# A Fast Cache-Oblivious Mesh Layout with Theoretical Guarantees

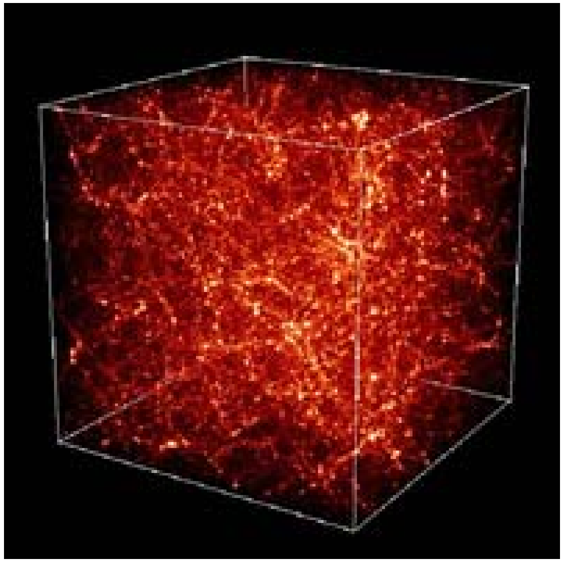
Vincent Danjean, Bruno Raffin,
**Marc Tchiboukdjian**

INRIA INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE
CNRS CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
GRENOBLE UNIVERSITÉS
LIG
cea

# Visualization & Massive Data Sets

TERA-10

*CEA supercomputer*
*$19^{th}$ Top500*
*9968 Itanium2*
*60 Tflops*
*30 TB of memory*
*1 PB of disk space*

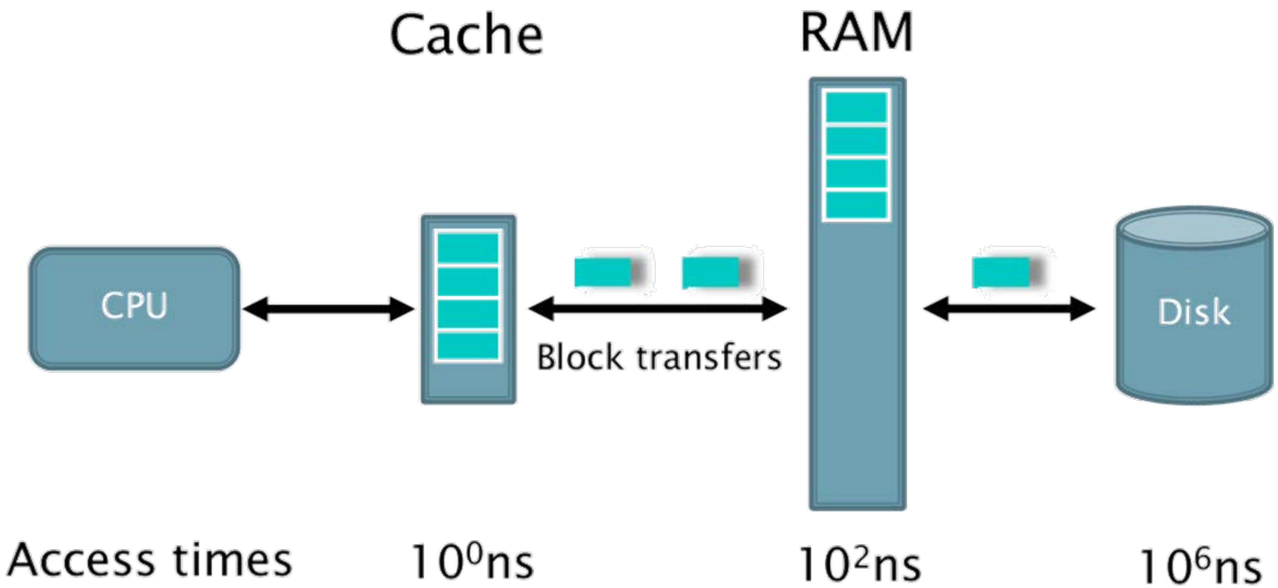


Simulation of half the observable universe
*50 TB mesh*

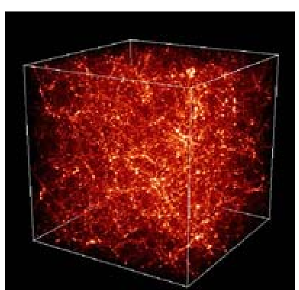
*We need good I/O performance to visualize such data sets*

# Cache-Efficient Ordering of a Mesh

Memory hierarchy




How to lay out a mesh in memory to minimize cache misses?



# Outline

- Cache-aware and cache-oblivious models
  example: matrix multiplication
- Previous work on mesh layouts
- Our algorithm
- Experiments

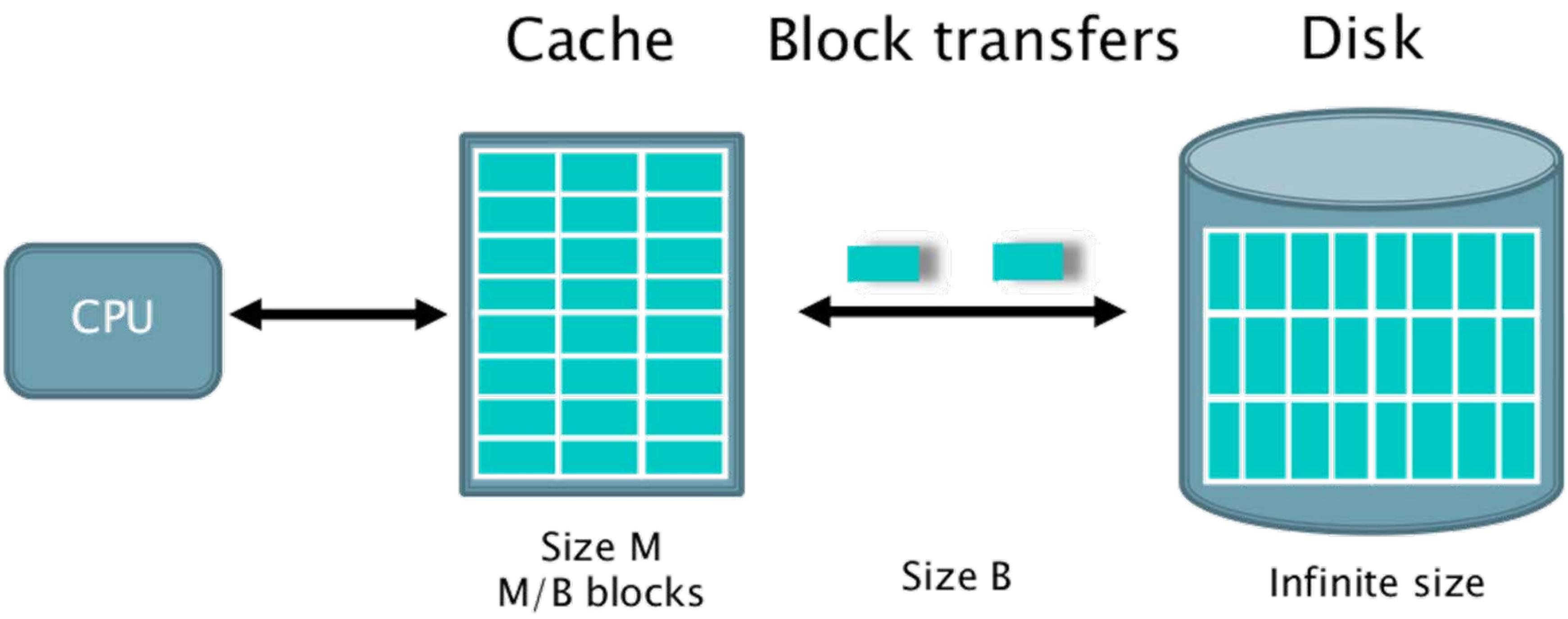
# Cache-Aware Model (CA)

[Aggarwal & Vitter 1988]

