Processes placement on multicore. Dynamic load balancing in Charm++

10th Annual Charm++ Workshop 2012

Emmanuel Jeannot Guillaume Mercier François Tessier

May 4, 2012

Introduction

State of Art

- Multi-node and multi-core architectures : Message passing paradigm
- Load balancing according to a flat topology

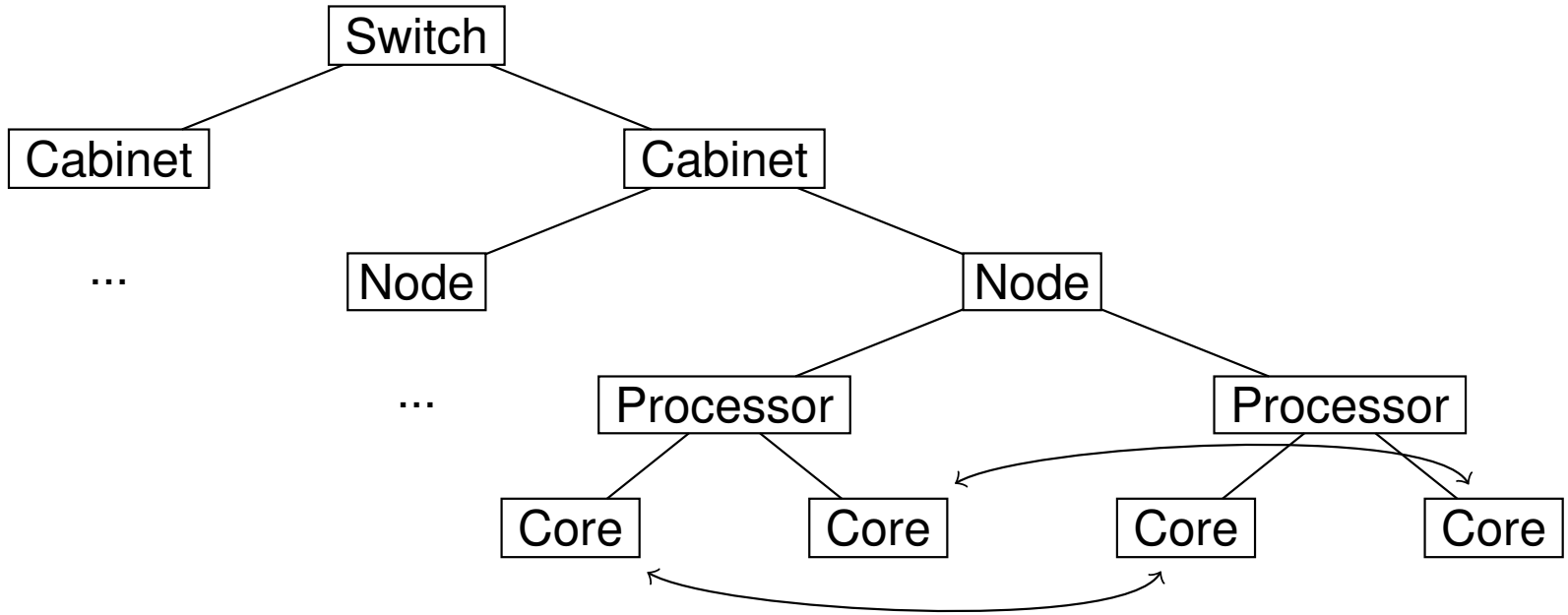
Problems

- Topology is not flat!
- Add the notion of processes affinity?
- Take into account the communication between processes?

Processes Placement

Why we should consider it

- Plenty of current and future parallel platforms have several levels of hierarchy
- Application processes don't exchange the same amount of data (affinity)
- The process placement policy may have an impact on performance
  - Cache hierarchy, memory bus, high-performance network...
  - Metrics : Amount of data, number of messages



Problems

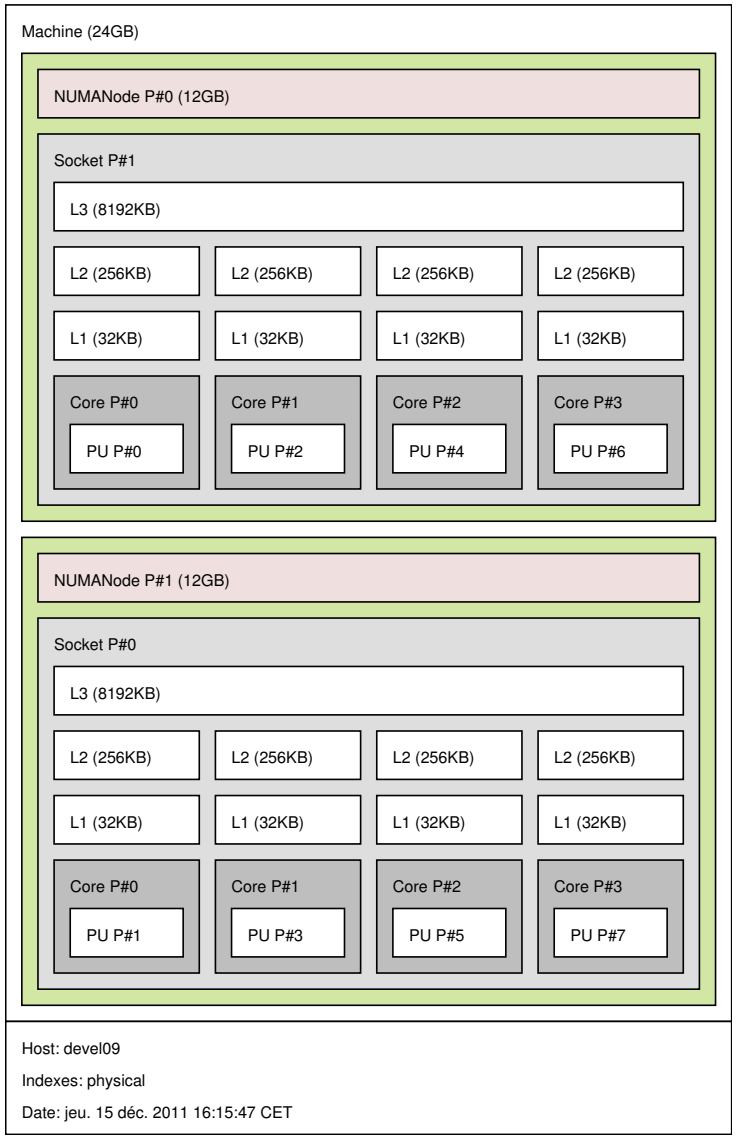
Given...

