Large-scale Graph Mining @ Google NY

Vahab Mirrokni

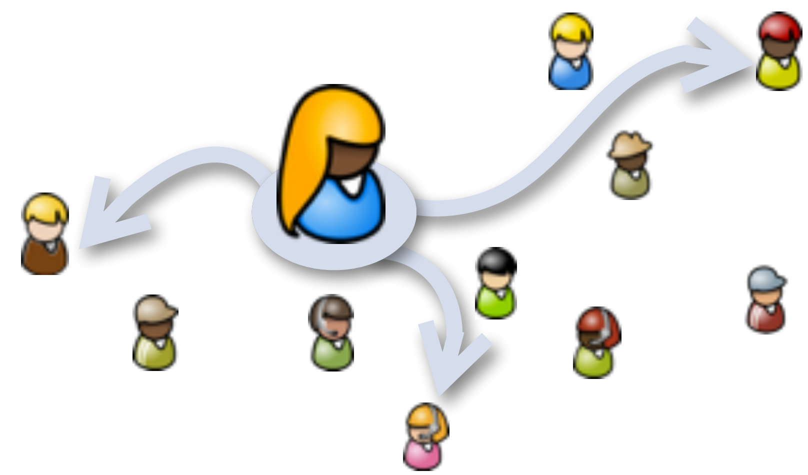
Google Research
New York, NY

DIMACS Workshop

Large-scale graph mining

Many applications

- Friend suggestions
- Recommendation systems
- Security
- Advertising



Benefits

- Big data available
- Rich structured information

New challenges

- Process data efficiently
- Privacy limitations

Google NYC Large-scale graph mining

Develop a *general-purpose library* of graph mining tools for XXXB nodes and XT edges

via **MapReduce+DHT(Flume), Pregel, ASYMP**

Goals:

- Develop scalable tools (Ranking, Pairwise Similarity, Clustering, Balanced Partitioning, Embedding, etc)
- Compare different algorithms/frameworks
- Help product groups use these tools across Google in a loaded cluster (clients in Search, Ads, Youtube, Maps, Social)
- Fundamental Research (Algorithmic Foundations and Hybrid Algorithms/System Research)

Outline

Three perspectives:

- Part 1: Application-inspired Problems
  - Algorithms for Public/Private Graphs
- Part 2: Distributed Optimization for NP-Hard Problems
  - Distributed algorithms via composable core-sets
- Part 3: Joint systems/algorithms research
  - MapReduce + Distributed HashTable Service

Problems Inspired by Applications

Part 1: Why do we need scalable *graph mining*?

Stories:

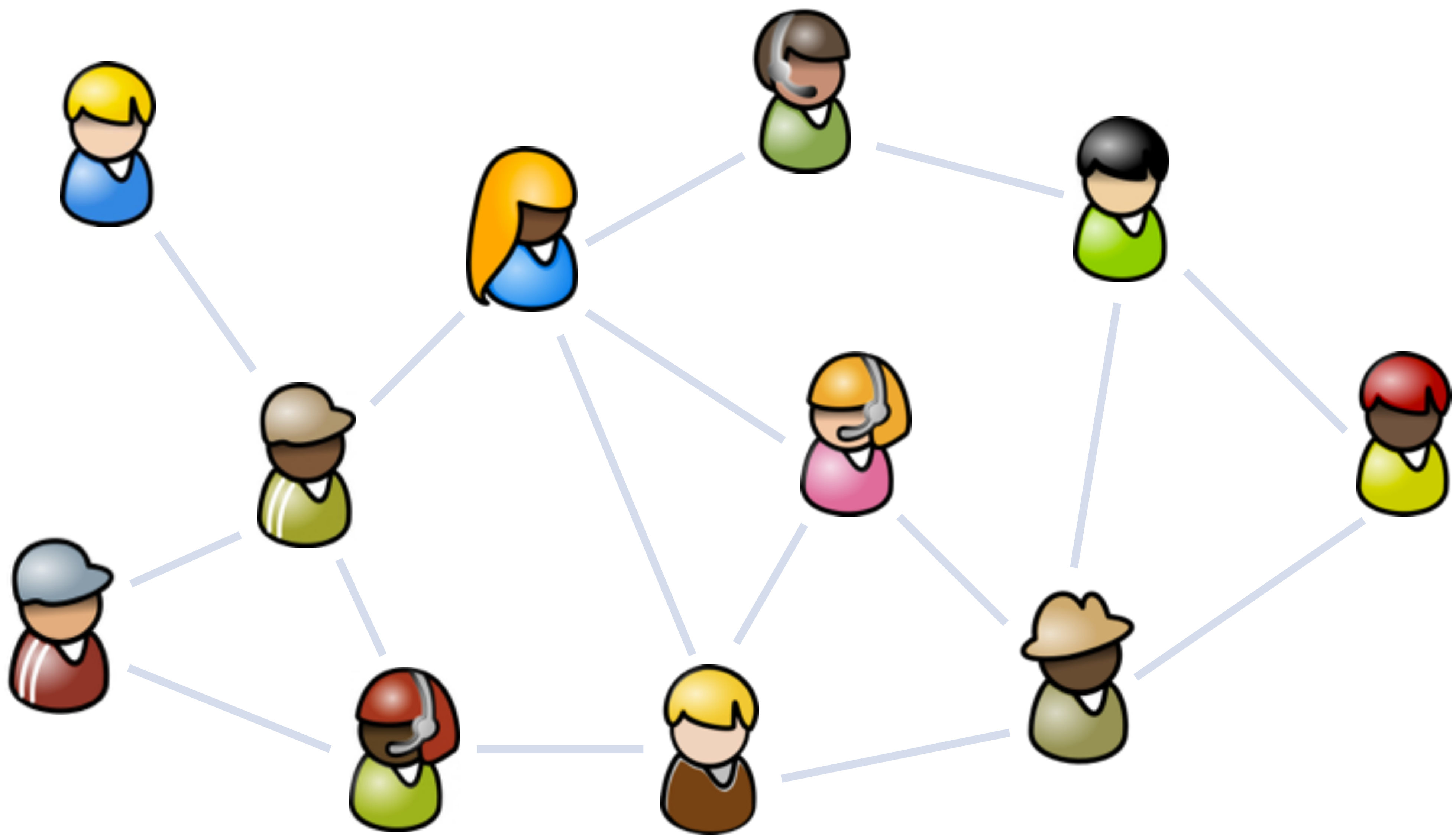
- **Algorithms for Public/Private Graphs,**
  - How to solve a problem for each node on a public graph+its own private network
  - with Chierchetti,Epasto,Kumar,Lattanzi,M: KDD'15

- Ego-net clustering
  - How to use graph structures and improve collaborative filtering
  - with EpastoLattanziSebeTaeiVerma, Ongoing

- Local random walks for conductance optimization,
  - Local algorithms for finding well connected clusters
  - with AllenZu,Lattanzi, ICML'13

Private-Public networks

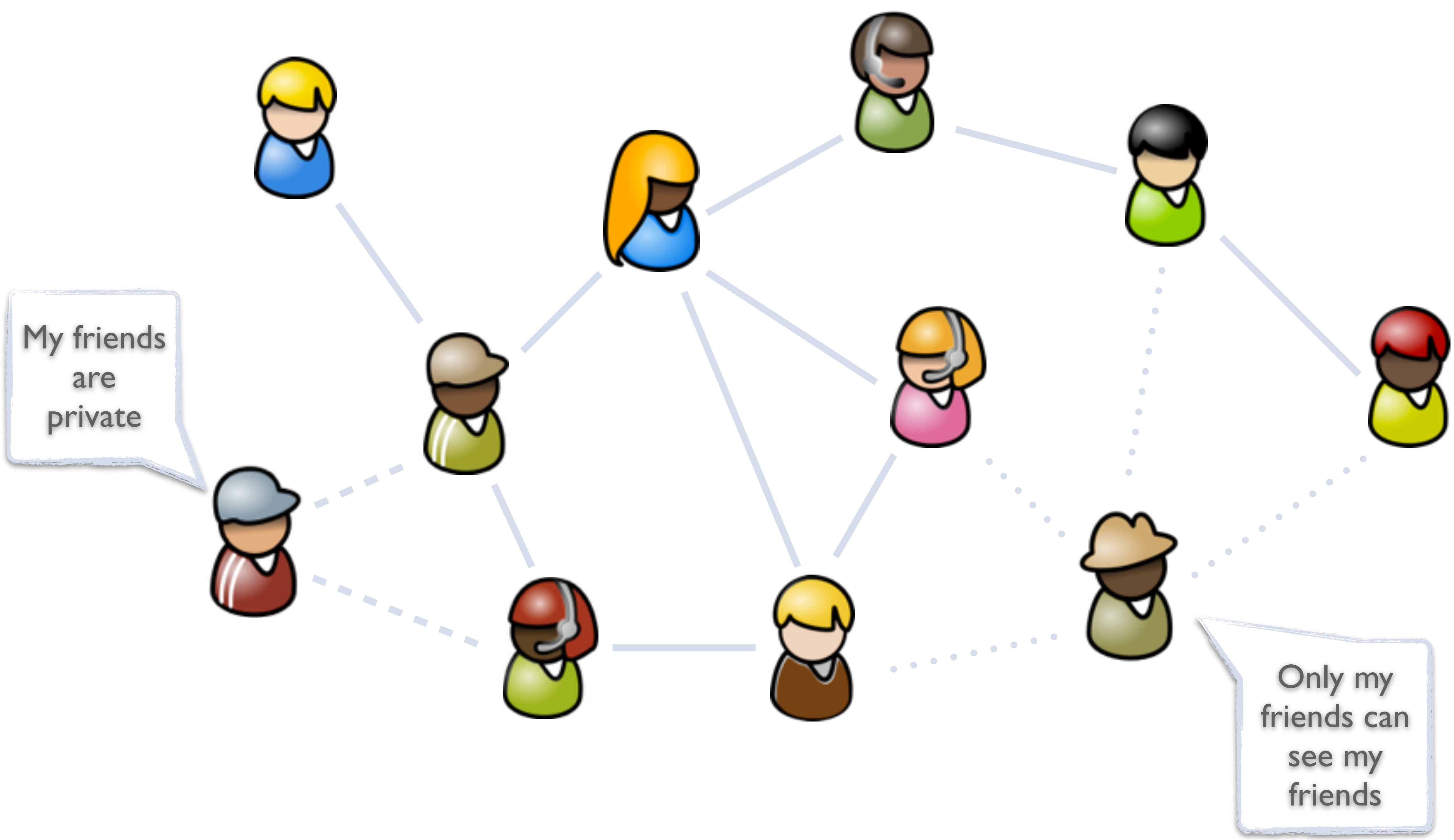
Idealistic vision



Private-Public networks

Reality

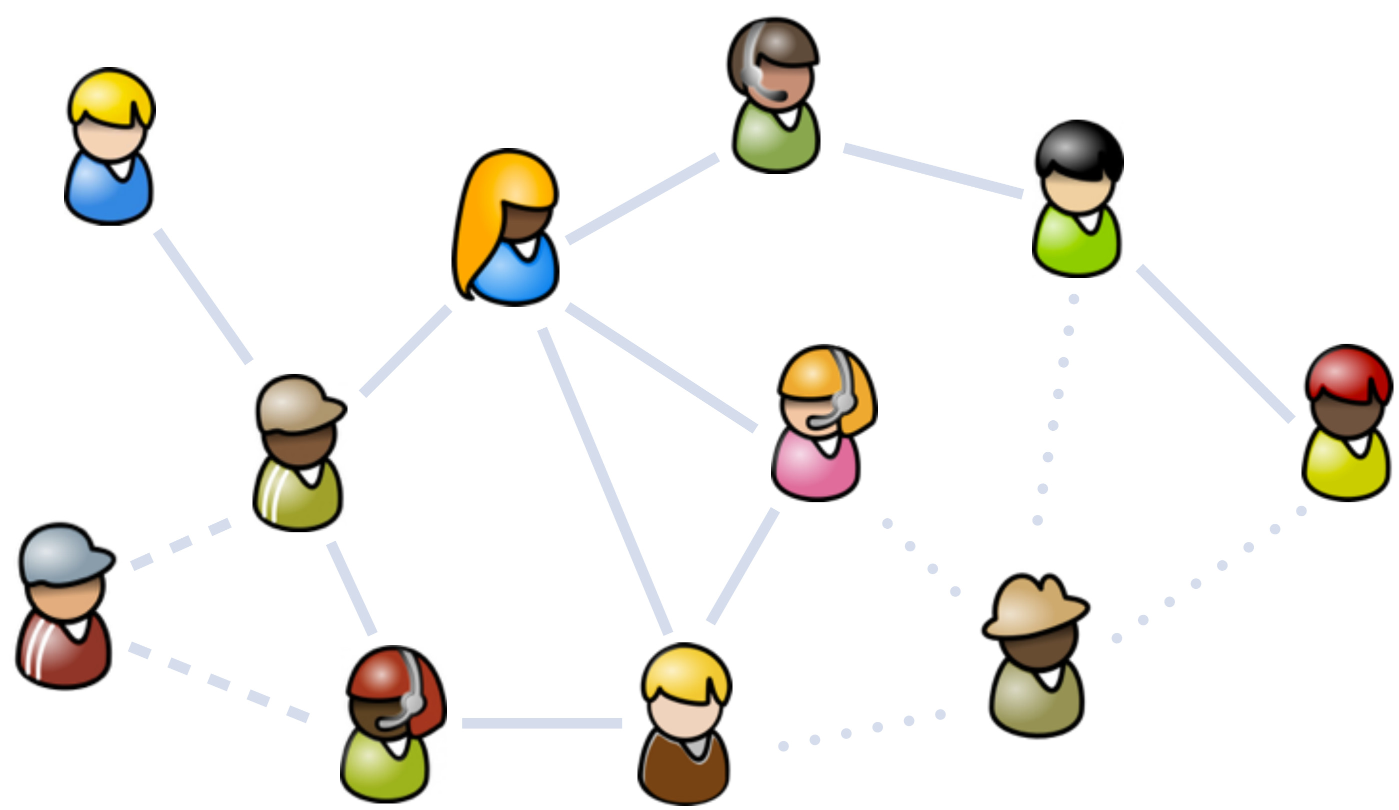
~52% of NYC Facebook users hide their friends



Applications: friend suggestions

Network signals are very useful [CIKM03]

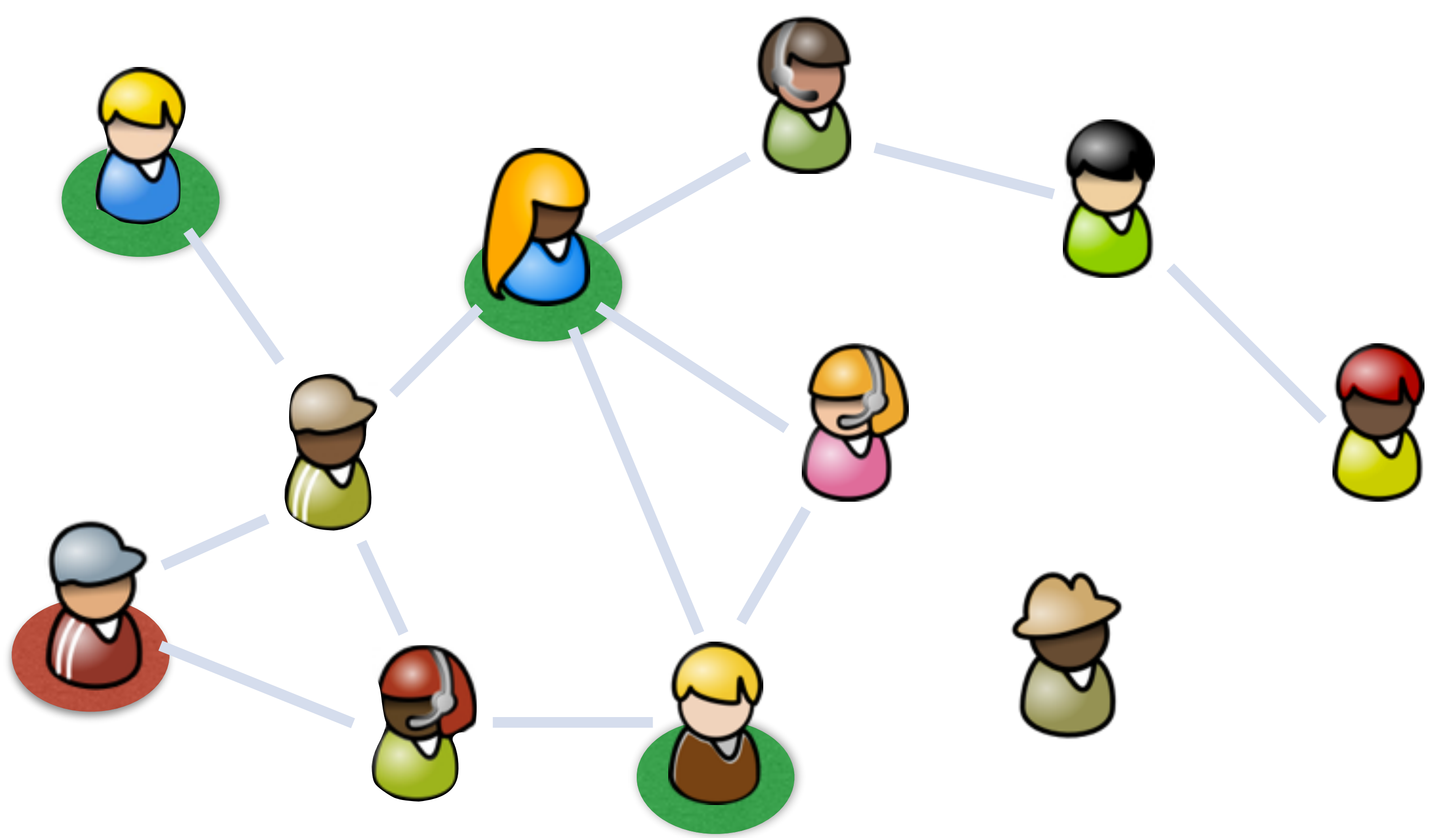
- Number of common neighbors
- Personalized PageRank
- Katz



