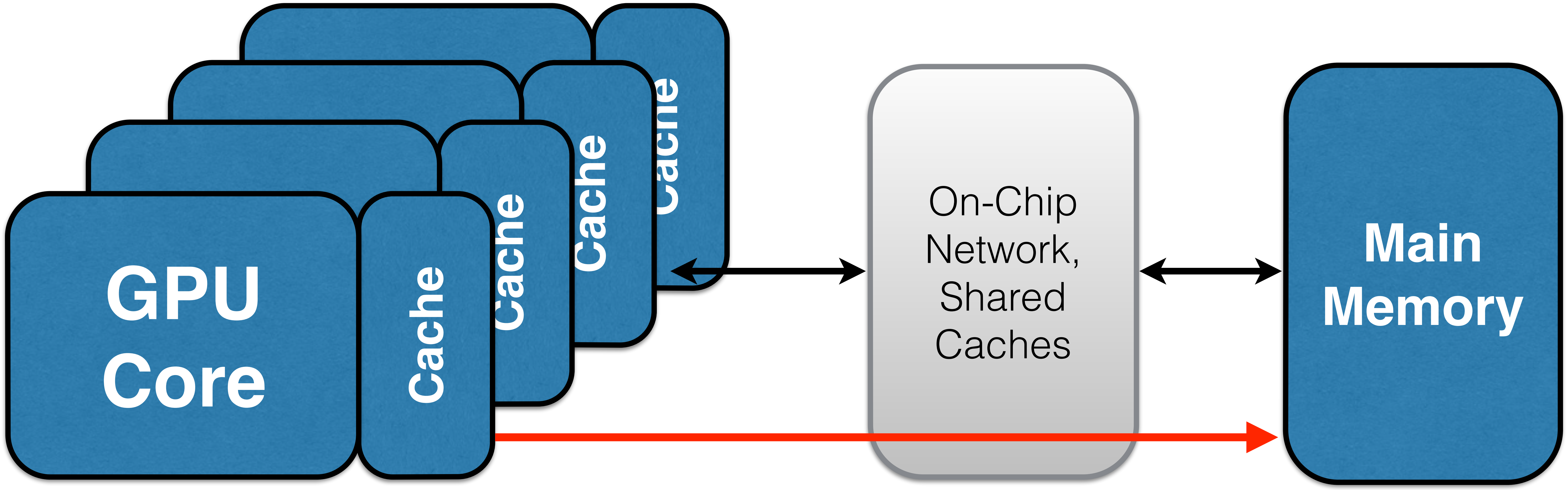
Texture Cache Approximation on GPUs

Mark Sutherland
Joshua San Miguel
Natalie Enright Jerger

{suther68,enright}@ece.utoronto.ca, joshua.sanmiguel@mail.utoronto.ca

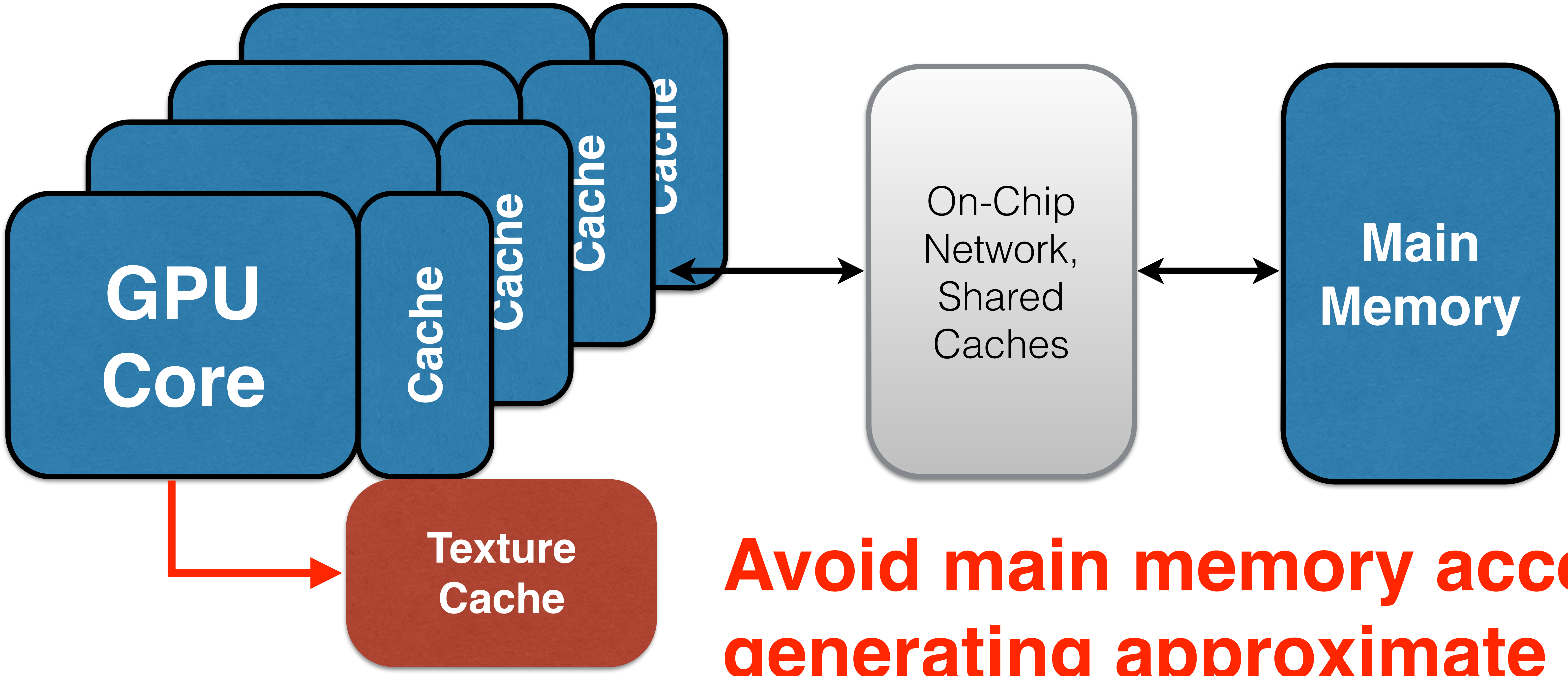
The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Our Contribution



Problem: High memory latency requires thread swapping.

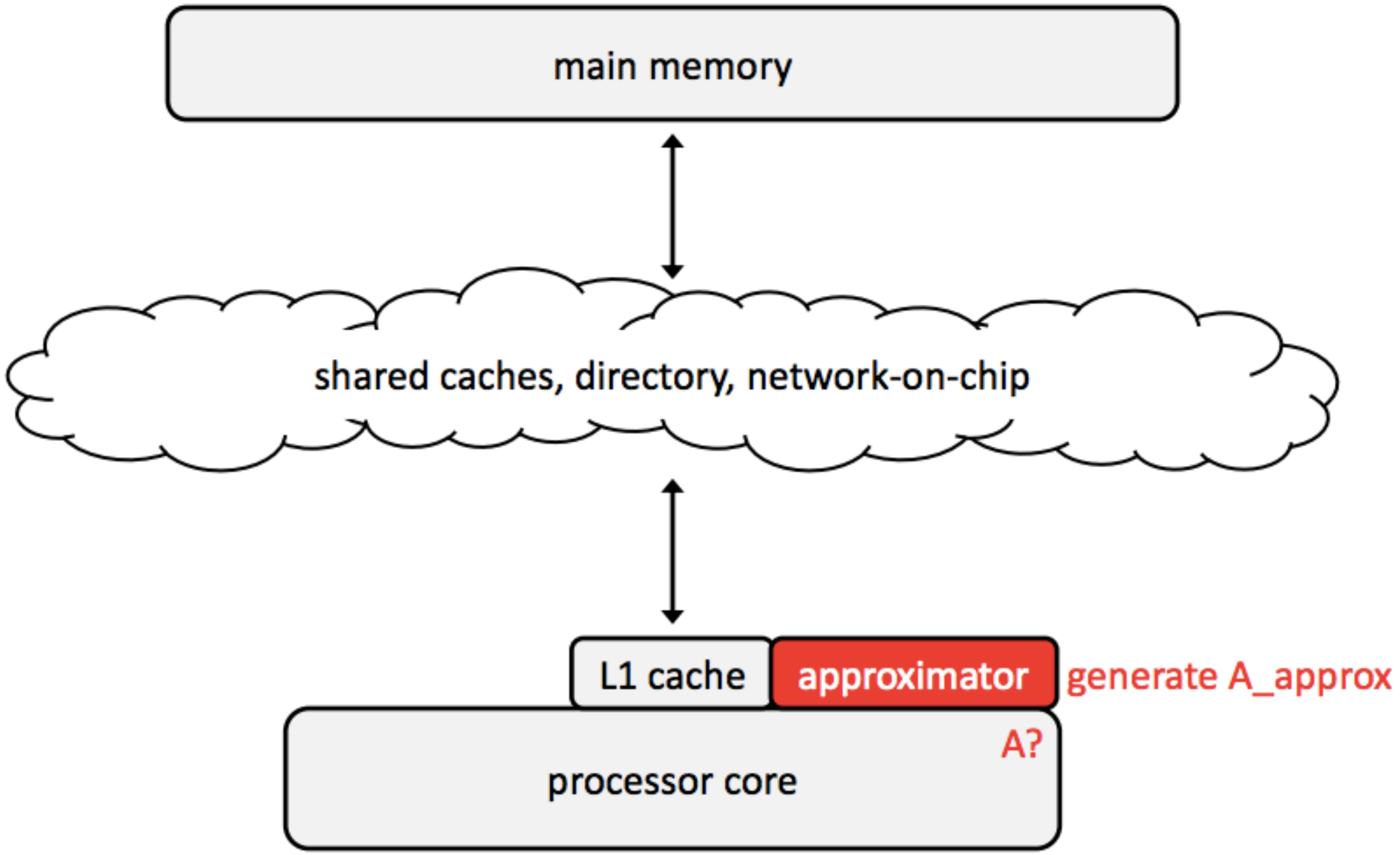
Our Contribution



Avoid main memory accesses by generating approximate values on-chip.

