LOAD BALANCING TECHNIQUES FOR CHAOS

Vlad Ioan Haprian

Laurent Bindschaedler

Willy Zwaenepoel

OUTLINE

1. ORIGINAL CHAOS WITH WORK STEALING

   a) general presentation

   b) possible improvements

2. DIFFERENT VERTEX SET SIZE PARTITIONS.

3. SAME EDGE SET SIZE PARTITIONS.

   a) implementation.

   b) analysis.

   c) results.

4. VERTEX RELABELING.

5. GRID PARTITIONING.

CHAOS

Scale-out Graph Processing from Secondary Storage using small clusters (speed v.s. cost trade-off)

A few machines

+

Secondary storage

CHAOS IDEAS

1. **Exploit sequentiality** =>

- a) Vertices in main memory for random access

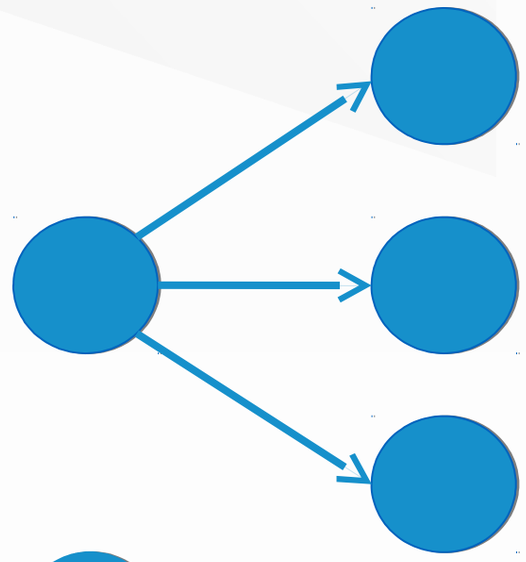
b) Edges in secondary storage for sequential access

=> Edge-centric graph processing

2. **Minimize preprocessing time** => partitioning phase is very simple.

Edge-centric Graph Processing

- Store state in vertices



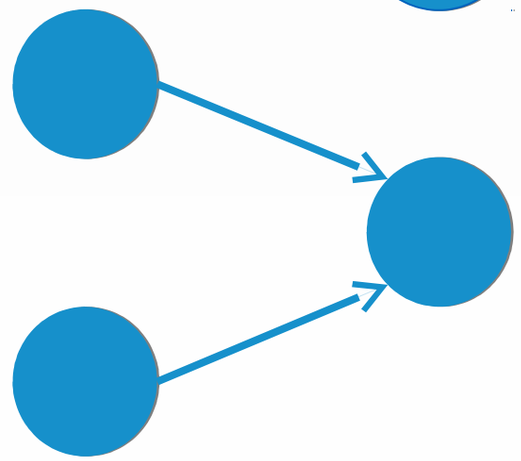
- **Scatter** – For all outgoing edges:

  `new update = f(vertex state)`

- **Gather** – For all incoming edges:

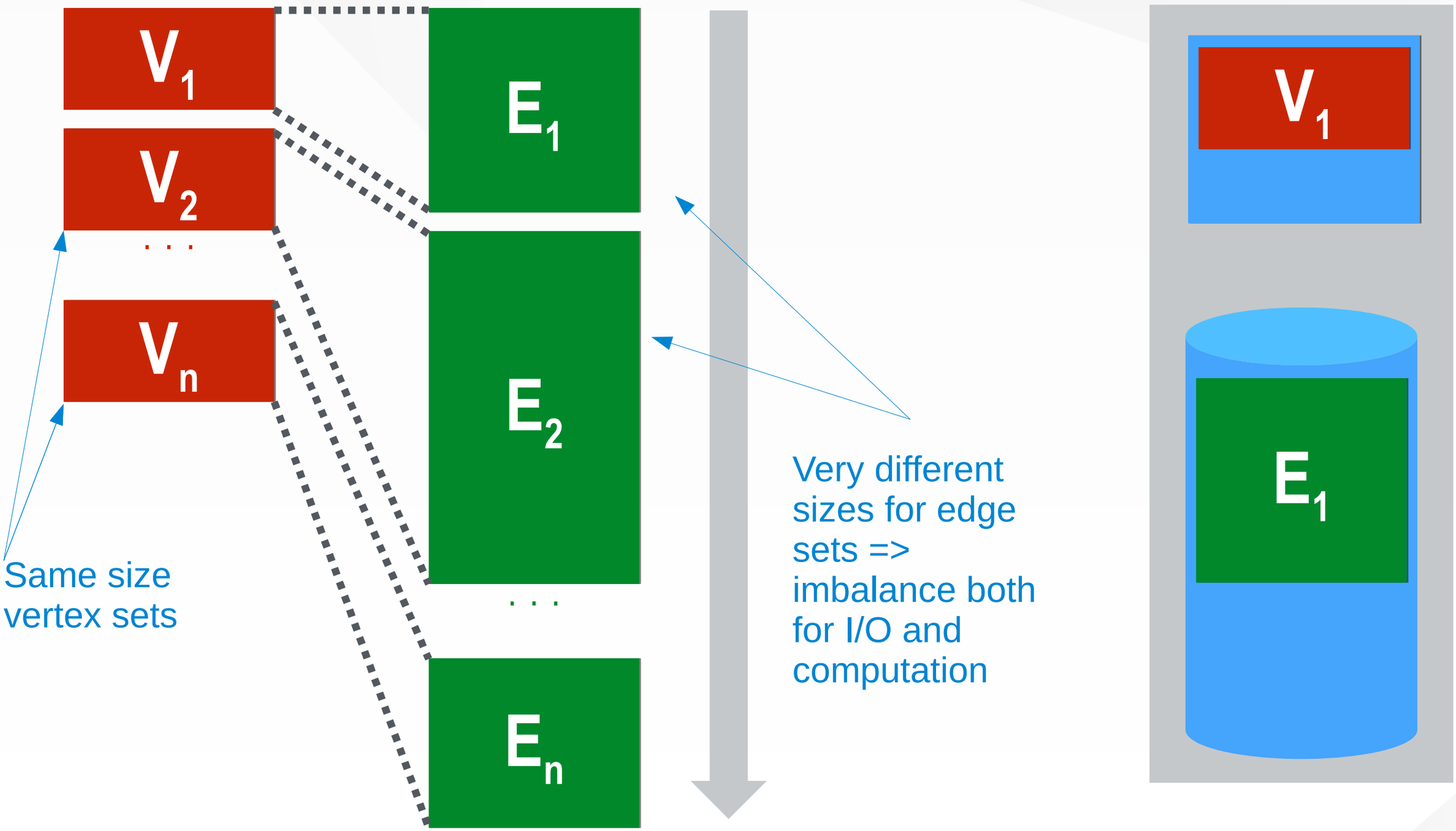
  `vertex value =`

  `g(vertex value, update)`

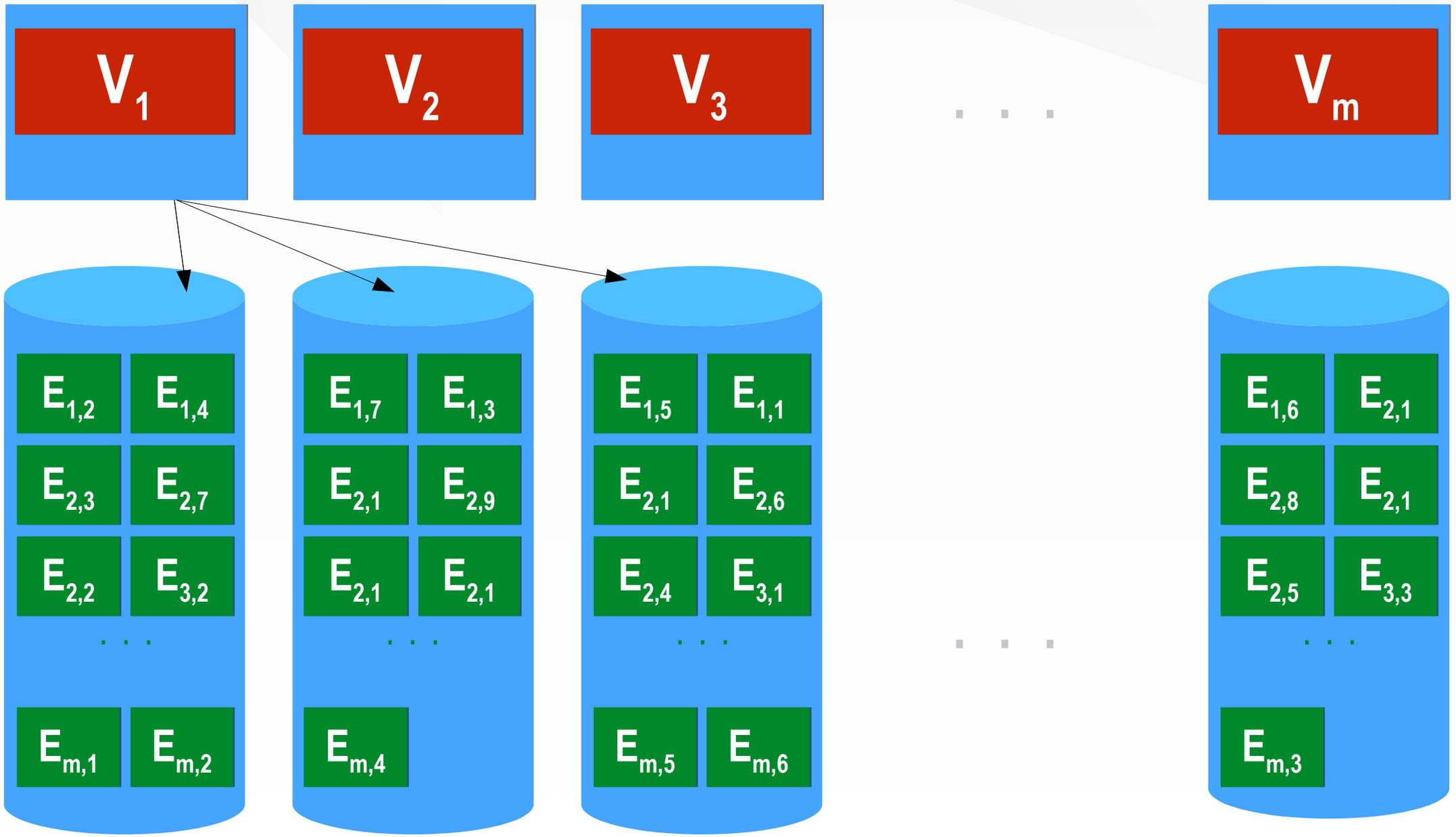


`Order independent!`

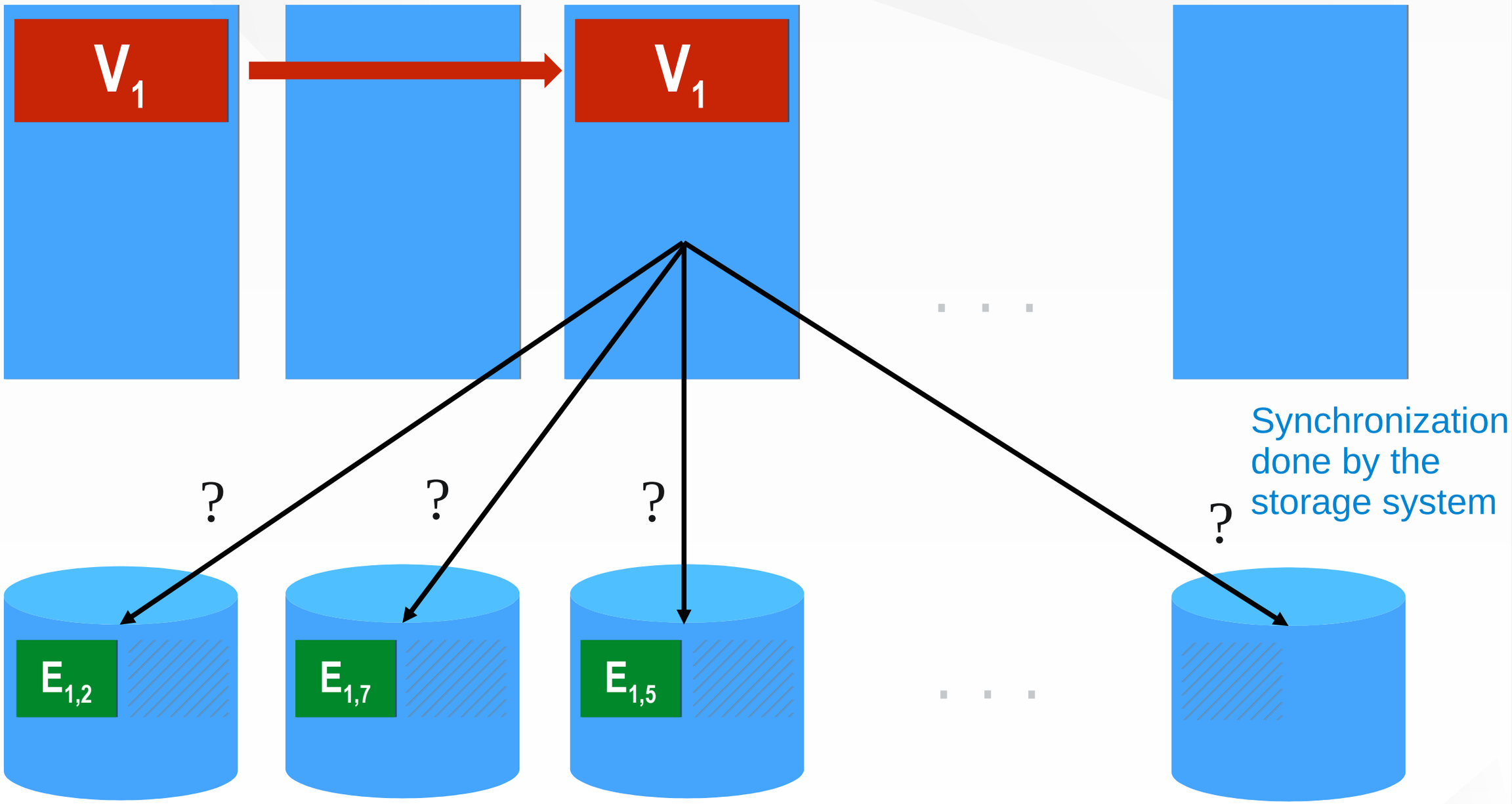
Very Simple Partitioning Phase => Computation and I/O are Unbalanced



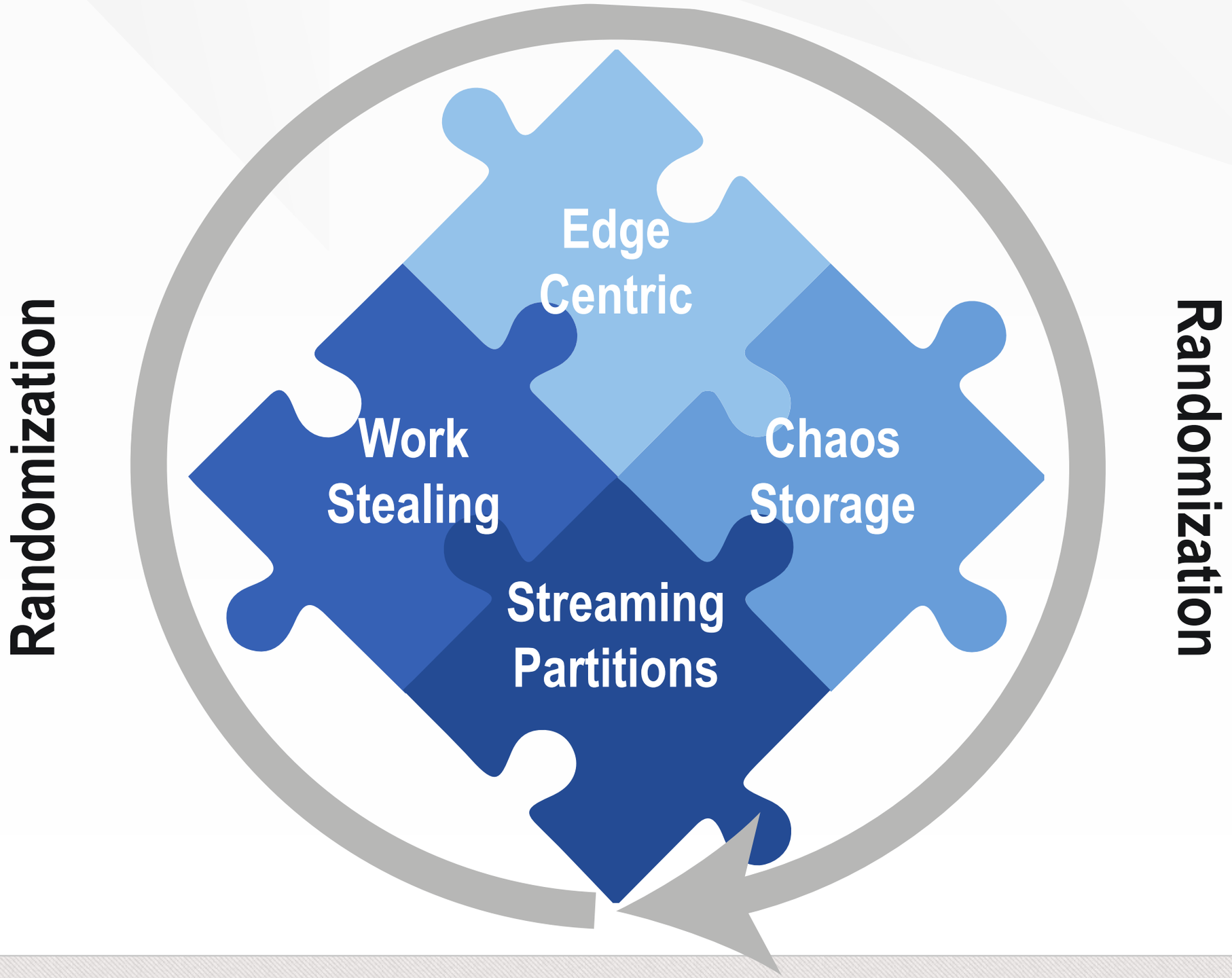
Remote bandwidth ~ local bandwidth **=> edges are striped and batch I/O => I/O balance**



