# PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs

Ashvin Goel

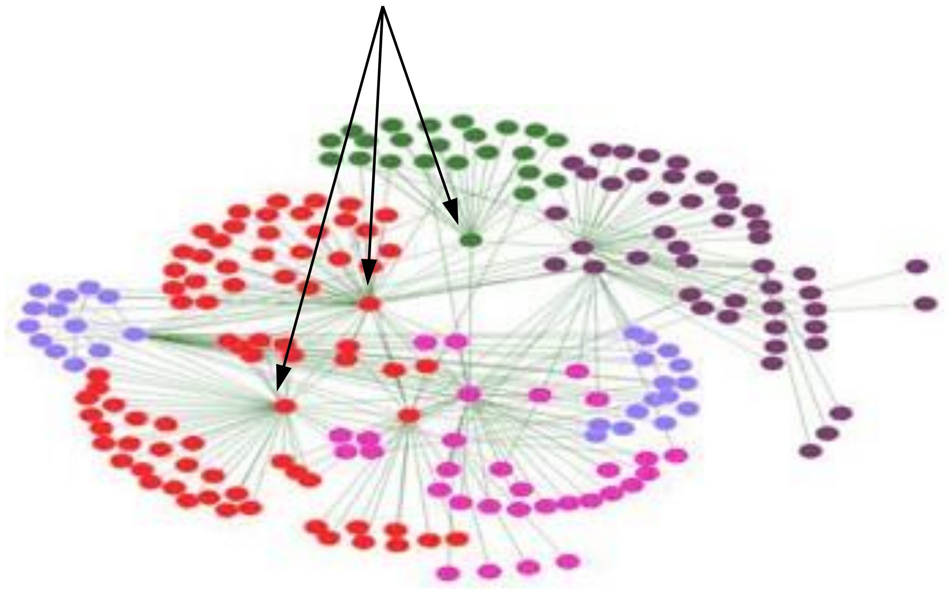
Electrical and Computer Engineering
University of Toronto

ECE1724

Authors: Joseph E. Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, Carlos Guestrin

# Power-Law Graphs

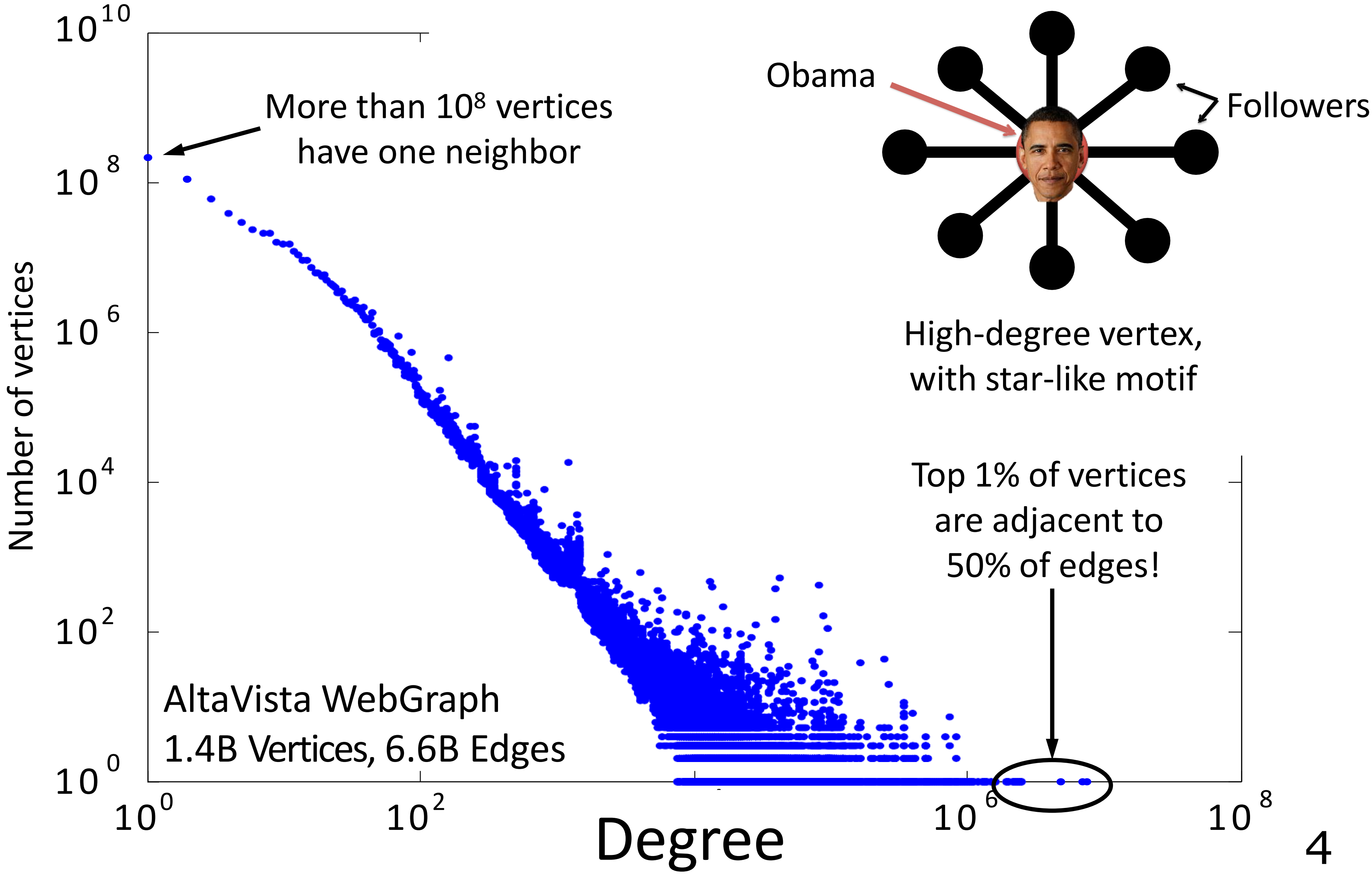
- Degree of vertex is number of edges attached to vertex
- Real-world graphs have power-law degree distribution
  - Highly skewed, long-tailed distribution
    - Most vertices have few edges
    - Some vertices have many edges