The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Load Value Approximation



The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Figure from "*Load Value Approximation*", Joshua San Miguel, Mario Badr, Natalie Enright Jerger

Load Value Approximation

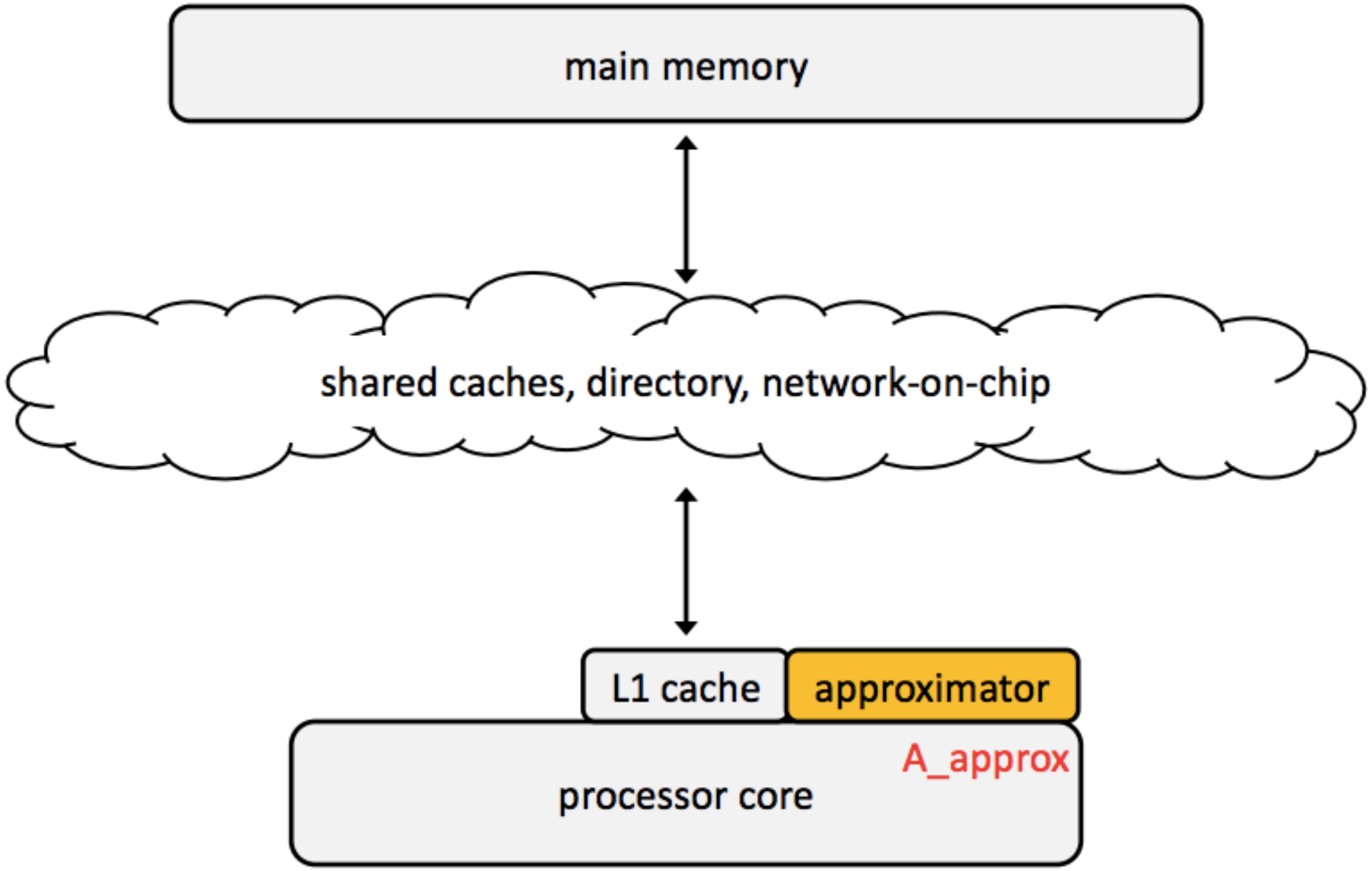
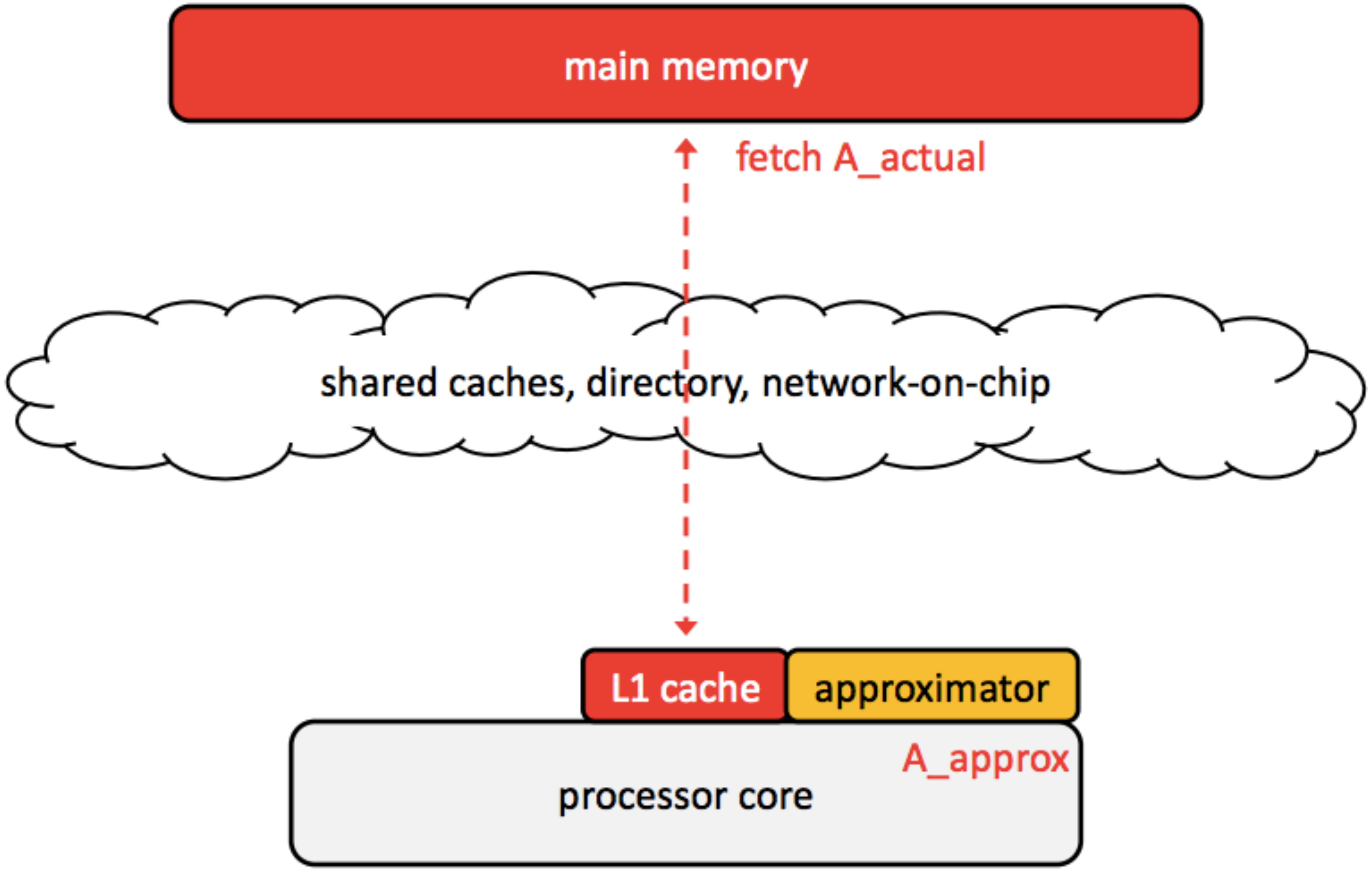


Figure from "*Load Value Approximation*", Joshua San Miguel, Mario Badr, Natalie Enright Jerger

Load Value Approximation



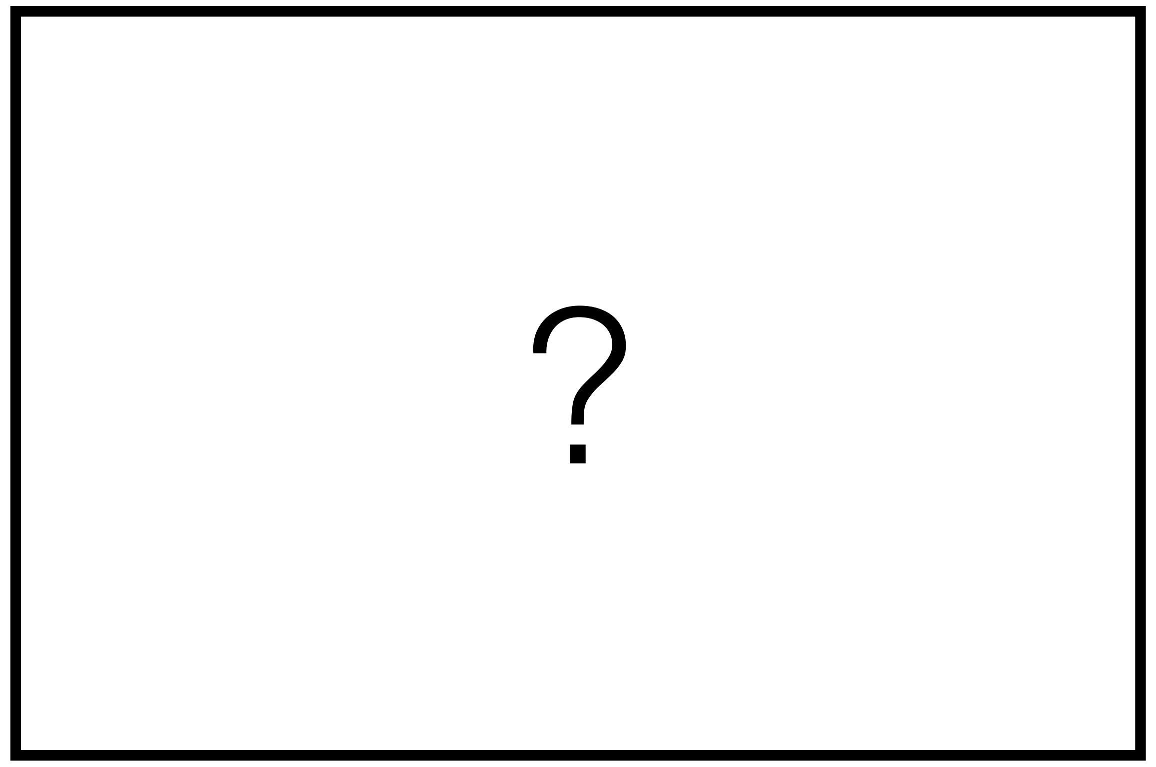
The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Figure from "*Load Value Approximation*", Joshua San Miguel, Mario Badr, Natalie Enright Jerger

