Techniques for Caches in GPUs

Günther Schindler
Seminar Talk 2015/16
Chair ASC

Outline

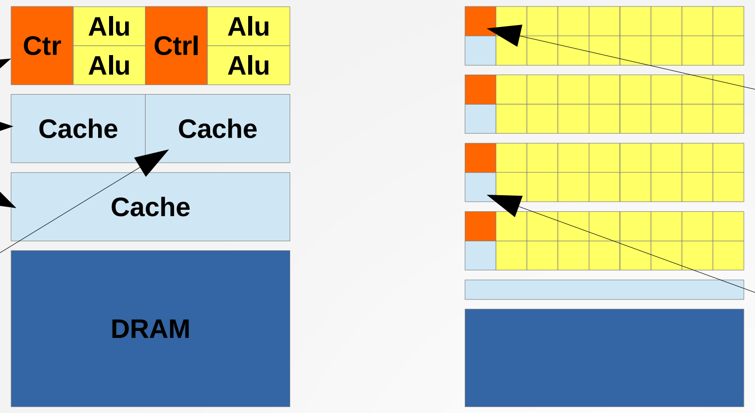
GPU vs. CPU

CPU
„Latency-oriented“

GPU
„Throughput-oriented“

Score performance via out-of-order processing and large Caches.

16 KB L1$/Thread
(Intel Haswell)



Low-overhead thread scheduling and hide memory latencies via multi-threading.

24 B L1$/Thread
(Worst Case:
8 blocks per SM
Nvidia Kepler)

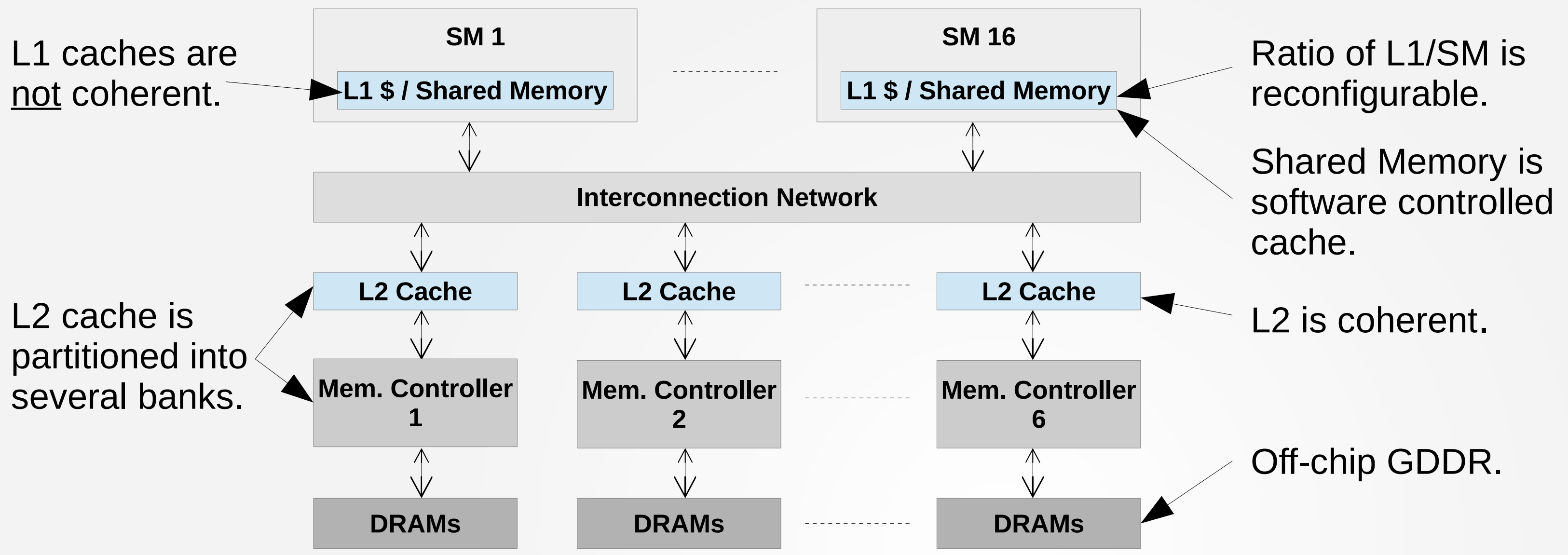
Unit	Intel i7-4770 (Haswell) [1]	Intel i7-6700 (Skylake) [1]	Tesla GT200 [2]	Fermi GF106 [2]	Kepler GK104 [2]	Maxwell GM107 [2]
L1 D$ (cycles)	4-5	4-5	X	45	30	X
L2 D$ (cycles)	12	12	X	310	175	194
L3 D$ (cycles)	36	42	X	X	X	X
SMem (cycles)	X	X	38	50	33	28
RAM (cycles)	36 + 57ns	36 + 57ns	440	685	300	350
L1 D$ Size	32 KB	32 KB	X	48 KB	48 KB	24 KB
L2 size	256 KB	256 KB	X	768 KB	1536 KB	2048 KB
L3 size	8 MB	8 MB	X	X	X	X

GPU chips spend more die-space on ALUs and less on caches.

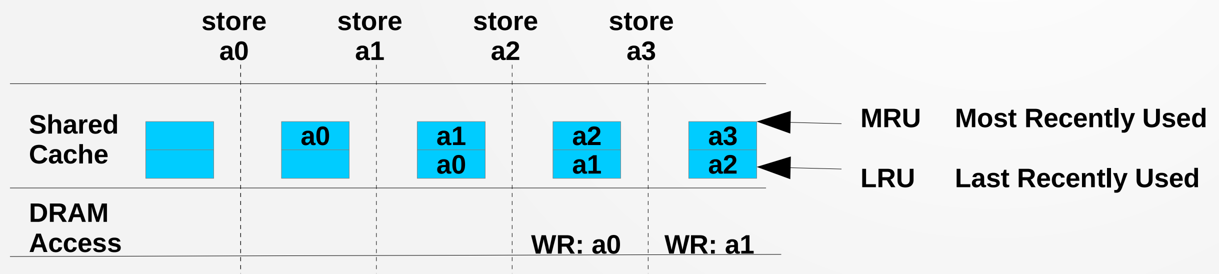
(1) http://www.7-cpu.com
(2) Michael Andersch, Jan Lucas, Mauricio Alvarez-Mesa, Ben Juurlink, “Analyzing GPGPU Pipeline Latency”, Poter 2014.

