Graph-parallel Computation

Rong Chen

Institute of Parallel and Distributed Systems

Shanghai Jiao Tong University

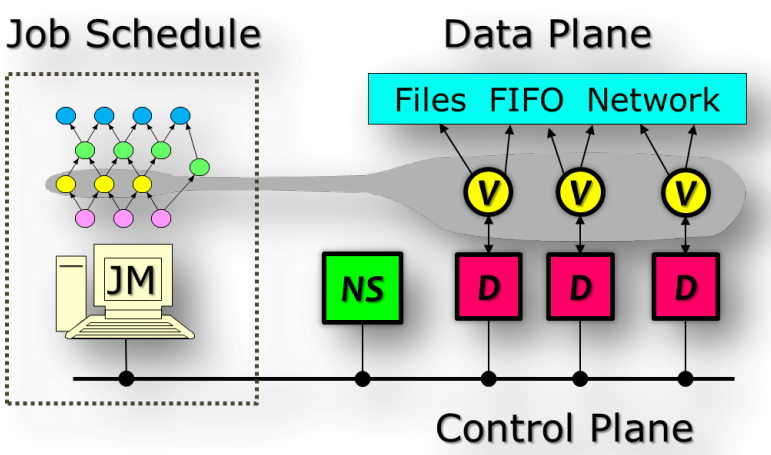
http://ipads.se.sjtu.edu.cn/rong_chen

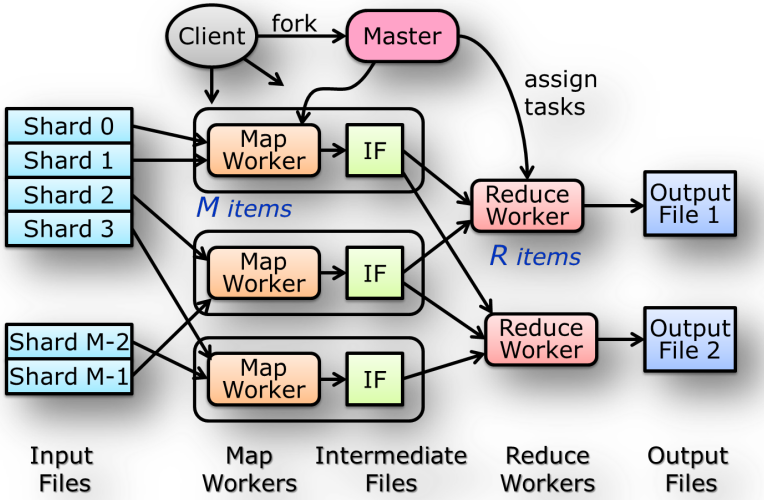
Credits: some slides and figures from J. Gonzalez (CMU) , Y. Low (CMU), G. Malewicz (Google), A. Khan (ETH) and S. Elnikety (MSR)

Programming Model so far

Data-parallel Programming Model

- MapReduce & Dryad
  - Parallelism
  - Load balance
  - Locality and data distribution
  - Fault tolerance





Big Data already Happened

1 Billion Tweets Per Week

750 Million Facebook Users

6 Billion Flickr Photos

48 Hrs of Video Per Minute

Big Learning

The New York Times

SundayReview

WORLD U.S. N.Y. / REGION BUSINESS TI

NEWS ANALYSIS

The Age of Big Data

By STEVE LOHR

Published: February 11, 2012

"...growing at 50 percent a year..."

"... data a new class of economic asset, like currency or gold."

How do we understand and use Big Data?

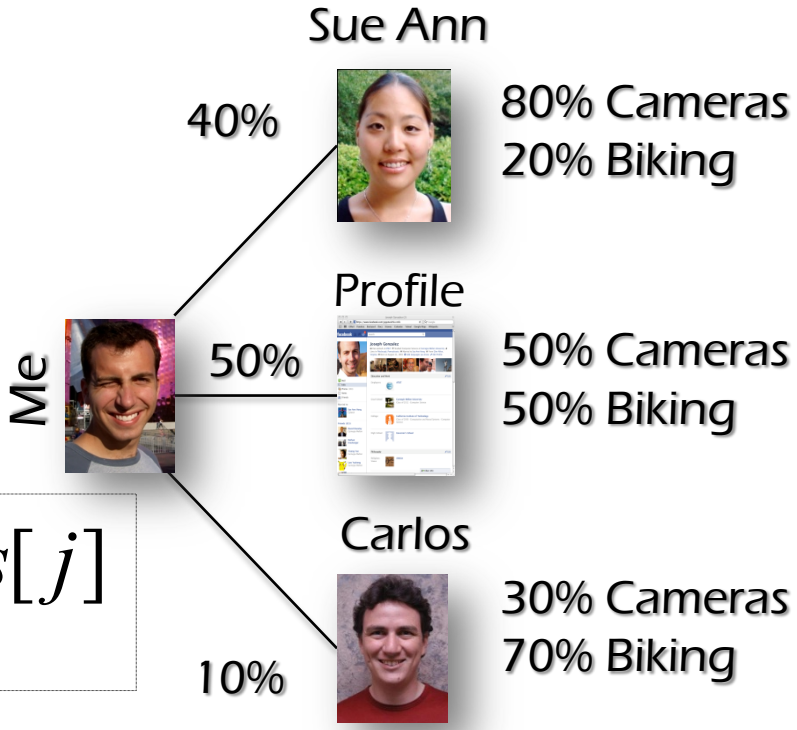
Big Learning Example

Social Arithmetic

50% What I list on my profile
40% Sue Ann Likes
\+ 10% Carlos Like

I Like: 60% Cameras, 40% Biking

Label Propagation



Recurrence Algorithm

$$Likes[i] = \sum_{j \in Friends[i]} W_{ij} \times Likes[j]$$

iterate until convergence

Parallelism: Compute all Likes[i] in parallel

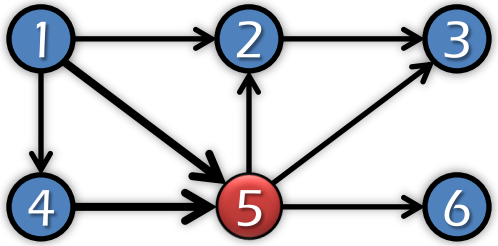
http://www.cs.cmu.edu/~zhuxj/pub/CMU-CALD-02-107.pdf

Big Learning Example

PageRank

http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf

$$R[i] = \alpha + (1-\alpha) \sum_{(j,i) \in E} \frac{1}{L[j]} R[j]$$



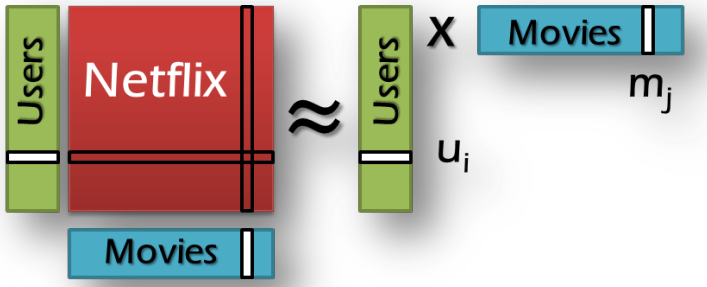
α is the random reset probability

L[j] is the number of links on page j

Alternating Least Squares

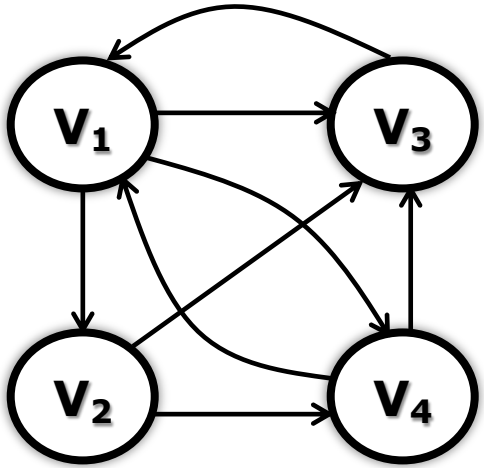
$$u_i = \arg\min_w \sum_{j \in N[i]} (r_{ij} - m_j \cdot w)^2$$

$$m_j = \arg\min_w \sum_{i \in N[j]} (r_{ij} - u_i \cdot w)^2$$



http://dl.acm.org/citation.cfm?id=1424269

Example: PageRank

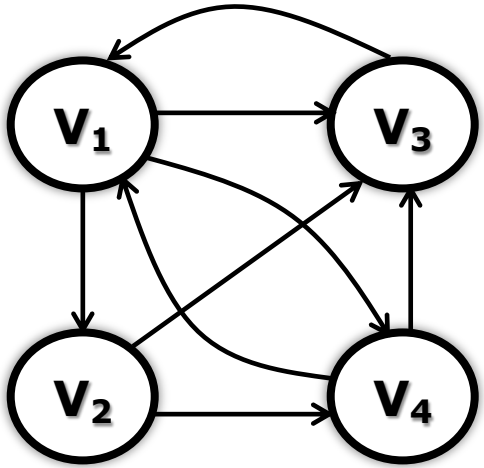


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

- PR(u): Page Rank of node u
- F_u: Out-neighbors of node u

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank

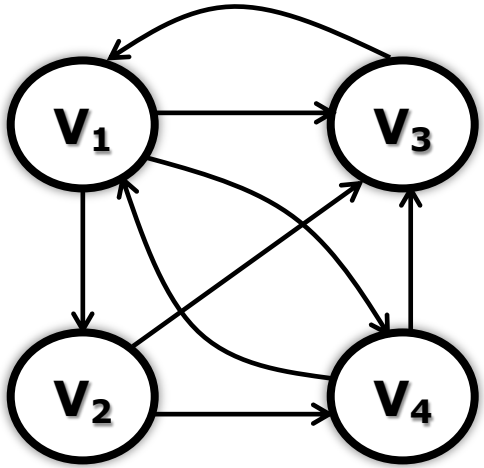


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0
PR(V_1)	0.25
PR(V_2)	0.25
PR(V_3)	0.25
PR(V_4)	0.25

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank

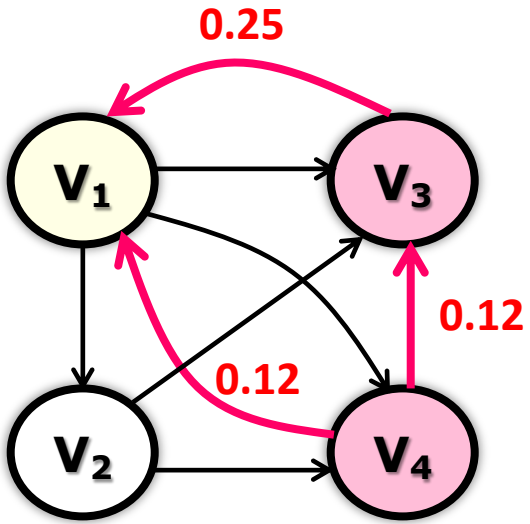


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0	K=1
PR(V_1)	0.25	???
PR(V_2)	0.25	
PR(V_3)	0.25	
PR(V_4)	0.25	

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank

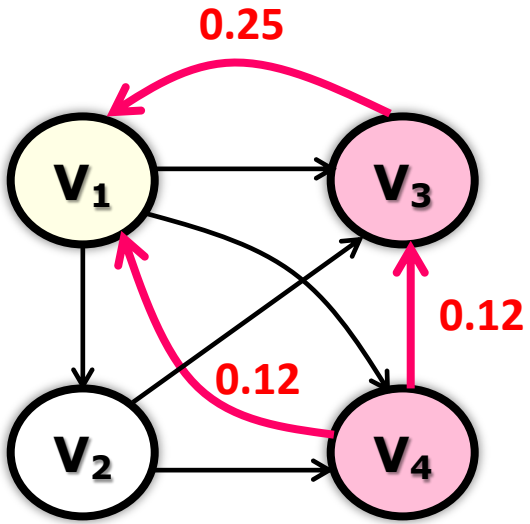


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0	K=1
PR(V_1)	0.25	???
PR(V_2)	0.25	
PR(V_3)	0.25	
PR(V_4)	0.25	

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank

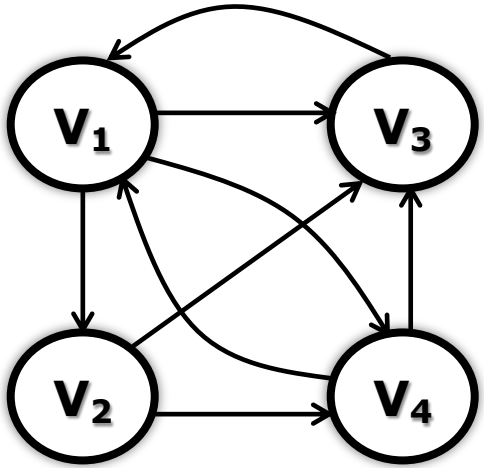


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0	K=1
PR(V_1)	0.25	0.37
PR(V_2)	0.25	
PR(V_3)	0.25	
PR(V_4)	0.25	

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank

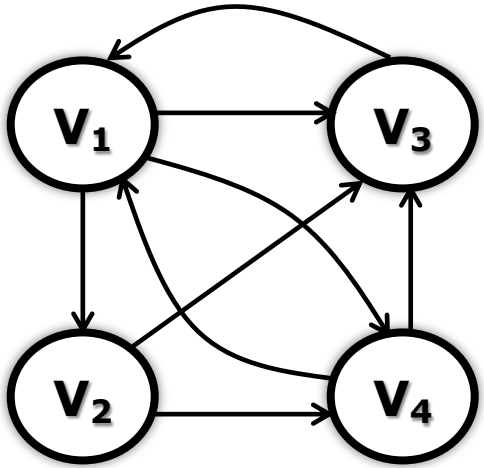


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0	K=1
PR(V_1)	0.25	0.37
PR(V_2)	0.25	0.08
PR(V_3)	0.25	0.33
PR(V_4)	0.25	0.20

