Towards achieving GPU-native adaptive mesh refinement

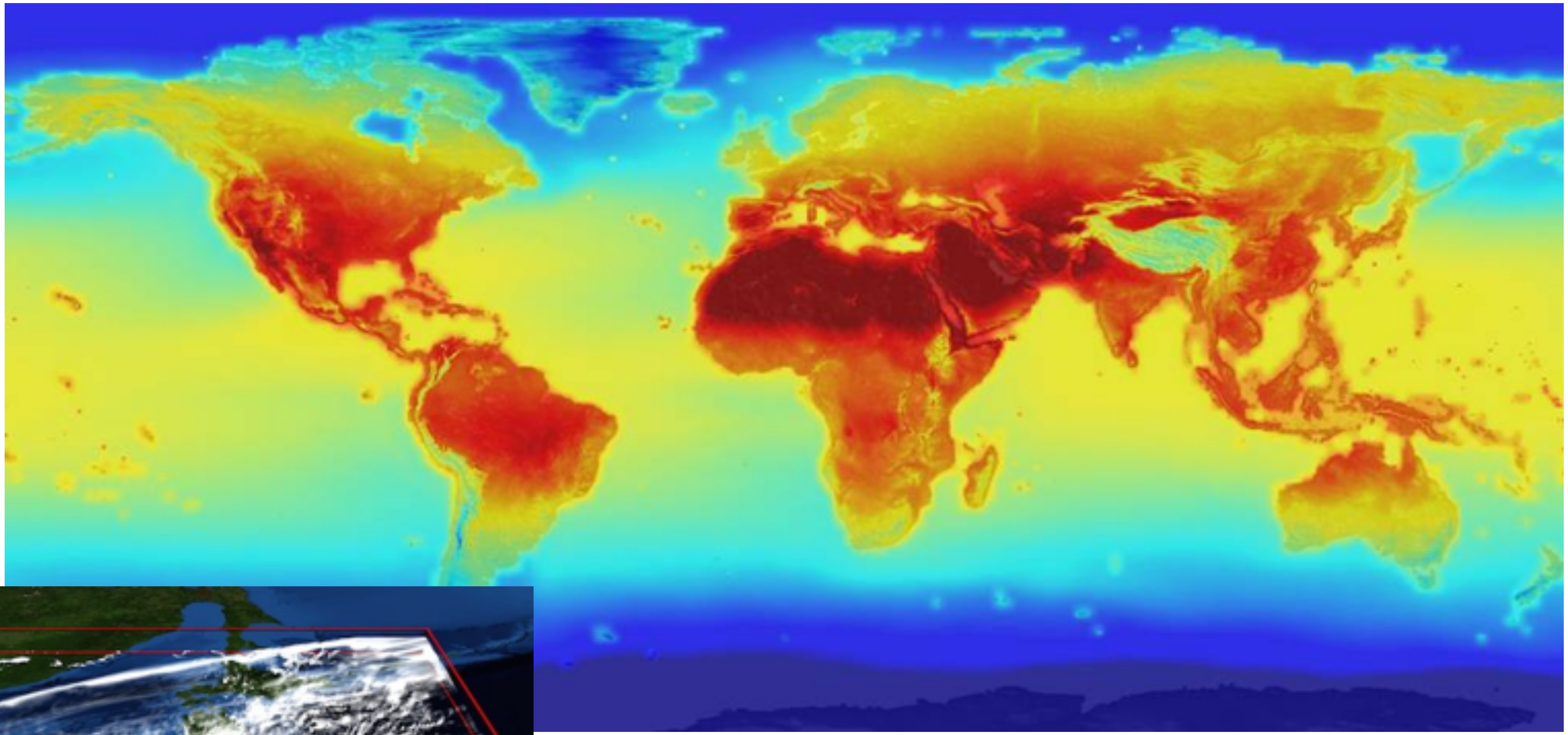
Ania Brown

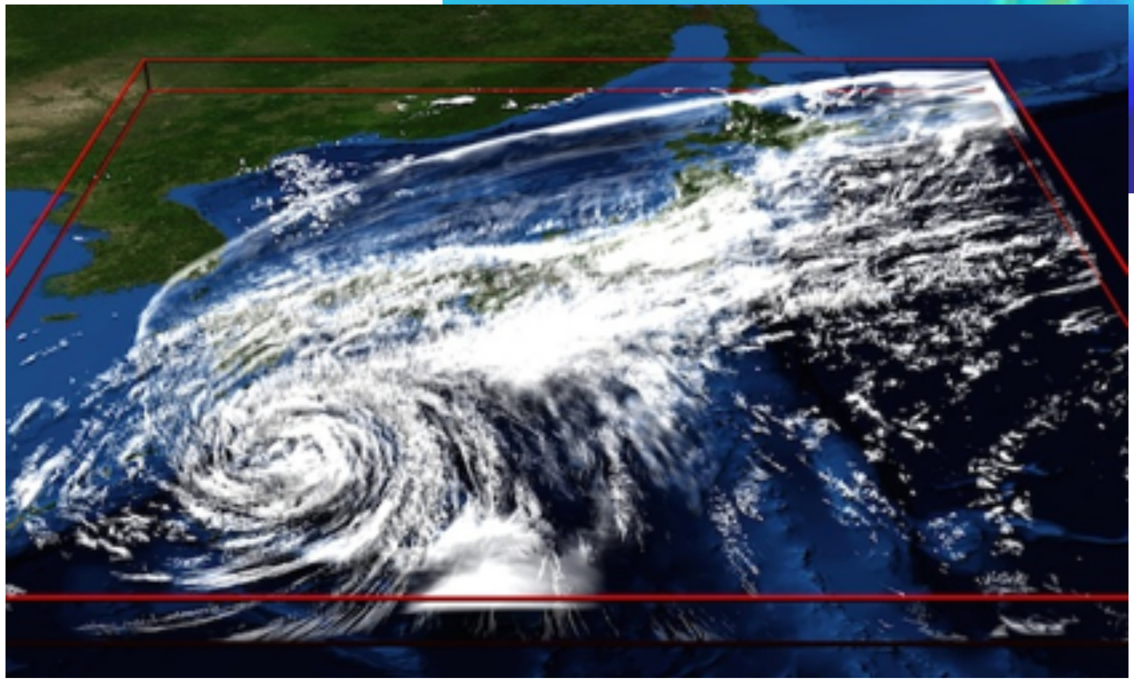
東京工業大学
Tokyo Institute of Technology

Prof Takayuki Aoki

Oxford e-Research Centre

Why AMR?



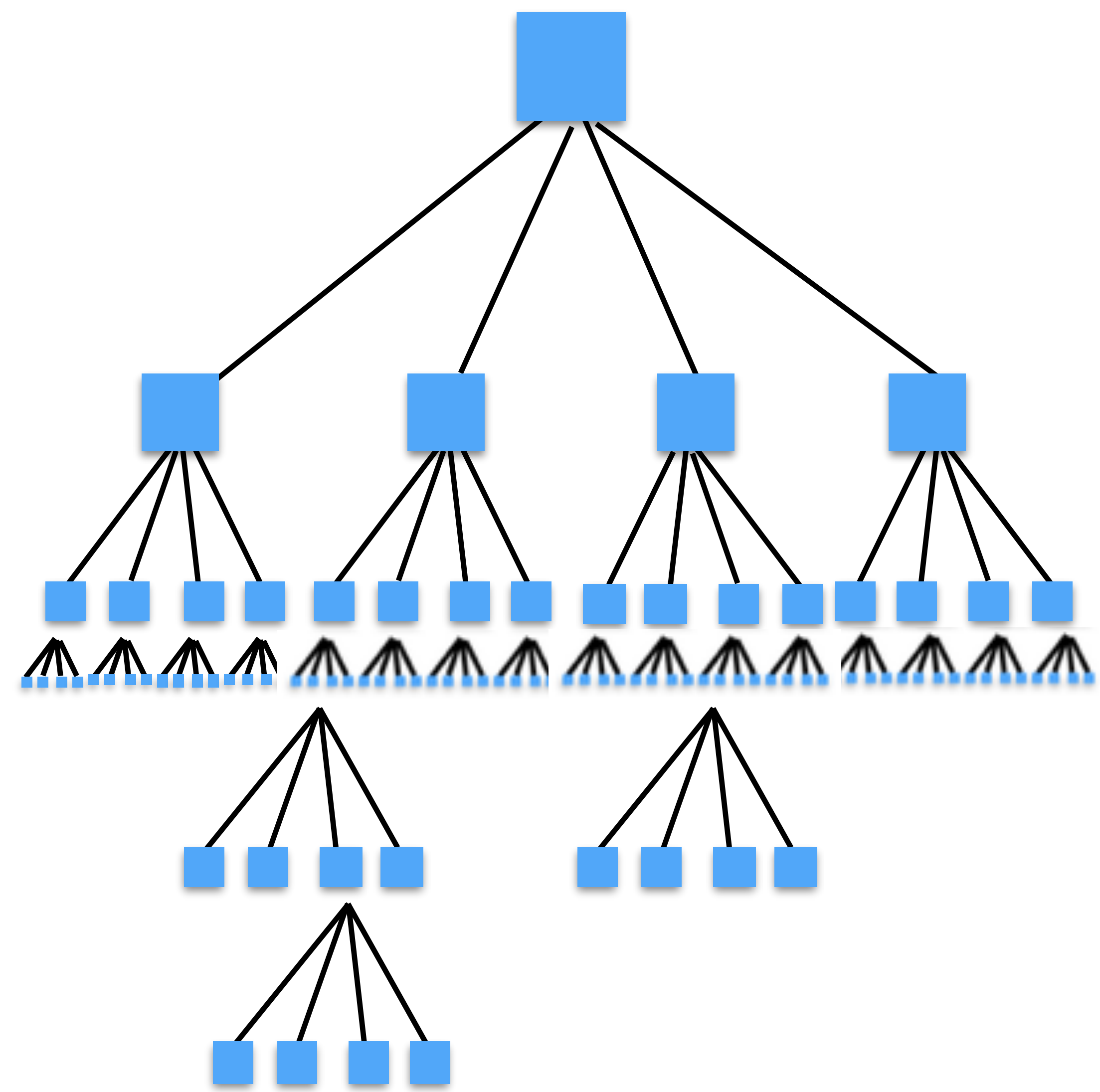


AMR is not GPU friendly

- Complicated, time varying data structures

- Can you use AMR and keep GPU performance?

  My conclusion: yes, but it's messy

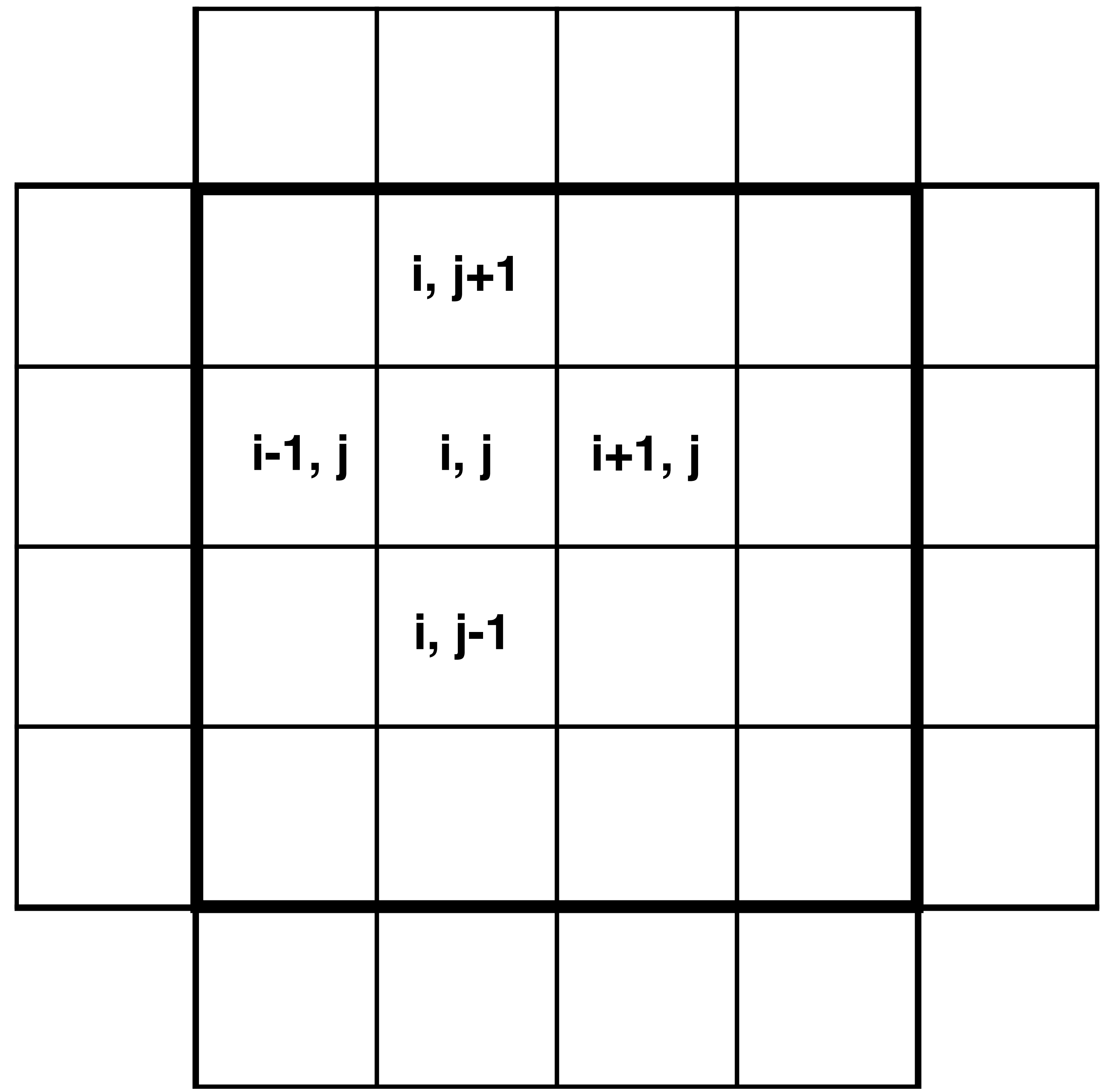


Contents

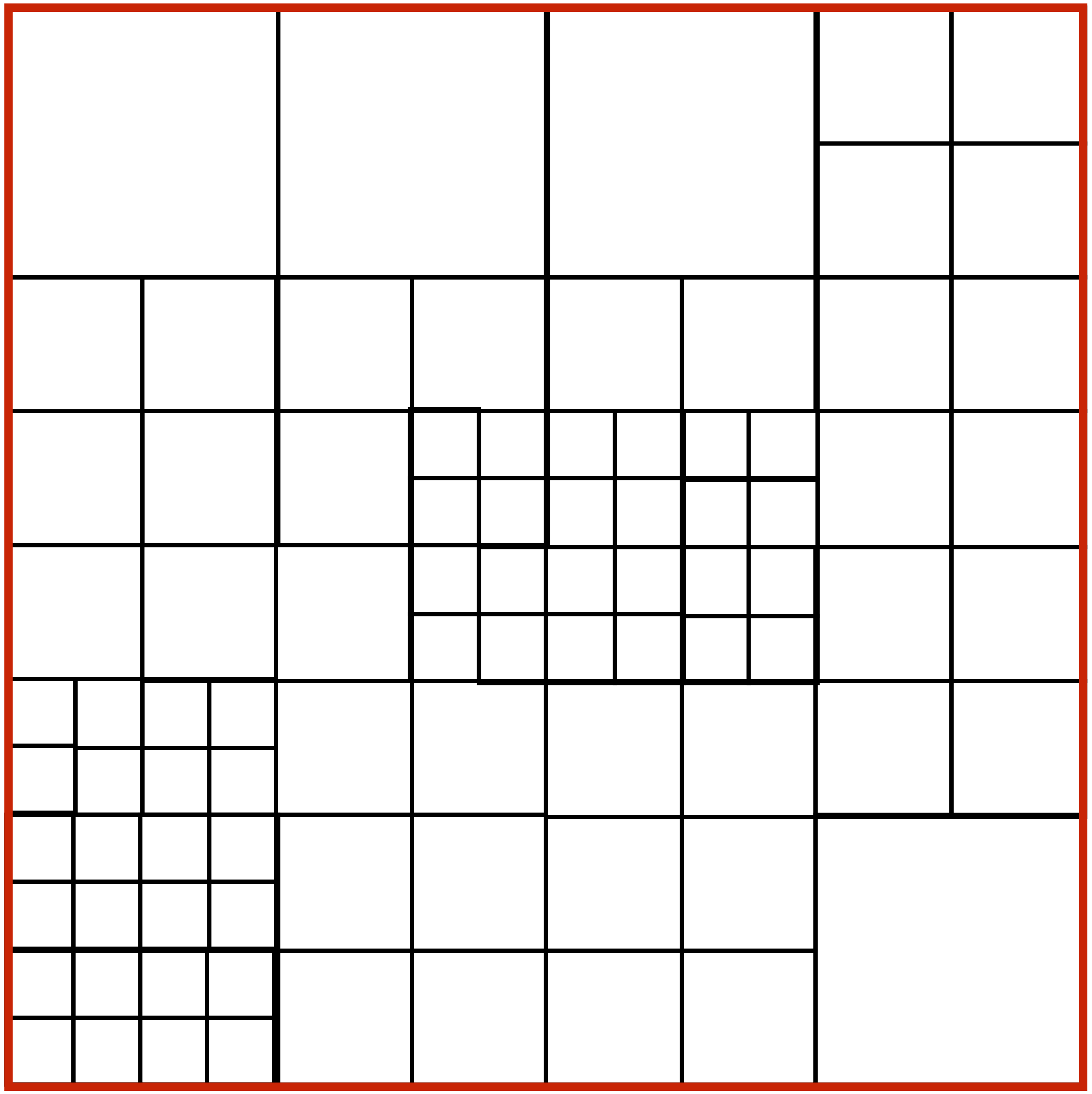
- Introduction to the algorithm + data structures
- The challenges
- Optimisation possibilities
- The RSE perspective — some lessons learnt

Problem domain

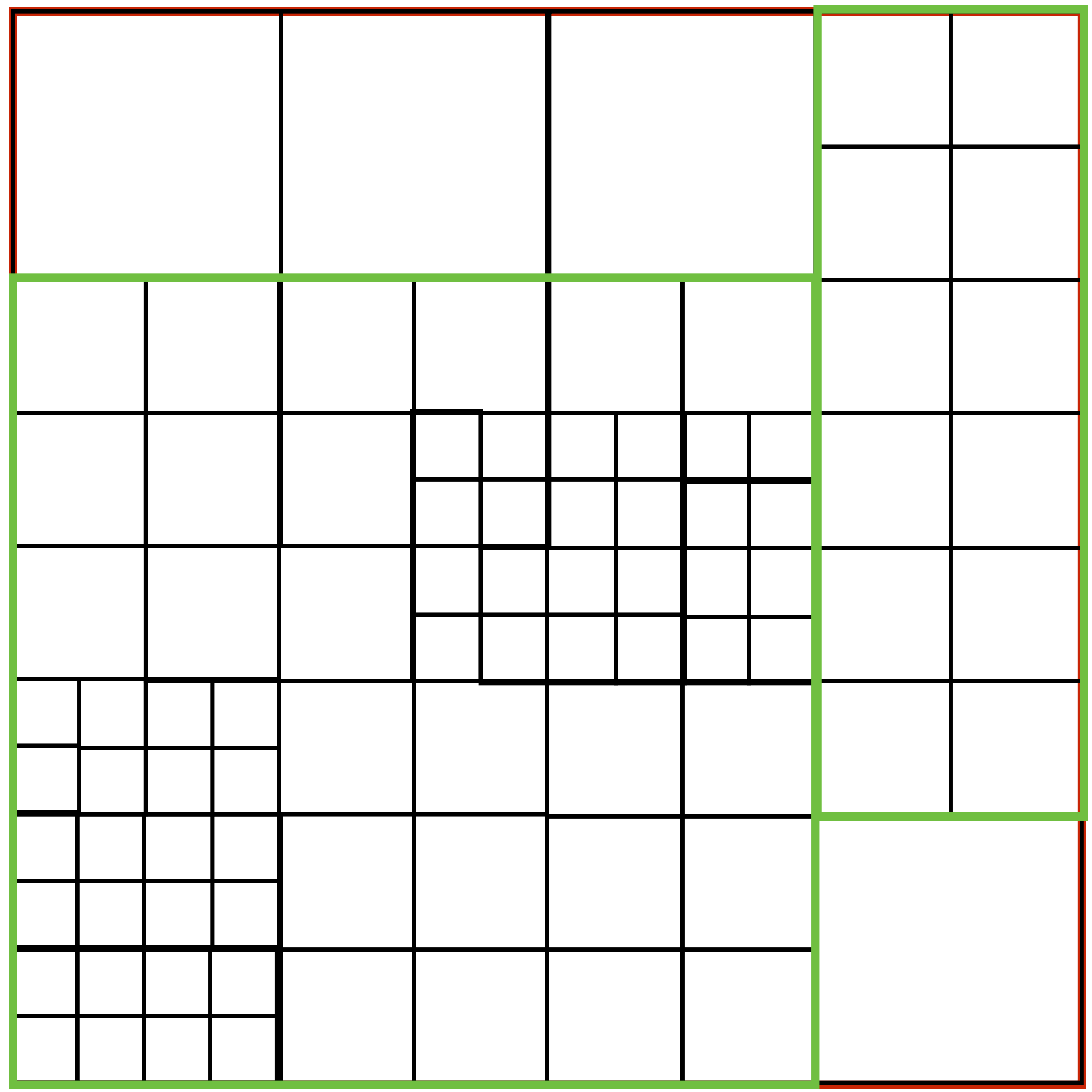
- Stencil calculations on a square structured mesh
- Cell centre values



