Memory Hierarchy (3)

Hung-Wei Tseng

Recap: Memory Hierarchy

fastest

< 1ns

a few ns

tens of ns

us/ms

Processor

Processor Core

Registers

SRAM $

DRAM

Storage

TBs

larger

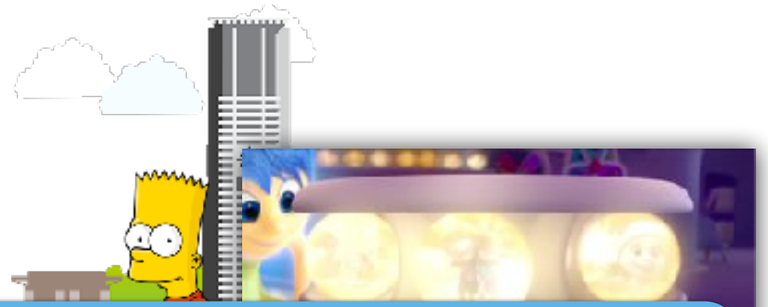
fastest

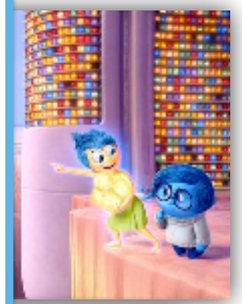
L1 $

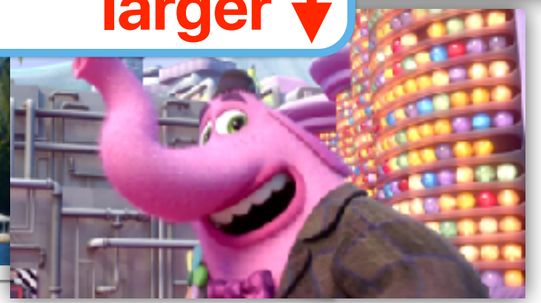
L2 $

L3 $

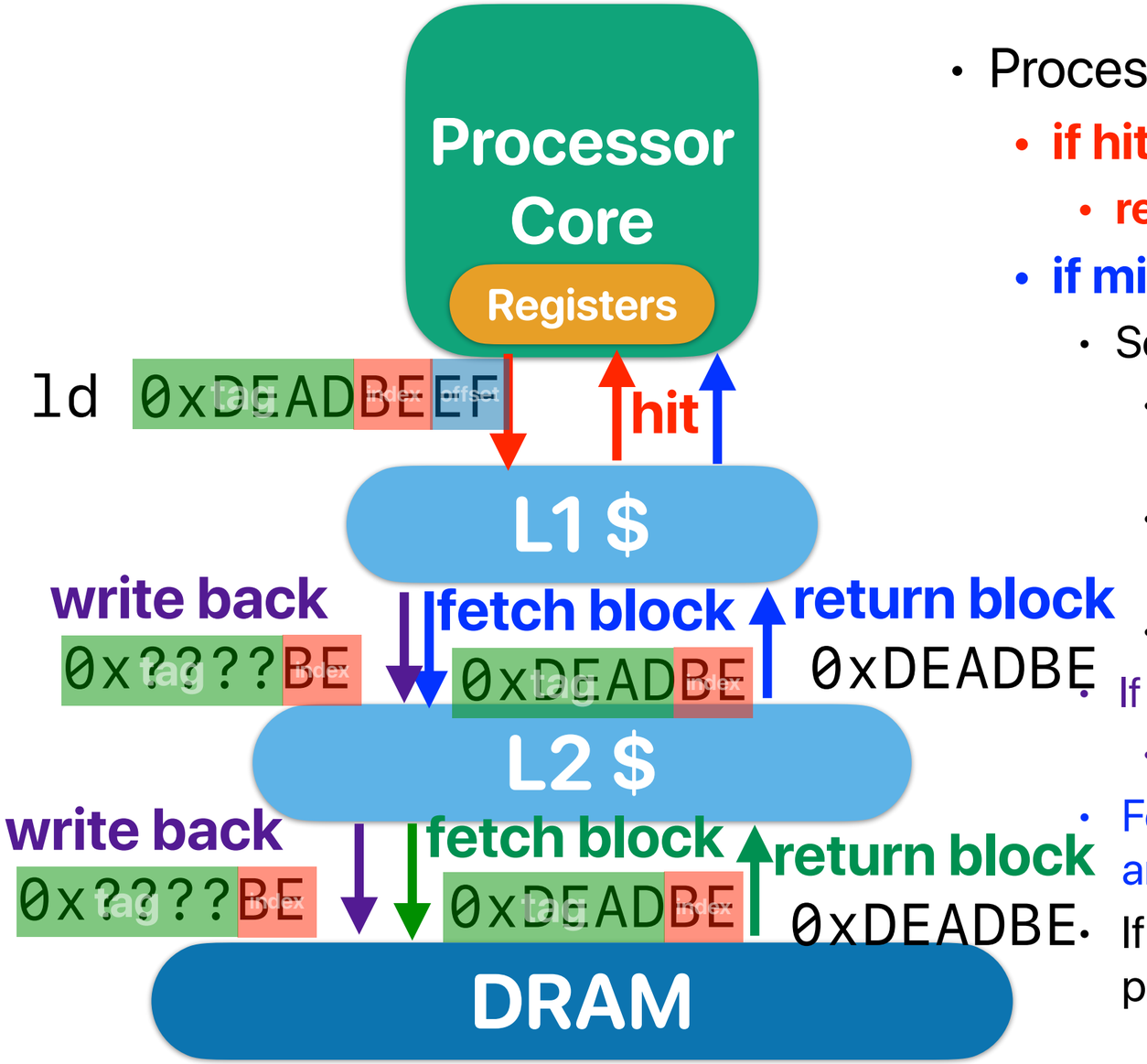
larger





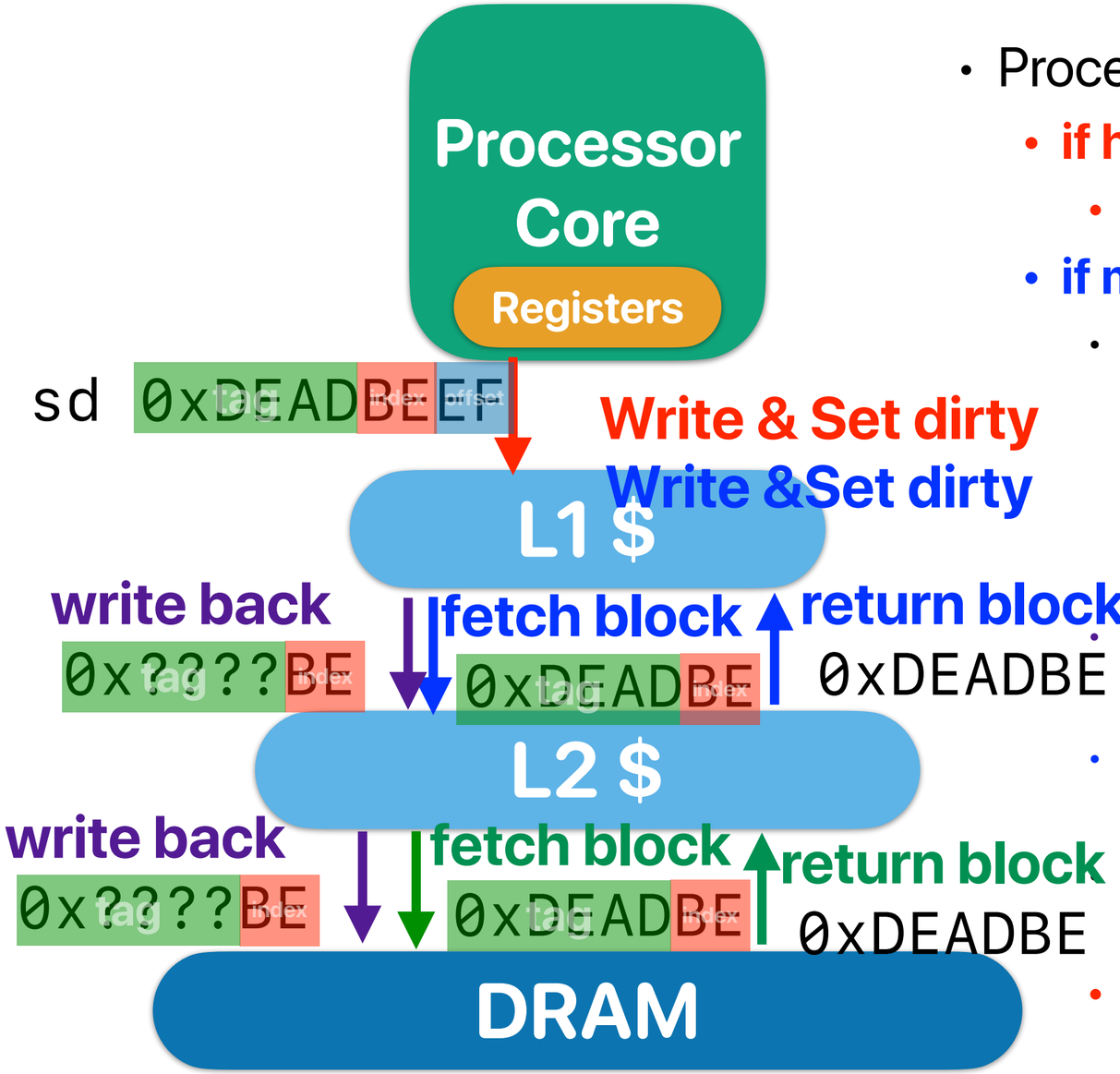


What happens when we read data



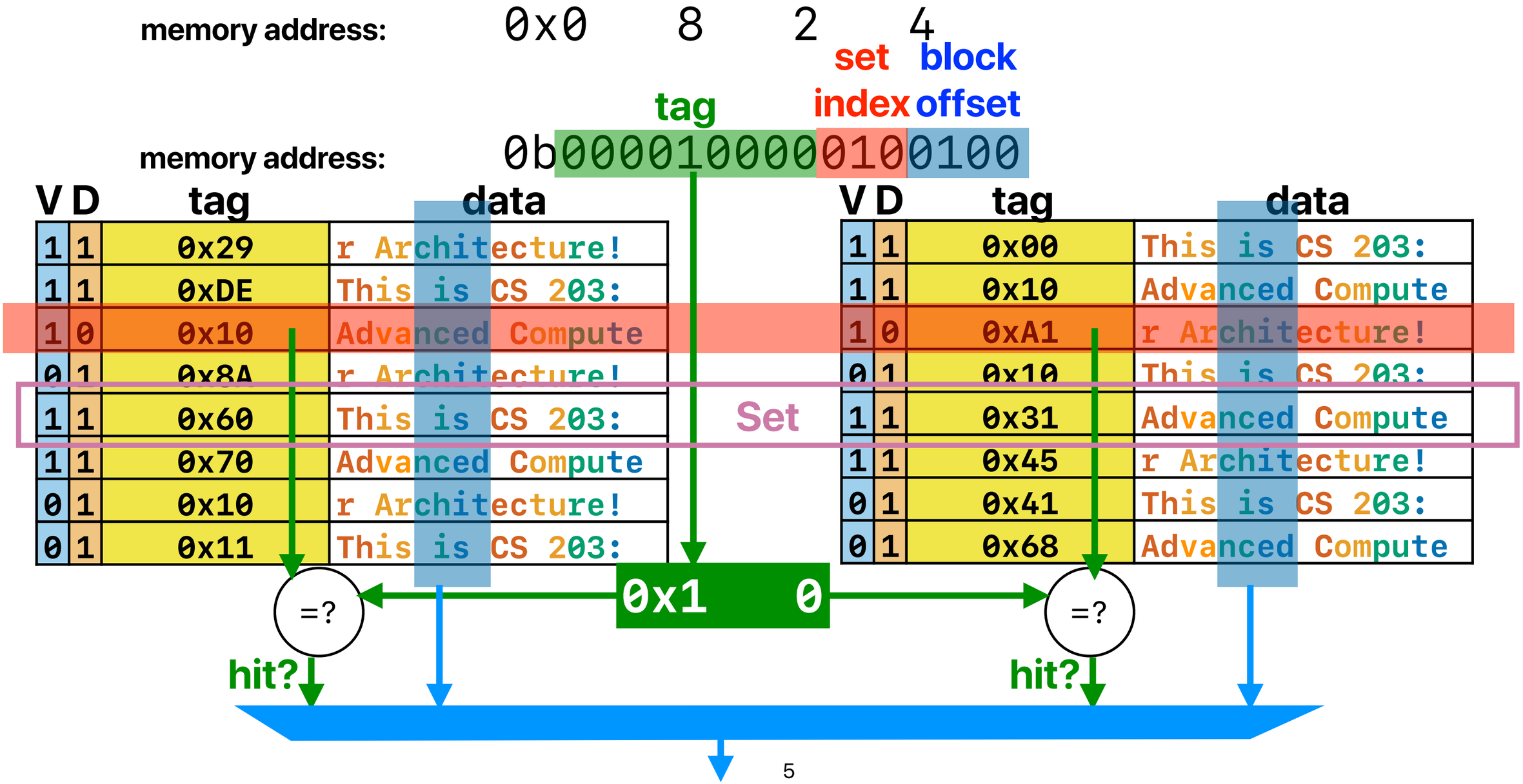
- Processor sends load request to L1-$
  - **if hit**
    - **return data**
  - **if miss**
    - Select a victim block
      - If the target "set" is not full — select an empty/invalidated block as the victim block
      - If the target "set is full — select a victim block using some policy
      - 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

What happens when we write data



- Processor sends load request to L1-$
  - **if hit**
    - **return data — 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**

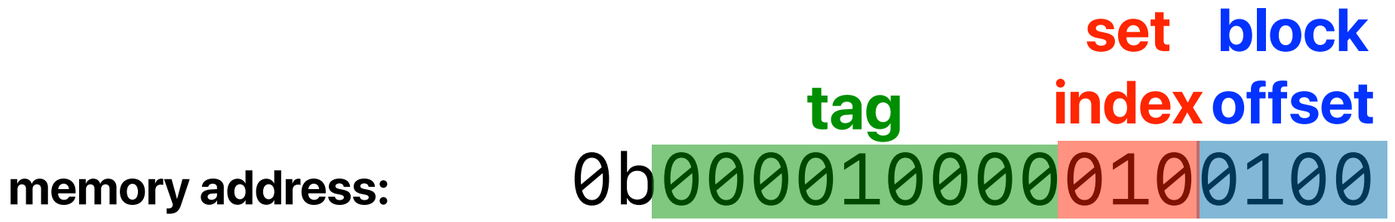
Recap: Way-associative cache



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

Corollary of C = ABS



- 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 32 bits for 32-bit machine
- (address / block_size) % S = set index

AMD Phenom II

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 32-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

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

A. 6.25%

B. 56.25%

C. 66.67%

D. 68.75%

E. 100%

C = ABS

64KB = 2 * 64 * S

S = 512

offset = lg(64) = 6 bits

index = lg(512) = 9 bits

tag = 64 - lg(512) - lg(64) = 49 bits

AMD Phenom II

100% miss rate!

- Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    c[i] = a[i] + b[i]; /*load a[i], load b[i], store c[i]*/
```

C = ABS
64KB = 2 * 64 * S
S = 512
offset = lg(64) = 6 bits
index = lg(512) = 9 bits
tag = the rest bits

	address in hex	address in binary (tag / index / offset)	tag	index	hit? miss?
load a[0]	0x20000	0b10 0000 0000 0000 0000	0x4	0	miss
load b[0]	0x30000	0b11 0000 0000 0000 0000	0x6	0	miss
store c[0]	0x10000	0b01 0000 0000 0000 0000	0x2	0	miss, evict 0x4
load a[1]	0x20004	0b10 0000 0000 0000 0100	0x4	0	miss, evict 0x6
load b[1]	0x30004	0b11 0000 0000 0000 0100	0x6	0	miss, evict 0x2
store c[1]	0x10004	0b01 0000 0000 0000 0100	0x2	0	miss, evict 0x4
⋮	⋮	⋮	⋮	⋮	⋮
load a[15]	0x2003C	0b10 0000 0000 0011 1100	0x4	0	miss, evict 0x6
load b[15]	0x3003C	0b11 0000 0000 0011 1100	0x6	0	miss, evict 0x2
store c[15]	0x1003C	0b01 0000 0000 0011 1100	0x2	0	miss, evict 0x4
load a[16]	0x20040	0b10 0000 0000 0100 0000	0x4	1	miss
load b[16]	0x30040	0b11 0000 0000 0100 0000	0x6	1	miss
store c[16]	0x10040	0b01 0000 0000 0100 0000	0x2	1	miss, evict 0x4

Outline

- Causes of cache misses: 3Cs
- Hardware optimizations for cache performance

intel Core i7

- D-L1 Cache configuration of intel Core i7 processor
  - Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

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

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

intel Core i7

- D-L1 Cache configuration of intel Core i7 processor
  - Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

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

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

C = ABS

32KB = 8 * 64 * S

S = 64

offset = lg(64) = 6 bits

index = lg(64) = 6 bits

tag = 64 - lg(64) - lg(64) = 52 bits

intel Core i7

C = ABS

32KB = 8 * 64 * S

S = 64

offset = lg(64) = 6 bits

index = lg(64) = 6 bits

tag = 64 - lg(64) - lg(64) = 52 bits

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
{
    c[i] = a[i] + b[i]; /*load a[i], load b[i], store c[i]*/
}
```

	address	tag	index	?
load a[0]	0x20000	0x20	0	miss
load b[0]	0x30000	0x30	0	miss
store c[0]	0x10000	0x10	0	miss
load a[1]	0x20004	0x20	0	hit
load b[1]	0x30004	0x30	0	hit
store c[1]	0x10004	0x10	0	hit
⋮	⋮	⋮	⋮	⋮
load a[15]	0x2003C	0x20	0	hit
load b[15]	0x3003C	0x30	0	hit
store c[15]	0x1003C	0x10	0	hit
load a[16]	0x20040	0x20	1	miss
load b[16]	0x30040	0x30	1	miss
store c[16]	0x1003C	0x10	1	miss

32*3/(512*3) = 1/16 = 6.25% (93.75% hit rate!)

intel Core i7

- D-L1 Cache configuration of intel Core i7 processor
  - Size 32KB, 8-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

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

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

C = ABS

32KB = 8 * 64 * S

S = 64

offset = lg(64) = 6 bits

index = lg(64) = 6 bits

tag = 64 - lg(64) - lg(64) = 52 bits

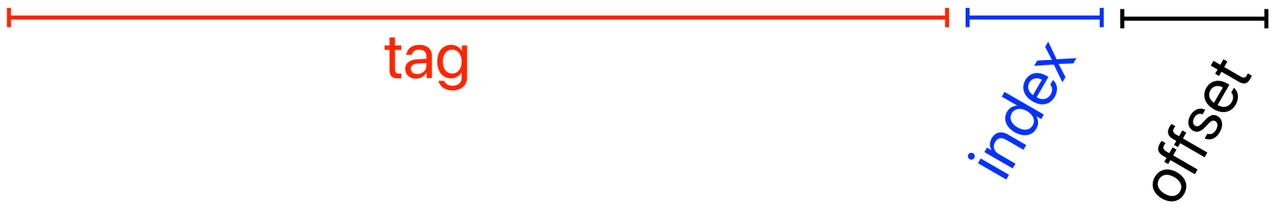
Cause of cache misses

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

Simulate a direct-mapped cache

- Consider a direct mapped (1-way) cache with 256 bytes total capacity, a block size of 16 bytes, and the application repeatedly reading the following memory addresses:
  - 0b1000000000, 0b1000001000, 0b1000010000, 0b1000010100, 0b1100010000
    - C = A B S
    - S=256/(16*1) = 16
    - lg(16) = 4 : 4 bits are used for the index
    - lg(16) = 4 : 4 bits are used for the byte offset
    - The tag is 48 - (4 + 4) = 40 bits
    - For example: `0b1000 0000 0000 0000 0000 0000 1000 0000`



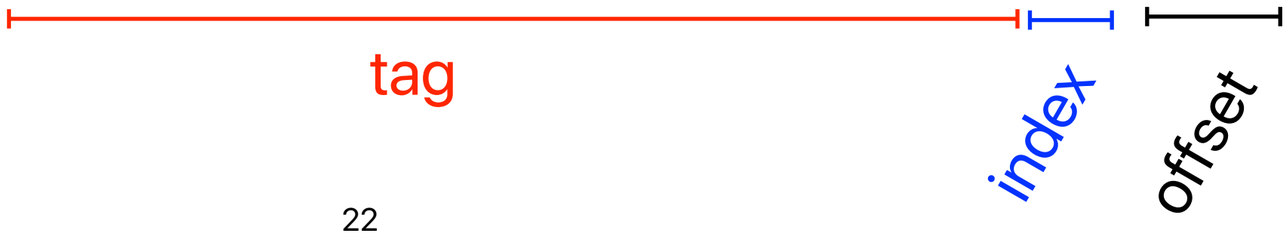
Simulate a direct-mapped cache

	V	D	Tag	Data
0	1	0	0b10	r Architecture!
1	1	0	0b10	This is CS 203:
2	0	0		
3	0	0		
4	0	0		
5	0	0		
6	0	0		
7	0	0		
8	0	0		
9	0	0		
10	0	0		
11	0	0		
12	0	0		
13	0	0		
14	0	0		
15	0	0		

	tag	index		
0b10	0000	0000	compulsory miss
0b10	0000	1000	hit!
0b10	0001	0000	compulsory miss
0b10	0001	0100	hit!
0b11	0001	0000	compulsory miss
0b10	0000	0000	hit!
0b10	0000	1000	hit!
0b10	0001	0000	conflict miss
0b10	0001	0100	hit!

Simulate a 2-way cache

- Consider a 2-way cache with 256 bytes total capacity, a block size of 16 bytes, and the application repeatedly reading the following memory addresses:
  - 0b1000000000, 0b1000001000, 0b1000010000, 0b1000010100, 0b1100010000
    - C = A B S
    - S=256/(16*2) = 8
    - 8 = 2^3 : 3 bits are used for the index
    - 16 = 2^4 : 4 bits are used for the byte offset
    - The tag is 32 - (3 + 4) = 25 bits
    - For example: `0b1000 0000 0000 0000 0000 0000 0001 0000`



Simulate a 2-way cache

	V	D	Tag	Data
0	1	0	0b100	r Architecture!
1	1	0	0b100	This is CS 203:
2	0	0		
3	0	0		
4	0	0		
5	0	0		
6	0	0		
7	0	0		

V	D	Tag	Data
0	0		
1	0	0b110	Advanced Compute
0	0		
0	0		
0	0		
0	0		
0	0		
0	0		

	tag	index		
0b10	0000	0000	compulsory miss
0b10	0000	1000	hit!
0b10	0001	0000	compulsory miss
0b10	0001	0100	hit!
0b11	0001	0000	compulsory miss
0b10	0000	0000	hit!
0b10	0000	1000	hit!
0b10	0001	0000	hit!
0b10	0001	0100	hit!

AMD Phenom II

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

How many of the cache misses are **conflict** misses?

A. 6.25%
B. 66.67%
C. 68.75%
D. 93.75%
E. 100%

AMD Phenom II

100% miss rate!

- Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++)
    c[i] = a[i] + b[i]; /*load a[i], load b[i], store c[i]*/
```

C = ABS
64KB = 2 * 64 * S
S = 512
offset = lg(64) = 6 bits
index = lg(512) = 9 bits
tag = the rest bits

	address in hex	address in binary (tag / index / offset)	tag	index	hit? miss?
load a[0]	0x20000	0b10 0000 0000 0000 0000	0x4	0	compulsory miss
load b[0]	0x30000	0b11 0000 0000 0000 0000	0x6	0	compulsory miss
store c[0]	0x10000	0b01 0000 0000 0000 0000	0x2	0	compulsory miss, evict
load a[1]	0x20004	0b10 0000 0000 0000 0100	0x4	0	conflict miss, evict 0x6
load b[1]	0x30004	0b11 0000 0000 0000 0100	0x6	0	conflict miss, evict 0x2
store c[1]	0x10004	0b01 0000 0000 0000 0100	0x2	0	conflict miss, evict 0x4
⋮	⋮	⋮	⋮	⋮	⋮
load a[15]	0x2003C	0b10 0000 0000 0011 1100	0x4	0	miss, evict 0x6
load b[15]	0x3003C	0b11 0000 0000 0011 1100	0x6	0	miss, evict 0x2
store c[15]	0x1003C	0b01 0000 0000 0011 1100	0x2	0	miss, evict 0x4
load a[16]	0x20040	0b10 0000 0000 0100 0000	0x4	1	compulsory miss
load b[16]	0x30040	0b11 0000 0000 0100 0000	0x6	1	compulsory miss
store c[16]	0x10040	0b01 0000 0000 0100 0000	0x2	1	compulsory miss, evict

AMD Phenom II

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 32-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

How many of the cache misses are **conflict** misses?

A. 6.25%
B. 66.67%
C. 68.75%
D. 93.75%
E. 100%

C = ABS

64KB = 2 * 64 * S

S = 512

offset = lg(64) = 6 bits

index = lg(512) = 9 bits

tag = 64 - lg(512) - lg(64) = 49 bits

Basic Hardware Optimization in Improving 3Cs

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

AMD Phenom II

- D-L1 Cache configuration of AMD Phenom II
  - Size 64KB, 2-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
int a[16384], b[16384], c[16384];
/* c = 0x10000, a = 0x20000, b = 0x30000 */
for(i = 0; i < 512; i++) {
    c[i] = a[i] + b[i];
    //load a, b, and then store to c
}
```

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

A. 6.25%
B. 56.25%
C. 66.67%
D. 68.75%
E. 100%

C = ABS

64KB = 2 * 64 * S

S = 512

offset = lg(64) = 6 bits

index = lg(512) = 9 bits

tag = 64 - lg(512) - lg(64) = 49 bits

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

Norman P. Jouppi

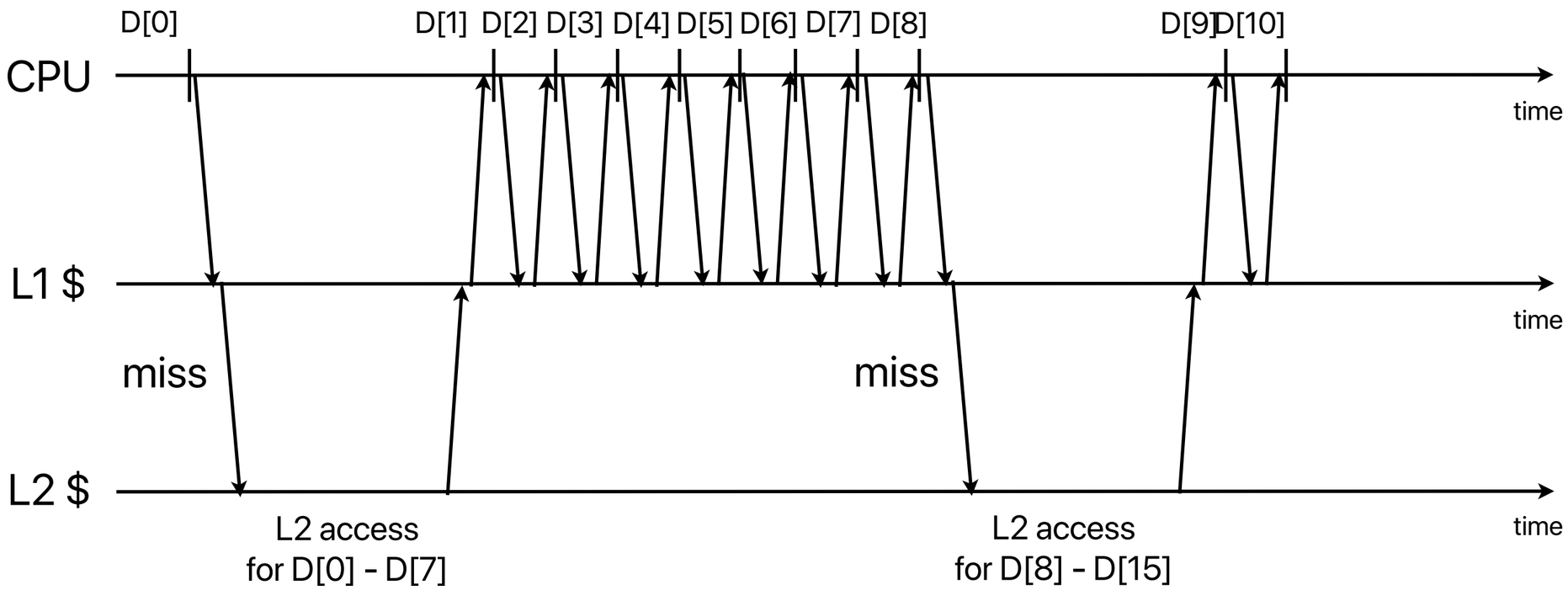
Which of the following schemes can help AMD Phenom II?

- 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 AMD Phenom II 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

Prefetching

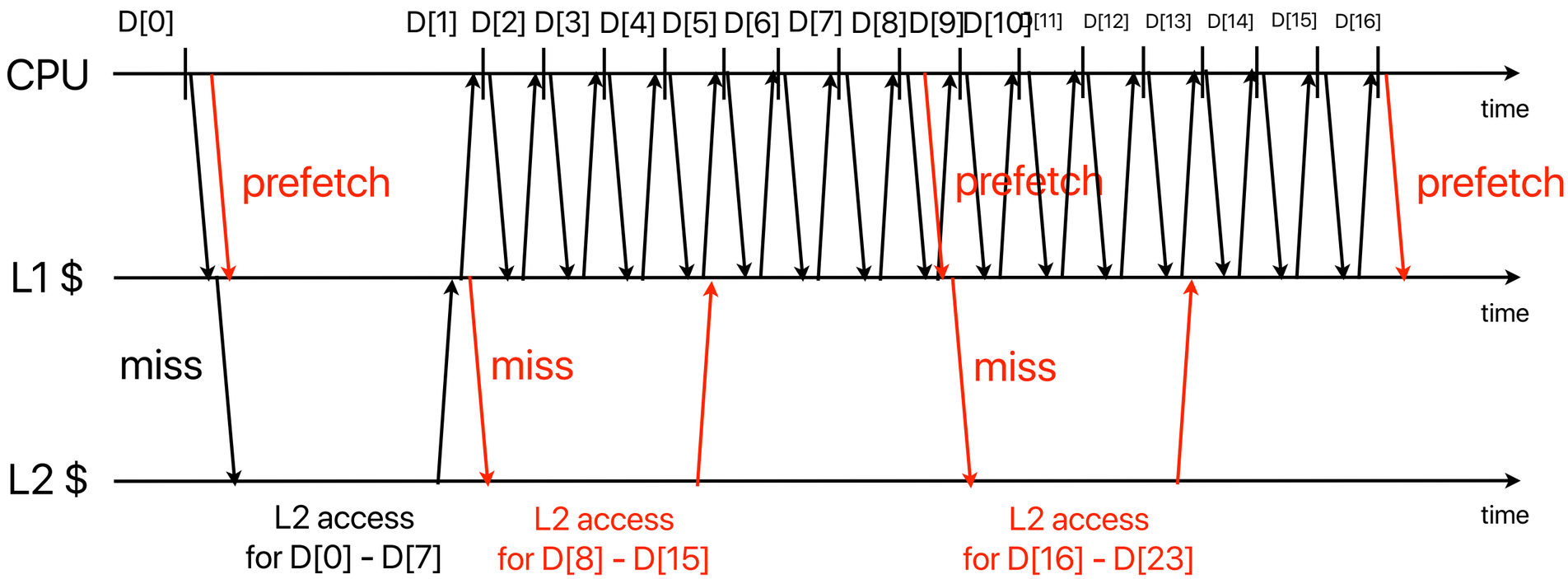
Characteristic of memory accesses

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



Prefetching

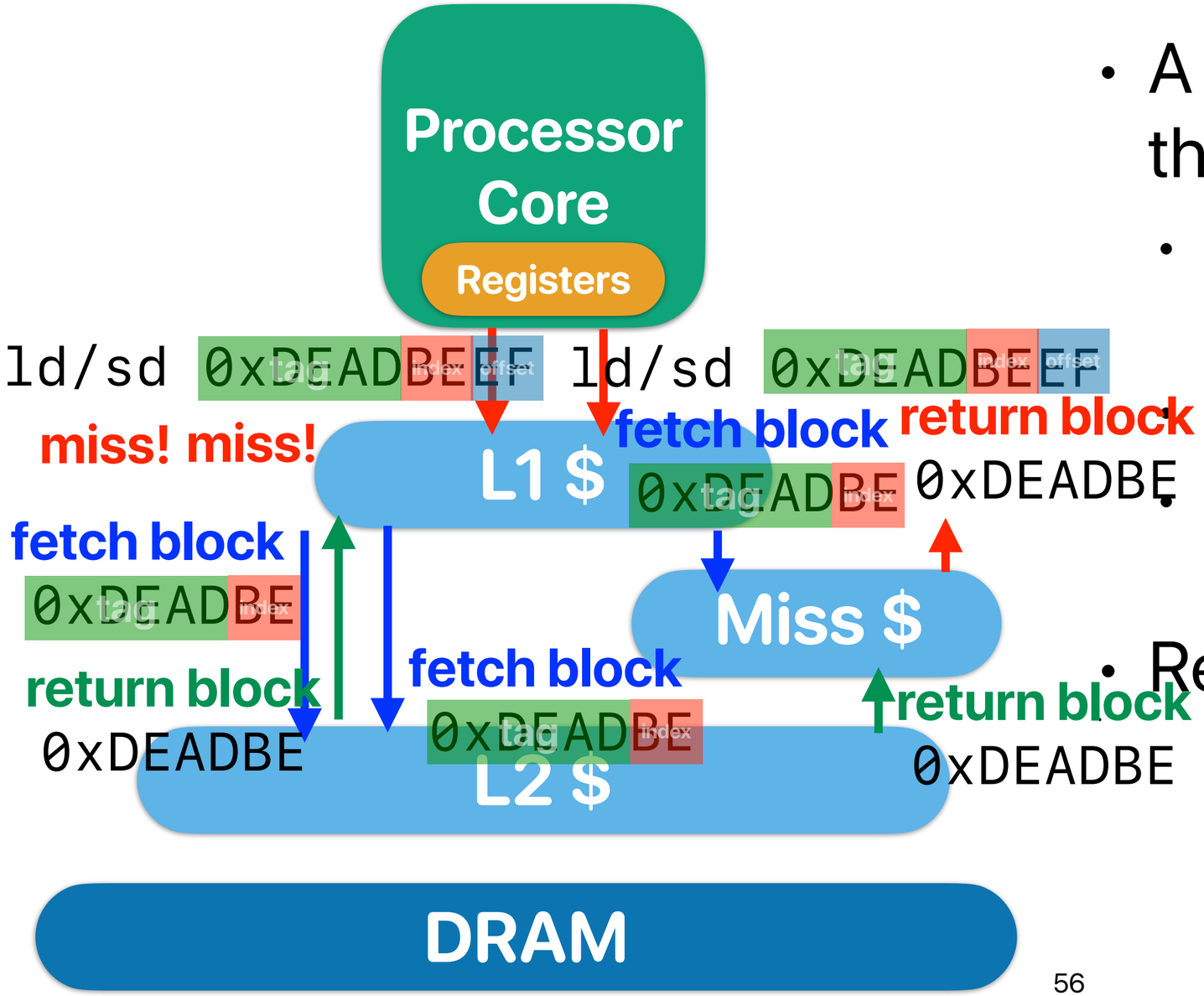
```
for(i = 0;i < 1000000; i++) {
      D[i] = rand();
      // prefetch D[i+8] if i % 8 == 0
}
```



Prefetching

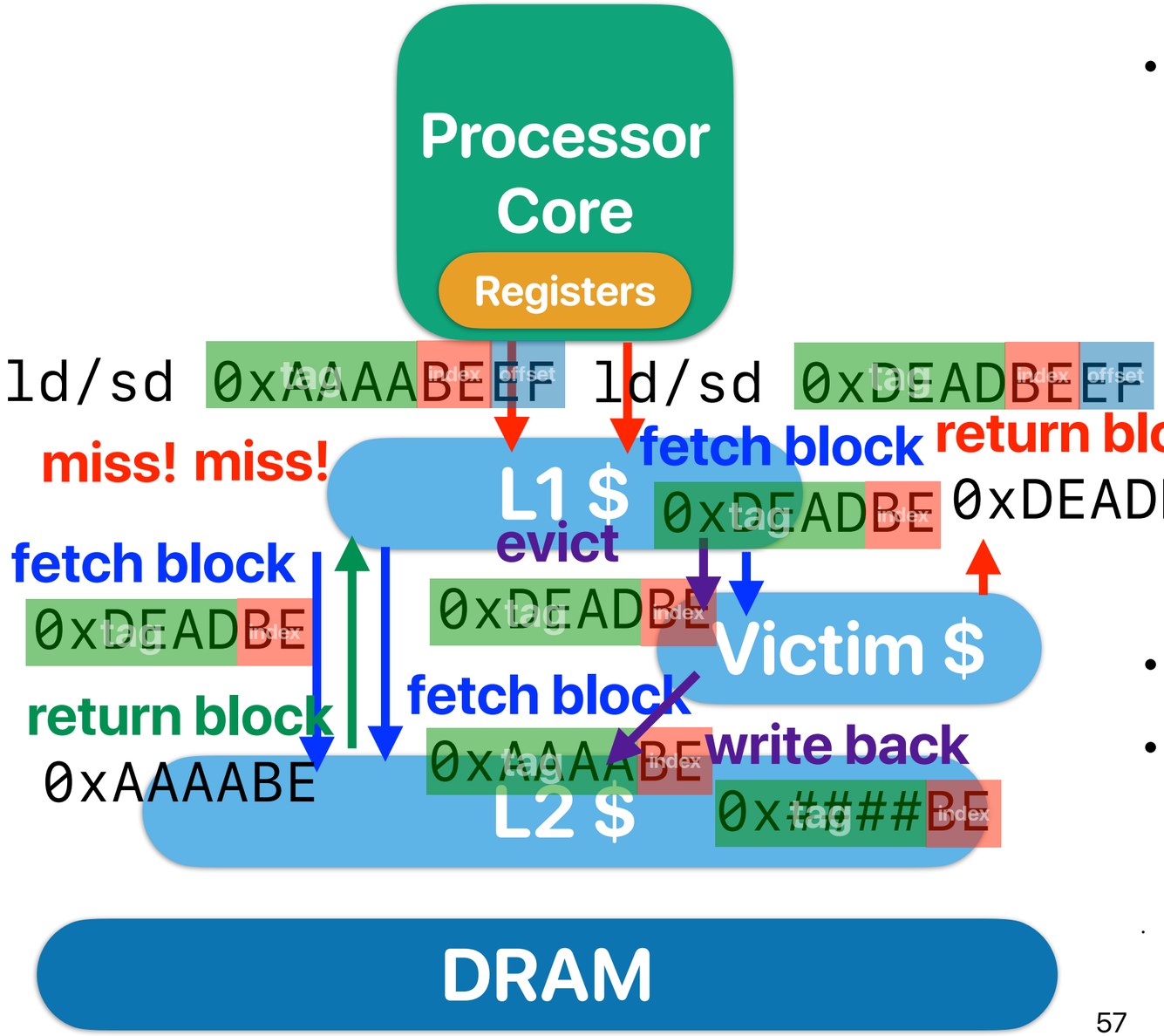
- Identify the access pattern and proactively fetch data/instruction before the application asks for the data/instruction
  - Trigger the cache miss earlier to eliminate the miss when the application needs the data/instruction
- Hardware prefetch
  - The processor can keep track the distance between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

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

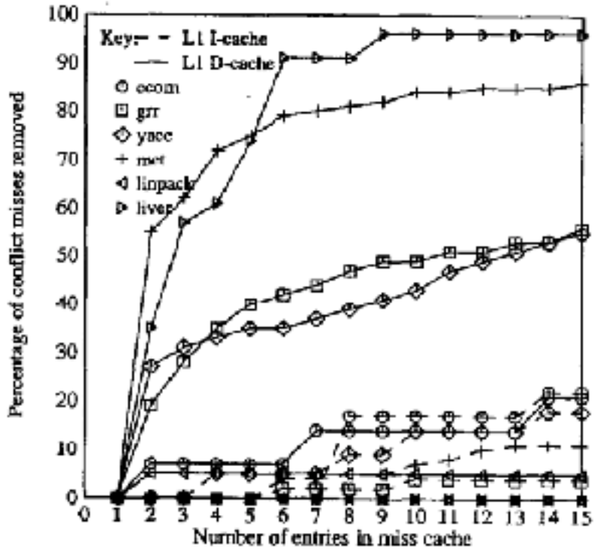


Figure 3-3: Conflict misses removed by miss caching

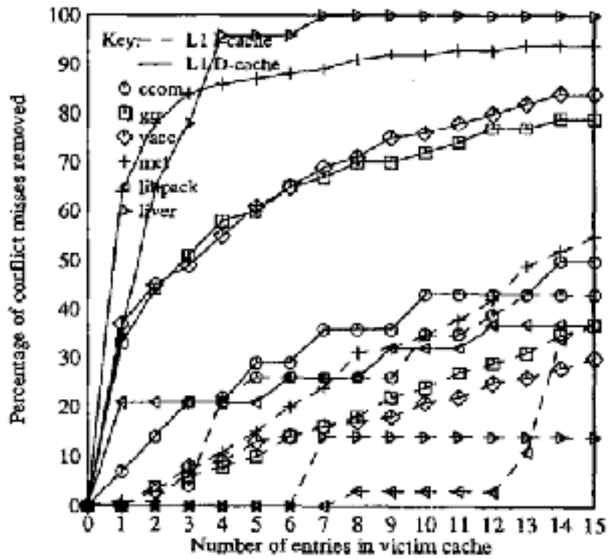


Figure 3-5: Conflict misses removed by victim caching

Which of the following schemes can help Athlon 64?

- 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 AMD Phenom II 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

Announcement

- Reading Quiz #5 next Monday before the lecture
- Office Hours
  - Walk-in, no appointment is necessary
  - Hung-Wei/Prof. Usagi: MTu 2p-3p (WCH 406 or on Zoom)
  - Abenezer Wudenhe: WTh 3p-4p (Zoom only)

Computer
Science &
Engineering

203

つづく