Efficient Management of LLCs in GPUs for 3D Scene Rendering Workloads

Jayesh Gaur (Intel)
Raghuram Srinivasan (Ohio State)
Sreenivas Subramoney (Intel)
Mainak Chaudhuri (IIT, Kanpur)

Sketch

- Talk in one slide
- Result highlights
- Understanding the potential
- Reuses in 3D graphics data
- Our policy proposals
- Evaluation methodology
- Simulation results
- Summary

Sketch

- Talk in one slide
- Result highlights
- Understanding the potential
- Reuses in 3D graphics data
- Our policy proposals
- Evaluation methodology
- Simulation results
- Summary

Talk in One Slide

- 3D scene rendering pipeline generates accesses to different types of data
  - Vertex, vertex index, depth, hierarchical depth, stencil, render targets (same as pixel colors), and textures for sampling
  - GPUs include small render caches for each such data type and more recently read/write last-level caches (LLCs) shared by all such data streams

Talk in One Slide

- 3D scene rendering pipeline generates accesses to different types of data
  - Vertex, vertex index, depth, hierarchical depth, stencil, render targets (same as pixel colors), and textures for sampling
  - GPUs include small render caches for each such data type and more recently read/write last-level caches (LLCs) shared by all such data streams
- Our proposal: graphics stream-aware probabilistic caching (GSPC) for GPU LLC
  - Learns inter- and intra-stream reuse probabilities from a few sample LLC sets and modulates insertion/promotion in other sets

Result highlights

- Three increasingly better policies coupled with uncached displayable color data
  - Baseline: two-bit DRRIP
  - Workloads: 52 DirectX frames selected from eight game titles and four benchmark applications using Direct3D 10 and 11 APIs

Result highlights

- Three increasingly better policies coupled with uncached displayable color data
  - Baseline: two-bit DRRIP
  - Workloads: 52 DirectX frames selected from eight game titles and four benchmark applications using Direct3D 10 and 11 APIs
  - LLC miss saving: up to 29.6% and on average 13.1% with an 8 MB 16-way LLC

Result highlights

- Three increasingly better policies coupled with uncached displayable color data
  - Baseline: two-bit DRRIP
  - Workloads: 52 DirectX frames selected from eight game titles and four benchmark applications using Direct3D 10 and 11 APIs
  - LLC miss saving: up to 29.6% and on average 13.1% with an 8 MB 16-way LLC
  - Frame rate improvement: up to 18.2% and on average 8.0%; with increasing LLC capacity, it gets even better (11.8% with a 16 MB LLC)

Result highlights

- Three increasingly better policies coupled with uncached displayable color data
  - Baseline: two-bit DRRIP
  - Workloads: 52 DirectX frames selected from eight game titles and four benchmark applications using Direct3D 10 and 11 APIs
  - LLC miss saving: up to 29.6% and on average 13.1% with an 8 MB 16-way LLC
  - Frame rate improvement: up to 18.2% and on average 8.0%; with increasing LLC capacity, it gets even better (11.8% with a 16 MB LLC)
  - More important as GPUs get more aggressive

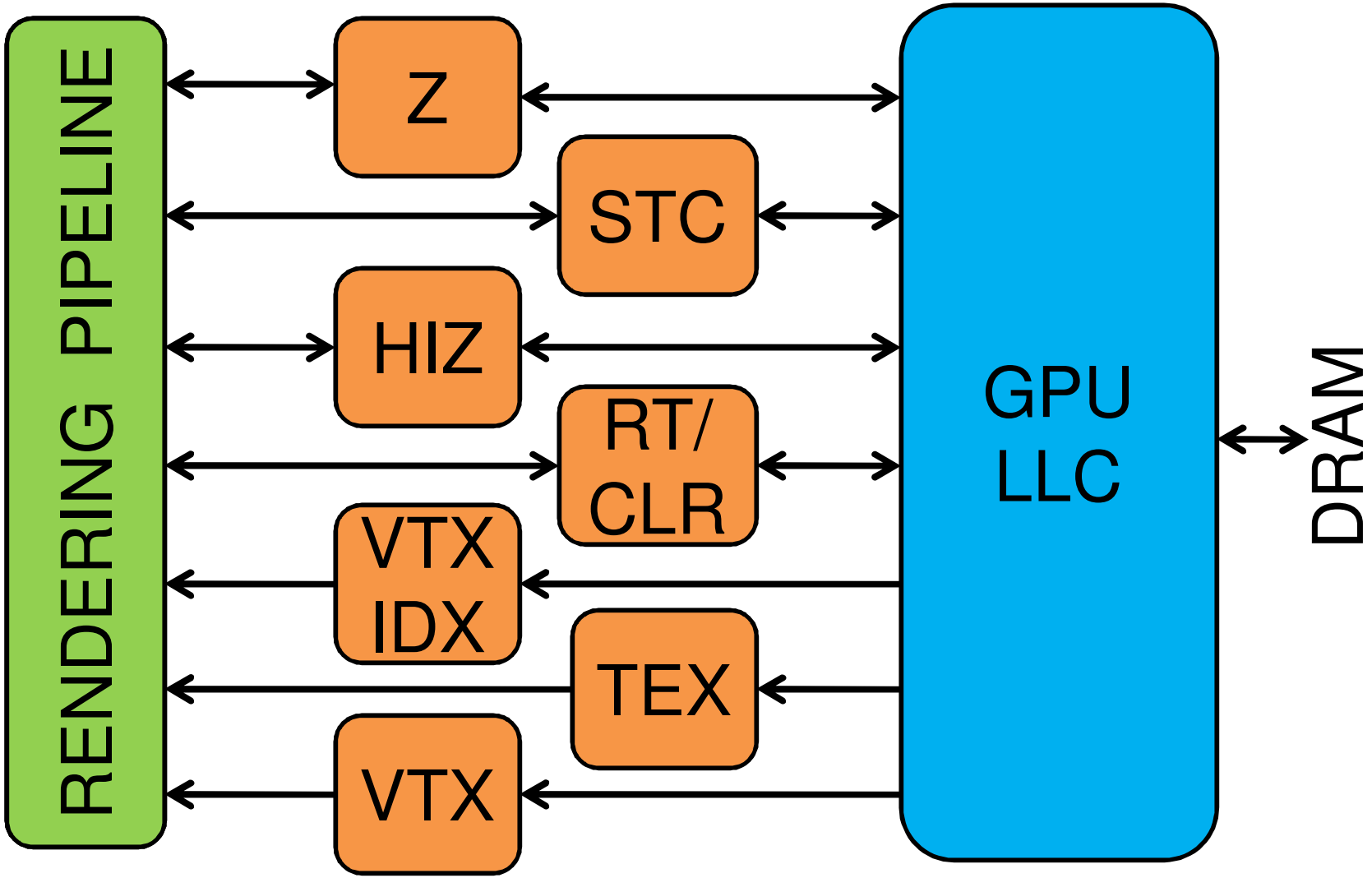
Sketch

- Talk in one slide
- Result highlights
- ➢ Understanding the potential
- Reuses in 3D graphics data
- Our policy proposals
- Evaluation methodology
- Simulation results
- Summary

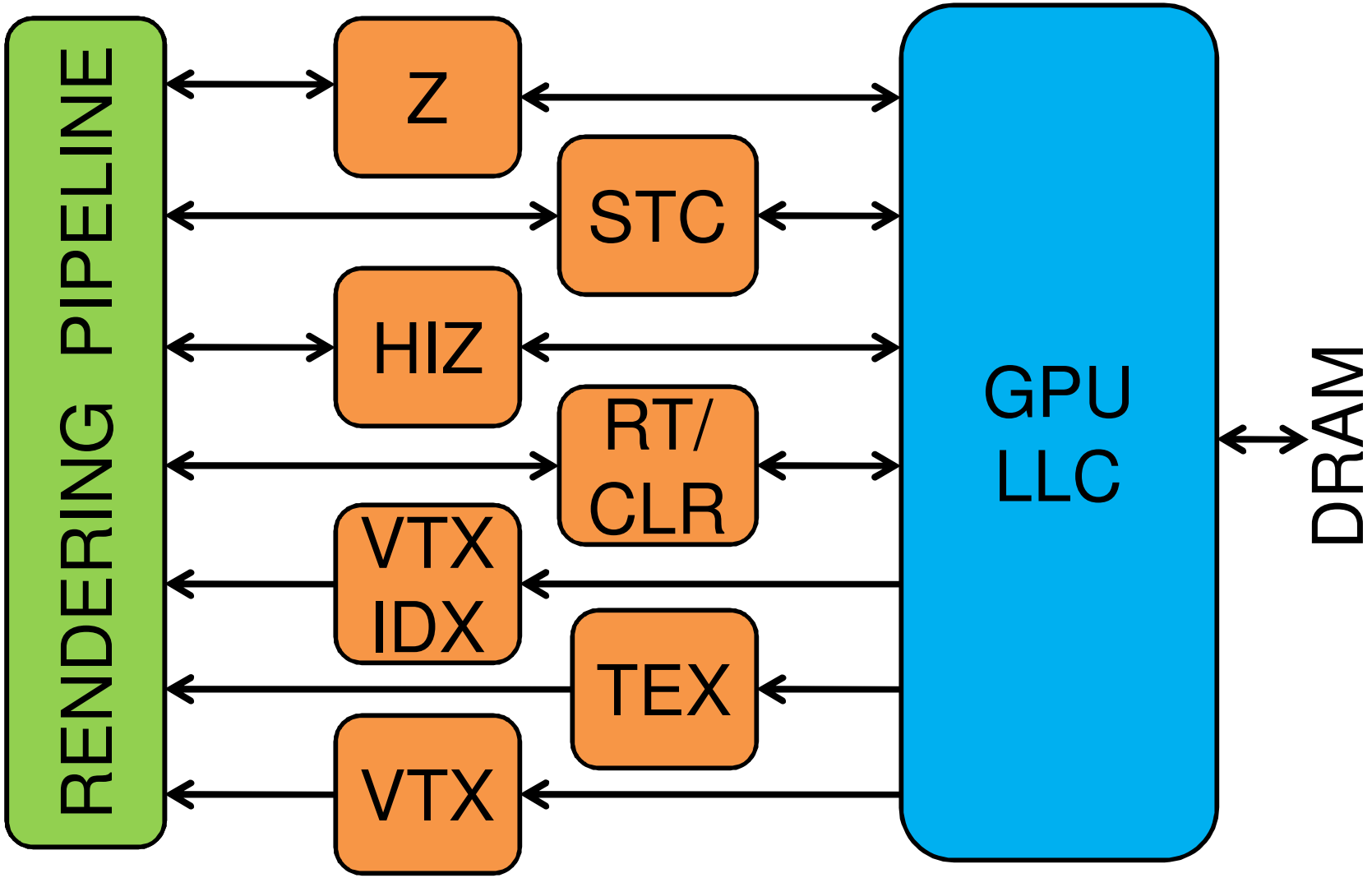
Characterization framework

- Functional LLC model
  - 8 MB 16-way 64-byte blocks
  - Digests LLC access traces collected from a detailed timing simulator of a high-end GPU
- Load/Store trace collection
  - 52 frames are selected from twelve DirectX applications that use Direct3D 10 and 11 APIs
    - Eight games and four benchmark applications
  - All Direct3D APIs are intercepted in each frame and replayed through the detailed simulator
  - All LLC accesses are logged in a trace
  - The modeled LLC is non-inclusive/non-exclusive

GPU last-level cache interface

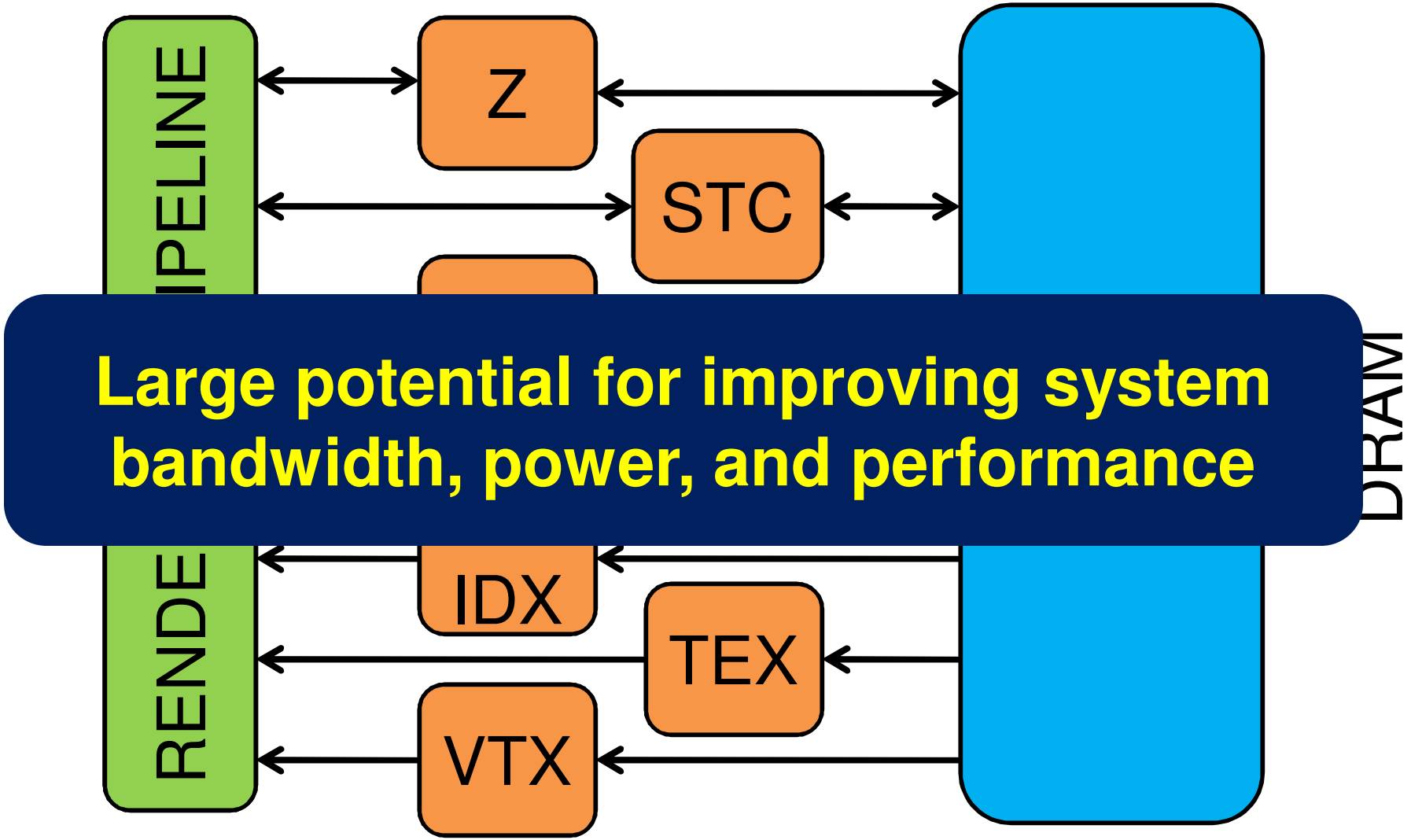


GPU last-level cache interface



Belady's optimal policy projects a 36.6% average saving in LLC misses compared to two-bit DRRIP