Do **work stealing** in order to achieve **computational balance**



Recipe for Chaos



OUTLINE

1. ORIGINAL CHAOS WITH WORK STEALING

    a) general presentation

    b) possible improvements

2. DIFFERENT VERTEX SET SIZE PARTITIONS.

3. SAME EDGE SET SIZE PARTITIONS.

    a) implementation.

    b) analysis.

    c) results.

4. VERTEX RELABELING.

5. GRID PARTITIONING.

Work Stealing is not free !

The stealer does additional I/O read cost in order to copy the vertex state !

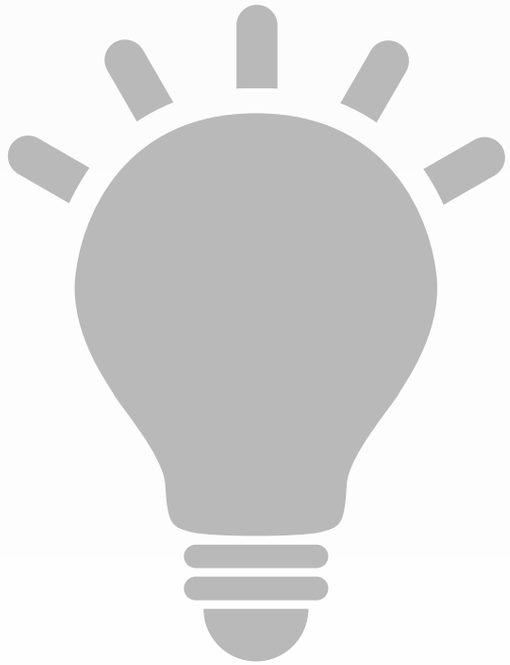
→ 16% of the scatter time is spent on copying vertex state.

→ 30 % of the gather time is spent on copying the vertex state (16%) and merging the updates (14%).

(only when lots of updates, thus need for work stealing).

Possible Solutions

1. Find optimal size of vertex-sets: more partitions might balance the work load better.



2. Create balanced partitions: increase preprocessing costs.

OUTLINE

1. ORIGINAL CHAOS WITH WORK STEALING

   a) general presentation

   b) possible improvements

2. DIFFERENT VERTEX SET SIZE PARTITIONS.

3. SAME EDGE SET SIZE PARTITIONS.

   a) implementation.

   b) analysis.

   c) results.

4. VERTEX RELABELING.

5. GRID PARTITIONING.

Stealing vs **Streaming** friendly

Trade-off :

→ smaller vertex sets =>

more partitions per machine =>

partitions are more balanced =>

work stealing is less needed and cheaper.

