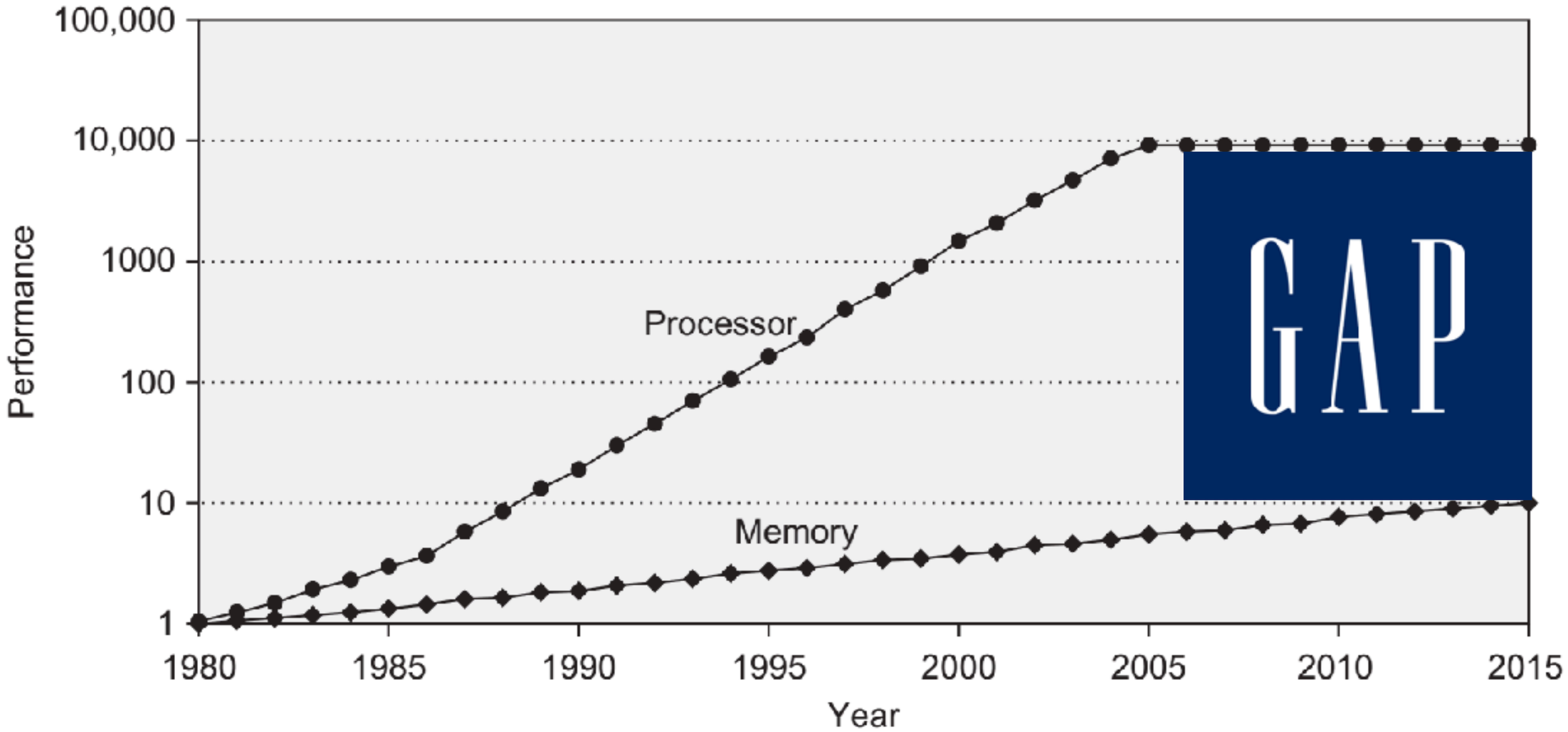
Memory Hierarchy (4): Cache misses and their remedies — the hardware version