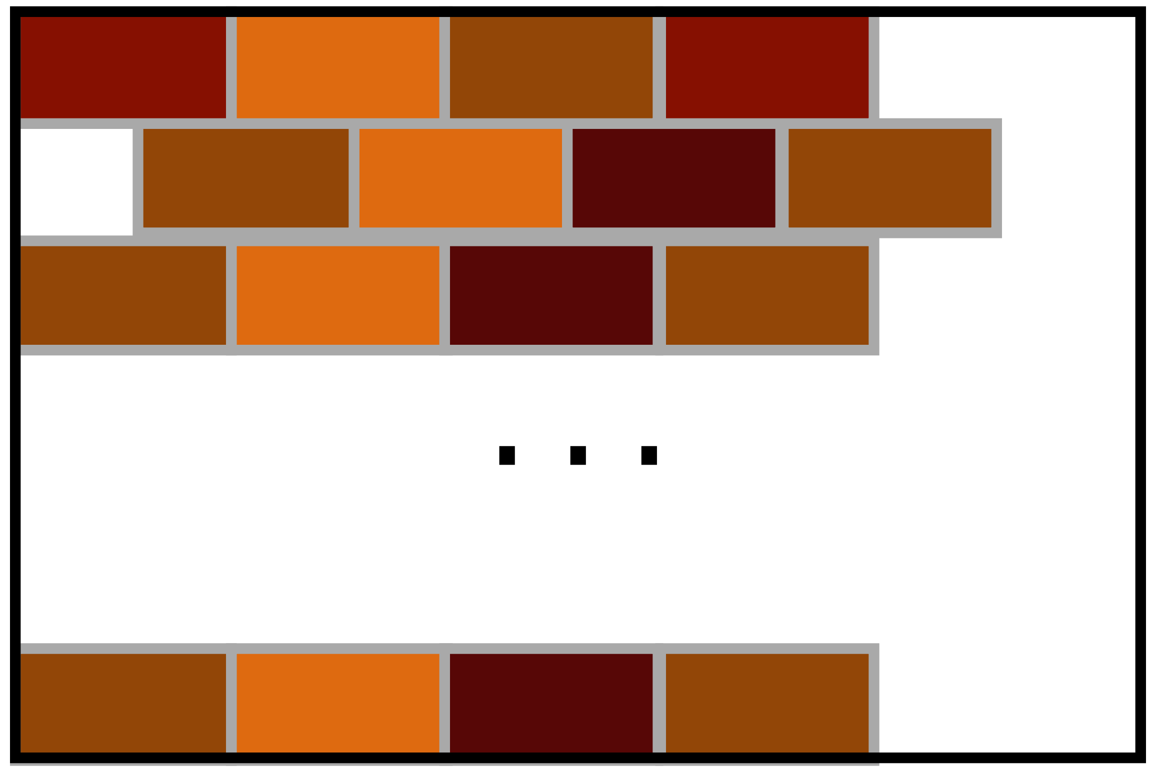
Goal: Use off-the-shelf GPU hardware to implement massively parallel value approximation.

What is a Texture?

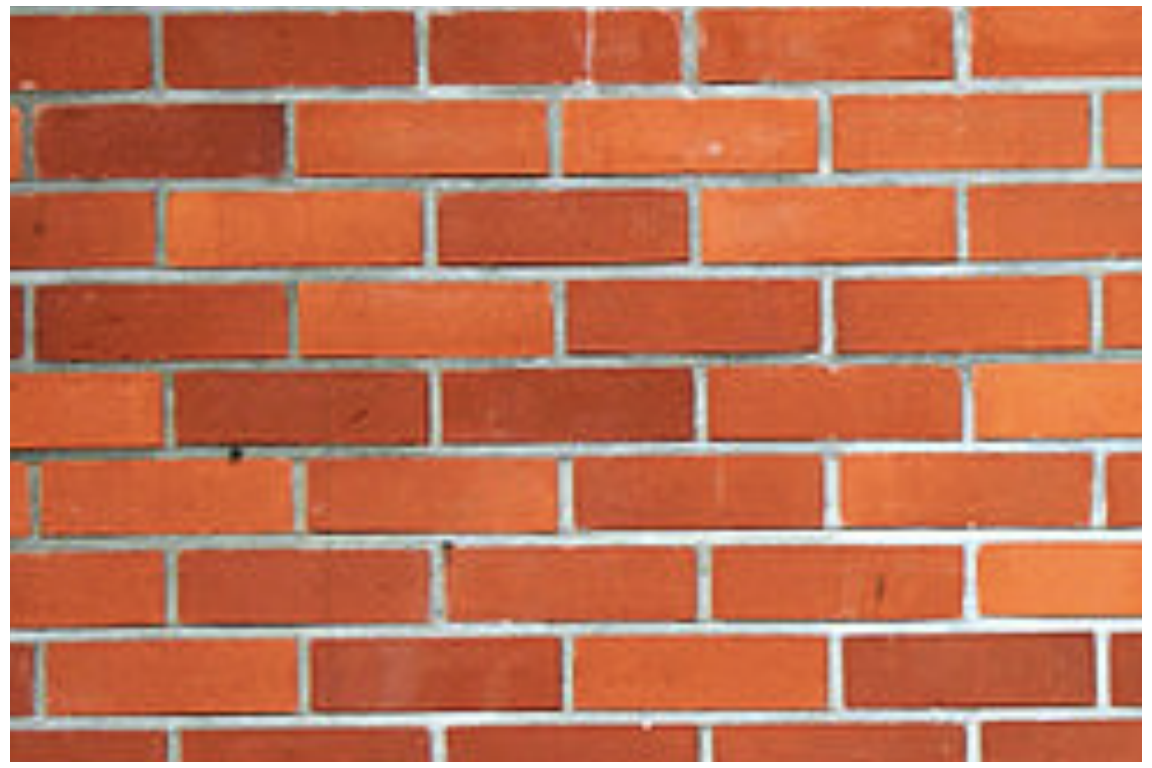
- Images are comprised of polygons, which the GPU fills using *shaders*.
- **Example**: Rendering a brick wall.
- Without a texture, the GPU can fill a polygon with a solid colour, or a gradient spanning from one vertex to another.
- Textures allow you to draw the whole wall, without filling *N* polygons for each brick.



Single-polygon (rectangle)



Multi-polygon solid



Single-polygon w. texture

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Source: http://upload.wikimedia.org/wikipedia/commons/2/28/-_Brickwall_01_-.jpg (Creative Commons)

Texture Hardware

- 12kB dedicated cache **per SM**, 4 dedicated fetch/interpolation units.
- Texture Unit Features:
  - Interpolating between data values, wrapping out-of-bounds indexes.
- Caveat: **Texture memory is read-only.**

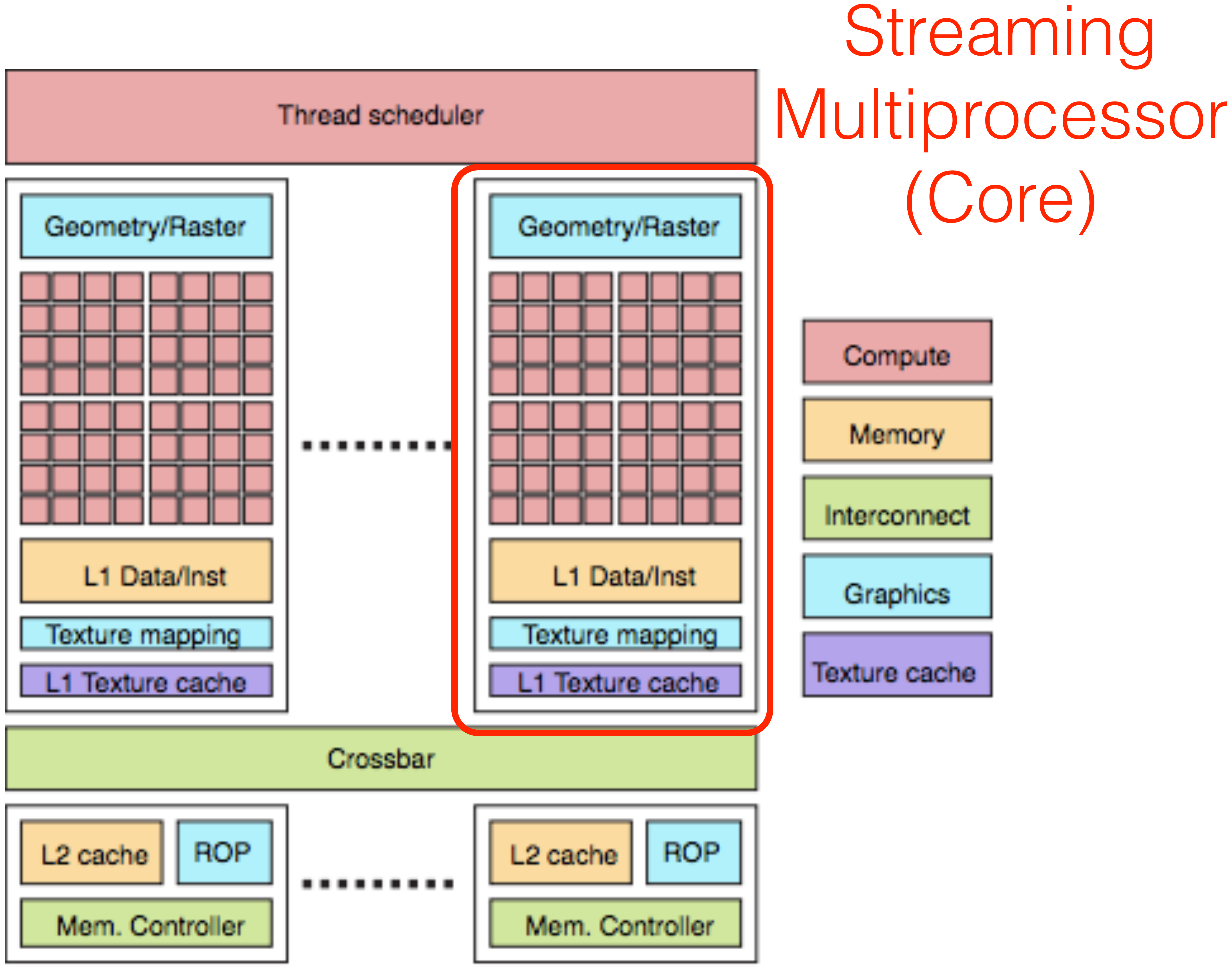


Figure source: M. Doggett. *Texture caches.* IEEE Micro, 32(3):136–141, May 2012.