# Understanding Power-Law Graphs

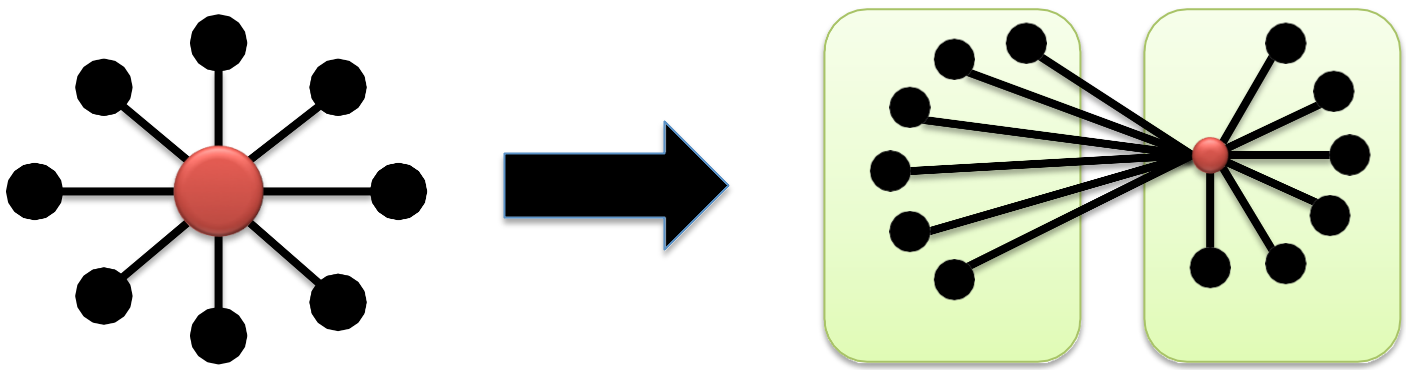
- Probability that a vertex has degree d: *P(d)* $\propto 1/d^{a}$
  - *a* is a constant, *a* > 0, typical value is 2
  - *a* $\downarrow \Rightarrow$ skew (vertices with high degree) $\uparrow$
    density (#edges/#vertices) $\uparrow$
- Intuition
  - Say d = 100, and it goes up by 1
  - P(101)/P(100) = $100^2/101^2$= 0.98 (close to 1)
    - So, significant probability of high degree vertices
  - Compare with exponential distribution: P(d) $\propto 1/e^{d}$
    - P(101)/P(100) = 1/*e* = 0.37, low probability of high degree vertices

# Power-Law Degree Distribution



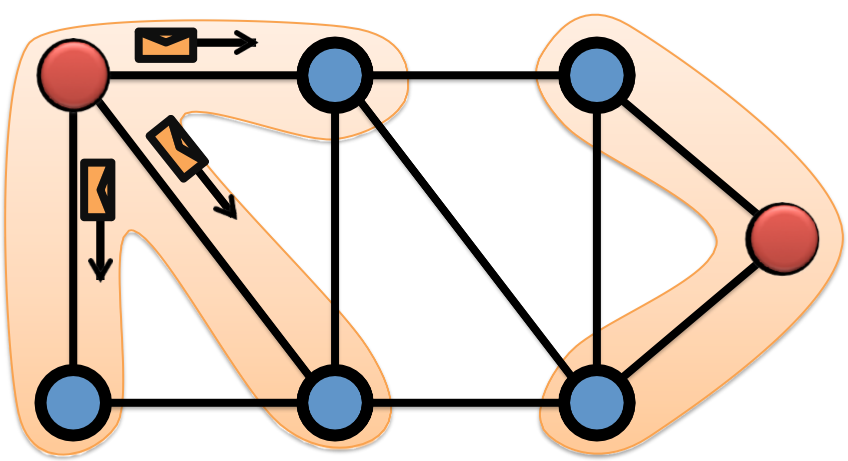
# Why PowerGraph?

- Power-law graphs are hard to partition well
- Distributed graph computation systems perform poorly on such graphs
  - Hard to balance computation and storage load
  - Significant communication across partitions



# The Graph-Parallel Abstraction

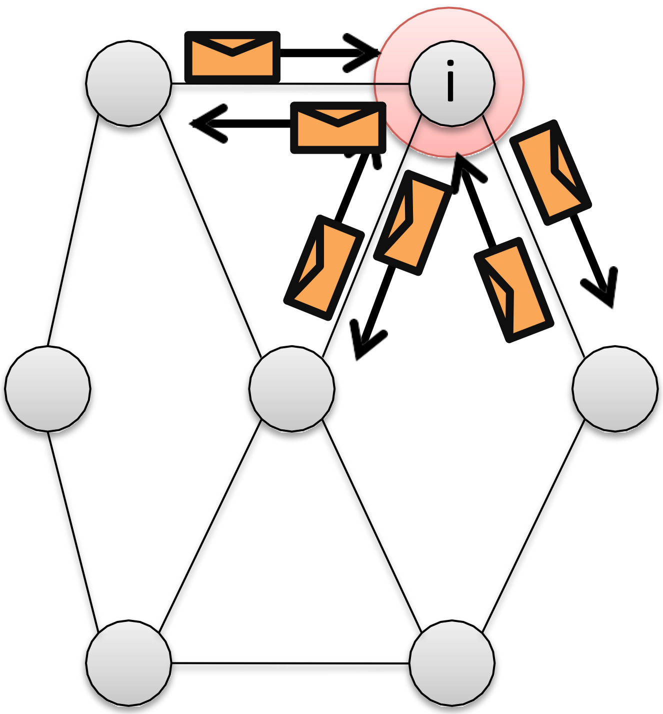
- A user-defined vertex program runs on each vertex
- Graph constrains interaction along edges
  - Using messages, e.g. Pregel
  - Using shared memory, e.g., GraphLab
- Parallelism: run multiple vertex programs concurrently



# The Pregel Abstraction

Vertex programs interact by sending **messages**

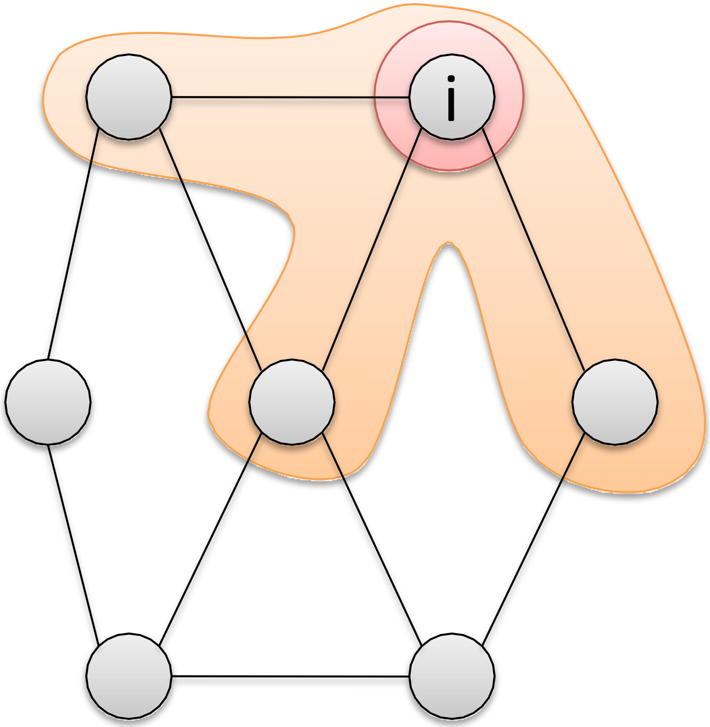
```
Pregel_PageRank(i, messages):
  // Receive all the messages
  total = 0
  foreach( msg in messages) :
    total = total + msg

  // 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
```