- ... The parallel machine topology
- The application communication pattern

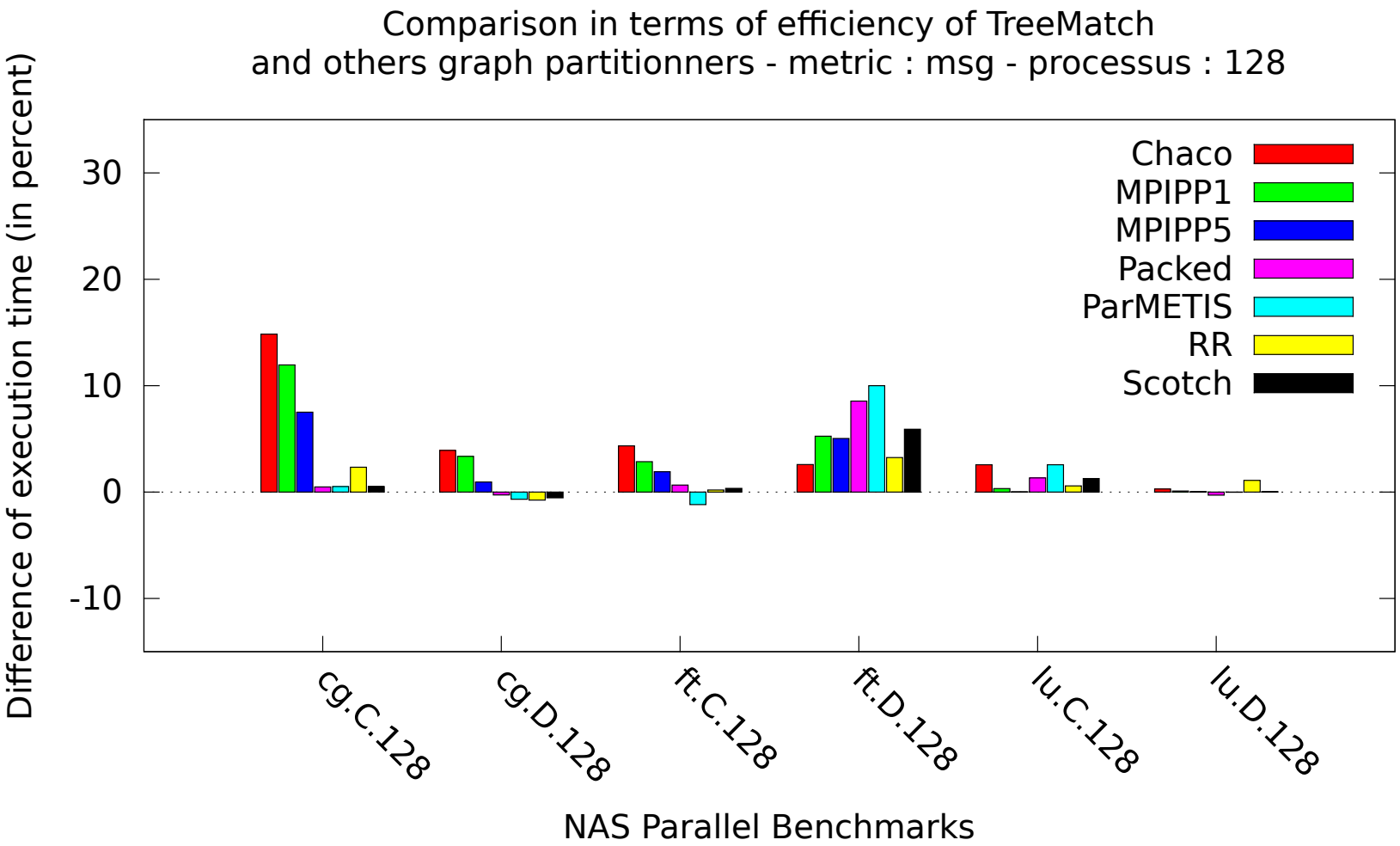
Map application processes to physical resources (cores) to reduce the communication cost.

The TreeMatch Algorithm

- Algorithm and tool to perform processes placement based on processes affinity and NUMA topology
  - Processes affinity can be given by Charm++
  - Hwloc can provide us the topology
- According to processes affinity, TreeMatch is able to find a permutation to map them on cores
- Two strategies for Charm++

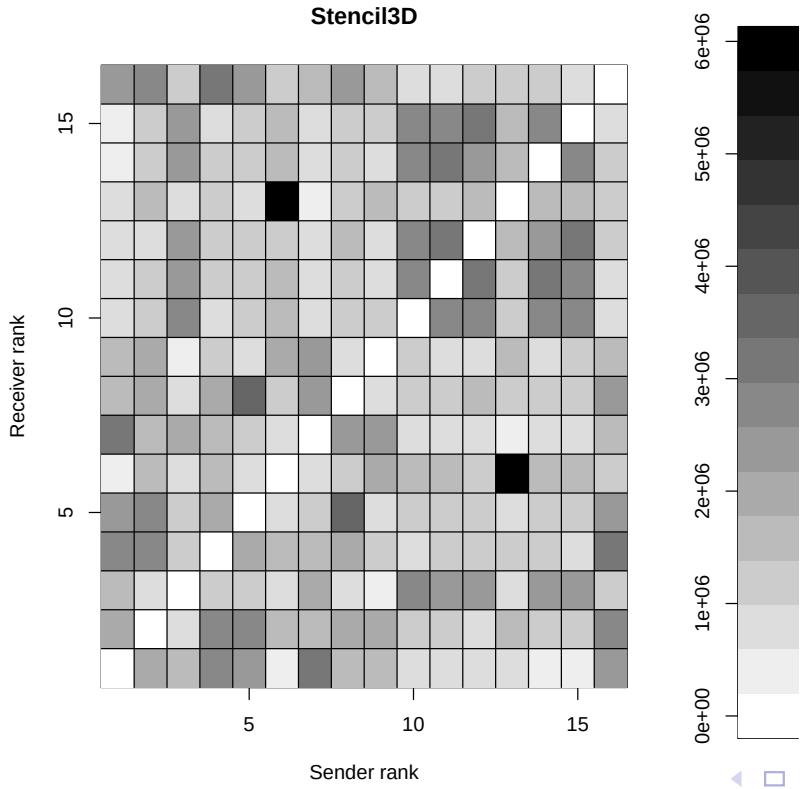


MPI Results


Comparison in terms of efficiency of TreeMatch
and others graph partitionners - metric : msg - processus : 128
Difference of execution time (in percent)
30
20
10
0
-10
Chaco
MPIPP1
MPIPP5
Packed
ParMETIS
RR
Scotch
cg.C.128
cg.D.128
ft.C.128
ft.D.128
lu.C.128
lu.D.128
NAS Parallel Benchmarks

