MapReduce Algorithms for Big Data Analysis

Kyuseok Shim

Seoul National University, Korea
http://ee.snu.ac.kr/~shim

About this Tutorial

- Tutorial is presented based solely on publicly available information
- Information is incomplete and could be inaccurate
- Presentation reflects my understanding which may be erroneous

Outline

- Introduction to MapReduce
  - MapReduce framework
  - MapReduce programming practices
  - Advanced MapReduce programming skills
- Joins
  - Theta-joins
  - Similarity joins
  - Join order optimizations
- Data mining
  - Clustering
  - Probabilistic modeling
  - Association rule mining
  - Classification
  - Graph analysis
- Potpourri
- Summary

MapReduce Framework

- For data-intensive applications with big data, it has recently received a lot of attention
- A simple programming model that allows easy development of scalable parallel applications to process big data on large clusters of commodity machines
- Google's MapReduce or its open-source equivalent Hadoop is a powerful implementation of MapReduce Framework
- User writes map, reduce and main functions

Hadoop

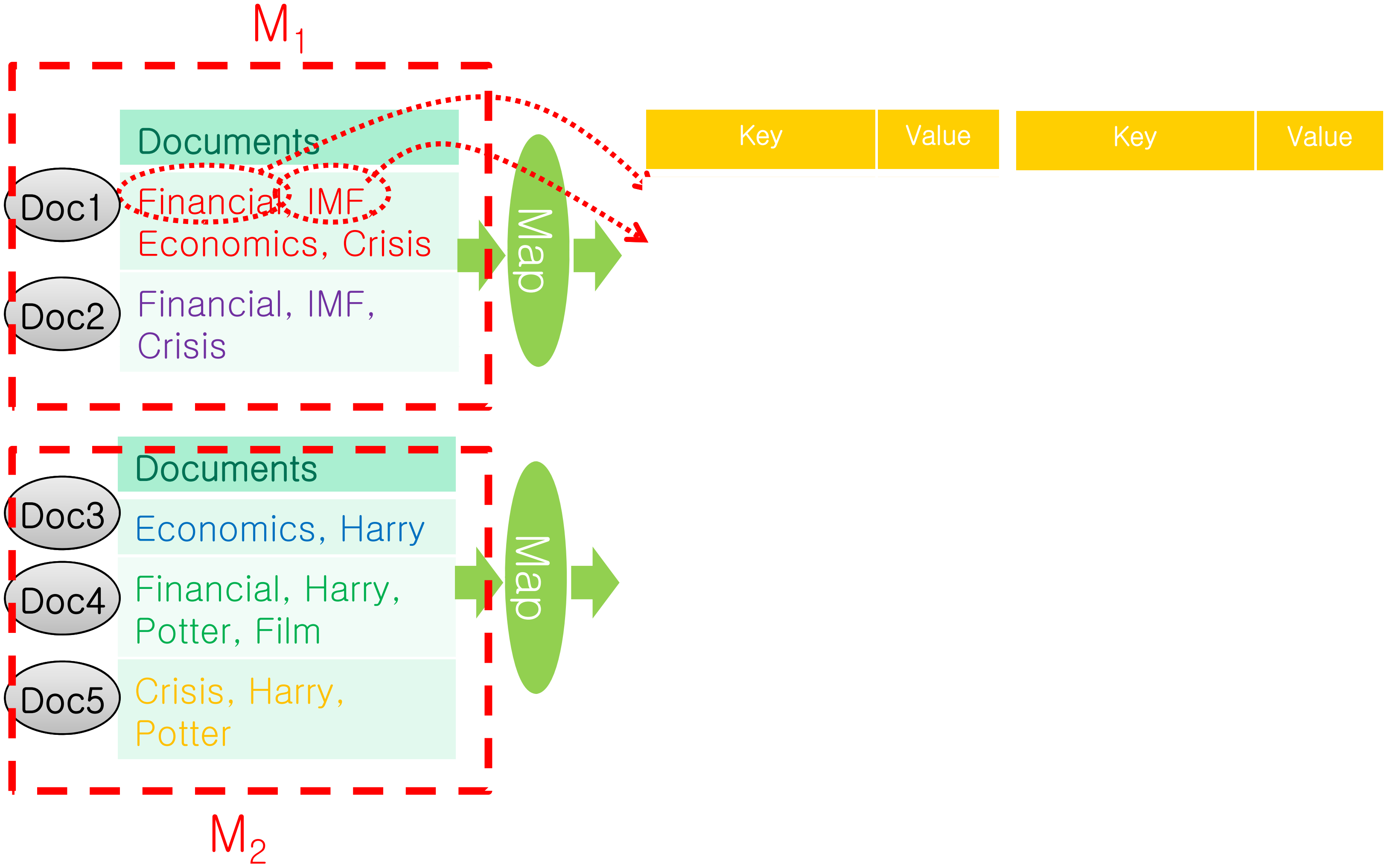
- Open source of MapReduce framework of Apache Project
- Hadoop Distributed File System (HDFS)
  - Store big files across machines
  - Store each file as a sequence of blocks
  - Each block of a file are replicated for fault tolerance
- Distribute processing of large data across up to thousands of commodity machines
- Key components
  - MapReduce - distributes applications
  - Hadoop Distributed File System (HDFS) - distributes data
- A single Namenode (master) and multiple Datanodes (slaves)
  - Namenode: manages the file system and access to files by clients
  - Datanode: manages the storages attached to the nodes running on

MapReduce Programming Model

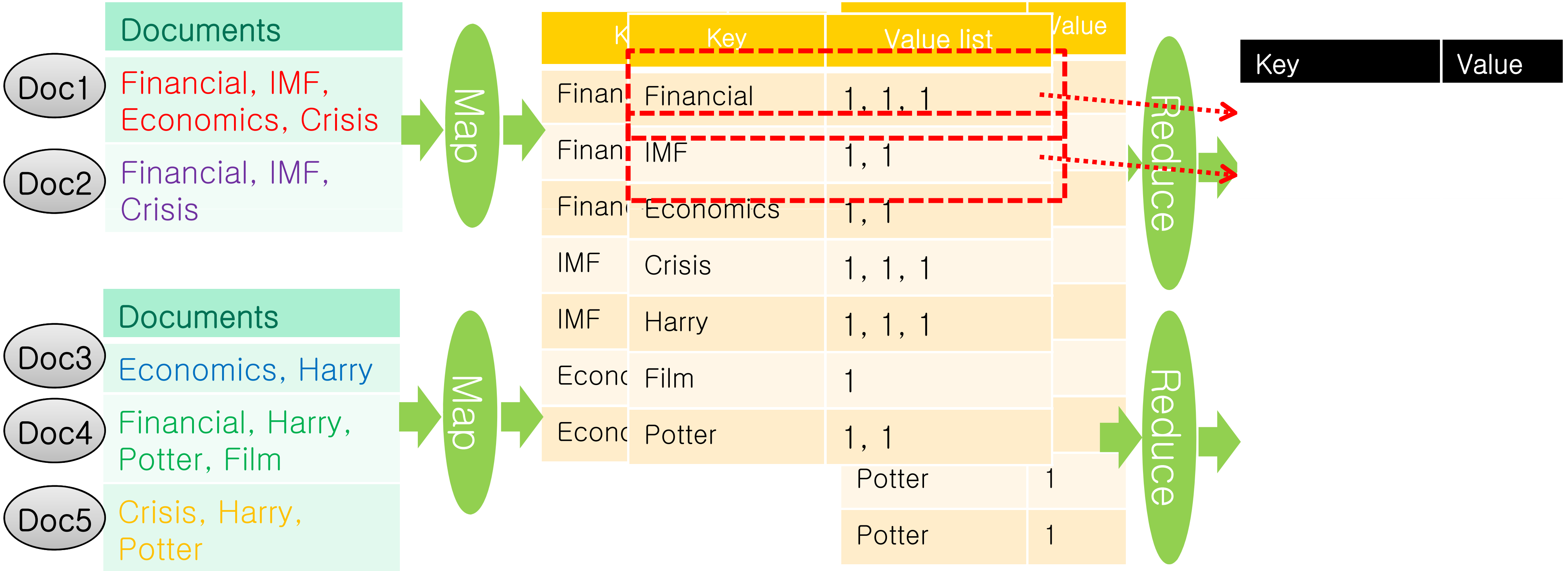
- Borrows from functional programming
- Users should implement two primary methods:
  - Map: (key1, val1) → [(key2, val2)]
  - Reduce: (key2, [val2]) → [(key3, val3)]

MapReduce Programming Practices

An Example of Word Counting with MapReduce



An Example of Word Counting with MapReduce



	Documents
Doc1	Financial, IMF, Economics, Crisis
Doc2	Financial, IMF, Crisis

	Documents
Doc3	Economics, Harry
Doc4	Financial, Harry, Potter, Film
Doc5	Crisis, Harry, Potter

Key	Value list
Financial	1, 1, 1
IMF	1, 1
Economics	1, 1
Crisis	1, 1, 1
Harry	1, 1, 1
Film	1
Potter	1, 1

Potter	1
Potter	1

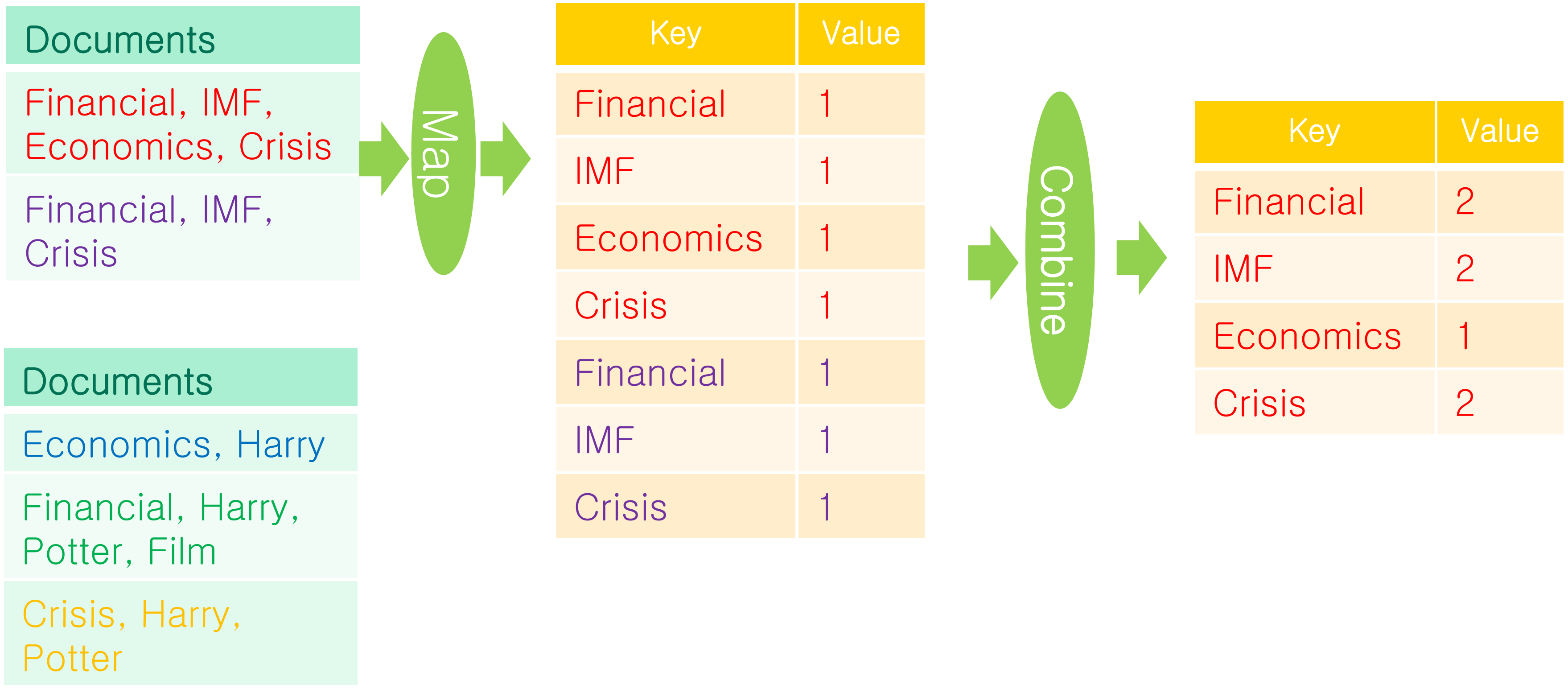
Key	Value

Before reduce functions are called,
for each distinct key, the list of its values are generated

Combine Function

- Reduce the result size of map functions
- Perform reduce-like function in each machine
- Decrease the shuffling cost
- It is desirable to design MapReduce algorithms to use combine functions

An Example of Word Counting with Combine Function



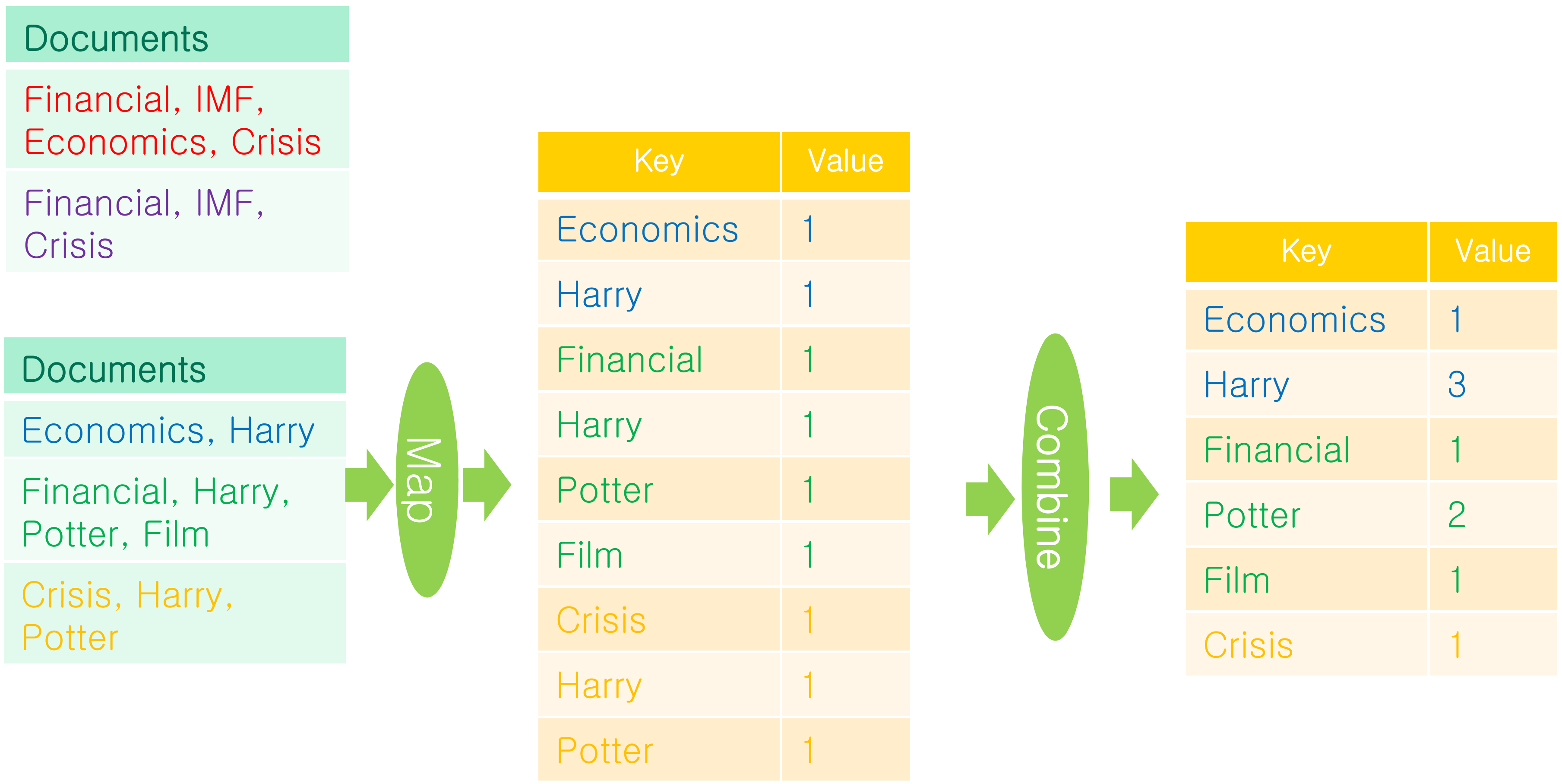
Documents
Financial, IMF, Economics, Crisis
Financial, IMF, Crisis

Documents
Economics, Harry
Financial, Harry, Potter, Film
Crisis, Harry, Potter

Key	Value
Financial	1
IMF	1
Economics	1
Crisis	1
Financial	1
IMF	1
Crisis	1

Key	Value
Financial	2
IMF	2
Economics	1
Crisis	2

An Example of Word Counting with Combine Function



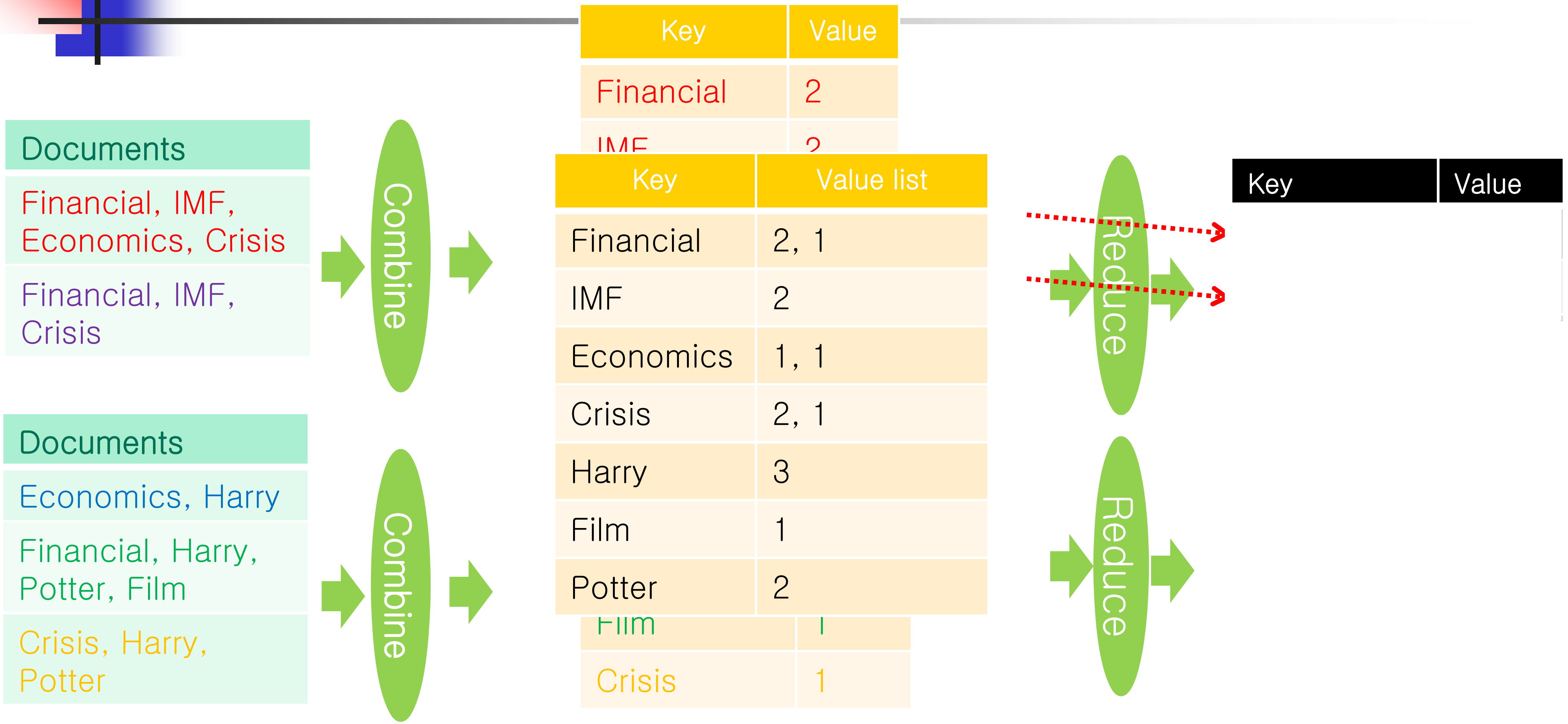
Documents
Financial, IMF, Economics, Crisis
Financial, IMF, Crisis

Documents
Economics, Harry
Financial, Harry, Potter, Film
Crisis, Harry, Potter

Key	Value
Economics	1
Harry	1
Financial	1
Harry	1
Potter	1
Film	1
Crisis	1
Harry	1
Potter	1

Key	Value
Economics	1
Harry	3
Financial	1
Potter	2
Film	1
Crisis	1

An Example of Word Counting with Combine Function



Documents
Financial, IMF, Economics, Crisis
Financial, IMF, Crisis

Documents
Economics, Harry
Financial, Harry, Potter, Film
Crisis, Harry, Potter

Key	Value
Financial	2
IMF	2

Key	Value list
Financial	2, 1
IMF	2
Economics	1, 1
Crisis	2, 1
Harry	3
Film	1
Potter	2

Film	1
Crisis	1

Key	Value

Before reduce functions are called,
for each distinct key, the list of its values are generated

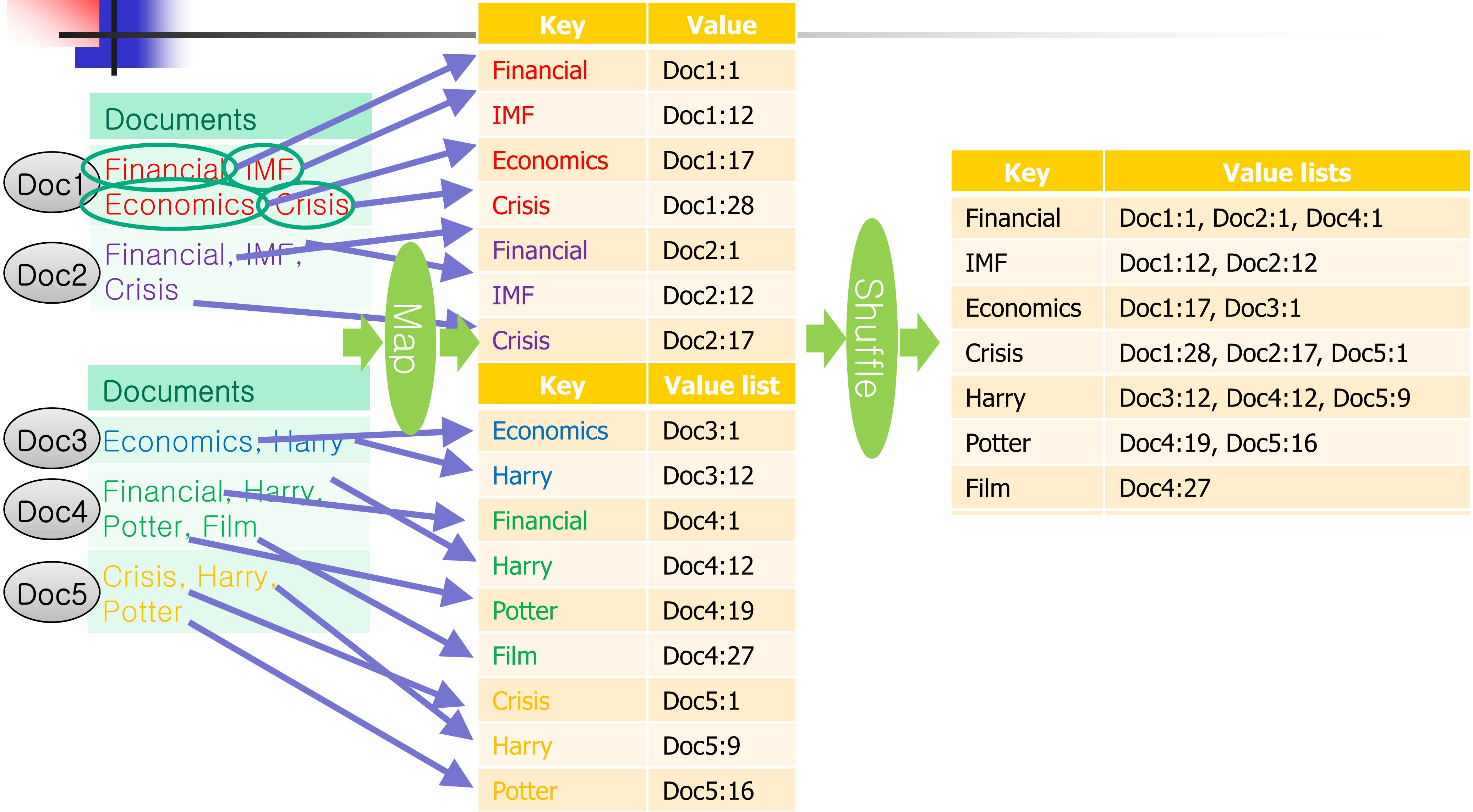
An Example of Building an Inverted Index

Doc1: IMF, Financial Economics Crisis
Doc2: IMF, Financial Crisis
Doc3: Harry Economics
Doc4: Financial Harry Potter Film
Doc5: Harry Potter Crisis

The following is the inverted index of the above data

IMF -> Doc1:1, Doc2:1
Financial -> Doc1:6, Doc2:6, Doc4:1
Economics -> Doc1:16, Doc3:7
Crisis -> Doc1:26, Doc2:16, Doc5:14
Harry -> Doc3:1, Doc4:11, Doc5:1
Potter -> Doc4:17, Doc5:7
Film -> Doc4:24

An Example of Building an Inverted Index



Documents

Doc1: Financial, IMF, Economics, Crisis

Doc2: Financial, IMF, Crisis

Documents

Doc3: Economics, Harry

Doc4: Financial, Harry, Potter, Film

Doc5: Crisis, Harry, Potter

Map

Key	Value
Financial	Doc1:1
IMF	Doc1:12
Economics	Doc1:17
Crisis	Doc1:28
Financial	Doc2:1
IMF	Doc2:12
Crisis	Doc2:17

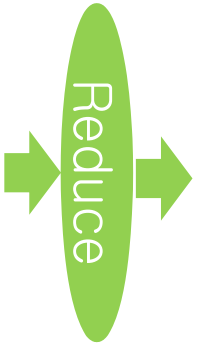
Key	Value list
Economics	Doc3:1
Harry	Doc3:12
Financial	Doc4:1
Harry	Doc4:12
Potter	Doc4:19
Film	Doc4:27
Crisis	Doc5:1
Harry	Doc5:9
Potter	Doc5:16

Shuffle

Key	Value lists
Financial	Doc1:1, Doc2:1, Doc4:1
IMF	Doc1:12, Doc2:12
Economics	Doc1:17, Doc3:1
Crisis	Doc1:28, Doc2:17, Doc5:1
Harry	Doc3:12, Doc4:12, Doc5:9
Potter	Doc4:19, Doc5:16
Film	Doc4:27

An Example of Building an Inverted Index

Key	Value lists
Financial	Doc1:1, Doc2:1, Doc4:1
IMF	Doc1:12, Doc2:12
Economics	Doc1:17, Doc3:1
Crisis	Doc1:28, Doc2:17, Doc5:1
Harry	Doc3:12, Doc4:12, Doc5:9
Potter	Doc4:19, Doc5:16
Film	Doc4:27



Key	Value lists
Financial	Doc1:1, Doc2:1, Doc4:1
IMF	Doc1:12, Doc2:12
Economics	Doc1:17, Doc3:1
Crisis	Doc1:28, Doc2:17, Doc5:1
Harry	Doc3:12, Doc4:12, Doc5:9
Potter	Doc4:19, Doc5:16
Film	Doc4:27

Overview of MapReduce

- Mapper and Reducer
  - Independent threads in each machine
  - Invoke map and reduce functions respectively
- Combine functions
  - Perform reduce function in each machine
  - Reduce shuffling cost and network traffics
- Each map or reduce task can optionally use two additional functions: init() and close()
  - init() : called at the start of each map or reduce task
  - close(): called at the end of each map or reduce task
- A MapReduce job can be configured to process map function phase only

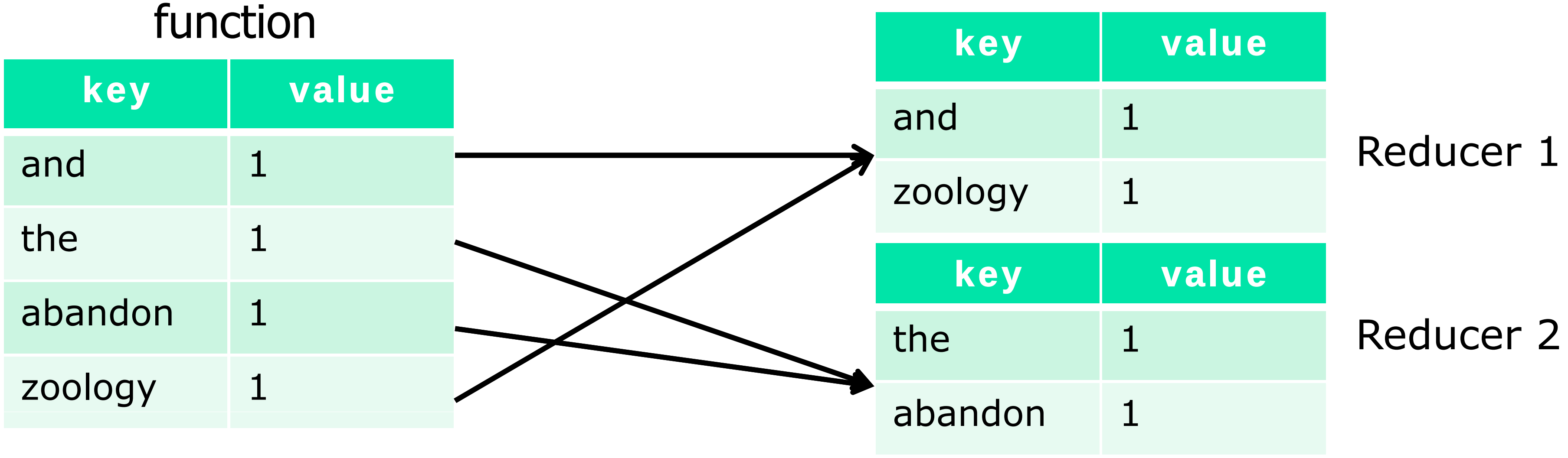
Advanced MapReduce Programming Skills

Advanced Programming Skills

- Forcing key distributions to reducers
  - Theta join, similarity join, PLDA, etc.
- Broadcasting to mappers and reducers
  - Theta join, similarity join, clustering, decision tree, matrix multiplication and factorization, EM algorithm, etc.
- Redefine partitioning scheme of shuffling
  - Theta join, similarity join, etc.
- Sharding data for multiple MapReduce Phases
  - PageRank, clustering, EM algorithm, etc.
- Grouping keys
  - Association rule, similarity join, etc.
- No reduce phase
  - Theta join

(1) Forcing Key Distributions to Reducers

- Partitioner class
  - Assign the reducer for each key-value pair emitted by map functions
  - Default Partitioner class uniformly distributes key-value pairs to every reducer
    - Key-value pairs with the same key goes to the same reduce function



key	value
and	1
the	1
abandon	1
zoology	1

key	value
and	1
zoology	1

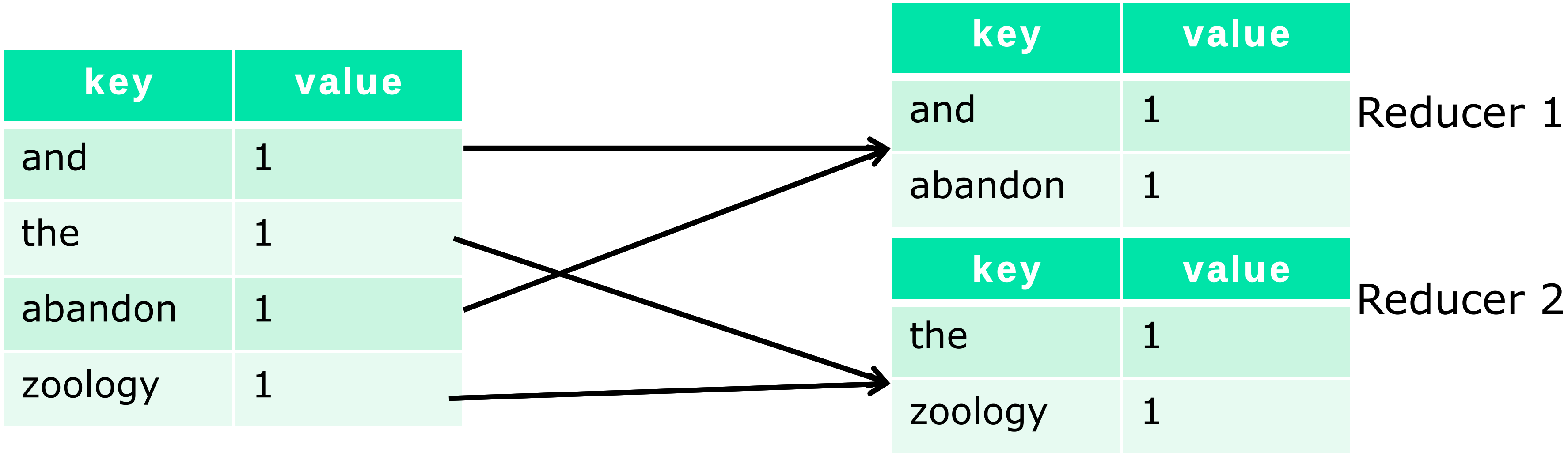
Reducer 1

key	value
the	1
abandon	1

Reducer 2

(1) Forcing Key Distributions to Reducers

- Assume that you want the key-value pairs are ordered by the alphabetical order of keys as the follwoing
  - The key-value pairs whose keys start with 'a' go to reducer 1,
  - The other key-value pairs go to reducer 2
- Modify Partitioner class!



key	value
and	1
the	1
abandon	1
zoology	1

Reducer 1

key	value
and	1
abandon	1

Reducer 2

key	value
the	1
zoology	1

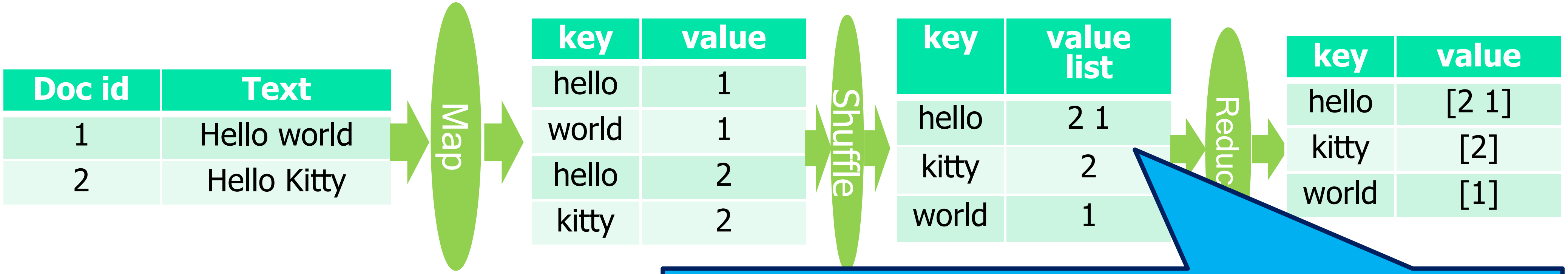
→ Redefine the class 'Partitioner'!!

(2) Broadcast to Map and Reduce

- Small data
  - Use the Configuration class provided in Hadoop
- Large data
  - Simply, write and read the data in HDFS
  - First, write a file to broadcast on HDFS in the main function before executing a MapReduce task
  - Read the broadcast file in the setup function of each map or reduce function from HDFS
  - Hadoop automatically calls “setup” function before the map and reduce functions are called

(3) Redefine Partitioning Scheme of Shuffling

- Hadoop only sorts on the keys in shuffling phase
- e.g.) Build an inverted list of each word where the documents ids are sorted in the increasing order of page ids



Doc id	Text
1	Hello world
2	Hello Kitty

key	value
hello	1
world	1
hello	2
kitty	2

key	value list
hello	2 1
kitty	2
world	1

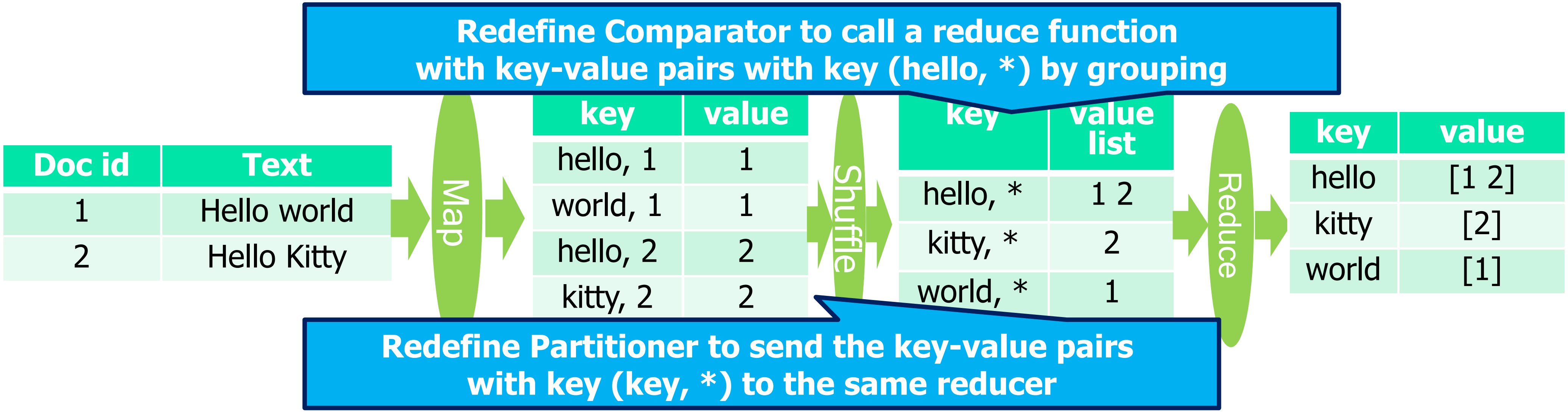
key	value
hello	[2 1]
kitty	[2]
world	[1]

We cannot guarantee that the value list is sorted in the increasing order

- To sort the input value list of each reduce function
  - We need to redefine the Partitioner and Comparator class

(3) Redefine Partitioning Scheme of Shuffling

- Assume we want that the key-value pairs are assigned to reducers by the first value in the keys
  - Override Partitioner class used in map phase
- Assume we want that the key-value pairs, output by map functions, are ordered by (first value, second value) in the keys
  - Override key class by extending Writable class
- Assume we want that the key-value pairs, the key-value pairs are assigned to reduce functions by the first value in the keys
  - Override Comparator class used in shuffling phase



Redefine Comparator to call a reduce function with key-value pairs with key (hello, *) by grouping

Doc id	Text
1	Hello world
2	Hello Kitty

key	value
hello, 1	1
world, 1	1
hello, 2	2
kitty, 2	2

key	value list
hello, *	1 2
kitty, *	2
world, *	1

key	value
hello	[1 2]
kitty	[2]
world	[1]

Redefine Partitioner to send the key-value pairs with key (key, *) to the same reducer

(4) Sharding Data for Multiple MapReduce Phases

- Split data into partitions and store each partition in a predefined machine
  - Sharding for mapper
    - Store the partition P_i in the machine M_i
  - Sharding for reducer
    - Key-value pairs, output by map functions, related to P_i are sent to the machine M_i
- Why sharding?
  - To reduce network overheads by distributing data intentionally when multiple MapReduce phases are used
- An example using sharding: computing PageRank

An Example of Sharding with PageRank Computation

- Let
  - D be the set of all Web pages
  - I(p) be the set of pages that link to the page p
  - |O(q)| be the total number of links going out of page q
- The PageRank of page p, denoted by PR(p), is

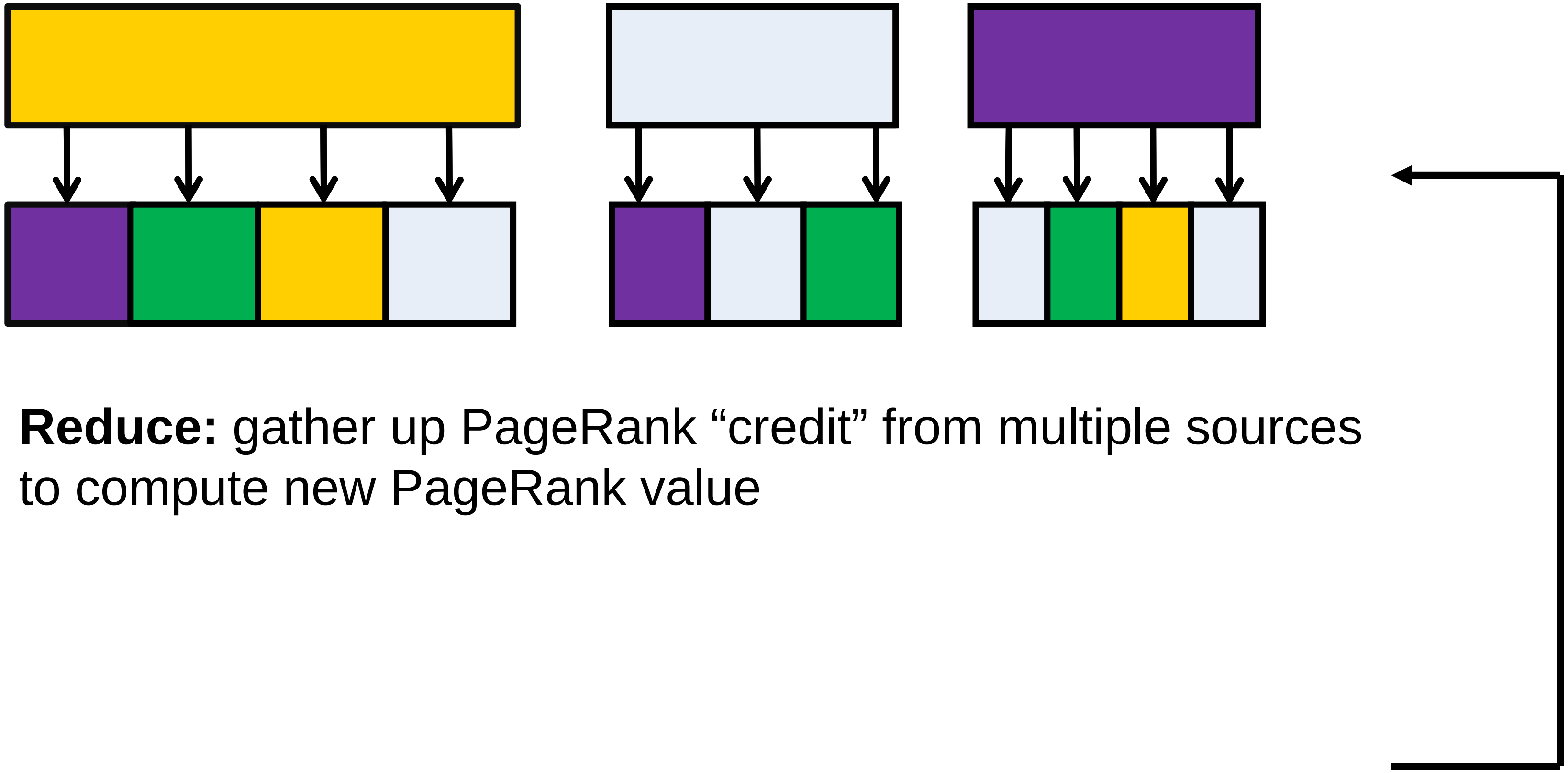
$$PR(p) = d\left[\sum_{q \in I(p)} \frac{PR(q)}{|O(q)|}\right] + (1-d)\frac{1}{|D|}$$

Computing PageRank

- Sketch of PageRank computation
  - Start with current PR(p_i) values
  - Each page p_i distributes current PR(p_i) “credit” evenly to all of its linked pages
  - Each target page adds up “credit” from all in-bound links to compute next PR(p_i) values
  - Iterate until values converge
- Properties of PageRank computation
  - Computed iteratively and effects at each iteration is local
  - Calculation depends on only the PageRank values of previous iteration
  - Individual rows of the adjacency matrix can be processed in parallel

PageRank with MapReduce

Map: distribute PageRank “credit” to link targets

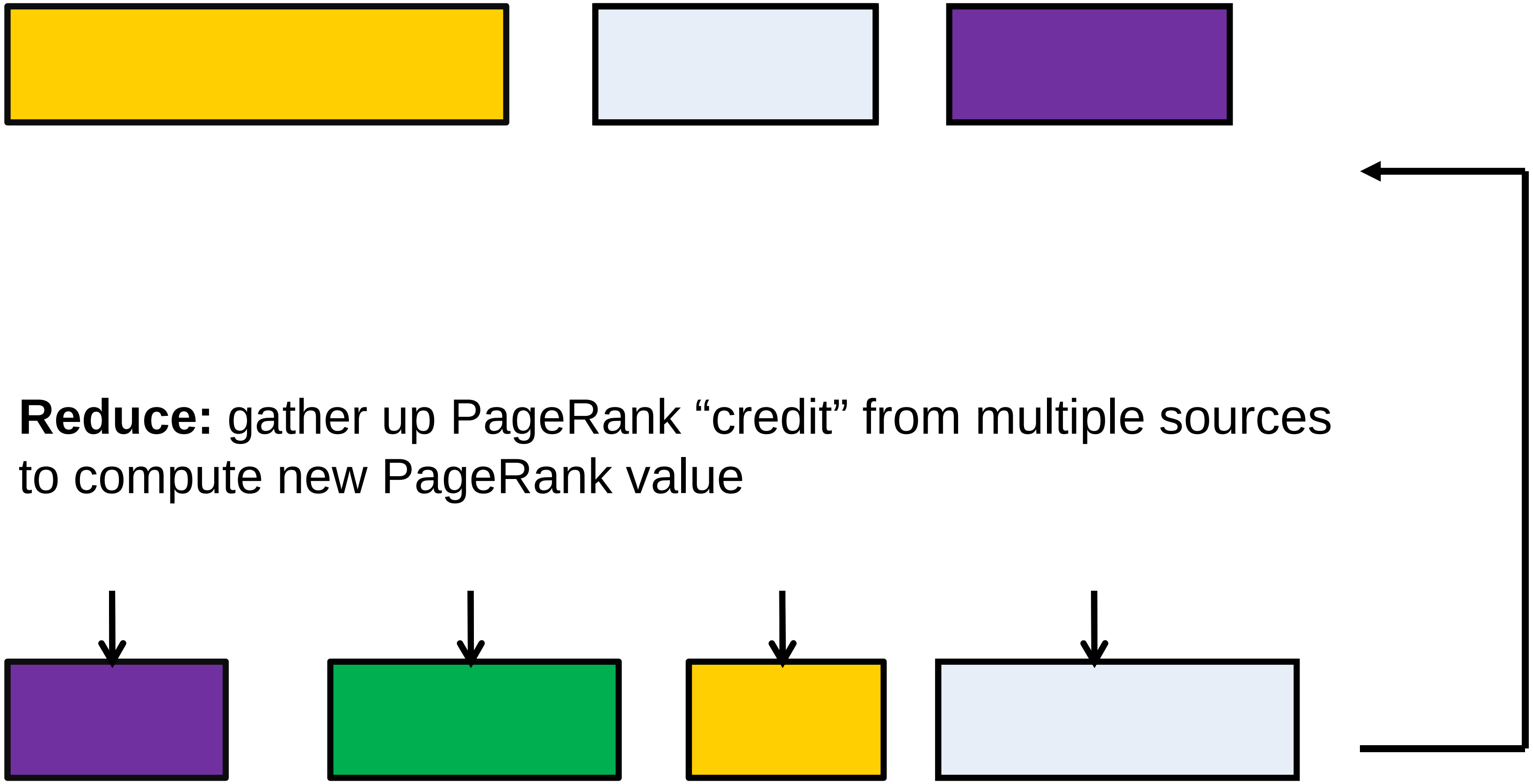


Reduce: gather up PageRank “credit” from multiple sources to compute new PageRank value

Iterate until convergence

PageRank with MapReduce

Map: distribute PageRank “credit” to link targets

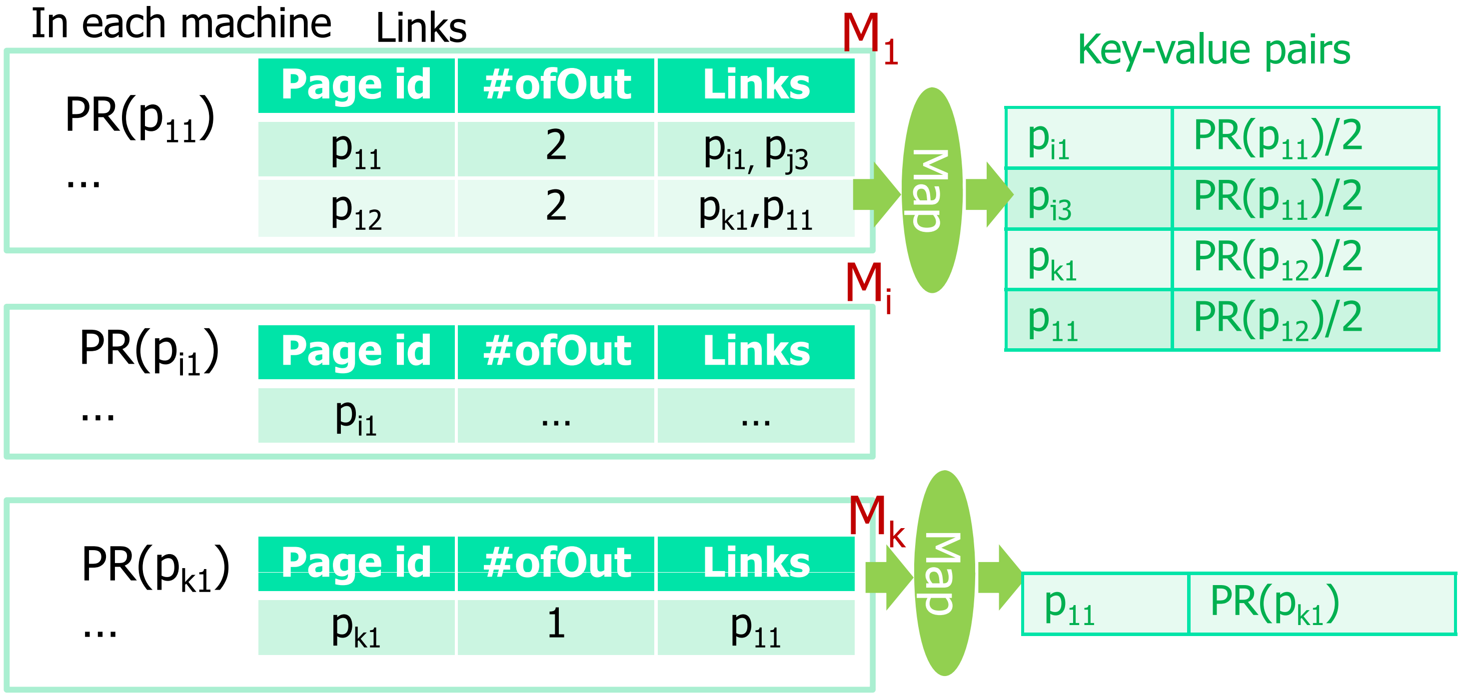


Reduce: gather up PageRank “credit” from multiple sources to compute new PageRank value

Iterate until convergence

Sharding for PageRank

- Sharding for map
  - Split the URLs into k partitions (i.e., P_1, …, P_k)
  - Each machine M_i has page ids in each partition $P_i = \{ p_{i1}, p_{i2}, \ldots, p_{in}\}$ and the adjacent page id list of each page $p_{ij} \in P_i$
  - In each machine M_i, we maintain the computed PageRank values of the URLs in P_i



M_1:

Page id	#ofOut	Links
p_{11}	2	p_{i1}, p_{j3}
p_{12}	2	p_{k1}, p_{11}

Key-value pairs:

p_{i1}	$PR(p_{11})/2$
p_{i3}	$PR(p_{11})/2$
p_{k1}	$PR(p_{12})/2$
p_{11}	$PR(p_{12})/2$

M_i:

Page id	#ofOut	Links
p_{i1}	…	…

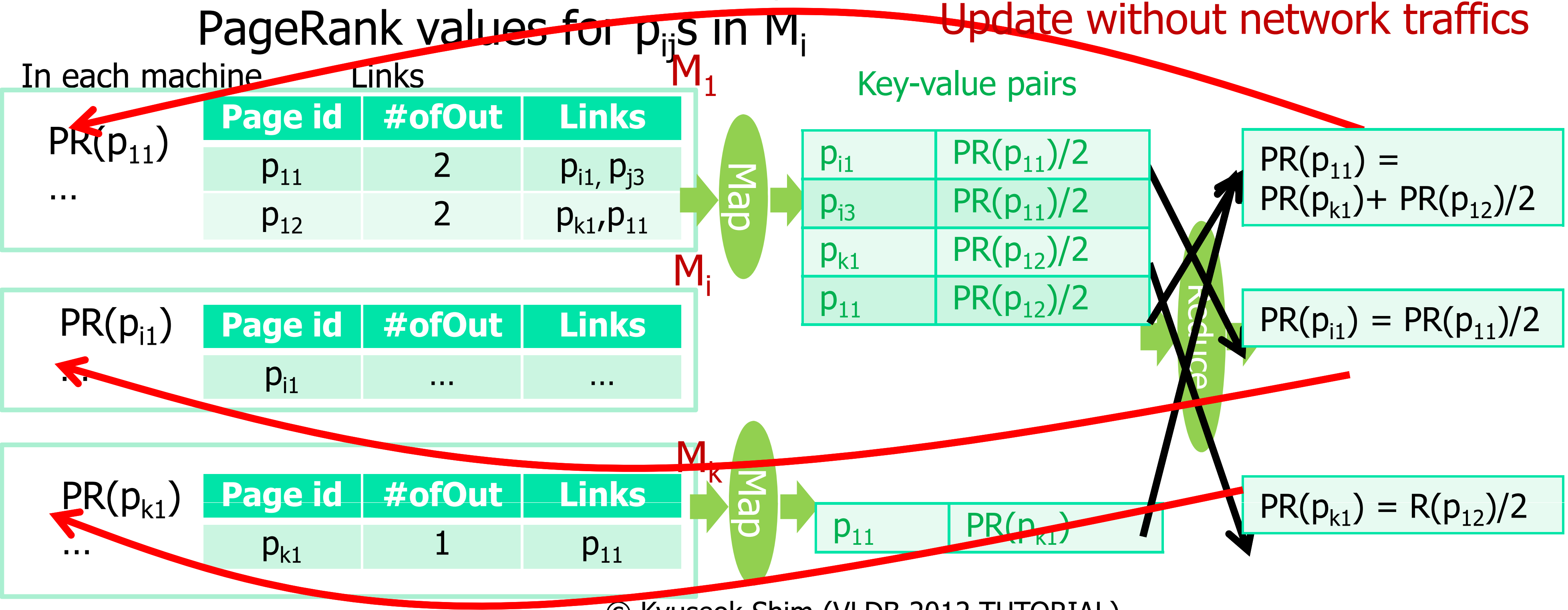
M_k:

Page id	#ofOut	Links
p_{k1}	1	p_{11}

p_{11}	$PR(p_{k1})$

Sharding for PageRank

- Sharding for reduce
  - Each emitted key-value pair <p_{ij}, PR> from map functions goes to the machine M_i
  - Each reducer executed in M_i sums the values and update the PageRank values for p_{ij}s in M_i



Page id	#ofOut	Links
p_{11}	2	p_{i1}, p_{j3}
p_{12}	2	p_{k1},p_{11}

Page id	#ofOut	Links
p_{i1}	…	…

Page id	#ofOut	Links
p_{k1}	1	p_{11}

p_{i1}	PR(p_{11})/2
p_{i3}	PR(p_{11})/2
p_{k1}	PR(p_{12})/2
p_{11}	PR(p_{12})/2

p_{11}	PR(p_{k1})

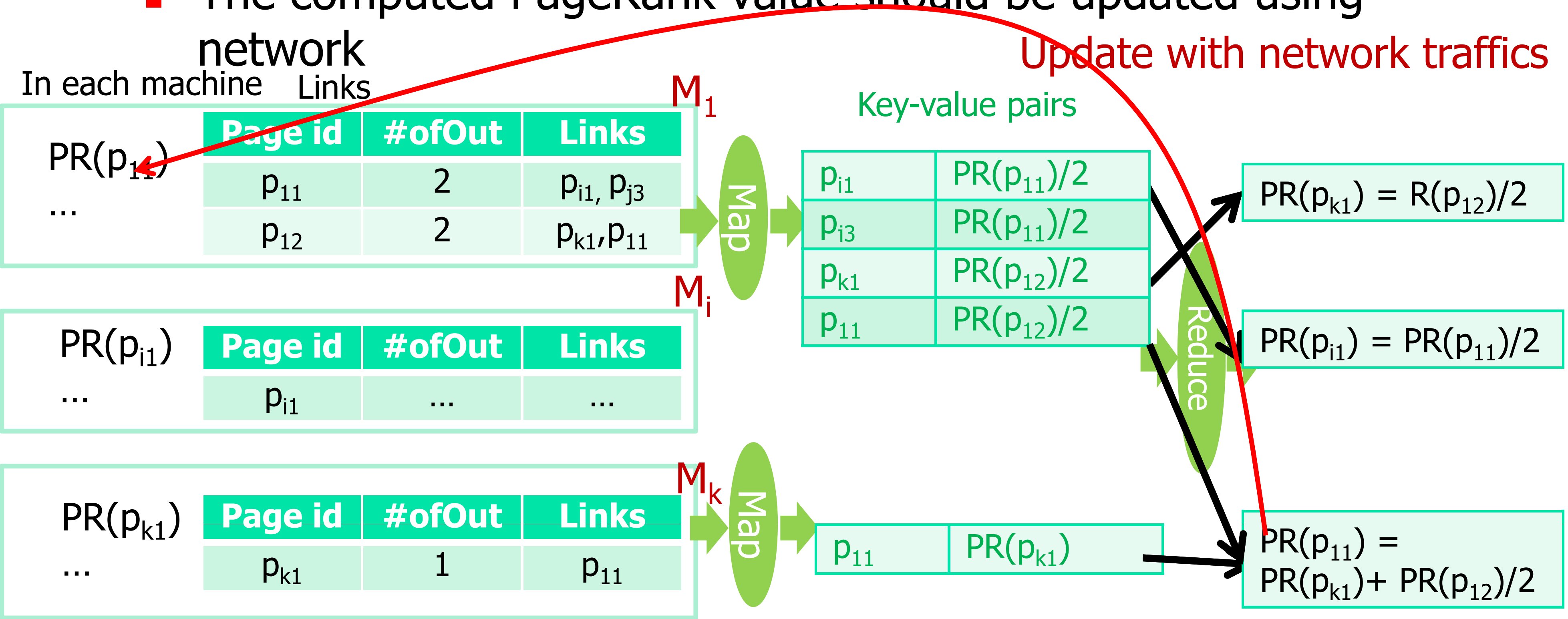
PR(p_{11}) = PR(p_{k1})+ PR(p_{12})/2

PR(p_{i1}) = PR(p_{11})/2

PR(p_{k1}) = R(p_{12})/2

PageRank without Sharding

- Without sharding for reduce
  - Each emitted key-value pair <p_{ij}, PR> from map functions can go to any machine
  - The computed PageRank value should be updated using network



Update with network traffics

In each machine

Links

M_1

PR(p_{11})

…

Page id	#ofOut	Links
p_{11}	2	p_{i1}, p_{j3}
p_{12}	2	p_{k1},p_{11}

M_i

PR(p_{i1})

…

Page id	#ofOut	Links
p_{i1}	…	…

M_k

PR(p_{k1})

…

Page id	#ofOut	Links
p_{k1}	1	p_{11}

Key-value pairs

p_{i1}	PR(p_{11})/2
p_{i3}	PR(p_{11})/2
p_{k1}	PR(p_{12})/2
p_{11}	PR(p_{12})/2

p_{11}	PR(p_{k1})

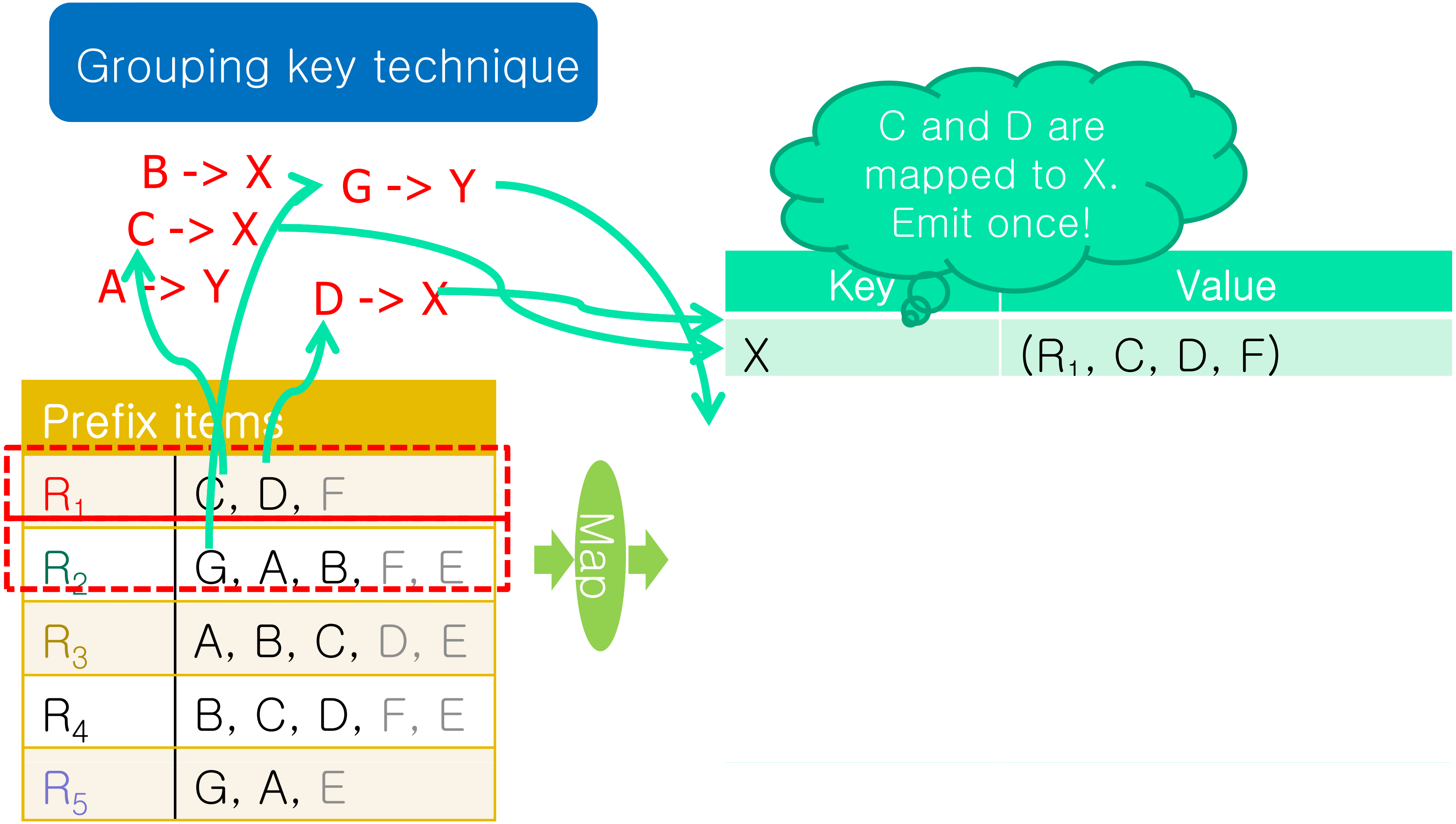
PR(p_{k1}) = R(p_{12})/2

PR(p_{i1}) = PR(p_{11})/2

PR(p_{11}) = PR(p_{k1})+ PR(p_{12})/2

(5) Grouping Keys

Set similarity join [Vernica, Carey, Li: SIGMOD 2010]

