- Yogesh Simmhan
- simmhan@iisc.ac.in

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

DS256 (3:1)

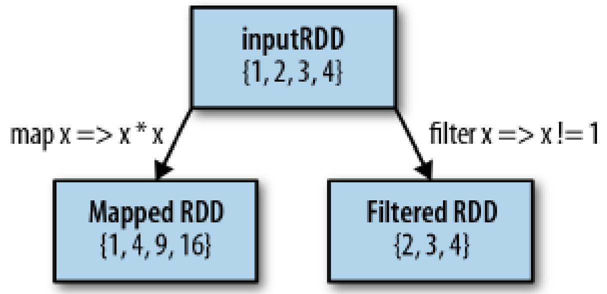
Scalable Systems for Data Science

Module 2

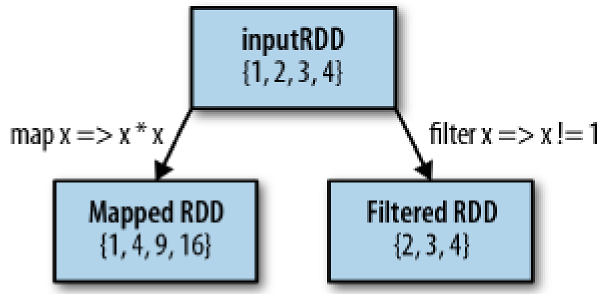
Processing Large Volumes of Big Data

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



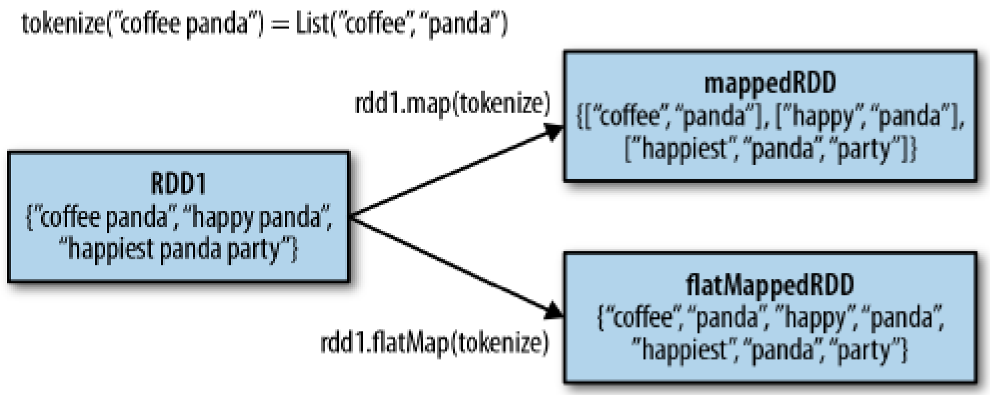
- ▷ Element-wise transformations
- ▷ **Map**
  - o Applies user logic to each element
  - o Logic returns exactly one output for each input item
  - o RDD output type can be different from input
- ▷ Can perform any user operation
  - o E.g., Parsing a string, fetching a webpage

