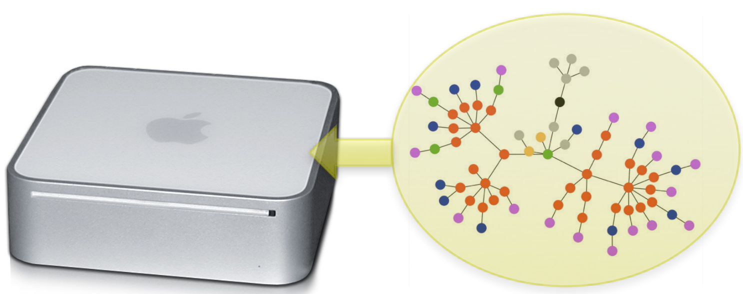
Big Data Summer School

Graph Systems

Research Goal

Compute on graphs with billions of edges, in *a reasonable time,* on a single PC.

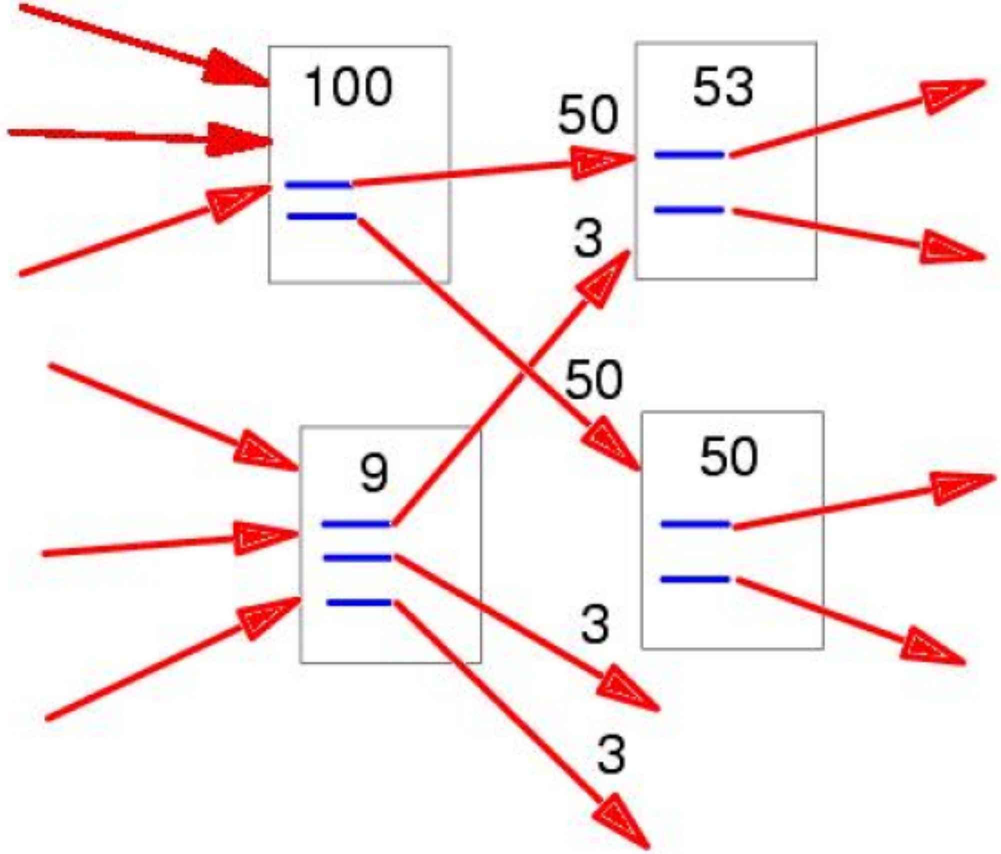
- *Reasonable* = close to numbers previously reported for distributed systems in the literature.



Experiment PC: Mac Mini (2012)

Pagerank

- Link-based analysis



Pagerank

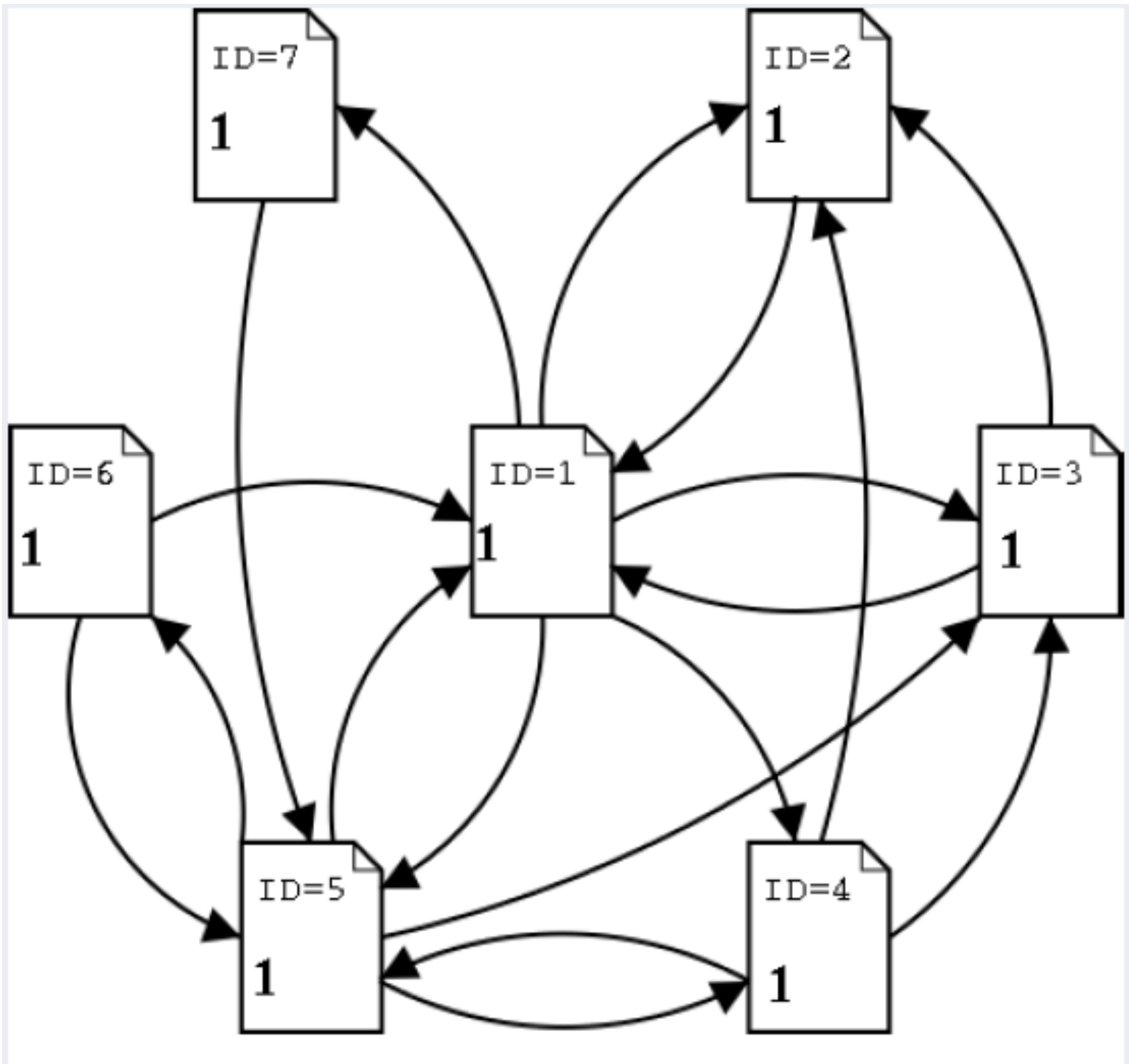
- Link-based analysis

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

- $\mathbf{B_u}$: the set containing all pages linking to page **u**
- $L(v)$: the number of links from page **v**.