GPU last-level cache interface



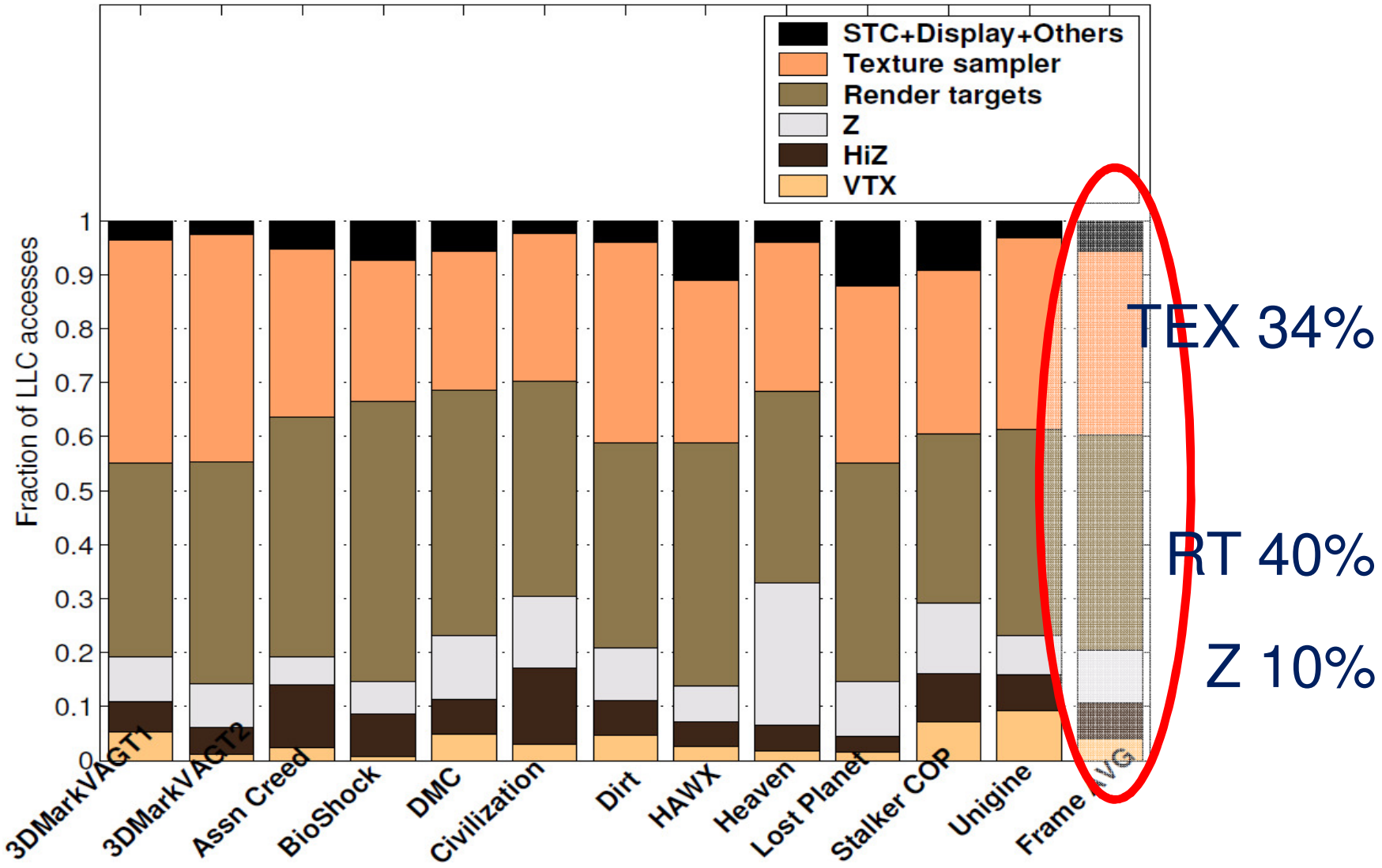
Belady's optimal policy projects a 36.6% average saving in LLC misses compared to two-bit DRRIP

LLC accesses

- LLC accesses arise due to misses in the GPU render caches
  - For example, a sampler request comes to the LLC only if the access has missed in all levels of the texture cache hierarchy of the GPU
- The LLC accesses can be partitioned based on the source of the request
  - Each such partition will be referred to as a 3D graphics stream
  - We consider eight streams: Vertex, HiZ, Z, render target (RT), texture sampler (TEX), stencil (STC), displayable color, and the rest (shader code, constants, etc.)

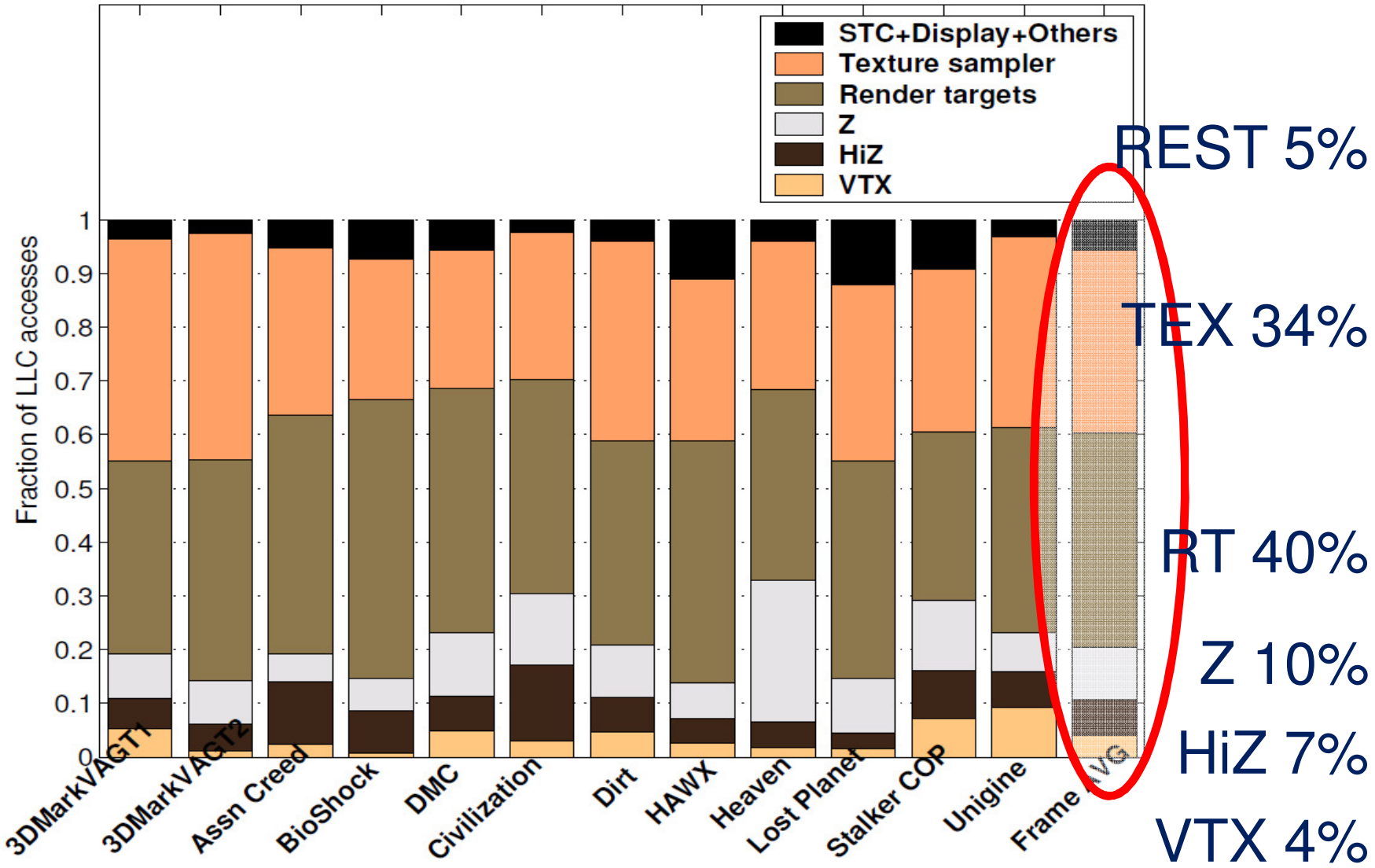
LLC access traffic

- Which 3D graphics streams are important?



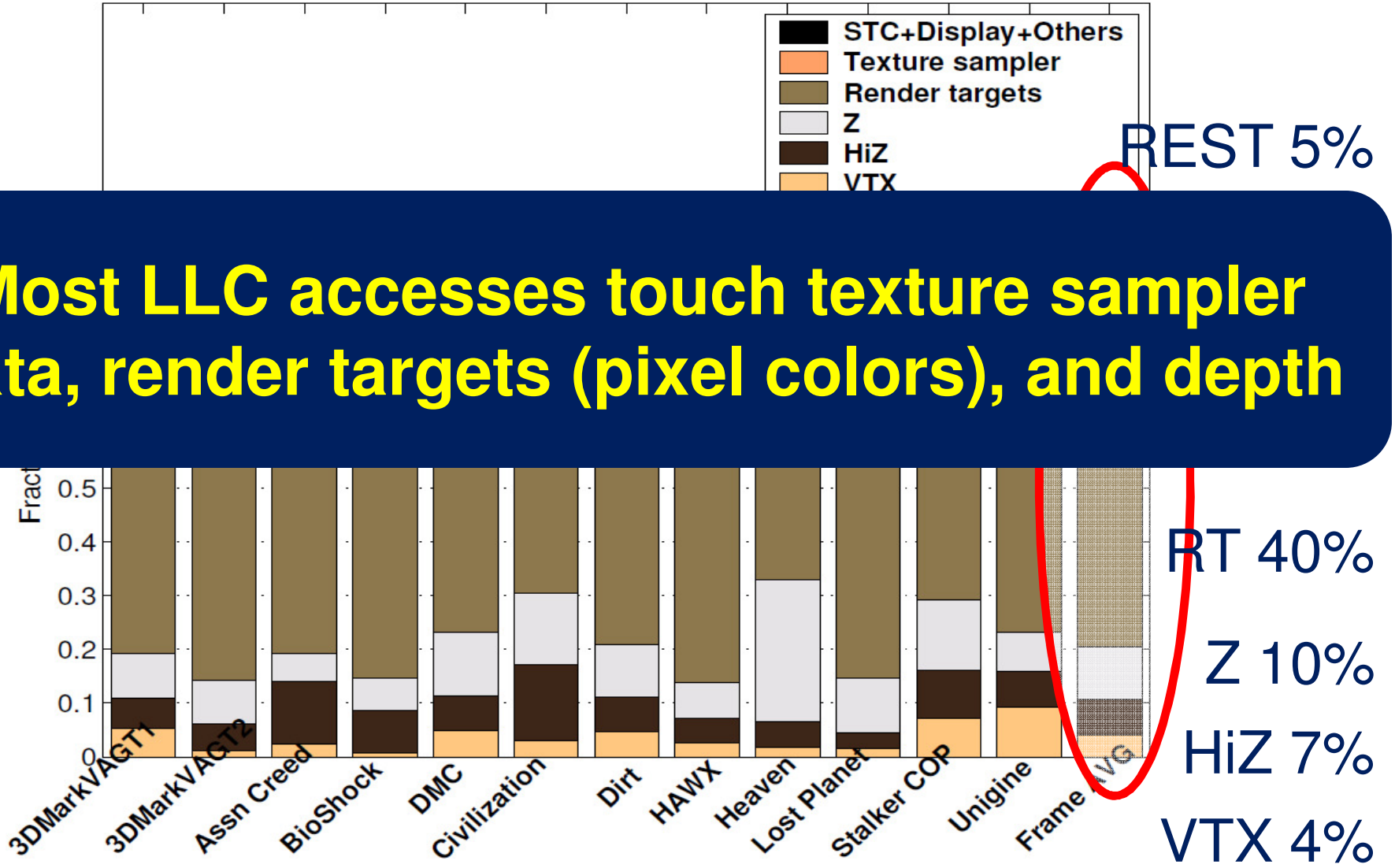
LLC access traffic

- Which 3D graphics streams are important?



LLC access traffic

- Which 3D graphics streams are important?



