Exploring and Analyzing Data with DataFrames

Janani Ravi
CO-FOUNDER, LOONYCORN

www.loonycorn.com

Overview

Explore data in Spark 2 using DataFrames

Transformations and actions on DataFrames

Built-in aggregate functions

Sampling, grouping, ordering data

Using broadcast variables and accumulators

SparkSession vs. SparkContext

Changes Starting Spark 2.0

Easier

Unifying Datasets and DataFrames, SQL support...

Faster

Optimize like a compiler, not a DBMS

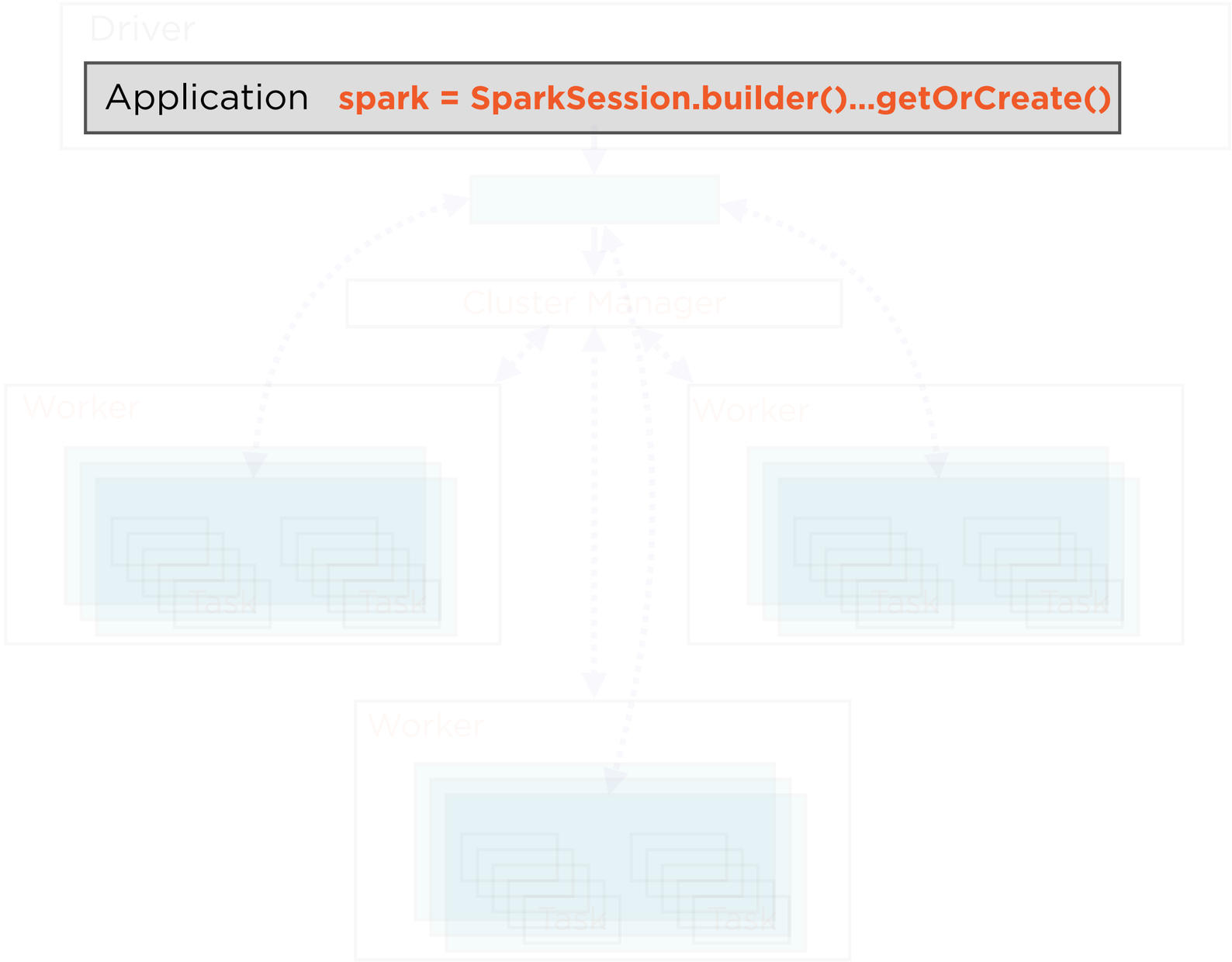
Ease of Use

SparkSession - simplified entry point

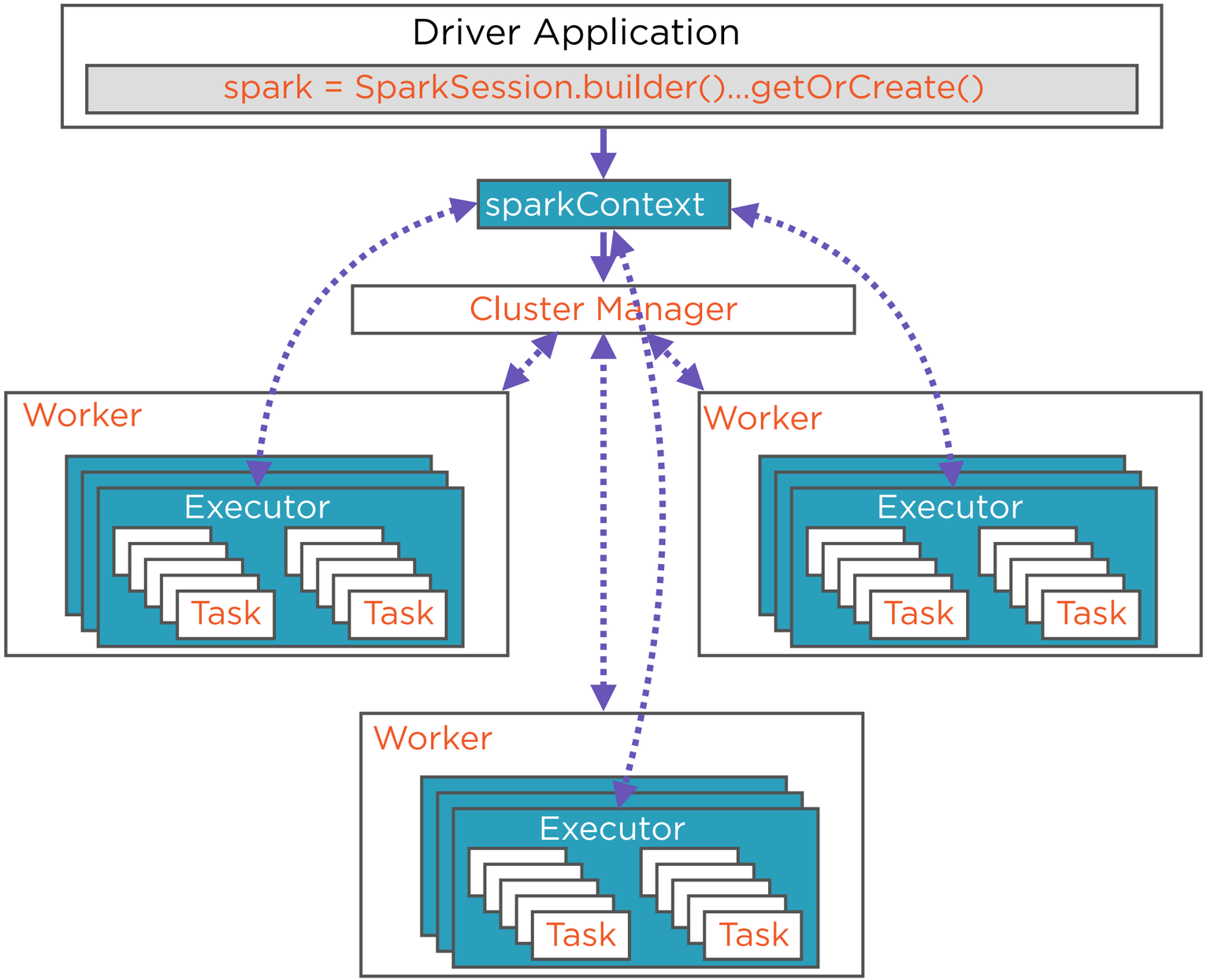
No confusion about which context to use

Subsumes SQLContext and HiveContext

Spark 2.x Architecture



Spark 2.x Architecture



Demo

Explore and analyze London crime data

- Transformations and actions on DataFrames
- Aggregating, grouping, sampling, ordering data

