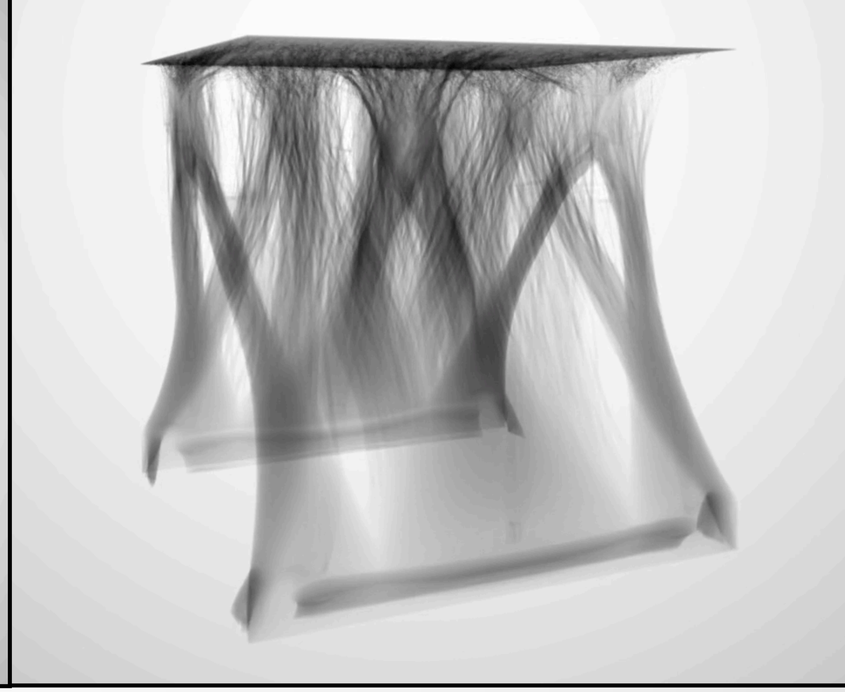


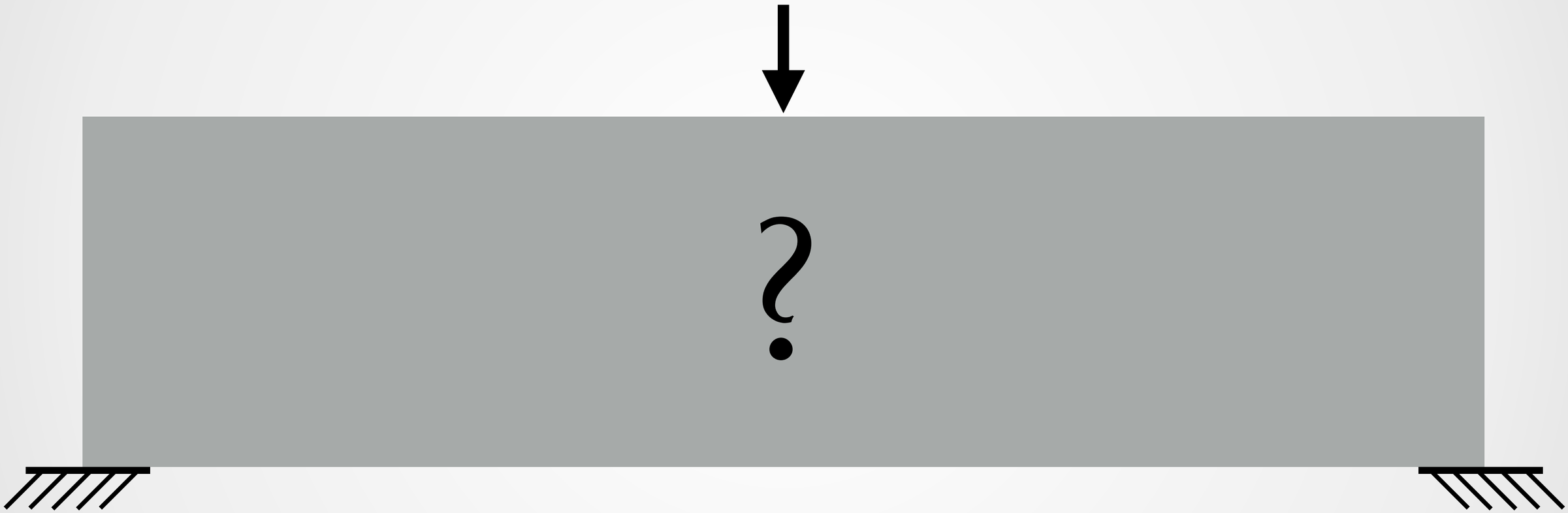
Narrow-band Topology Optimization on a Sparsely Populated Grid

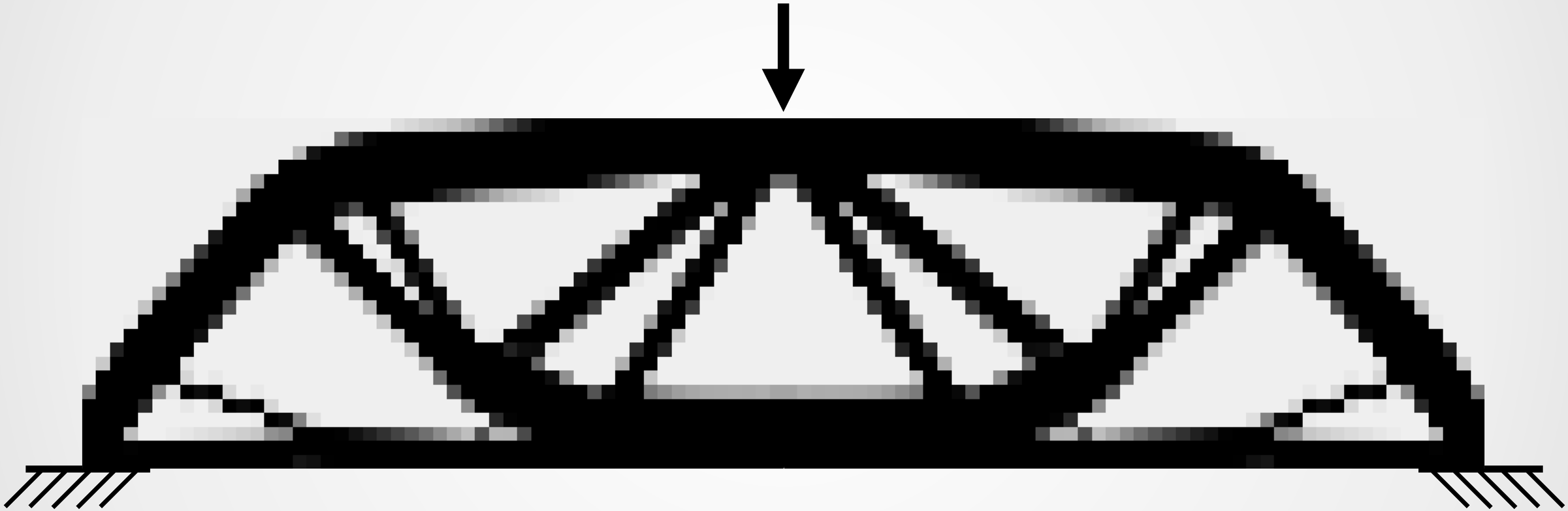
SIGGRAPH Asia 2018 ***Presented at TOP Webinar***

Haixiang Liu*[1] *Yuanming Hu*[*2] Bo Zhu[23] Wojciech Matusik[2] Eftychios Sifakis[1]

*equally contributed

[1]University of Wisconsin-Madison [2]MIT CSAIL [3]Dartmouth College





Design domain : 3D density field (array) of values in [0,1]

$$\begin{aligned} \min: \quad & S(\boldsymbol{\rho}, \mathbf{u}) = \mathbf{u}^T \mathbf{K}(\boldsymbol{\rho})\mathbf{u} \\ s.t.: \quad & \mathcal{F}(\boldsymbol{\rho}, \mathbf{u}) = \mathbf{K}(\boldsymbol{\rho})\mathbf{u} - \mathbf{f} = 0 \\ : \quad & V(\boldsymbol{\rho}) - \hat{V} \leq 0, \end{aligned}$$

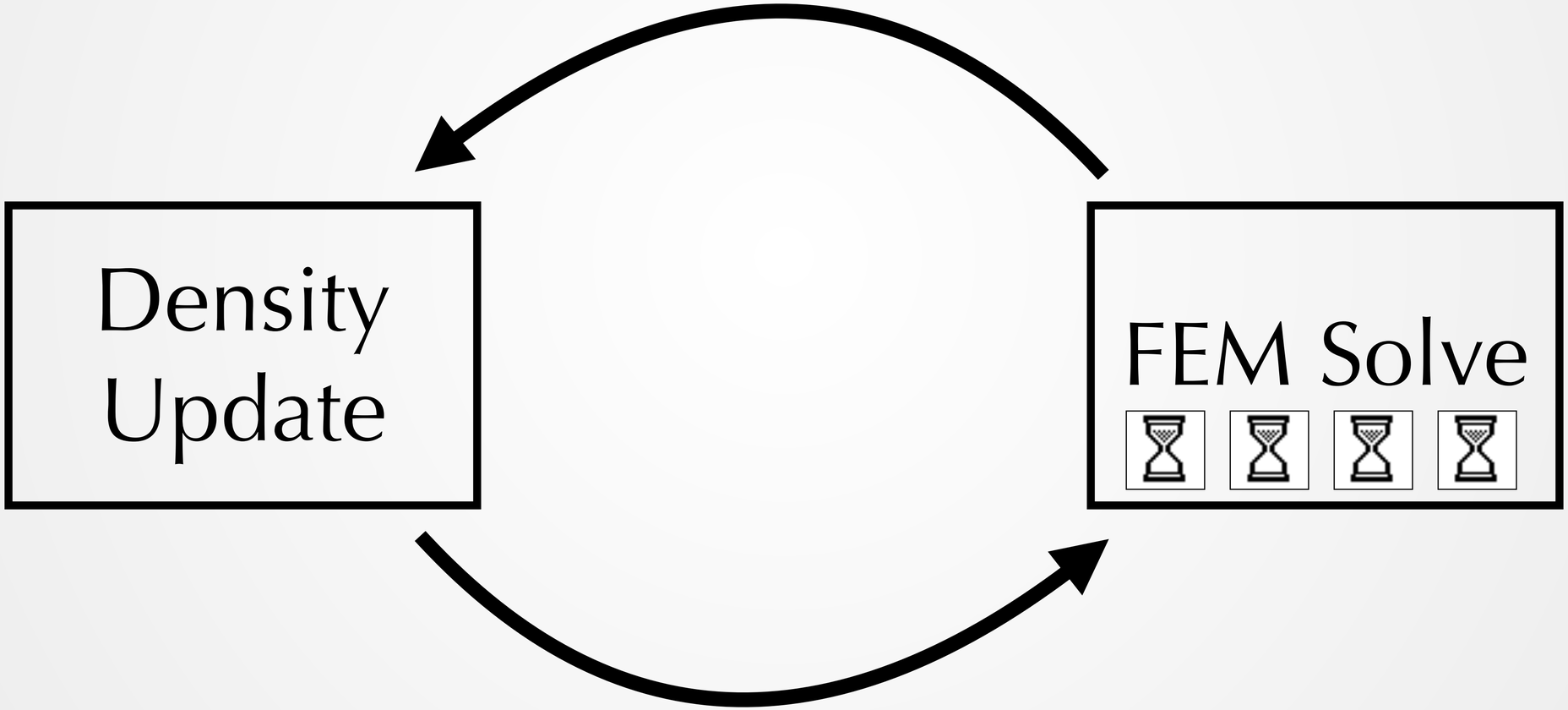
Minimal compliance objective (deformation)

