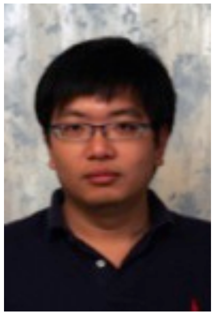
PowerGraph

Distributed Graph-Parallel Computation on Natural Graphs

Joseph Gonzalez

Joint work with:



Yucheng
Low

Haijie
Gu

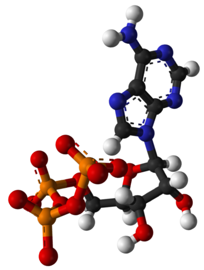
Danny
Bickson

Carlos
Guestrin

Carnegie Mellon University

Graphs *are ubiquitous..*

Social Media **Science** **Advertising** **Web**



- **Graphs** *encode* **relationships** between:

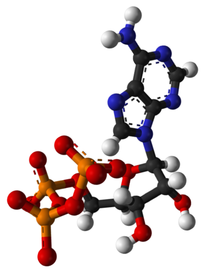
People Products Ideas

Facts Interests

- **Big: billions** of **vertices** and **edges** and rich metadata

Graphs are Essential to **Data-Mining** and **Machine Learning**

- Identify influential people and information
- Find communities
- Target ads and products
- Model complex data dependencies



Natural Graphs

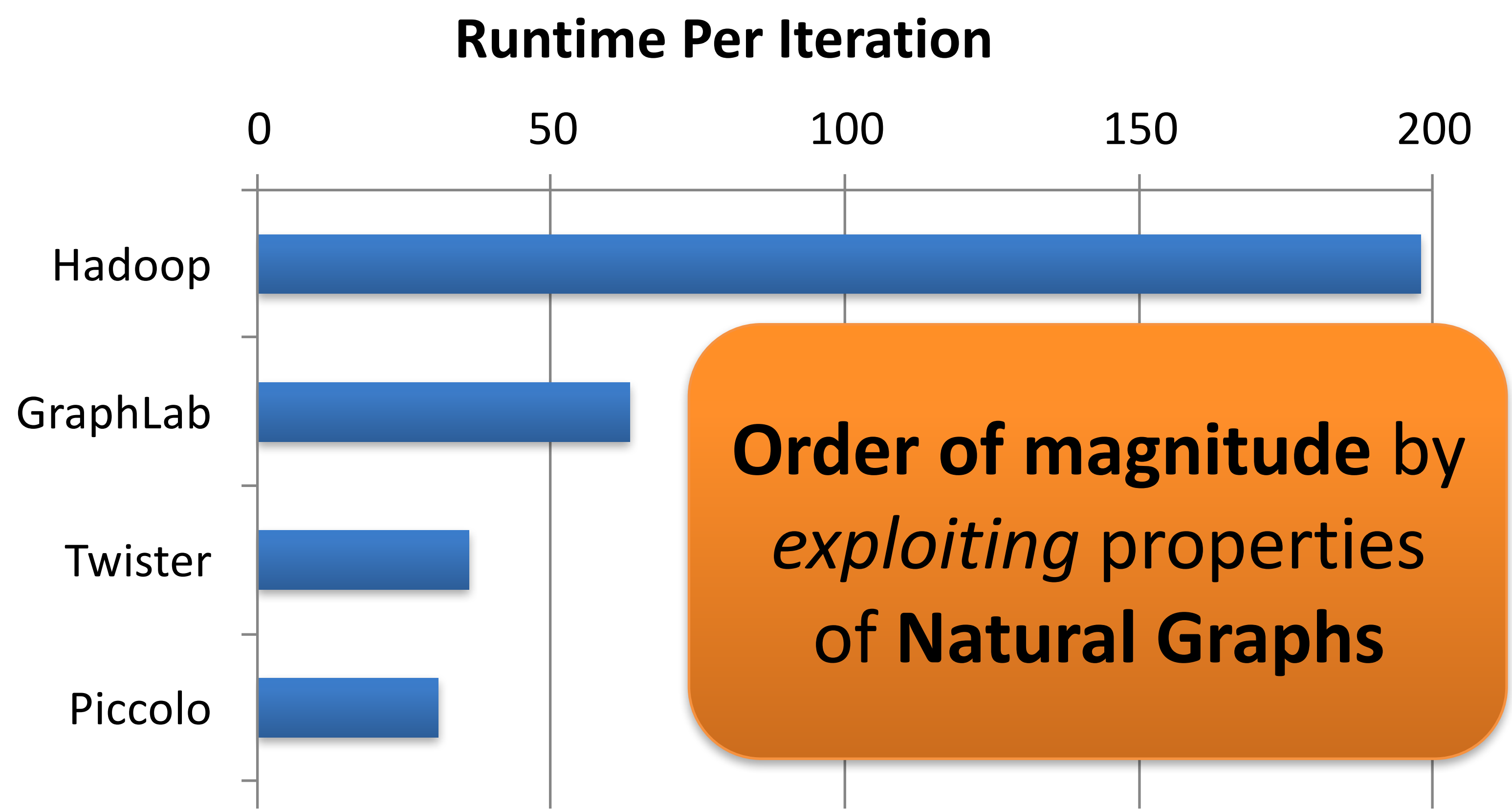
Graphs derived from natural phenomena

Problem:

Existing ***distributed*** graph computation systems perform poorly on **Natural Graphs**.

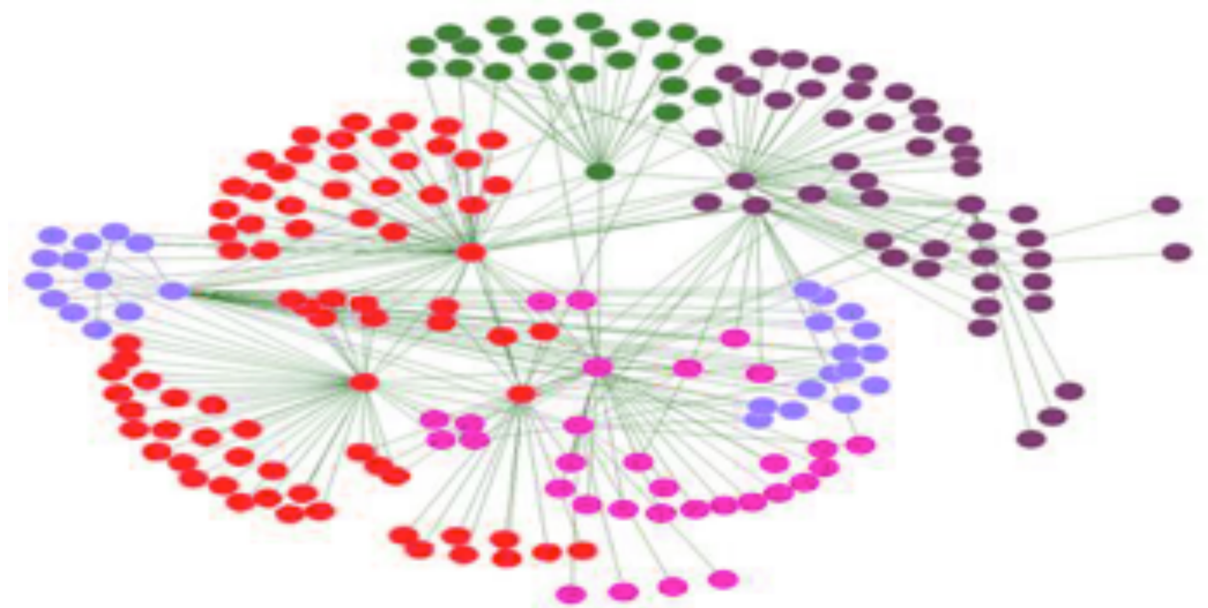
PageRank on Twitter Follower Graph

Natural Graph with 40M Users, 1.4 Billion Links



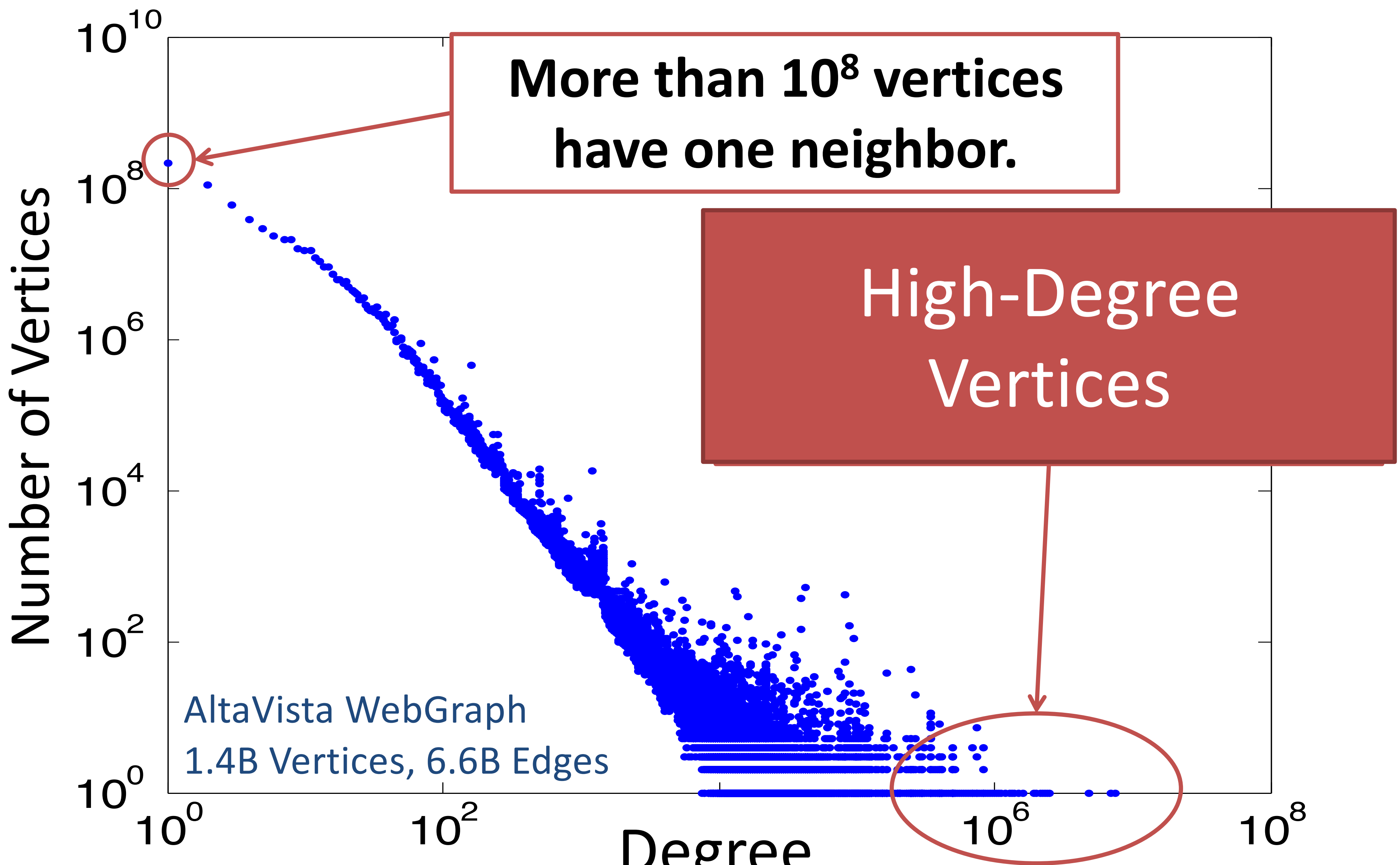
Hadoop results from [Kang et al. '11]
Twister (in-memory MapReduce) [Ekanayake et al. '10]

