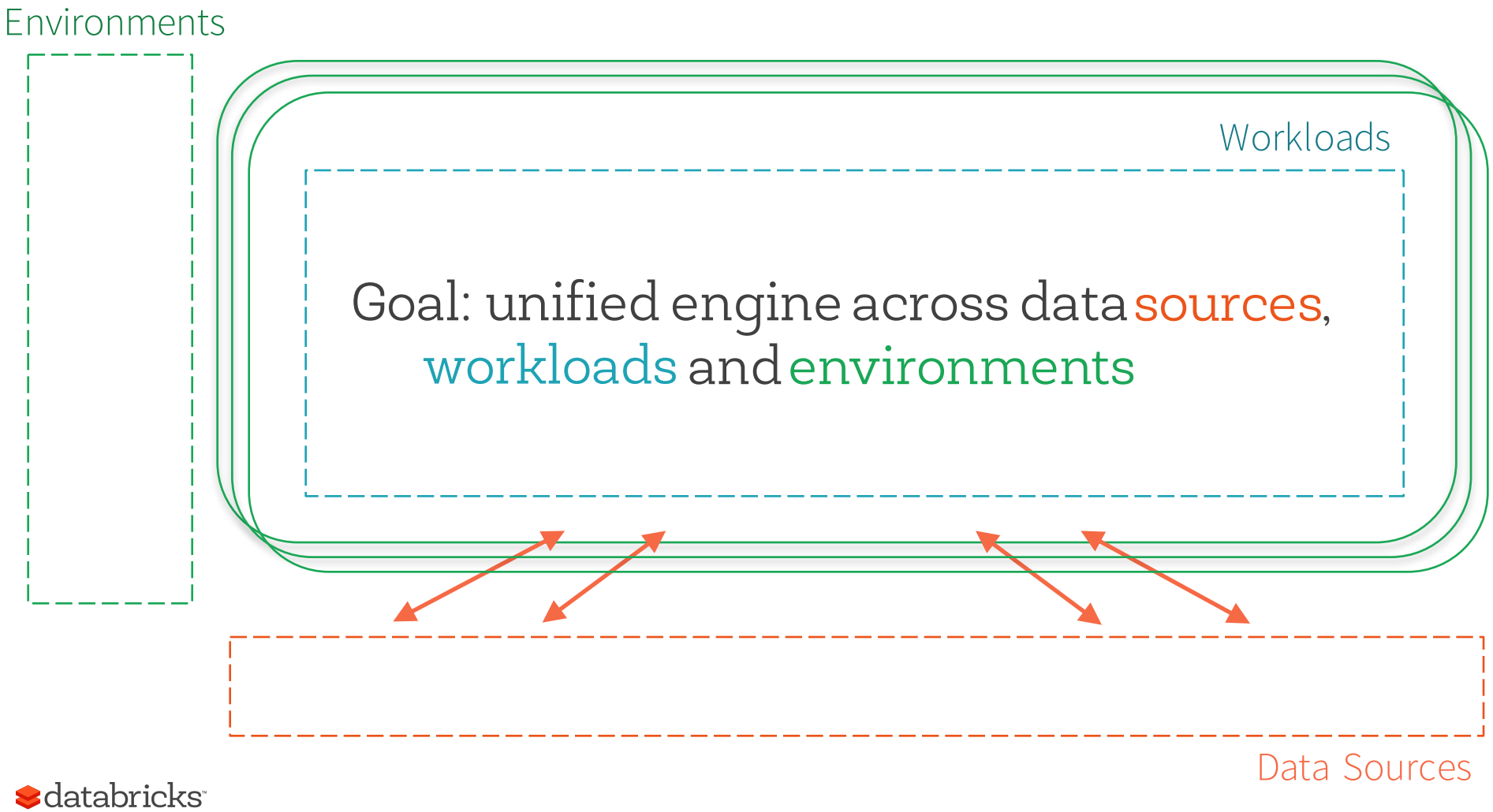
Exploring Wikipedia with Spark

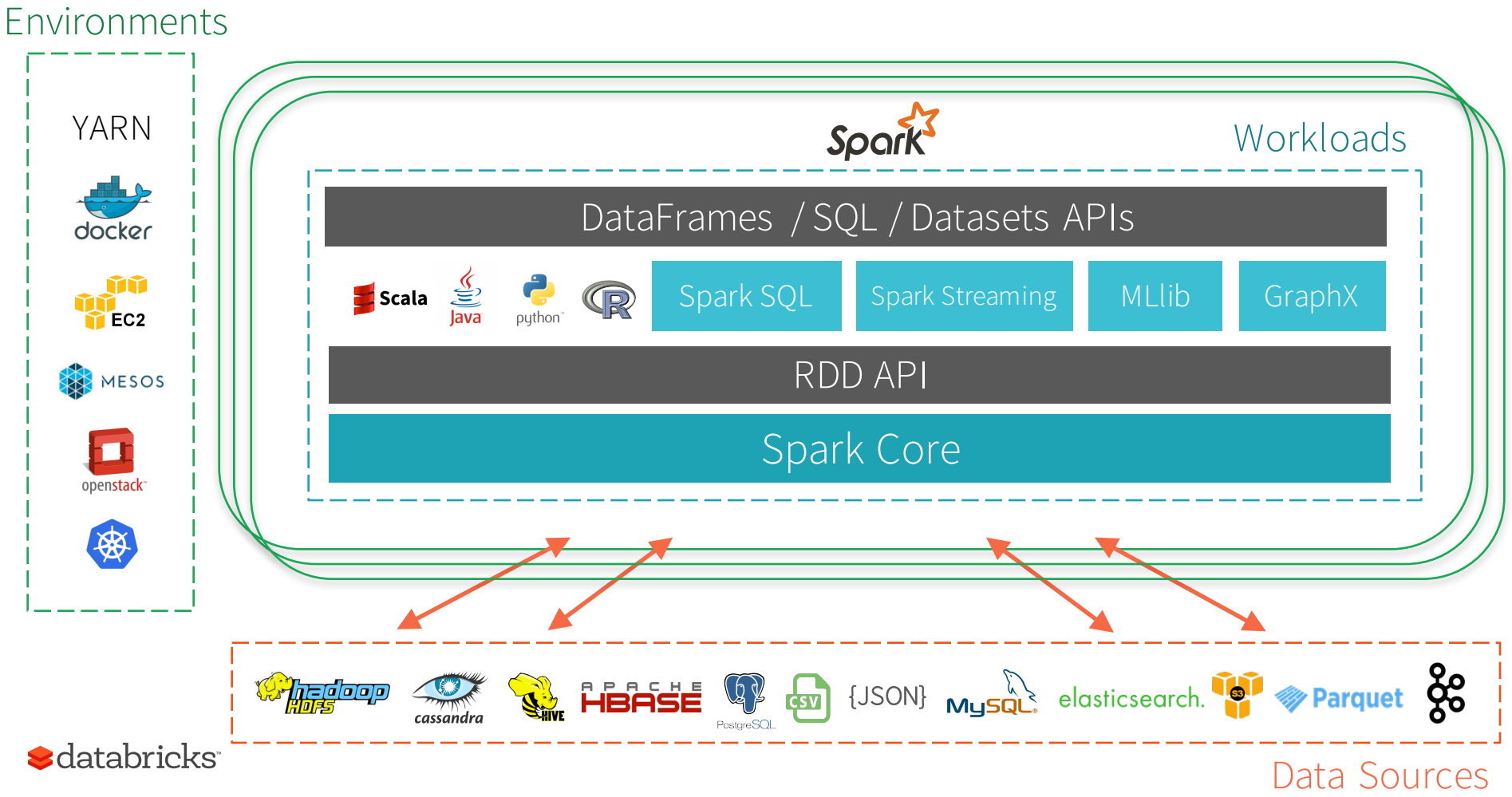
databrickstraining

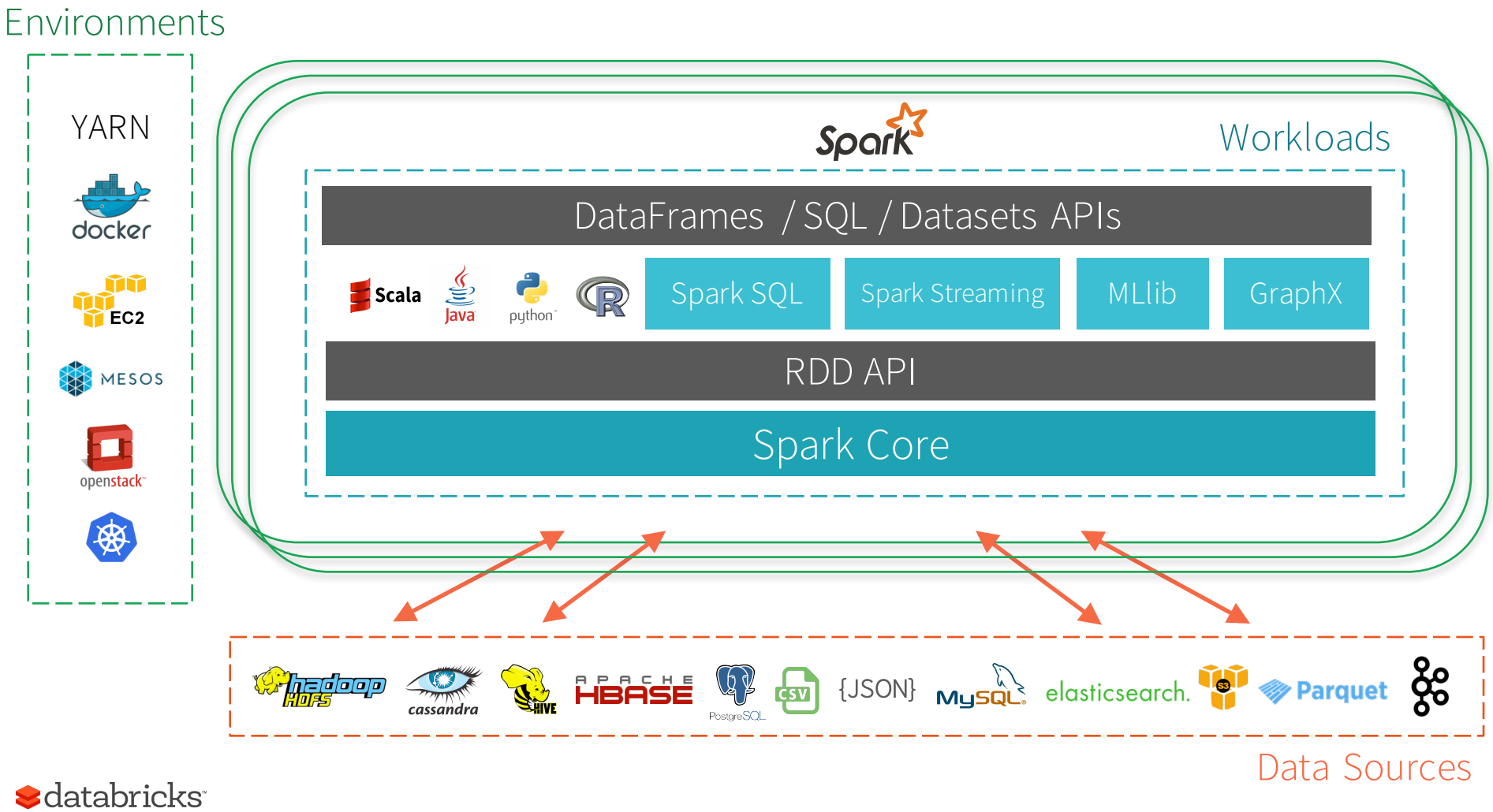
Spark Summit West 2016


Environments
Workloads
Goal: unified engine across data sources, workloads and environments
Data Sources

databricks

Goal: unified engine across data sources, workloads and environments




Environments
YARN
docker
EC2
MESOS
openstack
Spark
Workloads
DataFrames / SQL / Datasets APIs
Scala
Java
python
Spark SQL
Spark Streaming
MLlib
GraphX
RDD API
Spark Core
hadoop HDFS
cassandra
HIVE
APACHE HBASE
PostgreSQL
CSV
{JSON}
MySQL
elasticsearch.
Parquet
Data Sources
databricks


WIKIMEDIA
FOUNDATION

Today's Objectives

- Learn just enough to build simple prototypes/POCs with Spark
- Fast paced, high level overview of all major Spark components
- Hands on with Spark's programming APIs *(DataFrame/SQL, RDD, Datasets)*
- Overview of Spark architecture: Core, Streaming, Standalone Mode, DAG
- Mix of beginner + advanced topics
- Not all slides/labs covered *(reference & homework material)*
- Lots of ideas, code & datasets to play around with after class

Timings: ~6 hours total

morning: APIs: (DataFrames, SQL), Architecture

afternoon: APIs: (Datasets, RDD), GraphX/ML, Streaming

~5pm: end!

10:30am – 11am: Break

noon – 1pm: Lunch

3pm – 3:30pm: Break

Data

Spark Analytics

Pageviews (March 2015) - 255 MB → Dataframes

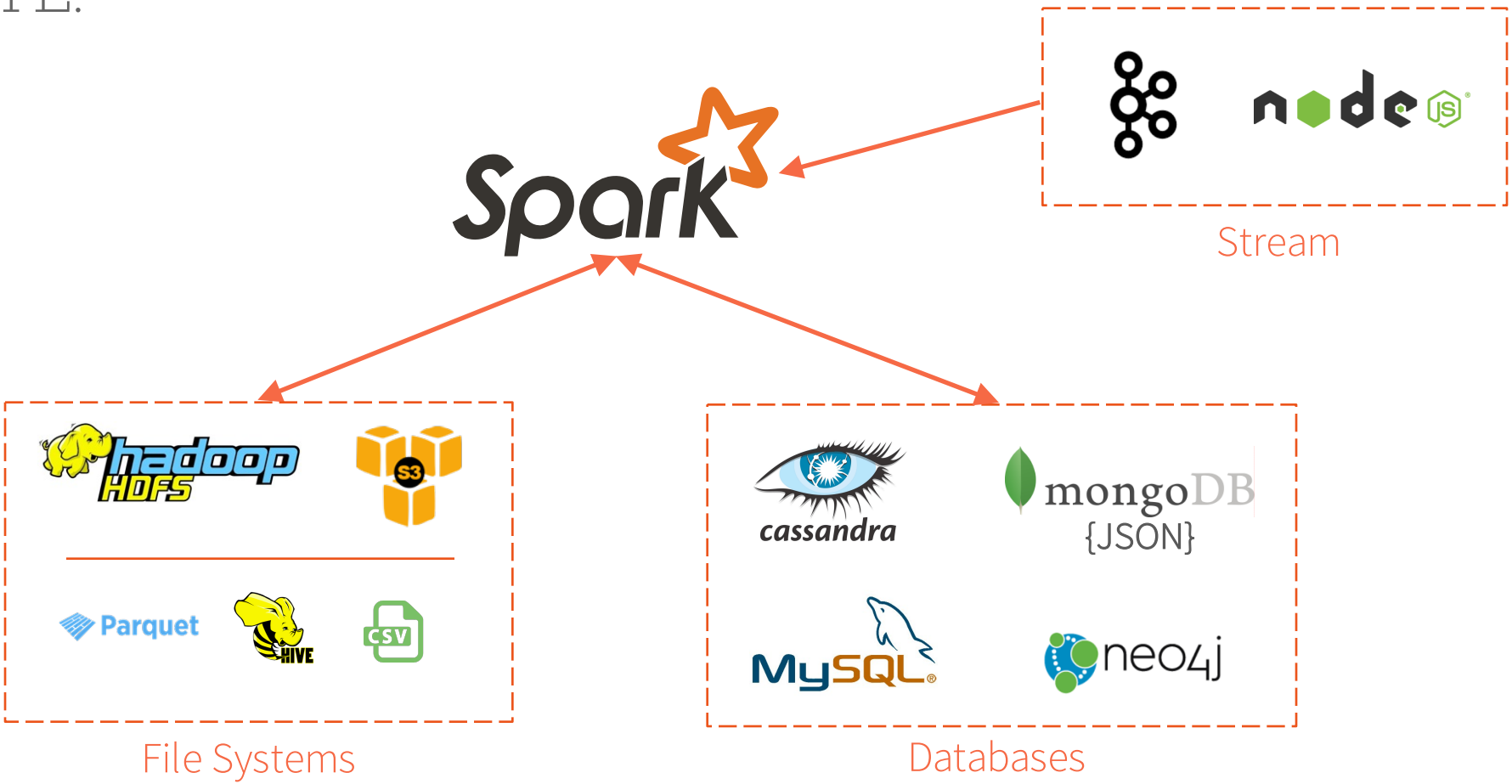
Clickstream (Feb 2015) - 1.2 GB → Dataframes, SQL, GraphX

Pagecounts (last hour) - ~500 MB → RDD, Datasets, Dataframes

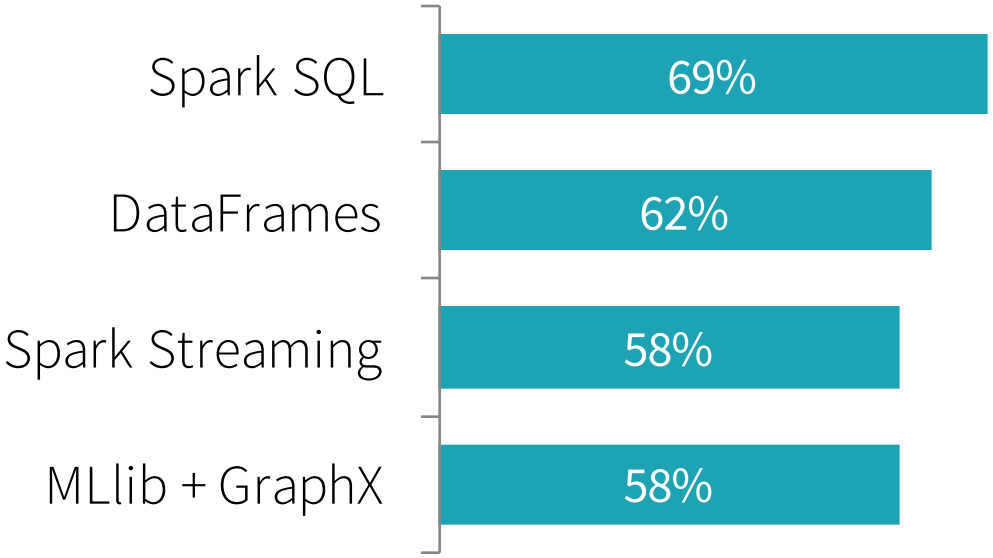
English Wikipedia (Mar 5, 2016) - 54 GB → Dataframes, SQL, ML

6 Lang Wikipedias (live streams) → Streaming, RDD, Dataframes, SQL

ETL:


Spark
Stream
hadoop HDFS
S3
Parquet
HIVE
CSV
File Systems
cassandra
mongoDB
{JSON}
MySQL
neo4j
Databases

Which Libraries Do People Use?



75%

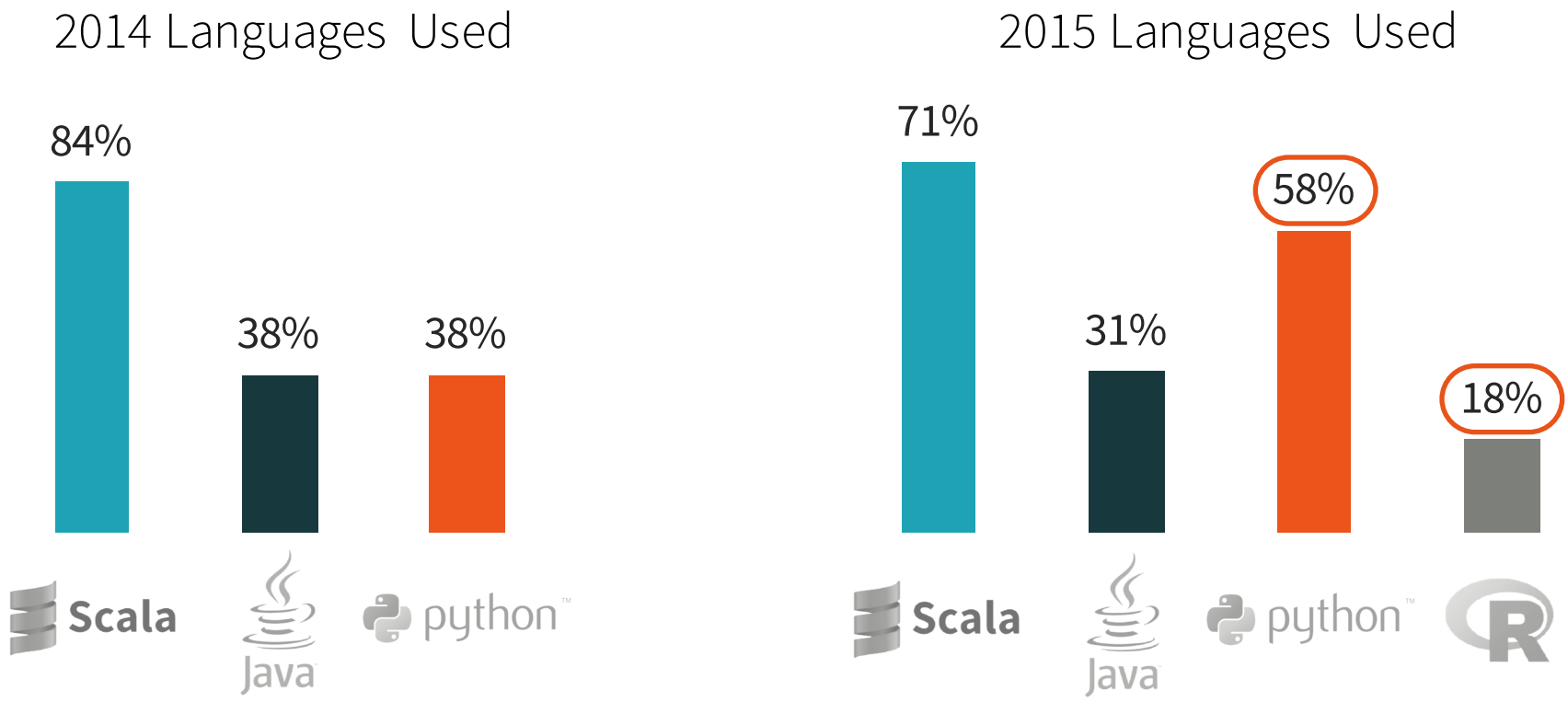
of users use 2+ components

51%

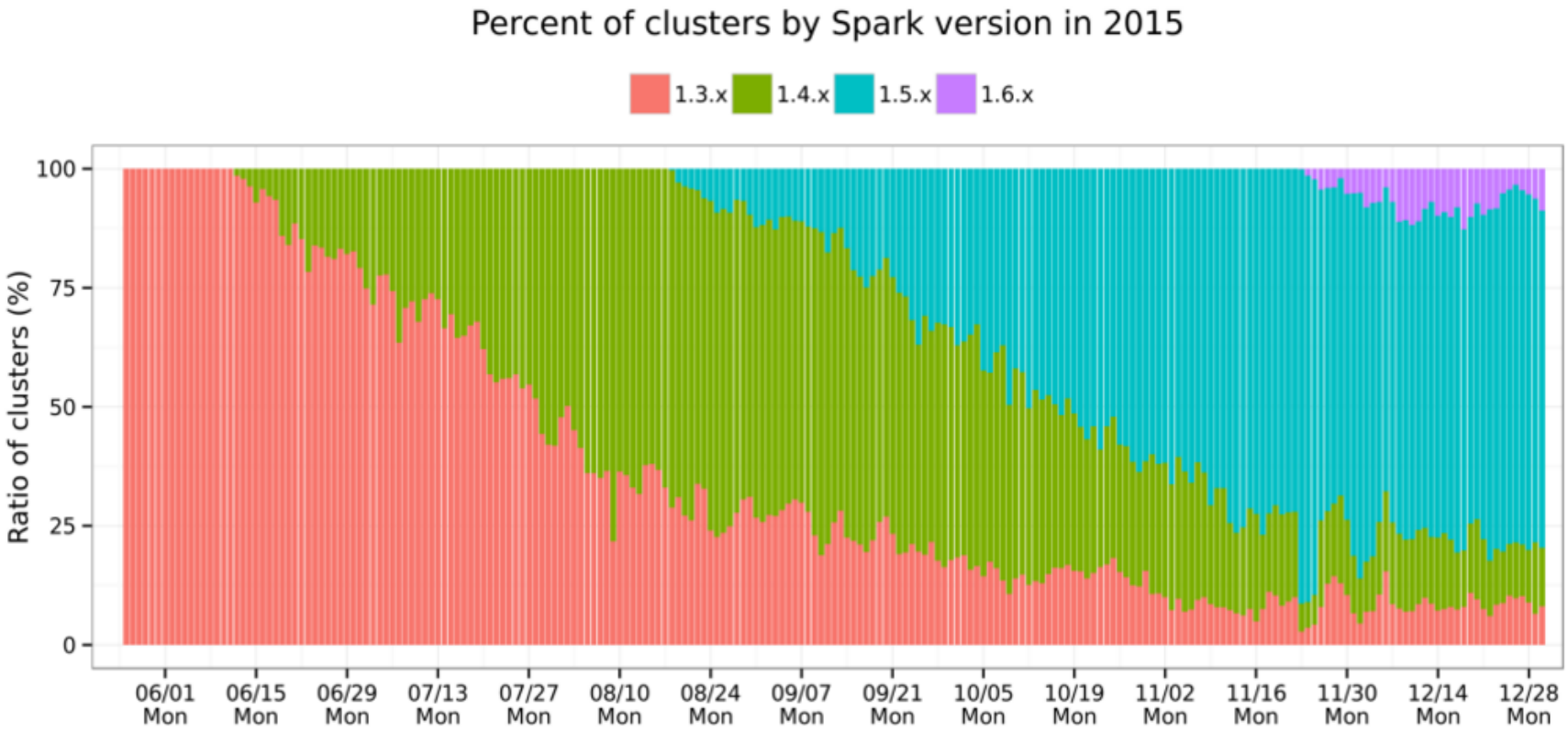
of users use 3+ components

Source: http://dbricks.co/1PtWxSu

Which Languages Do People Use?



Source: http://dbricks.co/1PtWxSu

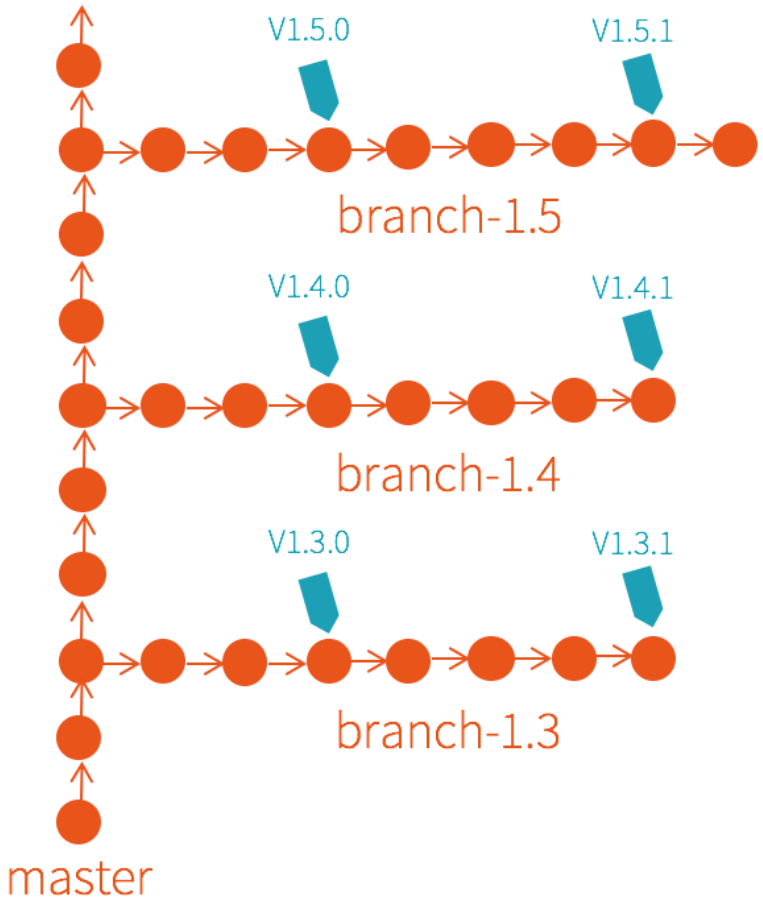


Reynold Xin @rxin · Jan 14

The most interesting thing in Spark 2015 Year in Review post is the version dist over time. databricks.com/blog/2016/01/0...

9 19

What do the version #s mean?



Spark project goal

Support diverse workflows:

- Batch *(jobs that run for hours)*
- Interactive / Ad Hoc *(via shell or notebooks)*
- Iterative computations *(graph and ML)*
- Streaming

Large Scale Usage:

Largest cluster: 8000 nodes 

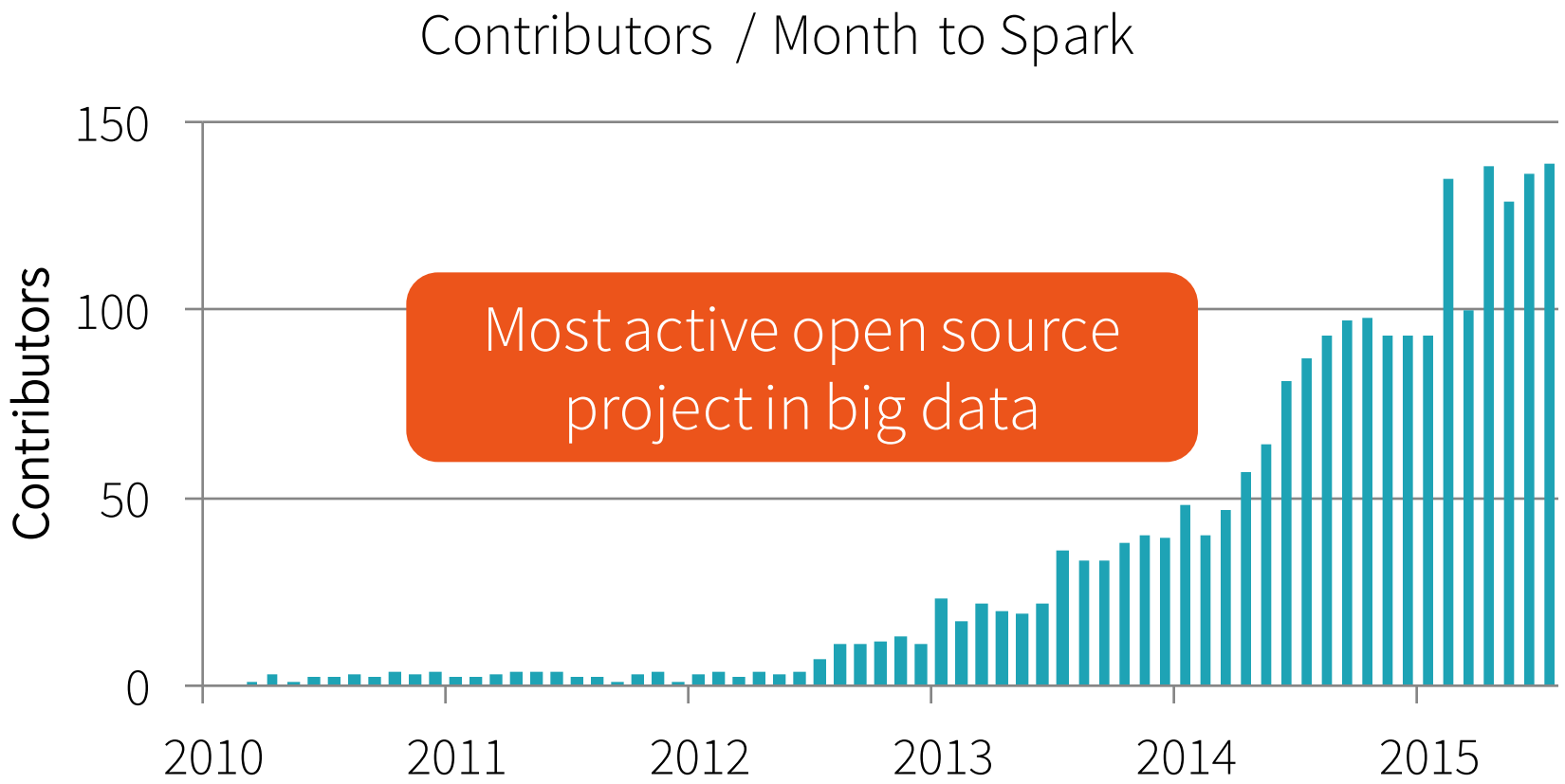
Largest single job: 1 petabyte 

Top streaming intake: 1 TB/hour 

2014 on-disk 100 TB sort record: 23 mins / 207 EC2 nodes

Spark Users: 1000+ production deployments

Community Growth:



Functional Programming & Scala

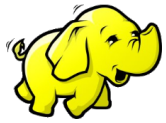
Typically used when a system's performance and integrity are both critical

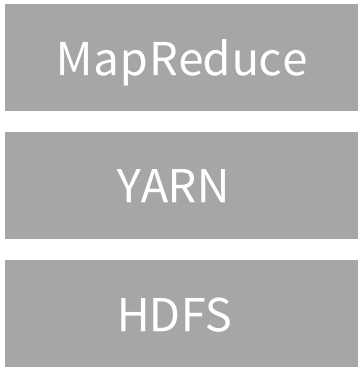
2 concepts:

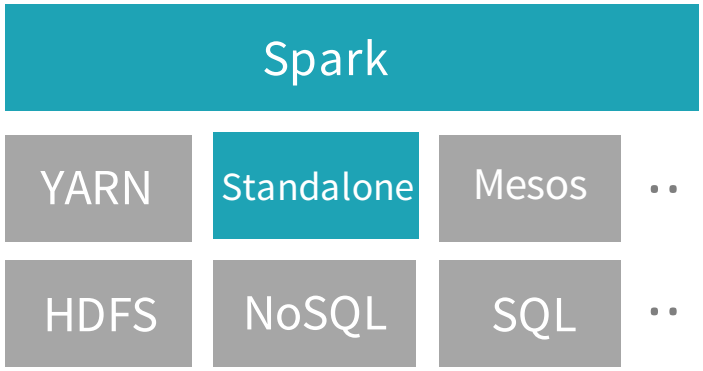
- **Immutable Data:** Create new data structures instead of modifying ones that already exist
- **Stateless:** Perform every task as if for the first time, with no knowledge of earlier state

Benefits: *simplifies multithreading, removes bugs around race conditions and deadlocks, reusable functions*

Comparison to Hadoop





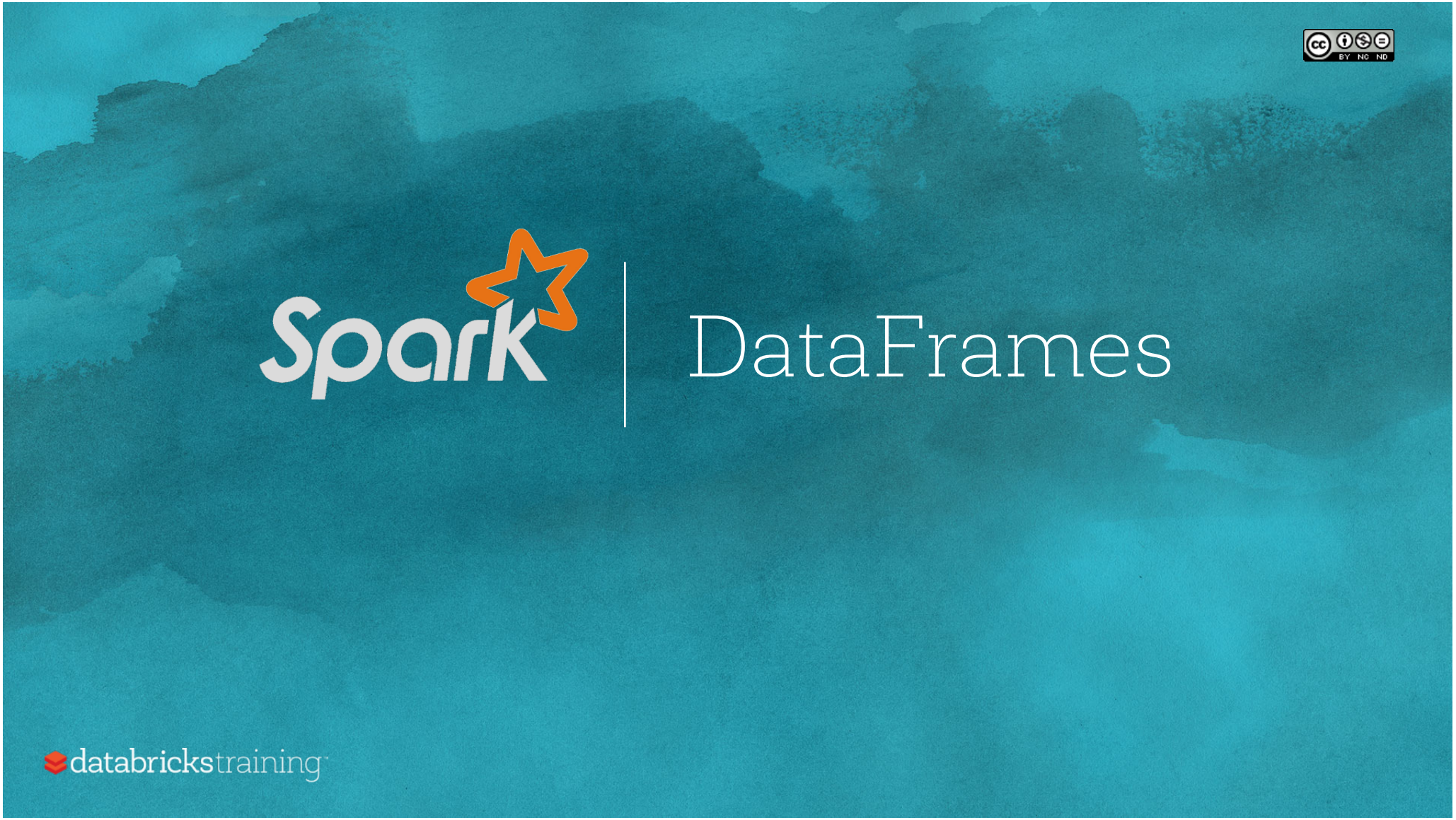


Hadoop: combined compute, resource management + storage

Spark: independent of resource management + storage layers

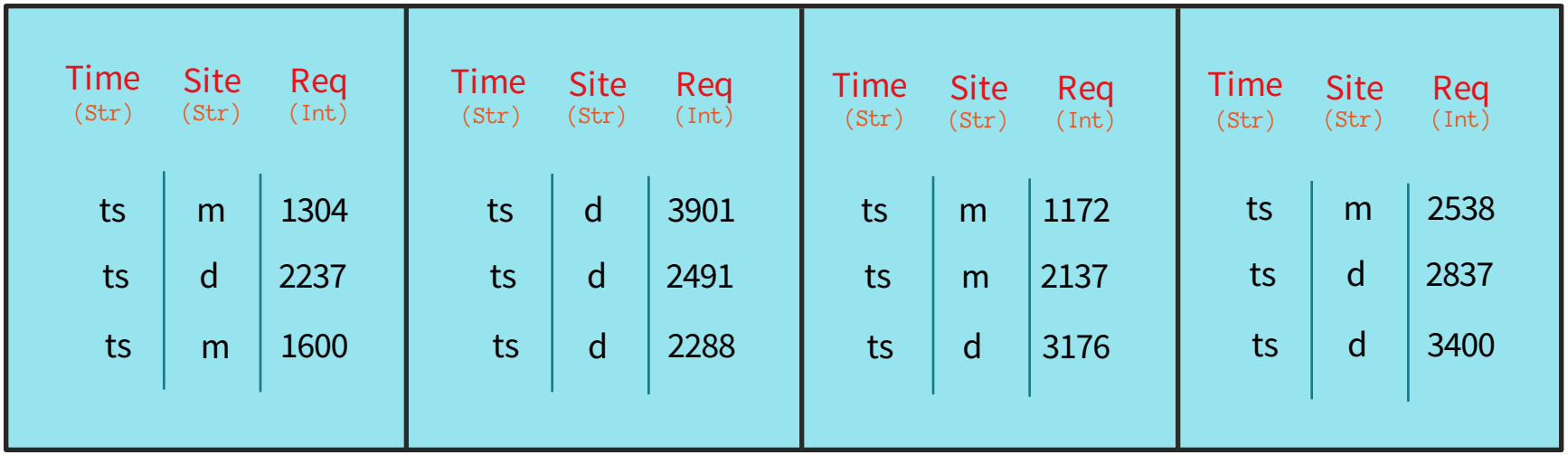
“Simple things should be simple, complex things should be possible”

-Alan Kay


Spark
DataFrames
databrickstraining

I have used DataFrames before (in any language)…

DataFrame: pageviewsDF




```
df.rdd.partitions.size  =  4
```

Spark Operations =

TRANSFORMATIONS

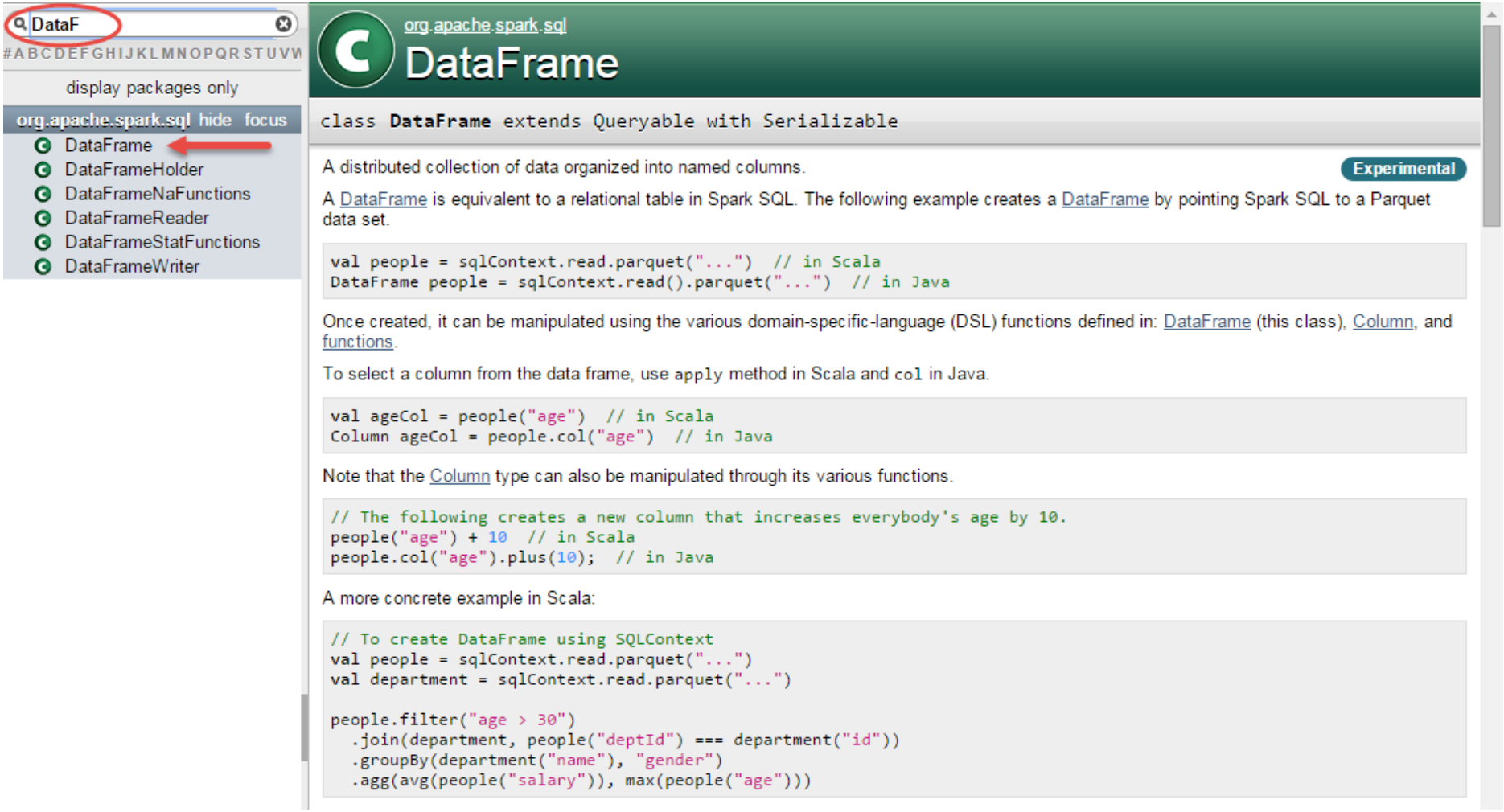
+

ACTIONS

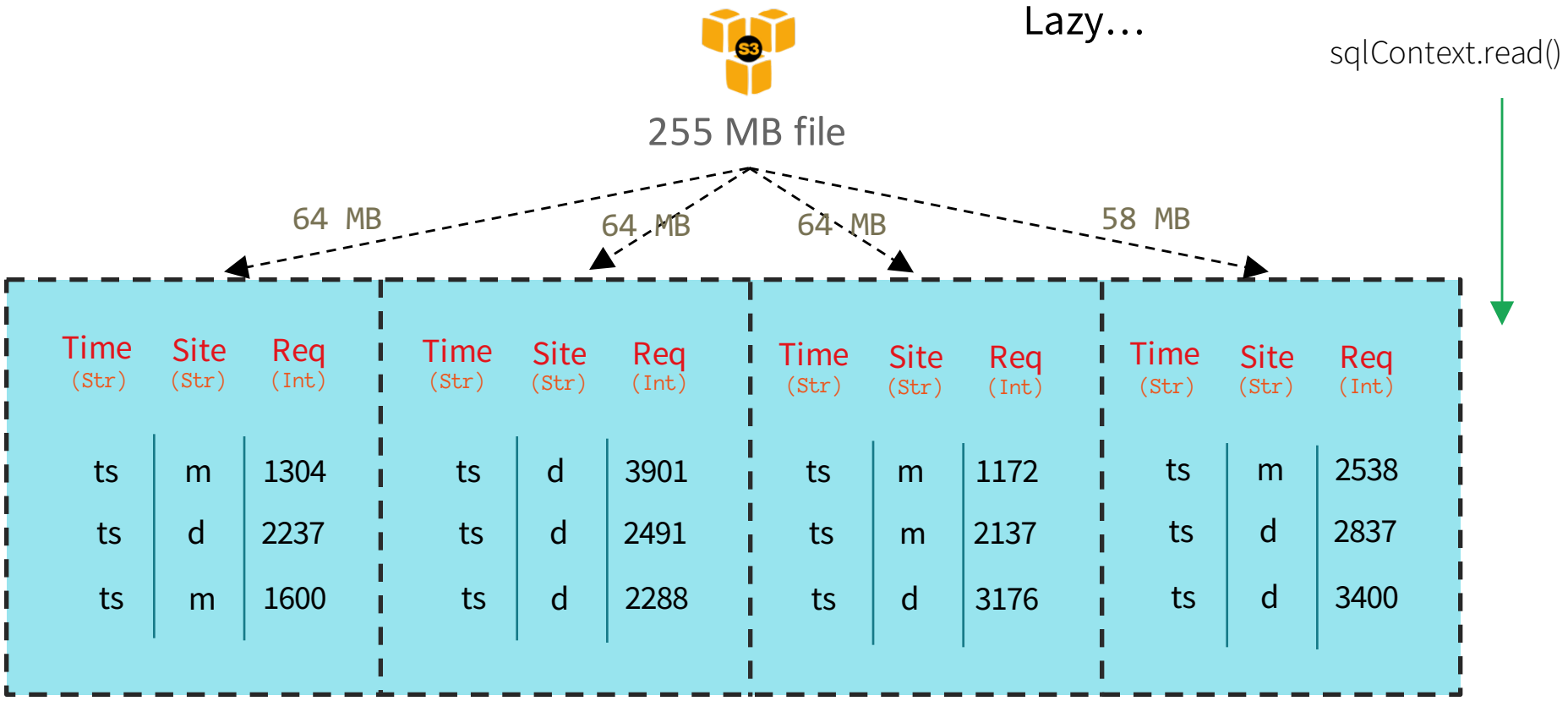
DataFrame: Transformations and Actions

Transformations *(lazy)*	Actions
`orderBy`	`show`
`filter`	`count`
`groupBy`	`take`
`select`	`collect`
`drop`	`save`
`join`	

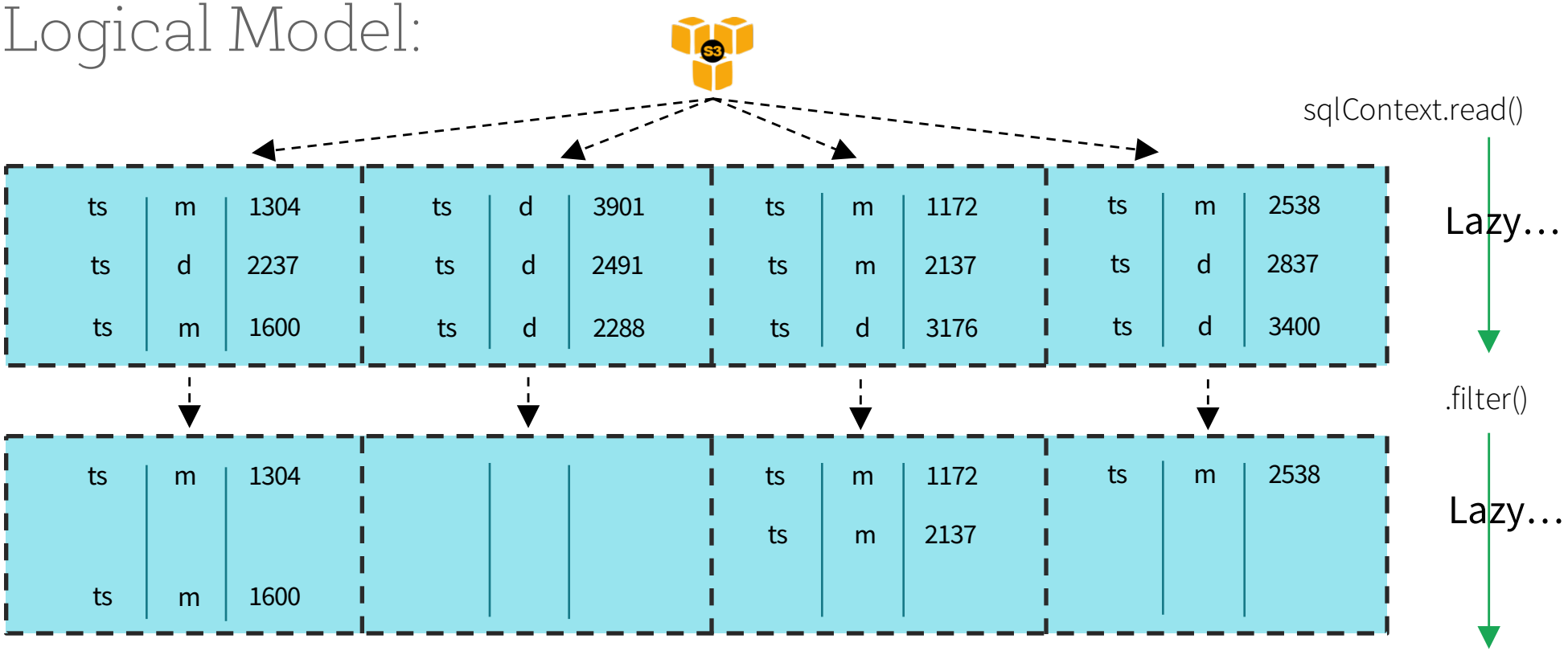
Transformations contribute to a query plan,
but nothing is executed until an action is called


DataF
#ABCDEFGHIJKLMNOPQRSTUVW
display packages only
org.apache.spark.sql hide focus
DataFrame
DataFrameHolder
DataFrameNaFunctions
DataFrameReader
DataFrameStatFunctions
DataFrameWriter
org.apache.spark.sql
DataFrame
class DataFrame extends Queryable with Serializable
A distributed collection of data organized into named columns.
Experimental
A DataFrame is equivalent to a relational table in Spark SQL. The following example creates a DataFrame by pointing Spark SQL to a Parquet data set.
val people = sqlContext.read.parquet("...") // in Scala
DataFrame people = sqlContext.read().parquet("...") // in Java
Once created, it can be manipulated using the various domain-specific-language (DSL) functions defined in: DataFrame (this class), Column, and functions.
To select a column from the data frame, use apply method in Scala and col in Java.
val ageCol = people("age") // in Scala
Column ageCol = people.col("age") // in Java
Note that the Column type can also be manipulated through its various functions.
// The following creates a new column that increases everybody's age by 10.
people("age") + 10 // in Scala
people.col("age").plus(10); // in Java
A more concrete example in Scala:
// To create DataFrame using SQLContext
val people = sqlContext.read.parquet("...")
val department = sqlContext.read.parquet("...")
people.filter("age > 30")
.join(department, people("deptId") === department("id"))
.groupBy(department("name"), "gender")
.agg(avg(people("salary")), max(people("age")))

