Communication Patterns

Reza Zadeh

Matroid

ICME
INSTITUTE for COMPUTATIONAL & MATHEMATICAL ENGINEERING
at STANFORD UNIVERSITY

Spark

@Reza_Zadeh | http://reza-zadeh.com

Outline

Shipping code to the cluster

Shuffling

Broadcasting

Other programming languages

Outline

Shipping code to the cluster

Life of a Spark Program

1) Create some input RDDs from external data or parallelize a collection in your driver program.

2) Lazily ***transform*** them to define new RDDs using transformations like `filter()` or `map()`

3) Ask Spark to `cache()` any intermediate RDDs that will need to be reused.

4) Launch ***actions*** such as `count()` and `collect()` to kick off a parallel computation, which is then optimized and executed by Spark.

Example Transformations

`map()`	`intersection()`	`cartesion()`
`flatMap()`	`distinct()`	`pipe()`
`filter()`	`groupByKey()`	`coalesce()`
`mapPartitions()`	`reduceByKey()`	`repartition()`
`mapPartitionsWithIndex()`	`sortByKey()`	`partitionBy()`
`sample()`	`join()`	`...`
`union()`	`cogroup()`	`...`

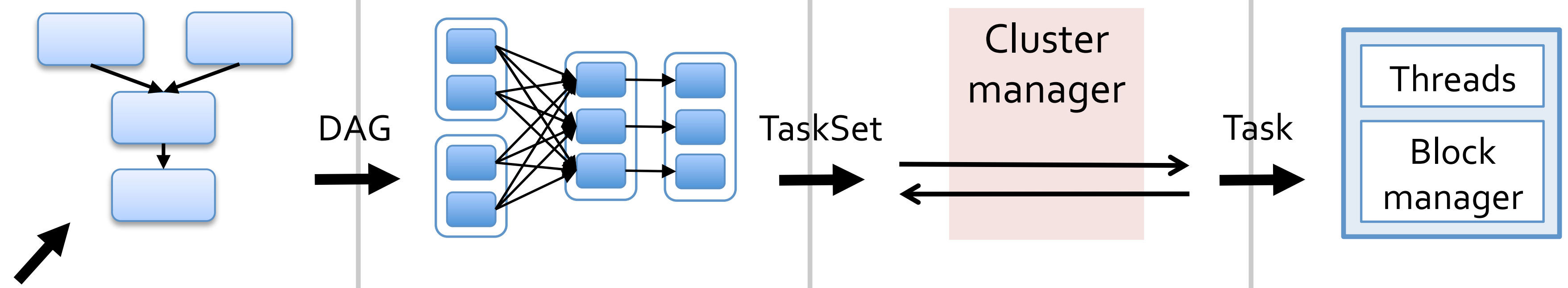
Example Actions

```
reduce()
collect()
count()
first()
take()
takeSample()
saveToCassandra()
takeOrdered()
saveAsTextFile()
saveAsSequenceFile()
saveAsObjectFile()
countByKey()
foreach()
...
```

Sending your code to the cluster

RDD → Stages → Tasks

RDD Objects | DAG Scheduler | Task Scheduler | Worker



```
rdd1.join(rdd2)
     .groupBy(…)
     .filter(…)
```

