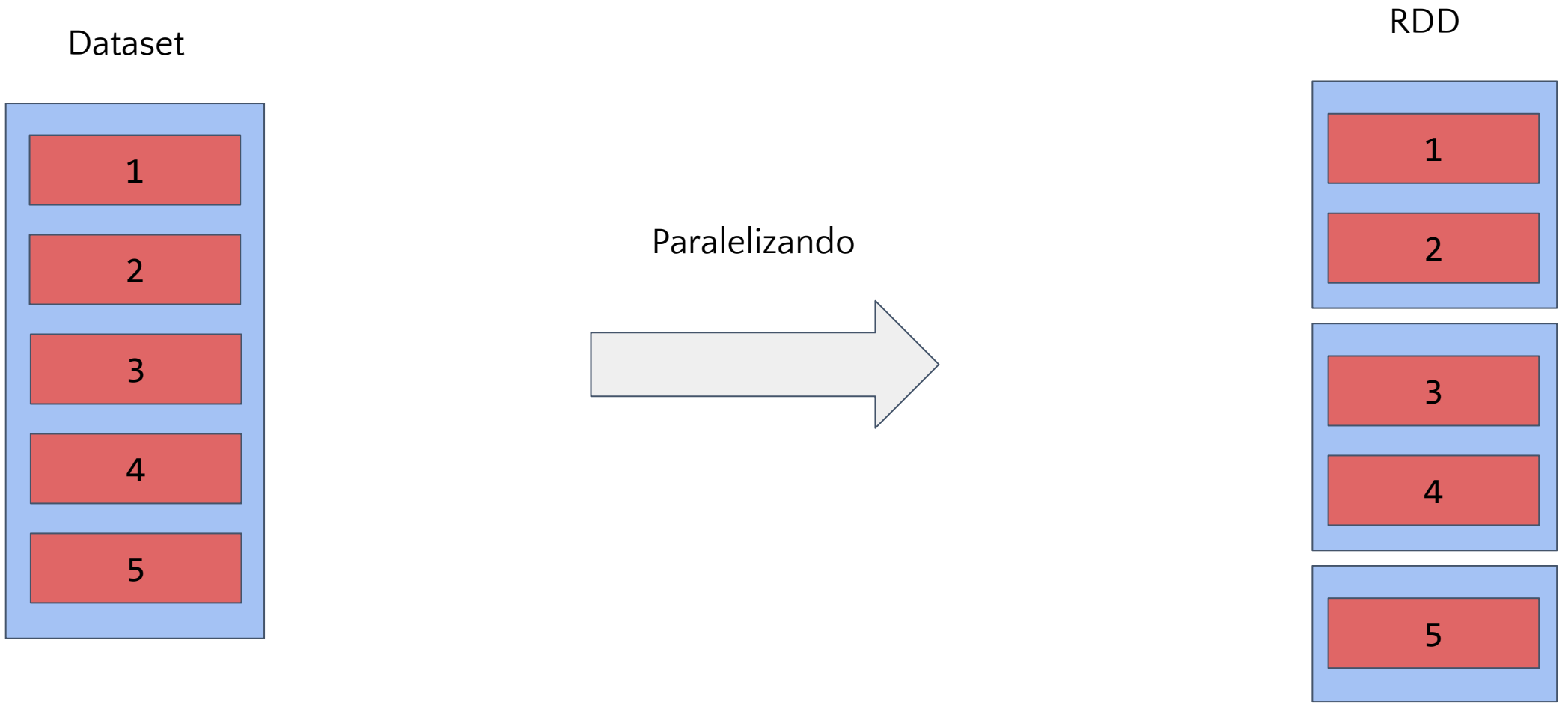
Dataset



Paralelizando

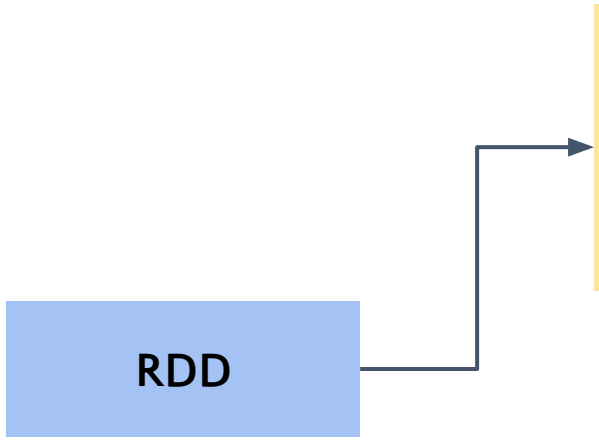
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark – RDD



Spark - RDD - Operaciones

Spark – RDD – Operaciones

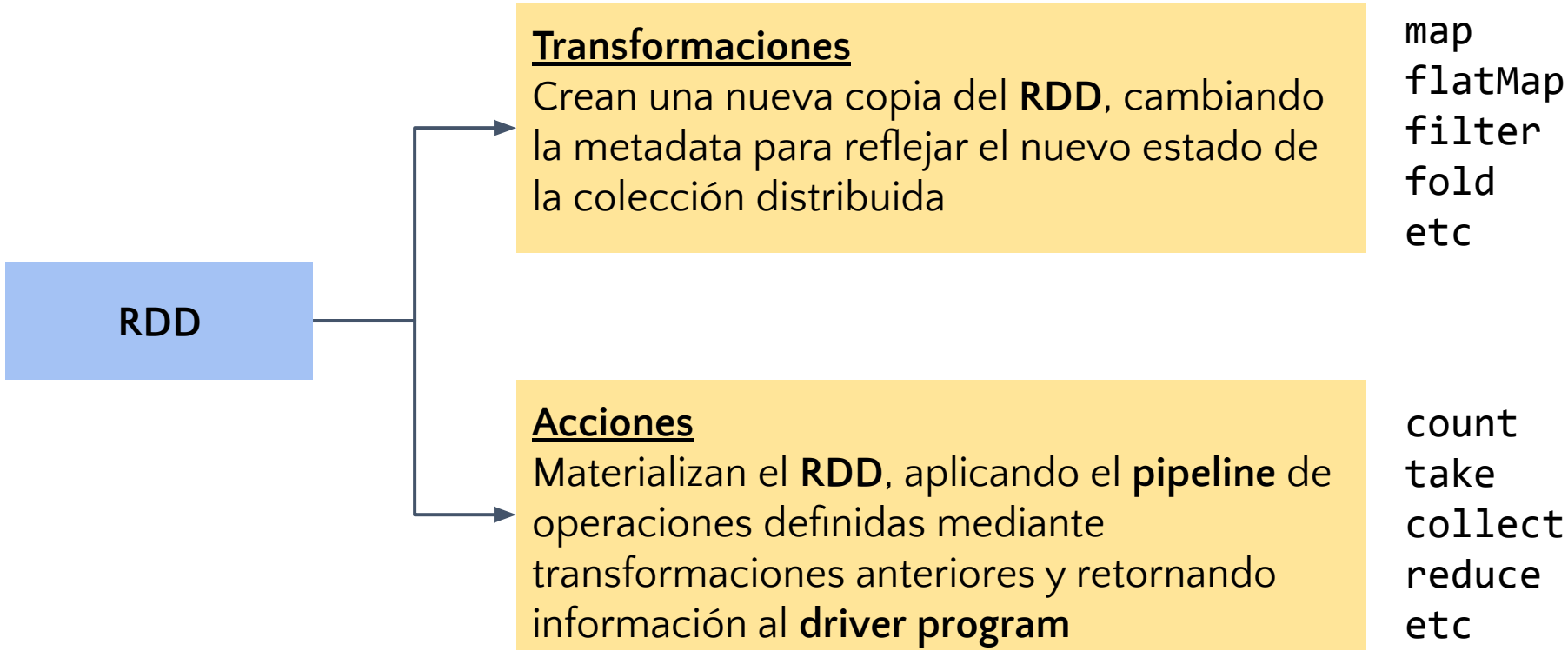


Transformaciones

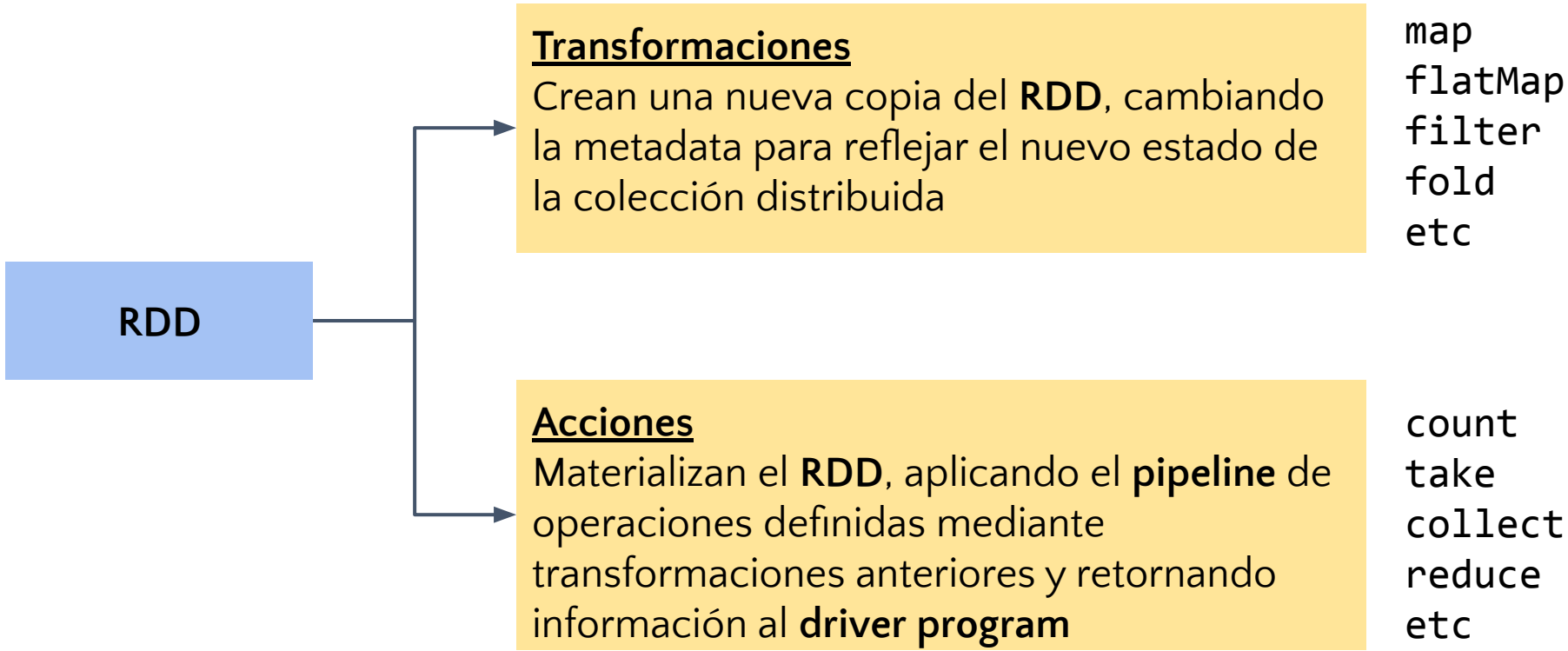
Crean una nueva copia del **RDD**, cambiando la metadata para reflejar el nuevo estado de la colección distribuida

```
map
flatMap
filter
fold
etc
```

Spark – RDD – Operaciones



Spark – RDD – Operaciones



Para distinguirlas, el retorno de todas las **transformaciones** es siempre otro **RDD**, mientras que el retorno de una **acción**, depende de la operación y puede ser un **valor o una lista de valores**

Spark – RDD – Transformaciones

Las transformaciones son operaciones que retornan un nuevo RDD.
Aplicar transformaciones a un RDD es "gratuito", ya que sólo alteran los metadatos de un **RDD** y son ejecutadas cuando se llama a una **acción** posterior.