Most LLC accesses touch texture sampler data, render targets (pixel colors), and depth

LLC read hit rates (8 MB 16-way)

- Texture sampler data
  - Belady's optimal: 53.4%
  - Two-bit DRRIP: 22.0%
  - Single-bit NRU: 18.4%

LLC hits arise from render to texture reuses and intra-stream texture reuses

LLC read hit rates (8 MB 16-way)

- Texture sampler data
  - Belady's optimal: 53.4%
  - Two-bit DRRIP: 22.0%
  - Single-bit NRU: 18.4%
- Render targets
  - Belady's optimal: 59.8%
  - Two-bit DRRIP: 50.1%
  - Single-bit NRU: 41.5%

LLC hits arise from render to texture reuses and intra-stream texture reuses

LLC hits arise from intra-stream render target blend ops

LLC read hit rates (8 MB 16-way)

- Texture sampler data
  - Belady's optimal: 53.4%
  - Two-bit DRRIP: 22.0%
  - Single-bit NRU: 18.4%
- Render targets
  - Belady's optimal: 59.8%
  - Two-bit DRRIP: 50.1%
  - Single-bit NRU: 41.5%
- Depth
  - Belady's optimal: 77.1%
  - Two-bit DRRIP: 58.0%
  - Single-bit NRU: 58.0%

LLC hits arise from render to texture reuses and intra-stream texture reuses

LLC hits arise from intra-stream render target blend ops

LLC hits arise from intra-stream depth reuses

LLC read hit rates (8 MB 16-way)

- Texture sampler data
  - Belady's optimal: 53.4%
  - Two-bit DRRIP: 22.0%
  - Single-bit NRU: 18.4%

LLC hits arise from render to texture reuses and intra-stream texture reuses

Texture sampler data presents the largest opportunity for improvement

  - Single-bit NRU: 41.5%

target blend ops

- Depth
  - Belady's optimal: 77.1%
  - Two-bit DRRIP: 58.0%
  - Single-bit NRU: 58.0%

LLC hits arise from intra-stream depth reuses

Sketch

- Talk in one slide
- Result highlights
- Understanding the potential
- ➢ Reuses in 3D graphics data
- Our policy proposals
- Evaluation methodology
- Simulation results
- Summary

LLC reuse study#1: Texture

- LLC hits enjoyed by the texture samplers come from two sources
- Inter-stream reuses
  - A previously created render target block is consumed by the texture samplers from the LLC
  - Arises from a technique called render to texture, very popular for generating dynamic textures that need to be updated on a per-frame basis
  - Examples include waves, moving clouds, foliage, fluttering cloth, smoke, fire, and many more
- Intra-stream reuses
  - A texture block, previously used by the samplers, is reused by the samplers from the LLC

LLC reuse study#1: Texture

- Distinguishing inter-stream reuses from intra-stream reuses
  - Attach one bit with each LLC block; call it RT bit
  - All LLC blocks accessed/filled by the render target stream have the RT bit set
  - A texture sampler access that consumes an LLC block with the RT bit set is identified as an inter-stream reuse; the RT bit gets reset at this point
  - All other texture sampler hits in the LLC are classified as intra-stream reuses

LLC reuse study#1: Texture

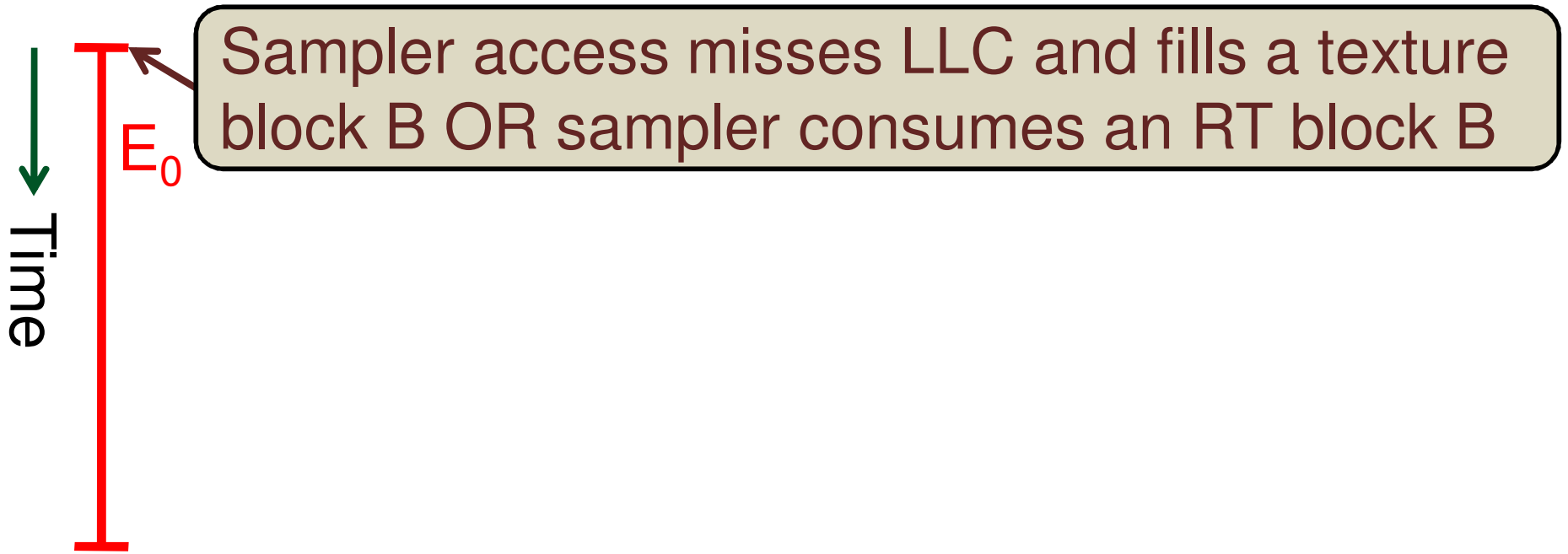
- Out of all LLC hits enjoyed by the texture sampler accesses in Belady's optimal policy
  - Inter-stream: 55%, Intra-stream: 45% averaged over 52 frames drawn from twelve DirectX apps
- Inter-stream reuses
  - Out of all LLC blocks with the RT bit set, 51% are consumed by the texture samplers in Belady's optimal policy
  - DRRIP and NRU: 16%, 13%
  - With Belady's optimal policy, each of the twelve applications has at least one-third RT blocks consumed by the texture samplers

LLC reuse study#1: Texture

- Inter-stream reuse: take-away
  - Retaining RT blocks in the LLC is important for improving texture sampler throughput
  - Without driver assistance, it is difficult to identify the render targets that will be used as textures
  - In this work, we consider all render targets to be potential source for dynamic textures and refine this set based on render to texture reuse probability learned at run-time
- Why DRRIP falls so much short of optimal
  - Fills 25% of the RT blocks with RRPV three
  - Level of protection for RT fills must be decided from the render to texture reuse probability

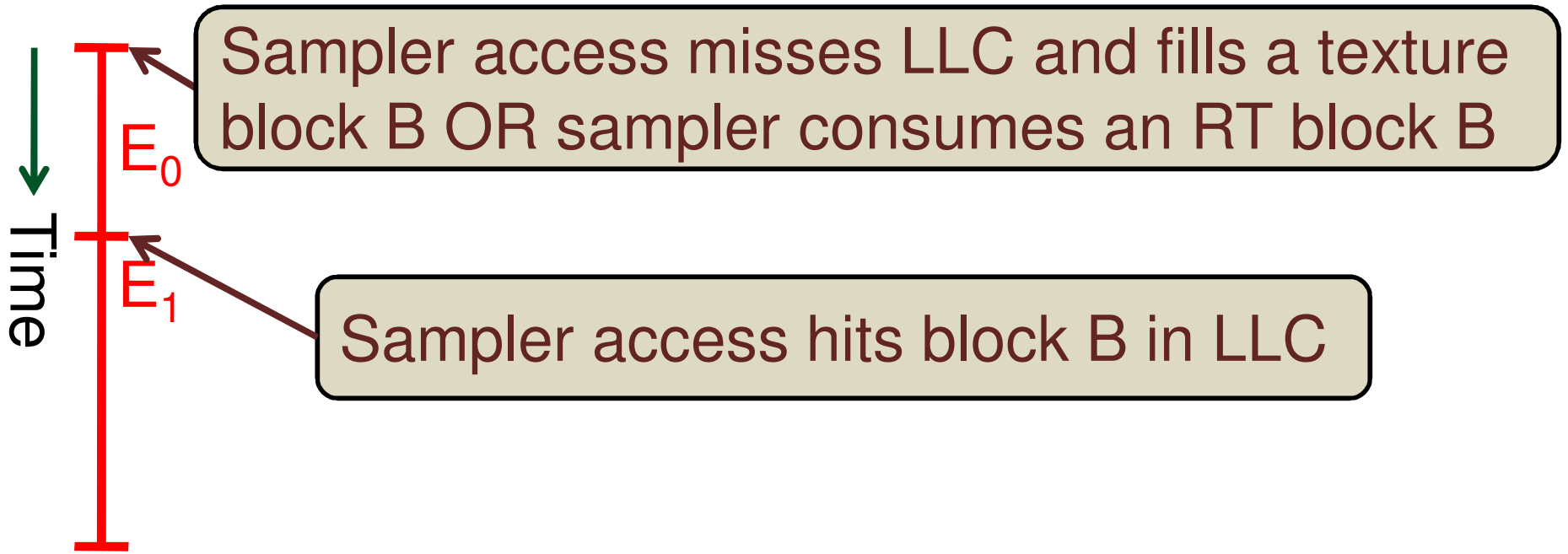
LLC reuse study#1: Texture

- Intra-stream reuses
  - The goal is to understand how a dead texture block in the LLC can be identified and evicted creating room for other live graphics data
  - Divide the life of a texture block in the LLC into epochs demarcated by hits



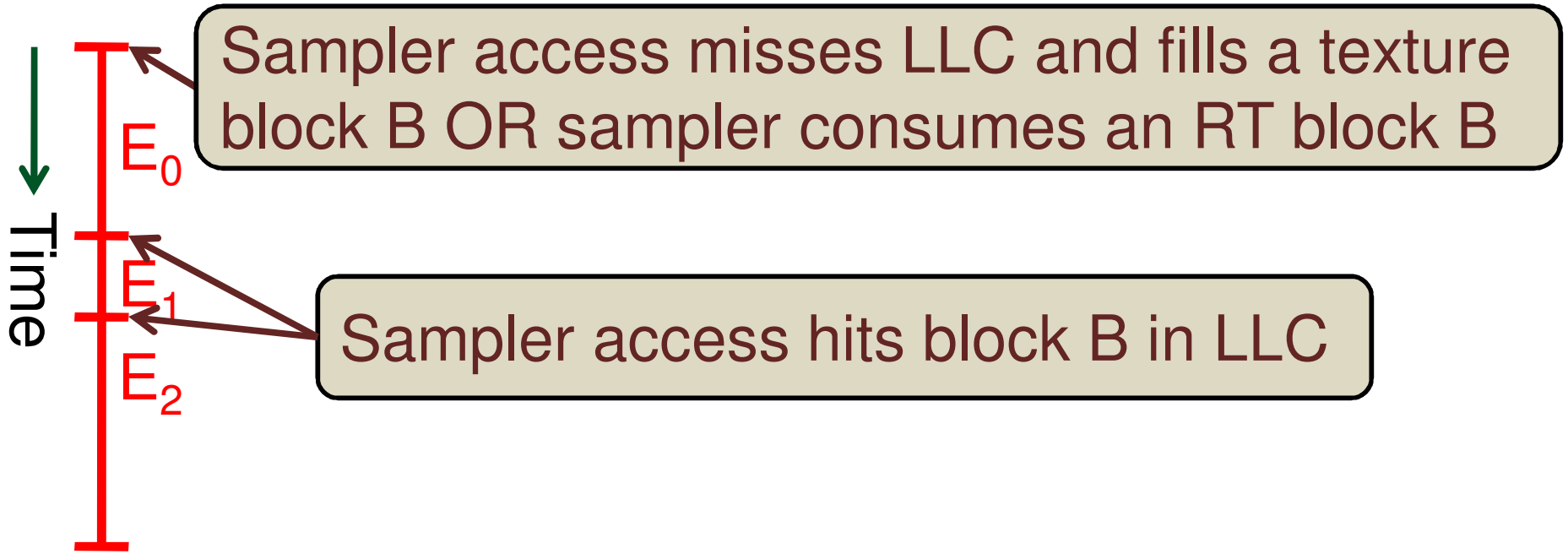
LLC reuse study#1: Texture

- Intra-stream reuses
  - The goal is to understand how a dead texture block in the LLC can be identified and evicted creating room for other live graphics data
  - Divide the life of a texture block in the LLC into epochs demarcated by hits



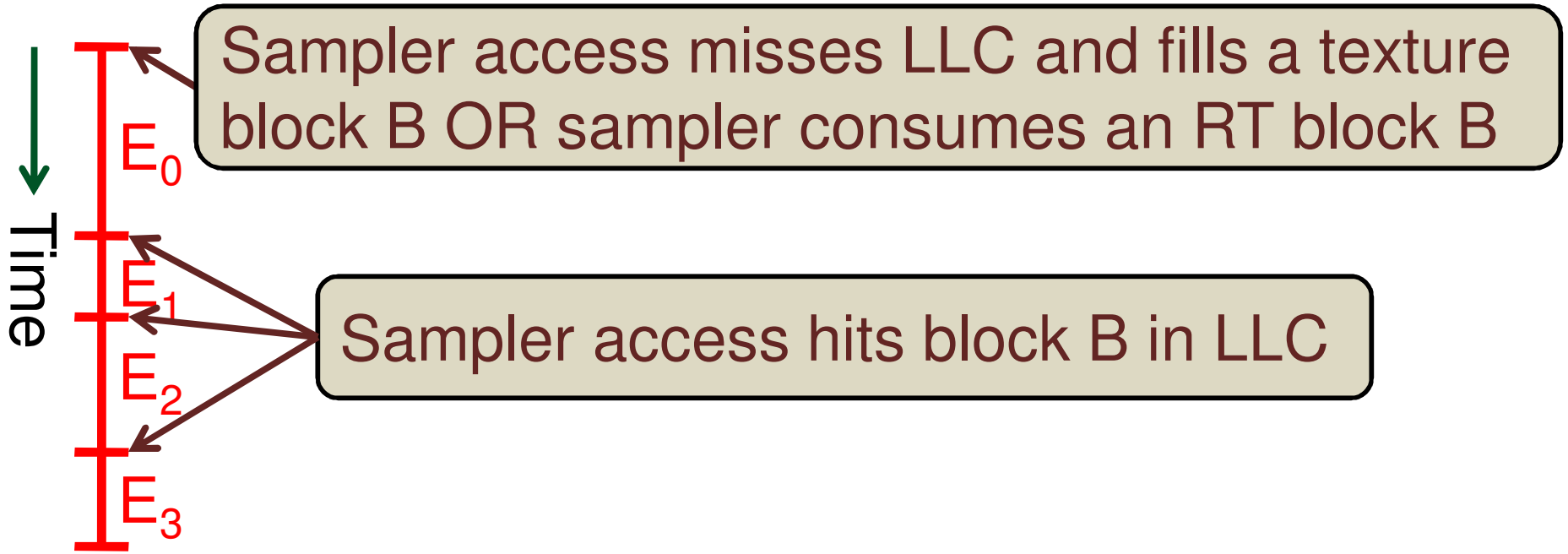
LLC reuse study#1: Texture

- Intra-stream reuses
  - The goal is to understand how a dead texture block in the LLC can be identified and evicted creating room for other live graphics data
  - Divide the life of a texture block in the LLC into epochs demarcated by hits



