

# Graph Processing

Acknowledgement: Arijit Khan, Sameh Elnikety

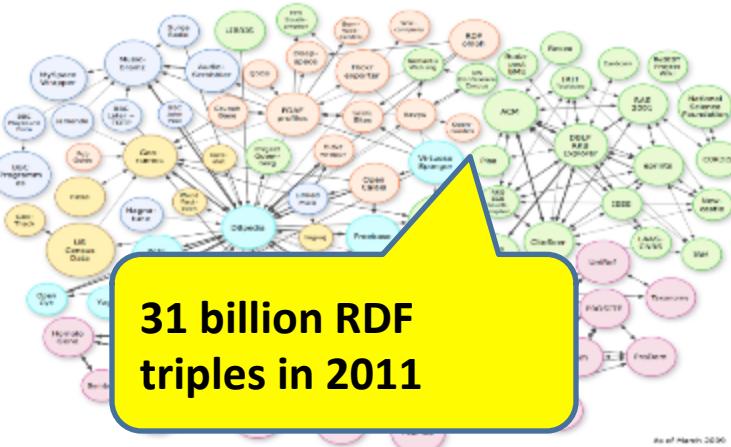
# Big Graphs



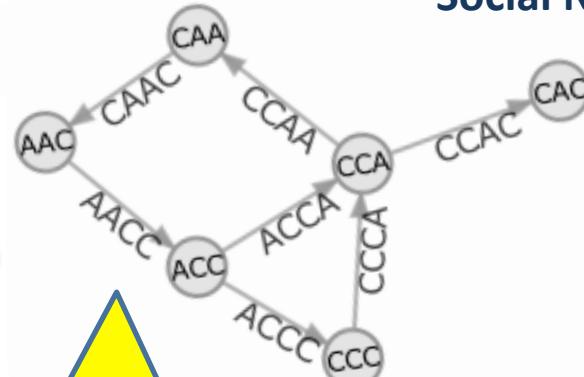
Web Graph



Social Network



Information Network



De Bruijn:  
4<sup>k</sup> nodes  
(k = 20, ..., 40)

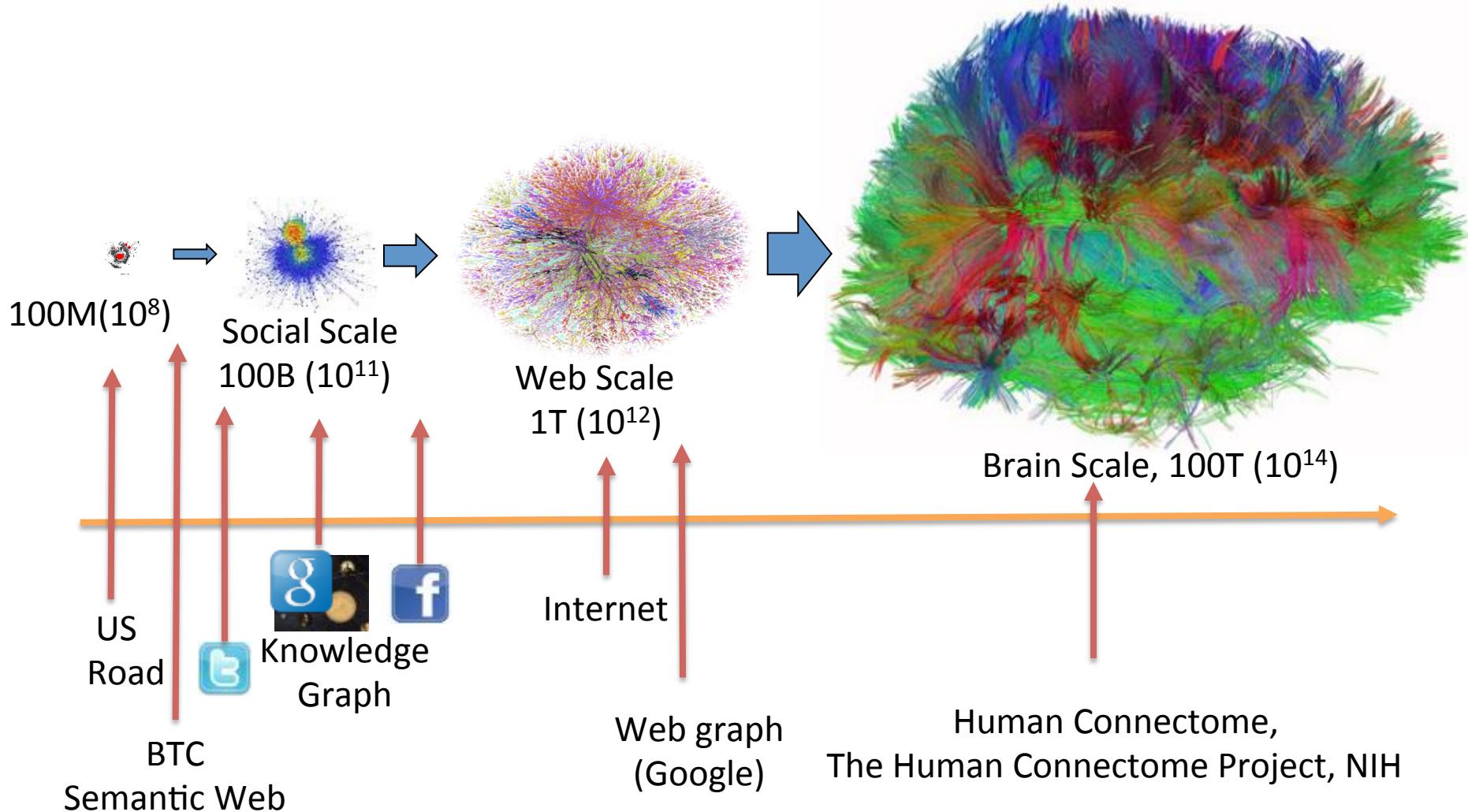
Biological Network



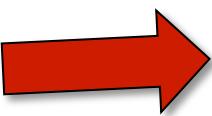
100M Ratings,  
480K Users,  
17K Movies

Graphs in Machine Learning

# Big-Graph Scales



# Graph Data: Structure + Attributes



**Peter Norvig**  
Research Director at Google  
San Francisco Bay Area | Computer Software

**Peter Norvig's Overview**

Current	<b>Engineering Director at Google</b>
Past	Division Chief, Computational Sciences at NASA Head, Computational Sciences Division at NASA Ames Chief Scientist at Junglee <a href="#">see all</a>
Education	University of California, Berkeley Brown University
Recommendations	<b>1 person has recommended Peter</b>
Connections	<b>500+ connections</b>
Websites	<a href="#">Personal Website</a> <a href="#">Company Website</a> <a href="#">RSS feed</a>

**Peter Norvig's Summary**

Programmer, designer, author, and manager in high tech R&D.