Se puede pensar en un RDD como la metainformación o el conjunto de operaciones para llegar a un resultado.

```
rdd5 = spark.sparkContext.textFile("/barney_el_dinosaurio.txt")
    .flatMap(lambda line: line.split(" "))
    .map(lambda word: (word, 1))
    .reduceByKey(lambda prev,new: prev + new)
    .sortBy(lambda tuple: tuple[1], False)
```

Spark - RDD - Transformaciones

Todo **RDD** mantiene el grafo acíclico de las transformaciones necesarias para llegar al estado final


```
rdd5 = spark.sparkContext.textFile("/barney_el_dinosaurio.txt")
    .flatMap(lambda line: line.split(" "))
    .map(lambda word: (word, 1))
    .reduceByKey(lambda prev,new: prev + new)
    .sortBy(lambda tuple: tuple[1], False)
```


UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark – RDD – Transformaciones

Suponiendo que tenemos un **`RDD1 = sc.parallelize([1,2,3,3])`** y **`RDD2 = sc.parallelize([3,4,5])`**

Funcion	Descripción	Ejemplo	Resultado
`map(func)`	Transforma una colleción aplicando la función a cada elemento	`rdd1.map(lambda x: x * 2)`	`RDD: {2,4,6,6}`

`flatMap(func)`	Transforma una colección, aplicando una función que devuelve otra colección y aplanado su resultado	`rdd1.flatMap(`
		`  lambda x:`
		`  [x, x+1, x+2])`
			`RDD:`
			`{1,2,3,2,3,4,3,4,5,3,4,5}`

`filter(func)`	Devuelve un rdd conteniendo los elementos que cumplan con la condición brindada	`rdd1.filter(lambda x:` `x % 2 == 0)`	`RDD: {2}`
`distinct()`	Elimina los duplicados del rdd	`rdd1.distinct()`	`RDD: {1,2,3}`
`reduceByKey(func)`	Reduce cada uno de los valores de los elementos que tengan igual key	`-----`	`-----`

Spark – RDD – Transformaciones

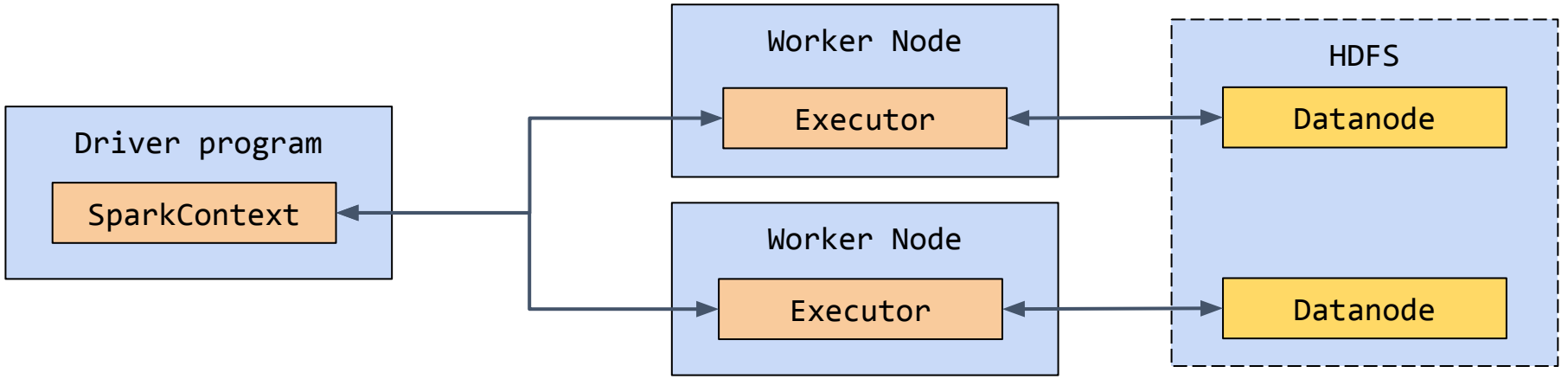
Suponiendo que tenemos un **`RDD1 = sc.parallelize([1,2,3,3])`** y **`RDD2 = sc.parallelize([3,4,5])`**

Funcion	Descripción	Ejemplo	Resultado
`sample(withReplacement, fraction, [seed])`	Devuelve una muestra del rdd, reemplazando el rdd, o devolviendo uno nuevo	`rdd1.sample(False, 0.1, 42)`	`RDD: -----`
`union()`	Retorna un RDD que contiene elementos de ambos rdds	`rdd1.union(rdd2)`	`RDD: {1,2,3,3,3,4,5}`
`intersection()`	Devuelve un RDD con los elementos que existen en ambos rdds	`rdd1.intersection(rdd2)`	`RDD: {3}`
`subtract()`	Elemina elementos de un RDD	`rdd1.subtract(rdd2)`	`RDD: {1,2}`
`cartesian()`	Realiza el producto cartesiano	`rdd1.cartesian(rdd2)`	`RDD: {(1,3),(1,4)...(3,5)}`

Spark - RDD - Acciones

Las acciones son operaciones que "hacen algo" con los datasets.
Materializan el grafo de transiciones del RDD para **retornar un valor al driver**, **operar sobre las particiones** o **escribir en una fuente de datos externa**

```
rdd = spark.sparkContext.textFile("/barney_el_dinosaurio.txt")

rdd.count()
```



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark – RDD – Acciones

Suponiendo que tenemos un **`RDD = sc.parallelize([1,2,3,3])`**

Funcion	Descripción	Ejemplo	Resultado
`collect()`	Retorna todos los elementos del RDD a la **memoria del driver**	`rdd.collect()`	`[1,2,3,3]`
`count()`	Retorna la cantidad de elementos en un rdd	`rdd.count()`	`4`
`take(n)`	Devuelve **n** elementos de un rdd	`rdd.take(2)`	`[2,3]`
`reduce(func)`	Reduce un rdd aplicando la función suministrada	`rdd.reduce(` `lambda x,y: x + y)`	`9`
`fold(seed, func)`	Aplica un **fold** al rdd	`rdd.fold(0,` `lambda x,y: x + y)`	`9`
`foreach(func)`	Aplica la función a cada elemento del rdd. **No retorna valores al driver**	`rdd.foreach(func)`	`------`

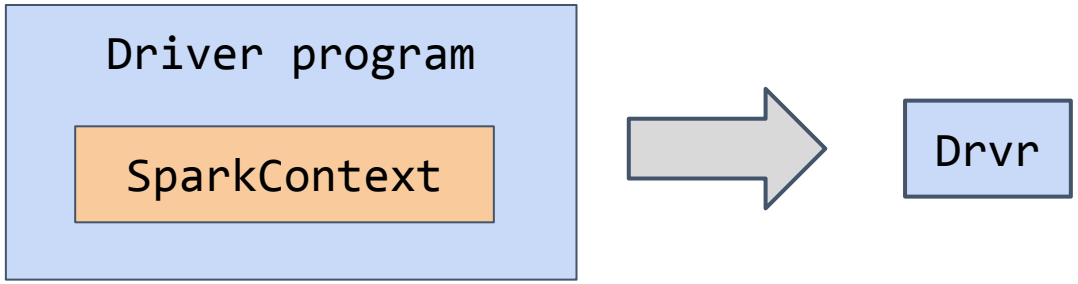
Spark - deploy

Spark - Deploy

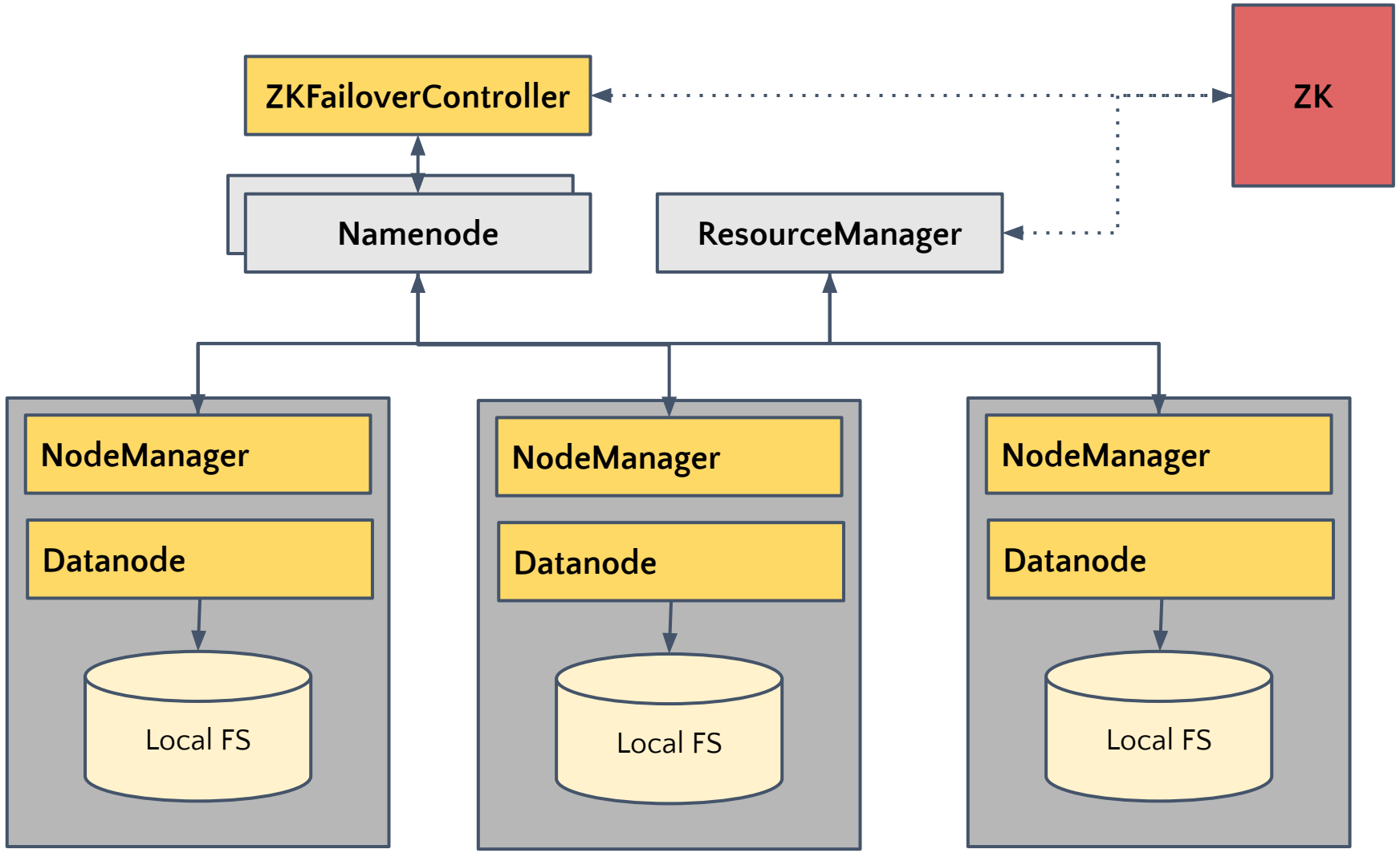
Al lanzar una aplicación de spark, existen dos opciones:

- **Modo client**
- **Modo cluster**

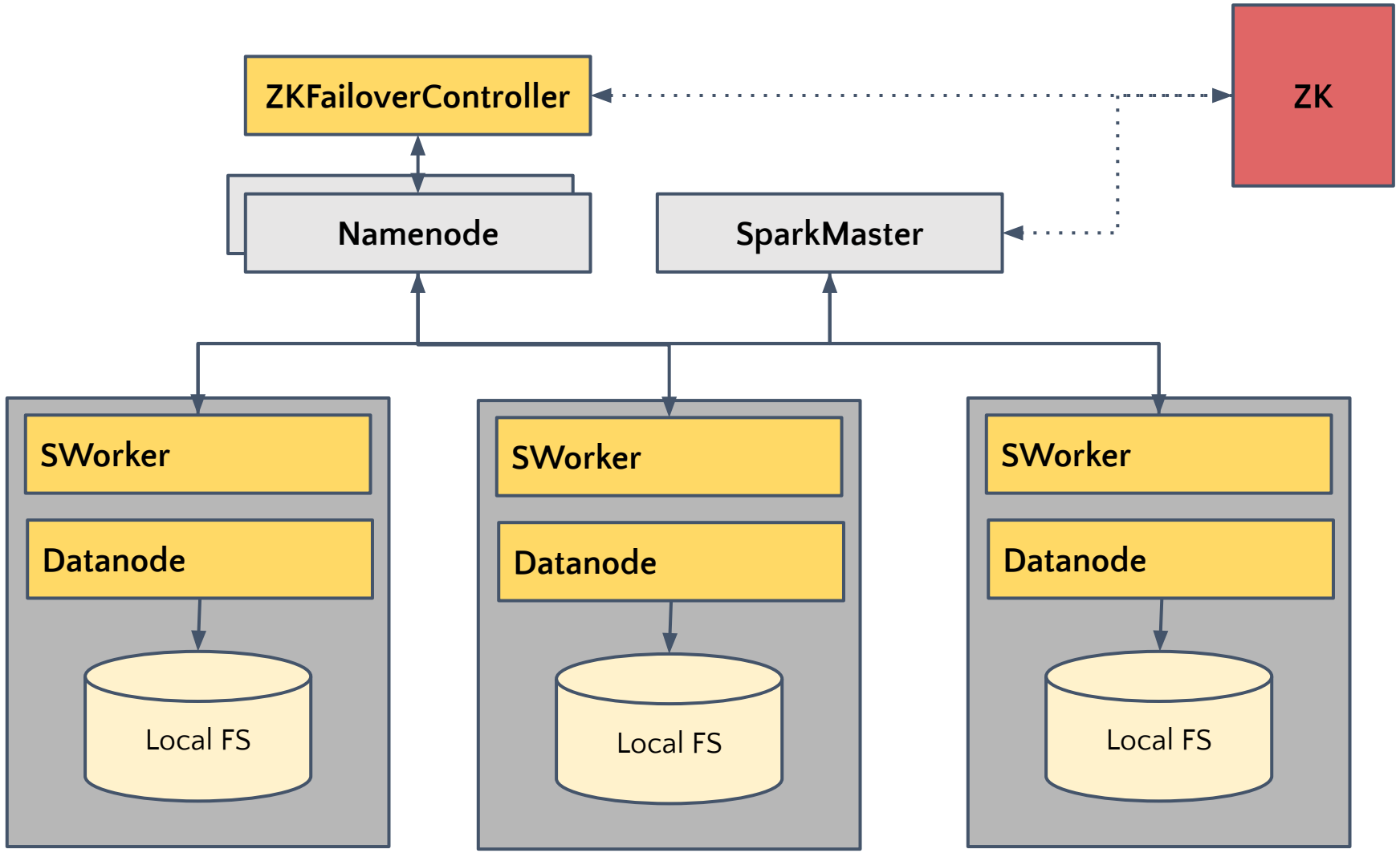
Dependiendo donde se quiera ejecutar el **driver program** se debe elegir un modo o el otro