Properties of Natural Graphs



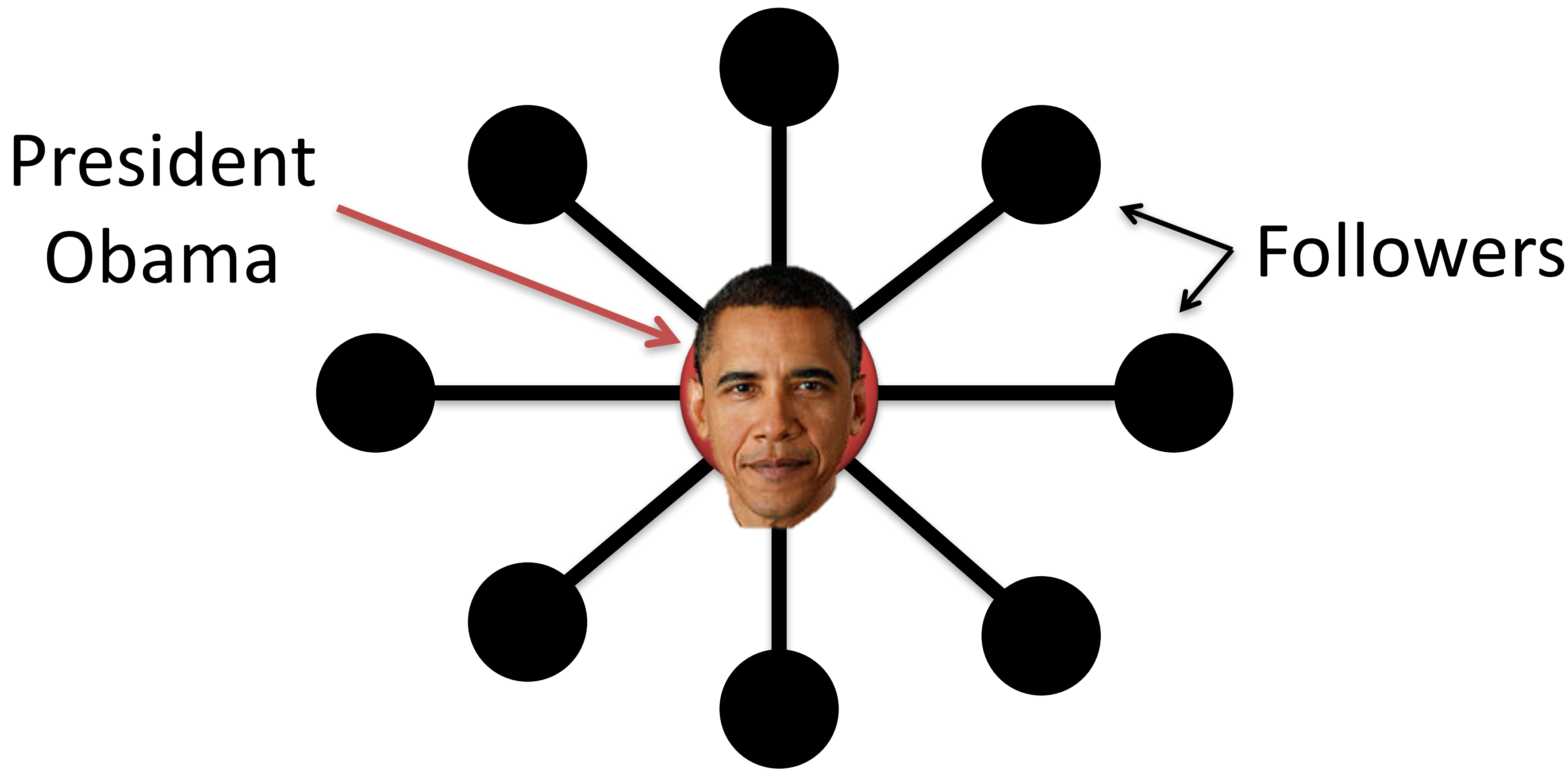
Power-Law Degree Distribution

Power-Law Degree Distribution

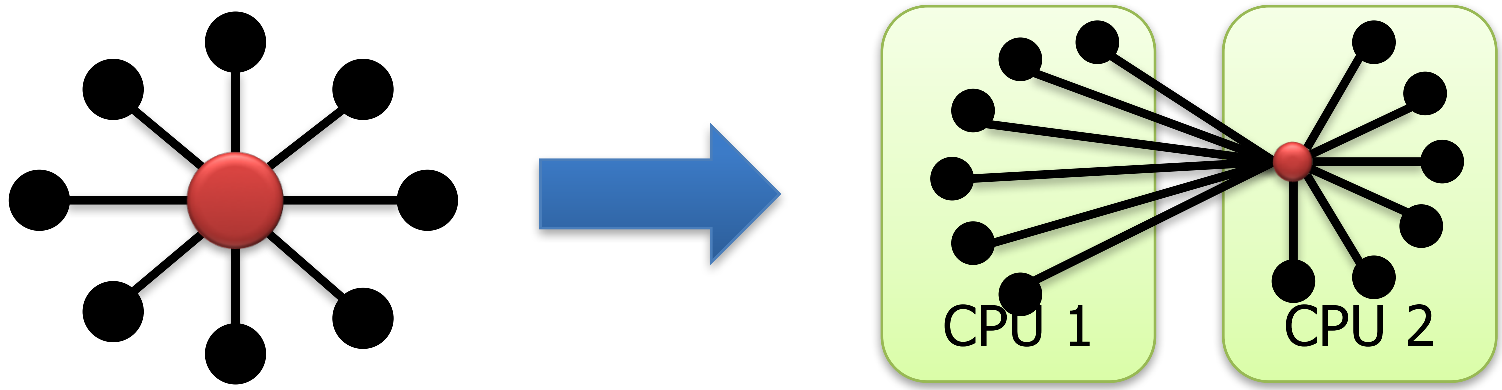


Power-Law Degree Distribution

"**Star Like**" Motif

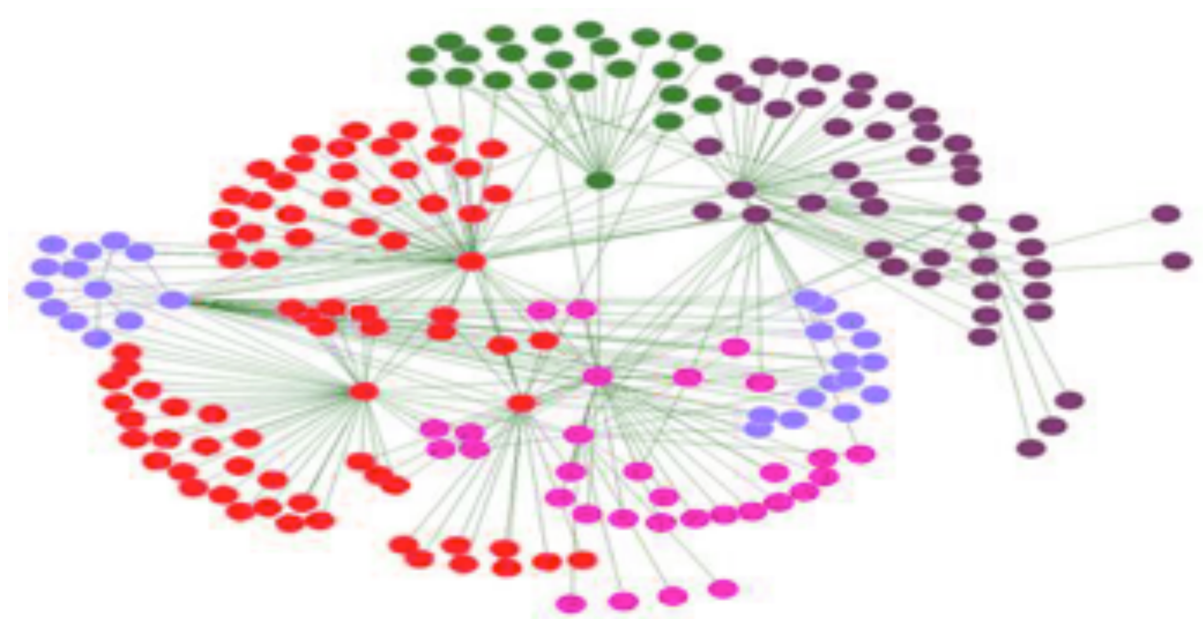


Power-Law Graphs are **Difficult to Partition**



- Power-Law graphs do not have **low-cost** balanced cuts *[Leskovec et al. 08, Lang 04]*
- Traditional graph-partitioning algorithms perform poorly on Power-Law Graphs.
  [Abou-Rjeili et al. 06]

Properties of Natural Graphs



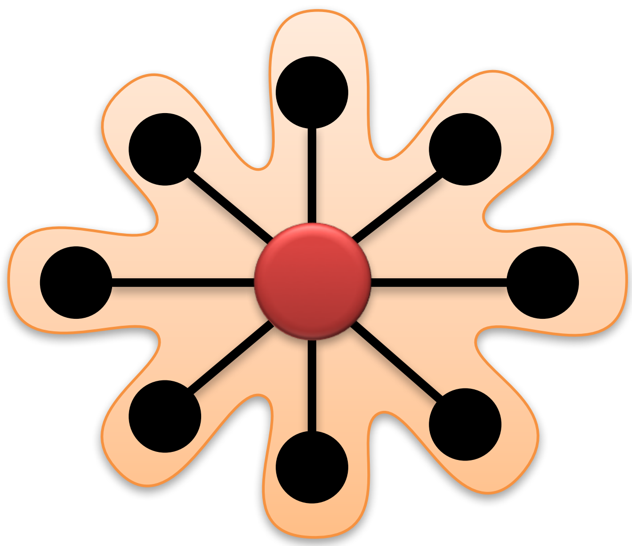
High-degree Vertices

Power-Law Degree Distribution

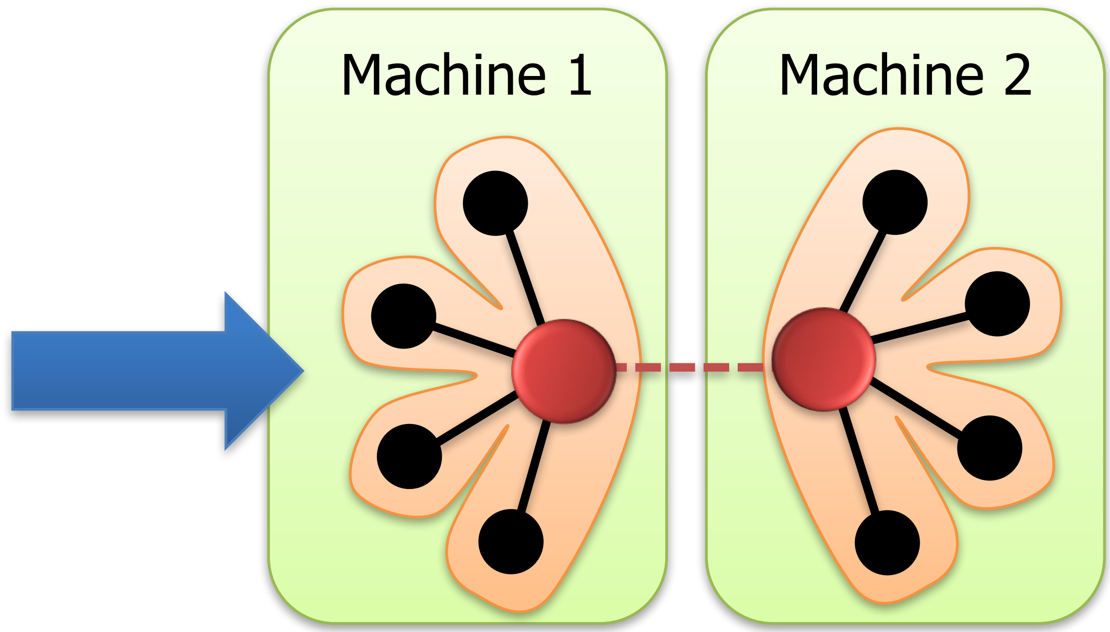
Low Quality Partition

PowerGraph

Program For This



Run on This



- Split **High-Degree** vertices
- **New Abstraction → *<u>Equivalence</u>*** *on Split Vertices*

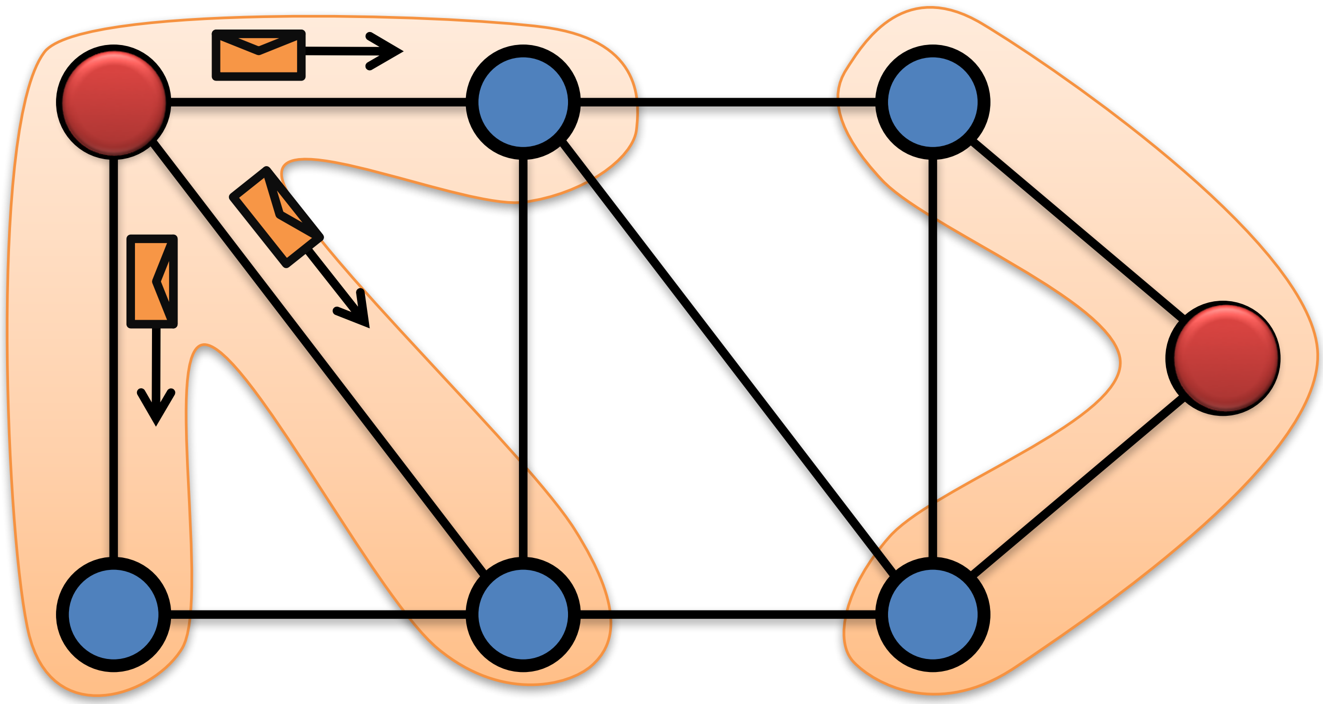
How do we *program* **graph** computation?

“Think like a Vertex.”

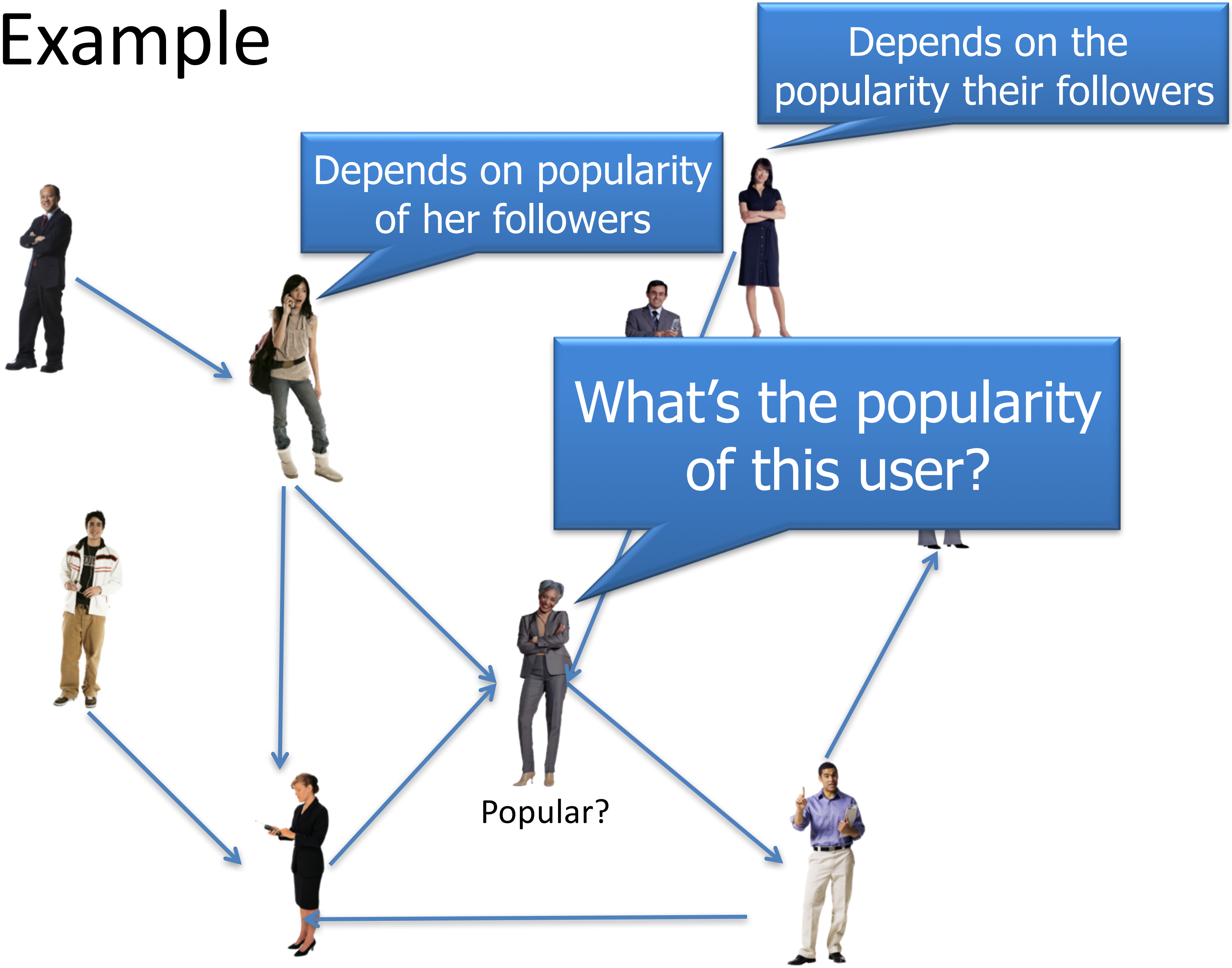
-Malewicz et al. [SIGMOD’10]

The **Graph-Parallel** Abstraction

- A user-defined **Vertex-Program** runs on each vertex
- **Graph** constrains **interaction** along edges
  - Using **messages** (e.g. **Pregel** [PODC'09, SIGMOD'10])
  - Through **shared state** (e.g., **GraphLab** [UAI'10, VLDB'12])
- **Parallelism**: run multiple vertex programs simultaneously



Example


Depends on the
popularity their followers
Depends on popularity
of her followers
What's the popularity
of this user?
Popular?