Applications: friend suggestions

Network signals are very useful [CIKM03]

- Number of common neighbors
- Personalized PageRank
- Katz

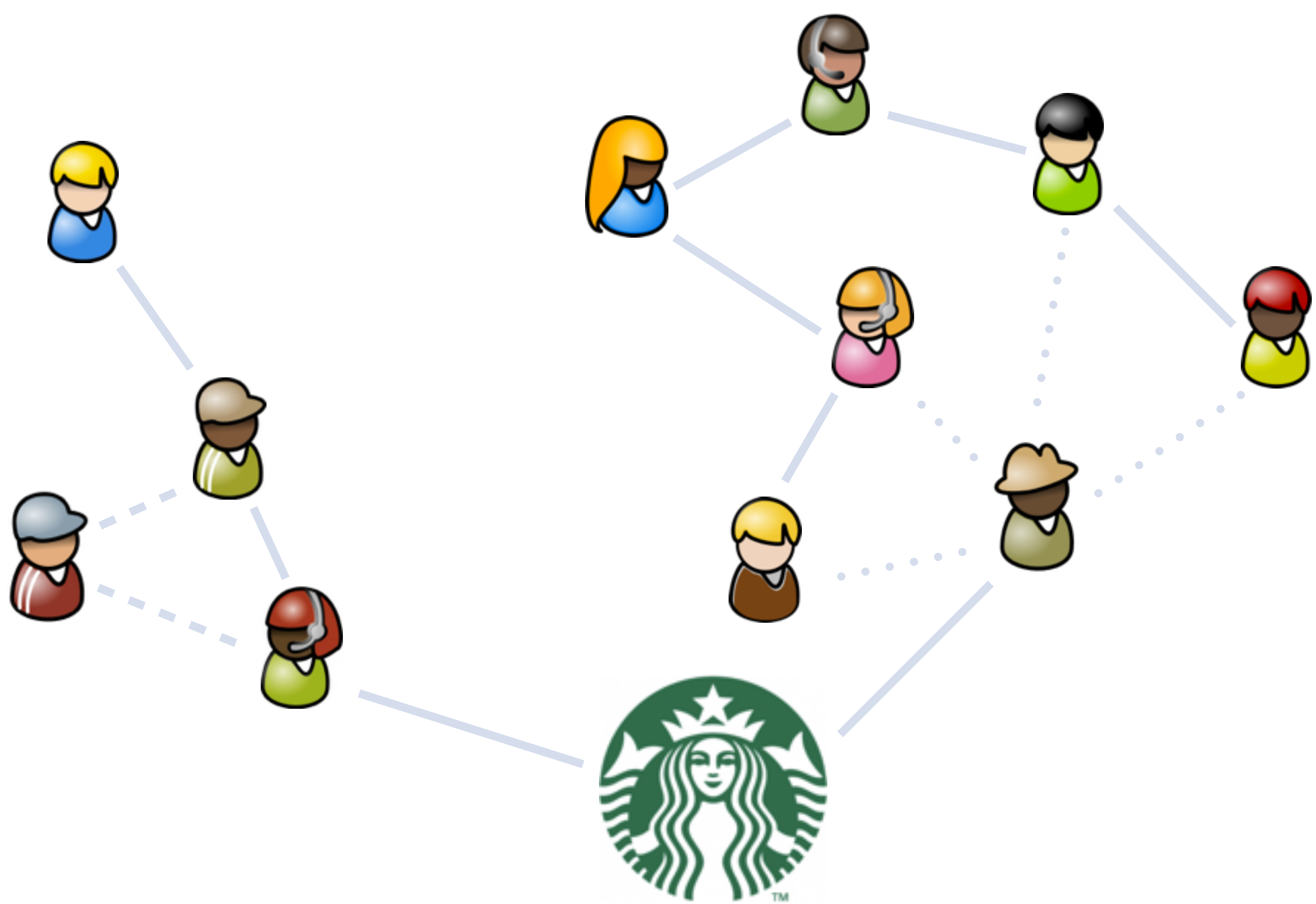


From a user' perspective, there are interesting signals

Applications: advertising

Maximize the reachable sets

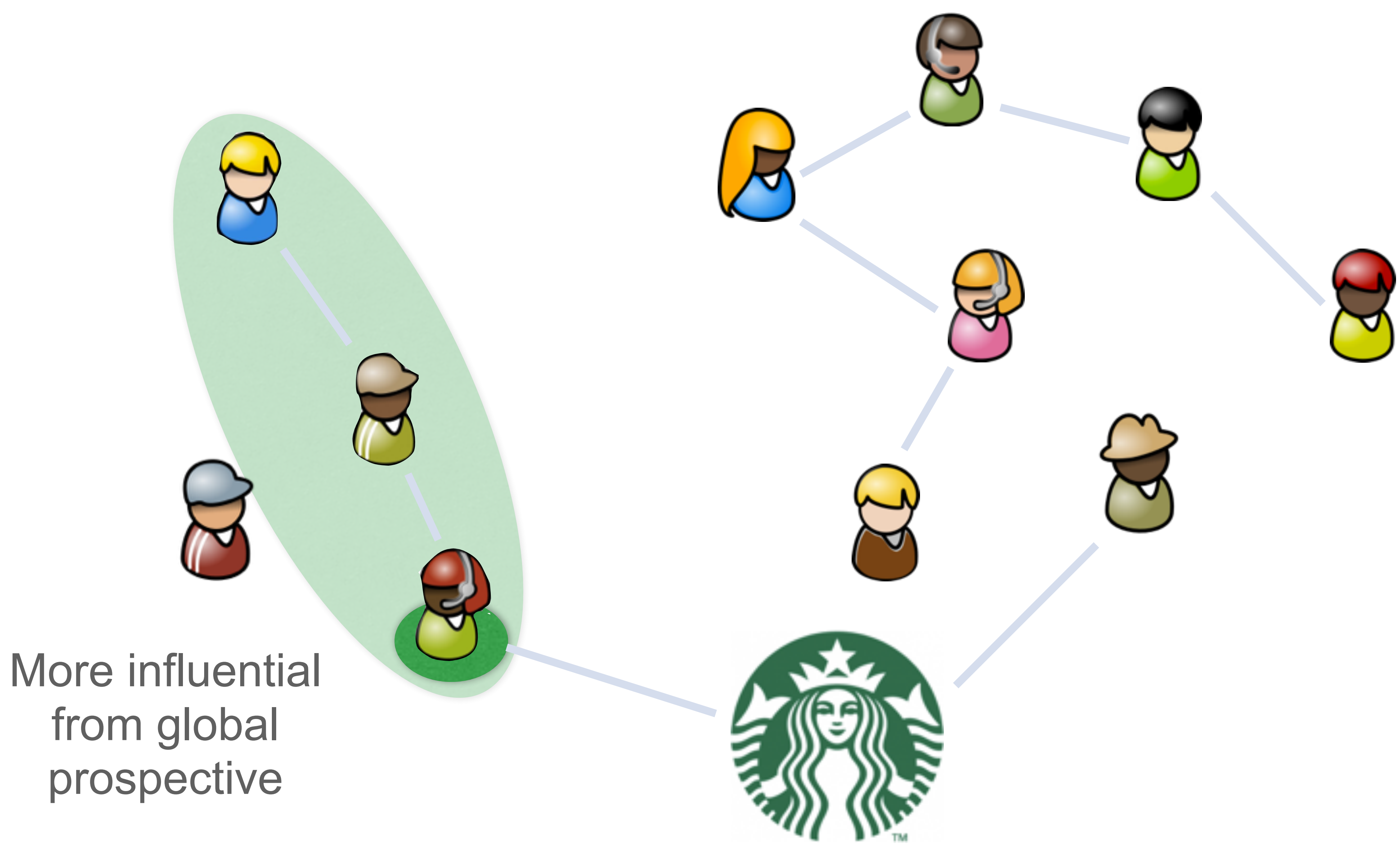
- How many can be reached by re-sharing?



Applications: advertising

Maximize the reachable sets

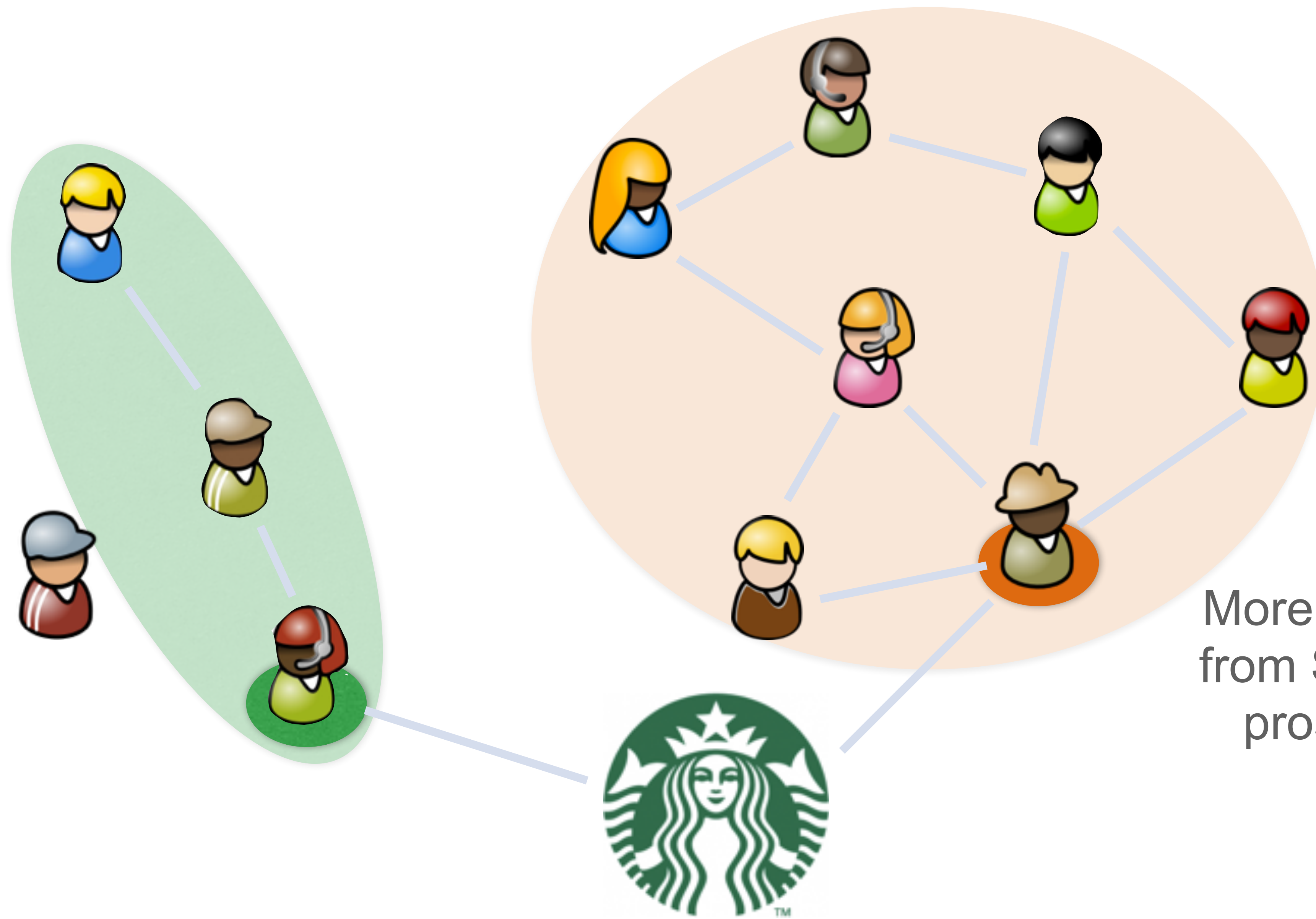
How many can be reached by re-sharing?



Applications: advertising

Maximize the reachable sets

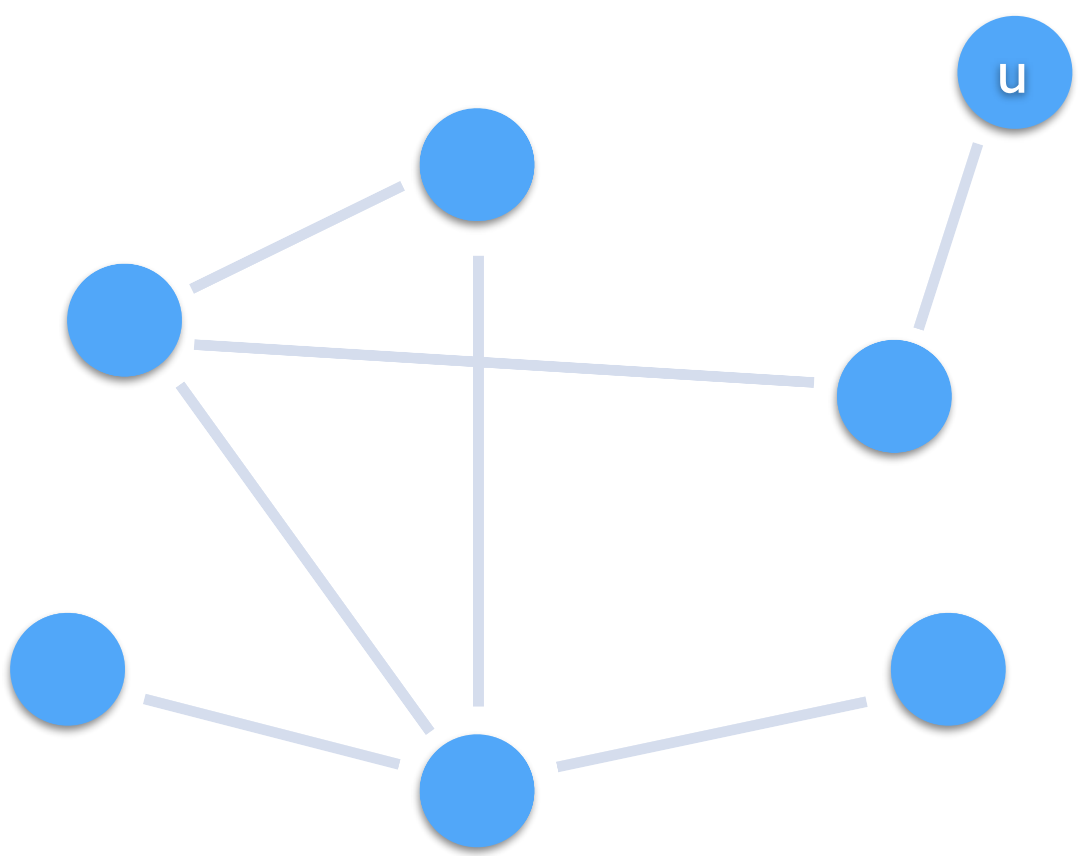
How many can be reached by re-sharing?



More influential from Starbucks' prospective

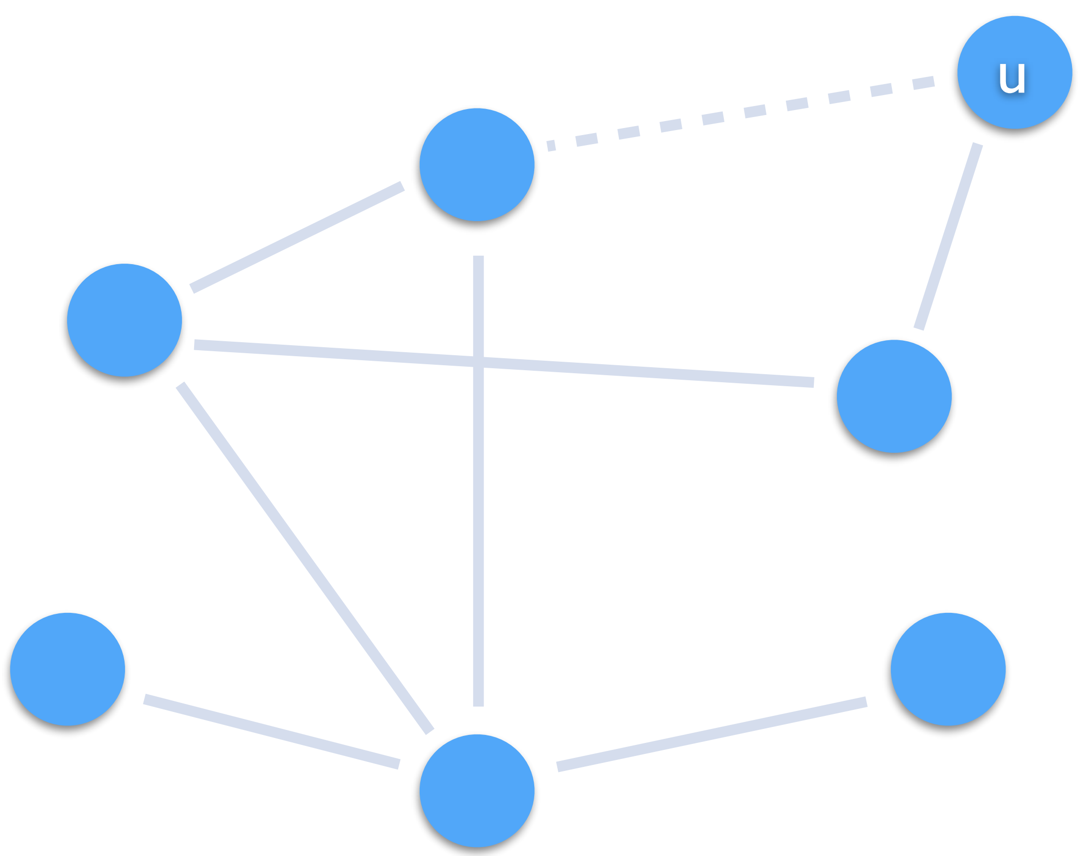
Private-Public problem

There is a public graph G in addition each node u has access to a local graph G_u



