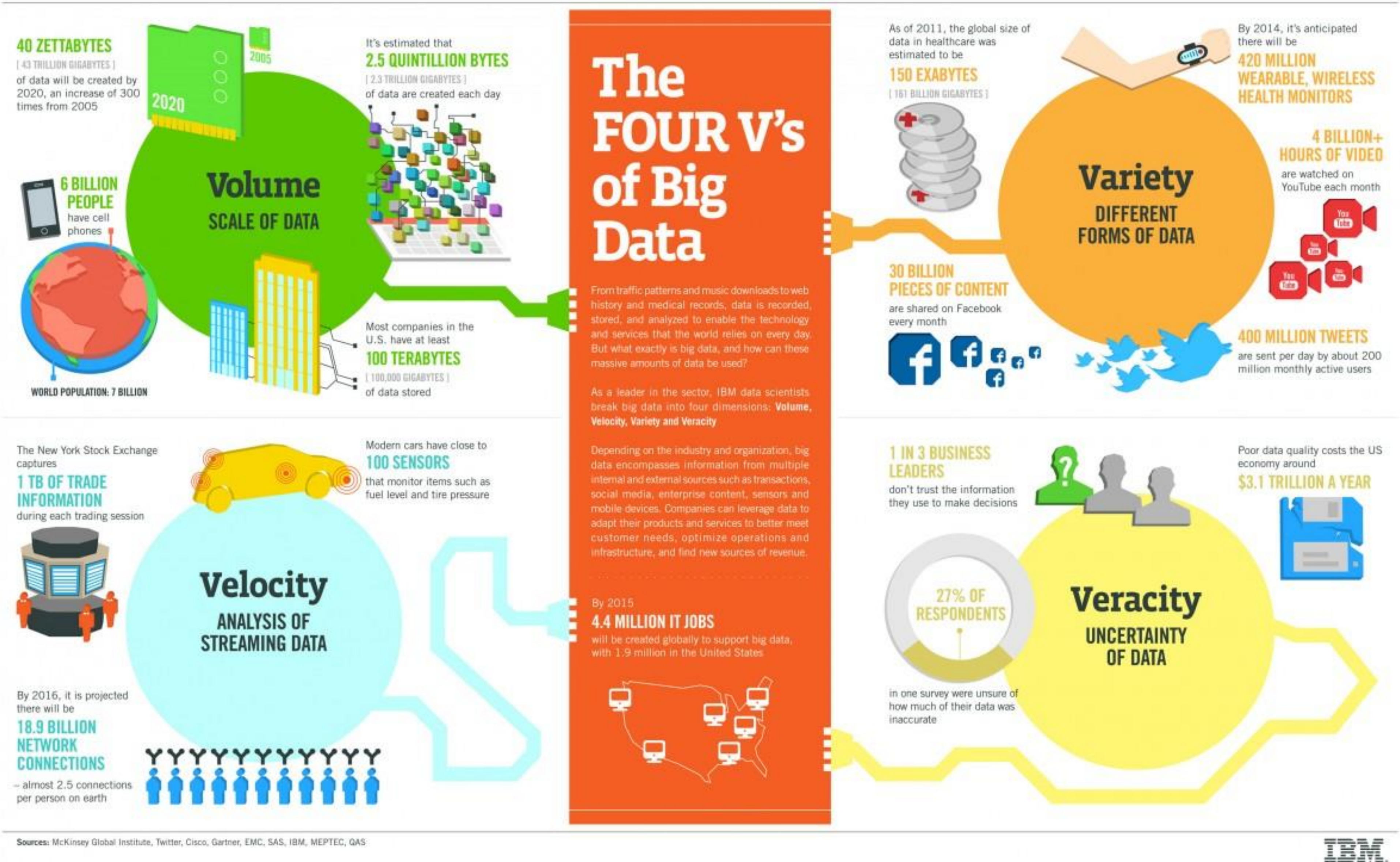
- Yogesh Simmhan
- simmhan@iisc.ac.in

- Department of Computational and Data Sciences
- Indian Institute of Science, Bangalore

Big Data Processing with Apache Spark

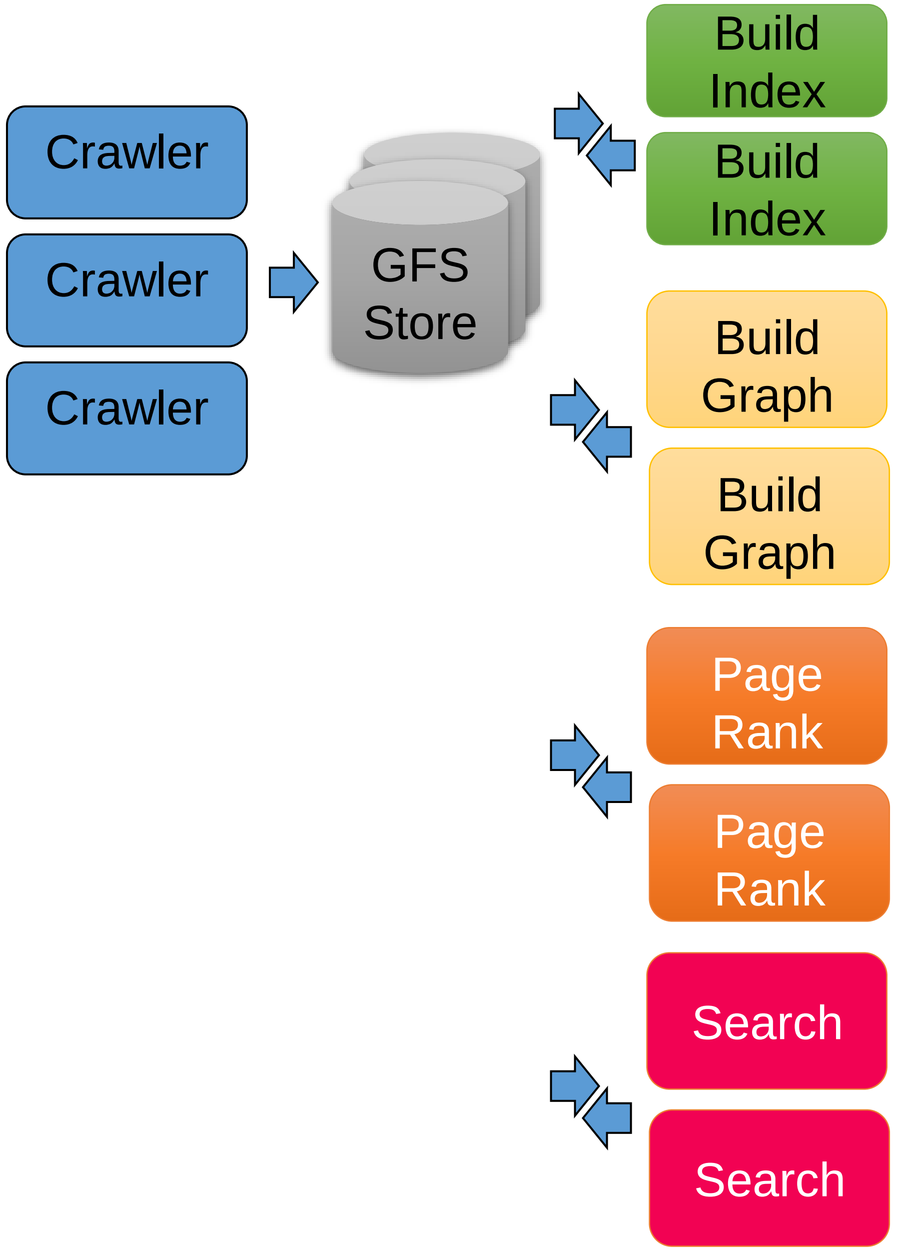
Motivation



http://www.ibmbigdatahub.com/infographic/four-vs-big-data

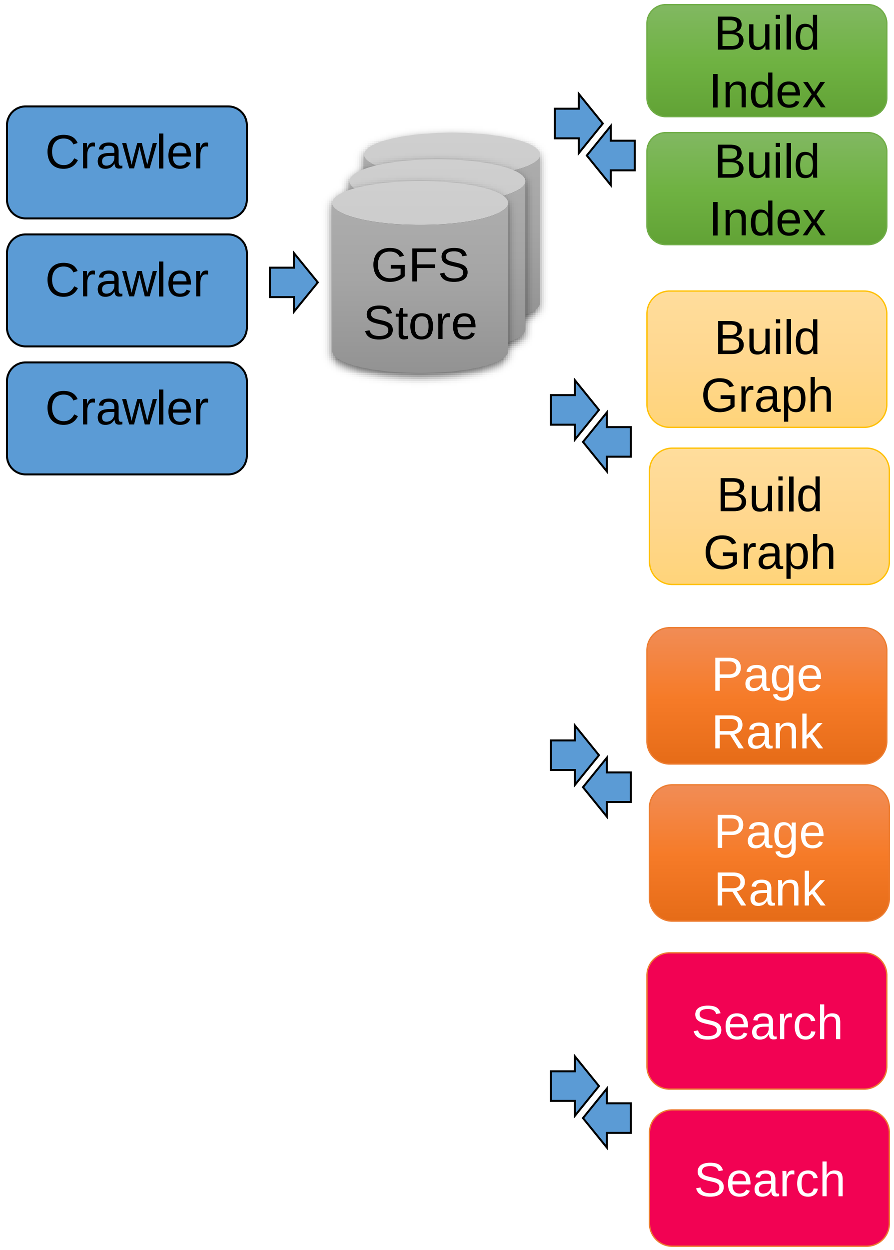
Motivation: Store

- Google wants to search the entire WWW
- How do we store the WWW at scale?
  - *“few million files, each typically 100 MB or larger in size”*
  - *“large streaming reads and small random reads”*
  - Google File System/HDFS



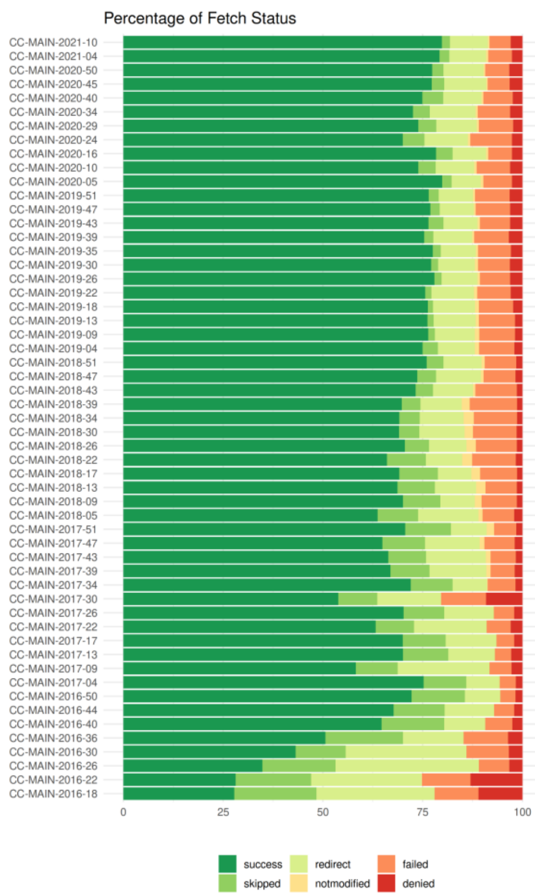
Motivation: Process

- Google wants to search the entire WWW
- How do we process this data at scale?
  - Inverted Index of Webpage keywords
  - PageRank algorithm for ranking



Motivation: Web Crawl & Search

- HTTP Response logs (~65k per day for CDS!)
  - Lines with HTTP response codes
  - Distribution of browser types




```
10.0.7.5 - - [04/Apr/2021:03:28:11 +0000] "GET / HTTP/1.1" 200 22613 "-" "-"
10.0.7.4 - - [04/Apr/2021:03:28:23 +0000] "GET / HTTP/1.1" 301 - "-" "-"
10.0.7.4 - - [04/Apr/2021:03:28:23 +0000] "GET / HTTP/1.1" 200 22613 "-" "-"
10.0.7.5 - - [04/Apr/2021:03:28:25 +0000] "GET /robots.txt HTTP/1.1" 200 160 "-" "Mozilla/5.0 (compatibl
n/seznambot-intro/)"
10.0.7.4 - - [04/Apr/2021:03:28:30 +0000] "GET /sitemap-pt-post-2018-08.xml HTTP/1.1" 200 501 "-" "Mozil
oveda.seznam.cz/en/seznambot-intro/)"
10.0.7.5 - - [04/Apr/2021:03:28:41 +0000] "GET / HTTP/1.1" 301 - "-" "-"
10.0.7.5 - - [04/Apr/2021:03:28:41 +0000] "GET / HTTP/1.1" 200 22613 "-" "-"
10.0.7.5 - - [04/Apr/2021:03:28:44 +0000] "GET /academics/contact-dcc/ HTTP/1.1" 200 26635 "-" "Mozilla/
9P) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.102 Mobile Safari/537.36 (compatible; Google
10.0.7.4 - - [04/Apr/2021:03:28:50 +0000] "GET / HTTP/1.1" 301 - "-" "Mozilla/5.0 (Windows NT 10.0; Win6
 Chrome/89.0.4389.114 Safari/537.36"
10.0.7.4 - - [04/Apr/2021:03:28:51 +0000] "GET / HTTP/1.1" 200 47769 "-" "Mozilla/5.0 (Windows NT 10.0;
cko) Chrome/89.0.4389.114 Safari/537.36"
10.0.7.4 - - [04/Apr/2021:03:28:51 +0000] "GET /wp-content/plugins/papercite/papercite.css?ver=5.6.2 HTT
lla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.114 Safari
```

https://commoncrawl.github.io/cc-crawl-statistics/plots/crawlermetrics

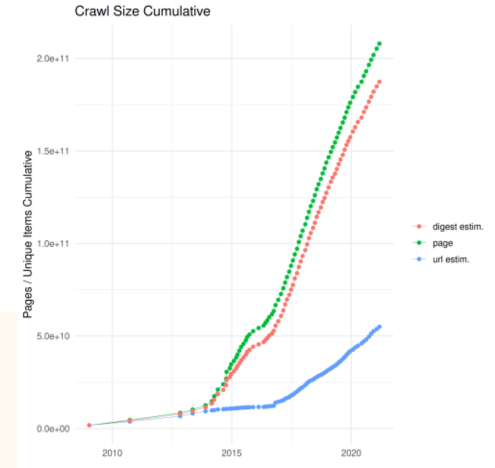
Motivation: Web Crawl & Search

- HTML file content
  - Build inverted index of words to URLs
  - Extract URL and Title
  - Extract links, build graph, find PageRank
  - Word co-occurrence and clustering