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank



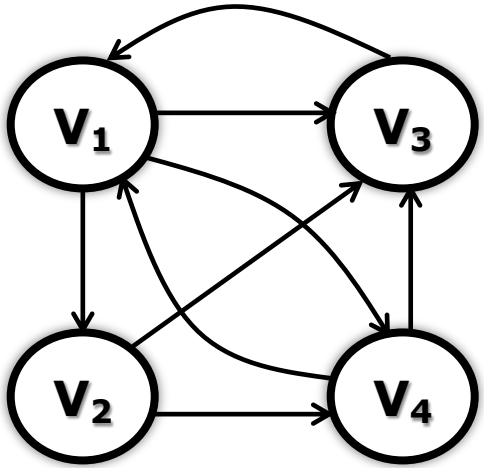
$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2
PR(V_1)	0.25	0.37	0.43
PR(V_2)	0.25	0.08	0.12
PR(V_3)	0.25	0.33	0.27
PR(V_4)	0.25	0.20	0.16

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank



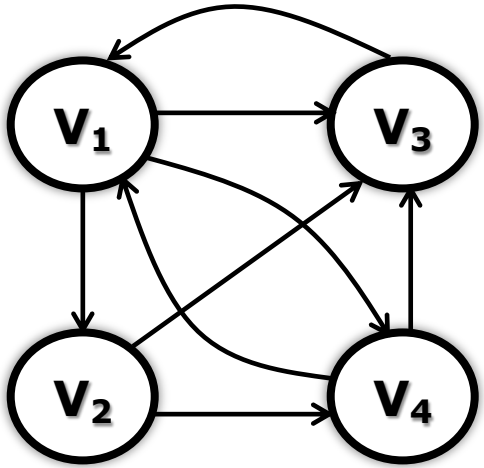
$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2	K=3
PR(V_1)	0.25	0.37	0.43	0.35
PR(V_2)	0.25	0.08	0.12	0.14
PR(V_3)	0.25	0.33	0.27	0.29
PR(V_4)	0.25	0.20	0.16	0.20

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank



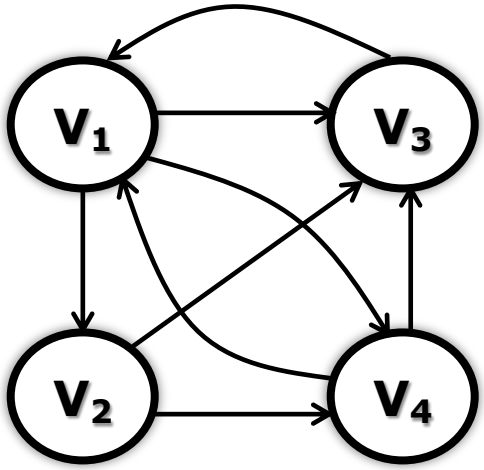
$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2	K=3	K=4
PR(V_1)	0.25	0.37	0.43	0.35	0.39
PR(V_2)	0.25	0.08	0.12	0.14	0.11
PR(V_3)	0.25	0.33	0.27	0.29	0.29
PR(V_4)	0.25	0.20	0.16	0.20	0.19

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank



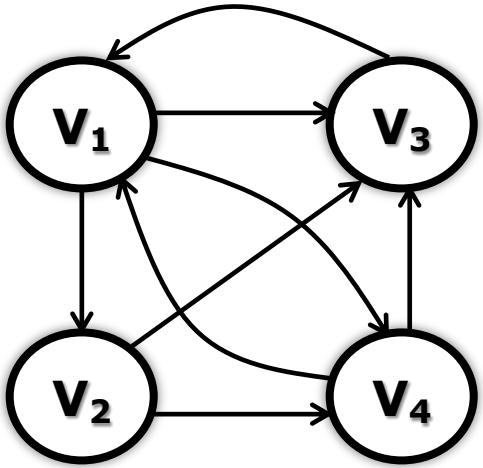
$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2	K=3	K=4	K=5
PR(V_1)	0.25	0.37	0.43	0.35	0.39	0.39
PR(V_2)	0.25	0.08	0.12	0.14	0.11	0.13
PR(V_3)	0.25	0.33	0.27	0.29	0.29	0.28
PR(V_4)	0.25	0.20	0.16	0.20	0.19	0.19

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Example: PageRank



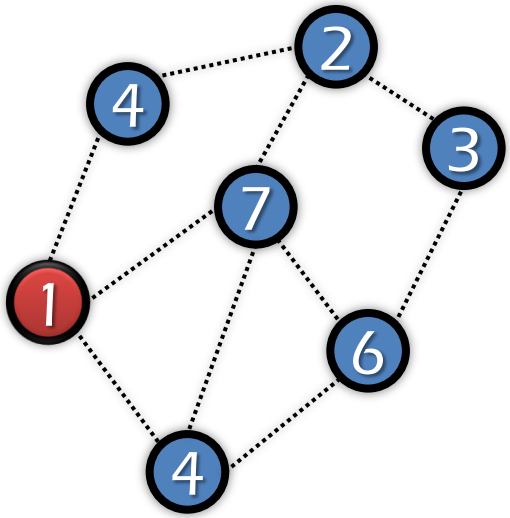
$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Convergence

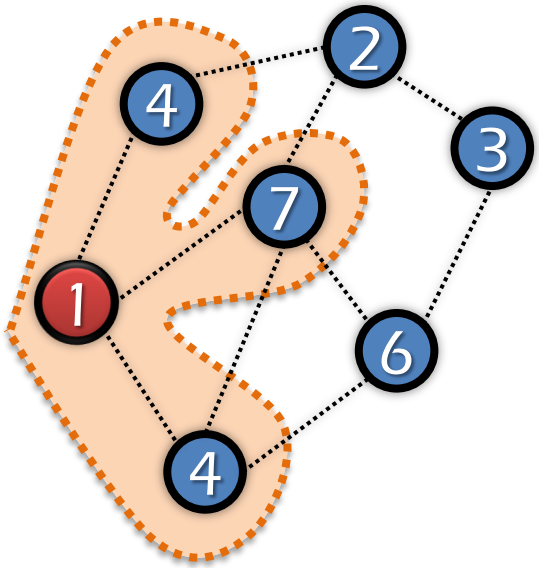
	K=0	K=1	K=2	K=3	K=4	K=5	K=6
PR(V_1)	0.25	0.37	0.43	0.35	0.39	0.39	0.38
PR(V_2)	0.25	0.08	0.12	0.14	0.11	0.13	0.13
PR(V_3)	0.25	0.33	0.27	0.29	0.29	0.28	0.28
PR(V_4)	0.25	0.20	0.16	0.20	0.19	0.19	0.19

Sergey Brin, Lawrence Page, "The Anatomy of Large-Scale Hypertextual Web Search Engine", WWW '98

Graph Parallel Algorithm



Dependency
Graph



Predictable
Updates

Iterative
Computation

What is the Right Tool
Graph-parallel ML

Data-Parallel

MapReduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

MapReduce

Lasso

PageRank

Label Propagation

Kernel Methods

Tensor Factorization

Belief Propagation

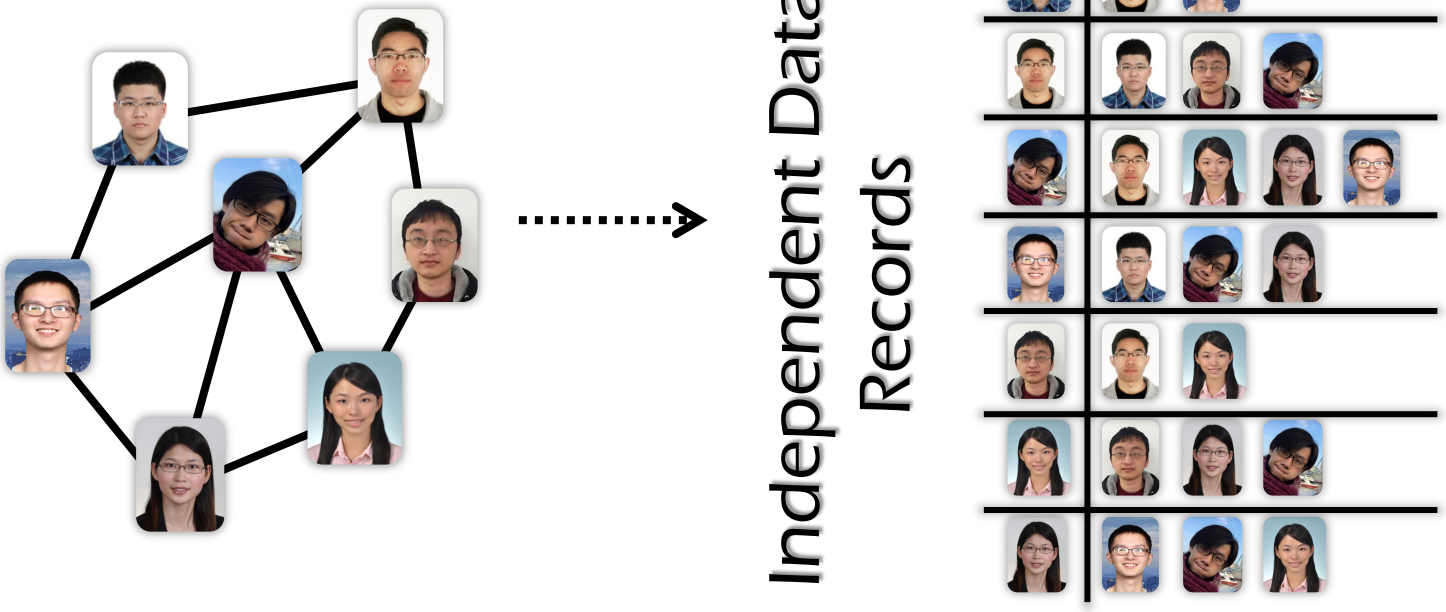
Deep Belief Networks

Neural Networks

Dependencies are Difficult

Difficult to express **dependent** data in MR

- Substantial data transformations
- User managed **graph structure**
- Costly data replication



PageRank on MapReduce

Multiple **MapReduce** iterations

Each PageRank iteration:

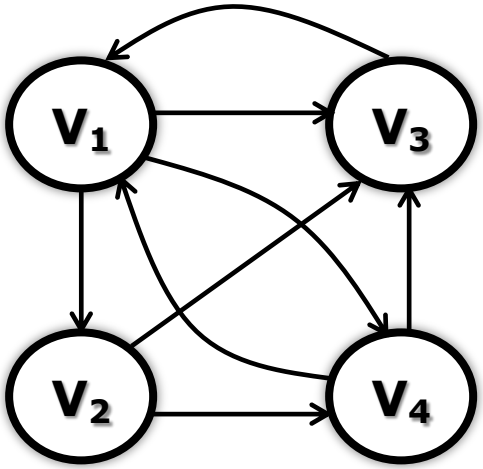
→ **Input:**

1. (**id**$_1$, [$PR_t(1)$, out_{11}, out_{12}, …]),
2. (**id**$_2$, [$PR_t(2)$, out_{21}, out_{22}, …]),
..

→ **Output:**

1. (**id**$_1$, [$PR_{t+1}(1)$, out_{11}, out_{12}, …]),
2. (**id**$_2$, [$PR_{t+1}(2)$, out_{21}, out_{22}, …]),
..

Iterate until **convergence**



Input:

V_1,[0.25, V_2, V_3, V_4]
V_2,[0.25, V_3, V_4]
V_3,[0.25, V_1]
V_4,[0.25, V_1, V_3]

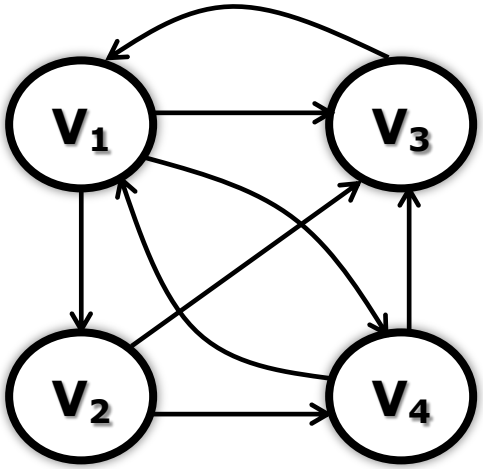
One MapReduce Iteration

Output:

V_1,[0.37, V_2, V_3, V_4]
V_2,[0.08, V_3, V_4]
V_3,[0.33, V_1]
V_4,[0.20, V_1, V_3]

PageRank on MapReduce

Map



- Input:

  (V1, [0.25, V2, V3, V4]);

  (V_2, [0.25, V_3, V_4]); (V_3, [0.25, V_1]);

  (V_4, [0.25, V_1, V_3])

- Output:

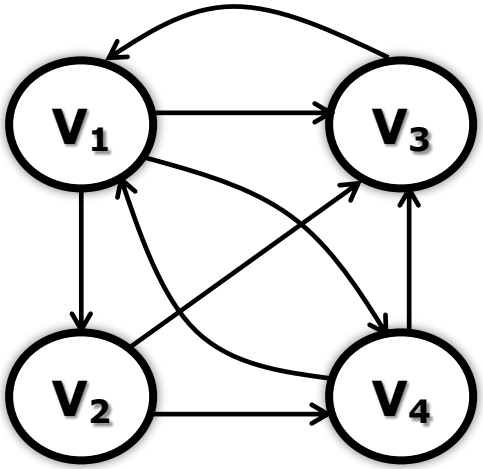
  (V2, 0.25/3), (V3, 0.25/3), (V4, 0.25/3),

  . . ., (V1, 0.25/2), (V3, 0.25/2);

  (V1, [V2, V3, V4]), (V2, [V3, V4]), (V3, [V1]), . . .