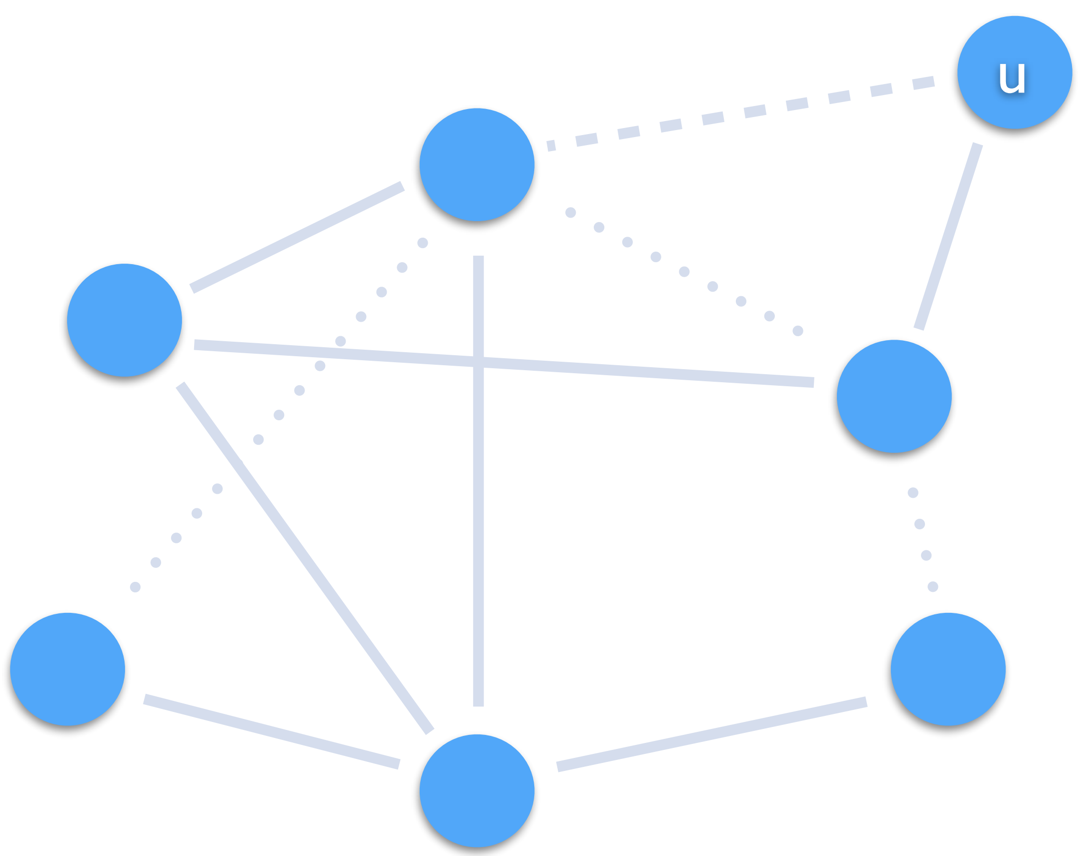
Private-Public problem

There is a public graph G in addition each node u has access to a local graph G_u



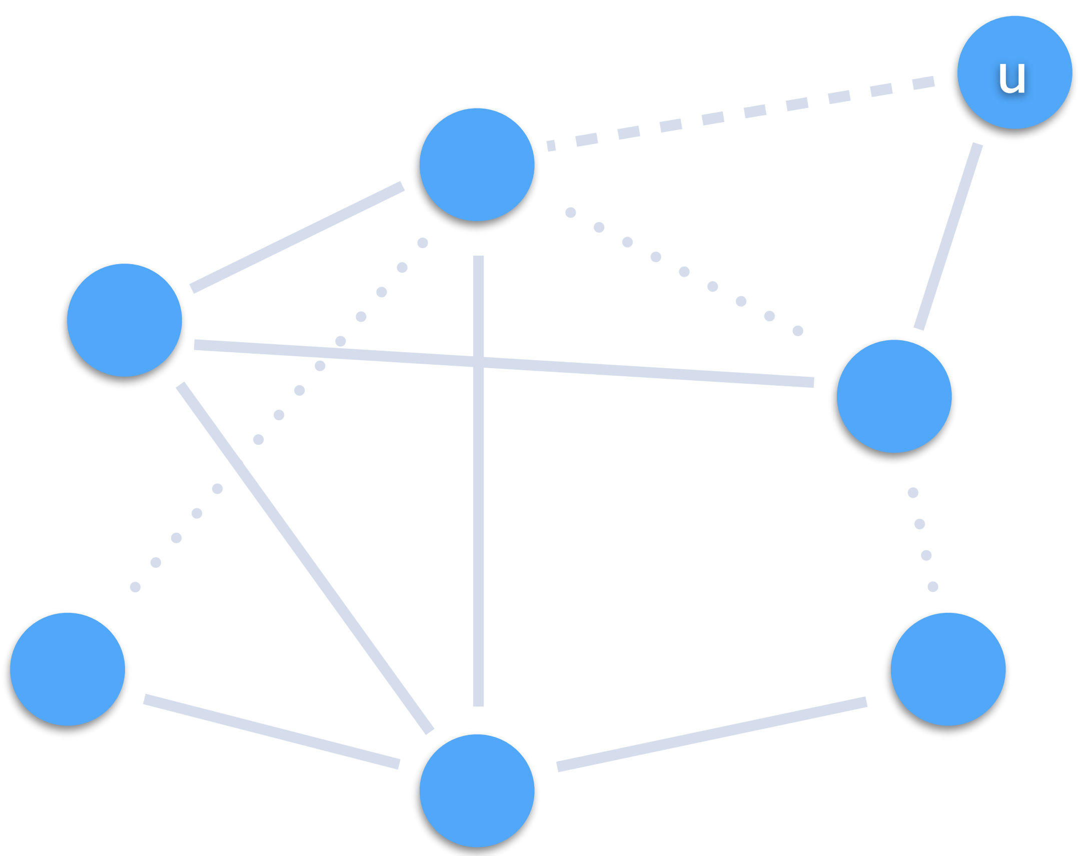
Private-Public problem

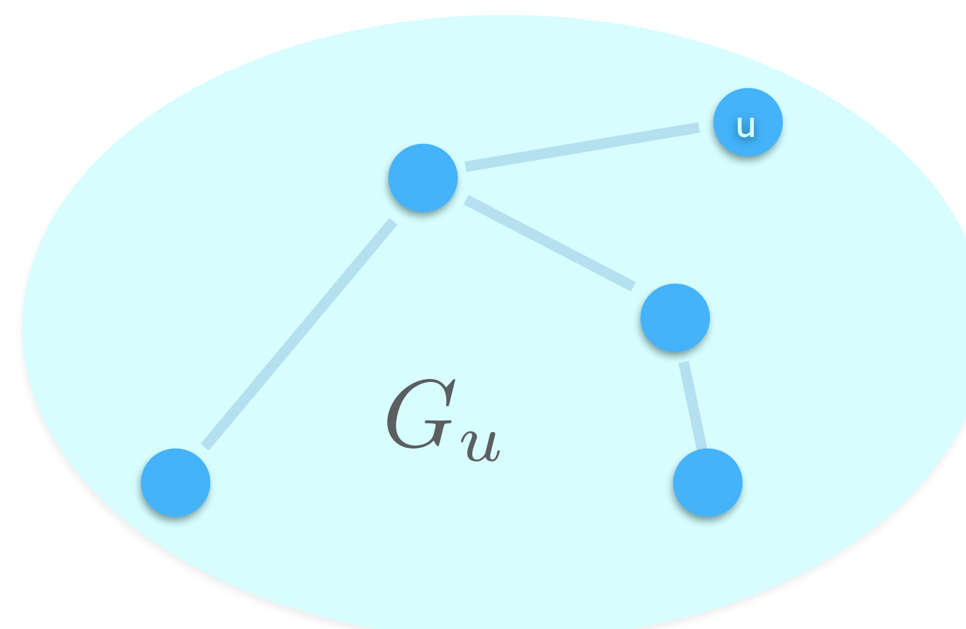
There is a public graph G in addition each node u has access to a local graph G_u



Private-Public problem

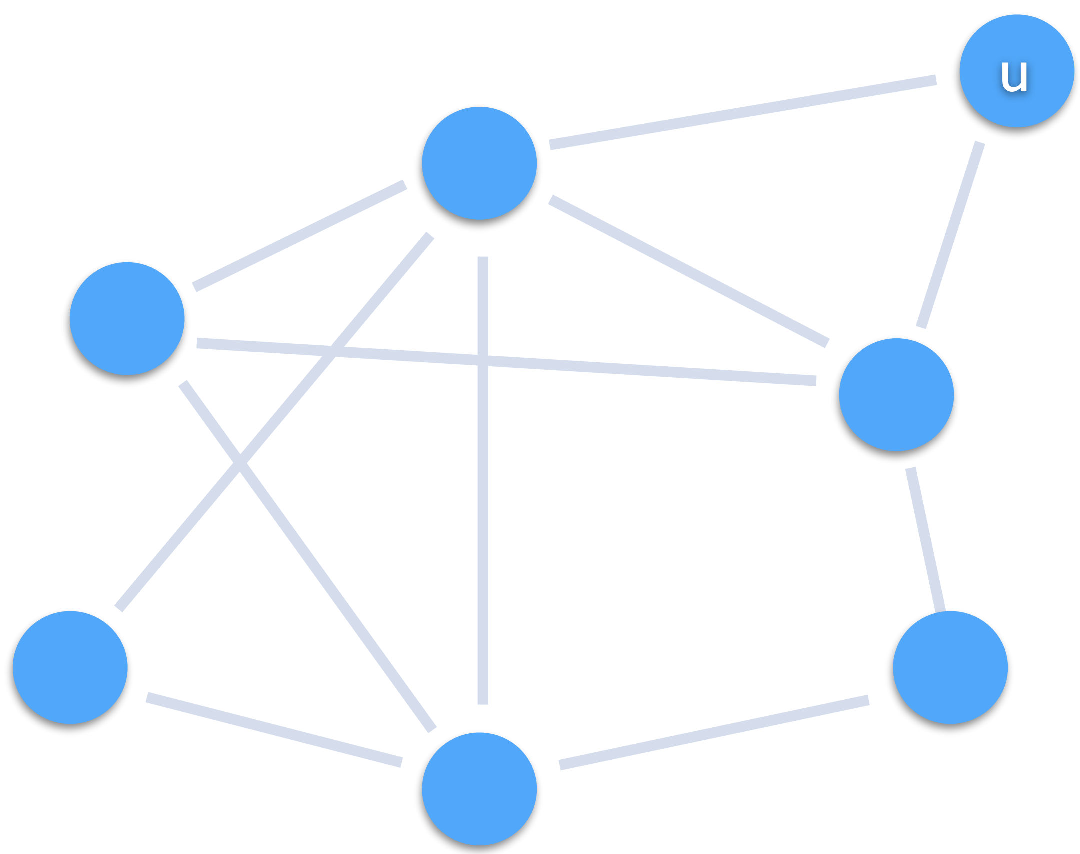
There is a public graph G in addition each node u has access to a local graph G_u





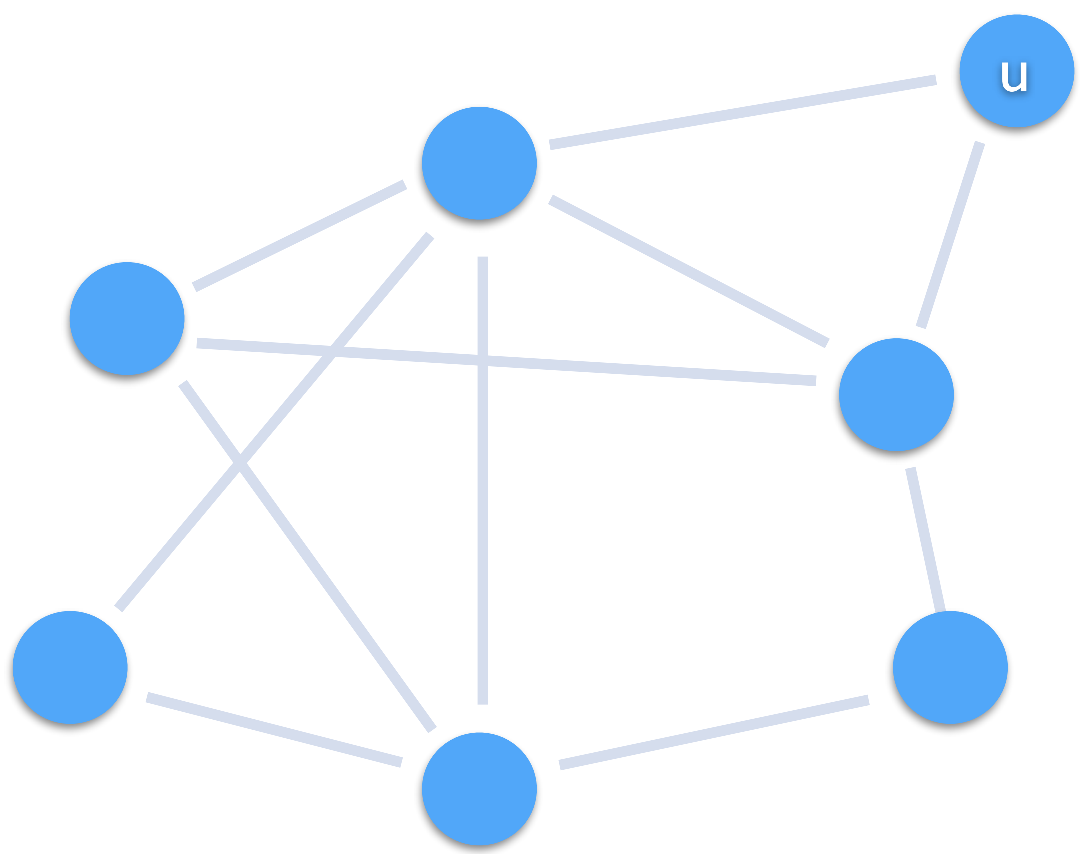
Private-Public problem

For each u, we like to execute some computation on $G \cup G_u$



Private-Public problem

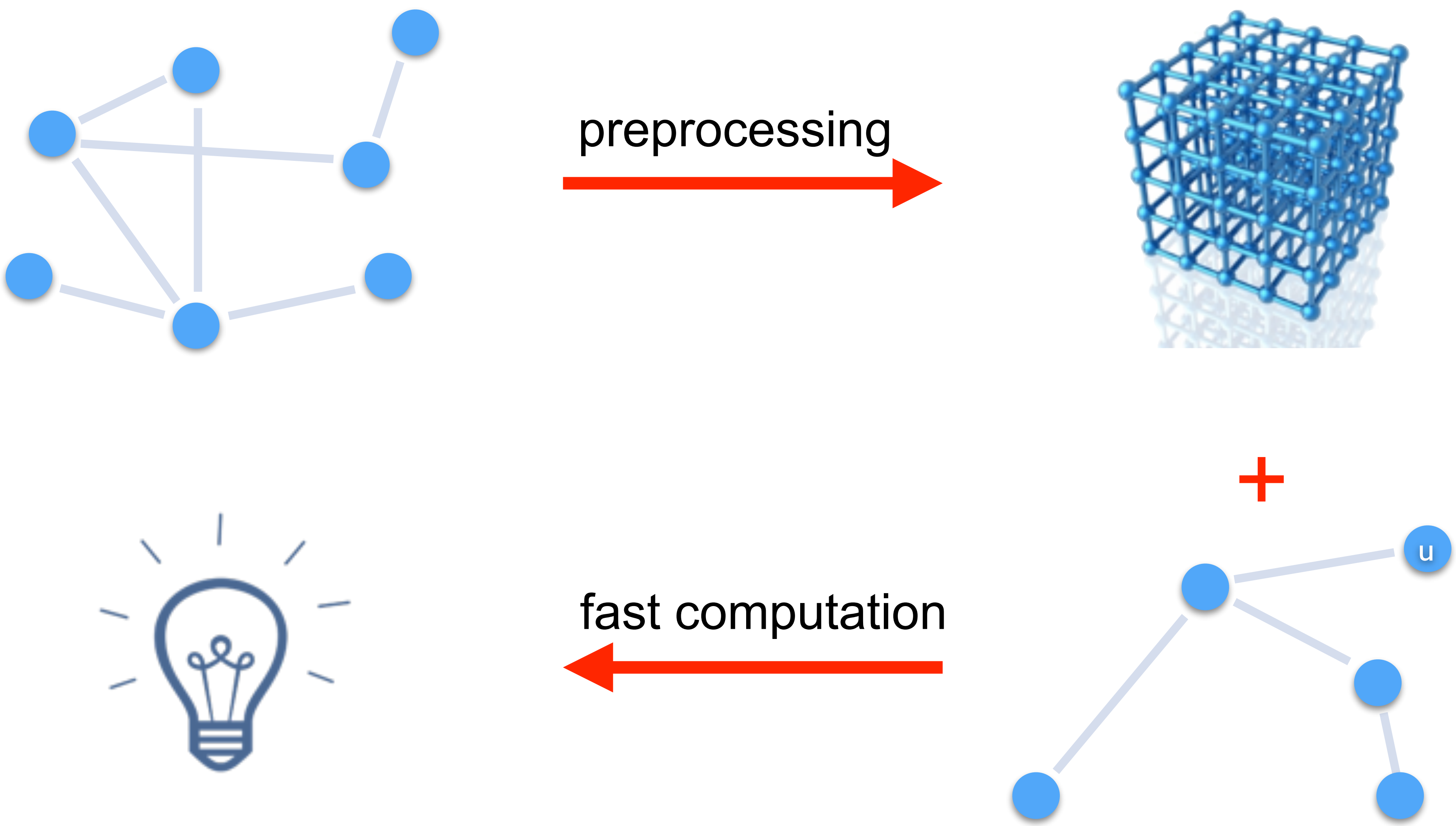
For each u, we like to execute some computation on $G \cup G_u$



Doing it naively is too expensive

Private-Public problem

Can we precompute data structure for G so that we can solve problems in $G \cup G_u$ efficiently?



