GraphLab2

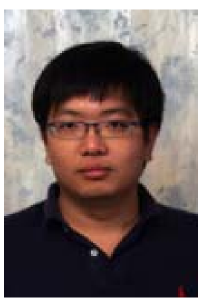
A System for Distributed Graph-Parallel Machine Learning

Joseph Gonzalez

Postdoc, UC Berkeley AMPLab

PhD @ CMU

The Team:


Yucheng Low


Haijie Gu


Aapo Kyrola


Danny Bickson


Carlos Guestrin


Alex Smola


Guy Blelloch

About me …

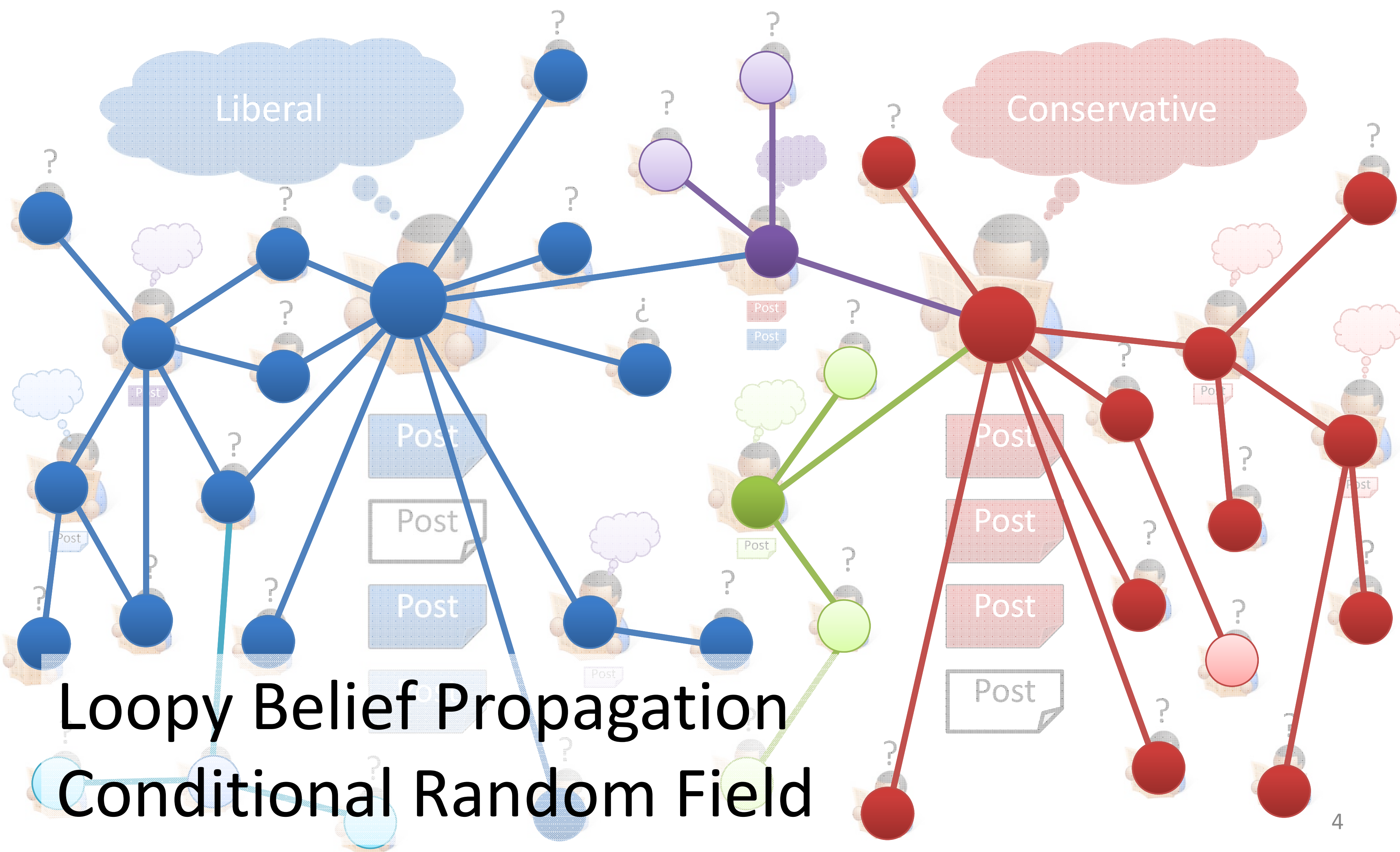
Scalable Machine Learning + Big Graphical Models

BigLearning

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

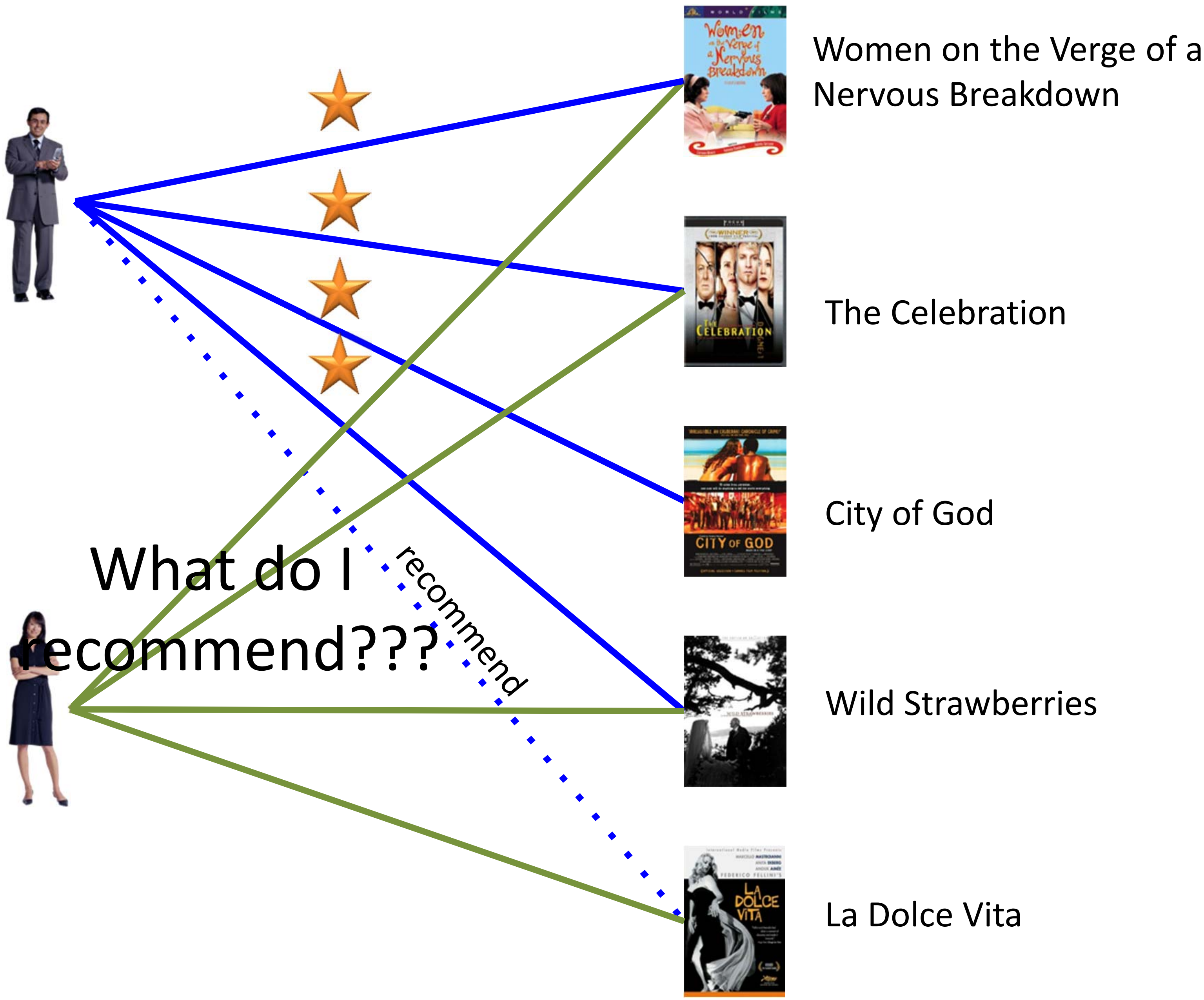
- Identify influential people and information
- Find communities
- Target ads and products
- Model complex data dependencies

Example: *Estimate Political Bias*

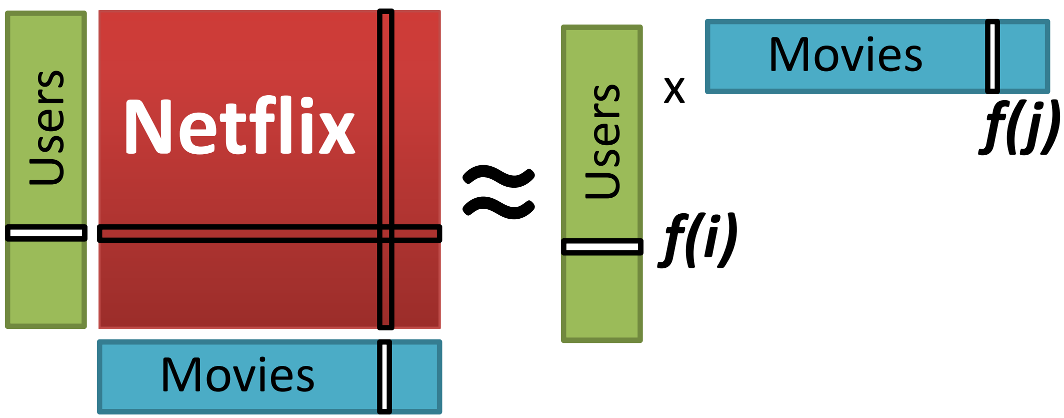


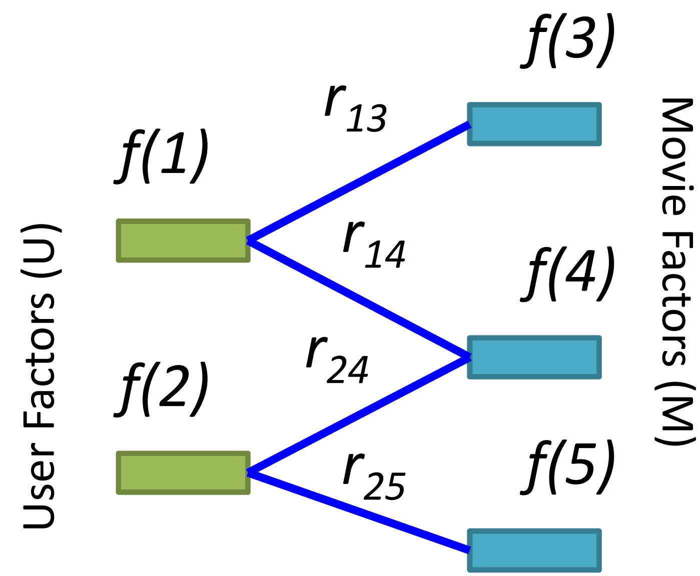
Loopy Belief Propagation
Conditional Random Field

Collaborative Filtering: Exploiting Dependencies



Matrix Factorization
Alternating Least Squares (ALS)





Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)$$

Factor for *User i*

Factor for *Movie j*

PageRank

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

Rank of user *i*

Weighted sum of neighbors' ranks

- Everyone starts with equal ranks
- Update ranks in parallel
- Iterate until convergence

How *should* we program **graph-parallel** algorithms?

Low-level tools like *MPI* and *Pthreads*?

- Me, during my first years of grad school

Threads, Locks, and MPI

- Graduate students **repeatedly** solve the same parallel design challenges:
  - *Implement* and ***debug*** complex parallel system
  - Tune for a ***single*** parallel platform
  - Six months later the conference paper contains:

    "We implemented ______ in parallel."
- The resulting code:
  - is difficult to **maintain** and **extend**
  - **couples** learning *model* and *implementation*

How *should* we program **graph-parallel** algorithms?

High-level Abstractions!

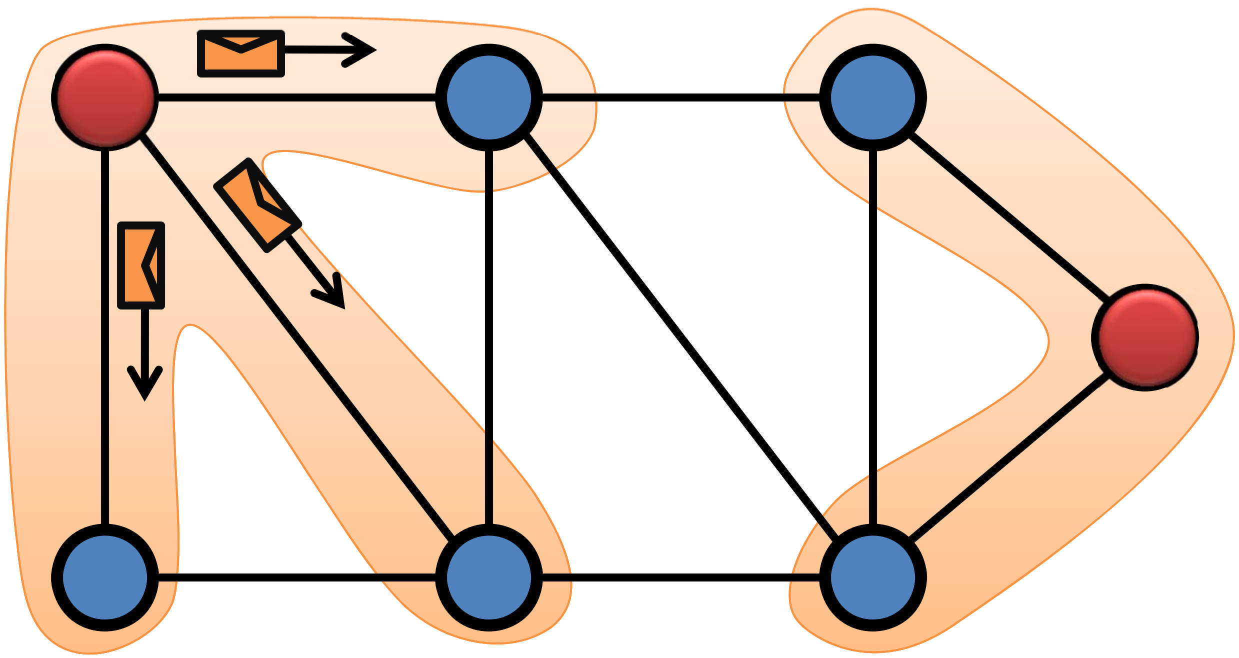
- Me, now

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

"Think like a Vertex."

