Cache-Coherent Layouts

Sung-Eui Yoon
(윤성의)

Course URL:
http://jupiter.kaist.ac.kr/~sungeui/SGA/

KAIST

Course Administration

- **Make progresses on your chosen topic**
  - **Read papers now, not later!**
  - **Think about pros and cons of each paper**
  - **Think about how we can further improve**
  - **Write down toward the mid-term report, whose deadline is Nov-6**

Organization of Report

- **Introduction**
  - **State a topic and a problem that you want to address**
  - **Give motivations**
  - **Present your main idea (and results)**
- **Related work**
  - **Identify a few major categories related to the topic**
  - **Emphasize benefits over your method**

Organization of Report

- **Overview**
  - **State the problem in detail**
  - **Present your idea**
  - **Why your idea address the problem**

Your mid-term report should have introduction, related work, and overview sections

Organization of Report

- **Main body of the papers**
  - **Describe your idea/solutions in detail**
- **Implementation & results**
  - **Describe your implementation and results**
  - **If you didn't implement, please provide a rough implementation sketch and the expected results**
- **Conclusion**
  - **Summary your topic, problem, and your idea**
  - **Emphasize results/benefits of your idea**
  - **Lay out future work**

At the Previous Classes

- **Studied visibility culling and LOD techniques for rasterization and ray tracing**
  - **Reduced the model complexity**

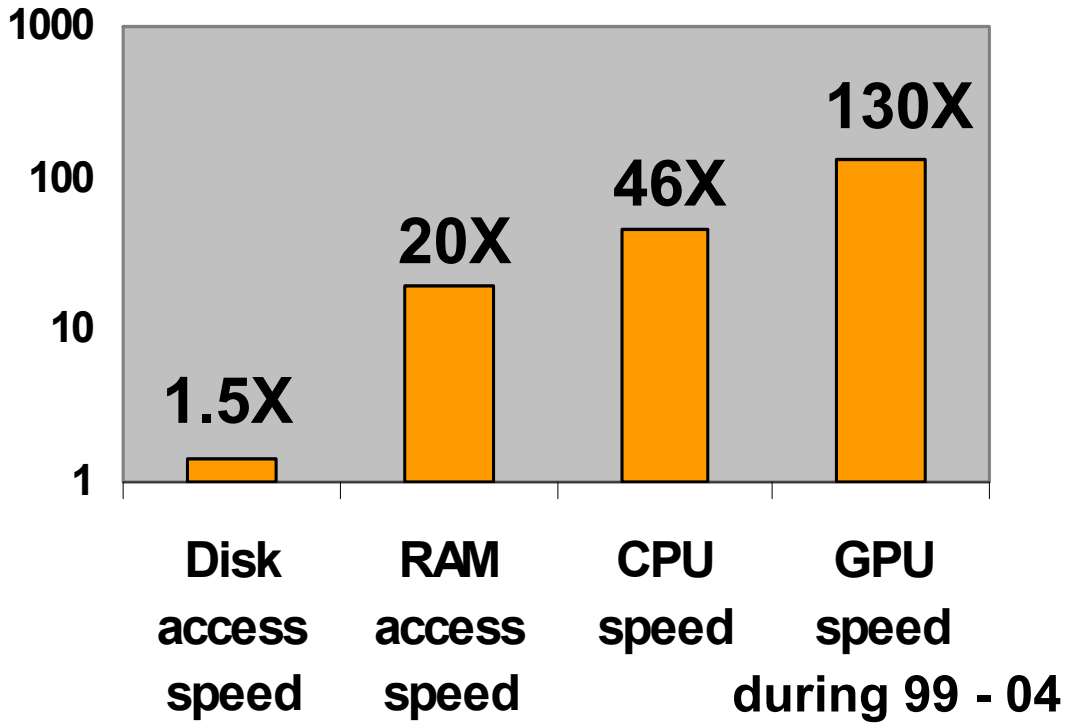
In this Class

- **Will study cache-coherent layouts of meshes, graphs, hierarchies**
  - **Re-organize the data for efficient geometric processing and rendering**

Motivation

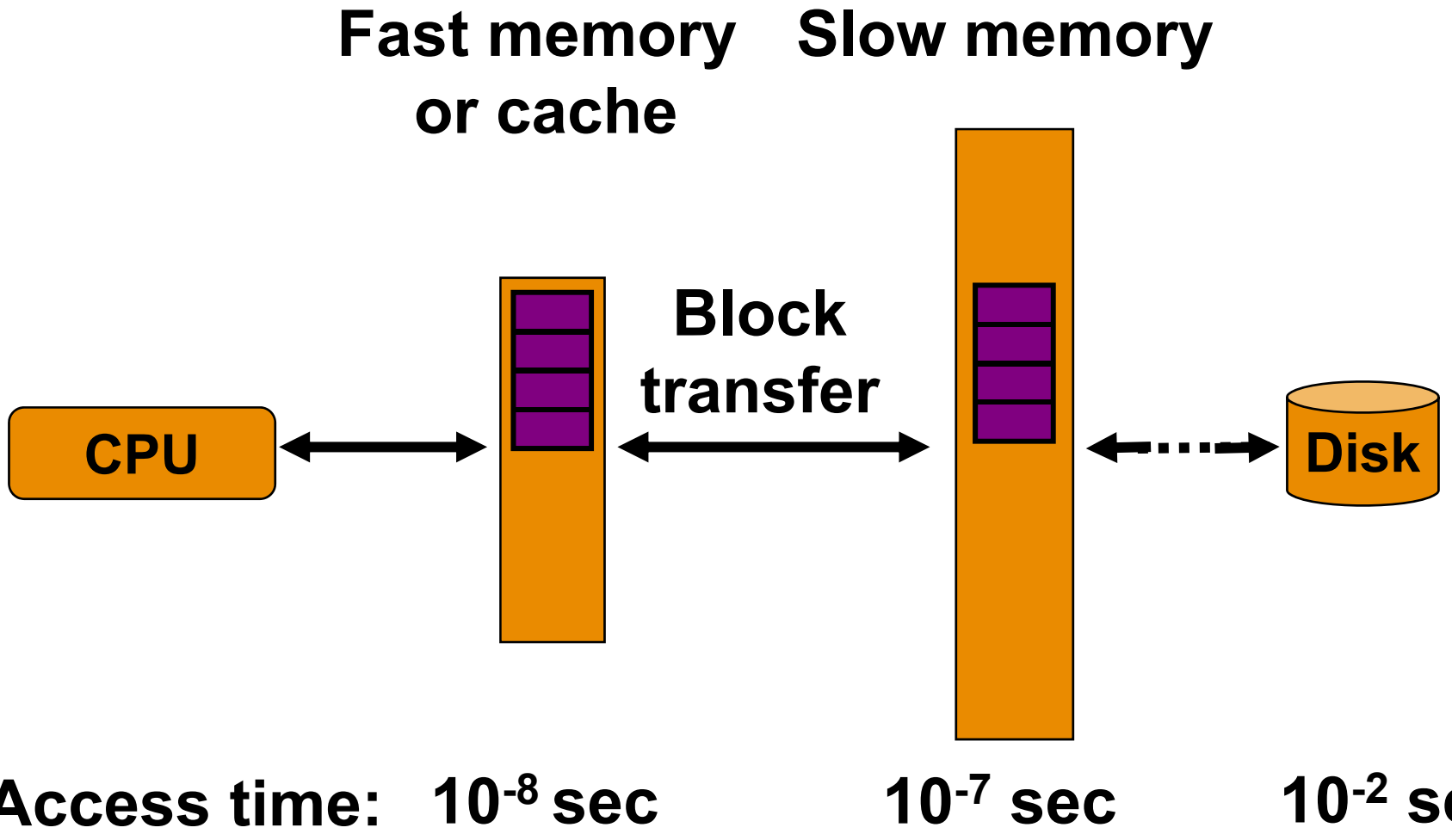
- Lower growth rate of data access speed

Accumulated growth rate during 1993 – 2004 (log scale)



Courtesy: Anselmo Lastra, http://www.hcibook.com/e3/online/moores-law/

Memory Hierarchies and Block-Based Caches

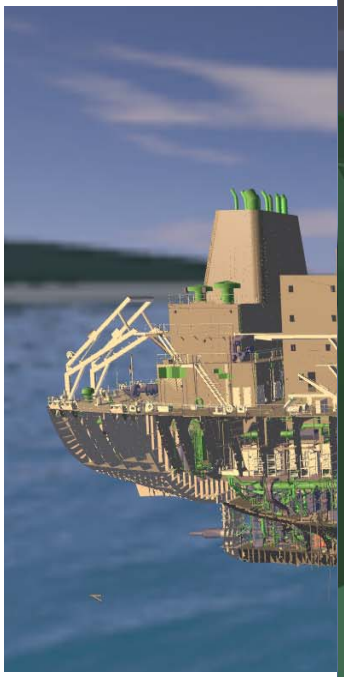


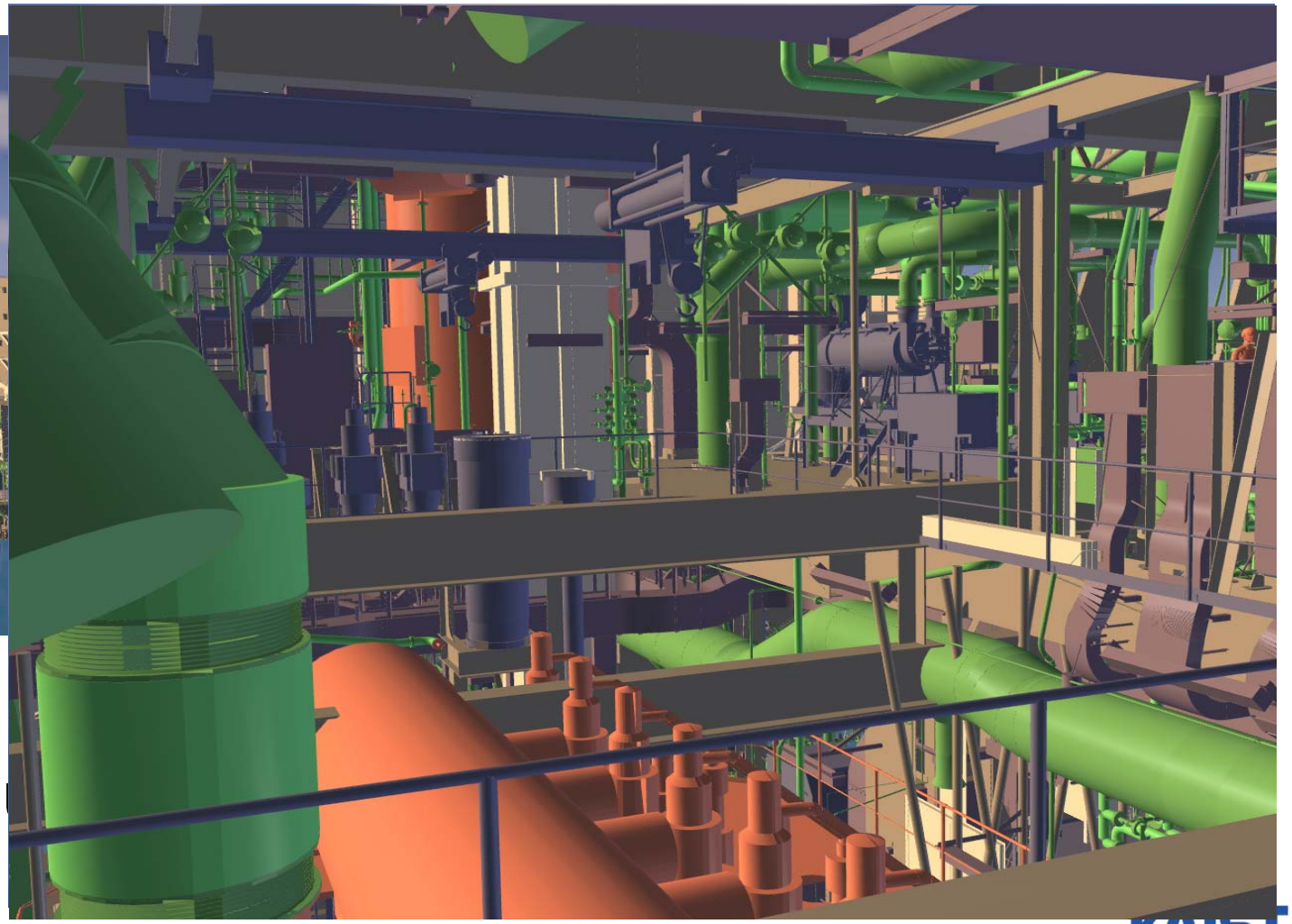
KAIST

Cache-Coherent Layouts

- **Stores related data closely in the 1D layout**
- **Cache-Aware**
  - **Optimized for particular cache parameters (e.g., block size)**
- **Cache-Oblivious**
  - **Minimizes data access time without any knowledge of cache parameters**
  - **Directly applicable to various hardware and memory hierarchies**

