Northeastern

SECURE FUNCTION EVALUATION AT SCALE

Stratis Ioannidis

joint work with:

Kartik Nayak,
Xiao Shaun Wang
University of Maryland

Udi Weinsberg
Facebook

Nina Taft
Google

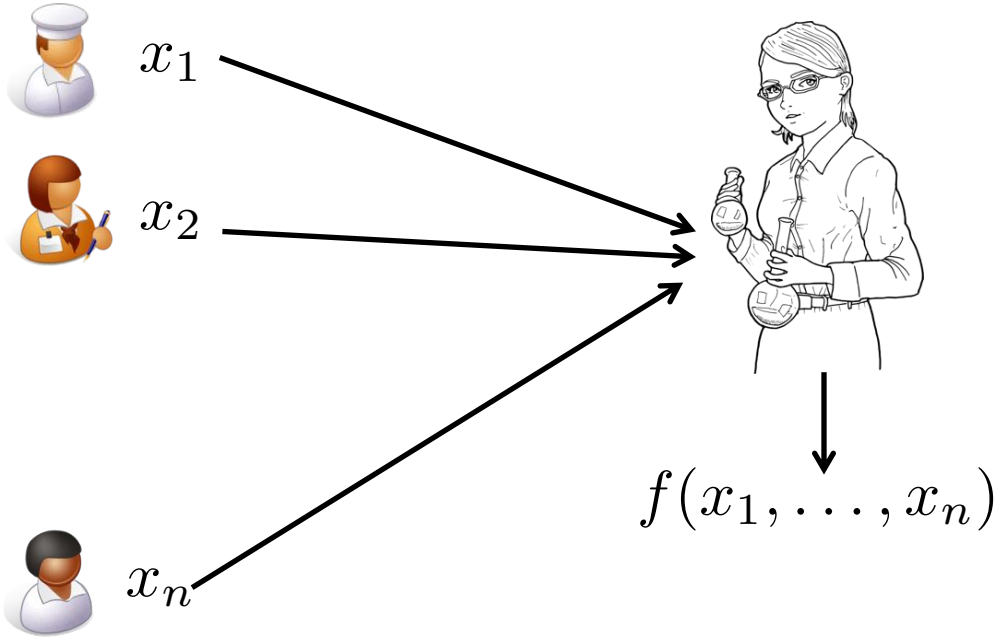
Elaine Shi
Cornell University

Analyzing Data from Human Subjects

- Long history in experimental/life sciences
  - Medicine
  - Psychology
  - Sociology
  - Behavioral Economics
  - …
- Ubiquitous practice
  - Display & search ads
  - e-Commerce
  - Streaming
  - ...
- Integral to daily operations
  - User profiling
  - Targeted advertising
  - Personalized recommendations

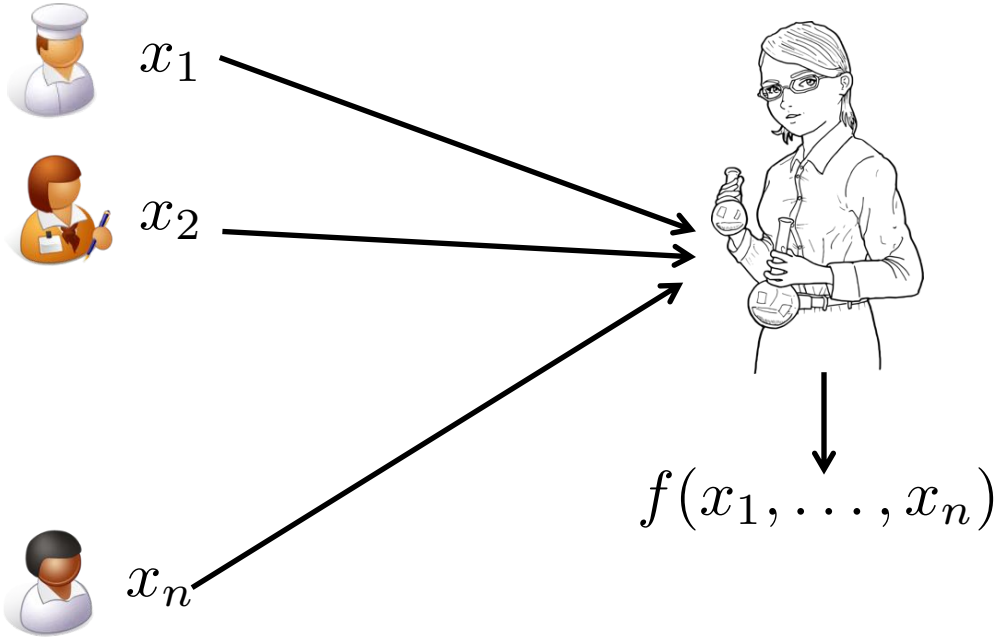
Northeastern

Secure Function Evaluation (SFE)



The analyst **learns only** $f(x_1, \ldots, x_n)$**, and nothing else.**

Grand Challenge



Apply SFE to execute **real-life**, **practical algorithms** over **massive datasets**

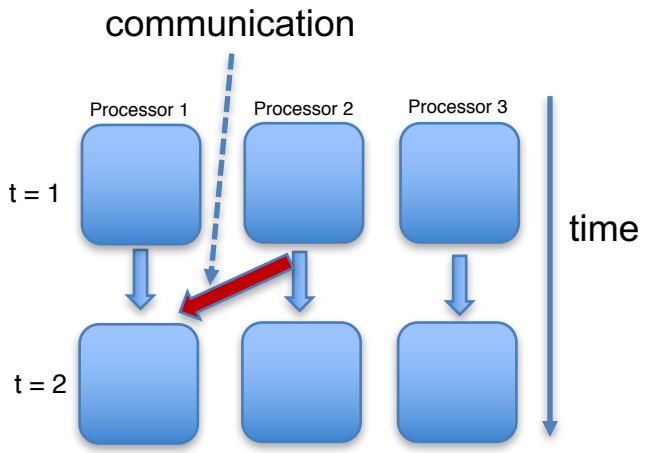
Challenge 1: Cost of Obliviousness

```
function bs(val, s, t)
  mid = (s + t) / 2;
  if (val < mem[mid])
    bs(val, 0, mid)
  else
    bs(val, mid+1, t)
```

- Translating to **data-oblivious algorithm** can **increase total work**
- For "big data", even going from $O(n)$ to $O(n^2)$ is prohibitive

Challenge 2: Maintaining Parallelism

- Very important for “big data”!
- Desirable properties:
  - Low parallel processing time
  - Low communication cost



Caveat: Communication **patterns** between processors ***may reveal something about the data***

Generic Approaches

- Oblivious RAM programs

[Gordon, Katz, Kolesnikov, Krell, Malkin, Raykova, Vahlis; CCS 2012]
[Lu, Ostrofsky; EUROCRYPT 2013]
[Gentry, Halevi, Raykova, Wichs; FOCS 2014]
[Liu, Huang, Shi, Katz, Hicks; IEEE S&P 2014]
[Zahur, Wang, Raykova, Gascon, Doerner, Evans, Katz; IEEE S&P 2016]

- Oblivious PRAM programs

[Boyle, Chung, Pass; TCC 2016]
[Chen, Lin, Tessaro; TCC 2016]

Tailored Approaches

- Pick function/algorithm of interest
  - Linear regression, matrix factorization, SVMs, LDA, …

- Apply SFE techniques such as
  - Yao's garbled circuits
  - Secret Sharing
  - Homomorphic Encryption
  - …

…

[De Cock, Dowsley, Nascimento, Newman, PMPML16]

[Takabi, Hesamifard, Ghasemi, PMPML16]

[Tian, Jayaraman, Gu, Evans, PMPML16]

[Schoppmann, Gascon, Raykova, Evans, Zahur, Doerner, Balle, PMPML16]

This Talk

[Nayak, Wang, Ioannidis, Weinsberg, Taft, Shi, IEEE S&P 2015]

- Generic approach for SFE of **graph-parallel algorithm**
  - **Scatter/Gather/Apply**
  - Includes several important DM+ML algorithms

- SFE is highly parallelizable
  - Input size: M
  - Total Work: $O(M \log^2 M)$ Blowup: $O(\log^2 M)$
  - Parallel Time: $O(\log^2 M)$

- GraphSC
  - Generic Implementation
  - MF: 1M ratings, 128 cores: 13 hours

Overview

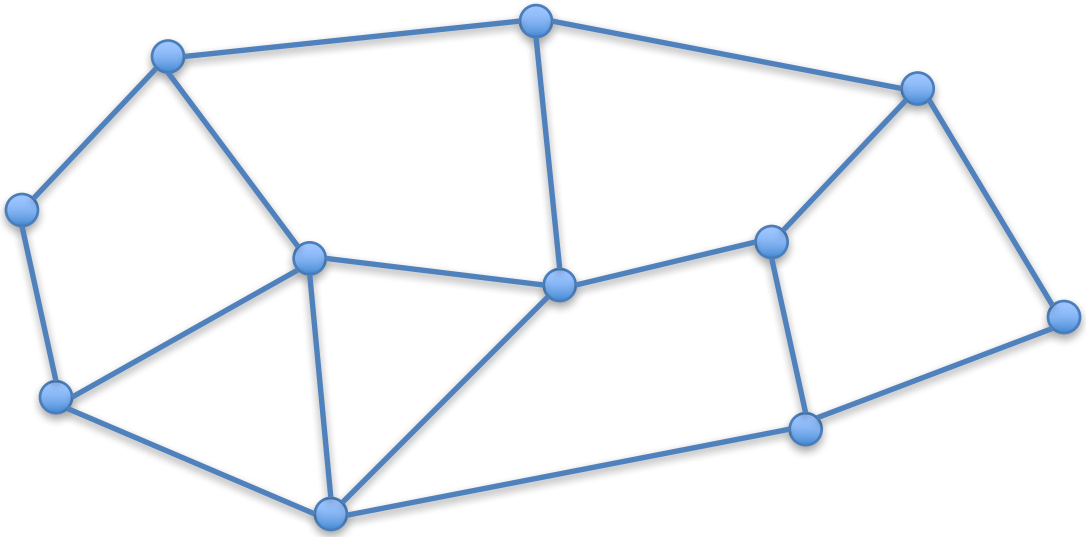
- Graph-Parallel Algorithms
- Graph-Parallel Secure Evaluation
- Implementation

Overview

- Graph-Parallel Algorithms
- Graph-Parallel Secure Evaluation
- Implementation

Graph-Parallel Algorithms

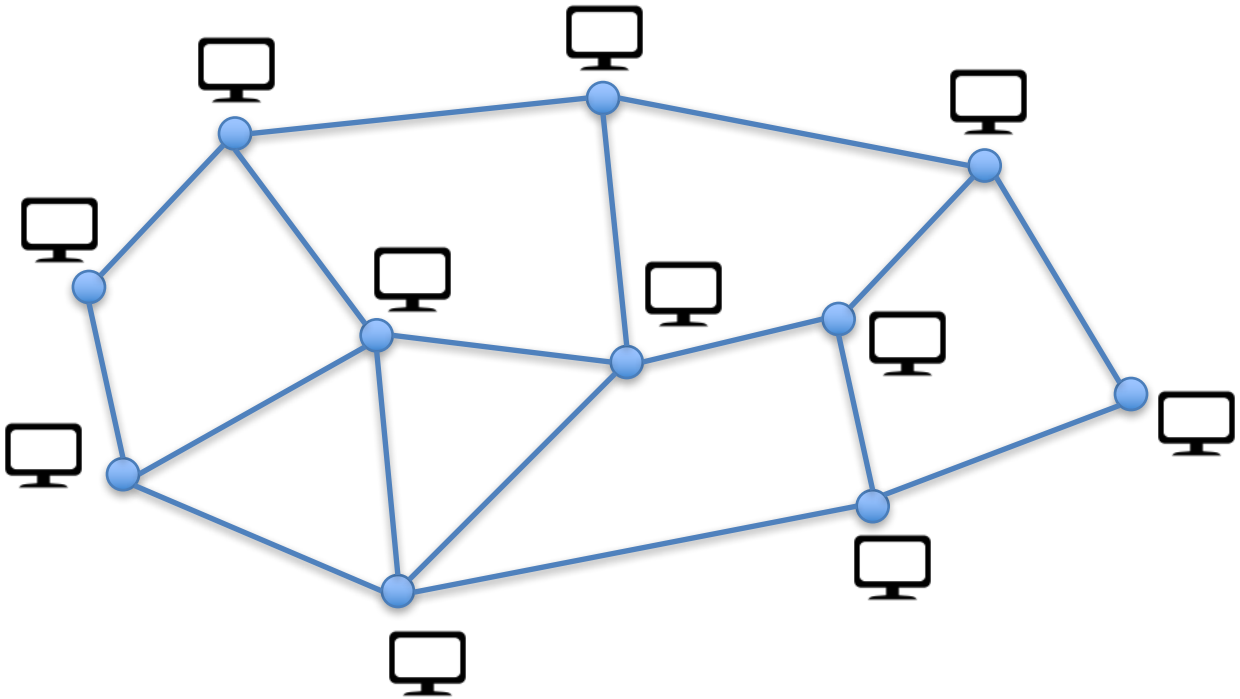
[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



Graph-Parallel Algorithm: Computation happens **over a directed graph** through **scatter**, **gather** and **apply** operations.