**Specialties**  
internet search, artificial intelligence, natural language processing, machine learning, programming, education

# Graph Data: Structure + Attributes



LinkedIn

- Web Graph: 20 billion web pages  $\times$  20KB  
= 400 TB
- 30-35 MB/sec disk data-transfer rate = 4 months to read the web

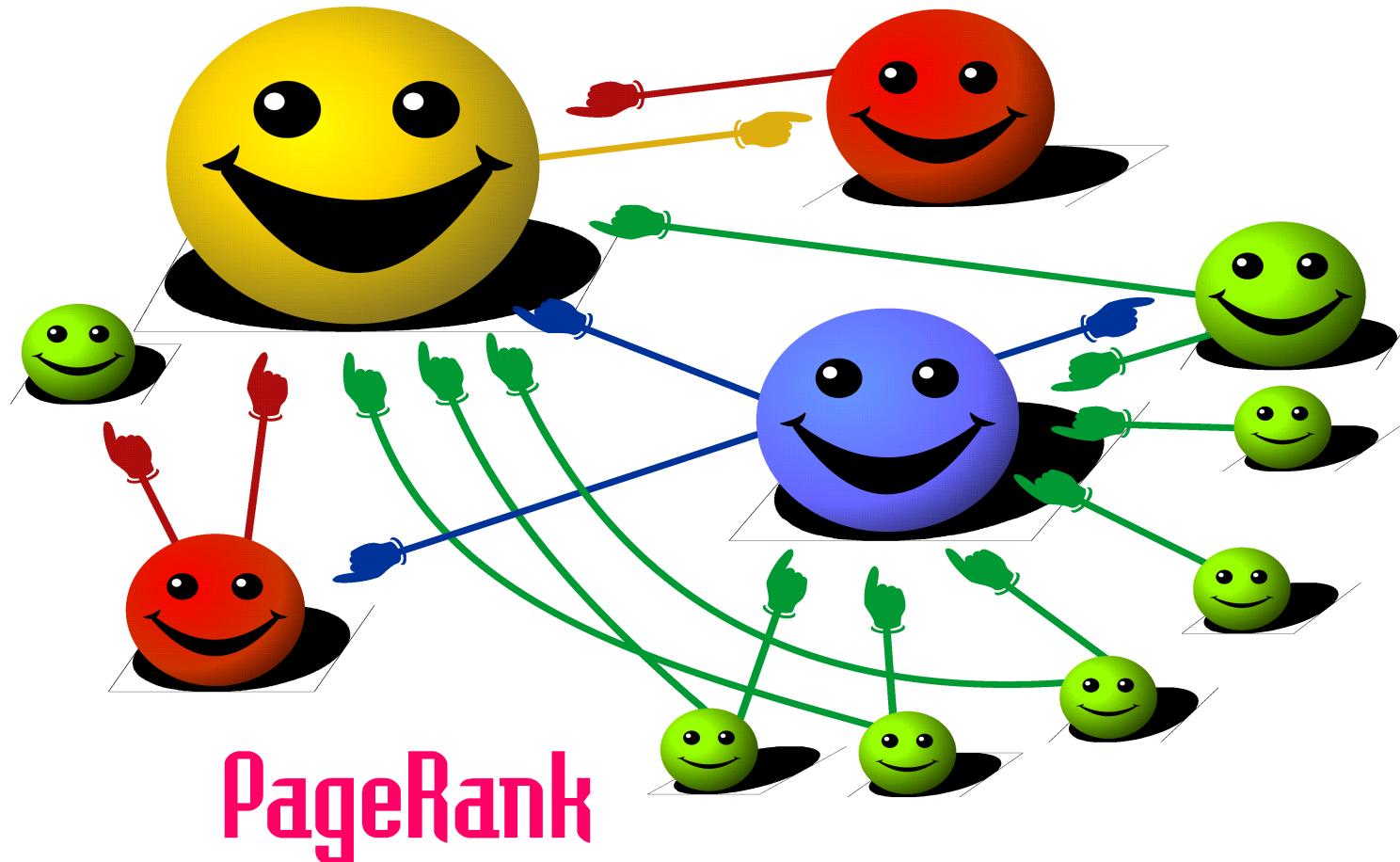
Peter Norvig's Summary

Programmer, designer, author, and manager in high tech R&D.

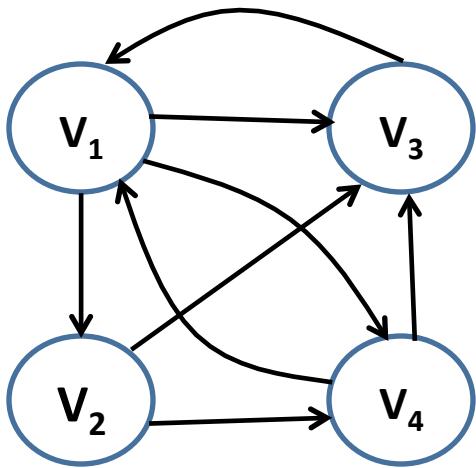
Specialties

internet search, artificial intelligence, natural language processing, machine learning, programming, education

# Page Rank Computation: Offline Graph Analytics



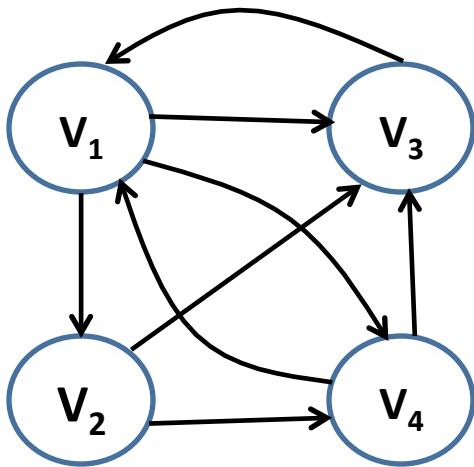
# Page Rank Computation: Offline Graph Analytics



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

- PR(u): Page Rank of node u
- F<sub>u</sub>: Out-neighbors of node u
- B<sub>u</sub>: In-neighbors of node u

# Page Rank Computation: Offline Graph Analytics

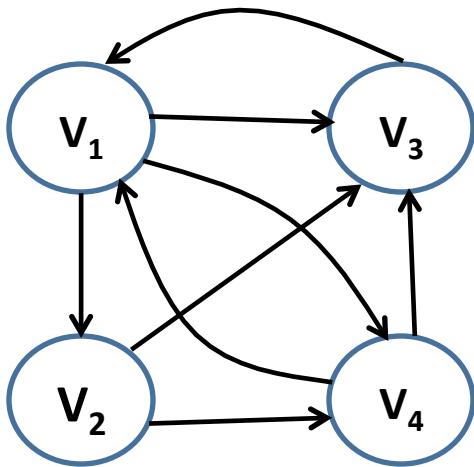


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0
PR(V <sub>1</sub> )	0.25
PR(V <sub>2</sub> )	0.25
PR(V <sub>3</sub> )	0.25
PR(V <sub>4</sub> )	0.25

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

# Page Rank Computation: Offline Graph Analytics

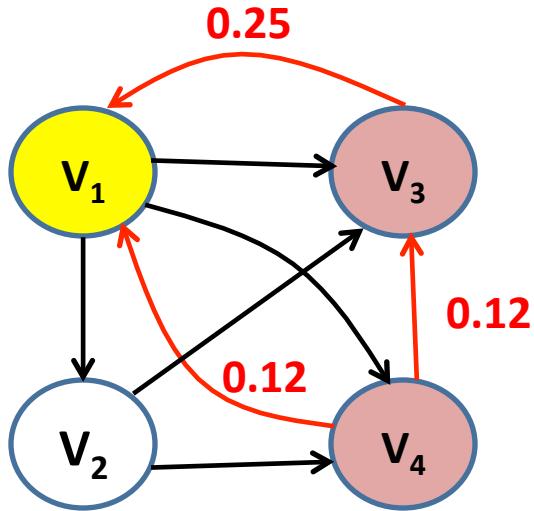


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	K=0	K=1
PR(v <sub>1</sub> )	0.25	?
PR(v <sub>2</sub> )	0.25	
PR(v <sub>3</sub> )	0.25	
PR(v <sub>4</sub> )	0.25	

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

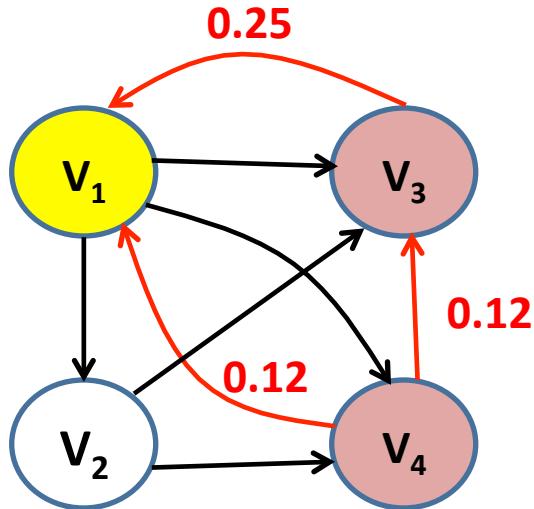
# Page Rank Computation: Offline Graph Analytics