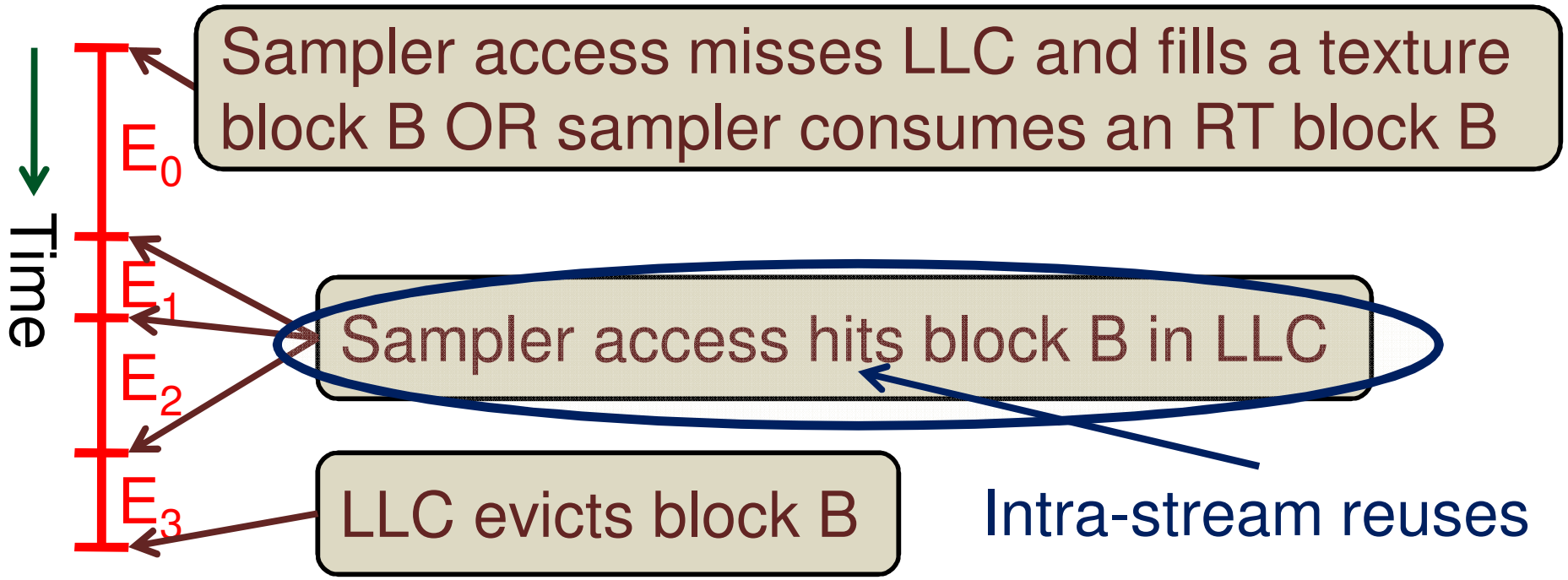
LLC reuse study#1: Texture

- Intra-stream reuses
  - The goal is to understand how a dead texture block in the LLC can be identified and evicted creating room for other live graphics data
  - Divide the life of a texture block in the LLC into epochs demarcated by hits



LLC reuse study#1: Texture

- Intra-stream reuses
  - The goal is to understand how a dead texture block in the LLC can be identified and evicted creating room for other live graphics data
  - Divide the life of a texture block in the LLC into epochs demarcated by hits



LLC reuse study#1: Texture

- Intra-stream reuses
  - All texture blocks residing in the LLC at any point in time can be partitioned into disjoint sets based on their epochs
  - Clearly, the set E_{k+1} is a subset of the set E_k for all $k \geq 0$
  - Define death ratio of epoch E_k as $(|E_k| - |E_{k+1}|)/|E_k|$
  - Define reuse probability of epoch E_k as $|E_{k+1}|/|E_k|$
  - Goal of a good policy should be to attach a high victimization priority to the epochs with low reuse probability

LLC reuse study#1: Texture

- How many epochs are statistically significant
  - When the LLC runs Belady's optimal policy, 79% of all texture sampler hits come from the E_0 epoch, 15% from the E_1 epoch, 4% from the E_2 epoch, and 2% from the $E_{\geq 3}$ epoch
  - It is enough to keep track of the E_0, E_1, and $E_{\geq 2}$ epochs for a texture block
- Average reuse probability of these epochs
  - E_0: 0.19 (at most 0.3 across the twelve apps)
  - E_1: 0.27 (varies a lot across applications: 0.6 to nearly zero)
  - E_2: 0.47 (can be assumed to be mostly live)

LLC reuse study#1: Texture

- Intra-stream reuse: take-away
  - Need to track the epoch membership of a texture block (one among E_0, E_1, $E_{\geq 2}$)
  - Need to learn the reuse probabilities of the E_0 and E_1 epochs dynamically
  - Texture blocks entering the $E_{\geq 2}$ epoch will be assumed to be live unconditionally
- Why DRRIP falls so much short of optimal
  - DRRIP fills slightly over a third of the texture blocks with RRPV three in the LLC
  - Need to eliminate more dead texture blocks
  - Cannot always promote to RRPV zero on hit

LLC reuse study#2: Depth

- Only intra-stream reuses from the LLC
  - Generated depth buffer values are consumed for further depth tests
  - Use the same epoch-based formalism
- Reuse probabilities of the first three epochs
  - E_0: 0.39, E_1: 0.62, E_2: 0.74
  - Very different from the texture epochs
  - Only the E_0 blocks have low reuse probability and the $E_{\geq 1}$ blocks are practically live
  - We will decide the insertion RRPV of the Z blocks by estimating the aggregate reuse probability of all Z blocks and won't consider epochs

LLC reuse: Render target

- Render targets source two types of LLC hits
  - Texture sampler hits for render to texture
  - Render target blending (also known as texture blending), where an already created render target is blended with another render target being created currently (transparency modeling)
- We do not implement any policy for improving render target blending
  - DRRIP is within 10% of optimal in hit rate
  - Some of the lost LLC hits in blending operations can be recovered by eliminating dead textures
  - Our render target hit rates are close to optimal

Sketch

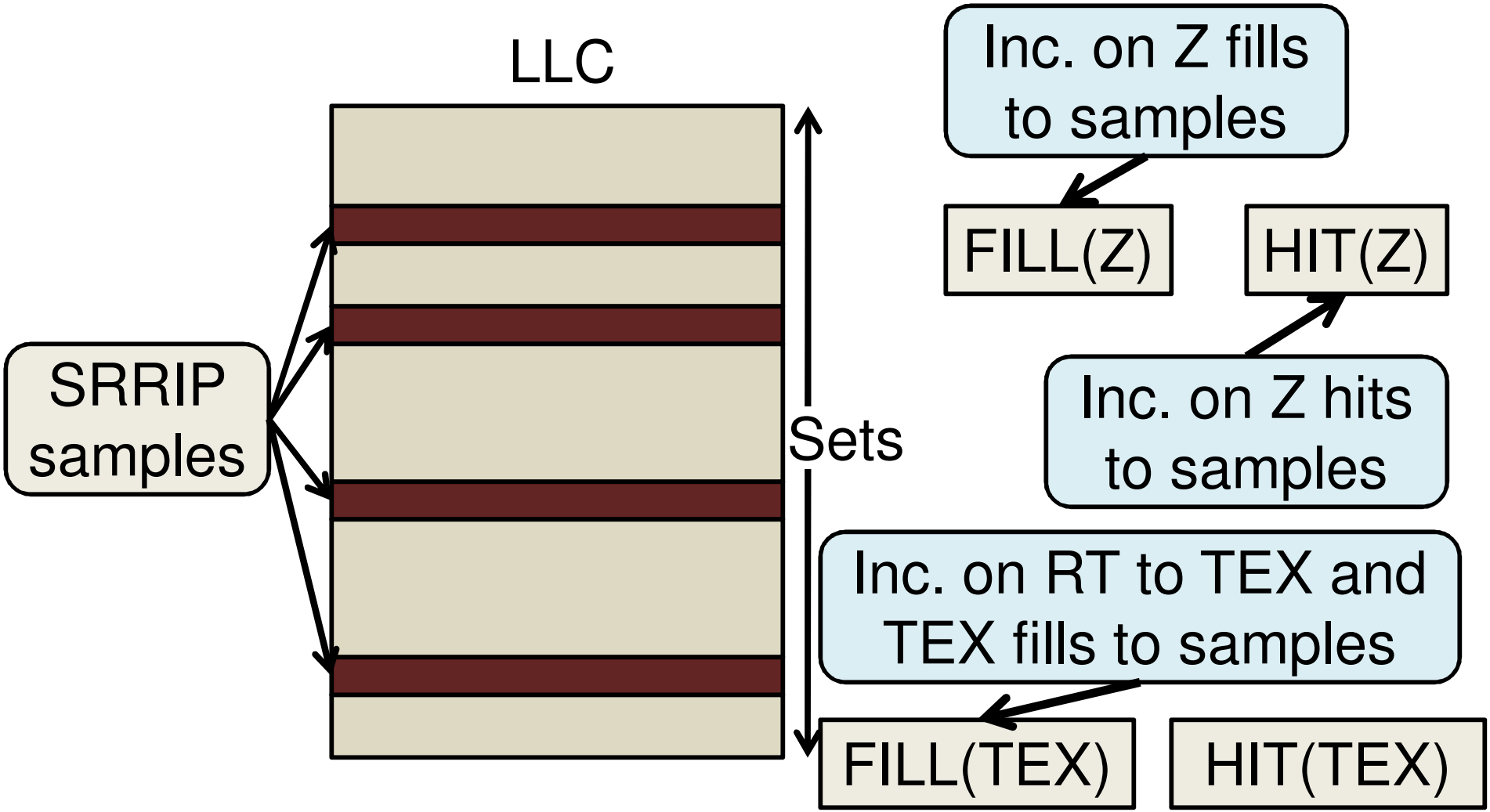
- Talk in one slide
- Result highlights
- Understanding the potential
- Reuses in 3D graphics data
- ➢ Our policy proposals
- Evaluation methodology
- Simulation results
- Summary

Graphics stream-aware policies

- Basic framework
  - LLC accesses are partitioned into four streams based on the source: texture samplers (TEX), render targets (RT), depth (Z), and the rest
  - All policies modulate the two-bit RRPV of an LLC block on insertion and promotion based on reuse probabilities
    - A larger RRPV corresponds to a smaller probability of reuse; the blocks with RRPV three are potential victim candidates
  - The reuse probabilities are estimated by maintaining fill and hit counters for a few sampled LLC sets that always use SRRIP [ISCA'10]

Graphics stream-aware policies

- Policy#1: Graphics stream-aware probabilistic Z and texture caching (GSPZTC)



Graphics stream-aware policies

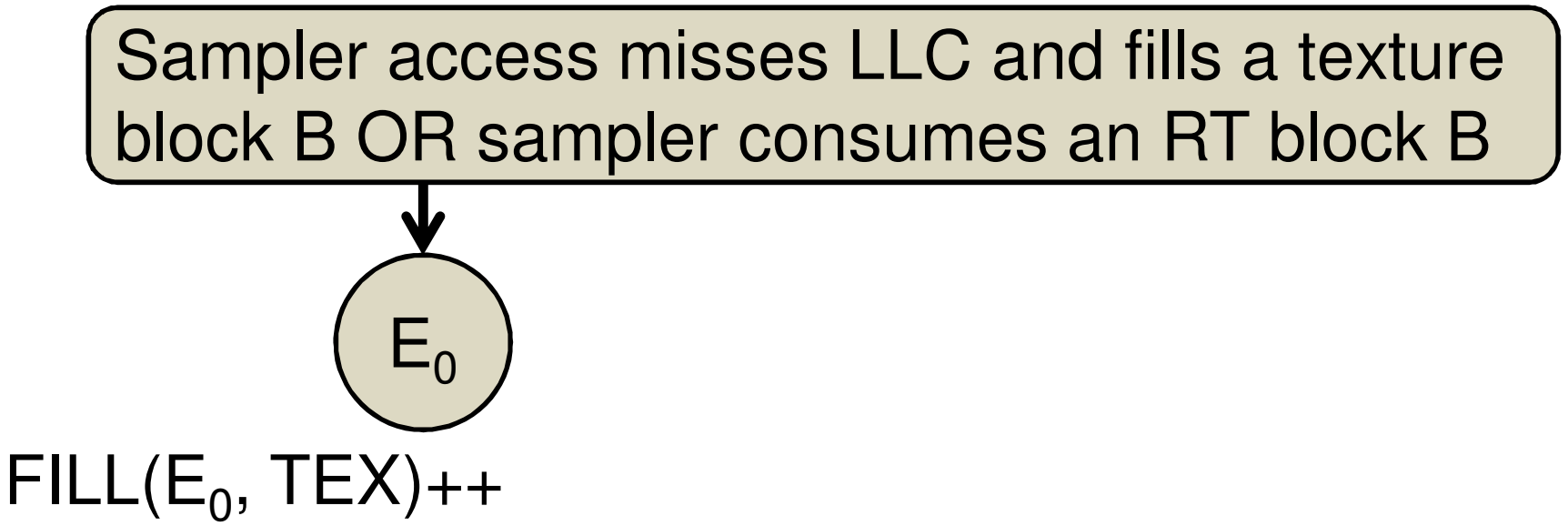
- GSPZTC policy for non-sample sets
  - The insertion RRPV of a block depends on the reuse probability of the stream it belongs to
  - Z fill: RRPV ← (FILL(Z) > t.HIT(Z)) ? 3:2
  - The reuse probability threshold of 1/(t+1) is determined empirically; we use t=8
  - TEX fill: RRPV ← (FILL(TEX) > t.HIT(TEX)) ? 3:0
  - RT fill: RRPV ← 0 (highest protection)
  - All other fills: RRPV ← 2 (like SRRIP)
  - All hits: RRPV ← 0 (like any RRIP)
  - Each LLC block has an RT bit to identify RT to TEX reuse

Graphics stream-aware policies

- Policy#2: GSPZTC with texture sampler epochs (GSPZTC+TSE)
  - Each LLC block has two state bits to keep track of E_0, E_1, and $E_{\geq 2}$ epochs for texture blocks; the fourth state serves the functionality of the RT bit
  - The FILL(TEX) and HIT(TEX) counters are replaced by FILL(E_0, TEX), FILL(E_1, TEX), HIT(E_0, TEX), and HIT(E_1, TEX)
  - Recall: enough to estimate the reuse probabilities of the E_0 and E_1 epochs
  - Recall: the $E_{\geq 2}$ blocks are unconditionally live (assumed to have high reuse probability)