Linear FEM

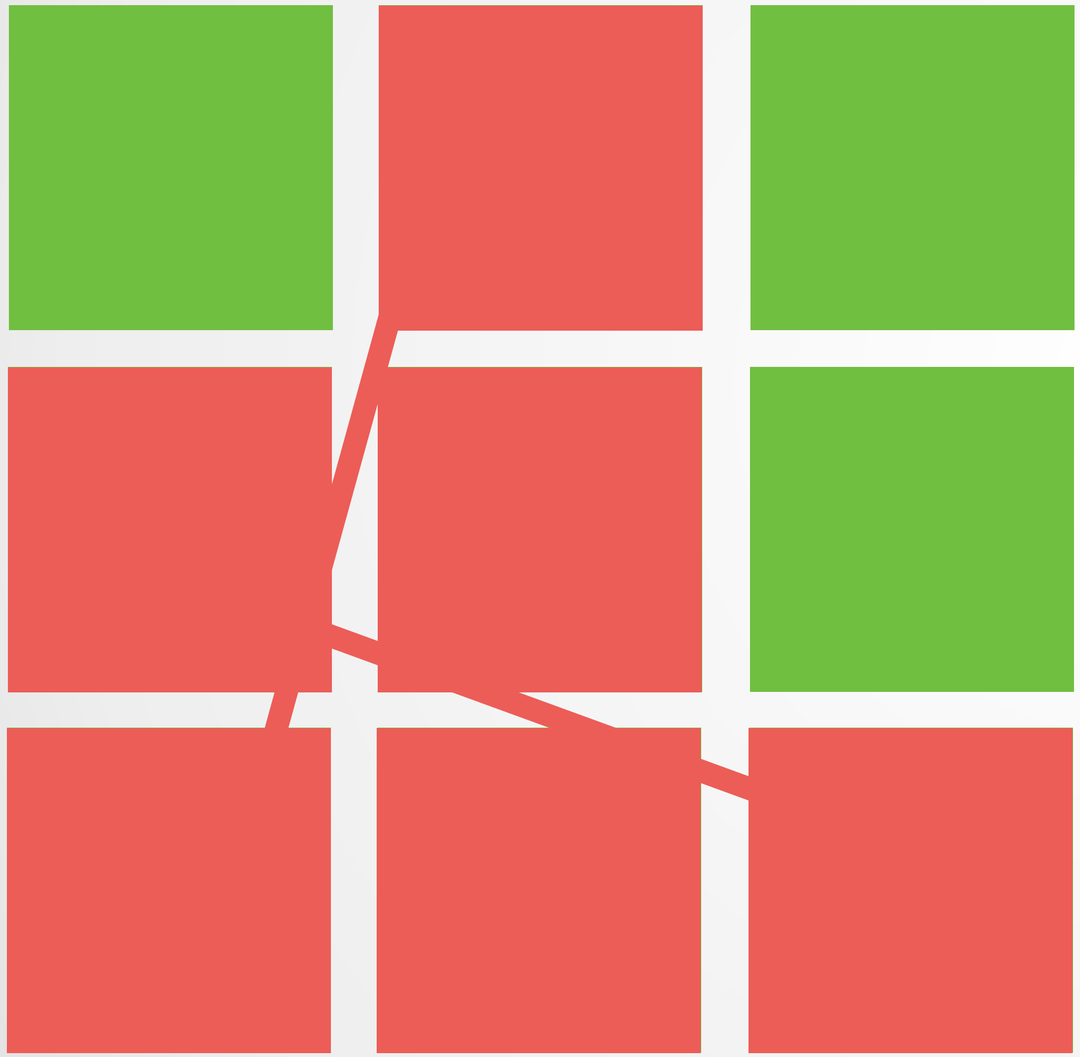
Volume constraint

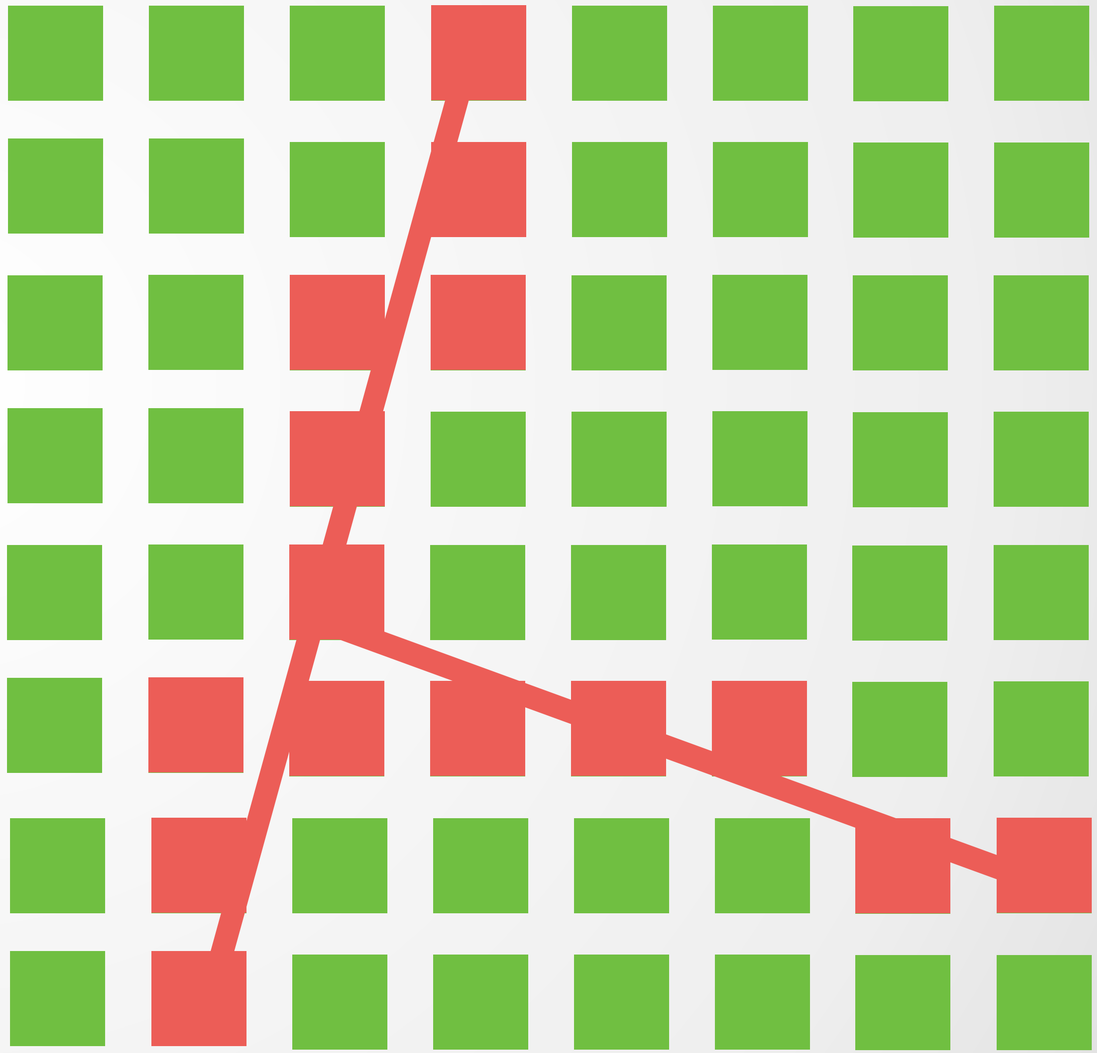
[Sigmund, A 99 line topology optimization code written in Matlab]

Topology Optimization

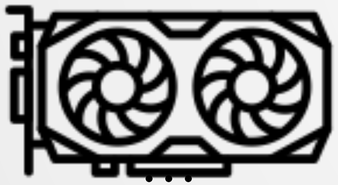


Grid Representation of Density





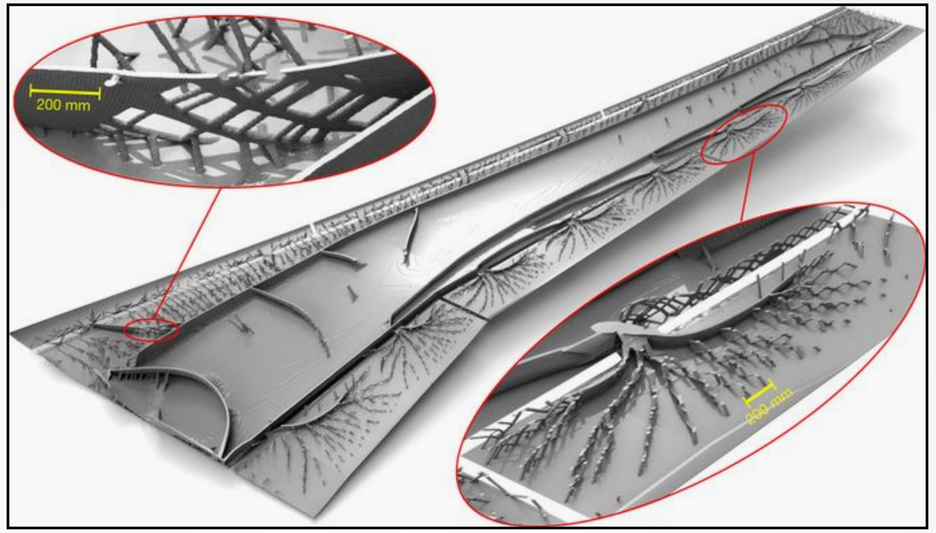
14 million voxels

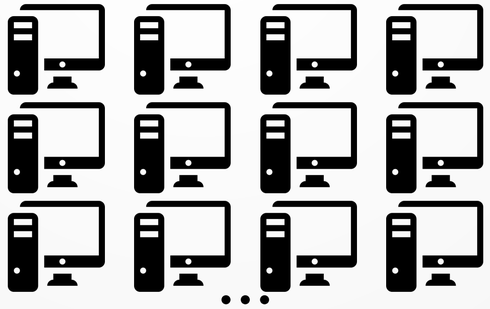


Single GPU

[Wu et al. 2017]

1 billion voxels



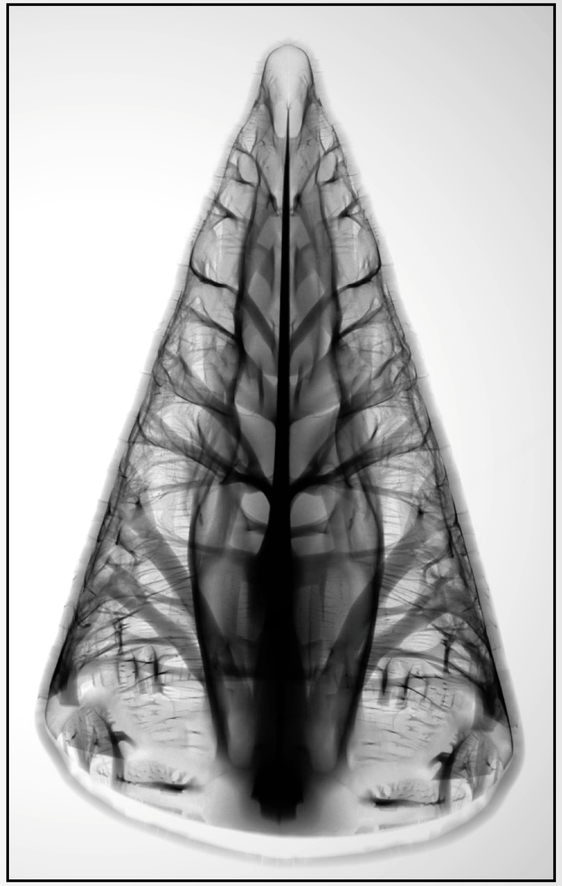


HPC Cluster

8000 cores

[Aage et al. 2017]

1 billion voxels



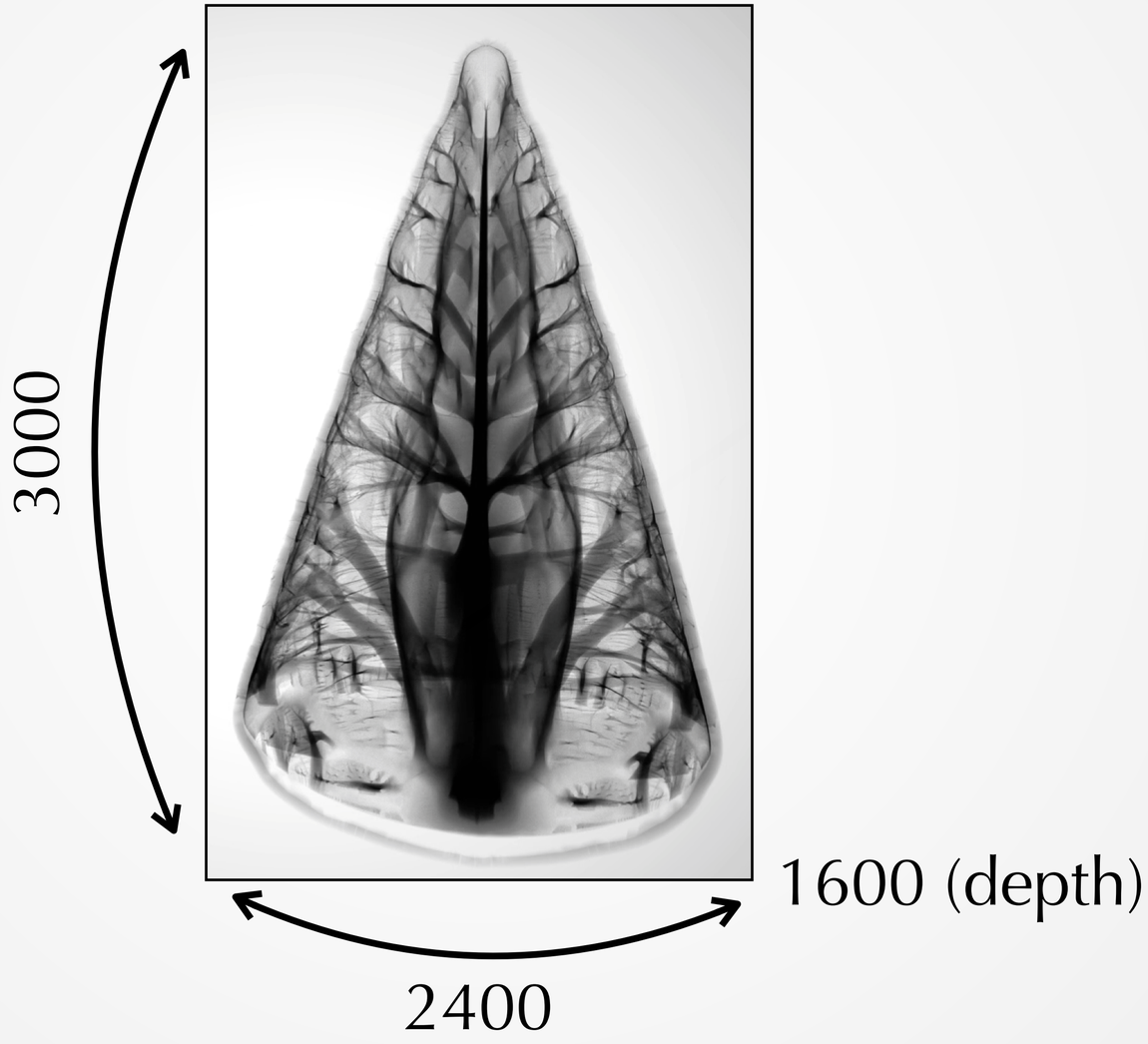
Single Workstation

56 cores

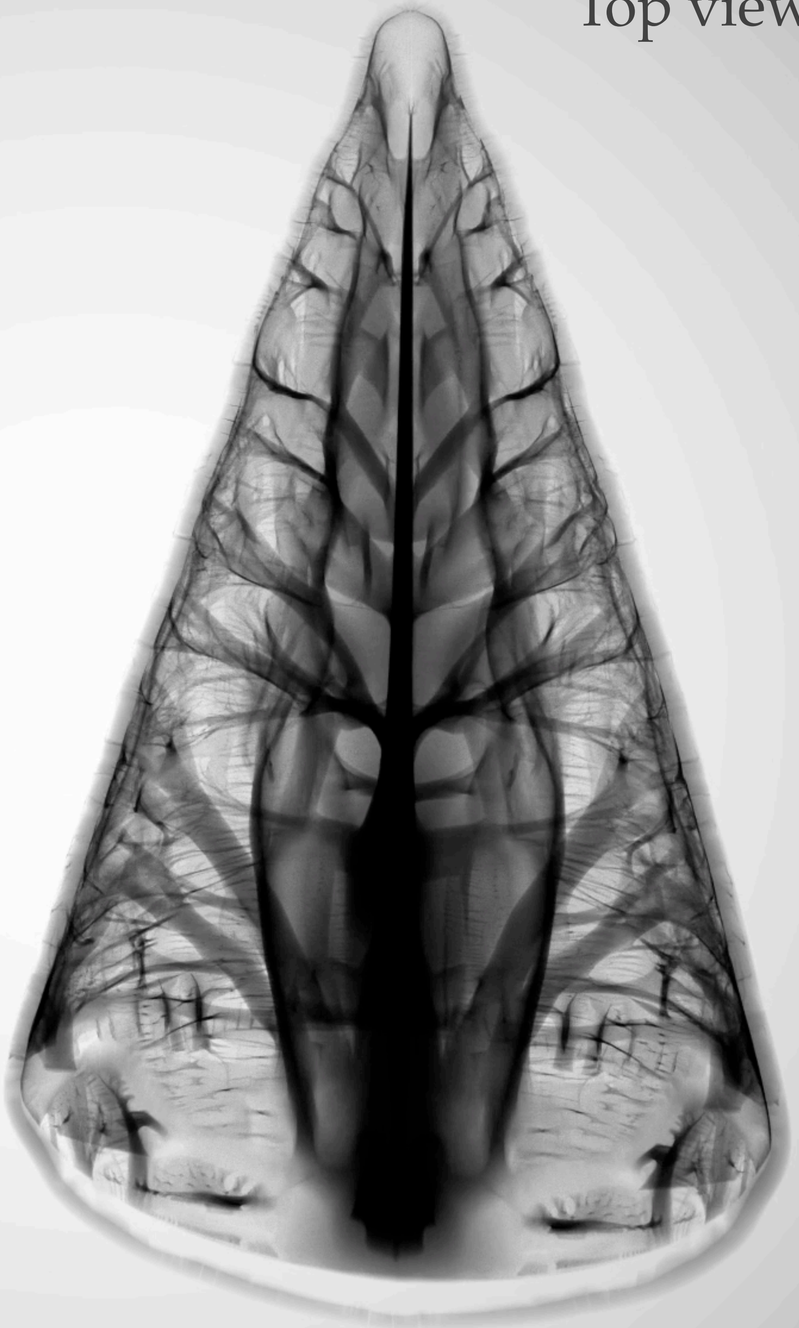
Ours

11,520,000,000 background voxels (3TB memory)