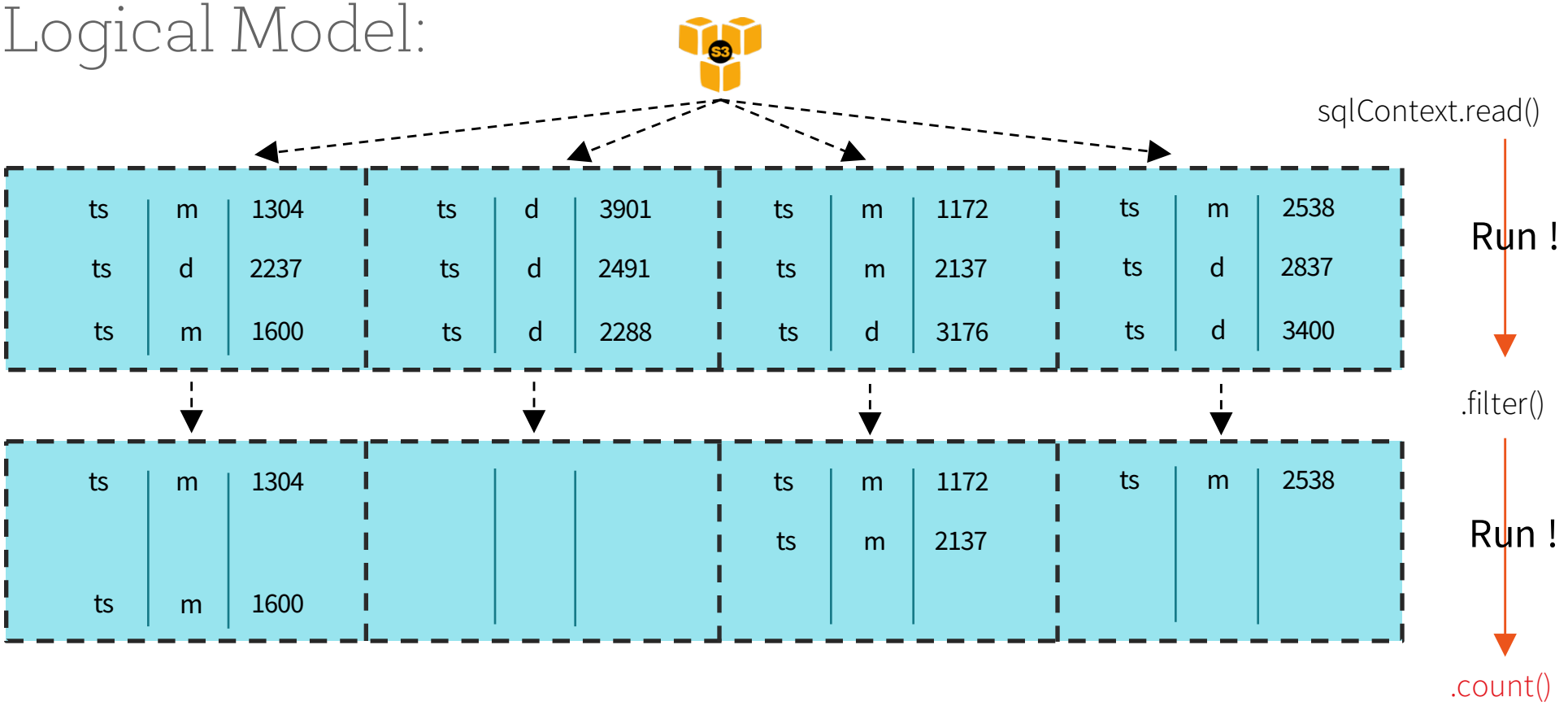
DataFrame: pageviewsDF



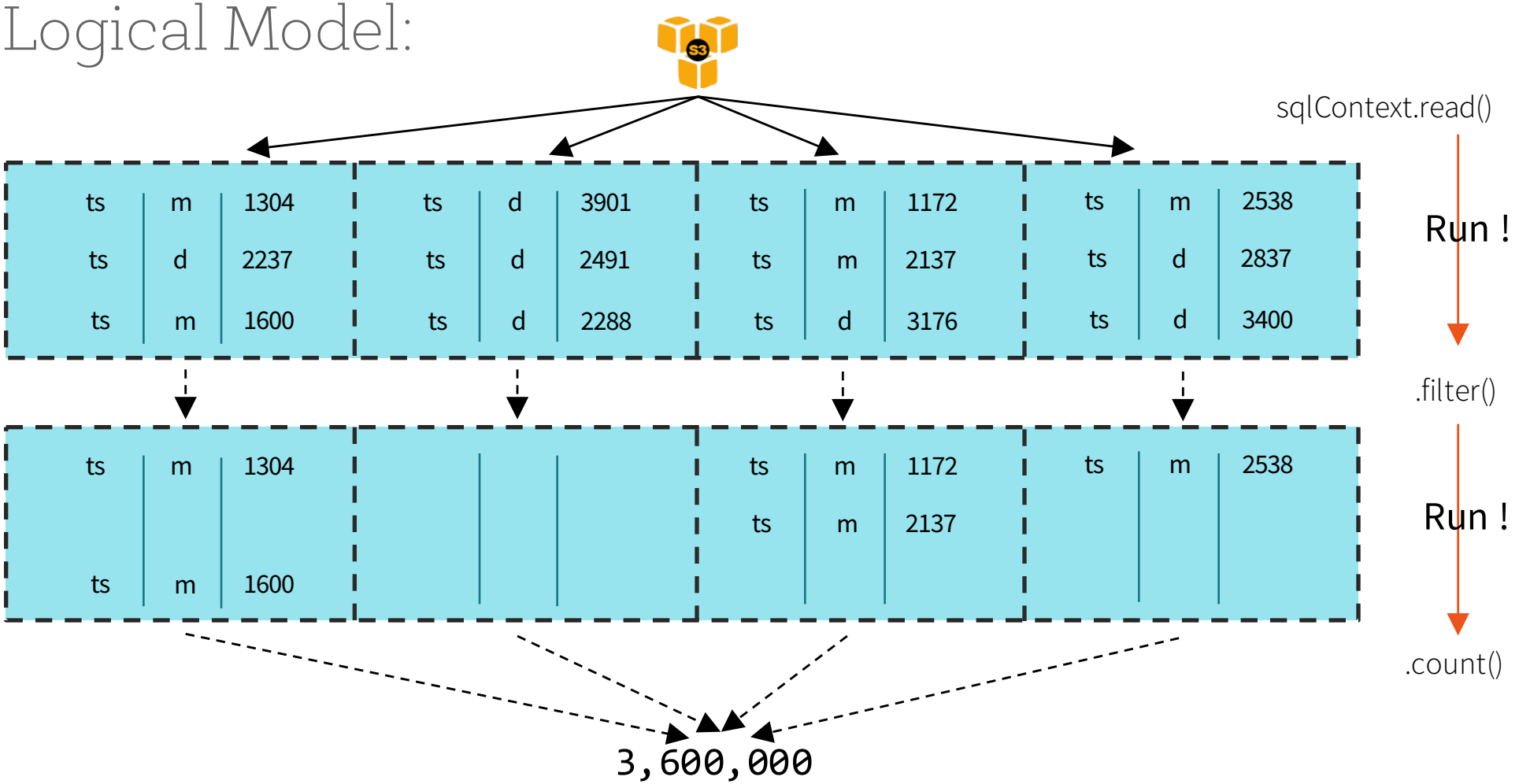
Logical Model:



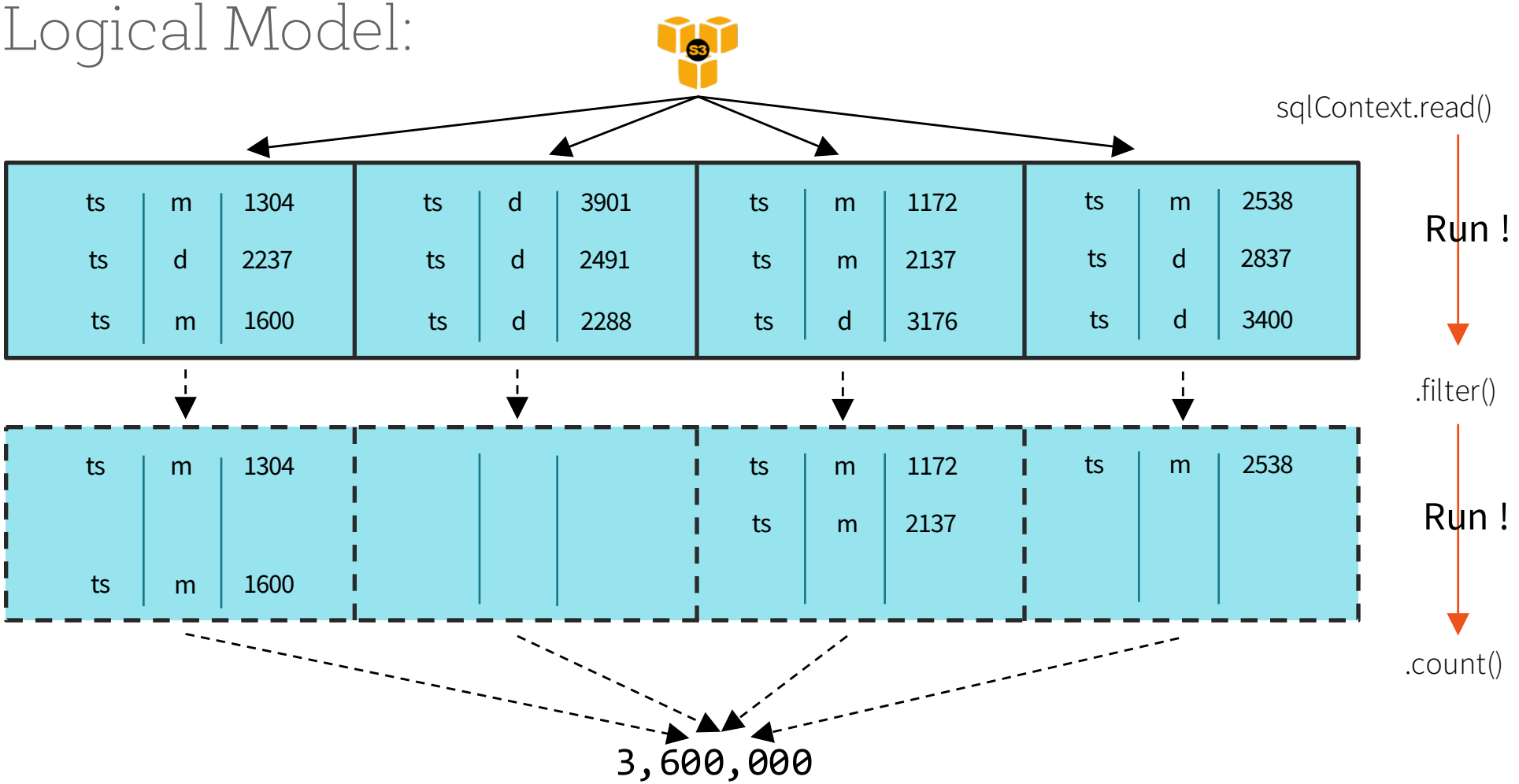
Logical Model:



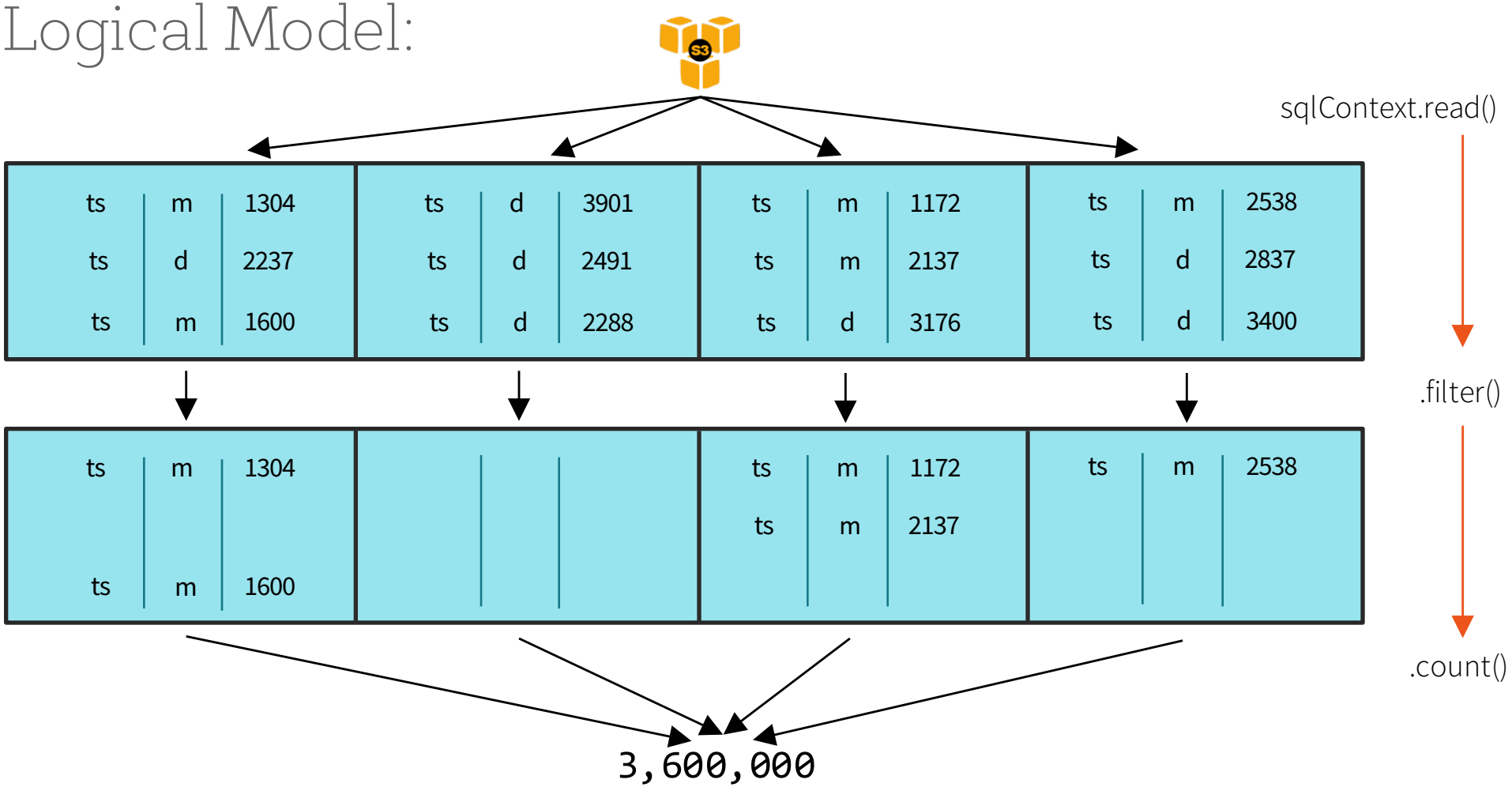
Logical Model:



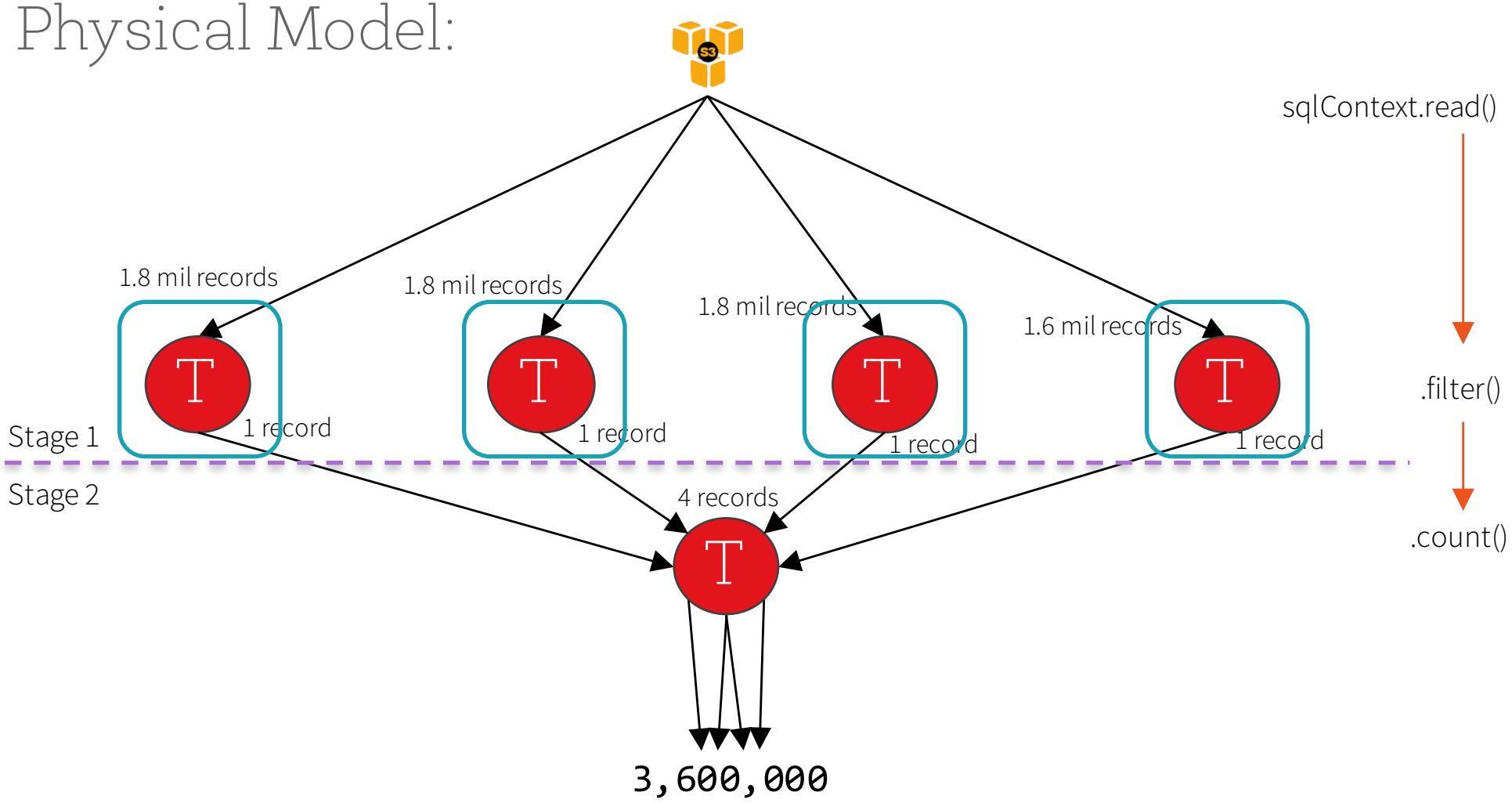
Logical Model:



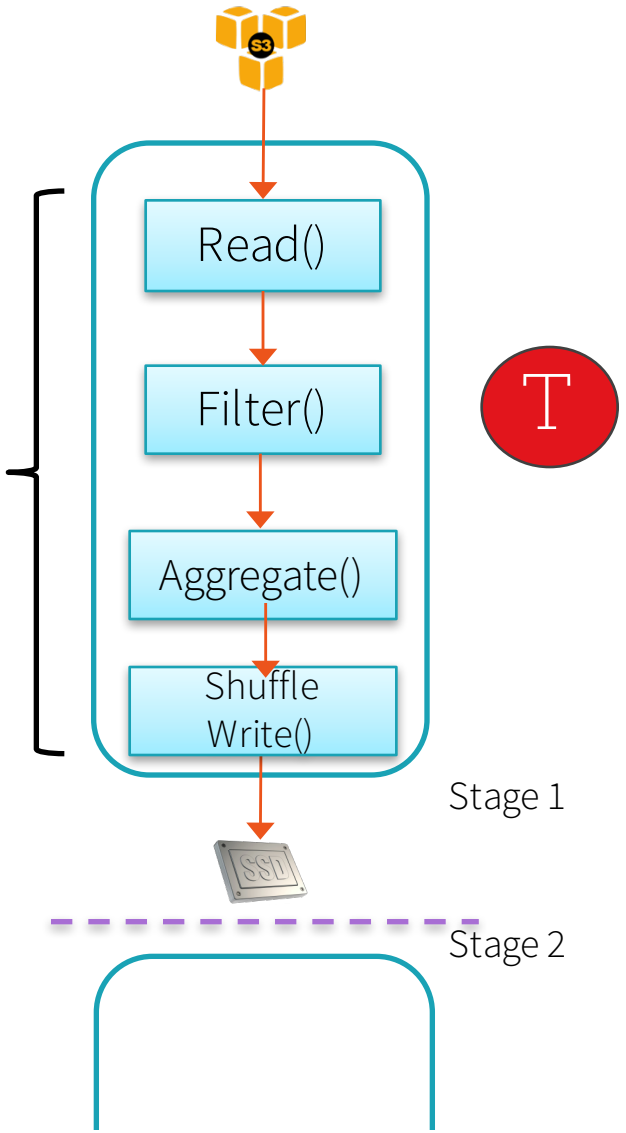
Logical Model:



Physical Model:



Pipelining



Physical Model:

Completed Stages (2)

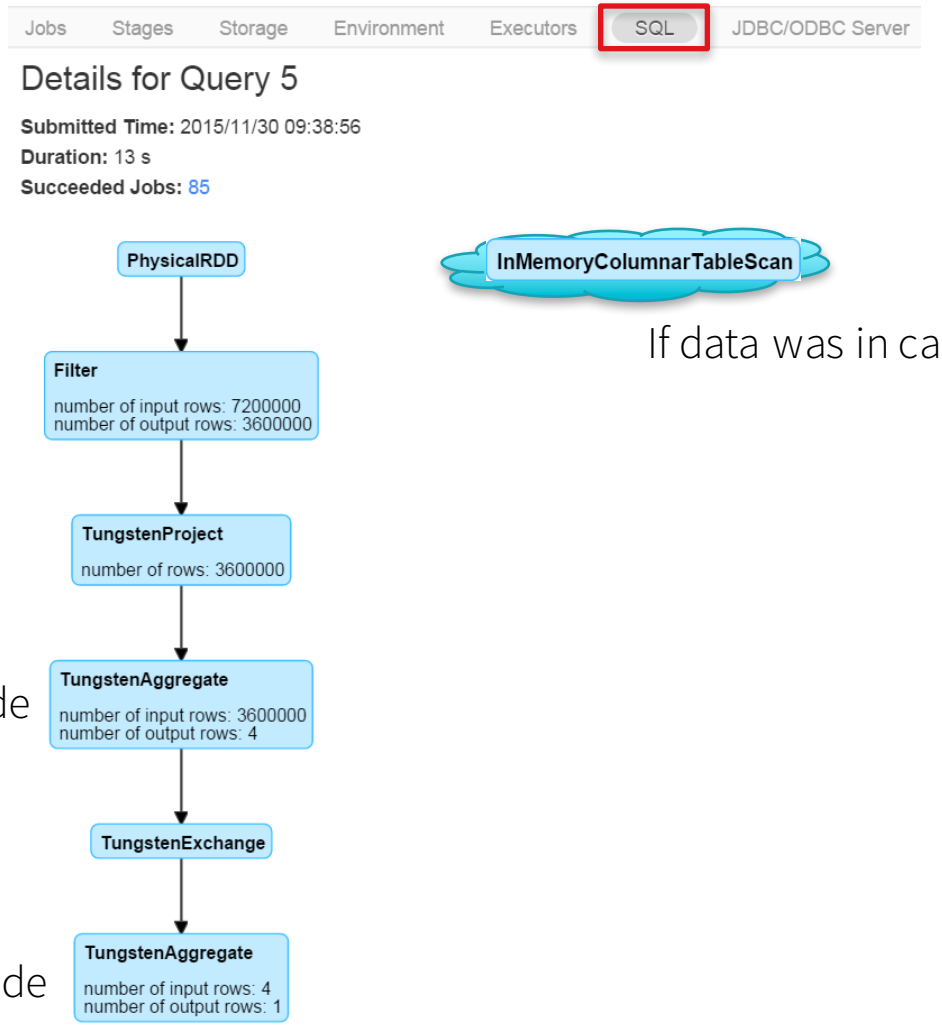
Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
114	1622753363940683362	pageviewsDF.filter($"site" === "mobile").count() count at <console>:27 +details	2015/11/30 09:39:09	7 ms	1/1			168.0 B	
113	1622753363940683362	pageviewsDF.filter($"site" === "mobile").count() count at <console>:27 +details	2015/11/30 09:38:56	12 s	4/4	250.0 MB			168.0 B

Physical Model:

(DAG Visualization / Job)

Physical Model:

(DAG Visualization / SQL)



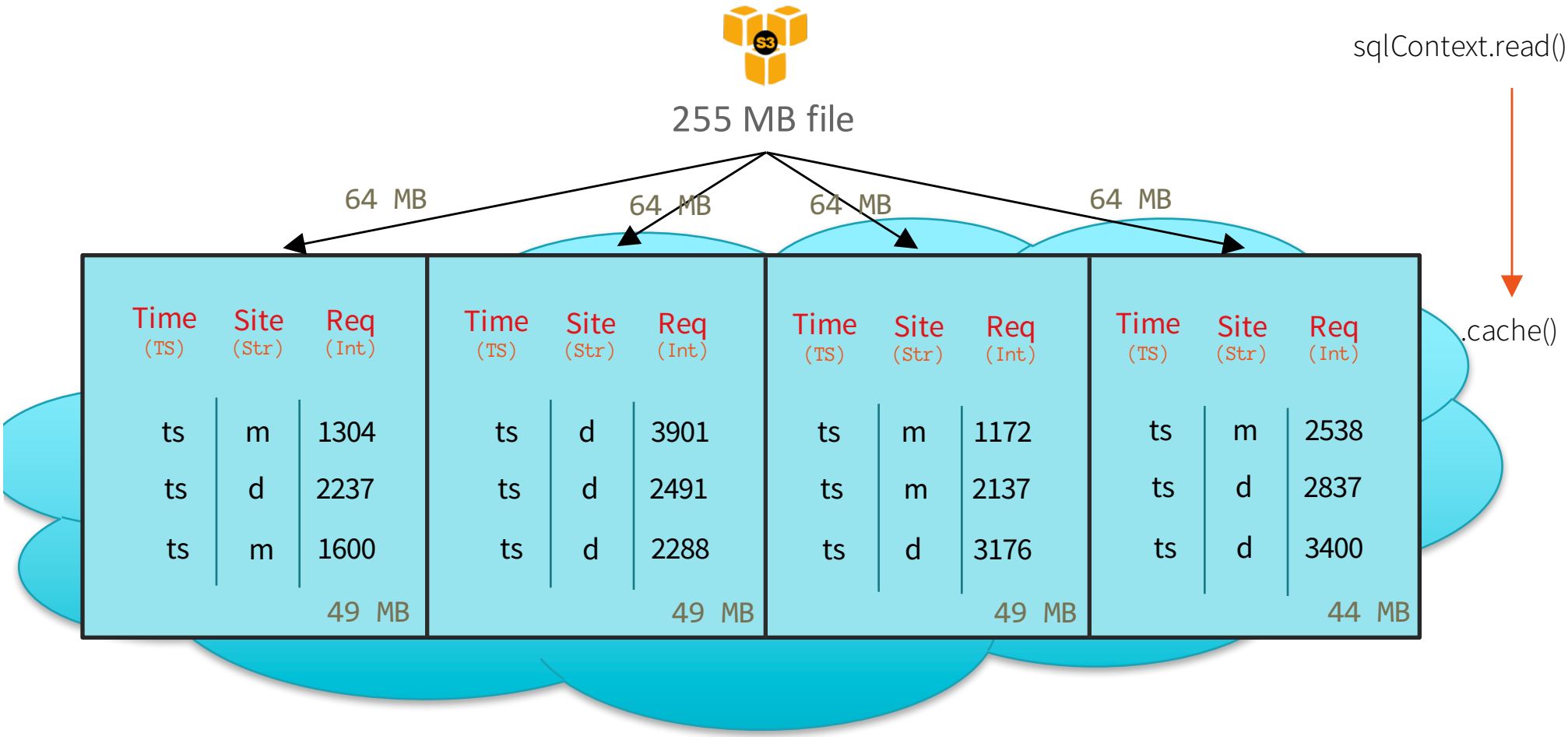
If data was in cache…

Partial aggregation per node

Shuffle!

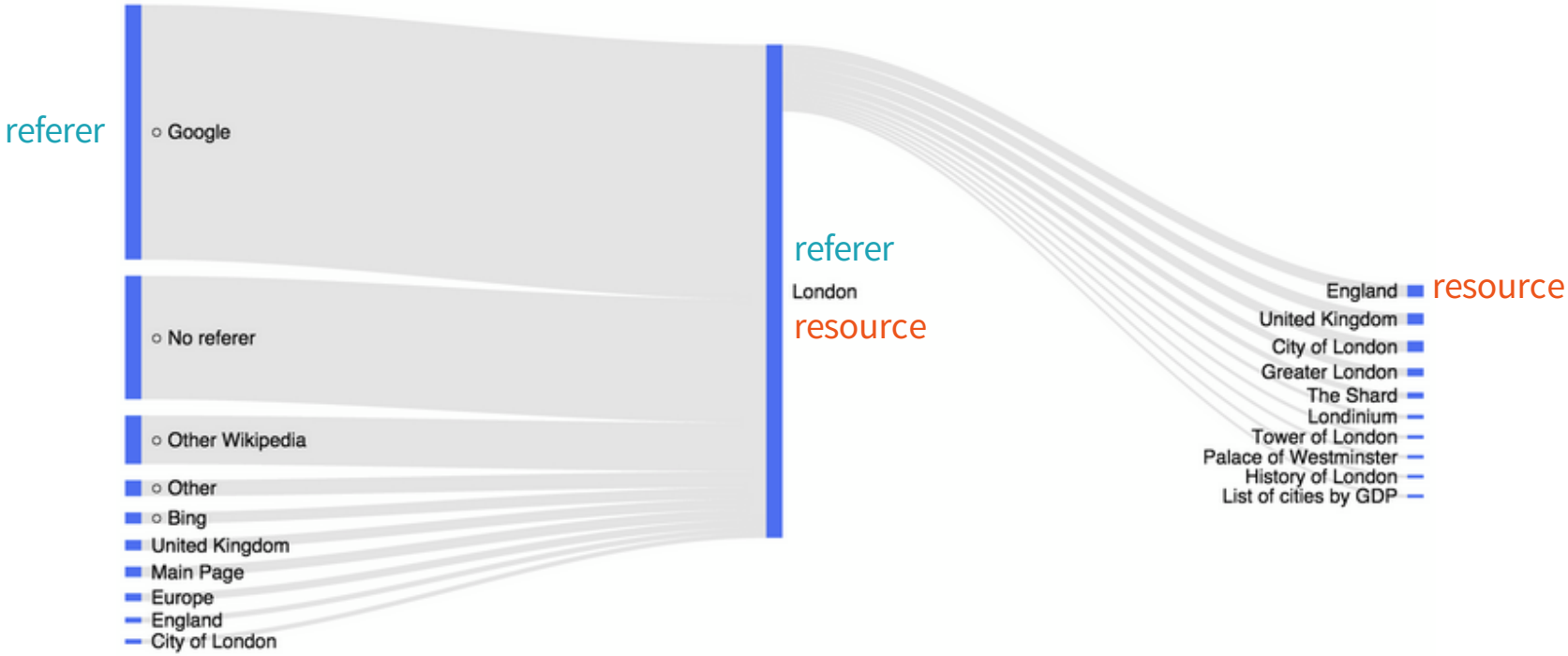
Partial aggregation per node

DataFrame: pageviewsDF



clickstream

(1.2 GB)



Wikipedia Clickstream

By Ellery Wulczyn
Data Scientist @ The Wikimedia Foundation

Clickstream Data: 1.2 GB tsv file

22 million

`(referer, resource)` pairs

out of 3.2 billion requests

in February 2015.

(only includes requests for articles in the desktop version of English Wikipedia)

```
> display(clickstreamDF)
```

(referer, resource)

prev_id	curr_id	n	prev_title	curr_title	type
	3632887	121	other-google	!!	other
	3632887	93	other-wikipedia	!!	other
	3632887	46	other-empty	!!	other
	3632887	10	other-other	!!	other
64486	3632887	11	!_(disambiguation)	!!	other
2061699	2556962	19	Louden_Up_Now	!!!_(album)	link
	2556962	25	other-empty	!!!_(album)	other
	2556962	16	other-google	!!!_(album)	other
	2556962	44	other-wikipedia	!!!_(album)	other

Showing the first 1000 rows.

February 2015

22 million records *(out of 3.2 billion requests)*


(referer)
article in the main namespace (ns) of Eng Wikipedia
article title
any Wikipedia page not in main Eng ns
other-wikipedia
an empty referer
other-empty
a page from any other Wikimedia project
other-internal
Google
other-google
Yahoo
other-yahoo
Bing
other-bing
Facebook
other-facebook
Twitter
other-twitter
anything else
other-other

prev_id	curr_id	n	prev_title	curr_title	type
	3632887	121	other-google	!!	other
	3632887	93	other-wikipedia	!!	other
	3632887	46	other-empty	!!	other

n: the number of occurrences of the `(referer, resource)` pair

prev_id	curr_id	n	prev_title	curr_title	type
	3632887	121	other-google	!!	other
	3632887	93	other-wikipedia	!!	other
	3632887	46	other-empty	!!	other

link: if the referer and request are both articles and the referer links to the request

redlink: links to a Wikipedia page that do not exist *(uncreated pages)*

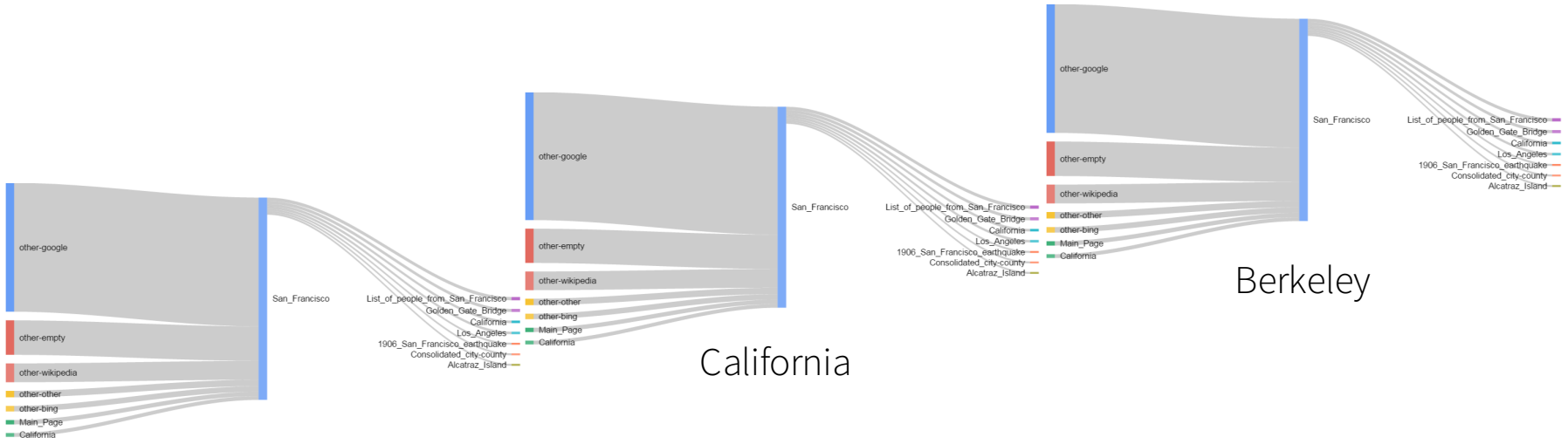
other: if the referer + request are both articles but the referer does not link to the request.
(this can happen when clients search or spoof their refer)


Wikipedia Clickstream - Fe
datahub.io/dataset/wikipedia-clickstream/resource/be85cc68-d1e6-4134-804a-fd36b94dbb82
datahub
The easy way to get, use and share data
Datasets
Organizations
About
Blog
Help
Search
/ Organizations / Wikimedia / Wikipedia Clickstream / February 2015 English ...
Go to resource
February 2015 English Wikipedia Clickstream
URL: http://files.figshare.com/1927919/2015_02_clickstream.tsv.gz
Data Preparation
The data contains counts of (referer, resource) pairs extracted from the request logs of English Wikipedia. When a client requests a resource by following a link or performing a search, the URI of the webpage that linked to the resource is included with the request in an HTTP header called the "referer". This data captures 22 million (referer, resource) pairs from a total of 3.2 billion requests collected during the month of February 2015.
The dataset only includes requests for articles in the main namespace of the desktop version of English Wikipedia.
Referers were mapped to a fixed set of values corresponding to internal traffic or external traffic from one of the top 5 global traffic sources to English Wikipedia, based on this scheme:
- an article in the main namespace of English Wikipedia -> the article title
- any Wikipedia page that is not in the main namespace of English Wikipedia -> `other-wikipedia`
- an empty referer -> `other-empty`
- a page from any other Wikimedia project -> `other-internal`
- Google -> `other-google`
- Yahoo -> `other-yahoo`
- Bing -> `other-bing`
- Facebook -> `other-facebook`
- Twitter -> `other-twitter`
- anything else -> `other-other`

What does the traffic inflow vs outflow look like for the San Diego article?

Riding the Clickstream:

Apache Spark?



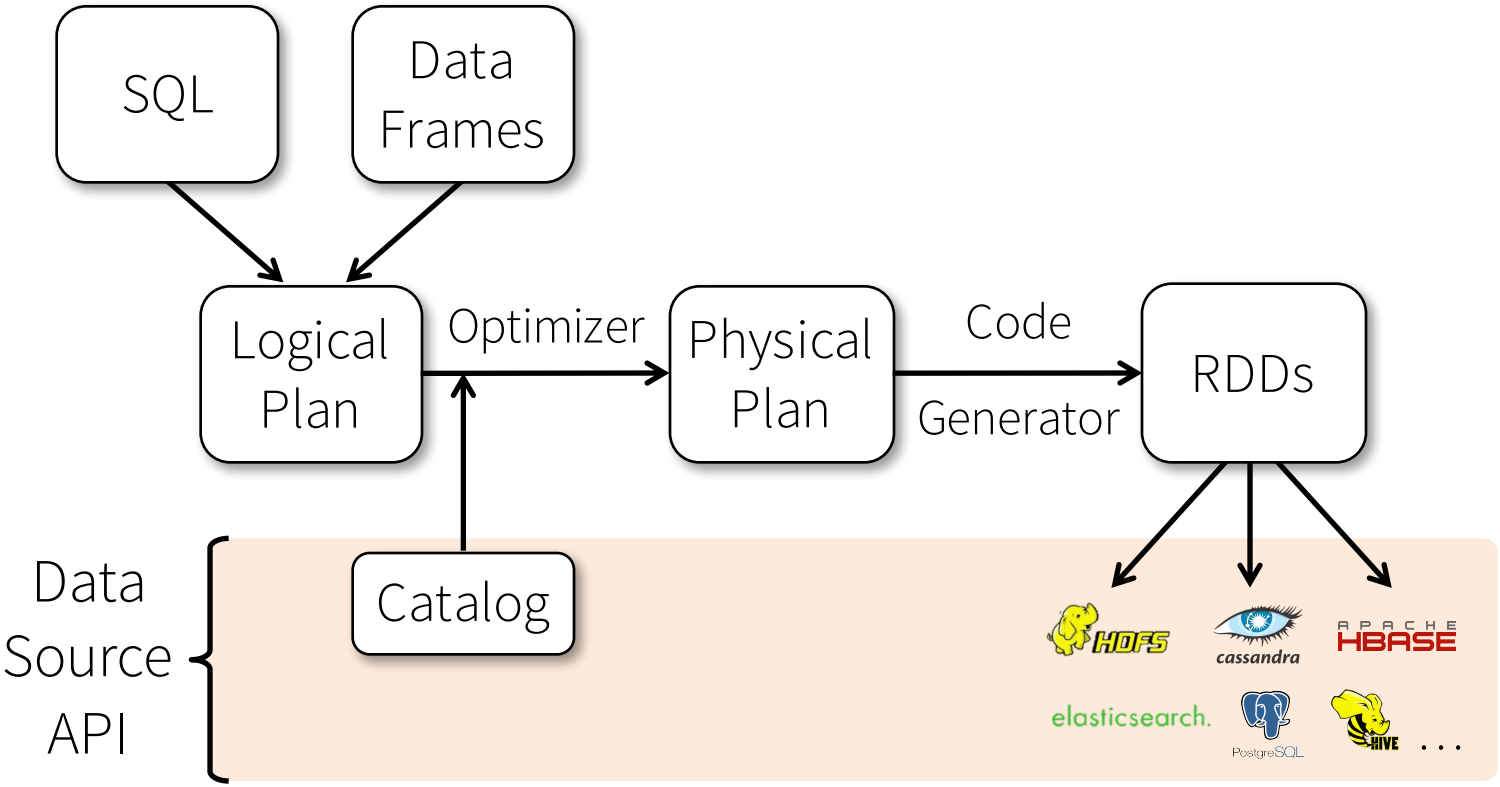
San Diego

California

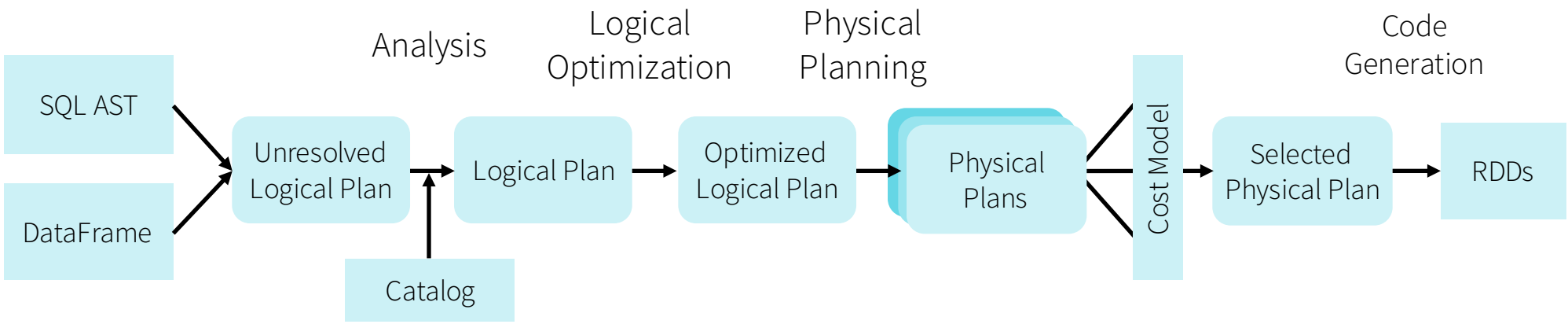
Berkeley


Spark
Catalyst
databrickstraining

Spark SQL Architecture

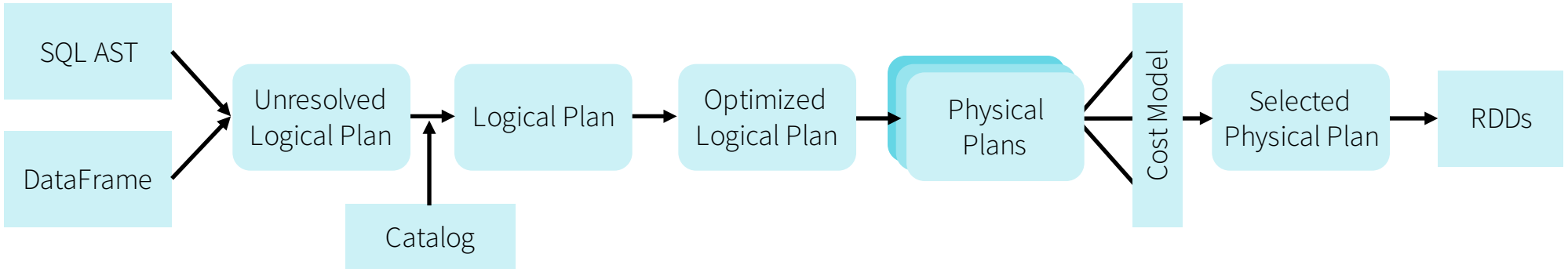


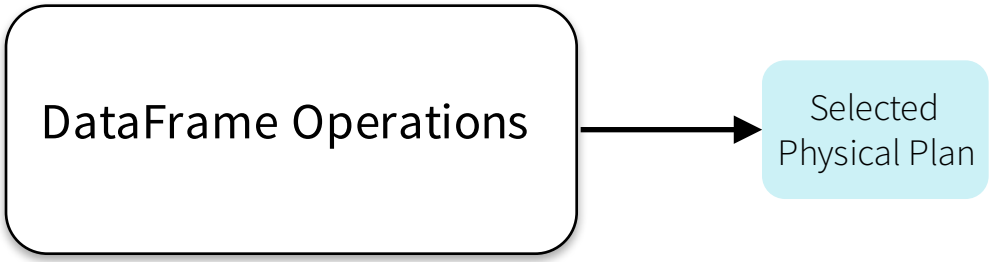
DataFrames + SQL: Under the hood



DataFrames and SQL share the same optimization/execution pipeline

DataFrames: Under the hood

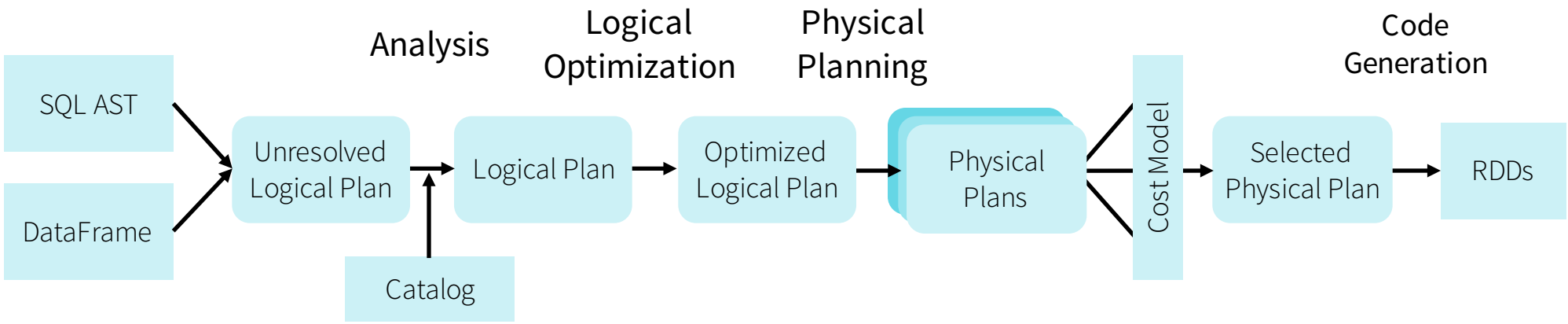




Details

```
== Parsed Logical Plan ==
Aggregate [count(1) AS count#25L]
 Filter (site#11 = mobile)
  Subquery pageviews_by_second
   Relation[timestamp#10,site#11,requests#12] CsvRelation(/FileStore/tables/rv3hb79r1449478143619,true, ,",null,#,PERMISSIVE,COMMONS,false,false,StructTyp
e(StructField(timestamp,StringType,true), StructField(site,StringType,true), StructField(requests,IntegerType,true)),UTF-8,false)

== Analyzed Logical Plan ==
count: bigint
Aggregate [count(1) AS count#25L]
 Filter (site#11 = mobile)
  Subquery pageviews_by_second
   Relation[timestamp#10,site#11,requests#12] CsvRelation(/FileStore/tables/rv3hb79r1449478143619,true, ,",null,#,PERMISSIVE,COMMONS,false,false,StructTyp
e(StructField(timestamp,StringType,true), StructField(site,StringType,true), StructField(requests,IntegerType,true)),UTF-8,false)

== Optimized Logical Plan ==
Aggregate [count(1) AS count#25L]
 Project
  Filter (site#11 = mobile)
   Relation[timestamp#10,site#11,requests#12] CsvRelation(/FileStore/tables/rv3hb79r1449478143619,true, ,",null,#,PERMISSIVE,COMMONS,false,false,StructTyp
e(StructField(timestamp,StringType,true), StructField(site,StringType,true), StructField(requests,IntegerType,true)),UTF-8,false)

== Physical Plan ==
TungstenAggregate(key=[], functions=[(count(1),mode=Final,isDistinct=false)], output=[count#25L])
 TungstenExchange SinglePartition
  TungstenAggregate(key=[], functions=[(count(1),mode=Partial,isDistinct=false)], output=[currentCount#28L])
   TungstenProject
    Filter (site#11 = mobile)
     Scan CsvRelation(/FileStore/tables/rv3hb79r1449478143619,true,     ,",null,#,PERMISSIVE,COMMONS,false,false,StructType(StructField(timestamp,StringTy
pe,true), StructField(site,StringType,true), StructField(requests,IntegerType,true)),UTF-8,false)[timestamp#10,site#11,requests#12]

Code Generation: true
```

Using Catalyst in Spark SQL



Analysis: analyzing a logical plan to resolve references

Logical Optimization: logical plan optimization

Physical Planning: Physical planning

Code Generation: Compile parts of the query to Java bytecode

Catalyst Optimizations

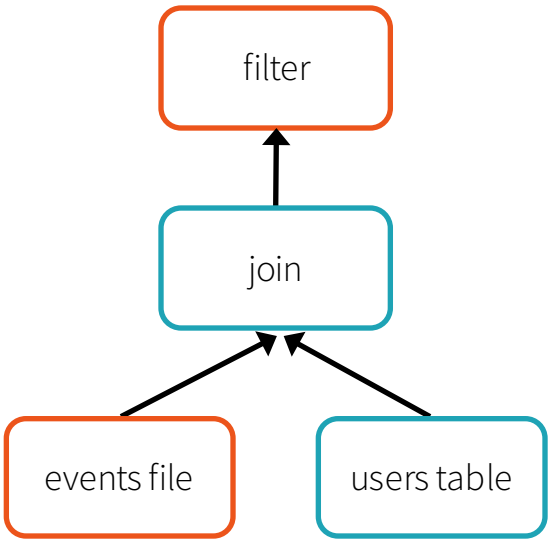
Logical Optimizations

- Push filter predicates down to data source, so irrelevant data can be skipped
- **Parquet:** skip entire blocks, turn comparisons on strings into cheaper integer comparisons via dictionary encoding
- **RDBMS:** reduce amount of data traffic by pushing predicates down

Create Physical Plan & generate JVM bytecode

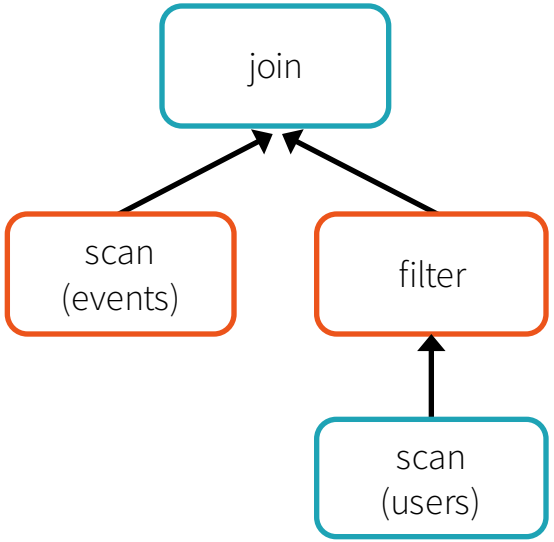
- Catalyst compiles operations into physical plans for execution and generates JVM bytecode
- Intelligently choose between broadcast joins and shuffle joins to reduce network traffic
- **Lower level optimizations:** eliminate expensive object allocations and reduce virtual function calls

Logical Plan



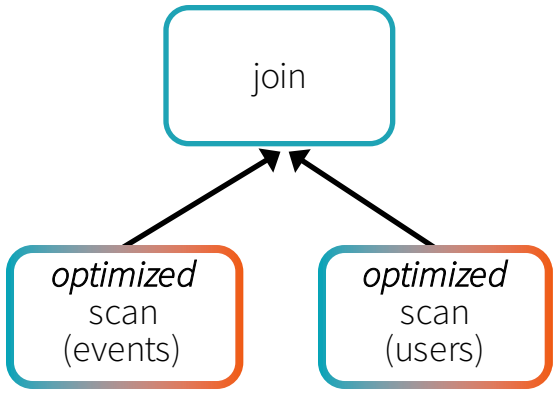
{JSON}

Physical Plan



{JSON}

Optimized Physical Plan
with Predicate Pushdown
and Column Pruning



Parquet

COMPANY
All Posts
Partners
Events
Press Releases

DEVELOPER
All Posts
Spark
Spark SQL
Spark Streaming
MLlib
Spark Summit

Search Blog

Subscribe

Deep Dive into Spark SQL's Catalyst Optimizer

April 13, 2015 | by Michael Armbrust, Yin Huai, Cheng Liang, Reynold Xin and Matei Zaharia

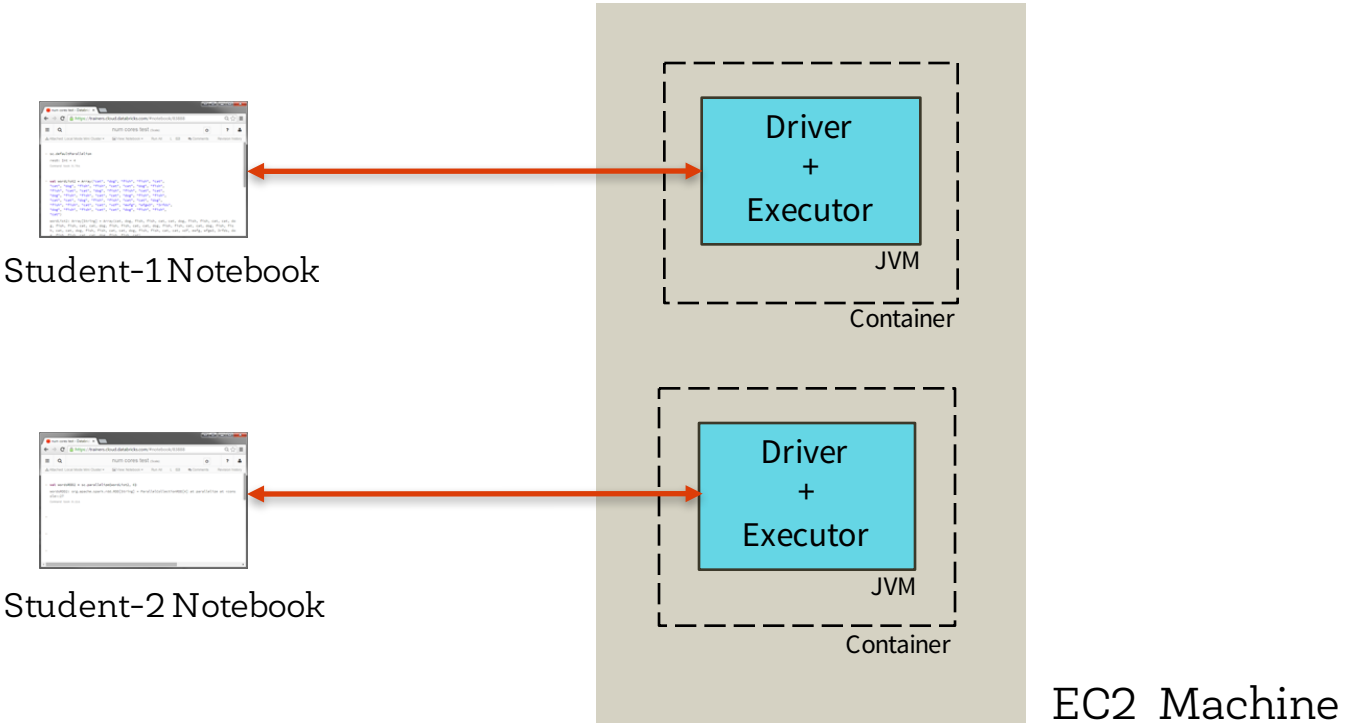
Spark SQL is one of the newest and most technically involved components of Spark. It powers both SQL queries and the new DataFrame API. At the core of Spark SQL is the Catalyst optimizer, which leverages advanced programming language features (e.g. Scala's pattern matching and quasiquotes) in a novel way to build an extensible query optimizer.

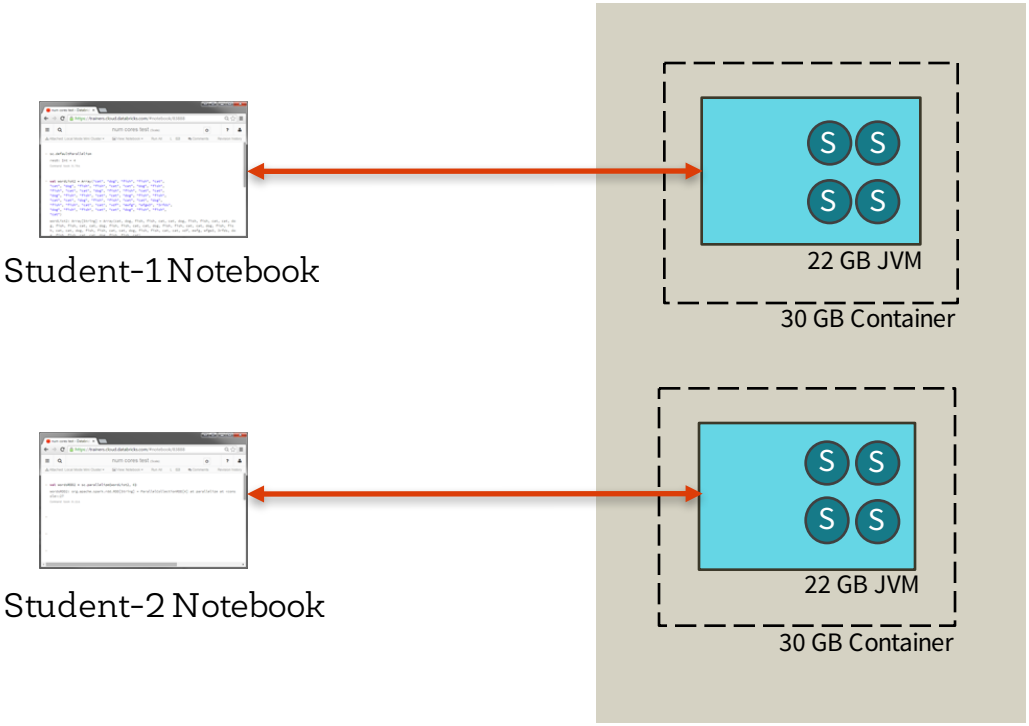
We recently published a paper on Spark SQL that will appear in SIGMOD 2015 (co-authored with Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, and Ali Ghodsi). In this blog post we are republishing a section in the paper that explains the internals of the Catalyst optimizer for broader consumption.