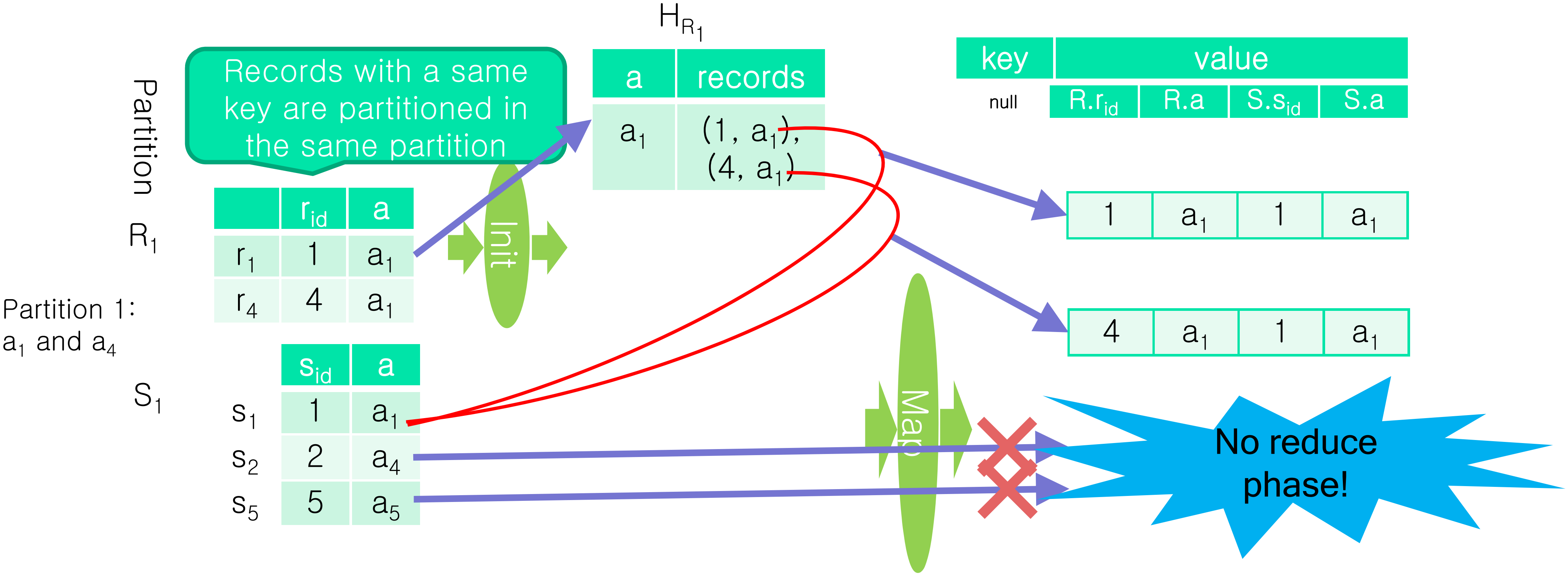


Prefix items	
R₁	C, D, F
R₂	G, A, B, F, E
R₃	A, B, C, D, E
R₄	B, C, D, F, E
R₅	G, A, E

Key	Value
X	(R_1, C, D, F)

(6) No Reduce Phase

- Repartition Join with Pre-partitioning [Blanas, Patel, Ercegovac, Rao, Shekita, Tian: SIGMOD 2010]



R_1

	r_{id}	a
r_1	1	a_1
r_4	4	a_1

S_1

	s_{id}	a
s_1	1	a_1
s_2	2	a_4
s_5	5	a_5

H_{R_1}

a	records
a_1	$(1, a_1)$, $(4, a_1)$

key	value			
	$R.r_{id}$	R.a	$S.s_{id}$	S.a
null				
	1	a_1	1	a_1
	4	a_1	1	a_1

Roadmap of MapReduce Algorithms

- Joins
  - Theta-joins
  - Similarity joins
  - Join order optimizations
- Data mining
  - Clustering
  - Probabilistic modeling
  - Association rule mining
  - Classification
  - Graph analysis
- Potpourri

Theta-Join Algorithms using MapReduce

Theta Joins

- Use primitive comparison operators (<,>,≤,≥,≠,=) in the join-predicates

```
SELECT *
FROM R, S
WHERE R.a > S.a;
```

R

	r_{id}	a
r_1	1	1
r_2	2	1
r_3	3	2
r_4	4	3

S

	s_{id}	a
s_1	1	1
s_2	2	1
s_3	3	2
s_4	4	2
s_5	5	3
s_6	6	4

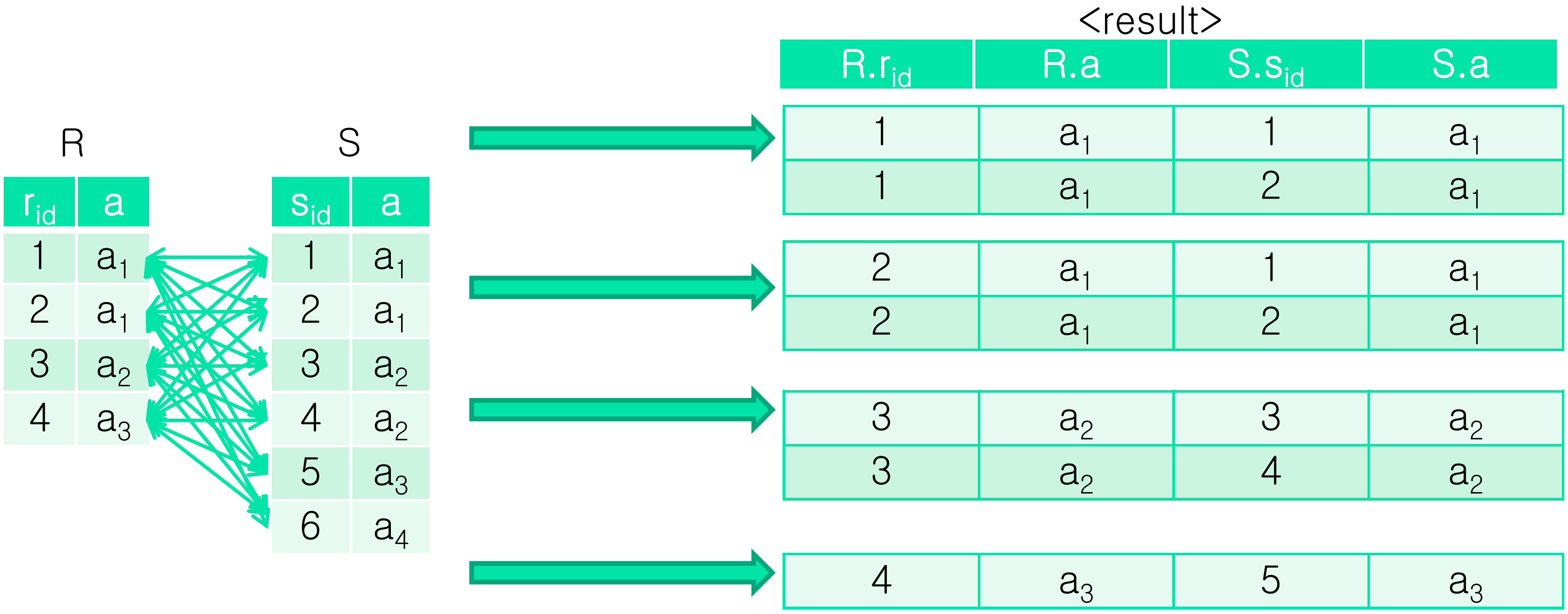
Equi-Join Algorithms

- [Blanas, Patel, Ercegovac, Rao, Shekita, Tian: SIGMOD 2010]
- [Okcan, Riedewald: SIGMOD 2011]

- All pair partitioning join algorithm
- Repartition join algorithms
  - Standard repartition
  - Improved repartition
  - Repartition with pre-partitioning
- Broadcast join algorithm
- Semi-join algorithms
  - Semi-join
  - Per-split semi-join

An Illustration of Equi-Joins

```
SELECT * FROM R, S WHERE R.a = S.a;
```



R

r_{id}	a
1	a_1
2	a_1
3	a_2
4	a_3

S

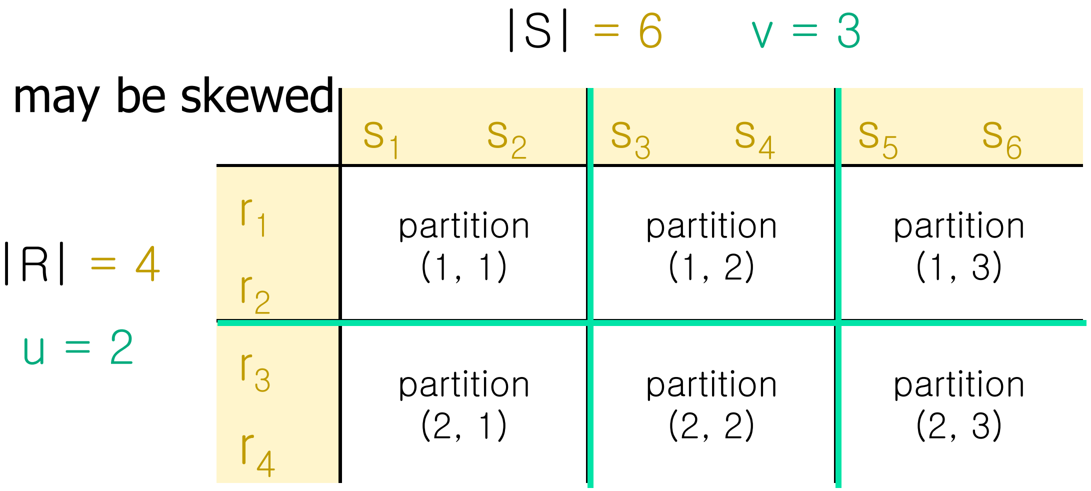
s_{id}	a
1	a_1
2	a_1
3	a_2
4	a_2
5	a_3
6	a_4

<result>

$R.r_{id}$	R.a	$S.s_{id}$	S.a
1	a_1	1	a_1
1	a_1	2	a_1
2	a_1	1	a_1
2	a_1	2	a_1
3	a_2	3	a_2
3	a_2	4	a_2
4	a_3	5	a_3

All Pair Partitioning Algorithm

- For tables R and S, consider |R|*|S| pairs of records
  - Splitting R and S into u and v partitions respectively
  - Divide |R|*|S| pairs of records into u*v disjoint partitions
  - Process each partition by a reduce function
- Advantages
  - Works for any join-predicate
  - Input sizes of reduce functions are similar
- Disadvantages
  - Enumerate all pairs
  - Output sizes of reduce functions may be skewed



|S| = 6 v = 3

|R| = 4

u = 2

	s_1 s_2	s_3 s_4	s_5 s_6
r_1 r_2	partition (1, 1)	partition (1, 2)	partition (1, 3)
r_3 r_4	partition (2, 1)	partition (2, 2)	partition (2, 3)

All Pair Partitioning Algorithm

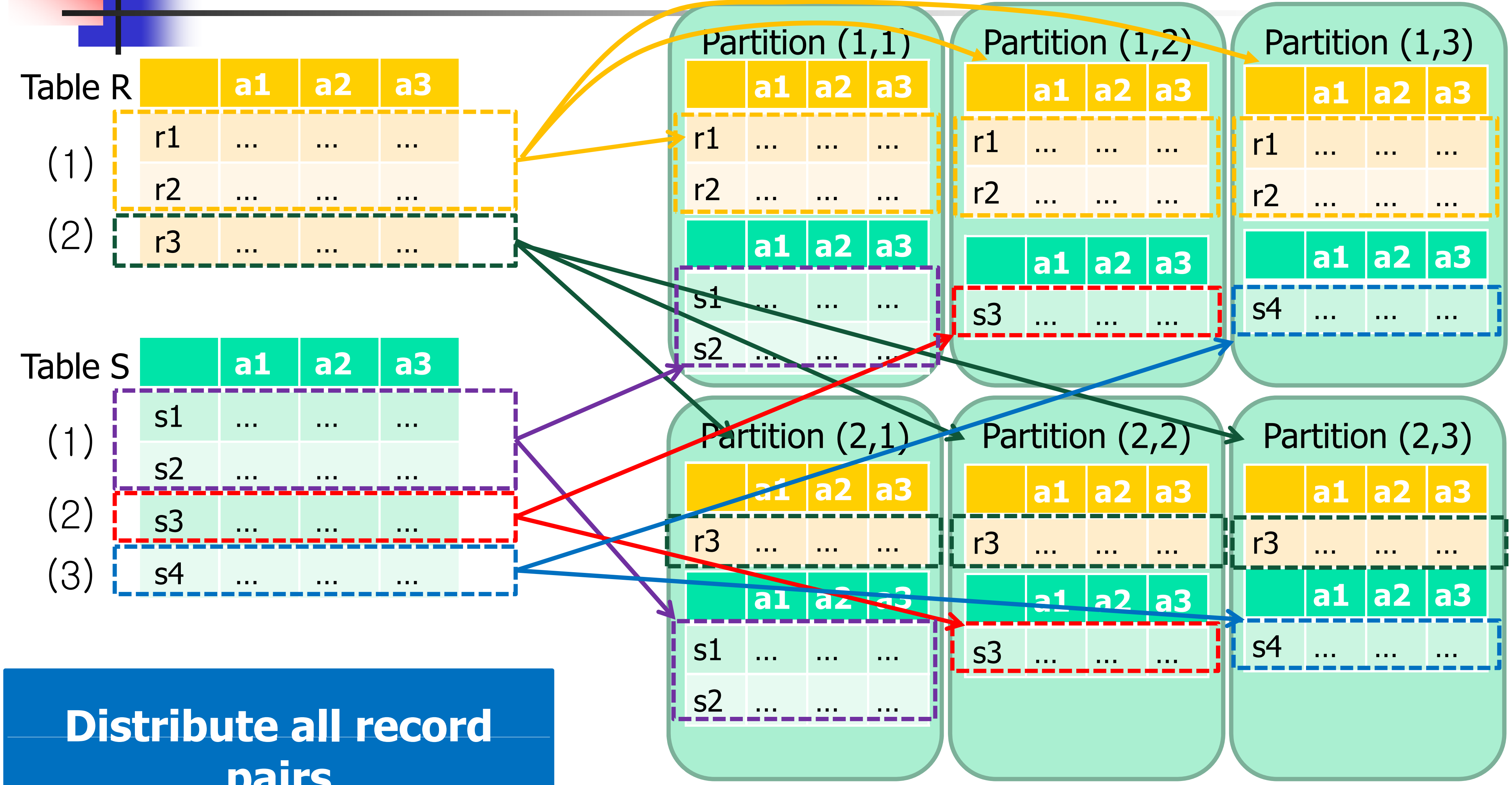


Table R

		a1	a2	a3
(1)	r1	…	…	…
	r2	…	…	…
(2)	r3	…	…	…

Table S

		a1	a2	a3
(1)	s1	…	…	…
	s2	…	…	…
(2)	s3	…	…	…
(3)	s4	…	…	…

Partition (1,1)

	a1	a2	a3
r1	…	…	…
r2	…	…	…

	a1	a2	a3
s1	…	…	…
s2	…	…	…

Partition (1,2)

	a1	a2	a3
r1	…	…	…
r2	…	…	…

	a1	a2	a3
s3	…	…	…

Partition (1,3)

	a1	a2	a3
r1	…	…	…
r2	…	…	…

	a1	a2	a3
s4	…	…	…

Partition (2,1)

	a1	a2	a3
r3	…	…	…

	a1	a2	a3
s1	…	…	…
s2	…	…	…

Partition (2,2)

	a1	a2	a3
r3	…	…	…

	a1	a2	a3
s3	…	…	…

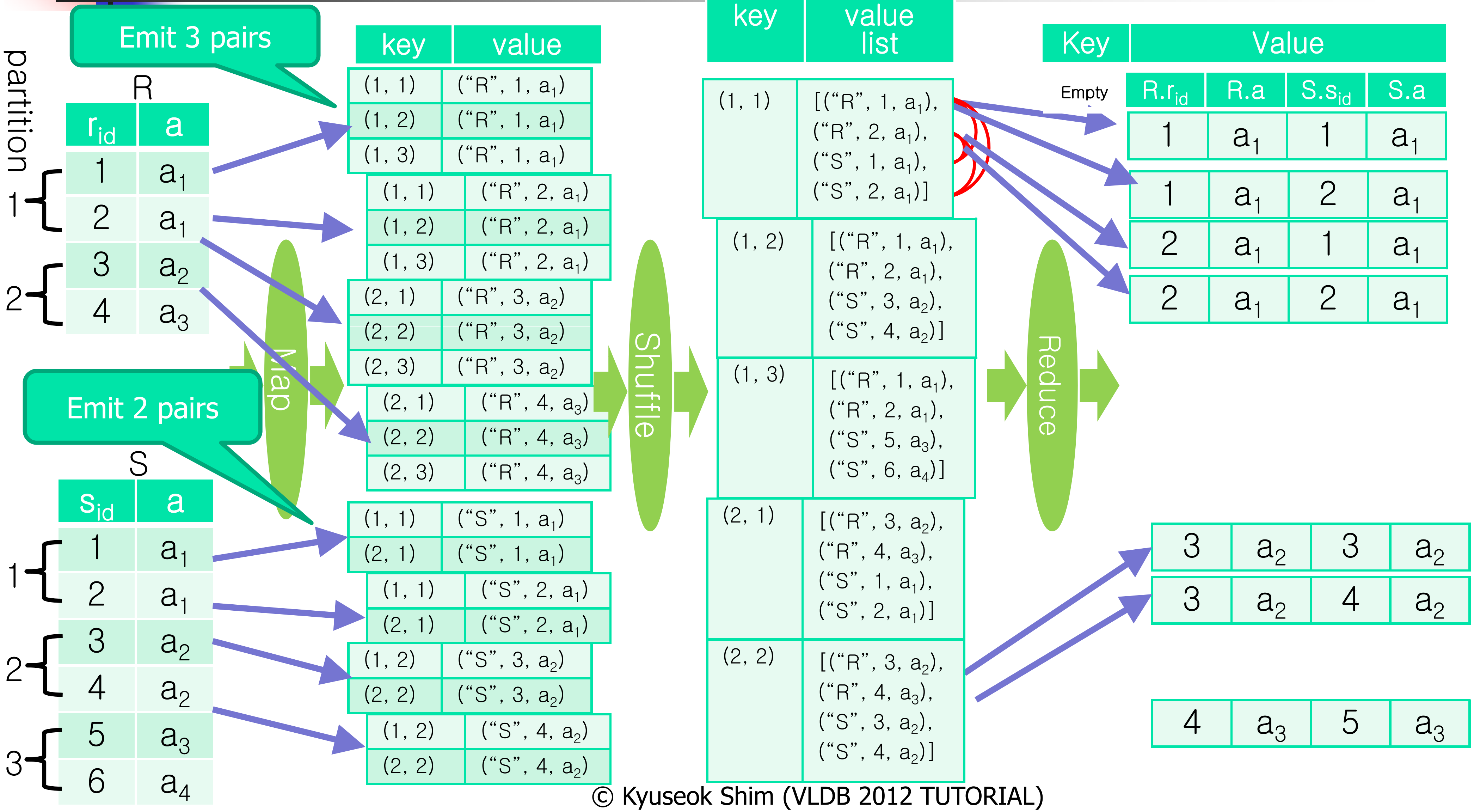
Partition (2,3)

	a1	a2	a3
r3	…	…	…

	a1	a2	a3
s4	…	…	…

Distribute all record pairs

An Illustration of All Pair Partitioning using MapReduce



Emit 3 pairs

R

partition	r_{id}	a
1	1	a_1
	2	a_1
2	3	a_2
	4	a_3

Emit 2 pairs

S

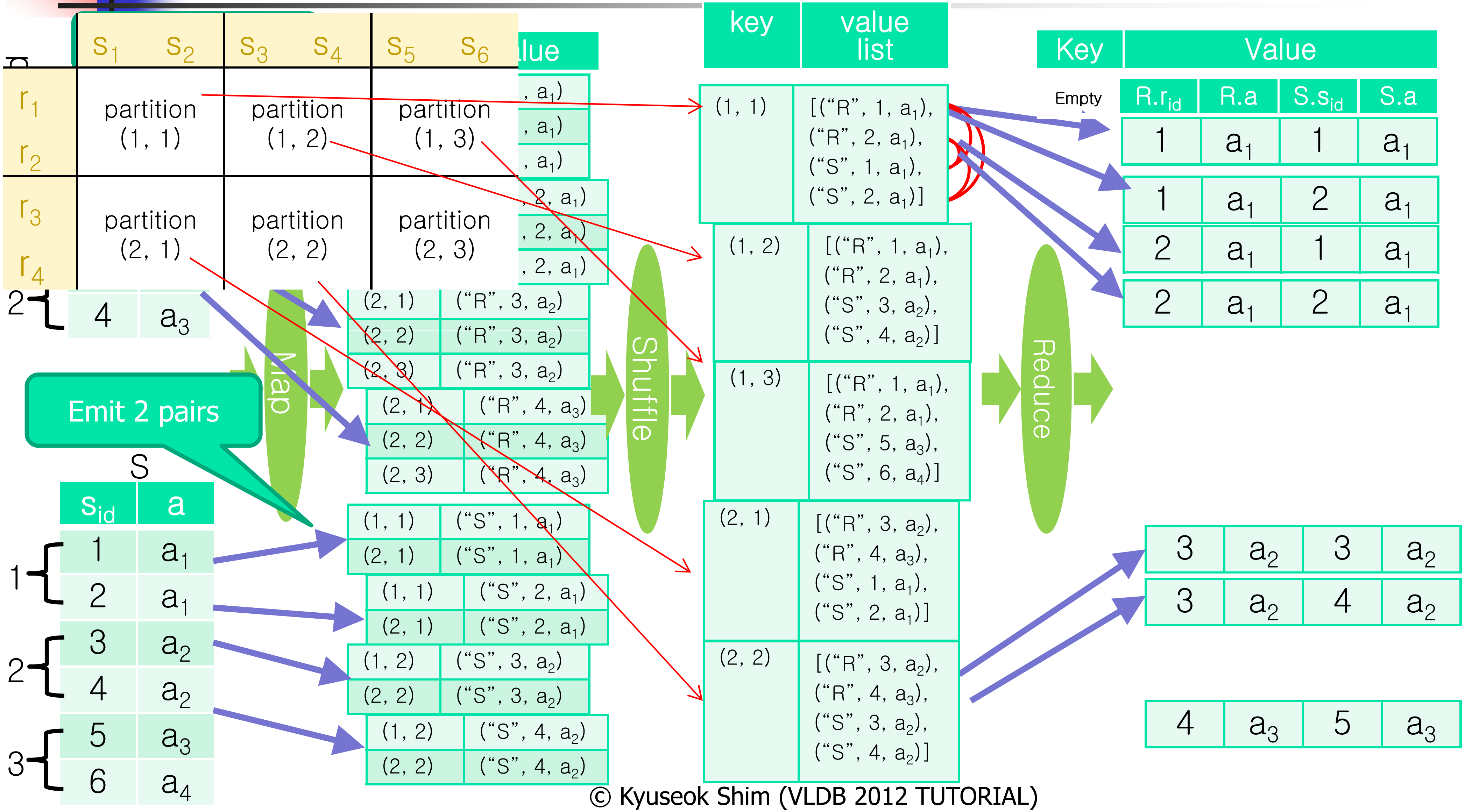
	s_{id}	a
1	1	a_1
	2	a_1
2	3	a_2
	4	a_2
3	5	a_3
	6	a_4

key	value
(1, 1)	(“R”, 1, a_1)
(1, 2)	(“R”, 1, a_1)
(1, 3)	(“R”, 1, a_1)
(1, 1)	(“R”, 2, a_1)
(1, 2)	(“R”, 2, a_1)
(1, 3)	(“R”, 2, a_1)
(2, 1)	(“R”, 3, a_2)
(2, 2)	(“R”, 3, a_2)
(2, 3)	(“R”, 3, a_2)
(2, 1)	(“R”, 4, a_3)
(2, 2)	(“R”, 4, a_3)
(2, 3)	(“R”, 4, a_3)
(1, 1)	(“S”, 1, a_1)
(2, 1)	(“S”, 1, a_1)
(1, 1)	(“S”, 2, a_1)
(2, 1)	(“S”, 2, a_1)
(1, 2)	(“S”, 3, a_2)
(2, 2)	(“S”, 3, a_2)
(1, 2)	(“S”, 4, a_2)
(2, 2)	(“S”, 4, a_2)

key	value list
(1, 1)	[(“R”, 1, a_1), (“R”, 2, a_1), (“S”, 1, a_1), (“S”, 2, a_1)]
(1, 2)	[(“R”, 1, a_1), (“R”, 2, a_1), (“S”, 3, a_2), (“S”, 4, a_2)]
(1, 3)	[(“R”, 1, a_1), (“R”, 2, a_1), (“S”, 5, a_3), (“S”, 6, a_4)]
(2, 1)	[(“R”, 3, a_2), (“R”, 4, a_3), (“S”, 1, a_1), (“S”, 2, a_1)]
(2, 2)	[(“R”, 3, a_2), (“R”, 4, a_3), (“S”, 3, a_2), (“S”, 4, a_2)]

Key	Value			
Empty	R.r_{id}	R.a	S.s_{id}	S.a
	1	a_1	1	a_1
	1	a_1	2	a_1
	2	a_1	1	a_1
	2	a_1	2	a_1
	3	a_2	3	a_2
	3	a_2	4	a_2
	4	a_3	5	a_3

An Illustration of All Pair Partitioning using MapReduce



	s1 s2	s3 s4	s5 s6
r1 r2	partition (1, 1)	partition (1, 2)	partition (1, 3)
r3 r4	partition (2, 1)	partition (2, 2)	partition (2, 3)

4	a3

S

sid	a
1	a1
2	a1
3	a2
4	a2
5	a3
6	a4

	lue
	, a1)
	, a1)
	, a1)
	2, a1)
	, 2, a1)
	, 2, a1)
(2, 1)	("R", 3, a_2)
(2, 2)	("R", 3, a_2)
(2, 3)	("R", 3, a_2)
(2, 1)	("R", 4, a_3)
(2, 2)	("R", 4, a_3)
(2, 3)	("R", 4, a_3)
(1, 1)	("S", 1, a_1)
(2, 1)	("S", 1, a_1)
(1, 1)	("S", 2, a_1)
(2, 1)	("S", 2, a_1)
(1, 2)	("S", 3, a_2)
(2, 2)	("S", 3, a_2)
(1, 2)	("S", 4, a_2)
(2, 2)	("S", 4, a_2)

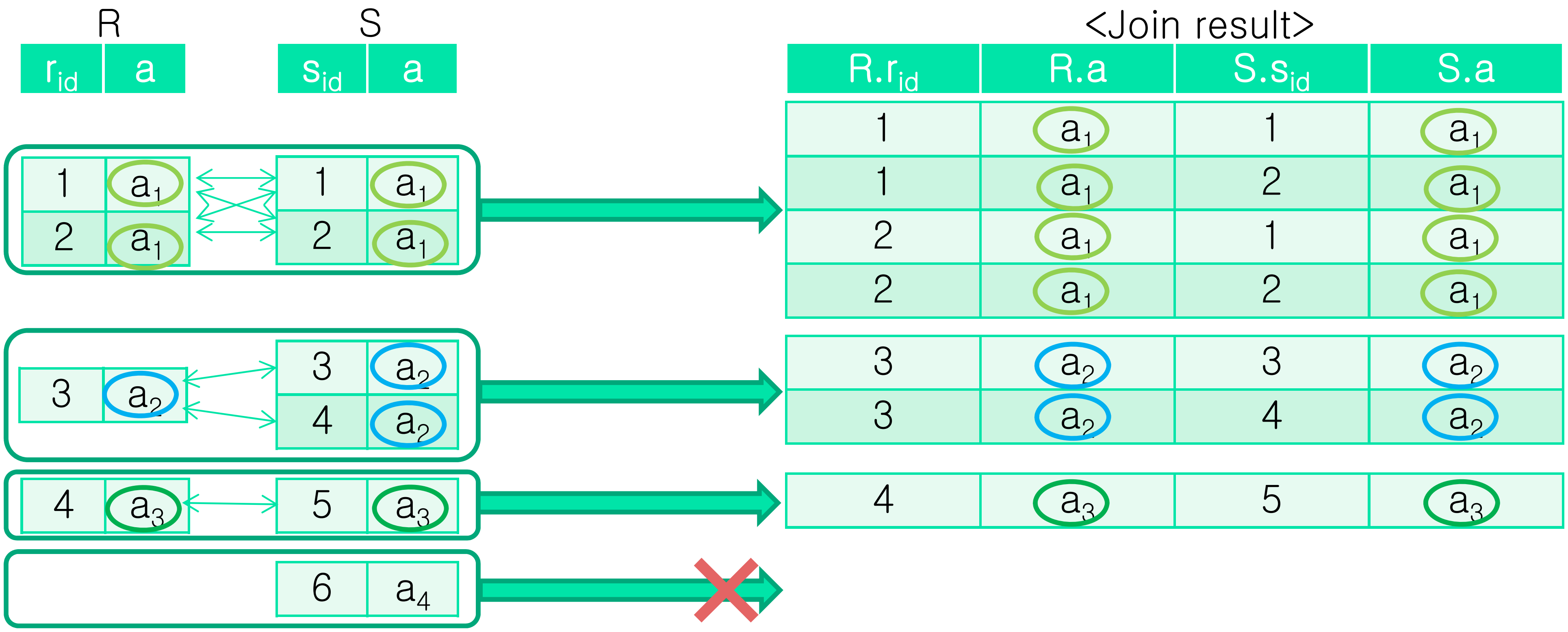
key	value list
(1, 1)	[("R", 1, a_1), ("R", 2, a_1), ("S", 1, a_1), ("S", 2, a_1)]
(1, 2)	[("R", 1, a_1), ("R", 2, a_1), ("S", 3, a_2), ("S", 4, a_2)]
(1, 3)	[("R", 1, a_1), ("R", 2, a_1), ("S", 5, a_3), ("S", 6, a_4)]
(2, 1)	[("R", 3, a_2), ("R", 4, a_3), ("S", 1, a_1), ("S", 2, a_1)]
(2, 2)	[("R", 3, a_2), ("R", 4, a_3), ("S", 3, a_2), ("S", 4, a_2)]

Key	Value			
	R.rid	R.a	S.sid	S.a
Empty				
	1	a_1	1	a_1
	1	a_1	2	a_1
	2	a_1	1	a_1
	2	a_1	2	a_1
	3	a_2	3	a_2
	3	a_2	4	a_2
	4	a_3	5	a_3

© Kyuseok Shim (VLDB 2012 TUTORIAL)

Remember Hash Joins!

```
SELECT * FROM R, S WHERE R.a = S.a;
```



R

r_{id}	a
1	a_1
2	a_1
3	a_2
4	a_3

S

s_{id}	a
1	a_1
2	a_1
3	a_2
4	a_2
5	a_3
6	a_4

<Join result>

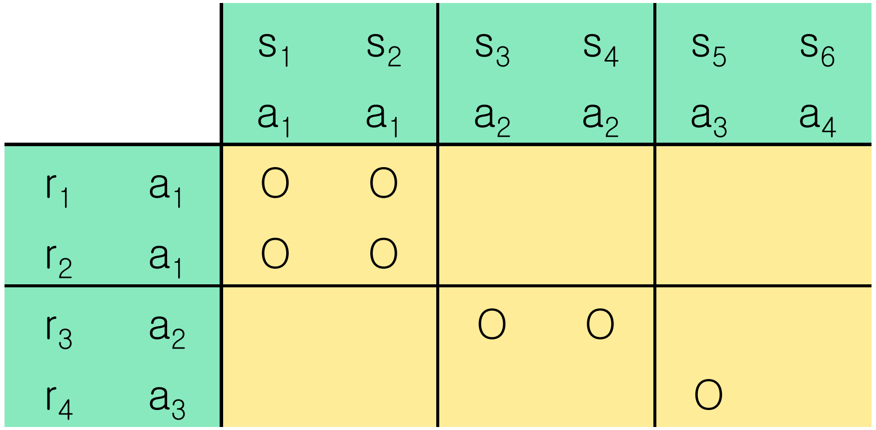
$R.r_{id}$	R.a	$S.s_{id}$	S.a
1	a_1	1	a_1
1	a_1	2	a_1
2	a_1	1	a_1
2	a_1	2	a_1
3	a_2	3	a_2
3	a_2	4	a_2
4	a_3	5	a_3

Standard Repartition Equi-Join Algorithm

Like hash join

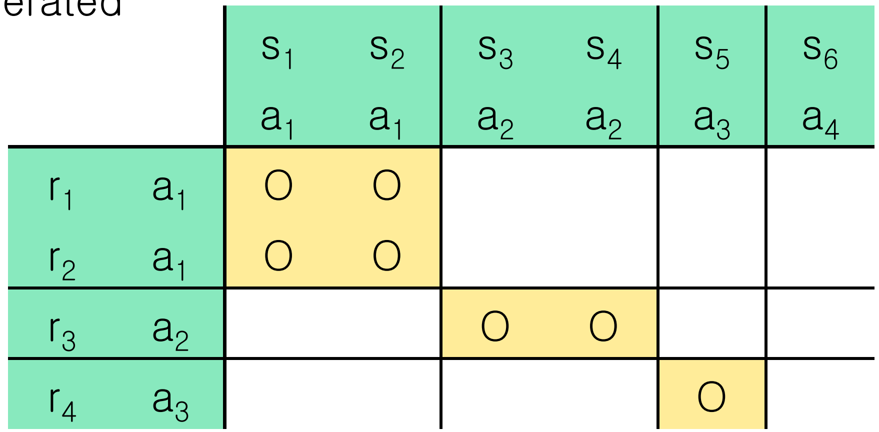
- [Okcan, Riedewald: SIGMOD 2011]
- Consider only the pairs with the same join attribute values
- A map function
  - Receives a record in R and S
  - Emits its join attribute value as a key and the record as a value
- A reduce function
  - Receives each join attribute value with its records from R and S
  - Emits all pairs between the records in R and S

🟨 enumerated pairs



		s_1	s_2	s_3	s_4	s_5	s_6
		a_1	a_1	a_2	a_2	a_3	a_4
r_1	a_1	O	O				
r_2	a_1	O	O				
r_3	a_2			O	O		
r_4	a_3					O	

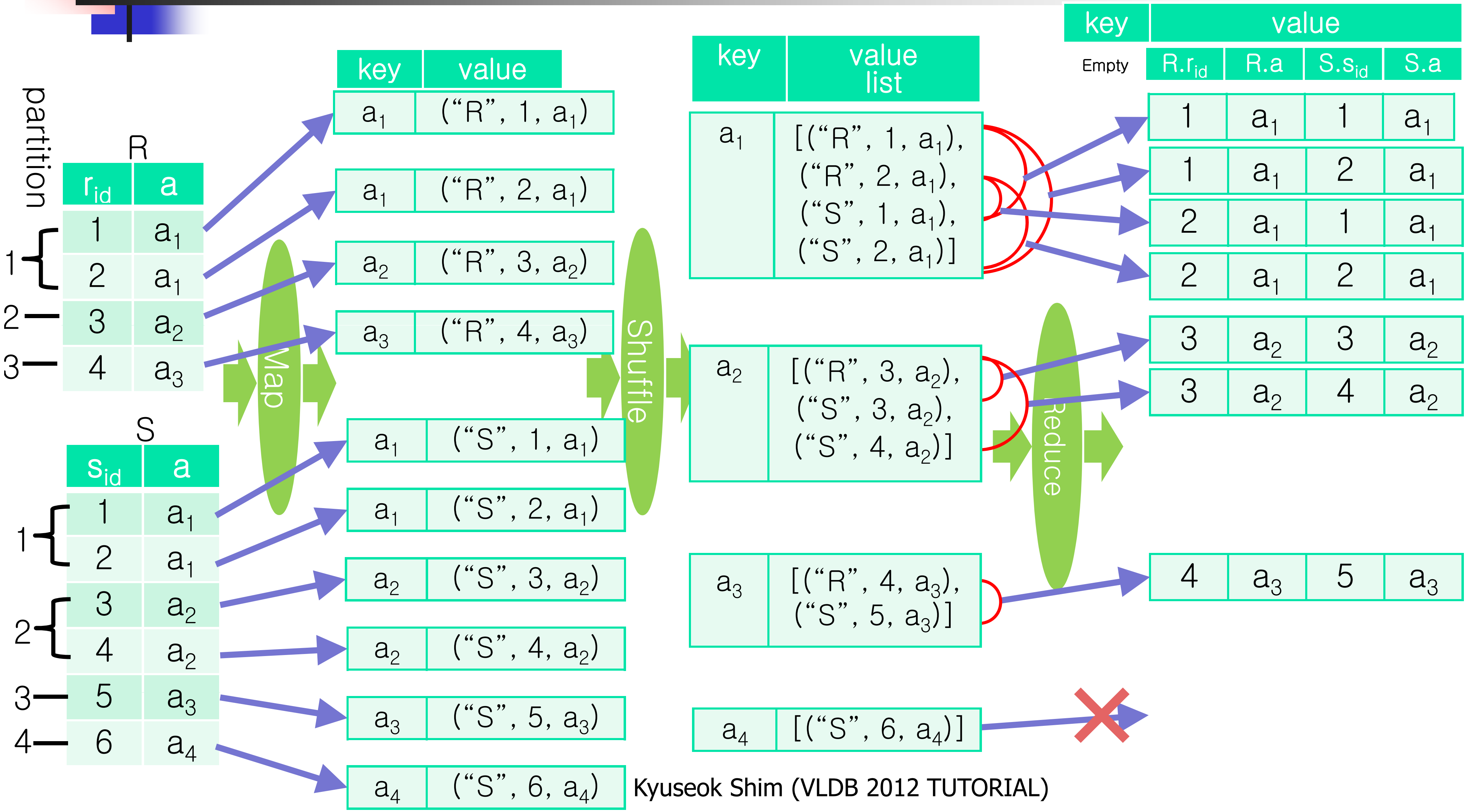
Naïve join algorithm



		s_1	s_2	s_3	s_4	s_5	s_6
		a_1	a_1	a_2	a_2	a_3	a_4
r_1	a_1	O	O				
r_2	a_1	O	O				
r_3	a_2			O	O		
r_4	a_3					O	

Standard repartition join algorithm

An Illustration of Standard Repartition Equi-Join Algorithm



R

partition	r_{id}	a
1	1	a_1
	2	a_1
2	3	a_2
3	4	a_3

S

partition	s_{id}	a
1	1	a_1
	2	a_1
2	3	a_2
	4	a_2
3	5	a_3
4	6	a_4

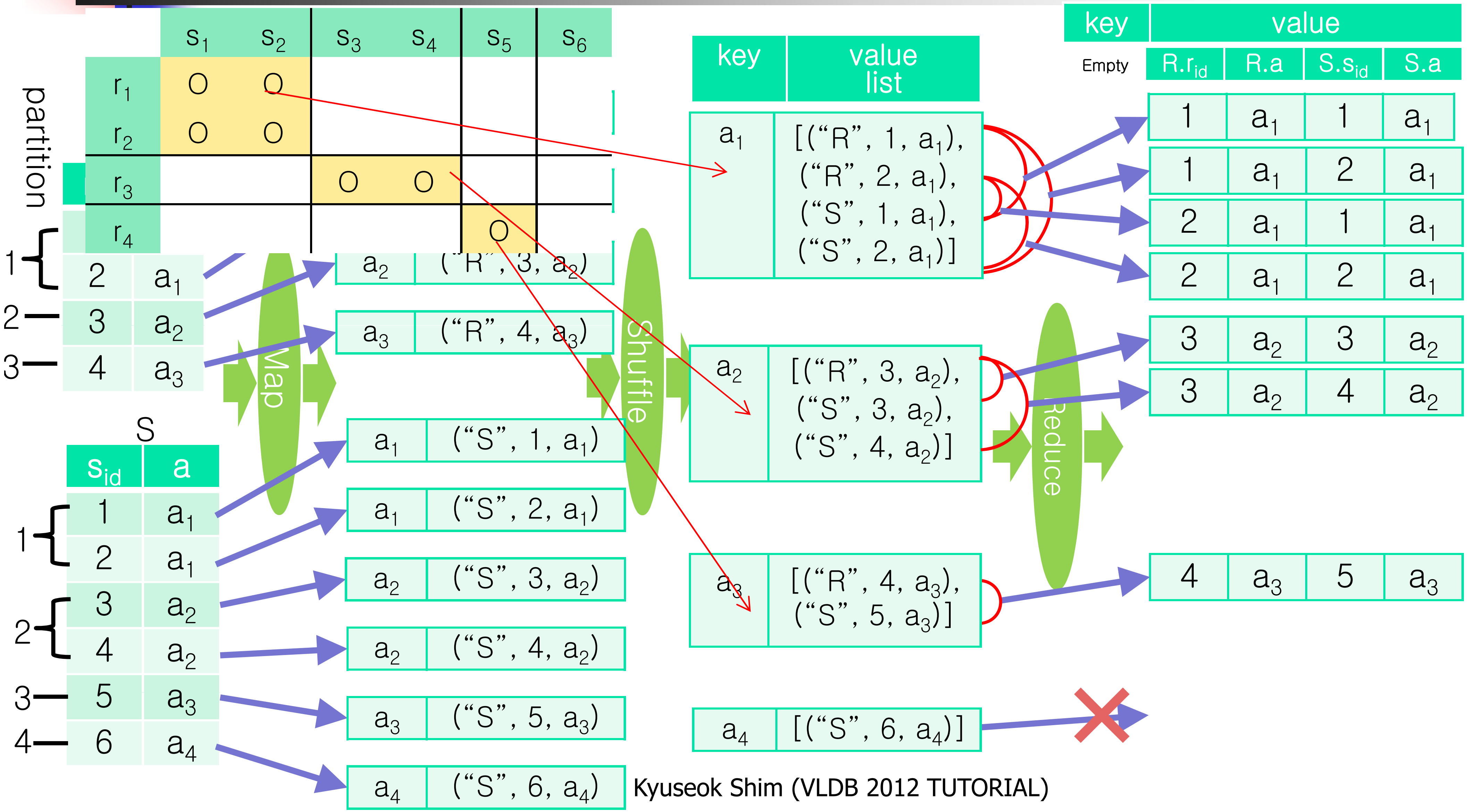
key	value
a_1	(“R”, 1, a_1)
a_1	(“R”, 2, a_1)
a_2	(“R”, 3, a_2)
a_3	(“R”, 4, a_3)
a_1	(“S”, 1, a_1)
a_1	(“S”, 2, a_1)
a_2	(“S”, 3, a_2)
a_2	(“S”, 4, a_2)
a_3	(“S”, 5, a_3)
a_4	(“S”, 6, a_4)

key	value list
a_1	[(“R”, 1, a_1), (“R”, 2, a_1), (“S”, 1, a_1), (“S”, 2, a_1)]
a_2	[(“R”, 3, a_2), (“S”, 3, a_2), (“S”, 4, a_2)]
a_3	[(“R”, 4, a_3), (“S”, 5, a_3)]
a_4	[(“S”, 6, a_4)]

key	value			
Empty	R.r_{id}	R.a	S.s_{id}	S.a
	1	a_1	1	a_1
	1	a_1	2	a_1
	2	a_1	1	a_1
	2	a_1	2	a_1
	3	a_2	3	a_2
	3	a_2	4	a_2
	4	a_3	5	a_3

Kyuseok Shim (VLDB 2012 TUTORIAL)

An Illustration of Standard Repartition Equi-Join Algorithm



	s_1	s_2	s_3	s_4	s_5	s_6
r_1	O	O				
r_2	O	O				
r_3			O	O		
r_4					O	

2	a_1
3	a_2
4	a_3

S

s_{id}	a
1	a_1
2	a_1
3	a_2
4	a_2
5	a_3
6	a_4

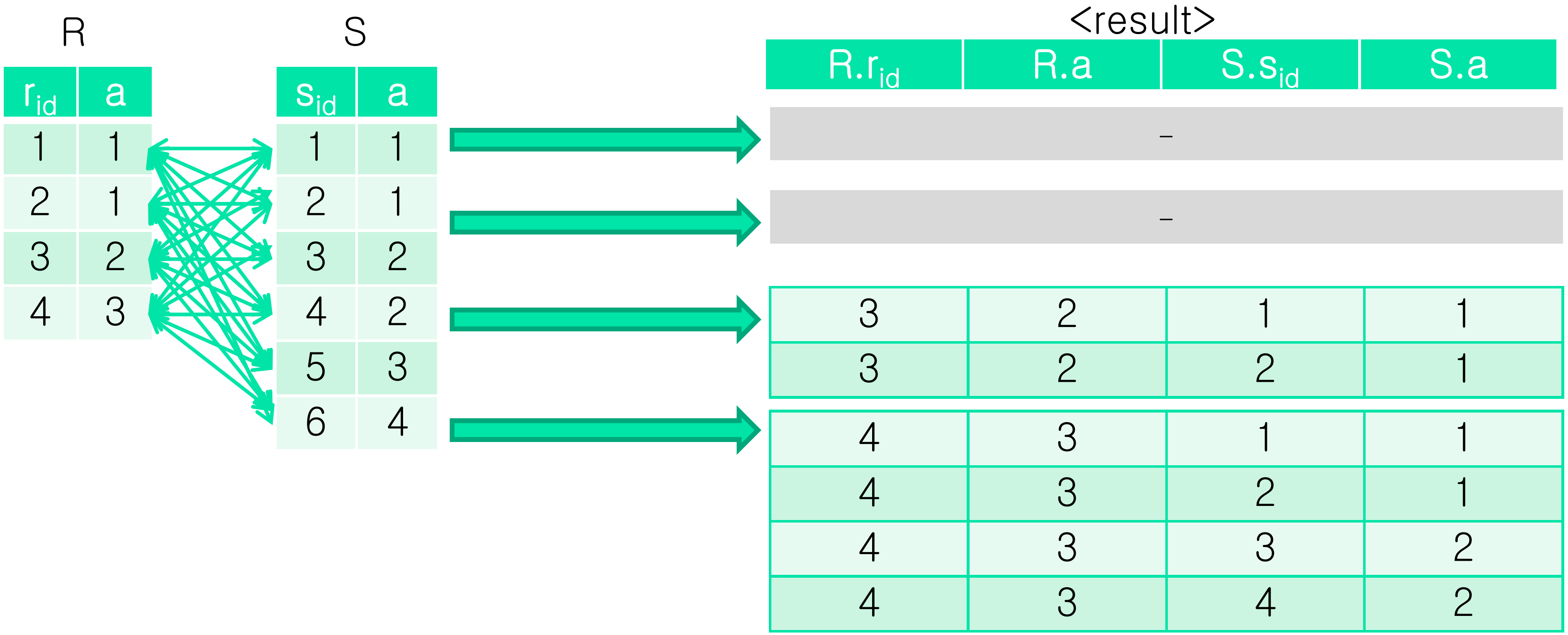
a_2	(“R”, 3, a_2)
a_3	(“R”, 4, a_3)
a_1	(“S”, 1, a_1)
a_1	(“S”, 2, a_1)
a_2	(“S”, 3, a_2)
a_2	(“S”, 4, a_2)
a_3	(“S”, 5, a_3)
a_4	(“S”, 6, a_4)

key	value list
a_1	[(“R”, 1, a_1), (“R”, 2, a_1), (“S”, 1, a_1), (“S”, 2, a_1)]
a_2	[(“R”, 3, a_2), (“S”, 3, a_2), (“S”, 4, a_2)]
a_3	[(“R”, 4, a_3), (“S”, 5, a_3)]
a_4	[(“S”, 6, a_4)]

key	value			
Empty	$R.r_{id}$	R.a	$S.s_{id}$	S.a
	1	a_1	1	a_1
	1	a_1	2	a_1
	2	a_1	1	a_1
	2	a_1	2	a_1
	3	a_2	3	a_2
	3	a_2	4	a_2
	4	a_3	5	a_3

An Example of Theta-Join

`SELECT * FROM R, S WHERE R.a > S.a;`



R

r_{id}	a
1	1
2	1
3	2
4	3

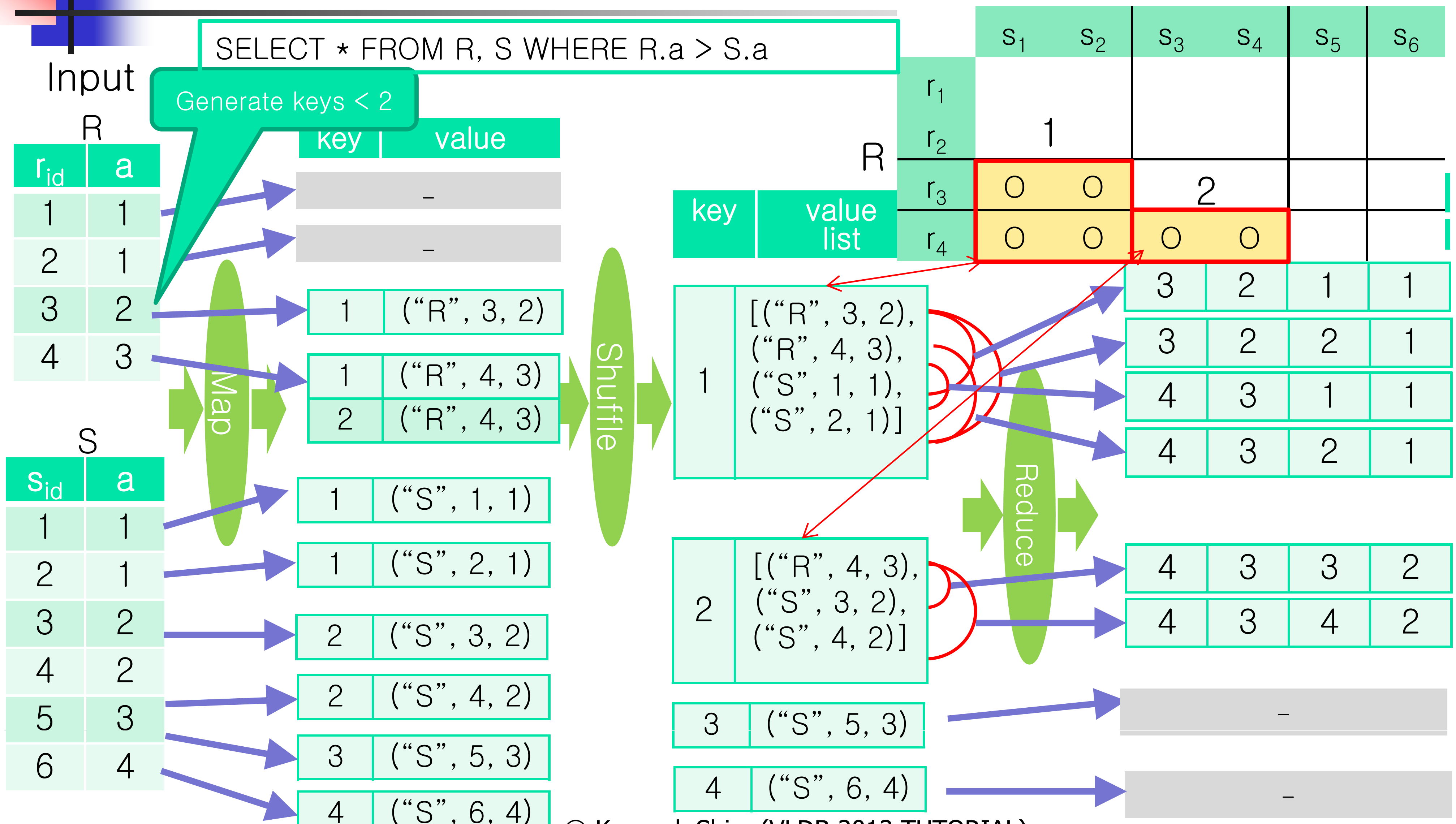
S

s_{id}	a
1	1
2	1
3	2
4	2
5	3
6	4

<result>

R.r_{id}	R.a	S.s_{id}	S.a
–			
–			
3	2	1	1
3	2	2	1
4	3	1	1
4	3	2	1
4	3	3	2
4	3	4	2

An Illustration of Standard Repartition Theta-Join Algorithm



SELECT * FROM R, S WHERE R.a > S.a

Input

R

r_{id}	a
1	1
2	1
3	2
4	3

S

s_{id}	a
1	1
2	1
3	2
4	2
5	3
6	4

Generate keys < 2

Map

key	value
	–
	–
1	("R", 3, 2)
1	("R", 4, 3)
2	("R", 4, 3)
1	("S", 1, 1)
1	("S", 2, 1)
2	("S", 3, 2)
2	("S", 4, 2)
3	("S", 5, 3)
4	("S", 6, 4)

Shuffle

key	value list
1	[("R", 3, 2), ("R", 4, 3), ("S", 1, 1), ("S", 2, 1)]
2	[("R", 4, 3), ("S", 3, 2), ("S", 4, 2)]
3	("S", 5, 3)
4	("S", 6, 4)

Reduce

3	2	1	1
3	2	2	1
4	3	1	1
4	3	2	1
4	3	3	2
4	3	4	2
–			
–			

S

R

	s_1	s_2	s_3	s_4	s_5	s_6
r_1						
r_2		1				
r_3	O	O	2			
r_4	O	O	O	O		

Analysis of Join Algorithms with MapReduce

- [Okcan, Riedewald: SIGMOD11]
- Execution times of map and reduce functions increase monotonically with their input and output sizes
- Job complete time depends on the slowest map and reduce functions
- Balancing the workloads of map functions is easy and thus we ignore map functions
- Balance the workloads of reduce functions as evenly as possible

Join-Matrix M of R and S

- M(i,j) = true, if r_i and s_j satisfy the join predicate
  = false, otherwise

R

	a
r_1	5
r_2	7
r_3	7
r_4	8
r_5	9
r_6	9

S

	a
s_1	5
s_2	7
s_3	7
s_4	7
s_5	8
s_6	9

S.a

R.a	5	7	7	7	8	9
5	O					
7		O	O	O		
7		O	O	O		
8					O	
9						O
9						O

R.a = S.a

S.a

R.a	5	7	7	7	8	9
5	O					
7		O	O	O	O	
7		O	O	O	O	
8		O	O	O	O	O
9					O	O
9					O	O

|R.a − S.a| < 2

S.a

R.a	5	7	7	7	8	9
5	O					
7	O	O	O	O		
7	O	O	O	O		
8	O	O	O	O	O	
9	O	O	O	O	O	O
9	O	O	O	O	O	O

R.a ≥ S.a

Reduce Allocations for Standard Repartition Equi-joins

- All records with the same join key goes to the same reduce function
- Assume 3 reduce functions are used

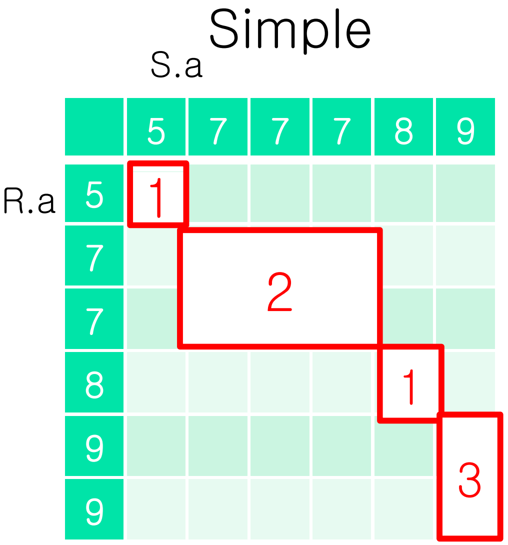
R

	a
r_1	5
r_2	7
r_3	7
r_4	8
r_5	9
r_6	9

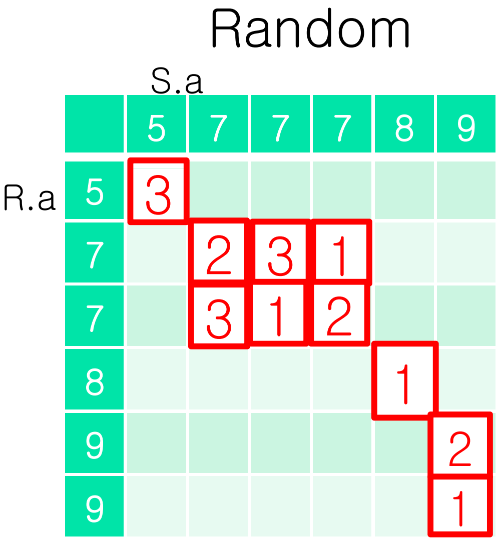
S

	a
s_1	5
s_2	7
s_3	7
s_4	7
s_5	8
s_6	9

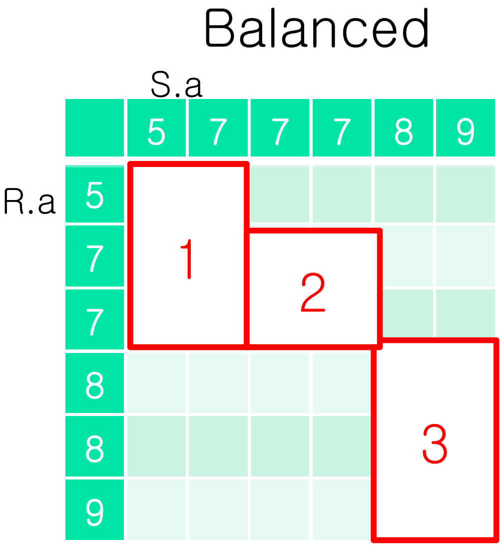
R.a = S.a



Max reduce input size = 5
Max reduce output size = 6



Max reduce input size = 8
Max reduce output size = 4



Max reduce input size = 5
Max reduce output size = 4

Comparisons of Reduce Allocation Methods

- Simple allocation
  - Minimize the maximum input size of reduce functions
  - Output size may be skewed
- Random allocation
  - Minimize the maximum output size of reduce functions
  - Input size may be increased due to duplication
- Balanced allocation
  - Minimize both maximum input and output sizes

How to Balance Reduce Allocation

- [Okcan, Riedewald: SIGMOD 2011]
- Assume r is desired number of reduce functions
- Partition join-matrix M into r regions
- A map function sends each record in R and S to mapped regions
- A reduce function outputs all possible (r,s) pairs satisfying the join predicates in its value-list
- Propose M-Bucket-I algorithm

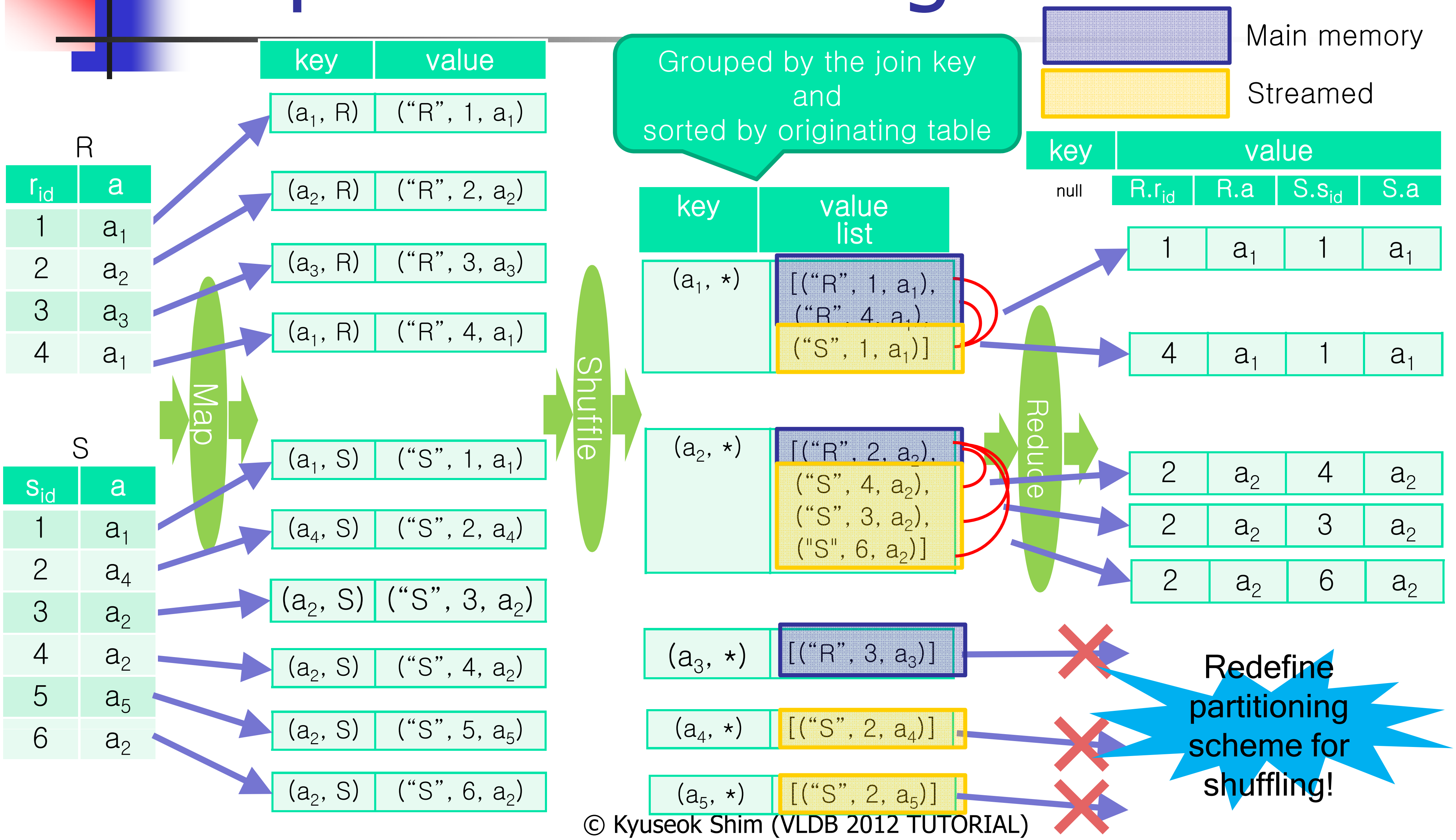
Other Join Algorithms with MapReduce

- [Blanas, Patel, Ercegovac, Rao, Shekita, Tian: SIGMOD 2010]
  - Assume |R| << |S|
- Repartition join algorithm
  - Improved repartition
  - Repartition with pre-partitioning
- Broadcast join algorithm
- Semi-join algorithms
  - Semi-join
  - Per-split semi-join

Improved Repartition Join

- In reduce functions
  - Only records from R are kept in main memory
  - Records from S are streamed to generate the join output
- In shuffling phase, redefine partitioning scheme by changing Partitioner and Comparator classes so that
  - Sorting is done with (join attribute value, relation id) in the keys output by map functions
  - Key-value pairs are assigned to reduce functions by join attribute value in the keys

An Illustration of Improved Repartition Join Algorithm



🟦 Main memory

🟨 Streamed

R

r_{id}	a
1	a_1
2	a_2
3	a_3
4	a_1

S

s_{id}	a
1	a_1
2	a_4
3	a_2
4	a_2
5	a_5
6	a_2

key	value
(a_1, R)	("R", 1, a_1)
(a_2, R)	("R", 2, a_2)
(a_3, R)	("R", 3, a_3)
(a_1, R)	("R", 4, a_1)
(a_1, S)	("S", 1, a_1)
(a_4, S)	("S", 2, a_4)
(a_2, S)	("S", 3, a_2)
(a_2, S)	("S", 4, a_2)
(a_2, S)	("S", 5, a_5)
(a_2, S)	("S", 6, a_2)

Grouped by the join key
and
sorted by originating table

key	value list
(a_1, *)	[("R", 1, a_1), ("R", 4, a_1), ("S", 1, a_1)]
(a_2, *)	[("R", 2, a_2), ("S", 4, a_2), ("S", 3, a_2), ("S", 6, a_2)]
(a_3, *)	[("R", 3, a_3)]
(a_4, *)	[("S", 2, a_4)]
(a_5, *)	[("S", 2, a_5)]

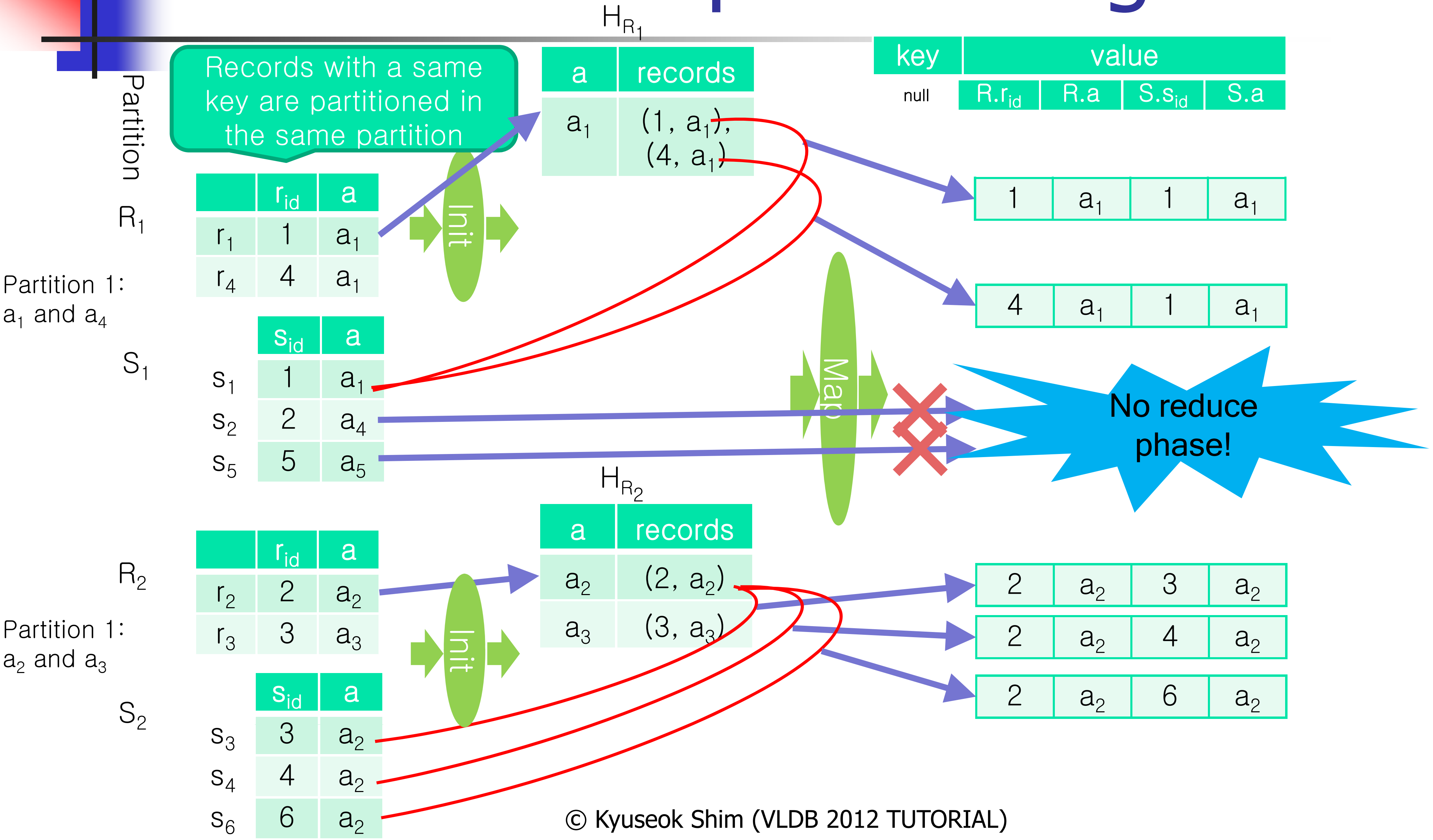
key	value			
null	R.r_{id}	R.a	S.s_{id}	S.a
	1	a_1	1	a_1
	4	a_1	1	a_1
	2	a_2	4	a_2
	2	a_2	3	a_2
	2	a_2	6	a_2

Redefine partitioning scheme for shuffling!