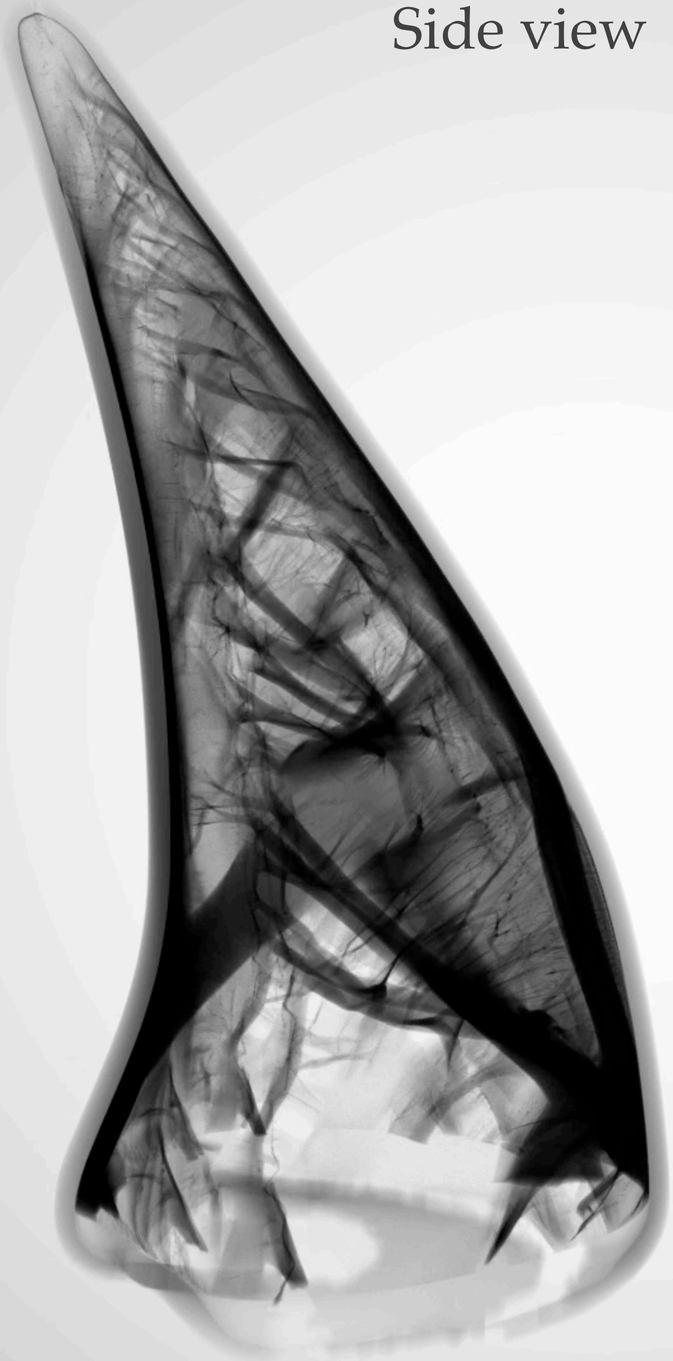
1,040,875,347 **active** voxels (8.6%)



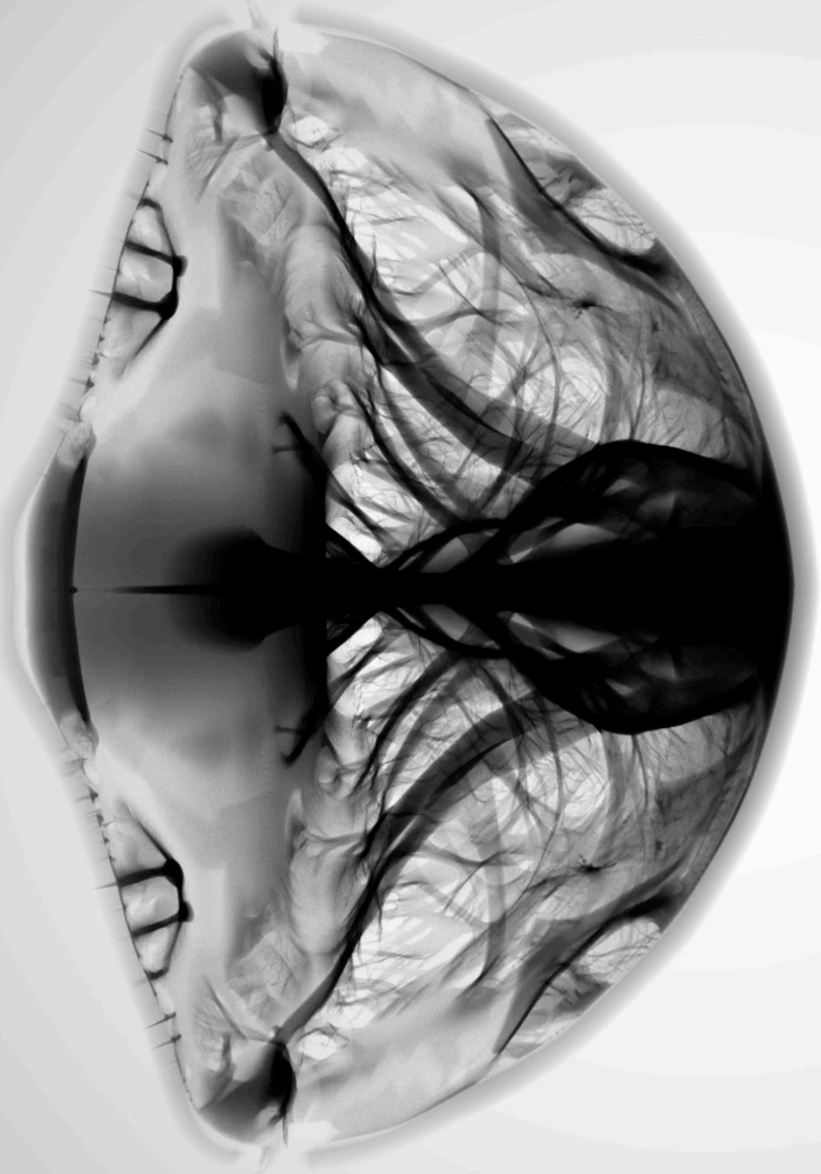
Top view



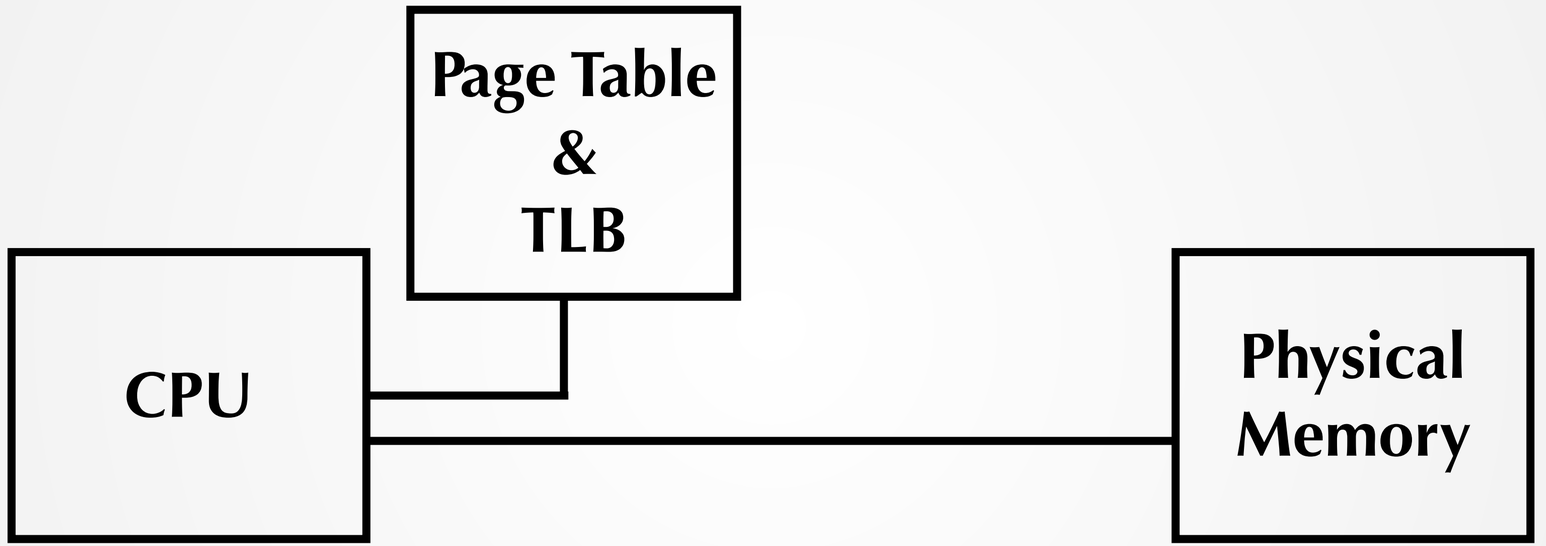
Side view

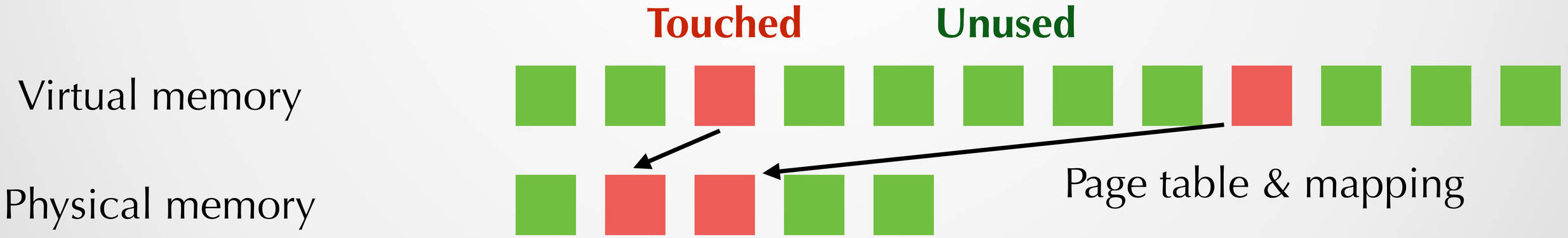


Back view



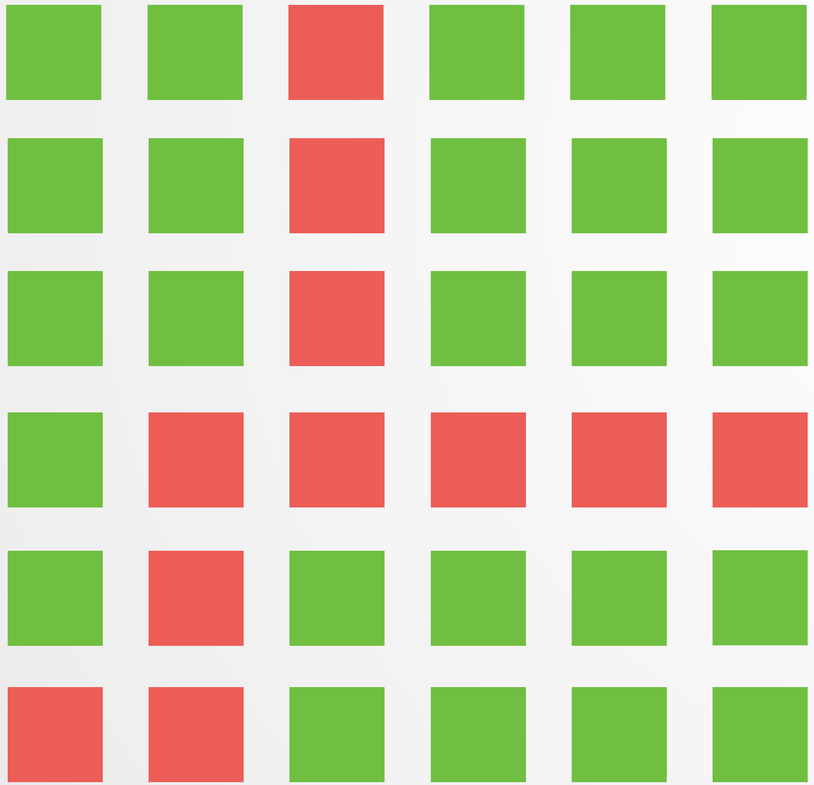
Virtual Memory System



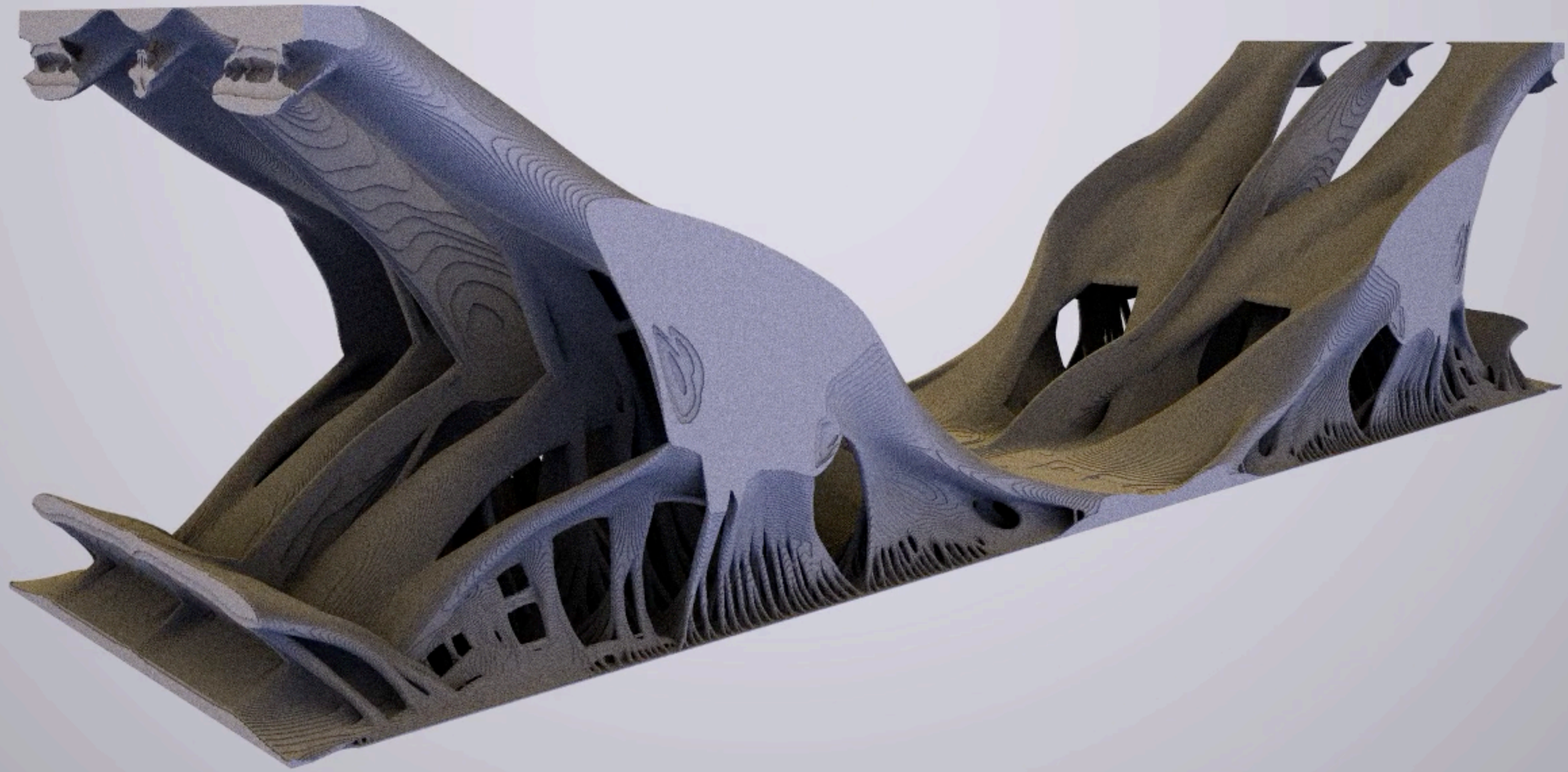


Sparsely Paged Grid [Setaluri et al. 2014]

