MapReduce Algorithms

Sergei Vassilvitskii

A Sense of Scale

At web scales...

- Mail: Billions of messages per day
- Search: Billions of searches per day
- Social: Billions of relationships

A Sense of Scale

At web scales...

- Mail: Billions of messages per day
- Search: Billions of searches per day
- Social: Billions of relationships

...even the simple questions get hard

- What are the most popular search queries?
- How long is the shortest path between two friends?
- ...

To Parallelize or Not?

Distribute the computation

- Hardware is (relatively) cheap
- Plenty of parallel algorithms developed

To Parallelize or Not?

Distribute the computation

- Hardware is (relatively) cheap
- Plenty of parallel algorithms developed

But parallel programming is hard

- Threaded programs are difficult to test. One successful run is not enough
- Threaded programs are difficult to read, because you need to know in which thread each piece of code could execute
- Threaded programs are difficult to debug. Hard to repeat the conditions to find bugs
- More machines means more breakdowns

MapReduce

MapReduce makes parallel programming easy

- Tracks the jobs and restarts if needed
- Takes care of data distribution and synchronization

But there's no free lunch:

- Imposes a structure on the data
- Only allows for certain kinds of parallelism

MapReduce Setting

Data:

- “Which search queries co-occur?”
- “Which friends to recommend?”
- Data stored on disk or in memory

Computation:

- Many commodity machines

MapReduce Basics

Data:

- Represented as <Key, Value> pairs

Example: A Graph is a list of edges

- Key = (u,v)
- Value = edge weight

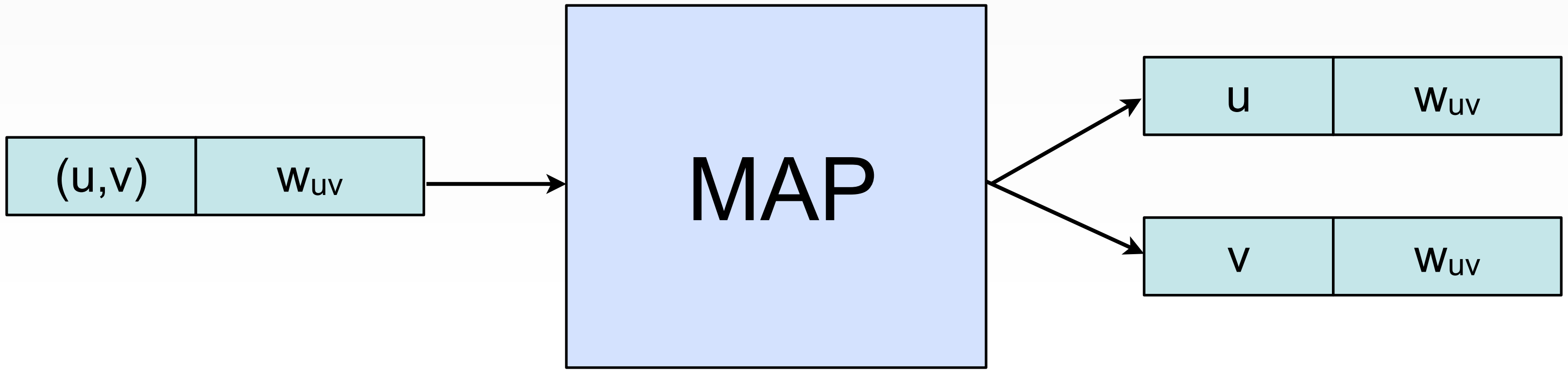
MapReduce Basics

Data:

- Represented as <Key, Value> pairs

Operations:

- Map: <Key, Value> → List(<Key, Value>)
  - Example: Split all of the edges



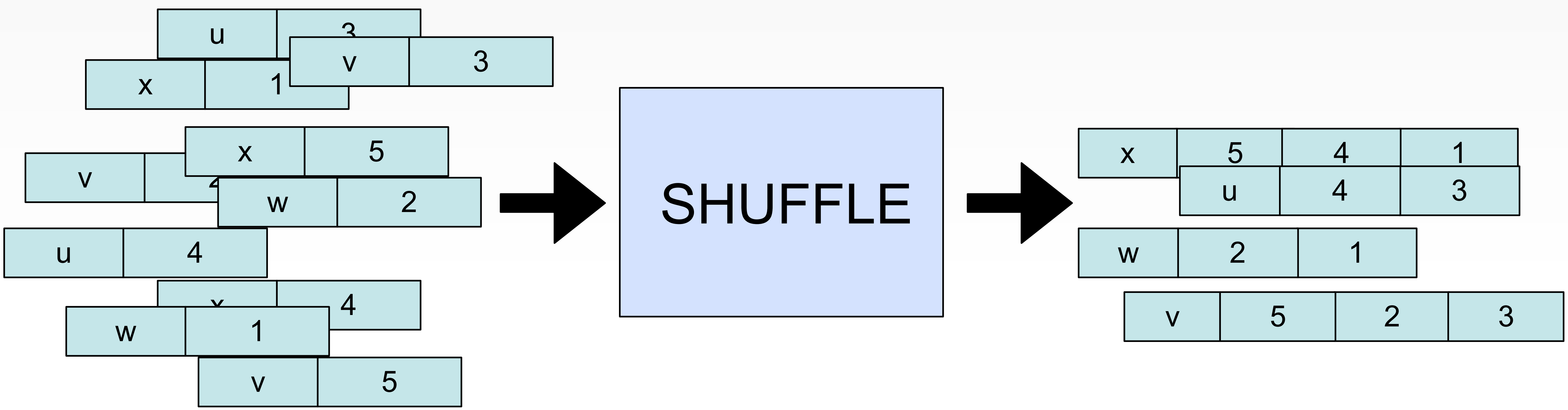
MapReduce Basics

Data:

- Represented as <Key, Value> pairs

Operations:

- Map: <Key, Value> → List(<Key, Value>)
- Shuffle: Aggregate all pairs with the same key



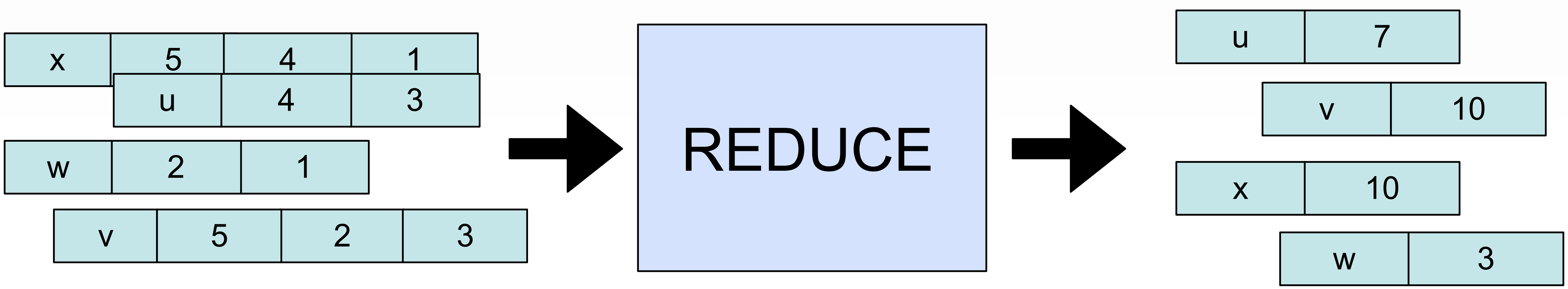
MapReduce Basics

Data:

- Represented as <Key, Value> pairs

Operations:

- Map: <Key, Value> → List(<Key, Value>)
- Shuffle: Aggregate all pairs with the same key
- Reduce: <Key, List(Value)> → <Key, List(Value)>
  - Example: Add values for each key



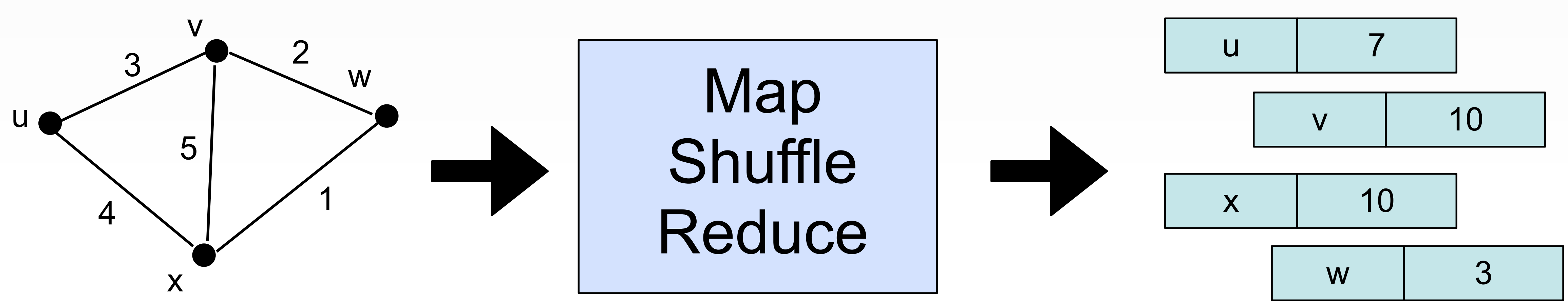
MapReduce Basics

Data:

- Represented as <Key, Value> pairs

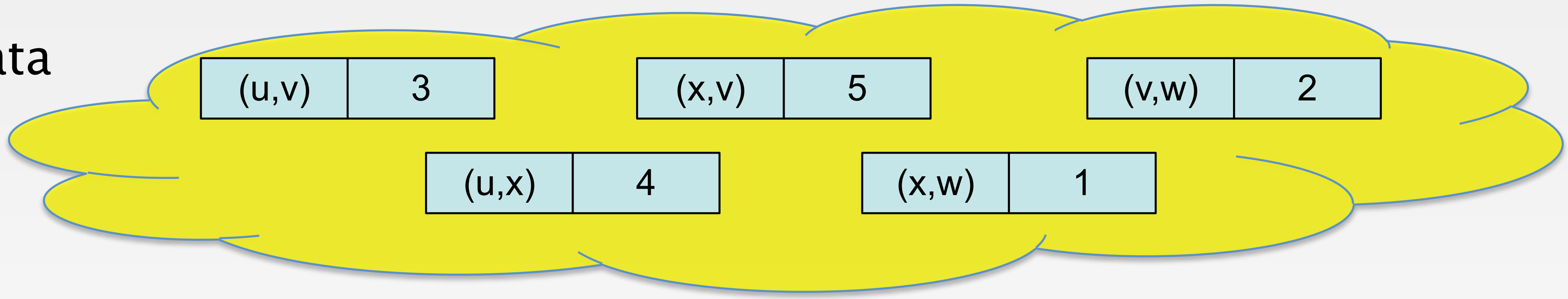
Operations:

- Map: <Key, Value> → List(<Key, Value>)
- Shuffle: Aggregate all pairs with the same key
- Reduce: <Key, List(Value)> → <Key, List(Value)>



MapReduce (Data View)

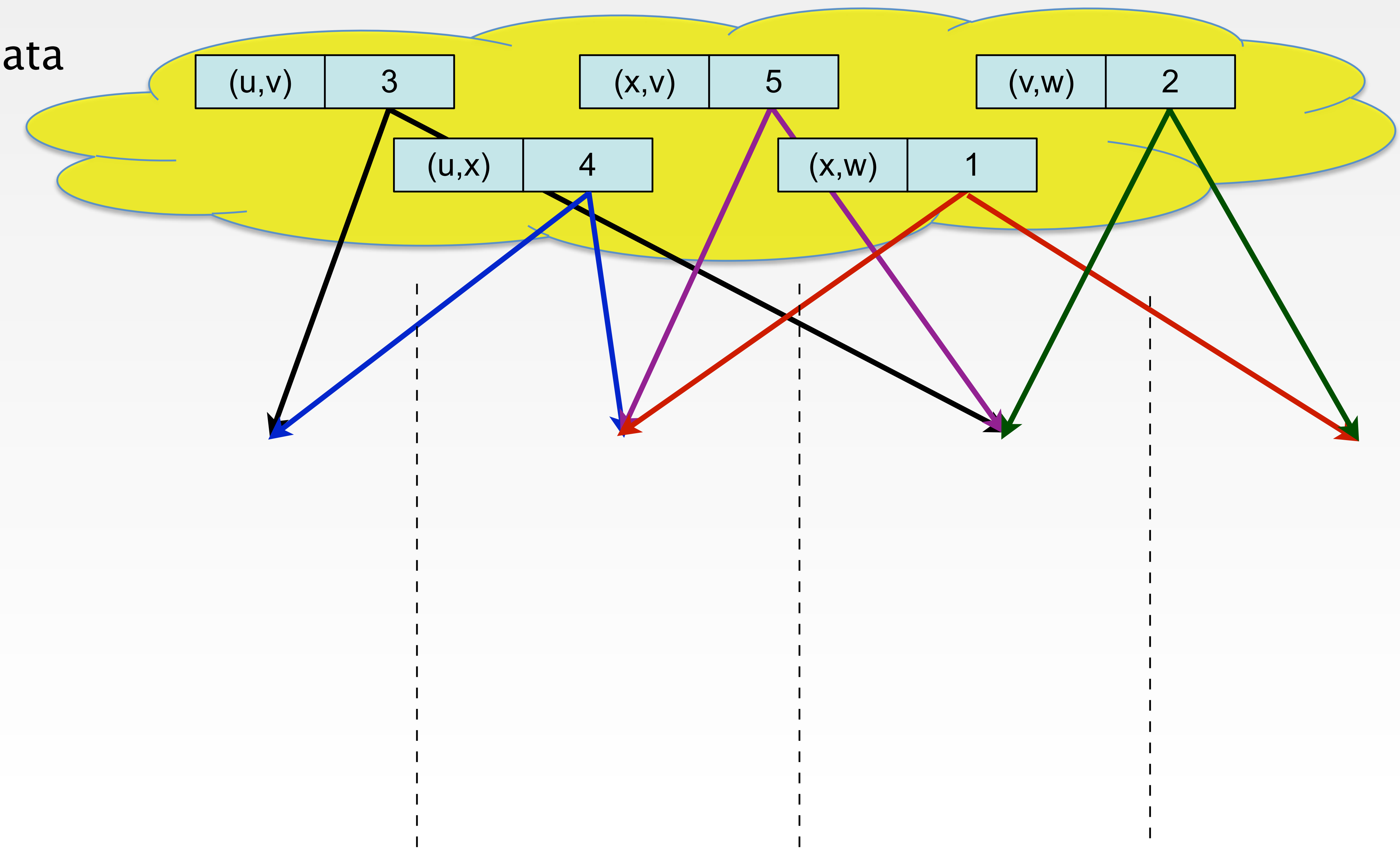
Unordered Data



MapReduce (Data View)

