Load balancing techniques for CHAOS

Vlad Ioan Haprian

Laurent Bindschaedler

Willy Zwaenepoel

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.

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

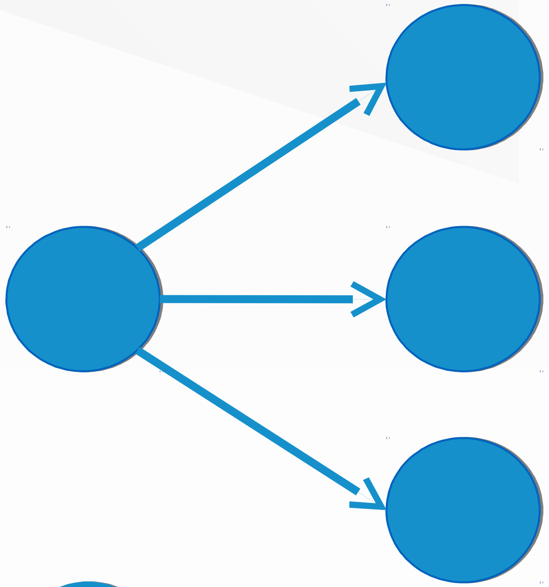
2. **Minimize preprocessing time** => partitioning phase is very simple.

Edge-centric Graph Processing

- Store state in vertices

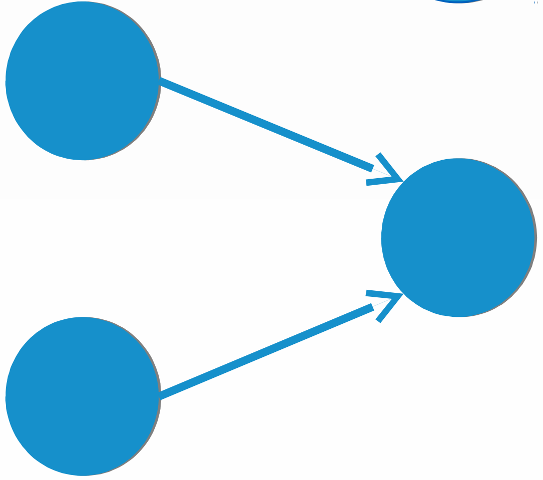
- **Scatter** – For all outgoing edges:

```
new update = f(vertex state)
```



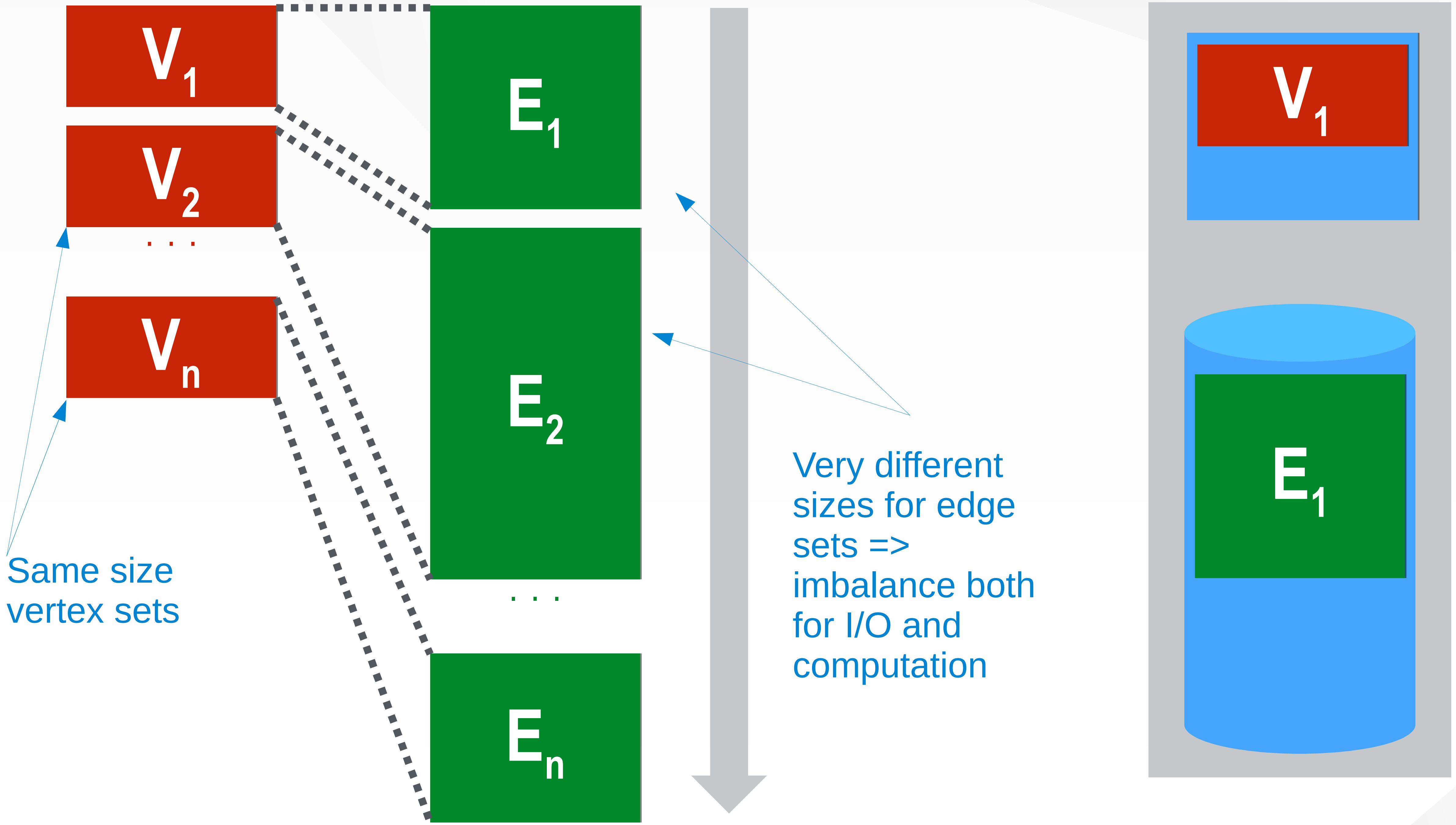
- **Gather** – For all incoming edges:

```
vertex value =
   g(vertex value, update)
```



```
Order independent!
```