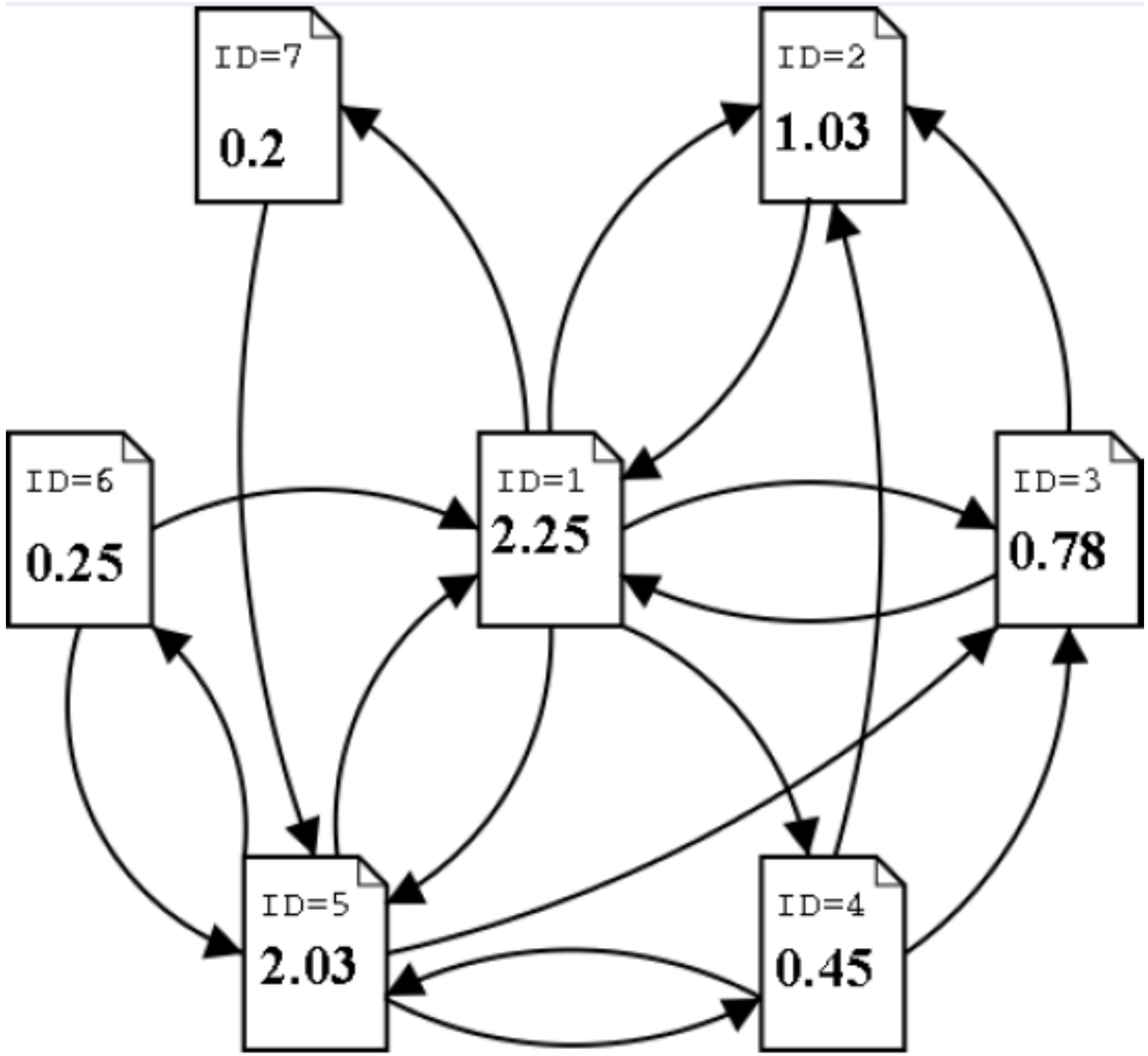
$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