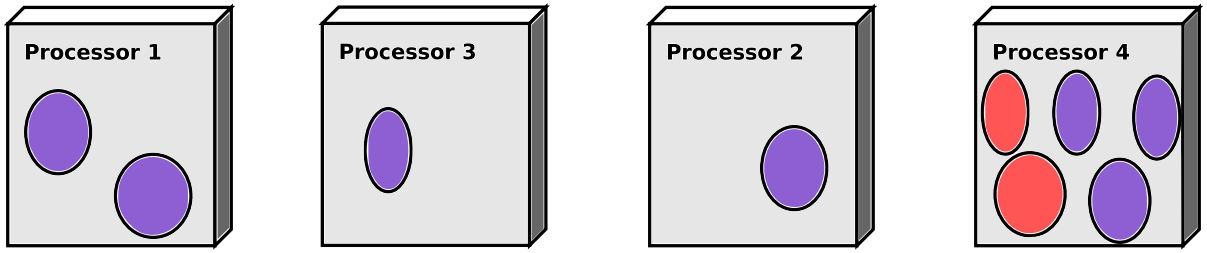
First Strategy

- GreedyLB to perform load balancing
- Create a communication matrix of groups of chares on each processor
- Run TreeMatch on this pattern and the corresponding topology
- Remap each group of chares on processors