or external memory
out-of-core
disk access machine
I/O model

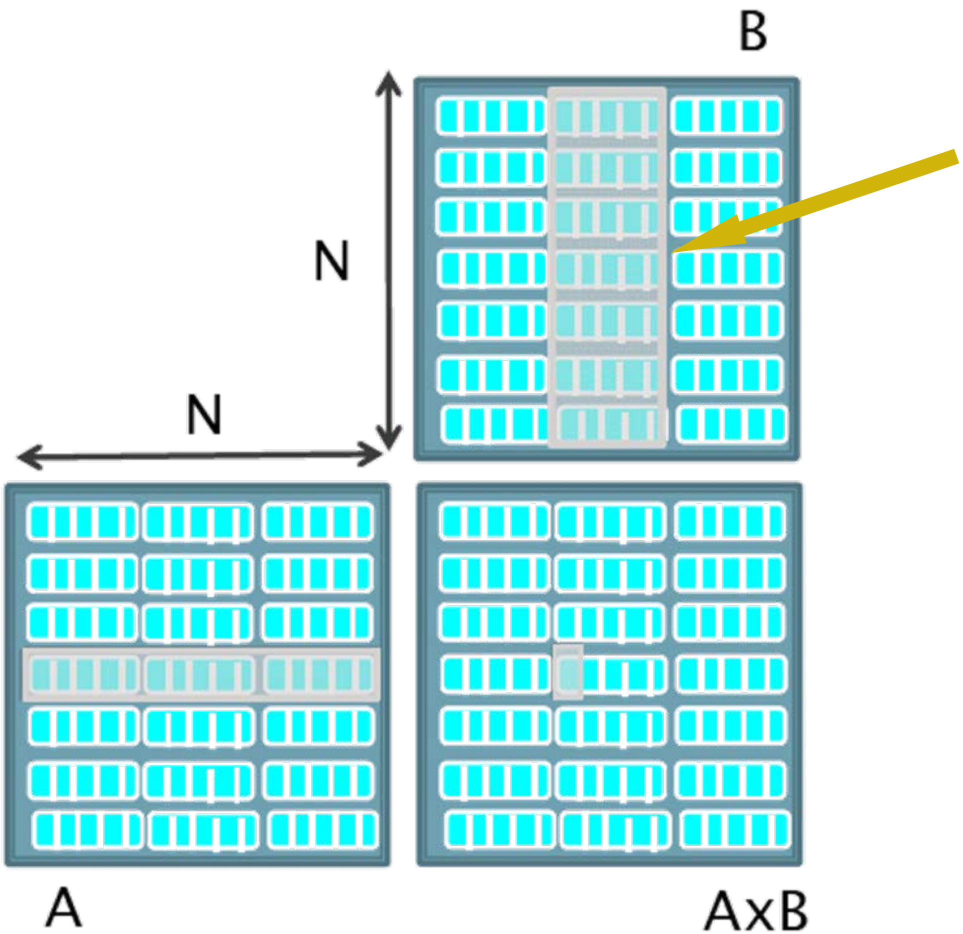
W: #operations

Q: #block transfers



# Multiplying in the CA Model

NxN matrices in row-major order: naive algorithm doesn't work




Memory accesses in B are suboptimal

$$W_{naive}(N) = O(N^3)$$

$$Q_{naive}(N) = O(N^3)$$

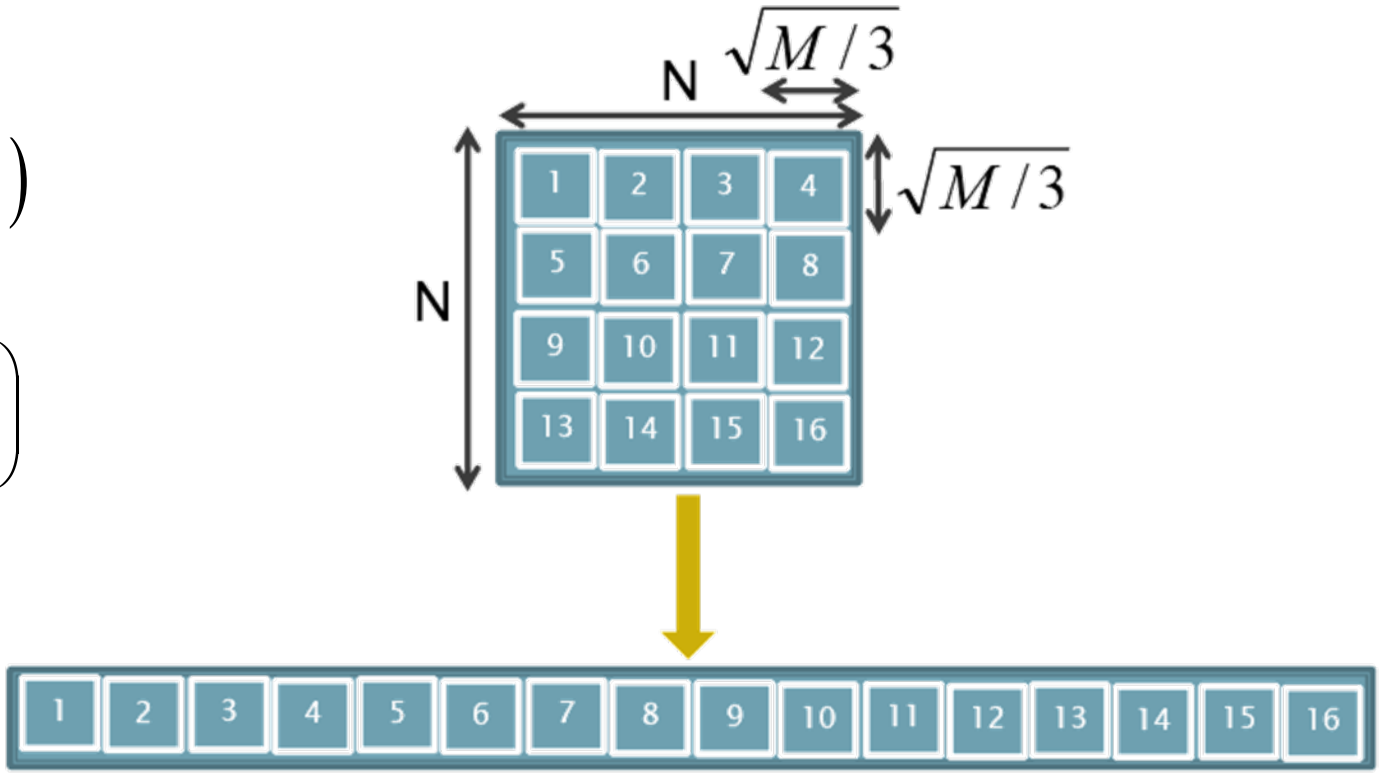
# Multiplying in the CA Model

NxN matrices in submatrices

Technique used in BLAS

$$Q_{naive}(N) = O\left(N^3\right)$$

$$Q_{CA}(N) = O\left(\frac{N^3}{B\sqrt{M}}\right)$$



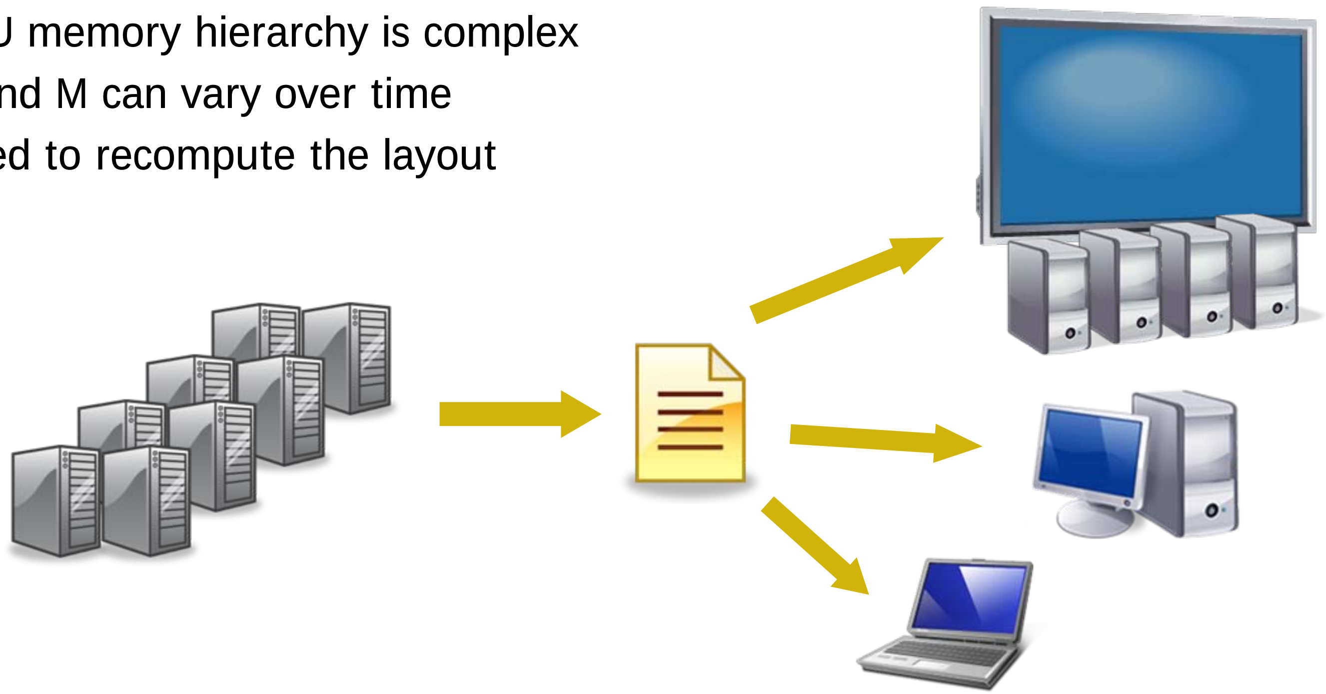
# Drawbacks of the CA Model

– Only two levels of the memory hierarchy
  - At least 4 levels on any modern CPU
  - Even deeper with multiprocessors computers

+ Efficient with all levels of the memory hierarchy

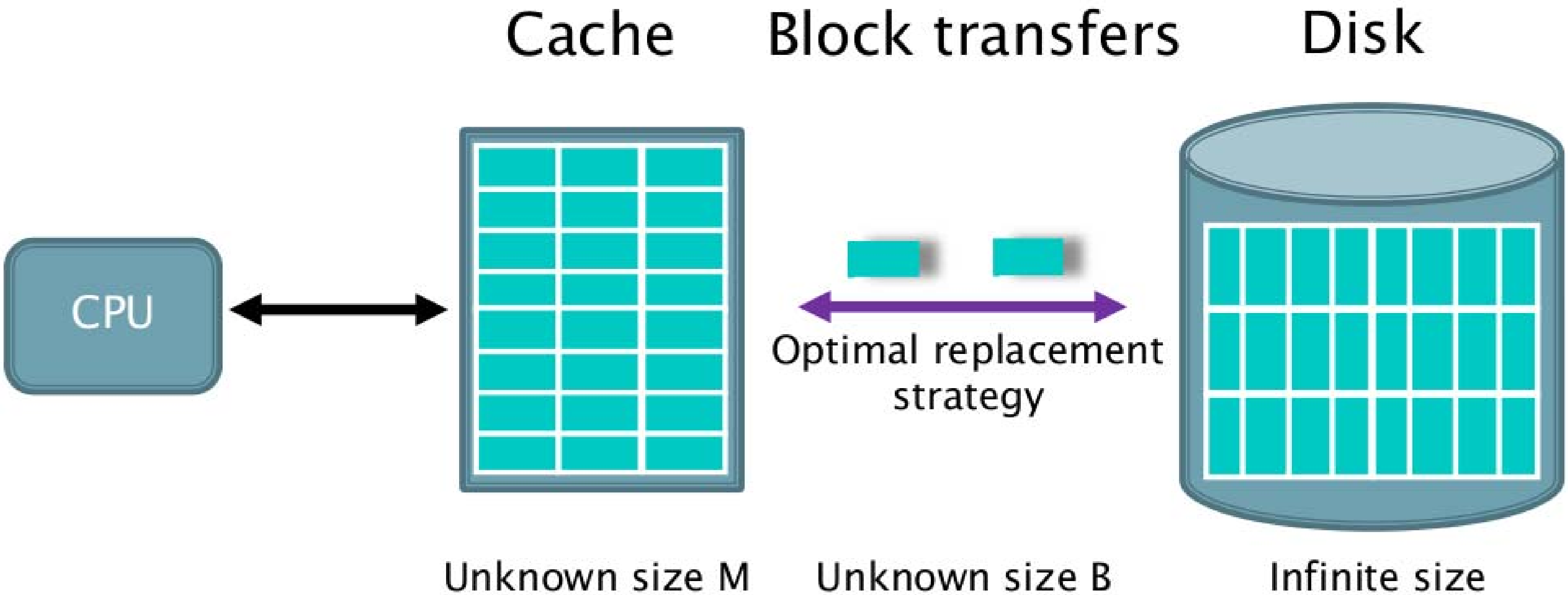
– Architecture dependent
  - Difficult to find optimal B and M (ATLAS)
  - GPU memory hierarchy is complex
  - B and M can vary over time
  - Need to recompute the layout

+ Architecture independent



# Cache-Oblivious Model (CO)

[Frigo & al 1999]