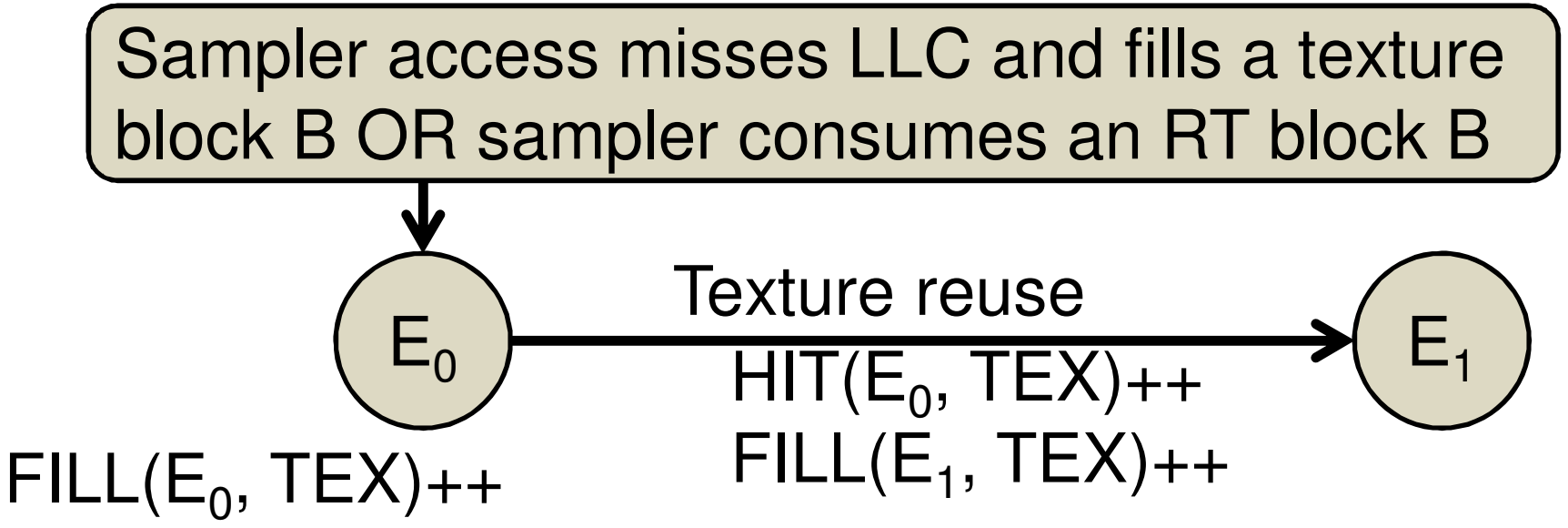
Graphics stream-aware policies

- Policy#2: GSPZTC with texture sampler epochs (GSPZTC+TSE)



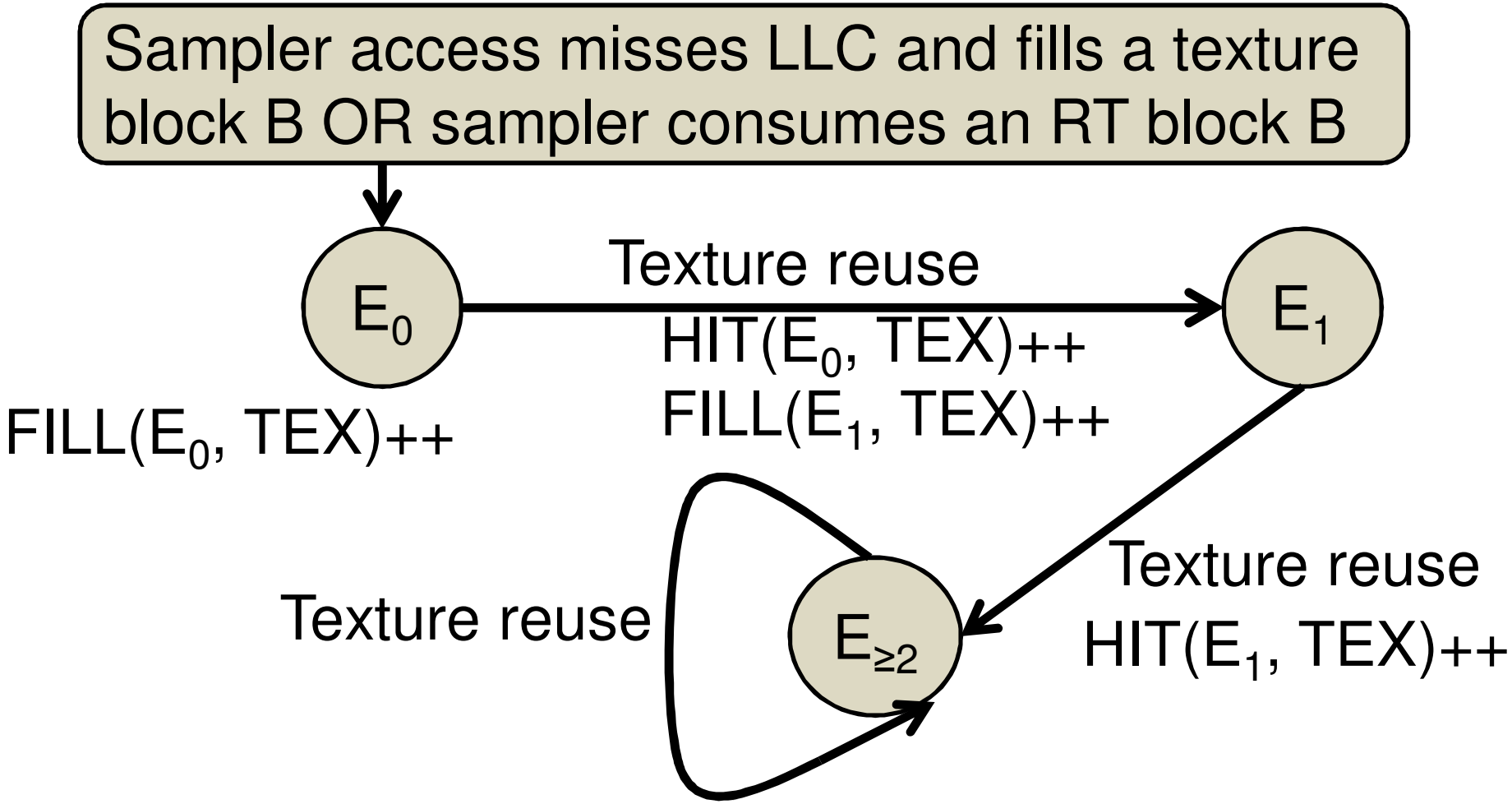
Graphics stream-aware policies

- Policy#2: GSPZTC with texture sampler epochs (GSPZTC+TSE)



Graphics stream-aware policies

- Policy#2: GSPZTC with texture sampler epochs (GSPZTC+TSE)



Graphics stream-aware policies

- Policy#2: GSPZTC+TSE for non-sample sets
  - TEX fill or RT to TEX reuse:
    RRPV $\leftarrow$ (FILL(E_0, TEX) > t.HIT(E_0, TEX)) ? 3:0
  - TEX hit to a block in epoch E_0:
    RRPV $\leftarrow$ (FILL(E_1, TEX) > t.HIT(E_1, TEX)) ? 3:0
  - TEX hit to a block in epoch E_1: RRPV $\leftarrow$ 0
  - Other rules are same as GSPZTC
  - Observe that both GSPZTC and GSPZTC+TSE offer the highest protection to the newly filled render target blocks
    - Unnecessarily wastes cache space if the likelihood of RT to TEX reuse is low
    - The next policy addresses this problem

Graphics stream-aware policies

- Policy#3: GSPZTC+TSE + RT insertion policy
  - Our final proposal: graphics stream-aware probabilistic caching (GSPC)
  - Incorporates two new counters PROD and CONS
  - PROD is incremented on an RT fill to a sample set; left untouched on RT blending hits
    - Approximately tracks the number of unique RT blocks mapping to the sample sets
  - CONS is incremented on RT to TEX reuses in the sample sets
    - One increment for every consumed RT block
    - The block enters the E_0 state after this
  - Inter-stream reuse probability is CONS/PROD

Graphics stream-aware policies

- Policy#3: GSPC for non-sample sets
  - RT fill: If PROD > 16.CONS then RRPV ← 3

    [[Low inter-stream reuse probability]]

    Else if 16.CONS ≥ PROD > 8.CONS then

    RRPV ← 2

    [[Medium inter-stream reuse probability]]

    Else RRPV ← 0

    [[High inter-stream reuse probability]]
  - RT hit (blending): RRPV ← 0
  - RT to TEX reuse: as in GSPZTC+TSE
  - All other rules are same as GSPZTC+TSE

Graphics stream-aware policies

- Hardware overhead of GSPC on top of two RRPV bits per LLC block
  - Two new state bits per LLC block
  - Eight short counters per LLC bank: reuse probabilities are de-centralized and maintained per bank to avoid counter hotspots
    - HIT(Z), FILL(Z), HIT(E_0, TEX), FILL(E_0, TEX), HIT(E_1, TEX), FILL(E_1, TEX), PROD, CONS: eight bits each
    - A seven-bit counter to maintain the interval at which the above counters are halved: probabilities are computed on exponentially averaged estimates
  - Overall, less than 0.5% of all LLC data bits

Sketch

- Talk in one slide
- Result highlights
- Understanding the potential
- Reuses in 3D graphics data
- Our policy proposals
- ➢ Evaluation methodology
- Simulation results
- Summary

Evaluation methodology

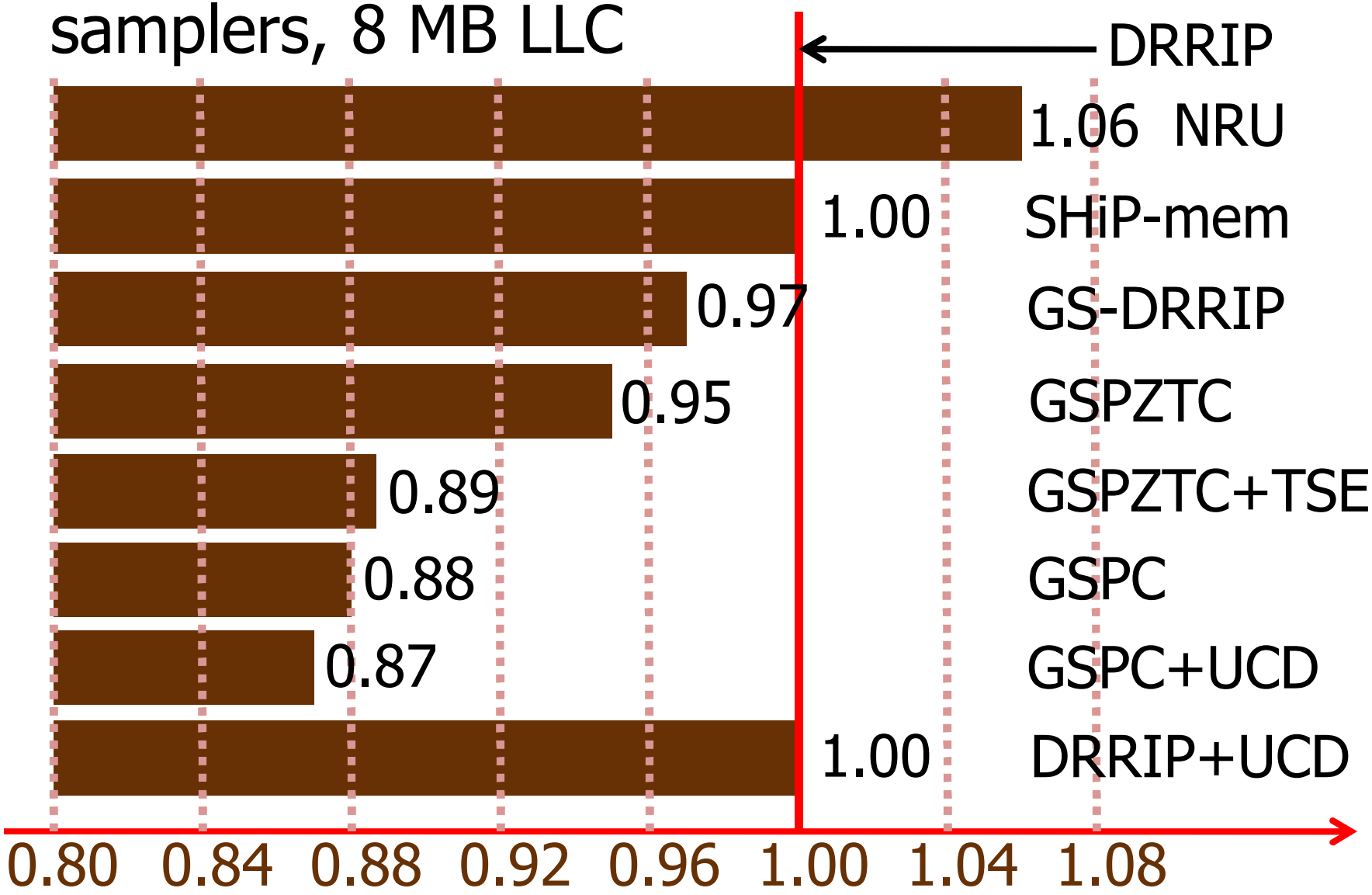
- Detailed timing model of a high-end GPU
  - Eight thread contexts per shader core
    - Two threads can issue one four-wide vector operation (including MAD) each per cycle
  - One texture sampler for every eight shader cores
  - Two configs: 64 and 96 shader cores @ 1.6 GHz
    - Peak shader throughput: 1.6 TFLOPS and 2.5 TFLOPS
    - 512 and 768 thread contexts
  - LLC configs: 8 MB and 16 MB 16-way 4 GHz
    - 2 MB per bank, non-inclusive/non-exclusive
  - DRAM configs: Dual-channel DDR3-1600 15-15-15 8-way banked
- 52 frames from twelve DirectX applications

Sketch

- Talk in one slide
- Result highlights
- Understanding the potential
- Reuses in 3D graphics data
- Our policy proposals
- Evaluation methodology
- ➢ Simulation results
- Summary

