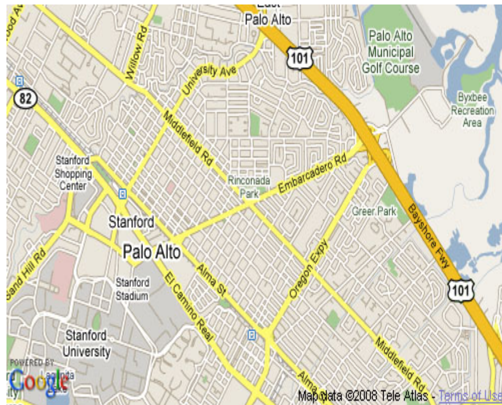
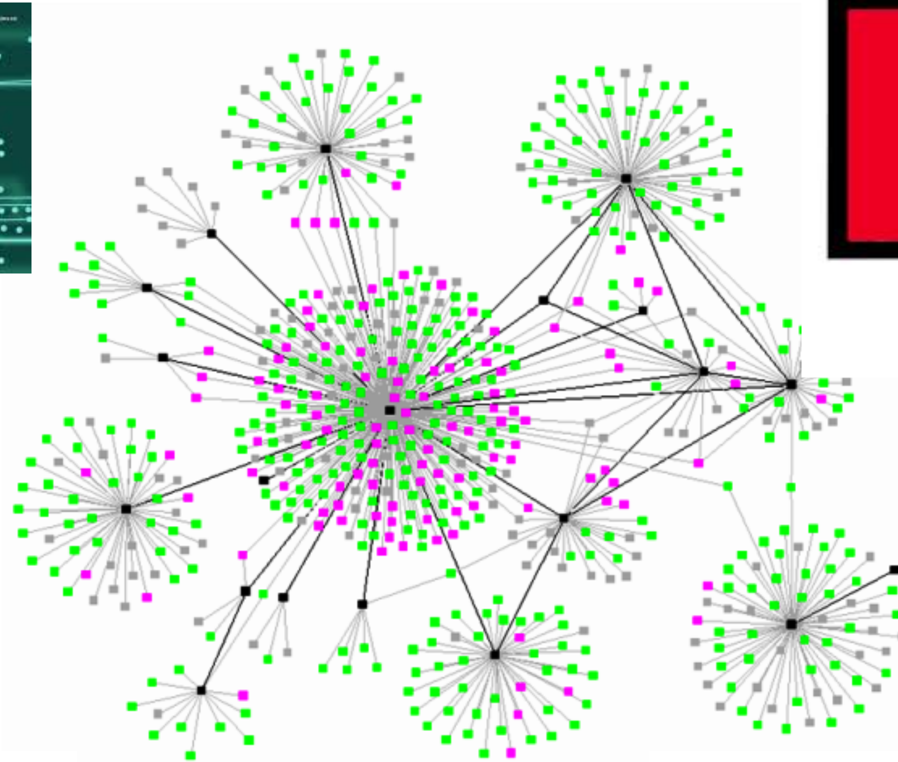
Analytics on Graphs with Trillions of Edges

Laurent Bindschaedler, Jasmina Malicevic,
Amitabha Roy, and Willy Zwaenepoel

Plenty of big graphs





What is big?

"A billion edges isn't cool. You know what is cool? A TRILLION edges."

Avery Ching, Facebook

How to do it? – The HPC Approach

single machine, in memory

How to do it? – The Facebook approach

many machines, in memory

How to do it? – Our "IKEA" approach

one or few machines, out-of-core

How to do it? – Our “IKEA” approach

X-Stream

Chaos

The challenge

- Graph processing produces random accesses
- Performance requires sequential access
- A fortiori for secondary storage

Programming model

- Vertex-centric
- Scatter-gather

Programming model

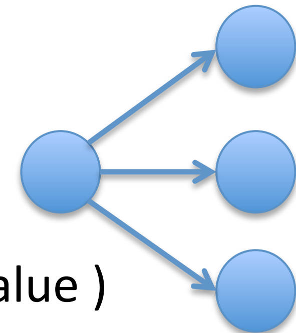
- Vertex-centric
  - Maintain state in vertex
  - Write a vertex program
- Scatter-gather

Programming model

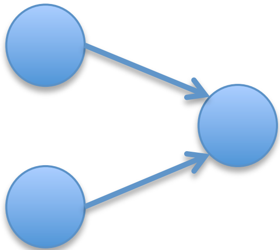
- Vertex-centric
  - Maintain state in vertex
  - Write a vertex program
- Scatter-gather
  - Vertex program has two methods
    - Scatter
    - Gather

Programming model

- Vertex-centric
  - Maintain state in vertex
  - Write a vertex program
- Scatter-gather
  - Vertex program has two methods
    - Scatter
      - For all outgoing edges: new update = f(vertex value)



    - Gather
      - For all incoming edges: vertex value = g(vertex value, update)



A vertex-centric program

Until convergence

/* Scatter phase */

For all vertices

For all outgoing edges: new update = f(vertex value)

/* Gather phase */

For all vertices

For all incoming edges: vertex value = g(vertex value, update)

Can express many graph algorithms

- Pagerank
- Weakly connected components
- Minimum cost spanning tree
- Maximal independent set
- Conductance
- SpMV
- Alternating least squares
- …

X-Stream

- Single-node (multi-core) graph processing
- Goal: all access to storage sequential!
- Two techniques:
  - Edge-centric graph processing
  - Streaming partitions

A vertex-centric program

```
Until convergence
    /* Scatter phase */
    For all vertices
        For all outgoing edges: new update = f( vertex value )
    /* Gather phase */
    For all vertices
        For all incoming edges: vertex value = g( vertex value, update )
```

Observation

Until convergence

/* Scatter phase */

For all vertices

For all outgoing edges: new update = f(vertex value)

/* Gather phase */

For all vertices

For all incoming edges: vertex value = g(vertex value, update)

Observation

Until convergence

/* Scatter phase */

For all vertices

For all outgoing edges: new update = f(vertex value)

/* Gather phase */

For all vertices

For all incoming edges: vertex value = g(vertex value, update)

These are loops over all edges – order does not matter

To edge-centric

Until convergence

/* Scatter phase */

For all edges: new update = f(vertex value)

/* Gather phase */

For all edges: vertex value = g(vertex value, update)

These are loops over all edges – order does not matter

Why is this good?

Until convergence

/* Scatter phase */

For all edges: new update = f(vertex value)

/* Gather phase */

For all edges: vertex value = g(vertex value, update)

Input

Vertex Set

	Value
1	5
2	6
3	3
4 …	12

Edge Set

Source	Destination
4	12
1	23
2	12
3	2
1	6
…	

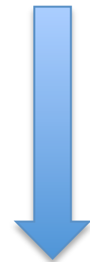
Edge-centric access to edge set

Vertex Set

	Value
1	5
2	6
3	3
4	12
…	

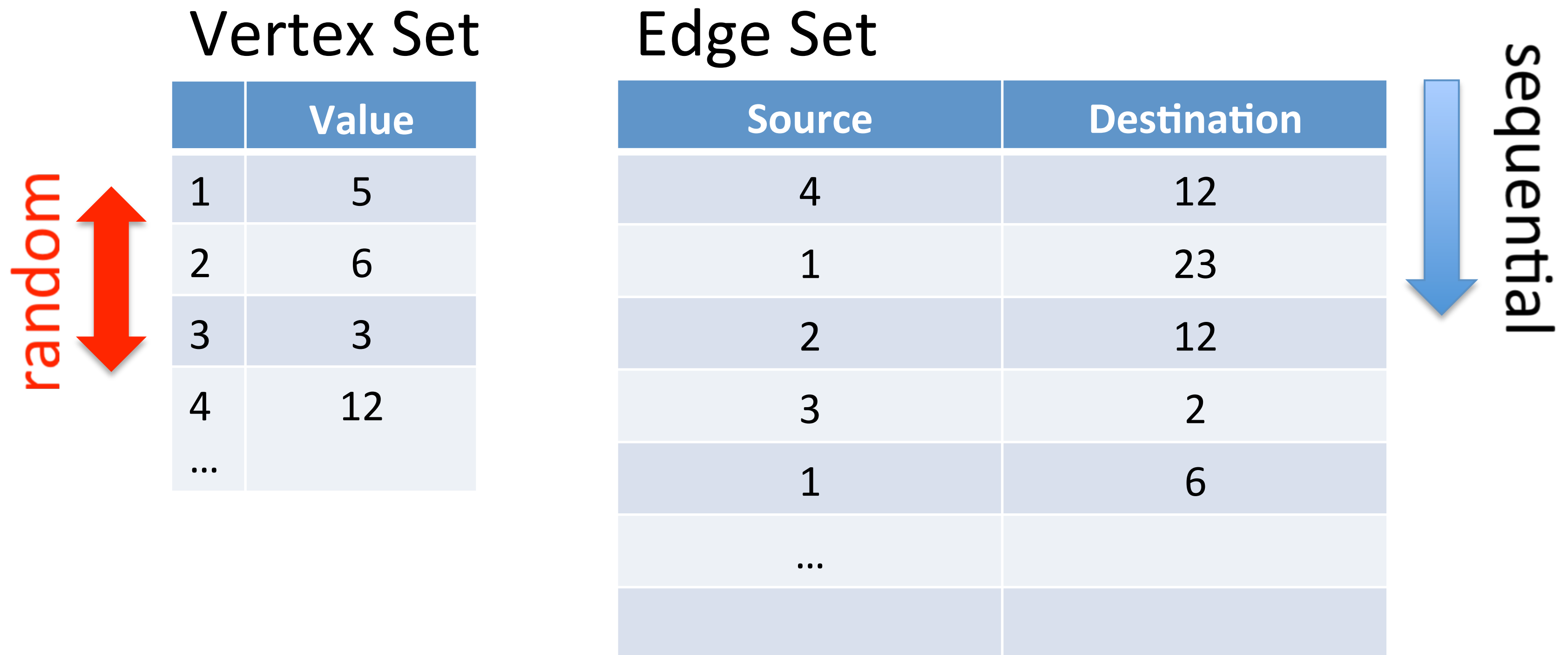
Edge Set

Source	Destination
4	12
1	23
2	12
3	2
1	6
…	



sequential

But ...



Vertex Set

	Value
1	5
2	6
3	3
4	12
...	

Edge Set

Source	Destination
4	12
1	23
2	12
3	2
1	6
...	

Streaming Partition

- Partition (V',E') of graph (V,E) such that
  - V' fits in *memory*
  - E' contains all edges *originating* in V'
- Created during pre-processing

Input

Vertex Set

	Value
1	5
2	6
3	3
4	12
…	

Edge Set

Source	Destination
4	12
1	23
2	12
3	2
1	6
…	

Creating streaming partitions

Partition 1

Vertex Set

	Value
1	5
2	6

in memory

Edge Set

Source	Destination
1	23
2	12
1	6
…	

Partition 2

Vertex Set

	Value
3	1
4	2

in memory

Edge Set

Source	Destination
4	12
3	6
…	

…

Scatter using streaming partitions

- Iterate over partitions

- For all partitions
  - Read vertex set from storage
  - Stream edge set from storage (in big chunks)

Access to streaming partitions - 1

Vertex Set

	Value
1	5
2	6

in memory

Access to streaming partitions - 1

Vertex Set

	Value
1	5
2	6

in memory

Edge Set

Source	Destination
1	23
2	12
1	6
...	

Access to streaming partitions - 1

Vertex Set

	Value
1	5
2	6

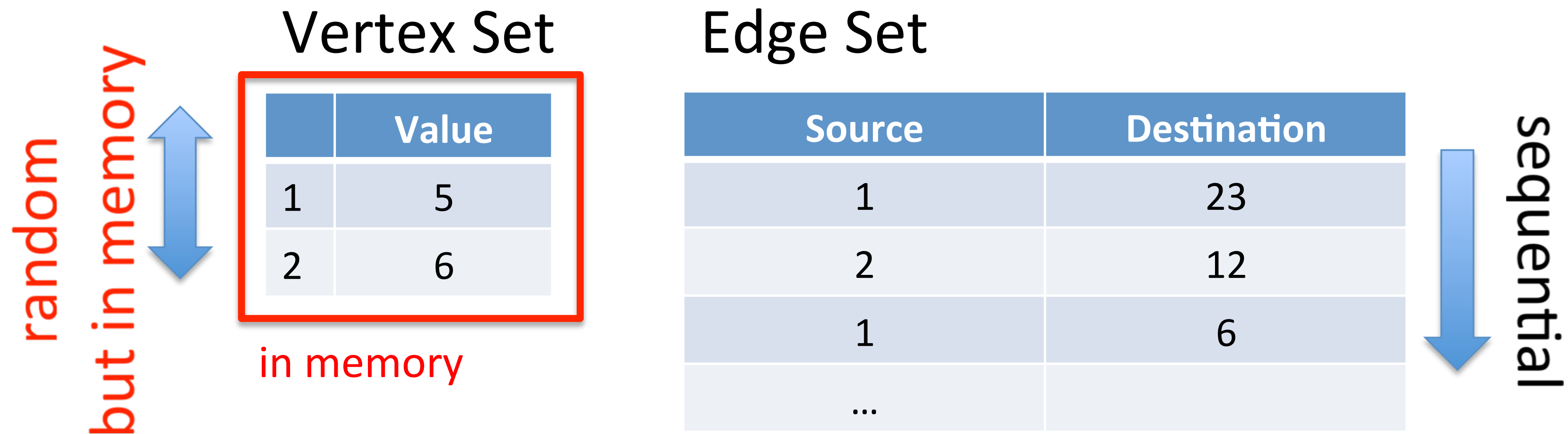
in memory

Edge Set

Source	Destination
1	23
2	12
1	6
...	