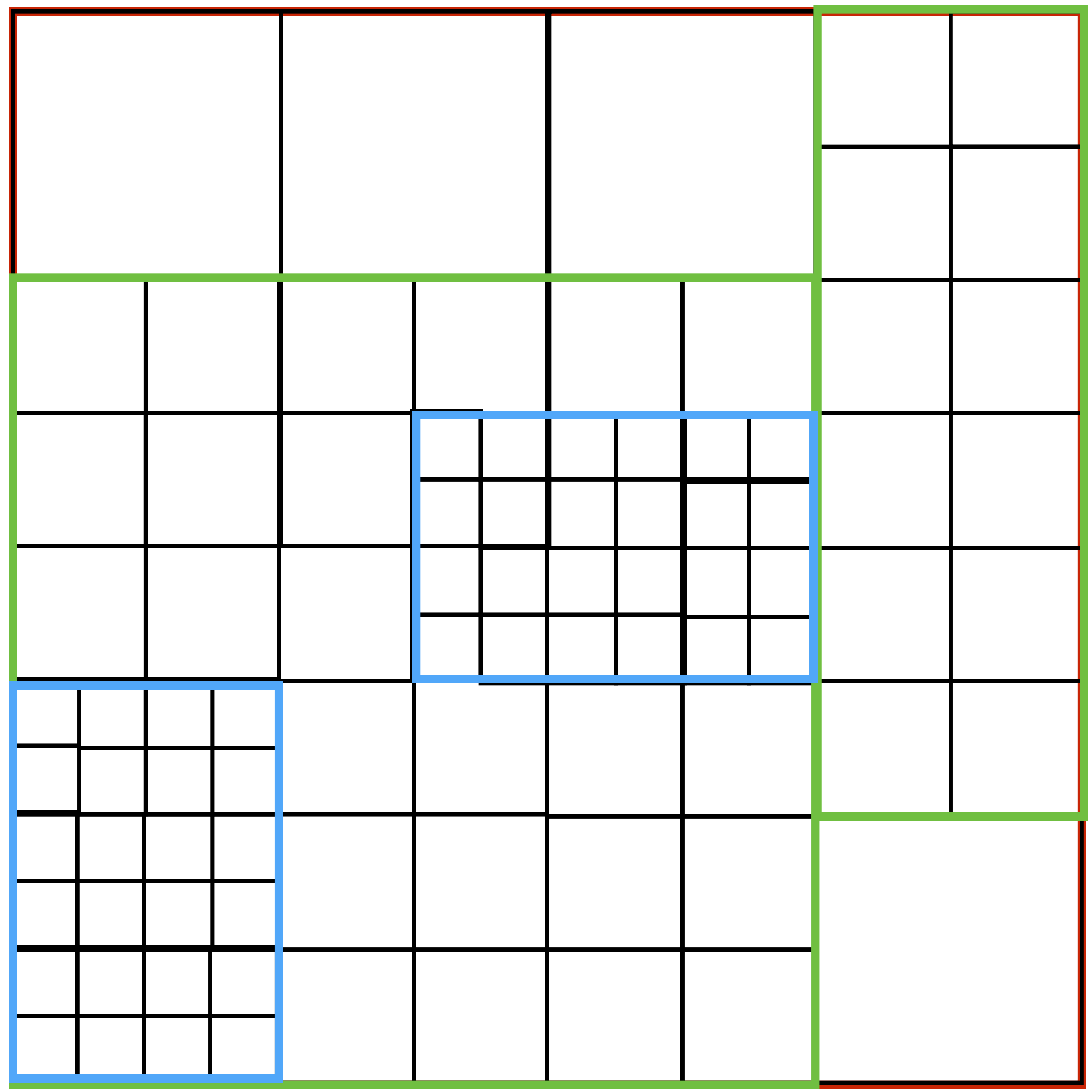
1) Block structured AMR



1) Block structured AMR

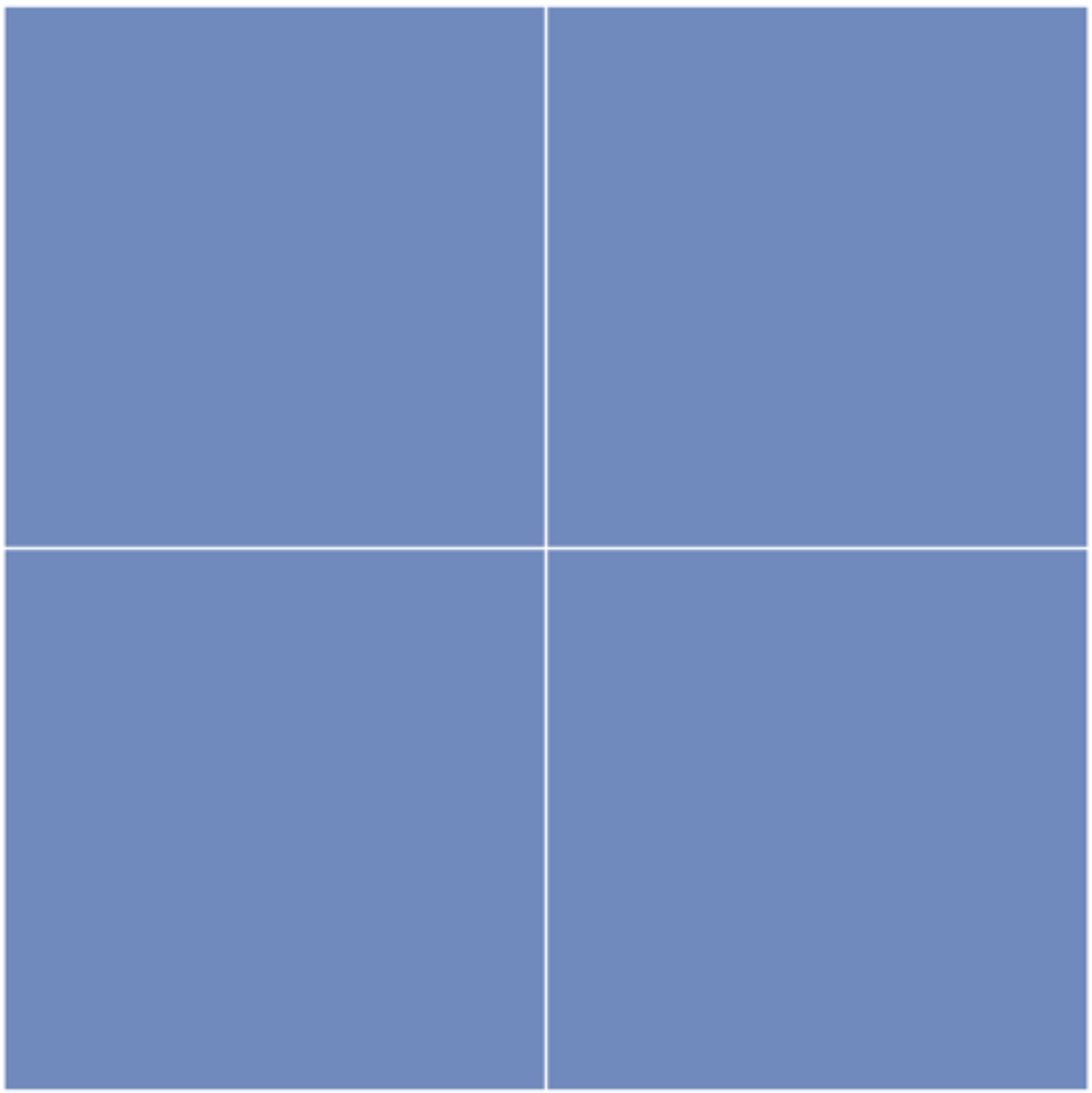


1) Block structured AMR

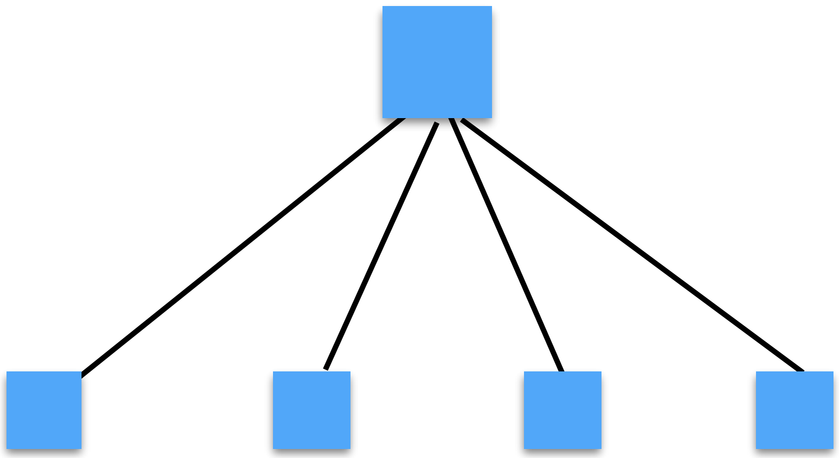


2) Tree based AMR

Simulation mesh

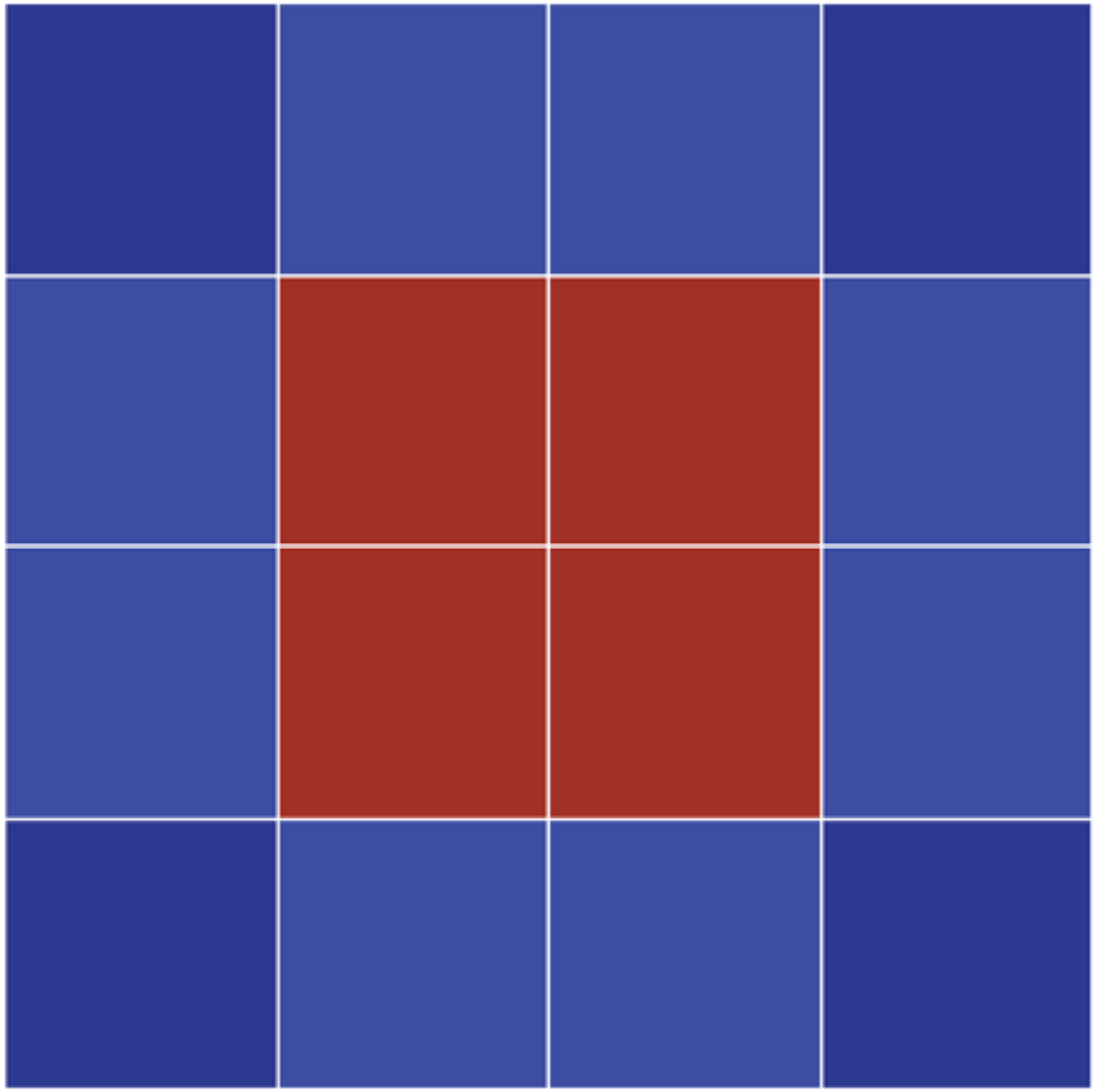


Refinement representation

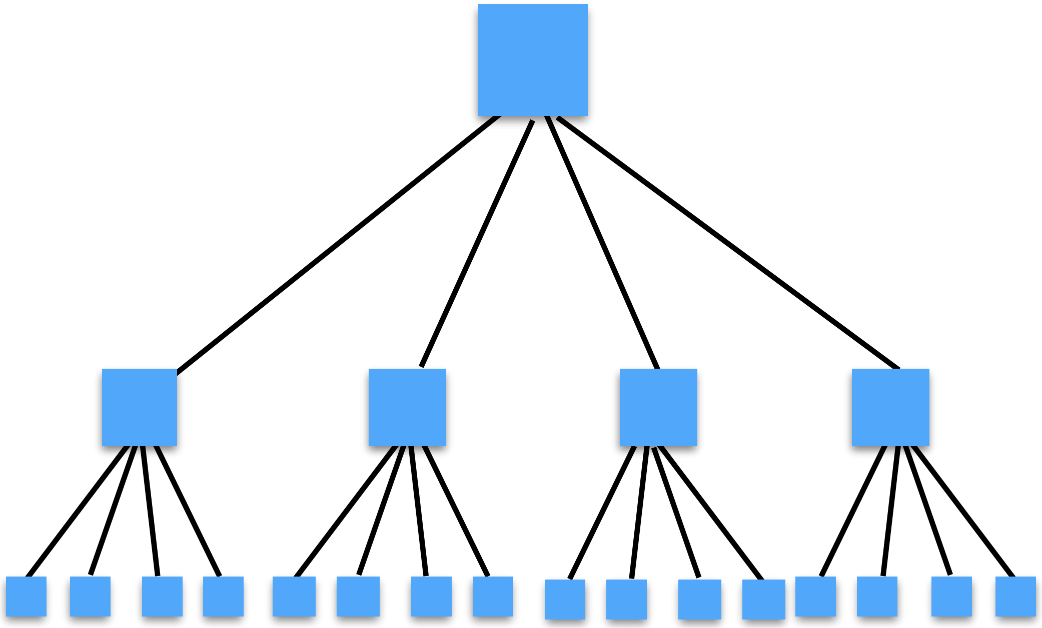


2) Tree based AMR

Simulation mesh

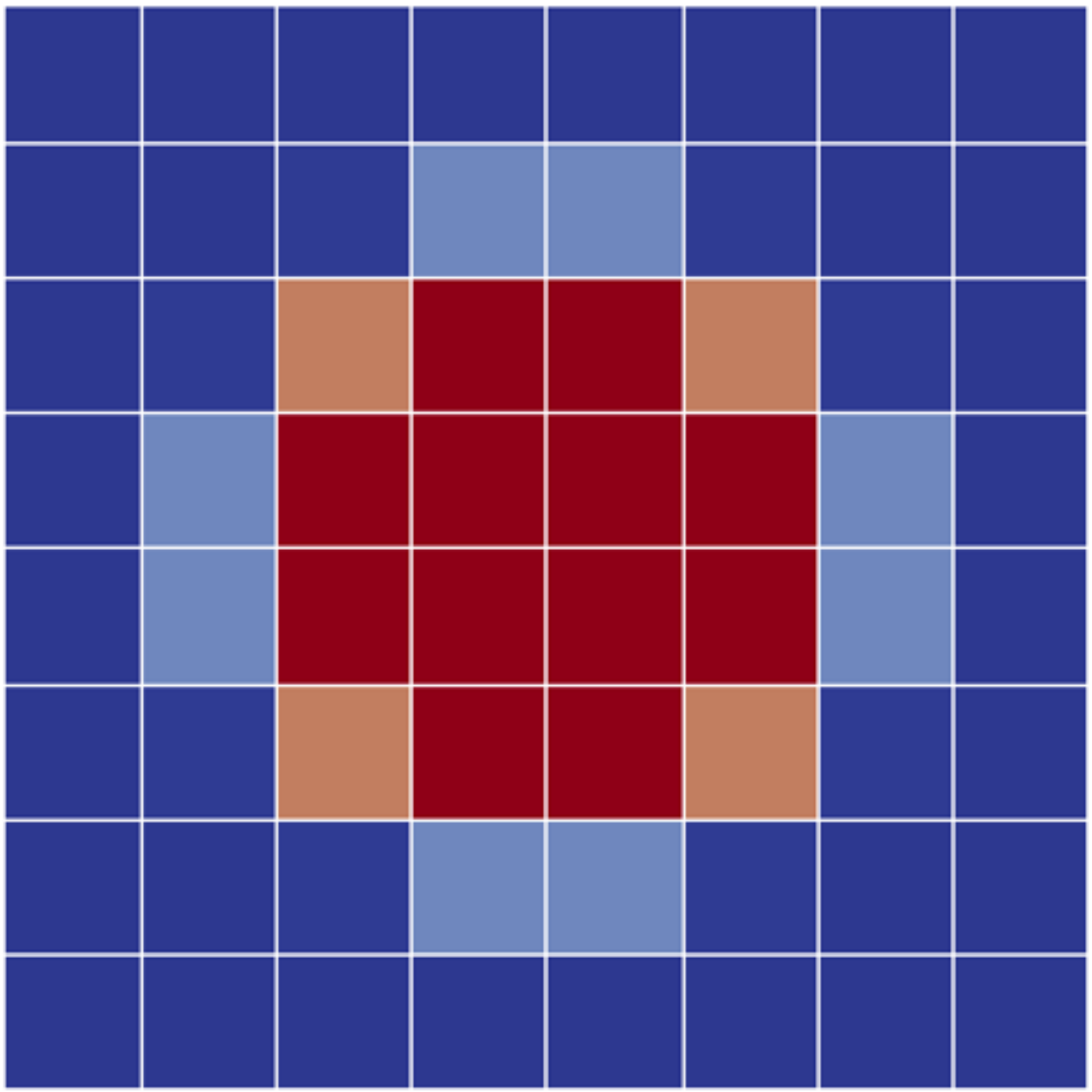


Refinement representation

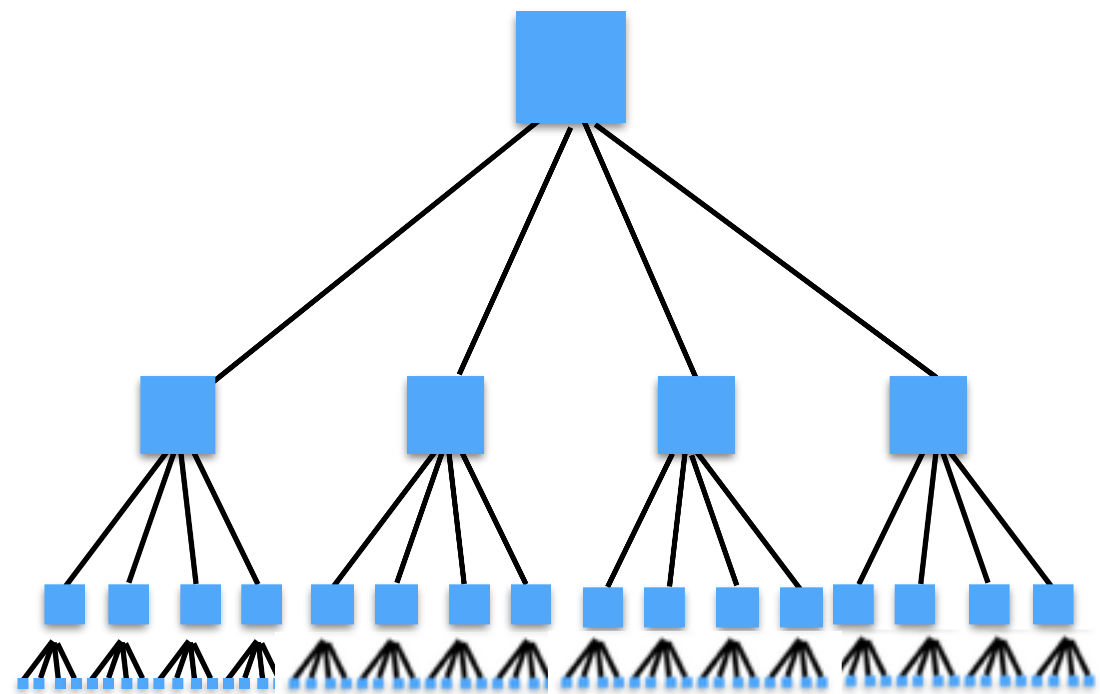


2) Tree based AMR

Simulation mesh

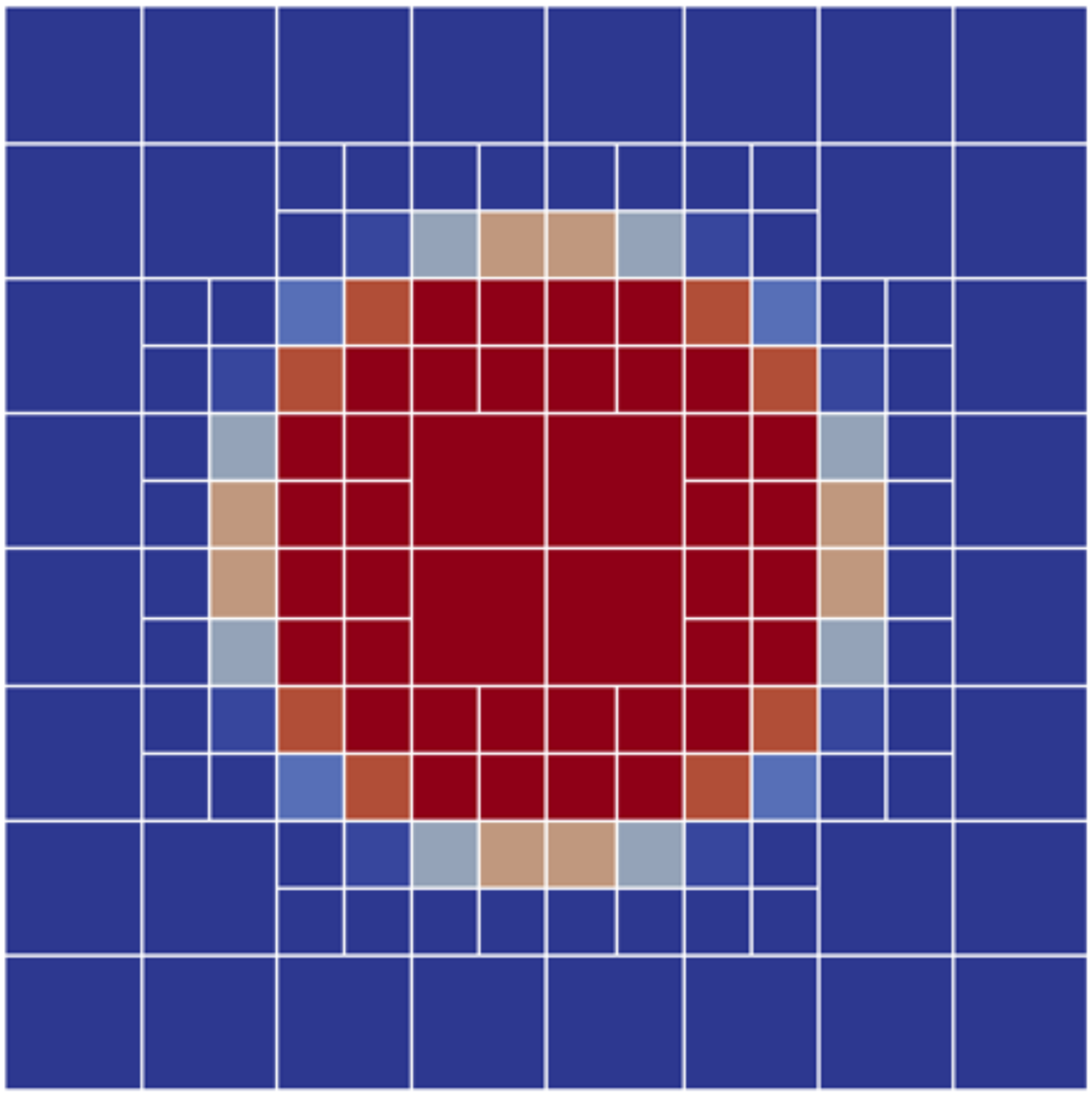


Refinement representation

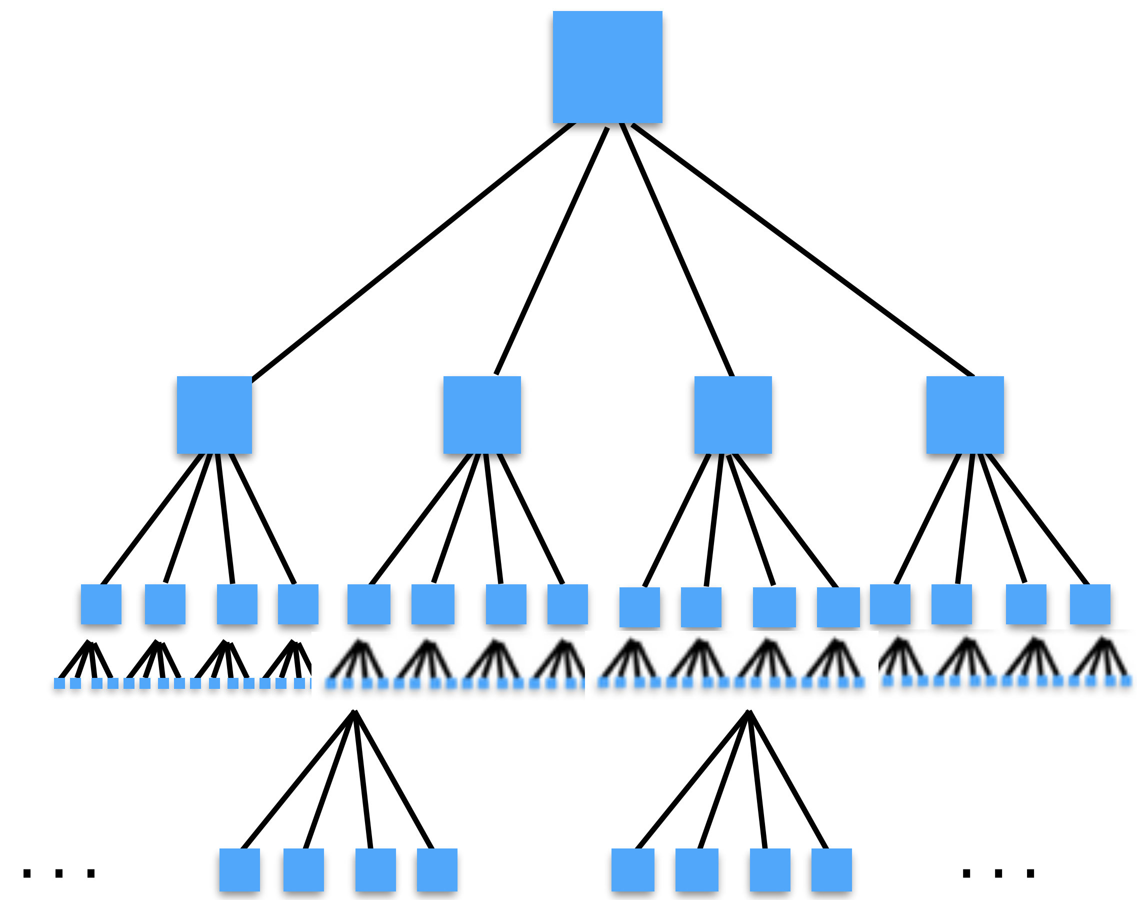


2) Tree based AMR

Simulation mesh

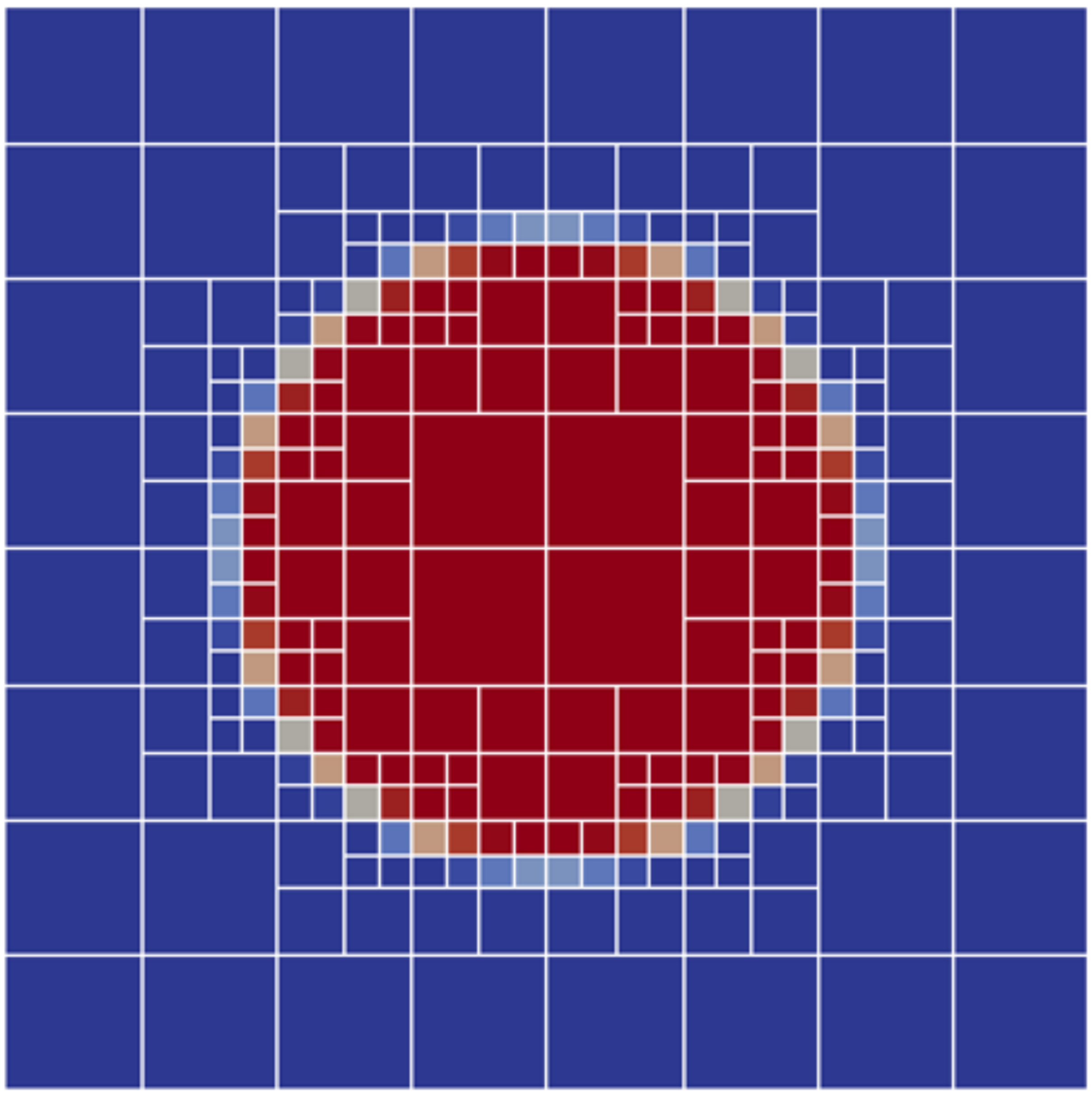


Refinement representation