Unordered Data

Map

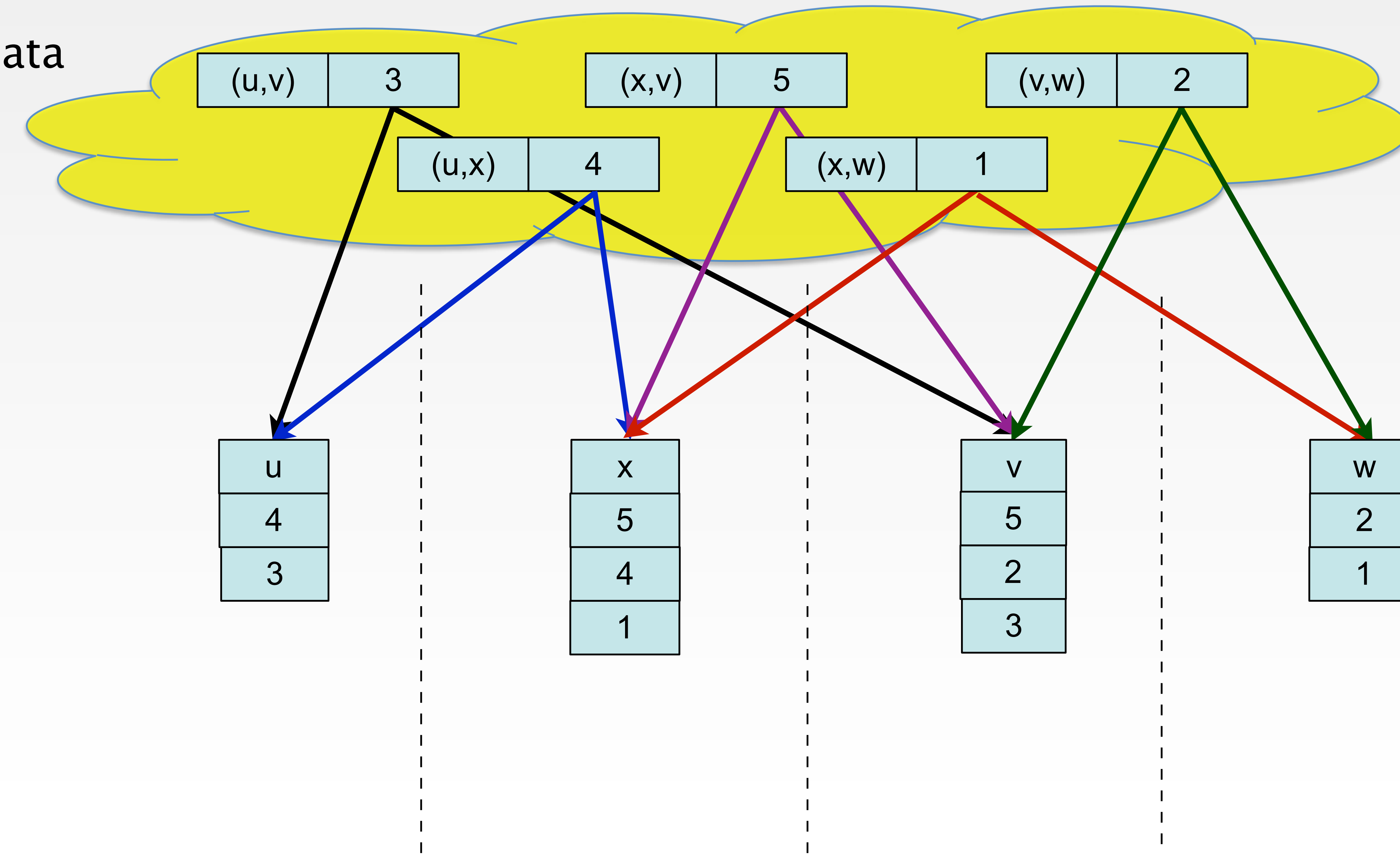


MapReduce (Data View)

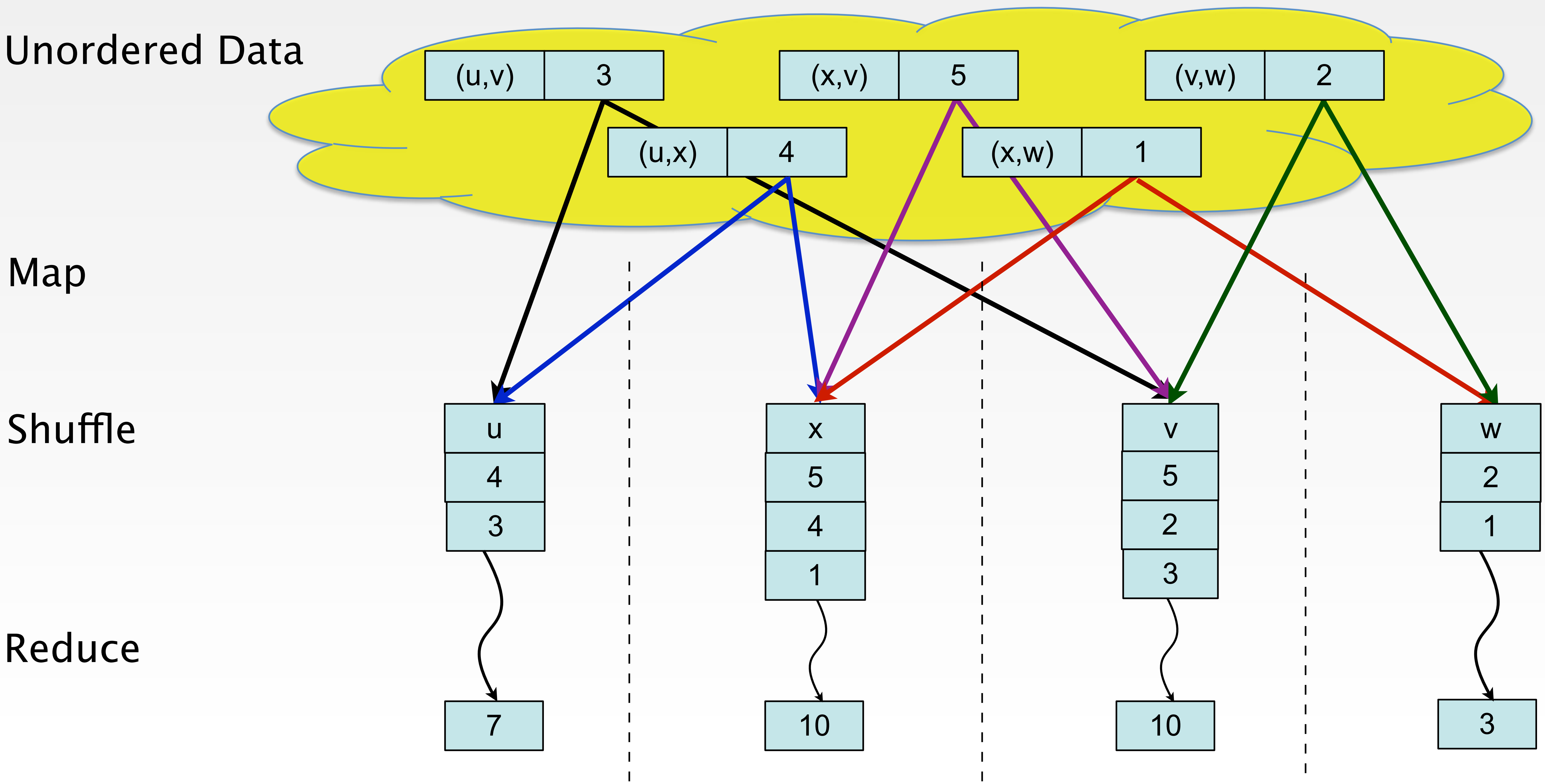
Unordered Data

Map

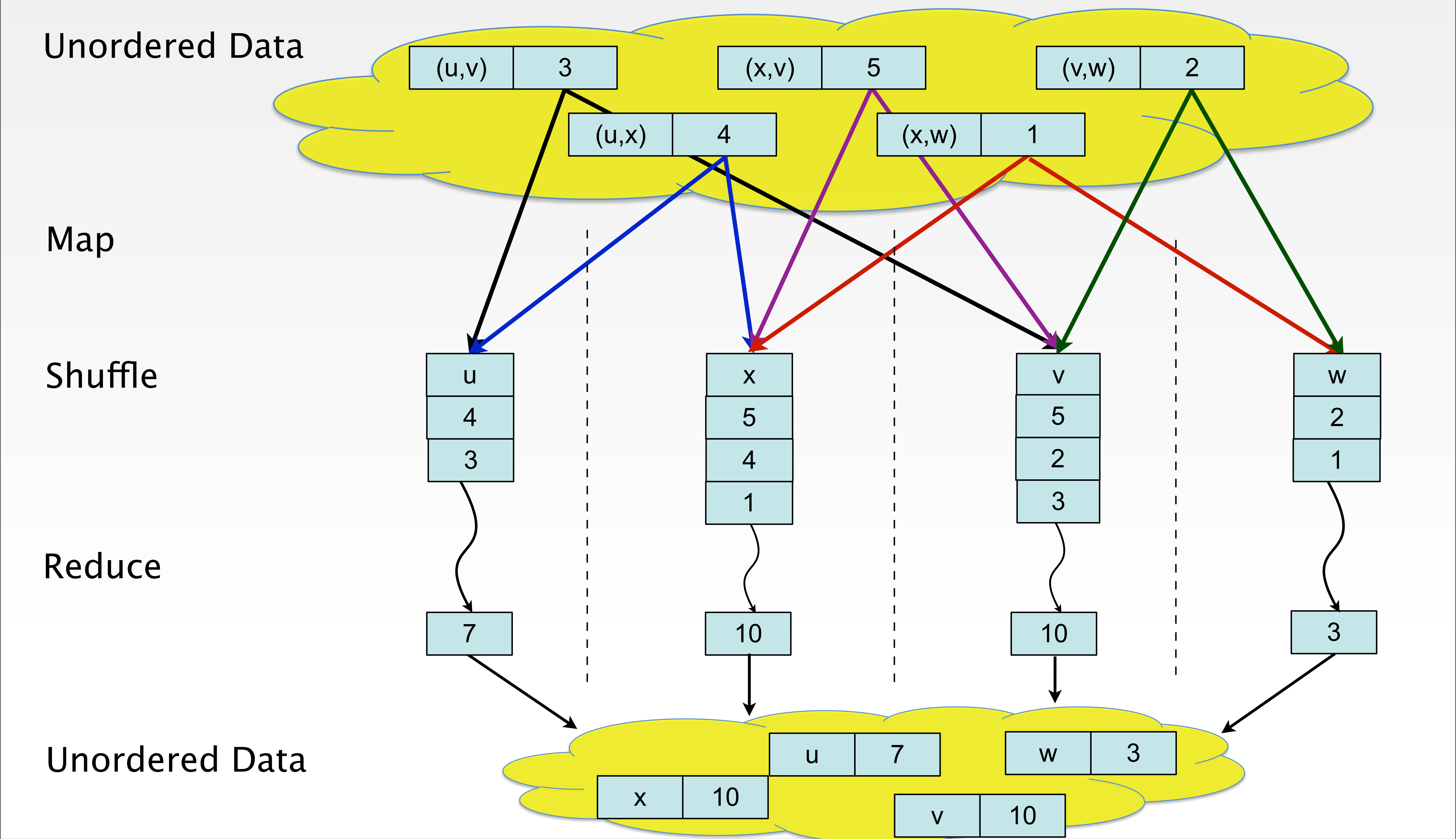
Shuffle



MapReduce (Data View)