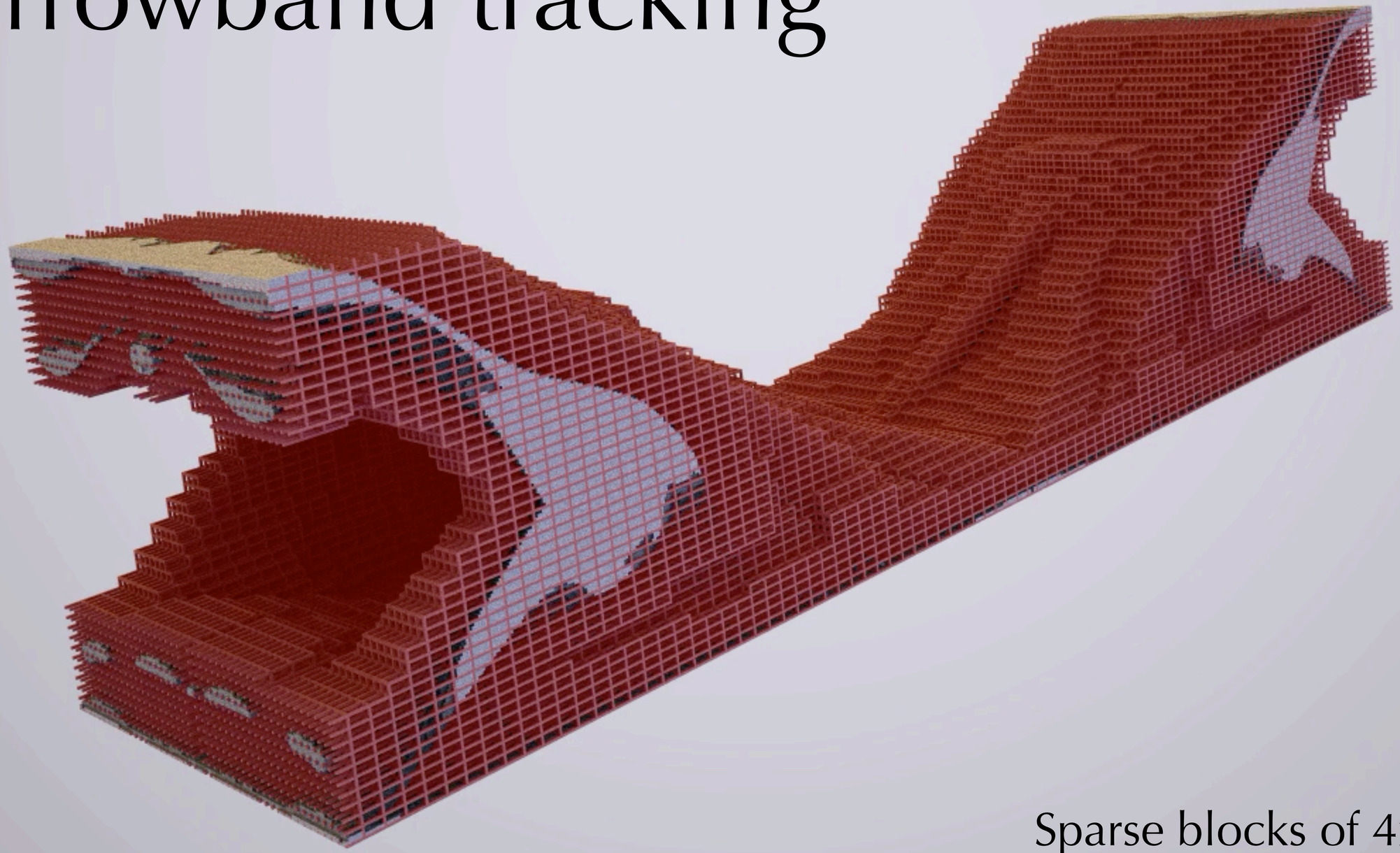
Virtual memory



Physical memory

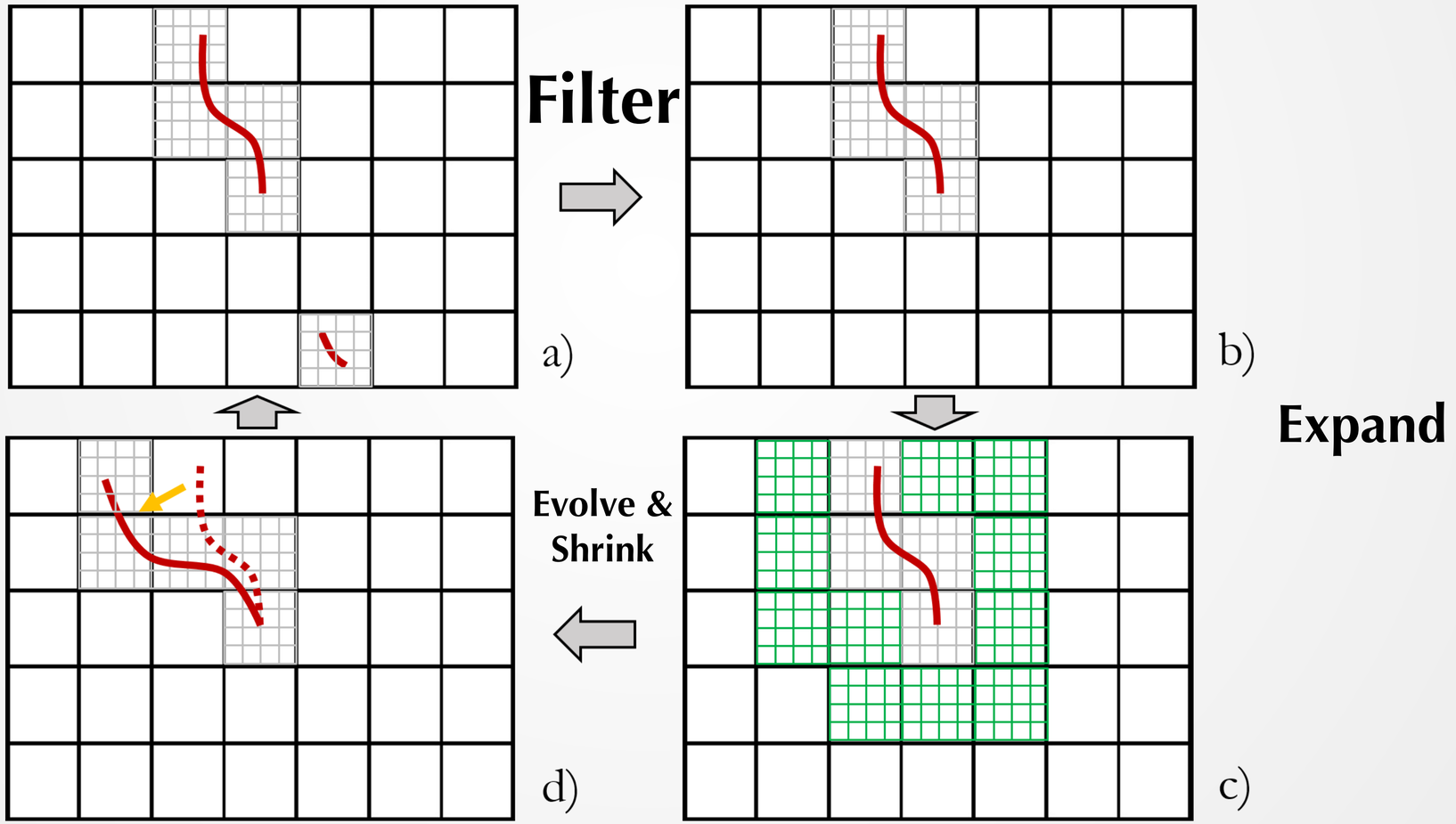


Narrowband tracking

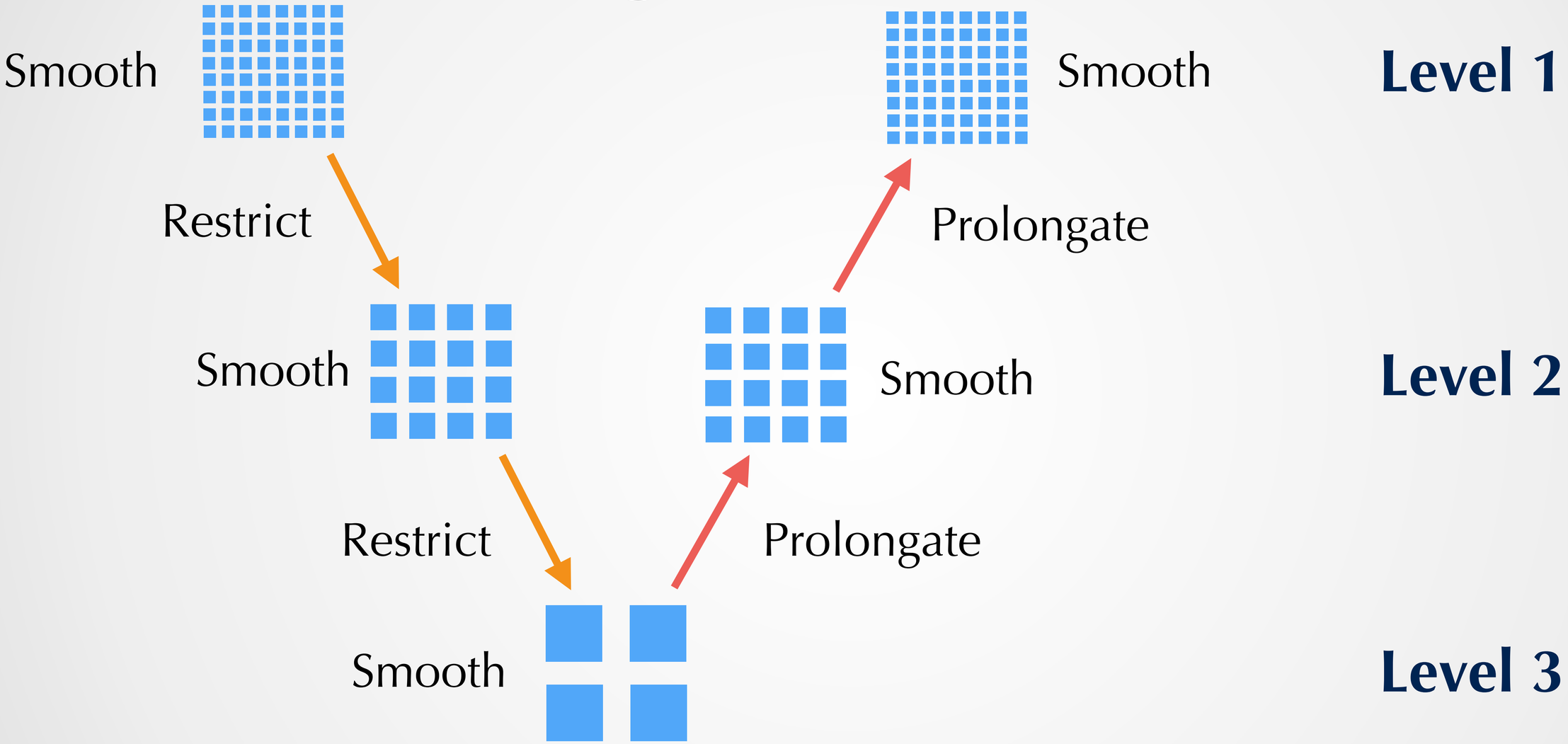


Sparse blocks of 4x4x8 nodes
Up to 3.1x acceleration

Narrowband Evolution



Multigrid FEM Solver



MGPCG Time Complexity: O(n)

Aggressively Optimized Multigrid FEM Solver

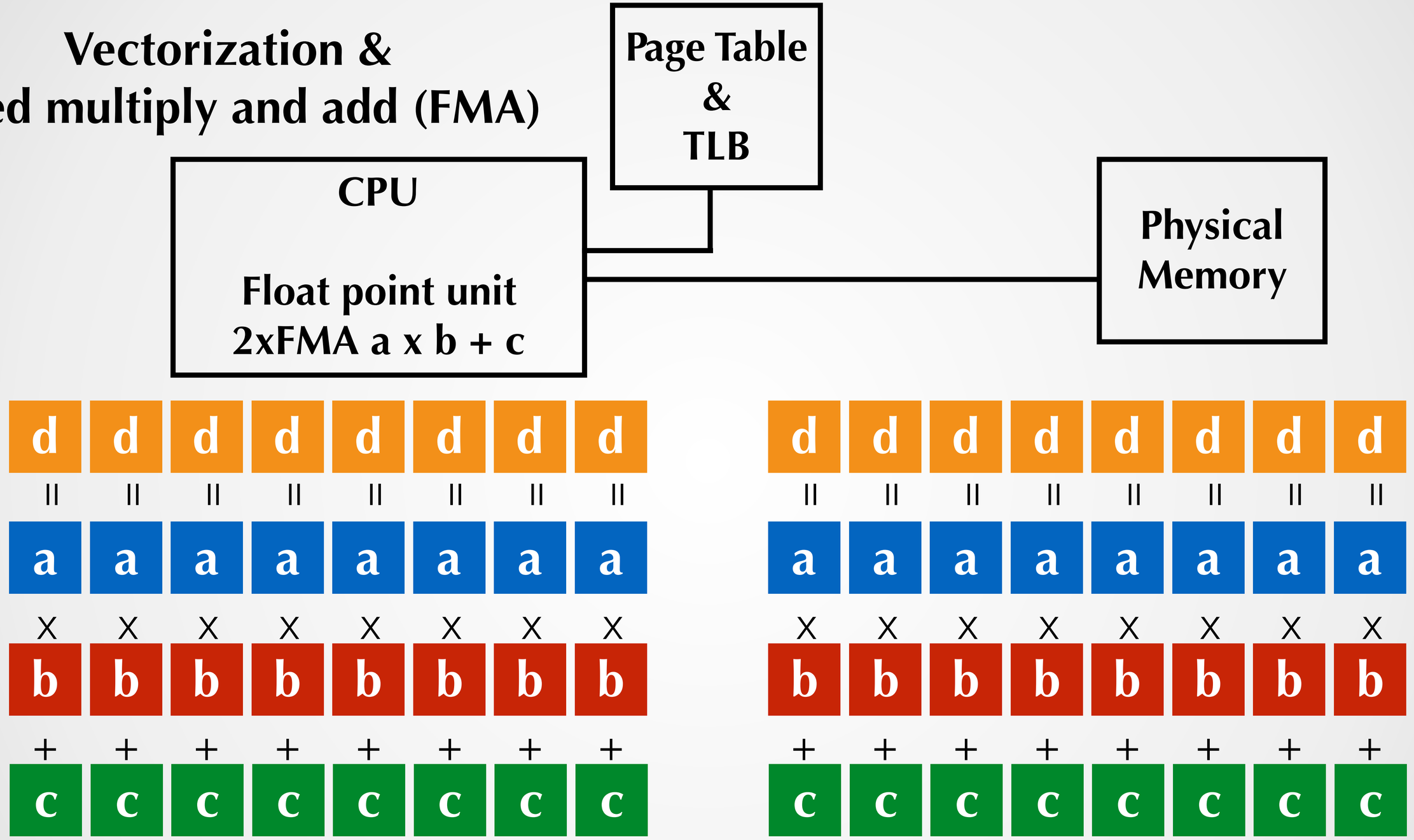
Optimized fine-level operator

Matrix-free Galerkin-coarsened operator construction

Improved eight-color Gauss-Seidel smoother

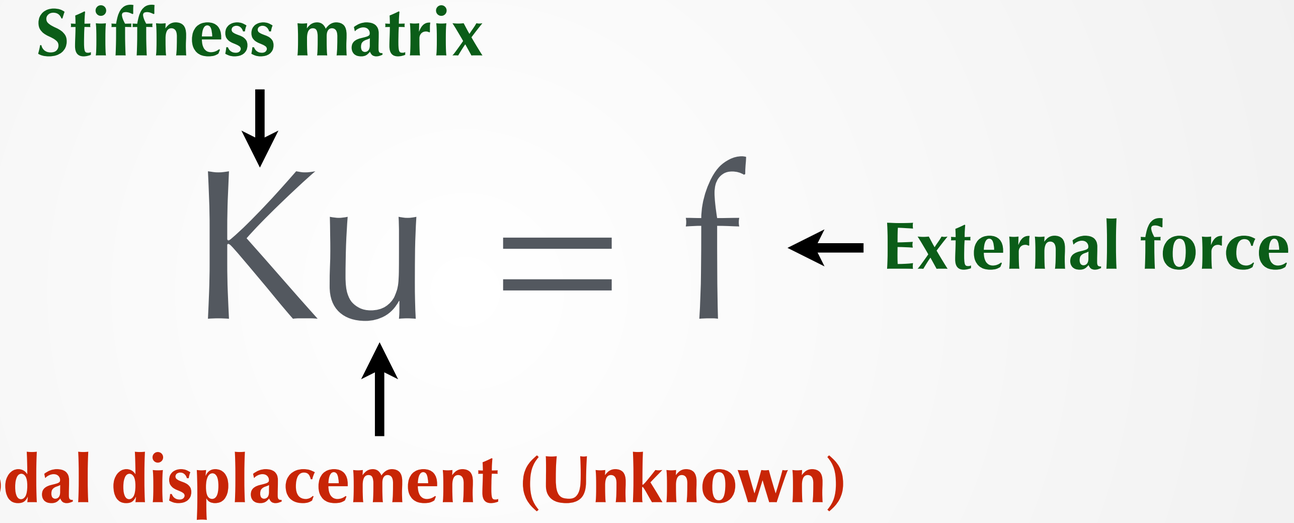
...