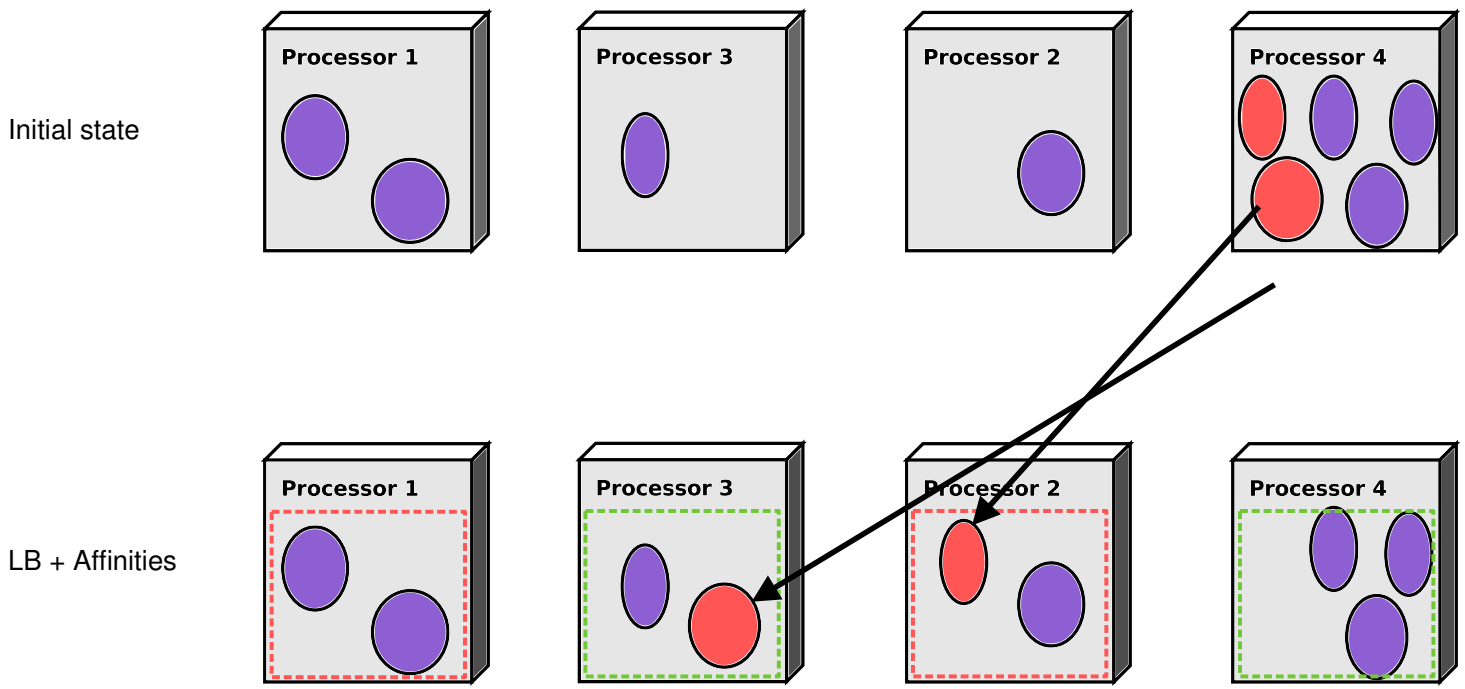


First Strategy

Initial state



First Strategy

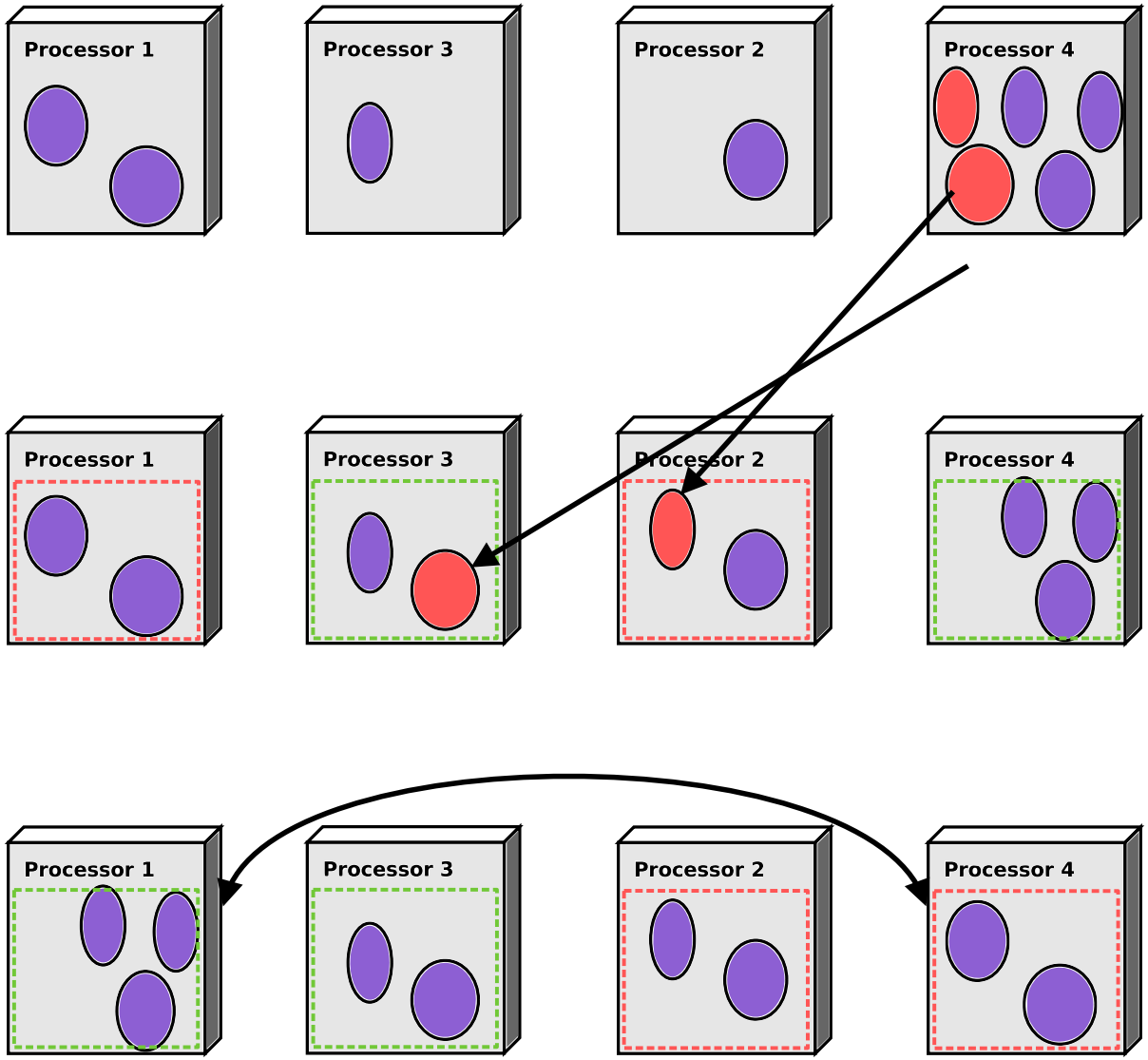


First Strategy

Initial state

LB + Affinities

Reorder groups

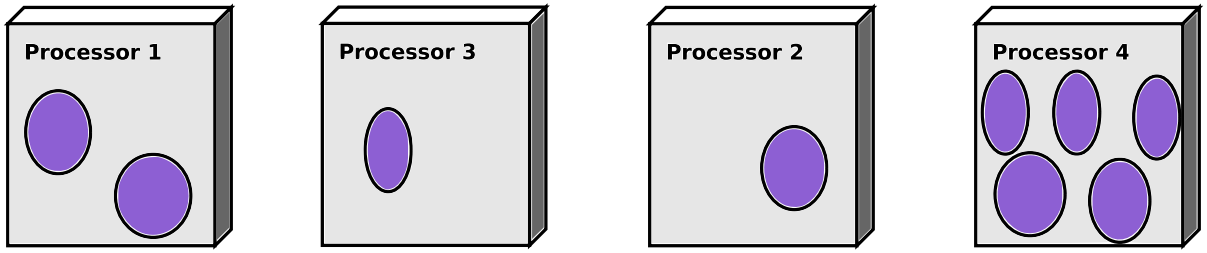


Second Strategy

- Create a communication matrix of chares
- Generate a fake topology, featuring as many leaf as chares (integer factorization)
- Run TreeMatch to find chares affinity
- Map chares to physical processors, taking into account the load and the affinity