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)
```

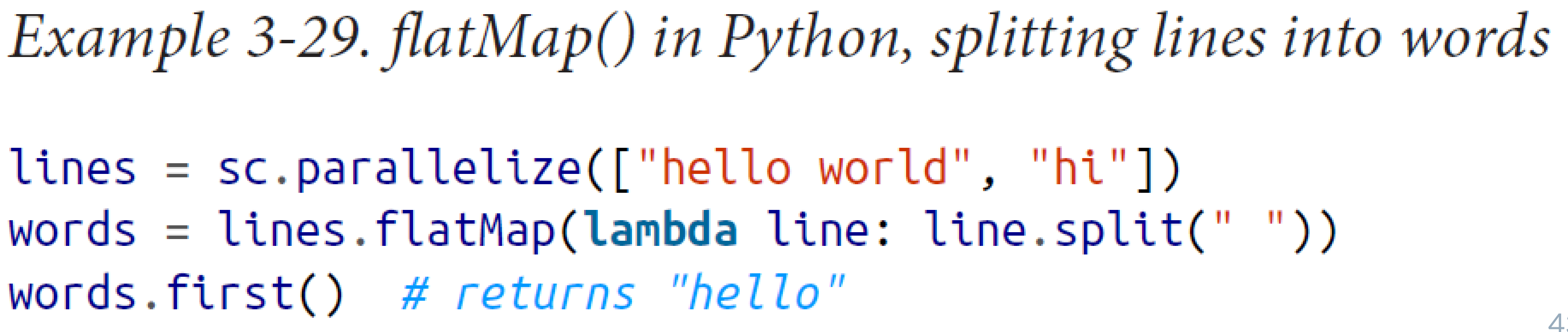
Common Transformations

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


tokenize("coffee panda") = List("coffee", "panda")
rdd1.map(tokenize)
mappedRDD
{["coffee", "panda"], ["happy", "panda"], ["happiest", "panda", "party"]}
RDD1
{"coffee panda", "happy panda", "happiest panda party"}
rdd1.flatMap(tokenize)
flatMappedRDD
{"coffee", "panda", "happy", "panda", "happiest", "panda", "party"}

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})

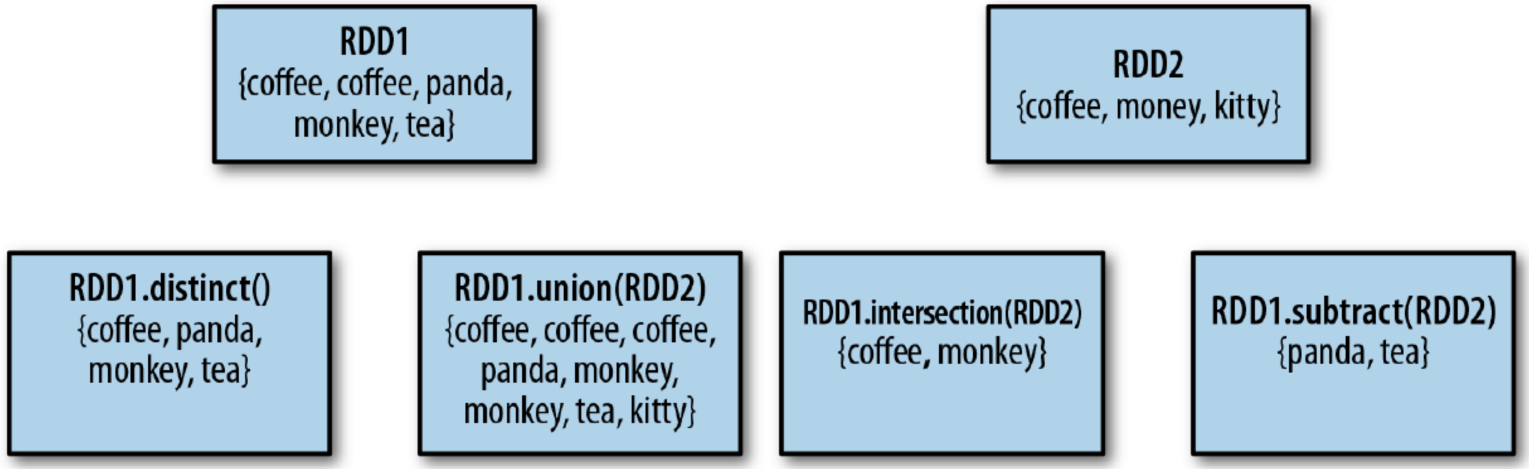
RDD2= RDD1.**flatMap**(item : if(*foo*(item)) then return item)

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

[null, item, item, null…]

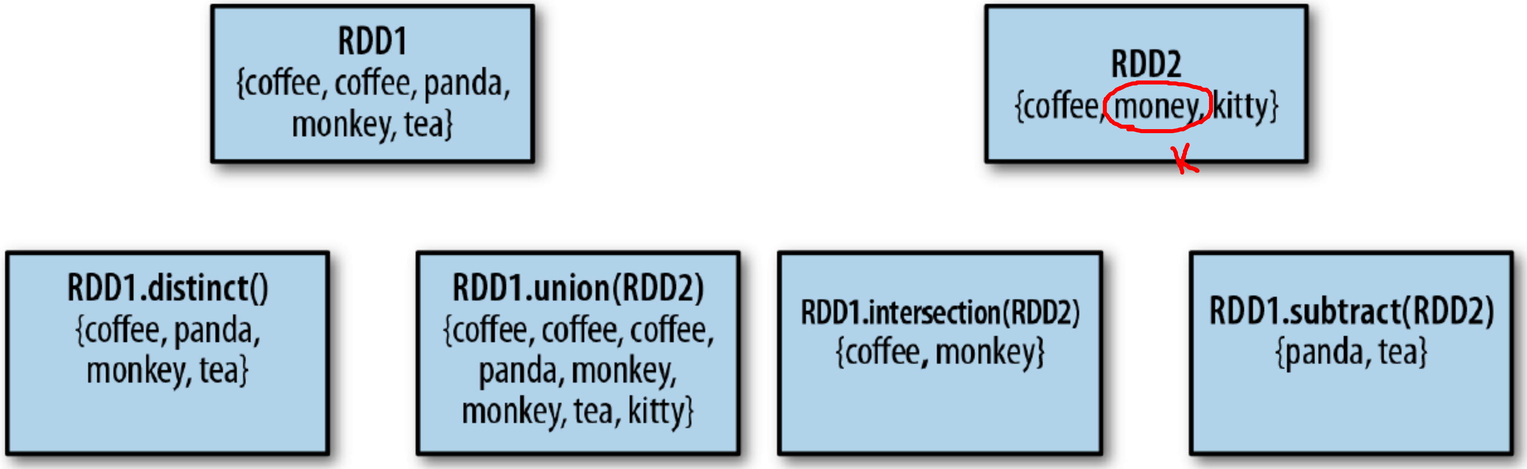
Common Transformations

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

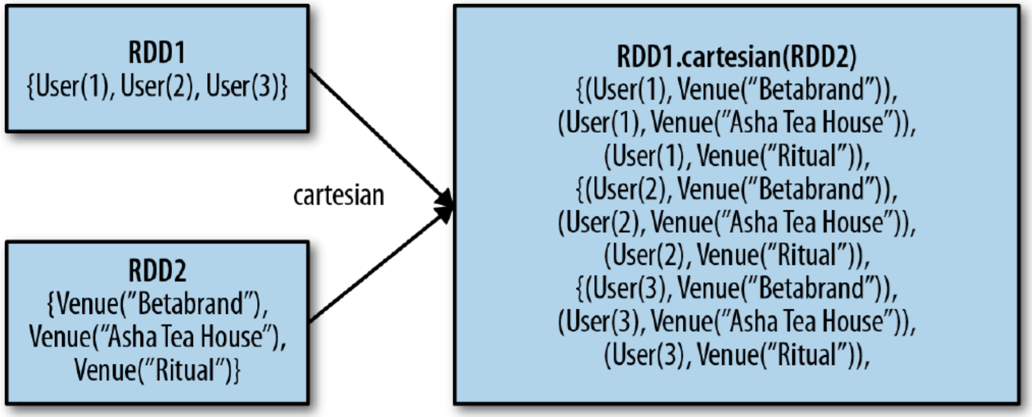


Common Transformations

- 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


RDD1 {coffee, coffee, panda, monkey, tea}
RDD2 {coffee, money, kitty}
RDD1.distinct() {coffee, panda, monkey, tea}
RDD1.union(RDD2) {coffee, coffee, coffee, panda, monkey, monkey, tea, kitty}
RDD1.intersection(RDD2) {coffee, monkey}
RDD1.subtract(RDD2) {panda, tea}

Common Transformations


RDD1
{User(1), User(2), User(3)}
RDD2
{Venue("Betabrand"), Venue("Asha Tea House"), Venue("Ritual")}
cartesian
RDD1.cartesian(RDD2)
{(User(1), Venue("Betabrand")),
(User(1), Venue("Asha Tea House")),
(User(1), Venue("Ritual")),
{(User(2), Venue("Betabrand")),
(User(2), Venue("Asha Tea House")),
(User(2), Venue("Ritual")),
{(User(3), Venue("Betabrand")),
(User(3), Venue("Asha Tea House")),
(User(3), Venue("Ritual")),

▷ Pseudo set operations

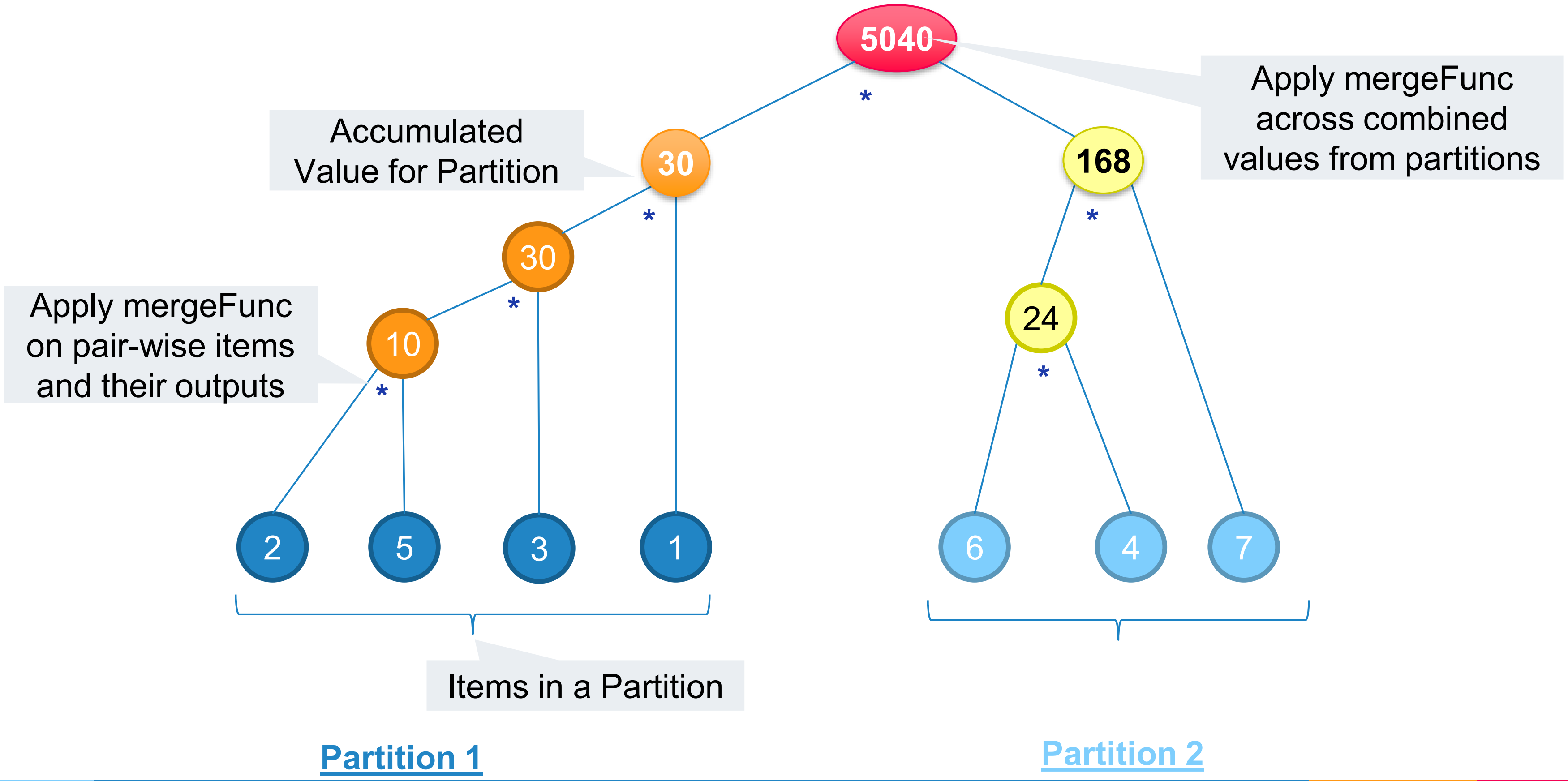
▷ **Cartesian Product**
- All-to-all combination of inputs from 2 RDDs in the output RDD

▷ **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