Texture Cache Approximation

1. Analyze the data values read by each GPU thread, and build a training set from repetitive patterns in this data.

2. Before GPU execution, load the texture cache with this training set.

3. Replace memory accesses with approximations derived from the texture cache.

Cache Contents

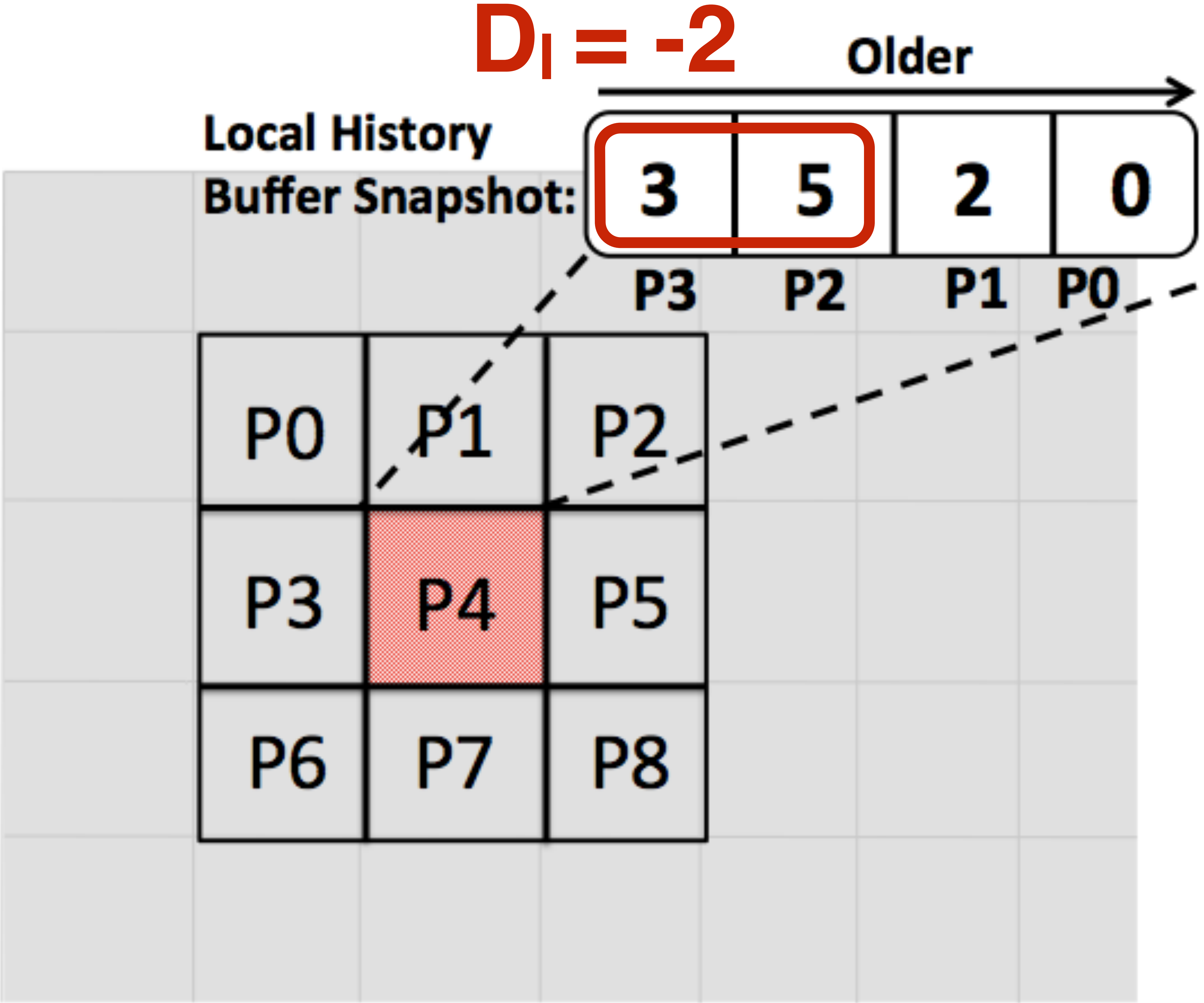
- Want our approximations to be compact, so they fit in the texture cache, as well as portable to many value patterns.

- Each approximation is the sum of the last data value, and a **delta approximation** from the texture cache:
  - $X_{apx} = X_{last} + D_{apx}$

Training Set Generation

Example: Image access pattern

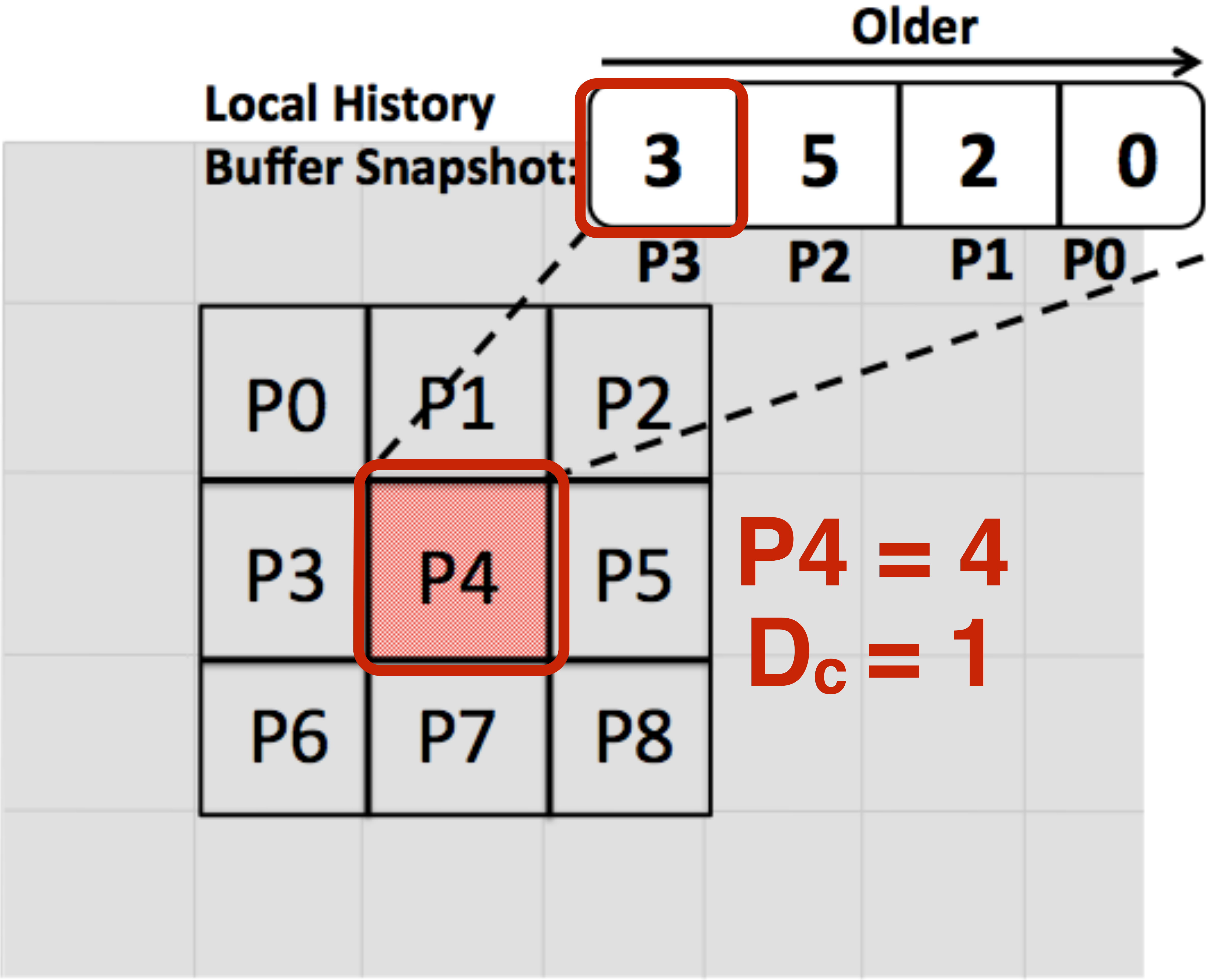
1. Upon accessing P4, thread inspects its LHB and calculates the **last** delta.
2. Baseline code then reads P4 from memory, and calculates the **current** delta.
3. Output a pair to the trace: [D_l,D_c]



Training Set Generation

Example: Image access pattern

1. Upon accessing P4, thread inspects its LHB and calculates the **last** delta.
2. Baseline code then reads P4 from memory, and calculates the **current** delta.
3. Output a pair to the trace: $[D_l, D_c]$

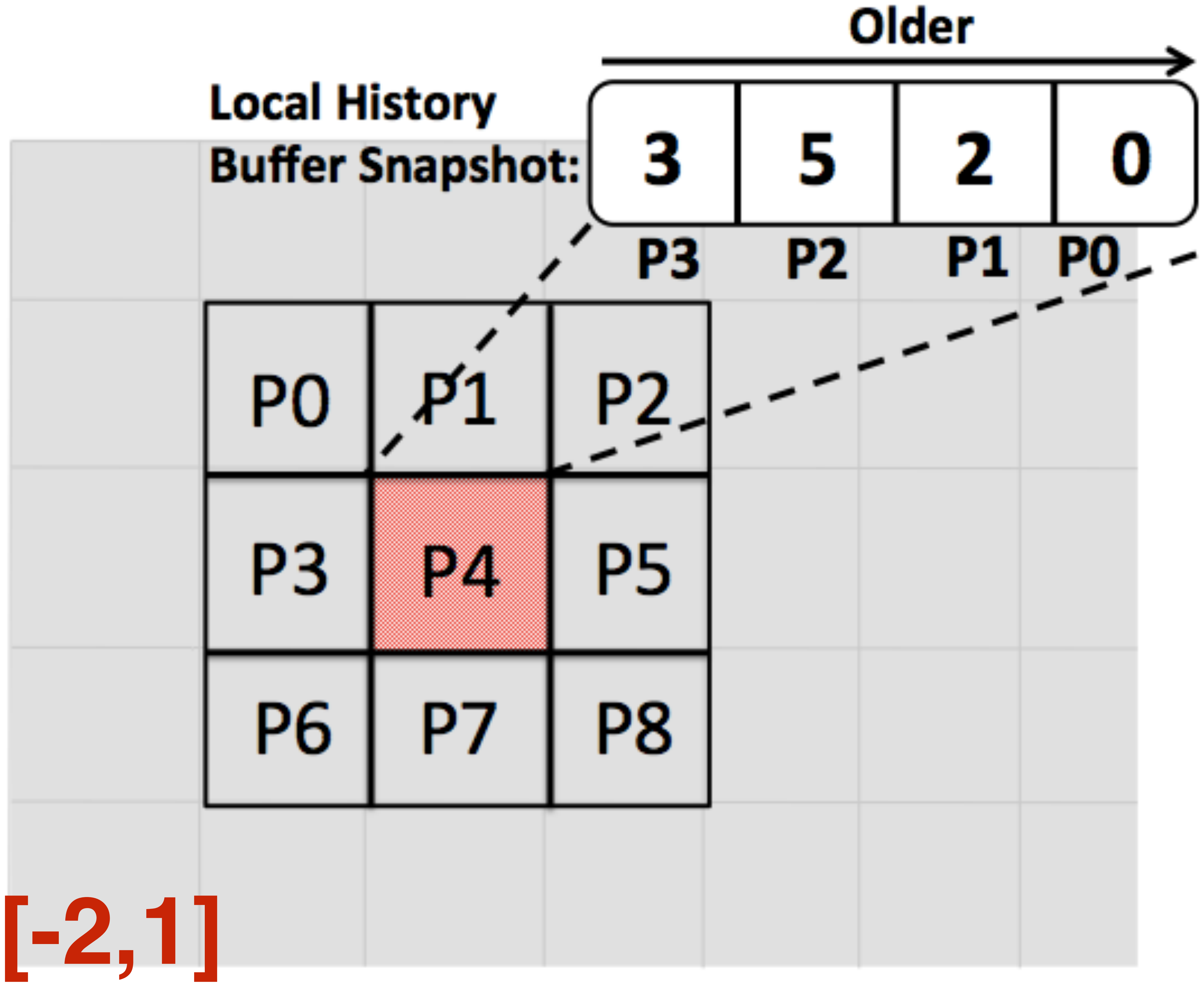


The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Training Set Generation

Example: Image access pattern

1. Upon accessing P4, thread inspects its LHB and calculates the **last** delta.
2. Baseline code then reads P4 from memory, and calculates the **current** delta.
3. Output a pair to the trace: $[D_l, D_c]$