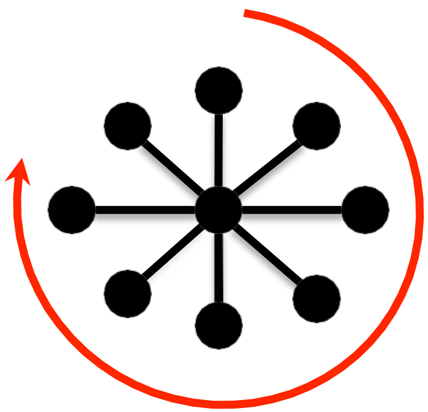
# The GraphLab Abstraction

Vertex programs directly **read** the neighbor's state

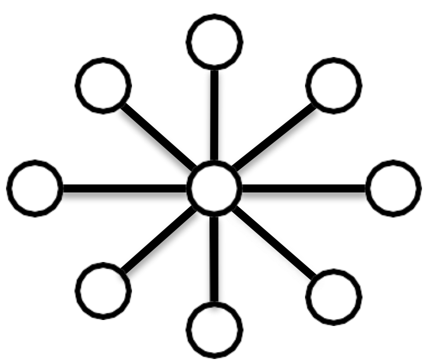
```
GraphLab_PageRank(i)
  // Compute sum over neighbors
  total = 0
  foreach( j in in_neighbors(i)):
    total = total + R[j] * w_ji

  // 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
```



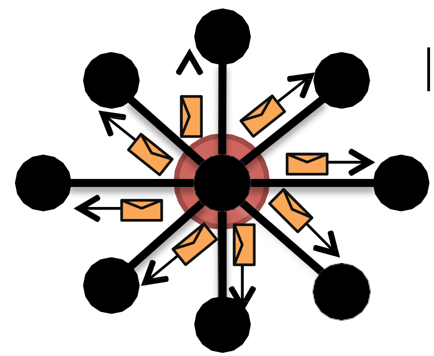
# Challenges of High-Degree Vertices



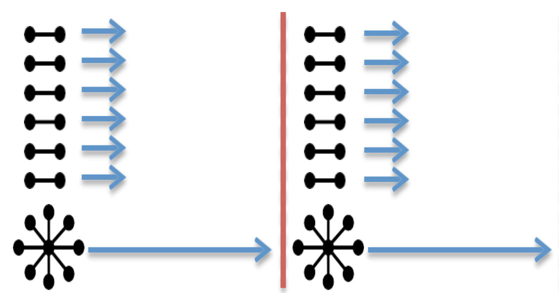
Sequentially process edges



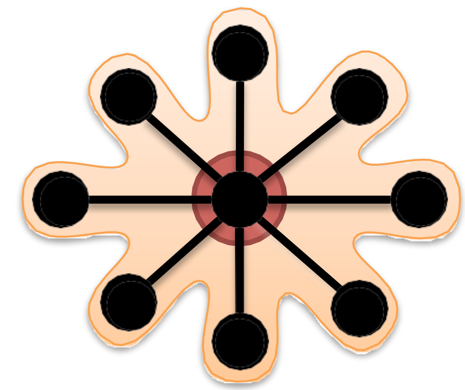
Edge metadata too large for single machine



Sends many messages (Pregel)



Synchronous execution prone to stragglers (Pregel)

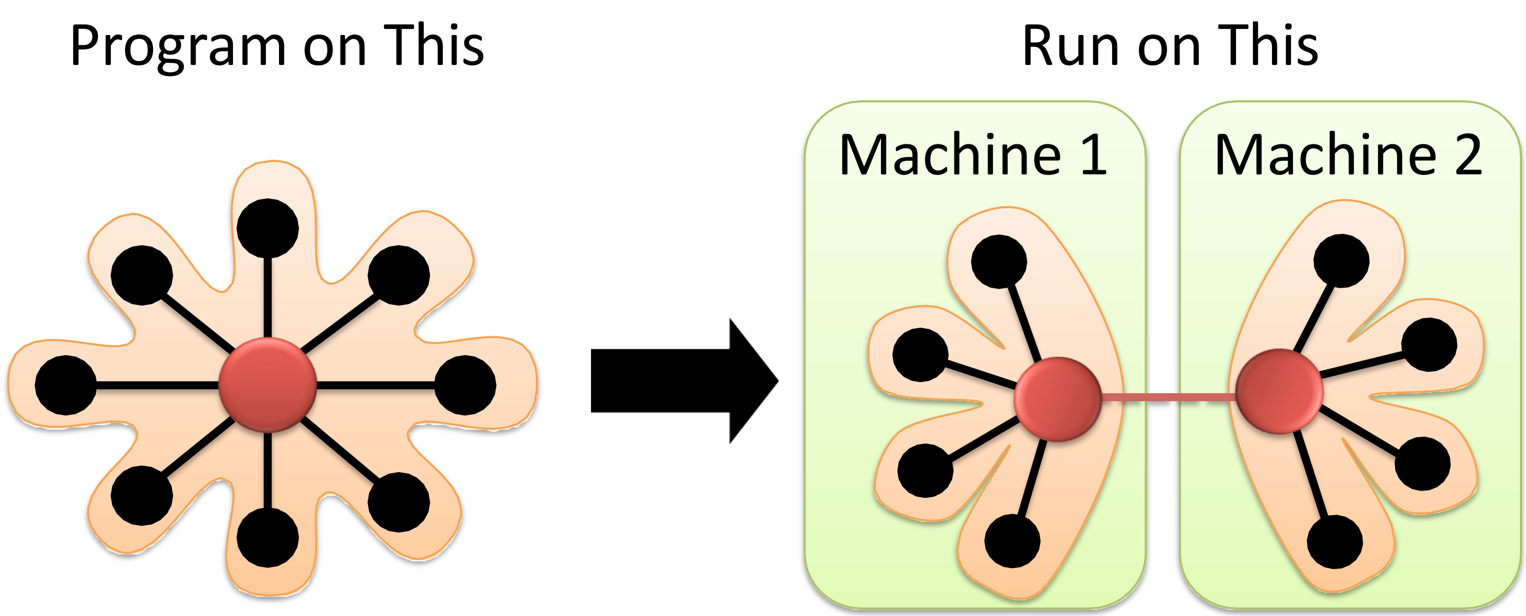


Accesses a large fraction of graph (GraphLab)

Asynchronous execution needs heavy locking (GraphLab)

# Key Idea in PowerGraph

- Split high-degree vertices
  - Parallelize processing of high-degree vertices
  - Guarantee split and non-split vertices operate equivalently