Accumulators and Broadcast Variables

Spark is written in Scala, and heavily utilizes closures

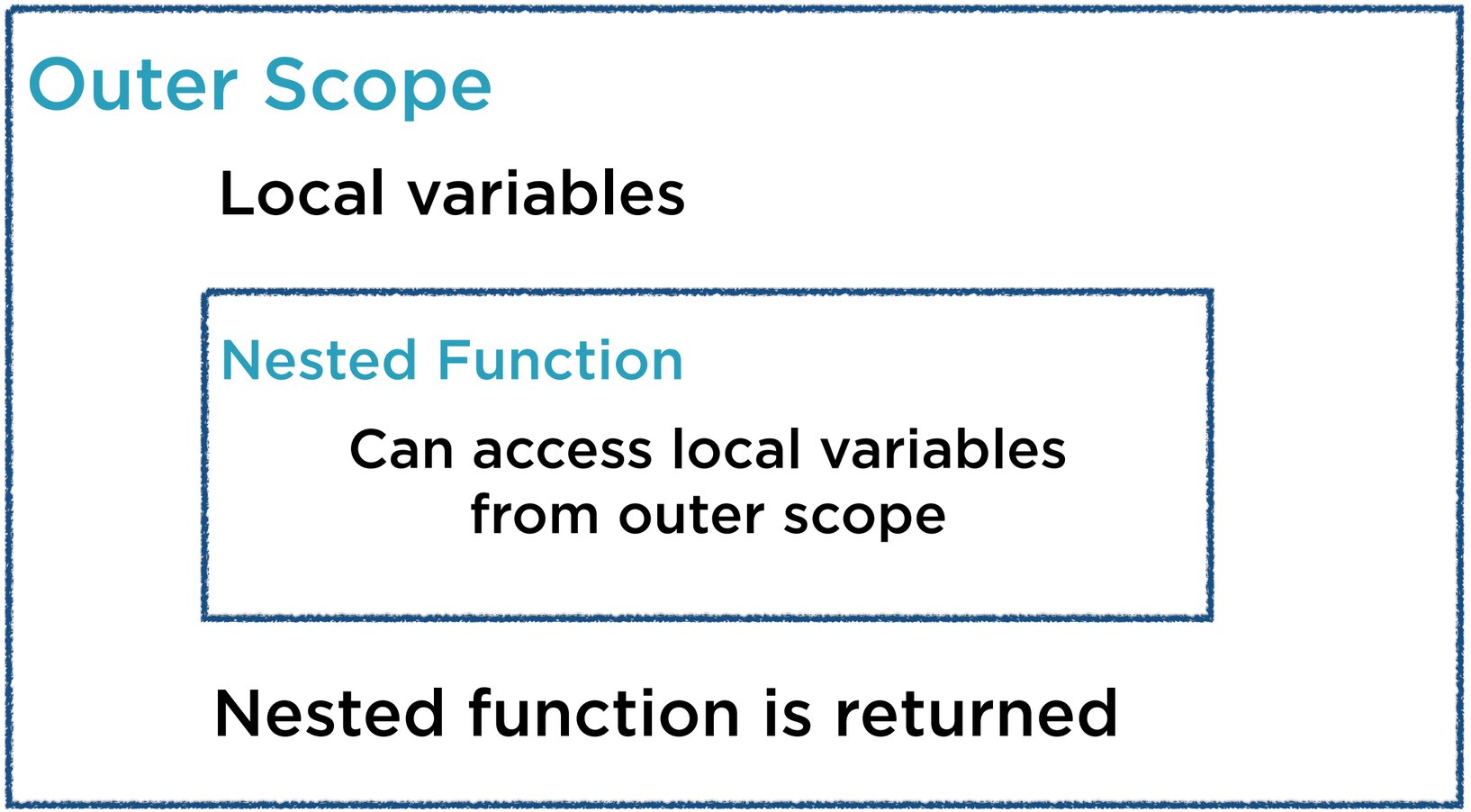
First Class Functions

A function can be stored in a variable or value

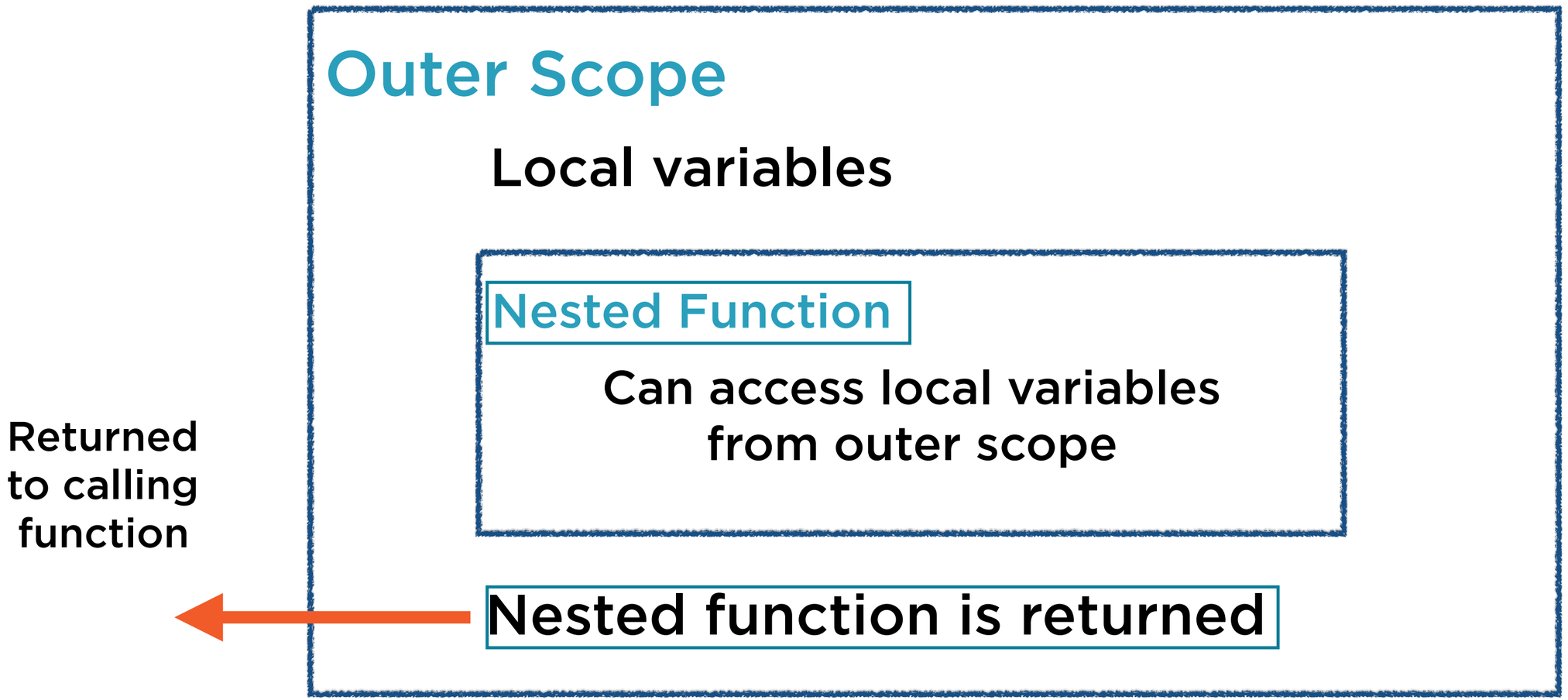
The return value of a function can be a function