GPU Architecture

Memory Model



Last Recently Used (LRU) Policy



Caches in GPUs

Motivation

- Caches improve the performance of **atomic operations**.
- Shared cache in **CPU-GPU heterogeneous processors** improve communication and save die space.
- Improves **inter-block communication**.
- Avoiding **off-chip accesses** and increasing **bandwidth and save energy**.

Limitations of existing cache management techniques

- Improvement in cache performance does not directly translate into improved program performance (due to multi-threading).
- Unique GPU characteristics.
- Small cache size.
- Negative effect of caches on performance.

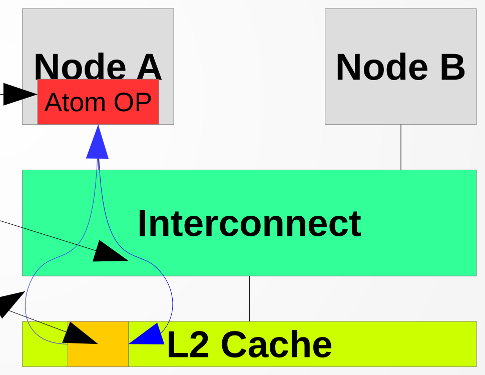
Atomic Operations

Motivation

- Slow atomic operations currently limit applicability.
- CPU atomic mechanisms require L1 coherence.
- Need cost-effective adaptation to improve atomics.
- Franey et al. (0) restrict coherence to atomic data and implemented a complexity-effective coherence mechanism.

State-of-the-art

- Executed like non-atomic instructions in the shader core.
- Traverse the interconnect to the appropriate L2 bank.
- Operation is ordered, data is acquired, and the operation is performed.
- Response is sent back to the core containing the previous value of the data.

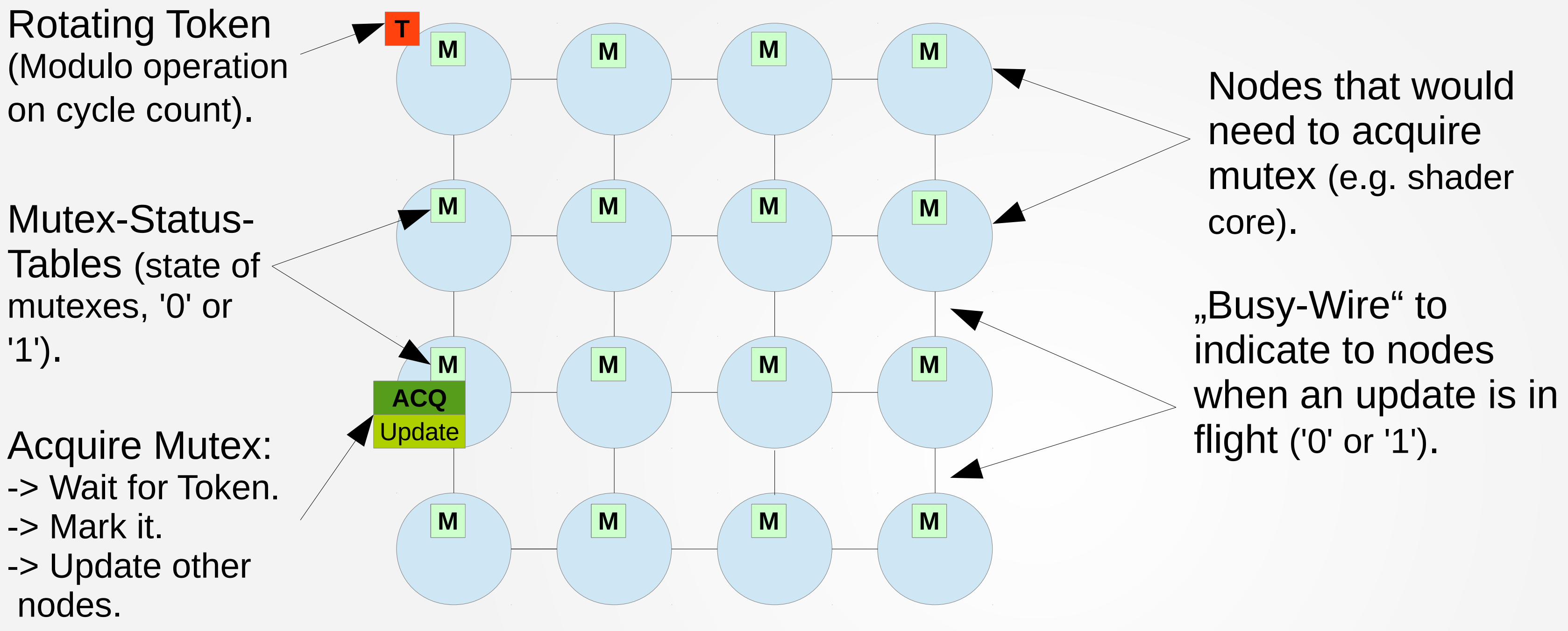


Goal: Avoiding the latency of traversing the interconnect (atomic operations must be performed locally).

(0) S. Franey and M. Lipasti, "Accelerating atomic operations on GPGPUs," in Seventh IEEE/ACM International Symposium on Networks on Chip (NoCS), 2013, pp. 1–8.