Graph-Parallel Algorithms

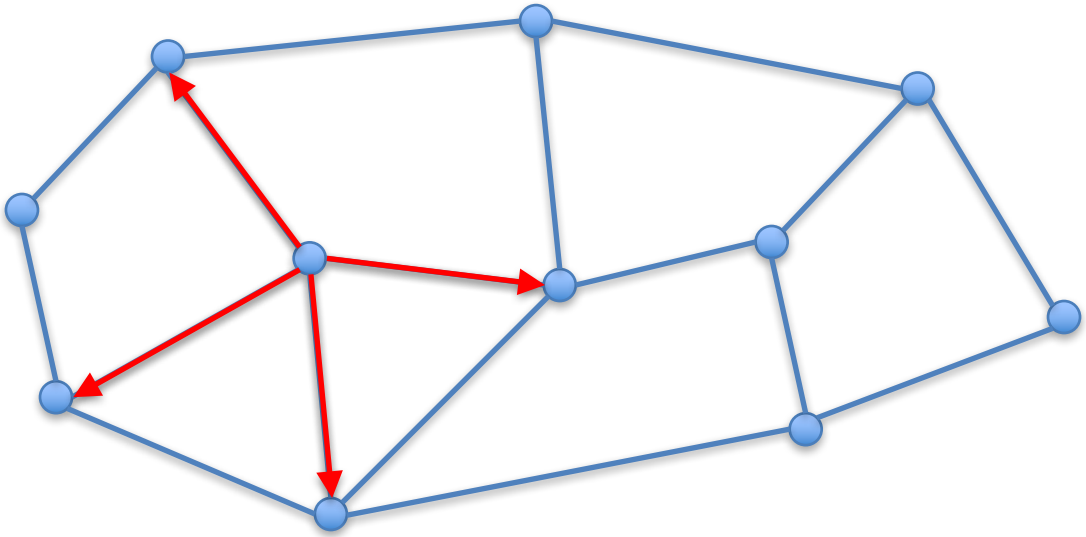
[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



Each node is a processor, edges describe communication links.

Graph-Parallel Algorithms

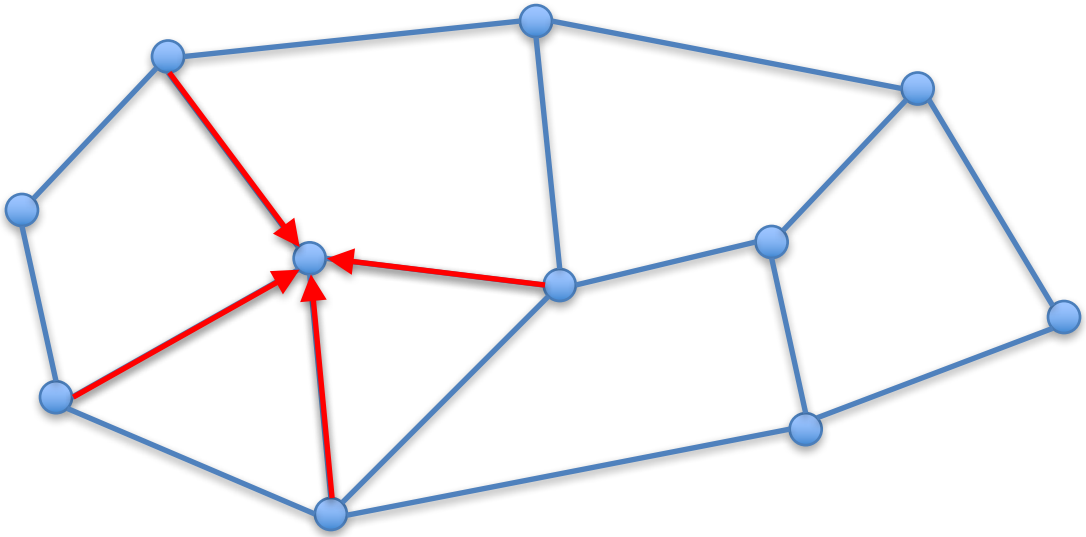
[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



Scatter: Every node sends data to its neighbors

Graph-Parallel Algorithms

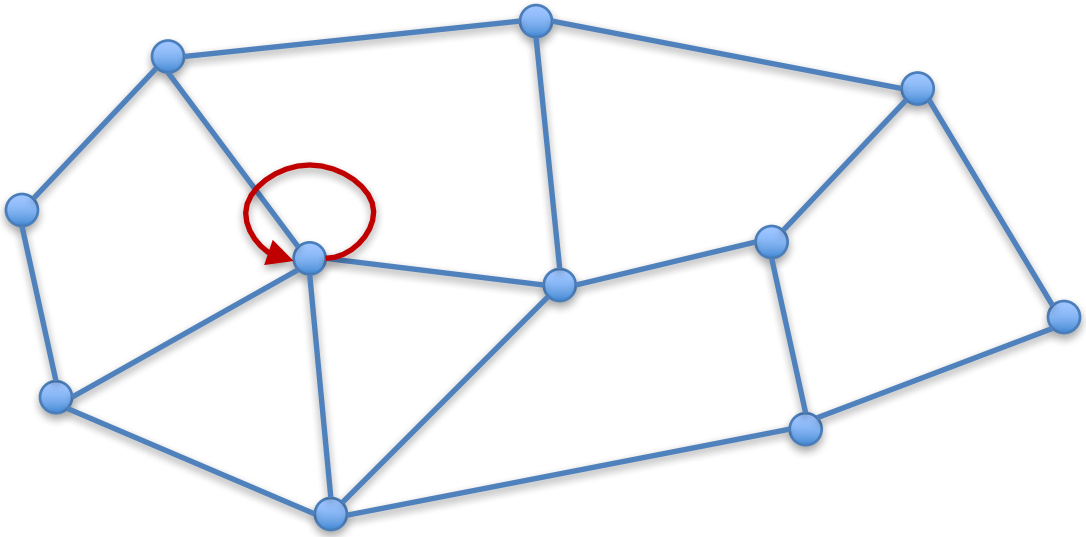
[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



Gather: Every node collects data from its neighbors and aggregates it.

Graph-Parallel Algorithms

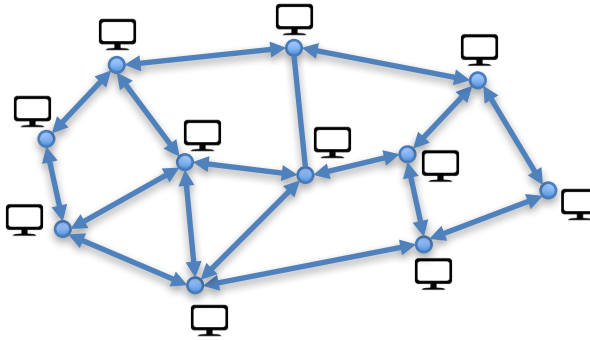
[Malewicz, Austern, Bik, Dehnert, Horn, Leiser, Czajkowski; SIGMOD 2010]
[Low, Bickson, Gonzalez, Guestrin, Kyrola, Hellerstein; PVLDB 2012]
[Ching; Hadoop Summit 2011]



Apply: Every node transforms its data locally

Graph-Parallel Algorithms

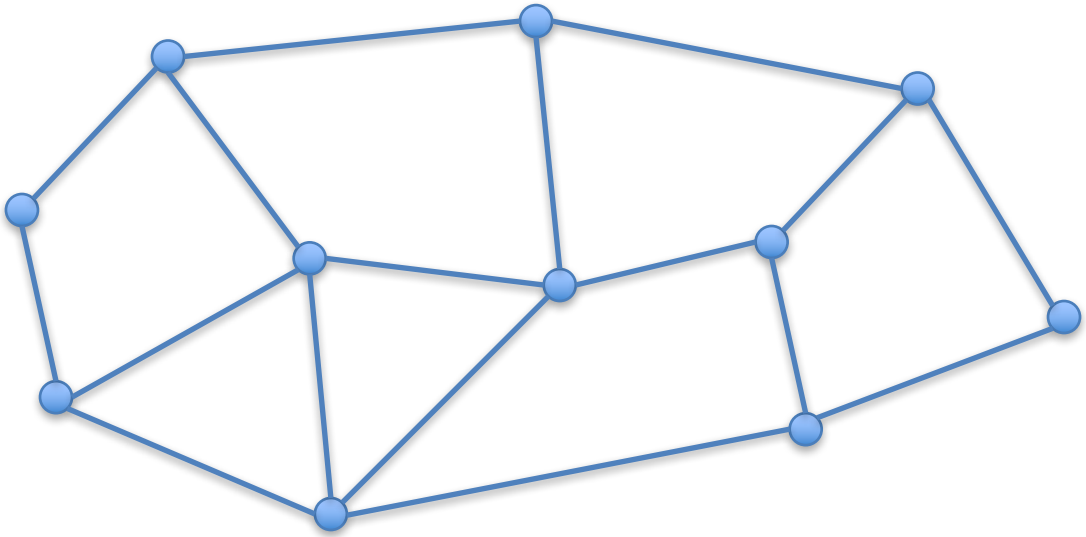
- There exist programming frameworks (e.g., GraphLab, Giraph) for parallelizing the execution of graph-parallel algorithms.



- Many interesting data mining, ML, and graph algorithms are graph-parallel:
  - Shortest paths
  - PageRank
  - Triangle counting
  - Graph coloring
  - Matrix Factorization through GD/ALS
  - ERM through GD
  - DNNs
  - …

- map, reduce, reduce-by-key operations are graph-parallel

Graph-Parallel Algorithms: Formal definition

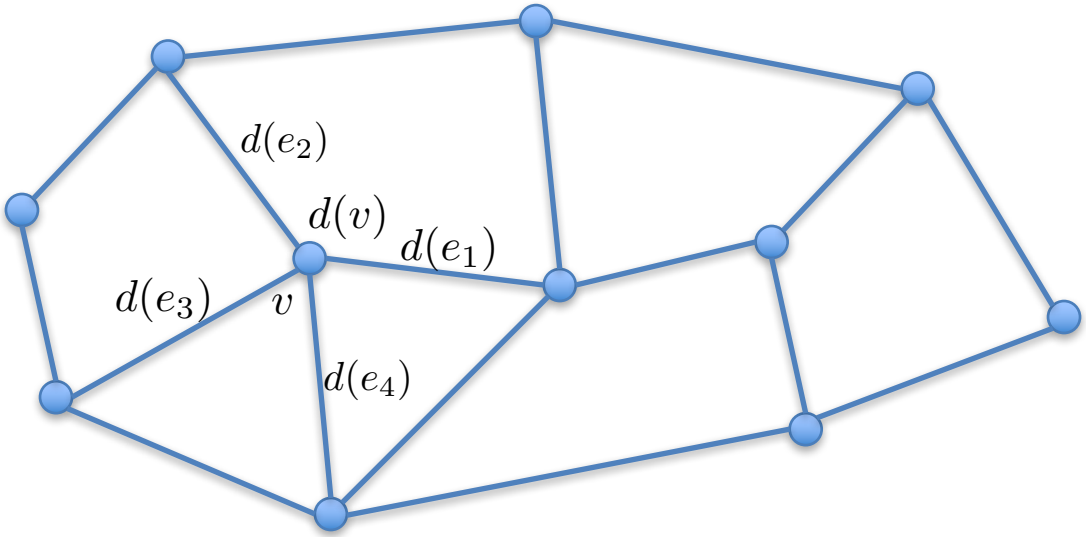


$G(V, E)$

$|E| = O(|V|)$

Consider a **directed, sparse graph** $G(V, E)$

Graph-Parallel Algorithms: Formal definition

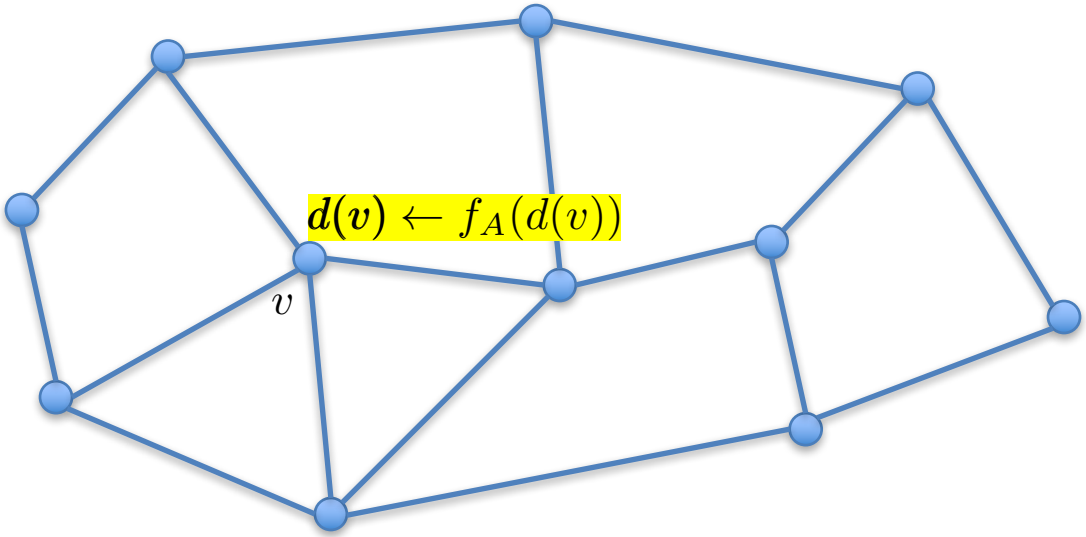


$$G(V, E)$$

$$|E| = O(|V|)$$

Both nodes **and** edges carry data.