Private-Public problem

Ideally

Preprocessing time: $\tilde{O}\left(|E_G|\right)$

Preprocessing space: $\tilde{O}\left(|V_G|\right)$

Post-processing time: $\tilde{O}\left(|E_{G_u}|\right)$

Problems Studied

(Approximation) Algorithms with provable bounds

- Reachability
- Approximate All-pairs shortest paths
- Correlation clustering
- Social affinity

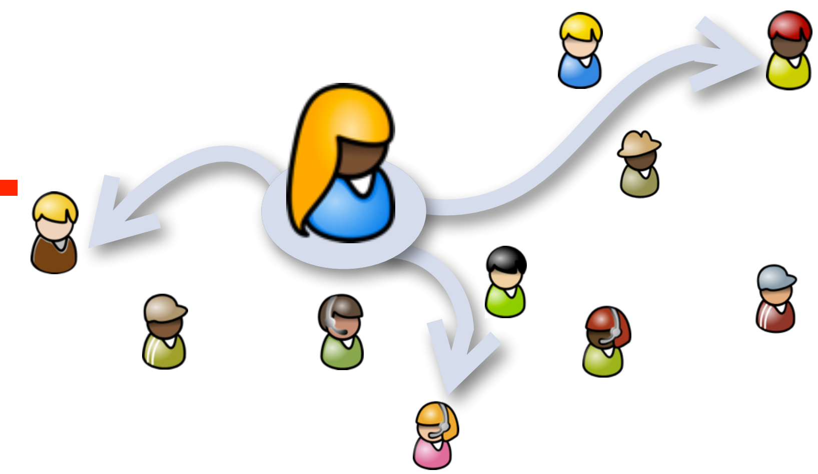
Heuristics

- Personalized PageRank
- Centrality measures

Problems Studied

Algorithms

- ***Reachability***
- Approximate All-pairs shortest paths
- Correlation clustering
- ***Social affinity***



Heuristics

- ***Personalized PageRank***
- Centrality measures

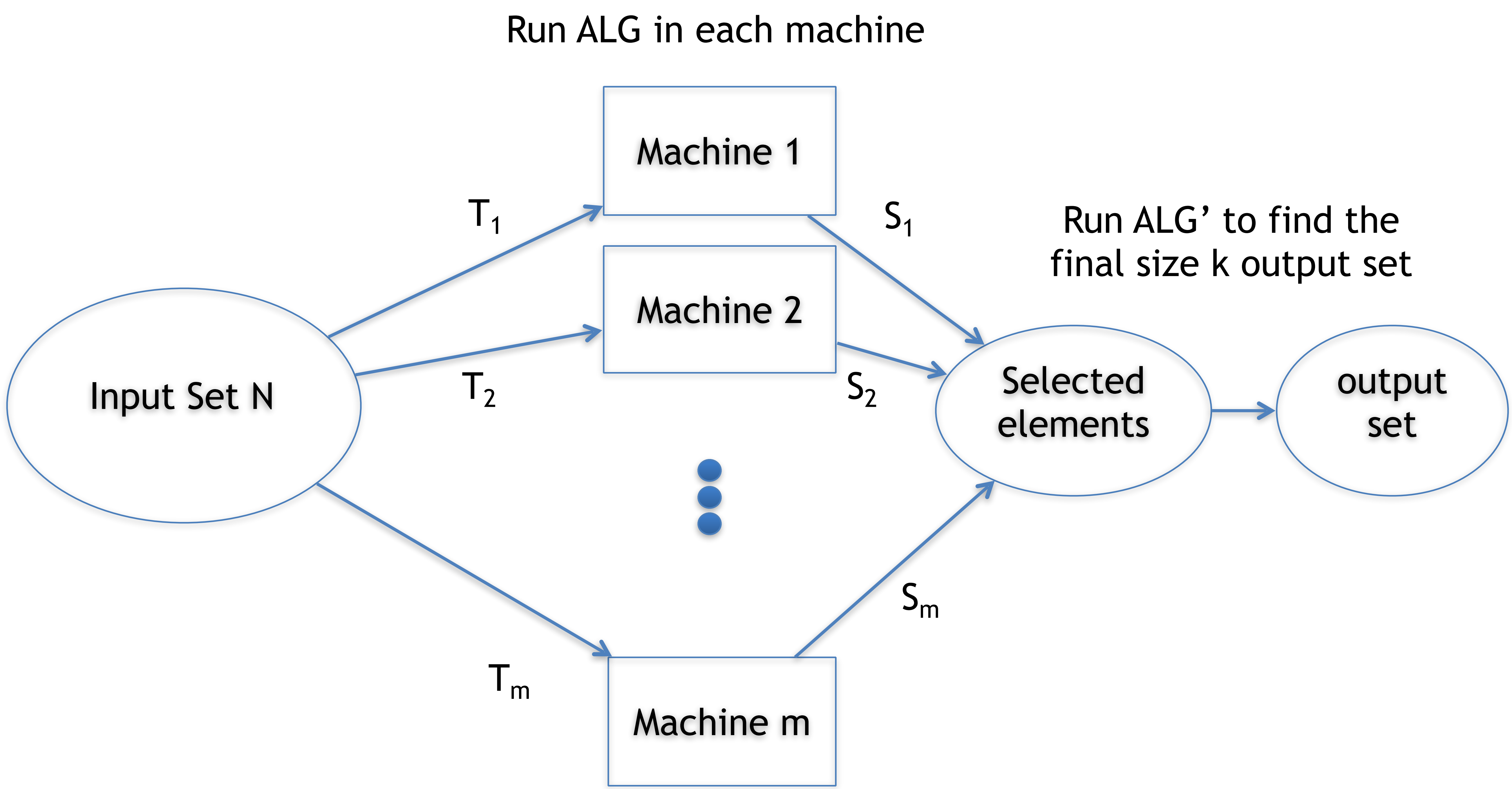
Part 2: Distributed Optimization

Distributed Optimization for NP-Hard Problems on Large Data Sets:

Two stories:

- Distributed Optimization via composable core-sets
  - Sketch the problem in composable instances
  - Distributed computation in constant (1 or 2) number of rounds
- Balanced Partitioning
  - Partition into ~equal parts & minimize the cut

Distributed Optimization Framework



Composable Core-sets

- **Technique for effective distributed algorithm**
  - **One or Two rounds of Computation**
  - **Minimal Communication Complexity**
  - **Can also be used in Streaming Models and Nearest Neighbor Search**

- **Problems**
  - Diversity Maximization
    - Composable Core-sets
    - Indyk, Mahabadi, Mahdian, Mirrokni, ACM PODS'14
  - Clustering Problems
    - Mapping Core-sets
    - Bateni, Bashkara, Lattanzi, Mirrokni, NIPS 2014
  - Submodular/Coverage Maximization:
    - Randomized Composable Core-sets
    - work by Mirrokni, ZadiMoghaddam, ACM STOC 2015

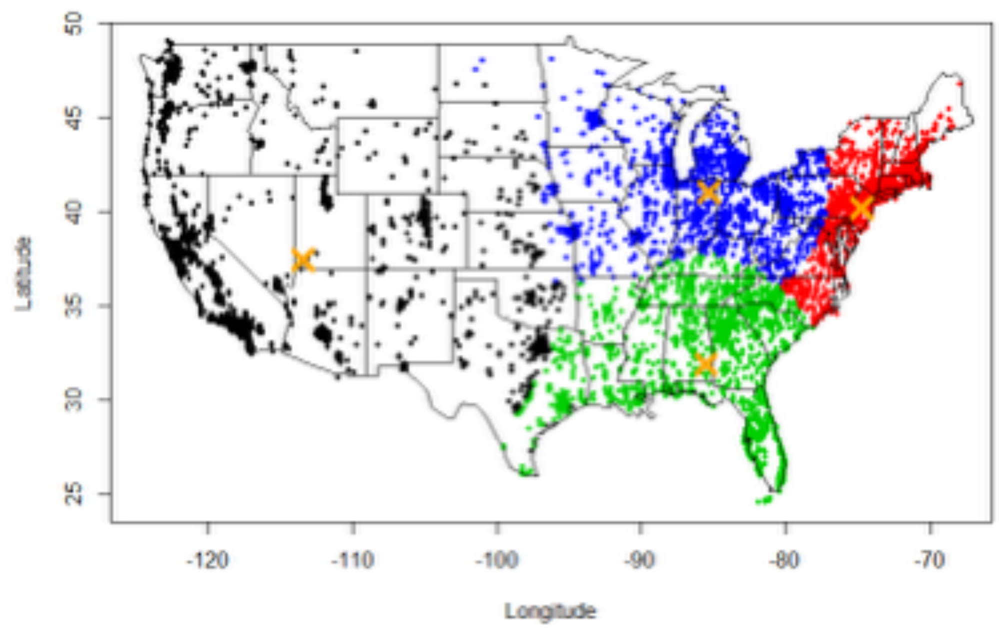
Problems considered:

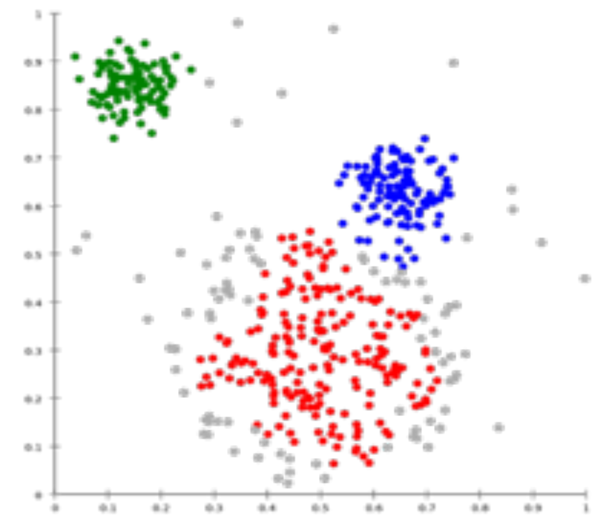
General: Find a set *S* of *k* items & maximize *f(S)*.

- **Diversity Maximization**: Find a set *S* of *k* points and maximize the sum of pairwise distances i.e. *diversity(S)*.
- **Capacitated/Balanced Clustering**: Find a set S of *k* centers and cluster nodes around them while minimizing the sum of distances to S.
- **Coverage/submodular Maximization**: Find a set S of *k* items. Maximize submodular function *f(S)*.

Distributed Clustering

Clustering: Divide data into groups containing





Minimize:

k-center : $\max_i \max_{u \in S_i} d(u, c_i)$

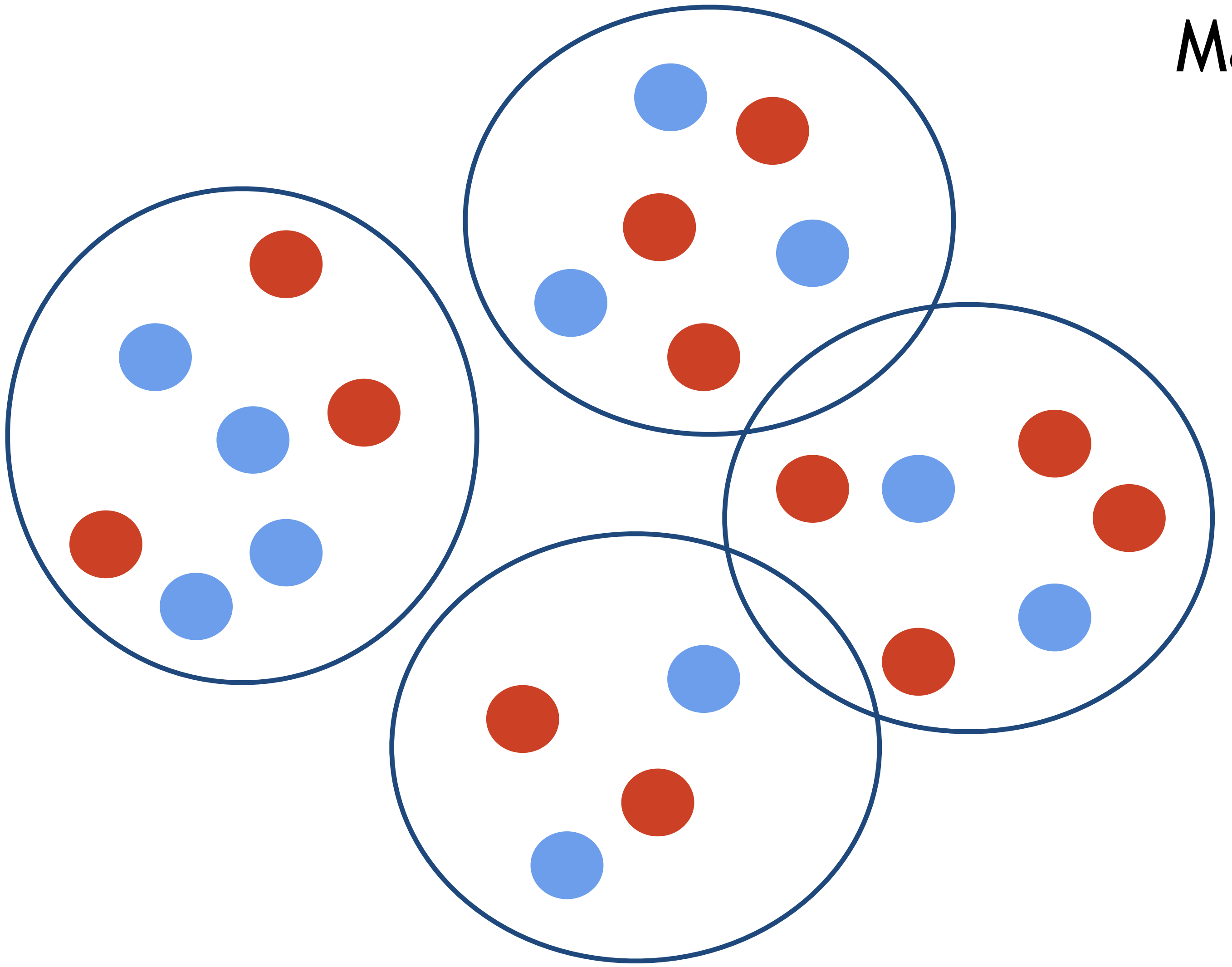
k-means : $\sum_i \sum_{u \in S_i} d(u, c_i)^2$

k-median : $\sum_i \sum_{u \in S_i} d(u, c_i)$

Metric space *(d, X)*

α-approximation algorithm: cost less than α*OPT

Distributed Clustering



Many objectives: k-means, k-median, k-center,…

minimize max cluster radius

Framework:

- Divide into chunks $V_1, V_2, \ldots, V_m$

- Come up with "representatives" S_i on machine i << $|V_i|$

- Solve on union of S_i, others by closest rep.

Balanced/Capacitated Clustering

Theorem(BhaskaraBateniLattanziM. NIPS'14): distributed balanced clustering with

- approx. ratio: (small constant) * (best "single machine" ratio)
- rounds of MapReduce: constant (2)
- memory: ~(n/m)^2 with m machines

Works for all L_p objectives.. (includes k-means, k-median, k-center)

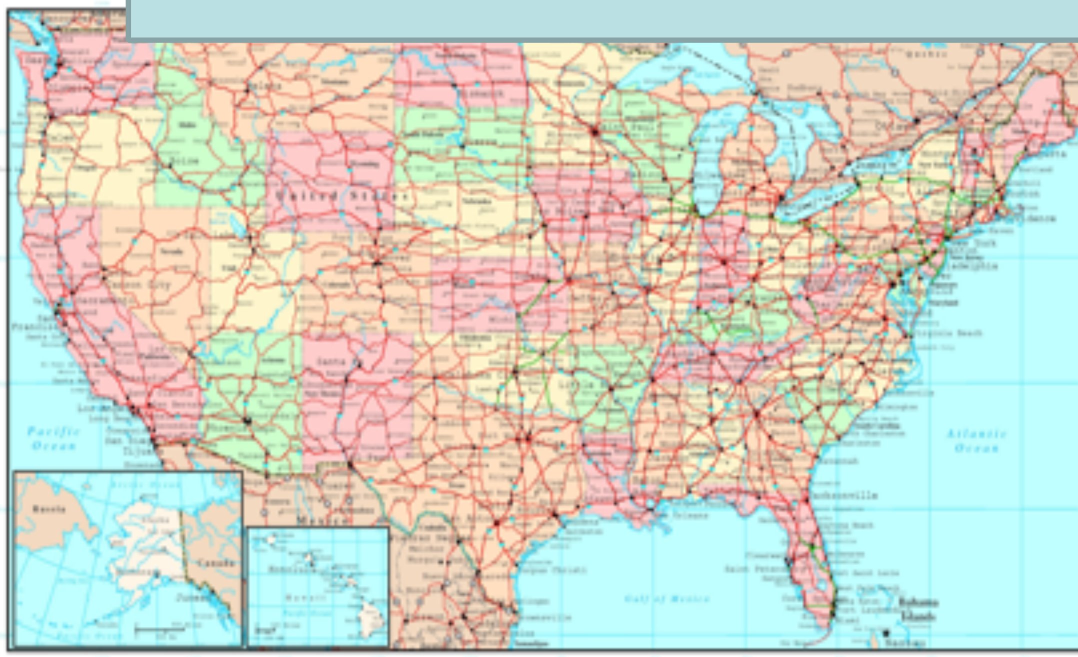
Improving Previous Work

- Bahmani, Kumar, Vassilivitskii, Vattani: Parallel K-means++
- Balcan, Enrich, Liang: Core-sets for k-median and k-center

Experiments

Aim: Test algorithm in terms of (a) scalability, and (b) quality of solution obtained

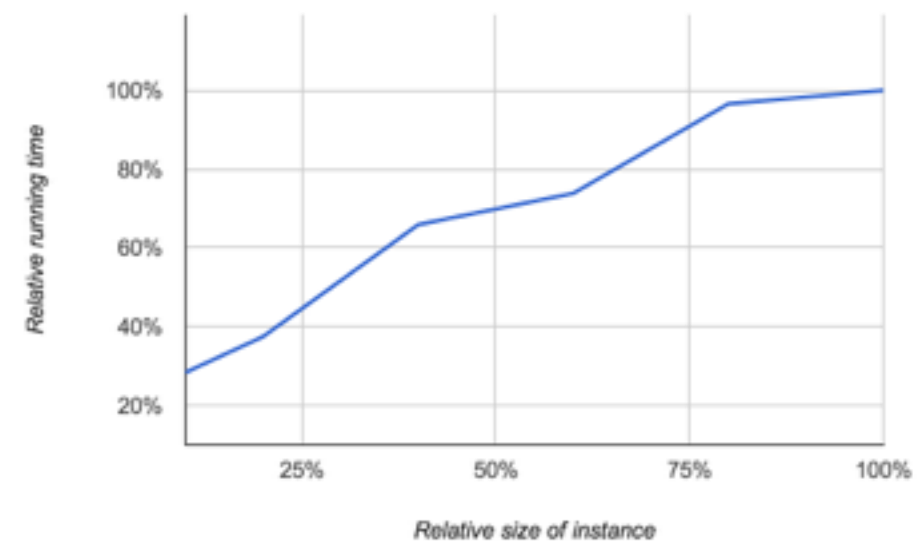
Setup: Two “base” instances and subsamples (used *k*=1000, #machines = 200)



US graph: N = x0 Million
distances: geodesic

World graph: N = x00 Million
distances: geodesic

	size of seq. inst.	increase in OPT
US	1/300	**1.52**
World	1/1000	**1.58**



Accuracy: analysis pessimistic

Scaling: sub-linear

Coverage/Submodular Maximization

- Max-Coverage:
  - Given: A family of subsets $S_1 \ldots S_m$
  - Goal: choose k subsets $S'_1 \ldots S'_k$ with the maximum union cardinality.
- Submodular Maximization:
  - Given: A submodular function f
  - Goal: Find a set S of k elements & maximize $f(S)$.
- Applications: Data summarization, Feature selection, Exemplar clustering, …

Bad News!