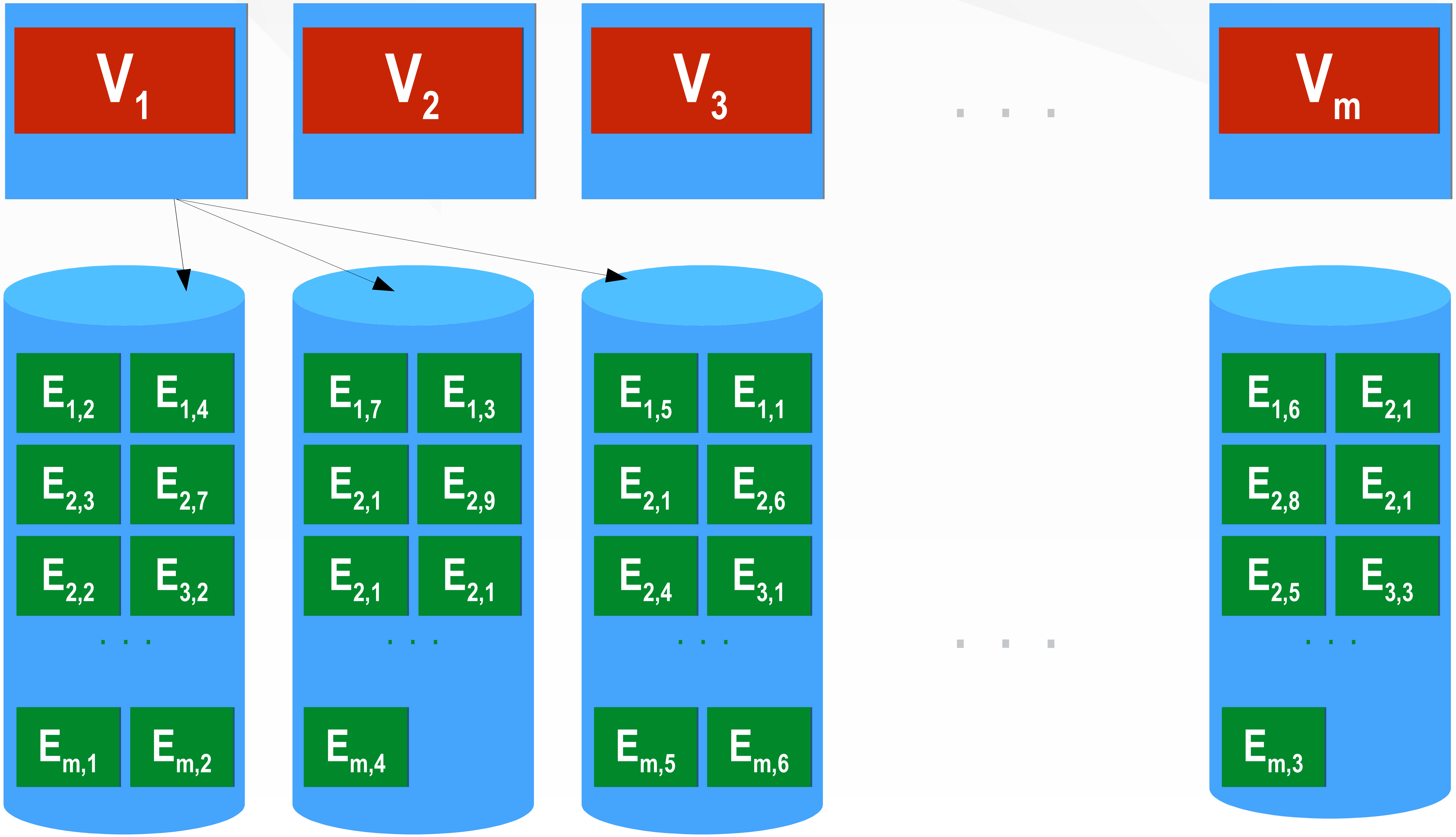
Very simple partitioning phase => computation and I/O are unbalanced