Remove stop words, contractions

URL	Keywords[]				
u1	We	The	People	Of	India
u2	It	Was	The	Best	Of
u3	Call	Me	Ishmael	Some	Years
u4	Here's	My	Number	Call	me
u5	People	Call	Me	The	Best
u6	Number	Of	People	In	India
u7	Best	Years	Of	My	Life

Keyword	URL List		
People	u1	u5	u6
India	u1	u6	
Best	u2	u5	u7
Call	u3	u4	u5
Ishmael	u3		
Some	u3		
Years	u3	u7	
Here	u4		
Number	u4	u6	
Life	u7		



Motivation: Web Crawl & Search

- HTML file content
  - Build inverted index of words to URLs
  - Extract URL and Title
  - Extract links, build graph adjacency list
  - Word co-occurrence and clustering

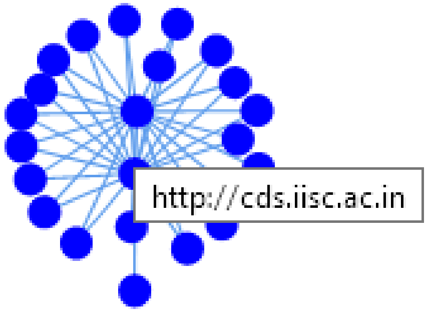
URL	Title
u1	The Constitution of India
u2	A Tale of Two Cities by Dickens
u3	Project Gutenberg - Moby Dick
u4	Carly Rae Jepsen - Call Me Maybe
u5	Shah Rukh Khan interview
u6	Wikipedia – India's Population
u7	Best Years of My Life Pistol Annies

```
WARC/1.0
WARC-Type: response
WARC-Date: 2014-08-02T09:52:13Z
WARC-Record-ID:
Content-Length: 43428
Content-Type: application/http; msgtype=response
WARC-Warcinfo-ID:
WARC-Concurrent-To:
WARC-IP-Address: 212.58.244.61
WARC-Target-URI: http://news.bbc.co.uk/2/hi/africa/3414345.stm
WARC-Payload-Digest: sha1:M63W6MNGFDWXDSLTHF7GWUPCJUH4JK3J
WARC-Block-Digest: sha1:YHKQUSBOS4CLYFEKQDVGJ457OAPD6IJO
WARC-Truncated: length

HTTP/1.1 200 OK
Server: Apache
Vary: X-CDN
Cache-Control: max-age=0
Content-Type: text/html
Date: Sat, 02 Aug 2014 09:52:13 GMT
Expires: Sat, 02 Aug 2014 09:52:13 GMT
Connection: close
Set-Cookie: BBC-UID=...; expires=Sun, 02-Aug-15 09:52:13 GMT; path=/; domain=bbc.
co.uk;

<!doctype html public "-//W3C//DTD HTML 4.0 Transitional//EN" "http://www.w3.org/
TR/REC-html40/loose.dtd">
<html>
<head>
<title>
        BBC NEWS | Africa | Namibia braces for Nujoma exit
</title>
...
```

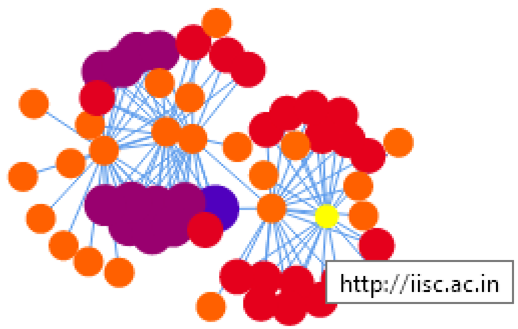
https://commoncrawl.org/the-data/get-started/

Motivation: Web Crawl & Search

- HTML file content
  - Build inverted index of words to URLs
  - Extract URL and Title
  - Extract links, build graph, find PageRank
  - Word co-occurrence and clustering




```
    <li id="menu-item-307" class="menu-item menu-item-type-post_type menu-item-object-page menu-item-has-
children menu-item-307"><a href="https://www.iisc.ac.in/about/student-corner/">Student Corner</a>
    <ul  class="sub-menu">
        <li id="menu-item-310" class="menu-item menu-item-type-custom menu-item-object-custom menu-item-has-
children menu-item-310"><a href="/about/student-corner/">General Information</a>
        <ul  class="sub-menu">
            <li id="menu-item-2324" class="menu-item menu-item-type-post_type menu-item-object-page menu-
item-2324"><a href="https://www.iisc.ac.in/campus-life/">Campus Life</a></li>
            <li id="menu-item-11152" class="menu-item menu-item-type-post_type menu-item-object-page menu-
item-11152"><a href="https://www.iisc.ac.in/my-iisc-my-life-a-student-perspective/">My IISc, my life: a student
perspective</a></li>
            <li id="menu-item-2274" class="menu-item menu-item-type-custom menu-item-object-custom menu-
item-2274"><a target="_blank" href="http://hostel.iisc.ernet.in/hostel/">Hostels/Mess</a></li>
            <li id="menu-item-312" class="menu-item menu-item-type-custom menu-item-object-custom menu-
item-312"><a target="_blank" href="https://iiscgym.iisc.ac.in/">Gymkhana</a></li>
            <li id="menu-item-315" class="menu-item menu-item-type-post_type menu-item-object-page menu-
item-315"><a href="https://www.iisc.ac.in/about/student-corner/procedure-for-obtaining-official-transcripts
/">Official transcripts</a></li>
            <li id="menu-item-3414" class="menu-item menu-item-type-post_type menu-item-object-page menu-
item-3414"><a href="https://www.iisc.ac.in/about/campus-facilities/">Campus Facilities</a></li>
            <li id="menu-item-317" class="menu-item menu-item-type-custom menu-item-object-custom menu-
item-317"><a target="_blank" href="/health-centre/">Health Centre</a></li>
            <li id="menu-item-11016" class="menu-item menu-item-type-post_type menu-item-object-page menu-
item-11016"><a href="https://www.iisc.ac.in/auditoria-and-seminar-halls/">Auditoria and Seminar Halls</a></li>
            <li id="menu-item-7170" class="menu-item menu-item-type-post_type menu-item-object-page menu-
item-7170"><a href="https://www.iisc.ac.in/icash/">Internal Committee Against Sexual Harassment (ICASH)
</a></li>
```

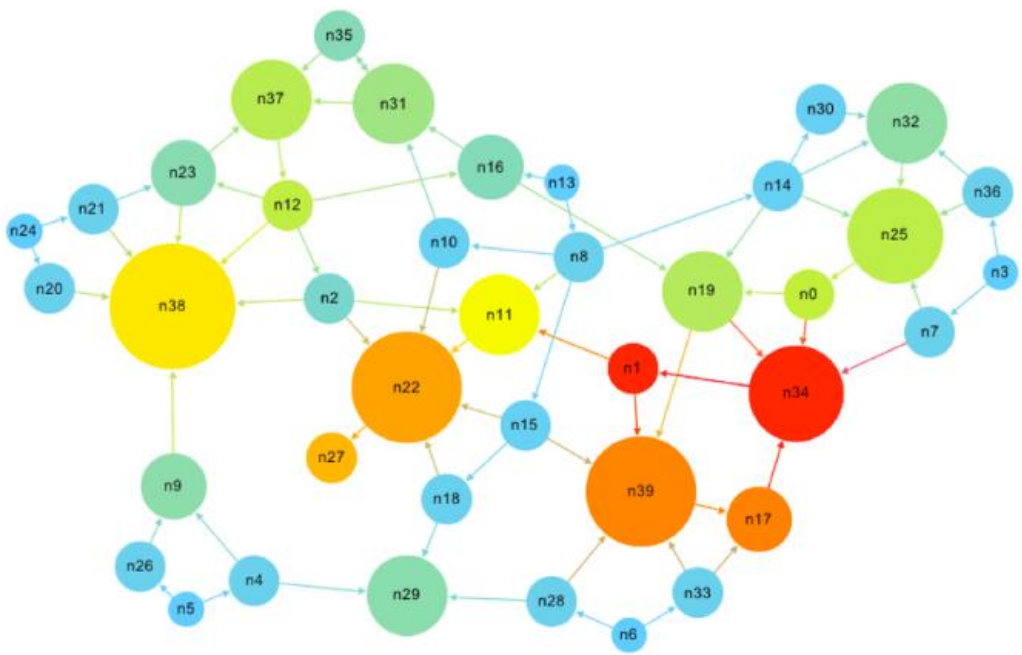


https://demos.algorithmia.com/sitemap
view-source:https://www.iisc.ac.in/

Motivation: Web Crawl & Search

- HTML file content
  - Build inverted index of words to URLs
  - Extract URL and Title
  - Extract links, build graph, find PageRank
  - Word co-occurrence and clustering

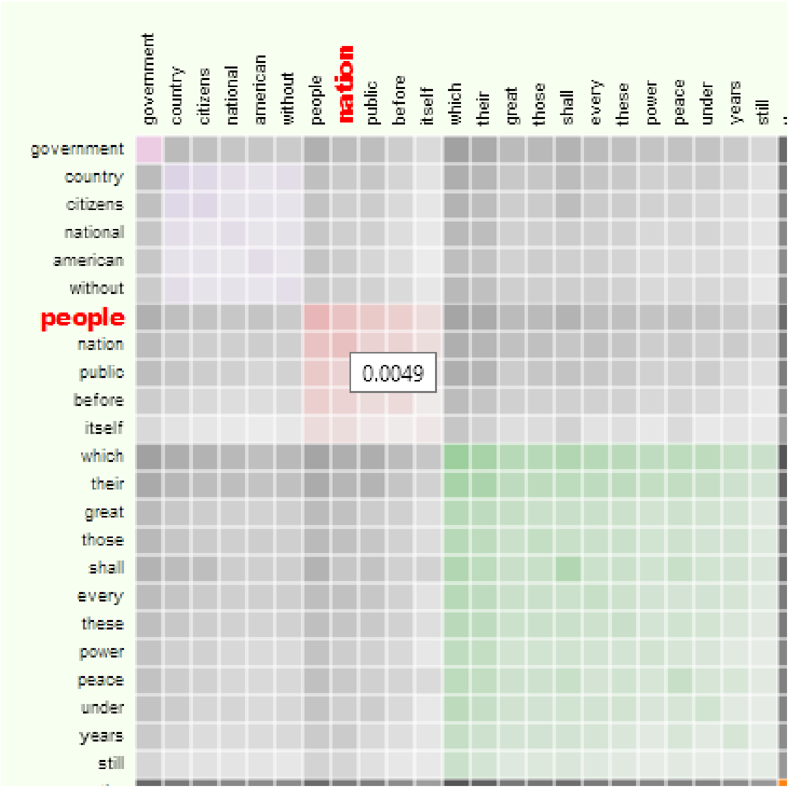
URL	PageRank
u1	0.02
u2	0.3
u3	0.08
u4	0.1
u5	0.2
u6	0.25
u7	0.05



https://medium.com/@a.chachra/implementation-of-page-rank-algorithm-in-python-using-random-walk-method-d61cf136a57f

Motivation: Web Crawl & Search

- HTML file content
  - Build inverted index of words to URLs
  - Extract URL and Title
  - Extract links, build graph, find PageRank
  - Word co-occurrence and associative rule mining



http://hobsonlane.com/pug/pug/miner/static/word_cooccurrence_matrix.html

Motivation: Web Crawl & Search

- Bringing it all together: **Doing a Search**
  - **Lookup** of keyword in inverted index, find common URLs for keywords
  - **Lookup** PageRank of all matching URLs
  - **Sort and Select top *n*** PageRank URLs
  - **Join** top n pages with URL and title
  - **Return** result to user
  - **Suggest** similar searches (co-occurrence)

Keywords

Keyword	URL List		
People	u1	u5	u6
India	u1	u6	
Best	u2	u5	u7
Call	u3	u4	u5
Ishmael	u3		
Some	u3		
Years	u3	u7	
Here	u4		
Number	u4	u6	
Life	u7		

Filter, Intersection

URL	PageRank
u1	0.02
u2	0.3
u3	0.08
u4	0.1
u5	0.2
u6	0.25
u7	0.05

Join, Sort, Select *n*

URL	Title
u1	The Constitution of India
u2	A Tale of Two Cities by Dickens
u3	Project Gutenberg - Moby Dick
u4	Carly Rae Jepsen - Call Me Maybe
u5	Shah Rukh Khan interview
u6	Wikipedia – India's Population
u7	Best Years of My Life Pistol Annies

Join, Return *n*

Google's MapReduce

*"A **simple and powerful interface** that enables **automatic parallelization** and distribution of **large-scale computations**, combined with an implementation of this interface that achieves high performance on **large clusters** of **commodity PCs**."*

Dean and Ghermawat, "MapReduce: Simplified Data Processing on Large Clusters", OSDI, 2004