- Theorem[IndykMahabadiMahdianM PODS'14] There exists no better than $\frac{\log k}{\sqrt{k}}$ approximate composable core-set for submodular maximization.

- Question: What if we apply *random partitioning*?

YES! Concurrently answered in two papers:

- Barbosa, Ene, Nugeon, Ward: ICML'15.
- M.,ZadiMoghaddam: STOC'15.

Summary of Results [M. ZadiMoghaddam - STOC'15]

1. A class of 0.33-approximate randomized composable core-sets of size k for non-monotone submodular maximization.
2. Hard to go beyond ½ approximation with size k. Impossible to get better than 1-1/e.
3. 0.58-approximate randomized composable core-set of size 4k for monotone f. Results in 0.54-approximate distributed algorithm.
4. For small-size composable core-sets of k' less than k: *sqrt{k'/k}*-approximate randomized composable core-set.

$(2-\sqrt{2})$-approximate Randomized Core-set

- Positive Result [M, ZadiMoghaddam]: If we increase the output sizes to be 4k, Greedy will be $(2-\sqrt{2})-o(1) \geq 0.585$-approximate randomized core-set for a monotone submodular function.
- Remark: In this result, we send each item to C random machines instead of one. As a result, the approximation factors are reduced by a $O(\ln(C)/C)$ term.

Summary: composable core-sets

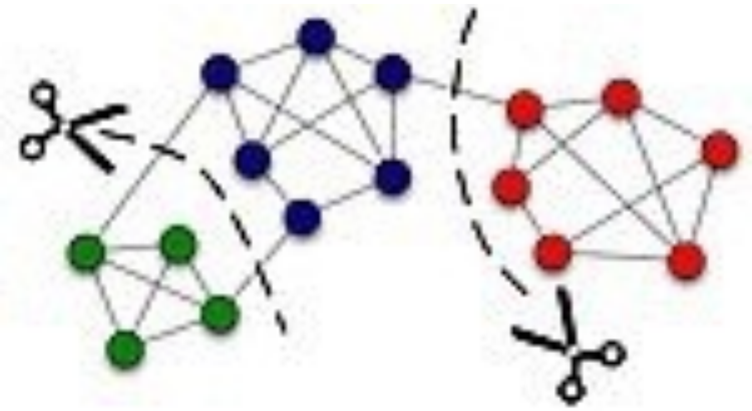
- Diversity maximization (PODS'14)
  - Apply constant-factor composable core-sets
- Balanced clustering (k-center, k-median & k-means) (NIPS'14)
  - Apply Mapping Core-sets → constant-factor
- Coverage and Submodular maximization (STOC'15)
  - Impossible for deterministic composable core-set
  - Apply *randomized core-sets* → 0.54-approximation

- Future:
  - Apply core-sets to other ML/graph problems, feature selection.
  - For submodular:
    - 1-1/e-approximate core-set
    - 1-1/e-approximation in 2 rounds (even with multiplicity)?

Distributed Balanced Partitioning via Linear Embedding

- Based on work by Aydin, Bateni, Mirrokni

Balanced Partitioning Problem



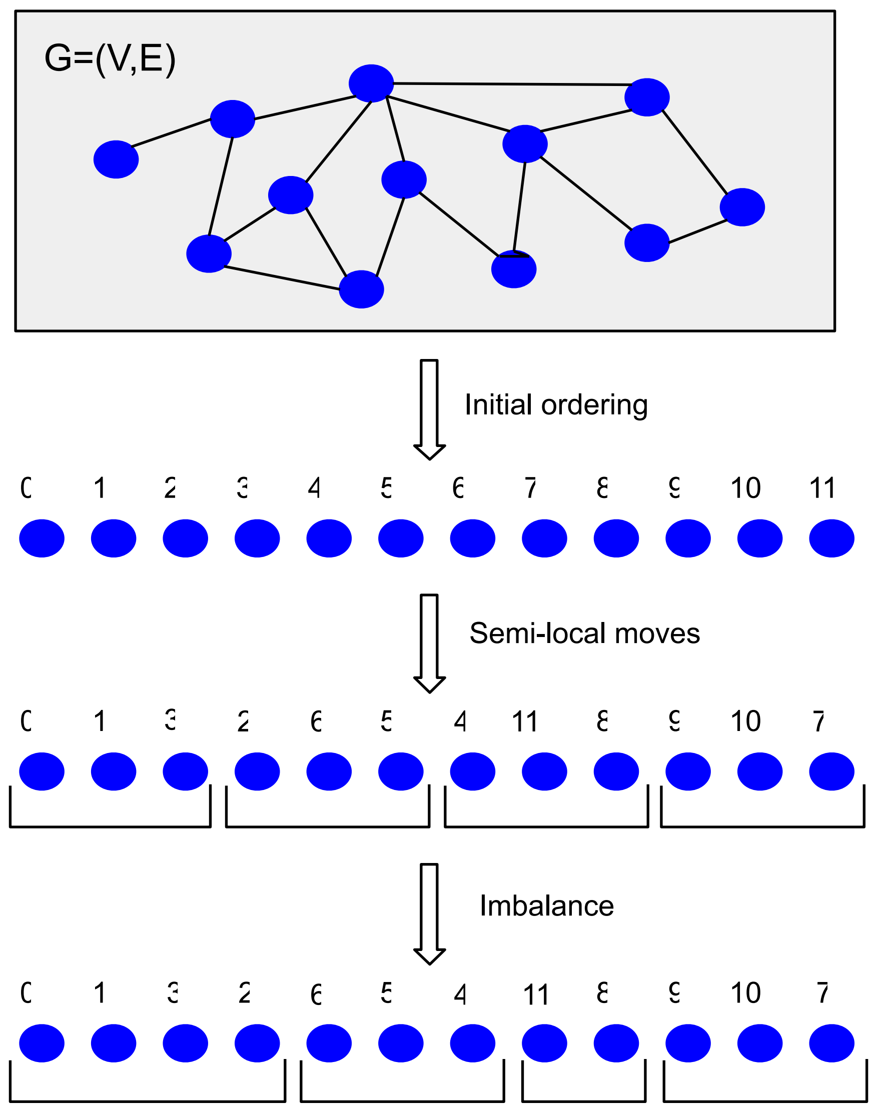
- Balanced Partitioning:
  - Given graph **G**(**V**, **E**) with edge weights
  - Find **k** clusters of approximately the same size
  - Minimize Cut, i.e., #intercluster edges

- Applications:
  - *Minimize communication complexity* in distributed computation
  - *Minimize number of multi-shard queries* while serving an algorithm over a graph, e.g., in computing shortest paths or directions on Maps

Outline of Algorithm

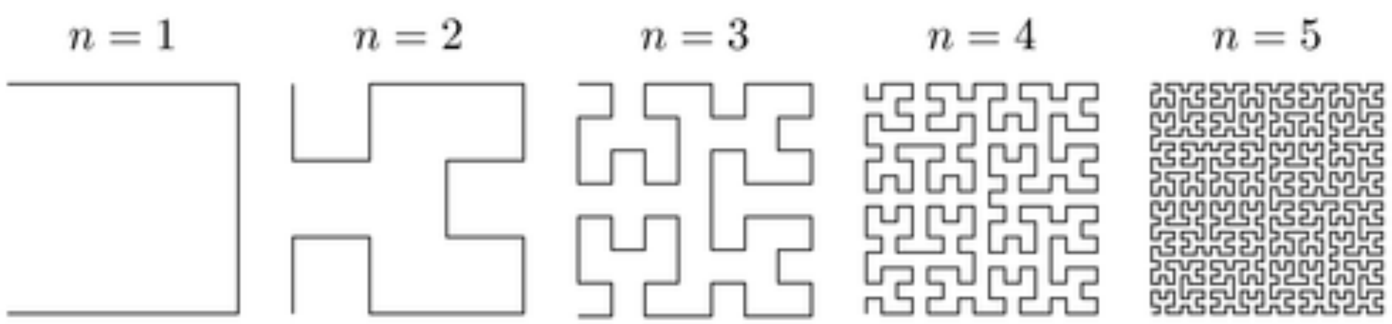
Three-stage Algorithm:

1. Reasonable Initial Ordering
   a. Space-filling curves
   b. Hierarchical clustering
2. Semi-local moves
   a. Min linear arrangement
   b. Optimize by random swaps
3. Introduce imbalance
   a. Dynamic programming
   b. Linear boundary adjustment
   c. Min-cut boundary optimization

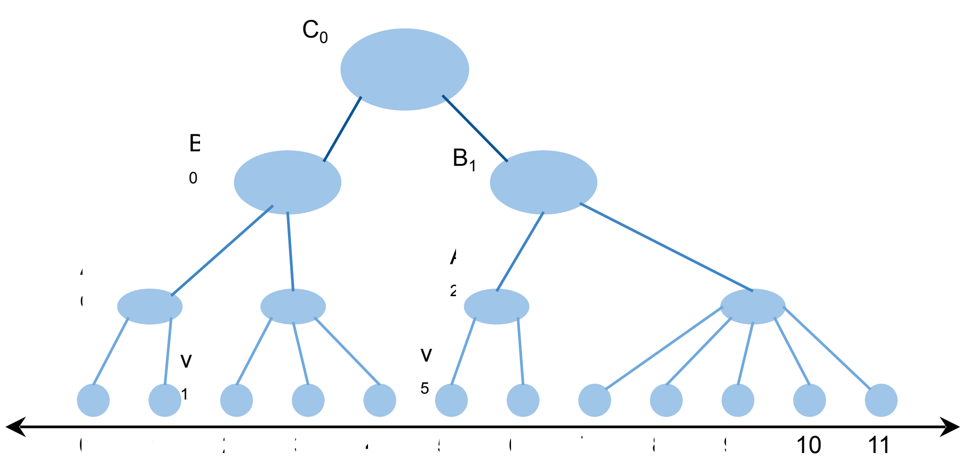


Step 1 - Initial Embedding

- Space-filling curves (Geo Graphs)



- Hierarchical clustering (General Graphs)



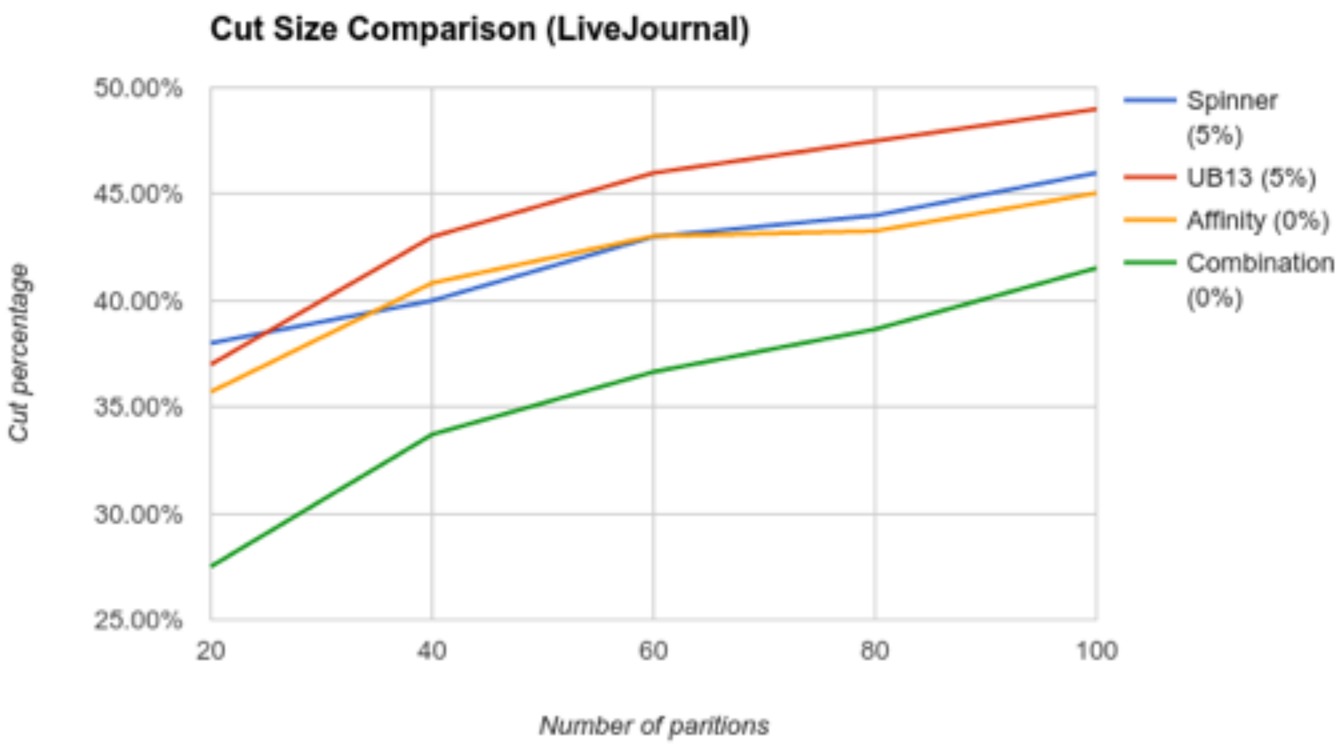
Datasets

- Social graphs
  - **Twitter**: 41M nodes, 1.2B edges
  - **LiveJournal**: 4.8M nodes, 42.9M edges
  - **Friendster**: 65.6M nodes, 1.8B edges
- Geo graphs
  - **World** graph > 1B edges
  - **Country** graphs (filtered)

Related Work

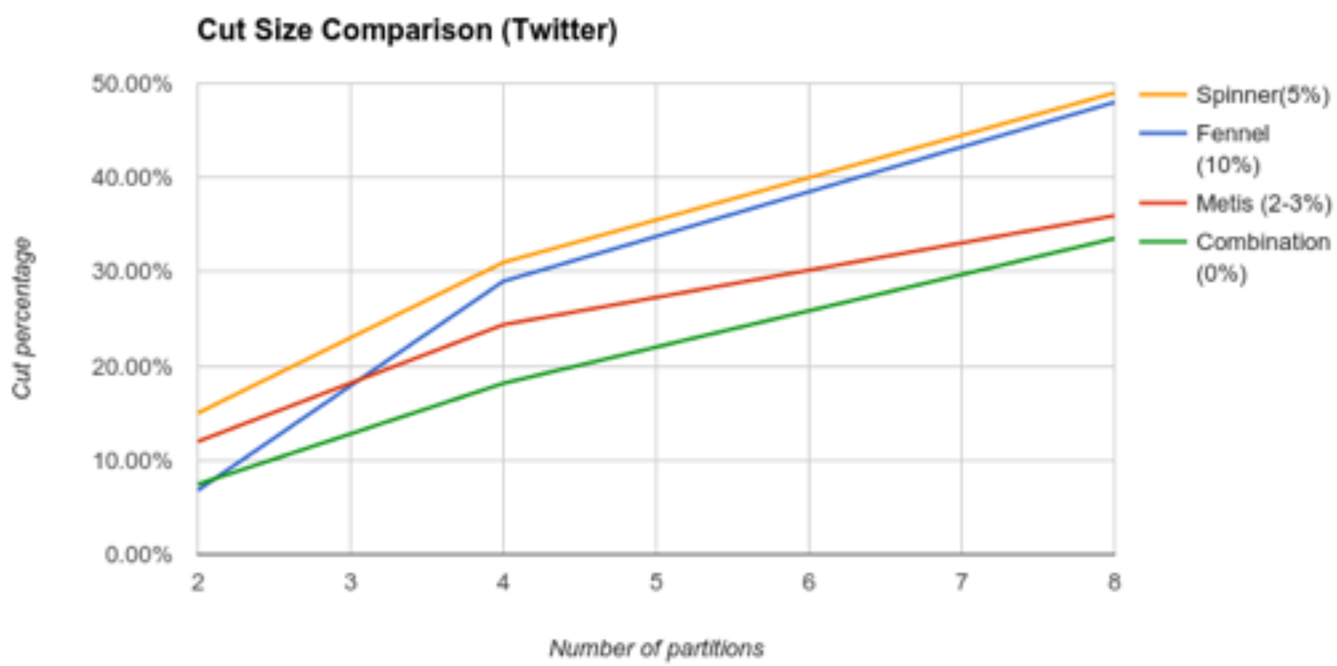
- FENNEL, WSDM'14 [Tsourakakis et al.]
  - Microsoft Research
  - Streaming algorithm
- UB13, WSDM'13 [Ugander & Backstorm]
  - Facebook
  - Balanced label propagation
- Spinner, (very recent) arXiv [Martella et al.]
- METIS
  - In-memory

Comparison to Previous Work



k	Spinner (5%)	UB13 (5%)	Affinity (0%)	Our Alg (0%)
20	38%	37%	35.71%	27.5%
40	40%	43%	40.83%	33.71%
60	43%	46%	43.03%	36.65%
80	44%	47.5%	43.27%	38.65%
100	46%	49%	45.05%	41.53%

Comparison to Previous Work



k	Spinner (5%)	Fennel (10%)	Metis (2-3%)	Our Alg (0%)
2	15%	6.8%	11.98%	7.43%
4	31%	29%	24.39%	18.16%
8	49%	48%	35.96%	33.55%

Outline: Part 3

Practice: Algorithms+System Research

Two stories:

- Connected components in MapReduce & Beyond

  Going beyond MapReduce to build efficient tool in practice.
- ASYMP

  A new asynchronous message passing system.