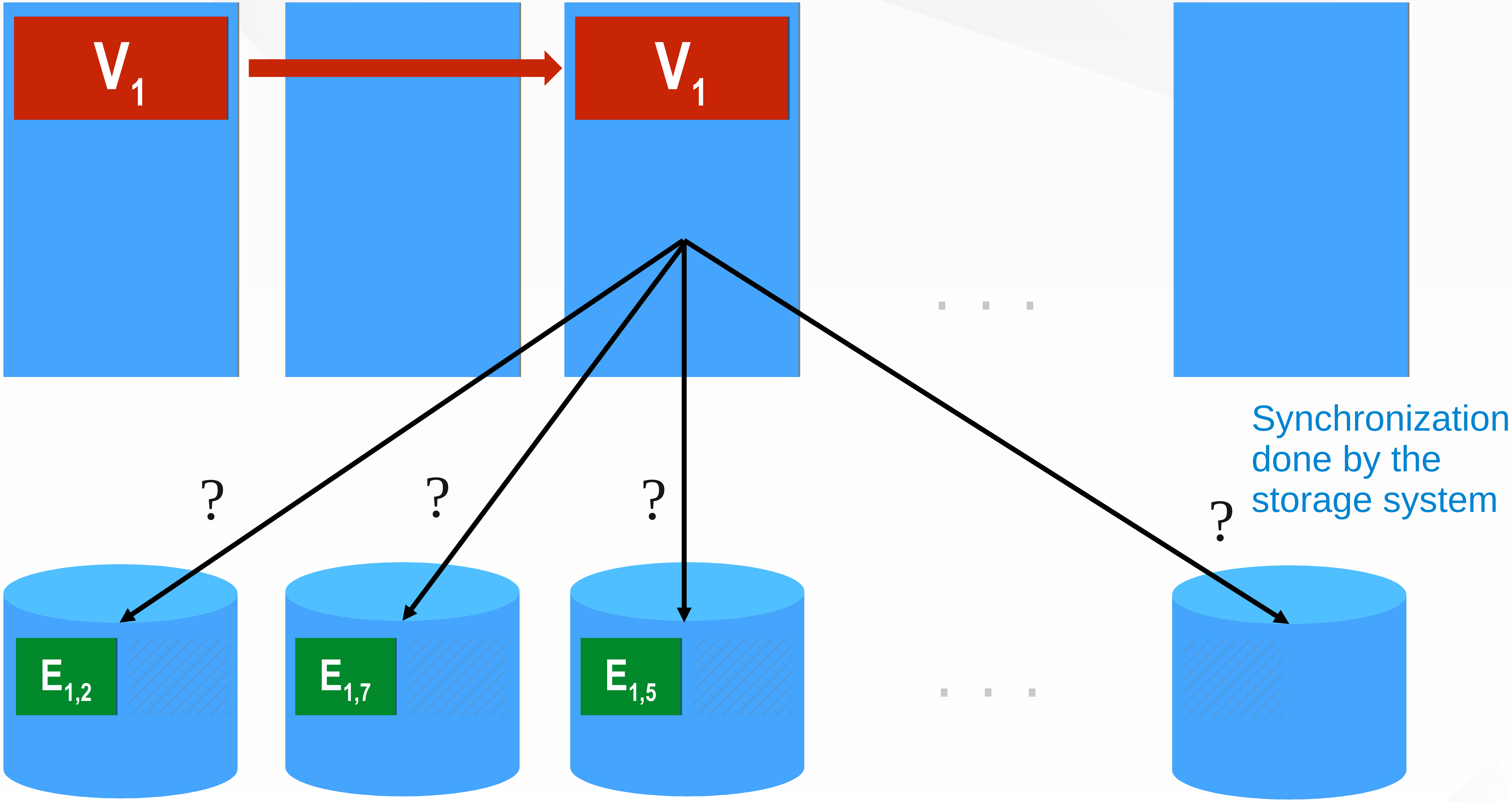
Same size vertex sets

Very different sizes for edge sets => imbalance both for I/O and computation

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

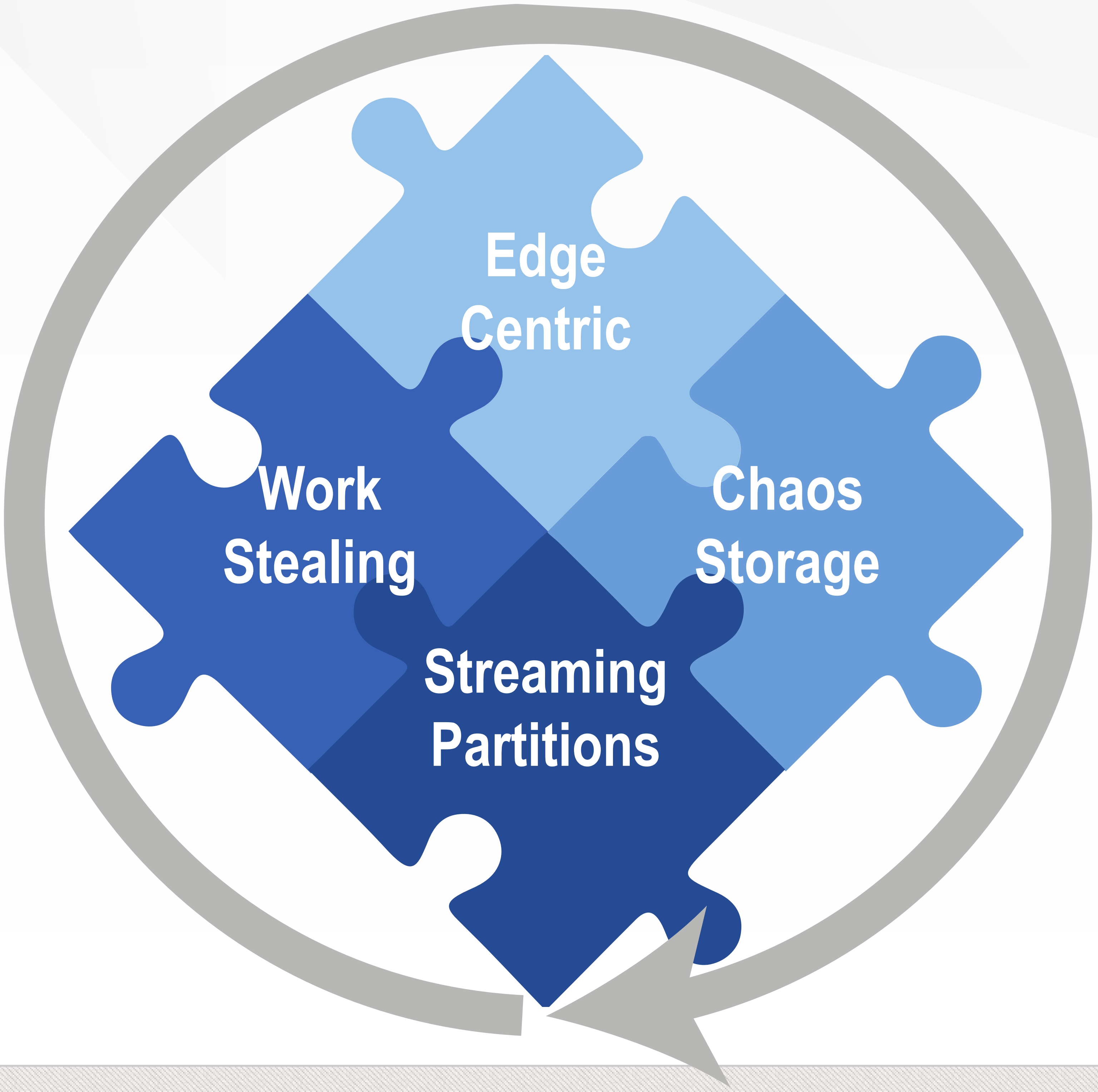


Do **work stealing** in order to achieve **computational balance**



Synchronization done by the storage system

Recipe for Chaos



Edge Centric

Work Stealing

Chaos Storage

Streaming Partitions

Randomization

Randomization

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.

Work stealing is not free !

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

How important this cost is ?

For RMAT-28 the read time increases with ~ 35 % when work stealing is on.

Possible solutions

1. Find optimal size of vertex-sets <=> optimal number of partitions per machine.

2. Do more preprocessing in order to obtain balanced partitions.

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.

Stealing vs **Streaming** friendly

Trade-off :

→ smaller vertex sets =>

more partitions per machine =>

partitions are more balanced =>

work stealing is less needed and cheaper.