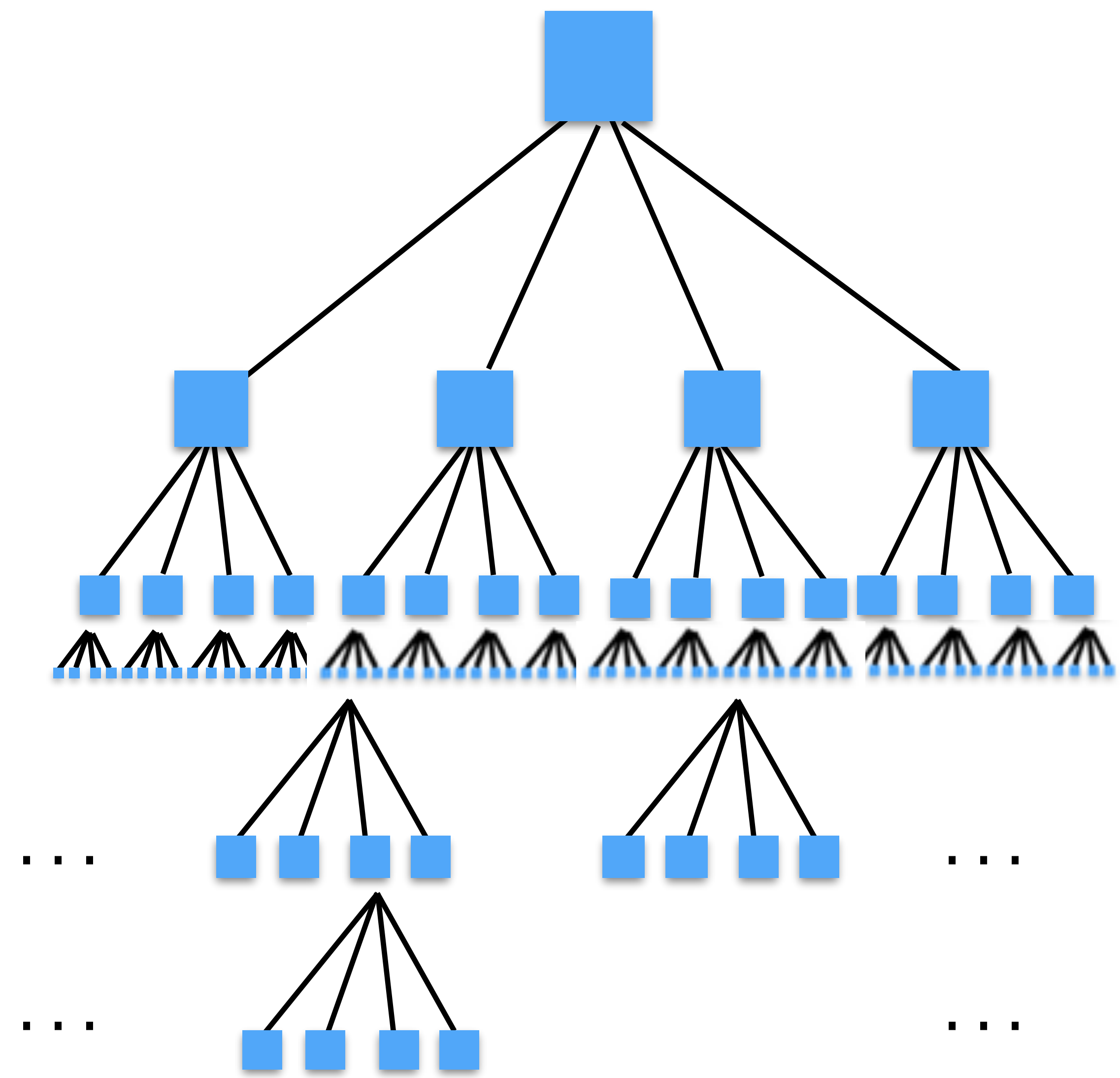


2) Tree based AMR

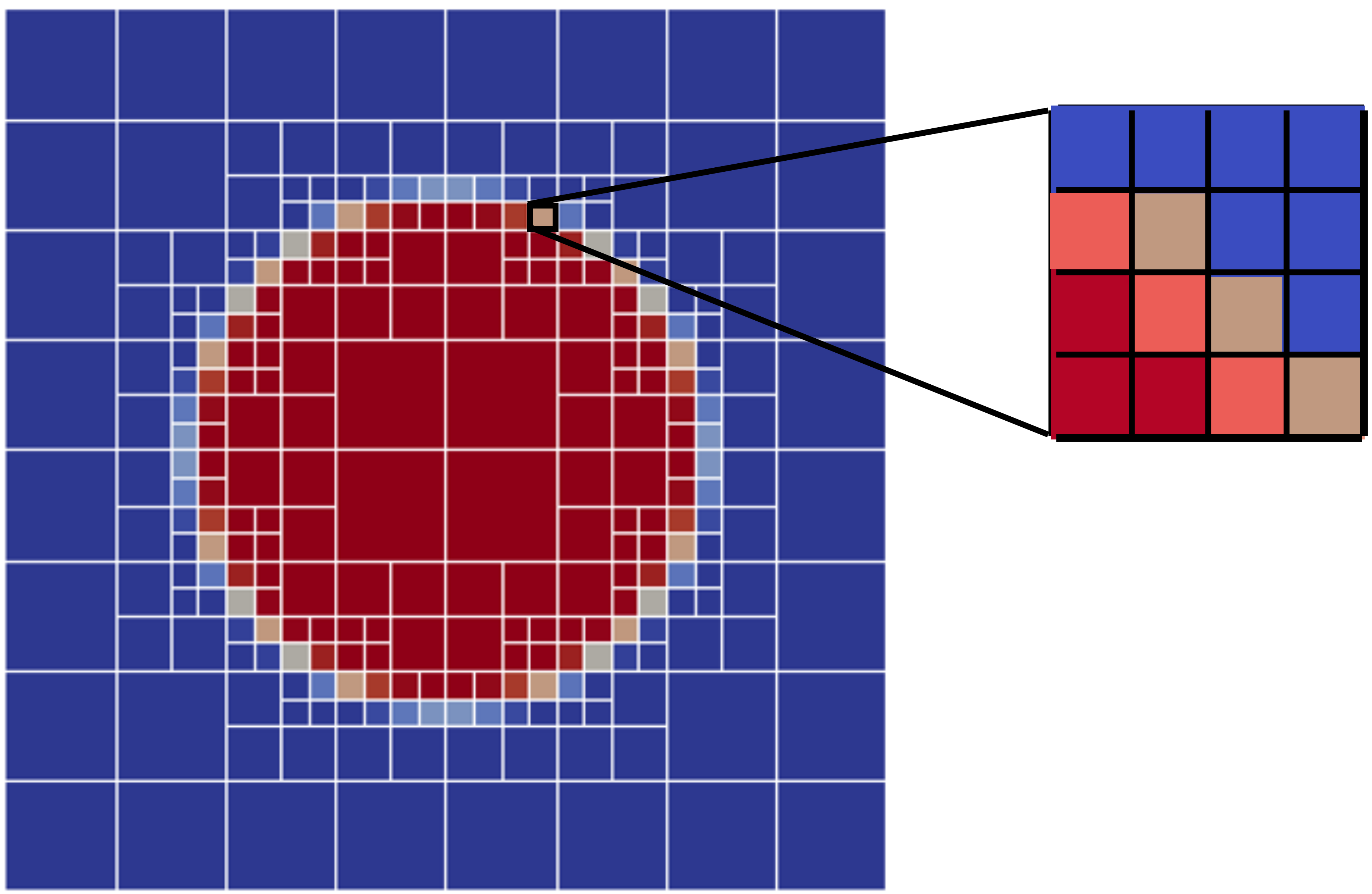
Simulation mesh



Refinement representation



3) Patches +Tree AMR



CPU	**GPU**
Block-structured	**Octree + patches**
Enzo	GAMER (2011)
CHOMBO	Daino (2016)
Octree	
RAMSES	
Octree + patches	
FLASH, using PARAMESH	
NIRVANA	

The AMR algorithm

Initialize data structures

Main loop:

Create halo regions

Update patch values

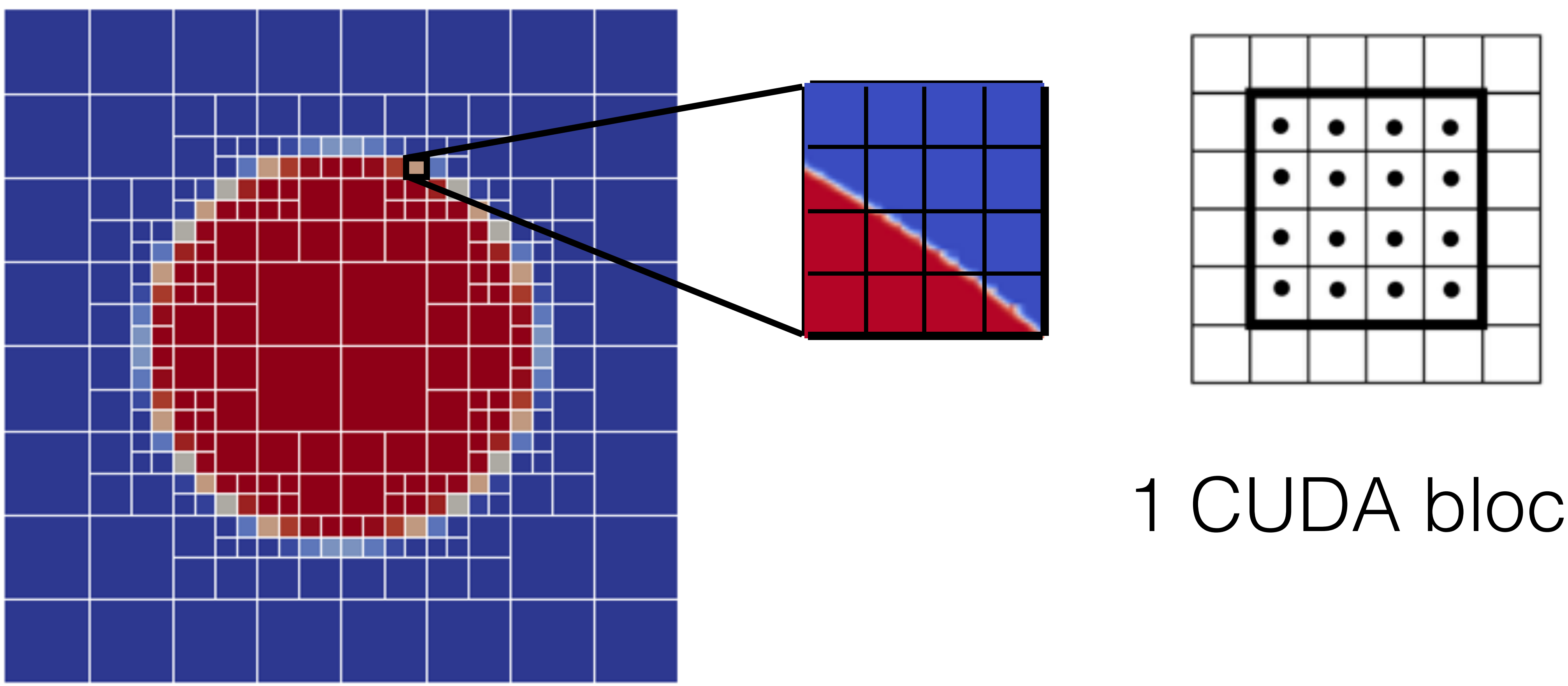
Refine/coarsen patches

Update neighbour relations

Output values for visualisation

CPU	GPU
Initialize data structures	
Main loop:	
Create halo regions	
	Update patch values
Refine/coarsen patches	
Update neighbour relations	
Output values for visualisation	