sequential

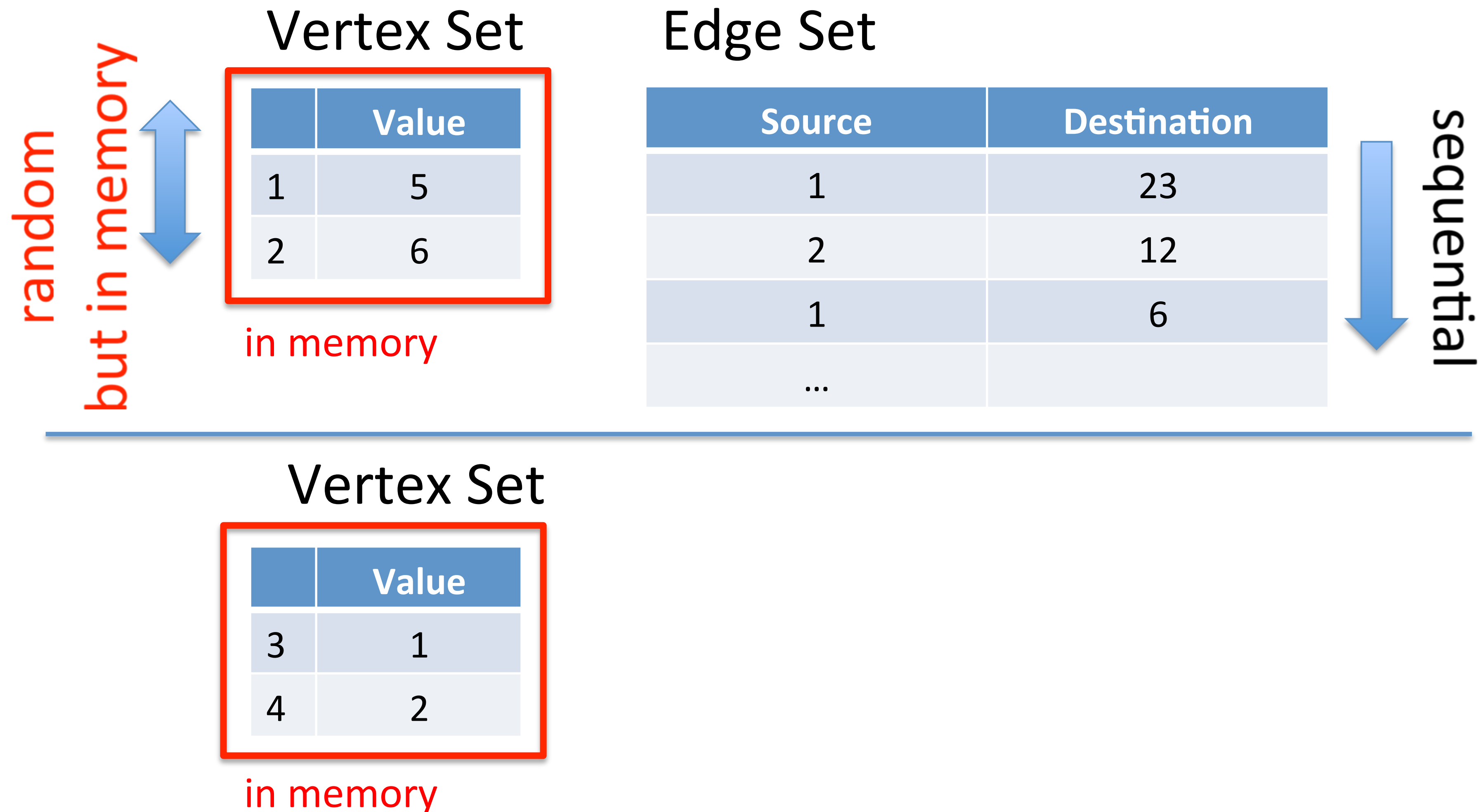
Access to streaming partitions - 1



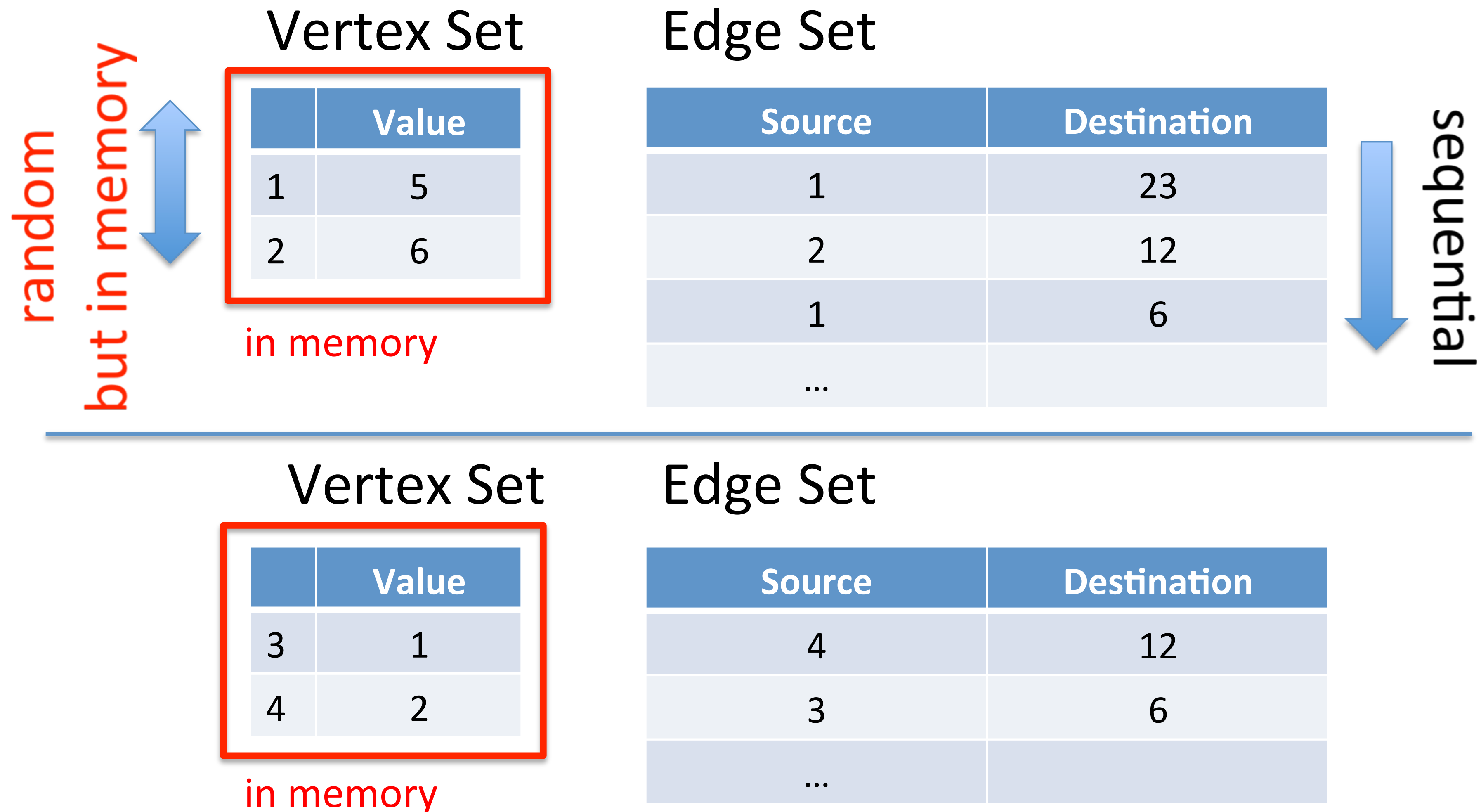
	Value
1	5
2	6

Source	Destination
1	23
2	12
1	6
...	

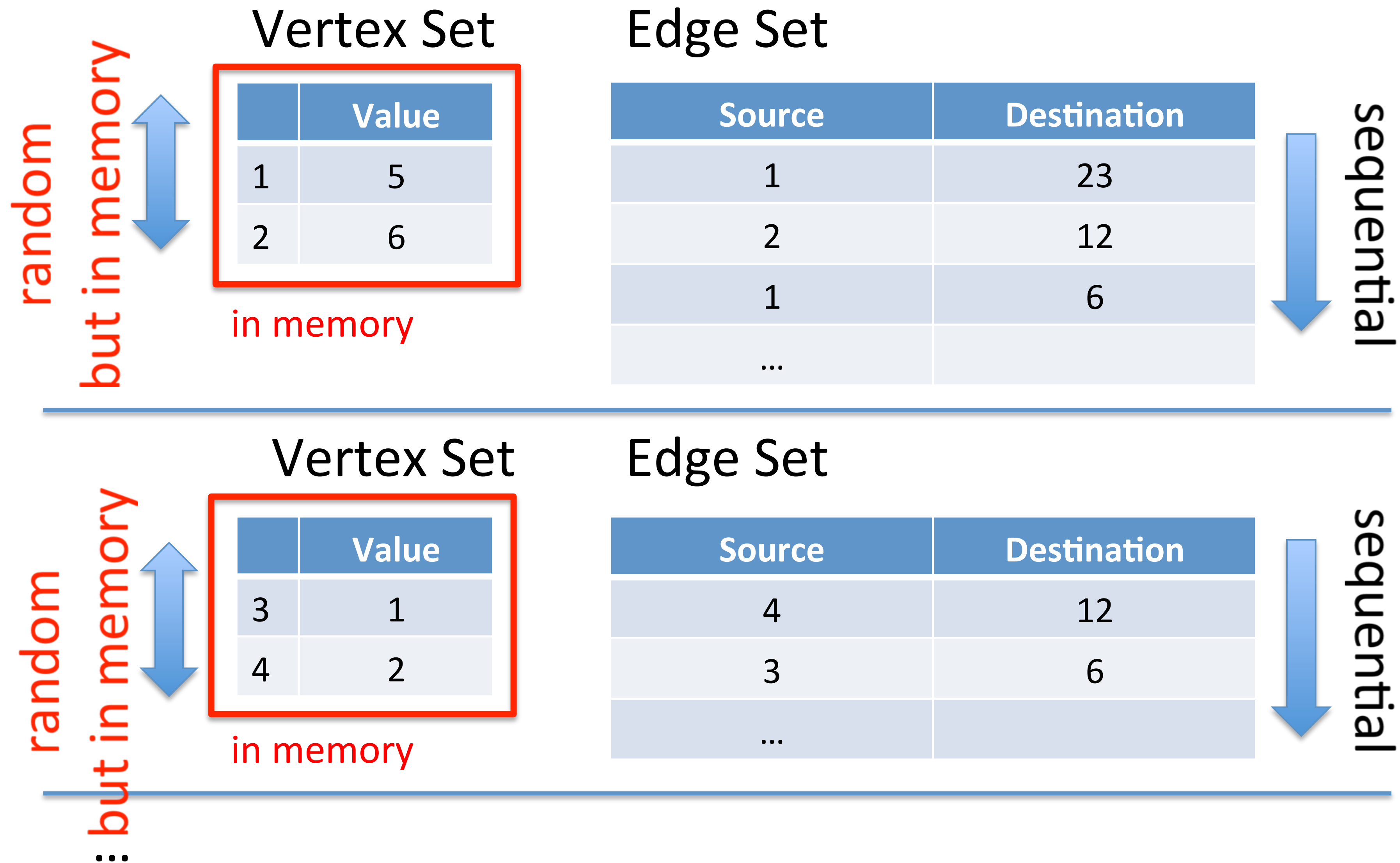
Access to streaming partitions - 2



Access to streaming partitions - 2



Access to streaming partitions - 2



Accesses

- Sequential access to storage for E'
- Random access for V' but in memory

- Almost all access to storage is sequential

What happens with updates?

Until convergence

/* Scatter phase */

For all edges: new update = f(vertex value)

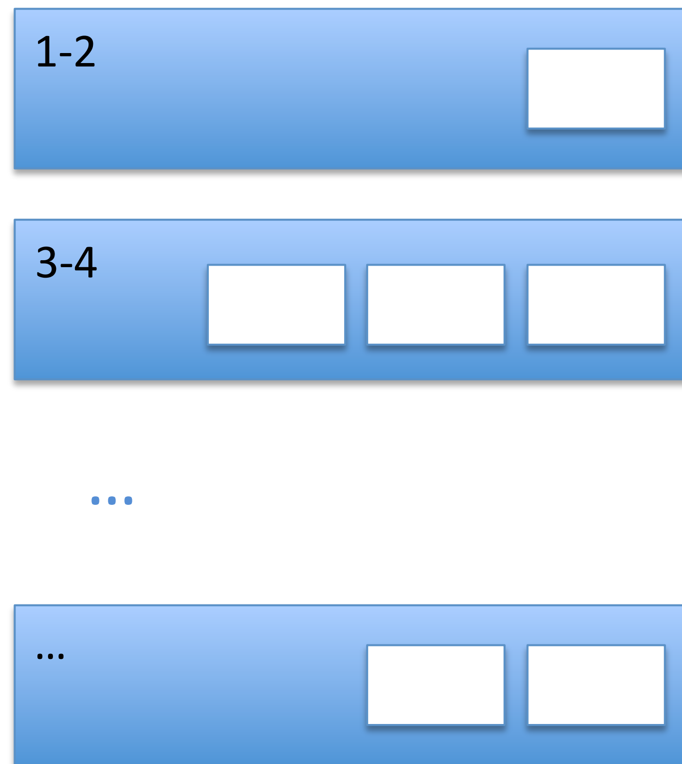
/* Gather phase */

For all edges: vertex value = g(vertex value, update)

What happens with updates?

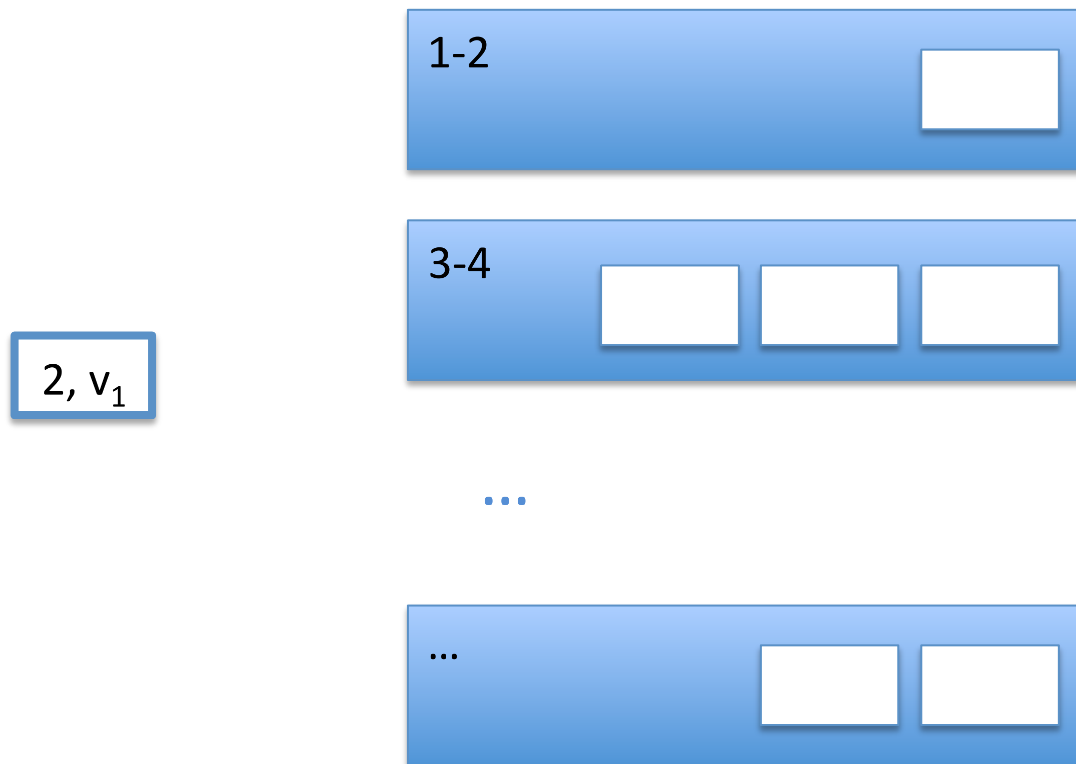
- Update = (target vertex, value)
- Updates are
  - Binned according to partition of *target* vertex
  - Buffered in memory
  - Streamed to storage (sequentially)

What happens with updates?



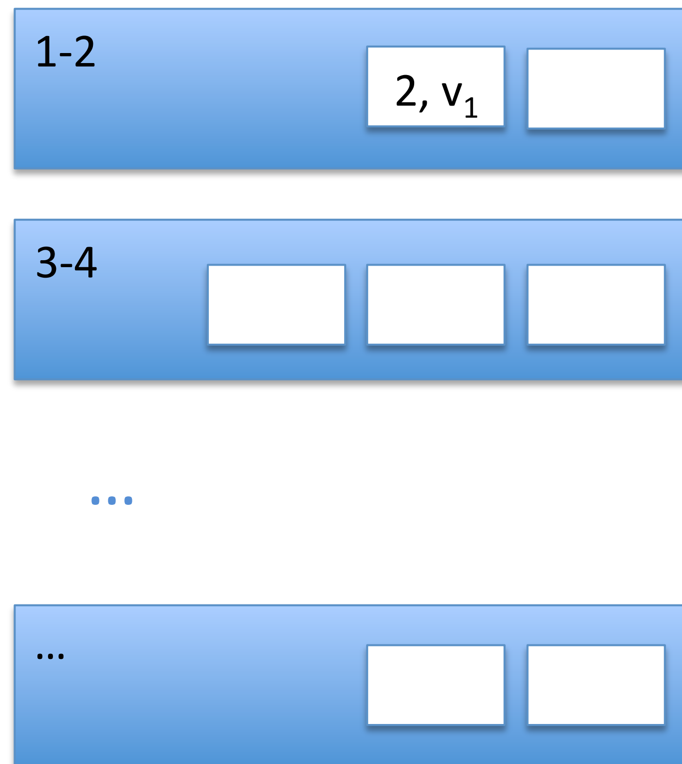
in memory

What happens with updates?



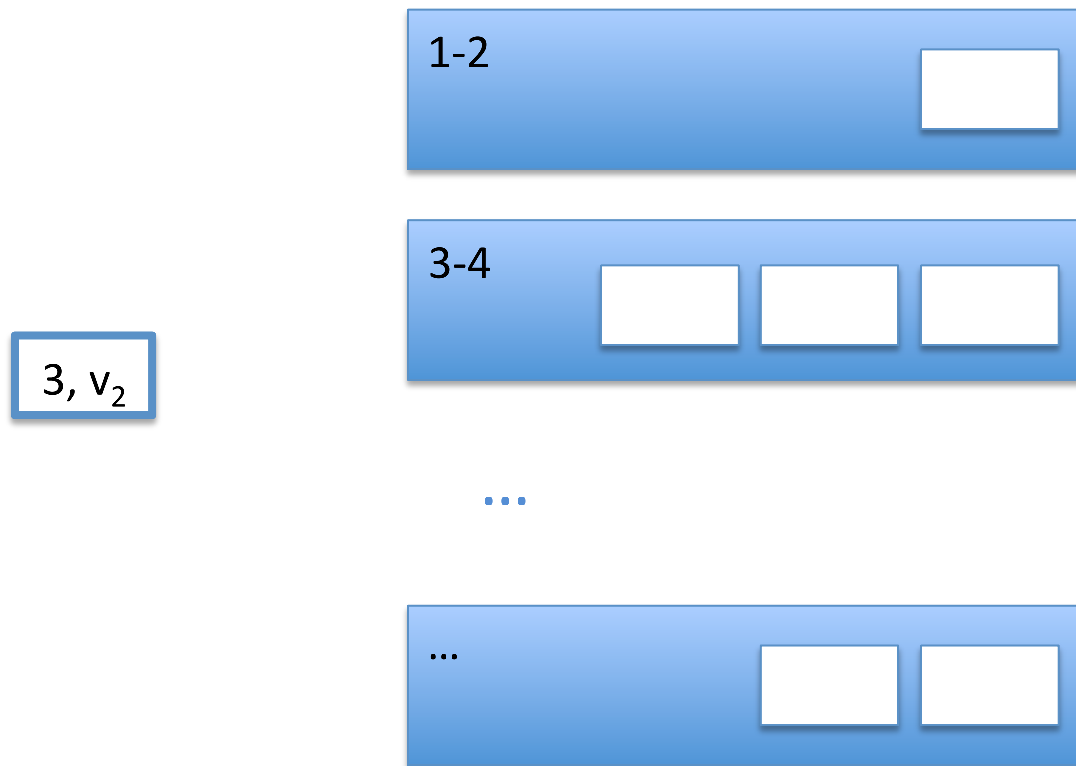
in memory

What happens with updates?

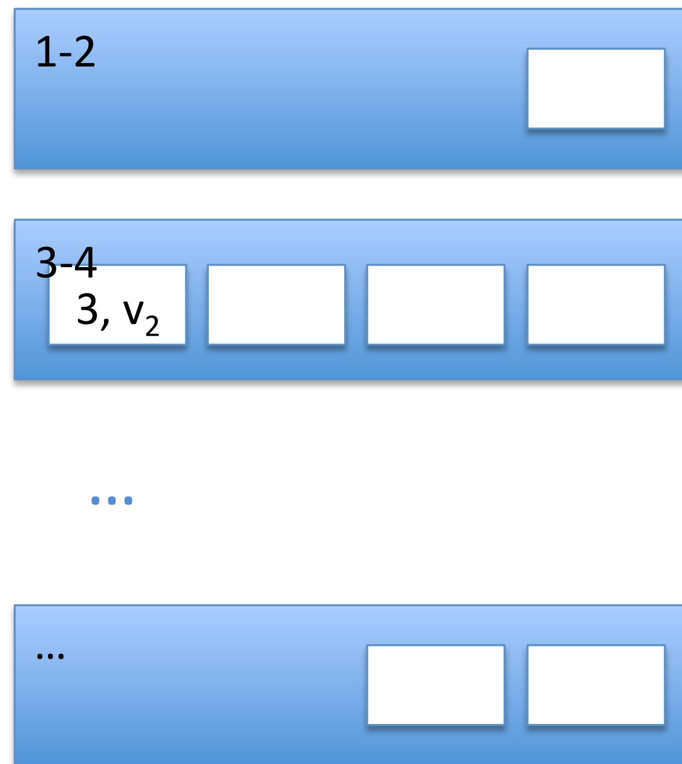


in memory

What happens with updates?