-Malewicz et al. [SIGMOD'10]

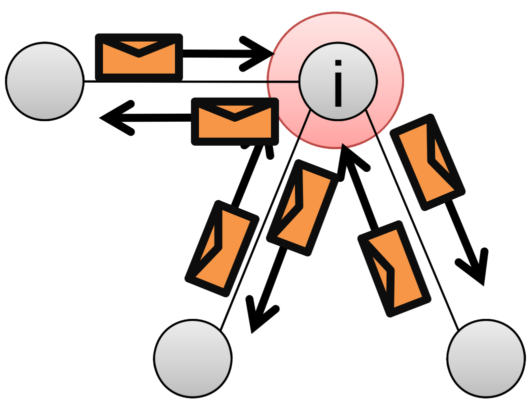


Graph-parallel Abstractions

Better for Machine Learning

Pregel

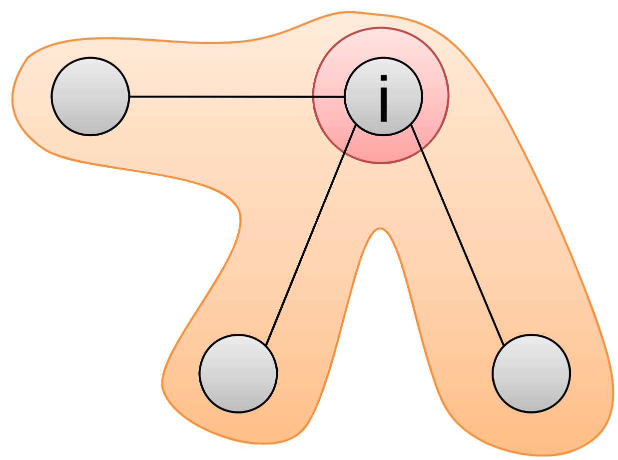
Messaging



Synchronous

GraphLab

Shared State

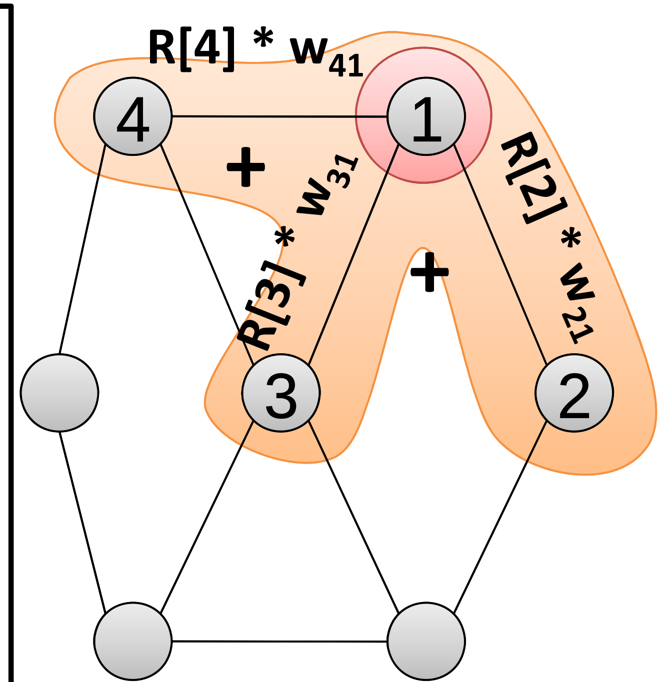


Dynamic Asynchronous

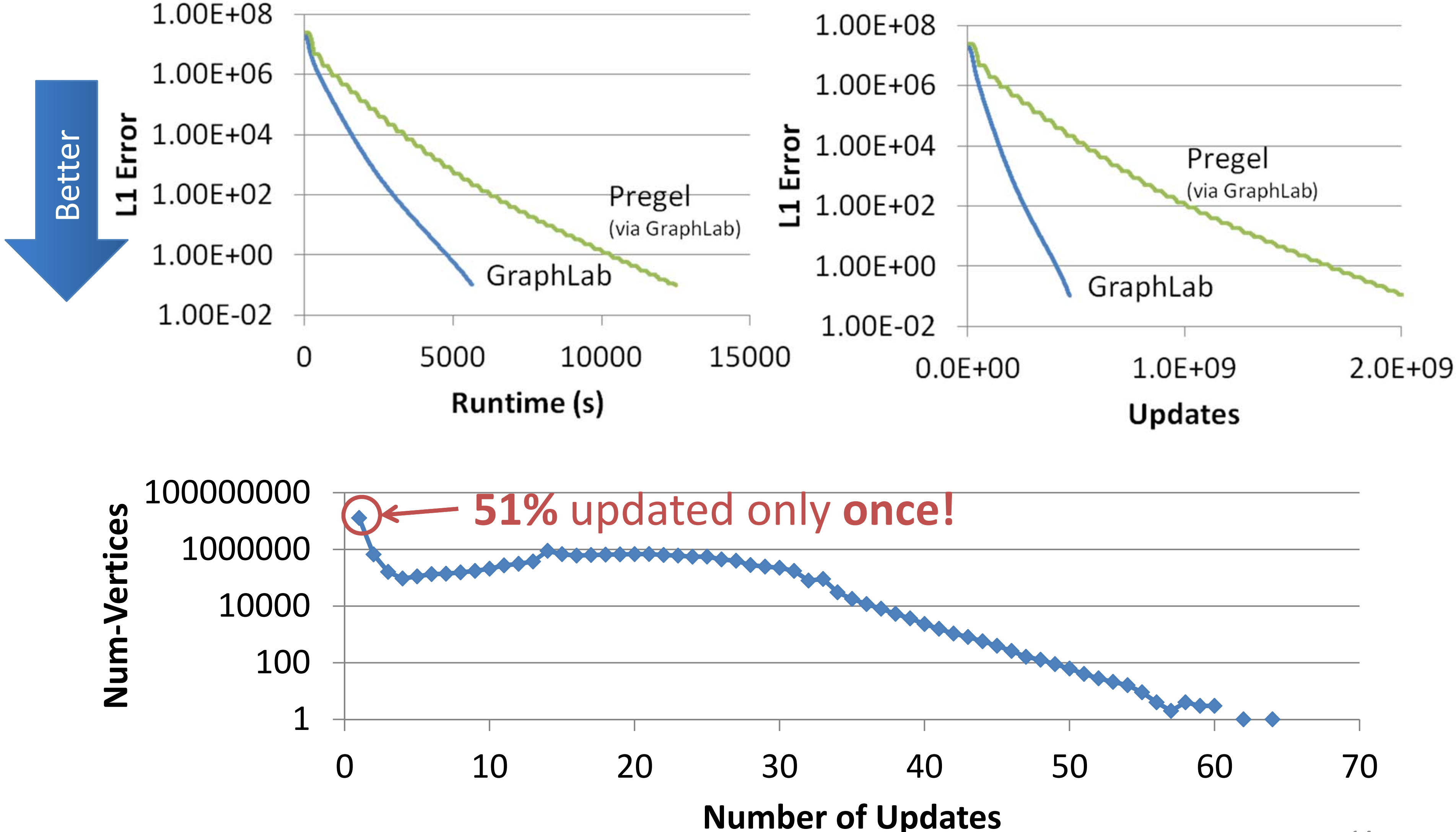
The GraphLab Vertex Program

Vertex Programs directly **access** adjacent vertices and edges

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

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

  // Trigger neighbors to run again
  priority = |R[i] – oldR[i]|
  if R[i] not converged then
   signal neighborsOf(i) with priority
```

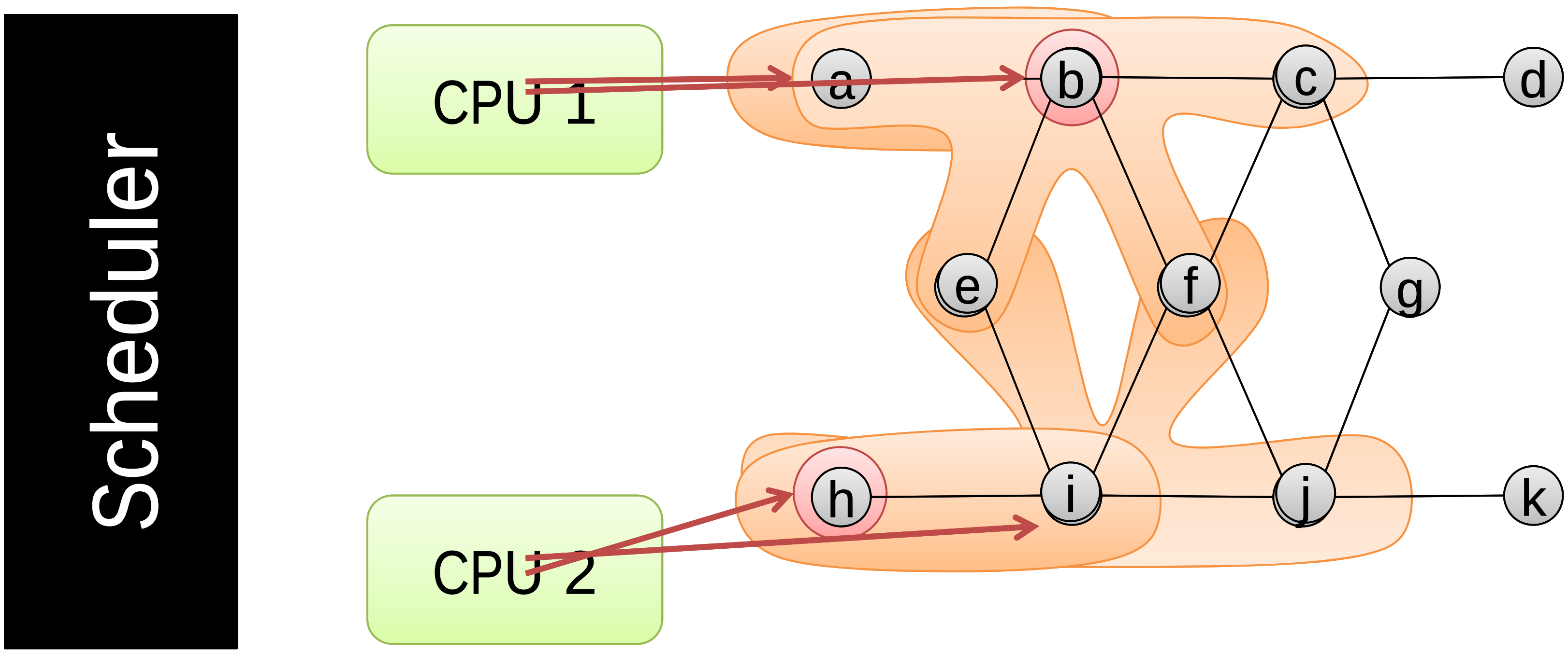


Benefit of Dynamic PageRank



GraphLab **Asynchronous** Execution

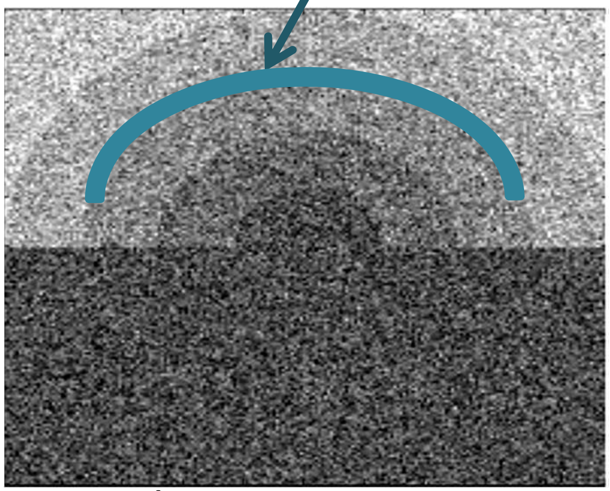
The **scheduler** determines the order that vertices are executed



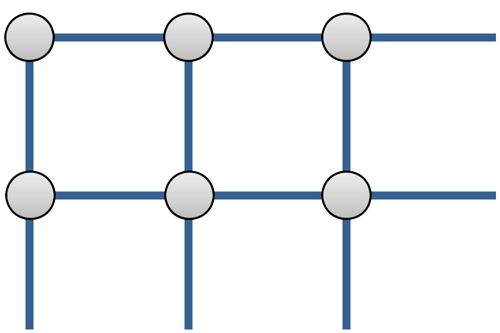
Scheduler can **prioritize** vertices.

Asynchronous Belief Propagation

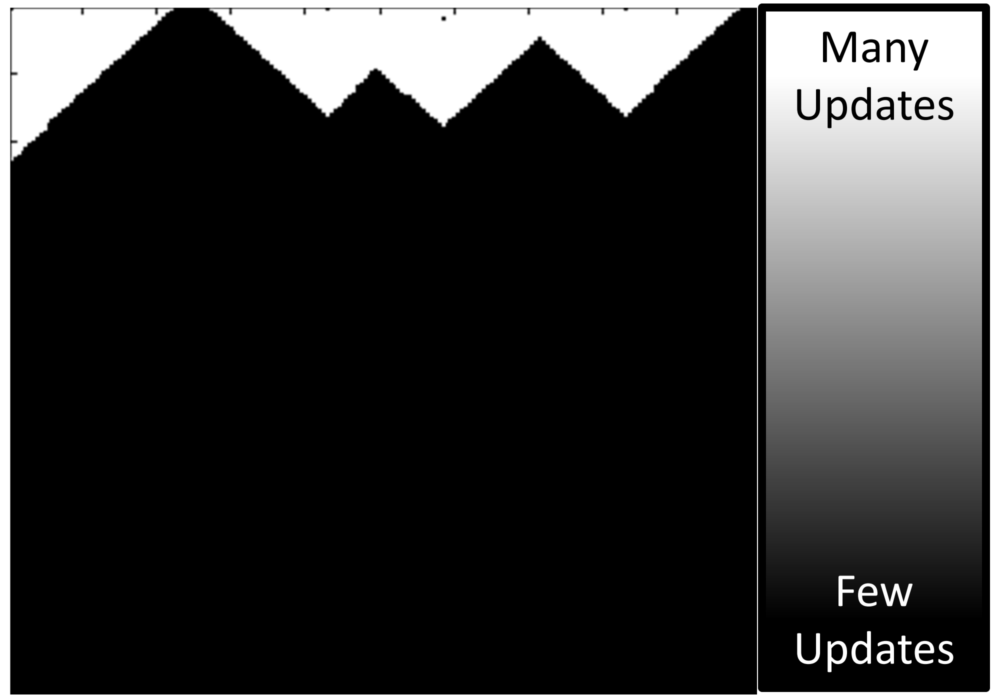
Challenge = Boundaries



Synthetic Noisy Image



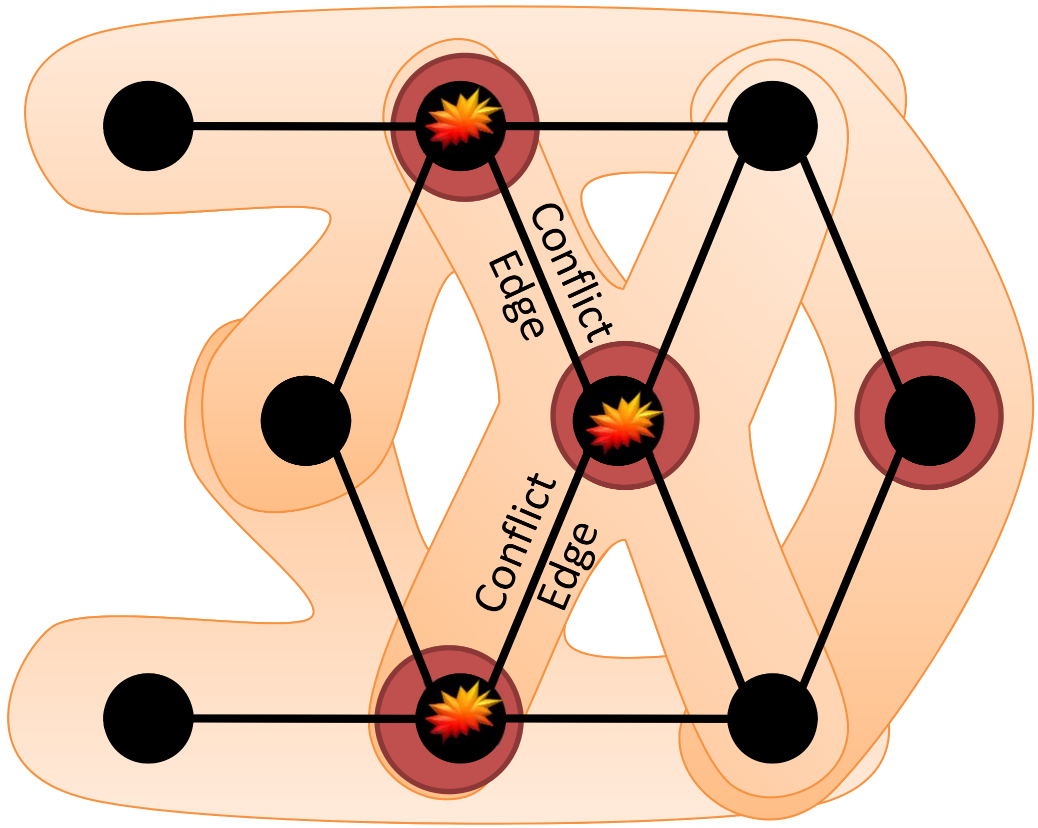
Graphical Model



Cumulative Vertex Updates

Algorithm identifies and focuses on hidden sequential structure

GraphLab Ensures a **Serializable** Execution



- Enables: Gauss-Seidel iterations, Gibbs Sampling, Graph Coloring, …

Never Ending Learner Project (CoEM)

- Language modeling: named entity recognition

Hadoop (BSP)	95 Cores	7.5 hrs
GraphLab	**16 Cores**	**30 min**
Distributed GraphLab	**32 EC2 machines**	**80 secs**

0.3% of Hadoop time

Thus far…

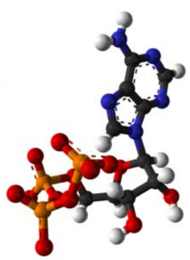
GraphLab provided a powerful new abstraction

But…

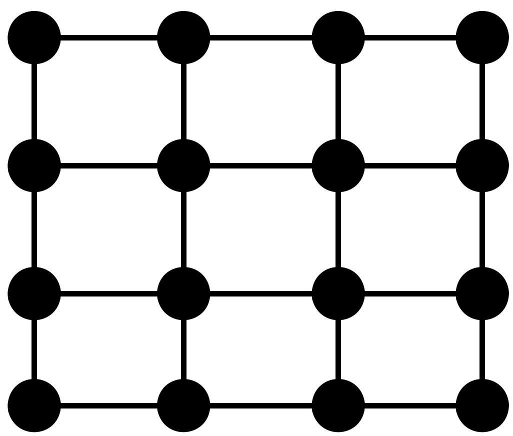