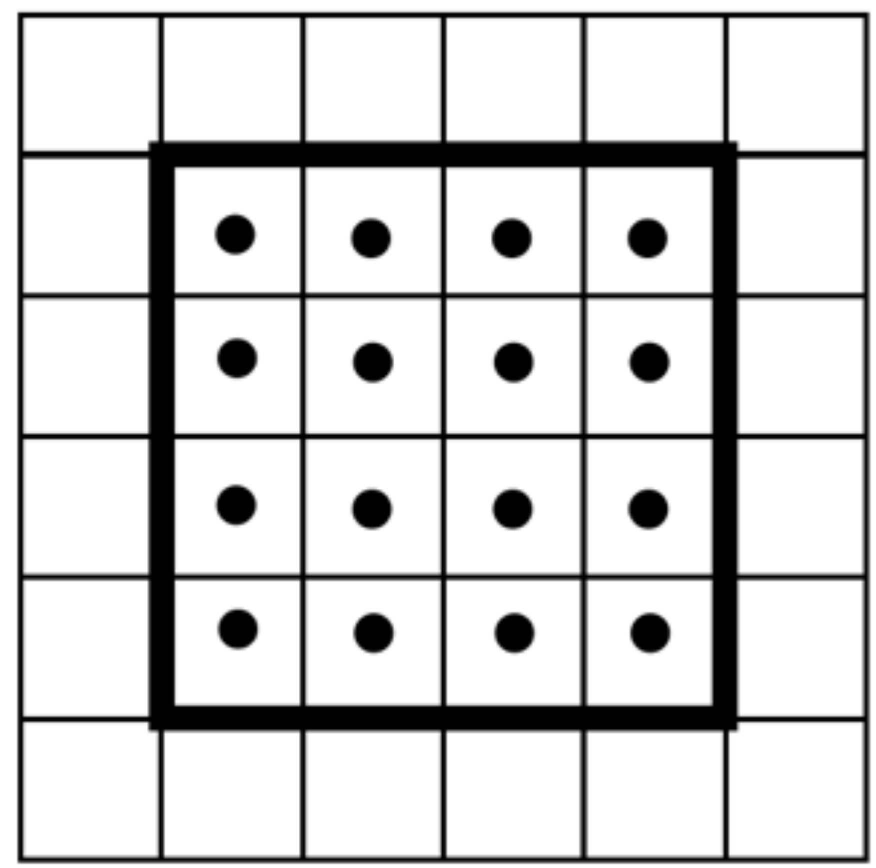
Update step

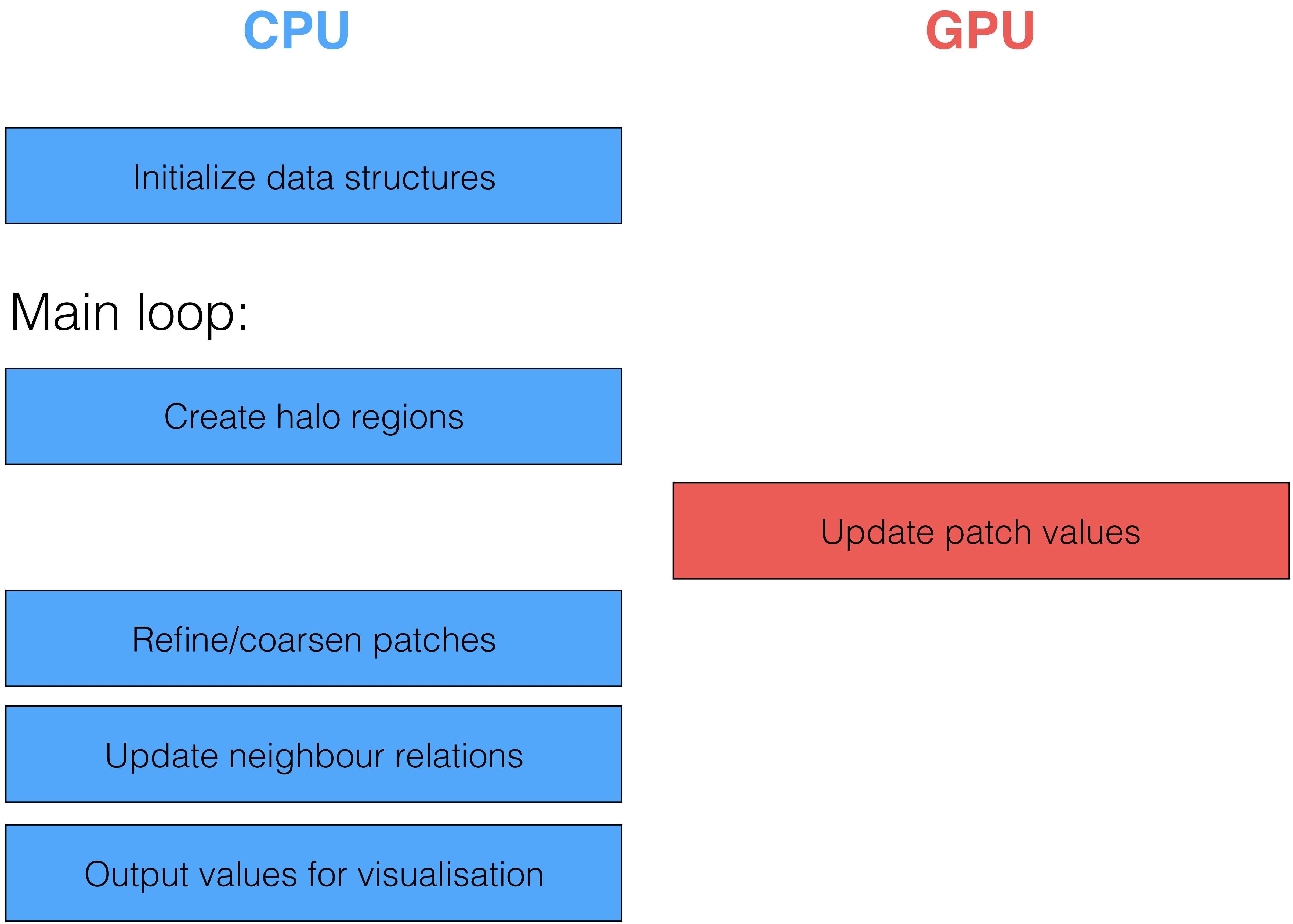


1 CUDA block

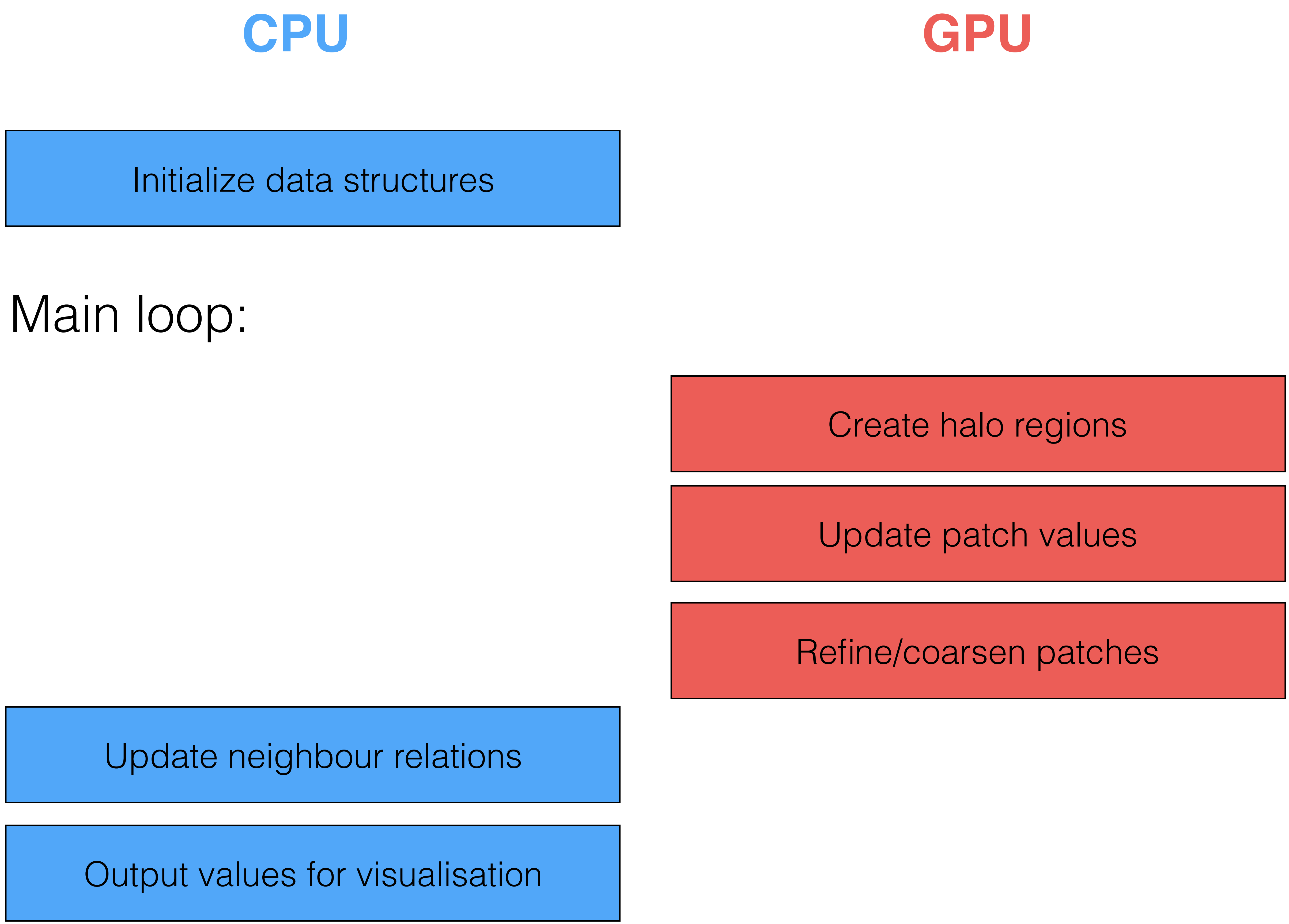
Update step

- Tune block size for coalesced access
- Zmarching

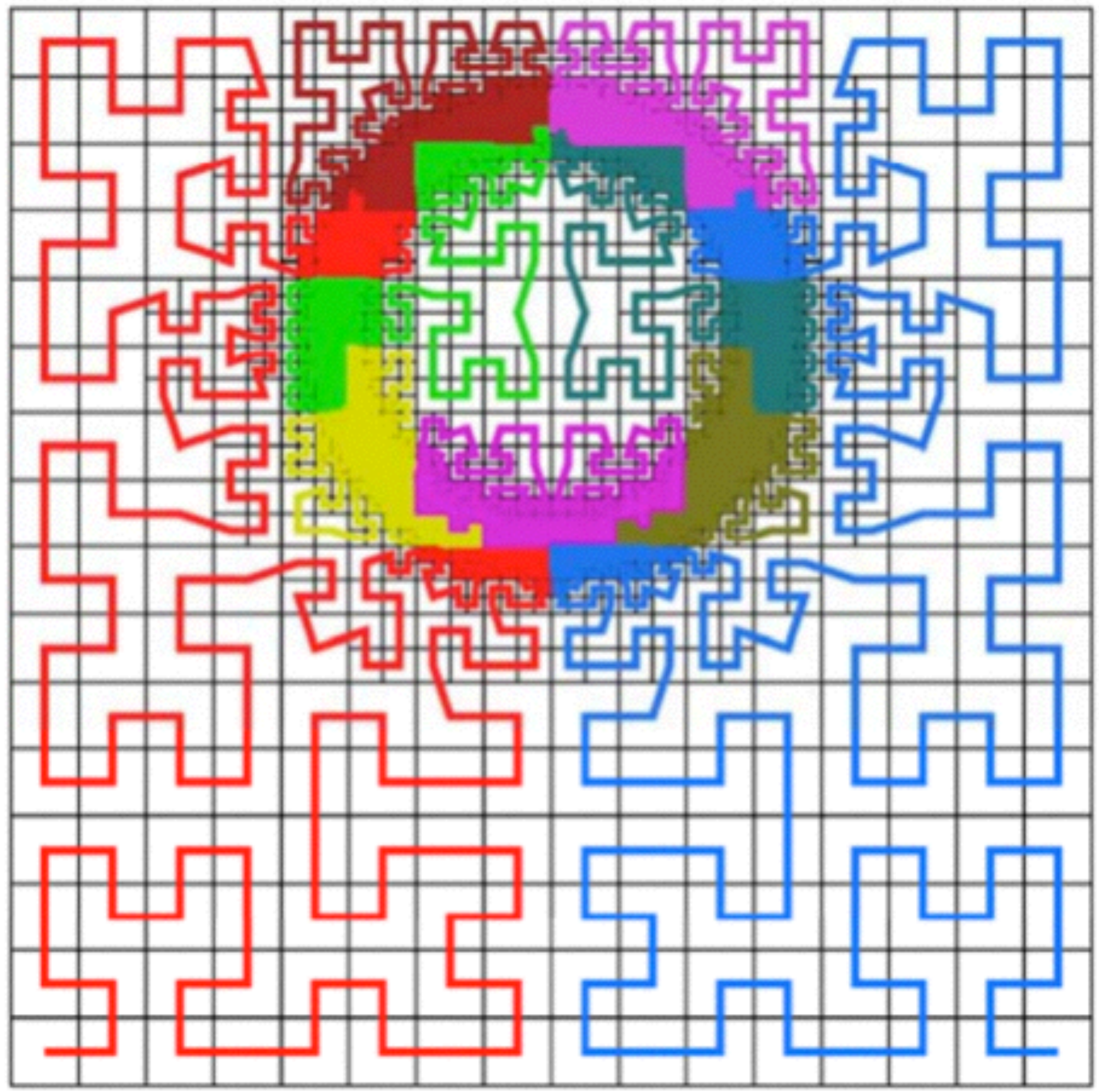




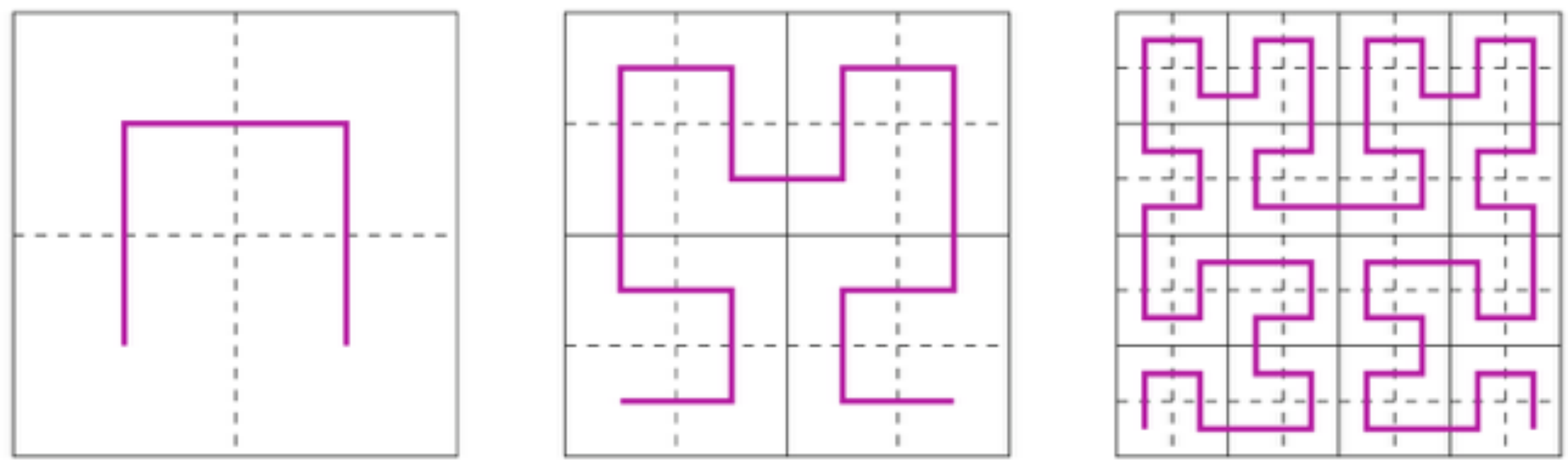
CPU
GPU
Initialize data structures
Main loop:
Create halo regions
Update patch values
Refine/coarsen patches
Update neighbour relations
Output values for visualisation


CPU
GPU
Initialize data structures
Main loop:
Create halo regions
Update patch values
Refine/coarsen patches
Update neighbour relations
Output values for visualisation

Ordering leaves



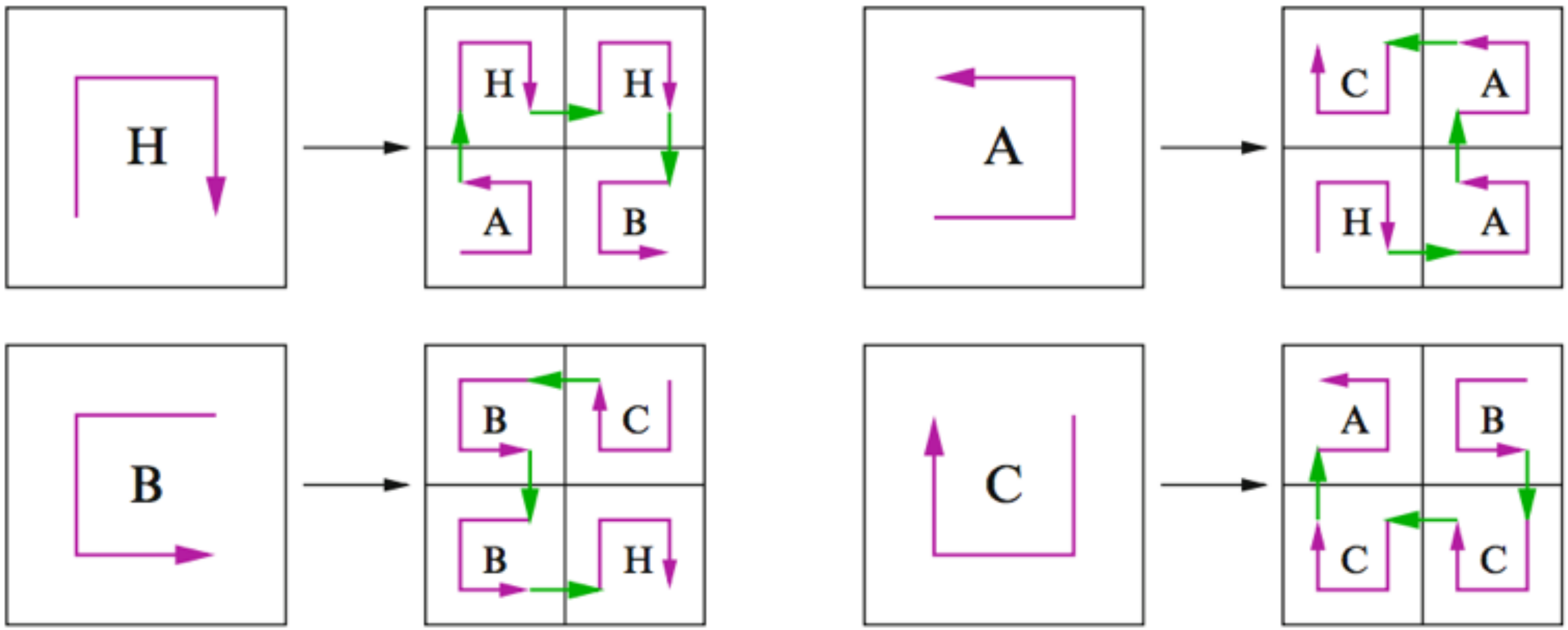
Hilbert curve



At each step:

- Divide space into 4
- Replace each quadrant with rotated or reflected versions of the original curve
- Connect such that that start and end points remain the same

Rules for refinement



$$H \longleftarrow A \uparrow H \rightarrow H \downarrow B$$

$$A \longleftarrow H \rightarrow A \uparrow A \leftarrow C$$

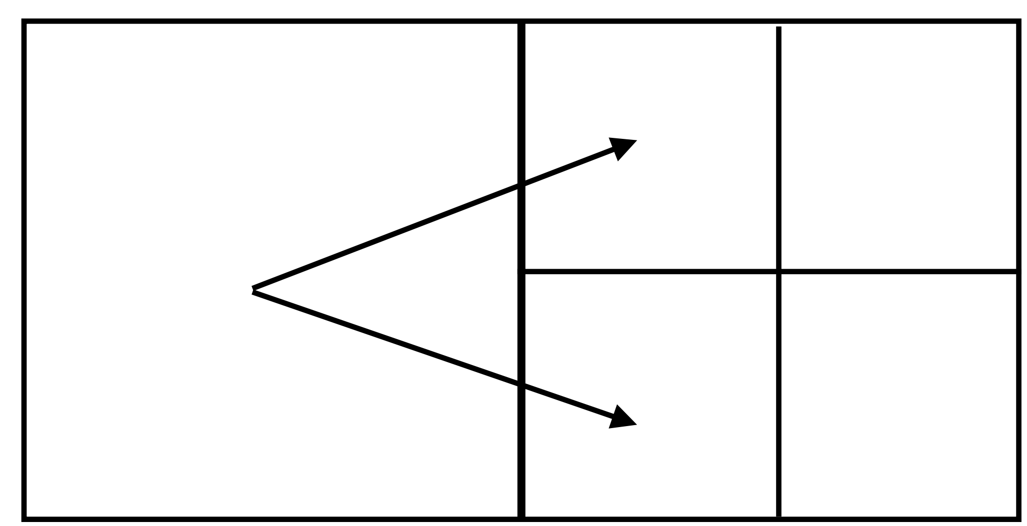
$$B \longleftarrow C \leftarrow B \downarrow B \rightarrow H$$

$$C \longleftarrow B \downarrow C \leftarrow C \uparrow A$$

Neighbour relations

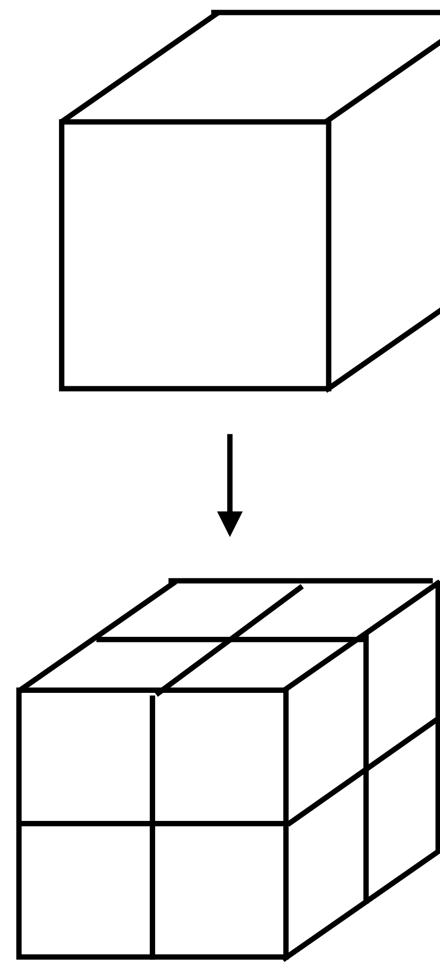
Leaf nodes:
Neighbour indices in each direction
Parent index