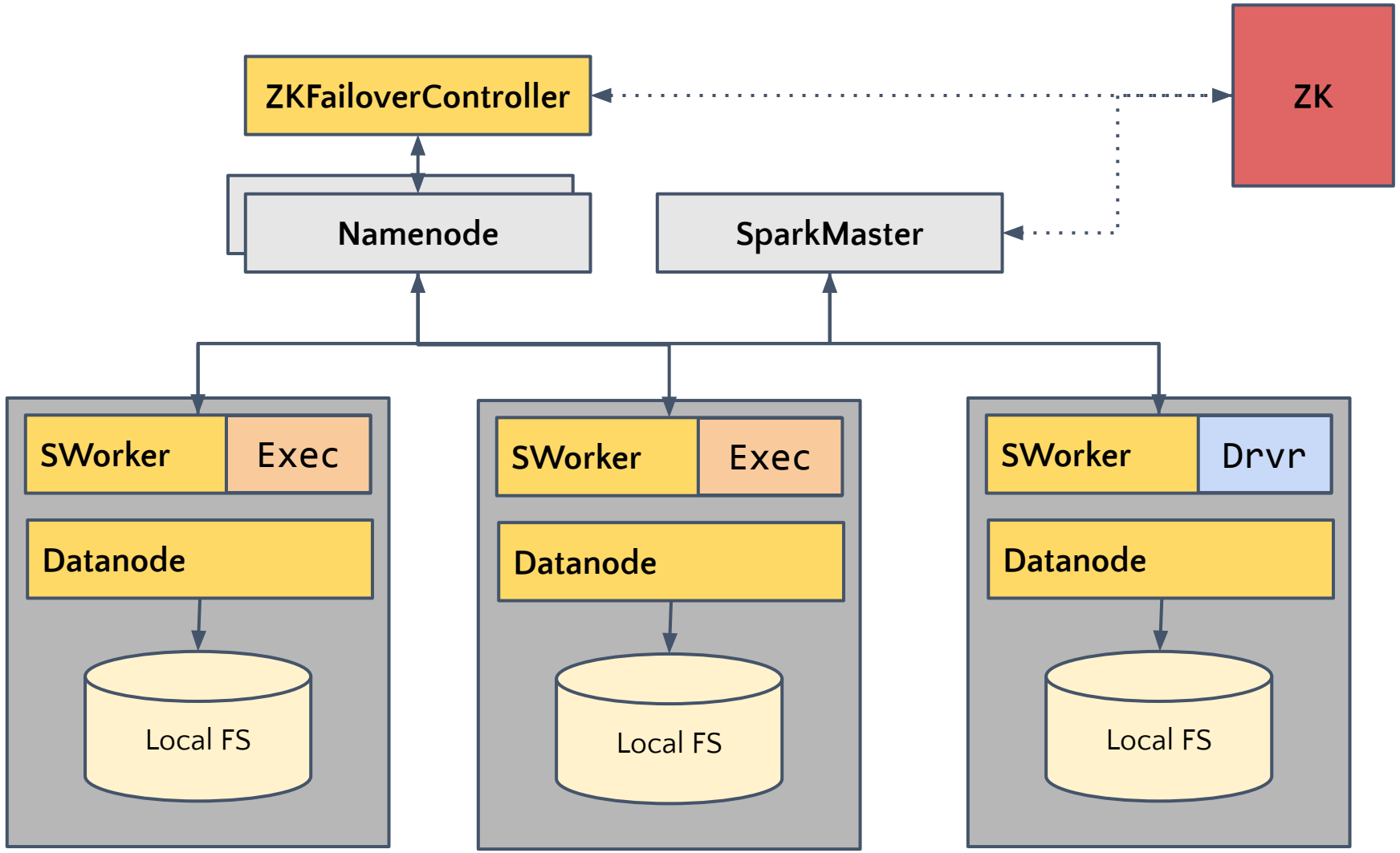
Spark – Deploy



Spark – Deploy



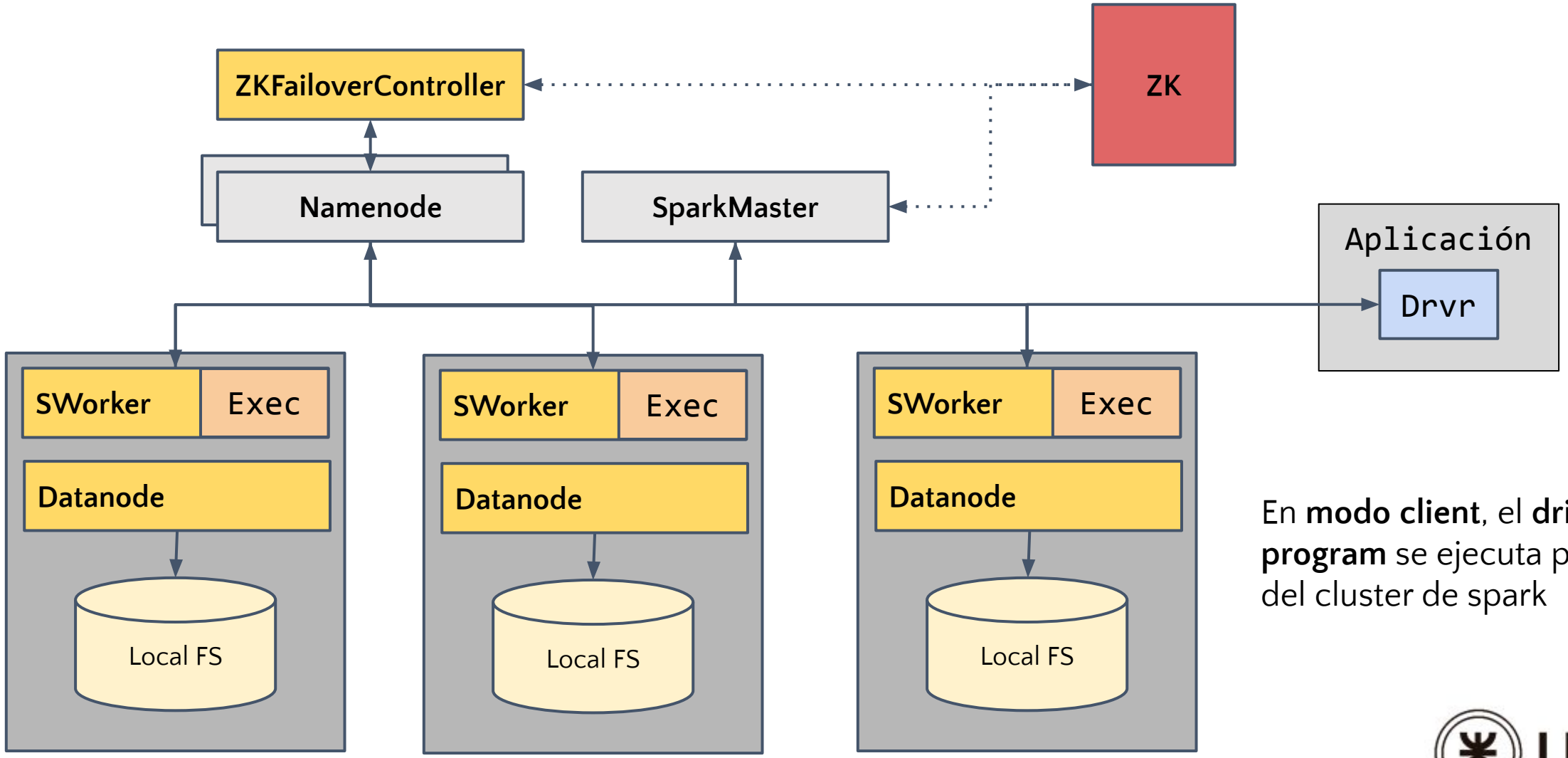
Spark - Deploy - Cluster Mode



En **modo cluster** tanto el **driver program** como los executors se ejecutan dentro de los workers de spark

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark - Deploy - Client mode



En **modo client**, el **driver program** se ejecuta por fuera del cluster de spark

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark - Deploy

Para conectarse a la consola de spark:

```
$SPARK_HOME/bin/pyspark
```

El comando anterior ejecuta una aplicación spark en modo client, dentro de una aplicación que nos permite compilar código en tiempo real contra el kernel de spark

La consola de spark en python, nos brinda un **sparkSession** (spark), un **sparkContext** (sc) y un **sqlContext** (sqlContext) gratuitamente, usando la configuración del binario.

Para deployar un programa y no una consola, es necesario construir esos objetos

```
rdd = spark.sparkContext.textFile("/barney_el_dinosaurio.txt")
```

Spark - Deploy

ejemplo.py

```
from pyspark.sql import SparkSession

spark = SparkSession.builder()
    .appName("miaplicacion")
    .master("spark://localhost:7077")
    .getOrCreate()

rdd = spark.sparkContext.textFile("/barney_el_dinosaurio.txt")
```

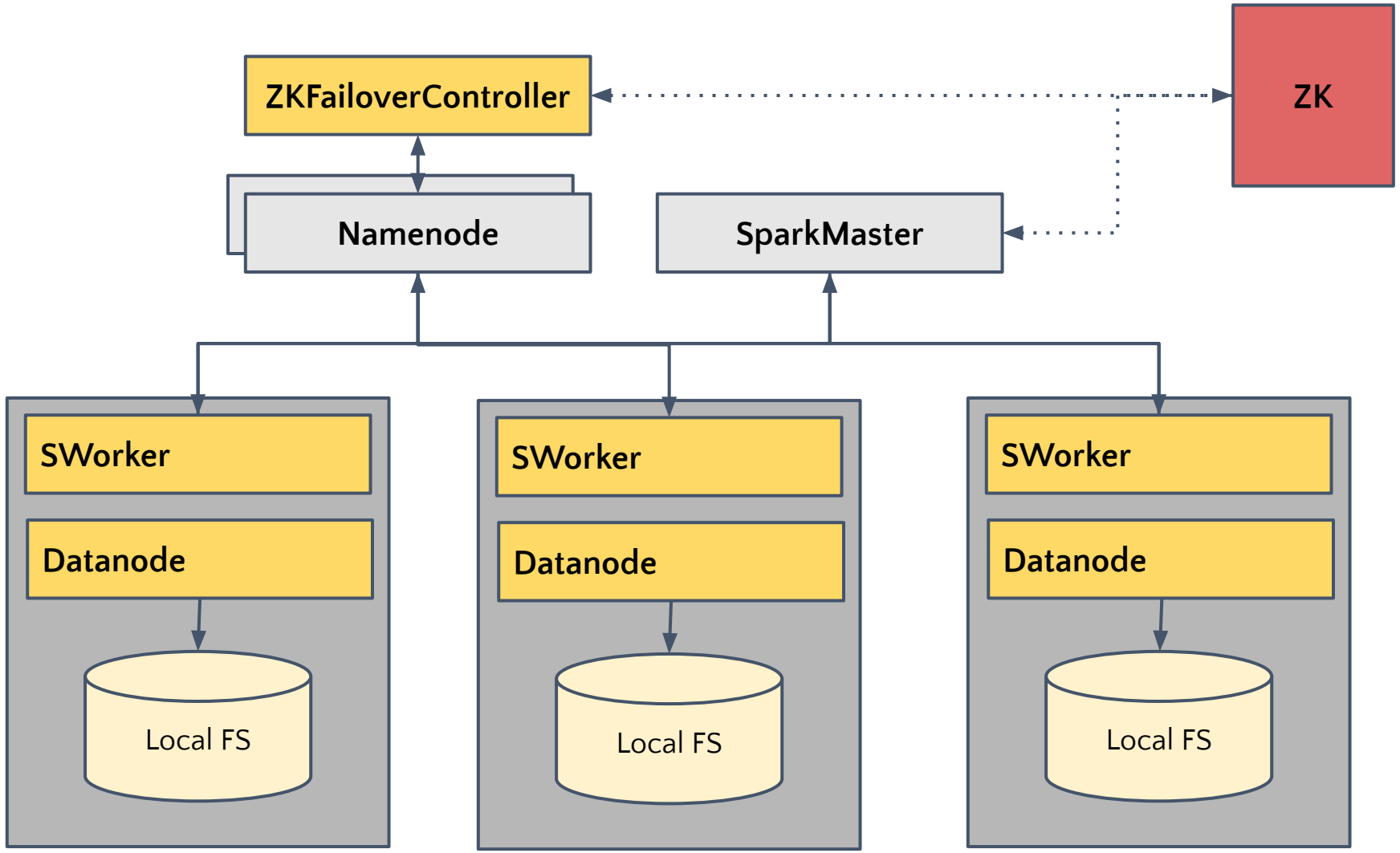
```
$SPARK_HOME/bin/spark-submit --master local[1] --deploy-mode client ./ejemplo.py
```