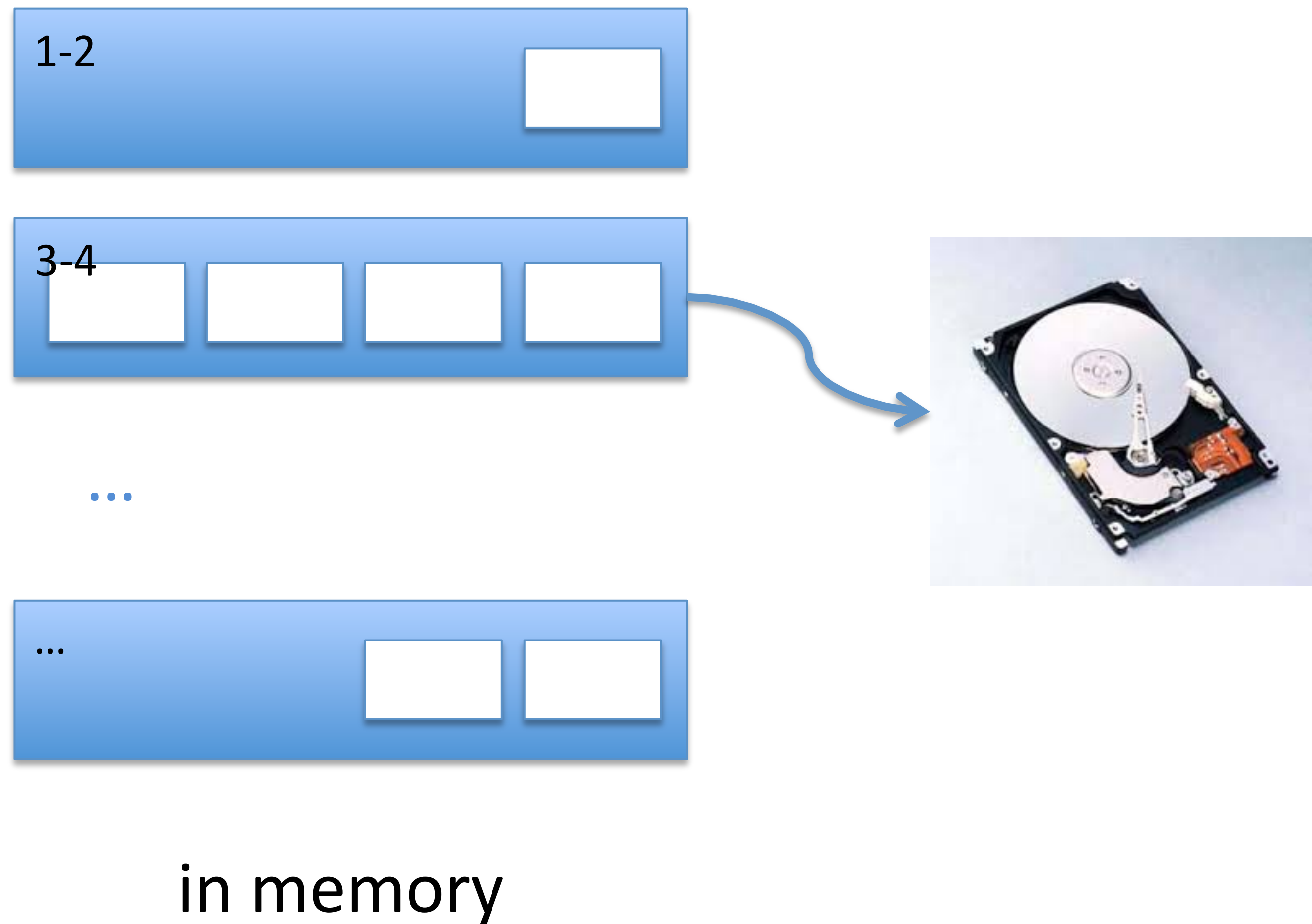


What happens with updates?



in memory

What happens with updates?



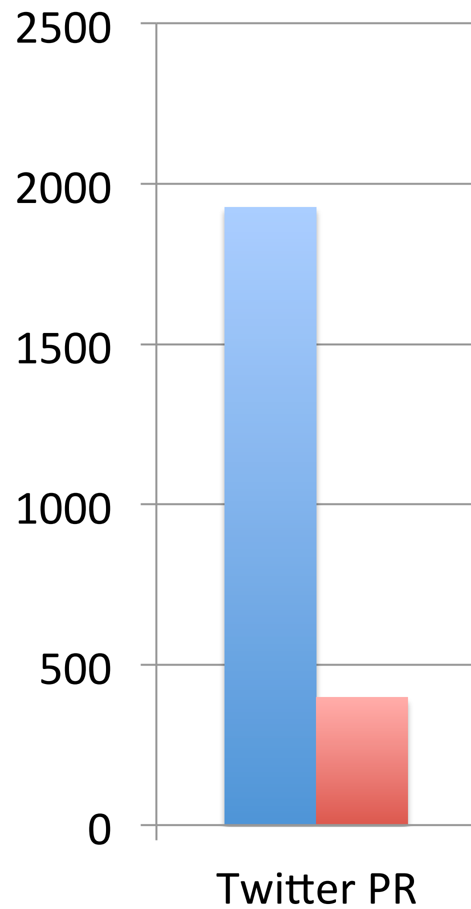
What happens with updates?

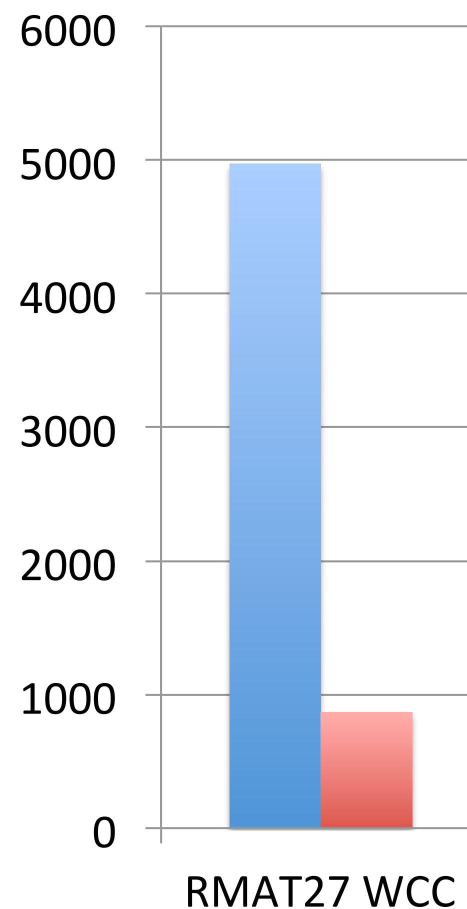
- Sequentially written during scatter
- Sequentially read during gather

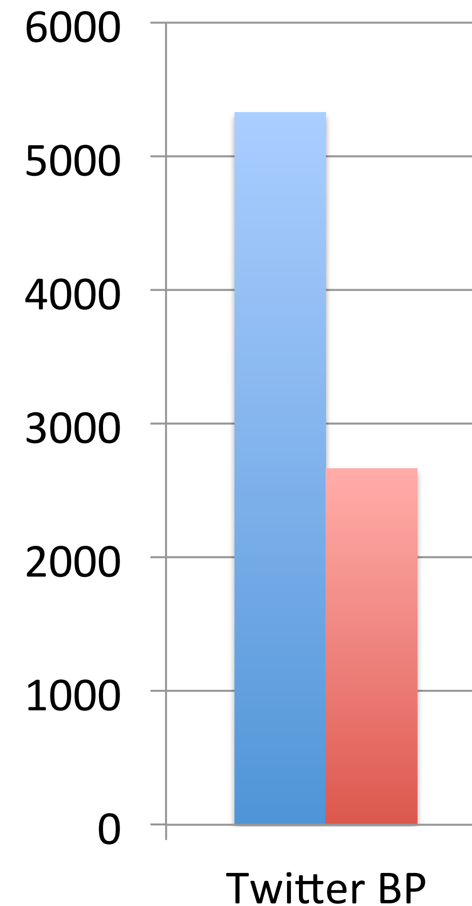
Experimental Evaluation: Comparison with GraphChi

- Use same storage medium: SSD
- Use same benchmarks:
  - Twitter Pagerank
  - RMAT27 WCC
  - Twitter Belief Propagation

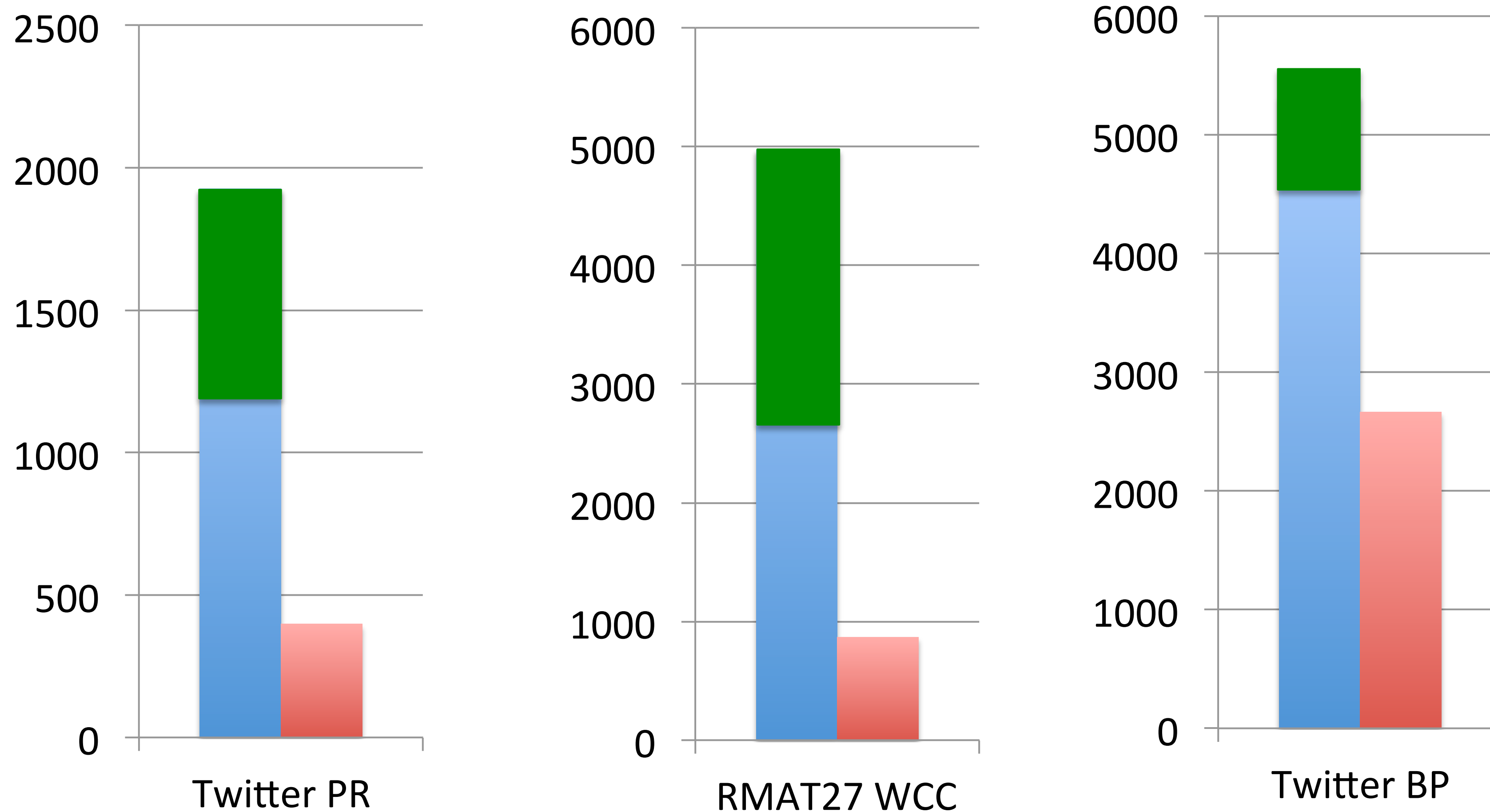
GraphChi vs X-Stream
Runtime comparison (in secs.)



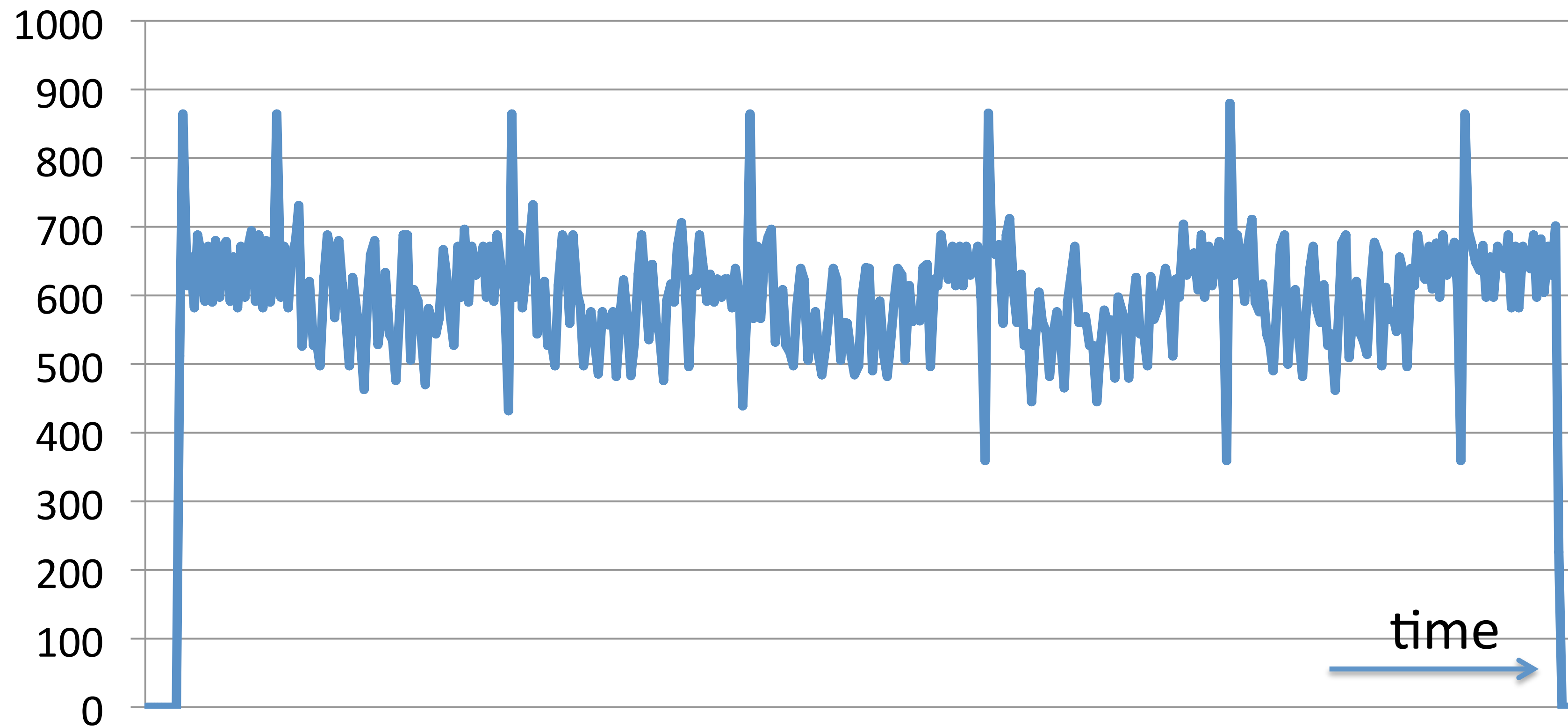




GraphChi - Preprocessing vs X-Stream
Runtime comparison (in secs.)