PageRank on MapReduce

Map



→ Input:

(V1, [0.25, V2, V3, V4]);

(V_2, [0.25, V_3, V_4]); (V_3, [0.25, V_1]);

(V_4, [0.25, V_1, V_3])

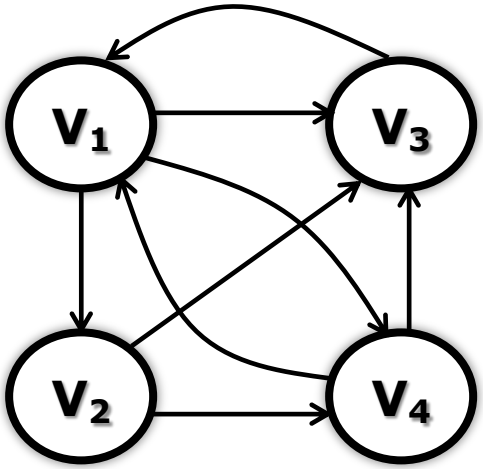
→ Output:

(V2, 0.25/3), (V3, 0.25/3), (V4, 0.25/3),

. . ., (V1, 0.25/2), (V3, 0.25/2);

(V1, [V2, V3, V4]), (V2, [V3, V4]), (V3, [V1]), . . .

PageRank on MapReduce



Map

→ Input:

(V1, [0.25, V2, V3, V4]);
(V_2, [0.25, V_3, V_4]); (V_3, [0.25, V_1]);
(V_4, [0.25, V_1, V_3])

→ Output:

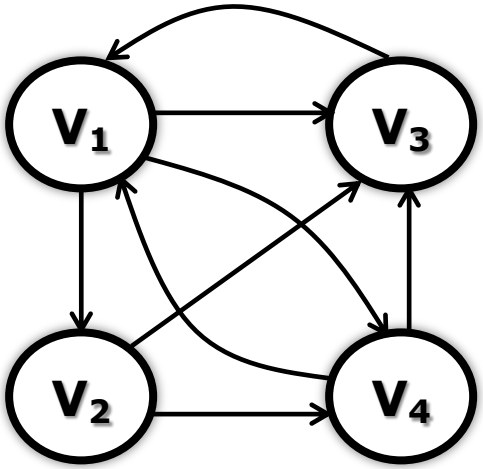
(V2, 0.25/3), (V3, 0.25/3), (V4, 0.25/3),
. . ., (V1, 0.25/2), (V3, 0.25/2);
(V1, [V2, V3, V4]), (V2, [V3, V4]), (V3, [V1]), . . .

Shuffle

→ Output:

(V1, 0.25/2), (V1, 0.25/1), (V1, [V2, V3, V4]); . . .;
(V4, 0.25/3), (V4, 0.25/2), (V4, [V1, V3])

PageRank on MapReduce



Map

→ Input:

(V1, [0.25, V2, V3, V4]);
(V_2, [0.25, V_3, V_4]); (V_3, [0.25, V_1]);
(V_4, [0.25, V_1, V_3])

→ Output:

(V2, 0.25/3), (V3, 0.25/3), (V4, 0.25/3),
. . ., (V1, 0.25/2), (V3, 0.25/2);
(V1, [V2, V3, V4]), (V2, [V3, V4]), (V3, [V1]), . . .

Shuffle

→ Output:

(V1, 0.25/2), (V1, 0.25/1), (V1, [V2, V3, V4]); . . .;
(V4, 0.25/3), (V4, 0.25/2), (V4, [V1, V3])

Reduce

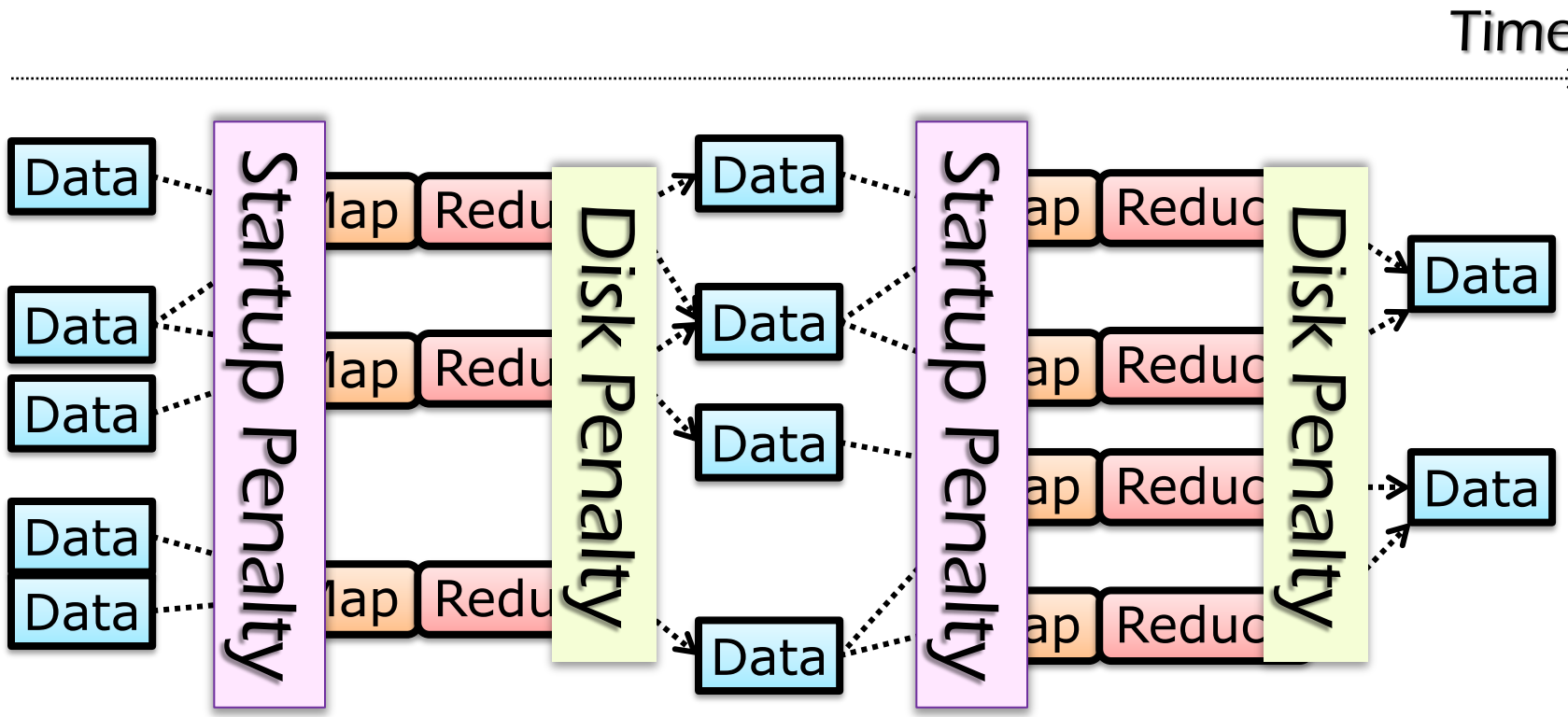
→ Output:

(V1, [0.37, V2, V3, V4]); (V2, [0.08, V3, V4]); (V3, [0.33, V1]); . .

Iterative Comp. is Difficult

MR runtime is not optimized for iteration

- Persistent I/O (e.g., GFS)



What is the Right Tool
Graph-parallel ML

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

MPI/Pthreads

Lasso

PageRank

Label Propagation

Kernel Methods

Tensor Factorization

Belief Propagation

Deep Belief Networks

Neural Networks

Threads, Locks and Messages

ML experts repeatedly solve the same parallel design challenges

- Impl. and debug complex parallel systems
- Tune for a specific parallel platform

Threads, Locks and Messages

Graduate Student

~~ML experts~~ repeatedly solve the **same** parallel design challenges

- Impl. and debug complex parallel systems
- Tune for a specific parallel platform

6 months later, the conference paper contains:

> *"We implemented ____ in parallel."*

The resulting **code**

- difficult to **maintain** and **extend**

couples **learning model** to **parallel impl.**

What is the Right Tool
Graph-parallel ML

Map Reduce

Feature Extraction

Cross Validation

Computing Sufficient Statistics

Graph-structured Computation

Lasso

PageRank

Label Propagation

Kernel Methods

Tensor Factorization

Belief Propagation

Deep Belief Networks

Neural Networks

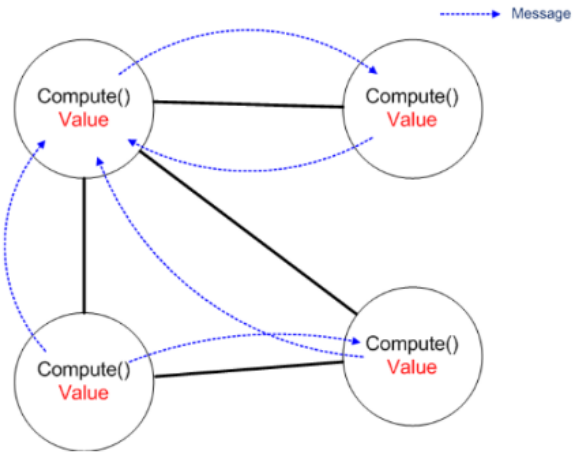
PREGEL

Pregel (Google, 2010)

Inspired by Valiant's
Bulk Synchronous Parallel (BSP) model

Communication through pure **message-passing**
(usually sent along the outgoing edges from each vertex)
+ **shared-nothing**

Vertex-centric programming
"Think like a vertex"



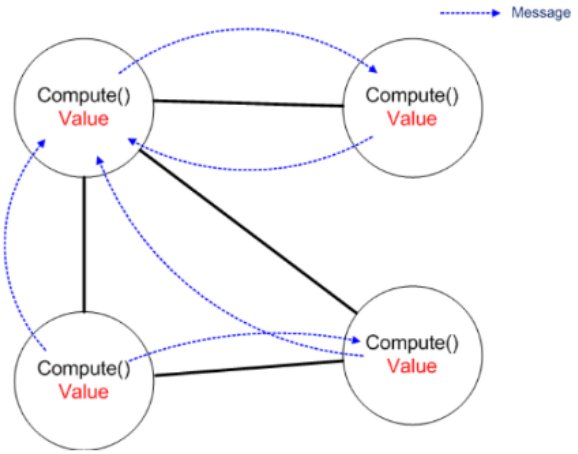
G. Malewicz et. al., "Pregel: A System for Large-Scale Graph Processing", SIGMOD '10

Pregel (Google, 2010)

Each vertex

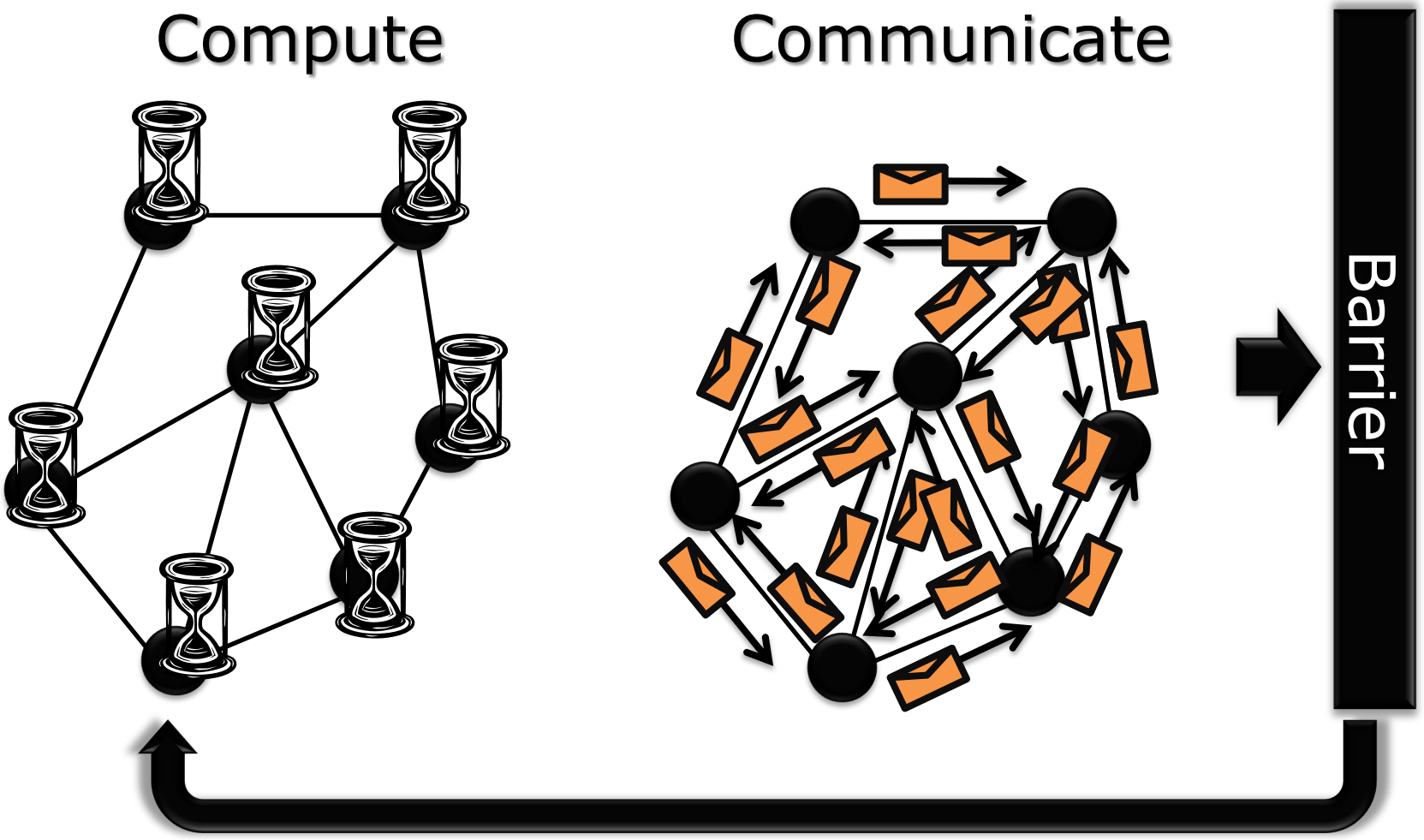
1. Receives messages sent in the previous superstep
2. Executes the same user-defined function
3. Modifies its value
4. If active, sends messages to other vertices (received in the next superstep)
5. Votes to halt if it has no further work to do → inactive