PageRank Algorithm

$$R[i] = 0.15 + \sum_{j \in \mathrm{Nbrs}(i)} w_{ji} R[j]$$

Rank of user *i*

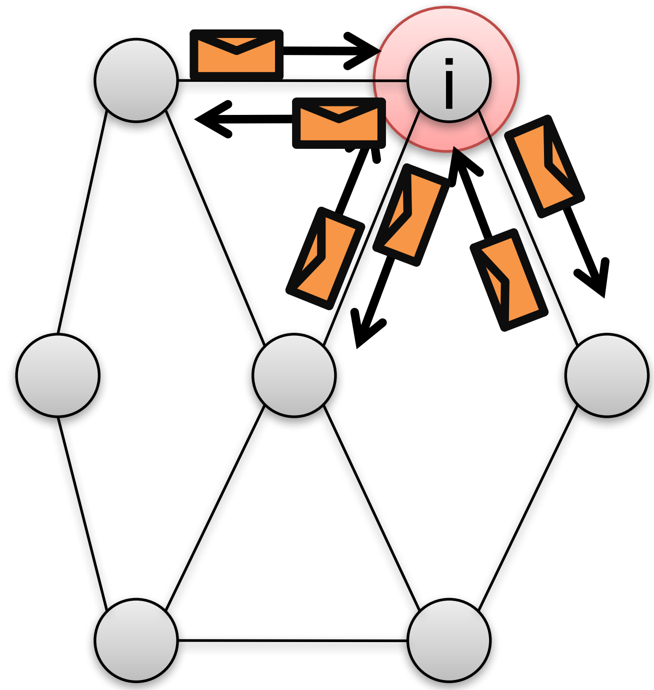
Weighted sum of neighbors' ranks

- Update ranks in parallel
- Iterate until convergence

The Pregel Abstraction

Vertex-Programs interact by sending **messages**.

```
Pregel_PageRank(i, messages) :
  // Receive all the messages
  total = 0
  foreach( msg in messages) :
    total = total + msg

  // Update the rank of this vertex
  R[i] = 0.15 + total

  // Send new messages to neighbors
  foreach(j in out_neighbors[i]) :
    Send  msg(R[i] * wij) to vertex j
```

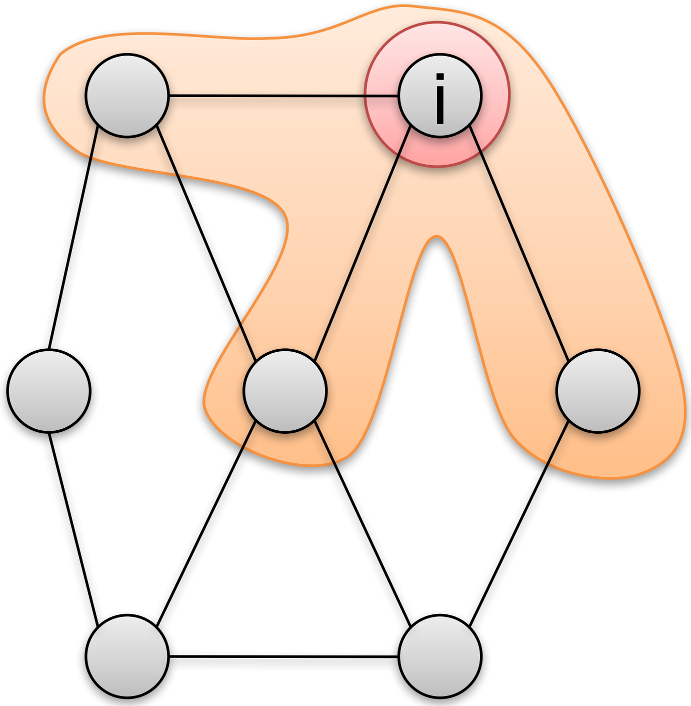


Malewicz et al. [PODC'09, SIGMOD'10]

The GraphLab Abstraction

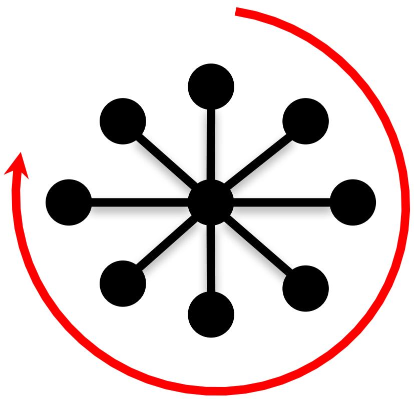
Vertex-Programs directly **read** the neighbors state

```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in in_neighbors(i)):
    total = total + R[j] * w_ji

  // Update the PageRank
  R[i] = 0.15 + total

  // Trigger neighbors to run again
  if R[i] not converged then
    foreach( j in out_neighbors(i)):
      signal vertex-program on j
```

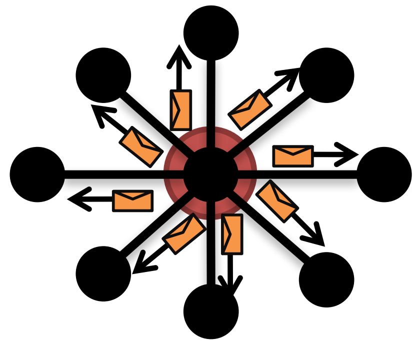


Low et al. [UAI'10, VLDB'12]

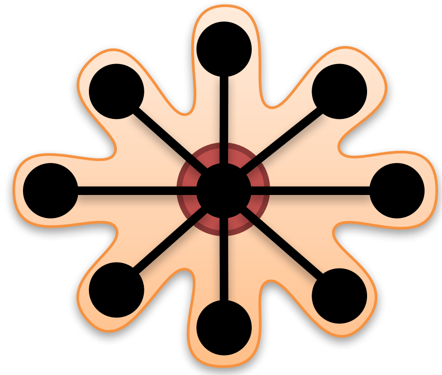
Challenges of **High-Degree** Vertices



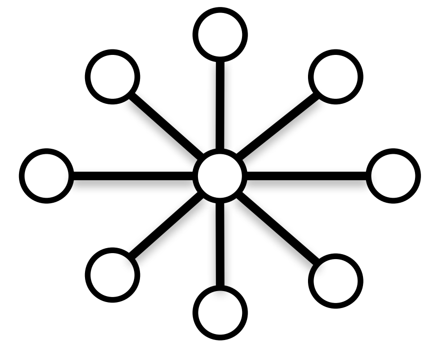
Sequentially process edges



Sends many messages (Pregel)

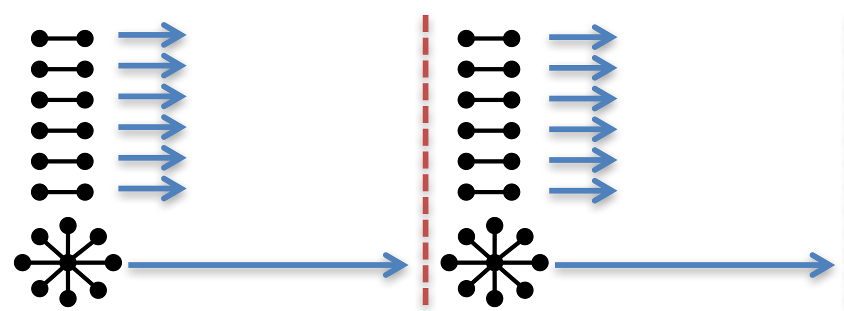


Touches a large fraction of graph (GraphLab)



Edge meta-data too large for single machine

Asynchronous Execution requires heavy locking (GraphLab)

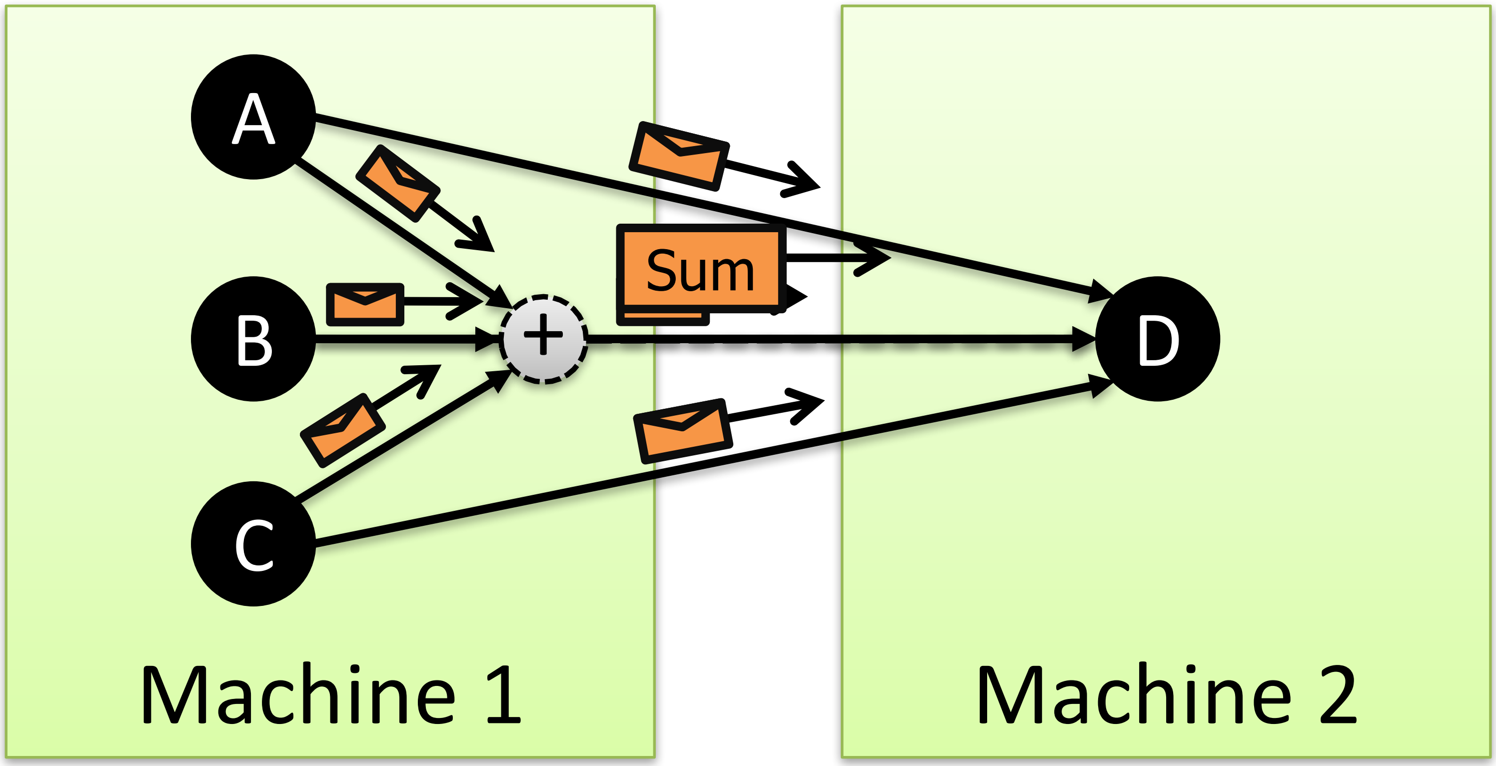


Synchronous Execution prone to stragglers (Pregel)