# How to Split Vertex Processing?

Insight: each vertex program consists of three steps

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

Gather information from neighbors

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

Update vertex

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

Signal neighbors & modify edge data

# How to Split Vertex Processing?

Work is proportional to vertex degree in first, third steps

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

Gather information from neighbors

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

Update vertex

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

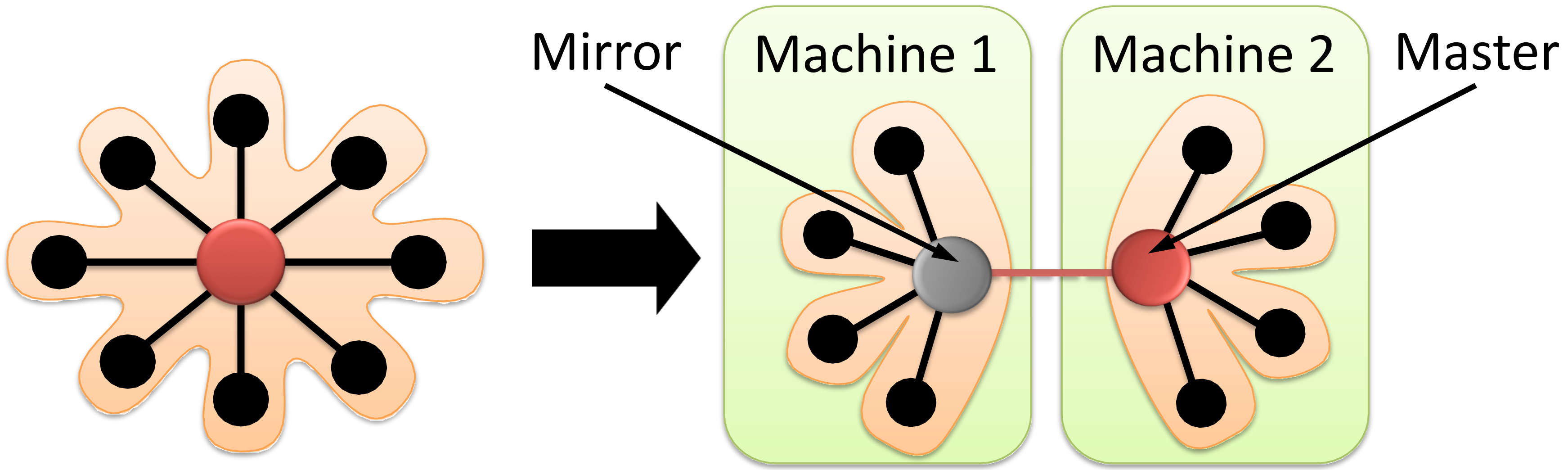
Signal neighbors & update edge data

# PowerGraph GAS Abstraction

- PowerGraph splits the 3 steps of vertex processing into:
  - Gather (gather information from neighbors)
  - Apply (update vertex)
  - Scatter (signal neighbors, update edge data)
- Parallelizes Gather and Scatter phases by moving these computation phases to the data

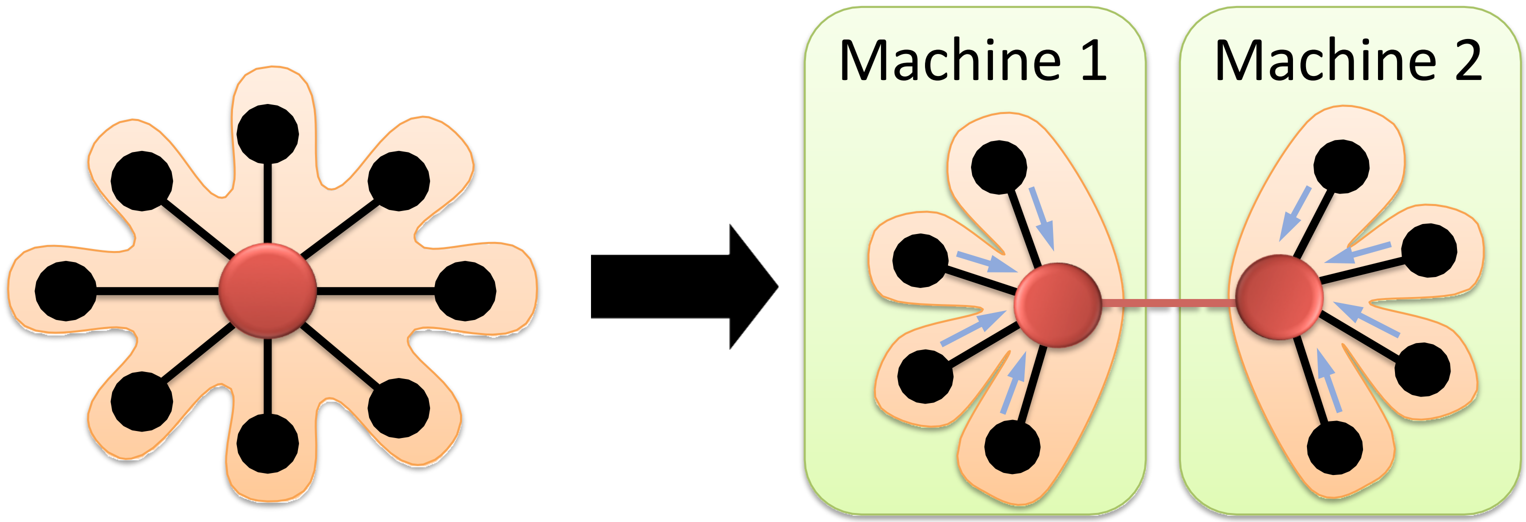
# GAS Abstraction

- Split a high-degree vertex across multiple machines
- Mark one a master, rest are mirrors