Terminate when
all vertices are **inactive** and
no messages in transmit



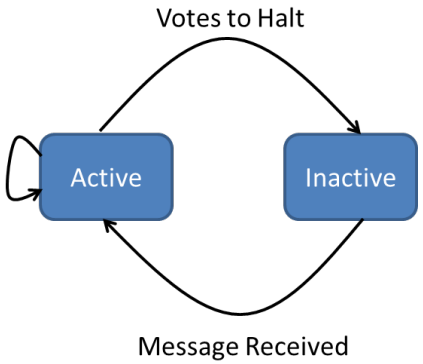
G. Malewicz et. al., "Pregel: A System for Large-Scale Graph Processing", SIGMOD '10

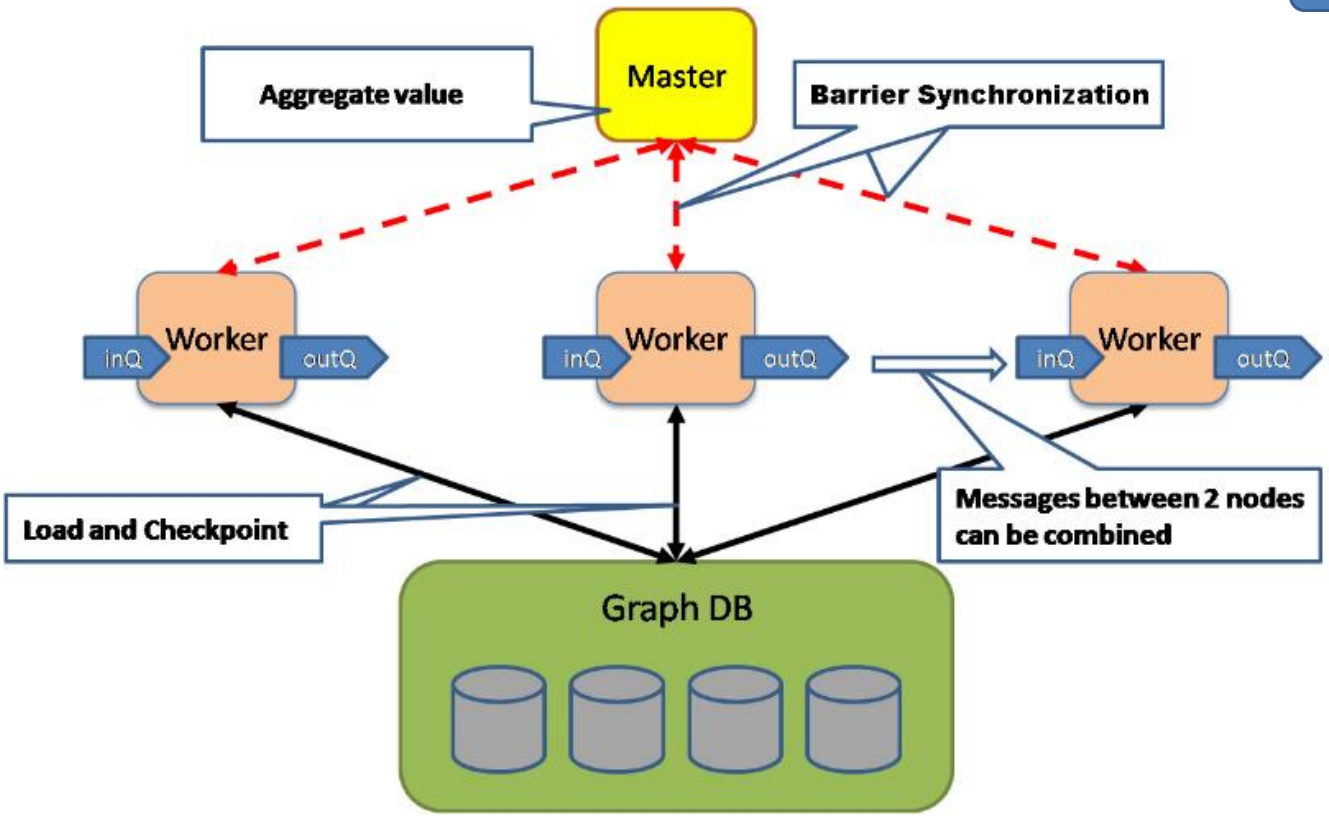
Bulk Synchronous Parallel Model



System Architecture of Pregel

Master-Slave architecture



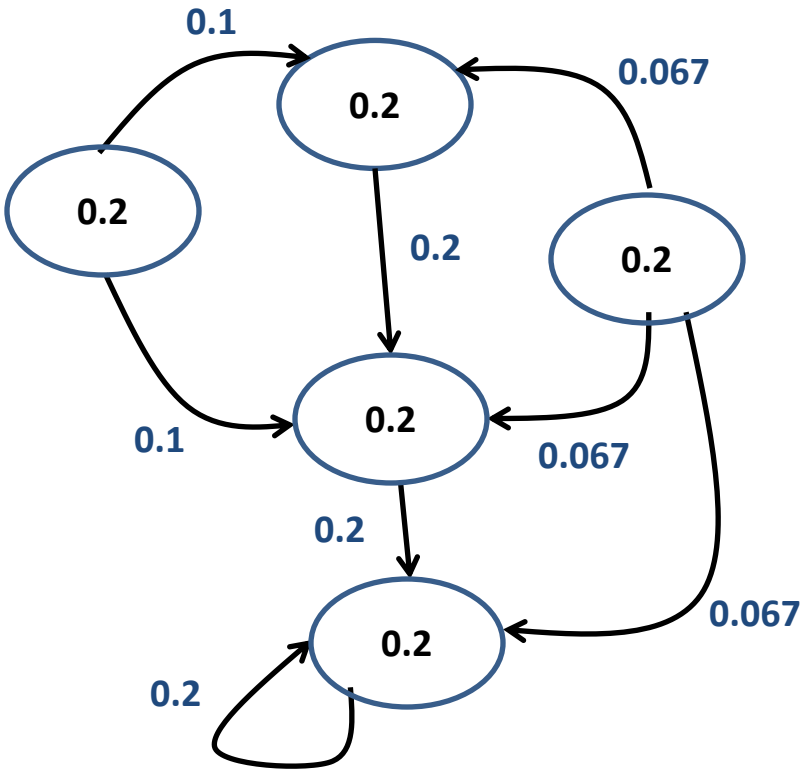


PageRank on Pregel

Superstep 0: PR value of each vertex 1/NumVertices()

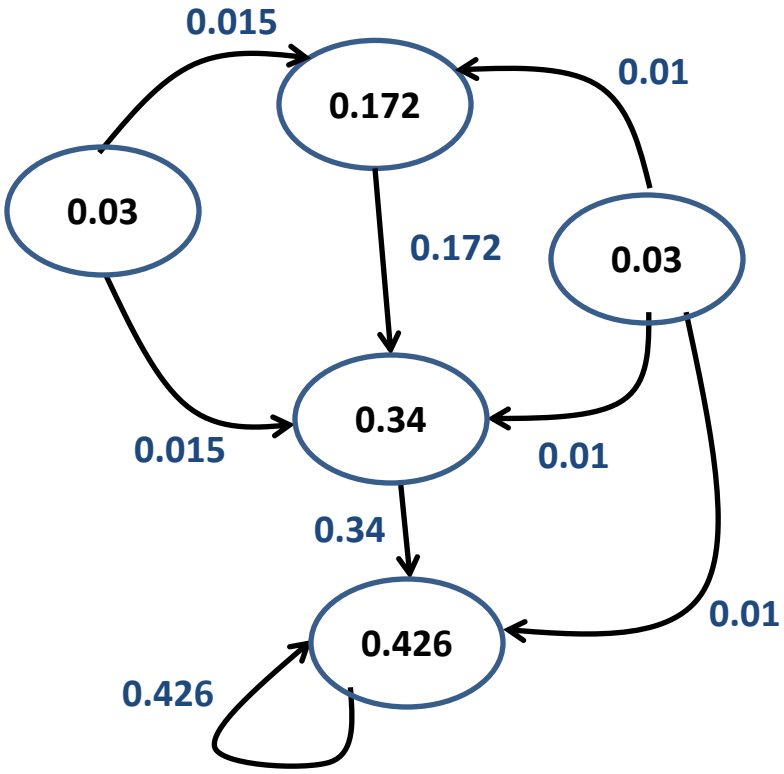
```
Class PageRankVertex {
public:
  virtual void Compute(MessageIterator* msgs) {
   if (superstep () >= 1) {
     double sum = 0;
     for (; !msgs->Done(); msgs->Next())
       sum += msgs->Value();
       *MutableValue() = 0.15/NumVertices() + 0.85*sum;
     }
     if (superstep() < 30) {
       const int64 n = GetOutEdgeIterator().size();
       SendMessageToAllNeighbors(GetValue() / n);
     } else {
       VoteToHalt();
     }
}
```