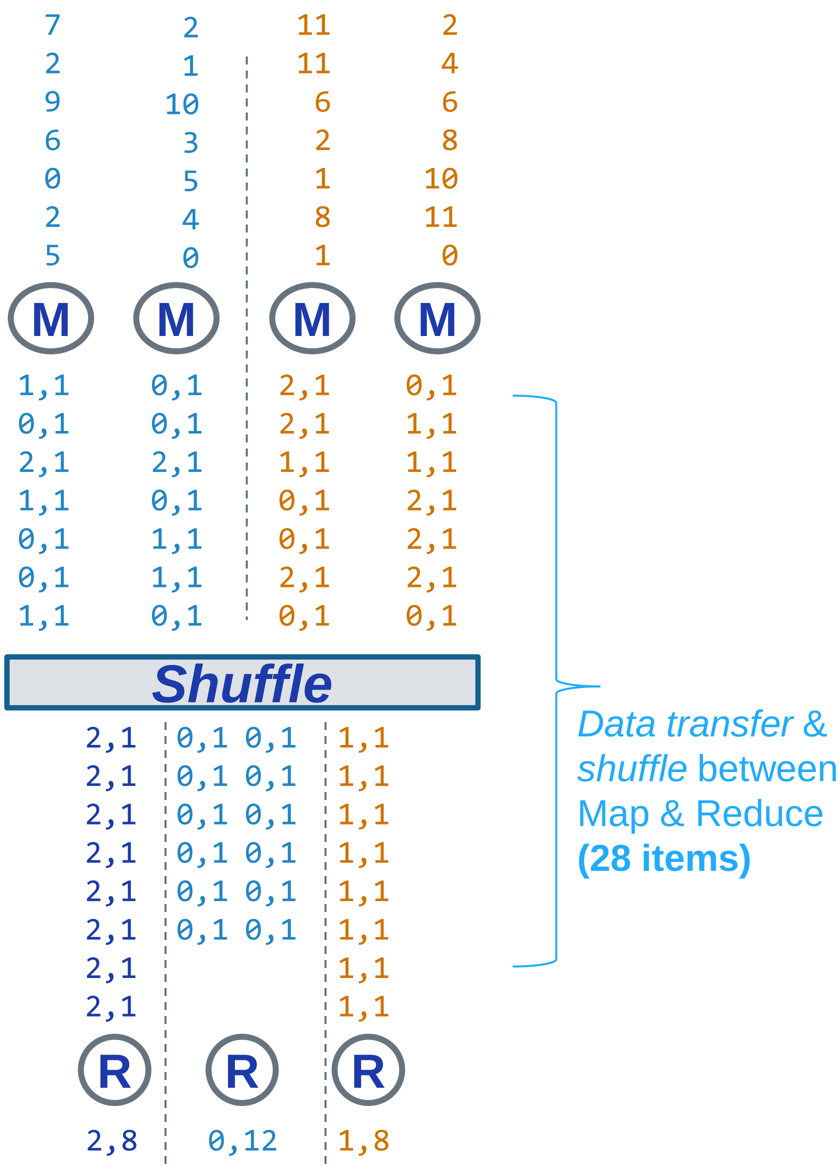
MapReduce Design Pattern

- Programming model for distributed applications
  - Clean abstraction for programmers
  - Automatic parallelization & distribution
- Fault-tolerance
- Batch data processing system
  - Large inputs sizes
- Simple data-intensive applications
  - Distributed Grep: Document list → Occurrence of search term
  - URL Access Frequency: URL access list → <URL, freq>
  - Reverse Web-Link Graph: <target,src> → <src, target[]>
  - Term-Vector per Host: <host,word[]> → <host,<word,freq>[]>

MapReduce: Data-parallel Programming Model

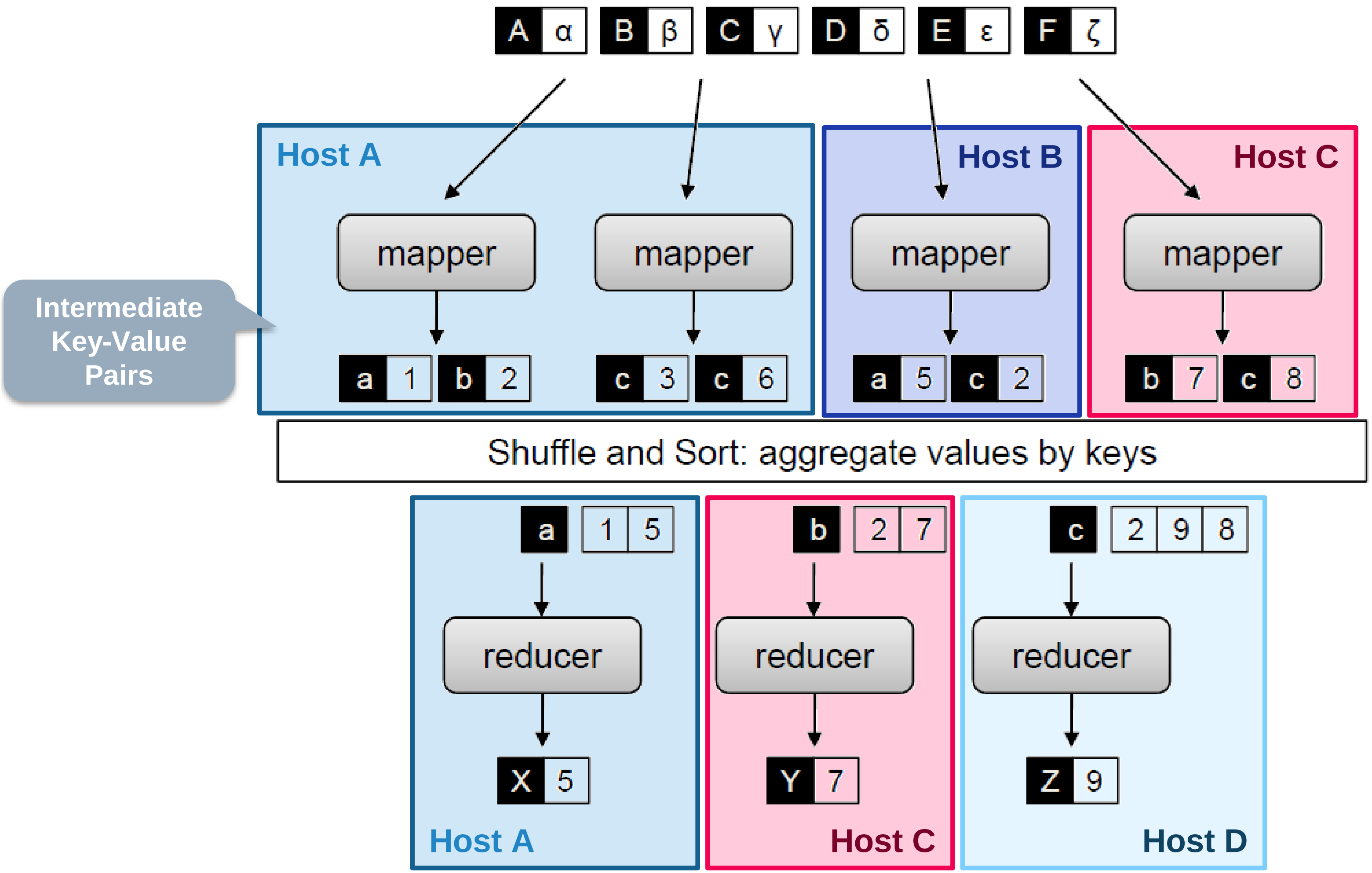
- Process data using **map** & **reduce** user-defined functions
- **map(k_i, v_i) → List<k_m, v_m>**
  - *map* is called once on every input item
  - Emits a series of intermediate key/value pairs
- **shuffle & sort phase**
  - All map output values (**v_m**) with a given key (**k_m**) are *grouped* together, keys *sorted* within a group
  - Happens internally within the framework
- **reduce(k_m, List<v_m>) → List<k_r, v_r>**
  - ***reduce*** is called once on *every unique key & all its values*
  - Emits a value that is added to the output

Histogram using MR




```
int bucketWidth = 4 // input

Map(k, v) {
    emit(floor(v/bucketWidth), 1)
    // <bucketID, 1>
}

// one reduce per bucketID
Reduce(k, v[]){
    sum=0;
    foreach(n in v[])  sum+=n;
    emit(k, sum)
    // <bucketID, frequency>
}
```

Map-*Shuffle-Sort*-**Reduce**

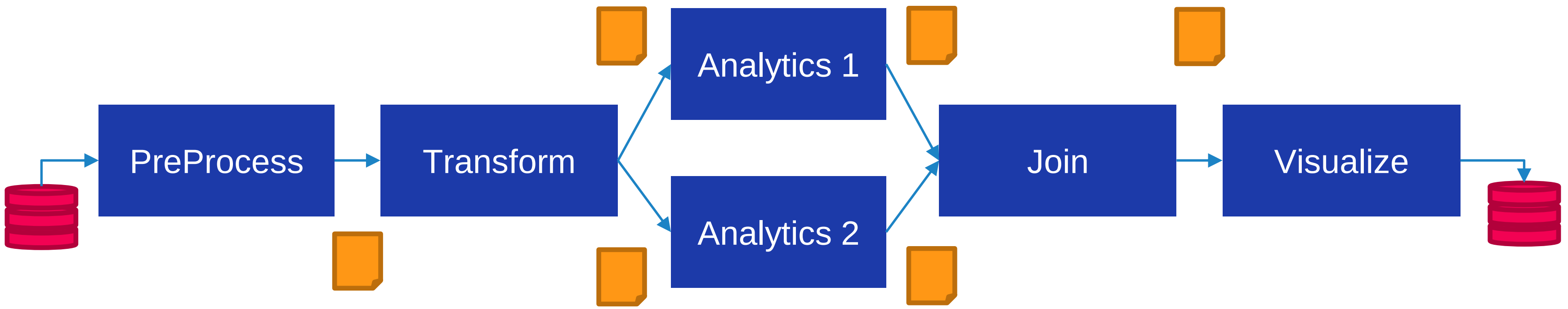


Data-Intensive Text Processing with MapReduce, Jimmy Lin, 2010

Limitations of MapReduce

Limitations in **Expressivity** of MapReduce

- Multi-stage computing not simple
  - Many different jobs
- Complex code for simple transformations
  - Repetitive, not *data centric*

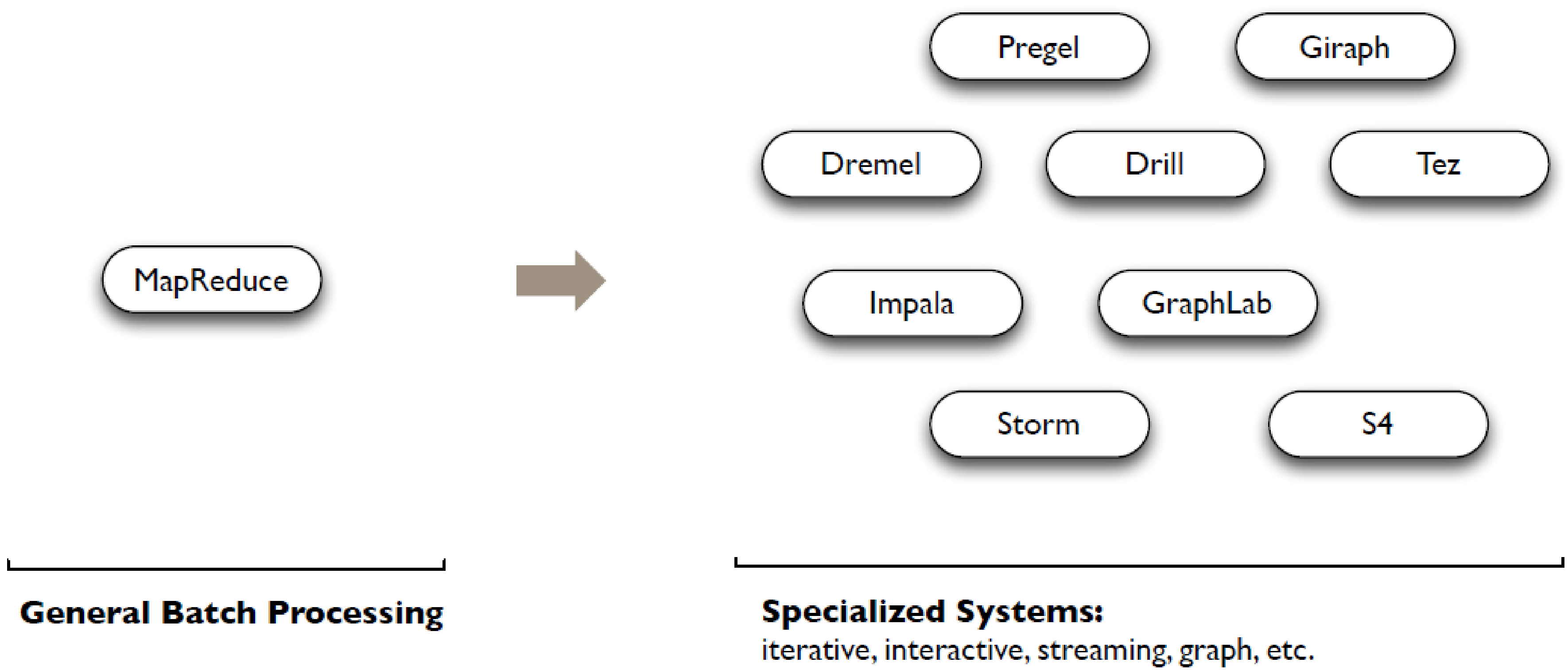


https://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf

Limitations of MapReduce

Limitations in **Expressivity** of MapReduce

- Limited support for non-text, Non-static data

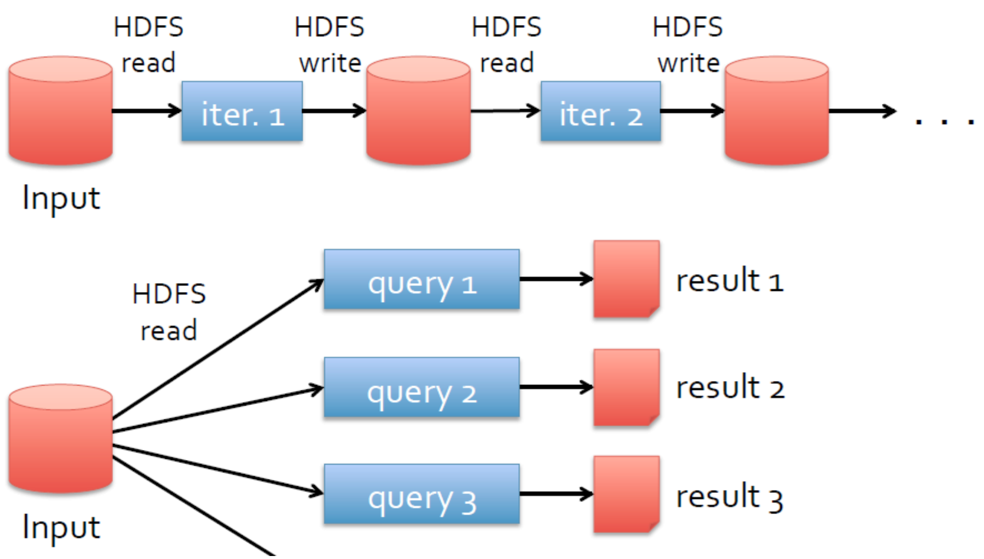


https://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf

Limitations in **Performance** of MapReduce

Limitations of MapReduce

- Poor performance for:
  - Complex, multi--stage applications (e.g. iterative machine learning & graph processing)
  - Interactive *ad hoc* queries



Latency & Bandwidth

- L1 cache reference
- L2 cache reference
- **Main memory reference**
- **Send 1K bytes over 1 Gbps network**
- **Read 4K randomly from SSD***
- **Read 1MB sequentially from memory**
- **Round trip within same datacenter**
- **Read 1MB sequentially from SSD***
- **Send 1MB over 1 Gbps network**
- **Disk seek**
- **Read 1MB sequentially from disk**
- **Send packet CA->NL->CA**

https://gist.github.com/jboner/2841832

Latency & Bandwidth

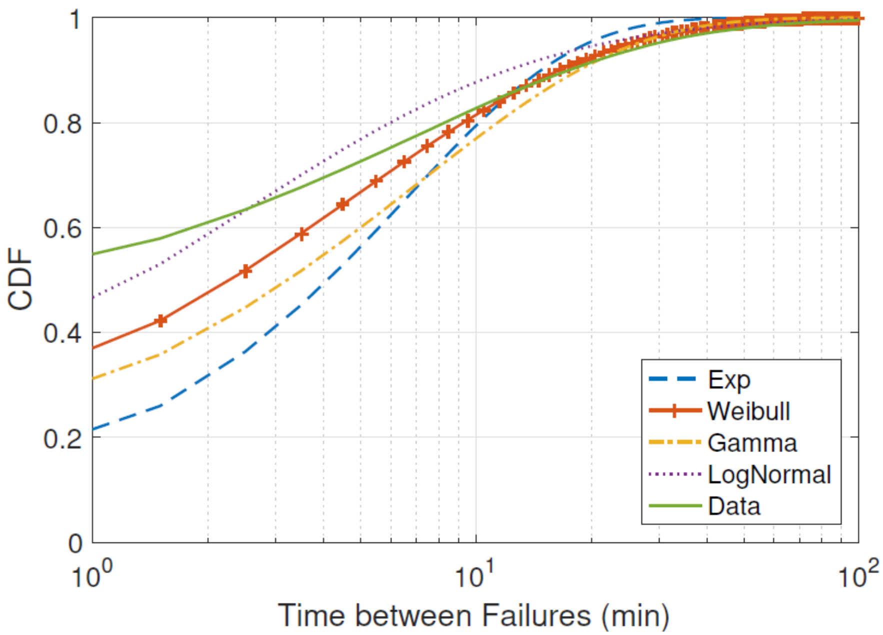
L1 cache reference	0.5 ns		
L2 cache reference	7 ns		
Main memory reference	100 ns		
Send 1K bytes over 1 Gbps network	10,000 ns	10 µs	
Read 4K randomly from SSD*	150,000 ns	150 µs	
Read 1MB sequentially from memory	250,000 ns	250 µs	
Round trip within same datacenter	500,000 ns	500 µs	
Read 1MB sequentially from SSD*	1,000,000 ns	1,000 µs	1 ms
Send 1MB over 1 Gbps network		8,250 µs	8 ms
Disk seek	10,000,000 ns	10,000 µs	10 ms
Read 1MB sequentially from disk	20,000,000 ns	20,000 µs	20 ms
Send packet CA->NL->CA	150,000,000 ns	150,000 µs	150 ms

https://gist.github.com/jboner/2841832

“

Bandwidth of Memory $\gg$ Network or Disk

MTTF in Data Center



"The MTBF (mean time between failures) across all data centers we investigate (with hundreds of thousands of servers) is only 6.8 minutes, while the MTBF in different data centers varies between 32 minutes and 390 minutes."