Vectorization & Fused multiply and add (FMA)



4.2G Hz x 2 FMA/cycle x 16 FLOPS/FMA x 4 cores = 538G FLOPs

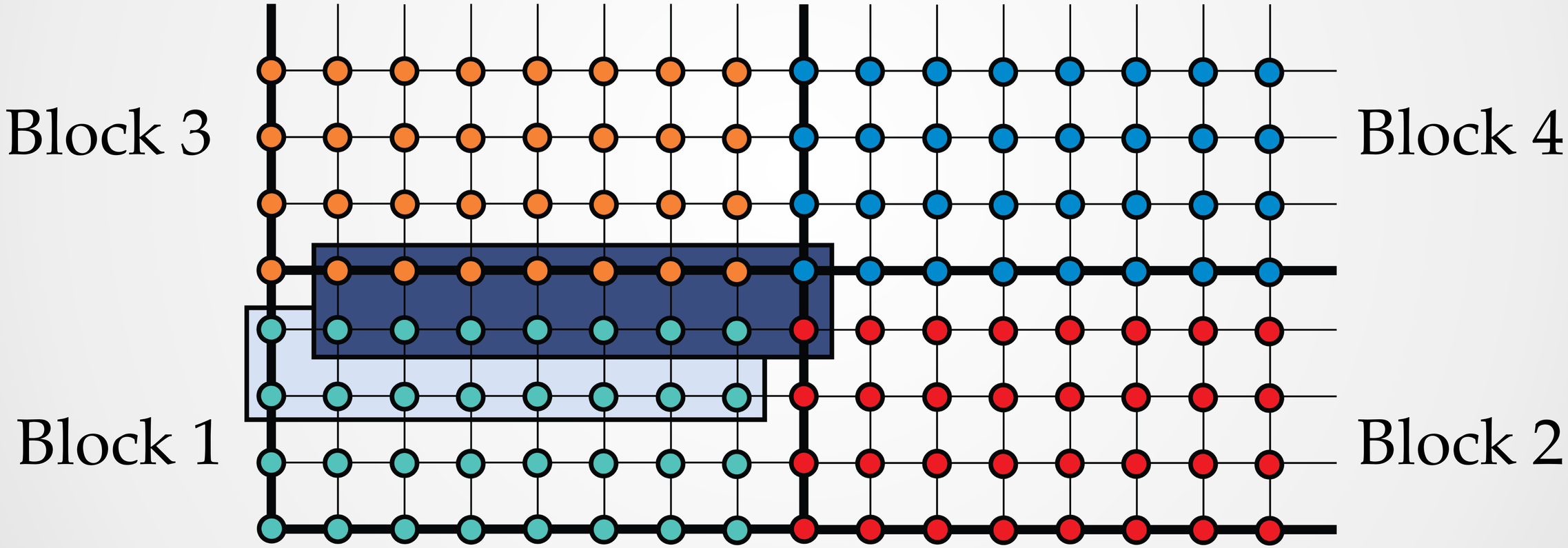
Fine Level Stiffness Operator



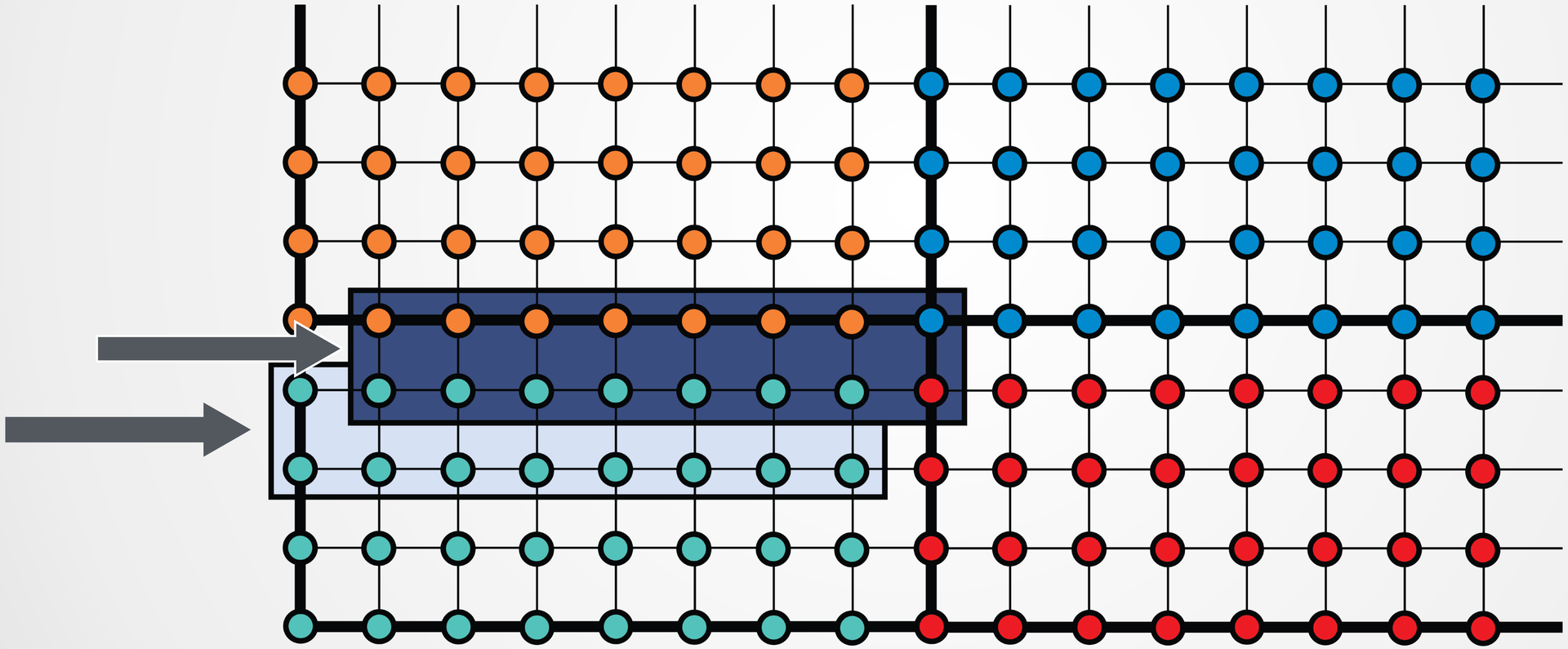
Matrix-Free

$2 \times 24^2 = \mathbf{1{,}152}$ vectorized FMA inst. per node

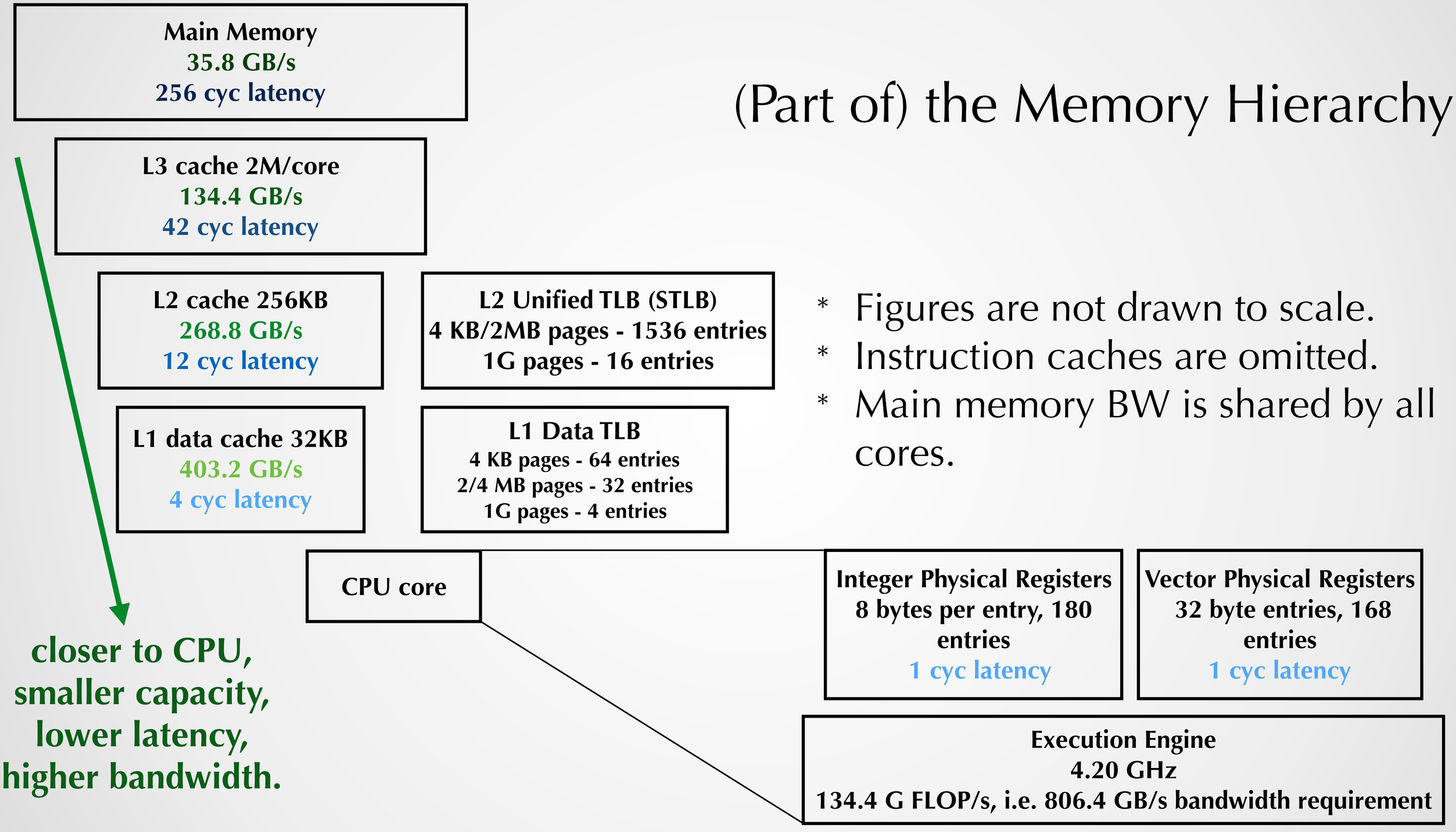
Fine Level Stiffness Operator



Fine Level Stiffness Operator



(Part of) the Memory Hierarchy



* Figures are not drawn to scale.
* Instruction caches are omitted.
* Main memory BW is shared by all cores.

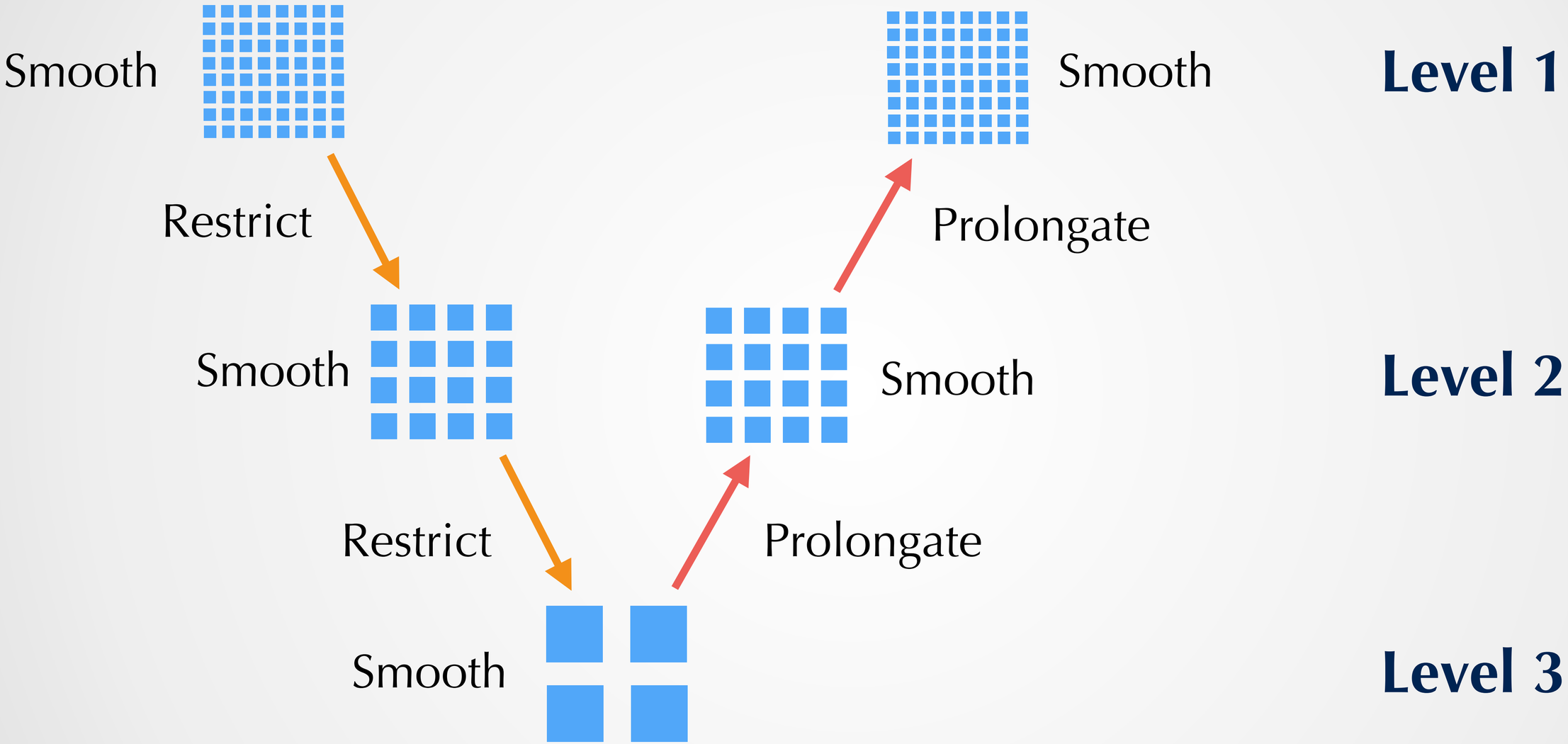
Fine Level Operator

SIMD **Gather** is expensive

Access pattern is regular

Blend is cheap

```
template <int di,int dj,int dk,class SPG_array>
__m512 VectorGet<di,dj,dk>(SPG_Array a,int64_t offset)
```


Smooth
Restrict
Smooth
Restrict
Smooth
Prolongate
Smooth
Prolongate
Smooth
Level 1
Level 2
Level 3

Matrix-Free Galerkin Coarsening

$$\mathbf{K}^{2h} = \mathbf{R}_{2h \to h}\, \mathbf{K}^{h}\, \mathbf{P}_{h \to 2h}$$