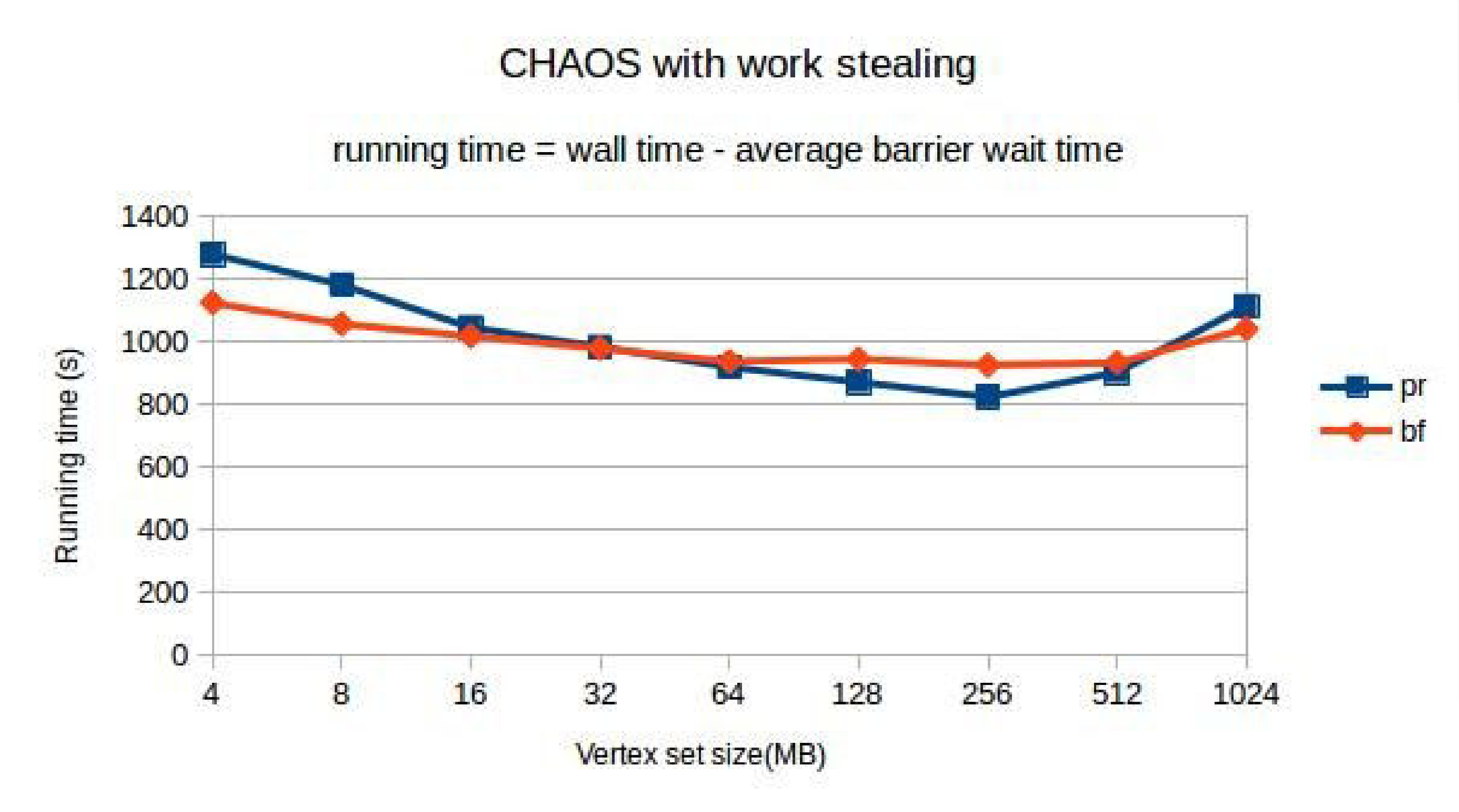
→ small vertex sets =>

smaller edge sets =>

sequentiality is lost

Variating vertex set size experiment

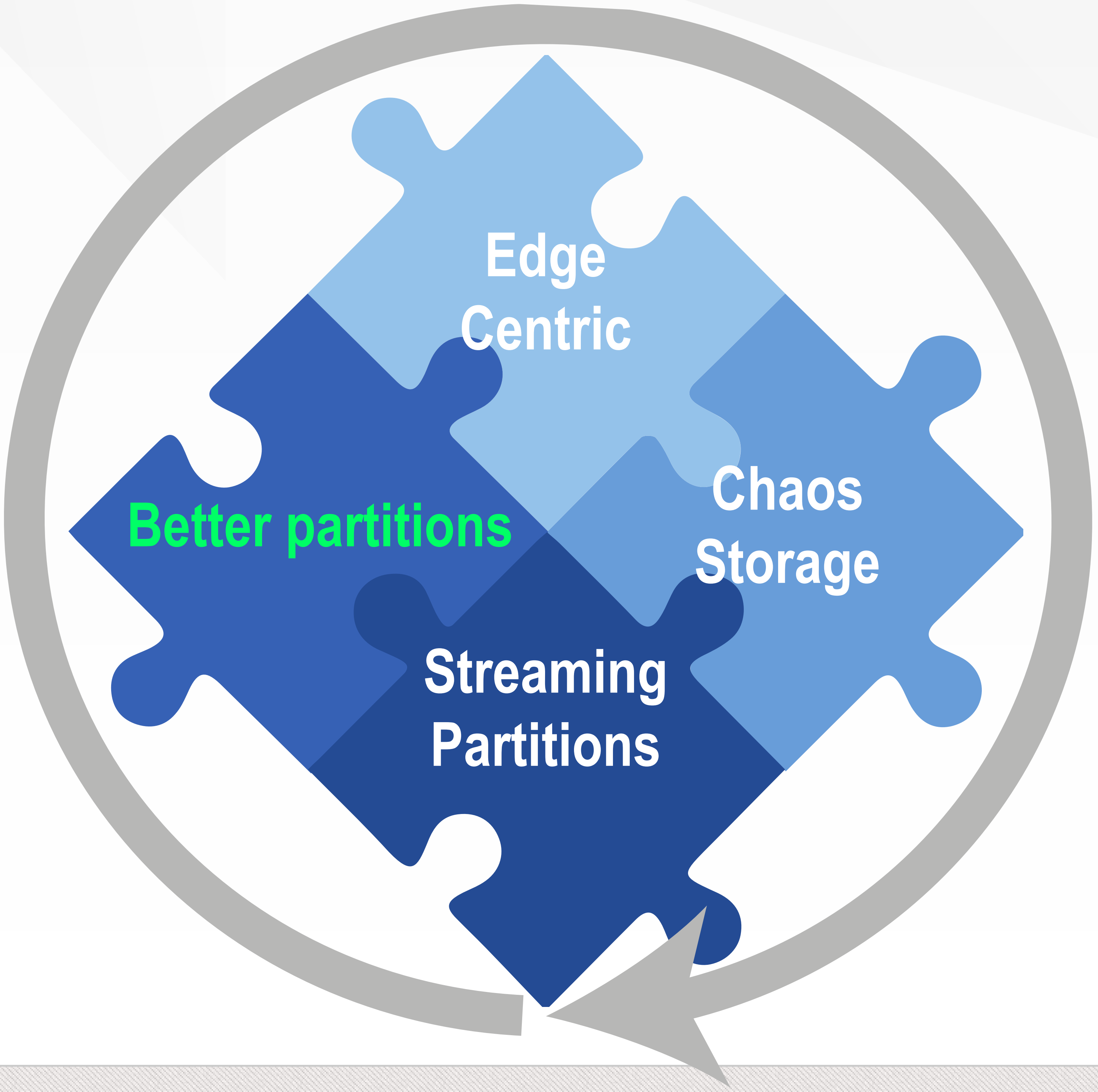
RMAT-28 (2GB of vertices) on a cluster of 8 machines with 2GB of RAM per machine.



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

Randomization



Edge Centric

Better partitions

Chaos Storage

Streaming Partitions

Randomization

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.