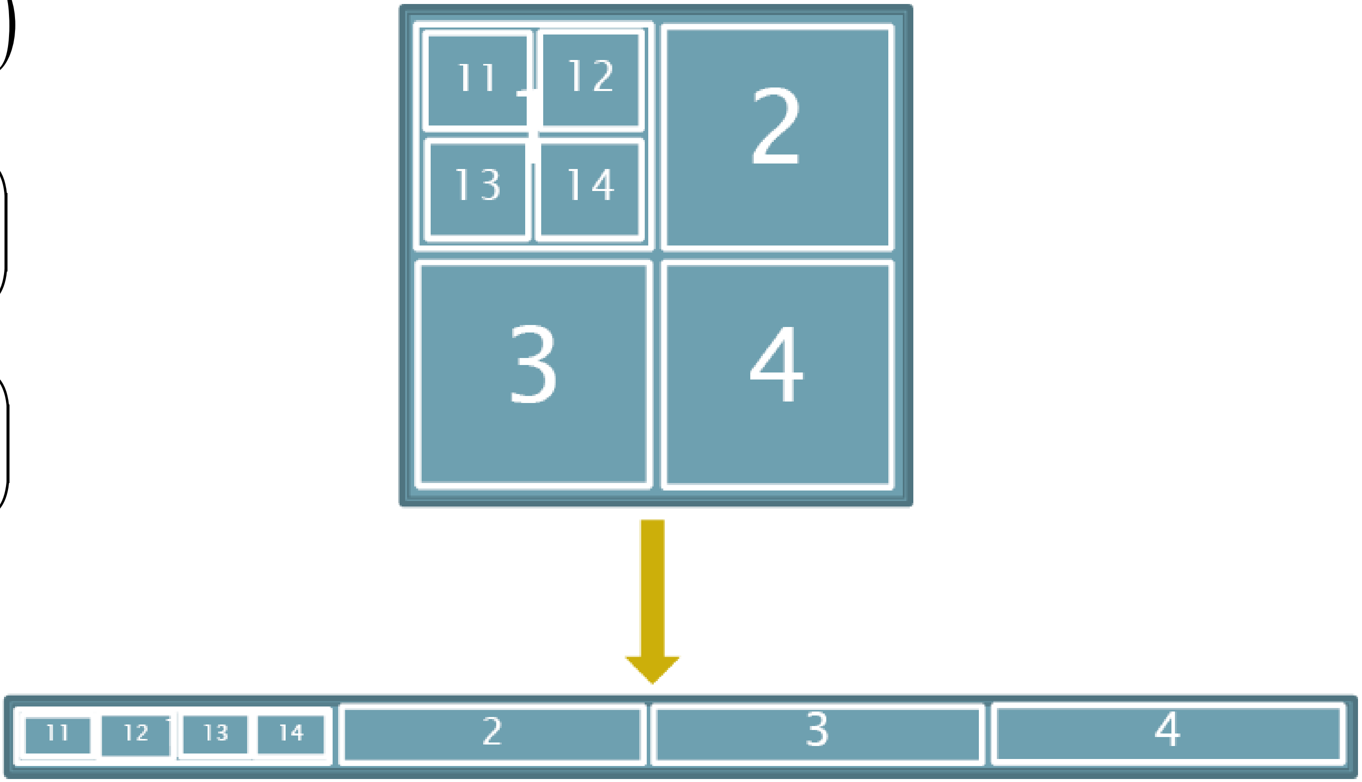
# Multiplying in the CO Model

D&C matrix multiplication using a recursive layout

$$Q_{naive}(N) = O\left(N^3\right)$$

$$Q_{CA}(N) = O\left(\frac{N^3}{B\sqrt{M}}\right)$$

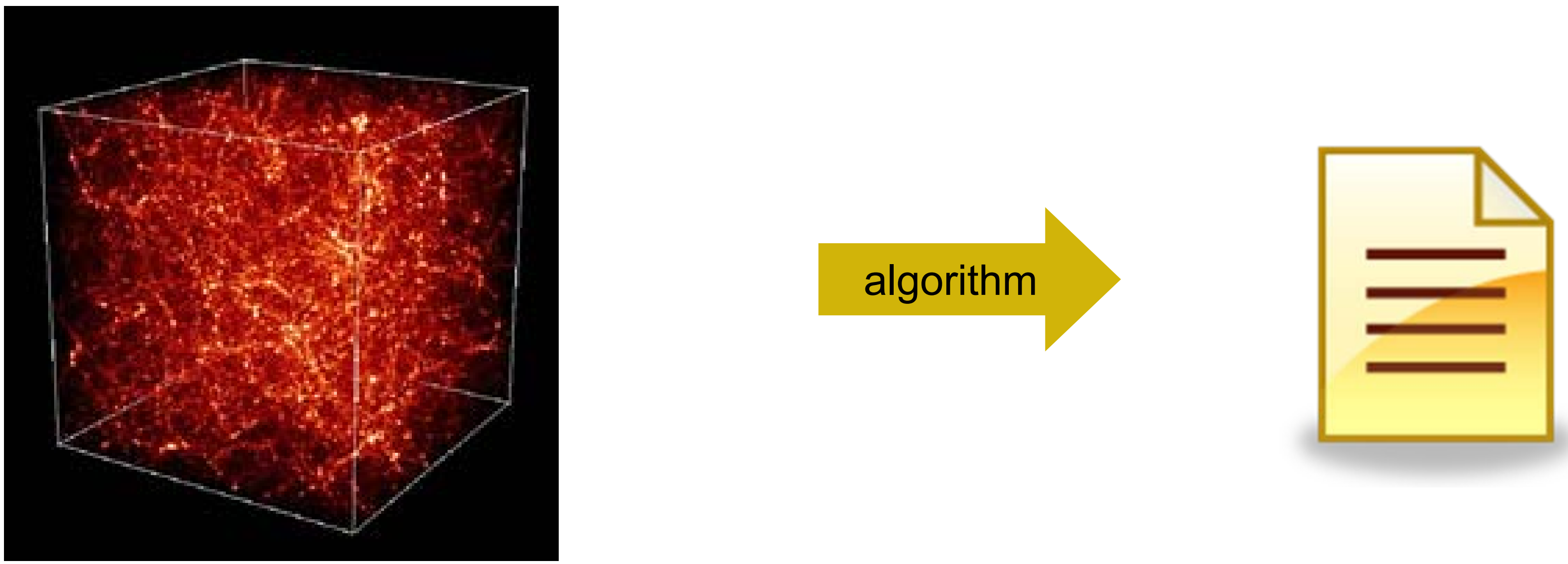
$$Q_{CO}(N) = O\left(\frac{N^3}{B\sqrt{M}}\right)$$



# Outline

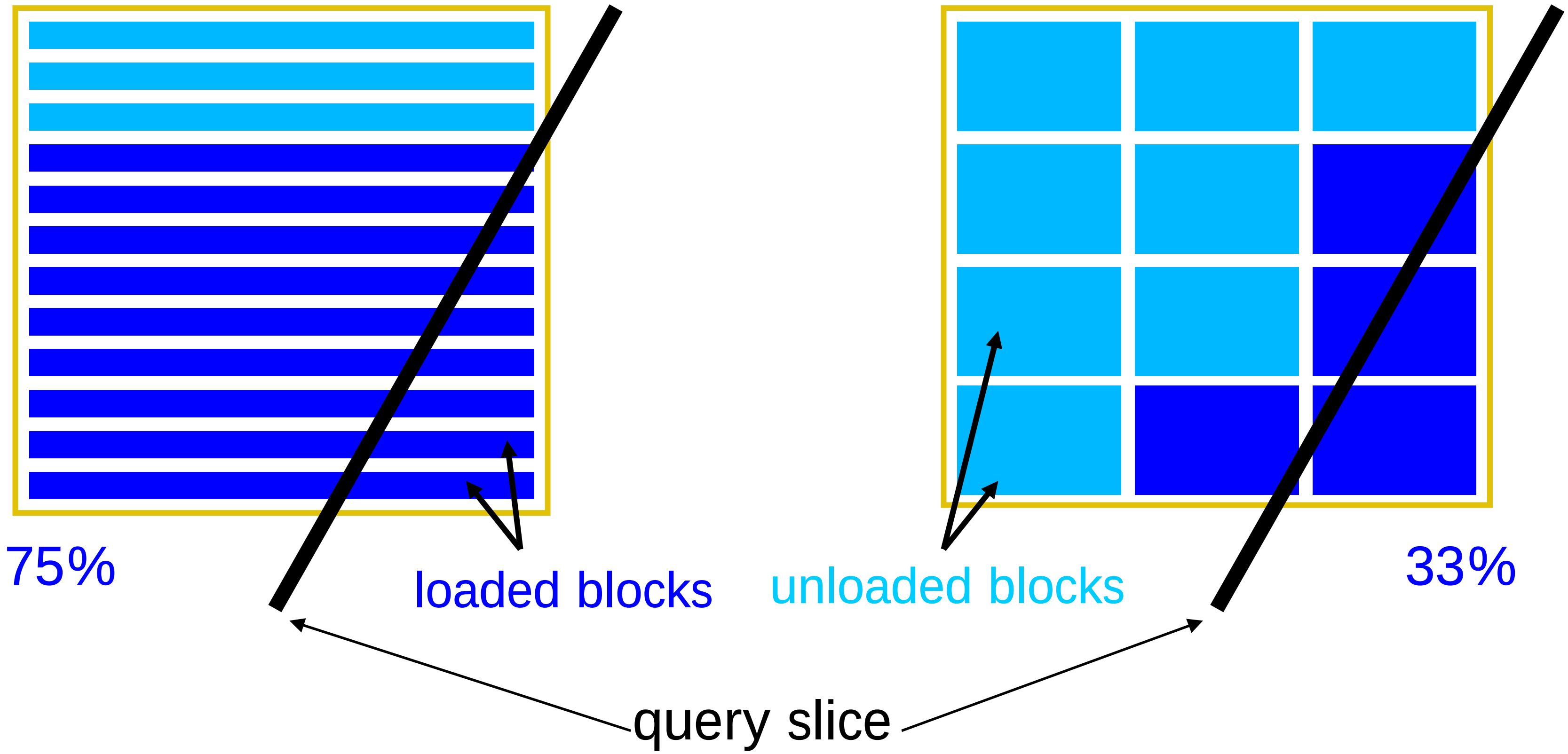
- Cache-aware and cache-oblivious models
  example: matrix multiplication
- The mesh layout problem and previous work
- Our algorithm
- Experiments

# How to lay out a mesh efficiently in memory?

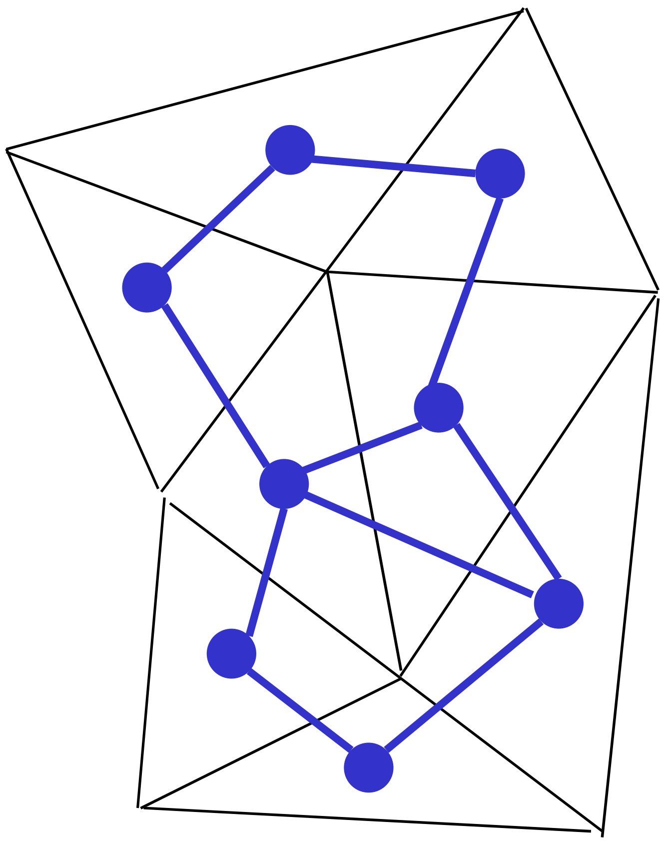


# Idea

Triangles (or vertices) that are most likely to be accessed sequentially should be stored nearby