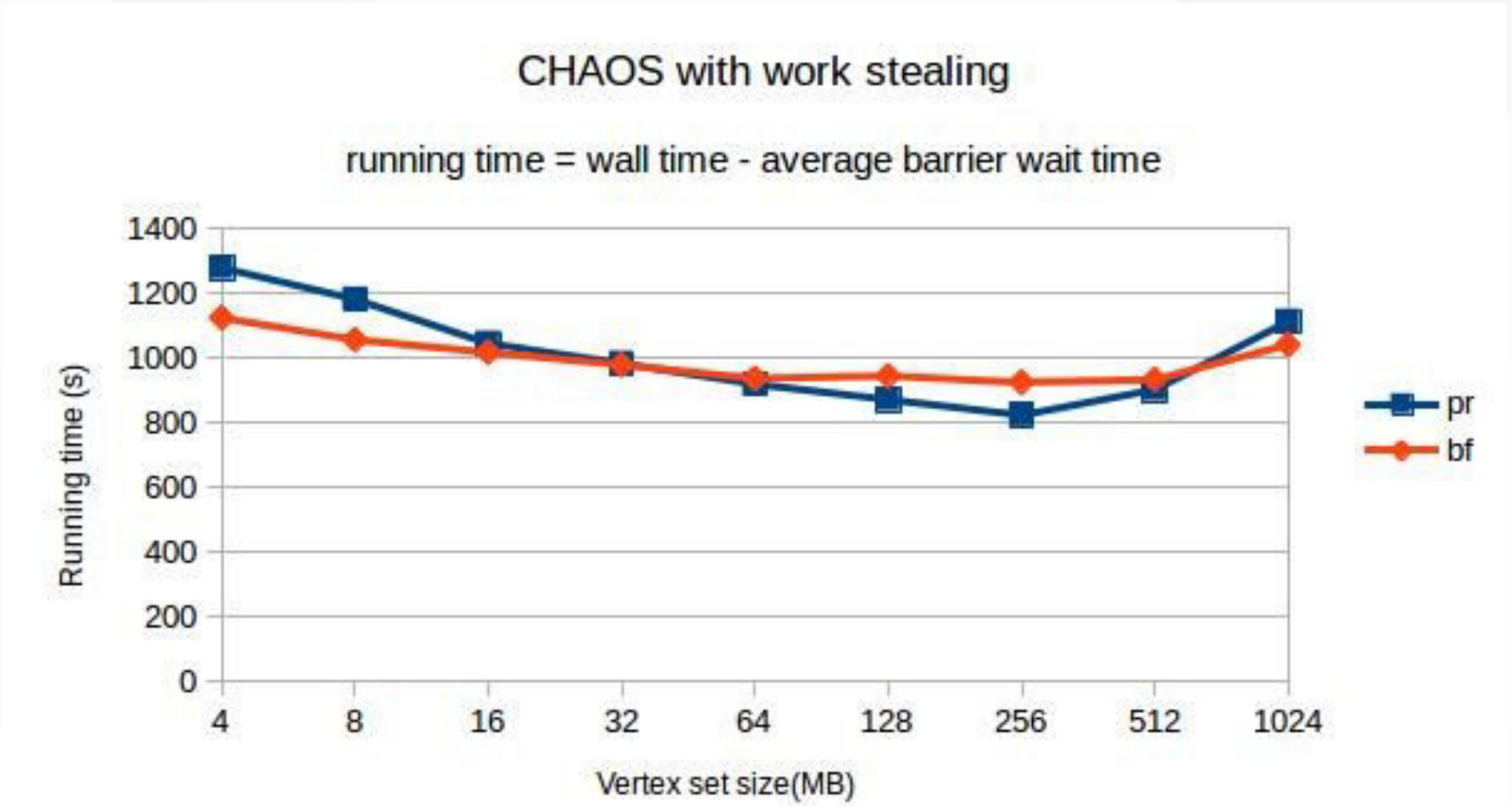
→ small vertex sets =>

smaller edge sets =>

sequentiality is lost

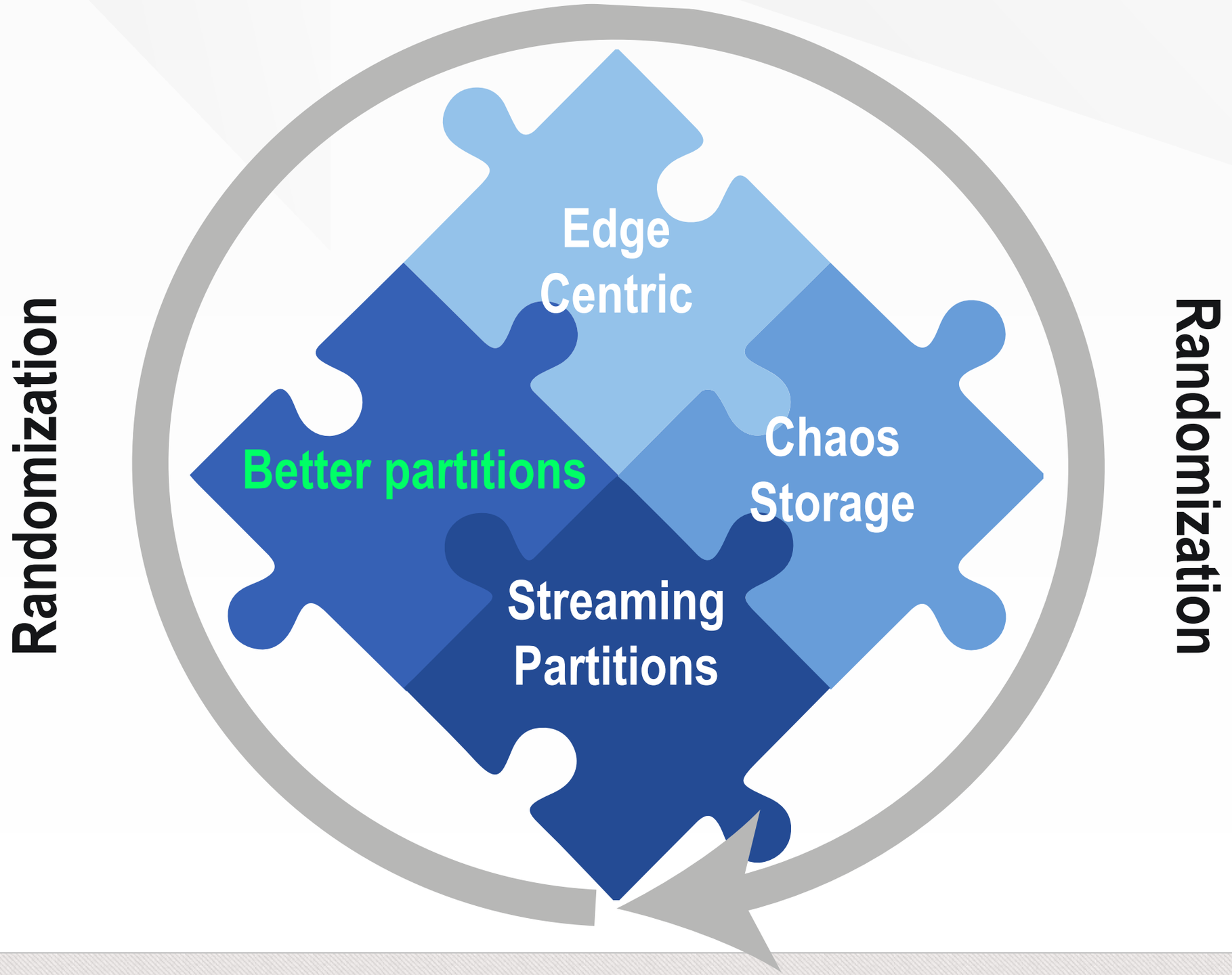
Varying Vertex Set Size Experiment

RMAT-28 (2GB of vertices) on a cluster of 8 machines with 2GB of RAM per machine.



Optimal value corresponds to the minimum number of partitions, such that the vertex set fits in memory and each node has at least one partition.

Recipe for new Chaos



OUTLINE

1. ORIGINAL CHAOS WITH WORK STEALING

   a) general presentation

   b) possible improvements

2. DIFFERENT VERTEX SET SIZE PARTITIONS.

3. SAME EDGE SET SIZE PARTITIONS.

   a) implementation.

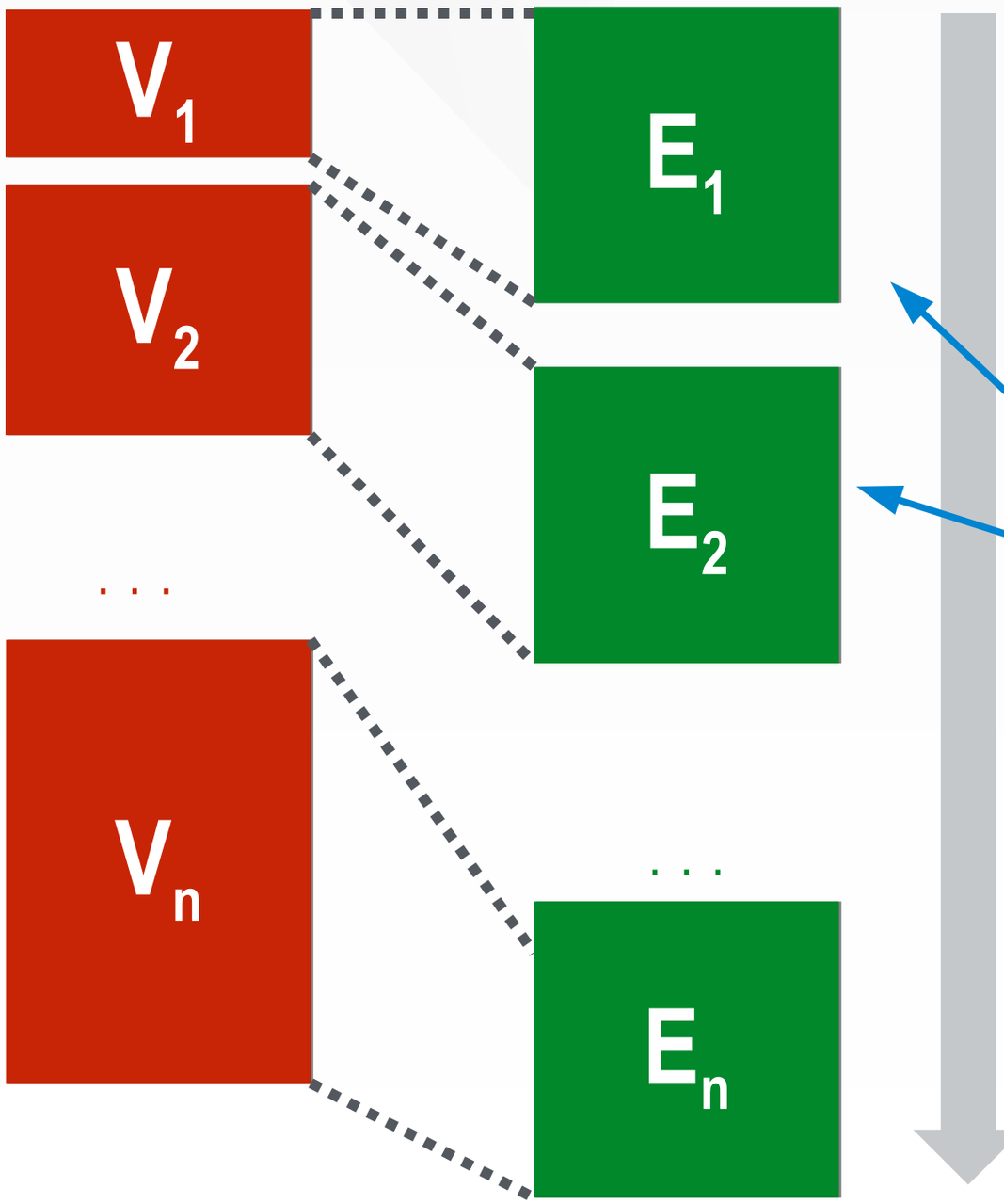
   b) analysis

   c) results.

4. VERTEX RELABELING.

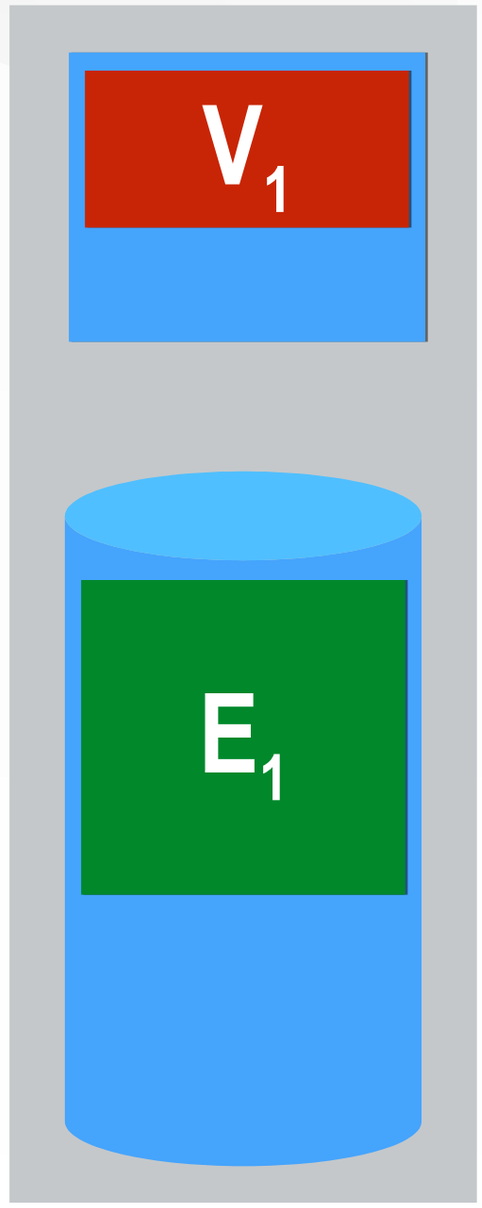
5. GRID PARTITIONING.