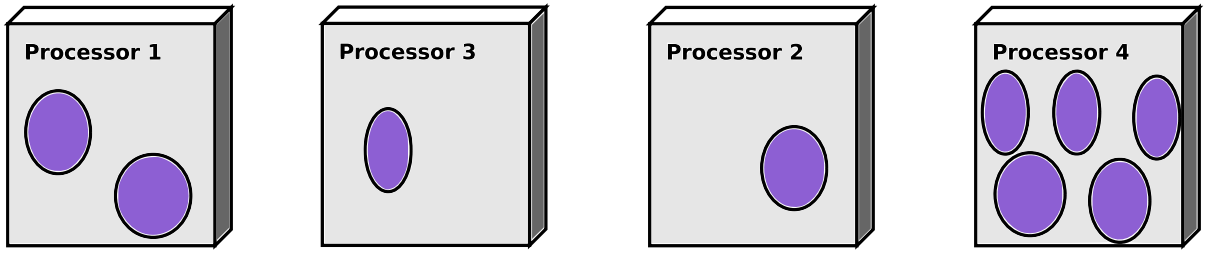
Example

Initial state

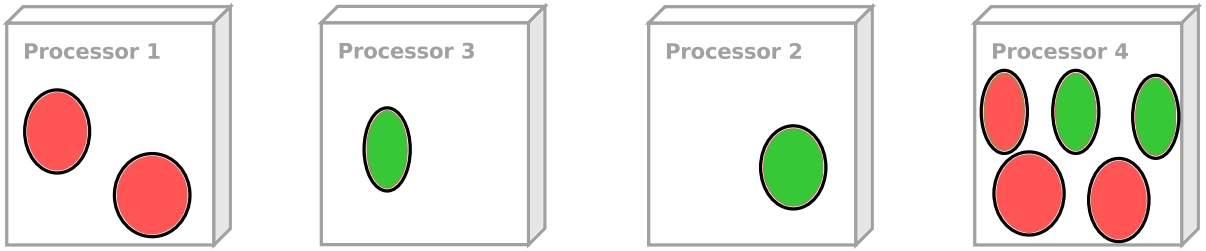


Example

Initial state

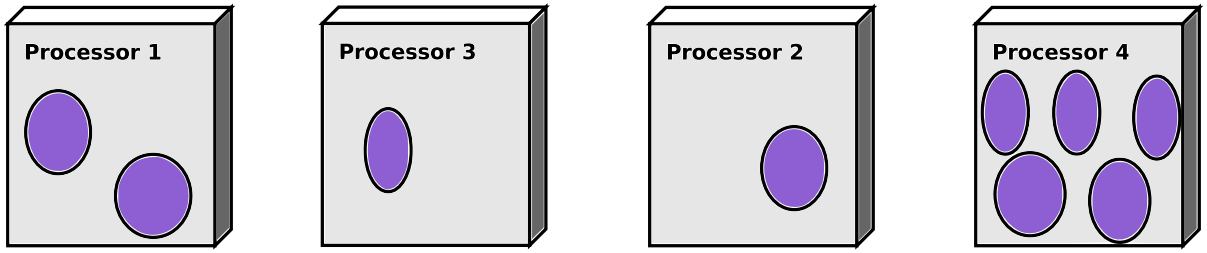


Find affinities

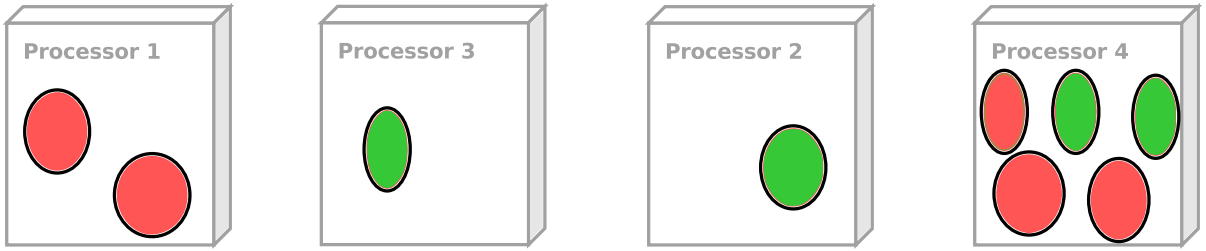


Example

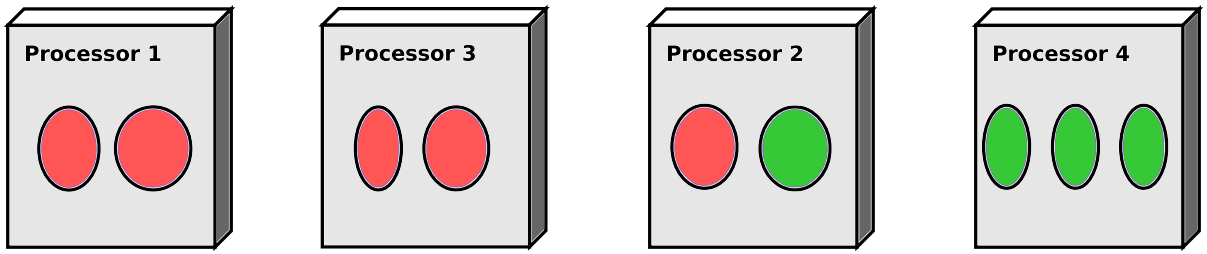
Initial state



Find affinities

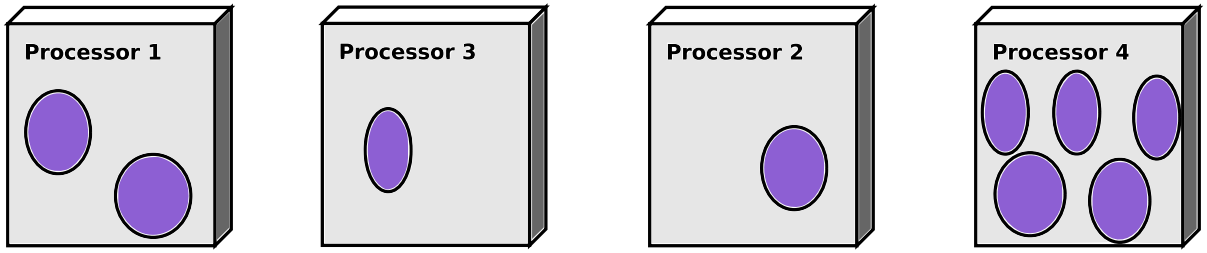


Reordering + LB

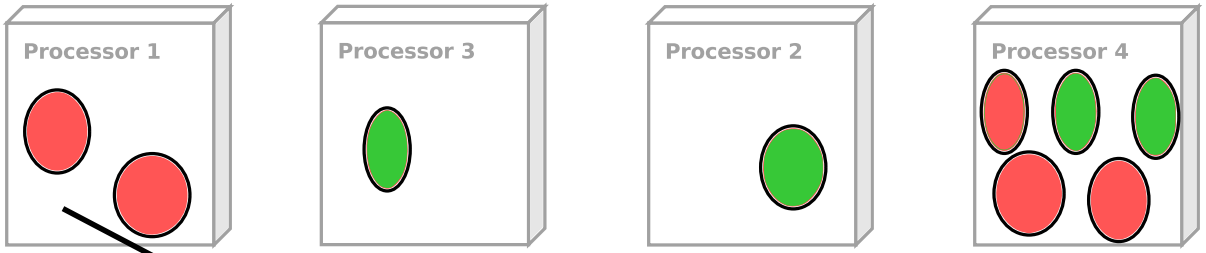


Example