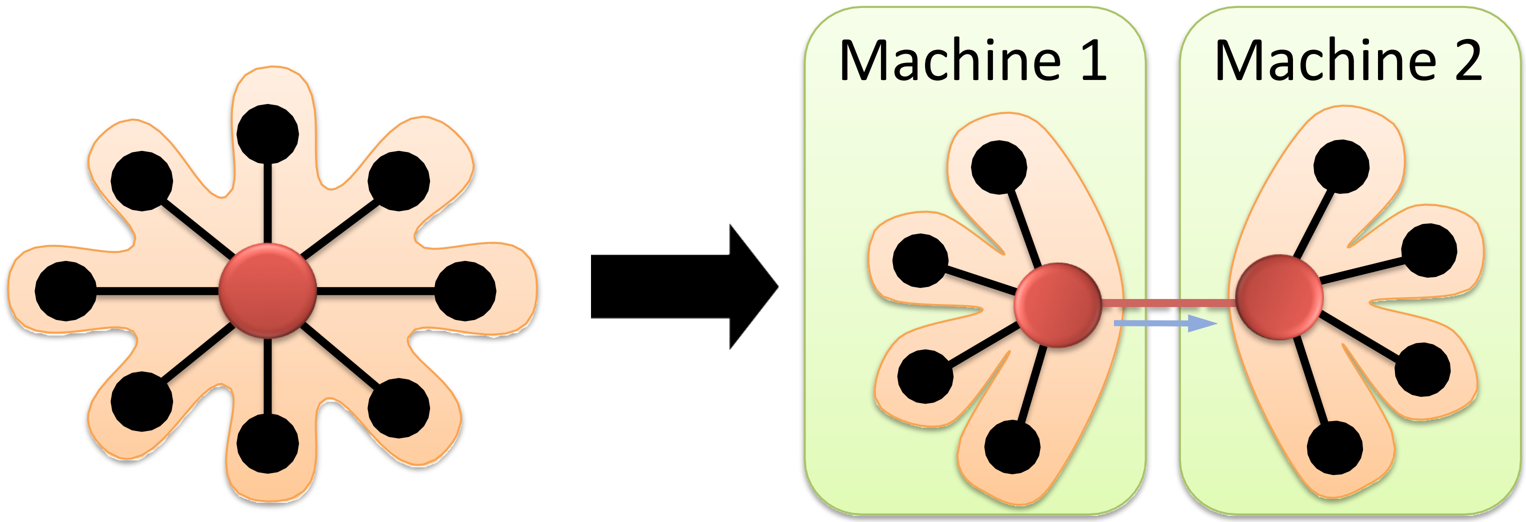
# GAS Abstraction

- Run Gather on all edges, in parallel on the machines



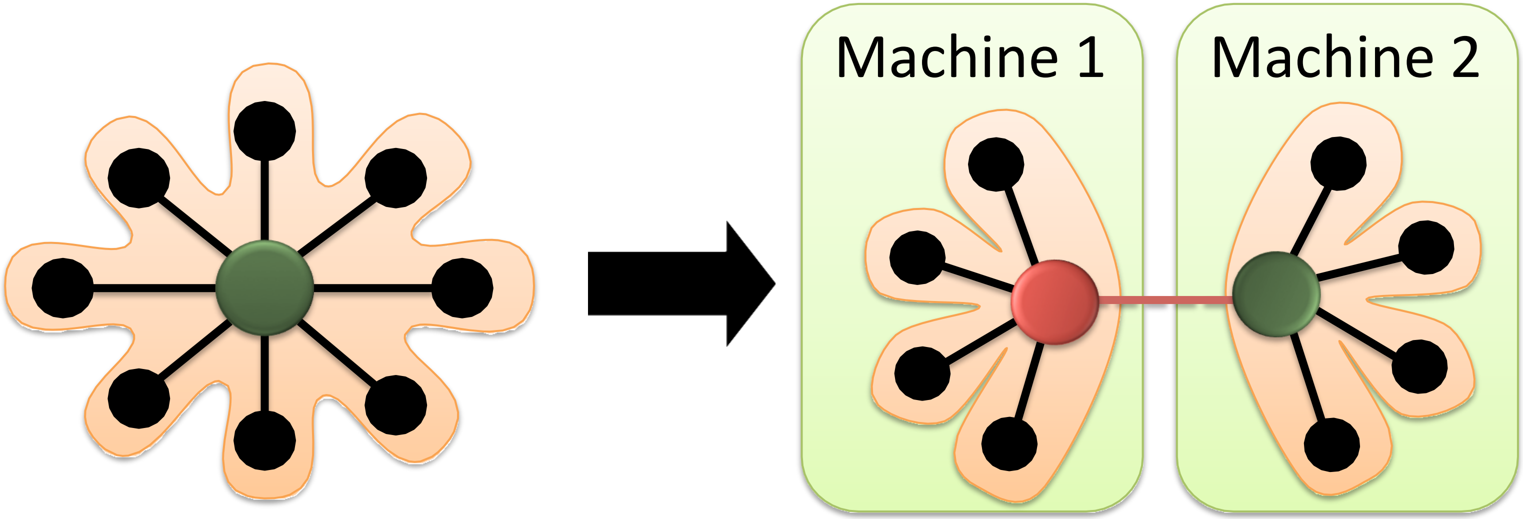
# GAS Abstraction

- Run Gather on all edges, in parallel on the machines
  - Send partial sum from mirrors to master
    - Similar to Pregel Combiners



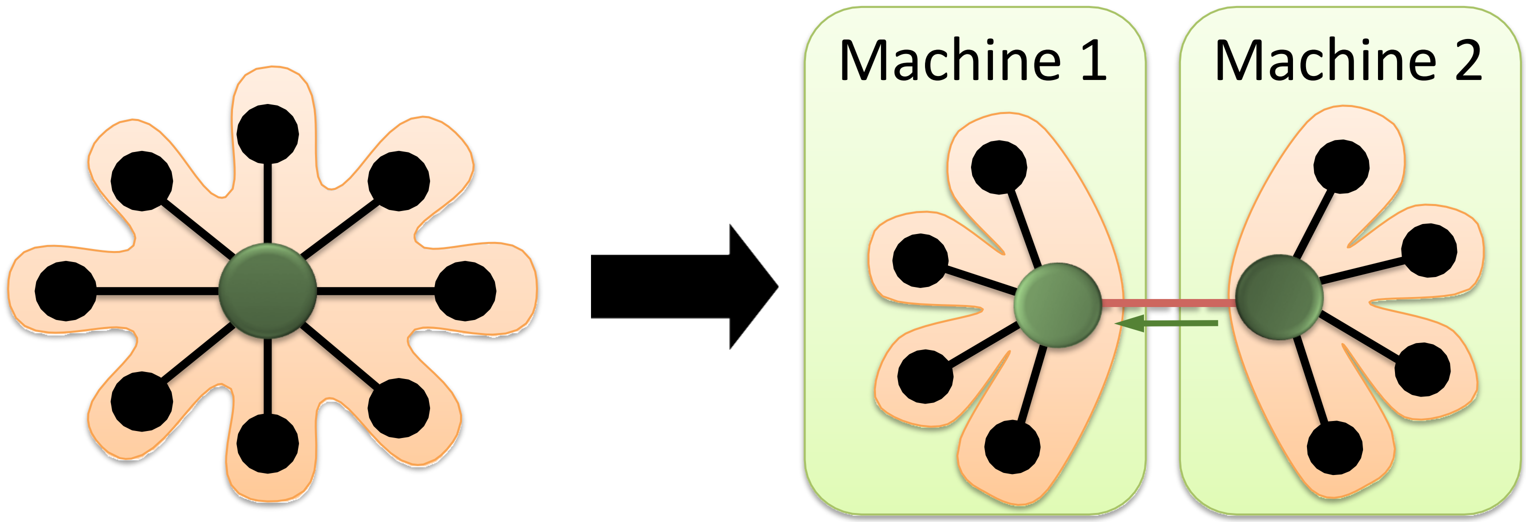
# GAS Abstraction

- Run Gather on all edges, in parallel on the machines
  - Send partial sum from mirrors to master
- Apply update based on partial sums on master vertex