Better Partitioning => No Need for Work Stealing



Same size edge sets balance the computation

Processing model is the same



How the New Partitions Are Generated ?

First pass

Second pass

E

Sum of out degrees of vertices in a partition should be the same

Out degree for each vertex

Max. number of edges we can have in a partition is equal to the max. number of vertices we can fit in memory

Same size edge sets partitions.

Prevents memory overflow.

New partitions offsets file

E_1

E_2

. . .

E_n

OUTLINE

1. ORIGINAL CHAOS WITH WORK STEALING

   a) general presentation

   b) possible improvements

2. DIFFERENT VERTEX SET SIZE PARTITIONS.

3. SAME EDGE SET SIZE PARTITIONS.

   a) implementation.

   b) analysis → drawbacks, optimizations, locality

   c) results.

4. VERTEX RELABELING.

5. GRID PARTITIONING.

How good we are ?

1. How much we manage to reduce the imbalance ?

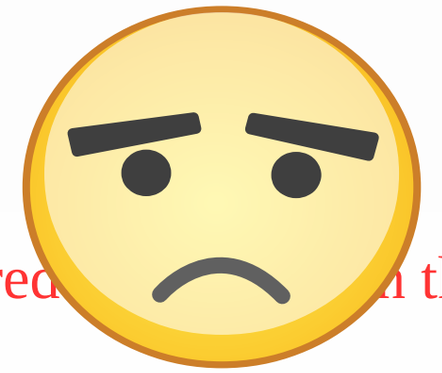
Scatter barrier time is ~4.5 x less than in the original Chaos without work stealing.

2. Are we better than the goal to beat version ? No. 30% slower

3. Are we achieving perfect balance:

a) I/O balance: not really

b) Computational balance: almost



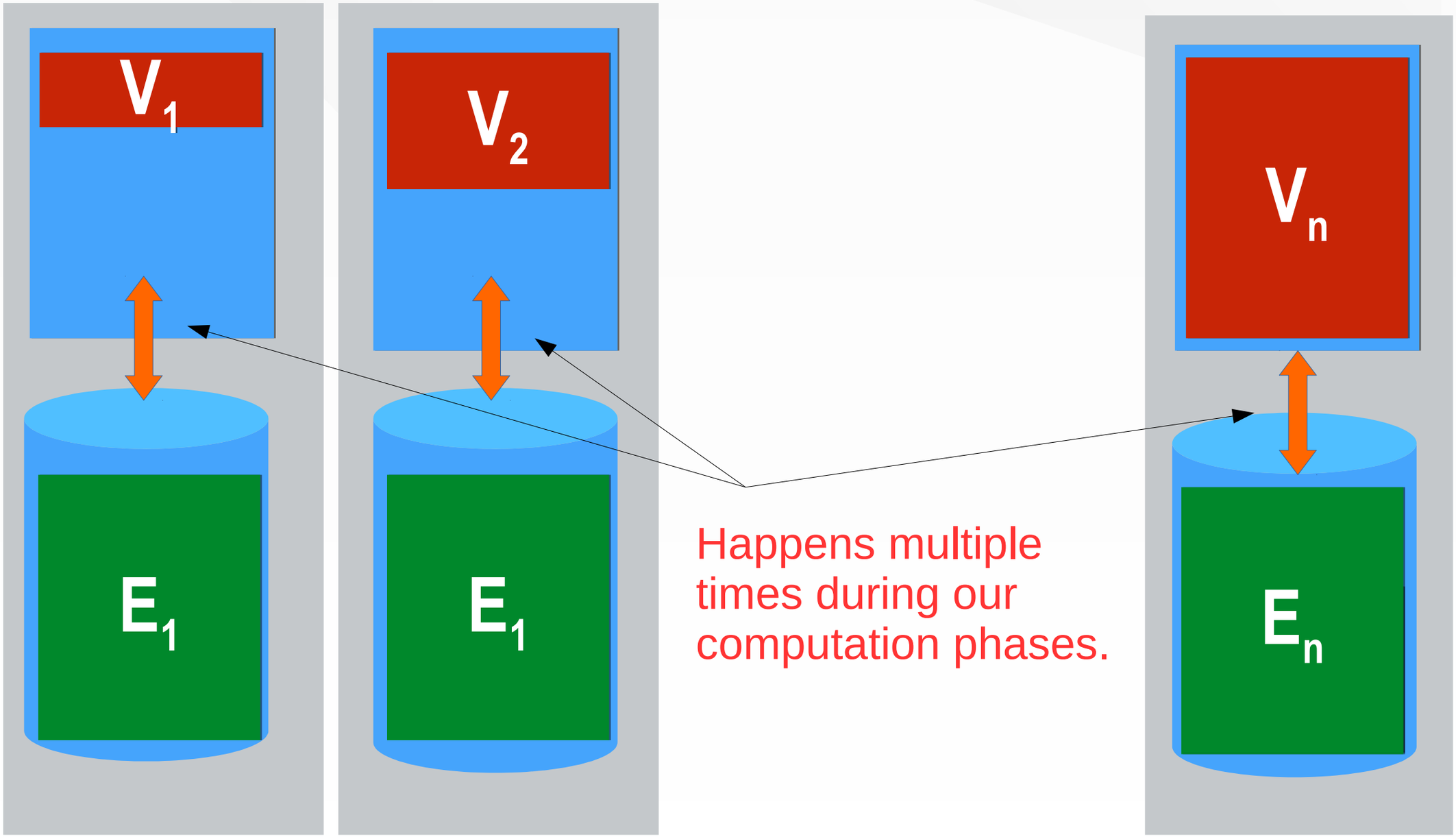
Scatter barrier time is still big. ~10% of the wall time compared [illegible] the old version of Chaos with work stealing.

What is Wrong?

→ I/O is still not balanced.

→ Computation is slower and not balanced.

I/O is not Balanced



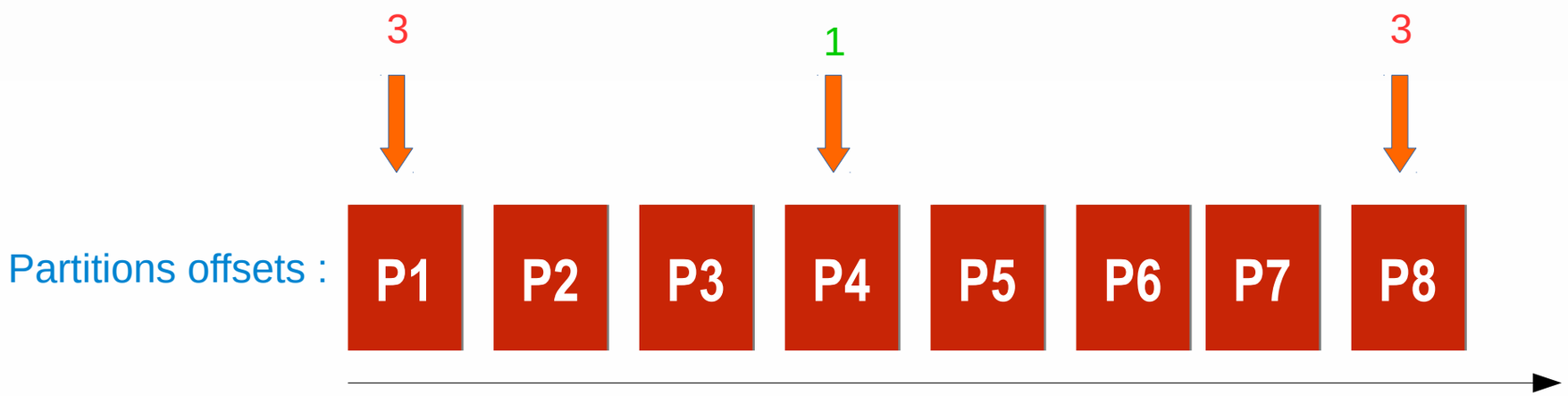
Happens multiple times during our computation phases.

Computation is Slower and Not Balanced

→ **Slower** : array access instead of bit shift for partition lookup.

- 
  this done O(|V| * #phases) times

→ **Not Balanced**: due to binary searching



OUTLINE

1. ORIGINAL CHAOS WITH WORK STEALING

   a) general presentation

   b) possible improvements

2. DIFFERENT VERTEX SET SIZE PARTITIONS.

3. SAME EDGE SET SIZE PARTITIONS.

   a) implementation.

   b) analysis → drawbacks, optimizations, locality

   c) results.

4. VERTEX RELABELING.

5. GRID PARTITIONING.

Reducing the I/O Imbalance

We use a slightly changed partition constraint :

Partition for c * vertex set + edge set same size partitions !