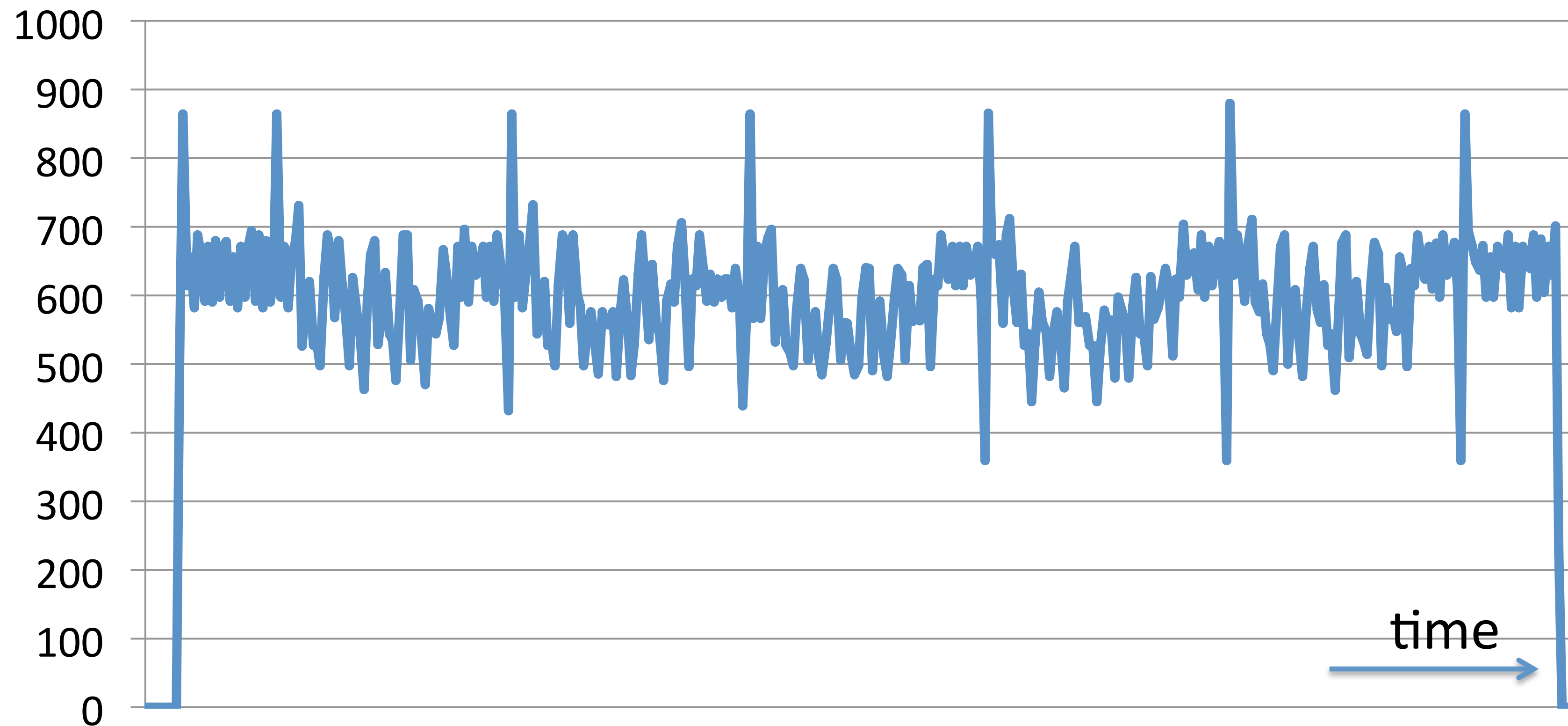


Fundamental reason



X-Stream bandwidth utilization (PageRank, 5 iterations)

Fundamental reason



X-Stream bandwidth utilization (PageRank, 5 iterations)

Runs constantly at near-maximum I/O bandwidth

X-Stream limitations

- Capacity: amount of storage on single machine
- Bandwidth: storage bandwidth on single machine

Chaos goals

- Extend to X-Stream to a cluster

- Goals:
  - Capacity: aggregate storage on all machines
  - Bandwidth: aggregate bandwidth on all machines

Back to sequential X-Stream design

- Iterate over partitions

- For all partitions
  - Read vertex set from storage
  - Stream edge set from storage (in big chunks)

Observation

- Iterate over partitions

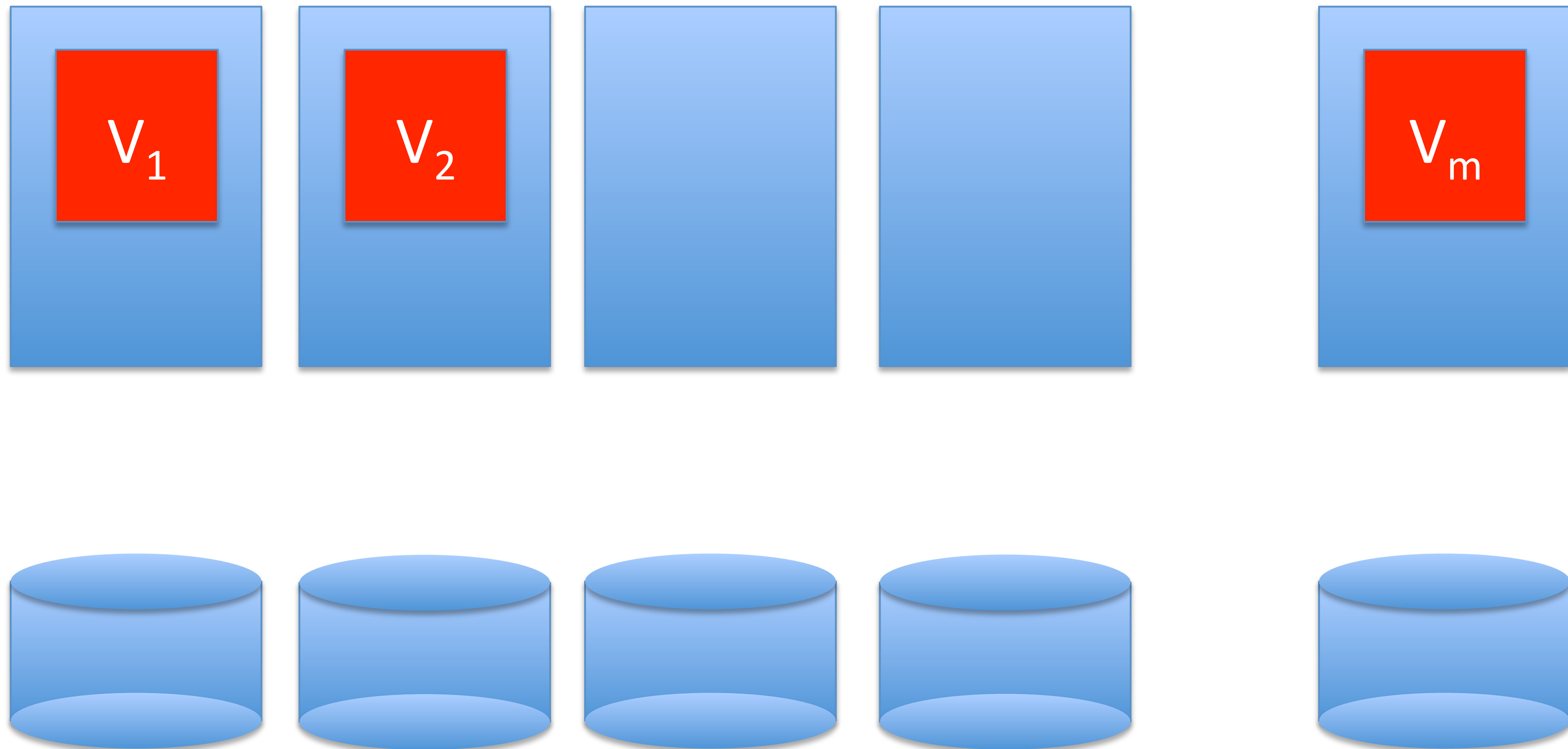
- For all partitions
  - Read vertex set from storage
  - Stream edge set from storage (in big chunks)

- Streaming partitions are independent

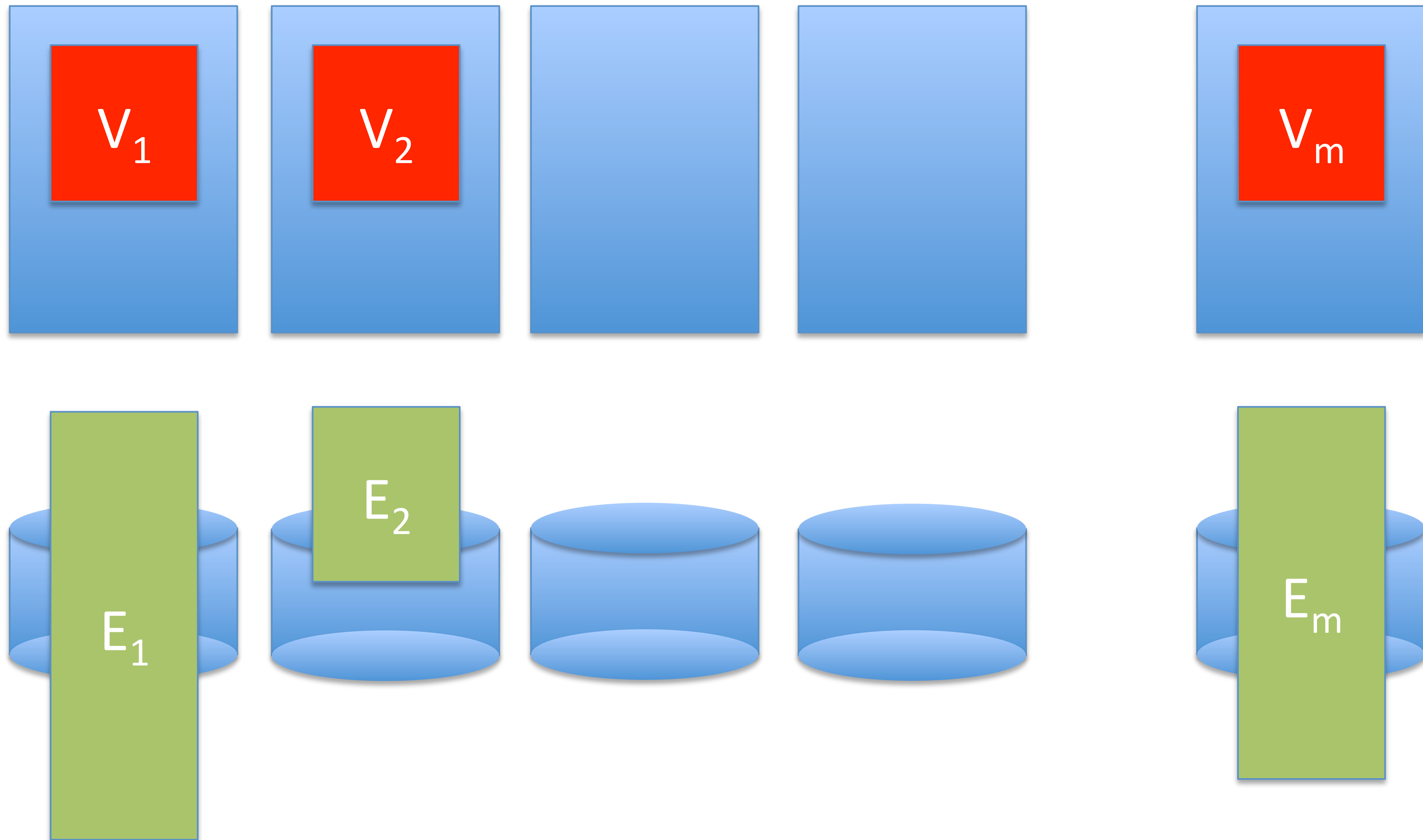
Distribution

- Iterate in parallel over partitions
- For all partitions
  - Read vertex set from storage
  - Stream edge set from storage (in big chunks)

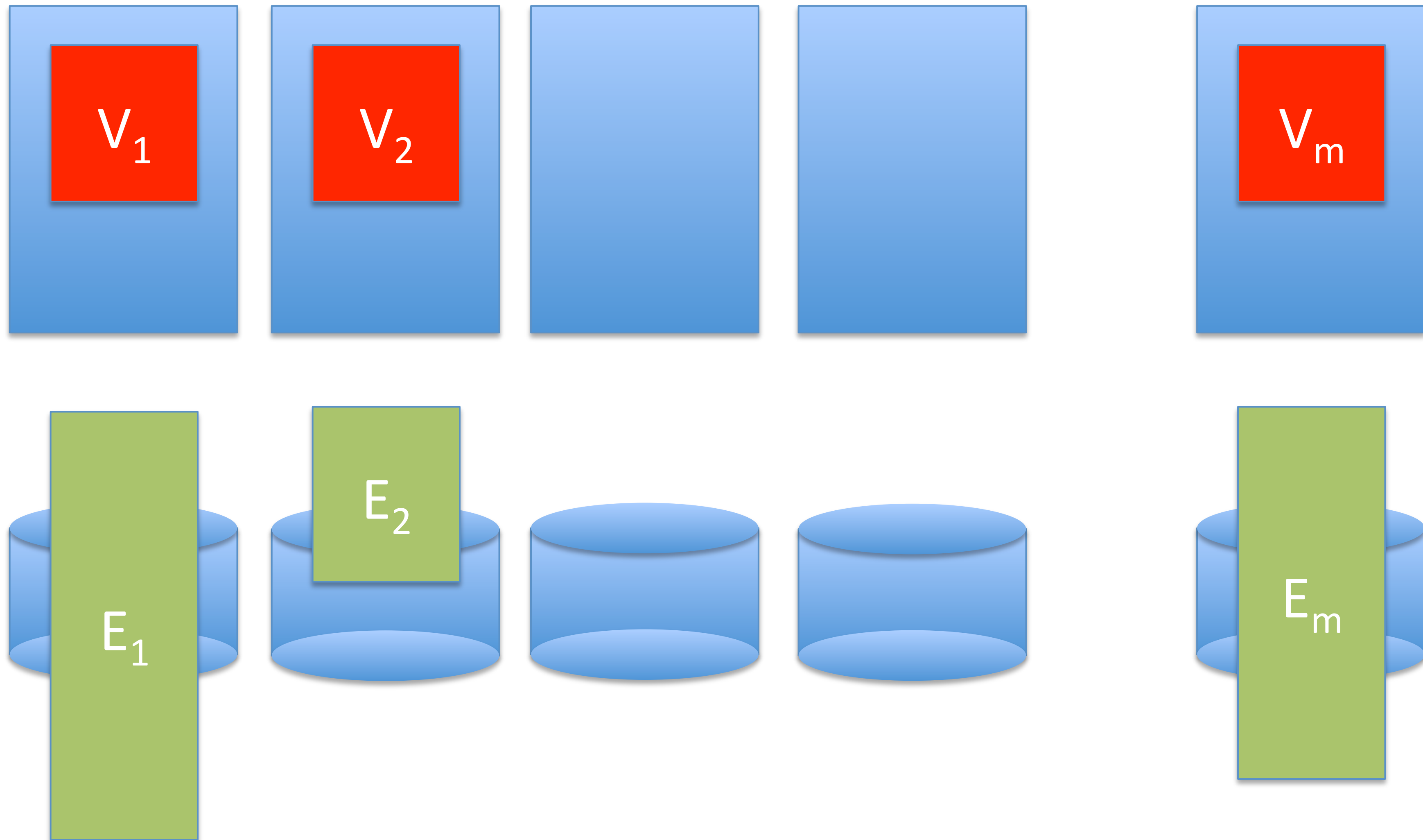
Vertex distribution



Edge distribution



Problem: load imbalance



Dealing with imbalance

- I/O imbalance: “flat” storage
- Computational imbalance: work stealing

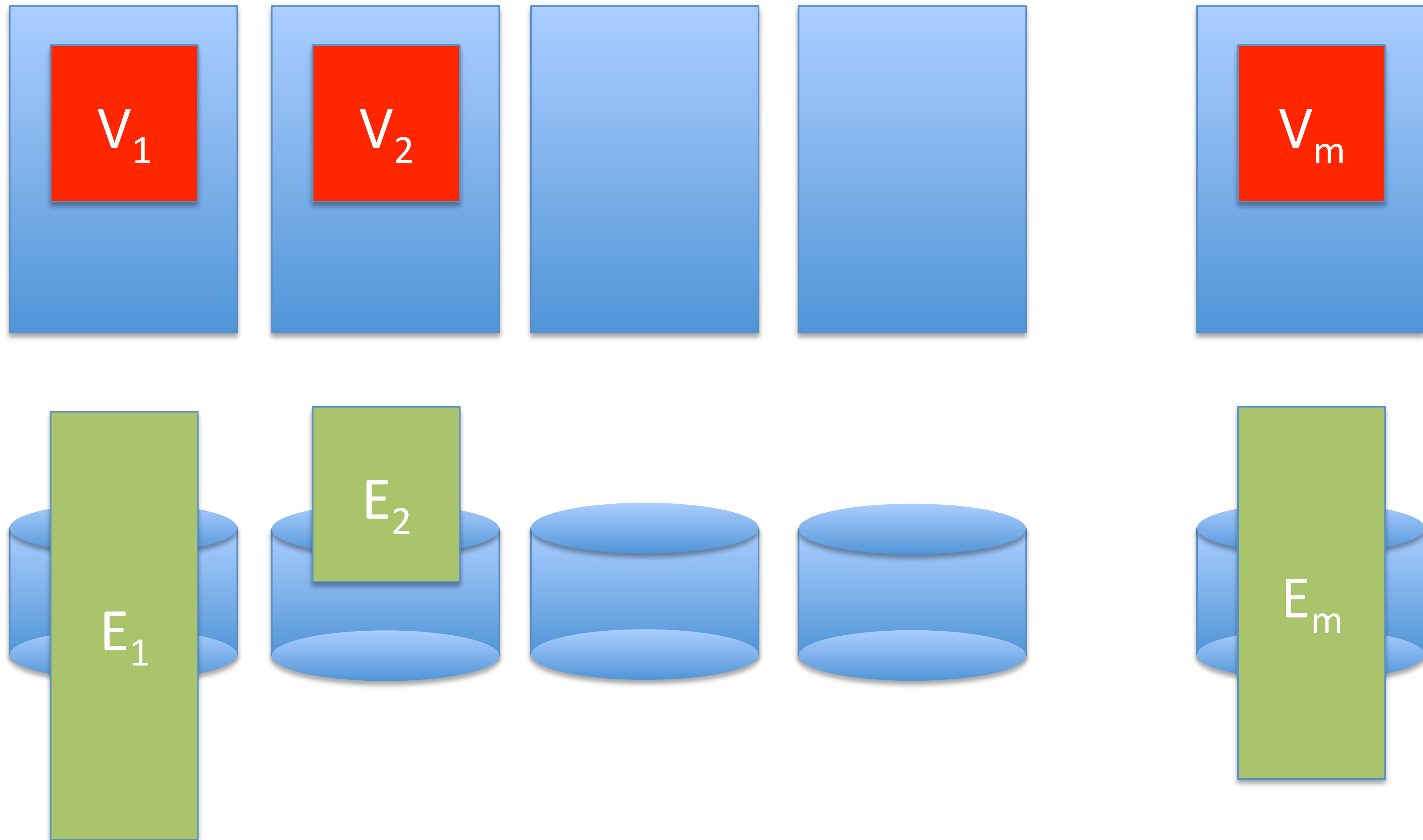
Dealing with imbalance

- ➡ I/O imbalance: “flat” storage
- Computational imbalance: work stealing

Insight

- For secondary storage in a cluster
  - Remote bandwidth ~ local bandwidth
- Locality hardly matters