Pagerank



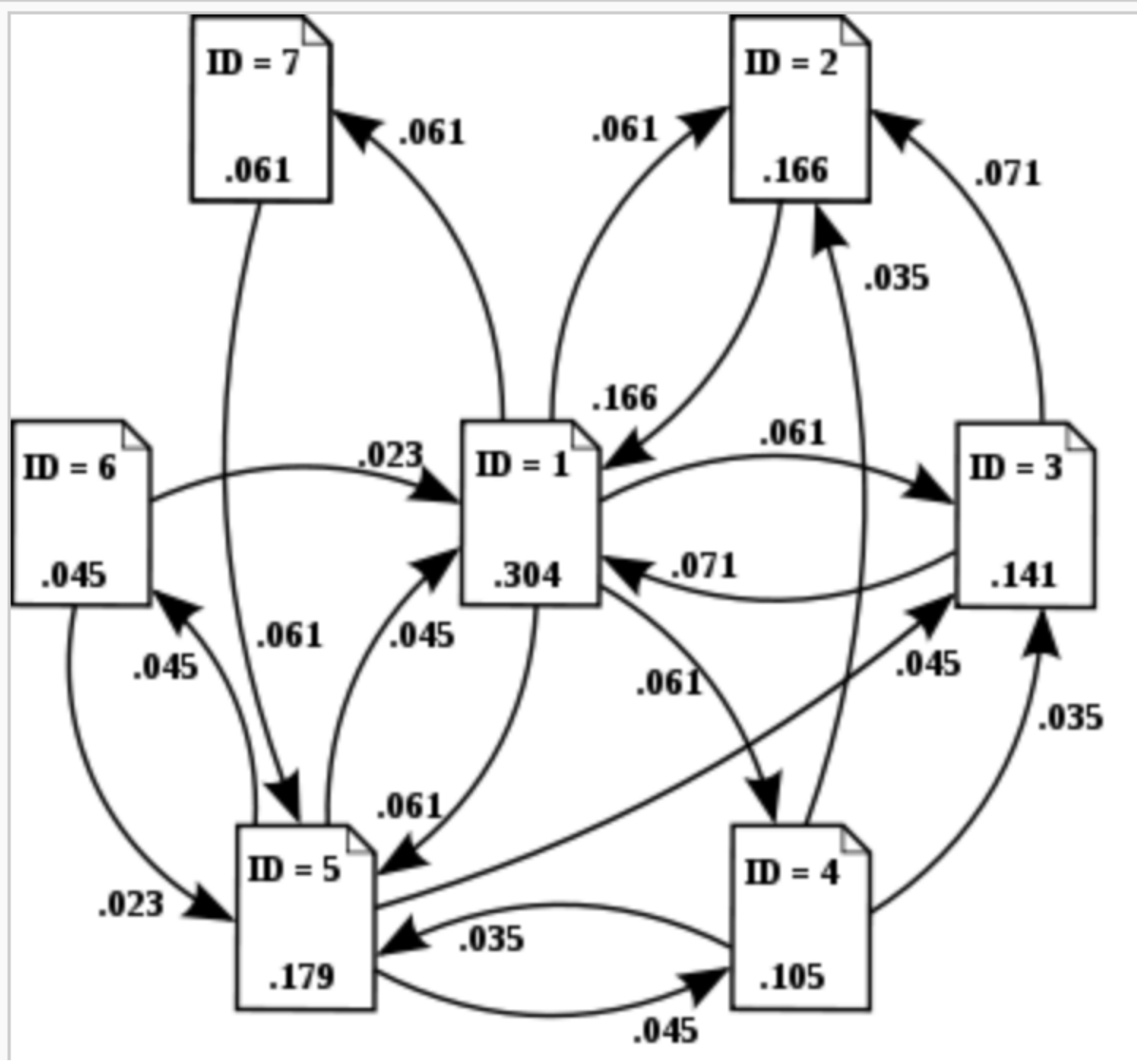
Pagerank

$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$



$$PR(u) = \sum_{v \in B_u} \frac{PR(v)}{L(v)}$$

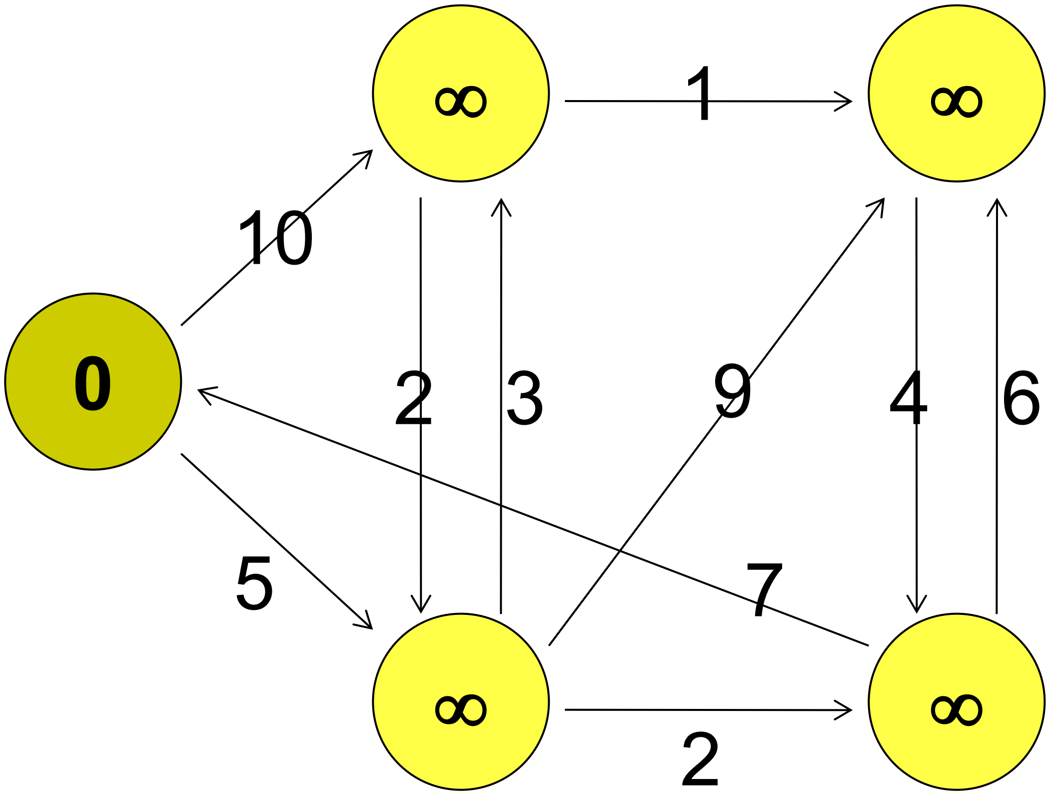
Pagerank



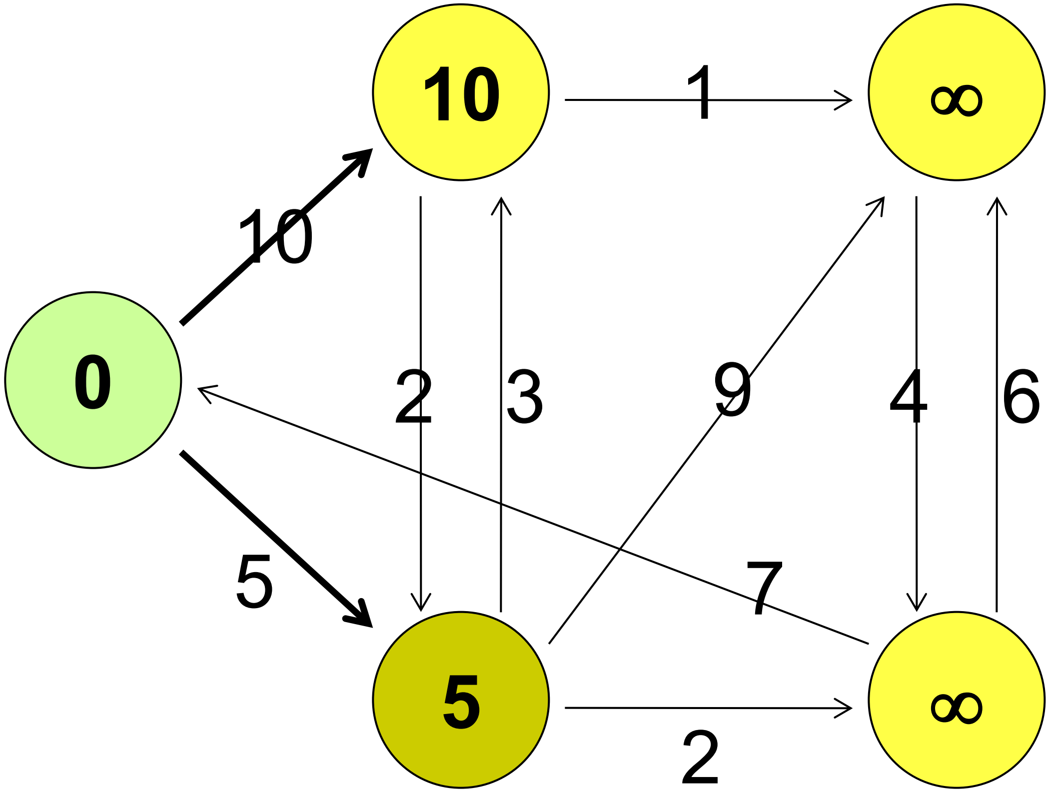
PageRank

- Iterations
  - Gather
  - Apply
  - Scatter

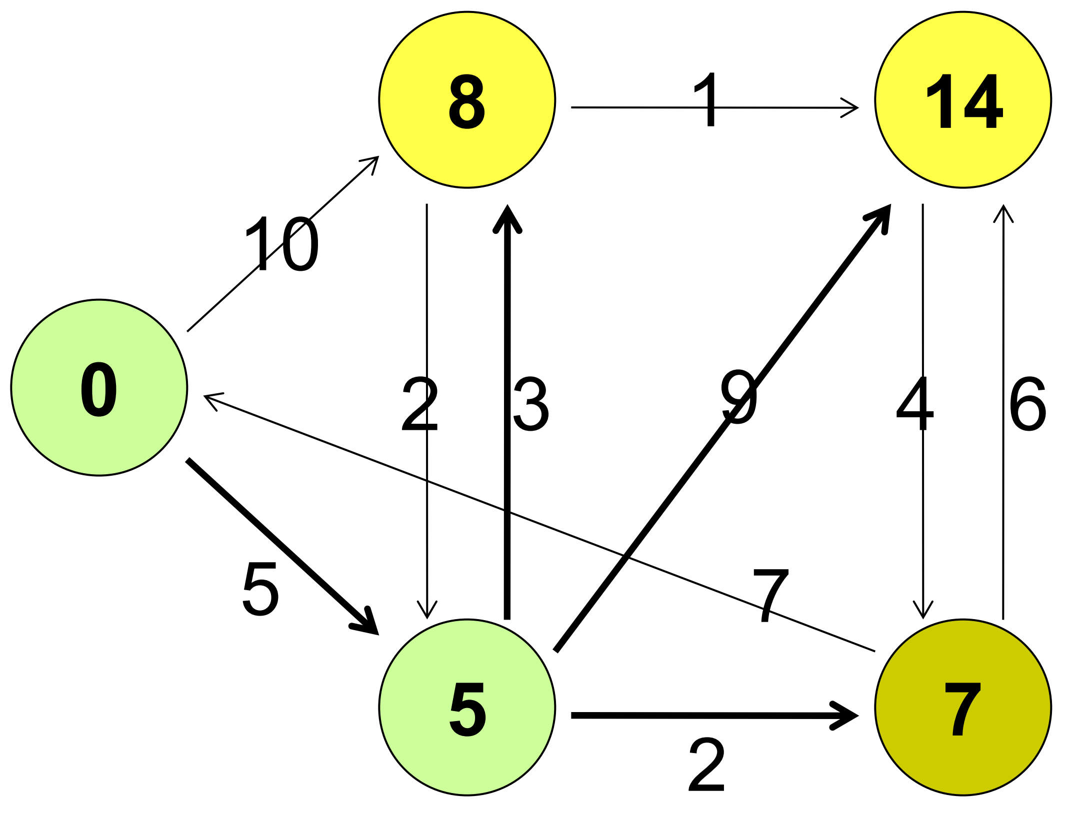
Example: SSSP – Dijkstra's Algorithm



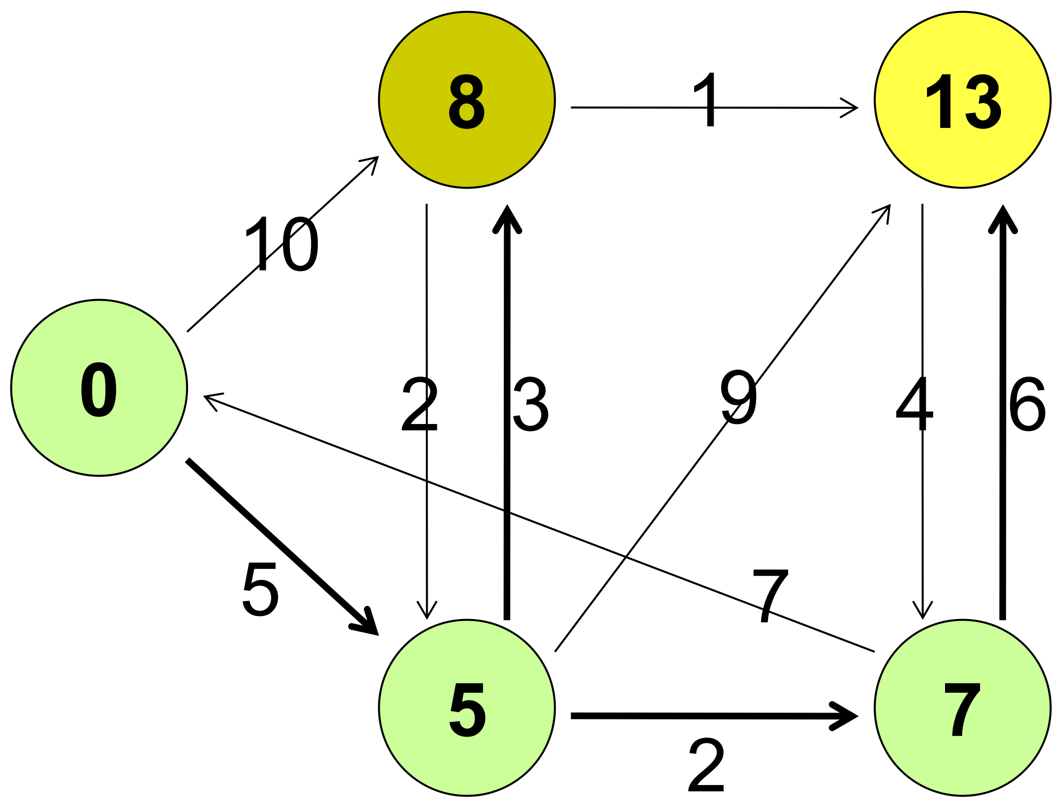
Example: SSSP – Dijkstra's Algorithm



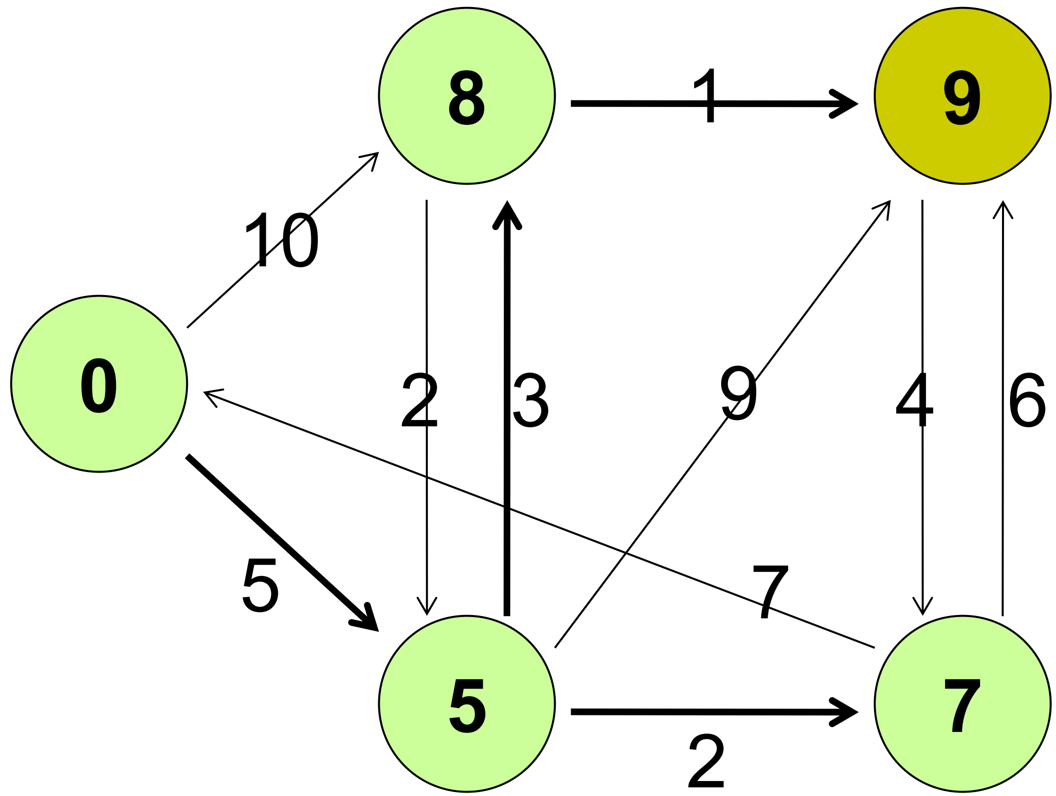