Graph Mining Frameworks

Applying various frameworks to graph algorithmic problems

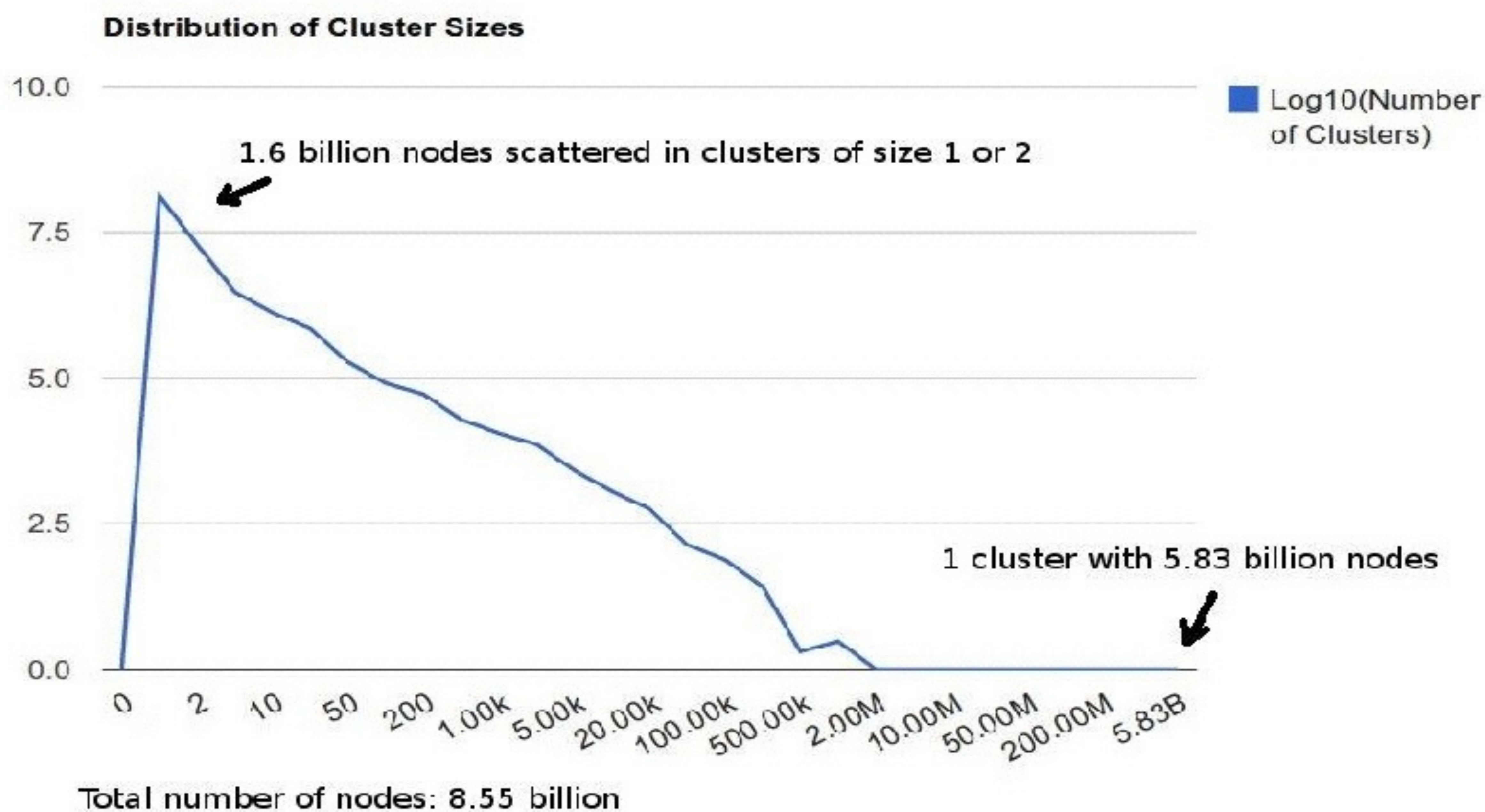
- **Iterative MapReduce (Flume):**
  - More widely fault-tolerant available tool
  - Can be optimized with algorithmic tricks
- **Iter. MapReduce + DHT Service (Flume):**
  - Better speed compared to MR
- **Pregel:**
  - Good for synch. computation w/ many rounds
  - Simpler implementation
- **ASYMP (ASYnchronous Message-Passing):**
  - More scalable/More efficient use of CPU
  - Asych. self-stabilizing algorithms

Metrics for MapReduce algorithms

- **Running Time**
  - Number of MapReduce rounds
  - Quasi-linear time processing of inputs
- **Communication Complexity**
  - Linear communication per round
  - Total communication across multiple rounds
- **Load Balancing**
  - No mapper or reducer should be overloaded
- **Locality of the messages**
  - Sending messages locally when possible
  - Use the same key for mapper/reducer when possible
  - Effective while using MR with DHT (more later)

Connected Components: Example output

Web Subgraph: 8.5B nodes, 700B edges



Prior Work: Connected Components in MR

Connected components in MapReduce,
Rastogi et al, ICDE'12

Algorithm	#MR Rounds	Communication / Round	Practice
Hash-Min	*D (Diameter)*	*O(m+n)*	*Many rounds*
Hash-to-All	*Log D*	*O(n*	*Long rounds*
Hash-to-Min	*Open*	*O(nlog n+m)*	*BEST*
Hash-Greater -to-Min	*3 log D*	*2(n+m)*	*OK, but not the best*

Connected Components: Summary

- Connected Components in MR & MR+DHT
  - Simple, local algorithms with $O(\log^2 n)$ round complexity
  - Communication efficient (#edges non-increasing)
- Use Distributed HashTable Service (DHT) to improve # rounds to O~(log n) [from ~20 to ~5]
- Data: Graphs with ~XT edges. Public data with 10B edges
- Results:
  - MapReduce: 10-20 times faster than HashtoMin
  - MR+DHT: 20-40 times faster than HashtoMin
  - ASYMP: A simple algorithm in ASYMP: 25-55 times faster than HashtoMin

KiverisLattnziM.RastogiVassilivitskii, SOCC'14.

ASYMP:ASYnchrouns Message Passing

- ASYMP: New graph mining framework
- Compare with MapReduce, Pregel
  - Computation does not happen in a synchronize number of rounds
  - Fault-tolerance implementation is also asynchronous
  - More efficient use of CPU cycles
- We study its fault-tolerance and scalability
- Impressive empirical performance (e.g., for connectivity and shortest path)

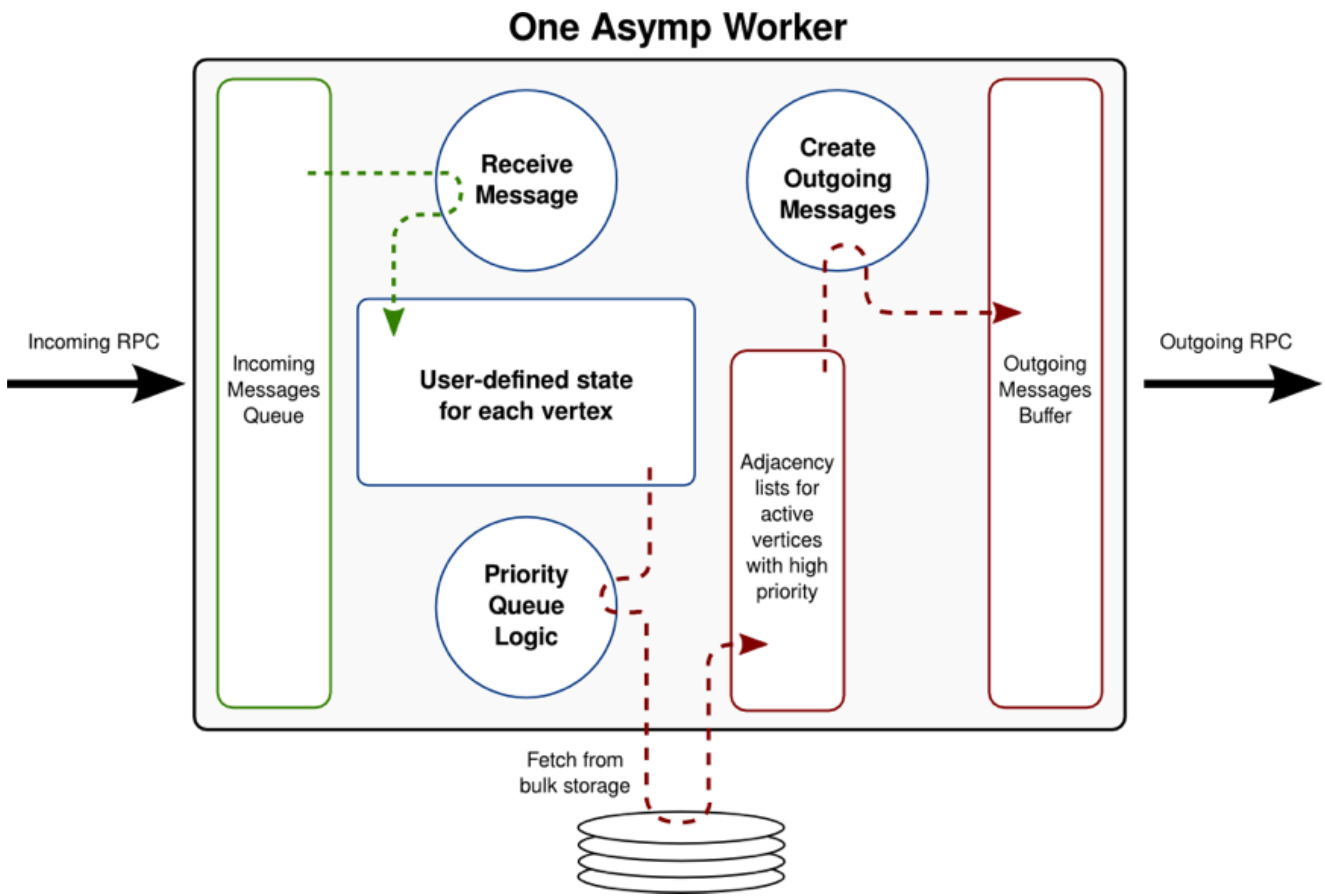
Fleury, Lattanzi, M.: ongoing.

Asymp model

- Nodes are distributed among many machines (workers)
- Each node keeps a *state* and send messages to its neighbors.
- Each machine has a *priority queue* for sending messages to other machines

- **Initialization**: Set nodes' states & activate some nodes
- Main **Propagation** Loop (Roughly):
  - Until all nodes converge to a stable state:
    - Asynchronously update states and send top messages in each priority queue
- **Stop Condition**: Stop when priority queues are empty…

Asymp worker design



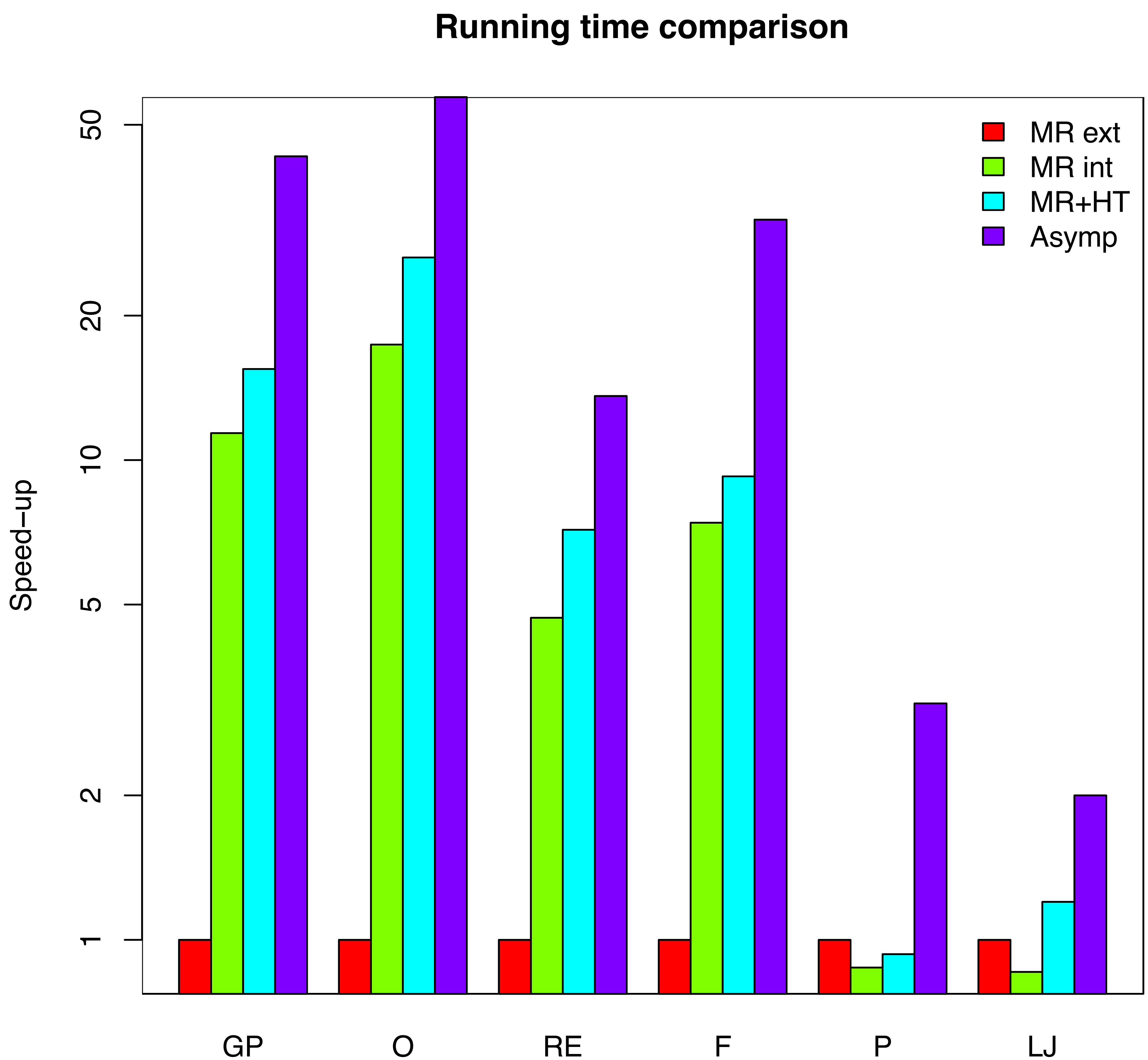
Data Sets

- 5 Public and 5 Internal Google graphs e.g.
  - UK Web graph: 106M nodes, 6.6B edges [Public]
  - Google+ subgraph: 178M nodes, 2.9B edges
  - Keyword similarity : 371M nodes, 3.5B edges
  - Document similarity: 4,700M nodes, 452B edges

- Sequence of **Web subgraphs**:
  - ~1B, 3B, 9B, 27B core nodes [16B, 47B, 110B, 356B]
  - ~36B, 108B, 324B, 1010B edges respectively

- Sequence of **RMAT graphs** [Synthetic and Public]:
  - ~2^{26}, 2^{28}, 2^{30}, 2^{32}, 2^{34} nodes
  - ~2B, 8B, 34B, 137B, 547B edges respectively.

Comparison with best MR algorithms



Asymp Fault-tolerance

- *Asynchronous* Checkpointing:
  - Store the current states of nodes once in a while
- Upon failure of a machine:
  - Fetch the last recorded state of each node, &
  - Activate these nodes (send messages to neighbors), and ask them to resend the messages it may have lost.

- Therefore, a *self-stabilizing* algorithm works correctly in ASYMP.

- Example: Dijsktra Shortest Path Algorithm

Impact of failures on running time

- ***Make a fraction/all of machines fail over time.***
  - Question: What is the impact of frequent failures?
- **Let *D* be the running time without any failures. Then**

% Machine Failures over the whole period (→ #per batch)	*6% of machine failures at a time*	*12% of machine failures at a time*
50%	***Time ~= 2D***	***Time ~= 1.4D***
100%	***Time ~= 3.6D***	***Time ~= 3.2D***
200%	***Time ~= 5.3D***	***Time ~= 4.1D***

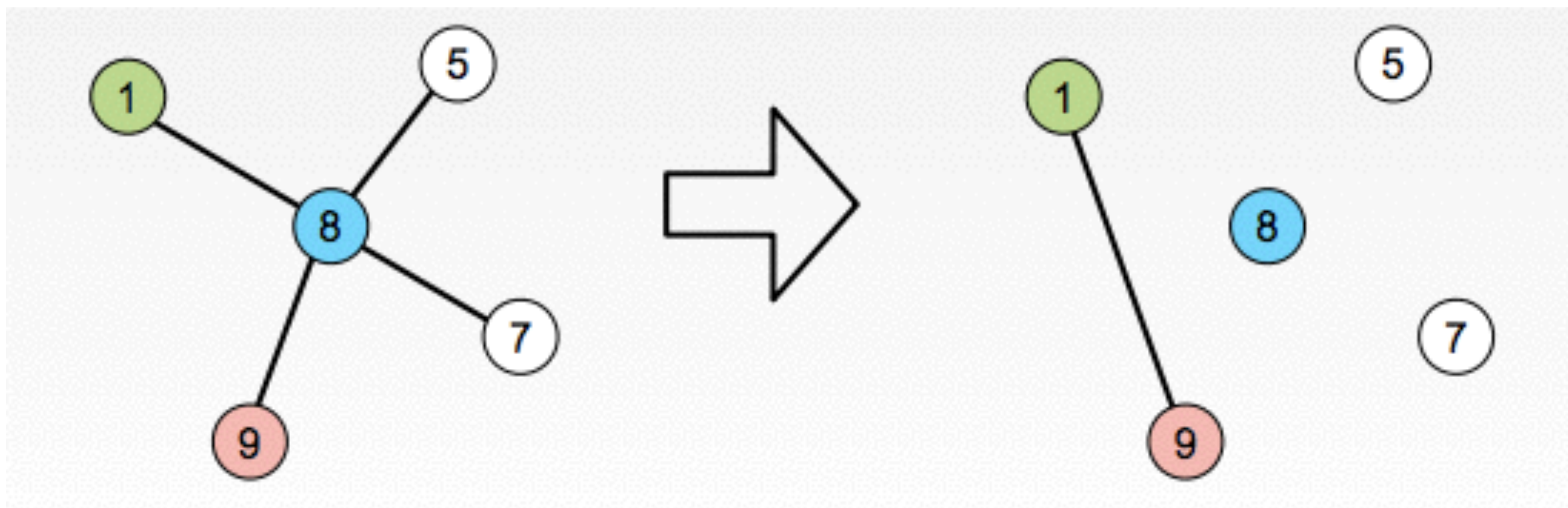
- **More frequent small-size failures is worse than less frequent large-size failures**
  - More robust against group-machine failures

Questions?