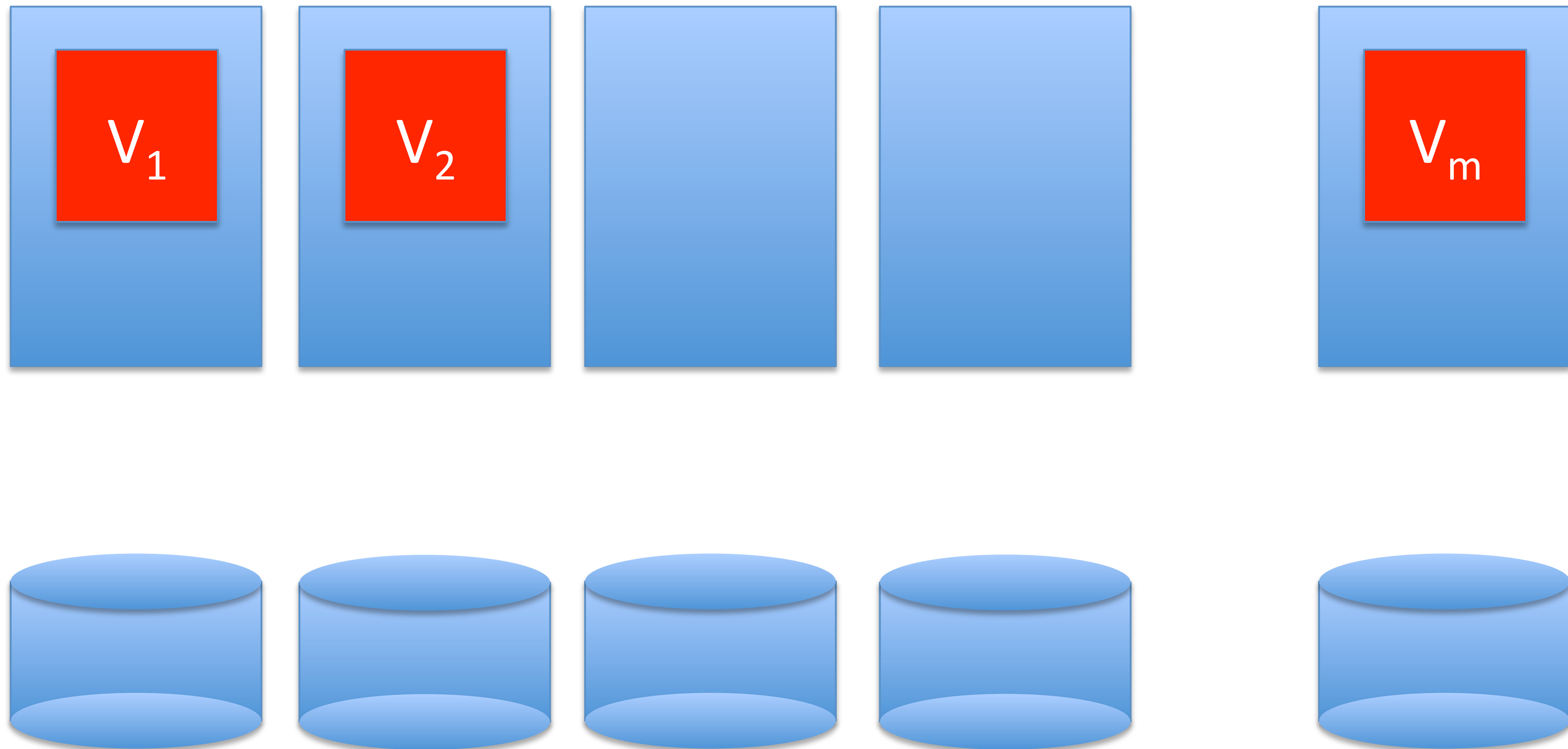
There is no point in putting vertices and edges of a partition together



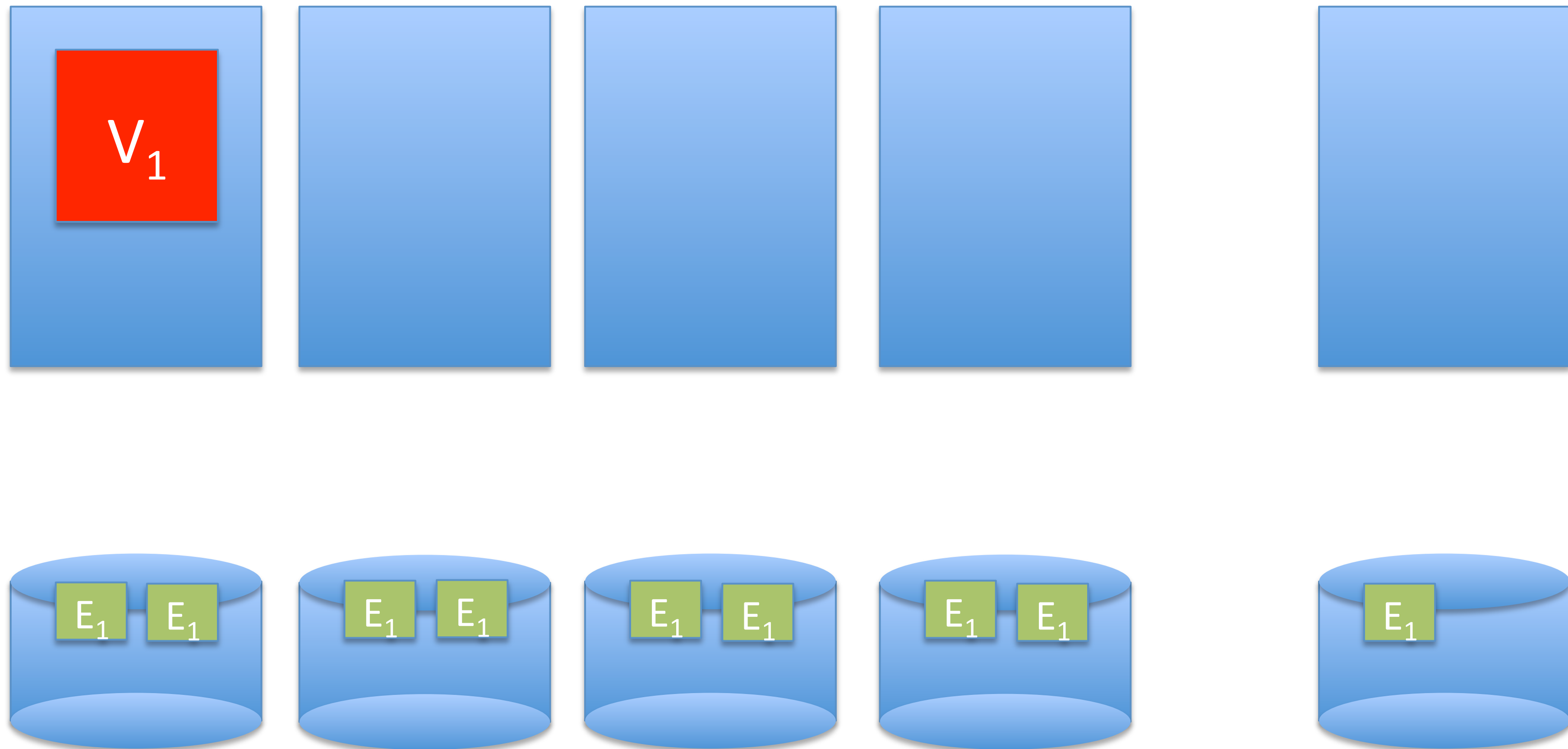
Instead

- Stripe graph data across nodes
  - Edge lists
  - Update lists

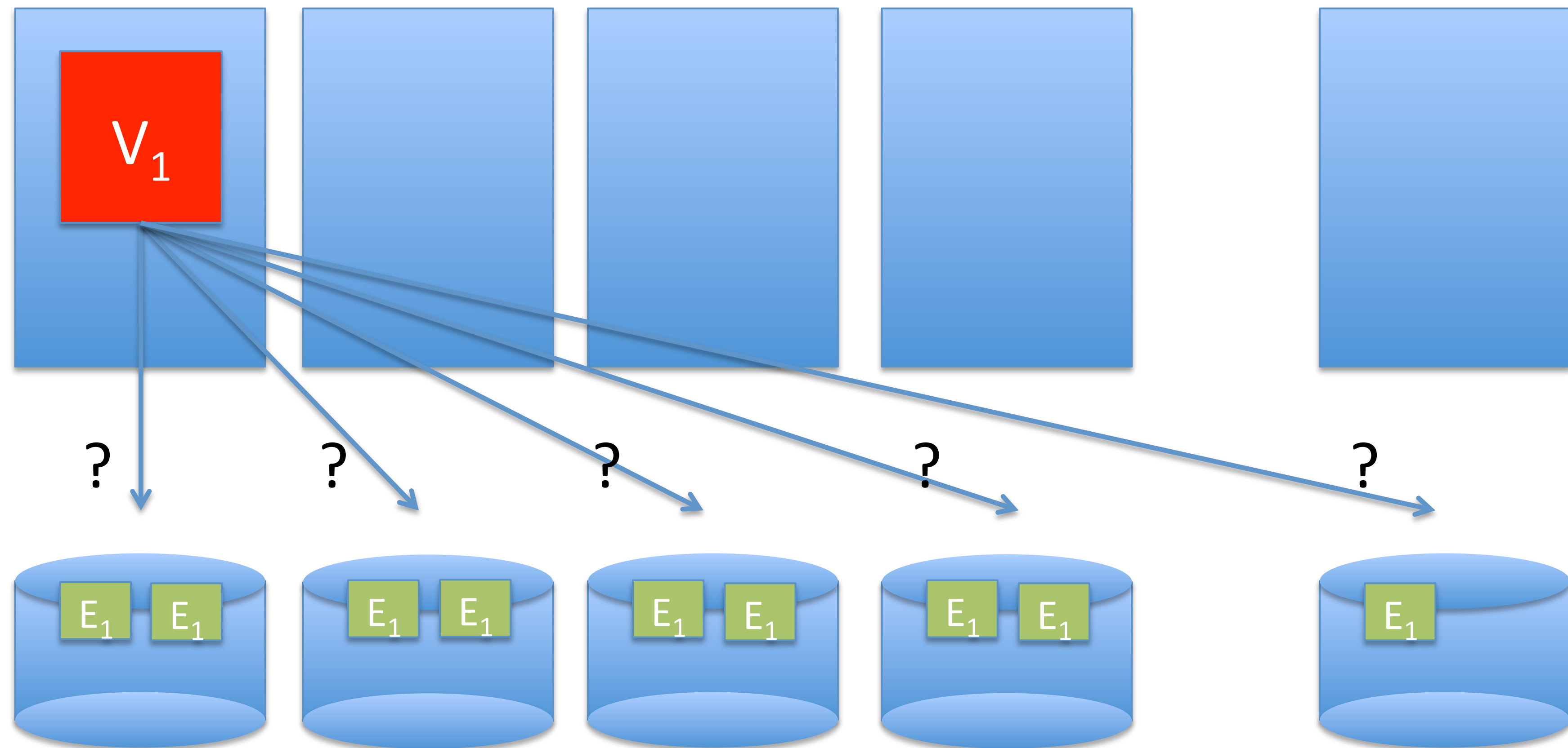
Vertex distribution



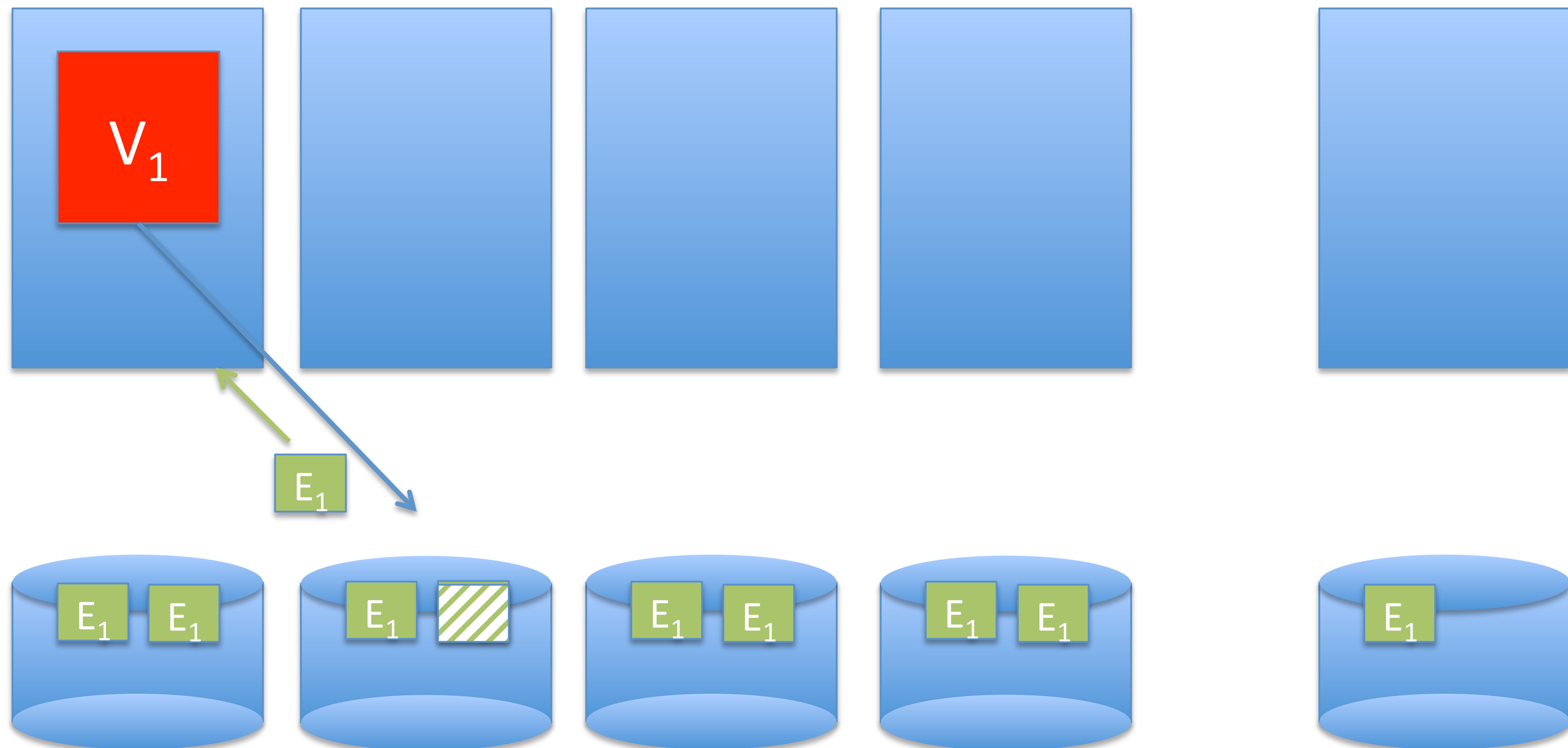
Edge distribution for V_1



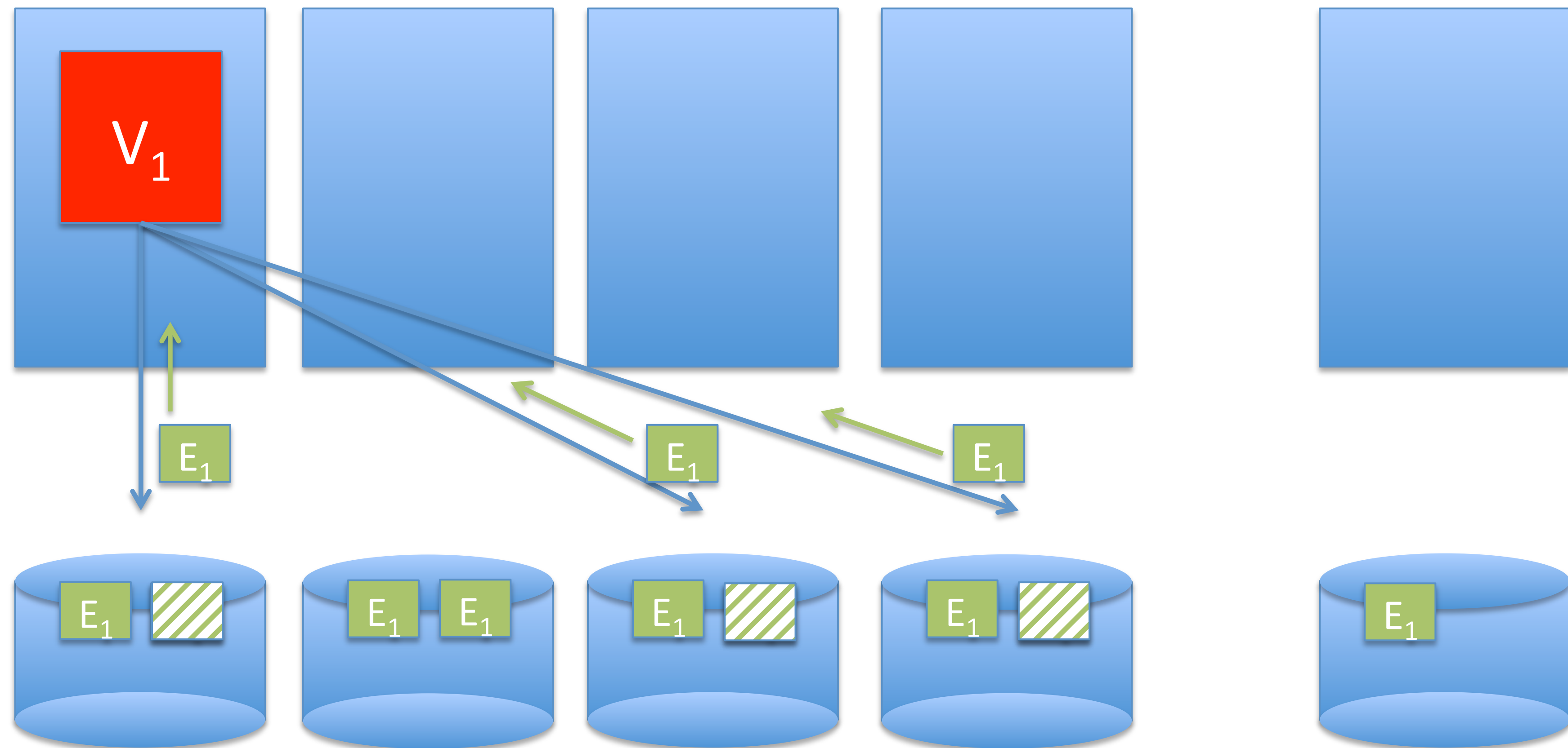
From where to read next edge stripe?