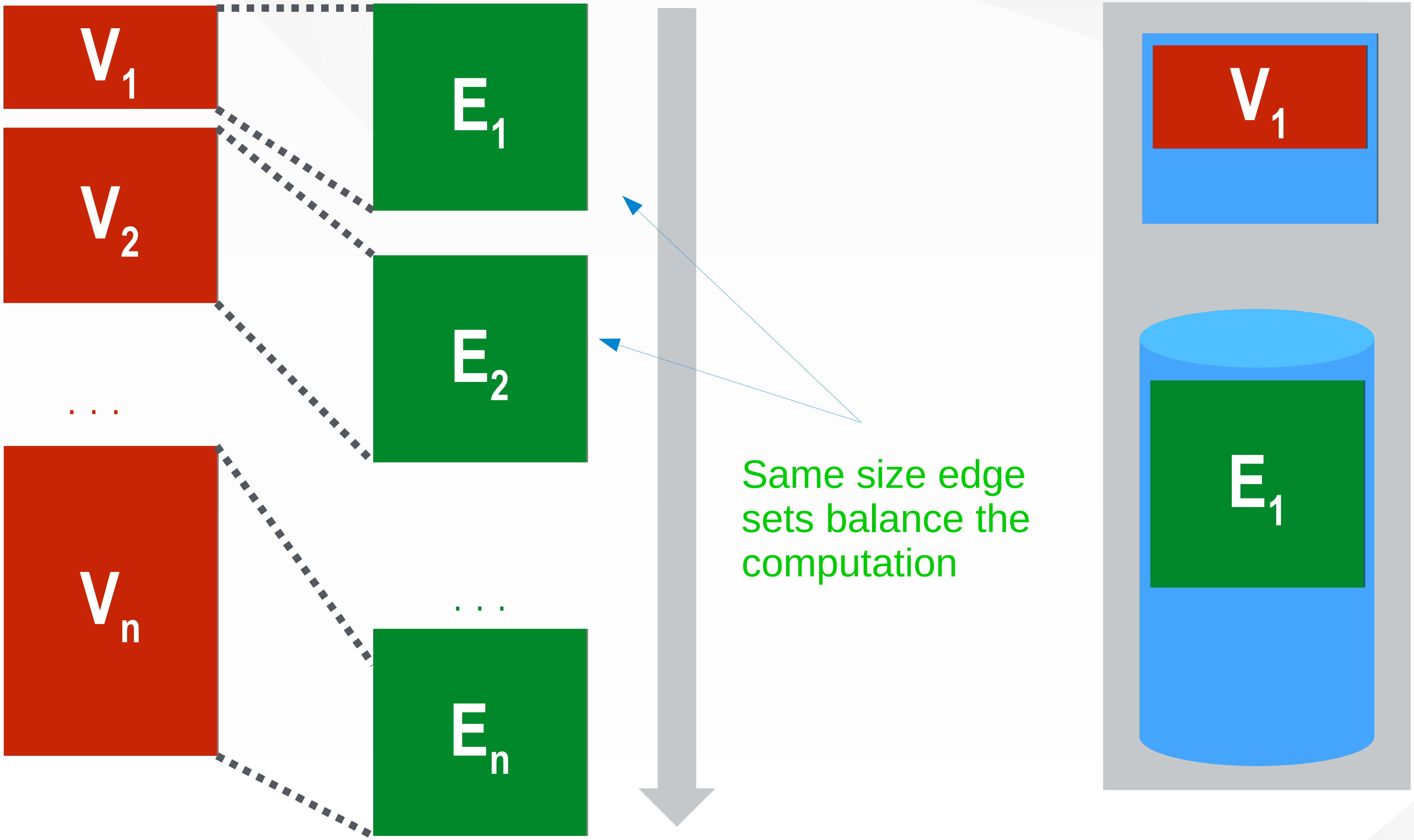
   c) results.

4. Vertex relabeling.

5. Grid partitioning.

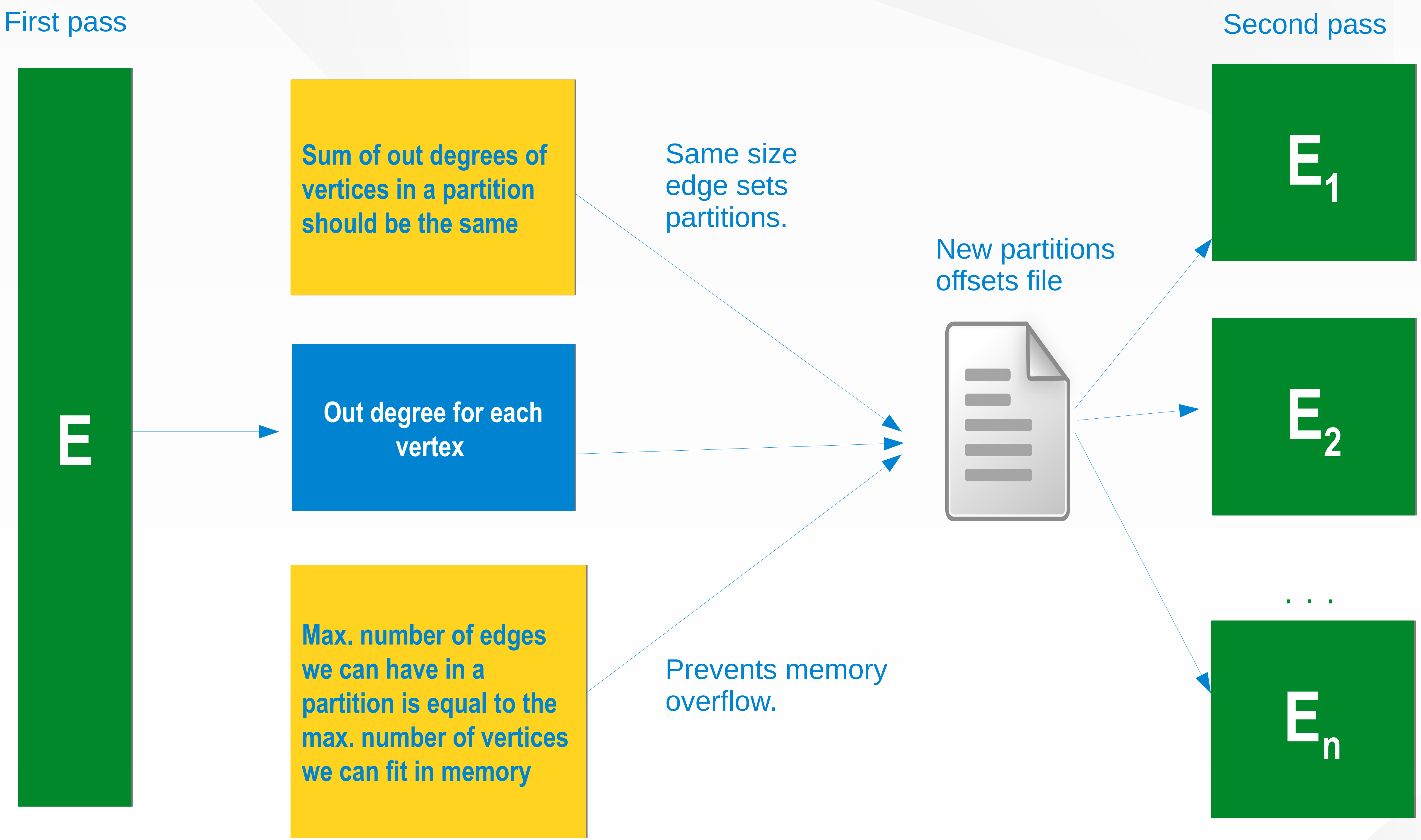
Better partitioning => no need for work stealing

Processing model is the same



Same size edge sets balance the computation

How the new partitions are generated ?