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

	K=0	K=1
PR( $v_1$ )	0.25	?
PR( $v_2$ )	0.25	
PR( $v_3$ )	0.25	
PR( $v_4$ )	0.25	

# Page Rank Computation: Offline Graph Analytics

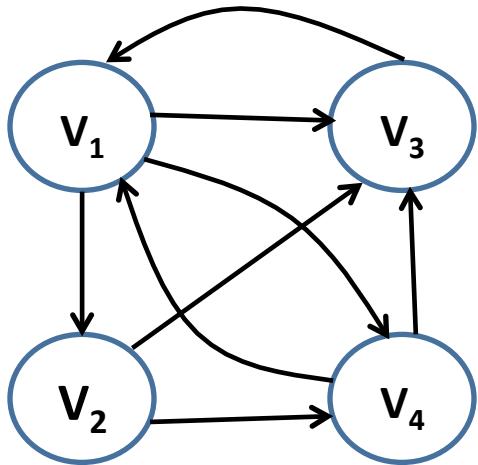


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	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

# Page Rank Computation: Offline Graph Analytics

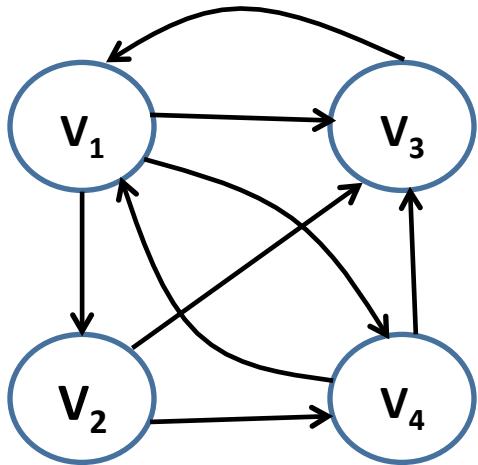


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

	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

# Page Rank Computation: Offline Graph Analytics



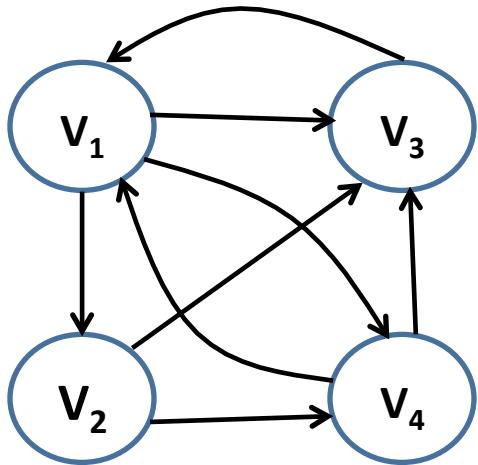
$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2
PR(V <sub>1</sub> )	0.25	0.37	0.43
PR(V <sub>2</sub> )	0.25	0.08	0.12
PR(V <sub>3</sub> )	0.25	0.33	0.27
PR(V <sub>4</sub> )	0.25	0.20	0.16

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

# Page Rank Computation: Offline Graph Analytics

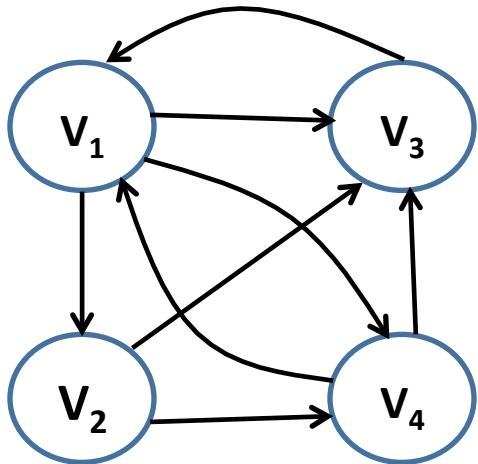


$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2	K=3
PR(V <sub>1</sub> )	0.25	0.37	0.43	0.35
PR(V <sub>2</sub> )	0.25	0.08	0.12	0.14
PR(V <sub>3</sub> )	0.25	0.33	0.27	0.29
PR(V <sub>4</sub> )	0.25	0.20	0.16	0.20

# Page Rank Computation: Offline Graph Analytics



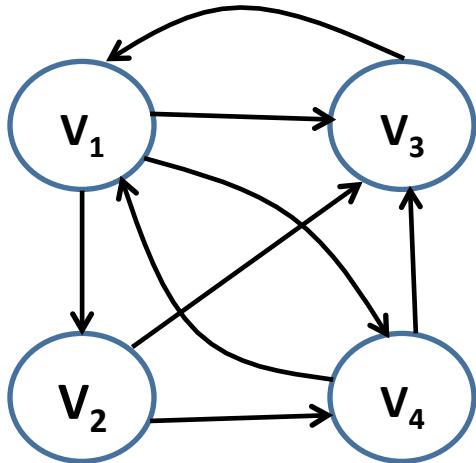
$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2	K=3	K=4
PR(v <sub>1</sub> )	0.25	0.37	0.43	0.35	0.39
PR(v <sub>2</sub> )	0.25	0.08	0.12	0.14	0.11
PR(v <sub>3</sub> )	0.25	0.33	0.27	0.29	0.29
PR(v <sub>4</sub> )	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

# Page Rank Computation: Offline Graph Analytics



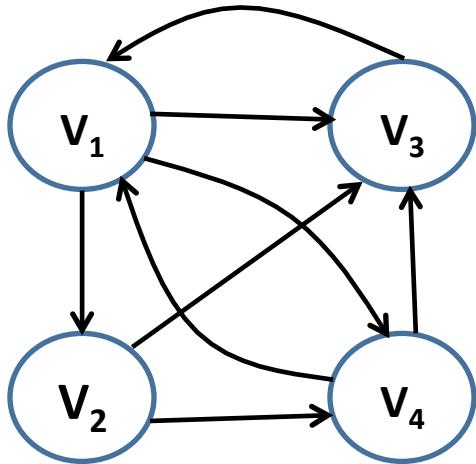
$$PR_{k+1}(u) = \sum_{v \in B_u} \frac{PR_k(v)}{|F_v|}$$

Iterative Batch Processing

	K=0	K=1	K=2	K=3	K=4	K=5
PR(V <sub>1</sub> )	0.25	0.37	0.43	0.35	0.39	0.39
PR(V <sub>2</sub> )	0.25	0.08	0.12	0.14	0.11	0.13
PR(V <sub>3</sub> )	0.25	0.33	0.27	0.29	0.29	0.28
PR(V <sub>4</sub> )	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