Choice of c is important:

→ c = 1: balances I/O for scatter.

→ c > 1: makes vertex sets to be less different !

→ c too big: brings us to the original Chaos.

Optimal value: c = 3.

Reducing the Computation Time

The main overhead in computation is partition search => use caching !

1) We compute the partition only once per scatter/gather phase.

2) During scatter the updates are written by threads to buffers corresponding to the update destination partition

=> each thread caches the computed partition and it recomputes it only when it becomes obsolete.

=> 20 to 25 % improved running time

OUTLINE

1. ORIGINAL CHAOS WITH WORK STEALING

   a) general presentation

   b) possible improvements

2. DIFFERENT VERTEX SET SIZE PARTITIONS.

3. SAME EDGE SET SIZE PARTITIONS.

   a) implementation.

   b) analysis → drawbacks, optimizations, locality

   c) results.

4. VERTEX RELABELING.

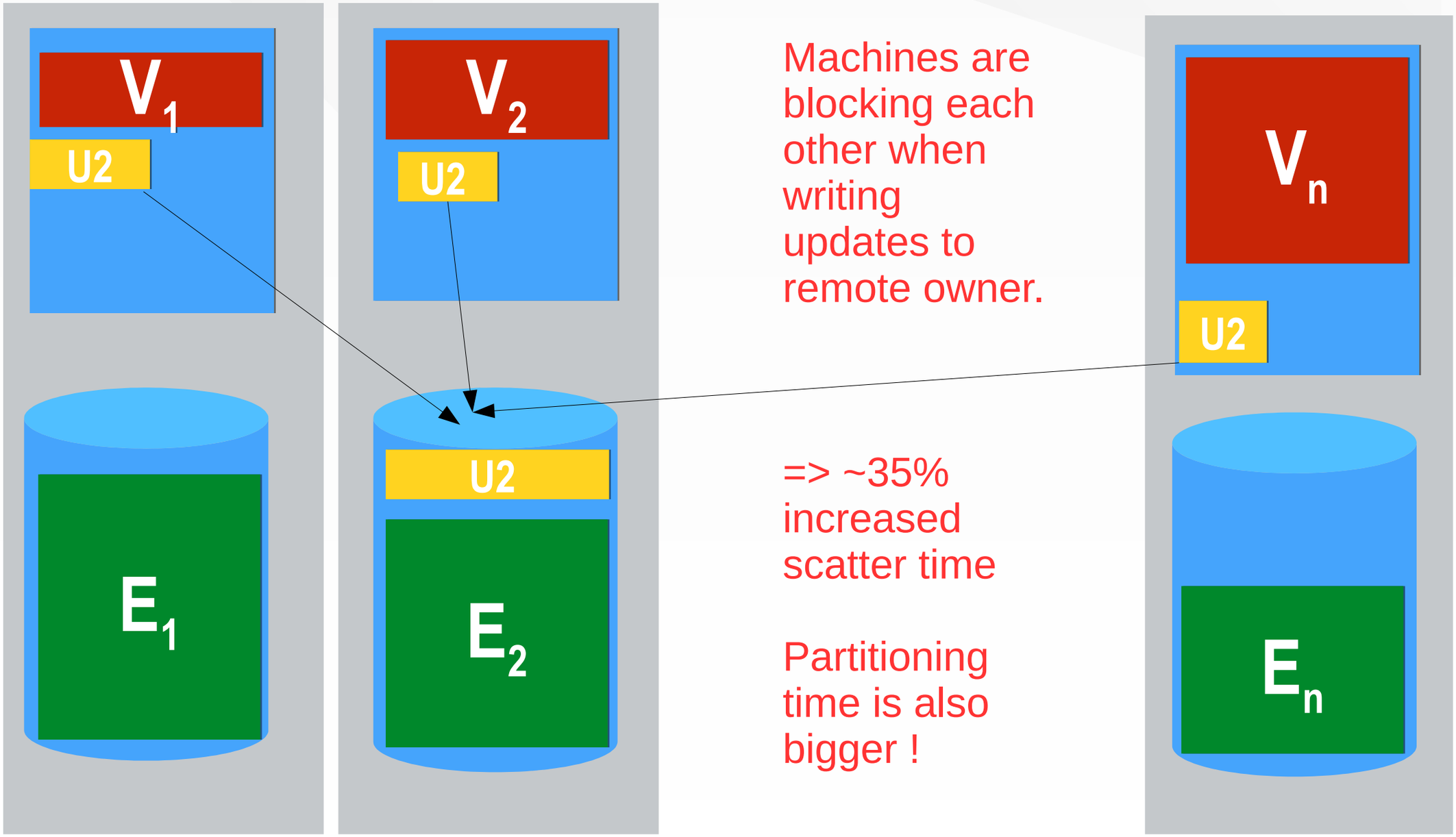
5. GRID PARTITIONING.

Locality

→ Load balance => not need for stripping ?

→ Problem of very different vertex sets.

Keep Everything on Local

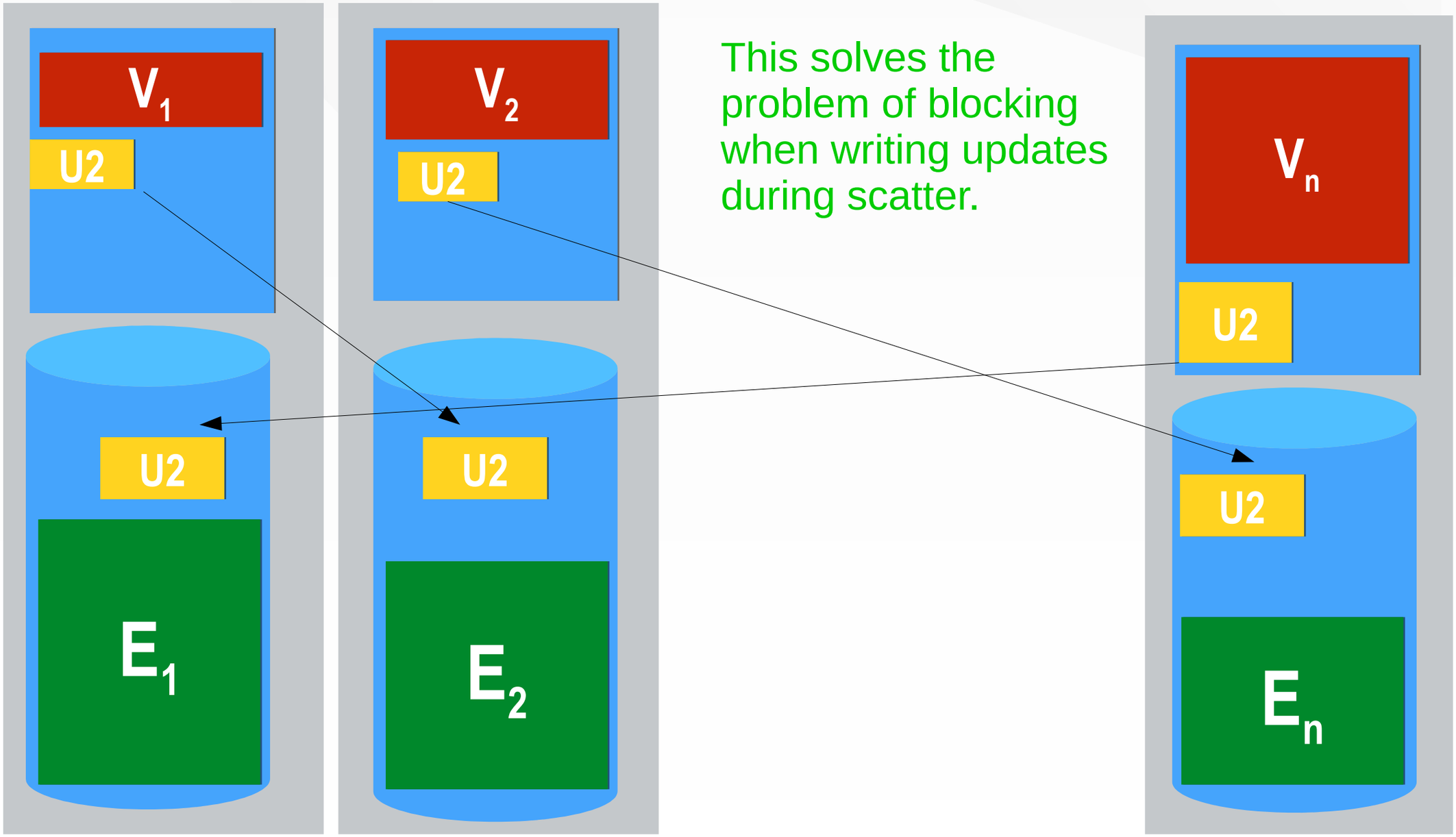


Machines are blocking each other when writing updates to remote owner.