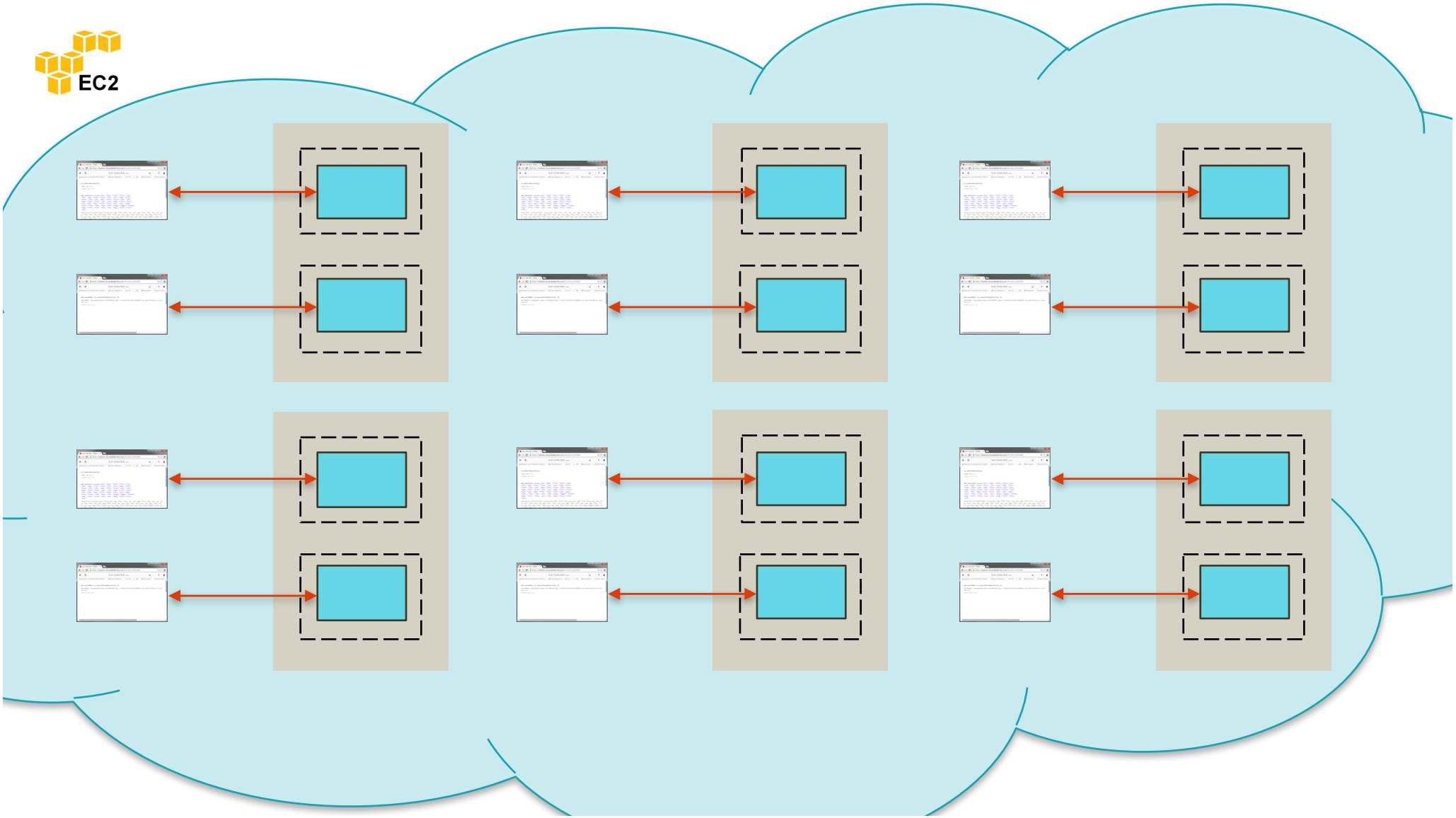
To implement Spark SQL, we designed a new extensible optimizer, Catalyst, based on functional programming constructs in Scala. Catalyst's extensible design had two purposes. First, we wanted to make it easy to add new optimization techniques and features to Spark SQL, especially for the purpose of tackling various problems we were seeing with big data (e.g., semistructured data and advanced analytics). Second, we wanted to enable external developers to extend the optimizer — for example, by adding data source specific rules that can push filtering or aggregation into external storage systems, or support for new data types. Catalyst supports both rule-based and cost-based optimization.

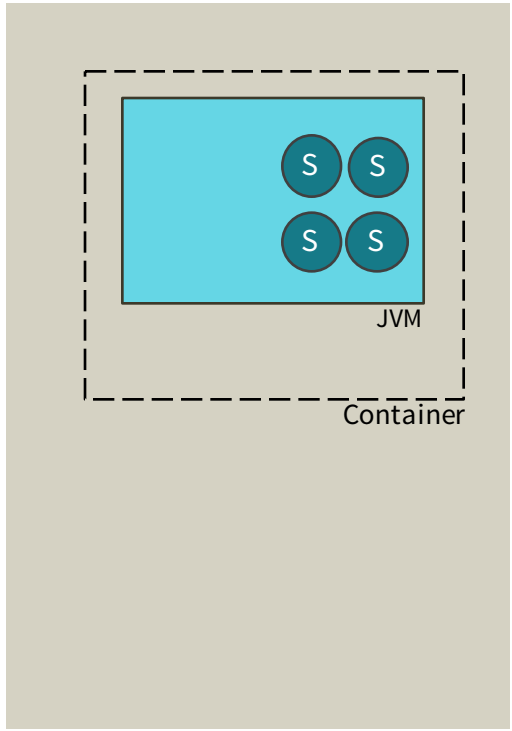
Architecture

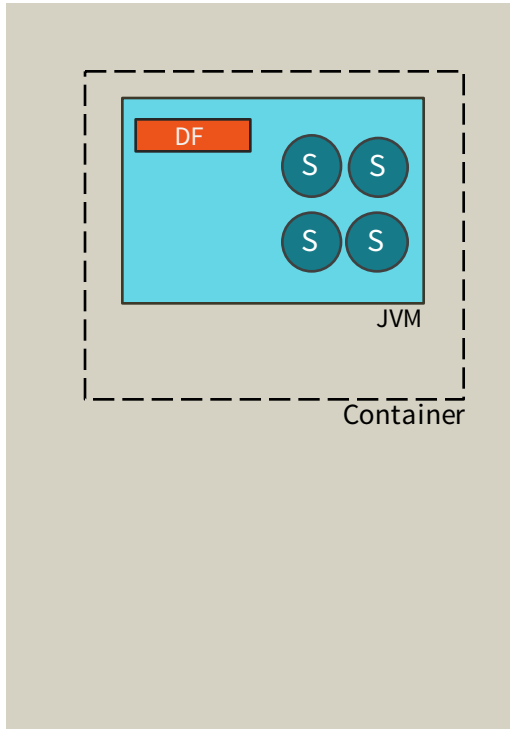
(local mode)

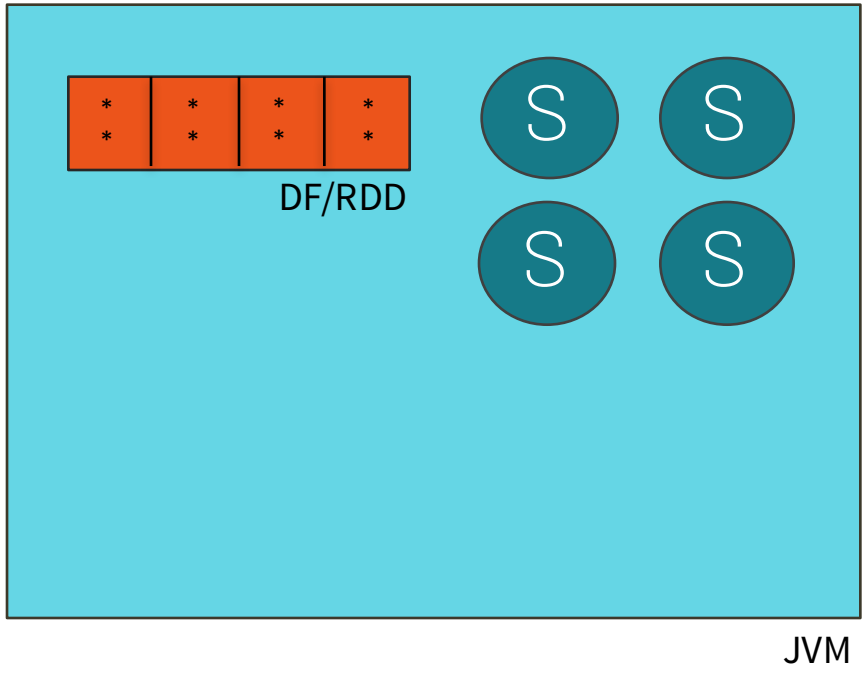

Student-1 Notebook
Student-2 Notebook
Driver
+
Executor
JVM
Container
Driver
+
Executor
JVM
Container
EC2 Machine

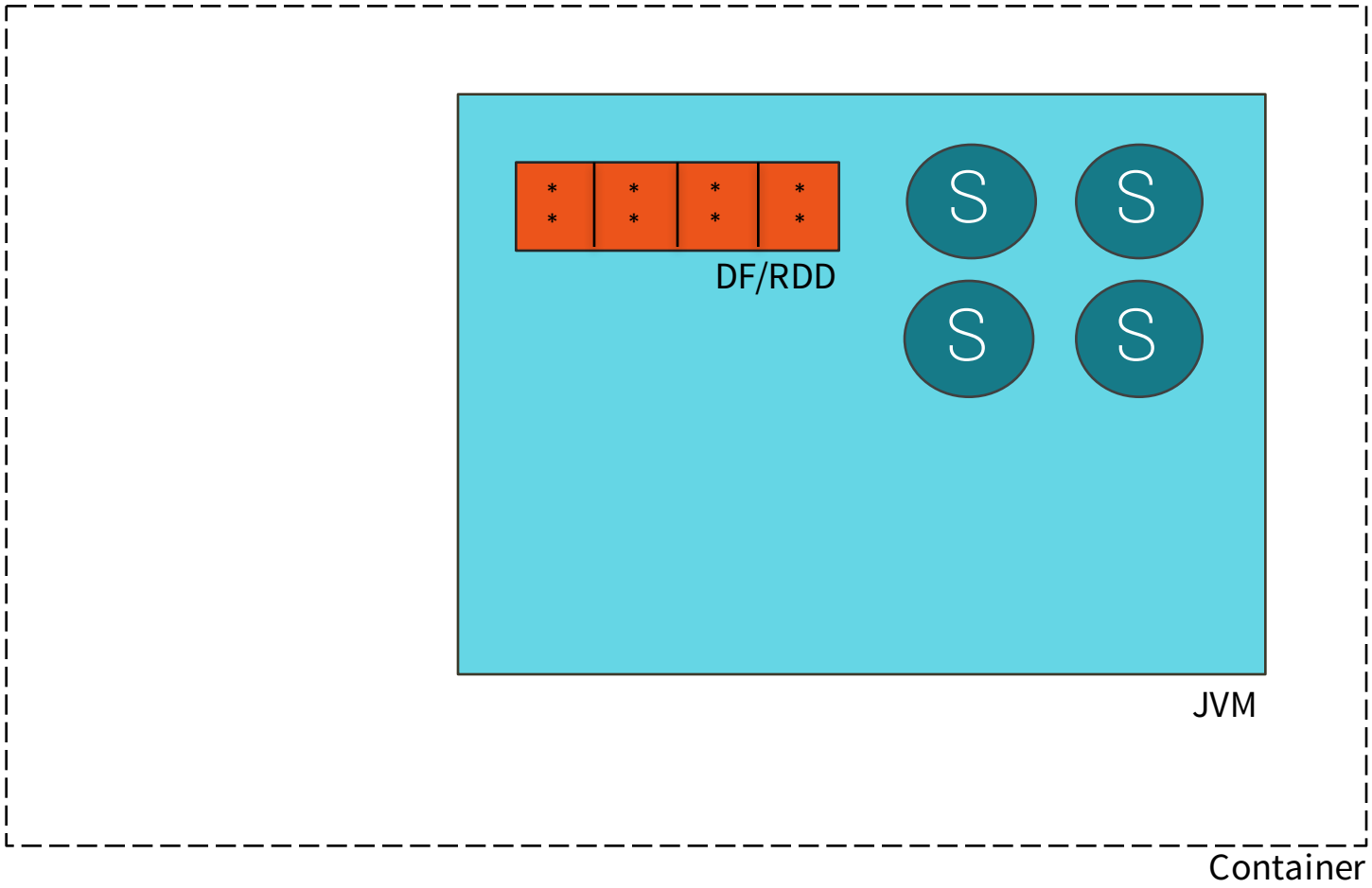

Student-1 Notebook
Student-2 Notebook
S
S
S
S
22 GB JVM
30 GB Container
S
S
S
S
22 GB JVM
30 GB Container
EC2 Machine
r3.2xlarge

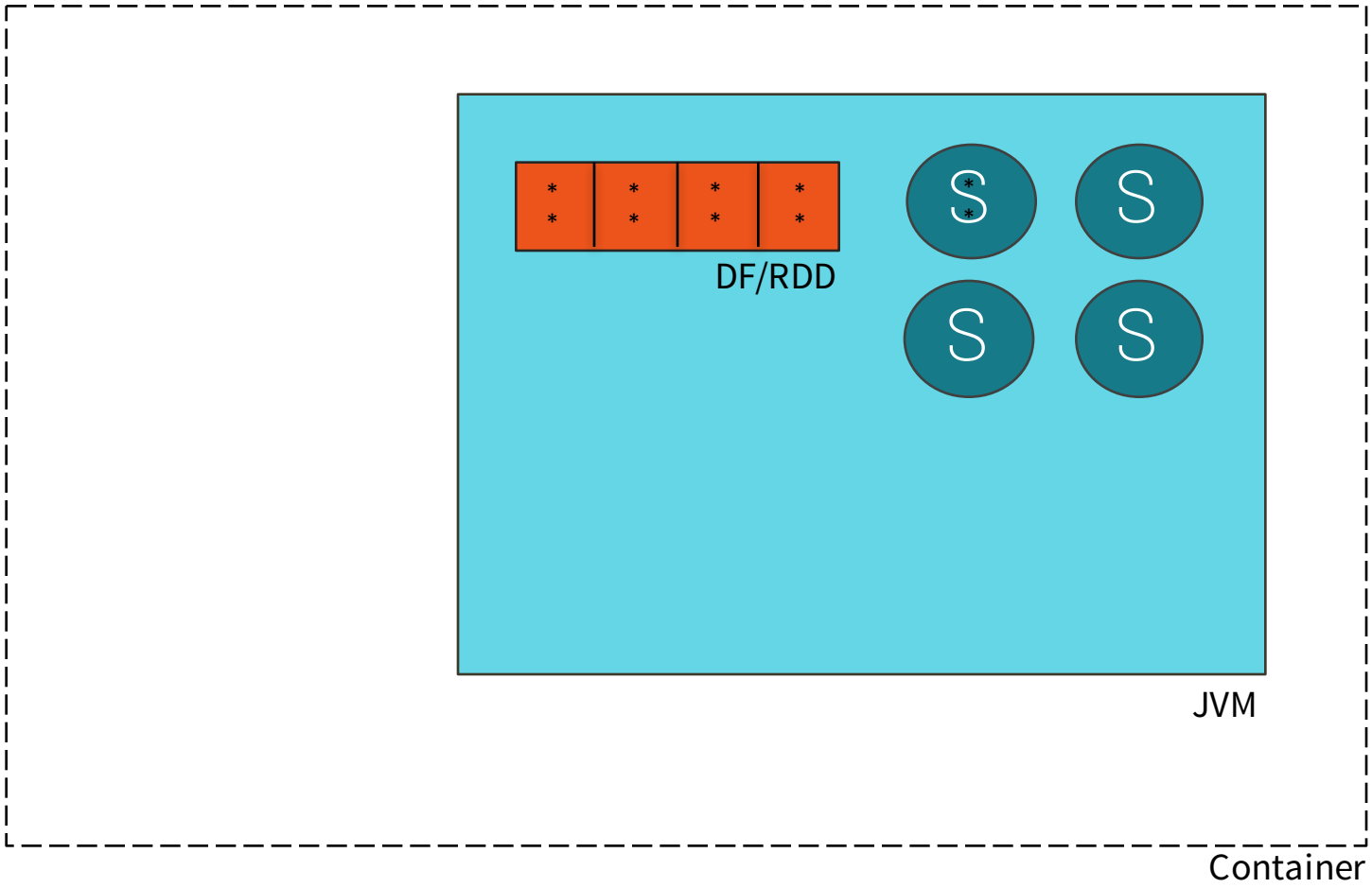

EC2

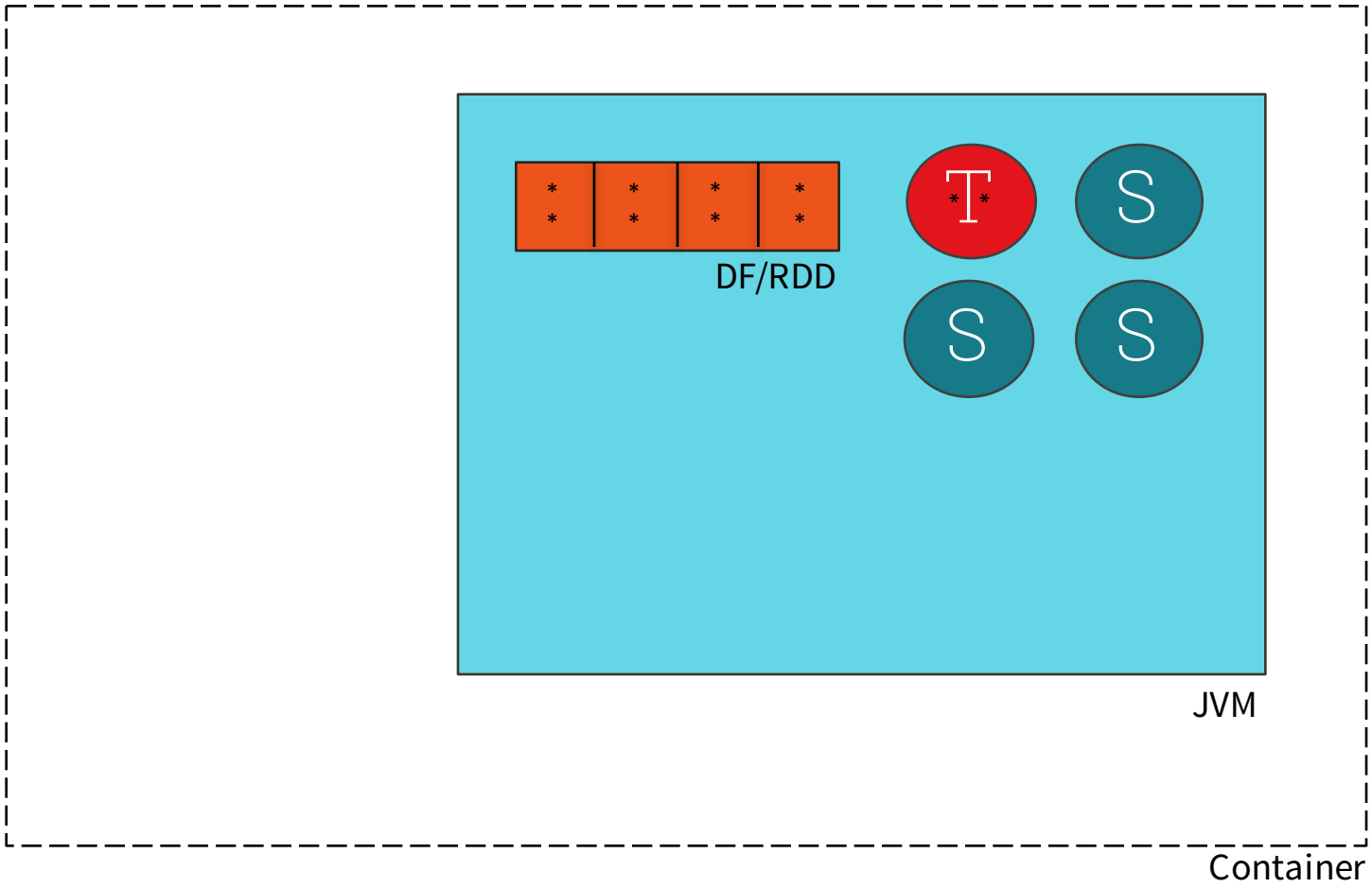

S
S
S
S
JVM
Container
EC2 Machine

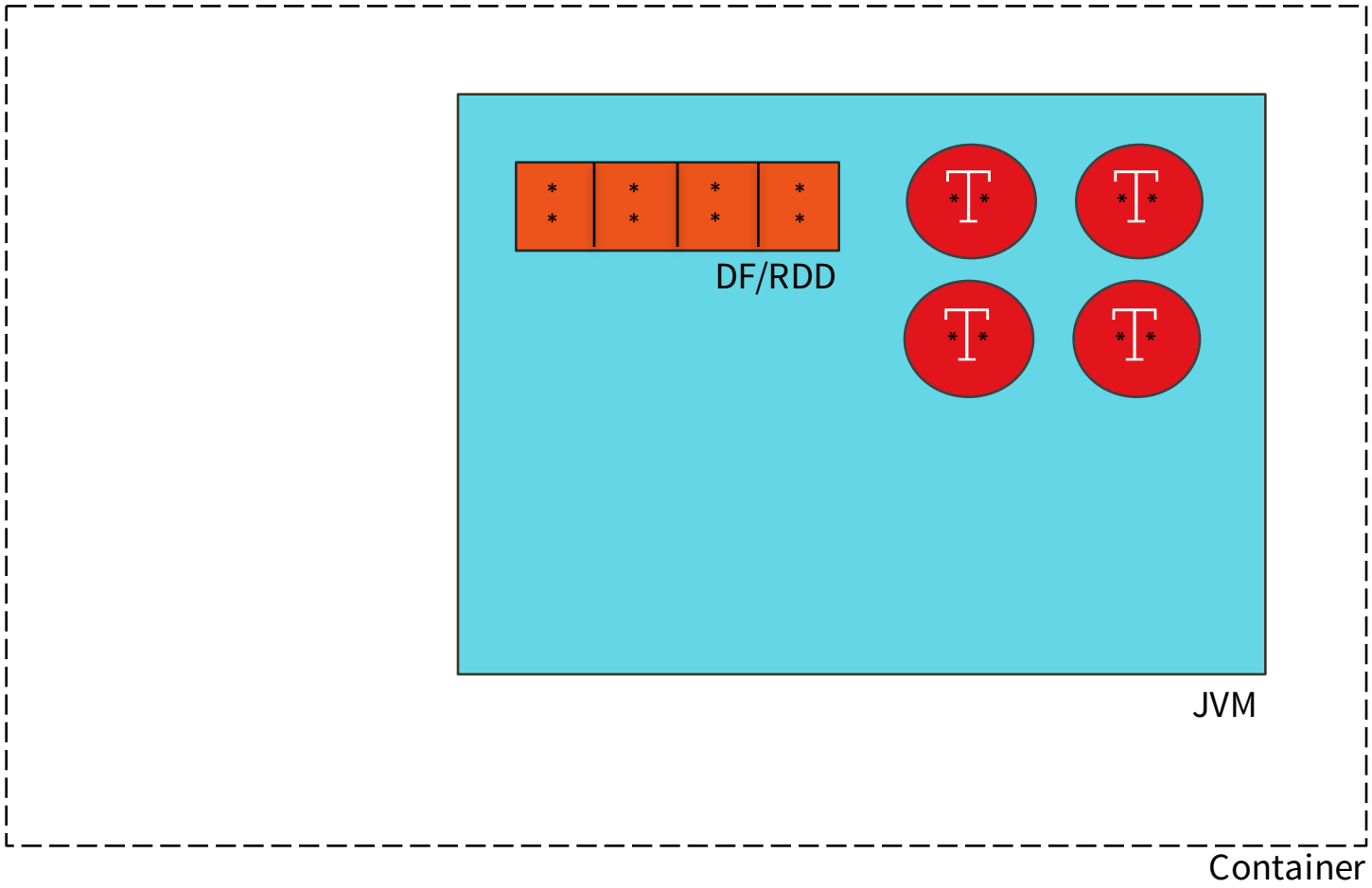

DF
S
S
S
S
JVM
Container
EC2 Machine

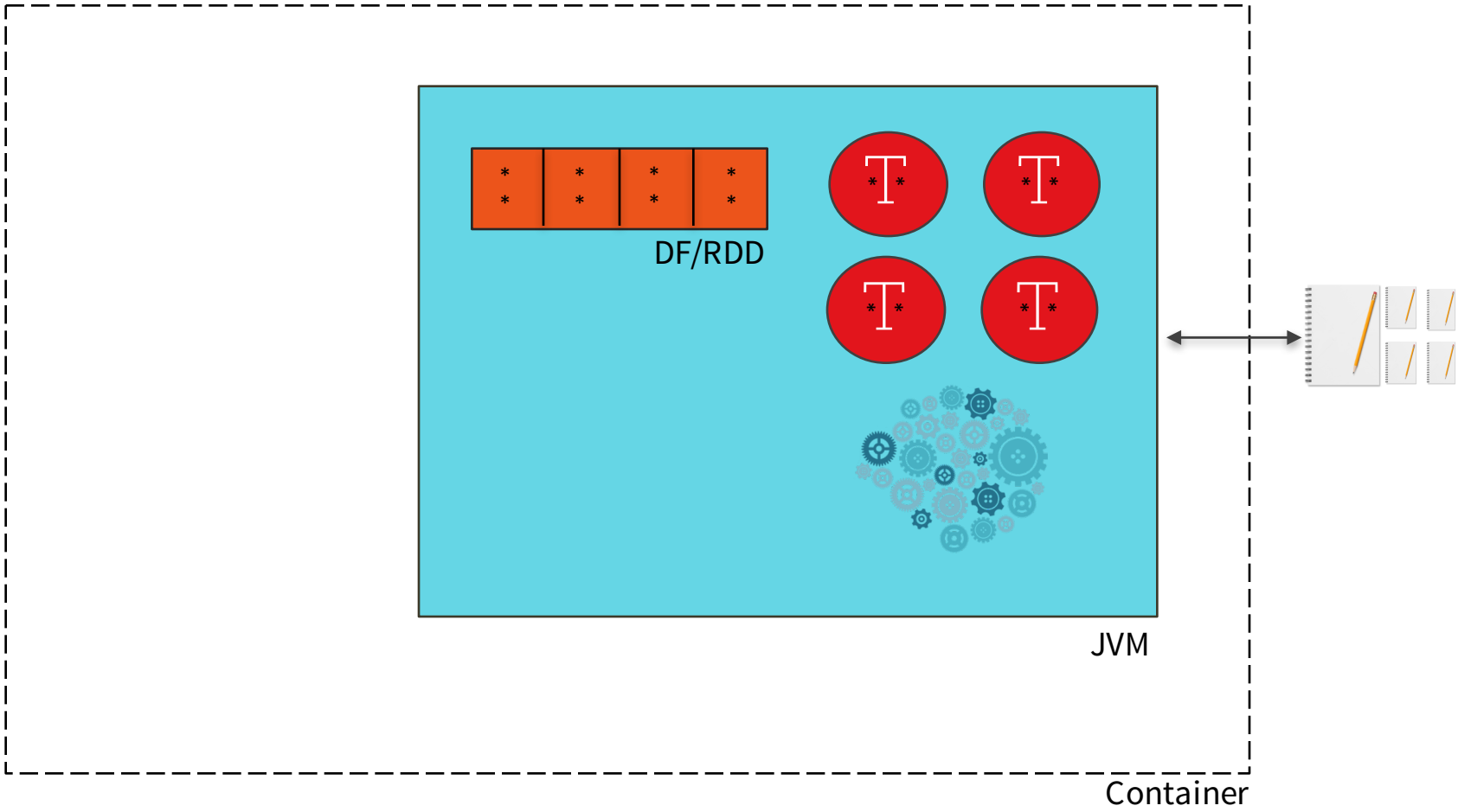

DF/RDD
S
S
S
S
JVM

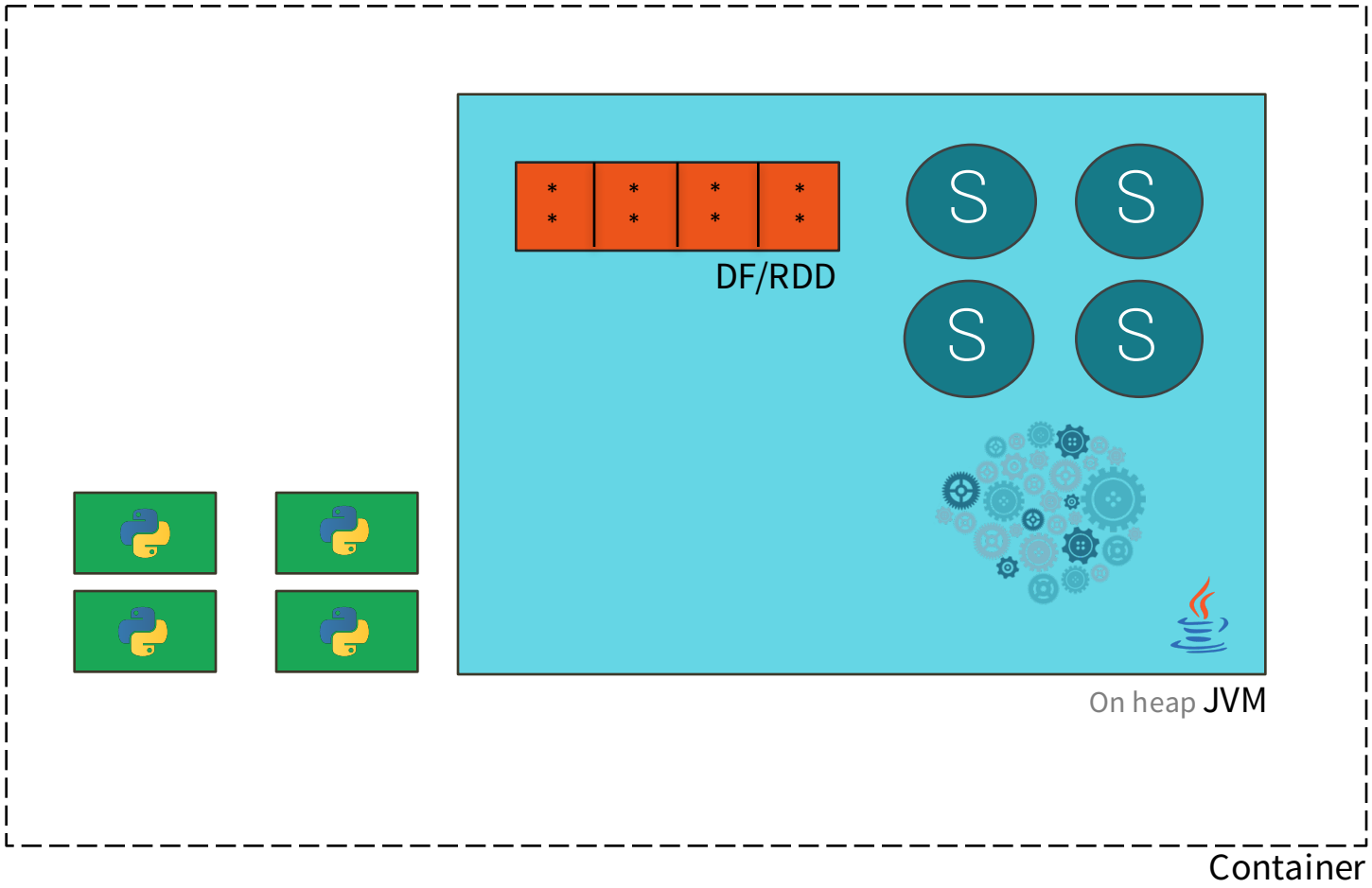

DF/RDD
S
S
S
S
JVM
Container

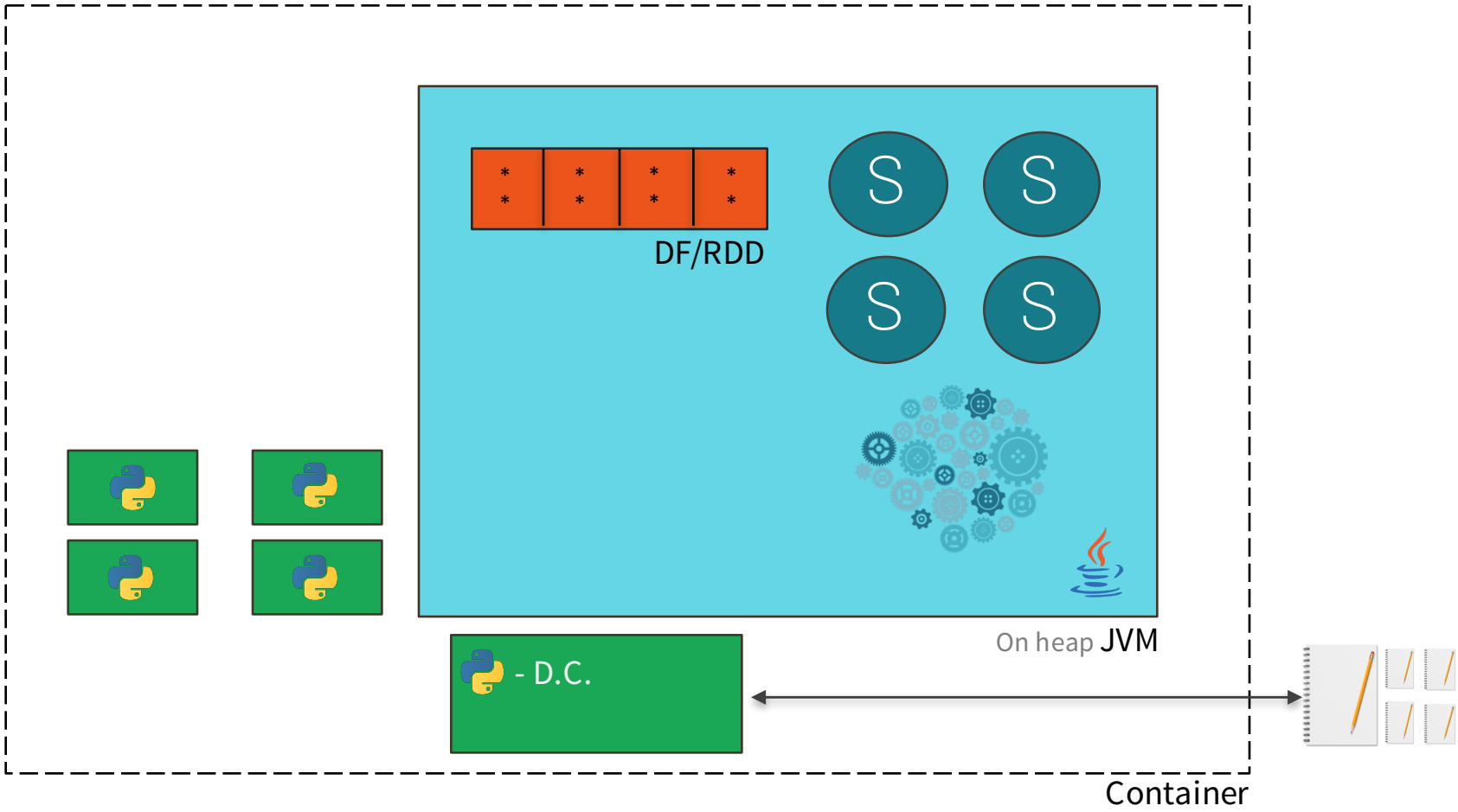

DF/RDD
S
S
S
S
JVM
Container

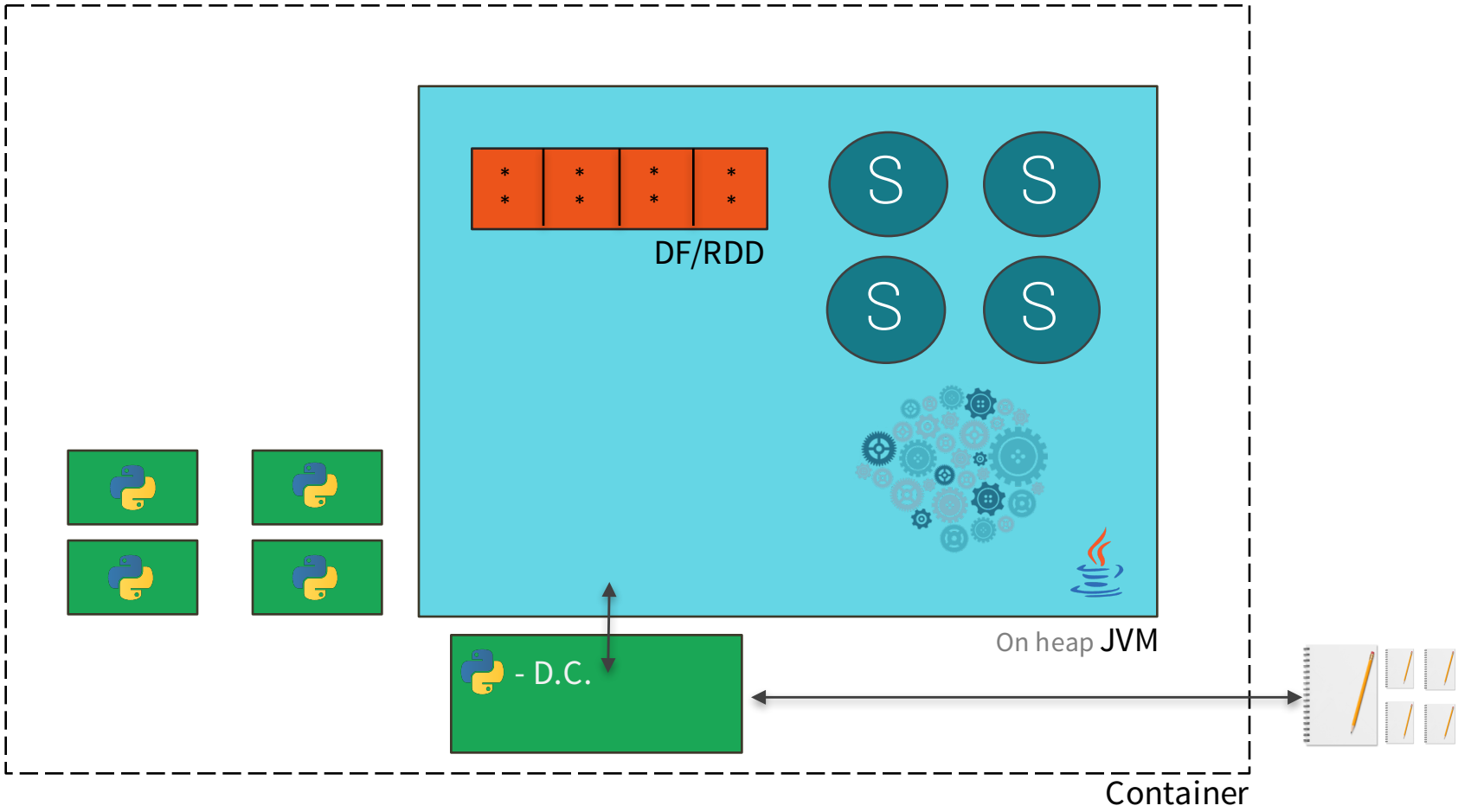

DF/RDD
T
S
S
S
JVM
Container


DF/RDD
JVM
Container


DF/RDD
JVM
Container

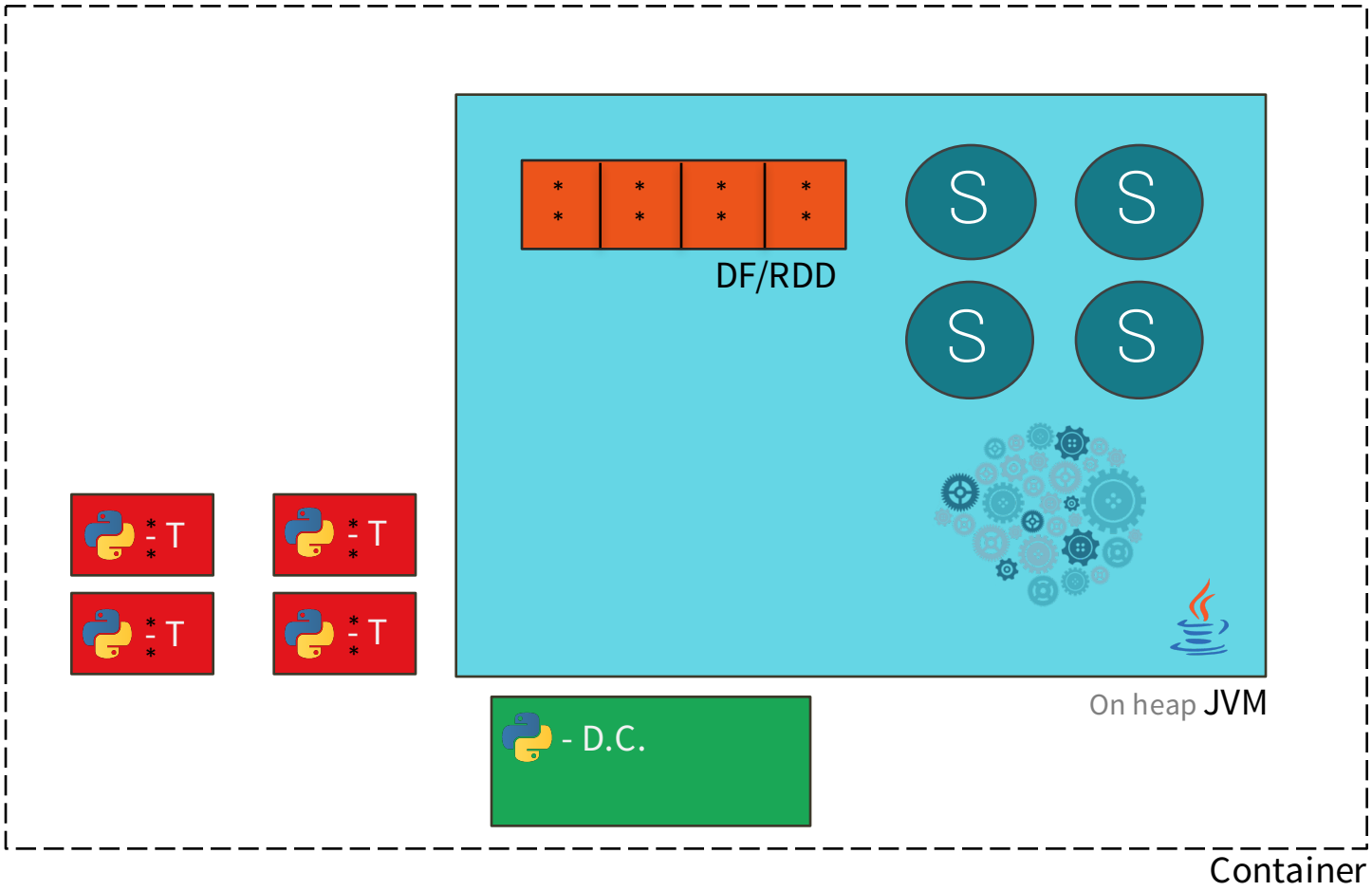

DF/RDD
S
S
S
S
On heap JVM
Container


DF/RDD
S
S
S
S
On heap JVM
- D.C.
Container

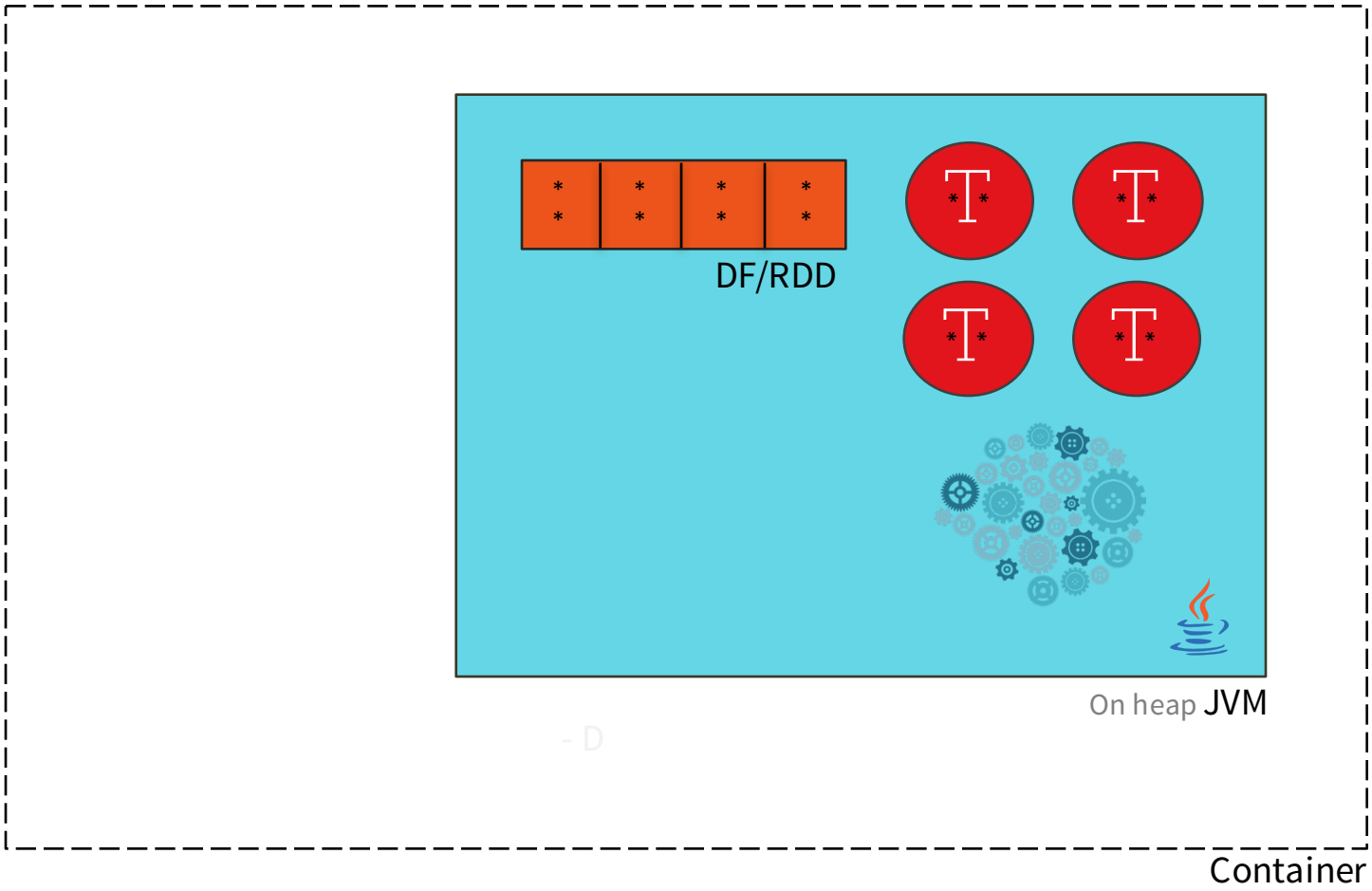

DF/RDD
S
S
S
S
On heap JVM
- D.C.
Container

Processing Python UDFs

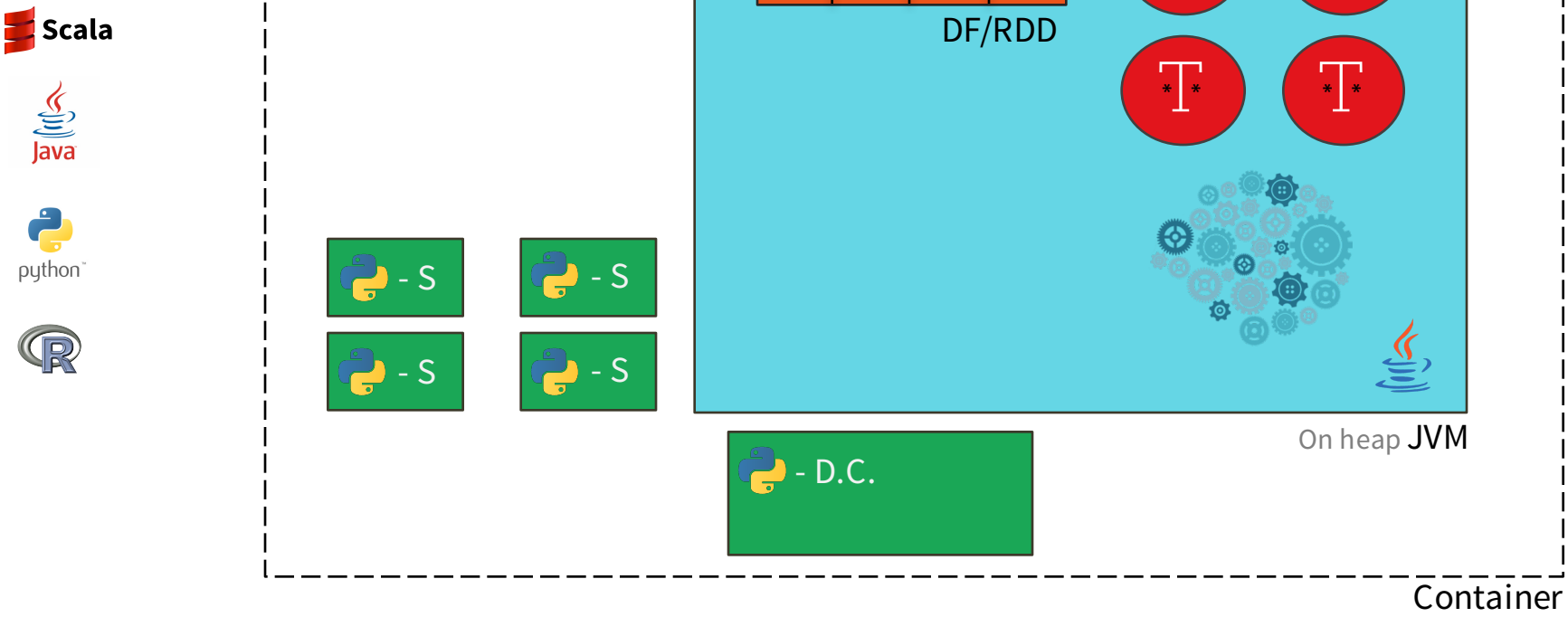
(slower)



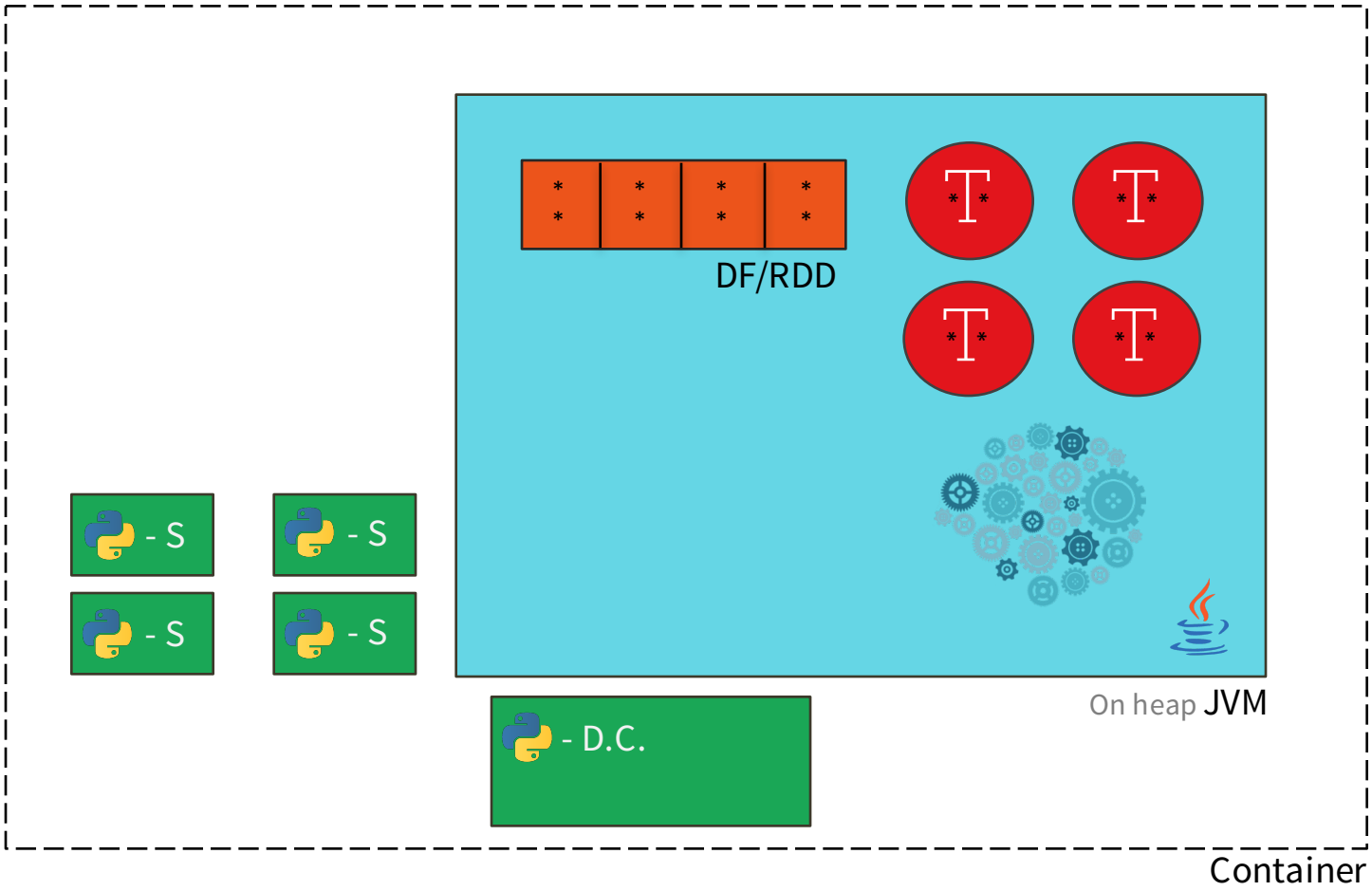
Processing
Scala / Java



Processing DataFrames

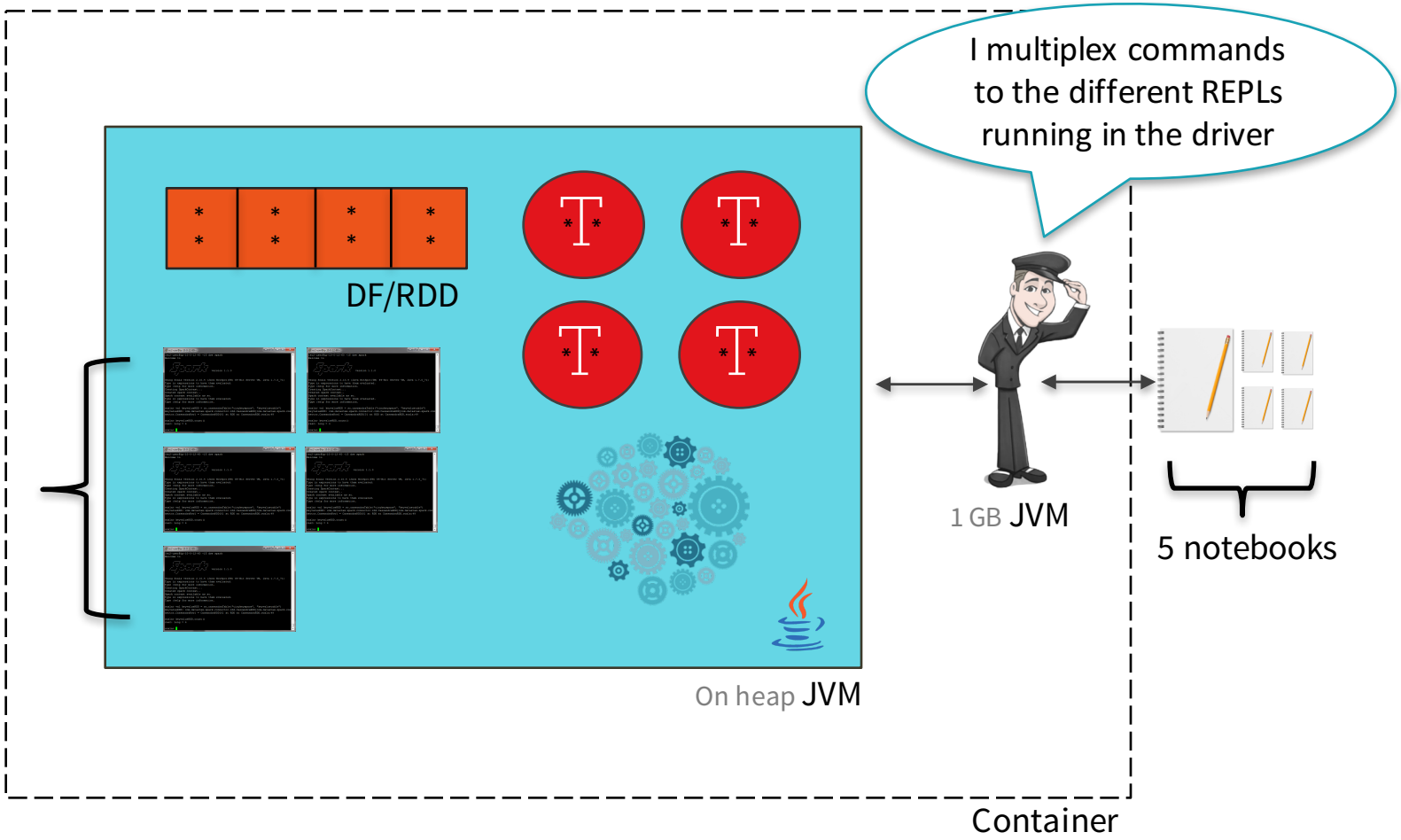


Processing
MLlib



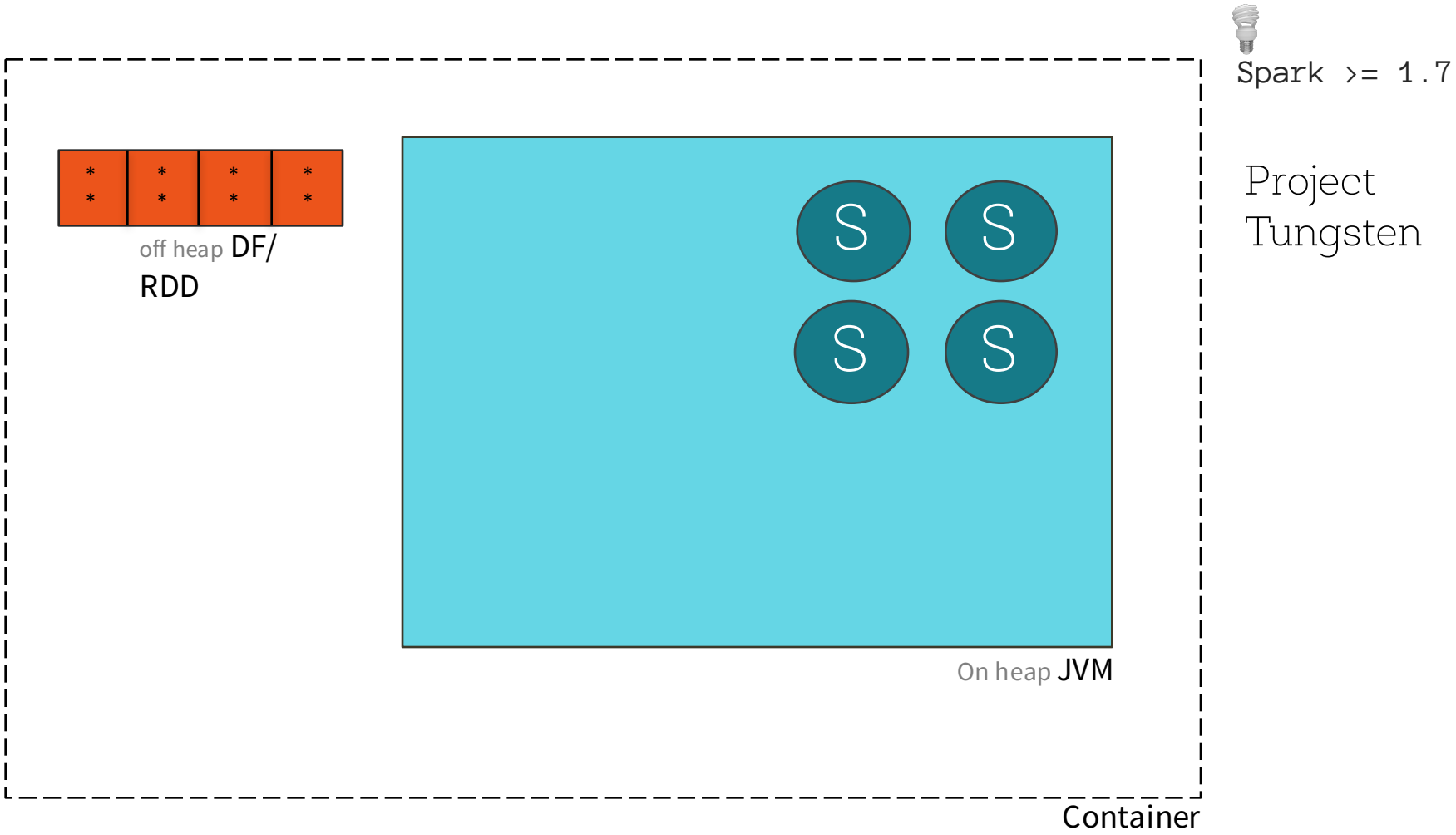
Chauffeur

"one last trick"

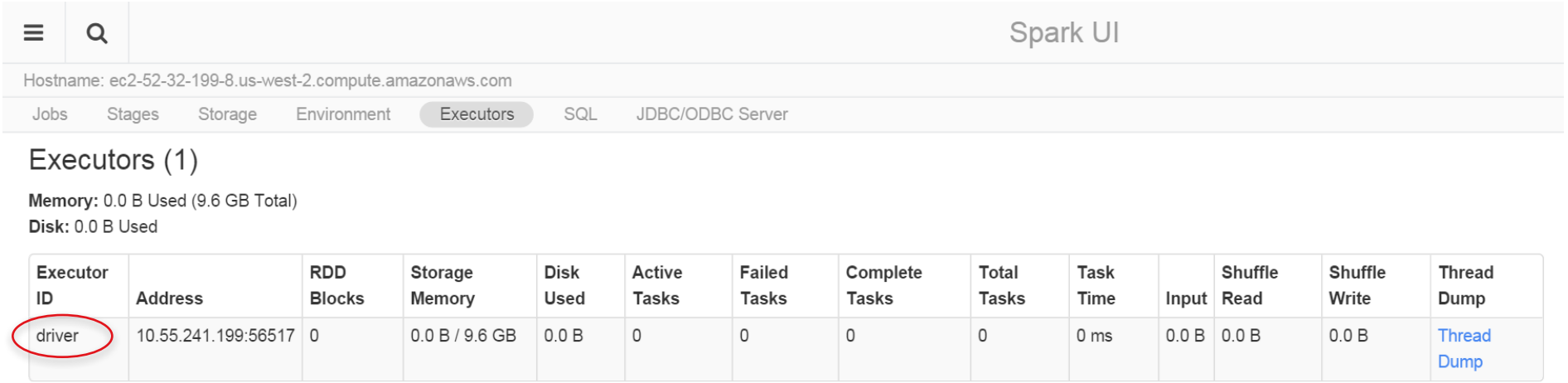


Demo!