Matrix-Free Galerkin Coarsening

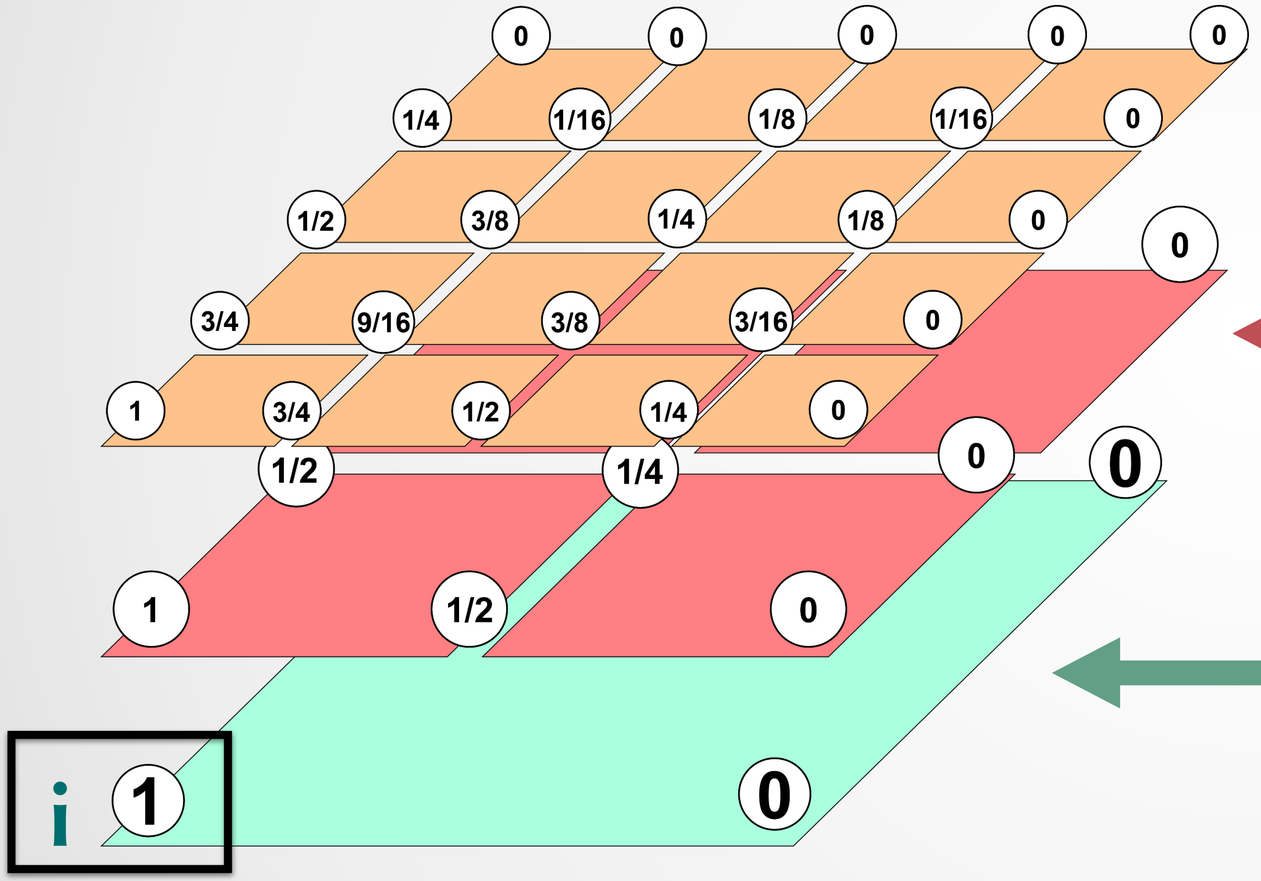
Top level is matrix free

Matrix-matrix multiply is **memory bound**

Matrix-Free Galerkin Coarsening

$$\mathbf{K}^{4h} e_i = \mathbf{R}_{2h \to 4h} \mathbf{R}_{h \to 2h} \mathbf{K}^{h} \mathbf{P}_{h \to 2h} \mathbf{P}_{2h \to 4h} \, e_i$$

Matrix-Free Galerkin Coarsening



$\mathbf{P}_{h\to 2h}\mathbf{P}_{2h\to 4h}\,\mathbf{e}_i$

$\mathbf{P}_{2h\to 4h}\,\mathbf{e}_i$

$\mathbf{e}_i$

$$\mathbf{K}^{4h}\mathbf{e}_i = \mathbf{R}_{2h\to 4h}\mathbf{R}_{h\to 2h}\mathbf{K}^{h}\mathbf{P}_{h\to 2h}\mathbf{P}_{2h\to 4h}\,\mathbf{e}_i$$

Matrix-Free Galerkin Coarsening

Recursive algorithm

Stack allocated variables

Cell material is read once/level

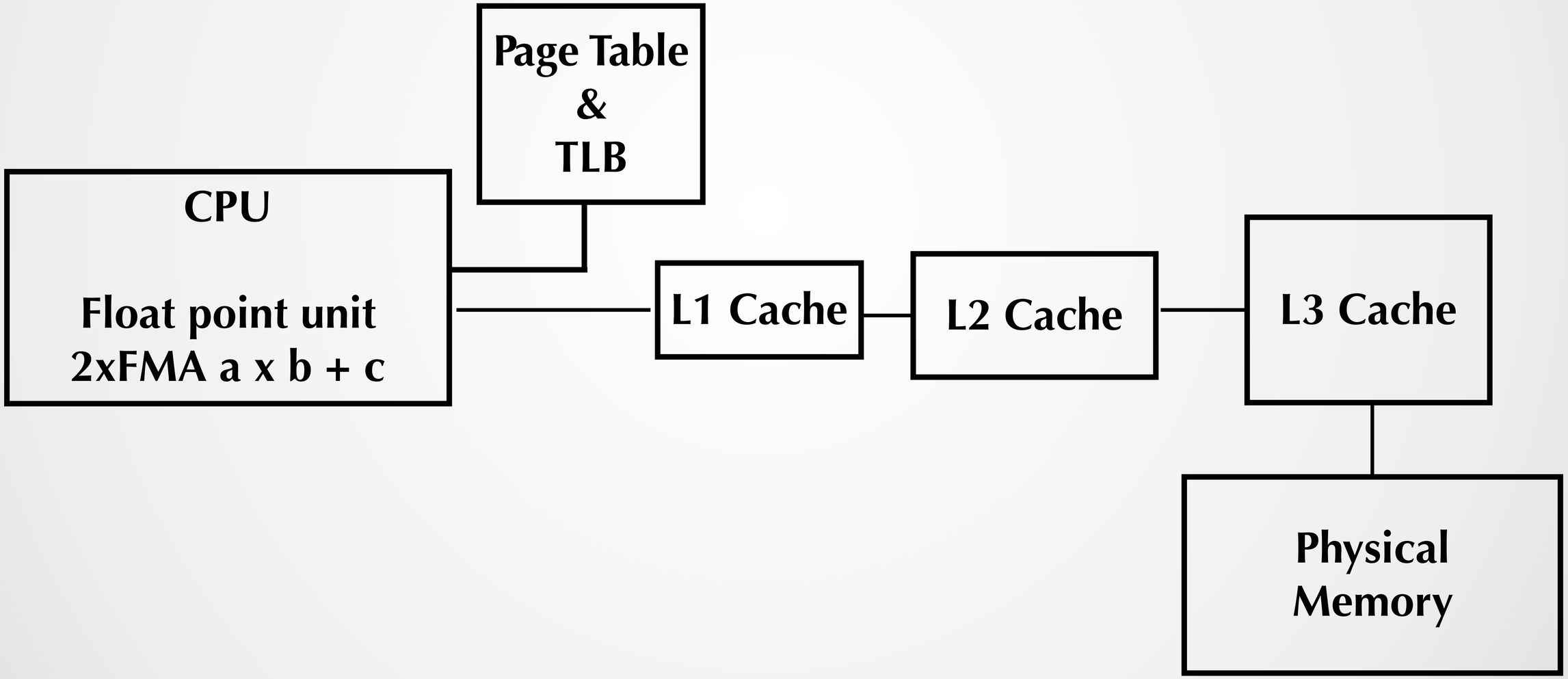
Total time: **113.9 sec for 1.04 Billion voxels**

~1.26 TFLOPS on Skylake SP

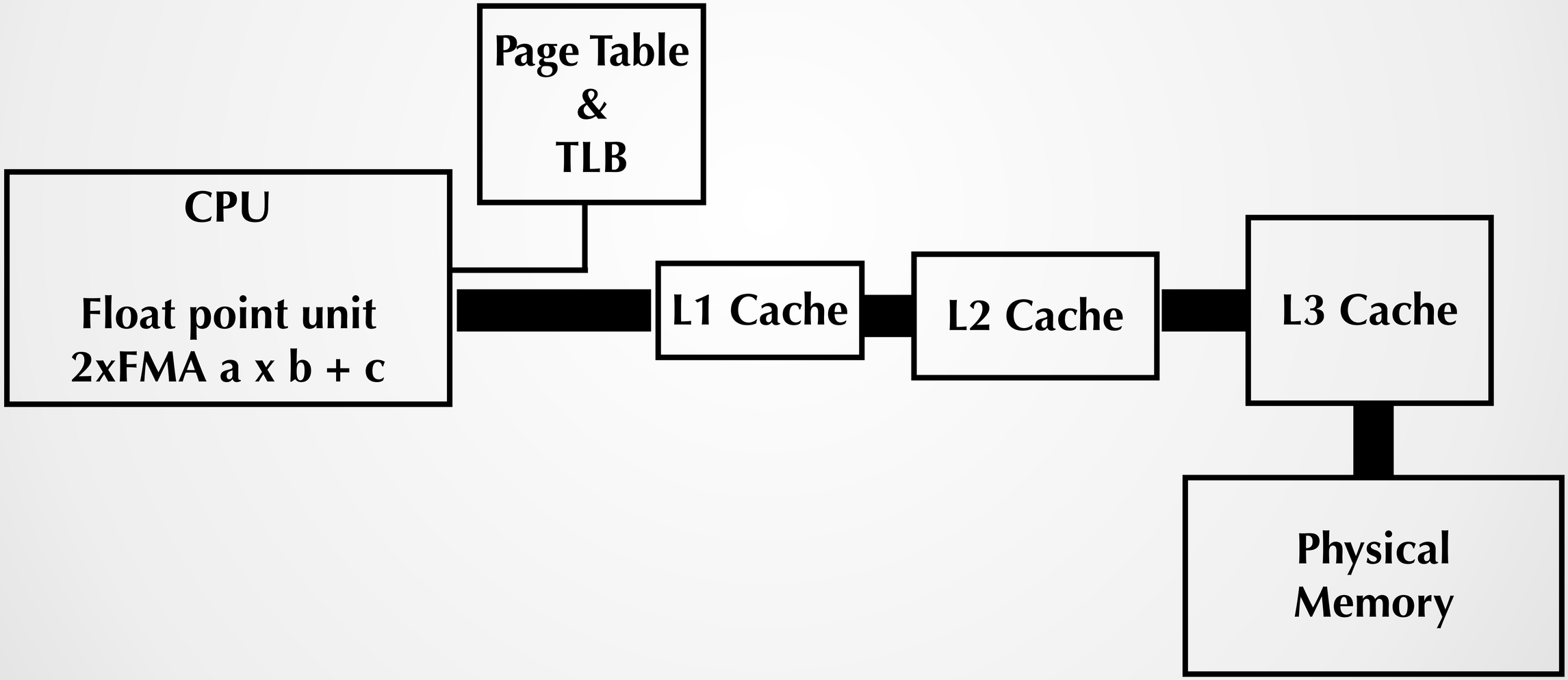
Modified Eight-Color Gauss-Seidel Smoother

Coupled Diagonal Smoothing

Shuffled Data Storage


Page Table
&
TLB
CPU
Float point unit
2xFMA a x b + c
L1 Cache
L2 Cache
L3 Cache
Physical
Memory

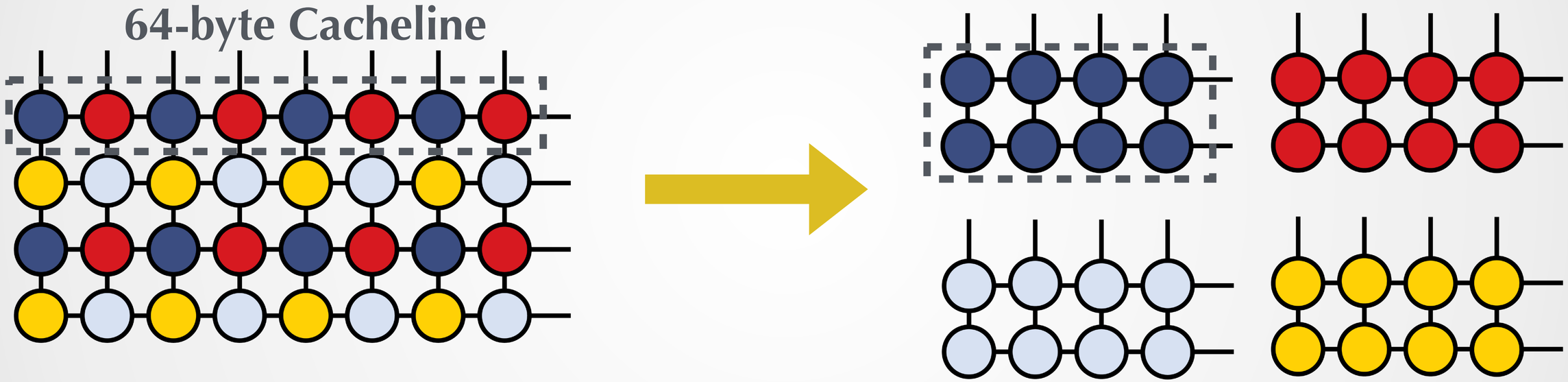
CPU talks to caches in the unit of 64B **cachelines**.



Modified Eight-Color Gauss-Seidel Smoother

Original Data Layout

Shuffled Data Layout

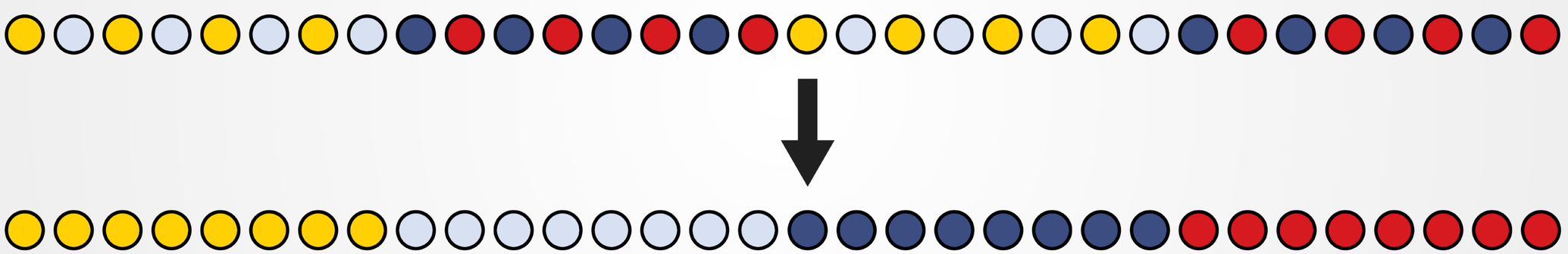


Comparing to matrix-free multiply

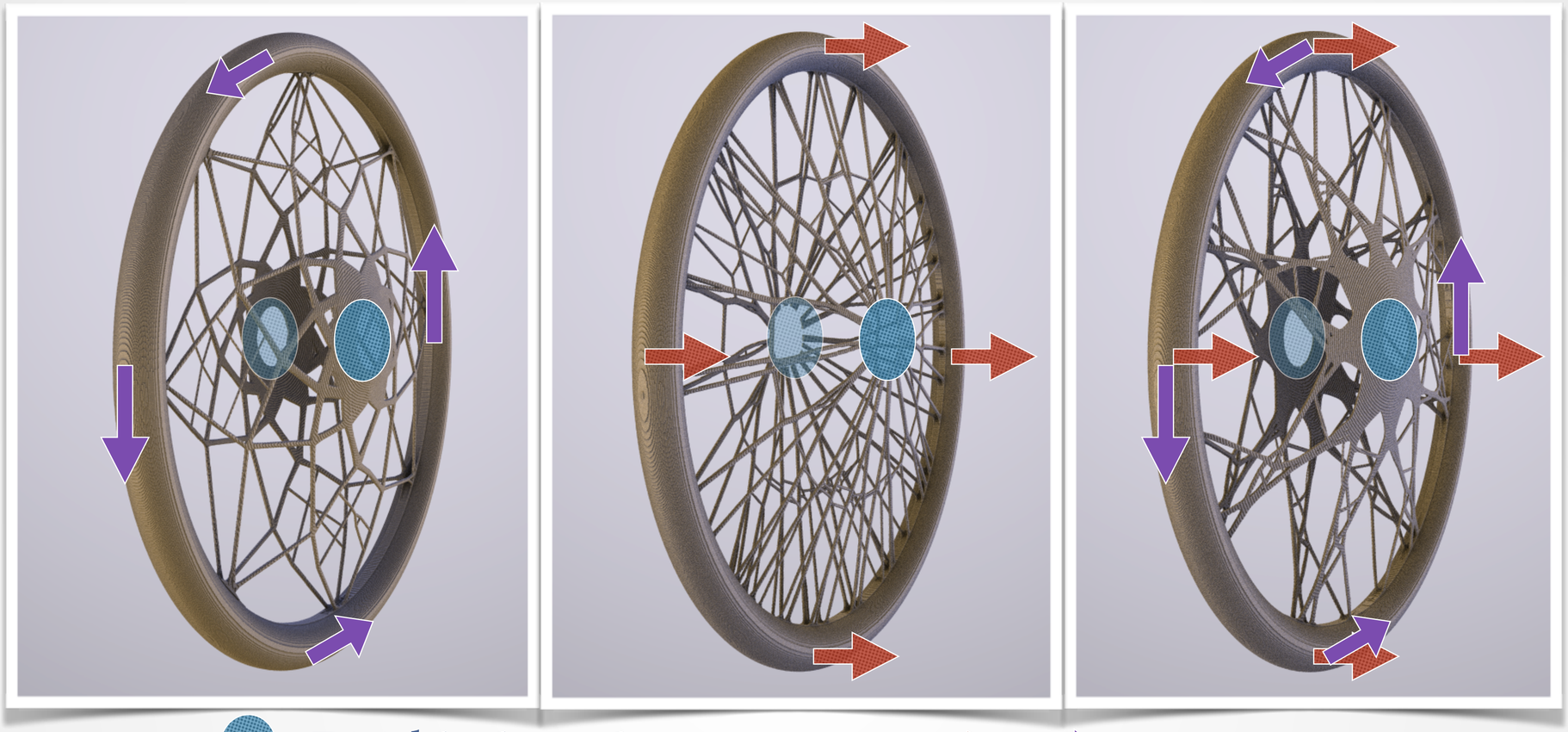
same amount memory access but 1/8 computation