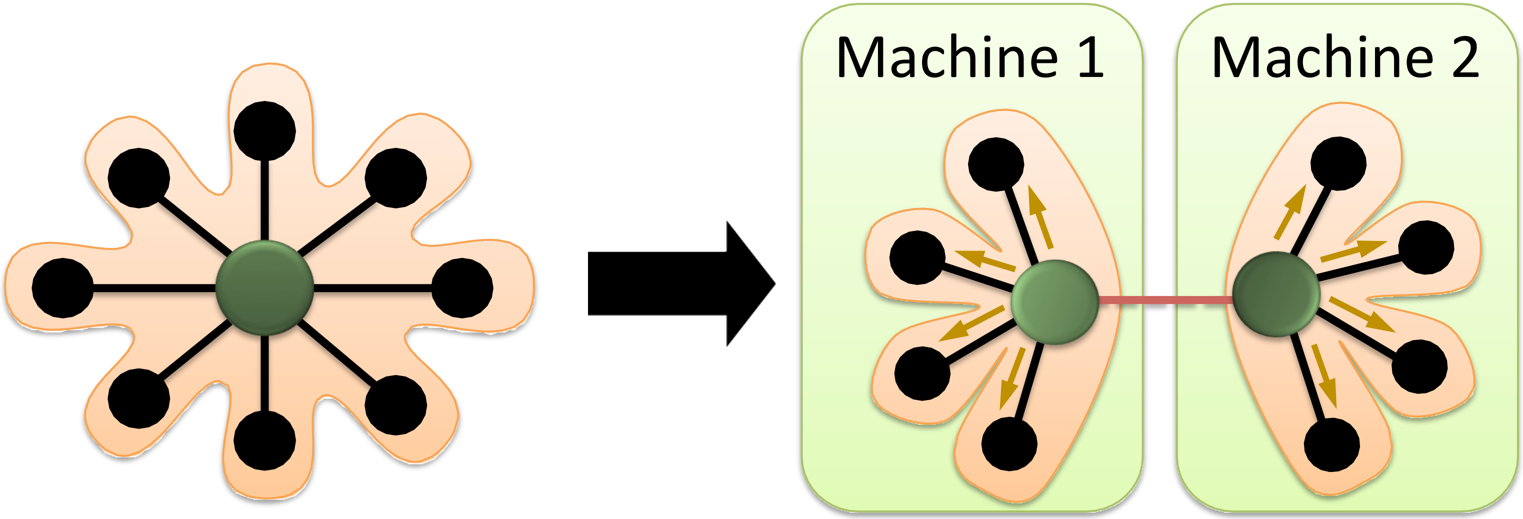
# GAS Abstraction

- Run Gather on all edges, in parallel on the machines
  - Send partial sum from mirrors to master
- Apply update based on partial sums on master vertex
  - Sent vertex update to mirrors



# GAS Abstraction

- Run Gather on all edges, in parallel on the machines
  - Send partial sum from mirrors to master
- Apply update based on partial sums on master vertex
  - Sent vertex update to mirrors
- Run Scatter on all edges, in parallel on the machines