A parameter of a function can be a function

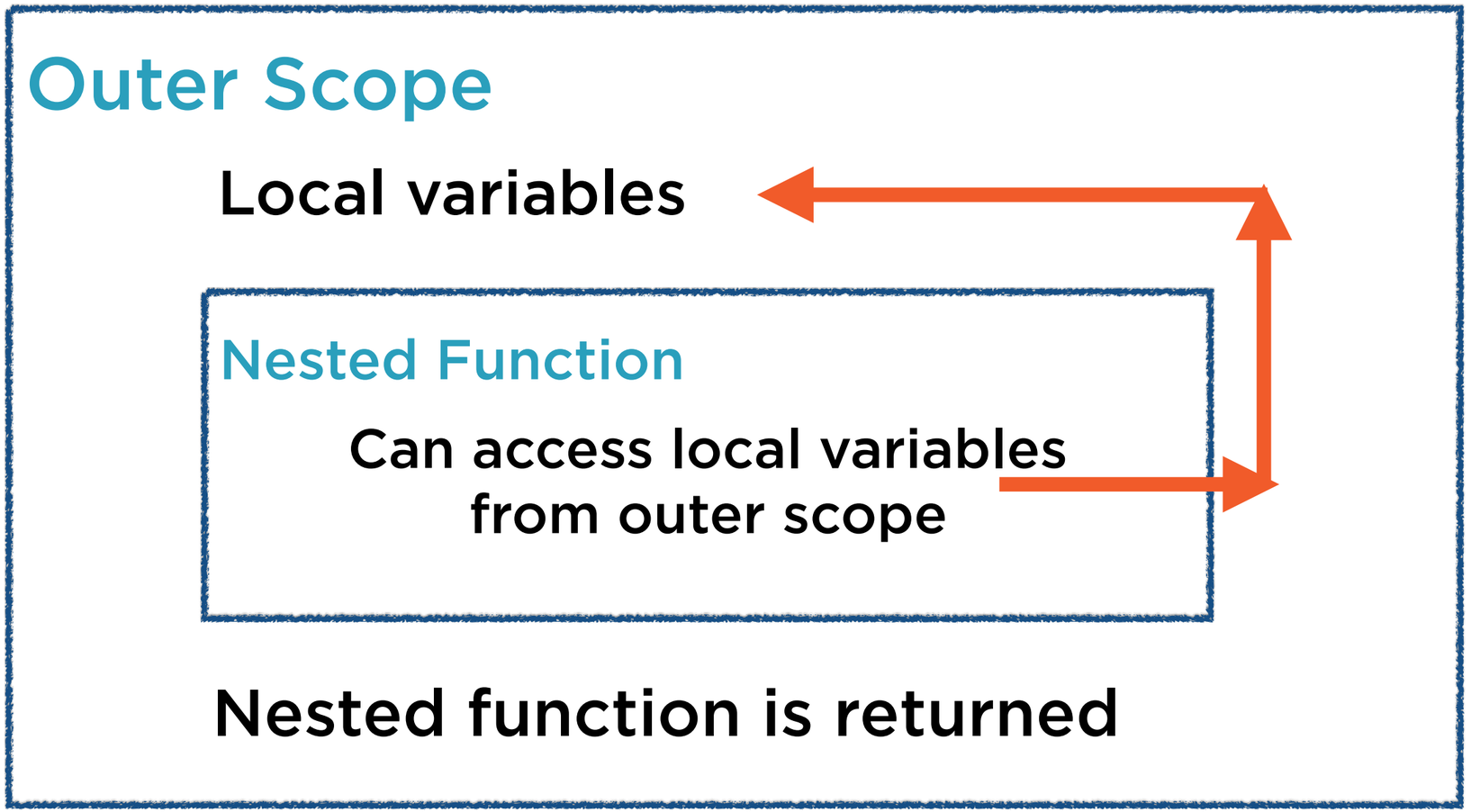
Closures


Outer Scope
Local variables
Nested Function
Can access local variables
from outer scope
Nested function is returned

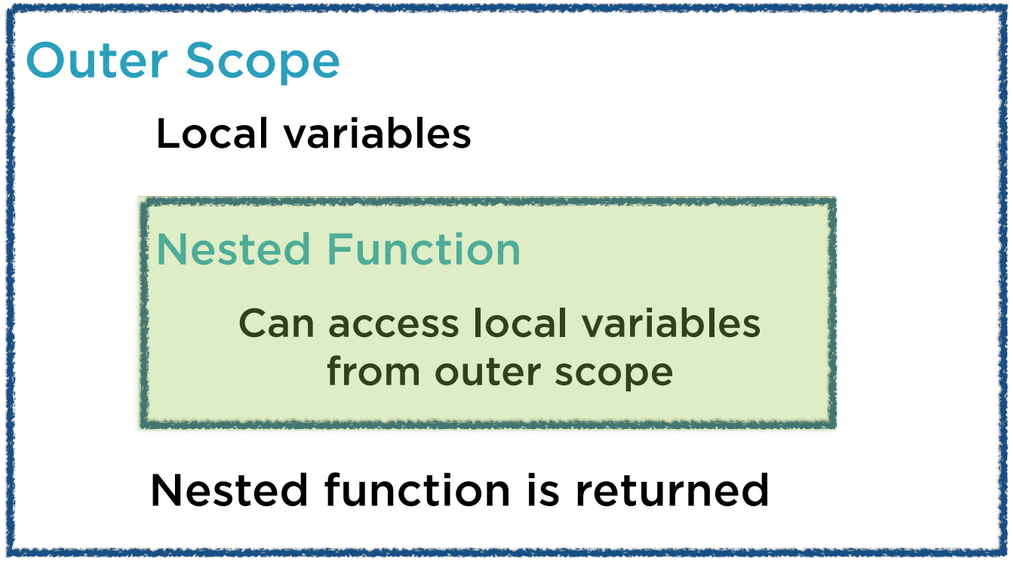
Closures


Outer Scope
Local variables
Nested Function
Can access local variables
from outer scope
Nested function is returned
Returned
to calling
function

Closures

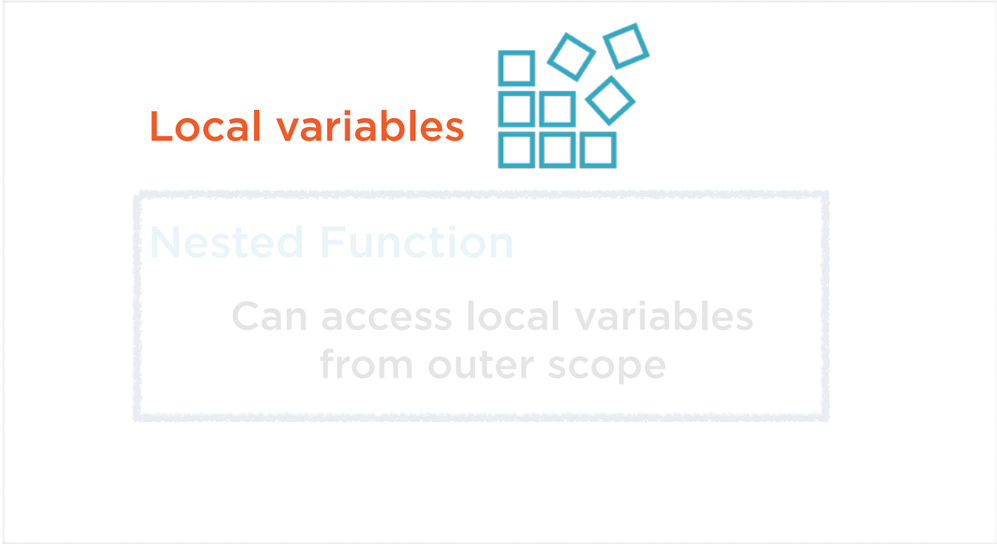

Outer Scope
Local variables
Nested Function
Can access local variables
from outer scope
Nested function is returned