Example: SSSP – Dijkstra's Algorithm



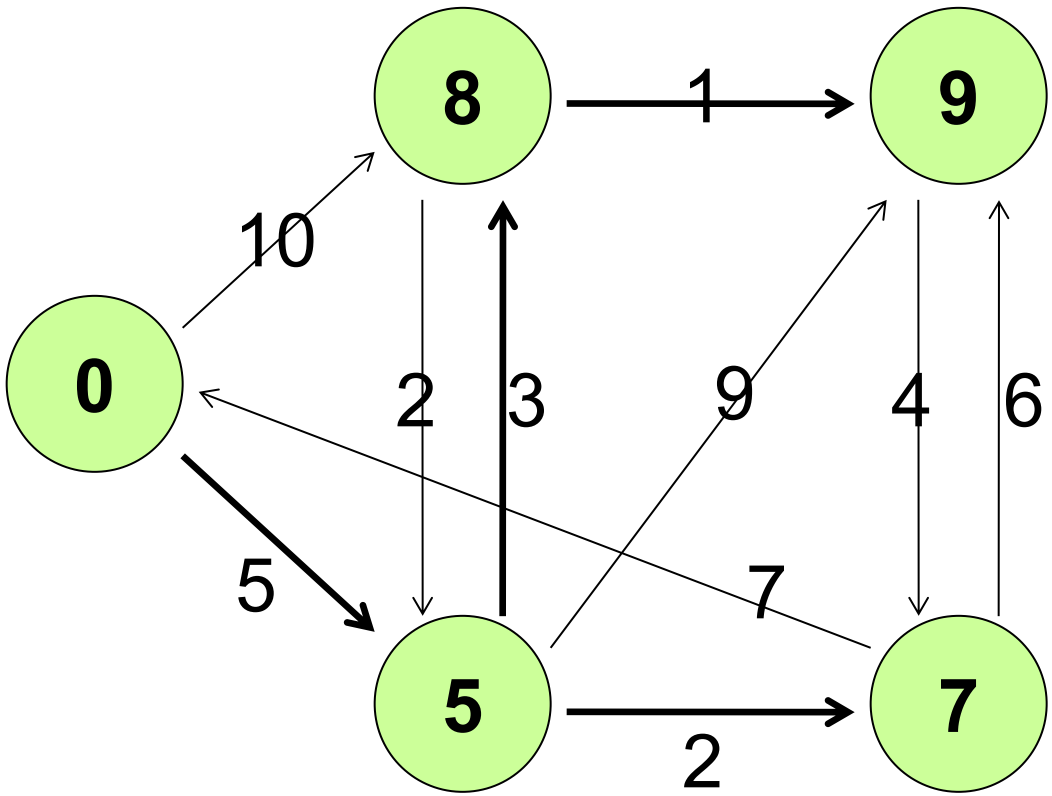
Example: SSSP – Dijkstra's Algorithm



Example: SSSP – Dijkstra's Algorithm

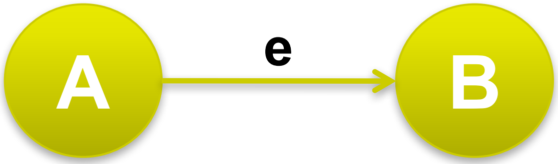


Example: SSSP – Dijkstra's Algorithm

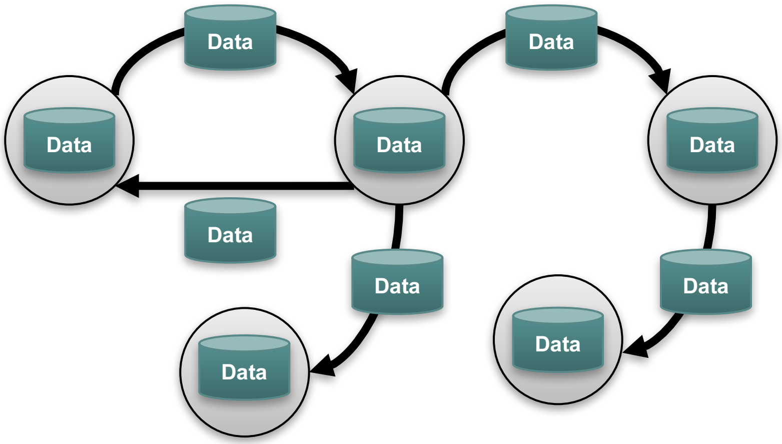


Computational Model

- Graph G = (V, E)
  - **directed edges**: e = (source, destination)
  - each edge and vertex **associated with a value** (user-defined type)
  - vertex and edge **values can be modified**
    - (structure modification also supported)