Hung-Wei Tseng

Recap: Performance gap between Processor/Memory



Recap: Memory Hierarchy

fastest

< 1ns

a few ns

tens of ns

us/ms

Processor

Processor Core

Registers

SRAM $

DRAM

Storage

TBs

larger

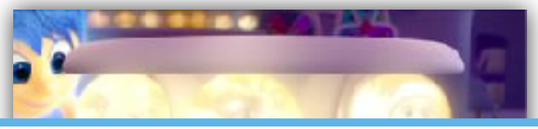
fastest

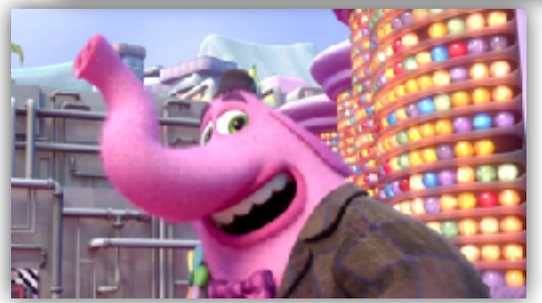
L1 $

L2 $

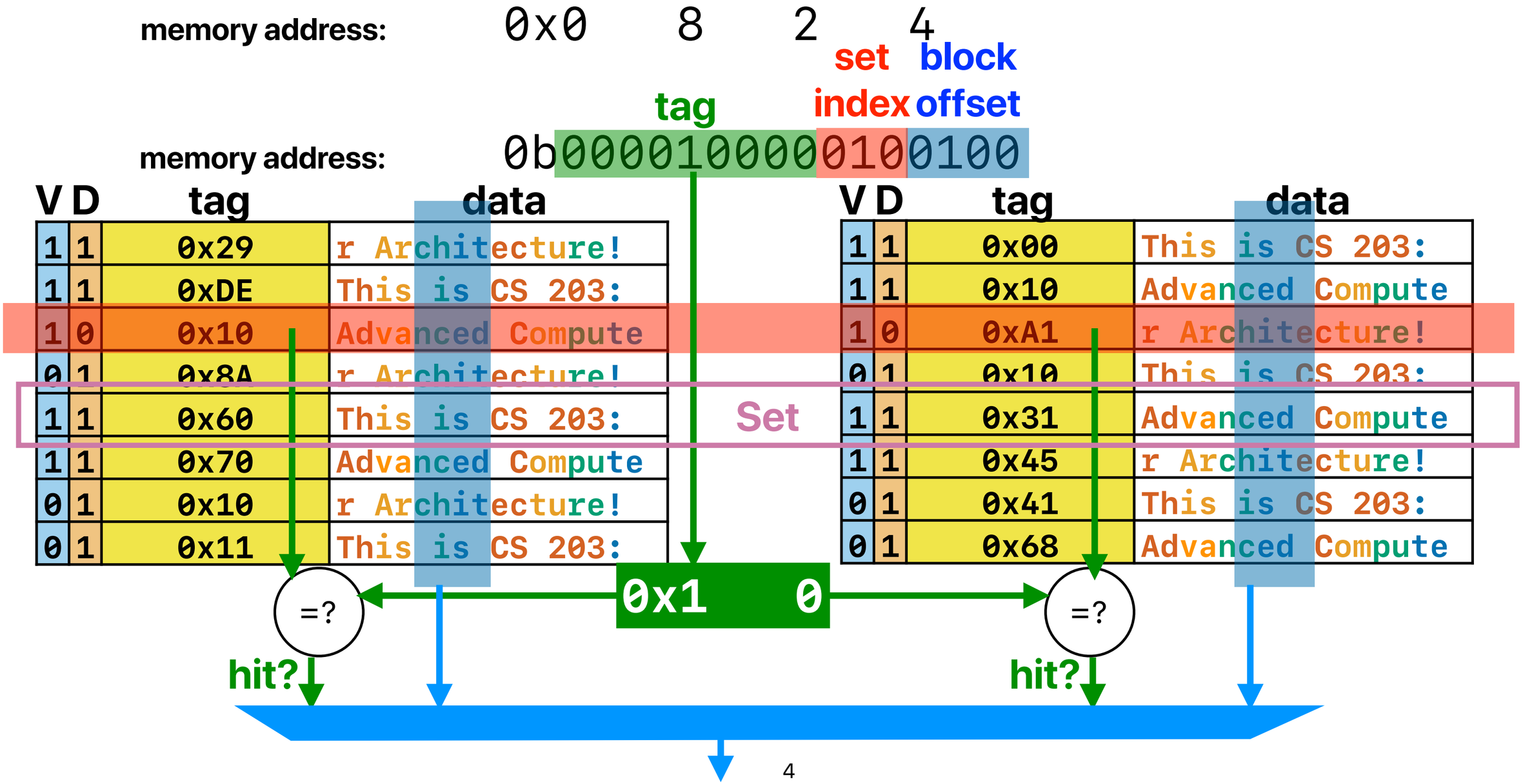
L3 $

larger





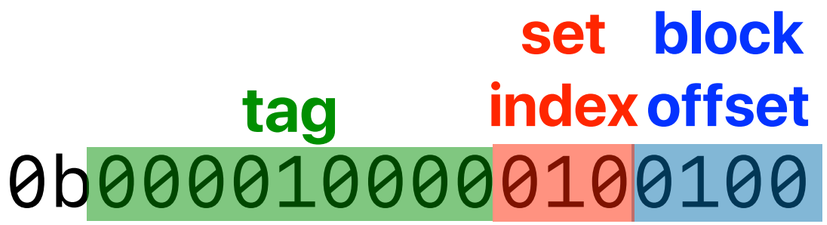
Recap: Way-associative cache



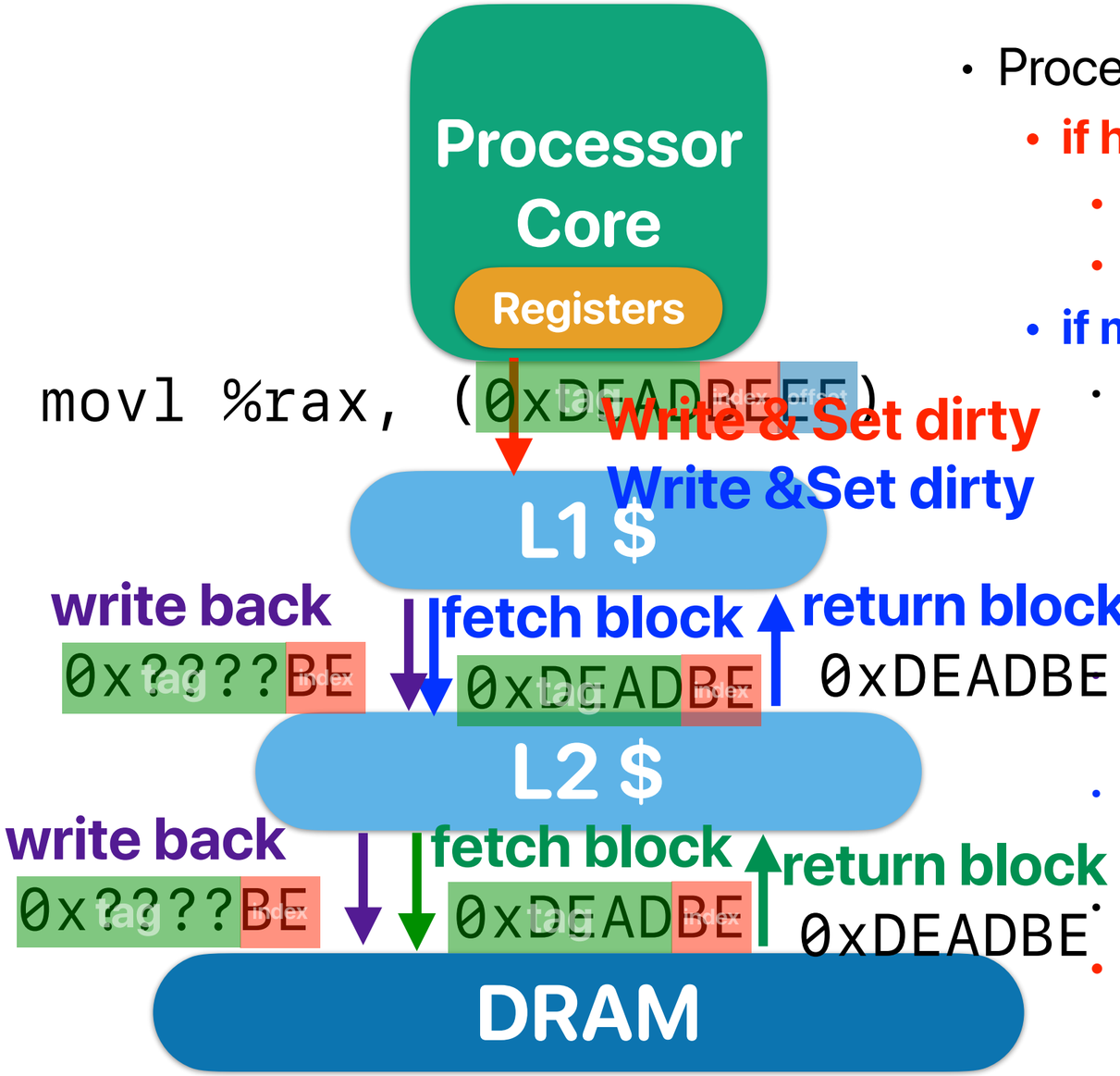
Review: C = ABS

- **C**: **C**apacity in data arrays
- **A**: Way-**A**ssociativity — how many blocks within a set
  - N-way: N blocks in a set, A = N
  - 1 for direct-mapped cache
- **B**: **B**lock Size (Cacheline)
  - How many bytes in a block
- **S**: Number of **S**ets:
  - A set contains blocks sharing the same index
  - 1 for fully associate cache
- number of bits in **b**lock offset — lg(**B**)
- number of bits in **s**et index: lg(**S**)
- tag bits: address_length - lg(S) - lg(B)
  - address_length is 64 bits for 64-bit machine
- $\frac{address}{block\_size} \pmod{S}$ = set index

memory address:



The complete picture



- Processor sends memory access request to L1-$
  - **if hit**
    - **Read - return data**
    - **Write - update & set DIRTY**
  - **if miss**
    - Select a victim block
      - If the target "set" is not full — select an empty/invalidated block as the victim block
      - If the target "set is full — select a victim block using some policy
      - LRU is preferred — to exploit temporal locality!
    - If the victim block is "dirty" & "valid"
      - **Write back** the block to lower-level memory hierarchy
    - Fetch the requesting block from lower-level memory hierarchy and place in the victim block
    - If write-back or fetching causes any miss, repeat the same process
  - **Present the write "ONLY" in L1 and set DIRTY**

Review: 3Cs of misses

- Compulsory miss
  - Cold start miss. First-time access to a block
- Capacity miss
  - The working set size of an application is bigger than cache size
- Conflict miss
  - Required data replaced by block(s) mapping to the same set
  - Similar collision in hash

Review: Software Optimizations

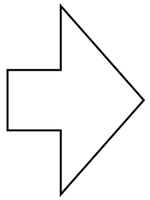
- Data layout — capacity miss, conflict miss, compulsory miss
- Blocking/tiling — capacity miss, conflict miss
- Loop fission — conflict miss — when $ has limited way associativity
- Loop fusion — capacity miss — when $ has enough way associativity
- Loop interchange — conflict/capacity miss

Outline

- Software optimizations for cache performance (cont.)
- Architectural support for optimizing cache performance

Matrix Transpose

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
              c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```



```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}
```

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

Block + Transpose

```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}


for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

Block + Transpose

```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}


for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

Summary of Software Optimizations

- Data layout — capacity miss, conflict miss, compulsory miss
- Blocking/tiling — capacity miss, conflict miss
- Loop fission — conflict miss — when $ has limited way associativity
- Loop fusion — capacity miss — when $ has enough way associativity
- Loop interchange — conflict/capacity miss

Basic Hardware Optimization in Improving 3Cs

https://www.pollev.com/hungweitseng close in 1:30

3Cs and A, B, C

- Regarding 3Cs: compulsory, conflict and capacity misses and A, B, C: associativity, block size, capacity
  How many of the following are correct?
  - ① Increasing associativity can reduce conflict misses
  - ② Increasing associativity can reduce hit time
  - ③ Increasing block size can increase the miss penalty
  - ④ Increasing block size can reduce compulsory misses
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

3Cs and A, B, C

- Regarding 3Cs: compulsory, conflict and capacity misses and A, B, C: associativity, block size, capacity
  How many of the following are correct?
  - ① Increasing associativity can reduce conflict misses
  - ② Increasing associativity can reduce hit time
  - ③ Increasing block size can increase the miss penalty
  - ④ Increasing block size can reduce compulsory misses

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Increases hit time because your data array is larger (longer time to fully charge your bit-lines)

You need to fetch more data for each miss

You bring more into the cache when a miss occurs

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

https://www.pollev.com/hungweitseng close in 1:30

Which of the following schemes can help NVIDIA Tegra?