→ MTBF with 1000 servers is 680mins

→ MTBF with 100 servers is 6800mins (4.7 days)

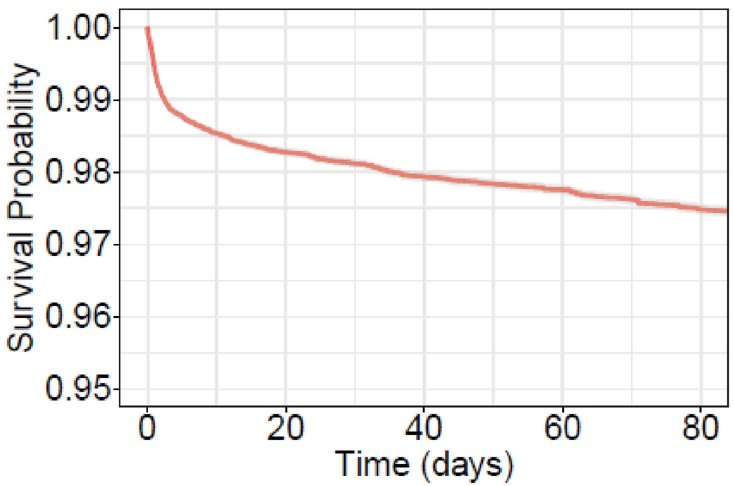


Figure 4: Kaplan Meier survival estimate of datacenter switches, shaded region shows 95% confidence intervals.

Surviving switch failures in cloud datacenters, ACM SIGCOMM Computer Communication Review, Volume 51Issue 2April 2021

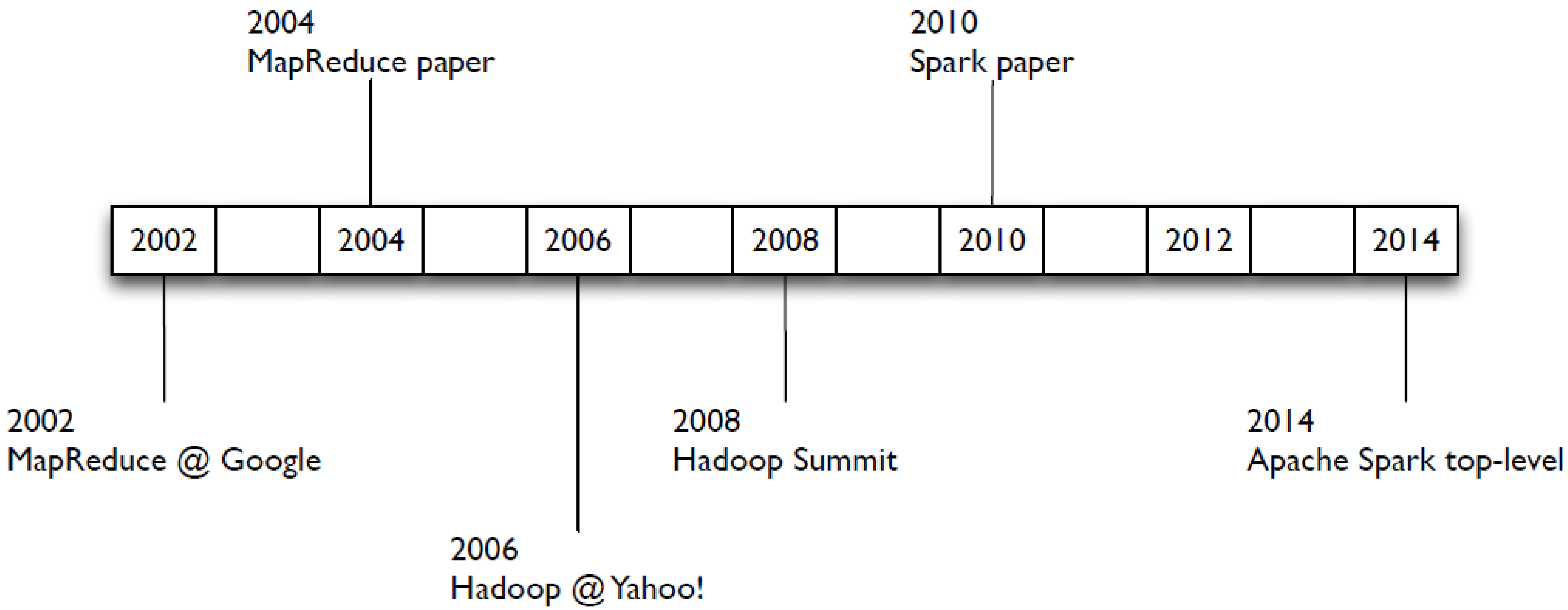
What Can We Learn from Four Years of Data Center Hardware Failures?, DSN, 2017

“

Failures may be infrequent during the lifetime of an application execution

From MapReduce to Spark

- Google's MapReduce
  - Programming Model
  - Apache Hadoop runtime environment



https://stanford.edu/~rezab/sparkclass/slides/itas_workshop.pdf

Apache Spark

Learning Spark
Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia,
O'Reilly, First and Second Editions

https://databricks.com/p/ebook/learning-spark-from-oreilly

“

- *Hands-on in the next class*
  - *Bring your laptops*
- *Access Google Colab using your IISc*
- *Shared URL for Notebook: https://bit.ly/ds221-spark*

The Spark Ecosystem

- Core Spark Engine
  - RDDs, Transformations, Actions, batch processing
- Higher level abstractions
  - Data frames, SQL-like queries
  - Discretized streams, semi-realtime data
  - Machine learning libraries, **Mllib**
  - Linked data analytics, **GraphX**

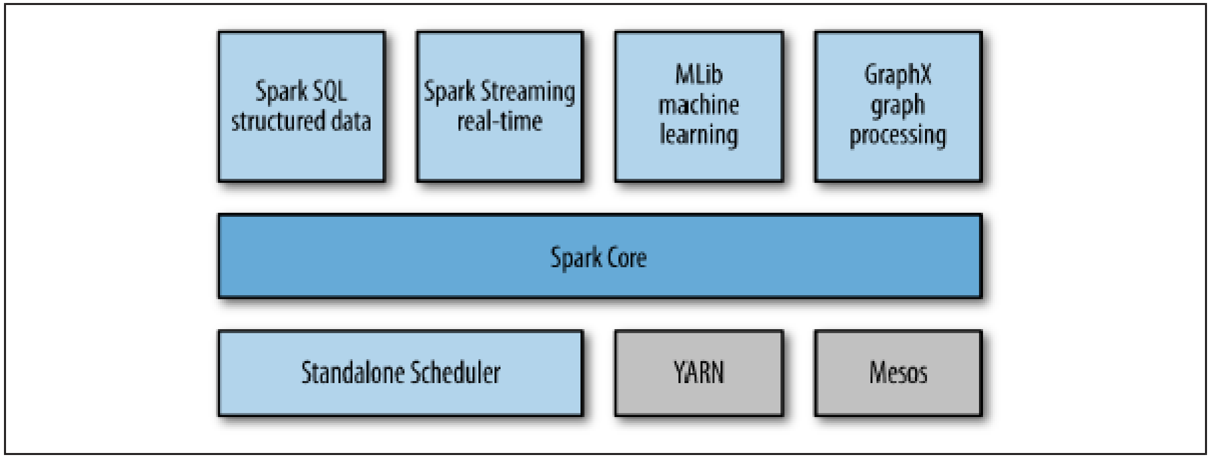
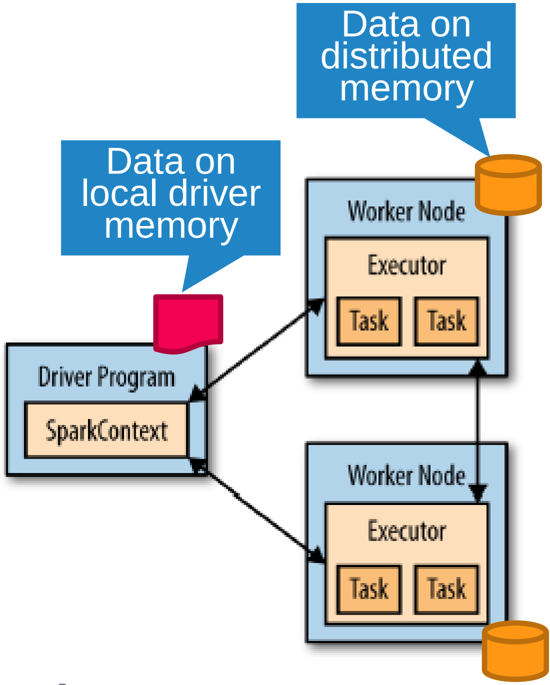


Figure 1-1. The Spark stack

Spark: A Distributed Execution Engine

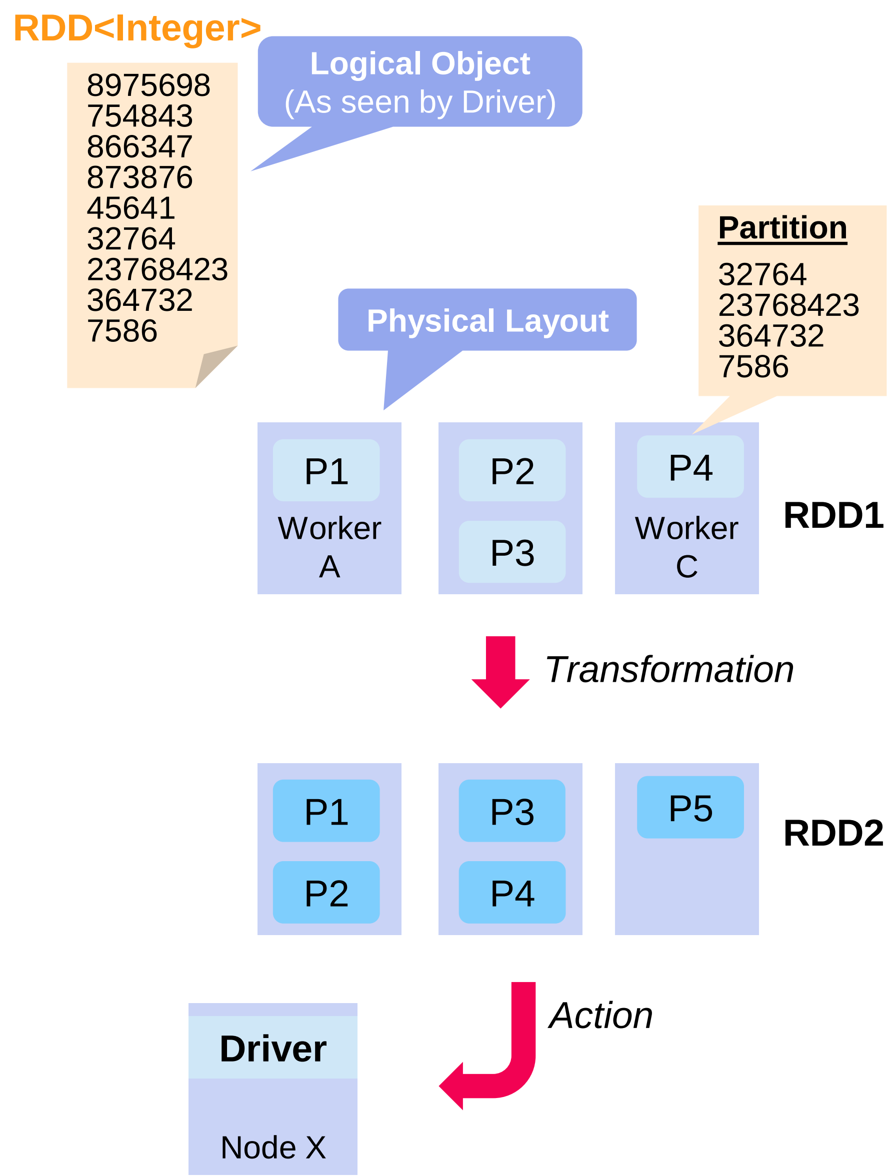
- **Driver**: User program for application, uses Spark Context, local variables
- **Spark Context**: Gives access to distributed computing environment
- **Worker**: Machines on which actual heavy-lift happens
- **Executor**: Spark execution environment in a worker, Process, exclusive to an application
- **Task**: Single operation on data, thread



Spark RDD

Resilient Distributed Dataset

- Collection of homo-geneous objects
  - Order is not preserved*
- **Distributed** on workers
  - 1 or more **Partitions**
- **Read-only**, immutable
- Can be **rebuilt**
- Can be **cached**
- MR like data-parallel operations
  - **Execute** on workers



Creating an RDD

- Create RDD by loading data
  - Can load from HDFS, local vars, filesystem, NoSQL DB, etc.
- Data is loaded on partitions on different workers
- RDD Object offers a logical view of the dataset
- Can perform operations on the object

Example 3-1. Creating an RDD of strings with textFile() in Python

```
>>> lines = sc.textFile("README.md")
```

Example 3-5. parallelize() method in Python

```
lines = sc.parallelize(["pandas", "i like pandas"])
```

Operations on an RDD

- **Transformations**: Creates another RDD, present on distributed workers
- **Actions**: Returns a value, local to the Driver

Example 3-2. Calling the filter() transformation

```
>>> pythonLines = lines.filter(lambda line: "Python" in line)
```