=> ~35% increased scatter time

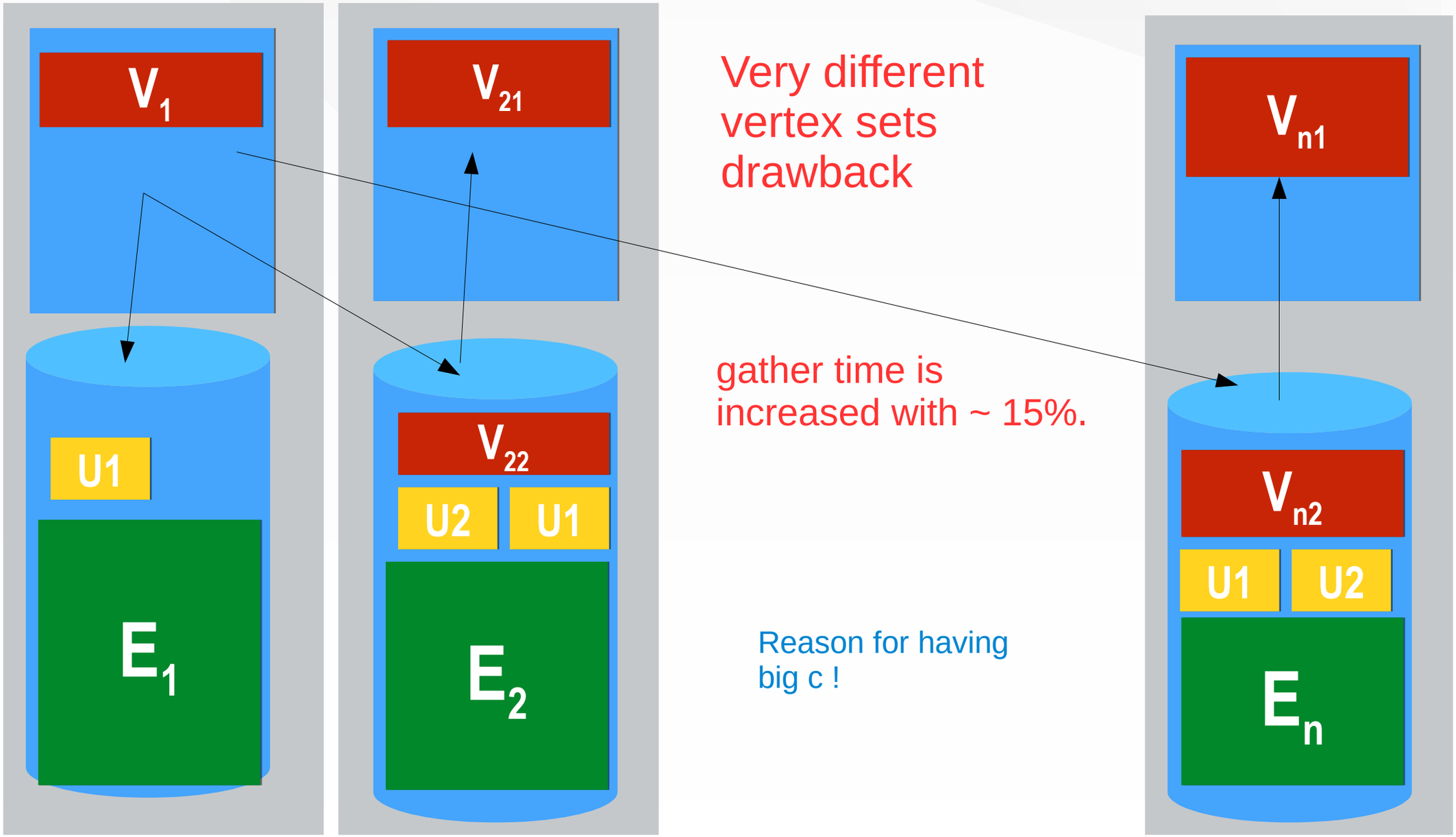
Partitioning time is also bigger !

Updates Striped, Edges and Vertices Local (1)



This solves the problem of blocking when writing updates during scatter.

Updates Striped, Edges and Vertices Local (2)



Very different vertex sets drawback

gather time is increased with ~ 15%.

Reason for having big c !

OUTLINE

1. ORIGINAL CHAOS WITH WORK STEALING

   a) general presentation

   b) possible improvements

2. DIFFERENT VERTEX SET SIZE PARTITIONS.

3. SAME EDGE SET SIZE PARTITIONS.

   a) implementation.

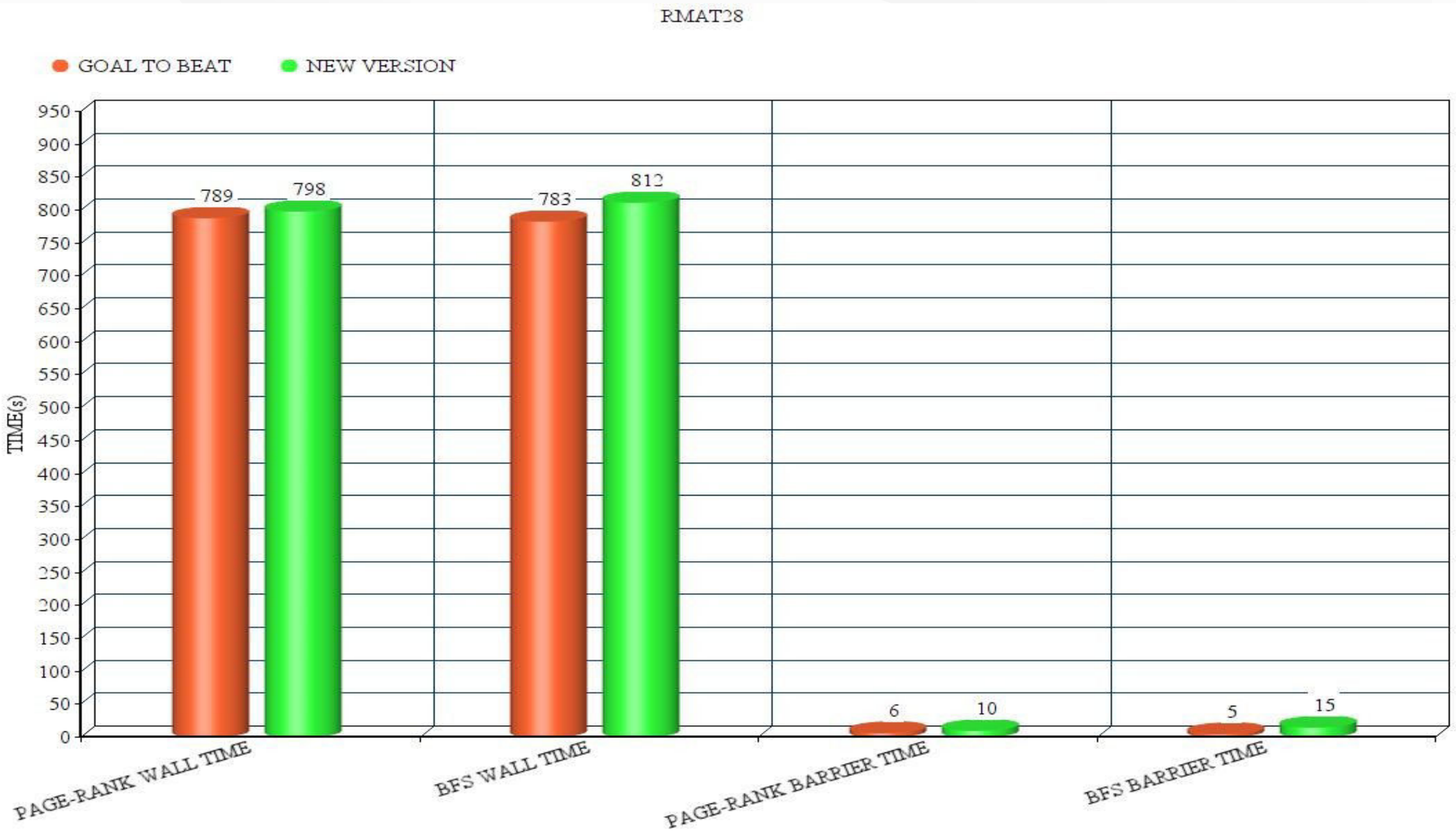
   b) analysis

   c) results.

4. VERTEX RELABELING.

5. GRID PARTITIONING.

Where We are Compared to the Goal to Beat ?



Where We are Compared to the Goal to Beat ?

