Memory Hierarchy (3): Cache misses and their remedies — the hardware version

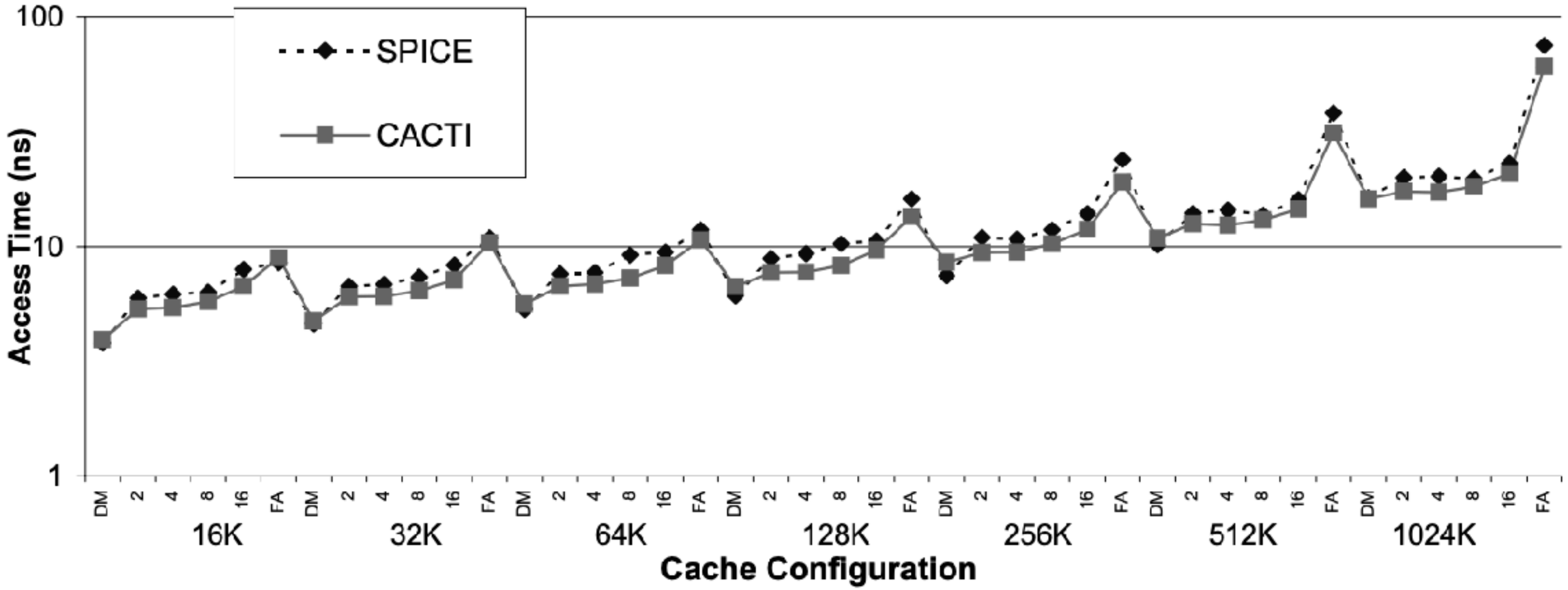
Hung-Wei Tseng

Outline

- A, B, C, S and cache misses
- Architectural support for optimizing cache performance

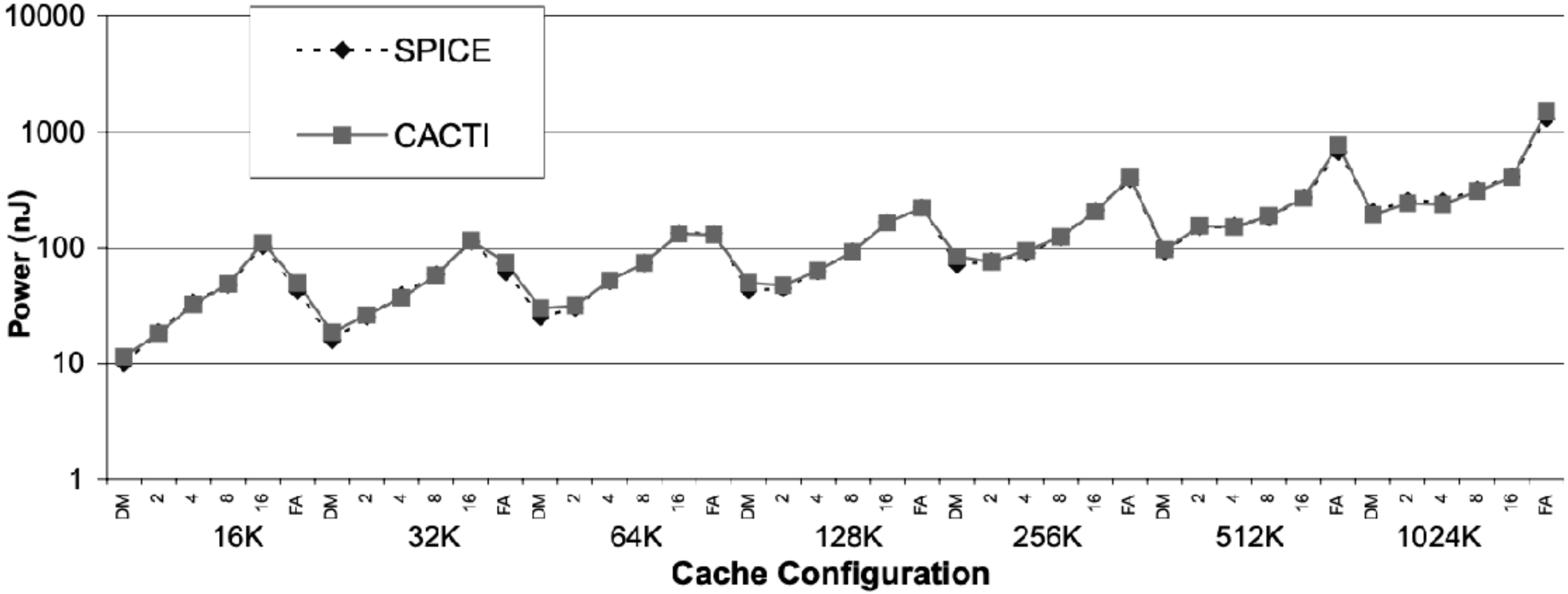
A, B, C, S and cache misses

Cache configurations and access time



Glen Reinman and Norman P. Jouppi. CACTI 2.0: An Integrated Cache Timing and Power Model. WRL Research Report 2000/7
https://bitsavers.trailing-edge.com/pdf/dec/tech_reports/WRL-2000-7.pdf

Cache configurations and access power



Glen Reinman and Norman P. Jouppi. CACTI 2.0: An Integrated Cache Timing and Power Model. WRL Research Report 2000/7
https://bitsavers.trailing-edge.com/pdf/dec/tech_reports/WRL-2000-7.pdf

Takeaways: Optimizing cache performance through hardware

- There is no optimal cache configurations — trade-offs are everywhere
  - Increasing C — (+): capacity misses; (-): cost, access time, power
  - Increasing A — (+): conflict misses; (-): access time, power
  - Increasing B — (+): compulsory misses; (-): miss penalty

Recap: NVIDIA Tegra X1

- D-L1 Cache configuration of NVIDIA Tegra X1
  - Size 32KB, 4-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[8192], b[8192], c[8192], d[8192], e[8192];
/* a = 0x10000, b = 0x20000, c = 0x30000, d = 0x40000, e = 0x50000 */
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
    //load a[i], b[i], c[i], d[i] and then store to e[i]
}
```

What's the data cache miss rate for this code?

A. 12.5%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

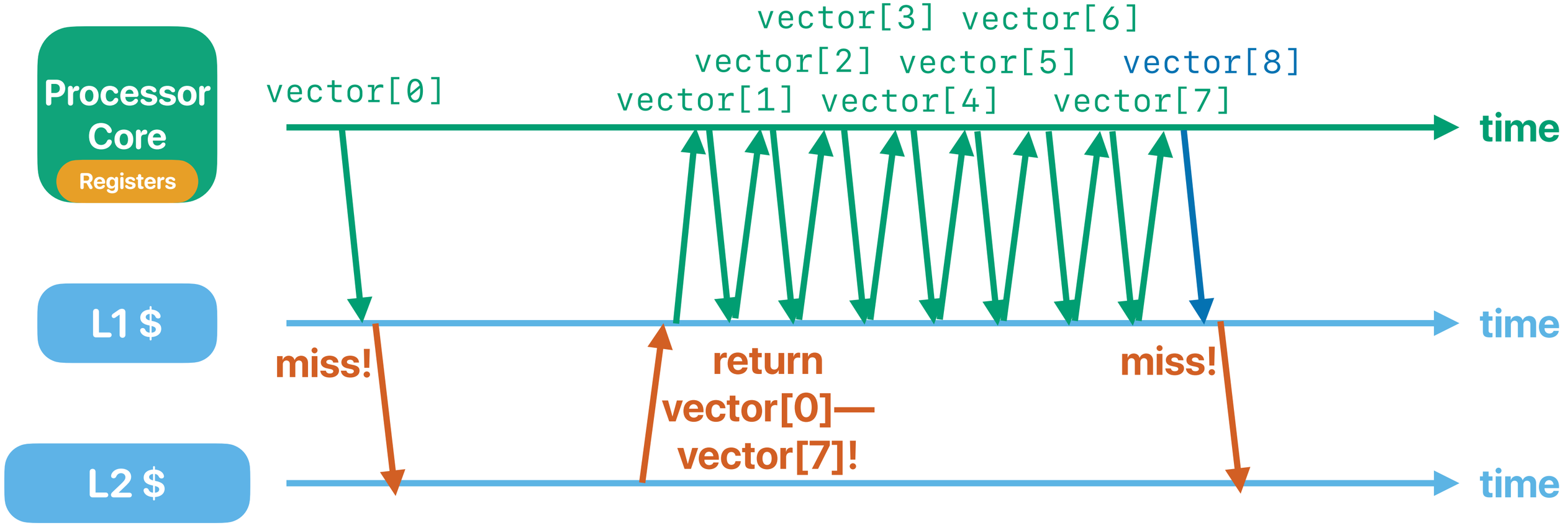
Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers

Norman P. Jouppi

Prefetching

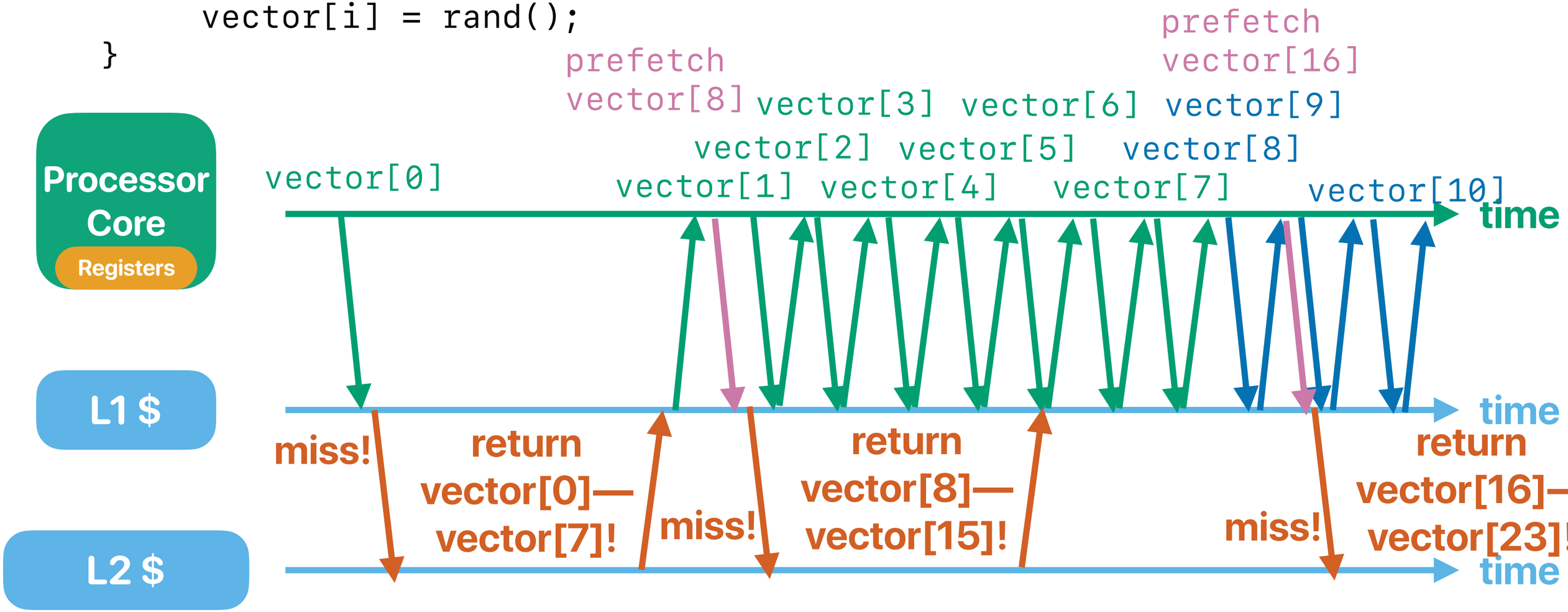
Spatial locality revisited

```
for(i = 0;i < size; i++) {
      vector[i] = rand();
}
```



What if we "pre-"fetch the next line?

```
for(i = 0;i < size; i++) {
     vector[i] = rand();
}
```



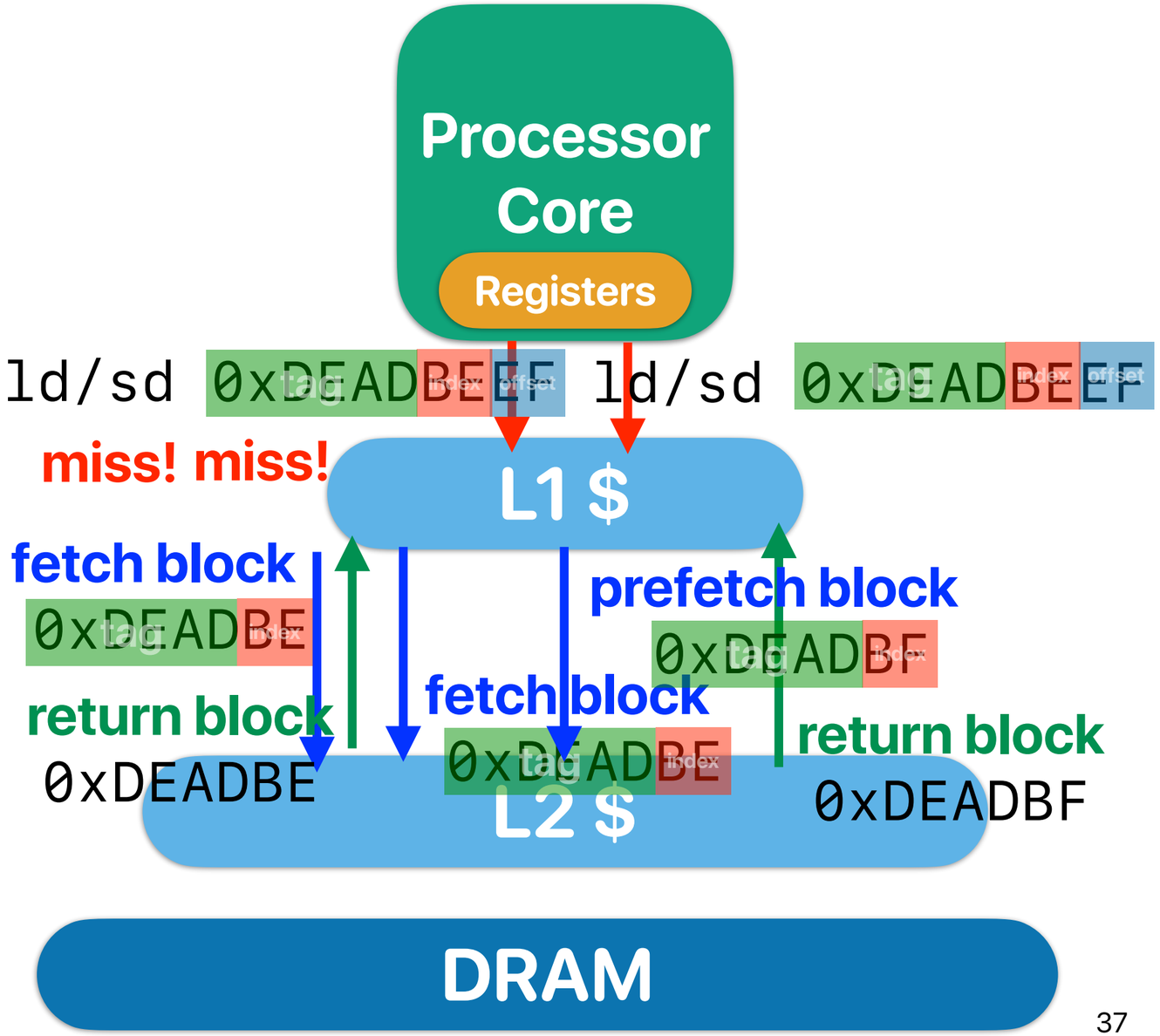
Prefetching

- Identify the access pattern and proactively fetch data/instruction before the application asks for the data/instruction
  - Trigger the cache miss earlier to eliminate the miss when the application needs the data/instruction
- Hardware prefetch
  - The processor can keep track the distance between misses. If there is a pattern, fetch miss_data_address+distance for a miss
- Software prefetch
  - Load data into X0
  - Using prefetch instructions

Demo

- x86 provide prefetch instructions
- As a programmer, you may insert `_mm_prefetch` in x86 programs to perform software prefetch for your code
- gcc also has a flag "-fprefetch-loop-arrays" to automatically insert software prefetch instructions

What's after prefetching?



NVIDIA Tegra X1 with prefetch

- Size 32KB, 4-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[8192], b[8192], c[8192], d[8192], e[8192];
/* a = 0x10000, b = 0x20000, c = 0x30000, d = 0x40000, e = 0x50000 */
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
    //load a[i], b[i], c[i], d[i] and then store to e[i]
```

C = ABS
32KB = 4 * 64 * S
S = 128
offset = lg(64) = 6 bits
index = lg(128) = 7 bits
tag = the rest bits

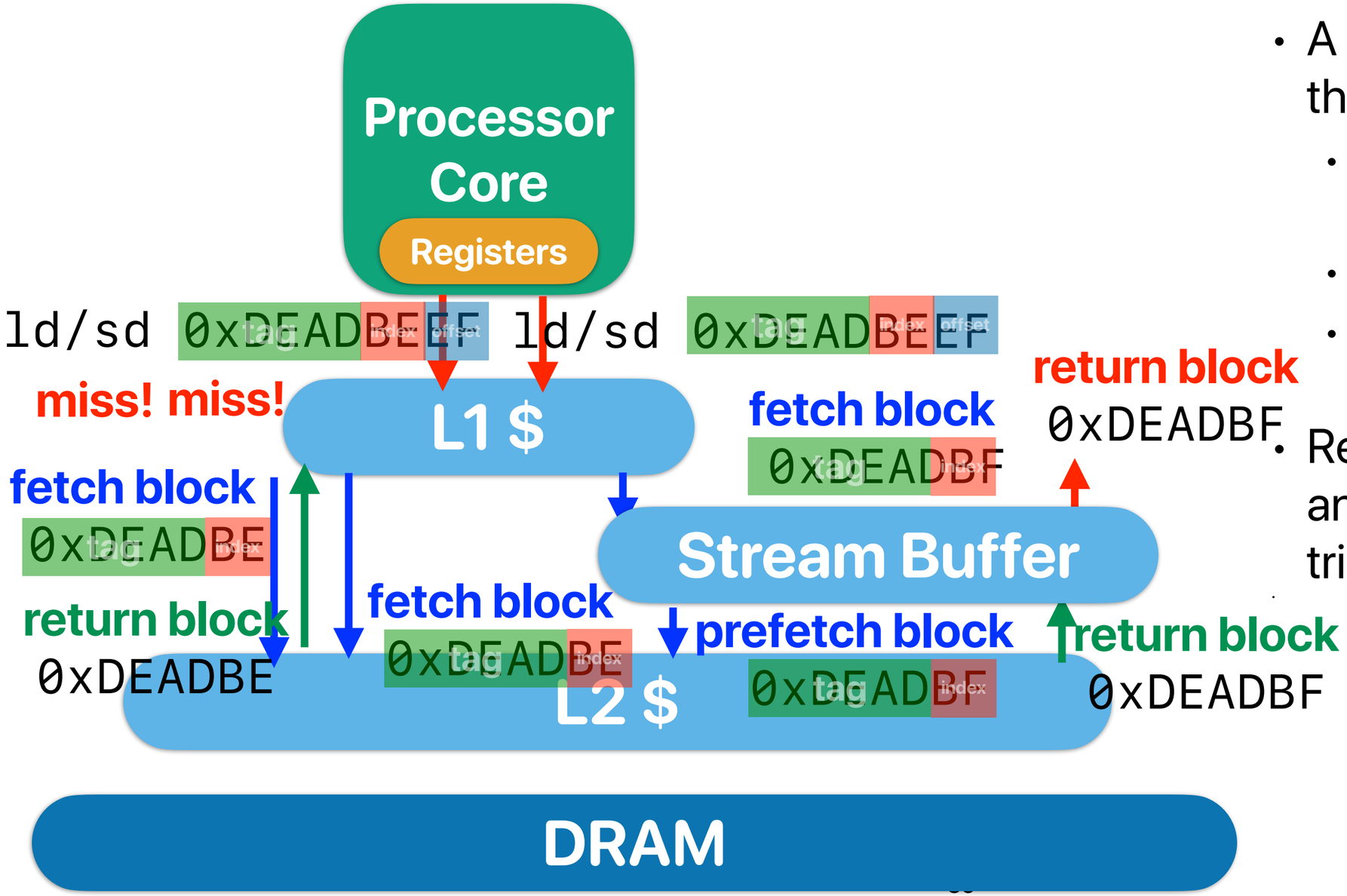
tag index offset

	Address (Hex)	Address in binary	Tag	Index	Hit? Miss?	Replace?	Prefetch
a[0]	0x10000	0b0001000000000000000000	0x8	0x0	Miss		a[8–15]
b[0]	0x20000	0b0010000000000000000000	0x10	0x0	Miss		b[8–15]
c[0]	0x30000	0b0011000000000000000000	0x18	0x0	Miss		c[8–15]
d[0]	0x40000	0b0100000000000000000000	0x20	0x0	Miss		d[8–15]
e[0]	0x50000	0b0101000000000000000000	0x28	0x0	Miss	a[0–7]	e[8–15]
a[1]	0x10008	0b0001000000000000001000	0x8	0x0	Miss	b[0–7]	
b[1]	0x20008	0b0010000000000000001000	0x10	0x0	Miss	c[0–7]	
c[1]	0x30008	0b0011000000000000001000	0x18	0x0	Miss	d[0–7]	
d[1]	0x40008	0b0100000000000000001000	0x20	0x0	Miss	e[0–7]	
e[1]	0x50008	0b0101000000000000001000	0x28	0x0	Miss	a[0–7]	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

e[8-15] will kick out a[8-15]

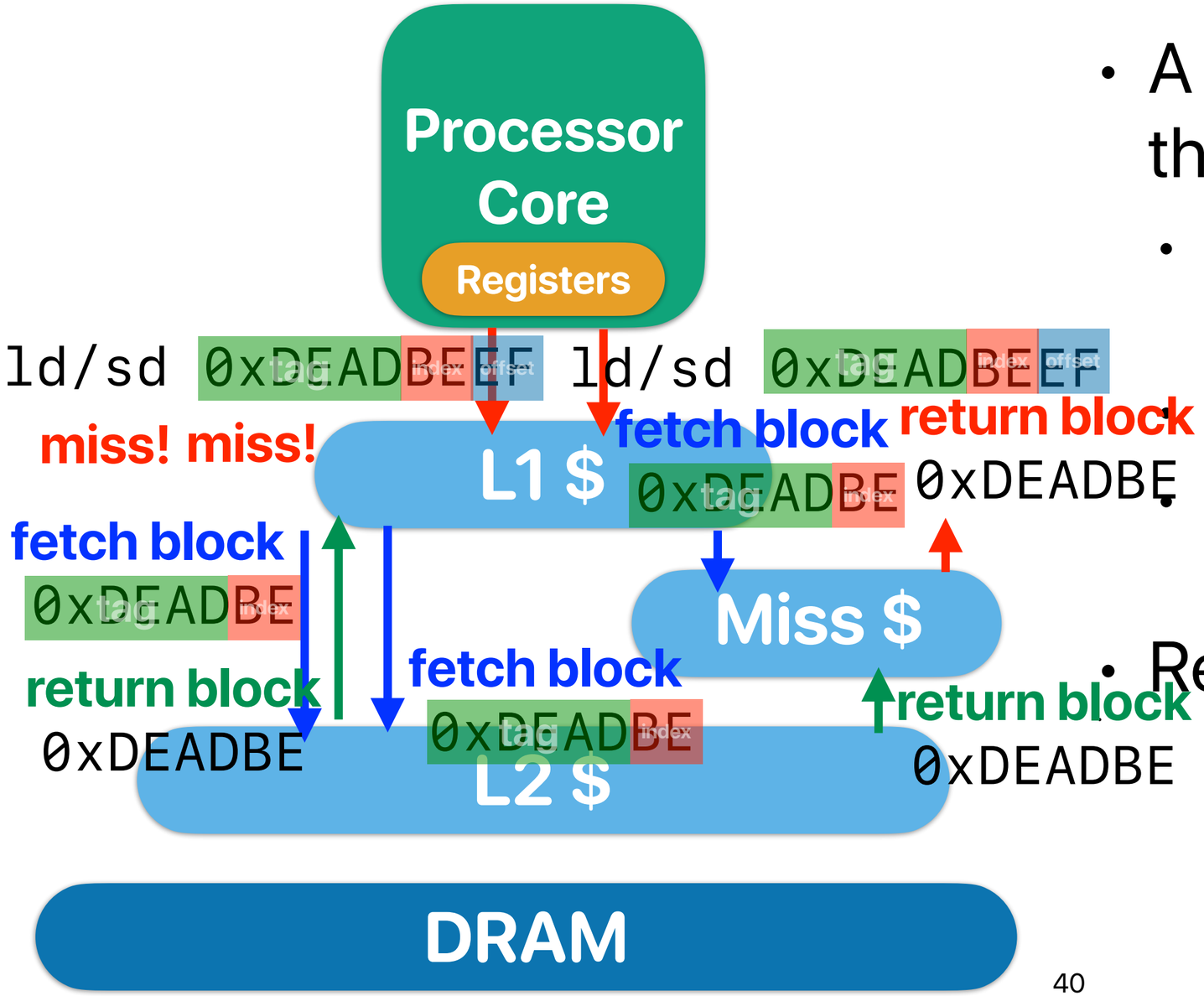
100% miss rate!

Stream buffer



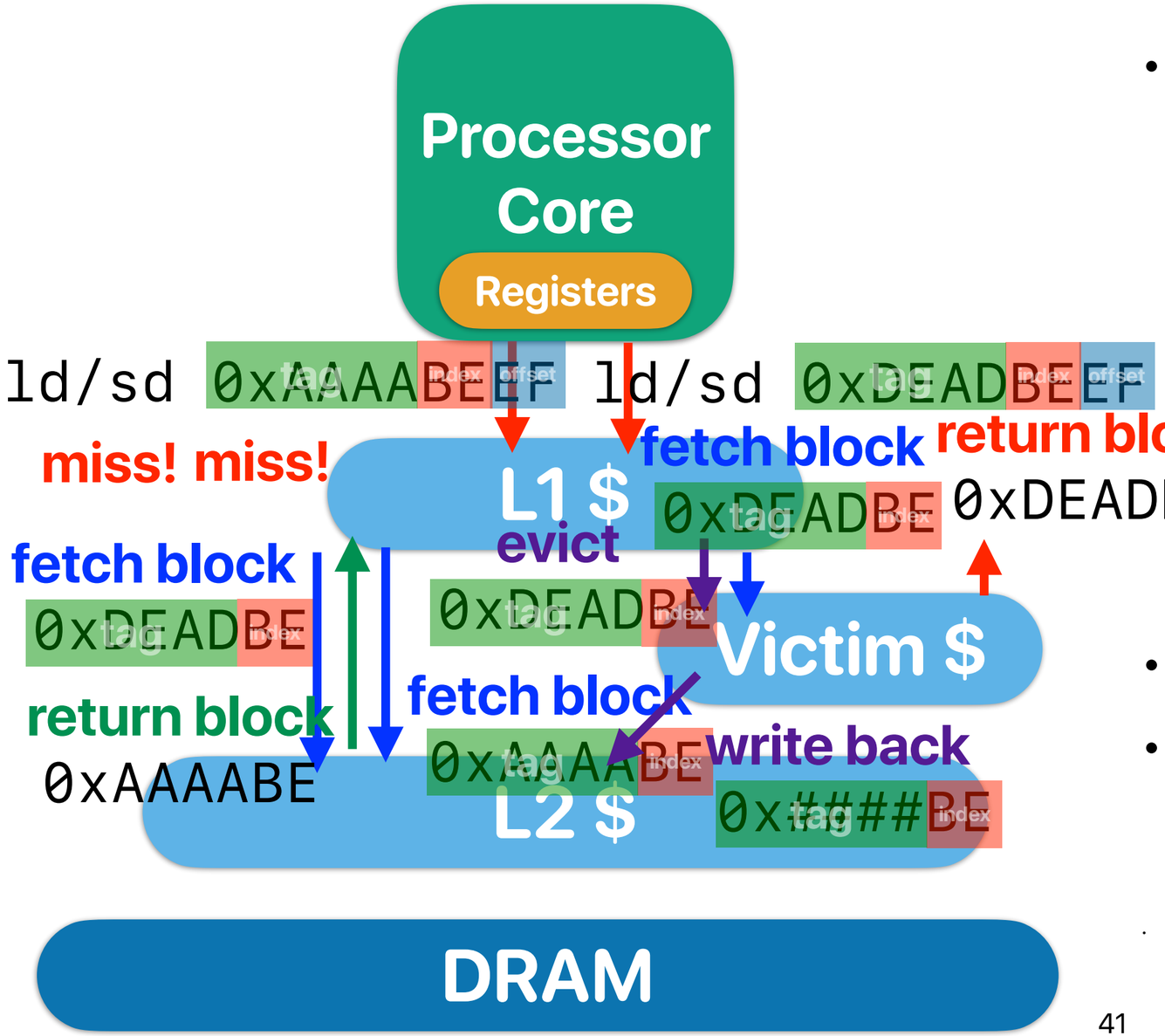
- A small cache that captures the prefetched blocks
  - Can be built as fully associative since it's small
  - Consult when there is a miss
  - Retrieve the block if found in the stream buffer
- Reduce compulsory misses and avoid conflict misses triggered by prefetching

Miss cache



- A small cache that captures the missing blocks
  - Can be built as fully associative since it's small
  - Consult when there is a miss
  - Retrieve the block if found in the missing cache
- Reduce conflict misses

Victim cache



- A small cache that captures the evicted blocks
  - Can be built as fully associative since it's small
  - Consult when there is a miss
  - Swap the entry if hit in victim cache
  - Athlon/Phenom has an 8-entry victim cache
- Reduce conflict misses
- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache

Victim cache v.s. miss caching

- Both of them improves conflict misses
- Victim cache can use cache block more efficiently — swaps when miss
  - Miss caching maintains a copy of the missing data — the cache block can both in L1 and miss cache
  - Victim cache only maintains a cache block when the block is kicked out
- Victim cache captures conflict miss better
  - Miss caching captures every missing block

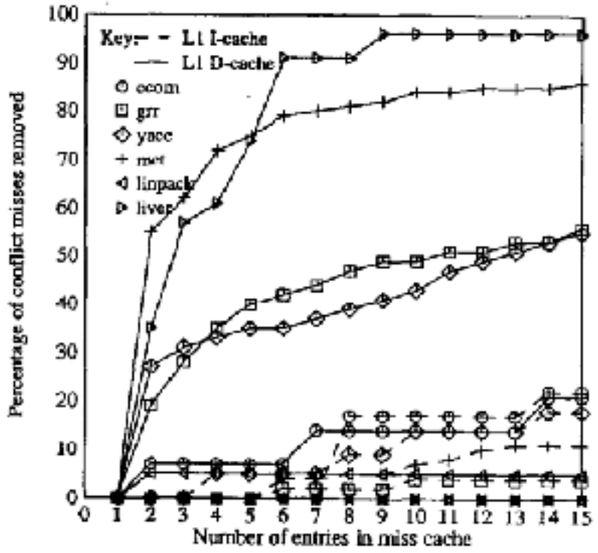


Figure 3-3: Conflict misses removed by miss caching

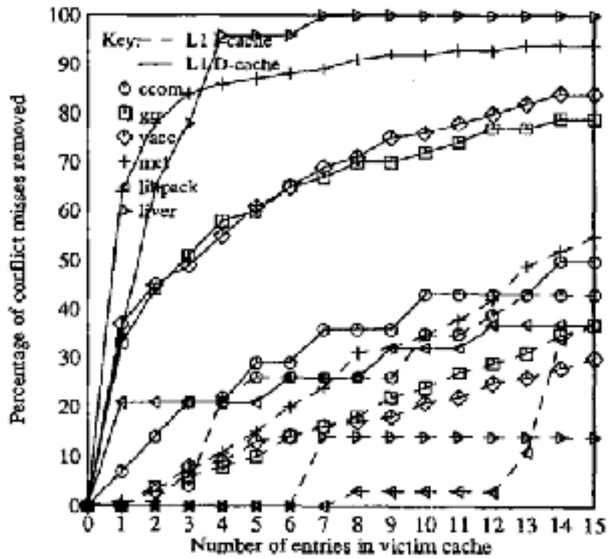


Figure 3-5: Conflict misses removed by victim caching

Takeaways: Optimizing cache performance through hardware

- There is no optimal cache configurations — trade-offs are everywhere
  - Increasing C — (+): capacity misses; (-): cost, access time, power
  - Increasing A — (+): conflict misses; (-): access time, power
  - Increasing B — (+): compulsory misses; (-): miss penalty
- Adding a small buffer alongside the L1 cache can —
  - Virtually add an associative set to frequently used data structures
  - Prefetched blocks won’t cause conflict misses

Takeaways: Optimizing cache performance through hardware

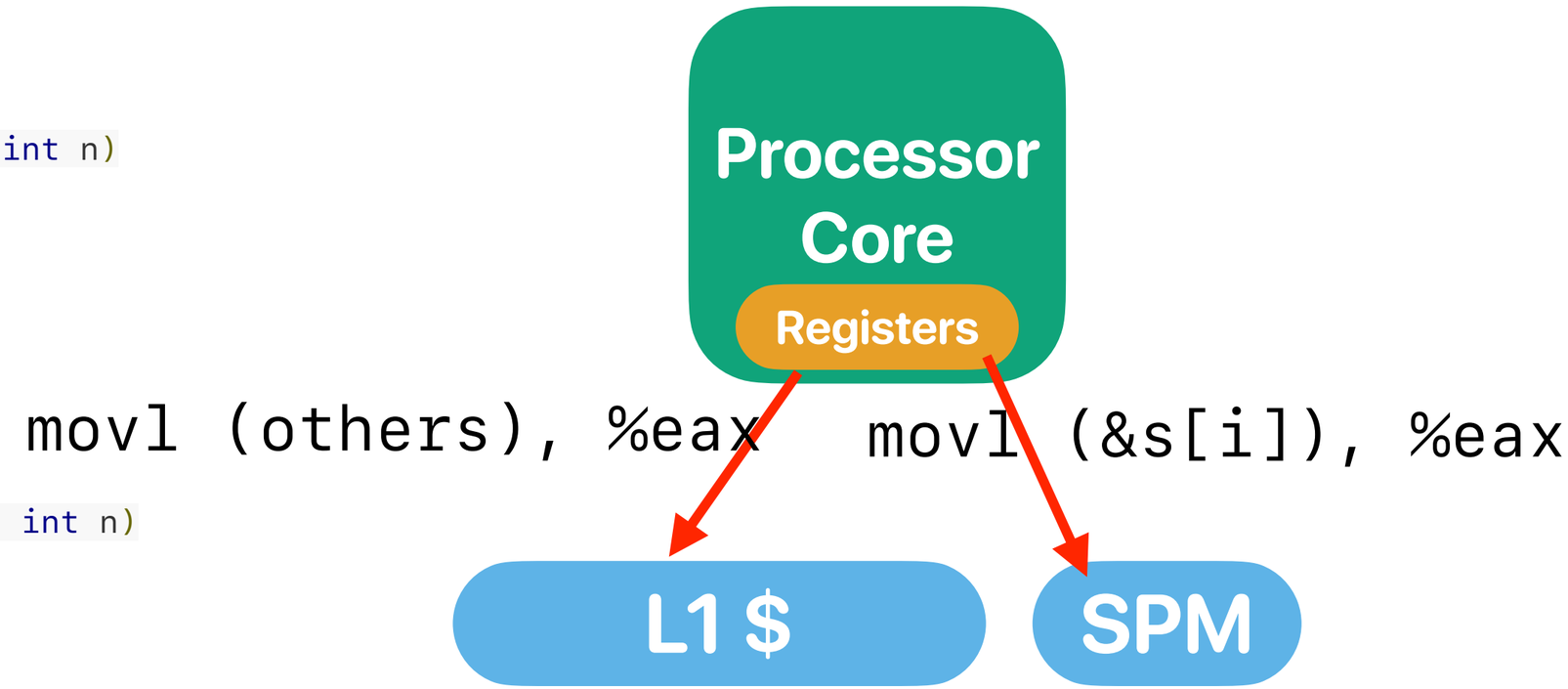
- There is no optimal cache configurations — trade-offs are everywhere
  - Increasing C — (+): capacity misses; (-): cost, access time, **power**
  - Increasing A — (+): conflict misses; (-): access time, **power**
  - Increasing B — (+): compulsory misses; (-): miss penalty
- Adding a small buffer alongside the L1 cache can —
  - Virtually add an associative set to frequently used data structures
  - Prefetched blocks won't cause conflict misses

We need additional search time and power

Implementation of SPM — GPU's shared memory

```
__global__ void staticReverse(int *d, int n)
{
  __shared__ int s[64];
  int t = threadIdx.x;
  int tr = n-t-1;
  s[t] = d[t];
  __syncthreads();
  d[t] = s[tr];
}

__global__ void dynamicReverse(int *d, int n)
{
  extern __shared__ int s[];
  int t = threadIdx.x;
  int tr = n-t-1;
  s[t] = d[t];
  __syncthreads();
  d[t] = s[tr];
}
```



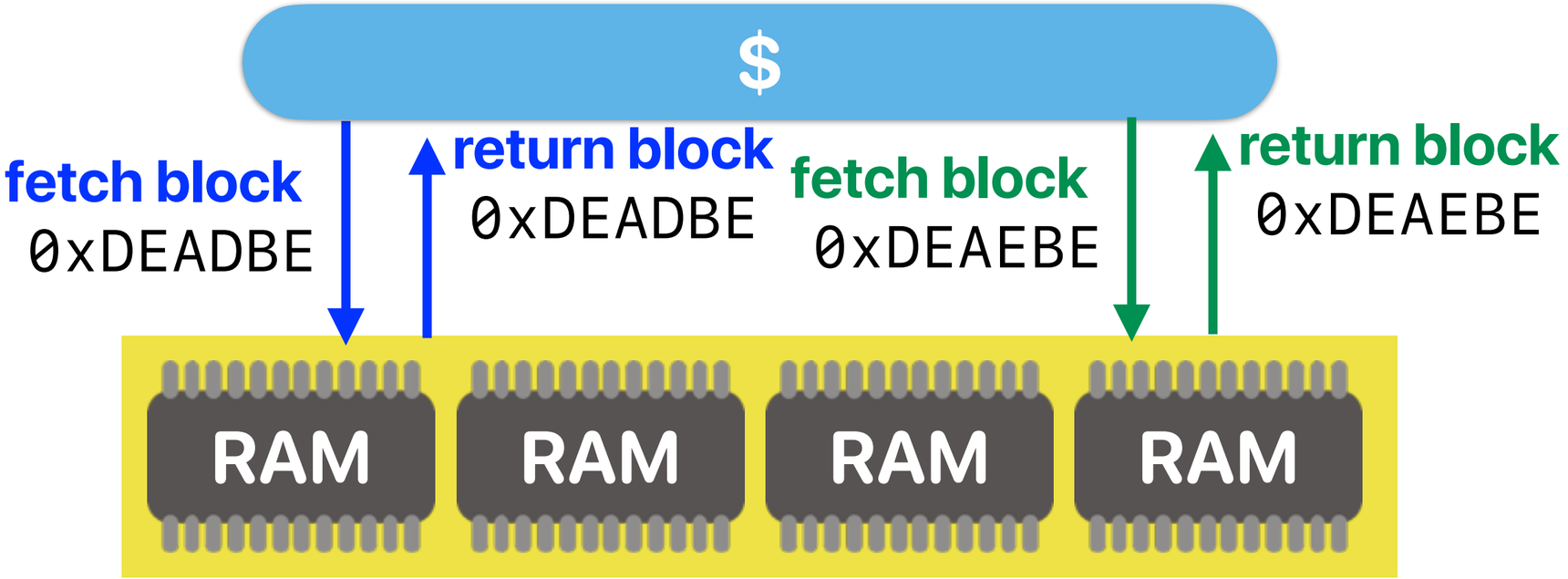
https://developer.nvidia.com/blog/using-shared-memory-cuda-cc/

Takeaways: Optimizing cache performance through hardware

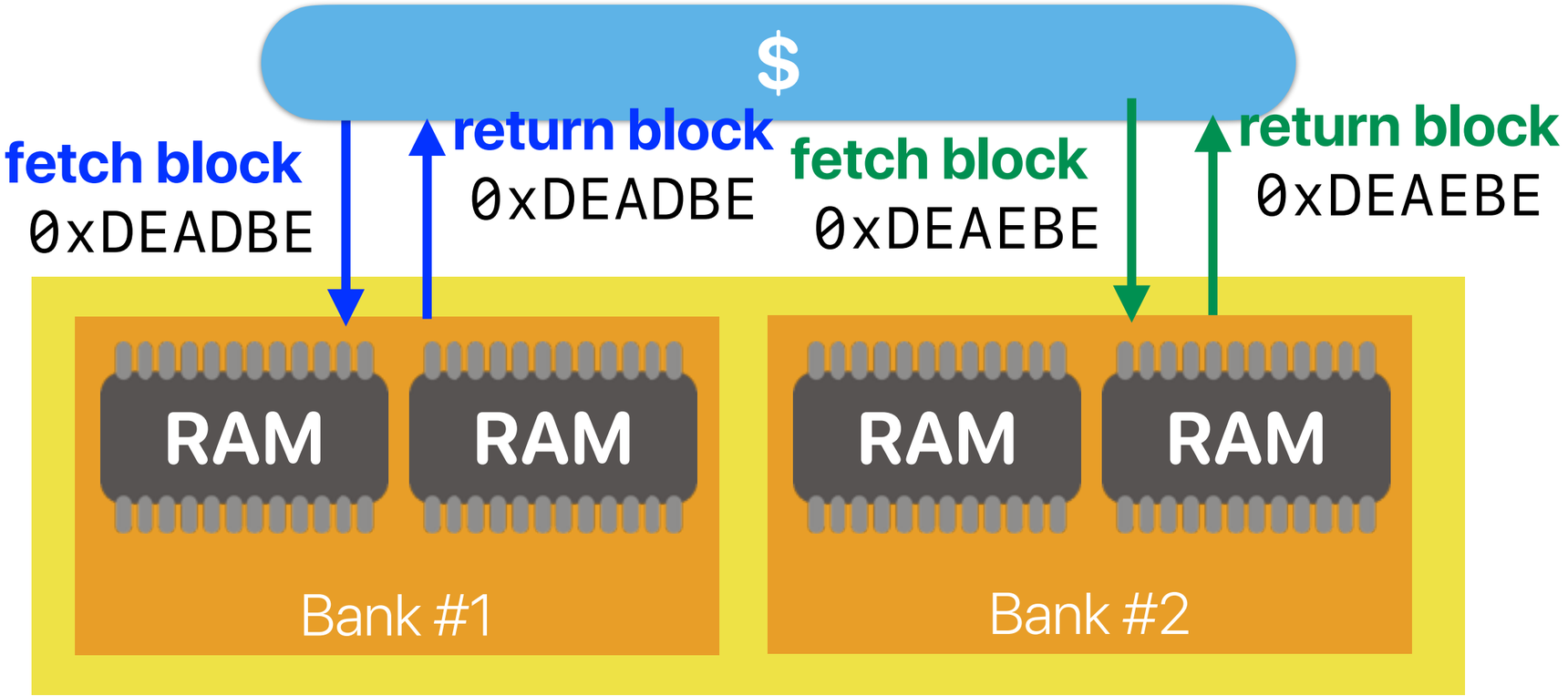
- There is no optimal cache configurations — trade-offs are everywhere
  - Increasing C — (+): capacity misses; (-): cost, access time, power
  - Increasing A — (+): conflict misses; (-): access time, power
  - Increasing B — (+): compulsory misses; (-): miss penalty
- Adding a small buffer alongside the L1 cache can —
  - Virtually add an associative set to frequently used data structures
  - Prefetched blocks won’t cause conflict misses
- Adding a tag-less, programmable small buffer alongside the L1 cache can reduce power consumption

Advanced Hardware Techniques in Improving Memory Performance

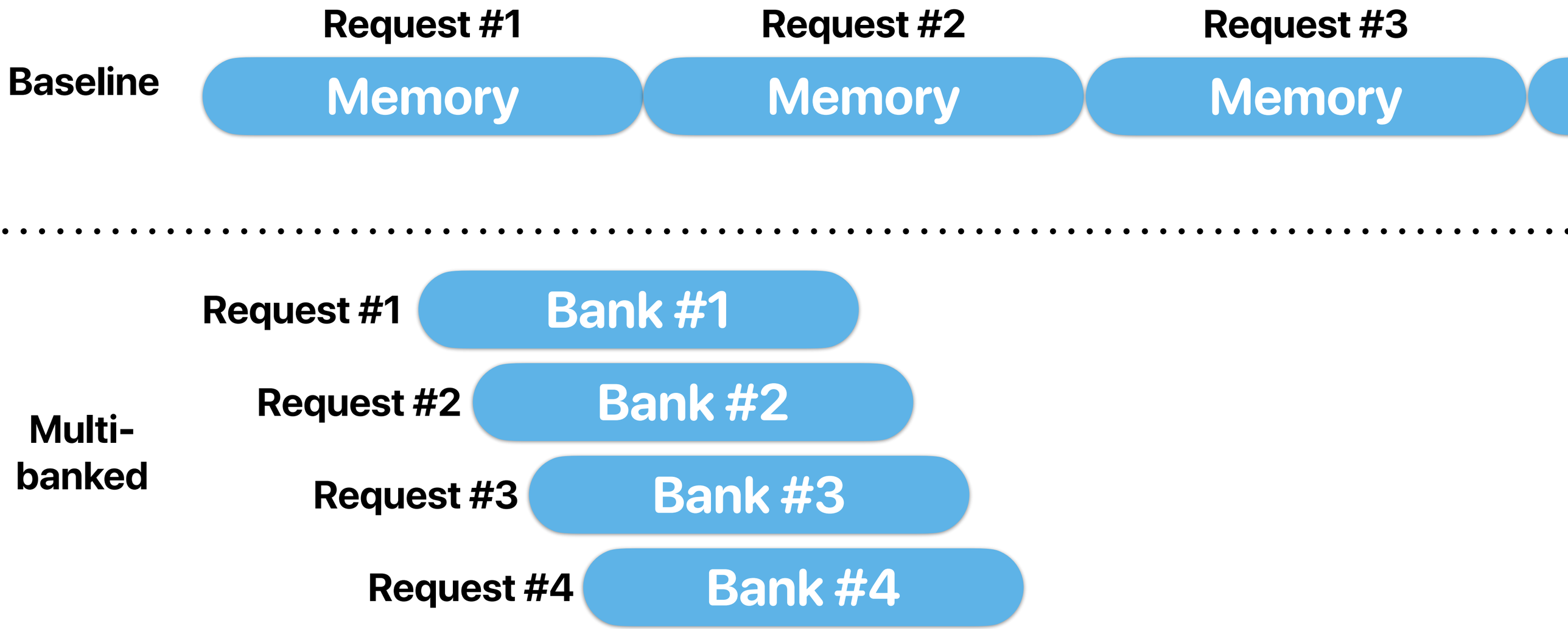
Blocking cache



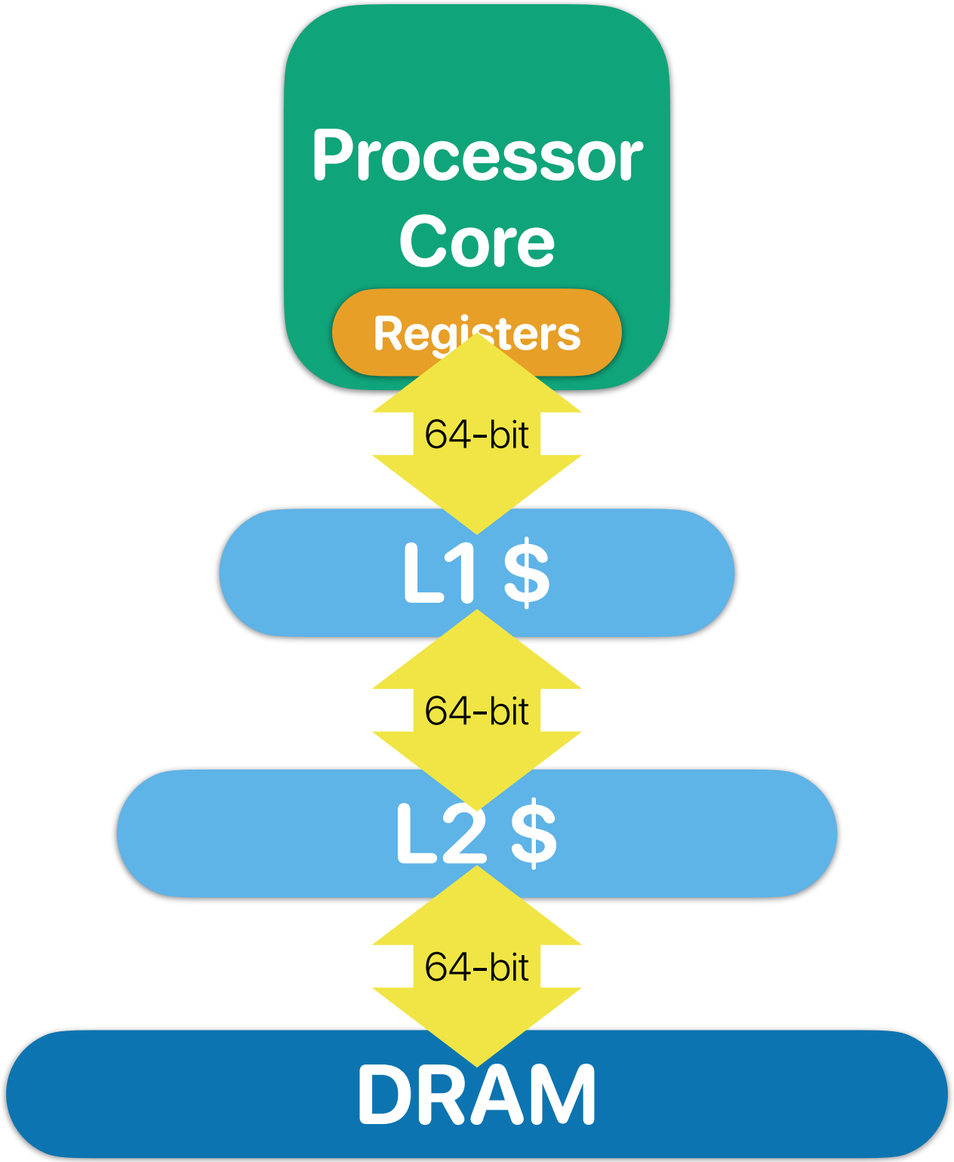
Multibanks & non-blocking caches



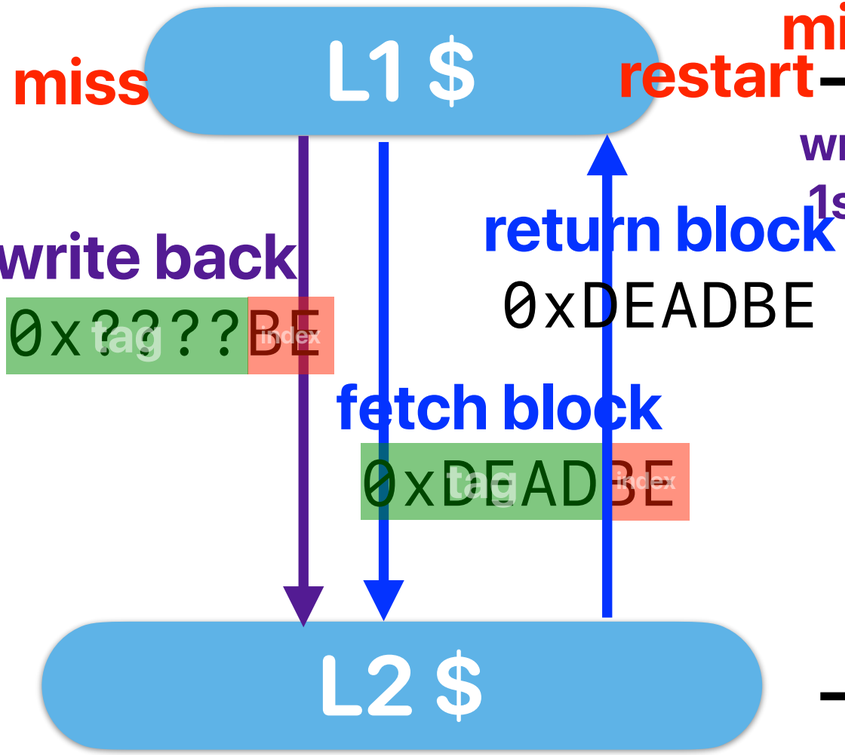
Pipelined access and multi-banked caches

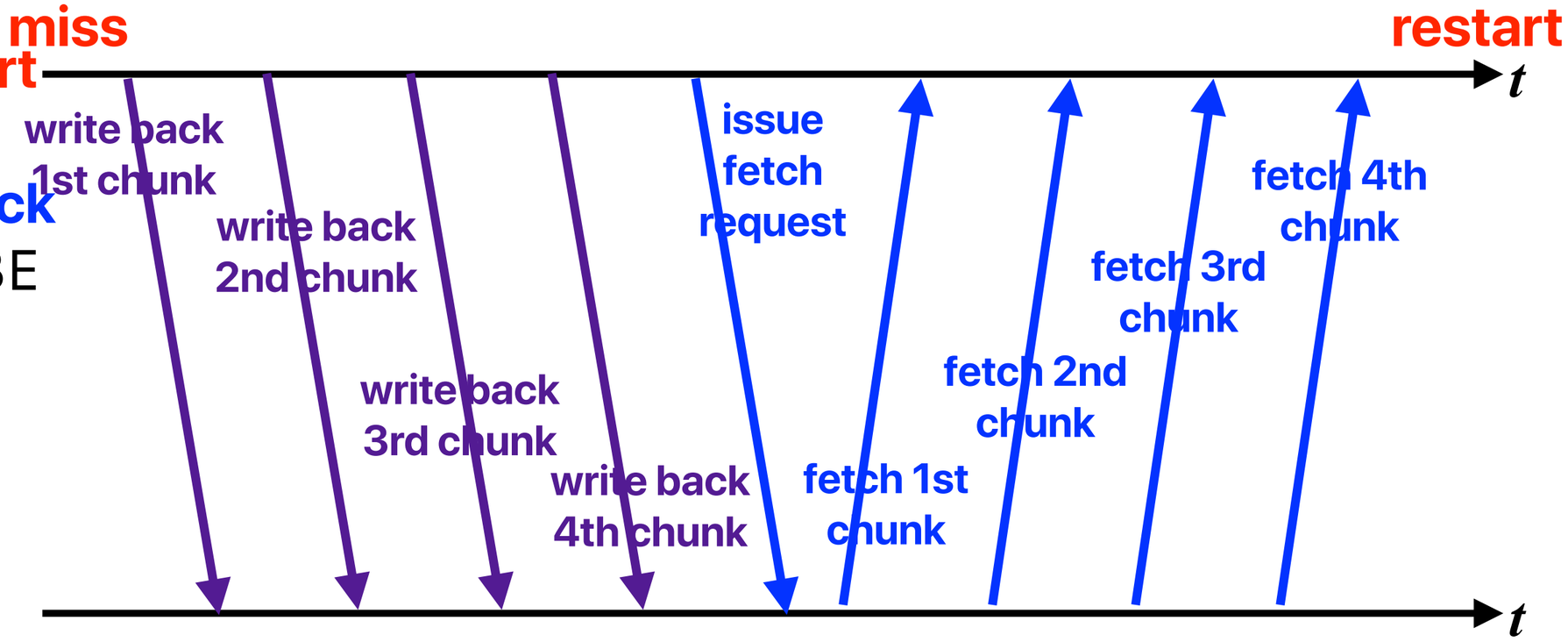


The bandwidth between units is limited



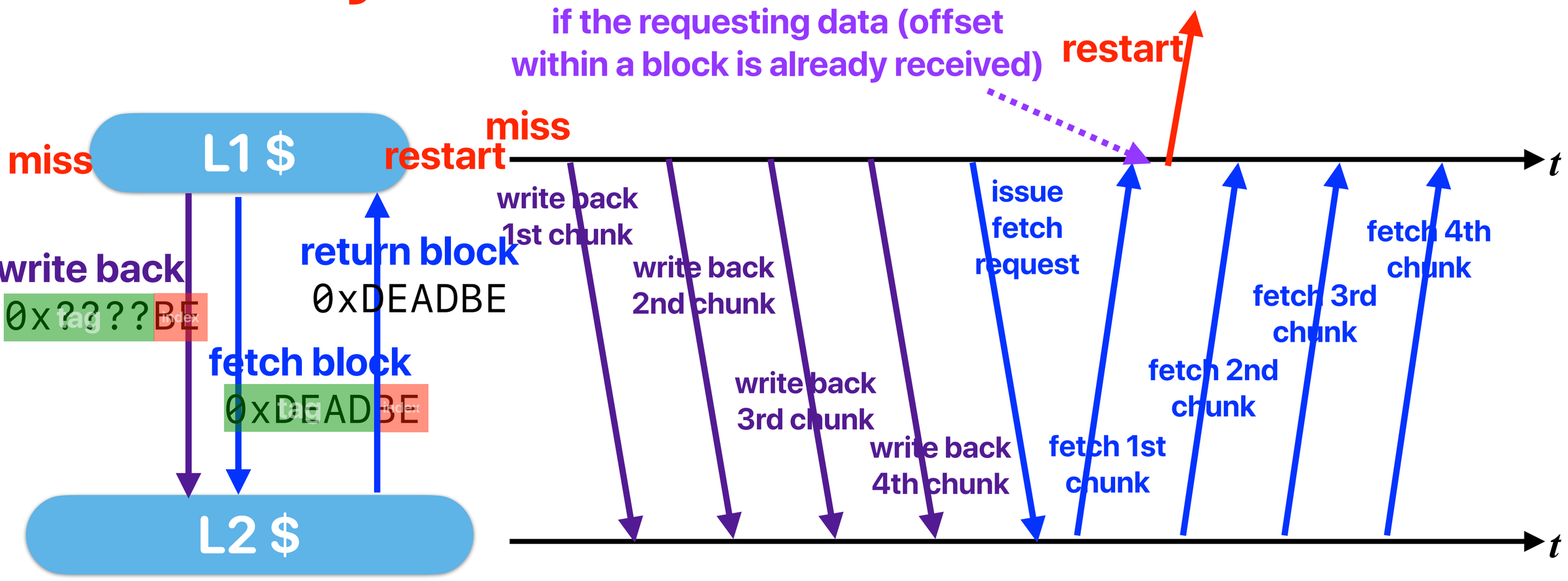
When we handle a miss





assume the bus between L1/L2 only allows a quarter of the cache block go through it

Early Restart and Critical Word First

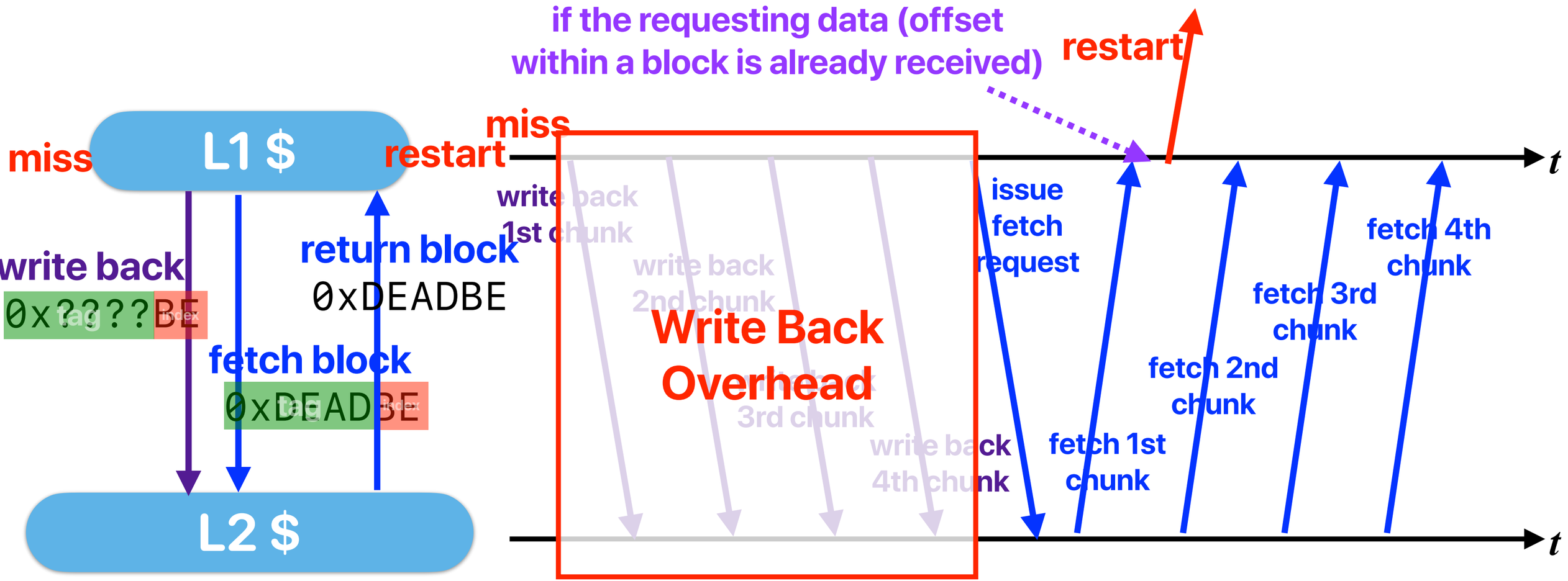


assume the bus between L1/L2 only allows a quarter of the cache block go through it

Early Restart and Critical Word First

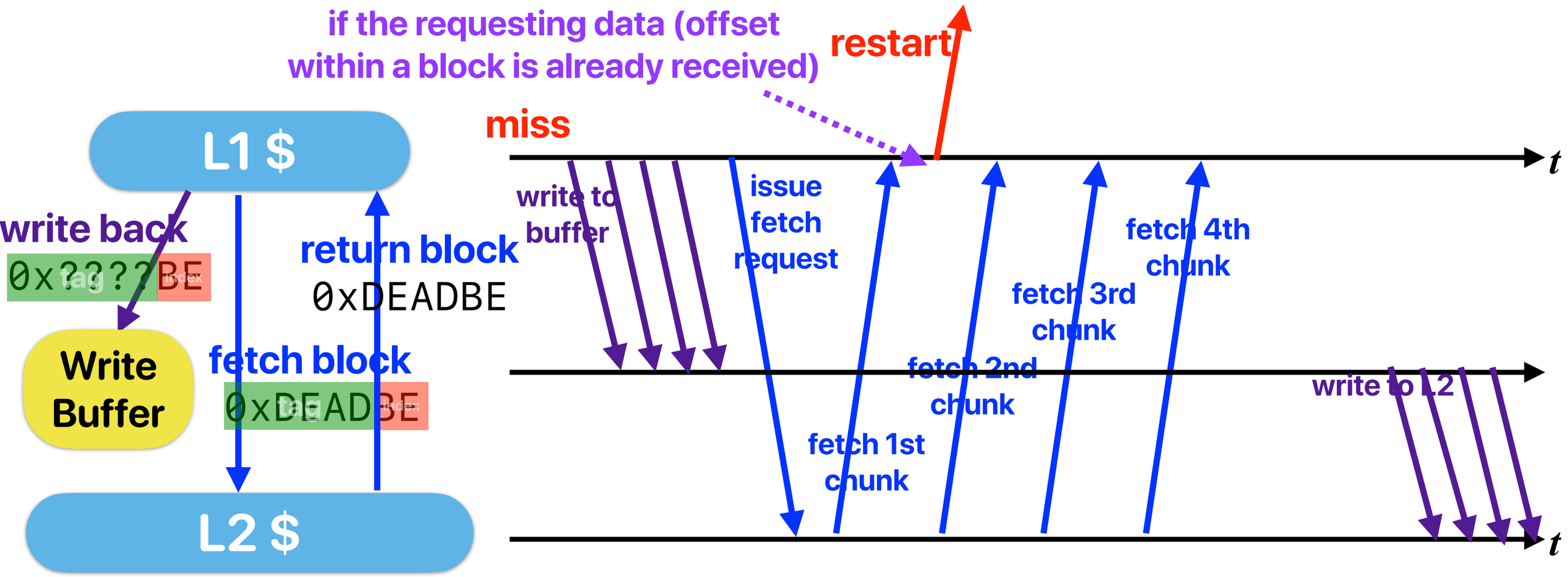
- Don’t wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first
- Most useful with large blocks
- Spatial locality is a problem; often we want the next sequential word soon, so not always a benefit (early restart).

Can we avoid the overhead of writes?



assume the bus between L1/L2 only allows a quarter of the cache block go through it

Write buffer!



assume the bus between L1/L2 only allows a quarter of the cache block go through it

Can we avoid the “double penalty”?

- Every write to lower memory will first write to a small SRAM buffer.
  - store does not incur data hazards, but the pipeline has to stall if the write misses
  - The write buffer will continue writing data to lower-level memory
  - The processor/higher-level memory can response as soon as the data is written to write buffer.
- Write merge
  - Since application has locality, it’s highly possible the evicted data have neighboring addresses. Write buffer delays the writes and allows these neighboring data to be grouped together.

Summary of Architectural Optimizations

- Hardware
  - Prefetch — compulsory miss
  - Write buffer — miss penalty
  - Bank/pipeline — miss penalty
  - Critical word first and early restart — miss panelty

Announcement

- Reading quiz #4 due **next Thursday** before the lecture
- Check your participation grade on https://www.escalab.org/my_grades/ (You may also find the link of the course website)
- Assignment #3 is up. Due in this Thursday (in 2 days)
- Programming assignments are perfectly linked to lectures
  - The solution of programming assignment #2 can be found in the first and the second lecture
  - You should be inspired today for PA #3
  - The code in conv2D_solution.hpp does not perform well — your job to improve or even rewrite that
- Plagiarism:
  - You cannot directly use any code that you found online due to copyright issues
  - Consulting others doesn't mean you are authorized to copy
  - Using online documents without citation is plagiarism
  - Please review the course website and the slide from the first lecture

Computer
Science &
Engineering

203

つづく