Spark – Deploy

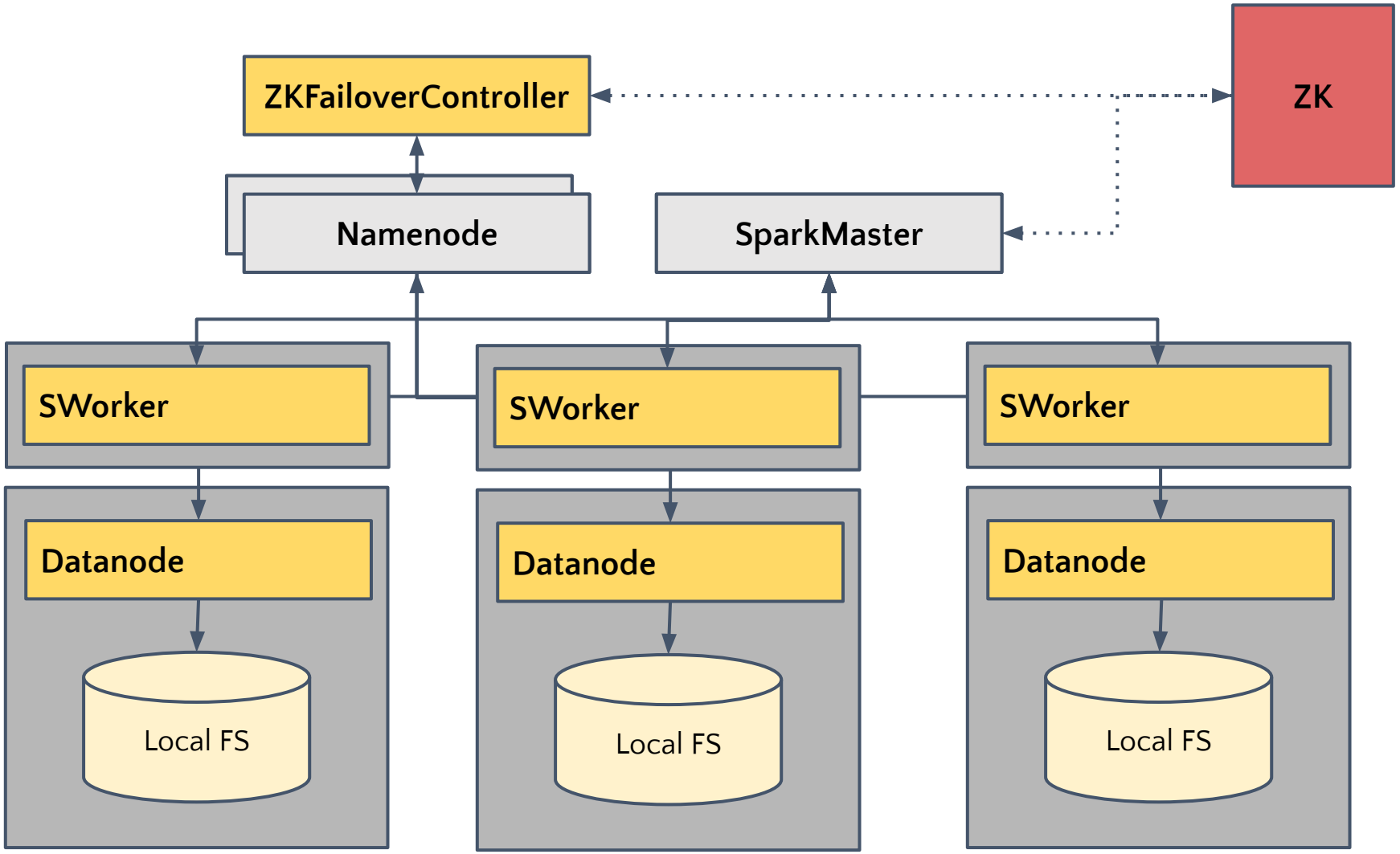
Spark submit

```
$SPARK_HOME/bin/spark-submit \
    --master <master-url> \
    [--class <nombre completo clase] \
    --deploy-mode <deploy-mode> \
    --conf <key>=<value> \
    --jars <lista de jars separados por coma> \
    --packages <paquetes para descargar de maven> \
    <python-file>|<jar file>
    [<argumentos>]
```

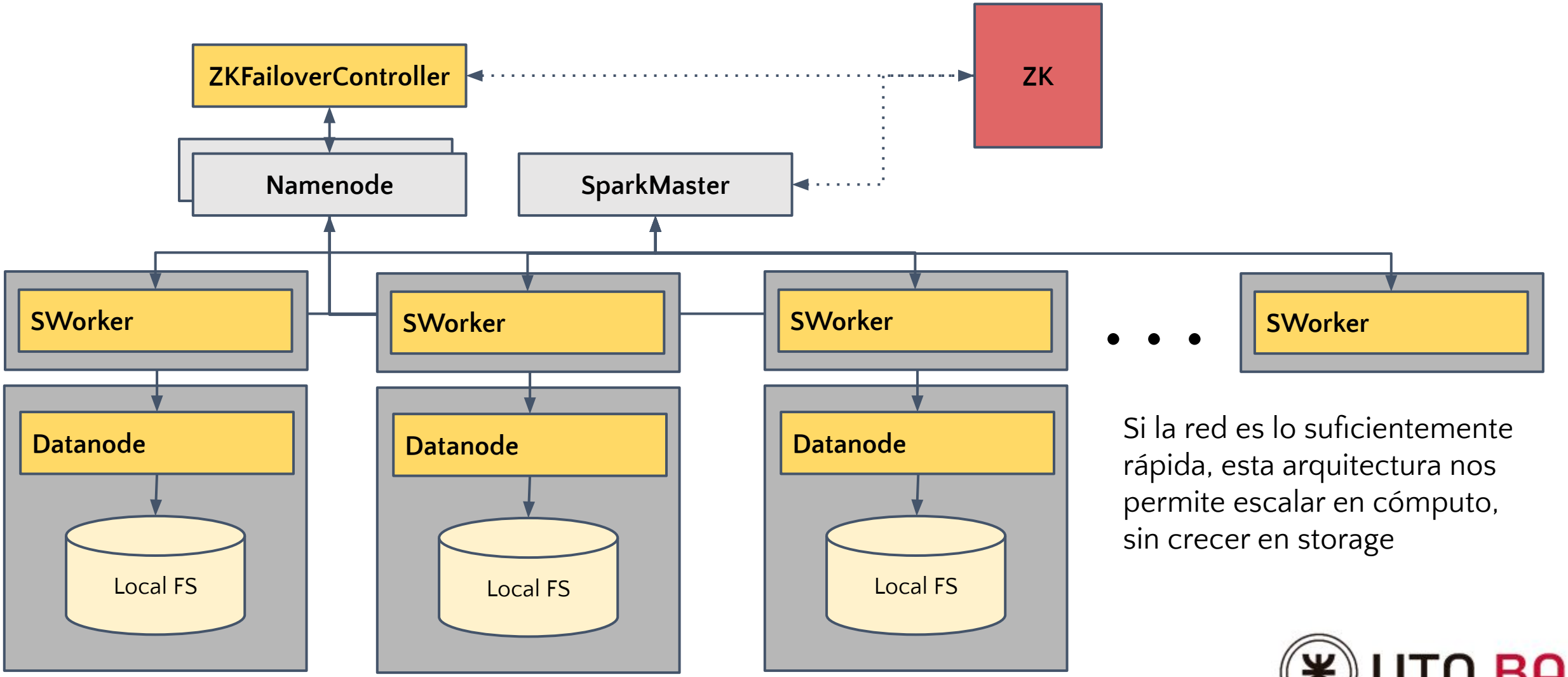
Spark – Deploy



Spark - Deploy



Spark – Deploy



Si la red es lo suficientemente rápida, esta arquitectura nos permite escalar en cómputo, sin crecer en storage

Spark – Round II

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark – Ejemplo aplicación

Cual es el problema con el siguiente código?

```
text = spark.sparkContext.textFile("/barney_el_dinosaurio.txt")
print "el archivo tiene {cantidad} lineas".format(cantidad=text.count())
print ",".join(text.collect())
```

Spark – Ejemplo aplicación

Cual es el problema con el siguiente código?

```
text = spark.sparkContext.textFile("/barney_el_dinosaurio.txt")
print "el archivo tiene {cantidad} lineas".format(cantidad=text.count())
print ",".join(text.collect())
```

La ejecución de una acción sobre un rdd puede ser muy costosa. En este caso, se están realizando dos acciones, una después de la otra, iterando dos veces por el archivo completo.

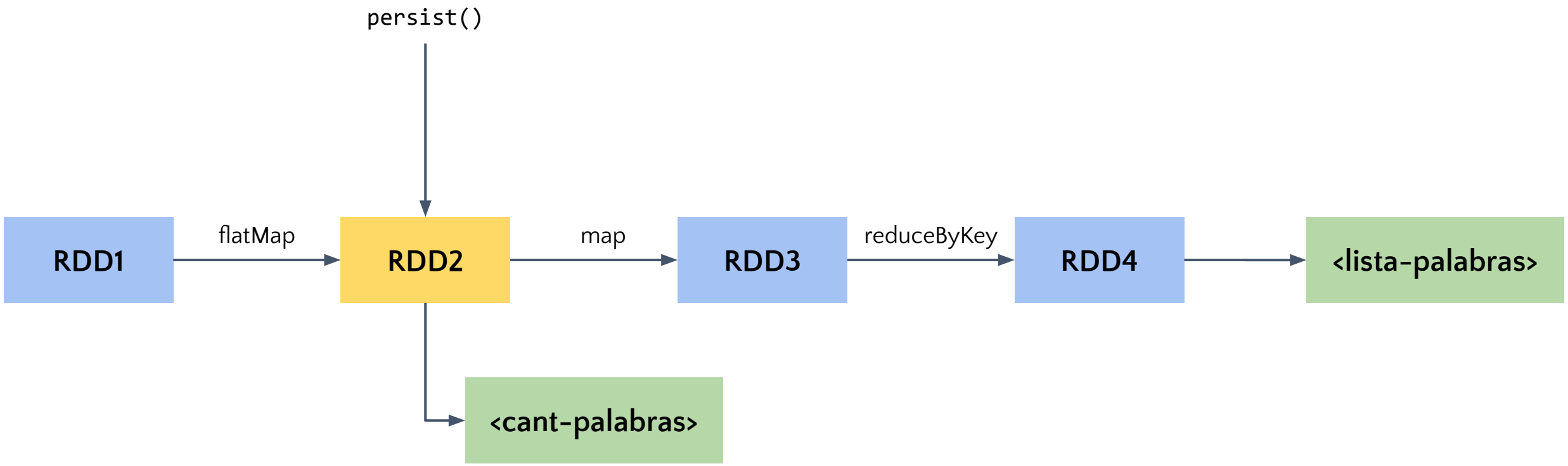
Spark – Persist

Es posible cachear y persistir el estado (cadena de transformaciones) y los datos de un rdd a memoria o disco.