build operator DAG

split graph into *stages* of tasks

submit each stage as ready

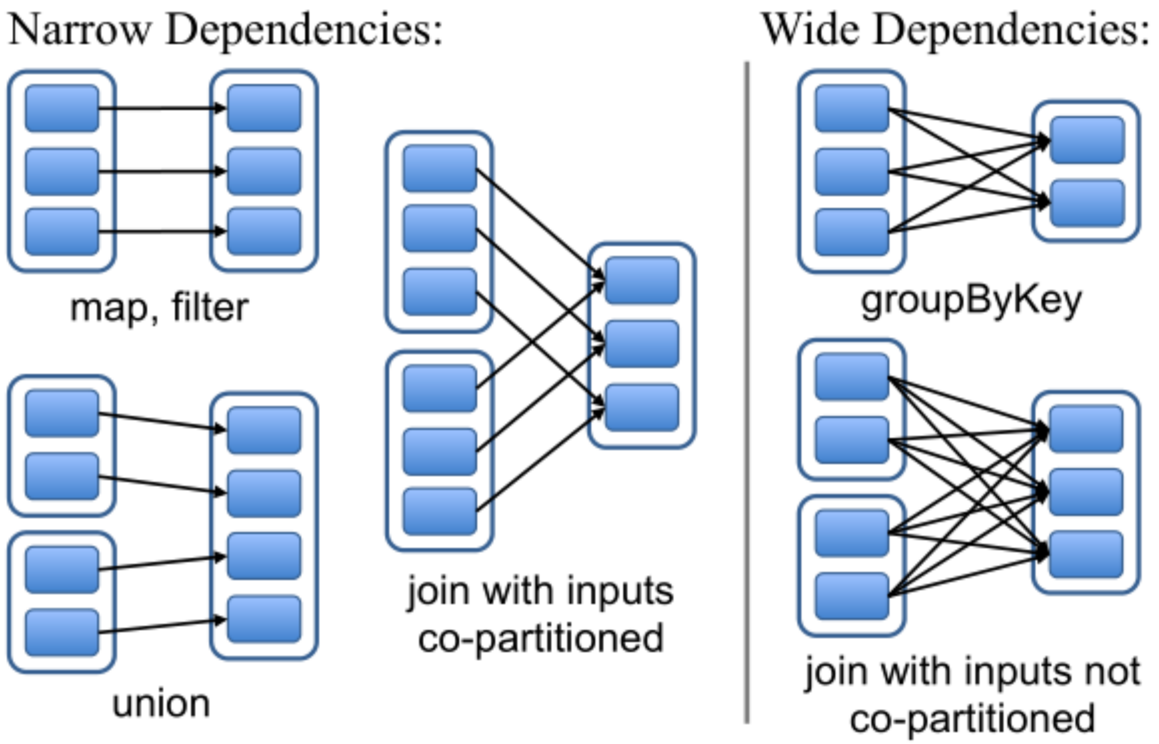
launch tasks via cluster manager

retry failed or straggling tasks

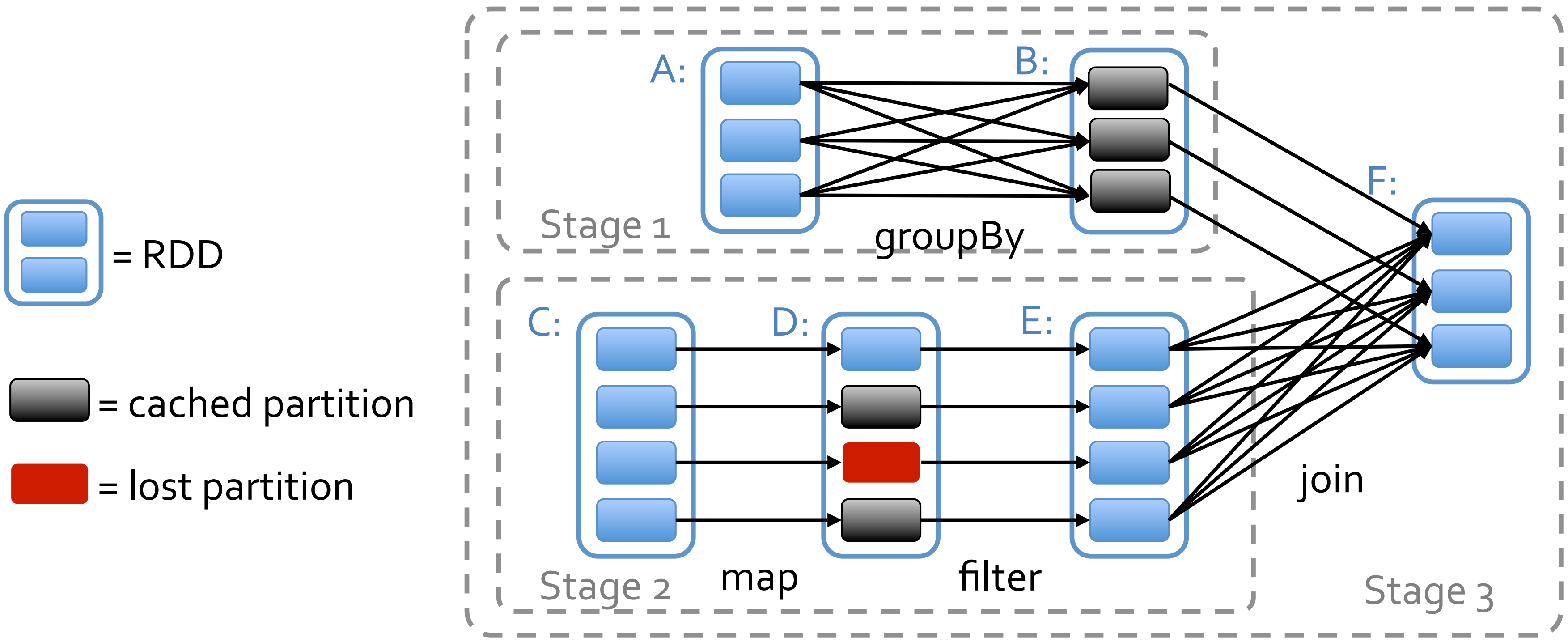
execute tasks

store and serve blocks

Communication Patterns


Narrow Dependencies:
map, filter
union
join with inputs co-partitioned
