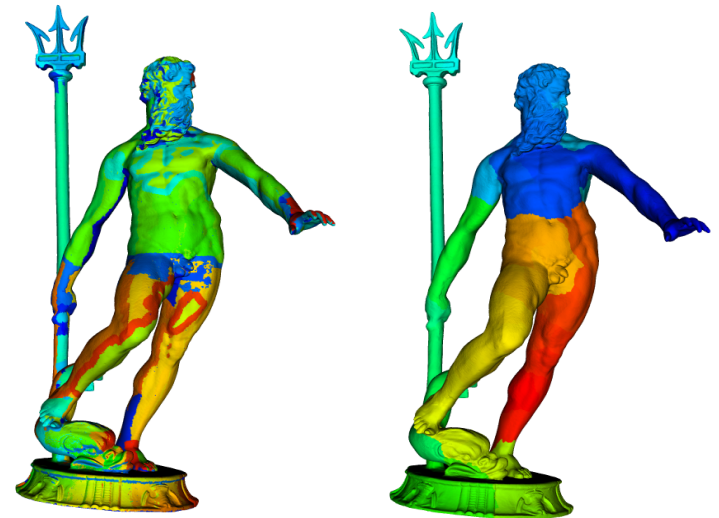
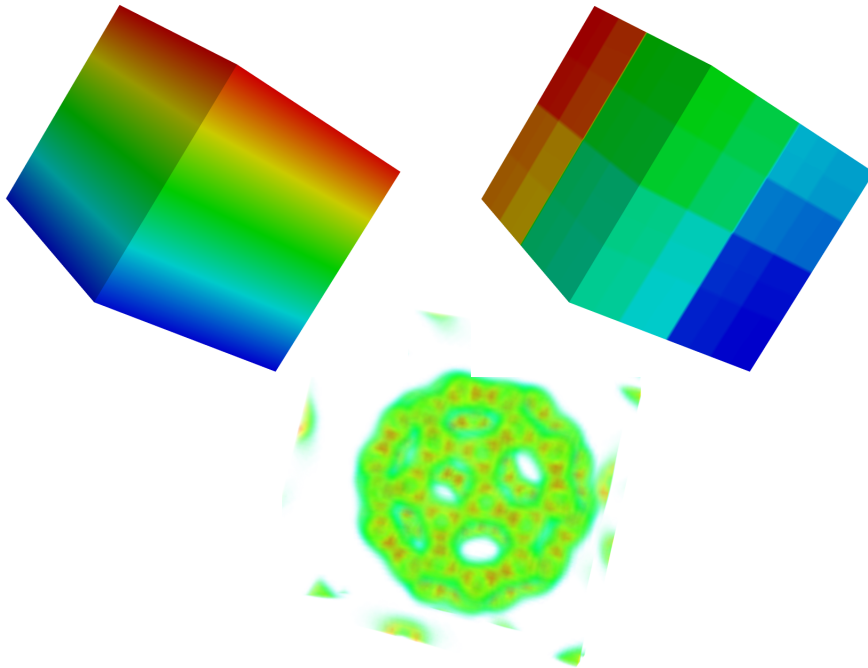


Cache-Oblivious Algorithms

Application to Scientific Visualization



Vincent Danjean, Bruno Raffin,
Marc Tchiboukdjian
MOAIS team-project
Laboratoire d'Informatique de Grenoble

Jean-Philippe Nominé
CEA/DAM/DIF

Visualization & Massive Data Sets



TERA-10

CEA supercomputer

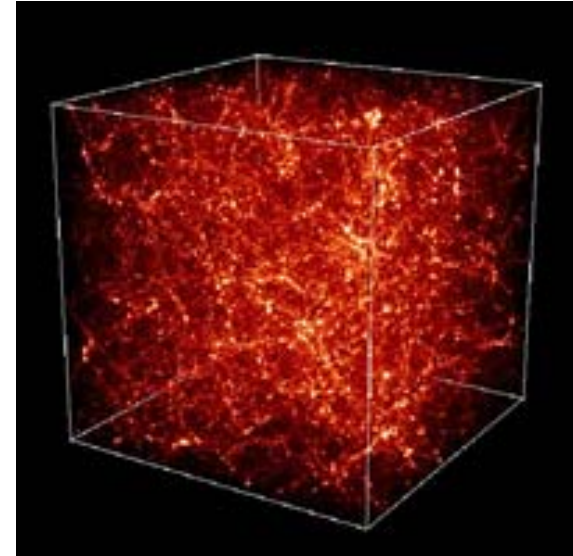
19th Top500

9968 Itanium2

60 Tflops

30 TB of memory

1 PB of disk space




Simulation of half the
observable universe

50 TB mesh

How do we visualize massive data sets ?

Techniques to Handle Massive Data Sets

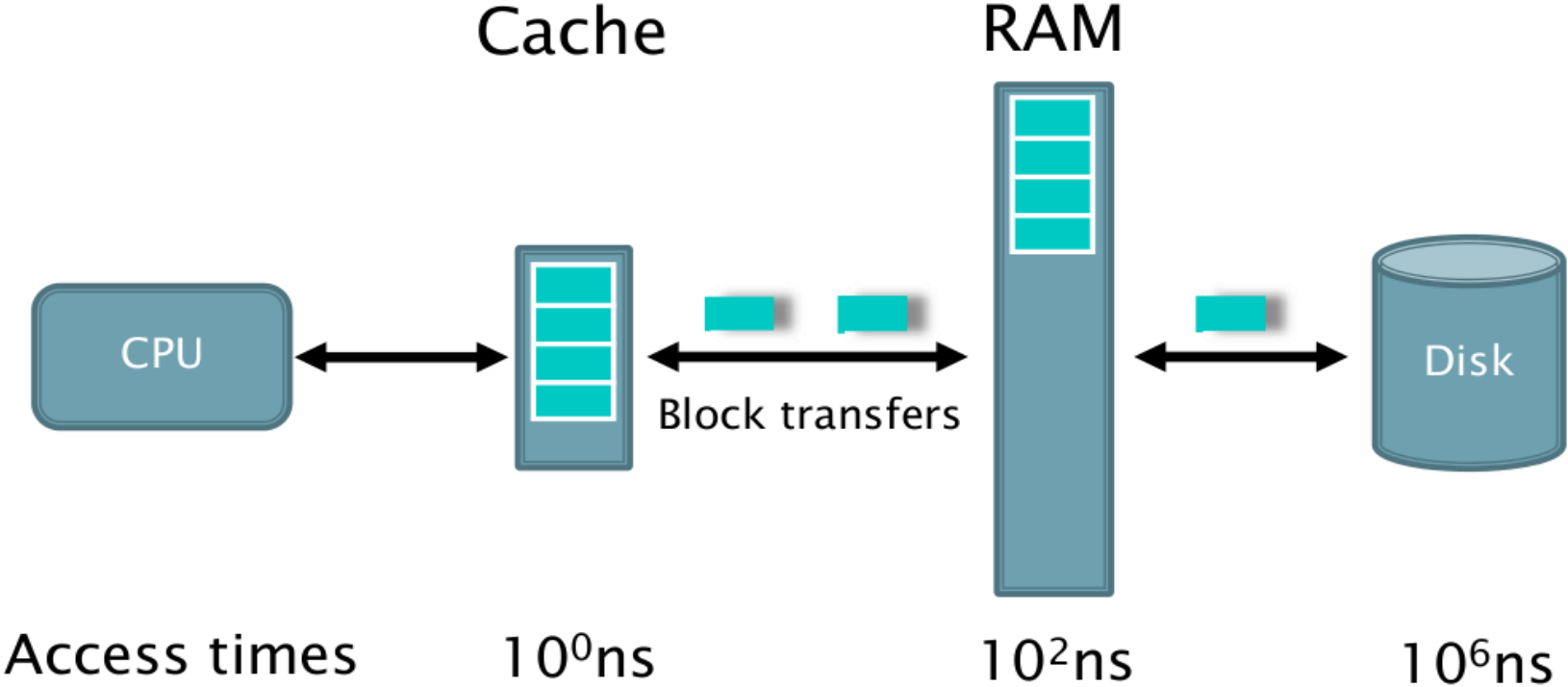
- Data Compression
 - Geometry compression
 - Topology compression
 - Attributes compression
- Cache-conscious techniques  Today's topic
 - Data reordering
 - Computation reordering

Outline

« *An algorithm is said **oblivious** if no program variables dependent on hardware configuration parameters need to be tune to reach optimal performances* » [Prokop&al]

- Cache-Aware Model
- Cache-Oblivious Model
- Cache-Oblivious Mesh Layout
- Preliminary Experiments
- Conclusion & Future Work

Memory Hierarchy



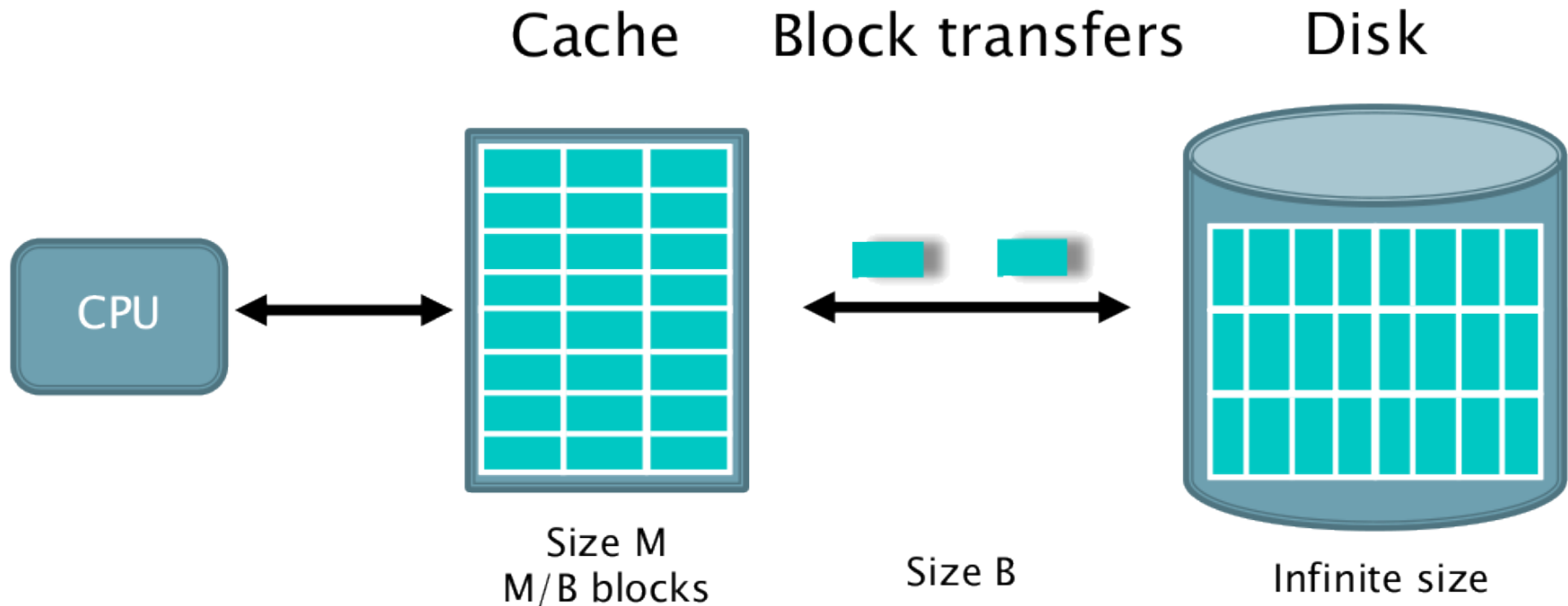
Cache-Aware Model (CA)

[Aggarwal & Vitter 1988]