- Local mode JVMs
- Executors tab
- %sh free -mh
- %sh jps
- %sh jps –v
- %sh ps -fe
- Events UI for tasks


Spark >= 1.7
Project Tungsten
off heap DF/ RDD
S
S
S
S
On heap JVM
Container

Local Mode UI:

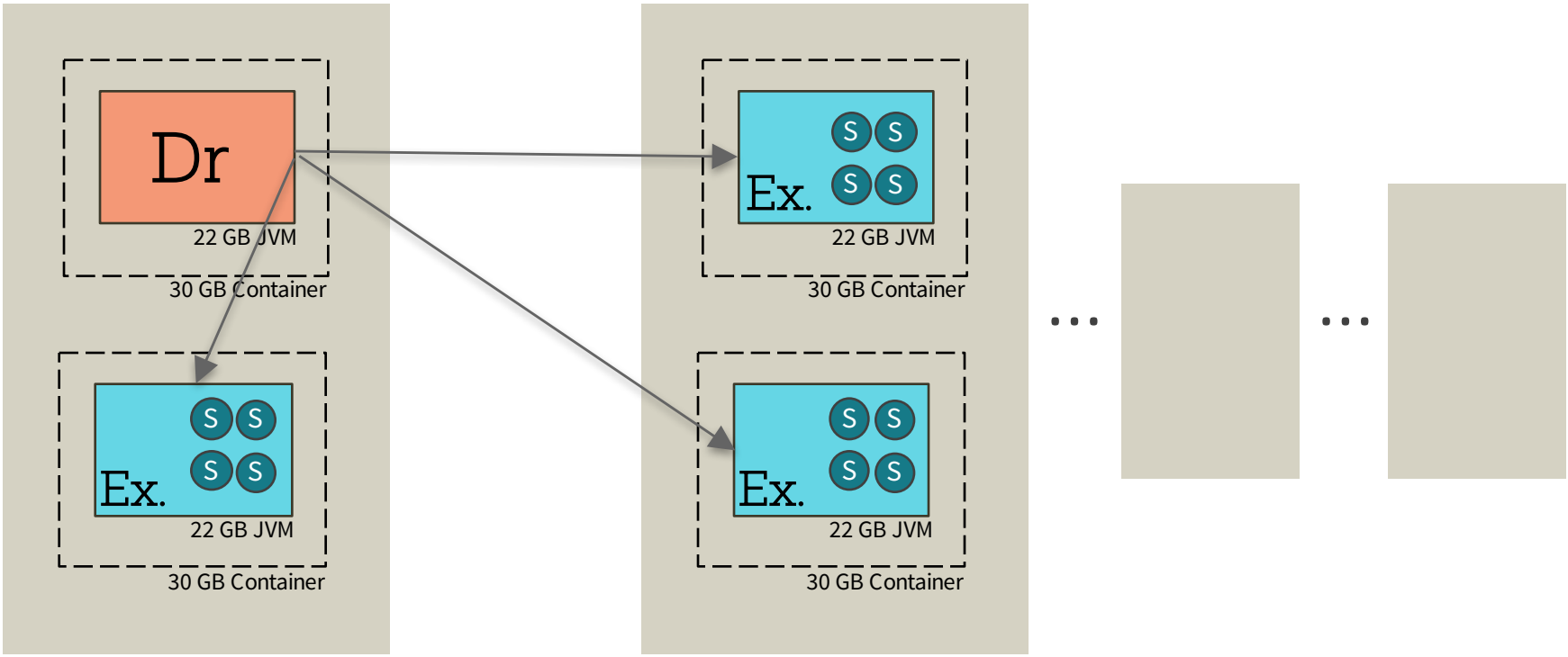


Architecture [revisited]

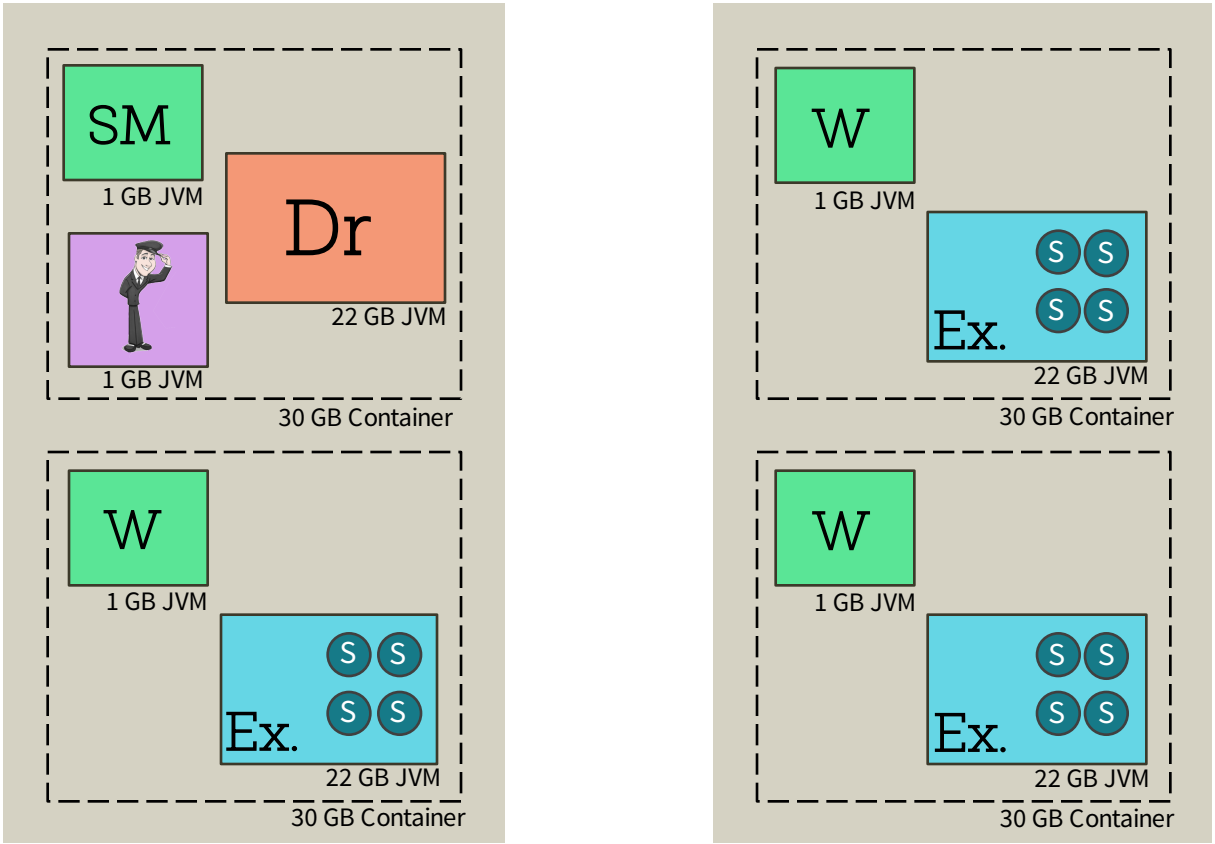
(standalone mode)

databrickstraining

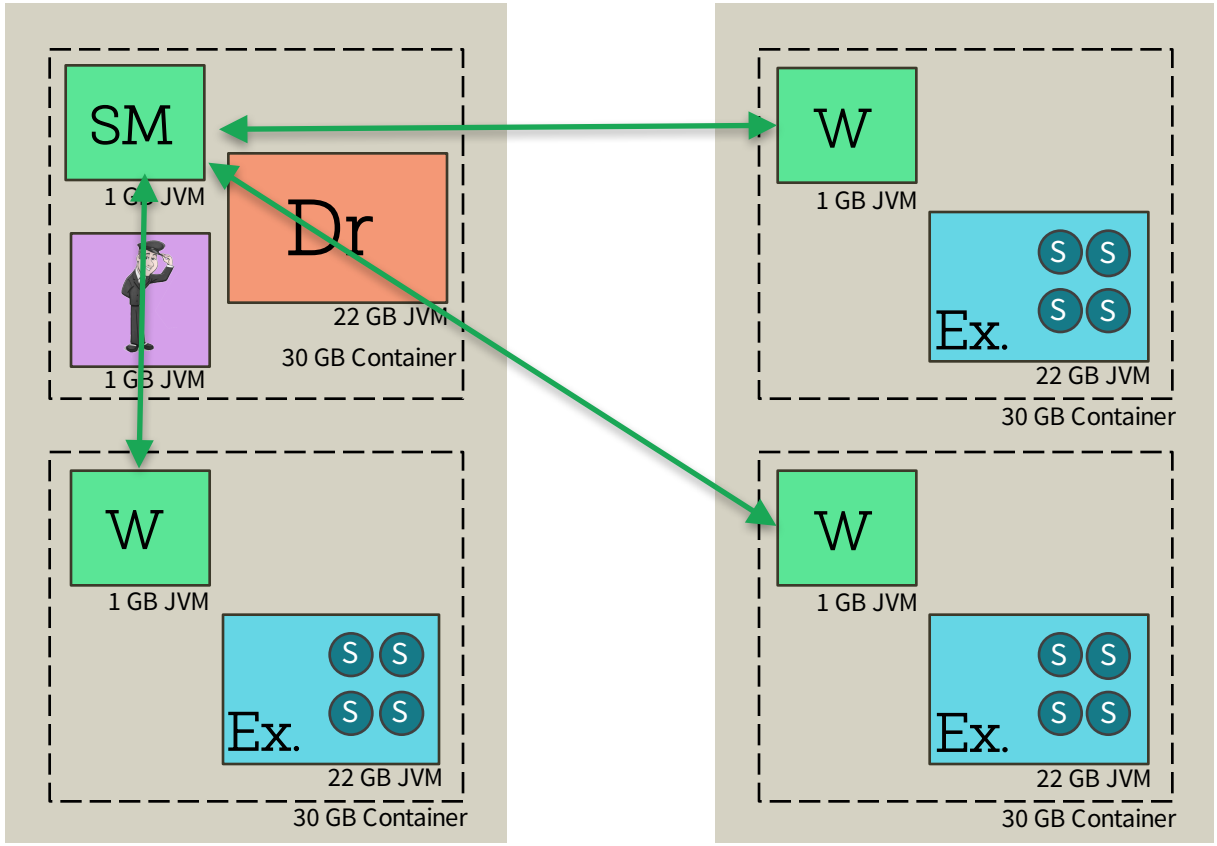
Standalone Mode:



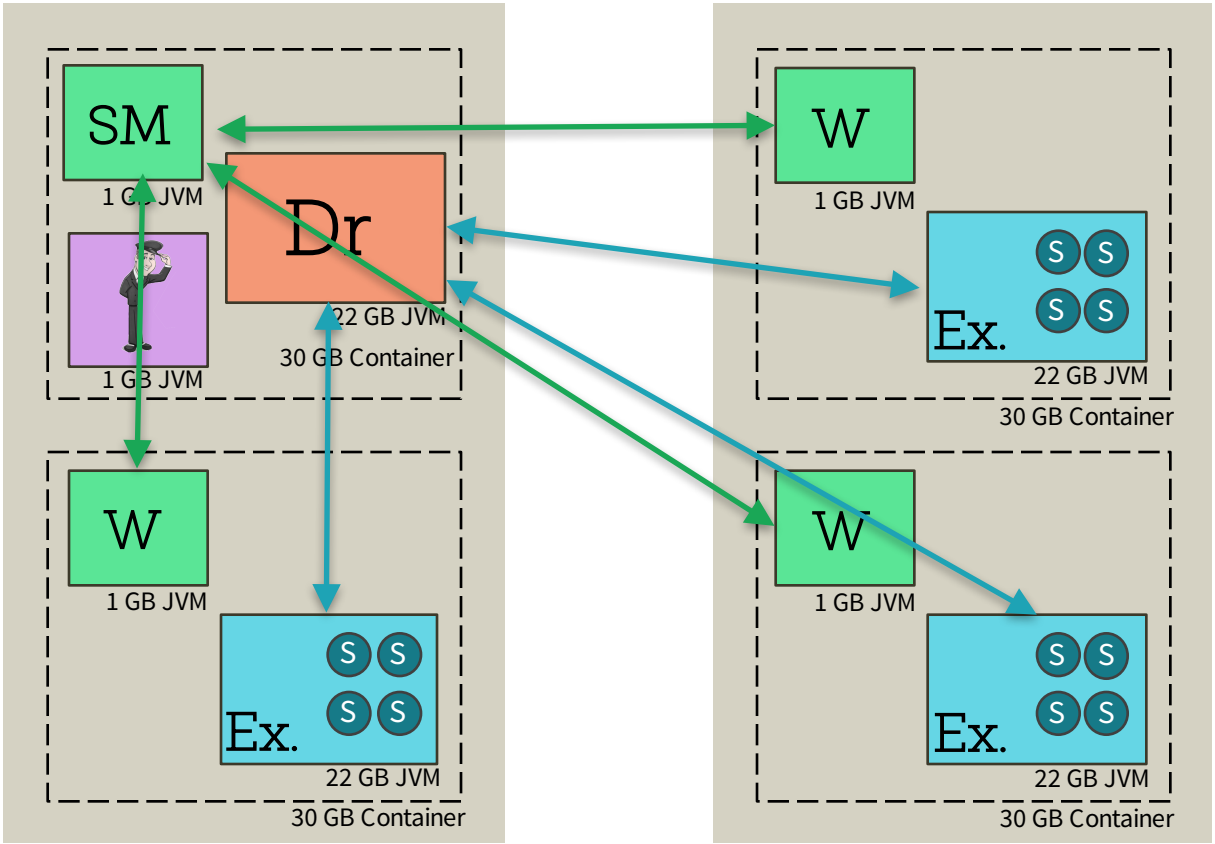
Standalone Mode:



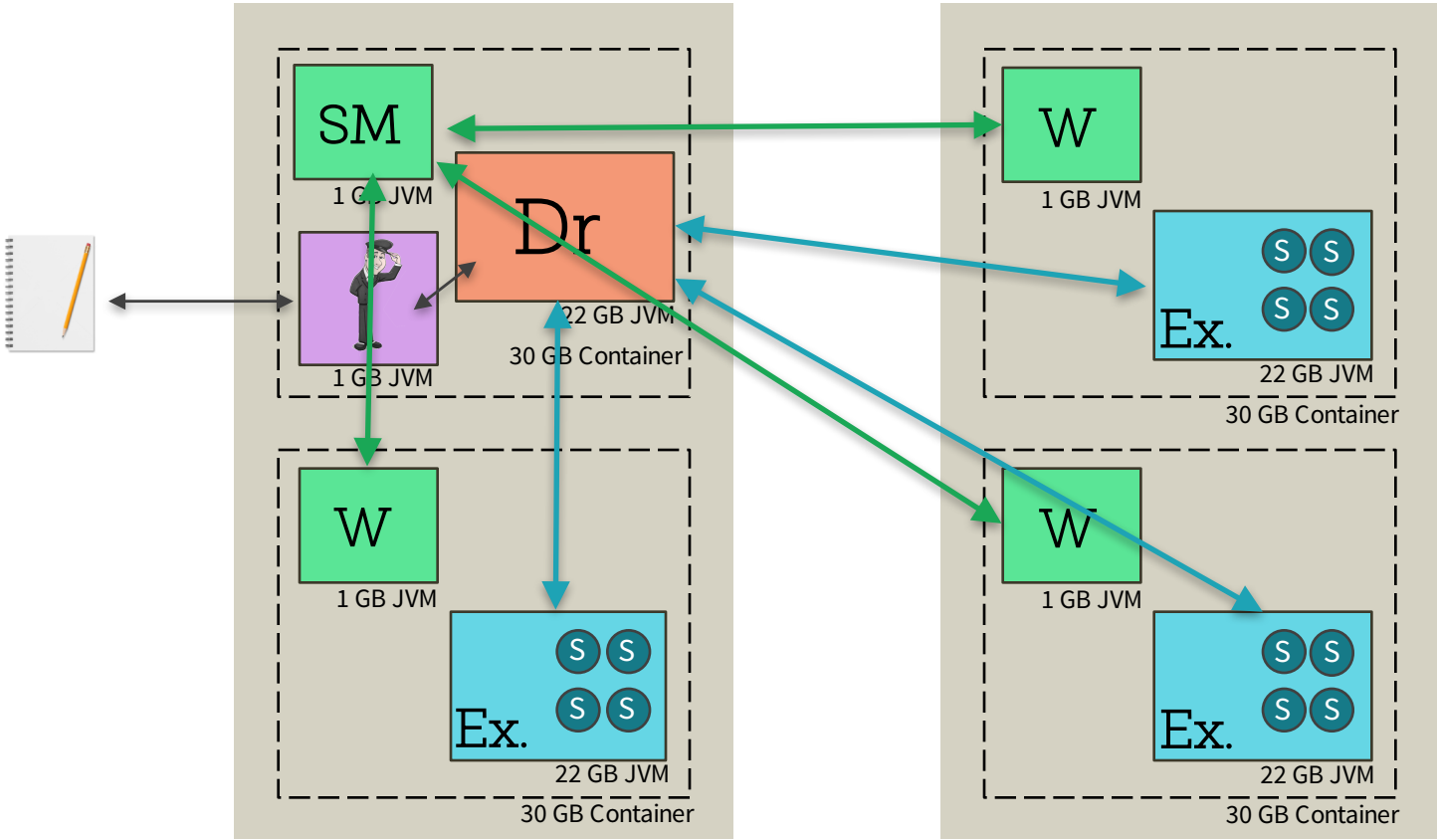
Standalone Mode:

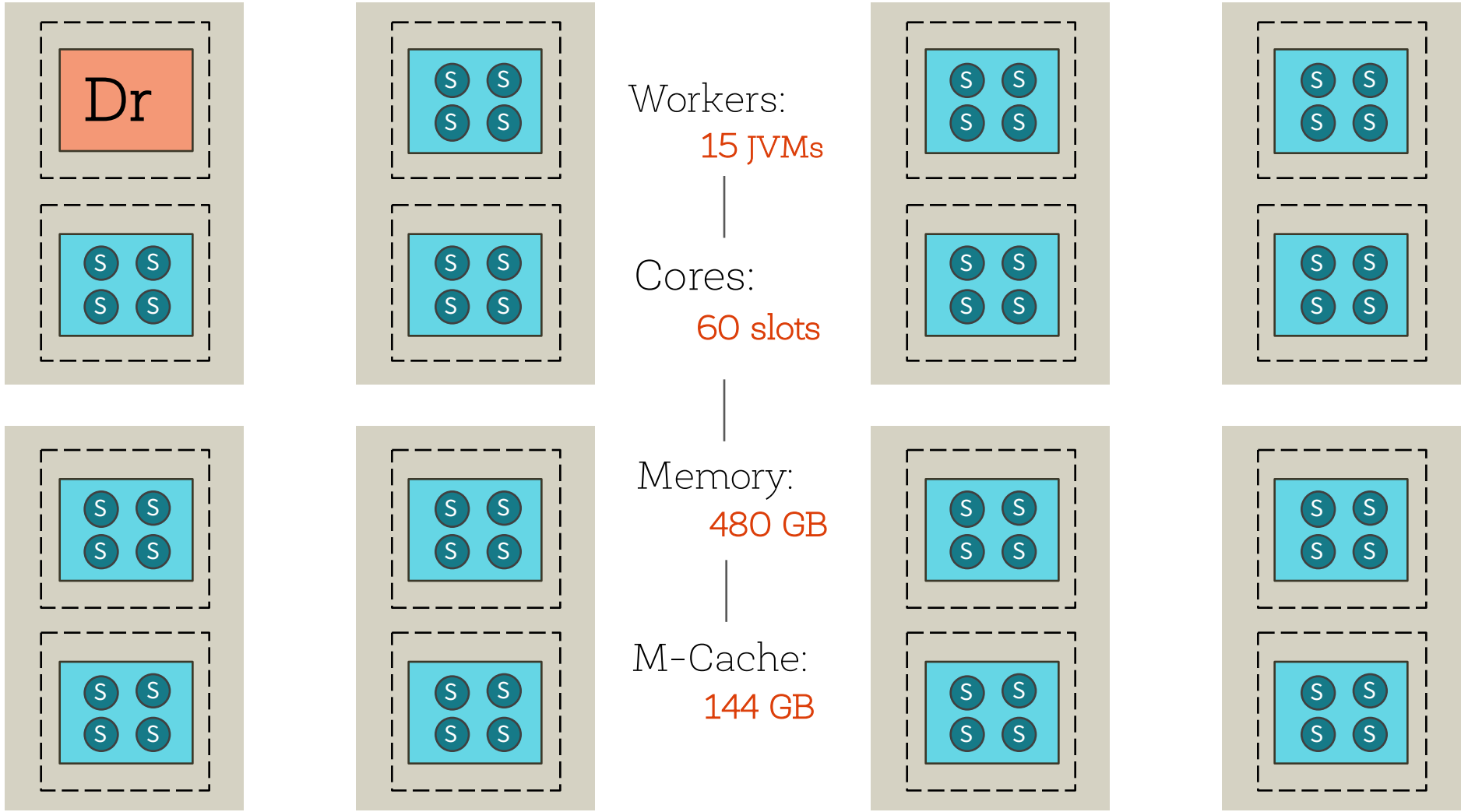


Standalone Mode:



Standalone Mode:

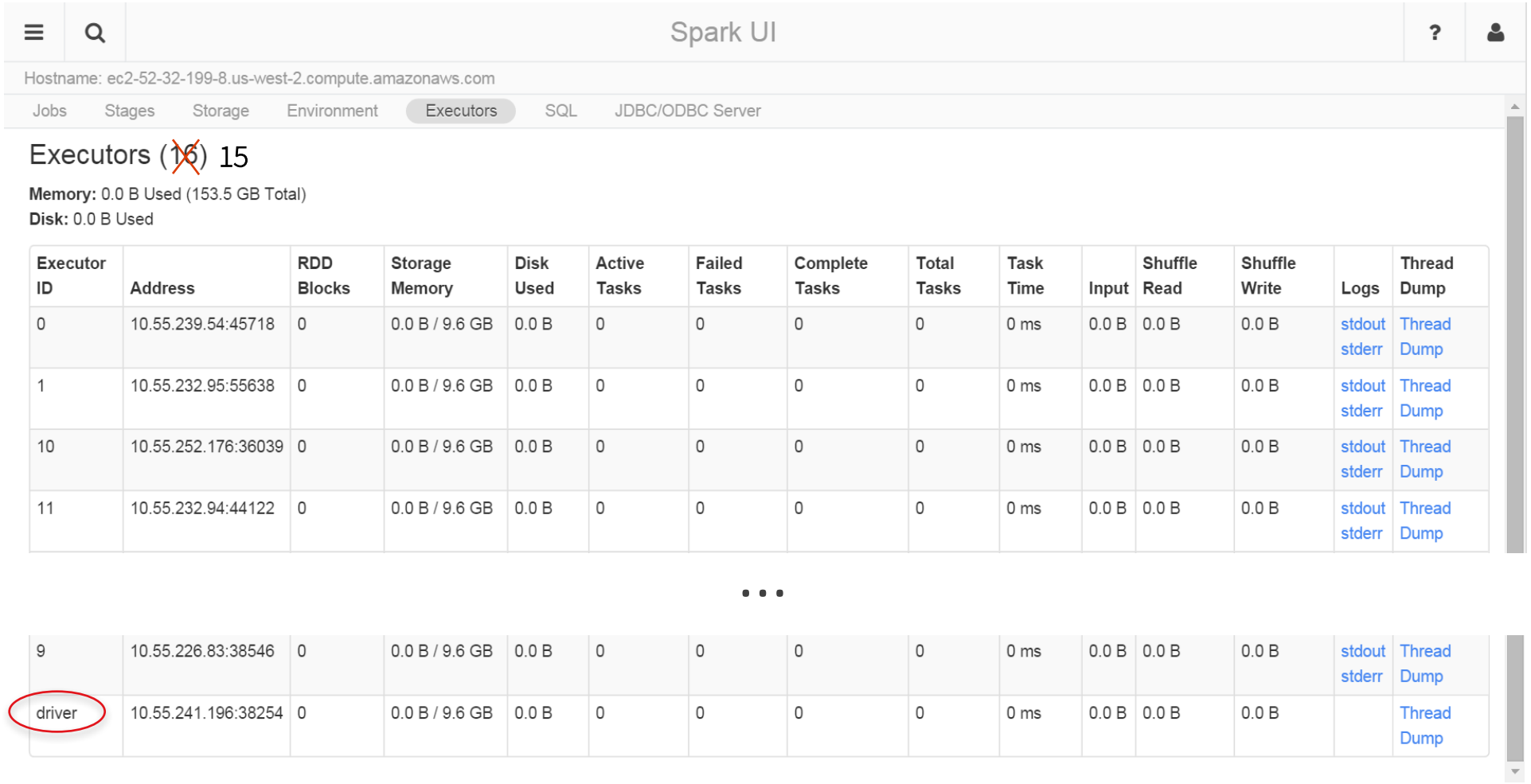



Dr
S
Workers:
15 JVMs
Cores:
60 slots
Memory:
480 GB
M-Cache:
144 GB

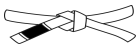
Demo!

- Standalone mode JVMs
- Executors tab
- Spark Master UI
- Spark Worker UI
- Print to stout in Driver & Executor
- conf() settings

Standalone Mode UI:




Spark
: RDDs
databrickstraining

 = easy  = medium

Essential Core & Intermediate Spark Operations

TRANSFORMATIONS

General

- map
- filter
- flatMap
- mapPartitions
- mapPartitionsWithIndex
- groupBy
- sortBy

Math / Statistical

- sample
- randomSplit

Set Theory / Relational

- union
- intersection
- subtract
- distinct
- cartesian
- zip

Data Structure / I/O

- keyBy
- zipWithIndex
- zipWithUniqueID
- zipPartitions
- coalesce
- repartition
- repartitionAndSortWithinPartitions
- pipe

ACTIONS

- reduce
- collect
- aggregate
- fold
- first
- take
- forEach
- top
- treeAggregate
- treeReduce
- forEachPartition
- collectAsMap

- count
- takeSample
- max
- min
- sum
- histogram
- mean
- variance
- stdev
- sampleVariance
- countApprox
- countApproxDistinct

- takeOrdered

- saveAsTextFile
- saveAsSequenceFile
- saveAsObjectFile
- saveAsHadoopDataset
- saveAsHadoopFile
- saveAsNewAPIHadoopDataset
- saveAsNewAPIHadoopFile

 = easy  = medium

Essential Core & Intermediate Spark Operations

TRANSFORMATIONS

General
- flatMapValues
- groupByKey
- reduceByKey
- reduceByKeyLocally
- foldByKey
- aggregateByKey
- sortByKey
- combineByKey
- keys
- values

Math / Statistical
- sampleByKey

Set Theory / Relational
- cogroup (=groupWith)
- join
- subtractByKey
- fullOuterJoin
- leftOuterJoin
- rightOuterJoin

Data Structure / I/O
- partitionBy

ACTIONS

- countByKey
- countByValue
- countByValueApprox
- countApproxDistinctByKey
- countApproxDistinctByKey
- countByKeyApprox
- sampleByKeyExact

Types of RDDs:

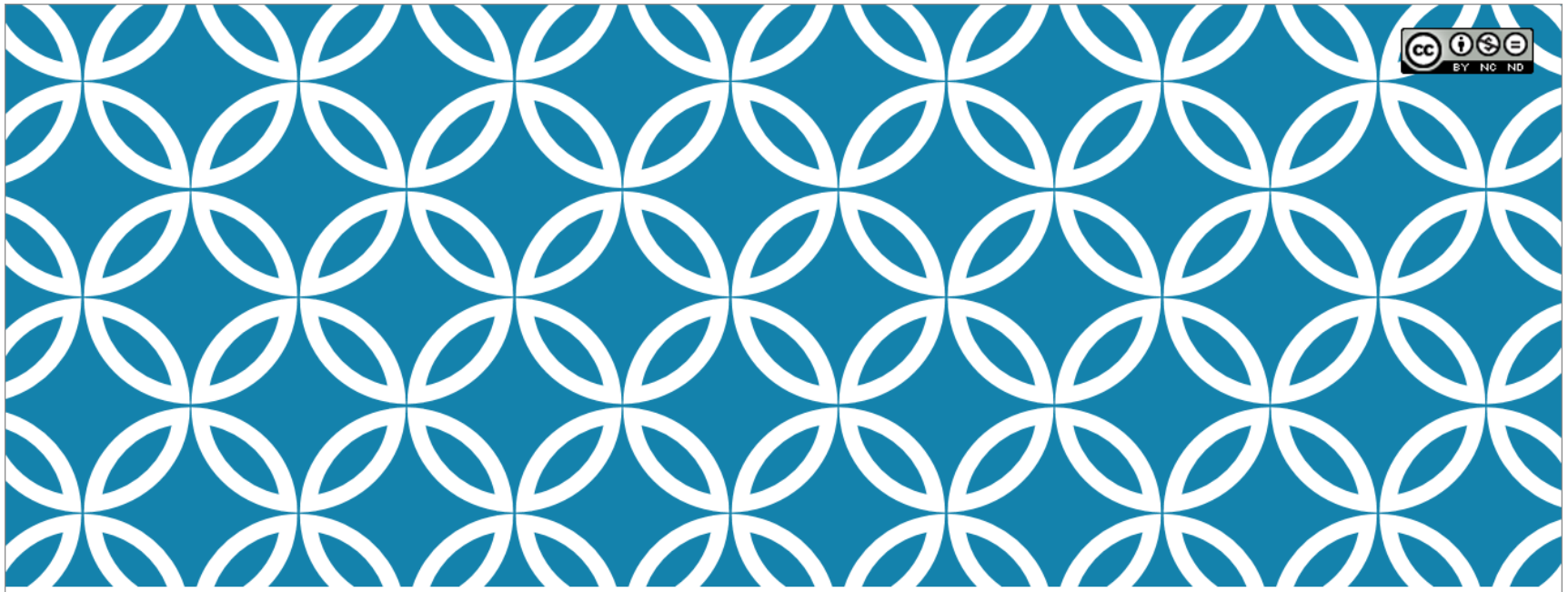
HadoopRDD: core functionality for reading data in HDFS using older MR API

MapPartitionsRDD: a result of calling actions like `map`, `flatMap`, `filter`, `mapPartitions`, etc

PairRDD: holds key/value pairs, a result of `groupByKey` and `join` operations

PipedRDD: result of piping elements to a forked external process

ShuffledRDD: a result after shuffling (`repartition` or `coalesce`)



TRANSFORMATIONS AND ACTIONS

http://training.databricks.com/visualapi.pdf

A Visual Guide of the API

- Most transformations operate on a per item basis, applying the user function to each item `(map, filter, etc)`

- A few transformations operate on a per partition basis, applying the user function *(of type Iterator)* separately on each partition
  `(mapPartitions, mapPartitionsWithIndex, etc)`

2 kinds of Actions

vs

distributed

occurs across the cluster

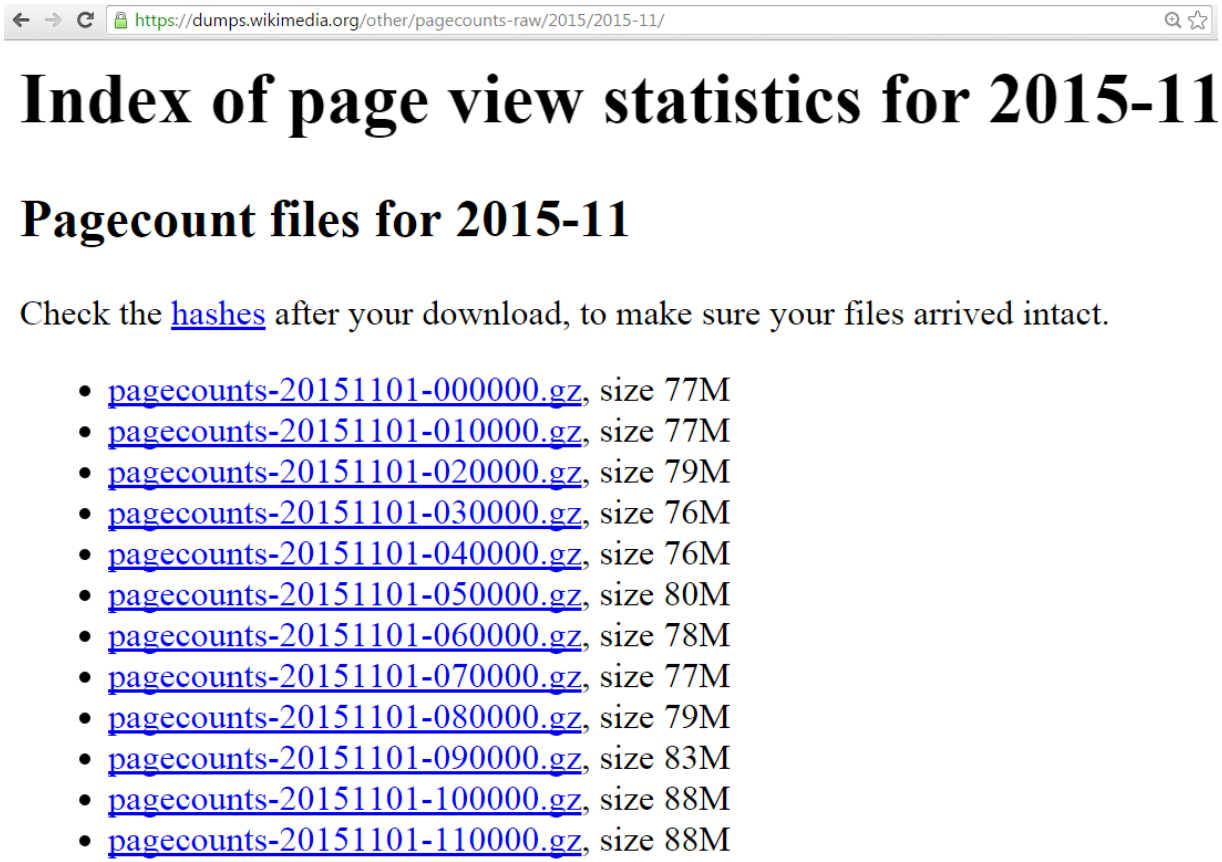
saveAsTextFile, (HDFS, S3, SQL, NoSQL, etc.)

driver

result must fit in driver JVM

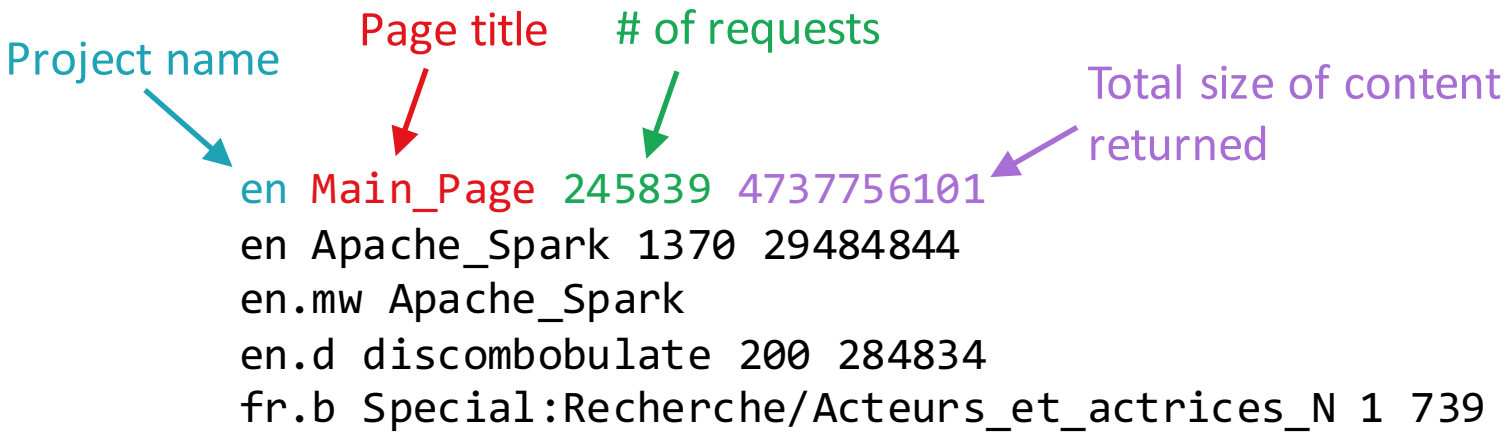
collect, count, reduce, take, show..

Pagecounts:

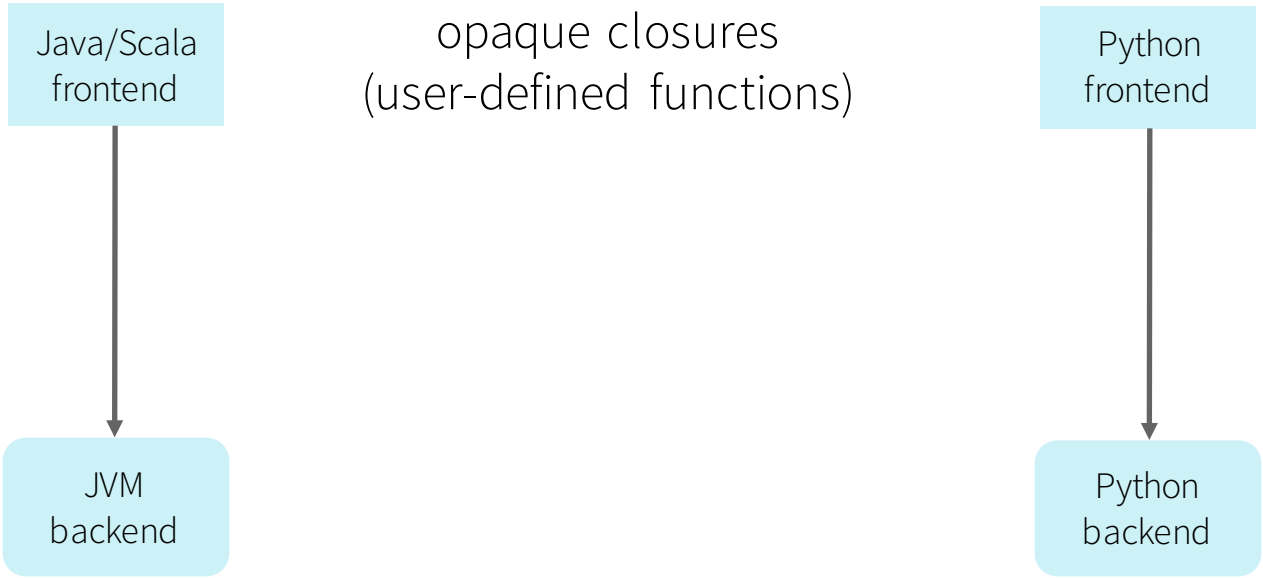

https://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-11/
Index of page view statistics for 2015-11
Pagecount files for 2015-11
Check the hashes after your download, to make sure your files arrived intact.
pagecounts-20151101-000000.gz, size 77M
pagecounts-20151101-010000.gz, size 77M
pagecounts-20151101-020000.gz, size 79M
pagecounts-20151101-030000.gz, size 76M
pagecounts-20151101-040000.gz, size 76M
pagecounts-20151101-050000.gz, size 80M
pagecounts-20151101-060000.gz, size 78M
pagecounts-20151101-070000.gz, size 77M
pagecounts-20151101-080000.gz, size 79M
pagecounts-20151101-090000.gz, size 83M
pagecounts-20151101-100000.gz, size 88M
pagecounts-20151101-110000.gz, size 88M

Pagecounts:

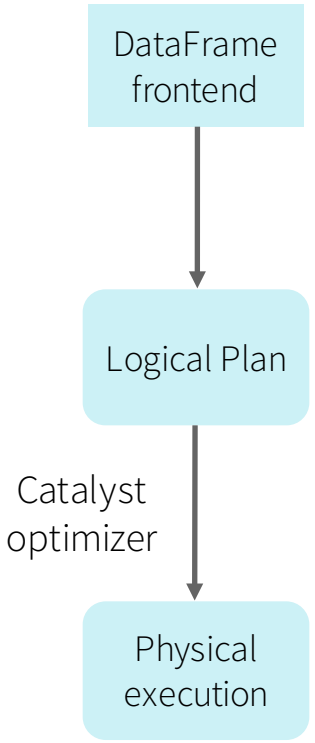
- Hourly Statistics (not unique visits)
- ~500 MB each file, each hour (uncompressed)
- Partitioned Parquet files



Spark RDD Execution

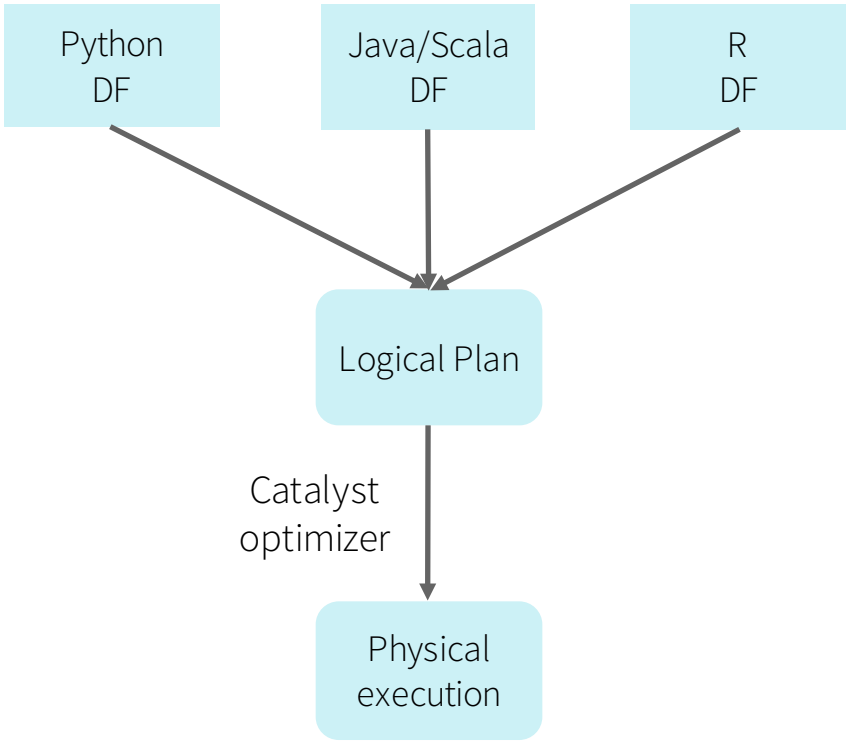


Spark DataFrame Execution



Intermediate representation for computation

Spark DataFrame Execution



Simple wrappers to create logical plan

Intermediate representation for computation

RDDs

Collections of Native JVM Objects

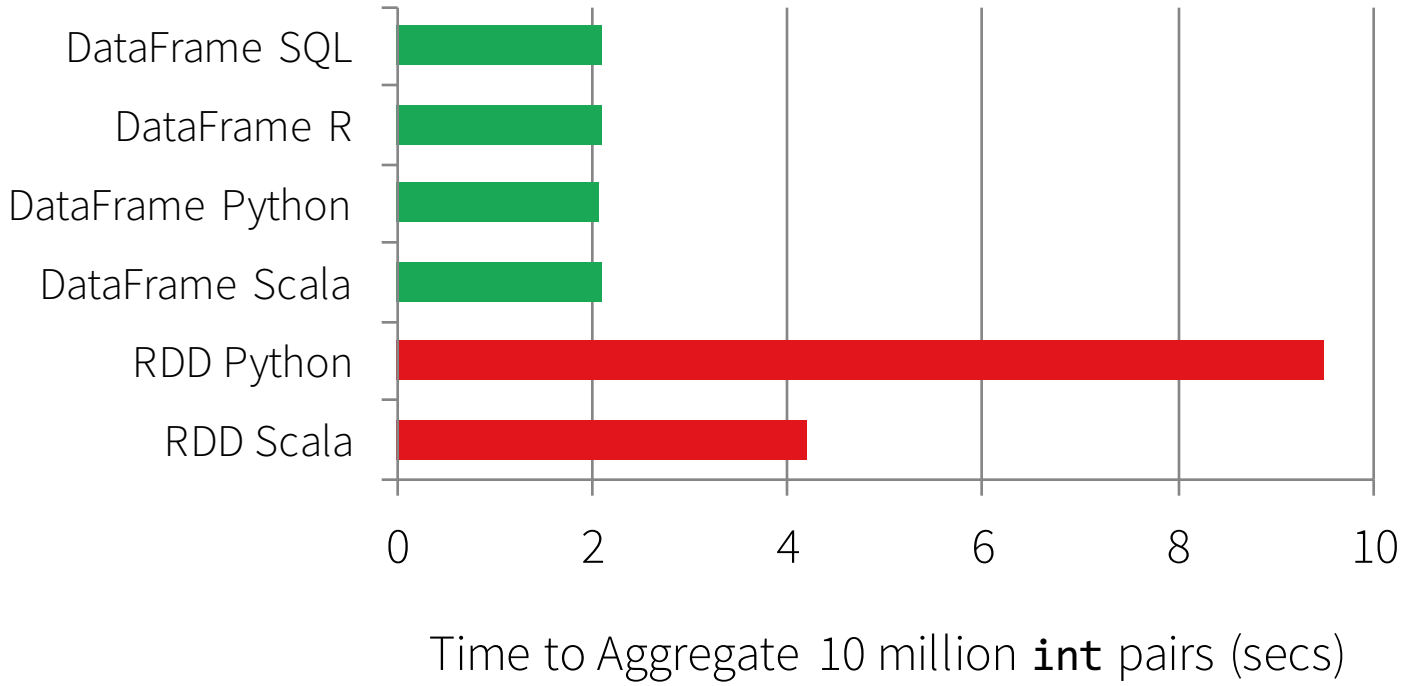
- Compile-time type-safety
- Easy to express certain types of logic
- Lots of existing code + users
- Lower level control of Spark
- Imperative

DataFrames / SQL

Structured Binary Data (Tungsten)

- Lower memory pressure (gc & space)
- Memory accounting (avoids OOMs)
- Faster sorting / hashing / serialization
- More opportunities for automatic optimization
- Declarative

Not Just Less Code: Faster Implementations

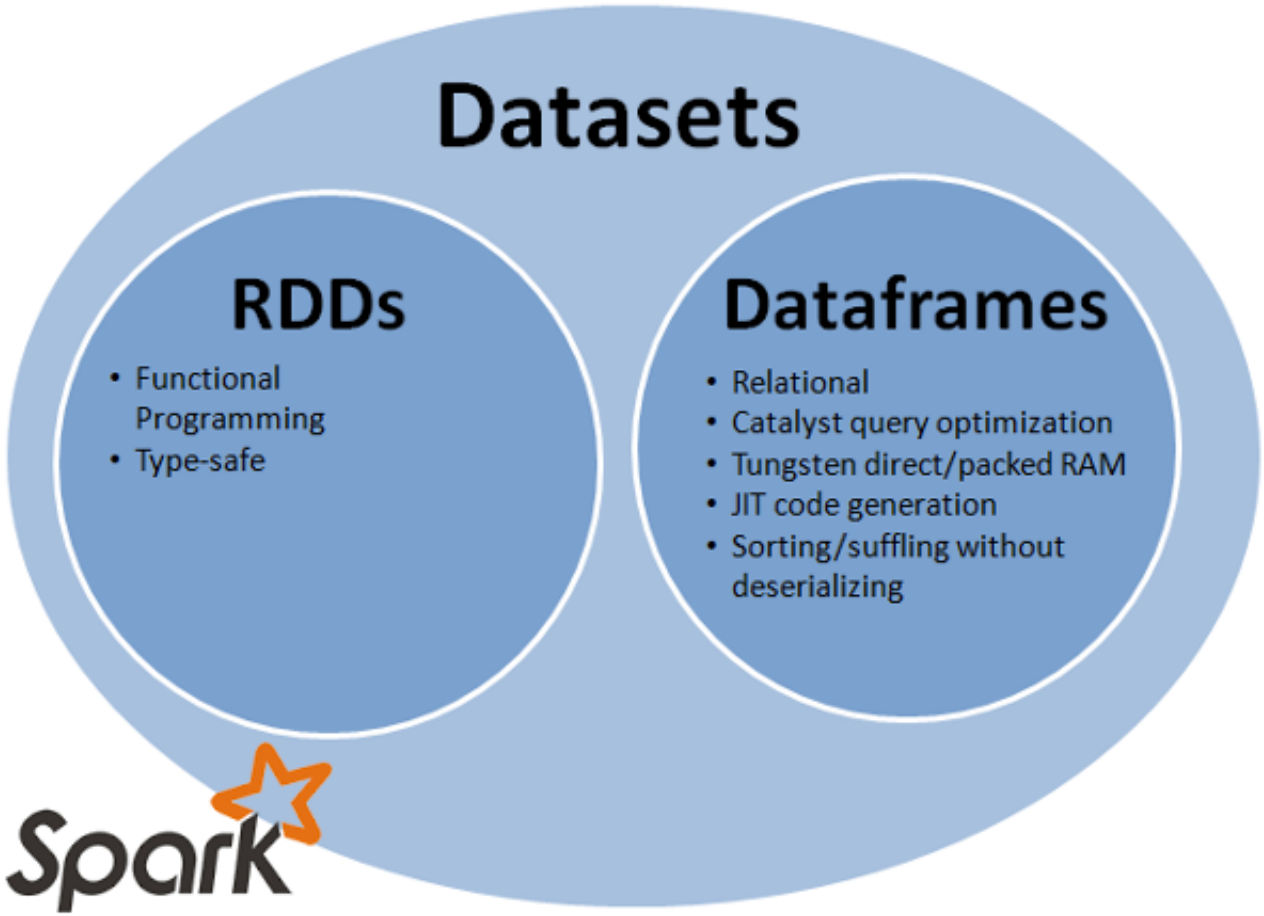


https://gist.github.com/rxin/c1592c133e4bccf515dd

Dataset API

- DataFrames are useful, but their API is dynamically typed: can we get their performance advantages while keeping static typing?