$[D_l, D_c] = [-2,1]$

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Kernel Code Transformation

Characteristic Loop
(per thread)

```
foreach (pixel in 4x4 grid) {
  foreach (neighbour pixel) {
   … = a[neighbor];
  }
  b[pix] = … ;
}
```

Loop Body
(exact data)

Kernel Code Transformation

Characteristic Loop
(per thread)

```
foreach (pixel in 4x3 grid) {
    … = a[neighbor];
    updateLHB( );
}
foreach (pixel in last row) {
    … = LHB[0] + texture[getIndex()];
    updateLHB( );
}
```

Loop Body
(exact w. update)

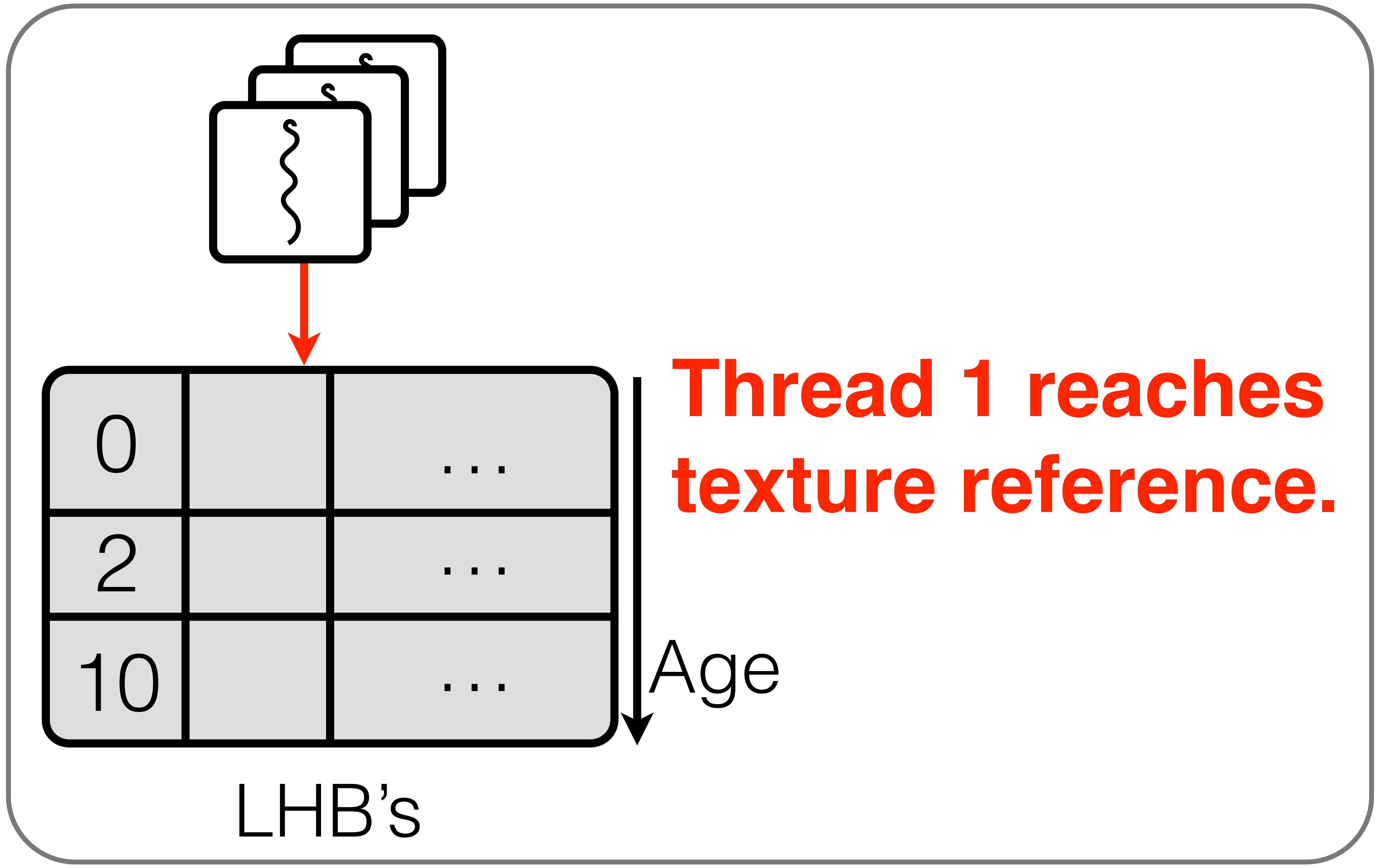
Epilogue Loop
(with approx)

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Online Approximations

… = LHB[0] + texture[getIndex( )];

In-Core Activity



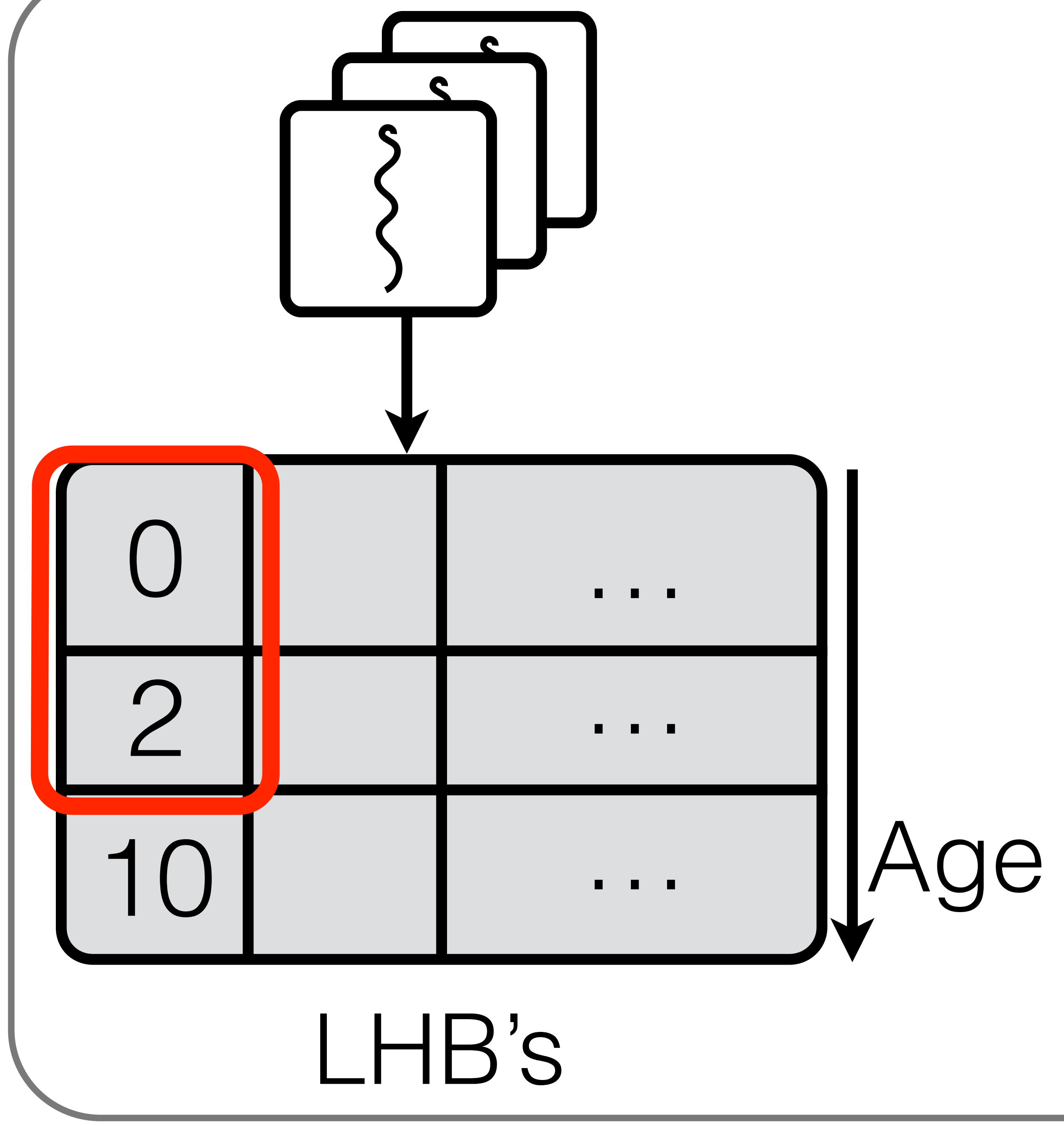
Texture Cache

Index	Value
0	1
1	4
2	8
3	-2

Online Approximations

… = LHB[0] + texture[**getIndex( )**];