Communication Overhead for High-Degree Vertices

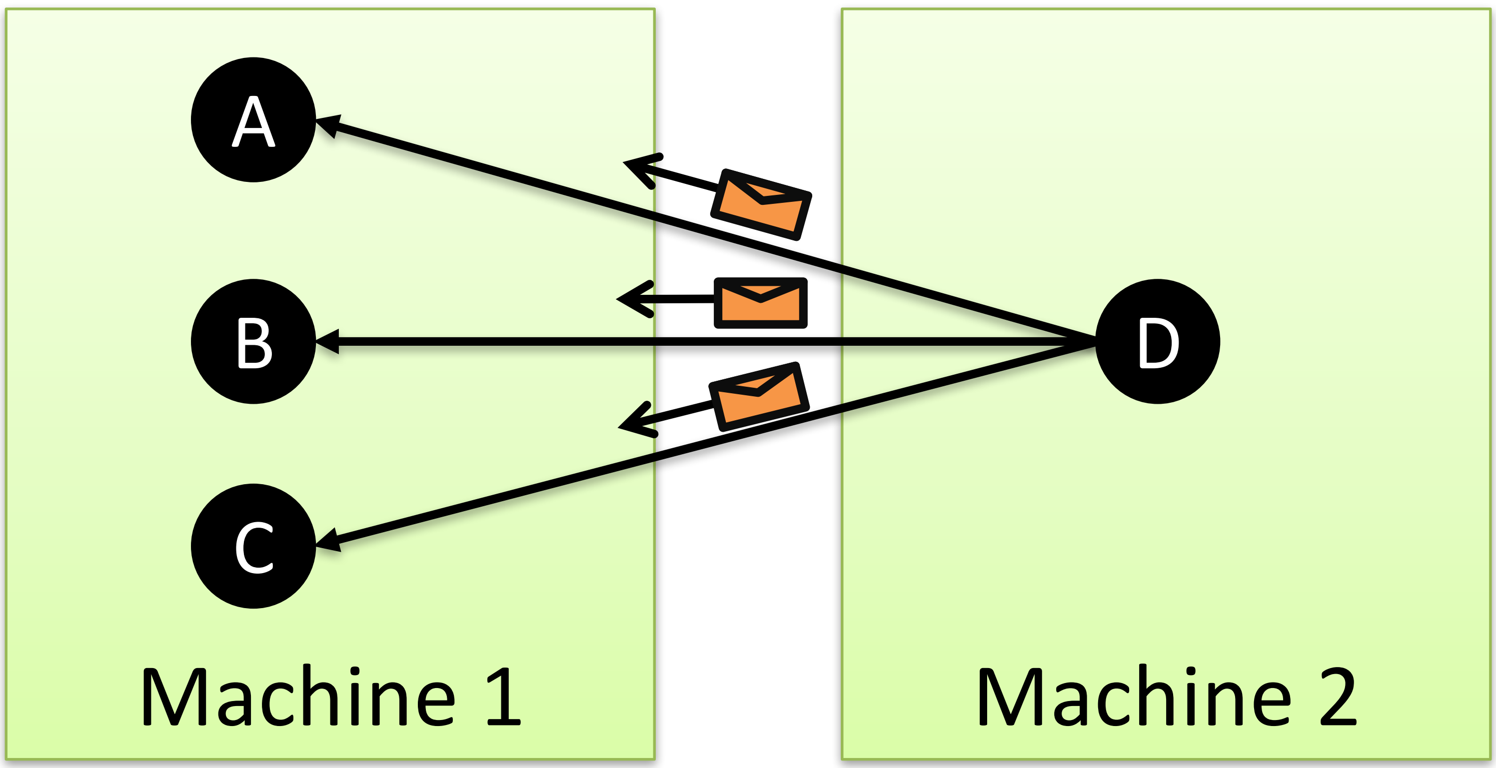
Fan-In vs. Fan-Out

Pregel **Message Combiners** on **Fan-In**



- User defined **commutative associative** (+) message operation:

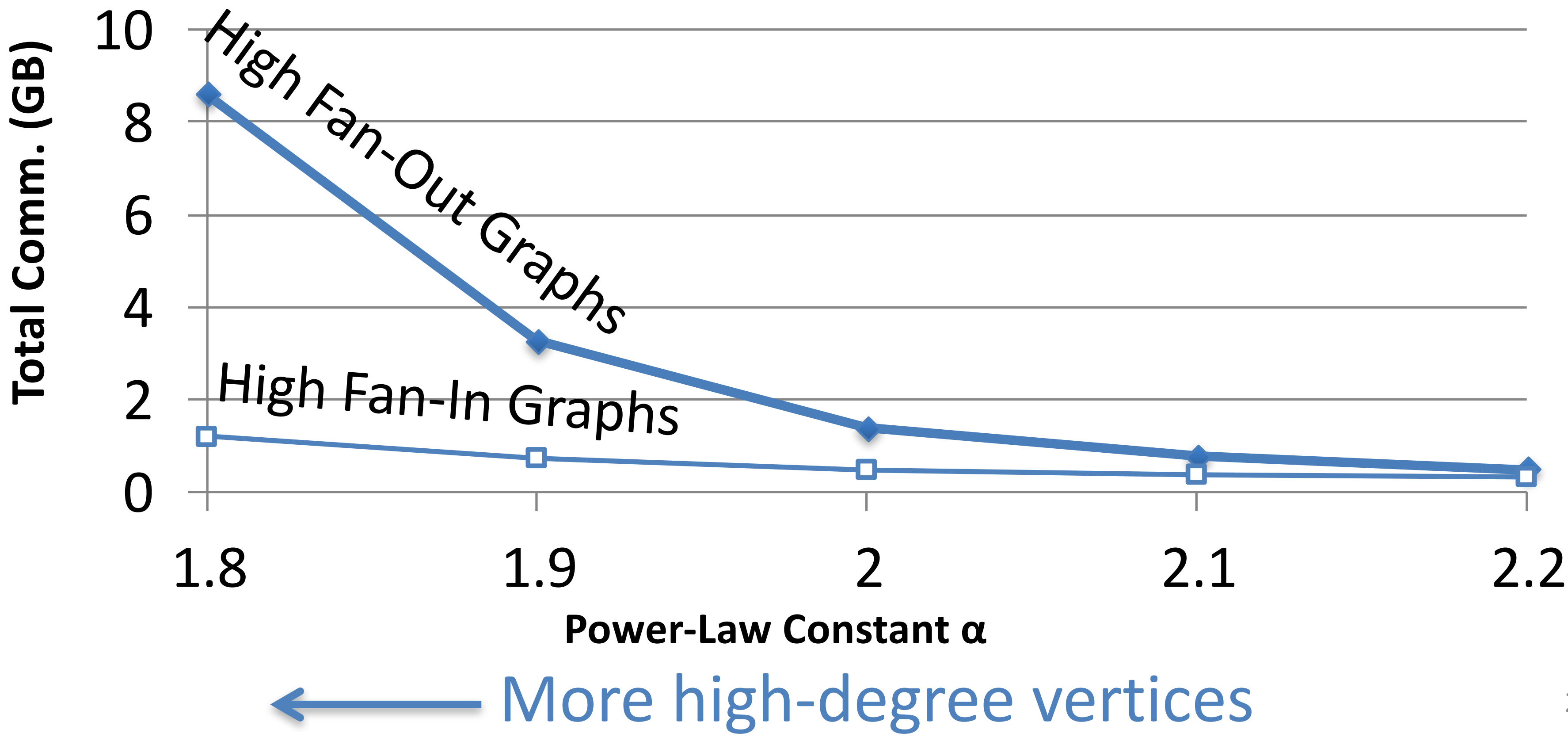
Pregel Struggles with **Fan-Out**



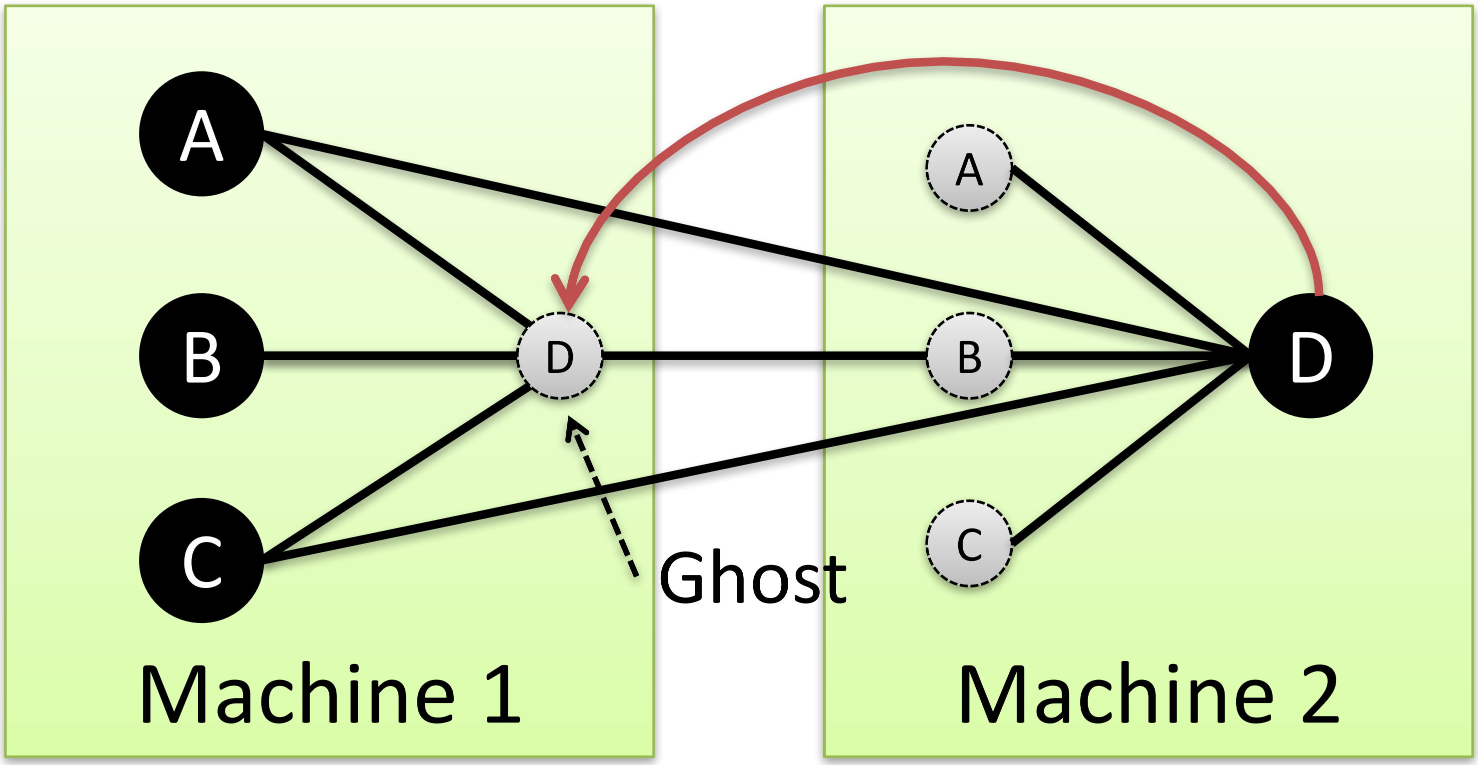
- **Broadcast** sends many copies of the same message to the same machine!

Fan-In and Fan-Out Performance

- PageRank on synthetic Power-Law Graphs
  - Piccolo was used to simulate Pregel with combiners