# Page Rank Computation: Offline Graph Analytics



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

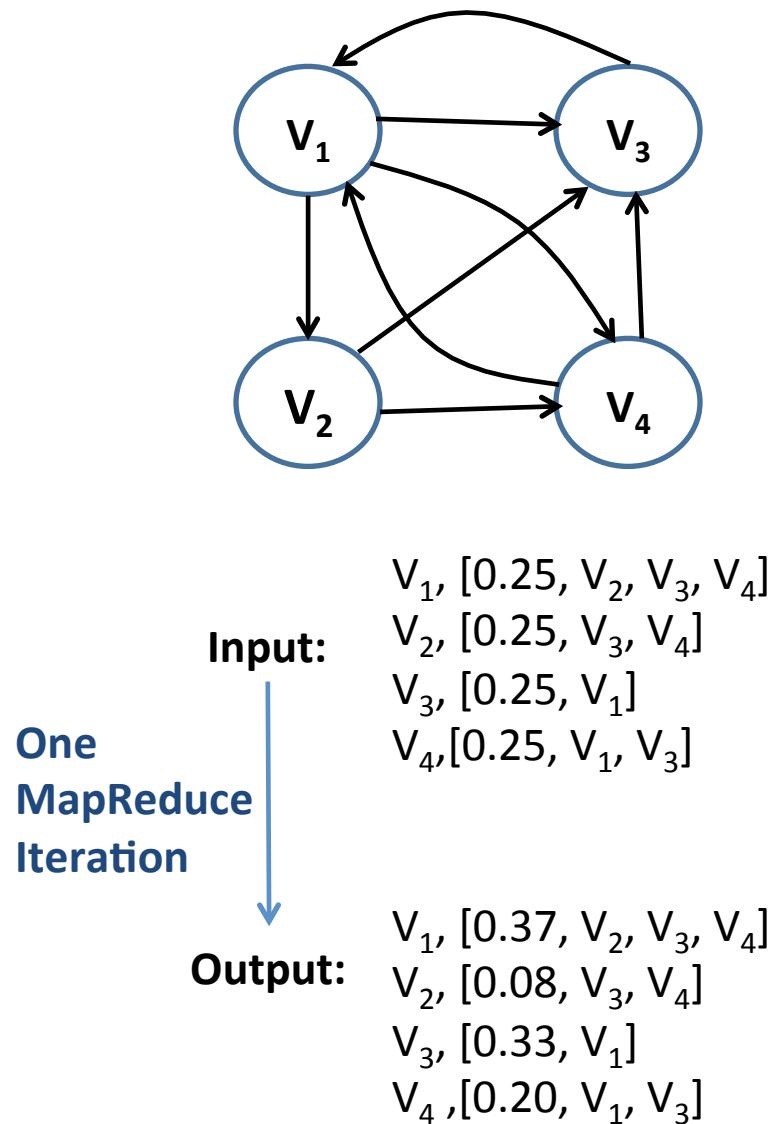
FixPoint

	K=0	K=1	K=2	K=3	K=4	K=5	K=6
PR(V <sub>1</sub> )	0.25	0.37	0.43	0.35	0.39	0.39	0.38
PR(V <sub>2</sub> )	0.25	0.08	0.12	0.14	0.11	0.13	0.13
PR(V <sub>3</sub> )	0.25	0.33	0.27	0.29	0.29	0.28	0.28
PR(V <sub>4</sub> )	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

# PageRank over MapReduce

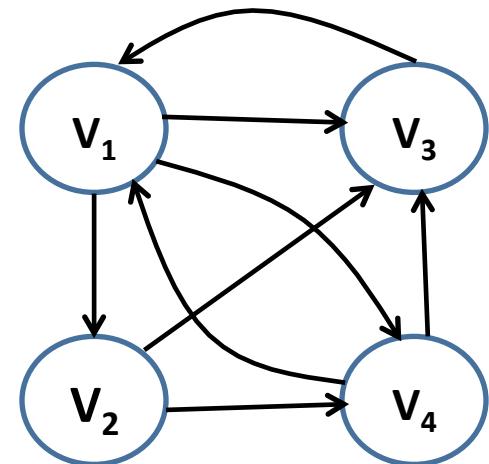
- Multiple MapReduce iterations
- Each Page Rank Iteration:
  - Input:**
    - $(id_1, [PR_t(1), out_{11}, out_{12}, \dots])$ ,
    - $(id_2, [PR_t(2), out_{21}, out_{22}, \dots])$ ,
    - ...
  - Output:**
    - $(id_1, [PR_{t+1}(1), out_{11}, out_{12}, \dots])$ ,
    - $(id_2, [PR_{t+1}(2), out_{21}, out_{22}, \dots])$ ,
    - ...
- Iterate until convergence → another MapReduce instance



# PageRank over MapReduce (One Iteration)

## Map

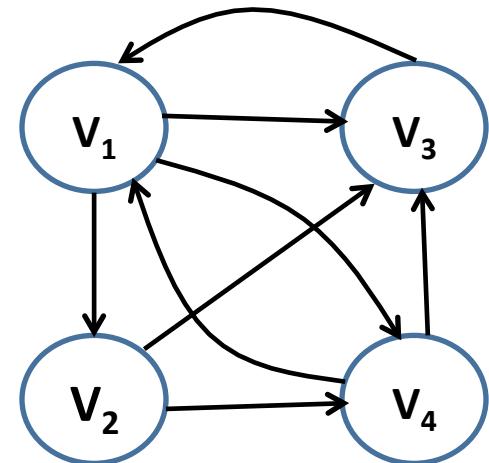
- Input:**  $(V_1, [0.25, V_2, V_3, V_4]);$   
 $(V_2, [0.25, V_3, V_4]); (V_3, [0.25, V_1]);$   
 $(V_4, [0.25, V_1, V_3])$
- Output:**  $(V_2, 0.25/3), (V_3, 0.25/3), (V_4, 0.25/3),$   
.....,  $(V_1, 0.25/2), (V_3, 0.25/2);$   
 $(V_1, [V_2, V_3, V_4]), (V_2, [V_3, V_4]), (V_3, [V_1]), (V_4, [V_1, V_3])$



# PageRank over MapReduce (One Iteration)

## Map

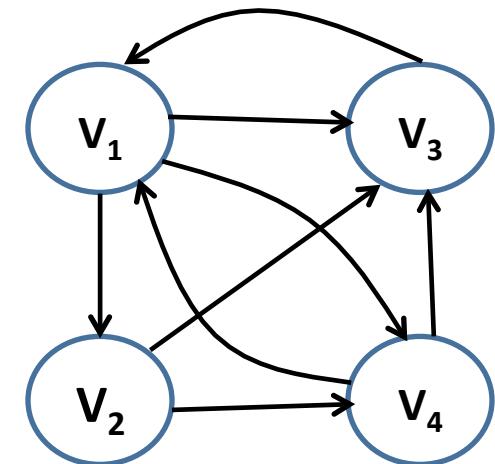
- Input:**  $(V_1, [0.25, V_2, V_3, V_4]);$   
 $(V_2, [0.25, V_3, V_4]); (V_3, [0.25, V_1]);$   
 $(V_4, [0.25, V_1, V_3])$
- Output:**  $(V_2, 0.25/3), (V_3, 0.25/3), (V_4, 0.25/3),$   
.....,  $(V_1, 0.25/2), (V_3, 0.25/2);$   
 $(V_1, [V_2, V_3, V_4]), (V_2, [V_3, V_4]), (V_3, [V_1]), (V_4, [V_1, V_3])$



# PageRank over MapReduce (One Iteration)

## Map

- Input:**  $(V_1, [0.25, V_2, V_3, V_4]);$   
 $(V_2, [0.25, V_3, V_4]); (V_3, [0.25, V_1]);$   
 $(V_4, [0.25, V_1, V_3])$
- Output:**  $(V_2, 0.25/3), (V_3, 0.25/3), (V_4, 0.25/3),$   
.....,  $(V_1, 0.25/2), (V_3, 0.25/2);$   
 $(V_1, [V_2, V_3, V_4]), (V_2, [V_3, V_4]), (V_3, [V_1]), (V_4, [V_1, V_3])$



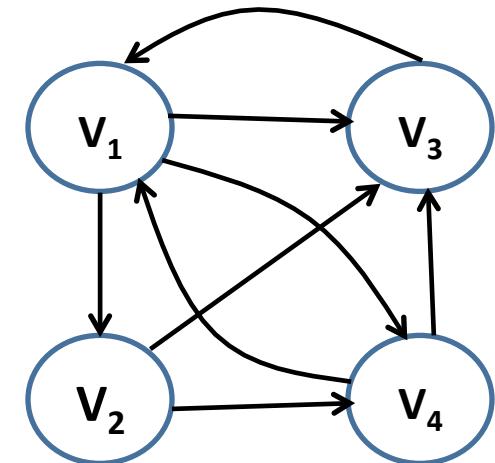
## Shuffle

- Output:**  $(V_1, 0.25/1), (V_1, 0.25/2), (V_1, [V_2, V_3, V_4]); \dots;$   
 $(V_4, 0.25/3), (V_4, 0.25/2), (V_4, [V_1, V_3])$