```
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


5040
*
Apply mergeFunc across combined values from partitions
Accumulated Value for Partition
30
*
168
*
30
*
24
*
Apply mergeFunc on pair-wise items and their outputs
10
*
2
5
3
1
6
4
7
Items in a Partition
Partition 1
Partition 2

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

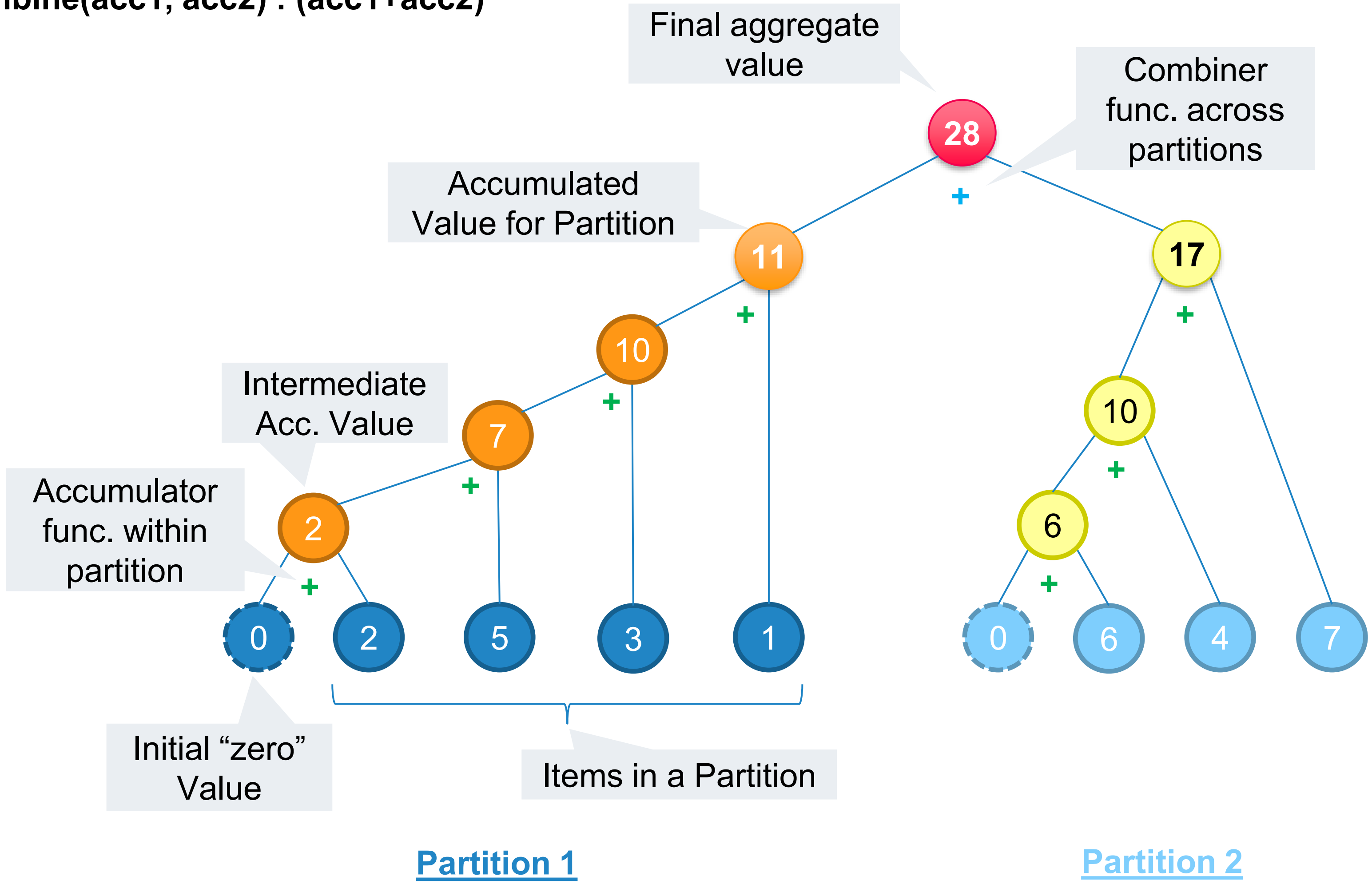
```
sum = nums.aggregate(0,
                     lambda x, y :x + y,
                     lambda x, y :x + y)
```

```
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])
```

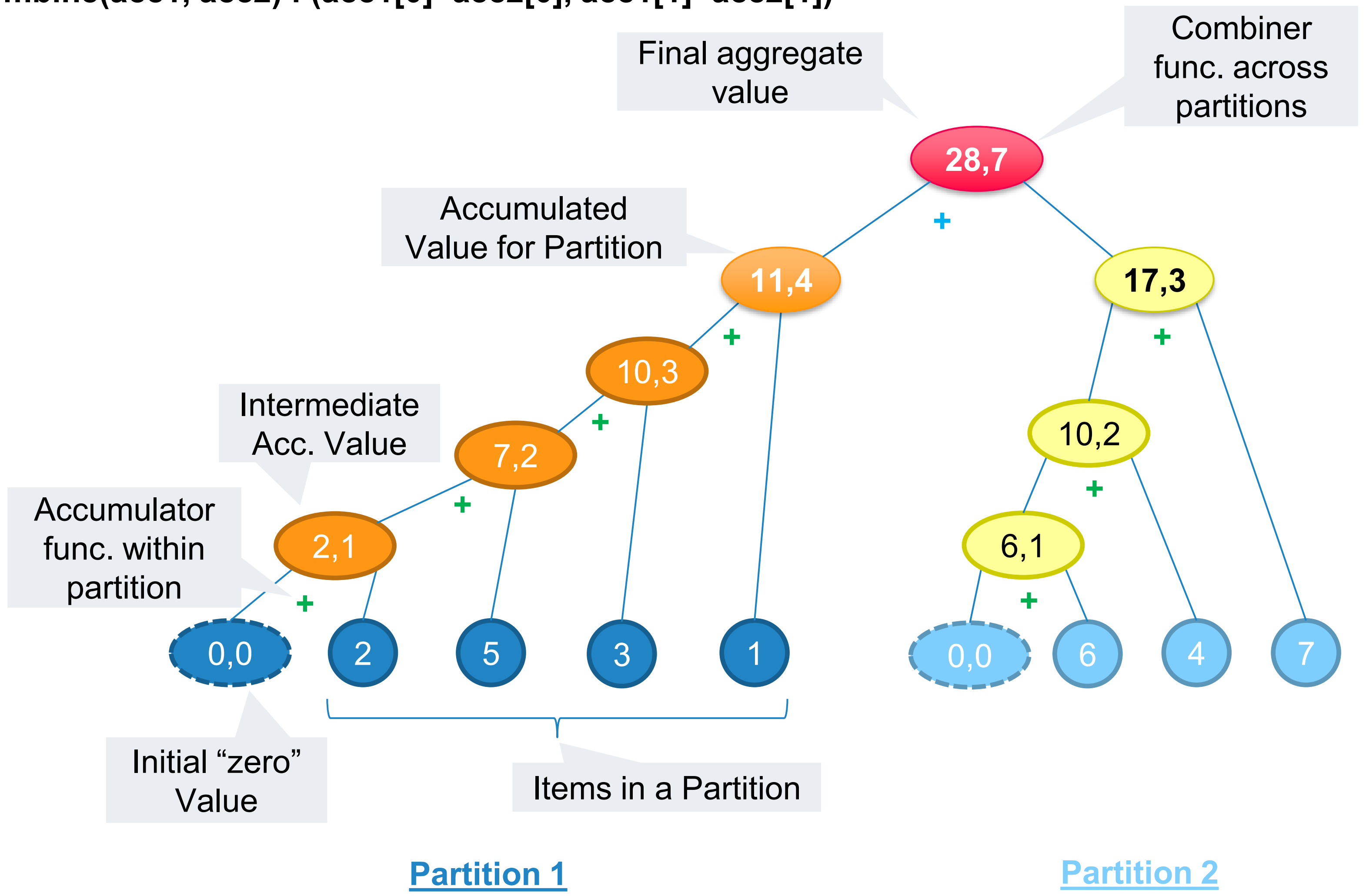
Aggregate: Incremental Evaluation within and across Partitions