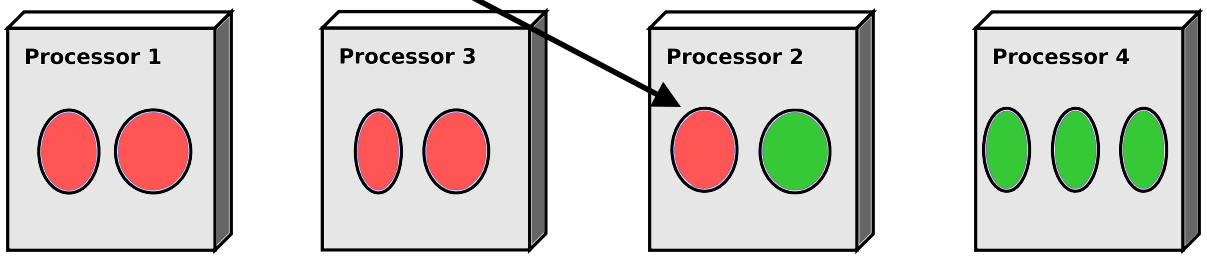
Initial state



Find affinities



Reordering + LB

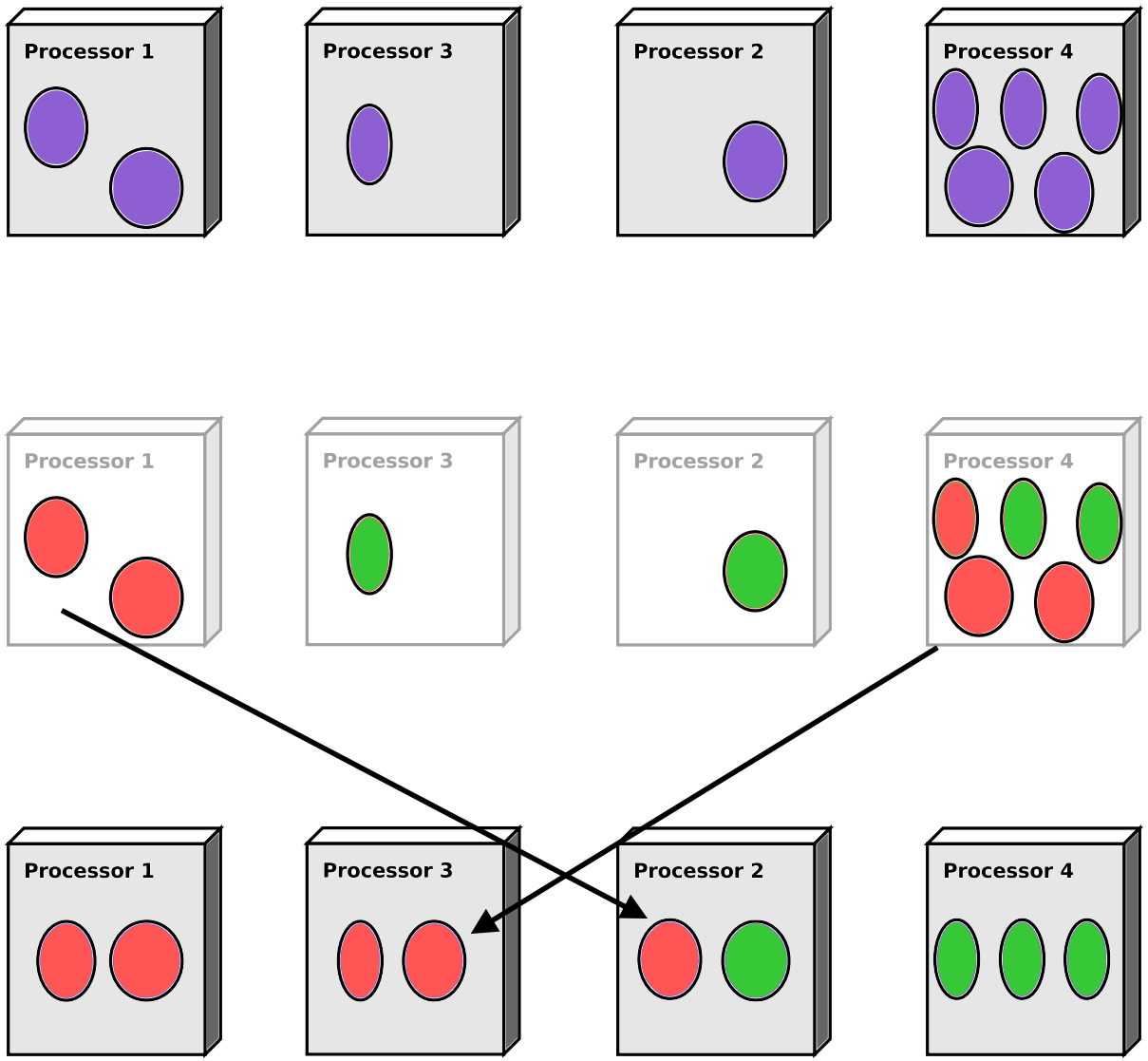


Example

Initial state

Find affinities

Reordering + LB

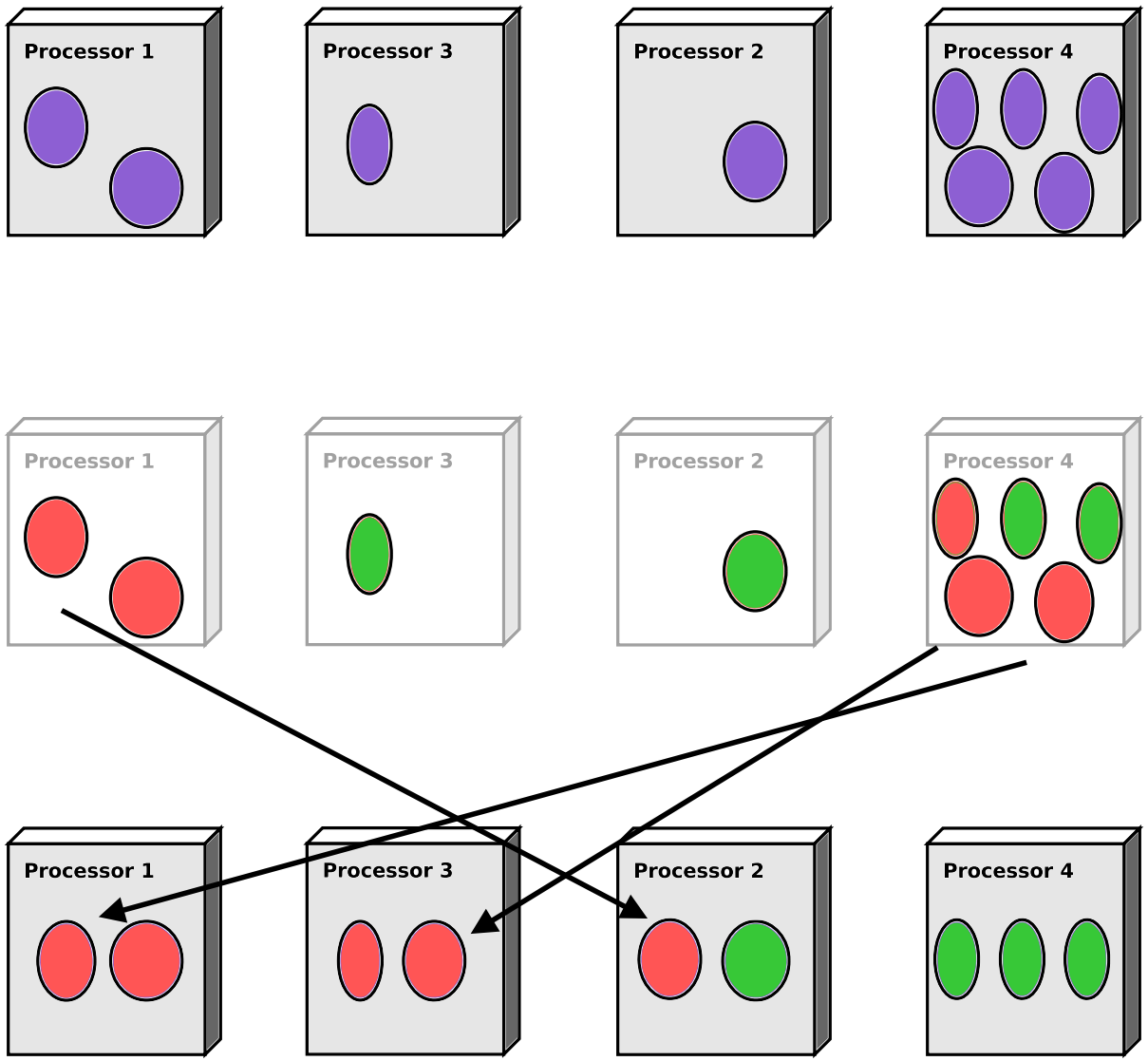


Example

Initial state

Find affinities

Reordering + LB

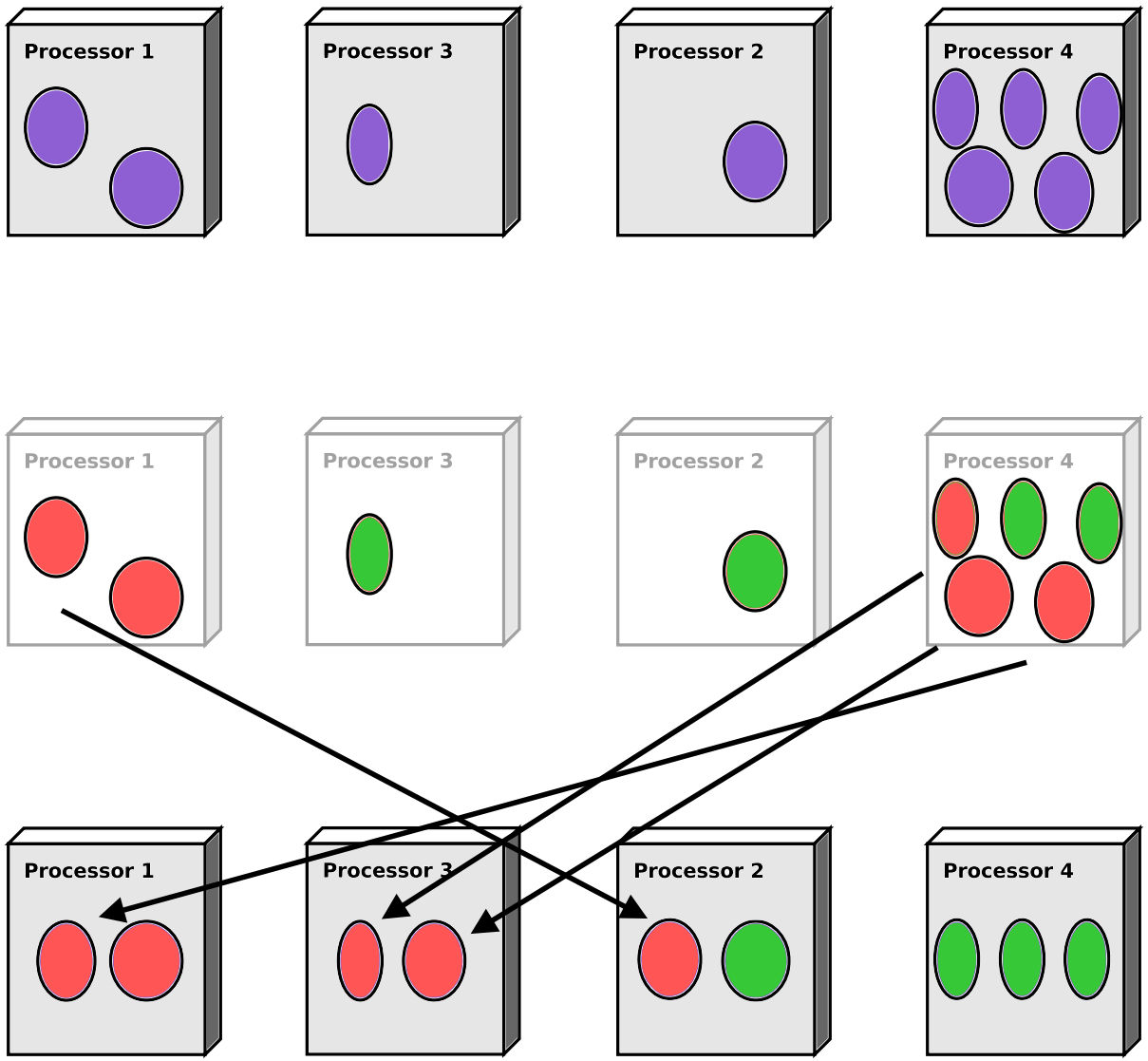


Example

Initial state

Find affinities

Reordering + LB

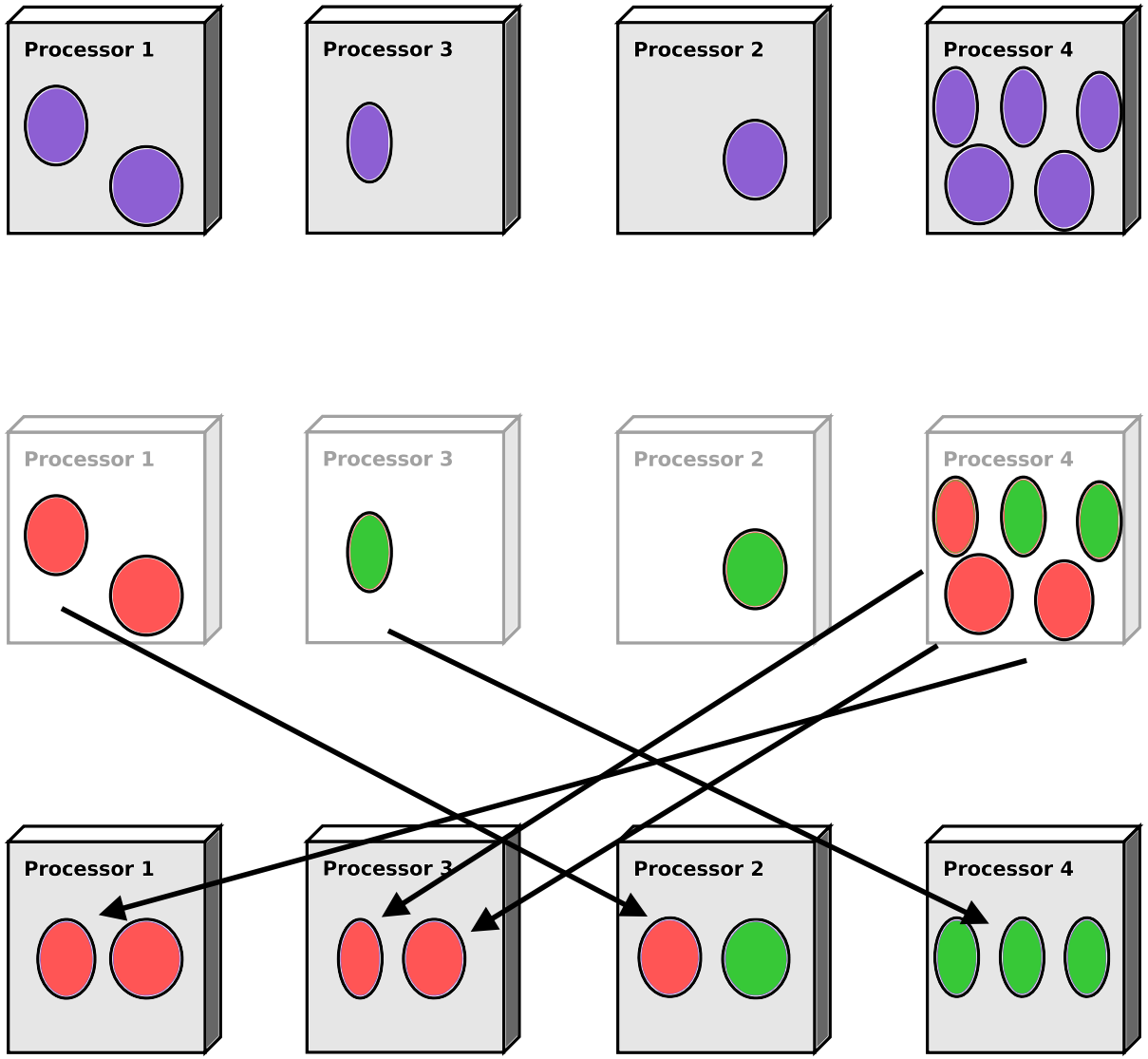


Example