Memory bound

Modified Eight-Color Gauss-Seidel Smoother



Effective bandwidth: 68GB/s (out of 120 GB/s)

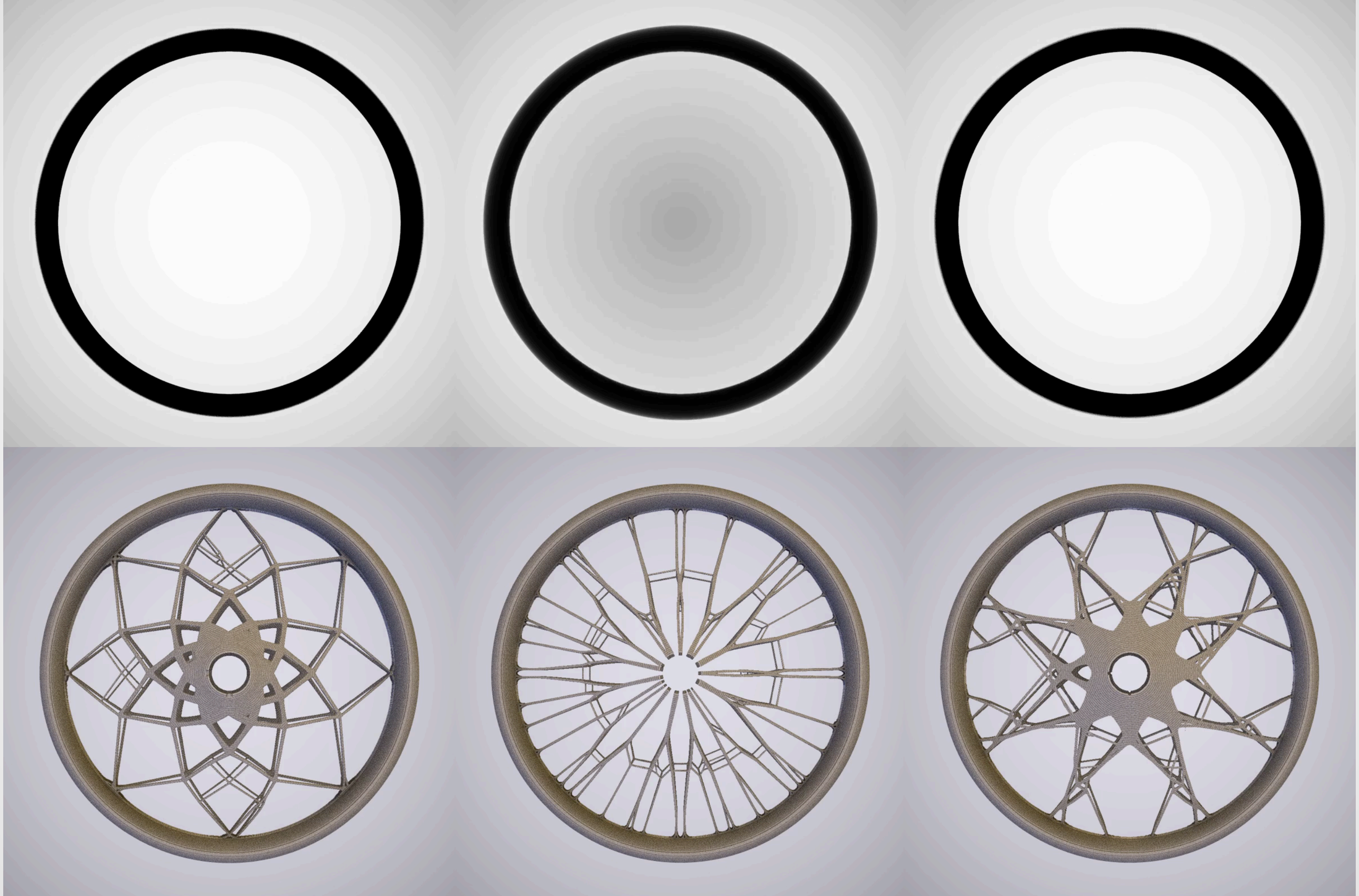


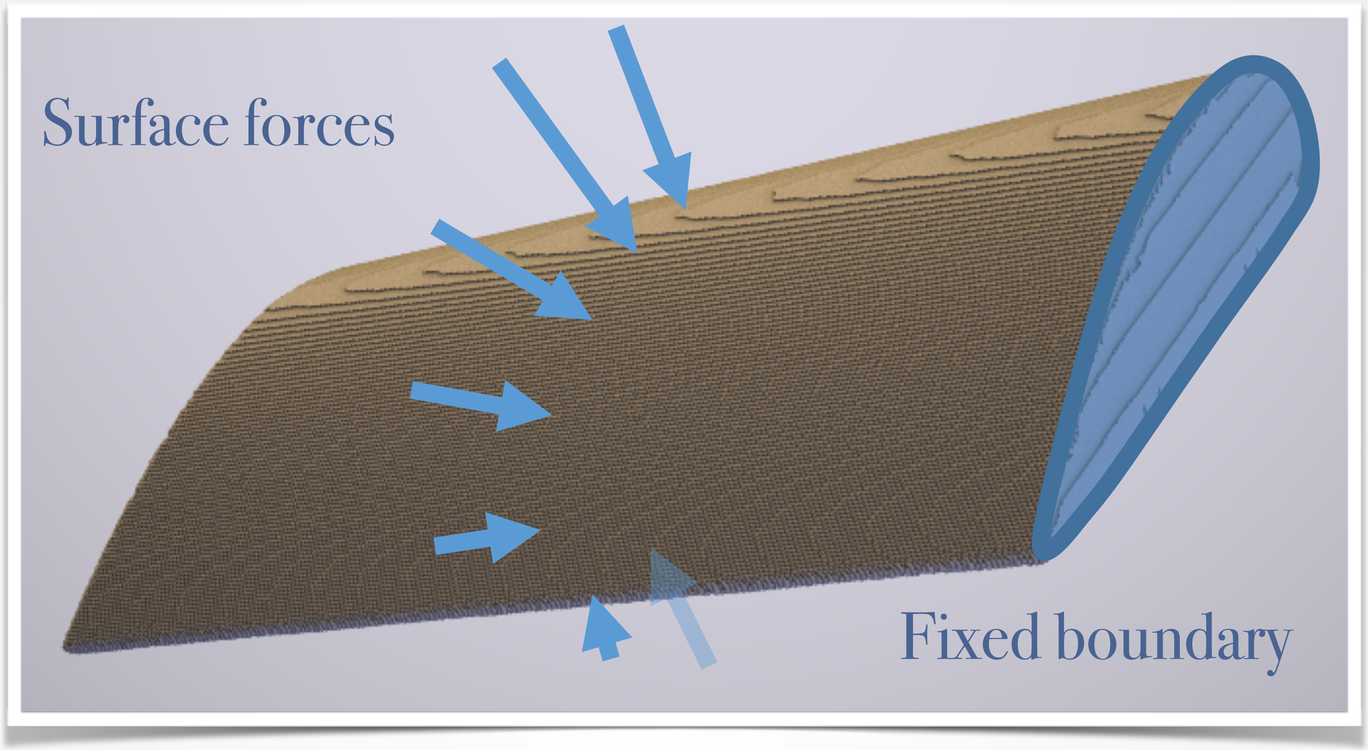
: Dirichlet boundary

: loads

Bicycle Wheels

53.56M voxels
984x984x204 background grid
1% volume fraction
~65 TopOpt iterations



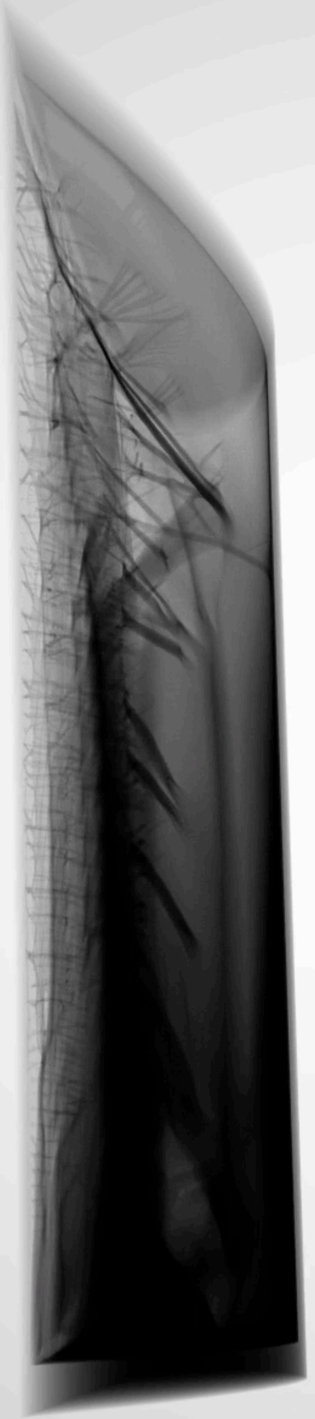


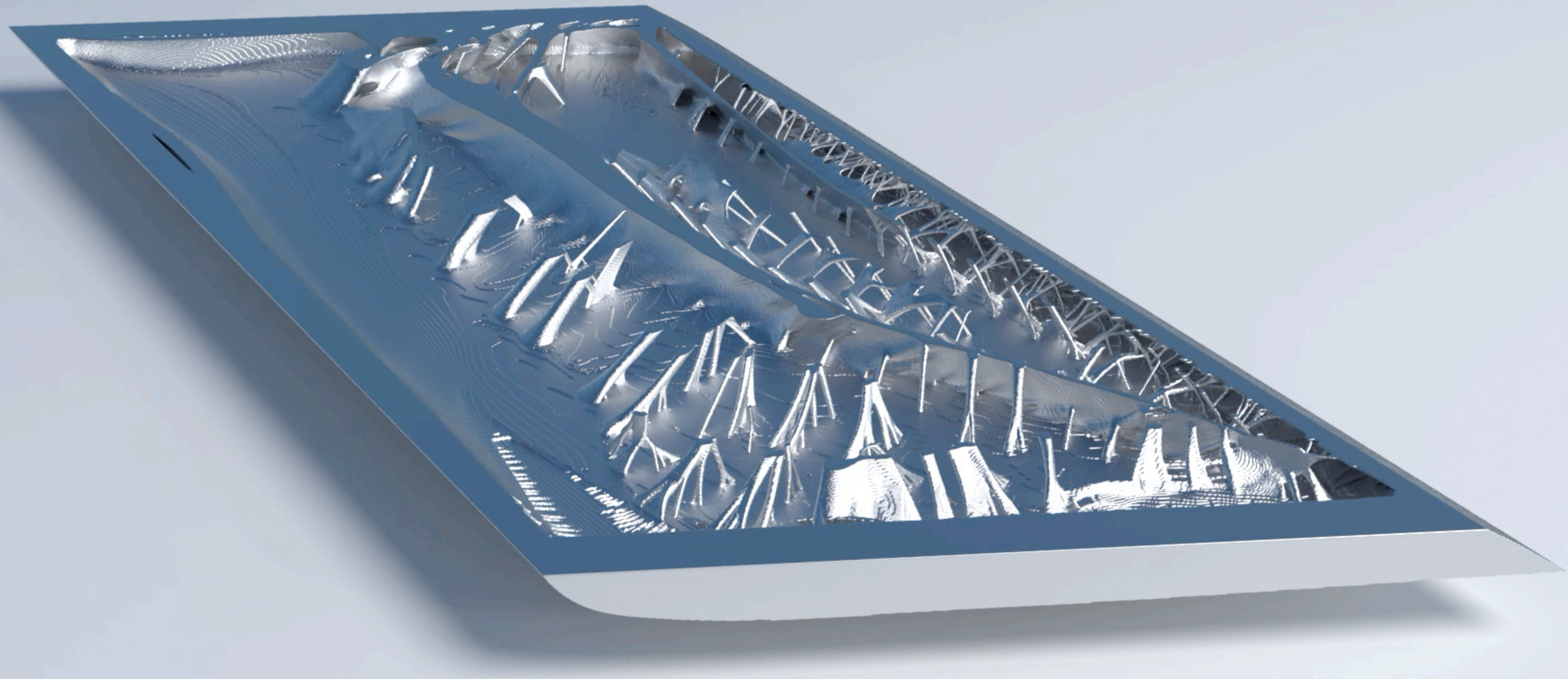
Plane Wing

401.53M voxels
1696x342x1971 background grid
20% volume fraction
43 TopOpt iterations