- New API arrived in Spark 1.6



Source: michaelmalak

Dataset API in Spark 1.6

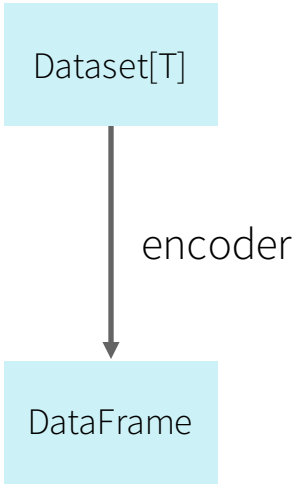
- Typed interface over DataFrames / Tungsten

```
• case class Person(name: String, age: Int)

• val dataframe = read.json(“people.json”)
• val ds: Dataset[Person] = dataframe.as[Person]

• ds.filter(p => p.name.startsWith(“M”))
    .groupBy(“name”)
    .avg(“age”)
```

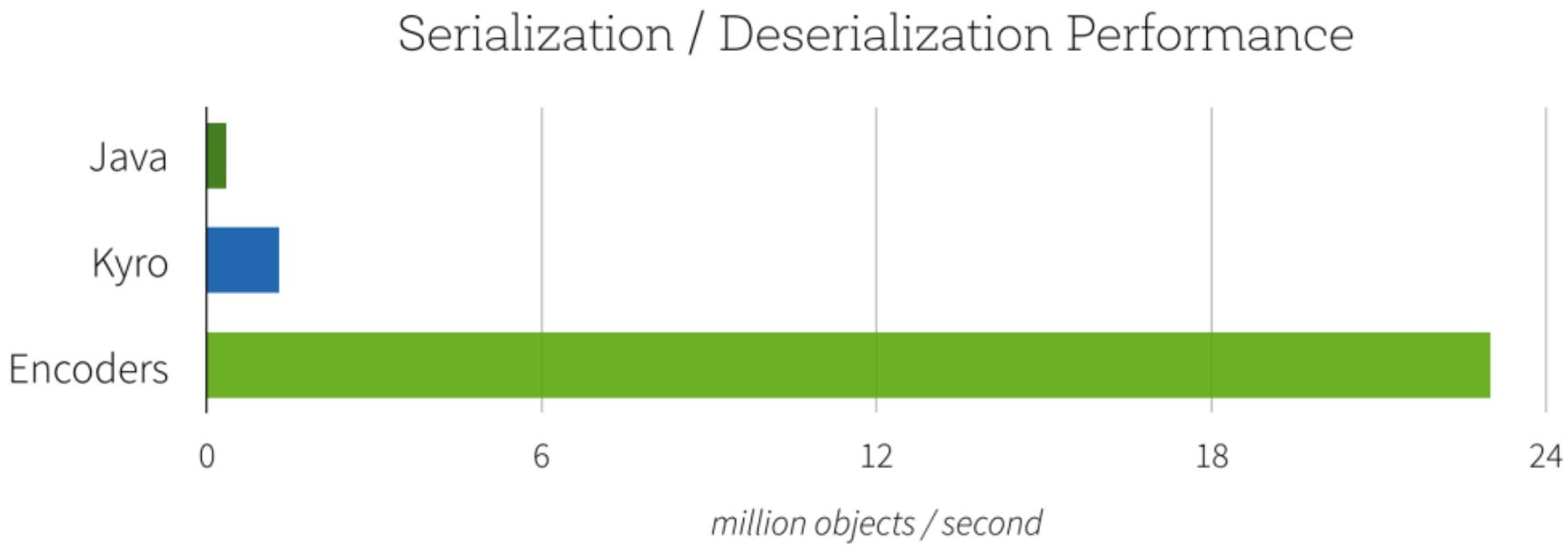
Dataset

- "Encoder" to specify type information so Spark can translate it into DataFrame and generate optimized memory layouts

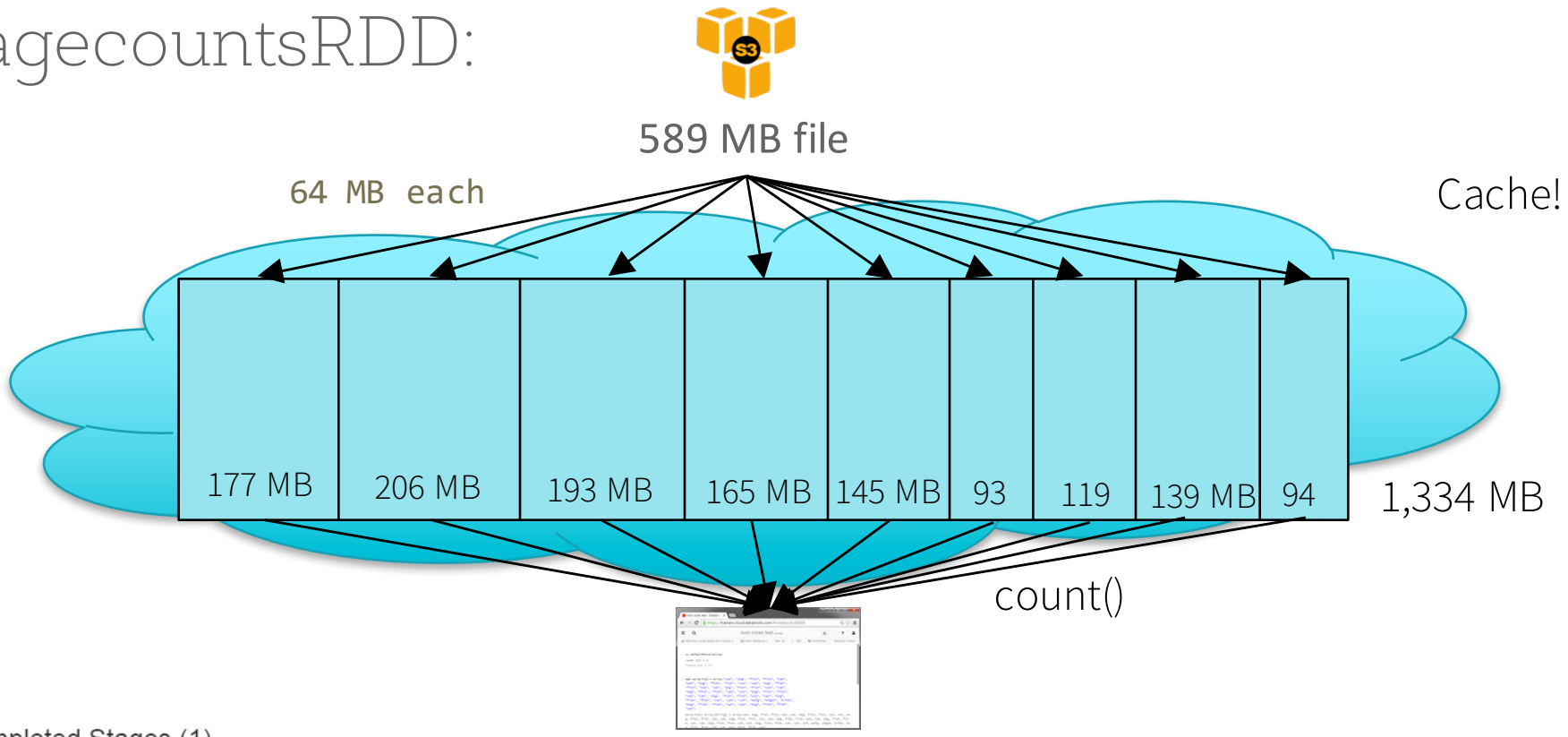
- Checkout SPARK-9999



Datasets: Lightning-fast Serialization with Encoders



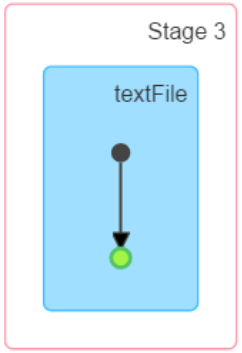
pagecountsRDD:



Completed Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
3	7463896219951833835	pagecountsRDD.setName("pagecountsRDD").cache.count count at <console>:29 +details	2015/11/30 10:26:45	5 s	9/9	562.4 MB			

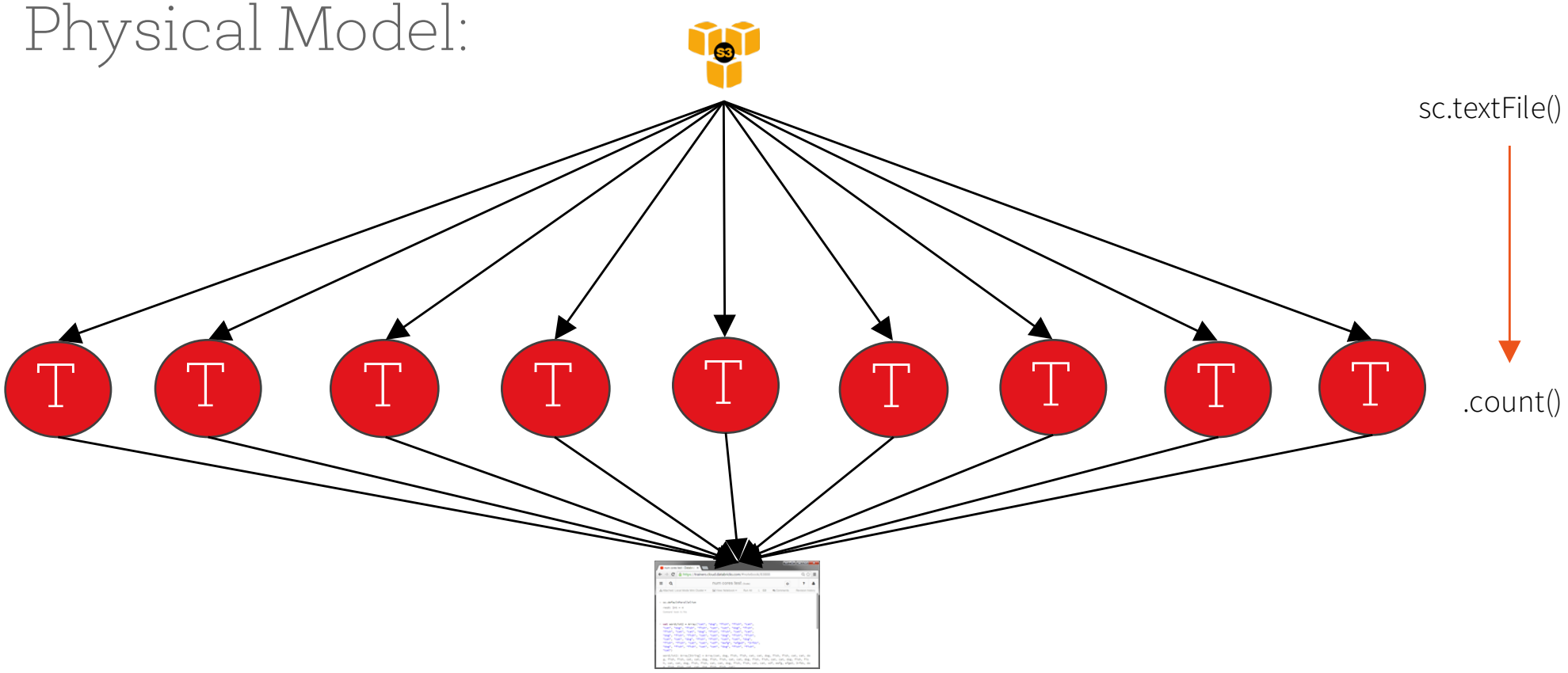
pagecountsRDD:



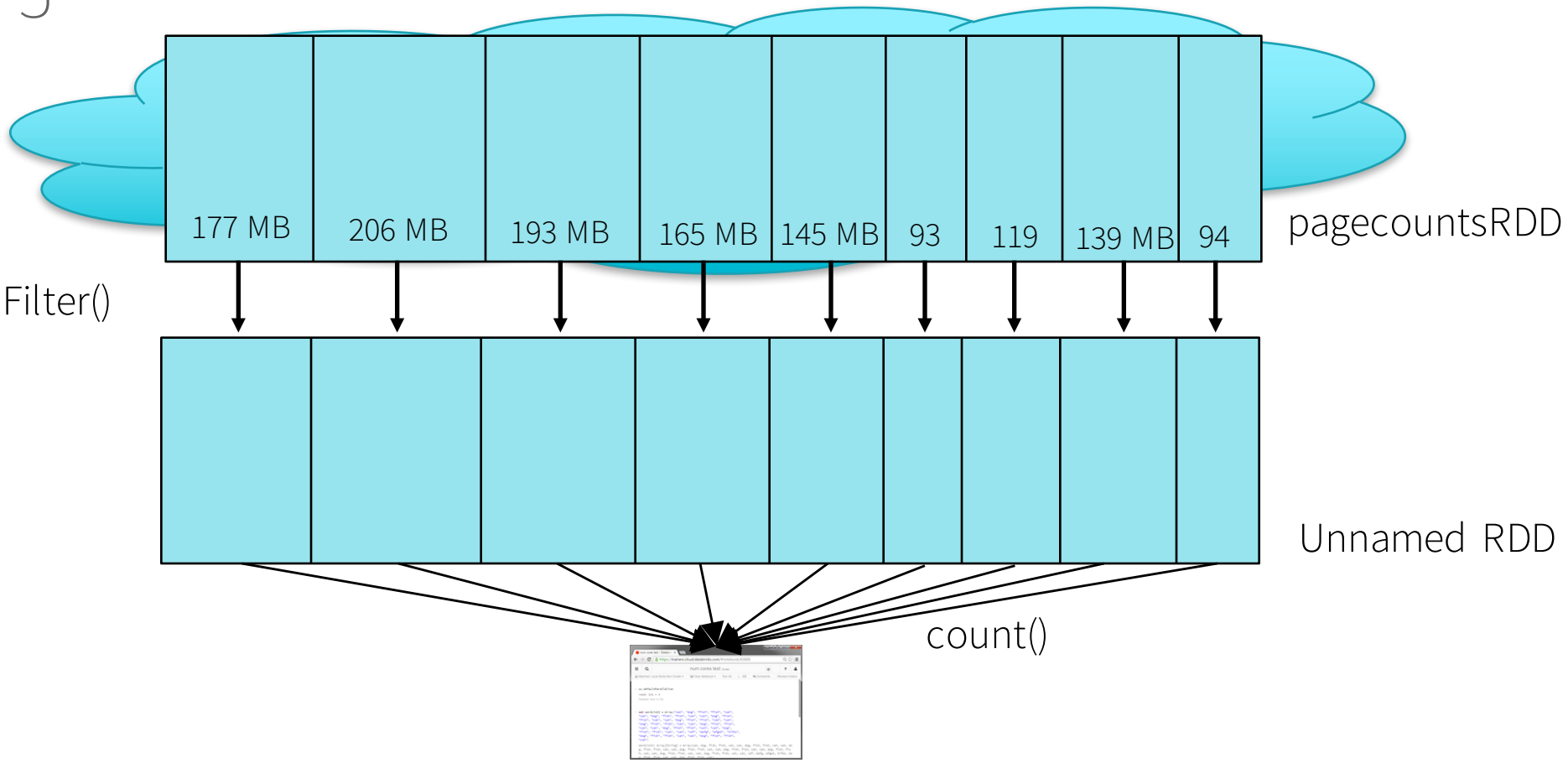
Tasks

Index ?	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	13	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:26:45	2 s		64.0 MB (hadoop) / 1398177	
1	14	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:26:45	2 s		64.0 MB (hadoop) / 1906223	
2	15	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:26:45	2 s		64.0 MB (hadoop) / 1709442	
3	16	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:26:45	2 s		64.0 MB (hadoop) / 1226943	
4	17	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:26:46	2 s	0.4 s	64.0 MB (hadoop) / 778096	
5	18	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:26:46	2 s	0.4 s	64.0 MB (hadoop) / 306501	
6	19	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:26:46	2 s	0.4 s	64.0 MB (hadoop) / 90926	
7	20	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:26:46	2 s	0.4 s	64.0 MB (hadoop) / 161412	
8	21	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:26:48	1 s		50.4 MB (hadoop) / 209041	

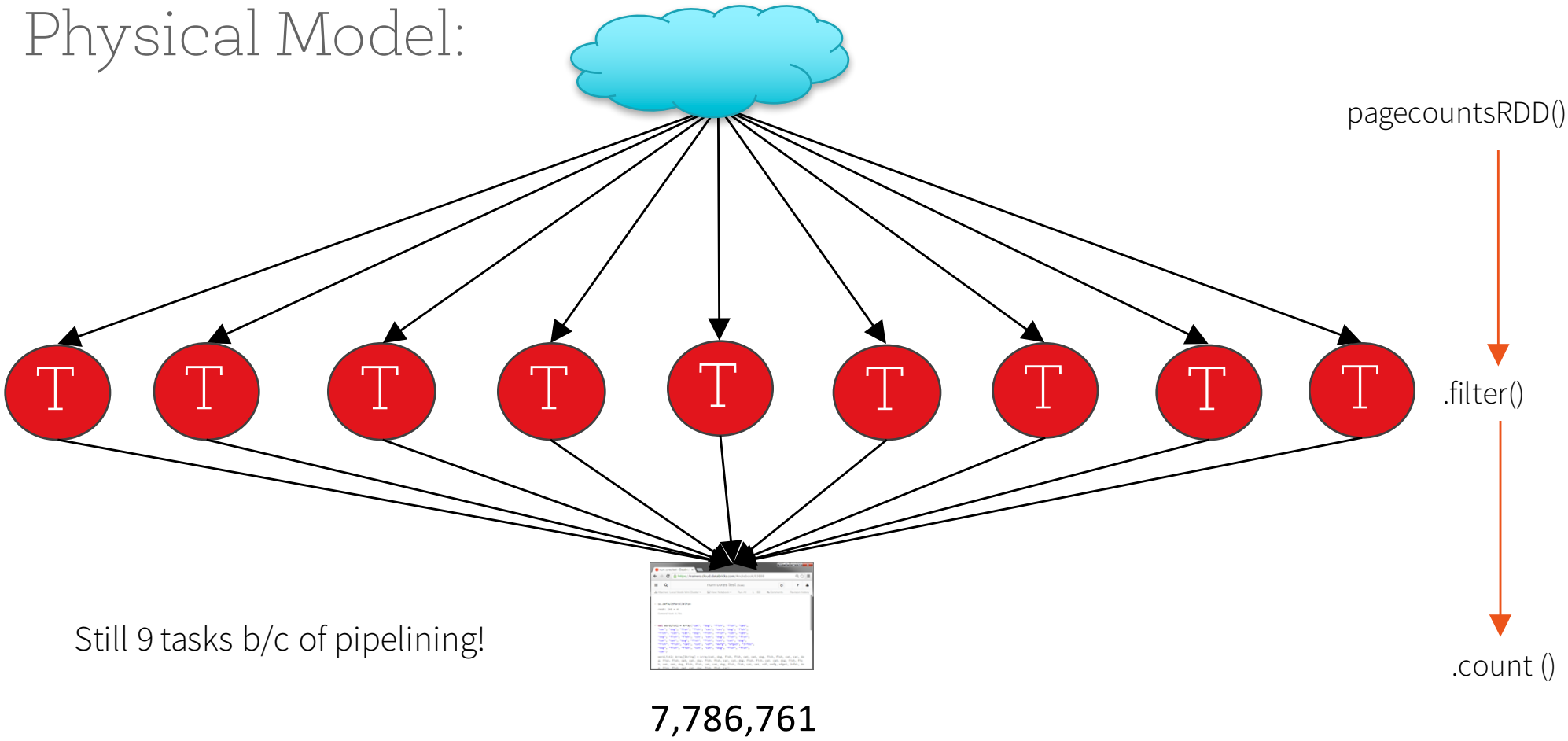
Physical Model:



pagecountsRDD:

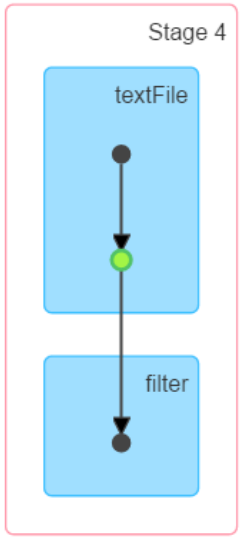


Physical Model:



Still 9 tasks b/c of pipelining!

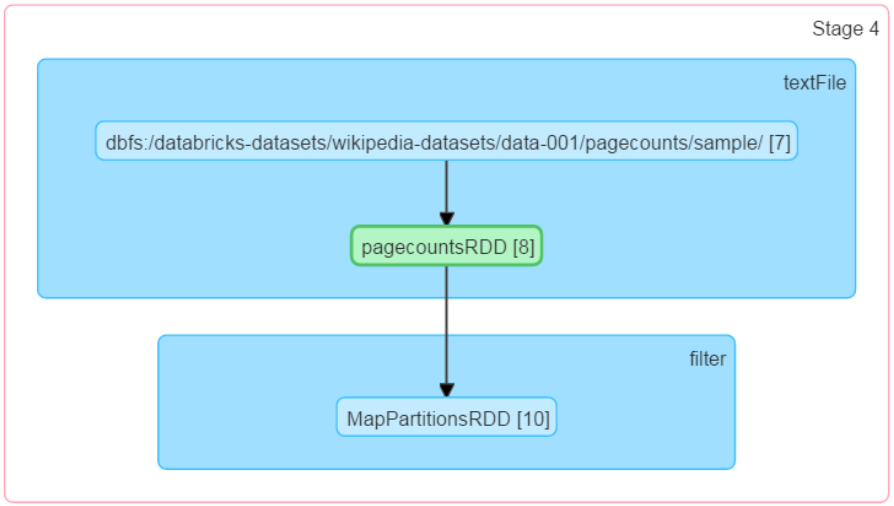
pagecountsRDD:



Completed Stages (1)

Stage Id	Pool Name	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
4	7463896219951833835	enPagecountsRDD.count() count at <console>:31 +details	2015/11/30 10:46:55	0.2 s	9/9	1334.3 MB			

pagecountsRDD:



Tasks

Index ?	ID	Attempt	Status	Locality Level	Executor ID / Host	Launch Time	Duration	GC Time	Input Size / Records	Errors
0	22	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:46:55	62 ms		177.2 MB (memory) / 1398177	
1	23	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:46:55	94 ms		206.1 MB (memory) / 1906223	
2	24	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:46:55	99 ms		193.8 MB (memory) / 1709442	
3	25	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:46:55	82 ms		165.2 MB (memory) / 1226943	
4	26	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:46:55	55 ms		145.2 MB (memory) / 778096	
5	27	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:46:55	21 ms		93.0 MB (memory) / 306501	
6	28	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:46:55	9 ms		119.8 MB (memory) / 90926	
7	29	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:46:55	14 ms		139.6 MB (memory) / 161412	
8	30	0	SUCCESS	PROCESS_LOCAL	driver / localhost	2015/11/30 10:46:55	16 ms		94.4 MB (memory) / 209041	

pagecountsRDD:

Summary Metrics for 9 Completed Tasks

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	9 ms	16 ms	55 ms	82 ms	99 ms
GC Time	0 ms	0 ms	0 ms	0 ms	0 ms
Input Size / Records	93.0 MB / 90926	119.8 MB / 209041	145.2 MB / 778096	177.2 MB / 1398177	206.1 MB / 1906223

Lab!

- Analyzing Pagecounts with RDDs

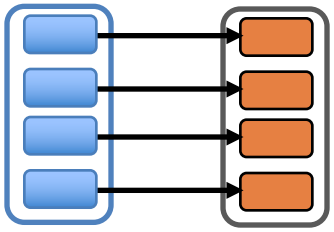
vs

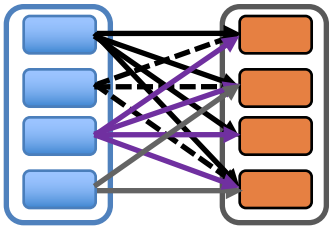
narrow

wide

each partition of the parent RDD is used by at most one partition of the child RDD

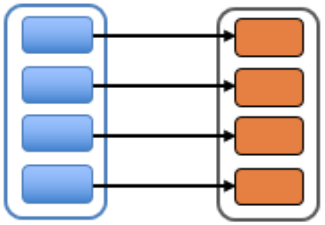
multiple child RDD partitions may depend on a single parent RDD partition



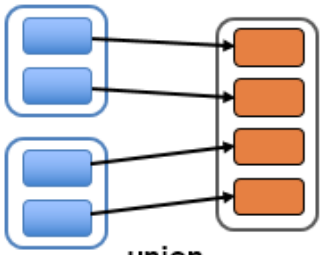


narrow

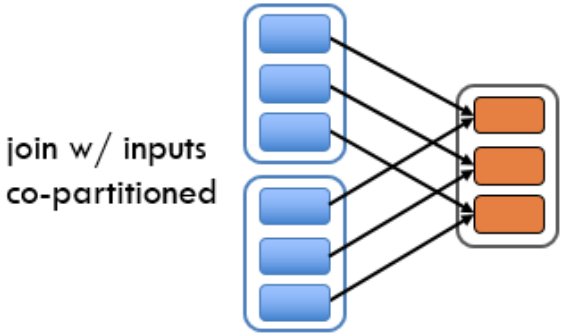
each partition of the parent RDD is used by at most one partition of the child RDD



map, filter



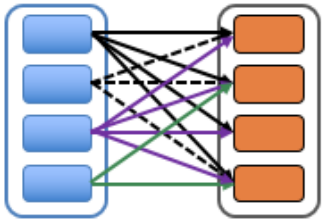
union



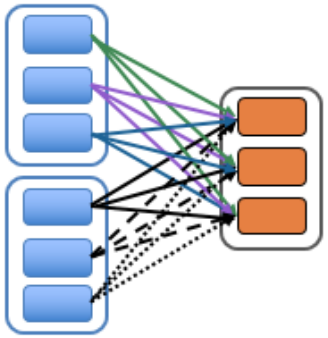
join w/ inputs co-partitioned

wide

multiple child RDD partitions may depend on a single parent RDD partition



groupByKey



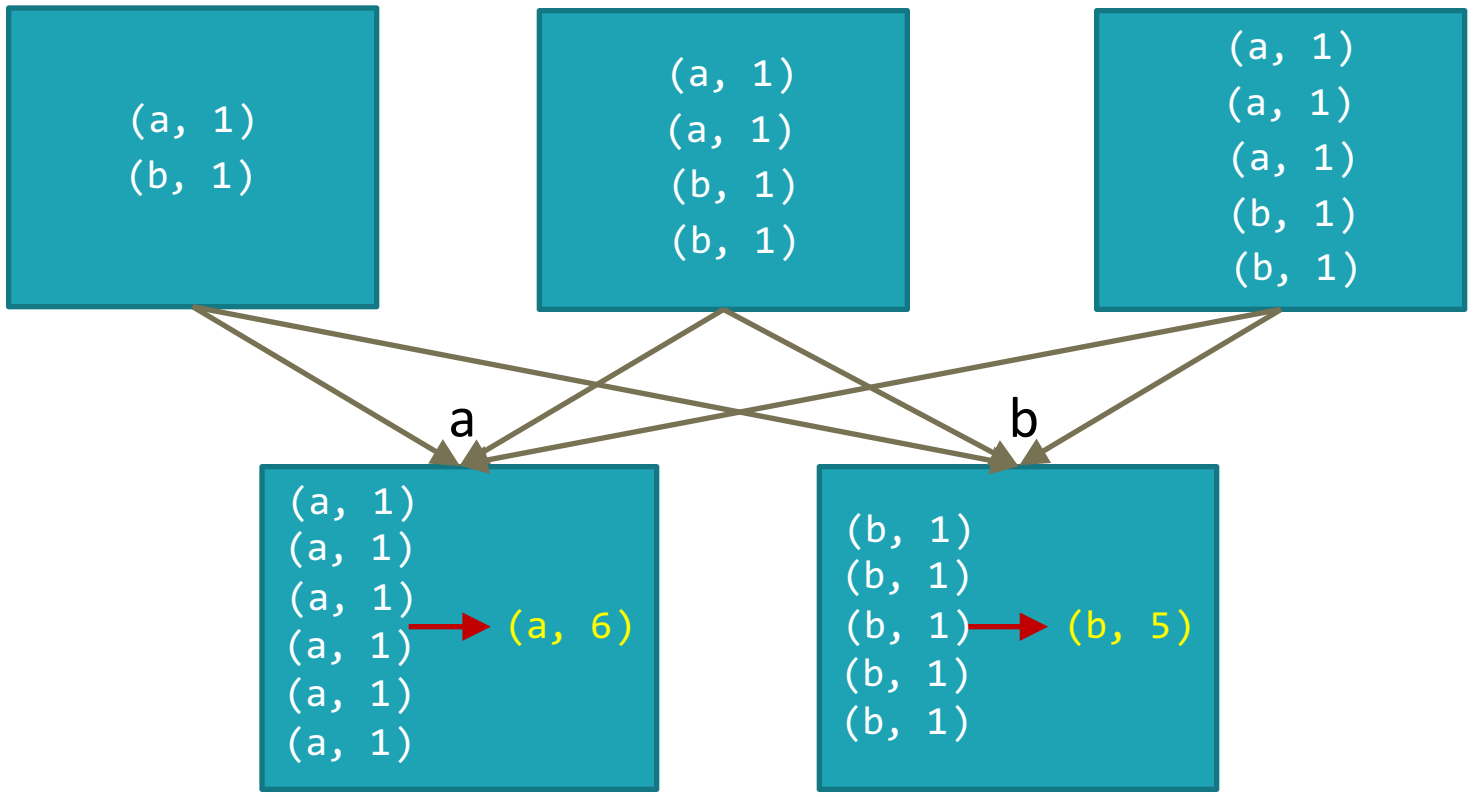
join w/ inputs not co-partitioned

Which API Call Causes Most Tickets?

- `map`
- `filter`
- `groupBy`
- `sort`
- `union`
- `join`
- `leftOuterJoin`
- `rightOuterJoin`
- `reduce`
- `count`
- `fold`
- `reduceByKey`
- `groupByKey`
- `cogroup`
- `cross`
- `zip`
- `sample`
- `take`
- `first`
- `partitionBy`
- `mapWith`
- `pipe`
- `save`
- `...`

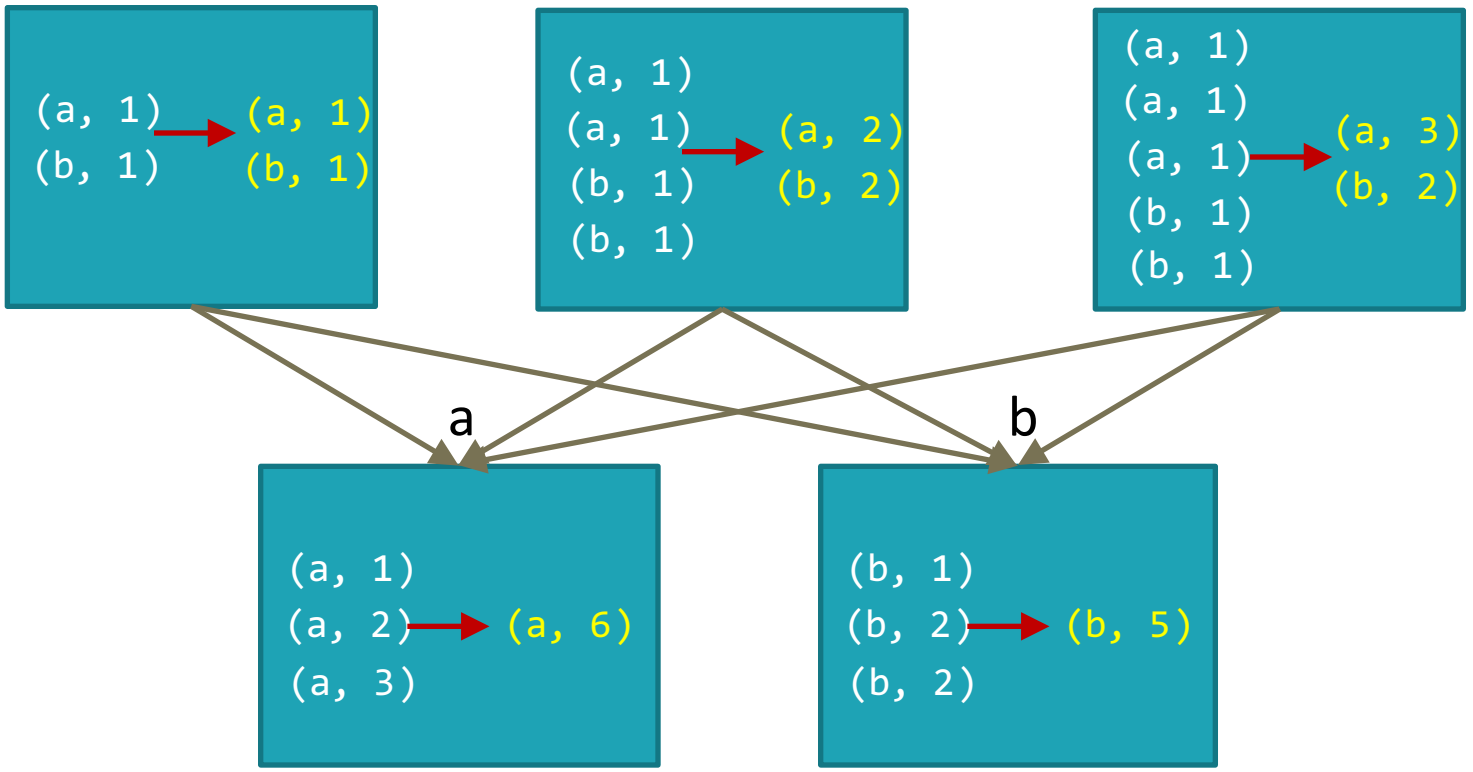
GroupByKey

Shuffles all the data



ReduceByKey

Shuffles only the results of sub-aggregations in each partition of the data



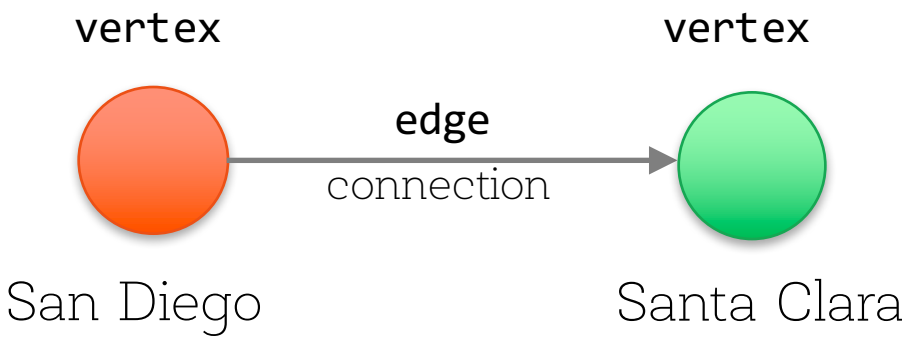

Spark
Memory
databrickstraining

Memory Storage Levels:

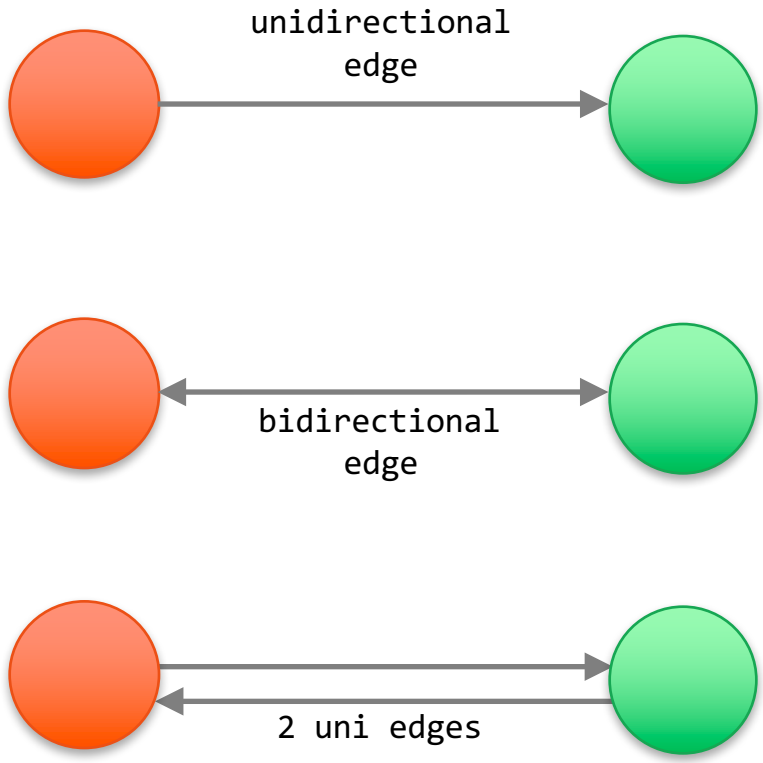
- MEMORY_ONLY
- MEMORY_AND_DISK
- MEMORY_ONLY_SER
- MEMORY_AND_DESK_SER
- DISK_ONLY
- MEMORY_ONLY_2, MEMORY_AND_DISK_2, etc
- OFF_HEAP

Special Case:

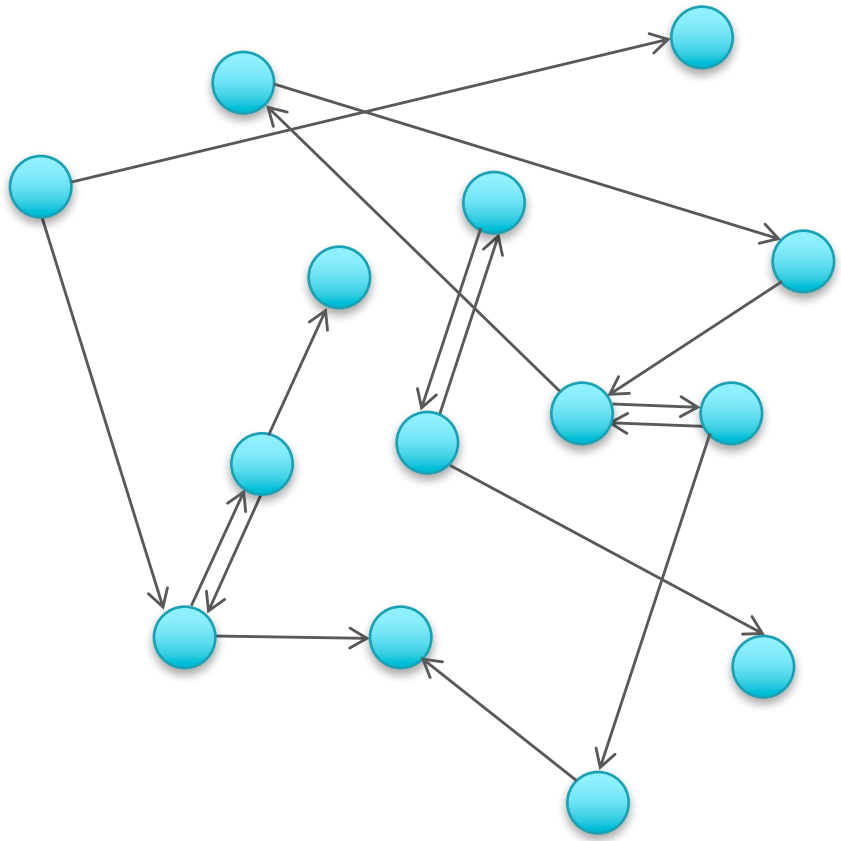

Spark
GraphX
databrickstraining

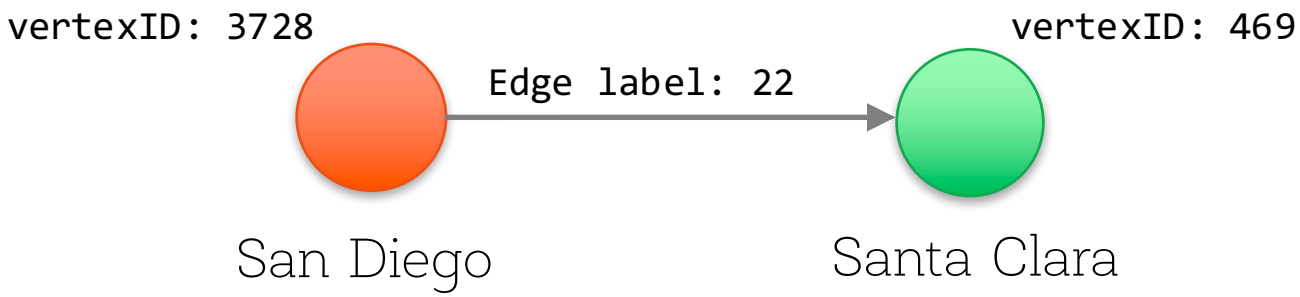

vertex
vertex
edge
connection
San Diego
Santa Clara

Edge types:

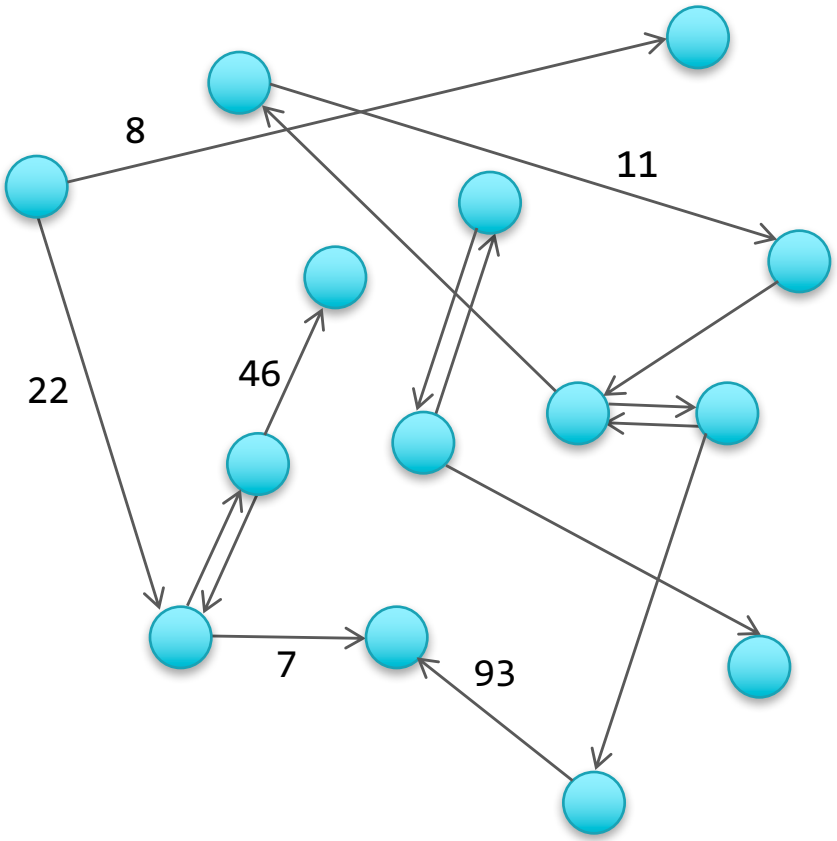


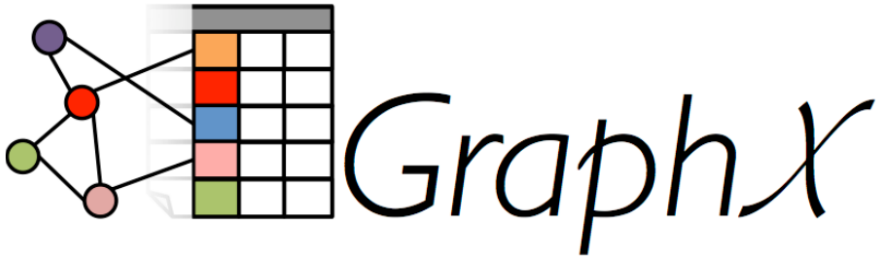
GraphX: directed multigraph




vertexID: 3728
vertexID: 469
Edge label: 22
San Diego
Santa Clara

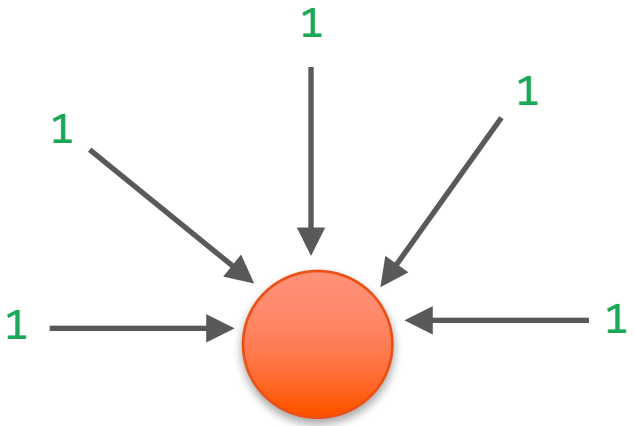
GraphX: directed multigraph w/ labels





Algorithms Tour

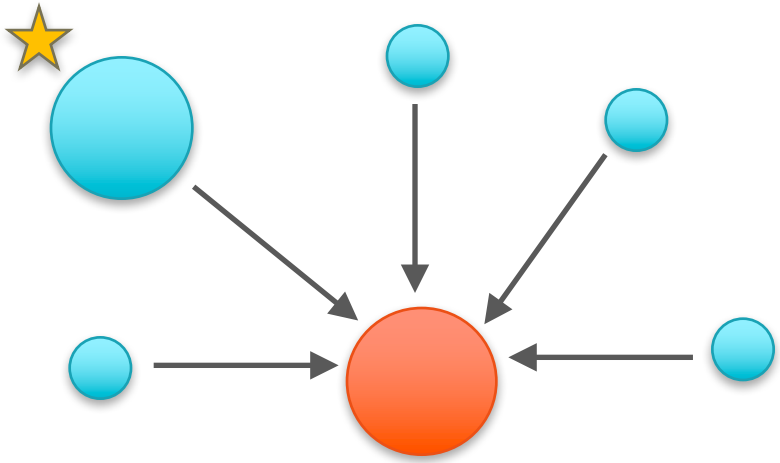
In-degree:



San Diego

`inDegree = 5`

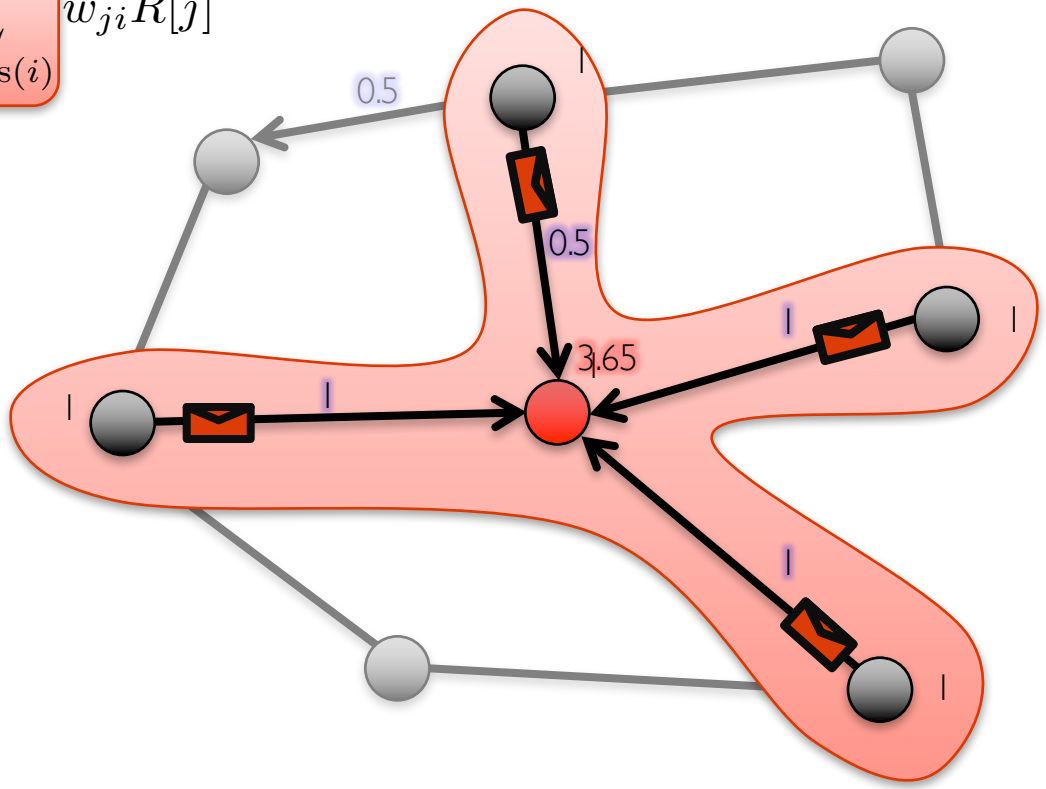
PageRank:



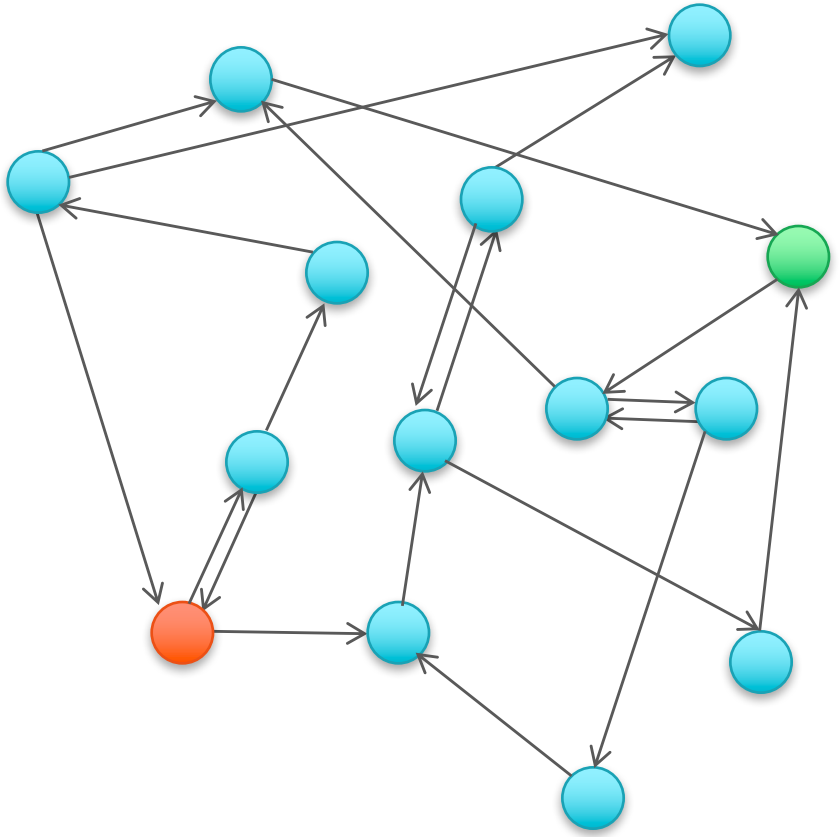
San Diego

PageRank

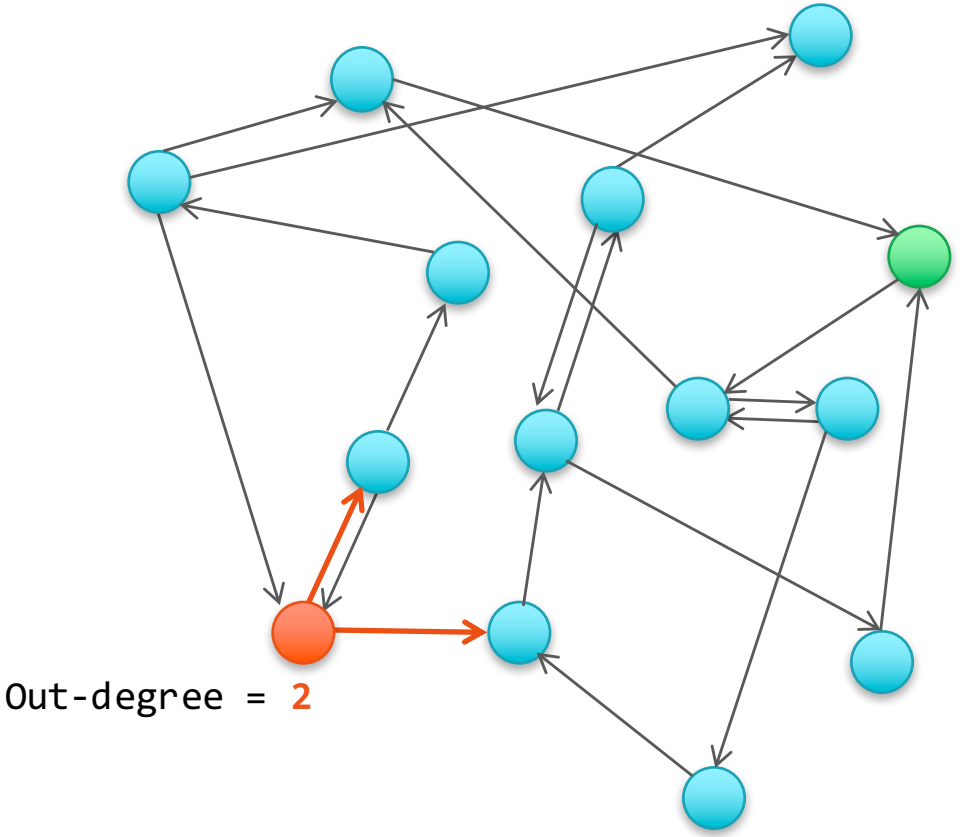
$$R[i] = 0.15 + \sum_{j \in \text{Nbrs}(i)} w_{ji} R[j]$$



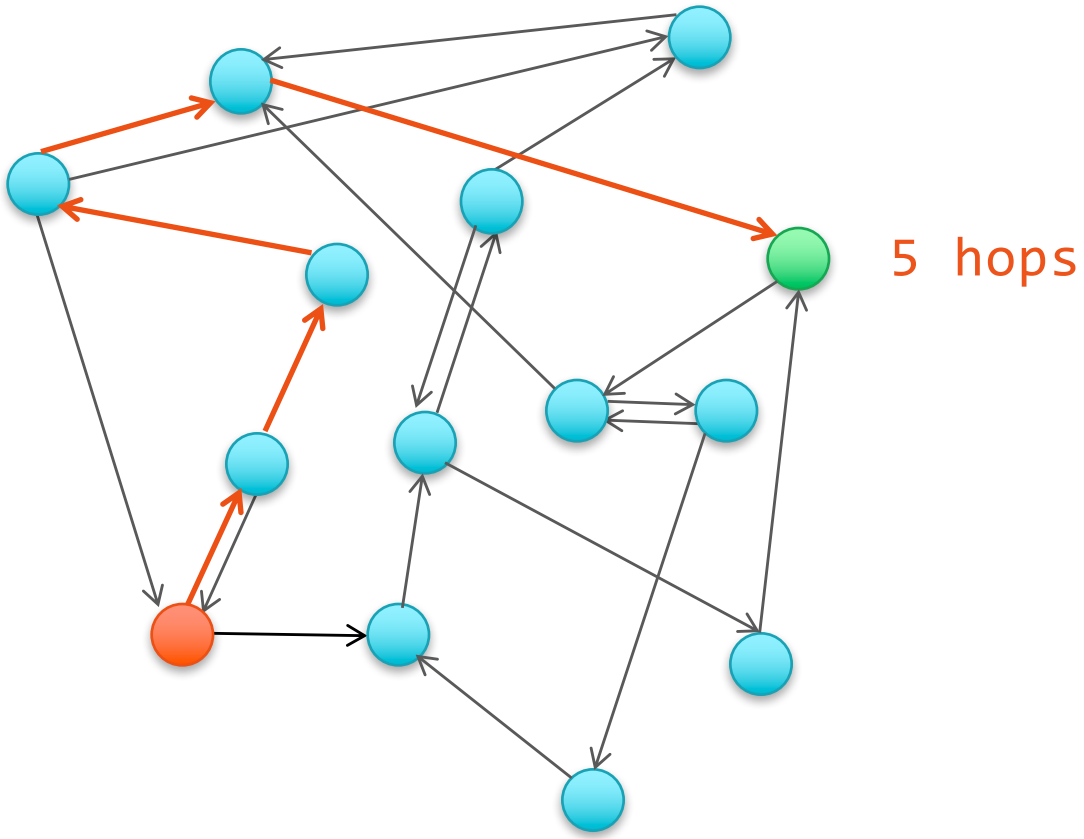
Shortest Path:



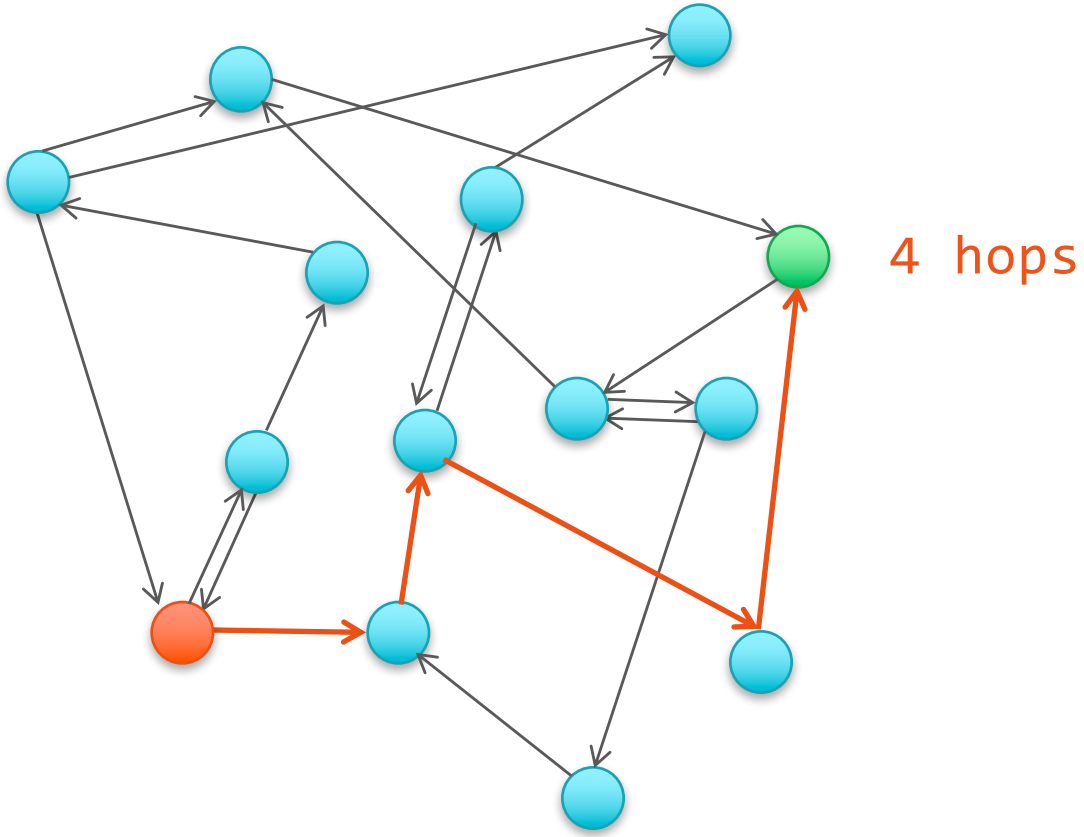
Shortest Path:



Shortest Path:

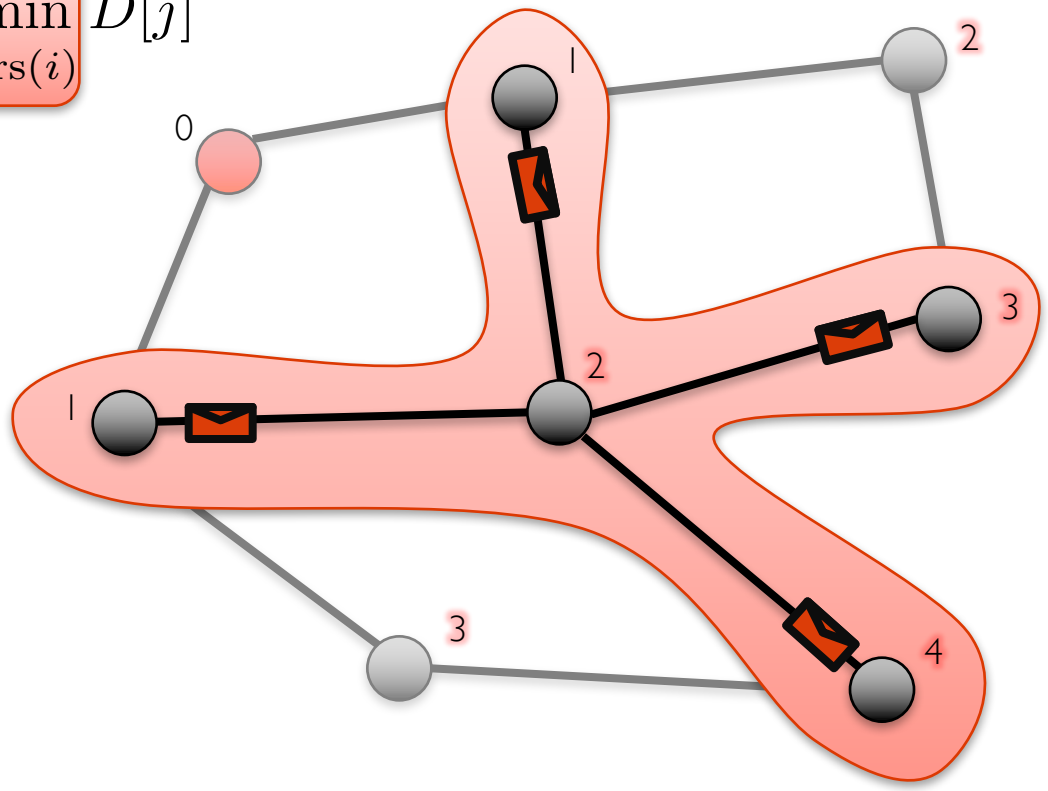


Shortest Path:

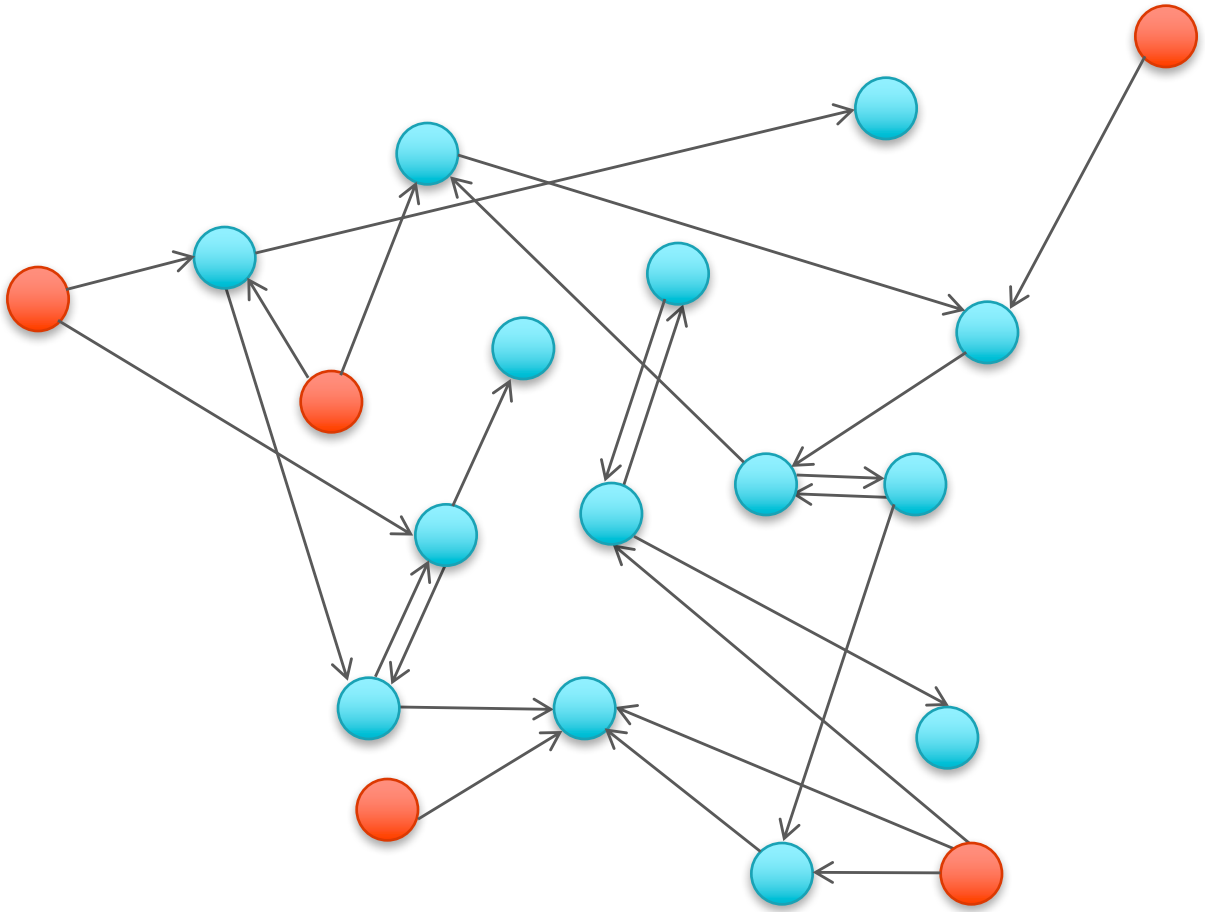


Single-Source Shortest Path

$$D[i] = 1 + \underset{j \in \mathrm{Nbrs}(i)}{\arg\min} D[j]$$



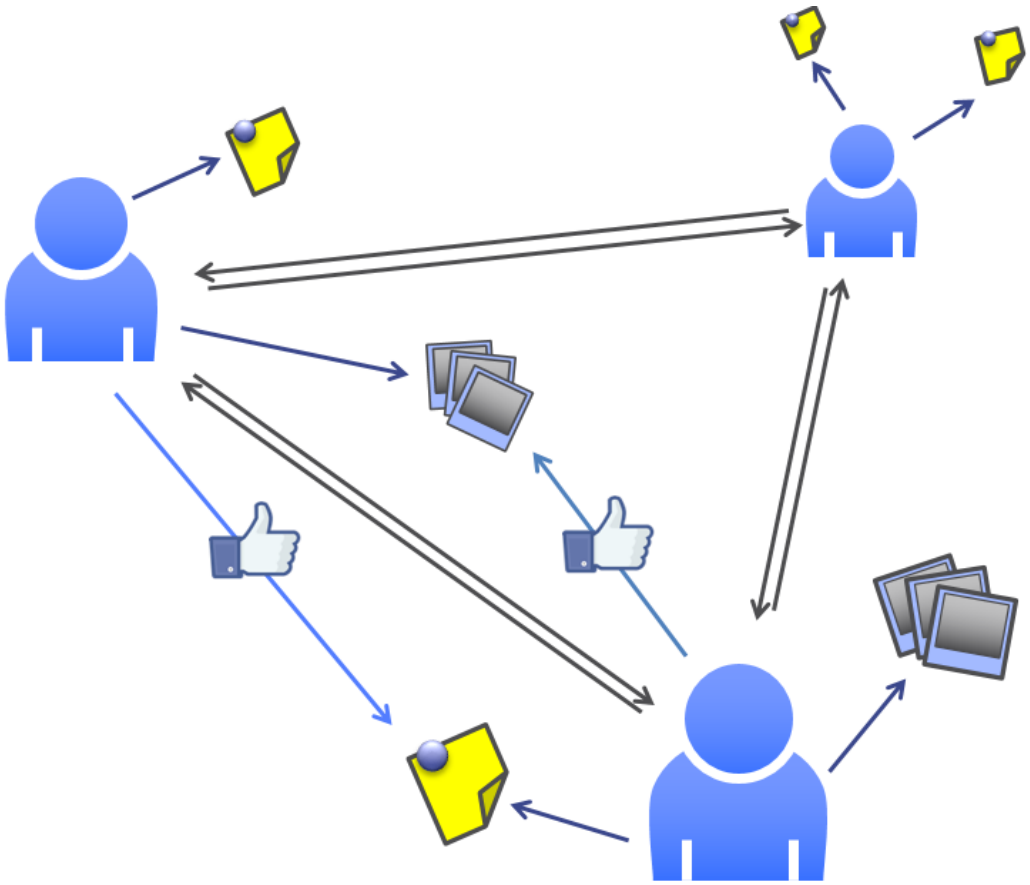
GraphX: 2 types of vertices



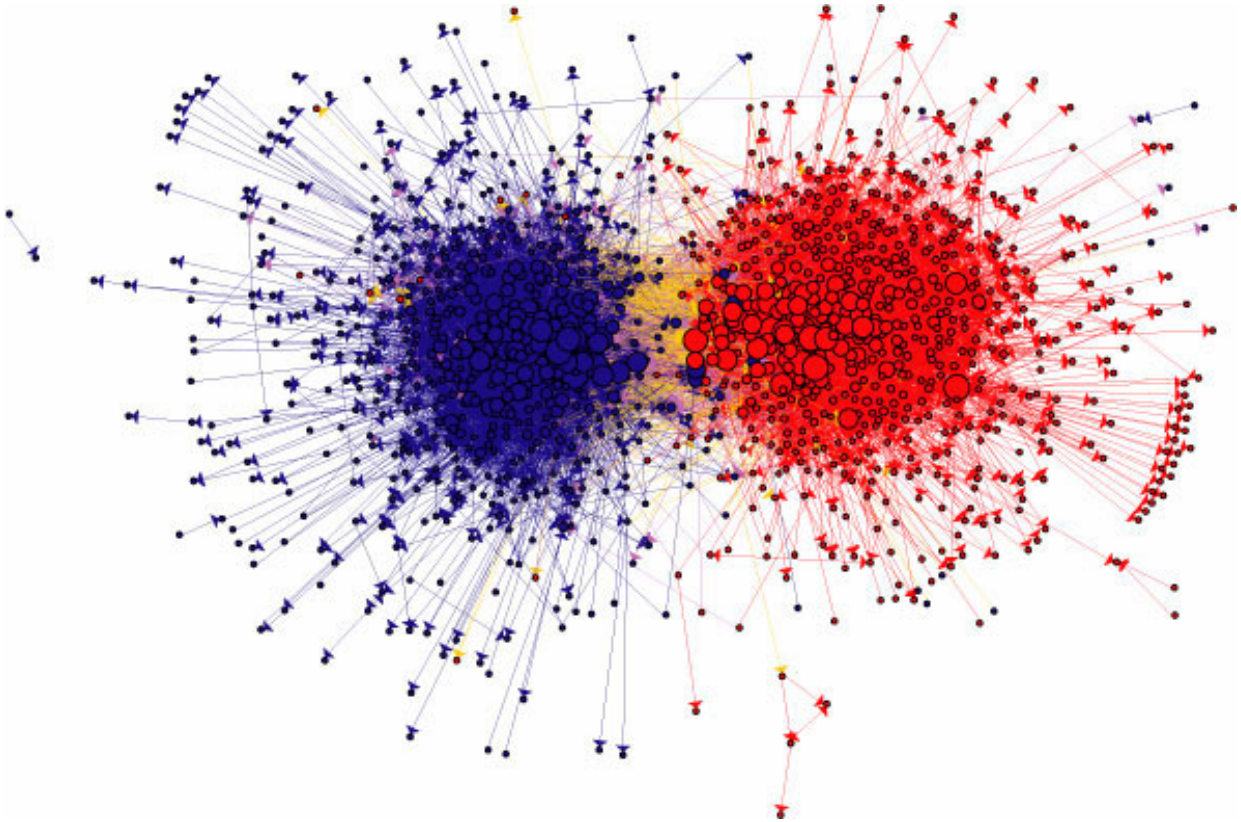
?

Graphs are everywhere...

Vertices: Users, Images, Posts
Edges: Social Relationships, Likes



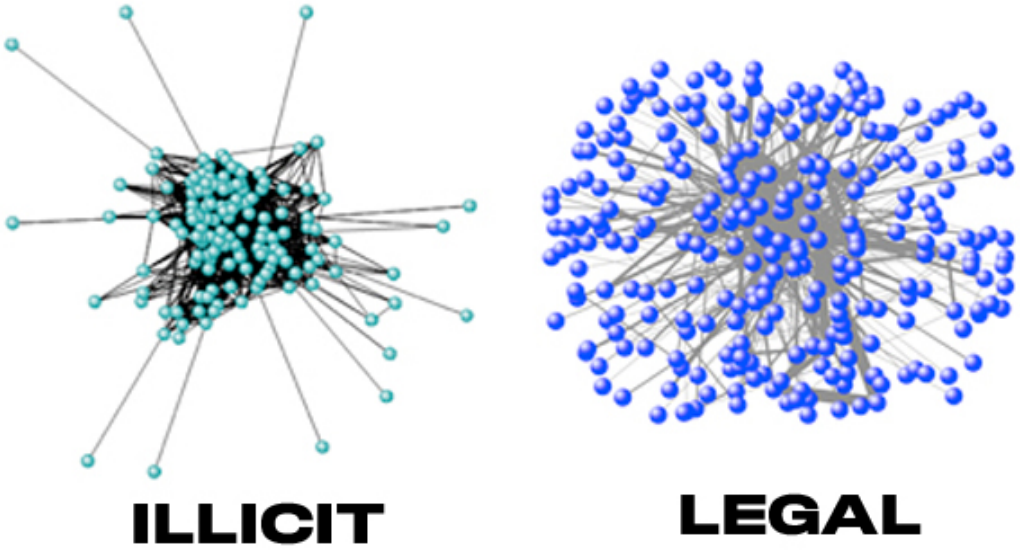
Vertices: Web Pages
Edges: Links



“The political blogosphere and the 2004 U.S. election: divided they blog.”

Source: Adamic and Glance, 2005.

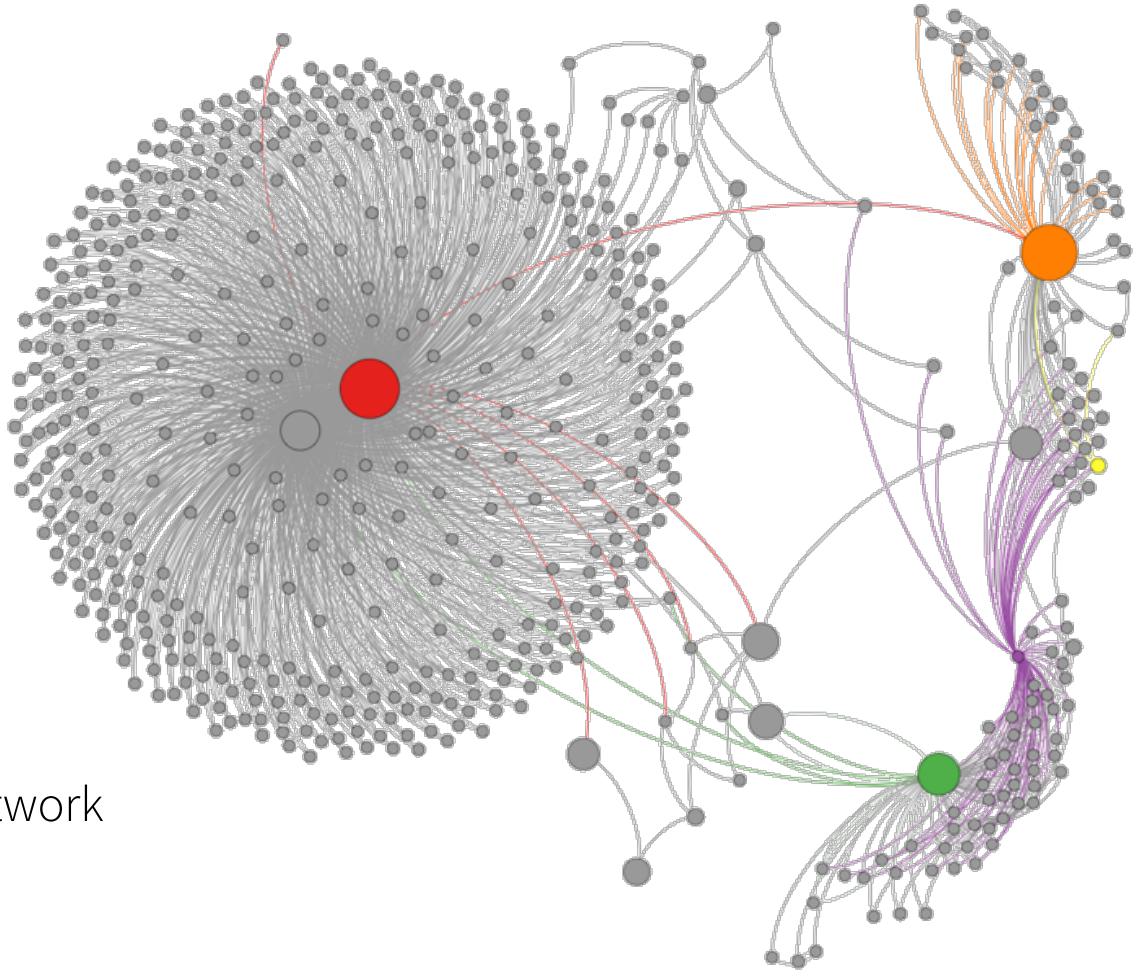
Vertices: Users
Directed Edges: Email from → to



Enron Email Graph

Source: https://www.sciencenews.org/article/information-flow-can-reveal-dirty-deeds

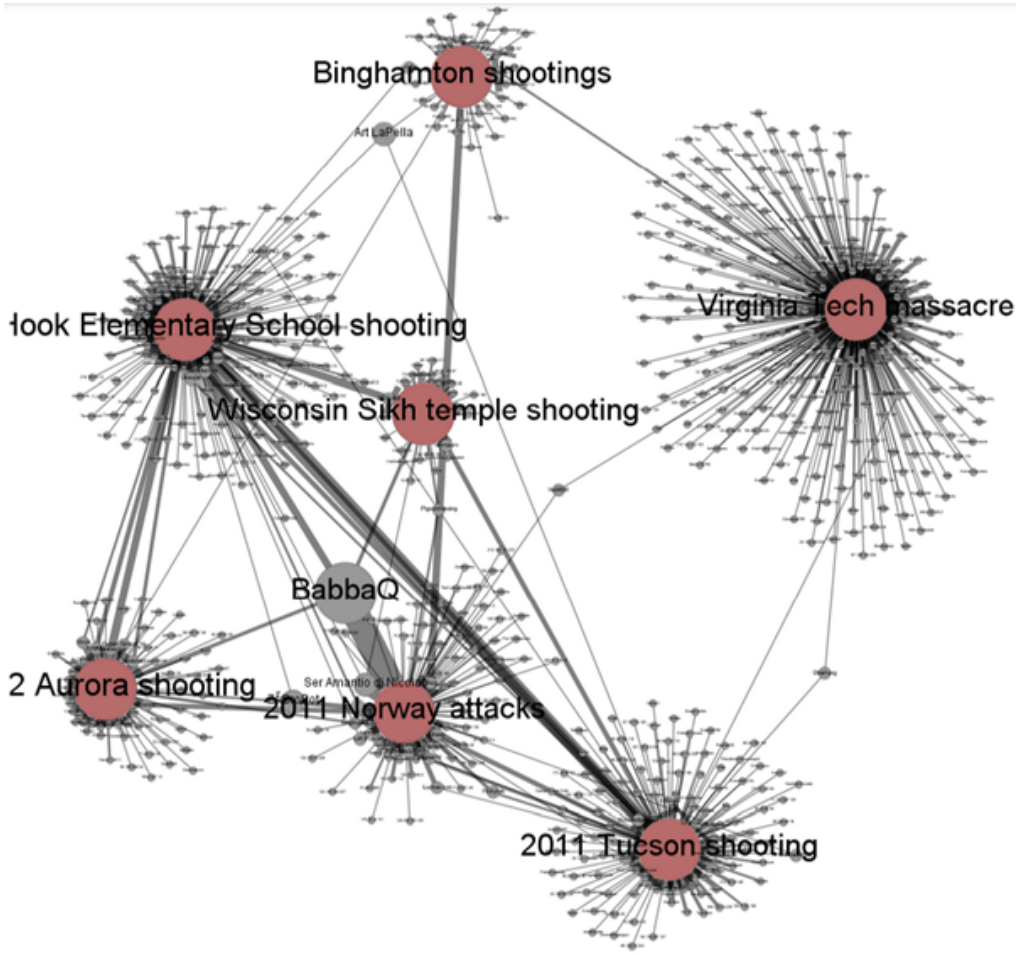
Vertices: People, Orgs
Edges: Exchange of currency



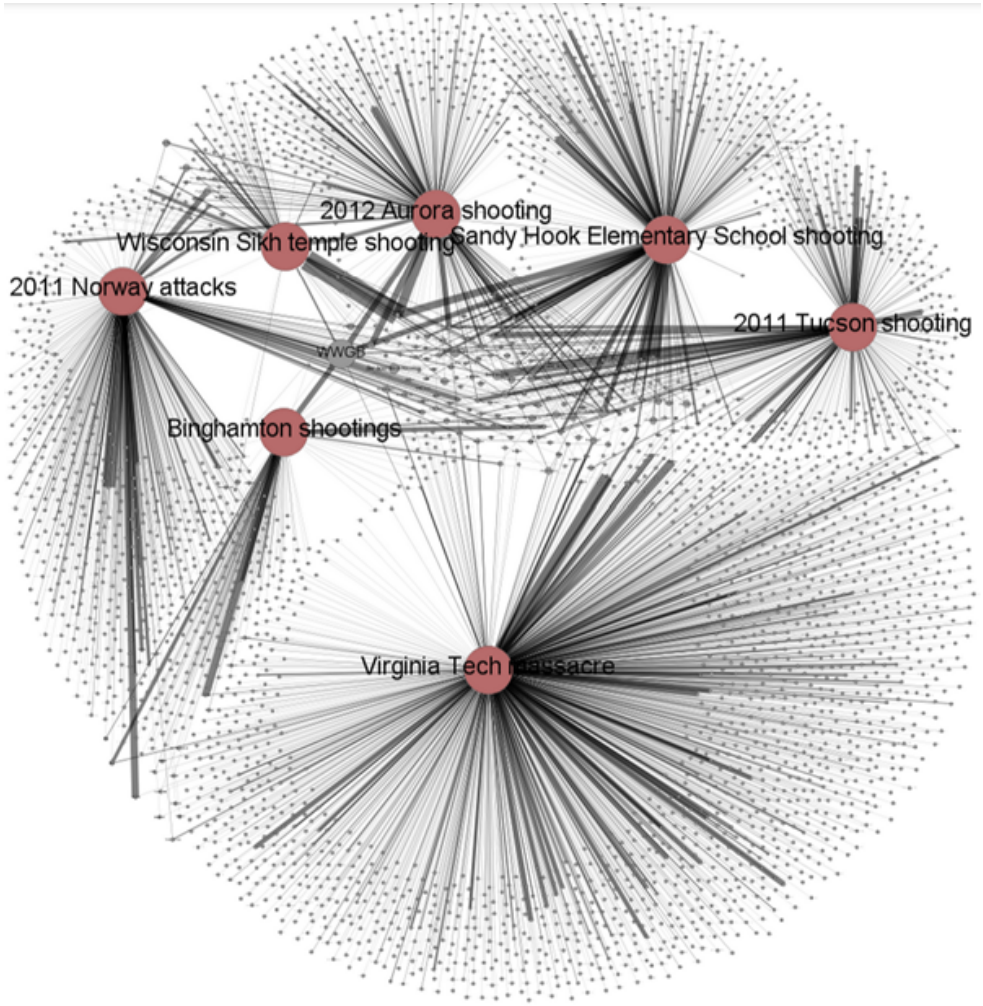
Case Study:
Bitcoin Theft Transaction Network

Source: http://anonymity-in-bitcoin.blogspot.com/2011/07/bitcoin-is-not-anonymous.html

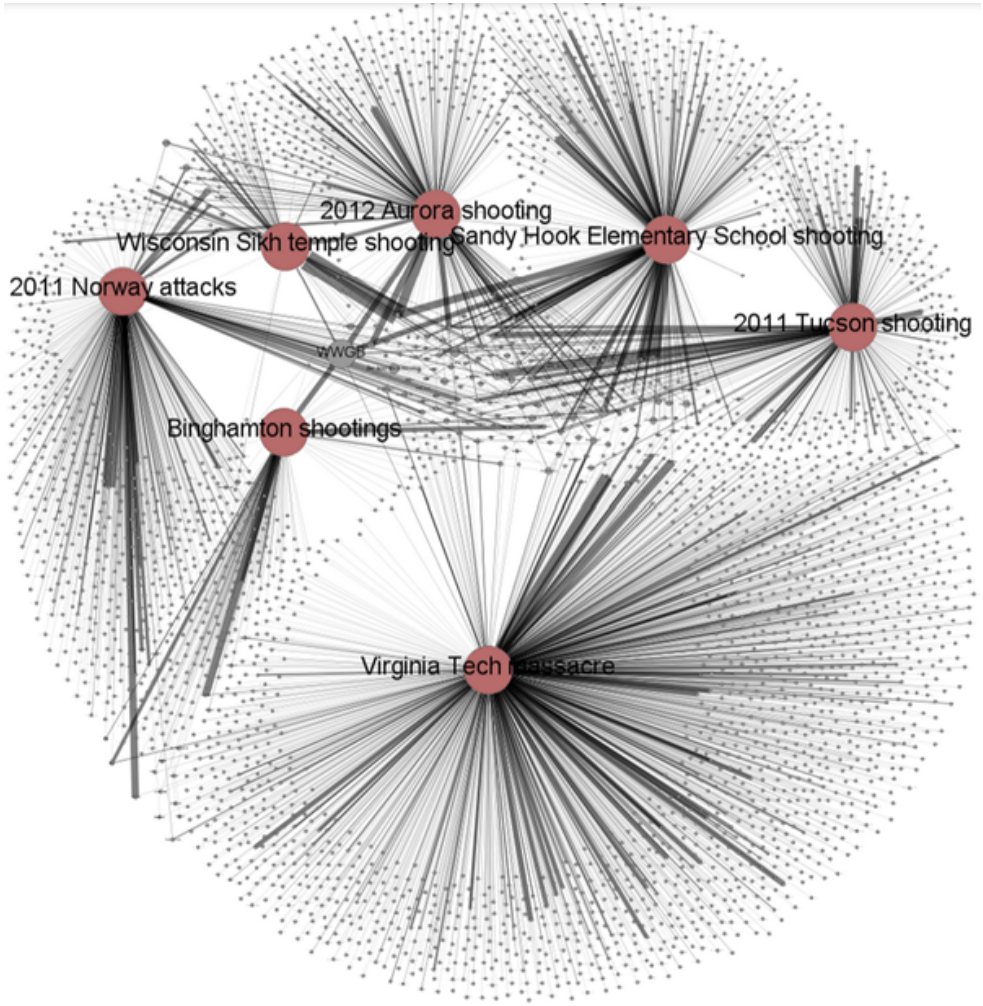
Vertices: Users, Items
Edges: Ratings



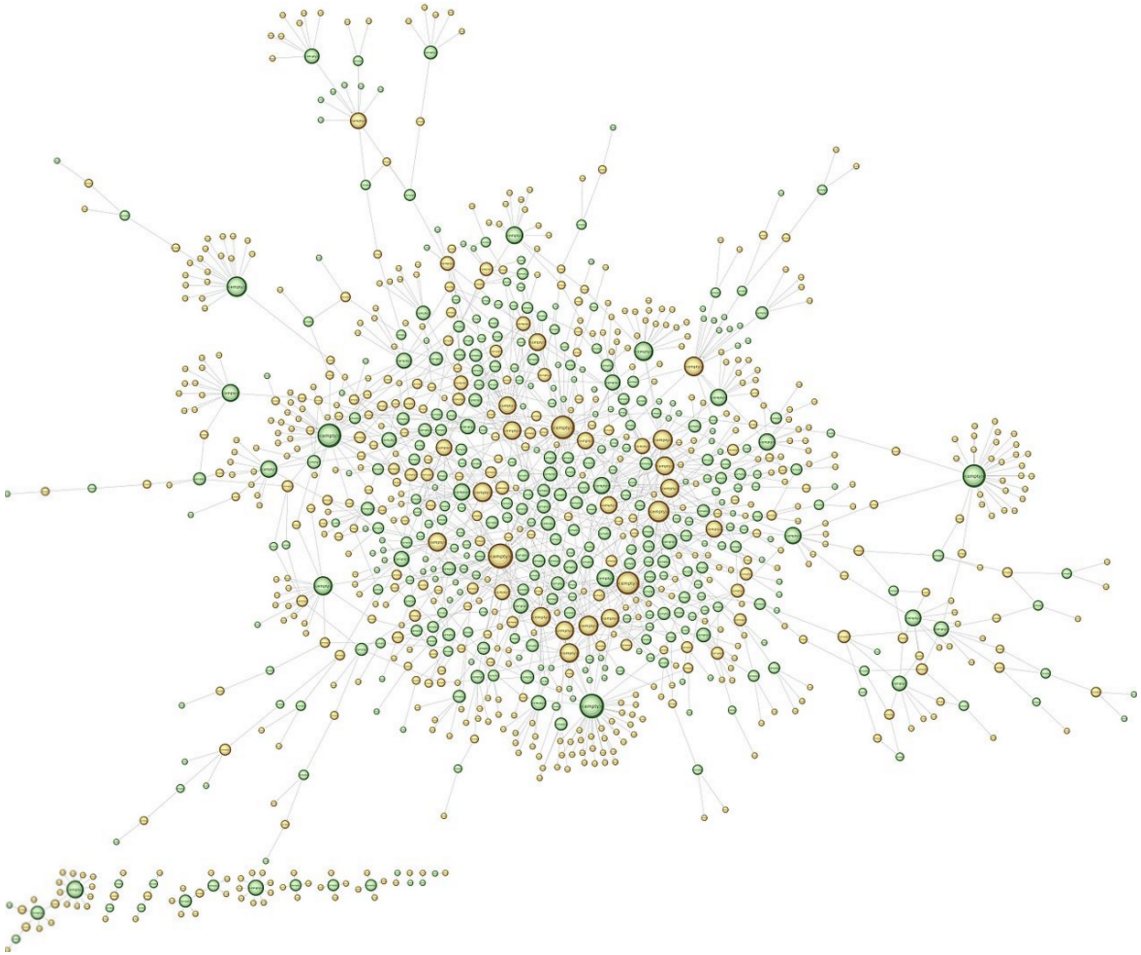
Source: Nieman Journalism Lab



Source: Nieman Journalism Lab

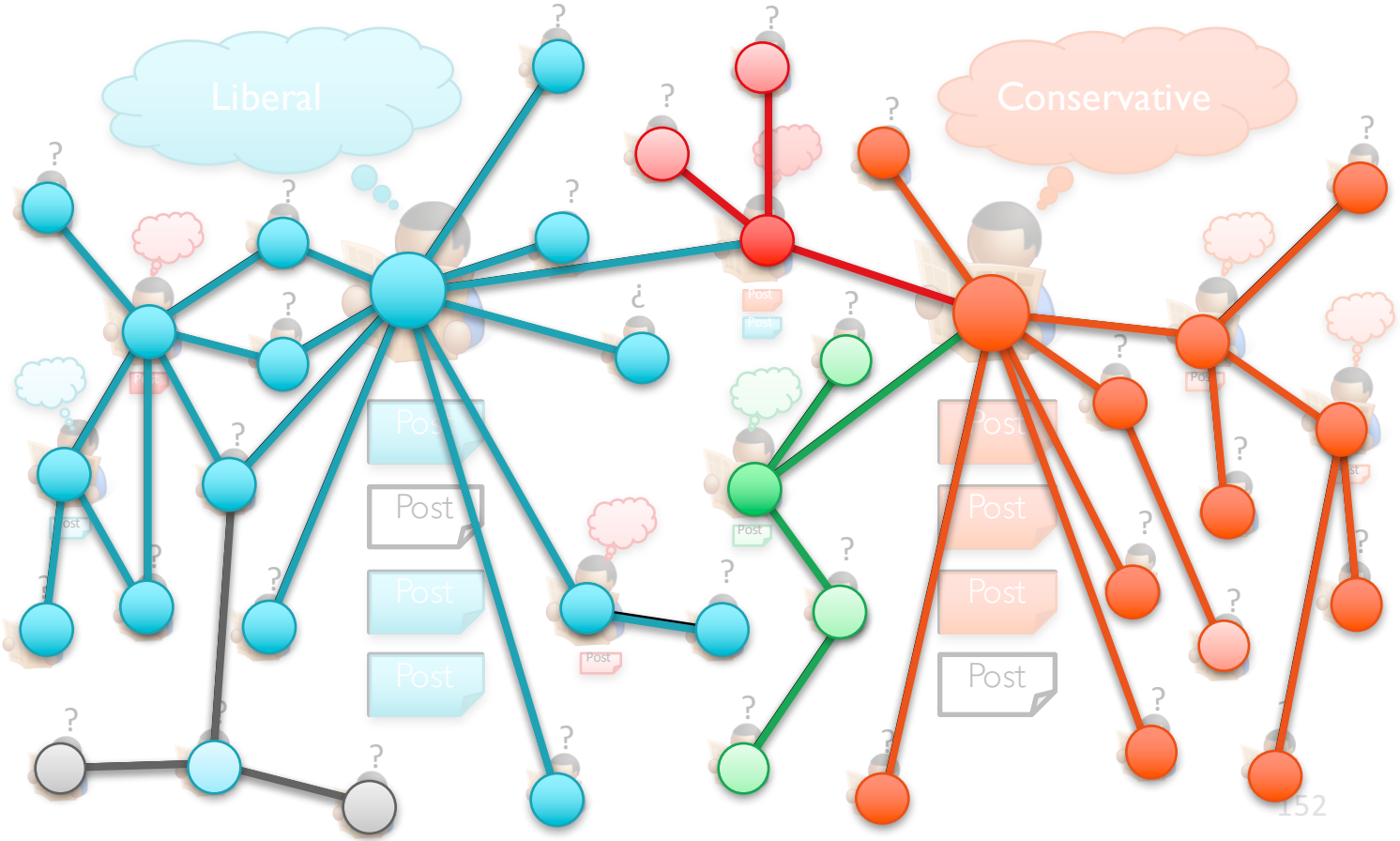


Source: Nieman Journalism Lab



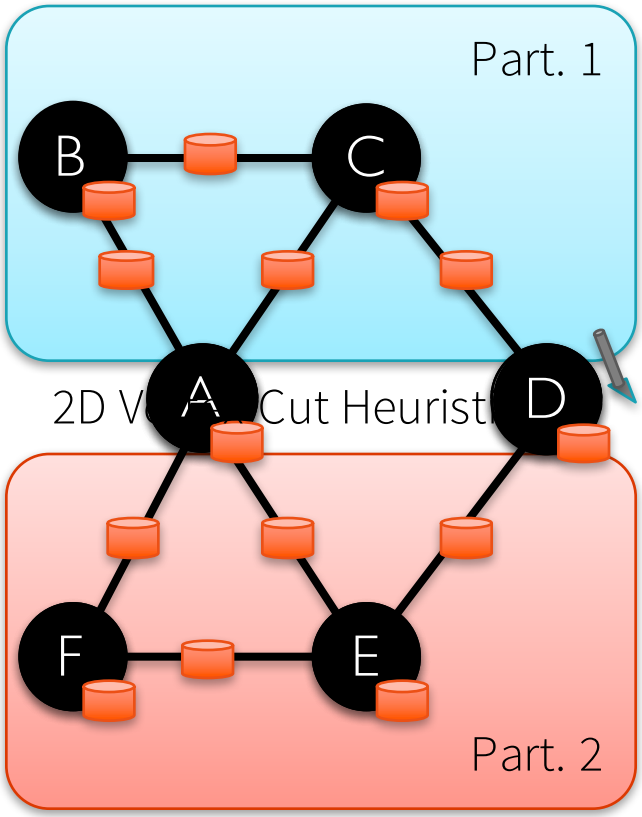
Source: Wikimedia Foundation

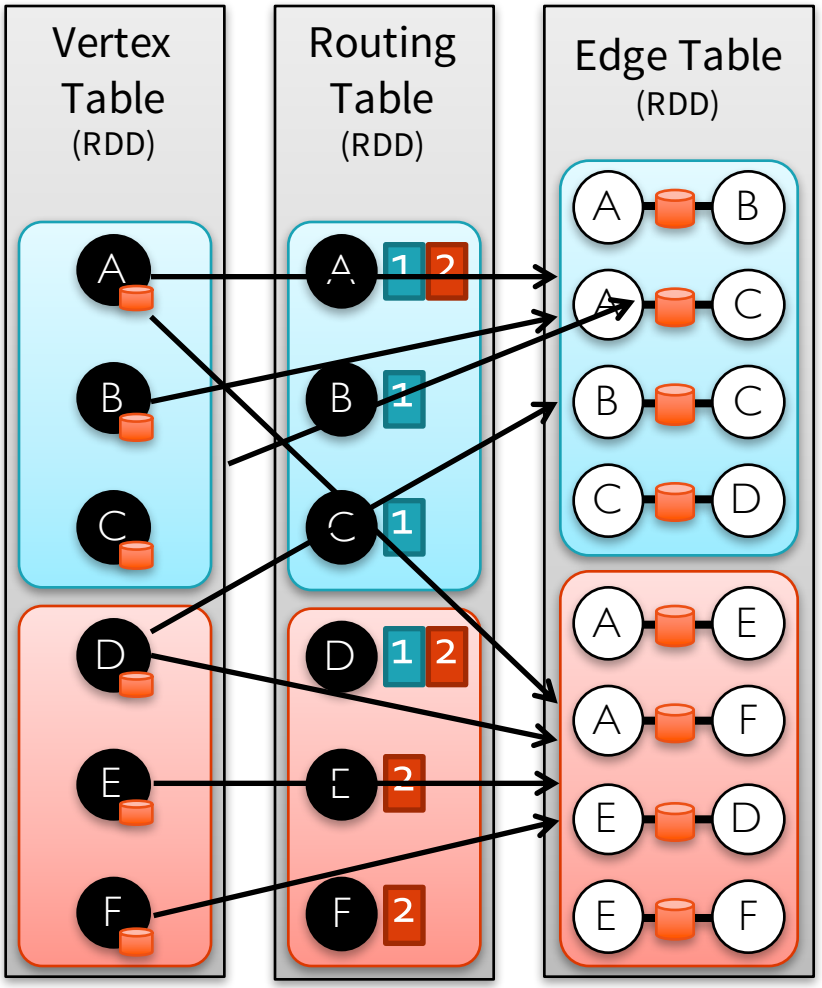
Belief Propagation: Predicting User Behavior



Distributed Graphs as Tables (RDDs)

Property Graph



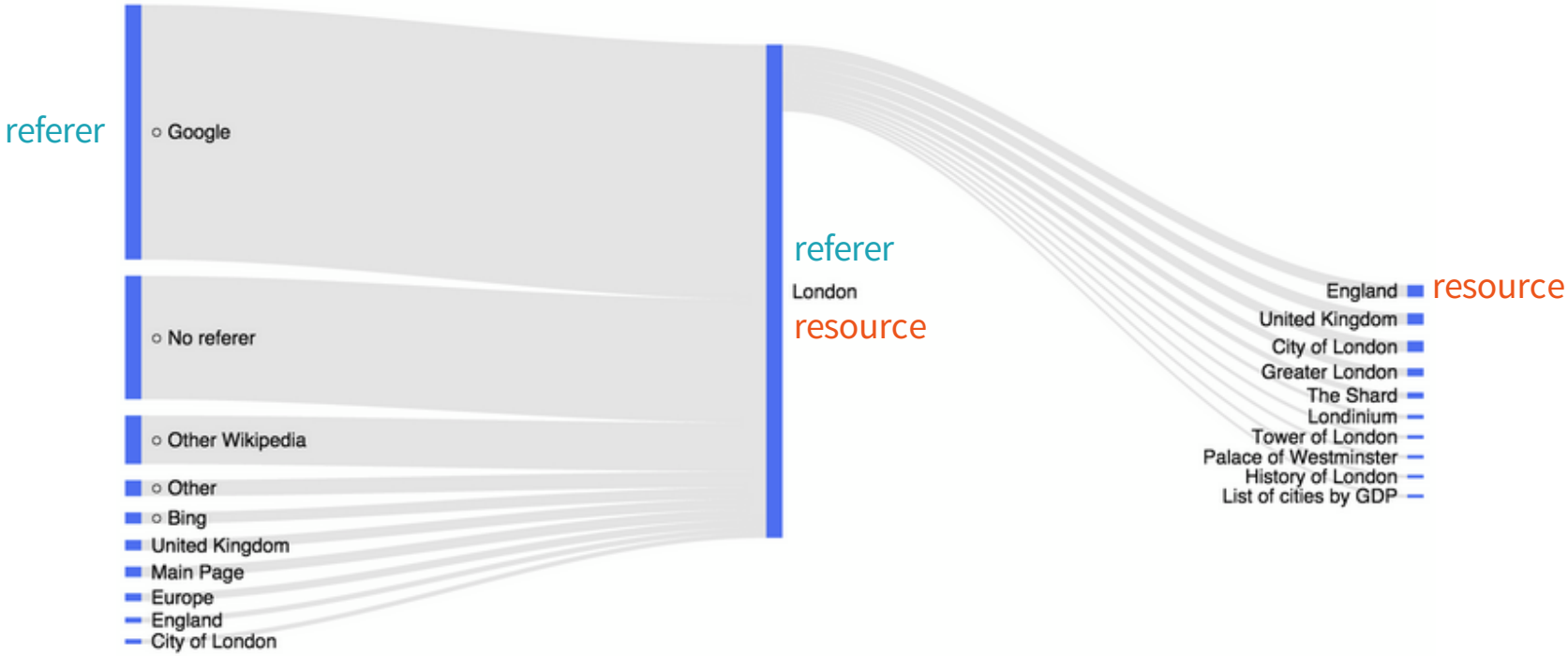


[revisited]

clickstream

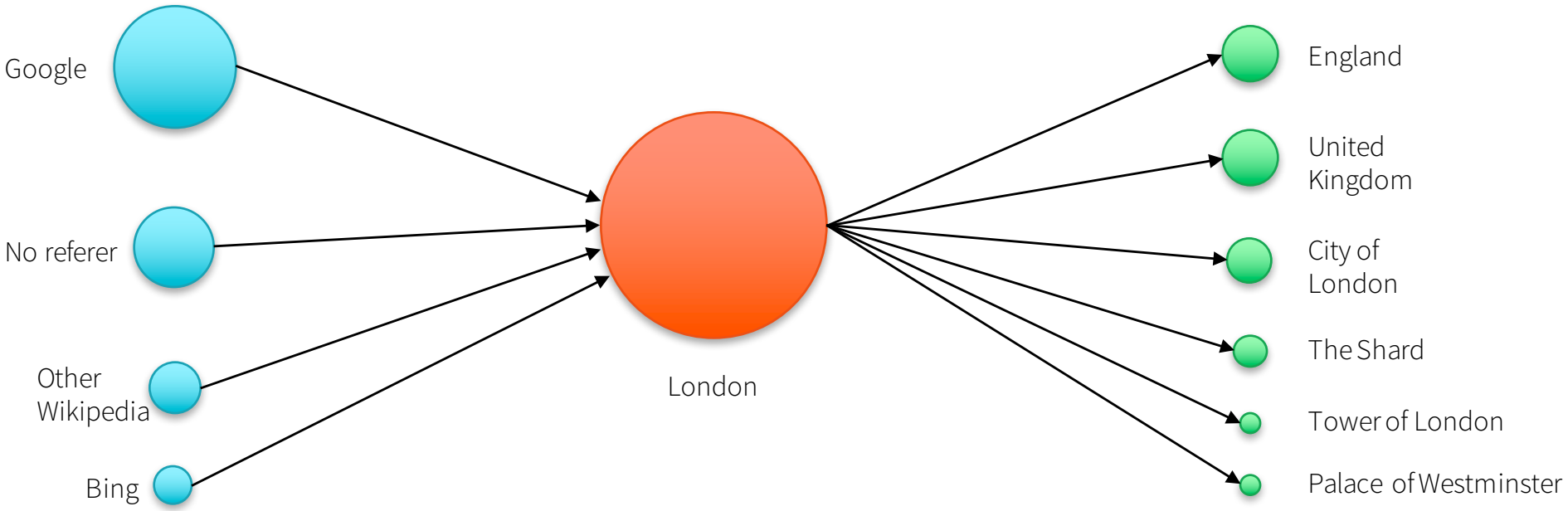
(1.2 GB)

databrickstraining



Wikipedia Clickstream

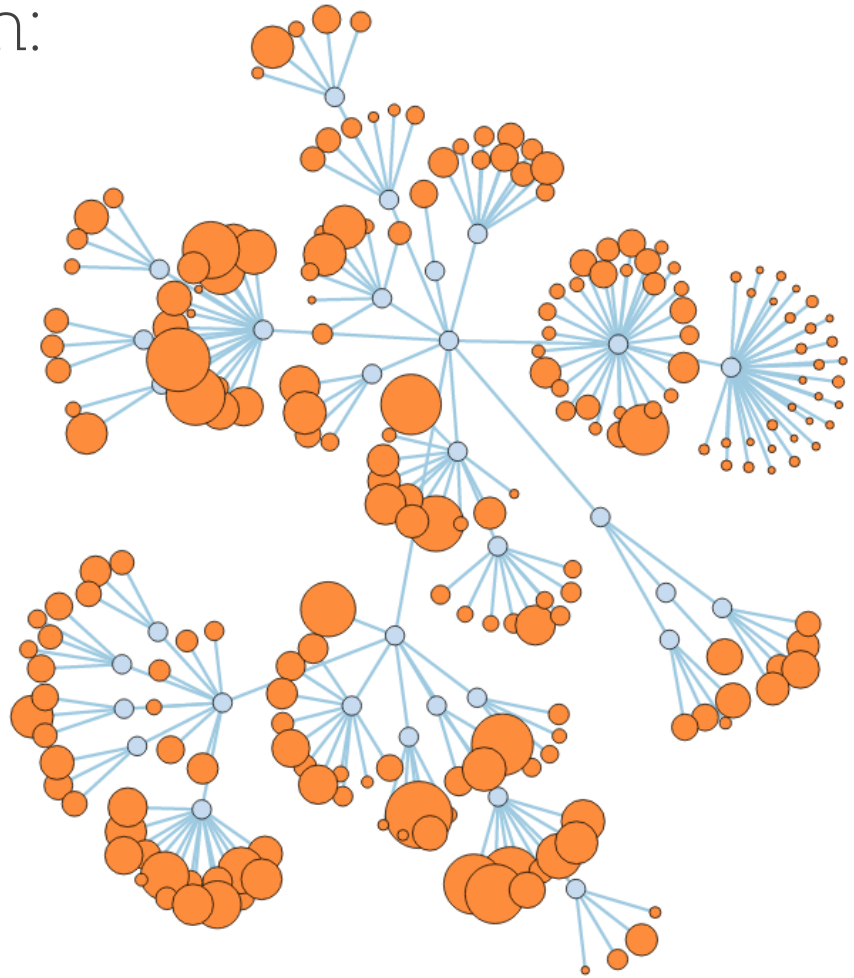
By Ellery Wulczyn
Data Scientist @ The Wikimedia Foundation



Wikipedia Clickstream

GraphX representation

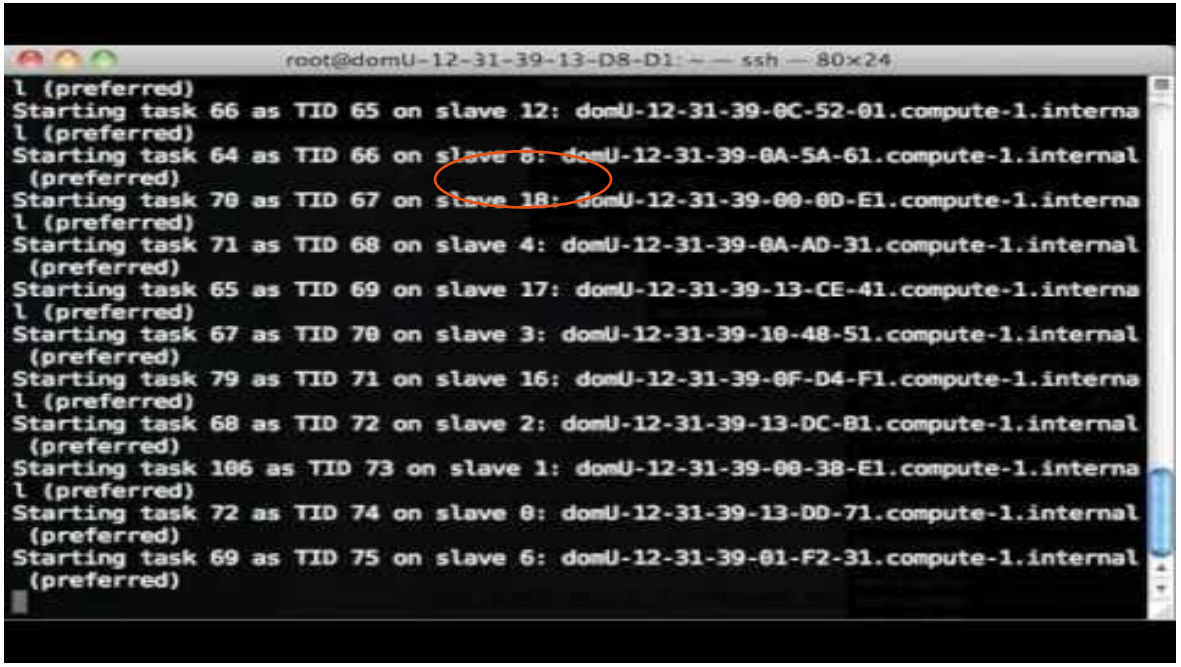
Subgraph from clickstream:




Spark
Machine Learning
databrickstraining

Spark Demo

June 2011



https://www.youtube.com/watch?v=Jw9yNTJI8iM

"We start by loading some fields from a tab-separated Wikipedia file into a distributed collection of Article objects, then we perform queries on it. Queries on the on-disk data take 20 seconds, but asking Spark to cache the Article objects in memory reduces the query latency to 1-2 seconds."