First 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

Second pass

E_1

E_2

. . .

E_n

Guideline

1. Original CHAOS with work stealing

   a) general presentation

   b) problems

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.

How good we are ?

1. How much we manage to reduce the imbalance ?

Comparing to the old version of Chaos, when work stealing is off the improvement is huge : the scatter barrier time is around 4.5 times smaller.

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 to almost 0 in the old version of Chaos with work stealing.

What is wrong?

1. Same size edge sets partitions do not balance the I/O:

This is because at each scatter and gather phase the vertex state need to be loaded at the beginning and stored at the end. As vertex state sizes are very different this produces I/O imbalance.

2. Computation is slower and not balanced due to the new partition search overhead:

→ in the old partitioning mode the partition is determined just by doing a bit shift of the vertex id. This is no longer possible in the new partitioning mode because of different size vertex sets => partition is determined by binary searching the vertex id in a partition offset array => some machines need 1 array access, others need 3 (consider a scenario of 8 machines and 1 partition per machine).

→ array access much more expensive than bit shift and this need to be done O(|V|) times (number of phases also matters here).

Reducing the I/O imbalance

We use a slightly changed partition constraint :

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

=> I/O balance (for c = 4)

=> we still have some computational imbalance due to partition search of updates.

Different values for c could be used. The drawback is that they will generate enough different edge sets for the computation to not be balanced.

Reducing the computation time

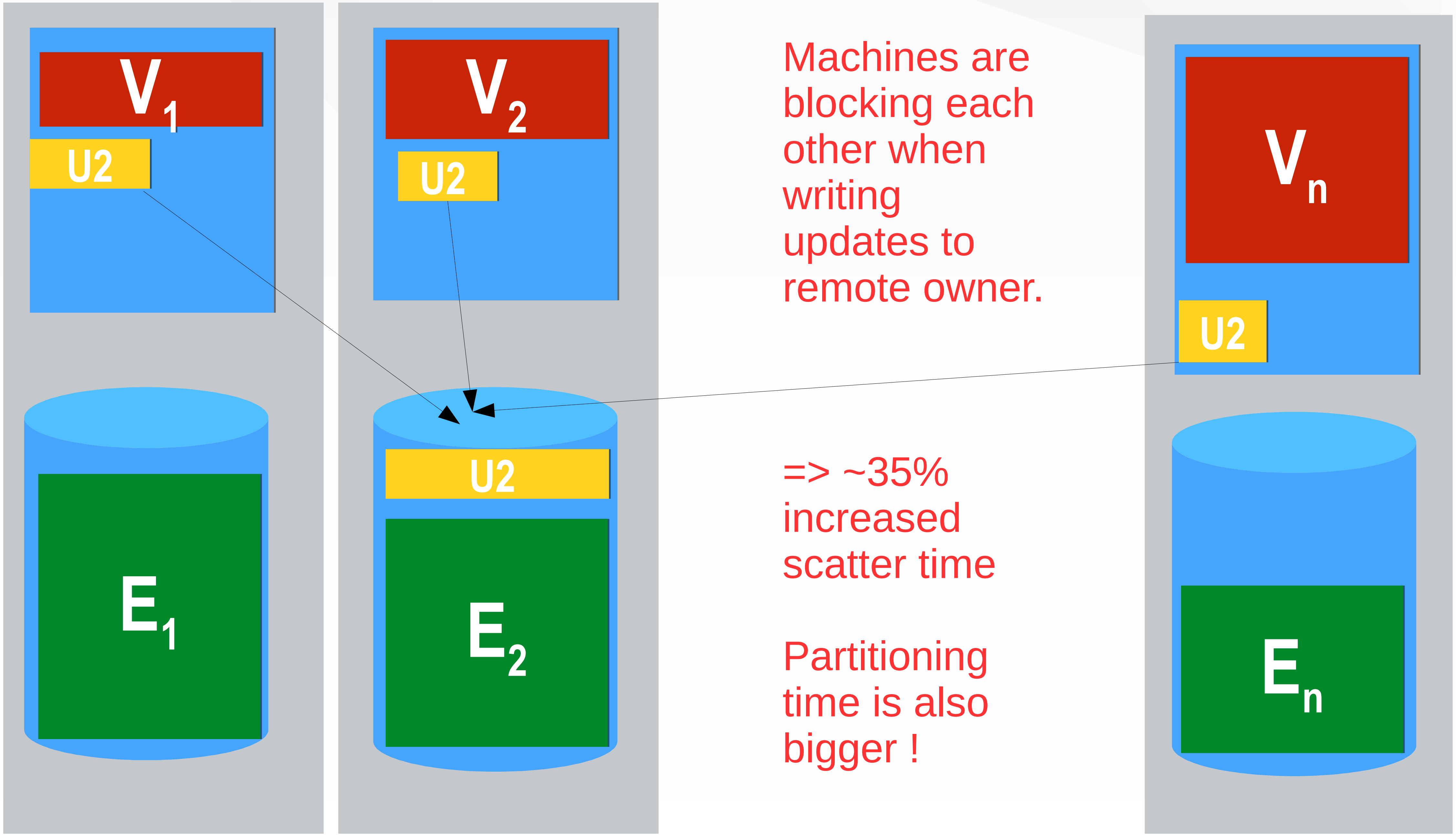
The main overhead in computation is partition search => use caching !

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

Do we still need edge stripping ? What about keeping everything local ?