Thank you!

Algorithmic approach: Operation 1

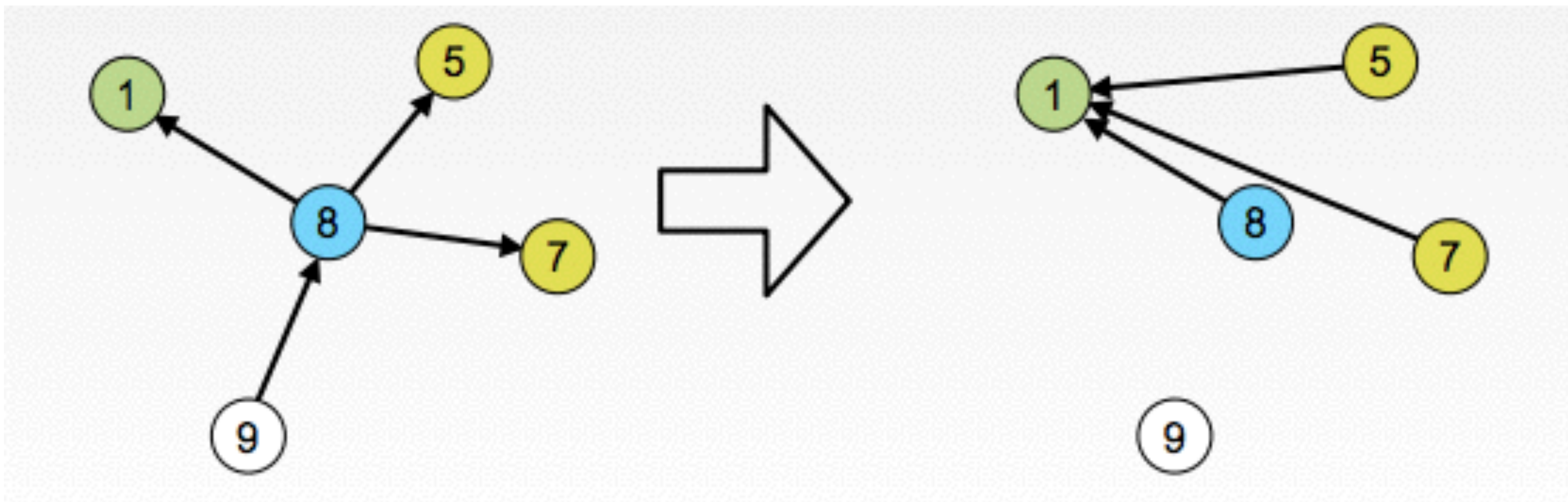
Large-star(v): Connect all strictly larger neighbors to the min neighbor including self



- Do this in parallel on each node & build a new graph
- Theorems (KLMRV’14):
  - Executing Large-star in parallel preserves connectivity
  - Every Large-star operation reduces height of tree by a constant factor

Algorithmic approach: Operation 2

Small-star(v): Connect all smaller neighbors and self to the min neighbor including self



- Connect all parents to the minimum parent

- Theorem(KLMRV'14):
  - Executing Small-star in parallel preserves connectivity

Final Algorithm: Combine Operations

- **Input**
  - Set of edges with a unique ID per node

Algorithm:

Repeat until convergence

- Repeated until convergence
  - Large-Star
- Small-star

- Theorem(KLMRV'14):
  - The above algorithm converges in $O(\log^2 n)$ rounds.

Improved Connected Components in MR

- Idea 1: Alternate between Large-Star and Small-Star
  - Less #rounds compared to Hash-to-Min, Less Communication compared to Hash-Greater-to-Min
  - Theory: Provable $O(\log^2 n)$ MR rounds
- Optimization: Avoid large-degree nodes by branching them into a tree of height two
- Practice:
  - Graphs with 1T edges. Public data w/ 10B edges
  - 2 to 20 times faster than Hash-to-Min (Best of ICDE'12)
  - Takes 5 to 22 rounds on these graphs

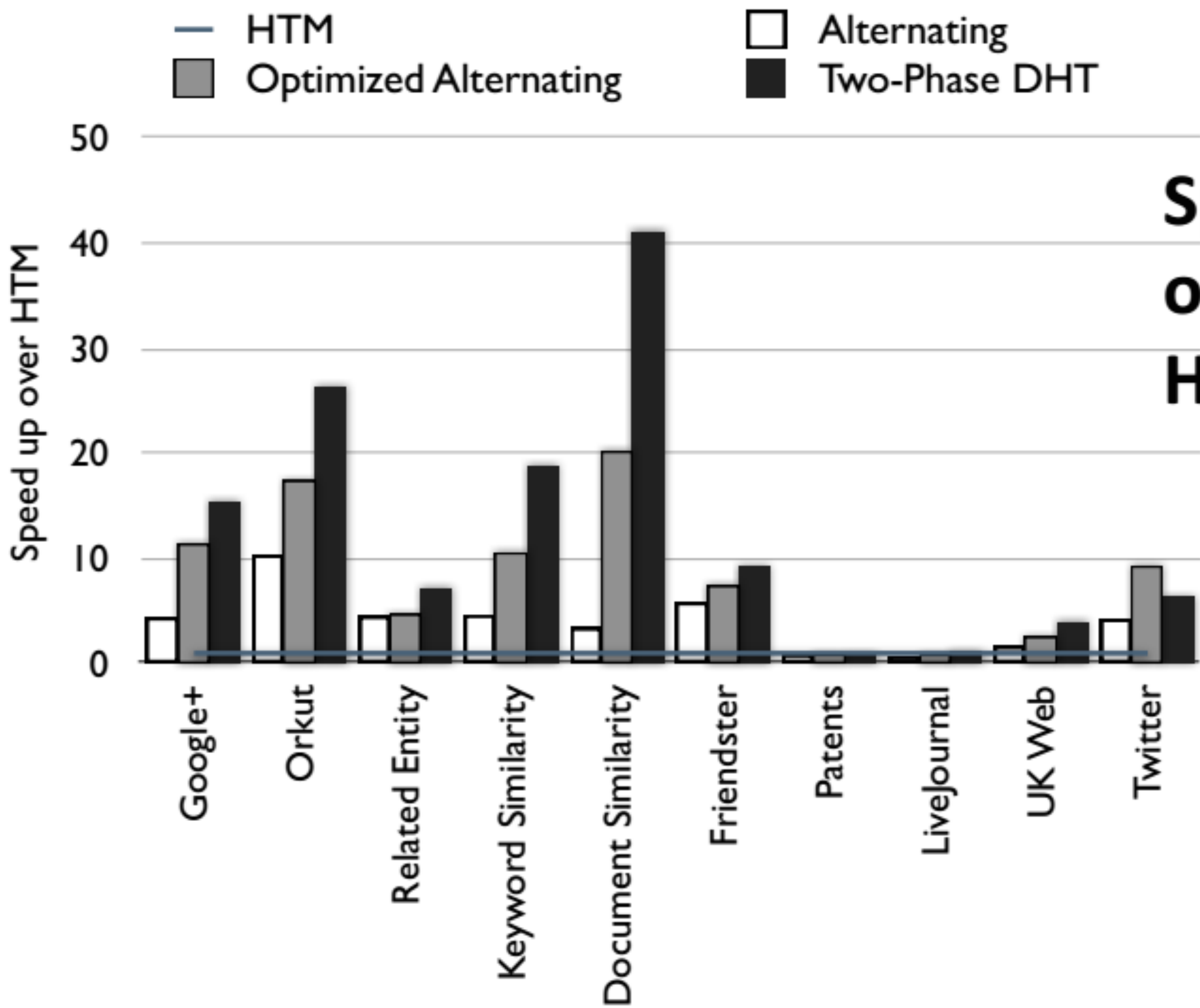
CC in MR + DHT Service

- Idea 2: Use Distributed HashTable (DHT) service to save in #rounds
  - After small #rounds (e.g., after 3rd round), consider all active cluster IDs, and resolve their mapping in an array in memory (e.g. using DHT)
  - Theory: $O\sim(\log n)$ MR rounds + $O(n/\log n)$ memory.
  - Practice:
    - Graphs with 1T edges. Public data w/ 10B edges.
    - 4.5 to 40 times faster than Hash-to-Min (Best of ICDE'12 paper), and 1.5 to 3 times faster than our best pure MR implementation. Takes 3 to 5 rounds on these graphs.

Data Sets

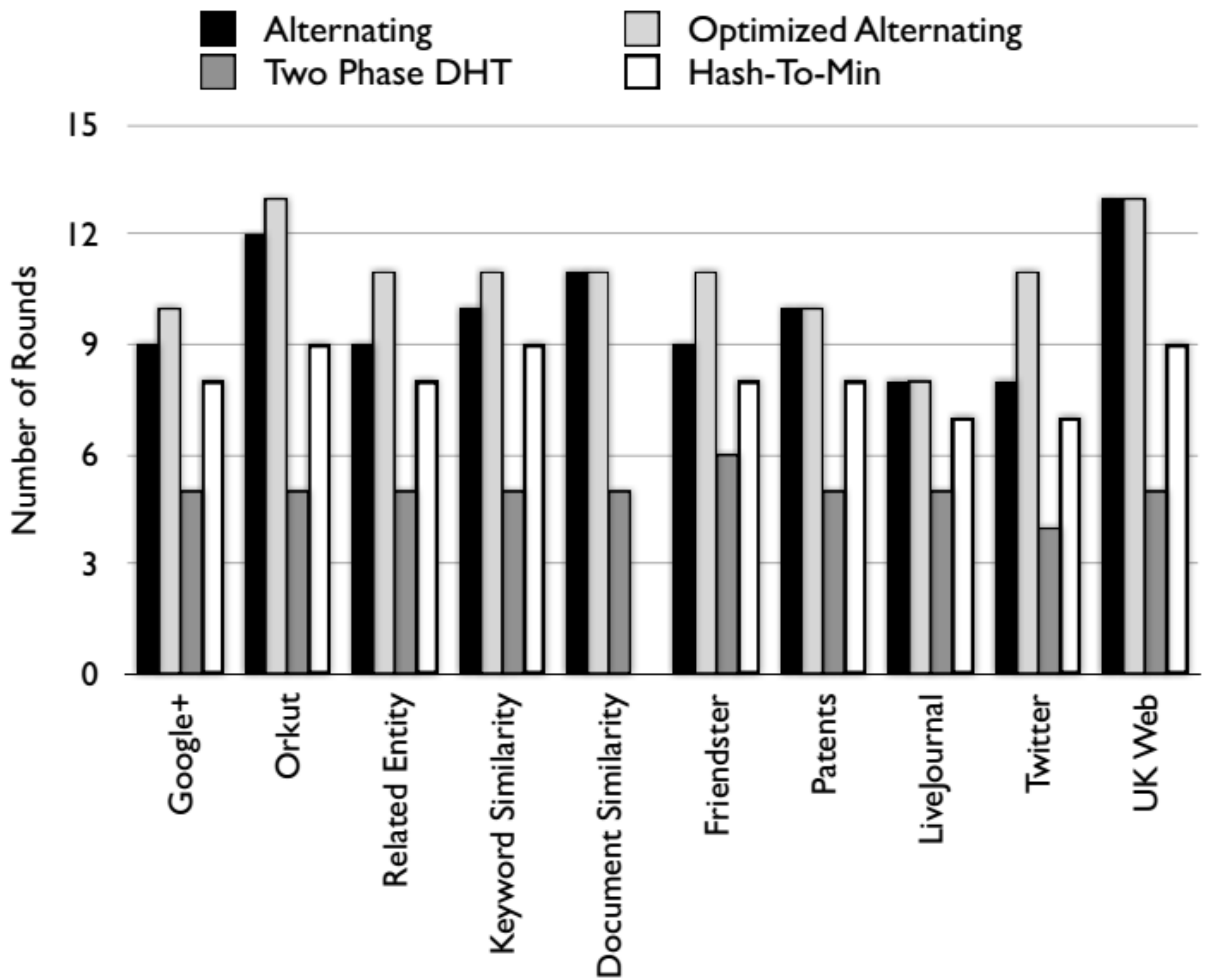
- 5 Public and 5 Internal Google graphs e.g.
  - UK Web graph: 106M nodes, 6.6B edges [Public]
  - Google+ subgraph: 178M nodes, 2.9B edges
  - Keyword similarity : 371M nodes, 3.5B edges
  - Document similarity: 4,700M nodes, 452B edges
- Sequence of **RMAT graphs** [Synthetic and Public]:
  - ~2^{26}, 2^{28}, 2^{30}, 2^{32}, 2^{34} nodes
  - ~2B, 8B, 34B, 137B, 547B edges respectively.
- Algorithms:
  - Min2Hash
  - Alternate Optimized (MR-based)
  - Our best MR + DHT Implementation
  - Pregel Implementation