- Matei

w/ Joseph Bradley @ Facebook

Machine Learning for Humans

- Make practical ML scalable and easy
- Inspired by scikit-learn

Algorithm coverage

Classification

- Logistic regression w/ elastic net
- Naive Bayes
- Streaming logistic regression
- Linear SVMs
- Decision trees
- Random forests
- Gradient-boosted trees
- Multilayer perceptron
- One-vs-rest

Regression

- Least squares w/ elastic net
- Isotonic regression
- Decision trees
- Random forests
- Gradient-boosted trees
- Streaming linear methods

Recommendation

- Alternating Least Squares

Frequent itemsets

- FP-growth
- Prefix span

Feature extraction & selection

- Binarizer
- Bucketizer
- Chi-Squared selection
- CountVectorizer
- Discrete cosine transform
- ElementwiseProduct
- Hashing term frequency
- Inverse document frequency
- MinMaxScaler
- Ngram
- Normalizer
- One-Hot Encoder
- PCA
- PolynomialExpansion
- RFormula
- SQLTransformer
- Standard scaler
- StopWordsRemover
- StringIndexer
- Tokenizer
- StringIndexer
- VectorAssembler
- VectorIndexer
- VectorSlicer
- Word2Vec

Clustering

- Gaussian mixture models
- K-Means
- Streaming K-Means
- Latent Dirichlet Allocation
- Power Iteration Clustering

Statistics

- Pearson correlation
- Spearman correlation
- Online summarization
- Chi-squared test
- Kernel density estimation

Linear algebra

- Local dense & sparse vectors & matrices
- Distributed matrices
  - Block-partitioned matrix
  - Row matrix
  - Indexed row matrix
  - Coordinate matrix
- Matrix decompositions

Model import/export

Pipelines

List based on Spark 1.5

High-level functionality

Learning tasks

- Classification
- Regression
- Recommendation
- Clustering
- Frequent itemsets

Data utilities

- Feature extraction & selection
- Statistics
- Linear algebra

Workflow utilities

- Model import/export
- Pipelines
- DataFrames
- Cross validation

Spark MLlib Components

spark.mllib
(package)

- Original “lower-level” API
- Built on top of RDDs
- Will go to maintenance mode starting Spark 2.0

vs.

spark.ml
(package)

- Newer “higher-level” API for constructing workflows
- Built on top of DataFrames

Term Frequency – Inverse Document Frequency

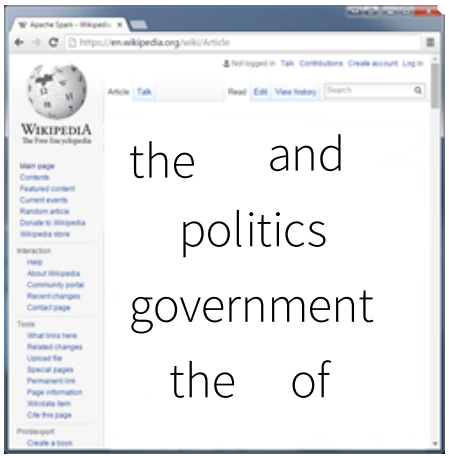
An information retrieval algorithm used to rank how important a word is to a collection of documents

---

TF: If a word appears frequently in a doc, it's important

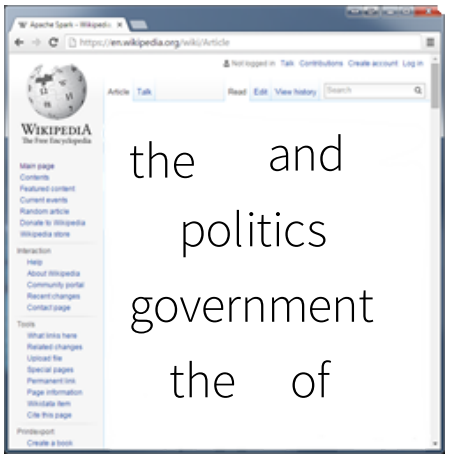
IDF: But if a word appears in many docs (the, and, of), the word is not meaningful, so lower its score

TF-IDF



$$\text{TF: } \frac{\text{\# of times word X appears in a document}}{\text{Total \# of words in the document}}$$

TF-IDF

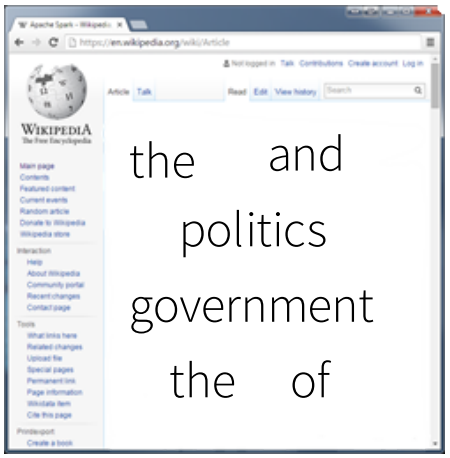


the: 2
and: 1
politics: 1
government: 1
of: 1

of: 2
congress: 2
and: 1
the: 1
politics: 1

sports: 2
the: 1
and: 1
games: 1
olympics: 1

TF–IDF

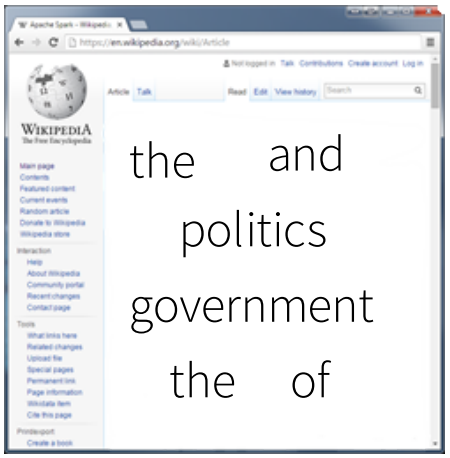


the: 2 / 6 = 0.3
and: 1 / 6 = 0.16
politics: 1 / 6 = 0.16
government: 1 / 6 = 0.16
of: 1 / 6 = 0.16

of: 2 / 7 = 0.28
congress: 2 / 7 = 0.28
and: 1 / 7 = 0.14
the: 1 / 7 = 0.14
politics: 1 / 7 = 0.14

sports: 2 / 6 = 0.3
the: 1 / 6 = 0.16
and: 1 / 6 = 0.16
games: 1 / 6 = 0.16
olympics: 1 / 6 = 0.16

TF-IDF

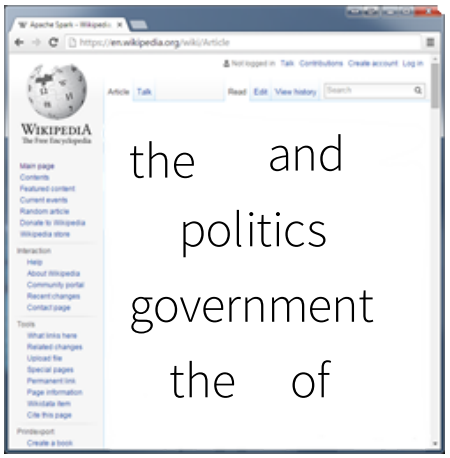


the: 2 / 6 = 0.3
and: 1 / 6 = 0.16
politics: 1 / 6 = 0.16
government: 1 / 6 = 0.16
of: 1 / 6 = 0.16

of: 2 / 7 = 0.28
congress: 2 / 7 = 0.28
and: 1 / 7 = 0.14
the: 1 / 7 = 0.14
politics: 1 / 7 = 0.14

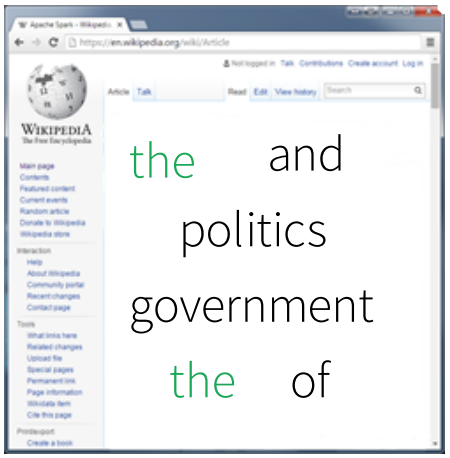
sports: 2 / 6 = 0.3
the: 1 / 6 = 0.16
and: 1 / 6 = 0.16
games: 1 / 6 = 0.16
olympics: 1 / 6 = 0.16

TF-IDF



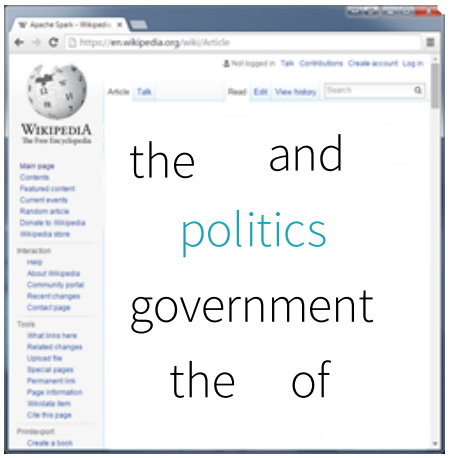
$$\text{IDF}: \quad \log \left[\frac{\text{Total \# of documents}}{\text{\# of documents with word X in it}} \right]$$

TF-IDF



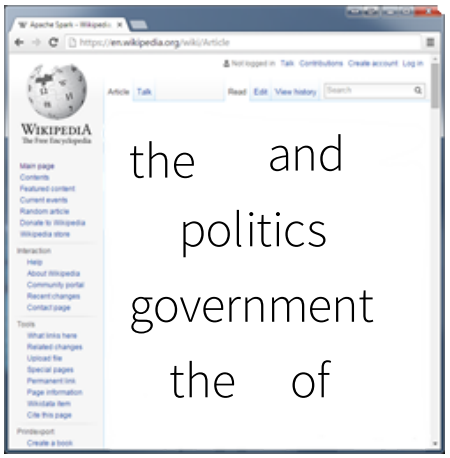
IDF: $\log\left[\frac{3}{3}\right] = 0$

TF-IDF



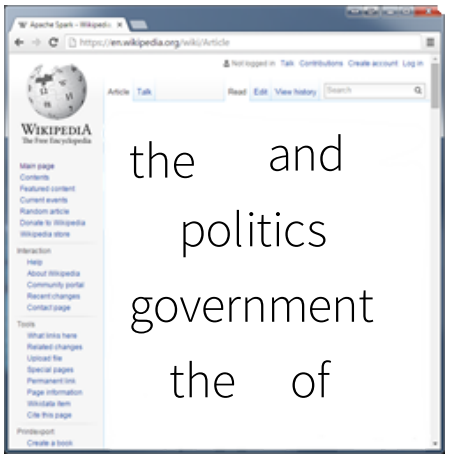
IDF: $\log\left[\frac{3}{2}\right] = 0.18$

TF-IDF



$$\text{IDF}: \quad \log\left[\frac{3}{1}\right] = 0.48$$

TF–IDF



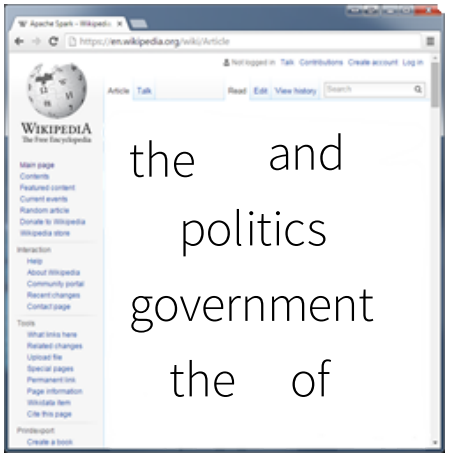
government: 0.16 * 0.48 = 0.08
politics: 0.16 * 0.18 = 0.03
of: 0.16 * 0.18 = 0.03
the: 0.3 * 0 = 0
and: 0.16 * 0 = 0

congress: 0.28 * 0.48 = 0.08
of: 0.28 * 0.18 = 0.05
politics: 0.14 * 0.18 = 0.03
the: 0.14 * 0 = 0
and: 0.14 * 0 = 0

sports: 0.3 * 0.48 = 0.144
olympics: 0.16 * 0.48 = 0.08
games: 0.16 * 0.48 = 0.08
the: 0.16 * 0 = 0
and: 0.16 * 0 = 0

TF-IDF

sports Search



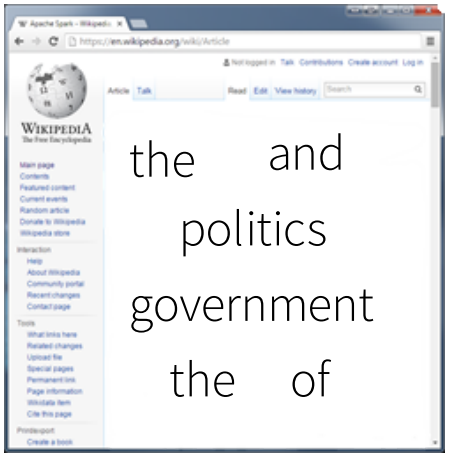
government: 0.16 * 0.48 = 0.08
politics: 0.16 * 0.18 = 0.03
of: 0.16 * 0.18 = 0.03

congress: 0.28 * 0.48 = 0.08
of: 0.28 * 0.18 = 0.05
politics: 0.14 * 0.18 = 0.03

sports: 0.3 * 0.48 = 0.144
olympics: 0.16 * 0.48 = 0.08
games: 0.16 * 0.48 = 0.08

TF-IDF

politics Search



government: 0.16 * 0.48 = 0.08
politics: 0.16 * 0.18 = 0.03
of: 0.16 * 0.18 = 0.03

congress: 0.28 * 0.48 = 0.08
of: 0.28 * 0.18 = 0.05
politics: 0.14 * 0.18 = 0.03

sports: 0.3 * 0.48 = 0.144
olympics: 0.16 * 0.48 = 0.08
games: 0.16 * 0.48 = 0.08

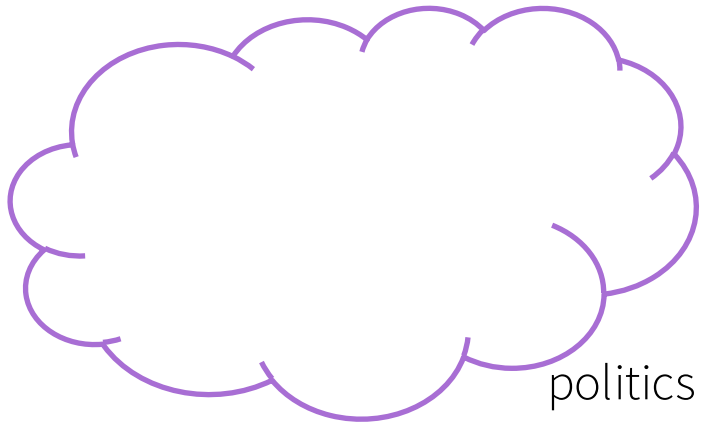
K-means clustering

An unsupervised ML algorithm for classifying items into k groups.

---

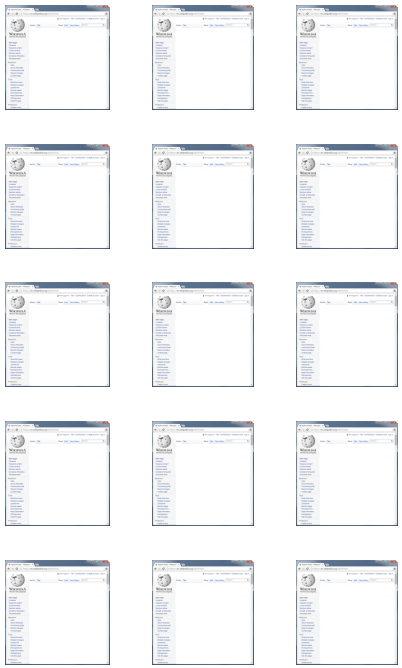
k: The # of pre-chosen groups

K–means clustering

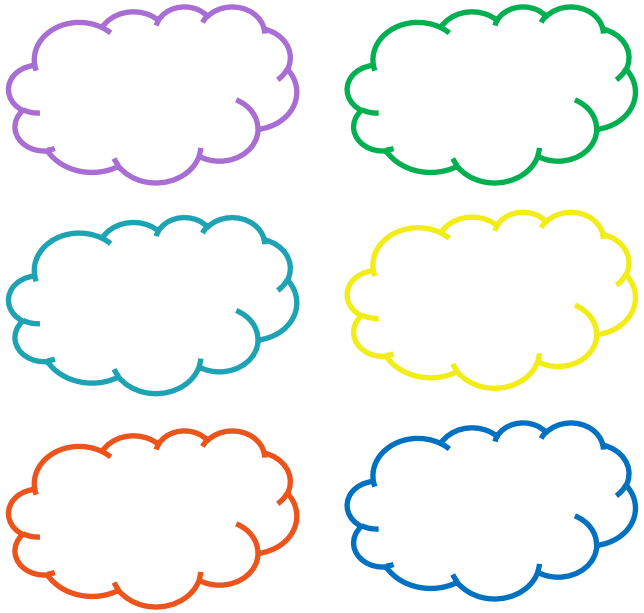


K–means clustering

K-means clustering

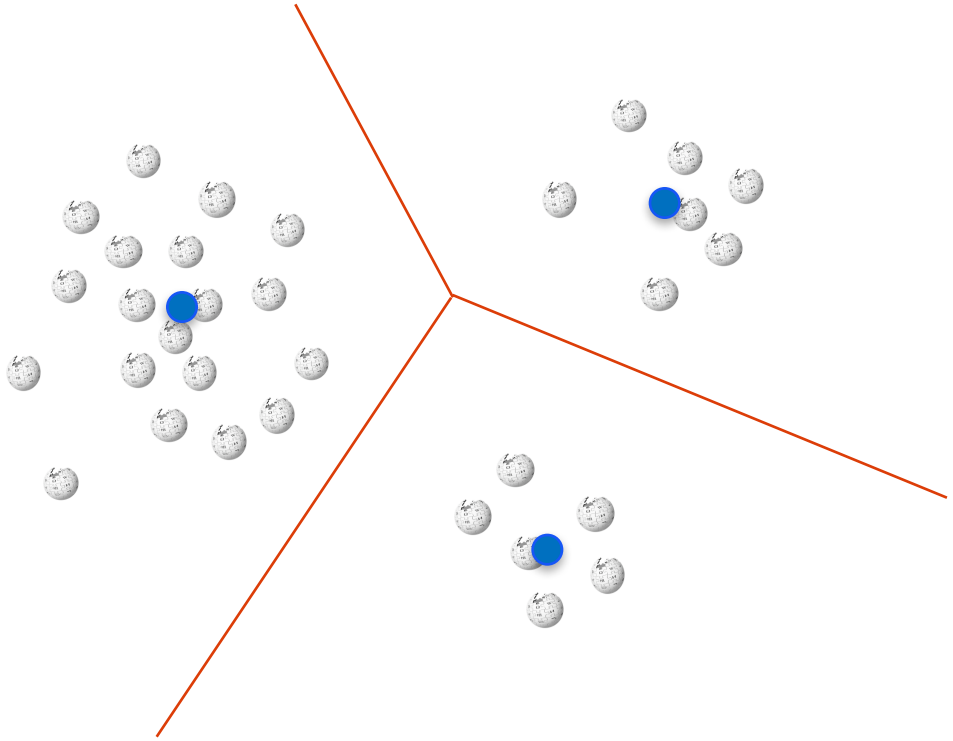


5 million articles



100 clusters

K-means clustering

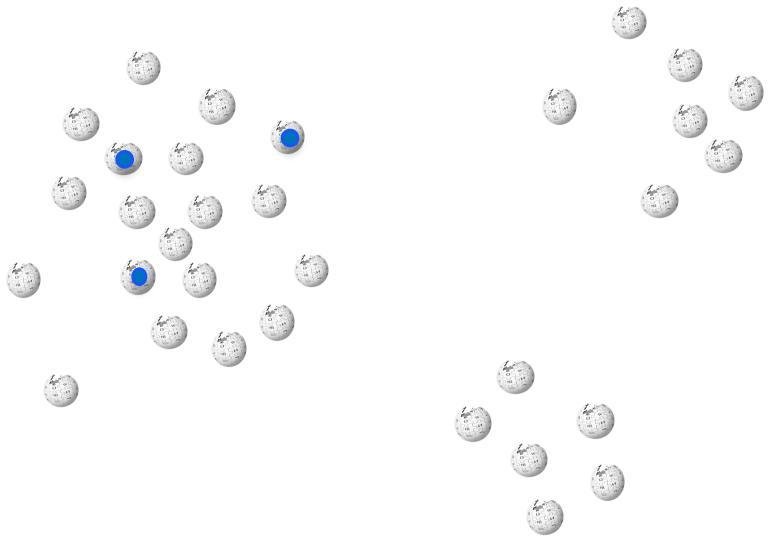


black dots: Wikipedia Articles (items)

red lines: Cluster Partitions

blue dots: Cluster Centroids

K-means clustering: Initialization



Approach #1:

Randomly select centroids to start

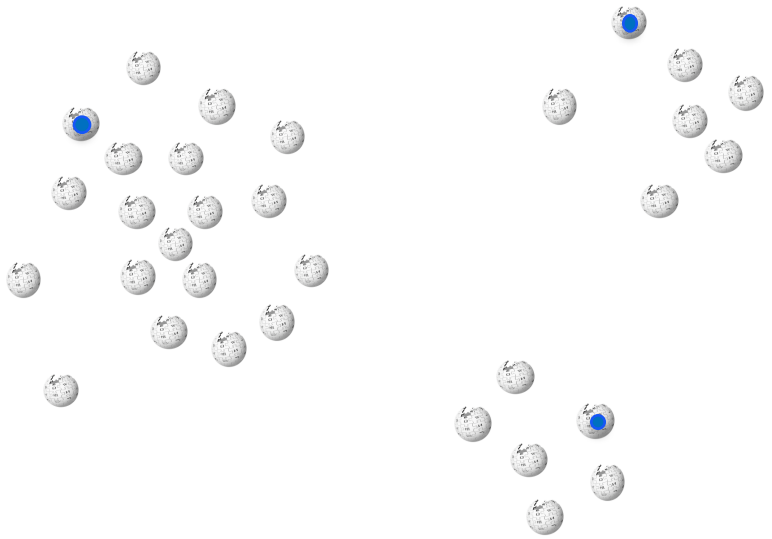
Pro:

Does not require full iteration

Con:

If a cluster is large compared to neighbors, initial centroids will likely end up in large cluster.

K-means clustering: Initialization



Approach #2:

k-means++

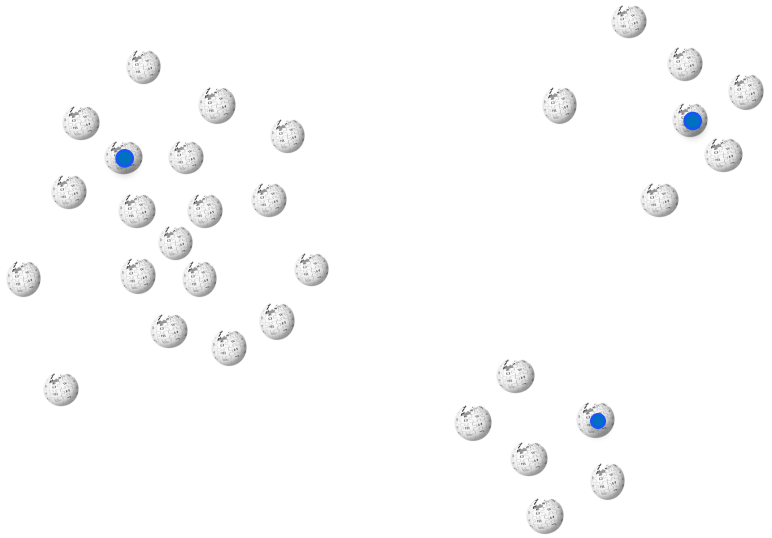
Pro:

Avoids the poor clusterings found by standard algorithm

Con:

Needs k passes over the data, so does not scale well to large data sets

K-means clustering: Initialization



Approach #3:

k-means||

Pro:

Also avoids the poor clusterings found by standard algorithm

Can select multiple candidate centroids in first iteration

K-means clustering: Initialization

(expert-only) Parameters

A list of advanced, expert-only (hyper-)parameter keys this algorithm can take. Users can set and get the parameter values through setters and getters, respectively.

final val **initMode**: Param[String]

Param for the initialization algorithm. This can be either "random" to choose random points as initial cluster centers, or "k-means||" to use a parallel variant of k-means++ (Bahmani et al., Scalable K-Means++, VLDB 2012). Default: k-means||.

Definition Classes KMeansParams

Annotations @Since("1.5.0")

final val **initSteps**: IntParam

Param for the number of steps for the k-means|| initialization mode.

Source: https://spark.apache.org/docs/1.6.0/api/scala/index.html#org.apache.spark.ml.clustering.KMeans

K-means clustering

k: 2

Article	Word X	Word Y
A	1.0	1.0
B	1.5	2.0
C	3.0	4.0
D	5.0	7.0
E	3.5	5.0
F	4.5	5.0
G	3.5	4.5

Source: http://mnemstudio.org

K-means clustering

k: 2

	Article	Mean Vector *(centroid)*
Group 1	A	(1.0, 1.0)
Group 2	D	(5.0, 7.0)

K-means clustering

k: 2

	Article	Mean Vector *(centroid)*
Cluster 1	A, B, C	(1.8, 2.3)
Cluster 2	D, E, F, G	(4.1, 5.4)

K-means clustering

k: 2

	Article	Distance to mean *(centroid)* of Cluster 1	Distance to mean *(centroid)* of Cluster 2
Cluster 1	A	1.5	5.4
	B	0.4	4.3
Cluster 2	C	2.1	1.8
	D	5.7	1.8
	E	3.2	0.7
	F	3.8	0.6
	G	2.8	1.1

K-means clustering

k: 2

	Article	Mean Vector *(centroid)*
Cluster 1	A, B	(1.3, 1.5)
Cluster 2	C, D, E, F, G	(3.9, 5.1)

K-means clustering

```scala
/**
 * :: Experimental ::
 * K-means clustering with support for k-means|| initialization proposed by Bahmani et al.
 *
 * @see [[http://dx.doi.org/10.14778/2180912.2180915 Bahmani et al., Scalable k-means++.]]
 */
@Since("1.5.0")
@Experimental
class KMeans @Since("1.5.0") (
    @Since("1.5.0") override val uid: String)
  extends Estimator[KMeansModel] with KMeansParams with DefaultParamsWritable {

  setDefault(
    k -> 2,
    maxIter -> 20,
    initMode -> MLlibKMeans.K_MEANS_PARALLEL,
    initSteps -> 5,
    tol -> 1e-4)

  @Since("1.5.0")
  override def copy(extra: ParamMap): KMeans = defaultCopy(extra)

  @Since("1.5.0")
  def this() = this(Identifiable.randomUID("kmeans"))

  /** @group setParam */
  @Since("1.5.0")
  def setFeaturesCol(value: String): this.type = set(featuresCol, value)
```

Source: https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/ml/clustering/KMeans.scala#L216

K-means clustering

k: 2

	Cluster 1		Cluster 2	
	Dist to (1, 1)		Dist to (5, 7)	
A		(1, 1)		
B			D	
C			D	
D			D, E	(5, 7)
			D, E, F	
mean		(1.1, 1.2)		(1, 2)

Spark ML Pipelines

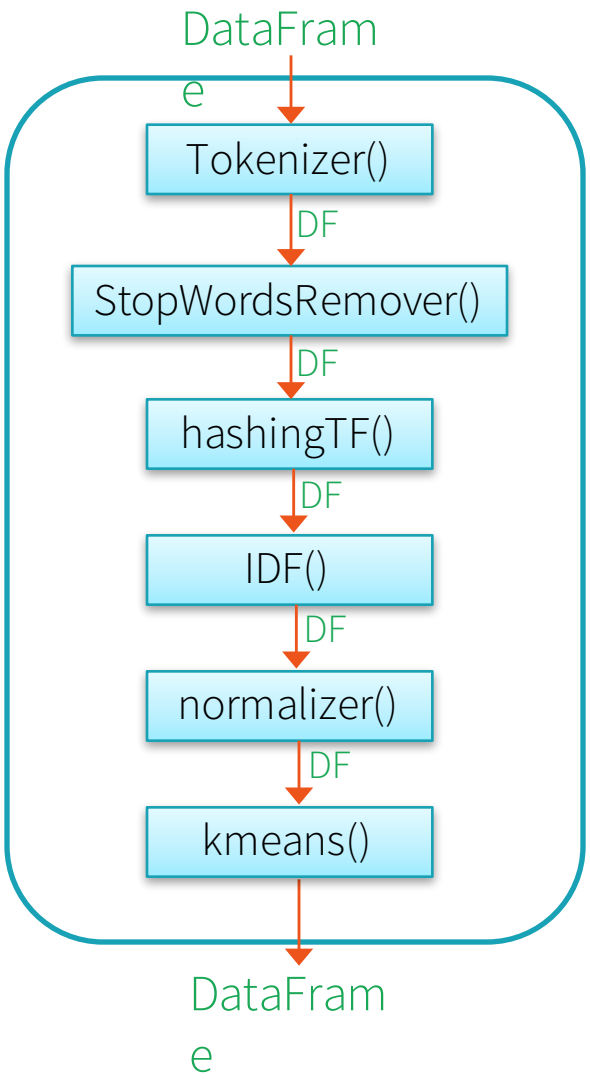
DataFrame: uses DF from Spark SQL as a ML dataset. Different columns can store text, feature vectors, true labels and predictions

Transformer: an algorithm which can transform one DataFrame into another DataFrame
(example: ML model is a transformer that transforms a DF with features into a DF with predictions)

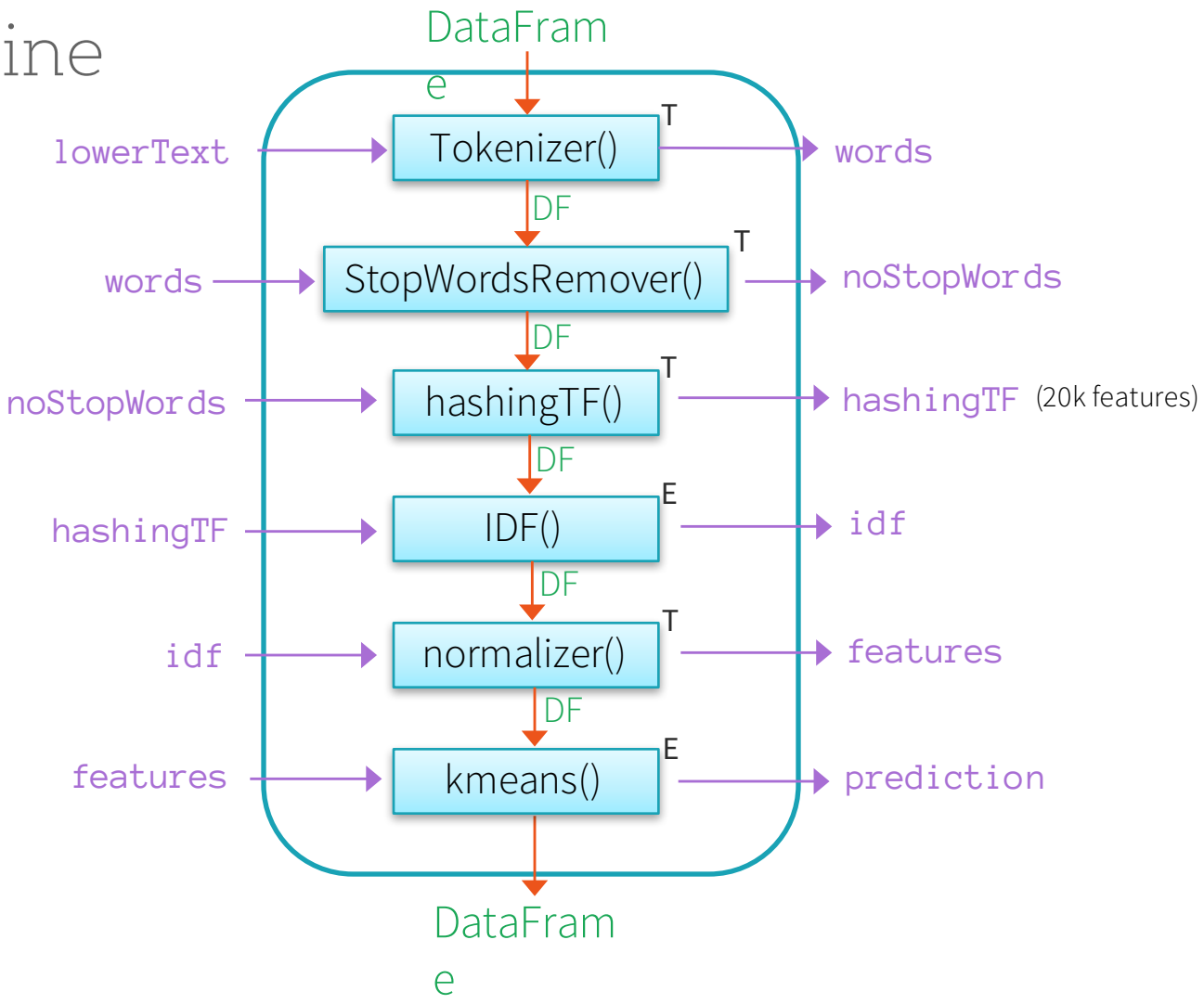
Estimator: an algorithm which can be fit on a DF to produce a Model
(example: a learning algorithm is an Estimator which trains on a DF and produces a model)

Pipeline: chains multiple Transformers and Estimators together to specify a ML workflow

Spark ML Pipeline



Spark ML Pipeline



Chat w/ Vida

Compression types

What are the benefits/disadvantages of various compression types?

- Non-splittable: gzip, zip
- Splittable: Lzo, lz4, bzip2, snappy
- Columnar: Parquet

Advantages & Pitfalls of File Types

Data Storage Options

- SQL Databases/Data Warehouses

- NoSQL Stores

- Specialized Engines

Want to Learn More?

Not your father's database:

How to use Apache Spark properly in your big data architecture

Wednesday, 4:20pm–5:00pm

Enterprise Adoption Track


Spark
Spark
Streaming
databrickstraining

Motivation

Stability condition for any streaming app

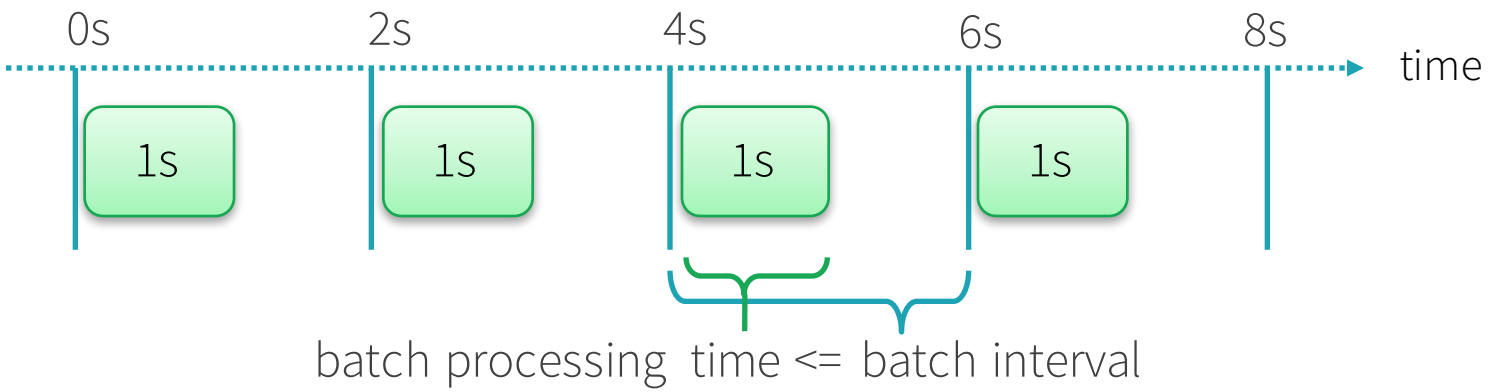
Receive data only as fast as the system can process it

Stability condition for Spark Streaming's "micro-batch" model

Finish processing previous batch before next one arrives

Stable micro-batch operation

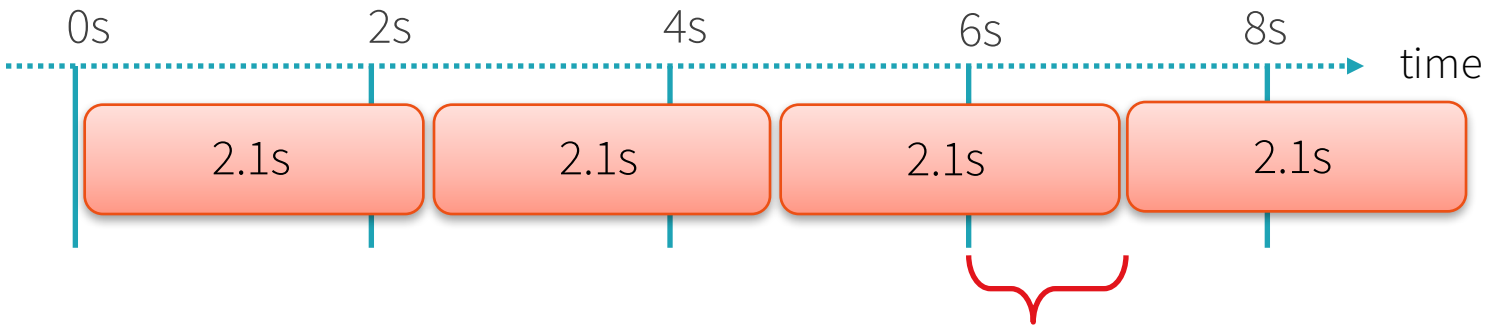
Spark Streaming runs micro-batches at fixed *batch intervals*



Previous batch is processed before next one arrives => stable

Unstable micro-batch operation

Spark Streaming runs micro-batches at fixed *batch intervals*



batch processing time > batch interval

scheduling delay builds up

Batches continuously gets delayed and backlogged => unstable

6 Edit Streams:

English : en  52.89.53.194:9002 ------->

German : de  54.68.10.240:9003 ------->

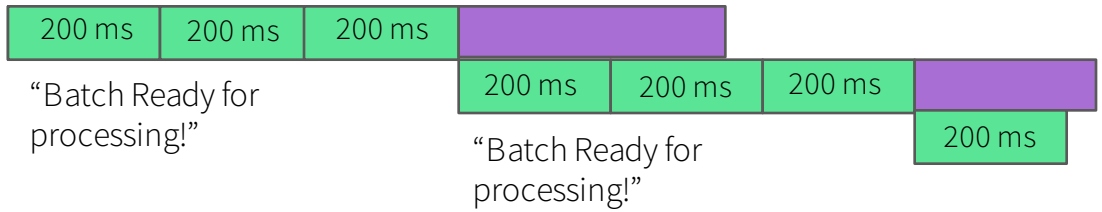
French : fr  54.68.10.240:9004 ------->

Russian : ru  54.68.10.240:9005 ------->

Italian : it  54.68.10.240:9006 ------->

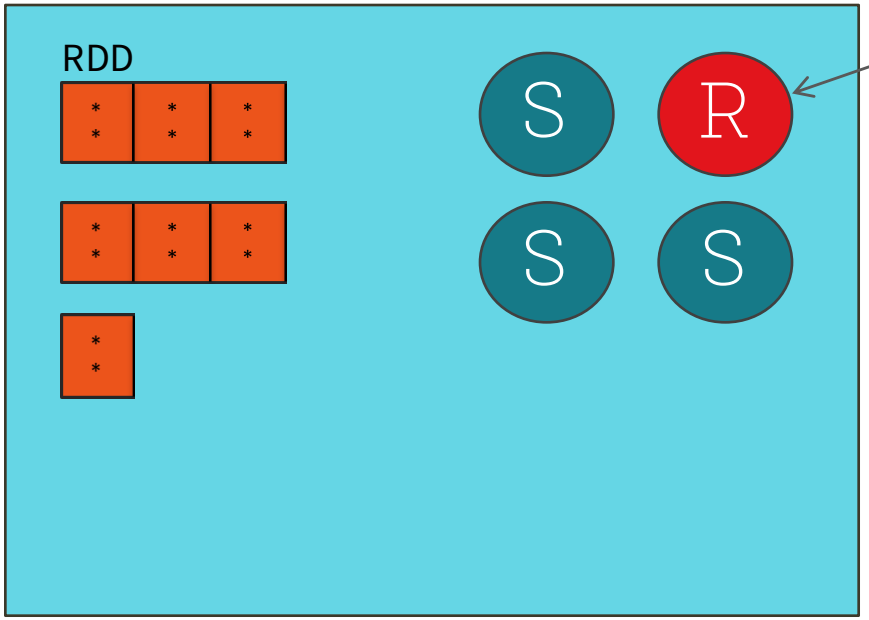
Spanish : es  54.68.10.240:9007 ------->

Batch Interval: 600 ms



:9002

English Edits



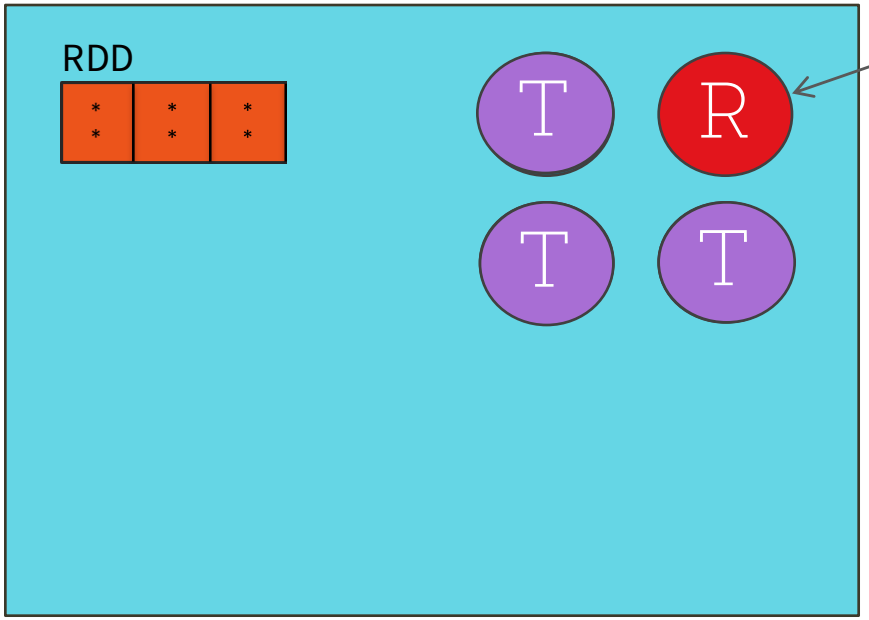
JVM

Batch Interval: 600 ms

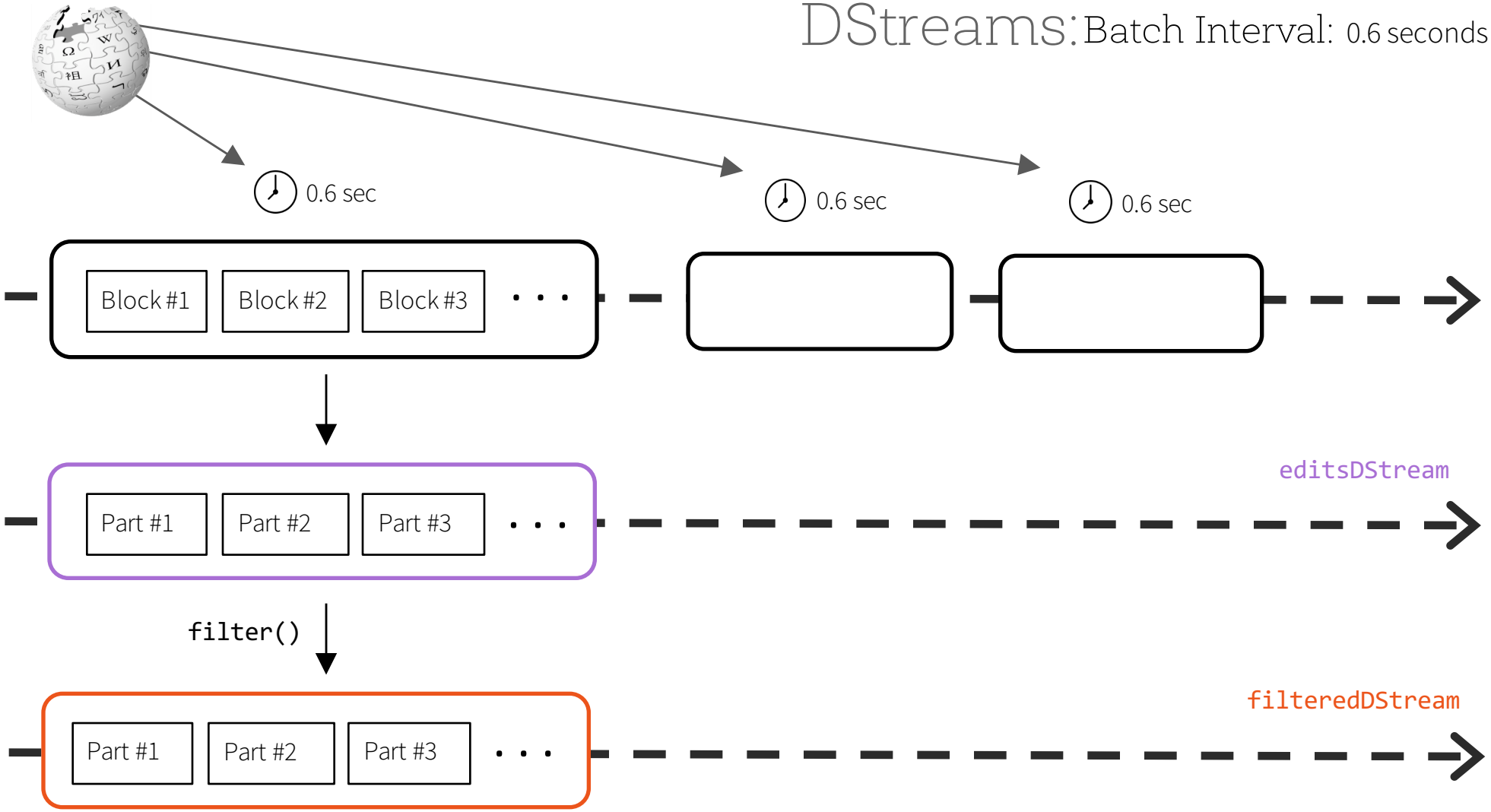
"Batch Ready for processing!"

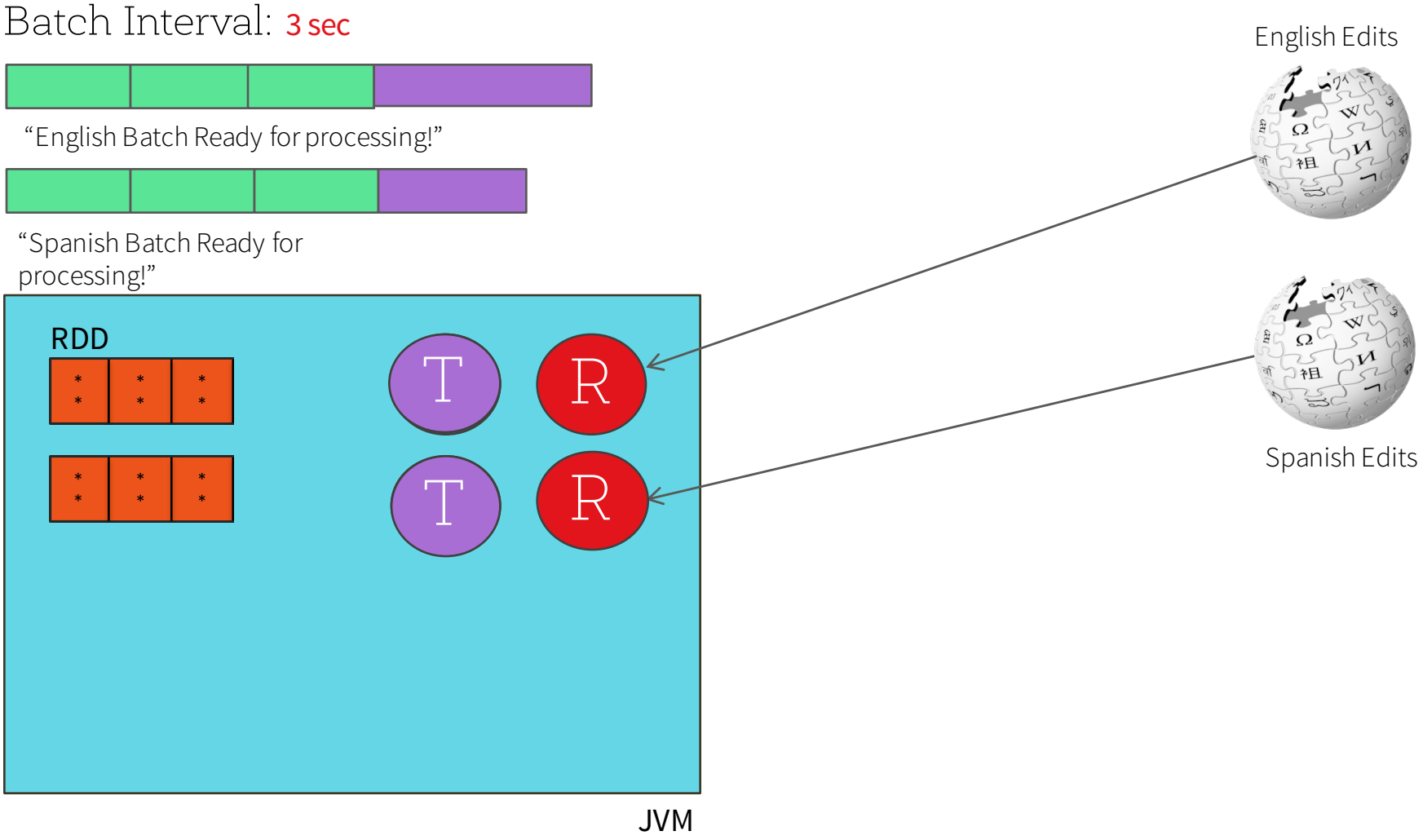
:9002

English Edits



JVM


DStreams: Batch Interval: 0.6 seconds
0.6 sec
0.6 sec
0.6 sec
Block #1
Block #2
Block #3
editsDStream
Part #1
Part #2
Part #3
filter()
filteredDStream
Part #1
Part #2
Part #3


Batch Interval: 3 sec
"English Batch Ready for processing!"
"Spanish Batch Ready for processing!"
RDD
T
R
T
R
JVM
English Edits
Spanish Edits

Transformations on DStreams:

reduce()

filter()

map()

repartition()

flatMap()

updateStateByKey()

countByValue()

join()

union()

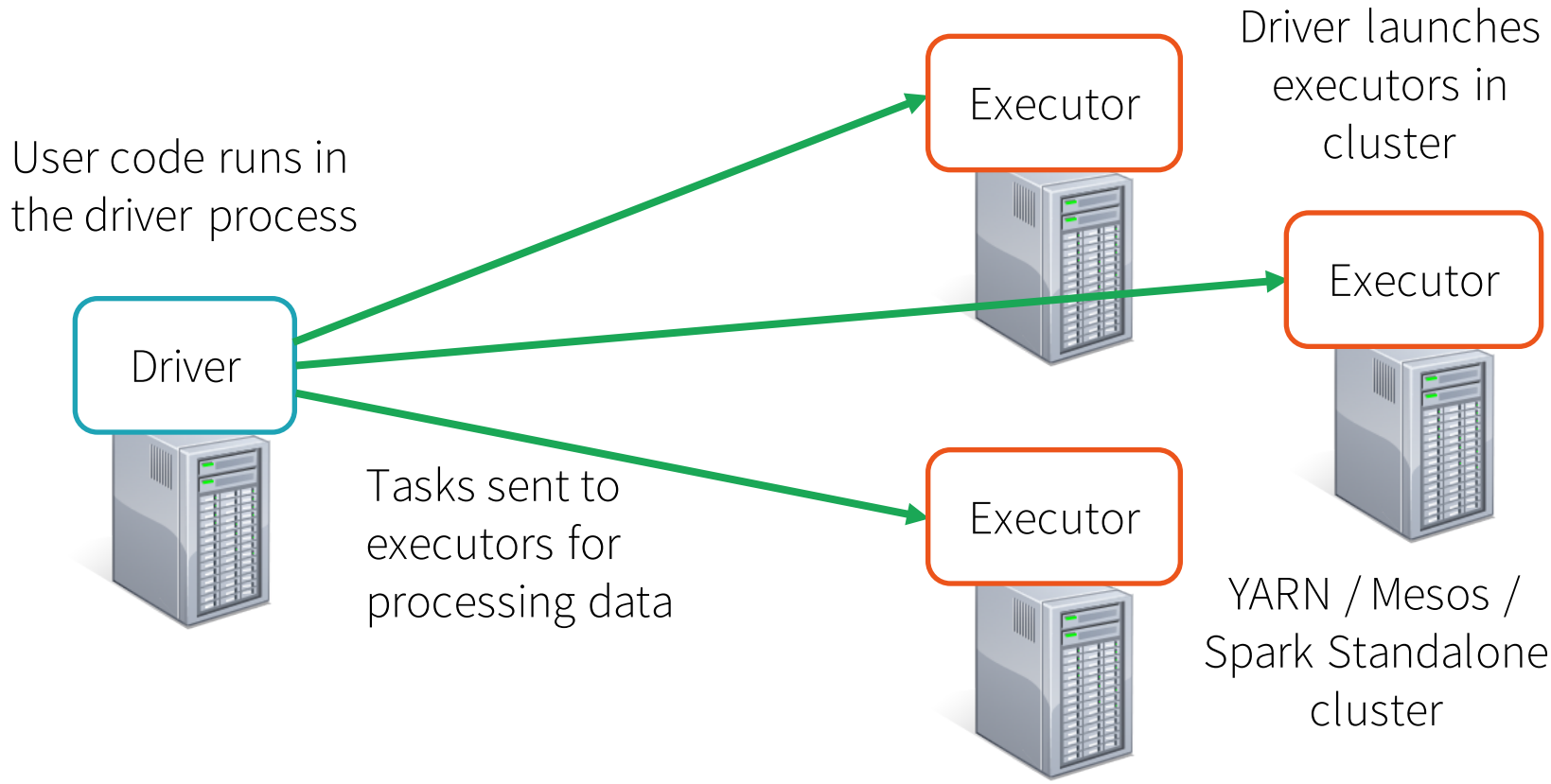
count()

transform()

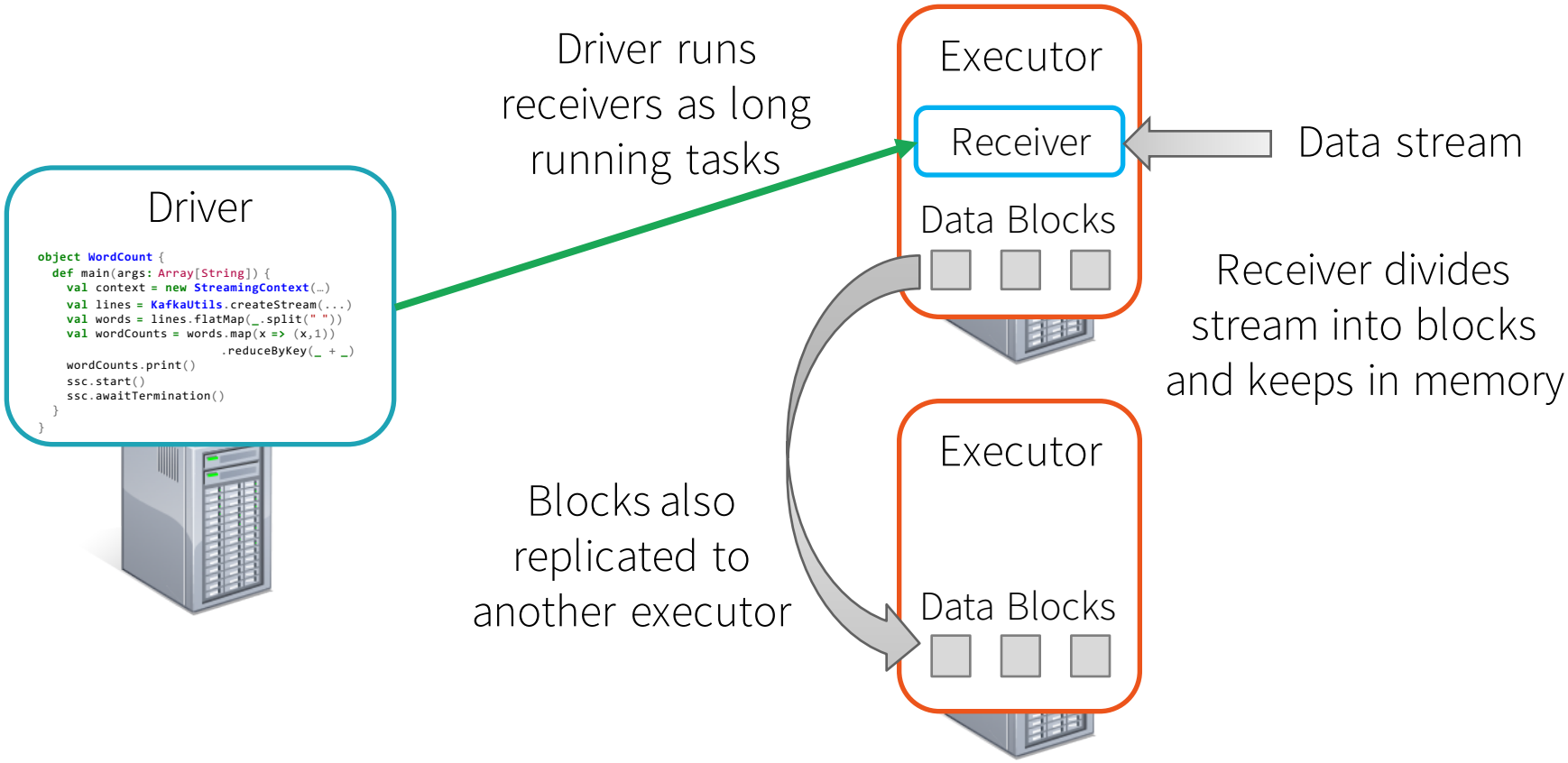
ReduceByKey()

cogroup()

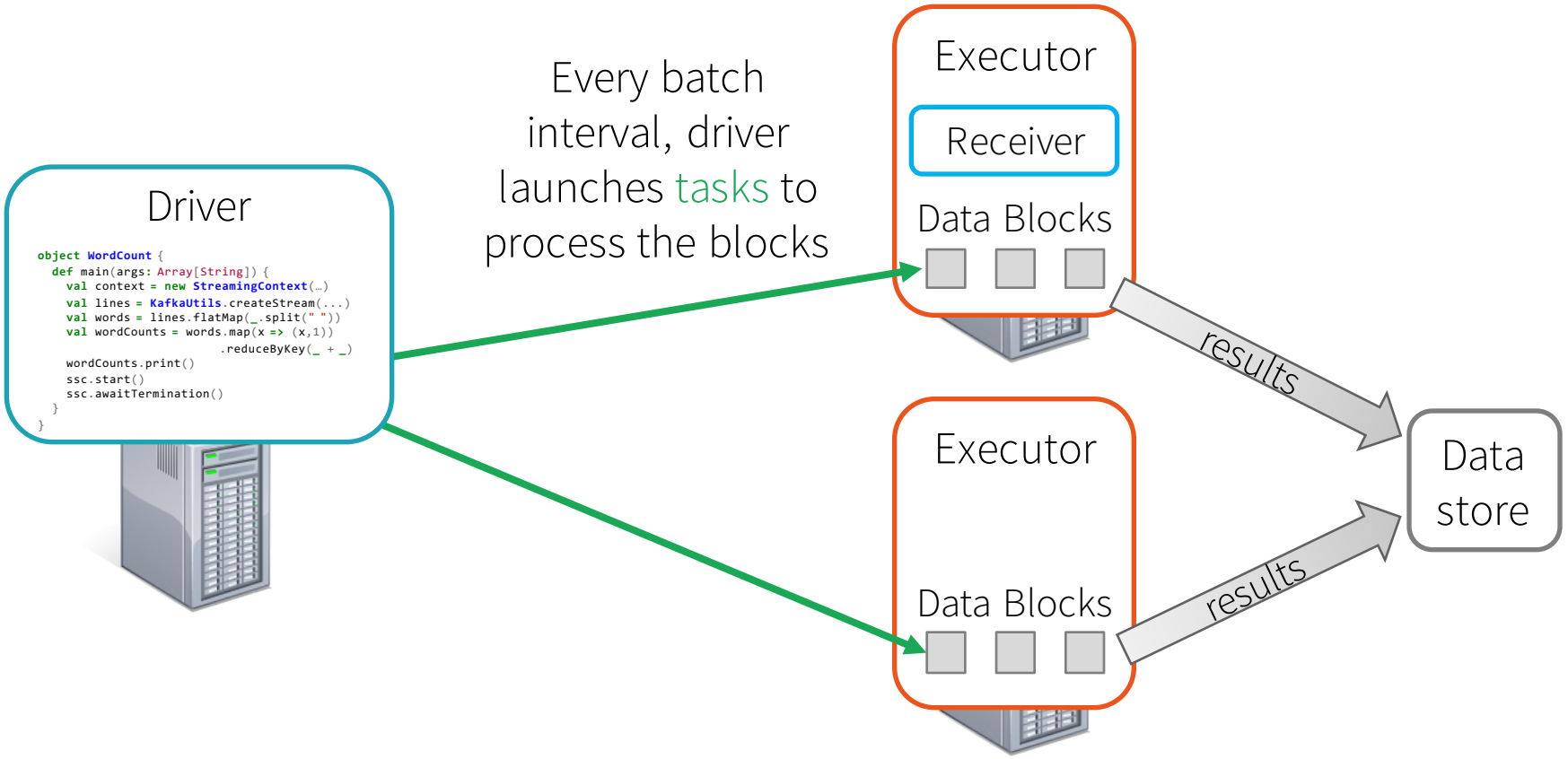
Any Spark Application



Spark Streaming Application: Receive data

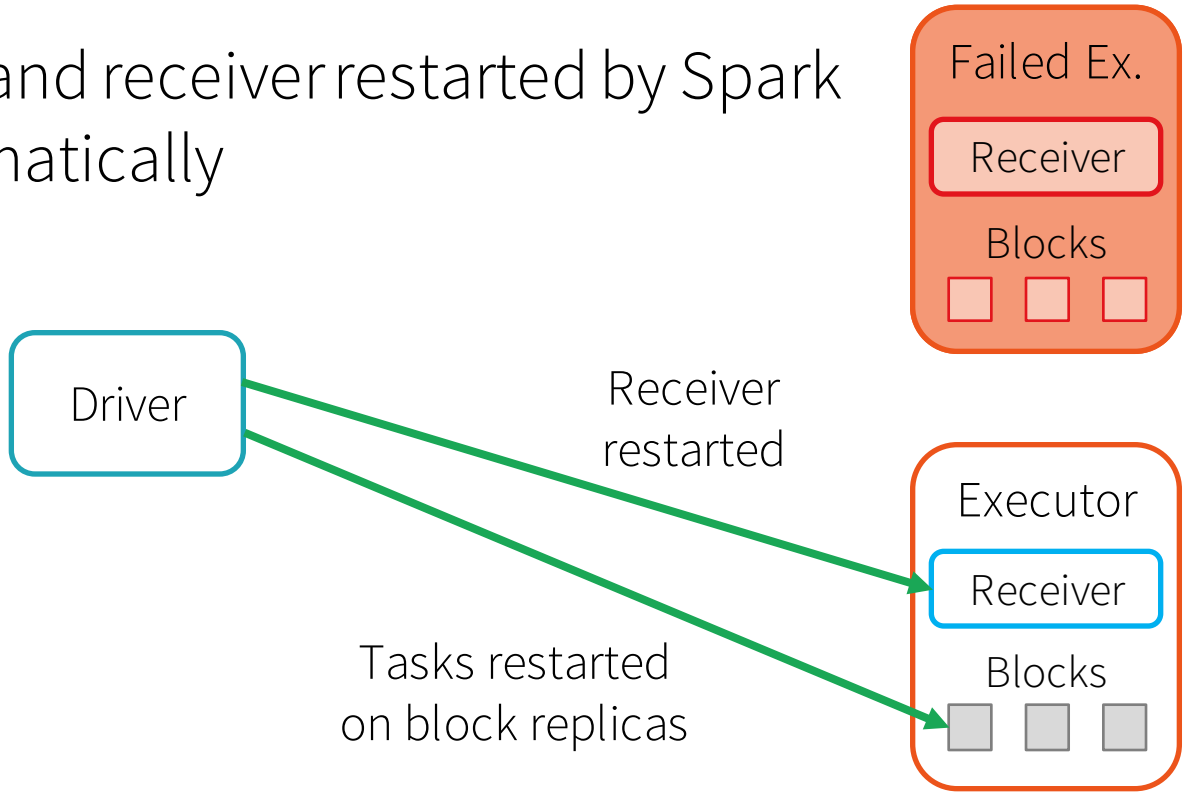


Spark Streaming Application: Process data



What if an executor fails?