Repartition Join with Pre-partitioning

- Do not use reduce functions
- To decrease the shuffling overhead in the repartition join
  - Split both S and R into partitions, S_is and R_is, in DFS based on the join attribute values before the join operation
  - The size of R_i is decided to be put in main memory of a map function
- Before the map functions are called with records in S_i, build a hash table in main-memory using R_i in DFS
- The map functions emit the pairs of the input record (from S) and records in hash table (from R)

An Illustration of Repartition Join with Pre-partitioning



Records with a same key are partitioned in the same partition

Partition

Partition 1: a_1 and a_4

R_1

	r_{id}	a
r_1	1	a_1
r_4	4	a_1

S_1

	s_{id}	a
s_1	1	a_1
s_2	2	a_4
s_5	5	a_5

Partition 1: a_2 and a_3

R_2

	r_{id}	a
r_2	2	a_2
r_3	3	a_3

S_2

	s_{id}	a
s_3	3	a_2
s_4	4	a_2
s_6	6	a_2

H_{R_1}

a	records
a_1	(1, a_1), (4, a_1)

H_{R_2}

a	records
a_2	(2, a_2)
a_3	(3, a_3)

key	value			
null	$R.r_{id}$	R.a	$S.s_{id}$	S.a

1	a_1	1	a_1
4	a_1	1	a_1

2	a_2	3	a_2
2	a_2	4	a_2
2	a_2	6	a_2

No reduce phase!

Broadcast Join

- To avoid the network overhead for moving the larger table S, broadcast the smaller table R
- Chunks S_i of table S are not transferred over the network
- Init function
  - R is split into partitions R_is based on the join attribute value in the local file system
  - If $|R| < |S_i|$, build the hash table H_R
- Map function
  - A map function is invoked with each record s in S
  - If the hash table H_R exits,
    - Emit all <r, s> pairs where r.a=s.a for $r \in H_R$
  - else
    - Add s to the hash table H_{S_i}
- Close function
  - If H_R not exist, perform the join between R and H_{S_i}

Semi-Join Algorithms

- Semi-Join
  - Avoid to send the records in R over the network which do not join with the records in S
  - Phase 1: Extract distinct join attribute values in S
  - Phase 2: Generate filtered R′ using distinct join attribute values in S
  - Phase 3: Join the filtered R′ and S
- Per-Split Semi-Join
  - Builds filtered R'_i corresponding to the chunk S_i
  - Each record in R'_i will join with at least one record in S_i
  - Phase 3 becomes cheaper
  - Access just filtered $R_{i'}$ for S_i over the network instead of accessing whole filtered R

Performing Multi-way Joins using MapReduce

- Performing Multi-way Joins in one MapReduce phase
    - [Afrati, Ullman: EDBT 2010]
- Optimization of MapReduce jobs from Hive
    - [Wu, Li, Mehrotra, Ooi: SOCC, 2011]

Similarity Self-Join Algorithms using MapReduce

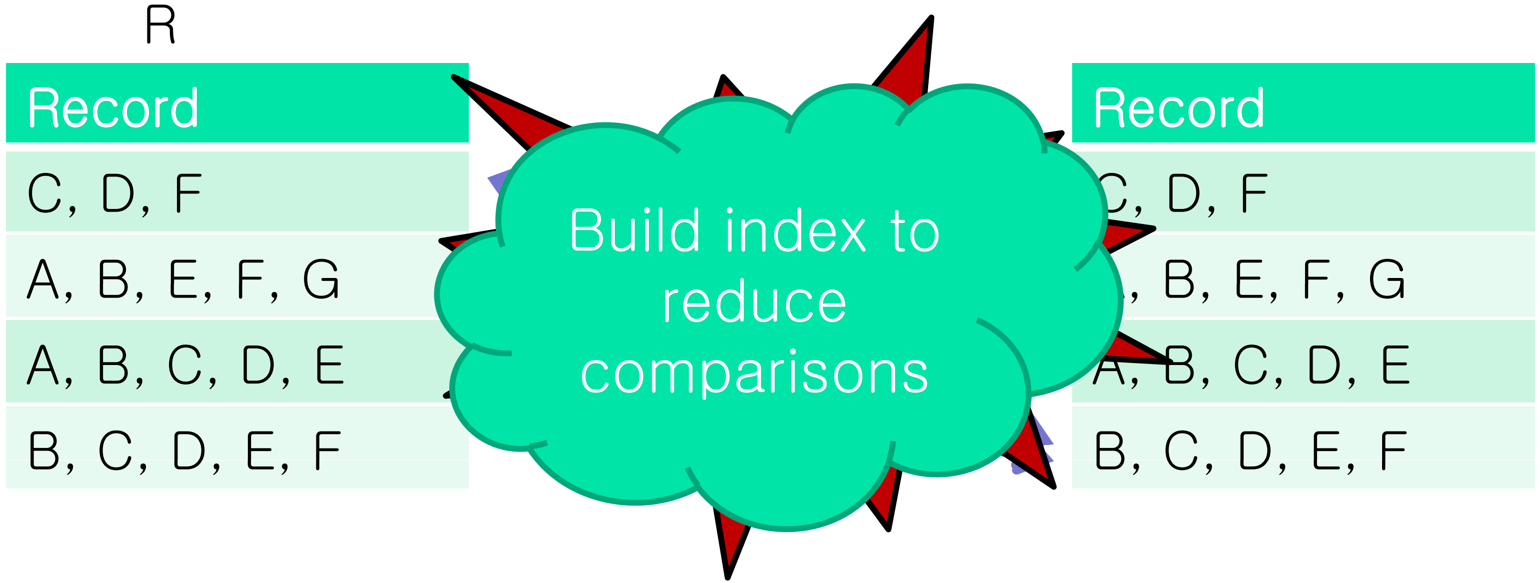
Problem Formulation

- Given
  - A set of records R
  - A similarity function, sim
    - Jaccard(x,y) = $|x \cap y| / |x \cup y|$
    - Cosine(x,y) = $(x \cdot y)/(||x|| \cdot ||y||)$
    - Euclidian(x,y) = $(\Sigma_i (x[i]-y[i])^2)^{1/2}$
  - A minimum similarity threshold σ
- Find all pairs of records (x,y) in R, such that $sim(x,y) \geq \sigma$

Serial Set Similarity Self-Join Algorithms

A Traditional Brute-force Algorithm

- Enumerate every pair of records and compute their similarities
- Expensive for large datasets
    - $O(|R|^2)$ similarity computations



R

Record
C, D, F
A, B, E, F, G
A, B, C, D, E
B, C, D, E, F

Build index to reduce comparisons

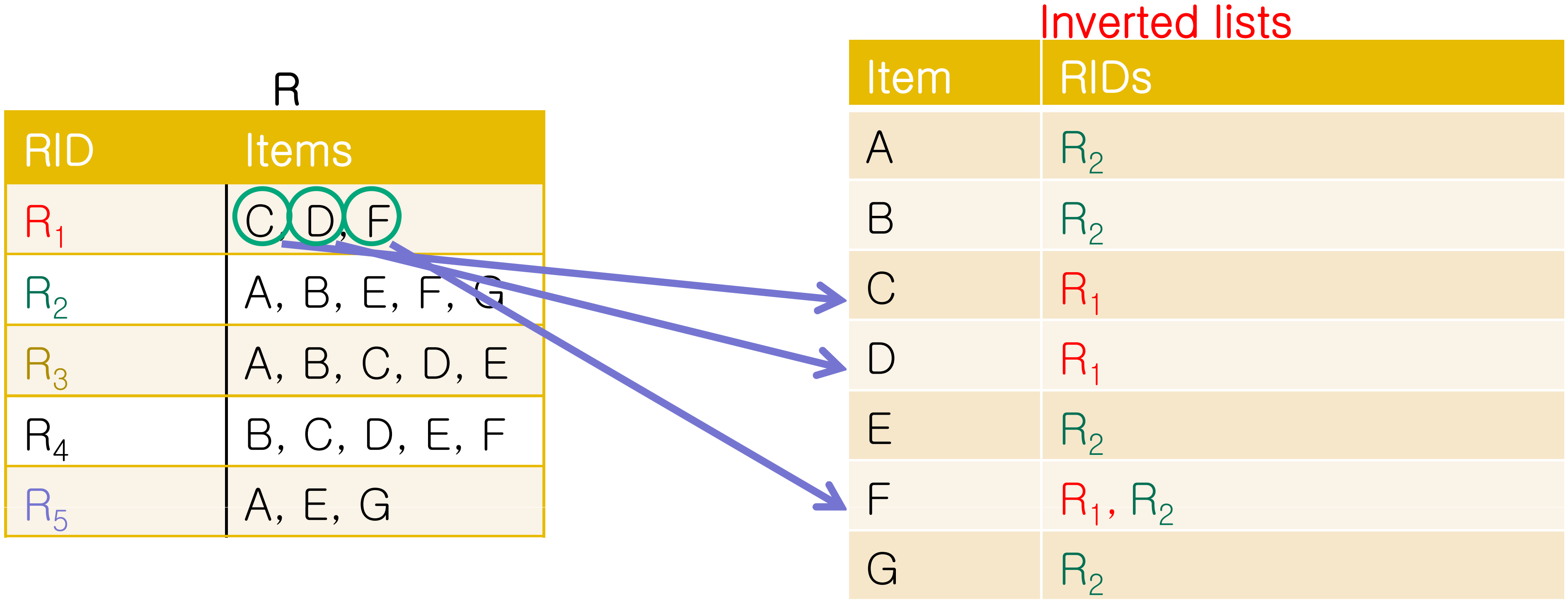
Record
C, D, F
, B, E, F, G
A, B, C, D, E
B, C, D, E, F

Similarity Self-Joins using Inverted Lists

- Make an inverted lists for all items in set data
- Generate candidates by considering every pair of record IDs in the each inverted list
- Find similar pairs by verifying each candidate
  - Relationship between Jaccard and Overlap similarity measures
    - Jaccard(x, y) ≥ σ ⇔
      Overlap(x, y) ≥ σ /(1+σ) · (|x| + |y|) = α
    - We call α the overlap threshold
  - Check overlap(x,y) ≥ α instead of Jaccard(x,y) ≥ σ

Building Inverted Lists

- While scan each record in the data
  - Insert the identifier of the record (RID) into the inverted list entries of its items



R

RID	Items
R_1	C, D, F
R_2	A, B, E, F, G
R_3	A, B, C, D, E
R_4	B, C, D, E, F
R_5	A, E, G

Inverted lists

Item	RIDs
A	R_2
B	R_2
C	R_1
D	R_1
E	R_2
F	R_1, R_2
G	R_2

Building Inverted Lists

- While scan each record in the data
    - Insert the identifier of the record (RID) into the inverted list entries of its items

R

RID	Items
R_1	C, D, F
R_2	A, B, E, F, G
R_3	A, B, C, D, E
R_4	B, C, D, E, F
R_5	A, E, G

inverted lists

Item	RIDs
A	R_2, R_3, R_5
B	R_2, R_3, R_4
C	R_1, R_3, R_4
D	R_1, R_3, R_4
E	R_2, R_3, R_4, R_5
F	R_1, R_2, R_4
G	R_2, R_5

Generating Candidates

- Generate candidates by making every RID pair in the each inverted list entry
  - Increase the overlap of the candidate pair

Inverted lists

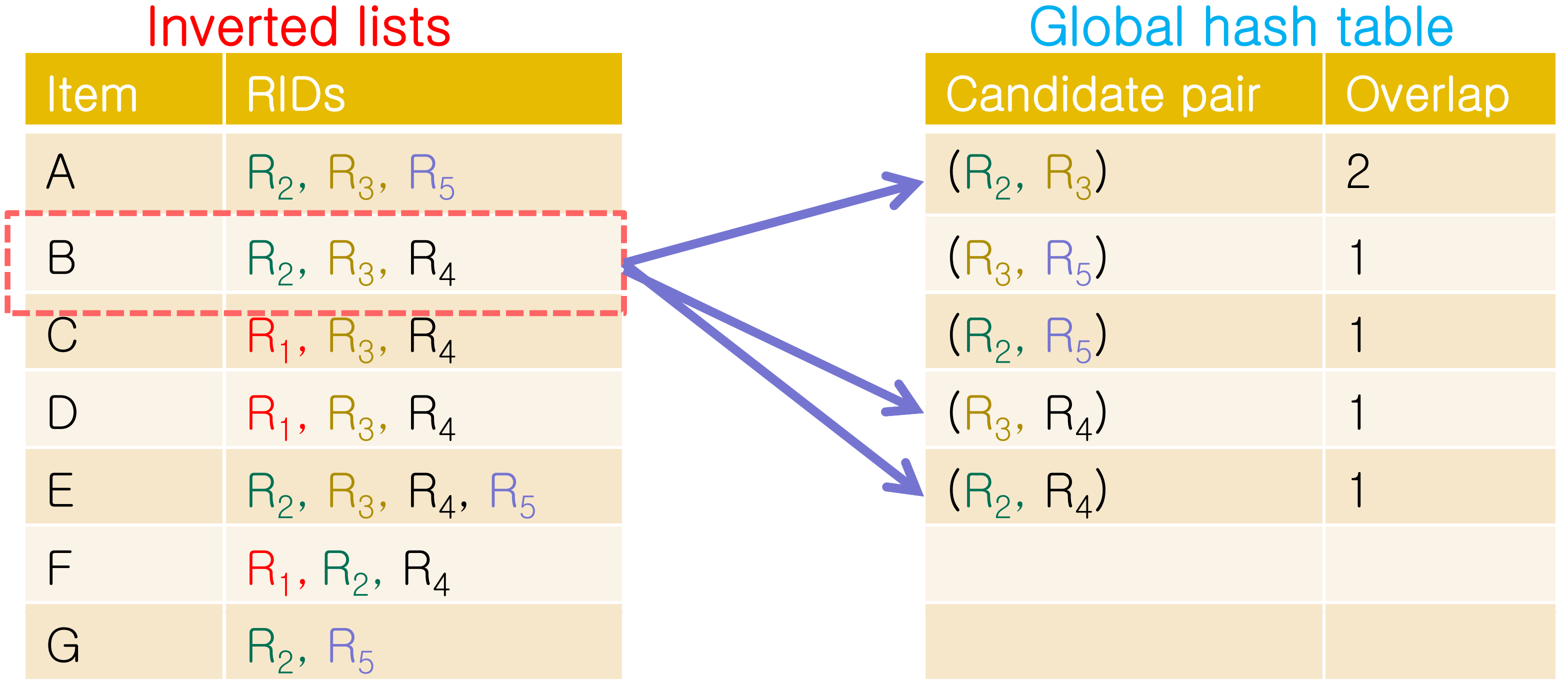
Item	RIDs
A	R_2, R_3, R_5
B	R_2, R_3, R_4
C	R_1, R_3, R_4
D	R_1, R_3, R_4
E	R_2, R_3, R_4, R_5
F	R_1, R_2, R_4
G	R_2, R_5

Global hash table

Candidate pair	Overlap
(R_2, R_3)	1
(R_3, R_5)	1
(R_2, R_5)	1

Generating Candidates

- Generate candidates by making every RID pair in the each inverted list entry
  - Increase the overlap of the candidate pair



Inverted lists

Item	RIDs
A	R_2, R_3, R_5
B	R_2, R_3, R_4
C	R_1, R_3, R_4
D	R_1, R_3, R_4
E	R_2, R_3, R_4, R_5
F	R_1, R_2, R_4
G	R_2, R_5

Global hash table

Candidate pair	Overlap
(R_2, R_3)	2
(R_3, R_5)	1
(R_2, R_5)	1
(R_3, R_4)	1
(R_2, R_4)	1

Generating Candidates

- Generate candidates by making every RID pair in the each inverted list entry
  - Increase the overlap of the candidate pair

Inverted lists

Item	RIDs
A	R_2, R_3, R_5
B	R_2, R_3, R_4
C	R_1, R_3, R_4
D	R_1, R_3, R_4
E	R_2, R_3, R_4, R_5
F	R_1, R_2, R_4
G	R_2, R_5

Global hash table

Candidate pair	Overlap
(R_2, R_3)	3
(R_3, R_5)	2
(R_2, R_5)	3
(R_3, R_4)	4
(R_2, R_4)	3
(R_1, R_3)	2
(R_1, R_4)	3
(R_4, R_5)	1
(R_1, R_2)	1

Finding Similar Pairs

Jaccard coefficient threshold $\sigma = 0.6$

Recall Jaccard(x, y) $\geq \sigma \Leftrightarrow$ Overlap(x, y) $\geq \alpha = \sigma/(1+\sigma)(|x| + |y|)$

Substitute σ values

We need the size of each record

Global hash table

Candidate pair	Overlap	Overlap threshold α
(R_2, R_3)	3	3.75
(R_3, R_5)	2	
(R_2, R_5)	3	
(R_3, R_4)	4	
(R_2, R_4)	3	
(R_1, R_3)	2	
(R_1, R_4)	3	
(R_4, R_5)	1	
(R_1, R_2)	1	

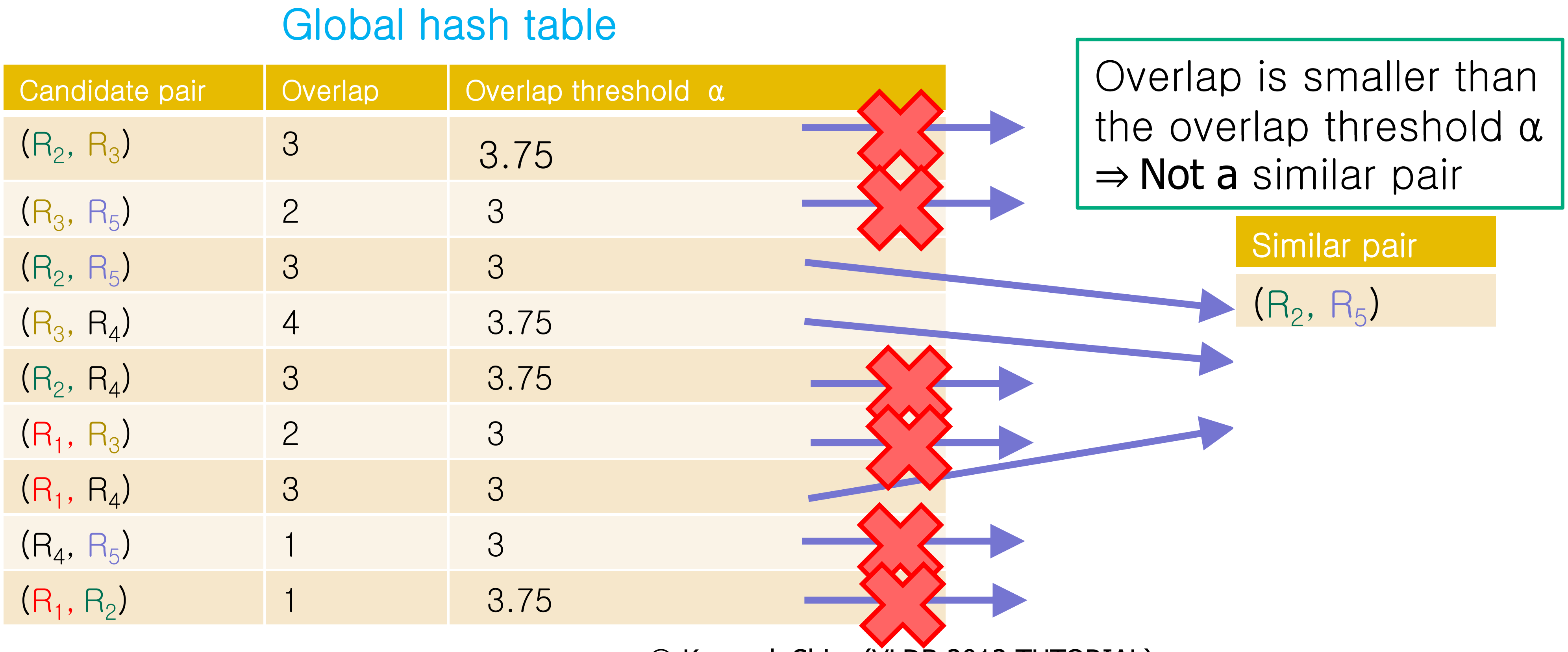
RID	Size
R_1	3
R_2	5
R_3	5
R_4	5
R_5	3

Calculate each record size

Verifying Candidates

Jaccard coefficient threshold $\sigma = 0.6$

Recall Jaccard(x, y) $\geq \sigma \Leftrightarrow$ Overlap(x, y) $\geq \alpha = \sigma/(1+\sigma)(|x| + |y|)$



Global hash table

Candidate pair	Overlap	Overlap threshold α
(R_2, R_3)	3	3.75
(R_3, R_5)	2	3
(R_2, R_5)	3	3
(R_3, R_4)	4	3.75
(R_2, R_4)	3	3.75
(R_1, R_3)	2	3
(R_1, R_4)	3	3
(R_4, R_5)	1	3
(R_1, R_2)	1	3.75

Overlap is smaller than the overlap threshold α
$\Rightarrow$ **Not a** similar pair

Similar pair
(R_2, R_5)

Filtering Techniques

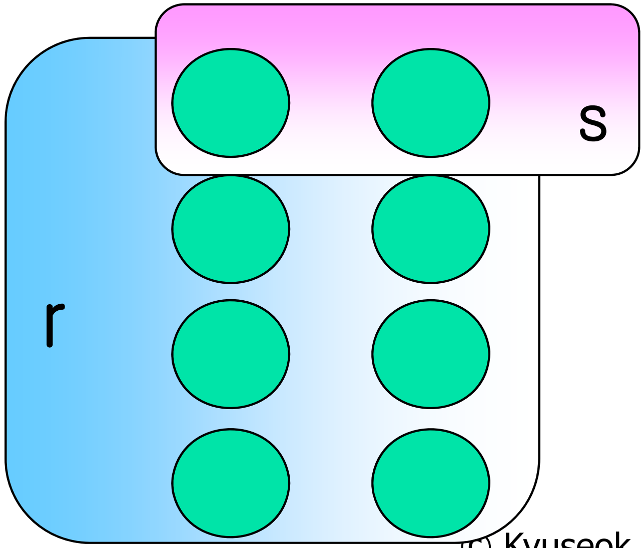
- Filtering techniques were proposed to reduce the number of candidate pairs to consider similarity
- Reduce the size of the global similarity hash table
  - Size filtering
    - [Bayardo, Ma, Srikant: WWW, 2007]
    - If |r|>>|s|, a pair(r,s) cannot be the similar pair
  - Positional filtering
    - [Xiao, Wang, Lin, Yu: WWW 2008]
    - Utilize an upper bound of similarities
- Reduce the sizes of inverted lists
  - Prefix filtering
    - [Xiao, Wang, Lin, Yu: WWW 2008]
    - Index the items in a subset of each set record only

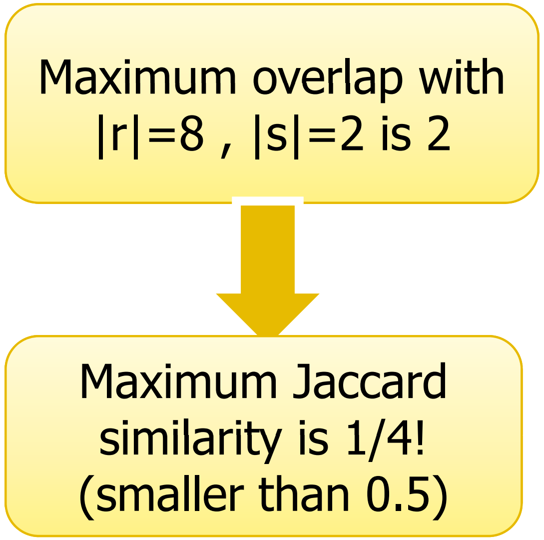
Size Filtering

- A pair of set (r,s) cannot be a similar pair if

  $$|s| < \sigma|r|$$

  where σ is the minimum Jaccard similarity threshold
- Let minimum Jaccard similarity threshold σ =0.5
- Suppose we have two sets r and s with |r|=8 and |s|=2
- Maximum possible Jaccard(r,s) is 1/4
- Jaccard(r,s) should be less than σ





Maximum overlap with |r|=8 , |s|=2 is 2

Maximum Jaccard similarity is 1/4! (smaller than 0.5)

Positional Filtering

- Given
  - A collection of records where
    - Items in each record are sorted by global item ordering $\mathcal{O}$
  - A minimum similarity threshold σ (equiva
- For two set records r and s
  - Let the pivot item w=r[i]
  - Partition r into two sets $r_{left}(w)$=r[1…(i-1)], $r_{right}(w)$=r[i…n]
  - For an item w∈r∩s,
    - If Overlap($r_{left}(w)$,$s_{left}(w)$) + min(|$r_{right}(w)$|,|$s_{right}(w)$|)<α
      - Overlap(r,s)<α
- e.g.) r={A,B,C,D,E}, s={B,C,D,E,F}, σ =0.8, α = 5, w = "B"
  - Overlap($r_{left}(B)$,$s_{left}(B)$)+min(|$r_{right}(B)$|,|$s_{right}(B)$|)

    = Overlap({A,B},{B})+min(|{C,D,E}|,|{C,D,E,F}|)

    = 1 + min(3,4) = 4 < α=5

Jaccard(r,s)≥ σ
⇔Overlap(r,s)≥α=(|r|+|s|)*σ/(1+σ)

Minimum number of unseen items

Overlap until pivot w

Prefix Filtering

$\text{Jaccard}(r,s) \geq \sigma$
$\Leftrightarrow \text{Overlap}(r,s) \geq \alpha$ where
$\alpha = \sigma/(1+\sigma)(|r|+|s|)$

- Given
  - A collection of set records where
    - Items in each record are sorted by global item ordering
  - A minimum similarity threshold σ (equivalent overlap threshold is α)
- Let the p-prefix of a record r be the first p items of r
- Insert $|r|-\lceil\sigma\cdot|r|\rceil+1$ prefix items into inverted lists instead of all items in r
  - e.g.) r={A,B,C,D,E}, σ =0.8
  - Prefix length of r: $|r|-\lceil\sigma\cdot|r|\rceil+1 = 5-\lceil 0.8*5\rceil+1=2$
  - Insert r to the inverted lists of A and B only

Prefix of r is {A,B}

Set Similarity Self-Joins using MapReduce

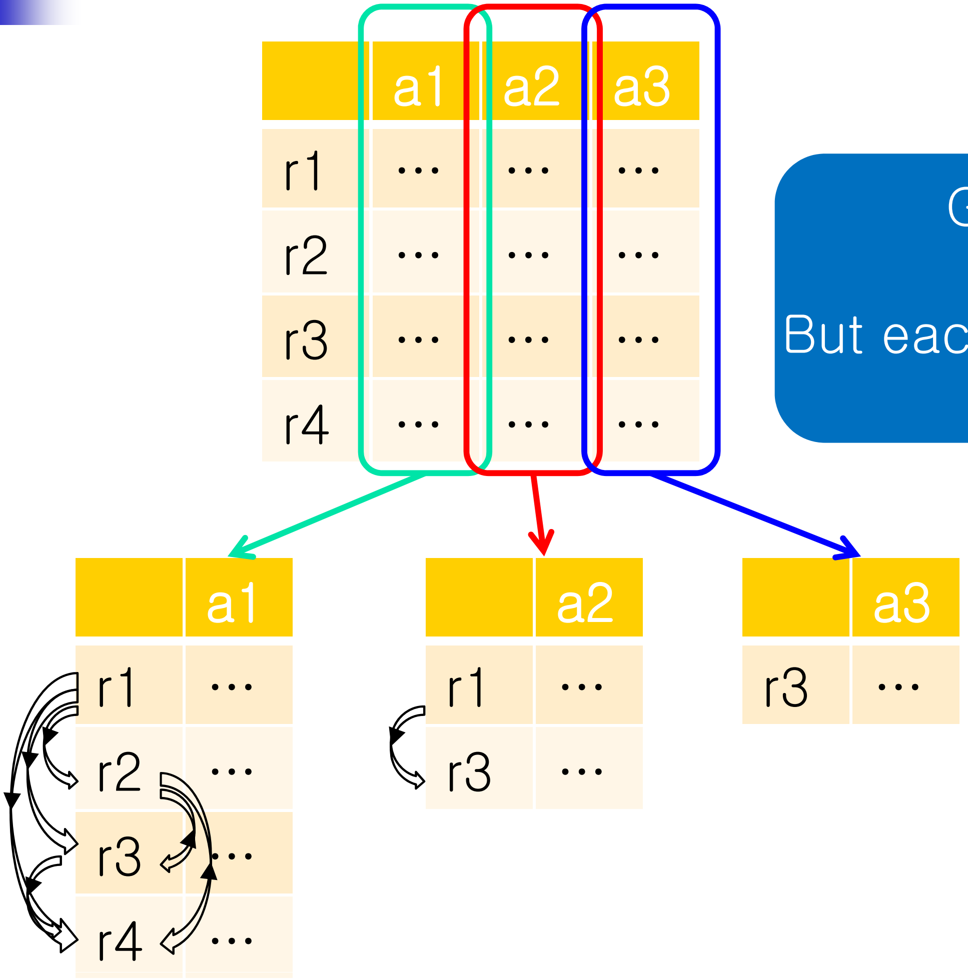
Set-Similarity Self-Join Algorithms with MapReduce

- Vernica Algorithm [Vernica, Carey, Li: SIGMOD 2010]
  - For each record, emit every item in the prefix with its entire record to the reduce functions
  - Generate and verify candidates pairs in each inverted list in parallel
  - For a pair of records, we may compute similarity value several times in different inverted lists
- V-SMART-Join [Metwally, Faloutsos: VLDB 2012]
  - Decompose similarity computations and parallelize each decomposed computation
  - Build inverted lists of all items in each record and calculate partial similarities of pairs in each inverted list
  - Compute the exact similarities of all pairs by aggregating partial similarities

Vernica Algorithm

- [Vernica, Carey, Li: SIGMOD 2010]
- The shorter an inverted list is, the less number of candidate pairs is
- Inverted lists of infrequent items are small
- Use infrequent items for prefixes
    - Order items in each set record based on frequency

Vernica Algorithm



	a1	a2	a3
r1	…	…	…
r2	…	…	…
r3	…	…	…
r4	…	…	…

	a1
r1	…
r2	…
r3	…
r4	…

	a2
r1	…
r3	…

	a3
r3	…

Generate a single inverted list
in each item.
But each record is inserted to inverted lists
of a subset of its items only!

Use inverted lists

Vernica Algorithm

- Stage 1: Find global item ordering
  - Sort the items based on frequency
  - 1-phase vs. 2-phase
- Stage 2: Produce similar record id pairs
  - Basic kernel vs. Indexed kernel
- Stage 3: Generate similar record pairs
  - Replace rid pairs with record pairs
  - 1-phase projection vs. 2-phase projection

Stage 2: Produce Similar Record ID Pairs

- Extract the prefix of each record using the global item ordering computed by Stage 1
- Extract the record ID and the join-attribute value of each record with prefix filtering
- Verify the record pairs in an inverted list using a reduce function
- Two algorithms
  - Basic kernel
    - Use individual items and apply the nested loop approach with filtering techniques
  - Indexed kernel
    - Use the grouping key technique and apply the PPJoin+[Xiao, Wang, Lin, Yu: WWW 2008]

Preprocessing: Order Items in a Record

- Sort the items in a record based on the broadcasted global item ordering

Reordered Record	
R_1	C, D, F
R_2	G, A, B, F, E
R_3	A, B, C, D, E
R_4	B, C, D, F, E
R_5	G, A, E

Reorder

Ordering
G
A
B
C
D
F
E

Preprocessing: Prefix Filtering

- Extracts the prefix items
- Prefix length = $|x|-\lceil \sigma|x| \rceil+1$

  where σ is the minimum similarity threshold

$\sigma = 0.6$

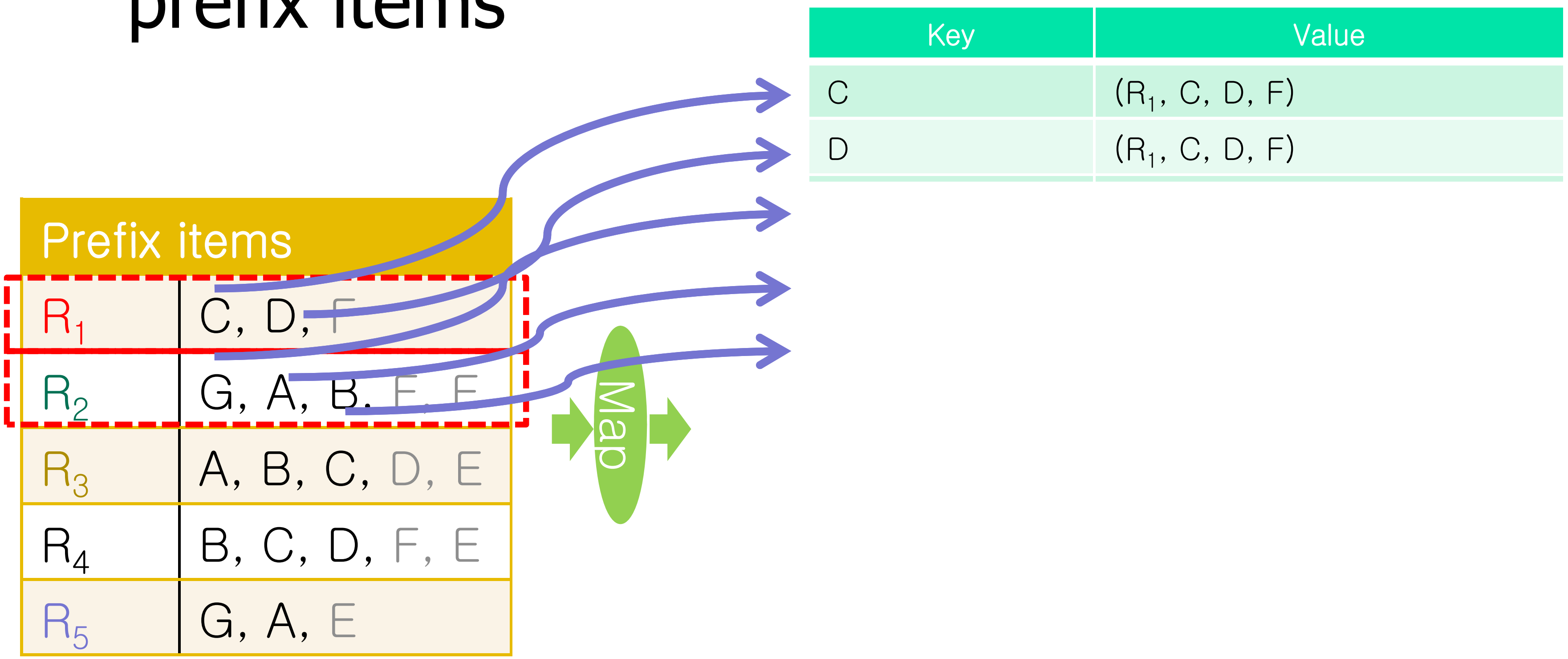
Reordered Record	
R_1	C, D, F
R_2	G, A, B, F, E
R_3	A, B, C, D, E
R_4	B, C, D, F, E
R_5	G, A, E

Id	Prefix items
R_1	C, D, F
R_2	G, A, B, F, E
R_3	A, B, C, D, E
R_4	B, C, D, F, E
R_5	G, A, E

■ : indexed element
▨ : unindexed element

Basic Kernel

- Generate a (key, value) pair for each of its prefix items



Prefix items	
R_1	C, D, F
R_2	G, A, B, F, E
R_3	A, B, C, D, E
R_4	B, C, D, F, E
R_5	G, A, E

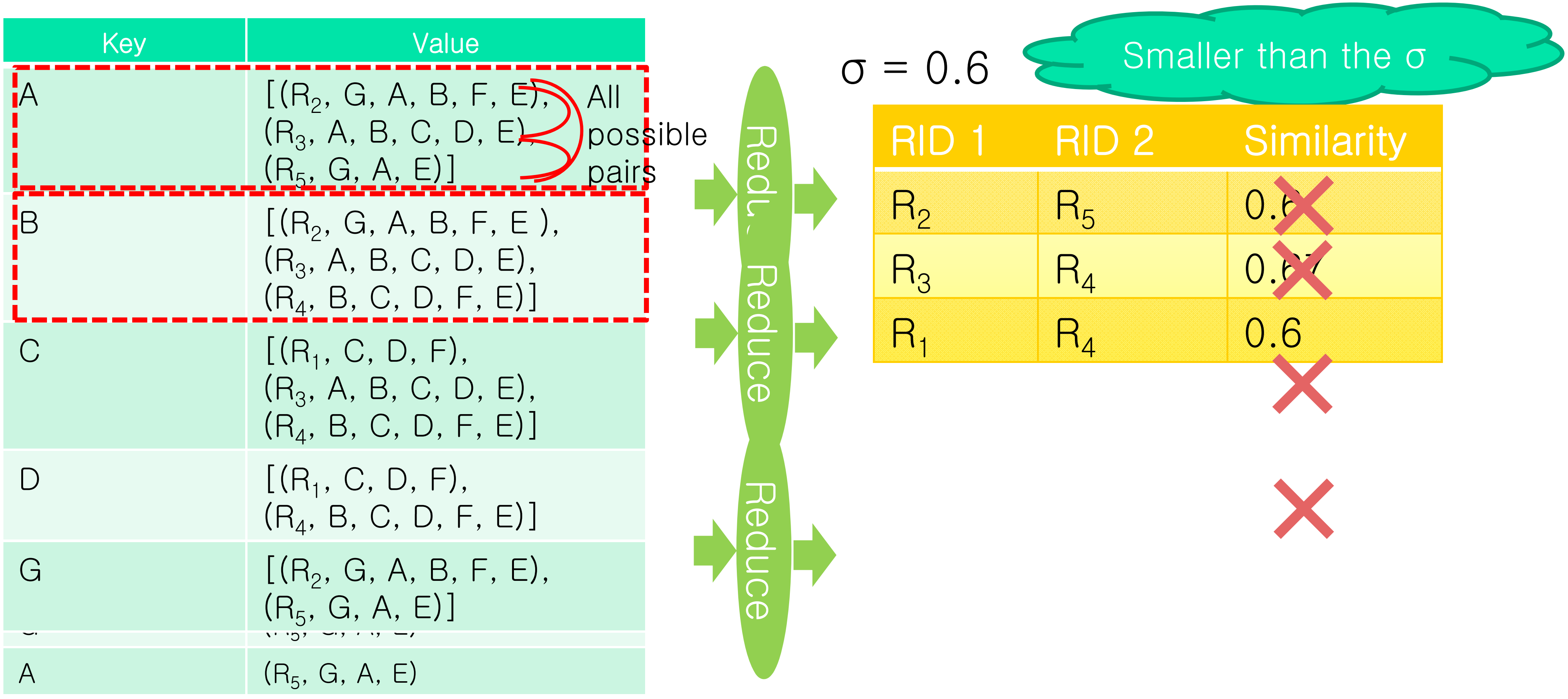
Key	Value
C	(R_1, C, D, F)
D	(R_1, C, D, F)

■ : indexed element

░ : unindexed element

Basic Kernel

- A reduce function compute the similarity for each pair



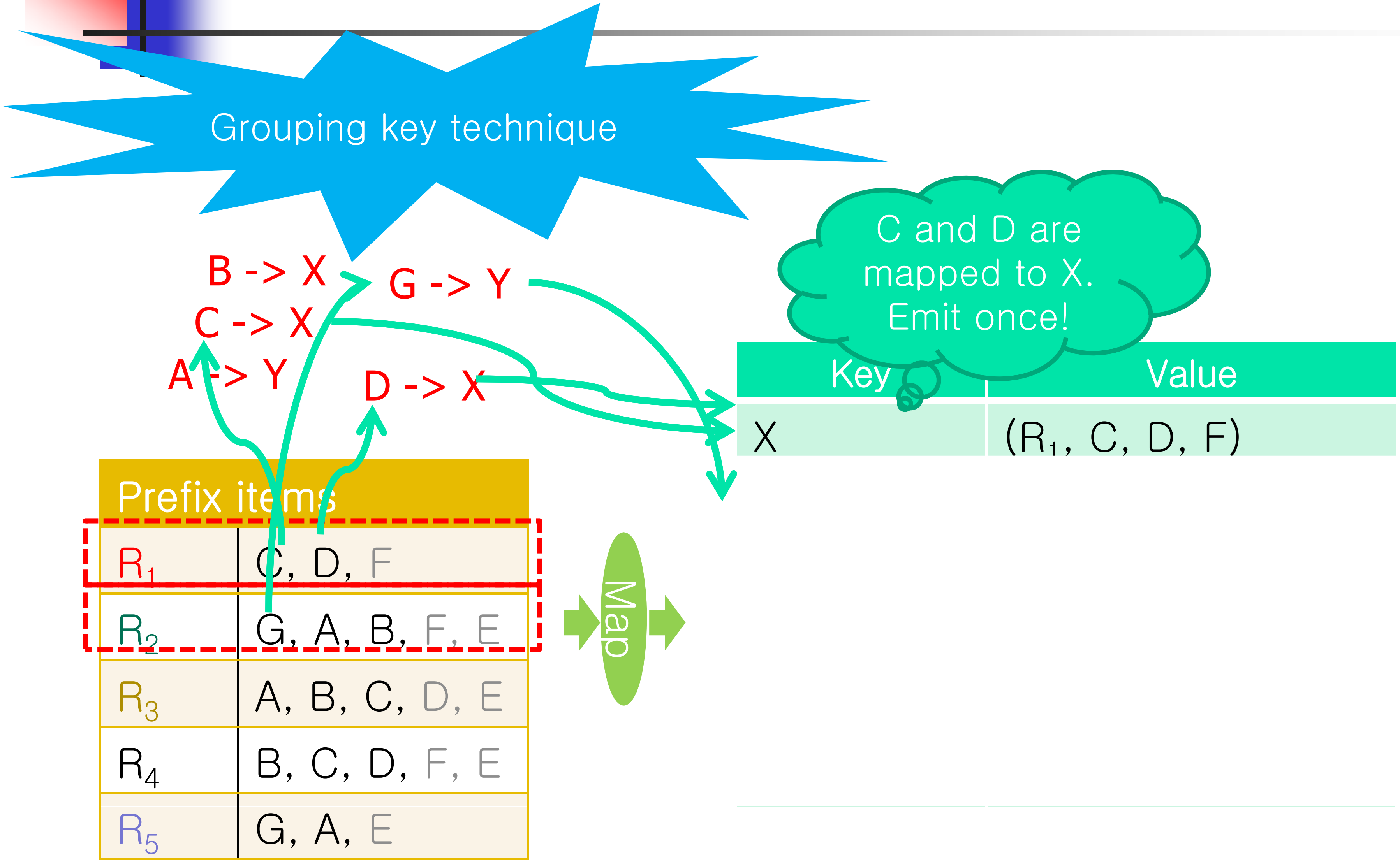
Key	Value
A	[(R_2, G, A, B, F, E), (R_3, A, B, C, D, E), (R_5, G, A, E)]
B	[(R_2, G, A, B, F, E), (R_3, A, B, C, D, E), (R_4, B, C, D, F, E)]
C	[(R_1, C, D, F), (R_3, A, B, C, D, E), (R_4, B, C, D, F, E)]
D	[(R_1, C, D, F), (R_4, B, C, D, F, E)]
G	[(R_2, G, A, B, F, E), (R_5, G, A, E)]
A	(R_5, G, A, E)

σ = 0.6

RID 1	RID 2	Similarity
R_2	R_5	0.6
R_3	R_4	0.67
R_1	R_4	0.6

Shuffling

Indexed Kernel



Grouping key technique

B -> X
C -> X
A -> Y
G -> Y
D -> X

C and D are mapped to X. Emit once!

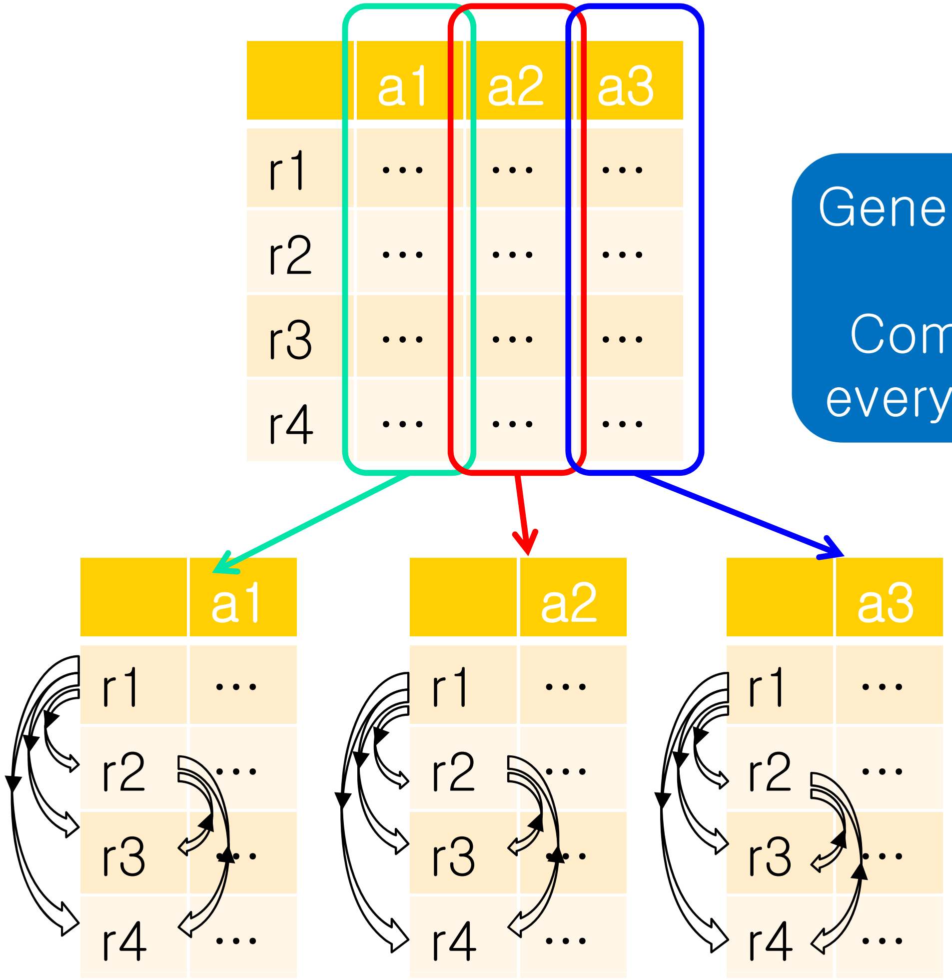
Prefix items	
R_1	C, D, F
R_2	G, A, B, F, E
R_3	A, B, C, D, E
R_4	B, C, D, F, E
R_5	G, A, E

Key	Value
X	(R_1, C, D, F)

V-SMART-Join

- [Metwally, Faloutsos: VLDB 2012]
- To reduce network overhead of Vernica algorithm, do not emit entire records
- Consider multiset and vector data
- Decompose similarity computations and parallelize each decomposed computation
- Build inverted lists of all items in each record and calculate partial similarities of pairs in each inverted list
- Compute the exact similarities of all pairs by aggregating partial similarities

V-SMART-Join

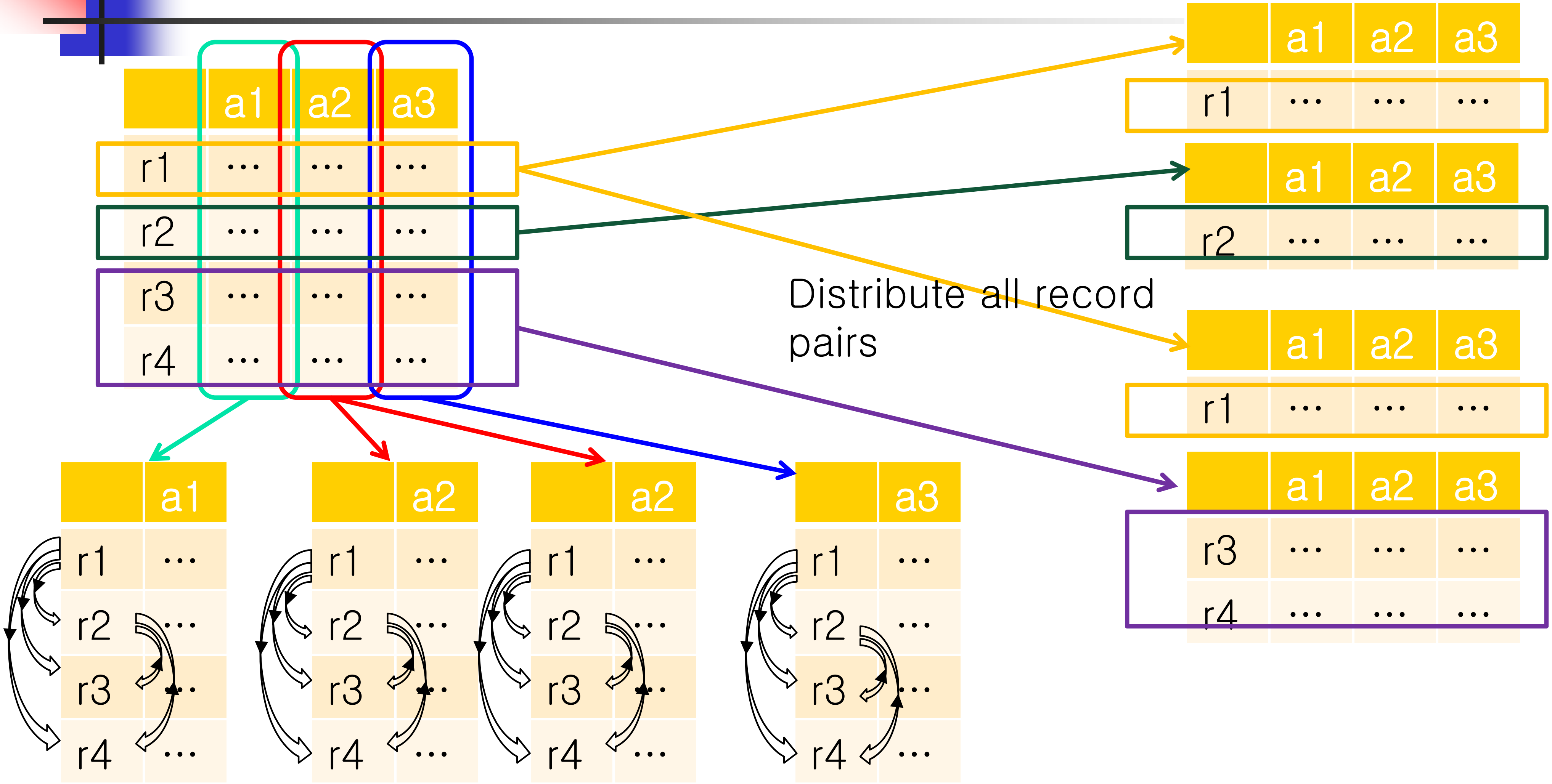


Generate a single inverted list in each dimension
Compute partial similarity of every pair in each inverted list

Use inverted lists

Vector Similarity Self-Joins using MapReduce

Classification of Similarity Self-Join Algorithms



Distribute all record pairs

Use inverted lists

Vector Similarity Self-Join Algorithms with MapReduce

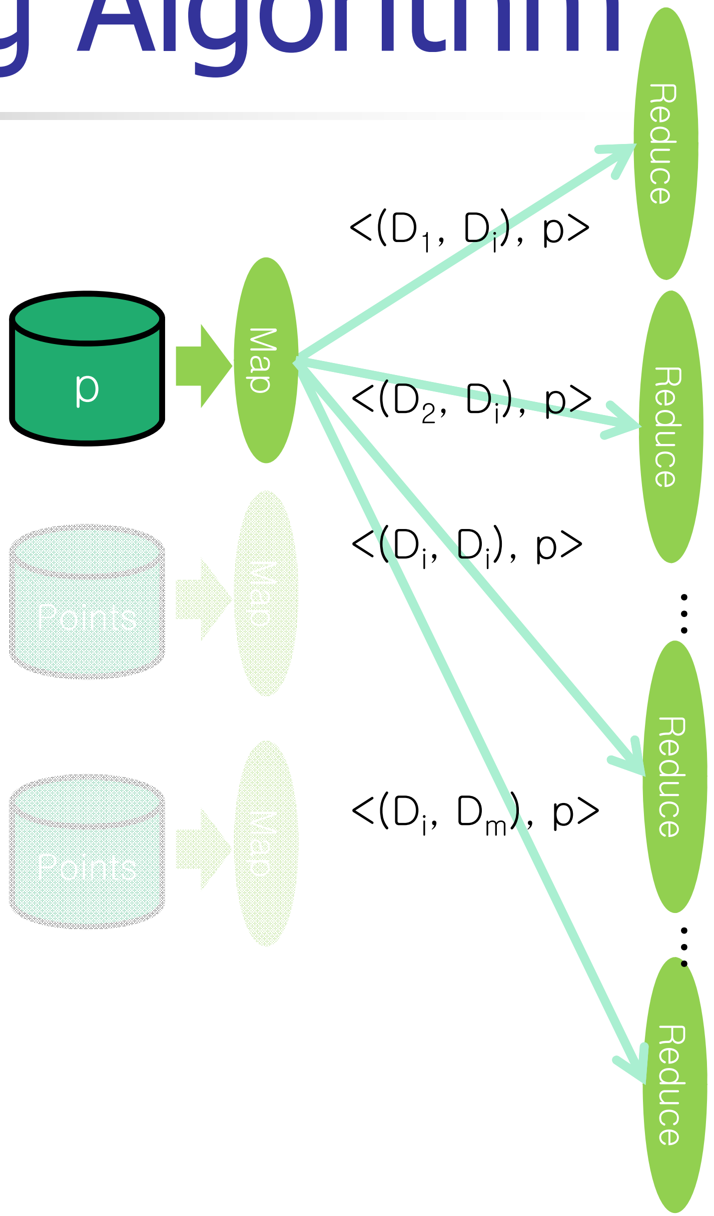
- All pair partitioning algorithm
    - Distribute all pairs of records
- Full inverted list algorithm
    - [Elsayed, Lin, Oard: HLT 2008]
    - Build inverted lists for all dimensions
- VSMART-JOIN algorithm
    - [Metwally and Faloutsos, VLDB, 2012]
    - Build inverted lists for all dimensions
    - Decompose and parallelize the similarity computations into sub-expressions
- Prefix-filtering algorithms
    - [Baraglia, Morales and Lucchese, ICDM, 2010]
    - Build inverted lists of a subset of dimensions
- Bucket-filtering algorithm for Euclidean distance
    - [Kim, Shim, ICDE: 2012]
    - Build inverted lists with a set of sub-ranges in a subset of dimensions