Wide Dependencies:
groupByKey
join with inputs not co-partitioned

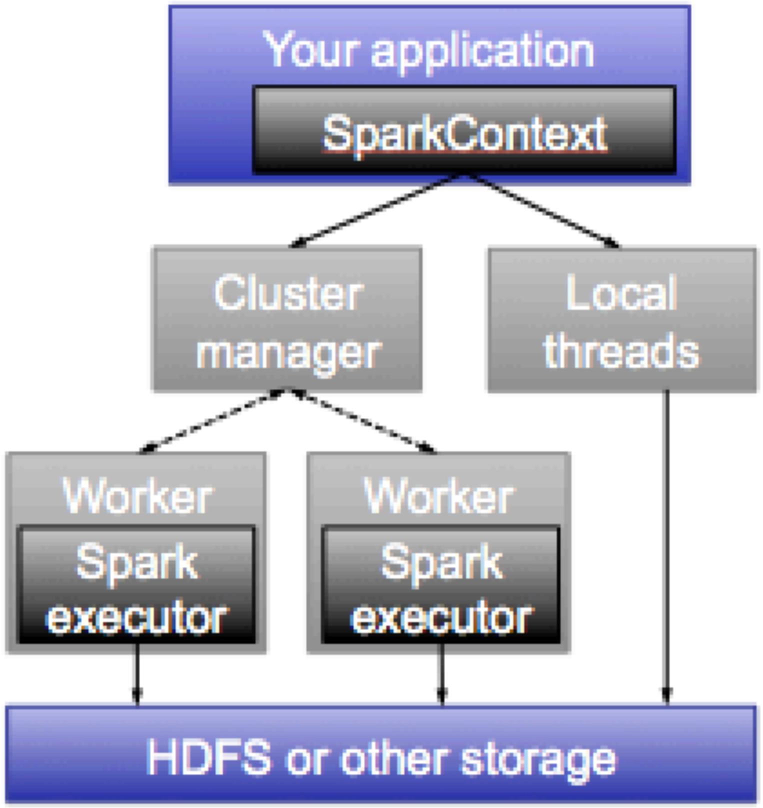
Example Stages



Talking to Cluster Manager

Manager can be:

- YARN
- Mesos
- Spark Standalone



Shuffling

Shuffle

=

groupByKey

sortByKey

reduceByKey

Sort: use advances in sorting single-machine memory-disk operations for all-to-all communication