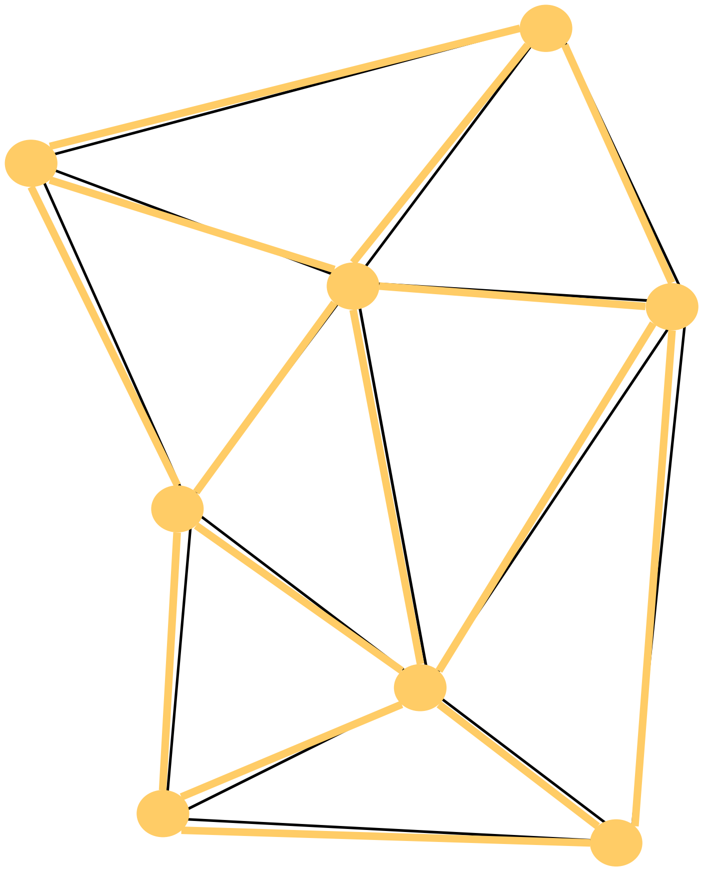


# Graph of Sequential Accesses



$G_1$: sequential access between triangles

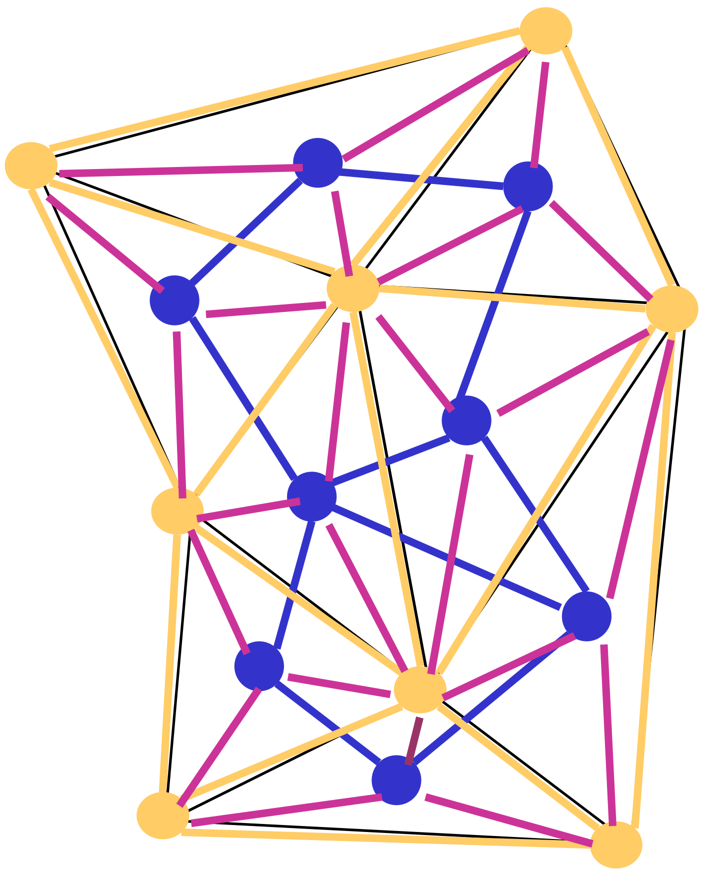
# Graph of Sequential Accesses



$G_1$: sequential access between triangles

$G_2$: sequential access between vertices

# Graph of Sequential Accesses



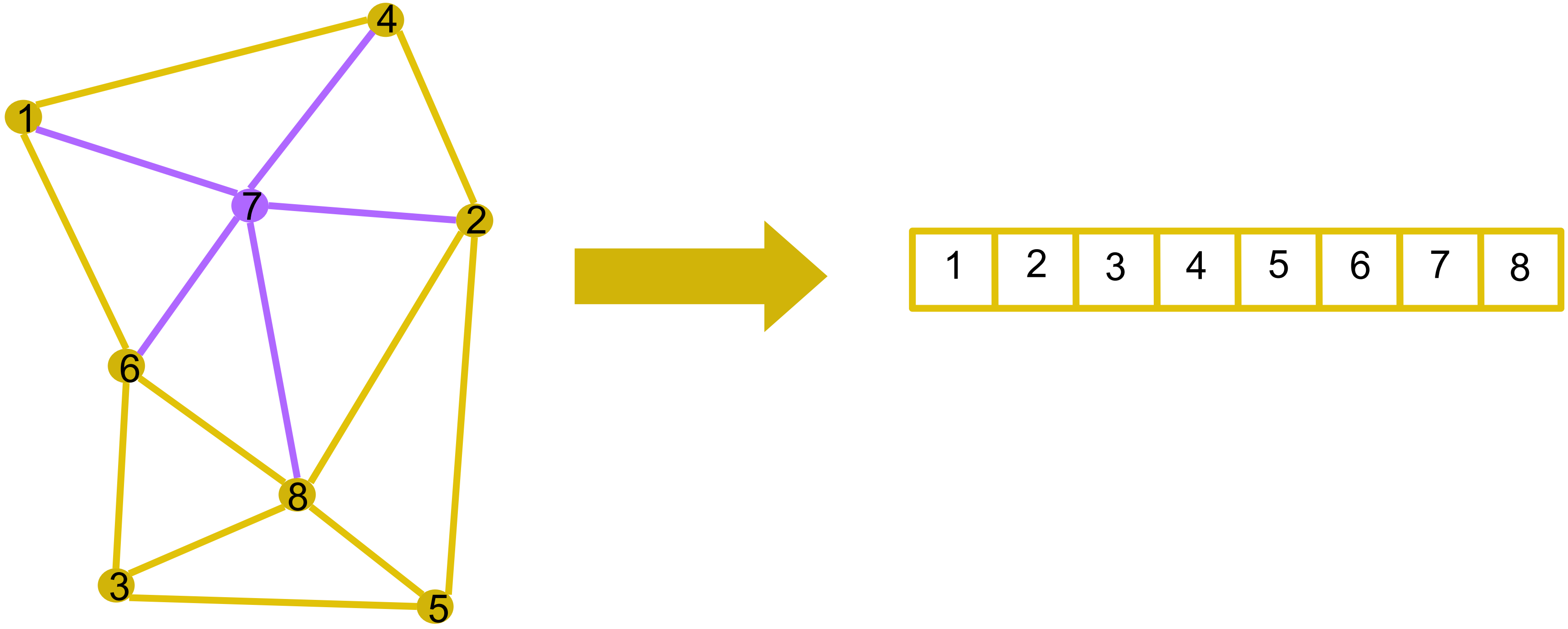
$G_1$: sequential access between triangles

$G_2$: sequential access between vertices

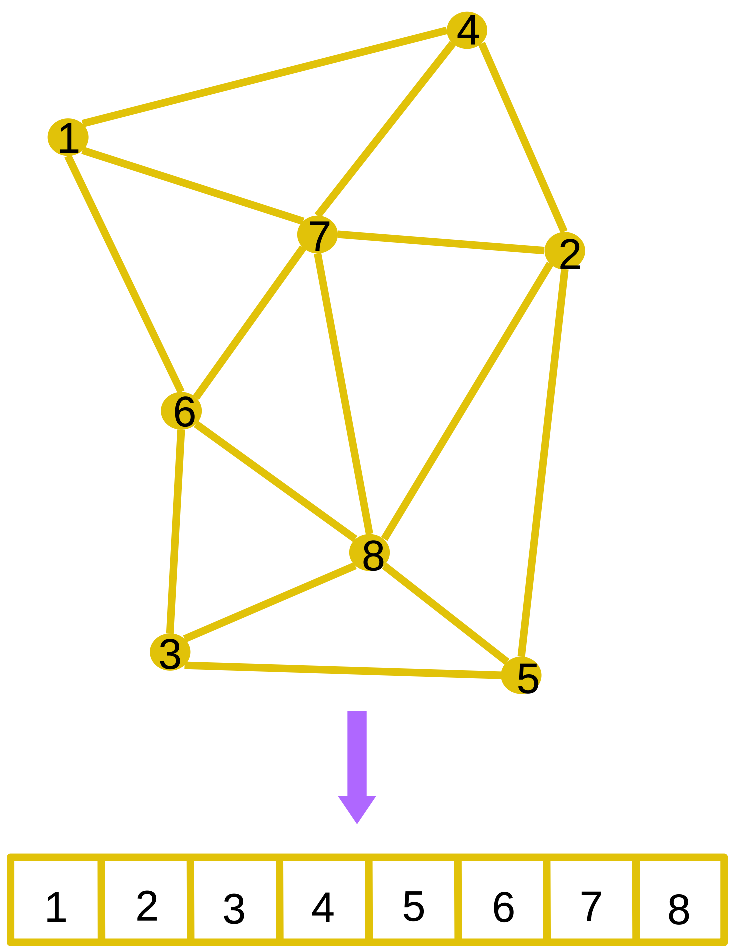
$G_3$: both

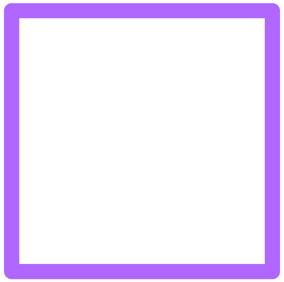
# Mesh Layout Problem

Minimize # of cache misses if each node touches all its neighbors

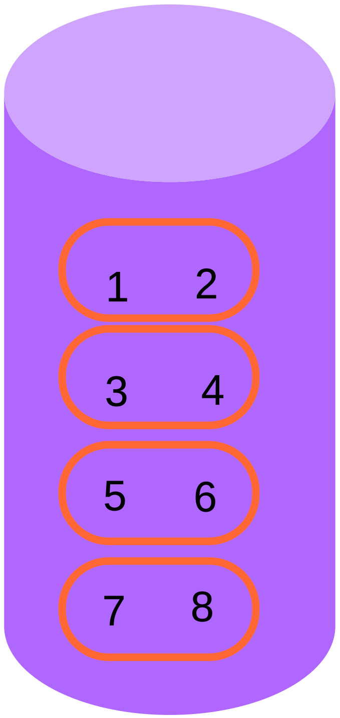


# Example



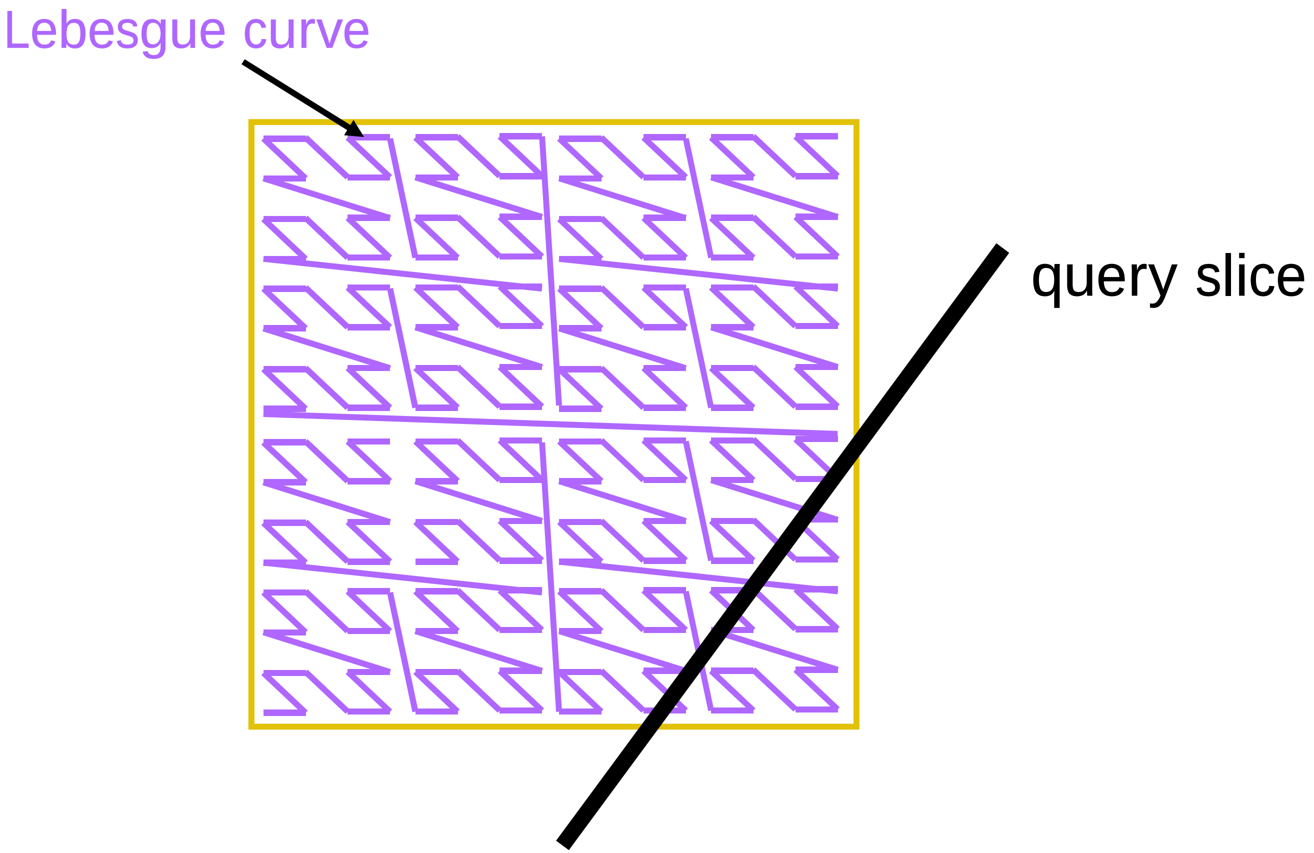
cache (B=2,M=4)