Closures



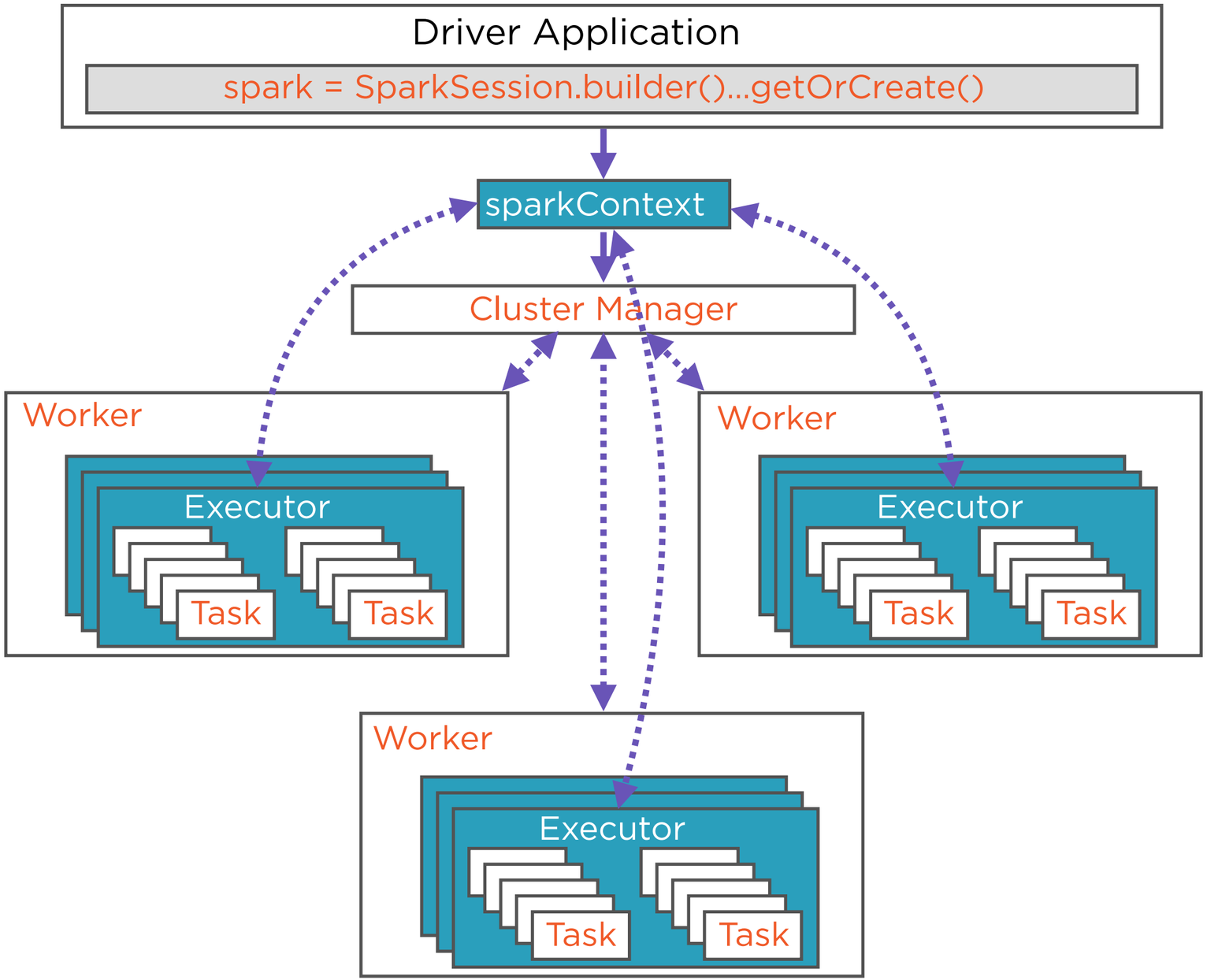
The returned nested function is called a closure

Closures



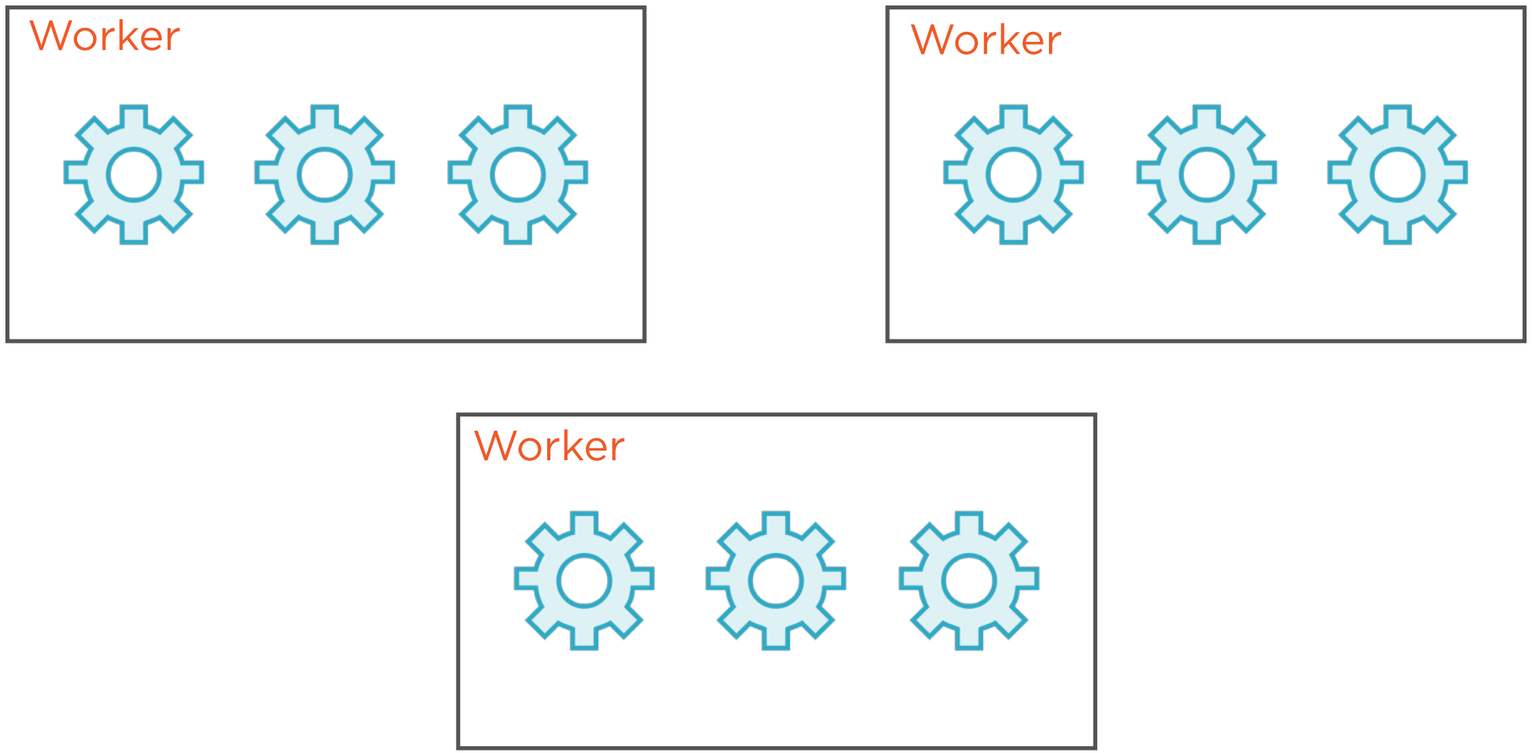
The closure retains its copies of local variables - even after the outer scope ceases to exist