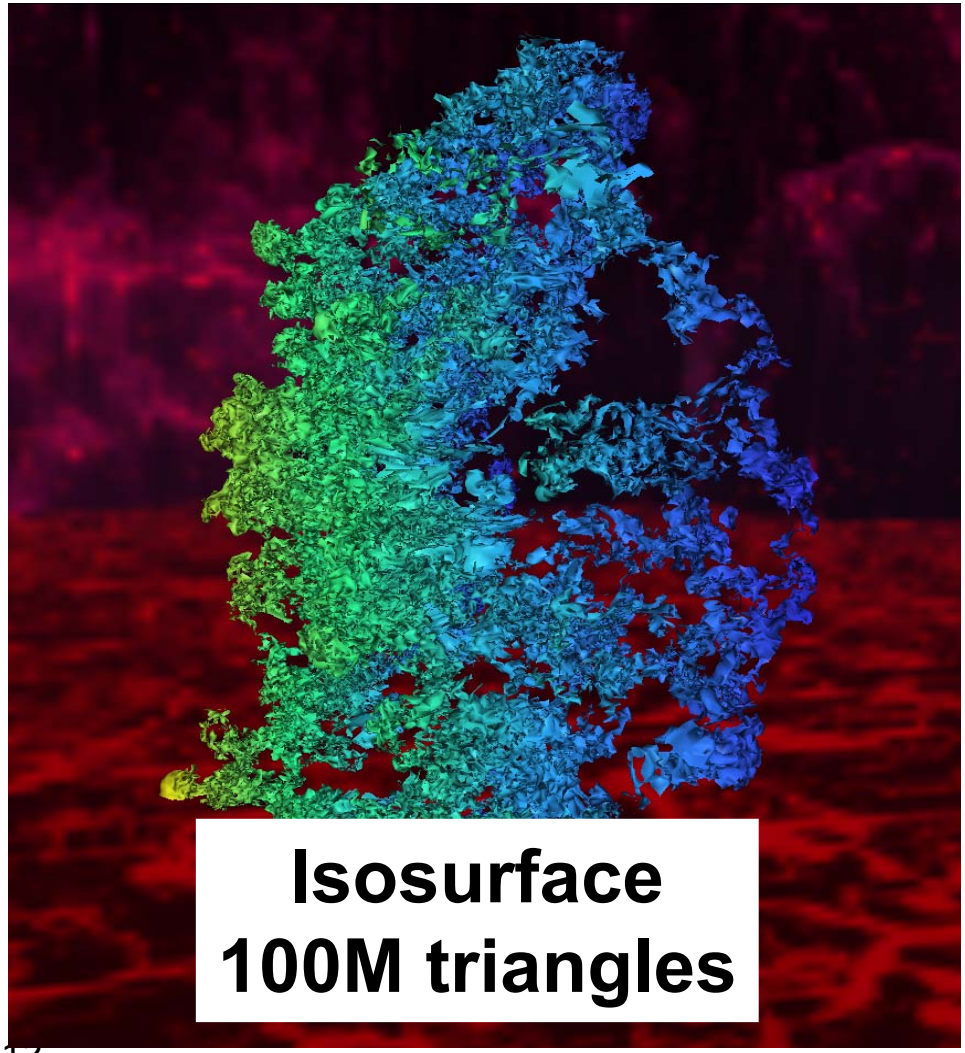
CAD Model – Double Eagle Tanker Model

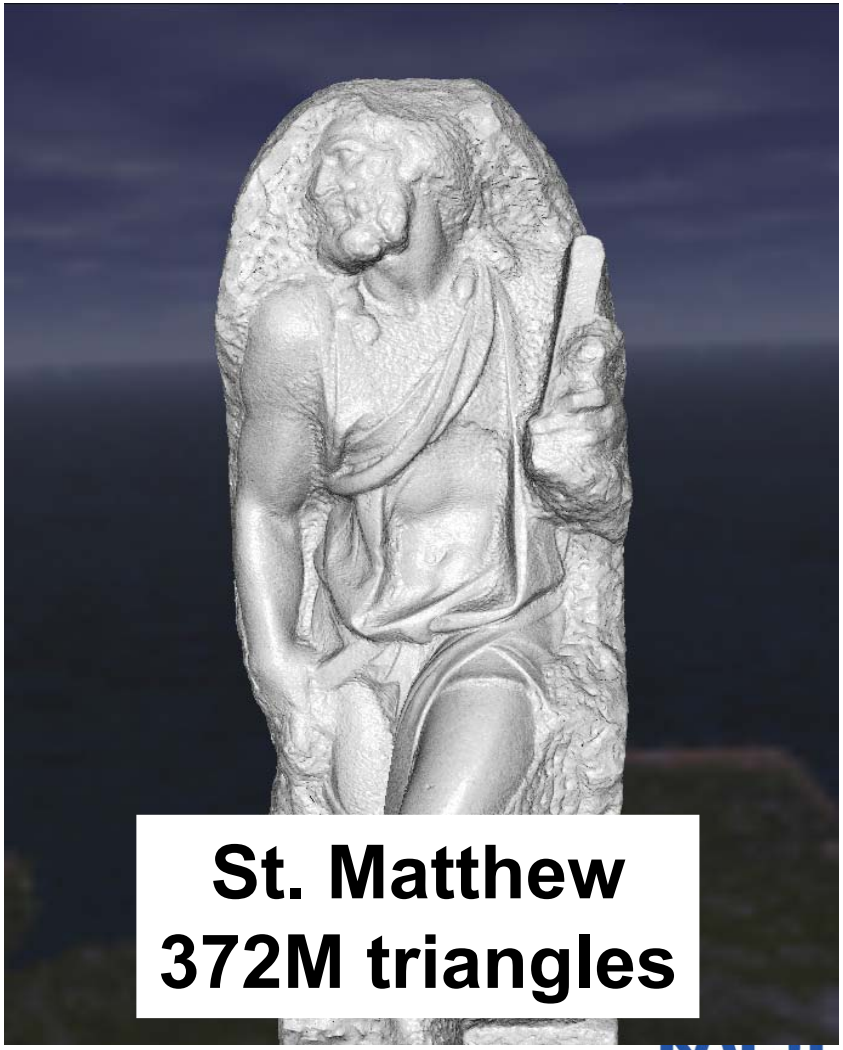




Irreg

Isosurface and Scanned Models





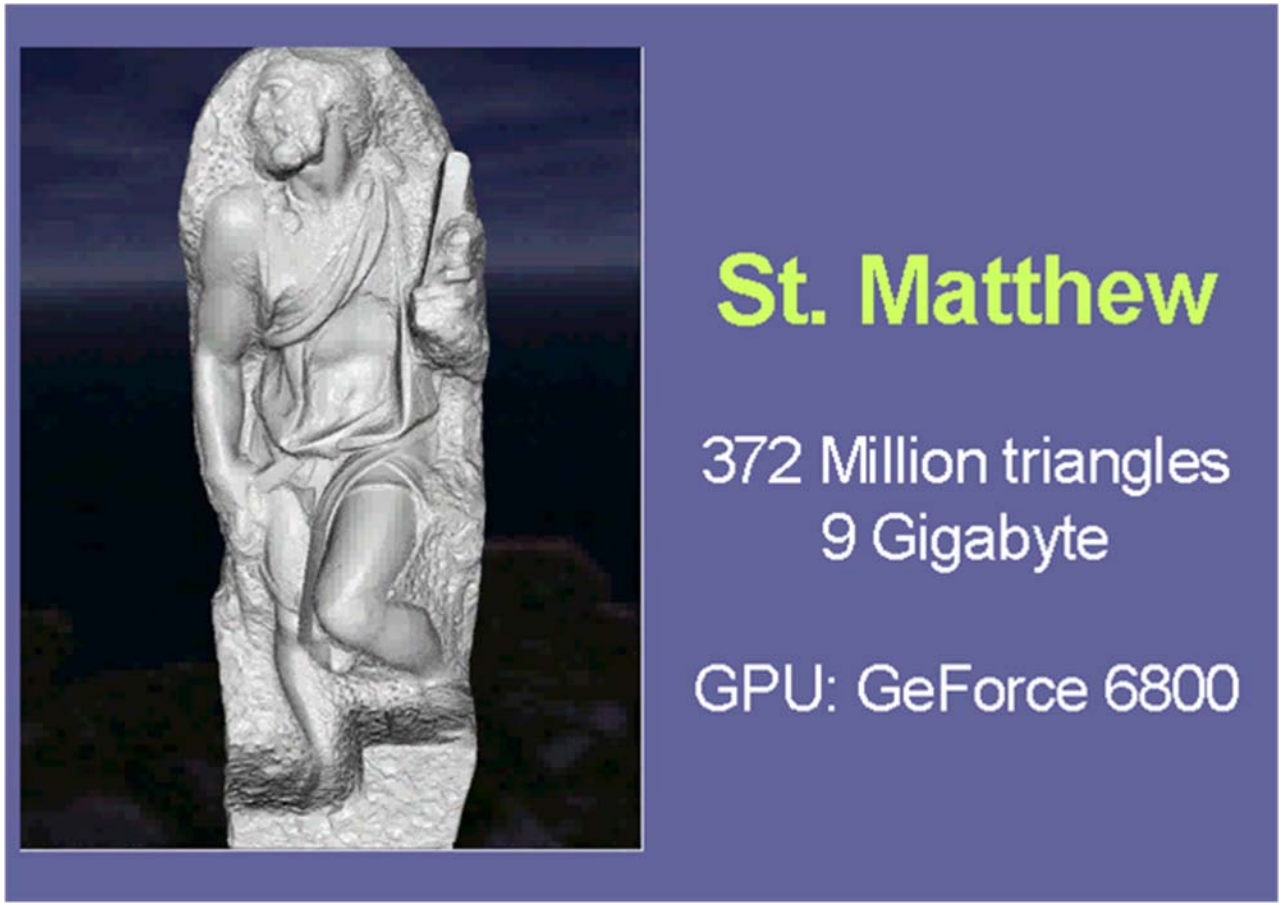
Main Approaches

- **Propose novel and practical cache-coherent metrics [Yoon et al. SIG 05, Yoon et al. VIS 06, Yoon et al. Euro 06]**
  - **Derive metrics given block-based caches**
  - **Propose efficient cache-coherent layout constructions**
  - **Apply to different applications**

Cache-Coherent Metrics

- **Measure the expected number of cache misses of a layout given block-based caches**
  - **Should correlate well with the observed number of cache misses**
- **Cache-aware metrics**
  - **Measure cache-coherence given known cache parameters (e.g., block size)**
- **Cache-oblivious metrics**
  - **Consider all possible cache parameters**

Run-time Captured Video – View-Dependent Rendering of St. Matthew Model



KAIST

Related Work

- Computation reordering
- Data layout optimization

Computational Reordering

- **Cache-aware** [Coleman and McKinley 95, Vitter 01, Sen et al. 02]
- **Cache-oblivious** [Frigo et al. 99, Arge et al. 04]
- **Streaming computations** [Isenburg et al. 05, 06]

Focus on specific problems such as sorting and linear algebra computations

Data Layout Optimization

- **Rendering sequences (e.g., triangle strips)**
  - **[Deering 95, Hoppe 99, Bogomjakov and Gotsman 02]**
- **Processing sequences**
  - **[Isenburg and Gumhold 03, Isenburg and Lindstrom 05]**

Assume that access pattern
globally follows the layout order!

Data Layout Optimization

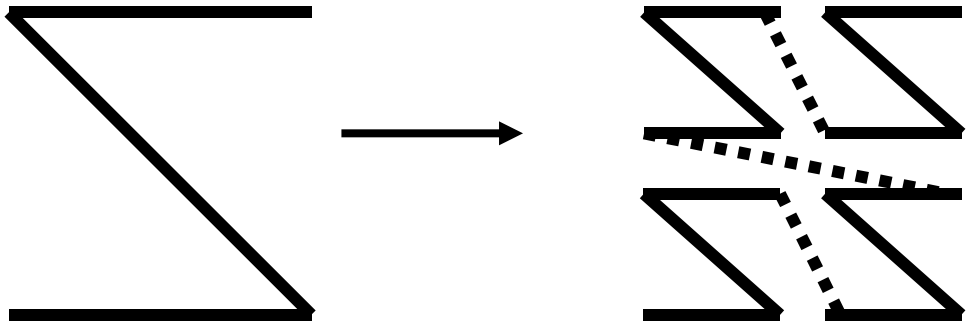
- **Graph and matrix layout**
  - **A survey [Diaz et al. 02]**
  - **Minimum linear arrangement (MLA)**
  - **Bandwidth, etc.**

Does not necessarily produce good layouts for block-based caches

Data Layout Optimization

- **Space-filling curves**
  - **[Sagan 94, Pascucci and Frank 01, Lindstrom and Pascucci 01, Gopi and Eppstein 04]**

Assume geometric regularity!