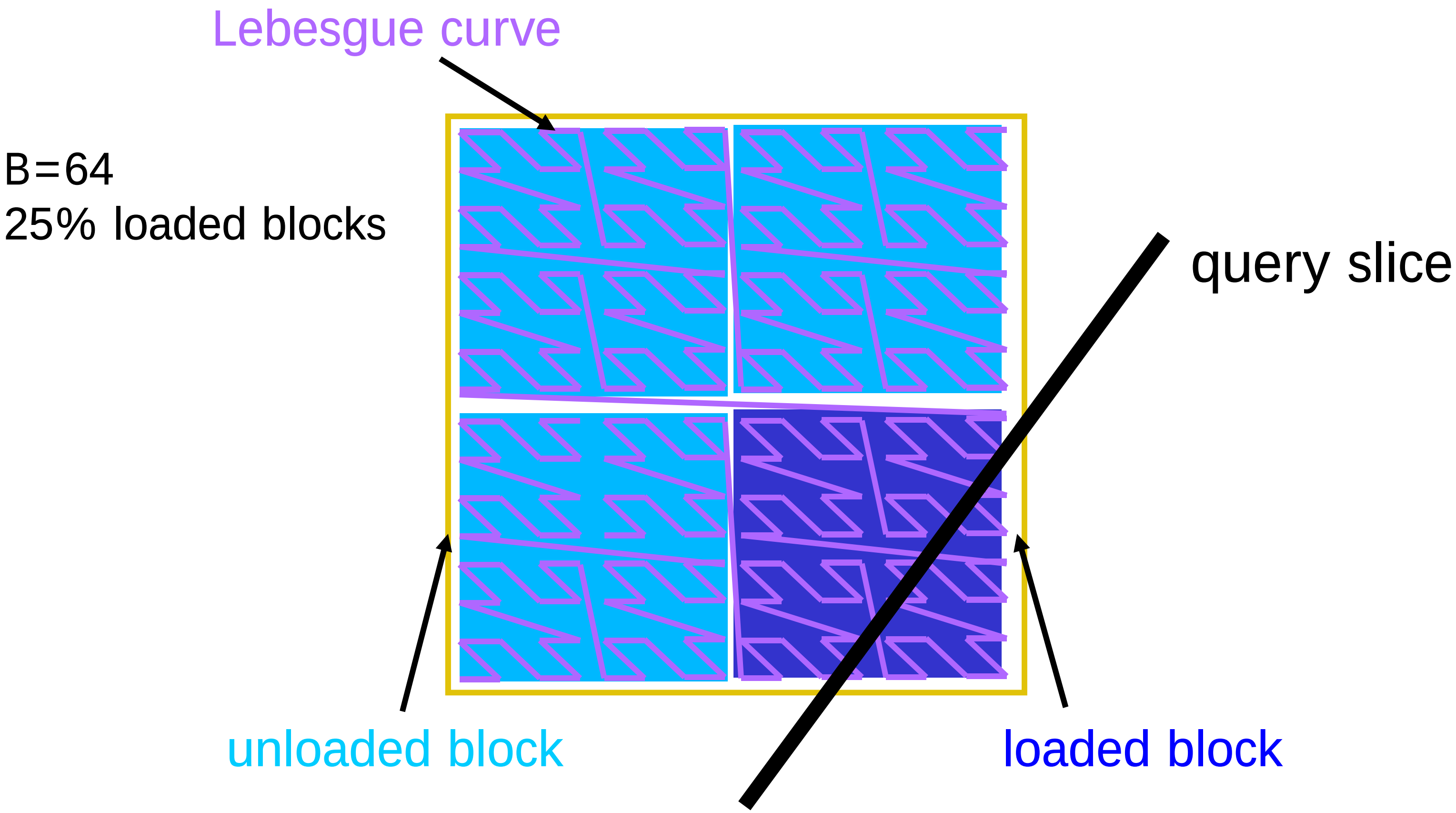

disk

25 cache misses

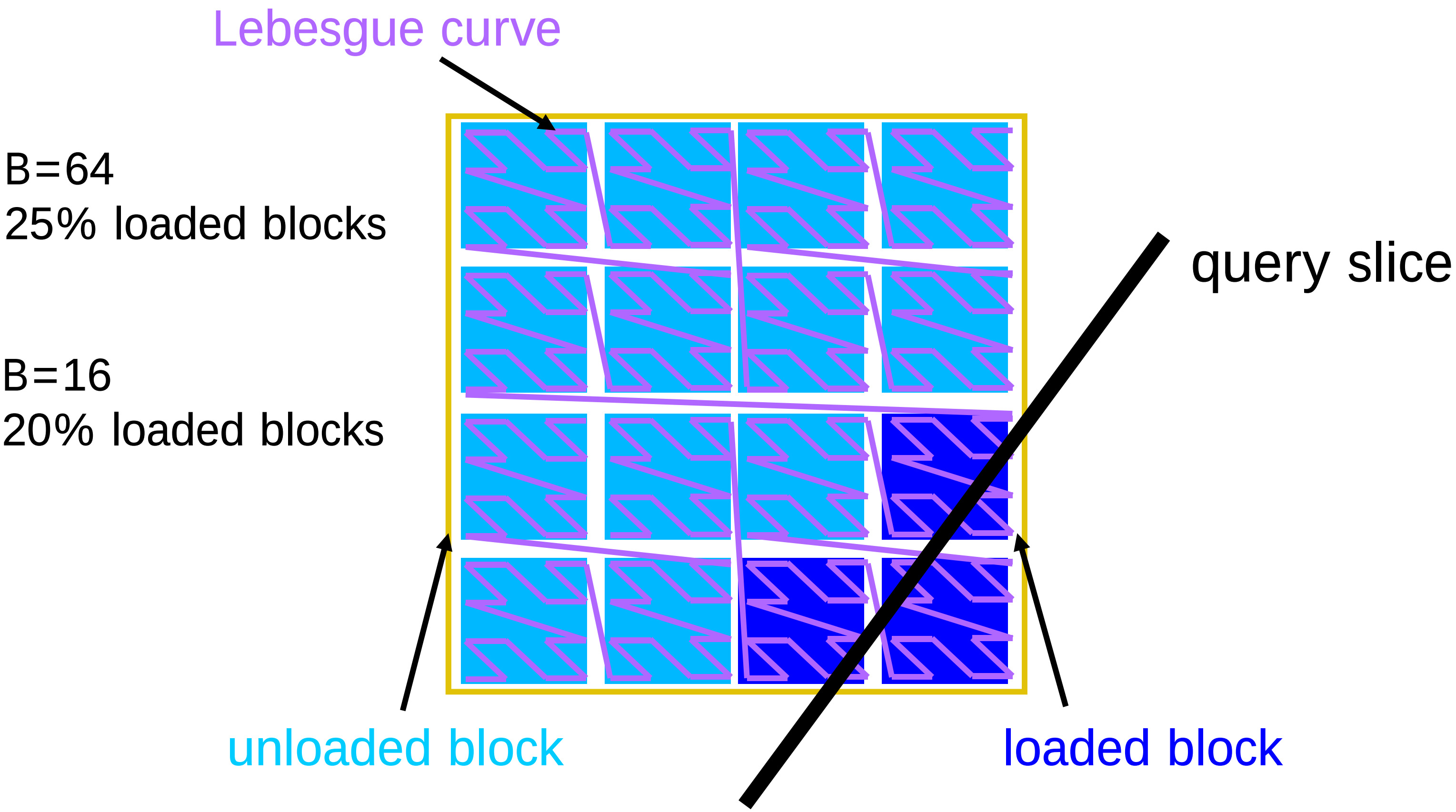
# Regular Meshes: Space-Filling Curves



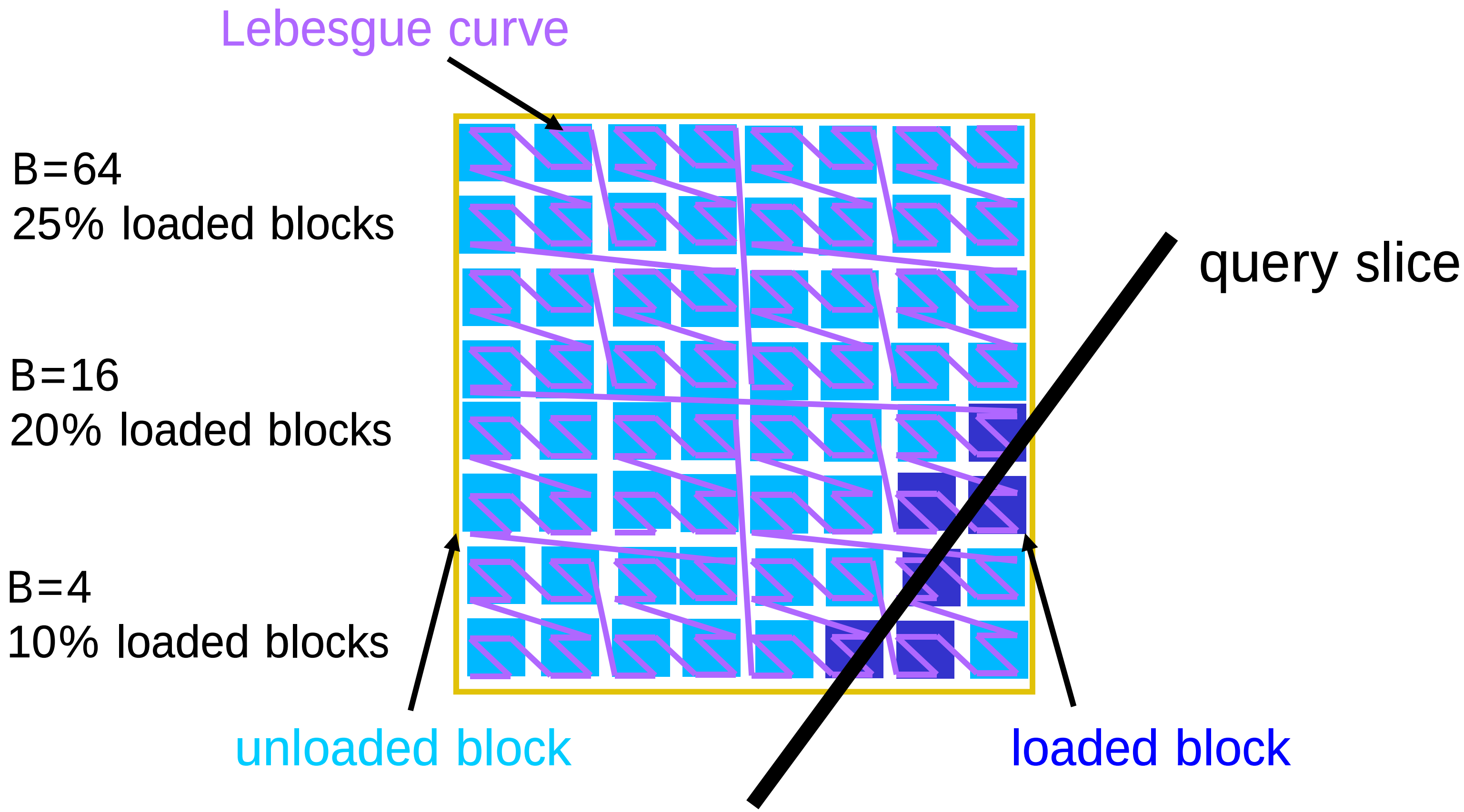
# Regular Meshes: Space-Filling Curves



# Regular Meshes: Space-Filling Curves



# Regular Meshes: Space-Filling Curves



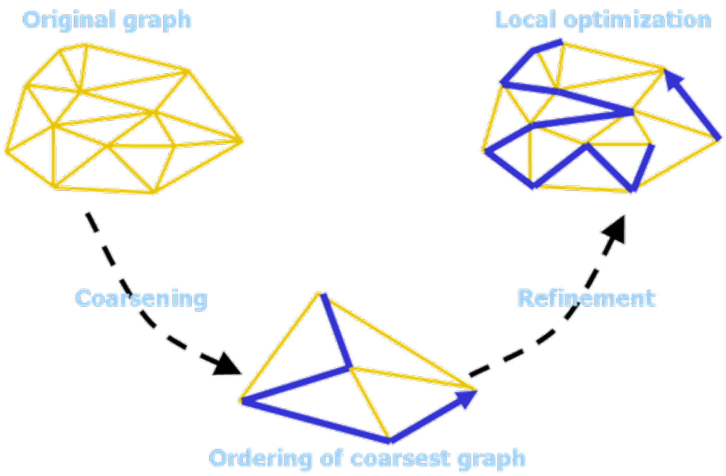
# Unstructured Meshes

[Pascucci & al SIGGRAPH 2005, OpenCCL]

- Heuristic algorithm based on multi-level optimization
  - slow
  - high memory usage

- Good experimental results (2-5x improvement)