PageRank on Pregel



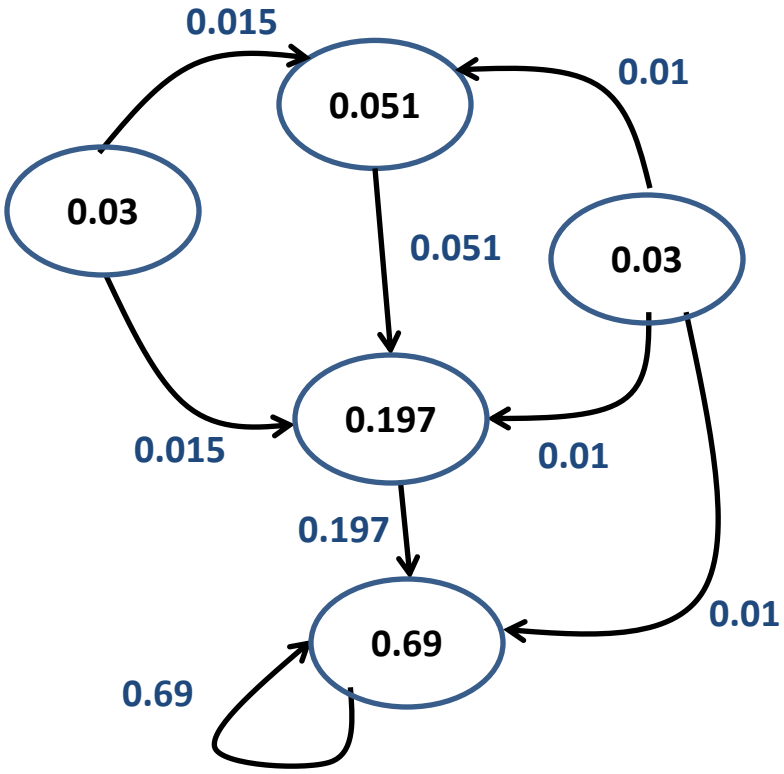
Superstep = 0

PageRank on Pregel



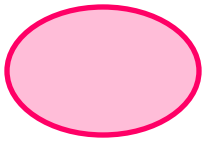
Superstep = 1

PageRank on Pregel

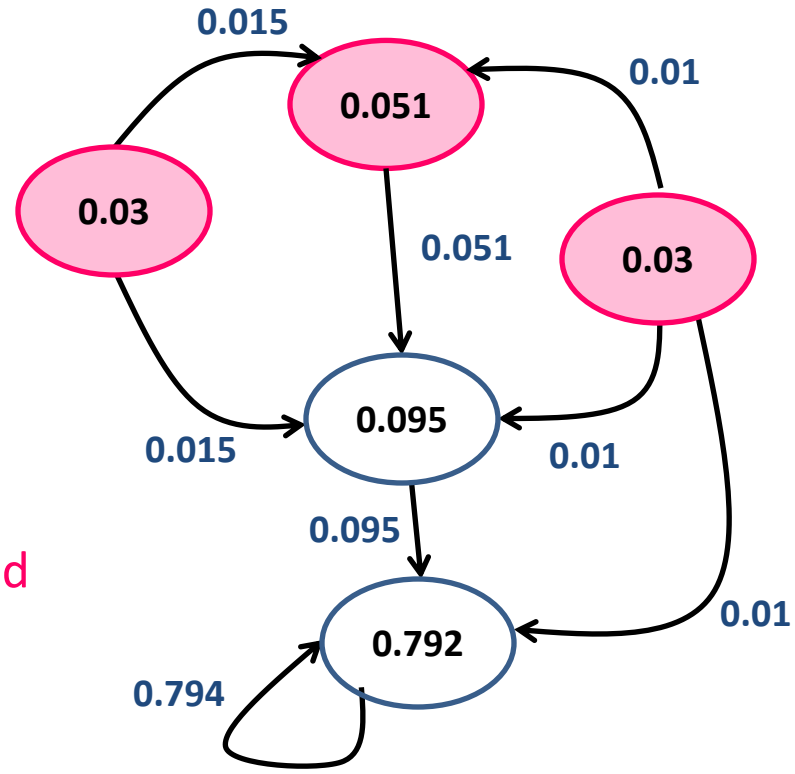


Superstep = 2

PageRank on Pregel

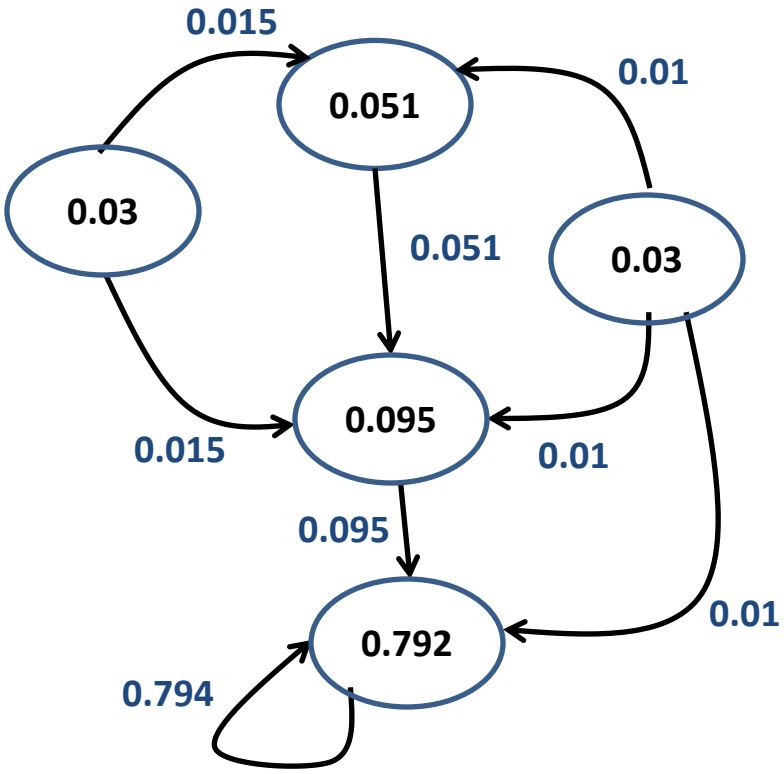


Computation converged



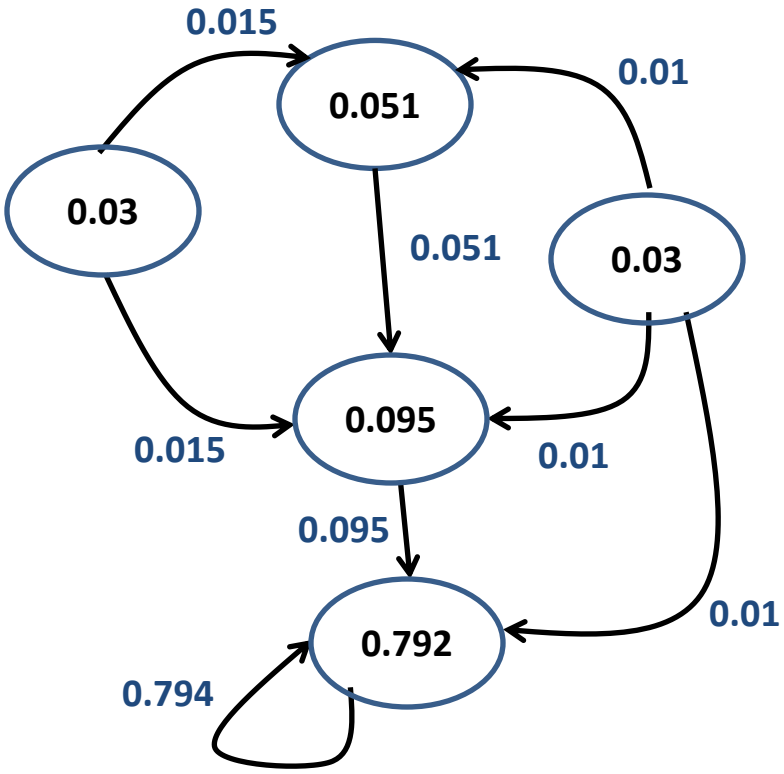
Superstep = 3

PageRank on Pregel



Superstep = 4

PageRank on Pregel

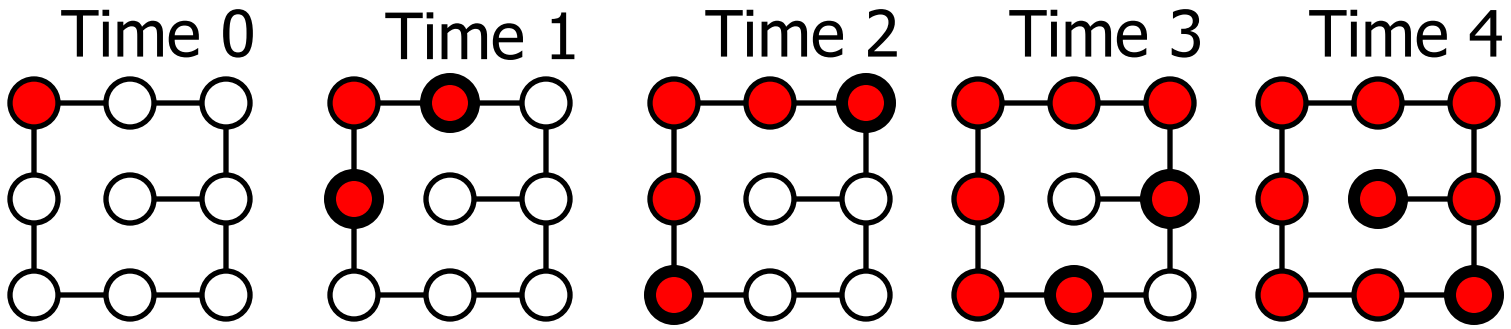


Superstep = 5

GRAPHLAB

Problem with BSP Model

Example Algorithm: If Red neighbor then turn Red



Bulk Synchronous Parallel Computation :

- Evaluate condition on all vertices for every phase

  4 Phases each with 9 computations → 36 Computations

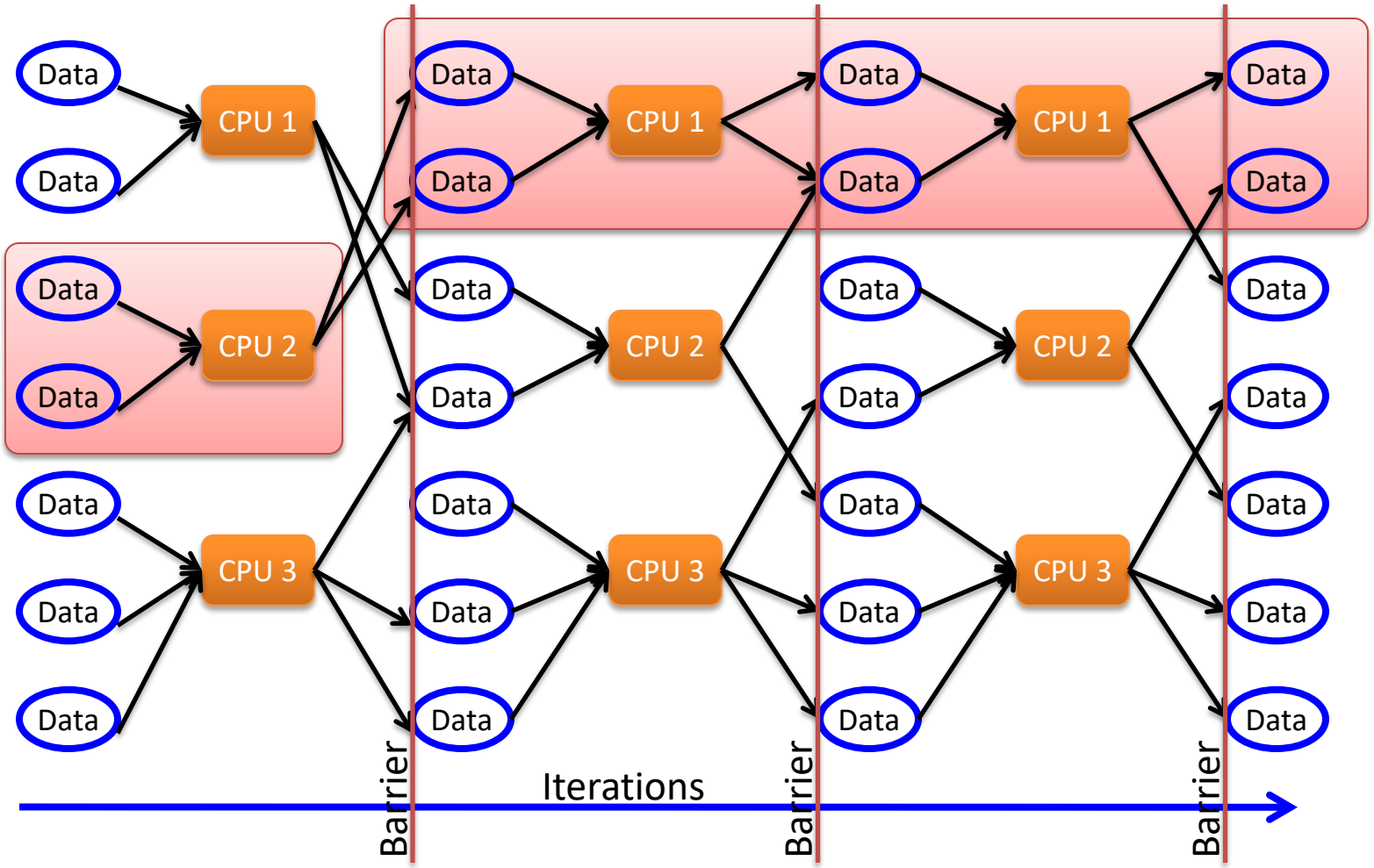
Asynchronous Computation:

- Evaluate condition only when neighbor changes

  4 Phases each with 2 computations → 8 Computations

Problem with BSP Model

performance is limited by the slowest machine



GraphLab Framework

Graph Based
Data Representation

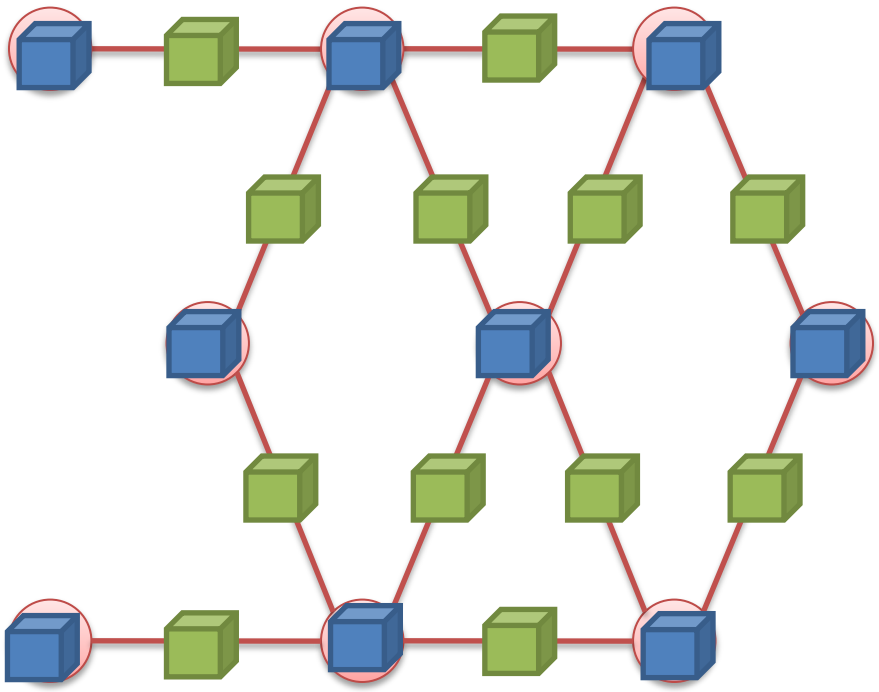
Update Functions
User Computation

Scheduler

Consistency Model

Data Graph

A **graph** with arbitrary data (*C++ Objects*) associated with each **vertex** and **edge**



Graph: Social Network

Vertex Data:

1. User profile text
2. Current interests estimates

Edge Data:

1. Similarity weights

Implementation

All data and structure is stored in memory

- Supports fast random lookup needed for dynamic computation

Multicore Setting

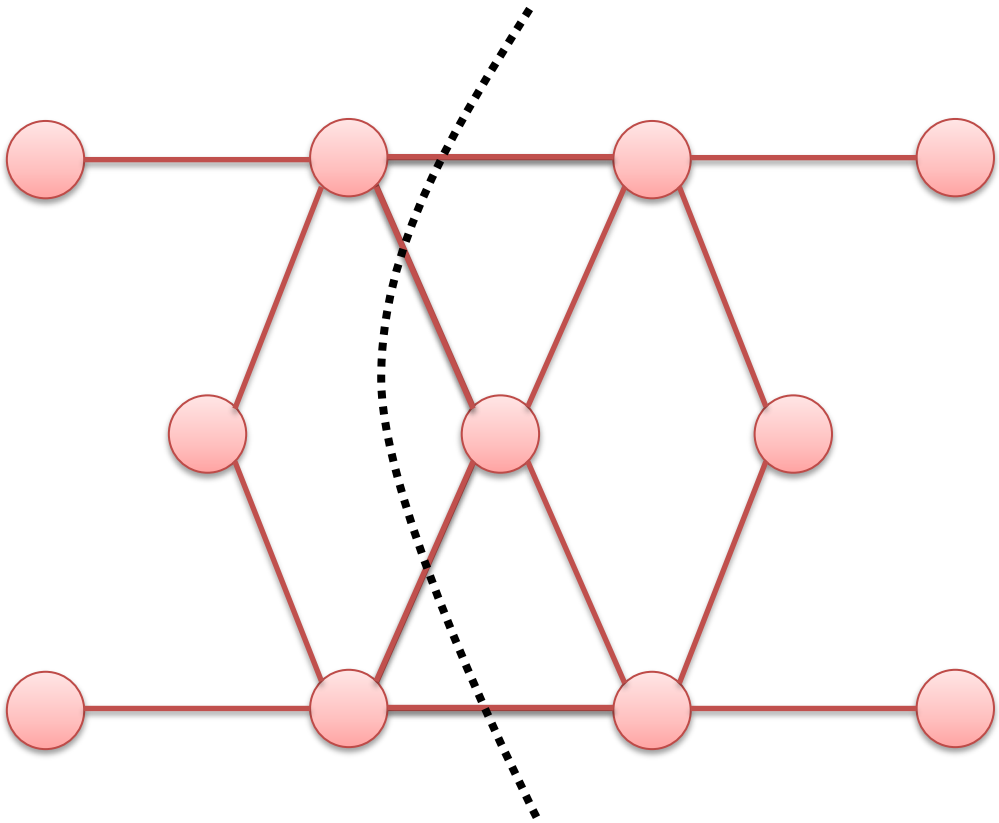
- Challenge: Fast lookup, low overhead
- Solution → dense data-structures (e.g., array)