We couldn’t scale up to Altavista Webgraph from 2002 1.4B vertices, 6.6B edges

Natural Graphs

Graphs derived from natural phenomena



Properties of Natural Graphs

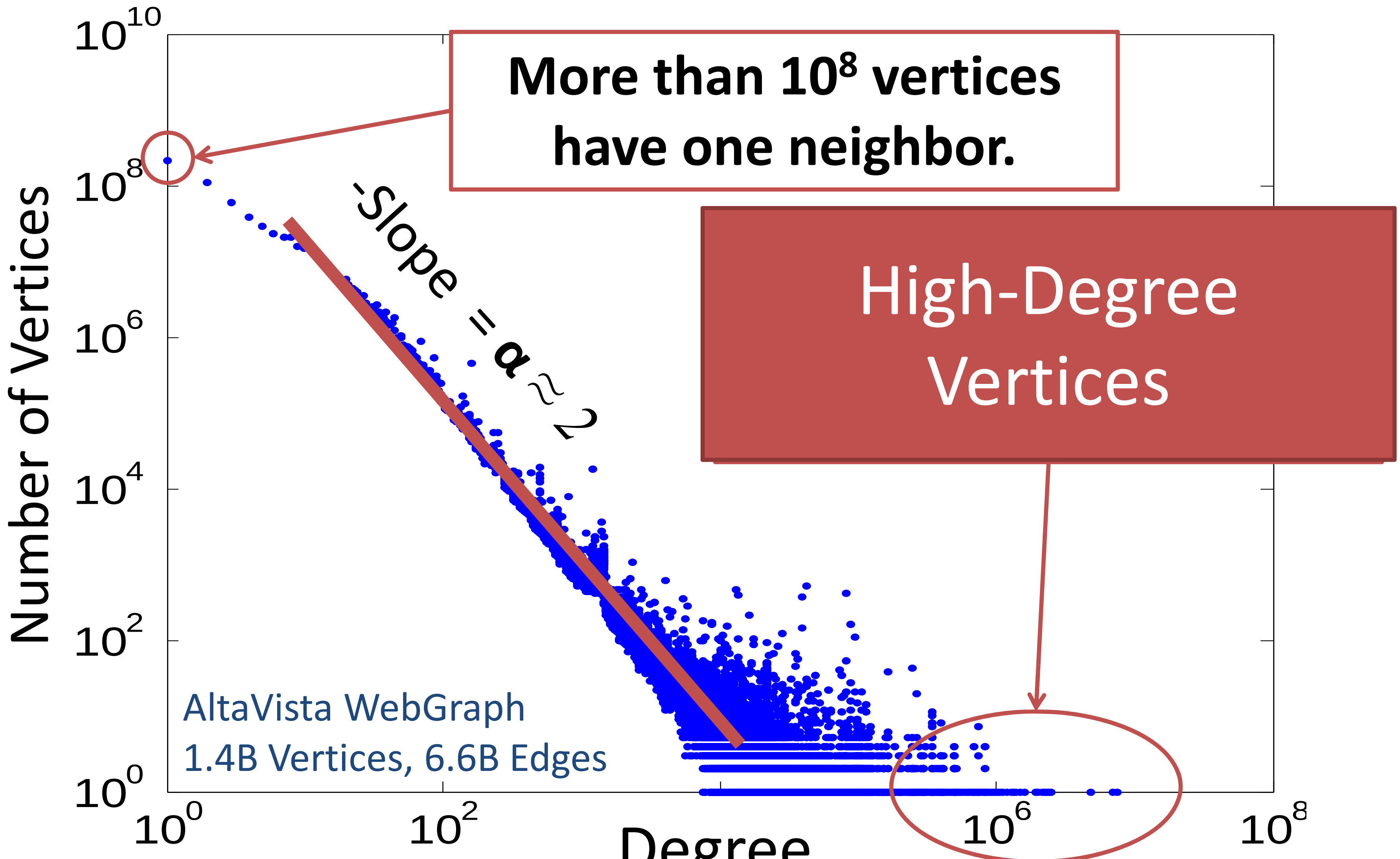


Regular Mesh

Natural Graph

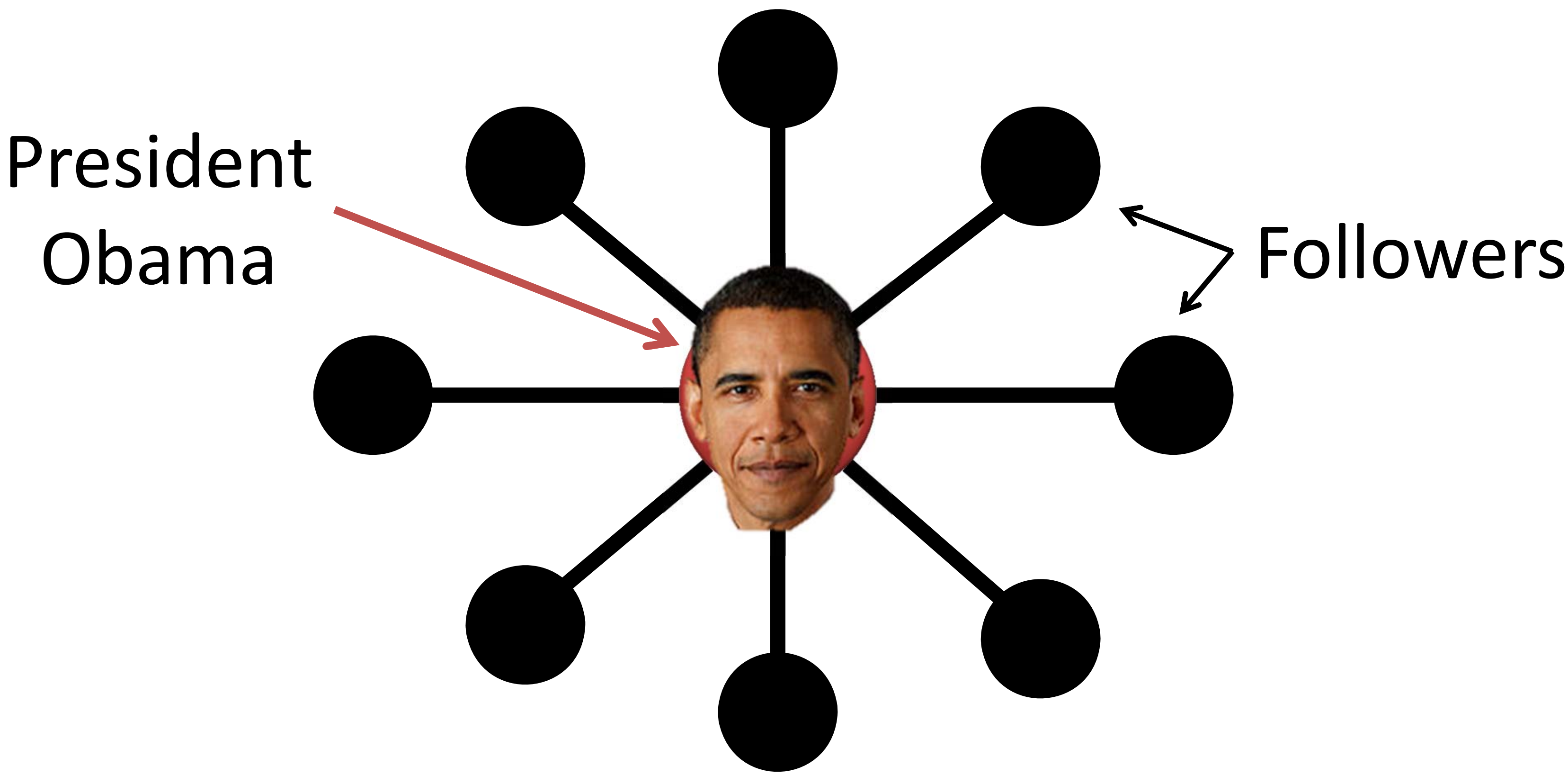
Power-Law Degree Distribution

Power-Law Degree Distribution

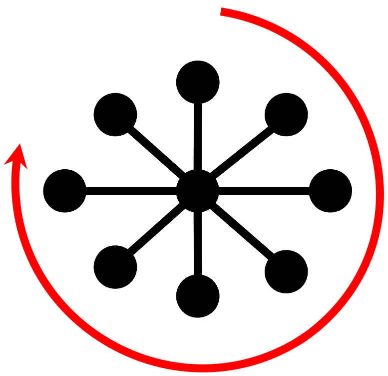


Power-Law Degree Distribution

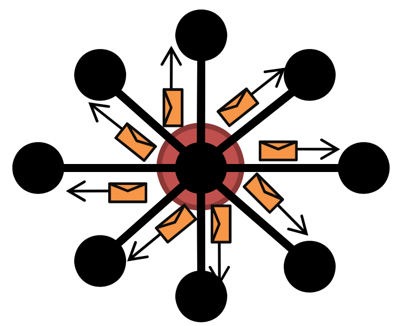
“Star Like” Motif



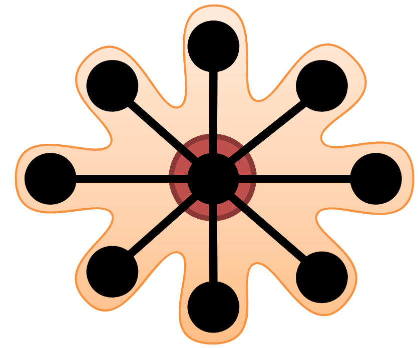
Challenges of **High-Degree** Vertices



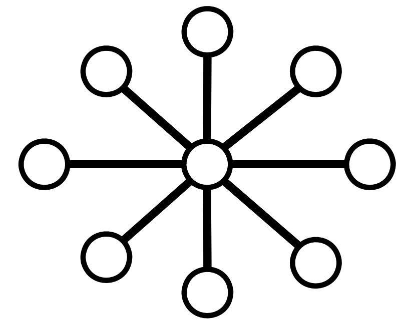
Sequentially process edges



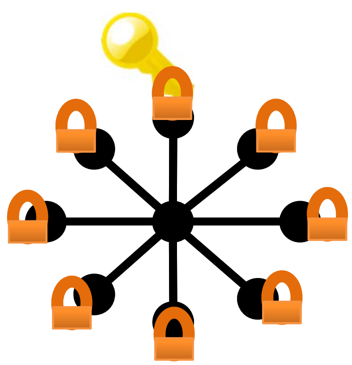
Sends many messages (Pregel)



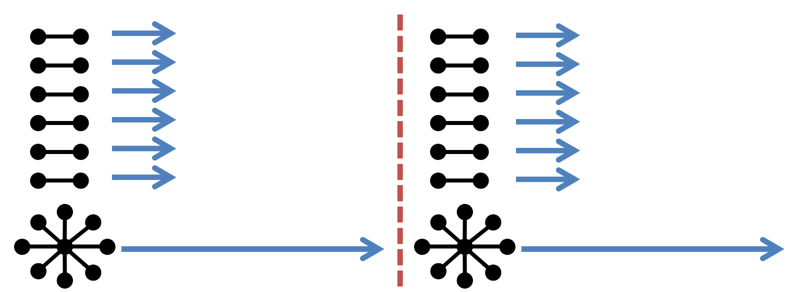
Touches a large fraction of graph (GraphLab)



Edge meta-data too large for single machine



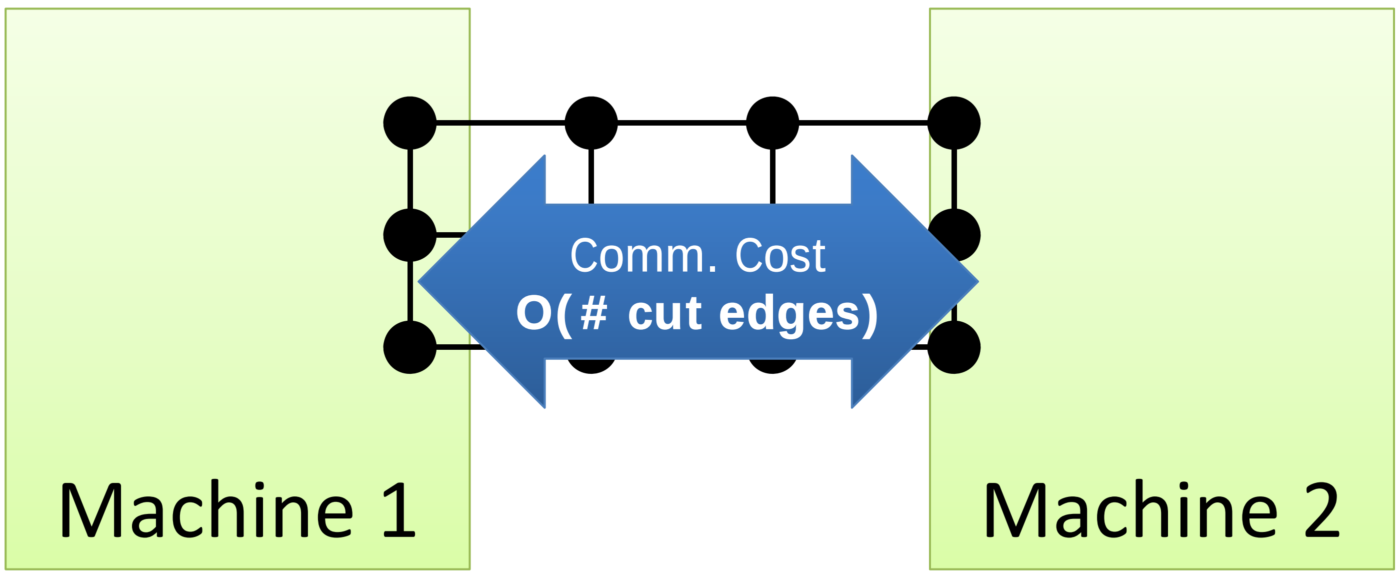
Asynchronous Execution requires heavy locking (GraphLab)



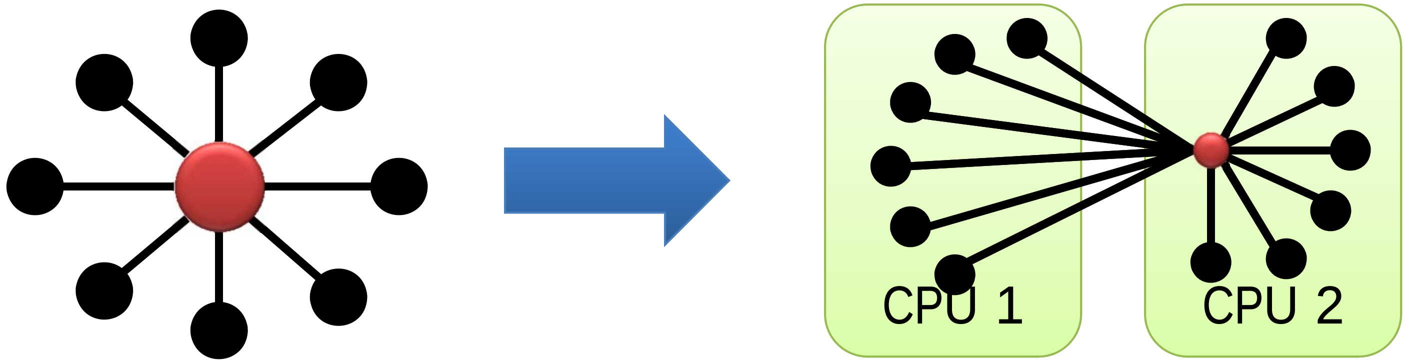
Synchronous Execution prone to stragglers (Pregel)

Graph Partitioning

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



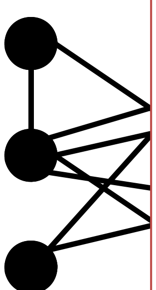
Power-Law Graphs are **Difficult to Partition**



- 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]

Random Partitioning

- GraphLab resorts to **random** (hashed) partitioning on **natural graphs**

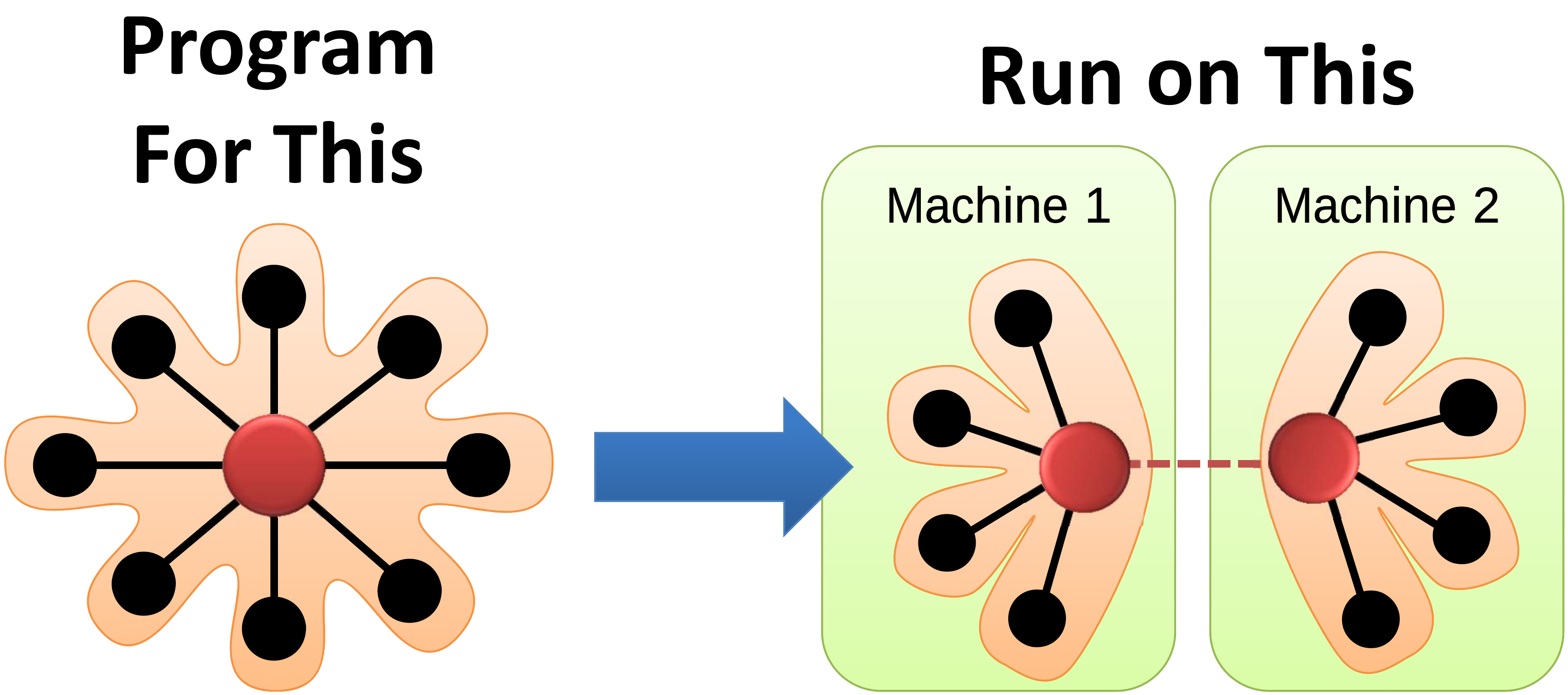


$$\mathbb{E}\left[\frac{|Edges\ Cut|}{|E|}\right] = 1 - \frac{1}{p}$$

10 Machines → 90% of edges cut

100 Machines → 99% of edges cut!

GraphLab2



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

A **Common Pattern** for Vertex-Programs

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

Gather Information About Neighborhood

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

Update Vertex