MapReduce (Data View)



Matrix Transpose

Given a sparse matrix in row major order

Output same matrix in column major order

Given:

row 1	(col 1, a)	(col 2, b)
row 2	(col 2, c)	(col 3, d)
row 3	(col 2, e)	

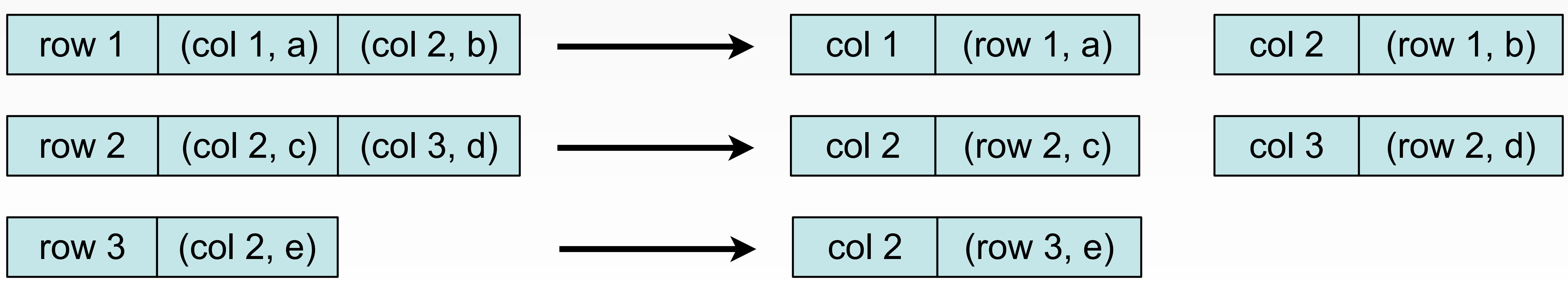
a	b	
	c	d
	e	

Matrix Transpose

Map:

- Input: <row i, (col$_{i1}$, val$_{i1}$), (col_$_{i2}$, val$_{i2}$), ... >
- Output: <col$_{i1}$, (row i, val$_{i1}$)>
- <col$_{i2}$, (row i, val$_{i2}$)>
- ....

a	b	
	c	d
	e	



Saturday, August 25, 12

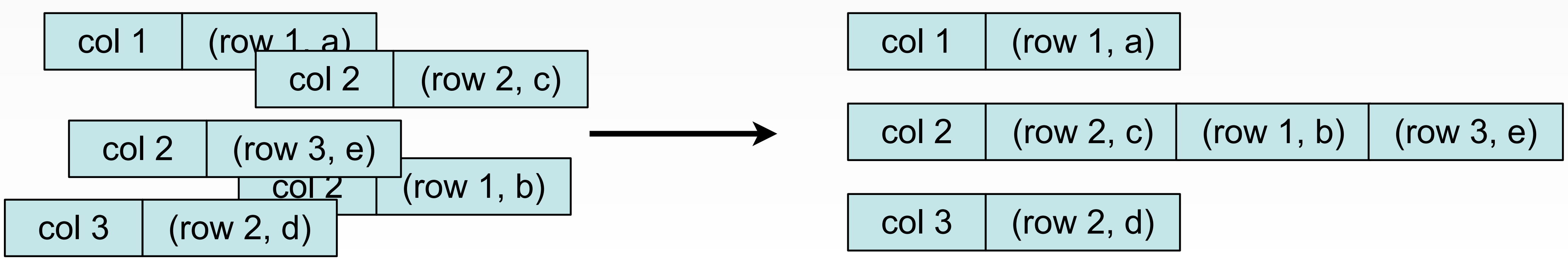
Matrix Transpose

a	b	
	c	d
	e	

Map:

- Input: <row i, (col$_{i1}$, val$_{i1}$), (col_$_{i2}$, val$_{i2}$), ... >
- Output: <col$_{i1}$, (row i, val$_{i1}$)>
- <col$_{i2}$, (row i, val$_{i2}$)>
- ....

Shuffle:



Matrix Transpose

a	b	
	c	d
	e	

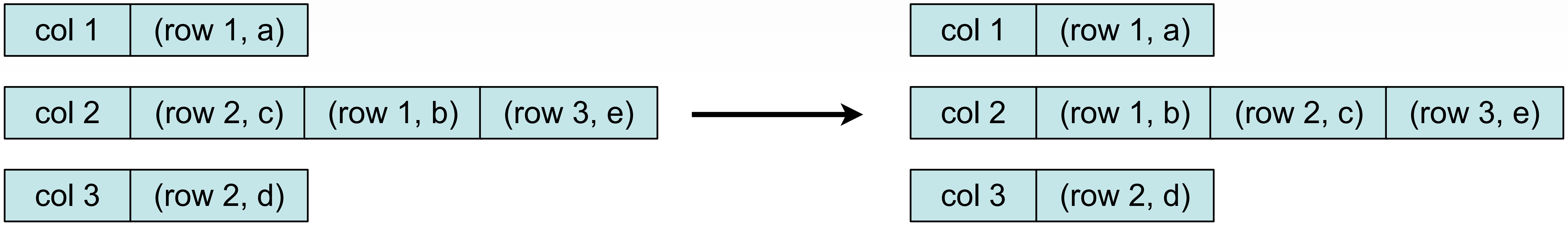
Map:

- Input: <row i, (col$_{i1}$, val$_{i1}$), (col_$_{i2}$, val$_{i2}$), ... >
- Output: <col$_{i1}$, (row i, val$_{i1}$)>
- <col$_{i2}$, (row i, val$_{i2}$)>
- ....

Shuffle

Reduce:

- Sort by row number



Saturday, August 25, 12

Matrix Transpose

Given a sparse matrix in row major order

Output same matrix in column major order

Given:

row 1	(col 1, a)	(col 2, b)
row 2	(col 2, c)	(col 3, d)
row 3	(col 2, e)	

a	b	
	c	d
	e	

Output:

col 1	(row 1, a)		
col 2	(row 1, b)	(row 2, c)	(row 3, e)
col 3	(row 2, d)		

MapReduce Implications

Operations:

- Map: <Key, Value> → List(<Key, Value>)
  - Can be executed in parallel for each pair.

- Shuffle: Aggregate all pairs with the same Key
  - Synchronization step

- Reduce: <Key, List(Value)> → <Key, List(Value)>
  - Can be executed in parallel for each Key

MapReduce Implications

Operations:

- Map: <Key, Value> → List(<Key, Value>)
  - Can be executed in parallel for each pair
  - Provided by the programmer
- Shuffle: Aggregate all pairs with the same Key
  - Synchronization step
  - Handled by the system
- Reduce: <Key, List(Value)> → <Key, List(Value)>
  - Can be executed in parallel for each Key
  - Provided by the programmer

The system also:

- Makes sure the data is local to the machine
- Monitors and restarts the jobs as necessary

MapReduce Implications

Operations:

- Map: <Key, Value> → List(<Key, Value>)
  - Can be executed in parallel for each pair
  - Provided by the programmer
- Shuffle: Aggregate all pairs with the same Key
  - Synchronization step
  - Handled by the system
- Reduce: <Key, List(Value)> → <Key, List(Value)>
  - Can be executed in parallel for each Key
  - Provided by the programmer

High Level view: MapReduce is about locality

- Map: Assign data to different machines to ensure locality
- Reduce: Sequential computation on local data blocks

Trying MapReduce

Hadoop:

- Open source version of MapReduce
- Can run locally

Amazon Web Services

- Upload datasets, run jobs
- Run jobs ... (Careful: pricing round to nearest hour, so debug first!)

Outline

1. What is MapReduce?
2. Modeling MapReduce
3. Dealing with Data Skew

Modeling MapReduce

	Memory	
	Polynomial	Sublinear
	RAM	Sketches External Memory Property Testing

Modeling MapReduce

		Memory: Polynomial	Memory: Sublinear
Processors	Single	RAM	Sketches External Memory Property Testing
	Multiple	PRAM	

Modeling MapReduce

Processors \ Memory	Polynomial	Sublinear
Single	RAM	Sketches External Memory Property Testing
Multiple	PRAM	MapReduce Distributed Sketches

MapReduce vs. Data Streams

	Input	
	Batch	Online
	RAM	Data Streams

MapReduce vs. Data Streams

		Input: Batch	Input: Online
Processors	Single	RAM	Data Streams
	Multiple	MapReduce	Distributed Sketches Active DHTs

The World of MapReduce

The World of MapReduce

Practice:

- Used very widely for big data analysis

Aside: Big Data

Aside: Big Data

Small Data:

- Mb sized inputs
- Quadratic algorithms finish quickly

Aside: Big Data

Small Data:

- Mb sized inputs
- Quadratic algorithms finish quickly

Medium Data:

- Gb sized inputs
- Aim for linear time algorithms

Aside: Big Data

Small Data:

- Mb sized inputs
- Quadratic algorithms finish quickly

Medium Data:

- Gb sized inputs
- Aim for linear time algorithms

Big Data:

- Tb+ sized inputs
- Need parallel algorithms

The World of MapReduce

The World of MapReduce

Practice:

- Used very widely for big data analysis
- Google, Yahoo!, Amazon, Facebook, LinkedIn, ...

The World of MapReduce

Practice:

- Used very widely for big data analysis
- Google, Yahoo!, Amazon, Facebook, LinkedIn, ...

Beyond Simple MR:

- Many similar implementations and abstractions on top of MR: Hadoop, Pig, Hive, Flume, Pregel, ...
- Same computational model underneath

The World of MapReduce

Practice:

- Used very widely for big data analysis
- Google, Yahoo!, Amazon, Facebook, LinkedIn, ...

Beyond Simple MR:

- Many similar implementations and abstractions on top of MR: Hadoop, Pig, Hive, Flume, Pregel, ...
- Same computational model underneath

Data Locality:

- Underscores the fact that data locality is crucial...
- ....which sometimes leads to faster sequential algorithms !

MapReduce: Overview

Multiple Processors:

- 10s to 10,000s processors

Sublinear Memory

- A few Gb of memory/machine, even for Tb+ datasets
- Unlike PRAMs: memory is not shared

Batch Processing

- Analysis of existing data
- Extensions used for incremental updates, online algorithms

Data Streams vs. MapReduce

Distributed Sum:

- Given a set of n numbers: $a_1, a_2, \ldots, a_n \in \mathbb{R}$, find $S = \sum_i a_i$

Data Streams vs. MapReduce

Distributed Sum:

- Given a set of n numbers: $a_1, a_2, \ldots, a_n \in \mathbb{R}$, find $S = \sum_i a_i$

Stream:

- Maintain a partial sum $S_j = \sum_{i \le j} a_i$
- update with every element

Data Streams vs. MapReduce

Distributed Sum:

- Given a set of n numbers: $a_1, a_2, \ldots, a_n \in \mathbb{R}$, find $S = \sum_i a_i$

Stream:

- Maintain a partial sum $S_j = \sum_{i \leq j} a_i$
- update with every element

MapReduce:

- Compute $M_j = a_{jk} + a_{jk+1} + \ldots + a_{j(k+1)-1}$ for $k = \sqrt{n}$ in Round 1
- Round 2: add the $\sqrt{n}$ partial sums.

Modeling

For an input of size n :

Modeling

For an input of size n :

Memory

- Cannot store the data in memory
- Insist on sublinear memory per machine: $O(n^{1-\epsilon})$ for some $\epsilon > 0$

Modeling

For an input of size n :

Memory

- Cannot store the data in memory
- Insist on sublinear memory per machine: $O(n^{1-\epsilon})$ for some $\epsilon > 0$

Machines

- Machines in a cluster do not share memory
- Insist on sublinear number of machines: $O(n^{1-\epsilon})$ for some $\epsilon > 0$

Modeling

For an input of size n :

Memory

- Cannot store the data in memory
- Insist on sublinear memory per machine: $O(n^{1-\epsilon})$ for some $\epsilon > 0$

Machines

- Machines in a cluster do not share memory
- Insist on sublinear number of machines: $O(n^{1-\epsilon})$ for some $\epsilon > 0$

Synchronization

- Computation proceeds in rounds
- Count the number of rounds
- Aim for $O(1)$ rounds

Not Modeling

Communication:

- Very important, makes a big difference

Not Modeling

Communication:

- Very important, makes a big difference
- Order of magnitude improvements due to
  - Move code to data (and not data to code)
  - Working with graphs: save graph structure locally between rounds
  - Job scheduling (same rack / different racks, etc)

Not Modeling

Communication:

- Very important, makes a big difference
- Order of magnitude improvements due to
  - Move code to data (and not data to code)
  - Working with graphs: save graph structure locally between rounds
  - Job scheduling (same rack / different racks, etc)
- Bounded by $n^{2-2\epsilon}$ (total memory of the system) in the model
  - Minimizing communication always a goal

How Powerful is this Model?

Different Tradeoffs from PRAM:

- PRAM: LOTS of very simple cores, communication every round
- PRAM: Worry less about data locality
- MR: Many real cores (Turing Machines), batch communication.

How Powerful is this Model?

Different Tradeoffs from PRAM:

- PRAM: LOTS of very simple cores, communication every round
- PRAM: Worry less about data locality
- MR: Many real cores (Turing Machines), batch communication.

Formally:

- Can simulate PRAM algorithms with MR
- In practice can use same idea without formal simulation
- One round of MR per round of PRAM: $O(\log n)$ rounds total
- Hard to break below $o(\log n)$, need new ideas!

How Powerful is this Model?

Different Tradeoffs from PRAM:

- PRAM: LOTS of very simple cores, communication every round
- PRAM: Worry less about data locality
- MR: Many real cores (Turing Machines), batch communication.

Formally:

- Can simulate PRAM algorithms with MR
- In practice can use same idea without formal simulation
- One round of MR per round of PRAM: $O(\log n)$ rounds total
- Hard to break below $o(\log n)$, need new ideas!

Both Approaches:

- Synchronous: computation proceeds in rounds
- Other abstractions (e.g. GraphLab are asynchronous)

How Powerful is this Model?

Compared to Data Streams:

- Solving different problems (batch vs. online)
- But can use similar ideas (e.g. sketching)

How Powerful is this Model?

Compared to Data Streams:

- Solving different problems (batch vs. online)
- But can use similar ideas (e.g. sketching)

Compared to BSP:

- Closest in spirit
- Do not optimize parameters in algorithm design phase
- Most similar to the CGP: Coarse Grained Parallel approach

Outline

1. What is MapReduce?
2. Modeling MapReduce
3. Dealing with Data Skew

(Social) Graph Mining

(Social) Graph Mining

Graphs:

- Web (directed, labeled edges)
- Friendship (undirected, potentially labeled edges)
- Follower (directed, unlabeled edges)
- ..

(Social) Graph Mining

Graphs:

- Web (directed, labeled edges)
- Friendship (undirected, potentially labeled edges)
- Follower (directed, unlabeled edges)
- ..

(Social) Graph Mining

Graphs:

- Web (directed, labeled edges)
- Friendship (undirected, potentially labeled edges)
- Follower (directed, unlabeled edges)
- ..

Questions:

- Identify tight-knit circles of friends (Today)
- Identify large communities (Tomorrow)

Defining Tight Knit Circles

Defining Tight Knit Circles

Looking for tight-knit circles:

- People whose friends are friends themselves

Defining Tight Knit Circles

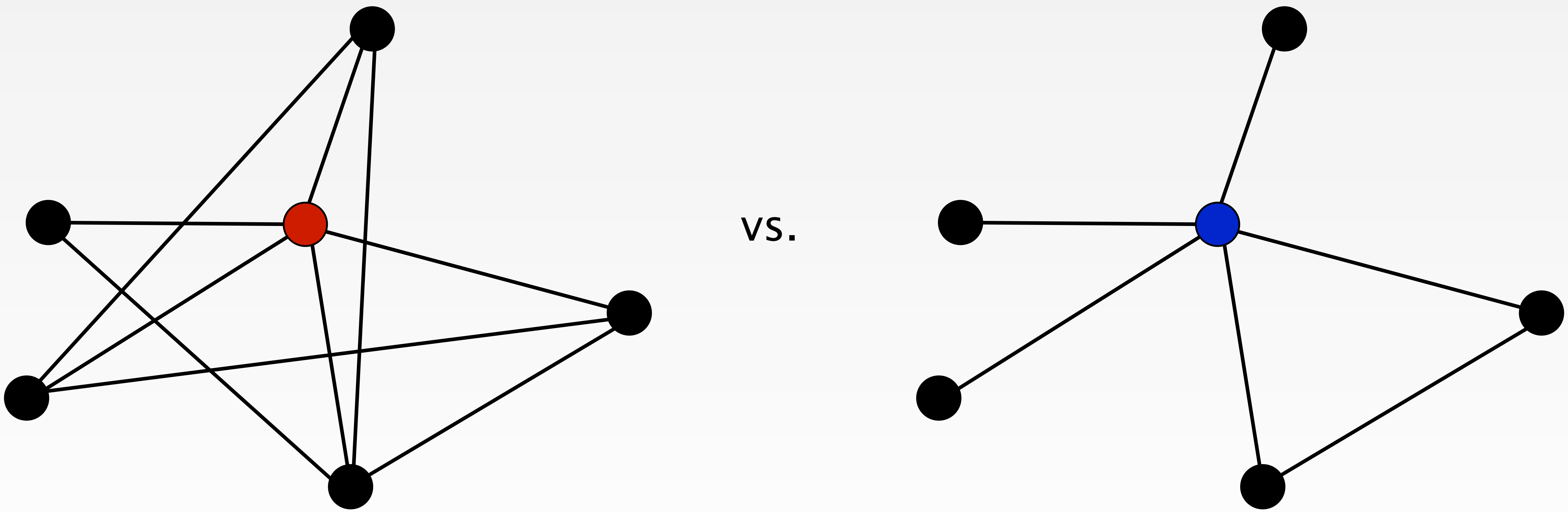
Looking for tight-knit circles:

- People whose friends are friends themselves

Why?

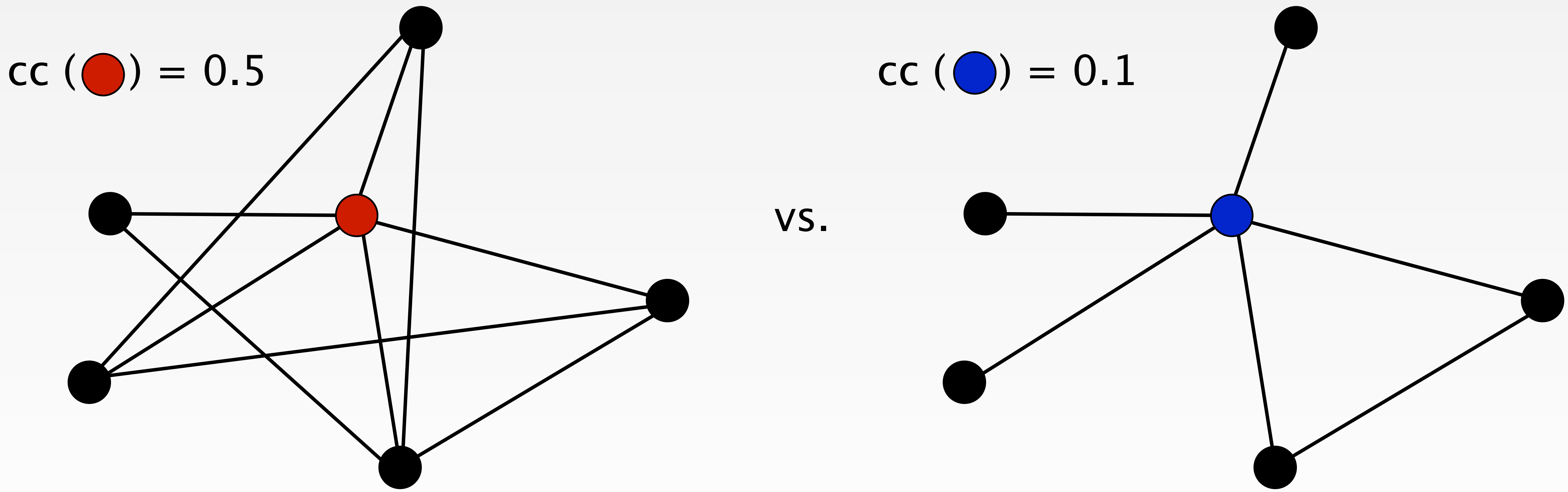
- Network Cohesion: Tightly knit communities foster more trust, social norms. [Coleman '88, Portes '88]
- Structural Holes: Individuals benefit form bridging [Burt '04, '07]

Clustering Coefficient



Saturday, August 25, 12

Clustering Coefficient



Given an undirected graph $G = (V, E)$

cc(v) = fraction of v's neighbors who are neighbors themselves

$$= \frac{|\{(u,w) \in E | u \in \Gamma(v) \wedge w \in \Gamma(v)\}|}{\binom{d_v}{2}} = \frac{\#\Delta s \text{ incident on } v}{\binom{d_v}{2}}$$

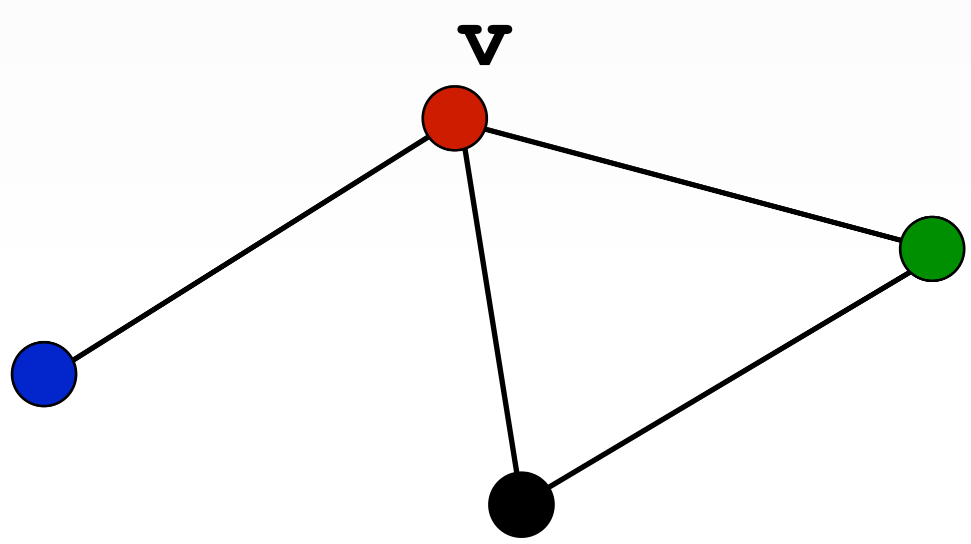
Saturday, August 25, 12

How to Count Triangles

How to Count Triangles

Sequential Version:

```
foreach v in V
    foreach u,w in Adjacency(v)
        if (u,w) in E
            Triangles[v]++
```

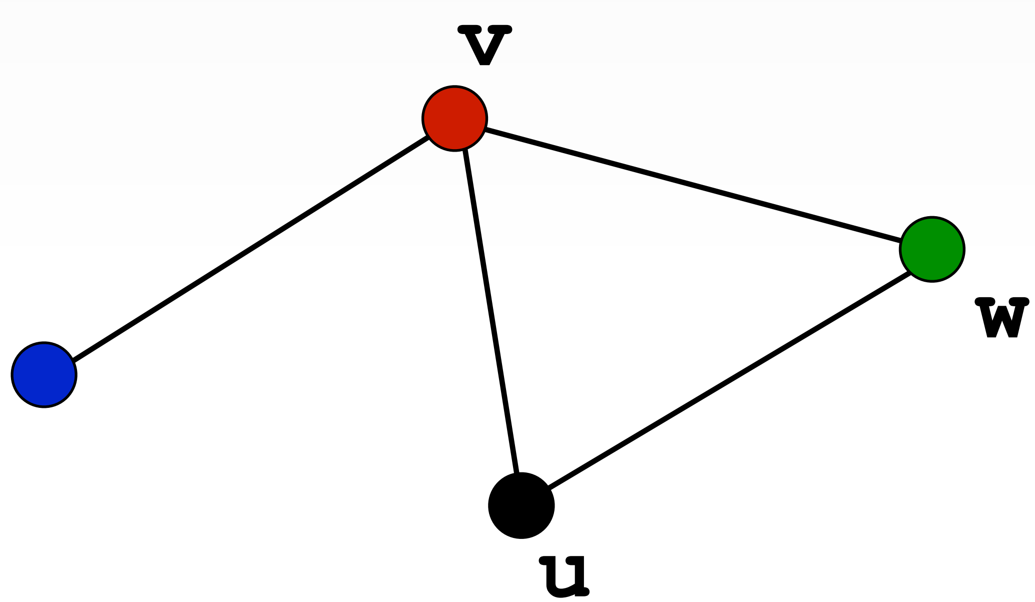


Triangles[v]=0

How to Count Triangles

Sequential Version:

```
foreach v in V
    foreach u,w in Adjacency(v)
        if (u,w) in E
            Triangles[v]++
```

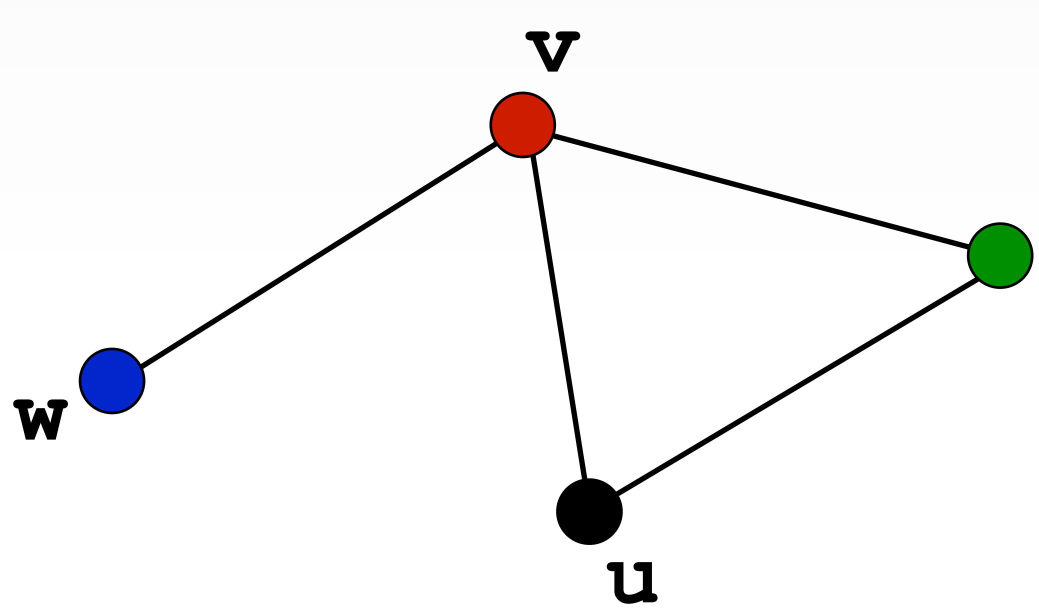


Triangles[v]=1

How to Count Triangles

Sequential Version:

```
foreach v in V
    foreach u,w in Adjacency(v)
        if (u,w) in E
            Triangles[v]++
```



Triangles[v]=1

How to Count Triangles

Sequential Version:

```
foreach v in V
    foreach u,w in Adjacency(v)
        if (u,w) in E
            Triangles[v]++
```

Running time: $\sum_{v \in V} d_v^2$

Big Data and Long Tails

What is the degree distribution ?

Big Data and Long Tails

What is the degree distribution ?

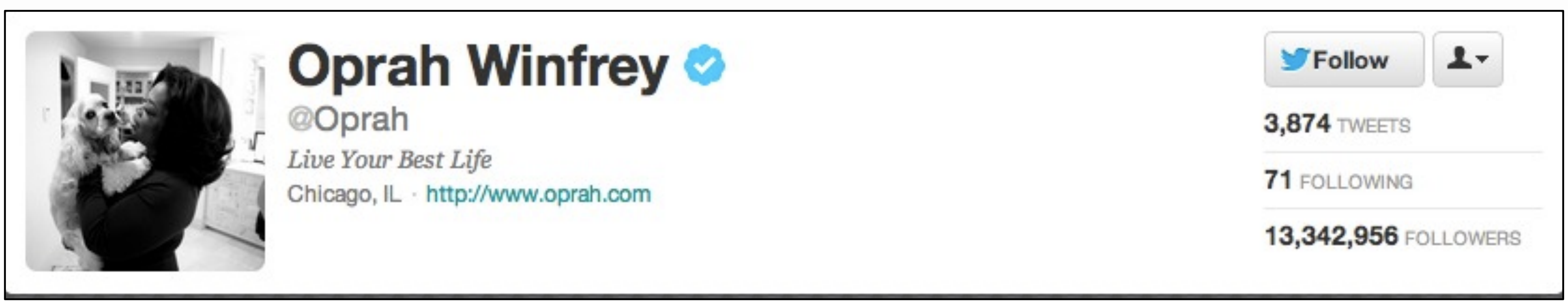
Many natural graphs have a very skewed degree distribution:

Big Data and Long Tails

What is the degree distribution ?

Many natural graphs have a very skewed degree distribution:

- Few nodes with extremely high degree

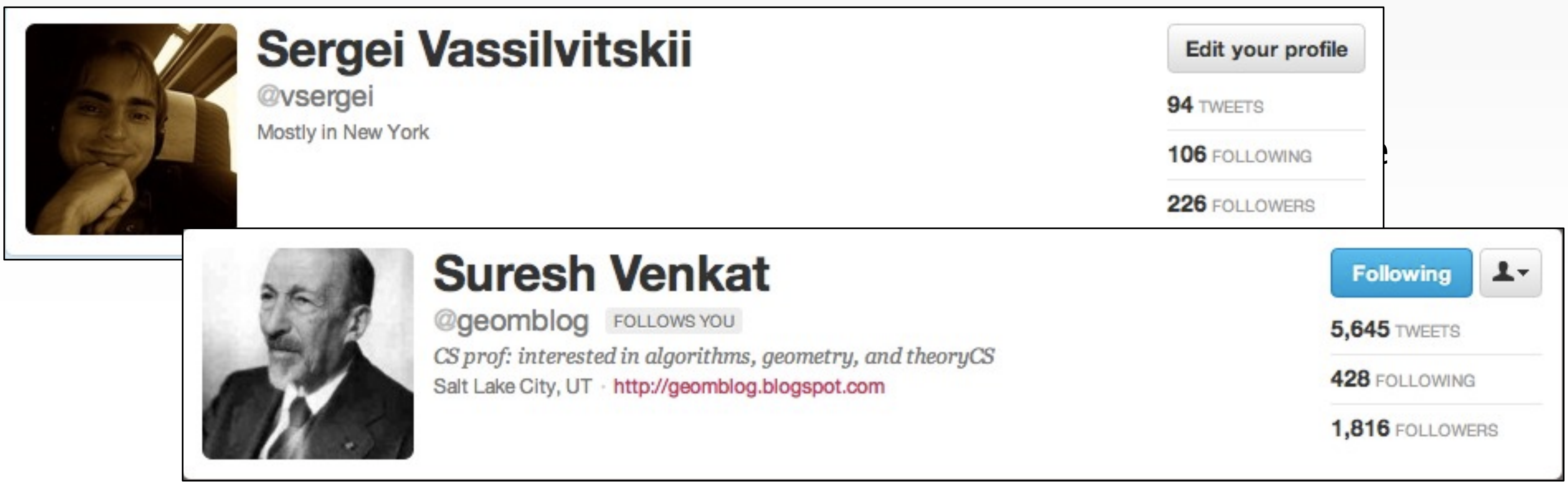


Big Data and Long Tails

What is the degree distribution ?

Many natural graphs have a very skewed degree distribution:

- Few nodes with extremely high degree
- Many nodes with low degree



Big Data and Long Tails

What is the degree distribution ?

Many natural graphs have a very skewed degree distribution:

- Few nodes with extremely high degree
- Many nodes with low degree

- Fat tails: the low degree nodes (tails of the distribution) form the majority of the nodes.
- The graph has a low average degree, but that is a misleading statistic

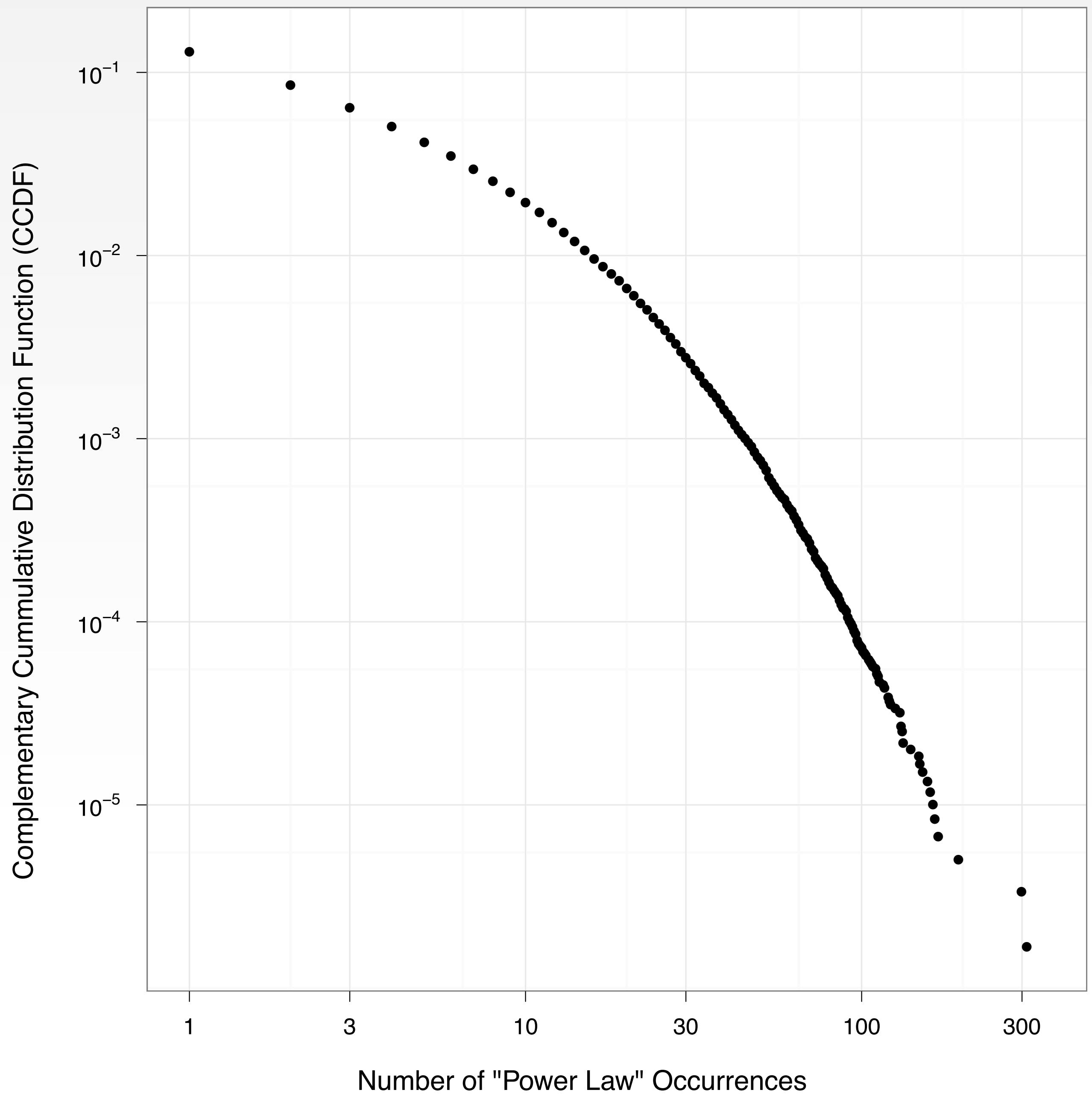
Power Law Hype

Is everything a power-law?

Power Law Hype

Is everything a power-law?

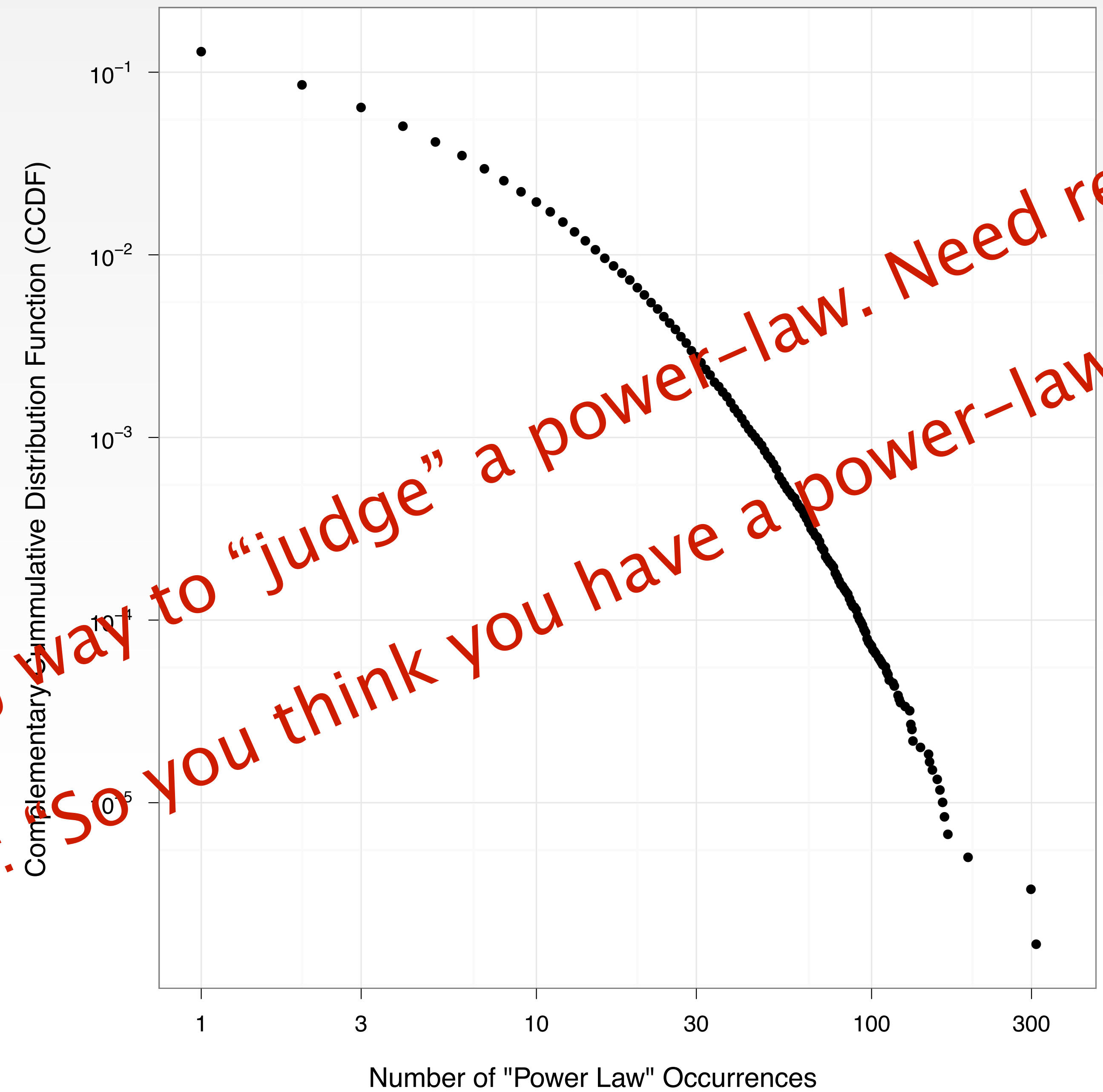
- Mentions of "power law" on ArXiV (circa 2011)



Power Law Hype

Is everything a power-law?

- Mentions of "power law" on ArXiV (circa 2011)



Wrong way to "judge" a power-law. Need real statistics

See : "So you think you have a power-law..isn't that special?"

How to Count Triangles

Sequential Version:

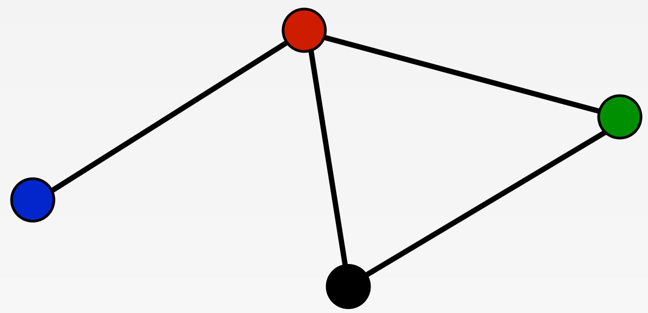
```
foreach v in V
    foreach u,w in Adjacency(v)
        if (u,w) in E
            Triangles[v]++
```

Running time: $\sum_{v \in V} d_v^2$

In practice this is quadratic, as some vertex will have very high degree

Parallel Version

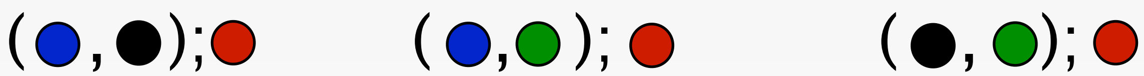
Parallelize the edge checking phase

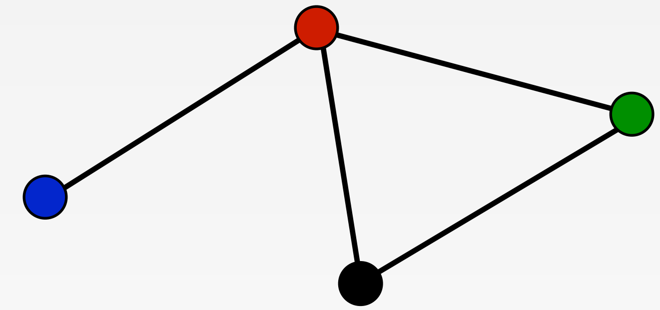


Parallel Version

Round 1: Generate all possible length 2 paths

- Map 1: For each v send $(v, \Gamma(v))$ to same reducer.
- Reduce 1: Input: $\langle v; \Gamma(v) \rangle$
  Output: all 2 paths $\langle (v_1, v_2); u \rangle$ where $v_1, v_2 \in \Gamma(u)$





Meaning:

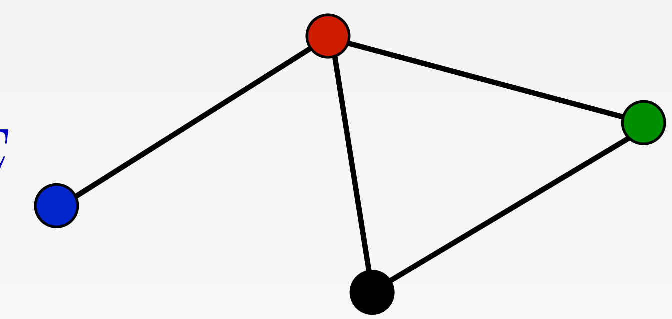
A path from ● to ● through ●

Saturday, August 25, 12

Parallel Version

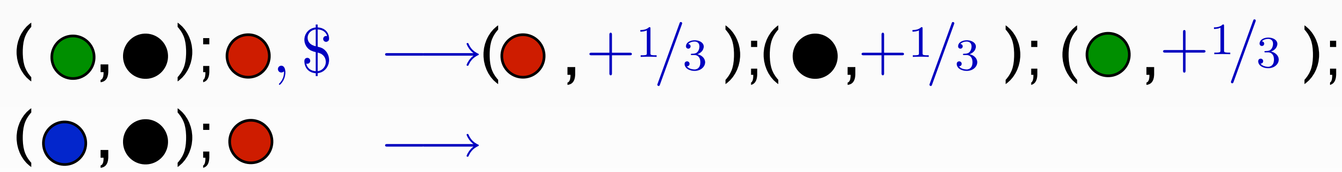
Round 1: Generate all possible length 2 paths

Round 2: Check if the triangle is complete



- Map 2: Send $\langle (v_1, v_2); u \rangle$ and $\langle (v_1, v_2); \$ \rangle$ for $(v_1, v_2) \in E$ to same machine.
- Reduce 2: input: $\langle (v, w); u_1, u_2, \ldots, u_k, \$? \rangle$

  Output: if $\$$ part of the input, then: $\langle v, 1/3 \rangle, \langle w, 1/3 \rangle, \langle u_1, 1/3 \rangle, \ldots, \langle u_k, 1/3 \rangle$



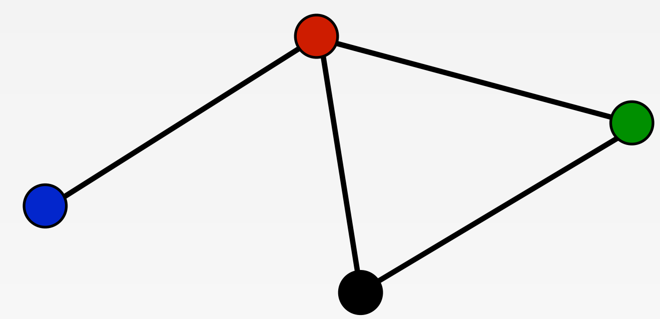
Saturday, August 25, 12

Parallel Version

Round 1: Generate all possible length 2 paths

Round 2: Check if the triangle is complete

Round 3: Sum all the counts



Data skew

How much parallelization can we achieve?

- Generate all the paths to check in parallel
- The running time becomes $\max_{v \in V} d_v^2$

Data skew

How much parallelization can we achieve?

- Generate all the paths to check in parallel
- The running time becomes $\max_{v \in V} d_v^2$

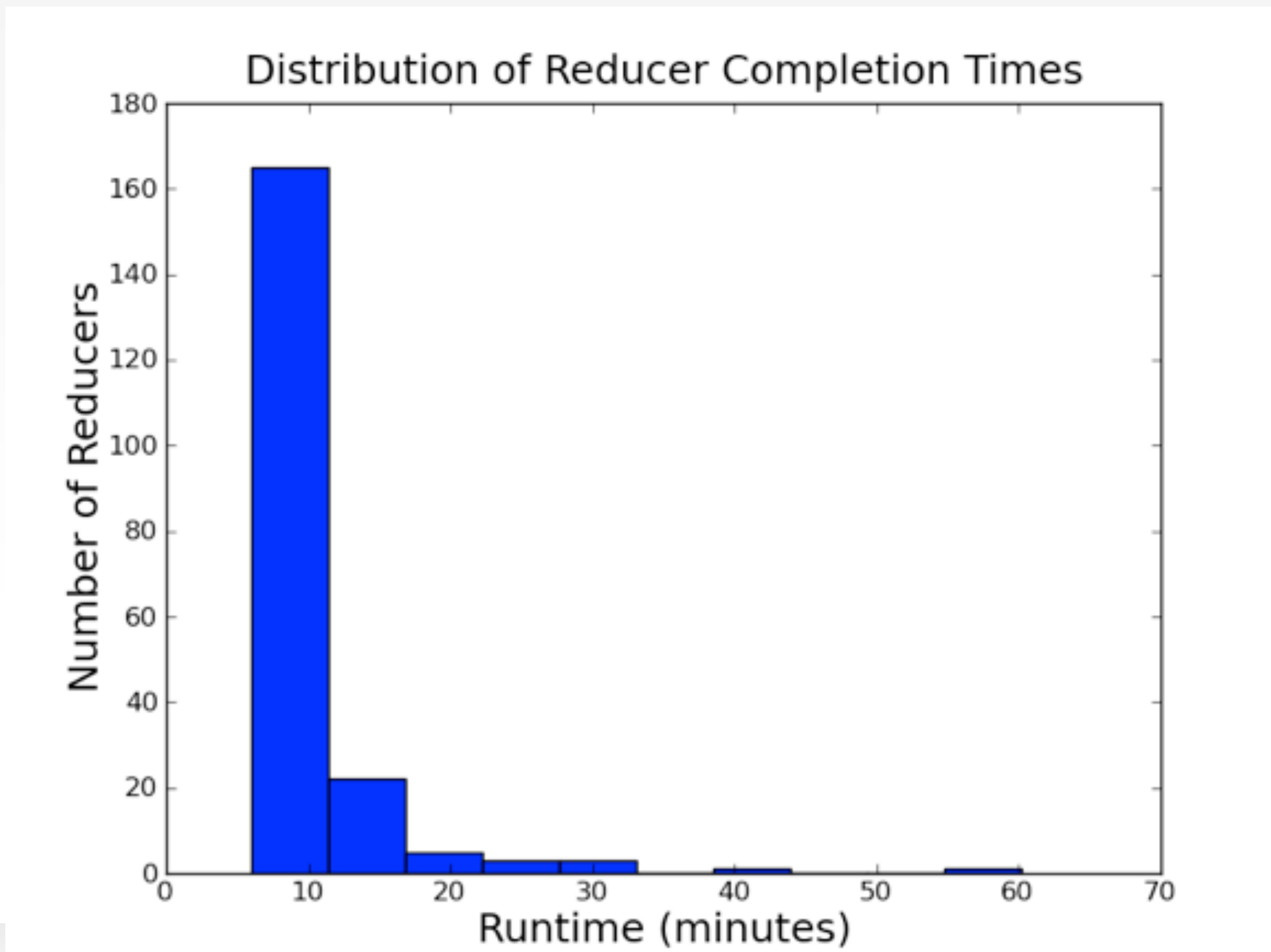
Naive parallelization does not help with data skew

- It was the few high degree nodes that accounted for the running time
- Example. 3.2 Million followers, must generate 10 Trillion (10^{13}) potential edges to check.
- Even if generating 100M edges to check per second, 100K seconds ~ 27 hours.

“Just 5 more minutes”

Running the naive algorithm on LiveJournal Graph

- 80% of reducers done after 5 min
- 99% done after 35 min



Adapting the Algorithm

Approach 1: Dealing with skew directly

- currently every triangle counted 3 times (once per vertex)
- Running time quadratic in the degree of the vertex
- Idea: Count each once, from the perspective of lowest degree vertex
- Does this heuristic work?

Adapting the Algorithm

Approach 1: Dealing with skew directly

- currently every triangle counted 3 times (once per vertex)
- Running time quadratic in the degree of the vertex
- Idea: Count each once, from the perspective of lowest degree vertex
- Does this heuristic work?

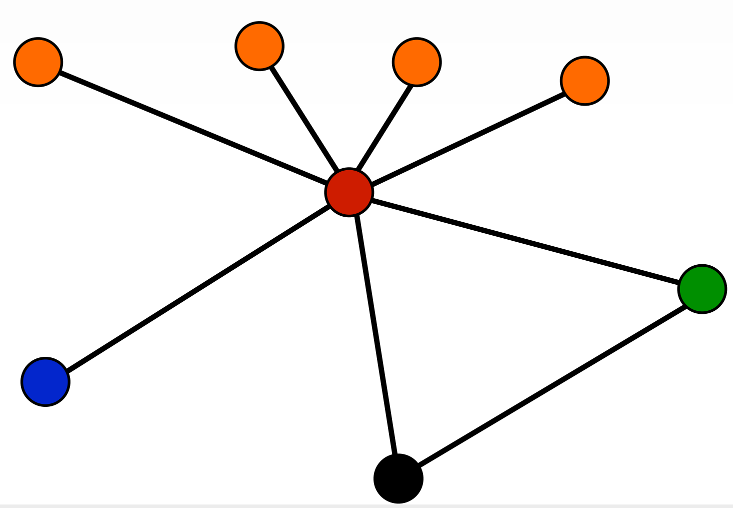
Approach 2: Divide & Conquer

- Equally divide the graph between machines
- But any edge partition will be bound to miss triangles
- Divide into overlapping subgraphs, account for the overlap

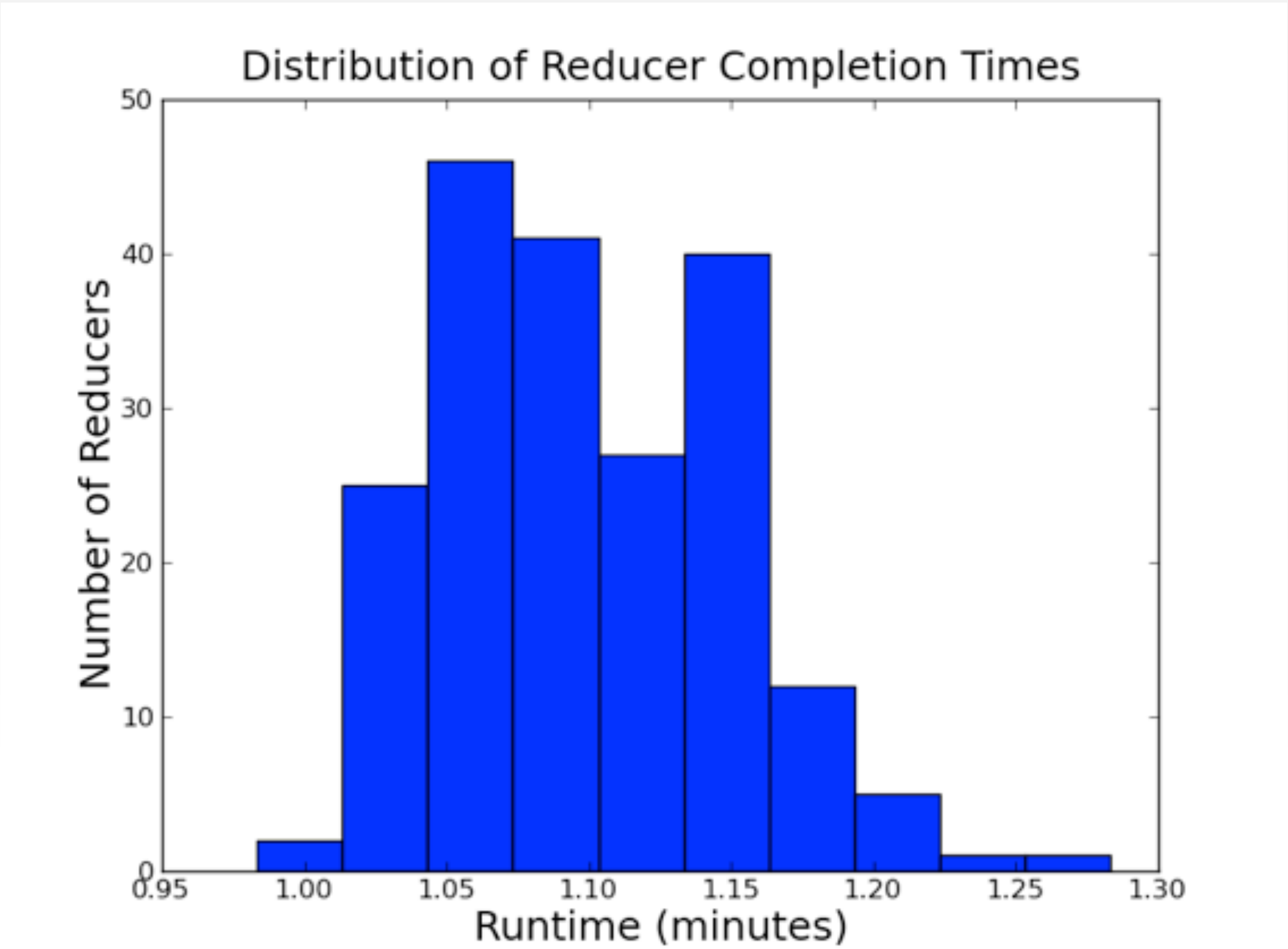
How to Count Triangles Better

Sequential Version [Schank '07]:

```
foreach v in V
   foreach u,w in Adjacency(v)
     if deg(u) > deg(v) && deg(w) > deg(v)
         if (u,w) in E
            Triangles[v]++
```



Does it make a difference?



Dealing with Skew

Why does it help?

- Partition nodes into two groups:
  - Low: $\mathcal{L} = \{v : d_v \leq \sqrt{m}\}$
  - High: $\mathcal{H} = \{v : d_v > \sqrt{m}\}$
- There are at most $2\sqrt{m}$ high nodes
  - Each produces paths to other high nodes: $O(m)$ paths per node
  - Therefore they generate: $O(m^{3/2})$ paths in total

Proof (cont.)

- Let n_i be the number of nodes of degree i .
- Then the total number of two paths is:

$$\sum_{i=1}^{\sqrt{m}} n_i \cdot i^2$$

Proof (cont.)

- Let n_i be the number of nodes of degree i .
- Then the total number of two paths generated by Low nodes is:

$$\sum_{i=1}^{\sqrt{m}} n_i \cdot i^2 \leq \sum_{i=1}^{\sqrt{m}} (n_i \cdot i) \cdot i$$

Proof (cont.)

- Let n_i be the number of nodes of degree i .
- Then the total number of two paths generated by Low nodes is:

$$\sum_{i=1}^{\sqrt{m}} n_i \cdot i^2 \leq \sum_{i=1}^{\sqrt{m}} (n_i \cdot i) \cdot i$$

$$\leq \sqrt{\left(\sum_{i=1}^{\sqrt{m}} (n_i \cdot i)^2\right) \left(\sum_{i=1}^{\sqrt{m}} i^2\right)}$$

By Cauchy-Schwarz

Proof (cont.)

- Let n_i be the number of nodes of degree i .
- Then the total number of two paths generated by Low nodes is:

$$\sum_{i=1}^{\sqrt{m}} n_i \cdot i^2 \leq \sum_{i=1}^{\sqrt{m}} (n_i \cdot i) \cdot i$$

$$\leq \sqrt{\left(\sum_{i=1}^{\sqrt{m}} (n_i \cdot i)^2\right)\left(\sum_{i=1}^{\sqrt{m}} i^2\right)}$$

By Cauchy-Schwarz

$$\leq \sqrt{4m^{3/2} \cdot m^{3/2}}$$

Since: $\sum_{i}^{\sqrt{m}} (n_i \cdot i) \leq 2m$

$$= O(m^{3/2})$$

Saturday, August 25, 12

Discussion

Why does it help?

- The algorithm automatically load balances
- Every node generates at most $O(m)$ paths to check
- Hence the mappers take about the same time to finish
- Total work is $O(m^{3/2})$, which is optimal

Improvement Factor:

- Live Journal:
  - 5M nodes, 86M edges
  - Number of 2 paths: 15B to 1.3B, ~12
- Twitter snapshot:
  - 42M nodes, 2.4B edges
  - Number of 2 paths: 250T to 300B

Approach 2: Graph Split

Partitioning the nodes:

- Previous algorithm shows one way to achieve better parallelization
- But what if even $O(m)$ is too much. Is it possible to divide input into smaller chunks?

Approach 2: Graph Split

Partitioning the nodes:

- Previous algorithm shows one way to achieve better parallelization
- But what if even $O(m)$ is too much. Is it possible to divide input into smaller chunks?

Graph Split Algorithm:

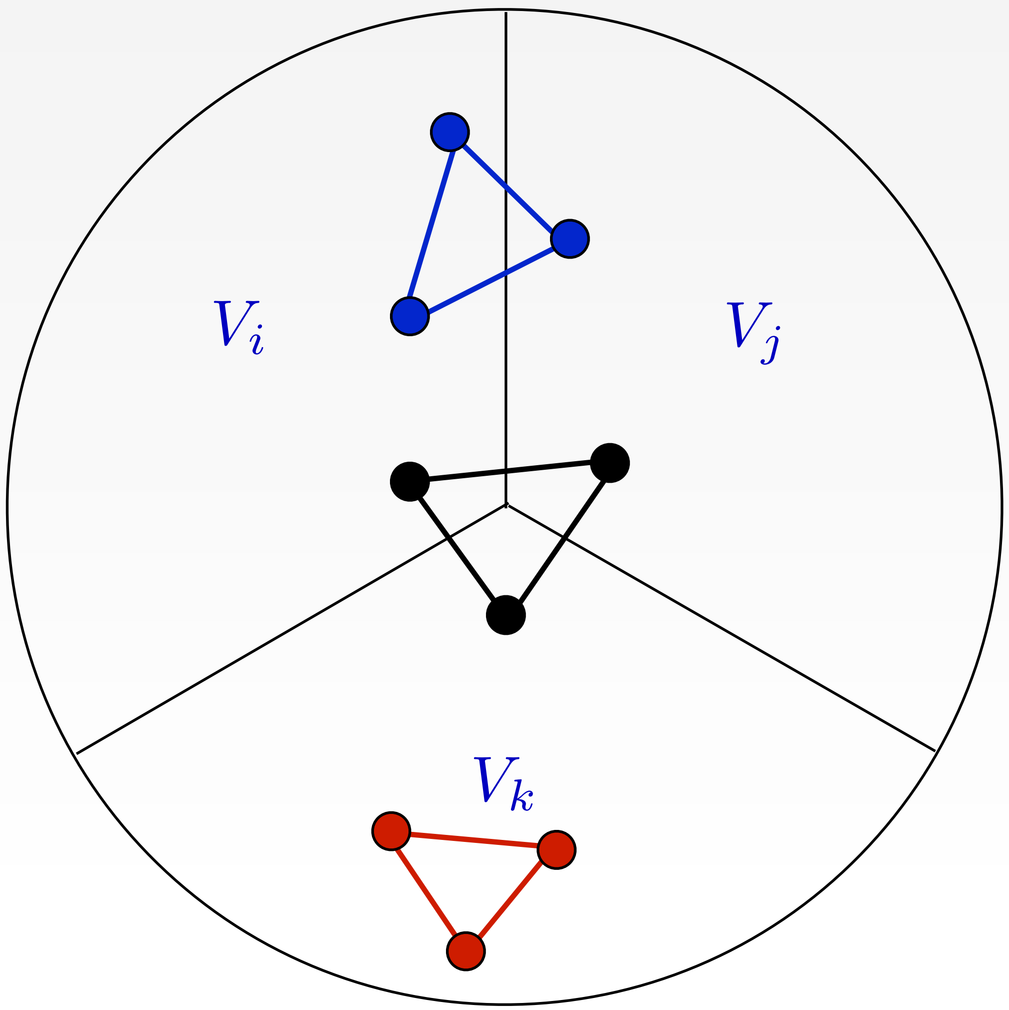
- Partition vertices into p equal sized groups $V_1, V_2, \ldots, V_p$.
- Consider all possible triples (V_i, V_j, V_k) and the induced subgraph:

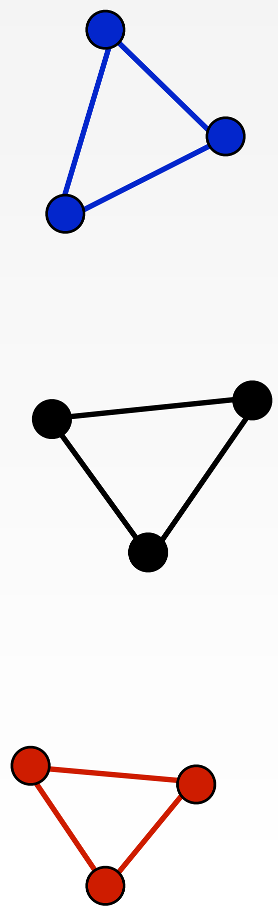
$$G_{ijk} = G\left[V_i \cup V_j \cup V_k\right]$$

- Compute the triangles on each G_{ijk} separately.