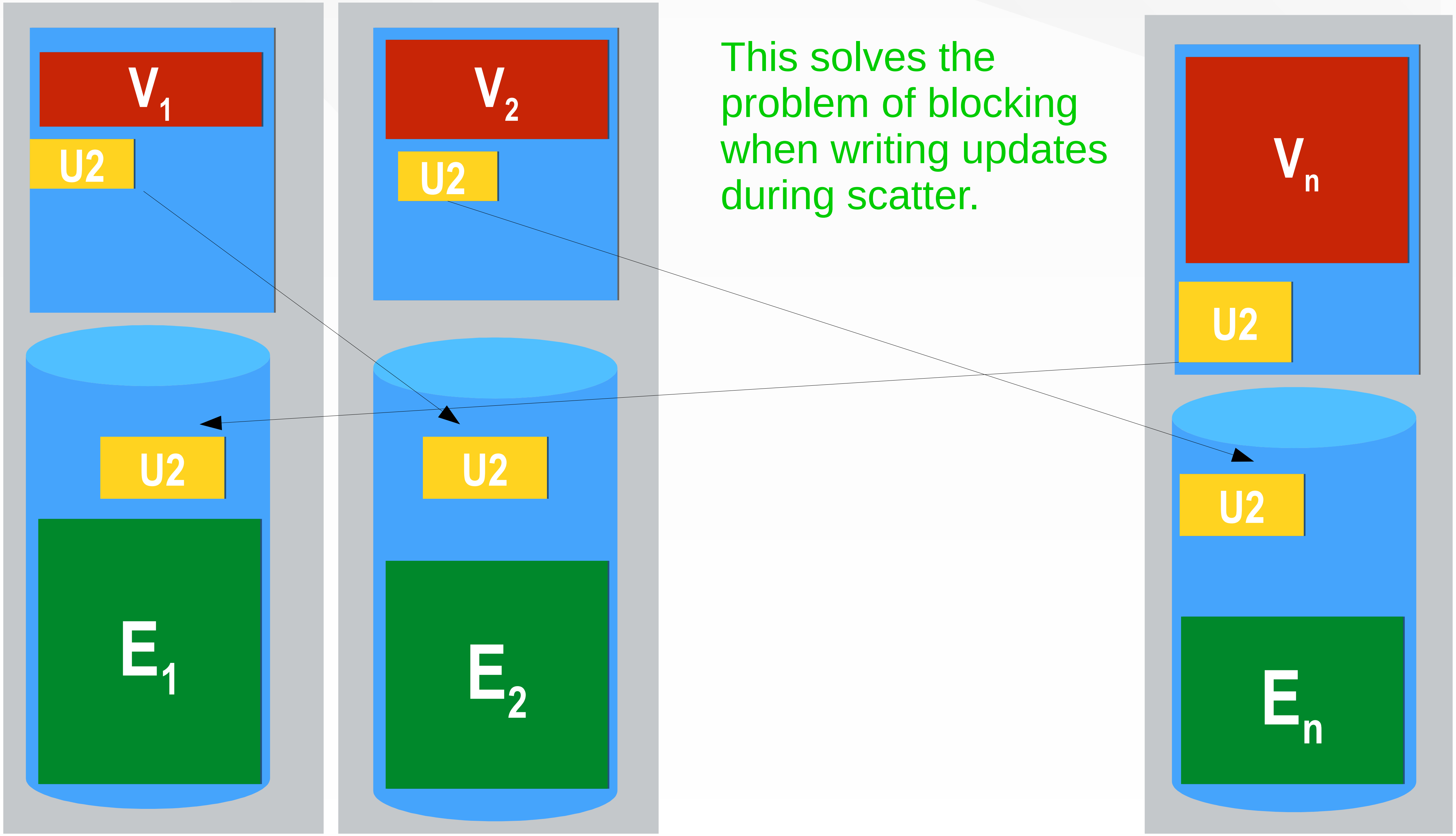


Machines are blocking each other when writing updates to remote owner.

=> ~35% increased scatter time

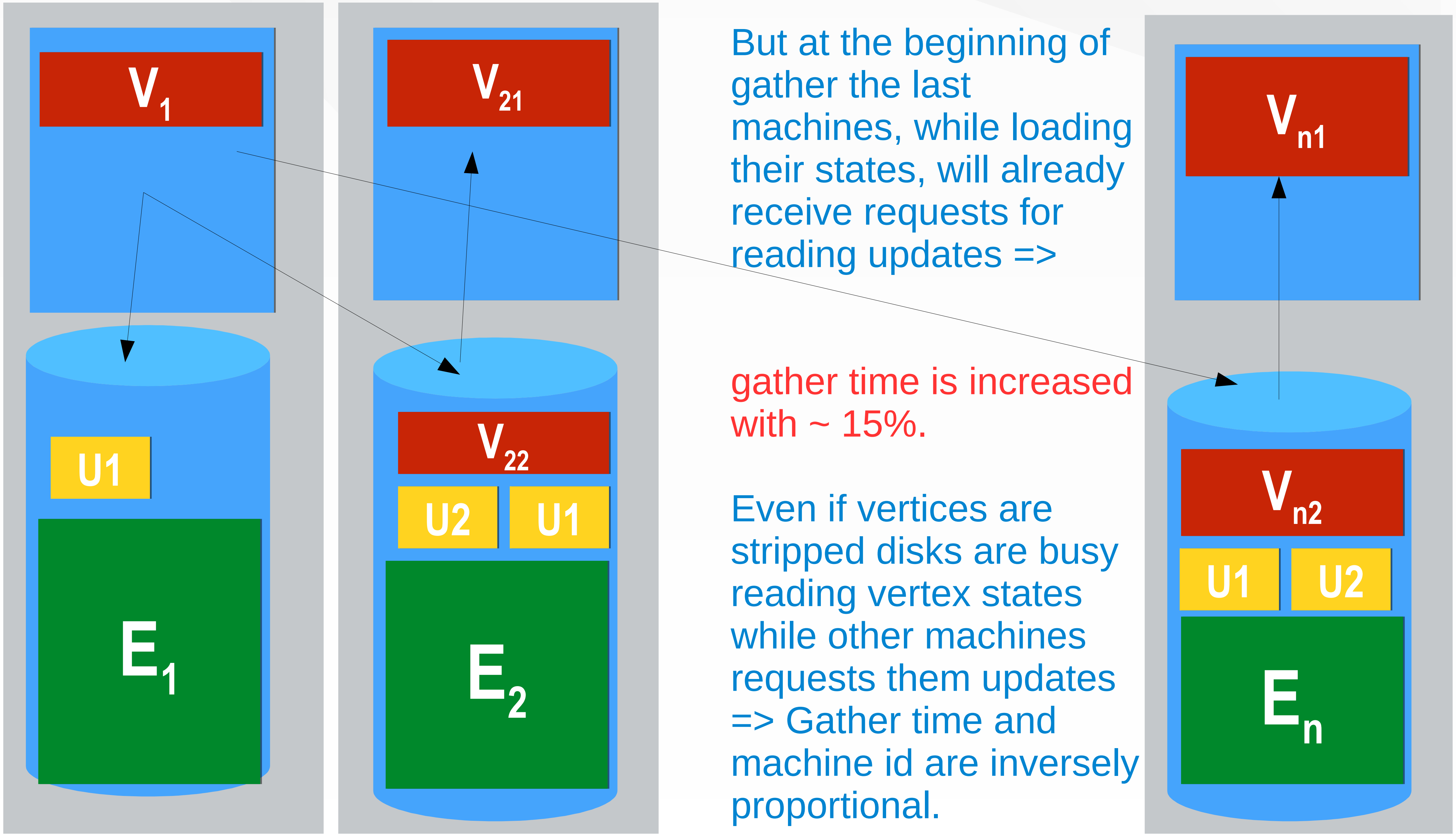
Partitioning time is also bigger !

Stripe the updates, keep edges and vertices local



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

Stripe the updates, keep edges and vertices local. (cont)



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

gather time is increased with ~ 15%.

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.

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.

Where we are compared to the goal to beat ?