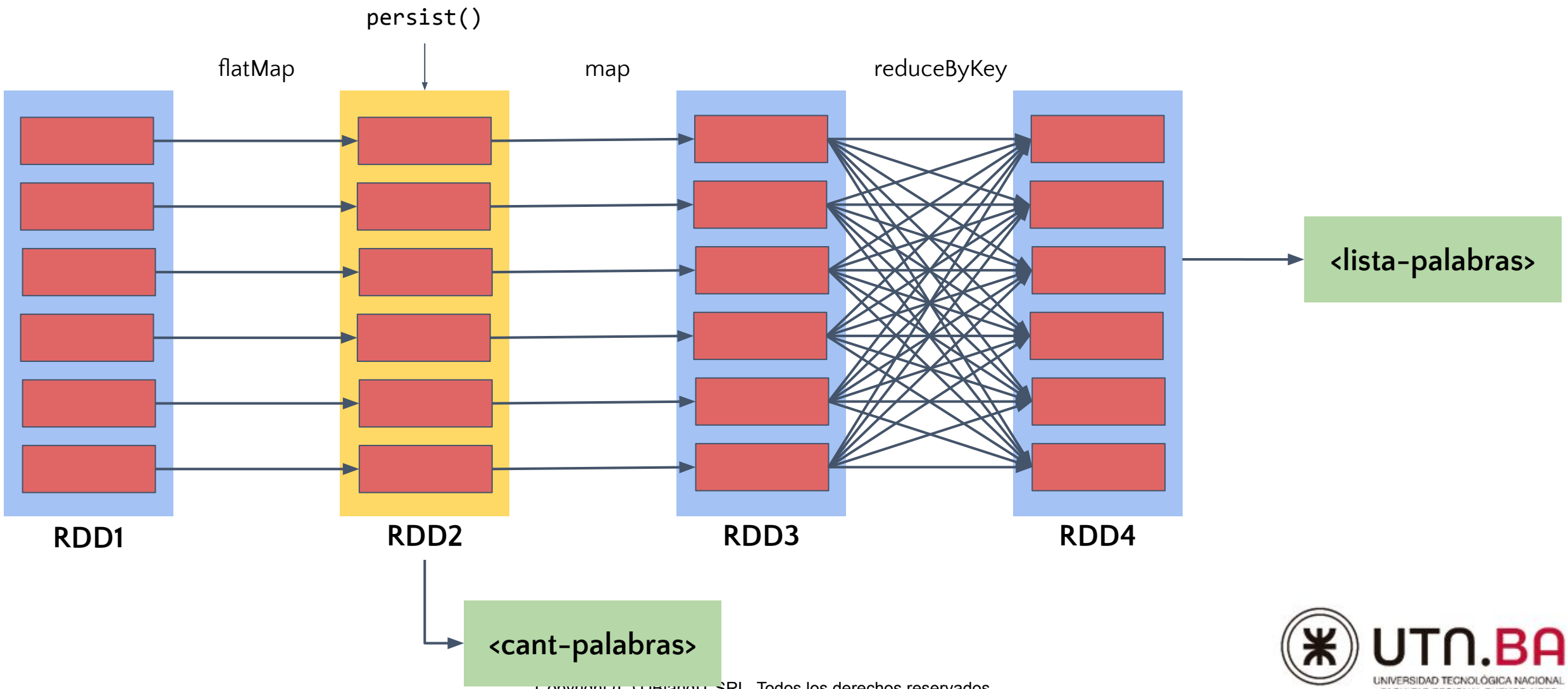
```
from pyspark.storagelevel import StorageLevel

text = spark.sparkContext.textFile("/barney_el_dinosaurio.txt")
words = text.flatMap(lambda x: x.split(" "))
words.persist(StorageLevel.MEMORY_ONLY)
print "el archivo tiene {cantidad} palabras".format(cantidad=words.count())

word_count = words.map(lambda x: (x, 1)).reduceByKey(lambda x,y: x + y)
print ",".join(word_count.collect())
```

Persistir **no es una acción**. Cuando se ejecute una acción posterior y se detecte un persist en algún paso, se guardará en cada nodo la **materialización de la partición del rdd correspondiente**

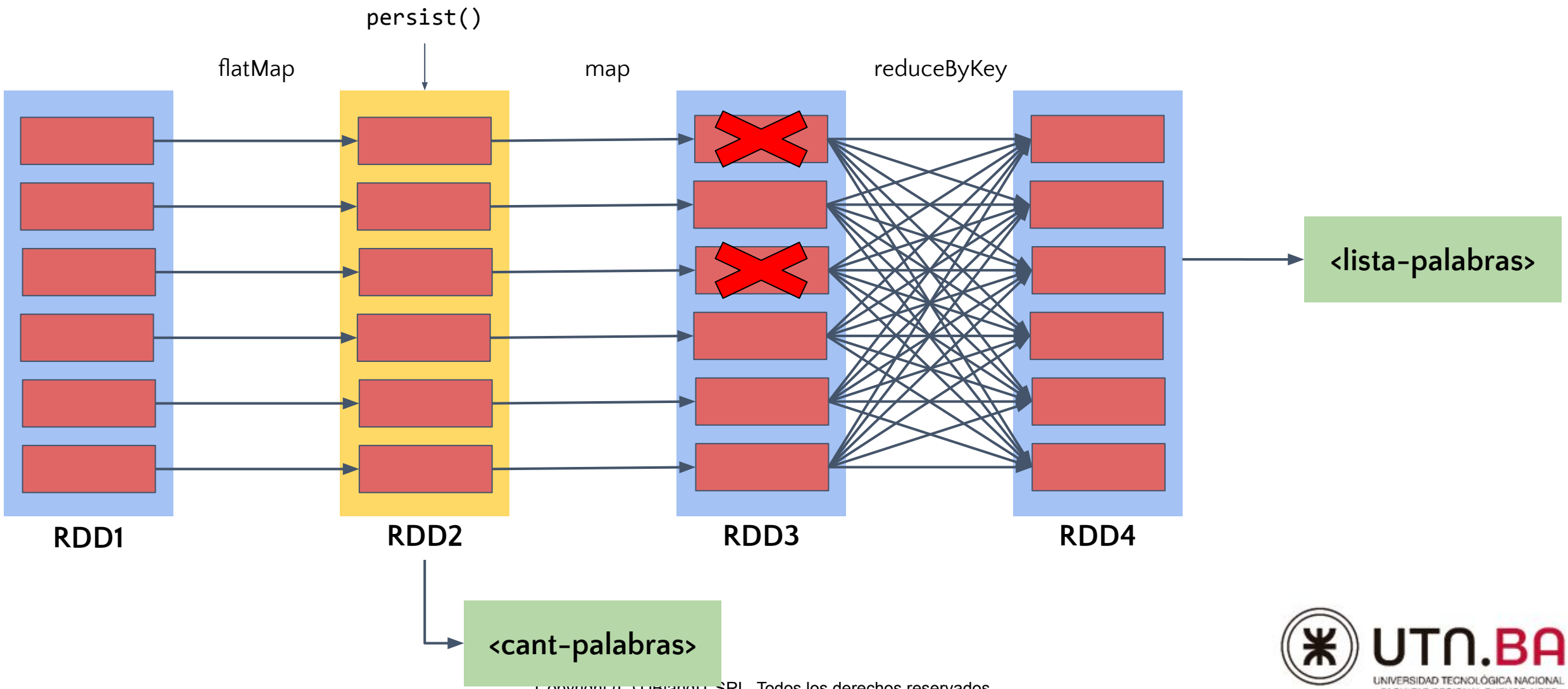
Spark - Persist



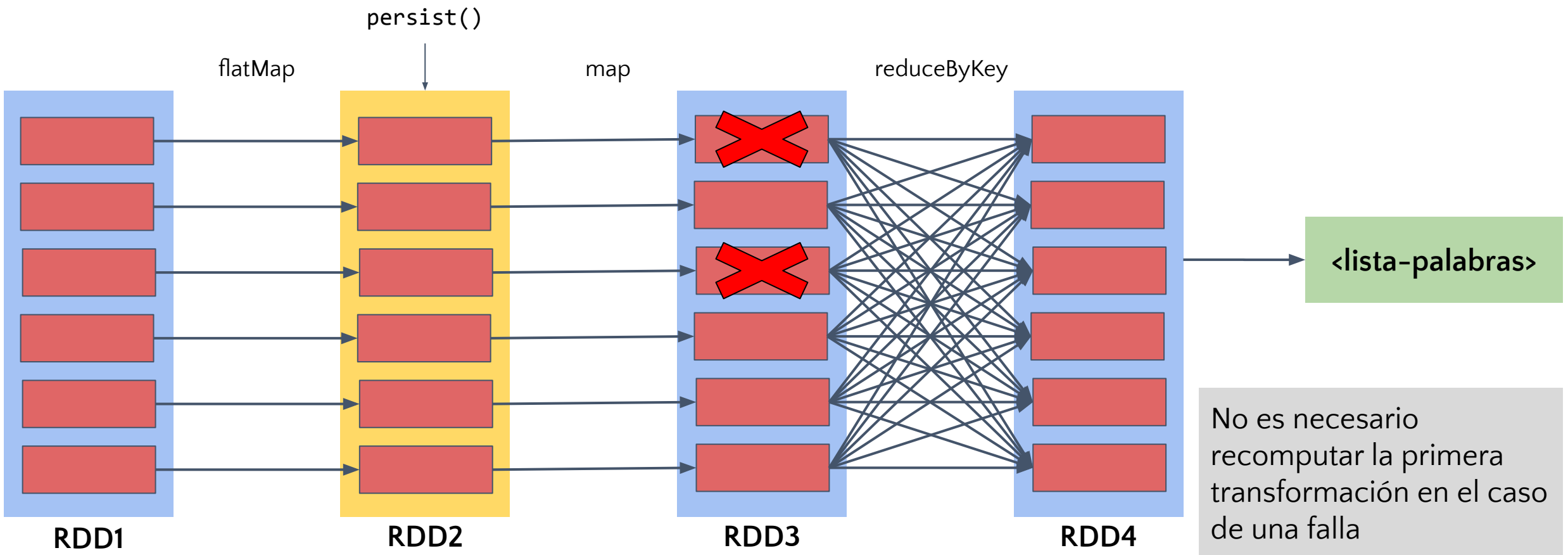
Spark - Persist



Spark - Persist



Spark – Persist



No es necesario recomputar la primera transformación en el caso de una falla

Spark – Persist

Nivel de persistencia	Descripción
`MEMORY_ONLY`	Almacena el contenido del rdd 100% en memoria. Si no alcanza el espacio en memoria usa LRU para desalojar
`MEMORY_AND_DISK`	Almacena lo que pueda en memoria, haciendo spill a disco de las partes menos usadas para las que no alcanza la memoria
`<LEVEL>_2`	Igual que `<LEVEL>`, replicando las particiones persistidas x2. Ejemplo: `MEMORY_AND_DISK_2`
`<LEVEL>_SER`	Igual que `<LEVEL>`, pero utilizando una versión serializada del rdd que ocupa menos memoria a costo de mayor uso de cpu. Ejemplo: `MEMORY_AND_DISK_SER`
`<LEVEL>_SER_2`	Igual que `<LEVEL>_SER`, pero replicando pariticiones persistidas x2. Ejemplo `MERMORY_ONLY_SER_2`
`OFF_HEAP`	Utiliza estructuras fuera de heap

Spark – Tipos RDD

Existen 3 tipos de RDD que exponen métodos diferentes. En scala es necesario importar conversiones implicitas para usarlos. En python, en cambio, todos los métodos están siempre disponibles, pero fallan si el RDD no es del tipo correcto

- RDD normales (vistos hasta ahora)
- RDD numéricos
- RDD clave-valor

Spark – RDD Numérico

Son rdds cuyos valores son numéricos. Exponen un conjunto de métodos exclusivos:

- `rdd.histogram(<buckets>)`
- `rdd.mean()`
- `rdd.meanApprox(<timeout>,<confidence>)`
- `rdd.min()`
- `rdd.max()`
- `rdd.stdev()`
- `rdd.stats()`

Spark - RDD clave-valor

Son RDDs que contienen tuplas de dos elementos. También llamados **Pair RDD**. El primero de los elementos se considera clave del rdd.

```
preferencias = [('fisica','maxi'),
                ('quimica','martin'),
                ('fisica','juan'),
                ('quimica', 'pato')]

rdd = sc.parallelize(preferencias)
```

Tener un rdd estructurado de esta manera nos da acceso a nuevas funciones (algunas ya introducidas). En general se utilizan para:

- Agrupar información similar
- Agregaciones
- **Joins de diferentes datasets**

Spark - RDD clave-valor

Los **Pair RDD** pueden ser sujetos a las mismas transformaciones y acciones que un RDD normal.

```
preferencias = [('fisica','maxi'),
                ('quimica','martin'),
                ('fisica','juan'),
                ('quimica', 'pato')]

rdd = sc.parallelize(preferencias)
```

```
rdd.filter(lambda tuple: tuple[0] == 'fisica')
```

```
rdd.map(lambda tuple: (tuple[0], tuple[0].upper()))
```

Spark – RDD clave-valor – Transformaciones

Suponiendo que tenemos un **RDD = {(1,2),(3,4),(3,6)}**

Funcion	Descripción	Ejemplo	Resultado
`reduceByKey(func)`	Reduce todos los valores por key	`rdd.reduceByKey( lambda x,y: x + y)`	`RDD: {(1,2), (3,10)}`
`groupByKey()`	Agrupa los resultados dentro de la misma key	`rdd.groupByKey()`	`RDD: {(1,[2]), (3,[4,6])}`
`mapValues(func)`	Aplica una función de transformación a cada elemento por clave	`rdd.mapValues( lambda x: x + 1)`	`RDD: {(1,3), (3,5), (3,7)}`
`keys()`	Devuelve un rdd solo con las claves	`rdd.keys()`	`RDD: {1,3,3}`
`values()`	Devuelve un rdd solo con los valores	`rdd.values()`	`RDD: {2,4,6}`

Spark – RDD clave-valor – Transformaciones

Suponiendo que tenemos un **RDD1 = {(1,2),(3,4),(3,6)]} y RDD2 = {(3,9)}**

Funcion	Descripción	Ejemplo	Resultado
`join(<pair rdd>)`	Realiza un inner join entre los rdd por clave	`rdd1.join(rdd2)`	`RDD: {(3,(4,9), (3,(6,9))}`
`cogroup(<pair rdd>)`	Agrupa los rdd por clave	`rdd1.cogroup(rdd2)`	`RDD: {(1,([2],[])), (3,([4,6],[9]))}`