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

- **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
- **takeSample**(withReplace, num, seed)
  - Return *n* items, sampled evenly from all partitions
  - Assumes each partition has uniform distribution
- **top**(n)
  - For RDD, returns the 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

Common Actions

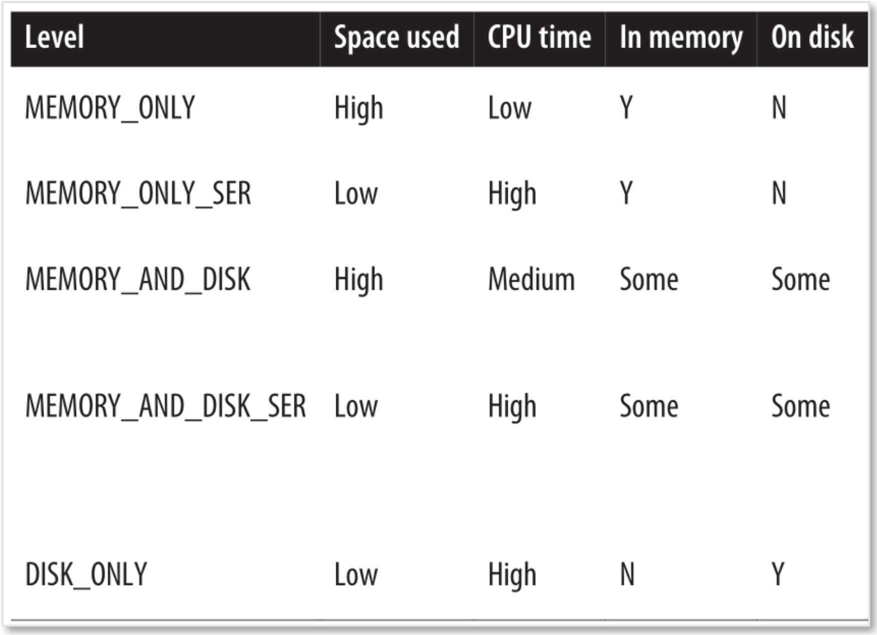
- **forEach**(fn)
  - Iterates through each item and applies function *fn*
  - Function needs to persist it. Not returned to driver.
- **countByValue**
  - Returns frequency of unique values, {val, count}

RDD Persistence

Double Exec.

```
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

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

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

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

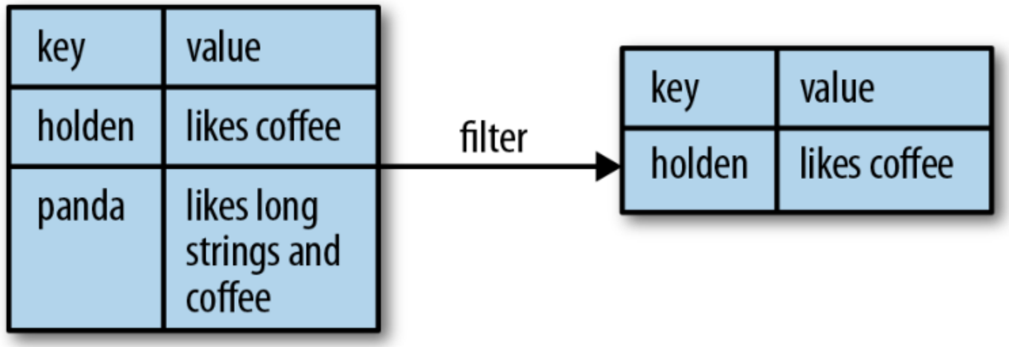
Java

```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

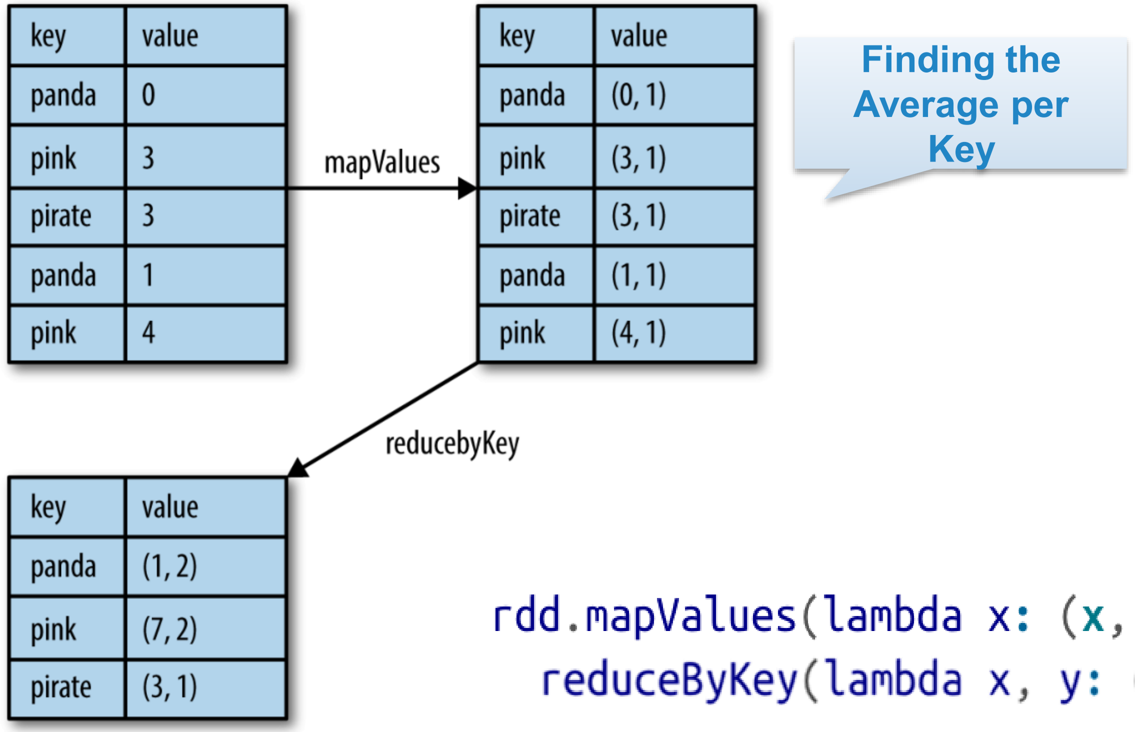
```
result = pairs.filter(lambda keyValue: len(keyValue[1]) < 20)
```