```
  // Trigger neighbors to run again
  priority = |R[i] – oldR[i]|
  if R[i] not converged then
    signal neighbors(i) with priority
```

Signal Neighbors & Modify Edge Data

Formal GraphLab2 Semantics

- **Gather**(SrcV, Edge, DstV) → A
  - Collect information from neighbors
- **Sum**(A, A) → A
  - Commutative associative Sum
- **Apply**(V, A) → V
  - Update the vertex
- **Scatter**(SrcV, Edge, DstV) → (Edge, signal)
  - Update edges and signal neighbors

PageRank in **GraphLab2**

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

GraphLab2_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) :

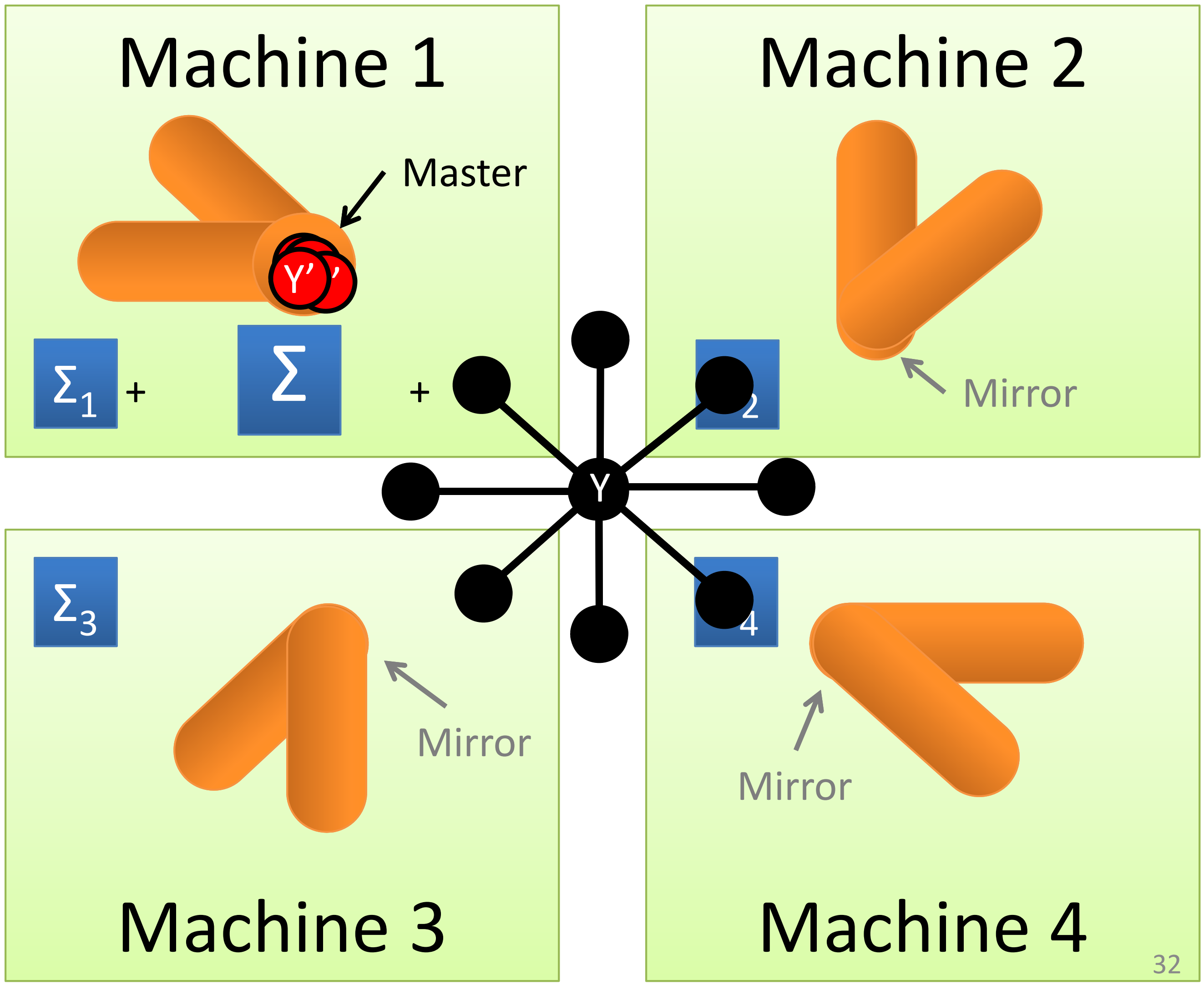
if *R[i]* changed then trigger *j* to be **recomputed**

GAS Decomposition

Gather

Apply

Scatter



Minimizing Communication in PowerGraph

New Theorem:

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

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

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 two **distributed** approaches:
  - ***Random*** *Vertex Cut*
  - ***Greedy*** *Vertex Cut*

Random Vertex-Cut

- Randomly assign edges to machines

Machine 1

Machine 2

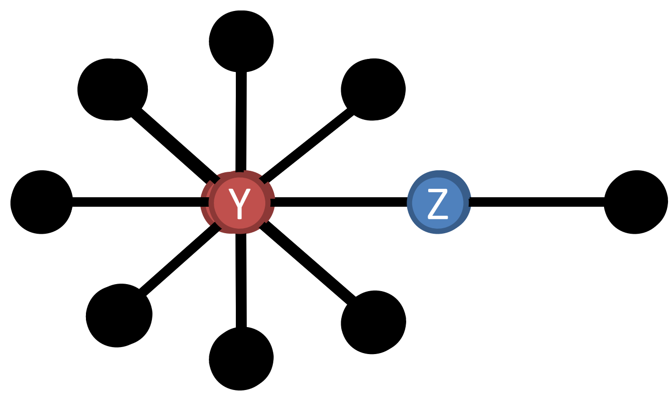
Machine 3

Balanced Vertex-Cut

Y Spans 3 Machines

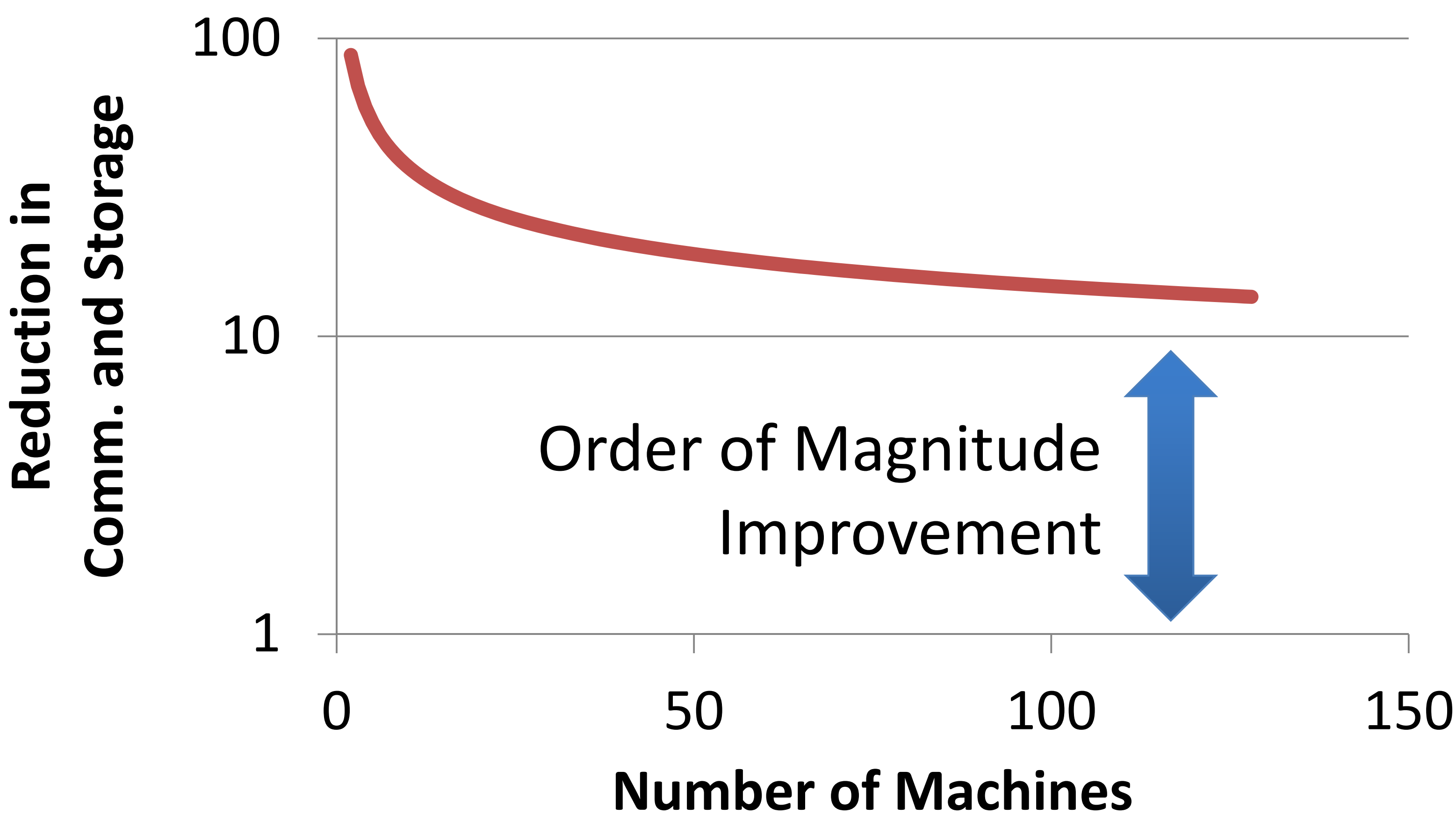
Z Spans 2 Machines

Not cut!



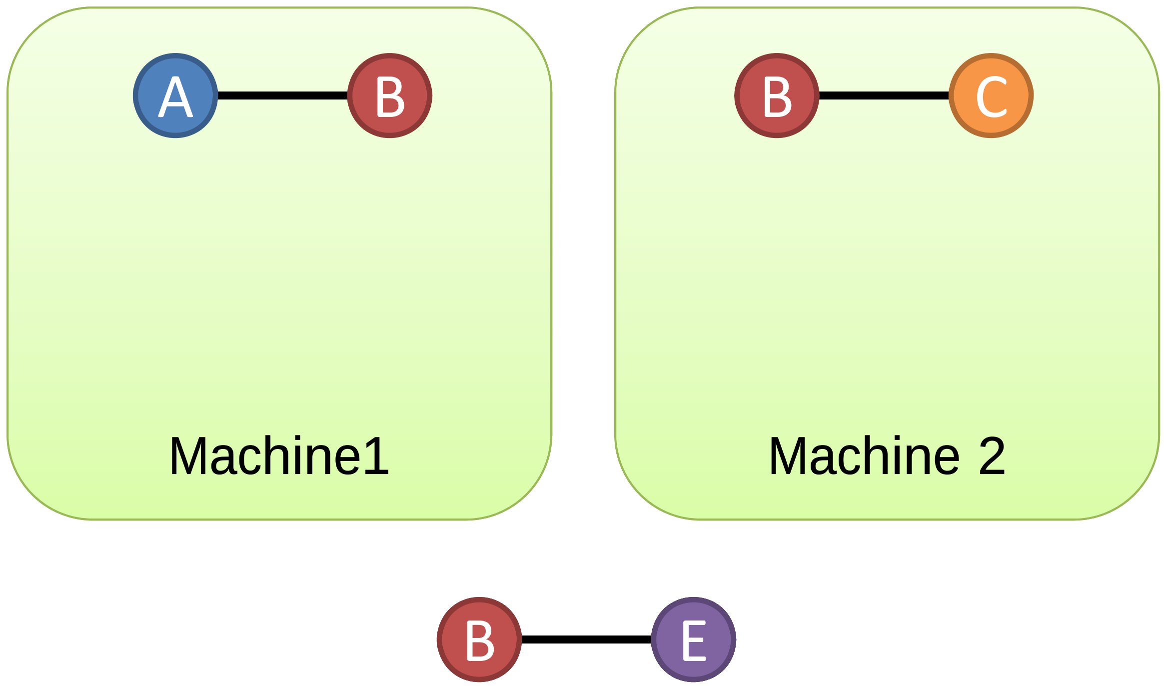
Random Vertex-Cuts vs. Edge-Cuts

- Expected improvement from vertex-cuts:



Streaming Greedy Vertex-Cuts

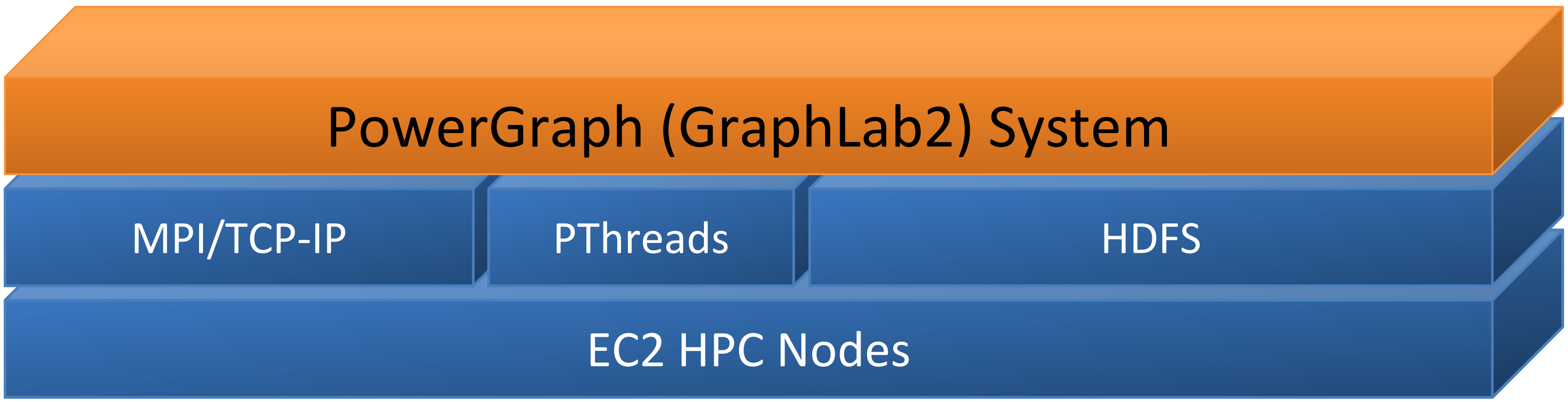
- Place edges on machines which already have the vertices in that edge.



Greedy Vertex-Cuts Improve Performance

Greedy partitioning improves computation performance.