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

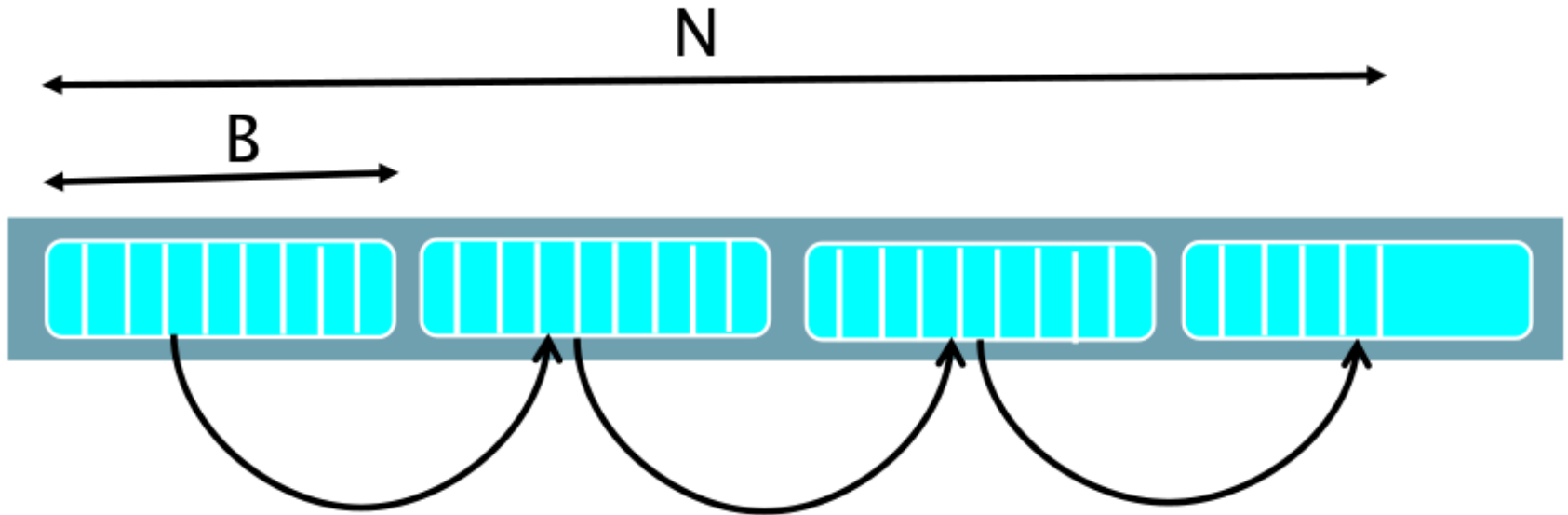
W: #operations

Q: #block transfers



Scanning in the CA Model

Read an N-elements array: the naive algorithm is optimal

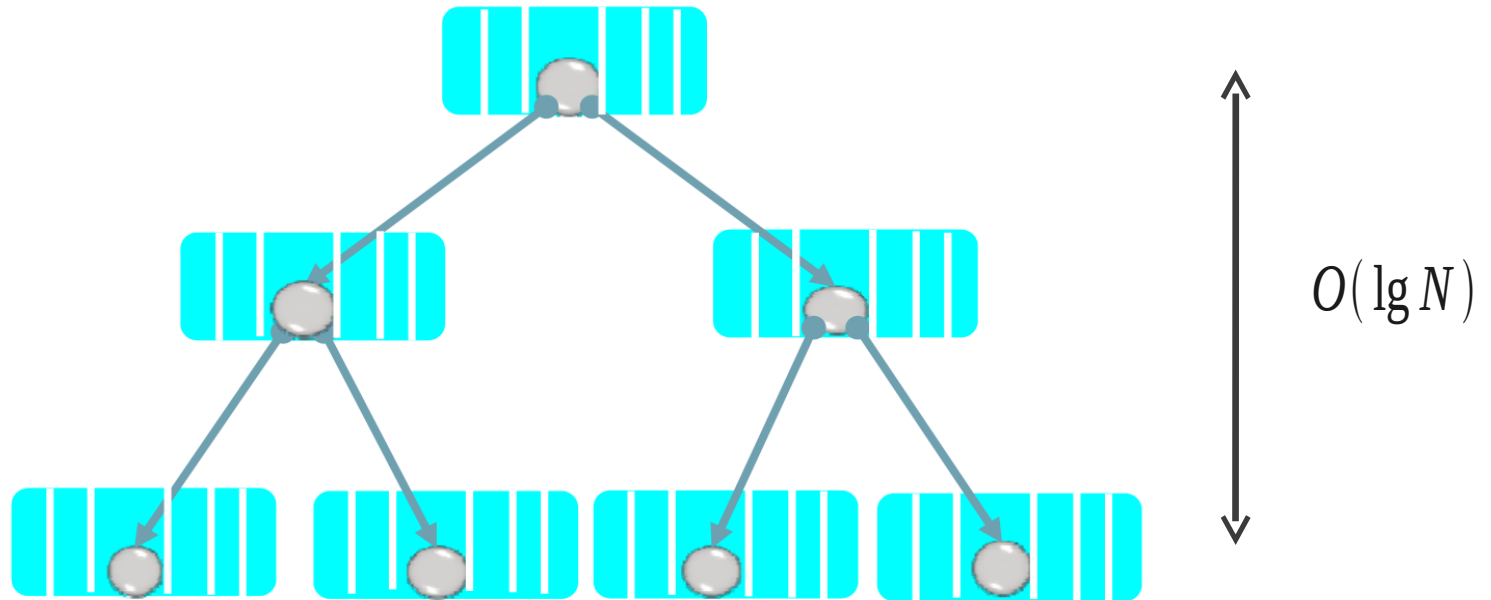


$$W(N) = N$$

$$Q_{CA}(N) = \lceil N/B \rceil$$

Searching in the CA Model

Searching a key in an N -nodes balanced binary tree: naive doesn't work



$$W(N) = 1 \cdot O(\lg N)$$

$$Q_{naive}(N) = 1 \cdot O(\lg N)$$

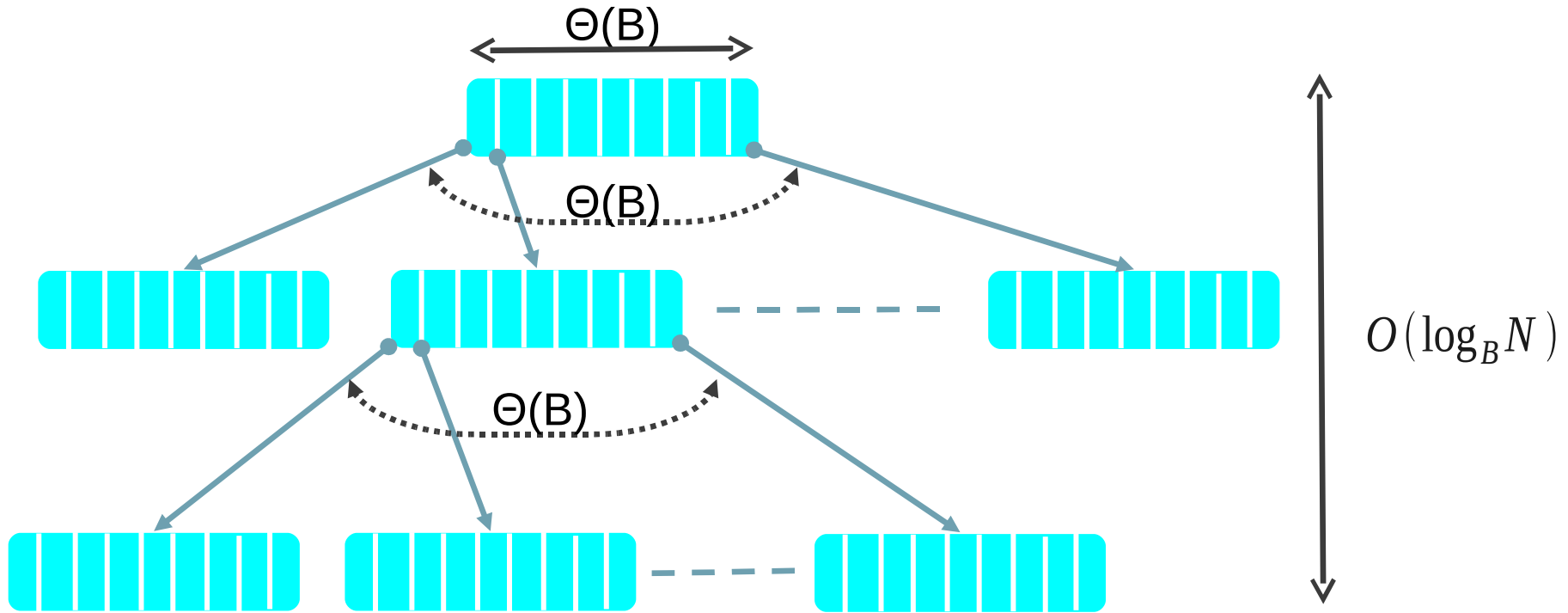
$$W(N) = O(\lg N)$$

$$Q_{naive}(N) = O(\lg N)$$

Searching in the CA Model

[Bayer and McCreight 1972]