Spark - RDD clave-valor - Acciones

Suponiendo que tenemos un **RDD = {(1,2),(3,4),(3,6)}**

Funcion	Descripción	Ejemplo	Resultado
`countByKey()`	Retorna la cantidad de elementos por key	`rdd.countByKey()`	`[(1,1),(3,2)]`
`collectAsMap()`	Retorna todos los resultados como mapa	`rdd.collectAsMap`	`{1: 2, 3: 6}`
`lookup(<key>)`	Devuelve todos los valores que tengan la clave suministrada	`rdd.lookup(3)`	`[4,6]`

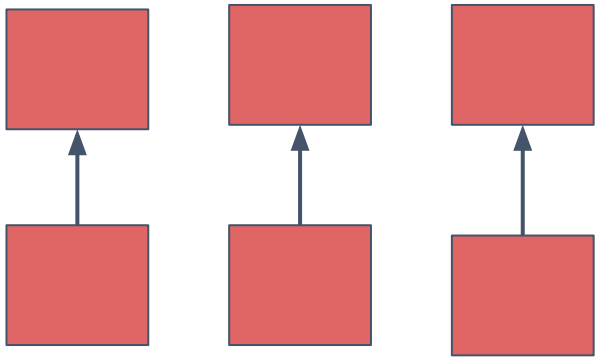
Spark – Round III

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

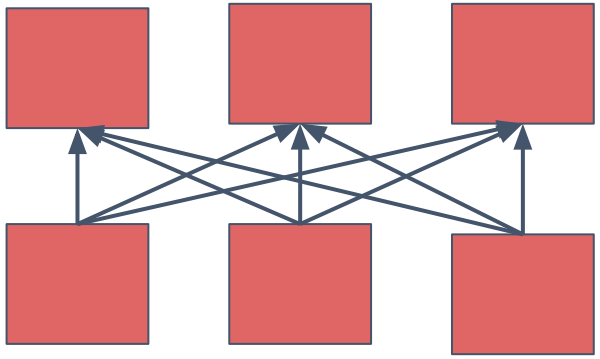
Spark – Transformaciones

Las transformaciones en spark pueden separarse en dos grandes grupos:

Con dependencia directa (*narrow dependencies*)

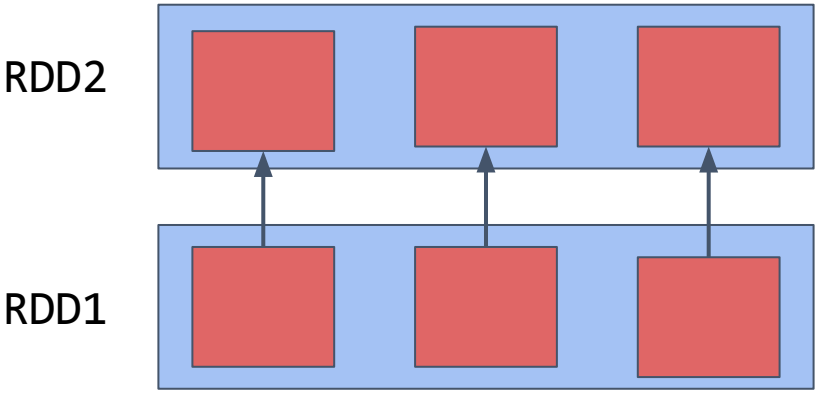


Con múltiples dependencias (*wide dependencies*)



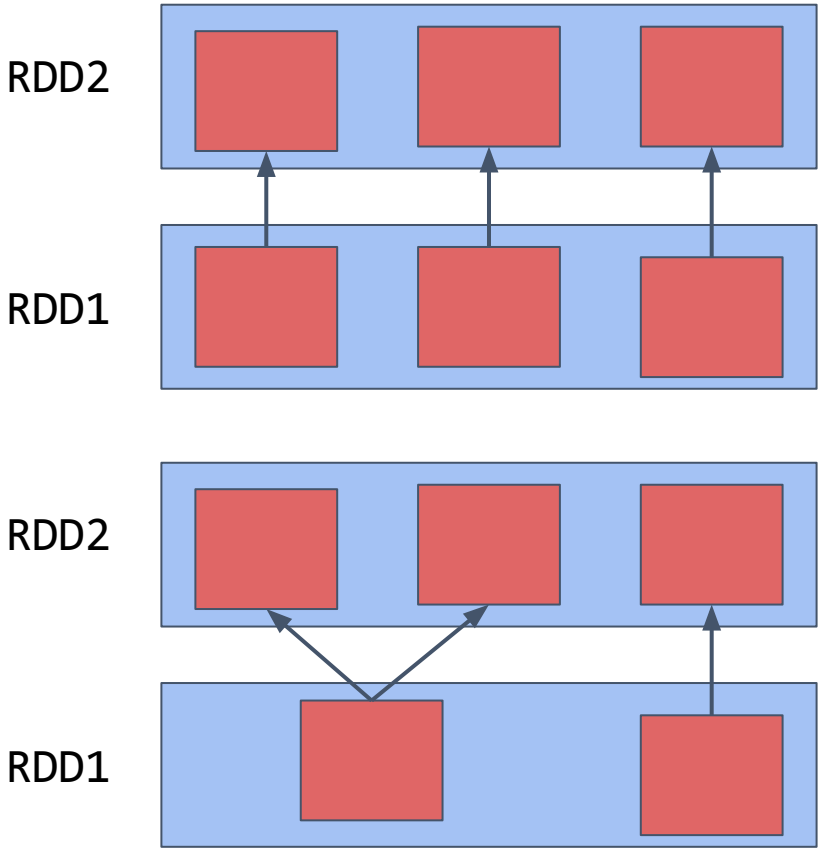
Spark – Narrow dependencies

Se dice que un **RDD** tiene *narrow dependencies* cuando el estado de cada **partición** del RDD depende de una única partición del RDD directamente anterior



Spark – Narrow dependencies

Se dice que un **RDD** tiene *narrow dependencies* cuando el estado de cada **partición** del RDD depende de una única partición del RDD directamente anterior



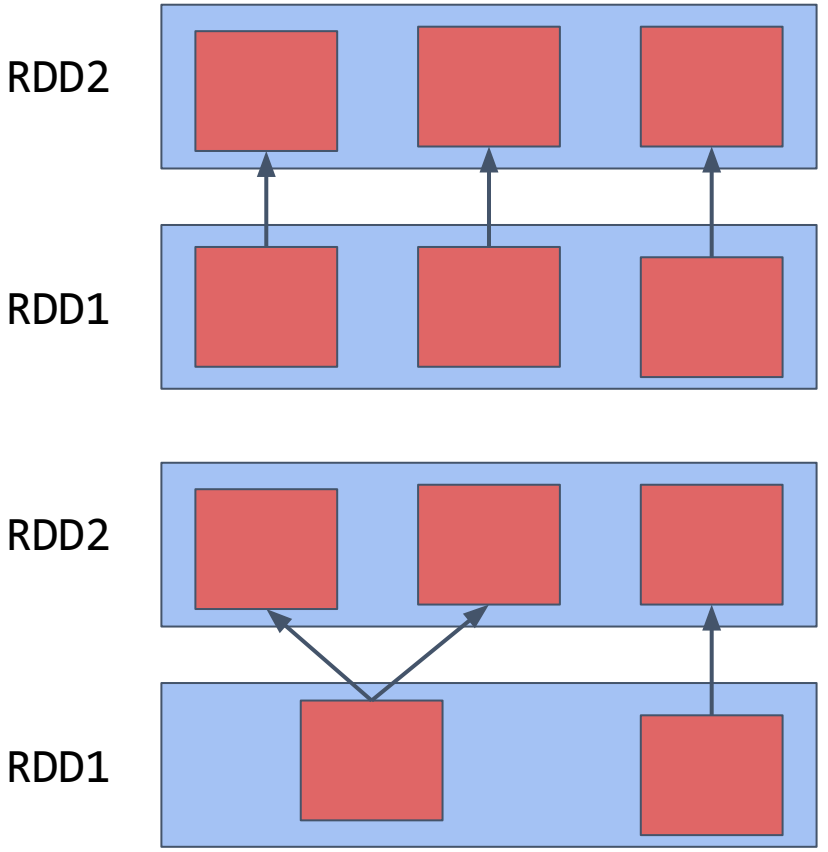
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark – Narrow dependencies

Se dice que un **RDD** tiene *narrow dependencies* cuando el estado de cada **partición** del RDD depende de una única partición del RDD directamente anterior.

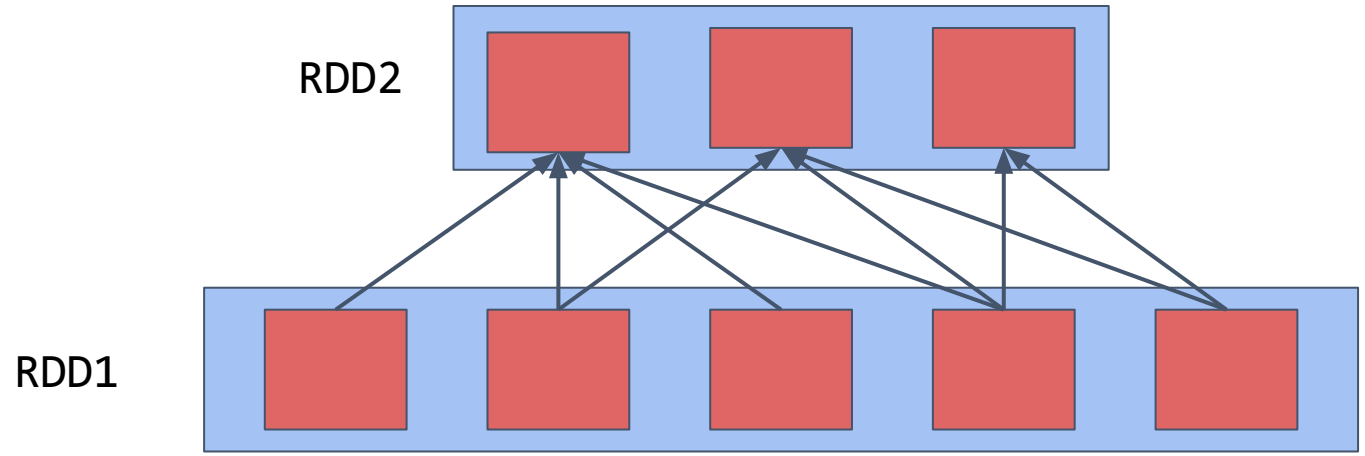
Algunas de las transformaciones que pueden generar este tipo de dependencias son:

- `map(func)`
- `filter(func)`
- `flatMap(func)`
- `sample()`



Spark – Wide dependencies

Cuando un **RDD** tiene *wide dependencies*, cada una de las **particiones** hijo depende de una o más particiones padre.

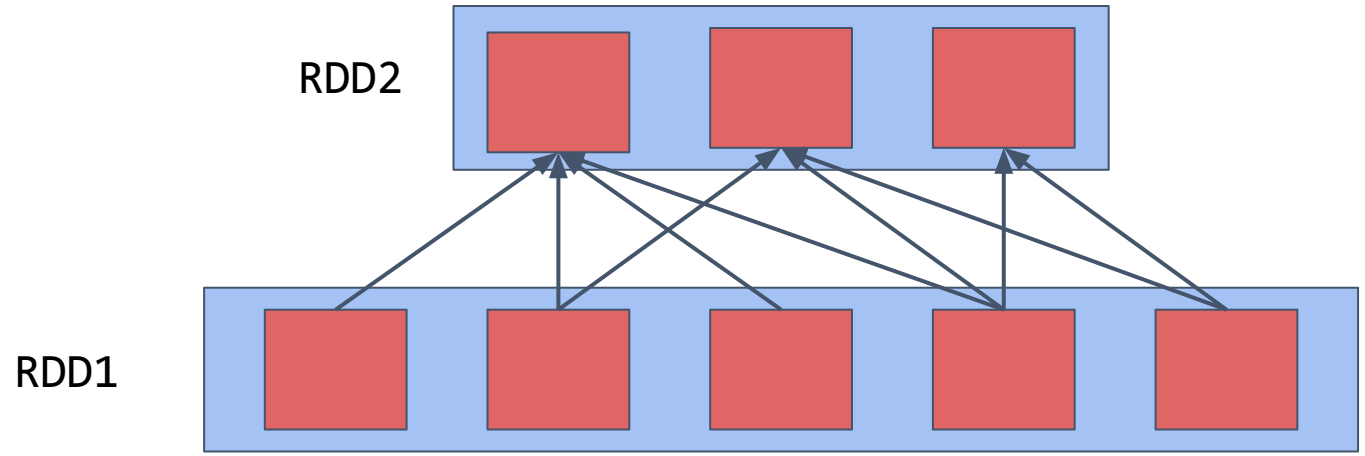


Spark – Wide dependencies

Cuando un **RDD** tiene *wide dependencies*, cada una de las **particiones** hijo depende de una o más particiones padre.