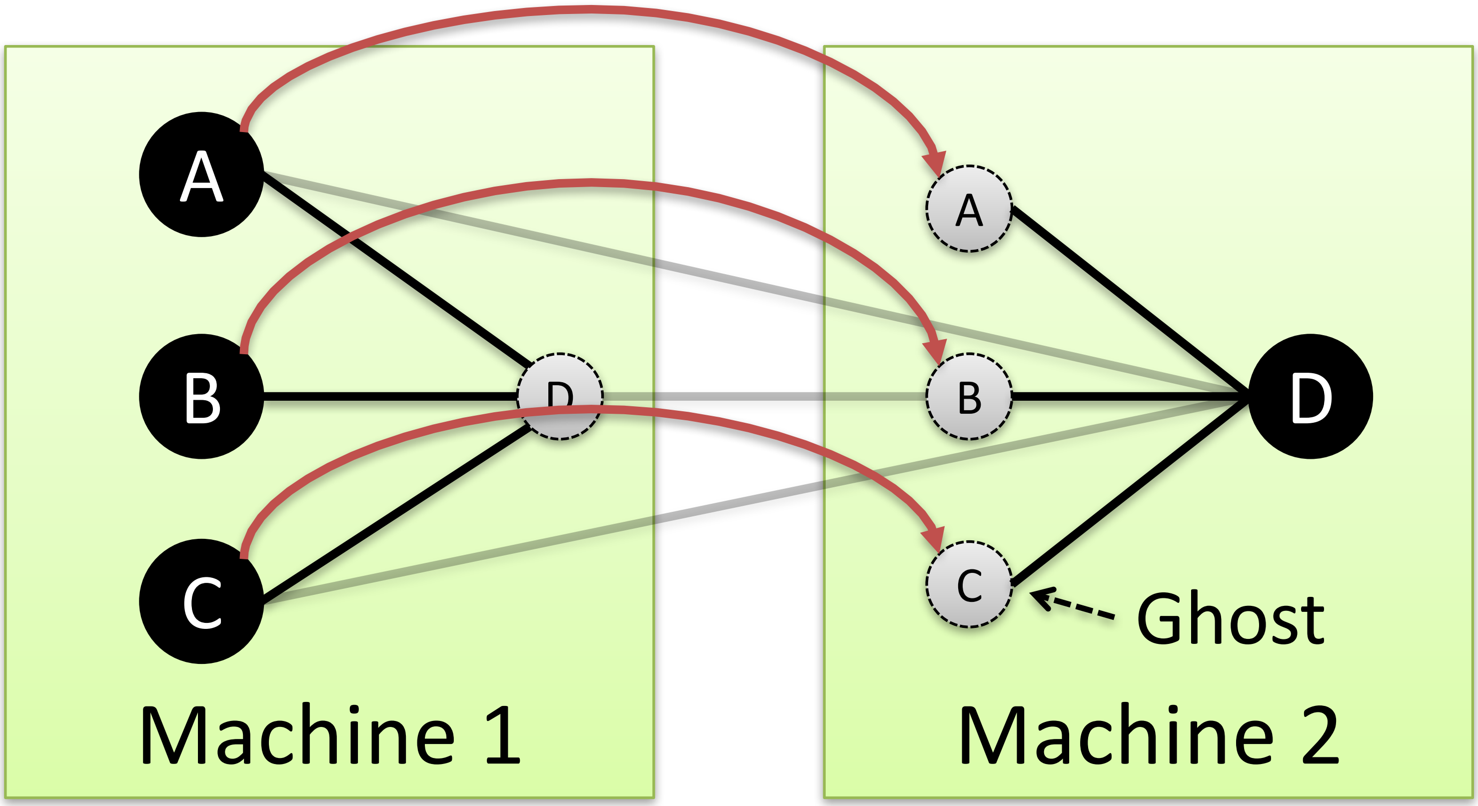


GraphLab Ghosting



- Changes to master are synced to ghosts

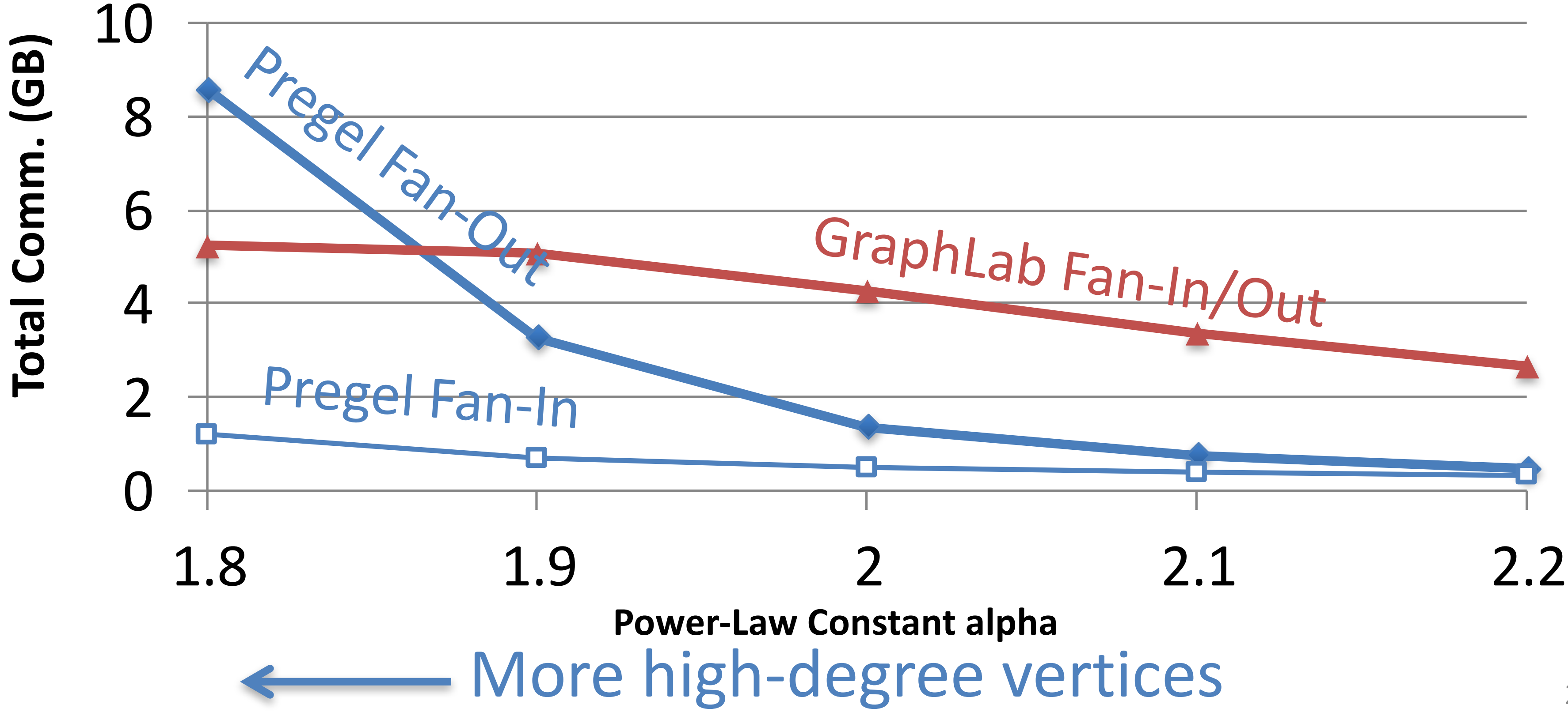
GraphLab Ghosting



- Changes to **neighbors** of **high degree vertices** creates substantial network traffic

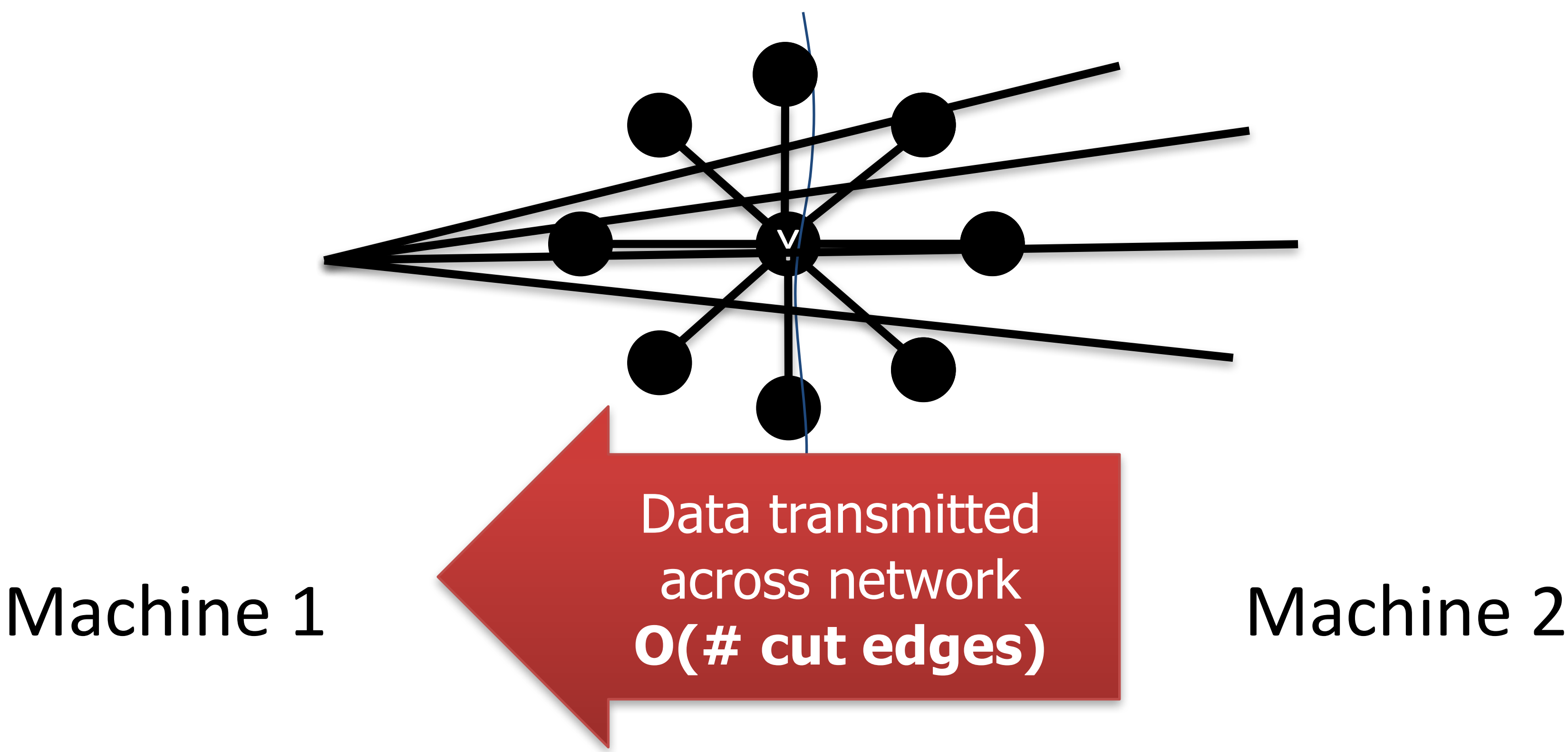
Fan-In and Fan-Out Performance

- PageRank on synthetic Power-Law Graphs
- GraphLab is **undirected**



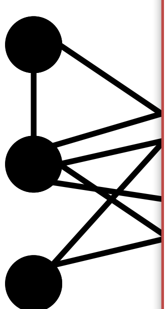
Graph Partitioning

- Graph parallel abstractions rely on partitioning:
  - Minimize communication
  - Balance computation and storage



Random Partitioning

- Both GraphLab and Pregel resort to **random** (hashed) partitioning on **natural graphs**



$$\mathbb{E}\left[\frac{|Edges\ Cut|}{|E|}\right] = 1 - \frac{1}{p}$$

10 Machines → 90% of edges cut

100 Machines → 99% of edges cut!

In Summary

GraphLab and **Pregel** are not well suited for **natural graphs**

- Challenges of **high-degree vertices**
- Low quality **partitioning**

PowerGraph

- **GAS Decomposition**: distribute vertex-programs
  - Move computation to data
  - Parallelize **high-degree** vertices
- **Vertex Partitioning:**
  - Effectively distribute large power-law graphs

A **Common Pattern** for Vertex-Programs

```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in in_neighbors(i)):
    total = total + R[j] * w_ji

  // Update the PageRank
  R[i] = 0.1 + total

  // Trigger neighbors to run again
  if R[i] not converged then
    foreach( j in out_neighbors(i))
      signal vertex-program on j
```

Gather Information About Neighborhood

Update Vertex

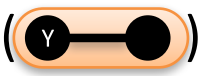
Signal Neighbors & Modify Edge Data

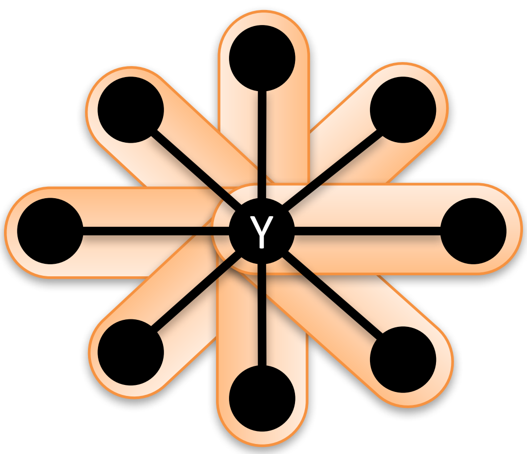
GAS Decomposition

Gather (Reduce)

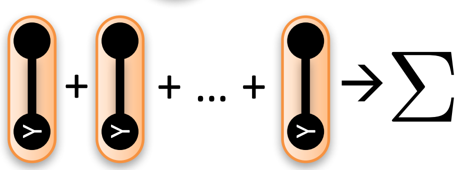
Accumulate information about neighborhood

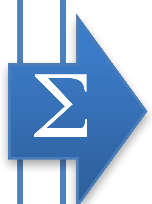
User Defined:

- **Gather**() → Σ
- $\Sigma_1 \oplus \Sigma_2 \rightarrow \Sigma_3$



Parallel Sum

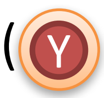


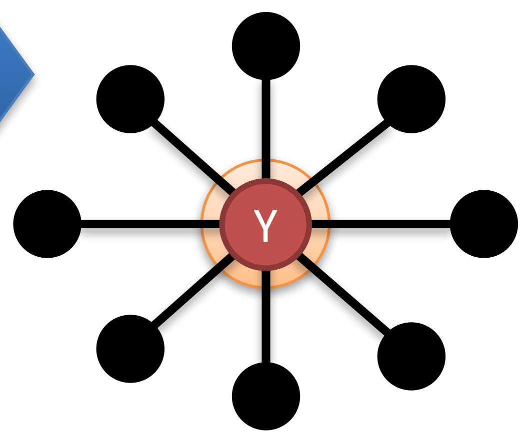


Apply

Apply the accumulated value to center vertex

User Defined:

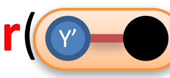
- **Apply**( , Σ) → 

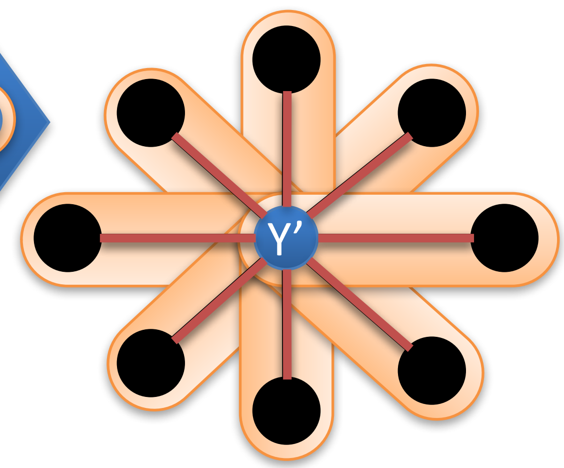


Scatter

Update adjacent edges and vertices.

User Defined:

- **Scatter**() →



Update Edge Data & Activate Neighbors

PageRank in PowerGraph

$$R[i] = 0.15 + \boxed{\sum_{j \in \mathrm{Nbrs}(i)} w_{ji} R[j]}$$

PowerGraph_PageRank(i)

Gather(j → i) : return w_{ji} * R[j]
sum(a, b) : return a + b;

Apply(i, Σ) : R[i] = 0.15 + Σ

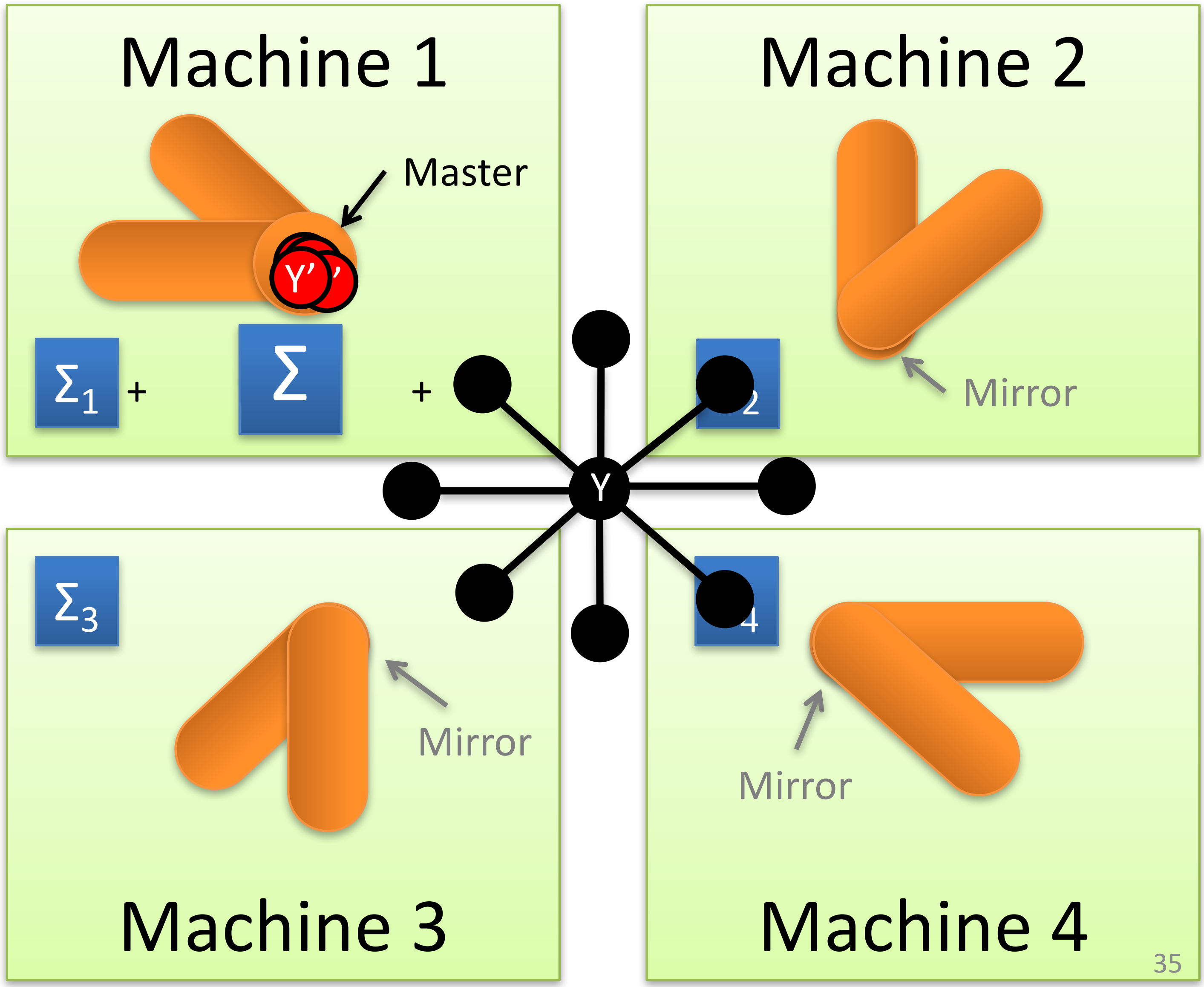
Scatter(i → j) :
if *R[i]* changed then trigger *j* to be **recomputed**