Cosine Similarity Measure

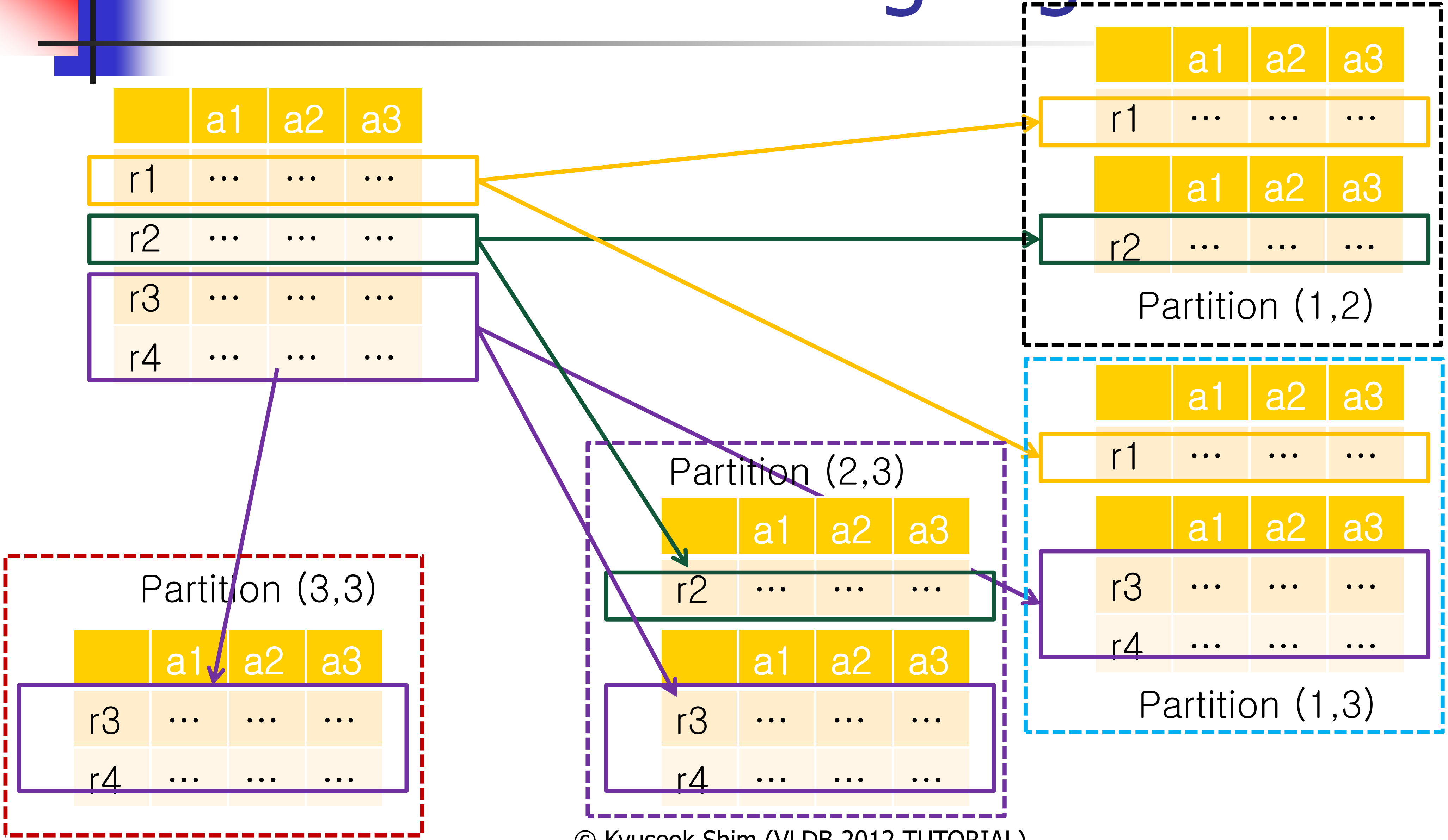
- For normalized vectors x and y, cosine similarity between x and y is the inner product of x and y
  - $sim(x,y) := cosine(x,y) = \sum_i x[i] \cdot y[i]$
    where x[i] is the value of x's i-th dimension
- Build the inverted list with non-zero values of each dimension to compute the products of non-zero values only

All Pair Partitioning Algorithm

- Simply divide and distribute the computations to find similar pairs into several reducers
- Record groups
  - $D_1, D_2, \ldots, D_m$: m distinct groups of records
- Map function
  - For each record p in the group D_i, emit key-value pairs
    - $<(1,D_i), p>,\ldots,<(D_i, D_i), p>,\ldots,<(D_i, D_m), p>$
- Reduce function
  - (D_x, D_y) : a partition to compute the similarities of all pairs of records from D_x and D_y $(x \leq y)$
    - $D_x = D_y$: self join in $D_x(=D_y)$
    - $D_x \neq D_y$: Cartesian join between D_x and D_y

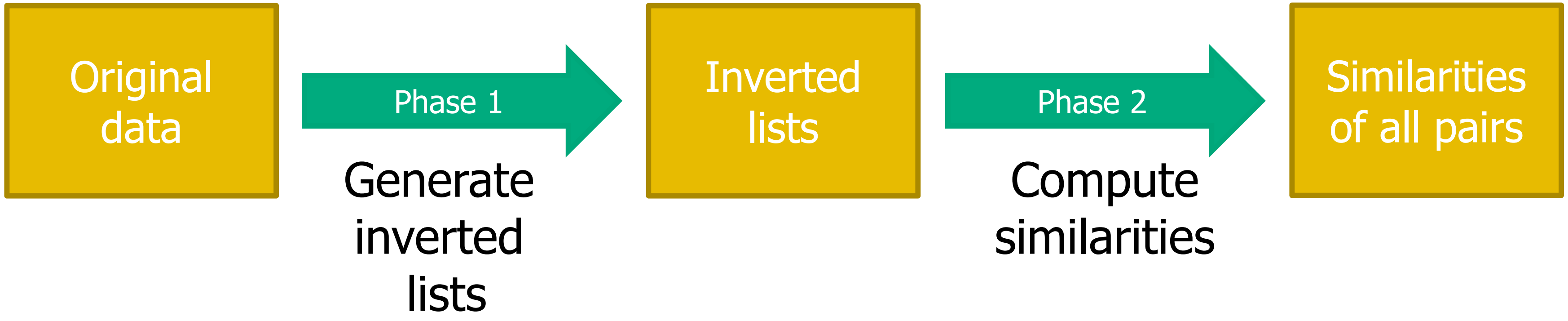


All Pair Partitioning Algorithm



Full Inverted List Algorithm

- [Elsayed, Lin, Oard: HLT 2008]
- Phase 1: Build inverted lists first
- Phase 2: Compute the similarity of every vector pair using inverted lists
  - A map function computes the similarities of all possible pairs in an inverted list for every dimension
  - A reduce function aggregates the similarity of every dimension for a pair of vectors



Original data

Phase 1

Generate inverted lists

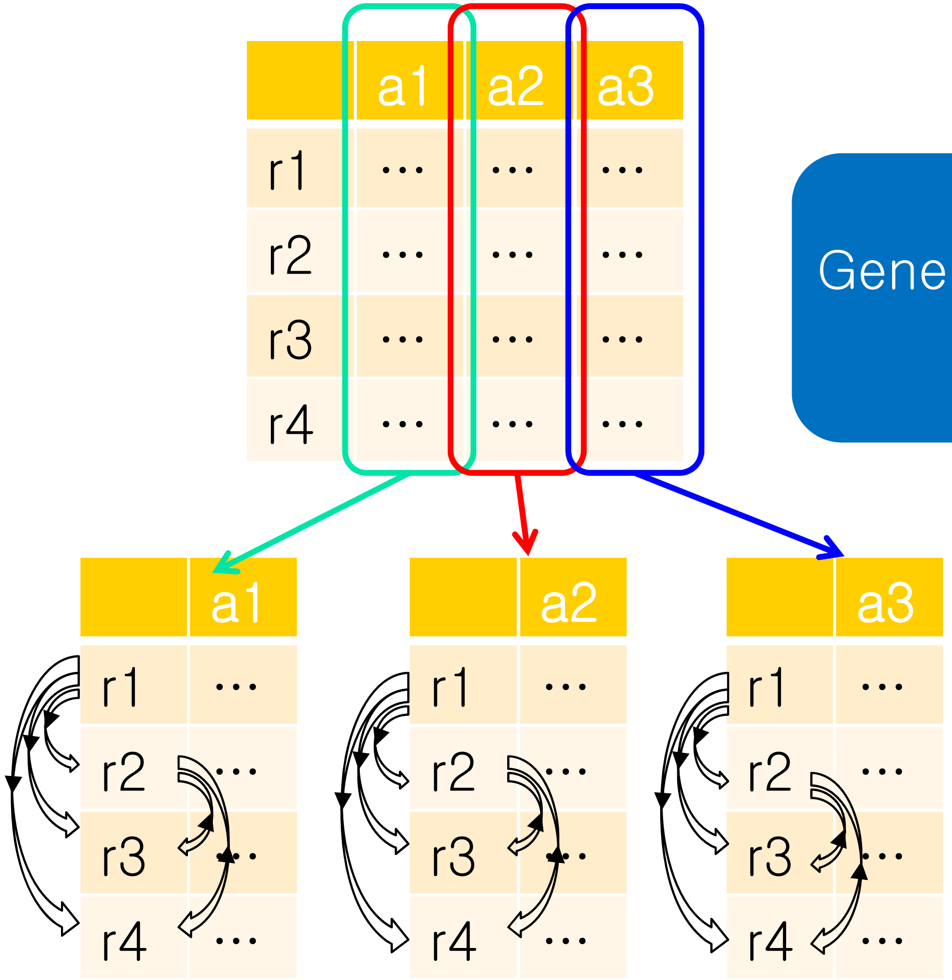
Inverted lists

Phase 2

Compute similarities

Similarities of all pairs

Full Inverted List Algorithm



	a1	a2	a3
r1	…	…	…
r2	…	…	…
r3	…	…	…
r4	…	…	…

	a1
r1	…
r2	…
r3	…
r4	…

	a2
r1	…
r2	…
r3	…
r4	…

	a3
r1	…
r2	…
r3	…
r4	…

Generate a single inverted list in each dimension

Use inverted lists

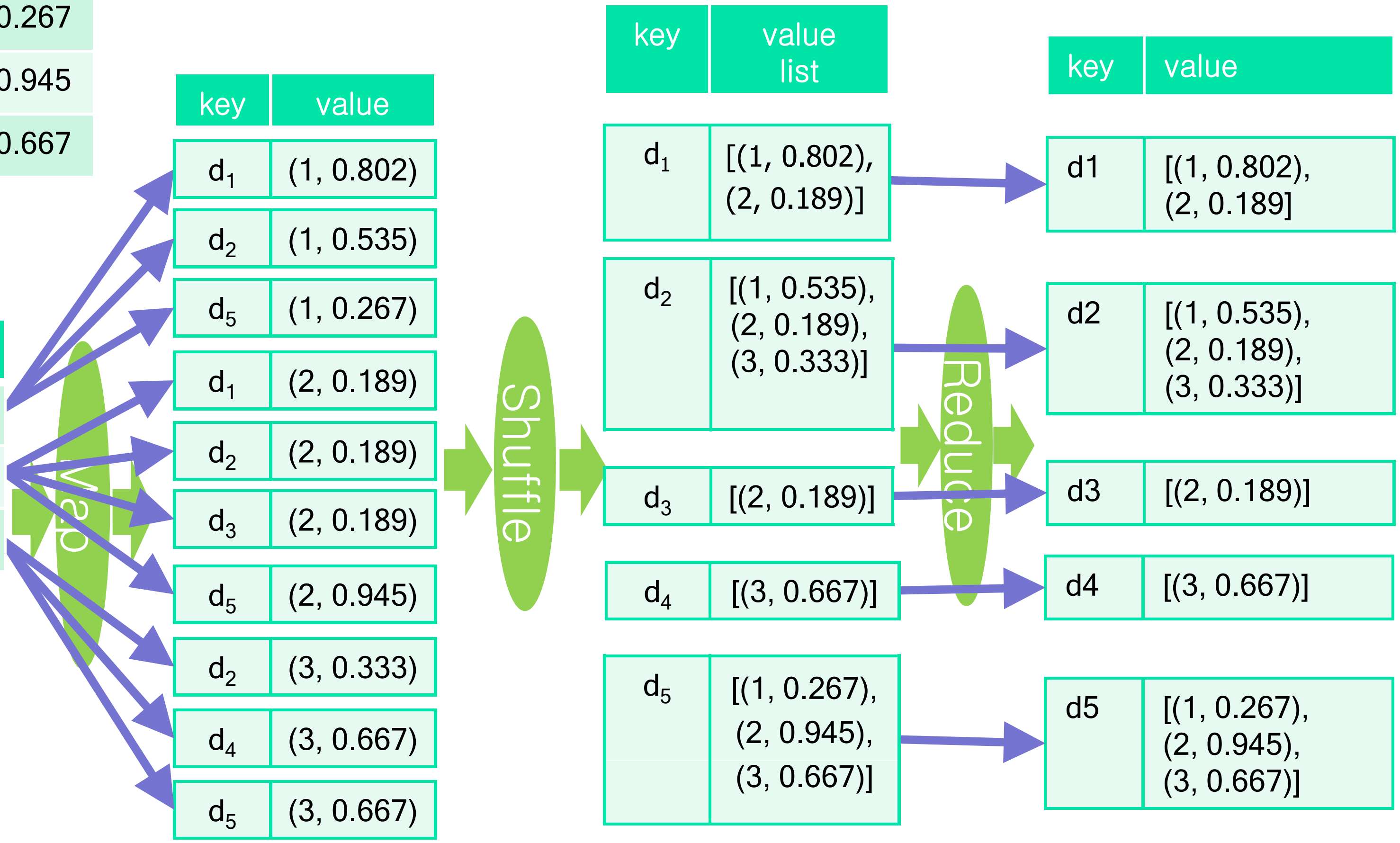
An Illustration of Full Inverted List Algorithm

- Phase 1: build inverted lists

v_{id}	d_1	d_2	d_3	d_4	d_5
1	0.802	0.535	0	0	0.267
2	0.189	0.189	0.189	0	0.945
3	0	0.333	0	0.667	0.667

Input Format

vector
1, d_1:0.802, d_2:0.535, d_5:0.267
2, d_1:0.189, d_2:0.189, d_3:0.189, d_5:0.945
3, d_2:0.333, d_4:0.667, d_5:0.667,



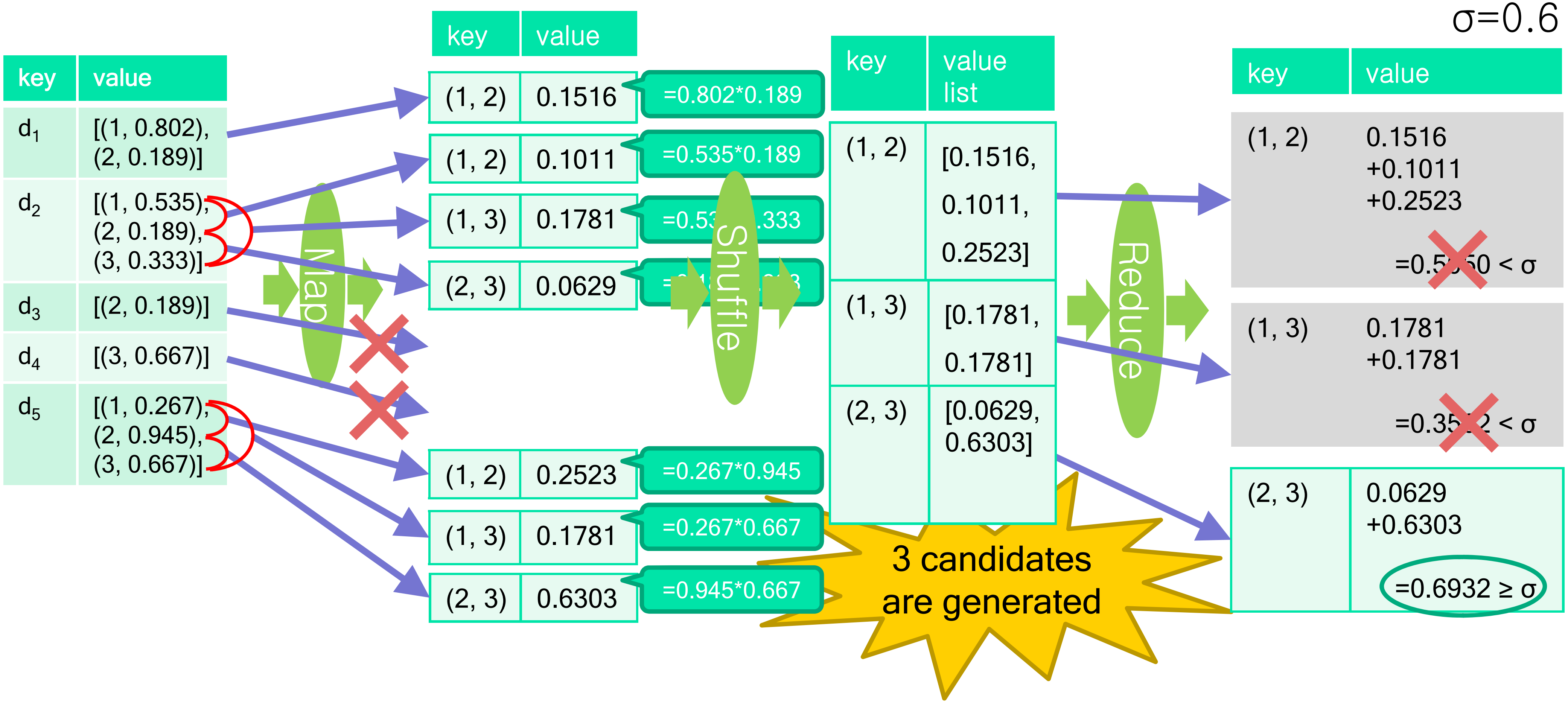
key	value
d_1	(1, 0.802)
d_2	(1, 0.535)
d_5	(1, 0.267)
d_1	(2, 0.189)
d_2	(2, 0.189)
d_3	(2, 0.189)
d_5	(2, 0.945)
d_2	(3, 0.333)
d_4	(3, 0.667)
d_5	(3, 0.667)

key	value list
d_1	[(1, 0.802), (2, 0.189)]
d_2	[(1, 0.535), (2, 0.189), (3, 0.333)]
d_3	[(2, 0.189)]
d_4	[(3, 0.667)]
d_5	[(1, 0.267), (2, 0.945), (3, 0.667)]

key	value
d1	[(1, 0.802), (2, 0.189]
d2	[(1, 0.535), (2, 0.189), (3, 0.333)]
d3	[(2, 0.189)]
d4	[(3, 0.667)]
d5	[(1, 0.267), (2, 0.945), (3, 0.667)]

An Illustration of Full Inverted List Algorithm

- Phase 2: compute similarities



σ=0.6

key	value
d_1	[(1, 0.802), (2, 0.189)]
d_2	[(1, 0.535), (2, 0.189), (3, 0.333)]
d_3	[(2, 0.189)]
d_4	[(3, 0.667)]
d_5	[(1, 0.267), (2, 0.945), (3, 0.667)]

key	value
(1, 2)	0.1516
(1, 2)	0.1011
(1, 3)	0.1781
(2, 3)	0.0629
(1, 2)	0.2523
(1, 3)	0.1781
(2, 3)	0.6303

key	value list
(1, 2)	[0.1516, 0.1011, 0.2523]
(1, 3)	[0.1781, 0.1781]
(2, 3)	[0.0629, 0.6303]

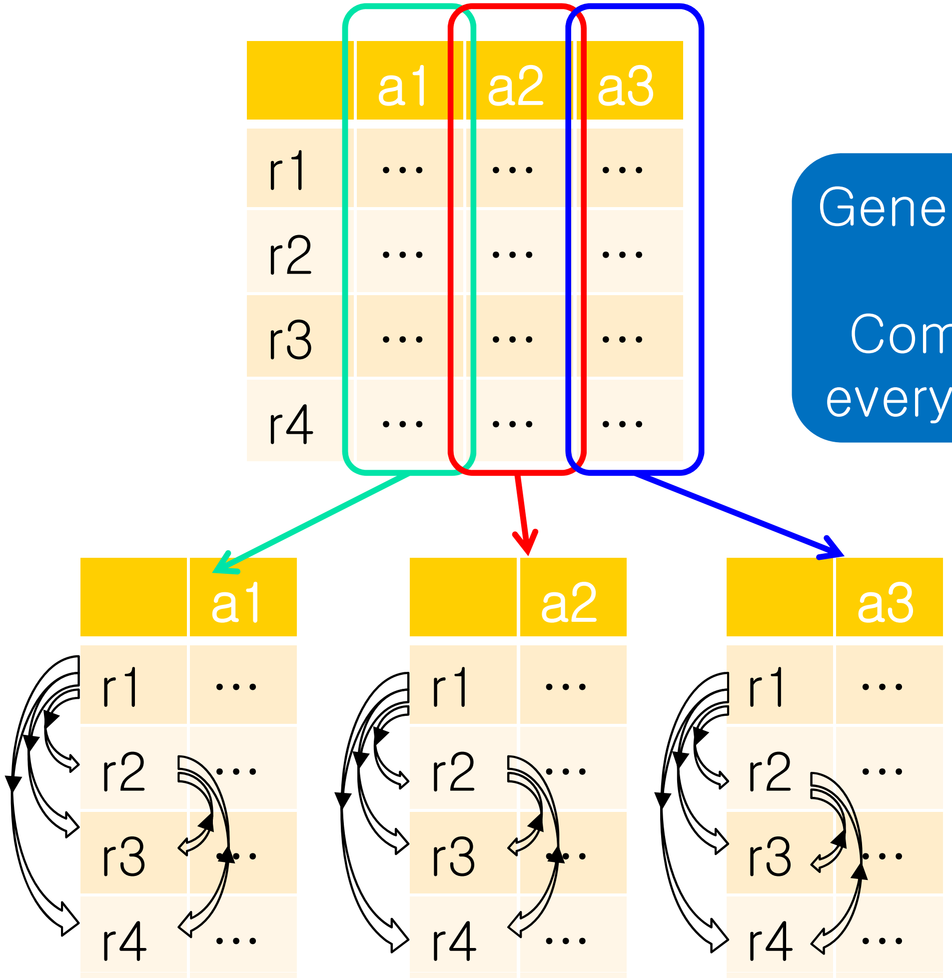
key	value
(1, 2)	0.1516 +0.1011 +0.2523 =0.5050 < σ
(1, 3)	0.1781 +0.1781 =0.3562 < σ
(2, 3)	0.0629 +0.6303 =0.6932 ≥ σ

3 candidates are generated

V-SMART-Join

- [Metwally, Faloutsos: VLDB 2012]
- Consider multiset and vector data
- Decompose similarity computations and parallelize each decomposed computation
- Build inverted lists of all items in each record and calculate partial similarities of pairs in each inverted list
- Compute the exact similarities of all pairs by aggregating partial similarities

V-SMART-Join



	a1	a2	a3
r1	…	…	…
r2	…	…	…
r3	…	…	…
r4	…	…	…

	a1
r1	…
r2	…
r3	…
r4	…

	a2
r1	…
r2	…
r3	…
r4	…

	a3
r1	…
r2	…
r3	…
r4	…

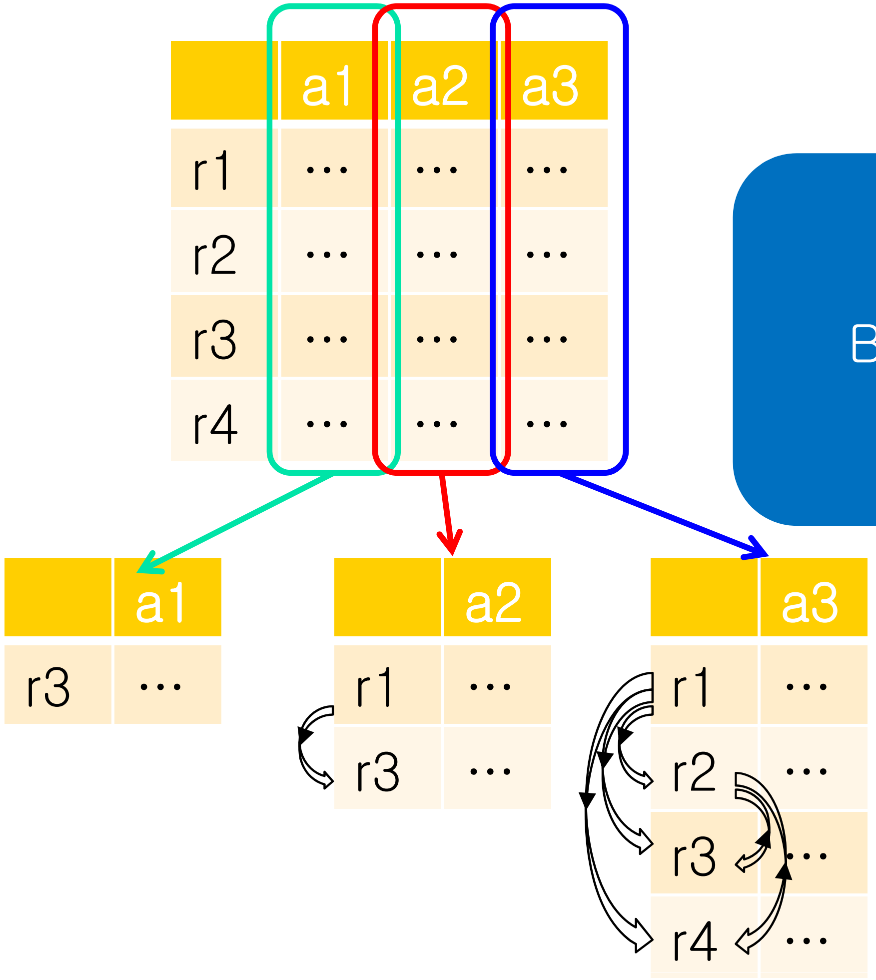
Use inverted lists

Generate a single inverted list in each dimension
Compute partial similarity of every pair in each inverted list

Prefix Filtering

- [Baraglia, Morales and Lucchese, ICDM, 2010]
- Document Similarity Self-Join with MapReduce
- Each record is inserted to the inverted lists of a subset of its dimensions only
- Compute partial similarities only
- Need to access original data to compute the exact similarities
- We can extend the previous naïve algorithm
    - SSJ-2
        - Access original data to compute the exact similarity for every candidate pair
    - SSJ-2R
        - Build additional file for non-indexed data
        - Access non-indexed data only to compute the exact similarity for every candidate pair

Prefix Filtering



	a1	a2	a3
r1	…	…	…
r2	…	…	…
r3	…	…	…
r4	…	…	…

	a1
r3	…

	a2
r1	…
r3	…

	a3
r1	…
r2	…
r3	…
r4	…

Generate a single inverted list
in each dimension.
But each record id is inserted to
inverted lists of a subset
of its dimensions only!

Use inverted lists

Prefix Filtering

- Let $D=\{v_1,v_2,\ldots,v_n\}$ where v_i is an m-dimensional vector and
  $v_i[j]$ is the v_i's j-th dimensional value
- Let $M_i=\max_{1\leq j\leq n}\{v_j[i]\}$ and $M=(M_1, \ldots, M_m)$
- Let b(y) be the smallest k with $k\leq m$ such that $\Sigma_{i=1}^{k} y[i]\cdot M_i \geq \sigma$
- Observation:
  - If two vectors x and y in D are similar
  - Both x and y have a common nonzero-valued dimension c s.t. $b(y) \leq c \leq m$
  - Otherwise, cosine(x,y) is less than σ

b(y)=2

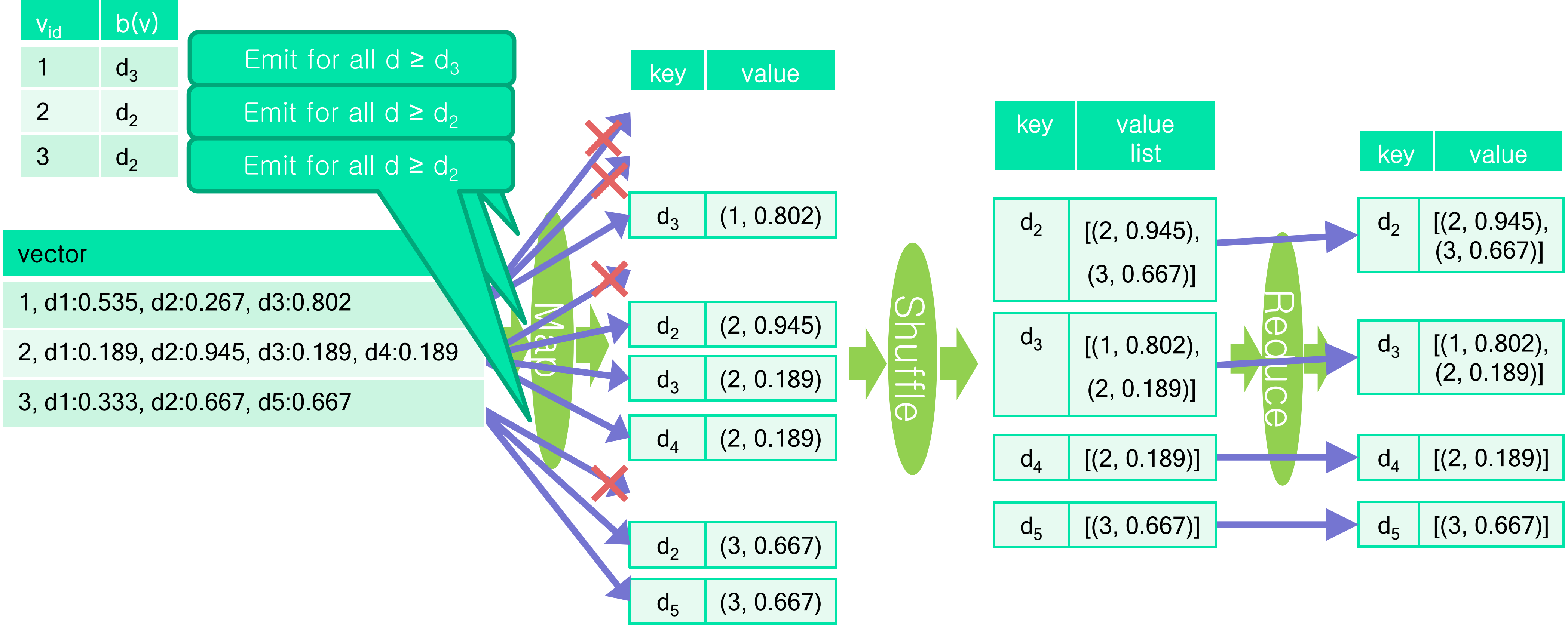
	d_1	d_2	d_3	d_4	d_5
M	0.70	0.54	1.00	0.67	0.57
y	0.70	0.42	0	0.12	0.57
x	0.67	0.54	0.23	0.40	0.19

Inserting only orange part is enough to find similar pairs

minimum similarity threshold: σ=0.6

An Illustration of Prefix Filtering

- Phase 1: build inverted lists



v_{id}	b(v)
1	d_3
2	d_2
3	d_2

Emit for all d ≥ d_3

Emit for all d ≥ d_2

Emit for all d ≥ d_2

vector
1, d1:0.535, d2:0.267, d3:0.802
2, d1:0.189, d2:0.945, d3:0.189, d4:0.189
3, d1:0.333, d2:0.667, d5:0.667

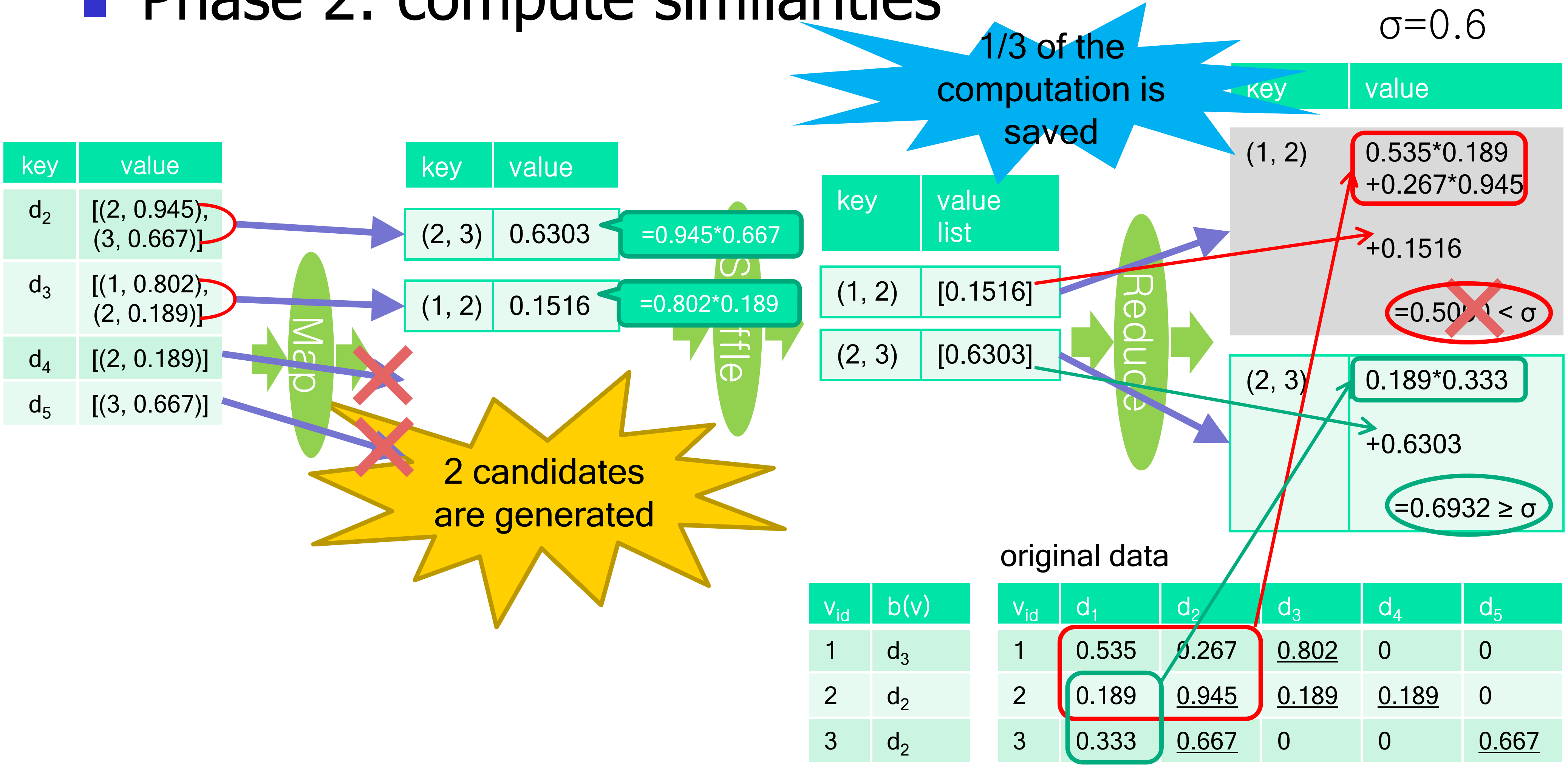
key	value
d_3	(1, 0.802)
d_2	(2, 0.945)
d_3	(2, 0.189)
d_4	(2, 0.189)
d_2	(3, 0.667)
d_5	(3, 0.667)

key	value list
d_2	[(2, 0.945), (3, 0.667)]
d_3	[(1, 0.802), (2, 0.189)]
d_4	[(2, 0.189)]
d_5	[(3, 0.667)]

key	value
d_2	[(2, 0.945), (3, 0.667)]
d_3	[(1, 0.802), (2, 0.189)]
d_4	[(2, 0.189)]
d_5	[(3, 0.667)]

An Illustration of Prefix Filtering

- Phase 2: compute similarities



σ=0.6

key	value
d_2	[(2, 0.945), (3, 0.667)]
d_3	[(1, 0.802), (2, 0.189)]
d_4	[(2, 0.189)]
d_5	[(3, 0.667)]

key	value
(2, 3)	0.6303
(1, 2)	0.1516

=0.945*0.667

=0.802*0.189

2 candidates are generated

1/3 of the computation is saved

key	value list
(1, 2)	[0.1516]
(2, 3)	[0.6303]

key	value
(1, 2)	0.535*0.189 +0.267*0.945 +0.1516 =0.50[illegible] < σ
(2, 3)	0.189*0.333 +0.6303 =0.6932 ≥ σ

original data

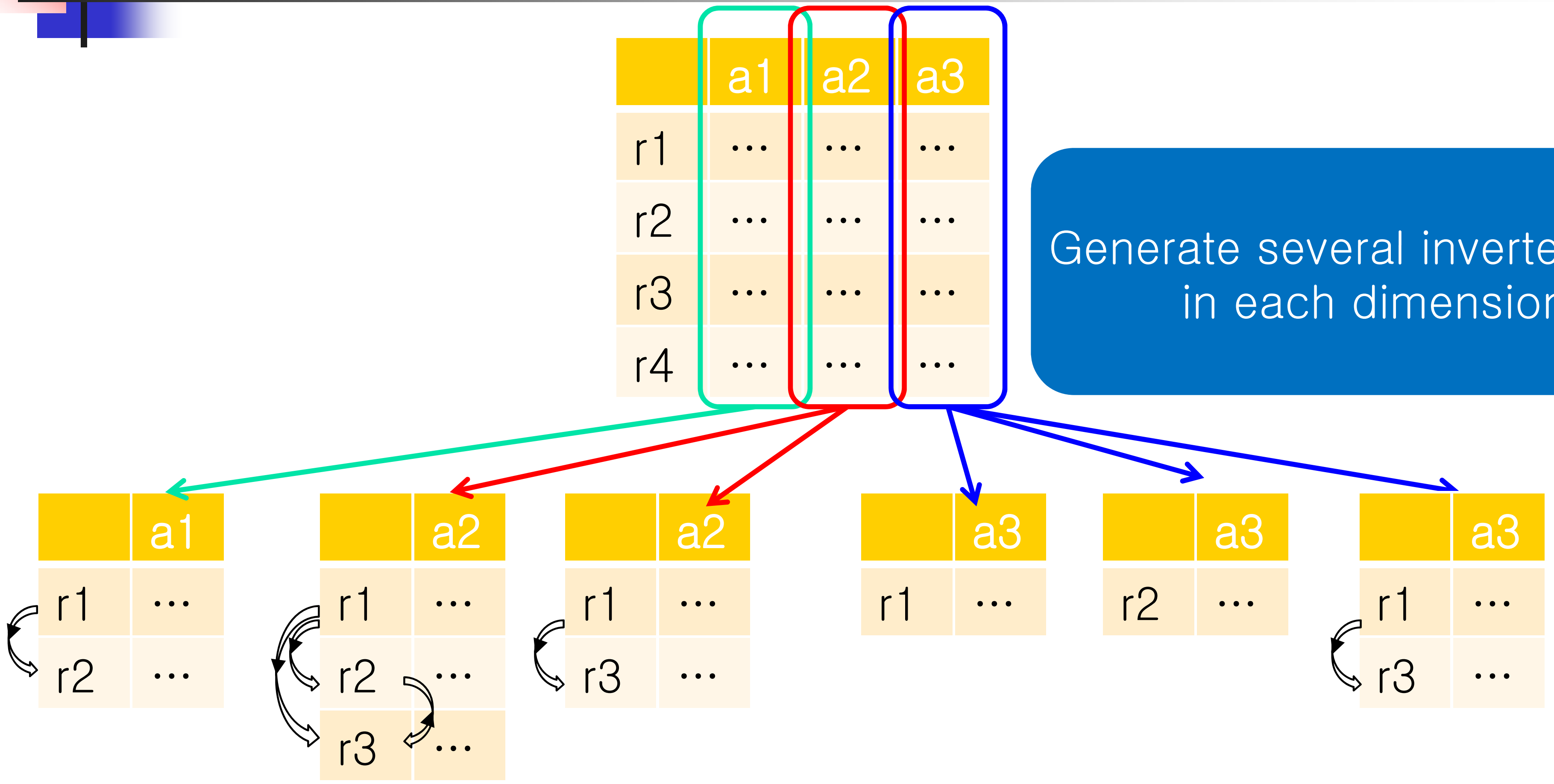
v_{id}	b(v)
1	d_3
2	d_2
3	d_2

v_{id}	d_1	d_2	d_3	d_4	d_5
1	0.535	0.267	<u>0.802</u>	0	0
2	0.189	<u>0.945</u>	<u>0.189</u>	<u>0.189</u>	0
3	0.333	<u>0.667</u>	0	0	<u>0.667</u>

Bucket Filtering

- [Kim, Shim: ICDE 2012]
- Parallel similarity self-join of vectors with Euclidean distance using MapReduce
- For Euclidean distance, zero values in each dimension should be also inserted in inverted lists
  - e.g.) $p_1=(1,0)$, $p_2=(0,1)$ → Euclidean distance = 1^2+1^2
- Build inverted lists with sub-ranges in a subset of dimensions
  - Map function
    - Divide the data points into partitions to make inverted lists
  - Reduce function
    - Output the similar pairs of vectors in each inverted list

Bucket Filtering



	a1	a2	a3
r1	…	…	…
r2	…	…	…
r3	…	…	…
r4	…	…	…

	a1
r1	…
r2	…

	a2
r1	…
r2	…
r3	…

	a2
r1	…
r3	…

	a3
r1	…

	a3
r2	…

	a3
r1	…
r3	…

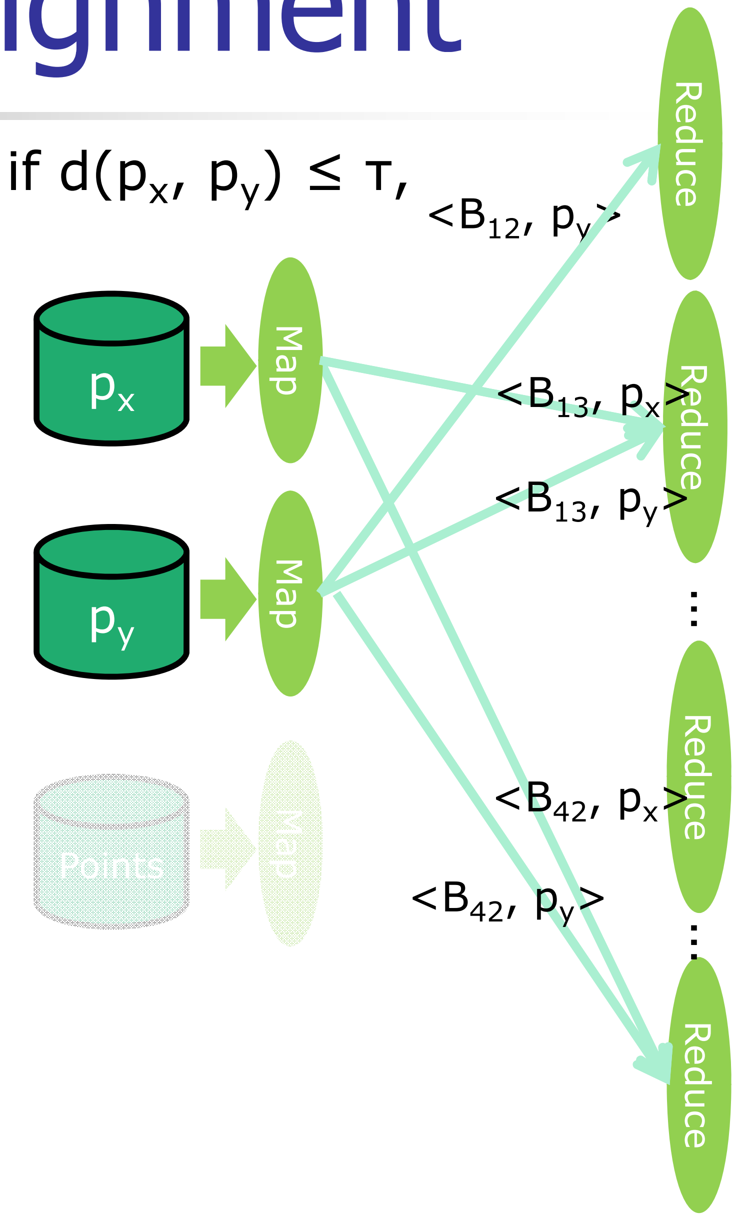
Generate several inverted lists in each dimension

Use inverted lists

τ-Safe Bucket Assignment

- Utilize a τ-safe bucket assignment
  - Given an upper bound distance τ
  - Partition the points into buckets $\{B_{ij}\}$ such that

    [every similar pair appears at least in a bucket]
- After τ-safe bucket assignment, we can find the similar pairs within distance τ in each bucket independently



An Illustration of Similarity Join with τ-Safe Bucket Assignment

	$p_i(1)$	$p_i(2)$	$p_i(3)$
p_1	0.78	0.4	0.01
p_2	0.07	0.21	0.57
p_3	0.51	0.11	0.32
p_4	0.31	0.79	0.9
p_5	0.77	0.42	0.02
p_6	0.8	0.39	0.04

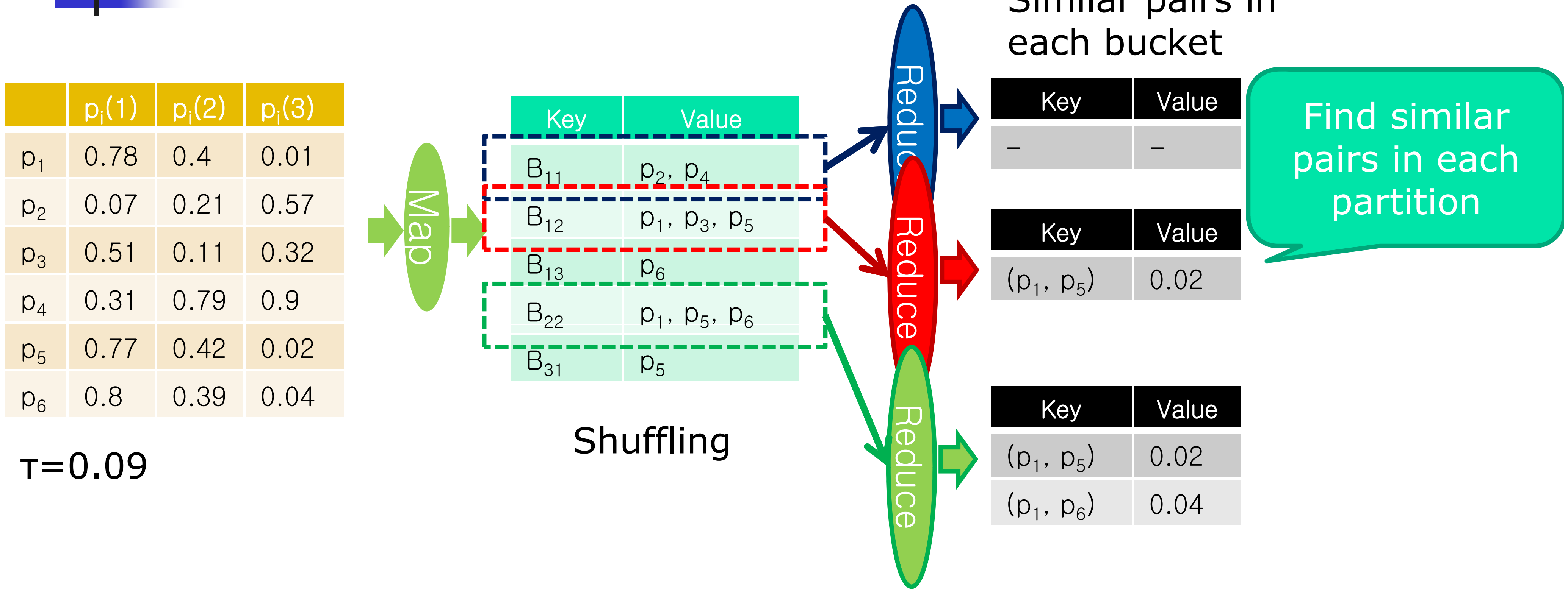
τ=0.09

Map

Key	Value
B_{12}	p_1
B_{22}	p_1

Key	Value
B_{12}	p_5
B_{22}	p_5
B_{31}	p_5
B_{13}	p_6
B_{22}	p_6

An Illustration of Similarity Join with Bucket Filtering



	$p_i(1)$	$p_i(2)$	$p_i(3)$
p_1	0.78	0.4	0.01
p_2	0.07	0.21	0.57
p_3	0.51	0.11	0.32
p_4	0.31	0.79	0.9
p_5	0.77	0.42	0.02
p_6	0.8	0.39	0.04

$\tau=0.09$

Key	Value
B_{11}	p_2, p_4
B_{12}	p_1, p_3, p_5
B_{13}	p_6
B_{22}	p_1, p_5, p_6
B_{31}	p_5

Shuffling

Similar pairs in each bucket

Key	Value
–	–

Key	Value
(p_1, p_5)	0.02

Key	Value
(p_1, p_5)	0.02
(p_1, p_6)	0.04

Find similar pairs in each partition

Parallel Top-K Similarity Joins Using MapReduce

- [Kim, Shim: ICDE 2012]
- Handle vector data with Euclidean distance
- Propose improved serial top-k similarity join algorithms
- Parallelize the improved top-k similarity join algorithms using MapReduce

Data Mining Algorithms using MapReduce

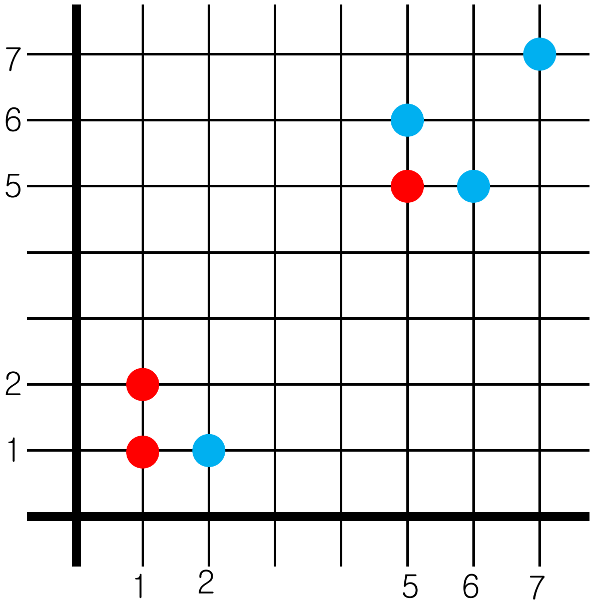
Clustering using MapReduce

K-Means Clustering using MapReduce

K-Means Clustering Method: Partitioning Approach

- Given *k*, the *k-means* algorithm performs the following repeatedly
  1. Partition objects into *k* nonempty subsets
  2. Compute the centroids of the clusters in the current partition (the centroid is the center, i.e., *mean point*, of the cluster)
  3. Assign each object to the cluster with the nearest centroid
  4. Stop when no more new assignments. Otherwise go back to Step 2
- The above loop finds a clustering. Thus, repeat the above many times and select the best clustering

An Illustration of K-Means Clustering (K = 2)

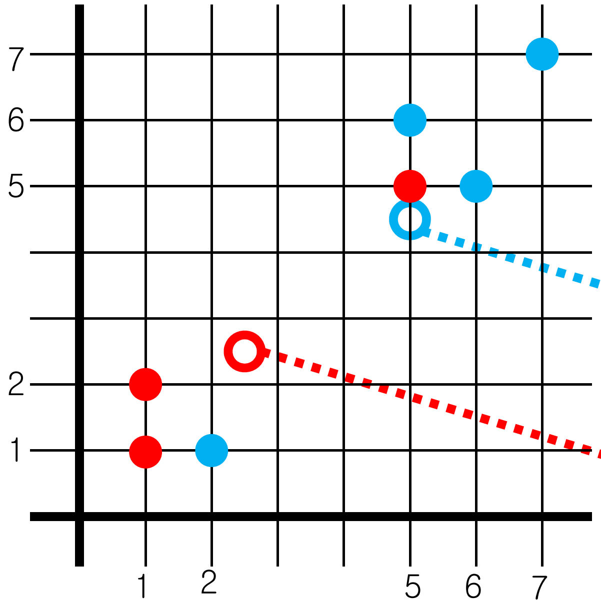


(1, 1)   (2, 1)   (1, 2)   (5, 5)   (6, 5)   (5, 6)   (7, 7)

Class A	Class B
(1, 1) (1, 2) (5, 5)	(2, 1) (6, 5) (5, 6) (7, 7)

Assume we randomly partitioned the objects into 2 nonempty subsets as above!

An Illustration of K-Means Clustering (K = 2)



(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) (5, 6) (7, 7)

Class A

(1, 1) (1, 2) (5, 5)

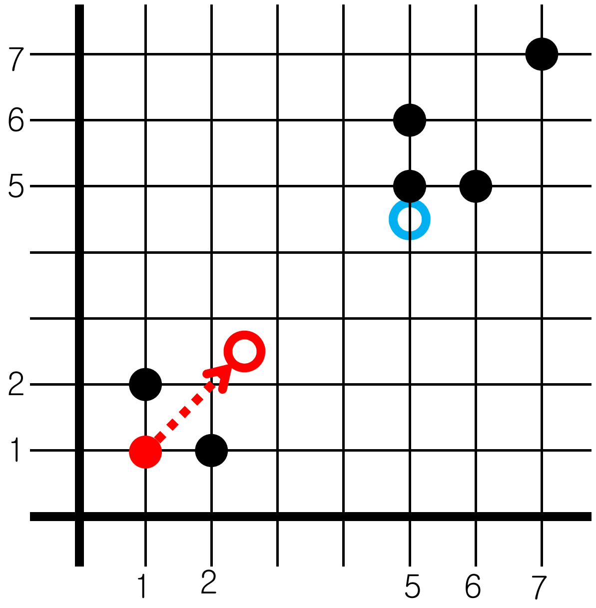
Class B

(2, 1) (6, 5) (5, 6) (7, 7)

Center : (7/3, 8/3)

Center : (20/4, 19/4)

An Illustration of K-Means Clustering (K = 2)

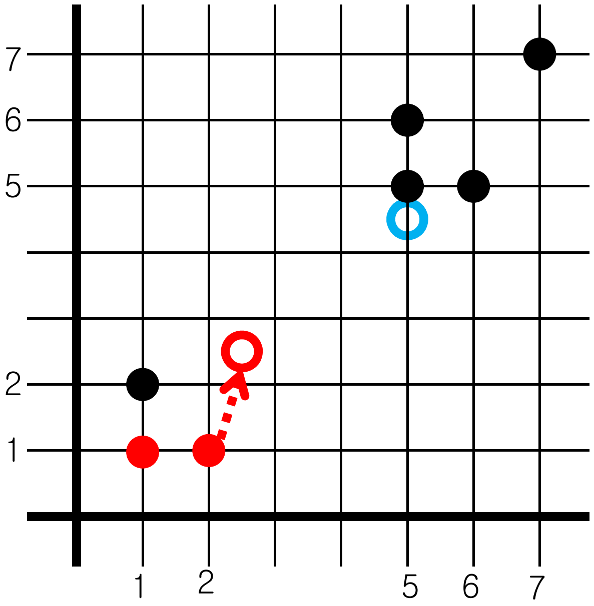


(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) (5, 6) (7, 7)

Class A	Class B
(1, 1)	

Center : (7/3, 8/3) Center : (20/4, 19/4)

An Illustration of K-Means Clustering (K = 2)



(1, 1)   *(2, 1)*   (1, 2)   (5, 5)   (6, 5)   (5, 6)   (7, 7)

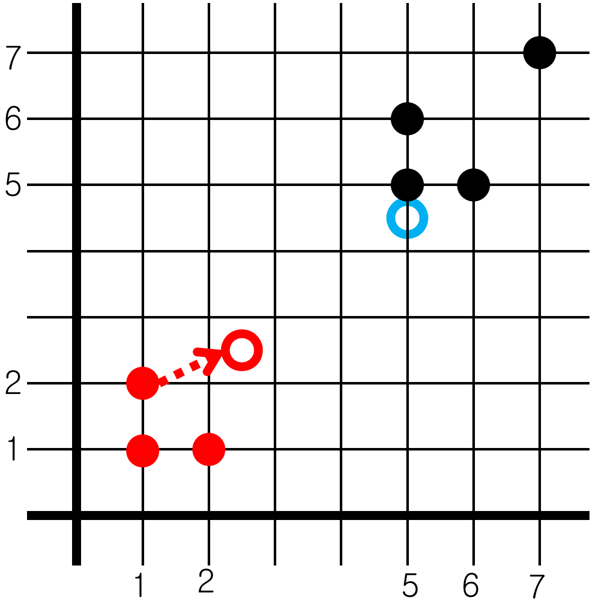
Class A

(1, 1)   (2, 1)

Class B

Center : (7/3, 8/3)   Center : (20/4, 19/4)

An Illustration of K-Means Clustering (K = 2)

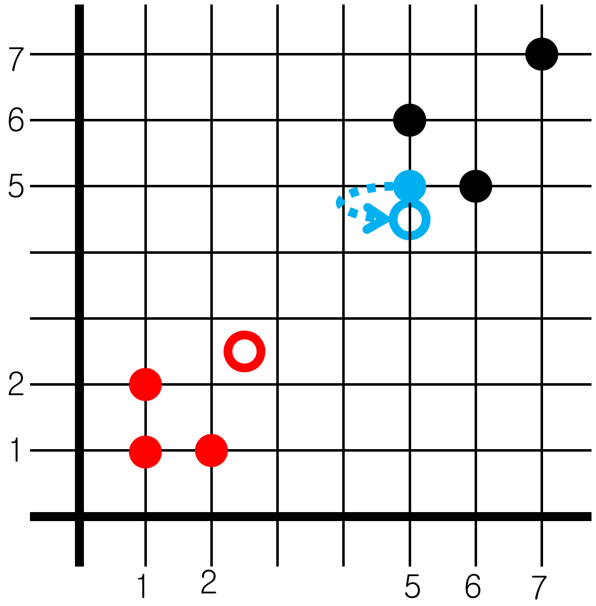


(1, 1) (2, 1) *(1, 2)* (5, 5) (6, 5) (5, 6) (7, 7)

Class A	Class B
(1, 1) (2, 1) (1, 2)	

Center : (7/3, 8/3) Center : (20/4, 19/4)

An Illustration of K-Means Clustering (K = 2)

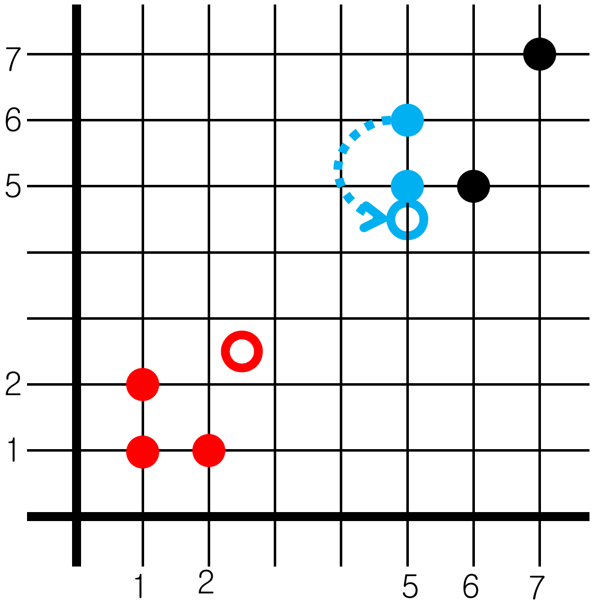


(1, 1) (2, 1) (1, 2) *(5, 5)* (6, 5) (5, 6) (7, 7)

Class A	Class B
(1, 1) (2, 1) (1, 2)	(5, 5)

Center : (7/3, 8/3) Center : (20/4, 19/4)

An Illustration of K-Means Clustering (K = 2)

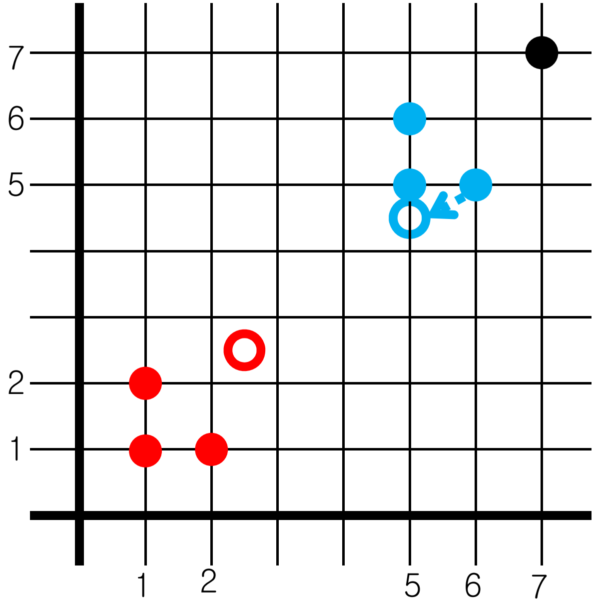


(1, 1) (2, 1) (1, 2) (5, 5) *(6, 5)* (5, 6) (7, 7)

Class A	Class B
(1, 1) (2, 1) (1, 2)	(5, 5) (6, 5)

Center : (7/3, 8/3) Center : (20/4, 19/4)

An Illustration of K-Means Clustering (K = 2)

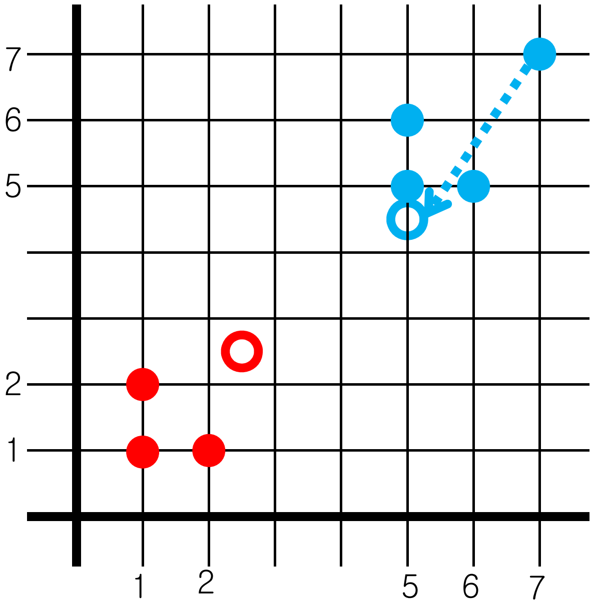


(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) *(5, 6)* (7, 7)

Class A	Class B
(1, 1) (2, 1) (1, 2)	(5, 5) (6, 5) (5, 6)

Center : (7/3, 8/3) Center : (20/4, 19/4)

An Illustration of K-Means Clustering (K = 2)

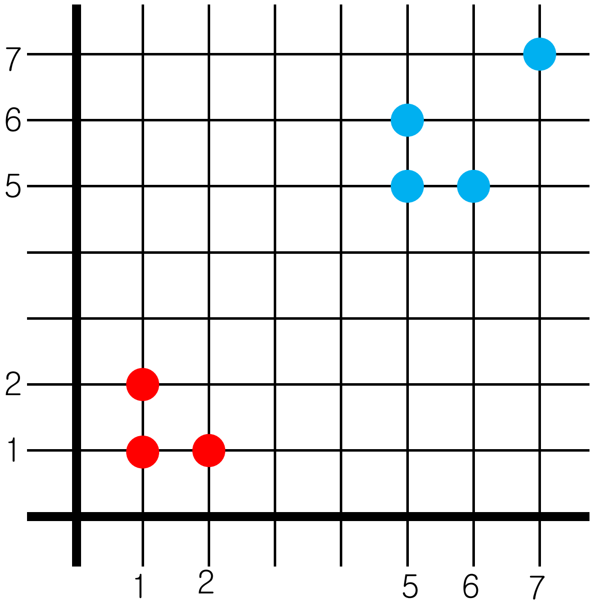


(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) (5, 6) *(7, 7)*

Class A	Class B
(1, 1) (2, 1) (1, 2)	(5, 5) (6, 5) (5, 6) (7, 7)