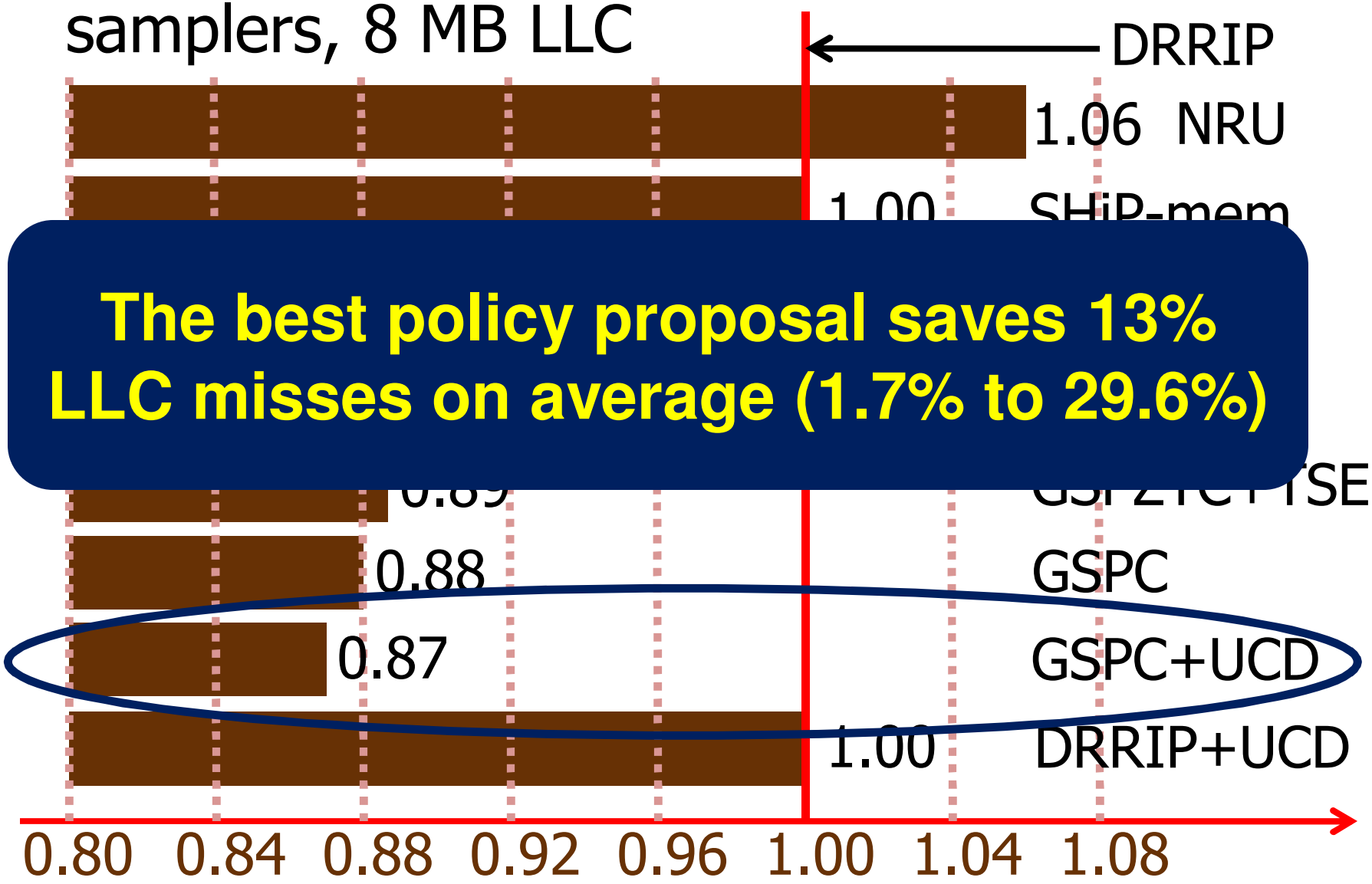
Total volume of LLC misses

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC



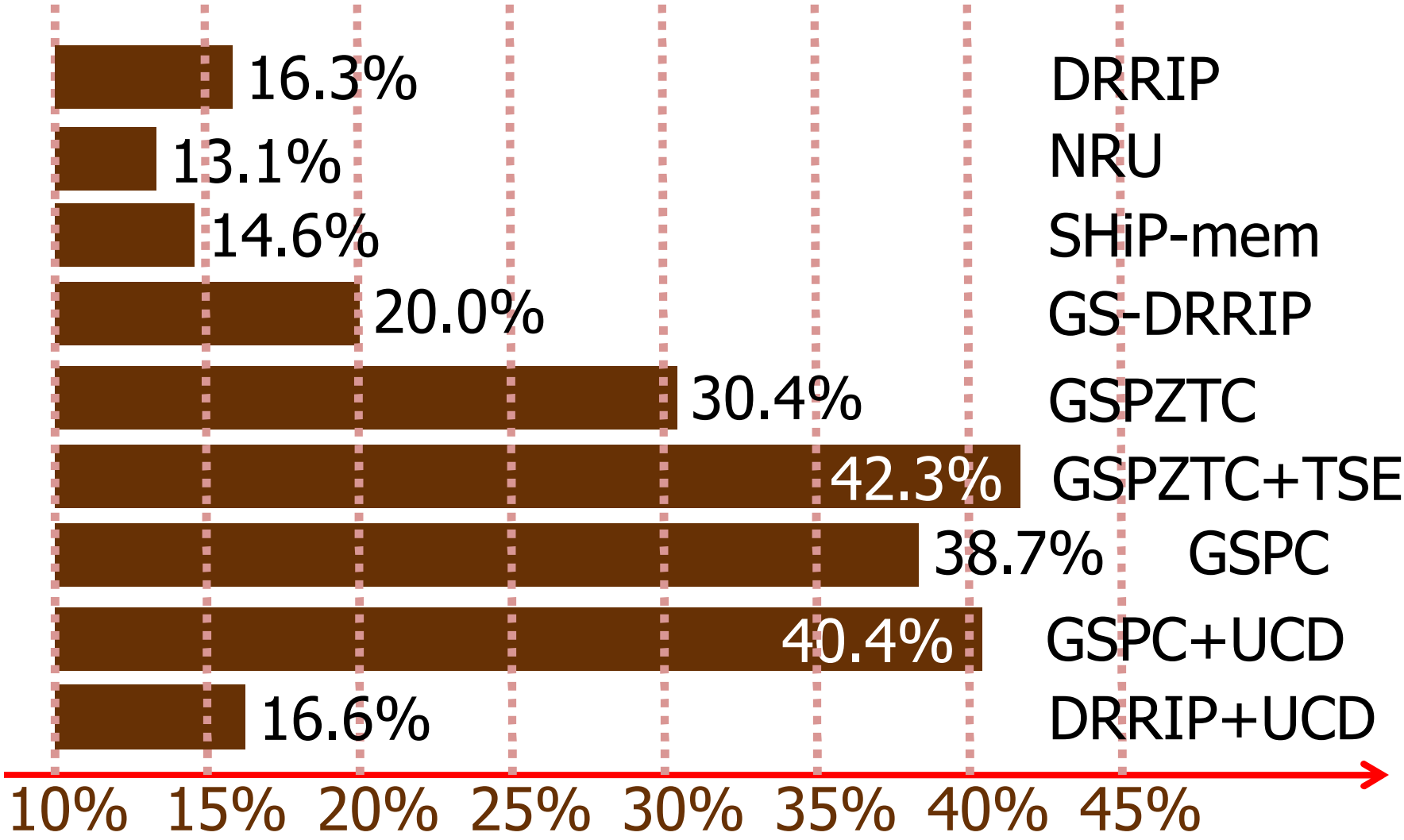
Total volume of LLC misses

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC



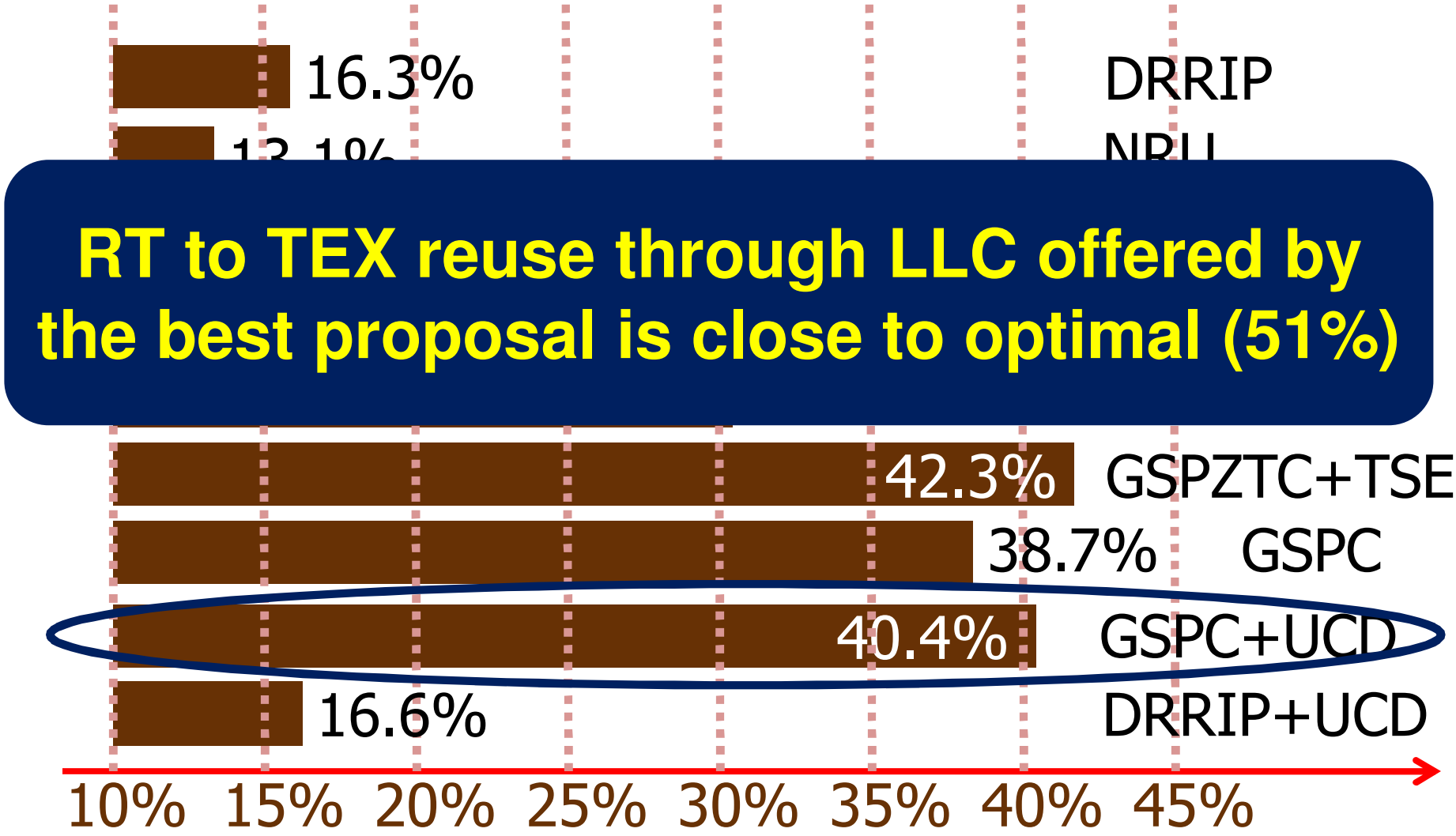
RT to TEX reuse through LLC

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC



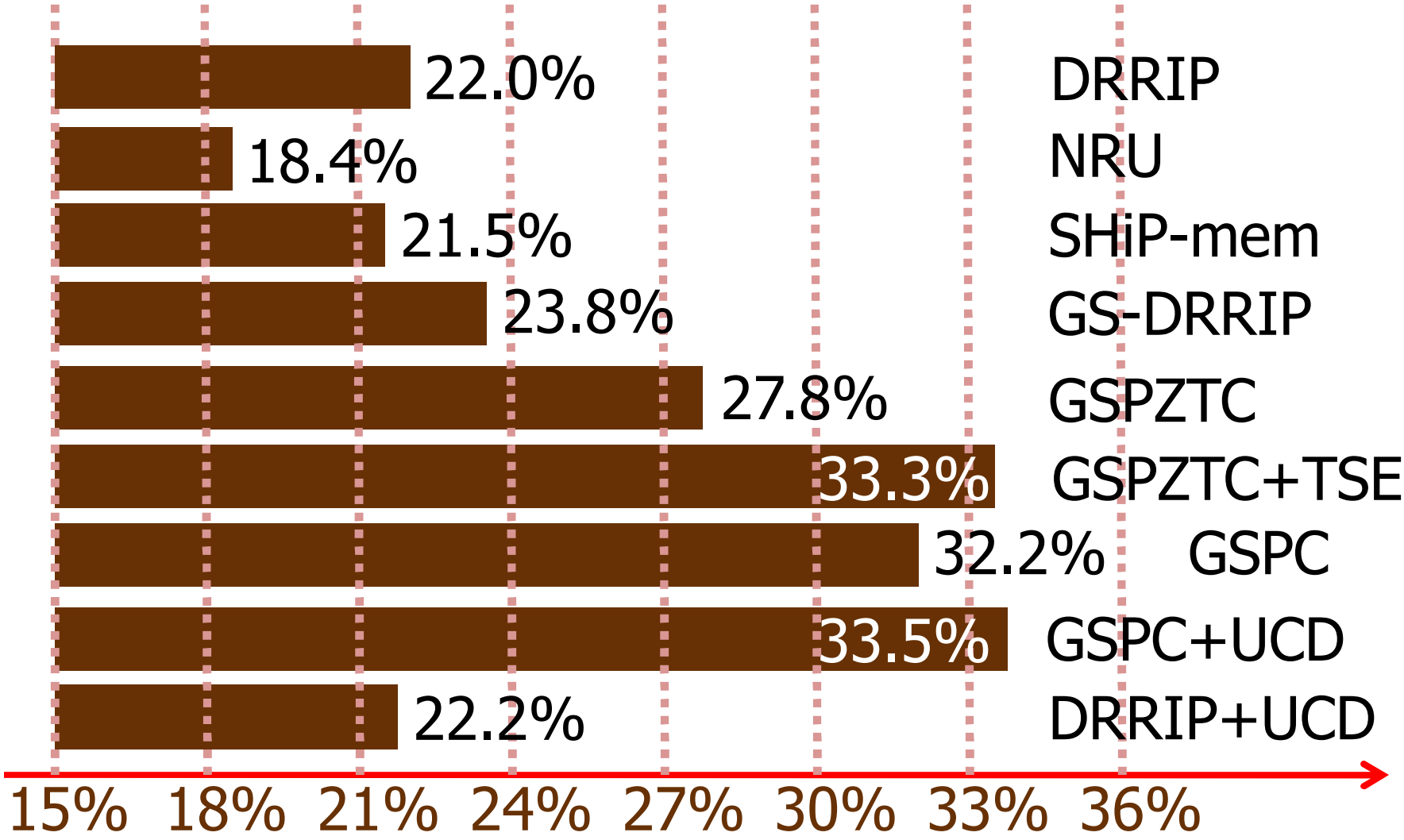
RT to TEX reuse through LLC

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC



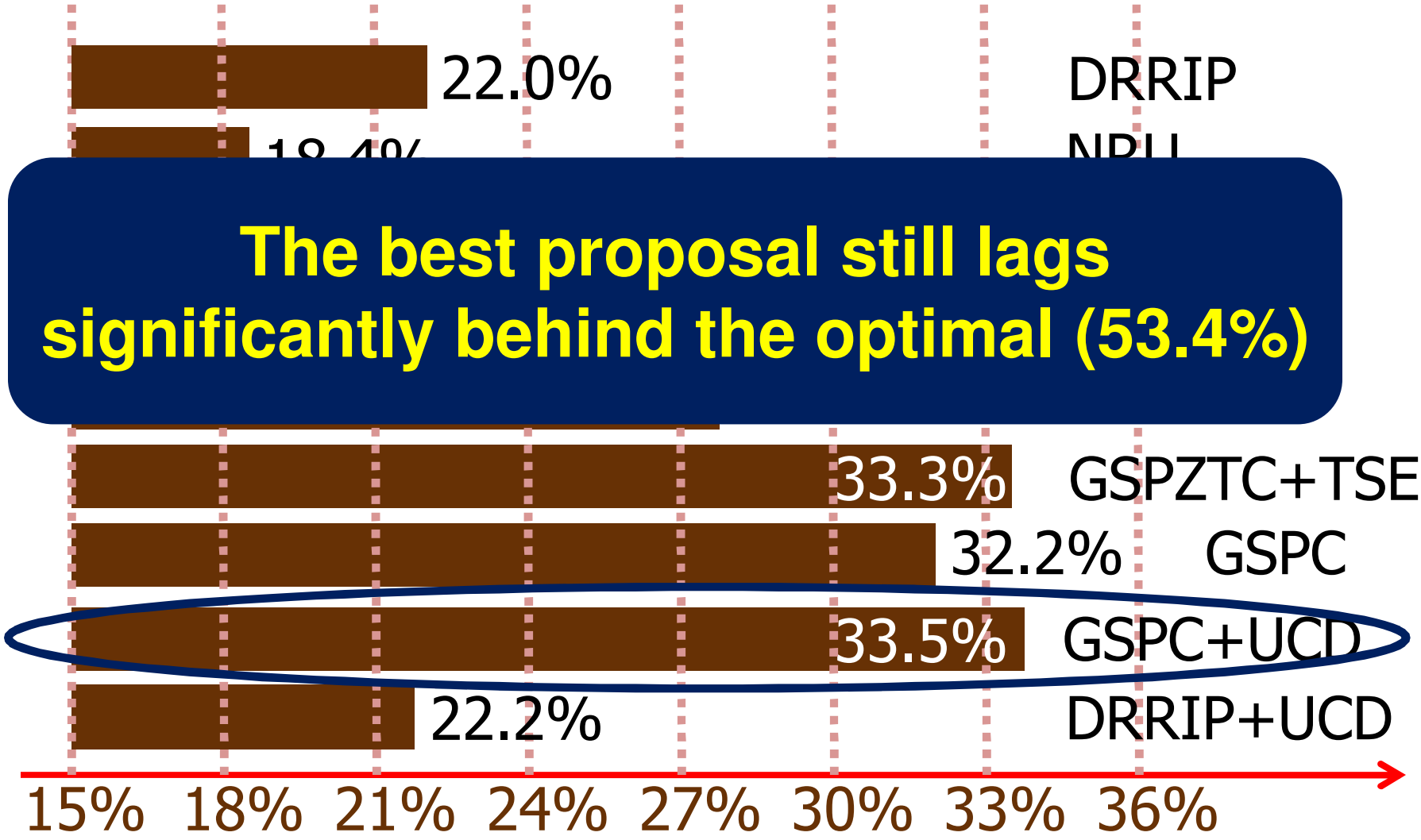
LLC hit rate: Texture sampler

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC



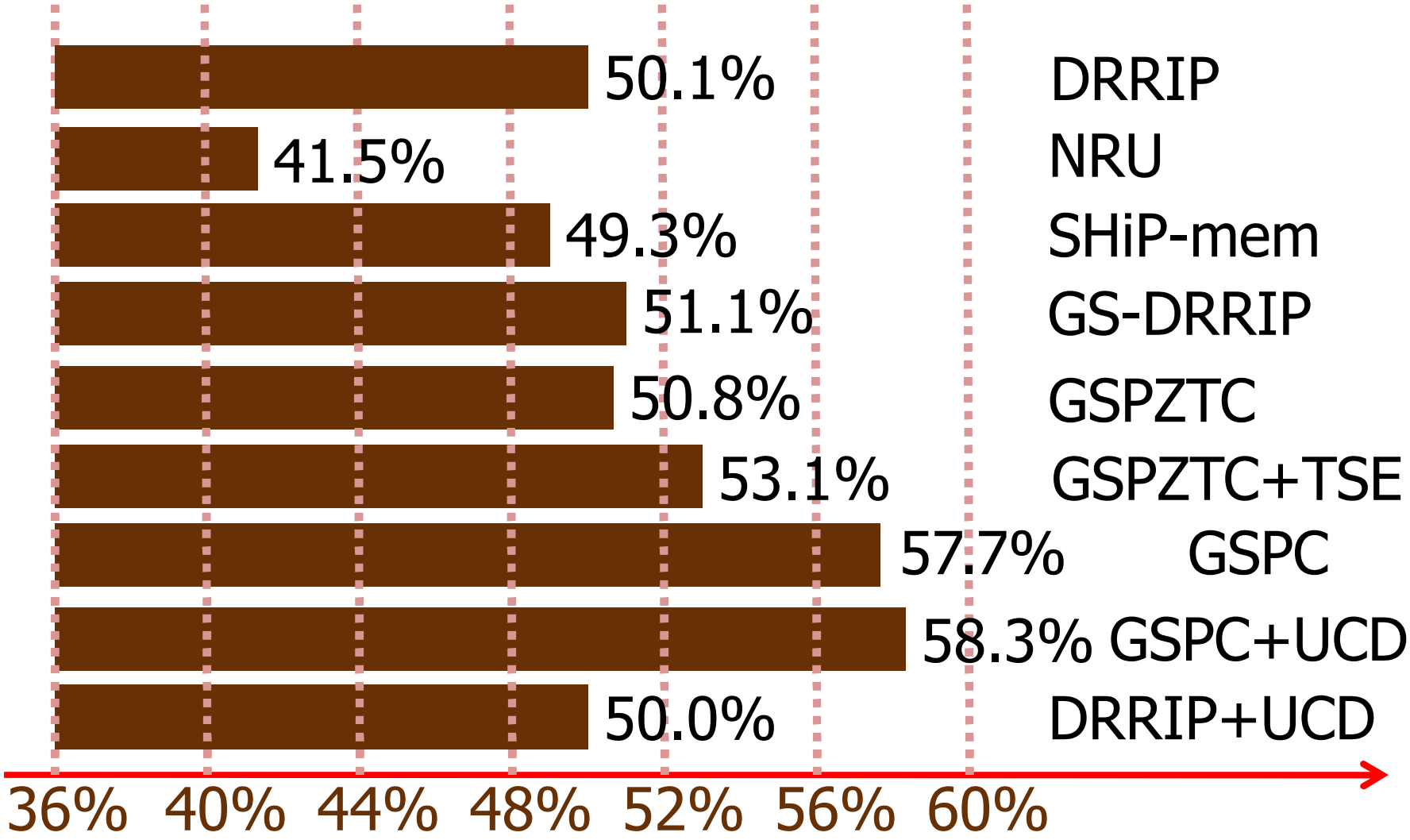
LLC hit rate: Texture sampler

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC



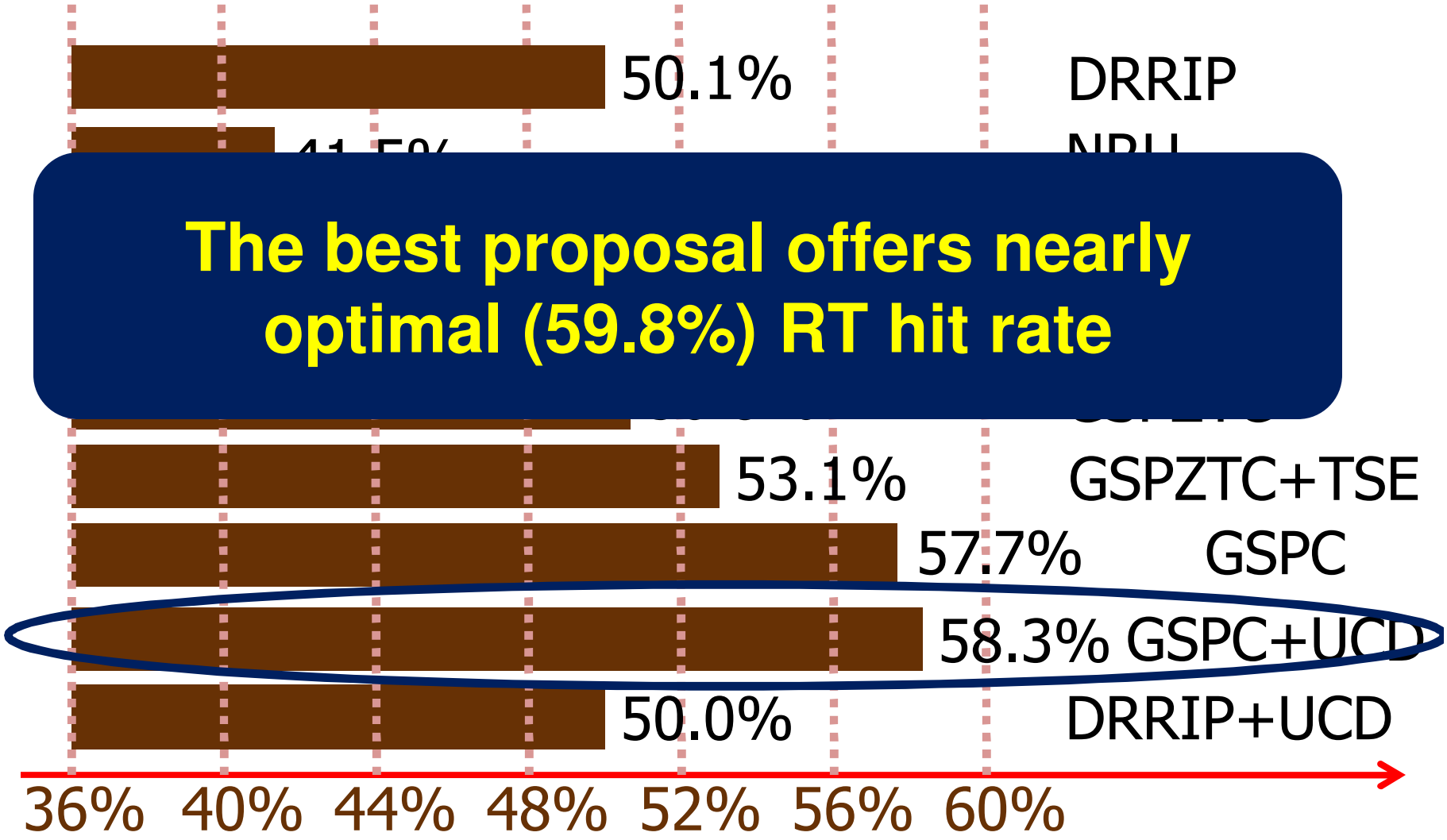
LLC hit rate: RT accesses

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC



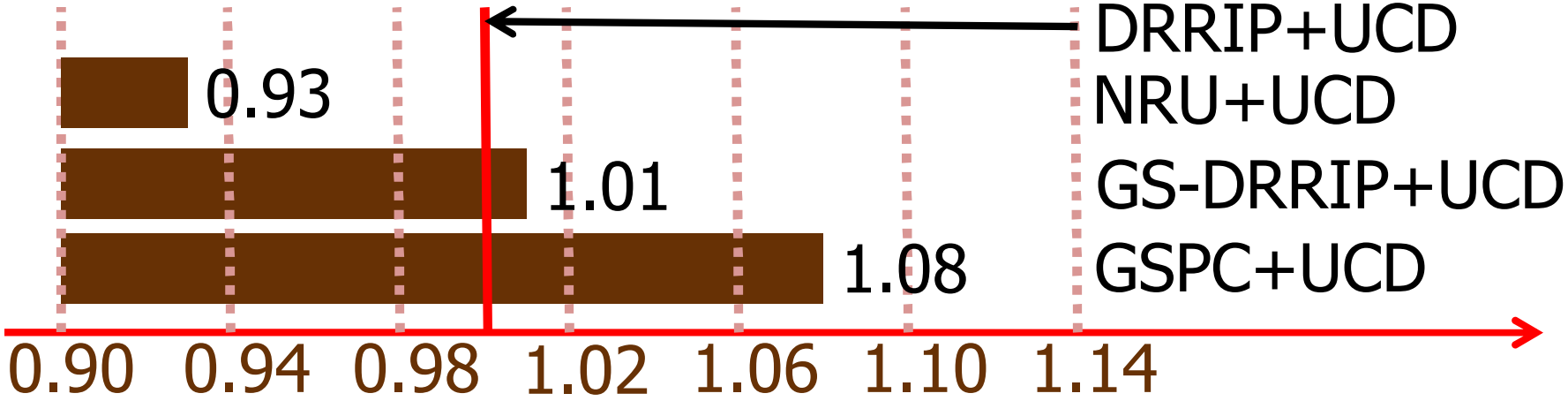
LLC hit rate: RT accesses

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC

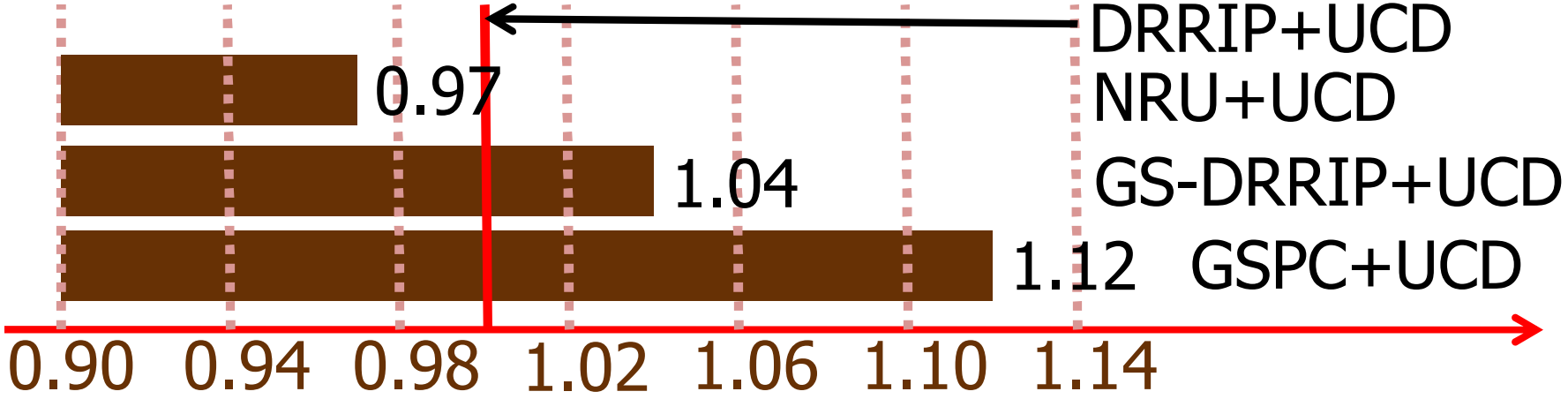


Frame rate improvement