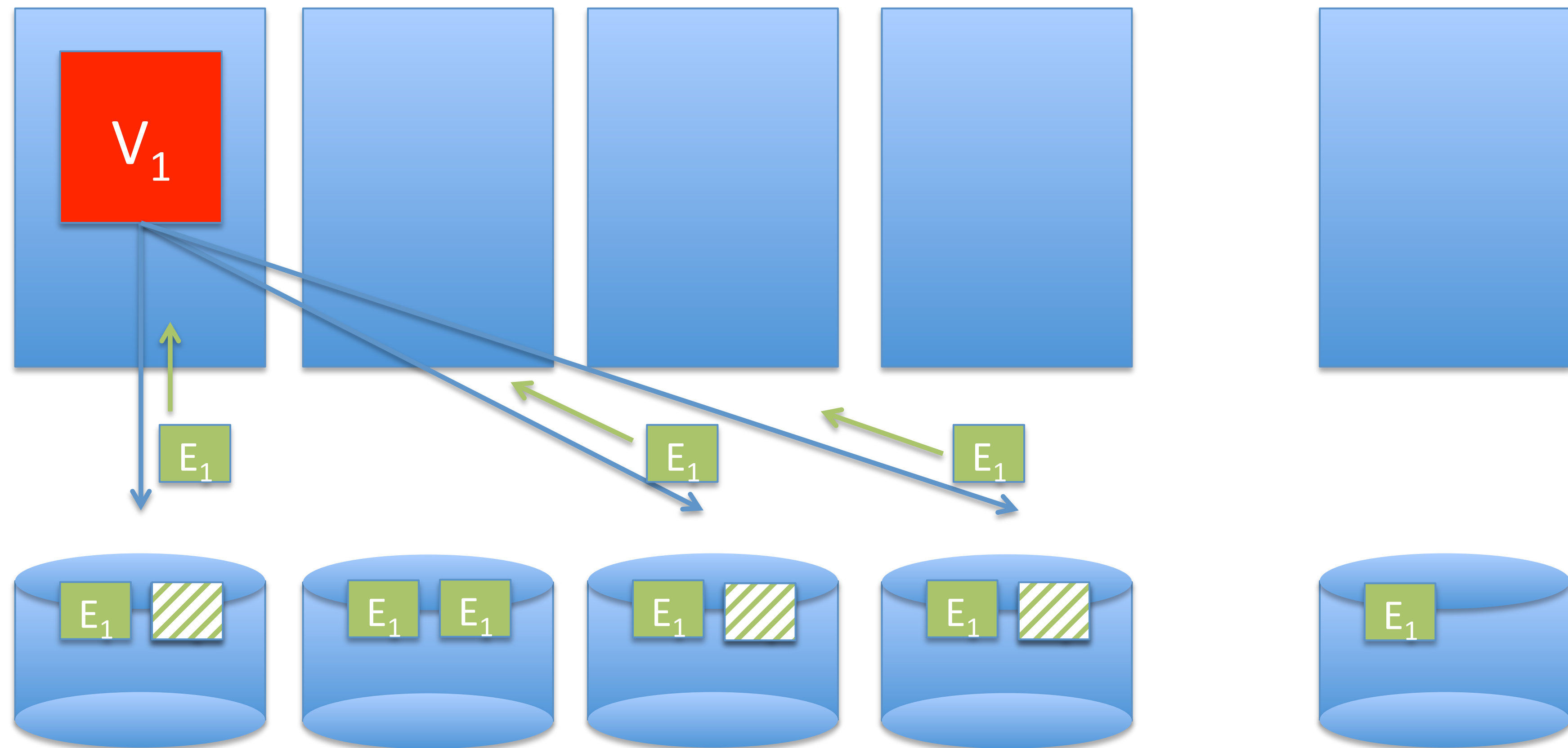
Answer: It can read any random stripe (that has not been read)



In fact, it reads several random stripes

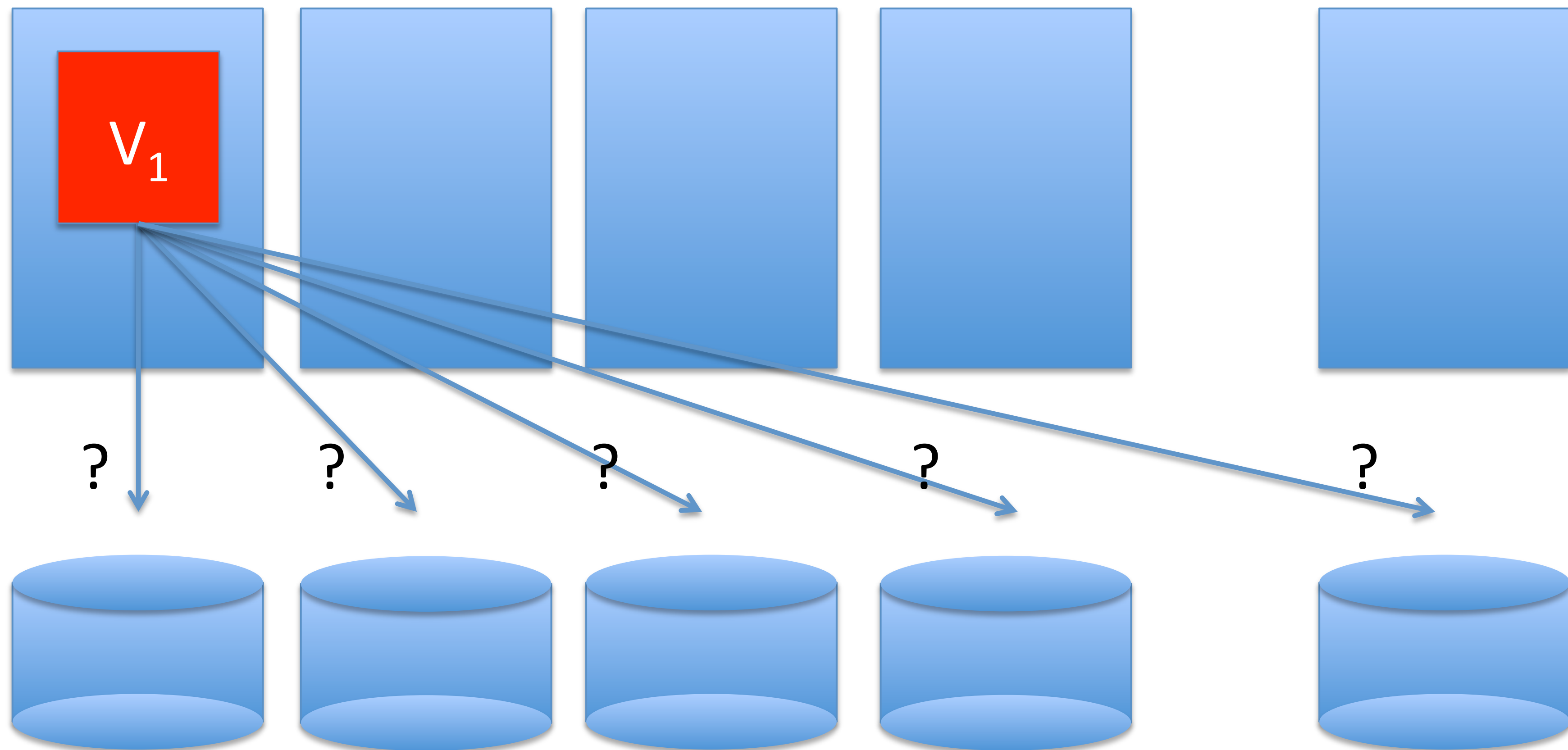


Final note on reading edge stripes

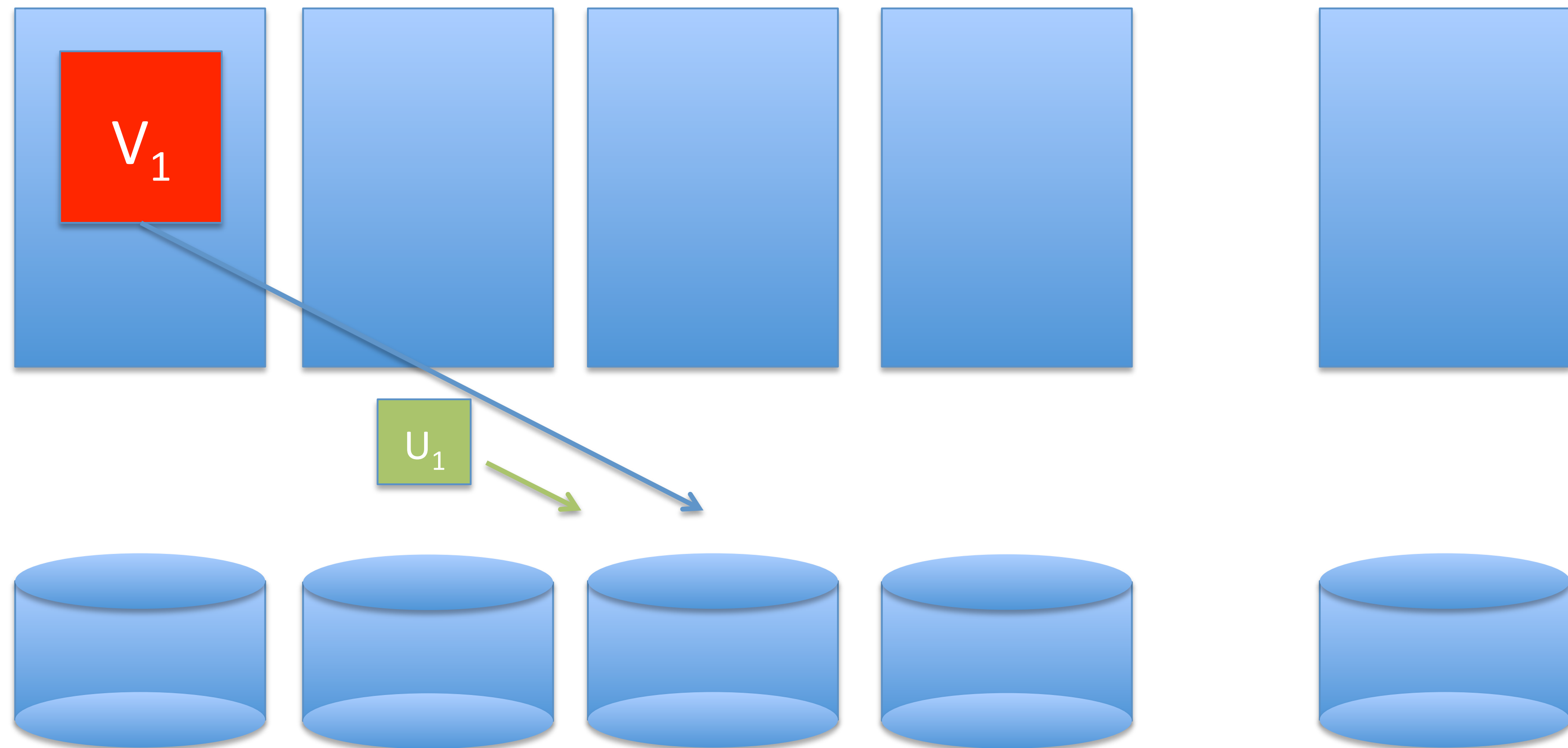


Storage side maintains what has and has not been read

Where to write update stripe?