Center : (7/3, 8/3) Center : (20/4, 19/4)

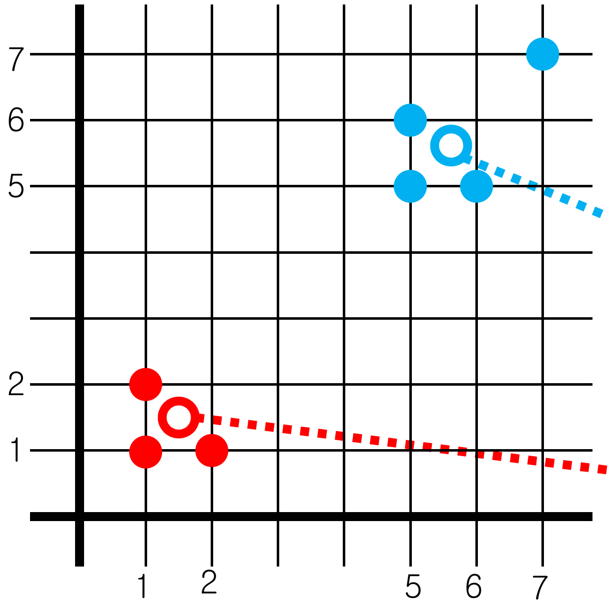
An Illustration of K-Means Clustering (K = 2)



(1, 1)	(2, 1)	(1, 2)	(5, 5)	(6, 5)	(5, 6)	(7, 7)

Class A	Class B
(1, 1) (2, 1) (1, 2)	(5, 5) (6, 5) (5, 6) (7, 7)

An Illustration of K-Means Clustering (K = 2)



(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) (5, 6) (7, 7)

Class A

(1, 1) (2, 1) (1, 2)

Class B

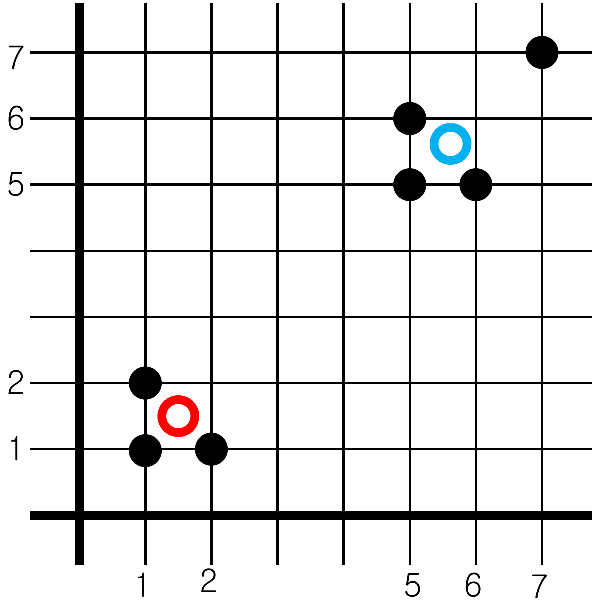
(5, 5) (6, 5) (5, 6) (7, 7)

Center : (4/3, 4/3)

Center : (23/4, 23/4)

Update the cluster means

An Illustration of K-Means Clustering (K = 2)

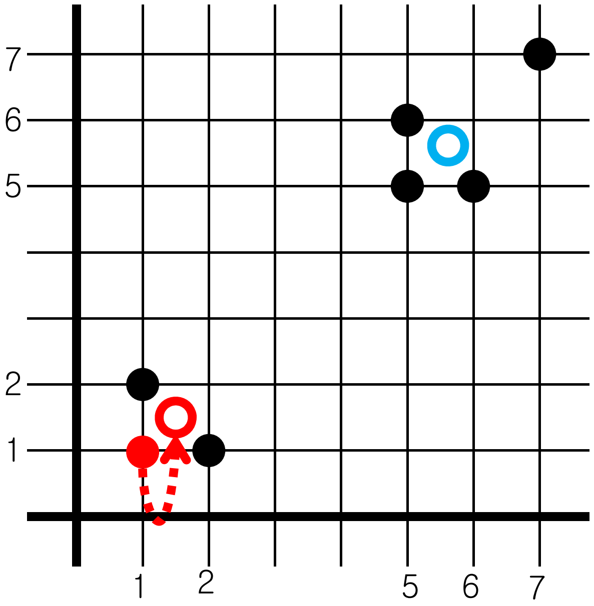


(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) (5, 6) (7, 7)

Class A	Class B
(1, 1)	

Center : (4/3, 4/3) Center : (23/4, 23/4)

An Illustration of K-Means Clustering (K = 2)

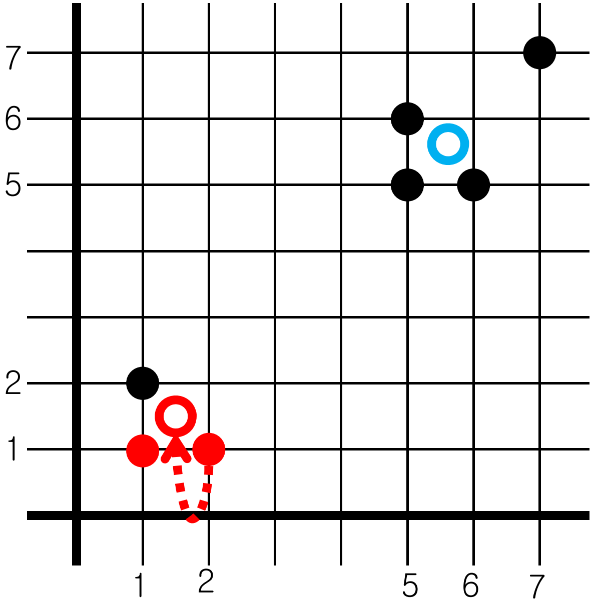


(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) (5, 6) (7, 7)

Class A	Class B
(1, 1)	

Center : (4/3, 4/3) Center : (23/4, 23/4)

An Illustration of K-Means Clustering (K = 2)



(1, 1) *(2, 1)* (1, 2) (5, 5) (6, 5) (5, 6) (7, 7)

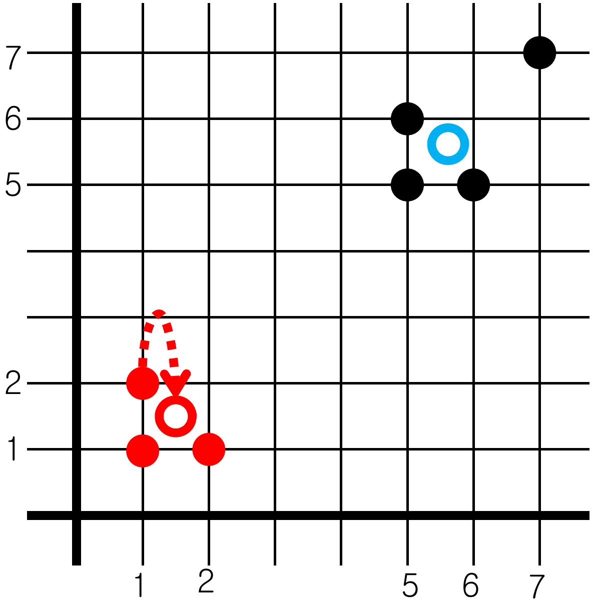
Class A

(1, 1) (2, 1)

Class B

Center : (4/3, 4/3) Center : (23/4, 23/4)

An Illustration of K-Means Clustering (K = 2)

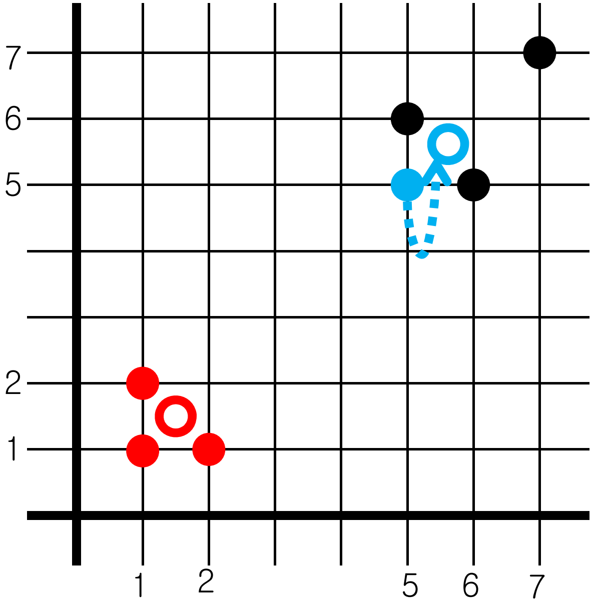


(1, 1) (2, 1) *(1, 2)* (5, 5) (6, 5) (5, 6) (7, 7)

Class A	Class B
(1, 1) (2, 1) (1, 2)	

Center : (4/3, 4/3) Center : (23/4, 23/4)

An Illustration of K-Means Clustering (K = 2)

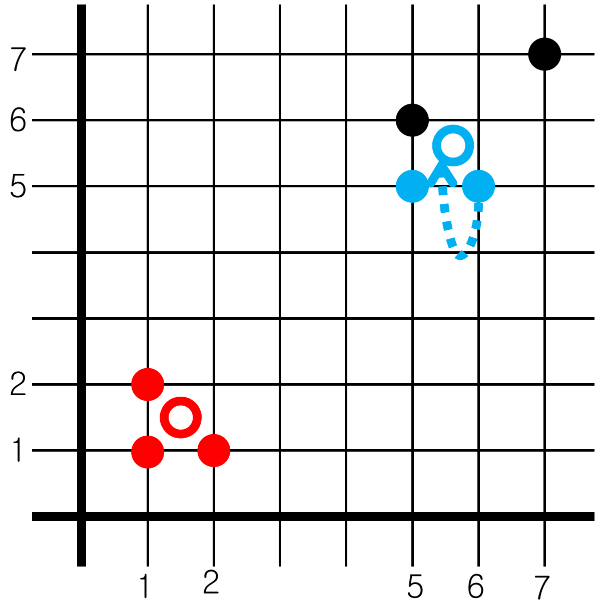


(1, 1) (2, 1) (1, 2) *(5, 5)* (6, 5) (5, 6) (7, 7)

Class A	Class B
(1, 1) (2, 1) (1, 2)	(5, 5)

Center : (4/3, 4/3) Center : (23/4, 23/4)

An Illustration of K-Means Clustering (K = 2)

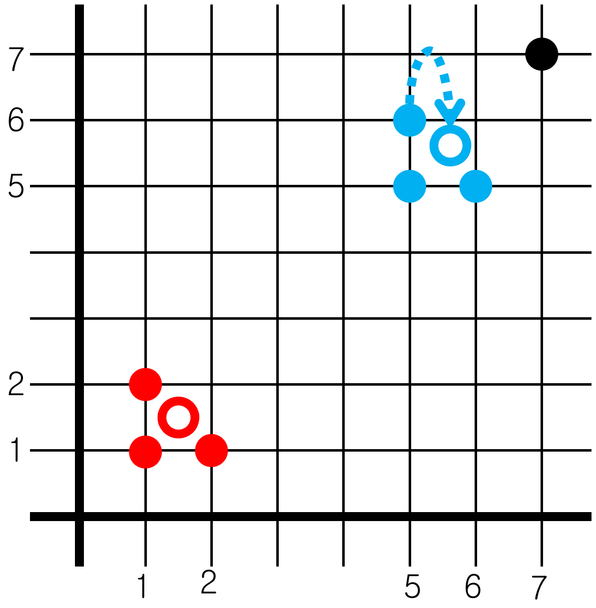


(1, 1)   (2, 1)   (1, 2)   (5, 5)   *(6, 5)*   (5, 6)   (7, 7)

Class A	Class B
(1, 1) (2, 1) (1, 2)	(5, 5) (6, 5)

Center : (4/3, 4/3)   Center : (23/4, 23/4)

An Illustration of K-Means Clustering (K = 2)



(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) *(5, 6)* (7, 7)

Class A

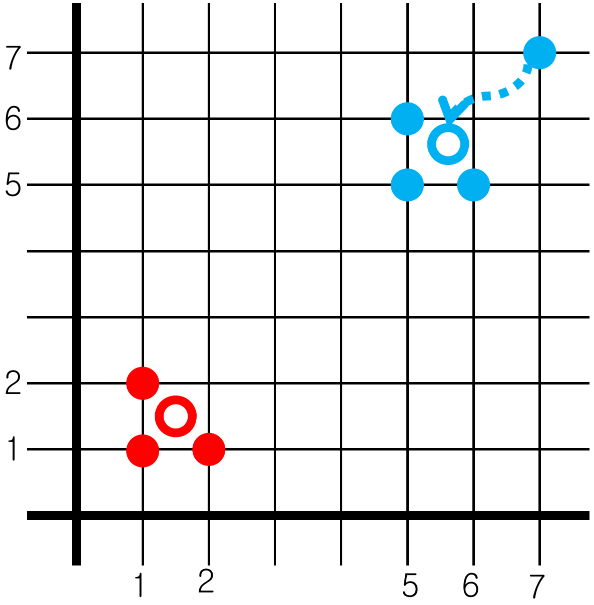
(1, 1) (2, 1) (1, 2)

Class B

(5, 5) (6, 5) (5, 6)

Center : (4/3, 4/3) Center : (23/4, 23/4)

An Illustration of K-Means Clustering (K = 2)

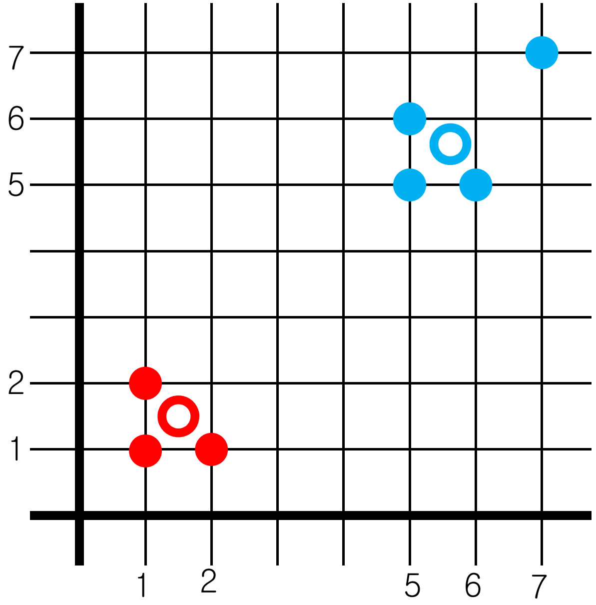


(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) (5, 6) *(7, 7)*

Class A	Class B
(1, 1) (2, 1) (1, 2)	(5, 5) (6, 5) (5, 6) (7, 7)

Center : (4/3, 4/3) Center : (23/4, 23/4)

An Illustration of K-Means Clustering (K = 2)



(1, 1) (2, 1) (1, 2) (5, 5) (6, 5) (5, 6) (7, 7)

Class A	Class B
(1, 1) (2, 1) (1, 2)	(5, 5) (6, 5) (5, 6) (7, 7)

Center : (4/3, 4/3) Center : (23/4, 23/4)

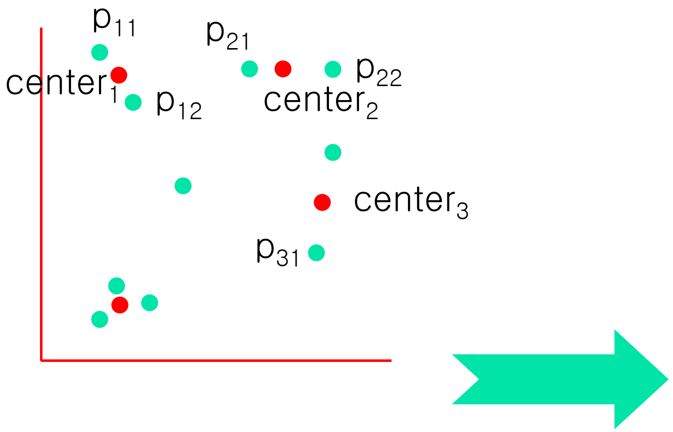
No change ⇒ STOP

K-Means using Map/Reduce

- Iteratively improves partitioning of data into *k* clusters
- Do
    - Map
        - Input is a data point and k centers are broadcasted
        - Finds the closest center among k centers for the input point
    - Reduce
        - Input is one of k centers and all data points having this center as their closest center
        - Calculates the new center using data points
- until all of new centers are not changed

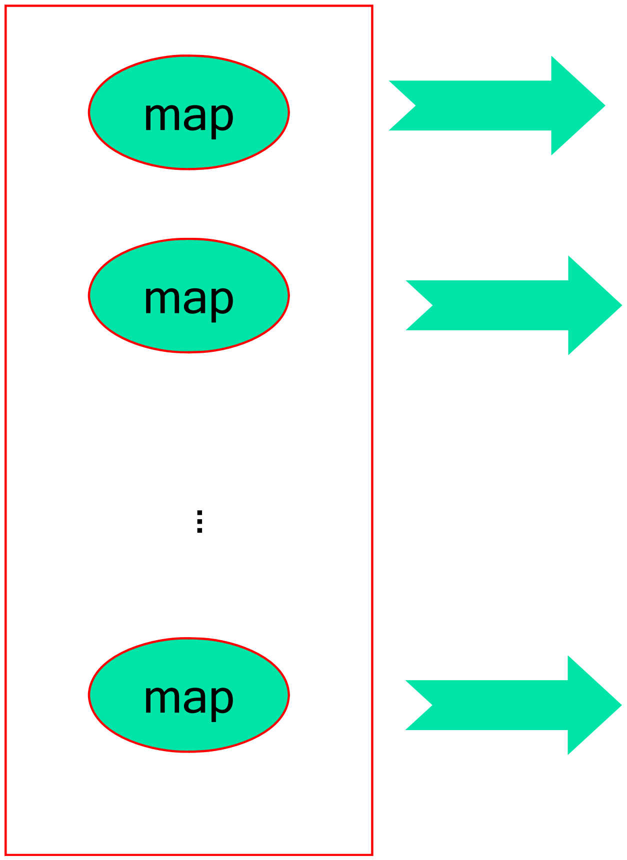
An Illustration of K-means Clustering: Map

K = 4



K centers are broadcasted to every map function!

Broadcasting is used!



key value

($center_1$, p_{11})

($center_1$, p_{12})

($center_2$, p_{21})

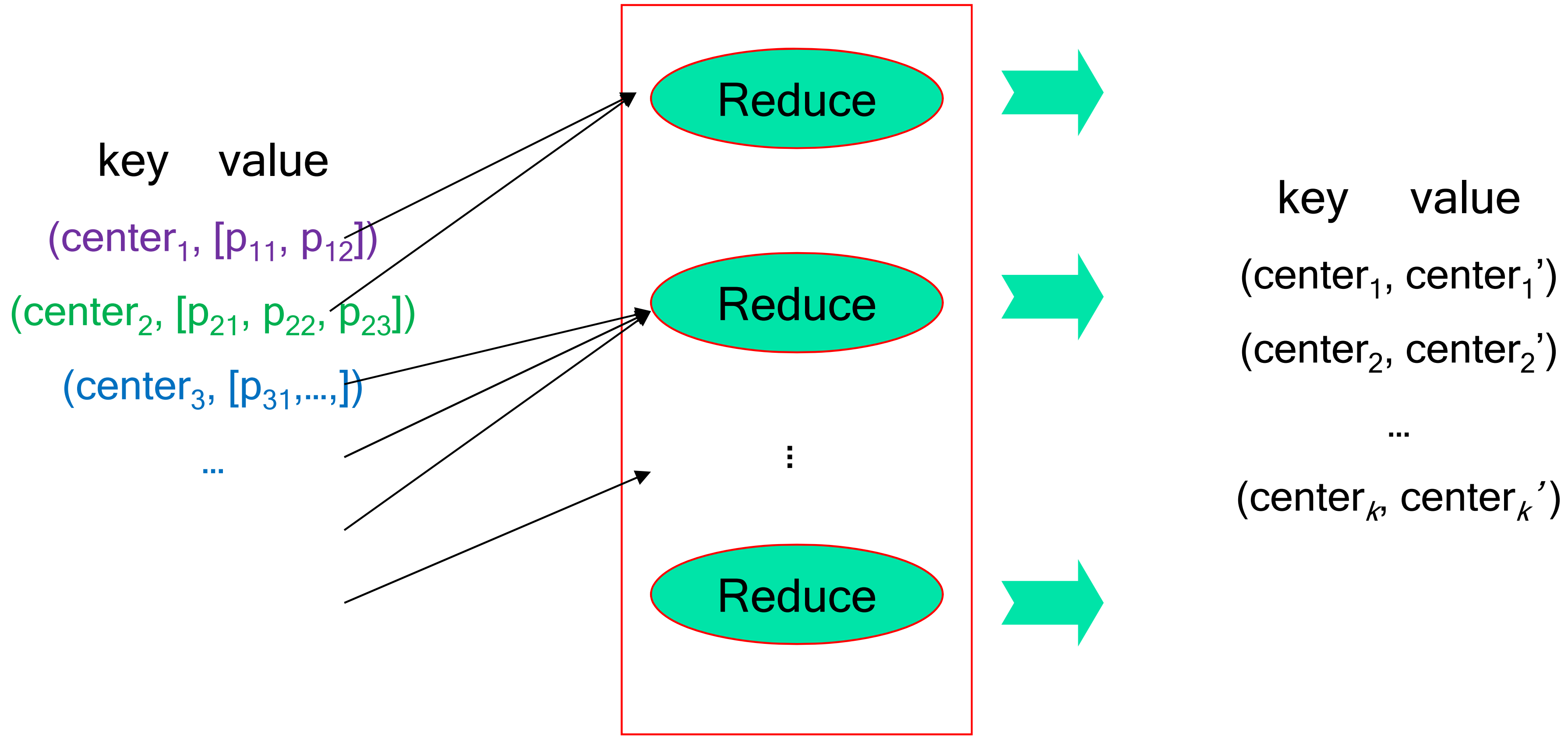
($center_3$, p_{31})

…

Find the closest center

An Illustration of K-means Clustering: Reduce

K = 4



key value

(center$_1$, [p$_{11}$, p$_{12}$])

(center$_2$, [p$_{21}$, p$_{22}$, p$_{23}$])

(center$_3$, [p$_{31}$,…,])

…

key value

(center$_1$, center$_1$')

(center$_2$, center$_2$')

…

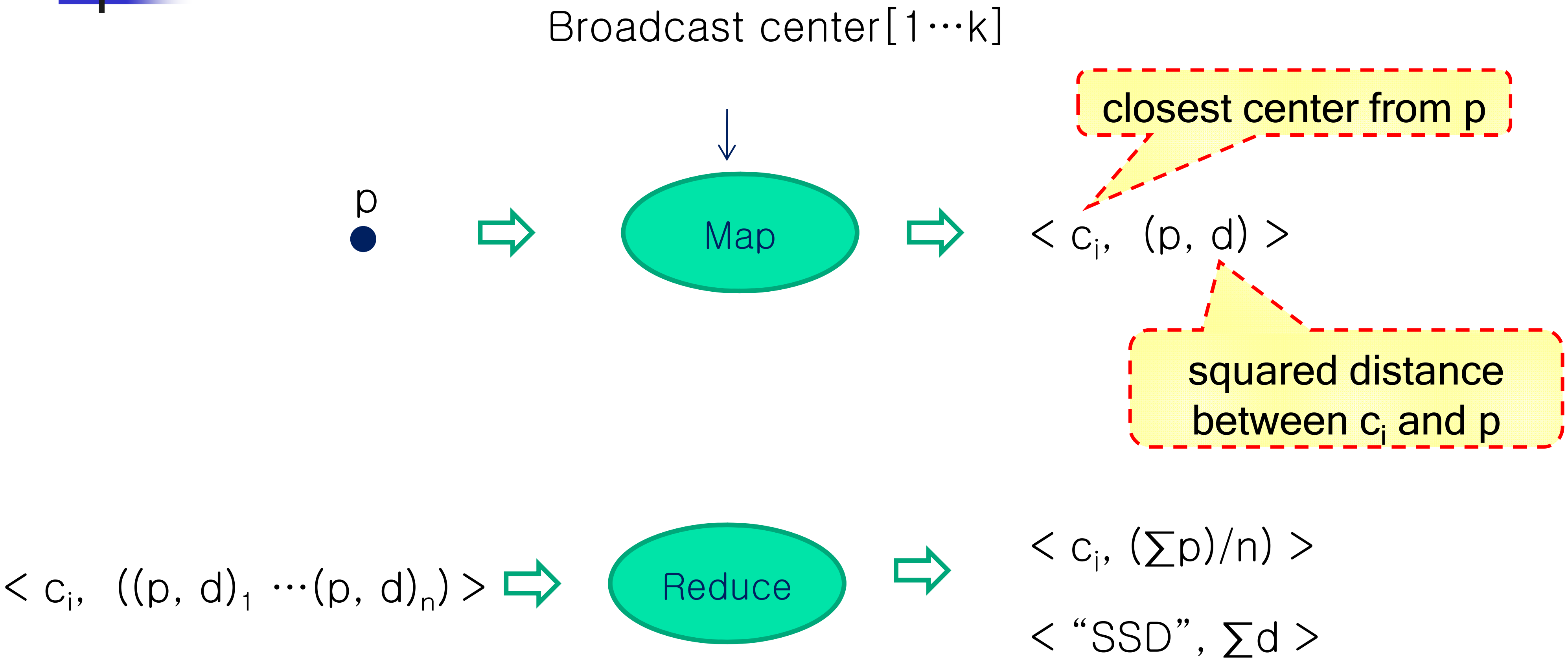
(center$_k$, center$_k$')

Calculate new center

K-means for Large Data

- Alternative terminating condition is needed
- Iteratively execute map/reduce procedure until $|E_{cur}-E_{prev}| \leq \varepsilon$
  - E_{cur}: sum of squared distance in the current step
  - E_{prev}: sum of squared distance in the previous step

K-means for Large Data



Broadcast center[1⋯k]

p

Map

$< c_i, (p, d) >$

closest center from p

squared distance between c_i and p

$< c_i, ((p, d)_1 \cdots (p, d)_n) >$

Reduce

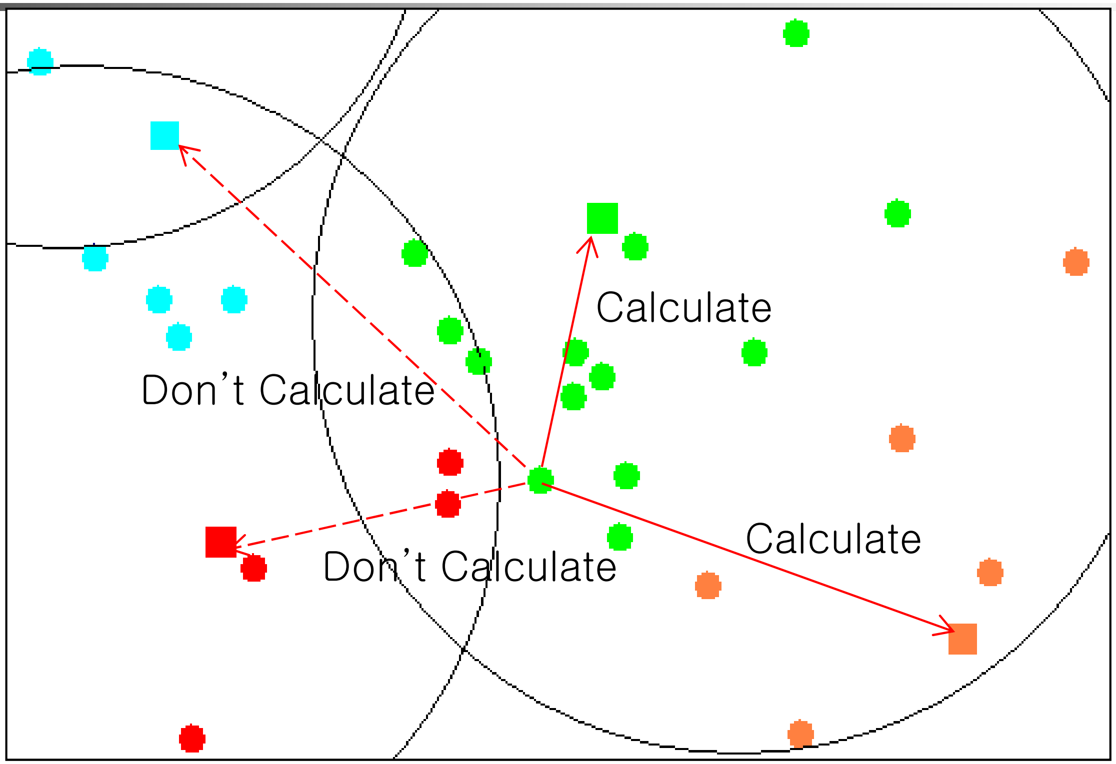
$< c_i, (\sum p)/n) >$

$<$ "SSD", $\sum d >$

Canopy Clustering with K-Means MapReduce Algorithm

- To reduce the distance computations between each point and cluster centers, use canopy clustering algorithm in [McCallum, Nigam, Ungar: SIGKDD 2000] as preliminary step
- Given
  - A list of the data points
  - Two distance thresholds,T1 and T2, (T1>T2)
- Repeat until the list is empty
  - Pick a point randomly and calculate its distance to all other points
  - Put all points that are within distance threshold T1 into a canopy
  - Remove from the list all points that are within distance threshold T2
- After the Canopy Clustering
  - Resume hierarchical or partitional clustering as usual
  - Treat objects in separate canopy clusters as being at infinite distances

K-Means Using Canopy

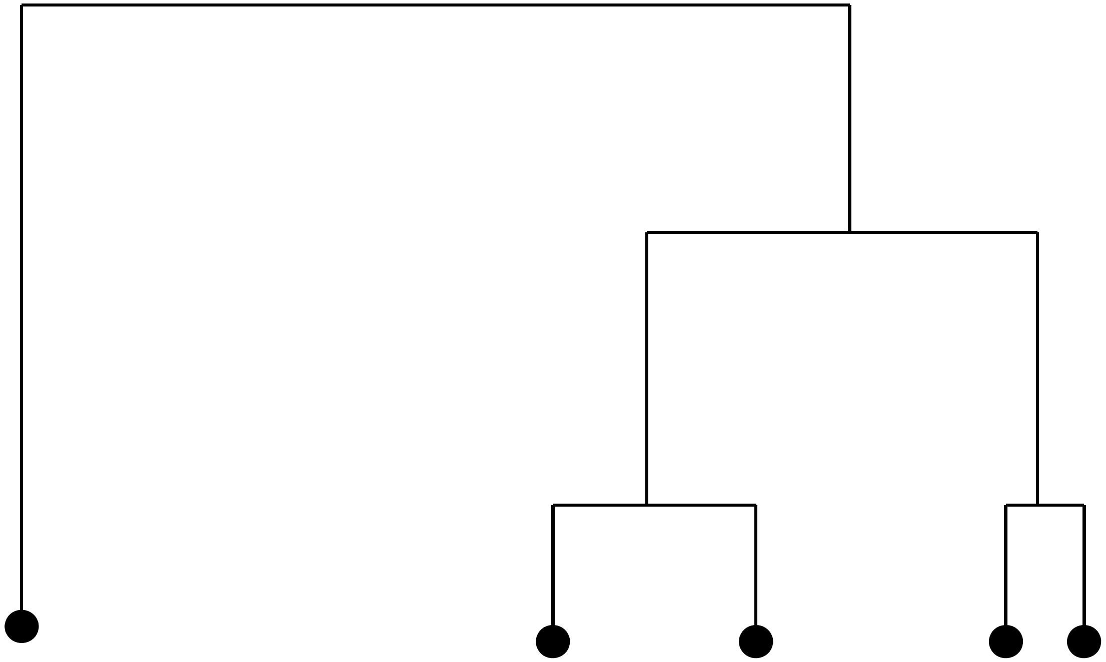


Calculate distance to centers which is <u>in the same canopy</u> and then perform k–Means

Hierarchical Clustering using MapReduce

Hierarchical Clustering

- Nested Partitions
- Tree structure



Agglomerative Hierarchical Clustering Algorithms

- Mostly used hierarchical clustering algorithm
- Initially each point is a distinct cluster
- Repeatedly merge closest clusters until the number of clusters becomes K
  - Closest: $d_{mean}(C_i, C_j) = \|m_i - m_j\|$

    $d_{min}(C_i, C_j) = \min_{p \in C_i, q \in C_j} \|p - q\|$

    Likewise $d_{ave}(C_i, C_j)$ and $d_{max}(C_i, C_j)$

Hierarchical Clustering using MapReduce

- Do
  - 1st MapReduce phase
    - Find the closest pair of clusters
  - 2nd MapReduce phase
    - Merge the closest pair of clusters
- until *k* clusters remain

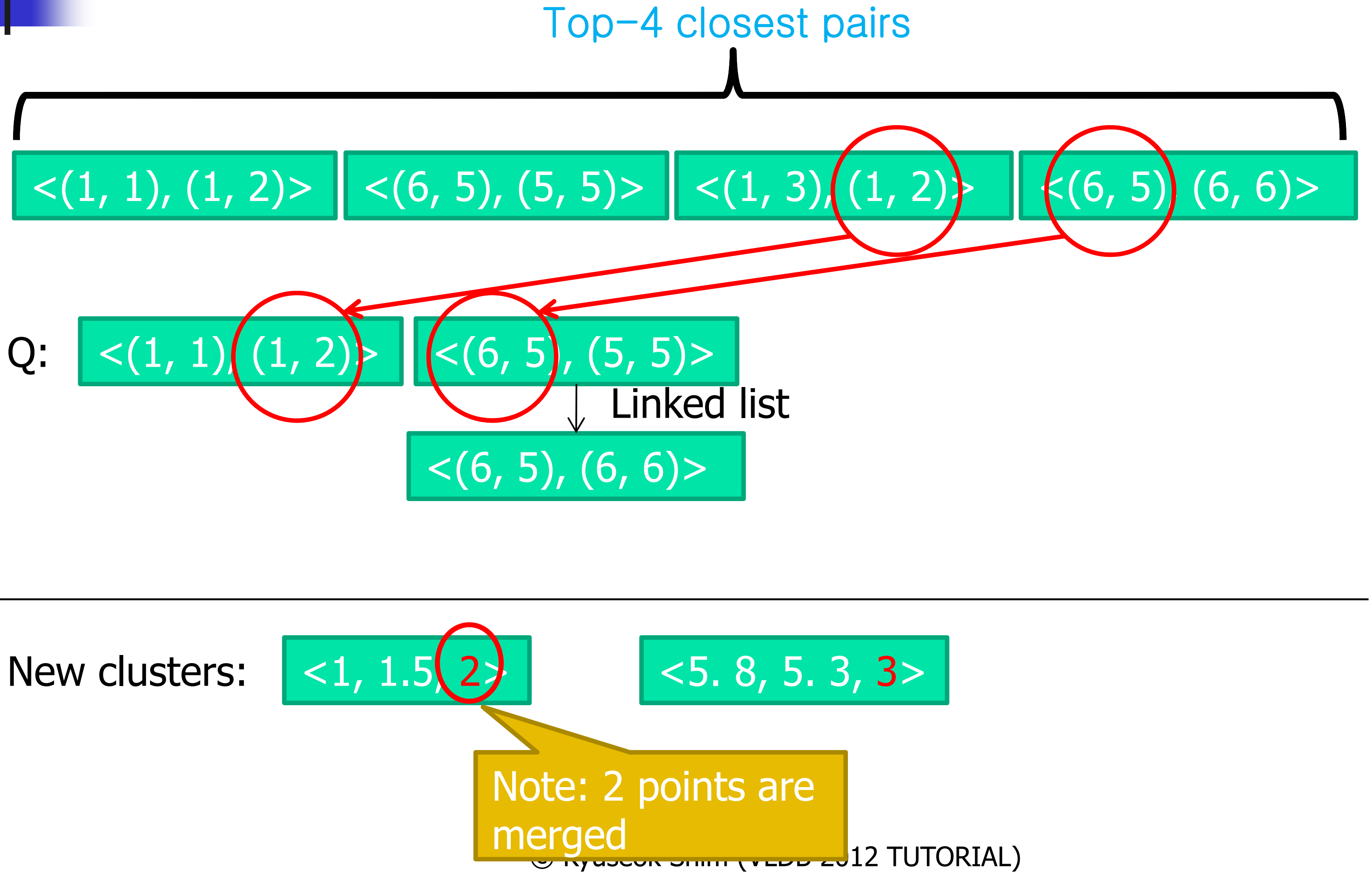
Hierarchical Clustering using MapReduce

- To find the closest pair, we perform top-1 similarity join
  - e.g.) utilize a top-k similarity join algorithm using MapReduce in [Kim, Shim: ICDE 2012]
- Extremely inefficient since we have to perform top-1 similarity joins O(n) times
- To speed up, utilize top-k similarity joins to find the top-k closest clusters instead of top-1 similarity join result
  - Approximate clustering algorithm [Sun, Shuy, Liy, Yuy, Ma, Fang: PDCAT, 2009]

Approximate Hierarchical Clustering using MapReduce

- Merge the points of the top-k closet clusters in a single machine
- Choose clusters to merge and keep in a queue Q
- While reading the top-k closest pairs (u,v) in the increasing order of their distances
  - If (u,v) share no points with the pairs in Q, insert it into Q
  - If v appears at least once in Q, ignore (u,v)
  - If (u, v’) appears in Q, merge (u, v) with (u, v’)

Illustration



Top-4 closest pairs

<(1, 1), (1, 2)> <(6, 5), (5, 5)> <(1, 3), (1, 2)> <(6, 5), (6, 6)>

Q: <(1, 1), (1, 2)> <(6, 5), (5, 5)>

Linked list

<(6, 5), (6, 6)>

New clusters: <1, 1.5, 2> <5. 8, 5. 3, 3>

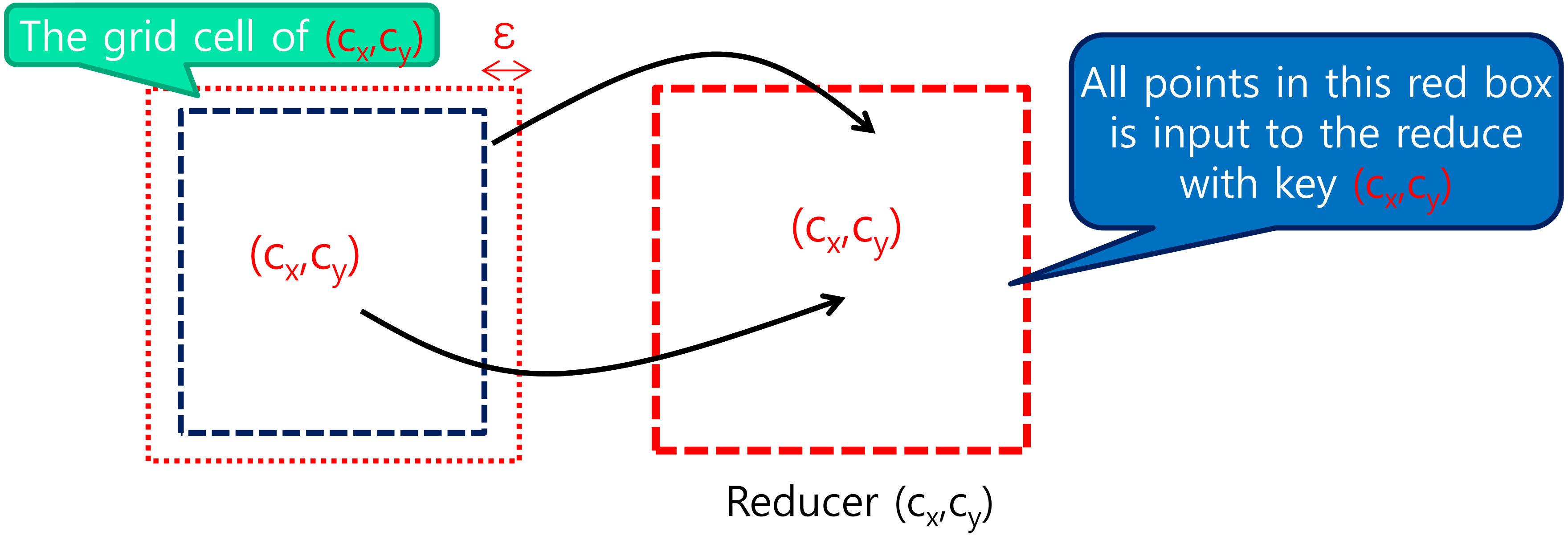
Note: 2 points are merged

DBSCAN using MapReduce

- MR-DBSCAN [He, Tan, Luo, Mao, Feng, Fan: ICPADS 2011]
- Step 1: Preprocess
  - Divide data space into a grid to distribute the data points into every grid cell evenly
- Step 2: Perform DBSCAN locally
  - Perform DBSCAN algorithm in each grid cell
- Step 3: Find clusters to be merged
  - With the clusters of the points in the border of grid cells, find every pair of cluster ids to be merged
- Step 4: Merge clusters
  - In a single machine, merge all clusters and label the cluster id for each point

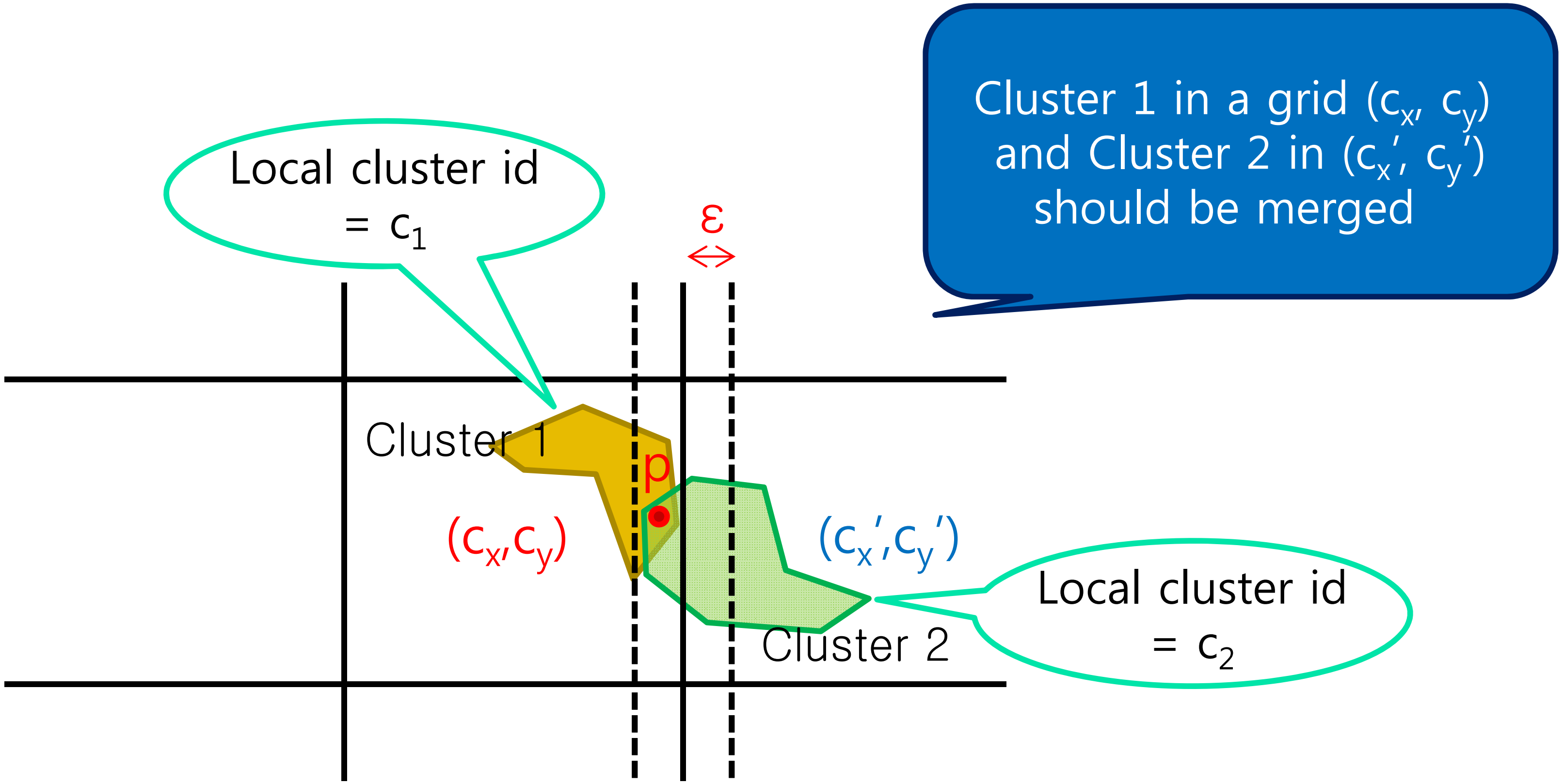
Step 2: Local DBSCAN

- Broadcast the ranges of each dimension for partitioning
- Perform DBSCAN locally for each grid
  - To compute the ε-neighborhood of every point in a grid correctly, we collect the points for the grid expanded by ε



Step 3: Find the Clusters to be Merged

- With the clusters of the points in the border of grid cells, find every pair of cluster ids to be merged

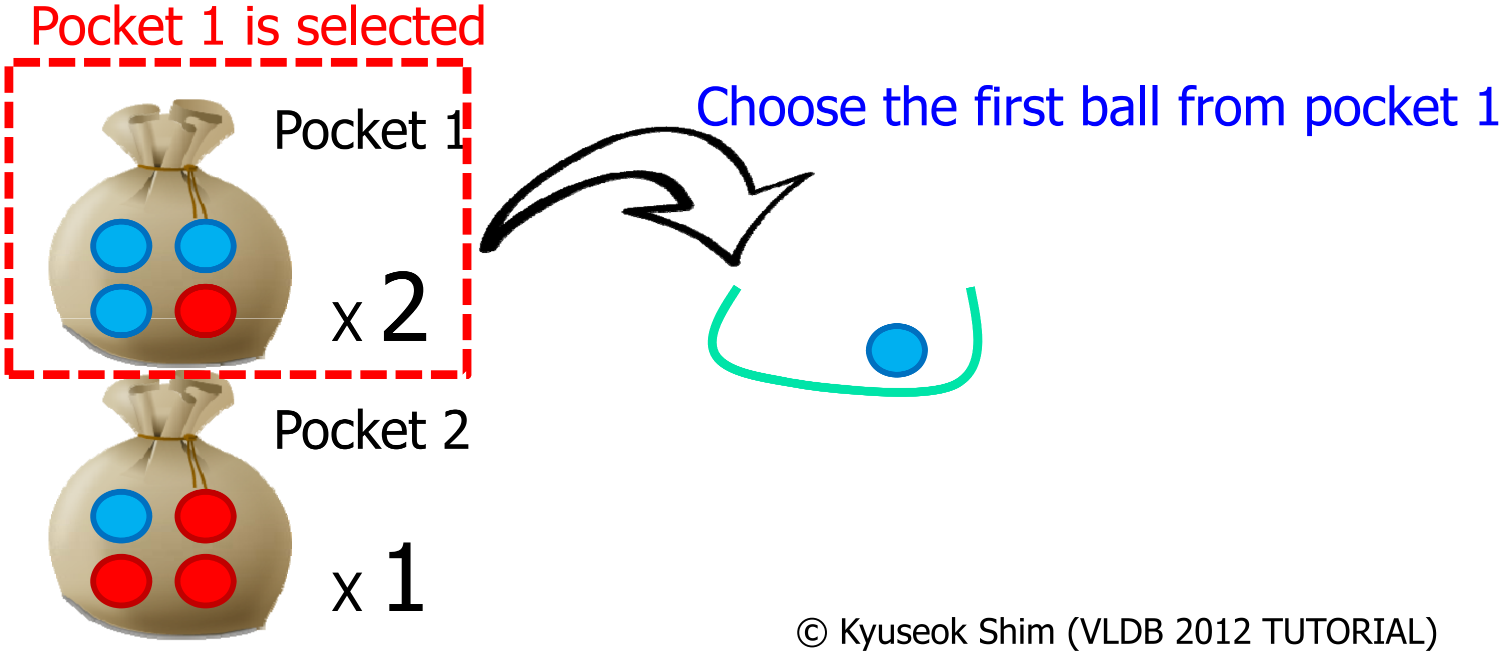


Cluster 1 in a grid (c_x, c_y) and Cluster 2 in (c_x', c_y') should be merged

Probabilistic Modeling using MapReduce

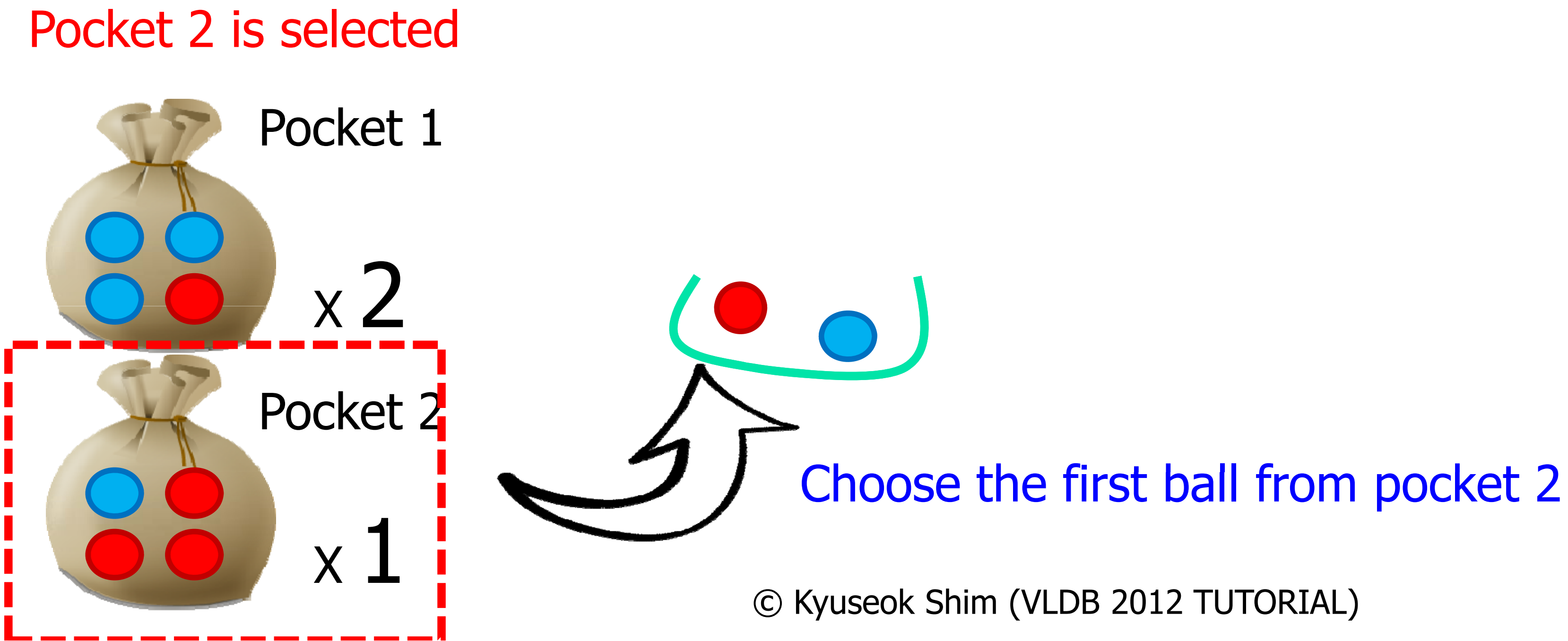
Probabilistic Modeling for Generating Documents

- Generative model
  - A model for randomly generating observable data, typically given some hidden parameters
  - e.g.) Select a pocket first and next draw a ball from the selected pocket with probability distributions



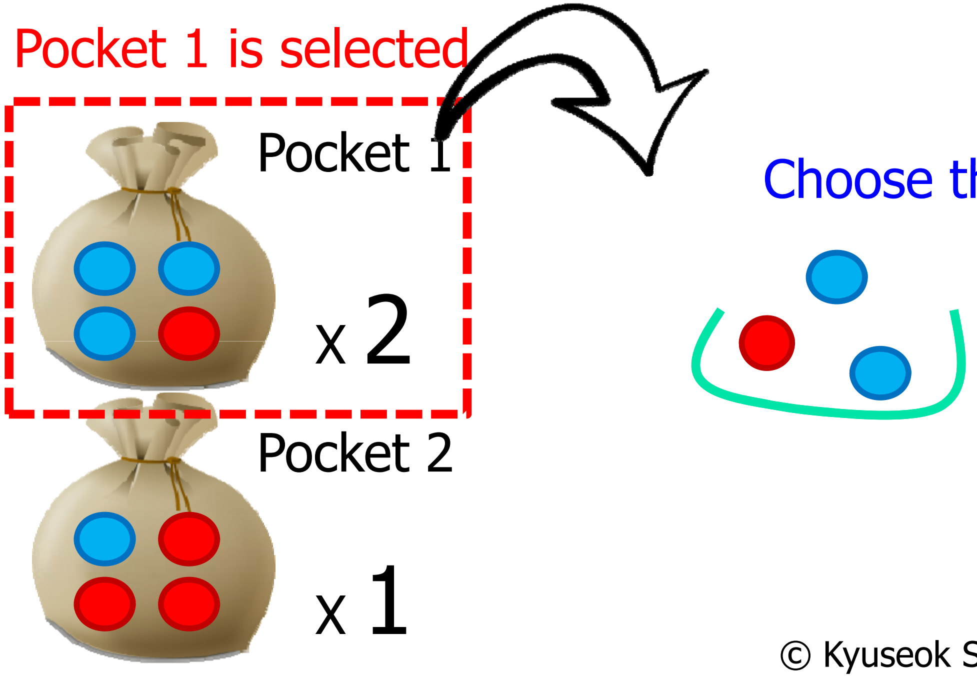
Probabilistic Modeling for Generating Documents

- Generative model
  - A model for randomly generating observable data, typically given some hidden parameters
  - e.g.) Select a pocket first and next draw a ball from the selected pocket with probability distributions



Probabilistic Modeling for Generating Documents

- Generative model
  - A model for randomly generating observable data, typically given some hidden parameters
  - e.g.) Select a pocket first and next draw a ball from the selected pocket with probability distributions

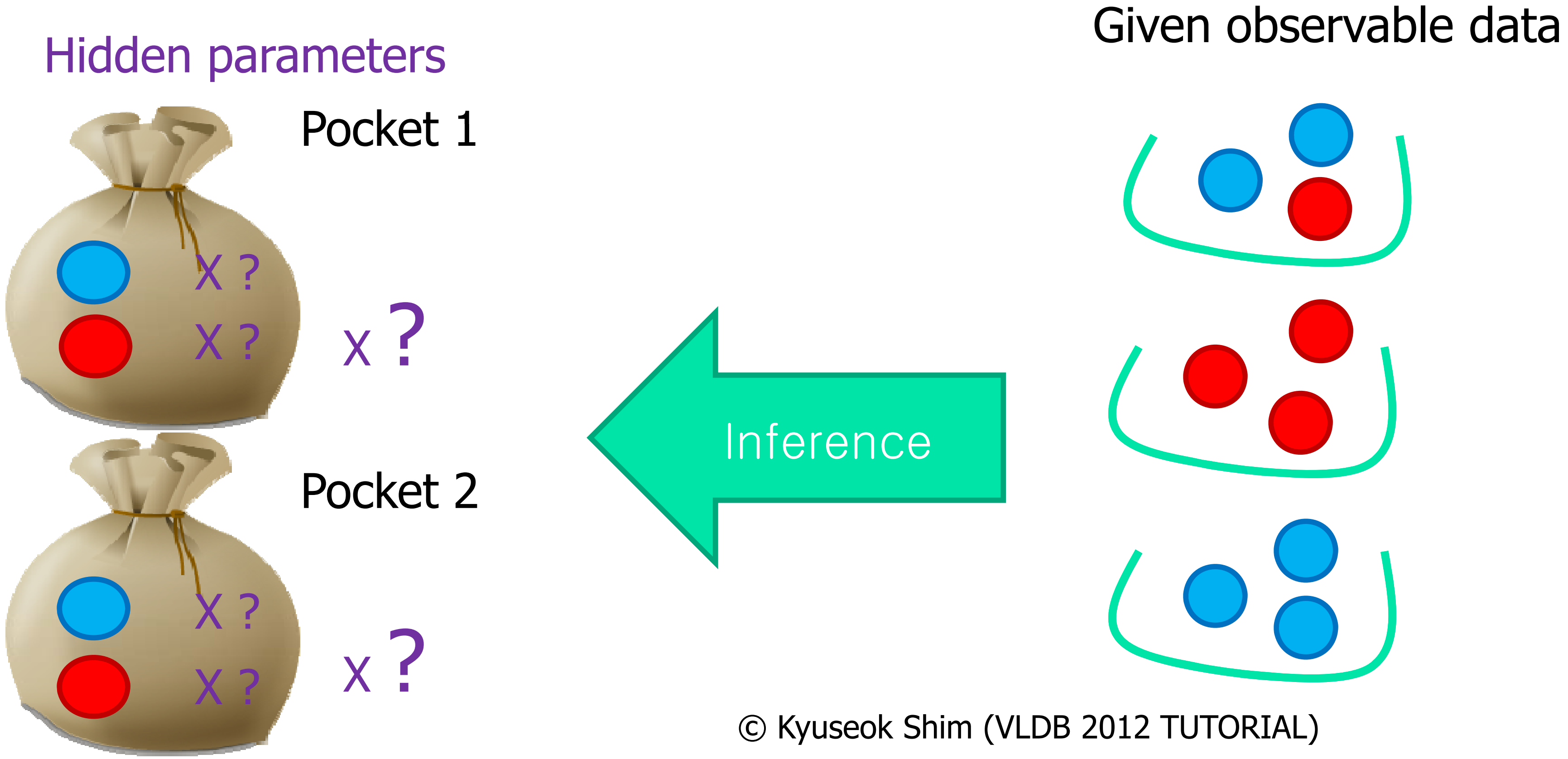


Pocket 1 is selected

Choose the first ball from pocket 1

Probabilistic Modeling for Generating Documents

- Generative model
  - A model for randomly generating observable data, typically given some hidden parameters



Hidden parameters

Pocket 1

X ?
X ?
x ?

Pocket 2

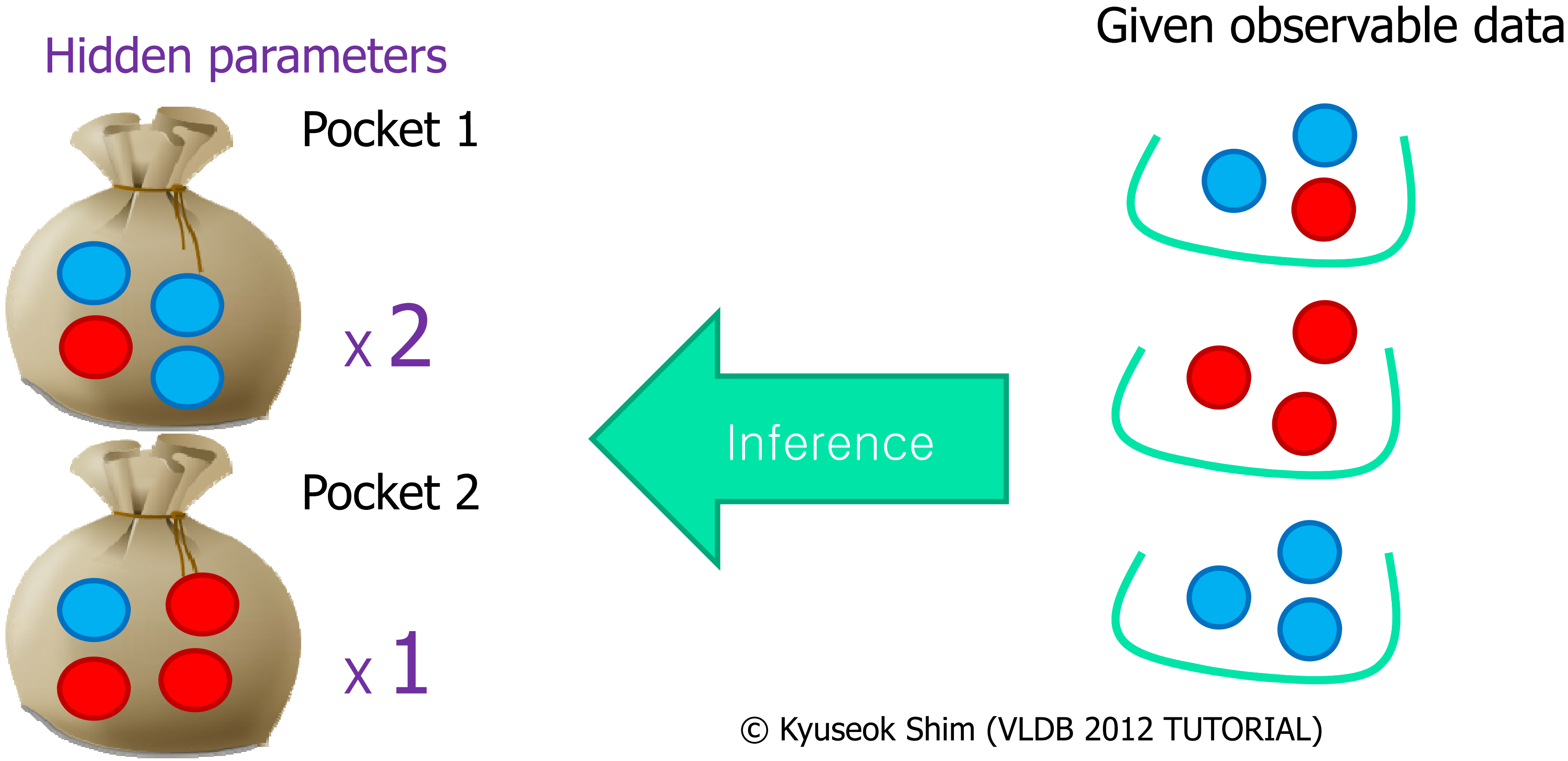
X ?
X ?
x ?

