BIG DATA

Spark : Architecture

K V Subramaniam
Computer Science and Engineering

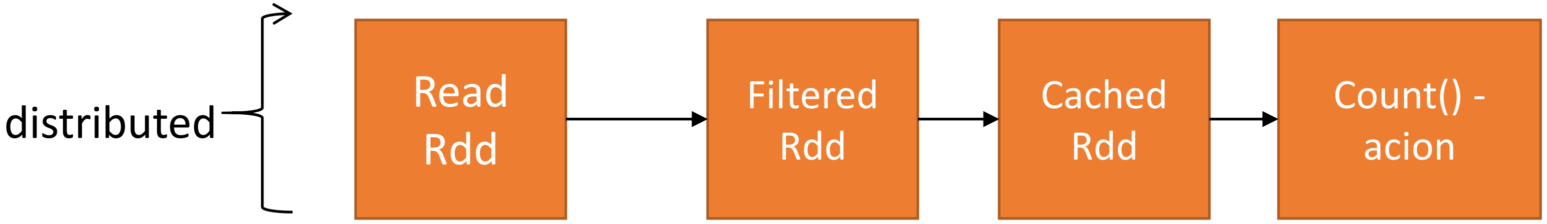
Overview of lecture

- Spark High Level Architecture
- Lifetime of a Spark Job
- Lazy Evaluation
- RDDs
- Spark Scheduling
- Dataframes

Spark High Level Architecture

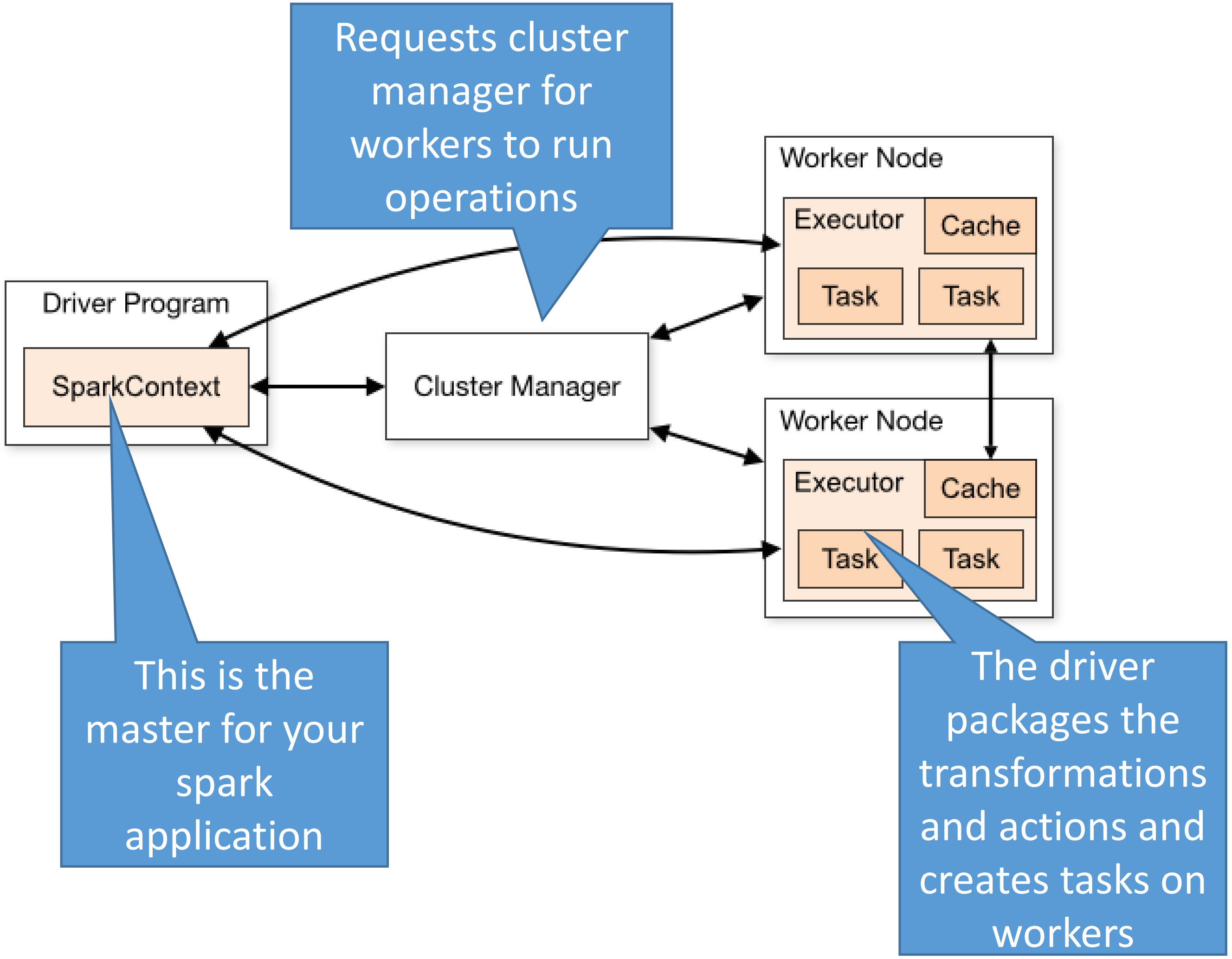
Recall: Log Mining Example

```
val sc = new SparkContext("spark://...", "MyJob", home,
    jars)

val file = sc.textFile("hdfs://...") // This is an RDD

val errors = file.filter(_.contains("ERROR")) // This is
    an RDD

errors.cache()

errors.count() // This is an action
```