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

Transforms on a Pair RDD

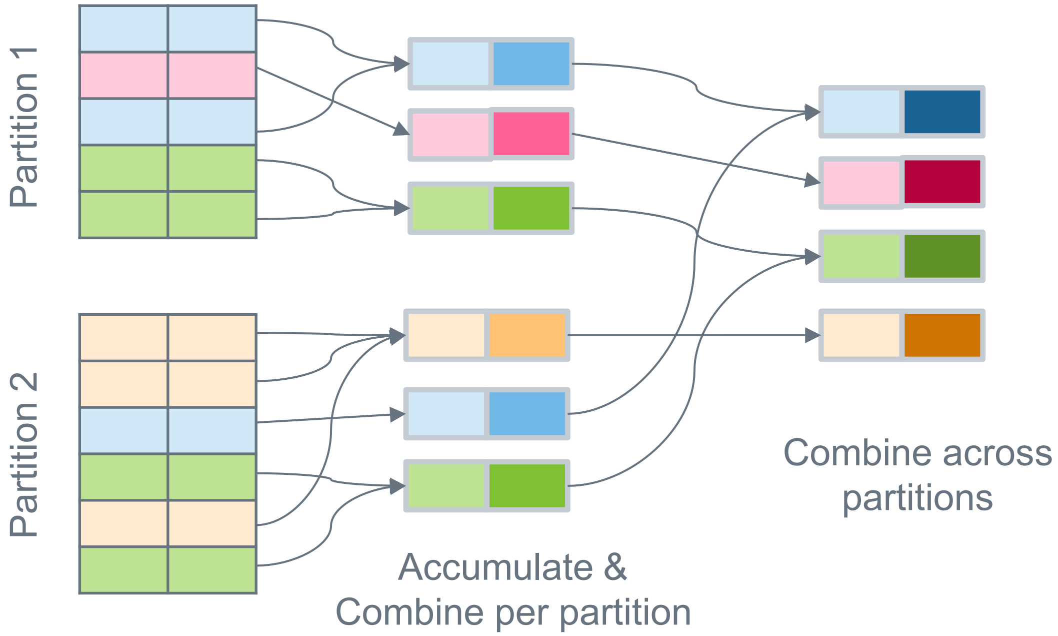
- **mapValues**(*valueFunc*)
  - Operates on value of a key to return a new output value, *retains* input key in the output
- **reduceByKey**(*mergeFunc*)
  - Combines the values, after grouping by key
  - Automatically triggers map-side *combiner*



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

Aggregation Transforms on a Pair RDD

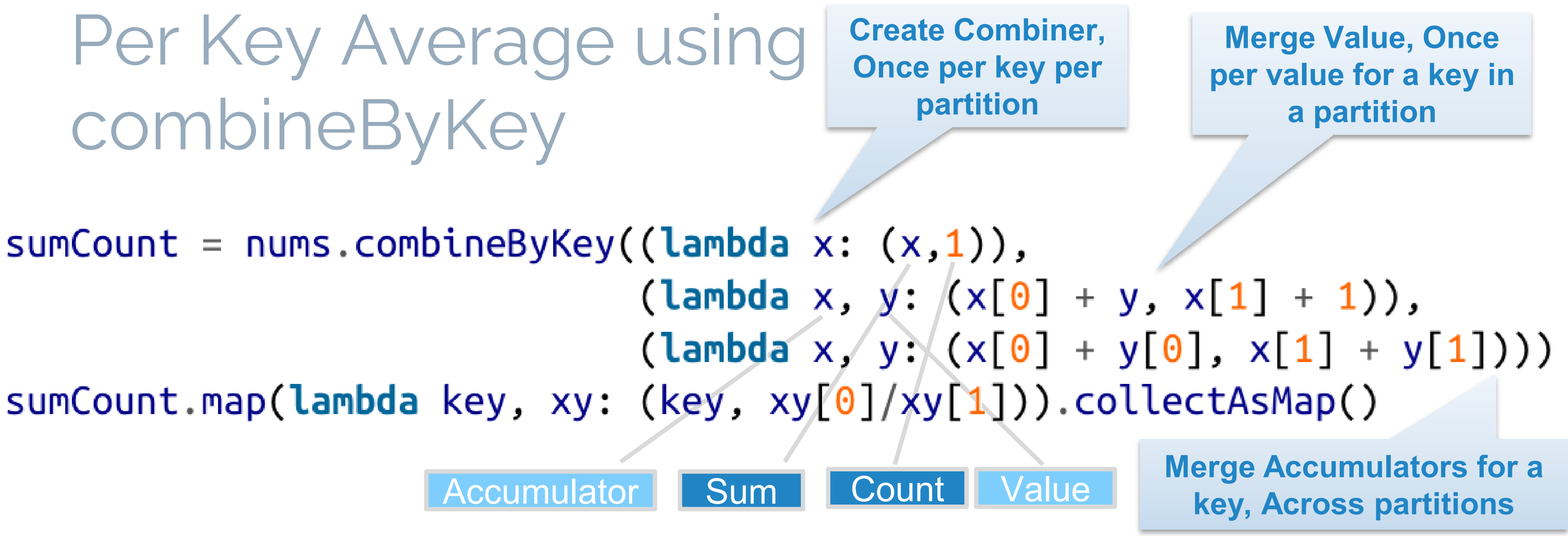
- **combineByKey** (*createCombiner*, *mergeValueFunc*, *mergeCombinersFunc*, *partitioner*)
  - Iterates through each (K,V) pair, on each partition
  - *Accumulates* values per key per partition
  - *Combines* accumulated values per key, across partitions



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 similar to combineByKey with default functions and no accumulator. *createCombiner* initialized to first initial *value* for a key on a partition. 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
Merge Accumulators for a key, Across partitions
Accumulator
Sum
Count
Value

```
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()
```

- 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
- **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
  - {(1, 2), (3, 4), (3, 6)} # {(3, 9),(4,7)} = {(1,([2],[])), (3, ([4, 6],[9])), (4,([],[7]))}
- **subtractByKey**(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)}

Join Transforms on two Pair RDDs

- **Join**
  - Performs inner join
  - Only keys in <u>both RDDs</u> are joined and returned
  - Cross product of values for same key from both RDDs
    - {(1, 2), (3, 4), (3, 6)} ⋈ {(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)} ⋈ {(3, 9)} = {(1,(2,*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)} ⋈ {(3, 9), (4,2)} = {(3,(4,9)), (3,(6,9)), (4,(*None*,2))}

Join Transforms on two Pair RDDs

▷ Inner Join

```
storeAddress = {
  (Store("Ritual"), "1026 Valencia St"), (Store("Philz"), "748 Van Ness Ave"),
  (Store("Philz"), "3101 24th St"), (Store("Starbucks"), "Seattle")}

storeRating = {
  (Store("Ritual"), 4.9), (Store("Philz"), 4.8))}

storeAddress.join(storeRating) == {
  (Store("Ritual"), ("1026 Valencia St", 4.9)),
  (Store("Philz"), ("748 Van Ness Ave", 4.8)),
  (Store("Philz"), ("3101 24th St", 4.8))}
```

Join Transforms on two Pair RDDs

▷ Left Outer Join

```
storeAddress = {
  (Store("Ritual"), "1026 Valencia St"), (Store("Philz"), "748 Van Ness Ave"),
  (Store("Philz"), "3101 24th St"), (Store("Starbucks"), "Seattle")}

storeRating = {
  (Store("Ritual"), 4.9), (Store("Philz"), 4.8))}


storeAddress.leftOuterJoin(storeRating) ==
{(Store("Ritual"),("1026 Valencia St",Some(4.9))),
  (Store("Starbucks"),("Seattle",None)),
  (Store("Philz"),("748 Van Ness Ave",Some(4.8))),
  (Store("Philz"),("3101 24th St",Some(4.8)))}
```