- But no guarantee on
  - time to compute the layout
  - layout quality



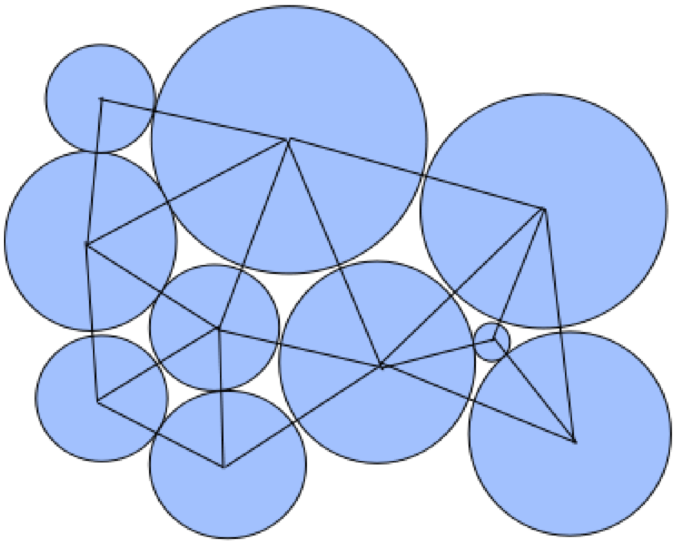
# Outline

- Cache-aware and cache-oblivious models
  example: matrix multiplication
- The mesh layout problem and previous work
- Our algorithm
- Experiments

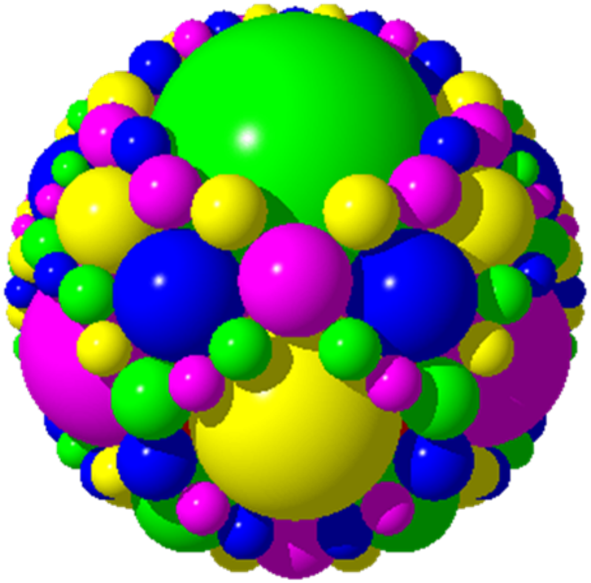
# Overlap graphs

[Miller & al 98]

- Generalize planar graphs
- Contain well-shaped meshes



circles



d-dim spheres

# Separator for overlap graphs

[Miller & al 98]

- Separate the mesh into two roughly equal-size pieces cutting few edges

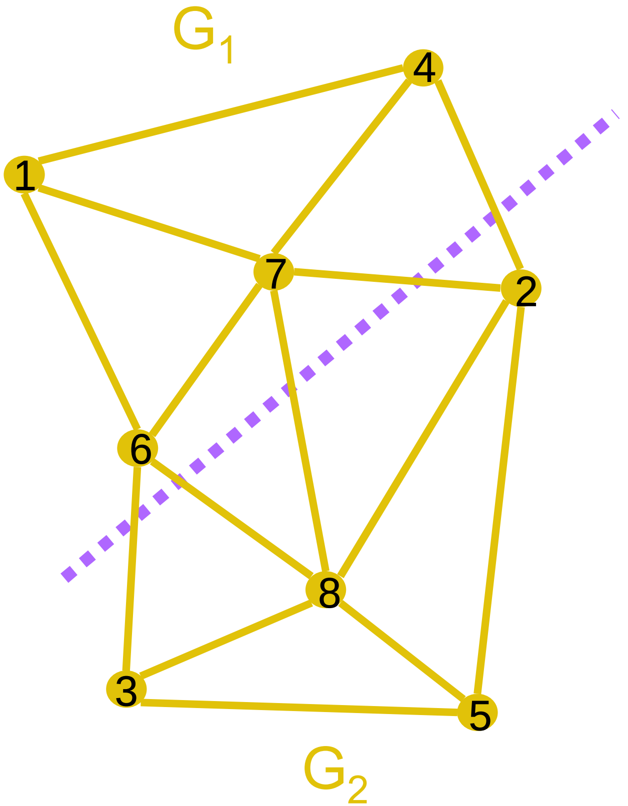
- Planar graphs [Lipton-Tarjan]

$$|G_1|, |G_2| \leq \frac{2}{3}|G| \qquad E(G_1, G_2) \leq \sqrt{8|G|}$$

- Overlap graphs (randomized linear time)

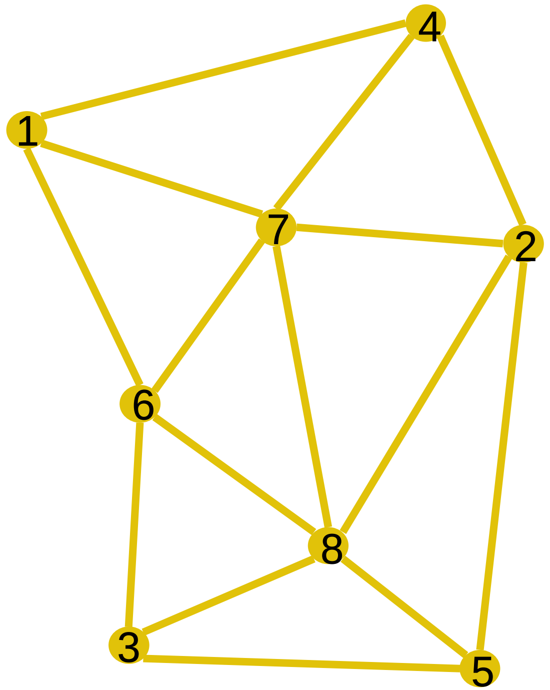
$$|G_1|, |G_2| \leq \frac{d+1}{d+2}|G|$$

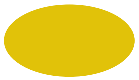
$$E(G_1, G_2) = O\left(|G|^{1-1/d}\right)$$