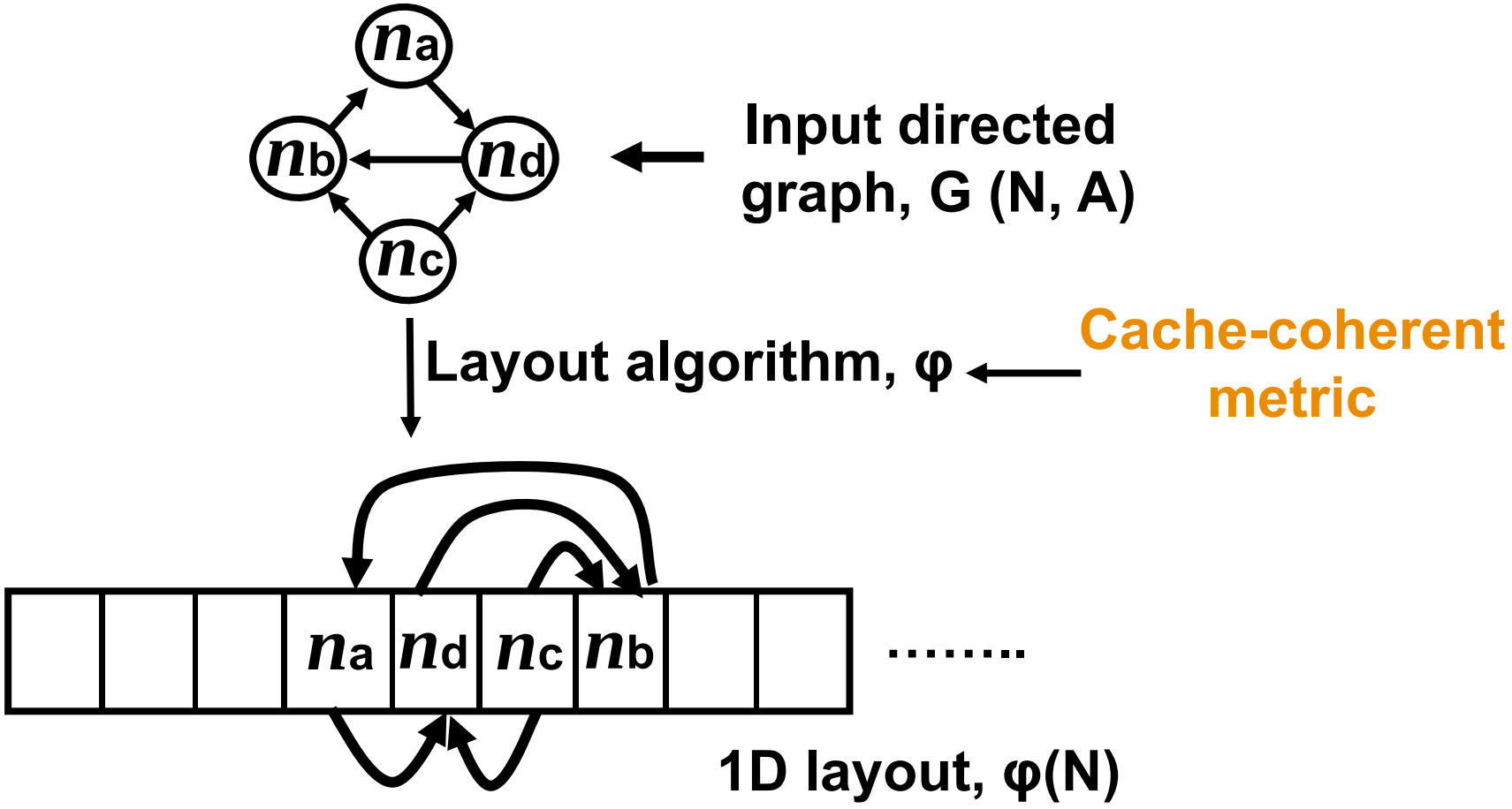
Outline

- Computation models
- Cache-aware and cache-oblivious metrics
- Results

Outline

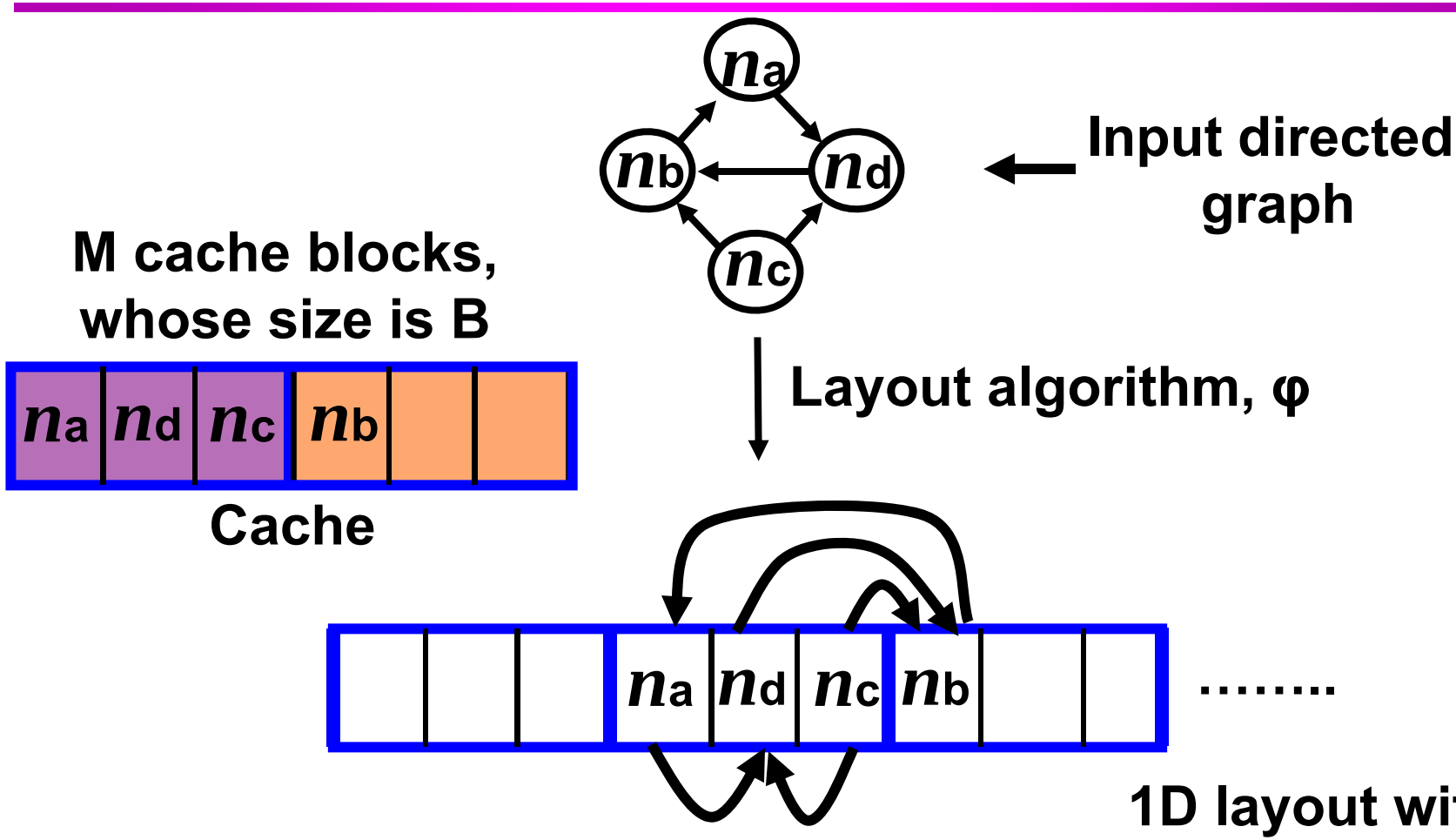
- Computation models
- Cache-aware and cache-oblivious metrics
- Results

General Framework of Layout Computation



KAIST

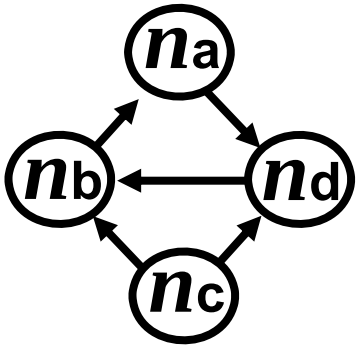
Two-Level I/O Model [Aggarwal and Vitter 88]



KAIST

Graph Representation

- **Directed graph, G = (N, A)**
  - **Represent access patterns between nodes**
- **Nodes, N**
  - **Data element**
  - **(e.g., mesh vertex or mesh triangle)**



- **Directed arcs, A**
  - **Connects two nodes if they are accessed sequentially**

KAIST

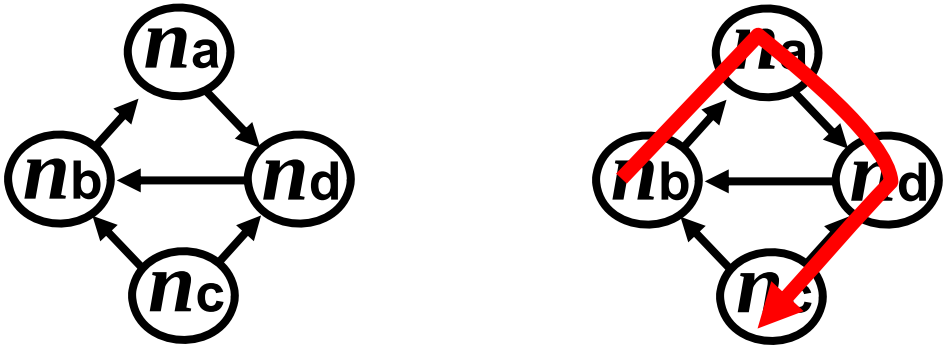
Weights of Nodes and Arcs

- Indicate probabilities that each element will be accessed
- Computed in an equilibrium status during infinite random walks
  - Assume that applications infinitely access the data according to the input graph
  - Correspond to eigen-values of the probability transition matrix

Problem Statement

- **Vertex layout of G = (N, A)**
  - **One-to-one mapping of vertices to indices in the 1D layout**

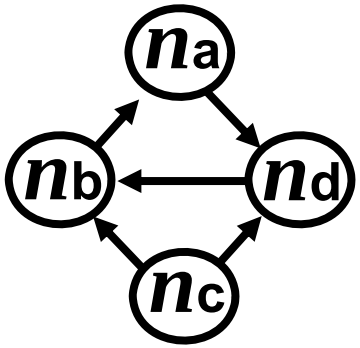
$$\varphi : N \rightarrow \{1, \ldots, |N|\}$$



- **Compute a φ that minimizes the expected number of cache misses**

Cache-Coherence of a Layout given Block-Based Caches

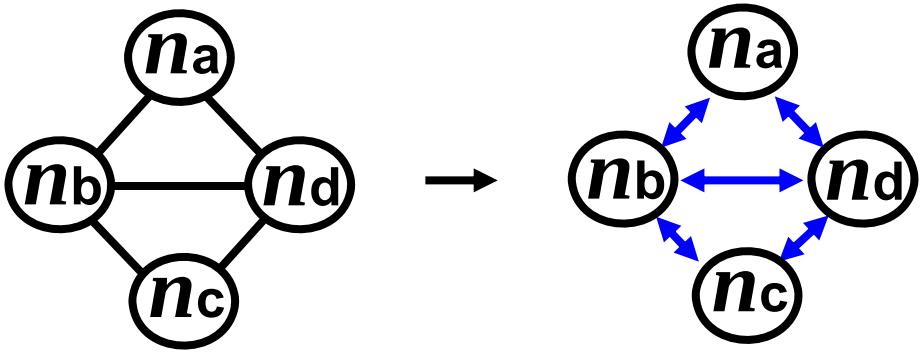
- Expected number of cache misses of a layout
  - Probability accessing a node from another node by traversing an arc
  - Conditional probability that we will have a cache miss given the above access pattern



KAIST

Specialization to Meshes

- **Expected number of cache misses of a layout**
  - **Probability accessing a node from another node by traversing an arc = constant**
  - **Conditional probability that we will have a cache miss given the above access pattern**



An input mesh

Implicitly derived graph

1. Two opposite directed arcs
2. Uniform distribution to access adjacent nodes given a node

KAIST

Outline

- Computation models
- Cache-aware and cache-oblivious metrics
- Results

Four Different Cases

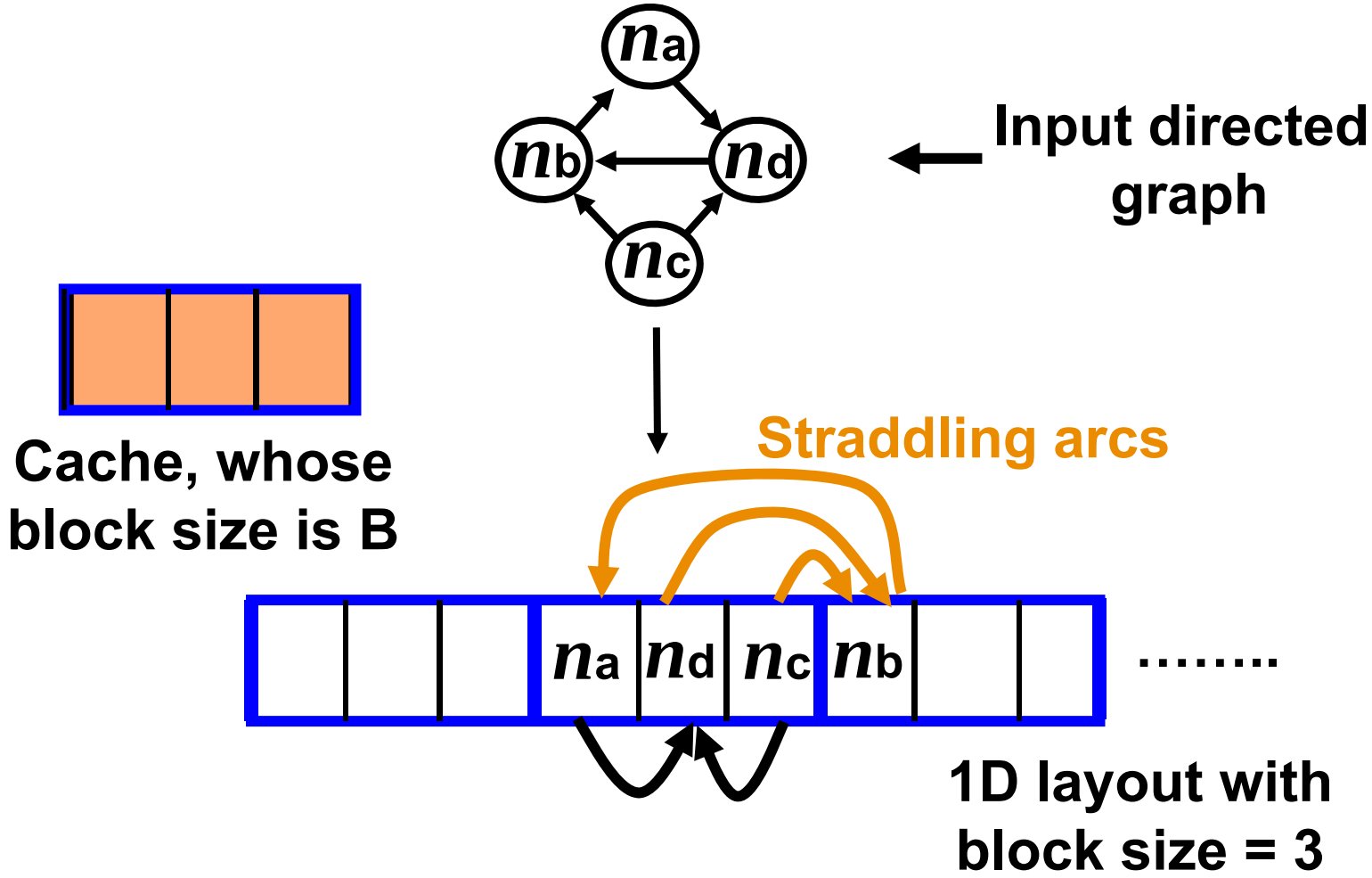
Cache-aware case single cache block, M=1

Cache-oblivious case single cache block, M=1

Cache-aware case multiple cache blocks, M>1

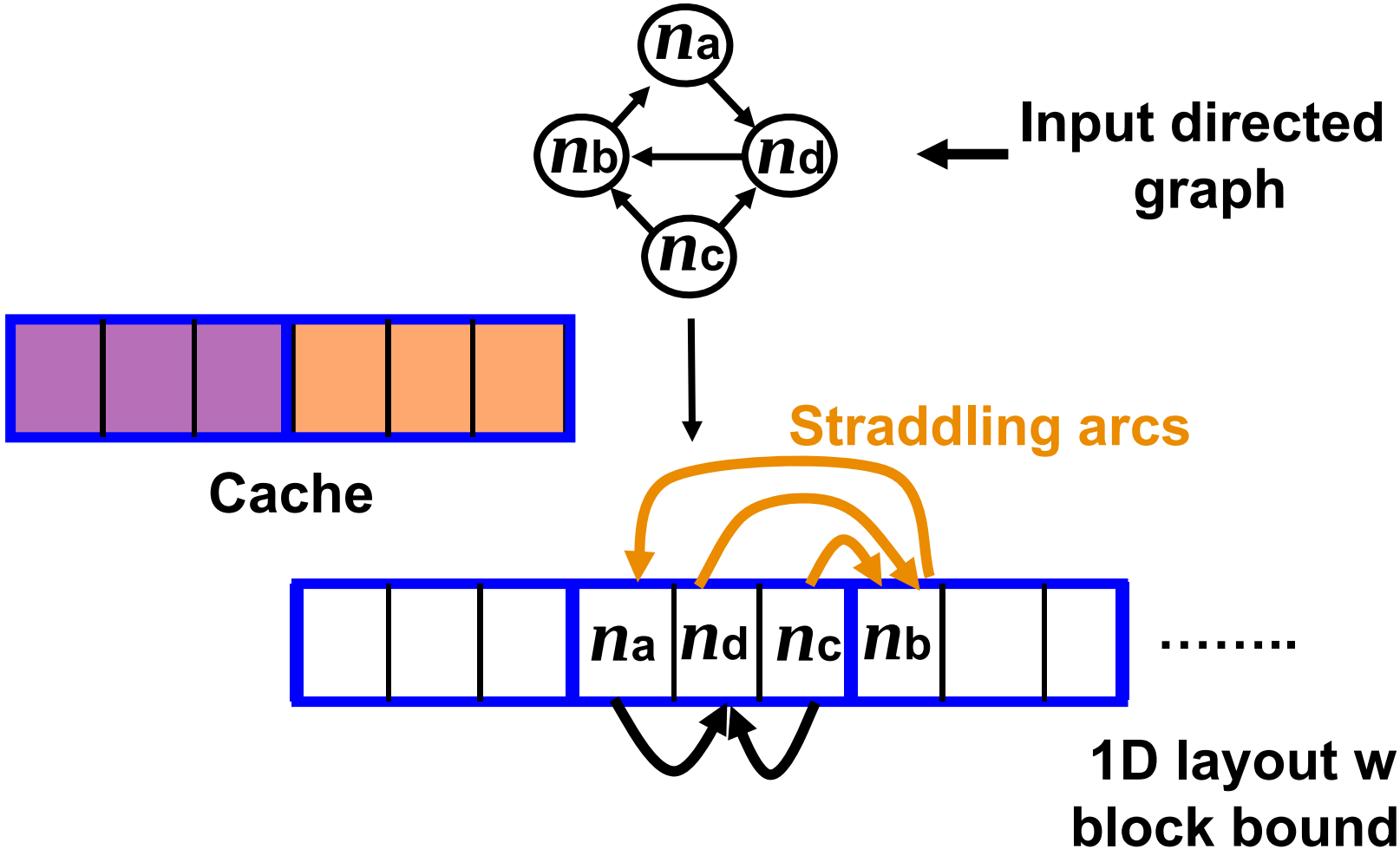
Cache-oblivious case multiple cache blocks, M>1

Cache-Aware: Single Cache Block, M=1



KAIST

Cache-Aware: Multiple Cache Blocks, M>1



Final Cache-Aware Metric

- **Counts the number of straddling arcs of the layout given a block size B**

$$\frac{1}{|A|}\sum_{(i,j)\in A} S(|\varphi^B(i)-\varphi^B(j)|)$$

where $\varphi^B(i)$ **: block index containing the node, i**

$S(x)$ **: Unit step function, 1 if x > 0**

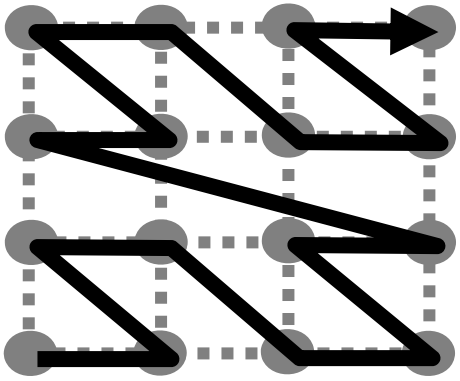
0 otherwise.