Parent nodes:
Child indices



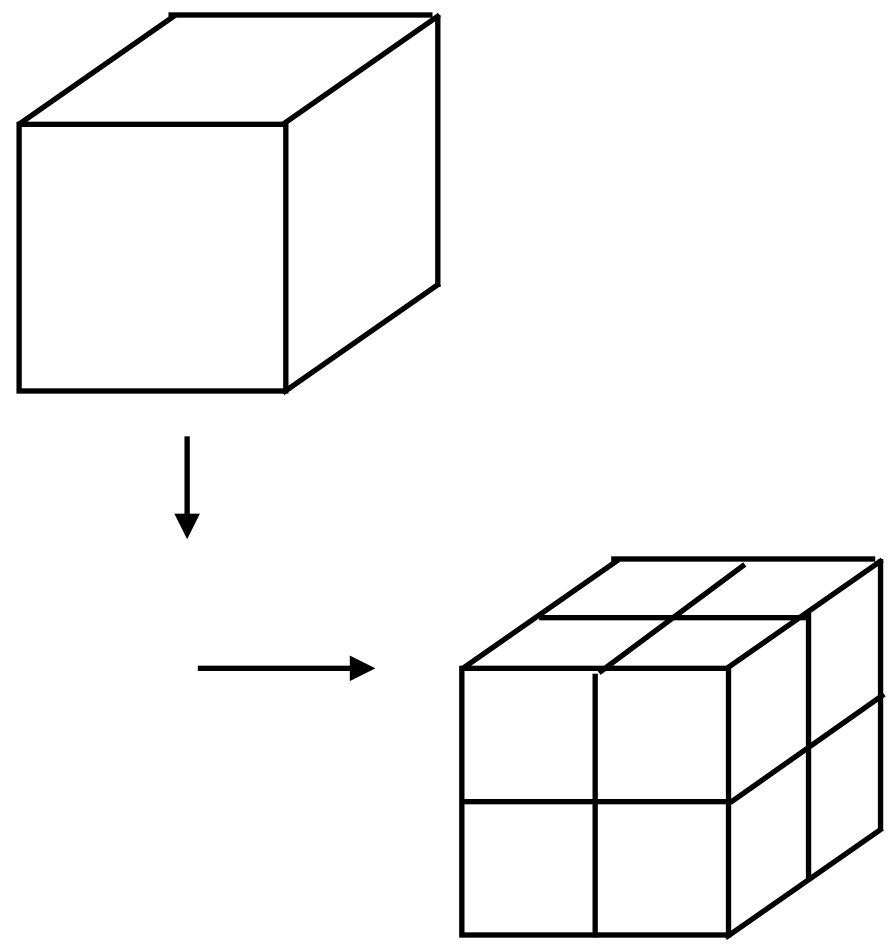
Create halo regions

Find correct neighbour node



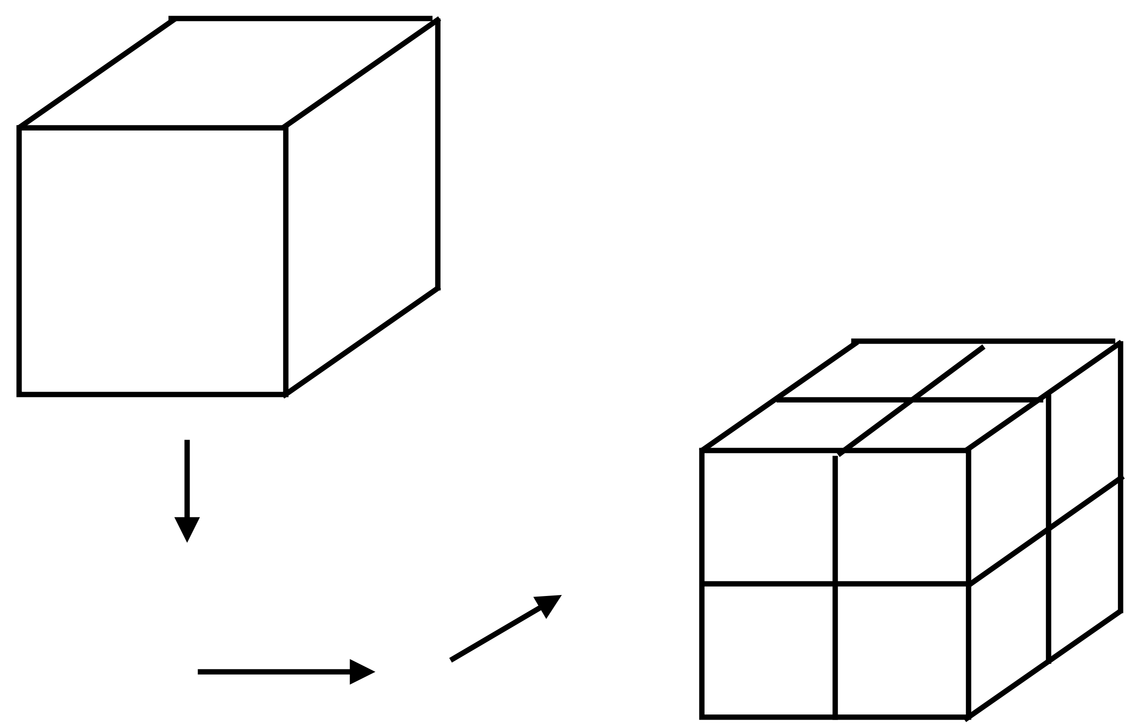
Create halo regions

Find correct neighbour node



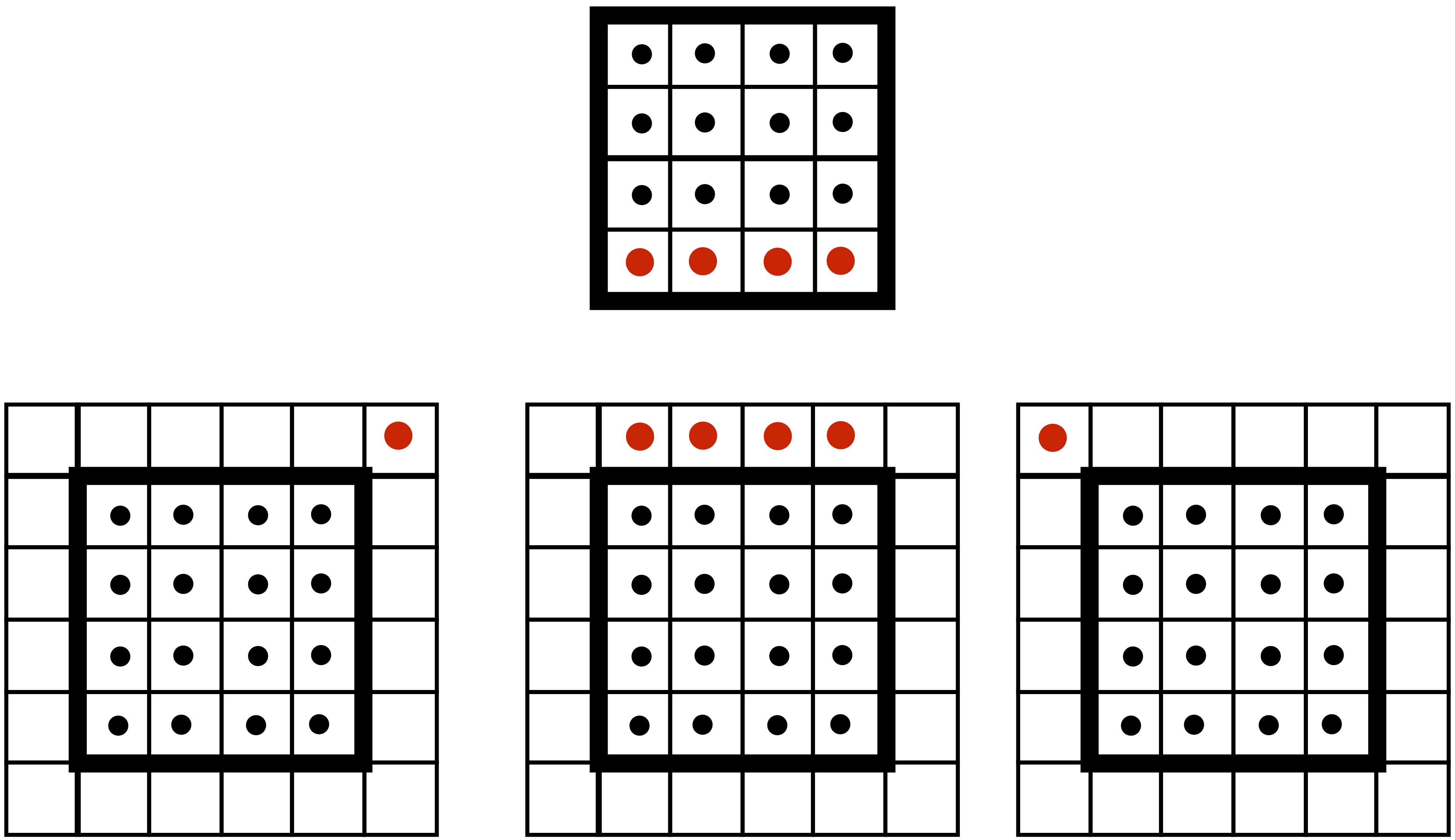
Create halo regions

Find correct neighbour node

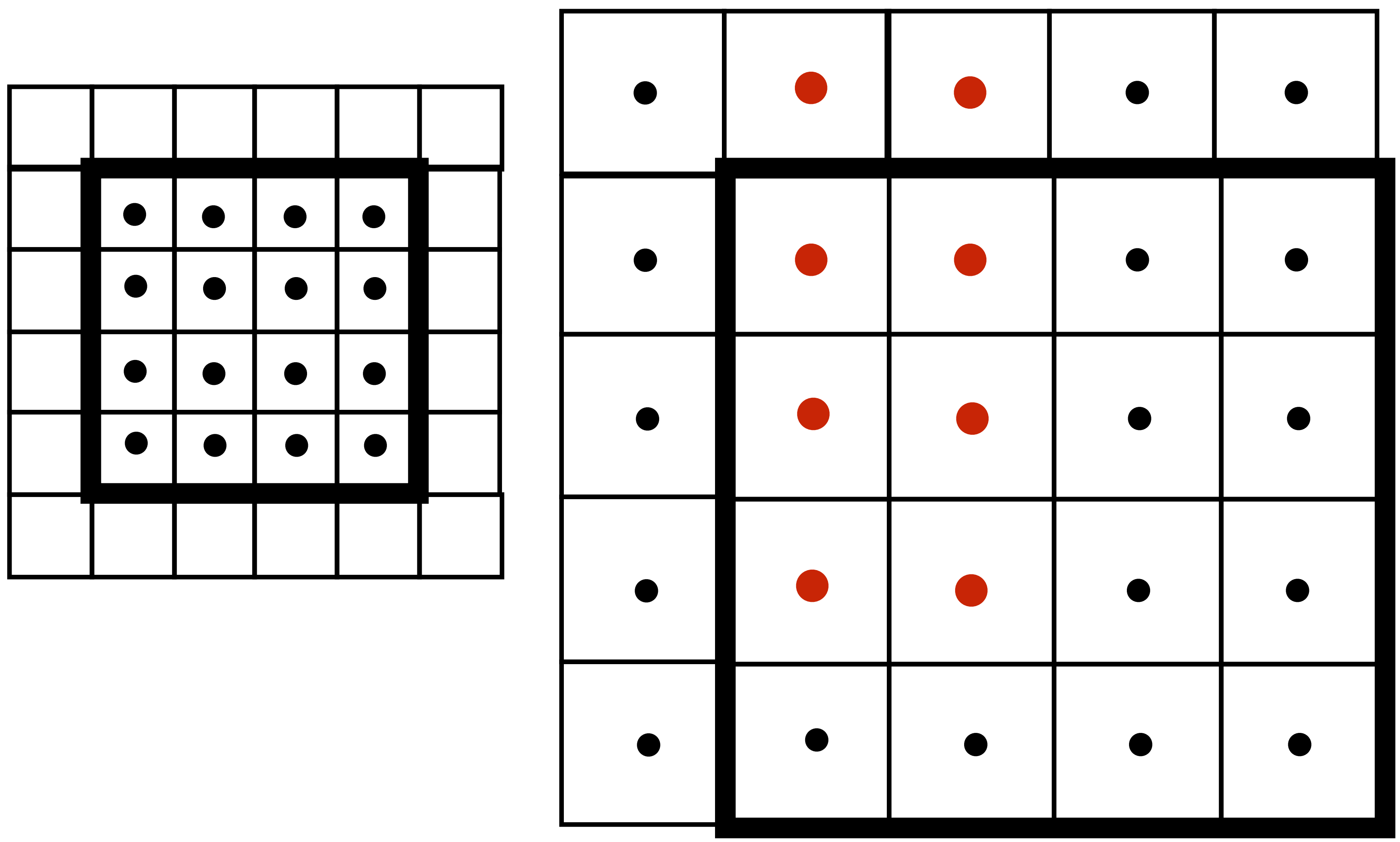


Create halo regions

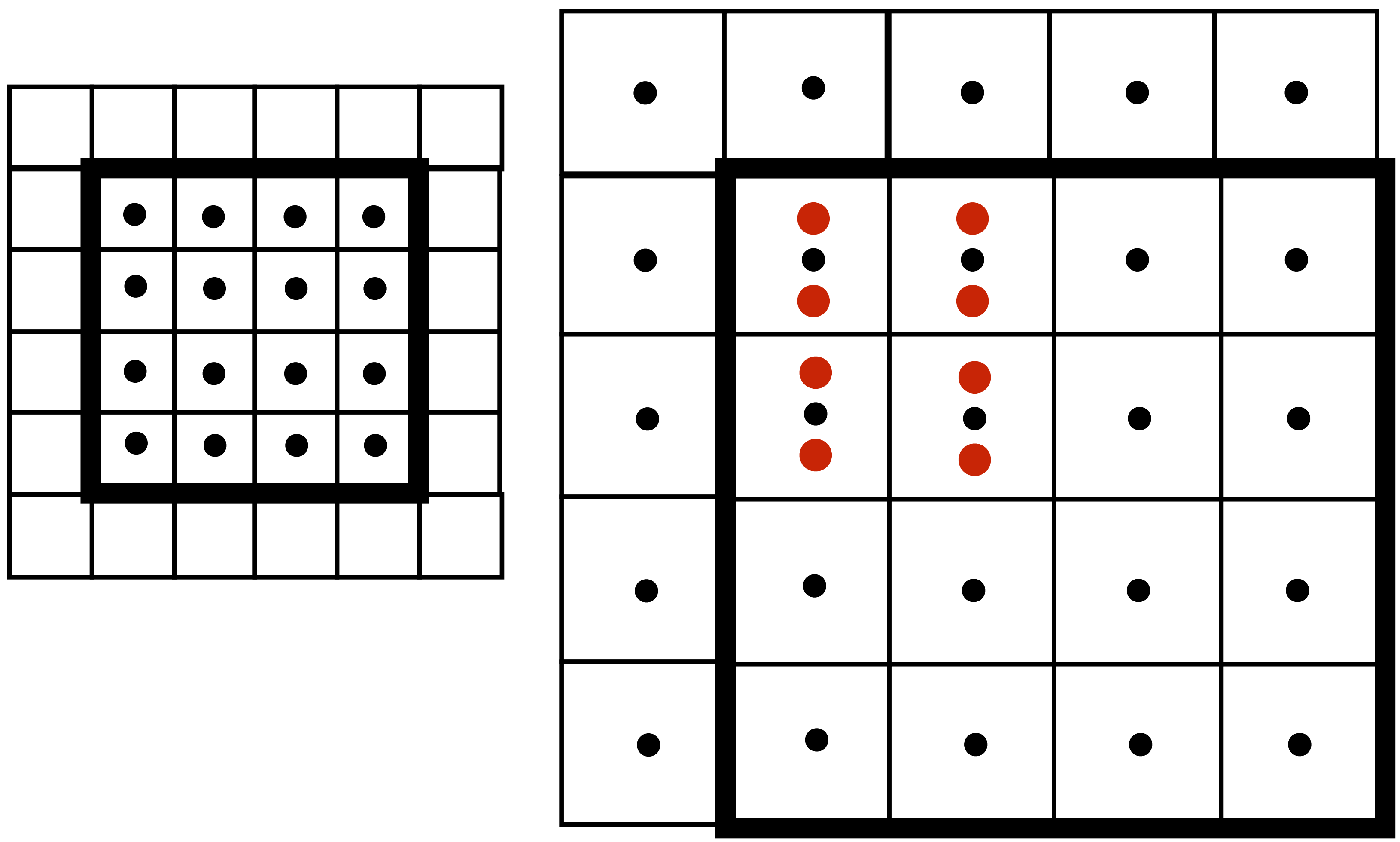
Copy halo values



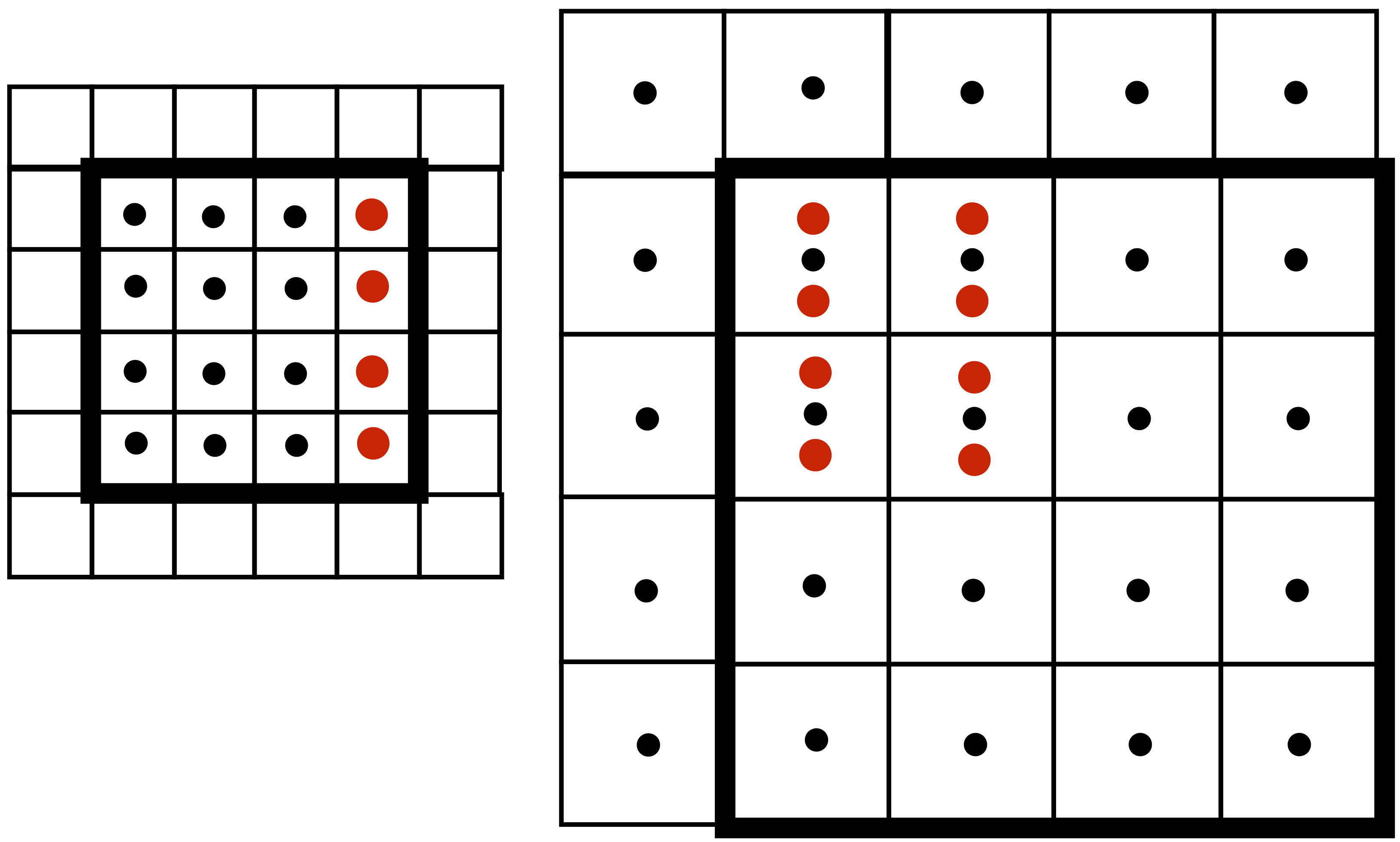
Interpolating halo values



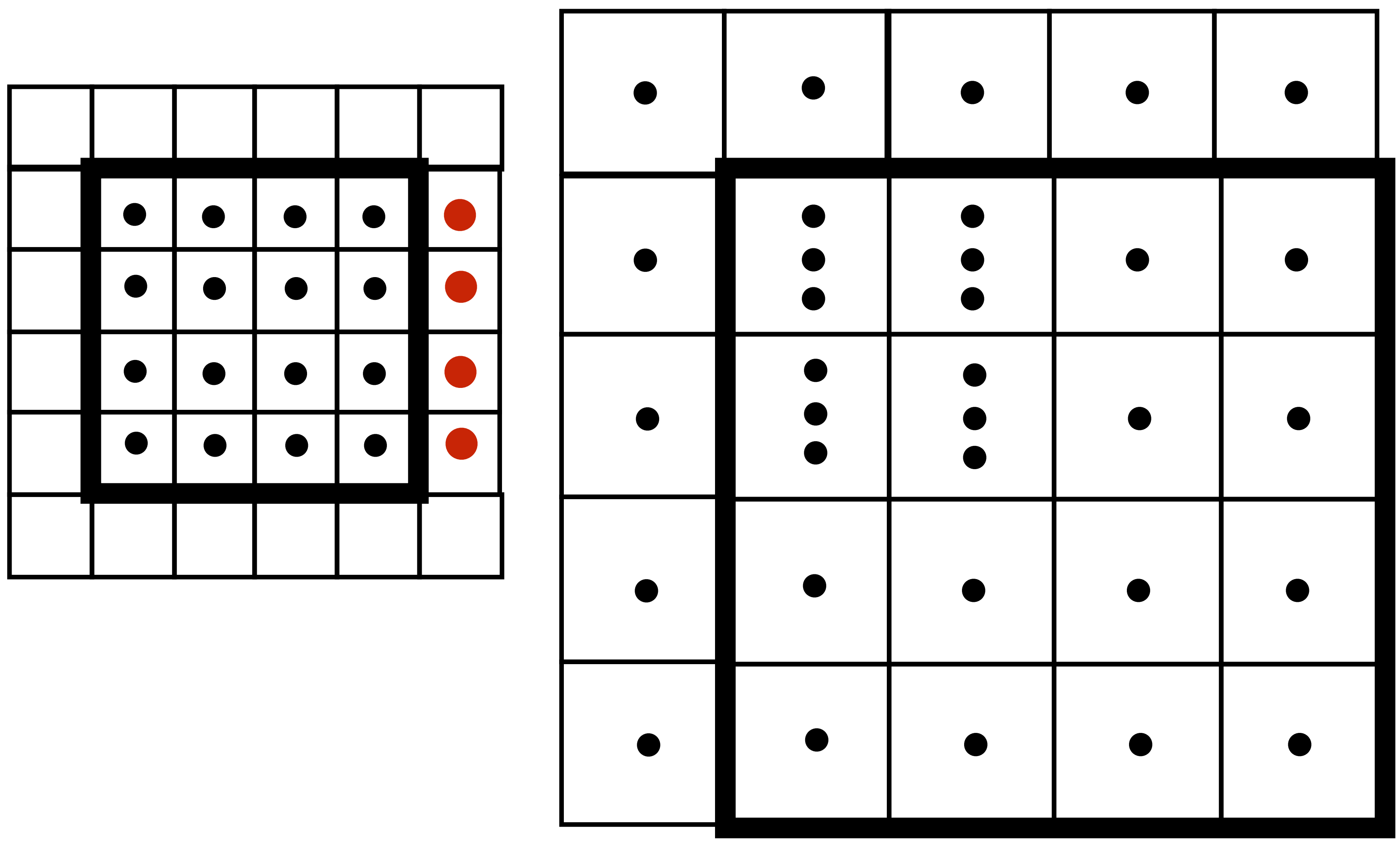
Interpolating halo values



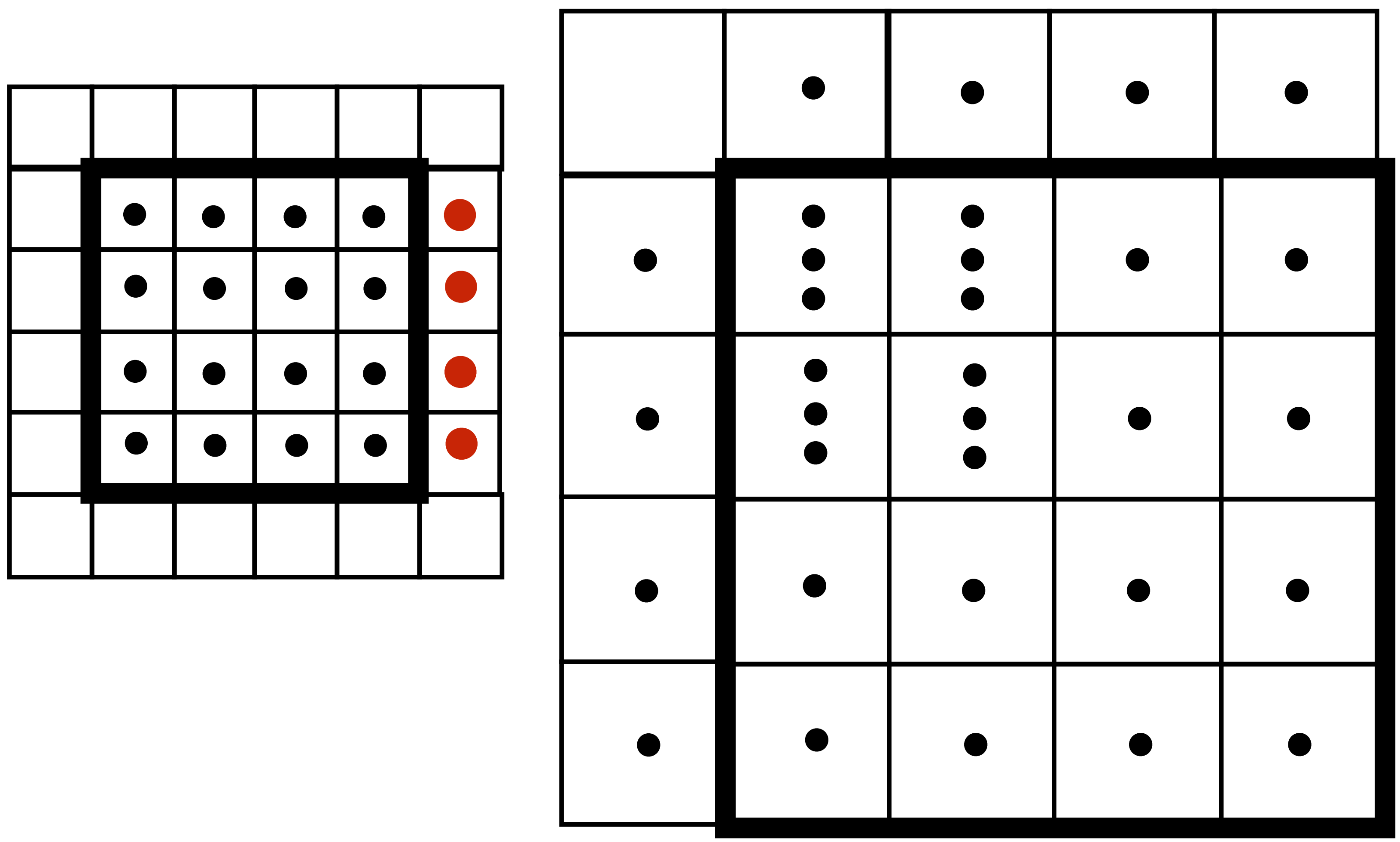
Interpolating halo values



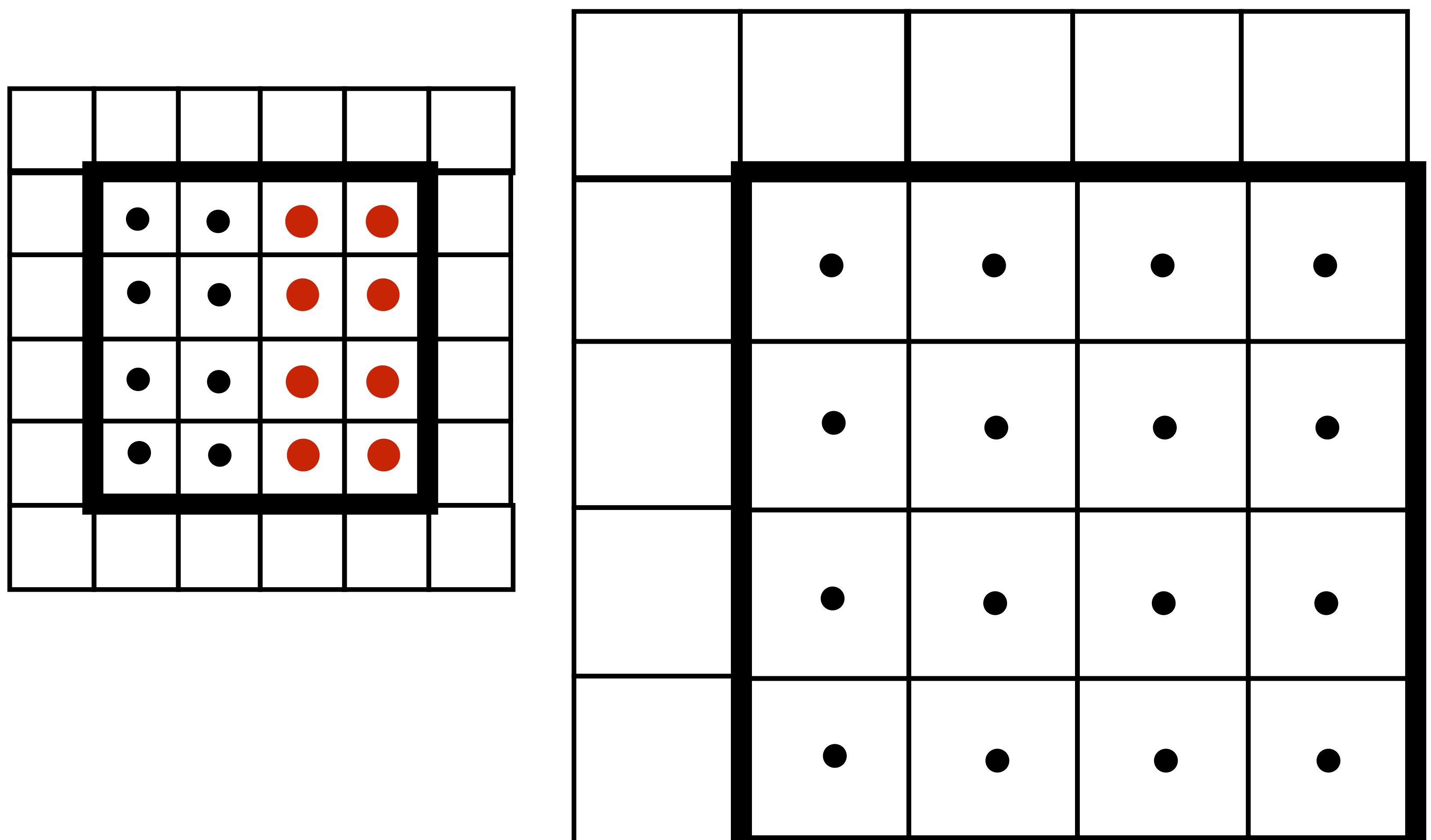
Interpolating halo values



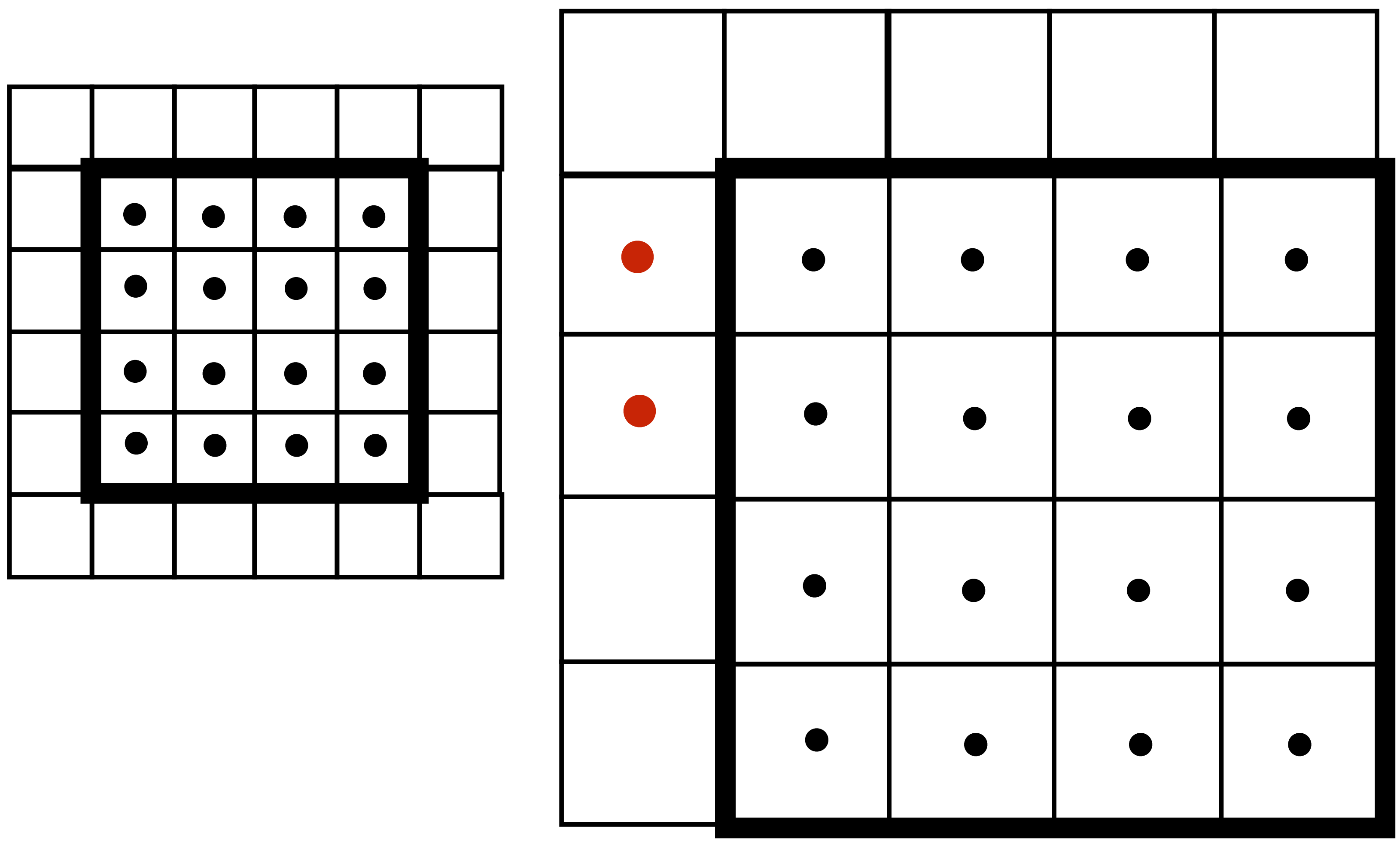
Interpolating halo values



Reducing halo values



Reducing halo values

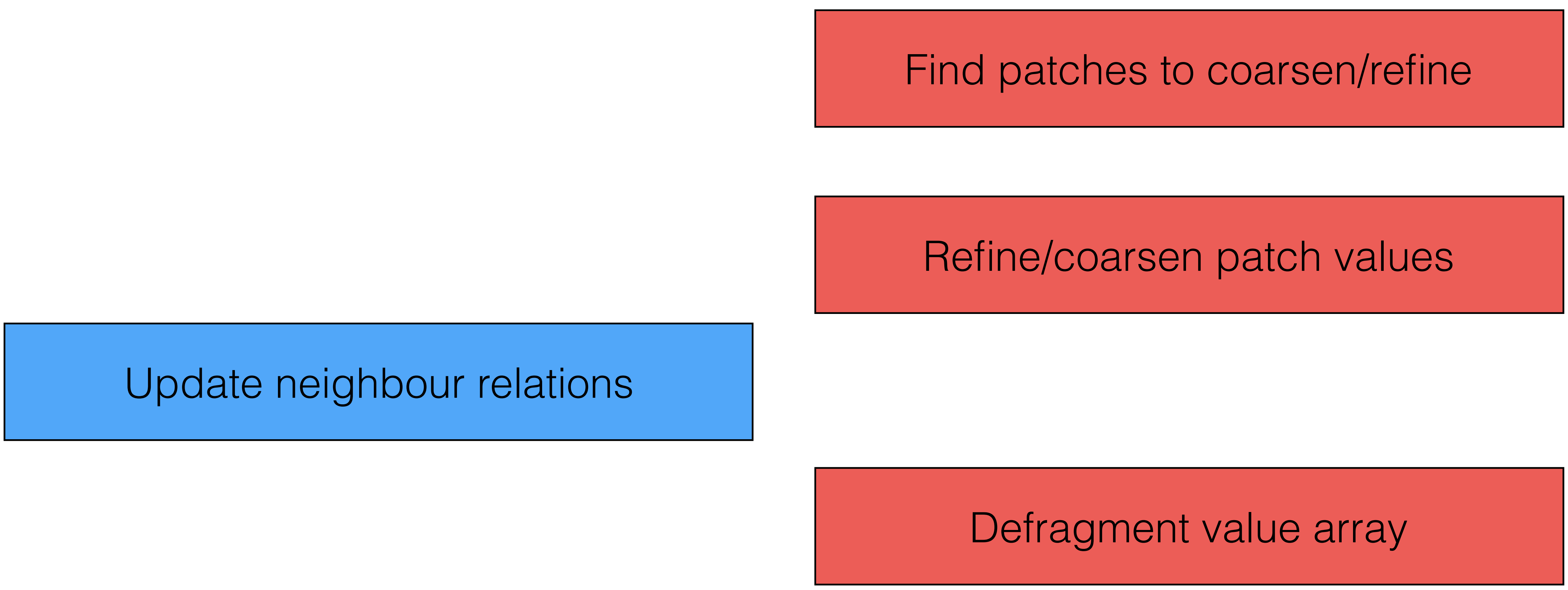


Coarsen/refine step

CPU

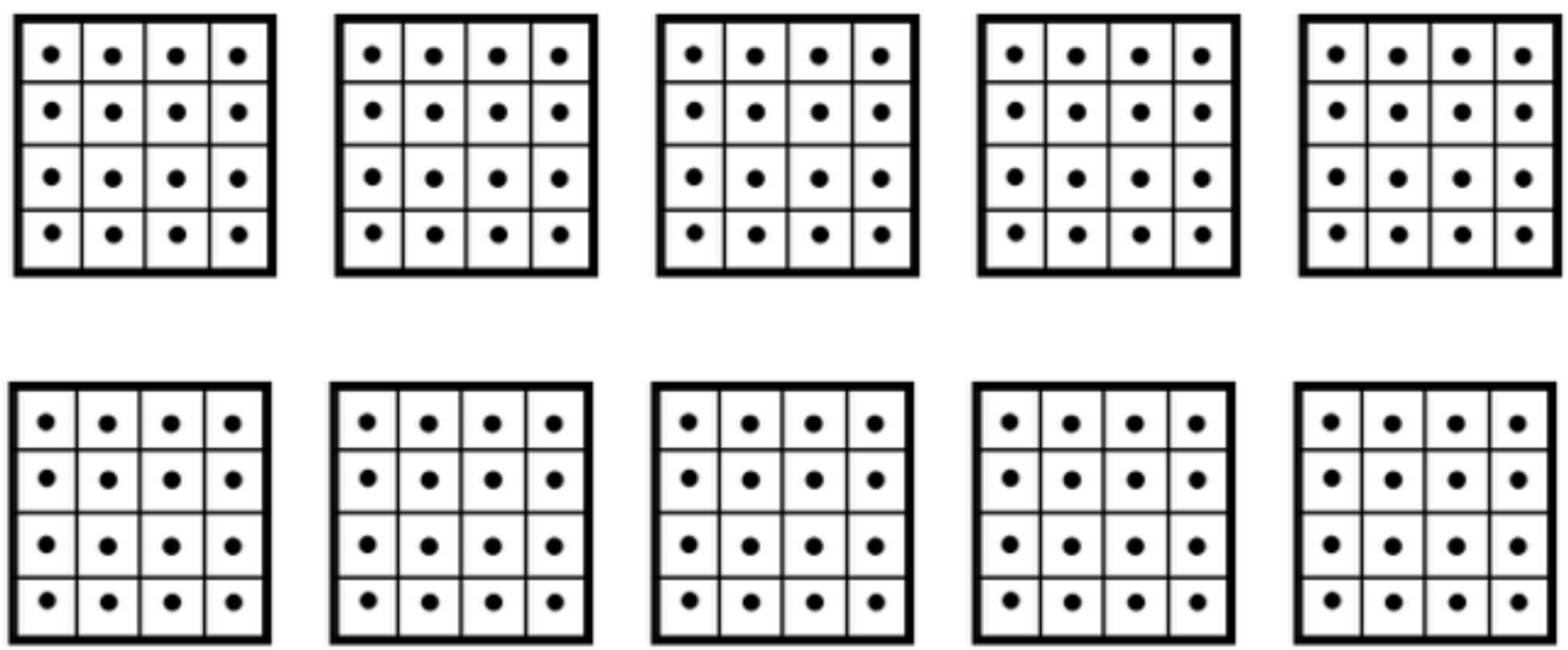
GPU

Main loop:



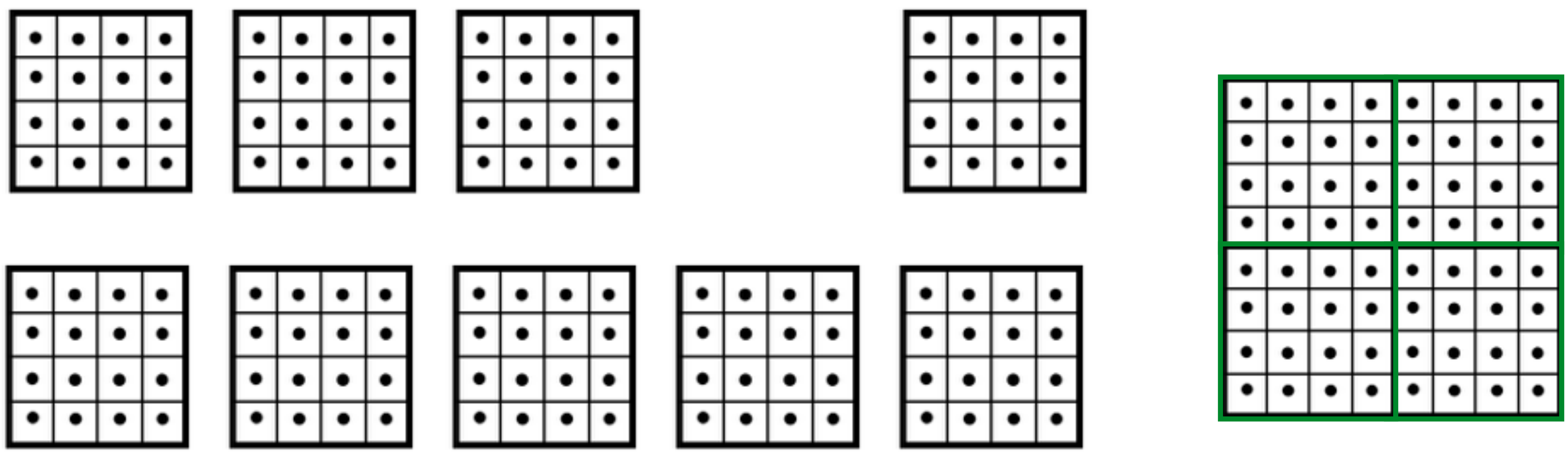
Defragment value array

refine node



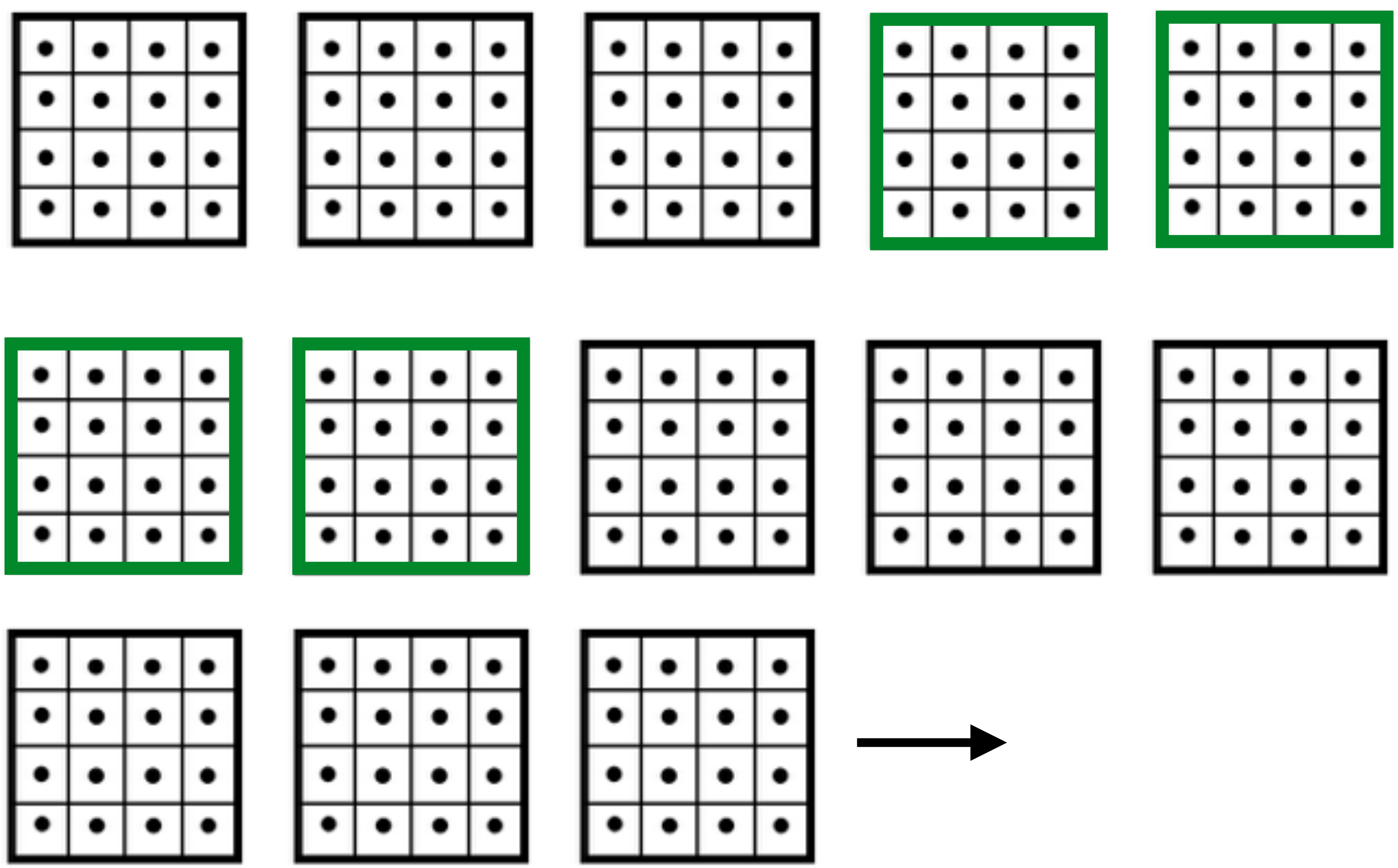
Defragment value array

refine node



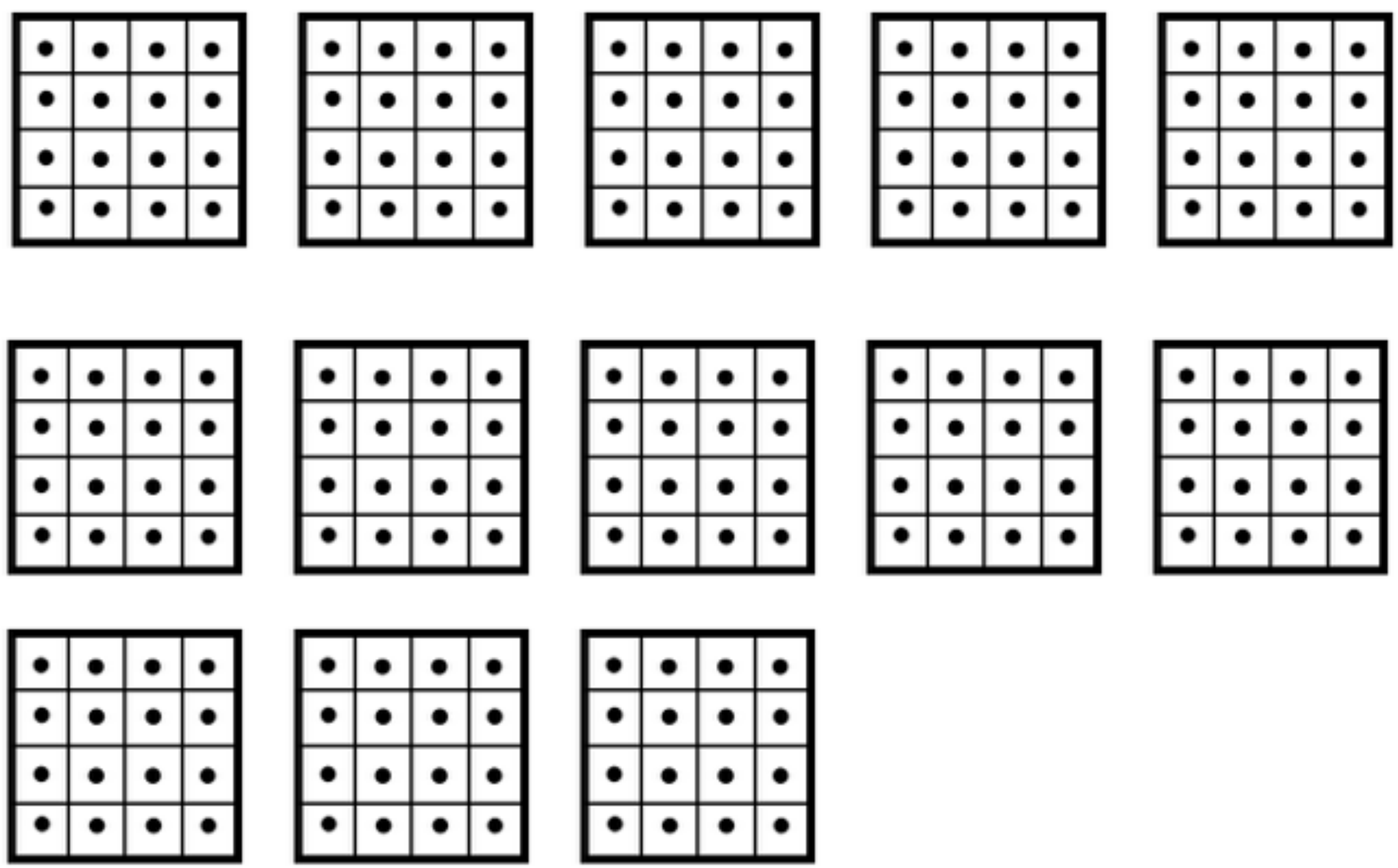
Defragment value array

refine node



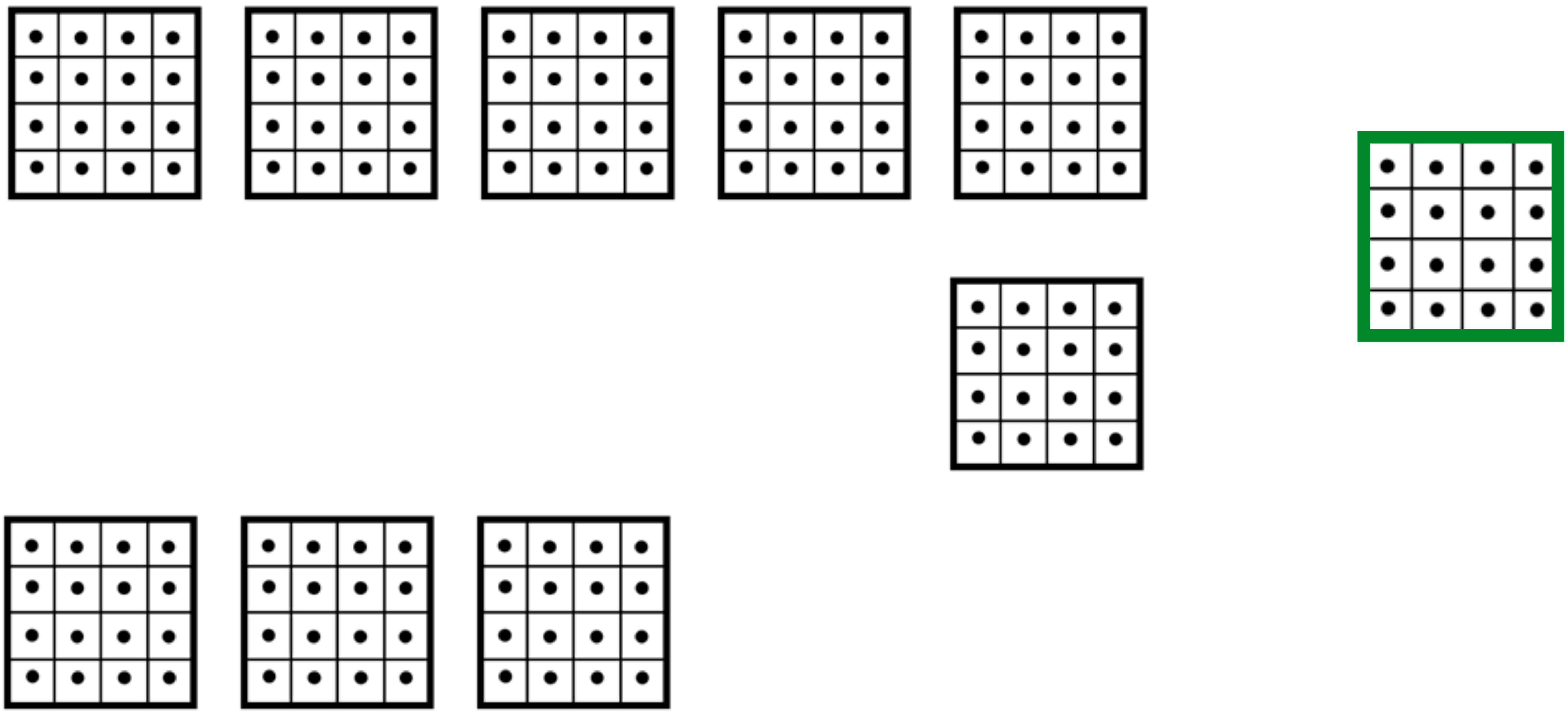
Defragment value array

coarsen node



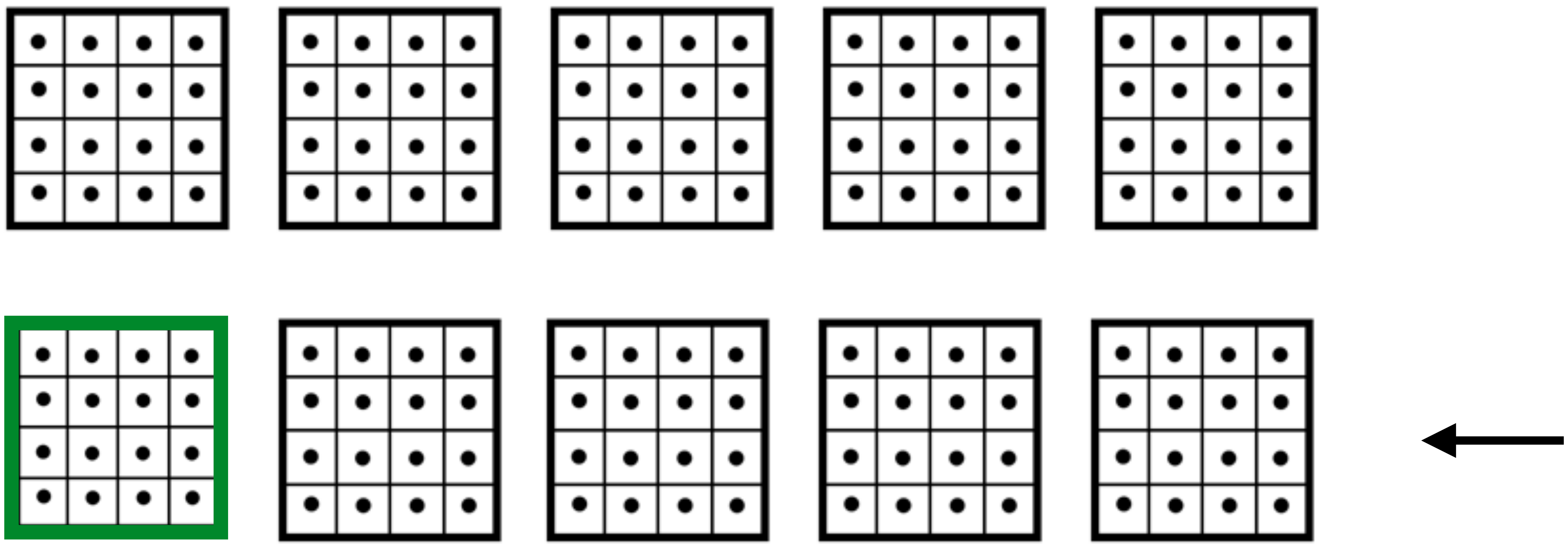
Defragment value array

coarsen node



Defragment value array

coarsen node

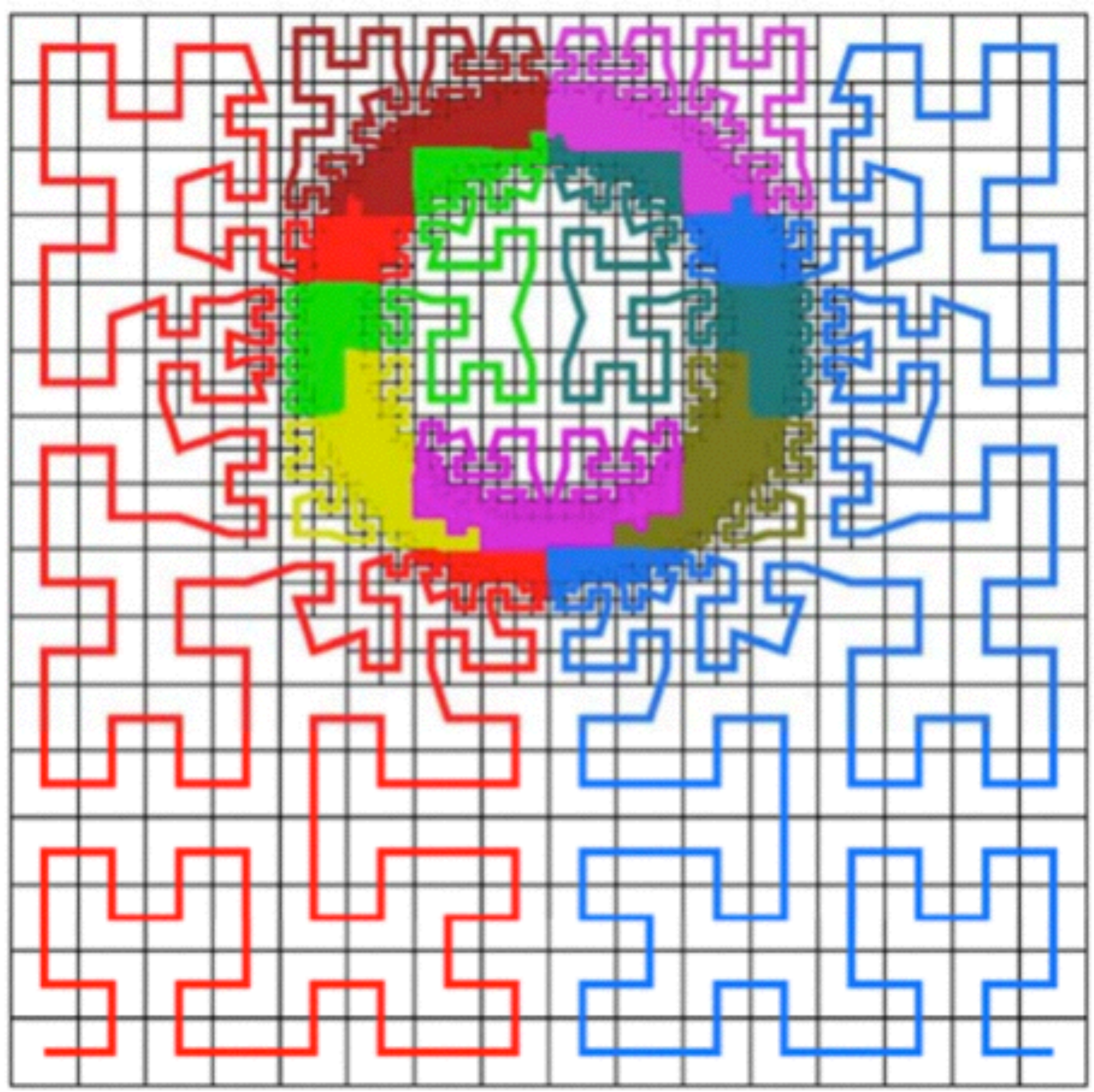


Calculate new defragmented position

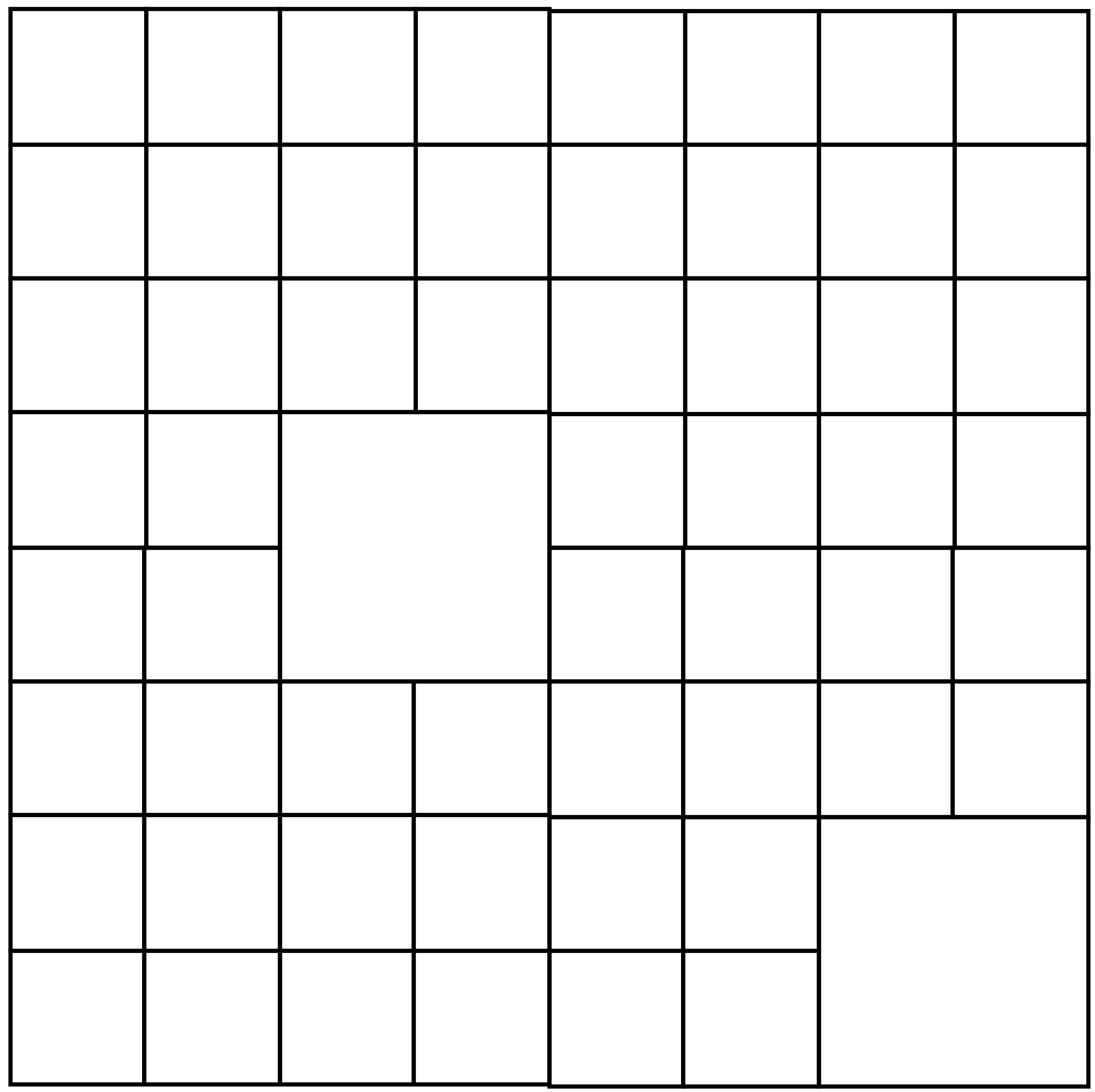
- Input: for all nodes, flag whether that node is to be refined, coarsened or unchanged
- Refined: +3
  Coarsened: -3
  Unchanged: 0
- For each element, sum all preceding elements in the array