AtomNaive

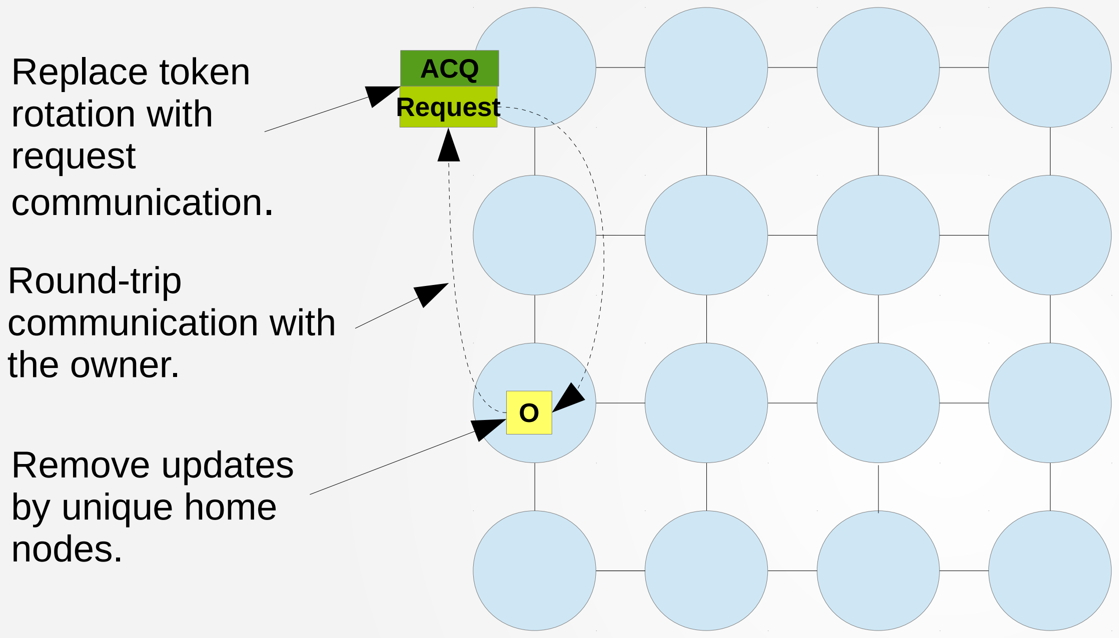
Approach: Restrict coherence to atomic data with Mutex.


Rotating Token (Modulo operation on cycle count).
T
M
Mutex-Status-Tables (state of mutexes, '0' or '1').
ACQ
Update
Acquire Mutex:
-> Wait for Token.
-> Mark it.
-> Update other nodes.
Nodes that would need to acquire mutex (e.g. shader core).
„Busy-Wire“ to indicate to nodes when an update is in flight ('0' or '1').

+ Ensures acquisition correctness
- Long latency to acquire token
- Additional latency for updates

AtomDir

Approach: Adapting techniques used in directory-based cache coherence.


Replace token rotation with request communication.
Round-trip communication with the owner.
Remove updates by unique home nodes.
ACQ
Request
O

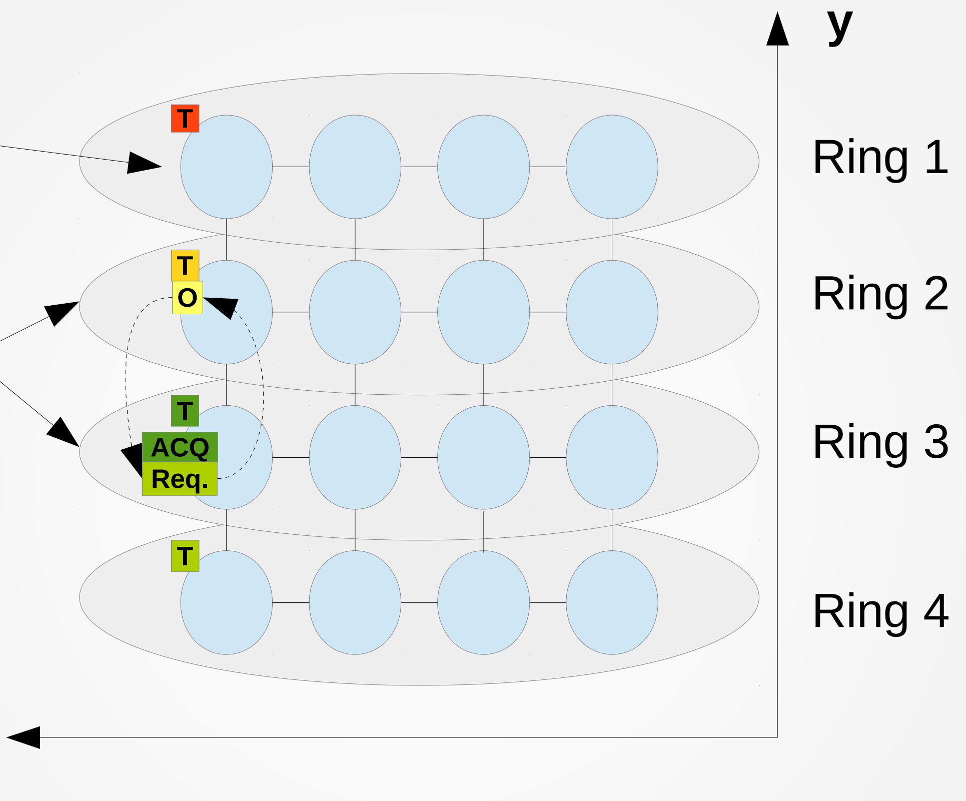
+ Ensures acquisition correctness
- Round-trip latency
- Minimal performance improvement

Hybrid Topology

Approach: Effectively finding a middle point between the AtomNaive and AtomDir configurations.

AtomNaive: Replicated mutex status tables with „Busy-Wire“ and Token (update communication).

AtomDir: Mutex state is distributed across some number of logical rings (request communication).



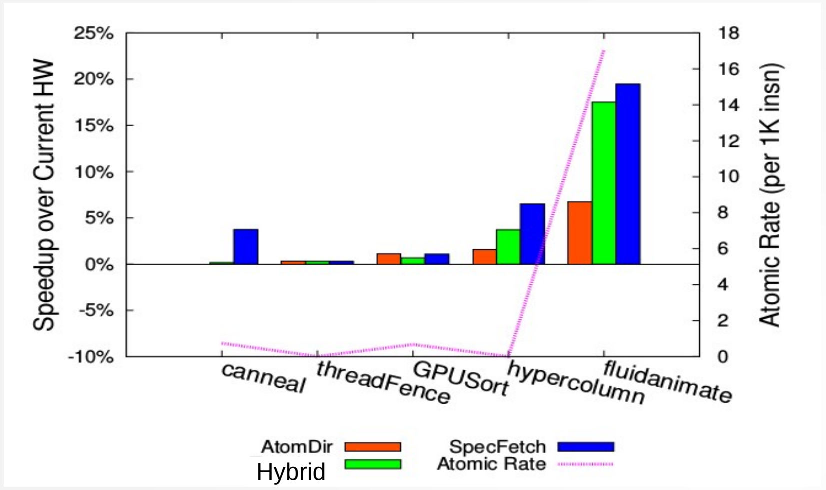
AtomDir: (round-trip)	Δx + Δy latency
AtomNaive: (one-way trip)	Δx/2 latency
Hybrid:	Δx/2 + Δy latency

- Mutex acquisition delays fetch
+ Issue fetch in parallel with mutex acquisition.