Spark 2.x Architecture



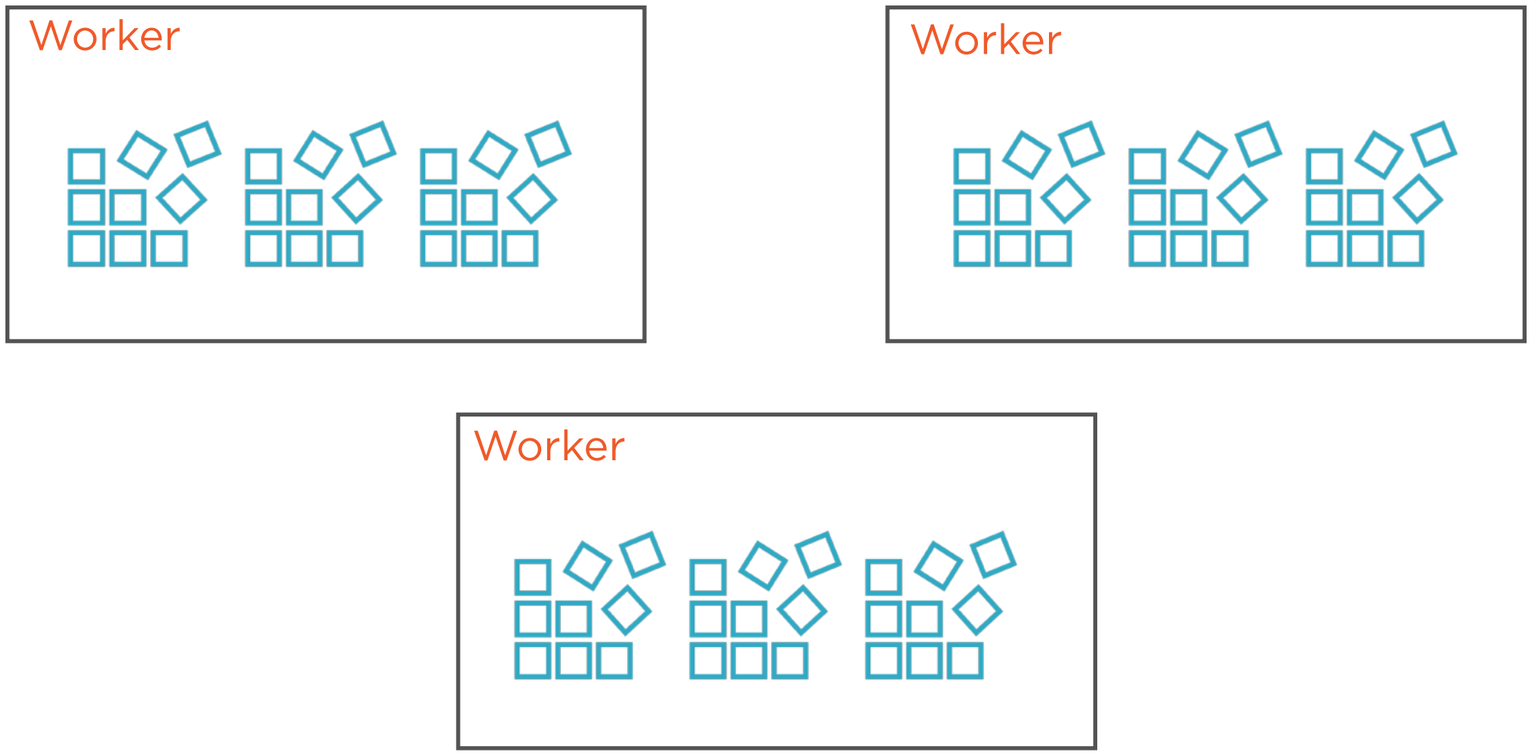
Spark 2.x Architecture

Tasks which run on individual workers are closures



Spark 2.x Architecture

Every task will contain a copy of the variables that it works on



```
“""MyScript.py"""

if __name__ == "__main__":
    delim = “ ”

    def myFunc(s):
        words = s.split(delim)
        return len(words)


    sc = SparkContext(…)

    sc.textFile("file.txt").map(myFunc)
```