- For n nodes, requires n/2 threads and O(log2(n)) serial steps

Multi-GPU

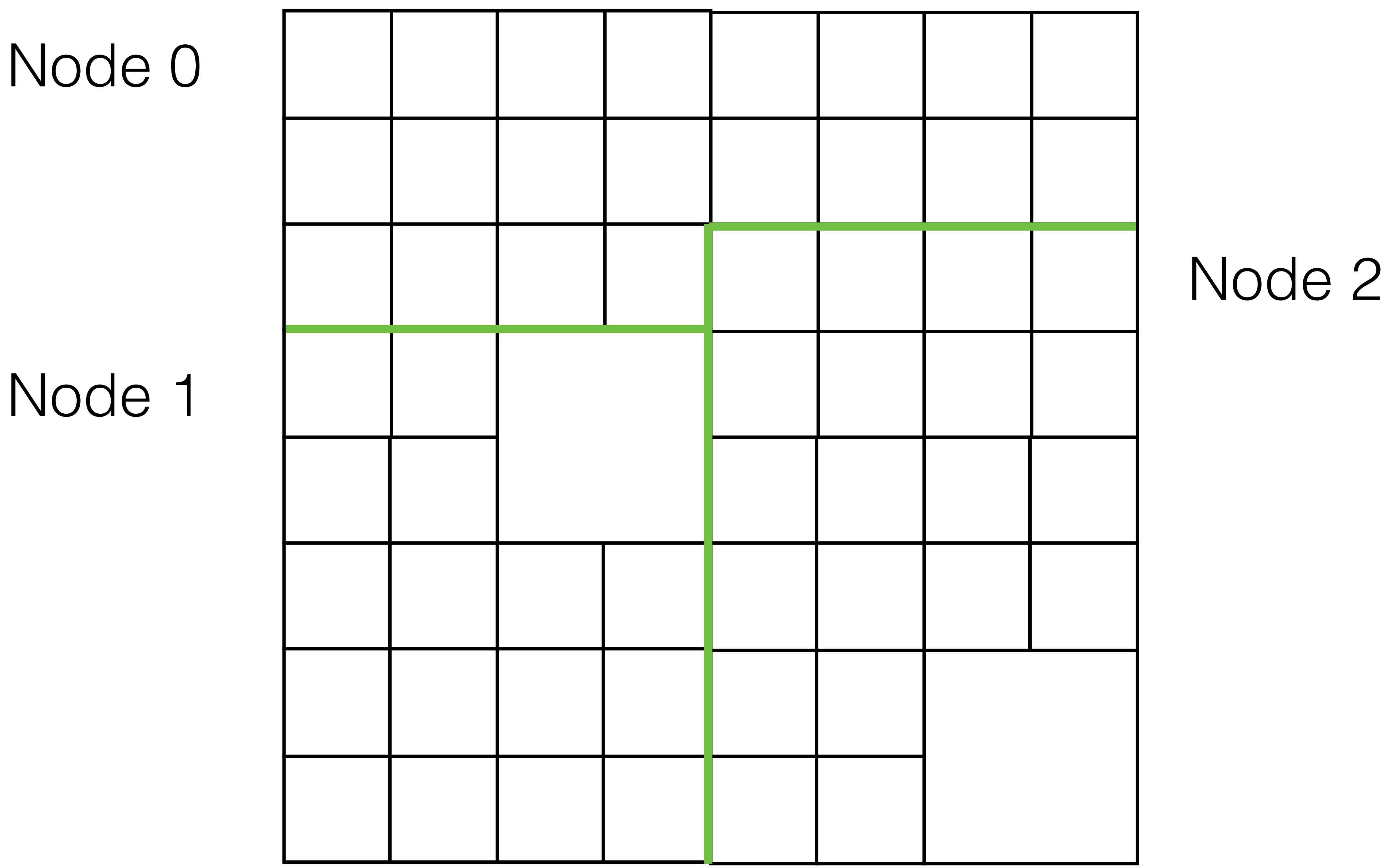
Load balancing



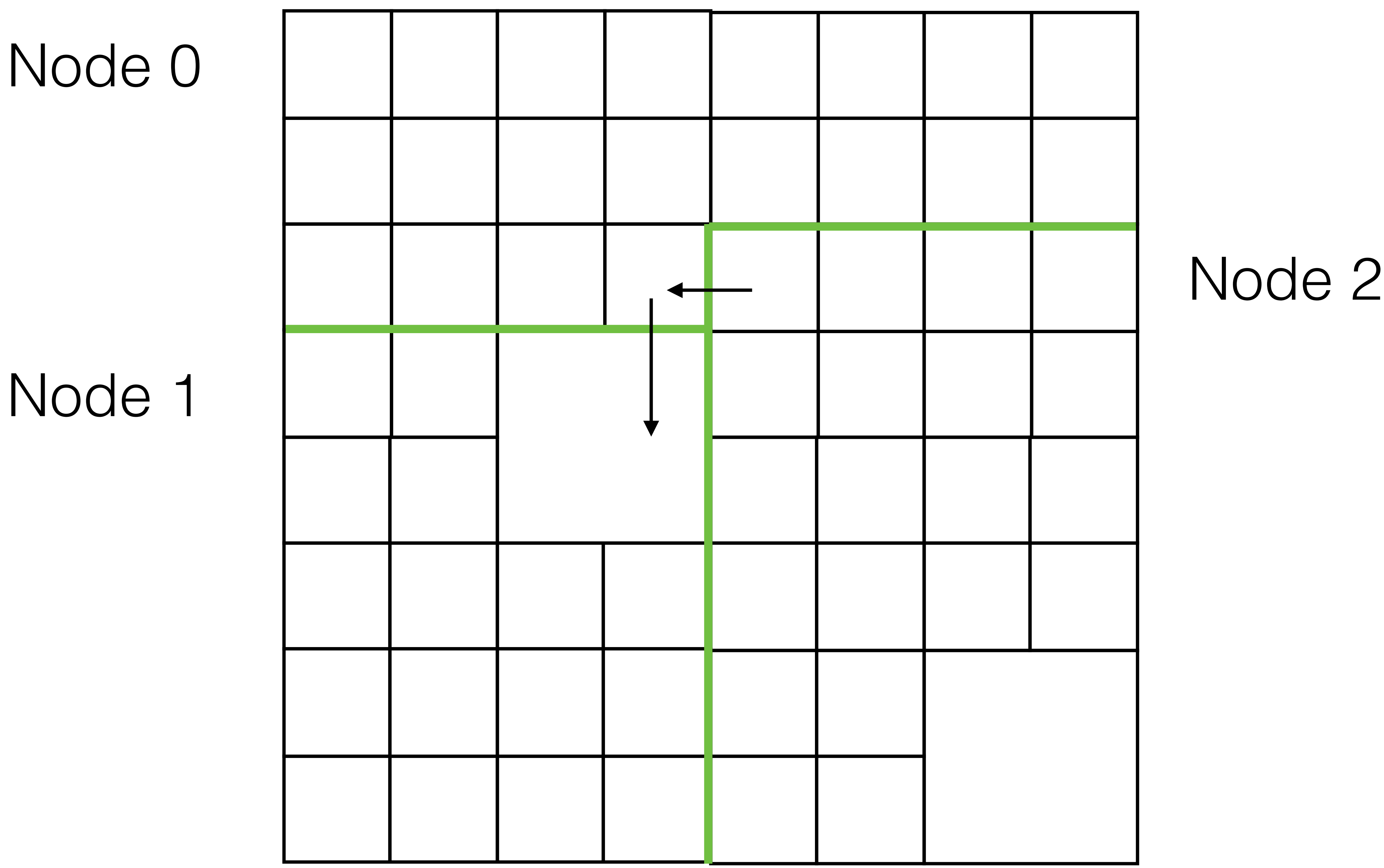
Boundaries between subdomains



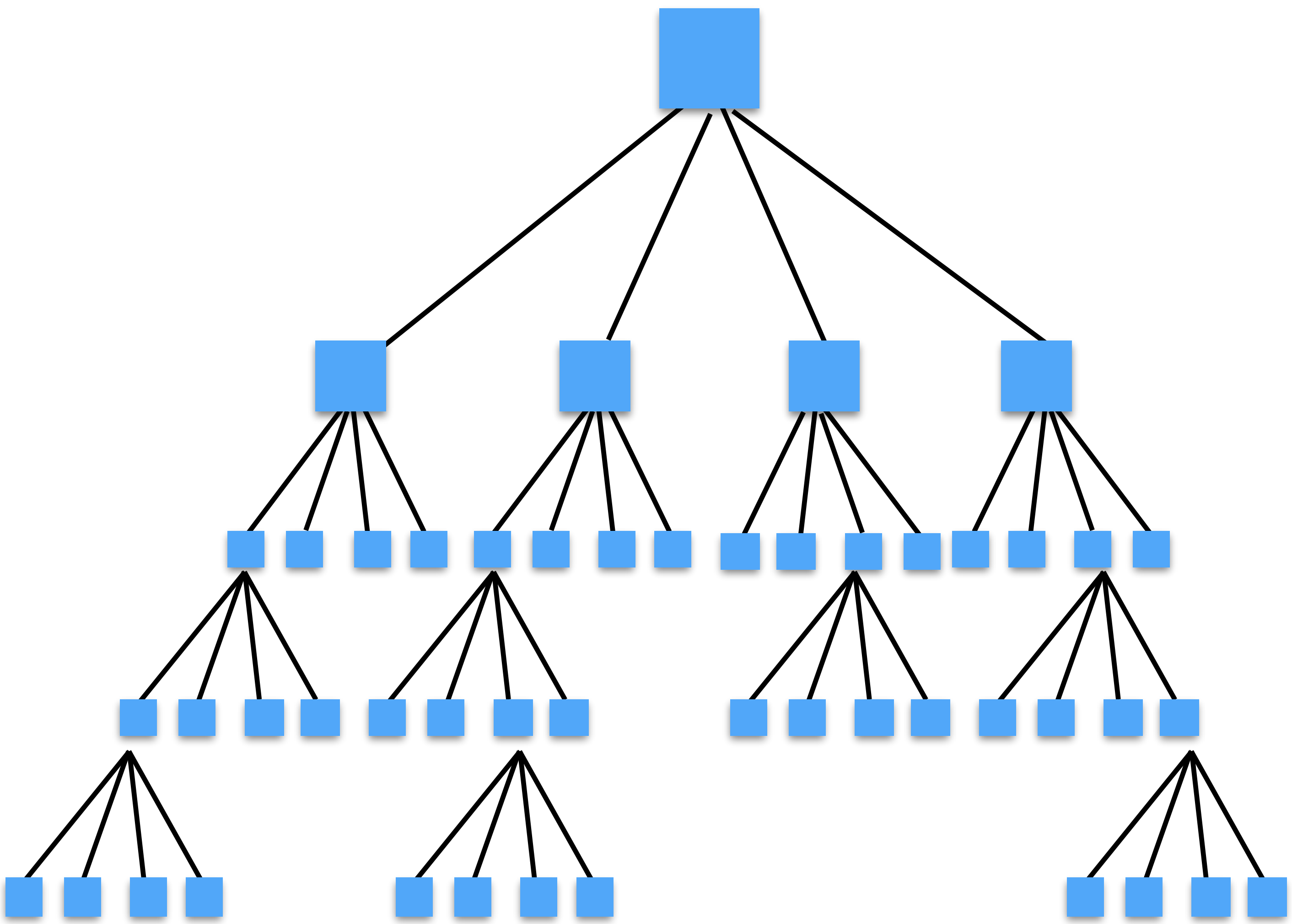
Boundaries between subdomains



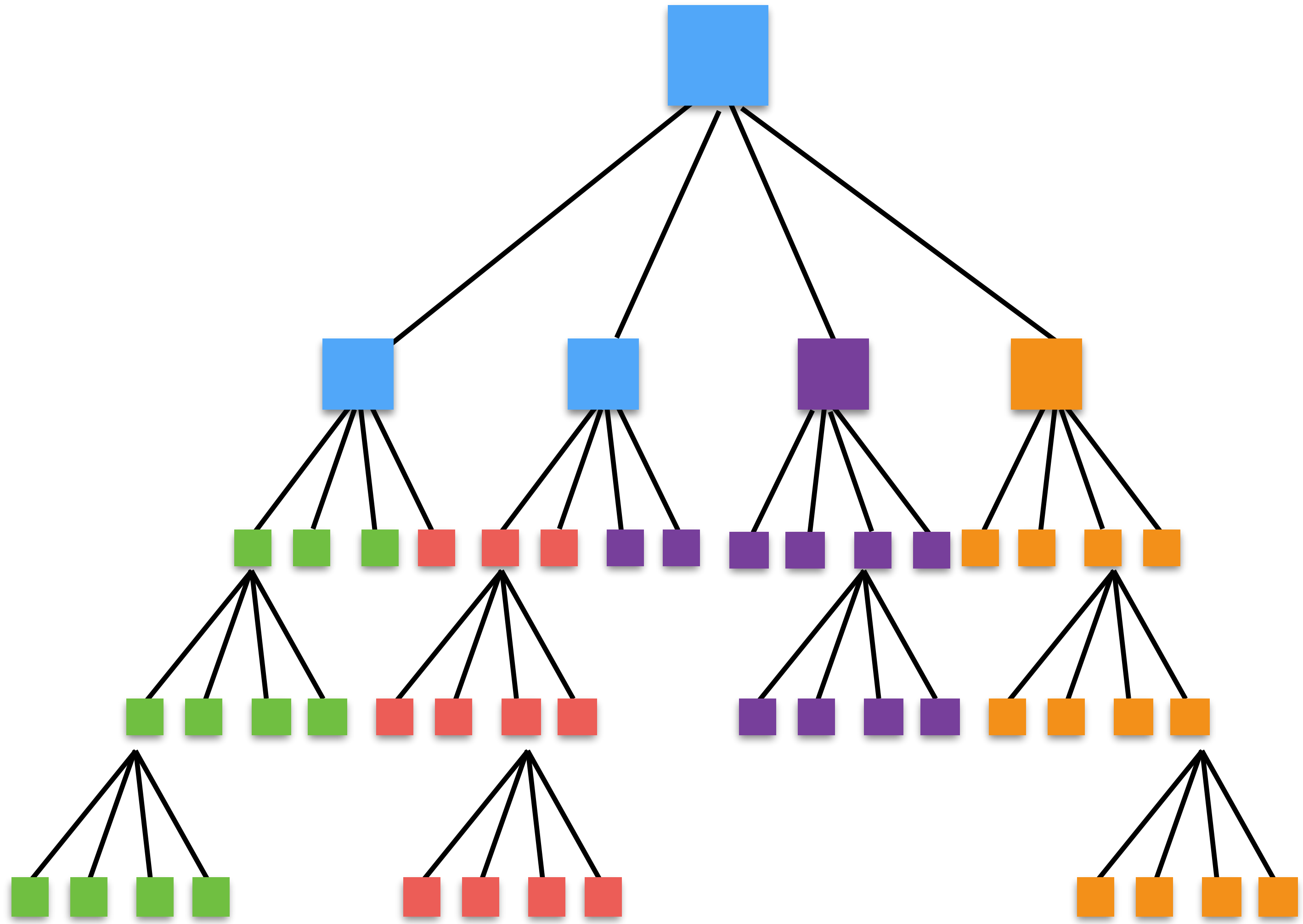
Boundaries between subdomains



How to distribute tree



How to distribute tree



Software design

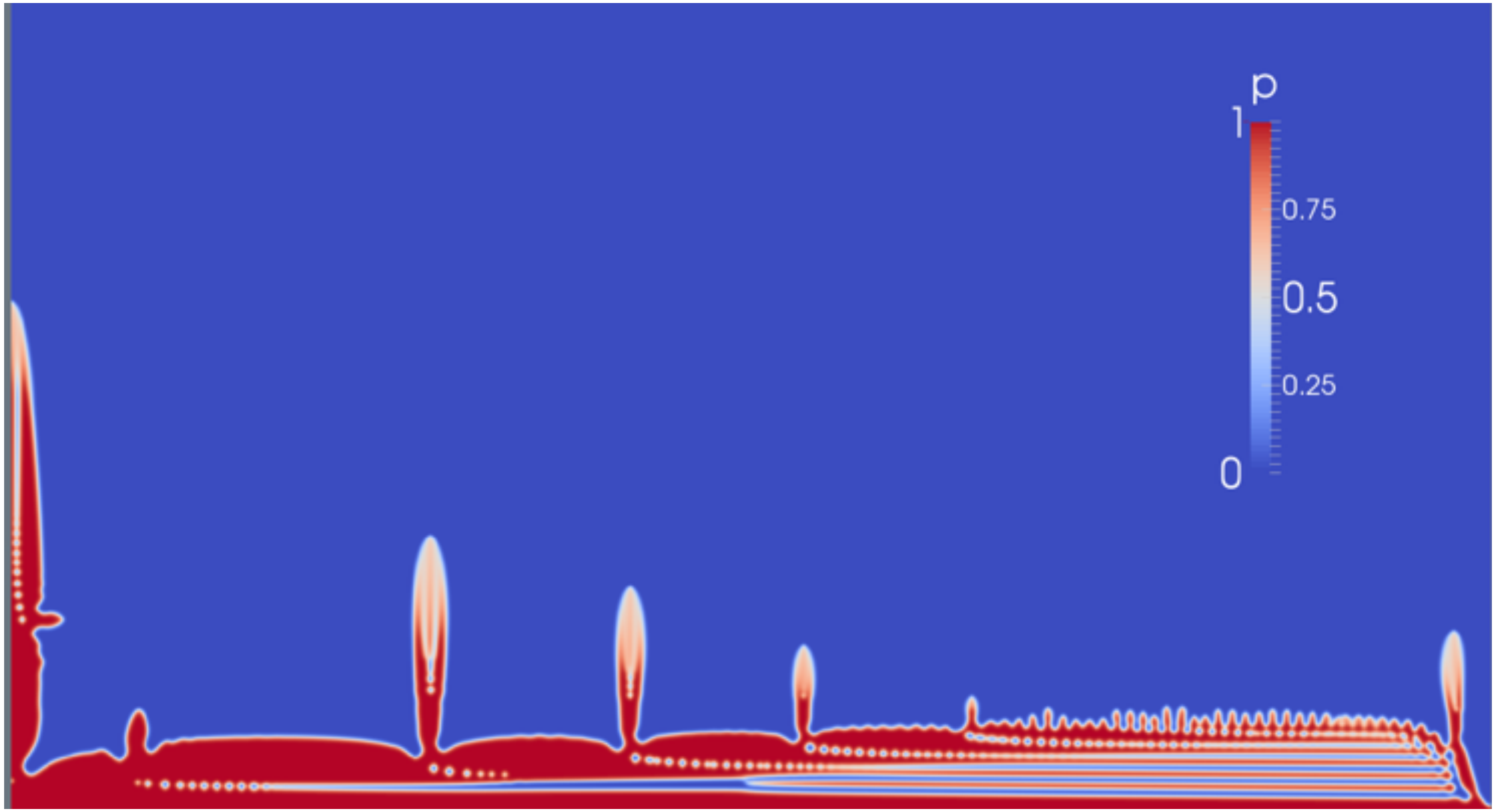
- Code framework — allow user to edit/add functions for initialisation, resolution criterion, stencil calc
- Code generation — annotated regular data structures
- How much to offer? — cell/node centre, interpolation level, stencil type