Distributed Setting

- Challenge: Graph partitioning
- Solution → ParMetis and Random placement

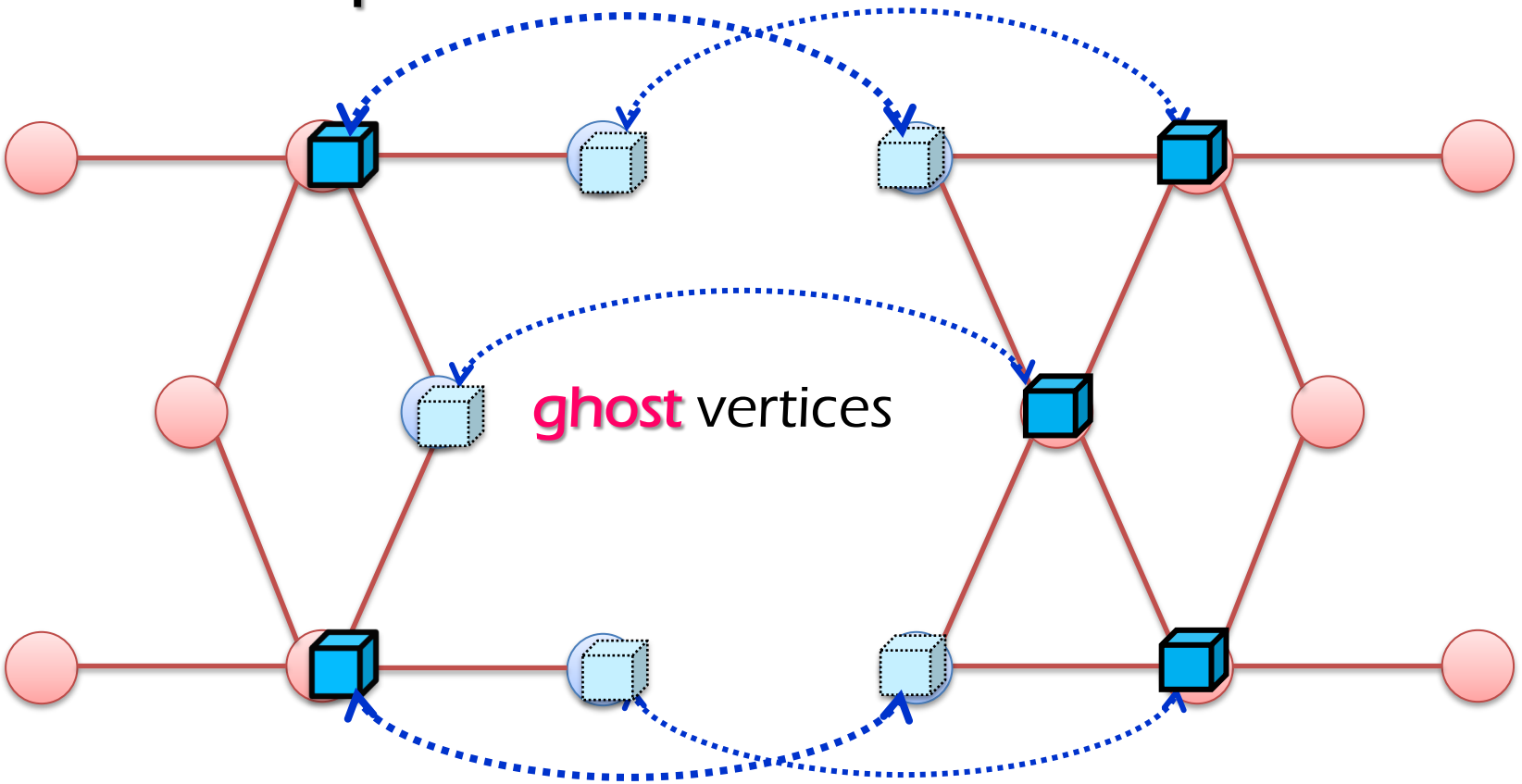
Distributed Graph

Partition the graph across multiple machines



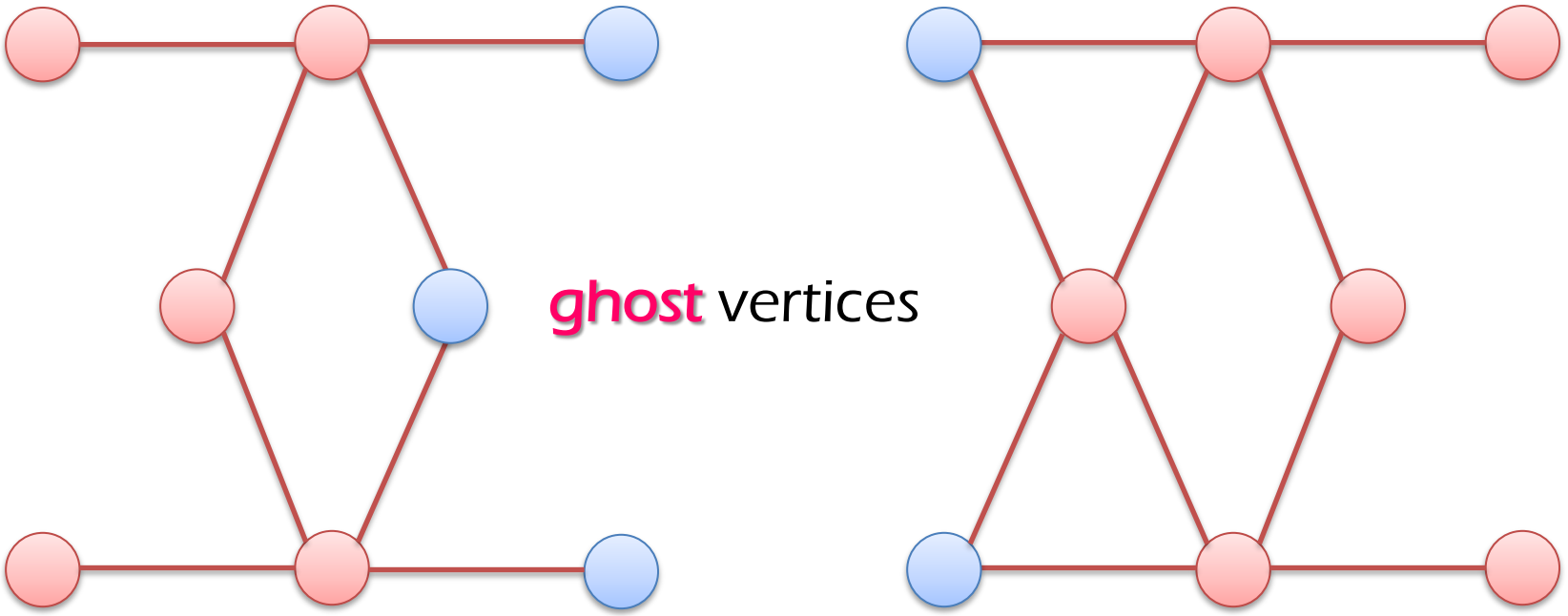
Distributed Graph

Ghost vertices maintain adjacency structure and replicate remote data



Distributed Graph

Cut **efficiently** using HPC Graph partitioning tools (e.g., ParMetis, …)



GraphLab Framework

Graph Based
Data Representation

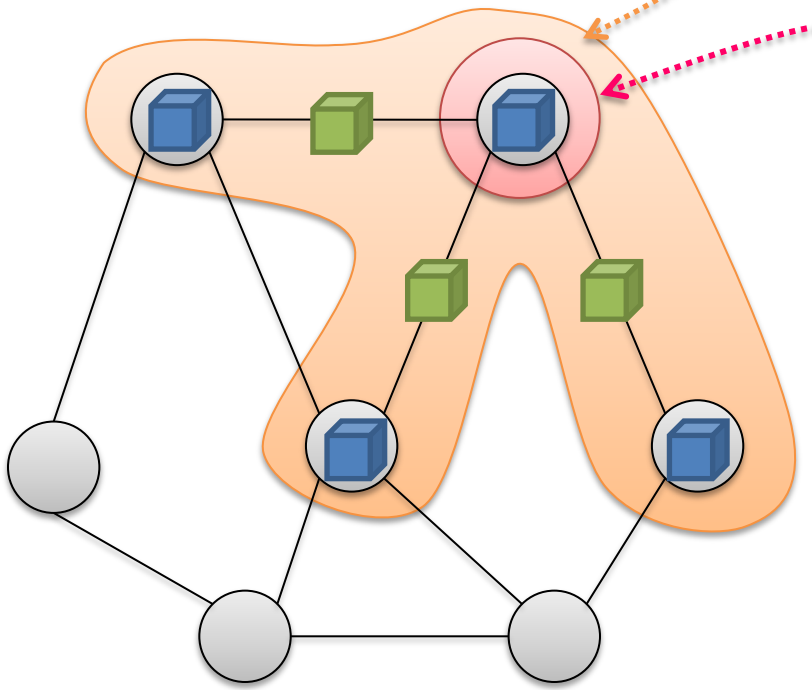
Update Functions
User Computation

Scheduler

Consistency Model

Update Function

update function: a user defined program which when applied to a **vertex** transforms the data in the **scope** of the vertex

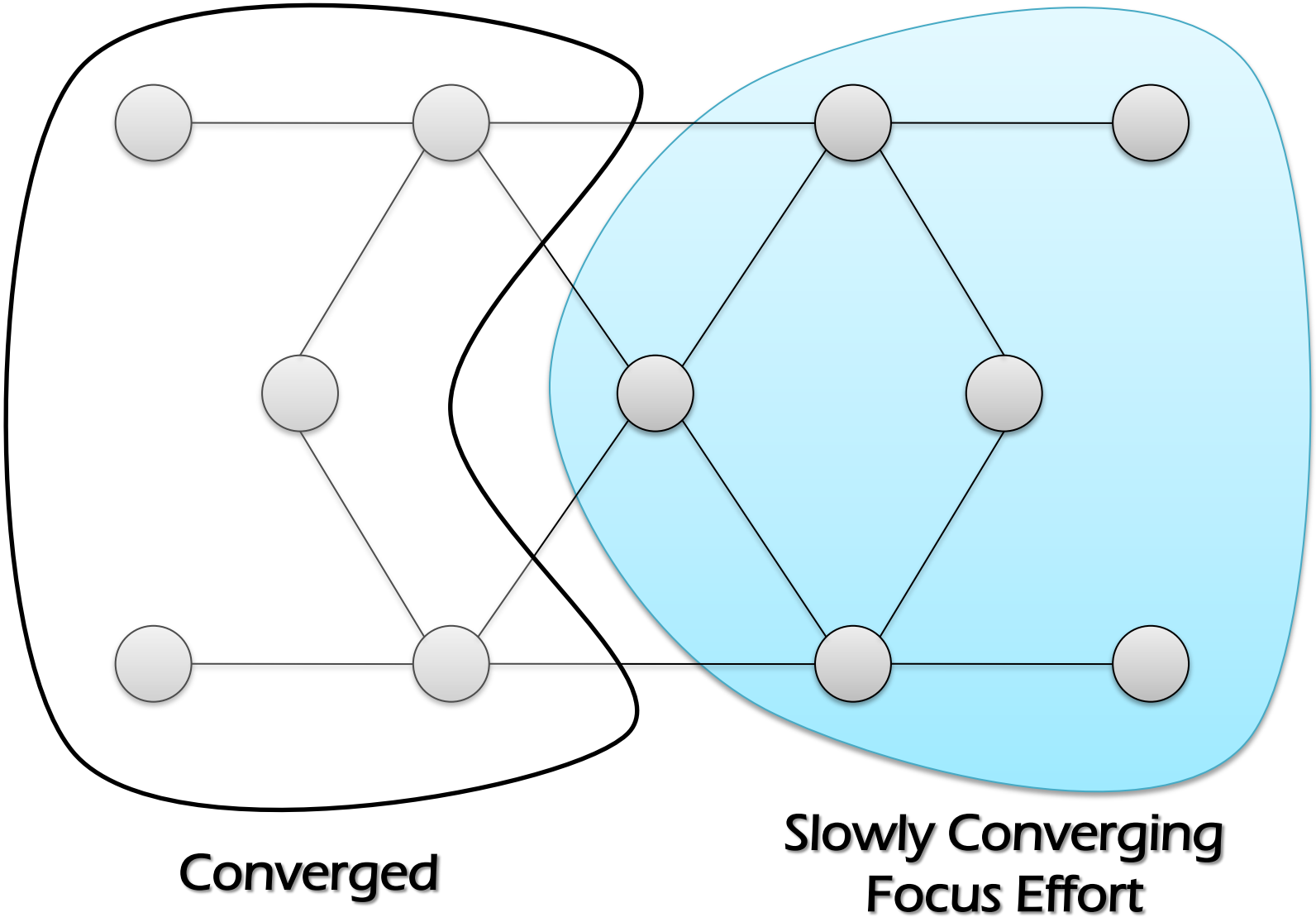


```
void PageRank(scope){
  // Get Neighborhood data
  float sum = 0, diff;
  float last = this.val;
  foreach (nbr in scope.in_nbrs)
    sum += nbr.val/nbr.nout_nbrs;

  // Update the vertex data
  this.val = 0.15 + 0.85*sum;

  // Reschedule Nbrs if needed
  diff = abs(this.val - last);
  if (diff > epsilon)
     reschedule(this.out_nbrs);
}
```