Answer: choose any device at random



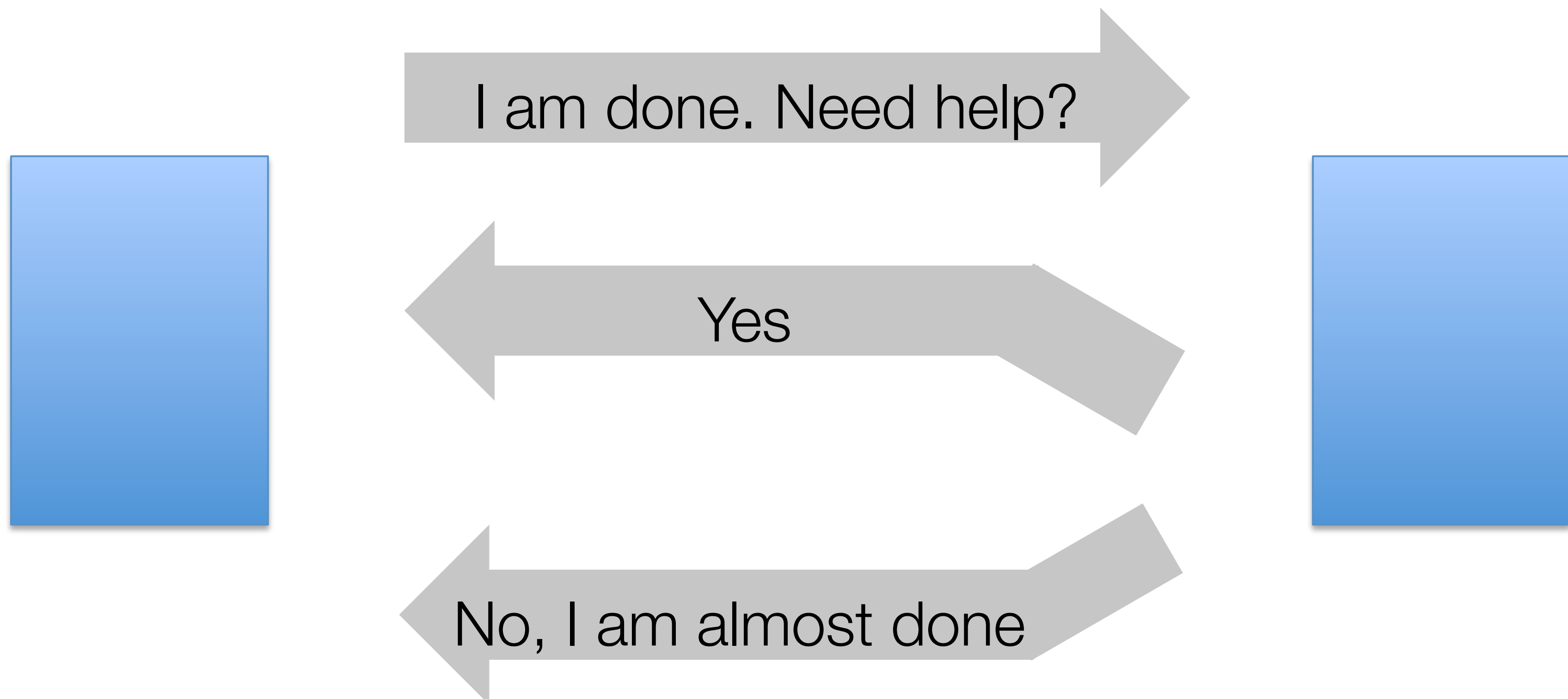
Chaos: I/O design: summary

- “Flat” storage
- Without any access ordering
- Without any central entity

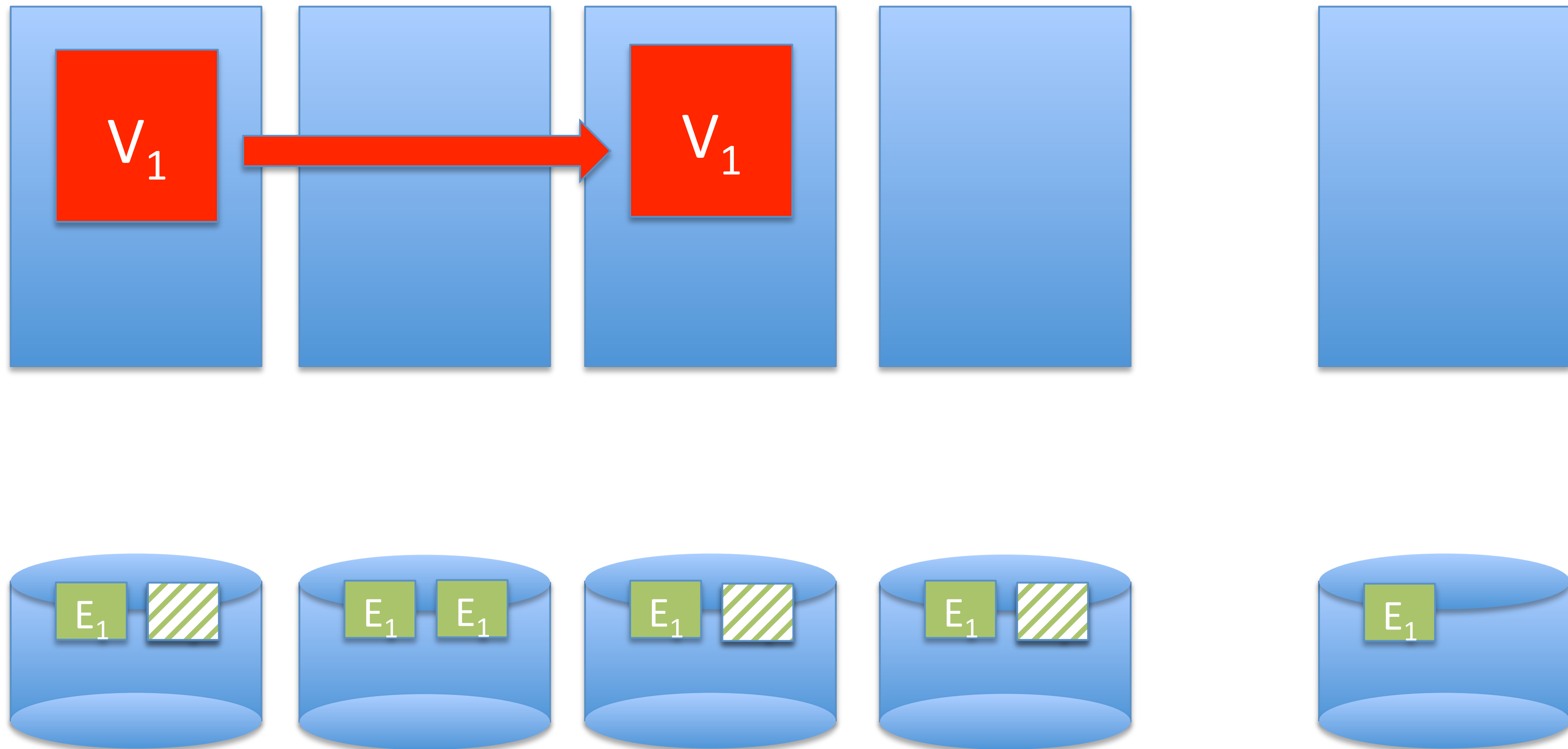
Dealing with imbalance

- I/O imbalance: “flat” storage

Work stealing



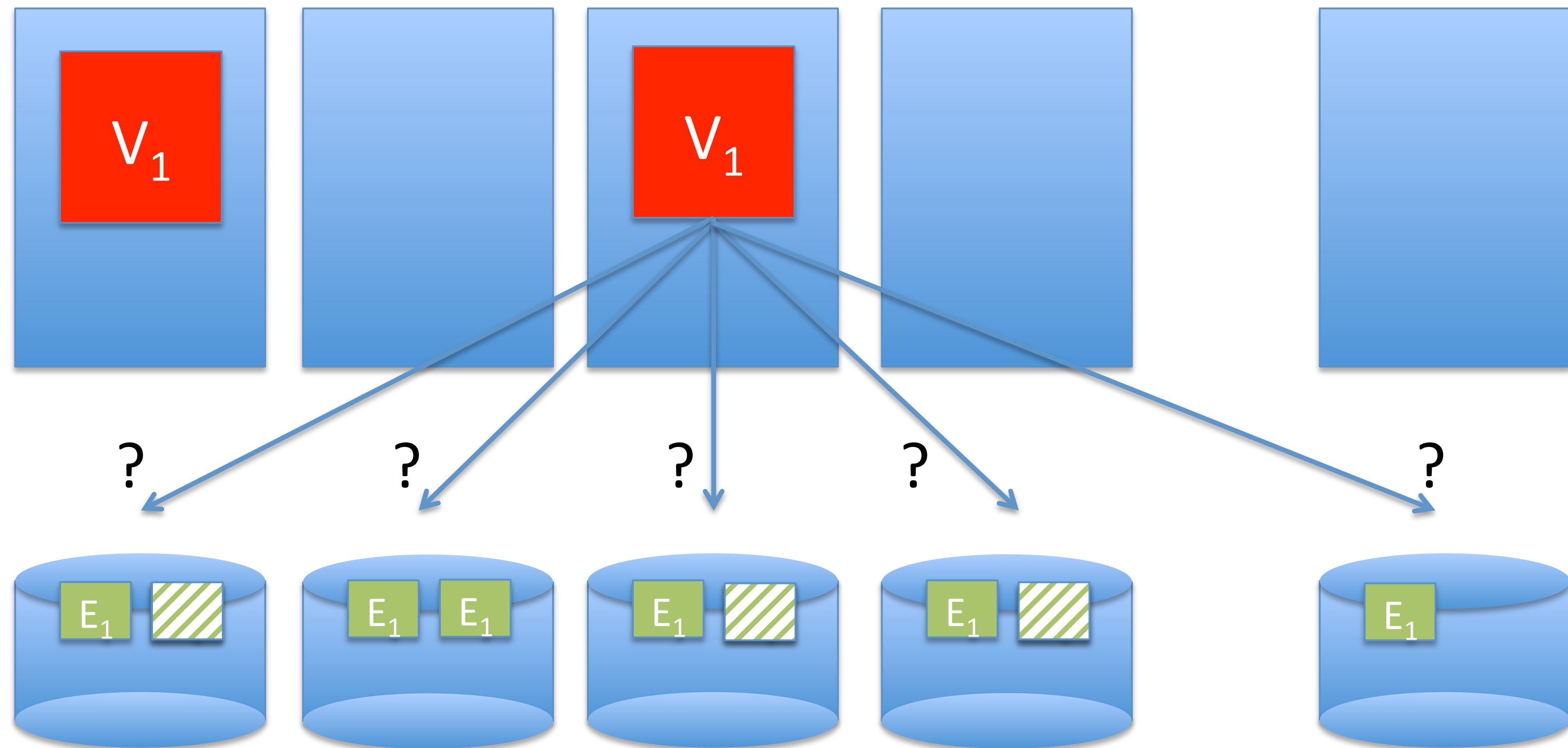
Work stealing: Copy vertex set



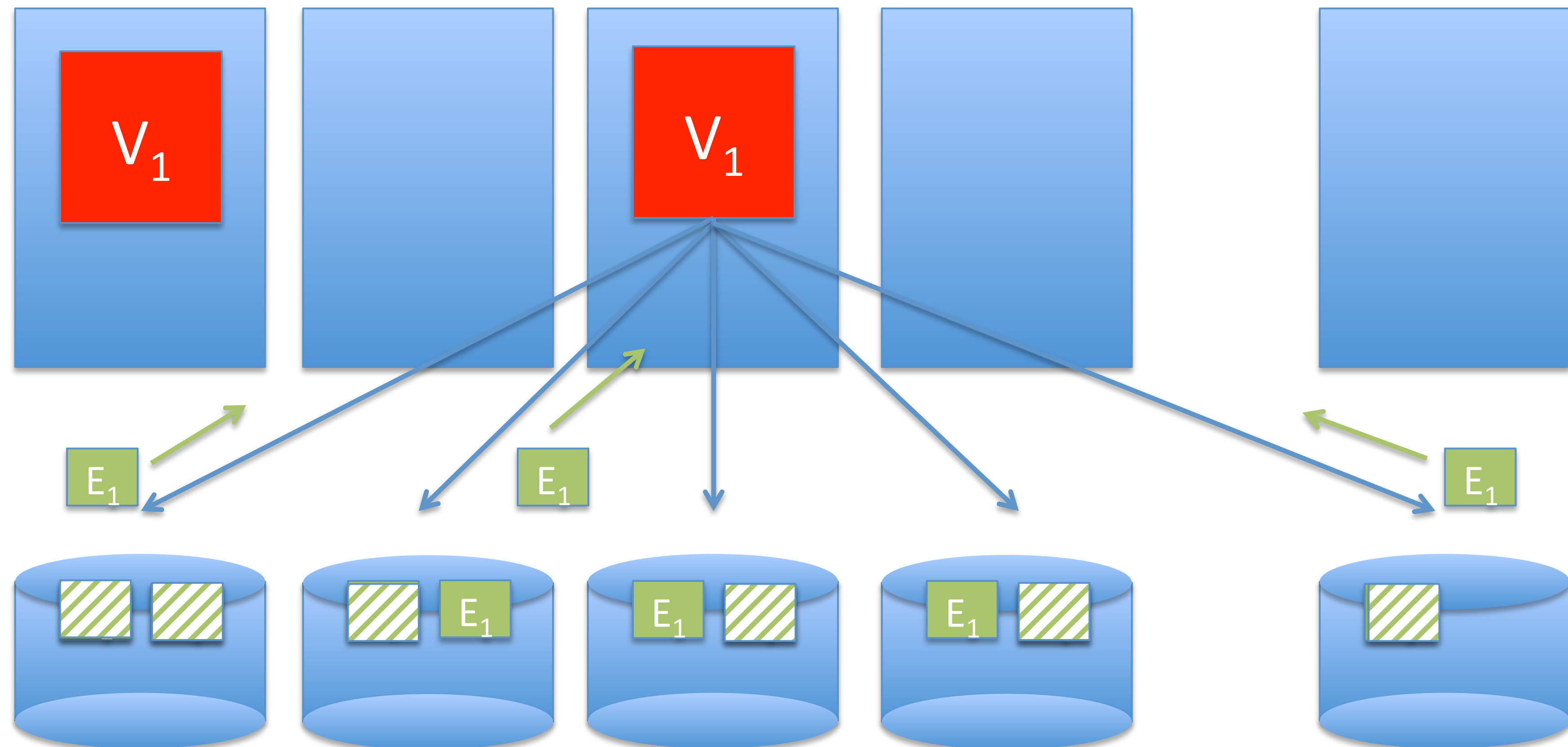
Work stealing issue?

- > 1 machines work on a streaming partition
- > 1 machines access same edge list
- Need for synchronization?

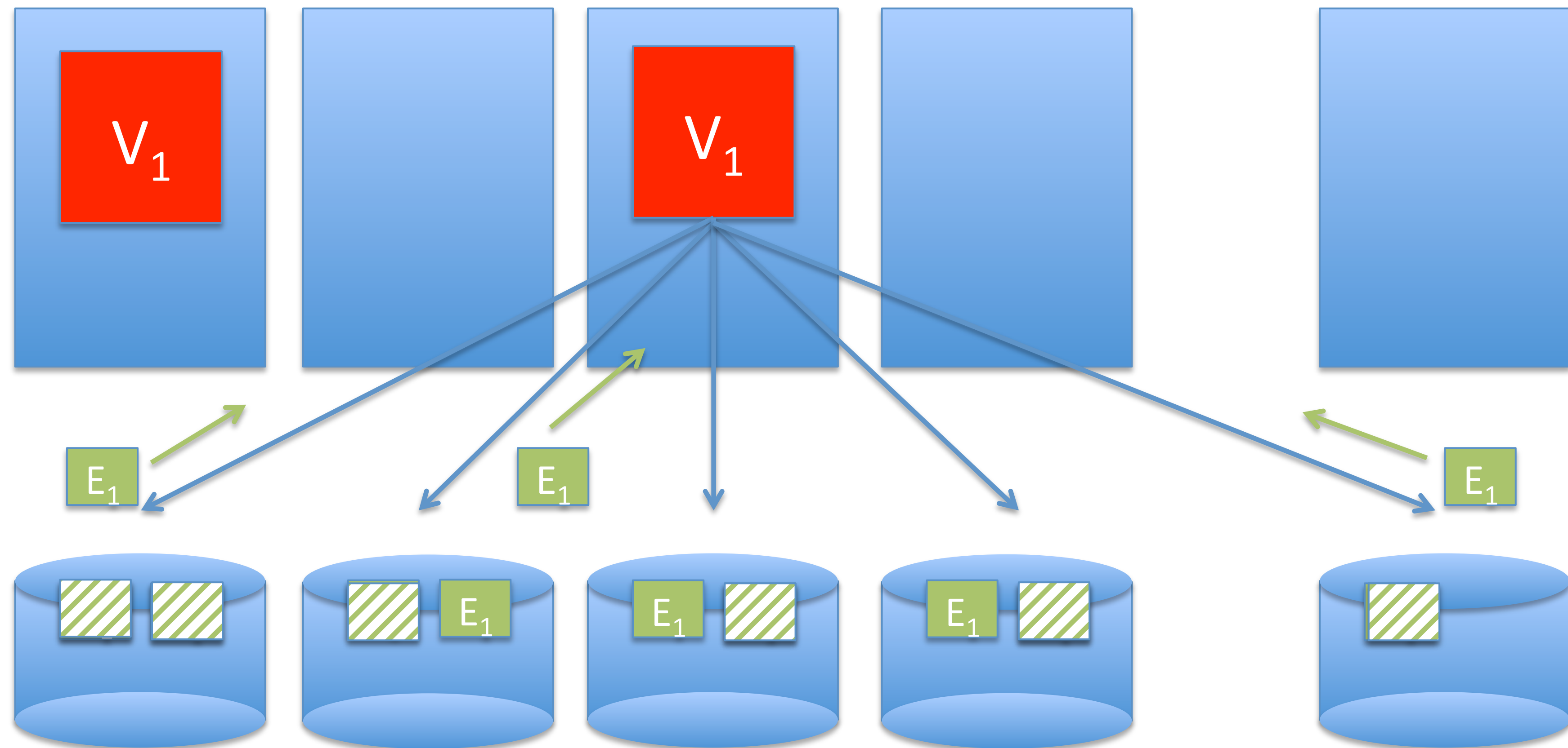
Stealing: Which edge stripe to read?



Stealing: It can read any stripe (that has not been read)



Remember



Storage side maintains what has and has not been read

Chaos: computation design: summary

- Work stealing
  - Without synchronization
  - Without centralized entity

A brief digression

- During gather (with work stealing):
  - Multiple machines update vertex state
- Each updates its own copy
- Copies are reconciled by Apply() function

- Similar to PowerGraph GAS model

Chaos: design summary

- Striping ➔ good I/O balance
- Work stealing ➔ good computational balance
- Streaming partition ➔ sequentiality

- And all of this
  - without expensive partitioning
  - without I/O synchronization

Evaluation

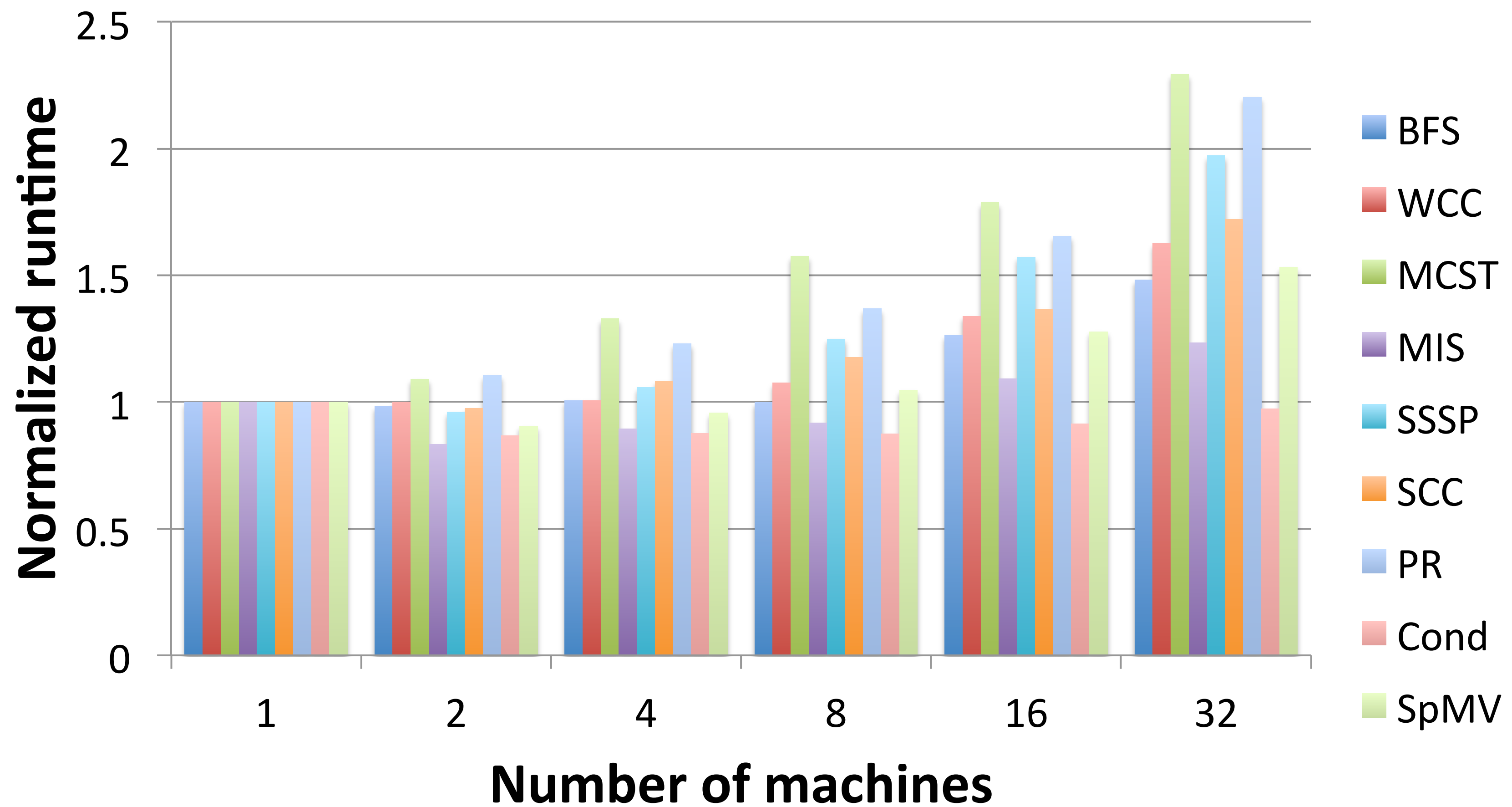
- 32 16-core machines (single rack)
- 32Gb RAM, 480Gb SSD, 2x6Tb HDD
- Full-bisection bandwidth 40GigE switch

- RMAT graphs
- Wall clock time (including pre-processing)