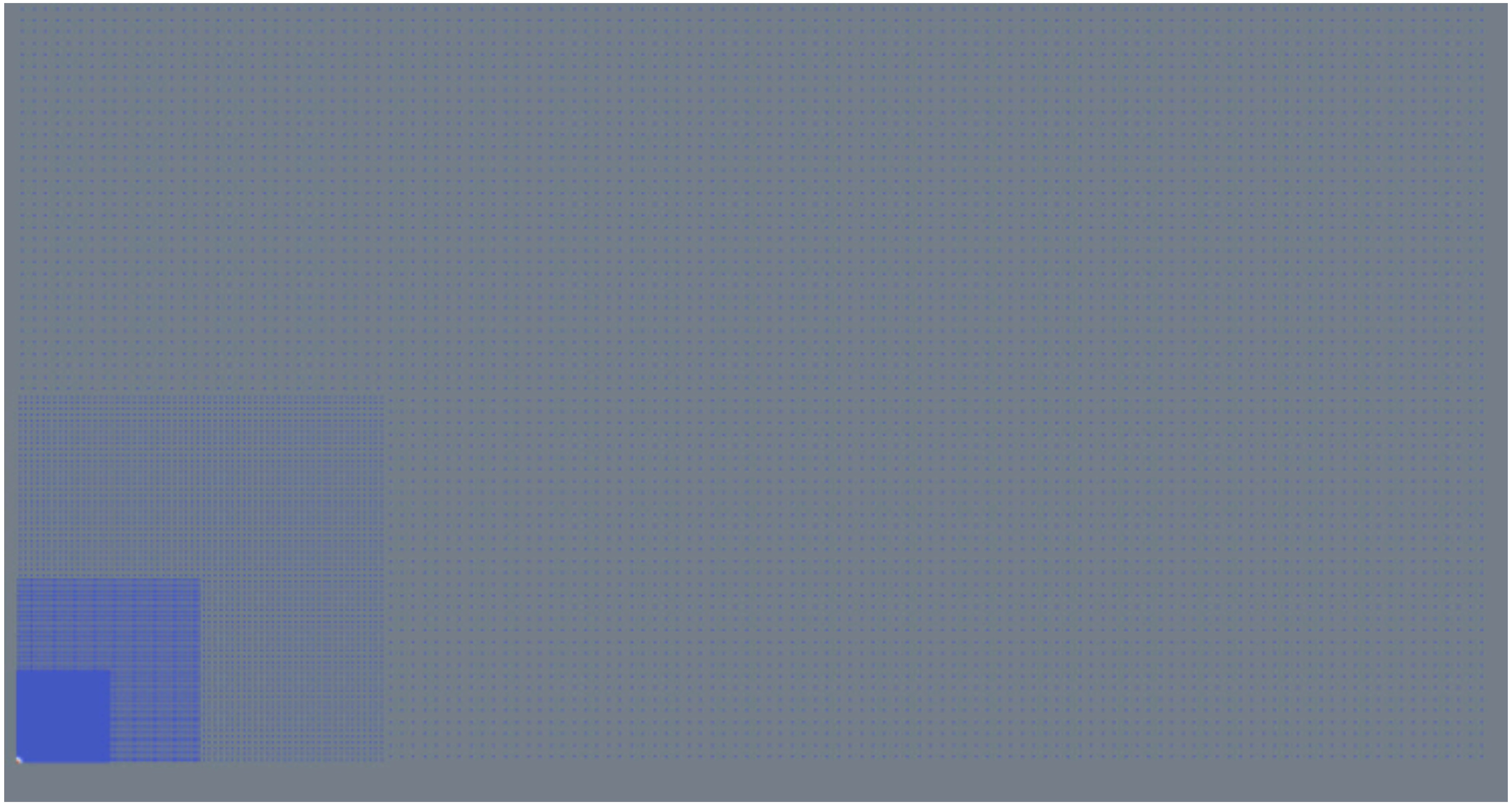
Software development process

- Unit testing
- Verification
- Profiling

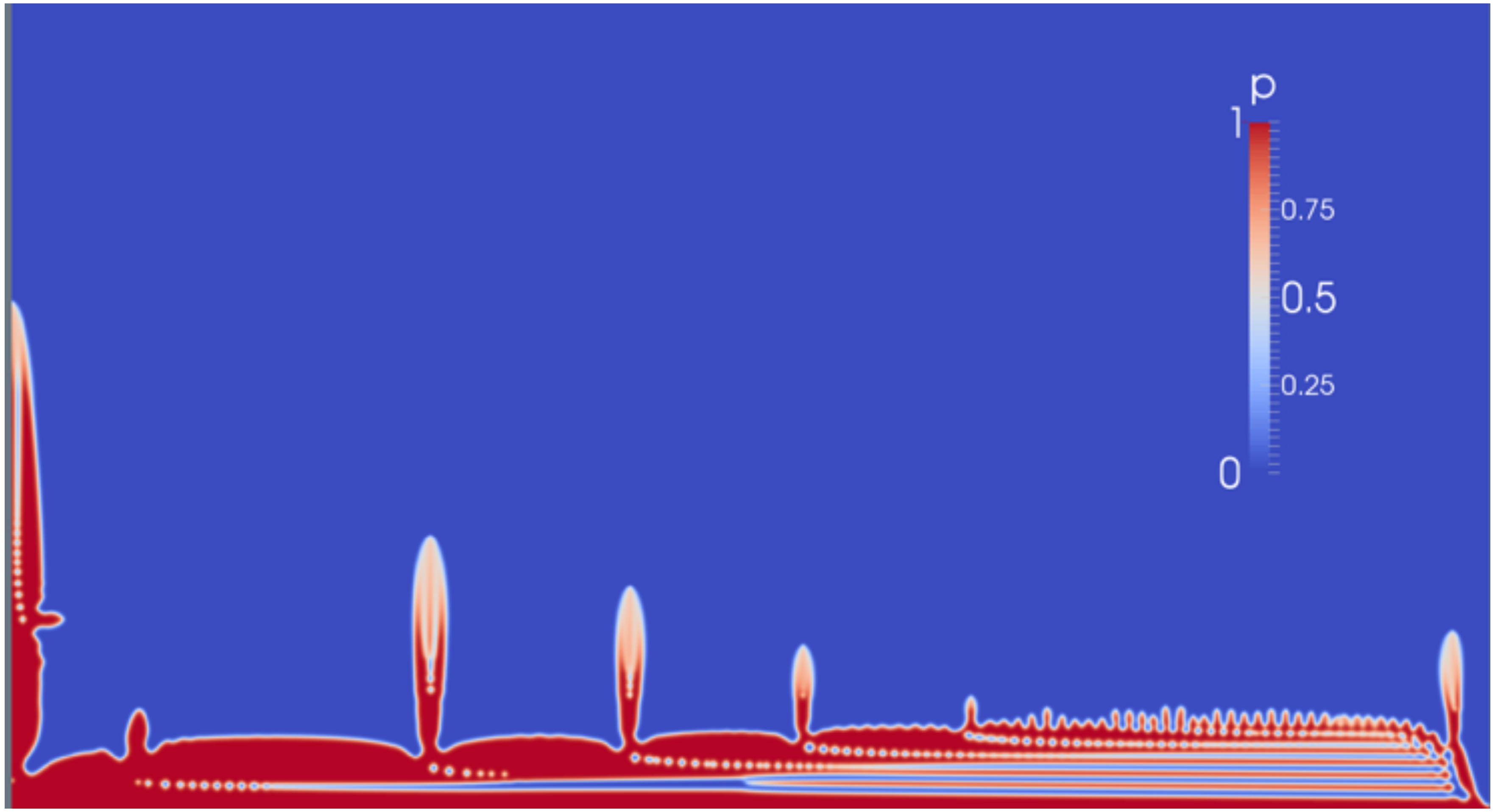
Phase field model of dendritic solidification in a binary alloy

Code by: T.Shimokawabe, T.Takaki (2011)

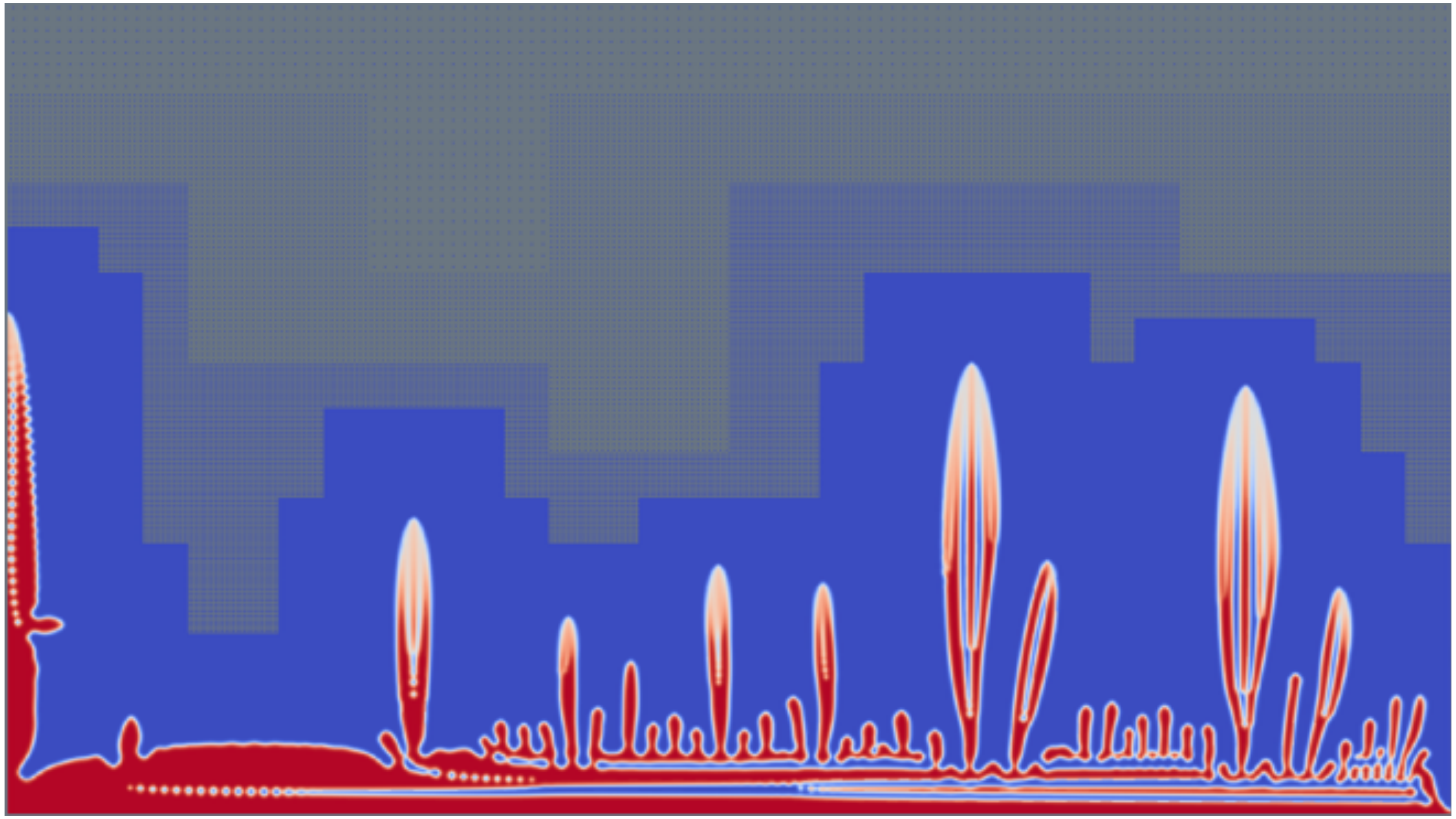




7 refinement levels in quad-tree



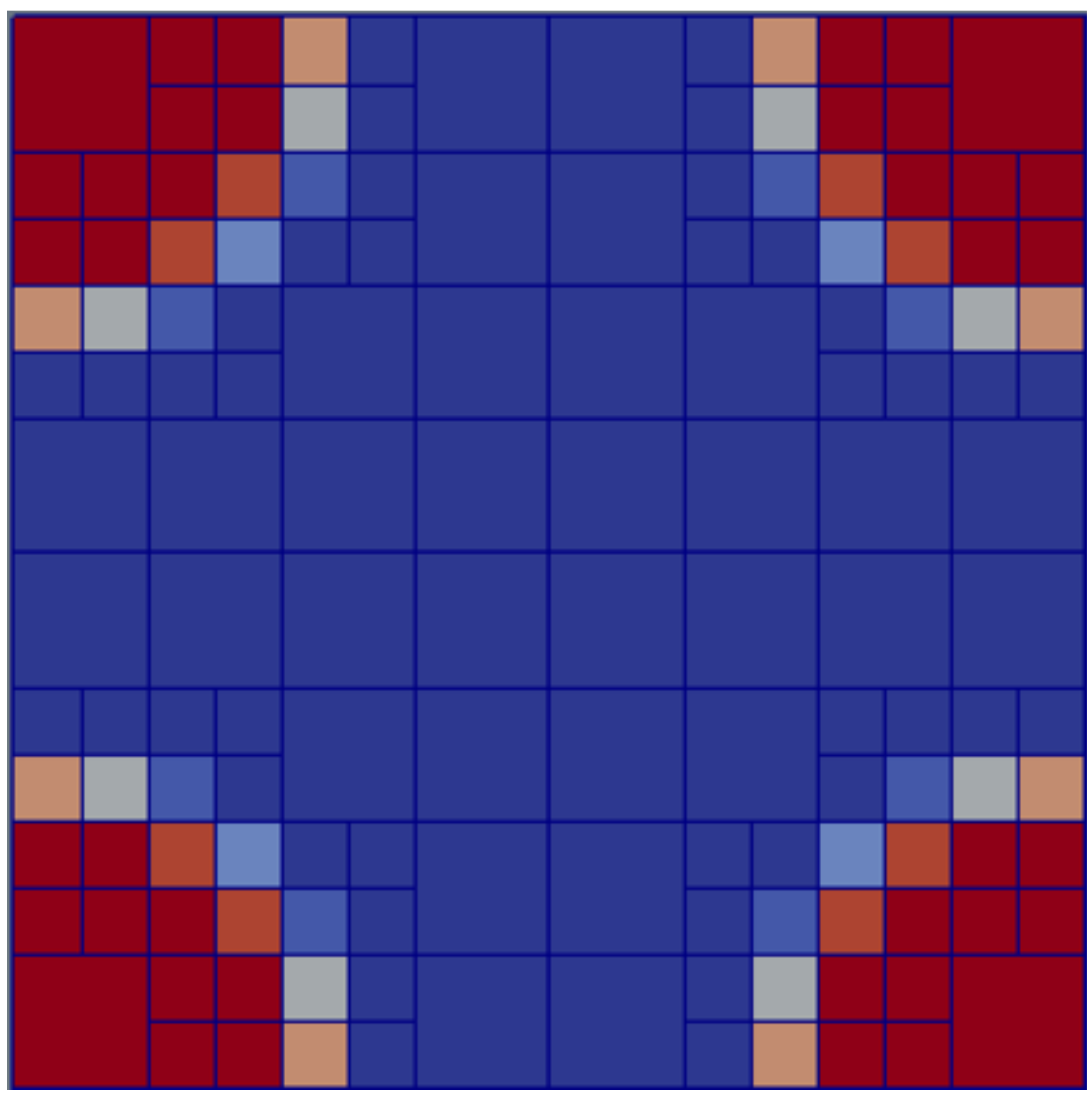
Regular mesh



Adaptive mesh

Performance testing for dendritic solidification model

$L = 1.5 \times 10^{-3} m$



$R = 4.5 \times 10^{-4} m$

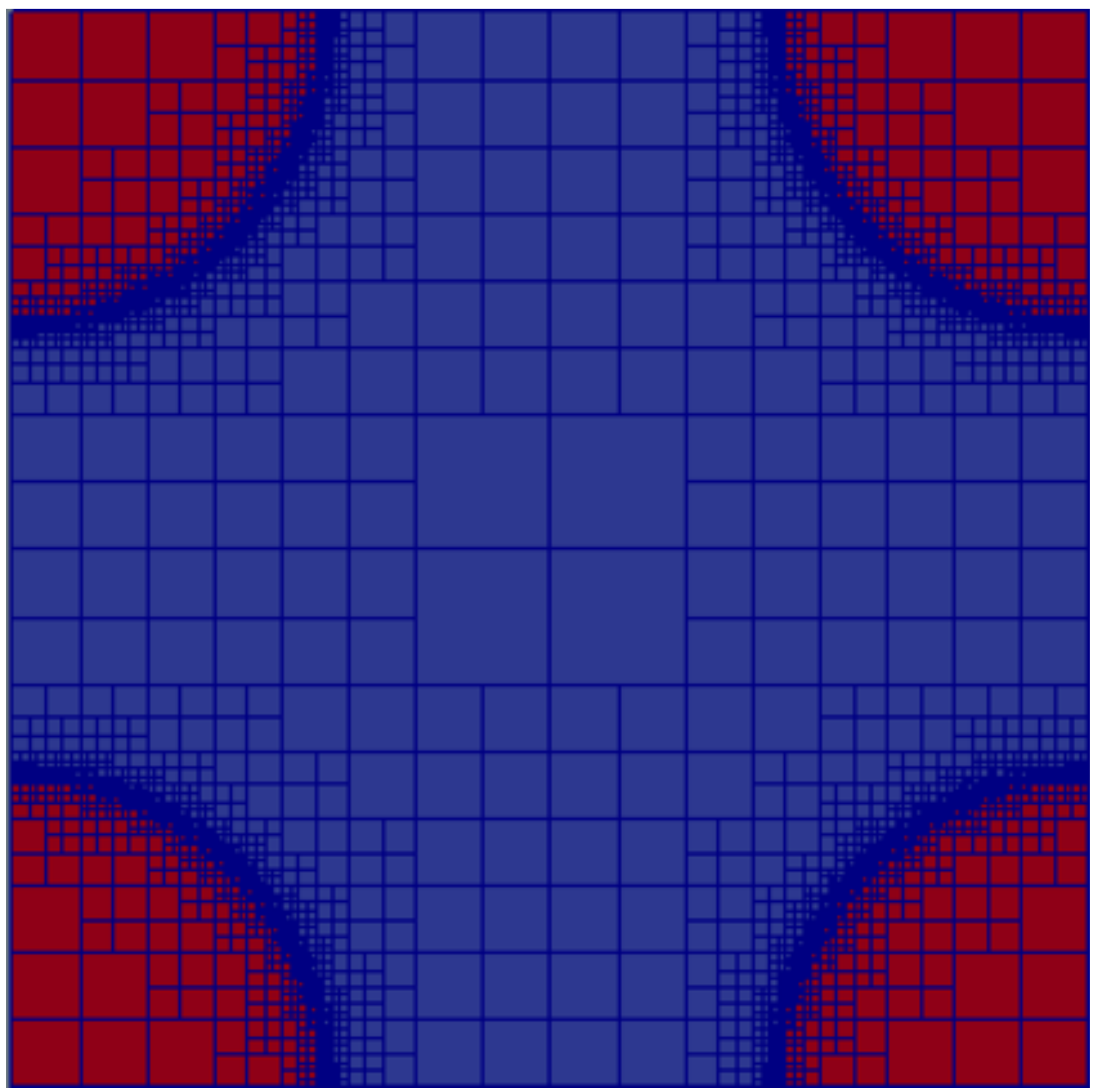
$\Delta x_{min} = 6 \times 10^{-6} m$

$\Delta x_{max} = 1.2 \times 10^{-5} m$

256 x 256

Performance testing for dendritic solidification model

$L = 1.5 \times 10^{-3} m$

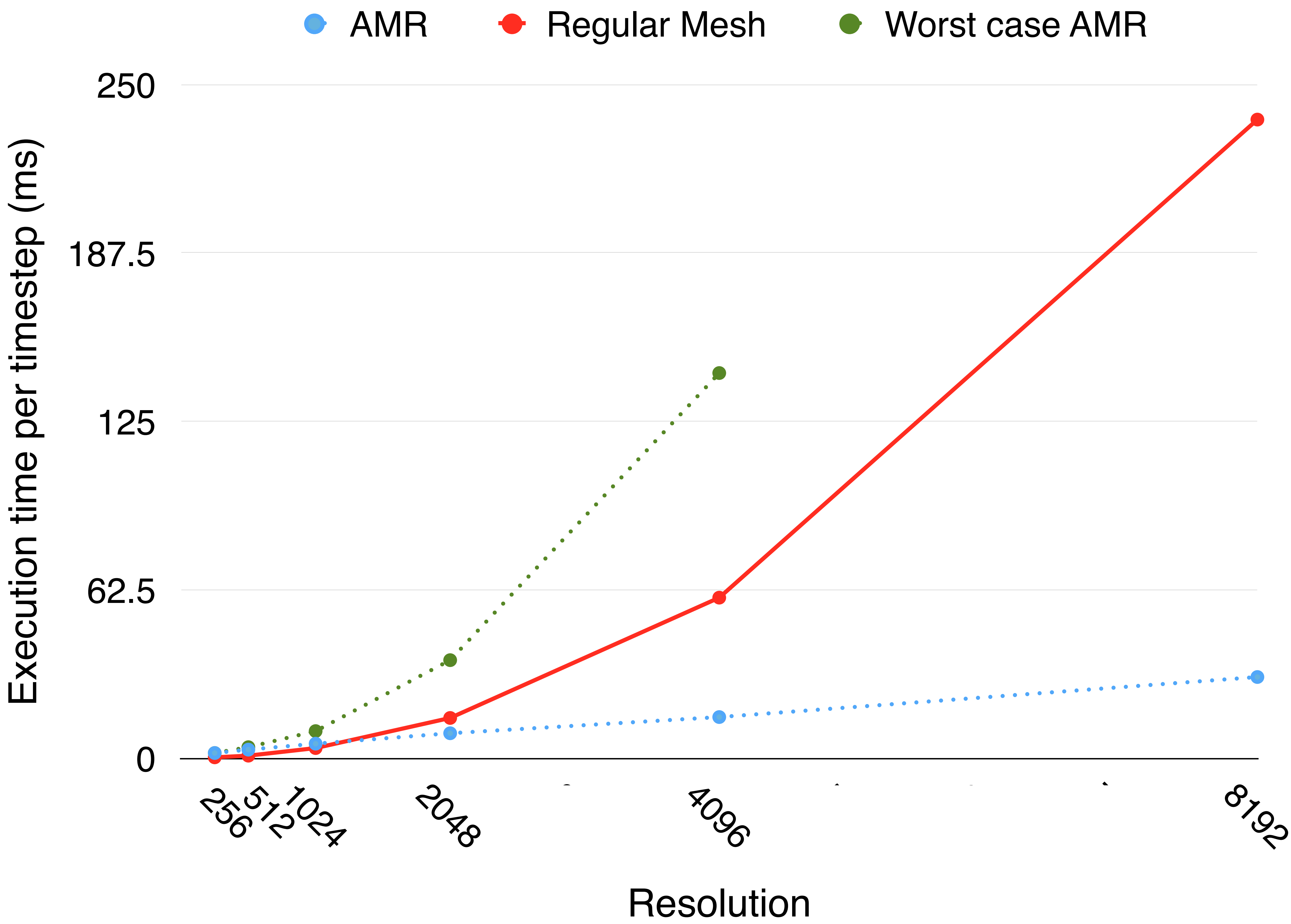


$R = 4.5 \times 10^{-4} m$

$\Delta x_{min} = 1.9 \times 10^{-7} m$

$\Delta x_{max} = 1.2 \times 10^{-5} m$

8192 x 8192


AMR
Regular Mesh
Worst case AMR
250
187.5
125
62.5
0
Execution time per timestep (ms)
256
512
1024
2048
4096
8192
Resolution

In summary

- Patch based tree-AMR
- For quick gains, offload update step to GPU
- GPU-native version possible — values on GPU, neighbour relations on CPU
- Likely won't be a one size fits all fix

Governing PDEs for phase field model

Diffusion Interface anisotropy

$$\frac{\partial \phi}{\partial t} = -M_\phi \Big[\nabla \cdot (a^2 \nabla \phi) + \frac{\partial}{\partial x}(a \frac{\partial a}{\partial \phi_x} |\nabla \phi|^2) + \frac{\partial}{\partial y}(a \frac{\partial a}{\partial \phi_y} |\nabla \phi|^2)$$

Chemical driving force Phase change

$$+ \frac{\partial}{\partial z}(a \frac{\partial a}{\partial \phi_z} |\nabla \phi|^2) - S \Delta T \frac{dp(\phi)}{d\phi} - W \frac{dq(\phi)}{d\phi} \Big]$$

$$\frac{\partial c}{\partial t} = \nabla \cdot [D_S \phi \nabla c_S + D_L (1 - \phi) \nabla c_L]$$

$\phi(x, y, t)$: phase

$c(x, y, t) = (1 - \phi) c_L + \phi c_S$

c_L : liquid concentration
c_S : solid concentration

M_ϕ : mobility
a : interface anisotropy
$p(\phi)$: interpolating function
$q(\phi)$: double well function
$D_S,\ D_L$: diffusion in solid, liquid
S : entropy of fusion
W : height of double well potential
T : temperature