Evaluation

Performance

- „AtomDir” shows the benefit of being able to **cache atomic data.**
- „Topology” shows the benefit of **distributing ownership.**
- “SpecFetch” shows the advantage of issuing **speculative memory fetches** along with mutex acquisition.



Sean Franey, ”Accelerating Atomic Operations on GPGPUs”, talk 2013.

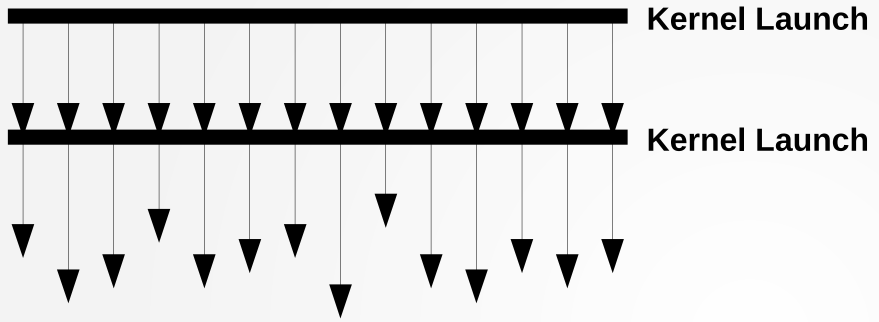
Summary

- Proposed mechanisms show **good performance improvements**.
- High **overhead for control logic and storage**.
- Needs resources (wires) from the underlying interconnection network.
- L2 cache latency has reduced since Fermi (Fermi 310 cycles, Maxwell 194 cycles).

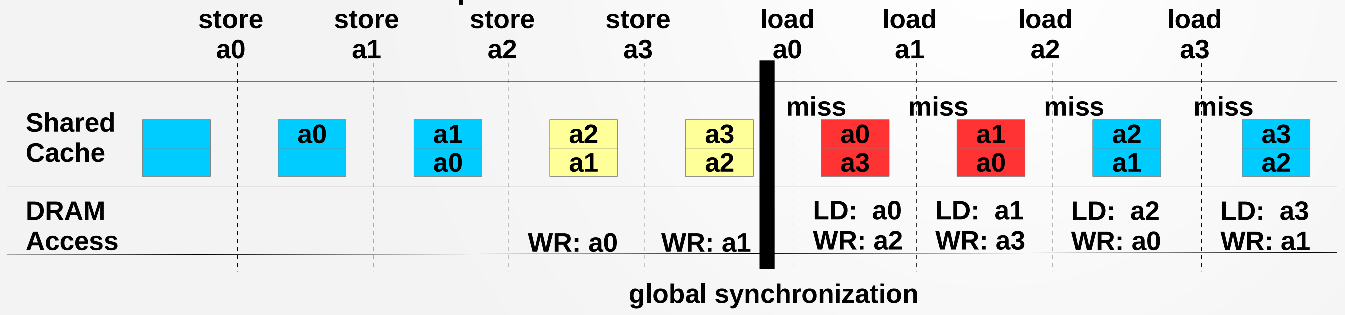
Communication Through Caches

Motivation

- GPU applications suffer from the **lack of an efficient inter-block synchronization** mechanism.
- Exit the current kernel and **re-launch** the **successive kernel** after a global synchronization by the host.



- L2 cache can be used to provide a buffer for inter-block communication.



Amount of off-chip memory accesses is the same, whether there is L2 cache or not.

Write-buffering
(for inter-block communication)

Approach

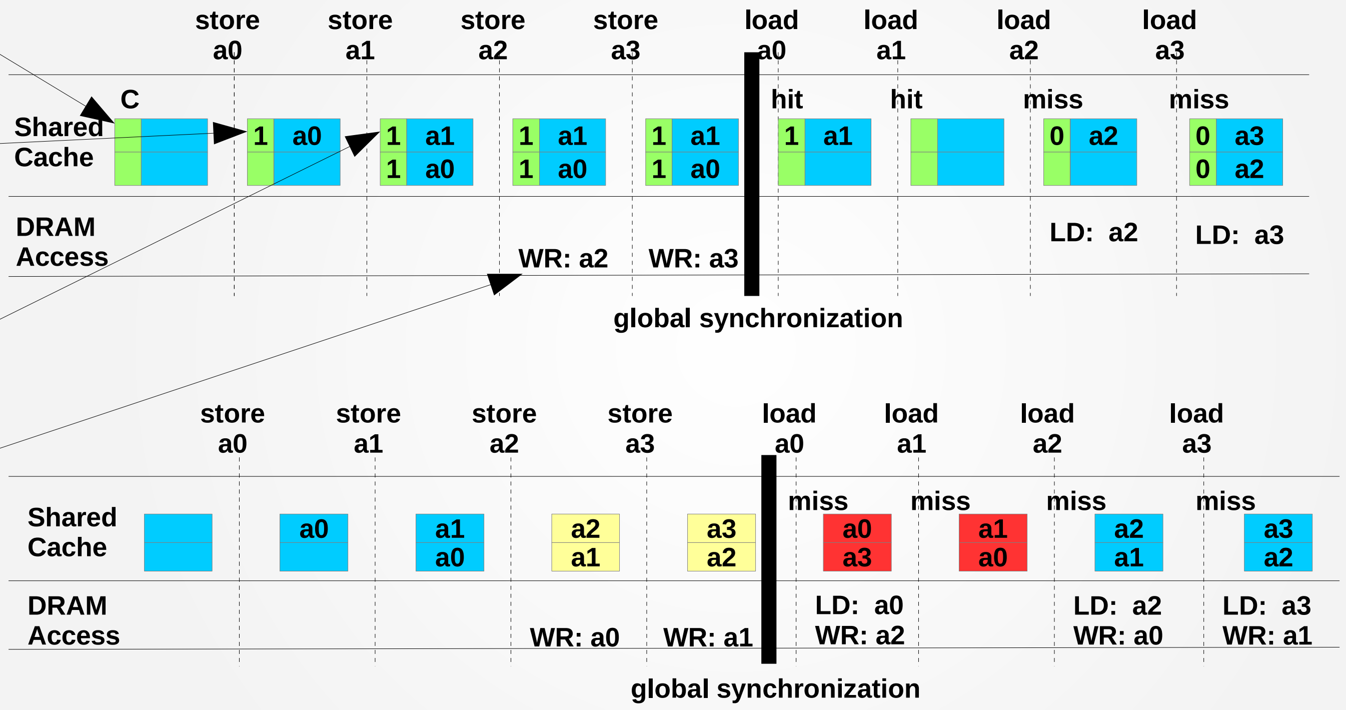
- L2 cache works as a FIFO (LRU replacement policy).
- Choi et al.(0) prevent this by modifying the cache management scheme.