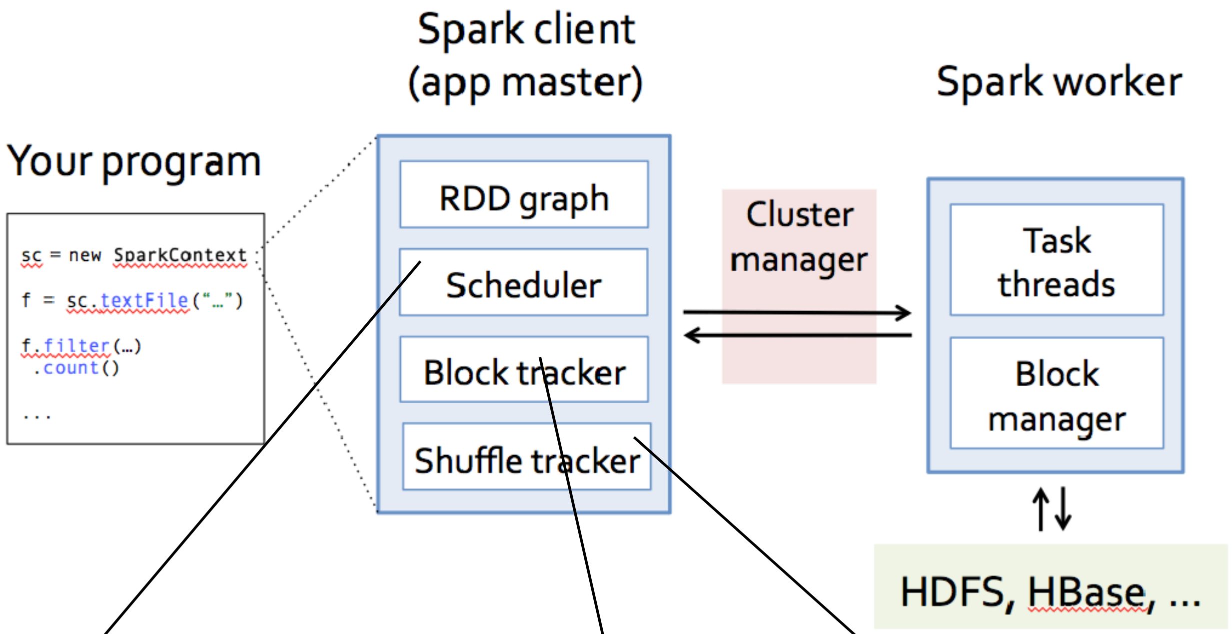
How is this executed by Spark?