Graph-Parallel Algorithms: Formal definition

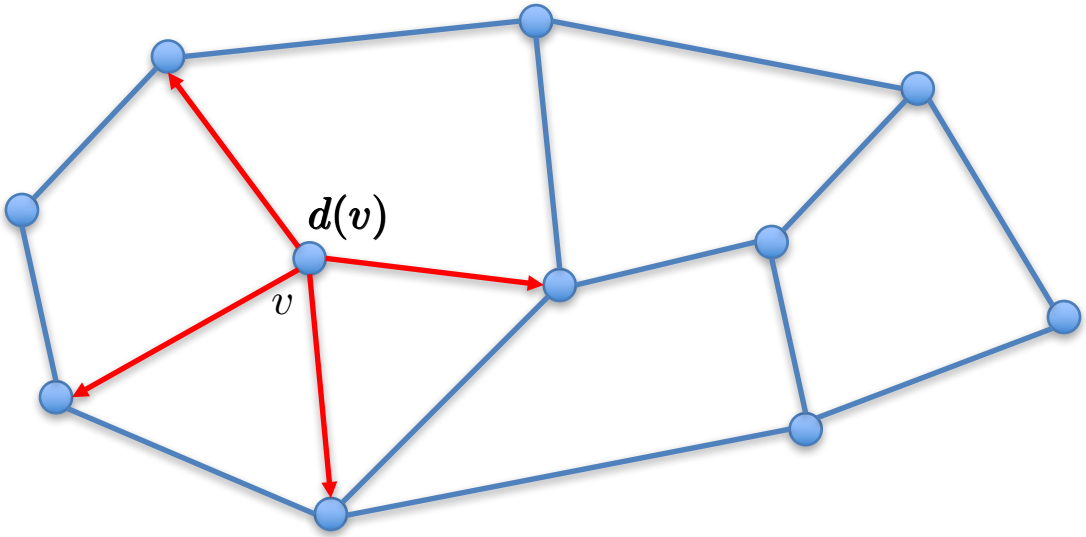


$G(V, E)$

$|E| = O(|V|)$

Apply(f_A): Every node $v \in V$ applies function f_A to their data $d(v)$ **in parallel**.

Graph-Parallel Algorithms: Formal Definition



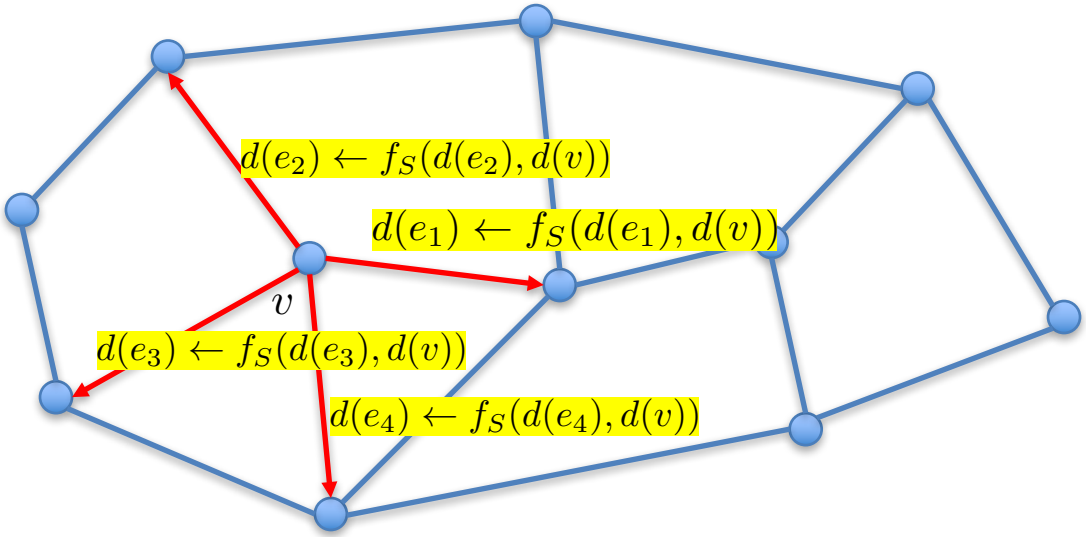
$G(V, E)$

$|E| = O(|V|)$

Scatter(f_S): Every node $v \in V$ combines their data $d(v)$ with data of adjoining edges through:

$$d(e) \leftarrow f_S(d(e), d(v)) \qquad \forall e \in N(v)$$

Graph-Parallel Algorithms: Formal Definition



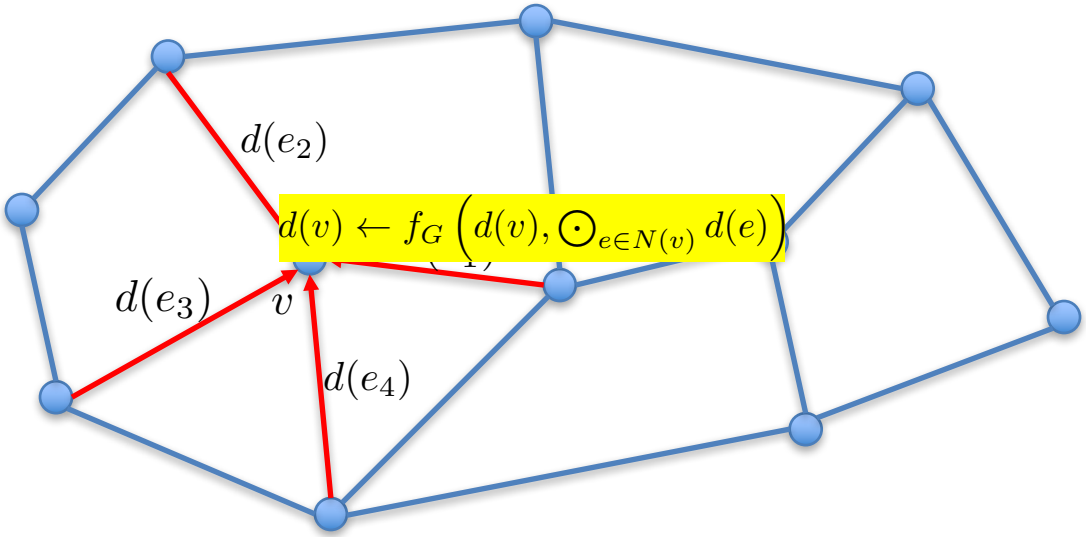
$G(V, E)$

$|E| = O(|V|)$

Scatter(f_S): Every node $v \in V$ combines their data $d(v)$ with data of adjoining edges through:

$$d(e) \leftarrow f_S(d(e), d(v)) \qquad \forall e \in N(v)$$

Graph-Parallel Algorithms: Formal Definition

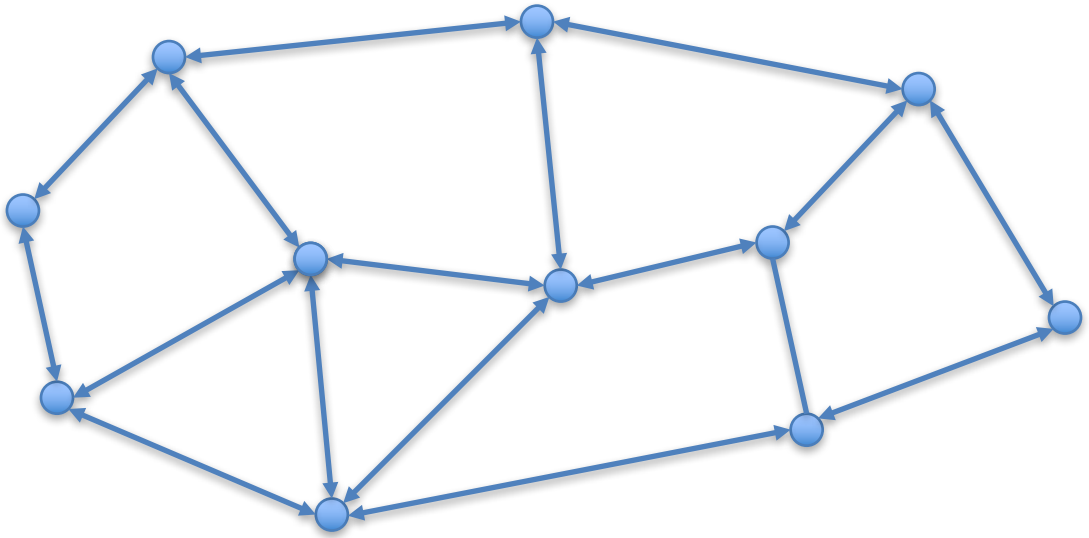


$$G(V, E)$$
$$|E| = O(|V|)$$

Gather($f_G, \odot$): Every node $v \in V$ aggregates adjacent edge data through binary operator $\odot$ and combines it with its local data through f_G. I.e.:

$$d(v) \leftarrow f_G\left(d(v), \bigodot_{e \in N(v)} d(e)\right), \quad \text{where} \quad \bigodot_{e \in N(v)} d(e) = \underbrace{d(e_1) \odot d(e_2) \odot \ldots \odot d(e_k)}_{e_i \in N(v)}$$

Example: PageRank

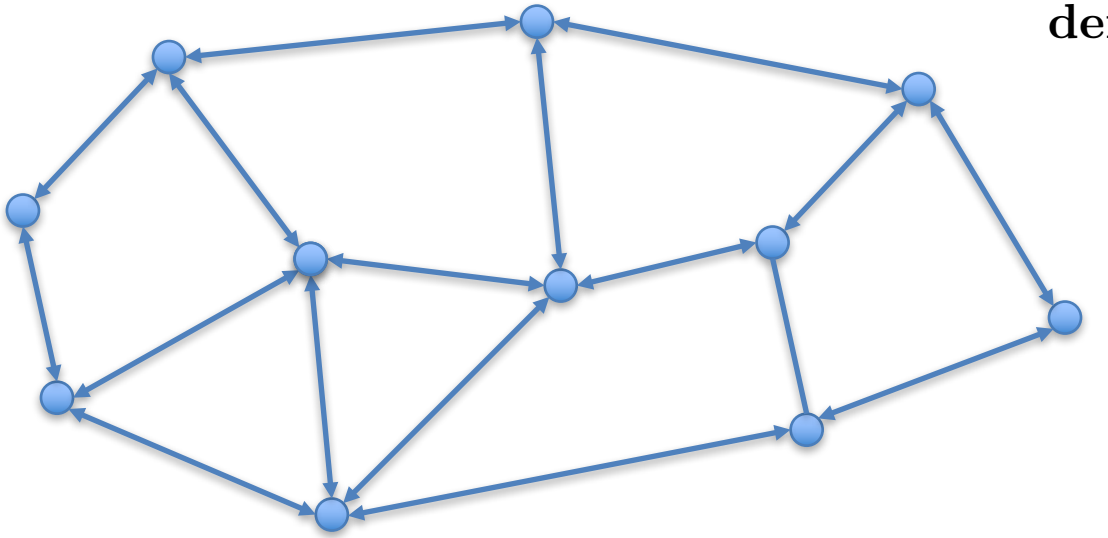


Given $G(V, E)$, repeat

$$d(v) \leftarrow \gamma \frac{1}{|V|} + (1-\gamma) \sum_{(u,v) \in N(v)} \frac{d(u)}{|N(u)|}, \quad \forall v \in V$$

until convergence

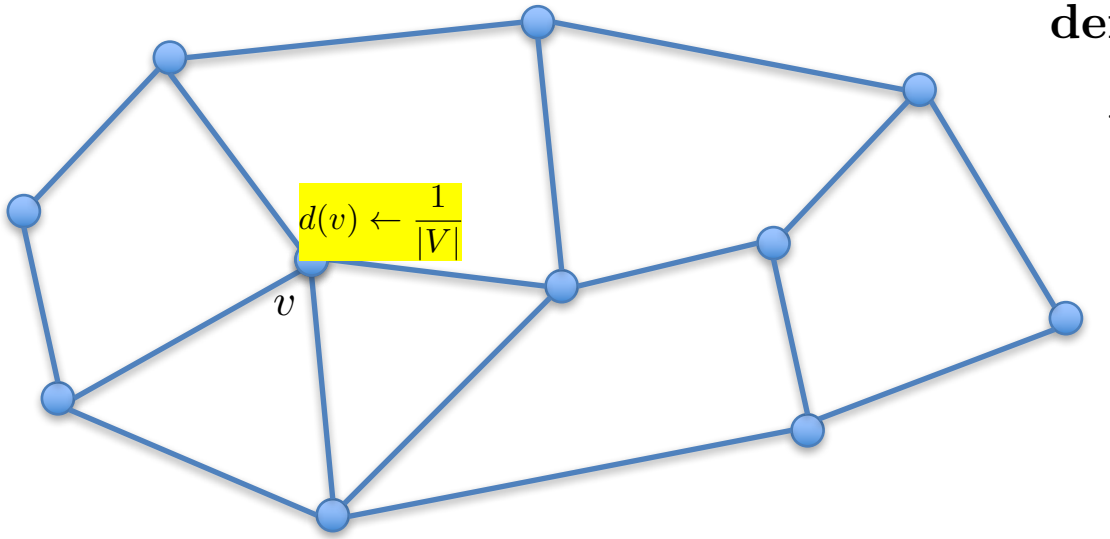
Example: PageRank



def PAGERANK(G) :