Distributed Execution of a PowerGraph Vertex-Program

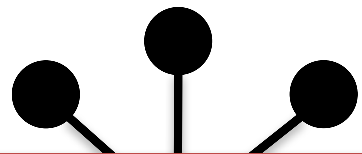
Gather

Apply

Scatter



Minimizing Communication in PowerGraph



Communication is linear in the number of machines each vertex spans

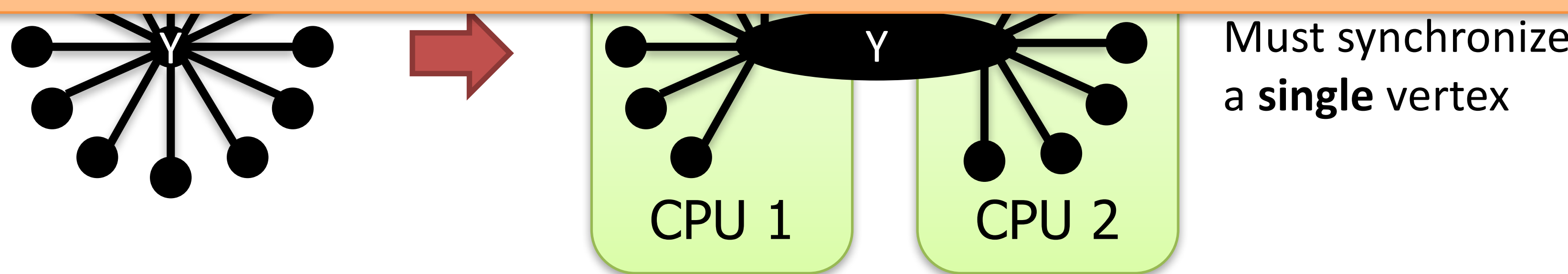
A **vertex-cut** minimizes machines each vertex spans

*Percolation theory suggests that power law graphs have **good vertex cuts**. [Albert et al. 2000]*

New Approach to Partitioning

- Rather than cut edges:

New Theorem:
*For **any edge-cut** we can directly construct a vertex-cut which requires **strictly less** communication and storage.*



Constructing Vertex-Cuts

- **Evenly** assign **edges** to machines
  - Minimize machines spanned by each vertex
- Assign each edge **as it is loaded**
  - Touch each edge only once
- Propose three **distributed** approaches:
  - ***Random*** *Edge Placement*
  - ***Coordinated Greedy*** *Edge Placement*
  - ***Oblivious Greedy*** *Edge Placement*

Random Edge-Placement

- Randomly assign edges to machines

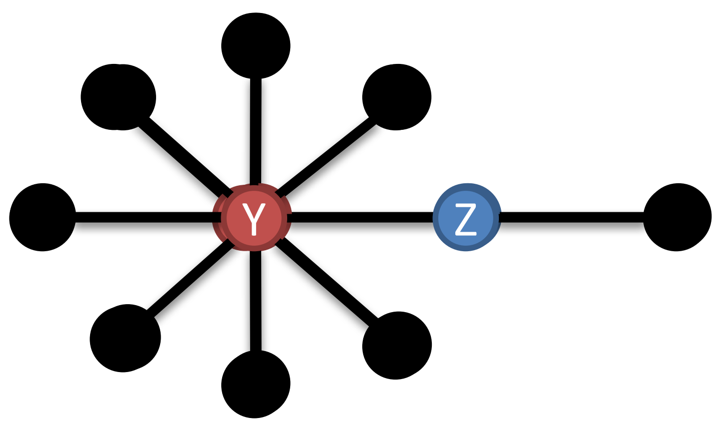
Machine 1	Machine 2	Machine 3

Balanced Vertex-Cut

Y Spans 3 Machines

Z Spans 2 Machines

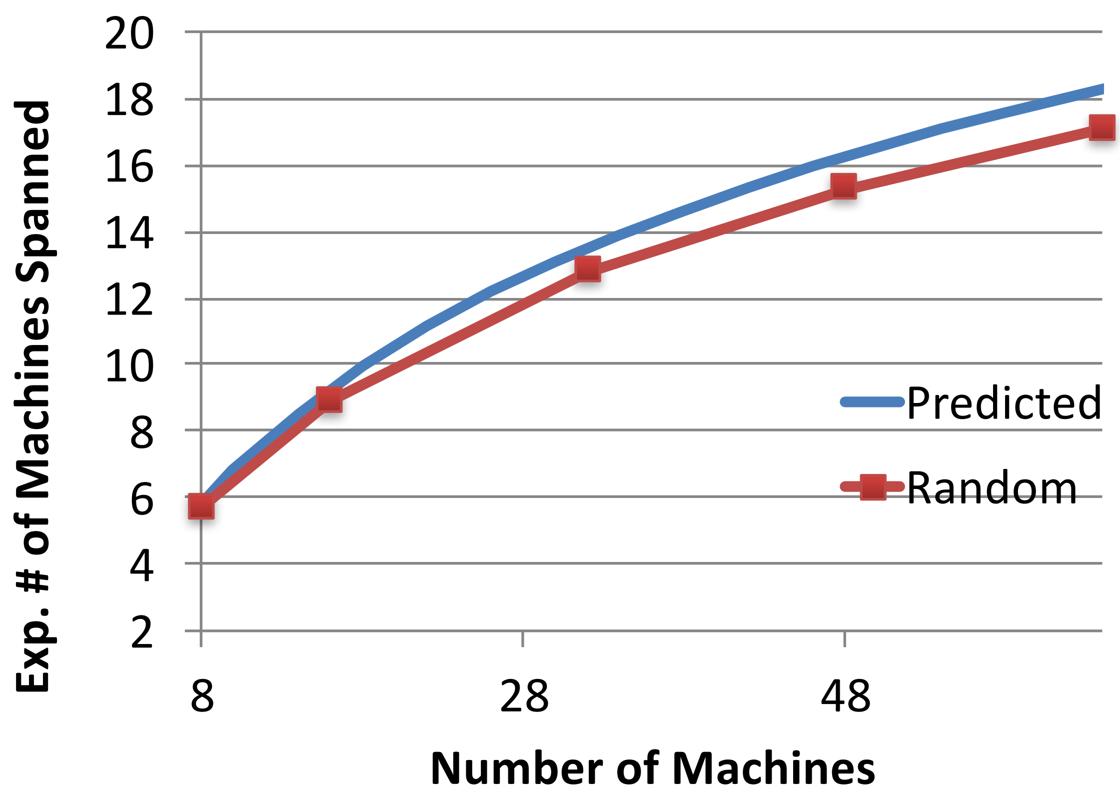
Not cut!



Analysis Random Edge-Placement

- Expected number of machines spanned by a vertex:

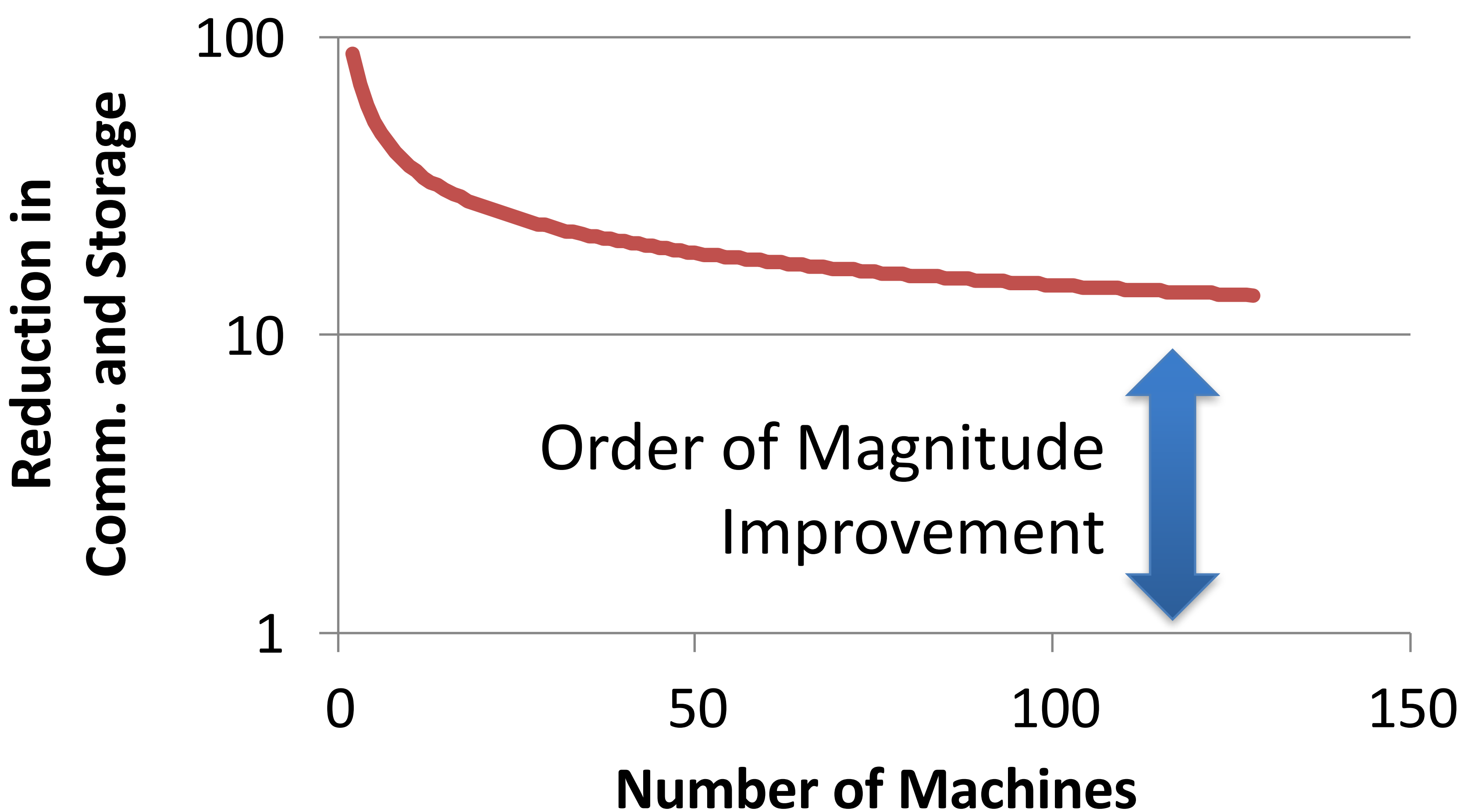
Twitter Follower Graph
41 Million Vertices
1.4 Billion Edges



Accurately Estimate Memory and Comm. Overhead

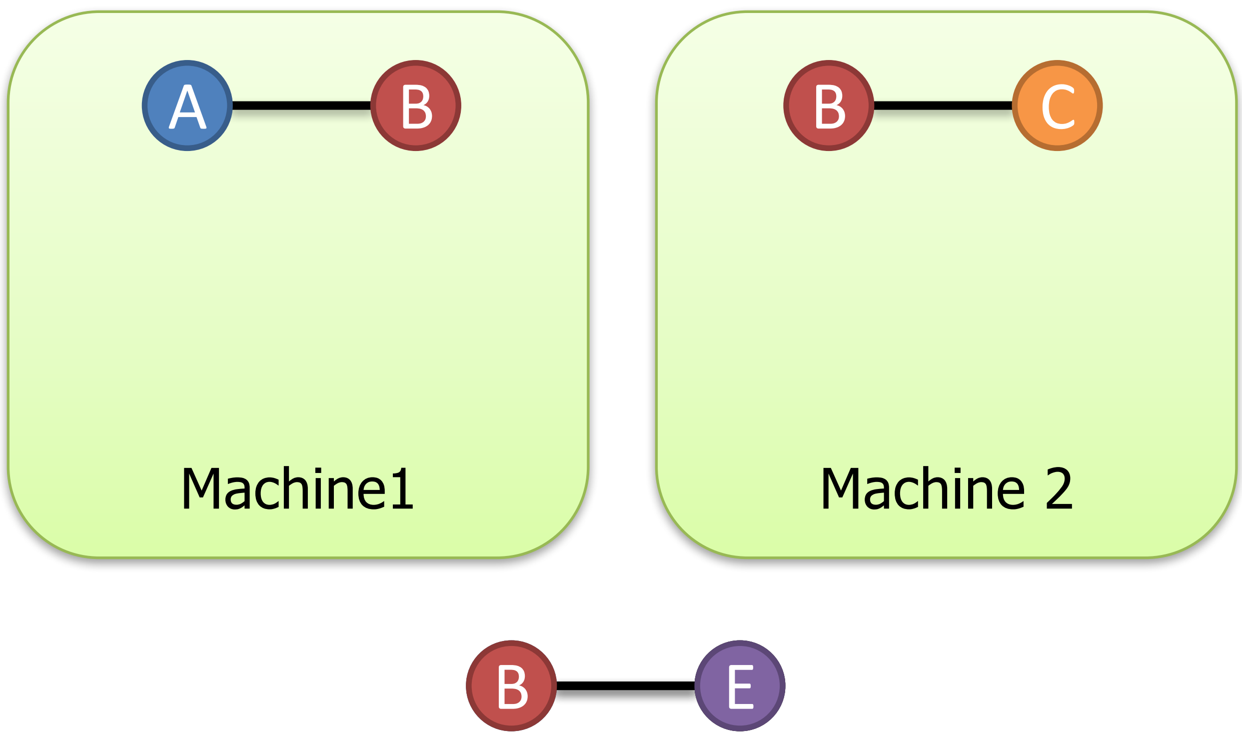
Random Vertex-Cuts vs. Edge-Cuts

- Expected improvement from vertex-cuts:



Greedy Vertex-Cuts

- Place edges on machines which already have the vertices in that edge.