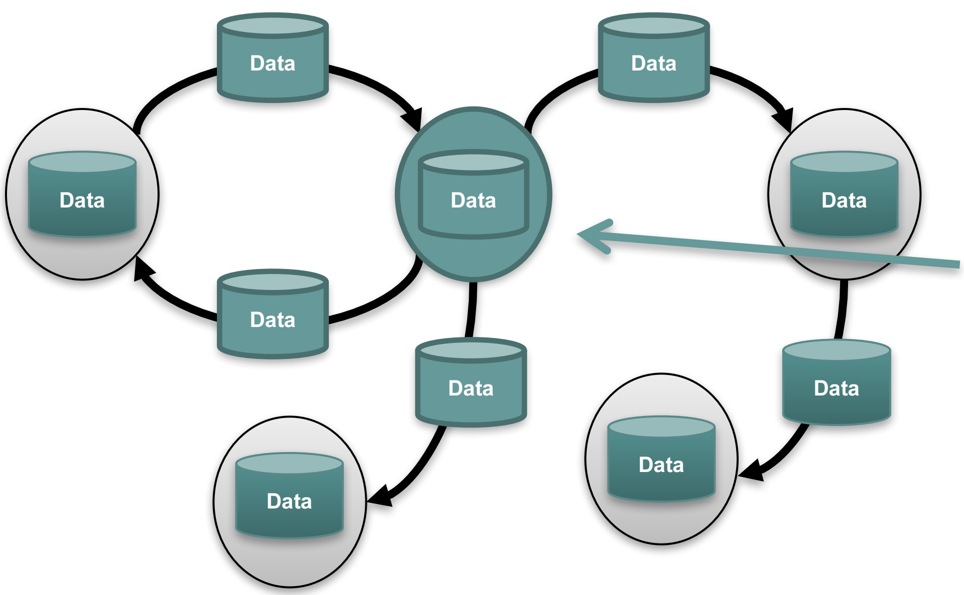


Terms: e is an out-edge of A, and in-edge of B.



Vertex-centric Programming

- "Think like a vertex"
- Popularized by the Pregel and GraphLab projects
  - Historically, systolic computation and the Connection Machine



MyFunc(vertex)
{ // modify neighborhood
}

The Main Challenge of Disk-based Graph Computation:

Random Access

Random Access Problem

- Symmetrized adjacency file with values,

vertex	in-neighbors	out-neighbors
5	**3**:2.3, **19**: 1.3, **49**: 0.65,...	**781**: 2.3, **881**: 4.2..
....		
19	**3**: 1.	

synchronize

Random write

For sufficient performance, millions of random accesses / second would be needed. Even for SSD, this is too much.

- ... or with

vertex	in-ne	bors
5	**3**: 881	81: 4.2..
....		
19	**3**: 882, **9**: 2872, ...	**5**: 1.3, 28: 2.2, ...

read

Random read

Parallel Sliding Windows: Phases

- PSW processes the graph one **sub-graph** a time:

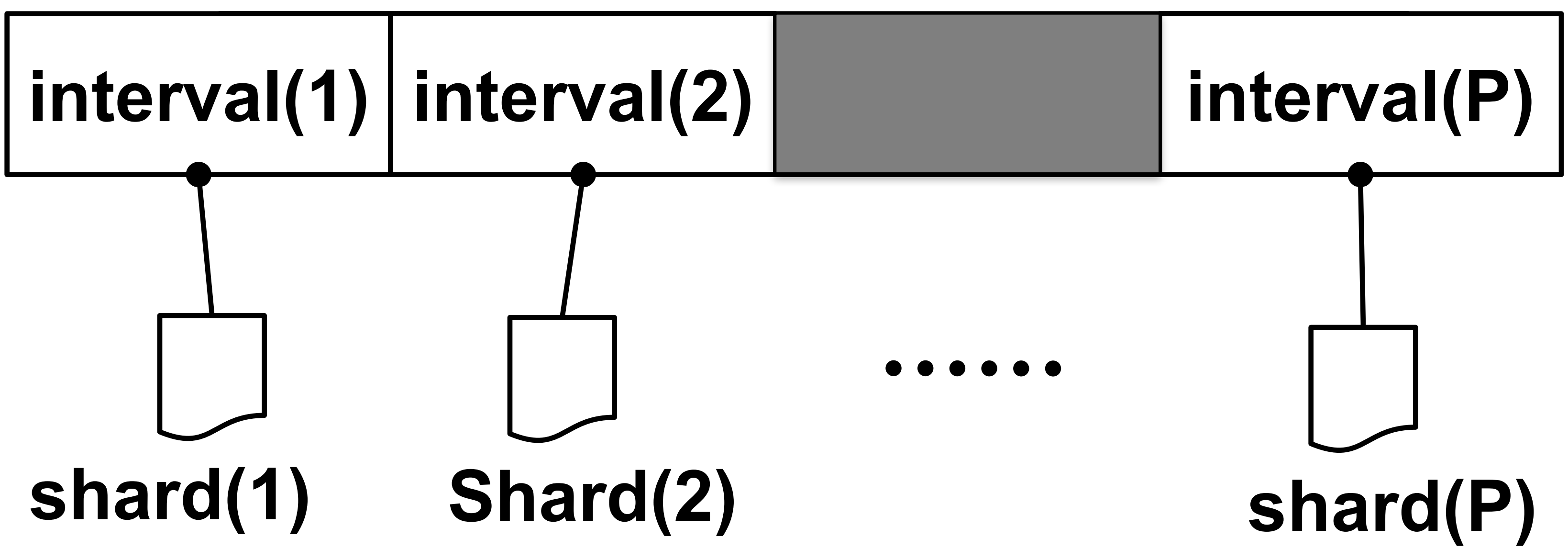
1. Load
2. Compute
3. Write

- In one **iteration**, the whole graph is processed.
  - And typically, next iteration is started.

PSW: Shards and Intervals

1. Load
2. Compute
3. Write

- Vertices are numbered from 1 to n
  - **P** intervals, each associated with a **shard** on disk.
  - **sub-graph =** interval of vertices



Example

Shard 1

src	dst	value
1		
	2	0.3
3		
	2	0.2
4		
	1	1.4
5		
	1	0.5
	2	0.6
6		
	2	0.8

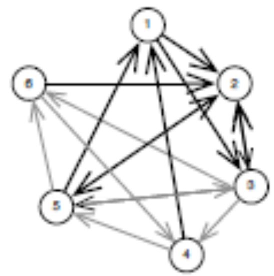
Shard 2

src	dst	value
1		
	3	0.4
2		
	3	0.3
3		
	4	0.8
5		
	3	0.2
6		
	4	1.9

Shard 3

src	dst	value
2		
	5	0.6
3		
	5	0.9
	6	1.2
4		
	5	0.3
5		
	6	1.1

(a) Execution interval (vertices 1-2)



(b) Execution interval (vertices 1-2)

Shard 1

src	dst	value
1		
	2	0.273
3		
	2	0.22
4		
	1	1.54
5		
	1	0.55
	2	0.66
6		
	2	0.88

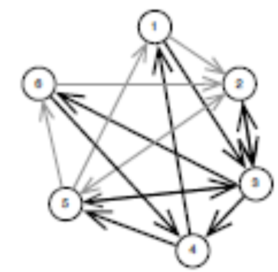
Shard 2

src	dst	value
1		
	3	0.364
2		
	3	0.273
3		
	4	0.8
5		
	3	0.2
6		
	4	1.9

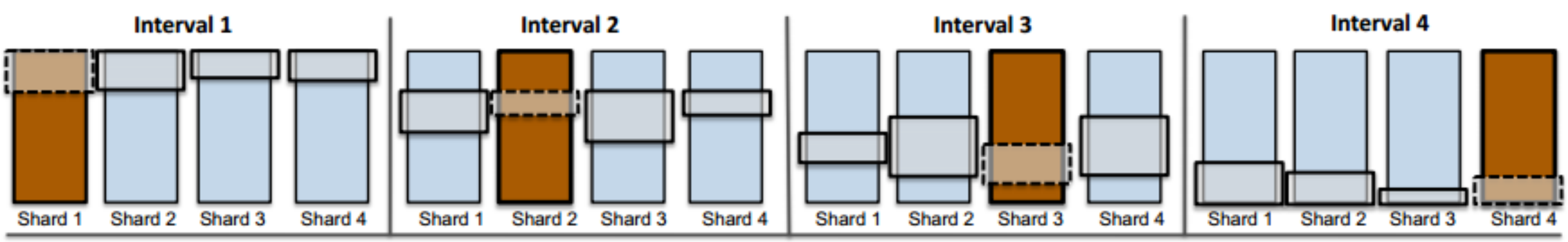
Shard 3

src	dst	value
2		
	5	0.545
3		
	5	0.9
	6	1.2
4		
	5	0.3
5		
	6	1.1

(c) Execution interval (vertices 3-4)