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

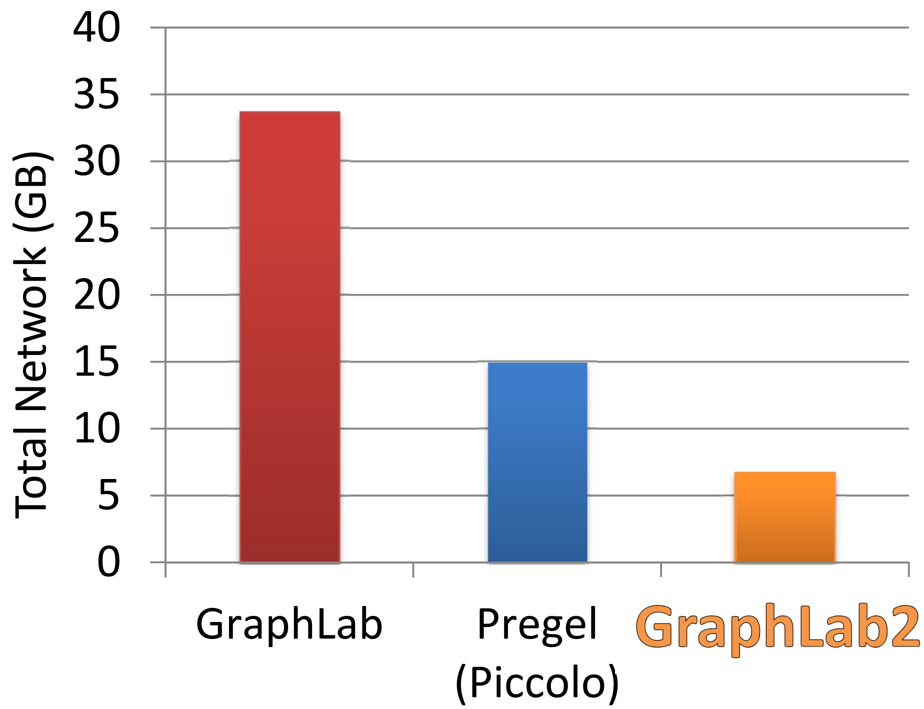
Implemented Many Algorithms

- **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

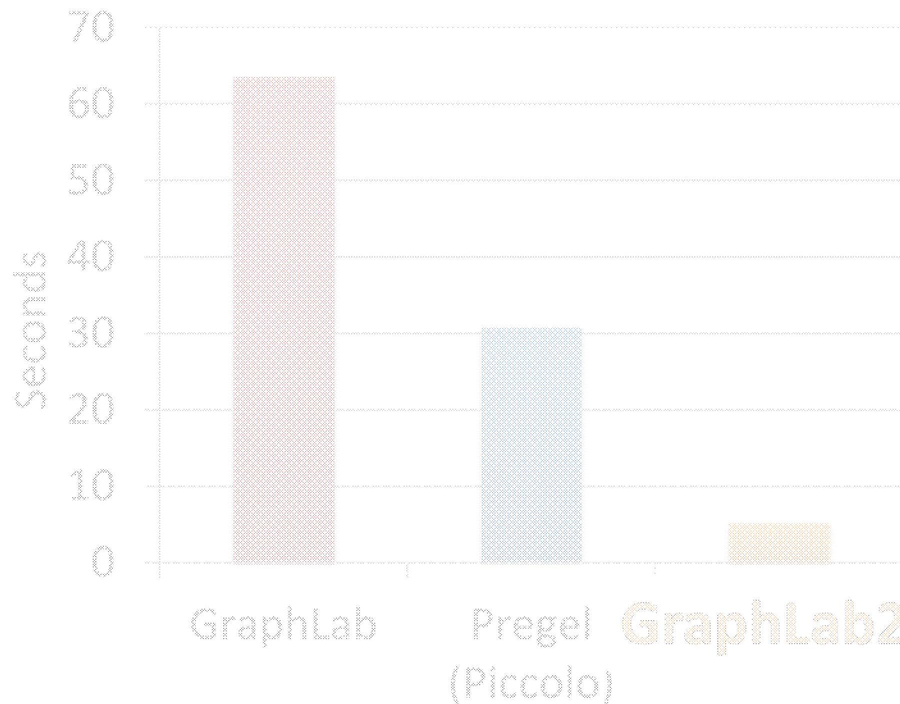
PageRank on the Twitter Follower Graph

Natural Graph with 40M Users, 1.4 Billion Links

Communication



Runtime



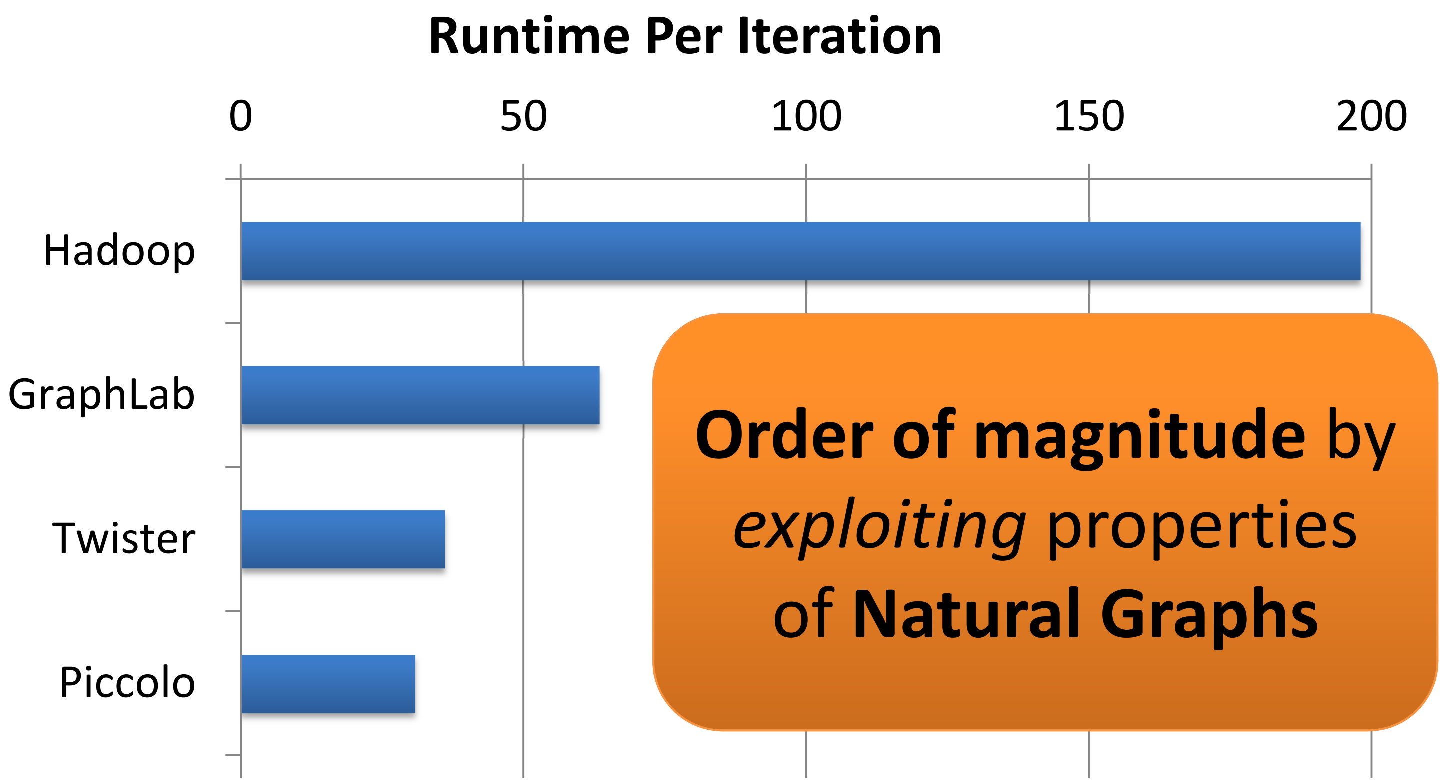
Reduces Communication

Runs Faster

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

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]

GraphLab2 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.

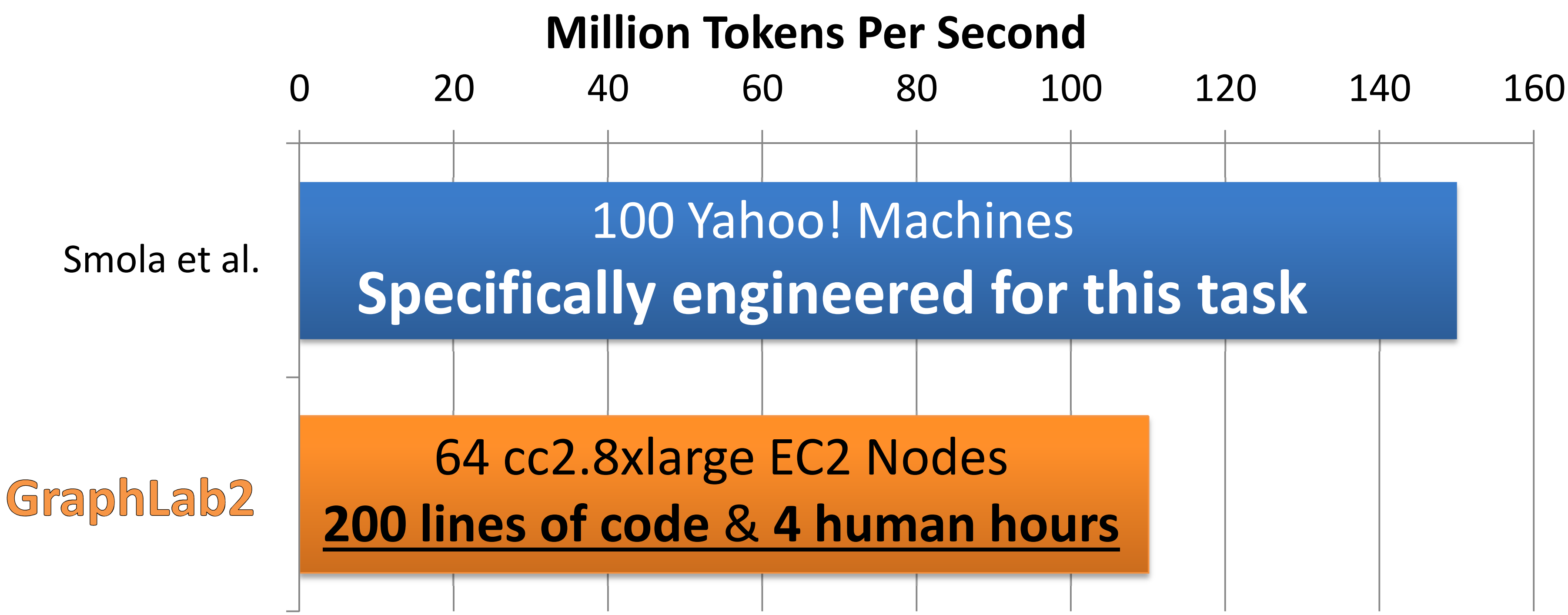
1B links processed per second

64 HPC Nodes

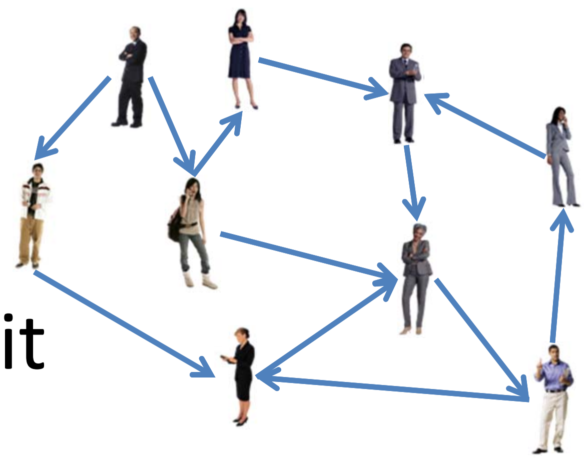
<u>30 lines of user code</u>

Topic Modeling

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

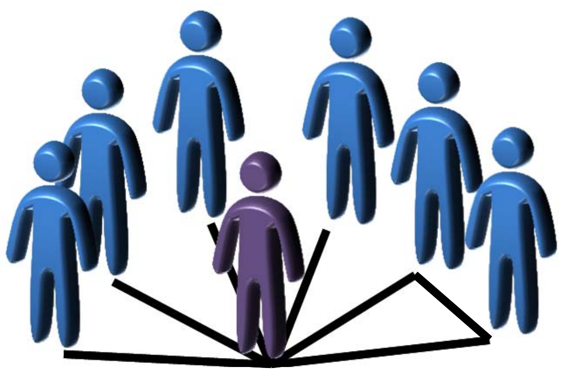


Triangle Counting

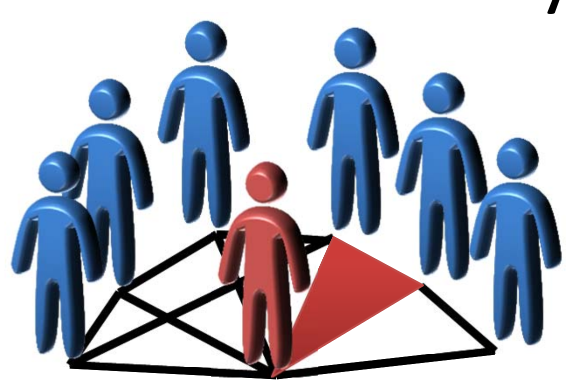


- For each vertex in graph, count number of triangles containing it

- Measures both "popularity" of the vertex and "cohesiveness" of the vertex's community:



Fewer Triangles
Weaker Community

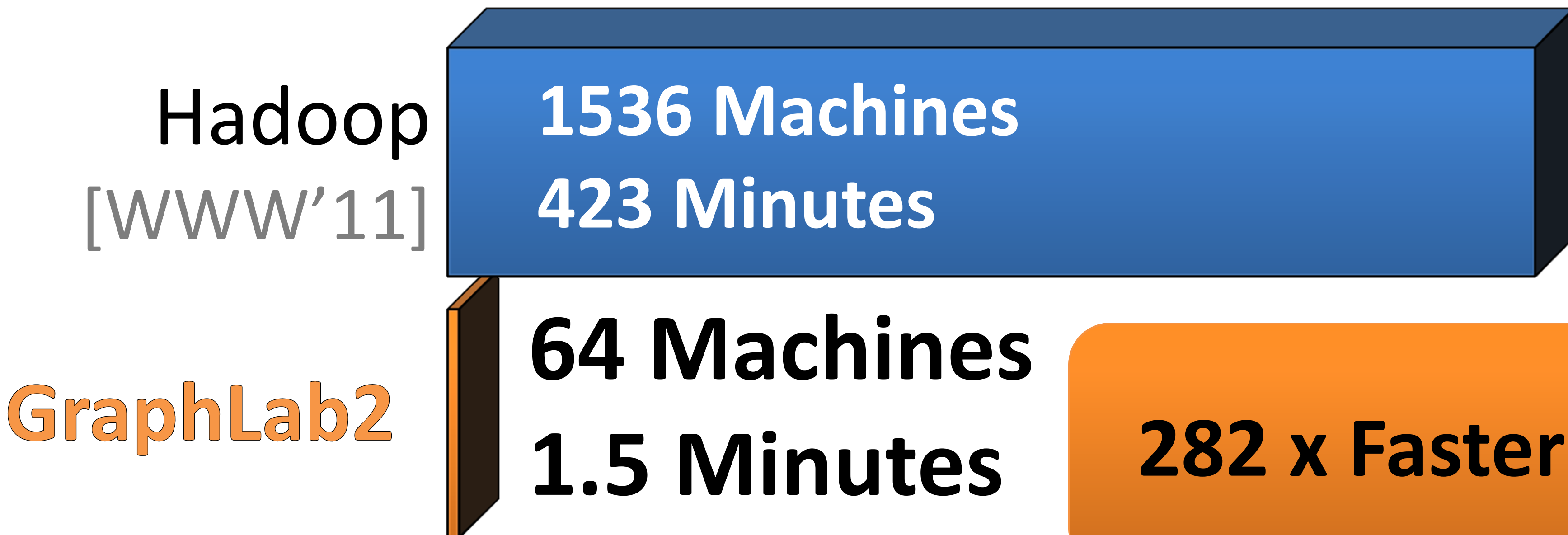


More Triangles
Stronger Community

Triangle Counting on The Twitter Graph

Identify individuals with **strong communities.**

Counted: 34.8 Billion Triangles



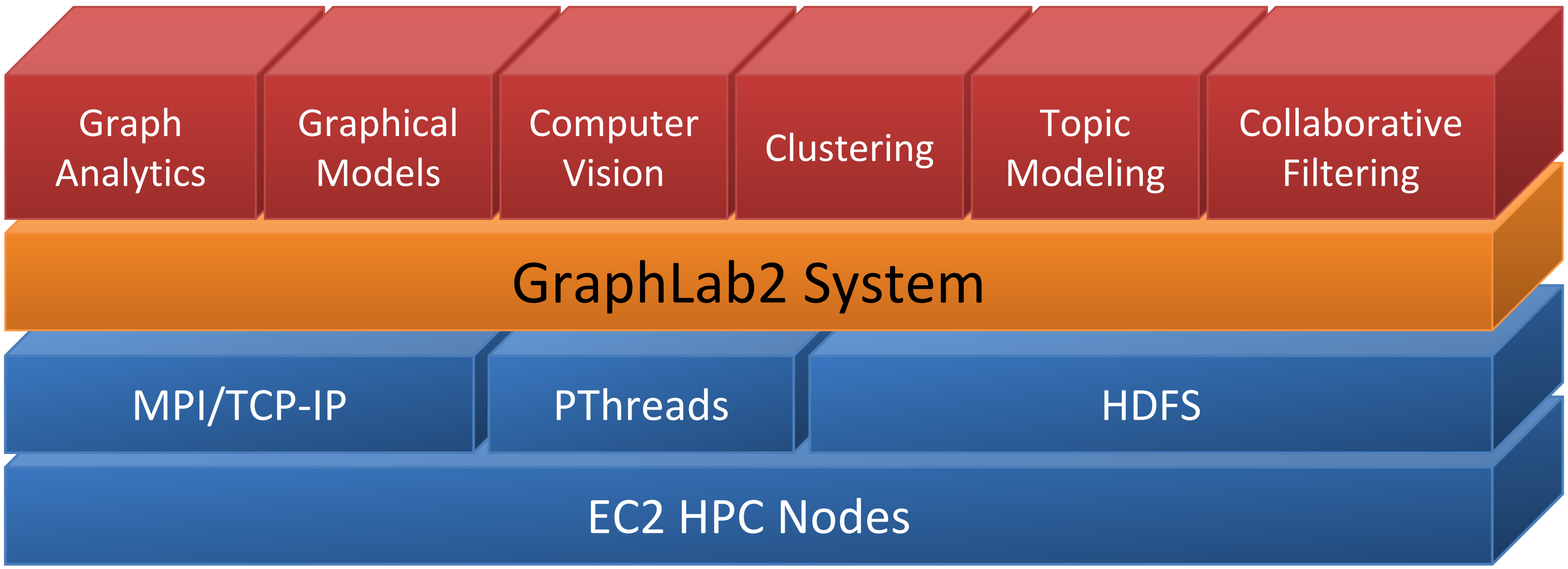
282 x Faster

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

Machine Learning and **Data-Mining** Toolkits



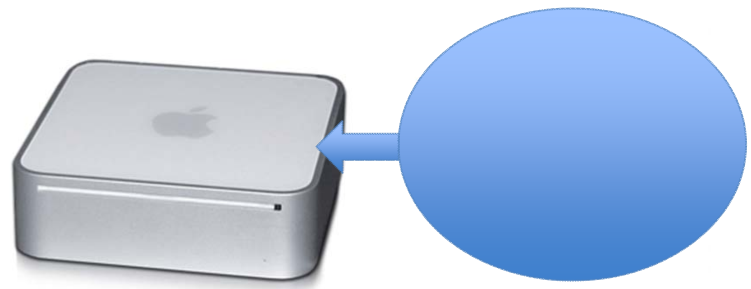
http://graphlab.org

Apache 2 License

GraphChi: Going small with GraphLab

GraphLab