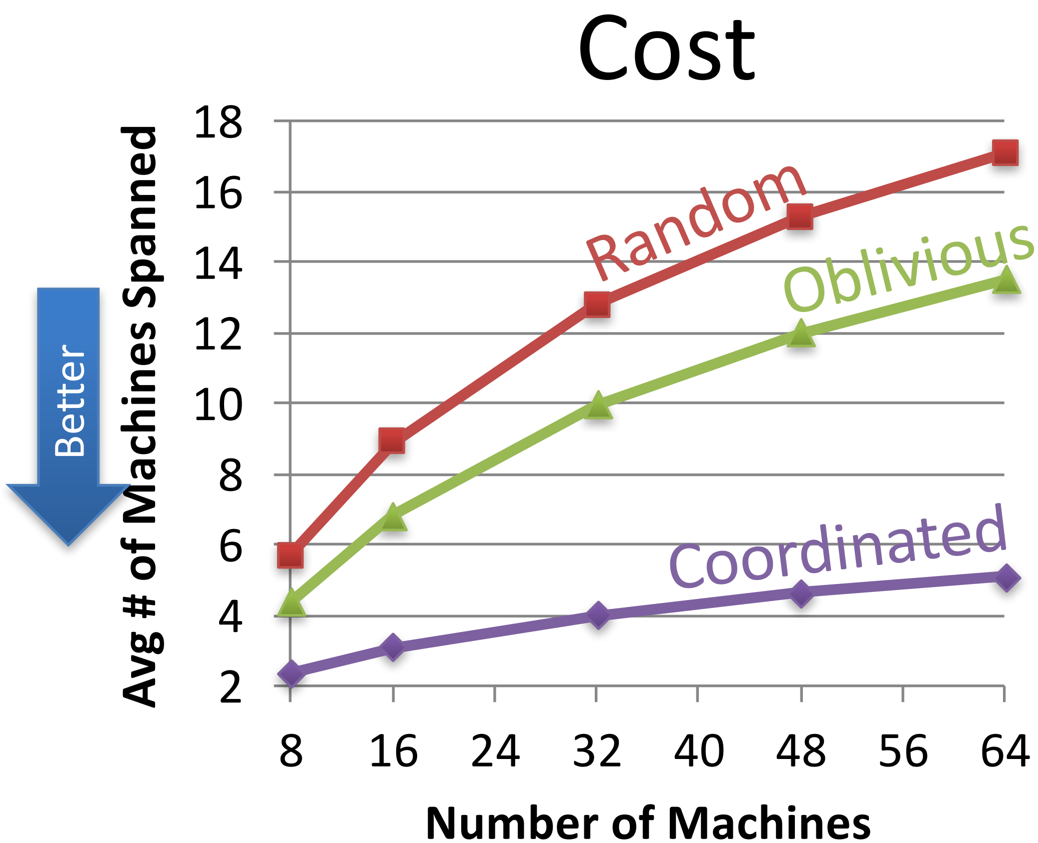


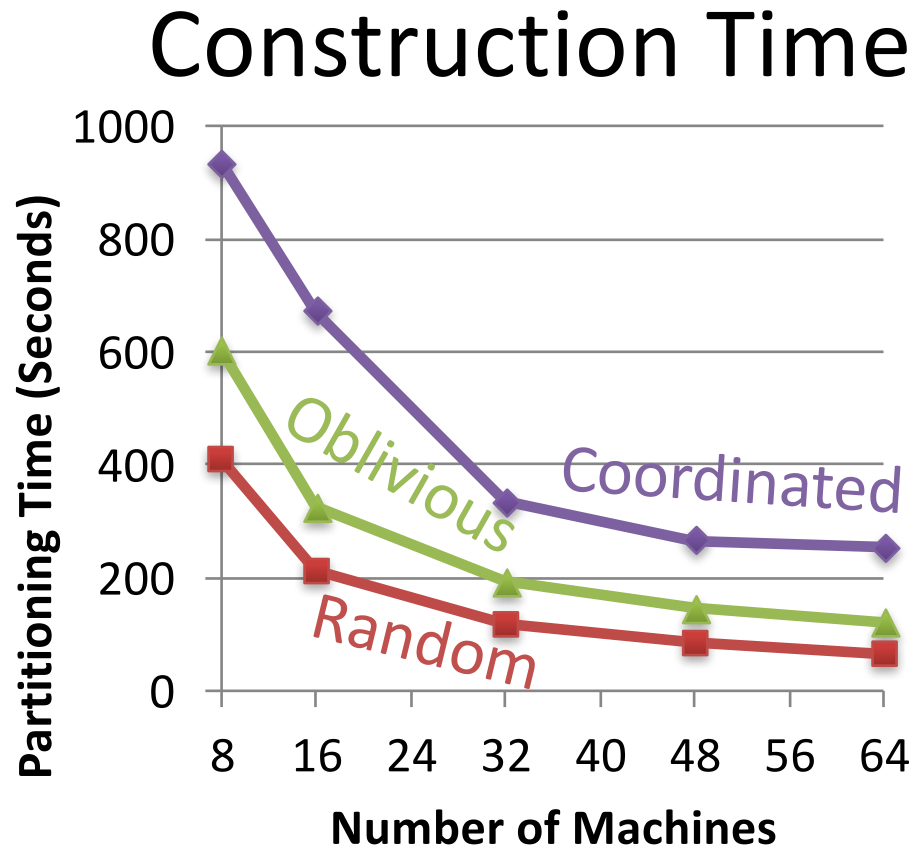
Greedy Vertex-Cuts

- **De-randomization** → greedily minimizes the expected number of machines spanned
- **Coordinated** Edge Placement
  - Requires coordination to place each edge
  - Slower: higher quality cuts
- **Oblivious** Edge Placement
  - Approx. greedy objective without coordination
  - Faster: lower quality cuts

Partitioning Performance

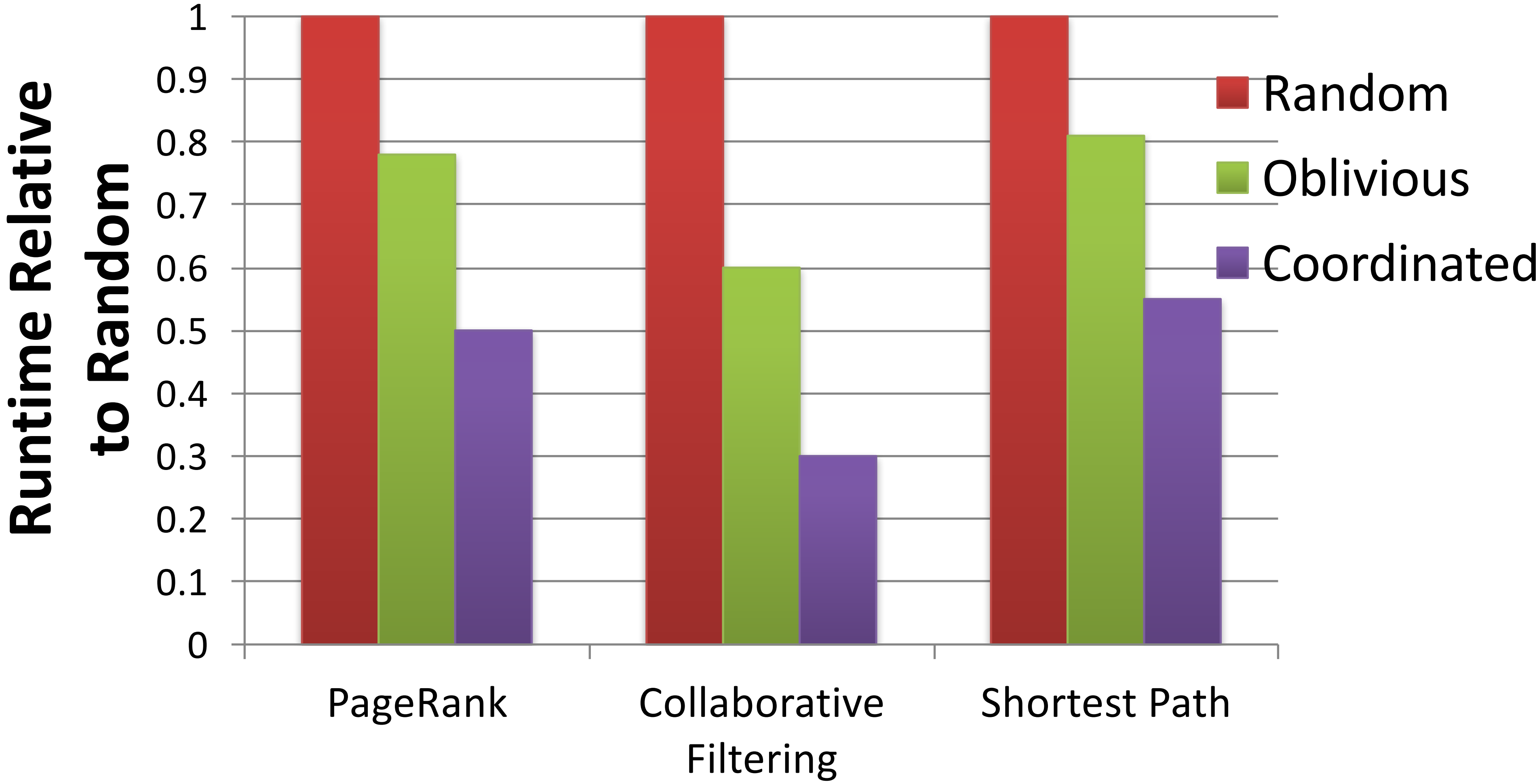
Twitter Graph: 41M vertices, 1.4B edges





Oblivious balances cost and partitioning time.

Greedy Vertex-Cuts Improve Performance



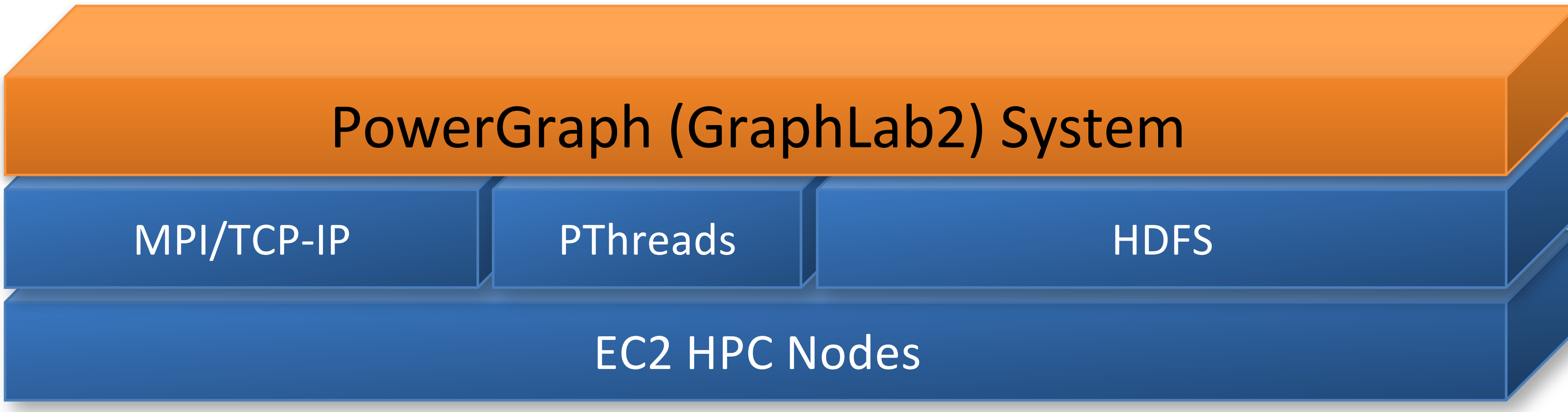
Greedy partitioning improves computation performance.

Other Features (See Paper)

- Supports three execution modes:
  - **Synchronous:** Bulk-Synchronous GAS Phases
  - **Asynchronous:** Interleave GAS Phases
  - **Asynchronous + Serializable:** Neighboring vertices do not run simultaneously
- Delta Caching
  - Accelerate gather phase by **caching** partial sums for each vertex

System Evaluation

System Design



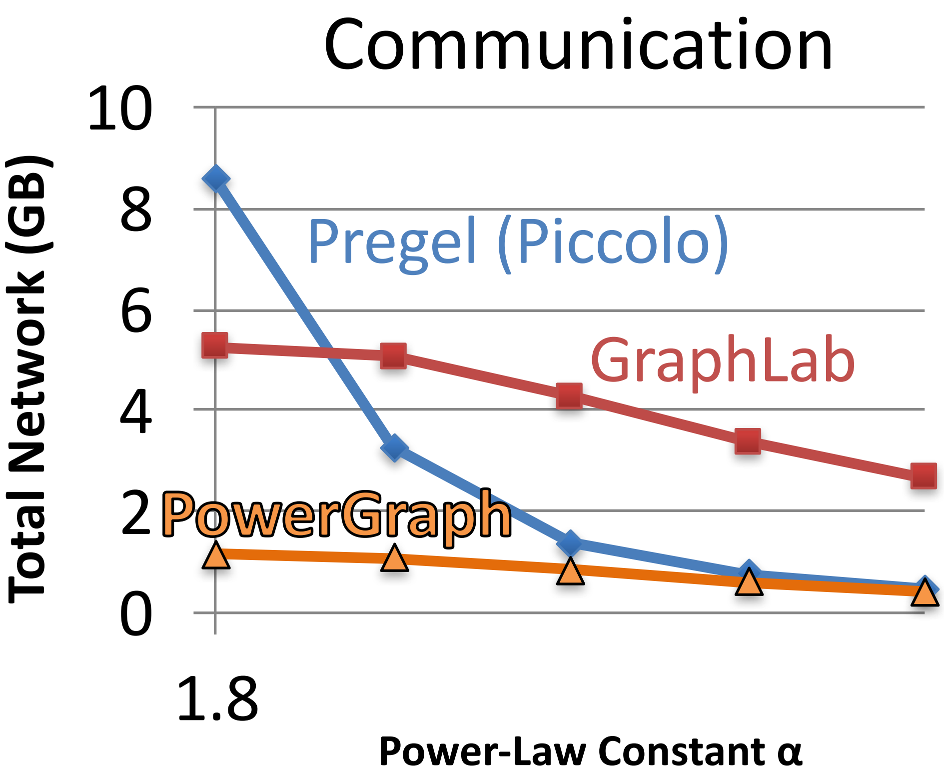
- Implemented as C++ API
- Uses HDFS for Graph Input and Output
- Fault-tolerance is achieved by check-pointing
  - Snapshot time < 5 seconds for twitter network