In-Core Activity



Calculate:
D_{last}= -2

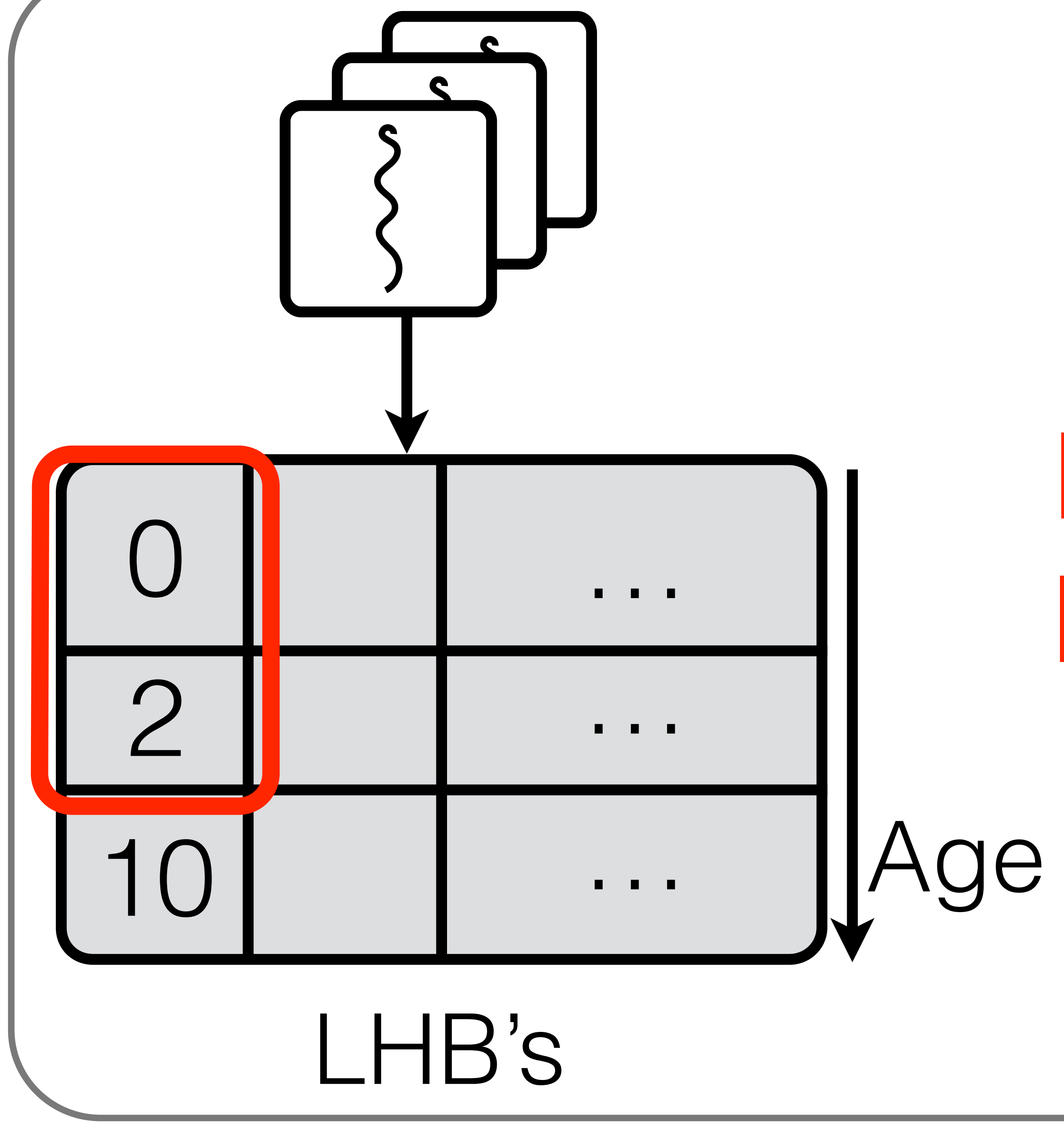
Texture Cache

Index	Value
0	1
1	4
2	8
3	-2

Online Approximations

… = LHB[0] + texture[**0**];

In-Core Activity



Normalize:
Index = 0

Texture Cache

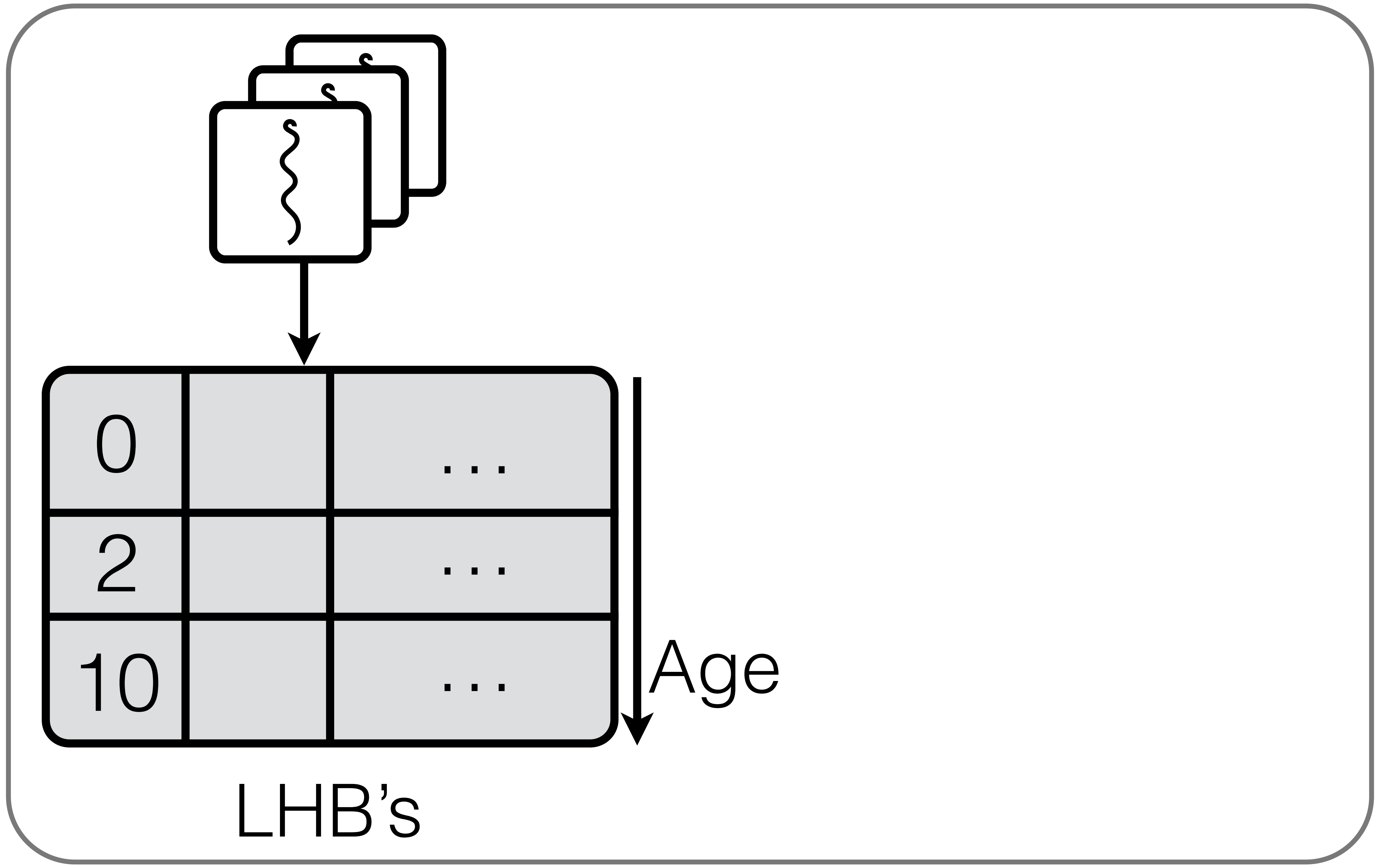
Index	Value
0	1
1	4
2	8
3	-2

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Online Approximations

… = LHB[0] + **texture[0];**

In-Core Activity



Access: texture[0]

Texture Cache

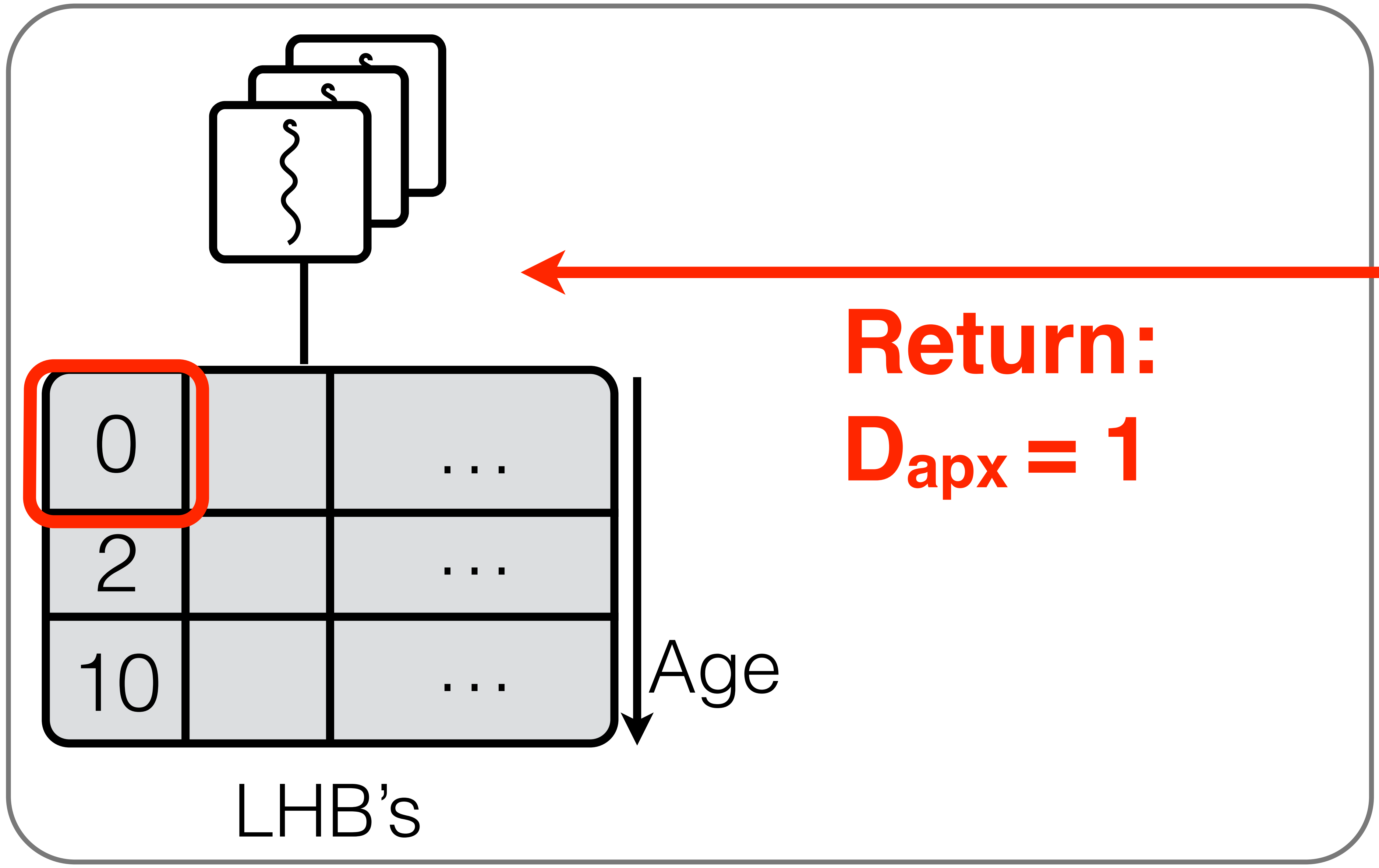
Index	**Value**
0	1
1	4
2	8
3	-2

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Online Approximations

… = **0** + **1**;