Graph $G(V, E)$ is bidirectional

Example: PageRank

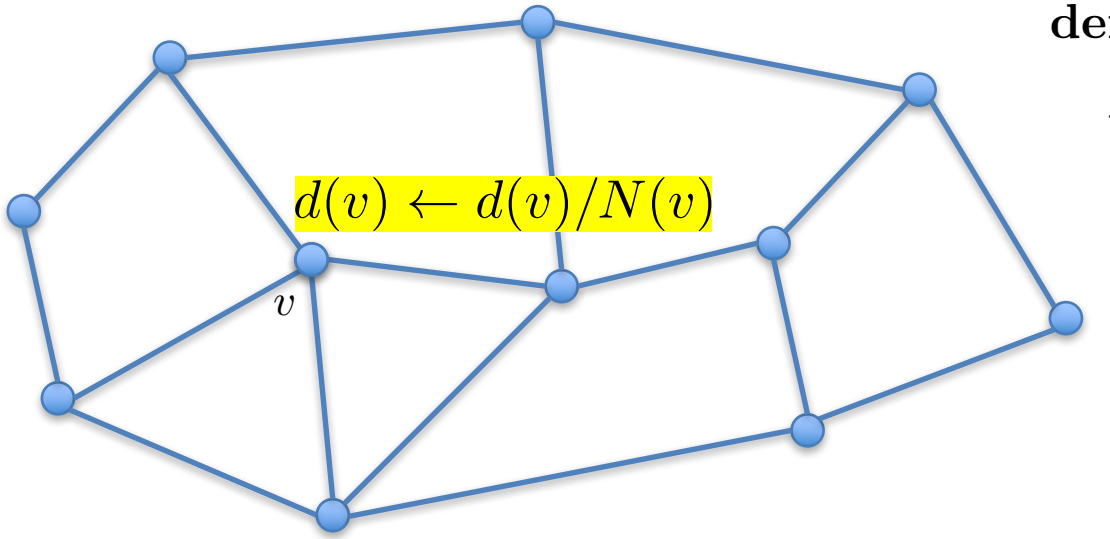


$$\textbf{def}\ \ \textsc{PageRank}(G):$$
$$\texttt{Apply}\,(d(v) \mapsto 1/|V|)$$

Nodes start with equal values:

$$d(v) = 1/|V|, \quad \forall v \in V$$

Example: PageRank



def $\textsc{PageRank}(G)$:

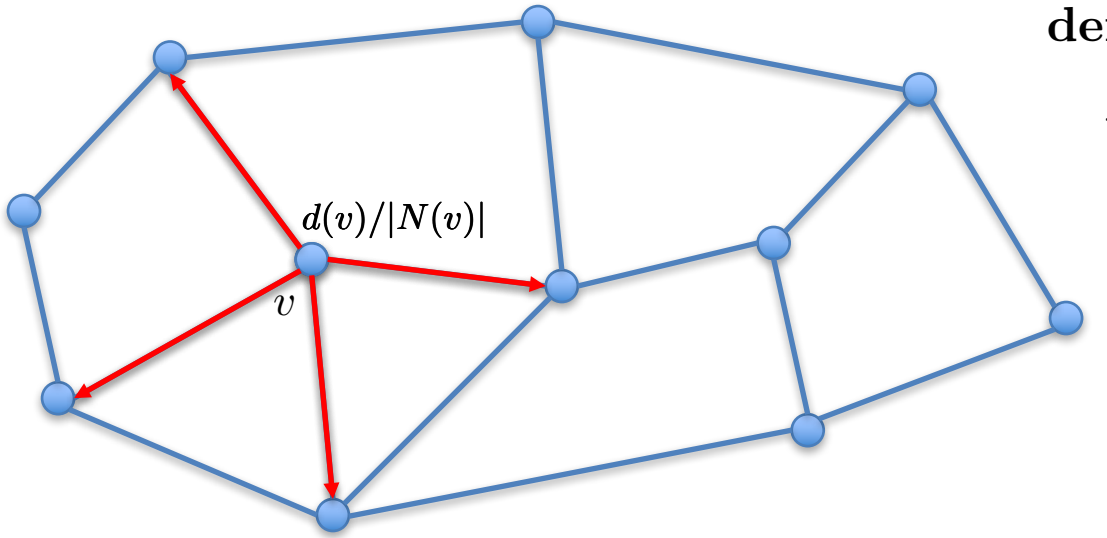
$\texttt{Apply}\,(d(v) \mapsto 1/|V|)$

repeat:

$\texttt{Apply}(d(v) \mapsto d(v)/N(v))$

In each iteration, nodes divide their values by their out-degree

Example: PageRank



$$\textbf{def}\ \ \textsc{PageRank}(G):$$

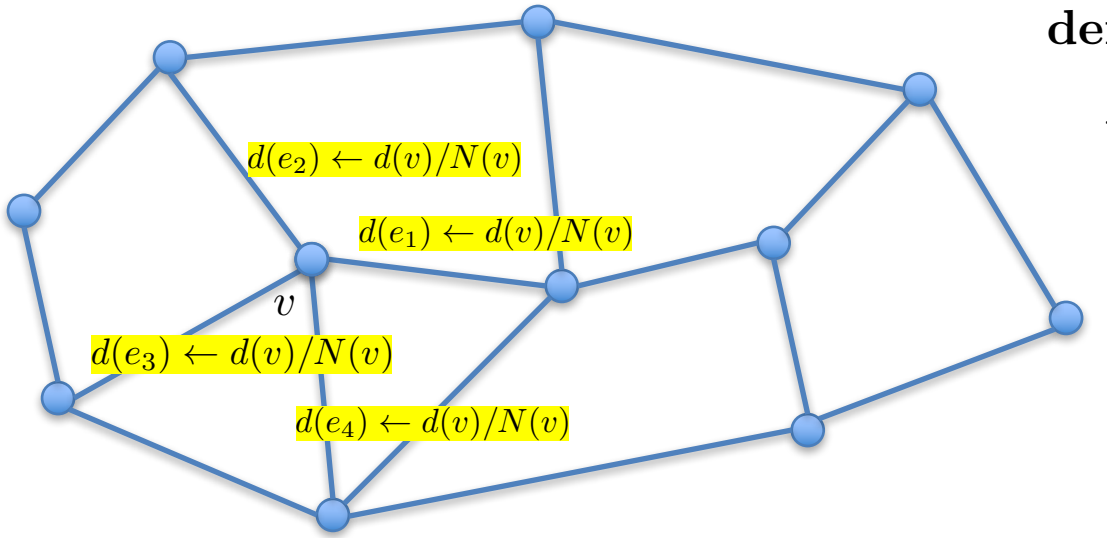
$$\texttt{Apply}\,(d(v) \mapsto 1/|V|)$$

repeat:

$$\texttt{Apply}(d(v) \mapsto d(v)/N(v))$$

Values are **scattered** over outgoing edges

Example: PageRank



def PageRank(G) :

`Apply` $(d(v) \mapsto 1/|V|)$

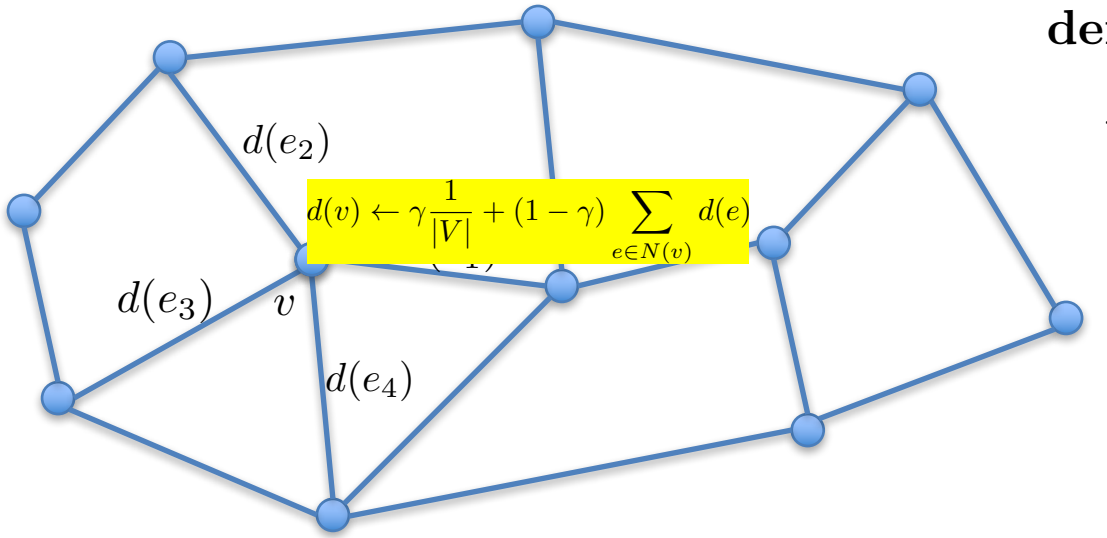
repeat:

`Apply`$(d(v) \mapsto d(v)/N(v))$

`Scatter`$((\underbrace{x, y) \mapsto x}_{f_S})$

Values are **scattered** over outgoing edges

Example: PageRank



$$\textbf{def}\ \ \textsc{PageRank}(G):$$

$$\texttt{Apply}\,(d(v) \mapsto 1/|V|)$$

repeat:

$$\texttt{Apply}(d(v) \mapsto d(v)/N(v))$$

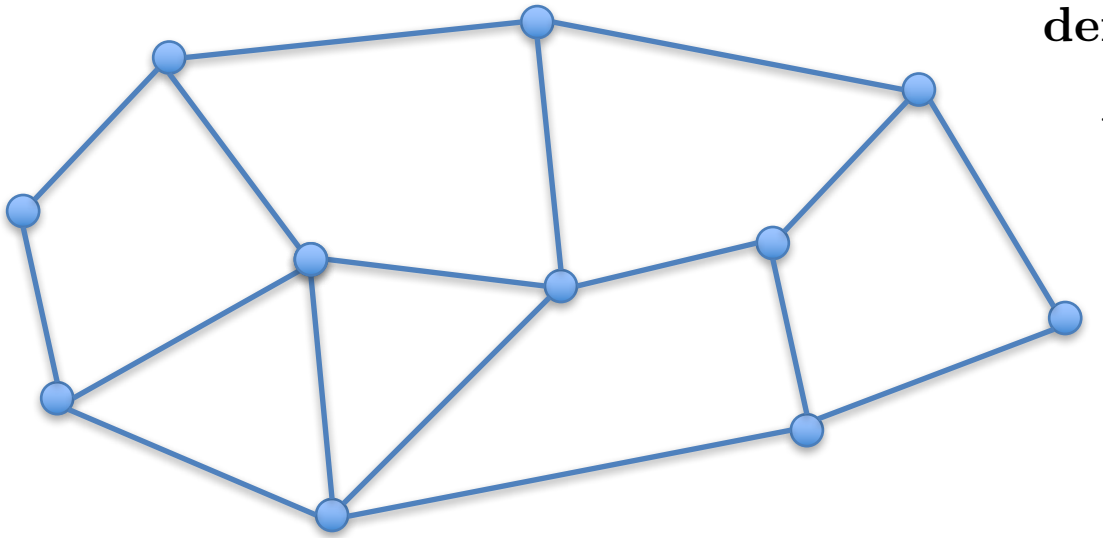
$$\texttt{Scatter}((x,y) \mapsto x)$$

$$\texttt{Gather}\Big(\underbrace{(x,y) \mapsto \gamma \frac{1}{|V|} + (1-\gamma)y}_{f_G},\ \underbrace{+}_{\odot}\Big)$$

Values in incoming edges are **gathered**, **added**, and interpolated with $1/|V|$:

$$d(v) \leftarrow \gamma \frac{1}{|V|} + (1-\gamma) \sum_{e \in N(v)} d(e)$$

Example: PageRank



$$\textbf{def}\ \ \textsc{PageRank}(G):$$

$$\texttt{Apply}\,(d(v) \mapsto 1/|V|)$$

repeat:

$$\texttt{Apply}(d(v) \mapsto d(v)/N(v))$$

$$\texttt{Scatter}((x,y) \mapsto x)$$

$$\texttt{Gather}\left((x,y) \mapsto \gamma\frac{1}{|V|} + (1-\gamma)y,\ +\right)$$