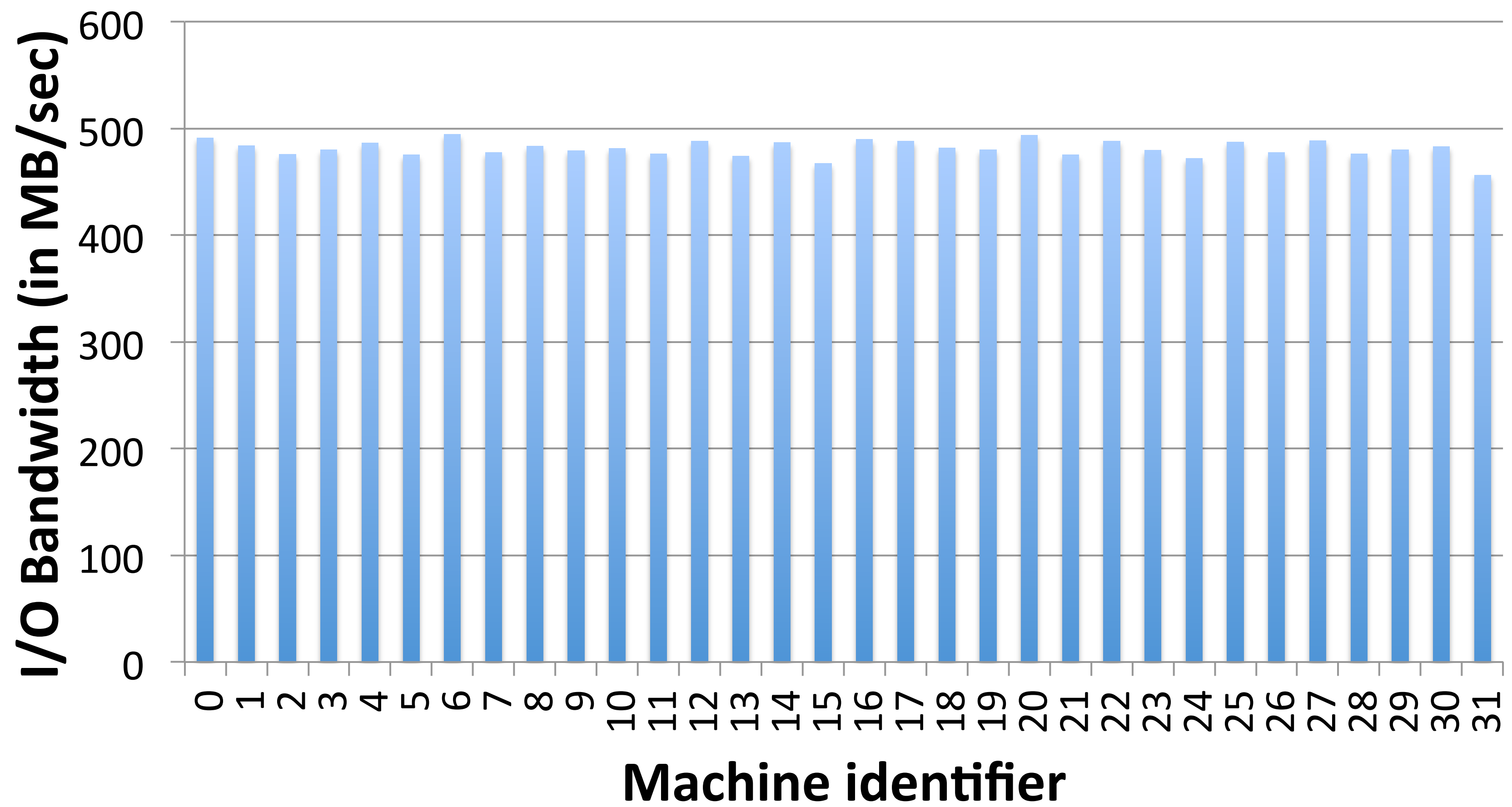
Weak scaling experiment

- For n machines
  - Use graph size n times for single machine
- Measure running time
  - For a number of algorithms
  - Normalize to running time to single machine
  - Ideally result would always be ~ 1

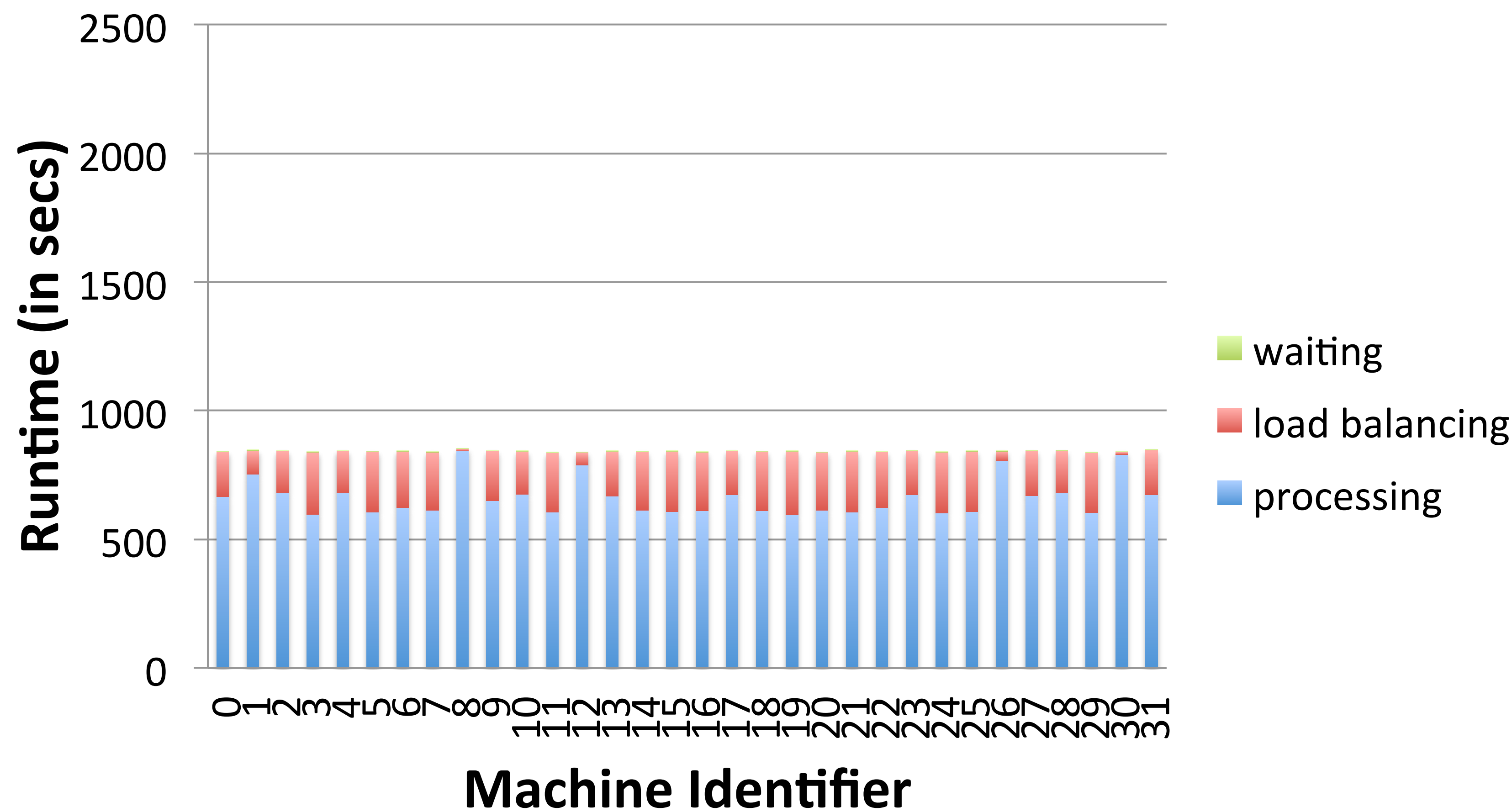
Weak scaling result



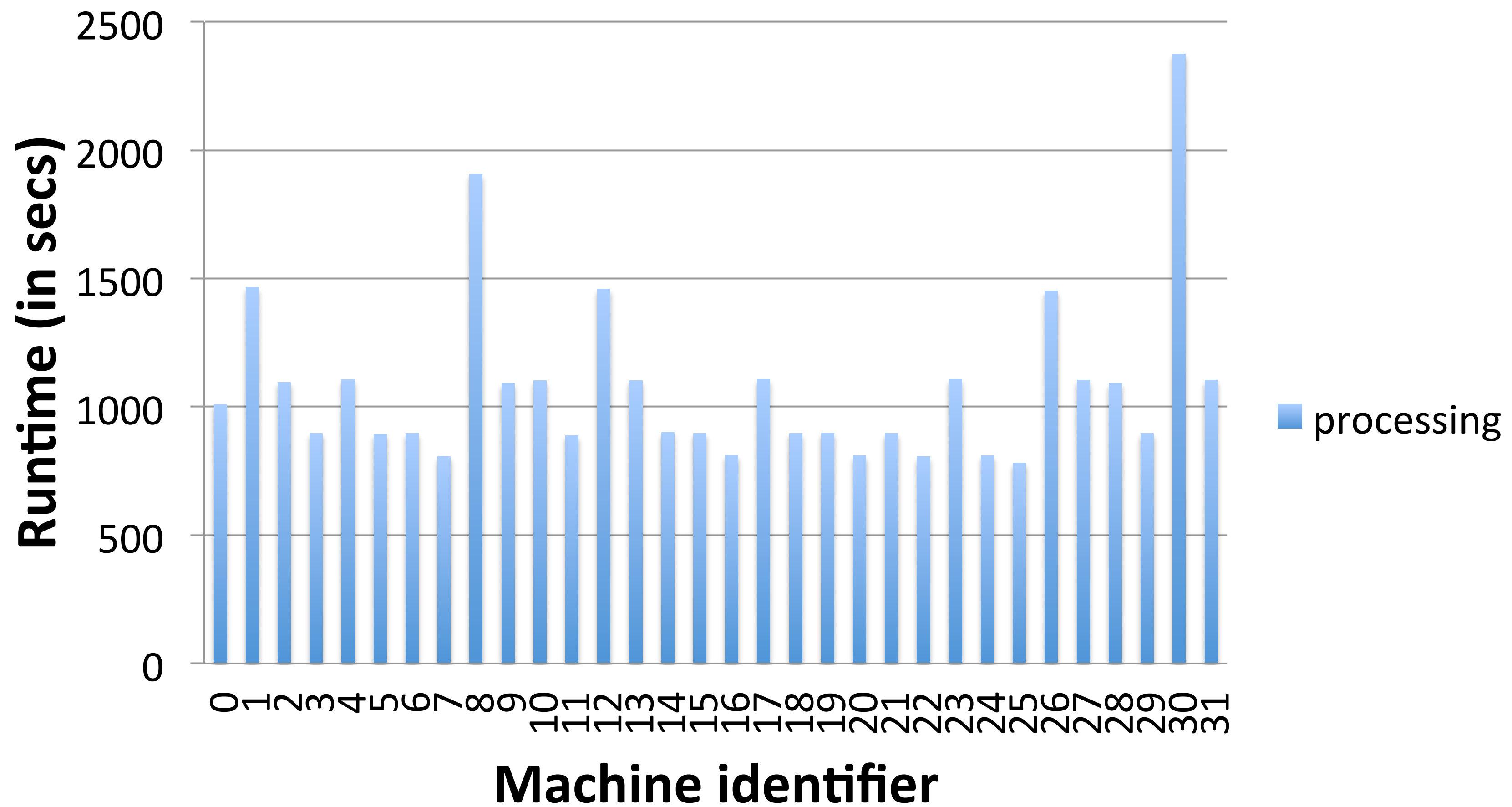
I/O Balance



Computational balance



Without work stealing



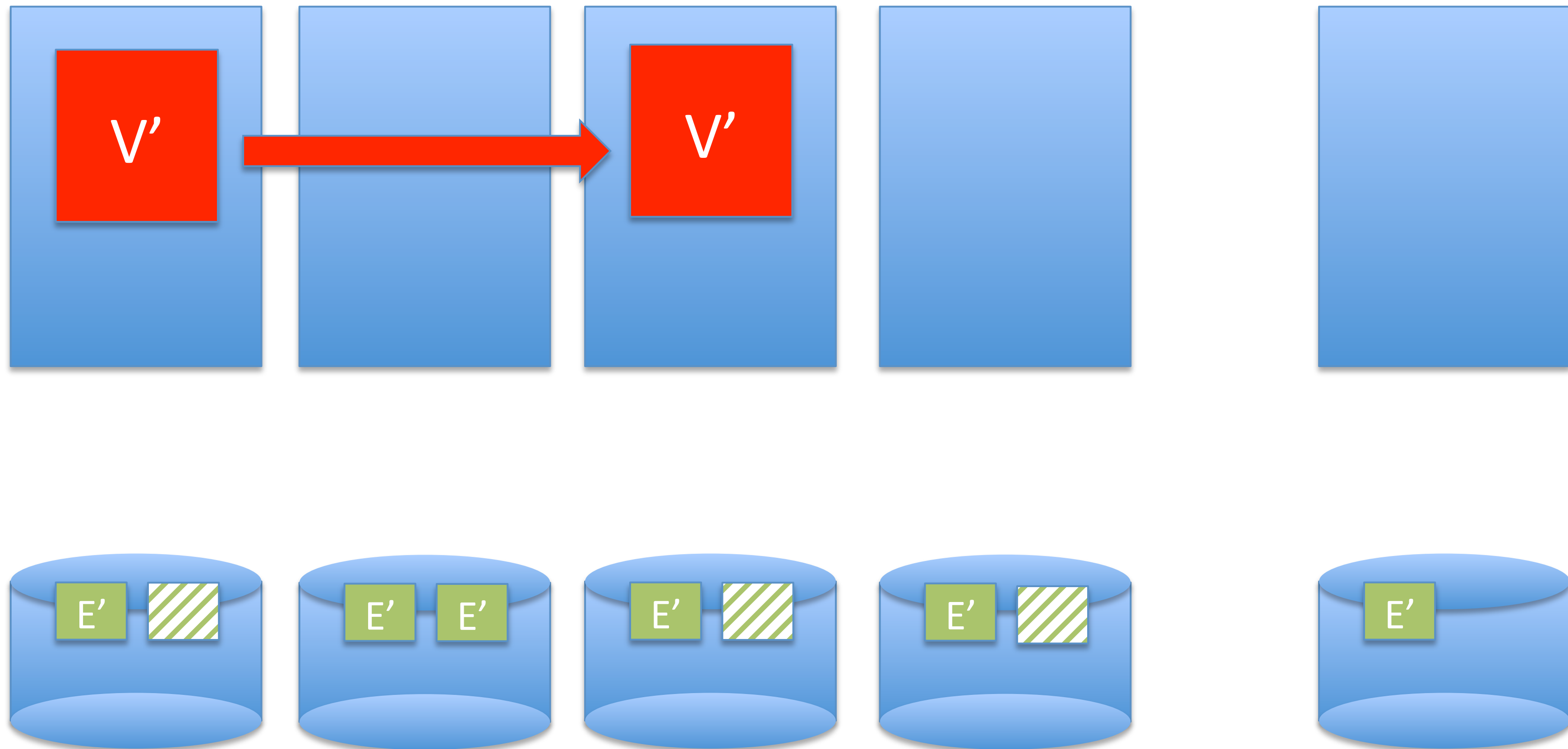
Why is scaling not perfect?

- Remote bandwidth ~ but < local bandwidth
- Load balance is not perfect
- Dynamic load balance has overhead
- Storage access less sequential

Why is scaling not perfect?

- Remote bandwidth ~ but < local bandwidth
- Load balance is not perfect
- *Dynamic load balance has overhead*
- Storage access less sequential

Stealing: Copy vertex set



I promised you:

Analytics on Graphs with *Trillions* of Edges

Laurent Bindschaedler, Jasmina Malicevic, Amitabha Roy, and Willy Zwaenepoel

I promised you:

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Benchmark

- Graph analytics benchmark
- Two rankings:
  - Speed
  - Capacity

Capacity Ranking

Rank	Size	Machine	Location	Nodes
1	32T	BlueGene/Q	Lawrence Livermore	98304
2	16T	BlueGene/Q	Argonne	49152
3	16T	Cray CS300	Lawrence Livermore	300
4	16T	K (Fujitsu Custom)	RIKEN AICS	82944
5	4T	PRIMEHPC FX10	University of Tokyo	4800
6	4T	BlueGene/Q	FZJ	16384
7	4T	PRIMEHPC FX10	University of Tokyo	4800
8	2T	T-Platforms - MPP	Moscow University	4096
9	2T	BlueGene/P	FZJ	16000
10	2T	T-Platforms - MPP	Moscow University	4096

Capacity Ranking

Rank	Size	Machine	Location	Nodes
1	32T	BlueGene/Q	Lawrence Livermore	98304
2	16T	BlueGene/Q	Argonne	49152
3	16T	Cray CS300	Lawrence Livermore	300
4	16T	K (Fujitsu Custom)	RIKEN AICS	82944
5	8T	Xeon E5	EPFL	32
6	4T	PRIMEHPC FX10	University of Tokyo	4800
7	4T	BlueGene/Q	FZJ	16384
8	4T	PRIMEHPC FX10	University of Tokyo	4800
9	2T	T-Platforms - MPP	Moscow University	4096
10	2T	BlueGene/P	FZJ	16000

Capacity Ranking

Rank	Size	Machine	Location	Nodes
1	32T	BlueGene/Q	Lawrence Livermore	98304
2	16T	BlueGene/Q	Argonne	49152
3	16T	Cray CS300	Lawrence Livermore	300
4	16T	K (Fujitsu Custom)	RIKEN AICS	82944
5	8T	Xeon E5	EPFL	32
6	4T	PRIMEHPC FX10	University of Tokyo	4800
7	4T	BlueGene/Q	FZJ	16384
8	4T	PRIMEHPC FX10	University of Tokyo	4800
9	2T	T-Platforms - MPP	Moscow University	4096
10	2T	BlueGene/P	FZJ	16000

Capacity Ranking

Rank	Size	Machine	Location	Nodes
1	32T	BlueGene/Q	Lawrence Livermore	98304
2	16T	BlueGene/Q	Argonne	49152
3	16T	Cray CS300	Lawrence Livermore	300
4	16T	K (Fujitsu Custom)	RIKEN AICS	82944
5	8T	Xeon E5	EPFL	32
6	4T	PRIMEHPC FX10	University of Tokyo	4800
7	4T	Bl… ene/Q	FZJ	16384
8	4T	PRIM…	…ity of Tokyo	4800
9	2T	T-Pla…	…University	4096
10	2T	BlueGe…	FZJ	16000

Input: **128TB**
I/O: **1.8PB**

Conclusion

- The "IKEA" approach to graph processing works
- Based on processing from secondary storage

- X-Stream:
  - Edge-centric processing
  - Streaming partitions

- Chaos:
  - Flat storage
  - Work stealing
  - Randomization

Further information

- Two publications:
  - A. Roy, I. Mihailovic and W. Zwaenepoel, X-Stream: Edge-centric Graph Progessing using Streaming Partitions, SOSP 2013
  - A. Roy, L. Bindschaedler, J. Malicevic and W. Zwaenepoel, Chaos: Scale-Out Graph Processing from Secondary Storage, SOSP 2015
- https://github.com/labos-epfl/chaos
- http://labos.epfl.ch