Join Transforms on two Pair RDDs

- ▷ Right Outer Join

```
storeAddress = {
  (Store("Ritual"), "1026 Valencia St"), (Store("Philz"), "748 Van Ness Ave"),
  (Store("Philz"), "3101 24th St"), (Store("Starbucks"), "Seattle")}

storeRating = {
  (Store("Ritual"), 4.9), (Store("Philz"), 4.8))}


storeAddress.rightOuterJoin(storeRating) ==
{(Store("Ritual"),(Some("1026 Valencia St"),4.9)),
  (Store("Philz"),(Some("748 Van Ness Ave"),4.8)),
  (Store("Philz"), (Some("3101 24th St"),4.8))}
```

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/

Stratified Sampling

- **sampleByKey**(*withReplace*, *keyFractions*, *seed*)
  - *keyFractions* is a map of $\langle k, f_k \rangle$
  - Sample <u>approximately</u> $\lceil f_k \times n_k \rceil$ 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

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

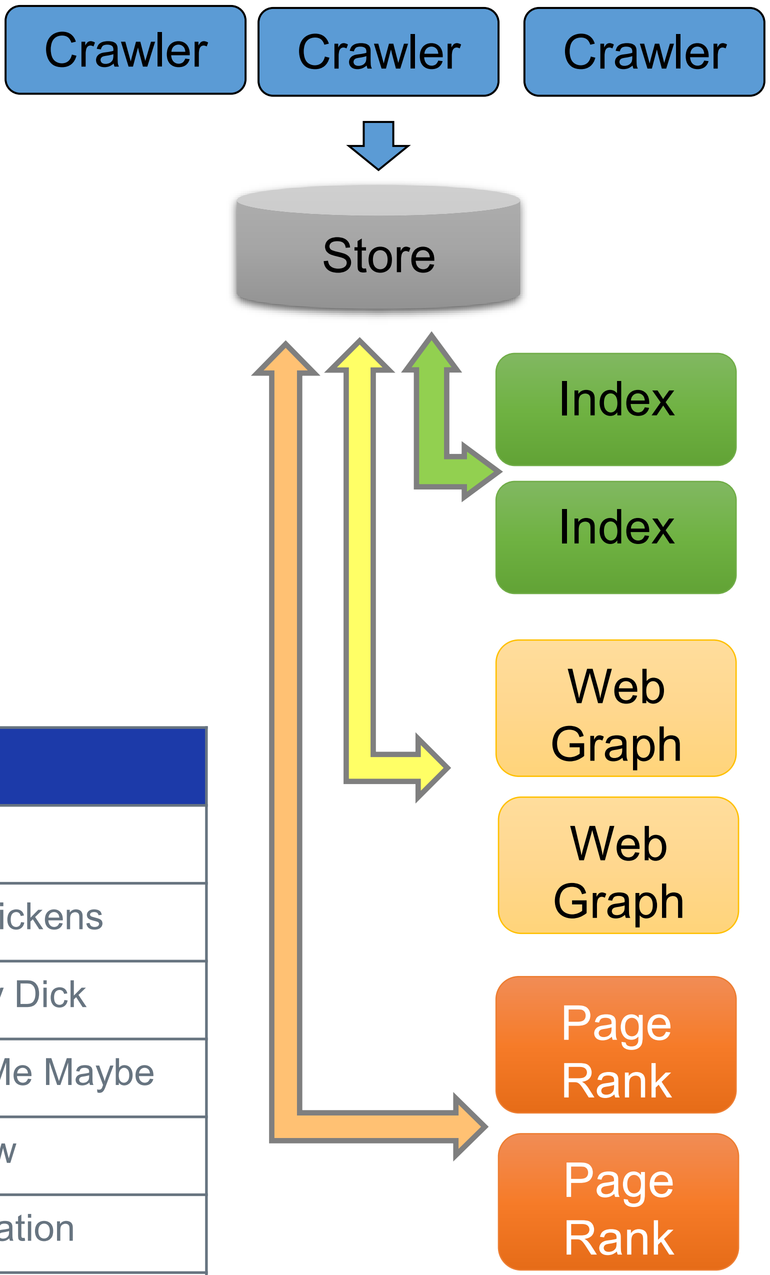
Case Study: Web Crawl, Index & Search

Web Crawl: Overview

▷ Extract Title for URL

```
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; pa
co.uk;

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

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

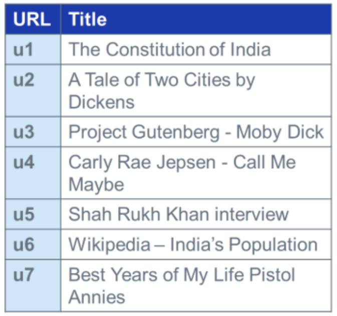


https://commoncrawl.org/the-data/get-started/

Spark: Extract Title for URL

- 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

Build Inverted Index

- Extract URL → Words
- Identify keywords
  - Remove Stopwords, normalize contractions, etc.
- Invert to Keyword→URL[]

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

Remove stop words, contractions

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		

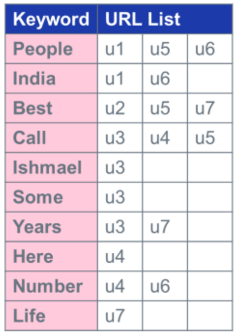
Spark: Inverted Index

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

```
HTMLWordRdd = HTMLRdd.flatMap((url, html) :
(url, parseOutWords(html)))
```

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

```
HTMLKeywordRdd = HTMLWordRdd.filter((url,
word) : word 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		