# PageRank in PowerGraph

Gather and Scatter operate on single edge, not all edges

```
PowerGraph_PageRank(i)
  // Compute sum over neighbors

  Gather(j->i): return R[j] * wji
  sum(a, b) = a + b:

  // Update the PageRank
  Apply(I, Σ): R[i] = 0.15 + Σ

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

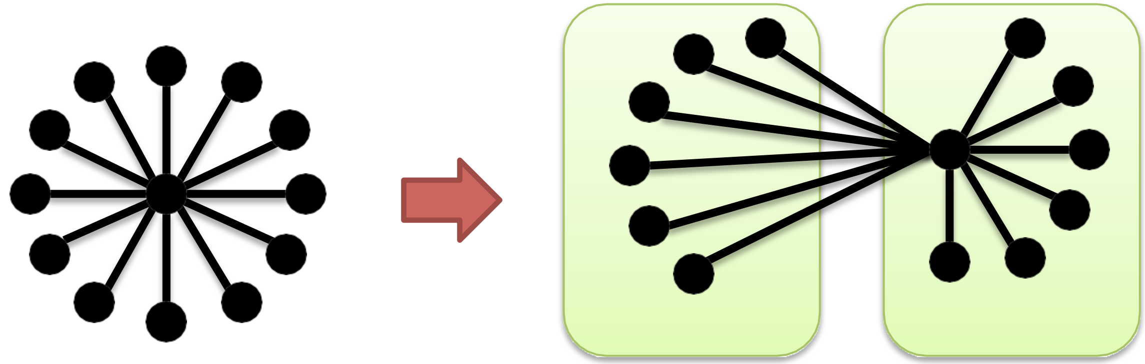
# Graph Partitioning

- GAS model spreads processing load for high-degree vertices by splitting them across machines
- This approach enables a new method of graph partitioning called vertex cut
  - Assign each edge to a machine
  - A vertex may span machines
- For power-law graph, vertex cuts help:
  - Improve load balancing
  - Reduce communication and storage overhead

# Edge Cuts versus Vertex Cuts

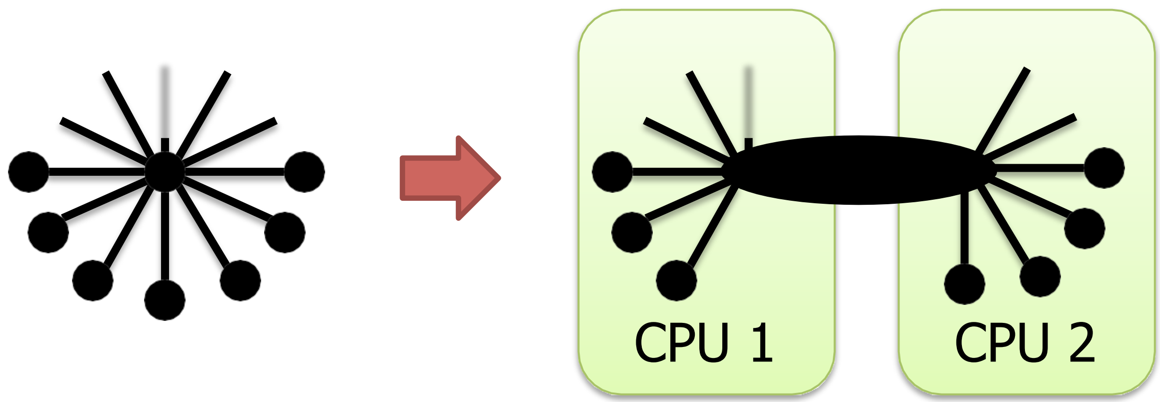
- Power-law graphs have many more edges than vertices

Edge Cut



1. Assign vertices to partitions
2. Edges may be duplicated across partitions
3. Must synchronize many edges

Vertex Cut




1. Assign edges to partitions
2. Vertices may be duplicated across partitions
3. Must synchronize fewer vertices

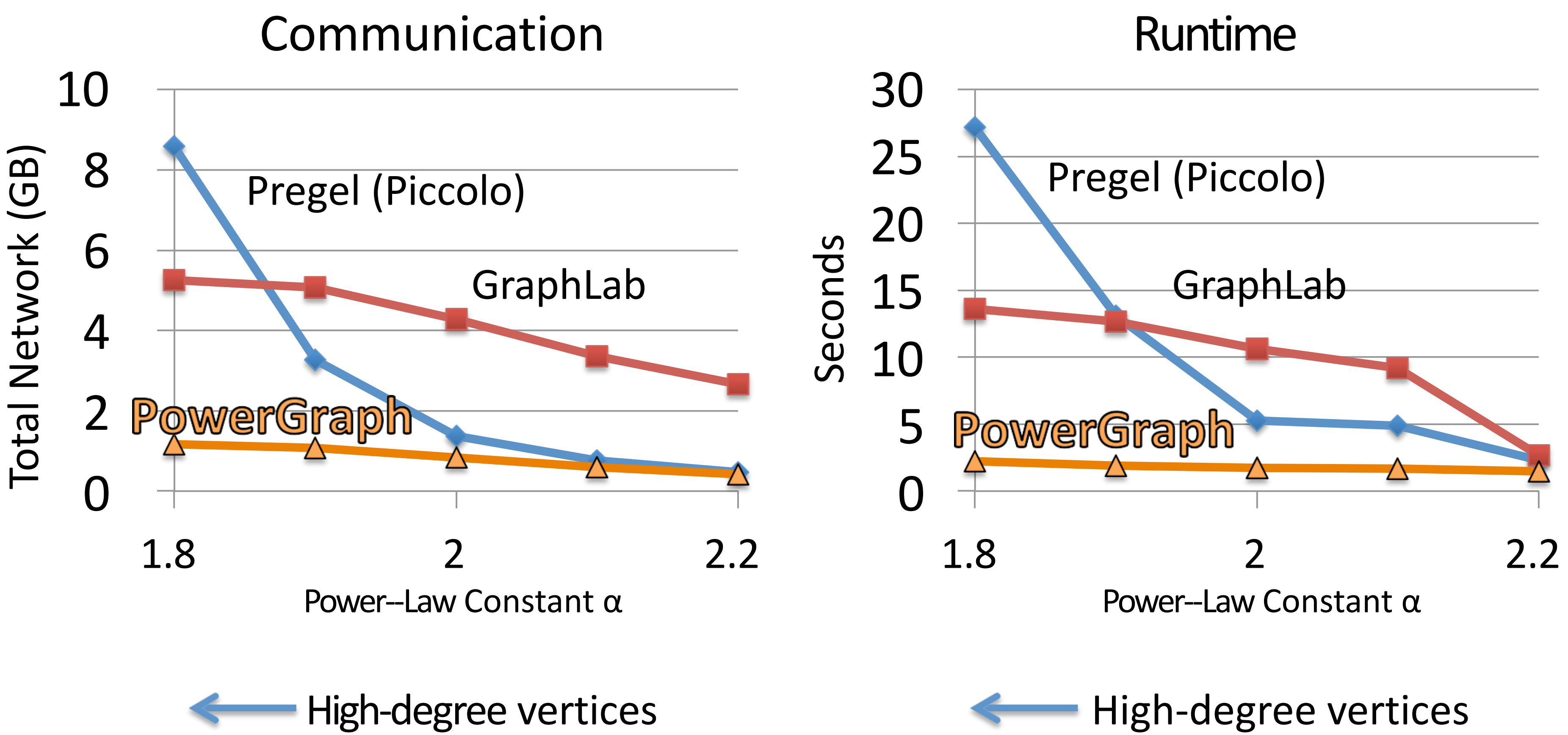
# Constructing Vertex Cuts

- Goals
  - Assign edges to machines evenly
  - Minimize number of machines spanned by each vertex
  - Assign each edge as it is loaded, without reassigning it again
- Three distributed approaches
  - Random edge placement
  - Coordinated greedy edge placement
    - Place an edge on a machine that already has vertices of that edge
    - Requires coordination to track current vertex->machine assignment
  - Oblivious greedy edge placement