Example 3-3. Calling the first() action

```
>>> pythonLines.first()
u'## Interactive Python Shell'
```

Language Bindings

- Users can provide driver code in multiple languages
  - Scala, Java, Python
- Spark offers equivalent transformations and actions in each language
- Logic within transformations and actions can also be in these languages
- Actual Spark execution environment is in Scala
  - Standard data structure mapping
  - Python code is pickled (de/serialize) and shipped remotely

https://cwiki.apache.org/confluence/display/SPARK/PySpark+Internals

Passing Functions to Spark Operations

- Lambda syntax
  - **Functions** are *input parameters* to other functions
  - Pass short functions concisely, inline

```
pythonLines = lines.filter(lambda line: "Python" in line)
```

```
def hasPython(line):
    return "Python" in line

pythonLines = lines.filter(hasPython)
```

Programming with RDDs

Learning Spark

Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia,
O'Reilly, First Edition

Chapter 3

Basics of Transformations

- Returns a new RDD, computed lazily
- Transforms tend to be **element-wise** operations
  - **Iterate** through each item, apply the operation, e.g. *Filter*
- **Filter** on *inputRDD* does not affect *inputRDD*
  - Returns a new RDD, *warningsRDD*
- **Union** operates on two RDDs
  - One of them is an input parameter

```
inputRDD = sc.textFile("log.txt")
errorsRDD = inputRDD.filter(lambda x: "error" in x)
warningsRDD = inputRDD.filter(lambda x: "warning" in x)
badLinesRDD = errorsRDD.union(warningsRDD)
```

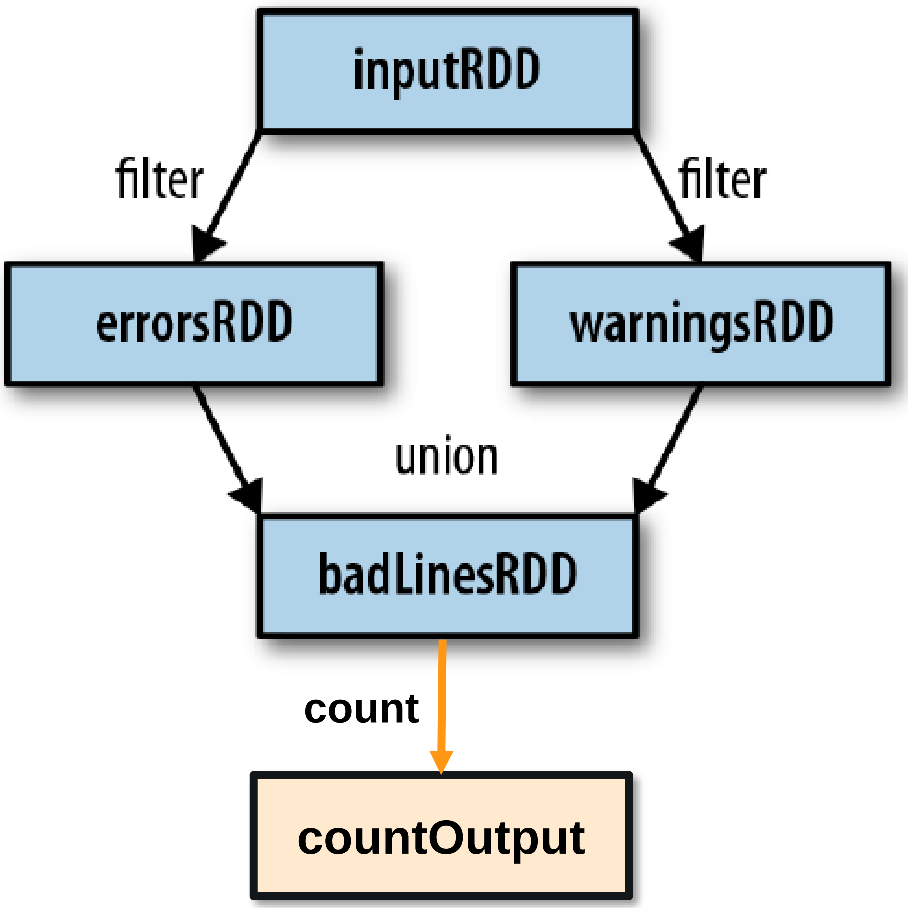
Basics of Actions

- *Actually* triggers operations, returns a final result to driver
  - Force any required transformations to be executed
  - Count, Take, Collect
- Result of action must fit in memory of driver
  - Else, can write RDD to HDFS, saveAsTextFile
- RDDs are computed from scratch when actions are called...See *persist/cache*

```
print "Input had " + badLinesRDD.count() + " concerning lines"
print "Here are 10 examples:"
for line in badLinesRDD.take(10):
    print line
```

Lineage Graph

- Keeps track of operations used to derive an RDD
- Helps *lazily materialize* RDD
- Helps *recover* RDD or their partitions that are lost



Lazy Evaluation

- Transformations are lazily evaluated
  - Calling a transform does NOT immediately execute it
- *Action* **triggers** execution of *dependent transformations*
- E.g., load().map().count()
  - Load & Map do not execute till we see Count
- Allows Spark to reduce the number of passes through the data
  - Materializes RDD only when required
  - Reused RDDs that have been materialized earlier
  - Immutability!

RDD Persistence

Double Exec.

```scala
val result = input.map(x => x*x)
println(result.count())
println(result.collect().mkString(","))
```

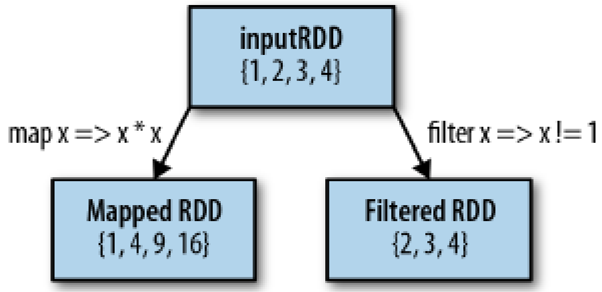
- Dependent RDDs recomputed for each action
- Need to *persist* RDDs to reuse without recompute
- Levels of Persistence
  - Memory (Obj. or Ser.)
    - LRU eviction
  - Memory and Disk (O | S)
    - Spill to disk if less memory
  - Disk only
- Recomputed if node fails or on LRU eviction
- Can manually *unpersist*

Level	Space used	CPU time	In memory	On disk
MEMORY_ONLY	High	Low	Y	N
MEMORY_ONLY_SER	Low	High	Y	N
MEMORY_AND_DISK	High	Medium	Some	Some
MEMORY_AND_DISK_SER	Low	High	Some	Some
DISK_ONLY	Low	High	N	Y

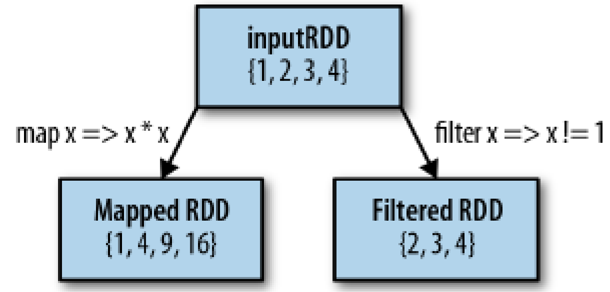
```scala
val result = input.map(x => x * x)
result.persist(StorageLevel.DISK_ONLY)
```

Common Transformations

- Element-wise transformations
- **Filter**
  - Applies conditional logic to each element
  - User logic (lambda fn.) returns true/false
    - If true, input element copies to output RDD
    - if false, input element omitted
  - RDD output type is same as input



Common Transformations



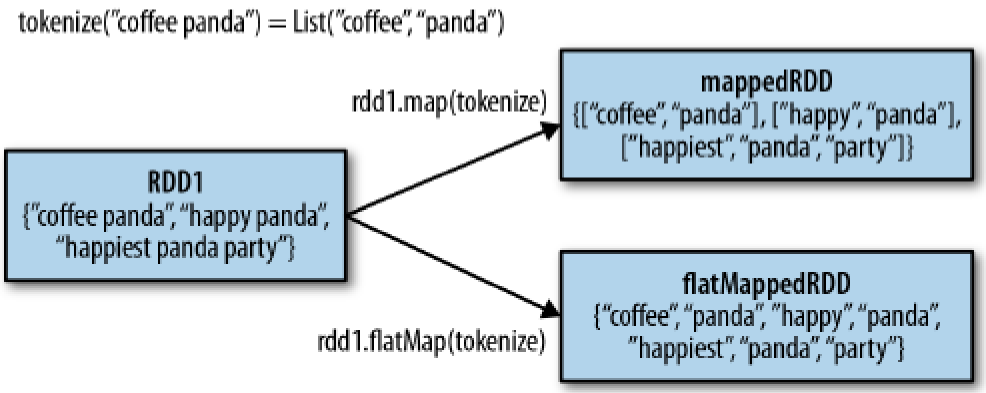
- Element-wise transformations
- **Map**
  - Applies user logic to each element
  - Logic returns exactly one output for each input item
  - RDD output type can be different from input
- Can perform any user operation
  - E.g., Parsing a string, fetching a webpage

Example 3-26. Python squaring the values in an RDD

```
nums = sc.parallelize([1, 2, 3, 4])
squared = nums.map(lambda x: x * x).collect()
for num in squared:
    print "%i " % (num)
```

Common Transformations

- Element-wise transformations
- **FlatMap**
  - Applies user logic to each element
  - Logic returns *zero or more* output items for each input item
  - RDD output type can be different from input



Common Transformations

- Element-wise transformations
- **FlatMap**
  - Applies user logic to each element
  - Logic returns *zero or more* output items for each input item
  - RDD output type can be different from input

Example 3-29. flatMap() in Python, splitting lines into words

```
lines = sc.parallelize(["hello world", "hi"])
words = lines.flatMap(lambda line: line.split(" "))
words.first()  # returns "hello"
```

Filter using FlatMap. Using Map?

RDD2=RDD1.**filter**(item : *foo*(item) {item > 10})

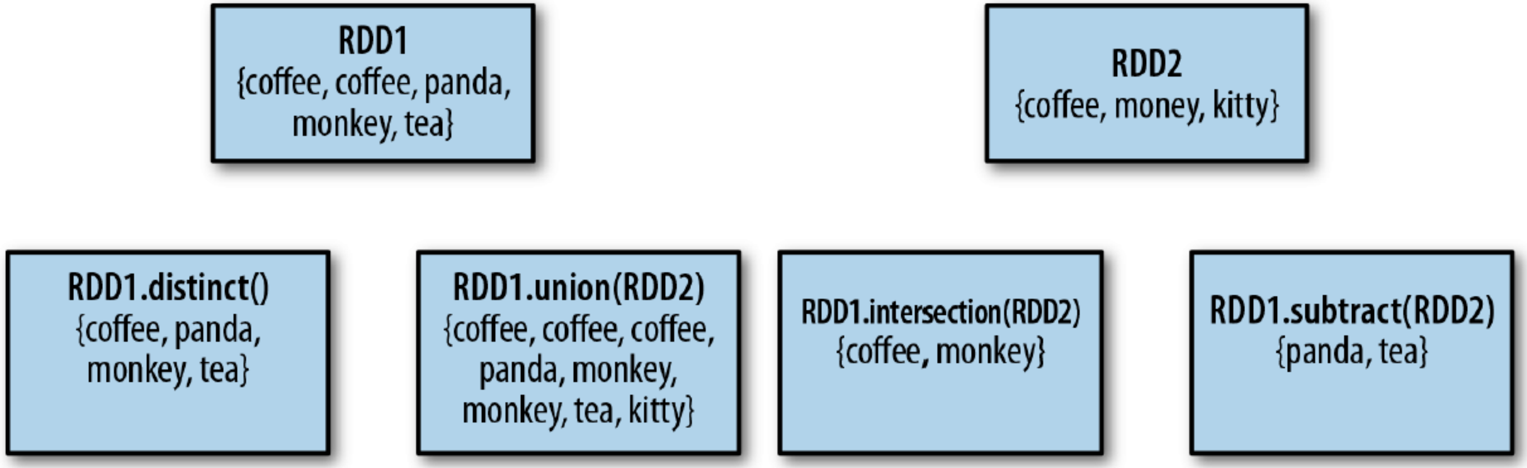
RDD2= RDD1.**flatMap**(item : if(*foo*(item)) then return item)

RDD2= RDD1.**map**(item : if(*foo*(item)) then return item)

[null, item, item, null…]

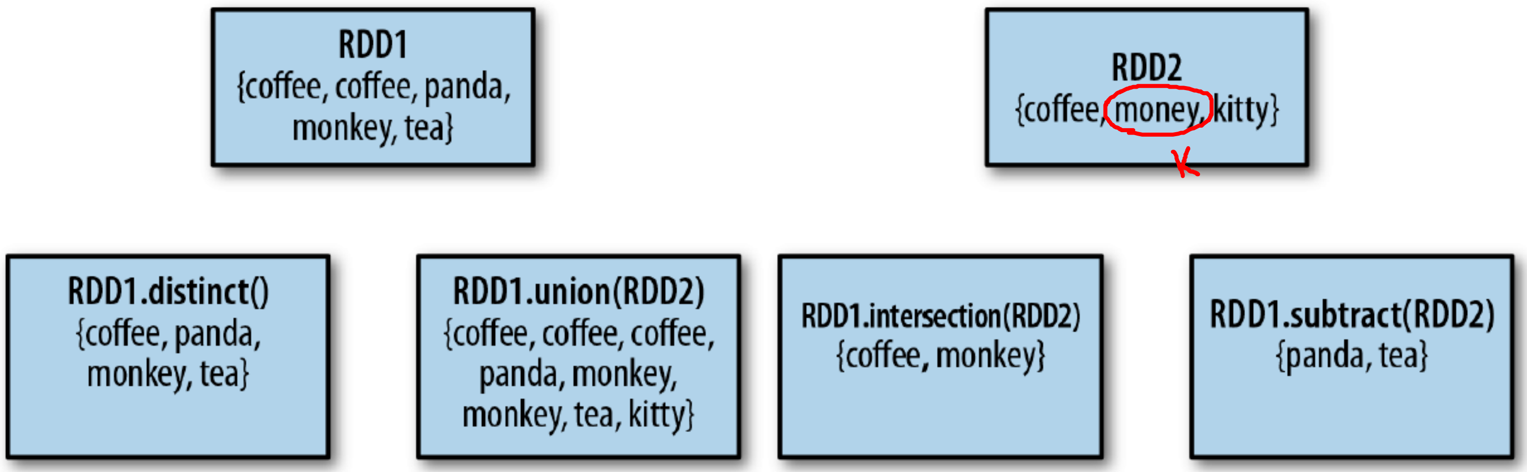
Common Transformations

- Pseudo set operations
- **Distinct**
  - Copy only unique items into output RDD
- **Union**
  - Concatenate items in two RDDs into output RDD
  - Duplicates are NOT removed