Inference

Given observable data

Probabilistic Modeling for Generating Documents

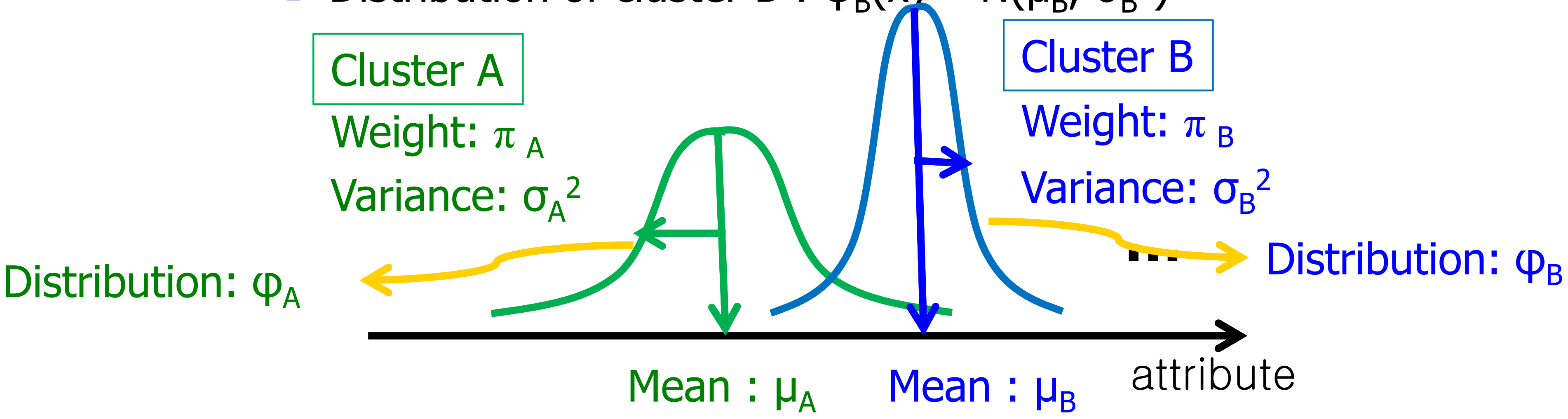
- Generative model
  - A model for randomly generating observable data, typically given some hidden parameters



EM-Clustering

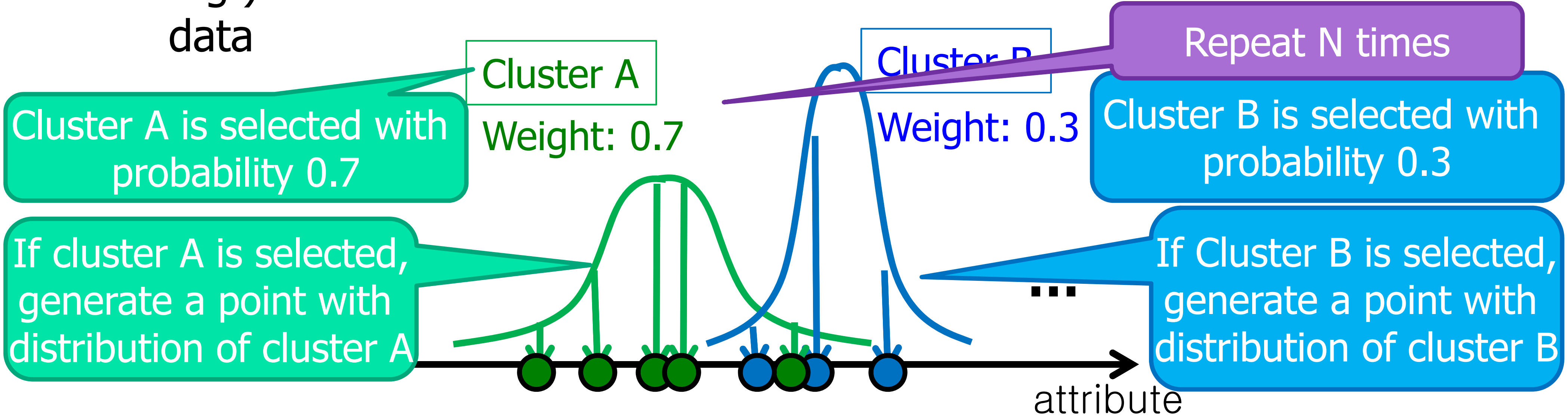
Gaussian Mixture Model

- Gaussian mixture: the weighted sum of k Gaussian probability distributions
  - Each Gaussian probability distribution represents a cluster
  - e.g.) Assume we have two clusters A and B for 1-dimensional data
    - Distribution of cluster A : $\varphi_A(x) \sim N(\mu_A, \sigma_A^2)$
    - Distribution of cluster B : $\varphi_B(x) \sim N(\mu_B, \sigma_B^2)$



The Generative Model

- Assume the data we have is sampled according to the generative model
- To generate each point in data
  - Select a cluster first following the weight distribution of the clusters
  - Generate a data point based on the distribution of points in the selected cluster
- e.g.) Assume we have two clusters A and B for 1-dimensional data



Repeat N times

Cluster A is selected with probability 0.7

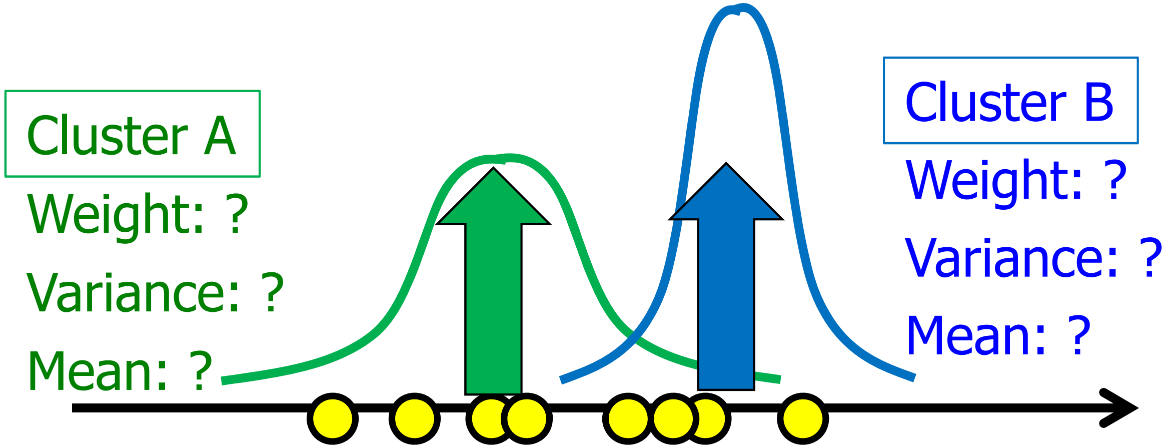
Cluster B is selected with probability 0.3

If cluster A is selected, generate a point with distribution of cluster A

If Cluster B is selected, generate a point with distribution of cluster B

Problem Definition

- Parameters to describe the clusters
  - π_i: weight of cluster i
  - μ_i: mean of cluster i
  - σ_i: variance of cluster i
- Given
  - Data points
  - K: the number of desired clusters
- Find the parameters which describe the given data points the best

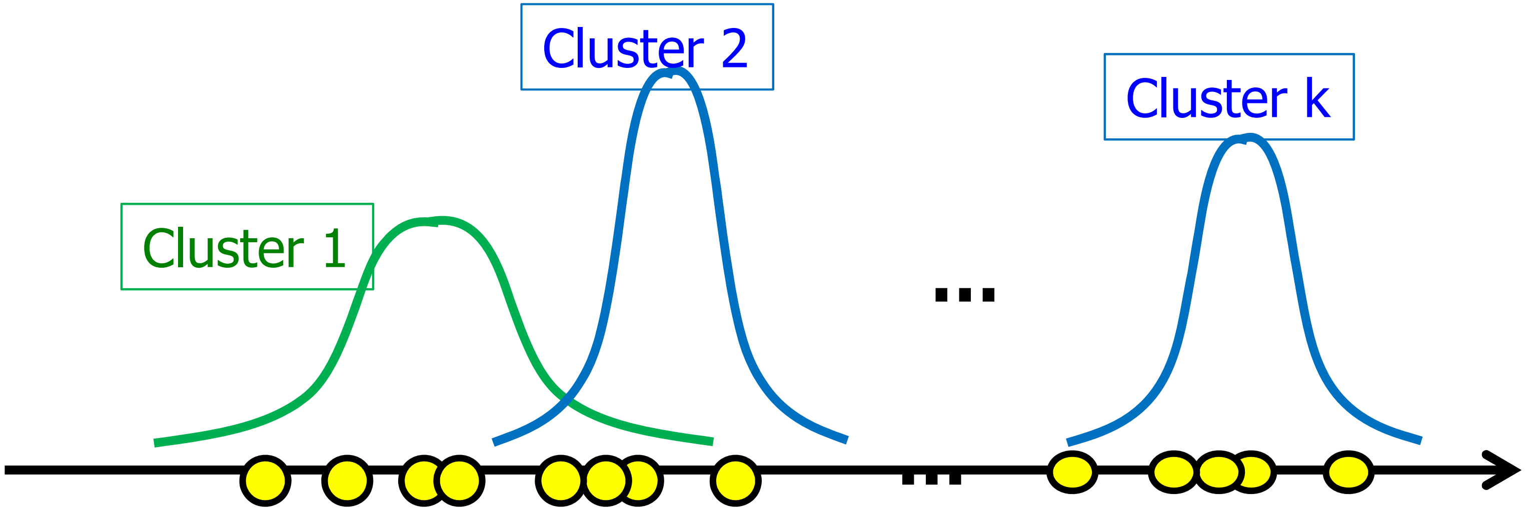


Cluster A
Weight: ?
Variance: ?
Mean: ?

Cluster B
Weight: ?
Variance: ?
Mean: ?

An Example of EM Clustering Algorithm

- Given data
  - $X=\{x_1,\ldots,x_N\}$: a set of N 1-dimensional points
  - k: the number of desired clusters



EM Clustering Algorithm

- Given
  - A data set $\{x_1, \ldots, x_n\}$
  - Assuming k Gaussian mixture
    - $\varphi_j(x) \sim N(\mu_j, \sigma_j)$ for $j=1,\ldots,k$
  - π_j: mixing weights for $j=1,\ldots,k$
    - Satisfying $\pi_1 + \ldots + \pi_k = 1$, $\pi_j \geq 0$
- A mixture density for a data point x
  - Probability of x to be generated from our k-Gaussian mixture model
  - $f_k(x) = \Sigma_j \pi_j \cdot \varphi_j(x)$
- Log likelihood
  - $\log \Pi_i f_k(x_i) = \Sigma_i \log f_k(x_i) = \Sigma_i \log \Sigma_j \pi_j \cdot \varphi_j(x)$

E-Step for EM Clustering Algorithm

- Compute expectation of hidden variables given observed variables
    - Hidden variable: c_j
    - Observed variables x_i

$$p(c_j \mid x_i) = \frac{\pi_j \phi_j (x_i)}{\sum_{l=1}^{k} \pi_l \phi_l (x_i)}$$

M-Step for EM Clustering Algorithm

- Maximize $\sum_{i=1}^{n} \log \sum_{j=1}^{k} \pi_j \phi_j(x_i)$
  - subject to $\sum_{j=1}^{k} \pi_j = 1$
- Lagrange function

$$L = \sum_{i=1}^{n} \log \sum_{j=1}^{k} \pi_j \phi_j(x_i) + \lambda_\pi (1 - \sum_{j=1}^{k} \pi_j)$$

M-Step for EM Clustering Algorithm

- Derivative for π_j

$$\frac{\partial L}{\partial \pi_j} = \frac{\partial}{\partial \pi_j}\left(\sum_{i=1}^{n}\log\sum_{l=1}^{k}\pi_l\phi_l(x_i) + \lambda_\pi(1-\sum_{u=1}^{k}\pi_u)\right)$$

(let $p(c_j \mid x_i) = \dfrac{\pi_j\phi_j(x_i)}{\sum_{l=1}^{k}\pi_l\phi_l(x_i)}$)

$$= \sum_{i=1}^{n}\frac{\phi_j(x_i)}{\sum_{l=1}^{k}\pi_l\phi_l(x_i)} - \lambda_\pi = \sum_{i=1}^{n}\frac{p(c_j \mid x_i)}{\pi_j} - \lambda_\pi = 0$$

$$(\log f(x))' = \frac{f'(x)}{f(x)}$$

$$\pi_j = \frac{\sum_{i=1}^{n} p(c_j \mid x_i)}{\lambda_\pi}$$

$$\lambda_\pi \pi_j = \sum_{i=1}^{n} p(c_j \mid x_i) \rightarrow \lambda_\pi \sum_{j=1}^{k}\pi_j = \sum_{j=1}^{k}\sum_{i=1}^{n} p(c_j \mid x_i)$$

$$\lambda_\pi = \sum_{j=1}^{k}\sum_{i=1}^{n} p(c_j \mid x_i) = \sum_{i=1}^{n}\sum_{j=1}^{k} p(c_j \mid x_i) = \sum_{i=1}^{n} 1 = n \quad (\because \sum_{j=1}^{k}\pi_j = 1)$$

$$\therefore \pi_j = \frac{1}{n}\sum_{i=1}^{n} p(c_j \mid x_i)$$

M-Step for EM Clustering Algorithm

- Derivative for μ_j

$$\frac{\partial L}{\partial \mu_j} = \frac{\partial}{\partial \mu_j}\left(\sum_{i=1}^{n} \log \sum_{l=1}^{k} \pi_l \phi_l(x_i) + \lambda_\pi (1 - \sum_{u=1}^{k} \pi_u) \right) = 0$$

$$\therefore \mu_j = \frac{\sum_{i=1}^{n} x_i p(c_j \mid x_i)}{\sum_{i=1}^{n} p(c_j \mid x_i)}$$

- Derivative for σ_j

$$\frac{\partial L}{\partial \sigma_j} = \frac{\partial}{\partial \sigma_j}\left(\sum_{i=1}^{n} \log \sum_{l=1}^{k} \pi_l \phi_l(x_i) + \lambda_\pi (1 - \sum_{u=1}^{k} \pi_u) \right) = 0$$

$$\therefore \sigma_j^2 = \frac{\sum_{i=1}^{n} (x_i - \mu_j)^2 p(c_j \mid x_i)}{\sum_{i=1}^{n} p(c_j \mid x_i)}$$

E-Step and M-Step of EM Clustering

- E-Step
  - Compute p(c_j | x_i) for every j=1,…,k and every x_i in {x_1,…,x_n}

$$p(c_j \mid x_i) = \frac{\pi_j \phi_j(x_i)}{\sum_{l=1}^{k} \pi_l \phi_l(x_i)}$$

- M-Step
  - Compute π_j, μ_j, σ_j for every j=1,…,k

$$\pi_j = \frac{1}{n}\sum_{i=1}^{n} p(c_j \mid x_i),\ \ \mu_j = \frac{\sum_{i=1}^{n} x_i p(c_j \mid x_i)}{\sum_{i=1}^{n} p(c_j \mid x_i)},\ \ \sigma_j^2 = \frac{\sum_{i=1}^{n} (x_i - \mu_j)^2 p(c_j \mid x_i)}{\sum_{i=1}^{n} p(c_j \mid x_i)}$$

Serial Algorithm for EM Clustering Algorithm

- Do
  - E step
    - For each point x_i
      - $\boxed{p(c_j|x_i)} = \pi_j \cdot \varphi_j(x_i) \,/\, \Sigma_k \pi_k \cdot \varphi_k(x_i)$ where c_j is hidden variable
  - M step
    - For each cluster j
      - $\pi_j = \Sigma_i \boxed{p(c_j|x_i)} \,/\, n$
      - $\mu_j = \Sigma_i \boxed{x_i \cdot p(c_j|x_i)} / \Sigma_i \boxed{p(c_j|x_i)}$
      - $\sigma_j^2 = \Sigma_i \boxed{(x_i - \mu_j)^2 \cdot p(c_j|x_i)} \,/\, \Sigma_i \boxed{p(c_j|x_i)}$
- until convergence

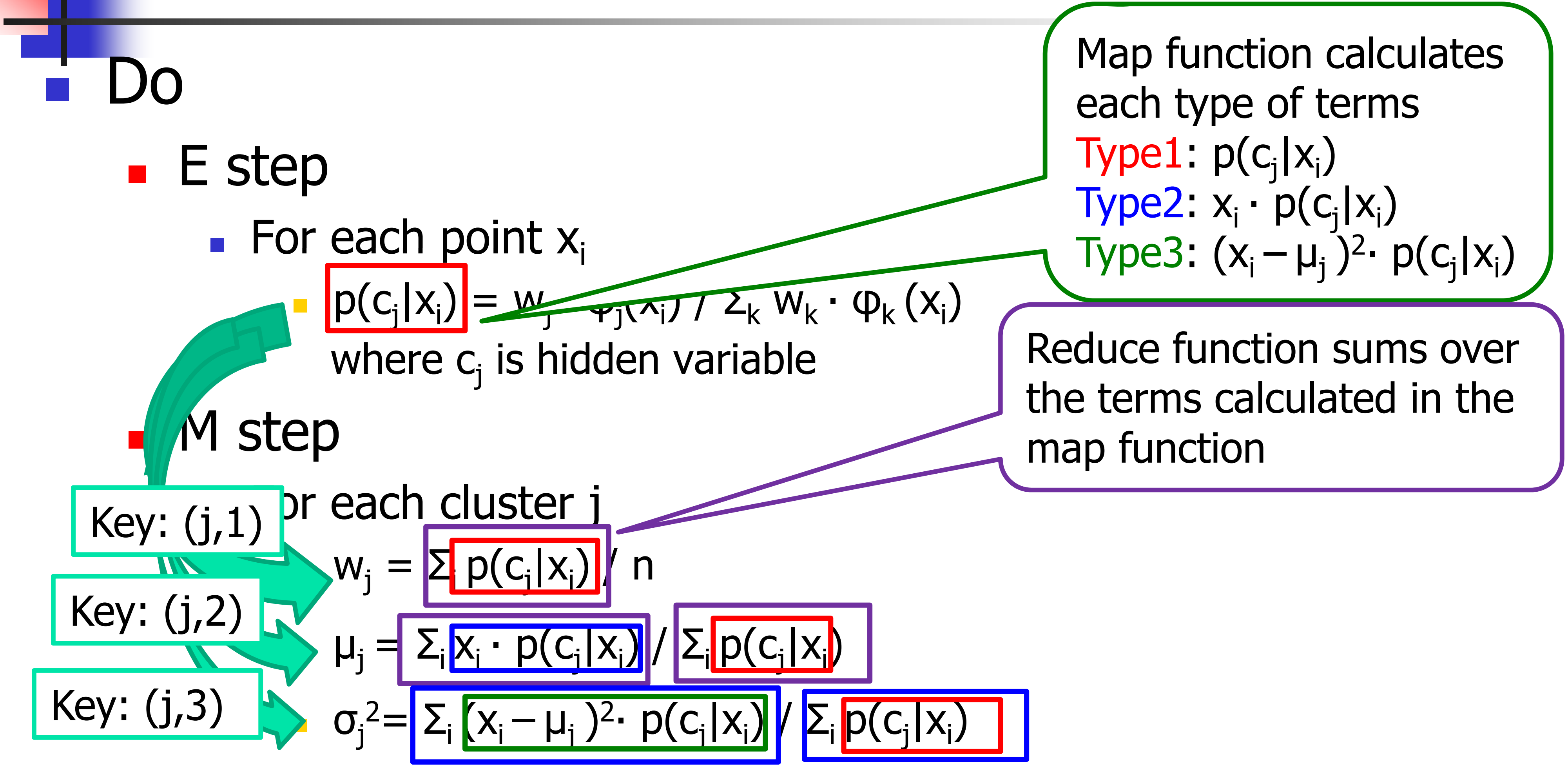
Used in the M-step for three equations

Type1: $p(c_j|x_i)$

Type2: $x_i \cdot p(c_j|x_i)$

Type3: $(x_i - \mu_j)^2 \cdot p(c_j|x_i)$

MapReduce Algorithm for EM Clustering Algorithm



- Do
  - E step
    - For each point x_i
      - $\boxed{p(c_j|x_i)} = w_j \cdot \varphi_j(x_i) / \Sigma_k w_k \cdot \varphi_k (x_i)$ where c_j is hidden variable
  - M step
    - For each cluster j
      - $w_j = \boxed{\Sigma_i \boxed{p(c_j|x_i)}} / n$
      - $\mu_j = \boxed{\Sigma_i \boxed{x_i \cdot p(c_j|x_i)}} / \boxed{\Sigma_i \boxed{p(c_j|x_i)}}$
      - $\sigma_j^2 = \boxed{\Sigma_i \boxed{(x_i - \mu_j)^2 \cdot p(c_j|x_i)}} / \boxed{\Sigma_i \boxed{p(c_j|x_i)}}$
- until convergence

Key: (j,1)

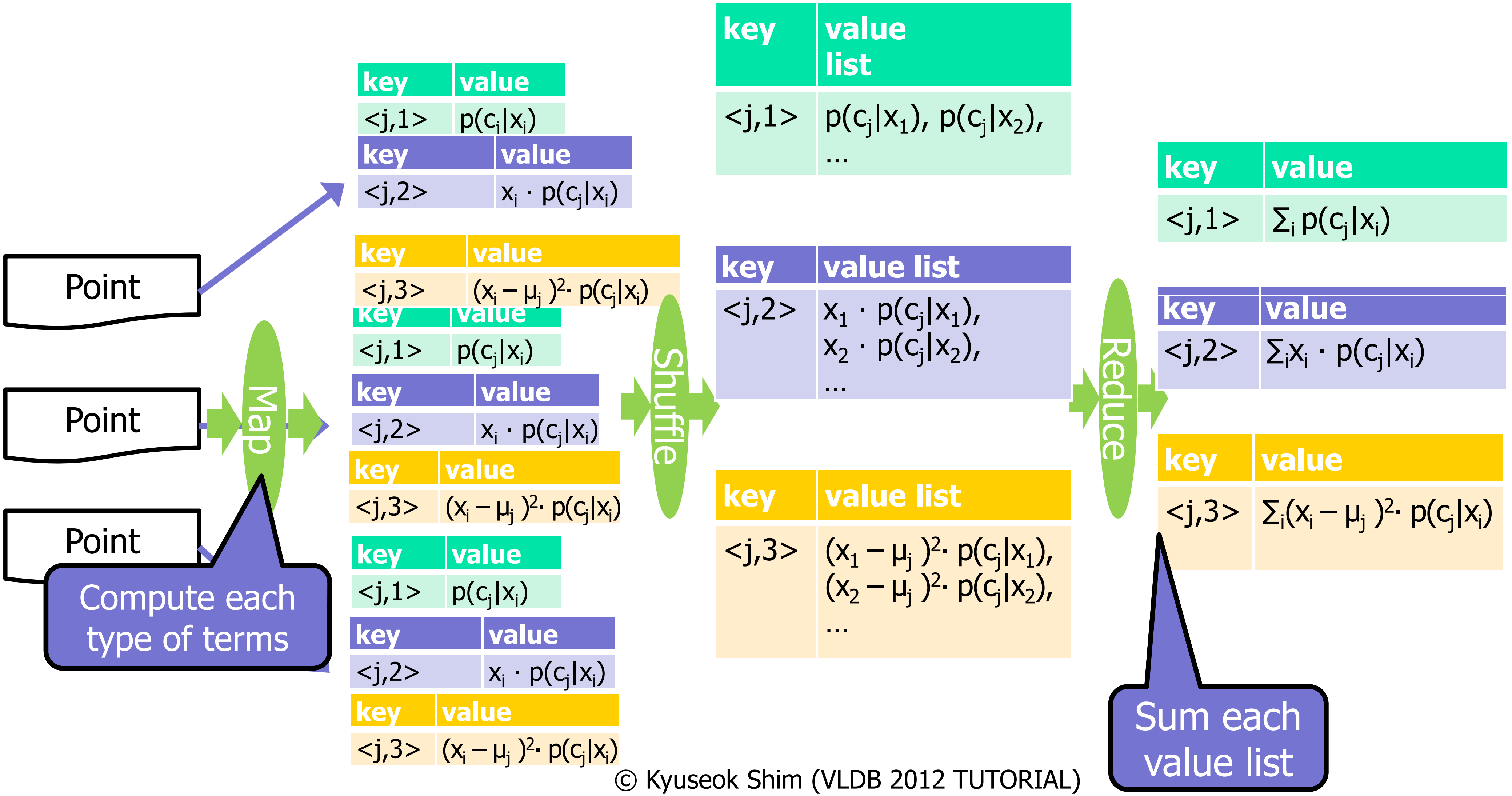
Key: (j,2)

Key: (j,3)

Map function calculates each type of terms
Type1: $p(c_j|x_i)$
Type2: $x_i \cdot p(c_j|x_i)$
Type3: $(x_i - \mu_j)^2 \cdot p(c_j|x_i)$

Reduce function sums over the terms calculated in the map function

An Illustration of EM Clustering



key	value
<j,1>	$p(c_j\|x_i)$

key	value
<j,2>	$x_i \cdot p(c_j\|x_i)$

key	value
<j,3>	$(x_i - \mu_j)^2 \cdot p(c_j\|x_i)$

key	value
<j,1>	$p(c_j\|x_i)$

key	value
<j,2>	$x_i \cdot p(c_j\|x_i)$

key	value
<j,3>	$(x_i - \mu_j)^2 \cdot p(c_j\|x_i)$

key	value
<j,1>	$p(c_j\|x_i)$

key	value
<j,2>	$x_i \cdot p(c_j\|x_i)$

key	value
<j,3>	$(x_i - \mu_j)^2 \cdot p(c_j\|x_i)$

Compute each type of terms

key	value list
<j,1>	$p(c_j\|x_1)$, $p(c_j\|x_2)$, …

key	value list
<j,2>	$x_1 \cdot p(c_j\|x_1)$, $x_2 \cdot p(c_j\|x_2)$, …

key	value list
<j,3>	$(x_1 - \mu_j)^2 \cdot p(c_j\|x_1)$, $(x_2 - \mu_j)^2 \cdot p(c_j\|x_2)$, …

key	value
<j,1>	$\Sigma_i\, p(c_j\|x_i)$

key	value
<j,2>	$\Sigma_i x_i \cdot p(c_j\|x_i)$

key	value
<j,3>	$\Sigma_i (x_i - \mu_j)^2 \cdot p(c_j\|x_i)$

Sum each value list

EM Clustering Algorithm for Multidimensional Points

- For d-dimensional data, the parameters to describe *k* Gaussian distributions are
  - k means $\mu_1, \mu_2, \ldots, \mu_k$
  - k covariance matrices $\Sigma_1, \Sigma_2, \ldots, \Sigma_k$
- The i-th d-dimensional Gaussian pdf is

$$f_i(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma_i|^{1/2}} \exp(-\frac{1}{2}(\mathbf{x}-\mu_i)^T \Sigma_i^{-1}(\mathbf{x}-\mu_i))$$

- We can derive EM steps similarly as in 1-dimensional case

EM Clustering Algorithm for Multidimensional Points

- Given d-dimensional data
  - $x_i = (x_{i1}, \ldots, x_{id}), 1 \le i \le m$
- Symbols for *k* Gaussian distributions
  - $y = \{ c_1, \ldots, c_k \}$
- Initialize parameters
  - $\mu_1, \ldots, \mu_k$: d-dimensional means of k Gaussian distributions
  - $\Sigma_1, \ldots, \Sigma_k$: d × d covariance matrices of k Gaussian distributions
  - $\pi_1, \ldots, \pi_k$: prioris for each Gaussian distribution

M Steps for Map/Reduce

$$\pi_j = \frac{1}{n}\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)}$$

A(j)

$$\mu_{j1} = \frac{\boxed{\sum_{i=1}^{n} x_{i1} P(c_j \mid x_i)}}{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)}}$$

$B_1(j)$

$$\mu_{jd} = \frac{\boxed{\sum_{i=1}^{n} x_{id} P(c_j \mid x_i)}}{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)}}$$

$B_d(j)$

$$\begin{pmatrix} \frac{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)(x_{i1}-\mu_{j1})^2}}{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)}} & \cdots & \frac{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)(x_{i1}-\mu_{j1})(x_{id}-\mu_{jd})}}{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)}} \\ \vdots & & \vdots \\ \frac{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)(x_{id}-\mu_{jd})(x_{i1}-\mu_{j1})}}{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)}} & \cdots & \frac{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)(x_{id}-\mu_{jd})^2}}{\boxed{\sum_{i=1}^{n} P(c_j \mid x_i)}} \end{pmatrix}$$

$C_{11}(j)$ $C_{1d}(j)$

$C_{d1}(j)$ $C_{dd}(j)$

Probabilistic Latent Semantic Indexing (PLSI) using MapReduce

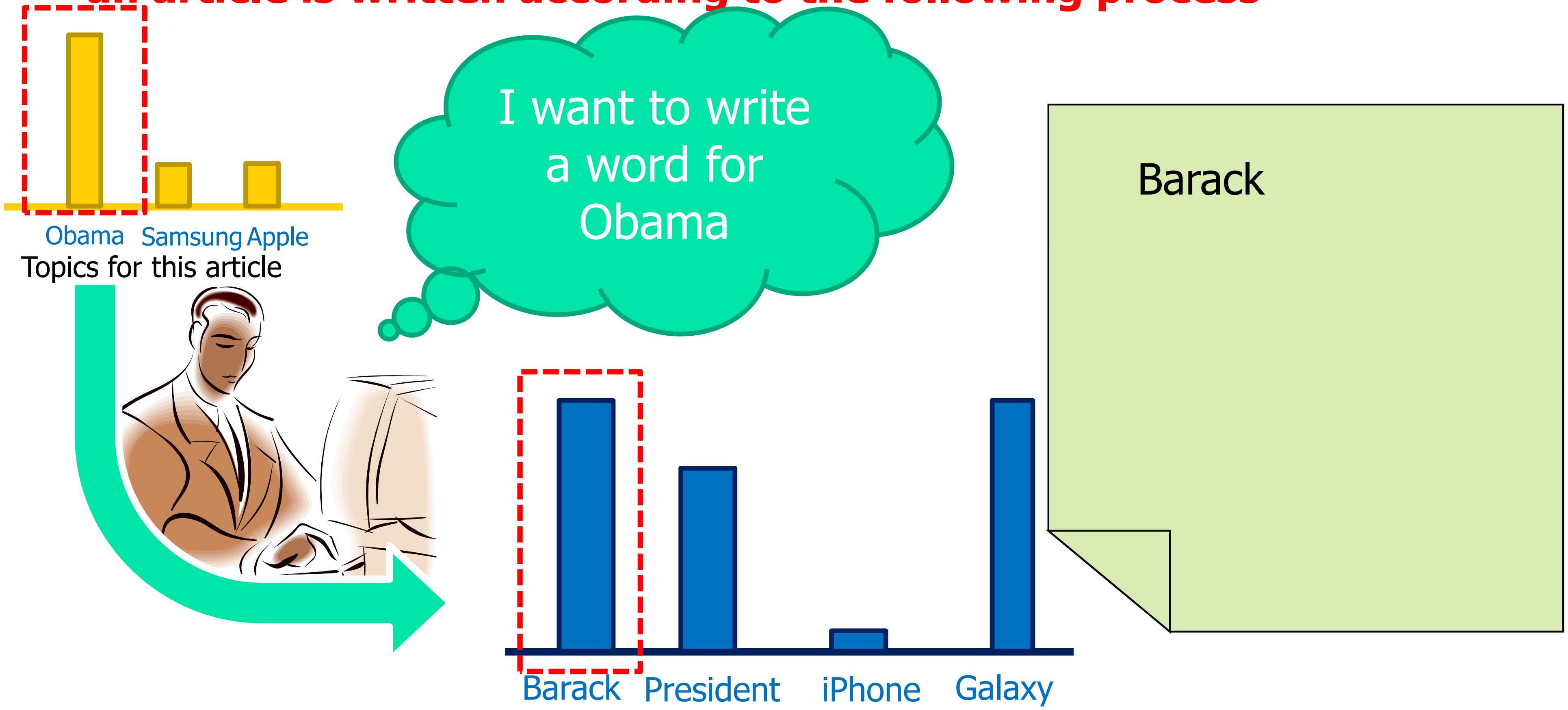
Generative Model Illustration

The generative model of PSLI assumes that
an article is written according to the following process

Generative Model Illustration

The generative model of PSLI assumes that

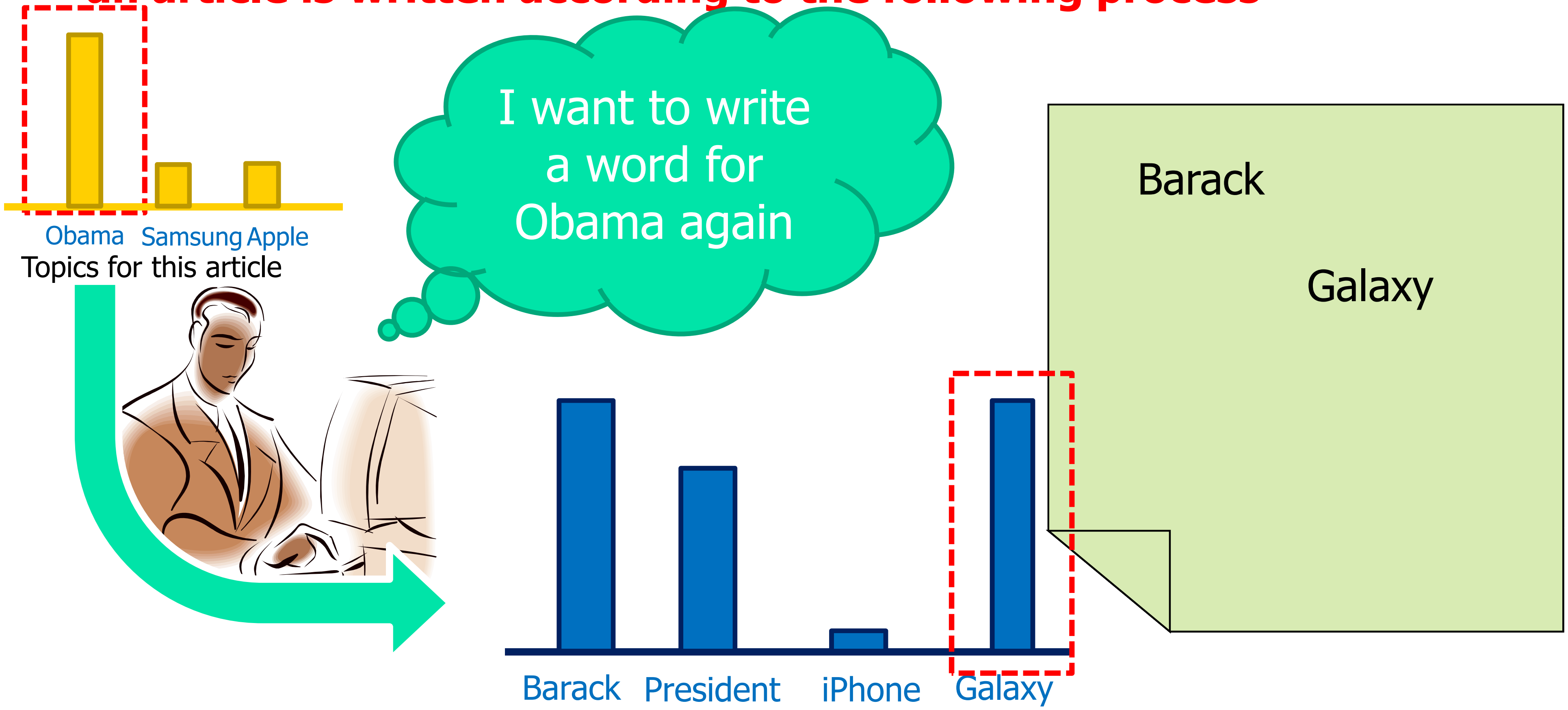
an article is written according to the following process



"Probabilities of Words for the topic for Barack Obama"

Generative Model Illustration

The generative model of PSLI assumes that
an article is written according to the following process

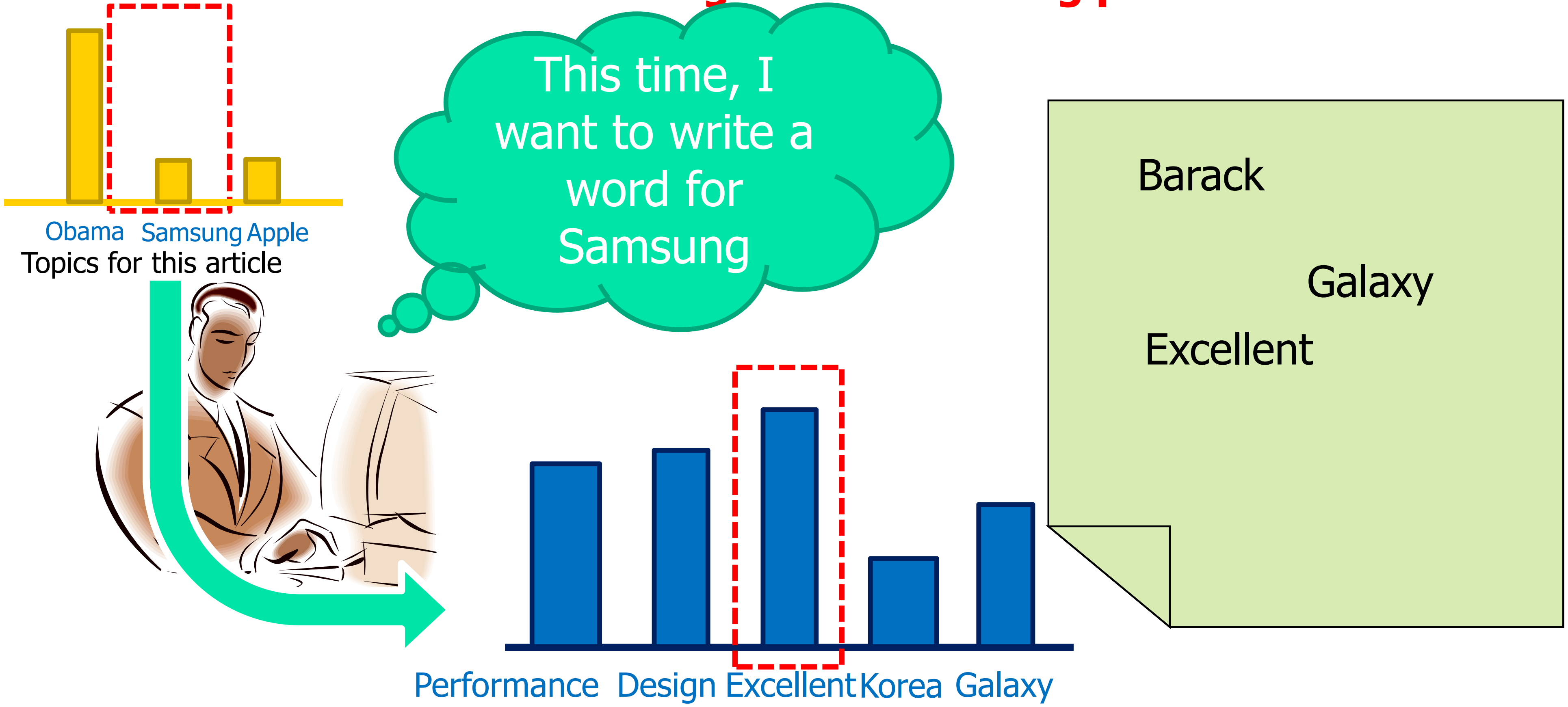


Topics for this article

“Probabilities over words for the topic of Obama”

Generative Model Illustration

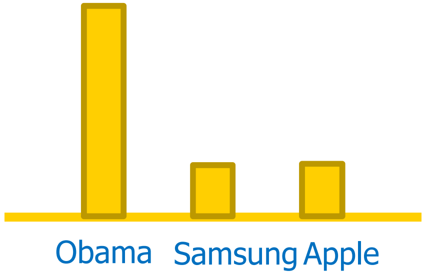
The generative model of PSLI assumes that
an article is written according to the following process



“Probabilities over words for the topic of Samsung”

Generative Model Illustration

Choose words i.i.d. following to the probability distribution

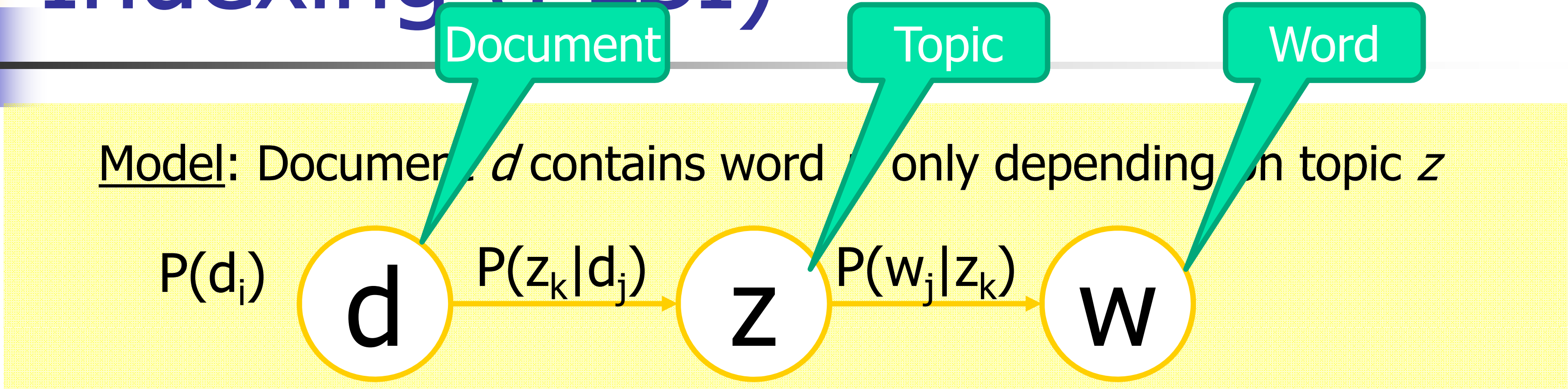


Topics for this article

Generate a document

Barack
Good
Excellent
Galaxy
Galaxy
Design
Performance

Probabilistic Latent Semantic Indexing (PLSI)



Model: Documen *d* contains word only depending n topic *z*

- Document consists of topics and words in the document generated based on those topics
  - d_i: the i-th document (observable)
  - z_k: the k-th latent topic (unobservable)
  - w_j: the j-th word (observable)
- Generate model: (d_i, w_j) is generated as follows:
  - Pick a document d_i with probability $P(d_i)$
  - Pick a topic z_k with probability $P(z_k|d_j)$
  - Generate a word w_j with probability $P(w_j|z_k)$

Likelihood Function for EM Algorithm

- Find parameters which maximize the log-likelihood,

$$L = \log \prod_{d \in D} \prod_{w \in W} p(d,w)^{n(d,w)} = \sum_{d \in D} \sum_{w \in W} n(d,w) \log p(d,w)$$

  - where

$$p(d,w) = \sum_{z \in Z} p(d) p(z \mid d) p(w \mid z)$$

Serial EM Algorithm

- Do
  - E-step

$$\boxed{P(z|d,w)} = \frac{P(z|d)p(w|z)}{\sum_{z'} P(z'|d)p(w|z')}$$

  - M-step

$$P(w|z) = \frac{\sum_{d} n(d,w)\boxed{P(z|d,w)}}{\sum_{d,w'} n(d,w')\boxed{P(z|d,w')}}$$

$$P(d|z) = \frac{\sum_{w} n(d,w)\boxed{P(z|d,w)}}{\sum_{d',w} n(d',w)\boxed{P(z|d',w)}}$$

$$P(z) = \frac{1}{R}\sum_{d,w} n(d,w)\boxed{P(z|d,w)},\ R \equiv \sum_{d,w} n(d,w)$$

- until convergence

MapReduce EM Algorithm

- [Das, Datar, Garg, Rajaram: WWW 2007]
- Do
  - E-ste

$$\boxed{P(z|d,w)}=\frac{P(z|d)p(w|z)}{\sum_{z'}P(z'|d)p(w|z')}$$

Calculate in map

  - Key: (w,z)

$$P(w|z)=\frac{\sum_{d}n(d,w)\boxed{P(z|d,w)}}{\sum_{d,w'}n(d,w')\boxed{P(z|d,w')}}$$

Summarize in reduce

Key: (d,z)

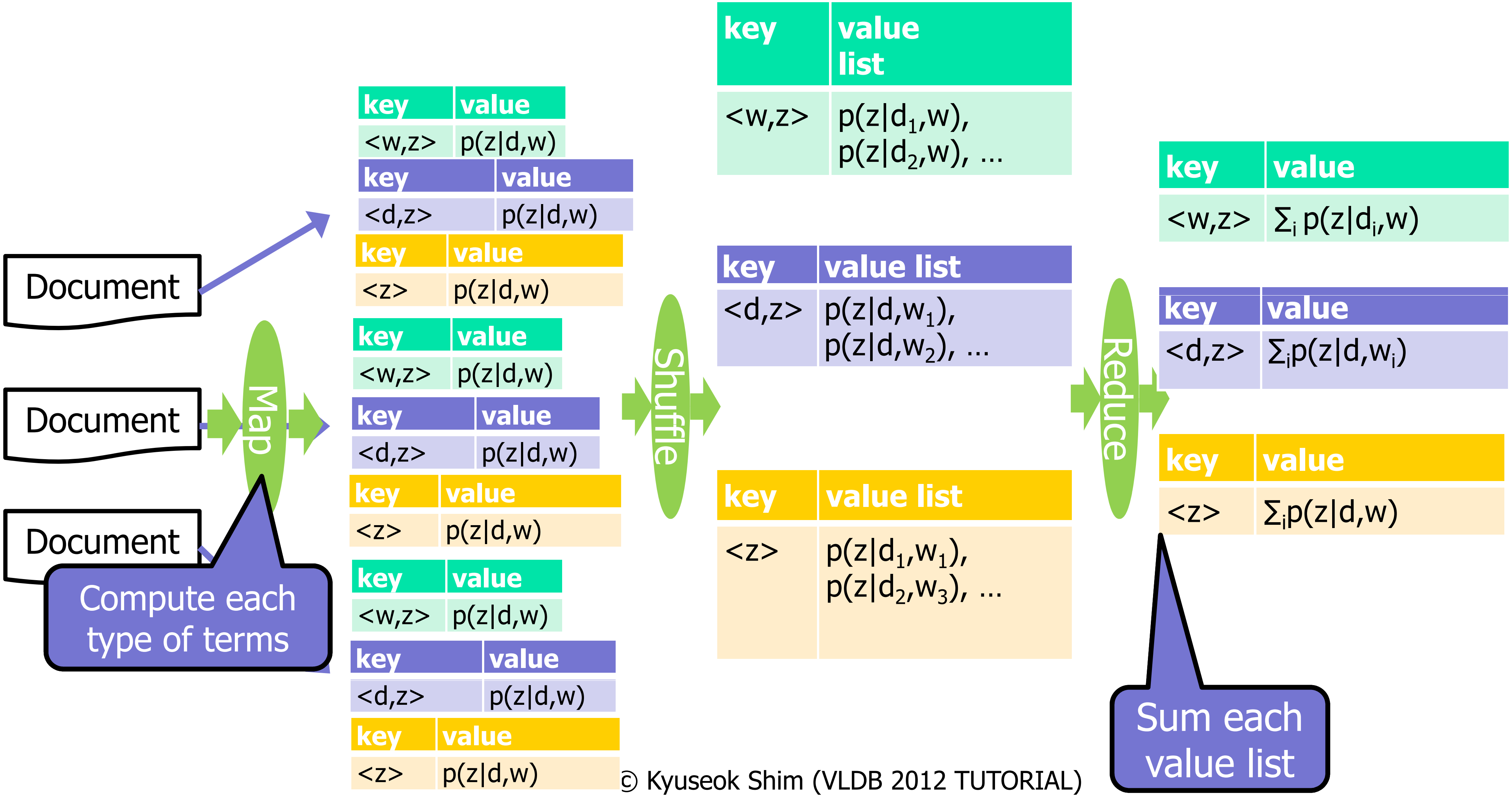
$$P(d|z)=\frac{\sum_{w}n(d,w)\boxed{P(z|d,w)}}{\sum_{d',w}n(d',w)\boxed{P(z|d',w)}}$$

Key: z

$$P(z)=\frac{1}{R}\sum_{d,w}n(d,w)\boxed{P(z|d,w)},\ R\equiv\sum_{d,w}n(d,w)$$

- Until convergence

An Illustration of PLSI



key	value
<w,z>	p(z|d,w)

key	value
<d,z>	p(z|d,w)

key	value
<z>	p(z|d,w)

key	value
<w,z>	p(z|d,w)

key	value
<d,z>	p(z|d,w)

key	value
<z>	p(z|d,w)

key	value
<w,z>	p(z|d,w)

key	value
<d,z>	p(z|d,w)

key	value
<z>	p(z|d,w)

key	value list
<w,z>	$p(z\|d_1,w)$, $p(z\|d_2,w)$, …

key	value list
<d,z>	$p(z\|d,w_1)$, $p(z\|d,w_2)$, …

key	value list
<z>	$p(z\|d_1,w_1)$, $p(z\|d_2,w_3)$, …

key	value
<w,z>	$\Sigma_i\, p(z\|d_i,w)$

key	value
<d,z>	$\Sigma_i p(z\|d,w_i)$

key	value
<z>	$\Sigma_i p(z\|d,w)$

Compute each type of terms

Sum each value list

Model Parameter Estimation for other Models using MapReduce

- Latent Dirichlet Allocation (LDA)
  - [Zhai, Boyd-Graber, Asadi, Alkhouja: WWW 2012]
    - Utilize Variational EM algorithm
  - [Wang, Bai, Stanton, Chen, Chang: AAIM 2009]
    - Utilize Gibbs sampling
- A Hidden Markov Model
  - [Cao, Jiang, Pei, Chen, Li: WWW 2009]

The Generative Model of LDA

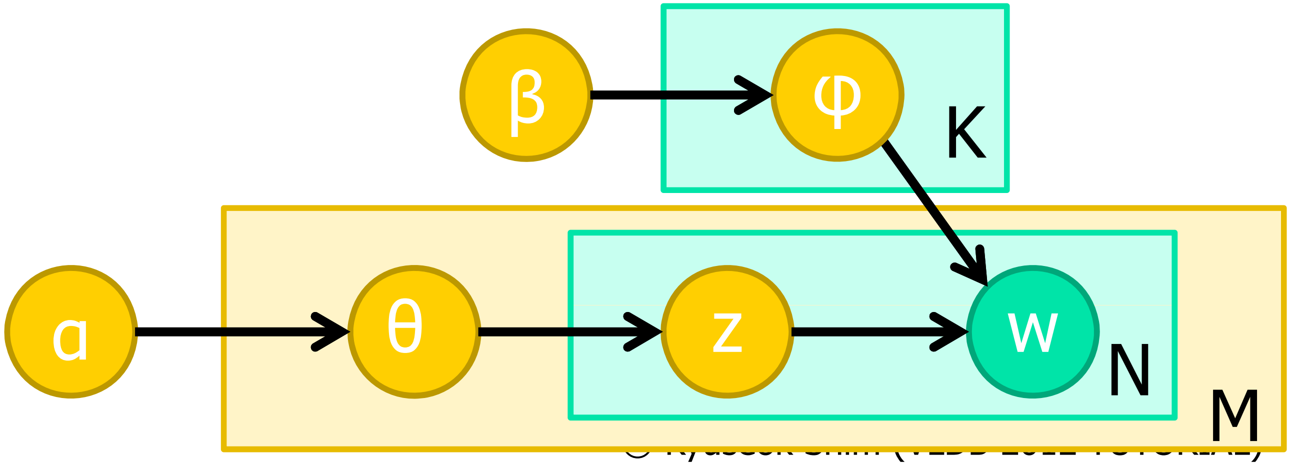
- For each topic k
  - Choose $\varphi_k \sim Dir(\beta)$
- For each document $\mathbf{w}_d$
  - Choose $\theta_d \sim Dir(\alpha)$
  - For each words w_n in $\mathbf{w}_d$
    - Choose a topic $z_{d,n} \sim Mult(\theta_d)$
    - Choose a word $w_{d,n} \sim p(w_{d,n}|z_{d,n},\varphi_k)$

Dirichlet distribution

φ_k is a vector of probabilities that each word is selected from the topic k

θ_d is a topic distribution in a document d

Multinomial distribution



K: number of topics
N: number of words in a document
M: number of documents

Problem Definition

- Given
  - A document collection D = {$\mathbf{w}_1, \ldots, \mathbf{w}_M$}
    - Each document $\mathbf{w}_d$ is represented as a term frequency sequence:
      $\mathbf{w}_d = (w_1^{(d)}, \ldots, w_v^{(d)}, \ldots, w_V^{(d)})$

Total number of distinct words is V

$w_v^{(d)}$ is the number of frequency for a word indexed by v

- Find
  - A model parameter Θ maximizing the likelihood p(D|Θ)
- We may use the following inference algorithms
  - Variational EM algorithm
  - Gibbs sampling

Serial Variational EM Algorithm

```
do
    Initialize $\gamma_{d,k} = \alpha_k$, $\lambda_{v,k} = \beta_v$
    For d=1 to M (for every document)
        For v=1 to V (for every word)
            For k=1 to K (for every topic)
                Compute $\Phi_{v,k}^{(d)} = \lambda_{v,k}/\Sigma_v \lambda_{v,k} \cdot \exp(\Psi(\gamma_{d,k}))$
            Normalize $\Phi_v^{(d)}$
            For k=1 to K
                Compute $\gamma_{d,k} = \gamma_{d,k} + w_v^{(d)} \cdot \Phi_{v,k}^{(d)}$
    For v=1 to V
        For k=1 to K
            Compute $\lambda_{v,k} = \lambda_{v,k} + \Sigma_d w_v^{(d)} \cdot \Phi_{v,k}^{(d)}$
    Compute $\alpha_k$
Until convergence
```

Iteratively find only α; β is fixed

$\Psi(x)=d/dx(\log\Gamma(x))$

Mr.LDA: The EM Algorithm for LDA Using MapReduce

- [Zhai, Boyd-Graber, Asadi, Alkhouja: WWW 2012]

```
do
    Initialize $\gamma_{d,k} = \alpha_k$, $\lambda_{v,k} = \beta_v$
    For d=1 to M (for every document)
        For v=1 to V (for every word)
            For k=1 to K (for every topic)
                Compute $\Phi_{d,v,k} = \lambda_{v,k}/\Sigma_v \lambda_{v,k} \cdot \exp(\Psi(\gamma_{d,k}))$
            Normalize $\Phi_{d,v}$
            For k=1 to K
                Compute $\gamma_{d,k} = \gamma_{d,k} + w_{d,v} \cdot \Phi_{d,v,k}$
    For v=1 to V
        For k=1 to K
            Compute $\lambda_{v,k} = \lambda_{v,k} + \Sigma_d w_{d,v} \cdot \Phi_{d,v,k}$
    Compute $\alpha_k$
Until convergence
```

A map function get a document as input

$\Psi(x)=d/dx(\log\Gamma(x))$

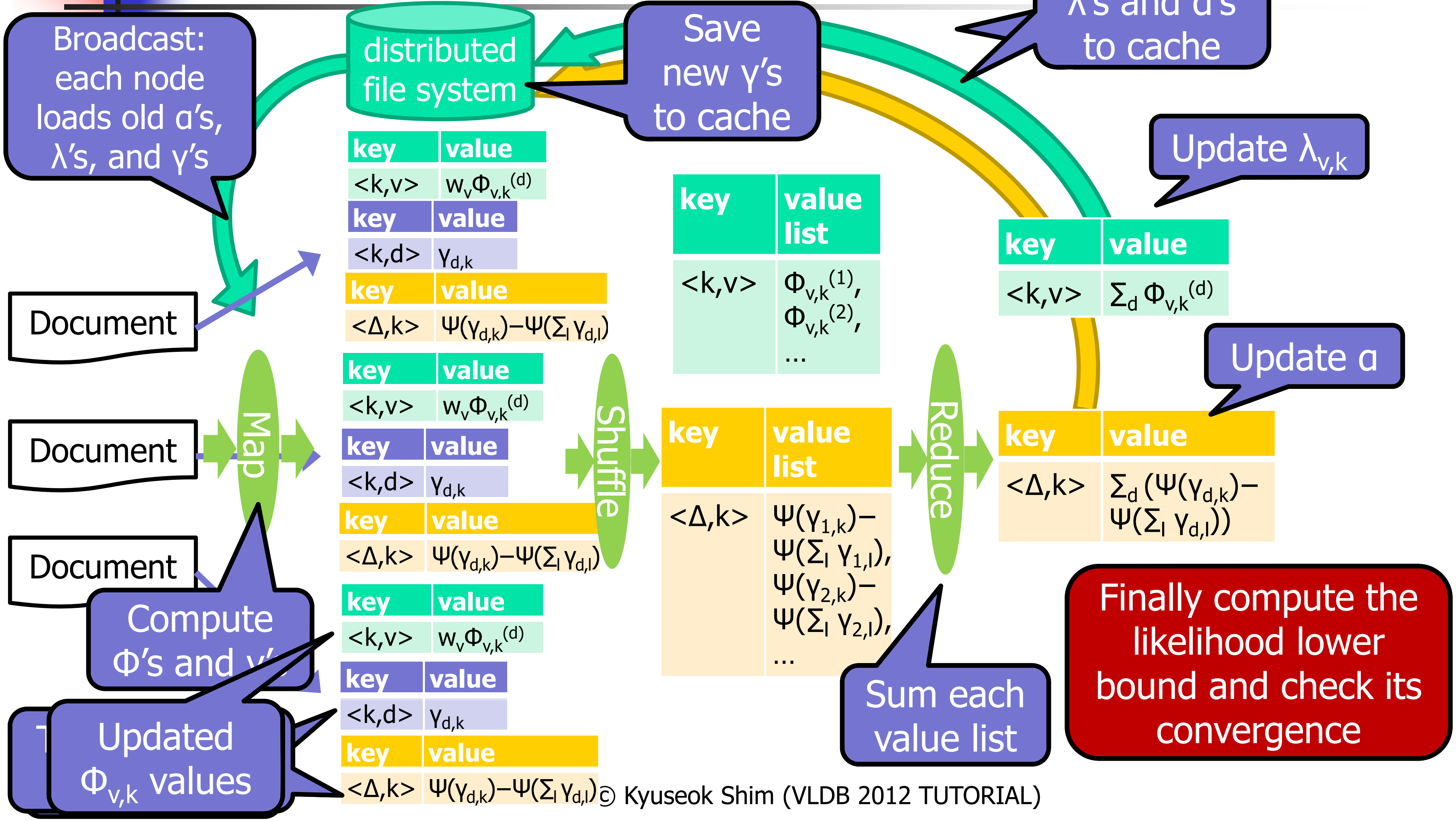
Compute document-specific parameters Φ's and γ's in map functions

Compute topic-specific parameter λ's in reduce functions

Main Function

- For each iteration
  - Broadcast α's, γ's and λ's to every machine
  - Call map and reduce functions
    - Φ's and γ's are computed
  - Update α
  - Compute the likelihood lower bound
- Determine whether the lower bound of the likelihood has converged

An Illustration of MR. LDA



Broadcast: each node loads old α's, λ's, and γ's

Save new γ's to cache

Save new λ's and α's to cache

Update $\lambda_{v,k}$

Update α

Compute Φ's and γ's

Updated $\Phi_{v,k}$ values

Sum each value list

Finally compute the likelihood lower bound and check its convergence

key	value
<k,v>	$w_v\Phi_{v,k}^{(d)}$

key	value
<k,d>	$\gamma_{d,k}$

key	value
<Δ,k>	$\Psi(\gamma_{d,k})-\Psi(\Sigma_l \gamma_{d,l})$

key	value
<k,v>	$w_v\Phi_{v,k}^{(d)}$

key	value
<k,d>	$\gamma_{d,k}$

key	value
<Δ,k>	$\Psi(\gamma_{d,k})-\Psi(\Sigma_l \gamma_{d,l})$

key	value
<k,v>	$w_v\Phi_{v,k}^{(d)}$

key	value
<k,d>	$\gamma_{d,k}$

key	value
<Δ,k>	$\Psi(\gamma_{d,k})-\Psi(\Sigma_l \gamma_{d,l})$

key	value list
<k,v>	$\Phi_{v,k}^{(1)}$, $\Phi_{v,k}^{(2)}$, …

key	value list
<Δ,k>	$\Psi(\gamma_{1,k})-\Psi(\Sigma_l \gamma_{1,l})$, $\Psi(\gamma_{2,k})-\Psi(\Sigma_l \gamma_{2,l})$, …