    - Same as above, but use local approximation of vertex->machine assignment, so no coordination required

# Synchronization

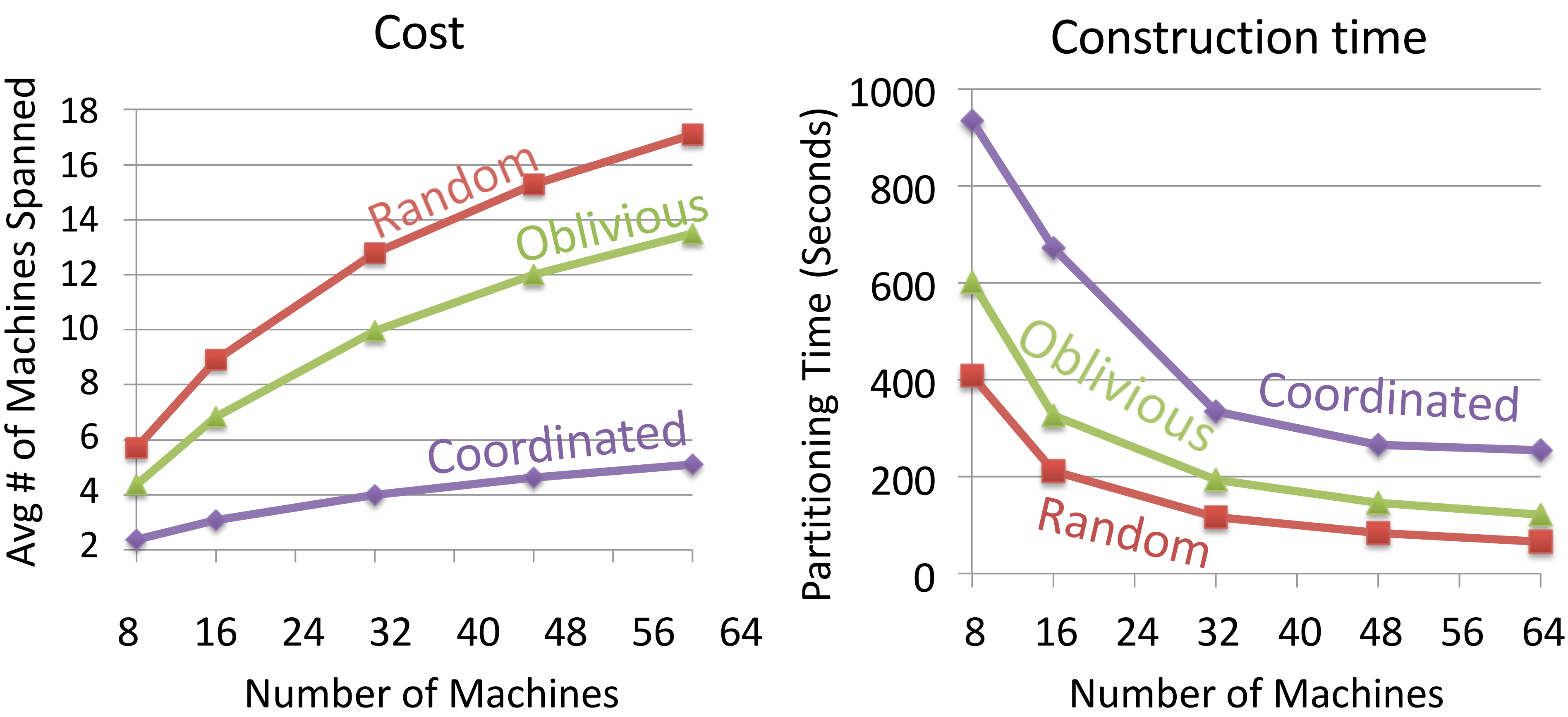
- PowerGraph supports three execution modes:
  - Synchronous
    - Each of the GAS phases run in bulk-synchronous model
  - Asynchronous
    - The GAS phases run completely asynchronously
    - Lock GAS parameters to avoid races
  - Asynchronous + Serializable
    - Neighboring vertices do not run simultaneously
    - Similar to Dining Philosopher problem

# Comparison with Pregel & GraphLab



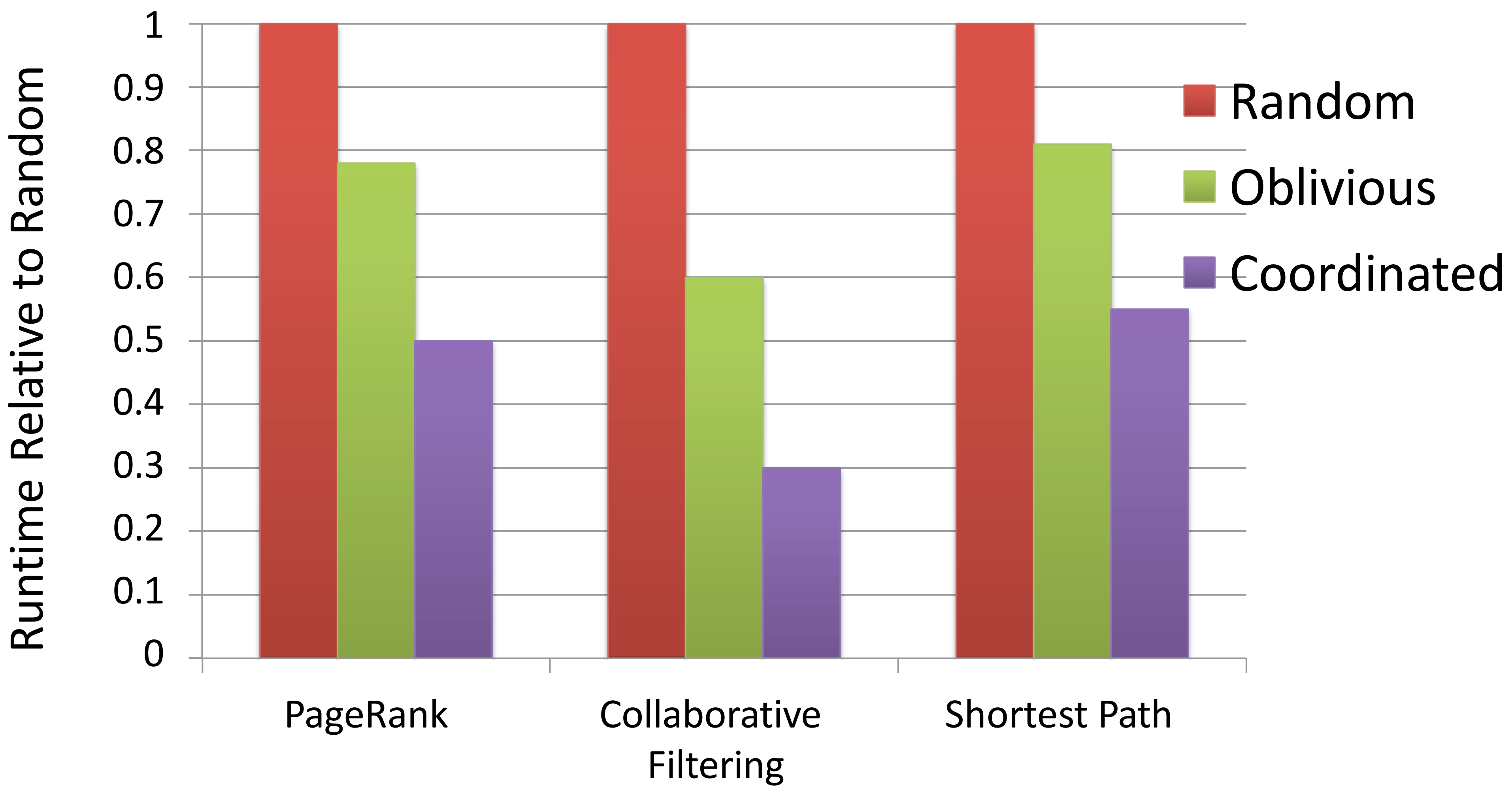

PageRank on Synthetic Power-Law Graphs

# Partitioning Cost




Twitter Graph: 41M vertices, 1.4B edges

# Performance With Different Partitioning Schemes



# Conclusions

- Real-world graphs are power-law graphs
- Computation on power-law graph is challenging
  - High-degree vertices
  - Low-quality edge-cuts
- PowerGraph proposes: 1) GAS decomposition model for splitting, parallelizing vertex programs, 2) Vertex cuts for partitioning power-law graphs
- PowerGraph theoretically and experimentally outperforms Pregel and GraphLab
- PowerGraph is available as Apache GraphLab 2.1

# Discussion

# Q1

- What problems do power-law distributions cause for graph processing?

# Q2

- Why do vertex cuts make it easier to perform load balancing compared to edge cuts?

# Q3

- Powergraph splits processing into three steps:

  gather + sum
  apply
  scatter

- Assuming a node X has four neighbors A, B, C, D, the **sum** operation is performed as follows:
- sum(gather(A), gather(B), gather(C), gather(D))
- Why does the **sum** operation need to be commutative and associative?