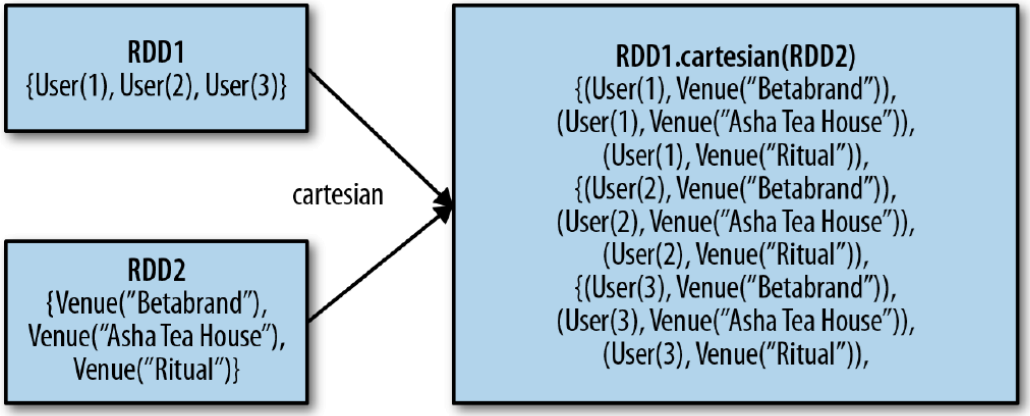


Common Transformations

- Pseudo set operations
- **Intersection**
  - Find common items in two RDDs, and copy into output RDD. Duplicates are removed.
- **Subtraction**
  - Copy items from first RDD into output RDD, except those present in second RDD



Common Transformations



- Pseudo set operations
- **Cartesian Product**
  - All-to-all combination of inputs from 2 RDDs in the output RDD

- **Sample**(withReplace, fraction, seed)
  - Copies a sampled subset of items into output RDD
  - Same fraction sampled from each partition
  - Output count *may not* exactly be (fraction*input count)
  - Seed guarantees same samples *IF* RDD content was not changed, e.g., due to lazy (re)evaluation

Common Actions

- **collect**
  - Returns the entire RDD to driver
- **take**(n)
  - Return *n* items to driver from fewest partitions
  - May not be evenly sampled, not ordered
- **takeOrdered**(num, order?)
  - Return *n* items using ascending (or given) ordering
  - If RDD is sorted, will return smallest *n* sorted items
- **takeSample**(withReplace, num, seed)
  - Return *n* items, sampled evenly from all partitions
  - Assumes each partition has uniform distribution
- **top**(n)
  - For sorted RDD, return largest *n* items.
  - *Opposite order of default ordering in TakeOrdered*

Example

P1	P2	P3
1	3	4
7	2	1
5	1	6
6		2
		5

- **count()**
  - 4+3+5=**12**
- **take(8)**
  - **3,2,1,4,1,6,2,5**
  - Returns items from fewest partitions
- **takeOrdered(4)**
  - **1,1,1,2**
  - Returns *n* items in ascending order
- **top(4)**
  - **7,6,6,5**
  - Returns *n* items in descending order
- **takeSample(6, replace=false)**
  - **1,5,3,1,6,5**
  - Uniformly samples items from each partitions, without picking same item twice
- **takeSample(6, replace=true)**
  - **1,5,2,2,6,5**
  - Uniformly samples items from each partitions, allowing same item to be picked twice

```python
# Convert our RDD of strings to numeric data so we can compute stats and
# remove the outliers.
distanceNumerics = distances.map(lambda string: float(string))
stats = distanceNumerics.stats()
mean = stats.mean()
```

Numeric RDD

- Common statistics for RDDs having numeric types
- Single **stats**() action to populate all stats
- Individual functions (actions) also available

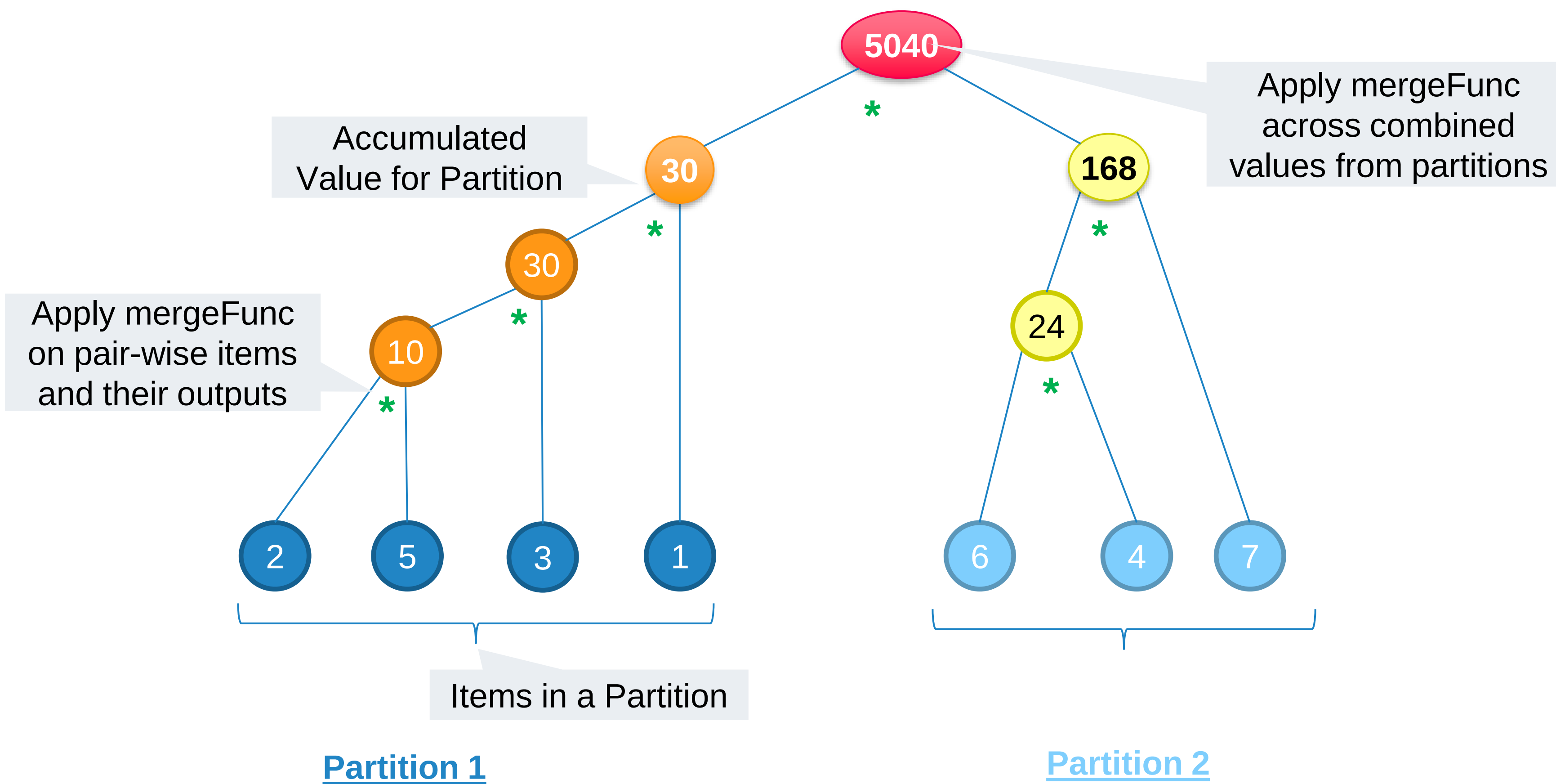
Table 6-2. Summary statistics available from StatsCounter

Method	Meaning
`count()`	Number of elements in the RDD
`mean()`	Average of the elements
`sum()`	Total
`max()`	Maximum value
`min()`	Minimum value
`variance()`	Variance of the elements
`sampleVariance()`	Variance of the elements, computed for a sample
`stdev()`	Standard deviation
`sampleStdev()`	Sample standard deviation

Common Actions

```
sum = rdd.reduce(lambda x, y: x + y)
prod = rdd.reduce(lambda x, y :x * y)
```

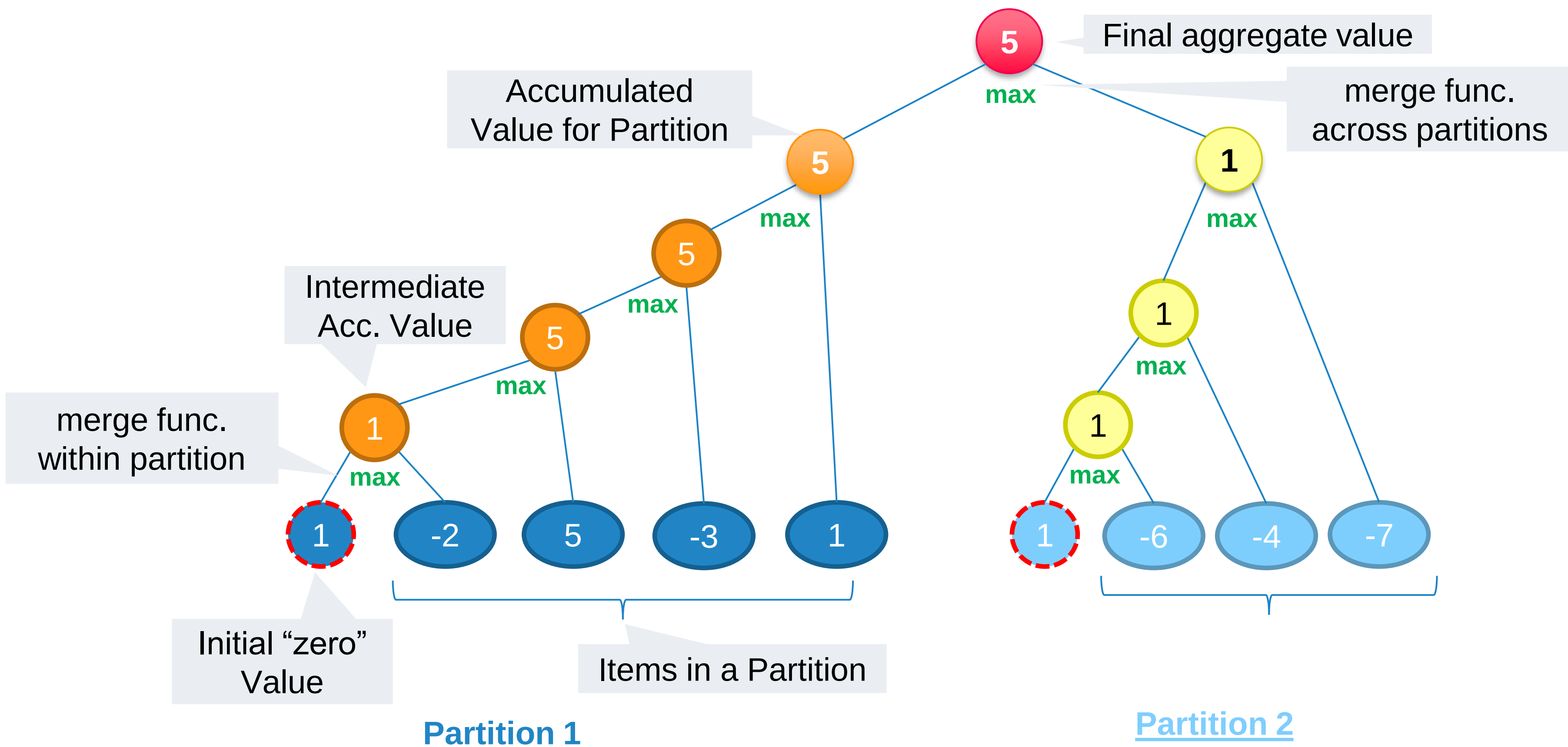
- **reduce**(*mergeFunc*)
  - Combines items in an RDD using an aggregation function
    - *mergeFunc* output type same as input type
    - *mergeFunc* must be Commutative and Associative
    - *mergeFunc* also applied on outputs from each partitions



Common Actions

```
prod = rdd.fold(1, lambda x, y :x * y)
mx = rdd.fold(1, lambda x, y :max(x,y))
```

- **fold**(*zeroVal, mergeFunc)*
  - Similar to reduce, but takes a *zeroValue* as initial accumulator per partition
    - Can have side-effects per (empty) partition!
  - Can be used as a threshold, e.g. avoid divide by zero



Common Actions

- **aggregate**(*zeroVal*, *mergeFunc*, *combineFunc*)
  - acc=zeroVal, acc=mergeFunc(acc, value), acc=combineFunc(acc1, acc2)
  - Combines items in RDD but can have different intermediate and output type from the input
  - Same as fold if *mergeFunc* and *combineFunc* are same

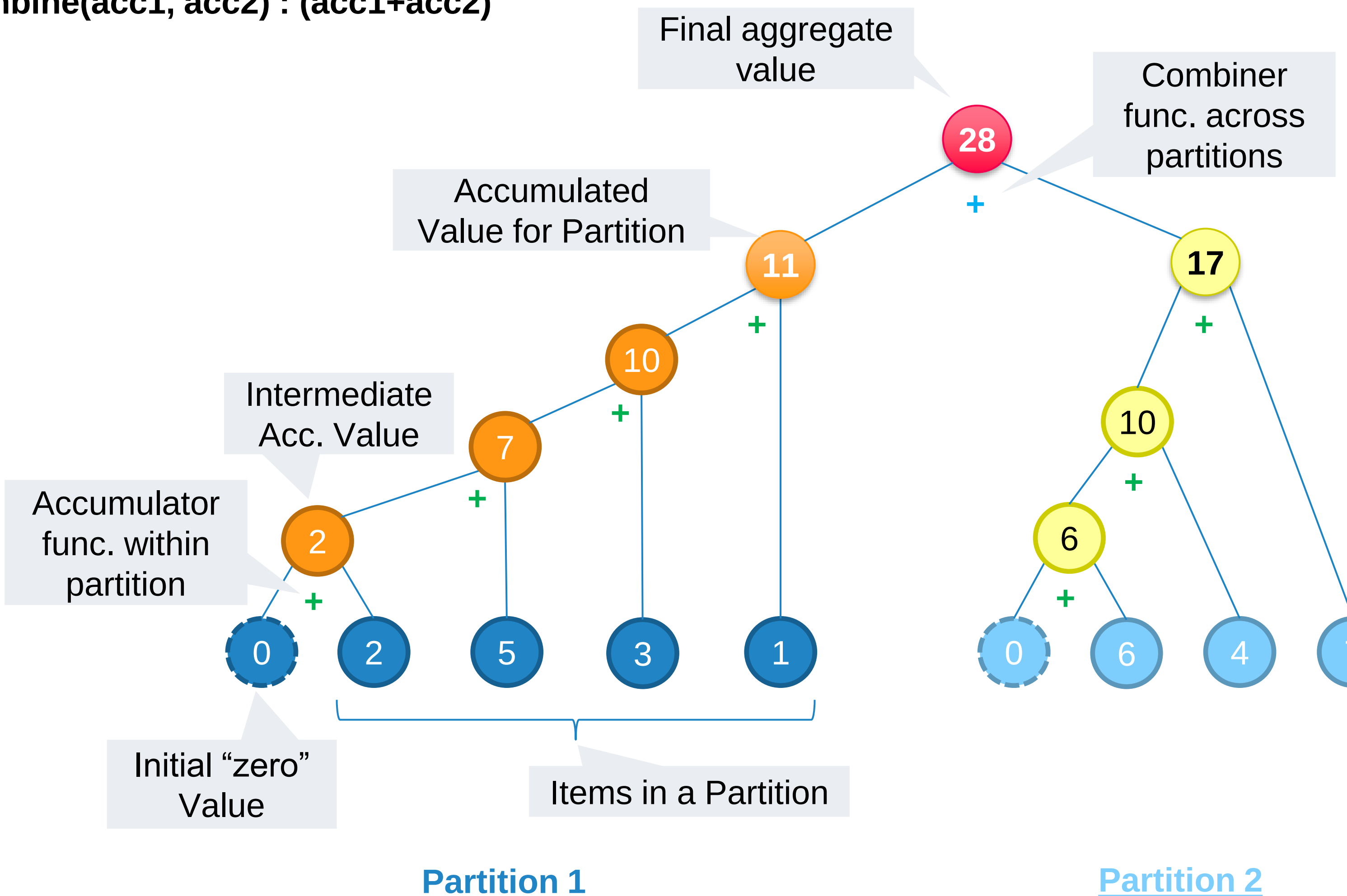
```
sum = nums.aggregate(0,
                    lambda x, y :x + y,
                    lambda x, y :x + y)
```

```
strs = sc.parallelize(['ababab','ab', 'abcd'])
strs.aggregate(0, lambda acc,v : acc+len(v), lambda a1,a2: a1+a2)
```

```
sumCount = nums.aggregate((0, 0),
             lambda acc, val : (acc[0] + val, acc[1] + 1)
             lambda acc1, acc2 : (acc1[0] + acc2[0], acc1[1] + acc2[1])
                    )
return sumCount[0] / float(sumCount[1])
```