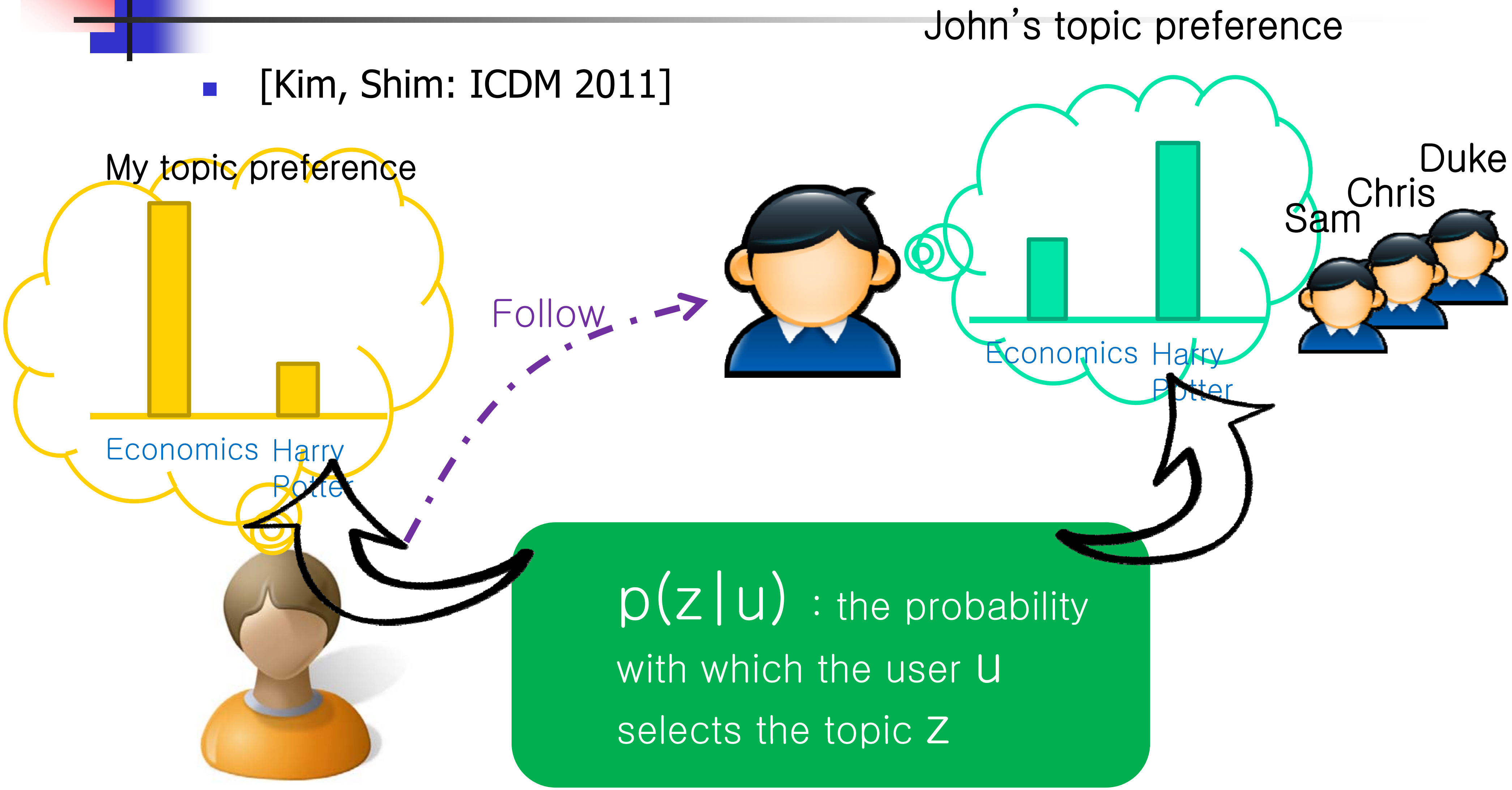
key	value
<k,v>	$\Sigma_d\, \Phi_{v,k}^{(d)}$

key	value
<Δ,k>	$\Sigma_d\, (\Psi(\gamma_{d,k})-\Psi(\Sigma_l \gamma_{d,l}))$

Probabilistic Modeling for Twitter using MapReduce

Our Generative Model

- [Kim, Shim: ICDM 2011]

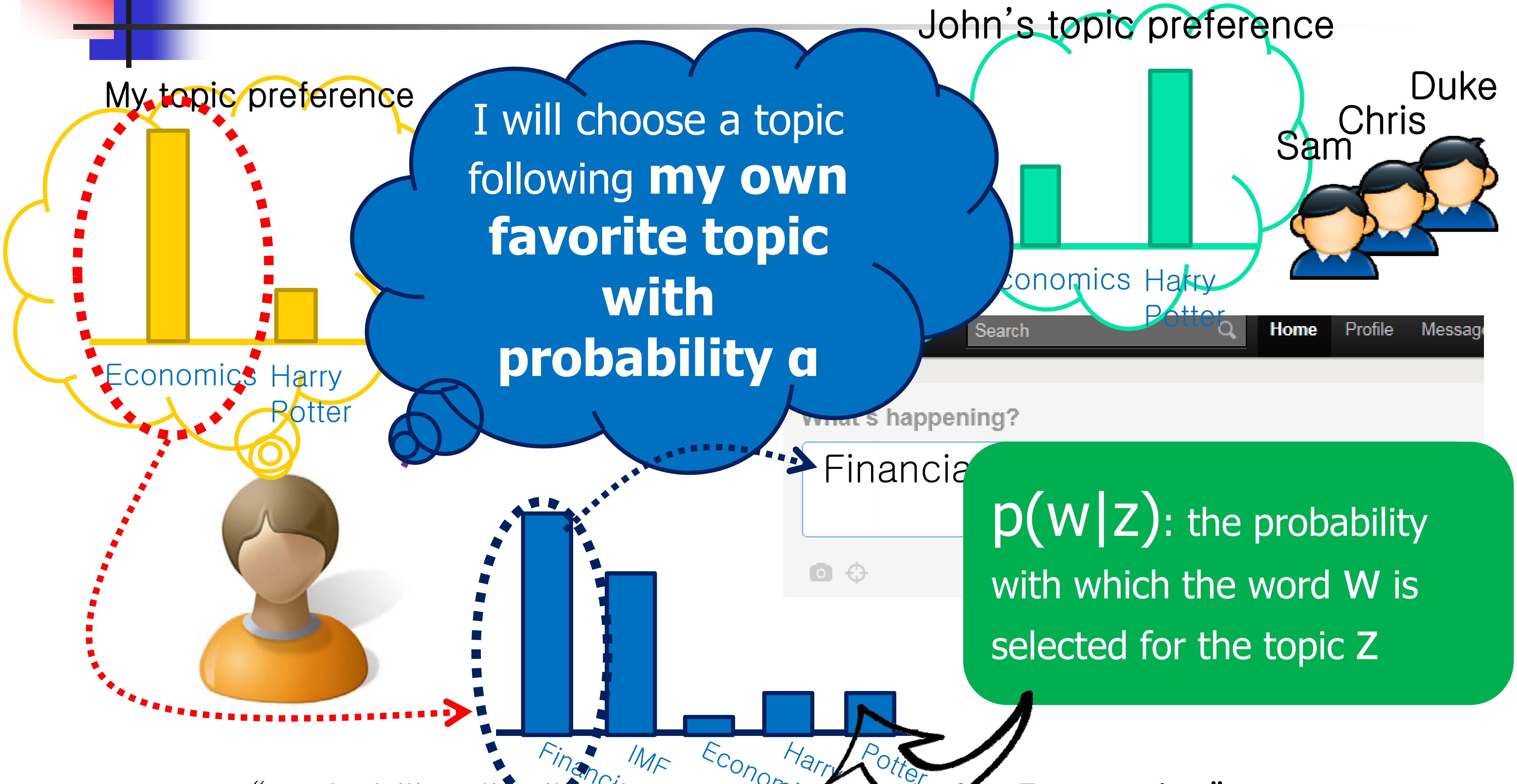


John's topic preference

My topic preference

$p(z|u)$: the probability with which the user u selects the topic z

Our Generative Model

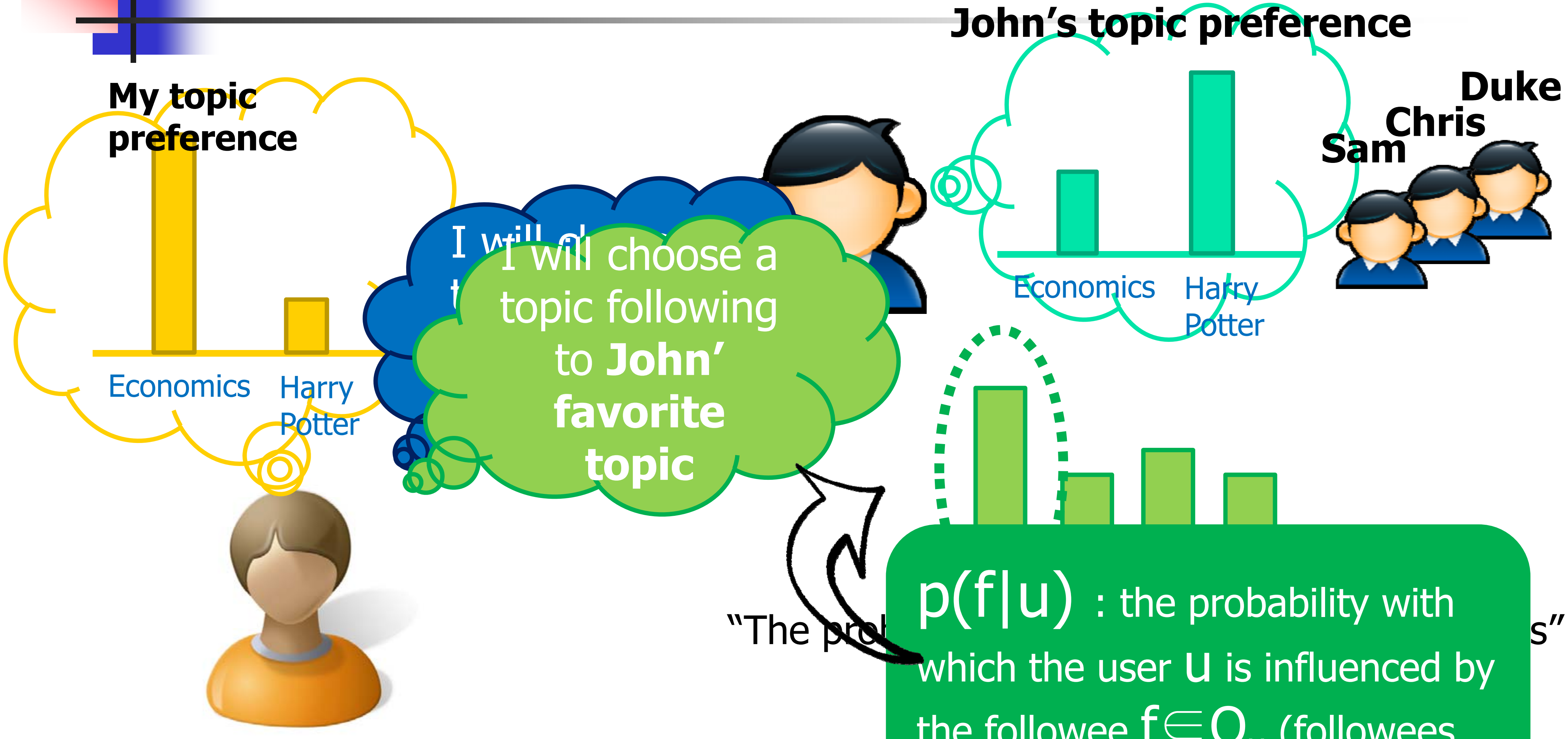


I will choose a topic following **my own favorite topic with probability α**

p(w|z): the probability with which the word W is selected for the topic Z

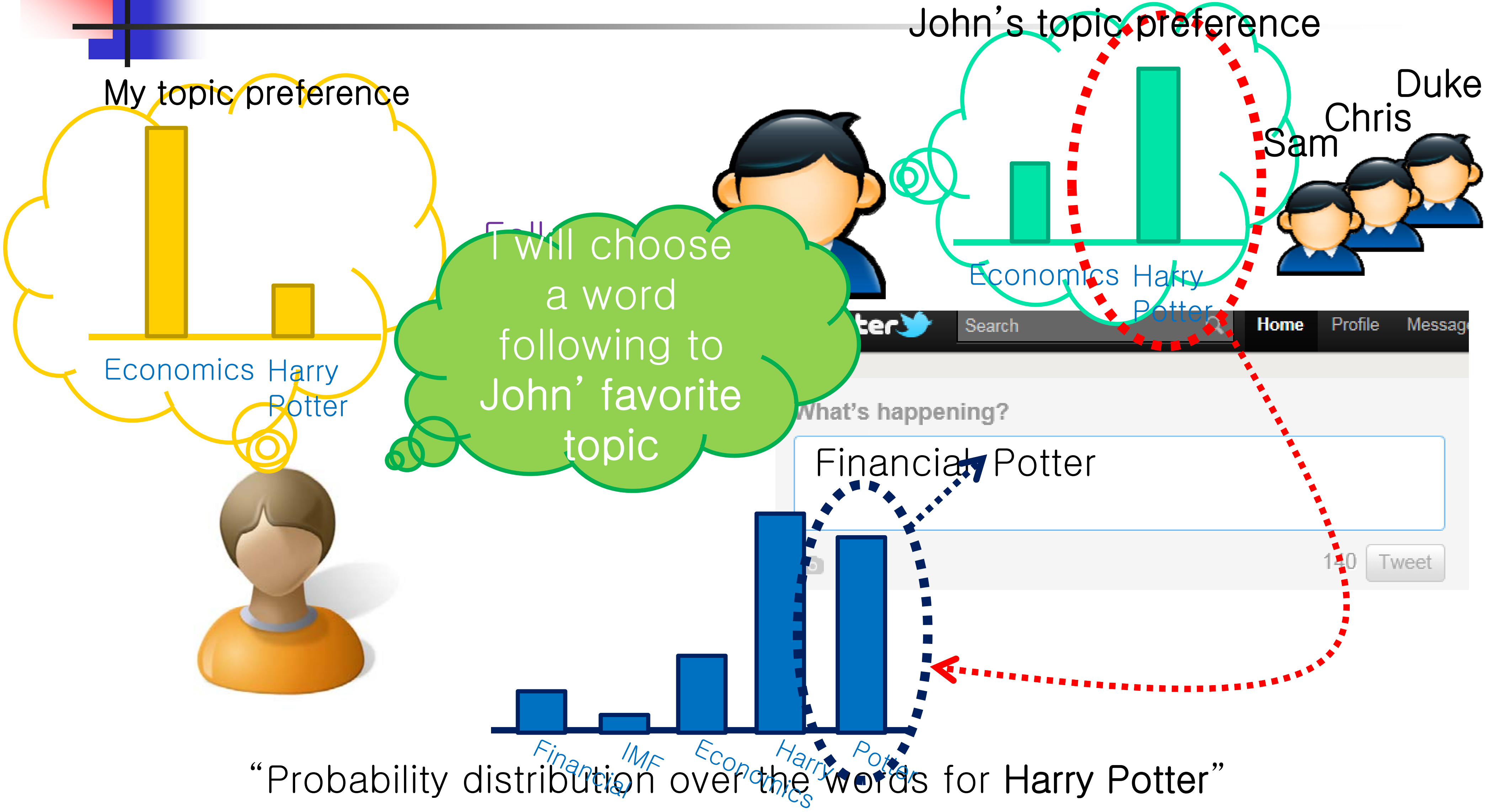
"Probability distribution over the words for Economics"

Our Generative Model



$p(f|u)$: the probability with which the user u is influenced by the followee $f \in O_u$ (followees of u) for selecting topics

Our Generative Model

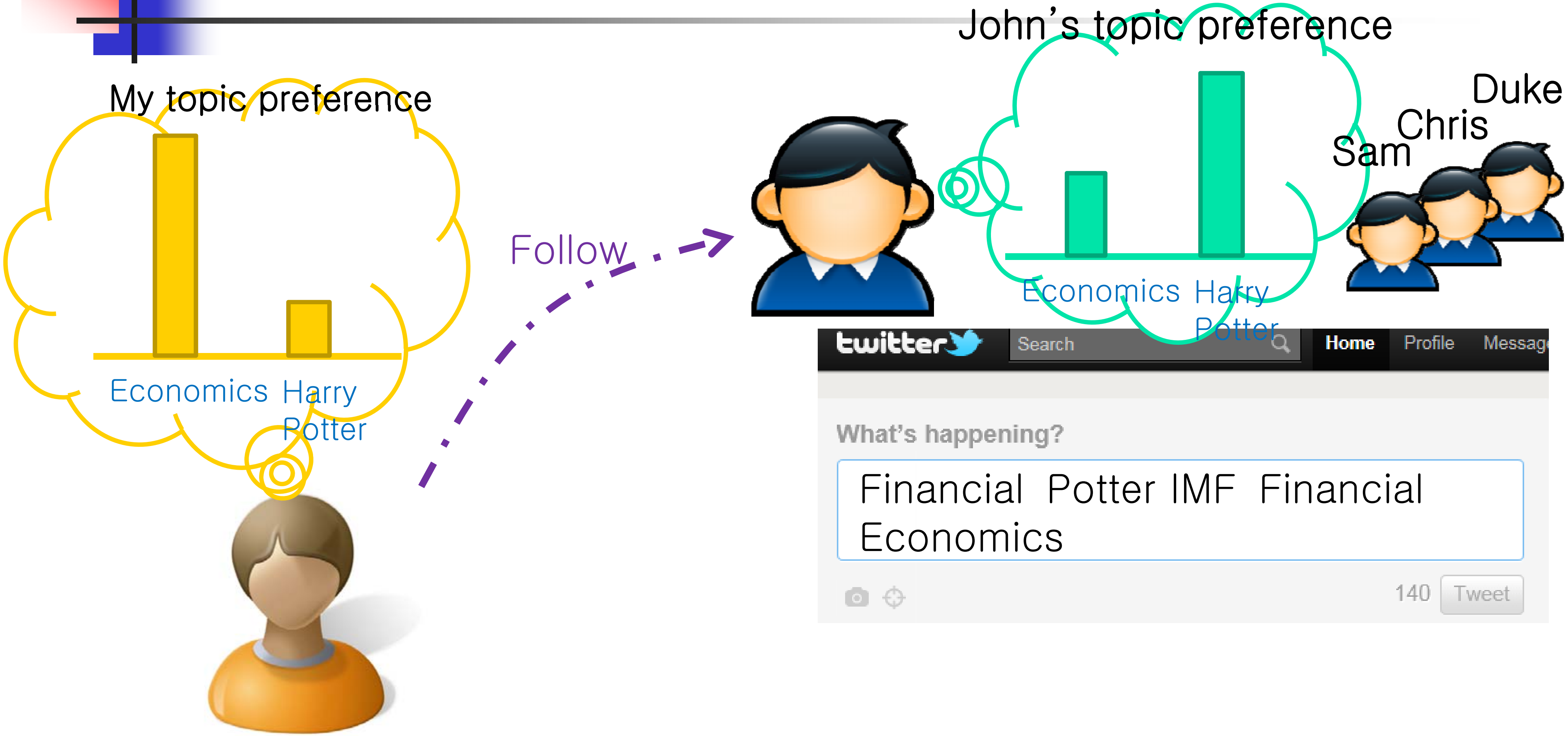


I will choose a word following to John’ favorite topic

Financial Potter

“Probability distribution over the words for Harry Potter”

Our Generative Model



Financial Potter IMF Financial Economics

Likelihood Function of EM Algorithm

- Find parameters which maximize the log-likelihood,

$$\log L = \sum_{u \in U} \sum_{f \in O_u} \log p(f \mid u)$$
$$+ \sum_{u \in U} \sum_{t \in T_u} \sum_{w \in W} n(t, w) \log \sum_{z \in Z} p(w \mid z)[\alpha p(z \mid u) + (1 - \alpha) \sum_{f \in O_u} p(f \mid u) p(z \mid f)]$$

  - where $\Sigma_{z \in Z} p(z|u)=1$, $\Sigma_{f \in Ou} p(f|u)=1$ and $\Sigma_{w \in W} p(w|z)=1$

Parallelizing Our EM Algorithm Using MapReduce

- Rewrite equations of E-Step and M-Step

E-Step:

$$p(\phi=z\mid w,u)=\frac{p(w\mid z)\boxed{\{\alpha p(z\mid u)+(1-\alpha)\sum_{f\in O_u}p(f\mid u)p(z\mid f)\}}}{\sum_{z'\in Z}\left[p(w\mid z')\boxed{\{\alpha p(z'\mid u)+(1-\alpha)\sum_{f\in O_u}p(f\mid u)p(z'\mid f)\}}\right]}$$

$$p(\theta=u\mid z,u)=\frac{\alpha p(z\mid u)}{\boxed{\alpha p(z\mid u)+(1-\alpha)\sum_{f\in O_u}p(f\mid u)p(z\mid f)}}$$

$$p(\theta=f\mid z,u)=\frac{(1-\alpha)p(f\mid u)p(z\mid f)}{\boxed{\alpha p(z\mid u)+(1-\alpha)\sum_{f'\in O_u}p(f'\mid u)p(z\mid f')}}$$

A common expression X(u,z)

$$X(u,z)=\alpha p(z\mid u)+$$
$$(1-\alpha)\sum_{f\in O_u}p(f\mid u)p(z\mid f)$$

M-Step:

$$p(w\mid z)=\frac{\sum_{u\in U}\sum_{t\in T_u}n(t,w)p(\phi=z\mid w,u)}{\sum_{w'\in W}\sum_{u\in U}\sum_{t\in T_u}n(t,w')p(\phi=z\mid w',u)}$$

$$p(z\mid u)=\frac{\sum_{t\in T_u}\sum_{w\in W}n(t,w)p(\phi=z\mid w,u)+\sum_{i\in I_u}\sum_{t\in T_i}\sum_{w\in W}n(t,w)p(\phi=z\mid w,i)p(\theta=u\mid z,i)}{\sum_{z'\in Z}\left[\sum_{t\in T_u}\sum_{w\in W}n(t,w)p(\phi=z'\mid w,u)+\sum_{i\in I_u}\sum_{t\in T_i}\sum_{w\in W}n(t,w)p(\phi=z'\mid w,i)p(\theta=u\mid z',i)\right]}$$

$$p(f\mid u)=\frac{1+\sum_{t\in T_u}\sum_{w\in W}\sum_{z\in Z}n(t,w)p(\phi=z\mid w,u)p(\theta=f\mid z,u)}{|O_u|+\sum_{f\in O_u}\sum_{t\in T_u}\sum_{w\in W}\sum_{z\in Z}n(t,w)p(\phi=z\mid w,u)p(\theta=f\mid z,u)}$$

Parallelizing Our EM Algorithm Using MapReduce

- Rewrite equations of E-Step and M-Step

E-Step:

$$p(\phi=z\,|\,w,u)=\frac{p(w\,|\,z)\boxed{X(u,z)}}{\sum_{z'\in Z}\left[p(w\,|\,z')\boxed{X(u,z)}\right]}$$

Compute X(u,z)
in the first MapReduce step

$$p(\theta=u\,|\,z,u)=\frac{\alpha p(z\,|\,u)}{\boxed{X(u,z)}}$$

$$p(\theta=f\,|\,z,u)=\frac{(1-\alpha)p(f\,|\,u)p(z\,|\,f)}{\boxed{X(u,z)}}$$

Compute model parameters
in the second MapReduce step

M-Step:

$$p(w\,|\,z)=\frac{\sum_{u\in U}\sum_{t\in T_u}n(t,w)p(\phi=z\,|\,w,u)}{\sum_{w'\in W}\sum_{u\in U}\sum_{t\in T_u}n(t,w')p(\phi=z\,|\,w',u)}$$

$$p(z\,|\,u)=\frac{\sum_{t\in T_u}\sum_{w\in W}n(t,w)p(\phi=z\,|\,w,u)+\sum_{i\in I_u}\sum_{t\in T_i}\sum_{w\in W}n(t,w)p(\phi=z\,|\,w,i)p(\theta=u\,|\,z,i)}{\sum_{z'\in Z}\left[\sum_{t\in T_u}\sum_{w\in W}n(t,w)p(\phi=z'\,|\,w,u)+\sum_{i\in I_u}\sum_{t\in T_i}\sum_{w\in W}n(t,w)p(\phi=z'\,|\,w,i)p(\theta=u\,|\,z',i)\right]}$$

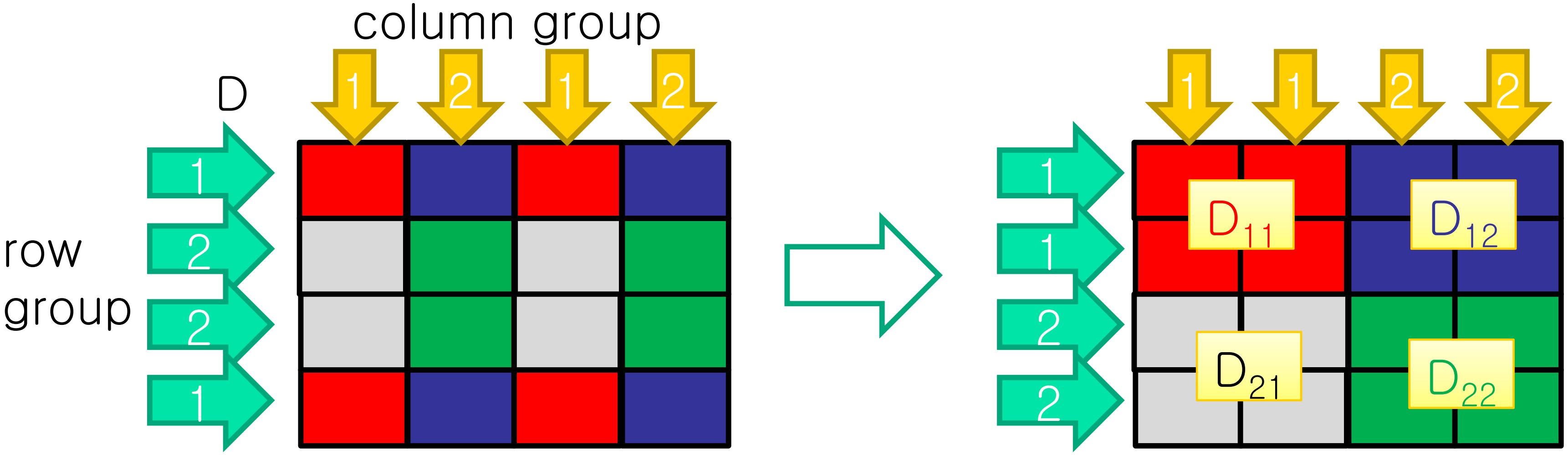
$$p(f\,|\,u)=\frac{1+\sum_{t\in T_u}\sum_{w\in W}\sum_{z\in Z}n(t,w)p(\phi=z\,|\,w,u)p(\theta=f\,|\,z,u)}{|O_u|+\sum_{f\in O_u}\sum_{t\in T_u}\sum_{w\in W}\sum_{z\in Z}n(t,w)p(\phi=z\,|\,w,u)p(\theta=f\,|\,z,u)}$$

Co-clustering using MapReduce

Co-clustering

- Given
  - Matrix D
- Find
  - Row group, column group s.t each cluster D_{ij} have similar characteristic

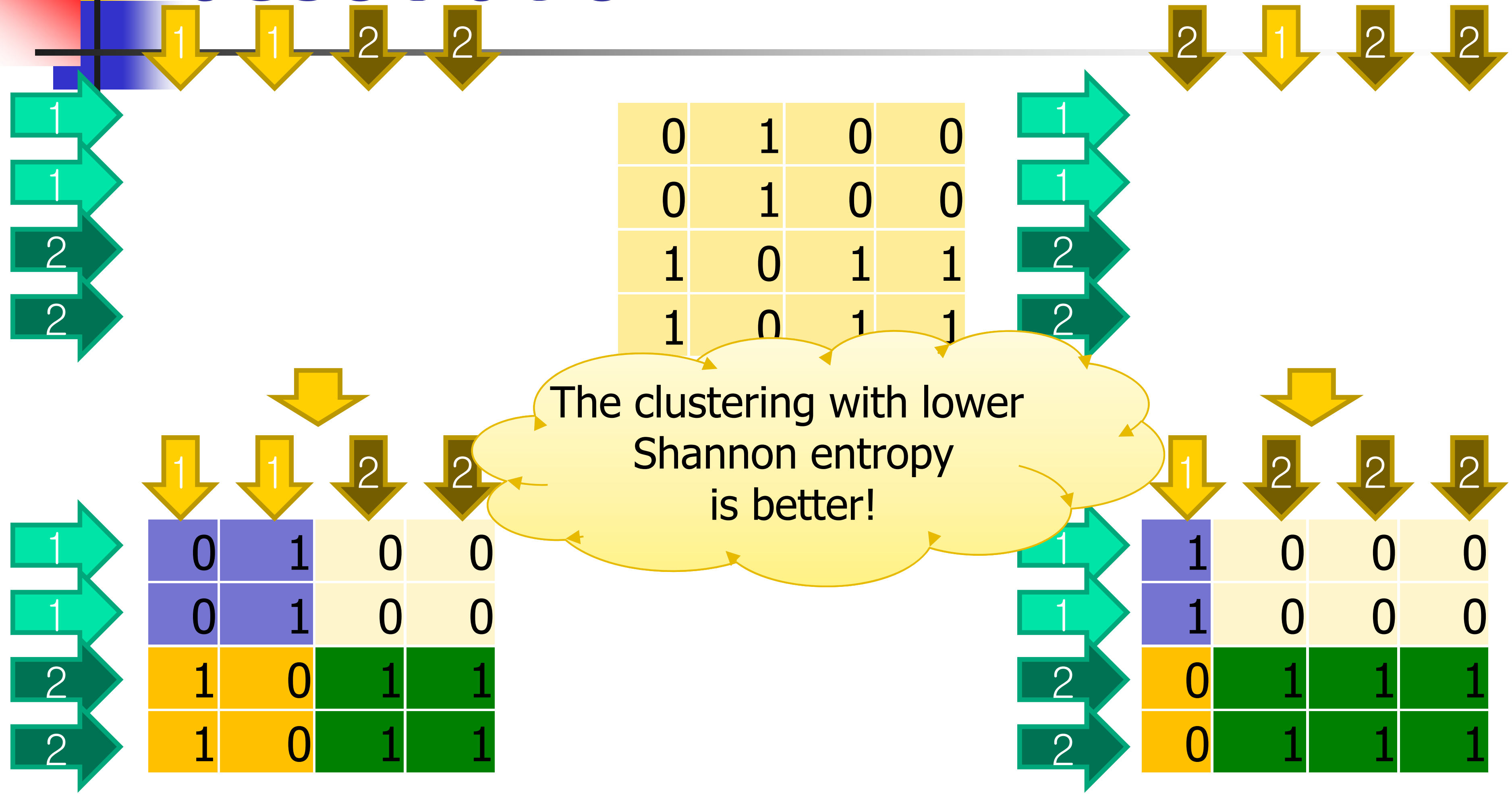
Cluster D_{ij} is the cross section of the i-th row group and the j-th column group



Serial Co-clustering Algorithms

- Co-clustering algorithms have a model for each cluster and minimize the encoding cost
  - Cross-association algorithm
    - [Chakrabarti, Modha, Papadimitriou, Faloutsos: KDD 2004]
    - Used for binary value matrix
    - Use Shannon entropy for encoding cost
  - SCOAL algorithm
    - [Deodhar, Ghosh: KDD 2007]
    - Used for real value matrix
    - Use linear approximation model for attribute values
    - Use square error sum for encoding cost

An Example of Cross-association



0	1	0	0
0	1	0	0
1	0	1	1
1	0	1	1

The clustering with lower Shannon entropy is better!

	1	1	2	2
1	0	1	0	0
1	0	1	0	0
2	1	0	1	1
2	1	0	1	1

	1	2	2	2
1	1	0	0	0
1	1	0	0	0
2	0	1	1	1
2	0	1	1	1

(Total Shannon entropy)
= 2log2+0+0+2log2 = 4log2

>

(Total Shannon entropy)
= 0+0+0+0 = 0

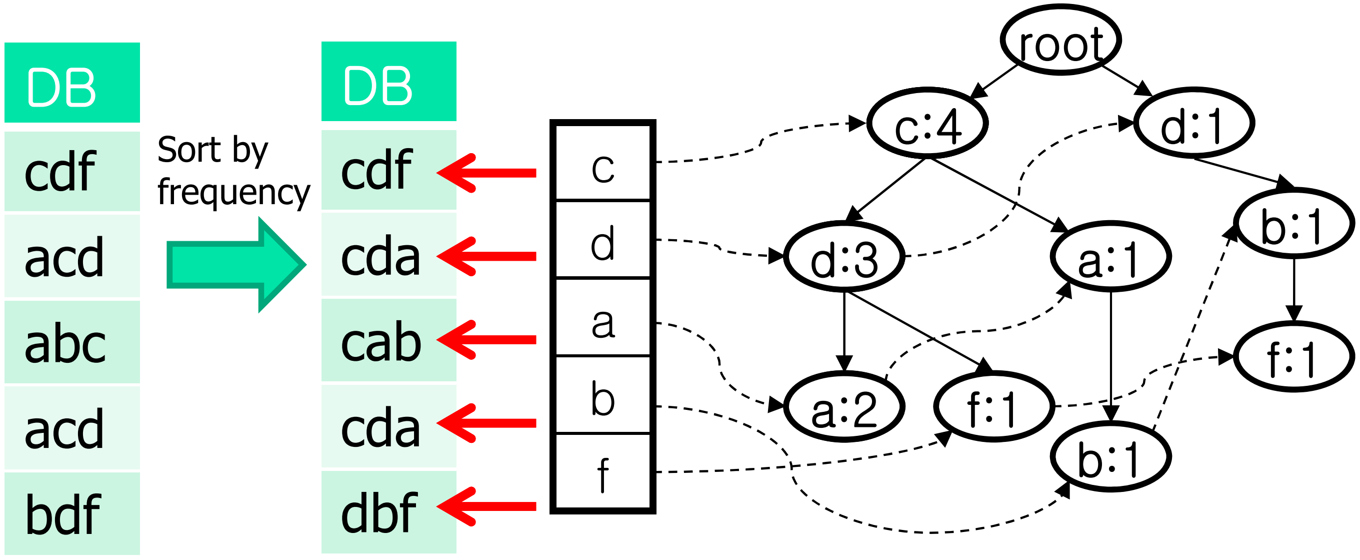
Co-clustering Algorithms Using MapReduce

- DisCo: Parallelize Cross-association algorithm
  - [Papadimitriou, Sun: ICDM 08]
  - A parallelized cross-association algorithm
  - Since each row or column group assignment is independent, parallelization is easy
- Parallelize SCOAL algorithm
  - [Deodhar, Jones, Ghosh: GrC 10]
  - Real value matrices
  - Linear approximation model for attribute values
  - Square error sum for encoding cost

Association Rule Mining using MapReduce

Build FP-tree

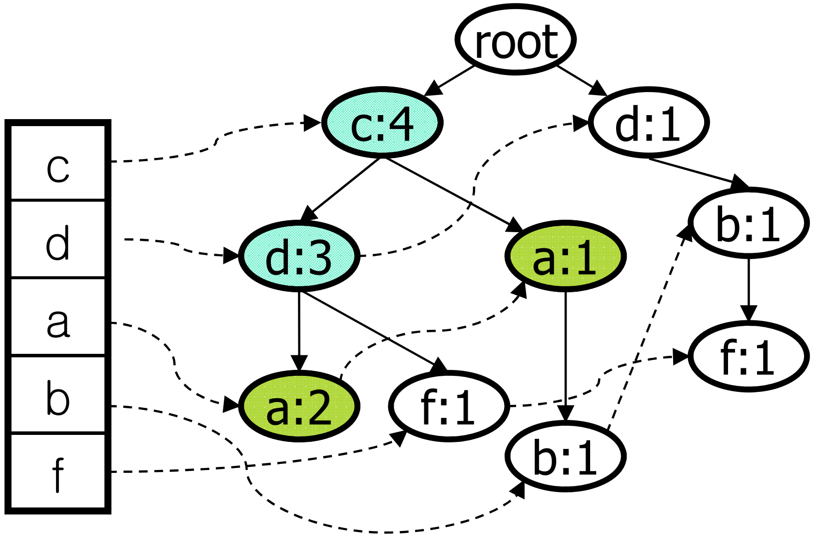
- Sort items in each record
- Build a tree structure using sorted records
- Maintain pointers which link the nodes with the same items together



Conditional Pattern Base

- A sub-pattern base under the condition of existence of a certain pattern

- e.g.) min_sup = 2



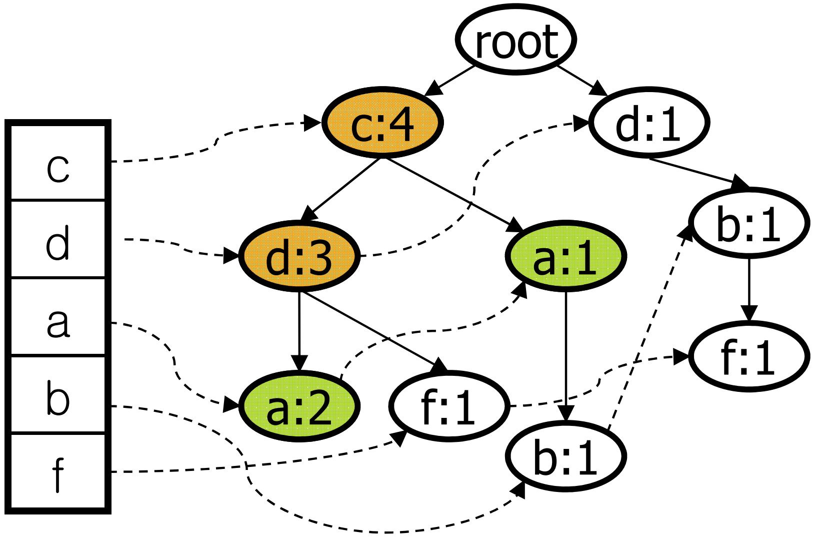
Nodes that contribute a’s conditional pattern bases

a’s conditional pattern bases

- (cd:2), (c:1)

Conditional FP-tree

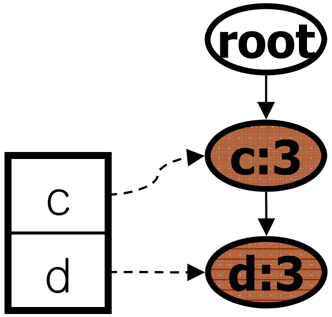
- FP-tree on the conditional pattern bases of a certain item
- If FP-tree consists of single path, all the combinations of items in the path are the freq patterns
- Example (min_sup = 2)



a's conditional pattern bases
- (cd:2), (c:1)

The nodes that contribute to a's conditional FP-tree

a's conditional FP-tree
- (cd:3)



FP-tree Algorithm with MapReduce

- [Li, Wang, Zhang, Zhang, Chang: ACM Recom. Systems 2008]
- Step 1: word counting (MapReduce)
  - Count all items (frequent items: F-List)
  - Sort each transaction in order of frequency
- Step 2: grouping items
  - Dividing all frequent items into Q
- Step 3: parallel FP-Growth (MapReduce)
  - Generate group-independent databases
  - FP-Growth on group-independent databases
- Step 4: aggregating
  - Aggregate frequent patterns
  - For each item, get the set of patterns including the item

Step 3: Map

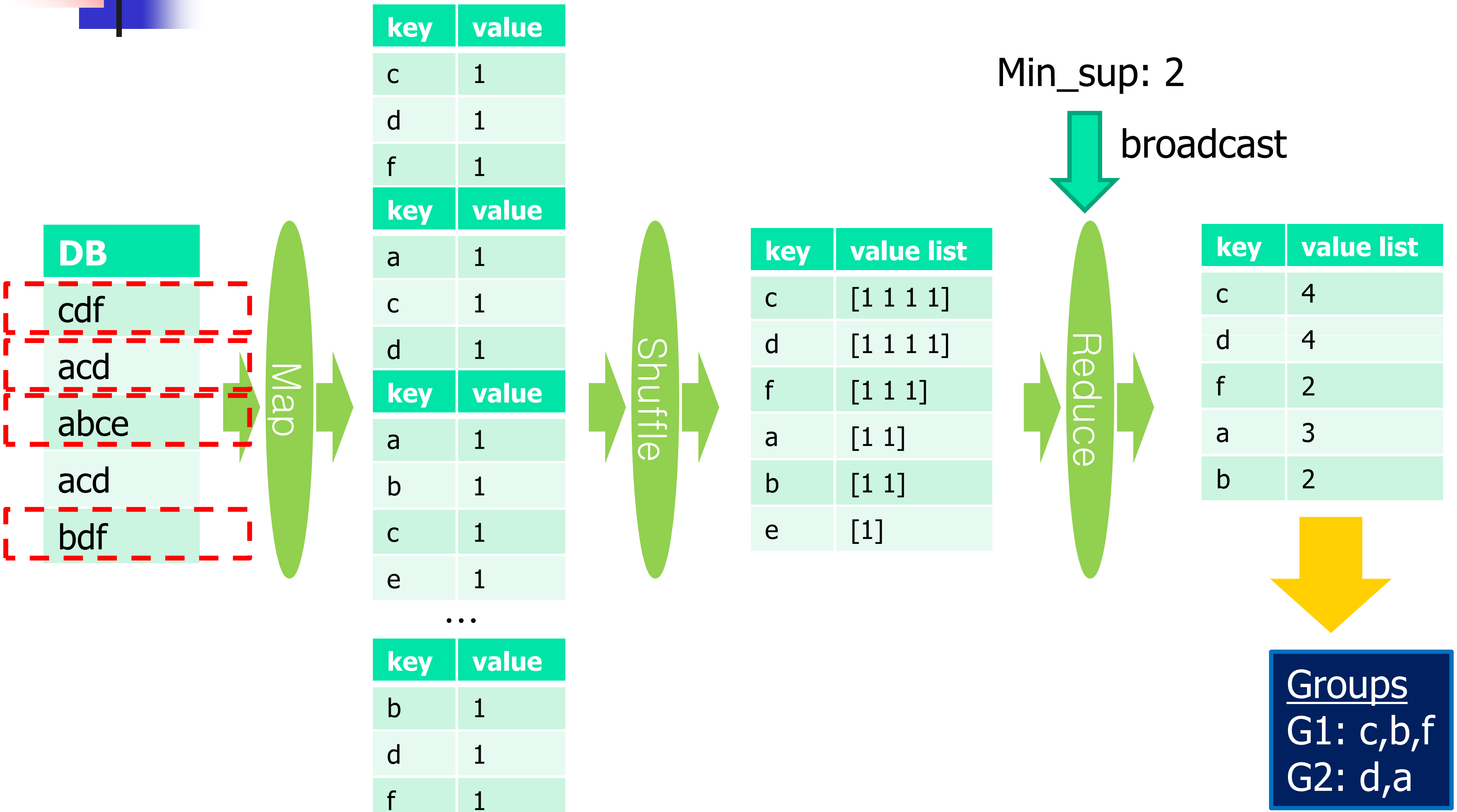
- Map (key, value(=T_i))
  - Load G-List by broadcasting
  - Generate Hash Table H from G-List
    - Map each item in T_i with its group id
  - For j= $|T_i|$-1 to 0 do
    - GroupID = getHash(H, a[j])
    - If GroupID is not null
      - Delete all entry in H whose group id is GroupID
      - Output < GroupID, a[0]+a[1]+…+a[j]>

To reduce the number of emitted duplicate transactions.
e.g.) if T_i=fcamp, and a, p are in same group, fcam and fc should not be emited together.

Step 3: Reduce

- Reduce (key=GID, value=DB_{gid})
  - Load G-List by broadcasting
  - nowItems = items of GID from G-List
  - Initialize LocalFPtree
  - For each T_i in DB_{gid} do
    - Insert (LocalFPtree, T_i)
    - Build header table with nowItems only
  - For each a_i in nowItems do
    - FPGrowth (LocalFPtree, a_i)
      - Output <pattern, support>

An Illustration of Step 1 and 2



DB
cdf
acd
abce
acd
bdf

key	value
c	1
d	1
f	1

key	value
a	1
c	1
d	1

key	value
a	1
b	1
c	1
e	1

…

key	value
b	1
d	1
f	1

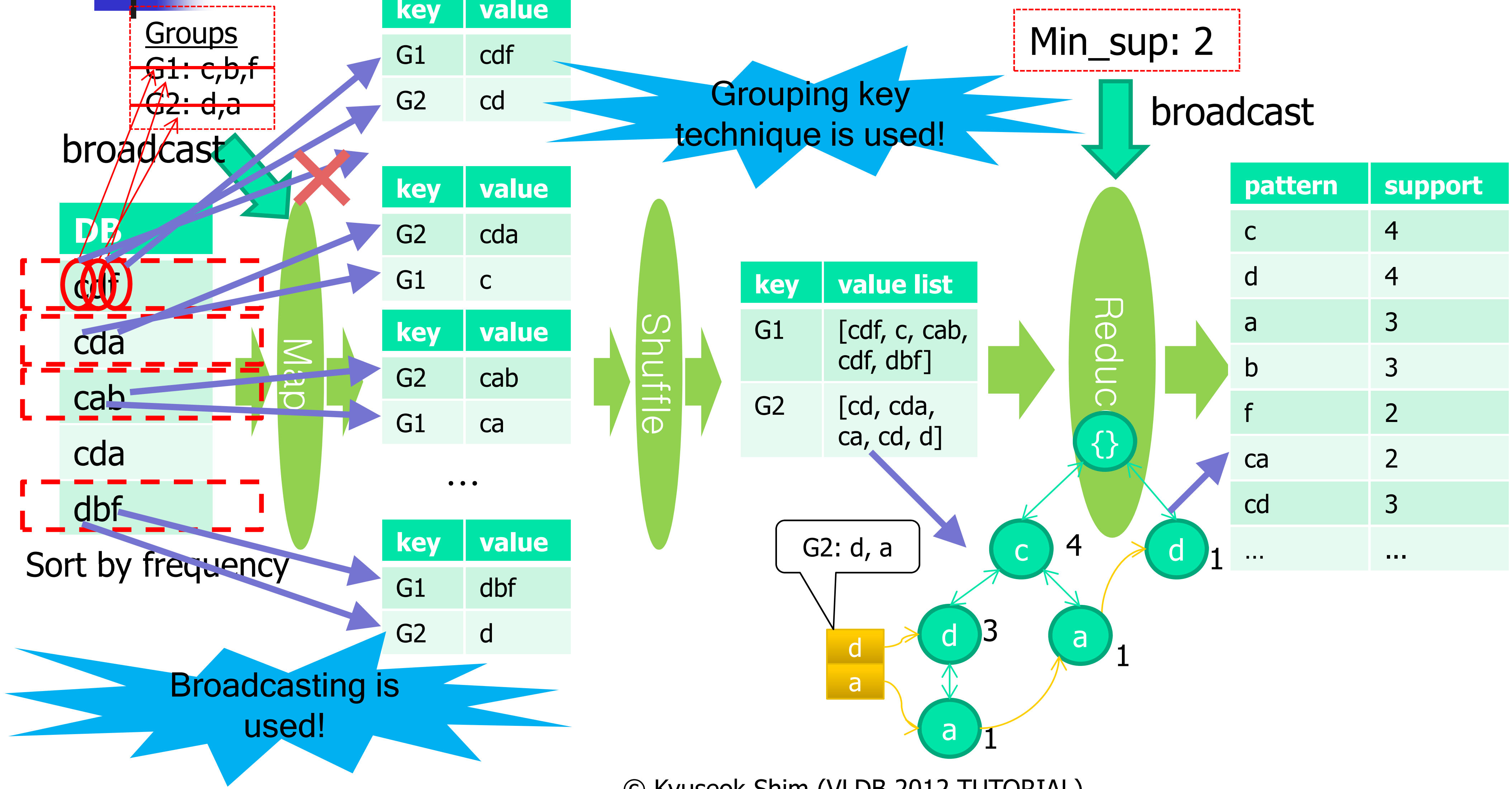
key	value list
c	[1 1 1 1]
d	[1 1 1 1]
f	[1 1 1]
a	[1 1]
b	[1 1]
e	[1]

Min_sup: 2

key	value list
c	4
d	4
f	2
a	3
b	2

Groups
G1: c,b,f
G2: d,a

An Illustration of Step 3



Grouping key technique is used!

Broadcasting is used!

key	value
G1	cdf
G2	cd

key	value
G2	cda
G1	c

key	value
G2	cab
G1	ca

…

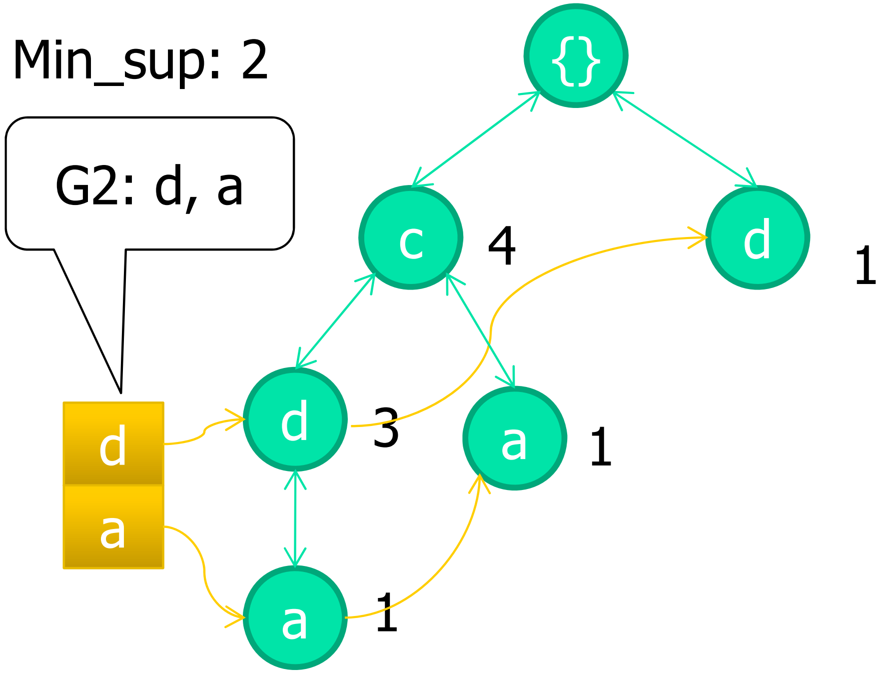
key	value
G1	dbf
G2	d

key	value list
G1	[cdf, c, cab, cdf, dbf]
G2	[cd, cda, ca, cd, d]

pattern	support
c	4
d	4
a	3
b	3
f	2
ca	2
cd	3
…	…

An Illustration of Step 3 - Reduce

- FPGrowth



Possible subsets: {c}, ∅

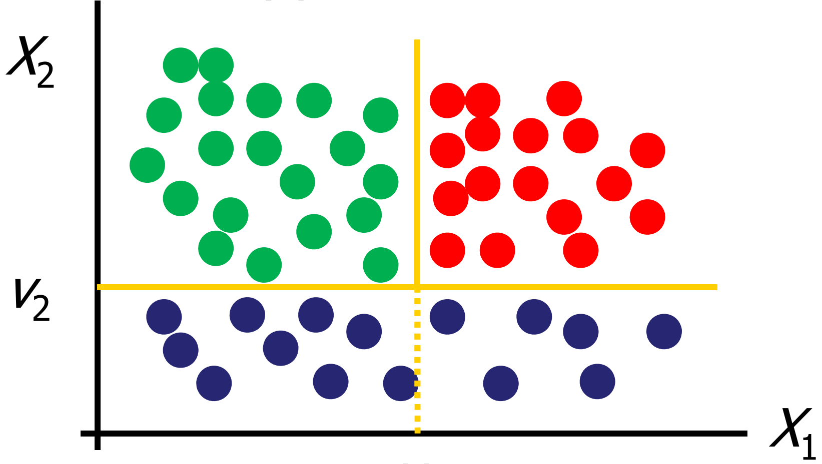
a's conditional patterns: {cd:1, c:1}
=> output {a:2, ca:2}
d's conditional pattern: {c:3, ∅:1}
=> output {d:4, cd:3}

Possible subsets: {d}, ∅

Decision Tree Classification using MapReduce

Problem Formulation

- Supervised learning problem
  - Given a dataset D^*
    - $\mathbf{X}=\{X_1, X_2, \ldots X_N\}$ is a set of attributes with domains $\mathbf{D}_{X_1}, \mathbf{D}_{X_2}, \ldots \ \mathbf{D}_{X_N}$
    - Y is an output with domain $\mathbf{D}_Y$
    - $D = \{(x_i, y_i) \mid x_i \in \mathbf{D}_{X_1} \text{ x } \mathbf{D}_{X_2} \text{ x} \ldots \ \mathbf{D}_{X_N}, y_i \in \mathbf{D}_Y\}$ where the i-th vector x_i has an output y_i
  - Find a function (or model) F: $\mathbf{D}_{X_1} \text{ x } \mathbf{D}_{X_2} \text{ x} \ldots \ \mathbf{D}_{X_N} \rightarrow \mathbf{D}_Y$ that is the best approximation of the true distribution of D^*



F

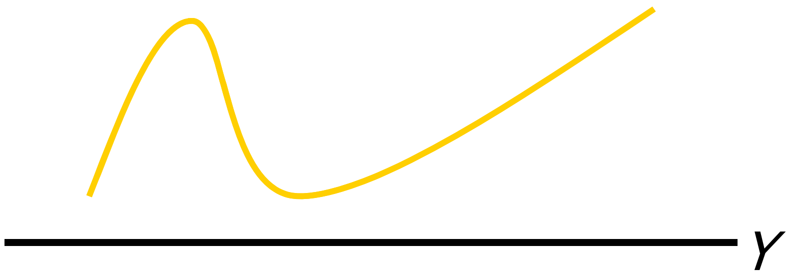
$X_2 < v_2 \rightarrow$ 🔵

$X_1 < v_1$ and $X_2 > v_2 \rightarrow$ 🟢

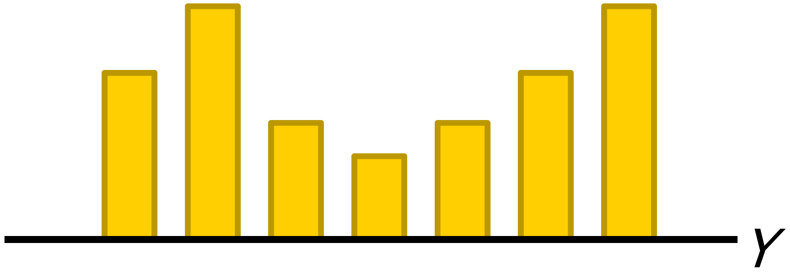
$X_1 > v_1$ and $X_2 > v_2 \rightarrow$ 🔴

A Supervised Learning Problem

- If $\mathbf{D}_Y$ is continuous, the learning problem is a regression problem

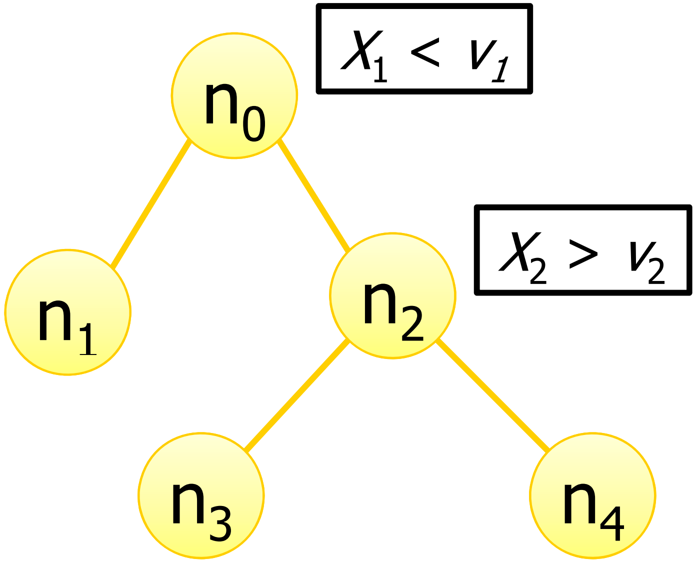


- If $\mathbf{D}_Y$ is categorical, the learning problem is a classification problem



Learning Regression Tree Models

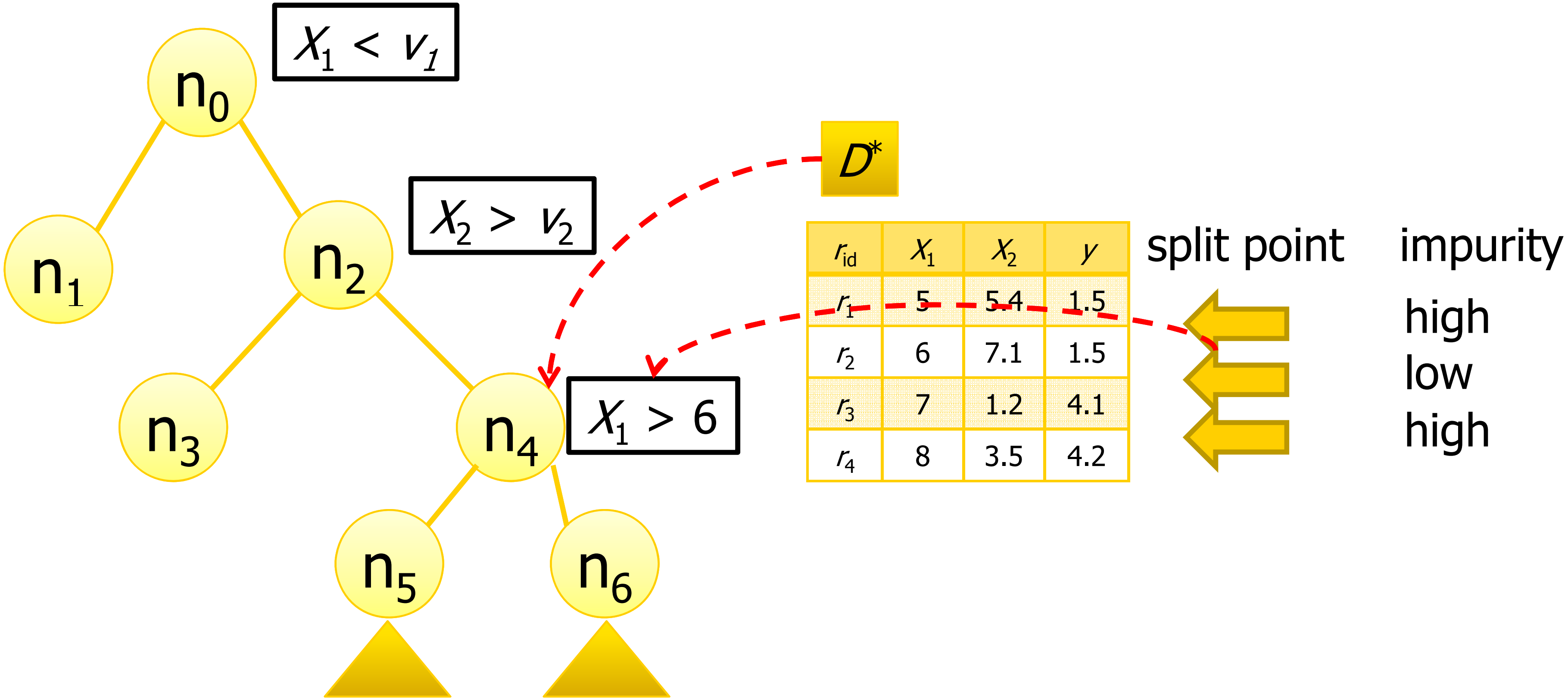
- Represent *F* by recursively partitioning the data space $\mathbf{D}_{X_1} \times \mathbf{D}_{X_2} \times \ldots \mathbf{D}_{X_N}$ into non-overlapping regions
- Constructing the optimal tree is known to be NP-Hard



- Most algorithms use a greedy top-down approach
- The dataset is partitioned along a split predicate
- The process is repeated recursively on the partitions

Learning Regression Tree Models

- Represent *F* by recursively partitioning the data space $\mathbf{D}_{X_1}$ x $\mathbf{D}_{X_2}$ x … $\mathbf{D}_{X_N}$ into non-overlapping regions
- Constructing the optimal tree is known to be NP-Hard



r_{id}	X_1	X_2	y
r_1	5	5.4	1.5
r_2	6	7.1	1.5
r_3	7	1.2	4.1
r_4	8	3.5	4.2

split point

impurity

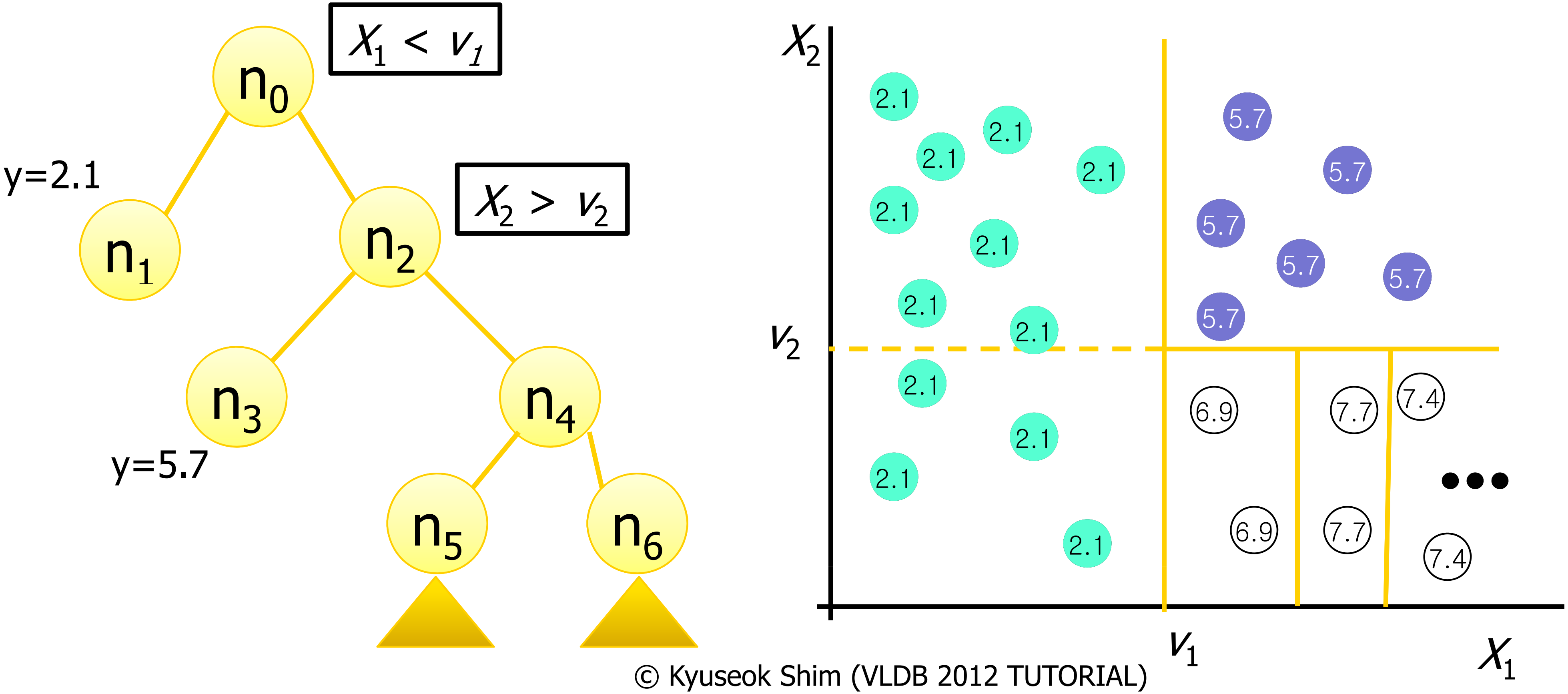
high

low

high

Learning Regression Tree Models

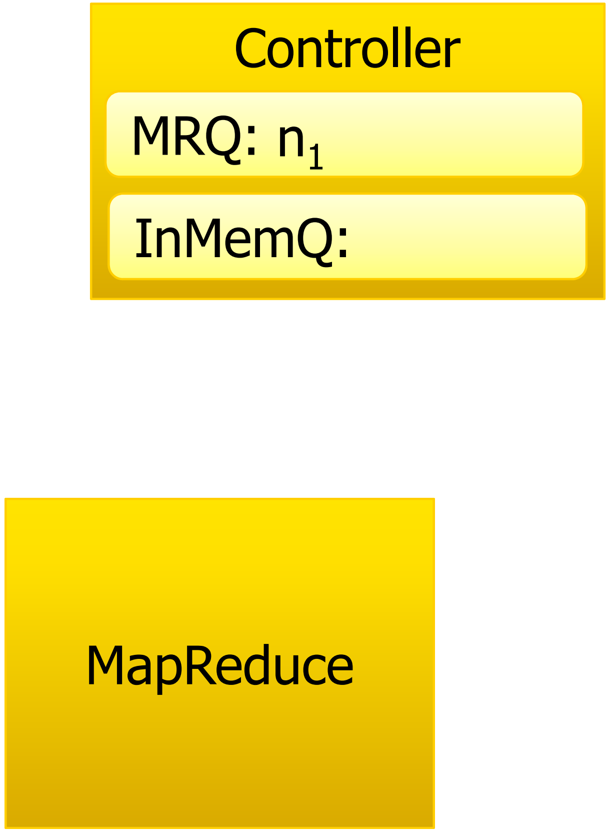
- Represent *F* by recursively partitioning the data space $\mathbf{D}_{X_1} \times \mathbf{D}_{X_2} \times \ldots \mathbf{D}_{X_N}$ into non-overlapping regions
- Constructing the optimal tree is known to be NP-Hard