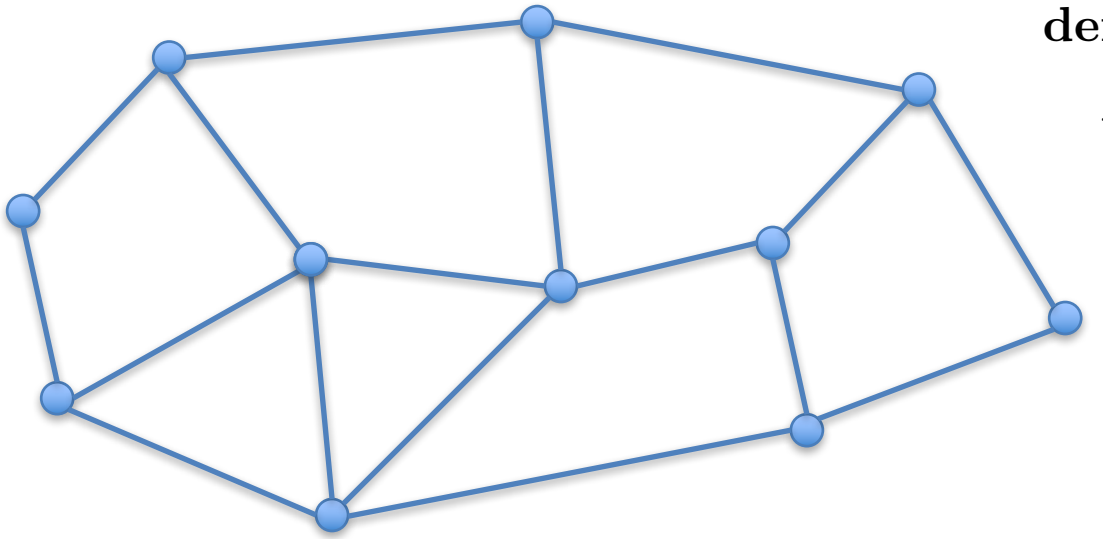
until convergence

Each iteration thus implements

$$d(v) \leftarrow \gamma\frac{1}{|V|} + (1-\gamma)\sum_{(u,v)\in N(v)}\frac{d(u)}{|N(u)|}, \quad \forall v \in V$$

repeated until convergence

Example: PageRank



def PageRank(G) :

Apply $(d(v) \mapsto 1/|V|)$

repeat:

Apply$(d(v) \mapsto d(v)/N(v))$

Scatter$((x, y) \mapsto x)$

Gather$\left((x, y) \mapsto \gamma \frac{1}{|V|} + (1-\gamma)y, +\right)$

until convergence

Input size: $M = |V| + |E| = O(|V|)$

Total work per iteration: $O(M)$

Messages per iteration: $O(M)$

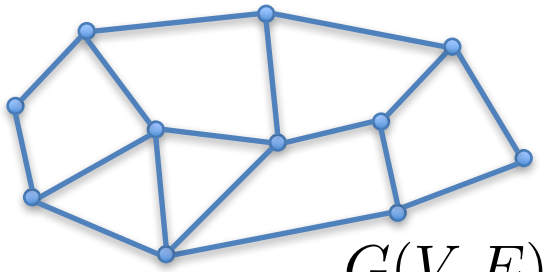
Communication pattern reveals input!

Graph-Parallel Algorithms: Definition

Definition: An algorithm is **graph-parallel** if there exist:
- a **sparse graph** $G(V, E)$ and
- initial **node** and **edge** values,

such that the algorithm can be written as a sequence of
- **Scatter**(f_S),
- **Gather**(f_G ,$\odot$), and
- **Apply**(f_A) operations,

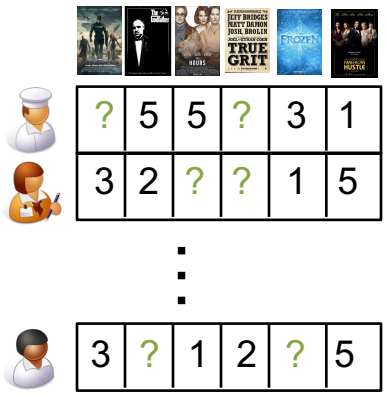
for appropriate f_S, f_G ,f_A , and $\odot$



Input size: $M = |V| + |E| = O(|V|)$
Total work per operation: $O(M)$
Messages per operation: $O(M)$

Example: Matrix Factorization

Dataset $\mathcal{D}$



r_{ij} : rating by user i to item j.

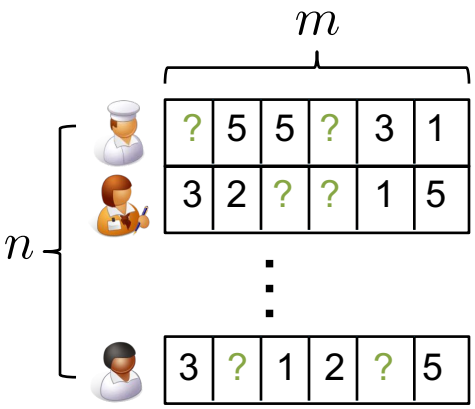
$r_{ij} = \langle u_i, v_j \rangle + \varepsilon_{ij}$, where $u_i \in \mathbb{R}^d$, $v_j \in \mathbb{R}^d$.

user profile — item profile

- Prediction: $\hat{r}_{ij} = \langle u_i, v_j \rangle$
- LSE: $(U, V) = \underset{U \in \mathbb{R}^{n \times d}, V \in \mathbb{R}^{m \times d}}{\arg\min} \sum_{(i,j,r_{ij}) \in \mathcal{D}} (r_{ij} - \langle u_i, v_j \rangle)^2$

Example: Matrix Factorization

Dataset $\mathcal{D}$



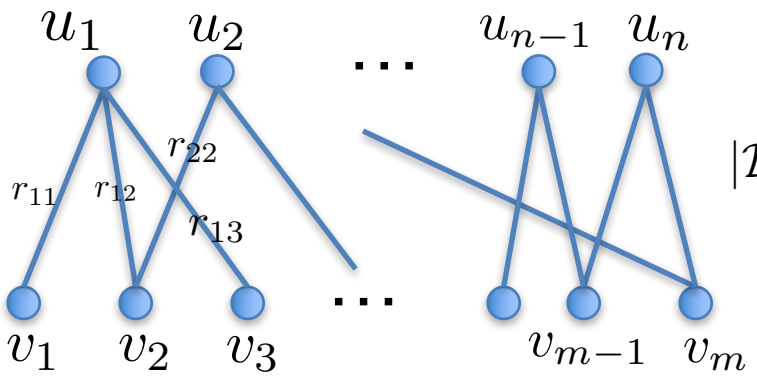
$$|\mathcal{D}| = O(n+m)$$

❑ Gradient Descent:

$$u_i \leftarrow u_i + \gamma \cdot \sum_{j:(i,j,r_{ij})\in\mathcal{D}} (r_{ij} - \langle u_i, v_j \rangle) v_j$$

$$v_j \leftarrow v_j + \gamma \cdot \sum_{i:(i,j,r_{ij})\in\mathcal{D}} (r_{ij} - \langle u_i, v_j \rangle) u_i$$

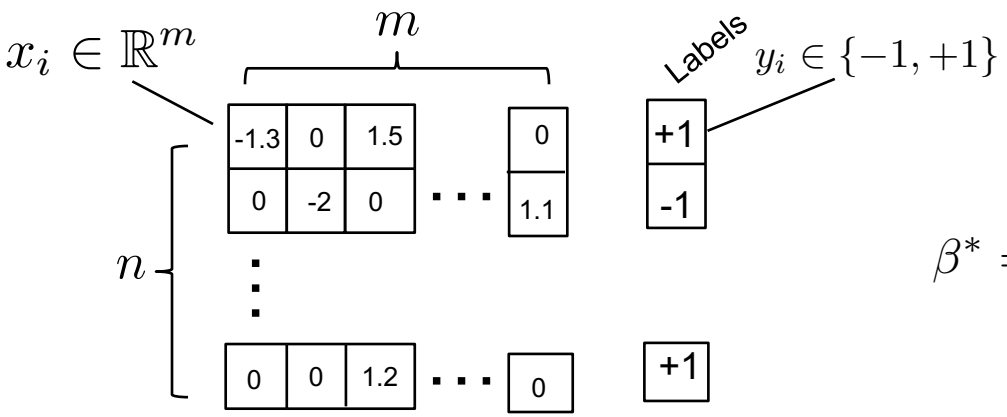
Communication pattern reveals who rated what!



$|\mathcal{D}|$ edges

Can be quite revealing [Weinsberg, Bhagat, Ioannidis, Taft, RecSys 2012]

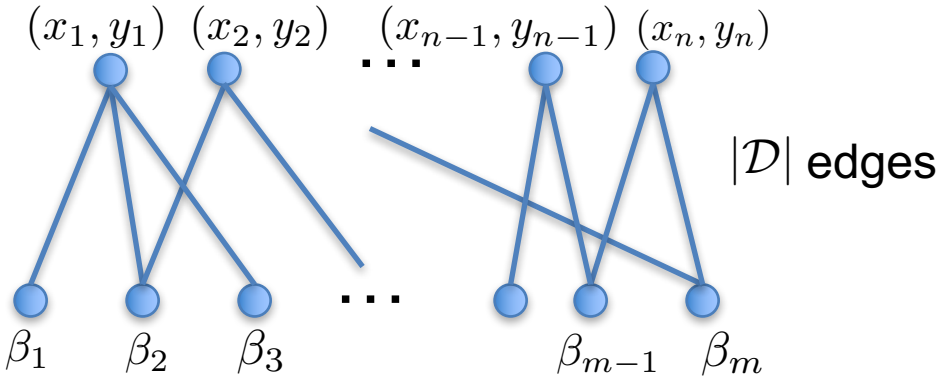
Example: High-Dimensional, Sparse ERM



$$\beta^* = \arg\min_{\beta \in \mathbb{R}^m} \sum_{i=1}^{n} \ell(\langle x_i, \beta \rangle, y_i) + \lambda \|\beta\|_2^2$$

Non-zero elements: $|\mathcal{D}| = O(n + m)$

Gradient Descent: $\beta_j \leftarrow \beta_j - \gamma \left(\sum_{i: x_{ij} \neq 0} \ell'(\langle x_i, \beta \rangle, y_i) \cdot x_{ik} + 2\lambda \beta_j \right), \qquad \forall j \in \{1, \ldots, m\}$



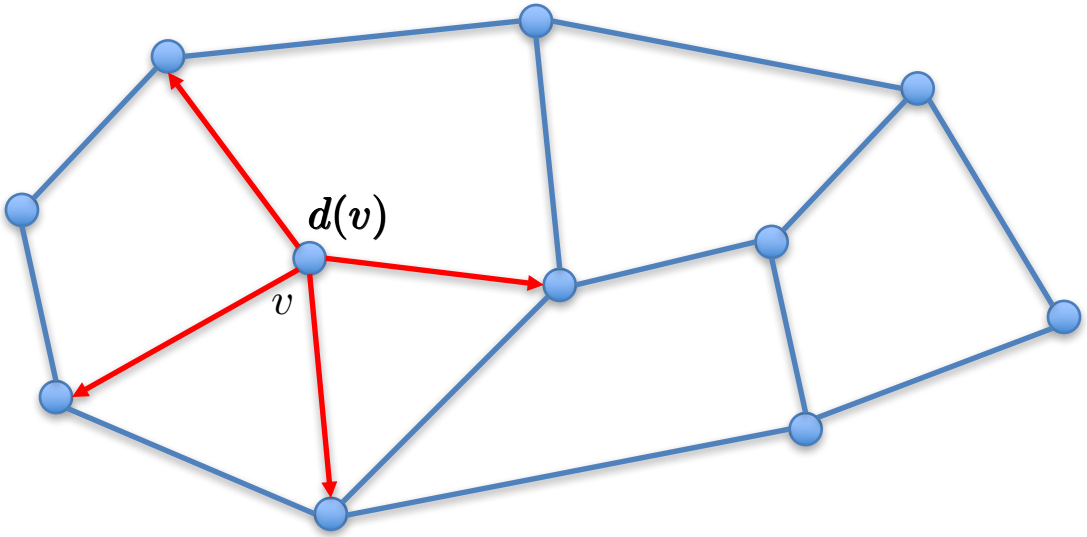
$|\mathcal{D}|$ edges

Communication pattern reveals features present!