1-bit status flag (C) is added to every cache line.

Write miss [C=0]:
Cache line is allocated and C is set.

Write miss [C=1]:
Line is not selected for replacement.

Every C is set:
Bypass L2 cache to off-chip memory.



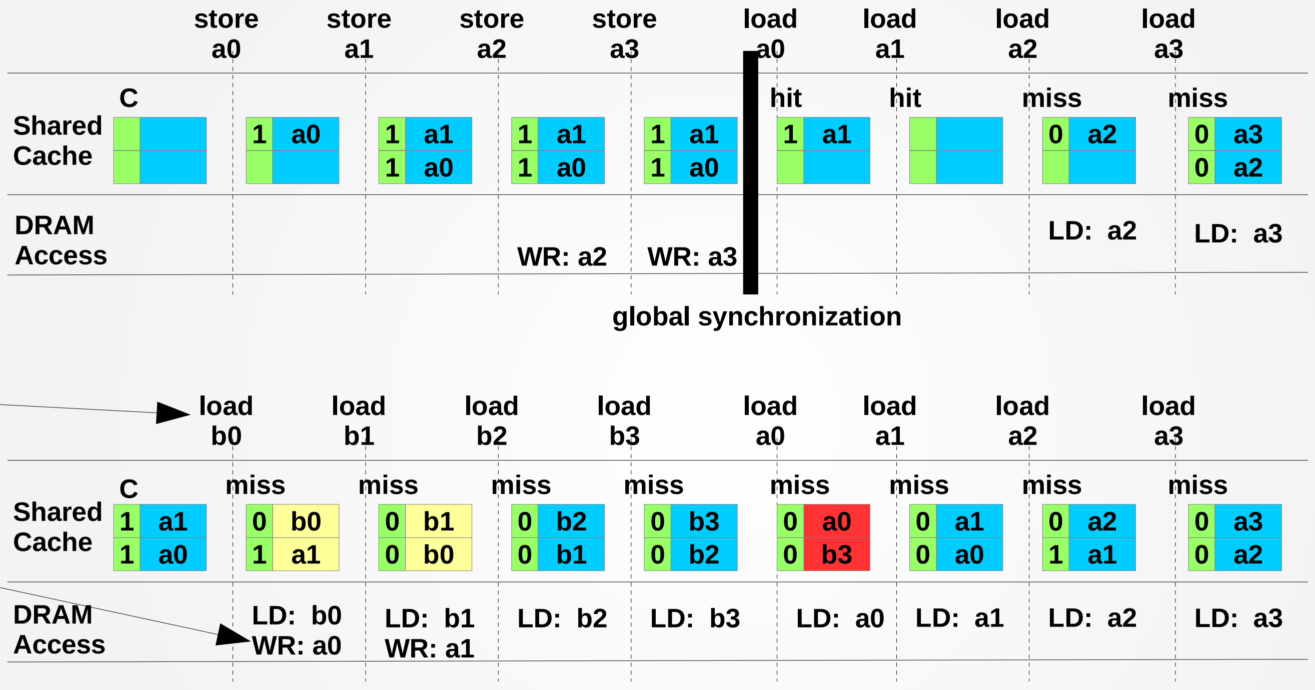
Two writes and two reads for a0 and a1 are reduced when compared with the LRU policy.

(0) H. Choi, J. Ahn, and W. Sung, “Reducing off-chip memory traffic by selective cache management scheme in GPGPUs,” in 5th Annual Workshop on General Purpose Processing with Graphics Processing Units. ACM, 2012, pp. 110–119.

Write-buffering
(for inter-block communication)

Issue

- With only the write-buffering, the shared cache may not retain the data until they are read in the successive kernel.



Load operations may evict cache lines due to conflict or capacity misses.

No benefit of Write-buffering.

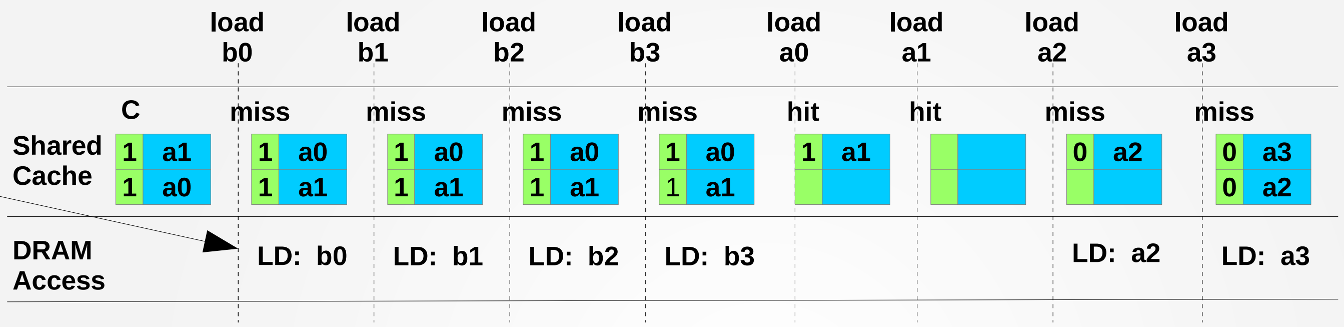
The number of off-chip memory access is the same with that of the pure LRU replacement policy.

Read-Bypassing
(for inter-block communication)

Approach

- Private data load operations, simply bypasses L2 cache to upper-level memory.

Load operations bypass to L1 cache and do not allocate L2 cache lines.



- Proposed scheme is software-controlled.
  (load and store instructions are marked with their respective scheme)
- Two additional cache operators are defined for PTX ISA.

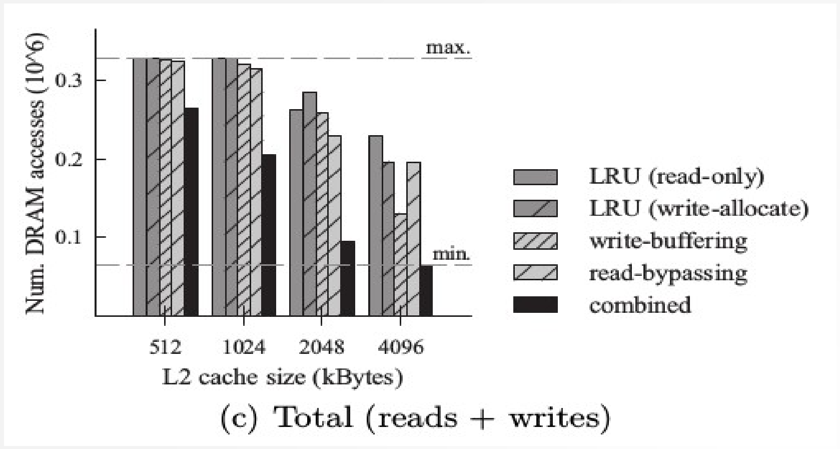
Instruction	Option	Description
ld.global	.cc	Bypasses L2 cache.
st.global	.cp	Allocates cache line on a cache miss an sets the C bit for write-buffering.

(0) H. Choi, J. Ahn, and W. Sung, "Reducing off-chip memory traffic by selective cache management scheme in GPGPUs," in 5th Annual Workshop on General Purpose Processing with Graphics Processing Units. ACM, 2012, pp. 110–119.

Evaluation

Performance

- Workloads: FFT, HotSpot and SRAD.
- Proposed technique reduces the amount of write and read traffic to the off-chip memory.



Effect on the off-chip memory traffic reduction in FFT [0].

Summary

- Very **low implementation costs**.
- Good **performance improvements**.
  - Larger L2 size also improves performance (Fermi 768KB, Maxwell 2048 KB).
  - Faster L2 caches should further improve performance (Fermi 310 Cycles, Maxwell 194 Cycles).
- High **programming overhead**.

[0] H. Choi, J. Ahn, and W. Sung, “Reducing off-chip memory traffic by selective cache management scheme in GPGPUs,” in 5th Annual Workshop on General Purpose Processing with Graphics Processing Units. ACM, 2012, pp. 110–119.

GPU-CPU
Heterogeneous Architectures

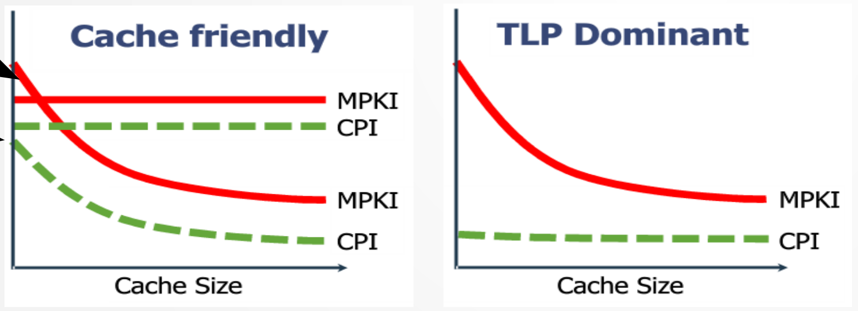
Combining GPU cores with conventional CPUs is a trend.

- Various resources are shared between GPU and CPU cores.
  (LLC, on-chip interconnect, memory controller and DRAM)
- Shared cache is one of the most important resources.

CPU and GPU cores have different characteristics.

- GPU cores have an order-of-magnitude more threads.
- GPUs have higher TLP (Thread-Level-Parallelism) than CPUs.
- TLP has significant impact on how caching affects performance of applications.

MPKI	**Misses Per Kilo Instruction**
CPI	**Cycles Per Instruction**



J. Lee and H. Kim, "TLP Aware Cache Management Policy", Talk HPCA-18.

We need to directly monitor performance effect of cache.

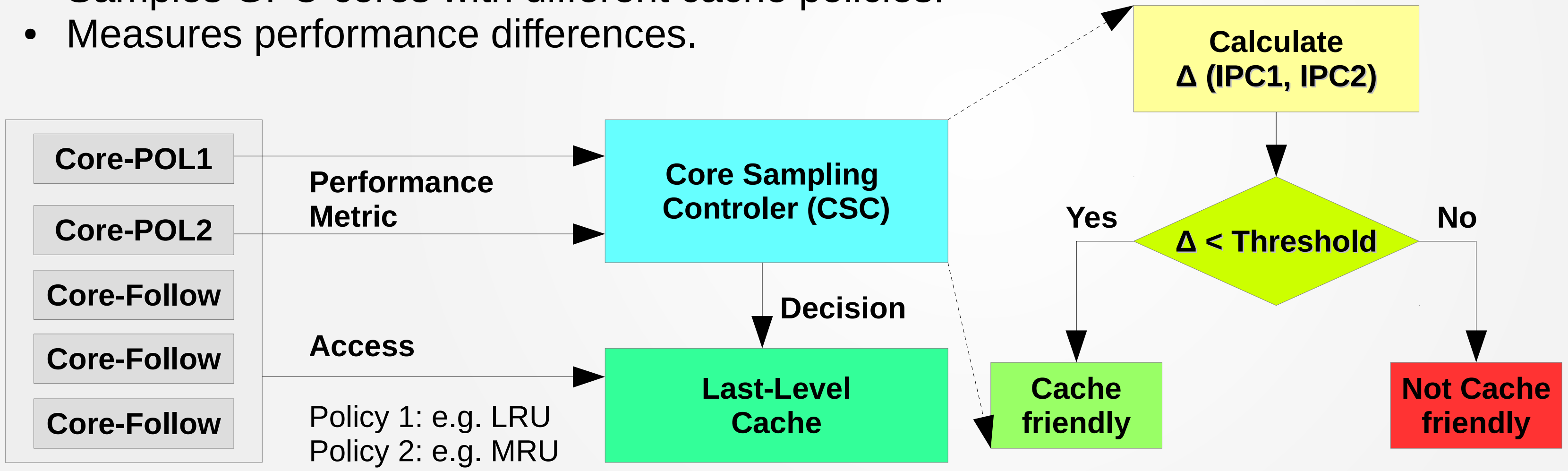
TLP-Aware Cache Management Policy (TAP)

Lee et al. introduced TAP mechanism (0)

- Bypass LLC.
- Core Sampling.
- Cache block lifetime normalization.
- TAP-UCP and TAP-RRIP.

Core Sampling

- Samples GPU cores with different cache policies.
- Measures performance differences.

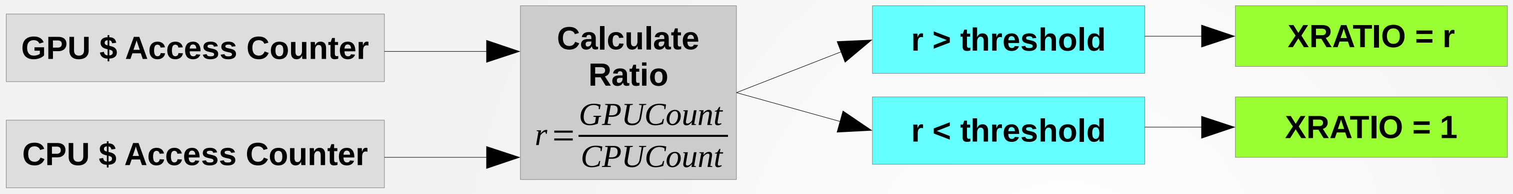


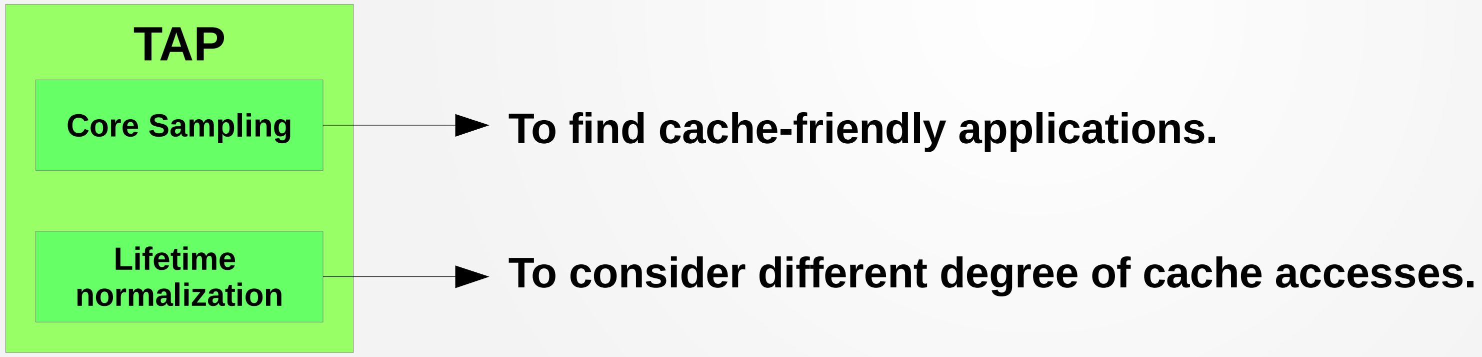
(0) J. Lee and H. Kim, “TAP: A TLP-aware cache management policy for a CPU-GPU heterogeneous architecture,” in 18th International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2012, pp. 1–12.

TLP-Aware Cache Management Policy (TAP)

Cache block lifetime normalization

- GPU cores have an order-of-magnitude more cache accesses.
- Monitor cache access rate differences between CPU and GPU applications and periodically calculate ratio.

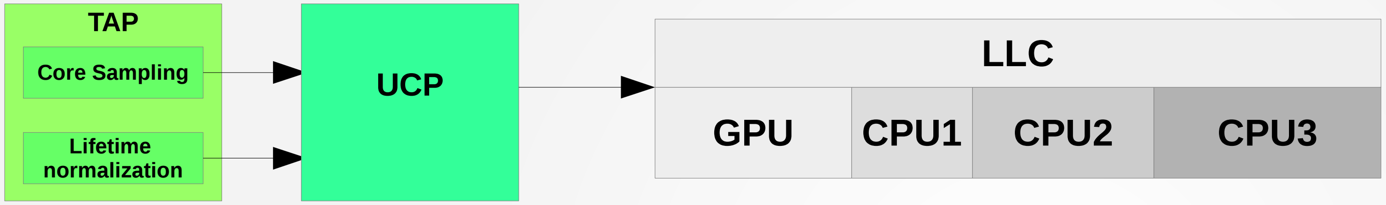



TAP
Core Sampling
Lifetime normalization
To find cache-friendly applications.
To consider different degree of cache accesses.

TLP-Aware
Cache Management Policy (TAP)

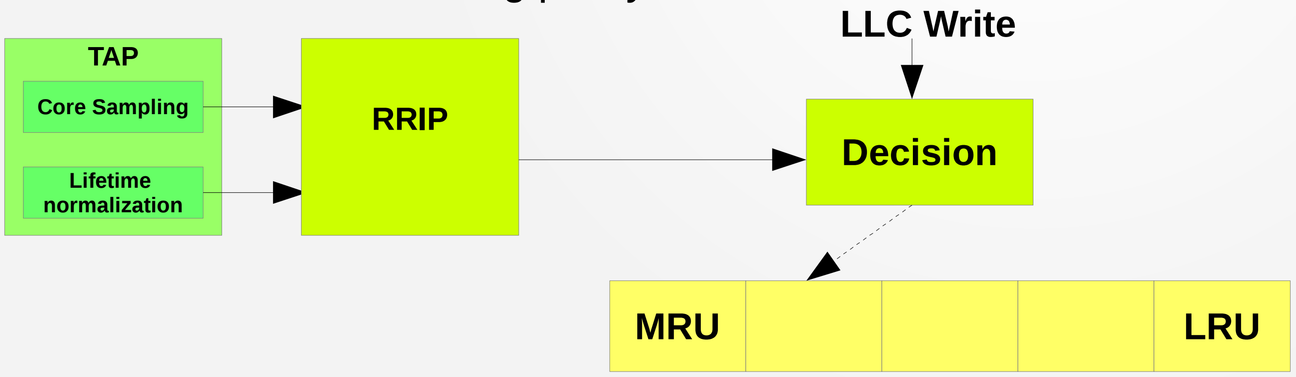
TAP-Utility-Based Cache Partitioning (TAP-UCP)

- UCP is a dynamic cache partitioning mechanism for only CPU workloads.
- Allocate more cache space to applications that obtain the most benefit from more space.



TAP-Re-Reference Interval Prediction (TAP-RRIP)

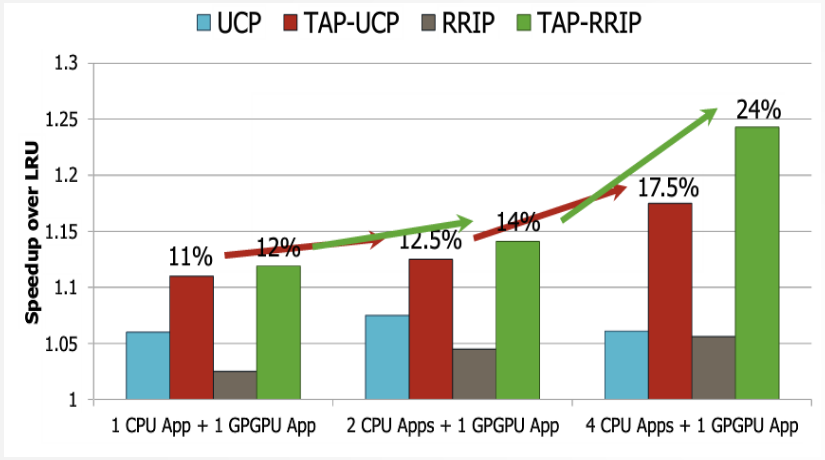
- Dynamically adapts between two competing cache insertion policies, Static RRIP (SRRIP) and Bimodal-RRIP (BRRIP).
- Policy Selector (PSEL), keeps track of which policy incurs fewer cache misses and decides the winning policy.



Evaluation

Performance

- 152 heterogeneous workloads.
- Improve the performance by 5% and 10% compared to UCP and RRIP and 11% and 12% to LRU.
- Higher benefits with more CPU applications.



J. Lee and H. Kim, "TLP Aware Cache Management Policy", Talk HPCA-18.

Summary

- LLC management is an important problem in future many-core-heterogeneous processors.
- TAP mechanism improves performance.
- High overhead for control logic and storage.
- Previous mechanisms don't consider GPGPU-specific characteristics in heterogeneous workloads.

Conclusion

- Multi-level hardware-managed caches are **recent addition to GPUs**.
- **Effective management** of caches is very important to fully exploit their potential in boosting **GPU performance and energy efficiency**.
- Various proposals have been published the last years.
- In this talk:
  - Low-latency mechanism for acquiring and releasing mutexes in a system.
  - Reduce off-chip memory accesses by write-buffering and read-bypassing.
  - Technique to profile a GPGPU application at run-time in heterogeneous architectures .
- More Literature: Sparsh Mittal, “*A Survey of Techniques for Managing and Leveraging Caches in GPUs*”, Journal of Circuits, Systems, and Computers 2014.

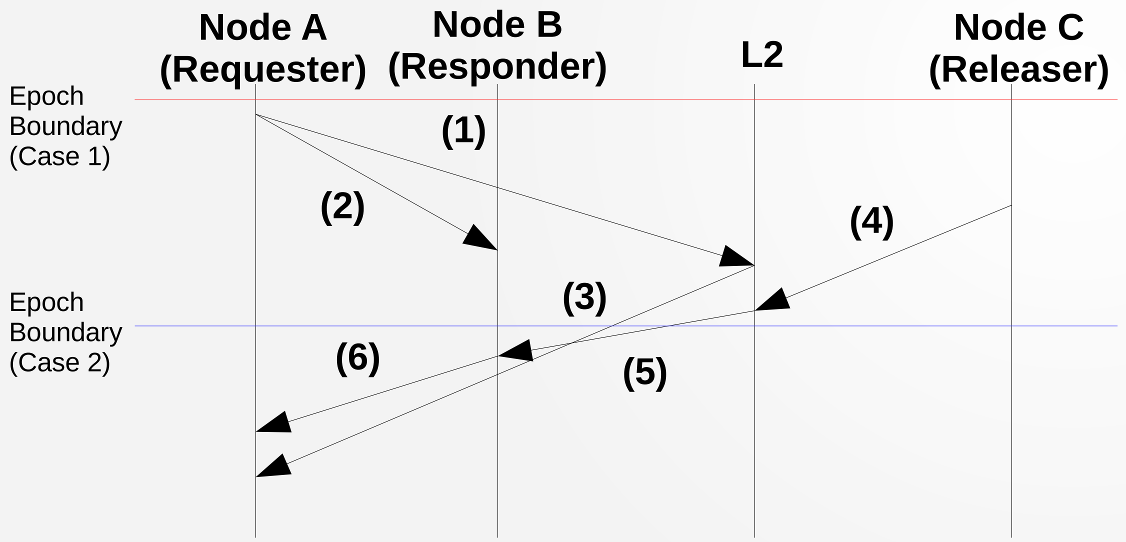
Discussion

Thank you!

Speculative Fetch

Mutex acquisition delays fetch. Issue fetch in parallel with mutex acquisition. Ensure correctness via epochs.

- Epoch consists of a fixed number of cycles.
- At the boundary of each epoch, all responders indicate that their mutex releases are mature and all requesters indicate that their outstanding mutex requests are stale.
- When the requester receives the mutex and both the release is mature and the request is not stale, the requesting node knows that no update could have occurred to the data.



(1) Speculative Fetch
(2) Mutex Request
(3) Speculative Response
(4) Write Back + Realese
(5) Mutex Release
(6) Mutex Response