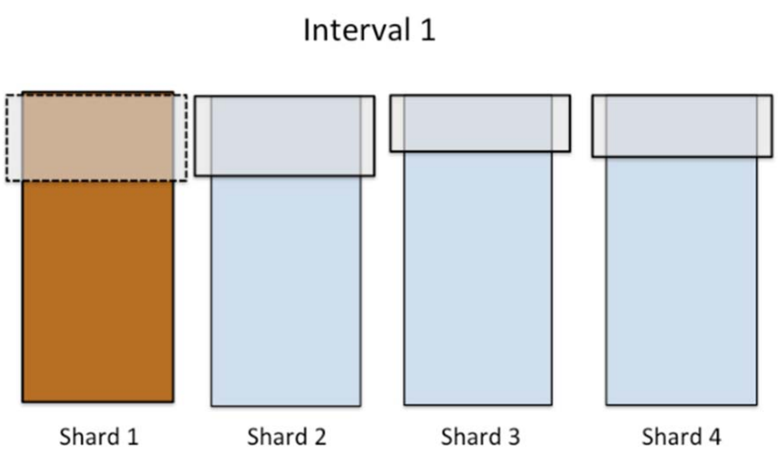
Solve huge problems on small or embedded devices?



Key: Exploit non-volatile memory (starting with SSDs and HDs)

GraphChi – disk-based GraphLab

Novel Parallel Sliding Windows algorithm

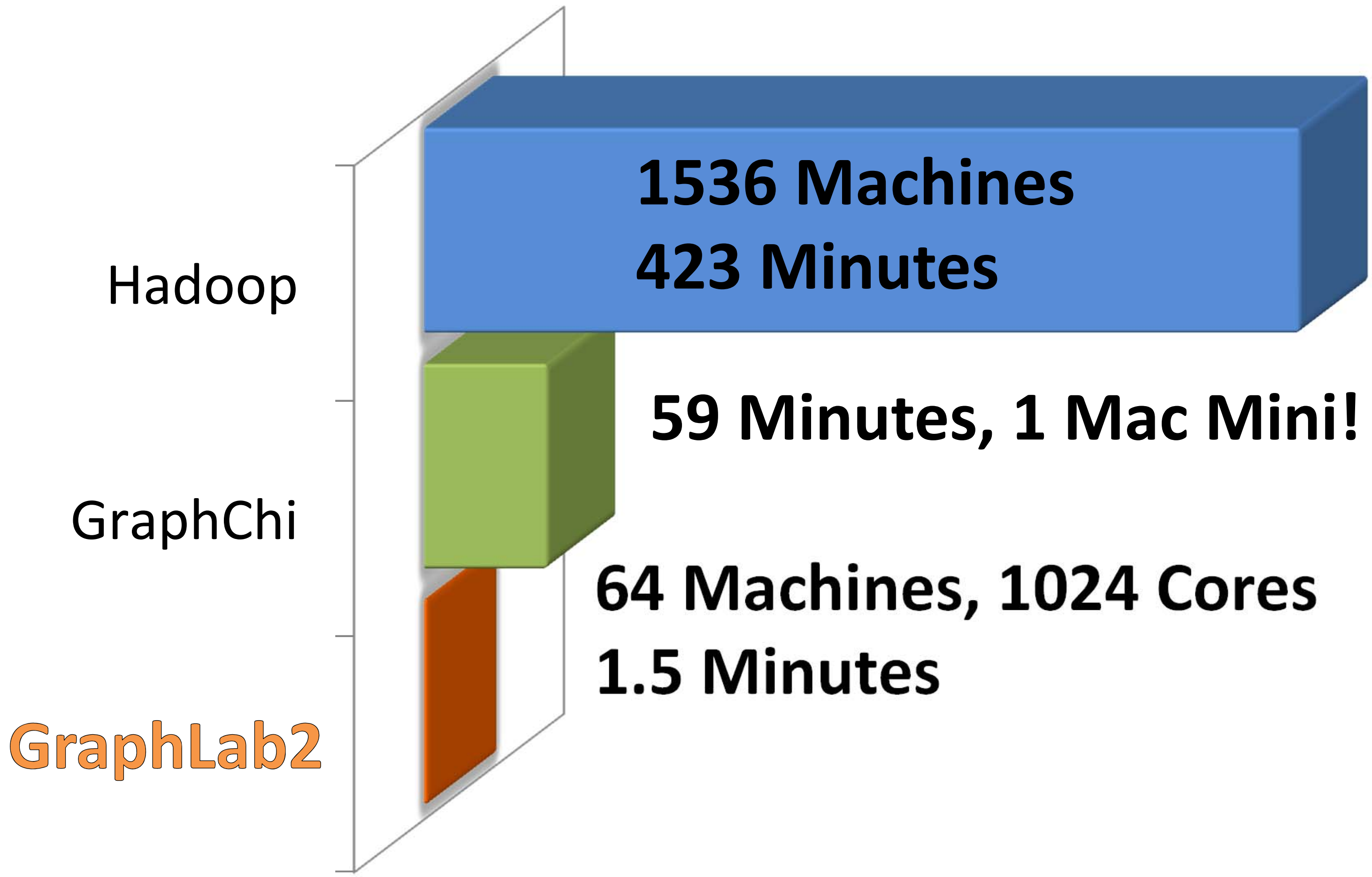


- Single-Machine
  - Parallel, asynchronous execution
- Solves big problems
  - That are normally solved in cloud
- Efficiently exploits disks
  - Optimized for stream acces
  - Efficient on *both* SSD and hard-drives

Triangle Counting in Twitter Graph

40M Users
1.2B Edges

Total: 34.8 Billion Triangles



Hadoop results from [Suri & Vassilvitskii '11]

GraphLab2

Apache 2 License

http://graphlab.org

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

Active Work

- Cross language support (Python/Java)
- Support for incremental graph computation
- Integration with Graph Databases
- Declarative representations of GAS decomposition:
  - my.pr := nbrs.in.map(x => x.pr).reduce((a,b) => a + b)

GraphLab 2

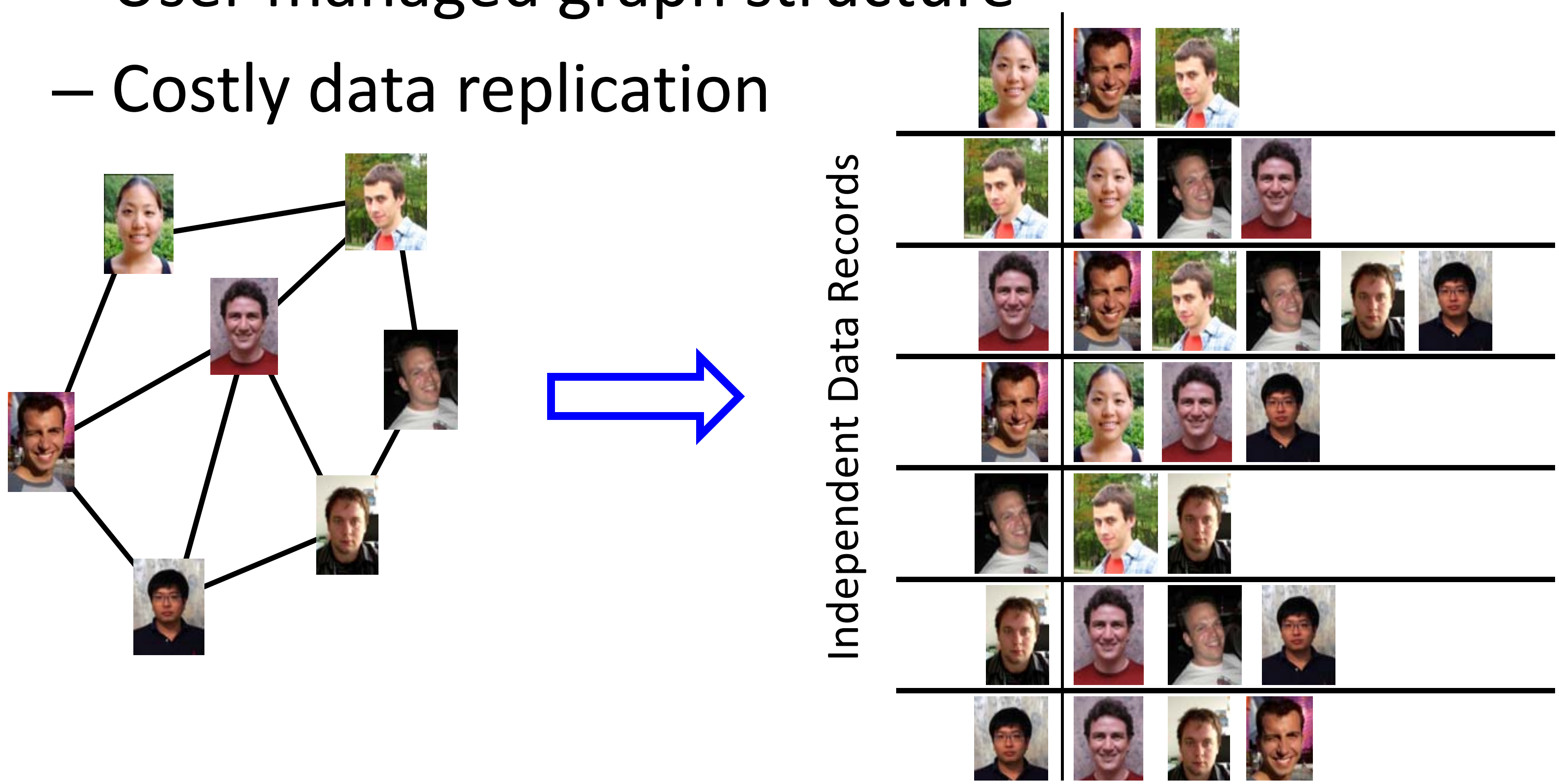
http://graphlab.org

Joseph E. Gonzalez
Postdoc, UC Berkeley
jegonzal@eecs.berkeley.edu
jegonzal@cs.cmu.edu

Why not use *Map-Reduce* for **Graph Parallel** algorithms?

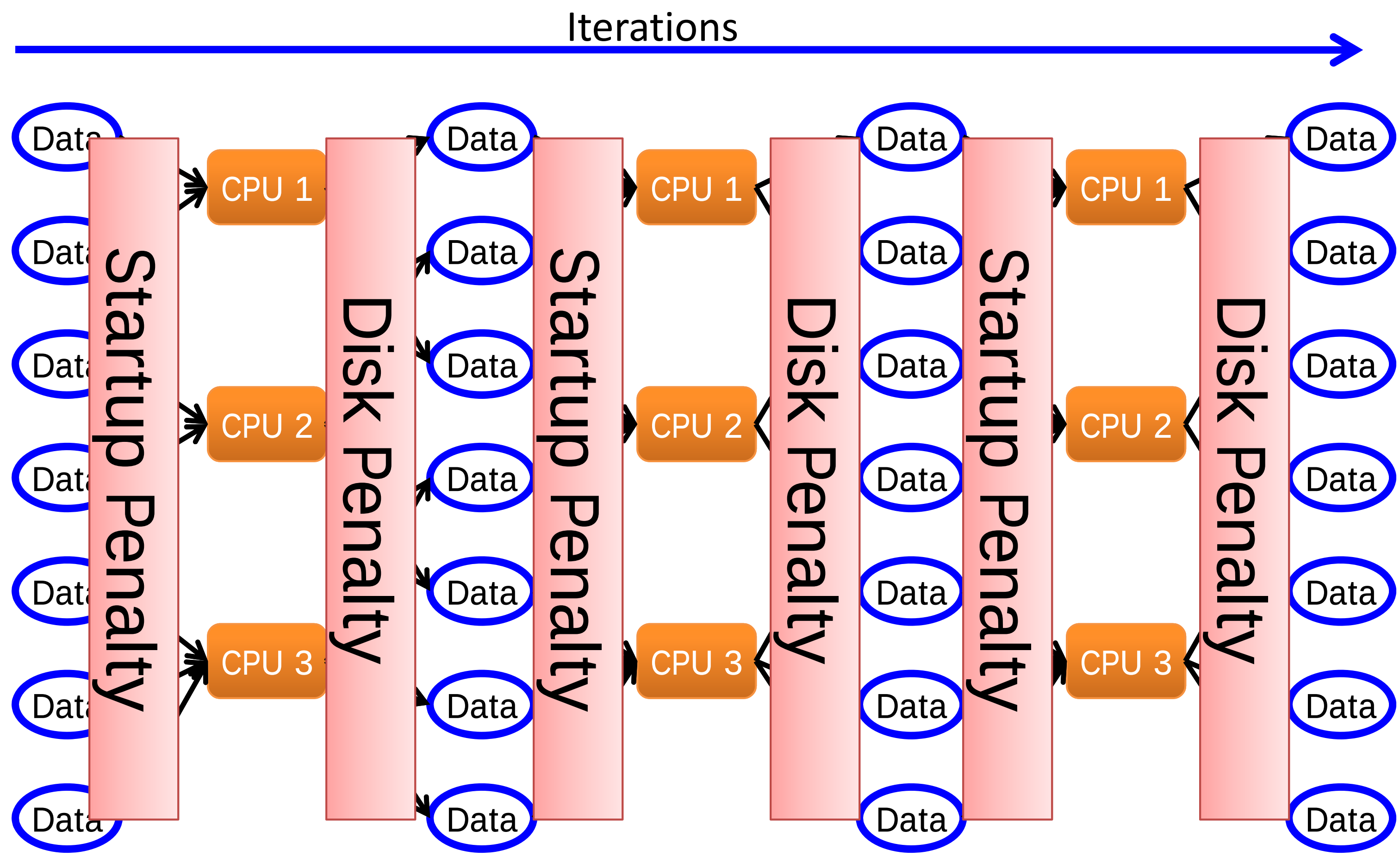
Data Dependencies are Difficult

- Difficult to express dependent data in Map Reduce
  - Substantial data transformations
  - User managed graph structure
  - Costly data replication



Iterative Computation is Difficult

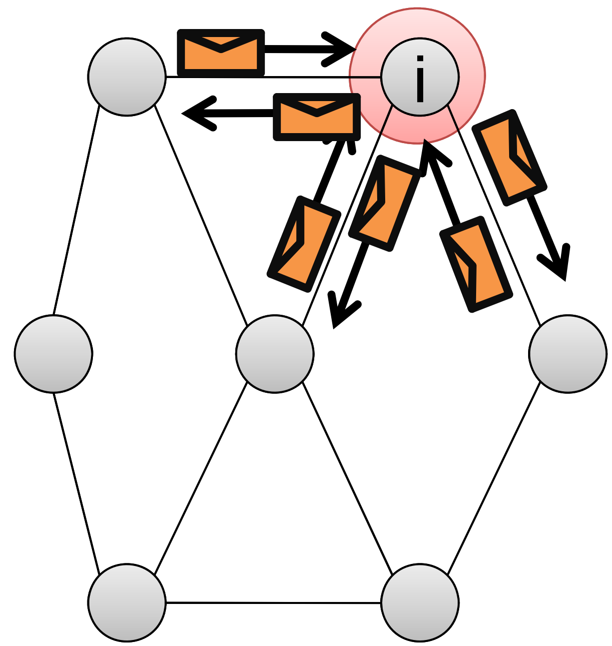
- System is not optimized for iteration:



The Pregel Abstraction

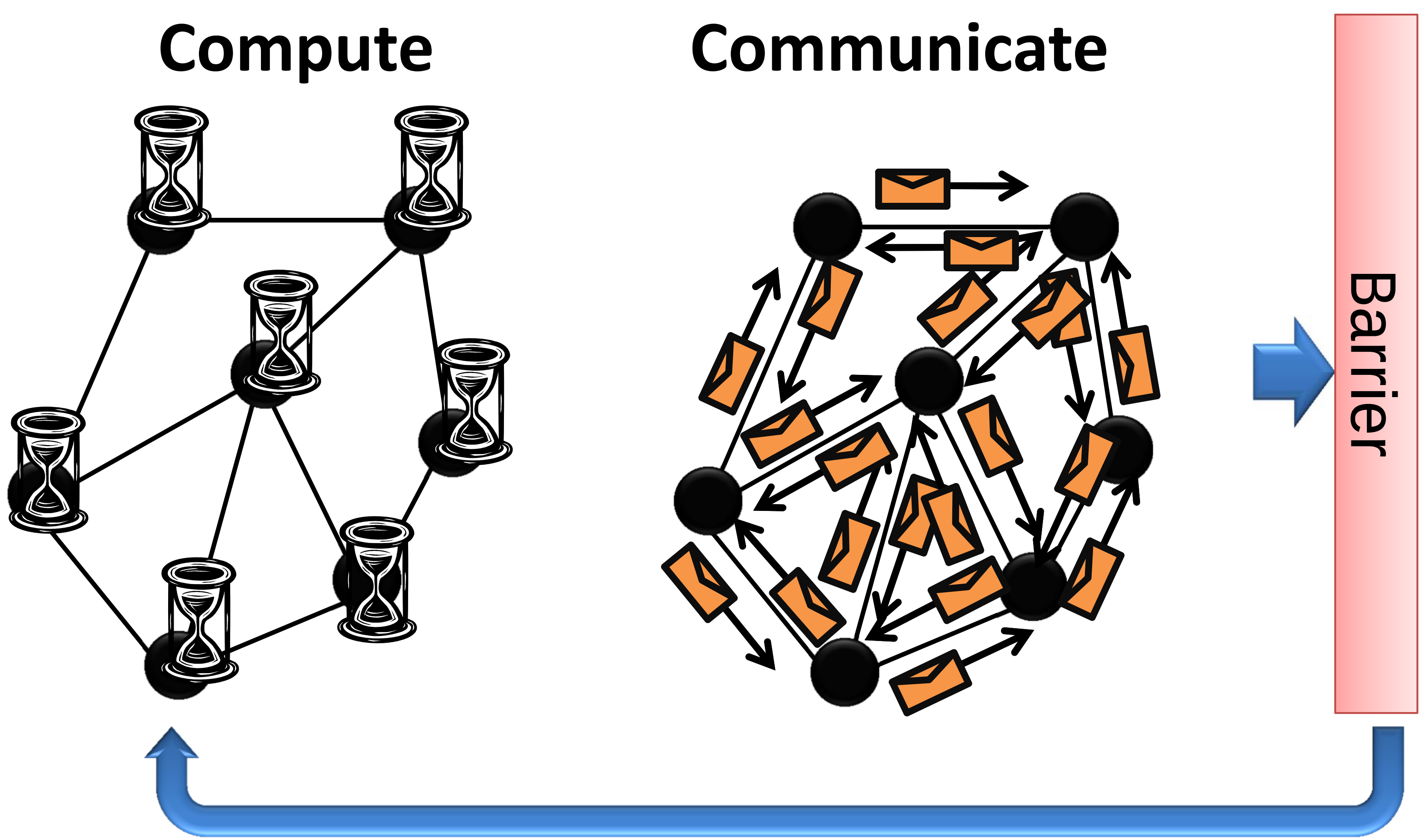
Vertex-Programs interact by sending **messages**.