Approach 2: Graph Split

Some Triangles present in multiple subgraphs:





in p–2 subgraphs

in 1 subgraph

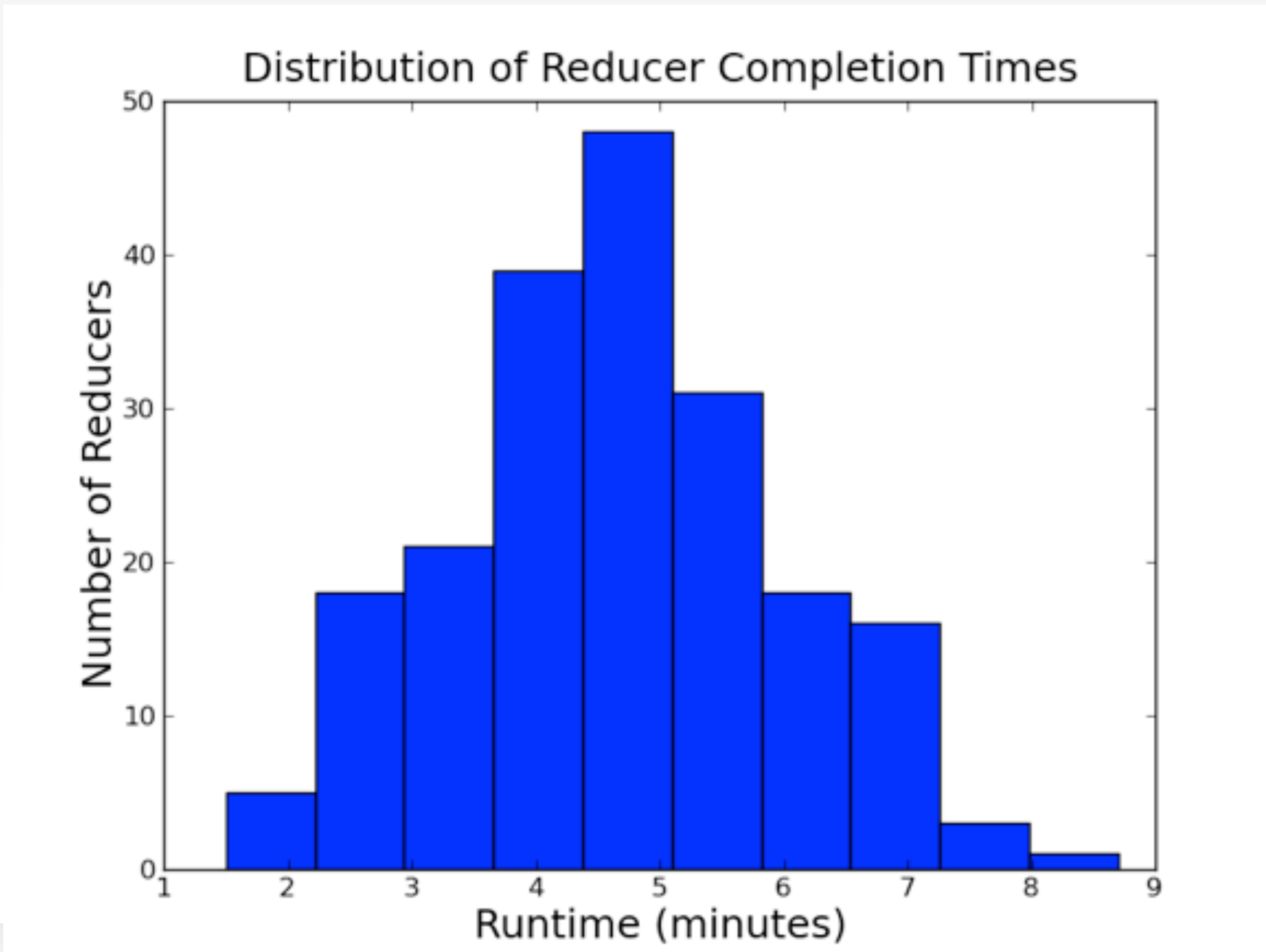
in ~p^2 subgraphs

Can count exactly how many subgraphs each triangle will be in

Approach 2: Graph Split

Analysis:

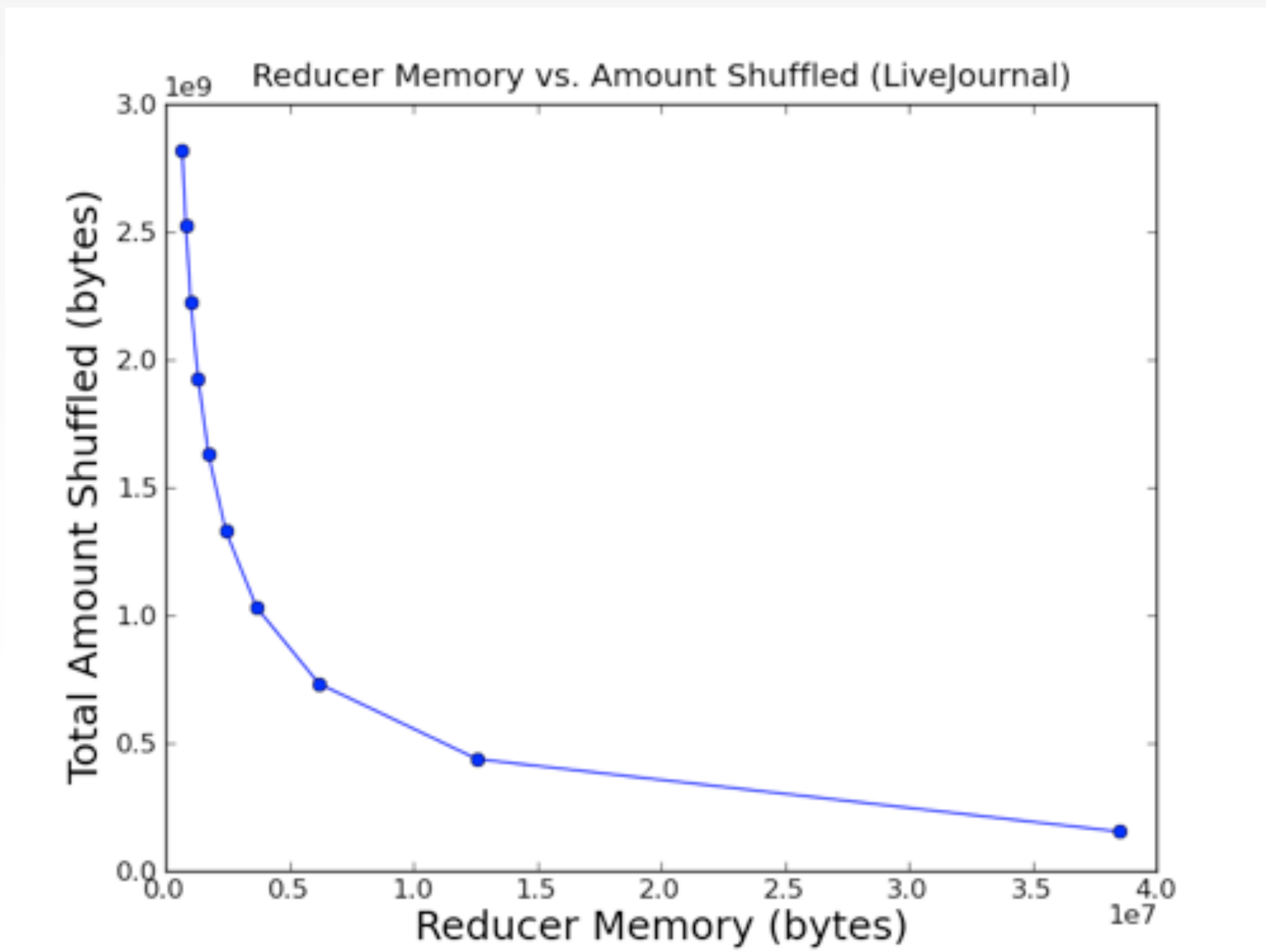
- Each subgraph has $O(m/p^2)$ edges in expectation.
- Very balanced running times



Approach 2: Graph Split

Analysis:

- Very balanced running times
- p controls memory needed per machine



Approach 2: Graph Split

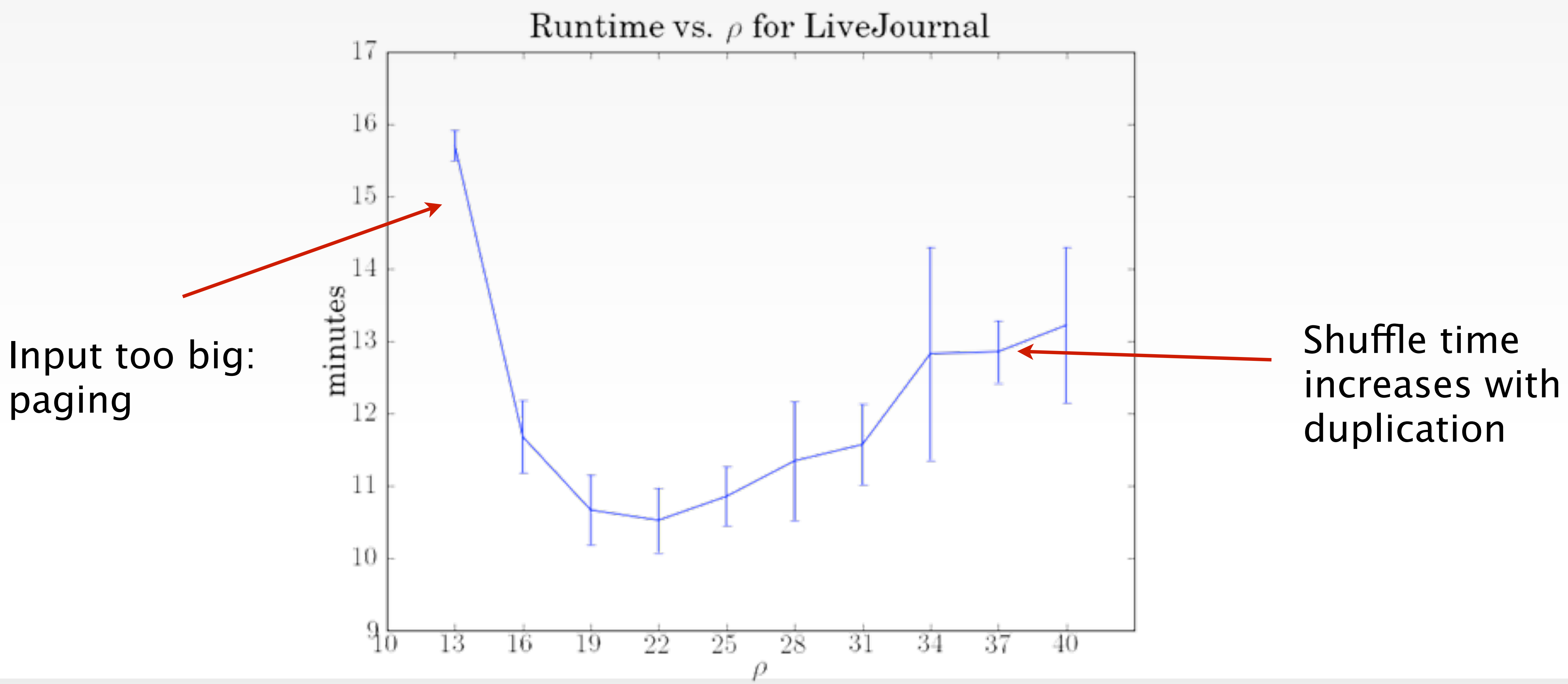
Analysis:

- Very balanced running times
- p controls memory needed per machine
- Total work: $p^3 \cdot O((m/p^2)^{3/2}) = O(m^{3/2})$, independent of p

Approach 2: Graph Split

Analysis:

- Very balanced running times
- p controls memory needed per machine
- Total work: $p^3 \cdot O((m/p^2)^{3/2}) = O(m^{3/2})$, independent of p



Saturday, August 25, 12

Beyond Triangles

Counting other subgraphs?

- Count number of subgraphs $H = (W, F)$
- Partition vertices into p equal sized groups. $V_1, V_2, \ldots, V_p$
- Consider all possible combinations of $|W|$ groups
- Correct for multiple counting of subgraphs

Data Skew

Naive parallelism does not always work

- Must be aware of skew in the data

Too much parallelism may be detrimental:

- Breaks data locality
- Need to find a sweet spot

Overview:

MapReduce:

- Lots of machines
- Synchronous computation

Data:

- MADly big: must be distributed
- Usually highly skewed

References

- MapReduce: Simplified Data Processing on Large Clusters. Jeffrey Dean, Sanjay Ghemawat, OSDI 2004.
- A Model of Computation for MapReduce. Howard Karloff, Siddharth Suri, S.V., SODA 2010.
- Counting Triangles and the Curse of the Last Reducer. Siddharth Suri, S.V., WWW 2011.
- Ode to Power Laws: http://messymatters.com/powerlaws
- So you think you have a power law -- Well Isn't that special: http://cscs.umich.edu/~crshalizi/weblog/491.html
- Optimizing Multiway Joins in a MapReduce Environment: Foto Afrati, Jeffrey Ullman, EDBT 2010.