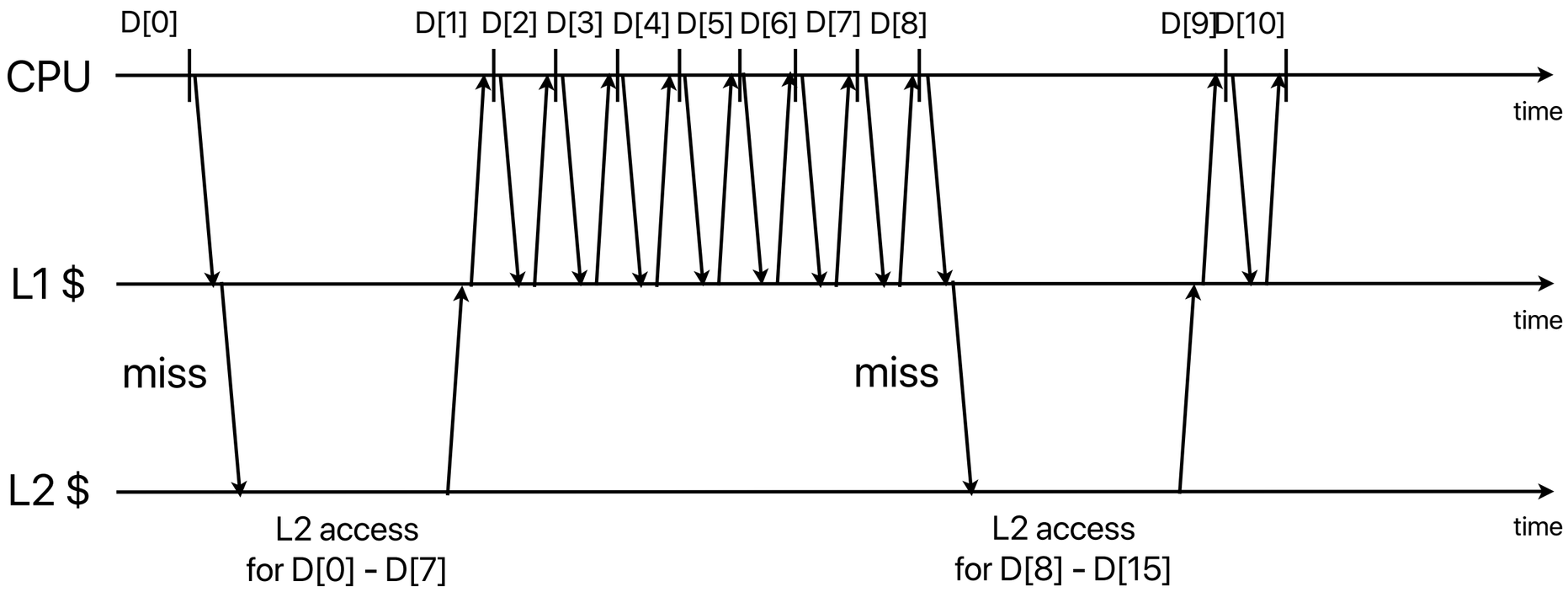
- How many of the following schemes mentioned in “improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers” would help NVIDIA’s Tegra for the code in the previous slide?
  - ① Missing cache
  - ② Victim cache
  - ③ Prefetch
  - ④ Stream buffer
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

```
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]
}
```

Prefetching

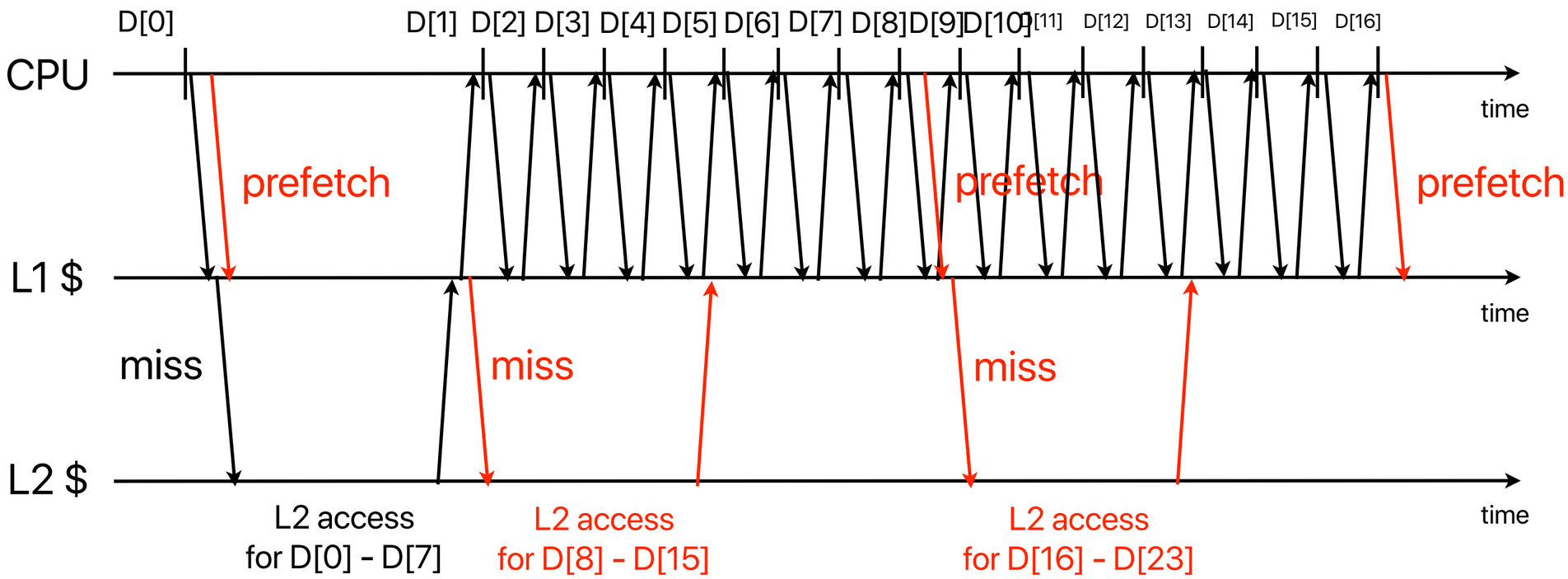
Characteristic of memory accesses

```
for(i = 0;i < 1000000; i++) {
     D[i] = rand();
}
```



Prefetching

```
for(i = 0;i < 1000000; i++) {
      D[i] = rand();
      // prefetch D[i+8] if i % 8 == 0
}
```



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

https://www.pollev.com/hungweitseng close in 1:30

Where can prefetch work effectively?

- How many of the following code snippet can “prefetching” effectively help improving performance?

(1)
```
while(node){
    node = node->next;
}
```

(2)
```
while(++i<100000)
    a[i]=rand();
```

(3)
```
while (root != NULL){
        if (key > root->data)
            root = root->right;

        else if (key < root->data)
            root = root->left;
        else
            return true;
}
```

(4)
```
    for (i = 0; i < 65536; i++) {
      mix_i = ((i * 167) + 13) & 65536;
      results[mix_i]++;
    }
```

A. 0
B. 1
C. 2
D. 3
E. 4

Where can prefetch work effectively?

- How many of the following code snippet can “prefetching” effectively help improving performance?

(1)
```
while(node){
    node = node->next;
}
```
— where the next pointing to is hard to predict

(2)
```
while(++i<100000)
    a[i]=rand();
```

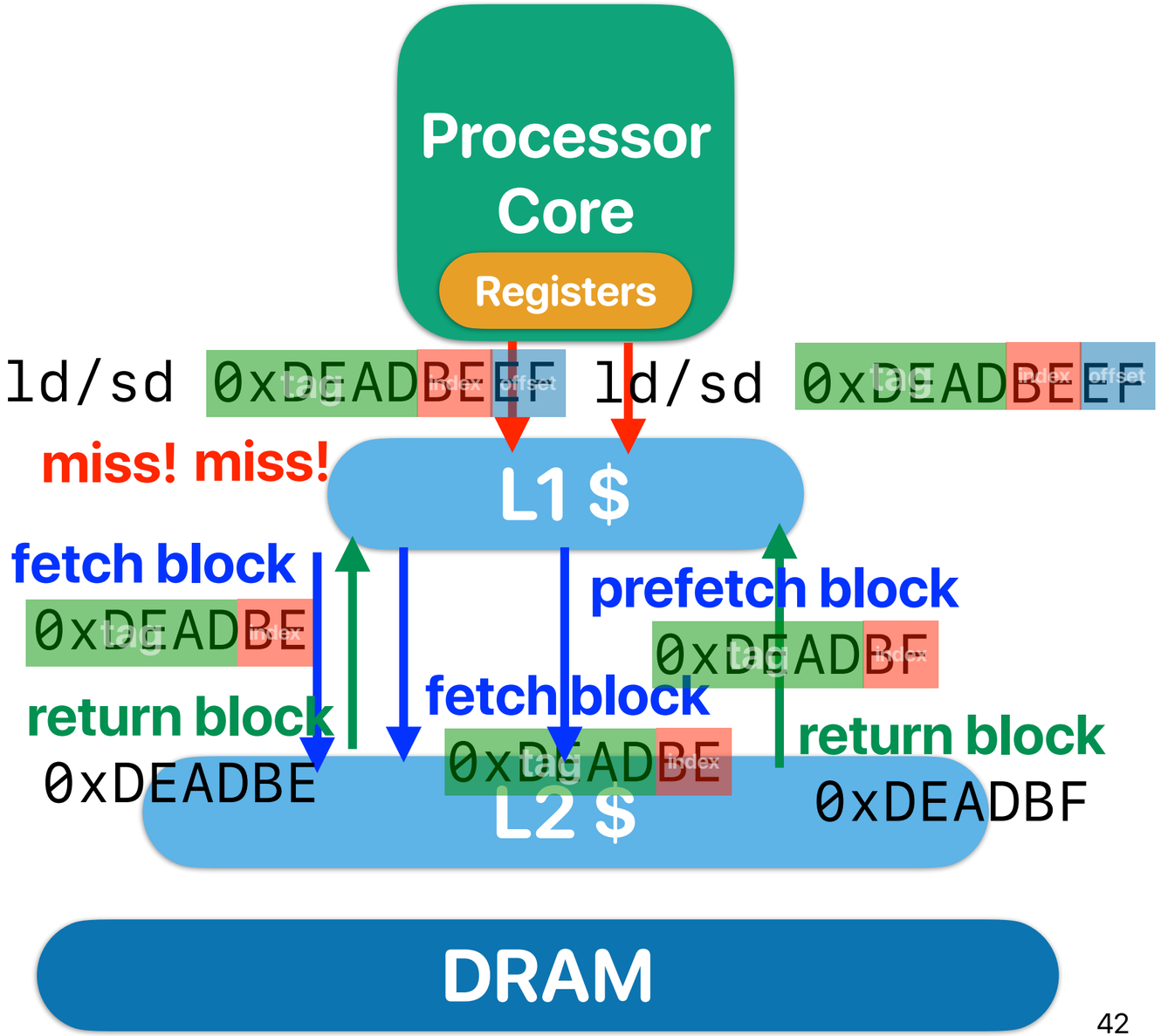
(3)
```
while (root != NULL){
        if (key > root->data)
            root = root->right;

        else if (key < root->data)
            root = root->left;
        else
            return true;
}
```
— where the next node is also hard to predict

(4)
```
    for (i = 0; i < 65536; i++) {
      mix_i = ((i * 167) + 13) & 65536;
      results[mix_i]++;
    }
```
— the stride to the next element is hard to predict…

A. 0
B. 1
C. 2
D. 3
E. 4

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

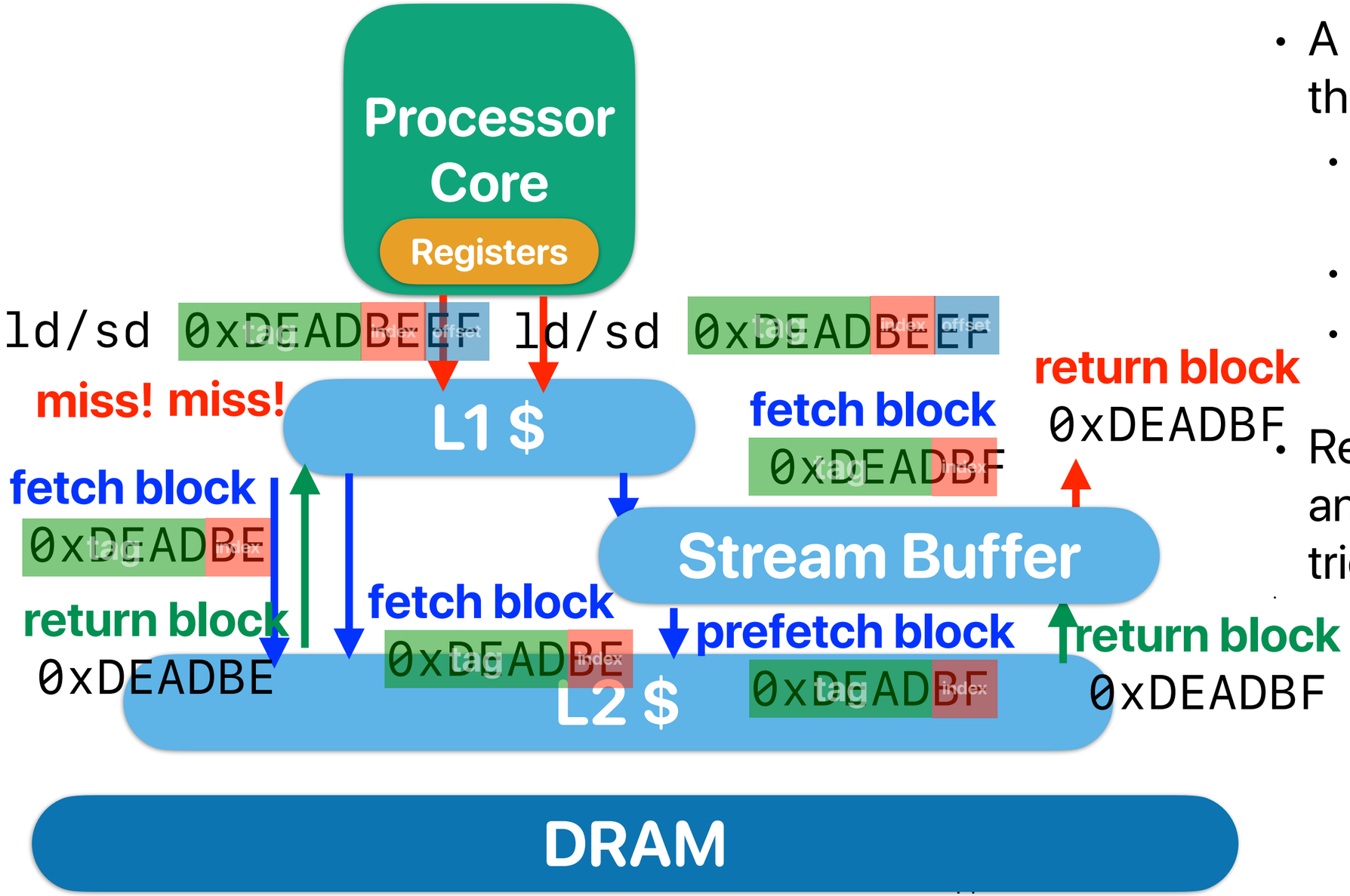
tag index offset

	Address (Hex)	Address in binary	Tag	Index	Hit? Miss?	Replace?	Prefetch
a[0]	0x10000	0b00010000000000000000	0x8	0x0	Miss		a[8–15]
b[0]	0x20000	0b00100000000000000000	0x10	0x0	Miss		b[8–15]
c[0]	0x30000	0b00110000000000000000	0x18	0x0	Miss		c[8–15]
d[0]	0x40000	0b01000000000000000000	0x20	0x0	Miss		d[8–15]
e[0]	0x50000	0b01010000000000000000	0x28	0x0	Miss	a[0–7]	e[8–15]
a[1]	0x10008	0b00010000000000001000	0x8	0x0	Miss	b[0–7]	
b[1]	0x20008	0b00100000000000001000	0x10	0x0	Miss	c[0–7]	
c[1]	0x30008	0b00110000000000001000	0x18	0x0	Miss	d[0–7]	
d[1]	0x40008	0b01000000000000001000	0x20	0x0	Miss	e[0–7]	
e[1]	0x50008	0b01010000000000001000	0x28	0x0	Miss	a[0–7]	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

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

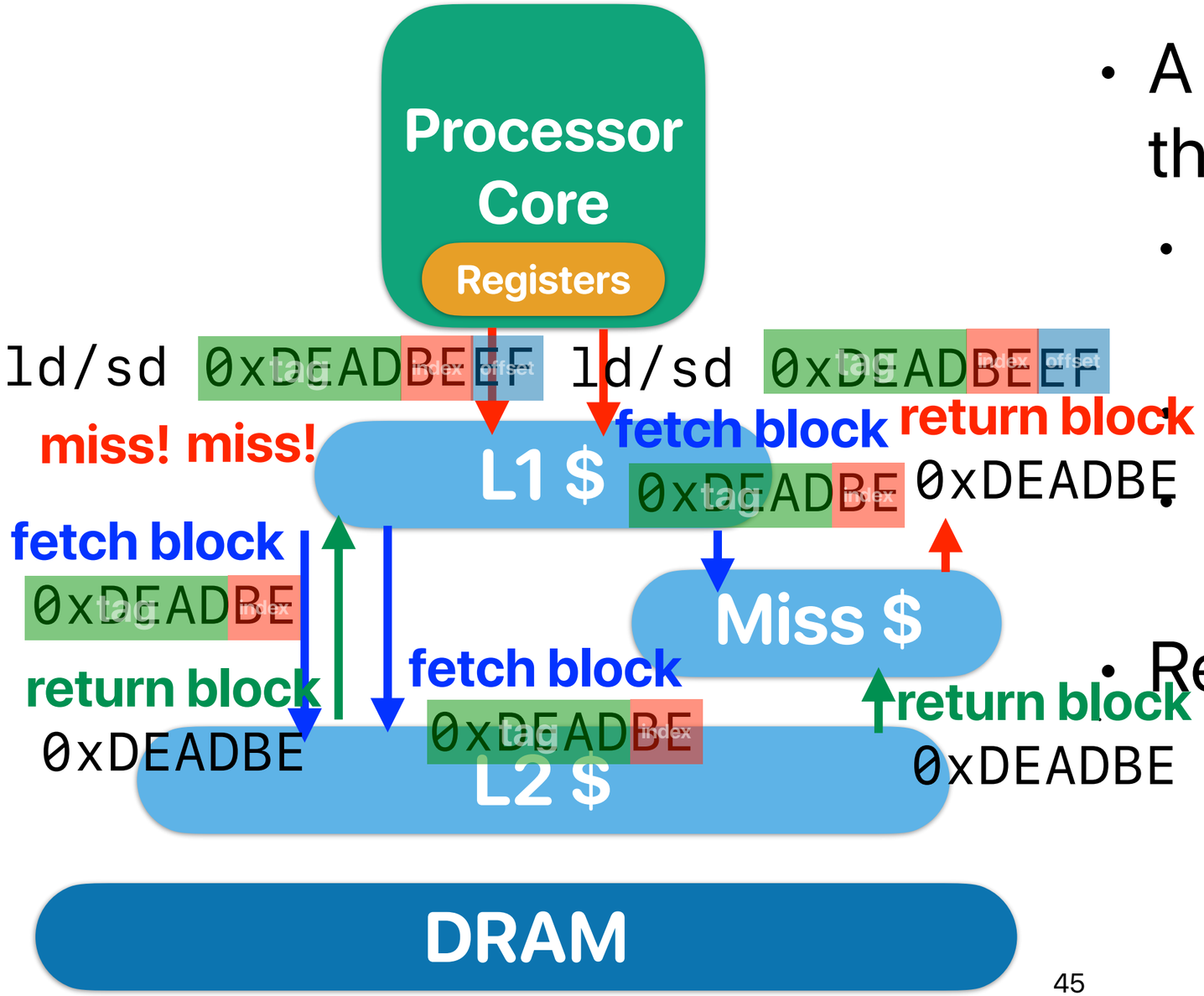
100% miss rate!

Stream buffer



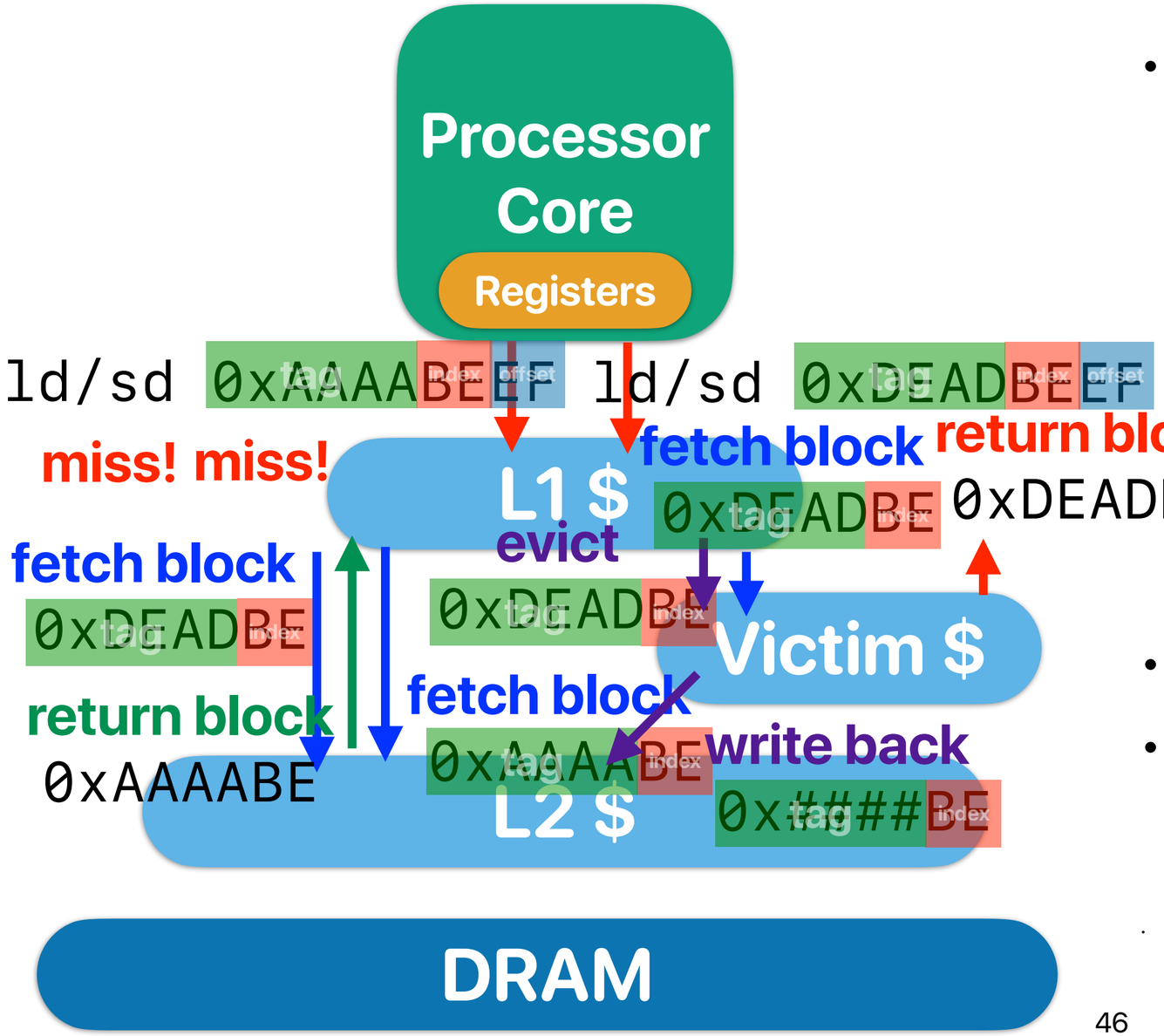
- A small cache that captures the prefetched blocks
  - Can be built as fully associative since it's small
  - Consult when there is a miss
  - Retrieve the block if found in the stream buffer
- Reduce compulsory misses and avoid conflict misses triggered by prefetching

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

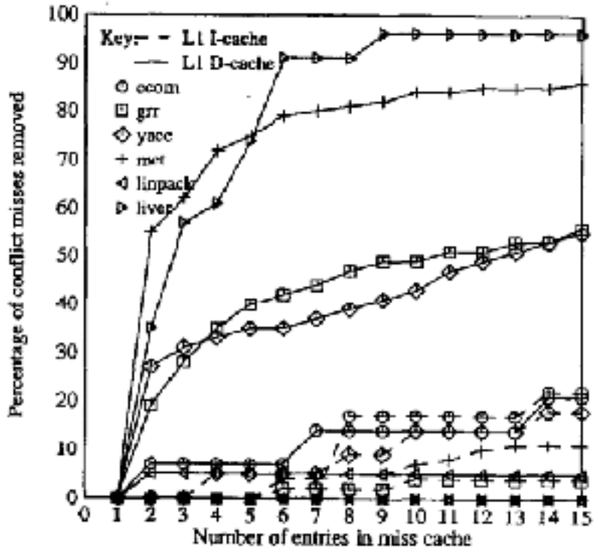


Figure 3-3: Conflict misses removed by miss caching

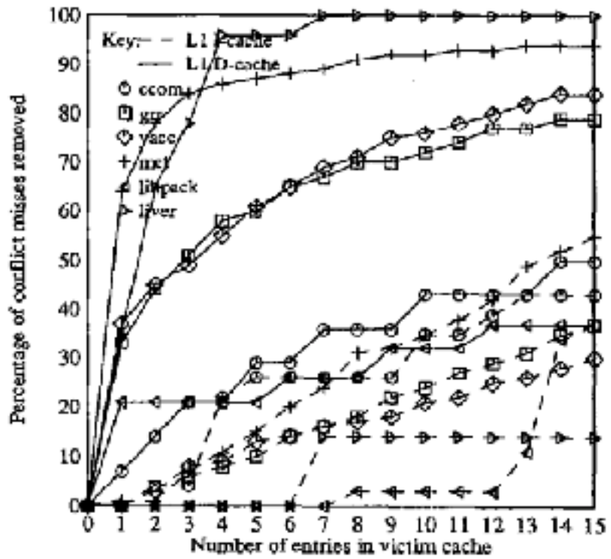


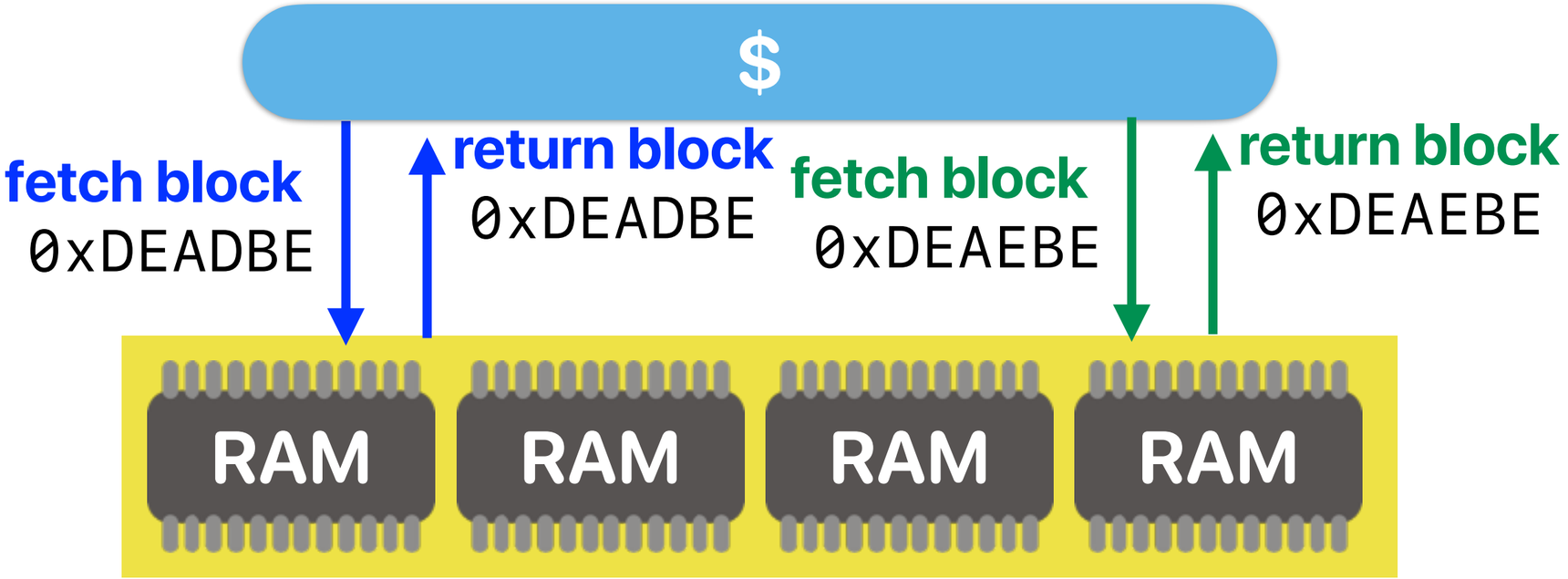
Figure 3-5: Conflict misses removed by victim caching

Which of the following schemes can help Tegra?

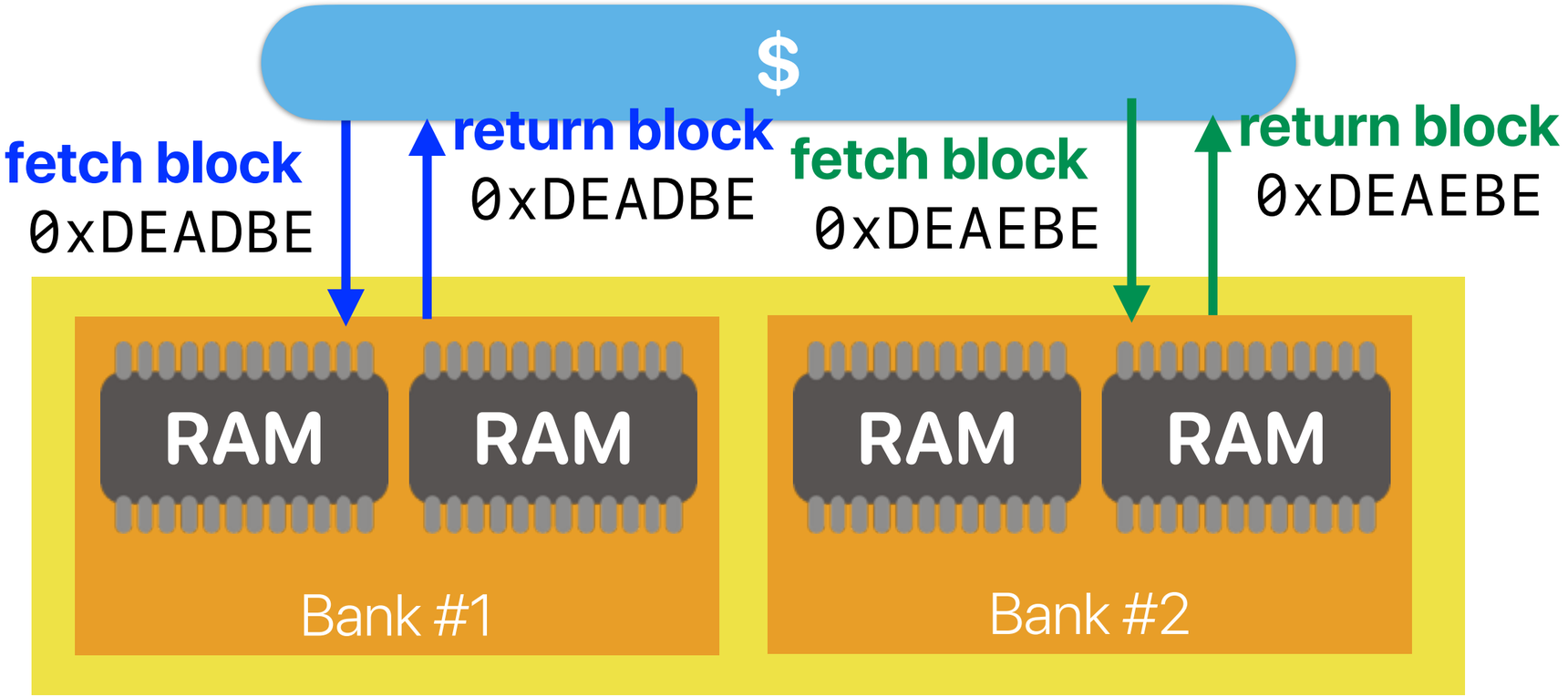
- How many of the following schemes mentioned in "improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers" would help NVIDIA's Tegra for the code in the previous slide?
  - ① Missing cache — help improving conflict misses
  - ② Victim cache — help improving conflict misses
  - ③ Prefetch — improving compulsory misses , but can potentially hurt, if we did not do it right
  - ④ Stream buffer — only help improving compulsory misses
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Advanced Hardware Techniques in Improving Memory Performance

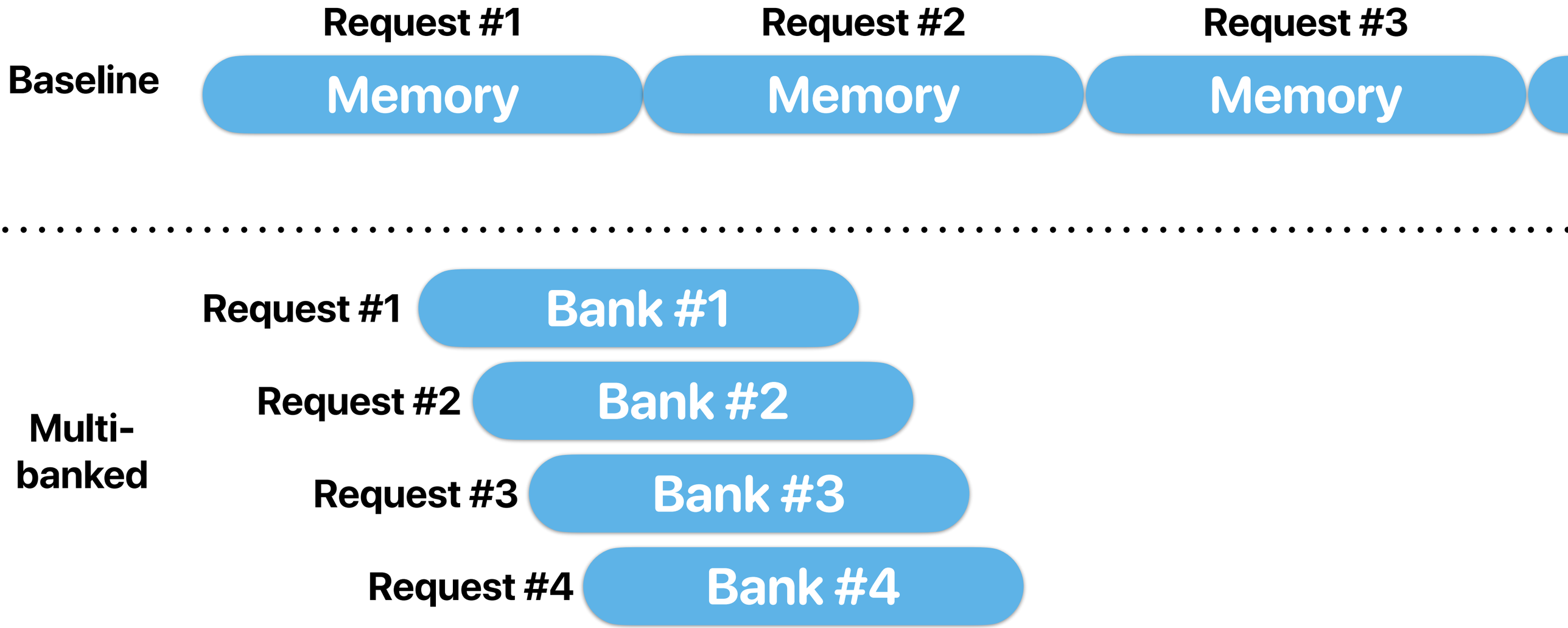
Blocking cache



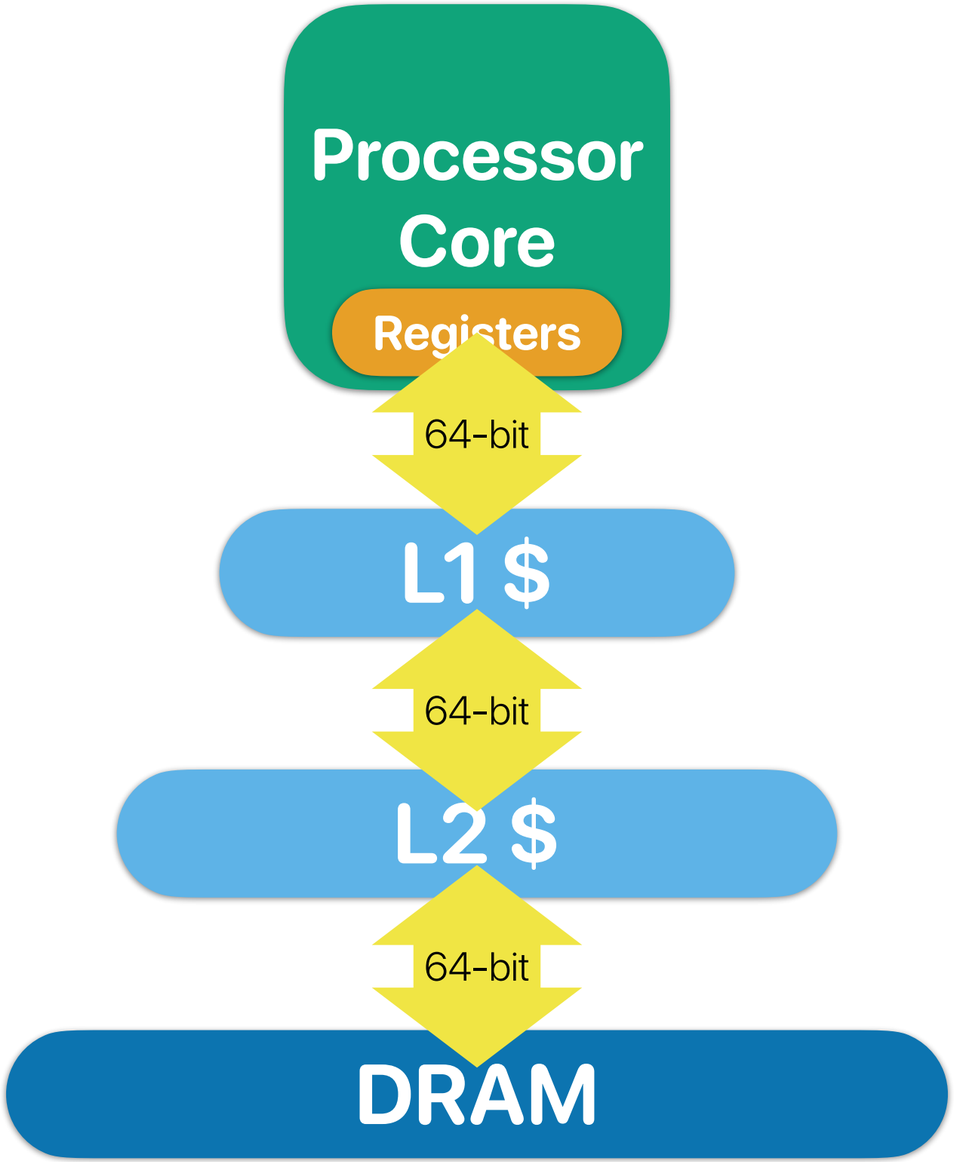
Multibanks & non-blocking caches



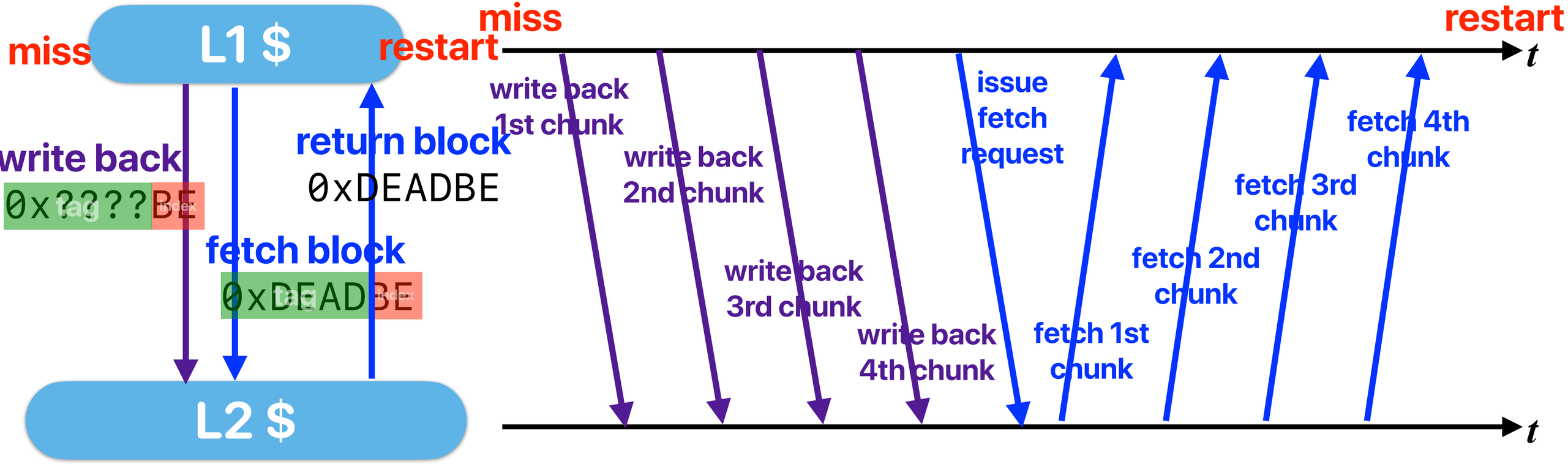
Pipelined access and multi-banked caches



The bandwidth between units is limited

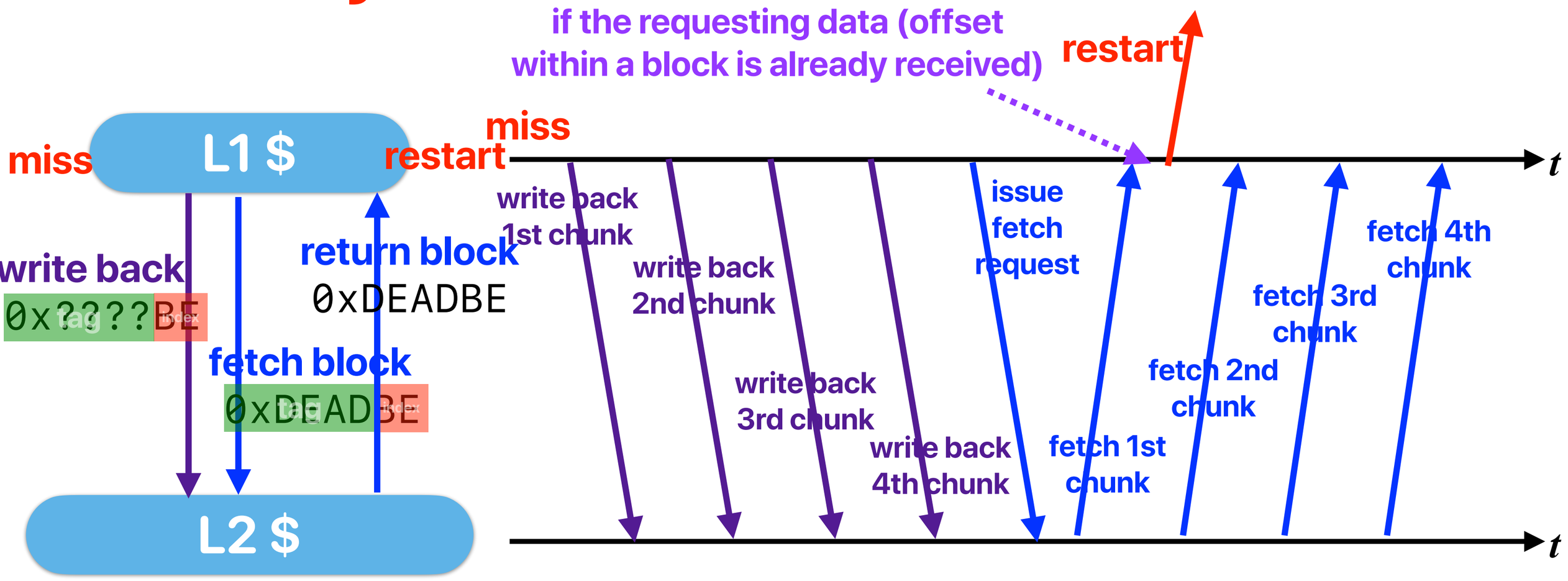


When we handle a miss



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

Early Restart and Critical Word First

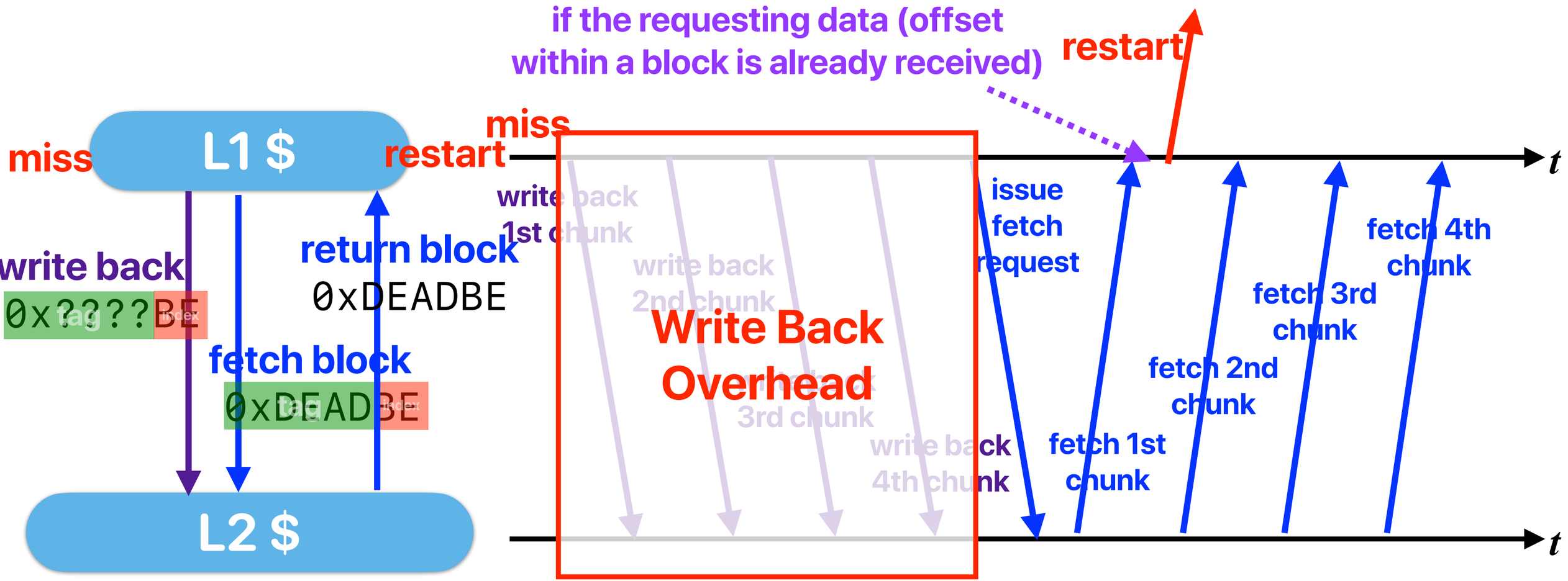


assume the bus between L1/L2 only allows a quarter of the cache block go through it

Early Restart and Critical Word First

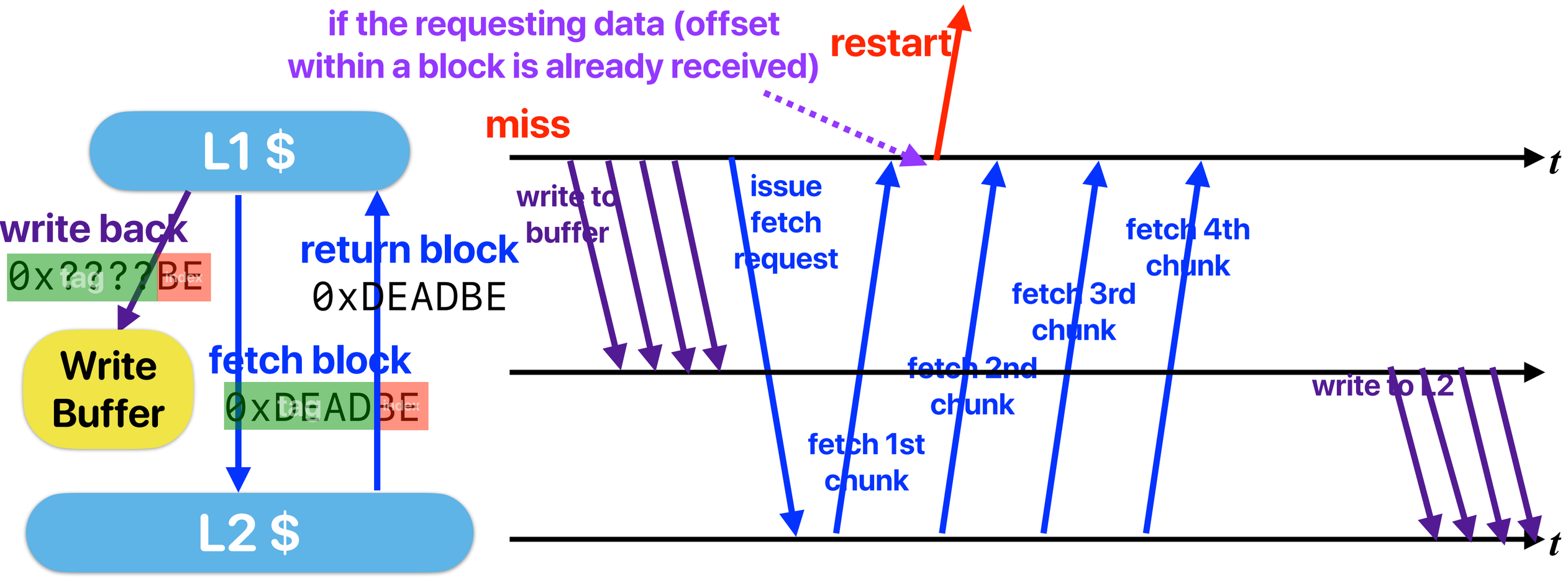
- Don’t wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let 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 Optimizations

- Regarding the following cache optimizations, how many of them would help improve miss rate?
  - ① Non-blocking/pipelined/multibanked cache **Miss penalty/Bandwidth**
  - ② Critical word first and early restart **Miss penalty**
  - ③ Prefetching **Miss rate (compulsory)**
  - ④ Write buffer **Miss penalty**

  A. 0

  B. 1

  C. 2

  D. 3

  E. 4

Midterm Related

- Closed-book, closed-note
- No cheatsheet, no scratch paper, no outside materials
- You may bring and use your-own calculator — borrowing from others is not allowed
- 80-minute in person
- Will release midterm sample questions together with the slides on Thursday

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