In-Core Activity



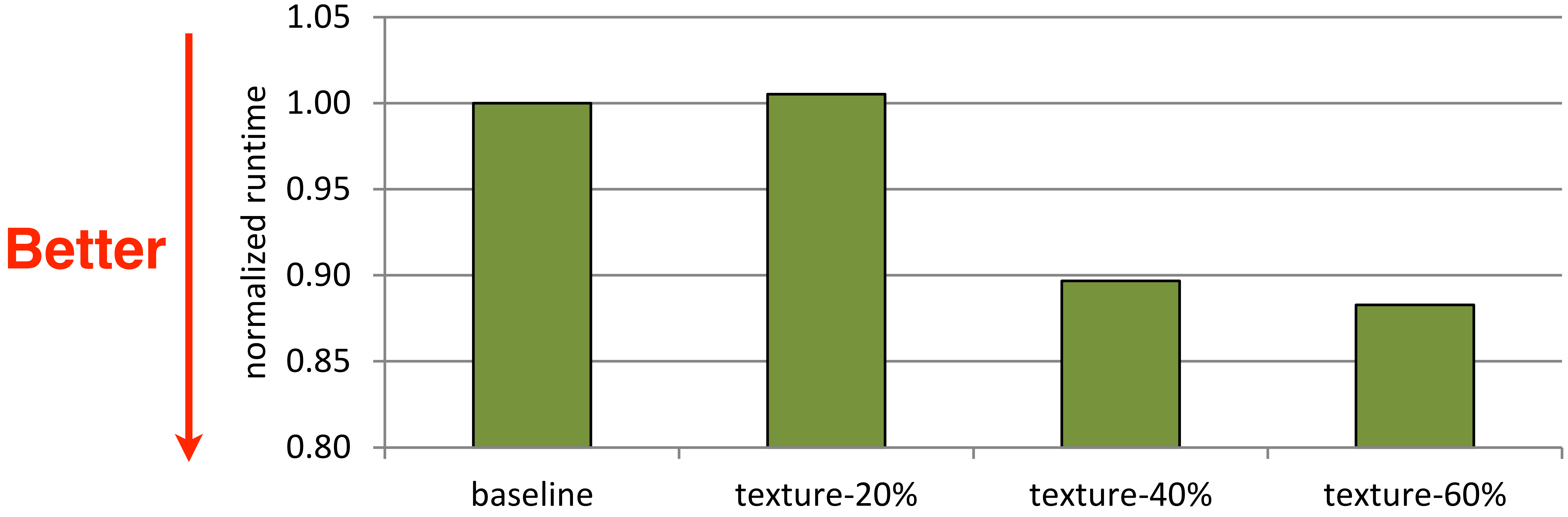
Texture Cache

Index	Value
0	1
1	4
2	8
3	-2

Evaluation

- Commodity hardware: NVIDIA 780GTX GPU (Kepler u-architecture)
- Image blur kernel derived from San Diego Computer Vision Benchmark Suite [Venkata, IISWC '09].
- Use CUDA Toolkit Profiler to measure:
  - Kernel runtime (cycles), texture cache hit rate.
- Error metric: Mean Pixel Difference [Samadi, MICRO '13]

Kernel Runtime



- Replaced *X* global memory reads (baseline has 5).
- Texture cache hit rate: **> 99%** in all cases.

Image Comparison

Exact

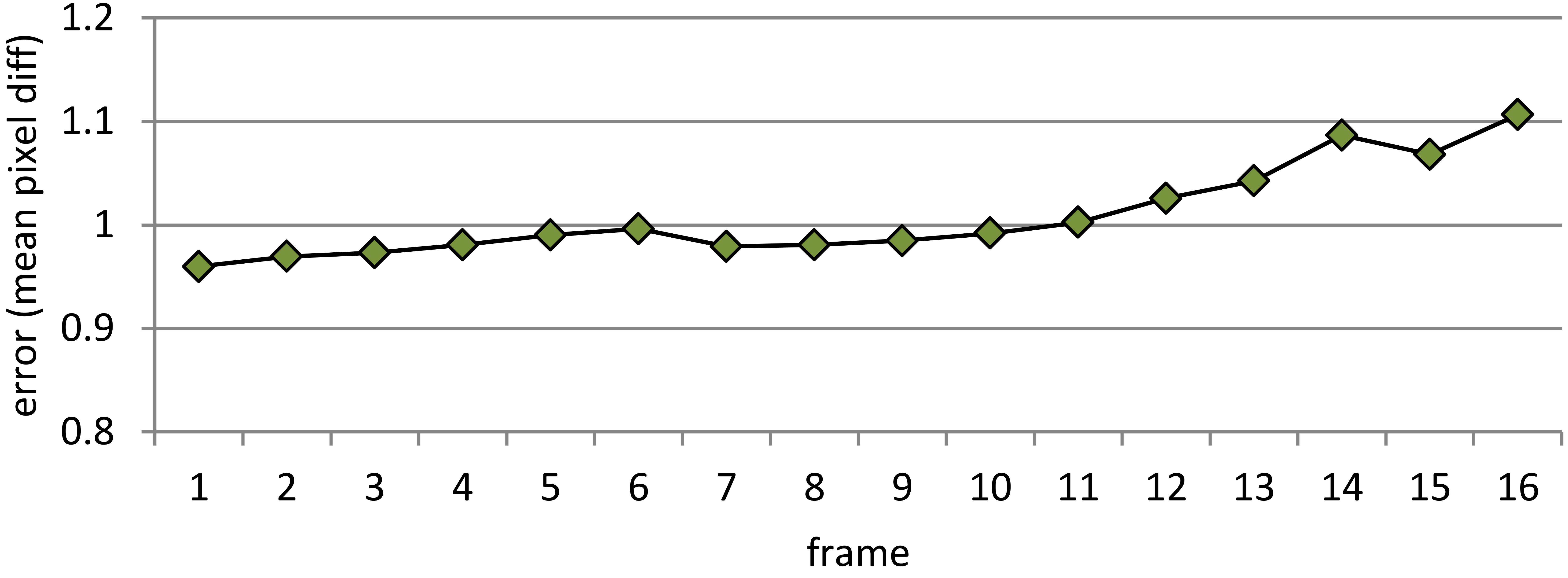
Approximate

Error: 0.4%

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Image source: Nature stock footage archive. http://downloadnatureclip.blogspot.ca/p/download-links.html. Accessed: 2015-03-28.

Error Evolution



- Used same training set (from F1) for 16 images in a video sequence, evaluated error with 40% of loads replaced.

Future Work

- Evaluate on applications from different application domains: machine learning, physics & fluid simulations, data queries.
- Can we eliminate the need for training sets?
- Improve speedup on floating point benchmarks.

Conclusion:

Under-utilized texture hardware on GPUs can be used to accelerate kernel execution using value approximation.

?

Comparison to Reduced Blur

Large Radius

Small Radius

Image source: Nature stock footage archive. http://downloadnatureclip.blogspot.ca/p/download-links.html. Accessed: 2015-03-28.

Approximate Value Computing

1. Certain applications are robust to inexact data values.
    - Data mining and pattern recognition [Chippa, DAC ’13]

2. Where can these values come from?
    - Reduced-voltage DRAM [Liu, ASPLOS ’12]
    - Kernels edited to remove synchronization [Samadi, MICRO ’13]
    - Load Value Approximation [San Miguel, MICRO ’14]

Why Approximate?

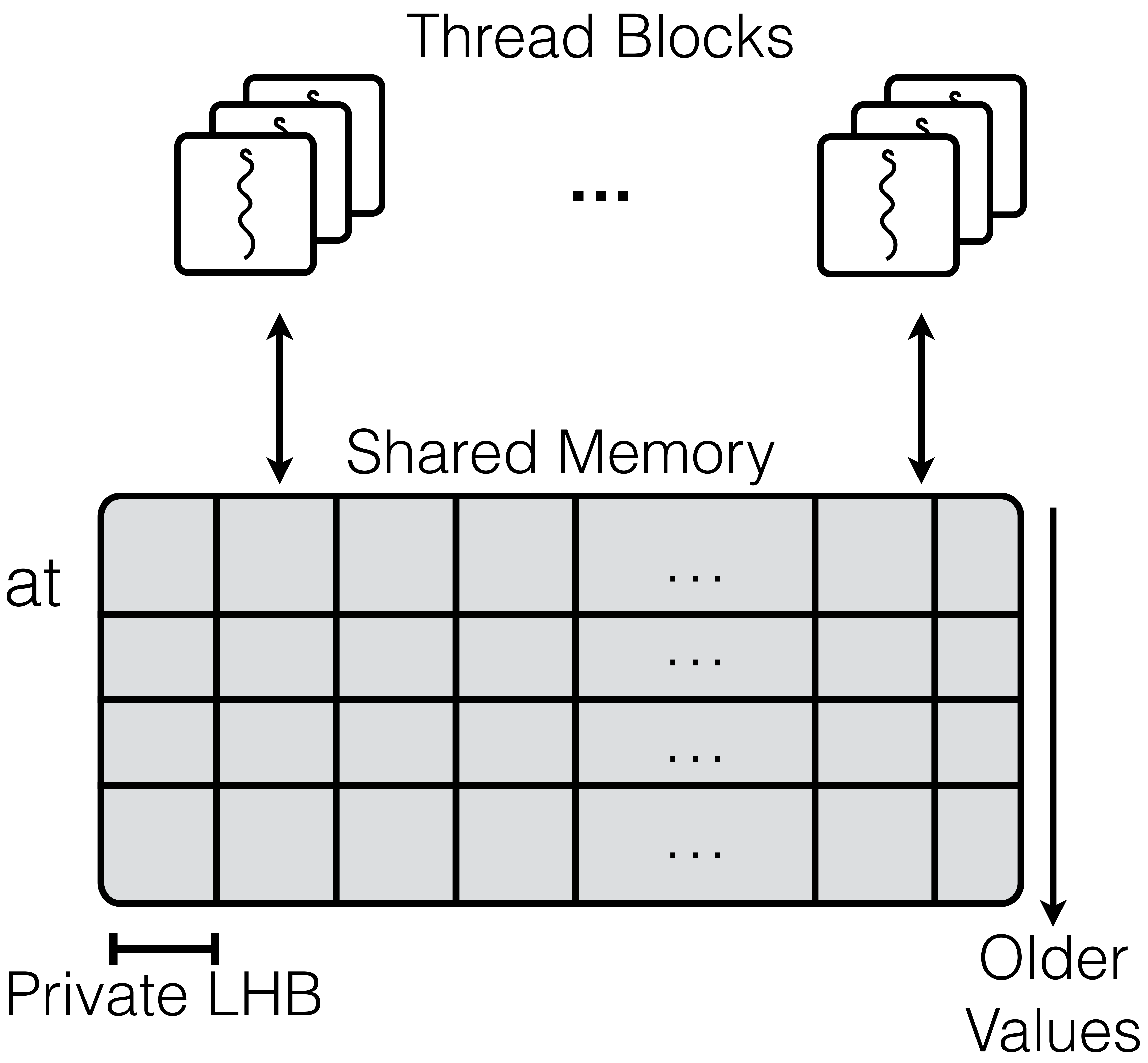
1. GPU memory architecture has similar drawbacks to that of a CPU.

2. Modern GPU’s choose to hide latency with thread swapping and context storage.
   - **Could easily put these transistors to better use!**

Training Set Generation

1. Give every thread its own Local History Buffer (LHB) in shared memory.
2. Upon every memory access, output the **previous** and **next** deltas to a trace.

- This allows us to analyze different **value** patterns, and generate approximations that are accurate for many thread blocks.