Searching a key in an N-elements B-tree



$$W(N) = \lg B \cdot O(\log_B N)$$

$$Q_{CA}(N) = 1 \cdot O(\log_B N)$$

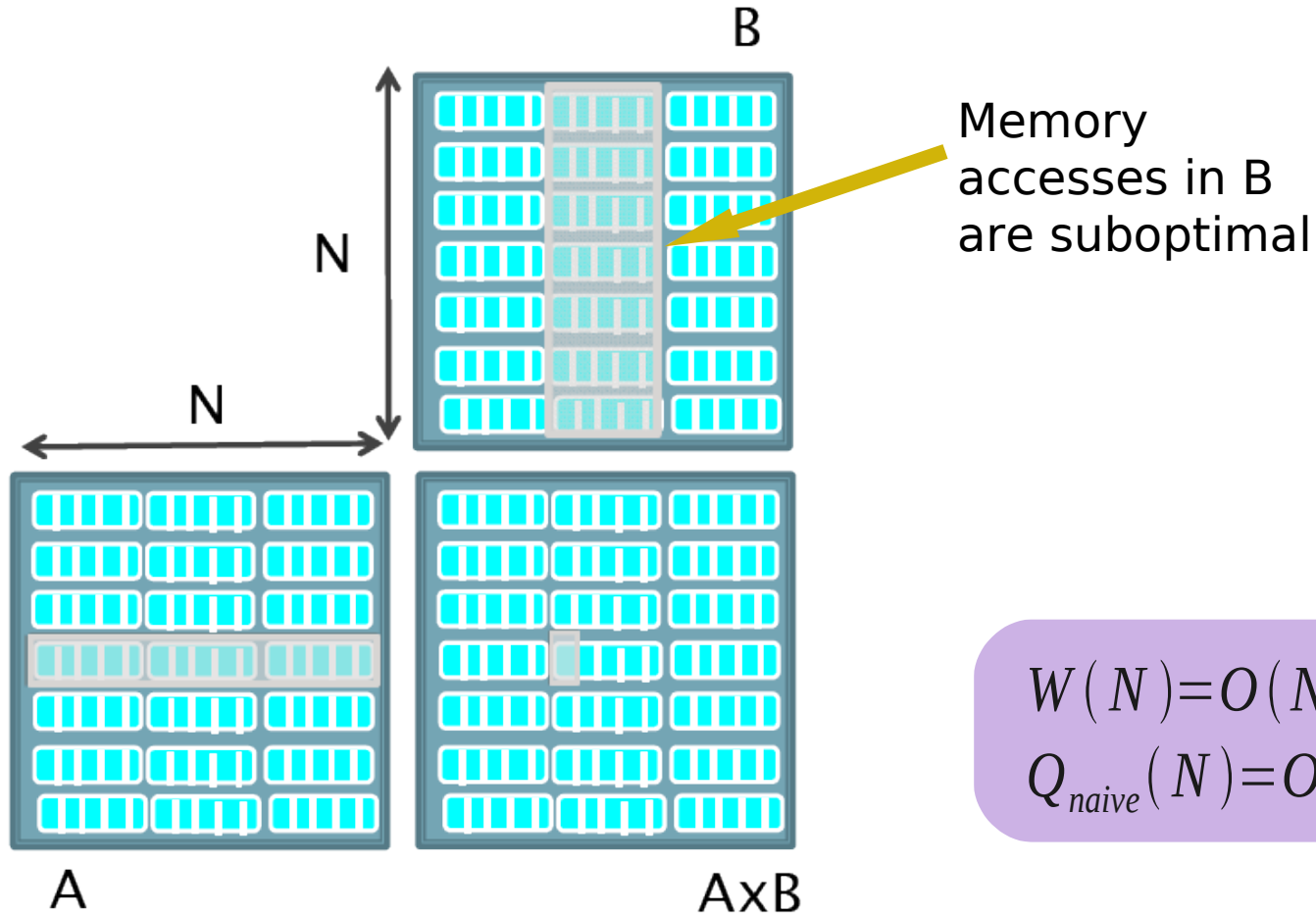
$$W(N) = O(\lg N)$$

$$Q_{naive}(N) = O(\lg N)$$

$$Q_{CA}(N) = O(\log_B N)$$

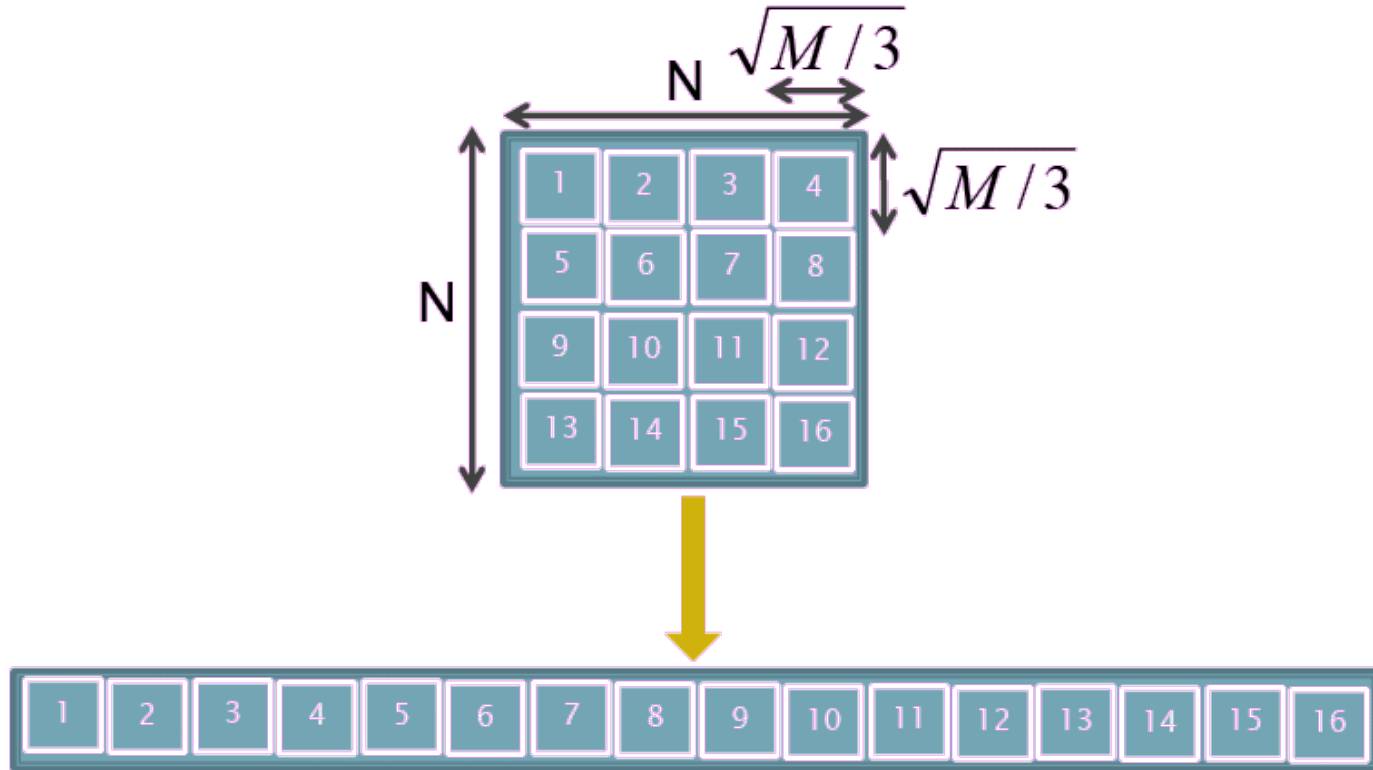
Multiplying in the CA Model

$N \times N$ matrices in row-major order: naive doesn't work



Multiplying in the CA Model

$N \times N$ matrices in blocks



Technique used in BLAS

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

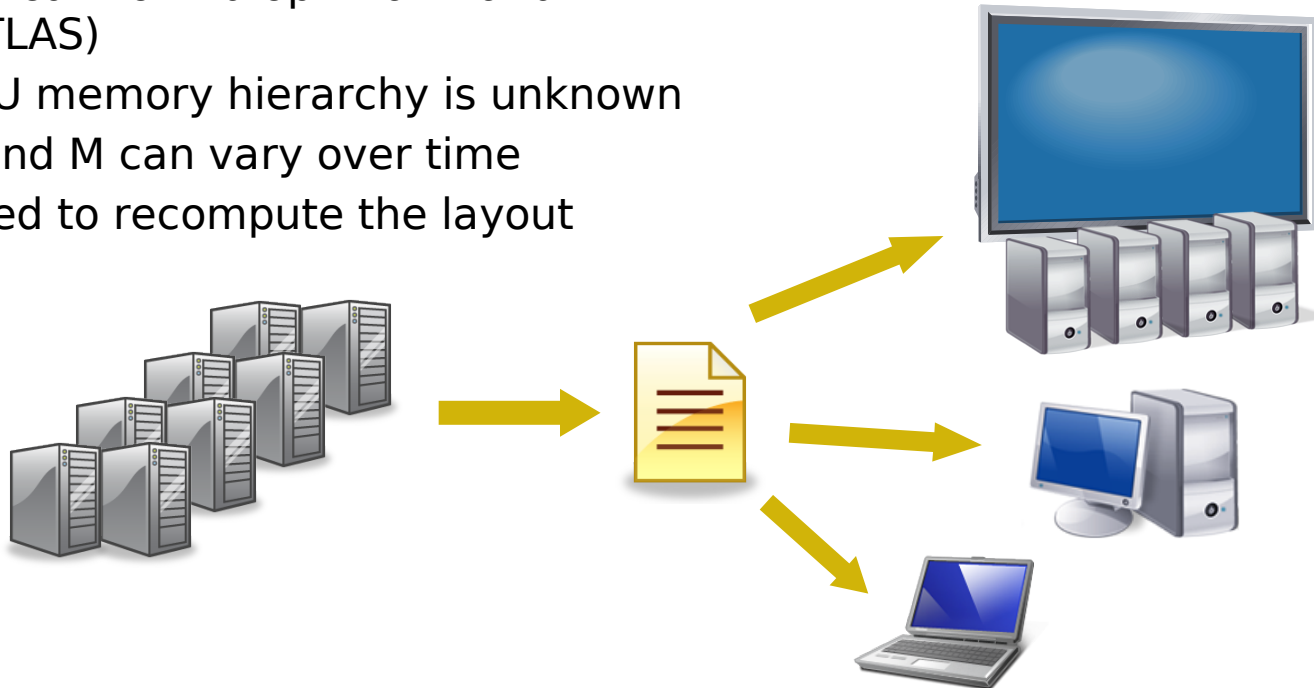
$$Q_{CA}(N) = O\left(\frac{N^3}{B\sqrt{M}}\right)$$

Drawbacks of the CA Model

- Only two levels of the memory hierarchy
 - At least 4 levels on any modern CPU
 - Even deeper with multiprocessors
- Architecture dependent
 - Difficult to find optimal B and M (ATLAS)
 - GPU memory hierarchy is unknown
 - B and M can vary over time
 - Need to recompute the layout