Running time:

→ **the same for Page-rank (each node produces an update)**

→ **3.5% slower for BFS (harder to balance than for PR)**

Other reasons:

→ **overhead when computing partitions for updates during scatter (caching is not perfect).**

→ **different size vertex sets slow down the gather phase.**

Very good balance is achieved: barrier time is < 1 %

More Optimizations

In the case of one partition per machine we just need to load the vertex state once at the beginning of the algorithm and store it once at the end.

=> notice that we have to use the same size edge set partitions otherwise I/O is not balanced.

=> running time: 3 % better than the goal to beat for BFS

the same for Page-rank

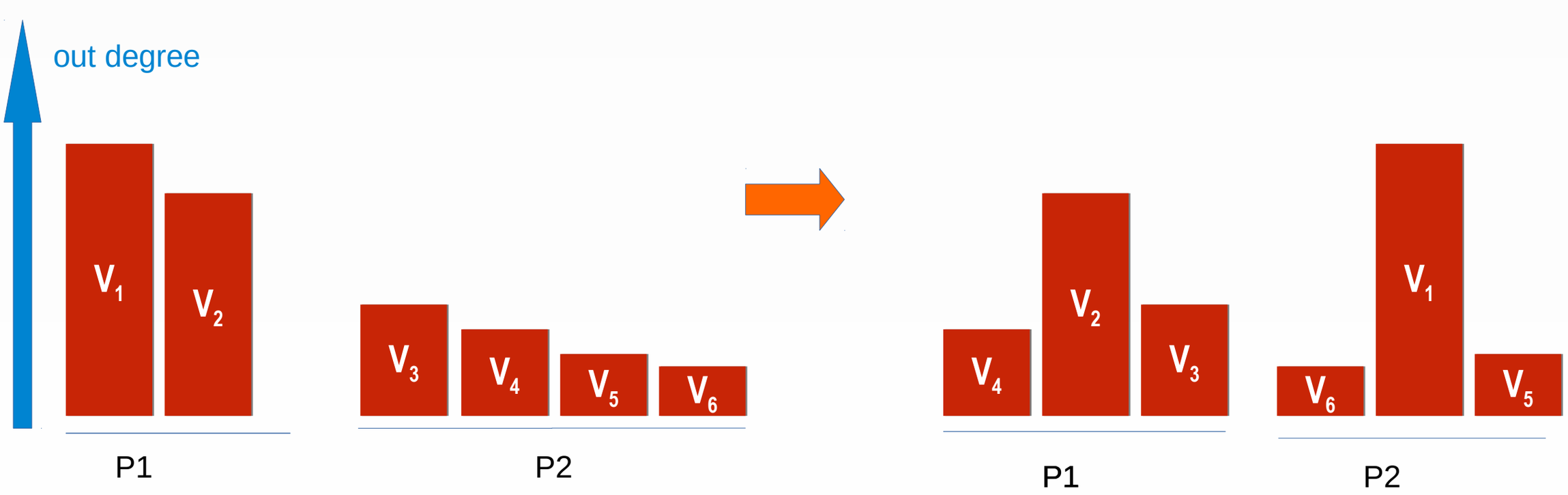
But might need one vertex relabeling phase in order to make this optimization possible without RAM overflow.

Guideline

1. Original CHAOS with work stealing
   a) general presentation
   b) possible improvements
2. Different vertex set size partitions.
3. Same edge set size partitions.
   a) implementation.
   b) drawbacks and optimizations.
   c) results.
4. Vertex relabeling.
5. Grid partitioning.

Vertex Relabeling

Power-law graphs (RMAT) have the vertex degree decreasing with vertex id => use random vertex relabeling.



Same size edge sets <=> Very different size vertex sets

Same size edge sets <=> 'balanced' size vertex sets

Discussion

Pros:

+ balanced vertex sets AND edge sets

+ bit shift for obtaining the partition

+ get rid of slow gather due to reading vertex sets

Cons:

- generating good random permutation is very expensive

- one more pass.

Guideline

1. Original CHAOS with work stealing

   a) general presentation

   b) possible improvements

2. Different vertex set size partitions.

3. Same edge set size partitions.

   a) implementation.

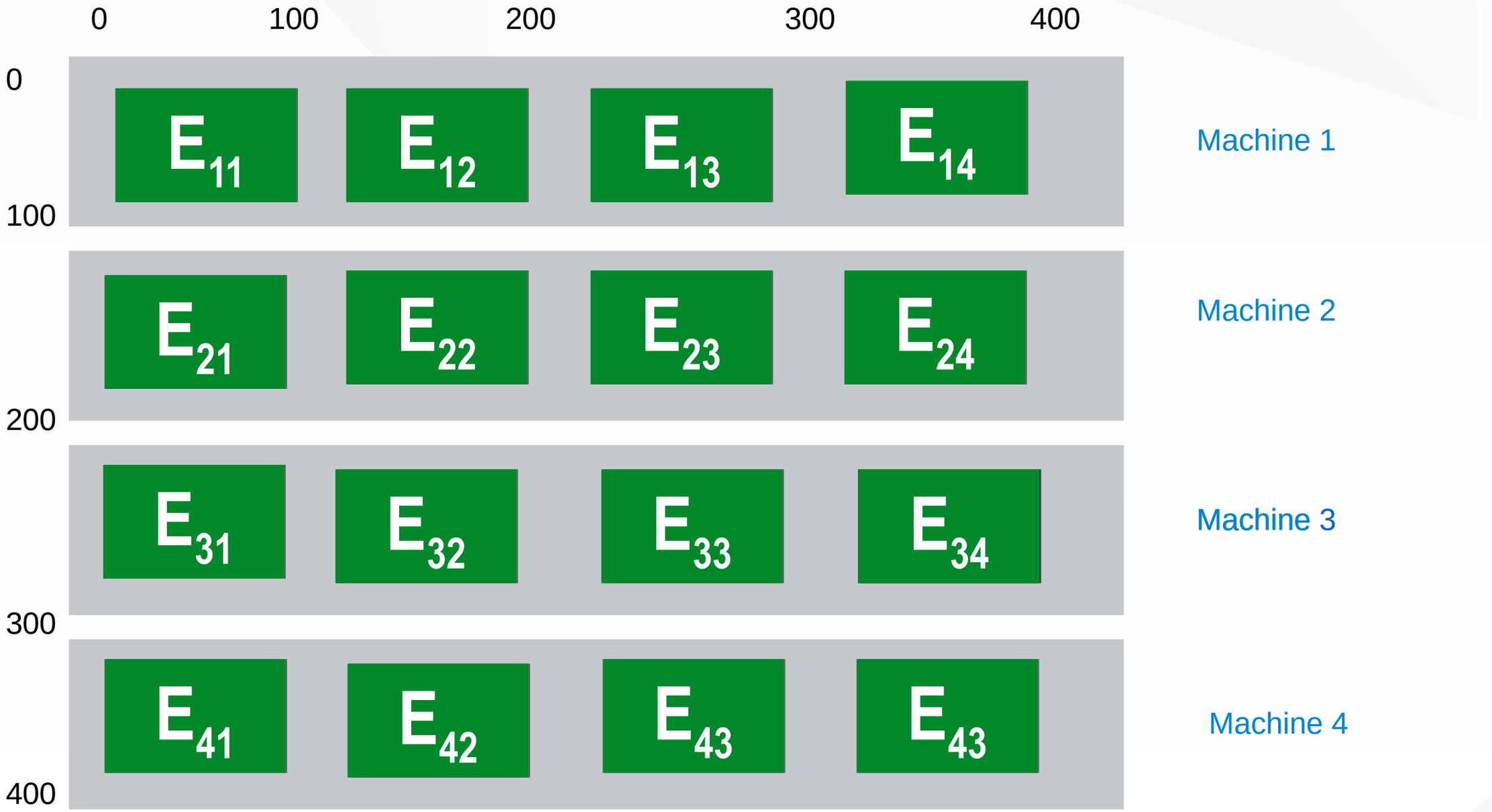
   b) drawbacks and optimizations.

   c) results.

4. Vertex relabeling.

5. Grid partitioning.

Grid Partitioning: Row Allocation



Discussion

Pros:

+ partition search for updates becomes a lookup

Cons:

- need two preprocessing passes

- poor work balance for threads within a partition

- hard to implement it efficiently with current code

- need for edge oriented code => race conditions during gather

=> Current implementation is 2x slower than the goal.

Conclusion

The final version is close to the goal to beat but :

→ The trade-off we used might be input dependent

→ Achieving static load balance is not an easy task

→ Solving one problems gives you its dual

→ Most simple operations does not scale up well.

Further Work

→ Explore the grid partitioning idea.

→ Focusing on other graphs than RMAT type graphs would be interesting

THANK YOU !

Backup-slides

Very different vertex sets problem.

At the beginning of gather the last machines, while loading their states, will already receive requests for reading updates => increased gather time

Even if vertices are stripped disks are busy reading vertex states while other machines requests them updates => Gather time and machine id are inversely proportional.