High Accuracy of Cache-Aware Metric

Tested block size = 4KB

Linear correlation [-1, 1]	Observed number of cache misses	
	With 5 cache blocks	With 25 cache blocks
Cache-aware metric	0.97	0.97

Z-curve on a uniform grid



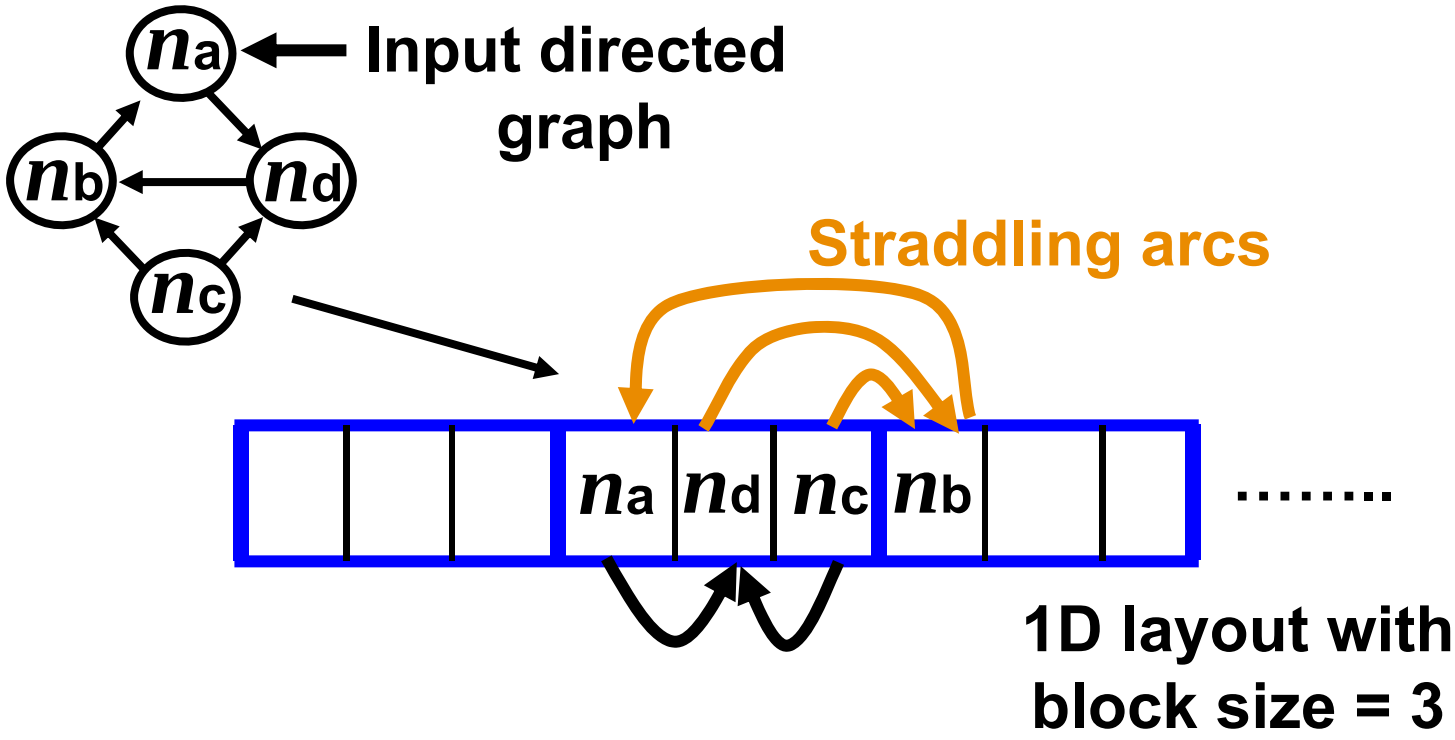
Tested layouts:

Z-curve, Hilbert curve, H-order, minimum linear arrangement layout, βΩ-layout, geometric CO layout, (bi or uni) row-by-row, (bi or uni) diagnoal layouts

KAIST

Cache-Aware Layouts

- **Optimized with cache-aware metric given a block size B**
  - **Computed from the graph partitioning**



KAIST

Four Different Cases

Cache-aware case single cache block, M=1

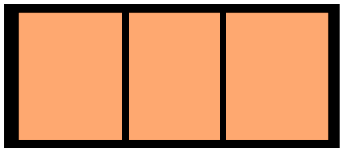
Cache-oblivious case single cache block, M=1

Cache-aware case multiple cache blocks, M>1

Cache-oblivious case multiple cache blocks, M>1

Cache-Oblivious: Single Cache Block, M=1

Does not assume a particular block size:
Then, what are good representatives for block sizes?



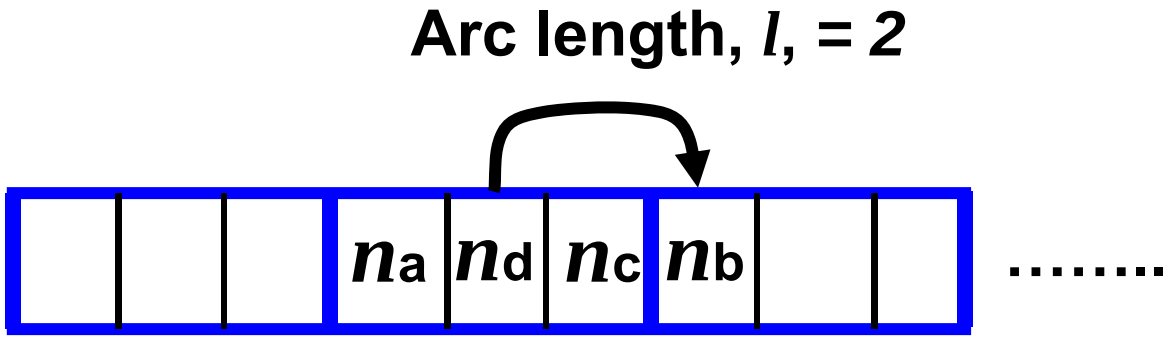
Cache

Two Possible Block Size Progressions

- Arithmetic progression
  - 1, 2, 3, 4, …
- Geometric progression
  - 2^0, 2^1, 2^2, 2^3, …
  - Well reflects current caching architectures
  - E.g., L1: 32B, L2: 64B, Page: 4KB, etc.

Probability that an Arc is a Straddling Arc

Is an arc straddling given a block size? → Computed as a probability as a function of arc length, l



KAIST

Two Cache-Oblivious Metrics

- **Arithmetic cache-oblivious metric,** $COM_a(\varphi)$

MLA metric, Arithmetic mean

$$\frac{1}{|A|}\sum_{(i,j)\in A} l_{ij}$$

Arc length of arc (i, j)

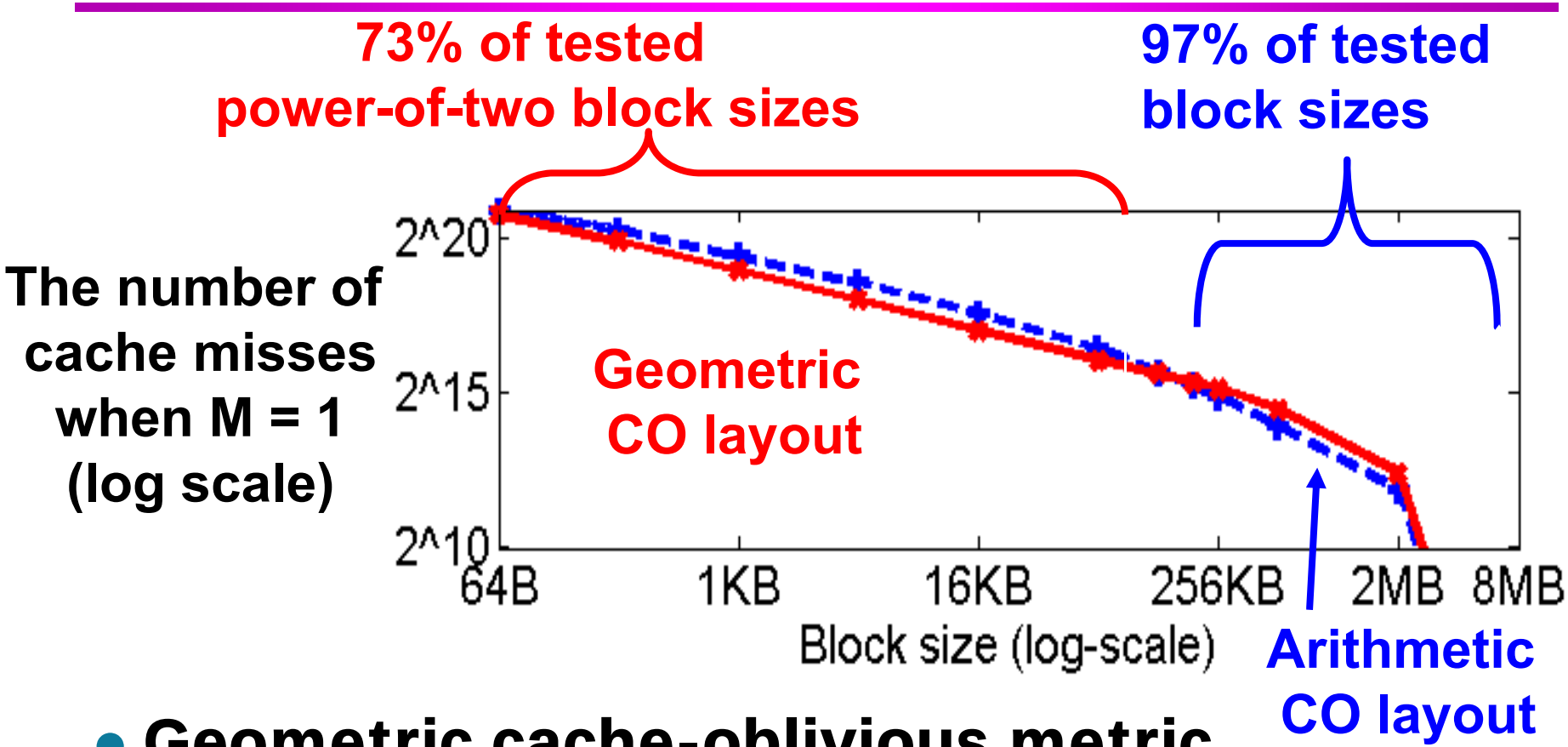
- **Geometric cache-oblivious metric,** $COM_g(\varphi)$

$$\frac{1}{|A|}\sum_{(i,j)\in A}\log(l_{ij}) = \log\left(\left(\prod_{(i,j)\in A} l_{ij}\right)^{\frac{1}{|A|}}\right)$$

Geometric mean of arc lengths

KAIST

Validation for Cache-Oblivious (CO) Metrics



- **Geometric cache-oblivious metric**
  - **Practical and useful**

KAIST

Correlations between Metrics and Observed Number of Cache Misses

Tested block size = 4KB

Linear correlation [-1, 1]	Observed number of cache misses	
	With 1 cache block	With 5 cache blocks
Geometric CO metric	0.98	0.81
Arithmetic CO metric	-0.19	-0.32

Tested with 10 different layouts on a uniform grid

Cache-Oblivious Layouts

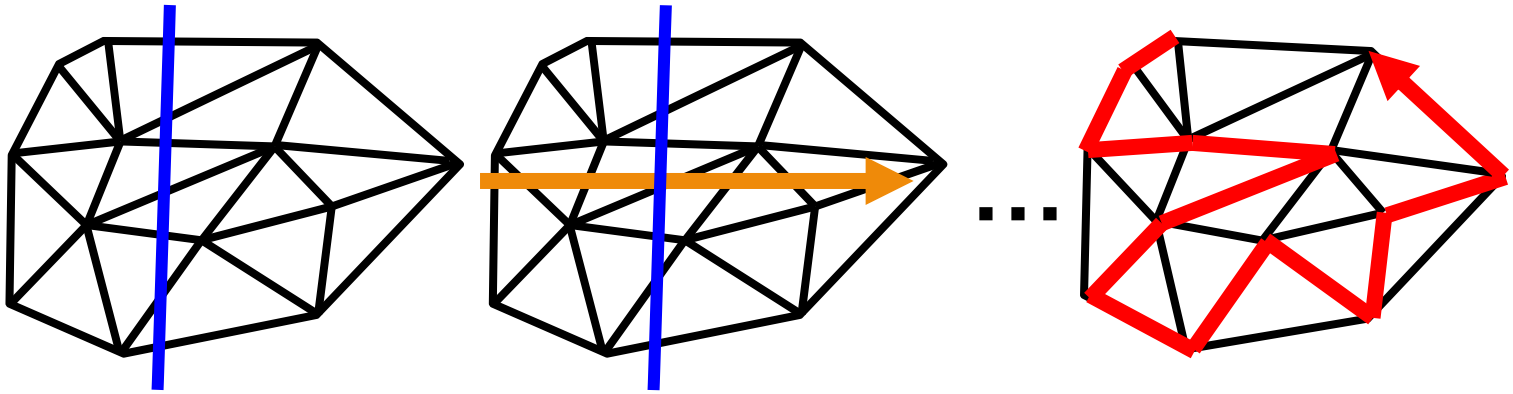
- Geometric cache-oblivious metric
  - Very efficient
  - Can be used in different layout optimization methods

Layout Computation with Geometric Cache-Oblivious Metric

- **Multi-level construction method**
  - **Partition an input mesh into k different sets**
  - **Layout partitions based on our metric**

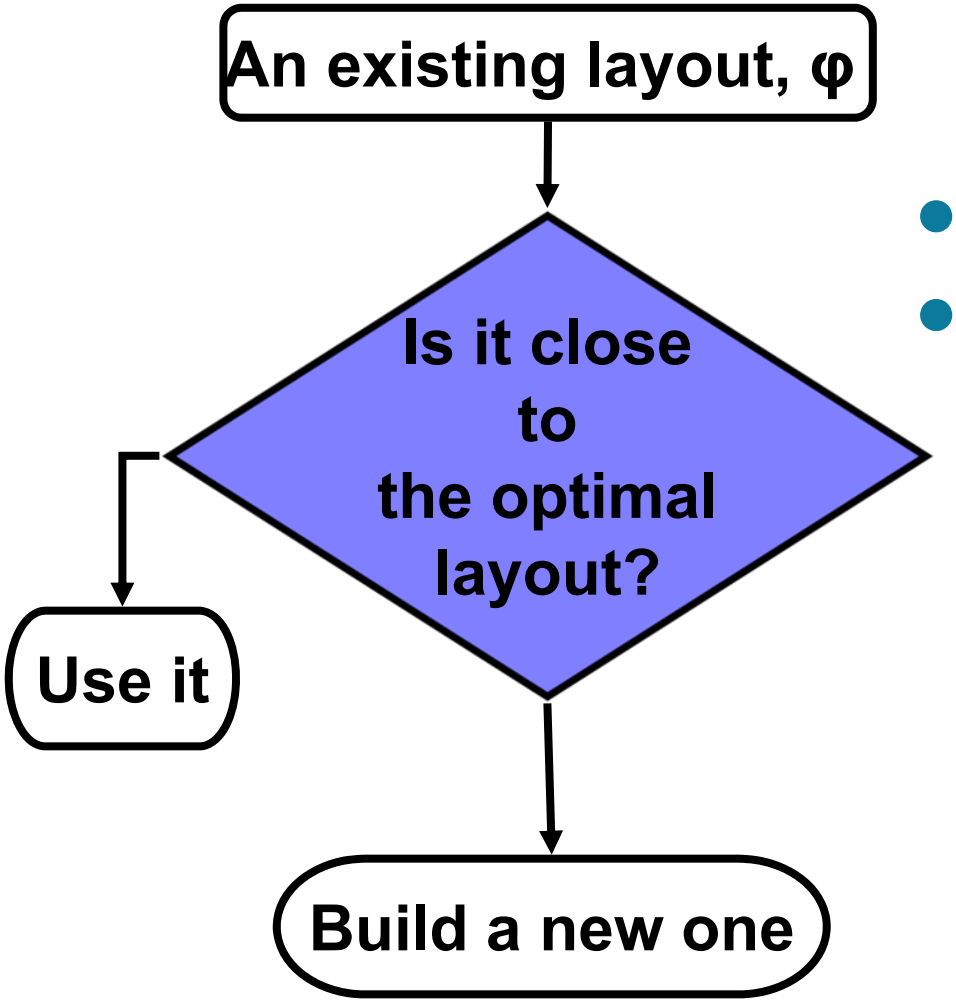
•Generalized layout method for unstructured meshes

1. Partition **2. Lay out**



KAIST

Evaluating Existing Layouts



- No known tight bound
- Compare against the best layout we can construct
  - Employ an efficient sampling method

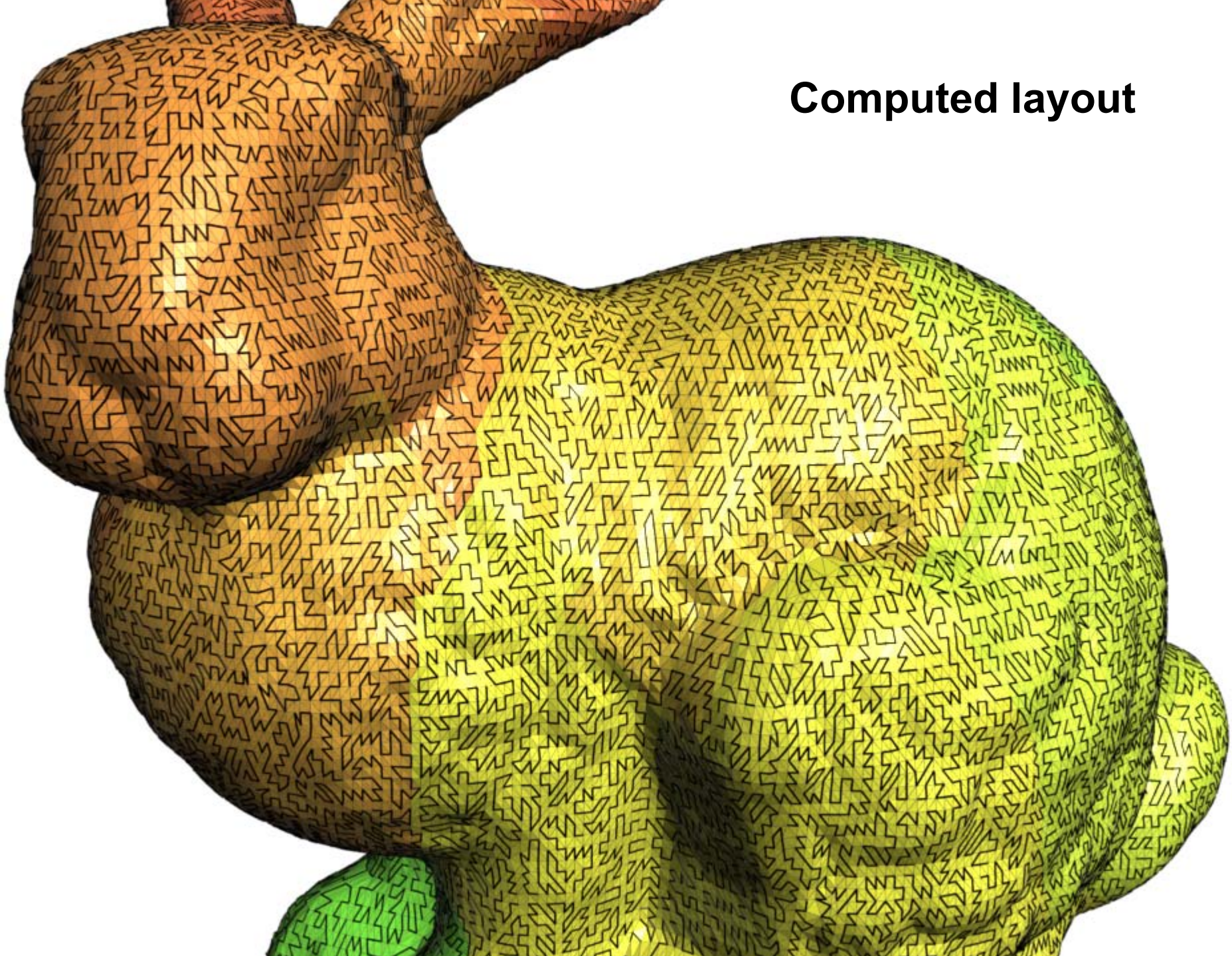
KAIST

Outline

- Computation models
- Cache-aware and cache-oblivious metrics
- **Results**

Layout Computation Time

- Process 70 million vertices per hour
  - Takes 2.6 hours to lay out St. Matthew model (372 million triangles)
  - 2.4GHz of Pentium 4 PC with 1 GB main memory

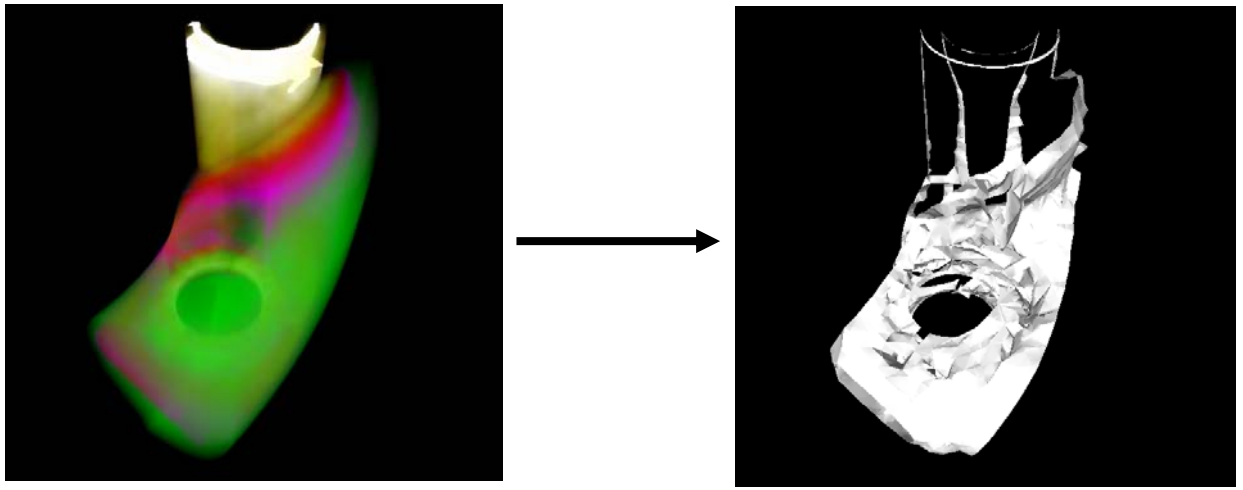

Computed layout

Applications

- Isosurface extraction
- View-dependent rendering
- Collision detection
- Ray tracing

Iso-Surface Extraction

Spx model
(140K vertices)



- Uses contour tree [van Kreveld et al. 97]
  - Runtime is dominated by the traversal of iso-surface
- Layout graph
  - Use an input tetrahedral mesh

High Correlation with Number of Cache Misses

Tested block size = 4KB

Linear correlation [-1, 1]	Observed number of cache misses	
	With 1 cache block	With 10K cache blocks
Geometric CO metric	0.99	0.98

Tested with 8 different layouts:
our geometric CO, our cache-aware, breadth-first (and depth-first) layouts, spectral [Juvan and Mohar 92], cache-oblivious mesh [Yoon et al. 05], Z-curve [Sagan 94], X-axis sorted layouts

KAIST

High Correlation with Runtime Performance

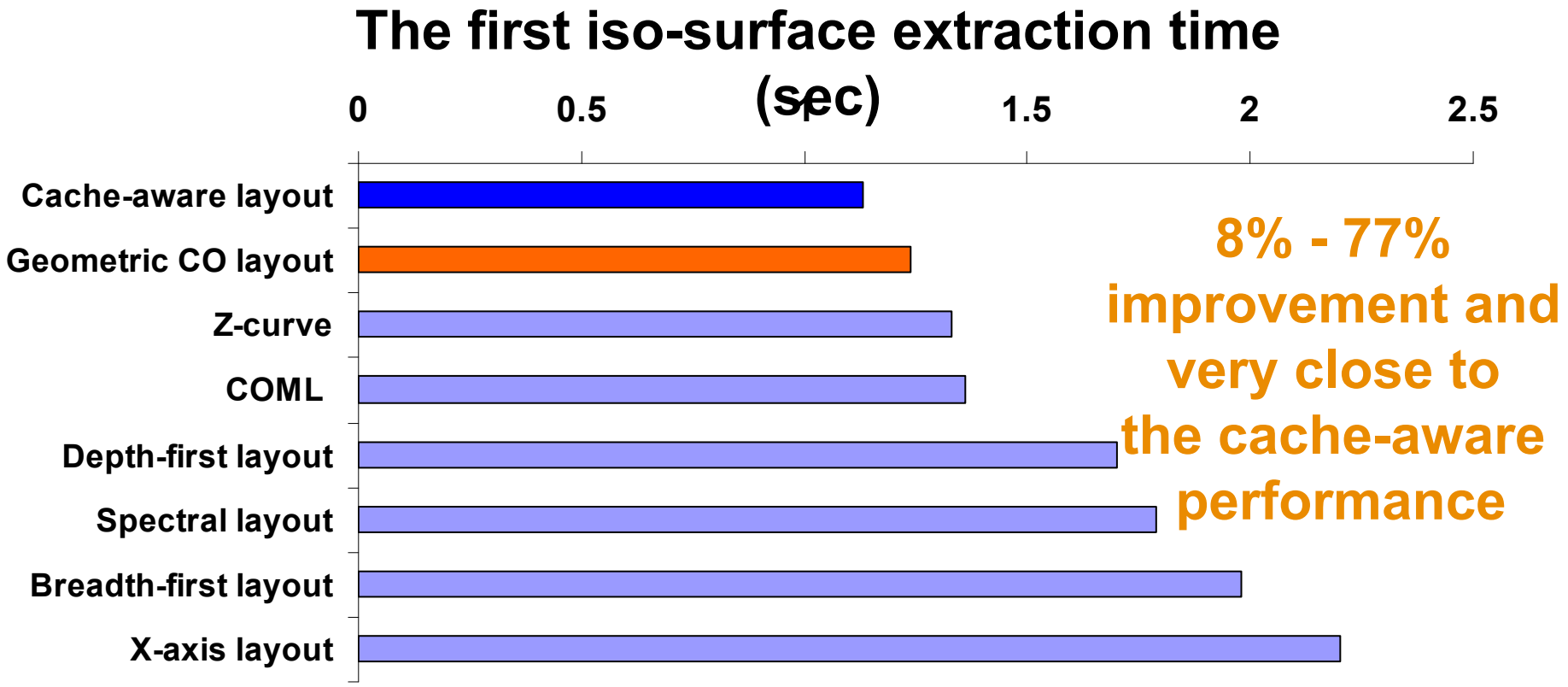
Disk I/O time is major bottleneck

Memory access time is major bottleneck

Linear correlation [-1, 1]	First iso-surface extraction time	Second iso-surface extraction time
Geometric CO metric	0.94	0.94

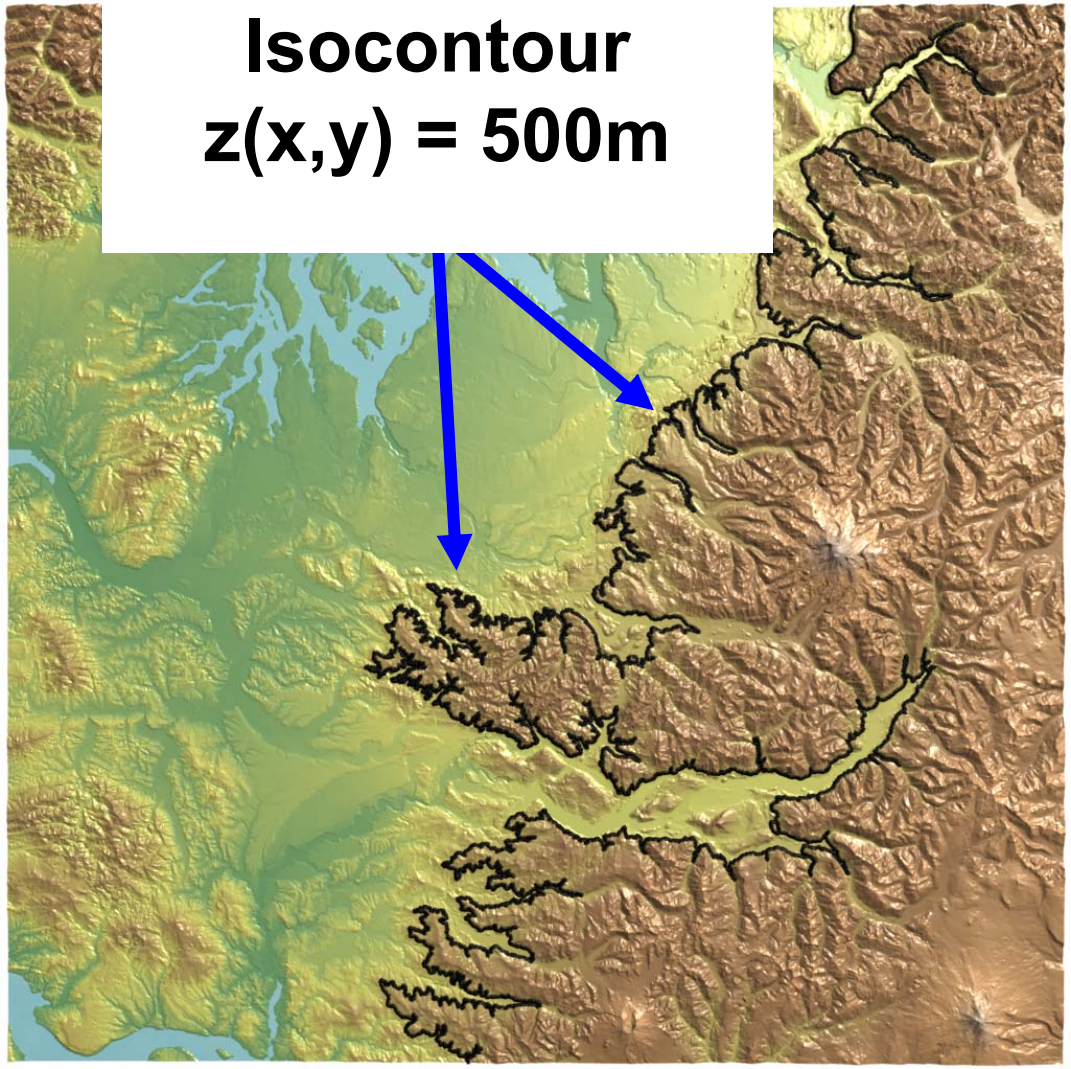
KAIST

Comparison with Other Layouts



KAIST

Isocontour Extraction – Puget Sound Model, 134M Triangles

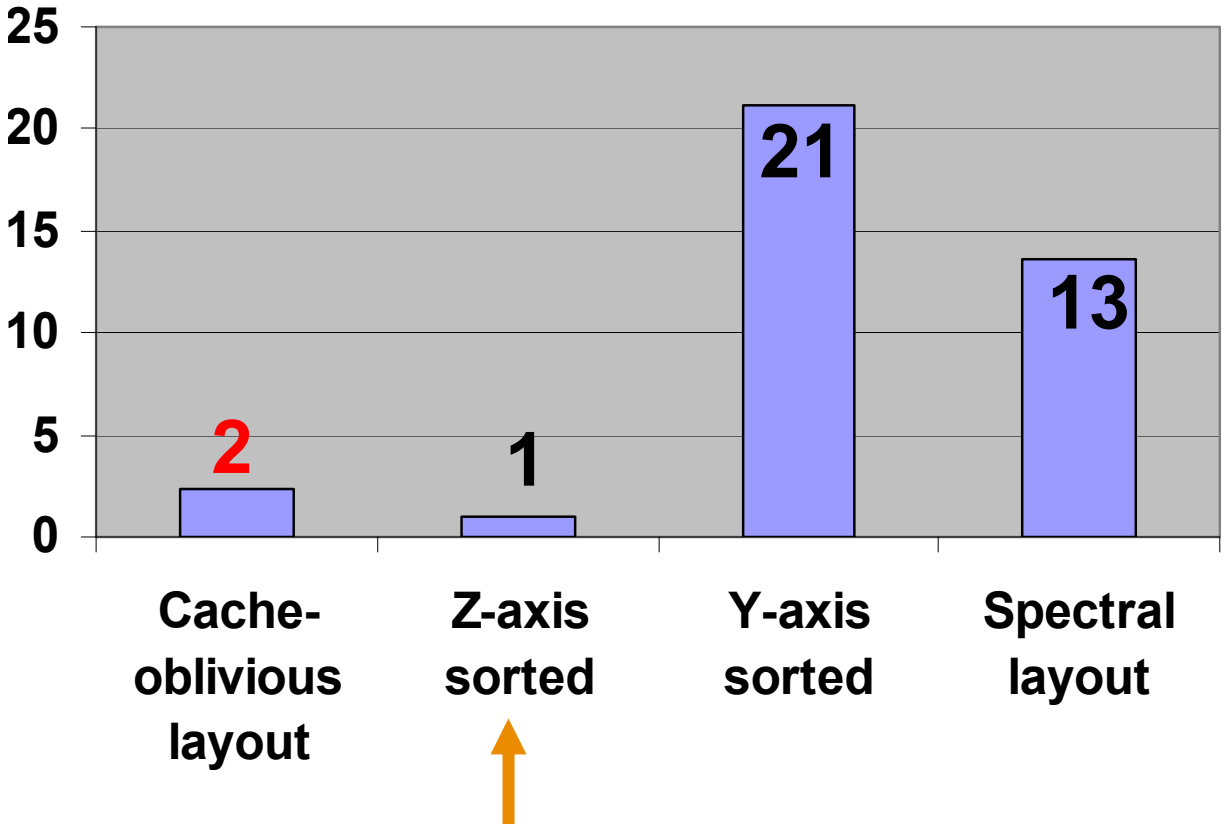


KAIST

Comparison – First Extraction of Z(x,y) = 500m

Disk access time is bottleneck

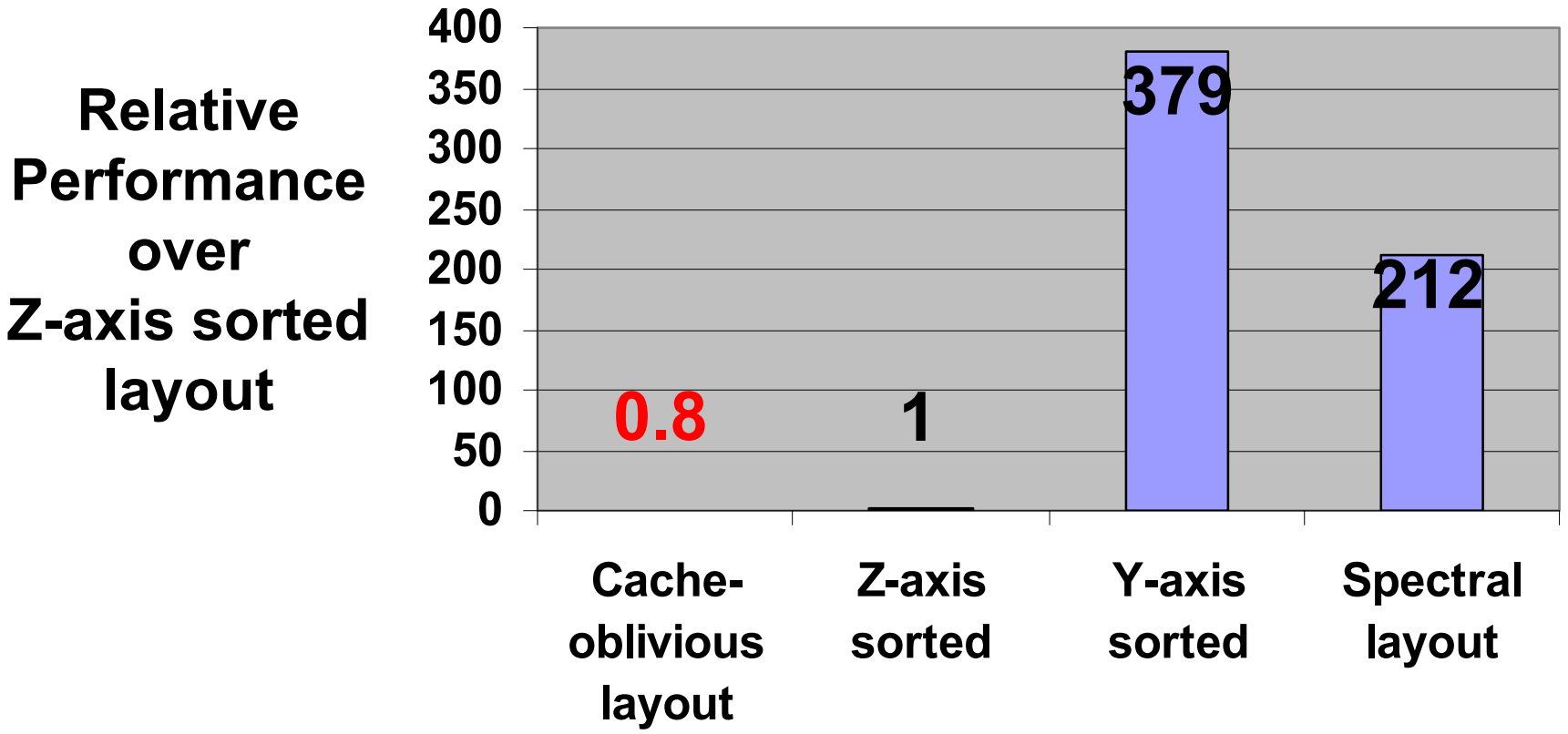
Relative Performance over Z-axis sorted layout



Nearly optimized for particular isocontour

KAIST

Comparison – Second Extraction of Z(x,y) = 500m

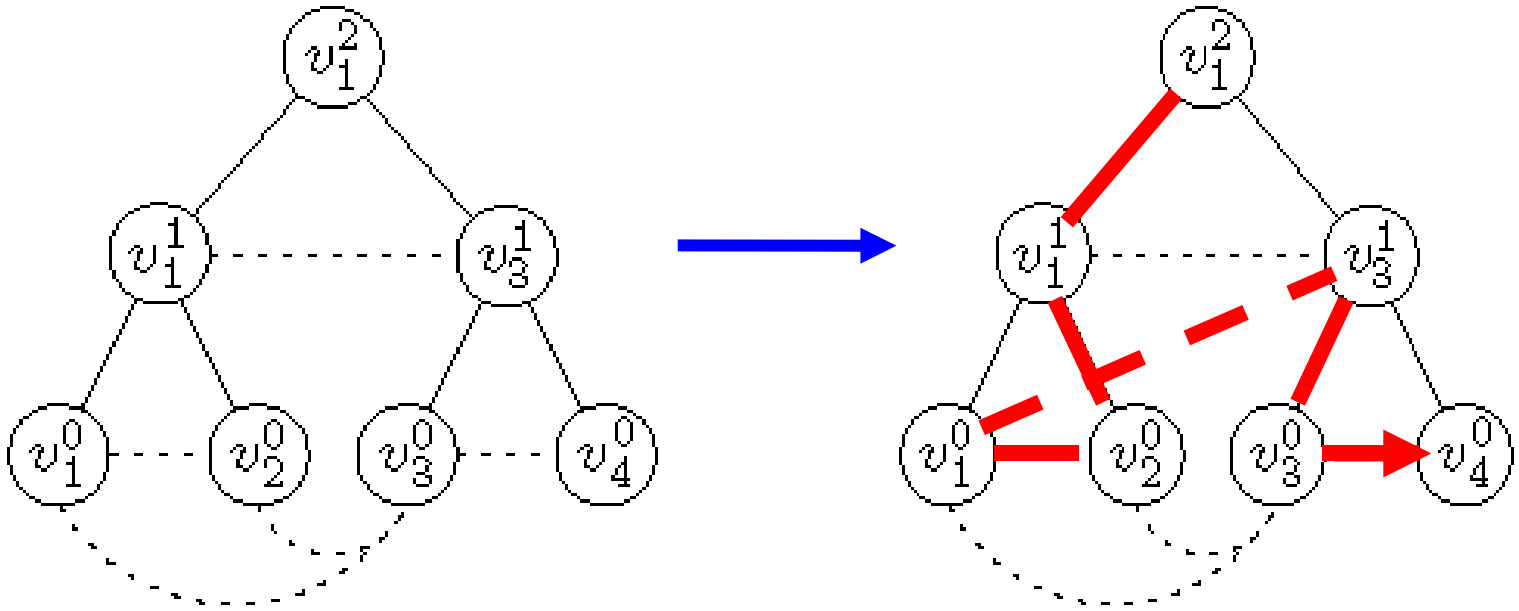


Memory and L1/L2 cache access times are bottleneck

KAIST

View-Dependent Rendering

- **Layout vertices and triangles of CHPM [Yoon et al. VIS 04]**
  - **Reduce misses of GPU vertex cache**



KAIST

View-Dependent Rendering

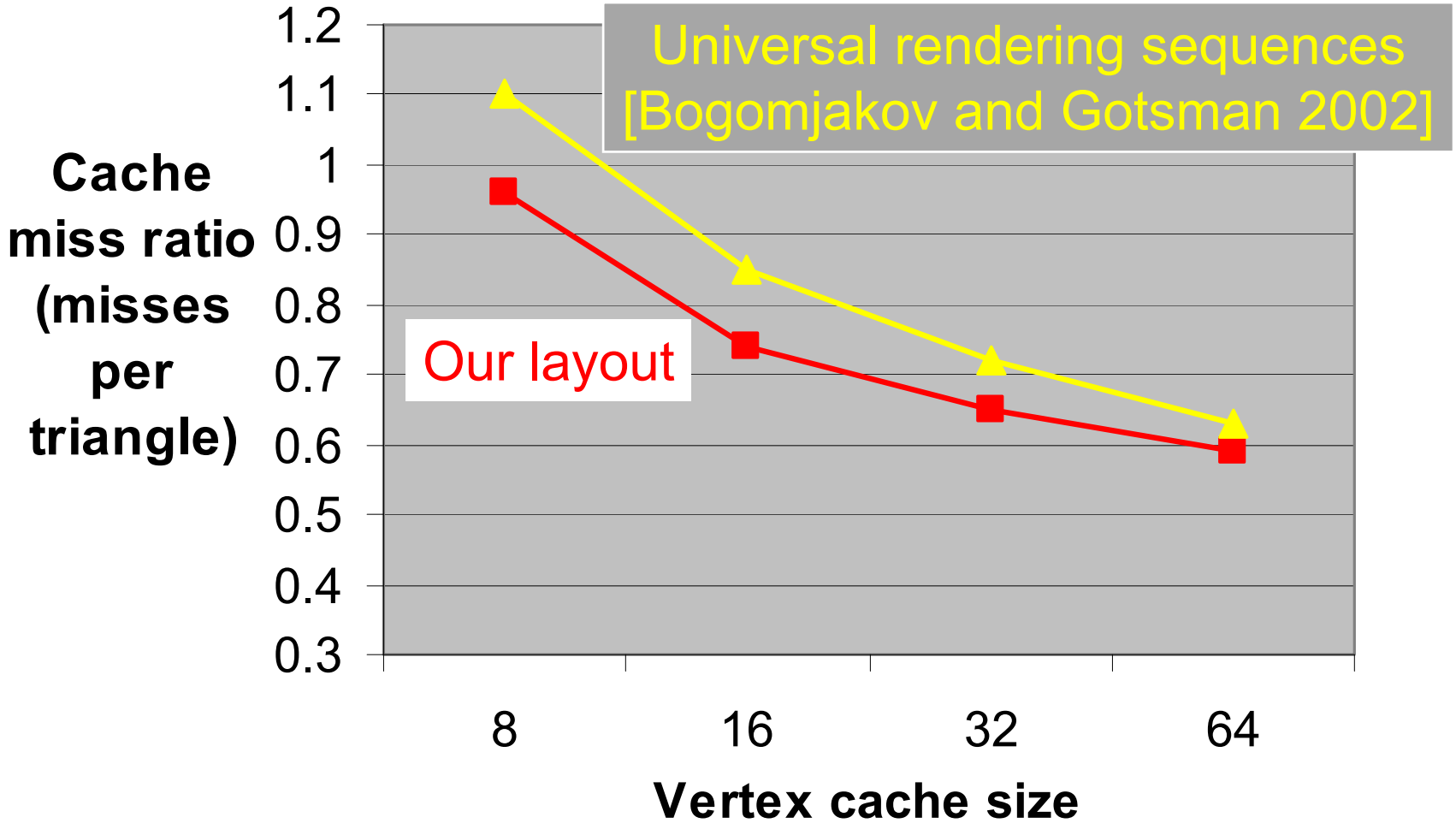
Peak performance: 145 M tri / s on GeForce 6800 Ultra

Models	# of Tri.	Our layout	Simplification layout
St. Matthew	372M	106 M/s	23 M/s
Isosurface	100M	90 M/s	20 M/s
Double Eagle Tanker	82M	47 M/s	22 M/s

4.5X

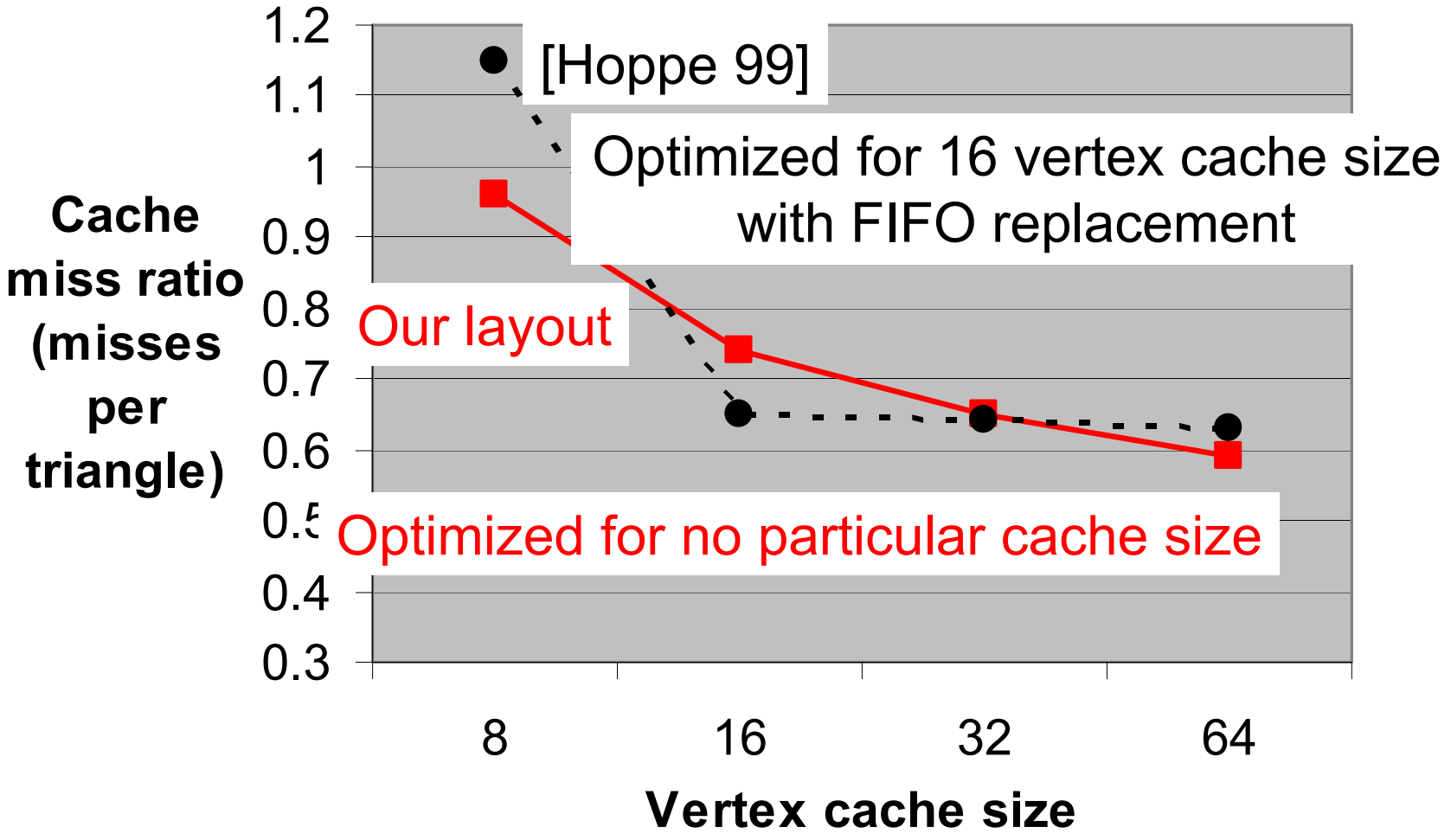
2.1X

Comparison with Other Rendering Sequences



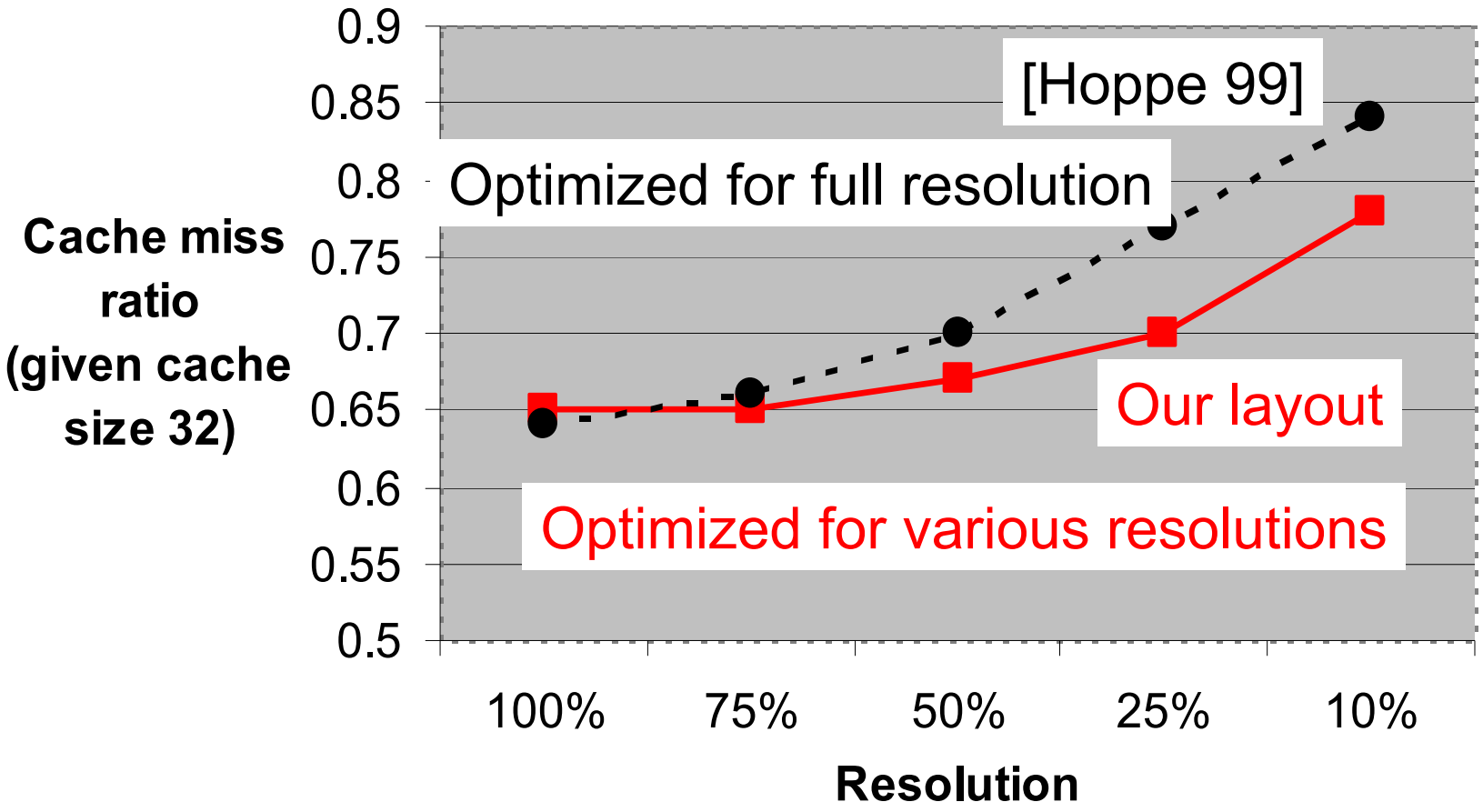
KAIST

Comparison with Other Rendering Sequences



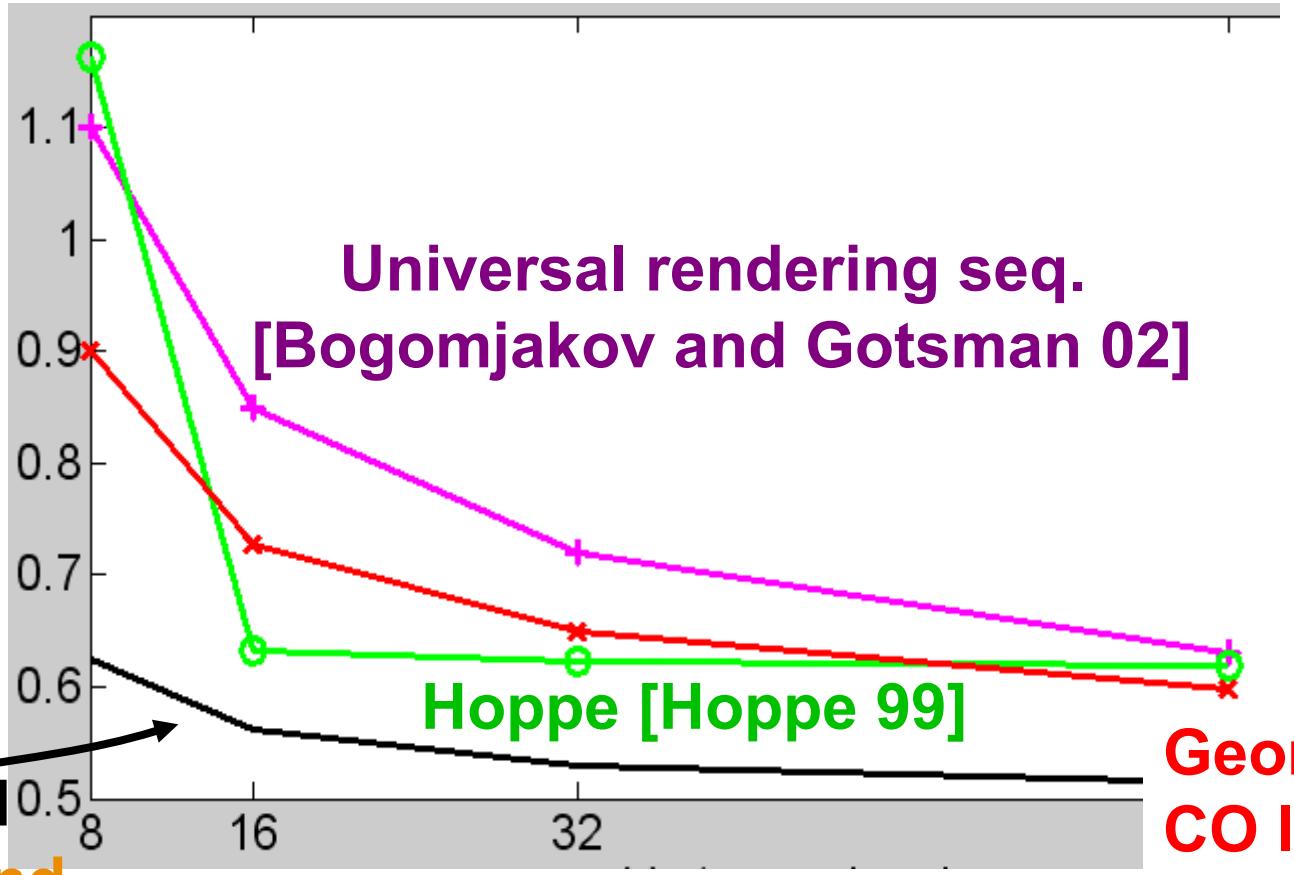
KAIST

Performance during View-Dependent Rendering



KAIST

Cache Miss Ratio on Bunny Model



KAIST

Cache Miss Ratio on Power Plant Model

KAIST

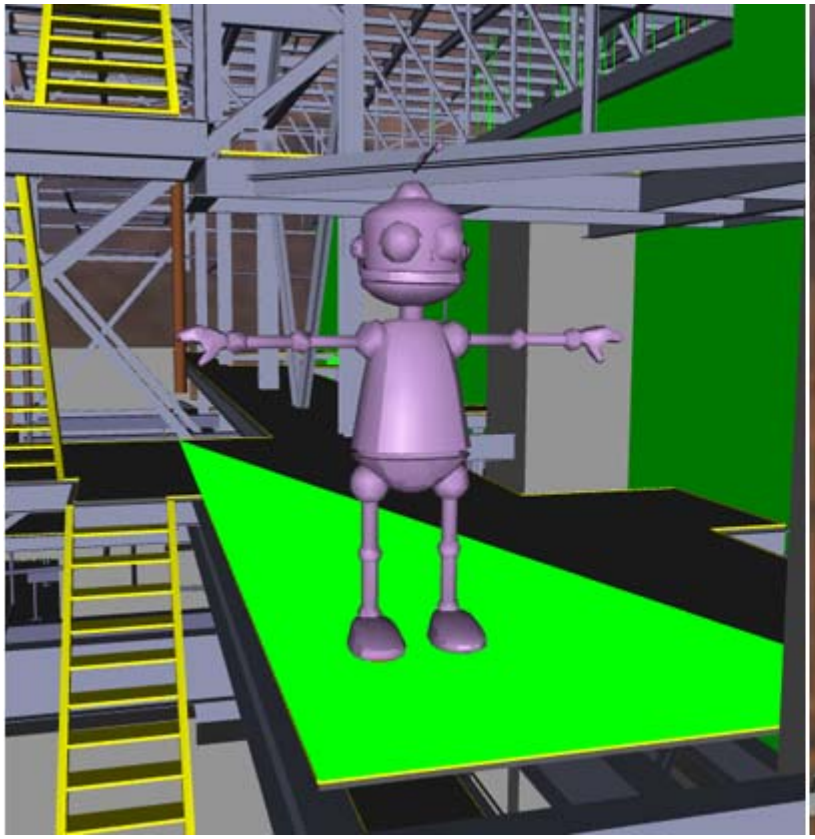
Collision Detection

- **Use oriented bounding box (OBB) [Gottschalk et al. 96]**
  - **Breadth-first tree traversal**

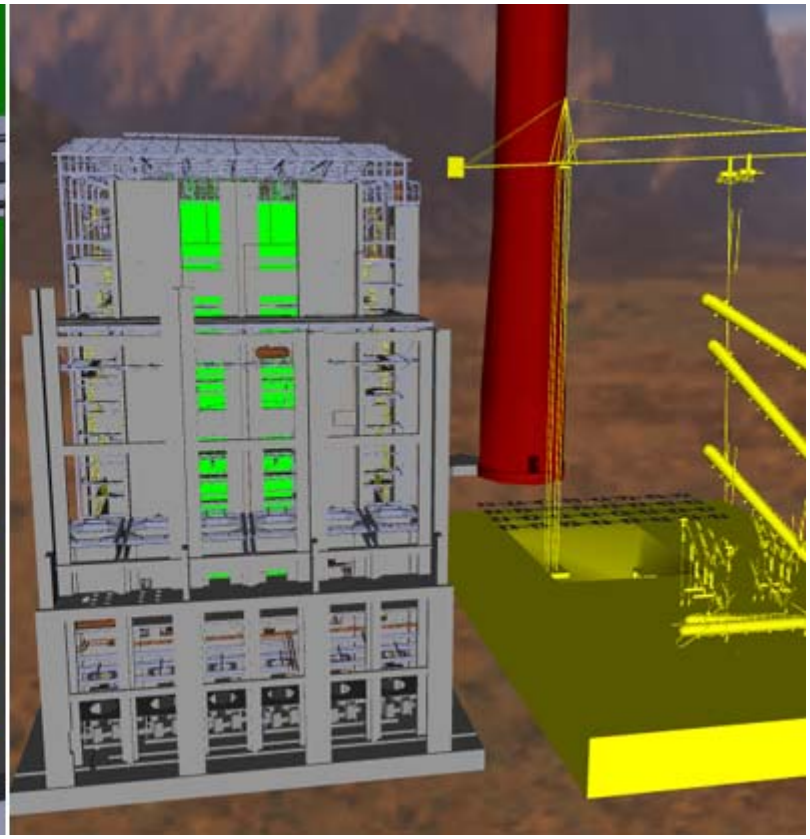
- **Use an input graph representing well the runtime access pattern on the hierarchy**

Collision Detection – Robot and Power Plant Models

20k triangles



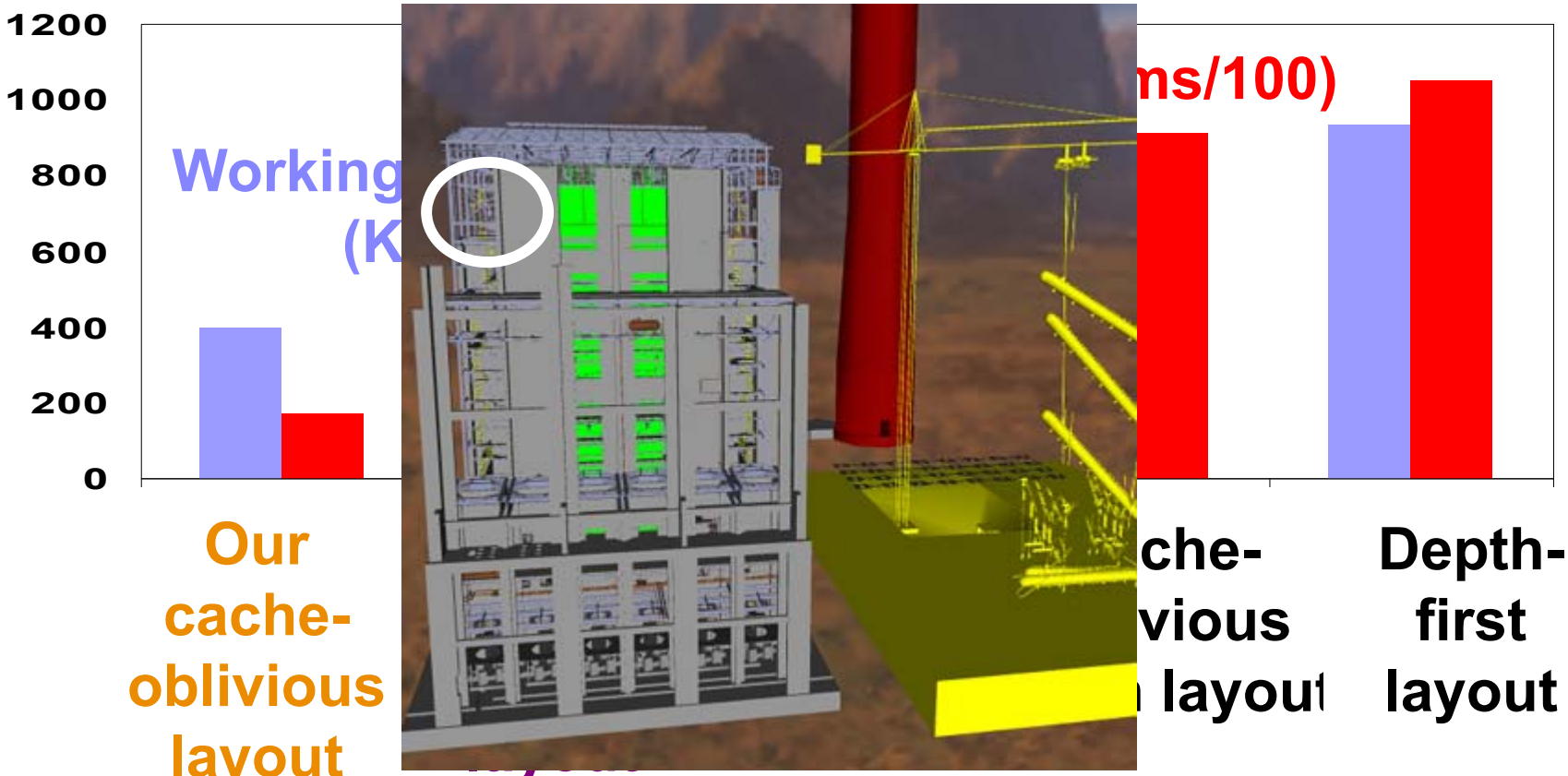
1M triangles



KAIST

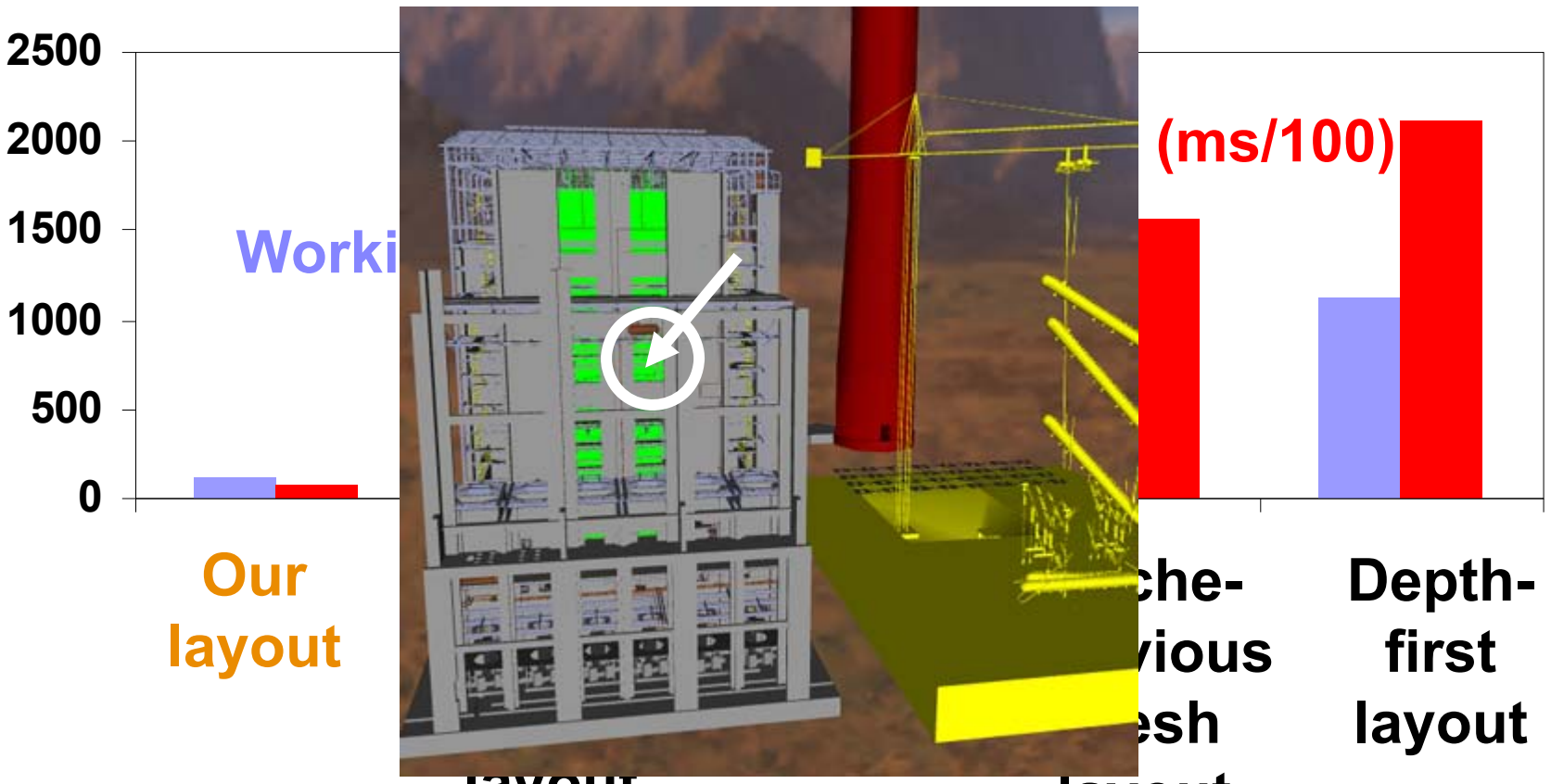
Collision Detection – Performance Comparison I

41% ~ 500% performance improvement



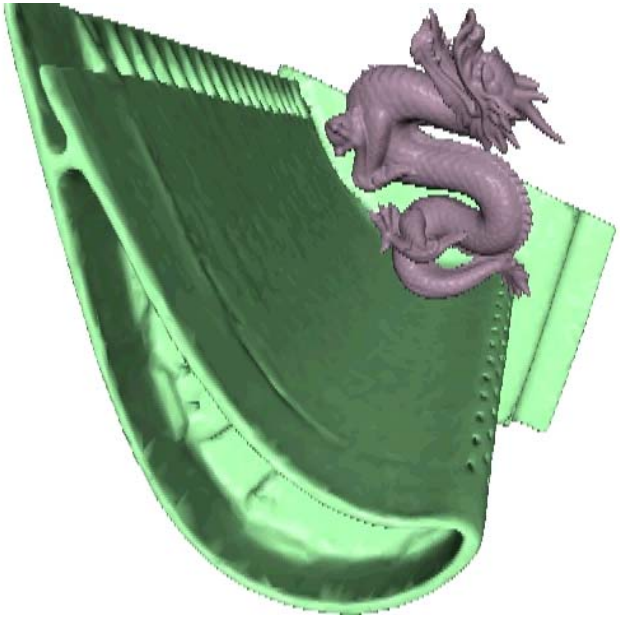
Collision Detection – Performance Comparison II

35% ~ 2600% performance improvement



Cache-Oblivious Layout vs Cache-Aware Layout

- Cache-aware layouts
  - Take advantage of block size information (4KB)
- Minor performance degradation
  - 8% compared to cache-aware layouts



Ray Tracing

- Use kd-tree [Wald 04]
  - Depth-first tree traversal

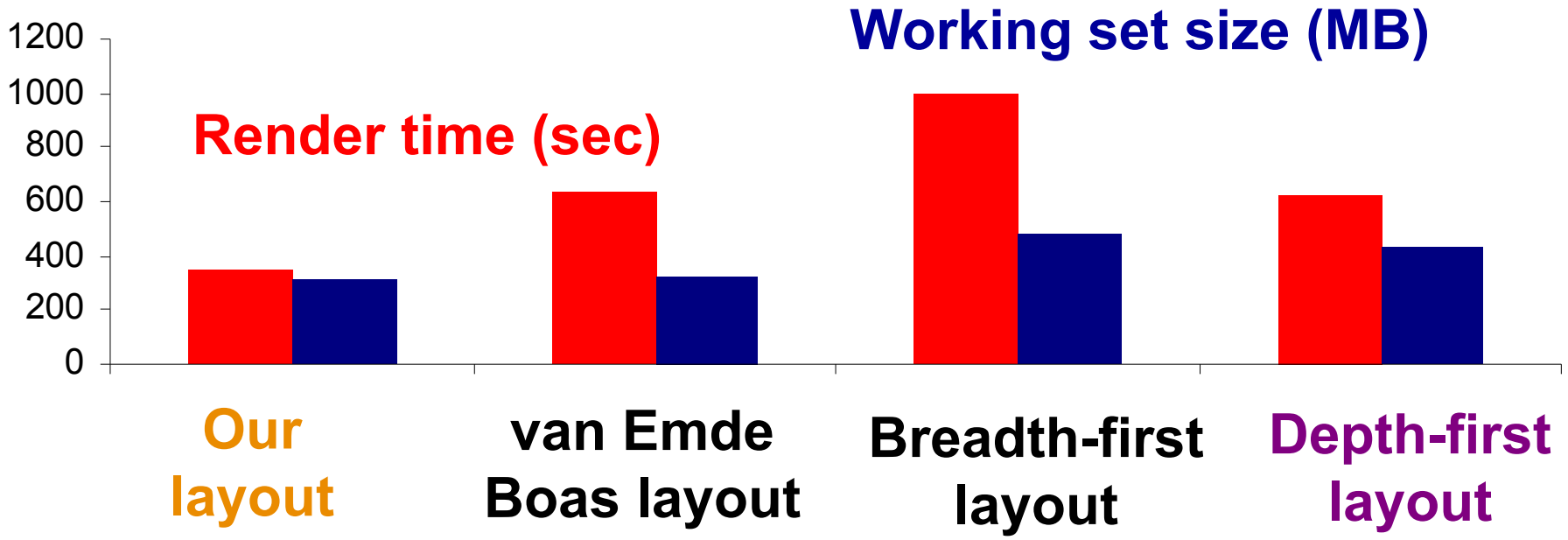
Ray Tracing – Lucy Model

28 million triangles **Pentium IV with 1GB**

KAIST

Ray Tracing – Performance Comparison

77% ~ 180% performance improvement



KAIST

Advantages

- General
  - Applicable to all kinds of polygonal models
  - Works well for various applications
- Cache-oblivious
  - Can have benefit from CPU/GPU cache to memory and disk
- Robust performance improvement
- No modification of runtime application
  - Only layout computation

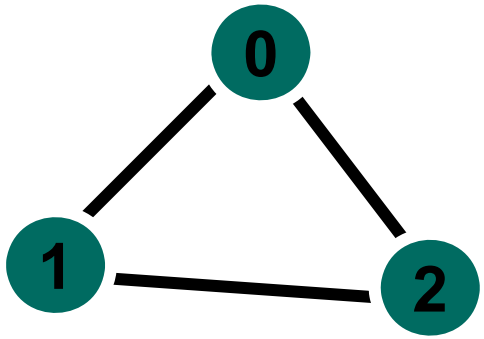
OpenCCL: Cache-Coherent Layouts of Graphs and Meshes

- **Source codes for computing a cache-coherent layout**
  - **Easy to use**
  - **Google "Cache Coherent Layouts"**

```
CLayoutGraph Graph (NumVertex);

Graph.AddEdge (0, 1);
Graph.AddEdge (0, 2);
Graph.AddEdge (1, 2);

        int Order [NumVertex];
Graph.ComputeOrdering (Order);
```



KAIST

Summary

- **Novel cache-aware and cache-oblivious metrics to evaluate layouts**
  - **Derived metrics based on two-level I/O model**
  - **Improved the performance of applications without modifying codes**

Ongoing and Future Work

- Derive a lower bound on our geometric cache-oblivious metric
- Employ mesh compression to further reduce disk I/O accesses
- Investigate efficient layout method for deforming/dynamic models
- Apply to non-graphics applications
  - e.g., shortest path or other graph computations
- Apply to other representations such as R-tree

At the Next Class

- **Will discuss data compression**