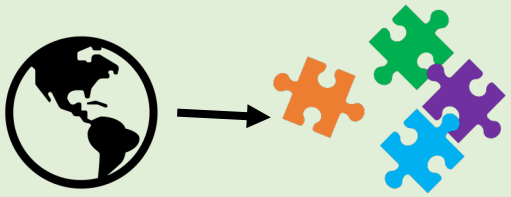
HPC



1)Algorithms

2)Architecture

Solutions

3)Shared Memory
OpenMP

4)Distributed Memory
MPI

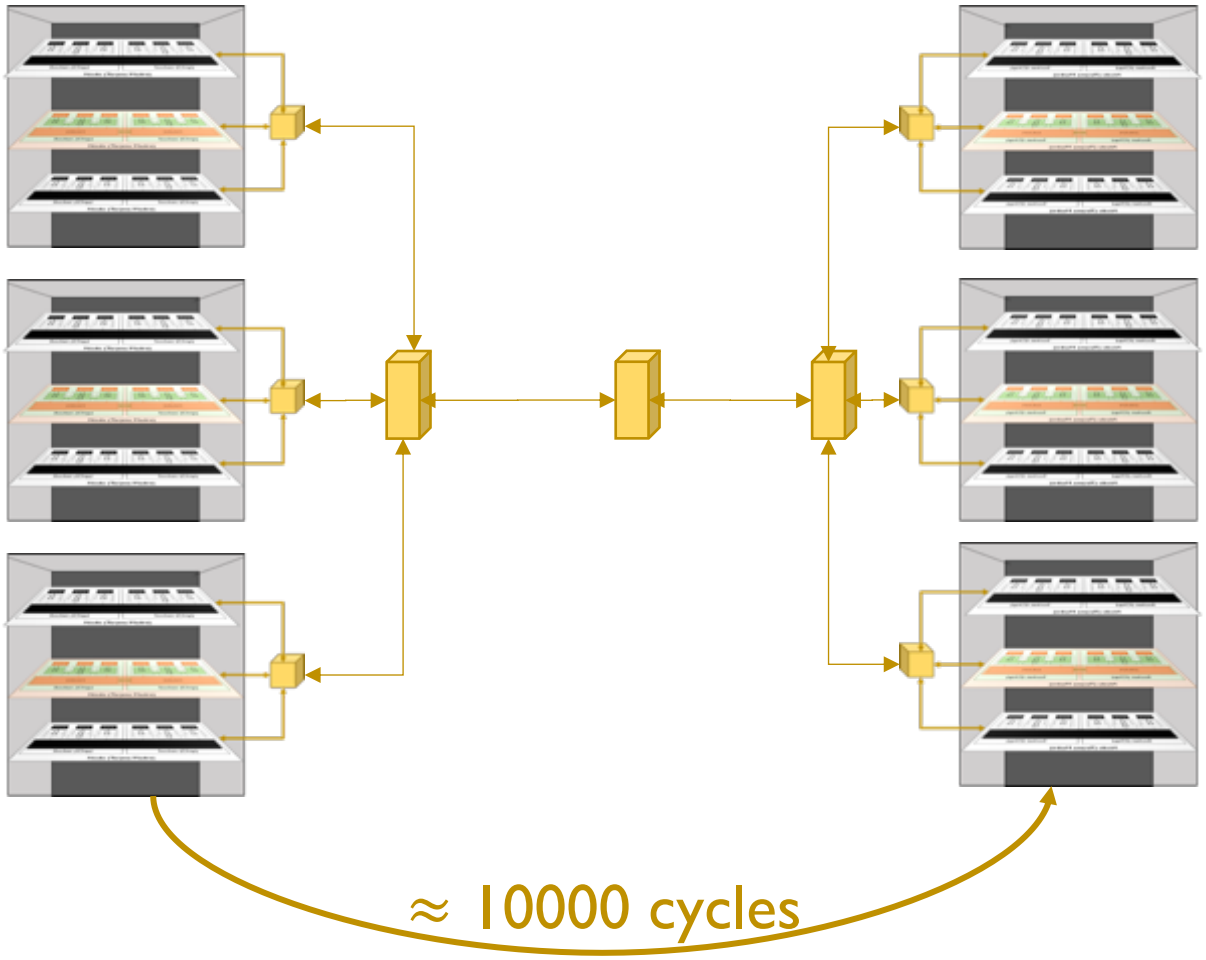
5) Unified Engine
Spark

What was challenging about the previous approaches?

3)Unified Engine

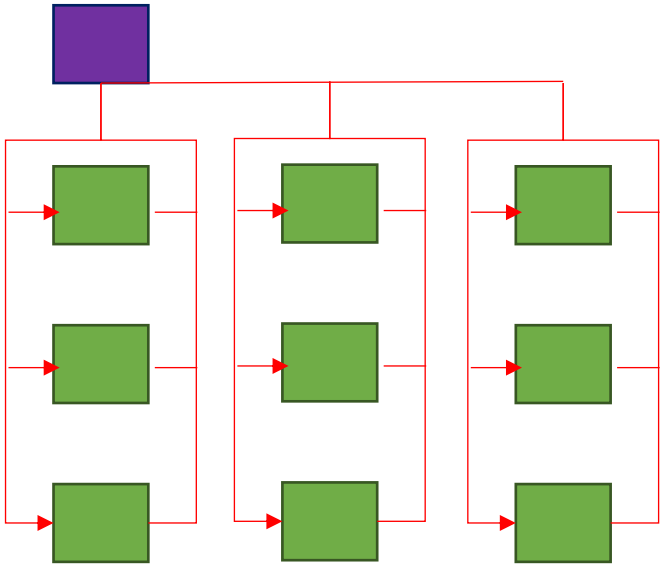
Spark + Hadoop

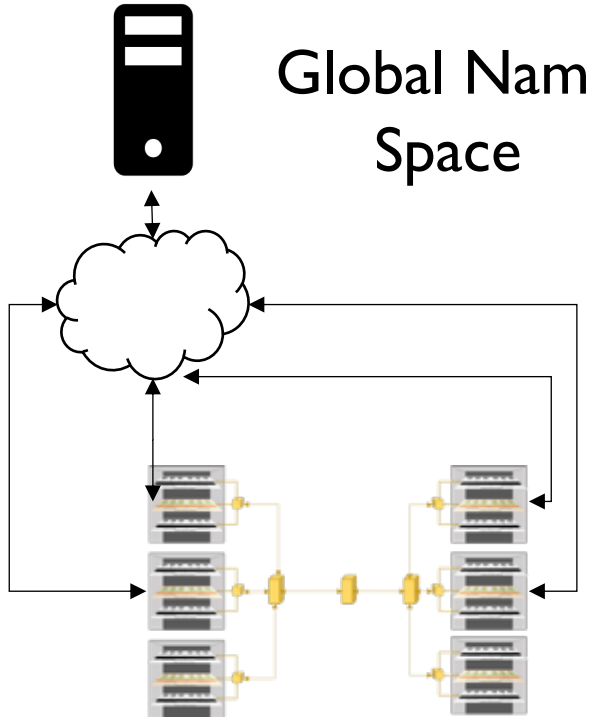
Distributed Memory



Spark

Fine granularity





Global Name Space

Parallel Regions in a distributed memory ecosystem

What do we need in a unified engine?

Distribute instructions

Unified Data Management

Instructions
MapReduce / Spark

Fine Level of granularity

1. Transformations & Actions
2. Resilient Distributed Datasets

Data Management
HDFS

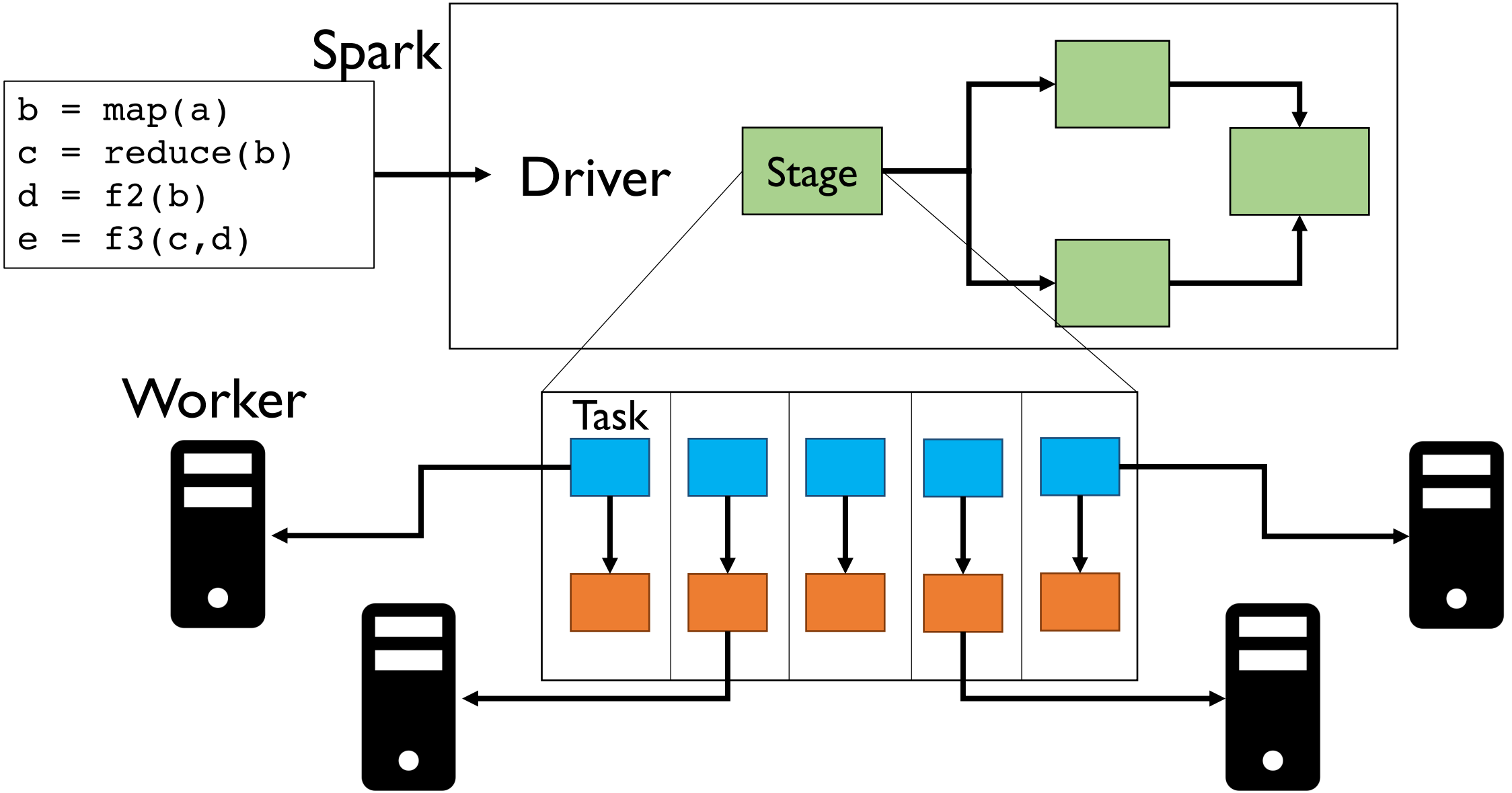
Global naming Space

Fault Tolerance & Portability

Cluster Manager: YARN

Resource Allocation & Job Scheduling

Spark in a nutshell

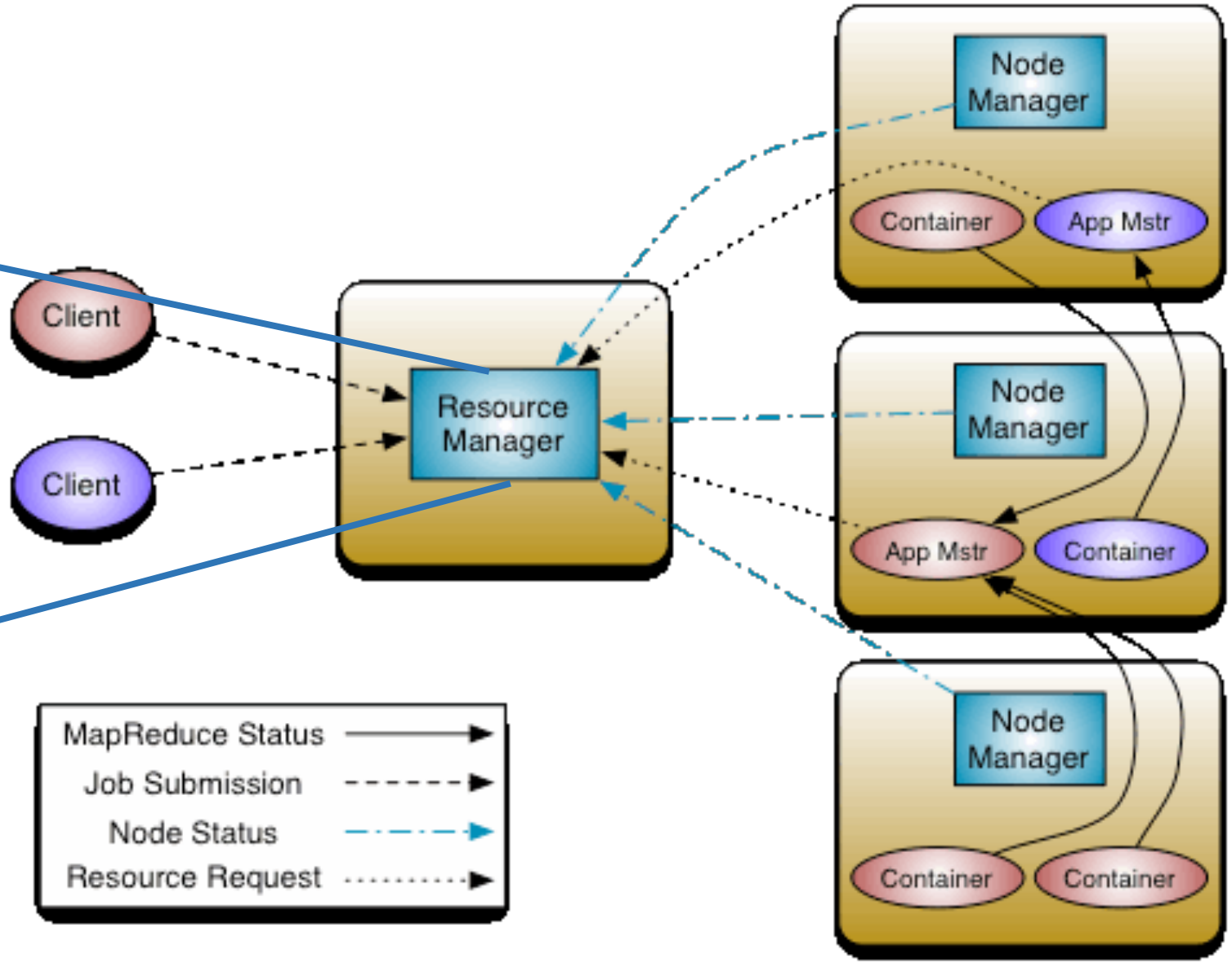


YARN: Provides the executors

YARN

Scheduler
Allocates Resorces

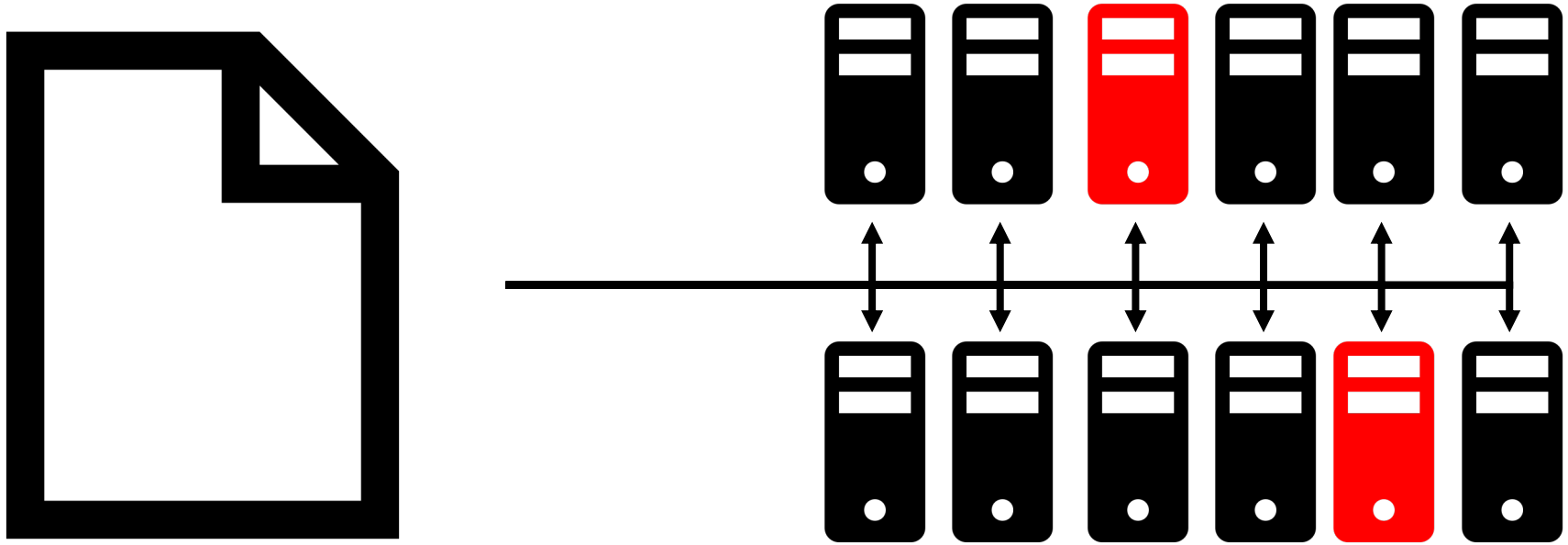
ApplicationsManager
Negotiates job-submissions
Recovery & Failure



https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html

What is Hadoop?

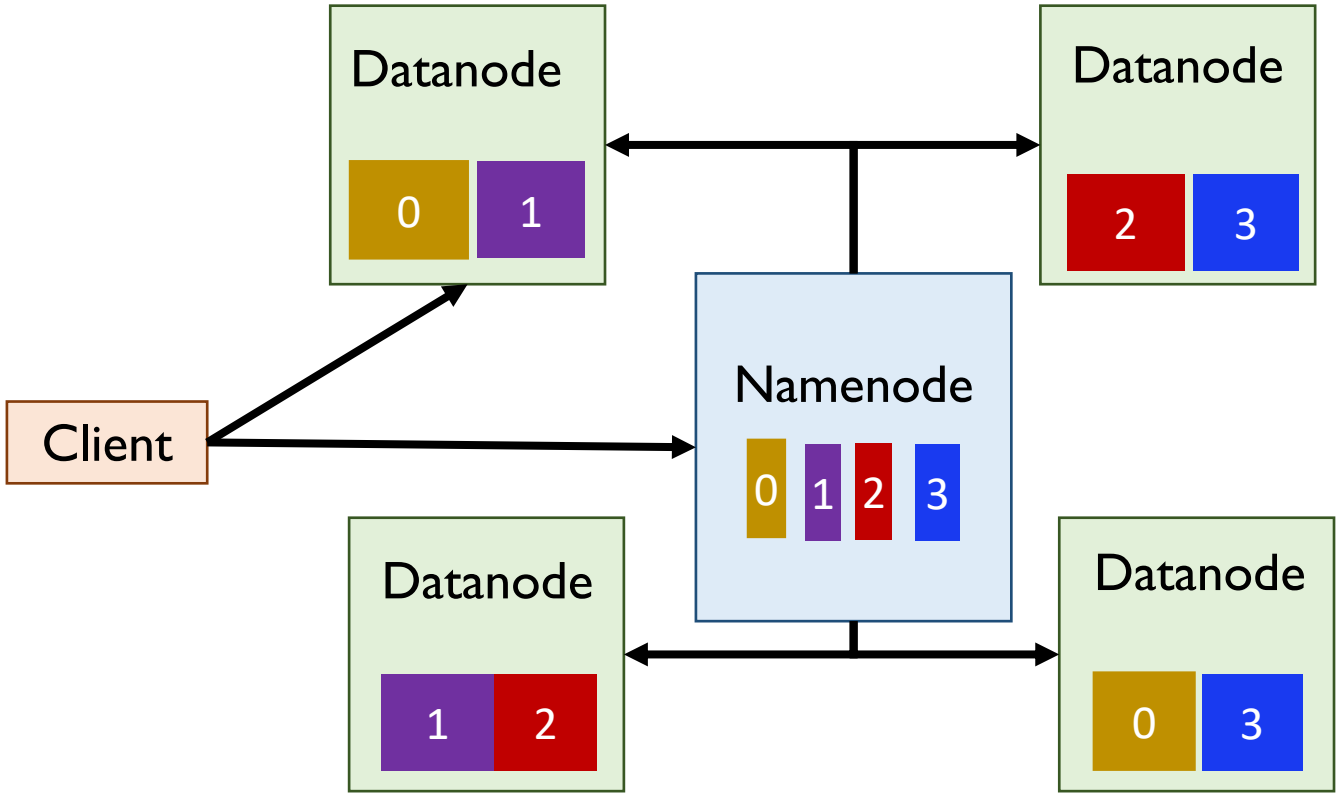
Library in Java to manage large data sets in clusters with failures



https://hadoop.apache.org

https://spark.apache.org

HDFS: Hadoop File System

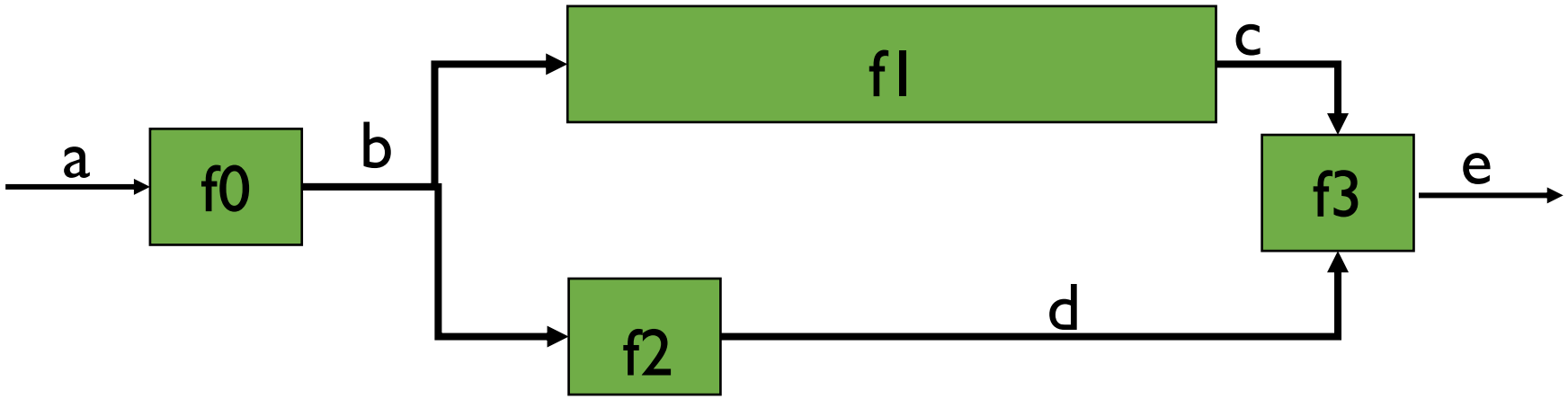


Fault Tolerance

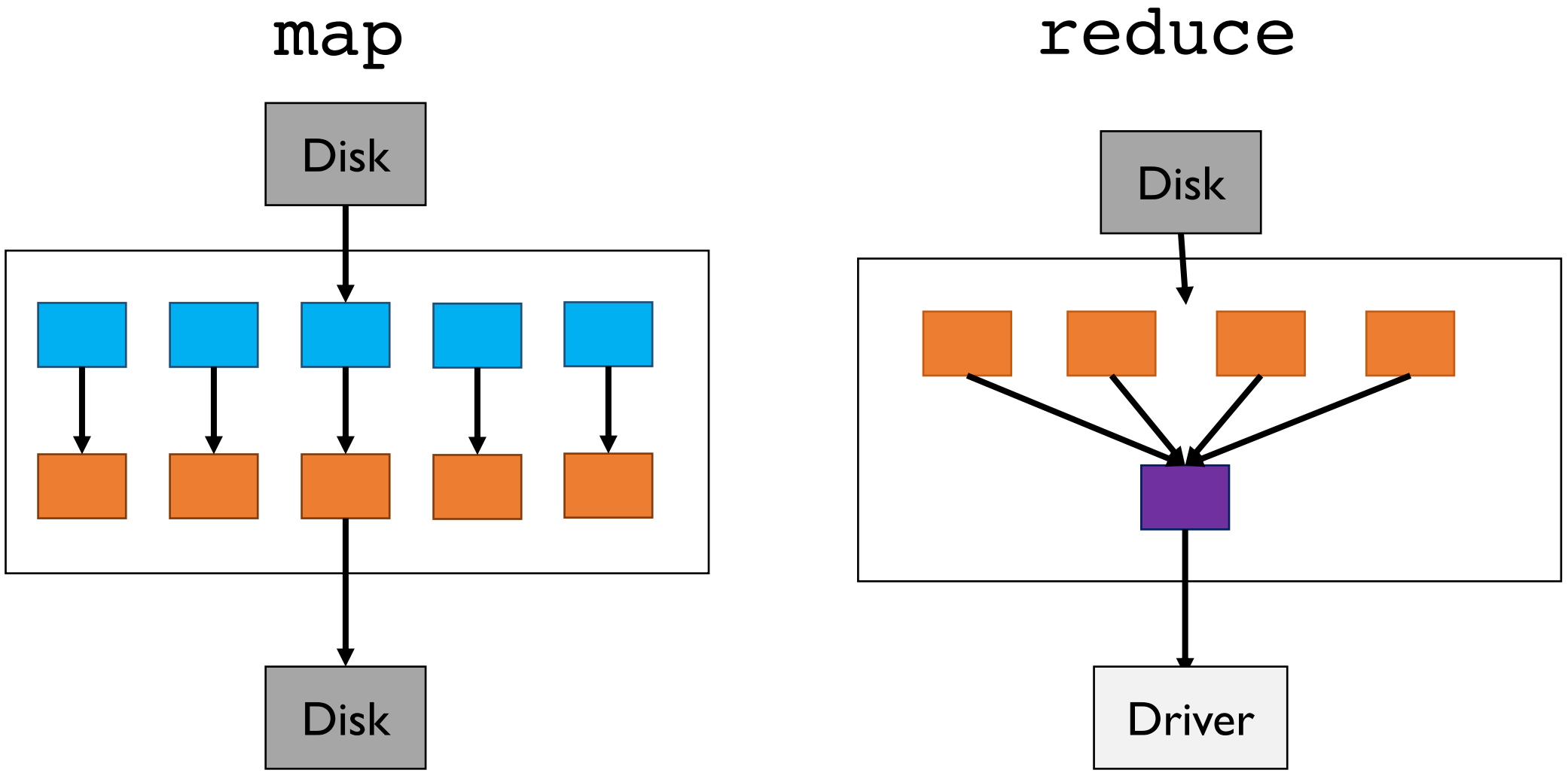
Moving Computation less expensive than moving data

Portability

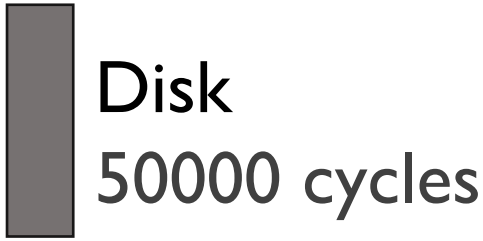
Why Spark is successful: Functional Programming

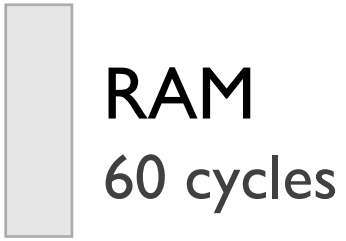


Map Reduce Paradigm



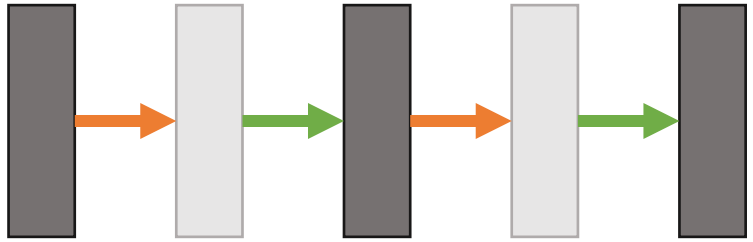
Spark: Resilient Distributed Datasets (RDDs)



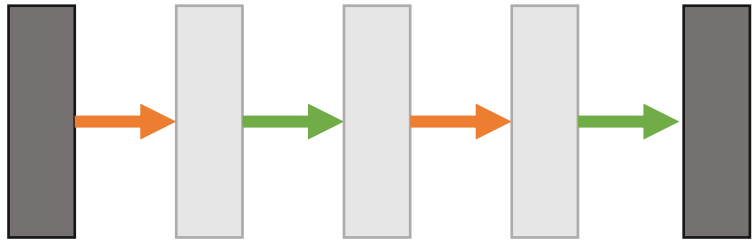



```
for i in 1:n
   b = map(a)
   a = reduce(b)
```

Map Reduce



Spark



How to create RDDS

parallelize(data)

Distribute data in driver program

textFile("data.txt")…

Distribute dataset in external storage: HDFS, shared filesystem …

Accessible to all workers

How to **persist** in memory

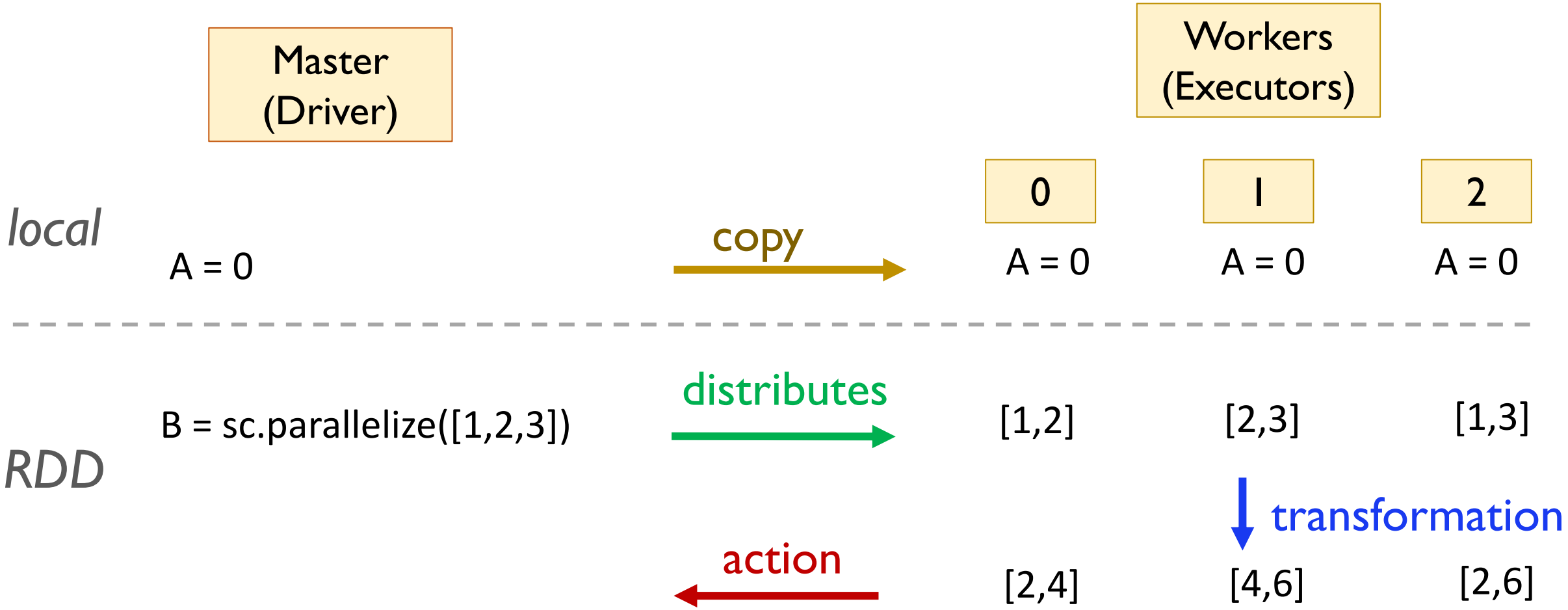
Default (*Lazy Operation*)

Compute transformation on RDDs each time an action is required

persist(Storage_level)…

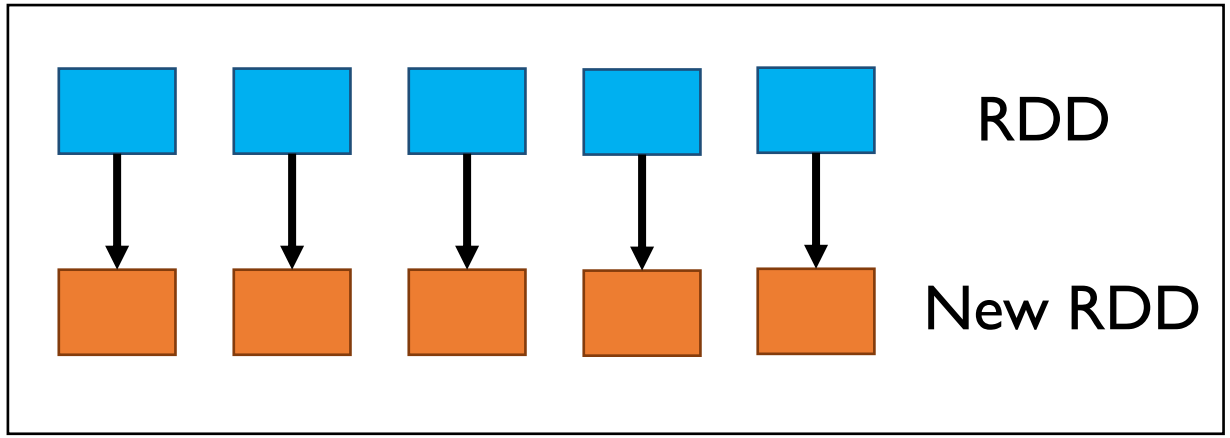
Keeps in memory RDD without recomputing each time

Local vs Global Variables



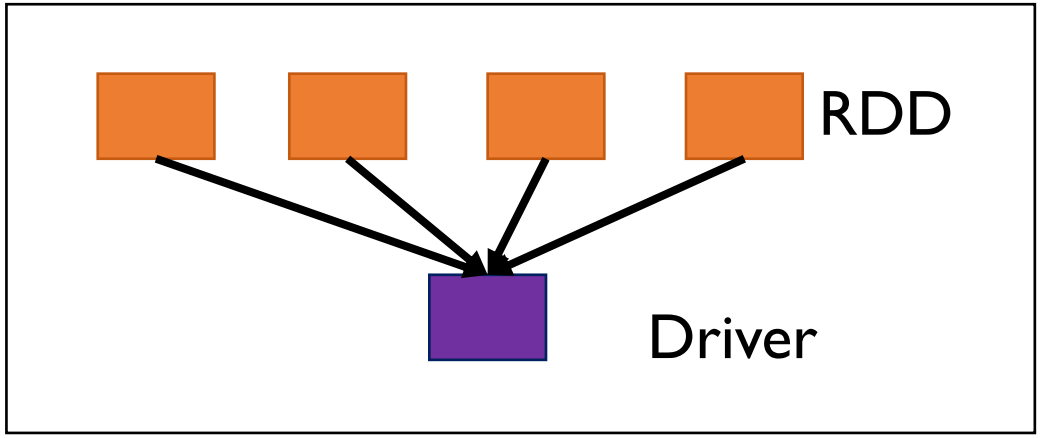
Spark Enhancing

Transformation




```
map(func)
filter(func)
flatMap(func)
distinct()
sample(k)
foreach(func)
```

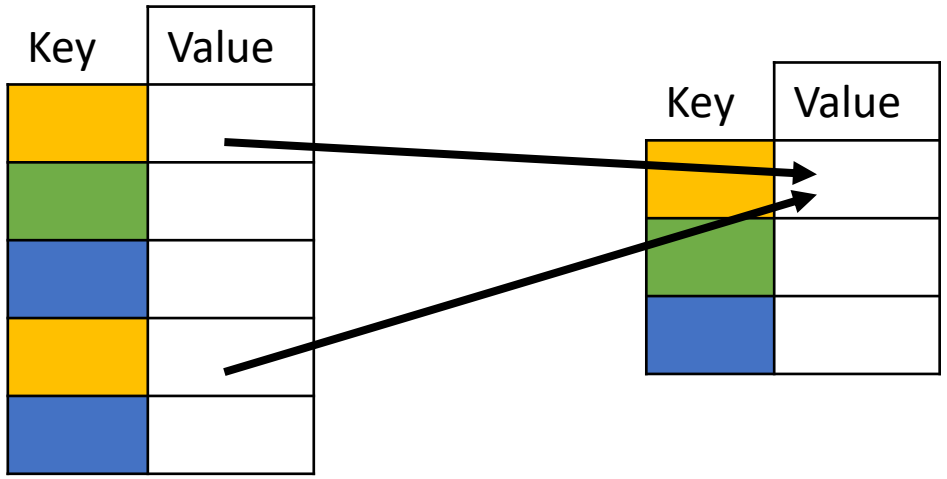
Action




```
collect()
take(k)
count()
reduce()
takeSample(k)
saveAsTextFile(…)
```

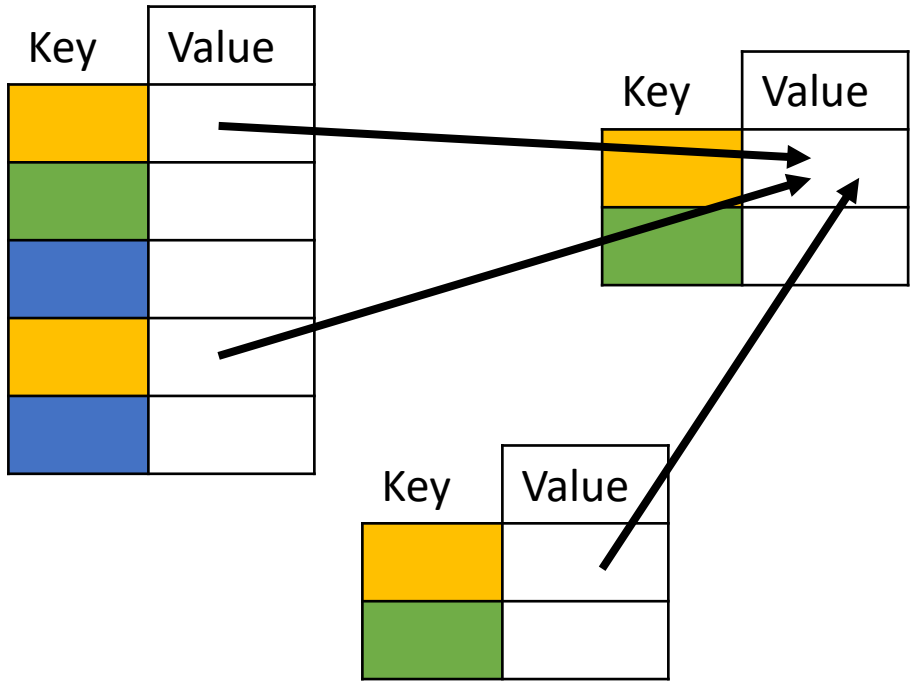
Spark RDD operations

Key-value operations




```
reduceByKey(func)
groupByKey()
sortByKey()
```

Interaction with other RDD




```
join(other)
cogroup(other)
```

Initialize Spark Application

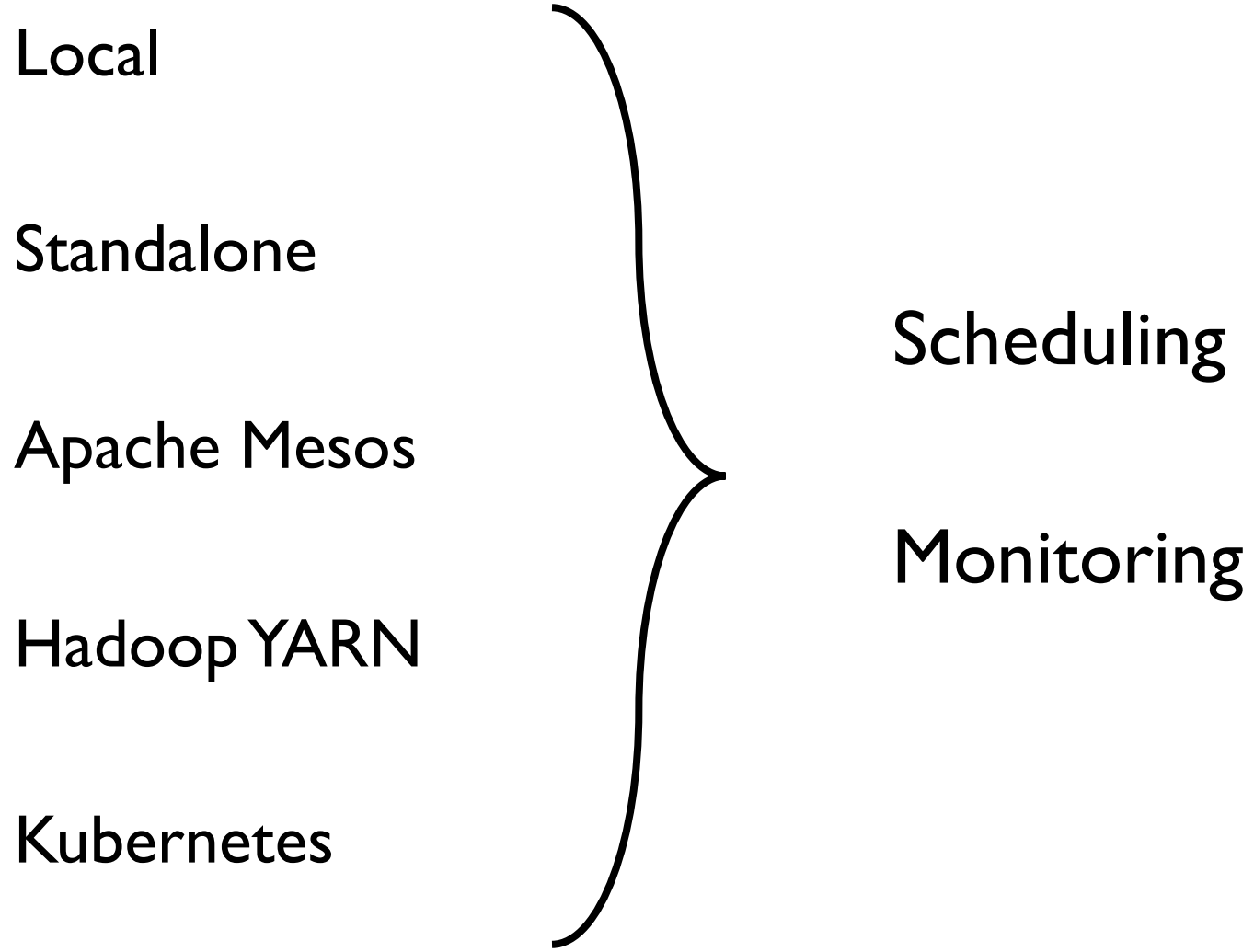
SparkConf()

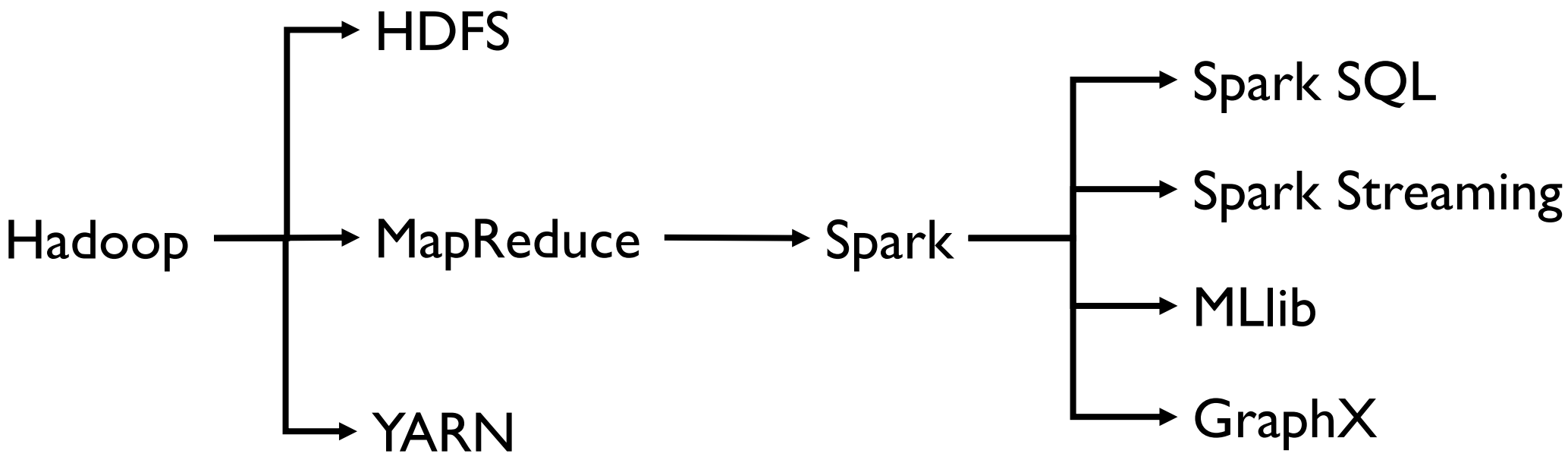
setAppName(appName) Define application name in cluster

setMaster(master) URL of cluster manager

SparkContext(conf) Access to cluster (No needed in the terminal)

Cluster Options for Spark





https://hadoop.apache.org

https://spark.apache.org

How to run my application?

1) Have a Apache Spark distribution, usually prebuilt for Apache Hadoop
2) Choose programming language:

   Java, Scala, **Python**, R, SQL

3) In file add library :

```
from pyspark.sql import SparkSession
```

4) Execute defining number of cores involve (to run locally)

```
bin/spark-submit --master local[4] hello_world_spark.py
```

Monitoring

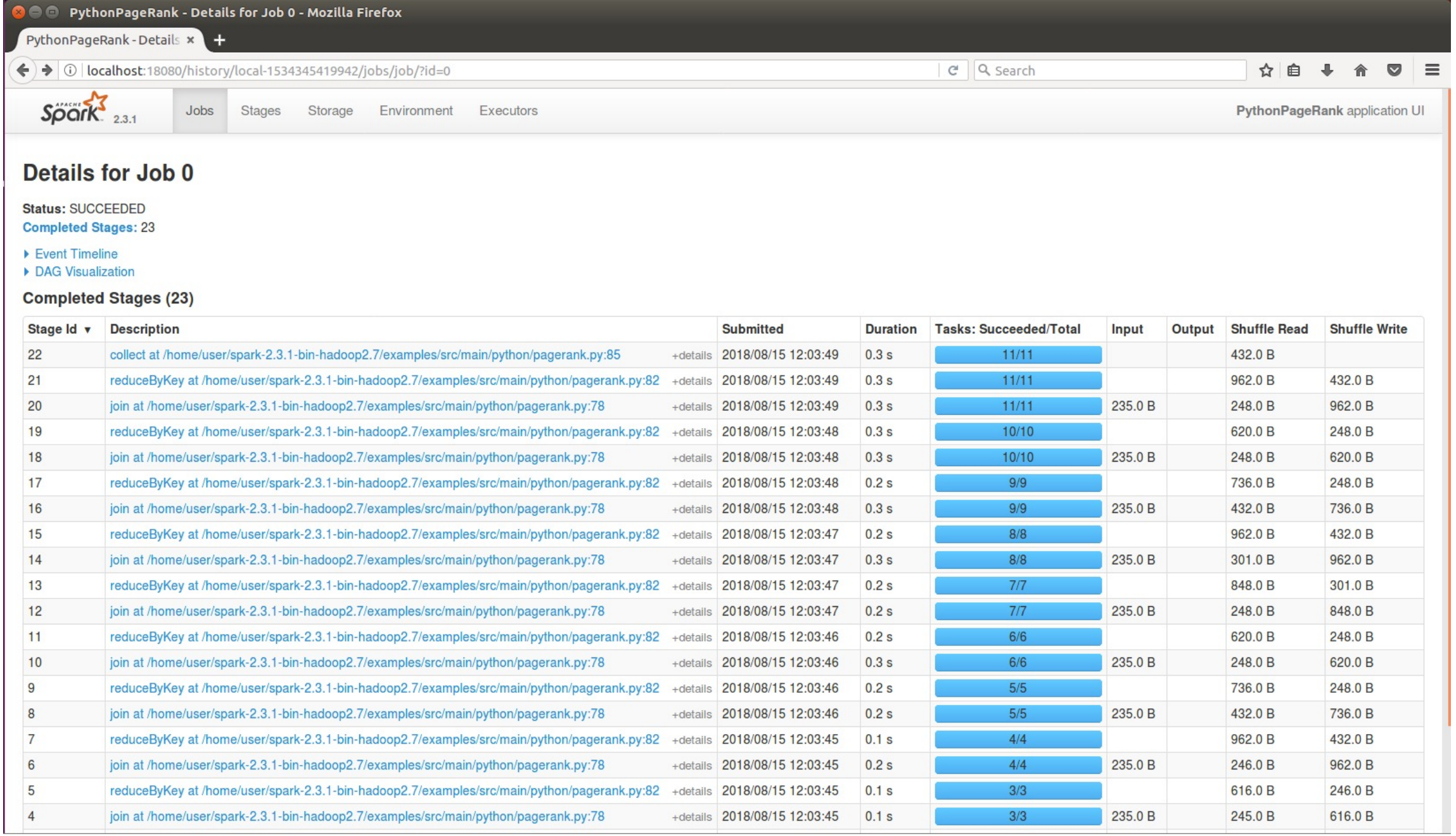
- Create folder /tmp/spark-events (only once)
- Execute with flag

```
bin/spark-submit --master local[4] --conf spark.eventLog.enabled=true pi.py
```

- Recover post-mortem data

```
./sbin/start-history-server.sh
```

- Check the info at `http://localhost:18080`
- During execution you can check `http://localhost:4040`


PythonPageRank - Details for Job 0 - Mozilla Firefox
PythonPageRank - Details
localhost:18080/history/local-1534345419942/jobs/job/?id=0
Search
Spark 2.3.1
Jobs
Stages
Storage
Environment
Executors
PythonPageRank application UI
Details for Job 0
Status: SUCCEEDED
Completed Stages: 23
Event Timeline
DAG Visualization
Completed Stages (23)
Stage Id ▼ | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write
22 | collect at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:85 +details | 2018/08/15 12:03:49 | 0.3 s | 11/11 | | | 432.0 B |
21 | reduceByKey at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:82 +details | 2018/08/15 12:03:49 | 0.3 s | 11/11 | | | 962.0 B | 432.0 B
20 | join at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:78 +details | 2018/08/15 12:03:49 | 0.3 s | 11/11 | 235.0 B | | 248.0 B | 962.0 B
19 | reduceByKey at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:82 +details | 2018/08/15 12:03:48 | 0.3 s | 10/10 | | | 620.0 B | 248.0 B
18 | join at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:78 +details | 2018/08/15 12:03:48 | 0.3 s | 10/10 | 235.0 B | | 248.0 B | 620.0 B
17 | reduceByKey at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:82 +details | 2018/08/15 12:03:48 | 0.2 s | 9/9 | | | 736.0 B | 248.0 B
16 | join at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:78 +details | 2018/08/15 12:03:48 | 0.3 s | 9/9 | 235.0 B | | 432.0 B | 736.0 B
15 | reduceByKey at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:82 +details | 2018/08/15 12:03:47 | 0.2 s | 8/8 | | | 962.0 B | 432.0 B
14 | join at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:78 +details | 2018/08/15 12:03:47 | 0.3 s | 8/8 | 235.0 B | | 301.0 B | 962.0 B
13 | reduceByKey at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:82 +details | 2018/08/15 12:03:47 | 0.2 s | 7/7 | | | 848.0 B | 301.0 B
12 | join at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:78 +details | 2018/08/15 12:03:47 | 0.2 s | 7/7 | 235.0 B | | 248.0 B | 848.0 B
11 | reduceByKey at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:82 +details | 2018/08/15 12:03:46 | 0.2 s | 6/6 | | | 620.0 B | 248.0 B
10 | join at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:78 +details | 2018/08/15 12:03:46 | 0.3 s | 6/6 | 235.0 B | | 248.0 B | 620.0 B
9 | reduceByKey at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:82 +details | 2018/08/15 12:03:46 | 0.2 s | 5/5 | | | 736.0 B | 248.0 B
8 | join at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:78 +details | 2018/08/15 12:03:46 | 0.2 s | 5/5 | 235.0 B | | 432.0 B | 736.0 B
7 | reduceByKey at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:82 +details | 2018/08/15 12:03:45 | 0.1 s | 4/4 | | | 962.0 B | 432.0 B
6 | join at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:78 +details | 2018/08/15 12:03:45 | 0.2 s | 4/4 | 235.0 B | | 246.0 B | 962.0 B
5 | reduceByKey at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:82 +details | 2018/08/15 12:03:45 | 0.1 s | 3/3 | | | 616.0 B | 246.0 B
4 | join at /home/user/spark-2.3.1-bin-hadoop2.7/examples/src/main/python/pagerank.py:78 +details | 2018/08/15 12:03:45 | 0.1 s | 3/3 | 235.0 B | | 245.0 B | 616.0 B

References

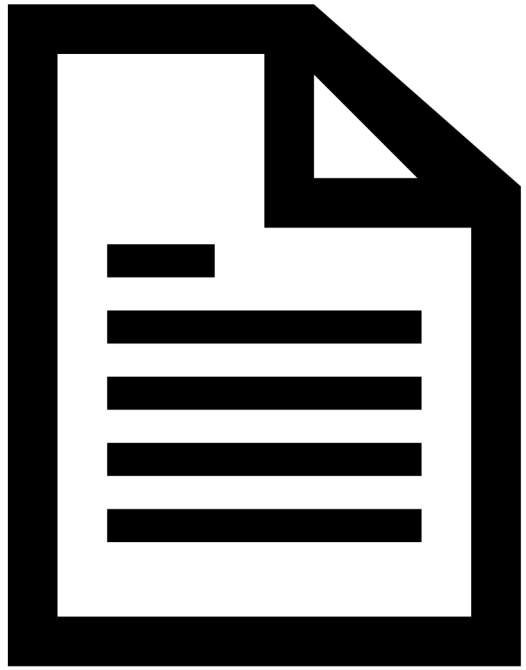
https://spark.apache.org/docs/latest/index.html

Intro to RDDs, extensions, APIs and deploying

http://hadoop.apache.org/docs/current/index.html

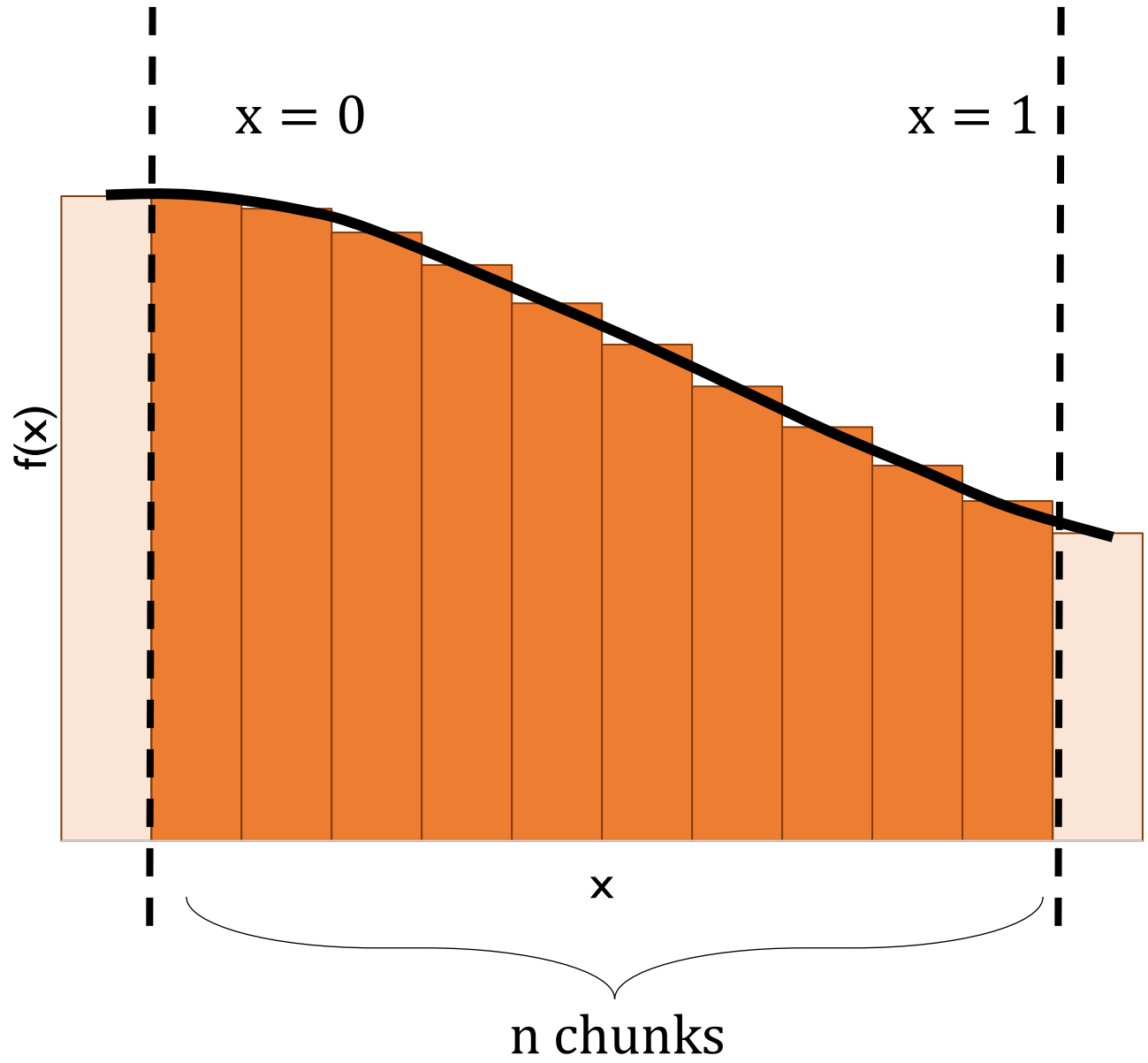
Hadoop overview, including HDFS and YARN

Spark: The Definitive Guide: Big Data Processing Made Simple, Matei Zaharia, 2018.

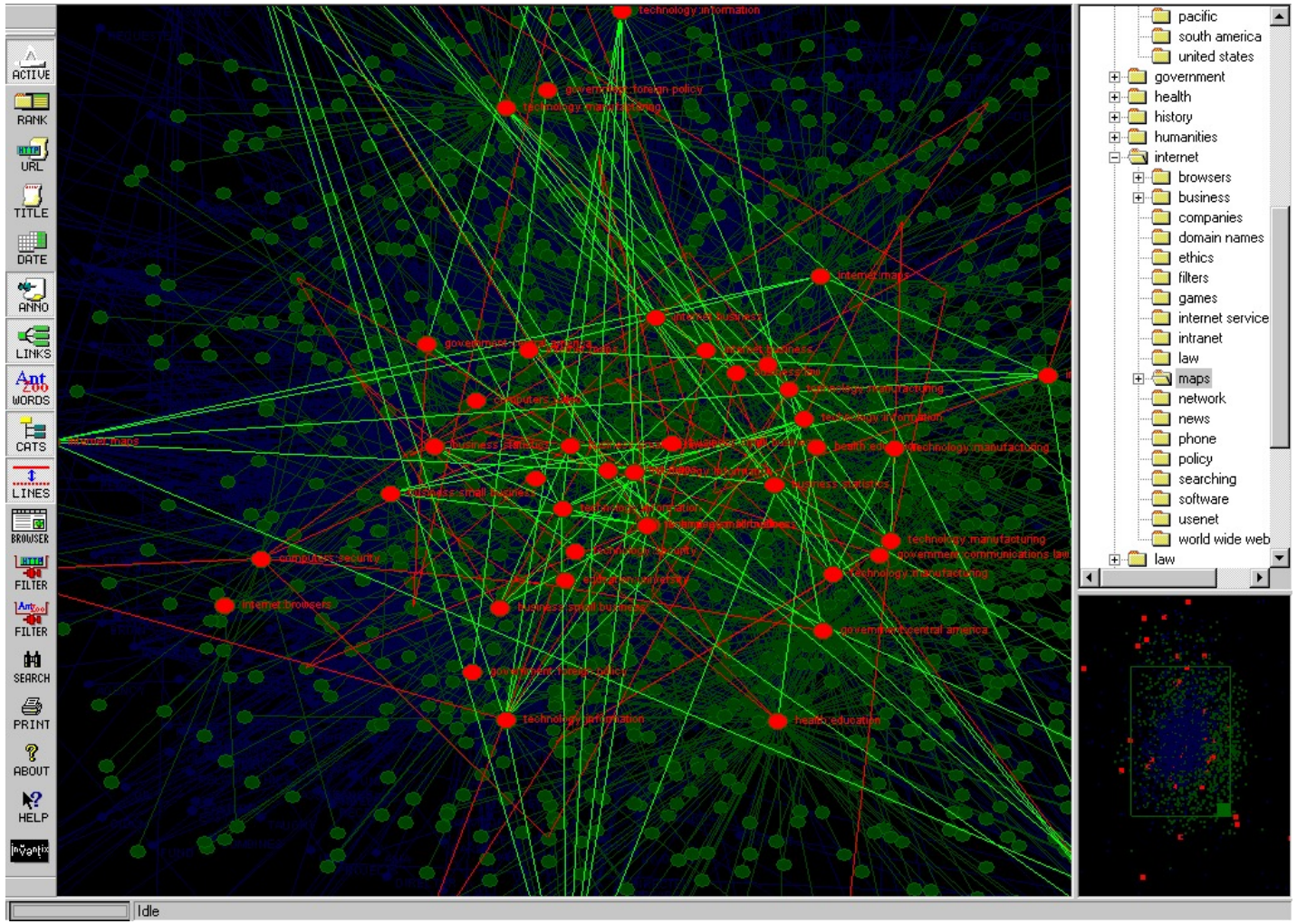


Example: Compute Pi ($\pi = 3.14159 \ldots$)

$$\pi = \int_0^1 \frac{4}{1 + x^2} dx$$

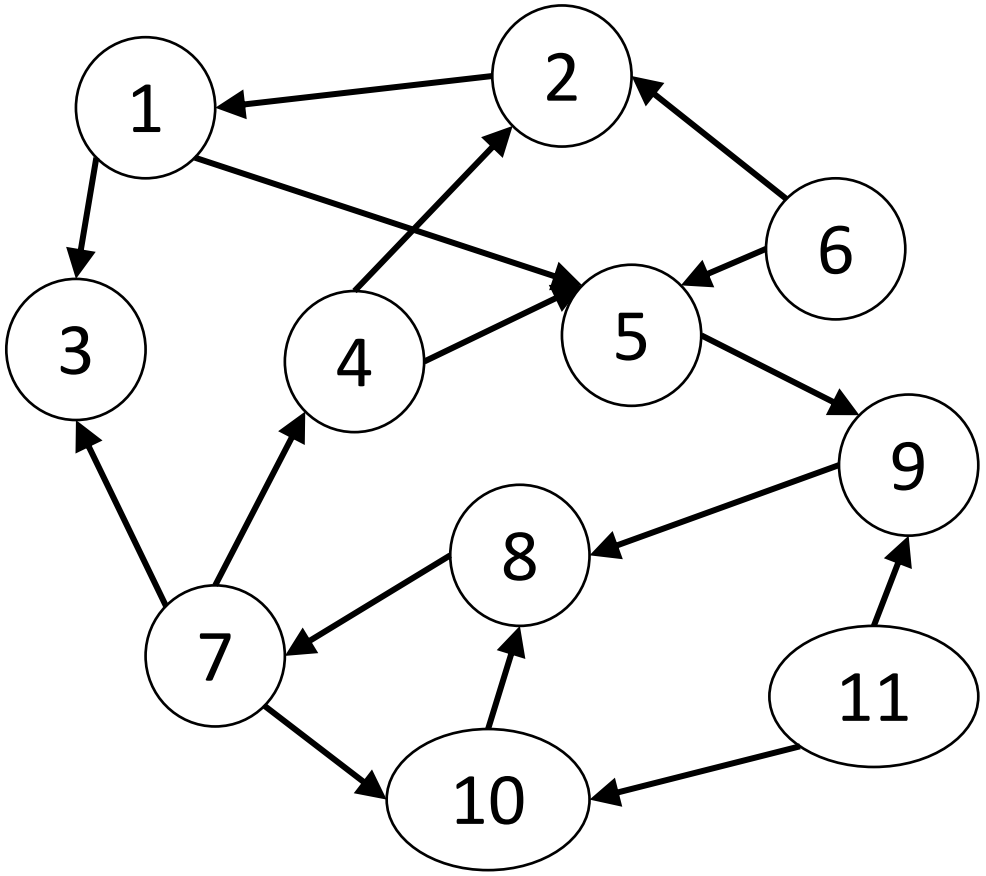


Example: PageRank

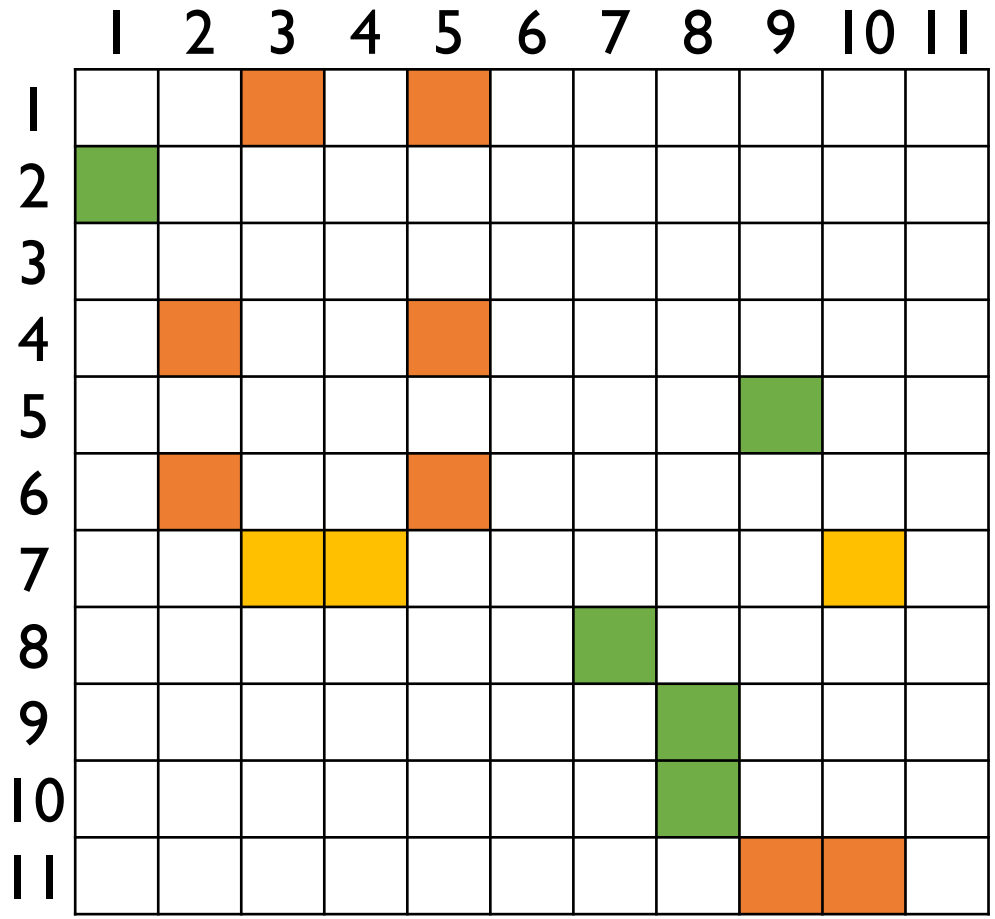


An Atlas Of Cyberspaces: Internet Cartographer by Inventix Sotware, mid 2000s

Example: PageRank



$S =$



Example: PageRank

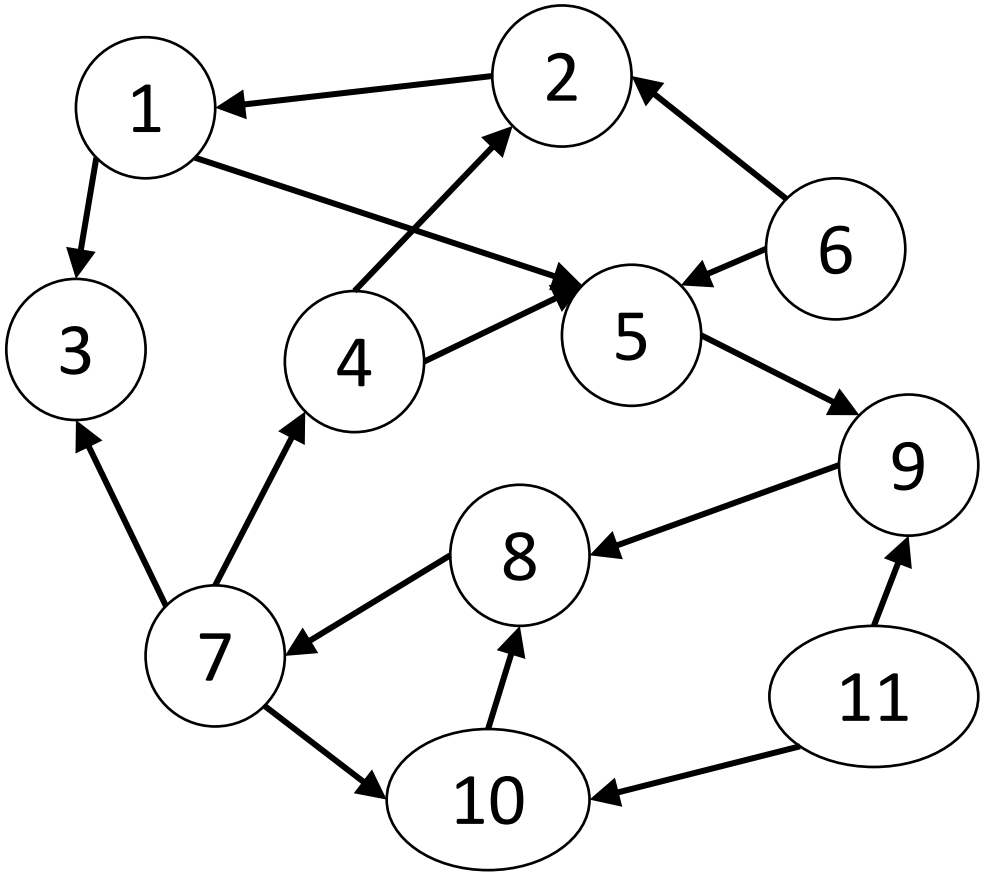
Find left eigenvalue $0.85 \cdot S + 0.15\,(1 \cdot 1^{\mathsf{T}})$

$$r^{(k+1)} = 0.85\underbrace{S^{\mathsf{T}} r^{(k)}} + 0.15 \cdot 1$$

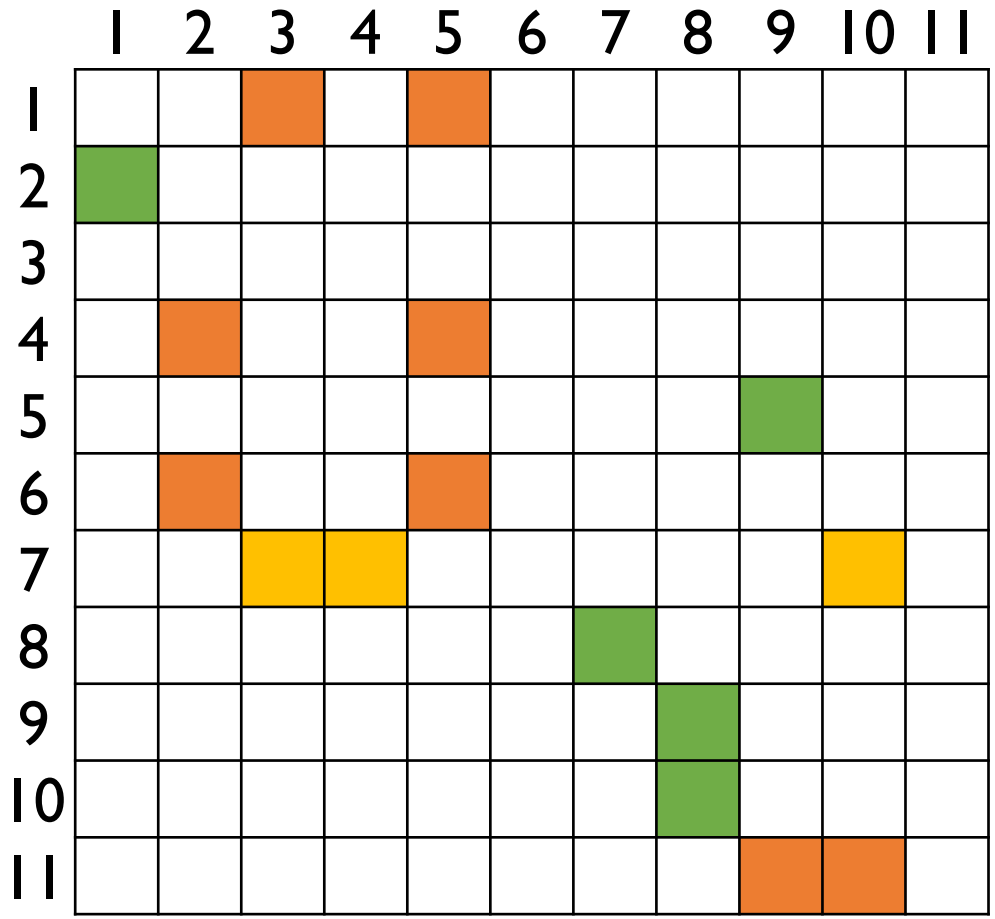
$$c_j = \sum_{i\ ant.of\ j} \frac{r_i}{\#outlinks\ i}$$

Example: PageRank

$(S^T r)_9$?

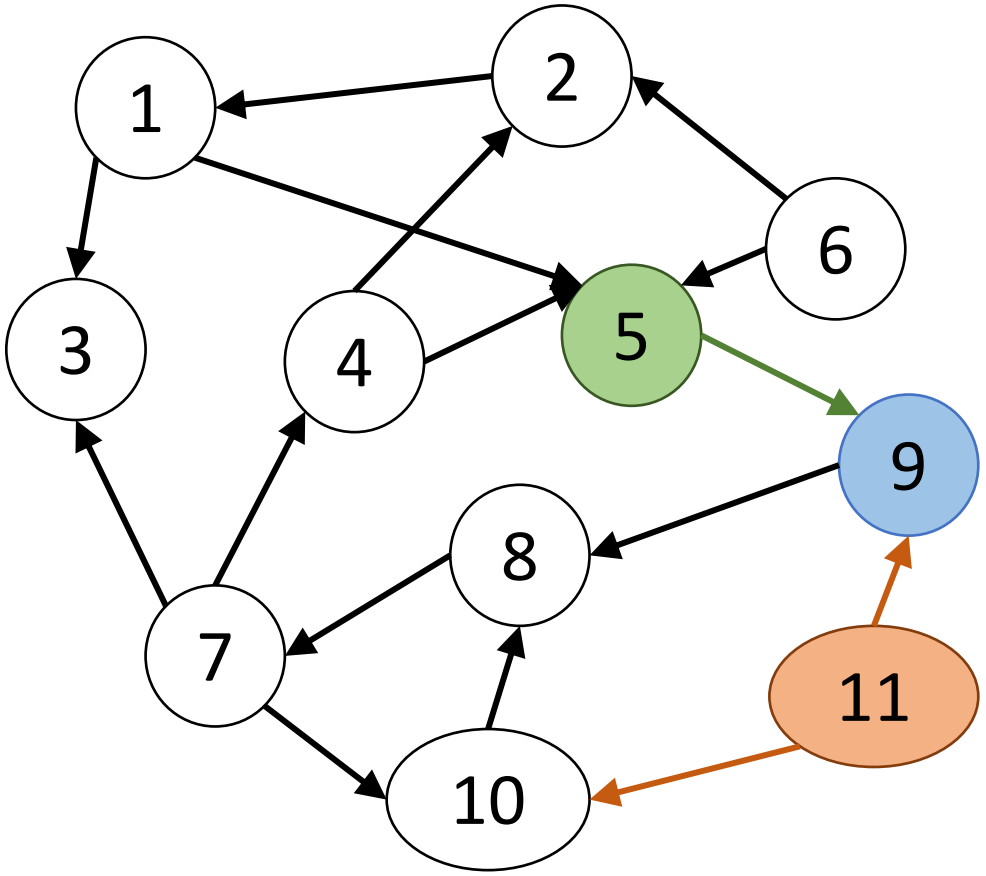


$S =$

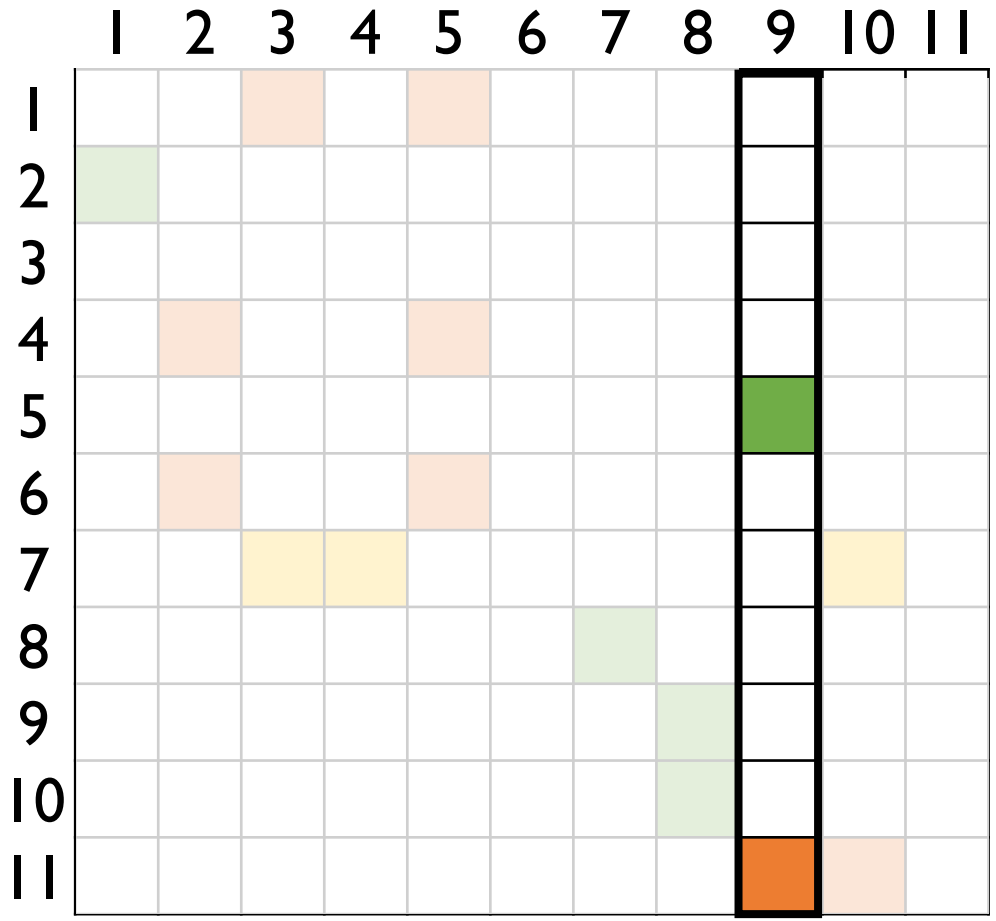


Example: PageRank

$$(S^T r)_9 \; ? \; = \frac{r_5}{1} + \frac{r_{11}}{2}$$



$S =$



antecedent	descendant
1	[3,5]
2	[1]
3	[]
4	[2,5]
5	[9]
6	[2,5]
7	[3,4,10]
8	[7]
9	[8]
10	[8]
11	[9,10]

Node	r_i
1	1
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	1
10	1
11	1

Join + map

$$i \to j:\ c_i(j) = \frac{r_i}{\#outlinks\ i}$$

antecedent	contribution
1	[[3,1/2], [5,1/2]]
2	[[1,1]]
3	[]
4	[[2,1/2], [5,1/2]]
5	[[9,1]]
6	[[2,1/2], [5,1/2]]
7	[[3,1/3], [4,1/3], [10,1/3]]
8	[[7,1]]
9	[[8,1]]
10	[[8,1]]
11	[[9,1/2], [10,1/2]]

Reduce by key

$$u_j = \sum_{i \to j} c_i(j)$$

descendant	contribution
1	1
2	1/2 +1/2
3	1/2+1/3
4	1/3
5	1/2+1/2+1/2
6	0
7	1
8	1+1
9	1+1/2
10	1/2+1/3
11	0

map

$$r_j = 0.85u_j + 0.15$$

Node	r_i
1	1
2	1
3	0.86
4	0.43
5	1.425
6	0.15
7	1
8	1.85
9	1.425
10	0.86
11	0