Dynamic Computation



GraphLab Framework

Graph Based
Data Representation

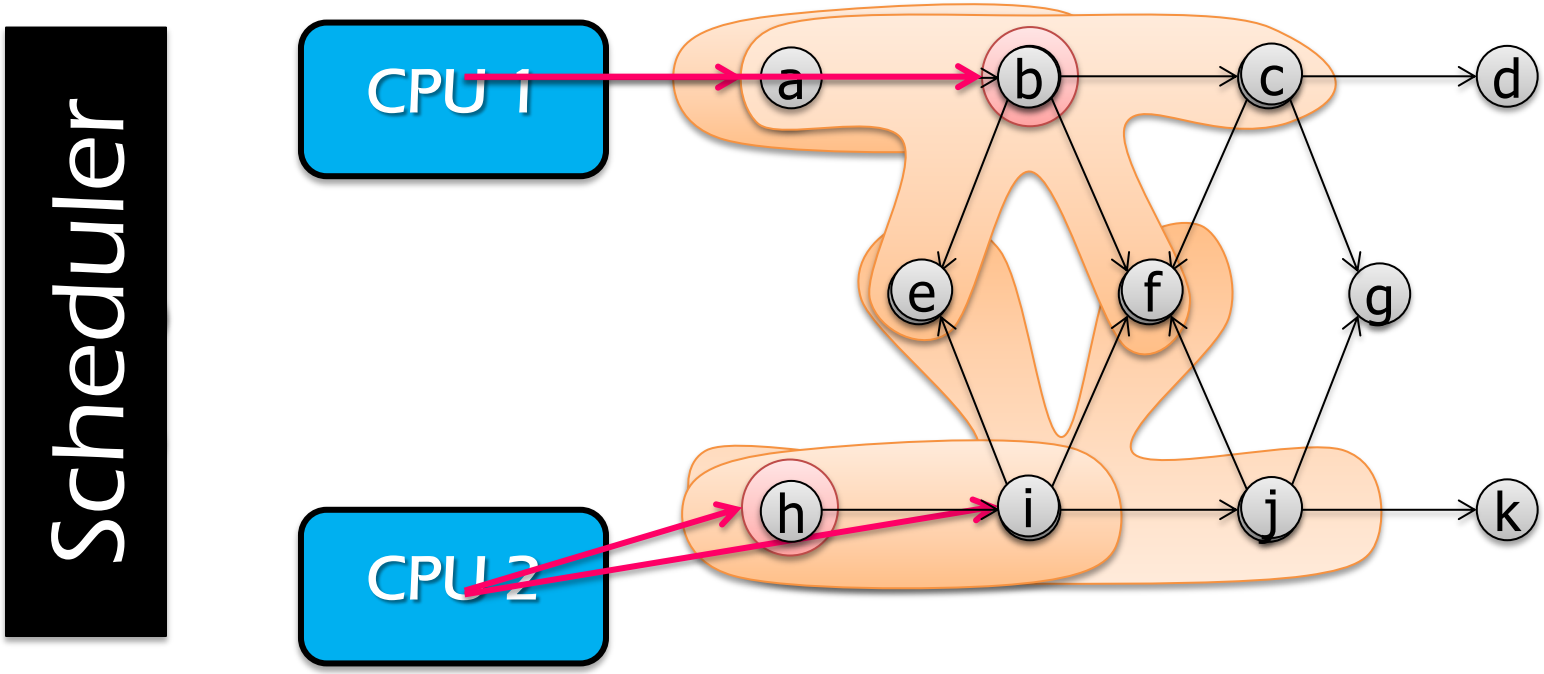
Update Functions
User Computation

Scheduler

Consistency Model

GraphLab Scheduler

The scheduler determines the order that vertices are updated



The process repeats until the scheduler is empty

GraphLab Scheduler

The choice of schedule affects the correctness and parallel performance of the algorithm

GraphLab provides several diff schedulers

- Vertices are updated

  → **Round Robin**, **FIFO**, **Priority**

GraphLab Framework

Graph Based
Data Representation

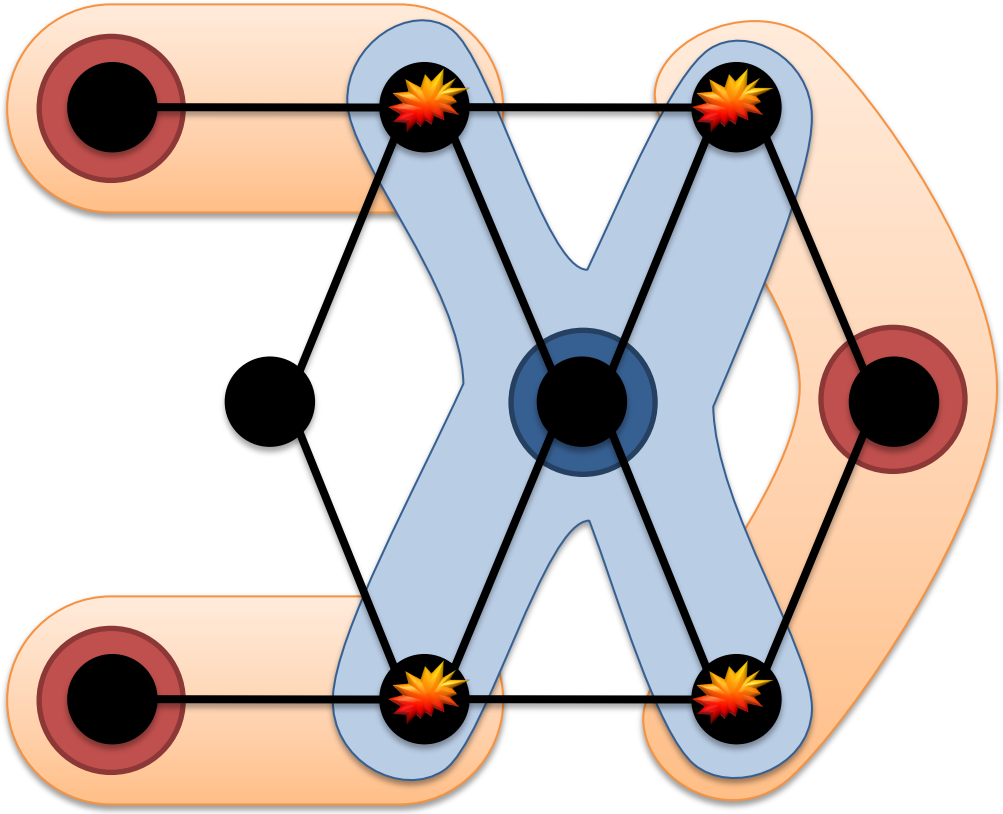
Update Functions
User Computation

Scheduler

Consistency Model

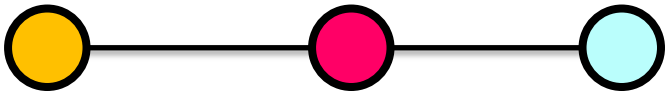
Race-free Execution

How much can computation overlap?



Sequential Consistency

For each **parallel** execution, there exists a **sequential** execution of **update** functions which produces the **same** result.