# Our layout

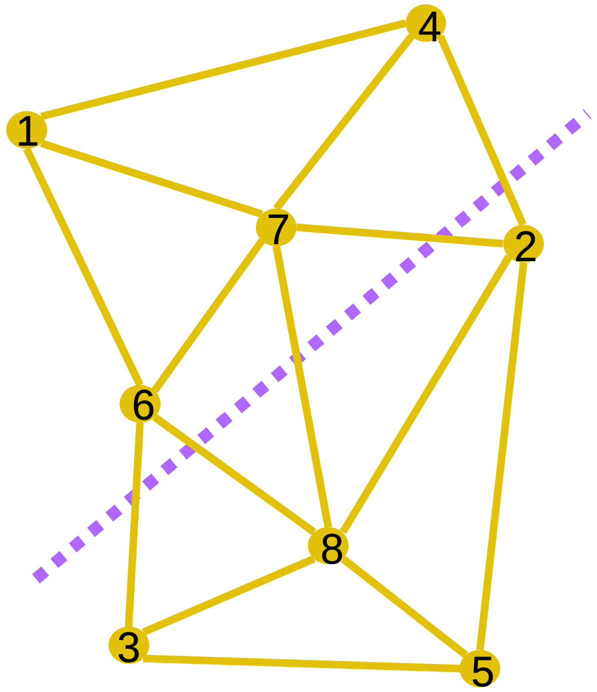
- Recursively cut the mesh

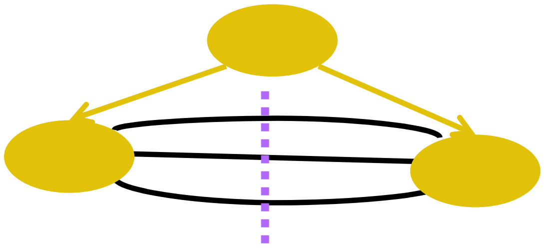


# Our layout

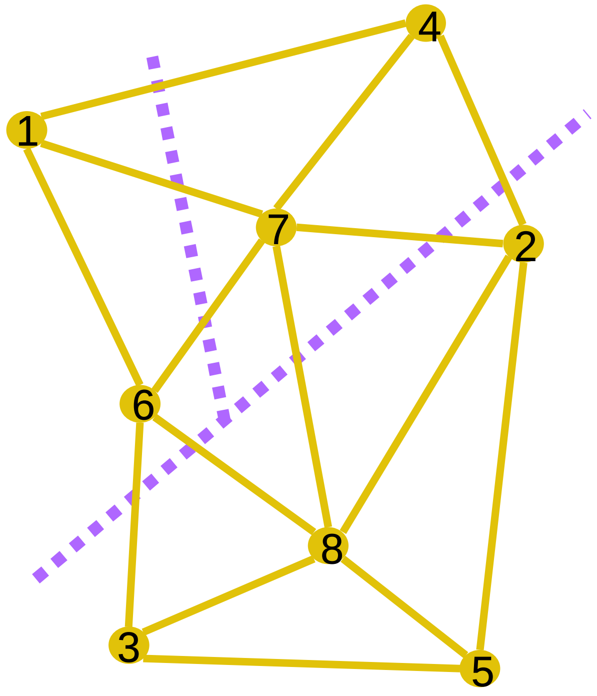
- Recursively cut the mesh

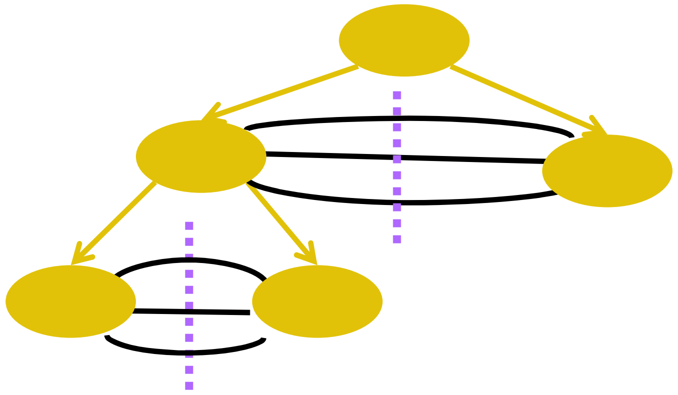


# Our layout

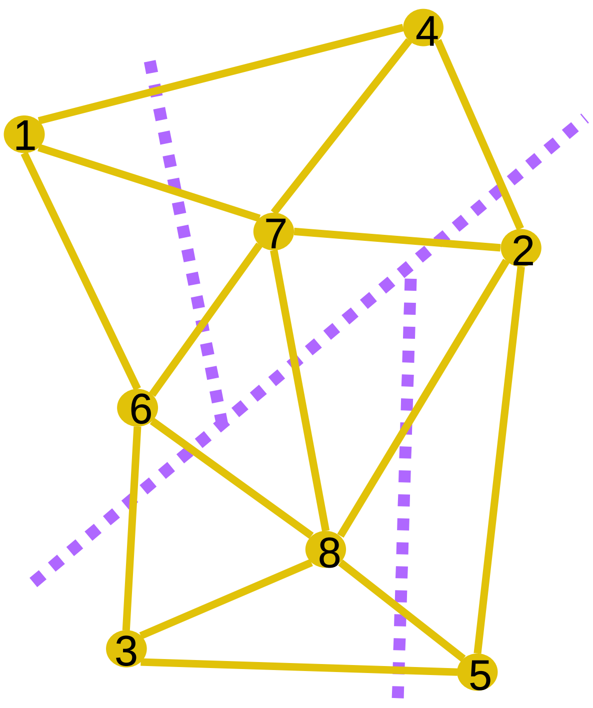
- Recursively cut the mesh

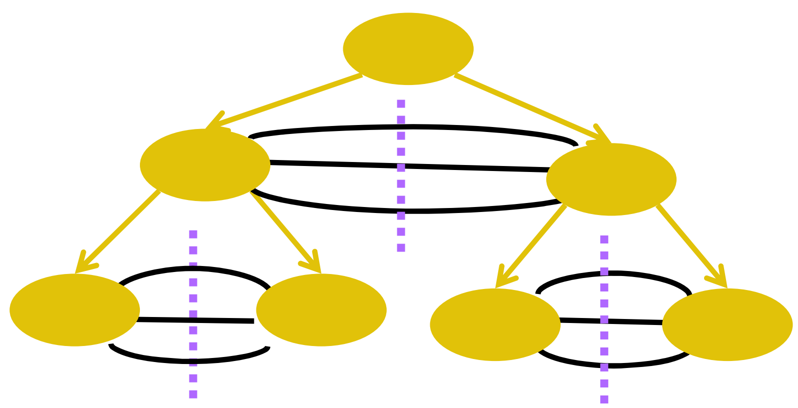


# Our layout

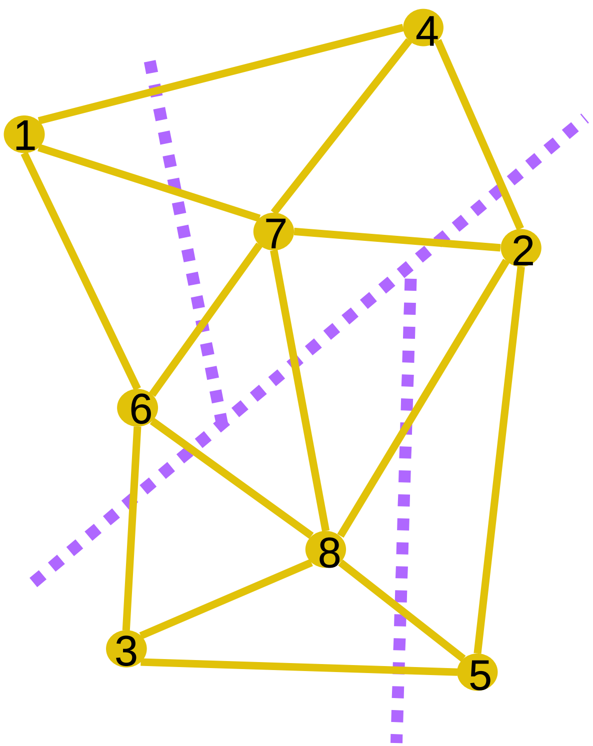
- Recursively cut the mesh

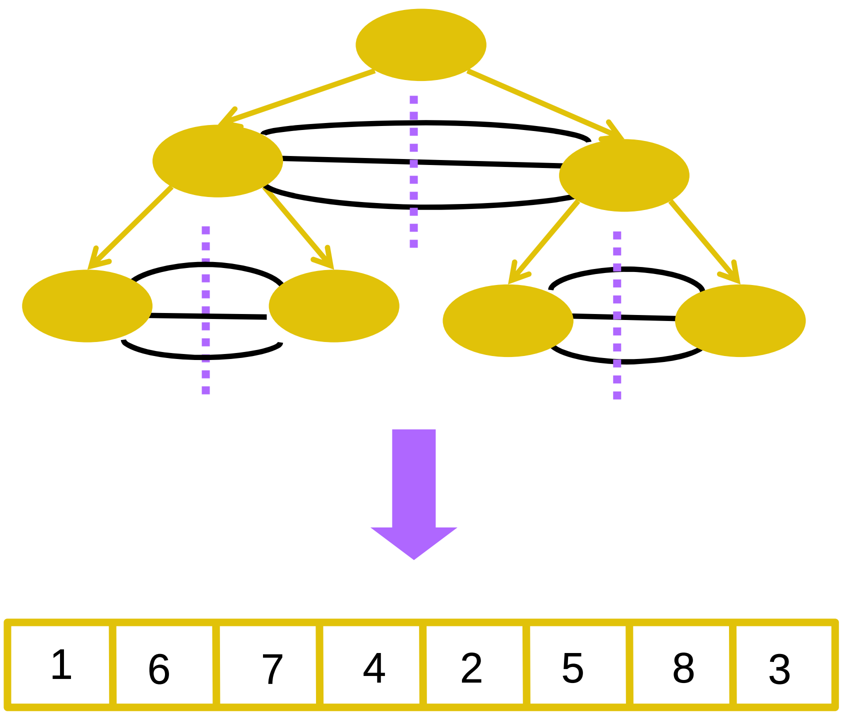


# Our layout

- Recursively cut the mesh $W(N) = O\left(N \log N\right)$
- The order of the leaves gives the layout



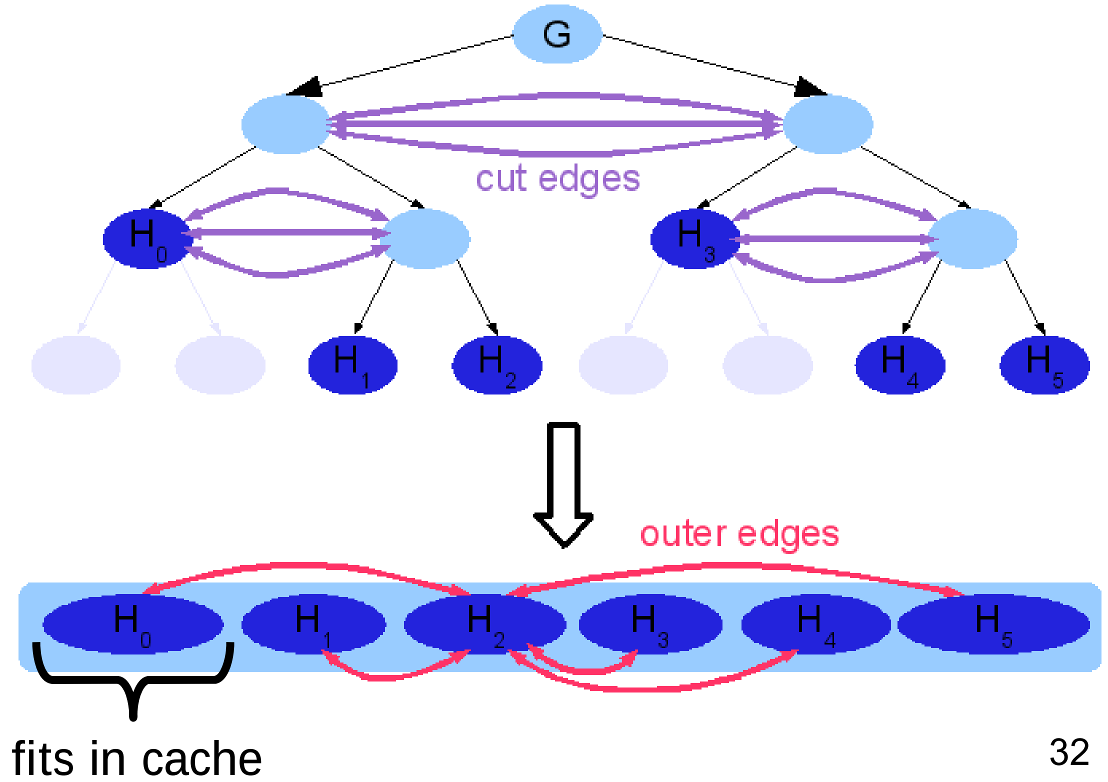



# Guarantee on the Quality of the Layout

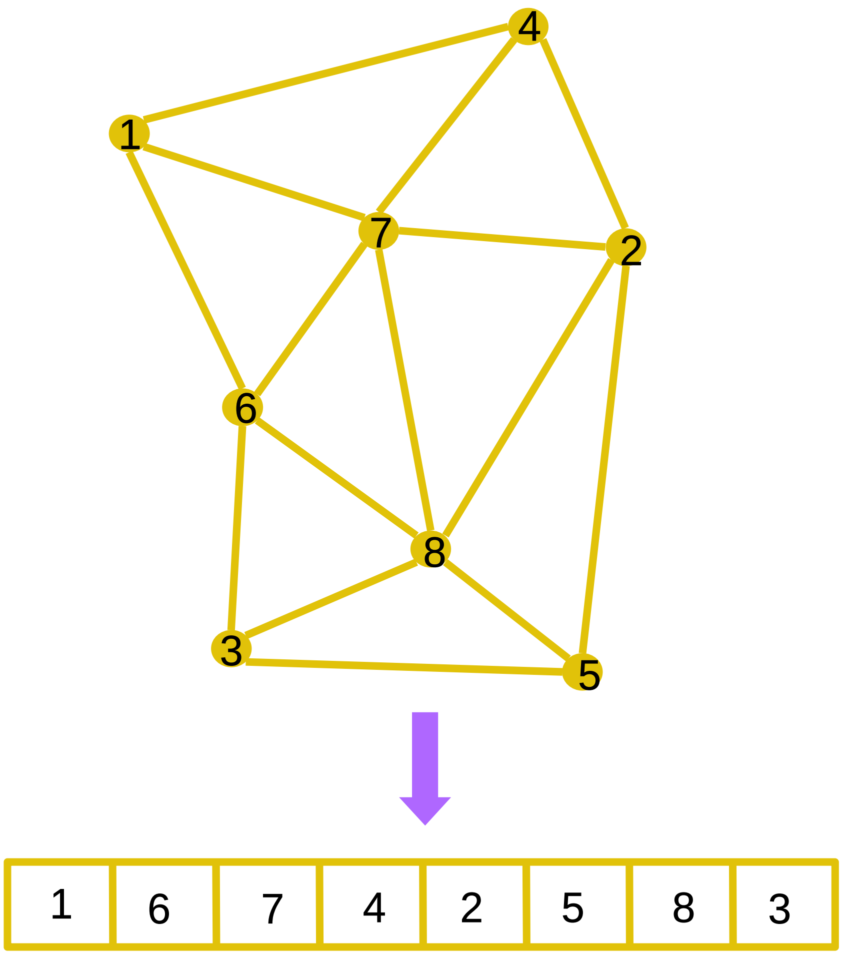
Theorem:

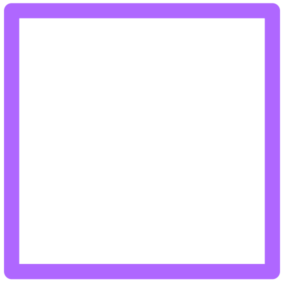
Our layout guarantees that a traversal of an O(N)-size d-dim mesh causes less than $O(N/B+N/M^{1/d})$ cache misses

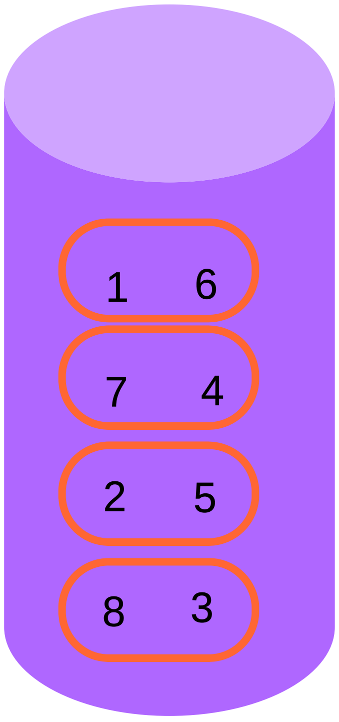
- Each subgraph fits in cache
- Edges inside a subgraph do not cause a cache miss
- Cache misses are bounded by the number of edges between two subgraphs (outer edges)
- One can show that there are few outer edges



# Back to the Example



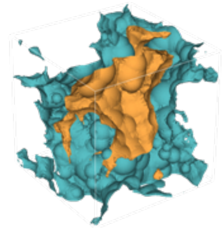
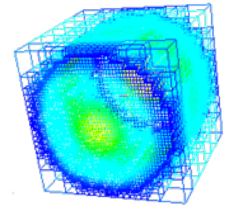
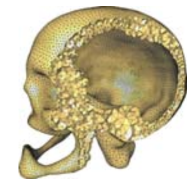




12 cache misses (25 previously)

# Outline

- Cache-aware and cache-oblivious models
  example: matrix multiplication
- The mesh layout problem and previous work
- Our algorithm
- Experiments

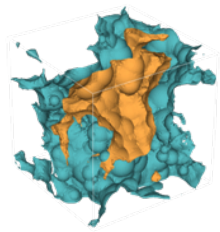
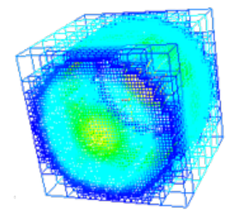
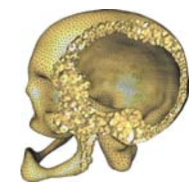
# Experiments

| | Plasma | Reactor | Skull | Neptune |
|---|---|---|---|---|
| Type | Structured | AMR | Unstructured | Unstructured |
| #Vertices | 274k | 84k | 37k | 2M |
| #Cells | 1.3M tetra | 78k hexa | 156k tetra | 4M tri |
| Size | 47MB | 8 MB | 6 MB | 169 MB |