```
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] = total

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



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

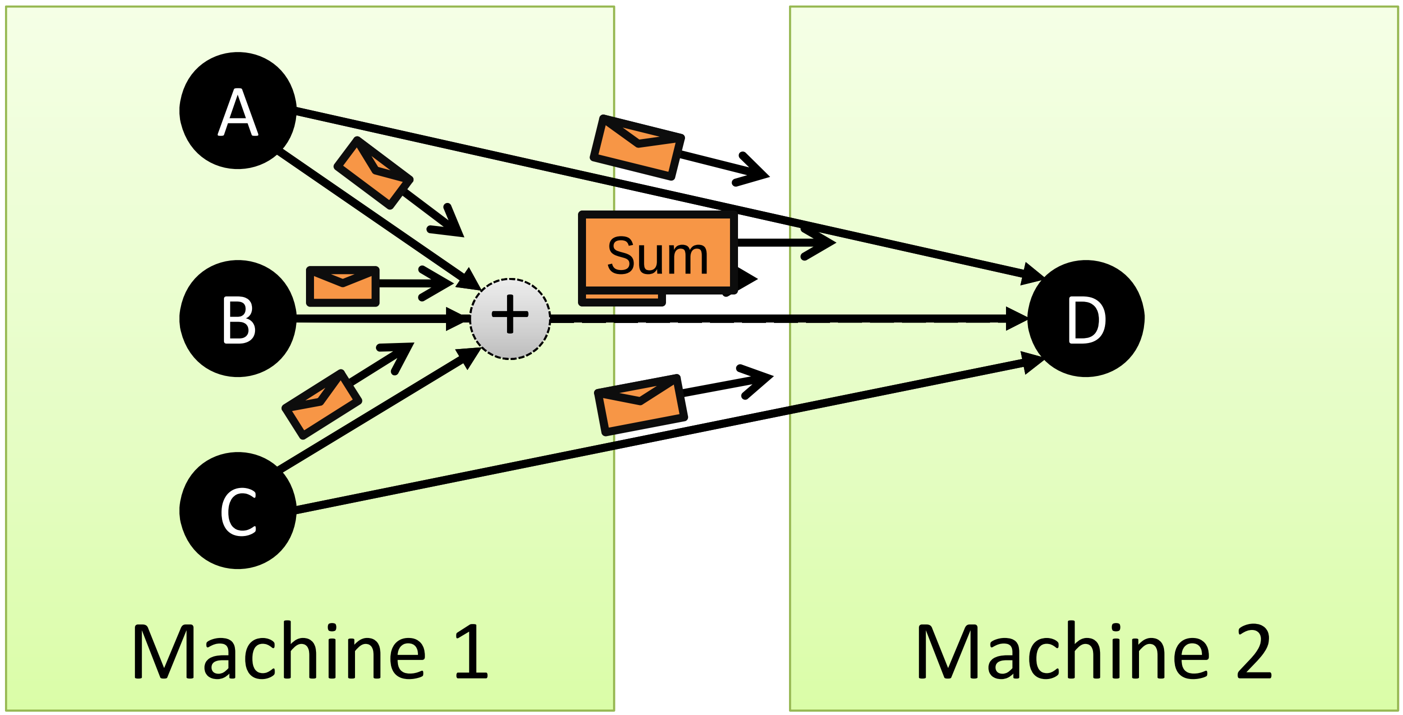
Pregel **Synchronous** Execution



Communication Overhead for High-Degree Vertices

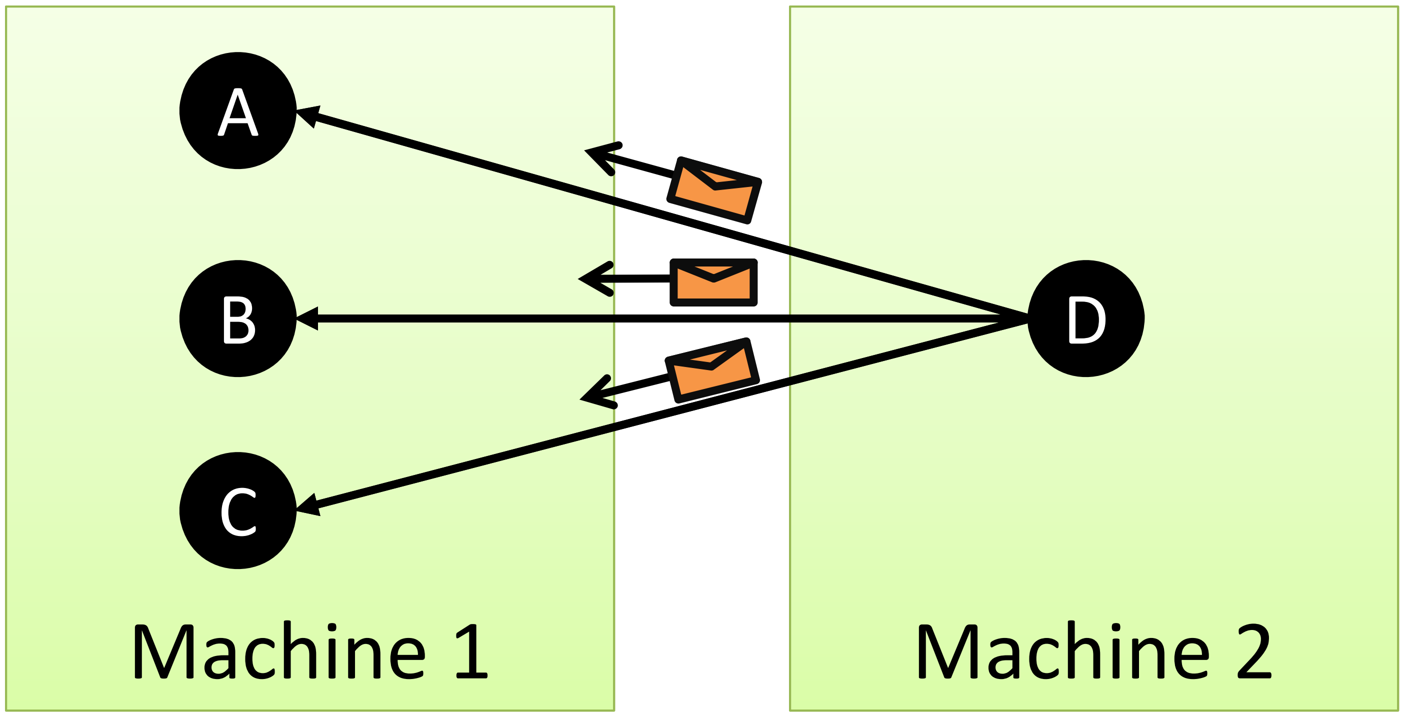
Fan-In vs. Fan-Out

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



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

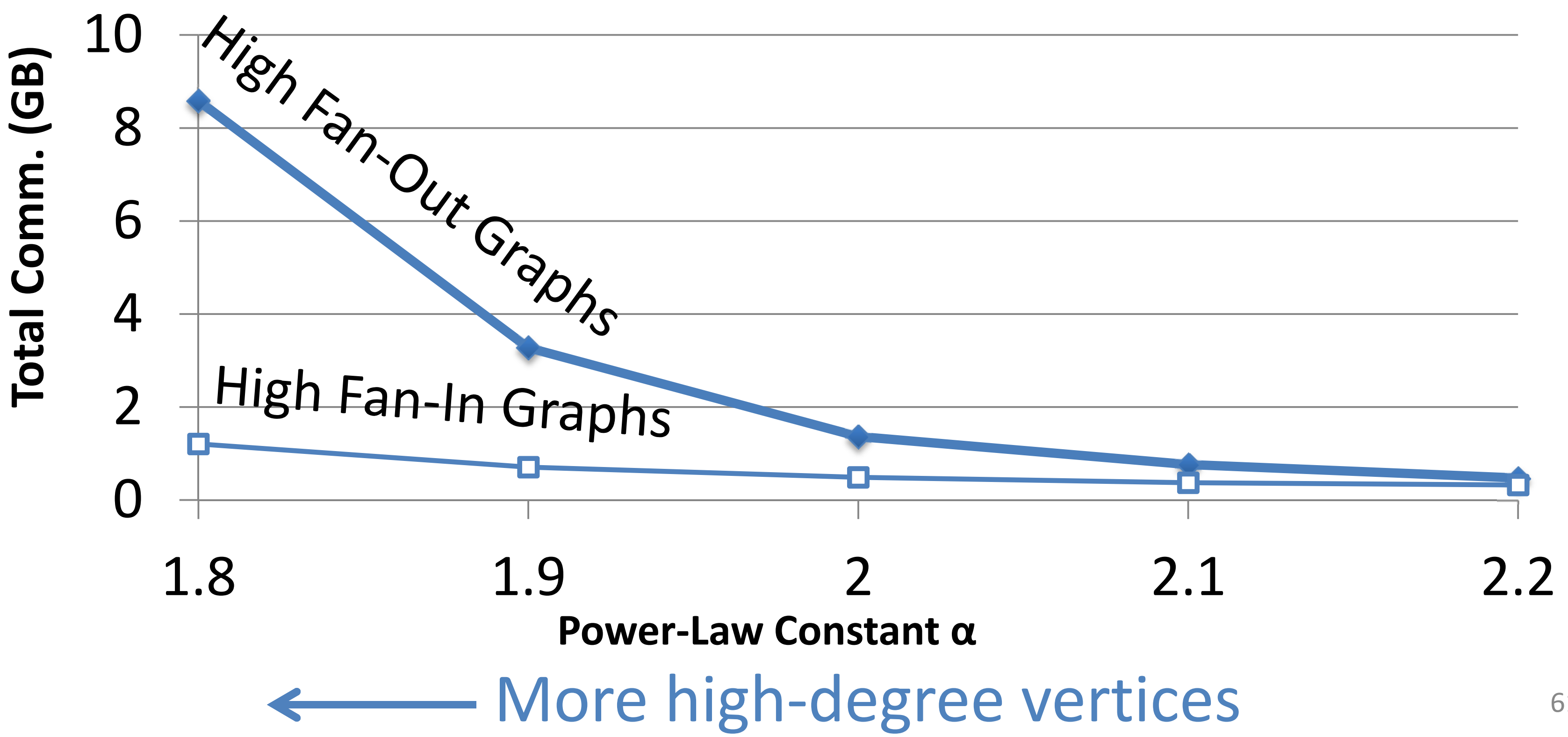
Pregel Struggles with **Fan-Out**



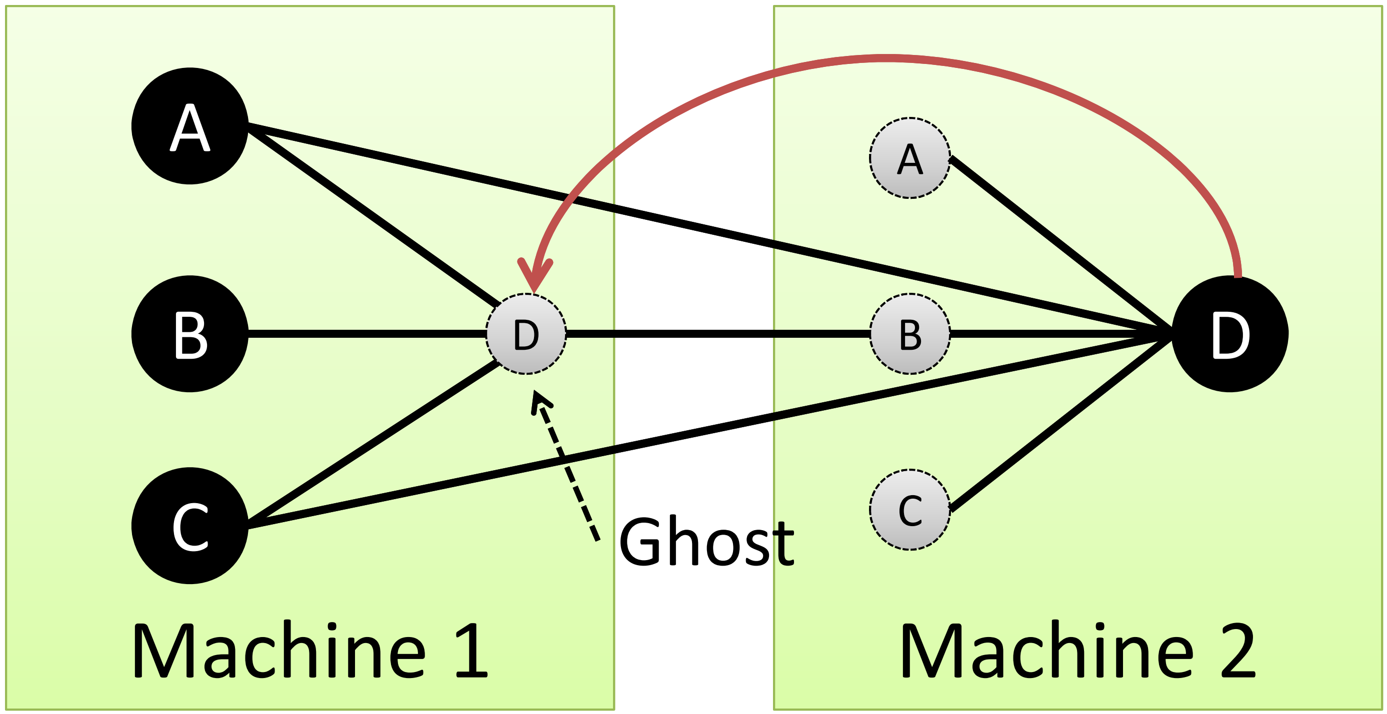
- **Broadcast** sends many copies of the same message to the same machine!

Fan-In and Fan-Out Performance

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

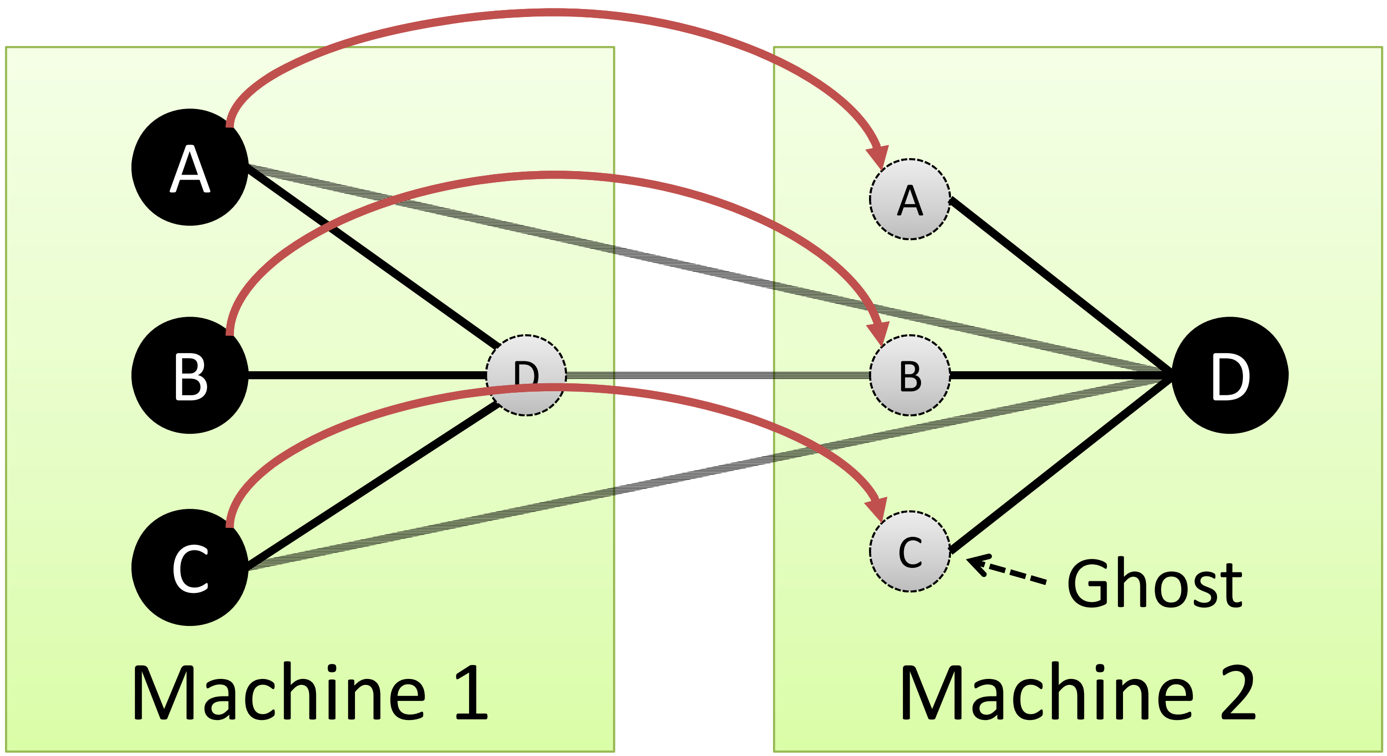


GraphLab Ghosting



- Changes to master are synced to ghosts

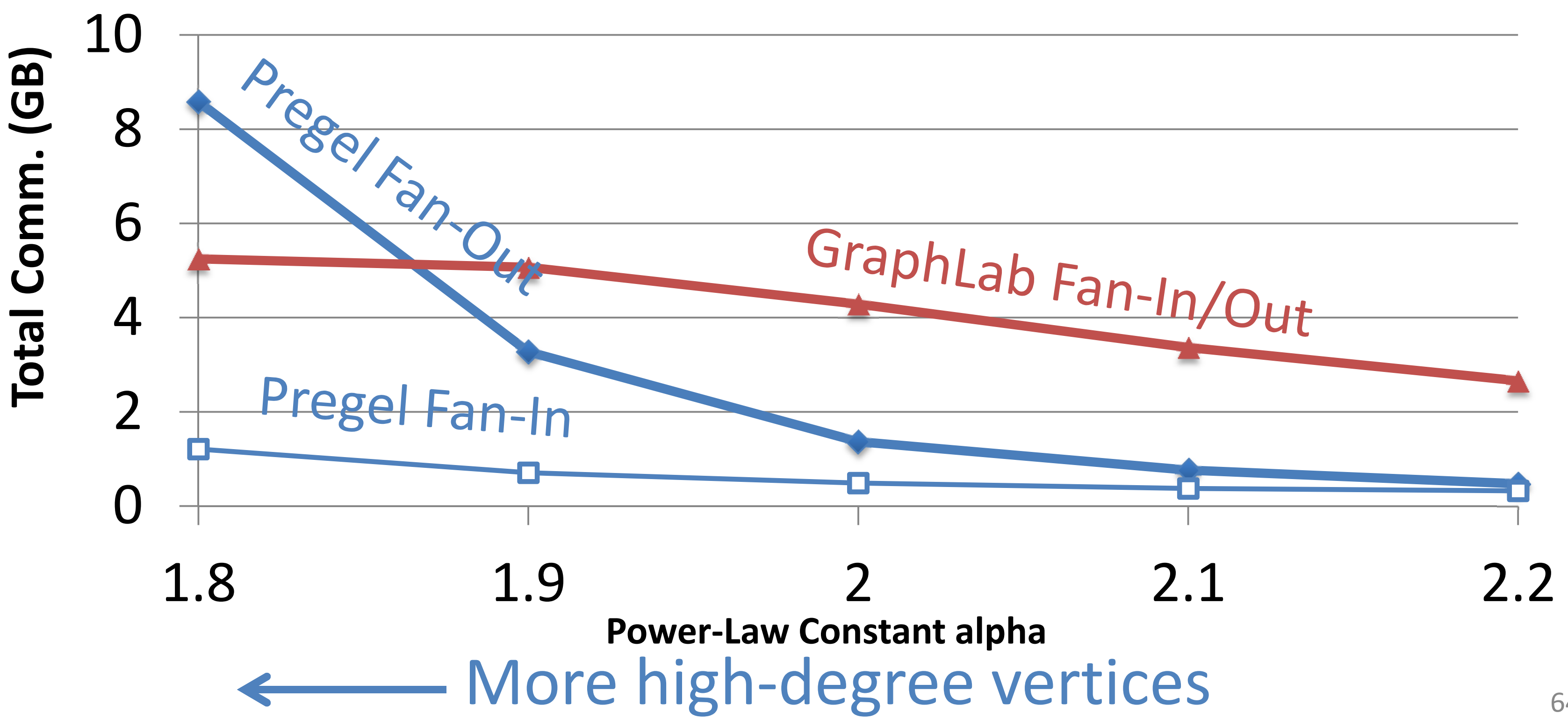
GraphLab Ghosting



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

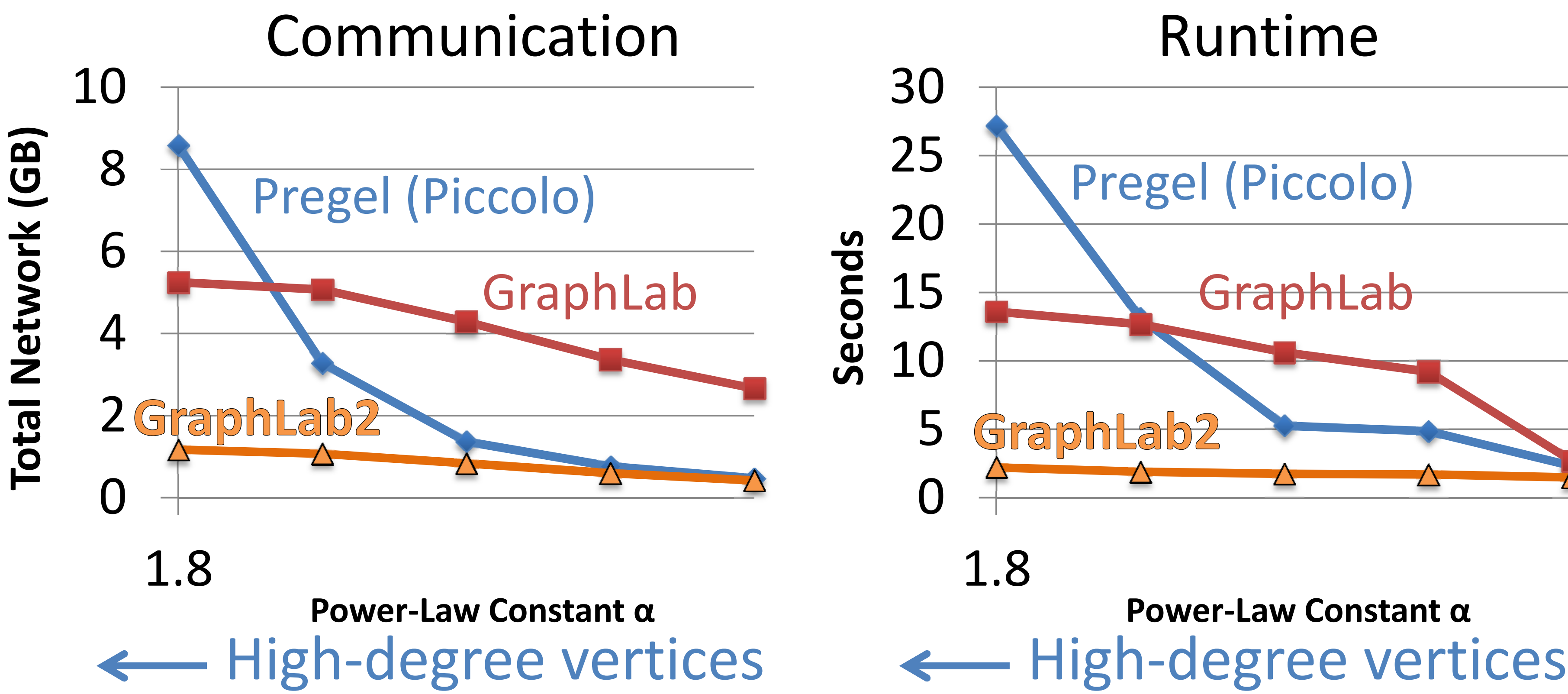
Fan-In and Fan-Out Performance

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



Comparison with GraphLab & Pregel

- PageRank on Synthetic Power-Law Graphs:



GraphLab2 is robust to **high-degree** vertices.

GraphLab

GraphLab on Spark

```
#include <graphlab.hpp>

struct vertex_data : public graphlab::IS_POD_TYPE { float rank;
  vertex_data() : rank(1) { }
};

typedef graphlab::empty edge_data;
typedef graphlab::distributed_graph<vertex_data, edge_data> graph_type;
class pagerank :
  public graphlab::ivertex_program<graph_type, float>,
  public graphlab::IS_POD_TYPE {
  float last_change;
public:
  float gather(icontext_type& context, const vertex_type& vertex,
               edge_type& edge) const {
    return edge.source().data().rank / edge.source().num_out_edges();
  }

  void apply(icontext_type& context, vertex_type& vertex,
             const gather_type& total) {
    const double newval = 0.15*total + 0.85;
    last_change = std::fabs(newval - vertex.data().rank);
    vertex.data().rank = newval;
  }

  void scatter(icontext_type& context, const vertex_type& vertex,
               edge_type& edge) const {
    if (last_change > TOLERANCE) context.signal(edge.target());
  }
};

struct pagerank_writer {
  std::string save_vertex(graph_type::vertex_type v) {
    std::stringstream strm;
    strm << v.id() << "\t" << v.data() << "\n";
    return strm.str();
  }
  std::string save_edge(graph_type::edge_type e) { return ""; }
};

int main(int argc, char** argv) {
  graphlab::mpi_tools::init(argc, argv);
  graphlab::distributed_control dc;

  graphlab::command_line_options clopts("PageRank algorithm.");
  graph_type graph(dc, clopts);
  graph.load_format("biggraph.tsv", "tsv");

  graphlab::omni_engine<pagerank> engine(dc, graph, clopts);
  engine.signal_all();
  engine.start();

  graph.save(saveprefix, pagerank_writer(), false, true false);

  graphlab::mpi_tools::finalize();
  return EXIT_SUCCESS;
}
```

```
import spark.graphlab._

val sc = spark.SparkContext(master, "pagerank")

val graph = Graph.textFile("bigGraph.tsv")
val vertices = graph.outDegree().mapValues((_, 1.0, 1.0))

val pr = Graph(vertices, graph.edges).iterate(
  (meId, e) => e.source.data._2 / e.source.data._1,
  (a: Double, b: Double) => a + b,
  (v, accum) => (v.data._1, (0.15 + 0.85*a), v.data._2),
  (meId, e) => abs(e.source.data._2-e.source.data._1)>0.01)

pr.vertices.saveAsTextFile("results")
```

Interactive!