Overview

- Graph-Parallel Algorithms
- **Graph-Parallel Secure Evaluation**
- Implementation

Naïve Data Oblivious Algorithm



$G(V, E)$

Scatter data to everyone, **gather** from all

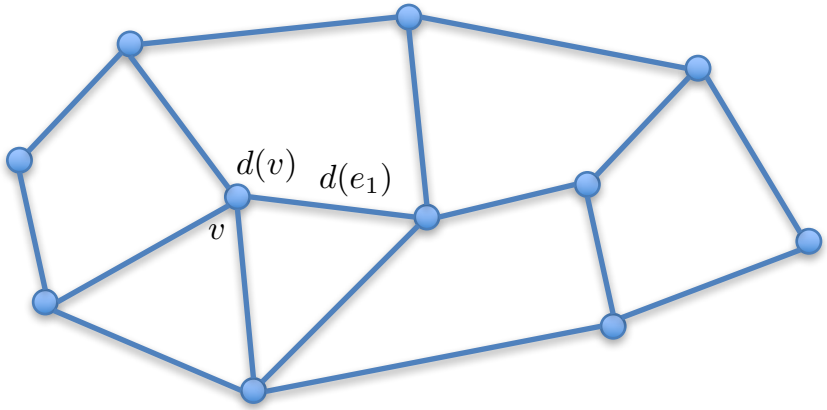
Total work+messages: $O(|V| + |E|) \rightarrow O(|V|^2)$

Our Solution

Two key ingredients:

- ❑ Sorting Networks
- ❑ Parallel Prefix Sum

Main Data Structure

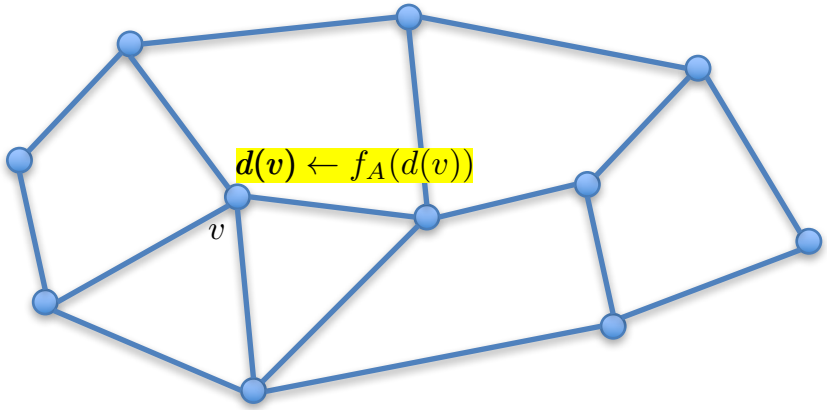


$G(V, E)$

$d(v_1)$	$d(v_2)$		$d(v_n)$	$d((v_1.v_2))$	$d((v_1, v_3))$		$d((v_n, v_{23}))$
v_1	v_2	$\cdots$	v_n	v_2	v_3	$\cdots$	v_{23}
v_1	v_2		v_n	v_1	v_1		v_n

$|V|$ $|E|$

Apply



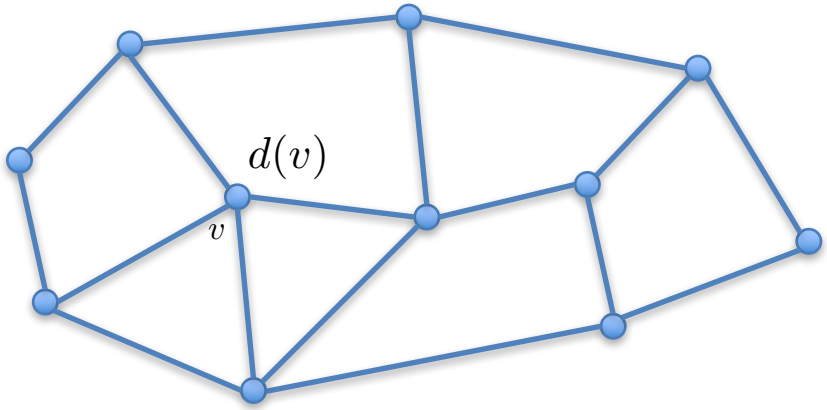
$G(V, E)$

$d(v_1)$	$d(v_2)$		$d(v_n)$	$d((v_1.v_2))$	$d((v_1, v_3))$		$d((v_n, v_{23}))$
v_1	v_2	$\cdots$	v_n	v_2	v_3	$\cdots$	v_{23}
v_1	v_2		v_n	v_1	v_1		v_n

For each column in array:

- If it is blue/node apply f_A on top row

Scatter

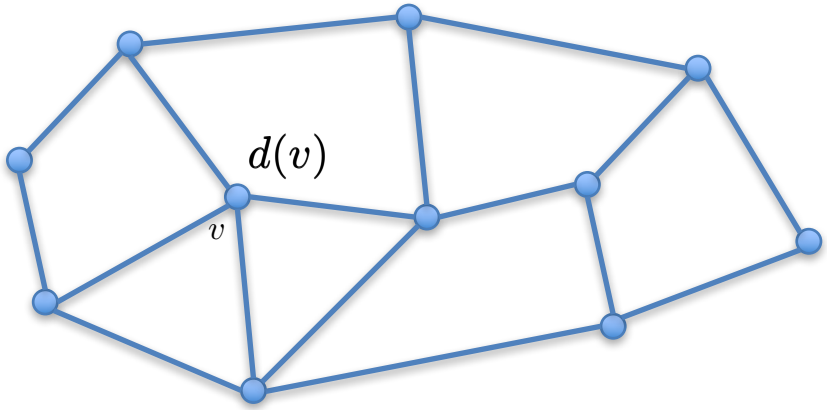


$G(V, E)$

$d(v_1)$	$d(v_2)$		$d(v_n)$	$d((v_1.v_2))$	$d((v_1, v_3))$		$d((v_n, v_{23}))$
v_1	v_2	$\cdots$	v_n	v_2	v_3	$\cdots$	v_{23}
v_1	v_2		v_n	v_1	v_1		v_n

Sort with respect to bottom row, prioritizing blue columns

Scatter

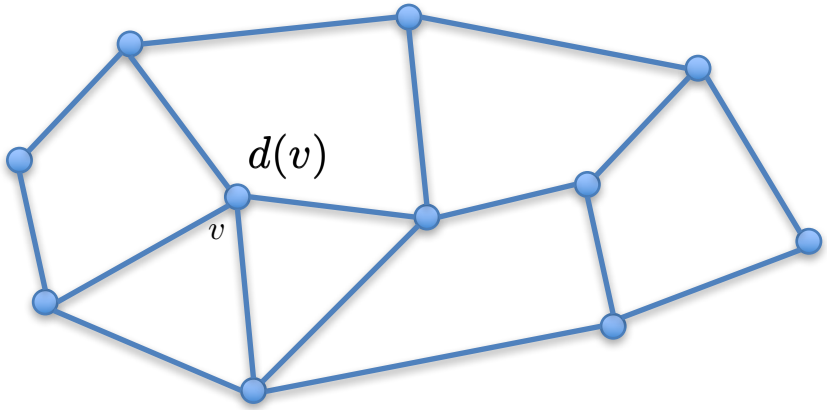


$G(V, E)$

$d(v_1)$	$d((v_1.v_2))$		$d((v_1, v_{16}))$	$d(v_2)$	$d((v_2, v_1))$			$d(v_n)$		$d((v_n, v_{23}))$
v_1	v_2	$\cdots$	v_{16}	v_2	v_1	$\cdots$		v_n	$\cdots$	v_{23}
v_1	v_1		v_1	v_2	v_2			v_n		v_n

Sort with respect to bottom row, prioritizing blue columns

Scatter

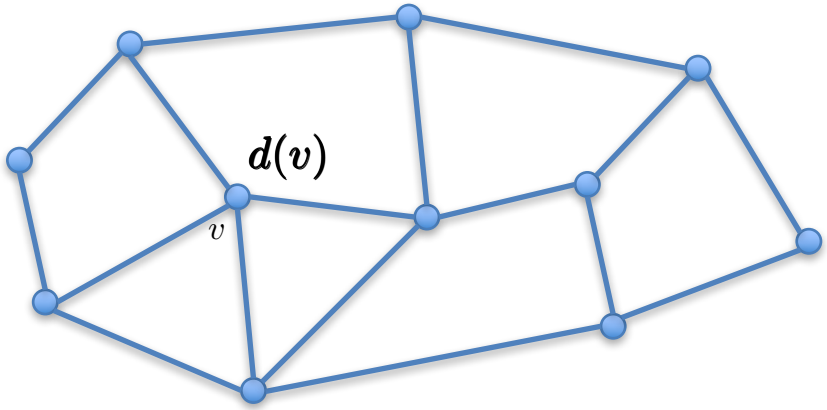


$G(V, E)$

$d(v_1)$	$d((v_1.v_2))$		$d((v_1, v_{16}))$	$d(v_2)$	$d((v_2, v_1))$			$d(v_n)$		$d((v_n, v_{23}))$
v_1	v_2	$\cdots$	v_{16}	v_2	v_1	$\cdots$		v_n	$\cdots$	v_{23}
v_1	v_1		v_1	v_2	v_2			v_n		v_n

Moving from **left** to **right**, copy item in third row until blue column is encountered

Scatter

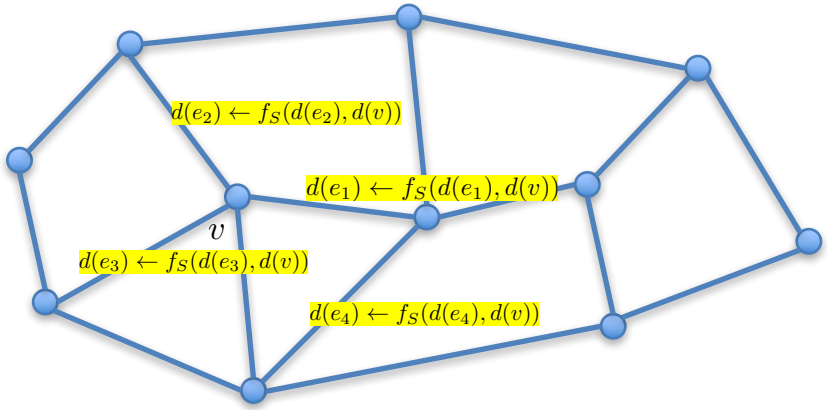


$G(V, E)$

$d(v_1)$	$d(v_1)$ $d((v_1.v_2))$	$d(v_1)$	$d(v_1)$ $d((v_1, v_{16}))$	$d(v_2)$	$d(v_2)$ $d((v_2, v_1))$			$d(v_n)$	$d(v_n)$	$d(v_n)$ $d((v_n, v_{23}))$
v_1	v_2	$\cdots$	v_{16}	v_2	v_1	$\cdots$		v_n	$\cdots$	v_{23}
v_1	v_1		v_1	v_2	v_2			v_n		v_n

Moving from **left** to **right**, copy item in third row until blue column is encountered

Scatter

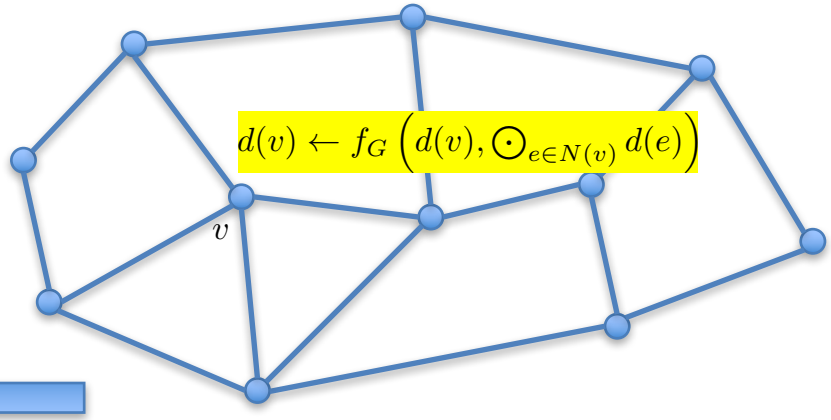


$G(V, E)$

$d(v_1)$	$d(v_1)$ $d((v_1.v_2))$	$d(v_1)$	$d(v_1)$ $d((v_1, v_{16}))$	$d(v_2)$	$d(v_2)$ $d((v_2, v_1))$			$d(v_n)$	$d(v_n)$	$d(v_n)$ $d((v_n, v_{23}))$
v_1	v_2	$\cdots$	v_{16}	v_2	v_1	$\cdots$		v_n	$\cdots$	v_{23}
v_1	v_1		v_1	v_2	v_2			v_n		v_n

For each column, apply f_S if it is red

Gather

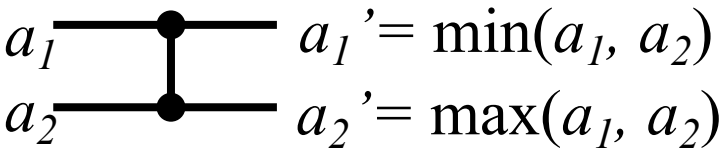


$d(v_1)$	$d((v_2, v_1))$		$d((v_1, v_{16}))$	$d(v_2)$	$d((v_1.v_2))$			$d(v_n)$		$d((v_n, v_{23}))$
v_1	v_1	$\cdots$	v_1	v_2	v_2	$\cdots$		v_n	$\cdots$	v_n
v_1	v_2		v_{16}	v_2	v_1			v_n		v_{23}

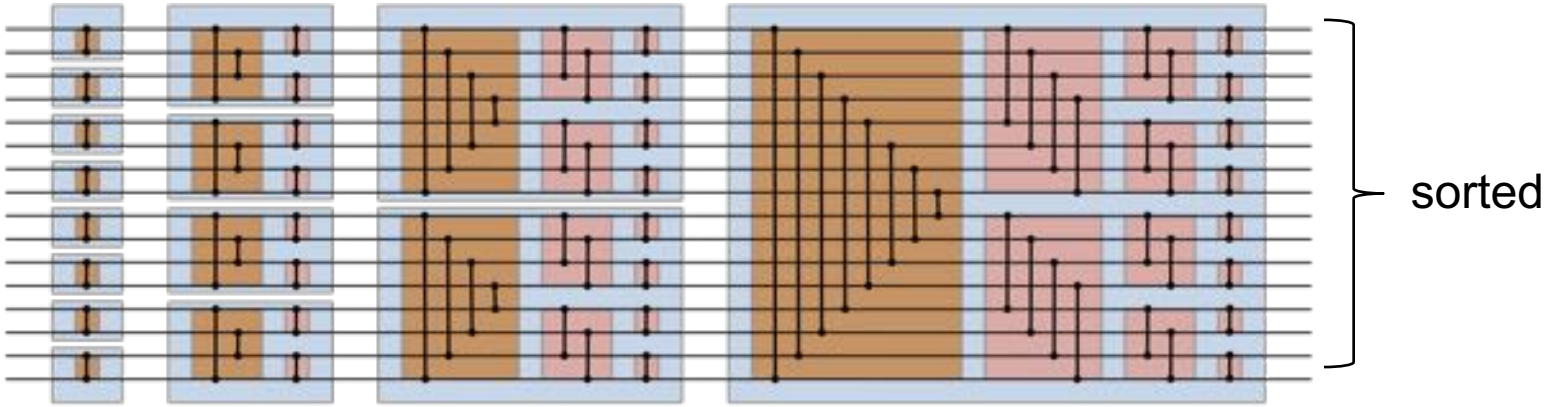
- **Sort** w.r.t. middle row, giving blue/node columns priority
- Aggregate data from **right** to **left**, using binary operator $\odot$, until Encountering a blue column
- For every column, apply f_G if it is blue