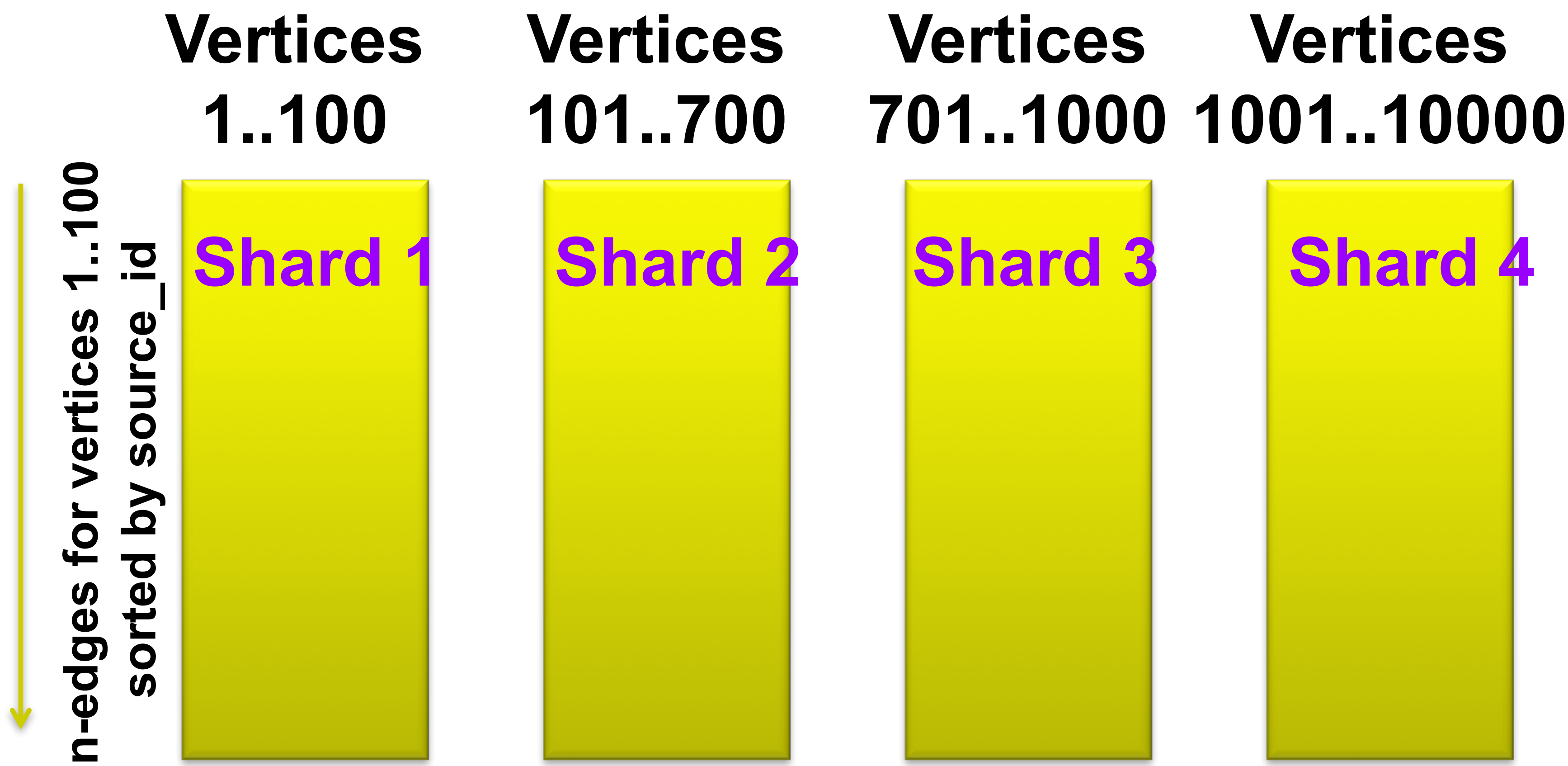
(d) Execution interval (vertices 3-4)