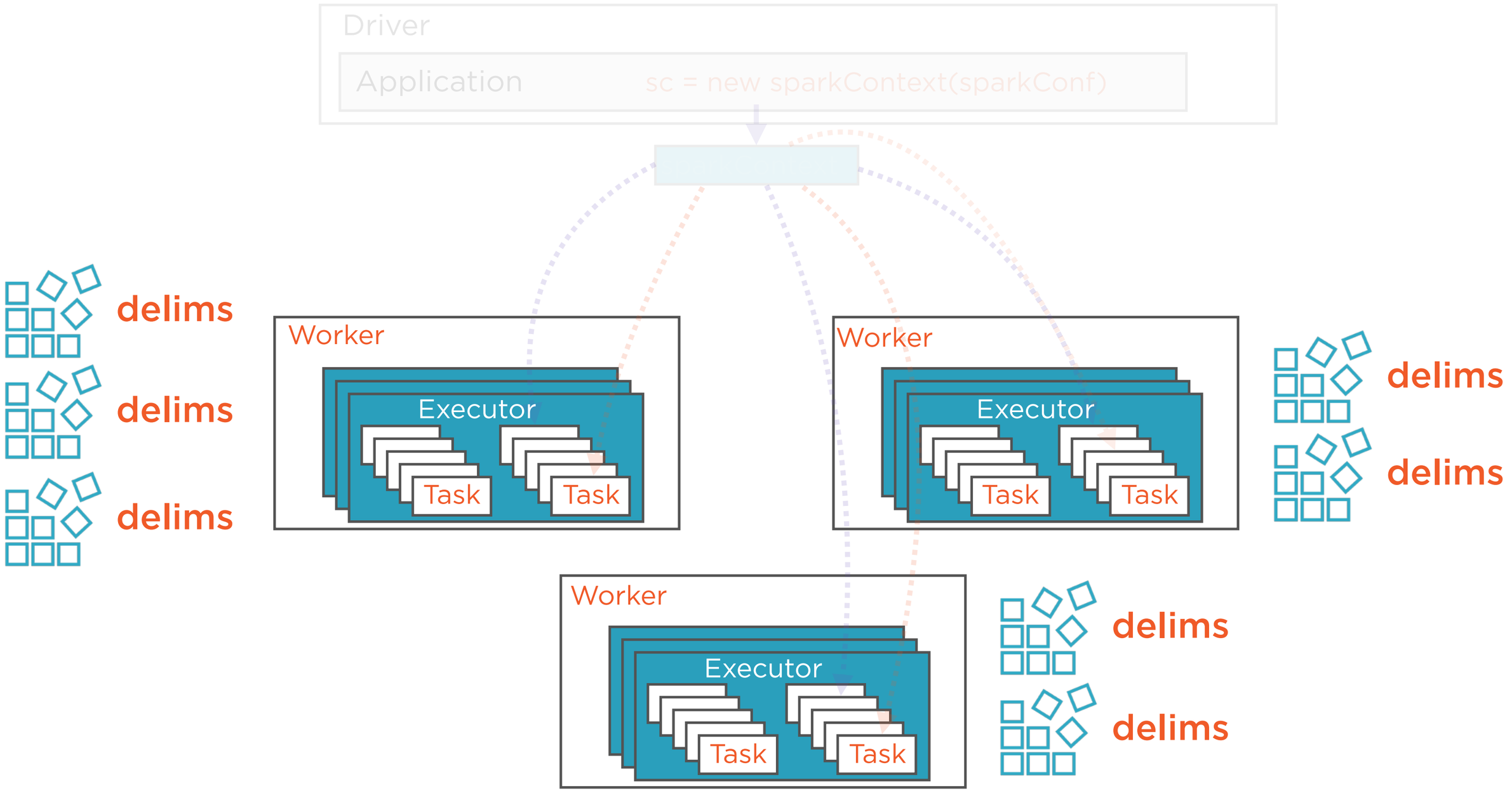
- Define a variable in outer scope
- Define a function
  - Reference that variable

- Map function to RDD

```
"""MyScript.py"""

if __name__ == "__main__":
    delim = " "

    def myFunc(s):
        words = s.split(delim)
        return len(words)


    sc = SparkContext(…)

    sc.textFile("file.txt").map(myFunc)
```

- Variable will be copied to each task on which map function runs
  - Part of closure
- This copying is done from master alone

One Copy Per Task



Closures

By default functions in Spark carry around copies of all variables

Each function copies each variable to each node it runs on

No updates sent back to master

Closures

1 copy per task

- Lots of copies passed around

Shuffling ~ further cost hit

All copying from master

- No copying from 1 task to another

Closures

Need shared variables across tasks?

- Broadcast variables
- Accumulators

Shared Variables

Broadcast Variables

Only 1 read-only copy per node
(not 1 copy per task)