# PageRank over MapReduce (One Iteration)

## Map

- Input:**  $(V_1, [0.25, V_2, V_3, V_4])$ ;  
 $(V_2, [0.25, V_3, V_4])$ ;  $(V_3, [0.25, V_1])$ ;  
 $(V_4, [0.25, V_1, V_3])$
- Output:**  $(V_2, 0.25/3), (V_3, 0.25/3), (V_4, 0.25/3),$   
.....,  $(V_1, 0.25/2), (V_3, 0.25/2);$   
 $(V_1, [V_2, V_3, V_4]), (V_2, [V_3, V_4]), (V_3, [V_1]), (V_4, [V_1, V_3])$



## Shuffle

- Output:**  $(V_1, 0.25/1), (V_1, 0.25/2), (V_1, [V_2, V_3, V_4]);$  ..... ;  
 $(V_4, 0.25/3), (V_4, 0.25/2), (V_4, [V_1, V_3])$

## Reduce

- Output:**  $(V_1, [0.37, V_2, V_3, V_4]); (V_2, [0.08, V_3, V_4]); (V_3, [0.33, V_1]); (V_4, [0.20, V_1, V_3])$

# Key Insight in Parallelization (Page Rank over MapReduce)

- The ‘future’ Page Rank values depend on ‘current’ Page Rank values, but not on any other ‘future’ Page Rank values.
- ‘Future’ Page Rank value of each node can be computed in parallel.

# Problems with MapReduce for Graph Analytics

- MapReduce does not directly support iterative algorithms
  - Invariant graph-topology-data re-loaded and re-processed at each iteration → wasting I/O, network bandwidth, and CPU

**Each Page Rank Iteration:**

**Input:**

$(\text{id}_1, [\text{PR}_t(1), \text{out}_{11}, \text{out}_{12}, \dots]), (\text{id}_2, [\text{PR}_t(2), \text{out}_{21}, \text{out}_{22}, \dots]), \dots$

**Output:**

$(\text{id}_1, [\text{PR}_{t+1}(1), \text{out}_{11}, \text{out}_{12}, \dots]), (\text{id}_2, [\text{PR}_{t+1}(2), \text{out}_{21}, \text{out}_{22}, \dots]), \dots$

- Materializations of intermediate results at every MapReduce iteration harm performance
- Extra MapReduce job on each iteration for detecting if a fixpoint has been reached

# Alternative to Simple MapReduce for Graph Analytics

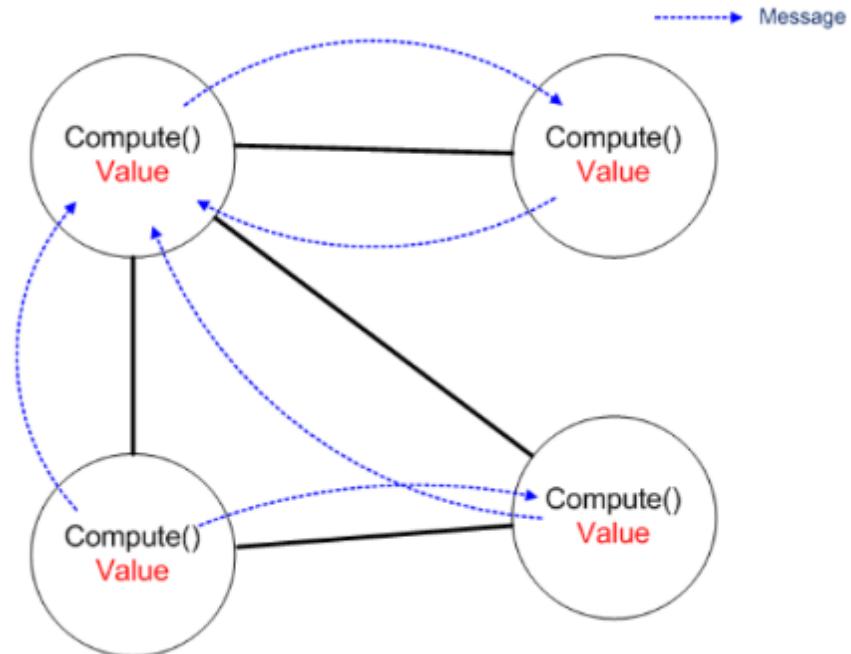
- HALOOP [Y. Bu et. al., VLDB '10]
- TWISTER [J. Ekanayake et. al., HPDC '10]
- Piccolo [R. Power et. al., OSDI '10]
- SPARK [M. Zaharia et. al., HotCloud '10]
- PREGEL [G. Malewicz et. al., SIGMOD '10]
- GBASE [U. Kang et. al., KDD '11]
- Iterative Dataflow-based Solutions: Stratosphere [Ewen et. al., VLDB '12]; GraphX [R. Xin et. al., GRADES '13]; Naiad [D. Murray et. al., SOSP'13]
- DataLog-based Solutions: SociaLite [J. Seo et. al., VLDB '13]

# Alternative to Simple MapReduce for Graph Analytics

- HALOOP [Y. Bu et. al., VLDB '10]
  - TWISTER [J. Ekanayake et. al., HPDC '10]
  - Piccolo [R. Power et. al., OSDI '10]
  - SPARK [M. Zaharia et. al., HotCloud '10]
  - **PREGEL** [G. Malewicz et. al., SIGMOD '10]
  - GBASE [U. Kang et. al., KDD '11]
  - Dataflow-based Solutions: Stratosphere [Ewen et. al., VLDB '12]; GraphX [R. Xin et. al., GRADES '13]; Naiad [D. Murray et. al., SOSP'13]
  - DataLog-based Solutions: SociaLite [J. Seo et. al., VLDB '13]
- 
- Bulk  
Synchronous  
Parallel (BSP)  
Computation

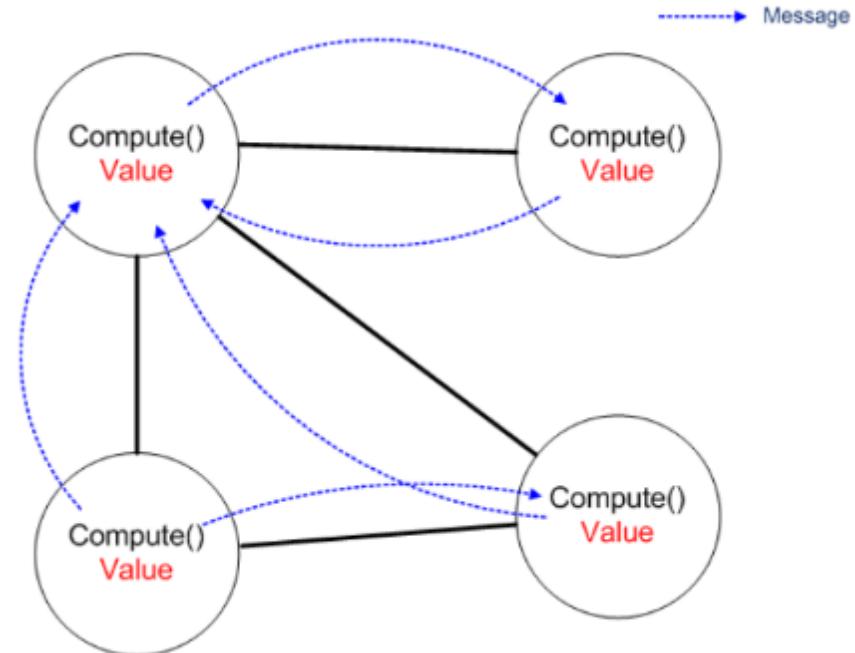
# PREGEL

- Inspired by Valiant's Bulk Synchronous Parallel (BSP) model
- Communication through message passing (usually sent along the outgoing edges from each vertex) + Shared-Nothing
- Vertex centric computation