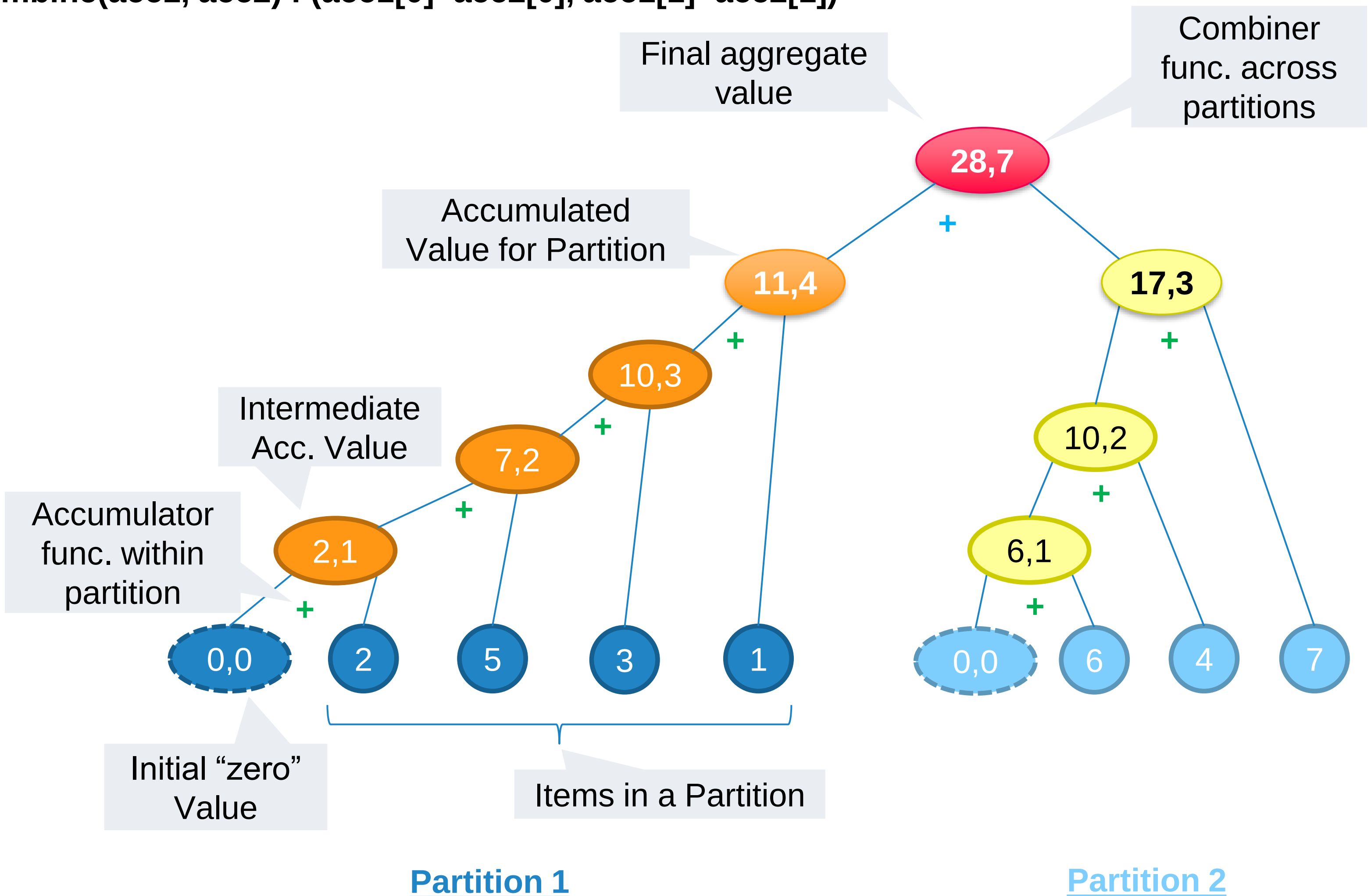
Aggregate: Incremental Evaluation within and across Partitions

zeroVal: *default for data type*
merge(acc, val) : (acc+val)
combine(acc1, acc2) : (acc1+acc2)



Aggregate: Incremental Evaluation within and across Partitions

zeroVal: (0,0)
merge(acc, val) : (acc[0]+val, acc[1]+1)
combine(acc1, acc2) : (acc1[0]+acc2[0], acc1[1]+acc2[1])



Common Actions

- **forEach**(fn)
  - Iterates through each item and applied function
  - Function needs to persist it. Not returned to driver.
- **count**
  - Returns the number of items in collection
- **countByValue**
  - Returns frequency of unique values, {val, count}

Working with Key/Value Pairs

Learning Spark

Holden Karau, Andy Konwinski, Patrick Wendell & Matei Zaharia,
O'Reilly, First Edition

Chapter 4

<Key, Value> RDDs (or) Pair RDD

- Has a key and associated value
  - Key is not distinct. Single value for each key.
- Used to perform aggregate operations
  - Pair RDD exposes additional transformation and actions
  - Derives from base RDD. All base operations supported.
- Use ETL to get your data into Pair RDD type
  - Enables join, reduce by key, data parallel operations by key

Creating Pair RDD

- Create by applying a *map* transform on an RDD
  - Return a Pair of (key, value) or a Tuple2 object

Python

```
pairs = lines.map(lambda x: (x.split(" ")[0], x))
```

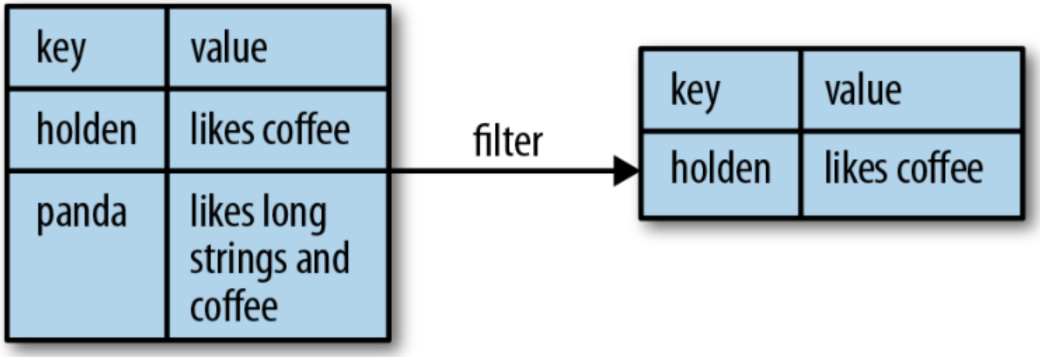
Java

```
PairFunction<String, String, String> keyData =
  new PairFunction<String, String, String>() {
  public Tuple2<String, String> call(String x) {
    return new Tuple2(x.split(" ")[0], x);
  }
};
JavaPairRDD<String, String> pairs = lines.mapToPair(keyData);
```

Transformations on Pair RDDs

- All operations of regular RDDs
  - Each item is a (Key,Value) pair
  - Special *MapValues* transform to operate only on vals

```
result = pairs.filter(lambda keyValue: len(keyValue[1]) < 20)
```



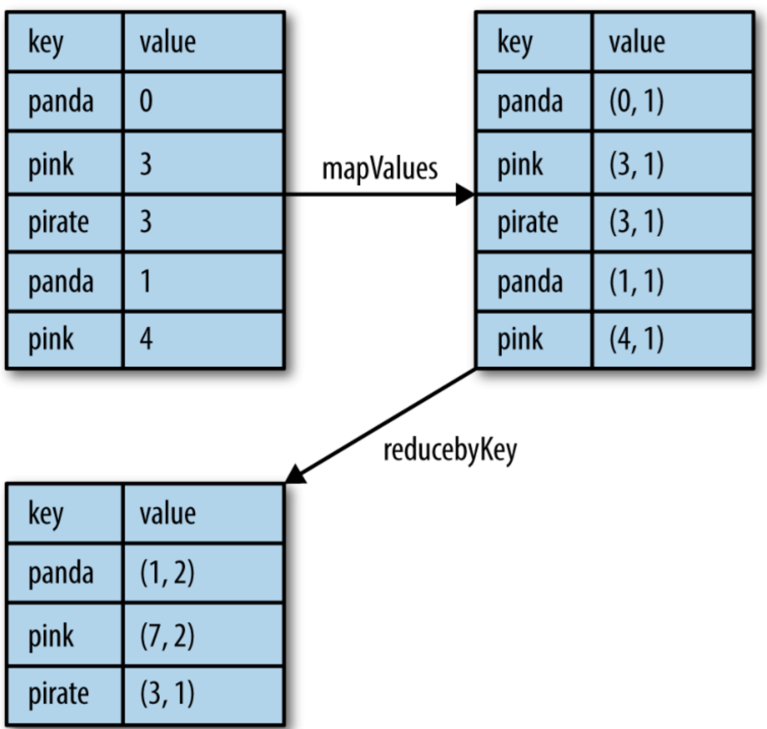
- Transformations on single Pair RDDs
  - *Aggregation, Grouping, Sorting*
- Transformations on two Pair RDDs: *Join*

Aggregation Transforms on a Pair RDD

- **reduceByKey**(*mergeFunc)*
  - Combines the values, after grouping by key
  - Automatically triggers map-side *combiner*

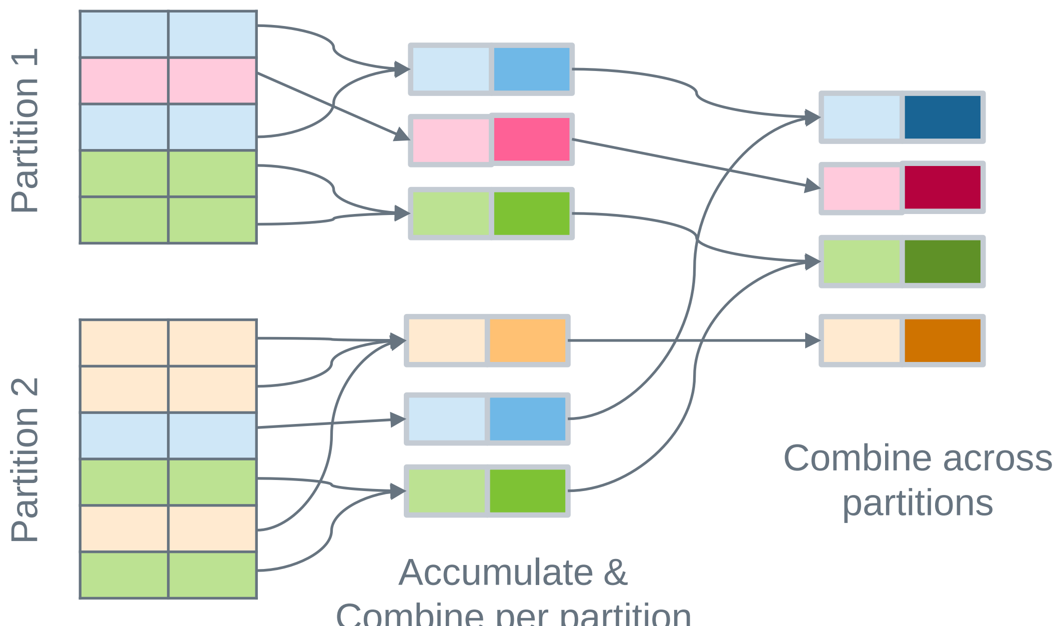
```
rdd.mapValues(lambda x: (x, 1)).reduceByKey(lambda x, y: (x[0] + y[0], x[1] + y[1]))
```

Finding the Average per Key



Aggregation Transforms on a Pair RDD

- **combineByKey** (*createCombiner, mergeValueFunc, mergeCombinersFunc, partitioner*)
  - Iterates through each (K,V) pair, on each partition
  - Accumulates values per key per partition
  - Combines accumulated values per key, across partitions



Aggregation Transforms on a Pair RDD

- **combineByKey** (*createCombiner*, *mergeValue*, *mergeCombiners*, *partitioner*)
  - *createCombiner*: Function called the first time a key is seen on each partition. Initializes the *accumulator* value for that key.
  - *mergeValue*: Function called for each subsequent value for a key on a partition. Merges value with current accumulator's value.
  - *mergeCombiners*: Function used to combine accumulator values for the same key from multiple partitions
- **reduceByKey** is just combineByKey with default functions. *createCombiner* initialized to same type as *value*. Input fn is used both as *mergeValue* and *mergeCombiner*

Per Key Average using combineByKey

Create Combiner, Once per key per partition

Merge Value, Once per value for a key in a partition

```
sumCount = nums.combineByKey((lambda x: (x,1)),
                             (lambda x, y: (x[0] + y, x[1] + 1)),
                             (lambda x, y: (x[0] + y[0], x[1] + y[1])))
sumCount.map(lambda key, xy: (key, xy[0]/xy[1])).collectAsMap()
```

Accumulator Sum Count Value

Merge Accumulators for a key, Across partitions

- ▷ Each value is a (sum, count)
- ▷ Combiner initializes sum to first value *x*, sets count to *1*
- ▷ Accumulator sums the values, increments the count for each value for a key
- ▷ Merge accumulators across partitions by adding their sums and their counts

Per Key Average using combineByKey

```
def createCombiner(value):
  (value, 1)

def mergeValue(acc, value):
  (acc[0] + value, acc[1] +1)

def mergeCombiners(acc1, acc2):
  (acc1[0] + acc2[0], acc1[1] + acc2[1])
```

User provided functions

Partition 1

coffee	1
coffee	2
panda	3

Partition 2

coffee	9

```
Partition 1 trace:
(coffee, 1) -> new key
accumulators[coffee] = createCombiner(1)
(coffee, 2) -> existing key
accumulators[coffee] = merge Value(accumulators[coffee], 2)
(panda, 3) -> new key
accumulators[panda] = createCombiner(3)

Partition 2 trace:
(coffee, 9) -> new key
accumulators[coffee] = createCombiner(9)

Merge Partitions:
mergeCombiners(partition1.accumulators[coffee],
               partition2.accumulators[coffee])
```