Spark Architecture



https://spark.apache.org/docs/latest/cluster-overview.html

Spark Architecture

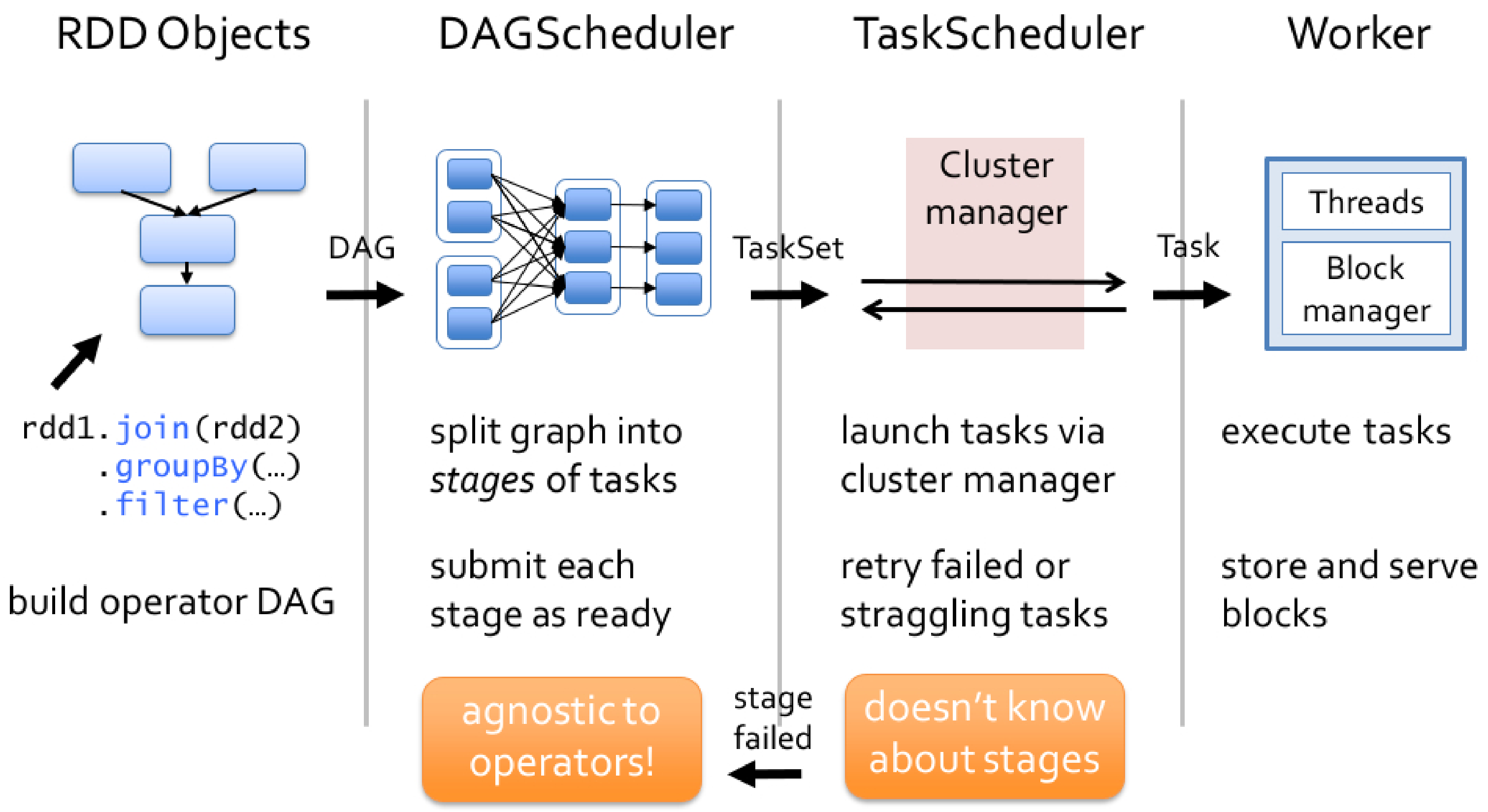


Where to run the tasks

Input Data

Do we have all the data to move forward

Spark Working details



https://hxquangnhat.com/2015/04/03/arch-spark-job-submission-breakdown/

Lazy Execution in Spark

Lazy Execution

- In Hadoop, when we submit a job the master starts executing it
- In Spark, when does the master start executing the job?
  - Spark uses a technique called Lazy execution

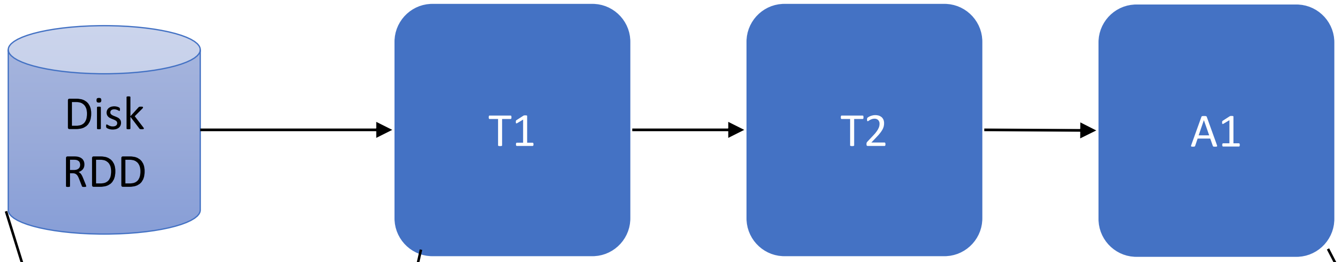
Lazy Execution

- Remember that we defined Spark operations into *transformations* and *actions*

- The spark driver does not execute anything till it encounters an *action*

- *Transformations* are only noted for purpose of lineage.