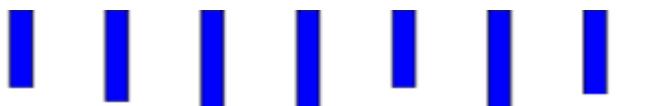
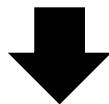
# PREGEL

- Inspired by Valiant's Bulk Synchronous Parallel (BSP) model
- Communication through message passing (usually sent along the outgoing edges from each vertex) + Shared-Nothing
- Vertex centric computation
- Each vertex:
  - Receives messages sent in the previous superstep
  - Executes the same user-defined function
  - Modifies its value
  - If active, sends messages to other vertices (received in the next superstep)
  - Votes to halt if it has no further work to do → becomes inactive
- Terminate when all vertices are inactive and no messages in transmission

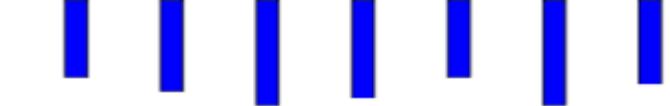
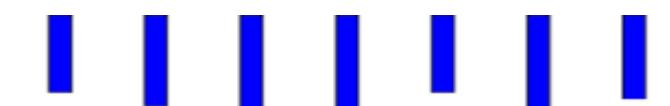