Opteron 875 2,2Ghz
L1 = 64K
L2 = 1024K
Cache lines = 64B
32G of RAM

# Layout Computation

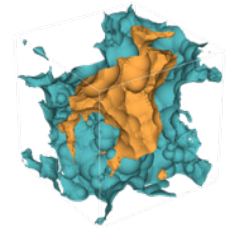
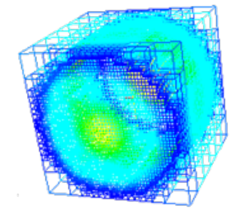
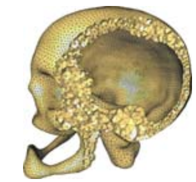





| | Our Layout | | OpenCCL* | |
|---|---|---|---|---|
| | Time (s) | Memory (MB) | Time (s) | Memory (MB) |
| Plasma | 107 | 124 | 282 | 6,843 |
| Reactor | 8.8 | 15 | 27.6 | 458 |
| Skull | 10.6 | 16 | 26.9 | 814 |
| Neptune | 410 | 269 | 843 | 20,500 |

***At least twice as fast using 30 times less memory***

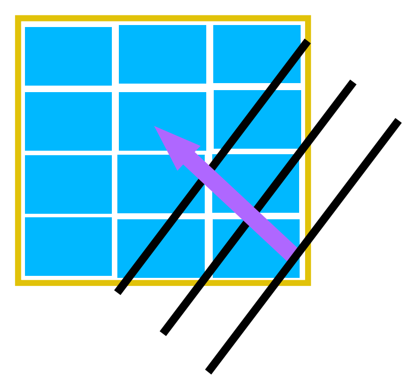
* www.cs.unc.edu/~geom/COL/OpenCCL/

# Layout Quality

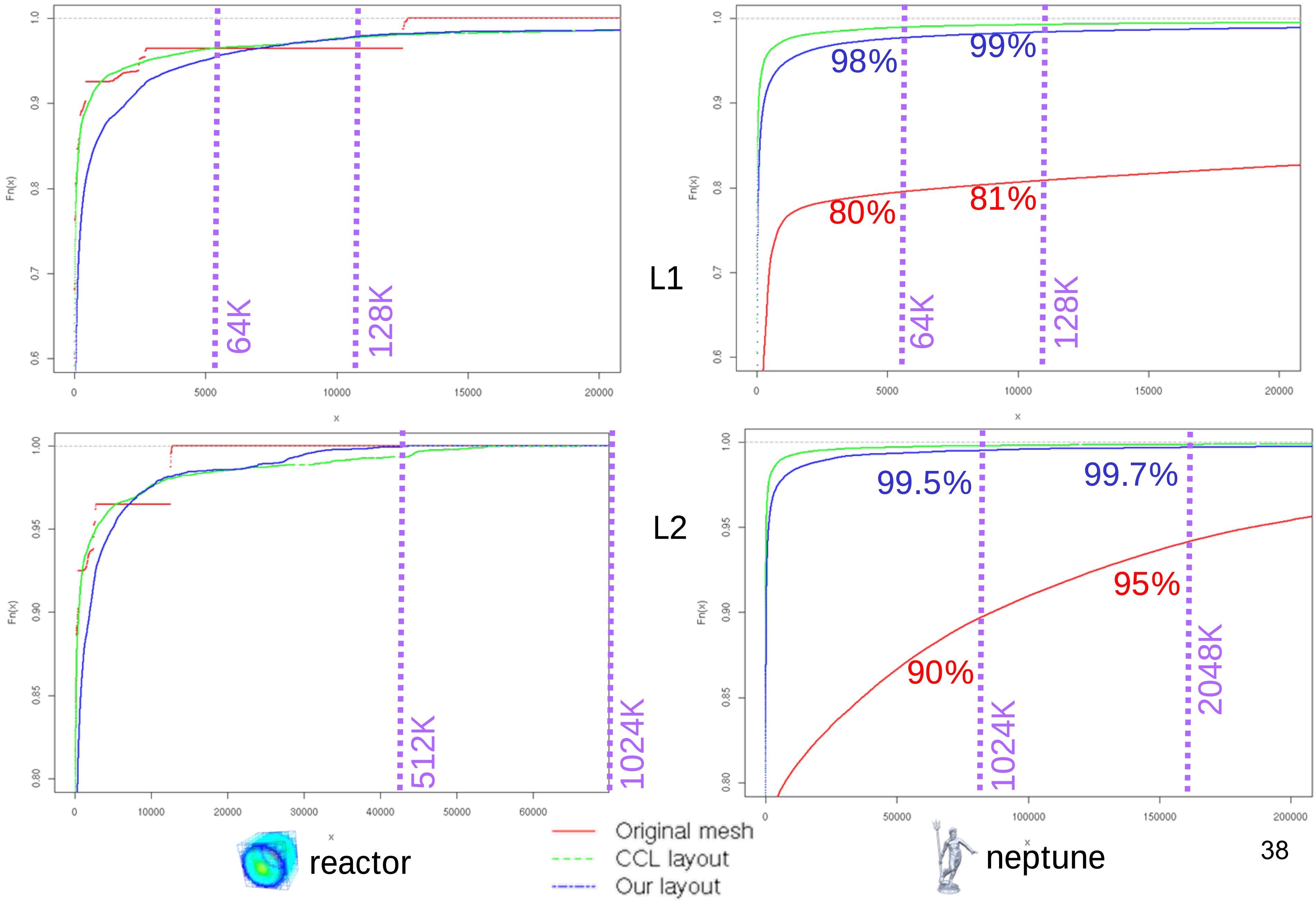





| | Original Layout Time (s) (Dev.) | Our Layout Time (s) (Dev.) | OpenCCL Time (s) (Dev.) | |
|---|---|---|---|---|
| Plasma | 32.4 (0.25) | 33.4 (0.22) | 32.5 (0.22) | |
| Reactor | 3.3 (0.05) | 3.4 (0.09) | 3.3 (0.10) | |
| Skull | 5.1 (0.04) | 4.96 (0.02) | 4.95 (0.02) | 3% |
| Neptune | 121 (2.5) | 110 (1.3) | 110 (1.0) | 9% |

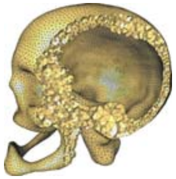
- VTK 5
- a cut plane is moved through the whole mesh
- experiment is repeated 30 times



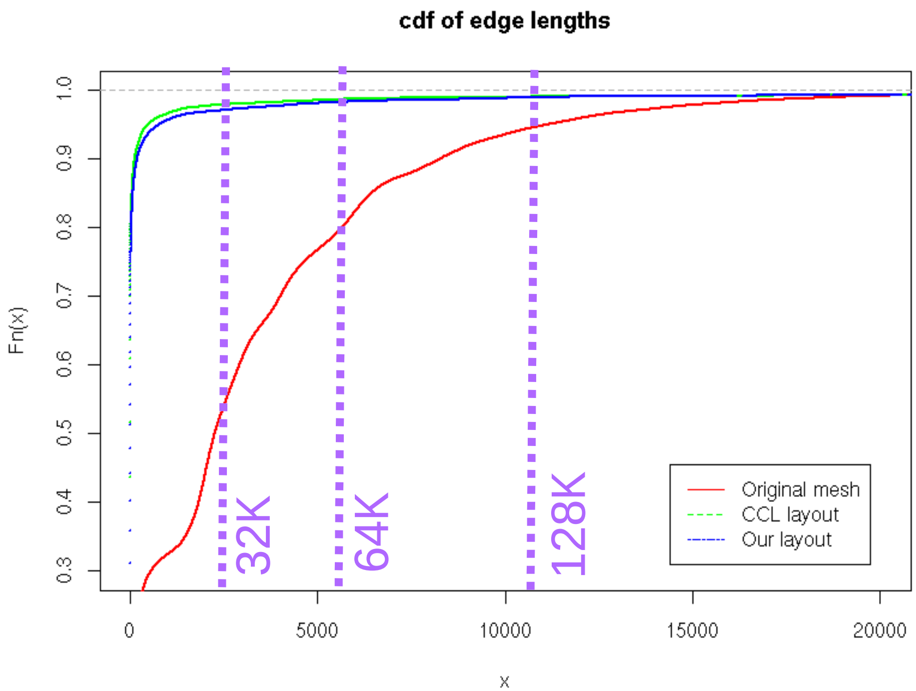
# Cdf of edge lengths

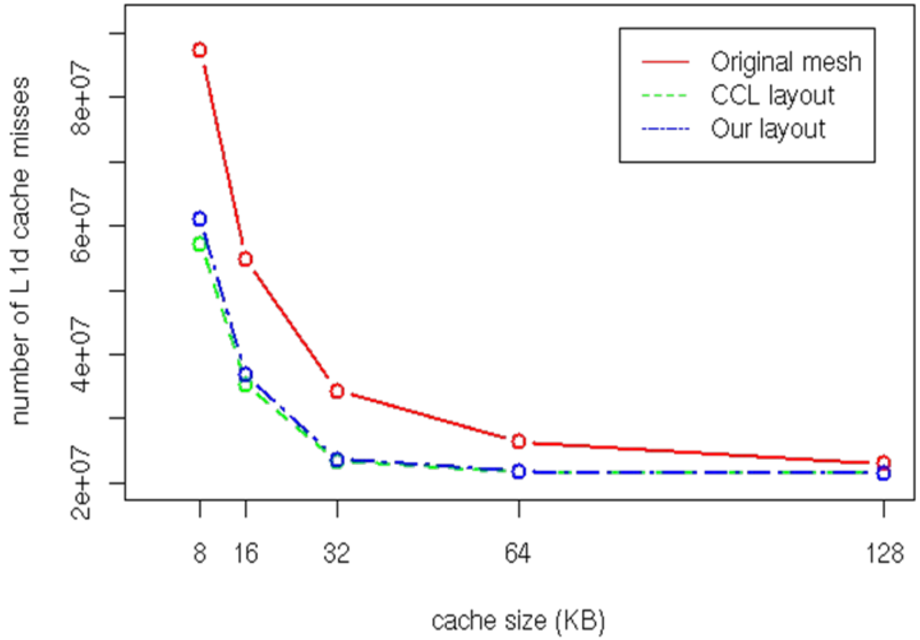


# Correlation Edge Lengths / Cache Misses



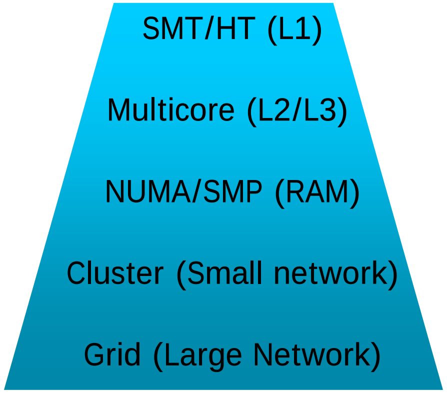
skull




Simulation (valgrind) of the number of cache misses with varying L1 cache sizes

# Conclusion & Future Work




- Our algorithm
  - Fast  O(N log N)
  - Quality & time guarantees
  - Architecture independent
- Better validation
  - Big 3D unstructured meshes
  - Find the factor 2 improvement of Pascucci SIGGRAPH05
- Improve the layout
  - What if only part of the mesh is accessed?
  - Parallelism

Figure 9: **Dynamic Simulation:** Dragons consisting of 800K triangles are dropping on the Lucy model consisting of 28M triangles. We obtain 2 times improvement by using COL on average.

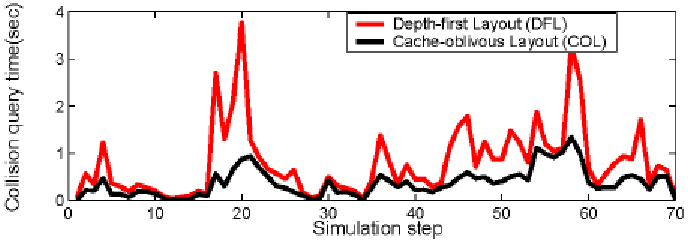



Figure 11: **Performance of Collision Detection:** Average query times for collision detection between the Lucy model and the dragon model with COL and DFL are shown. We obtain 2 times improvement in the query time on average.

# Questions?

*Thank you*