Algunos ejemplos de estas operaciones son:

- `groupByKey()`
- `reduceByKey()`
- `sortByKey()`
- `sortBy()`



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

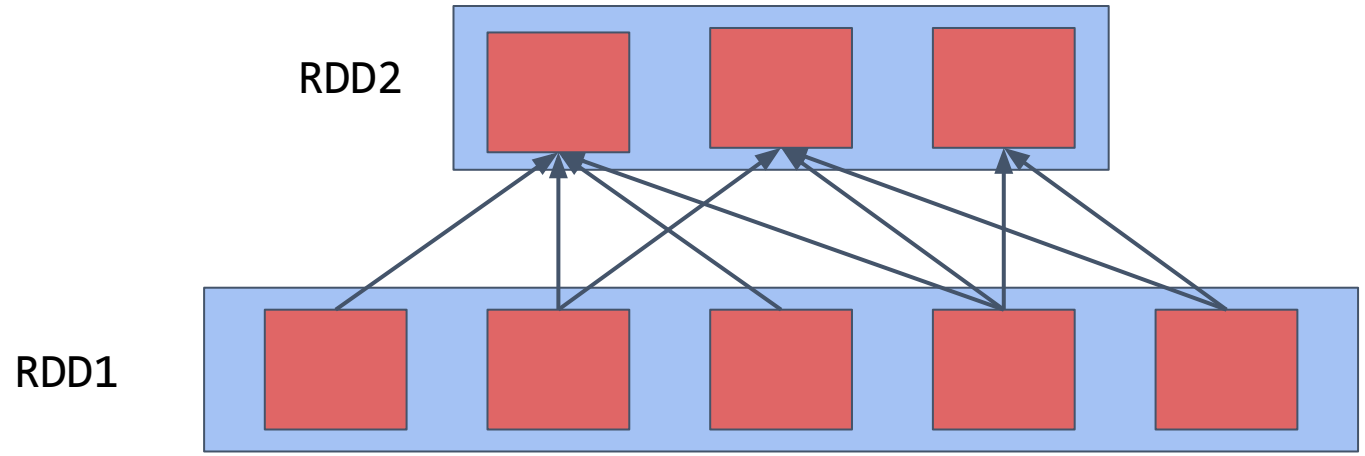
Spark – Wide dependencies

Cuando un **RDD** tiene *wide dependencies*, cada una de las **particiones** hijo depende de una o más particiones padre.

Este tipo de operaciones, involucra comunicación entre diferentes particiones y **siempre** causa **shuffle**

Algunos ejemplos de estas operaciones son:

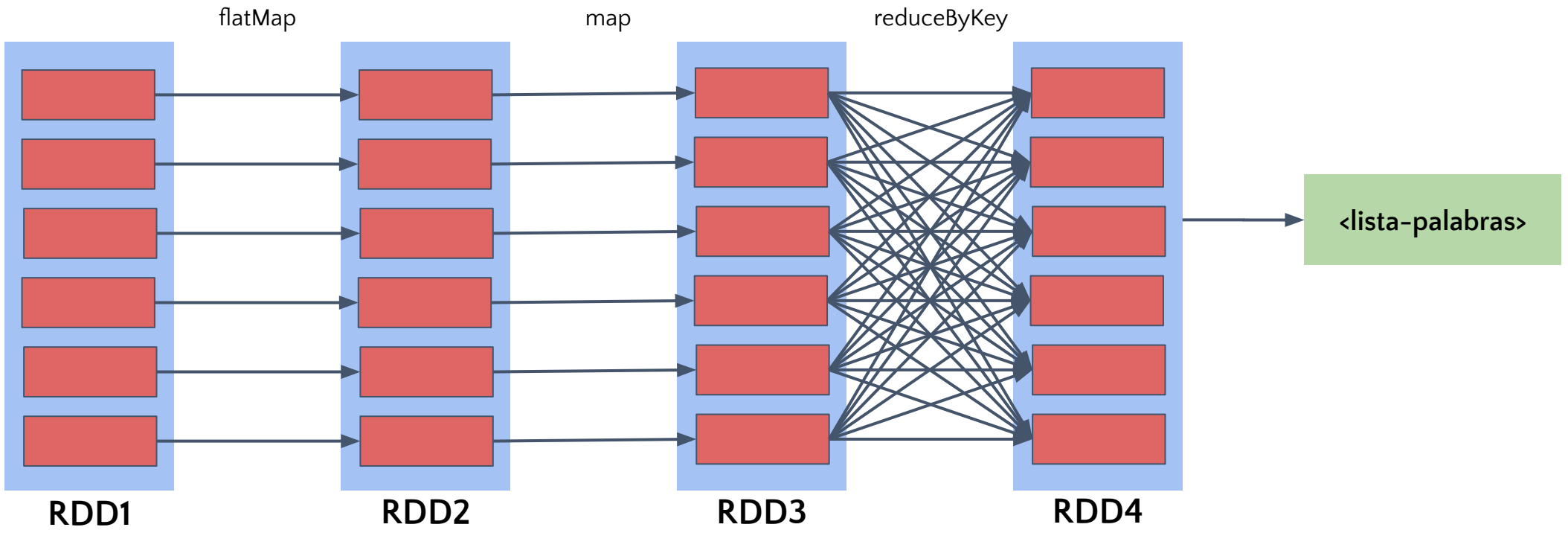
- `groupByKey()`
- `reduceByKey()`
- `sortByKey()`
- `sortBy()`



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

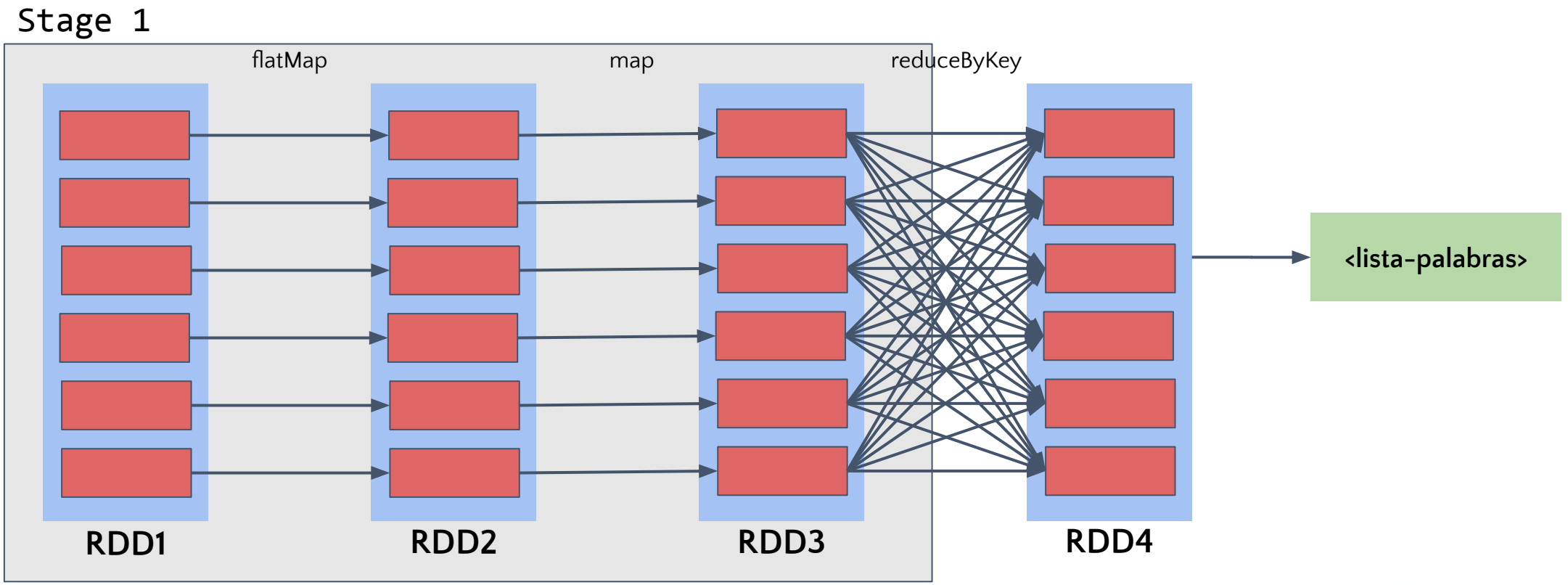
Spark – Shuffle

El shuffle sucede cuando se encuentran dependencias amplias (*wide dependencies*) en el árbol de dependencias de un RDD. Esto tiene serias **implicancias de performance** y de **tolerancia a fallas**.



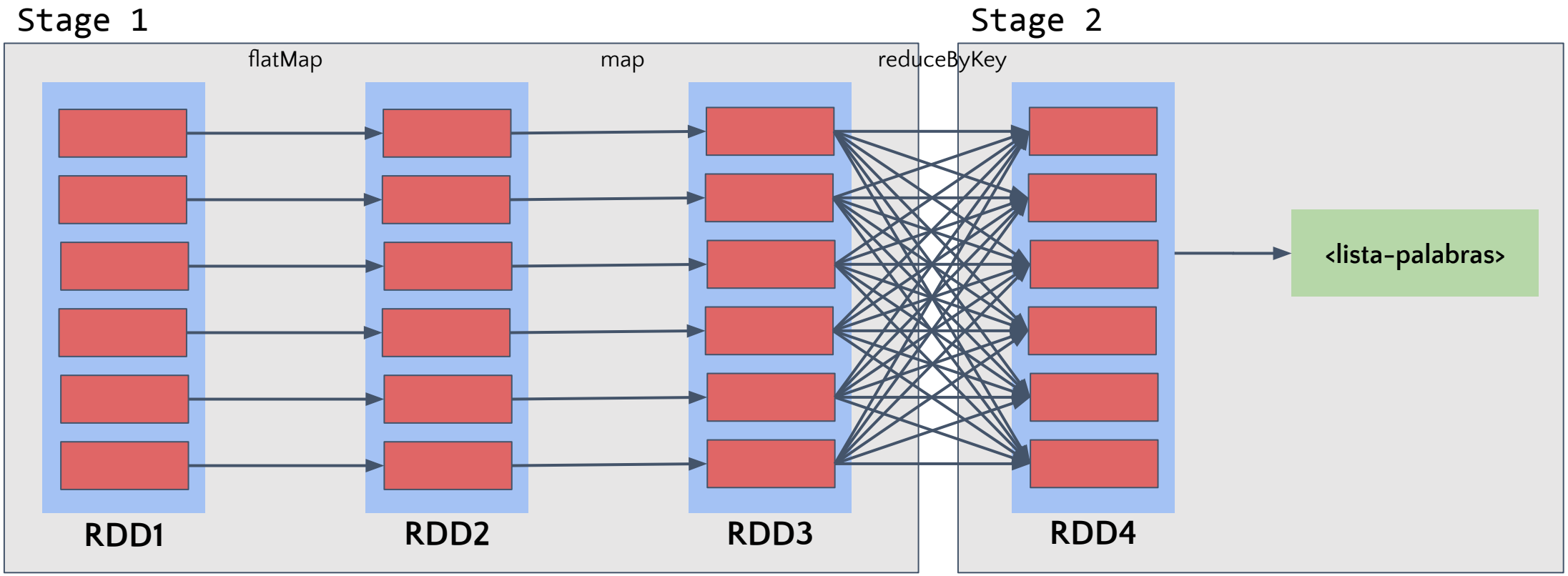
Spark - Shuffle

Spark divide su plan físico de ejecución entre shuffles. Cada etapa que separa un shuffle de otro se llama **stage**.