Parallel

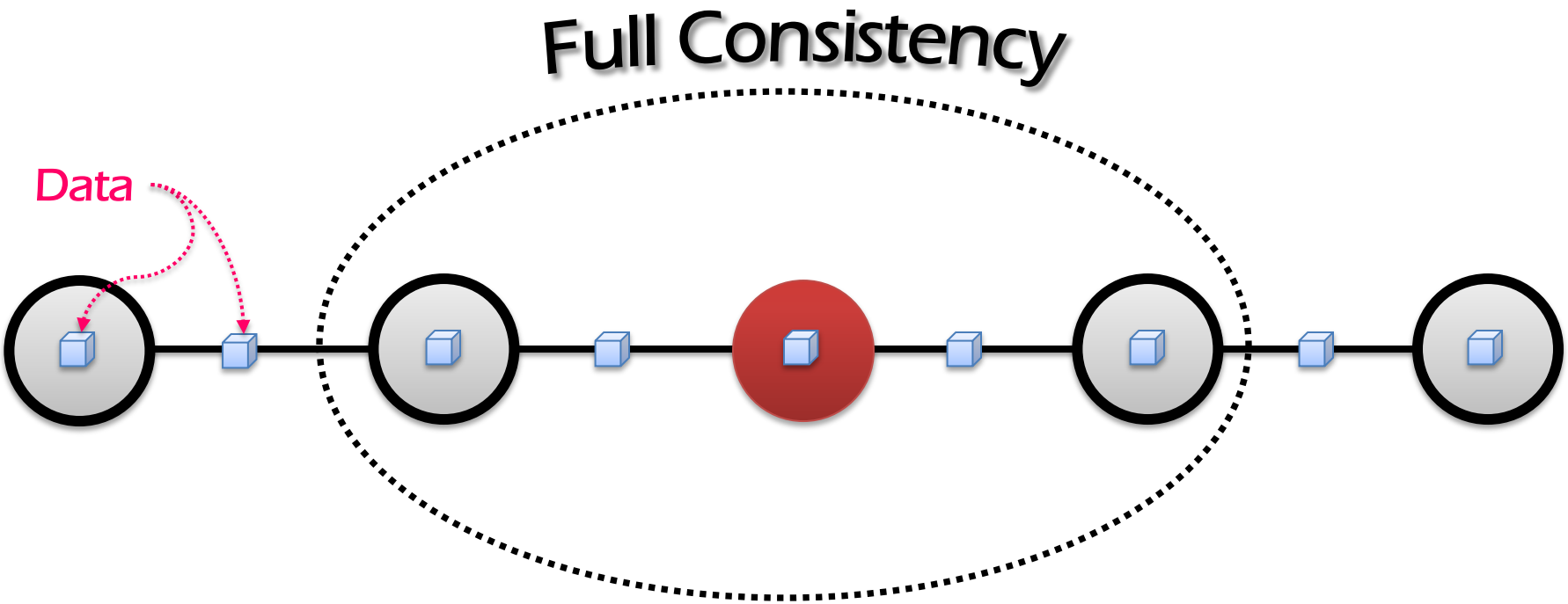
CPU 1

CPU 2

Sequential

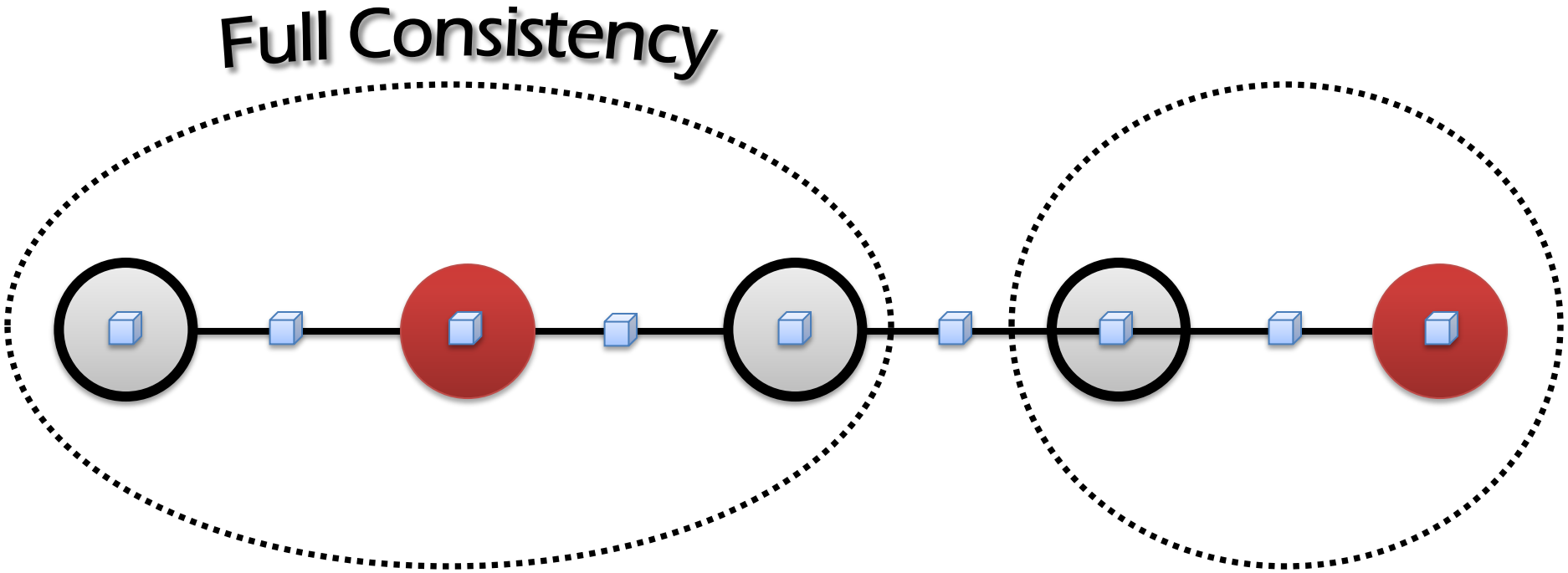
Single CPU

Consistency Rule

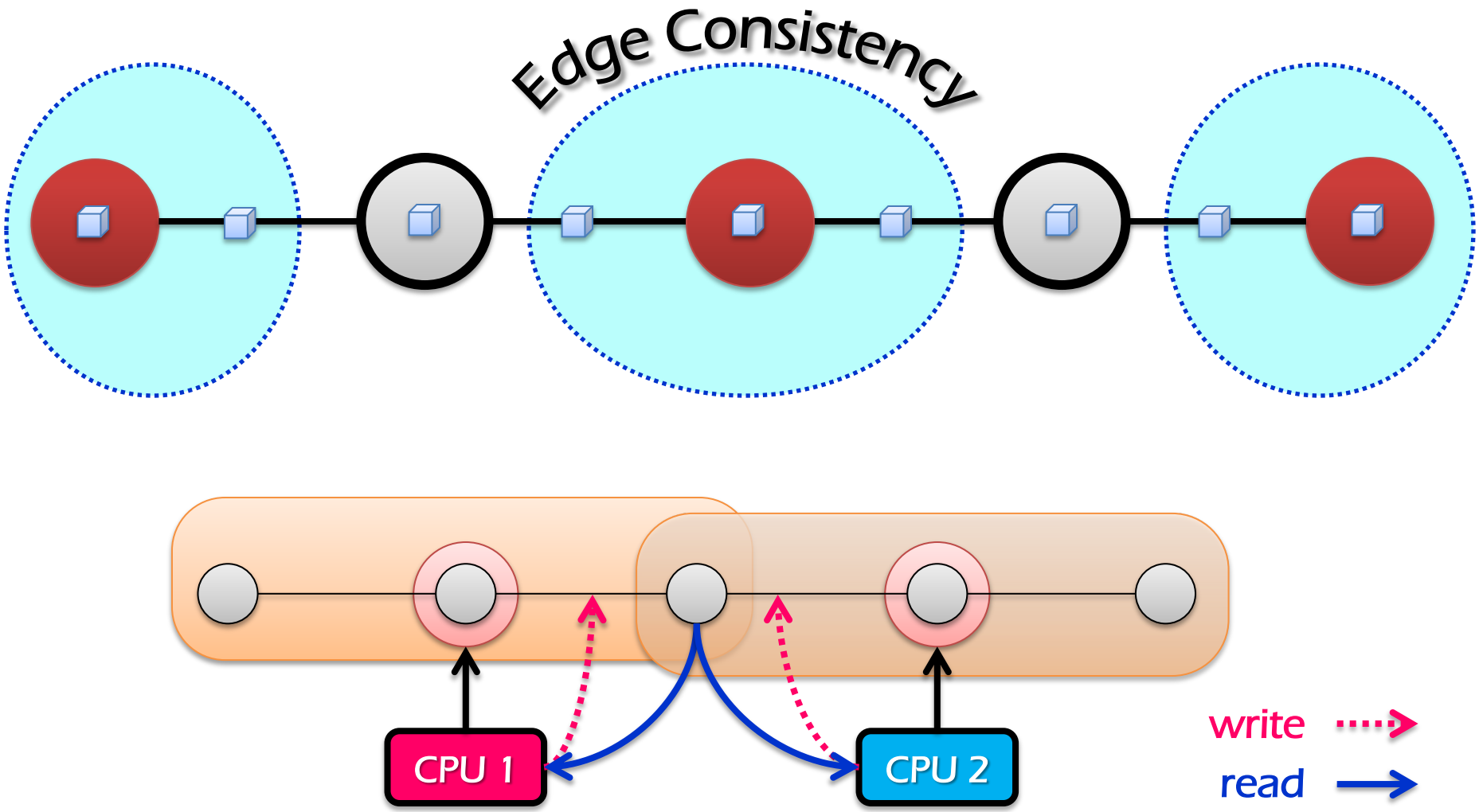


Guaranteed sequential consistency for all update functions

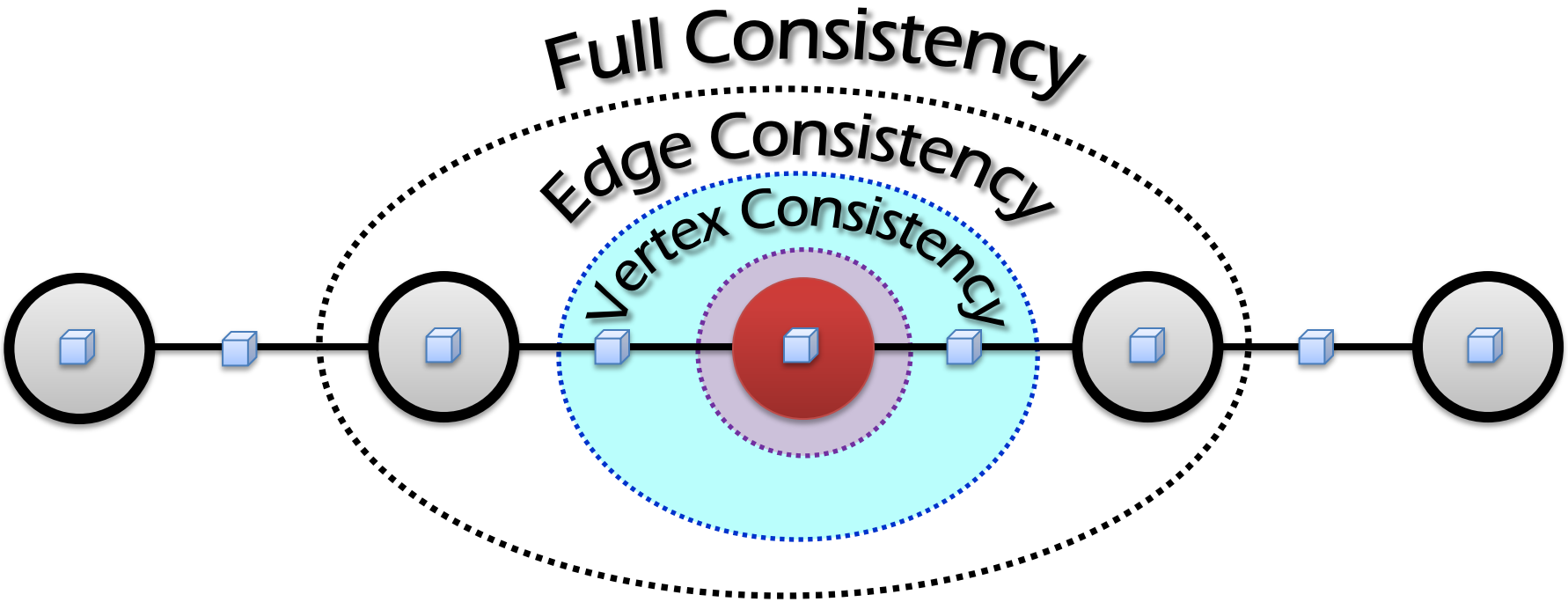
Full Consistency



Edge Consistency



More Parallelism



Consistency through Scheduling

Edge Consistency Model

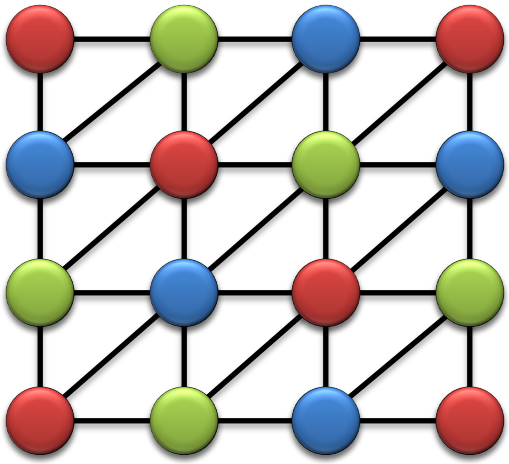
- Two vertices can be updated **simultaneously** → they do not share an edge

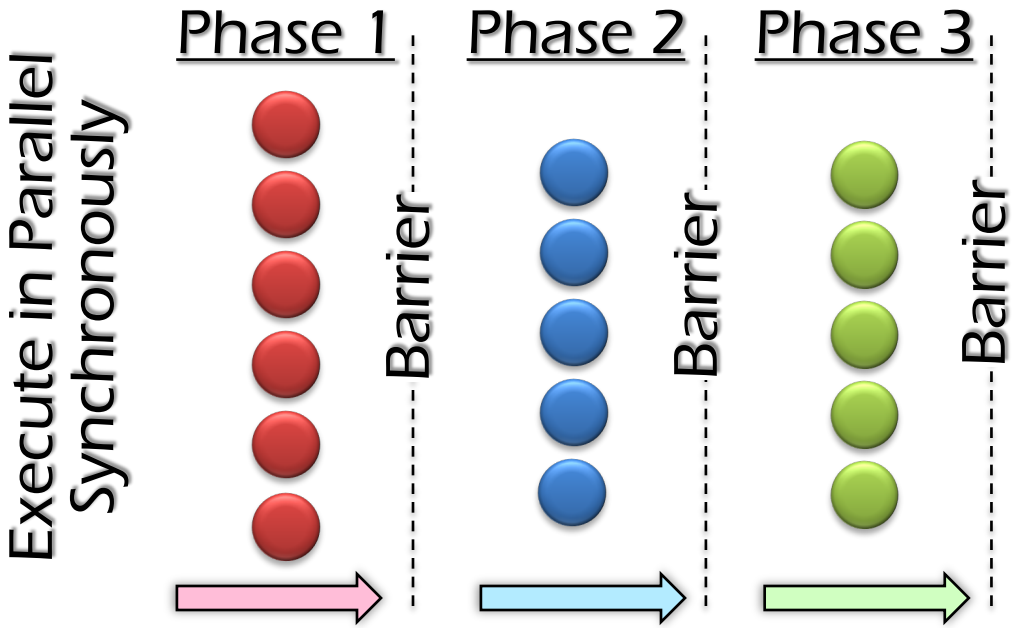
Consistency through Scheduling

Edge Consistency Model

- Two vertices can be updated ~~simultaneously~~ same color
  → they do not share an edge

Graph Coloring





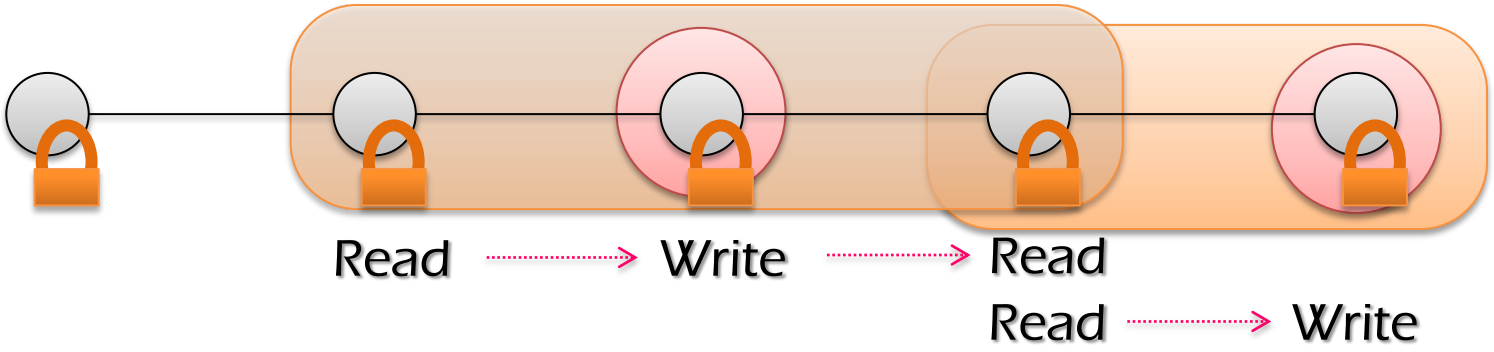
Consistency through R/W Locks

Multicore Setting: R/W Locks (Pthread)

- **Full** consistency

Canonical Lock Ordering

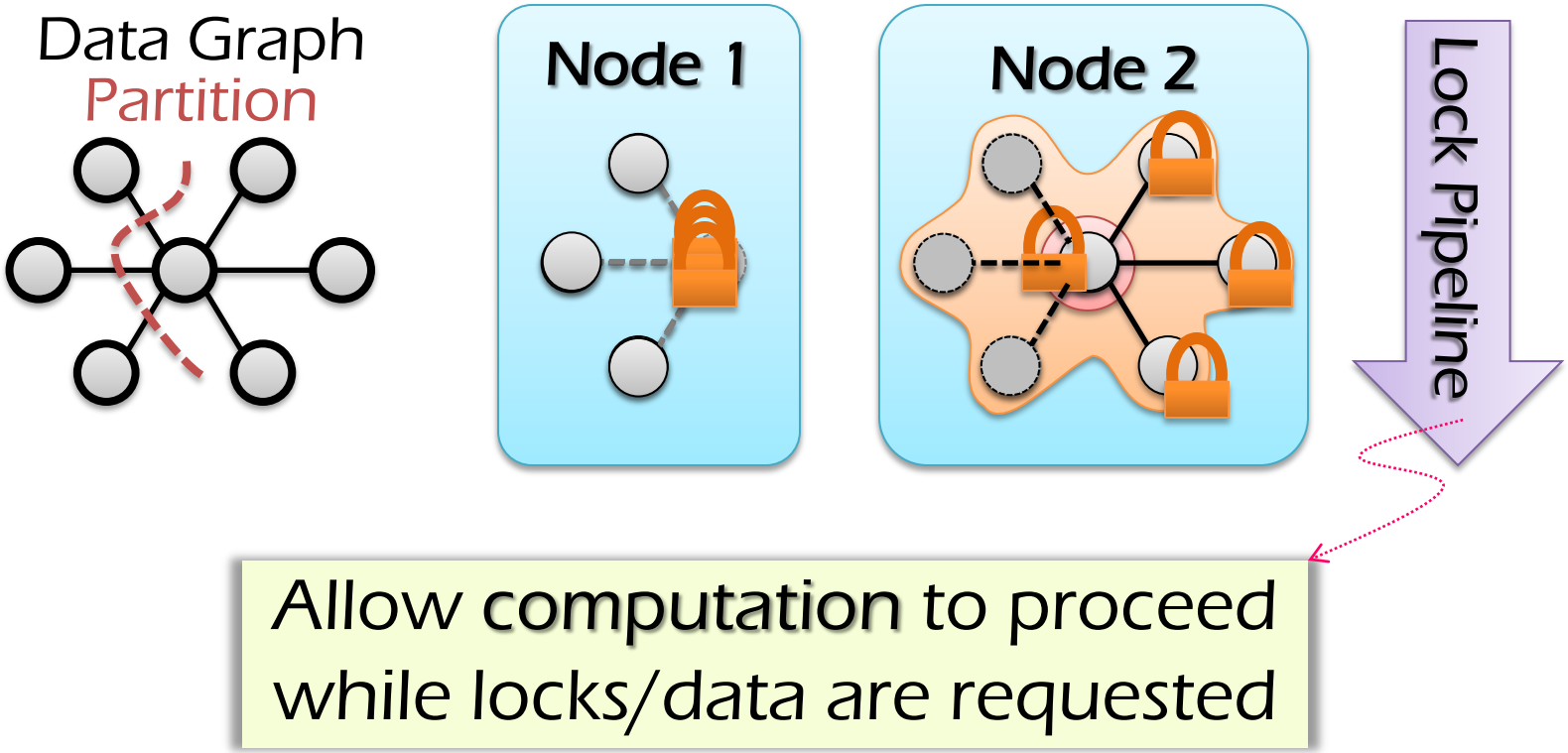
- **Edge** consistency



Consistency through R/W Locks

Distributed Setting: Distributed Locking

- Prefetch Locks and Data



Conclusion

An **abstraction** tailored to **Machine Learning**

A **distributed** and **parallel** framework which compactly expresses

- Data/computational **dependencies**
- **Iterative** computation

http://www.graphlab.org/

Next Lecture

Topic: Distributed Graph Partitioning

THANKS