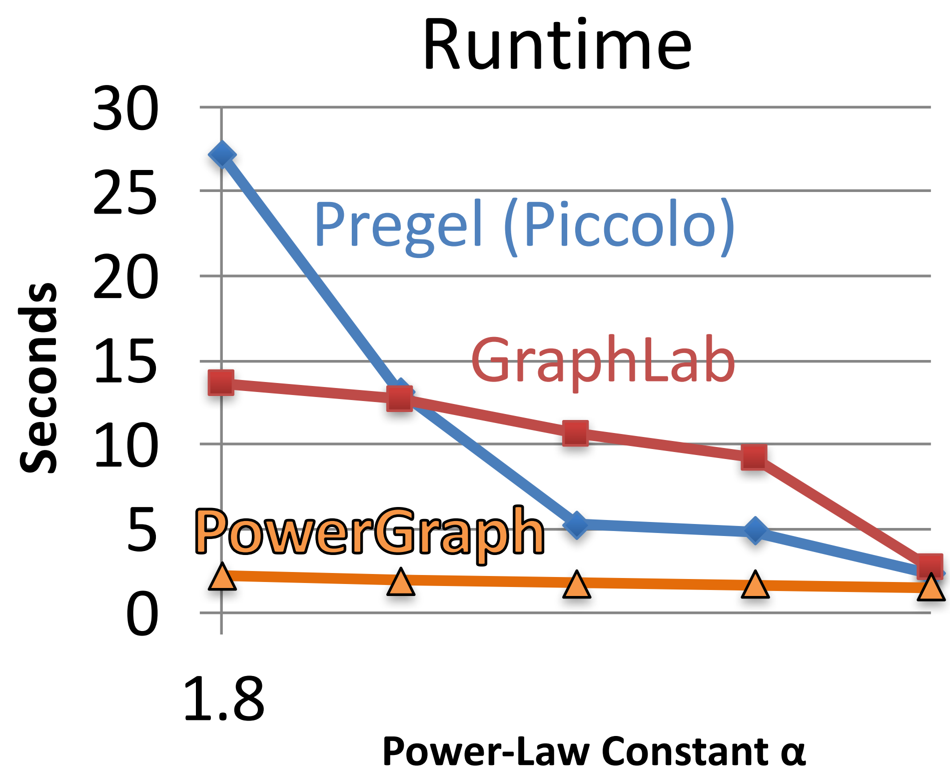
Implemented Many Algorithms

- **Collaborative Filtering**
  - Alternating Least Squares
  - Stochastic Gradient Descent
  - SVD
  - Non-negative MF
- **Statistical Inference**
  - Loopy Belief Propagation
  - Max-Product Linear Programs
  - Gibbs Sampling
- **Graph Analytics**
  - PageRank
  - Triangle Counting
  - Shortest Path
  - Graph Coloring
  - K-core Decomposition
- **Computer Vision**
  - Image stitching
- **Language Modeling**
  - LDA

Comparison with GraphLab & Pregel

- PageRank on Synthetic Power-Law Graphs:





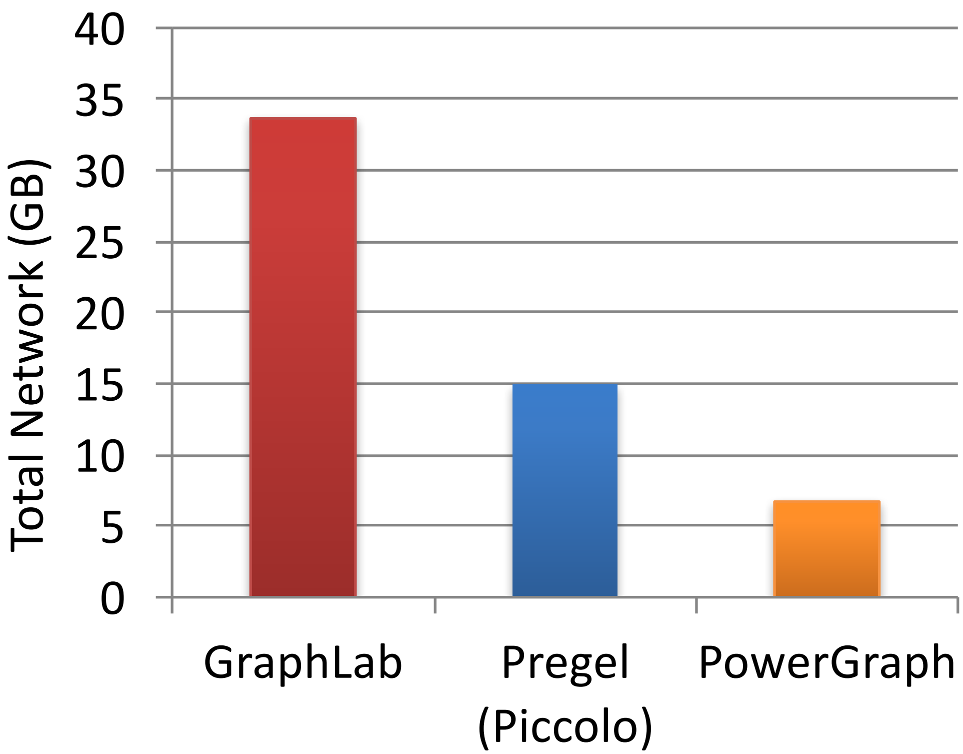
← High-degree vertices ← High-degree vertices

PowerGraph is robust to **high-degree** vertices.

PageRank on the Twitter Follower Graph

Natural Graph with 40M Users, 1.4 Billion Links

Communication



Runtime

Reduces Communication

Runs Faster

32 Nodes x 8 Cores (EC2 HPC cc1.4x)

PowerGraph is Scalable

Yahoo Altavista Web Graph (2002):

One of the largest publicly available web graphs

1.4 Billion Webpages, 6.6 Billion Links

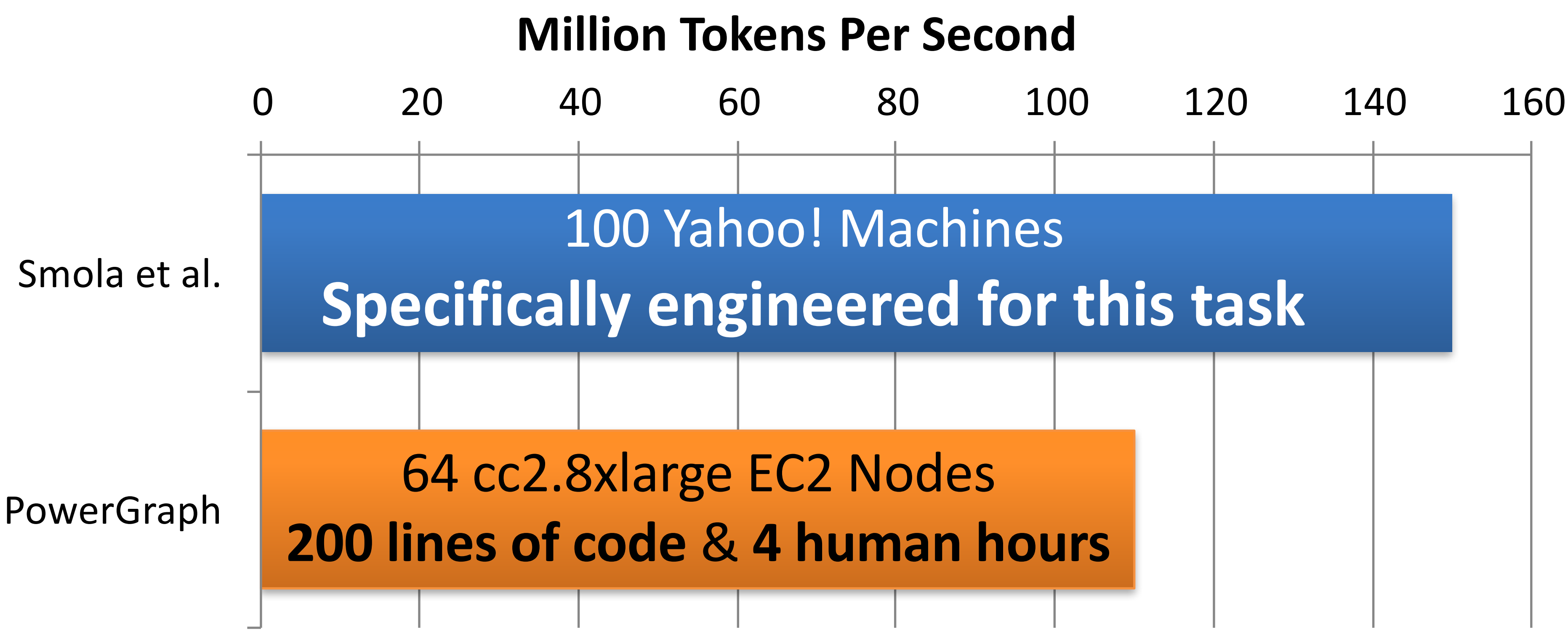
7 Seconds per Iter.

1B links processed per second

30 lines of user code

Topic Modeling

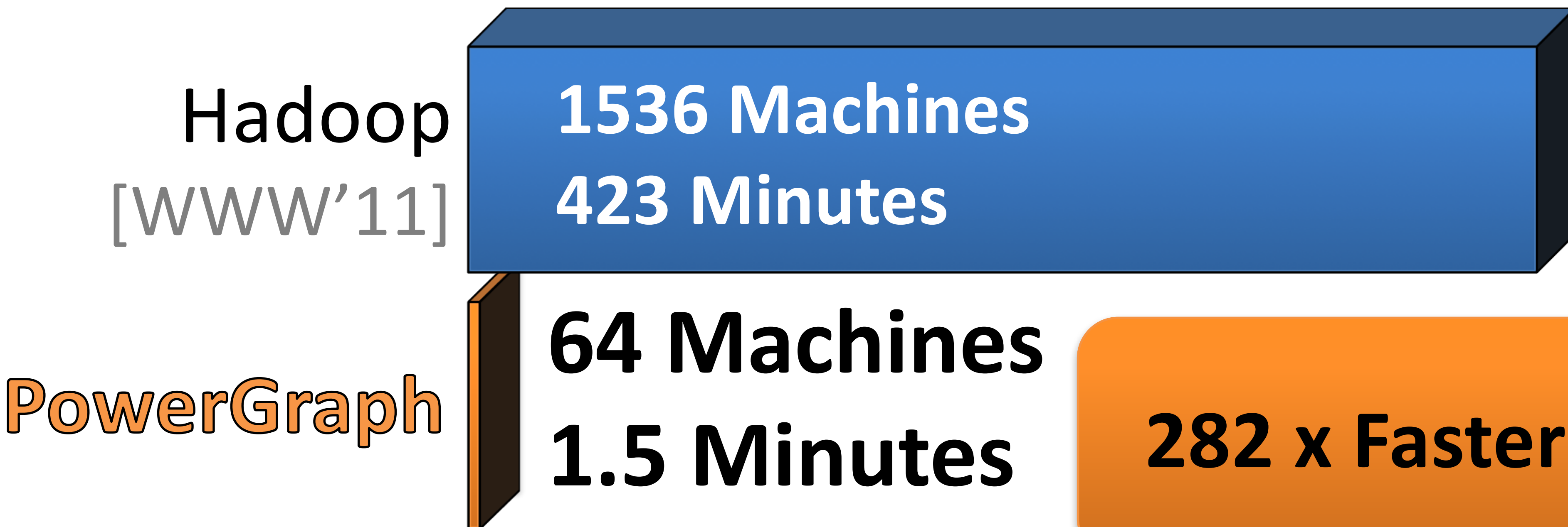
- English language Wikipedia
  - 2.6M Documents, 8.3M Words, 500M Tokens
  - Computationally intensive algorithm



Triangle Counting on The Twitter Graph

Identify individuals with **strong communities.**

Counted: 34.8 Billion Triangles



282 x Faster

Why? **Wrong Abstraction** →

Broadcast O(degree^2) messages per Vertex

S. Suri and S. Vassilvitskii, “Counting triangles and the curse of the last reducer,” WWW’11

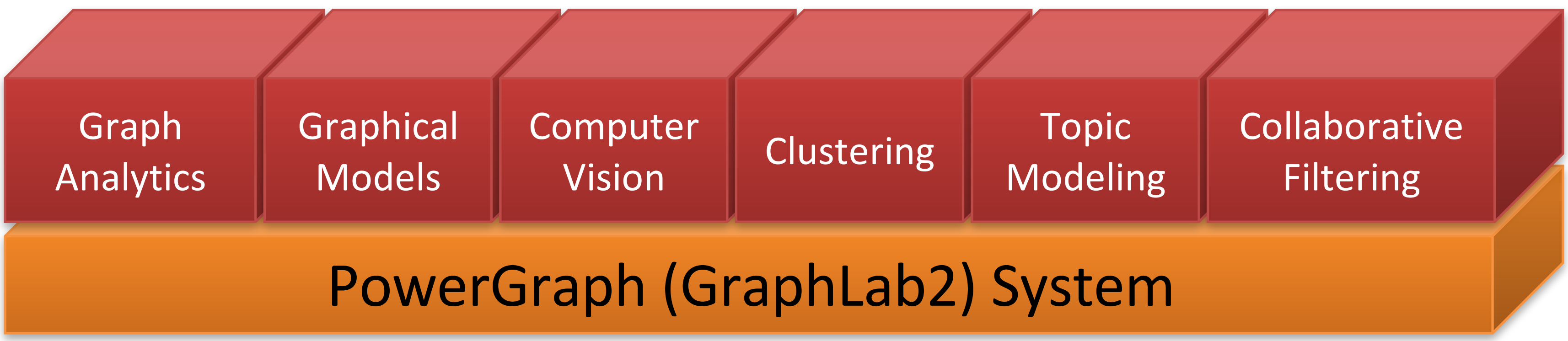
Summary

- *Problem*: Computation on **Natural Graphs** is challenging
  - High-degree vertices
  - Low-quality edge-cuts

- *Solution*: **PowerGraph System**
  - **GAS Decomposition**: split vertex programs
  - **Vertex-partitioning**: distribute natural graphs

- PowerGraph **theoretically** and **experimentally** outperforms existing graph-parallel systems.

Machine Learning and **Data-Mining** Toolkits



Future Work

- Time evolving graphs
  - Support **structural changes** during computation
- Out-of-core storage (GraphChi)
  - Support graphs that don't fit in memory
- Improved Fault-Tolerance
  - Leverage **vertex replication** to reduce snapshots
  - *Asynchronous* recovery

PowerGraph

is GraphLab Version 2.1

Apache 2 License

http://graphlab.org

Documentation… Code… Tutorials… (more on the way)