PLANET

- [Panda, Herbach, Basu, and Bayardo: VLDB, 2012]
- Breaks up the process of constructing a tree model into a set of MapReduce tasks
- Uses a schedule to efficiently execute and manage MapReduce tasks
- Controller - the core of PLANET
  - A machine controlling the entire tree induction process
  - PLANET maintains the followings
    - ModelFile (*M*): contains the entire tree constructed so far
    - MapReduceQueue (*MPQ*): contains nodes whose *D*s are too large to fit in memory
    - InMemoryQueye (*InMemQ*): contains nodes whose *D*s fit in memory

An Illustration of PLANET



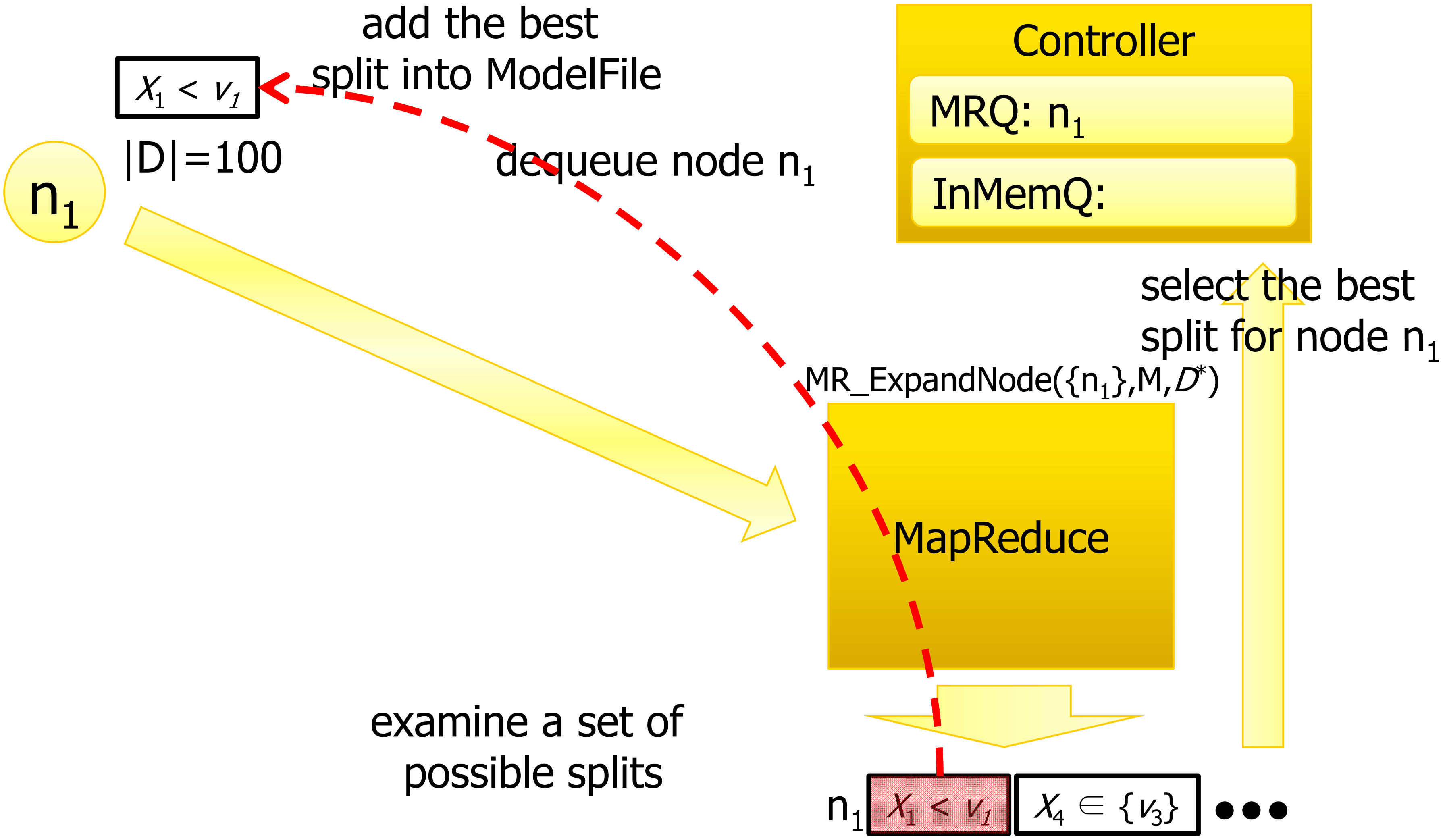
Controller

MRQ: n_1

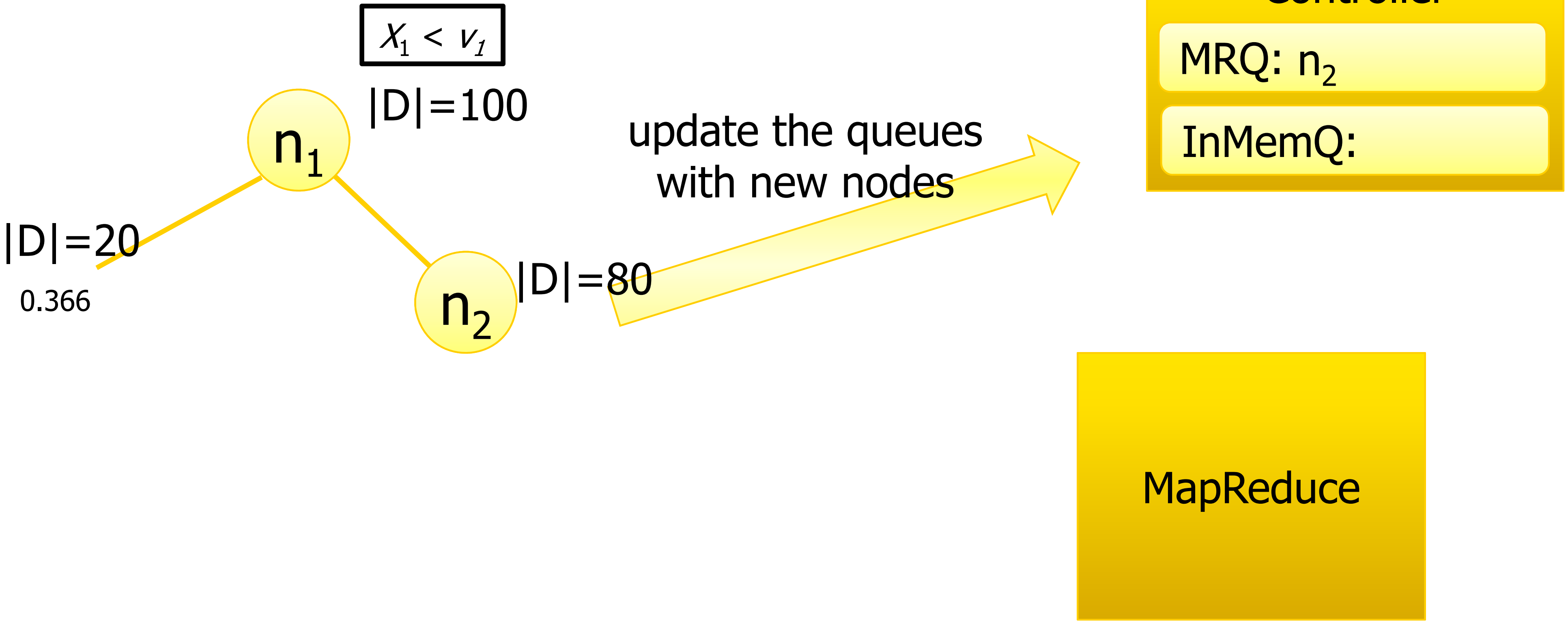
InMemQ:

MapReduce

An Illustration of PLANET



An Illustration of PLANET



$X_1 < v_1$

|D|=100

n_1

|D|=20

0.366

n_2

|D|=80

update the queues with new nodes

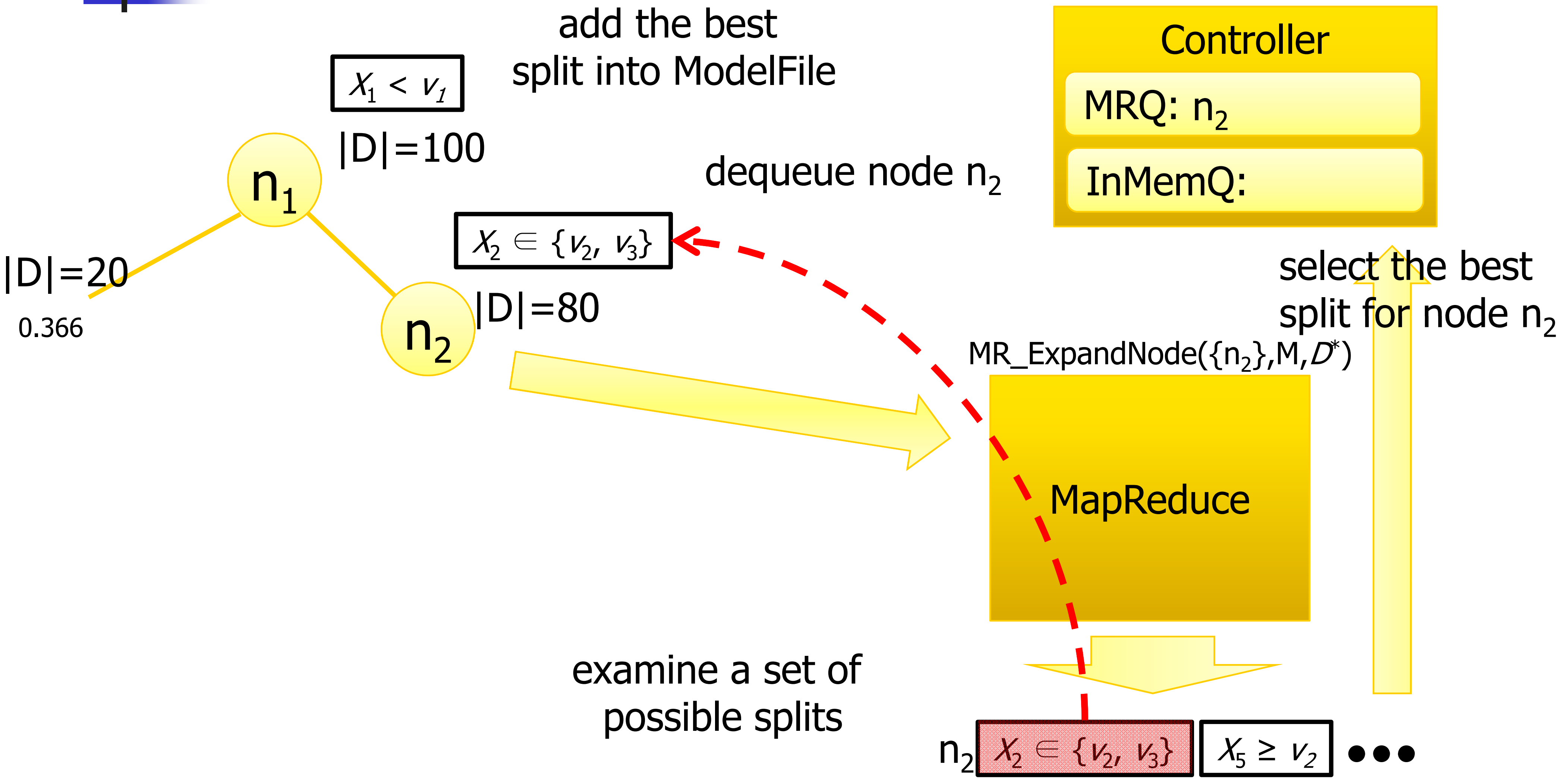
Controller

MRQ: n_2

InMemQ:

MapReduce

An Illustration of PLANET



add the best split into ModelFile

$X_1 < v_1$

$|D|=100$

n_1

dequeue node n_2

Controller

MRQ: n_2

InMemQ:

$|D|=20$

0.366

$X_2 \in \{v_2, v_3\}$

$|D|=80$

n_2

select the best split for node n_2

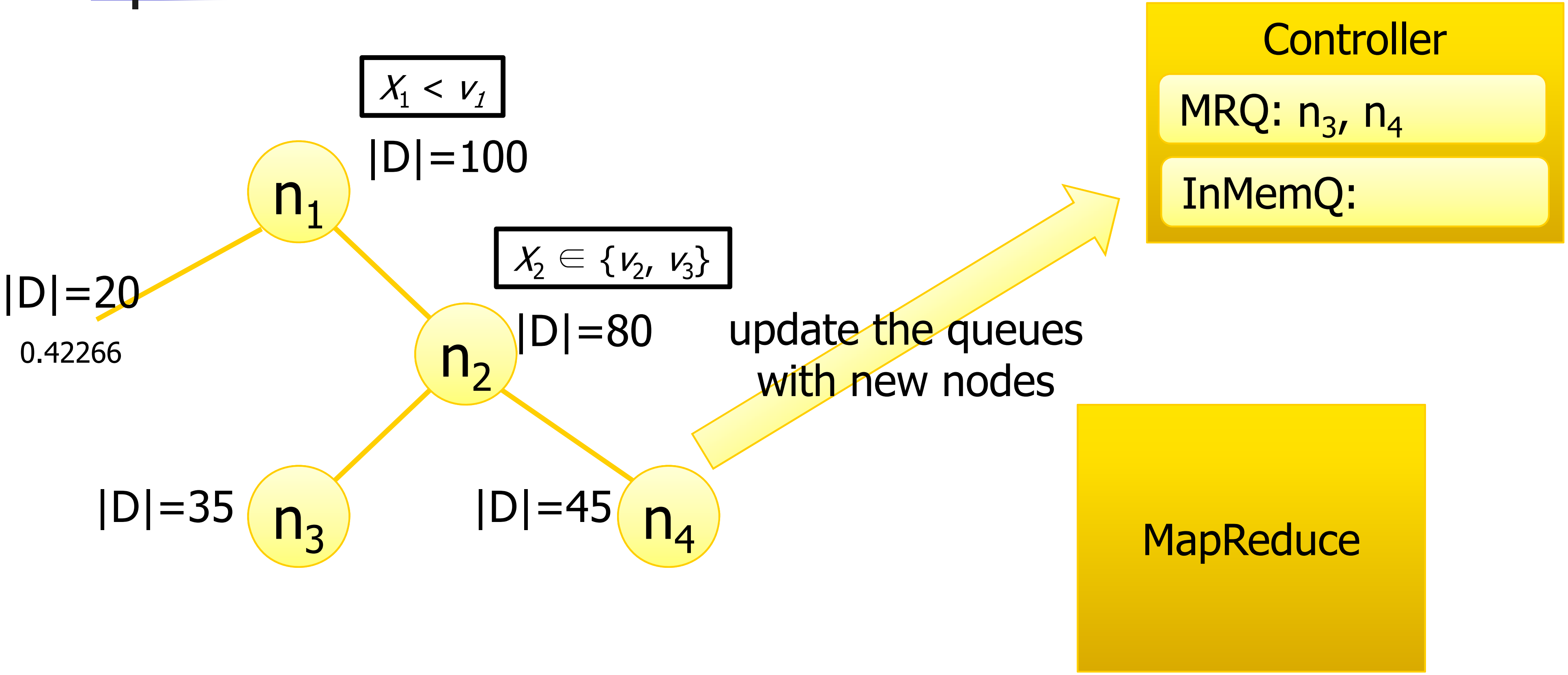
MR_ExpandNode($\{n_2\}$,M,D^*)

MapReduce

examine a set of possible splits

n_2 $X_2 \in \{v_2, v_3\}$ $X_5 \geq v_2$ •••

An Illustration of PLANET



$X_1 < v_1$

|D|=100

n_1

|D|=20

0.42266

$X_2 \in \{v_2, v_3\}$

|D|=80

n_2

|D|=35

n_3

|D|=45

n_4

update the queues
with new nodes

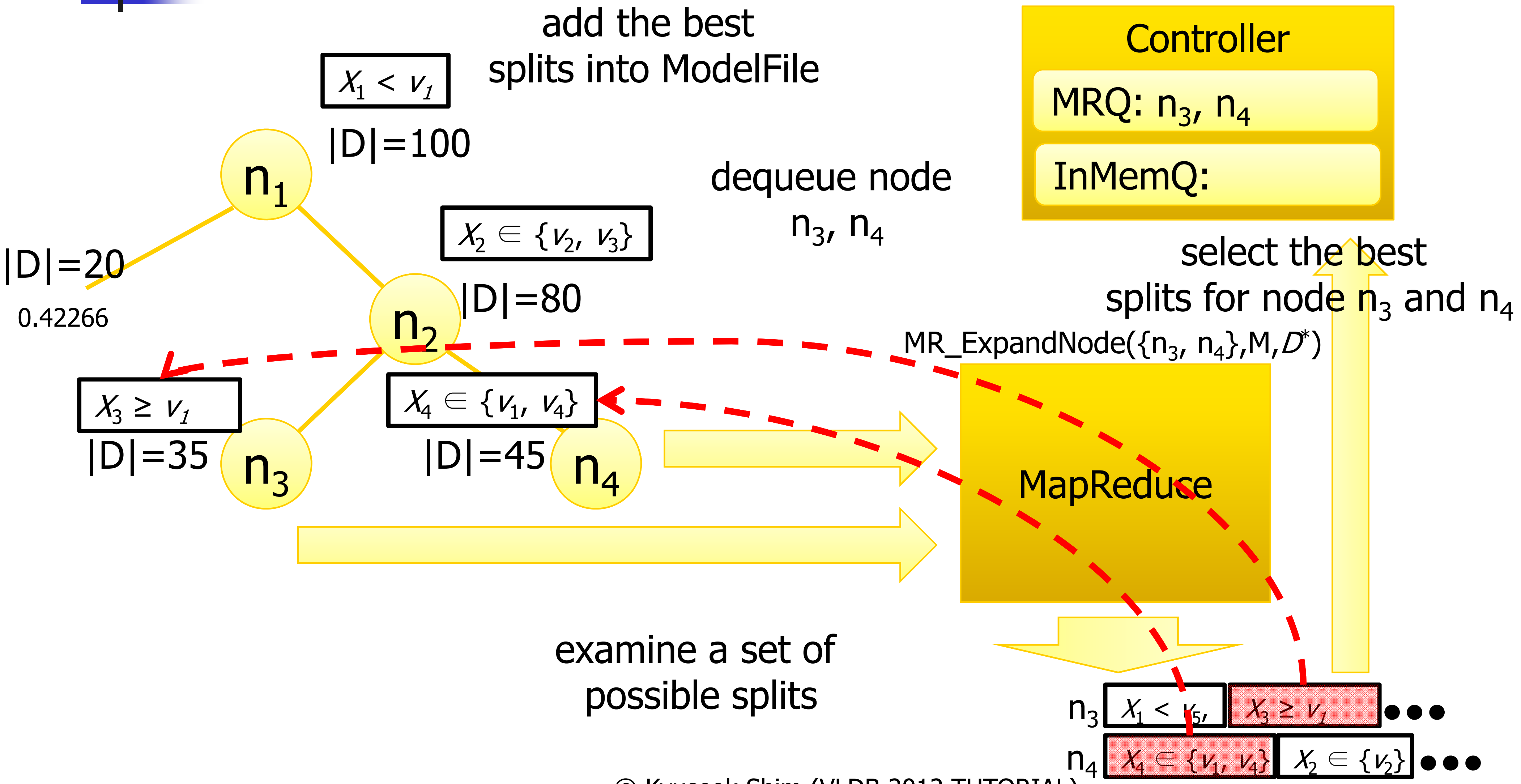
Controller

MRQ: n_3, n_4

InMemQ:

MapReduce

An Illustration of PLANET



add the best splits into ModelFile

Controller

MRQ: n$_3$, n$_4$

InMemQ:

$X_1 < v_1$

|D|=100

n$_1$

|D|=20

0.42266

$X_2 \in \{v_2, v_3\}$

|D|=80

n$_2$

dequeue node n$_3$, n$_4$

select the best splits for node n$_3$ and n$_4$

MR_ExpandNode({n$_3$, n$_4$},M,D^*)

$X_3 \geq v_1$

|D|=35

n$_3$

$X_4 \in \{v_1, v_4\}$

|D|=45

n$_4$

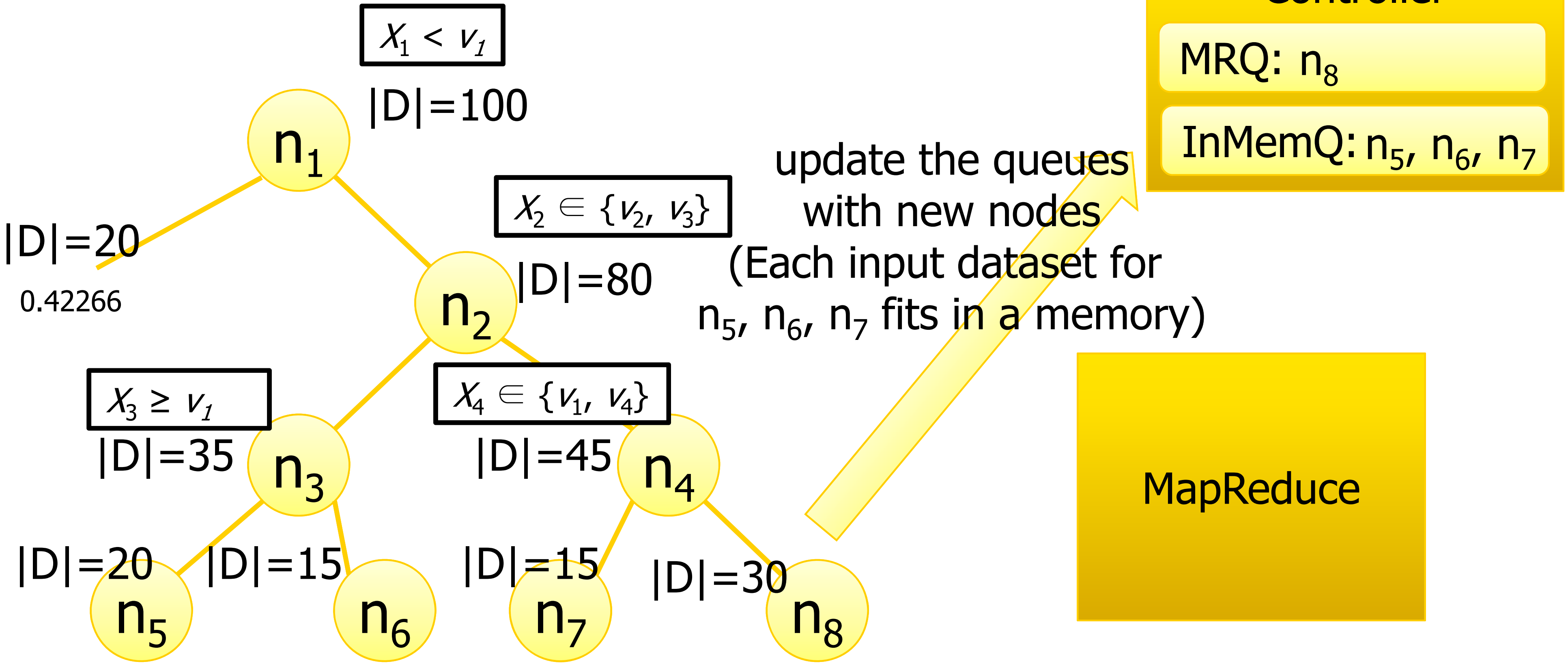
MapReduce

examine a set of possible splits

n$_3$ $X_1 < v_5$, $X_3 \geq v_1$ ●●●

n$_4$ $X_4 \in \{v_1, v_4\}$ $X_2 \in \{v_2\}$ ●●●

An Illustration of PLANET



Controller

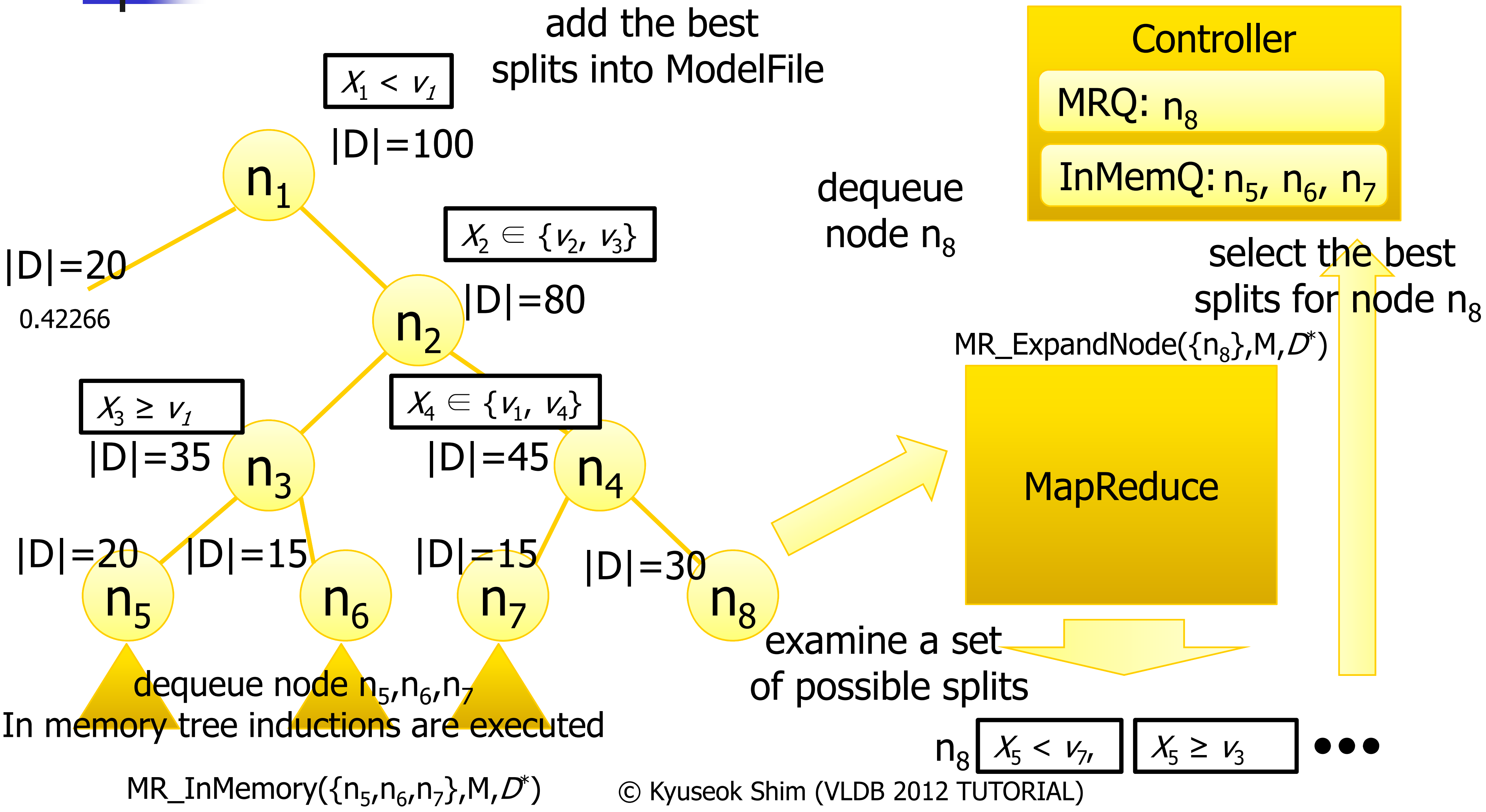
MRQ: n_8

InMemQ: n_5, n_6, n_7

update the queues with new nodes (Each input dataset for n_5, n_6, n_7 fits in a memory)

MapReduce

An Illustration of PLANET



add the best splits into ModelFile

$X_1 < v_1$

|D|=100

n_1

|D|=20

0.42266

$X_2 \in \{v_2, v_3\}$

|D|=80

n_2

$X_3 \geq v_1$

|D|=35

n_3

$X_4 \in \{v_1, v_4\}$

|D|=45

n_4

|D|=20

n_5

|D|=15

n_6

|D|=15

n_7

|D|=30

n_8

dequeue node n_5,n_6,n_7

In memory tree inductions are executed

MR_InMemory($\{n_5,n_6,n_7\}$,M,D^*)

dequeue node n_8

Controller

MRQ: n_8

InMemQ: n_5, n_6, n_7

select the best splits for node n_8

MR_ExpandNode($\{n_8\}$,M,D^*)

MapReduce

examine a set of possible splits

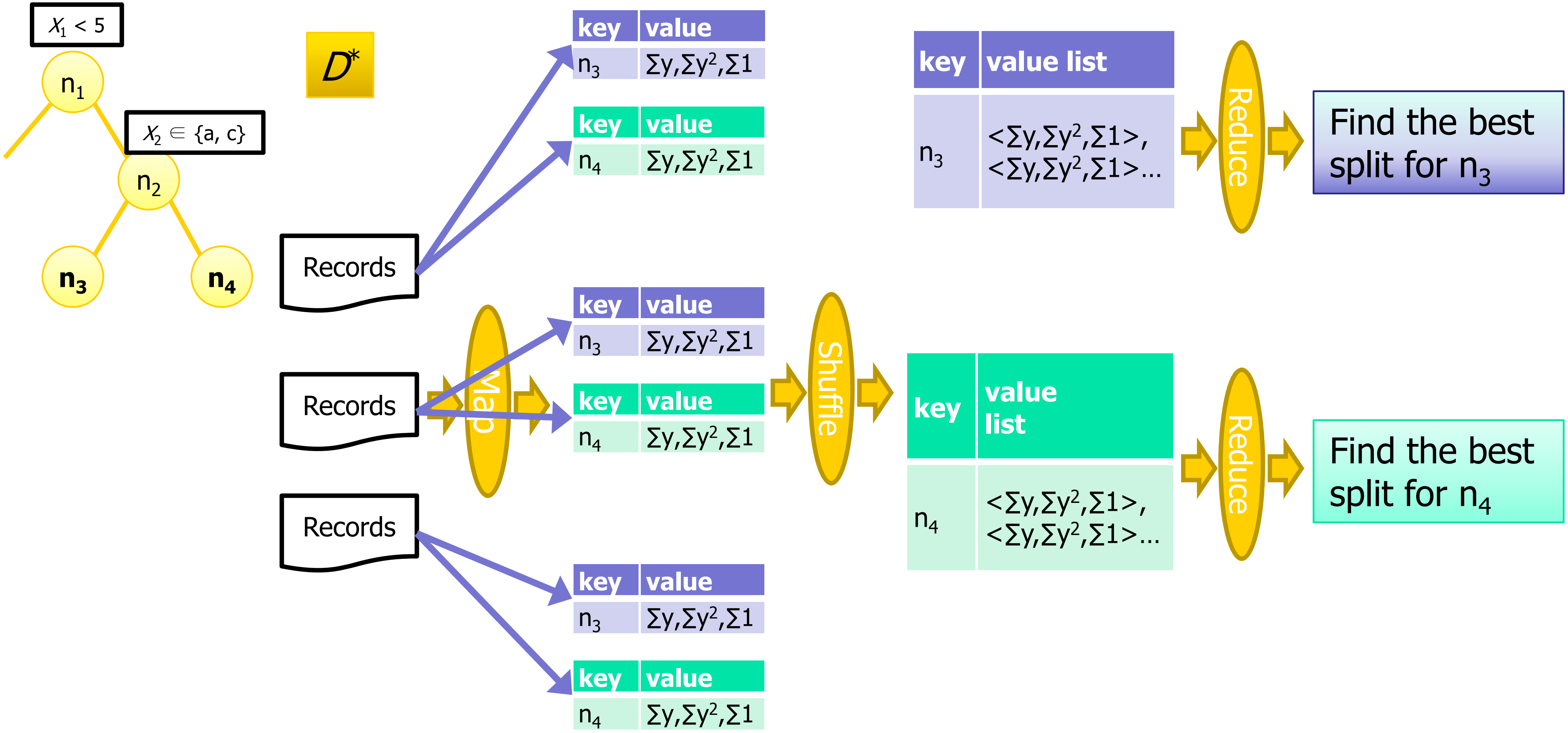
n_8 $X_5 < v_7,$ $X_5 \geq v_3$ •••

Technical Details

- MR_ExpandNode(N, D^*, M)
  - To find the best split, Calculate
    $|D| \times \mathrm{Var}(D) - (|D_L| \times \mathrm{Var}(D_L) + |D_R| \times \mathrm{Var}(D_R))$

- Map Phase
  - D^* is partitioned across a set of mappers
  - Each mapper loads into memory M, N
  - Emit the values of the form $\{\Sigma y, \Sigma y^2, \Sigma 1\}$ where y is the output of the record

An Illustration of MR_ExpandNode($\{n_3, n_4\}, D^*, M$)



key	value
n_3	$\Sigma y, \Sigma y^2, \Sigma 1$

key	value
n_4	$\Sigma y, \Sigma y^2, \Sigma 1$

key	value
n_3	$\Sigma y, \Sigma y^2, \Sigma 1$

key	value
n_4	$\Sigma y, \Sigma y^2, \Sigma 1$

key	value
n_3	$\Sigma y, \Sigma y^2, \Sigma 1$

key	value
n_4	$\Sigma y, \Sigma y^2, \Sigma 1$

key	value list
n_3	$<\Sigma y, \Sigma y^2, \Sigma 1>$, $<\Sigma y, \Sigma y^2, \Sigma 1>$…

Find the best split for n_3

key	value list
n_4	$<\Sigma y, \Sigma y^2, \Sigma 1>$, $<\Sigma y, \Sigma y^2, \Sigma 1>$…

Find the best split for n_4

Graph Analysis using MapReduce

PEGASUS: Mining Peta-scale Graphs

- [Kang, Tsourakakis, Faloutsos: Knowledge and Info. Systems 2011]
- An open source peta graph mining library
  - Implemented on the top of the Hadoop
  - Use a repeated matrix-vector multiplication
  - Achieve scale-up on the number of machines and linear running time on the number of edges
- Perform typical graph mining tasks such as
  - Computing the diameter of the graph
  - Computing the radius of each node
  - Finding the connected components
  - Computing the importance score of nodes (PageRank, personalized PageRank…)

Operations in the Usual Matrix-Vector Multiplication

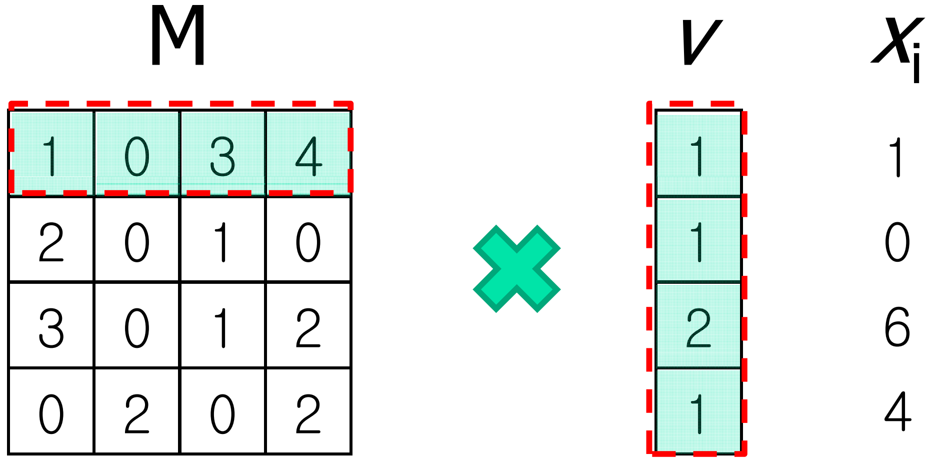
- combine2($m_{i,j}, v_j$)
  - Multiply $m_{i,j}$ and v_j
- combAll$_i$($x_1,\ldots,x_n$)
  - Sum n multiplication result for node i
- assign(v_i, v_{new})
  - Overwrite v_i with v_{new}

combine2($m_{1,1}, v_1$)=1x1=1

combine2($m_{1,2}, v_2$)=0x1=0

combine2($m_{1,3}, v_3$)=3x2=6

combine2($m_{1,4}, v_4$)=4x1=4



$$\underset{M}{\begin{bmatrix} 1 & 0 & 3 & 4 \\ 2 & 0 & 1 & 0 \\ 3 & 0 & 1 & 2 \\ 0 & 2 & 0 & 2 \end{bmatrix}} \times \underset{v}{\begin{bmatrix} 1 \\ 1 \\ 2 \\ 1 \end{bmatrix}} \qquad \underset{x_i}{\begin{matrix} 1 \\ 0 \\ 6 \\ 4 \end{matrix}}$$

Operations in the Usual Matrix-Vector Multiplication

- combine2($m_{i,j}$, v_j)
  - Multiply $m_{i,j}$ and v_j
- combAll$_i$(x_1,…,x_n)
  - Sum n multiplication result for node i
- assign(v_i, v_{new})
  - Overwrite v_i with v_{new}

combAll$_1$(x_1,x_2,x_3,x_4)
=1+0+6+4=11

assign(v_1,v_{new})=v_{new}

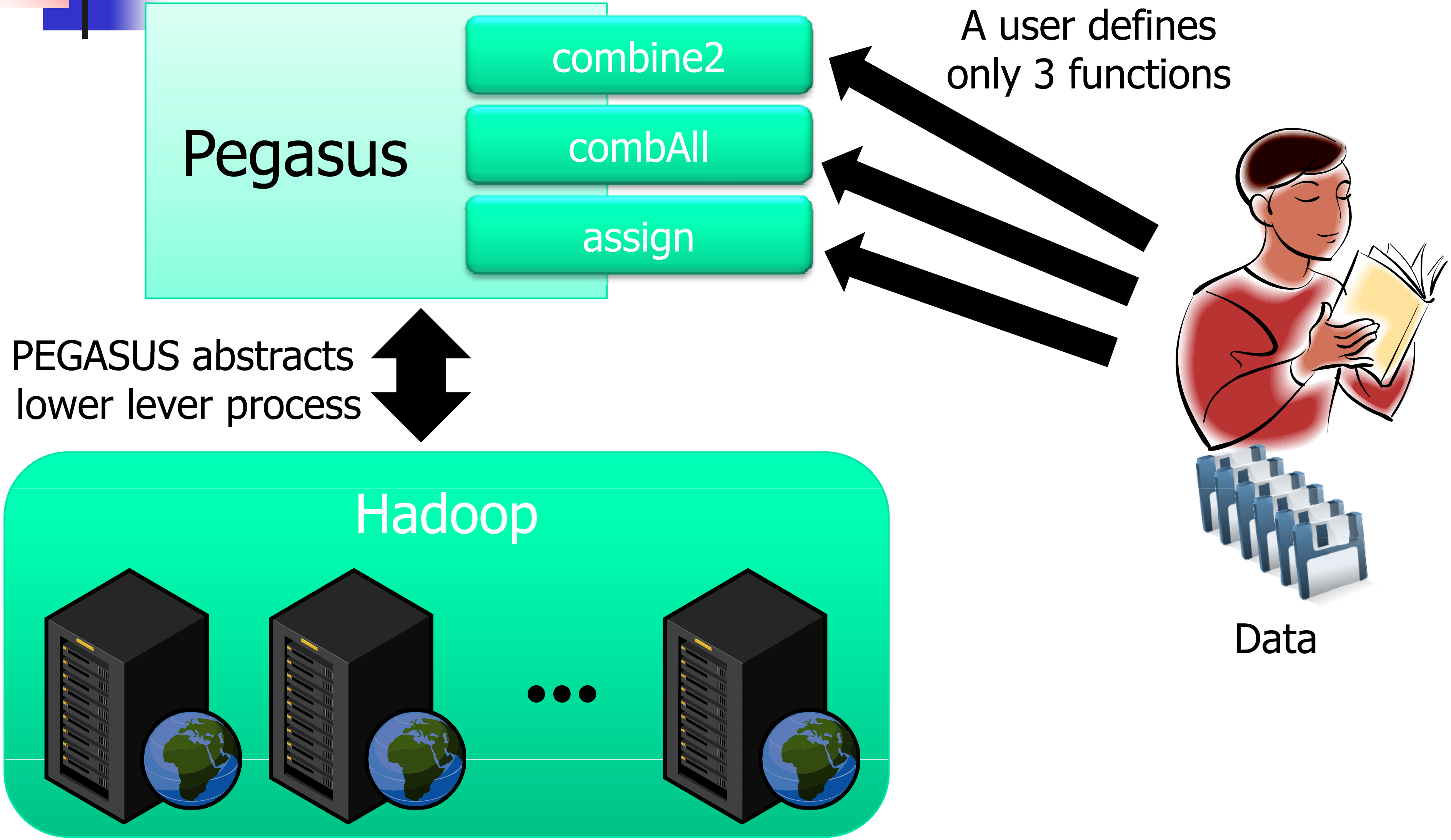
$$\begin{array}{c}M\\ \begin{array}{|c|c|c|c|}\hline 1&0&3&4\\ \hline 2&0&1&0\\ \hline 3&0&1&2\\ \hline 0&2&0&2\\ \hline\end{array}\end{array} \times \begin{array}{c}v\\ \begin{array}{|c|}\hline 1\\ \hline 1\\ \hline 2\\ \hline 1\\ \hline\end{array}\end{array} \quad \begin{array}{c}x_i\\ \begin{array}{c}1\\ 0\\ 6\\ 4\end{array}\end{array} = \begin{array}{c}v_{new}\\ \\ \end{array}$$

GIM-V: Generalized Iterative Matrix-Vector Operator

- Define the operator x_G
  - $v' = M \, \mathrm{x}_G \, v$ can be represented as below
    $v_i' = \mathrm{assign}(v_i, \mathrm{combAll}_i\{x_j \mid x_j = \mathrm{combine2}(m_{i,j}, v_j)\})$

    where
    - combine2($m_{i,j}, v_j$)
      - Combine $m_{i,j}$ and v_j
    - combAll$_i$($x_1,\ldots,x_n$)
      - Combine all the results from combine2() for node i
    - assign(v_i, v_{new})
      - Decide how to update v_i with v_{new}

How PEGASUS Works



A user defines
only 3 functions

Pegasus

combine2

combAll

assign

PEGASUS abstracts
lower lever process

Hadoop

Applications of GIM-V

- PageRank
- Random walk with restart (RWR)
- Diameter estimation
- Connected components

An Application of GIM-V to PageRank

- PageRank vector $p=(cM+(1-c)U)p$
  - c is a damping factor
  - M is a transposed adjacency matrix
  - U is a matrix with all elements set to 1/n
- Define $p_{new}=M \times_G p$ with
  - $combine2(m_{i,j}, v_j)=c \times m_{i,j} \times v_j$
  - $combAll_i(x_1,\ldots,x_n)=(1-c)/n+\sum_j x_j$
  - $assign(v_i, v_{new})=v_{new}$

c=0.85

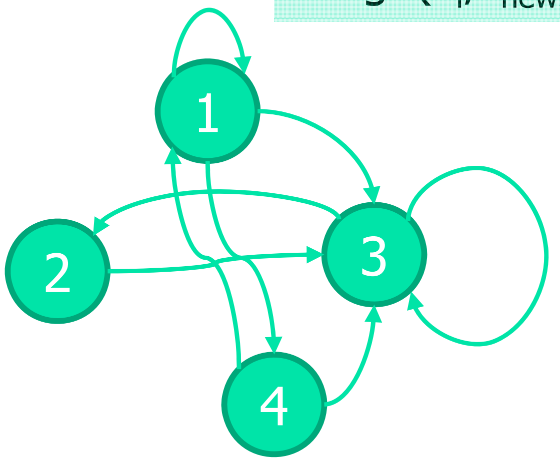
combine2(1,0.25)=c x 1 x 0.25= 0.21

combine2(0,0.25)=c x 0 x 0.25= 0

combine2(1,0.25)=c x 1 x 0.25= 0.21

combine2(1,0.25)=c x 1 x 0.25= 0.21

$combAll_1(x_1,x_2,x_3,x_4)$
$=(1-c)/n+\sum_j x_j=0.675$



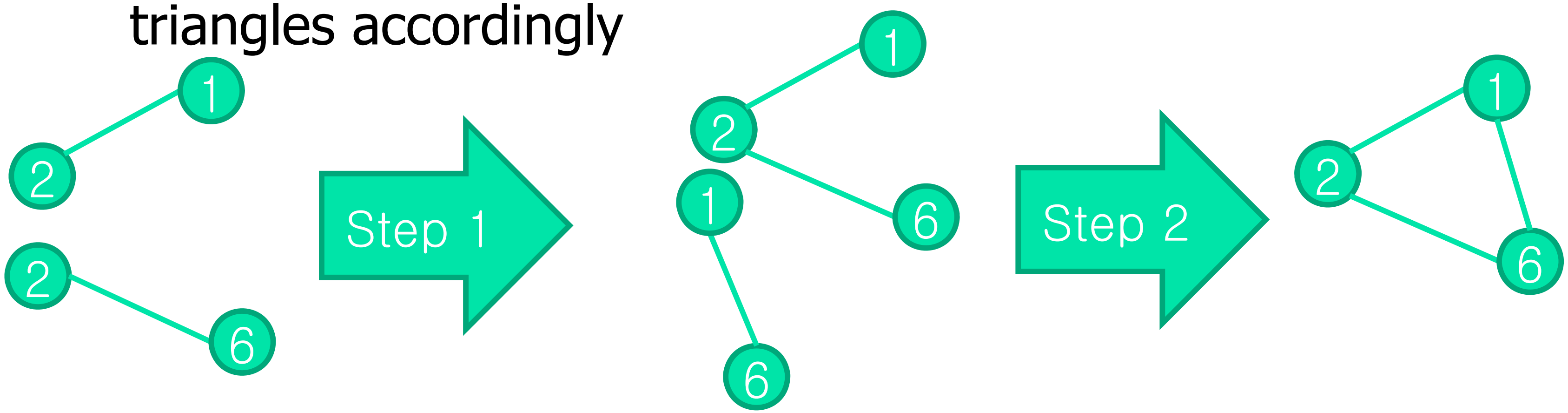
$$\begin{array}{c} M \\ \begin{bmatrix}1&0&1&1\\0&0&1&0\\0&1&1&0\\1&0&1&0\end{bmatrix}\end{array} \times \begin{array}{c} p_{old} \\ \begin{bmatrix}0.25\\0.25\\0.25\\0.25\end{bmatrix}\end{array} \begin{array}{c} x_i \\ \begin{matrix}0.21\\0\\0.21\\0.21\end{matrix}\end{array} = \begin{array}{c} p_{new} \\ \begin{matrix}0.68\\0.25\\0.46\\0.46\end{matrix}\end{array} \cdots$$

Generalized Iterative Matrix-Vector Operator Using MapReduce

- [Kang, Tsourakakis, Faloutsos: Knowledge and Info. Systems 2011]
  - Four algorithms are proposed
    - GIM-V BL: block multiplication
    - GIM-V CL: clustered edges
    - GIM-V DI: diagonal block iteration
    - GIM-V NR: node renumbering
- [Kang, Meeder, Faloutsos: PAKDD 2011]
  - For a small vector, broadcast the small vector to all map functions

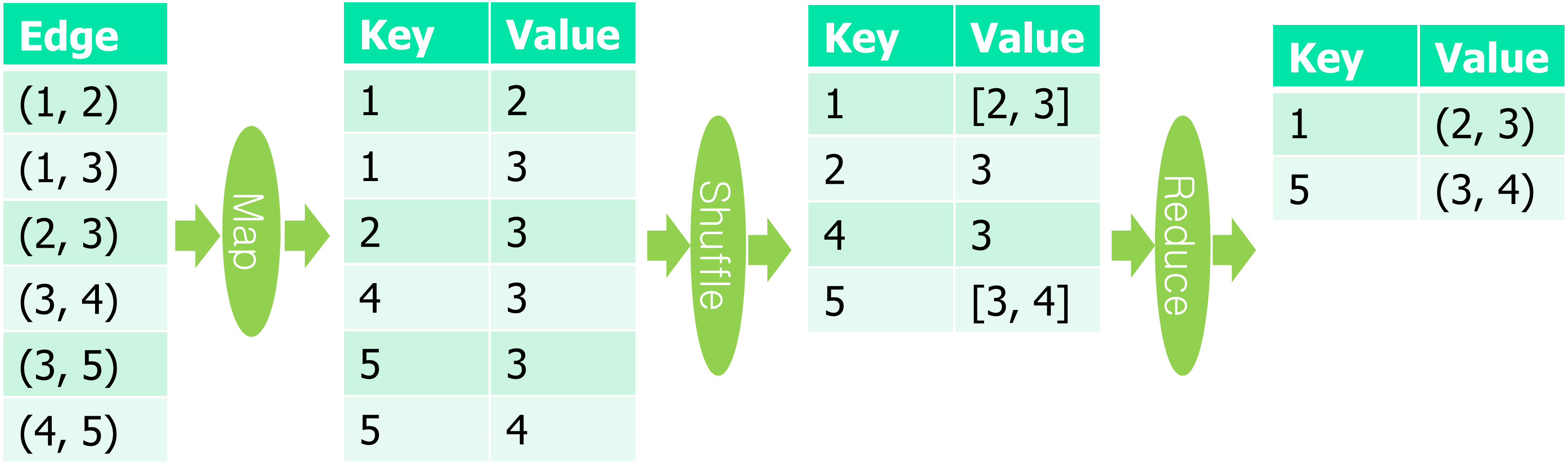
Triangle Counting Algorithm Using MapReduce

- [Suri, Vassilvitskii: WWW 2011]
- Step 1
  - Generate the possible length two paths in the graph by pivoting on every node in parallel
- Step 2
  - Check which of the length two paths generated in Step 1 can be closed by an edge in the graph and count the triangles accordingly



Triangle Counting Algorithm Using MapReduce

Ordering
$3 \succ 2 \succ 1 \succ 4 \succ 5$



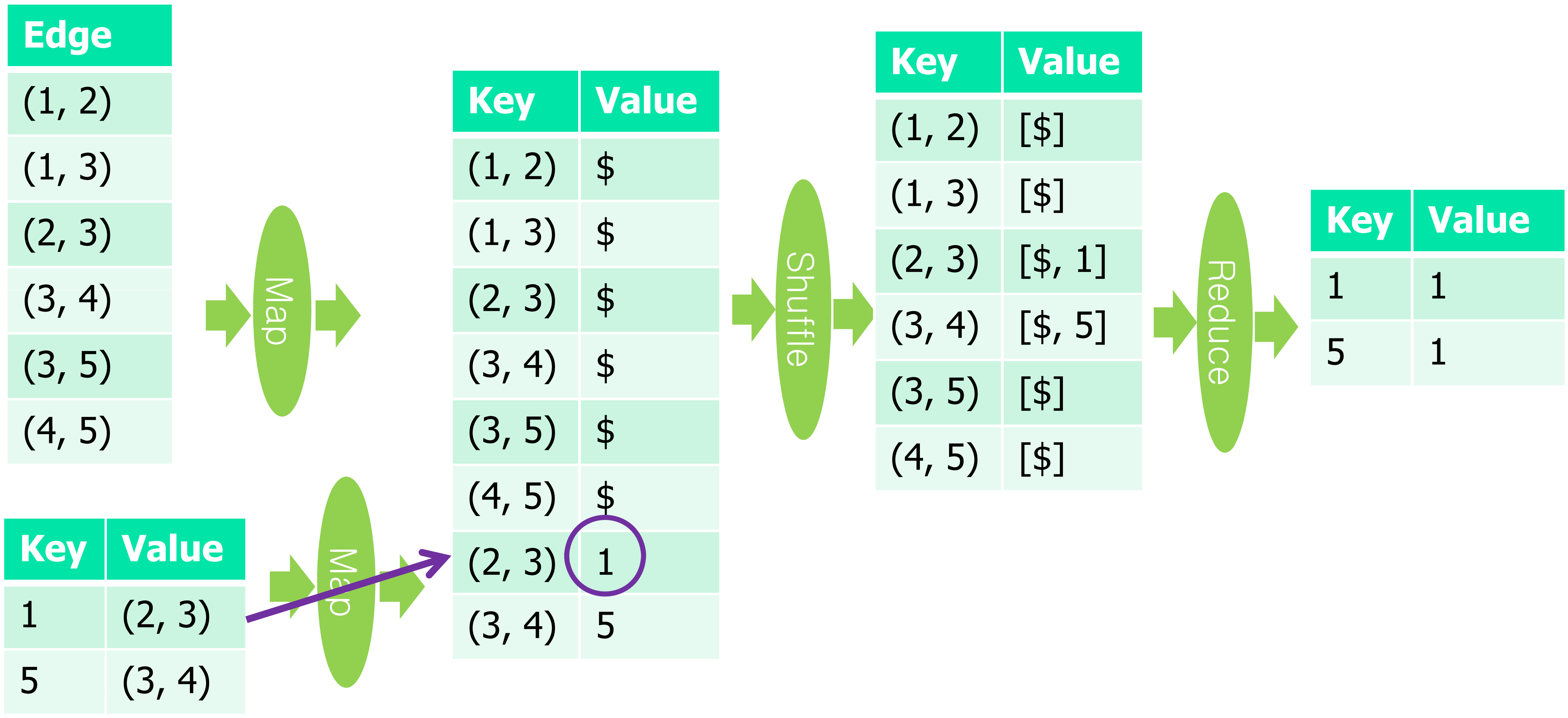
Edge
(1, 2)
(1, 3)
(2, 3)
(3, 4)
(3, 5)
(4, 5)

Key	Value
1	2
1	3
2	3
4	3
5	3
5	4

Key	Value
1	[2, 3]
2	3
4	3
5	[3, 4]

Key	Value
1	(2, 3)
5	(3, 4)

Triangle Counting Algorithm Using MapReduce



Edge
(1, 2)
(1, 3)
(2, 3)
(3, 4)
(3, 5)
(4, 5)

Key	Value
1	(2, 3)
5	(3, 4)

Key	Value
(1, 2)	$
(1, 3)	$
(2, 3)	$
(3, 4)	$
(3, 5)	$
(4, 5)	$
(2, 3)	1
(3, 4)	5

Key	Value
(1, 2)	[$]
(1, 3)	[$]
(2, 3)	[$, 1]
(3, 4)	[$, 5]
(3, 5)	[$]
(4, 5)	[$]

Key	Value
1	1
5	1

Counting of Triangles without Counting

- [Kang, Meeder, Faloutsos: PAKDD 2011]
- Compute the approximate count of the triangles by using eigenvalues and eigenvectors of the adjacency matrix [Tsourakakis: ICDM 2008]
- Use Lanczos-SO algorithm in [Lanczos: J. Res. Nat. Bur. Stand 1950] to find eigenvalues and eigenvectors
- Develop a MapReduce algorithm for Lanczos-SO

Potpourri using MapReduce

Nonnegative Matrix Factorization

- [Liu, Yang, Fan, He, Wang: WWW 2010]
- Nonnegative matrix factorization (NMF)
  - Given a user-item matrix,
  - NMF factors the matrix into a user-topic matrix and topic-item matrix
  - Frequently, used for recommendations
- They develop parallel algorithms of NMF using MapReduce

Wavelet Histogram Construction using MapReduce

- [Jestes, Yi, Feifei Li: VLDB 2012]
- Optimal and Approximate histogram construction with minimizing L_2 error measures
- Optimal algorithm is transformed to find the top-K largest normalized coefficients
- To improve the speed, approximation is proposed

Optimizing General MapReduce Programs

- [Babu: SoCC, 2010]
  - Find good job configuration parameters automatically for MapReduce code like learning optimizers
- [Jahani, Cafarella, Re`: PVLDB, 2011]
  - Detects optimization opportunities in MapReduce code, as done by a typical compiler.
  - Exploits B+-tree and compression for speed-up

Summary

- MapReduce algorithms
  - Google's MapReduce or its open-source equivalent Hadoop is a powerful developing tool
  - Recent progress for big data analysis: join algorithms, association rules, clustering, classification, probabilistic modeling, graph analysis, EM-algorithm, etc.
  - Many papers were starting to be published in major conferences
  - Still promising and rich field with many challenging research issues

Acknowledgements

- National Research Foundation of Korea
- Samsung Electronics
- SK Telecom

Thank you very much!

- Any Question?

References

- F. N. Afrati and J. D. Ullman. Optimizing Joins in a Map-Reduce Environment. EDBT, 2010
- S. Babu. Towards automatic optimization of MapReduce programs. SoCC, 2010
- R. Baraglia, G. D. F. Morales, and C. Lucchese. Document similarity self-join with mapreduce. ICDM, 2010
- S. Blanas, J. M. Patel, V. Ercegovac, J. Rao, E. J. Shekita, and Y. Tian. A comparison of join algorithms for log processing in mapreduce. SIGMOD, 2010
- H. Cao, D. Jiang, J. Pei, E. Chen, and H. Li. Towards context-aware search by learning a very large variable length hidden markov model from search logs. WWW, 2009
- D. Chakrabarti, D. Modha, S. Papadimitriou, C. Faloutsos. Fully Automatic Cross-associations. KDD, 2004
- J. F. L.-W. H. Y.-M. W. Chao Liu, Hung-chih Yang. Distributed nonnegative matrix factorization for web-scale dyadic data analysis on mapreduce. WWW, 2010
- A. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. WWW, 2007
- J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. OSDI, 2004
- S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. WWW, 2011
- Jeffery Jestes, Ke Yi, Feifei Li. Building Wavelet Histograms on Large Data in MapReduce. VLDB, 2012

References

- M. Deodhar, J. Ghosh. A Framework for Simultaneous Co-clustering and Learning from Complex Data. KDD, 2007
- M. Deodhar, C. Jones, and J. Ghosh. Parallel simultaneous co-clustering and learning with map-reduce. GrC, 2000
- T. Elsayed, J. Lin, , and D. W. Oard. Pairwise document similarity in large collections with mapreduce. In HLT, 2008
- Y. He, H. Tan, W. Luo, H. Mao, D. Ma, S. Feng, and J. Fan. Mr-dbscan: An efficient parallel density-based clustering algorithm using mapreduce. ICPADS, 2011
- H. Herodotou and S. Babu. Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs. PVLDB, 2011
- E. Jahani, M. J. Cafarella, C. Re'. Automatic Optimization for MapReduce Programs. PVLDB, 2011
- U. Kang, B. Meeder, and C. Faloutsos. Spectral analysis for billion-scale graphs: Discoveries and implementation. PAKDD, 2011
- U. Kang, C. E. Tsourakakis, and C. Faloutsos. PEGASUS: mining peta-scale graphs. Knowledge and Information Systems, 27(2), 2011
- T. Schank. Algorithmic aspects of triangle-based network analysis. PhD thesis, Universitat Karlsruhe, 2007

References

- Y. Kim and K. Shim. TWITOBI: A recommendation system for twitter using probabilistic modeling. ICDM, 2011
- Y. Kim and K. Shim. Parallel top-k similarity join algorithms using mapreduce. ICDE, 2012
- Lee, R., Luo, T., Huai, Y., Wang, F., He, Y., Zhang, X. YSmart: Yet Another SQL-to-MapReduce Translator. ICDCS, 2011
- H. Li, Y. Wang, D. Zhang, M. Zhang, and E. Chang. PFP: Parallel FP-Growth for query recommendation. ACM Recommender Systems, 2008
- A. McCallum, K. Nigam, and L. H. Ungar. Efficient clustering of high-dimensional data sets with application to reference matching. In KDD, 2000
- A. Metwally and C. Faloutsos. V-SMART-Join: A scalable mapreduce framework for all-pair similarity joins of multisets and vectors. In VLDB, 2012
- A. Okcan and M. Riedewald. Processing theta-joins using mapreduce. SIGMOD, 2011
- B. Panda, J. S. Herbach, S. Basu, and R. J. Bayardo. Planet: Massively parallel learning of tree ensembles with mapreduce. VLDB, 2012
- S. Papadimitriou and J. Sun. DisCo: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. ICDM, 2008
- T. Sun, C. Shuy, F. Liy, H. Yuy, L. Ma, and Y. Fang. An efficient hierarchical clustering method for large datasets with map-reduce. PDCAT, 2009

References

- A. Thusoo, R. Murthy, J. S. Sarma, Z. Shao, N. Jain, P. Chakka, S. Anthony, H. Liu, and N. Zhang. Hive - a petabyte scale data warehousing using hadoop. In ICDE, 2010
- R. Vernica, M. J. Carey, and C. Li. Efficient parallel set-similarity joins using mapreduce. SIGMOD, 2010
- Y. Wang, H. Bai, M. Stanton, W.-Y. Chen, and E. Y. Chang. PLDA: Parallel latent dirichlet allocation for large-scale applications. AAIM, 2009
- S. Wu, F. Li, S. Mehrotra, B. C. Ooi. Query Optimization for Massively Parallel Data Processing. SOCC, 2011
- Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu: Efficient similarity joins for near duplicate detection. WWW 2008
- K. Zhai, J. L. Boyd-Graber, N. Asadi, and M. L. Alkhouja. Mr. LDA: A flexible large scale topic modeling package using variational inference in mapreduce. WWW, 2012
- David Newman, Arthur Asuncion, Padhraic Smyth and Max Welling: Distributed inference for latent Dirichlet allocation. NIPS 2007
- C.E. Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws, In ICDM, 2008
- C. Lanczos. An iteration method for the solution of the eigenvalue problem of linear differential and integral operators, J. Res. Nat. Bur. Stand, 1950