Initial state

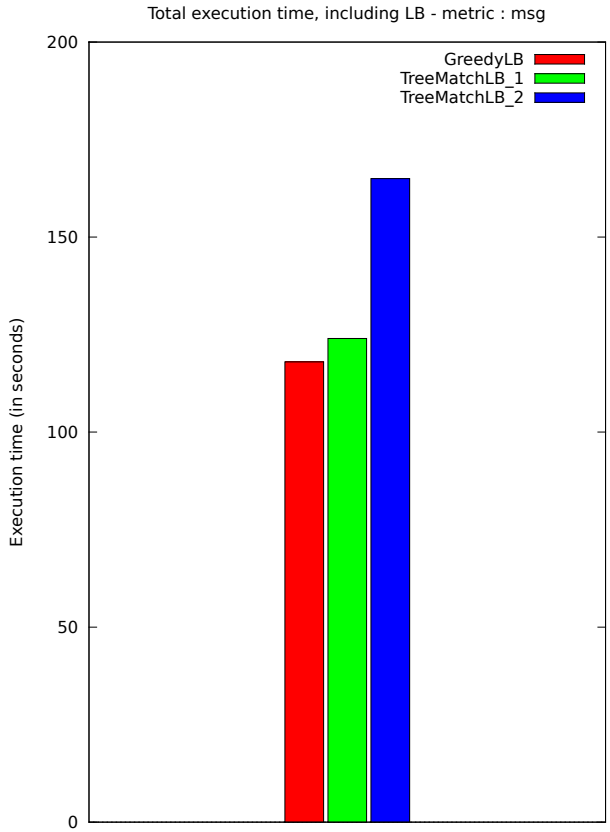
Find affinities

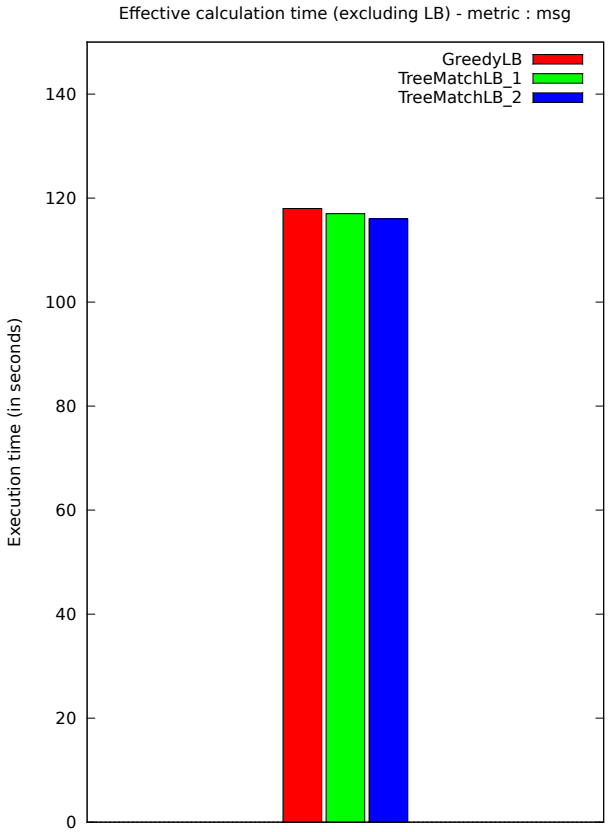
Reordering + LB



lb_tests

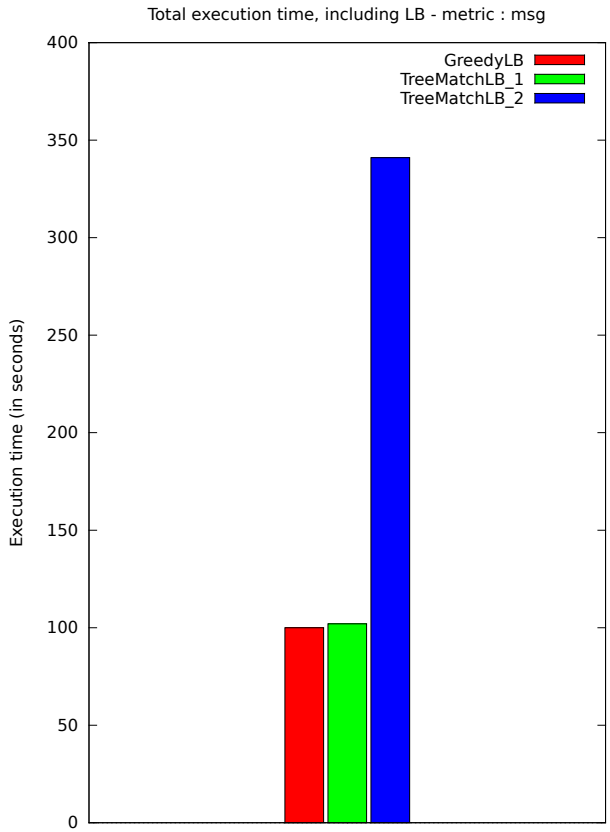
- Few communications
- *charmrun +p16 lb_test 10000 100 10 40 10 1000 randgraph +balancer TreeMatchLB +LBDebug 1 +setcpuaffinity +pemap 0-7*

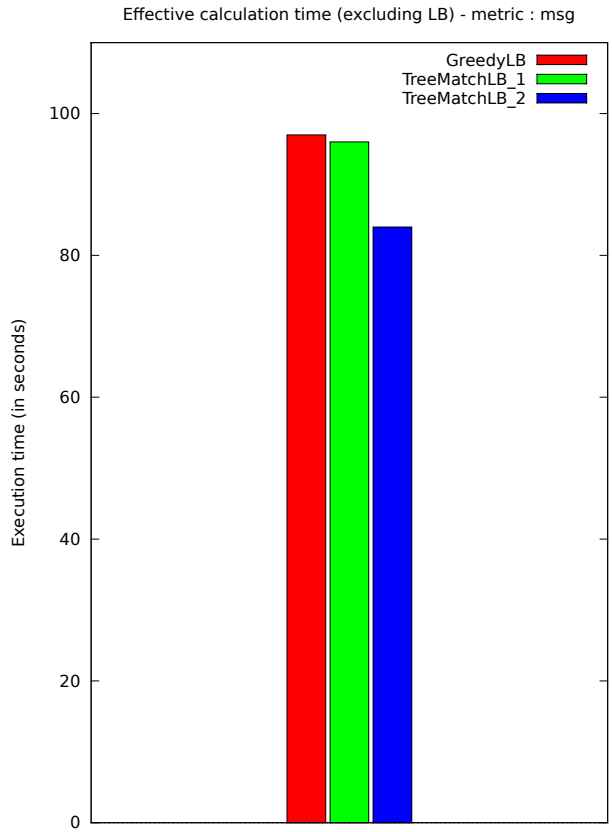




Stencil3D

- Large communications : lots of messages between processes
- *charmrun +p16 stencil3d 200 10 +balancer TreeMatchLB +LBDebug 1 +setcpuaffinity +pemap 0-7*





Conclusion

... and future works

- Topology is not flat
- Processes affinities are not uniform
- Take into account these informations to map chares could be interesting
- Adapt our algorithm to large problems (20K chares for example)
- Here at Urbana to work on these problems with the Charm++ team

The End

Thanks for your attention !
Any questions?