Execution within combineByKey

Grouping Transforms on a Pair RDD

- **groupByKey**
  - Groups all values for each key, {Key, Iterator<Value>}
  - Returns an iterator over values for each key
  - User can perform *map*, etc. to operate over values
- pair_rdd1.**cogroup**(pair_rdd2)
  - Combines values for two RDDs having the same key
  - Returns <key, (iter1, iter2)>
  - If key is missing in an RDD, its iterator is empty
  - Can also work on more than 2 RDDs
  - {(a, 2), (c, 4), (c, 6)} # {(c, 9),(b,7)} = {(a,([2],[])), (b,[],[7]), (c, ([4, 6],[9]))}
- **subtract**(pair_rdd2)
  - Removes entries from RDD1 where the same key is also present in RDD2
  - {(1, 2), (3, 4), (3, 6)} - {(3, 9)} = {(1,2)}

Stratified Sampling

- **sampleByKey**(*withReplace*, *keyFractions*, *seed*)
  - *keyFractions* is a map of $\langle k, f_k \rangle$
  - Sample approximately $\lceil f_k \times n_k \rceil$ items, where f_k is the fraction of values for key k, and n_k is the number of key-value pairs for key k
  - Return *n* items where $\mathrm{n} \approx \sum_k \lceil f_k \times n_k \rceil$, sampled evenly from all partitions

Join Transforms on two Pair RDDs

- **Join**
  - Performs inner join
  - Only keys in both RDDs are joined and returned
  - Cross product of values for same key from both RDDs
    - $\{(1, 2), (3, 4), (3, 6)\} \bowtie \{(3, 9)\} = \{(3, (4, 9)), (3, (6, 9))\}$
- **Left Outer Join**
  - Returns an entry for all keys in first RDD
    - $\{(1, 2), (3, 4), (3, 6)\} \bowtie \{(3, 9)\} = \{(1,(2,\mathit{None})), (3,(4,9)), (3,(6,9))\}$
- **Right Outer Join**
  - Returns an entry for all keys in other RDD
    - $\{(1, 2), (3, 4), (3, 6)\} \bowtie \{(3, 9)\} = \{(3,(4,9)), (3,(6,9))\}$

Sorting Transforms on a Pair RDD

- Sorting useful just before returning result
  - Collect, Save
- **sortByKey**: Sorting done by key for Pair RDD
  - Default is ascending. Values are NOT considered.
- Key function can be used to transform key to apply its default comparator
  - E.g., treat *number* key as a *string* key

```
rdd.sortByKey(ascending=True, numPartitions=None, keyfunc = lambda x: str(x))
```

- {(1,2), (3,6), (3,5), (2, 4)} → {(1,2), (2,4), (3,6), (3,5)}

http://codingjunkie.net/spark-secondary-sort/

Actions on a Pair RDD

- All normal RDD actions can be done
- In addition, some special actions
  - **countByKey**: Returns a count for each key as (key,count)
  - **collectAsMap**: Returns the RDD as a native Dictionary or Map object
  - **lookup**(key): Returns *all* the value(s) for a specific key

Summary

- Load data from diverse sources to form RDDs
- Different types of data transformations and actions using Spark
  - Helps to process large datasets, across 10s of machines **at scale**
- Put together data analytics pipelines, ETL pipelines
  - Operate and structured and semi-structured data
  - Data preparation and analytics
  - Complex workflows

Using Spark RDD for Web Crawl & Search

ETL for Web Crawl & Search

- Crawl the web and store files into HDFS
  - Append each URL+HTML file as a “record” in HDFS
- Load RDD with URL as key as HTML content as value
- Parse the HTML file and extract <title>
  - <url>,<title>

```
titleRdd = HTMLRdd.mapValue(html :parseOutTitle(html))
```

URL	Title
u1	The Constitution of India
u2	A Tale of Two Cities by Dickens
u3	Project Gutenberg - Moby Dick
u4	Carly Rae Jepsen - Call Me Maybe
u5	Shah Rukh Khan interview
u6	Wikipedia – India's Population
u7	Best Years of My Life Pistol Annies

https://commoncrawl.github.io/cc-crawl-statistics/plots/crawlermetrics

ETL for Web Crawl & Search

- Parse the HTML file and extract <a href> URL links
  - <url>, List<url>[ ]

```
links = HTMLRdd.mapValue(html : parseOutLinks(html))
```

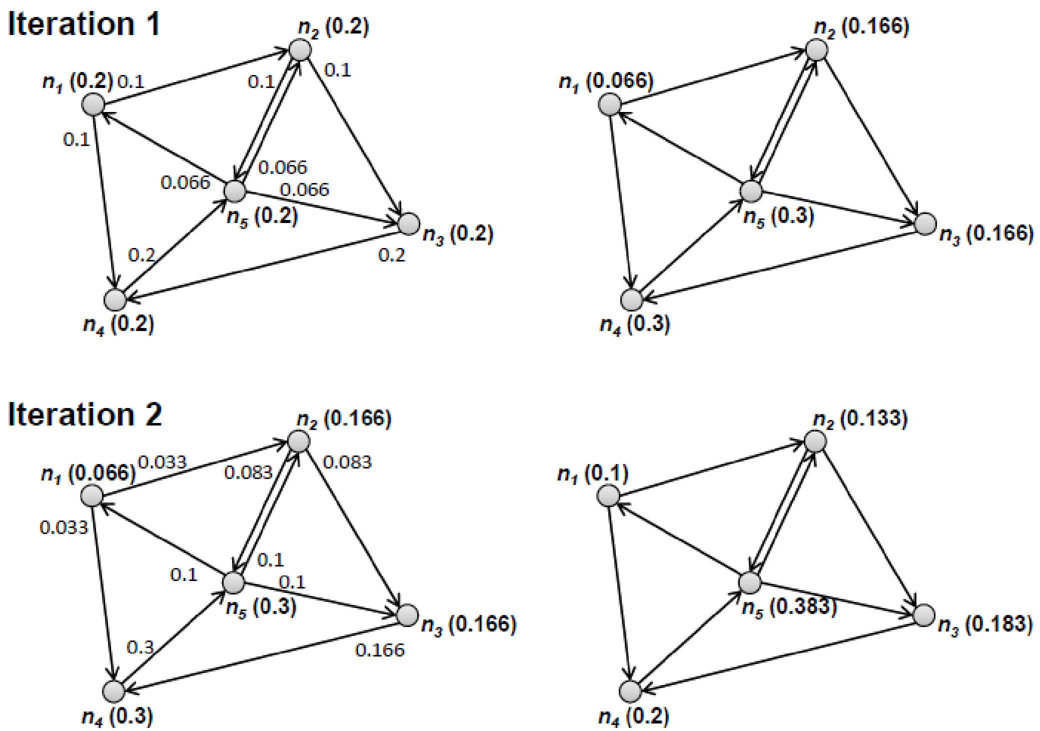
- Run PageRank on the Adjacency List
  - <url>, PageRank
  - *How?*

URL	PageRank
u1	0.02
u2	0.3
u3	0.08
u4	0.1
u5	0.2
u6	0.25
u7	0.05

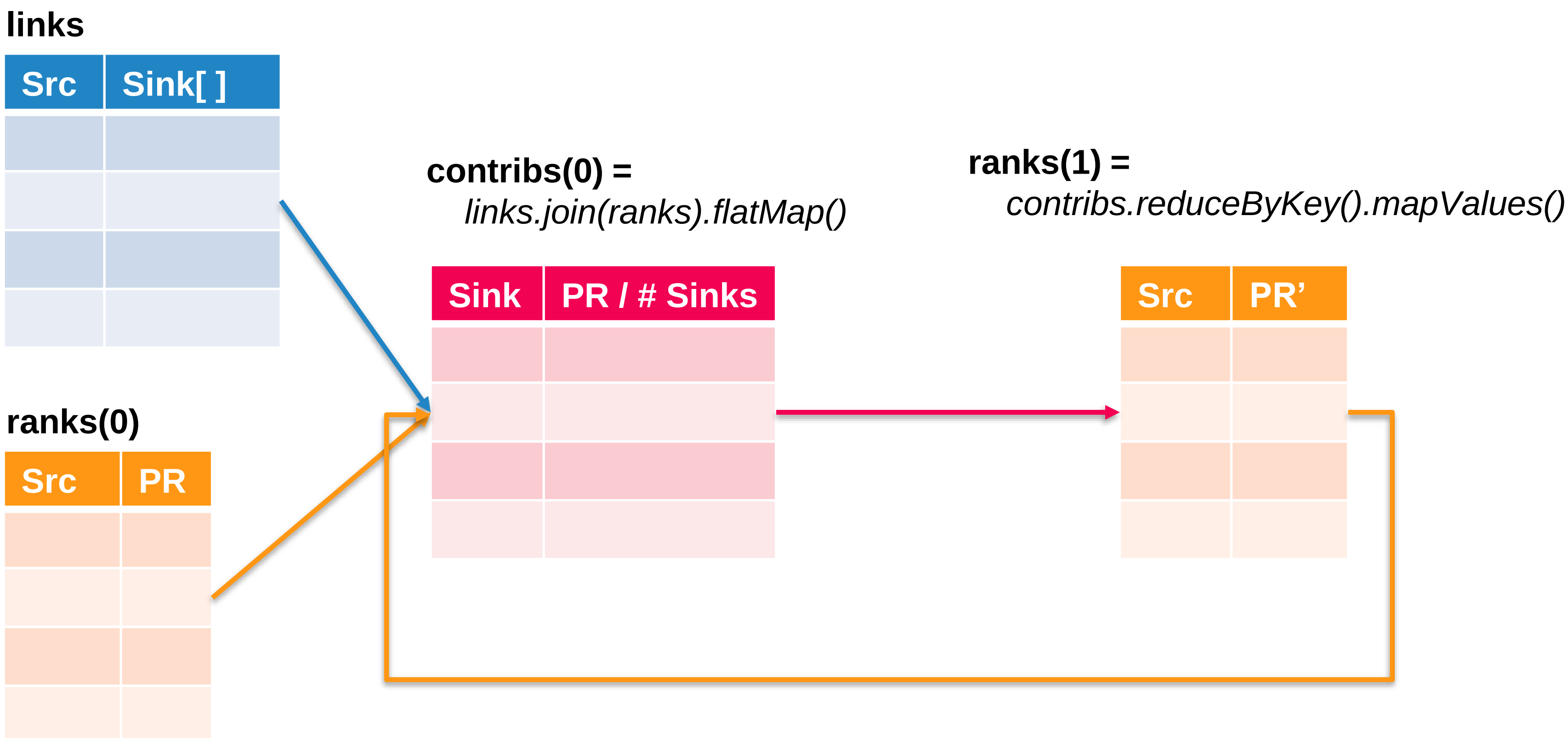
PageRank

$$P(n) = \alpha \left(\frac{1}{|G|} \right) + (1 - \alpha) \sum_{m \in L(n)} \frac{P(m)}{C(m)}$$

- Vertex Centrality metric. Importance of a vertex.
- Calculated iteratively
- Rank of vertex (**n**) depends on rank of neighbors (**L(n)**), normalized by # of out edges for neighbors (**C(m)**)



PageRank using Spark



PageRank using Spark

```
def computeContribs(sink_urls, src_rank):
    for sink_url in sink_urls:
        yield (sink_url, src_rank / len(sink_urls))
---------------------------------------------------------------
# Loads all URLs with other URL(s) link
# and initialize ranks of them to 1.0
ranks = links.map(lambda (src, sinks): (src, 1.0))

# Calculates and updates URL ranks continuously using PageRank algorithm.
for iteration in range(30):
    # Calculates URL contributions to the rank of other URLs.
    contribs = links.join(ranks).flatMap(lambda src_sinks_rank:
                    computeContribs(src_sinks_rank[1][0], src_sinks_rank[1][1]))

    # Re-calculates URL ranks based on neighbor contributions.
    ranks = contribs.reduceByKey(add).mapValues(lambda rank: rank*0.85 + 0.15)

# Collects all URL ranks and dump them to console.
for (link, rank) in ranks.collect():
    print("%s has rank: %s." % (link, rank))
```

https://github.com/apache/spark/blob/master/examples/src/main/python/pagerank.py

ETL for Web Crawl & Search

- Parse the HTML file and extract list of words
  - <url>, <words>

```
HTMLKeyRdd = HTMLRdd.flatMap((url, html) :
(url, html.parseOutKeyWords()))
```

- Remove stop words, etc. Identify keywords
  - <url>, <words>
  - <keyword>, <url>

```
HTMLOkKeyRdd = HTMLKeyRdd.filter((url, keys)
: keys NOT IN STOP_LIST)
```

Keyword	URL List		
People	u1	u5	u6
India	u1	u6	
Best	u2	u5	u7
Call	u3	u4	u5
Ishmael	u3		
Some	u3		
Years	u3	u7	
Here	u4		
Number	u4	u6	
Life	u7		

ETL for Web Crawl & Search

- Build inverted index from keywords
  - <keyword>, List<url>[ ]

```
keyUrlRdd = HTMLOkKeyRdd.map((url, okKeys) :
(okKeys, url))

keysUrlRdd = keyUrlRdd.groupByKey()
```

ETL for Web Crawl & Search

- Bringing it all together: **Doing a Search**
  - **Lookup** of keyword in inverted index, find common URLs for keywords

```
for (item in searchPhrase.split())
      urls[item] = keysUrlRdd.lookup(item)
matchUrls = urls.intersect()
```

  - **Lookup** PageRank of all matching URLs
  - **Sort and Select top *n*** PageRank URLs

```
bestMatches = ranks.filter(url in matchUrls)
      .map((url, rank) : (rank, url)
      .sortByKey.takeOrdered(10)
```

Keywords →

Keyword	URL List		
People	u1	u5	u6
India	u1	u6	
Best	u2	u5	u7
Call	u3	u4	u5
Ishmael	u3		
Some	u3		
Years	u3	u7	
Here	u4		
Number	u4	u6	
Life	u7		

Filter, Intersection →

URL	PageRank
u1	0.02
u2	0.3
u3	0.08
u4	0.1
u5	0.2
u6	0.25
u7	0.05

Join, Sort, Select *n* →

URL	Title
u1	The Constitution of India
u2	A Tale of Two Cities by Dickens
u3	Project Gutenberg - Moby Dick
u4	Carly Rae Jepsen - Call Me Maybe
u5	Shah Rukh Khan interview
u6	Wikipedia – India's Population
u7	Best Years of My Life Pistol Annies

Join, Return *n* →

ETL for Web Crawl & Search

- Bringing it all together: **Doing a Search**
  - **Join** top n pages with URL and title

    ```
    titleRdd.filter(url in bestMatches)
    ```

  - **Return** result to user
  - **Suggest** similar searches (co-occurrence)
    - *How?*

Keywords

Keyword	URL List		
People	u1	u5	u6
India	u1	u6	
Best	u2	u5	u7
Call	u3	u4	u5
Ishmael	u3		
Some	u3		
Years	u3	u7	
Here	u4		
Number	u4	u6	
Life	u7		

Filter, Intersection

URL	PageRank
u1	0.02
u2	0.3
u3	0.08
u4	0.1
u5	0.2
u6	0.25
u7	0.05

Join, Sort, Select *n*

URL	Title
u1	The Constitution of India
u2	A Tale of Two Cities by Dickens
u3	Project Gutenberg - Moby Dick
u4	Carly Rae Jepsen - Call Me Maybe
u5	Shah Rukh Khan interview
u6	Wikipedia – India's Population
u7	Best Years of My Life Pistol Annies

Join, Return *n*