Compared to the old version of CHAOS with work stealing

our implementation is:

→ 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

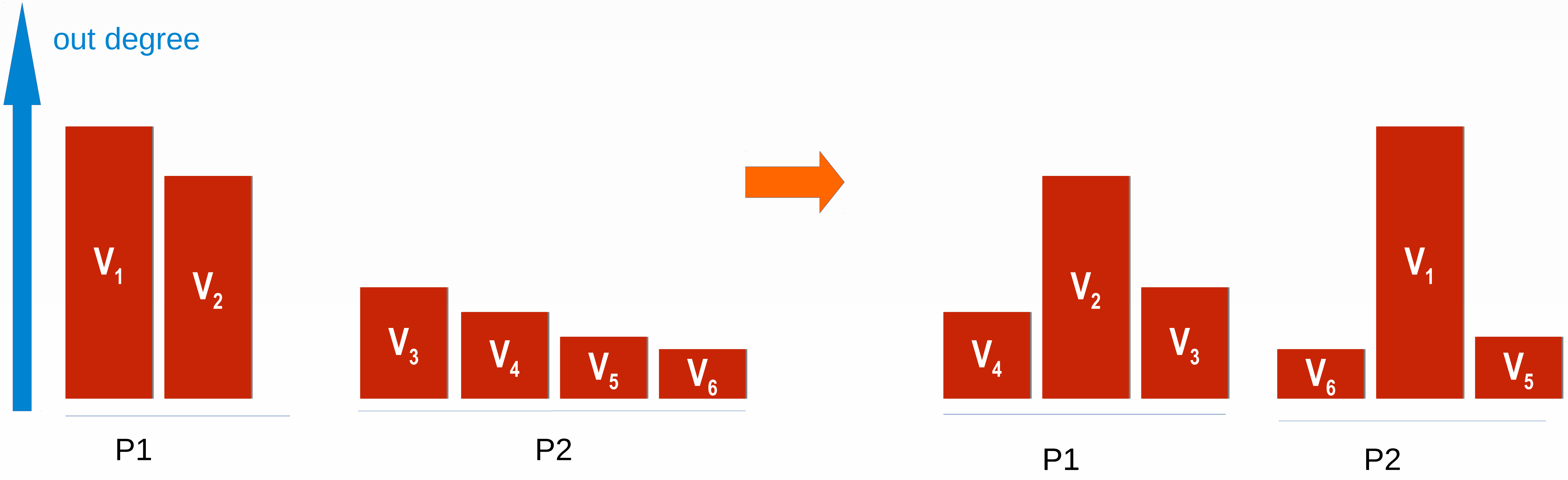
But might need one vertex relabeling phase in order to make this optimization possible without RAM overflow.

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:

- the random permutation need to be generated (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.

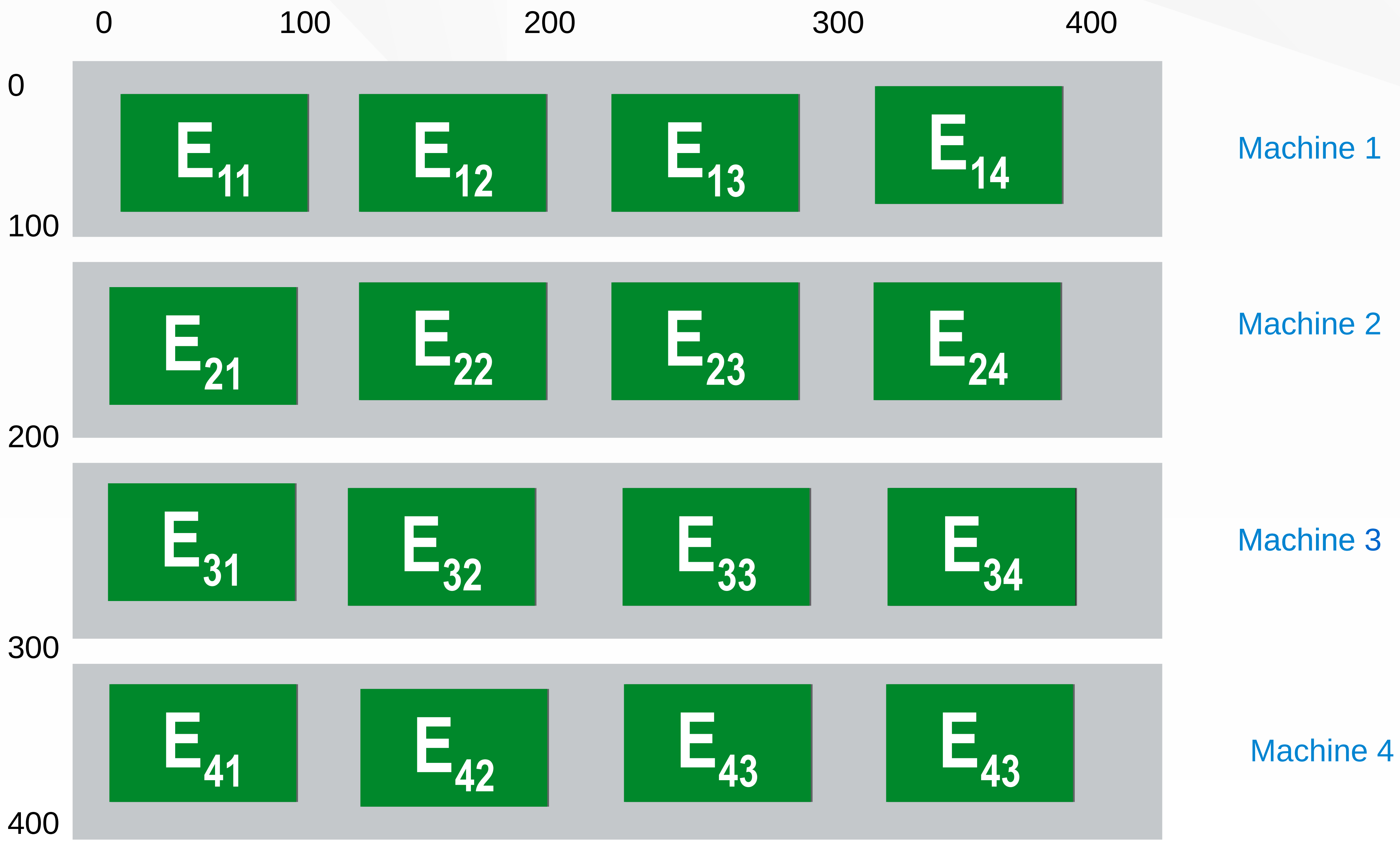
   b) drawbacks and optimizations.

   c) results.

4. Vertex relabeling.

5. Grid partitioning.

Grid partitioning: idea (row allocation)



0 100 200 300 400

0 – 100: E_{11} E_{12} E_{13} E_{14} Machine 1

100 – 200: E_{21} E_{22} E_{23} E_{24} Machine 2

200 – 300: E_{31} E_{32} E_{33} E_{34} Machine 3

300 – 400: E_{41} E_{42} E_{43} E_{43} Machine 4

Discussion

Pros:

+ partition search for updates becomes a lookup

Cons:

- need two preprocessing passes
- poor work balance for threads within a partition (unless additional preprocessing is done)
- gather phase for a row implies 4 state load/store

THANK YOU !