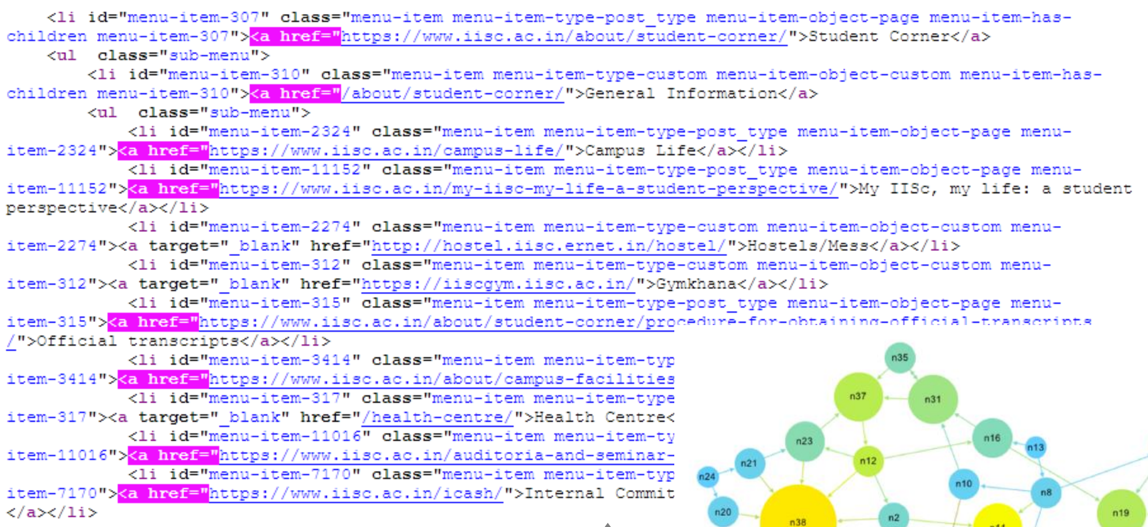
Spark: Inverted Index

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

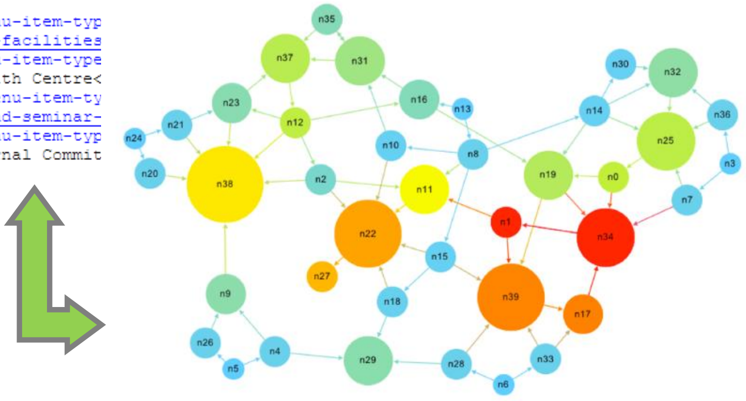
```
keyUrlRdd = HTMLKeywordRdd.map((url, keyword)
: (keyword, url))

keyUrlsRdd = keyUrlRdd.groupByKey()
```

Web Graph and PageRank

- ▷ Build WWW Link Graph
  - ○ Extract links, build graph adjacency list
- ▷ Calculate PageRank



```
    <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-typ
item-3414"><a href="https://www.iisc.ac.in/about/campus-facilities
            <li id="menu-item-317" class="menu-item menu-item-type
item-317"><a target="_blank" href="/health-centre/">Health Centre<
            <li id="menu-item-11016" class="menu-item menu-item-ty
item-11016"><a href="https://www.iisc.ac.in/auditoria-and-seminar-
            <li id="menu-item-7170" class="menu-item menu-item-typ
item-7170"><a href="https://www.iisc.ac.in/icash/">Internal Commit
</a></li>
```



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

Spark: Build Web Graph

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

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

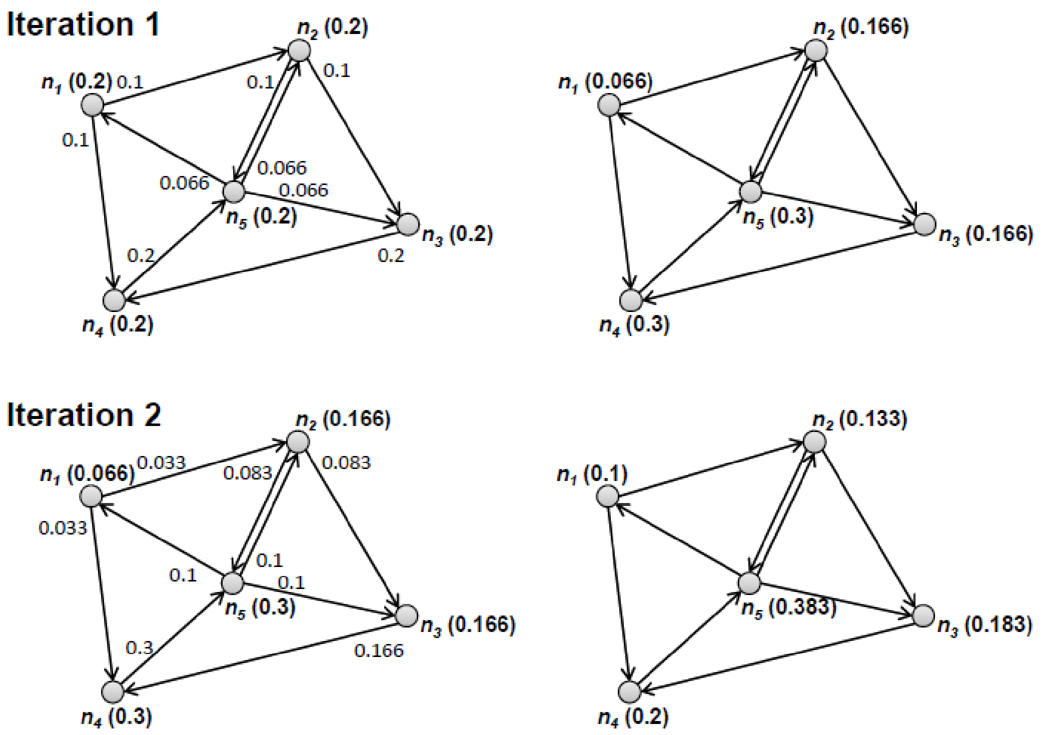
- Run PageRank on the Adjacency List
  - <url>, PageRank
  - *How?*

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

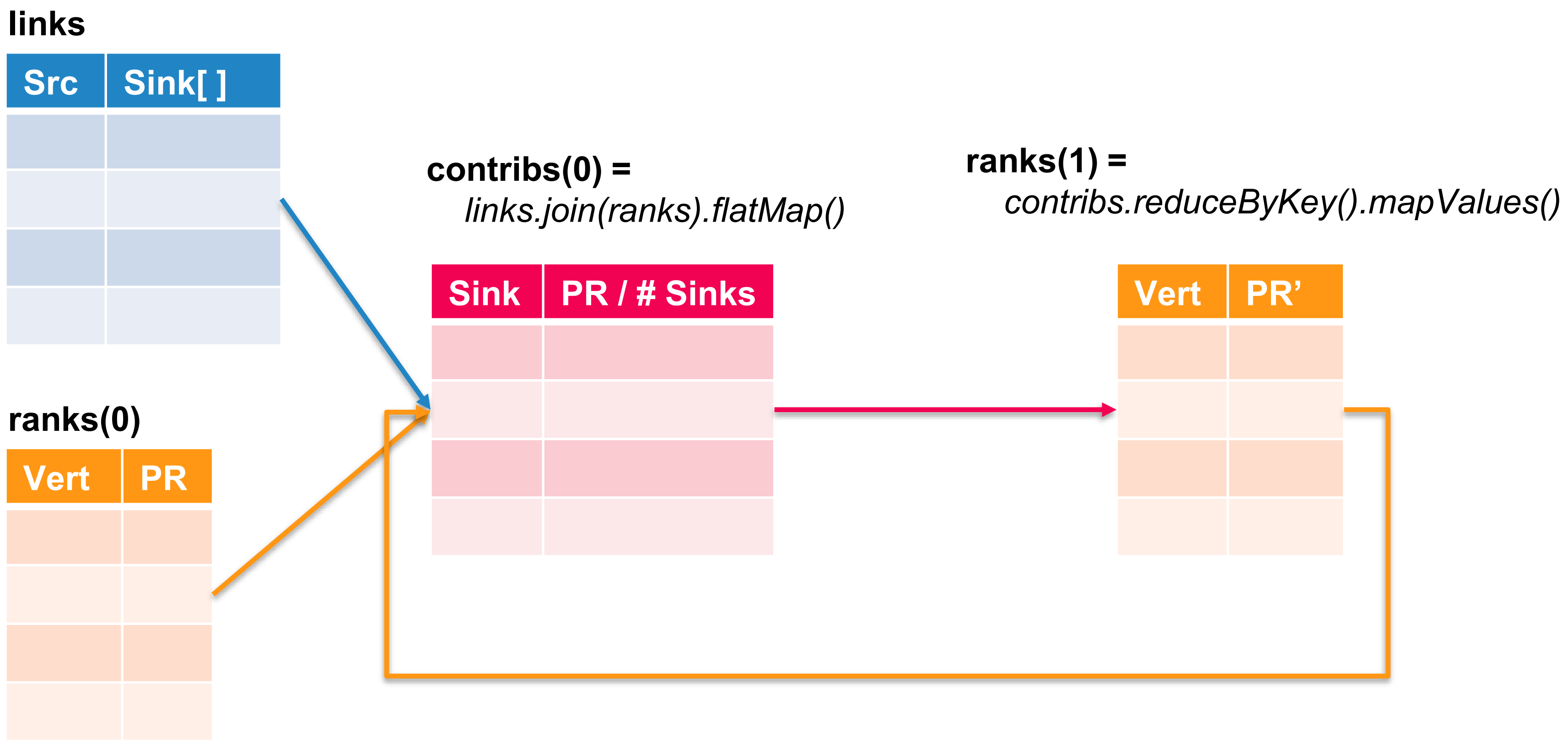
PageRank

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

- 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)**)