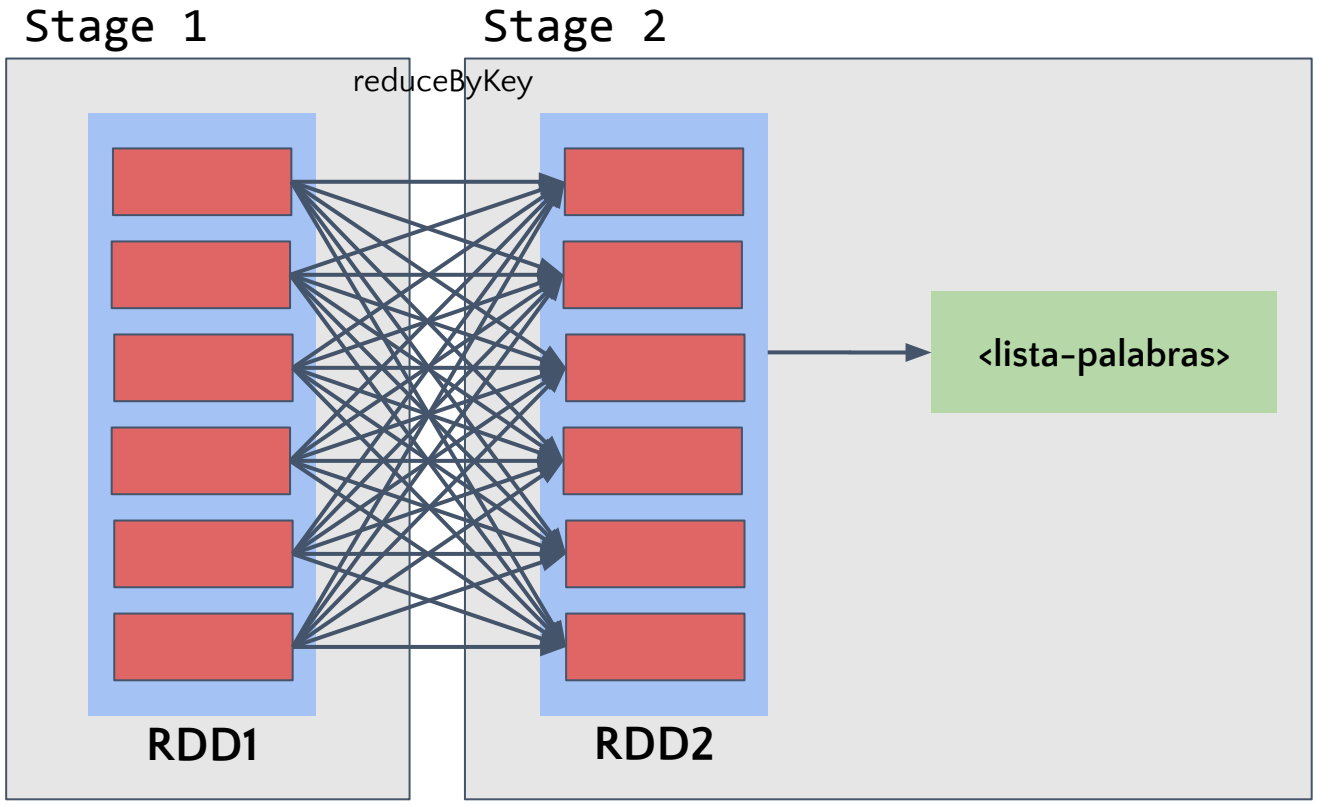
Spark – Shuffle

Spark divide su plan físico de ejecución entre shuffles. Cada etapa que separa un shuffle de otro se llama **stage**.



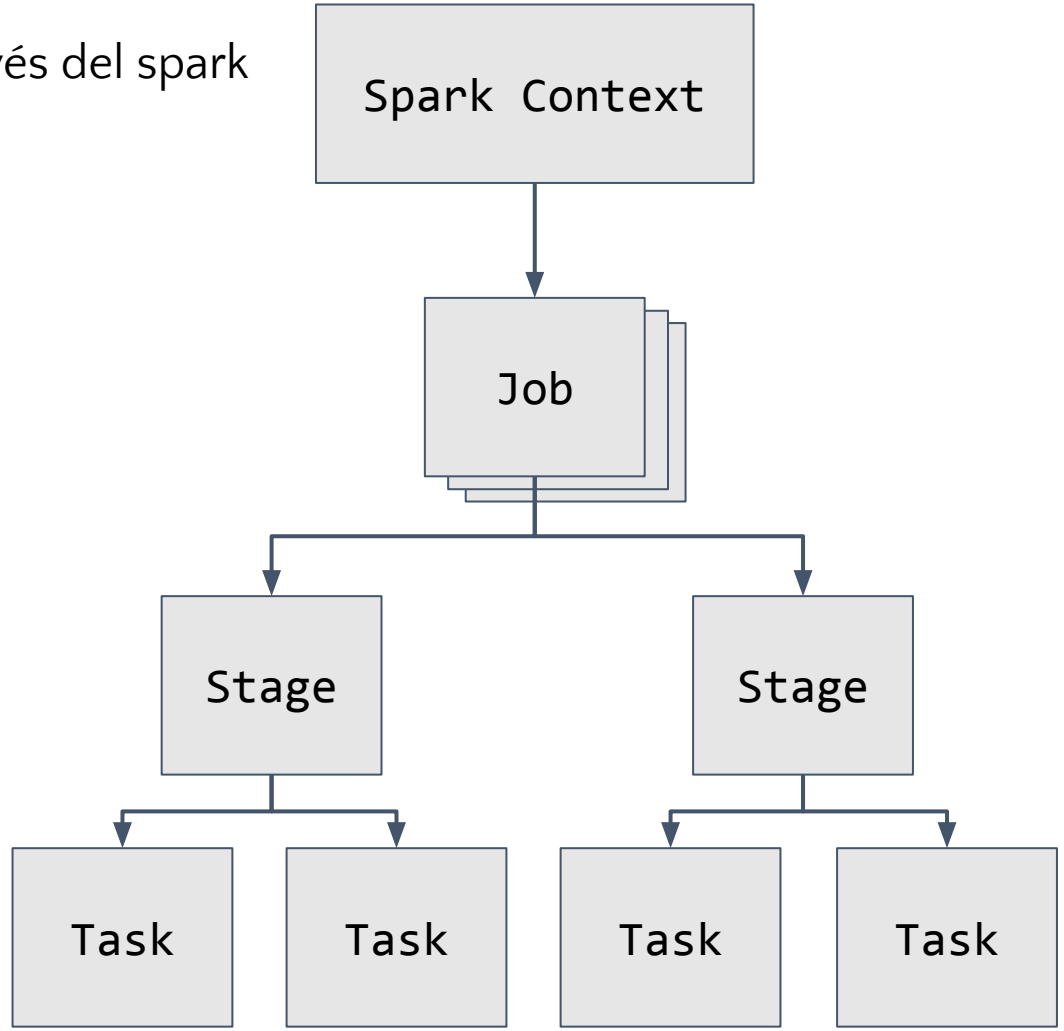
Spark – Shuffle

Todas las operaciones de un mismo stage se pueden compactar mediante **pipelining** (automáticamente), mientras que todas las operaciones posteriores deben esperar a que el **shuffle** termine en su totalidad.



Spark – Application

Inicio de la aplicación a través del spark context

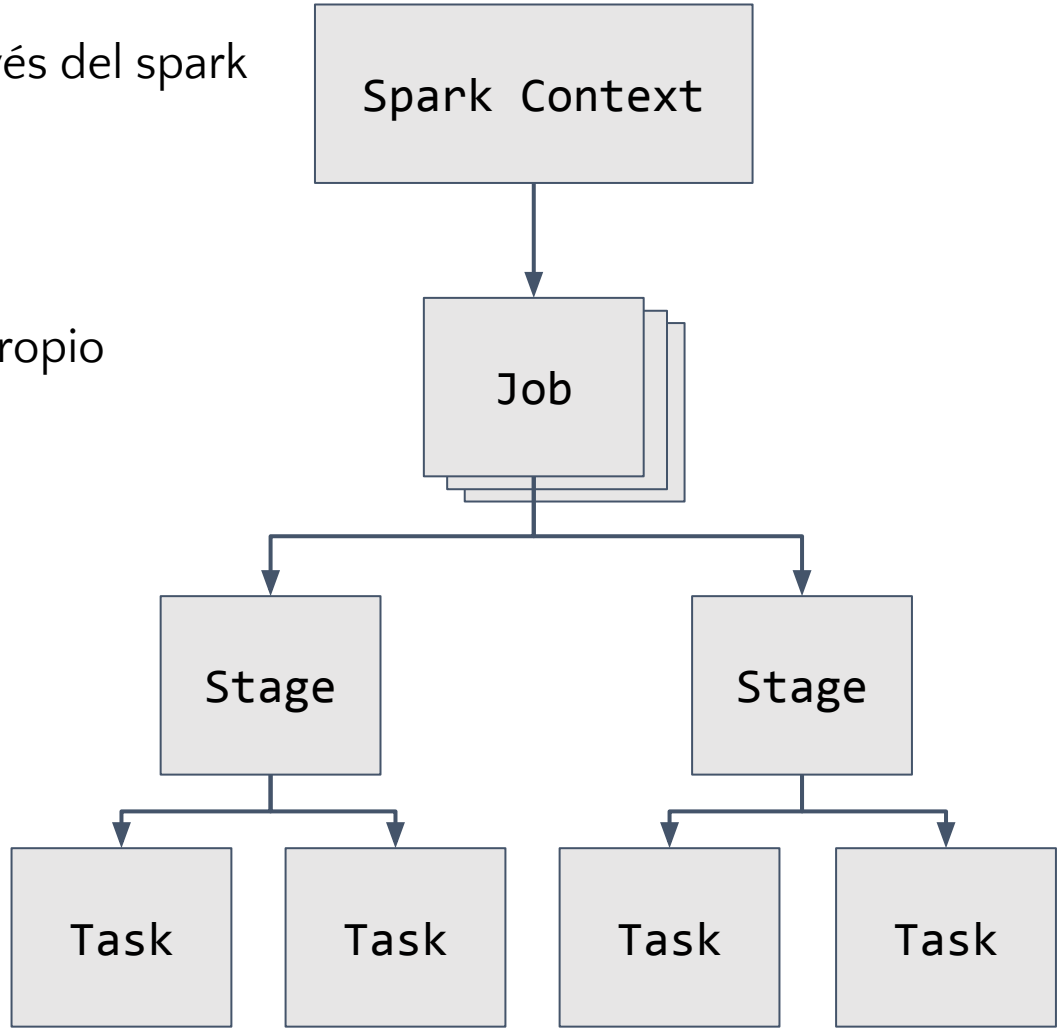


UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark – Application

Inicio de la aplicación a través del spark context

Cada **acción** lanza un job propio



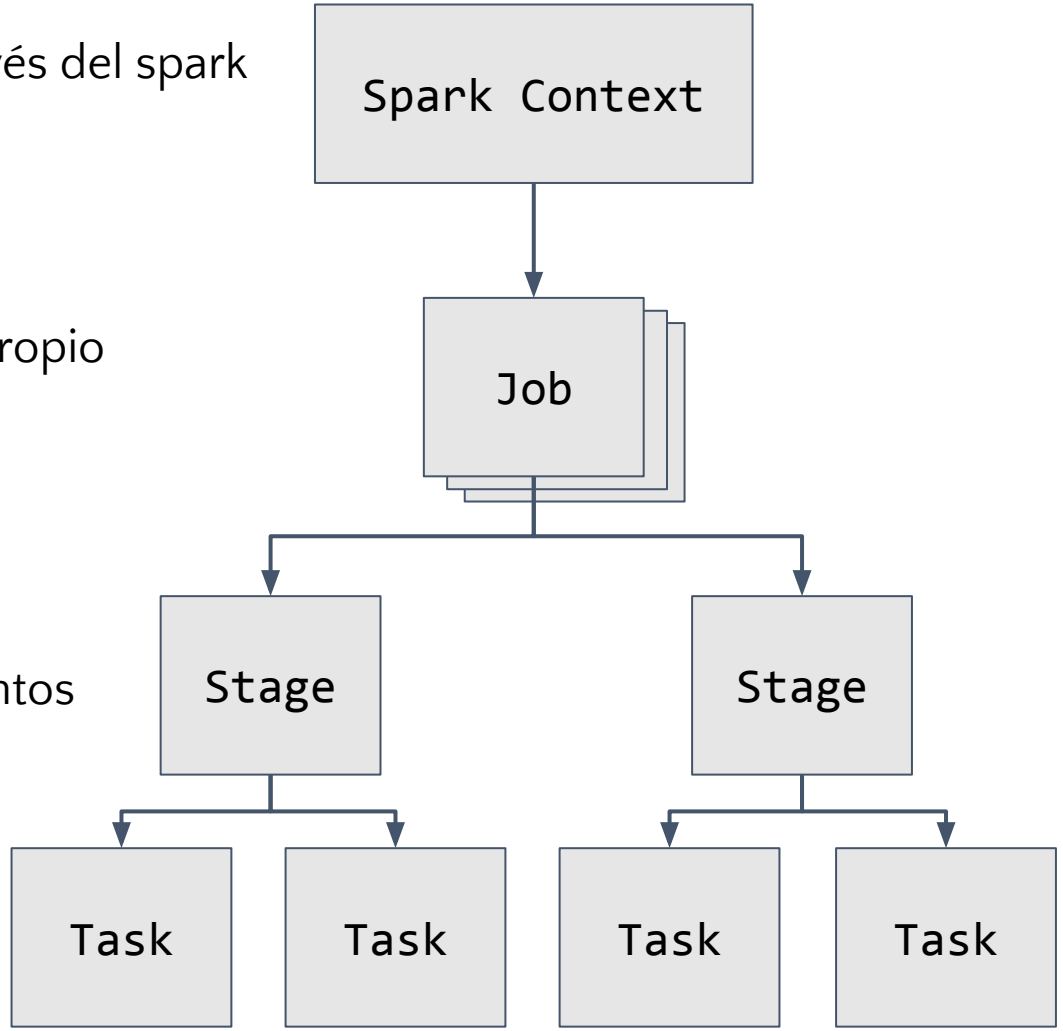
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Spark – Application

Inicio de la aplicación a través del spark context

Cada **acción** lanza un job propio

Las acciones se separan en **stages** separados por los puntos de **shuffle**



Spark – Y mucho más...

- Accumulators
- Broadcasts
- Partitioners
- Manejo de memoria
- Performance Python vs Java vs Scala
- Configuración
- StateListeners