1. Load
2. Compute
3. Write

PSW: Layout

Shard: in-edges for interval of vertices; sorted by source-id

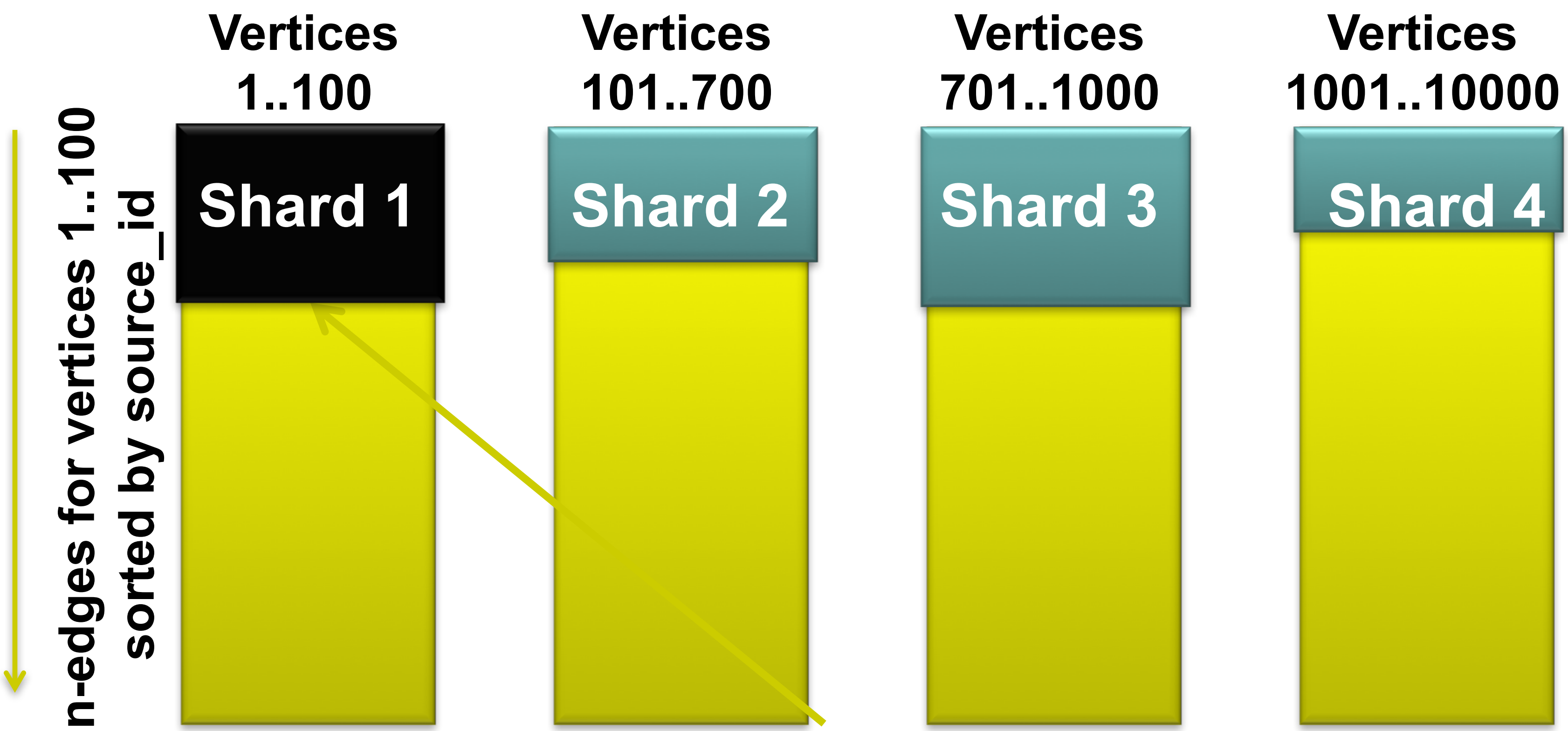


Shards small enough to fit in memory; balance size of shards

PSW: Loading Sub-graph

Load subgraph for vertices 1..100

1. Load
2. Compute
3. Write



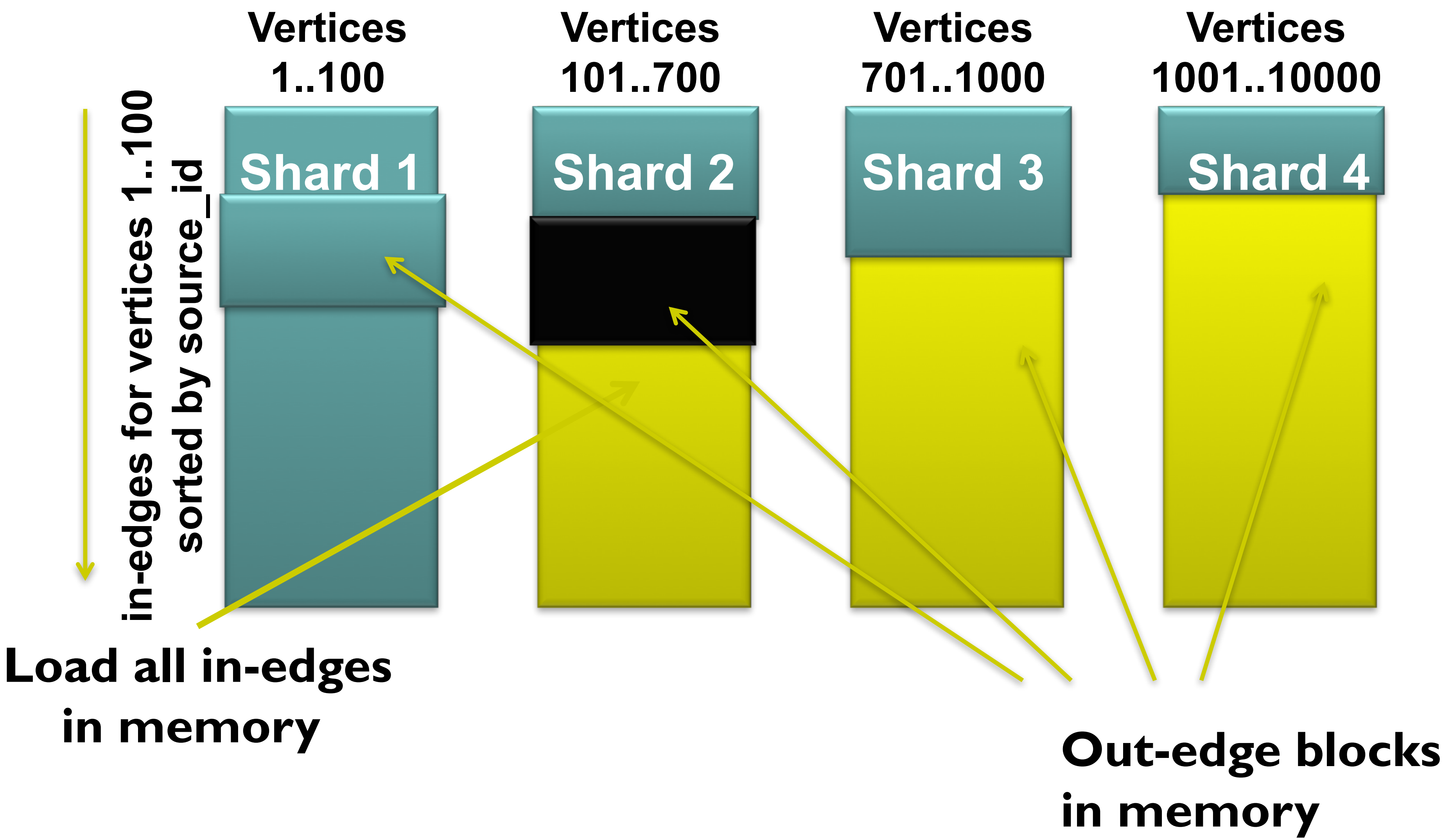
Load all in-edges in memory

What about out-edges?
Arranged in sequence in other shards

PSW: Loading Sub-graph

1. Load
2. Compute
3. Write

Load subgraph for vertices 101..700

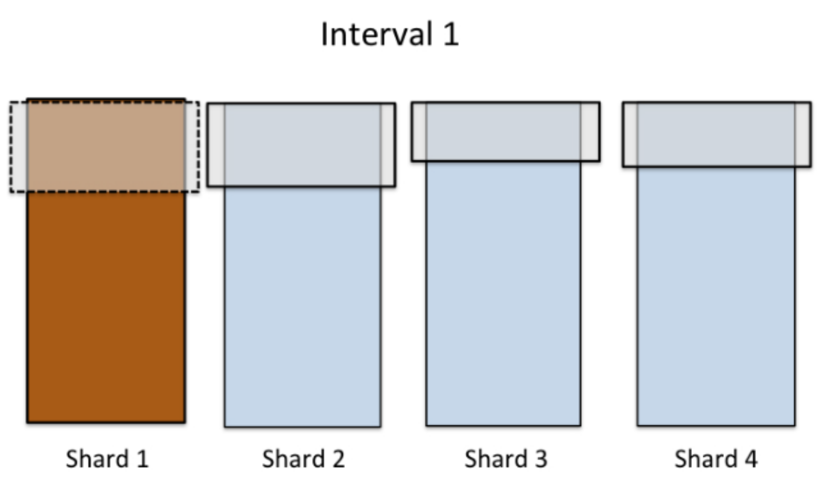


PSW Load-Phase

1. Load
2. Compute
3. Write

Only P large reads for each interval.

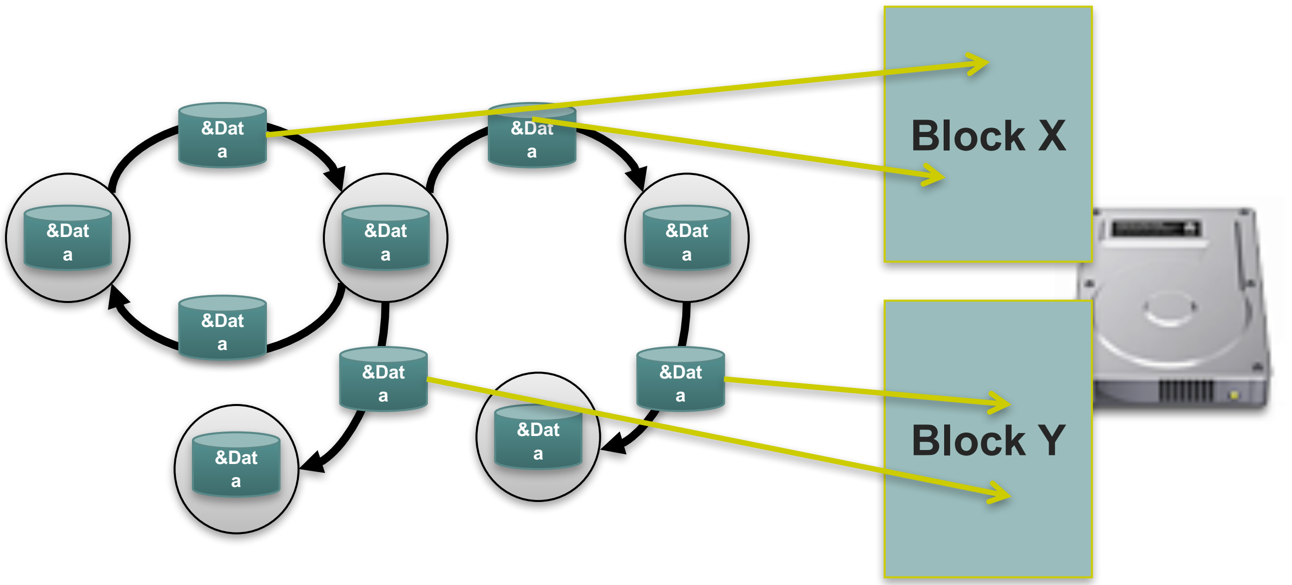
P^2 reads on one full pass.



1. Load
2. Compute
3. Write

PSW: Execute updates

- Update-function is executed on interval's vertices
- Edges have pointers to the loaded data blocks
  - Changes take effect immediately → asynchronous.



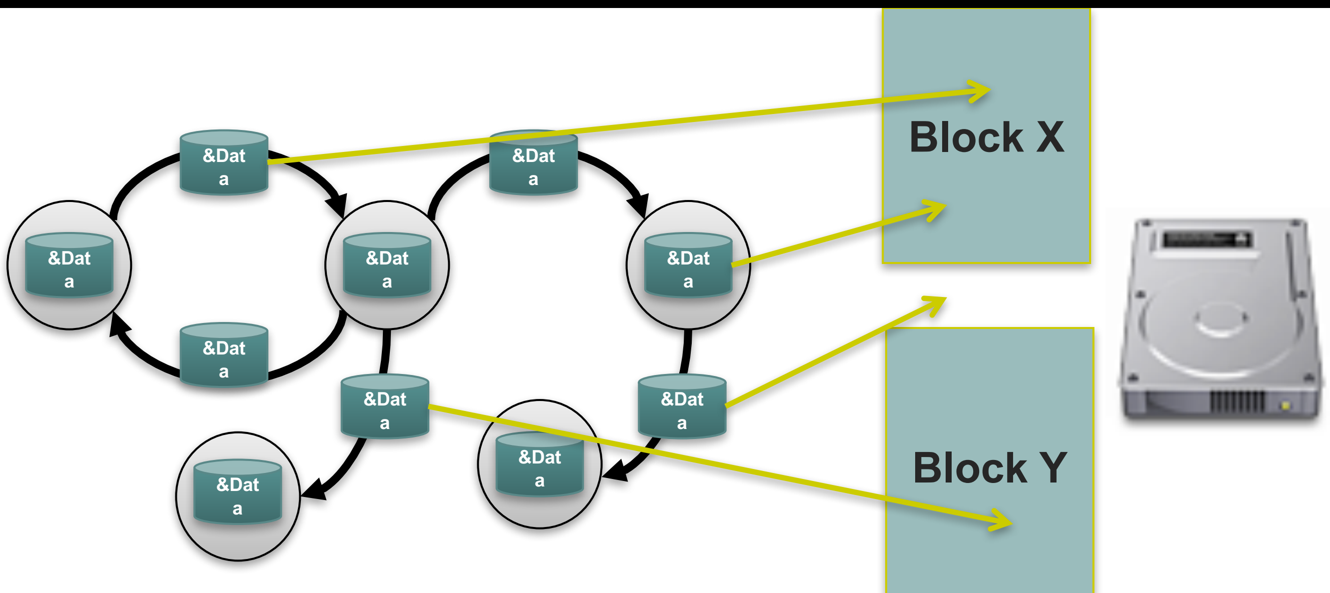
Deterministic scheduling prevents races between neighboring vertices.