+ Efficient with all levels of the memory hierarchy

+ Architecture independent

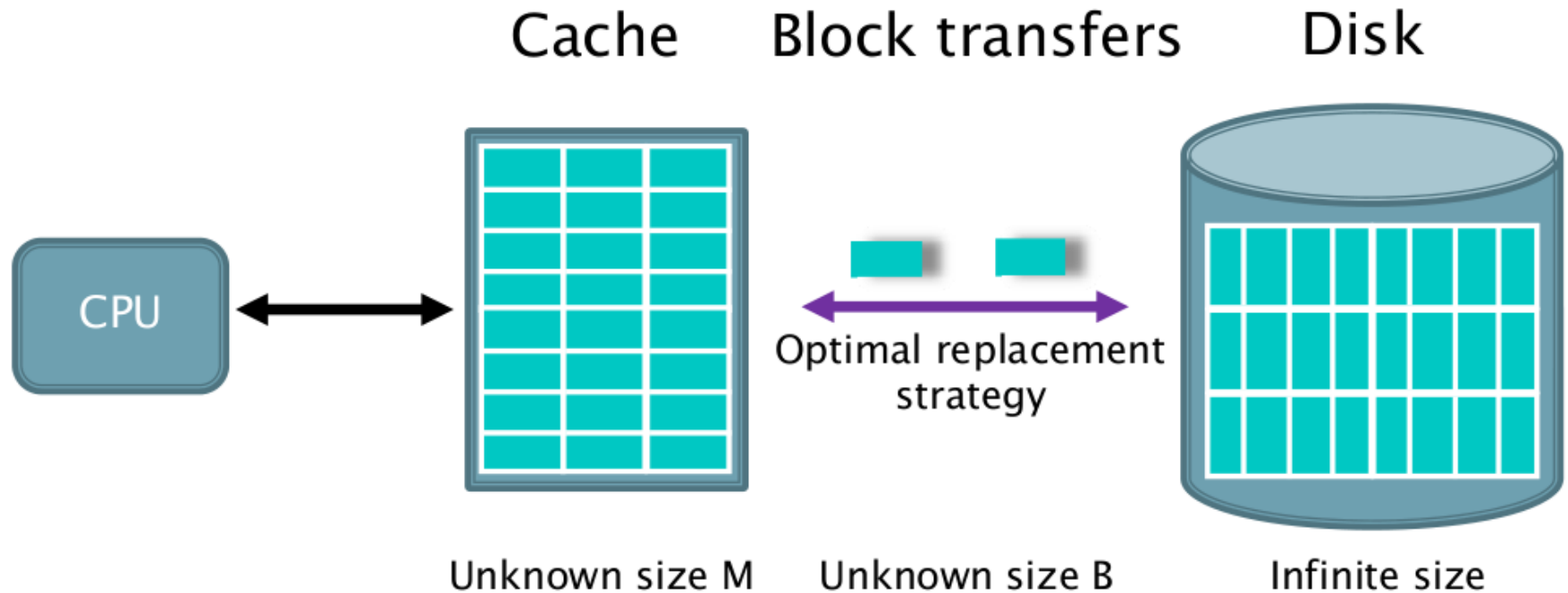


Outline

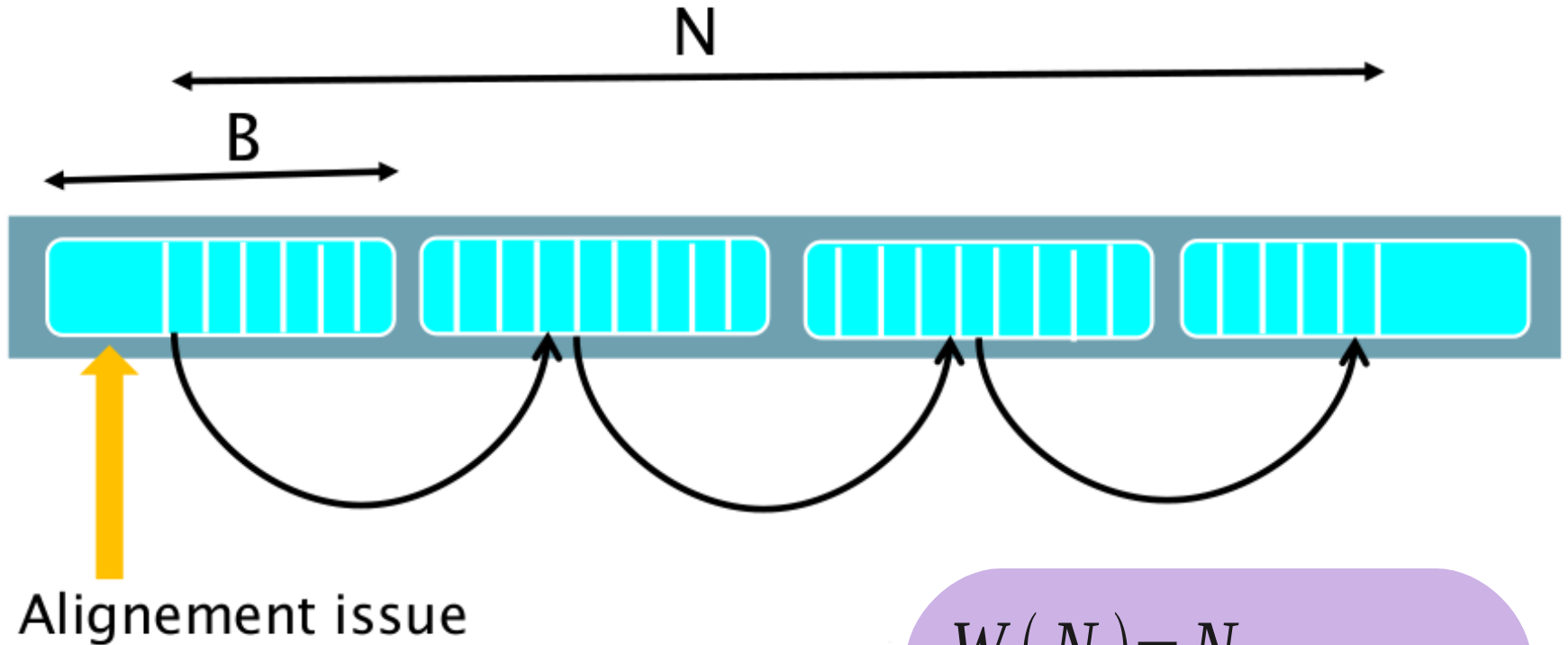
- Cache-Aware Model
- Cache-Oblivious Model
- Cache-Oblivious Mesh Layout
- Preliminary Experiments
- Conclusion & Future Work

Cache-Oblivious Model (CO)

[Frigo & al 1999]



Scanning in the CO Model



$$W(N) = N$$

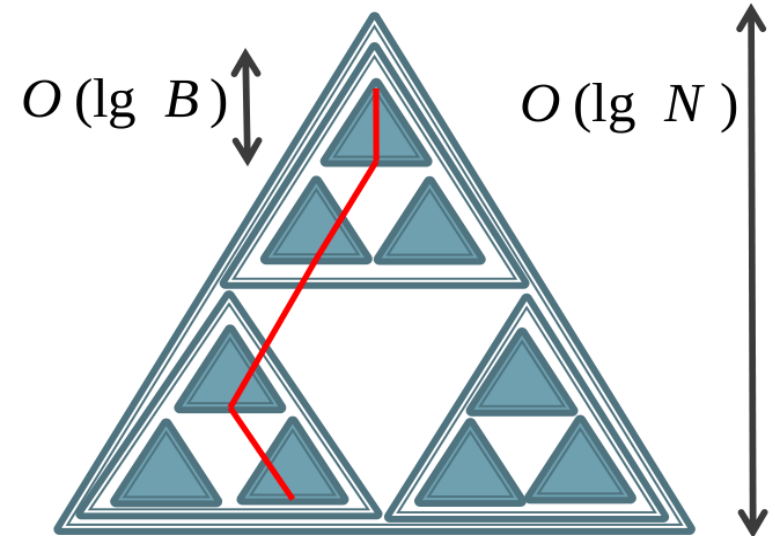
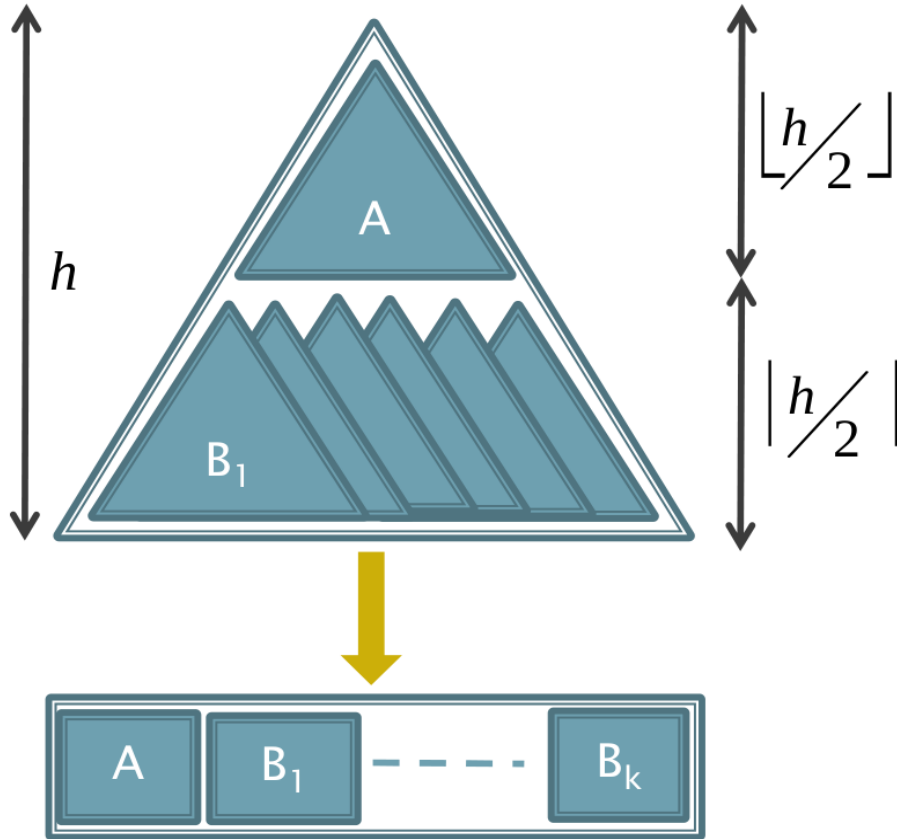
$$Q_{CA}(N) = \lceil N/B \rceil$$

$$Q_{CO}(N) = \lceil N/B \rceil + 1$$

Searching in the CO Model

[Bender et al 2000]

Binary tree mapped in memory using a recursive layout



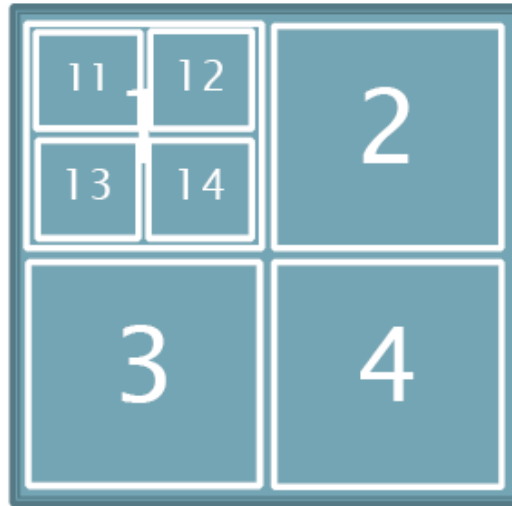
$$Q_{CO}(N) = O(1) \cdot \frac{O(\lg N)}{O(\lg B)}$$

$$Q_{CA}(N) = O(\log_B N)$$

$$Q_{CO}(N) = O(\log_B N)$$

Multiplying in the CO Model

D&C matrix multiplication using a recursive layout



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

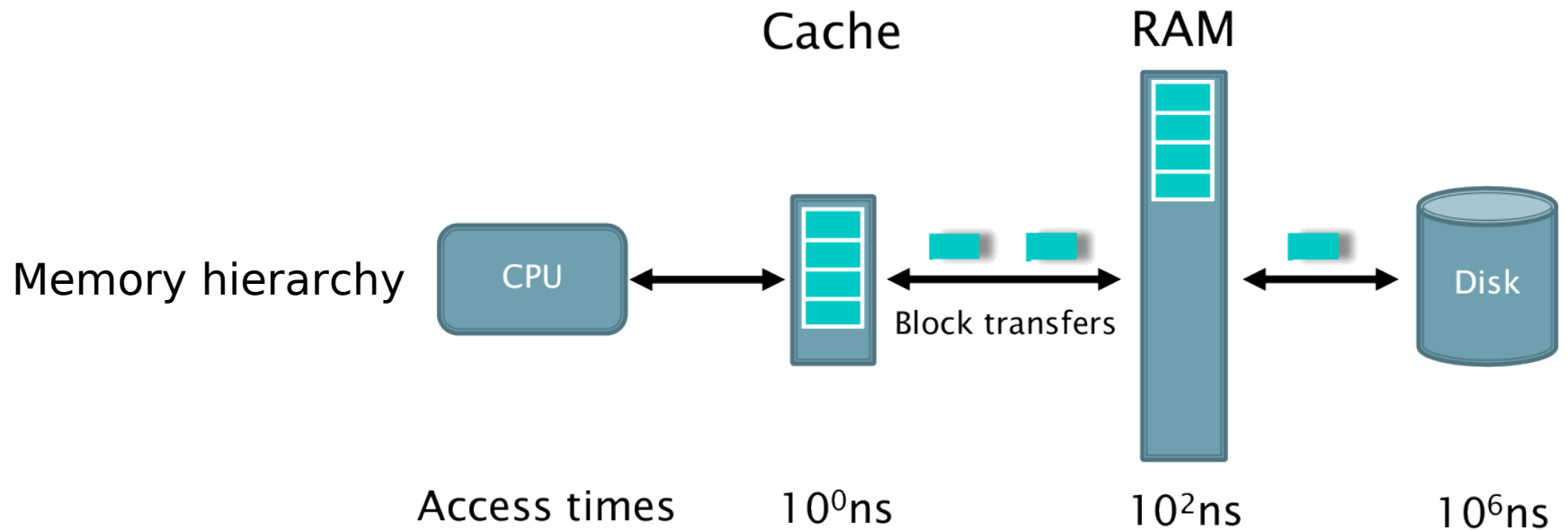
$$Q_{CA}(N) = O\left(\frac{N^3}{B\sqrt{M}}\right)$$

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

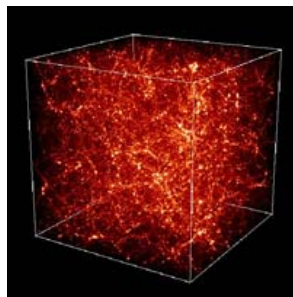
Outline

- Cache-Aware Model
- Cache-Oblivious Model
- Cache-Oblivious Mesh Layout
- Preliminary Experiments
- Conclusion & Future Work

Cache-Oblivious Ordering of a Mesh



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

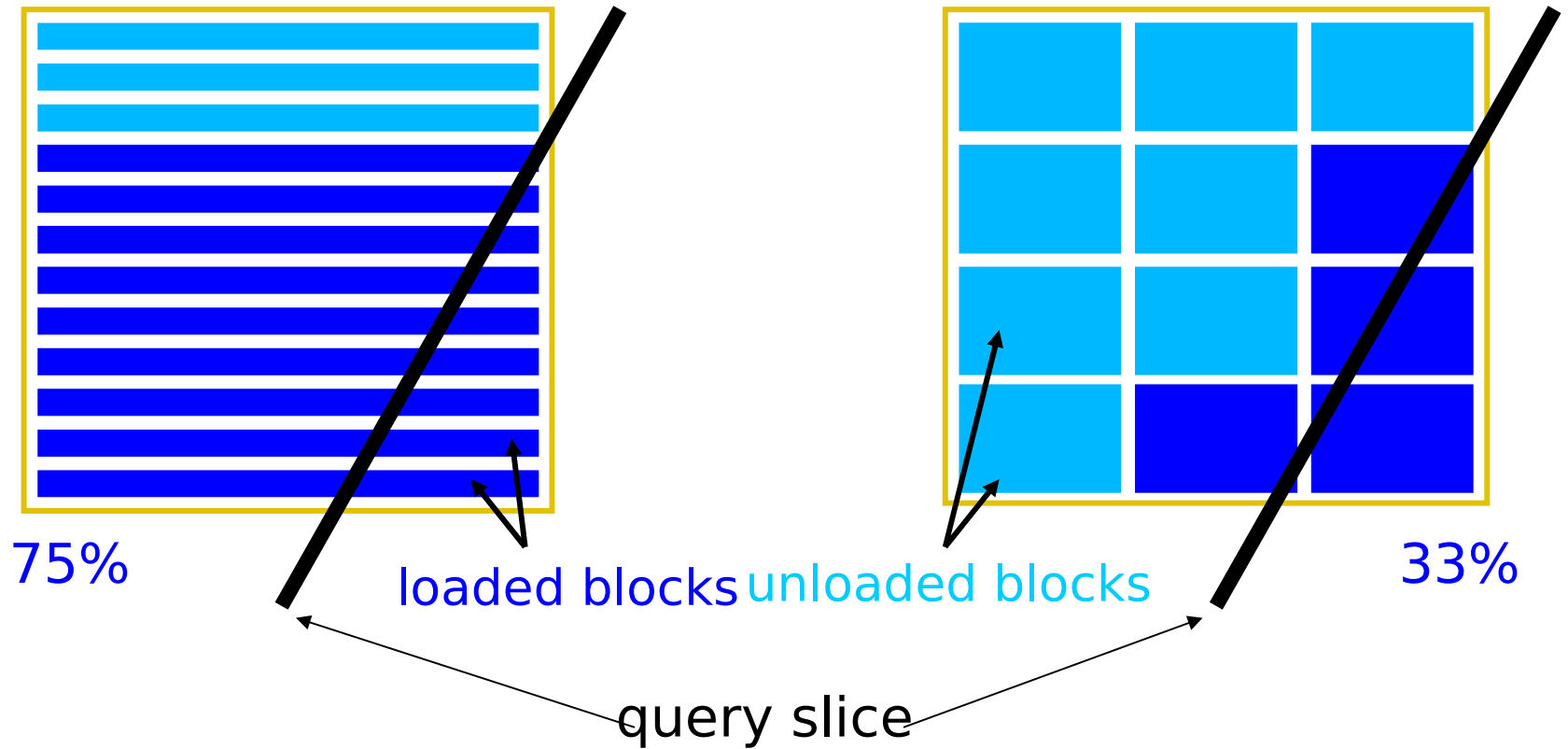


algorithm

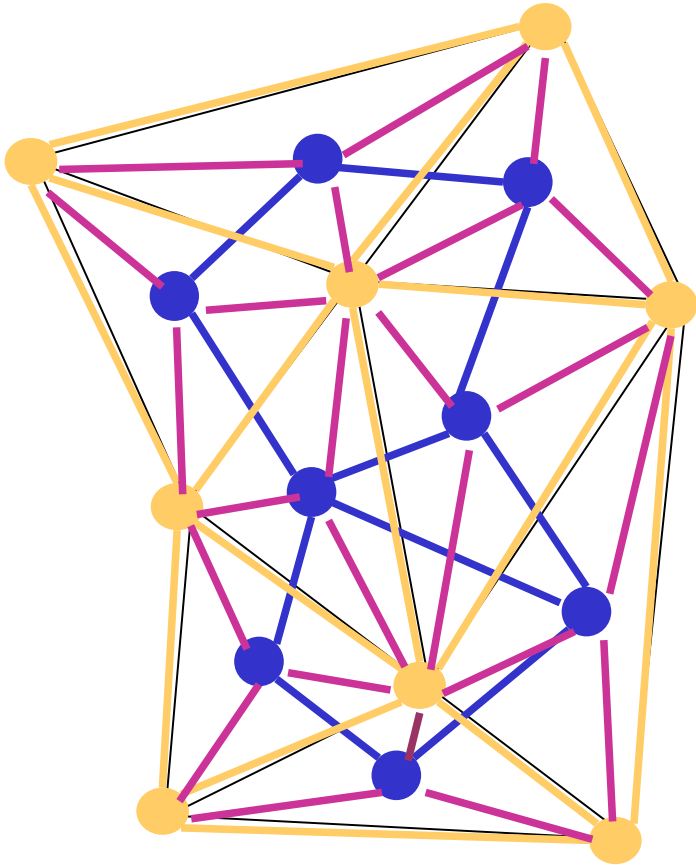


Idea

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



Graph of Sequential Accesses



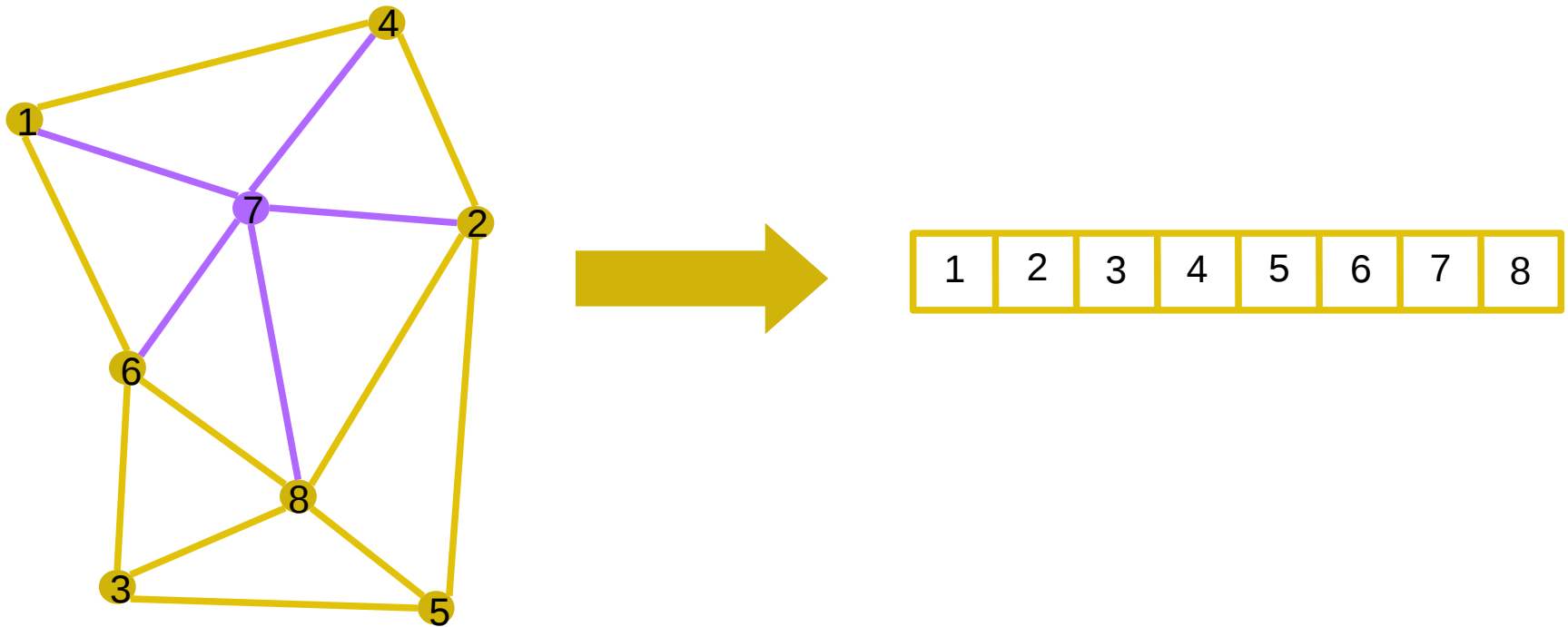
G_1 : sequential access between triangles

G_2 : sequential access between vertices

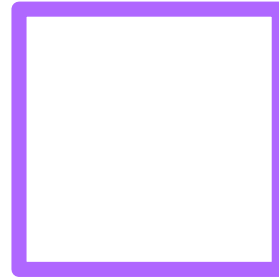
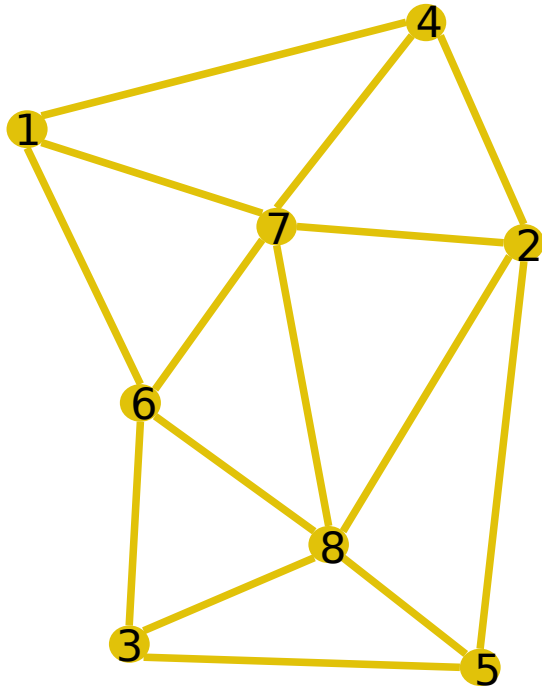
G_3 : both

Mesh Layout Problem

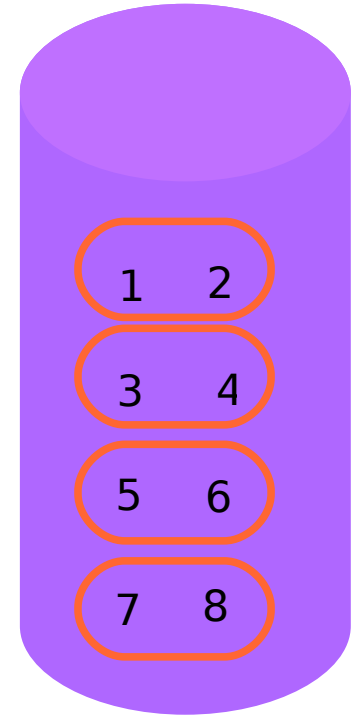
Minimize # of cache misses if each node touches all its neighbors



Example



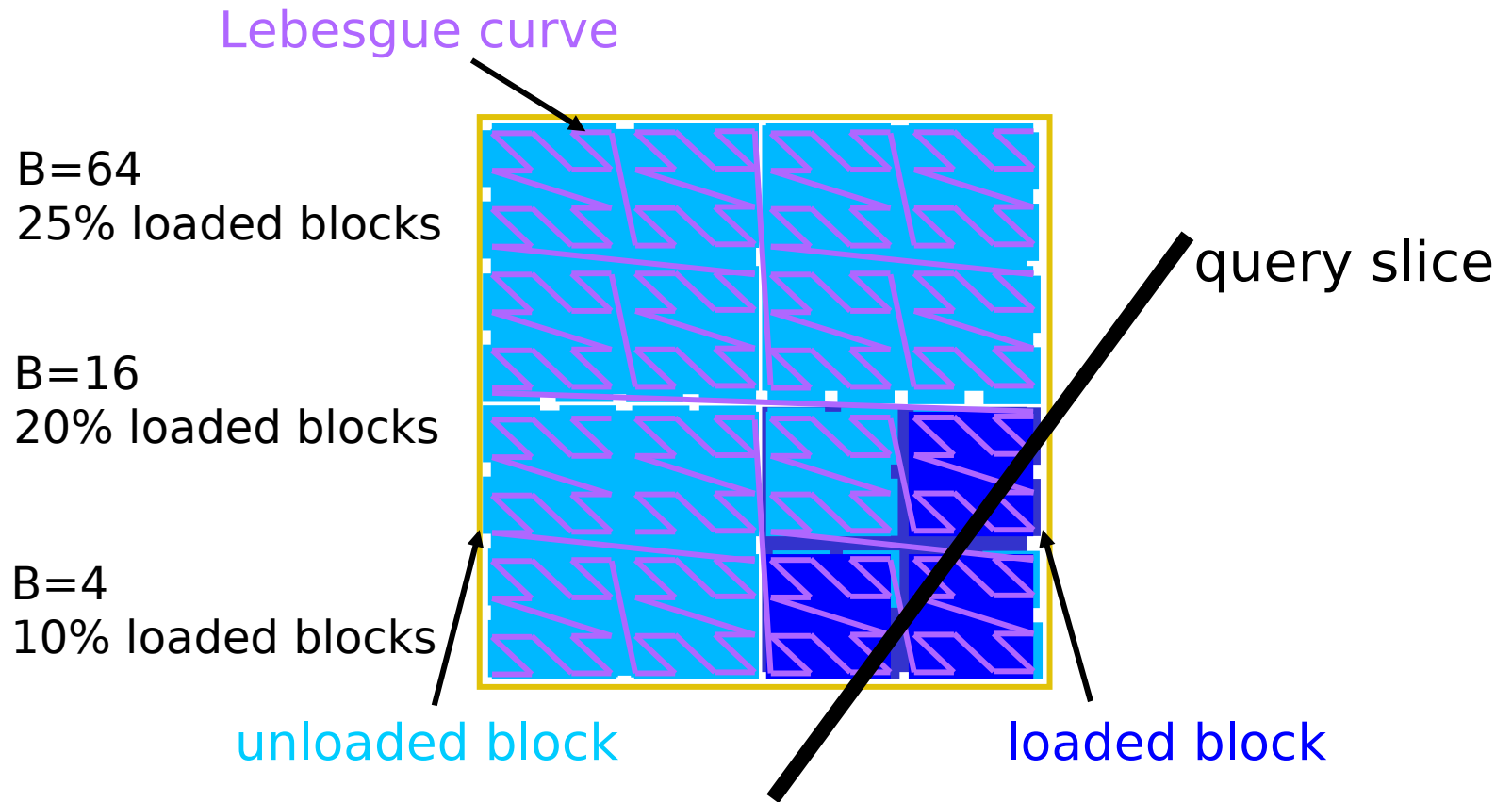
cache ($B=2, M=4$)



disk

25 cache misses

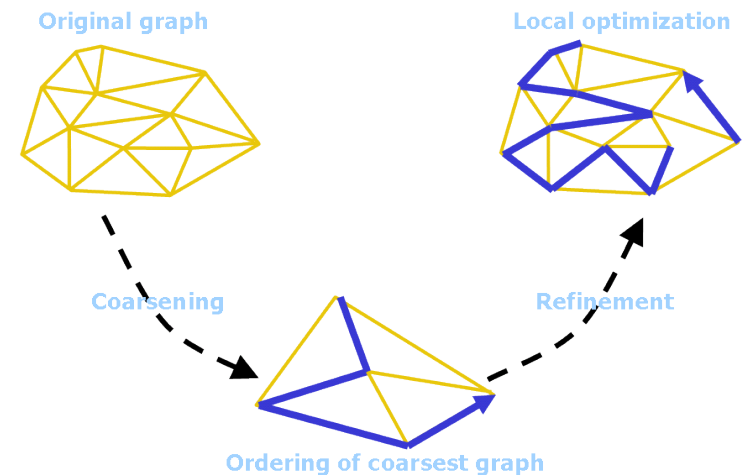
Regular Meshes: Space-Filling Curves



Unstructured Meshes

[Pascucci & al 2005]

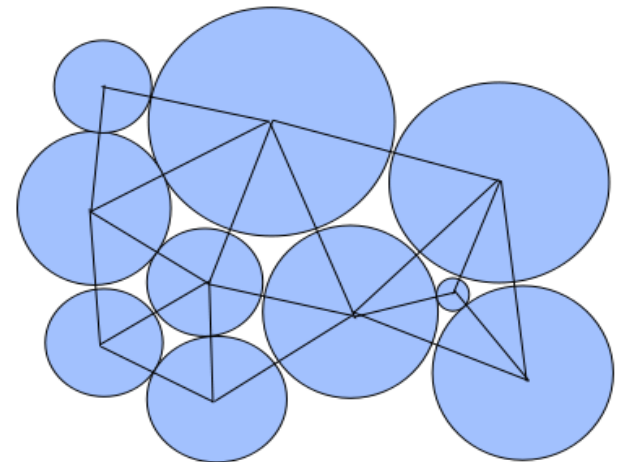
- Heuristic algorithm based on multi-level optimization
 - slow
 - high memory usage
- Good experimental results (2-5x improvement)
- But no guarantee on
 - time to compute the layout
 - layout quality



Overlap graphs

[Miller & al 98]

- Planar graphs
 - Edge between two nodes iff two circles touch each other
- Overlap graphs (extension in dimension d)
 - Edge between two nodes iff two spheres touch each other
- Overlap graphs contain well-shaped meshes



Separator for overlap graphs

[Miller & al 98]

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

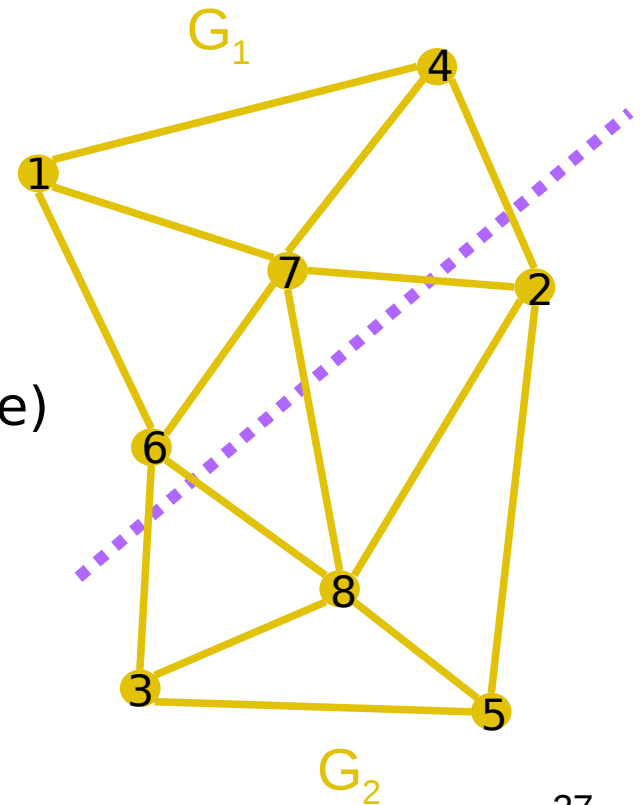
- Planar graphs [Lipton-Tarjan]

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

- Overlap graphs (randomized linear time)

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

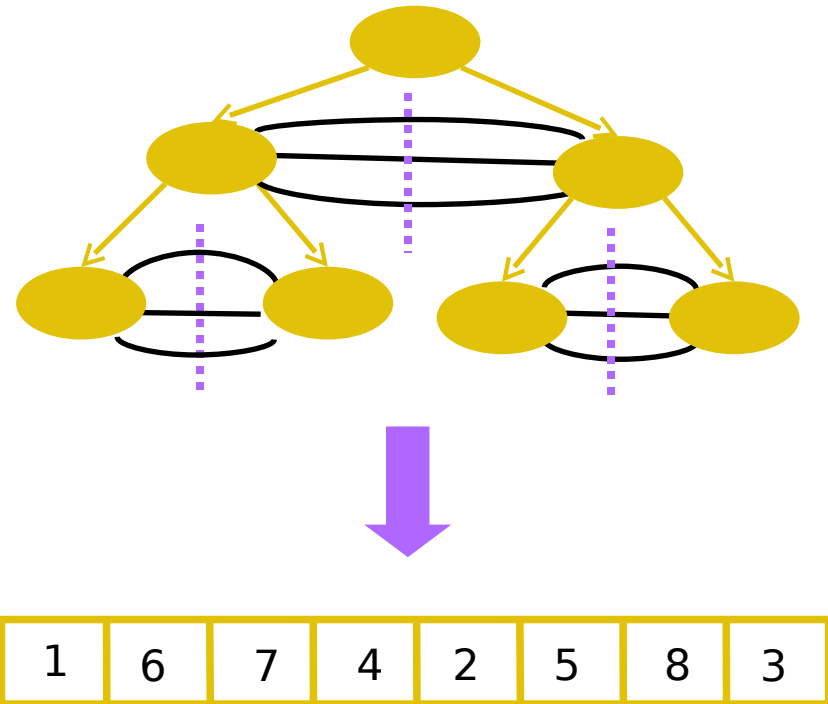
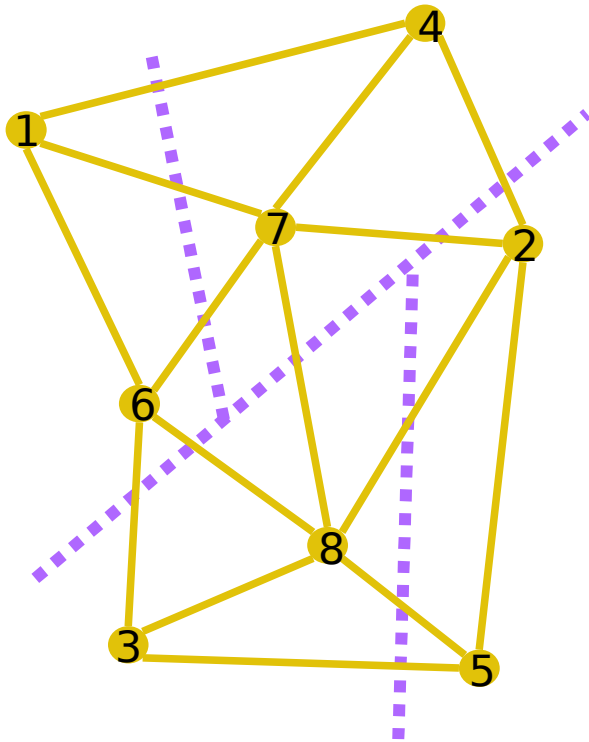
$$E(G_1, G_2) = O(|G|^{1-1/d})$$



Our layout

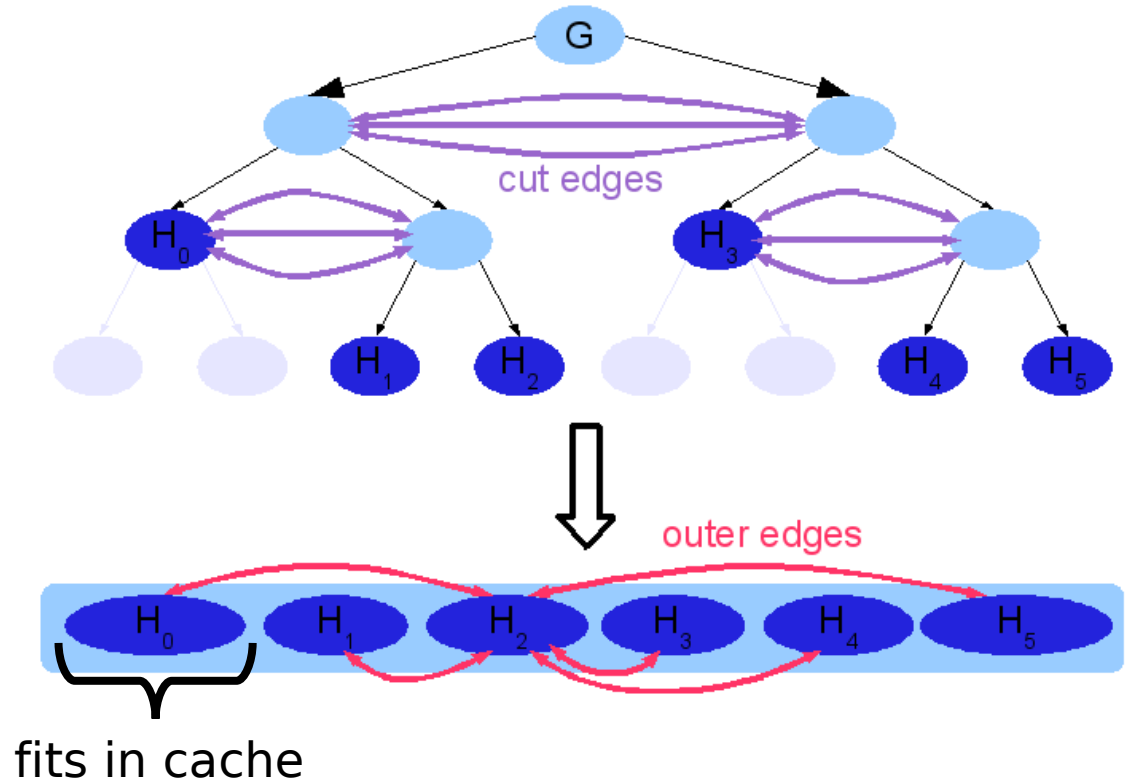
$$W(N) = O(N \log N)$$

- Recursively cut the mesh
- The order of the leaves gives the layout



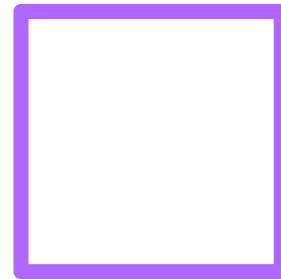
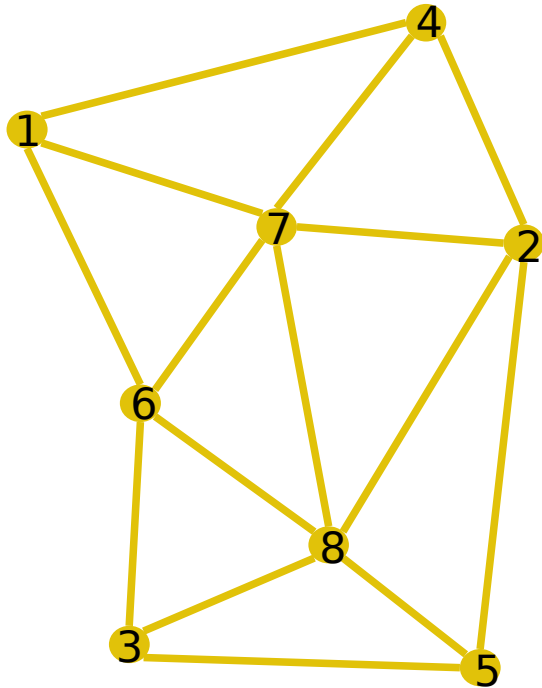
Guarantee on the Quality of the Layout

- 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

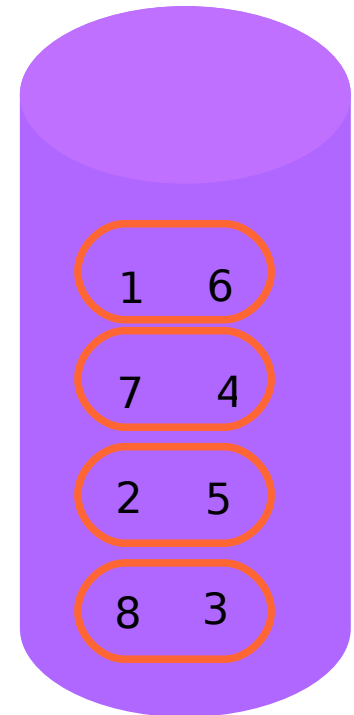


Theorem: Our layout guarantees that a traversal of an $O(N)$ -size d -dim mesh causes less than $O(N/B + N/M^{1/d})$ cache misses

Back to the Example



cache ($B=2, M=4$)



disk

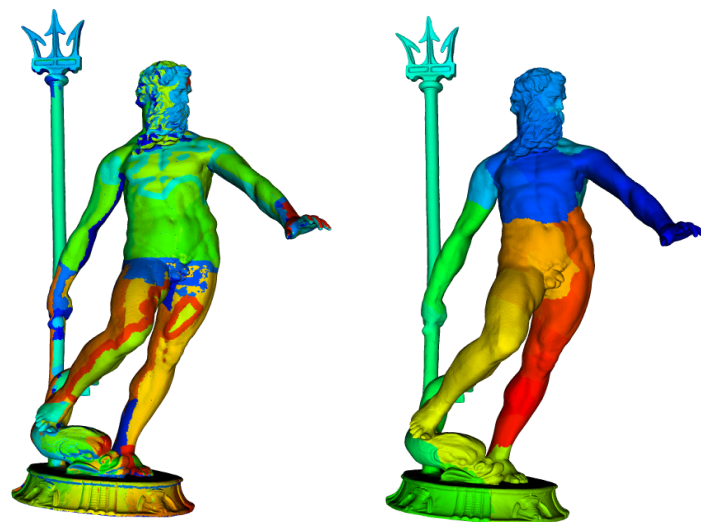
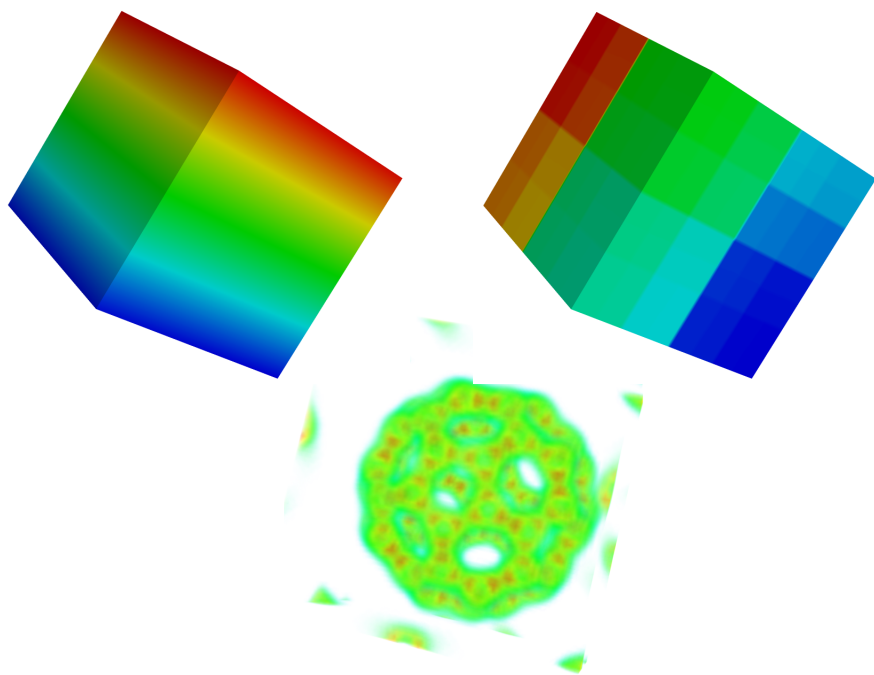


12 cache misses (25 previously)

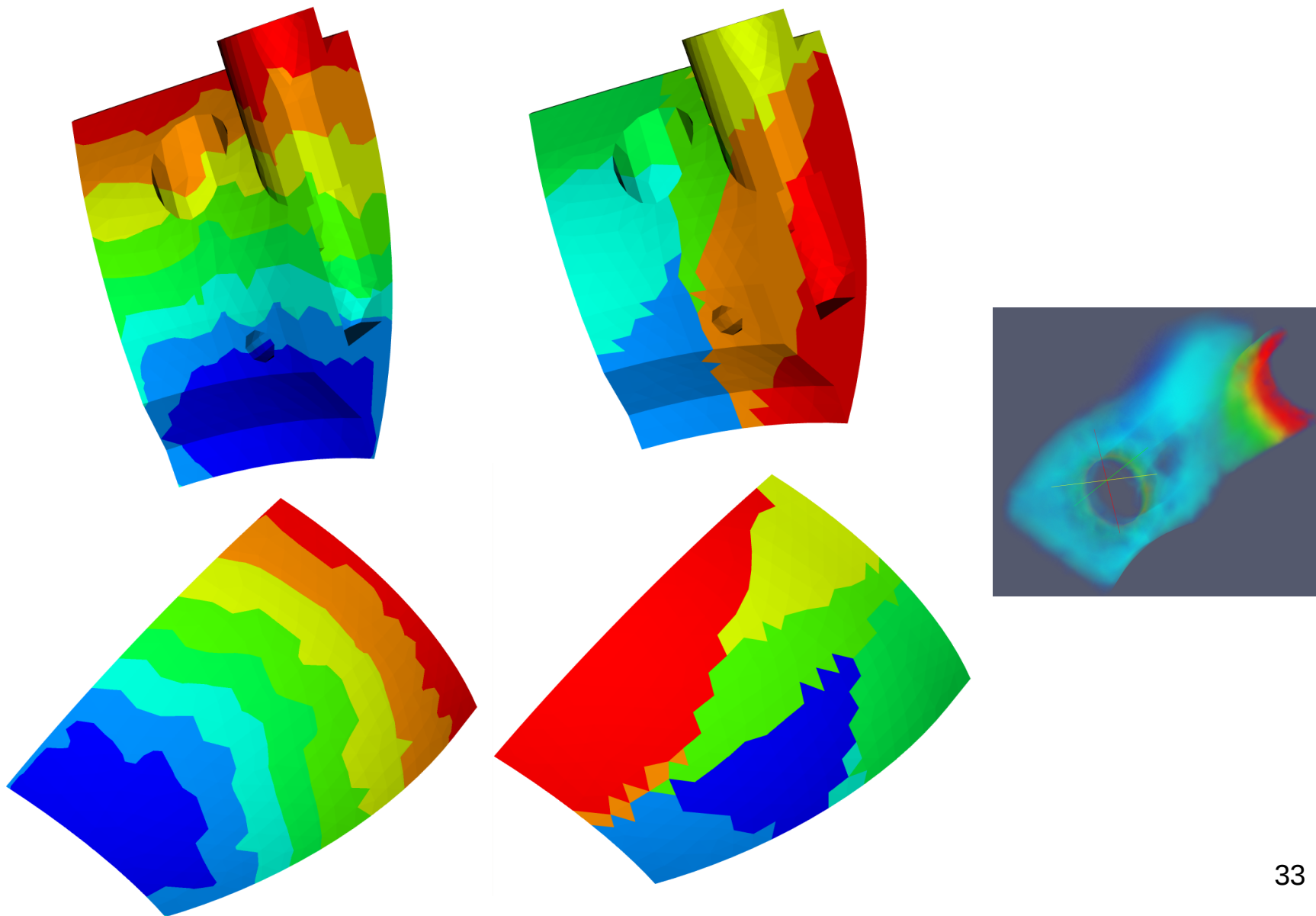
Outline

- Cache-Aware Model
- Cache-Oblivious Model
- Cache-Oblivious Mesh Layout
- Preliminary Experiments
- Conclusion & Future Work

Preliminary Experiments

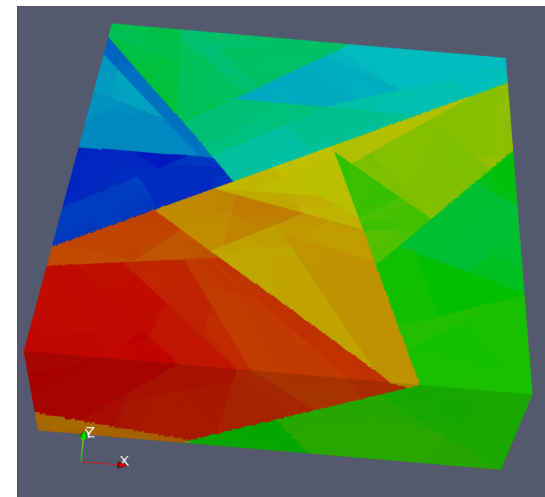
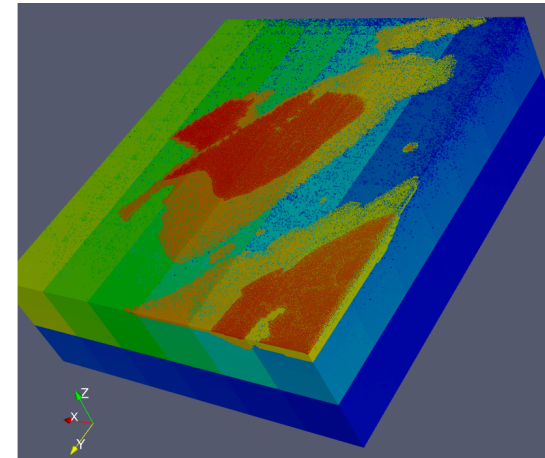
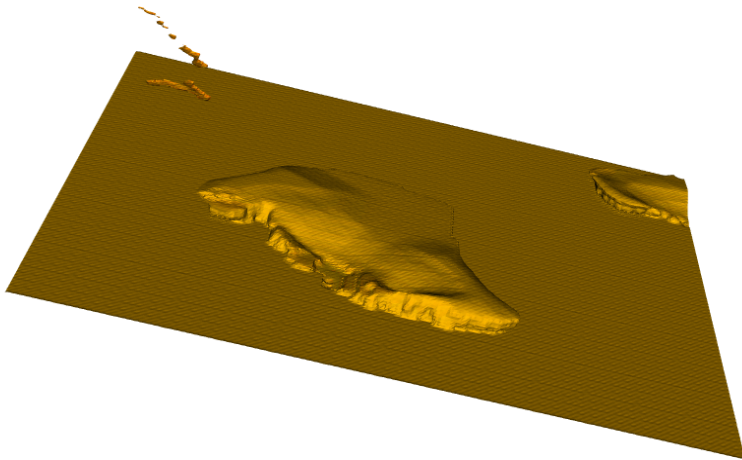


Preliminary Experiments



Isosurface extraction

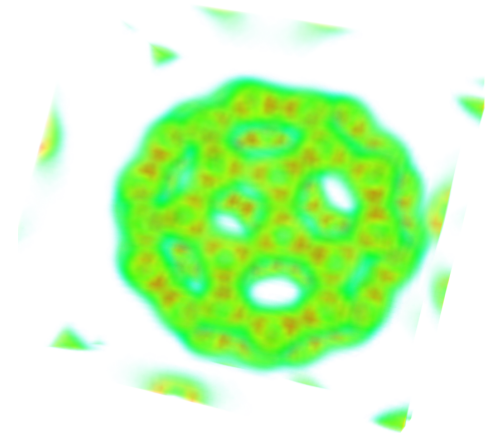
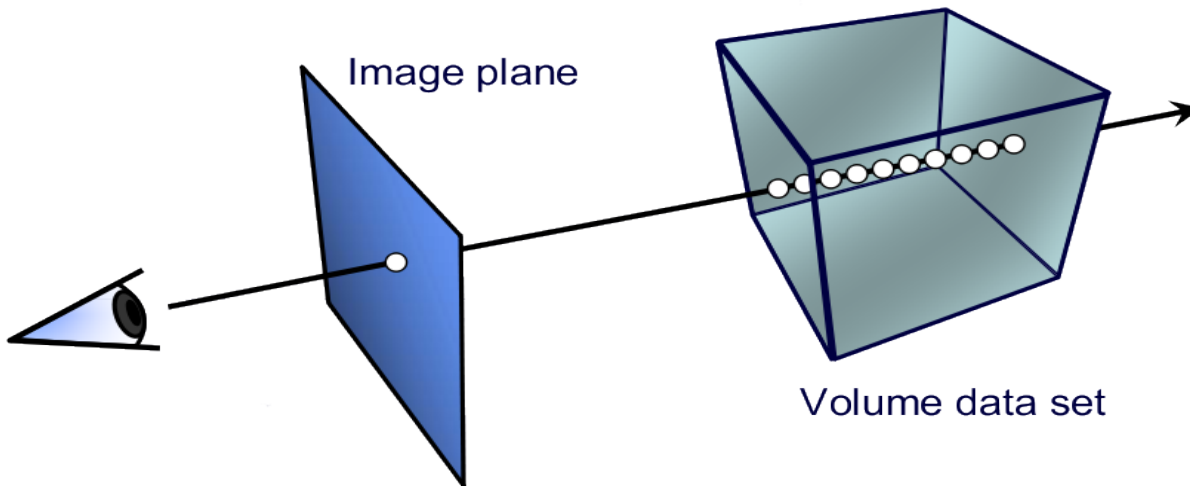
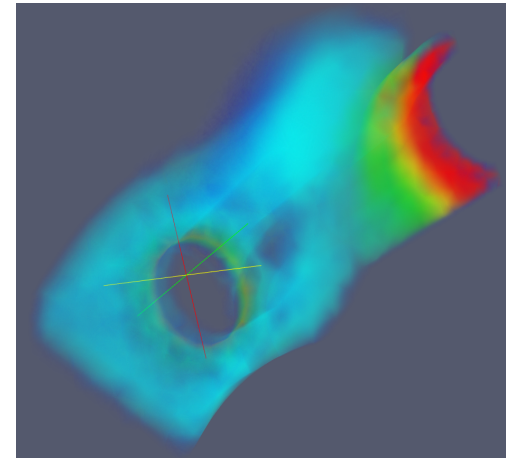
- Marching Cube (On CPU with VTK)
- Marching Cube (On GPU with CUDA)
- Using a tree to speed up cell search
 - Tree from VTK
 - Our tree (min-max tree)



2,4M points 34
14M tetras

Volume Rendering

- Using VTK on CPU (several methods)
- Our code in CUDA on GPU



Outline

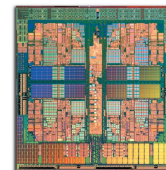
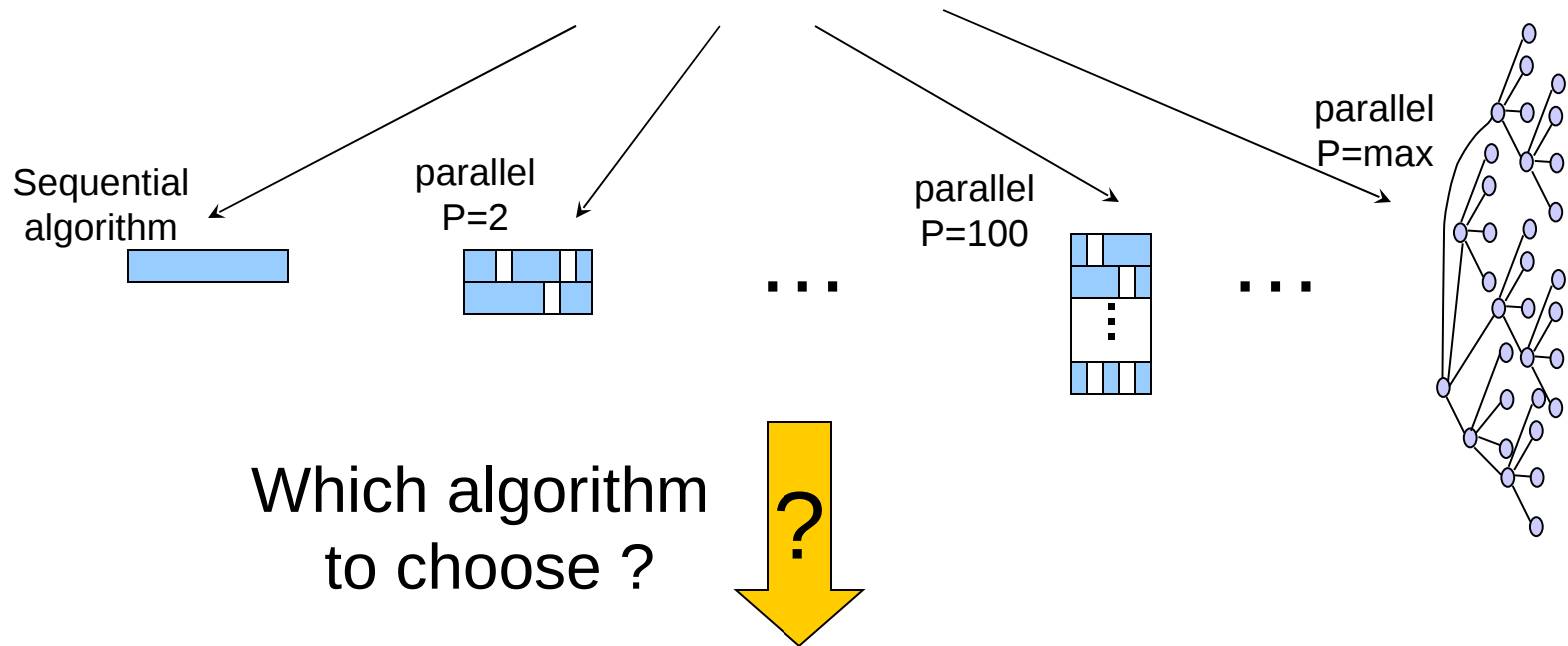
- Cache-Aware Model
- Cache-Oblivious Model
- Cache-Oblivious Mesh Layout
- Preliminary Experiments
- Conclusion & Future Work

Conclusion & Future Work

- Our algorithm
 - Fast
 - Quality & time guarantee
 - Architecture independent
- Experimental validation
 - Difficult to find big 3D unstructured meshes
 - Multiresolution methods
- Visualization in parallel
 - How do we keep the good cache performance ?

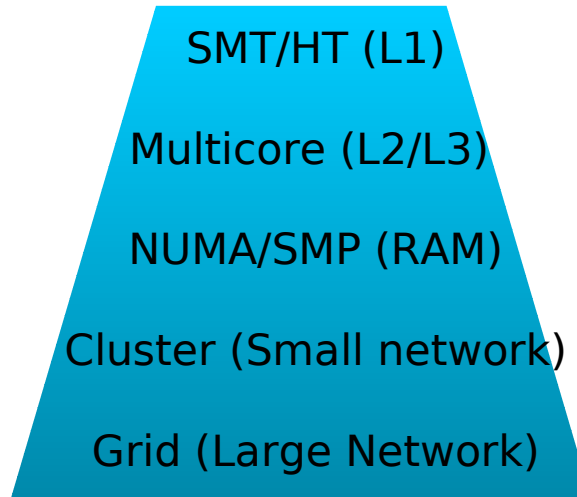
Processor oblivious algorithms

*To design a single efficient algorithm
with provable performances on an arbitrary architecture*



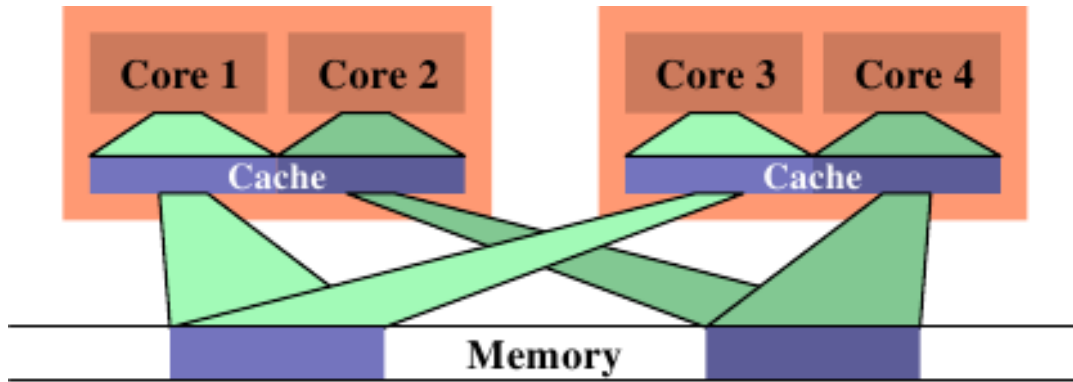
A processor & cache oblivious model

- Deeper memory hierarchy



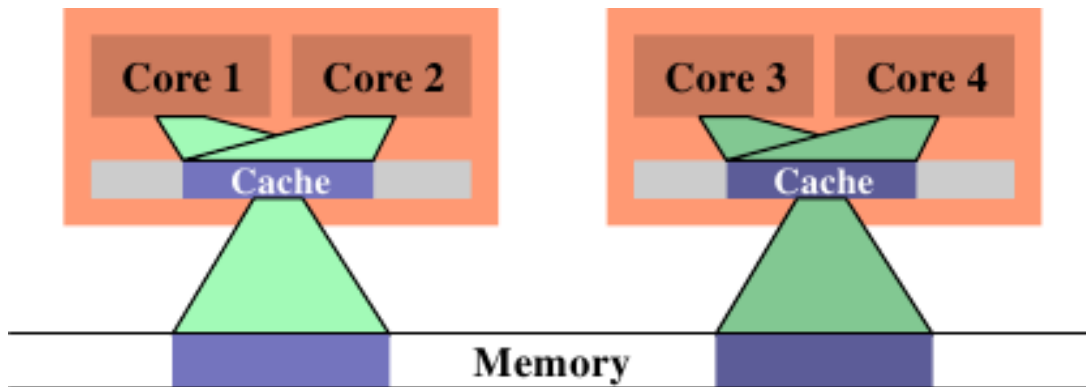
- Processor + Cache → Communications
 - sharing a cache ↔ low cost communication

Interaction cache / scheduling



Can be worse than
each core has a cache of half size

Cache sharing



Can be as good as
each core has a cache of full size

Questions?

Thank you