Kernel Code Transformation

Characteristic Loop
(per thread)

```
foreach (pixel in 4x3 grid) {
        foreach (neighbour pixel) {
          … = a[np];
          updateLHB(a[np]);
        }
        b[p] = … ;
}
foreach (pixels left in 4x1 stripe) {
       …
}
```

Loop Body
(exact w. update)

Epilogue Loop Body

Kernel Code Transformation

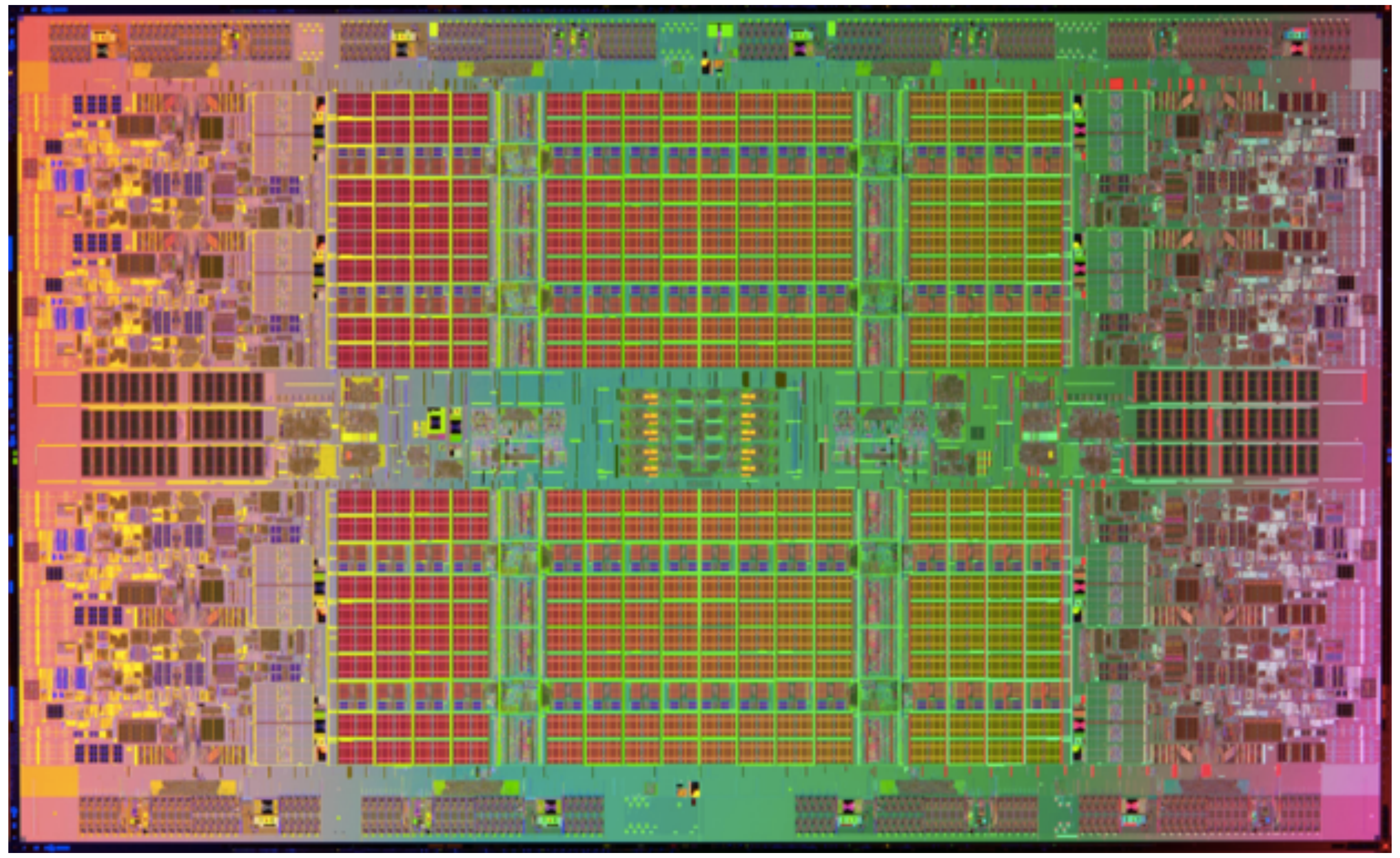
```
foreach (pixel in 4x3 grid) {
        …
        updateLHB(a[np]);
}
foreach (pixels left in 4x1 stripe) {
    foreach(neighbour pixel) {
        … = getTexture(my_tex, getDeltaFromLHB( ) );
        updateLHB(…);
    }
    b[p] = … ;
}
```

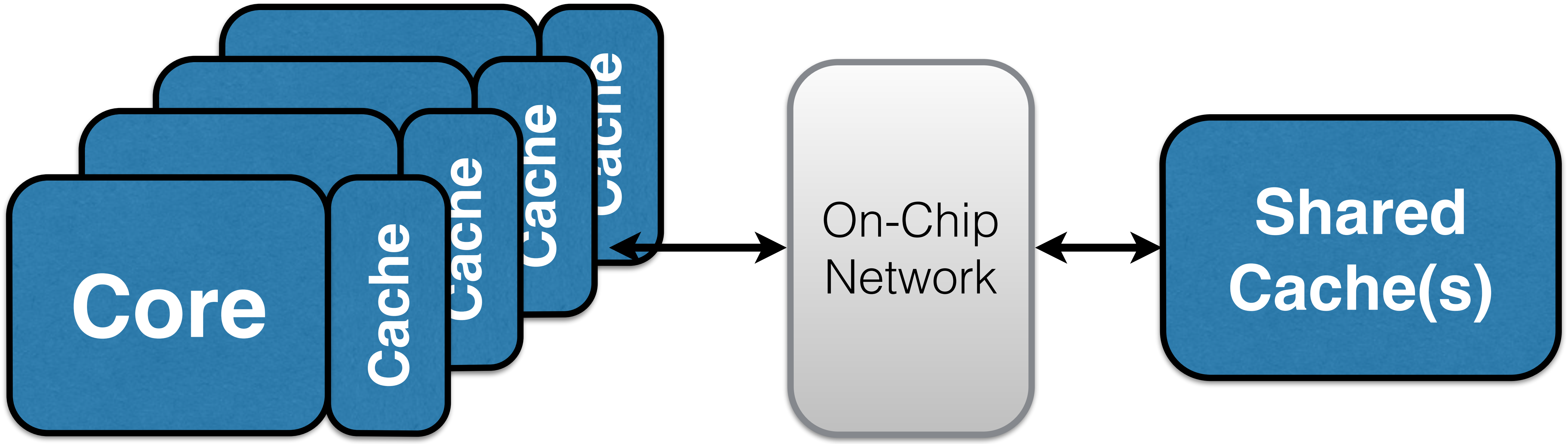
Loop Body
(exact w. update)

Epilogue Loop Body
(replace loads with texture references)

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Processing Continuum





Accelerator[1] — Multi-Core CPU

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

1. Source: www.newsroom.intel.com/community/intel_newsroom

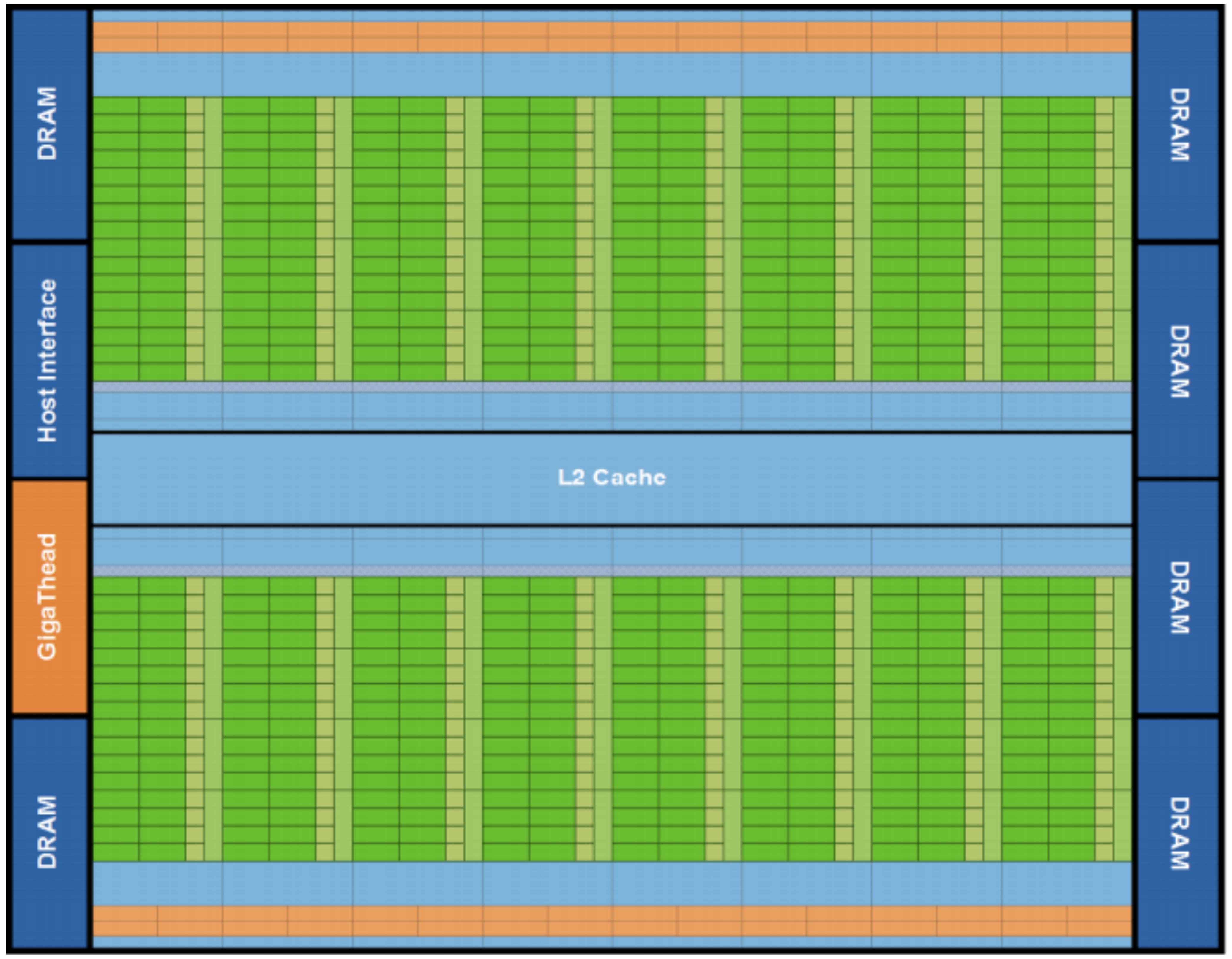
GPU Niche

GPGPU

S ⟷ **G**

Characteristics

- Massively multi-core.
- Trades latency for throughput.
- Very few optimizations for single-thread performance.



Source: http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO