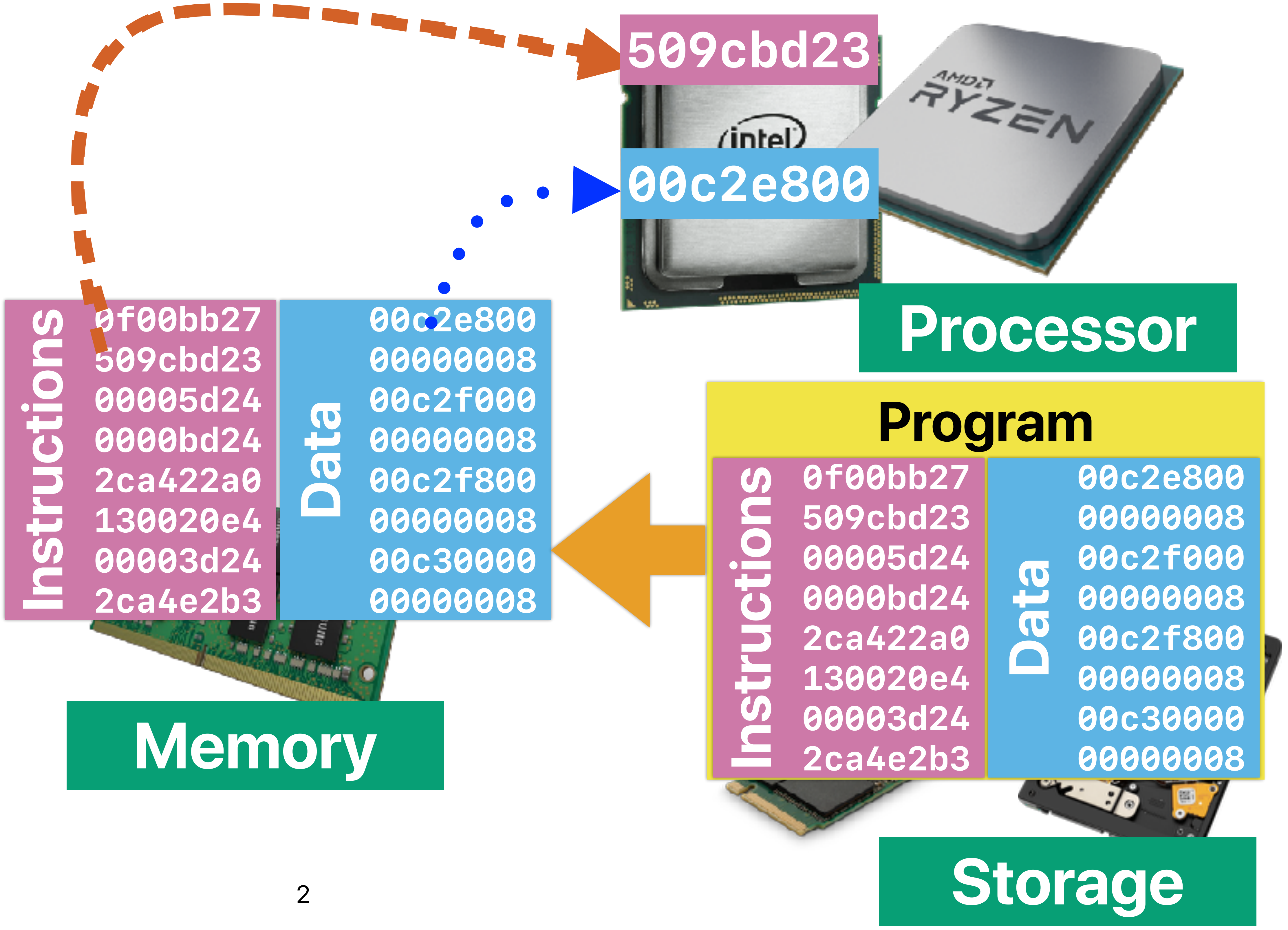
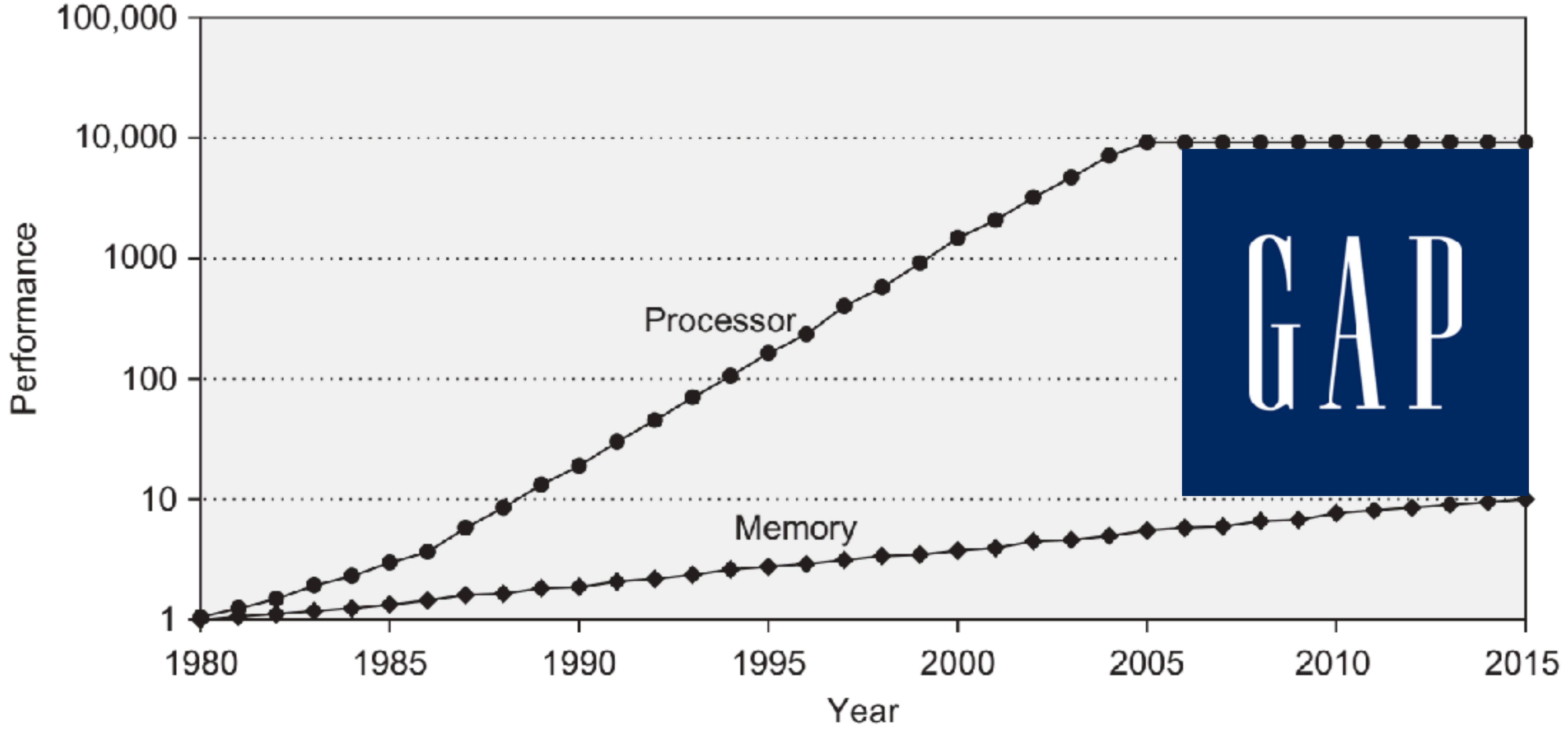
Memory Hierarchy Inside Out: (4) Cache misses and their remedies (cont.)

Hung-Wei Tseng

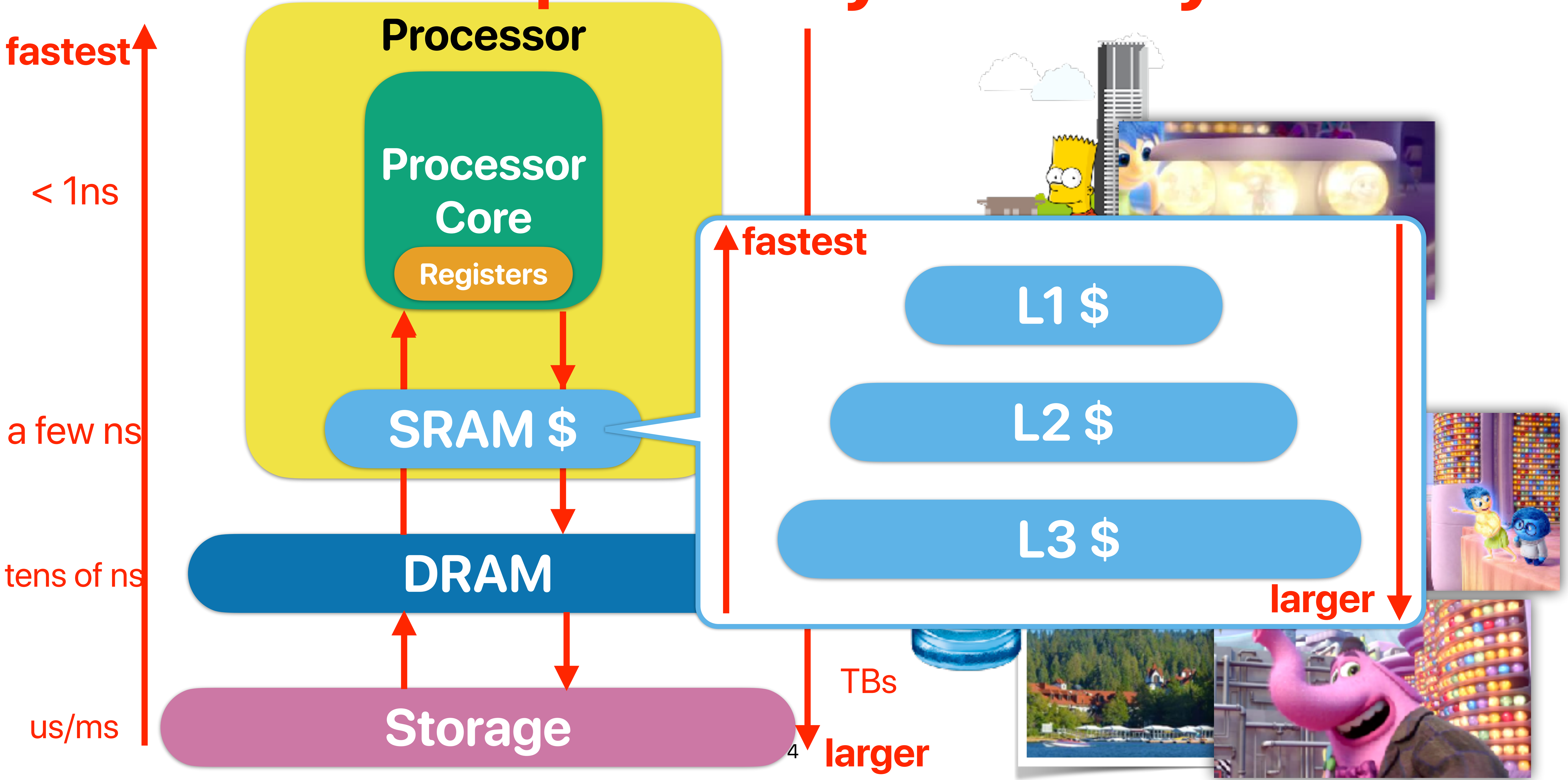
von Neumman Architecture



Recap: Performance gap between Processor/Memory



Recap: Memory Hierarchy



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: 0b000010000 010 0100

tag: 000010000 | set index: 010 | block offset: 0100

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

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

Recap: NVIDIA Tegra X1

100% miss rate!

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

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

```
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]
```

tag index offset

	Address (Hex)	Address in binary	Tag	Index	Hit? Miss?	Replace?
a[0]	0x10000	0b0001000000000000000000	0x8	0x0	Compulsory Miss	
b[0]	0x20000	0b0010000000000000000000	0x10	0x0	Compulsory Miss	
c[0]	0x30000	0b0011000000000000000000	0x18	0x0	Compulsory Miss	
d[0]	0x40000	0b0100000000000000000000	0x20	0x0	Compulsory Miss	
e[0]	0x50000	0b0101000000000000000000	0x28	0x0	Compulsory Miss	a[0-7]
a[1]	0x10008	0b0001000000000000001000	0x8	0x0	Conflict Miss	b[0-7]
b[1]	0x20008	0b00100000000000001000	0x10	0x0	Conflict Miss	c[0-7]
c[1]	0x30008	0b00110000000000001000	0x18	0x0	Conflict Miss	d[0-7]
d[1]	0x40008	0b01000000000000001000	0x20	0x0	Conflict Miss	e[0-7]
e[1]	0x50008	0b01010000000000001000	0x28	0x0	Conflict Miss	a[0-7]
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Recap: intel Core i7 (cont.)

- Size 32KB, 8-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 = 8 * 64 * S

S = 64

offset = lg(64) = 6 bits

index = lg(64) = 6 bits

tag = the rest bits

tag index offset

	Address (Hex)	Address in binary	Tag	Index	Hit? Miss?	Replace?
a[7]	0x10038	0b00010000000000111000	0x10	0x0	Hit	
b[7]	0x20038	0b00100000000000111000	0x20	0x0	Hit	
c[7]	0x30038	0b00110000000000111000	0x30	0x0	Hit	
d[7]	0x40038	0b01000000000000111000	0x40	0x0	Hit	
e[7]	0x50038	0b01010000000000111000	0x50	0x0	Hit	
a[8]	0x10040	0b00010000000001000000	0x10	0x1	Compulsory Miss	
b[8]	0x20040	0b00100000000001000000	0x20	0x1	Compulsory Miss	
c[8]	0x30040	0b00110000000001000000	0x30	0x1	Compulsory Miss	
d[8]	0x40040	0b01000000000001000000	0x40	0x1	Compulsory Miss	
e[8]	0x50040	0b01010000000001000000	0x50	0x1	Compulsory Miss	
a[9]	0x10048	0b00010000000001001000	0x10	0x1	Hit	
b[9]	0x20048	0b00100000000001001000	0x20	0x1	Hit	
c[9]	0x30048	0b00110000000001001000	0x30	0x1	Hit	
d[9]	0x40048	0b01000000000001001000	0x40	0x1	Hit	

12.5% miss rate! Conflict misses are gone!

Outline

- The remedies of cache misses — the hardware version
- The remedies of cache misses — the software version

Basic Hardware Optimization in Improving 3Cs

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?

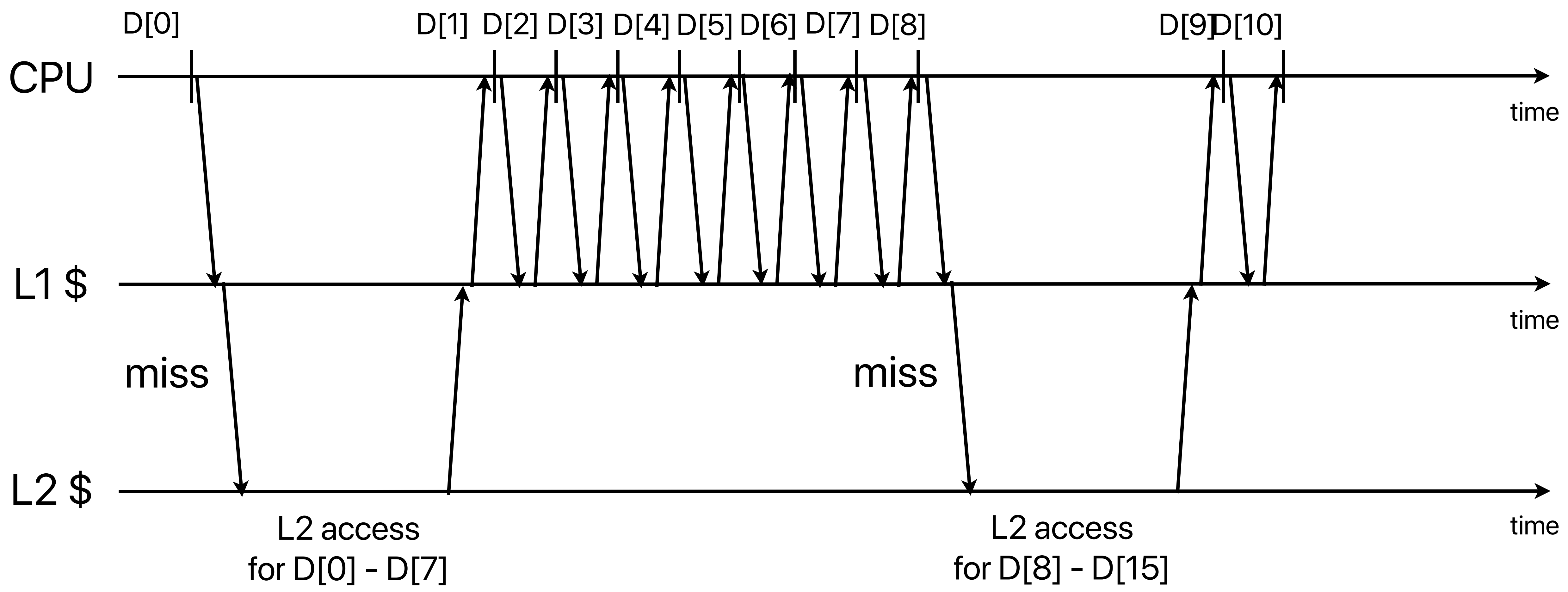
- How many of the following schemes mentioned in “improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers” would help NVIDIA’s Tegra for the code in the previous slide?
  - ① Missing cache
  - ② Victim cache
  - ③ Prefetch
  - ④ Stream buffer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

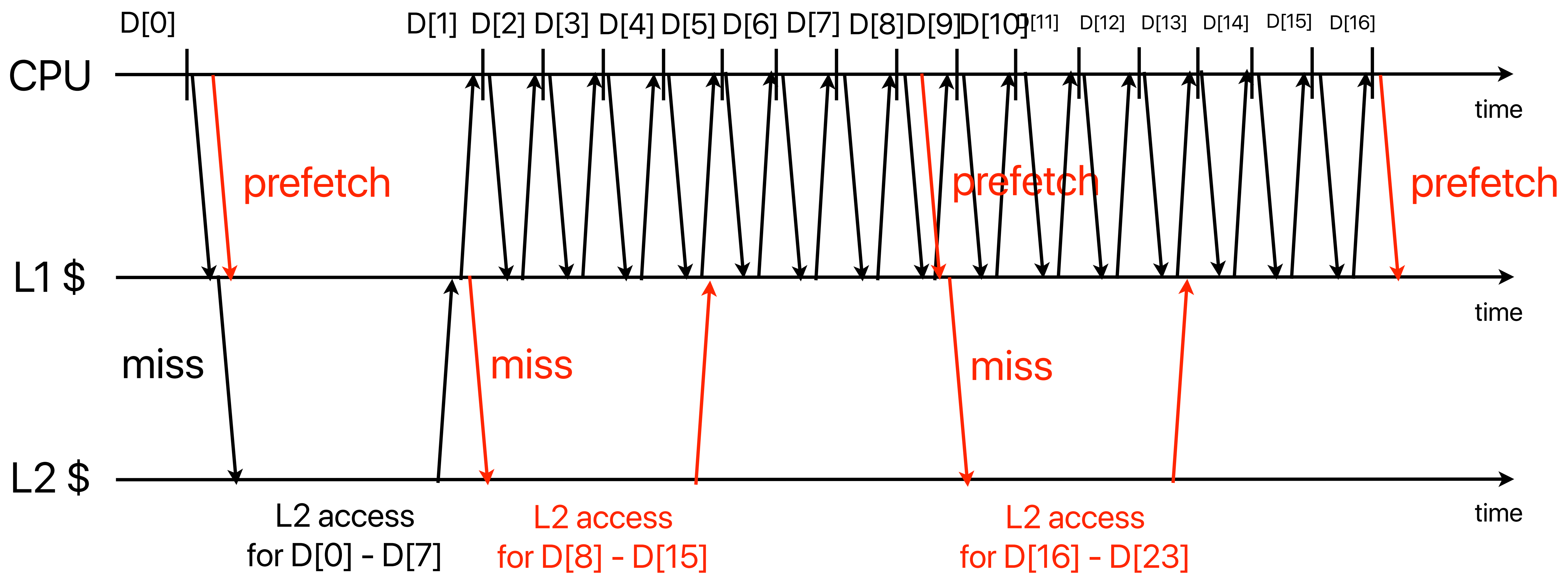
Characteristic of memory accesses

```
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
}
```



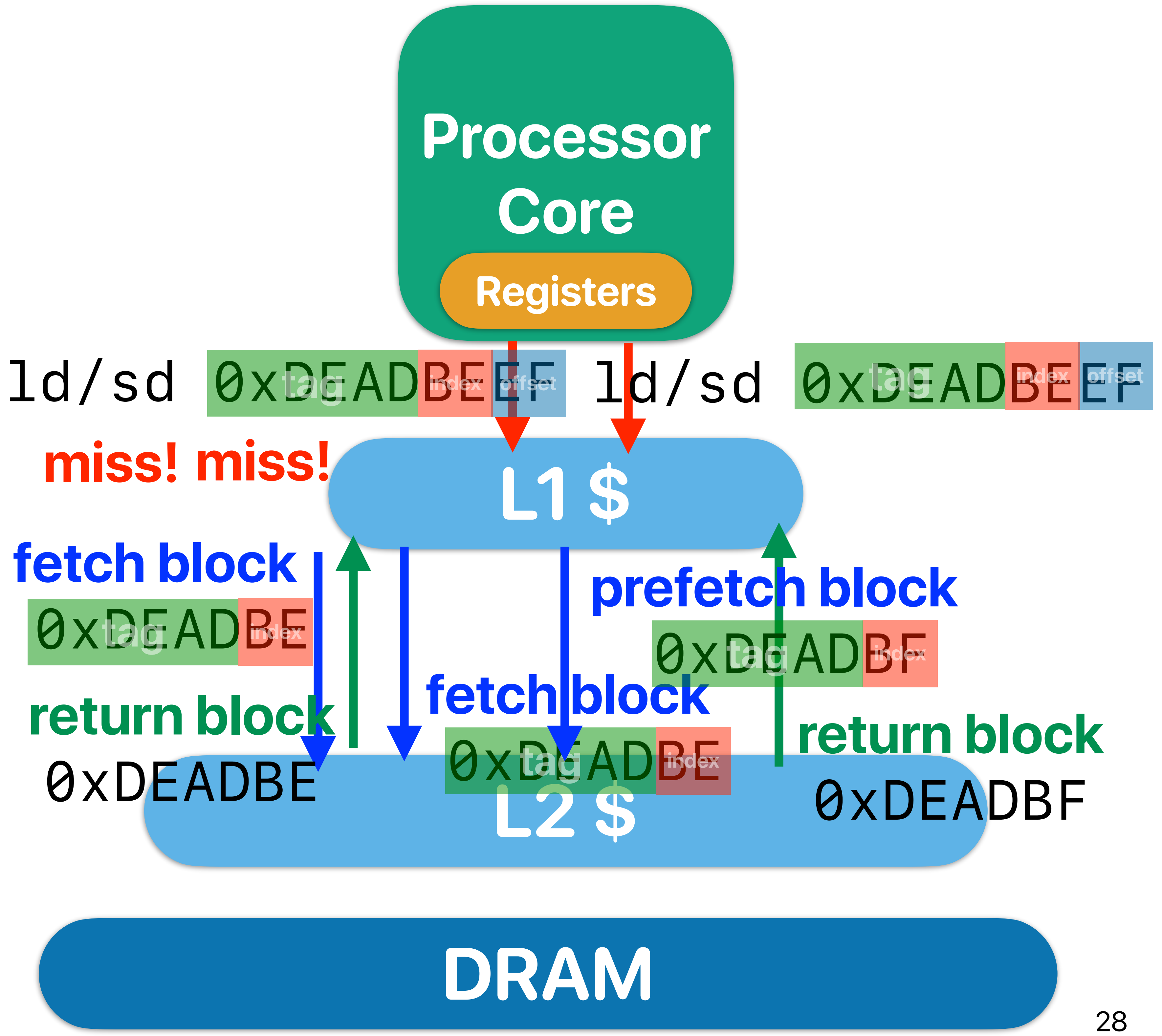
Prefetching

- 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

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

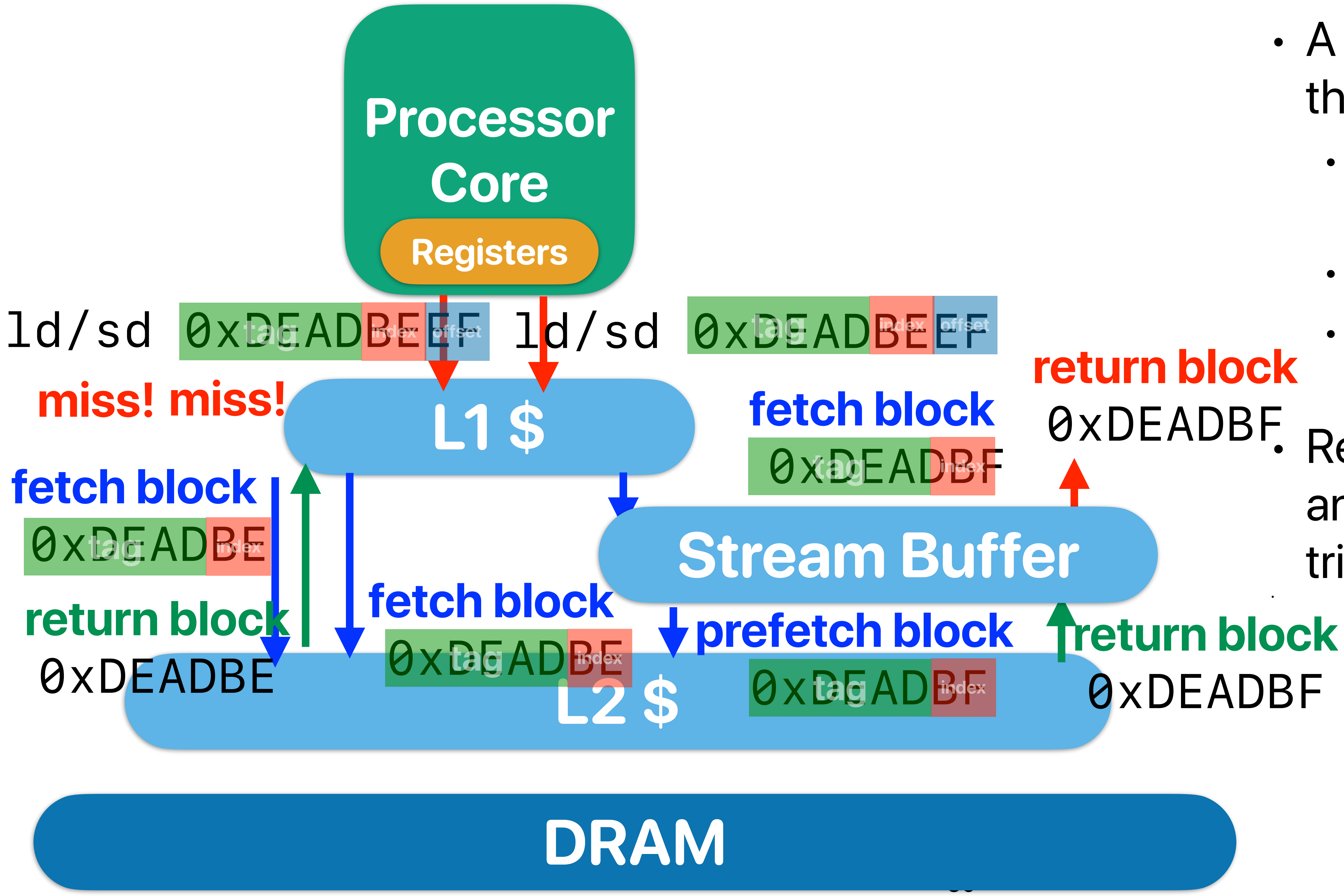
tag **index** **offset**

	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]

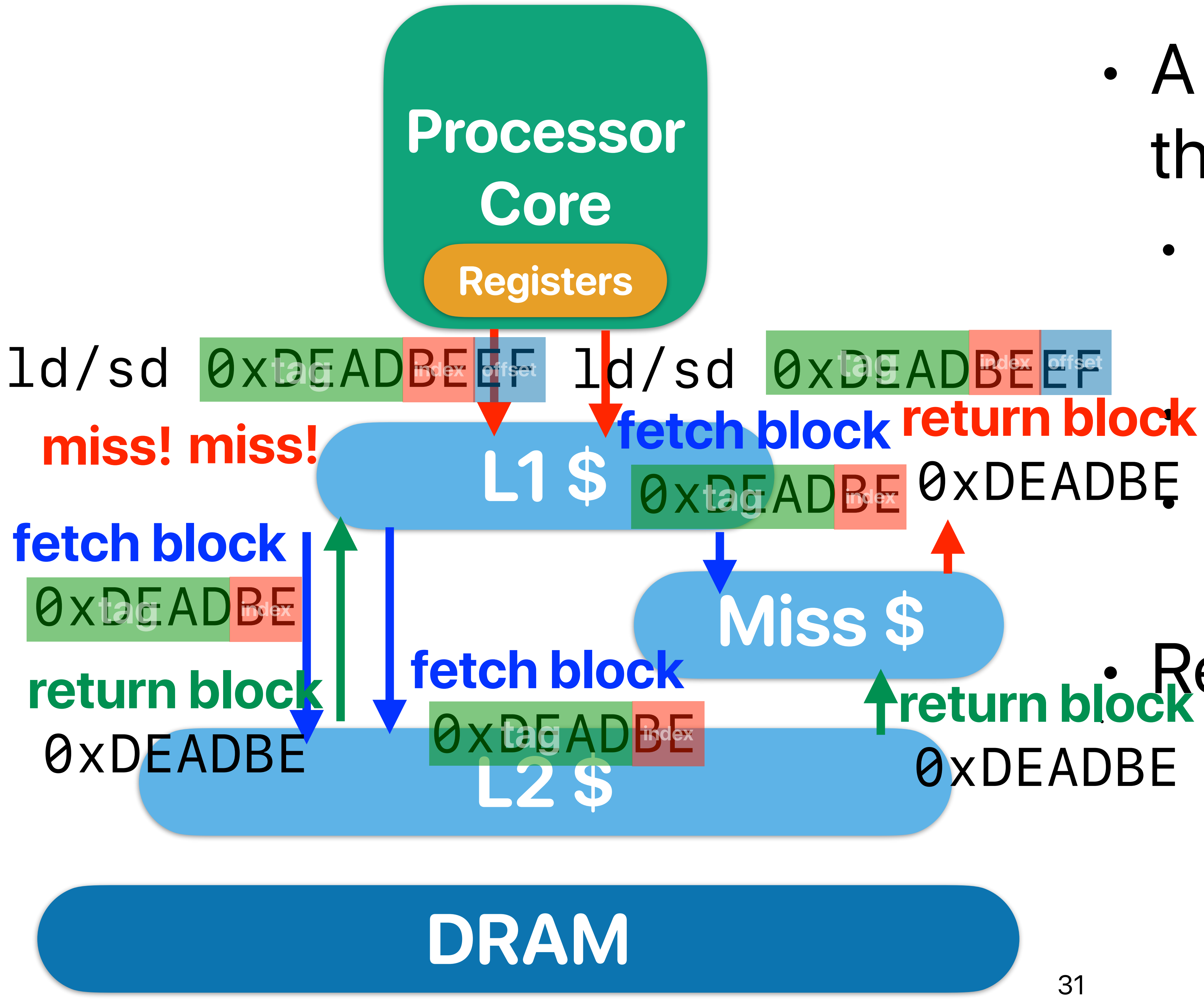
100% miss rate!

Stream buffer



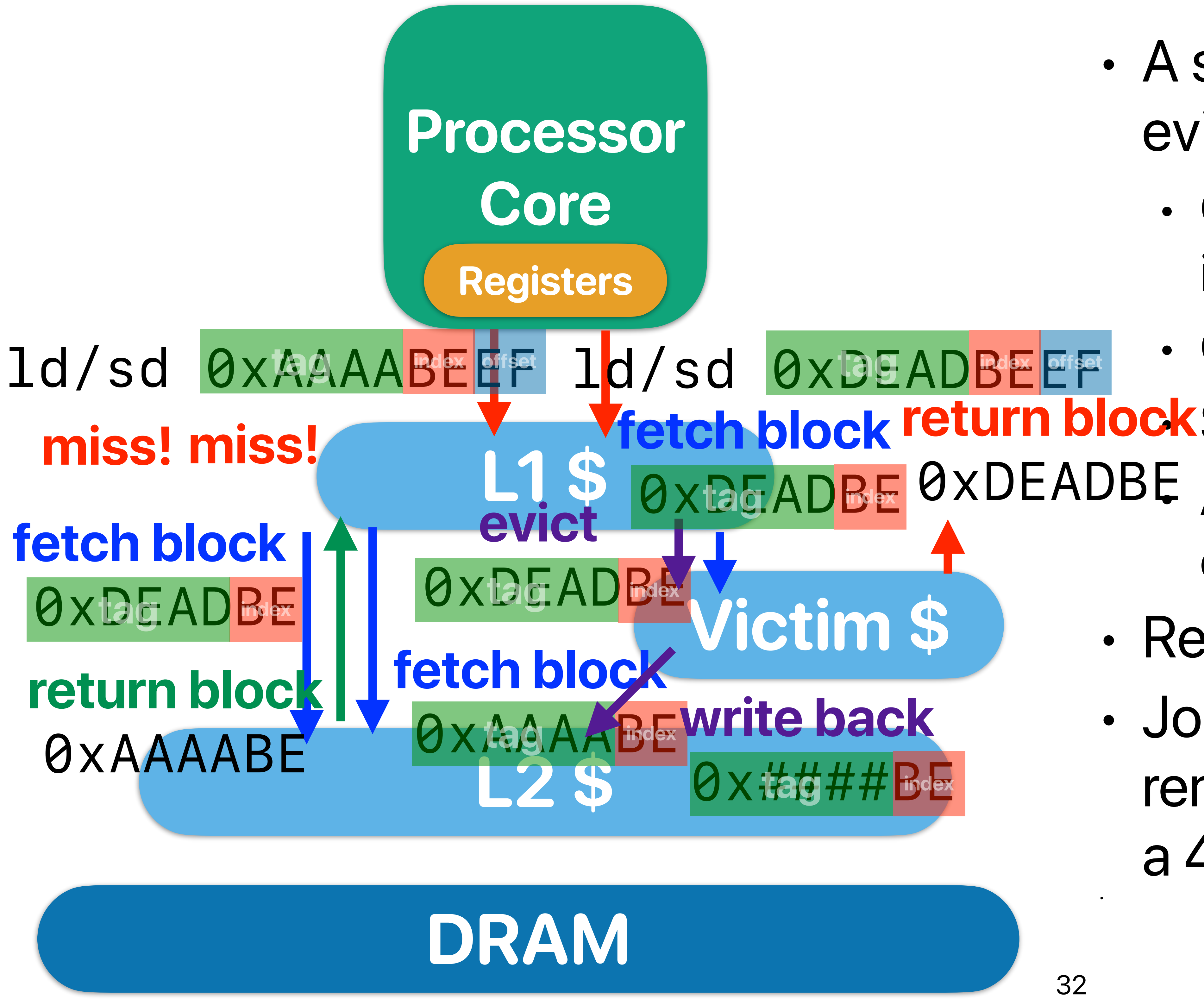
- A small cache that captures the prefetched blocks
  - Can be built as fully associative since it's small
  - Consult when there is a miss
  - Retrieve the block if found in the stream buffer
- Reduce compulsory misses and avoid conflict misses triggered by prefetching

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

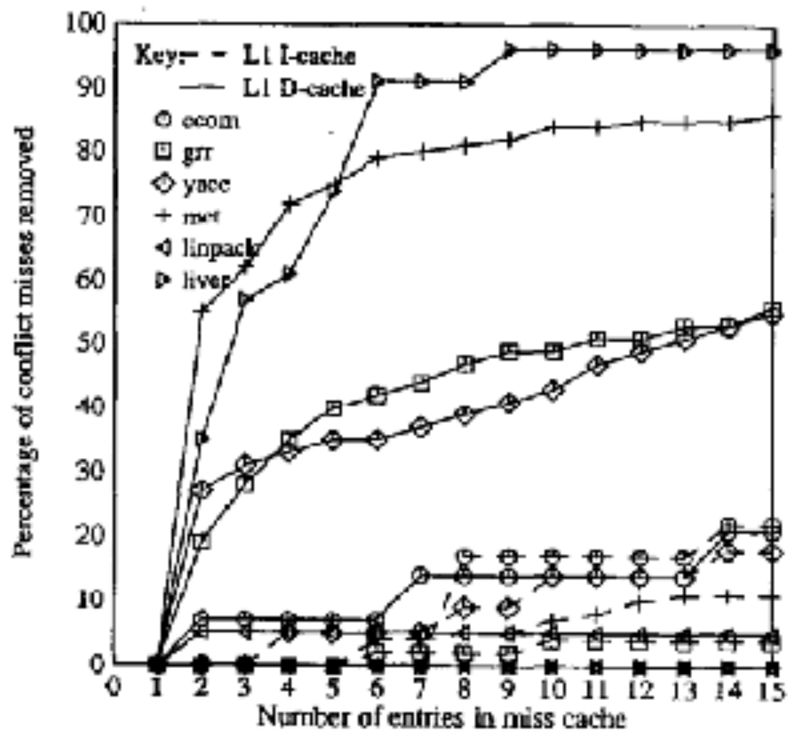


Figure 3-3: Conflict misses removed by miss caching

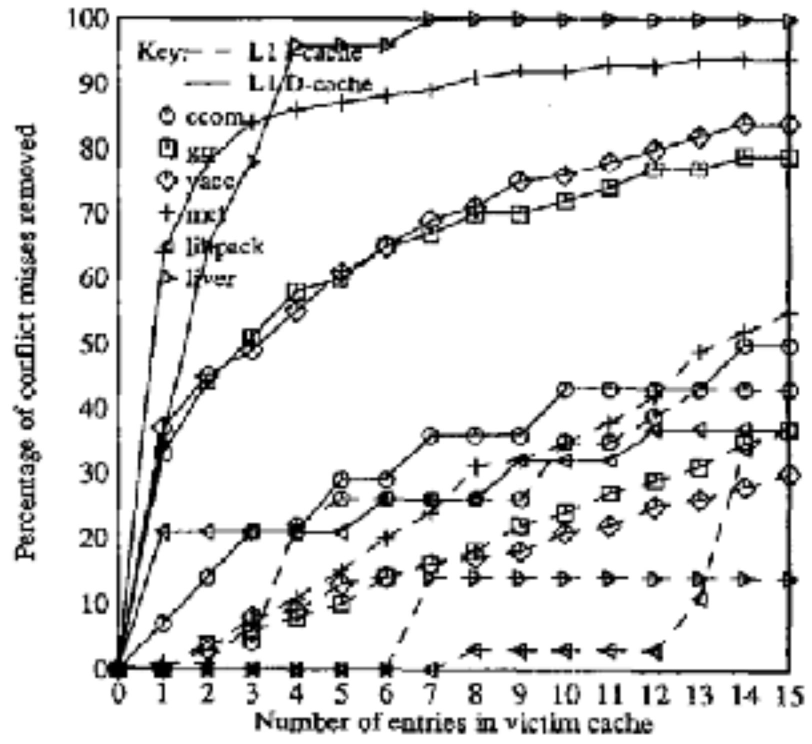


Figure 3-5: Conflict misses removed by victim caching

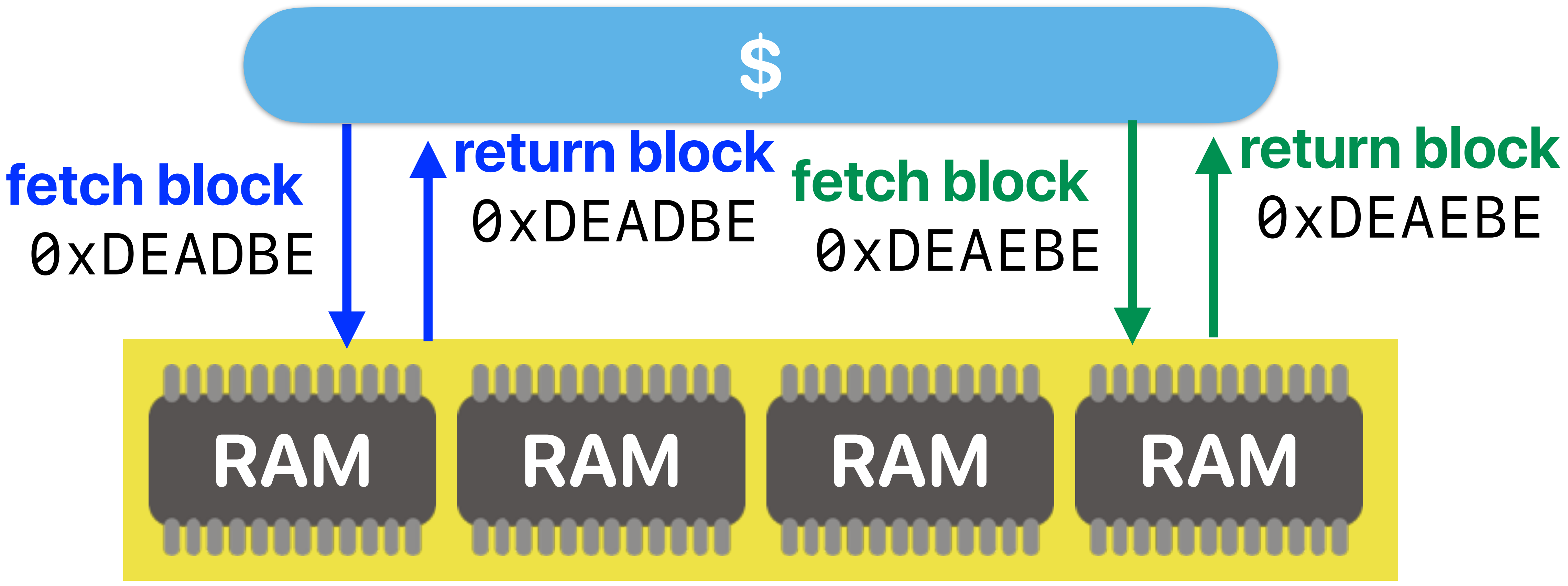
Which of the following schemes can help Tegra?

- 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 cache 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**

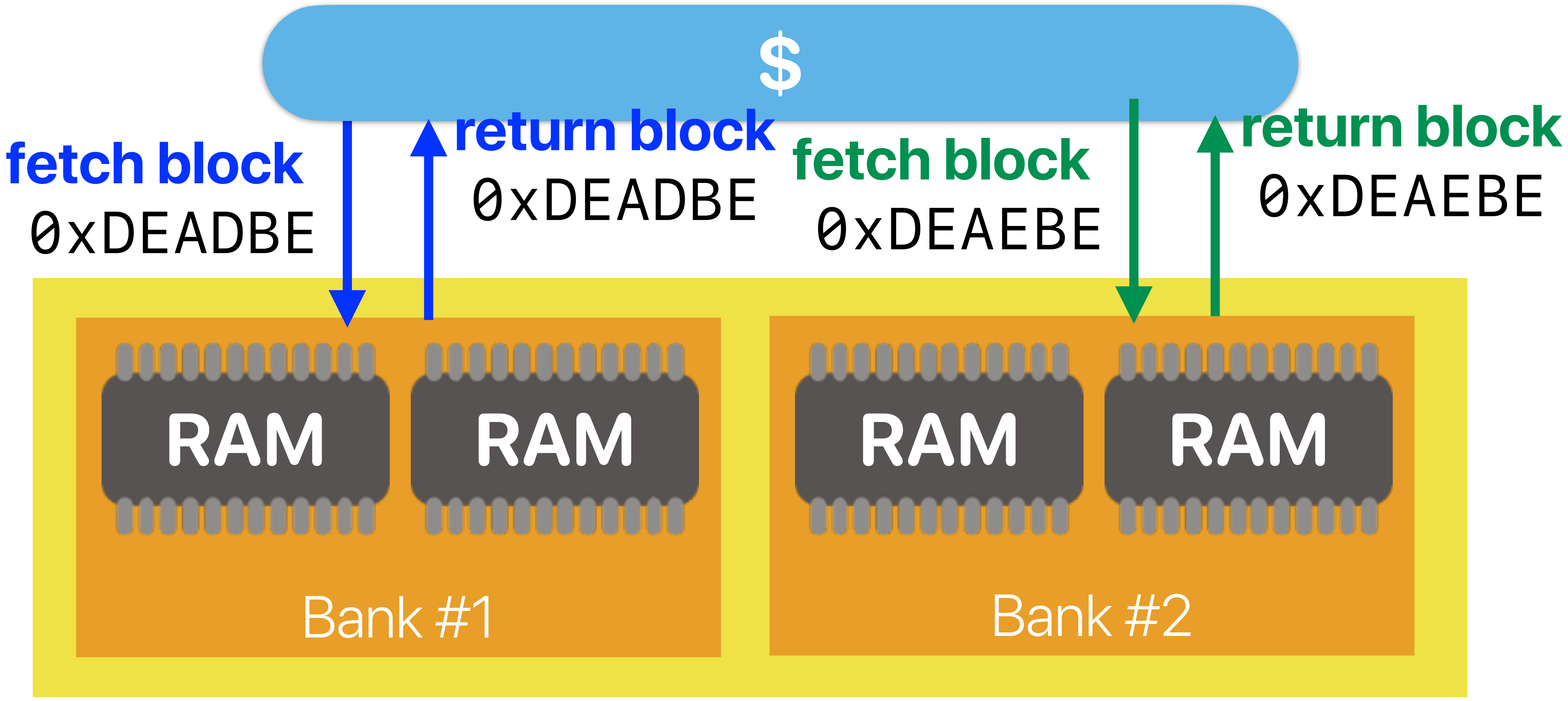
  A. 0
  B. 1
  C. 2
  [✓ D. 3]
  E. 4

Advanced Hardware Techniques in Improving Memory Performance

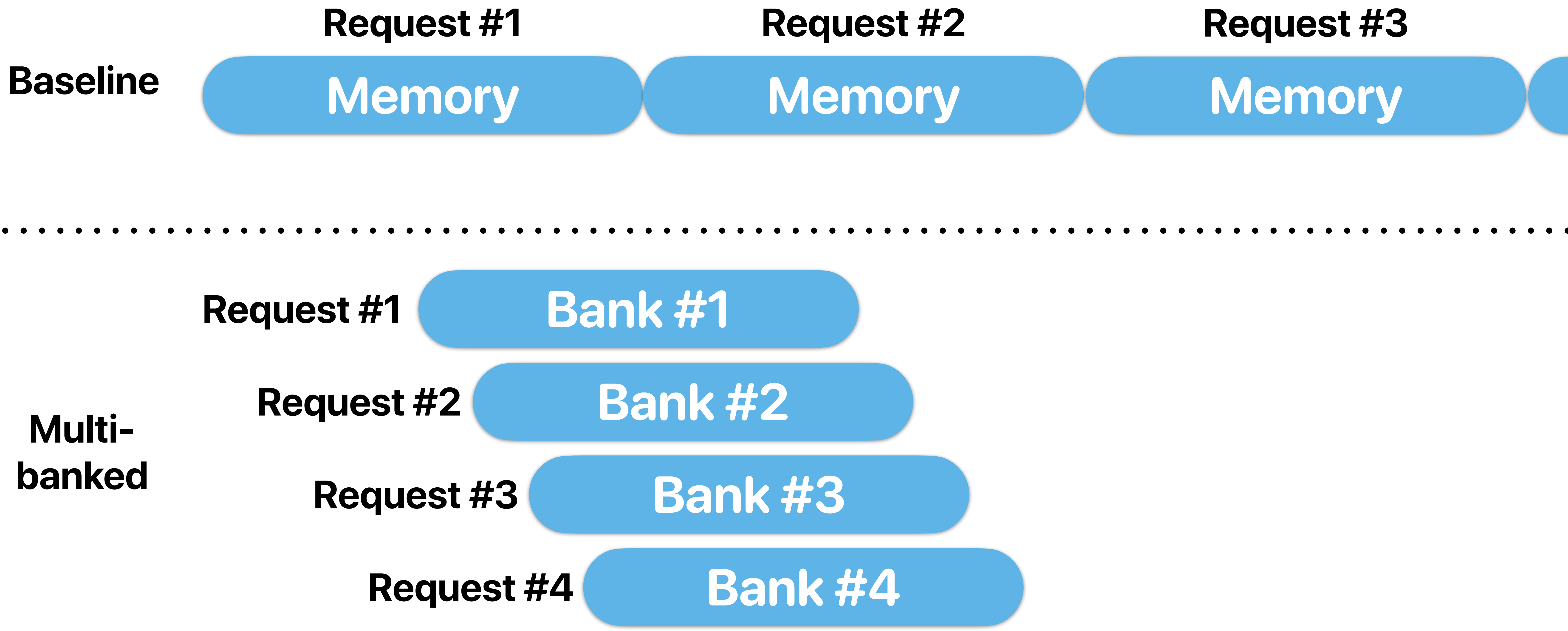
Blocking cache



Multibanks & non-blocking caches



Pipelined access and multi-banked caches



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

Pipelined access and multi-banked caches

- Assume each bank in the $ takes 10 ns to serve a request, and the $ can take the next request 1 ns after assigning a request to a bank — if we have 4 banks and we want to serve 4 requests, what's the speedup over non-banked, non-pipelined $? — pick the closest one
  A. 1x — no speedup
  B. 2x
  C. 3x
  D. 4x
  E. 5x

Pipelined access and multi-banked caches

- Assume each bank in the $ takes 10 ns to serve a request, and the $ can take the next request 1 ns after assigning a request to a bank — if we have 4 banks and we want to serve 4 requests, what's the speedup over non-banked, non-pipelined $? — pick the closest one
  - A. 1x — no speedup
  - B. 2x
  - [✓ C. 3x]
  - D. 4x
  - E. 5x

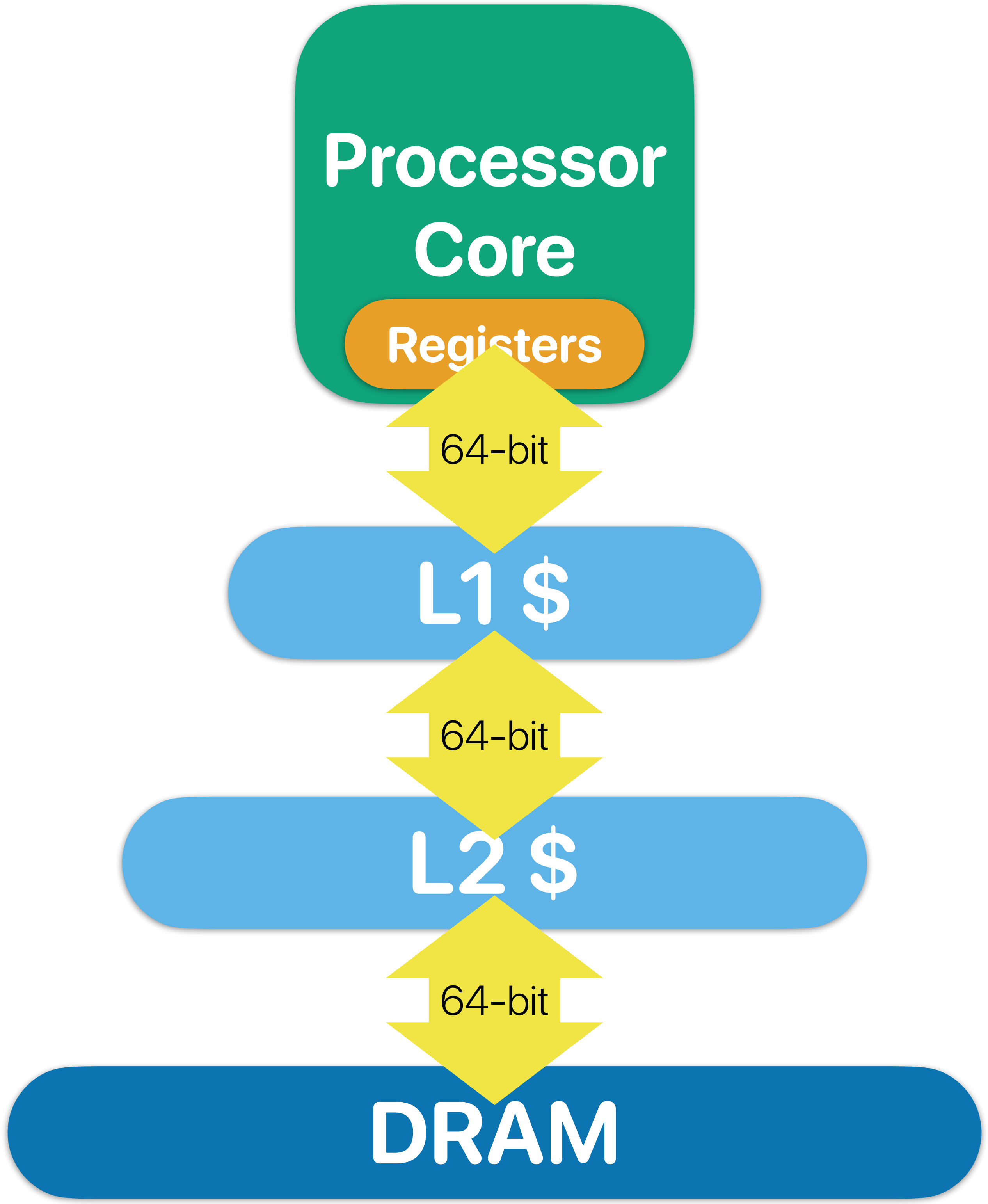
$$ET_{baseline} = 4 \times 10\ ns = 40\ ns$$

$$ET_{banked} = 10\ ns + 3 \times 1\ ns = 13\ ns$$

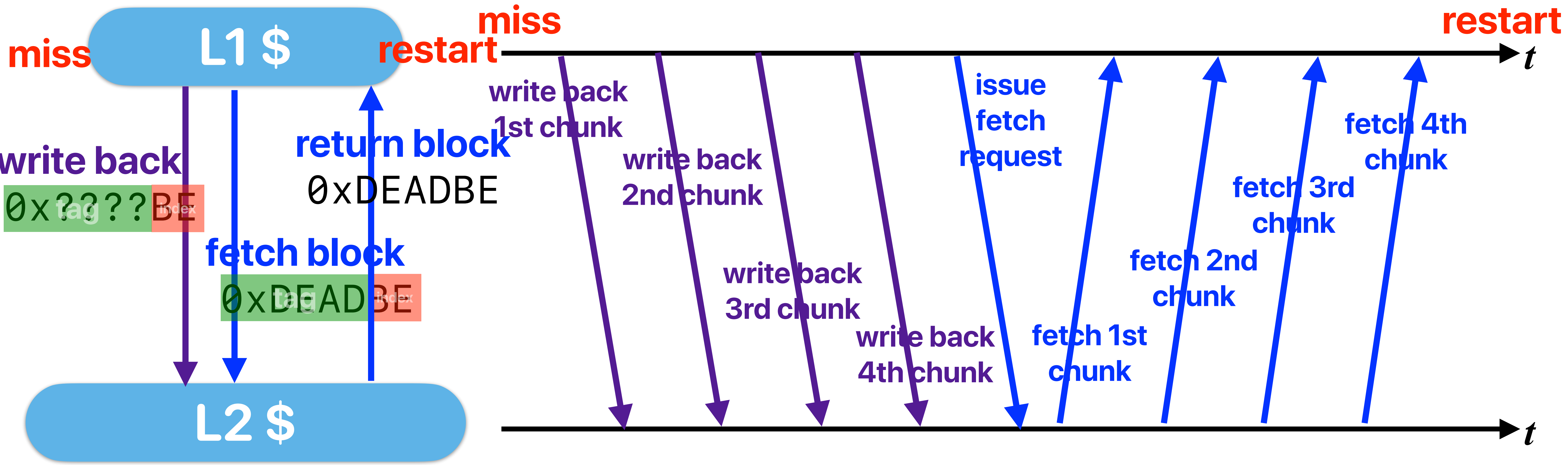
$$Speedup = \frac{Execution\ Time_{baseline}}{Execution\ Time_{banked}}$$

$$= \frac{40}{13} = 3.08 \times$$

The bandwidth between units is limited

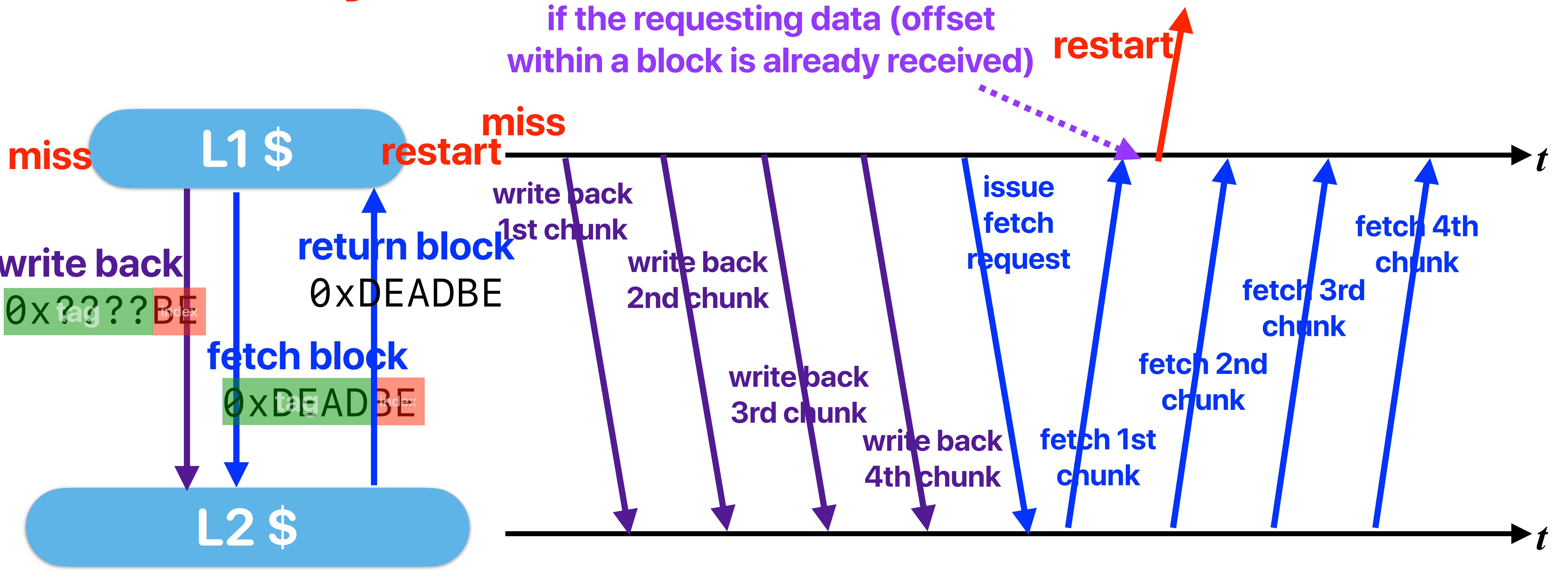


When we handle a miss



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

Early Restart and Critical Word First

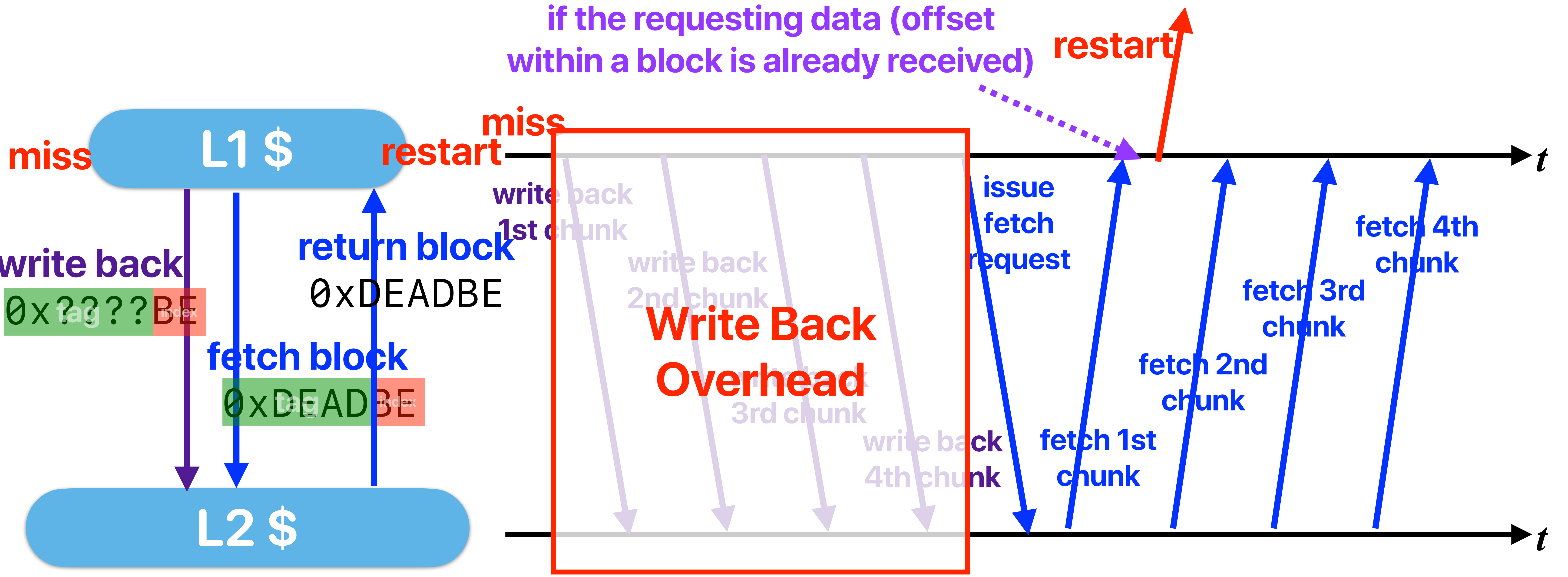


assume the bus between L1/L2 only allows a quarter of the cache block go through it

Early Restart and Critical Word First

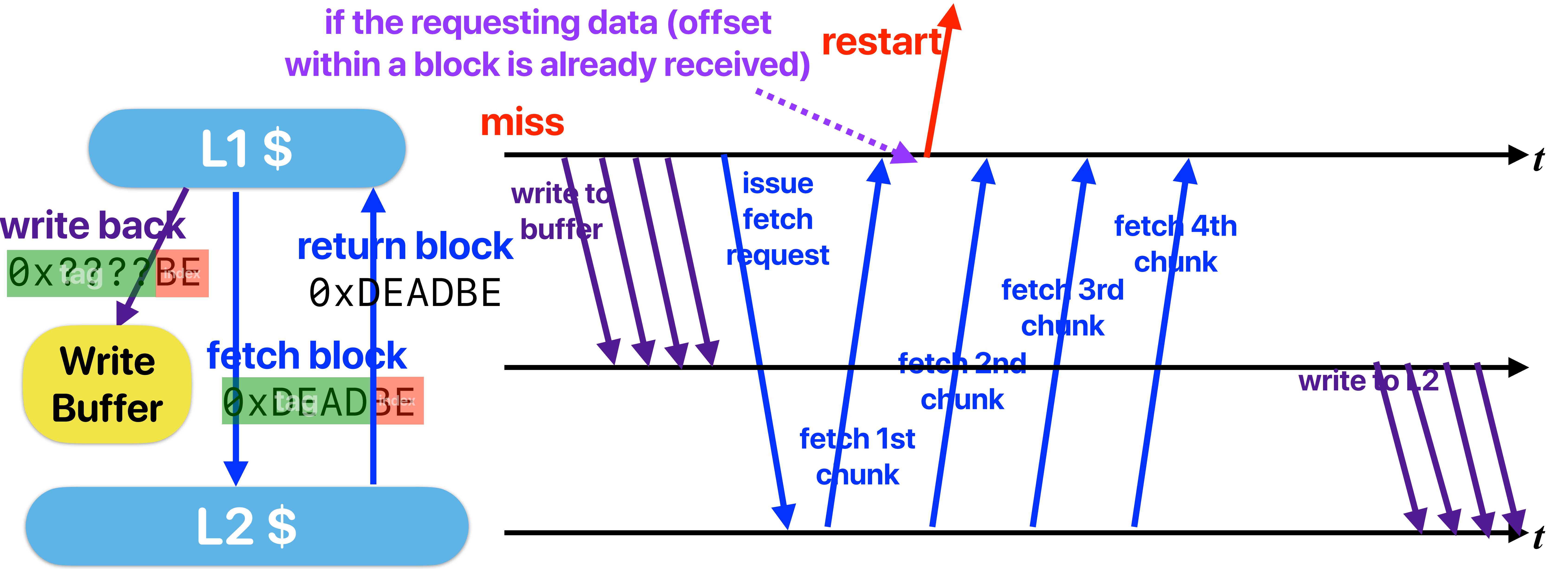
- Don’t wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let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

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

Summary of Optimizations

- Regarding the following cache optimizations, how many of them would help improve miss rate?
  ① Non-blocking/pipelined/multibanked cache
  ② Critical word first and early restart
  ③ Prefetching
  ④ Write buffer
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

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

Summary of Optimizations

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

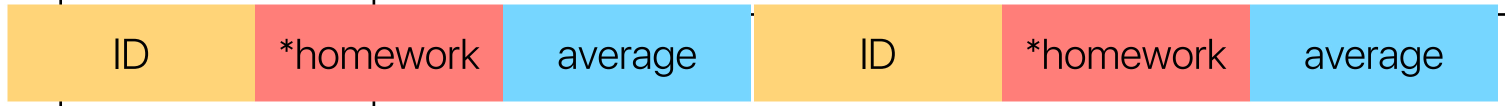
How can programmer improve memory performance?

Data layout

Array of structures or structure of arrays

Array of objects

```c
struct grades
{
  int id;
  double *homework;
  double average;
};
```

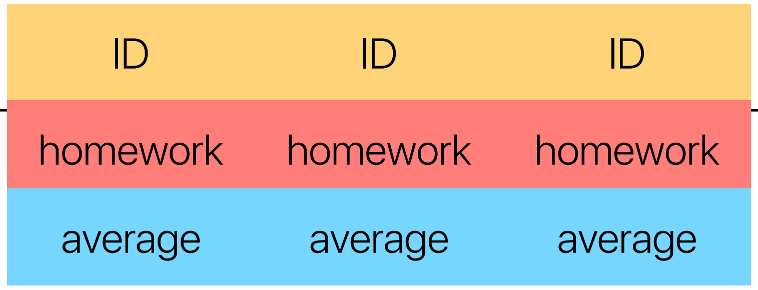


average of each homework

```c
for(i=0;i<homework_items; i++)
{
gradesheet[total_number_students].homework[i] = 0.0;
   for(j=0;j<total_number_students;j++)
gradesheet[total_number_students].homework[i]
+=gradesheet[j].homework[i];
   gradesheet[total_number_students].homework[i] /=
(double)total_number_students;
}
```

object of arrays

```c
struct grades
{
  int *id;
  double **homework;
  double *average;
};
```



average of each homework

```c
for(i = 0;i < homework_items; i++)
{
  gradesheet.homework[i][total_number_students] = 0.0;
  for(j = 0; j <total_number_students;j++)
  {
      gradesheet.homework[i][total_number_students] +=
gradesheet.homework[i][j];
  }
      gradesheet.homework[i][total_number_students] /=
total_number_students;
}
```

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

Database table layout

- Considering your the most frequently used queries in your database system are similar to
  `SELECT AVG(assignment_1) FROM table`
  Which of the following would be a data structure that better implements the table supporting this type of queries?

A

```
struct record
{
  int id;
  double *assignments;
  double average;
};

struct table
{
  struct record *records
};
```

B

```
struct table
{
  int *id;
  double **assignments;
  double *average;
};
```

What data structure is performing better

	Array of objects	object of arrays

	`struct grades`
	`{`
	`  int id;`
	`  double *homework;`
	`  double average;`
	`};`
		`struct grades`
		`{`
		`  int *id;`
		`  double **homework;`
		`  double *average;`
		`};`

average of each homework	`for(i=0;i<homework_items; i++)`
	`{`
	`gradesheet[total_number_students].homework[i`
	`] = 0.0;`
	`   for(j=0;j<total_number_students;j++)`
	`gradesheet[total_number_students].homework[i`
	`] +=gradesheet[j].homework[i];`
	
	`gradesheet[total_number_students].homework[i`
	`] /= (double)total_number_students;`
	`}`
		`for(i = 0;i < homework_items; i++)`
		`{`
		`  gradesheet.homework[i][total_number_students] =`
		`0.0;`
		`  for(j = 0; j <total_number_students;j++)`
		`  {`
		`      gradesheet.homework[i][total_number_students]`
		`+= gradesheet.homework[i][j];`
		`  }`
		`      gradesheet.homework[i][total_number_students] /`
		`= total_number_students;`
		`}`

Database table layout

- Considering your the most frequently used queries in your database system are similar to
  `SELECT AVG(assignment_1) FROM table`
  Which of the following would be a data structure that better implements the table supporting this type of queries?

A

```
struct record
{
  int id;
  double *assignments;
  double average;
};

struct table
{
  struct record *records
};
```

B

```
struct table
{
  int *id;
  double **assignments;
  double *average;
};
```

Column-store or row-store

- If you're designing an in-memory database system, will you be using

RowId	EmpId	Lastname	Firstname	Salary
1	10	Smith	Joe	40000
2	12	Jones	Mary	50000
3	11	Johnson	Cathy	44000
4	22	Jones	Bob	55000

  - column-store — stores data tables column by column

```
10:001,12:002,11:003,22:004;
Smith:001,Jones:002,Johnson:003,Jones:004;
Joe:001,Mary:002,Cathy:003,Bob:004;
40000:001,50000:002,44000:003,55000:004;
```

if the most frequently used query looks like —

```
select Lastname, Firstname from table
```

  - row-store — stores data tables row by row

```
001:10,Smith,Joe,40000;
002:12,Jones,Mary,50000;
003:11,Johnson,Cathy,44000;
004:22,Jones,Bob,55000;
```

Announcement

- Reading quiz due next Tuesday as usual
  - We will drop two of your lowest reading quizzes
  - No make up, no extension — they are designed to let you "preview", makes no sense of extension and it's unfair.
- Assignment #2 is up, due next Thursday
  - We will drop your lowest scored assignment
    - Please don't email to ask for extension — the dropping policy is to accommodate any potential reason for that
    - We don't accept late assignment
    - Library outside STL is not allowed — gradescope does not have it
  - Start early
    - We don't work 24/7 and we cannot help you last minute
    - Server could get busy last minute, too.
    - Gradescope has different test cases than released ones to prevent any shortcut of performance results — you have to test your code carefully to prevent failed execution on gradescope.
- Midterm is 5/9 in class, in-person, no-cheatsheet-allowed
  - We will provide a sample midterm at the end of slides on next Thursday

Computer
Science &
Engineering

つづく