Top view

Side view



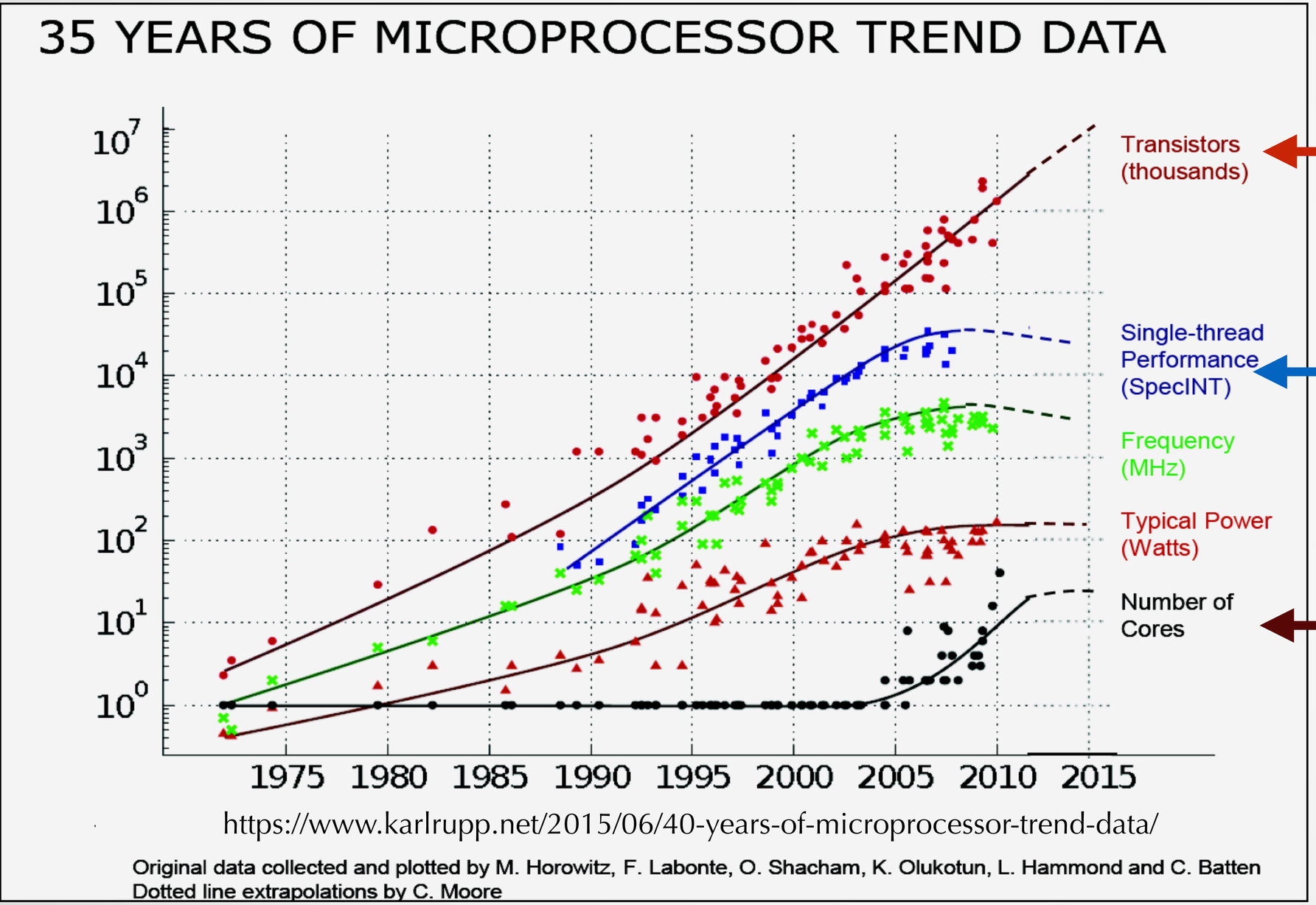


Cut-away view

Summary:

1. Use a **sparse** grid to compactly store irregular geometry
2. Track the evolution of the structure using a **narrowband**
3. Do aggressive **low-level optimization** to speed up FEM solve

…and parallelism.



Moore's law continues

Single-core performance stops growing

Instead of "better" cores, we have "more" cores (with wider SIMD)

Conclusion

Data Layout:	Narrowband domain evolution
Vectorization & Data Layout:	Optimized fine-level operator
Vectorization:	Matrix-free Galerkin-coarsening
Data Layout:	SPGrid-specific sparse matrix storage
Vectorization & Data Layout:	Improved eight-color Gauss-Seidel smoother
Vectorization & Data precision:	Mixed-precision scheme

Performance in numerical computation is mostly about **data**.
Good **data layout** and **effective vectorization** are keys to performance.

We need better language & compiler to reduce the development cost
for high-performance computer graphics.

…and we did write a compiler for that!

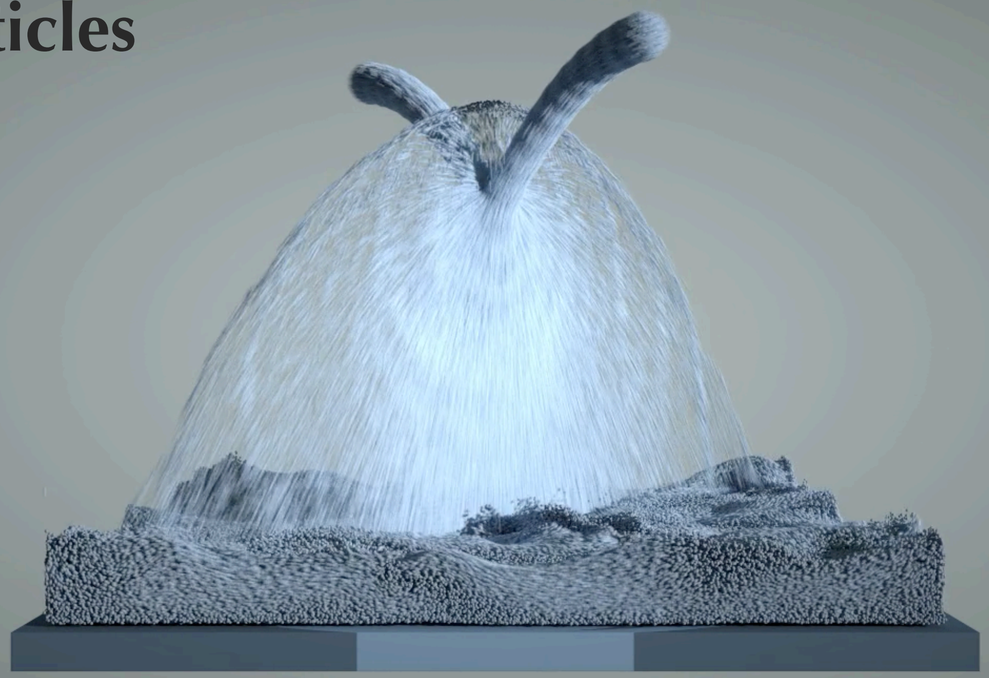
Taichi: a language for high-performance computation on spatially sparse data structures

(SIGGRAPH Asia 2019)

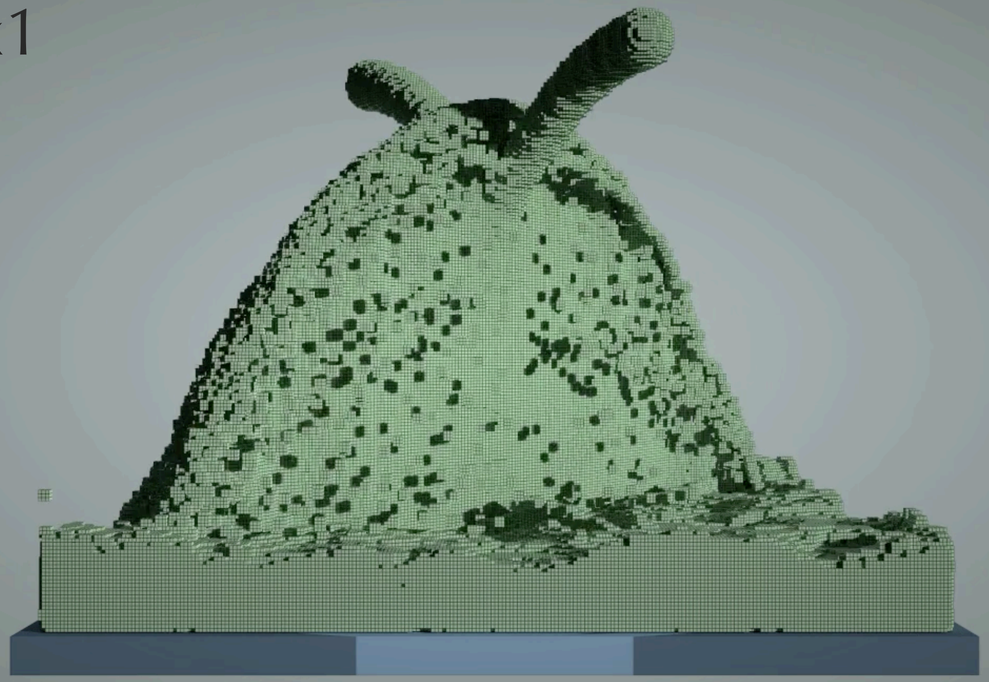
Write high-performance CPU/GPU solvers on sparse grids, in a programming language like Python!

https://github.com/taichi-dev/taichi

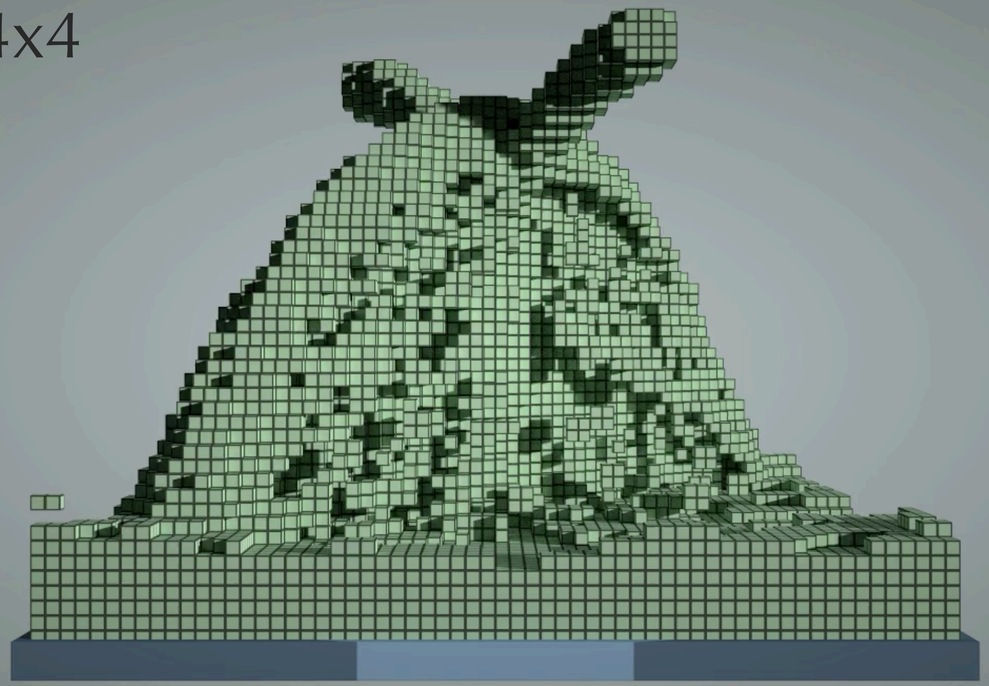
Particles



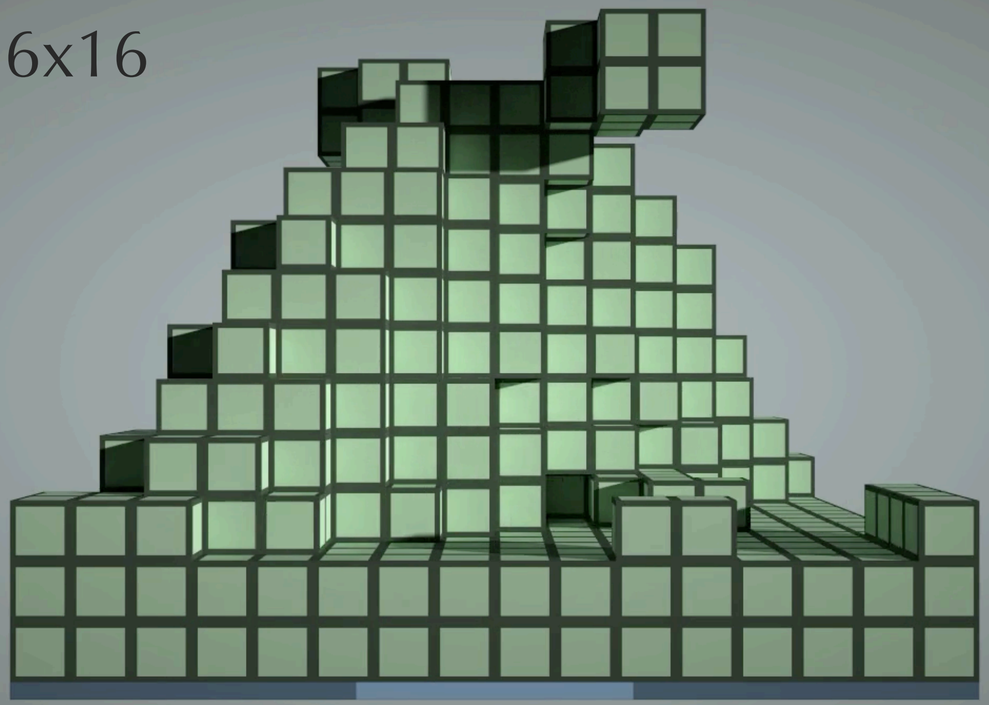
1x1x1



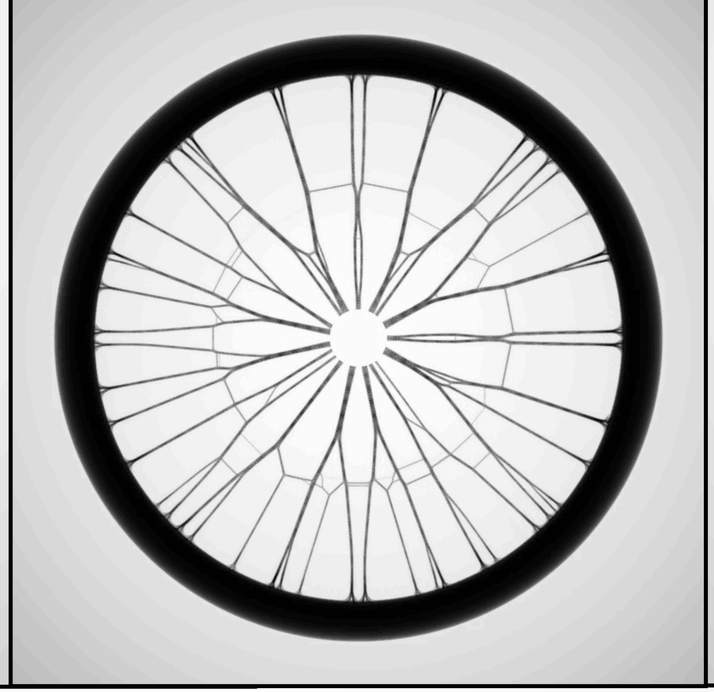
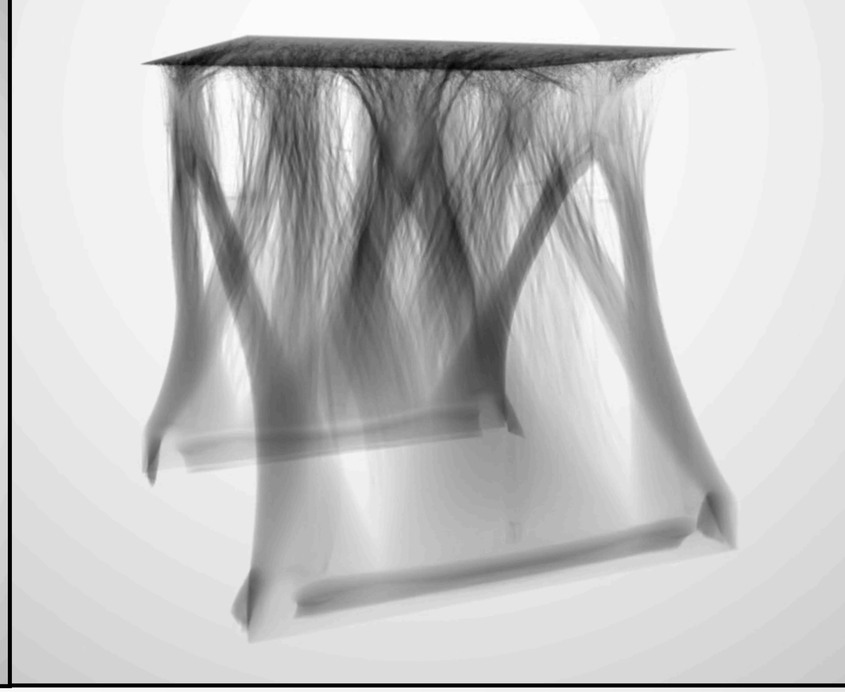
4x4x4



16x16x16





Code: https://github.com/yuanming-hu/spgrid_topo_opt

Thanks!

Questions are welcome!