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC, DDR3-1600 15-15-15

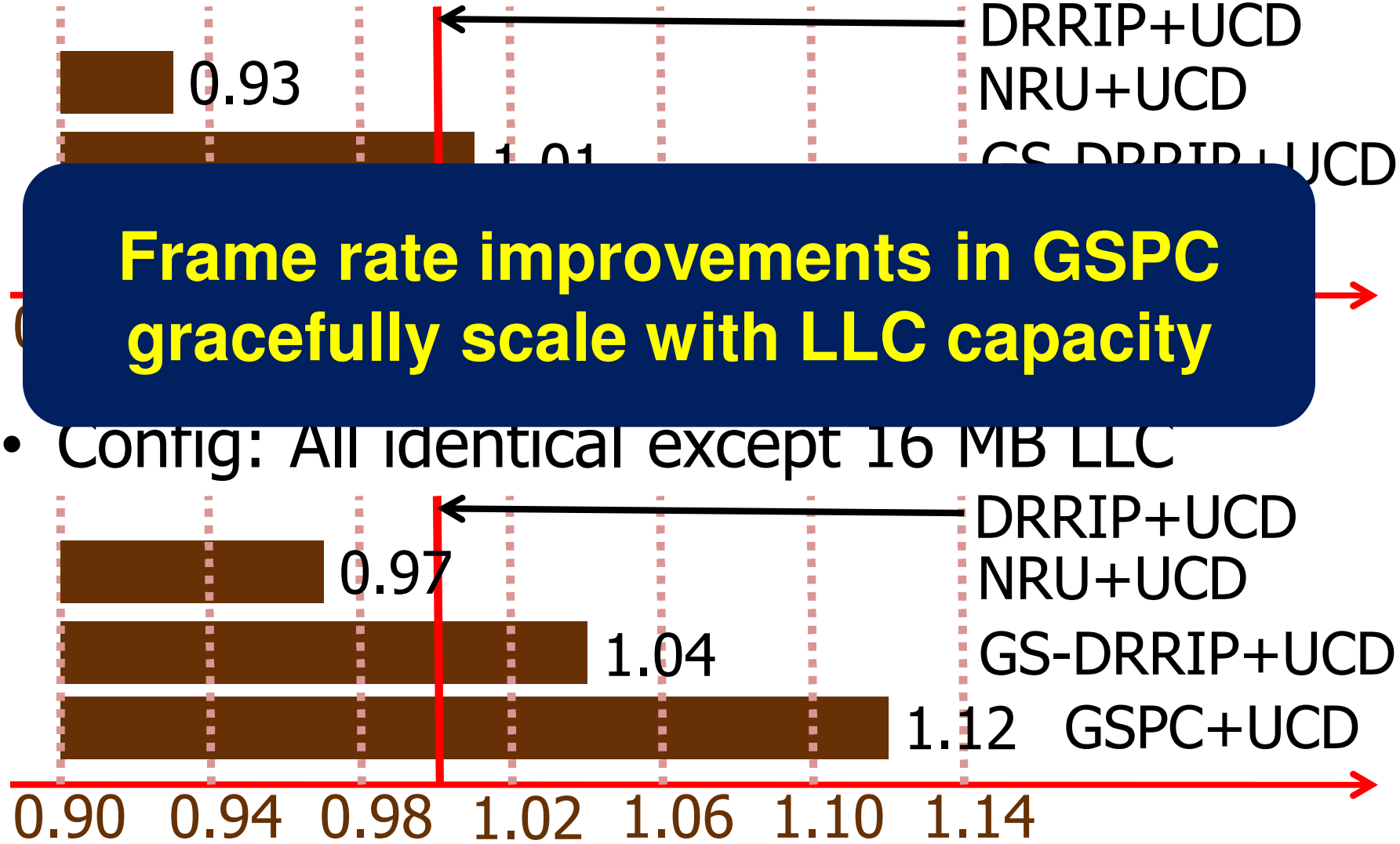


- Config: All identical except 16 MB LLC



Frame rate improvement

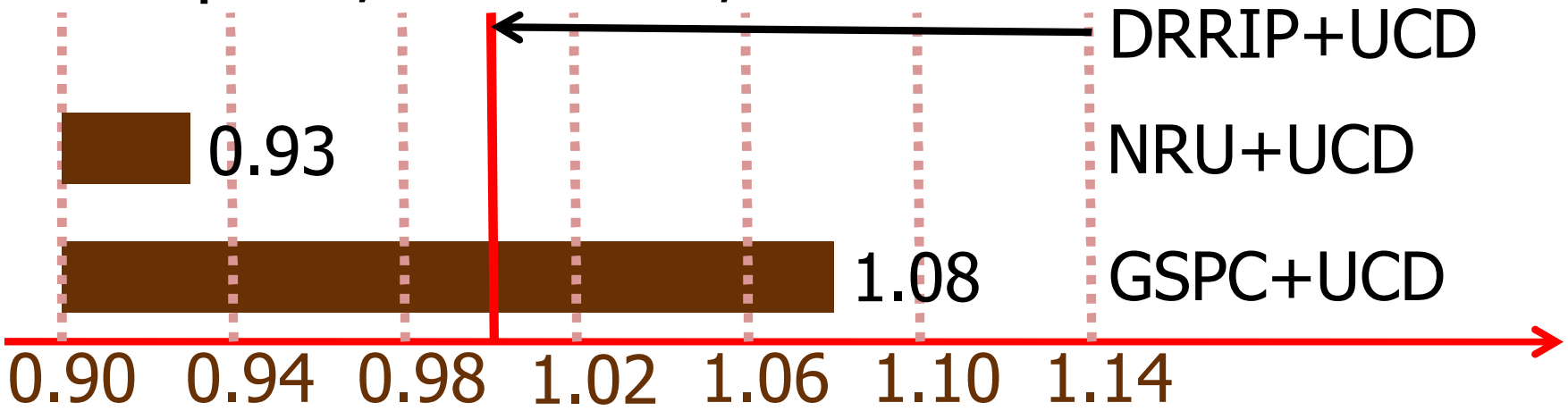
- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC, DDR3-1600 15-15-15



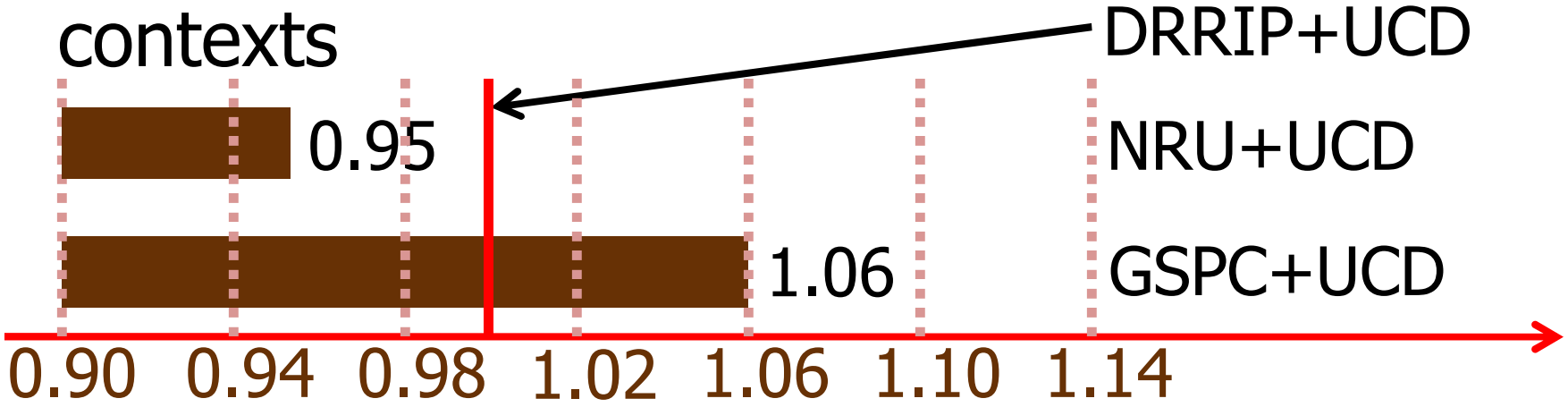
- Config: All identical except 16 MB LLC

Sensitivity to shader thread count

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC, DDR3-1600 15-15-15

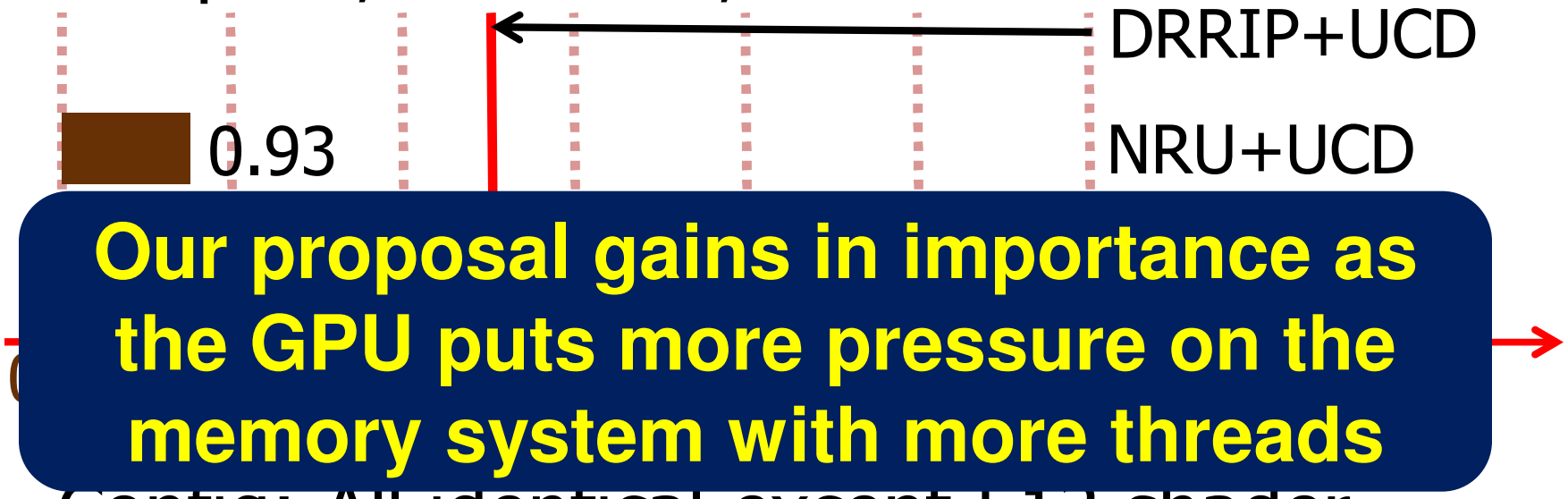


- Config: All identical except 512 shader contexts



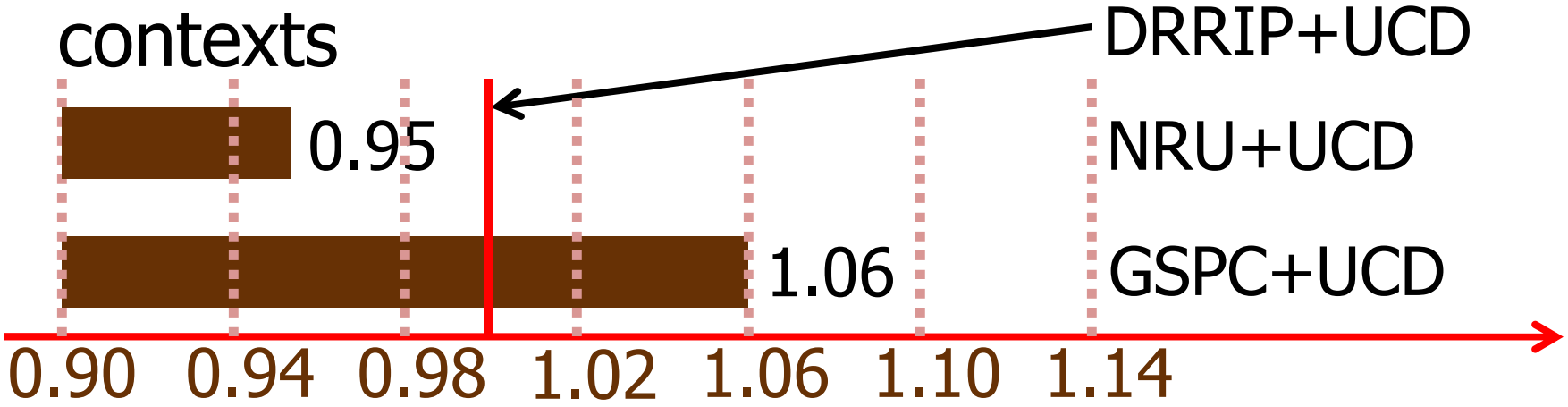
Sensitivity to shader thread count

- Config: 768 shader contexts, 12 texture samplers, 8 MB LLC, DDR3-1600 15-15-15



Our proposal gains in importance as the GPU puts more pressure on the memory system with more threads

- Config: All identical except 512 shader contexts



Sketch

- Talk in one slide
- Result highlights
- Understanding the potential
- Reuses in 3D graphics data
- Our policy proposals
- Evaluation methodology
- Simulation results

➢ Summary

Summary

- Graphics processor's LLC is shared by different data structures used in 3D scene rendering applications
- Render targets (same as pixel colors), textures, and depth buffer contribute most to the LLC access traffic
- We propose reuse probability-based algorithms to efficiently manage the GPU LLC
- Our best proposal saves 13.1% LLC misses and speeds up rendering by 8% on average in a GPU with an 8 MB LLC
- Speedup improves to 11.8% for a 16 MB LLC

Where do we lose against optimal

- Render target to texture reuse
  - Optimal: 51%, GSPC+UCD: 40.4%

About 10%

- Render target blending hit rate
  - Optimal: 59.8%, GSPC+UCD: 58.1%

About 2%

- Texture sampler hit rate
  - Optimal: 53.4%, GSPC+UCD: 33.5%

About 20%

- Z hit rate
  - Optimal: 77.1%, GSPC+UCD: 59%

About 18%

- Need a better model for intra-stream texture and Z reuses that can construct partitions more useful than reuse count-based epochs

Nothing clears up a case so much
as stating it to another person.

-Sir Arthur Conan Doyle
[*Silver Blaze* (1892)]

Thank you