Lazy Execution



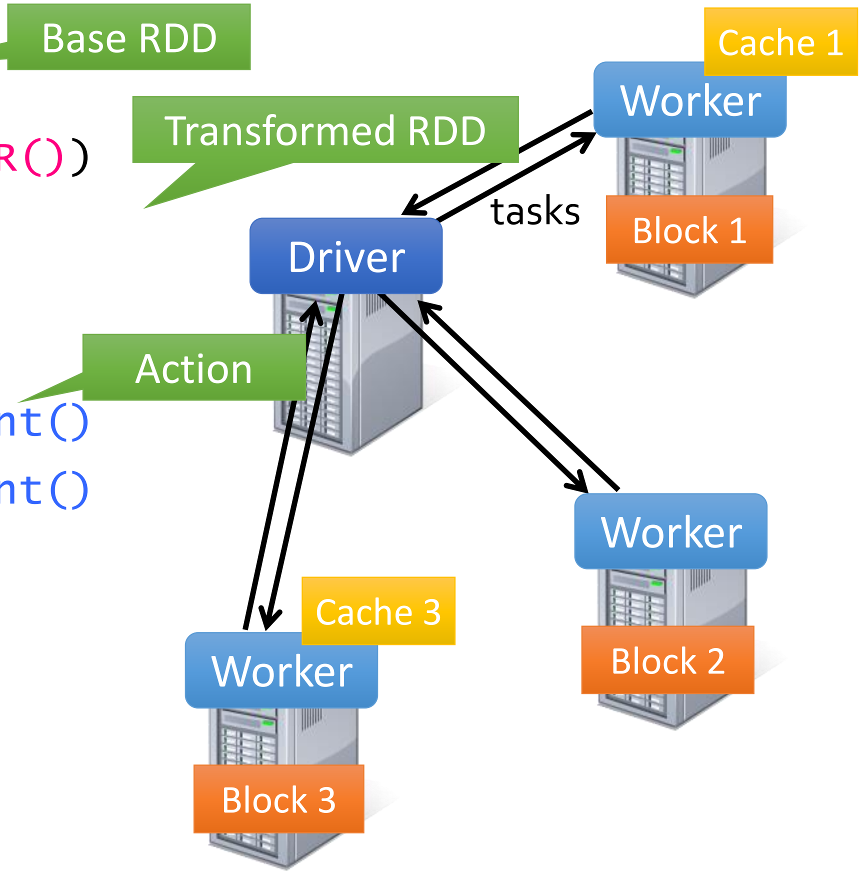
lines = spark.textFile(“hdfs://...”)
// Doesn’t read file. Notes that RDD is to be created

Transformation say *map*
Doesn’t do transformation immediately

Action say *count.* Triggers execution: all transformations done block by block on same server followed by count

Spark Working with Log Mining Example


```
lines = spark.textFile(“hdfs://...”)
errors = lines.filter(startsWithERROR())
messages = errors.map(split(“\t”),2)
cachedMsgs = messages.cache()

cachedMsgs.filter(containsfoo()).count()
cachedMsgs.filter(containsbar()).count()
```



RDDs - details

What is an RDD

- RDD is partitioned, locality aware, distributed collections
  - RDDs are immutable. (Why it's necessary?)

- RDDs are data structures that either
  - Point to the source (HDFS)
  - Apply some transformations to the parent RDDs to generate new data elements

- Computations on RDDs
  - Lazily evaluated lineage DAGs composed of chained RDDs

Why the RDD abstraction?

- Support operations other than map and reduce
- Support in memory computation
- Arbitrary composition of such operators
- Simplify scheduling

How to capture dependencies generically?

Representing RDDs

- Set of partitions (“splits”)
  - Much like Hadoop. Each RDD associated with a input partitions
- List of dependencies on parent RDDs
  - Not there in Hadoop. This is new
- Function to compute a partition given parents
  - User defined code. (similar to map()/reduce() in Hadoop
- Optional preferred locations
  - For data locality
- Optional partitioning information (partitioner)
  - Advanced – for shuffle (see later)

RDDs Interface

Operation	Meanning
partitions()	Return a list of Partition objects
preferredLocations(*p*)	List nodes where partition *p* can be accessed faster due to data locality
dependencies()	Return a list of dependencies
iterator(*p, parentIters*)	Compute the elements of partition *p* given iterators for its parent partitions
partitioner()	Return metadata specifying whether the RDD is hash/range partitioned

Examples of RDDs

- Hadoop RDD
  - Partitions – one per block
  - Dependencies – none
  - Compute (partition) – read corresponding block
  - Preferred locations – HDFS block location
  - Partitioner - none

- Filtered RDD (as in sample application)
  - Partitions – same as parent
  - Dependencies – 1-1 with parent
  - Compute – compute parent and filter it.
  - Preferred locations – ask parent (none)
  - Partitioner - none

Exercise

Based on the sample of the Filter RDD, can you work out what will be the partitions, compute, dependencies, preferred locations and partitioner for a joinRDD

- Filtered RDD (as in sample application)
    - Partitions – same as parent
    - Dependencies – 1-1 with parent
    - Compute – compute parent and filter it.
    - Preferred locations – ask parent (none)
    - Partitioner - none
-

Examples of RDDs

- Joined RDD
  - RDDPartitions – one per reduce task
  - Dependencies – shuffle on each parent
  - Compute (partition) – read and join shuffled data
  - Preferred locations – none
  - Partitioner – HashPartioner (num tasks)

Spark Scheduling

Page Rank example in Spark

- lines = textfile ("urls.txt"))
- links = lines.map (lambda urls: urls.split()).groupByKey().cache()
- ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))
- for iteration in range(MAXITER)):
- contribs = links.join(ranks).flatMap(lambda url_neighbors_rank: computeContribs (url_neighbors_rank)
- ranks = contribs.reduceByKey(add).mapValues (lambda rank: rank * 0.85 + 0.15)

```
def computeContribs (url_neighbors_rank):
"""Calculates URL contributions to the rank of other URLs.
"""
num_neighbors = len (url_neighbors_rank) – 2

rank = url_neighbors_rank [len (url_neighbors_rank) – 1]

for i in range (1, num_neighbors):
        yield (url_neighbors_rank[i], rank / num_neighbors)
```

1. Start each page with a rank of 1
2. On each iteration, update each page's rank to

$$\Sigma_{i \in neighbors} rank_i / |neighbors_i|$$

DAG representation

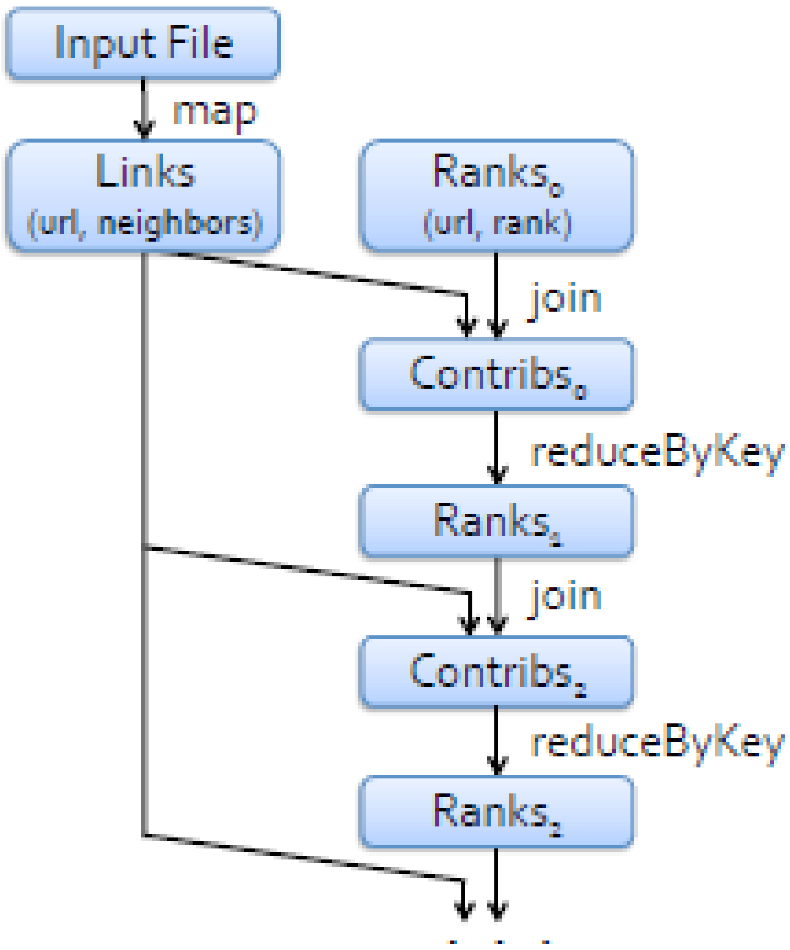
- The Spark Driver will first convert this program into a DAG representation

- What does the DAG representation contain?

  - Each RDD is a node in the graph and

  - all transformations/actions on the RDD as edges

DAG representation of Page Rank



lines = textfile (“urls.txt”))

links = lines.map (lambda urls: urls.split()).groupByKey().cache()

ranks = links.map(lambda url_neighbors: (url_neighbors[0], 1.0))

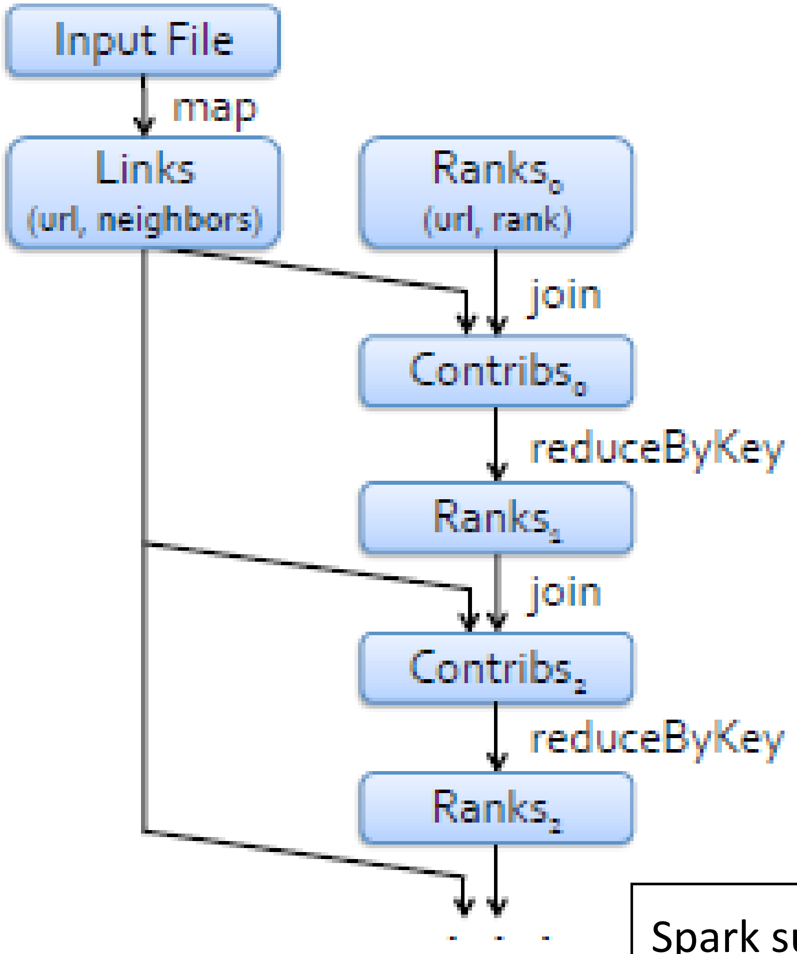
for iteration in range(MAXITER)):

contribs = links.join(ranks).flatMap(lambda url_neighbors_rank: computeContribs (url_neighbors_rank)

ranks = contribs.reduceByKey(add).mapValues(lambda rank: rank * 0.85 + 0.15)

Ranks and Links are spread across multiple nodes. How does Spark ensure join works properly? Hint: Think about how join works.

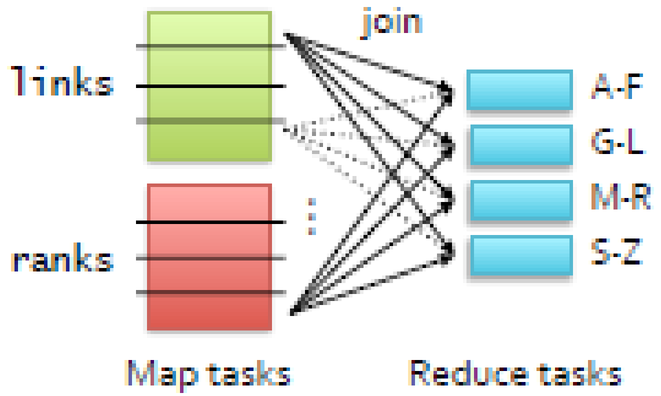
DAG representation of Page Rank



`links` and `ranks` are repeatedly joined

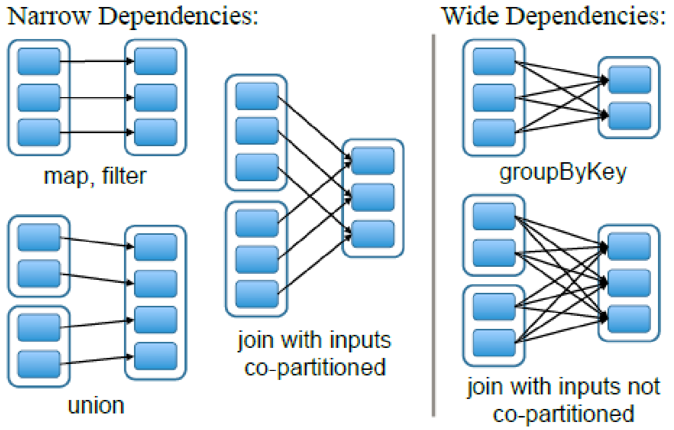
Each join requires a full shuffle over the network

» Hash both onto same nodes



Spark supports two types of partitioning – hash and range

- narrow dependencies
  - where each partition of the parent RDD is used by at most one partition of the child RDD
  - Does not need a shuffle; pipeline operations
  - Shuffle: movement of data from one node to another

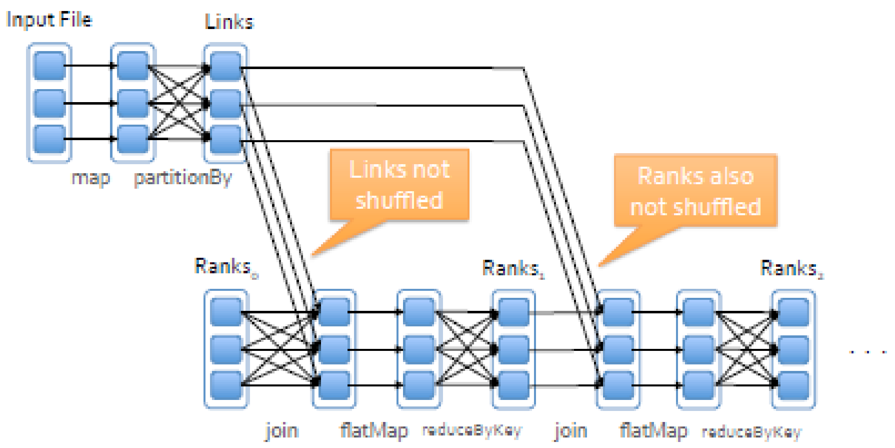


- wide dependencies
  - where multiple child partitions may depend on it.
  - May need a shuffle

- Copartition → technique to make sure that both inputs to a join are partitioned using same function

Lineage and Optimizing Placement

- links & ranks repeatedly joined
- Can copartition them (e.g.hash both on URL) to avoid shuffles
- Spark supports two types of partitioning: hash and range




```
lines = textfile ("urls.txt"))

links = lines.map (lambda urls:
urls.split()).groupByKey().cache()

ranks = links.map(lambda url_neighbors:
(url_neighbors[0], 1.0))

 for iteration in range(MAXITER)):

        contribs = links.join(ranks).flatMap(
lambda url_neighbors_rank:
computeContribs
(url_neighbors_rank)

        ranks =
contribs.reduceByKey(add).mapValues(lam
bda rank: rank * 0.85 + 0.15)
```

Narrow and Wide Dependencies

Narrow Dependencies

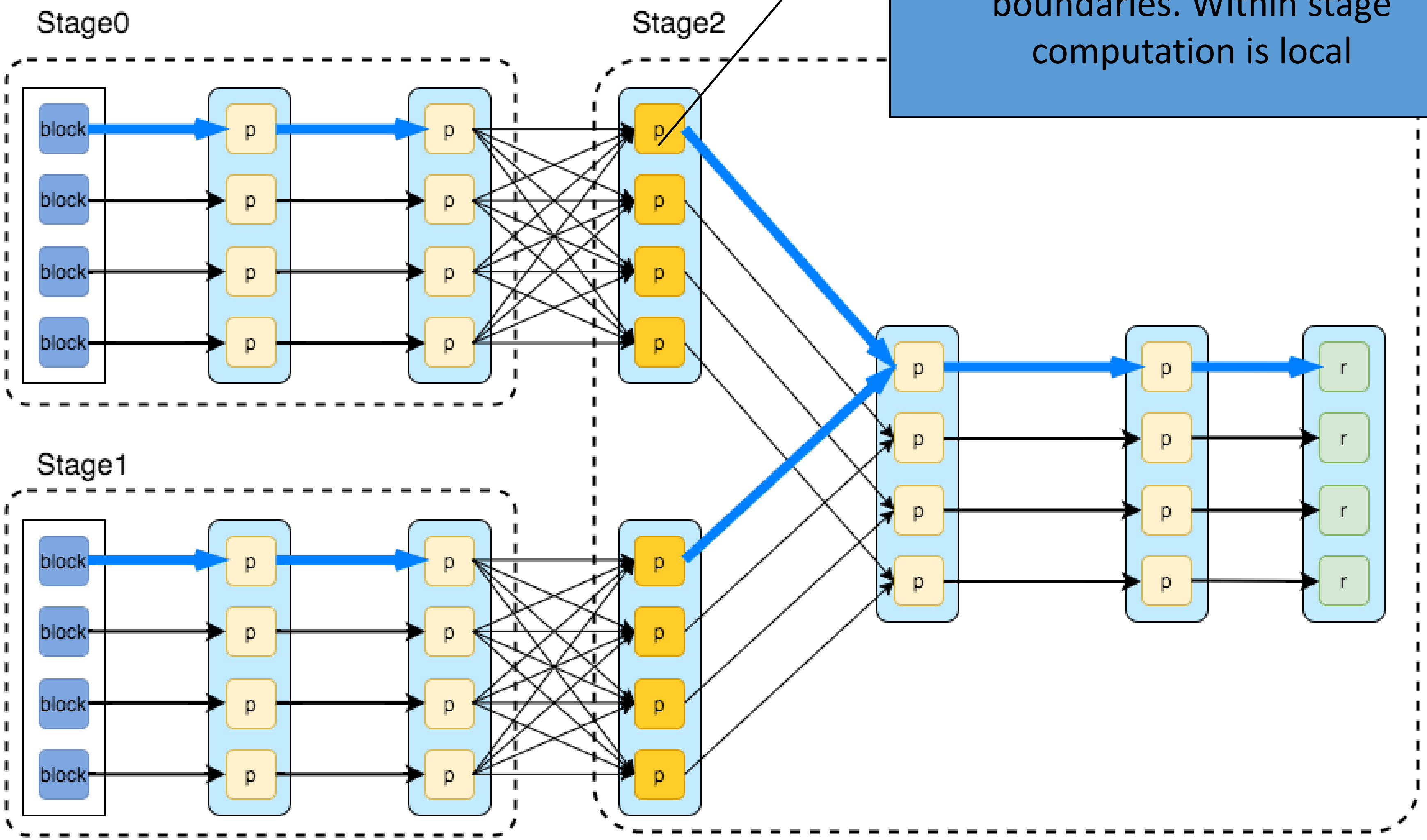
- Map
- FlatMap
- MapPartitions
- Filter
- Sample
- Union

W I d e Dependencies

- Intersection
- Distinct
- ReduceByKey
- GroupByKey
- Join
- Cartesian
- Repartition
- Coalesce

BIG DATA

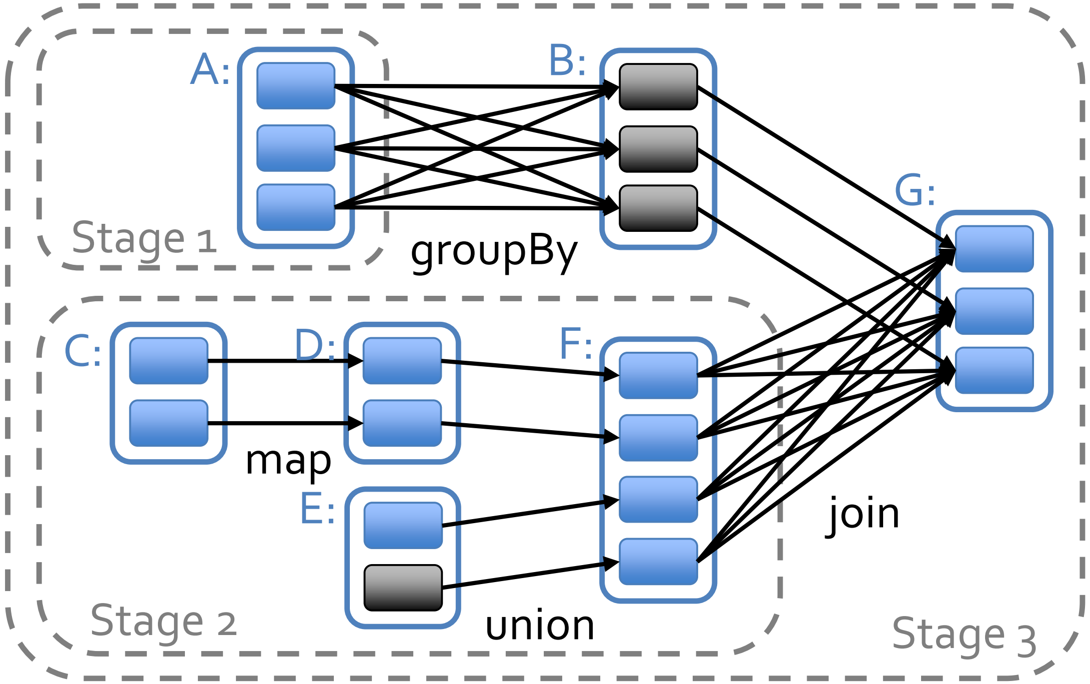
Scheduling



Breakup DAG into stages at shuffle boundaries. Within stage computation is local

Task Assignment

- scheduler assigns tasks to machines based on data locality using delay scheduling
    - if a task needs to process a partition that is available in
    - memory on a node, then send it to that node
    - otherwise, a task processes a partition for which the containing RDD provides preferred locations (e.g., an HDFS file), then send it to those



Simplifying tasks for programmers - DataFrames

Need for dataframes

- Data RDDs are completely opaque to Spark
  - Meaning Spark cannot parse these values
- Is there some way to make Spark understand the format, so that we can do processing more easily
  - Like sql type queries
- Consider data like on the right that we need to run a query on?

USN	Name	Marks
45	Vkoli	11
10	Stendul	43
195	Abachpan	28

What is a dataframe

- Introduced in 2015
- Inspired by Dataframes in R and Pandas in python
- Distributed collection of data into named columns
- An abstraction built over RDD that allows
  - Schema to be defined on a RDD
- Also has an optimizer built in for queries

USN	Name	Marks
45	Vkoli	11
10	Stendul	43
195	Abachpan	28

Creating dataframes

Dataframes can be created from existing RDDs, HIVE tables other Data sources.

This example is creating from a JSON file

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)

df = sqlContext.jsonFile("pes/students.json")

# let us display the contents
df.show()
## USN  name    marks
## 045  Vkoli    11
## 010  Stendul   43
## 195  Abachpan   28

Df = rdd.toDF("age", "name")
```

Alternatively, from an existing RDD by naming the columns

Using a dataframe

```
# Print the schema in a tree format
df.printSchema()
## root
## |-- usn: long (nullable = true)
## |-- name: string (nullable = true)
## |-- marks: long (nullable = true)

# Select only the "name" column
df.select("name").show()
## name
## VKoli
## STendul
## ABachpan

# Select everybody, but increment the age by 1
df.select("name", df.marks + 1).show()
## name    (marks + 1)
## Vkoli       12
## STendul     44
## ABachpan    29
```

Exercise

- Consider a case where you have data in a CSV file that consists of <pan number, date, tax_paid> and you wanted to find out the total tax paid by each individual pan holder
  - How will you do it in Spark?
  - How will you do it with Spark Data frames

- Consider a case where you have data in a CSV file that consists of <pan number, date, tax_paid> and you wanted to find out the total tax paid by each individual pan holder
  - How will you do it in Spark?
  - How will you do it with Spark Data frames

Using Dataframes

```
Df = rdd.toDF("pan number", "date", "taxpaid")
Df.select("pan number", "tax paid").groupBy("pan number").sum()
```

Note that this is done using the name of the column rather than by splitting the data which we would do if used Spark.

DryadLINQ, FlumeJava

- Similar “distributed collection” API, but cannot reuse datasets efficiently *across* queries

- Relational databases
  - Lineage/provenance, logical logging, materialized views

GraphLab, Piccolo, BigTable, RAMCloud

- Fine-grained writes similar to distributed shared memory

- Iterative MapReduce (e.g. Twister, HaLoop)
  - Implicit data sharing for a fixed computation pattern
- Caching systems (e.g. Nectar)
  - Store data in files, no explicit control over what is cached

THANK YOU

K V Subramaniam, Usha Devi
Dept. of Computer Science and Engineering

subramaniamkv@pes.edu
ushadevibg@pes.edu