Accumulators

Broadcast to workers but can be
modified by adding to it

Broadcast Variables

Shared, read-only variables

One copy per node

Not one copy per task

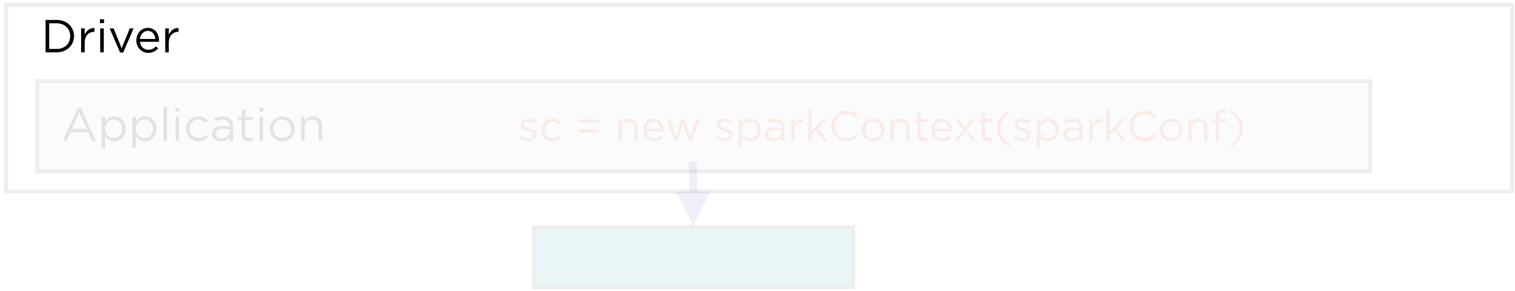
Broadcast Variables

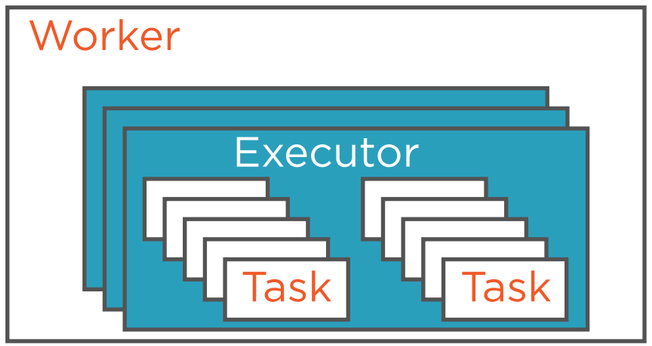
Distributed efficiently by Spark

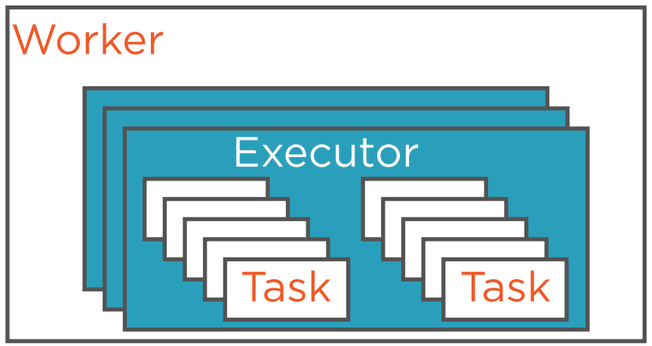
All nodes in cluster distribute

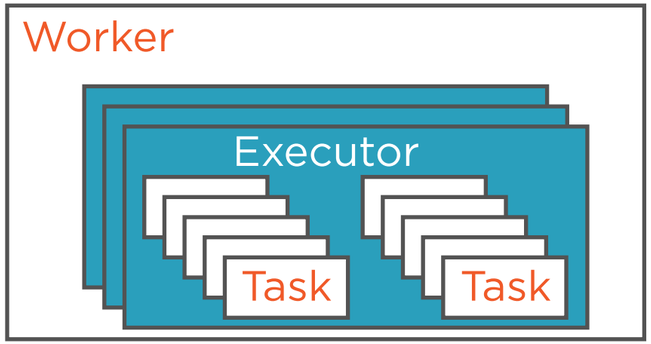
No shuffling

One Copy Per Node

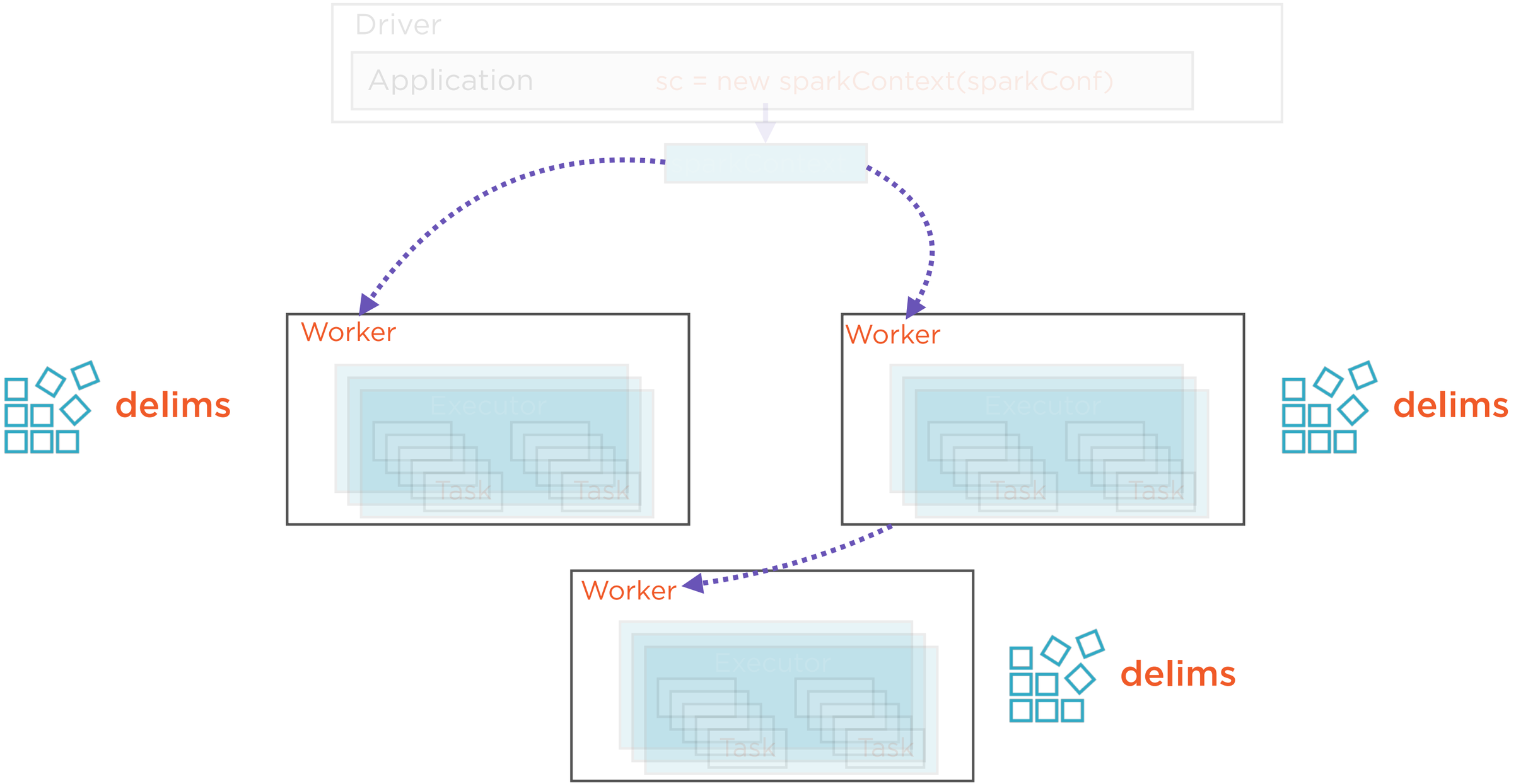








Peer-to-peer Copying Too



Broadcast Variables

Will be cached in-memory on each node

So, can be large, but not too large

Broadcast Variables

Use whenever tasks across stages need same data

Share dataset with all nodes

- training data in ML
- static lookup tables

Shared Variables

Broadcast Variables

Only 1 read-only copy per node
(not 1 copy per task)

Accumulators

Broadcast to workers but can be
modified by adding to it

Accumulator Variables

Read-write shared variables

Added associatively and commutatively

Commutativity: A + B = B + A

Associativity: A + (B+C) = (A+B) + C

Accumulator Variables

Spark native support for accumulators of type

- Long
- Double
- Collections

Can extend by subclassing AccumulatorV2

Accumulator Variables

Counters or sums

Workers can only modify state

Only the driver program can read state

Demo

Explore and analyze soccer data

- Joins using broadcast variables
- Sum using accumulators

Demo

Miscellaneous operations in Spark

- Creating custom accumulators
- Different join operations

Summary

Loading, cleaning and filtering data using DataFrames

Common transformations and actions

Built-in aggregate functions

Sampling, grouping, ordering data

Using broadcast variables and accumulators