Speedup: Comparison with HTM

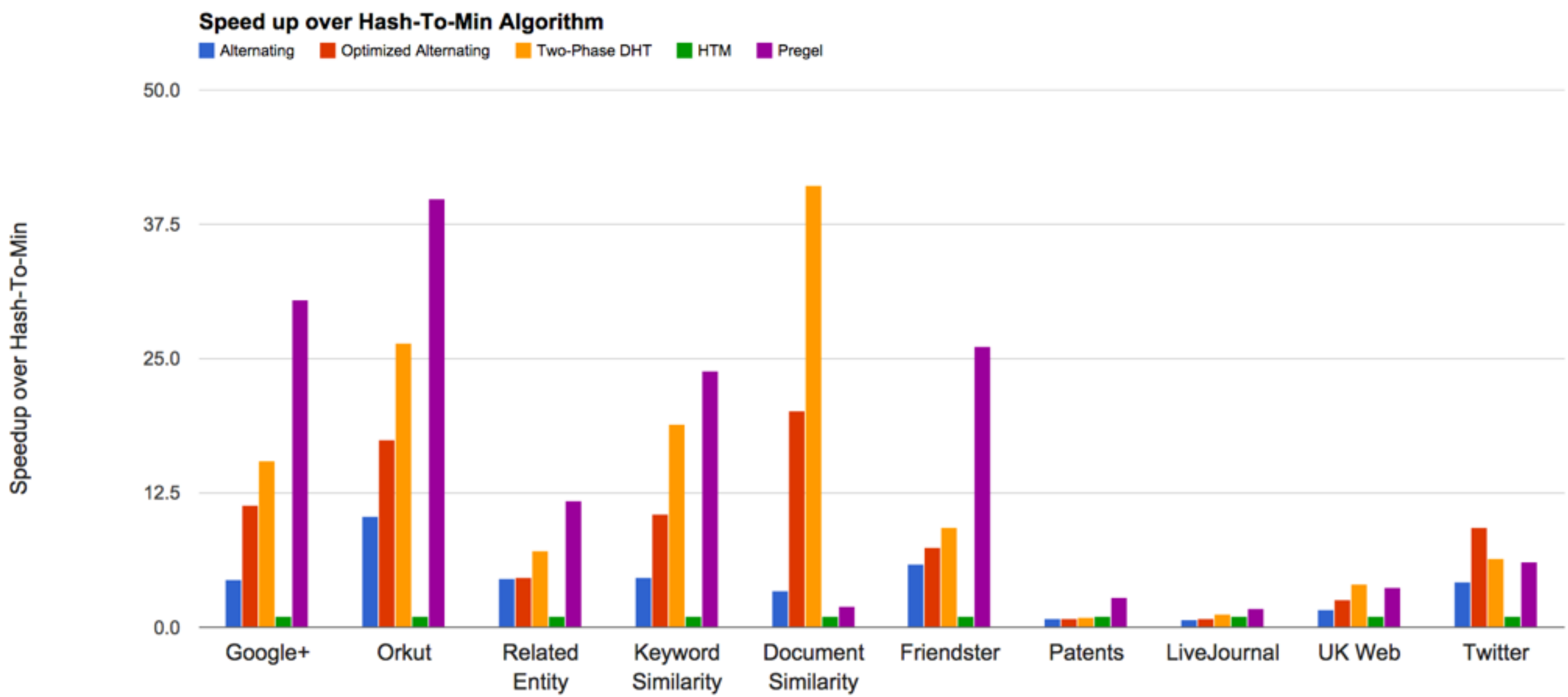


Speedup over Hash2Min

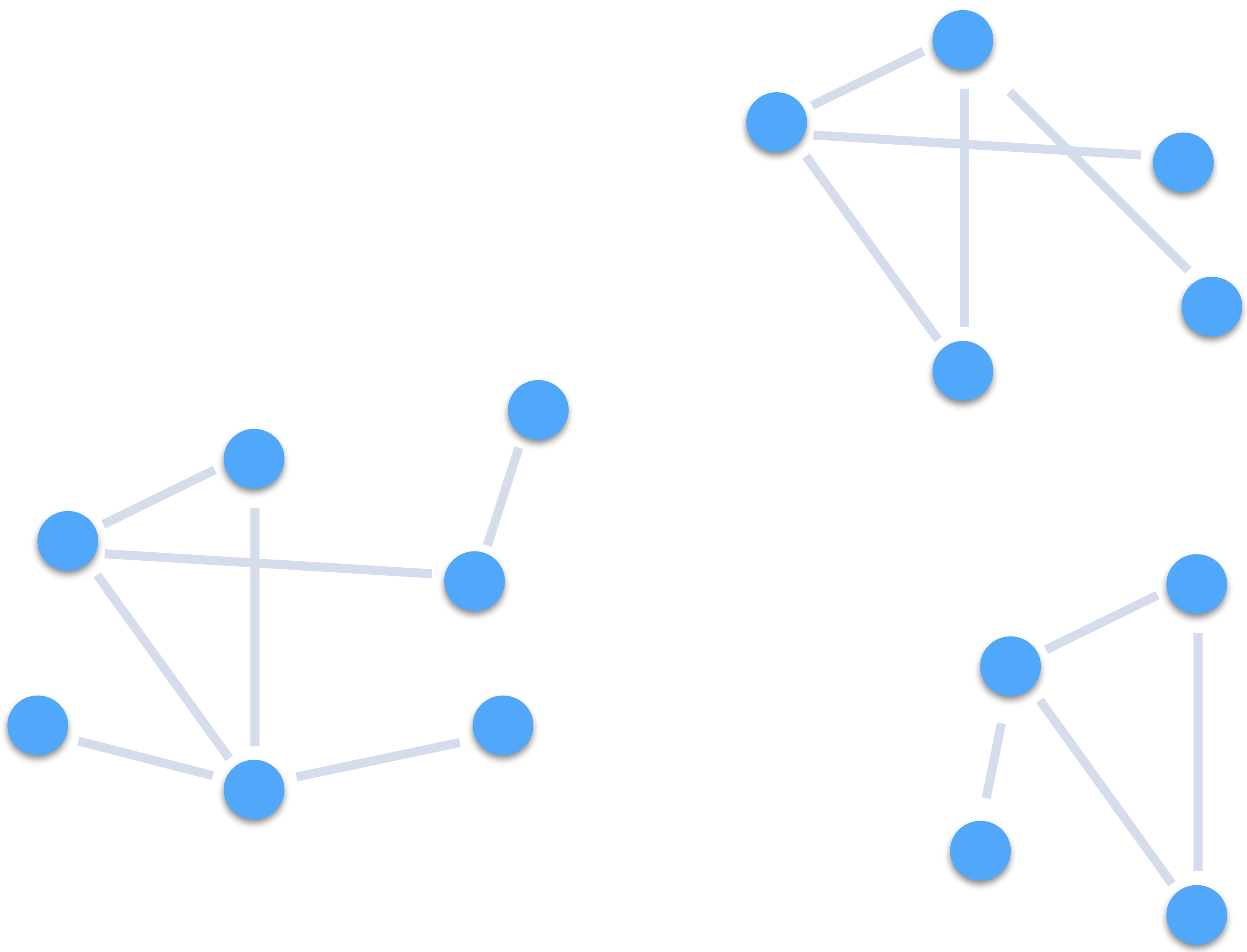
#Rounds: Comparing different algorithms



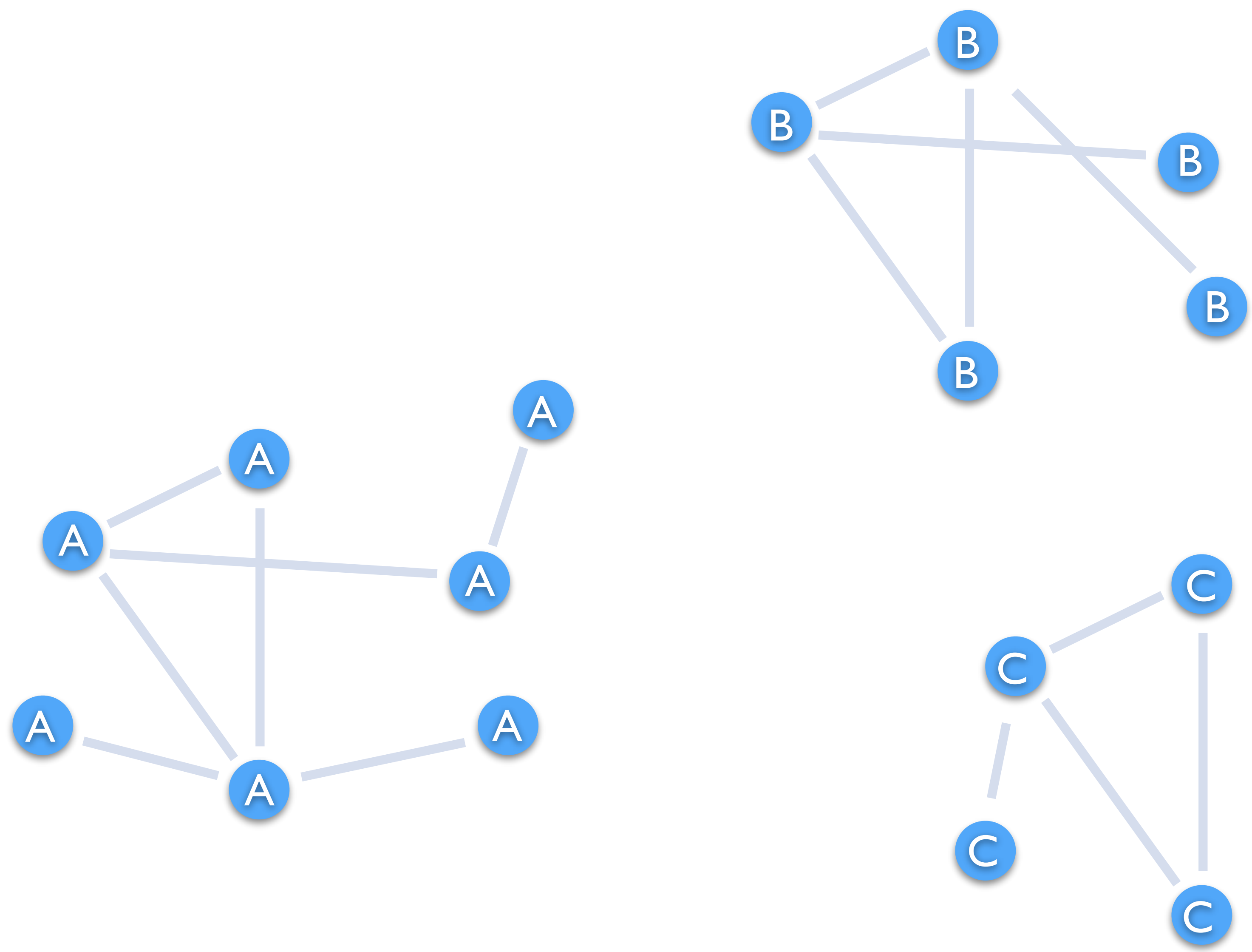
Comparison with Pregel



Warm-up: # connected components

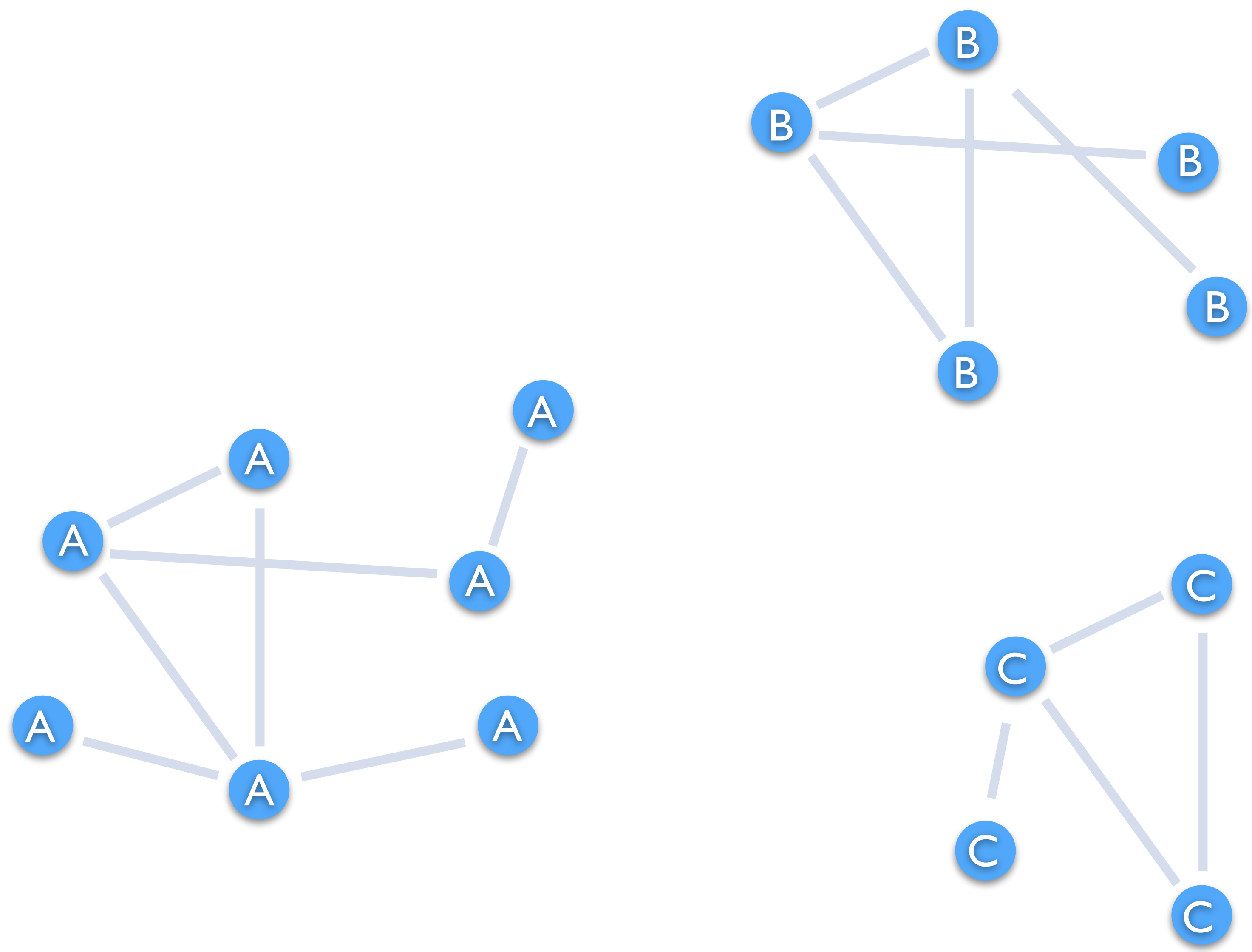


Warm-up: # connected components



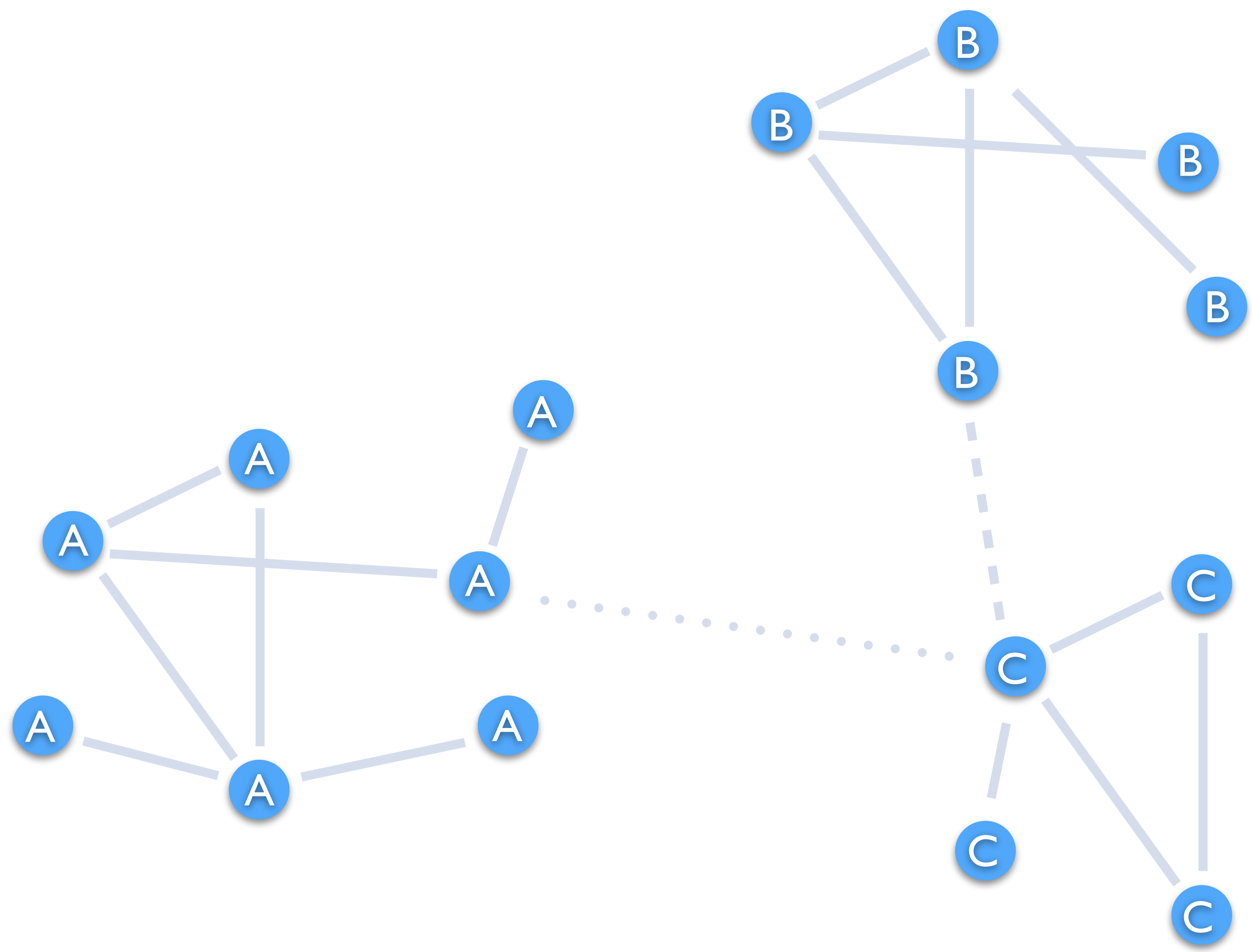
We can compute the components and assign to each component an id.

Warm-up: # connected components



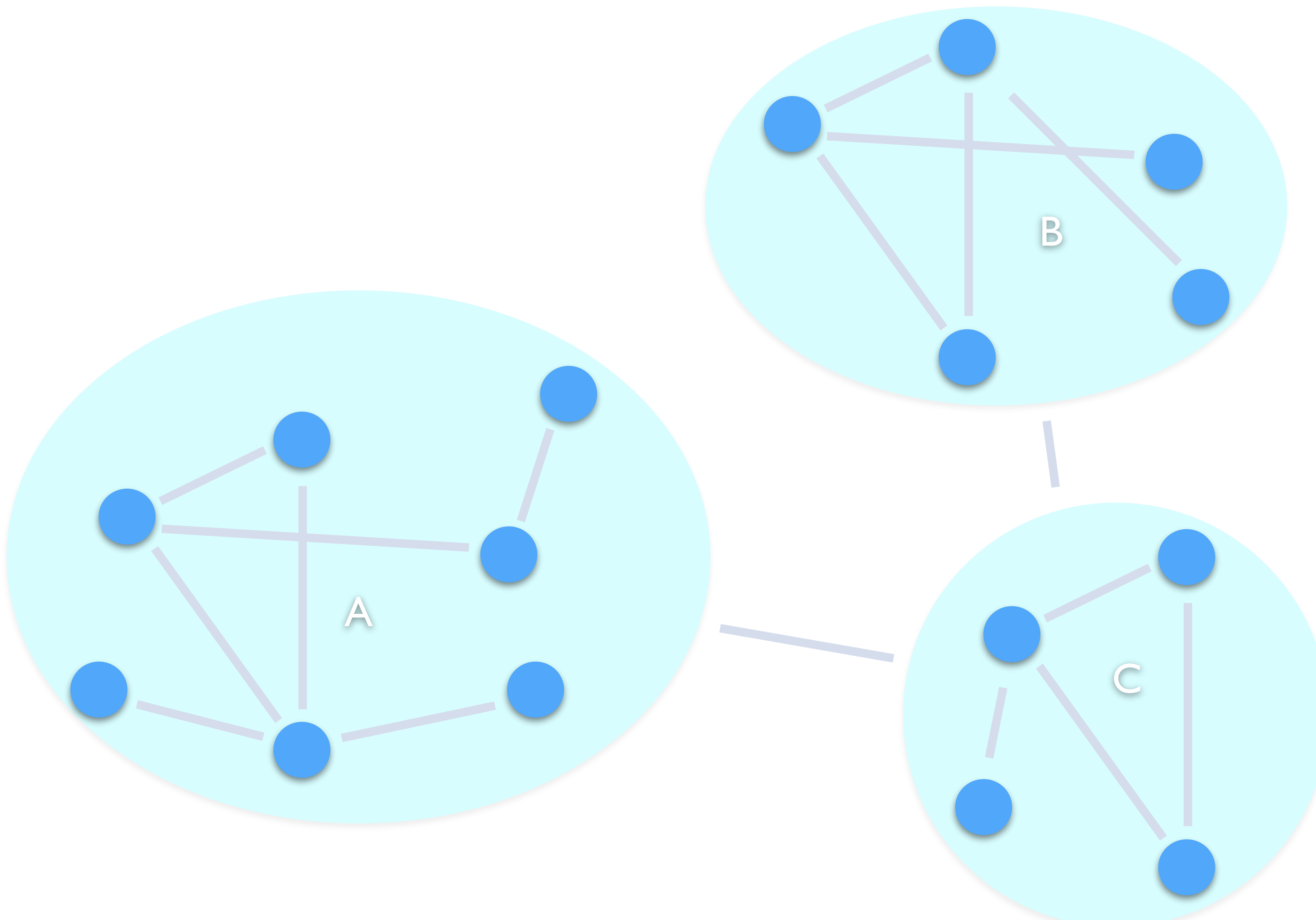
After adding private edges it is possible to recompute it by counting # newly connected components

Warm-up: # connected components



After adding private edges it is possible to recompute it by counting # newly connected components

Warm-up: # connected components



After adding private edges it is possible to recompute it by counting # newly connected components