1. Load
2. Compute
3. Write

PSW: Commit to Disk

- In write phase, the blocks are written *back* to disk
  - Next load-phase sees the preceding writes → asynchronous.

In total:
P^2 reads and writes / full pass on the graph.
→ Performs well on *both* SSD and hard drive.



Programming

virtual void before_iteration(int iteration, graphchi_context &gcontext)
virtual void after_iteration(int iteration, graphchi_context &gcontext)
virtual bool repeat_updates(graphchi_context &gcontext)
virtual void before_exec_interval(vid_t window_st, vid_t window_en, graphchi_context &gcontext)
virtual void after_exec_interval(vid_t window_st, vid_t window_en, graphchi_context &gcontext)
virtual void update(vertex_t &v, graphchi_context &gcontext)=0

Evolving Graphs

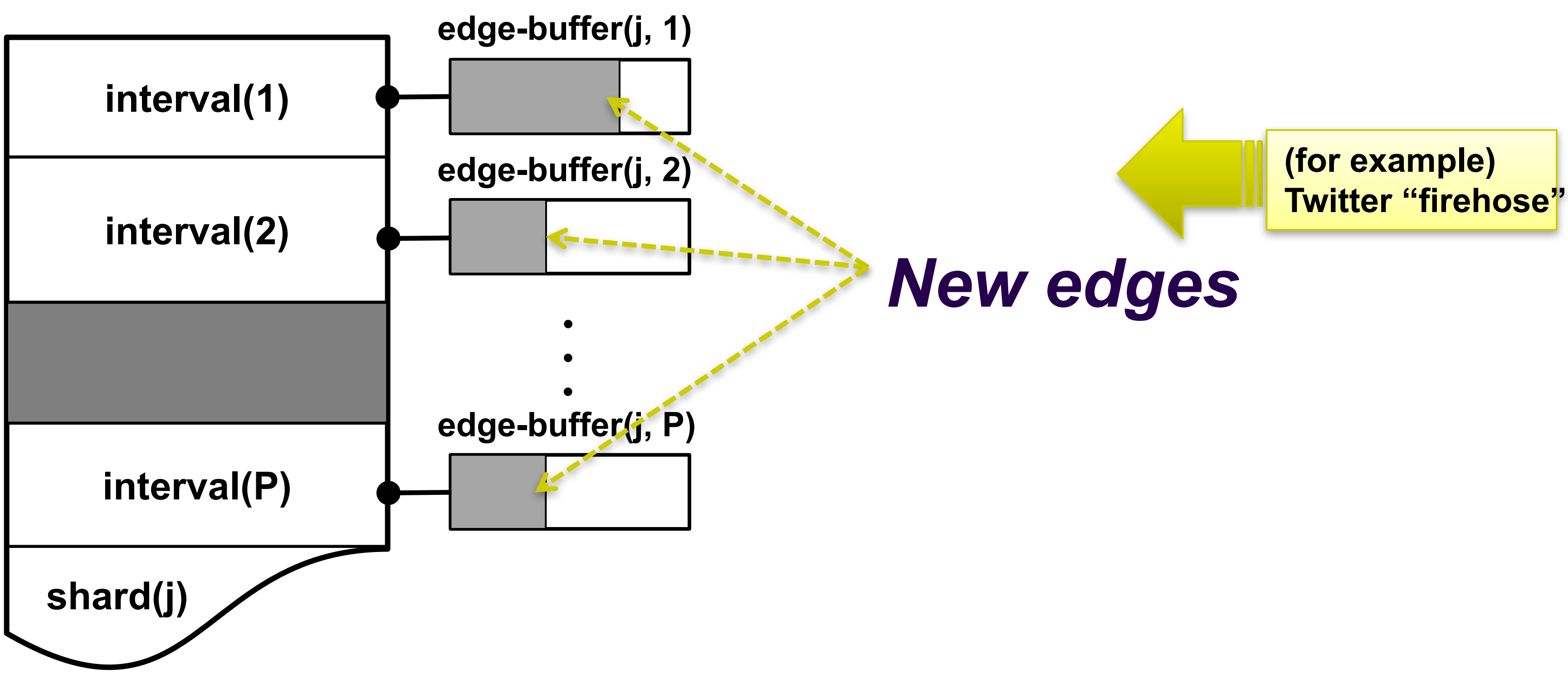
Graphs whose structure changes over time

Evolving Graphs: Introduction

- Most interesting networks grow continuously:
  - New connections made, some ‘unfriended’.
- Desired functionality:
  - Ability to add and remove edges in streaming fashion;
  - ... while continuing computation.
- Related work:
  - *Kineograph* (EuroSys ‘12), distributed system for computation on a changing graph.

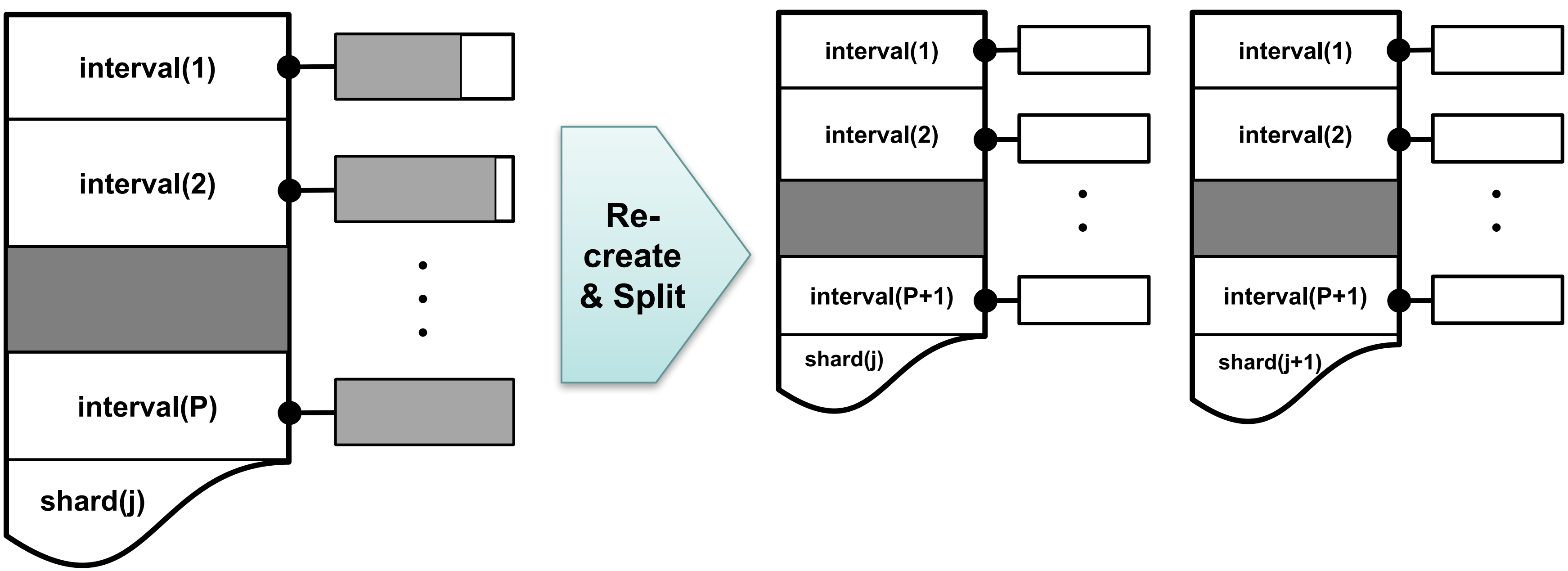
PSW and Evolving Graphs

- Adding edges
  - Each (shard, interval) has an associated edge-buffer.
- Removing edges: Edge flagged as “removed”.



Recreating Shards on Disk

- When buffers fill up, shards a **recreated** on disk
  - Too big shards are **split**.
- During recreation, deleted edges are permanently removed.



Distributed Graph Systems

- Pregel
- GraphLab
- GraphX