Spark: PageRank



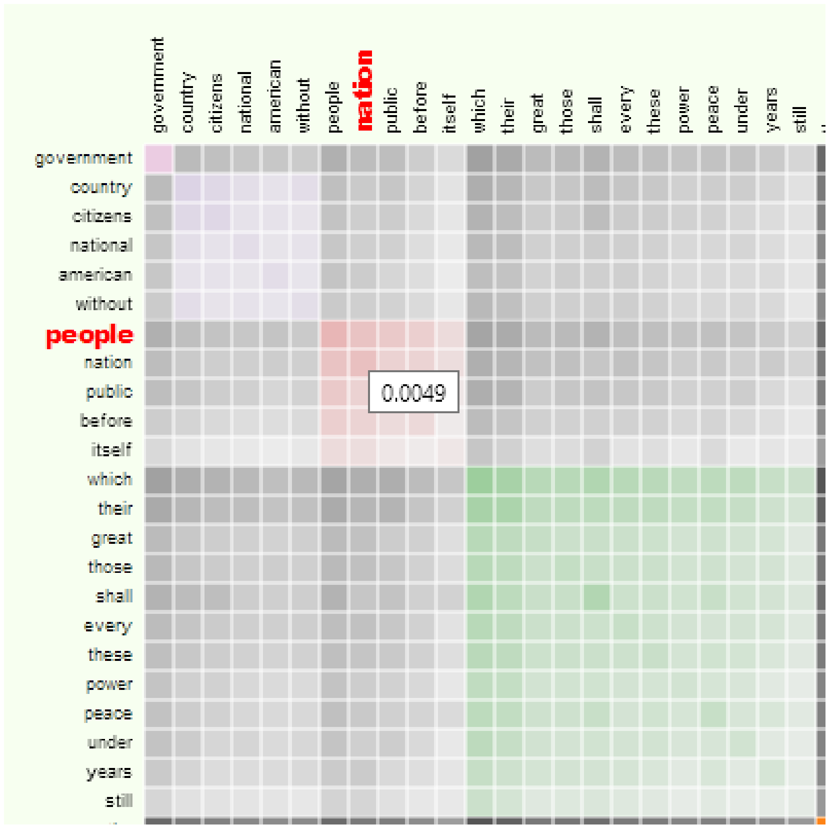
Spark: PageRank

```
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

Word Co-occurrence

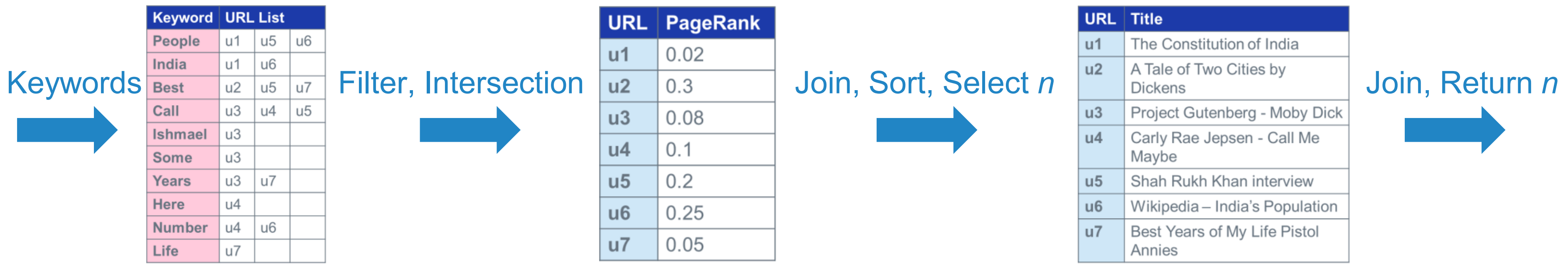
- Word co-occurrence and associative rule mining
  - Search suggestions



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

Web 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)



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		

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

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

Spark: Doing a Search

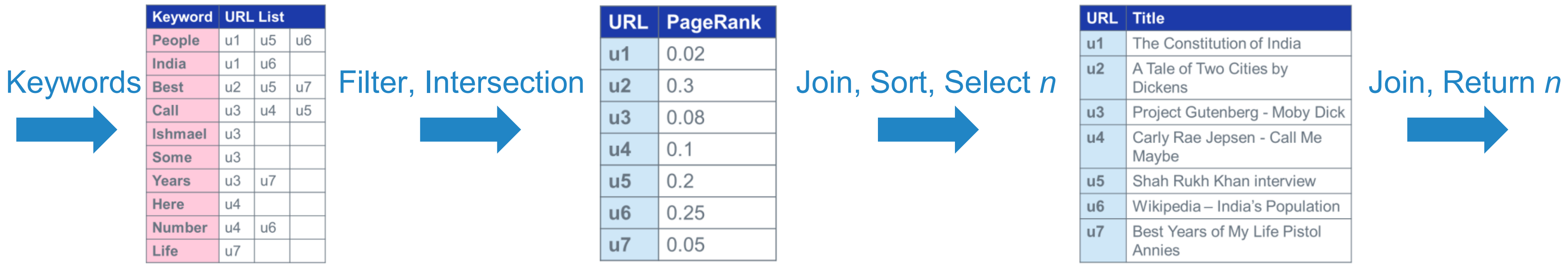
How do I do these fully using Spark RDDs and not the driver memory?

- Execution coordinated using driver memory
- **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)
```



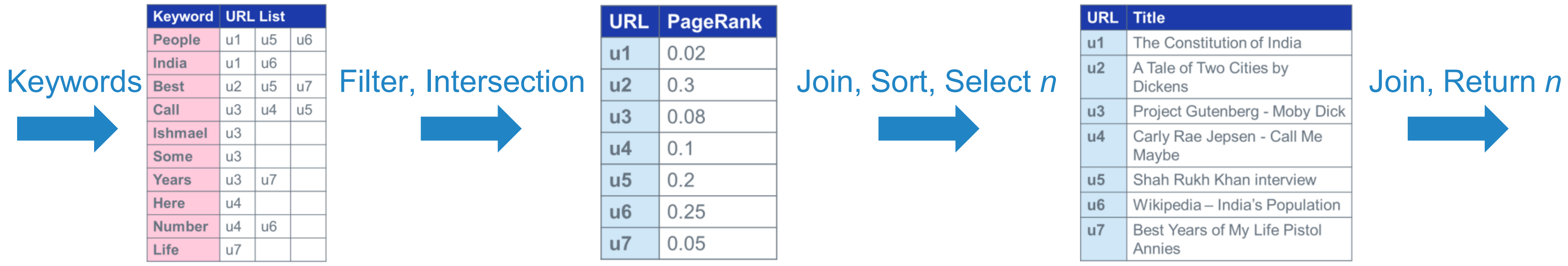
Keyword	URL List		
People	u1	u5	u6
India	u1	u6	
Best	u2	u5	u7
Call	u3	u4	u5
Ishmael	u3		
Some	u3		
Years	u3	u7	
Here	u4		
Number	u4	u6	
Life	u7		

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

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

Spark: Doing a 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?*



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		

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

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