Task and receiver restarted by Spark automatically



If executor fails, receiver is lost and all blocks are lost

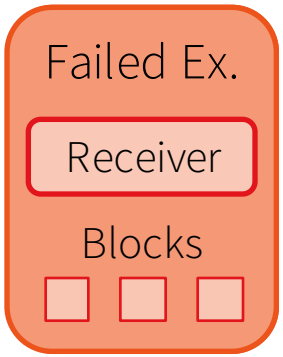
What if the driver fails?

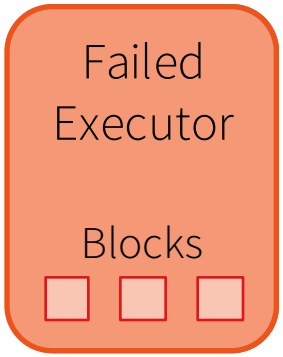
Failed Driver

When the driver fails, all the executors fail

All computation, all received blocks are lost

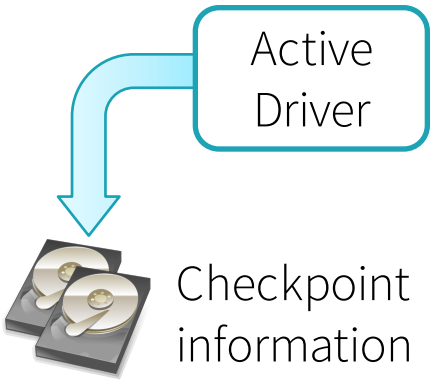
How to recover?

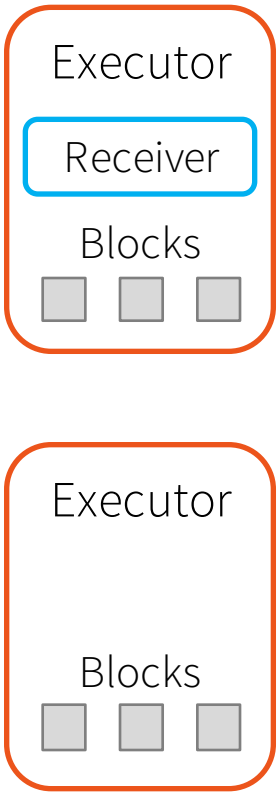




Recovering Driver with Checkpointing

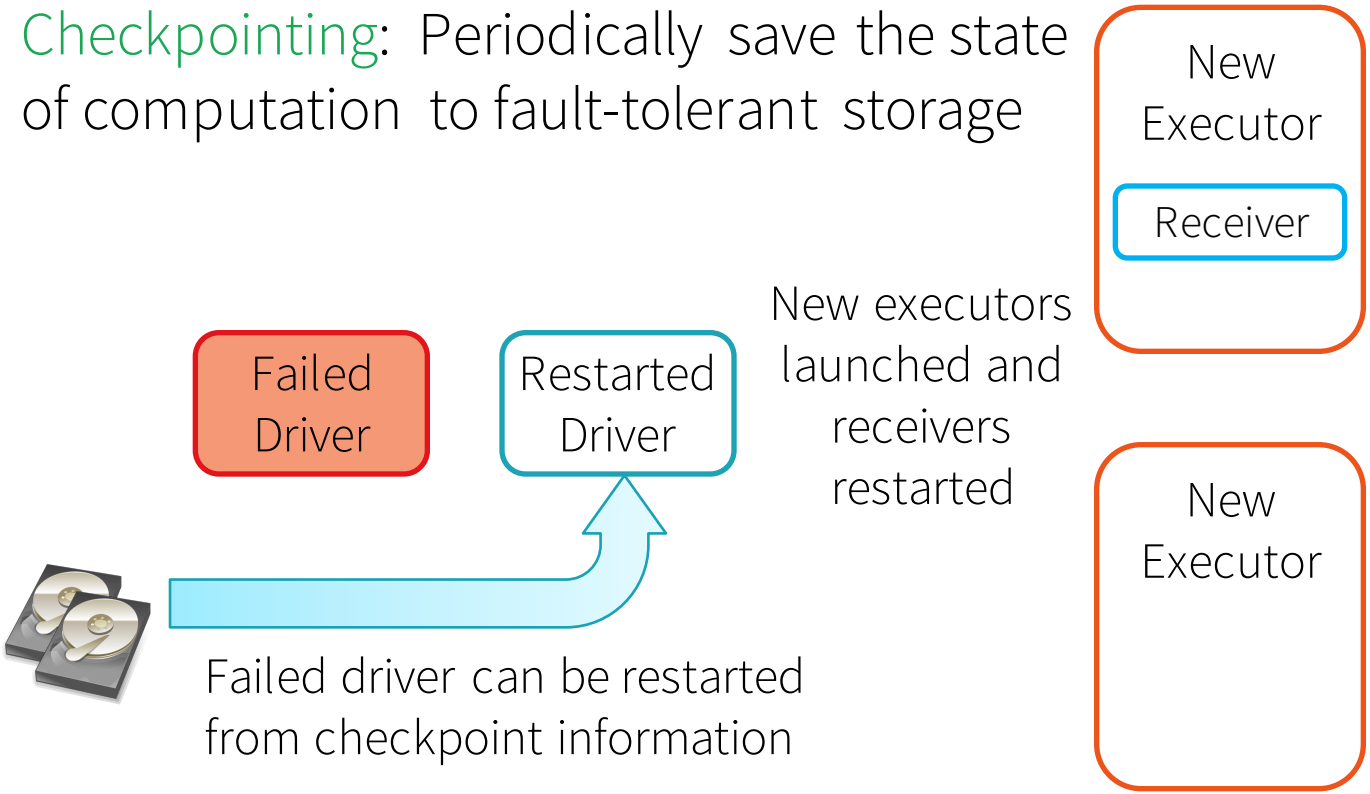
Checkpointing: Periodically save the state of computation to fault-tolerant storage



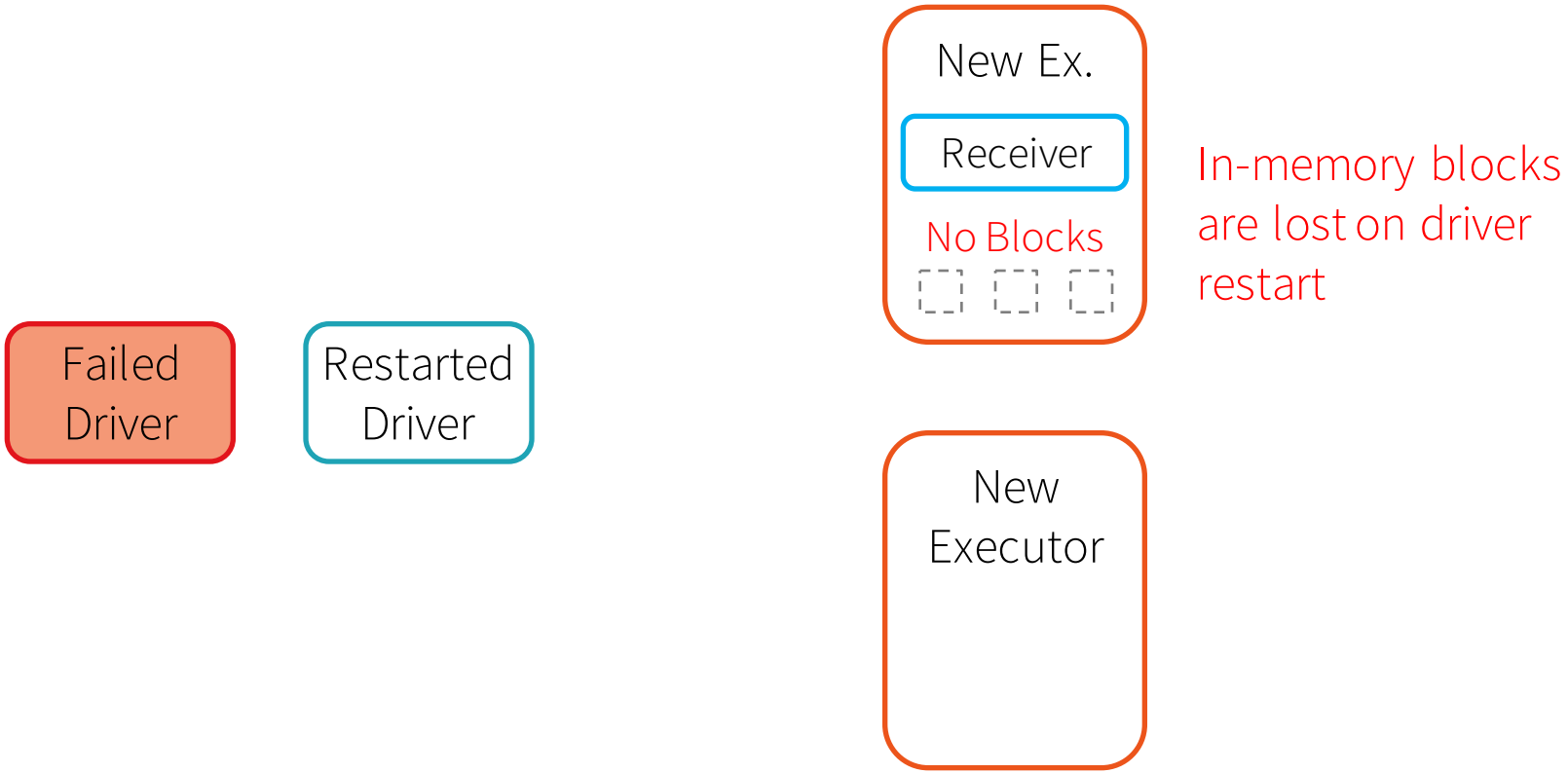


Recovering Driver with Checkpointing

Checkpointing: Periodically save the state of computation to fault-tolerant storage

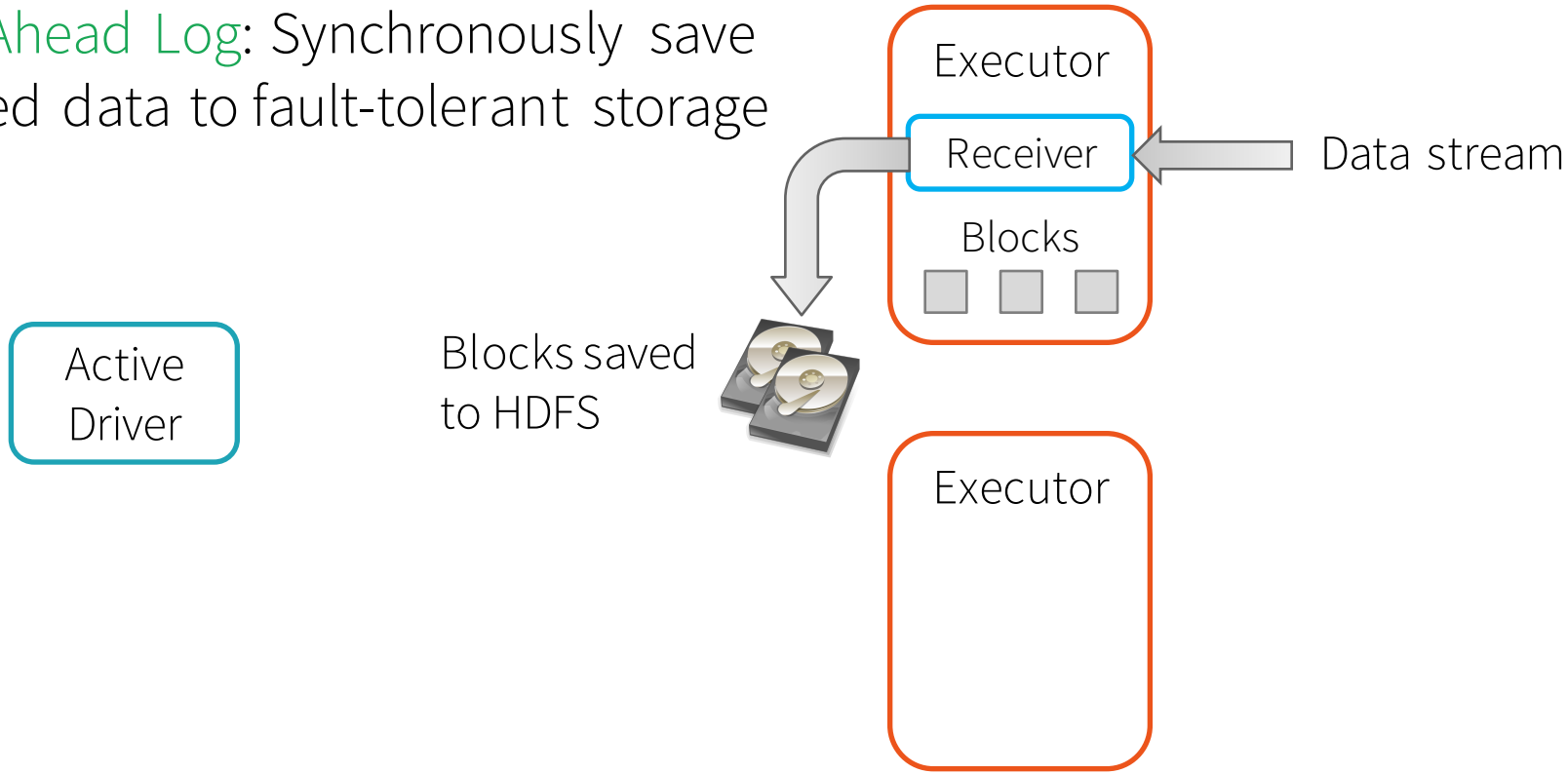


Received blocks lost on Restart!



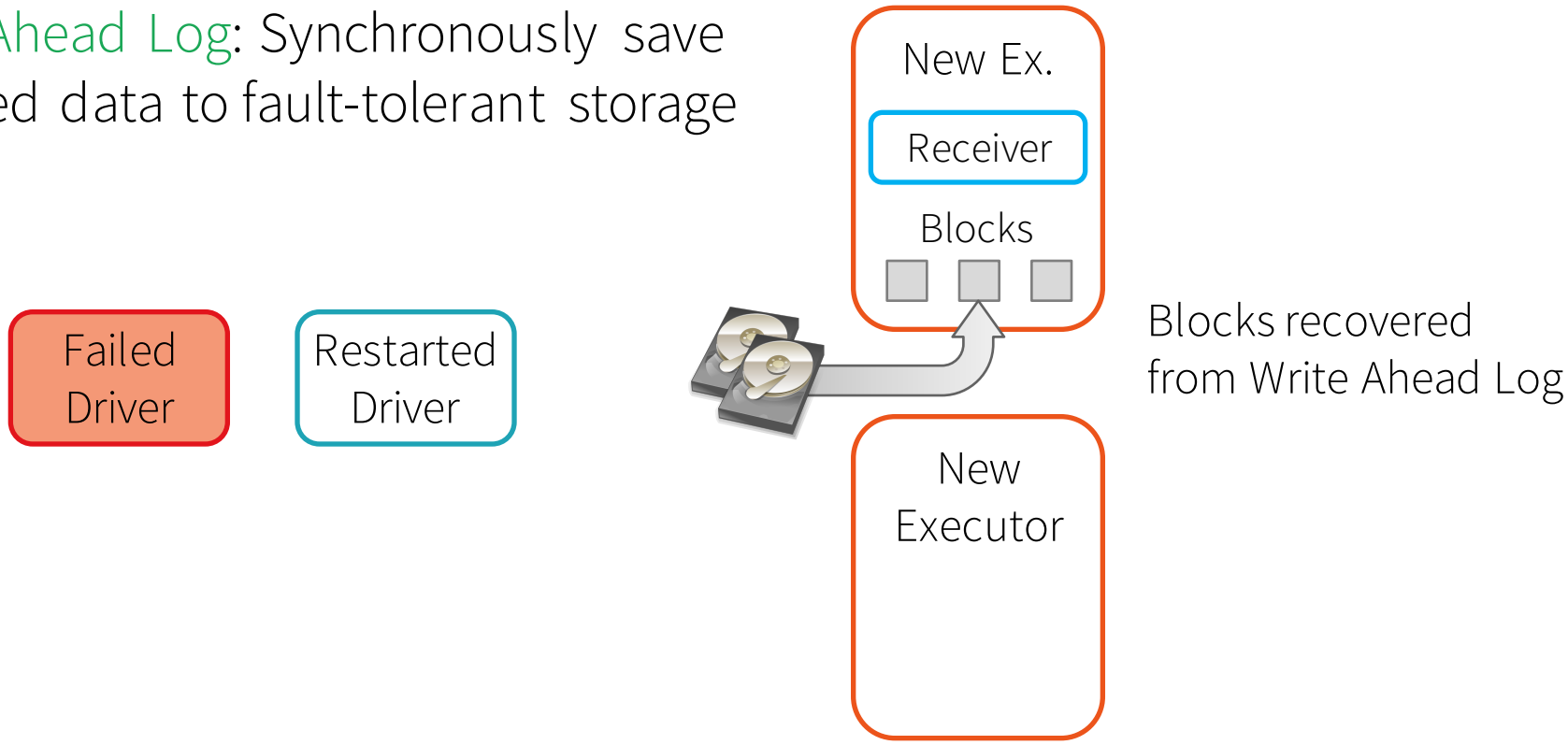
Recovering data with Write Ahead Logs

Write Ahead Log: Synchronously save received data to fault-tolerant storage



Received blocks lost on Restart!

Write Ahead Log: Synchronously save received data to fault-tolerant storage



Kafka:

Approach 1: Receiver-based Approach

This approach uses a Receiver to receive the data.

Approach 2: Direct Approach (No Receivers)

New receiver-less “direct” approach since Spark 1.3

Periodically queries Kafka for the latest offsets in each topic + partition and accordingly defines the offset ranges to process in each batch

```
val directKafkaStream = KafkaUtils.createDirectStream[
    [key class], [value class], [key decoder class], [value decoder class] ](
    streamingContext, [map of Kafka parameters], [set of topics to consume])
```

MLlib Operations on DStreams:

Streaming Linear Regression

Streaming Logistic Regression

Streaming KMeans

Online hypothesis testing (A/B testing)

Streaming DataFrames

Umbrella ticket to track what's needed to
make streaming DataFrame a reality:

https://issues.apache.org/jira/browse/SPARK-8360

Streaming DataFrames

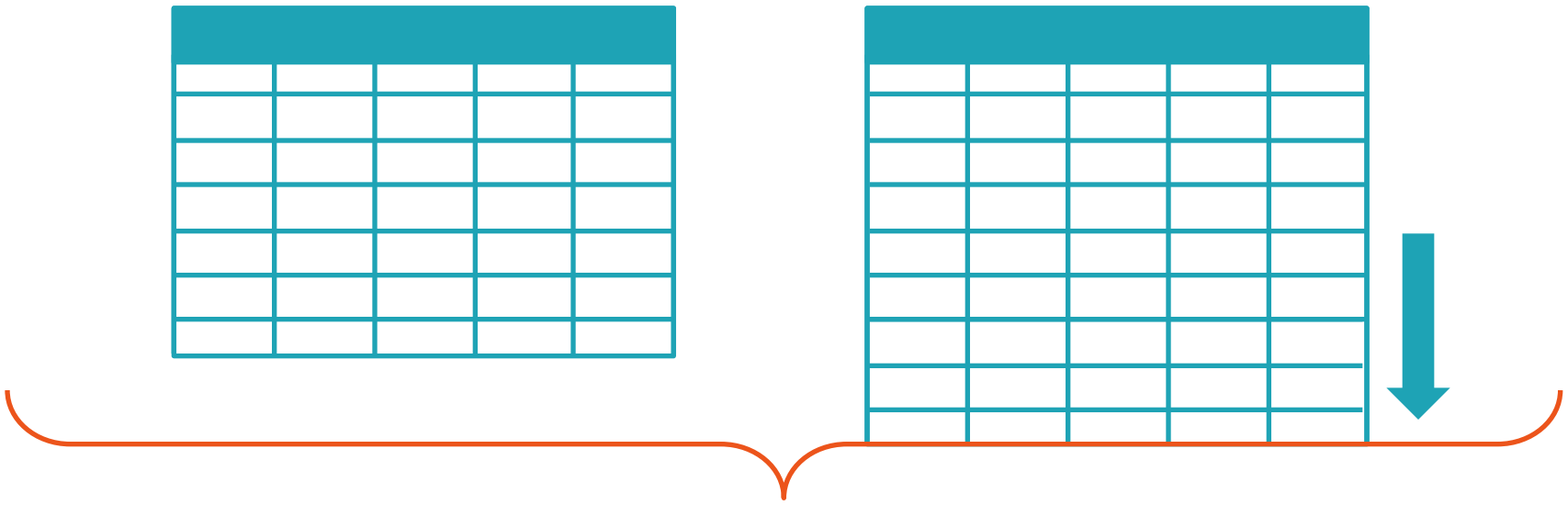
- Easier-to-use APIs (batch, streaming, and interactive)

```
val stream = read.kafka("...")
stream.window(5 mins, 10 secs)
  .agg(sum("sales"))
  .write.jdbc("mysql://...")
```

- And optimizations:
- - Tungsten backends
- - native support for out-of-order data
- - data sources and sinks

Spark 1.3
Static DataFrames

Spark 2.0
Continuous DataFrames



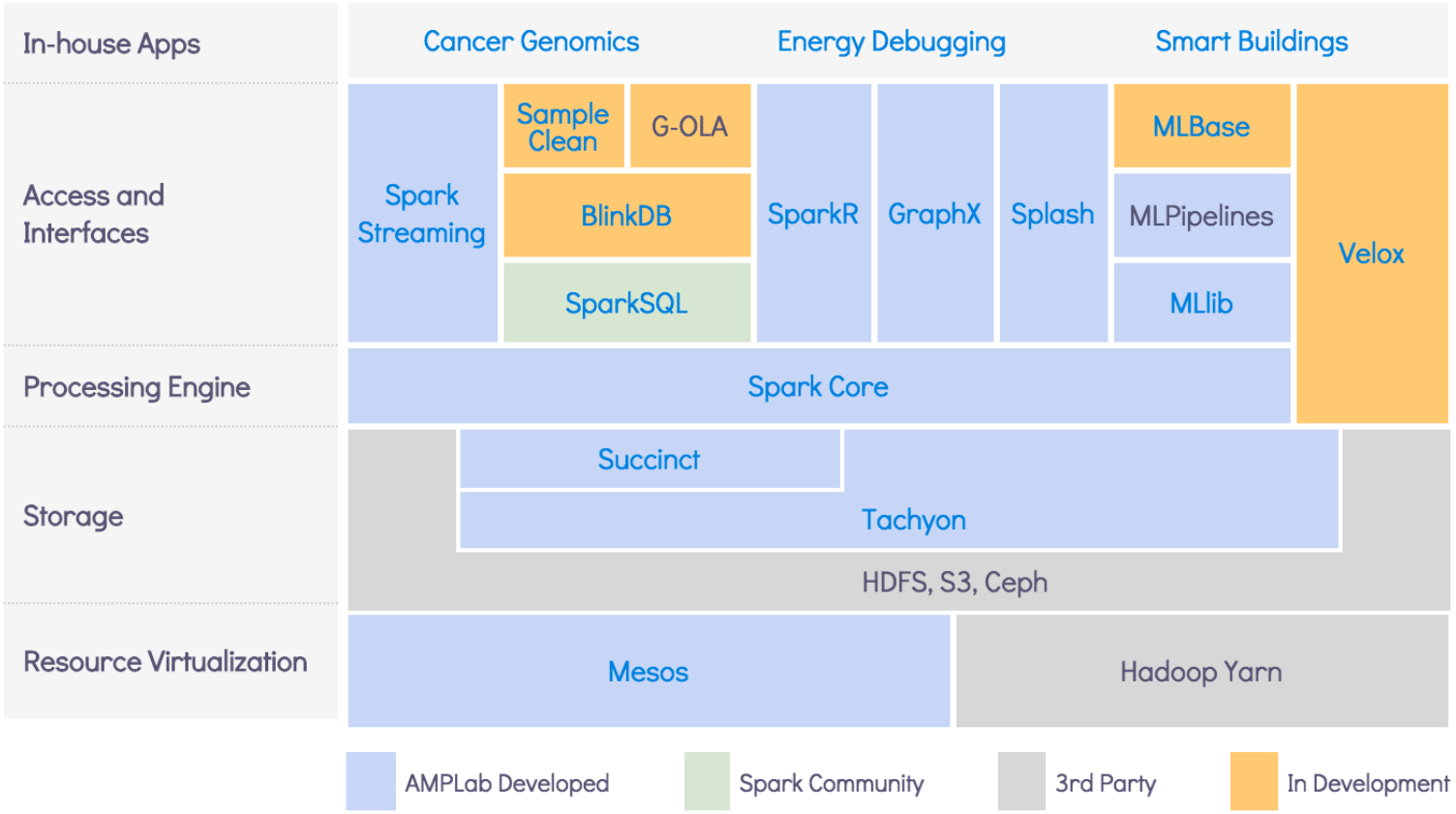
Single API !

Closing Remarks

databrickstraining

BDAS: Berkeley Data Analytics Stack

amplab UC BERKELEY



Hardware Trends

	2010	2015	
Storage	50+MB/s (HDD)	500+MB/s (SSD)	10X
Network	1Gbps	10Gbps	10X
CPU	~3GHz	~3GHz	☹

Project Tungsten

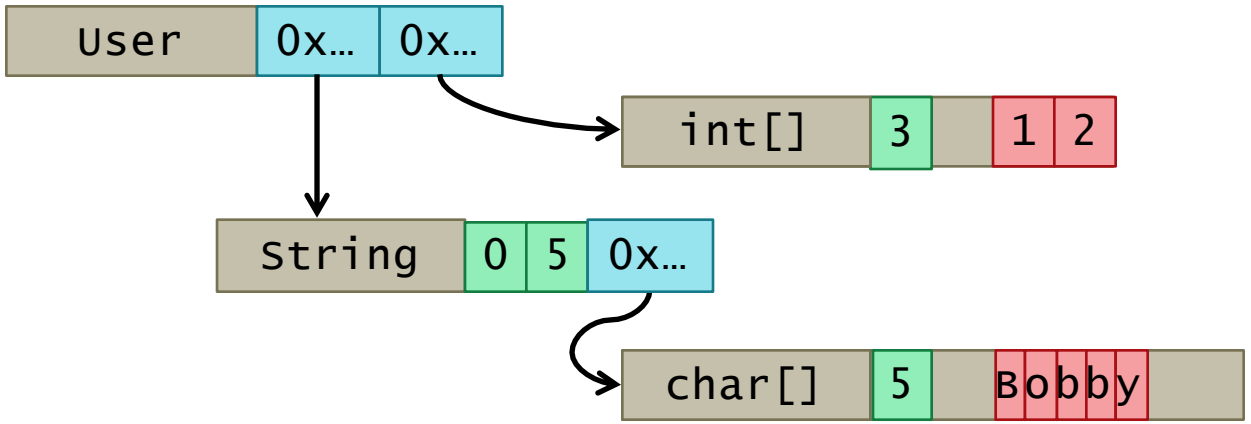
- Substantially speed up execution by optimizing CPU efficiency, via:

(1) Runtime code generation
(2) Exploiting cache locality
(3) Off-heap memory management

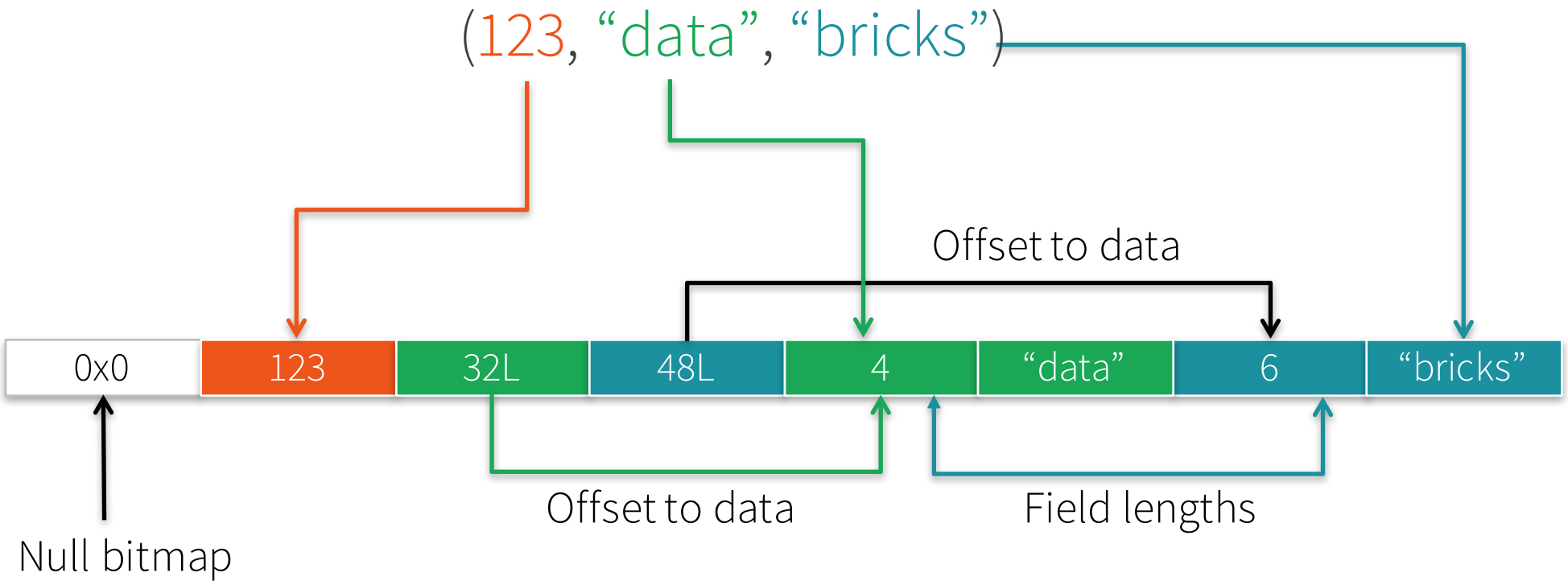
Challenge: Data Representation

- Java objects often many times larger than underlying fields

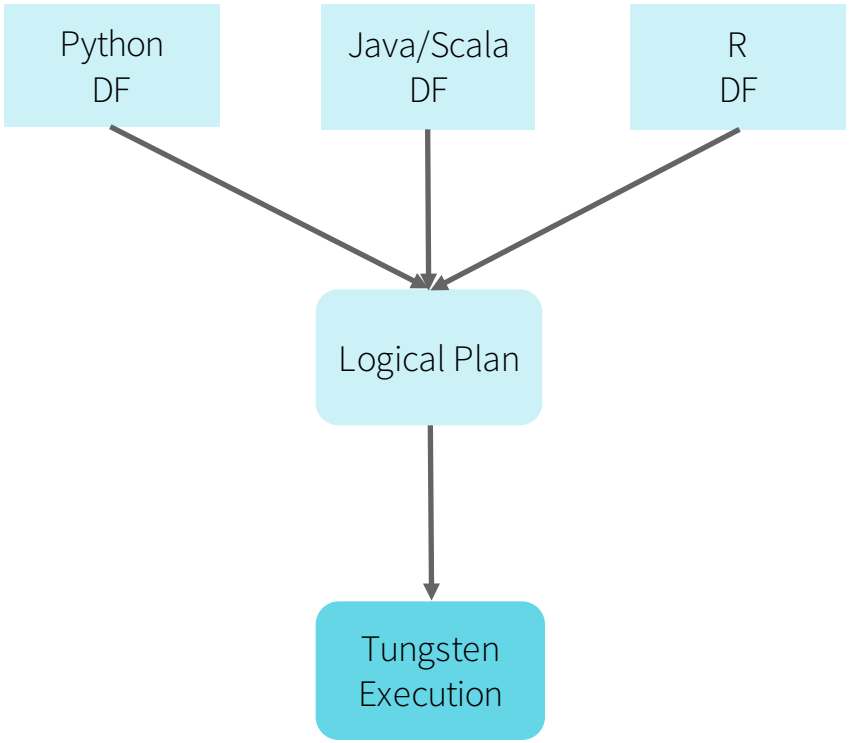
- `class User(name: String, friends: Array[Int])`
- `User("Bobby", Array(1, 2))`



Tungsten's Compact Encoding



From DataFrame to Tungsten



Initial phase in Spark 1.5

More work coming in 2016

Microsoft Azure

SALES 1-800-867-1389 | MY ACCOUNT

Why Azure | Products | Documentation | Pricing | Partners | Blog | Resources | Support

BLOG > ANNOUNCEMENTS , VIRTUAL MACHINES

Largest VM in the Cloud

THURSDAY, JANUARY 8, 2015

DREW MCDANIEL
Principal Program Manager, Azure

G-Series Size Details

VM Size	Cores	RAM		
Standard_G1	2	2		
Standard_G2	4	5		
Standard_G3	8	11		
Standard_G4	16	22		
Standard_G5	32	448 GiB	6596 GB	64

AWS Announces X1 Instances For EC2 With 2TB Of Memory, Launching Next Year

Posted Oct 8, 2015 by **Frederic Lardinois** (@fredericl)

926 SHARES

Amazon today announced a massive new instance type for its AWS EC2 compute service. The

CrunchBase

Amazon

FOUNDED
1994

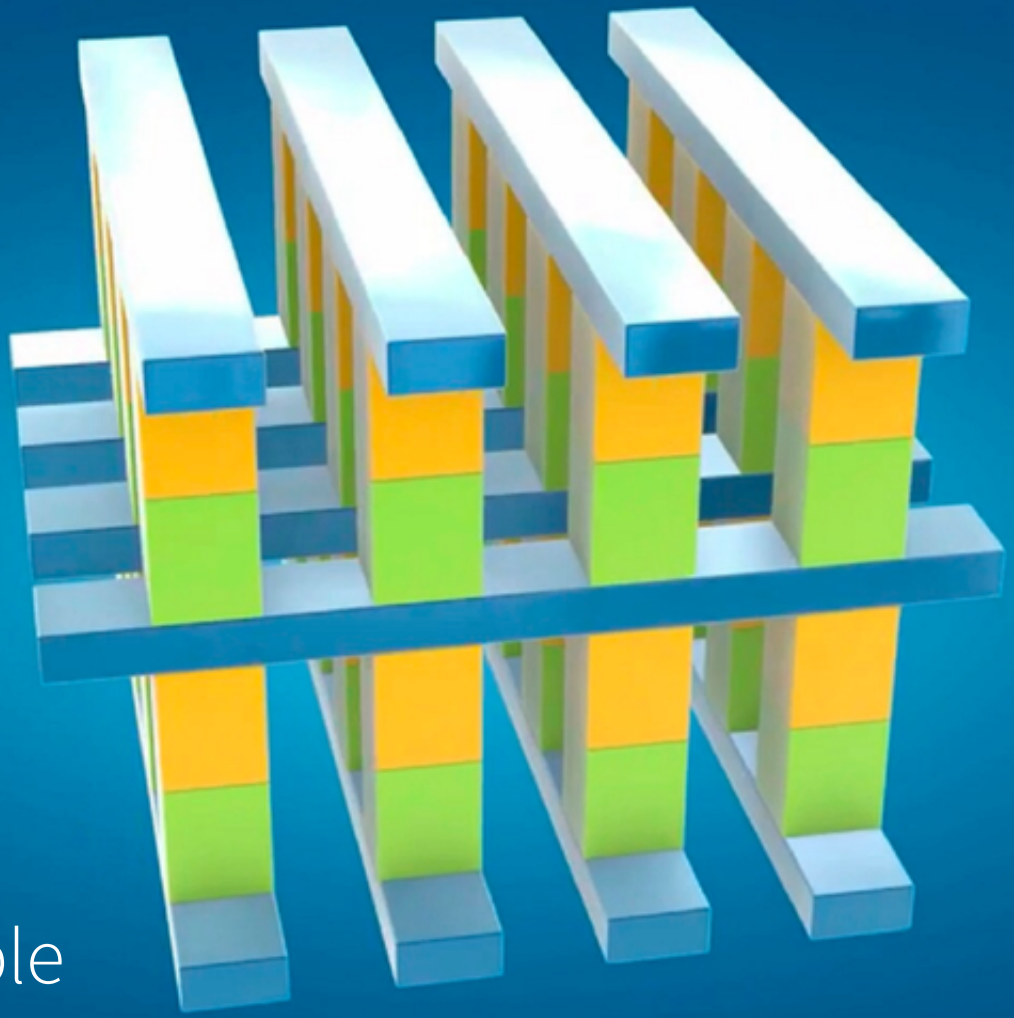
OVERVIEW
Amazon is an e-commerce retailer forme
to provide consumers with products in tw
It offers users with merchandise and cont
purchased for resale from vendors and th
by third-party sellers. Operating in North
and International markets, Amazon provi
services through websites such as amazo
amazon.ca. It also enables authors, music
filmmakers, ...

LOCATION
Seattle, WA

CATEGORIES
E-Commerce, Crowdsourcing, Groceries, Co
Goods, Delivery, Software, Retail, Internet

FOUNDERS

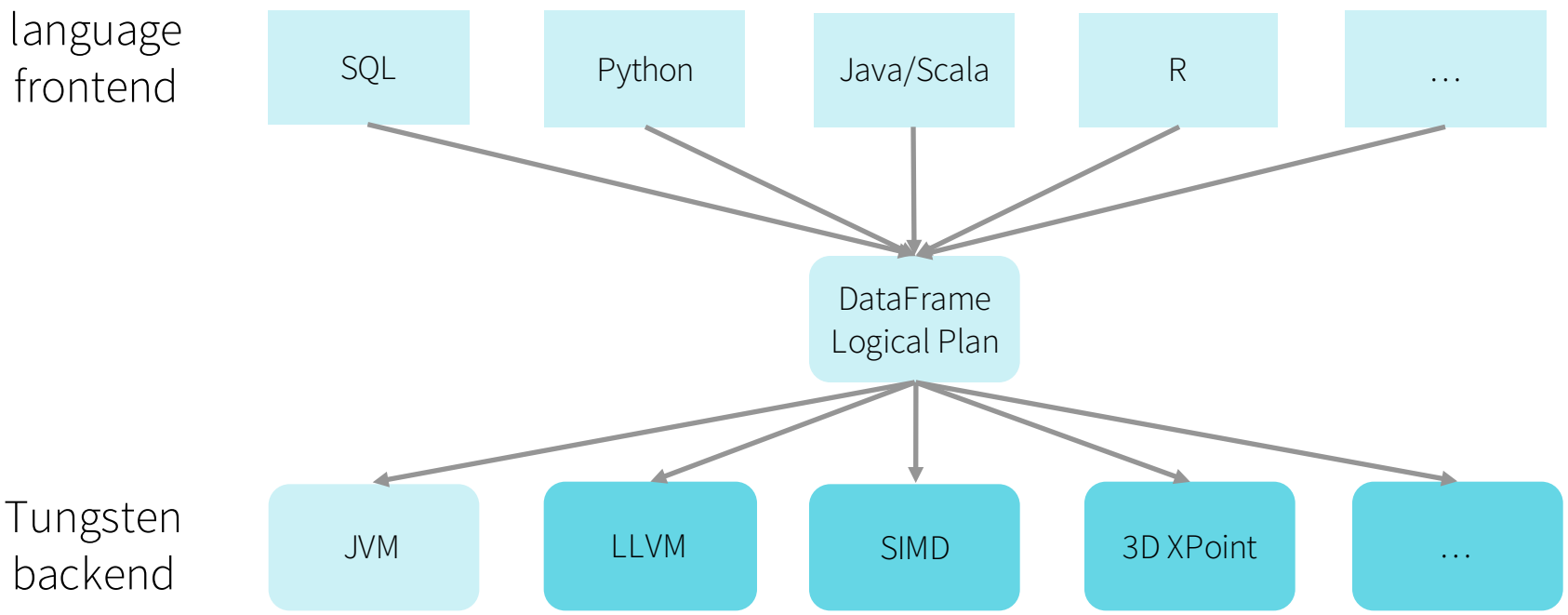
3D XPoint

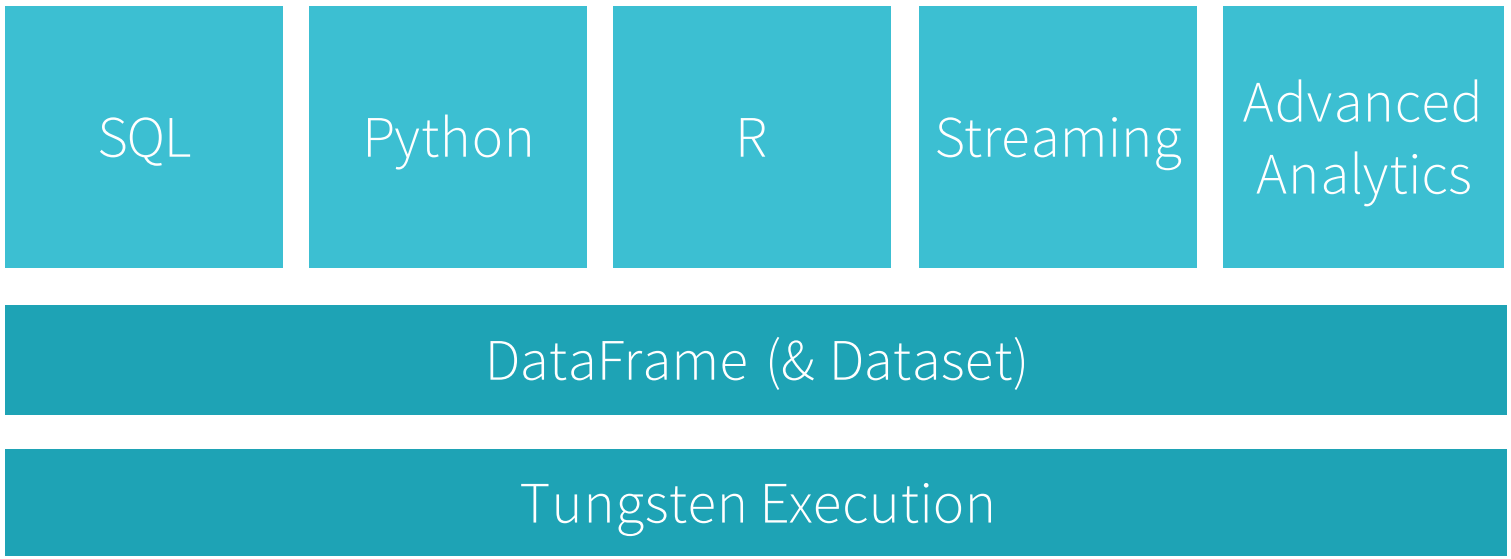


- DRAM latency
- SSD capacity
- Byte addressible

Learn more: https://en.wikipedia.org/wiki/3D_XPoint

Unified API, One Engine, Automatically Optimized




SQL
Python
R
Streaming
Advanced Analytics
DataFrame (& Dataset)
Tungsten Execution

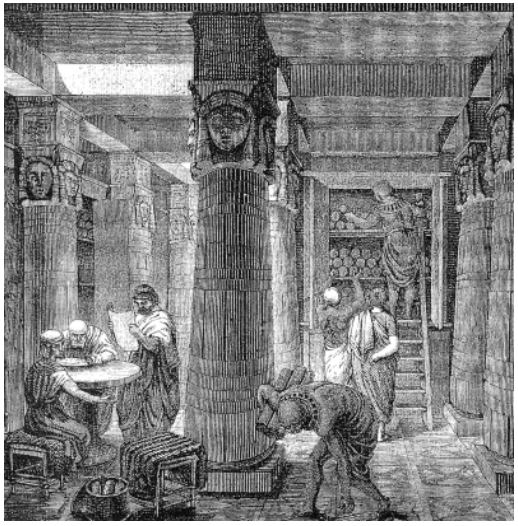
(1 Nov 2015)

Q) How many articles are needed to cover the sum of all human knowledge?

104 million!

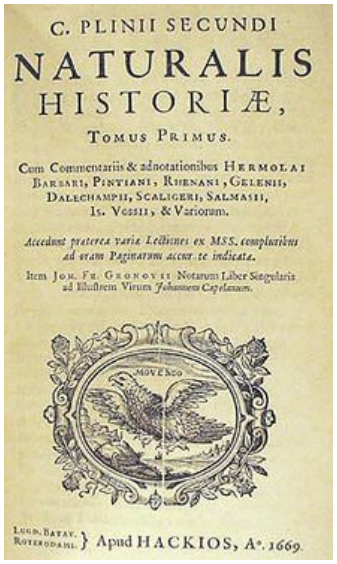
Source: https://en.wikipedia.org/wiki/User:Emijrp/All_human_knowledge

Library of Alexandria



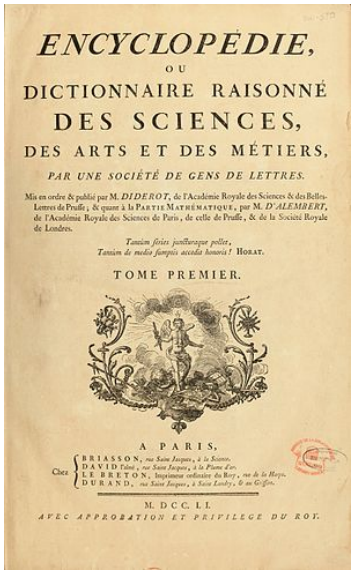
- From Egypt
- 3rd century B.C.
- 400,000 scrolls
- Burned down by Julius Caesar

Naturalis Historia



- From Roman Empire
- 77 A.D.
- 37 books

L ‘Encyclopédie

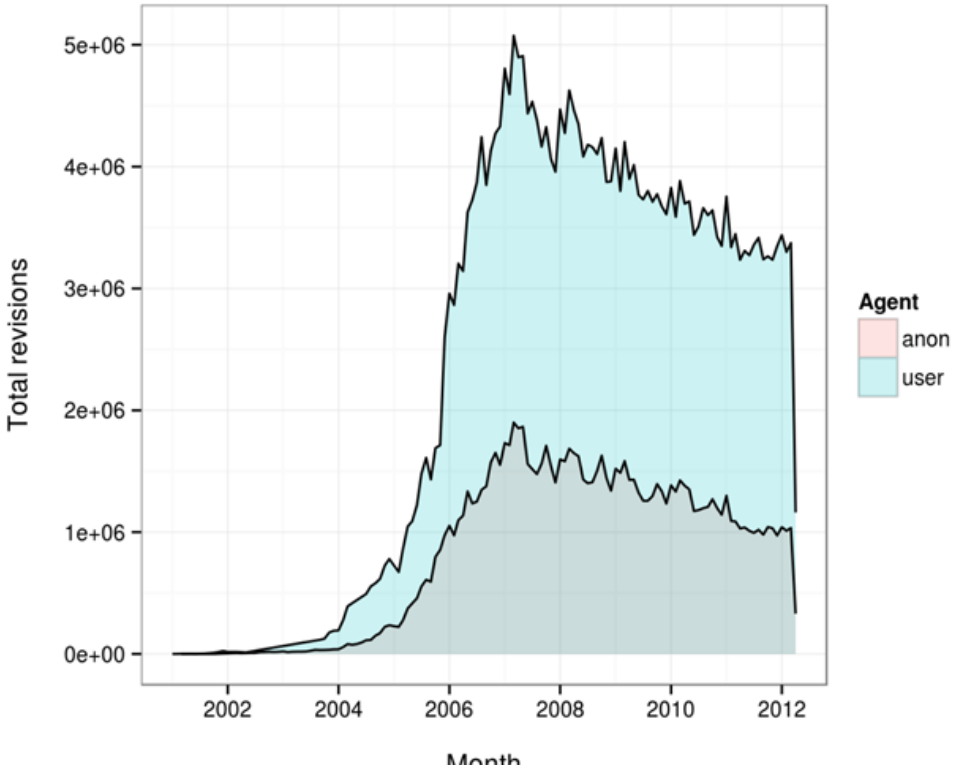


- French
- 1751
- 17 volumes
- 18,000 pages of text
- 75,000 entries

Upcoming Challenges

- Rise of smartphones
- Sock puppets, bias, paid editors…

Drop in # of Edits:



Source:
https://meta.wikimedia.org/wiki/Research:Anonymous_editor_acquisition/Volume_and_impact

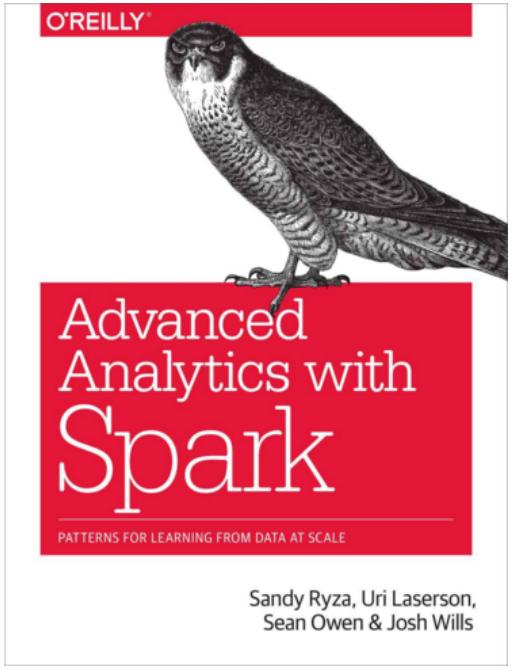
CADE METZ BUSINESS 12.01.15 7:00 AM

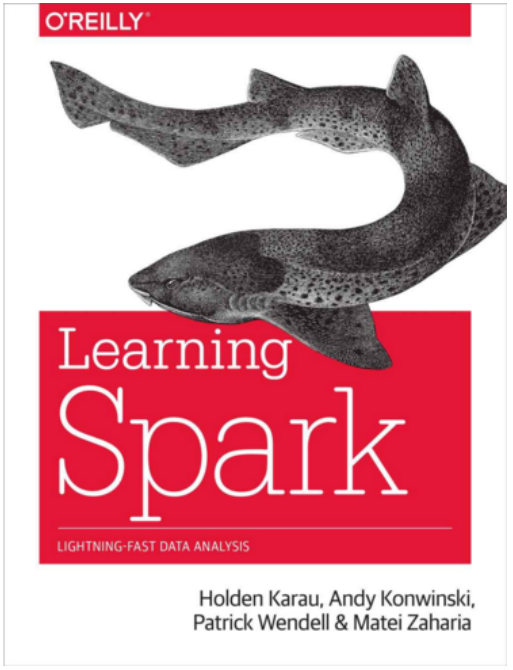
WIKIPEDIA DEPLOYS AI TO *EXPAND* ITS RANKS OF HUMAN EDITORS

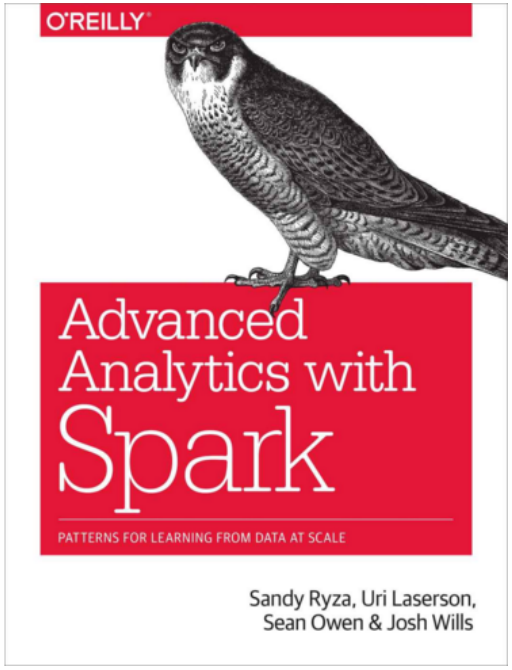
EVAN MILLS/WIRED

AARON HALFAKER JUST built an artificial intelligence engine designed to automatically analyze changes to Wikipedia.

Wikipedia is the online encyclopedia anyone can edit. In crowdsourcing the creation of an encyclopedia, the not-for-profit website forever changed the way we get information. It's among the ten most-visited sites on the Internet, and it has swept tomes like World Book and Encyclopedia Britannica into the dustbin of history. But it's not without flaws. If anyone can edit Wikipedia, anyone can mistakenly add bogus information. And anyone can vandalize the site, purposefully adding bogus information. Halfaker, a senior research scientist at the Wikimedia Foundation, the organization that oversees Wikipedia, built his AI engine as a way of identifying such vandalism.


O'REILLY
Advanced Analytics with Spark
PATTERNS FOR LEARNING FROM DATA AT SCALE
Sandy Ryza, Uri Laserson,
Sean Owen & Josh Wills


O'REILLY
Learning Spark
LIGHTNING-FAST DATA ANALYSIS
Holden Karau, Andy Konwinski,
Patrick Wendell & Matei Zaharia



Chapter 6: Understanding Wikipedia with Latent Semantic Analysis

- The Term-Document Matrix
- Getting the Data
- Parsing and Preparing the Data
- Lemmatization
- Computing the TF-IDFs
- Singular Value Decomposition
- Finding Important Concepts
- Querying and Scoring with the Low-Dimensional Representation
- Term-Term Relevance
- Document-Document Relevance
- Term-Document Relevance
- Multiple-Term Queries

You will have access to Databricks and these datasets via Community Edition after class…

Good luck!