# PREGEL

Input

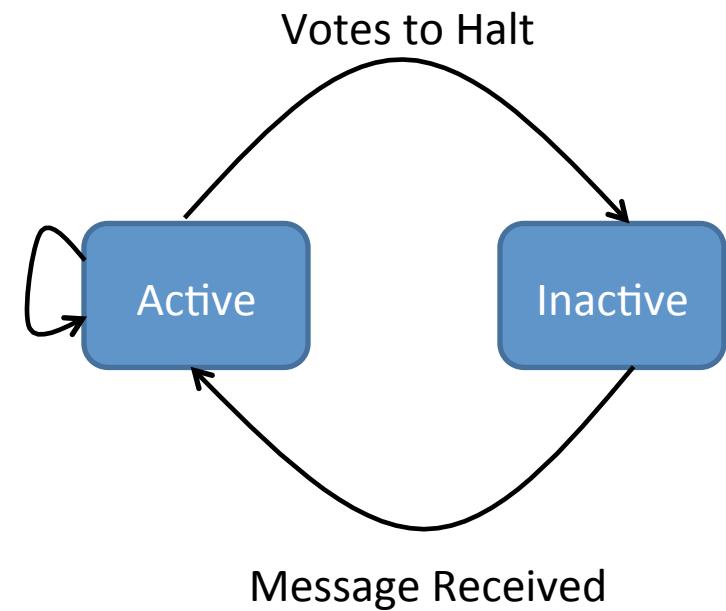


Computation  
Communication  
Superstep  
Synchronization



Output

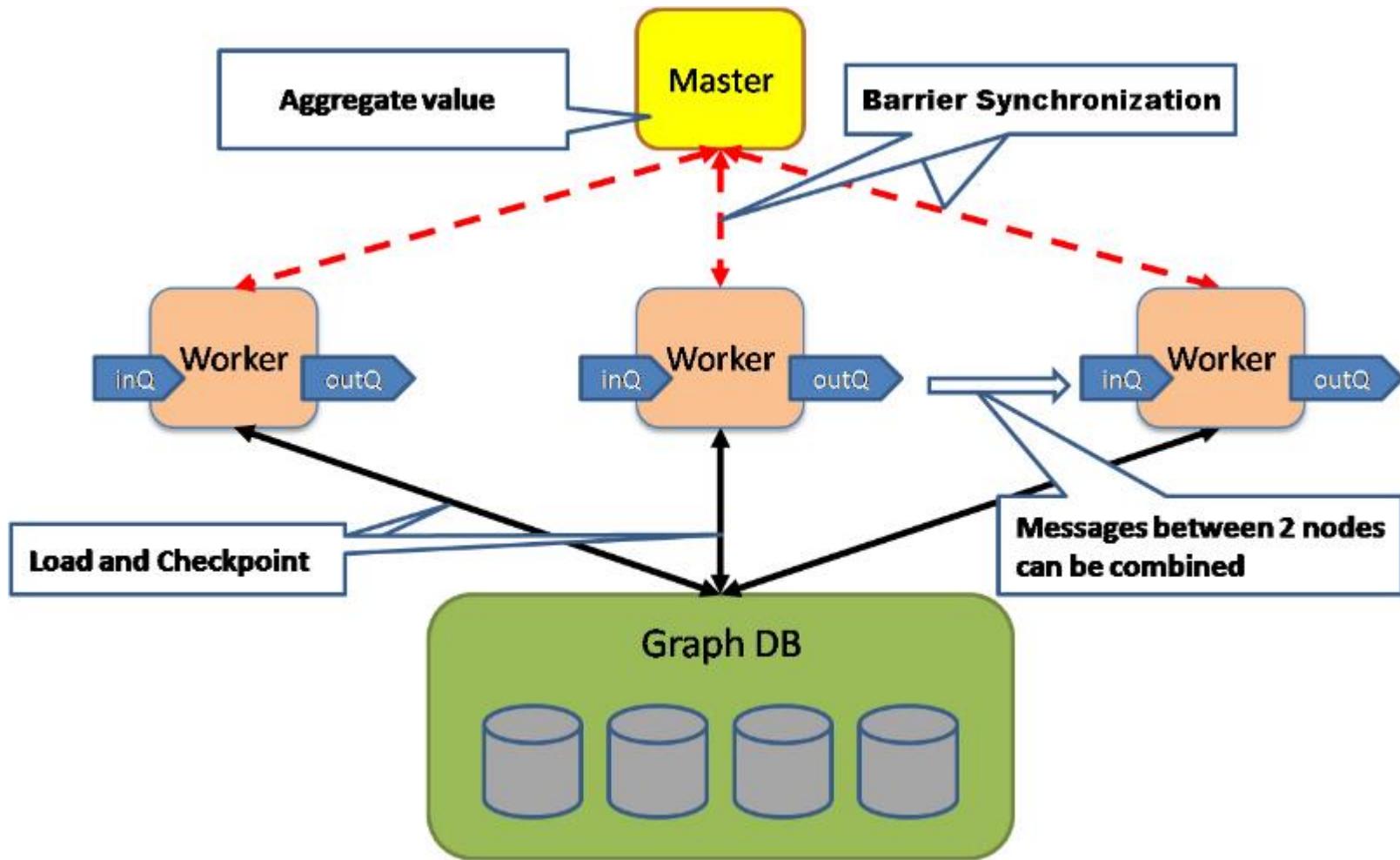
PREGEL Computation Model



**State Machine for a Vertex in PREGEL**

# PREGEL System Architecture

- Master-Slave architecture



# PageRank in Giraph (Pregel)

Suppose:  $\text{PageRank} = 0.15/\text{NUM\_VERTICES} + 0.85 * \text{SUM}$

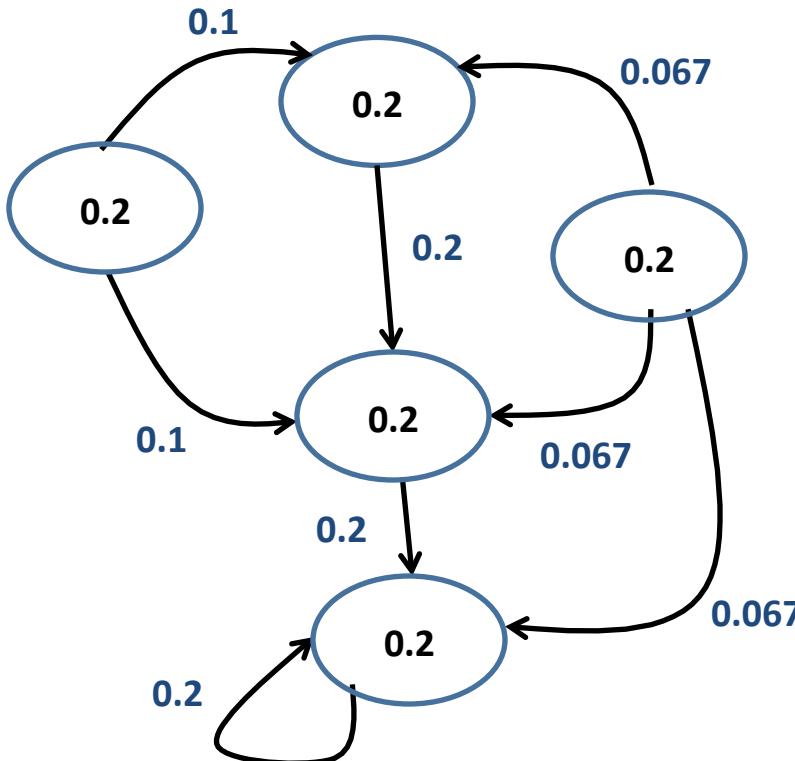
```
public void compute(Iterator<DoubleWritable> msgIterator) {  
    double sum = 0;  
    while (msgIterator.hasNext())  
        sum += msgIterator.next().get();  
    DoubleWritable vertexValue =  
        new DoubleWritable(0.15/NUM_VERTICES + 0.85 * sum);  
    setVertexValue(vertexValue);  
}
```

Sum PageRank  
over incoming  
messages

```
if (getSuperstep() < MAX_STEPS) {  
    long edges = getOutEdgeMap().size();  
    sentMsgToAllEdges(  
        new DoubleWritable(getVertexValue().get() / edges));  
} else voteToHalt();  
}
```