Parallel Sort

- Bitonic Sort



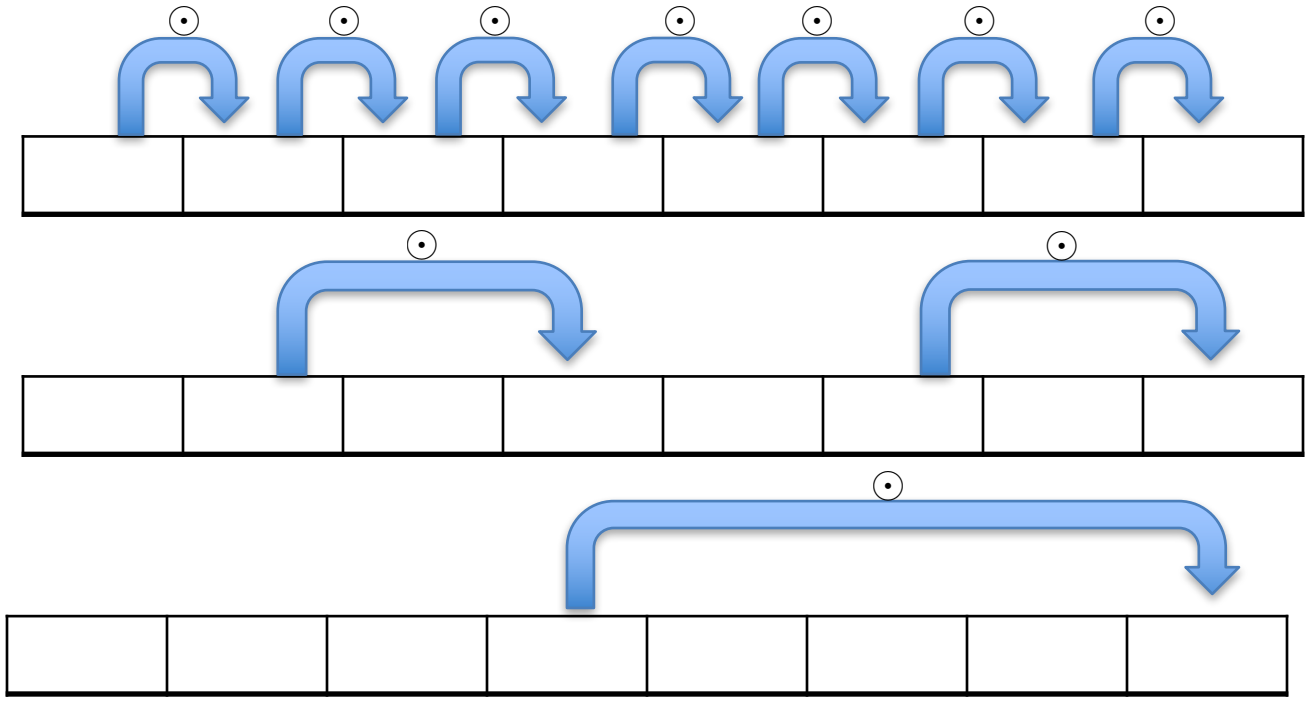
Compare and swap



- Total Work: $O(M \log^2 M)$ Depth: $O(\log^2 M)$

Parallel Left/Right Passes

- Modification on Parallel Prefix Sum



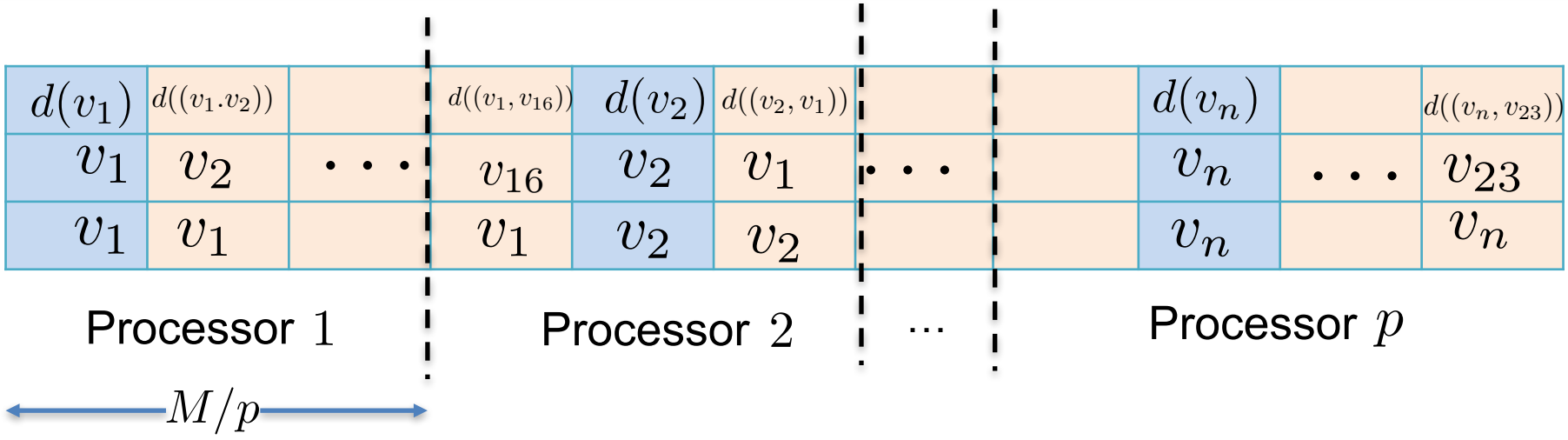
- Total Work: $O(M \log M)$ Rounds: $O(\log M)$

Parallel Properties

$d(v_1)$	$d((v_1.v_2))$		$d((v_1, v_{16}))$	$d(v_2)$	$d((v_2, v_1))$			$d(v_n)$		$d((v_n, v_{23}))$
v_1	v_2	$\cdots$	v_{16}	v_2	v_1	$\cdots$		v_n	$\cdots$	v_{23}
v_1	v_1		v_1	v_2	v_2			v_n		v_n

$$M = |V| + |E|$$

Parallel Properties



If processors connect in a **hypercube** network then:

- **Scatter**, **Gather**: Parallel Time: $O(\frac{M}{p}\log^2 M)$ Total Work: $O(M\log^2 M)$
- **Apply**: Parallel Time: $O(M/p)$ Total Work: $O(M)$

Serial execution in the clear: $O(M)$

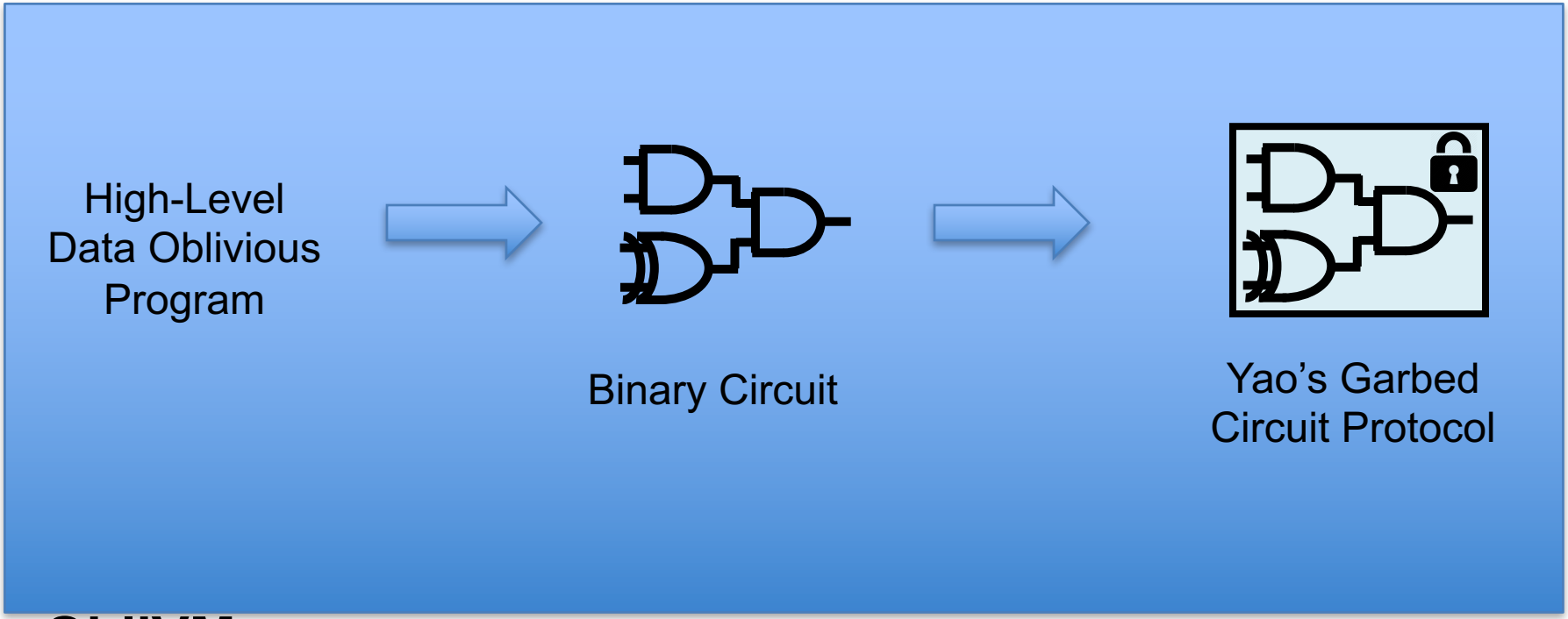
CoO: $\sim \log^2 M$

Speedup: $\sim p/\log^2 M$

Overview

- Graph-Parallel Algorithms
- Graph-Parallel Secure Evaluation
- **Implementation**

From Data Oblivious Algorithm to SFE

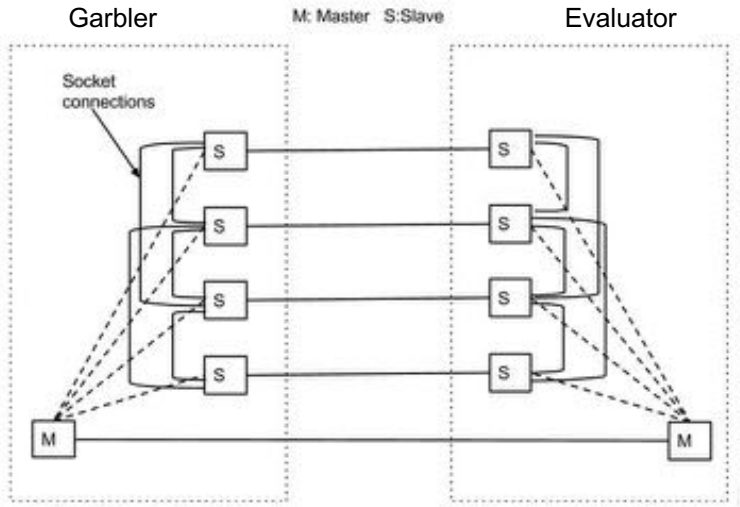


ObliVM

[Liu, Wang, Nayak, Huang, Shi; IEEE S&P, 2015]

Our Implementation: GraphSC

- Extension to ObliVM
- Supports **thread** + **cluster** parallelism
- Programmer defines f_S ,f_G ,f_A ,$\odot$ and sequence of Scatter/Gather/Apply calls, GraphSC takes care of:
  - Implementing **scatter**, **gather**, and **apply**, through **sorting** and **left/right passes**
  - Translation to **binary circuit** through **ObliVM**
  - Garbling and Evaluation over multiple nodes forming **hypercube** network.



https://github.com/kartik1507/GraphSC

Example: PageRank over GraphSC

def $\textsc{PageRank}(G)$:

$\texttt{Apply}\,(d(v) \mapsto 1/|V|)$

repeat:

$\texttt{Apply}(d(v) \mapsto d(v)/N(v))$

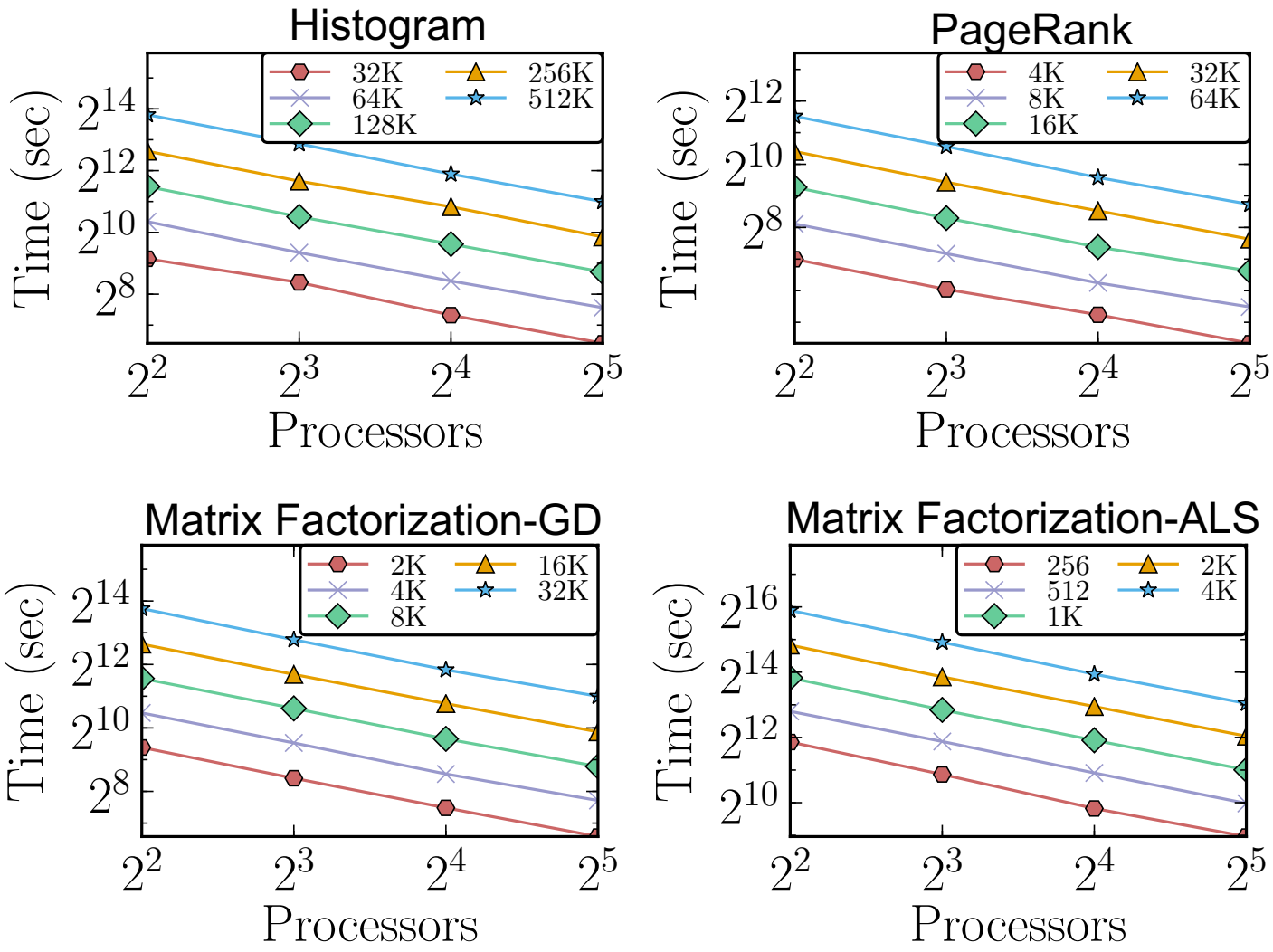
$\texttt{Scatter}((x,y) \mapsto x)$

$\texttt{Gather}\left((x,y) \mapsto \gamma \frac{1}{|V|} x + \gamma y,\ +\right)$

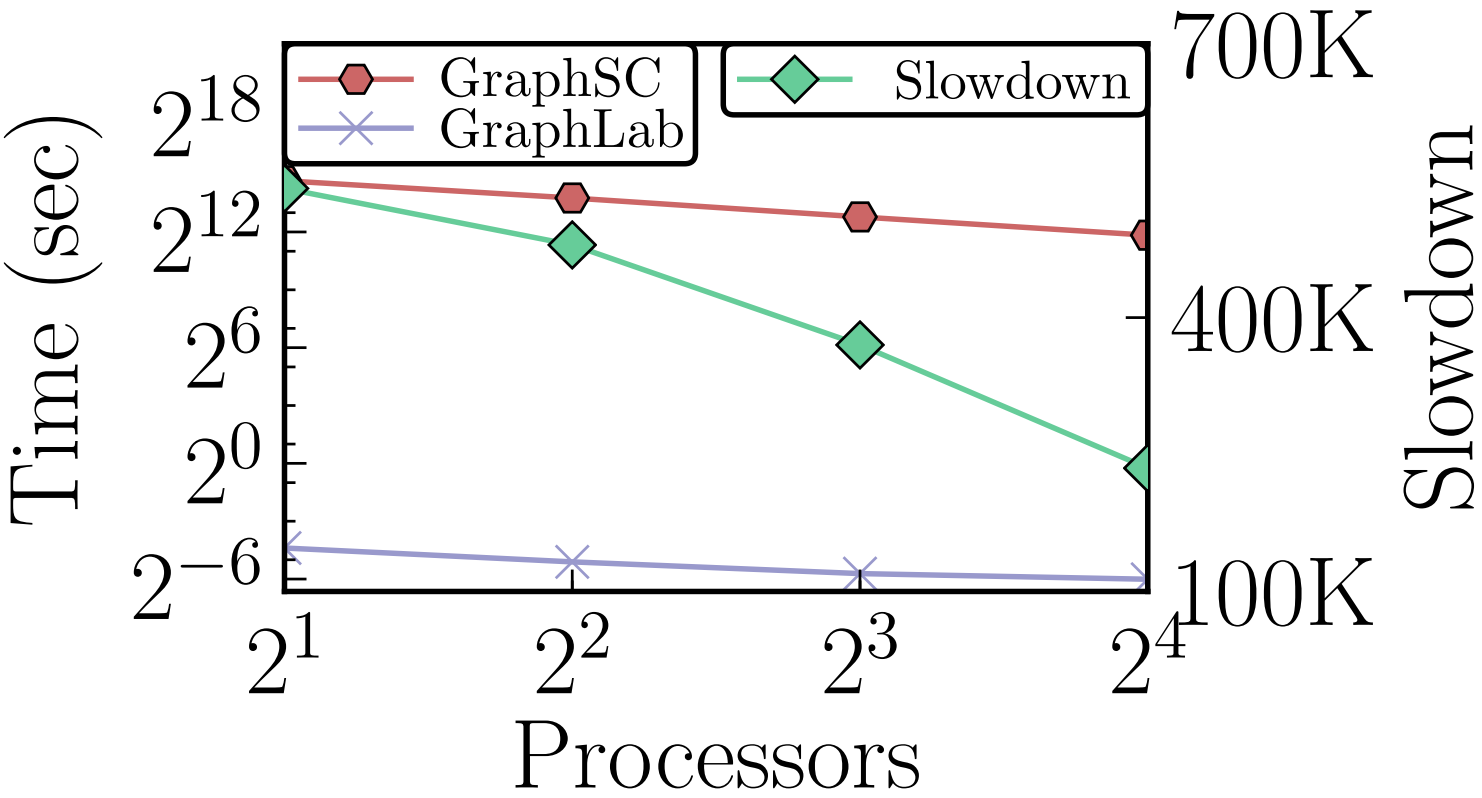
until convergence

```
public class PageRankNode<T> extends GraphNode<T> {
       T[] pr;
       T[] l;
}
public class PageRank<T> implements ParallelGadget<T> {
  @Override
  public <T> void compute(PageRankNode<T>[] graph,
       final CompEnv<T> env, T[] N) {

    final IntegerLib<T> lib = new IntegerLib<>(env);
    final ArithmeticLib<T> flib = new FixedPointLib<T>(env,
             40/*Width*/, 20/*Offset*/);
    // Set initial PageRank
    new SetInitialPageRankGadget<T>(env).setInputs(graph).compute();

    for (int i = 0; i < ITERATIONS; i++) {
      // 1. Write weighted PR to edges
       new Scatter<T>(env, false /* isEdgeIncoming */) {
         @Override
         public void writeToEdge(PageRankNode<T> vertex,
                PageRankNode<T> edge, T cond) {
             T[] div = flib.div(vertex.pr, vertex.l);
             edge.pr = lib.mux(div, edge.pr, cond);
         }
       }.setInputs(graph).compute();

       // 2. Compute PR based on edges
       new Gather<T>(env, true /* isEdgeIncoming */,
              new PageRankNode<T>(env)) {
         @Override
         public PageRankNode<T> aggregate(PageRankNode<T> agg,
                PageRankNode<T> edge) {
             PageRankNode<T> ret = new PageRankNode<T>(env);
             ret.pr = flib.add(agg.pr, edge.pr);
             return ret;
          }

          @Override
          public void writeToVertex(PageRankNode<T> agg,
                 PageRankNode<T> node) {
            T[] d = flib.publicValue(0.15);
            T[] dp = flib.publicValue(0.85);
            T[] newPageRank = flib.add(flib.div(d, N),
                       flib.multiply(dp, agg));
            node.pr = lib.mux(node.pr, newPageRank, node.isVertex);
          }
       }.setInputs(graph).compute();
    }
    new SortGadget<T>(env)
      .setInputs(graph, PageRankNode.vertexFirstComparator(env))
      .compute();
       }
}
```


Speedup

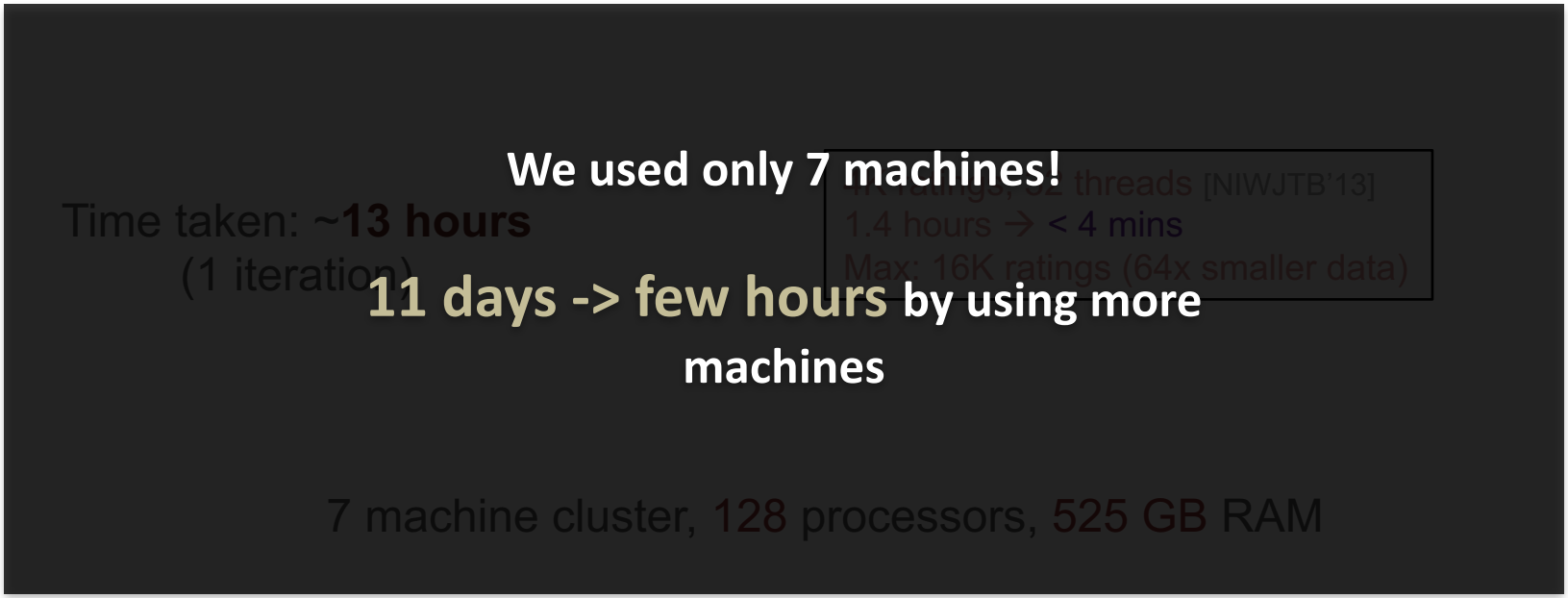


GraphSC vs. GraphLab



Matrix Factorization, 32K ratings

Matrix Factorization using gradient descent: **1M** ratings, 6K users, 4K movies



One run: 20 iterations - **11** days

Towards a Platform for Massive SFE

- Parallel SFE at scale is possible for a broad array of algorithms
- Implementations of additional algorithms
  - DNN, ERM, graph coloring, ADMM, etc.
- Further accelaration over FPGAs in the datacenter

[Fang, Ioannidis, Leeser; FPGA 2017]

Northeastern

Thank you!

Kartik Nayak, Xiao Shaun Wang, Stratis Ioannidis, Udi Weinsberg, Nina Taft, and Elaine Shi. "GraphSC: Parallel secure computation made easy." In *IEEE Symposium on Security and Privacy* (S&P/OAKLAND), 2015.

Nikolaenko, Valeria, Stratis Ioannidis, Udi Weinsberg, Marc Joye, Nina Taft, and Dan Boneh. "Privacy-preserving matrix factorization." In C*onference on Computer & Communications Security* (CCS), 2013.

Xin Fang, Stratis Ioannidis, Miriam Leeser. “Secure Function Evaluation using an FPGA Overlay Architecture”. In *International Symposium on Field-Programmable Gate Arrays* (FPGA), 2017