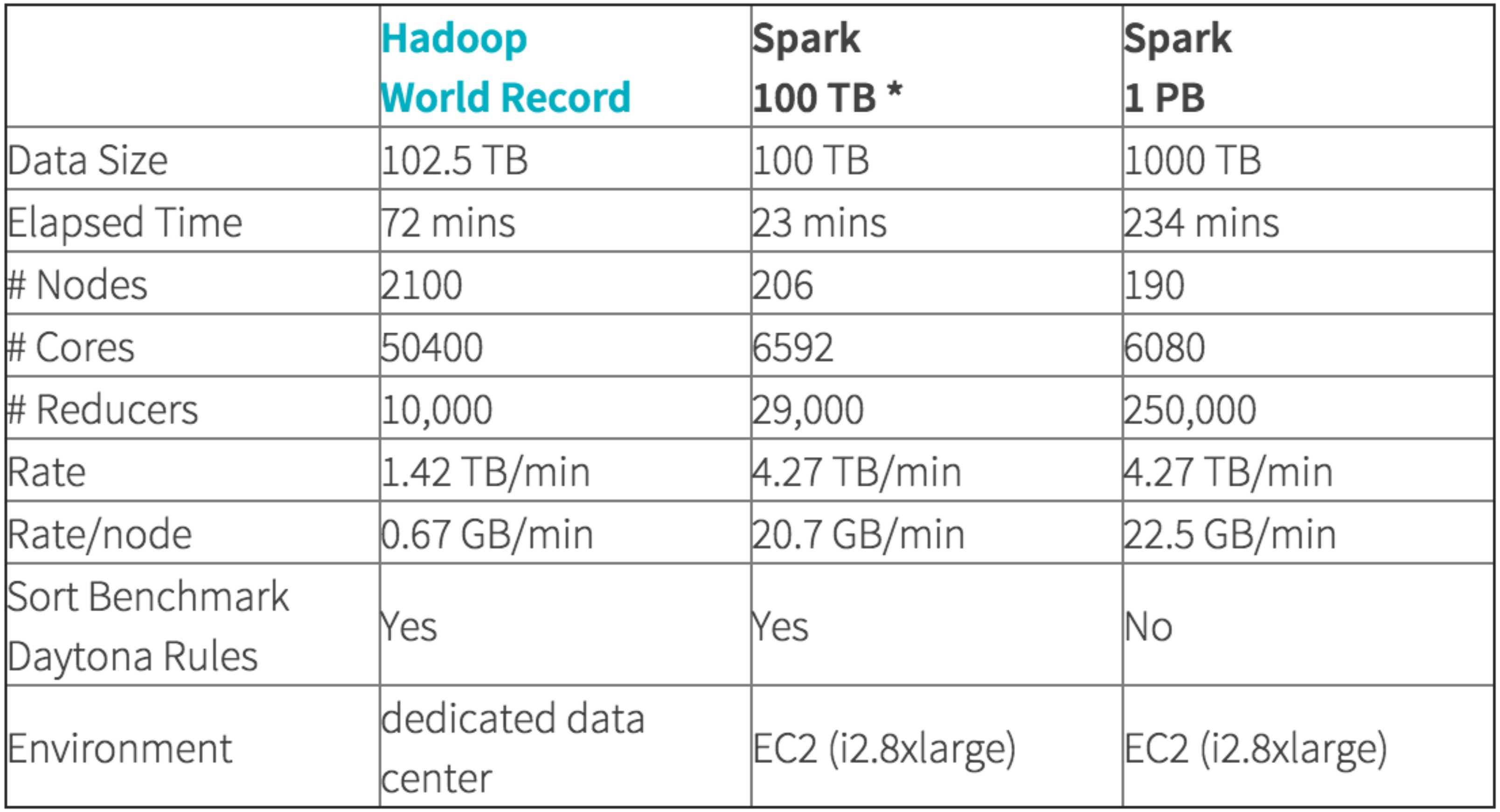
Sorting

Distribute Timsort, which is already well-adapted to respecting disk vs memory

Sample points to find good boundaries

Each machines sorts locally and builds an index

Sorting (shuffle)



	Hadoop World Record	Spark 100 TB *	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400	6592	6080
# Reducers	10,000	29,000	250,000
Rate	1.42 TB/min	4.27 TB/min	4.27 TB/min
Rate/node	0.67 GB/min	20.7 GB/min	22.5 GB/min
Sort Benchmark Daytona Rules	Yes	Yes	No
Environment	dedicated data center	EC2 (i2.8xlarge)	EC2 (i2.8xlarge)

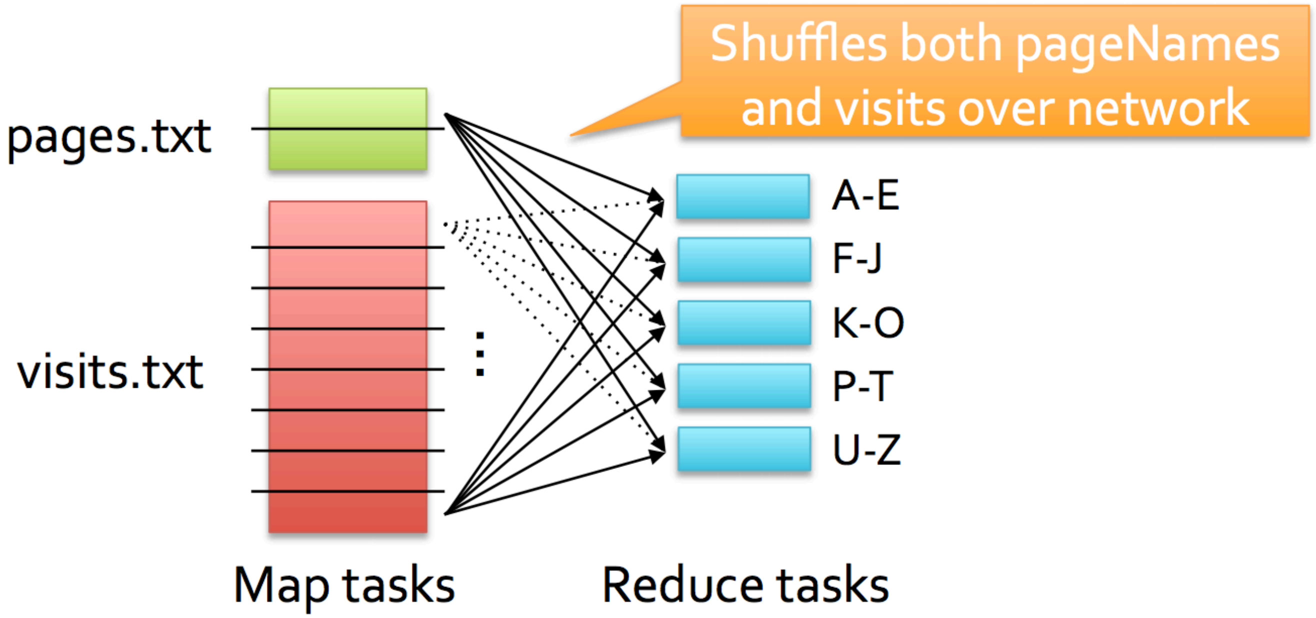
Distributed TimSort

Example Join

```
// Load RDD of (URL, name) pairs
val pageNames = sc.textFile("pages.txt").map(...)

// Load RDD of (URL, visit) pairs
val visits = sc.textFile("visits.txt").map(...)

val joined = visits.join(pageNames)
```



Broadcasting

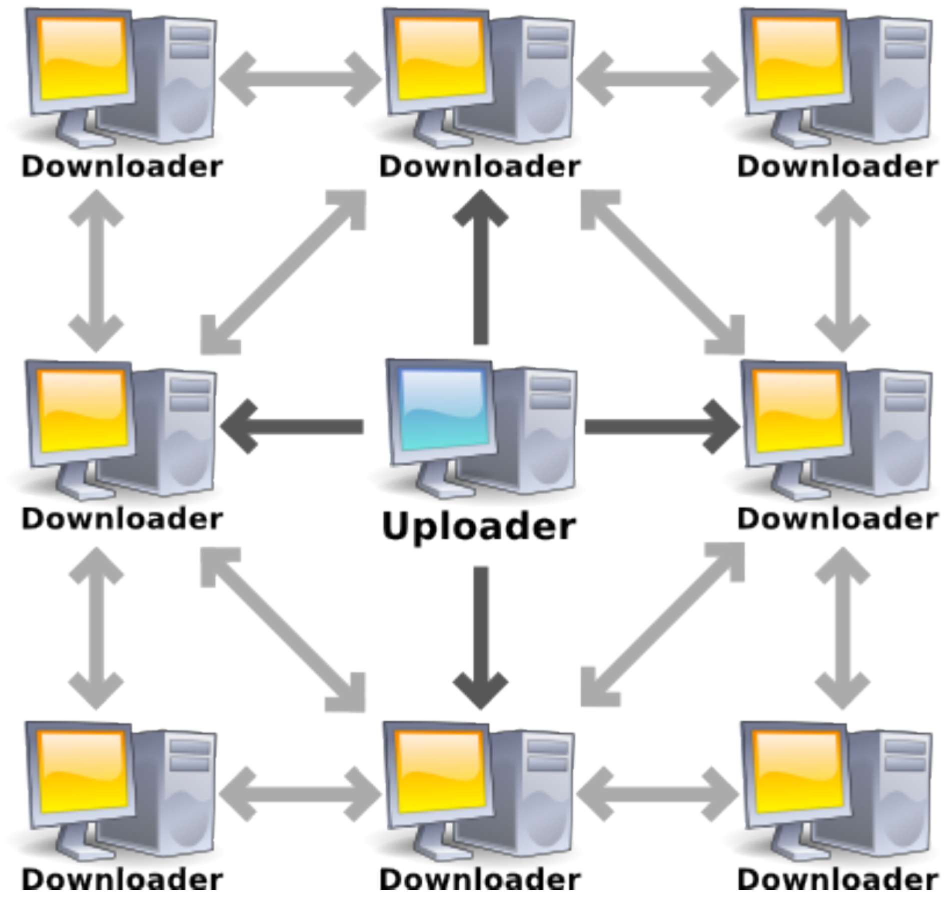
Broadcasting

Often needed to propagate current guess for optimization variables to all machines

The exact wrong way to do it is with “one machines feeds all” – use bit-torrent instead

Needs log(n) rounds of communication

Bit-torrent Broadcast



Broadcast Rules

Create with SparkContext.broadcast(initialVal)

Access with .value inside tasks (first task on each node to use it fetches the value)

Cannot be modified after creation

Replicated Join

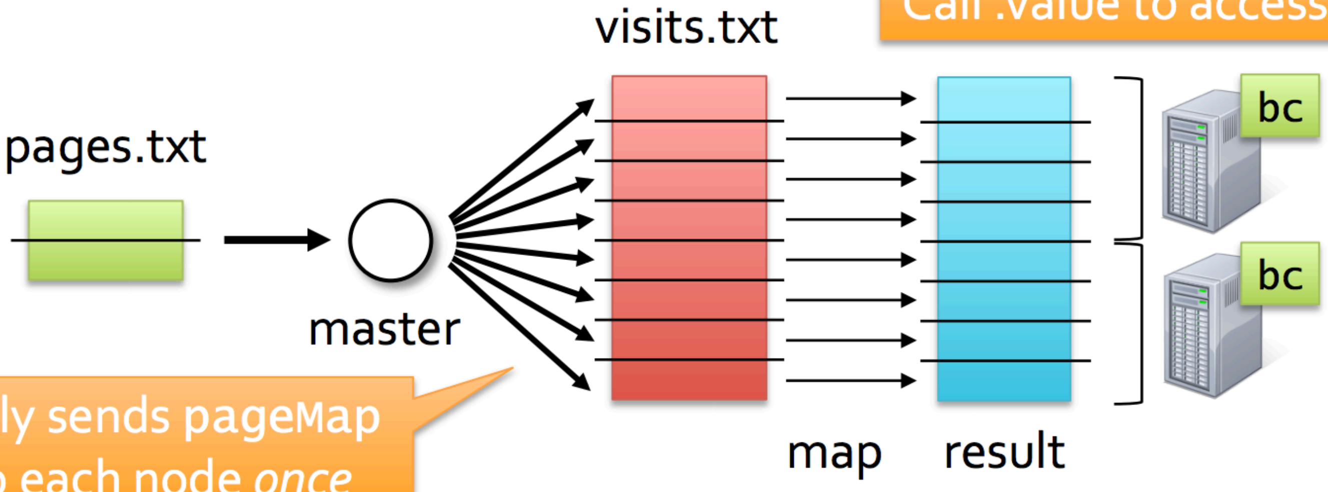
```
val pageNames = sc.textFile("pages.txt").map(...)
val pageMap = pageNames.collect().toMap()
val bc = sc.broadcast(pageMap)

val visits = sc.textFile("visits.txt").map(...)

val joined = visits.map(v => (v._1, (bc.value(v._1), v._2)))
```

Type is Broadcast[Map[...]]

Call .value to access value



Only sends pageMap to each node *once*

Model Broadcast

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.zeros(d)
for (i <- 1 to numIterations) {
  val gradient = points.map { p =>
    (1 / (1 + exp(-p.y * w.dot(p.x)) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= alpha * gradient
}
```

Model Broadcast

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

Call sc.broadcast

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.zeros(d)
for (i <- 1 to numIterations) {
  val gradient = points.map { p =>
    (1 / (1 + exp(-p.y * w.dot(p.x)) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= alpha * gradient
}
```

Use via .value

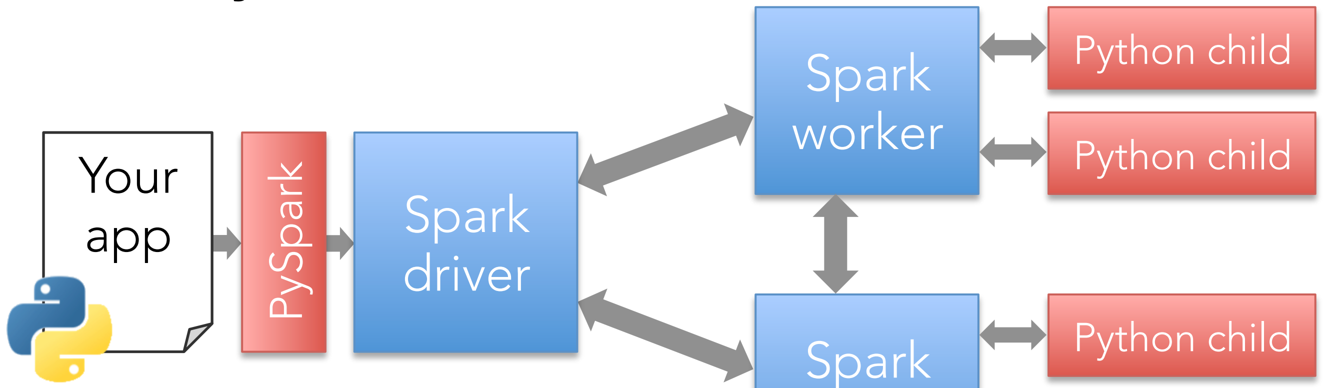
Spark for Python (PySpark)

PySpark and Pipes

Spark core is written in Scala

PySpark calls existing scheduler, cache and networking layer (2K-line wrapper)

No changes to Python



Your app
PySpark
Spark driver
Spark worker
Spark worker
Python child
Python child
Python child
Python child