# Page Rank with PREGEL

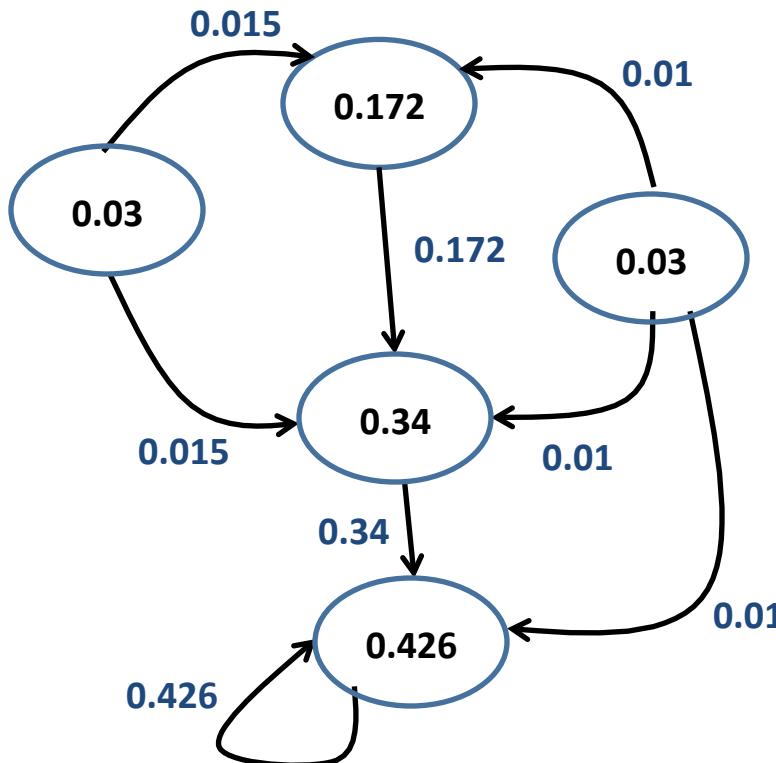
$$PR = 0.15/5 + 0.85 * \text{SUM}$$



Superstep = 0

# Page Rank with PREGEL

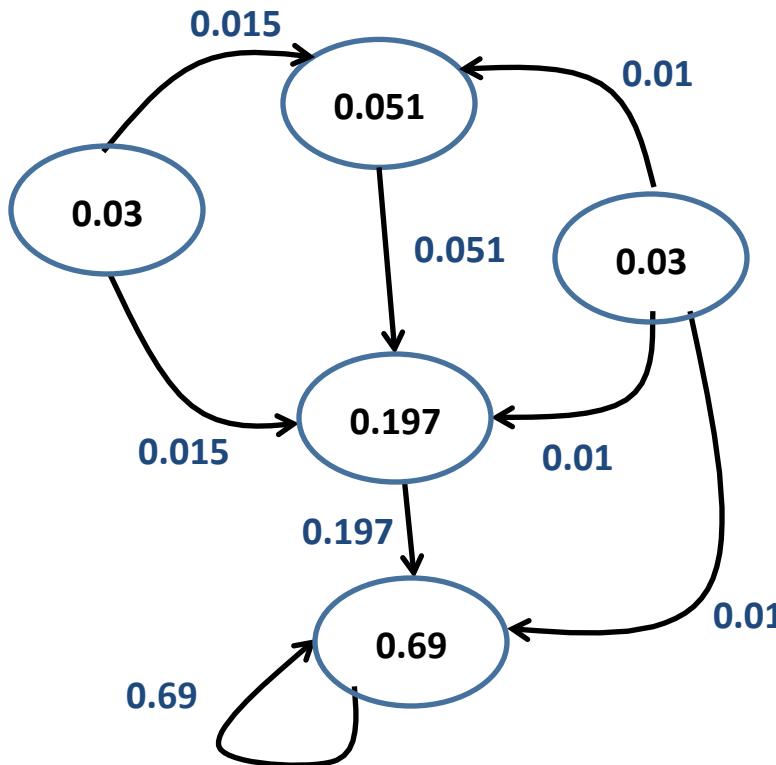
$$PR = 0.15/5 + 0.85 * \text{SUM}$$



Superstep = 1

# Page Rank with PREGEL

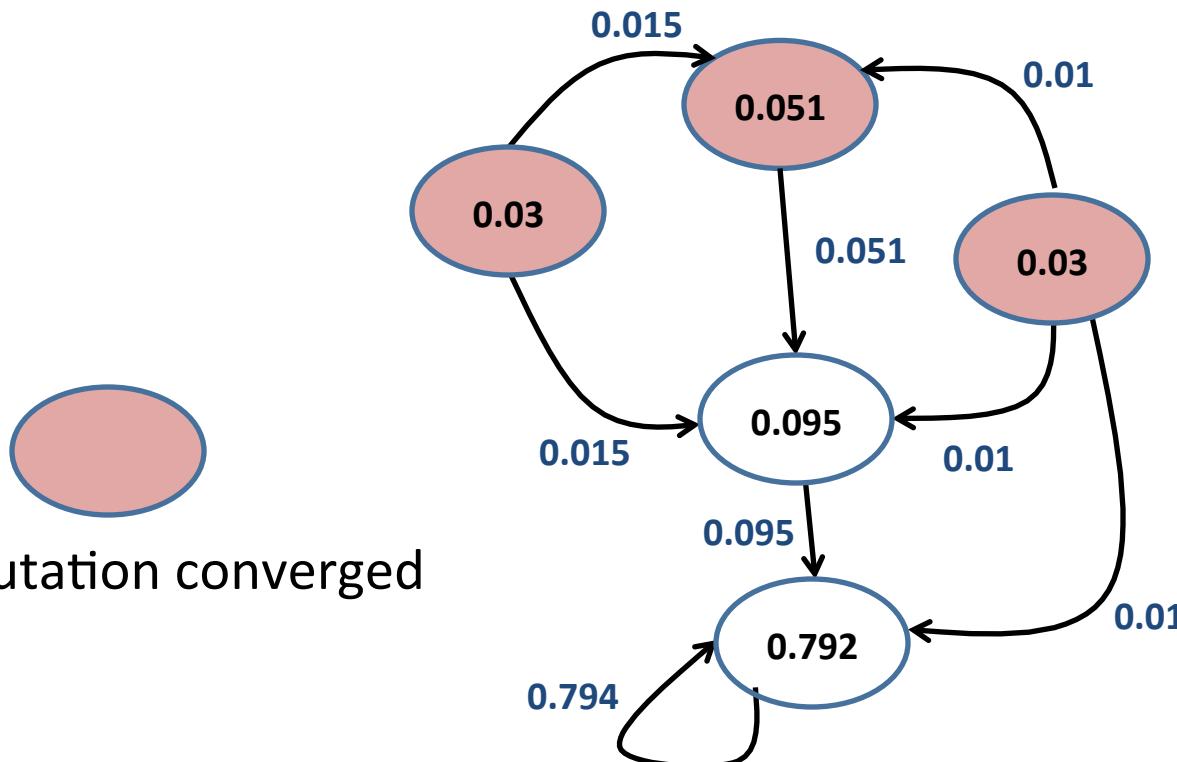
$$PR = 0.15/5 + 0.85 * \text{SUM}$$



Superstep = 2

# Page Rank with PREGEL

$$PR = 0.15/5 + 0.85 * \text{SUM}$$

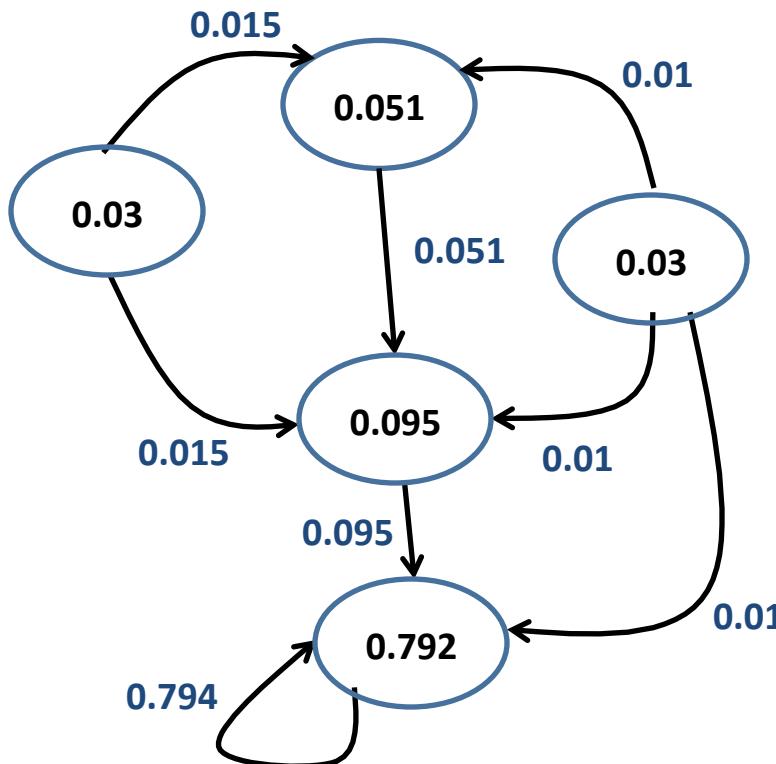


Computation converged

Superstep = 3

# Page Rank with PREGEL

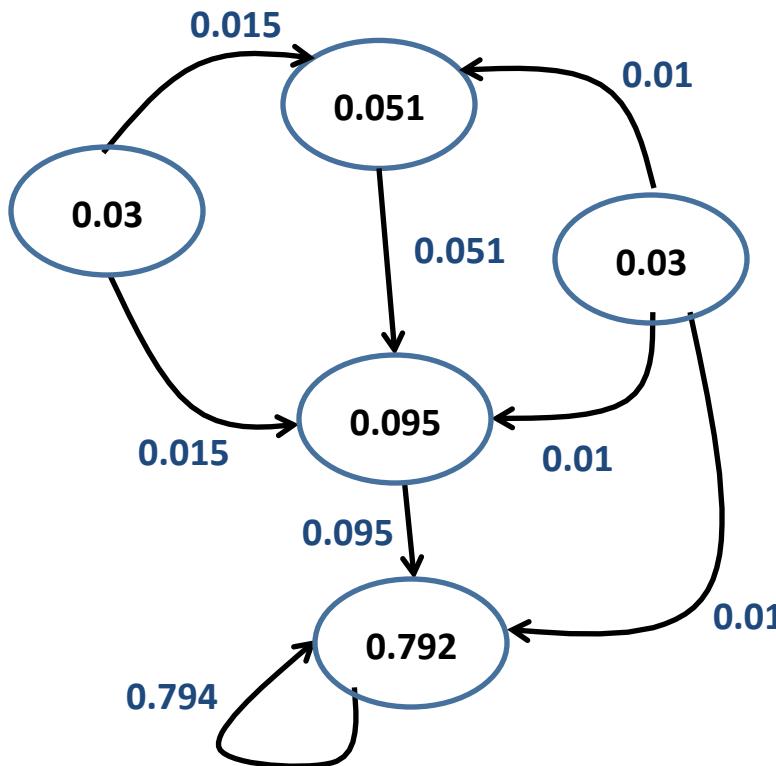
$$PR = 0.15/5 + 0.85 * \text{SUM}$$



Superstep = 4

# Page Rank with PREGEL

$$PR = 0.15/5 + 0.85 * \text{SUM}$$



Superstep = 5

# Benefits of PREGEL over MapReduce

## MapReduce

- Requires passing of entire graph topology from one iteration to the next
- Intermediate results after every iteration is stored at disk and then read again from the disk
- Programmer needs to write a driver program to support iterations; another MapReduce program to check for fixpoint

## PREGEL

- Each node sends its state only to its neighbors.  

- Main memory based  
(20X faster for k-core decomposition problem; B. Elser et. al., IEEE BigData '13)  

- Usage of supersteps and master-client architecture makes programming easy  


# Graph Algorithms Implemented with PREGEL (and PREGEL-Like-Systems)

- Page Rank
- Triangle Counting
- Connected Components
- Shortest Distance
- Random Walk
- Graph Coarsening
- Graph Coloring
- Minimum Spanning Forest
- Community Detection
- Collaborative Filtering
- Belief Propagation
- Named Entity Recognition

Not an Exclusive List

# Disadvantages of PREGEL

- In Bulk Synchronous Parallel (BSP) model, performance is limited by the slowest machine
  - Real-world graphs have power-law degree distribution, which may lead to a few highly-loaded servers

# BSP Programming Model and its Variants: Offline Graph Analytics

- **PREGEL** [G. Malewicz et. al., SIGMOD '10]
  - **GPS** [S. Salihoglu et. al., SSDBM '13]
  - **X-Stream** [A. Roy et. al., SOSP '13]
- }
- Synchronous
- 
- **GraphLab/ PowerGraph** [Y. Low et. al., VLDB '12]
  - **Grace** [G. Wang et. al., CIDR '13]
  - **SIGNAL/COLLECT** [P. Stutz et. al., ISWC '10]
  - **Giraph++** [Tian et. al., VLDB '13]
  - **GraphChi** [A. Kyrola et. al., OSDI '12]
  - **Asynchronous Accumulative Update** [Y. Zhang et. al., ScienceCloud '12], **PrIter** [Y. Zhang et. al., SOCC '11]
- }
- Asynchronous