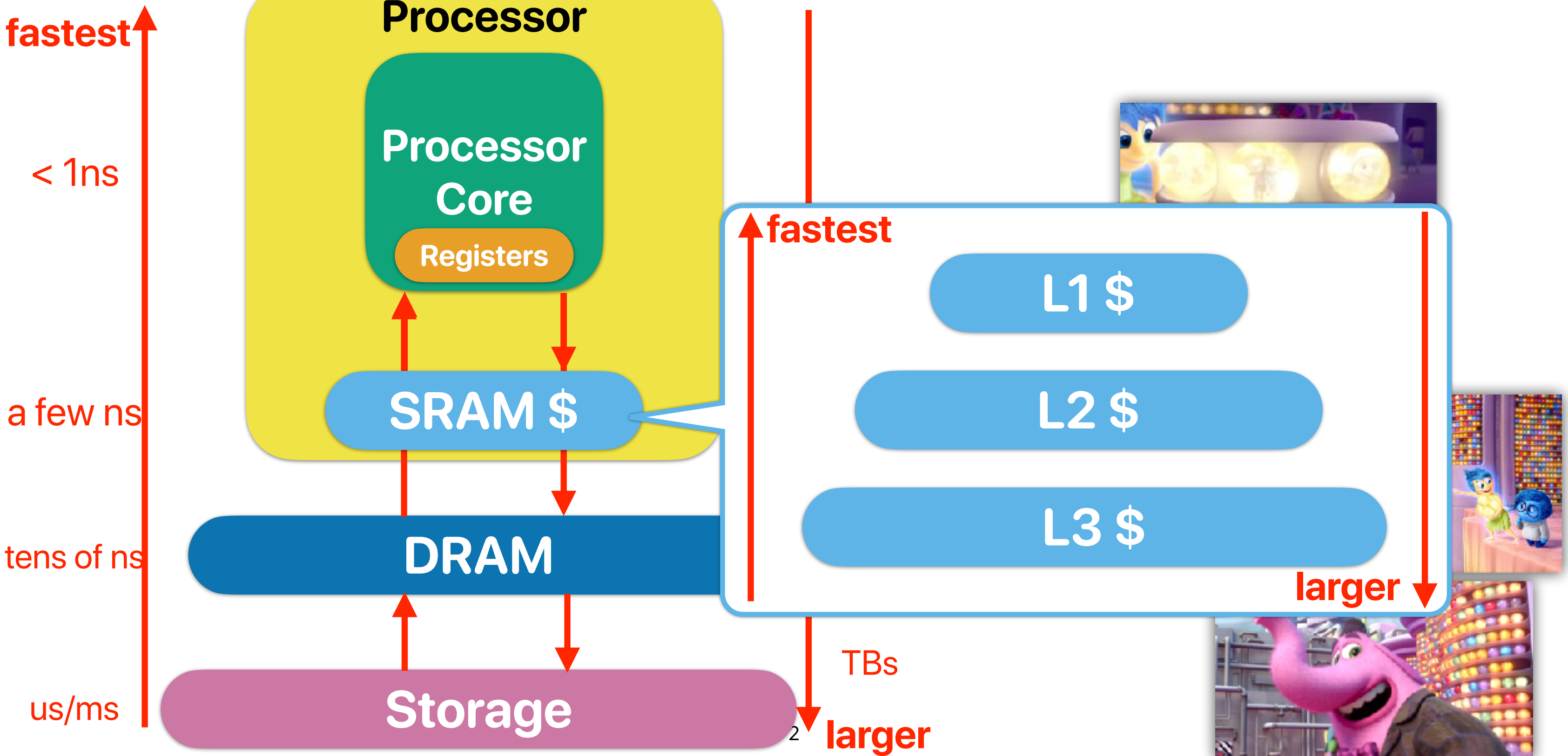


Lab 2: Memory

Hung-Wei Tseng

Recap: Memory Hierarchy



Know your cache configuration

- `getconf -a | grep CACHE`

```
[6]: !cse142 job run ' getconf -a | grep CACHE'
```

LEVEL1_ICACHE_SIZE	32768
LEVEL1_ICACHE_ASSOC	8
LEVEL1_ICACHE_LINESIZE	64
LEVEL1_DCACHE_SIZE	49152
LEVEL1_DCACHE_ASSOC	12
LEVEL1_DCACHE_LINESIZE	64
LEVEL2_CACHE_SIZE	1310720
LEVEL2_CACHE_ASSOC	10
LEVEL2_CACHE_LINESIZE	64
LEVEL3_CACHE_SIZE	12582912
LEVEL3_CACHE_ASSOC	12
LEVEL3_CACHE_LINESIZE	64
LEVEL4_CACHE_SIZE	0
LEVEL4_CACHE_ASSOC	0
LEVEL4_CACHE_LINESIZE	0

C
A
B

$$C = A \times B \times S$$

$$48 \times 1024 = 12 \times 64 \times S$$

$$S = 64$$

$$\text{Offset} = \log_2(64) = 6$$

$$\text{Index} = \log_2(64) = 6$$

$$\text{Tag} = 64 - 6 - 6 = 52$$

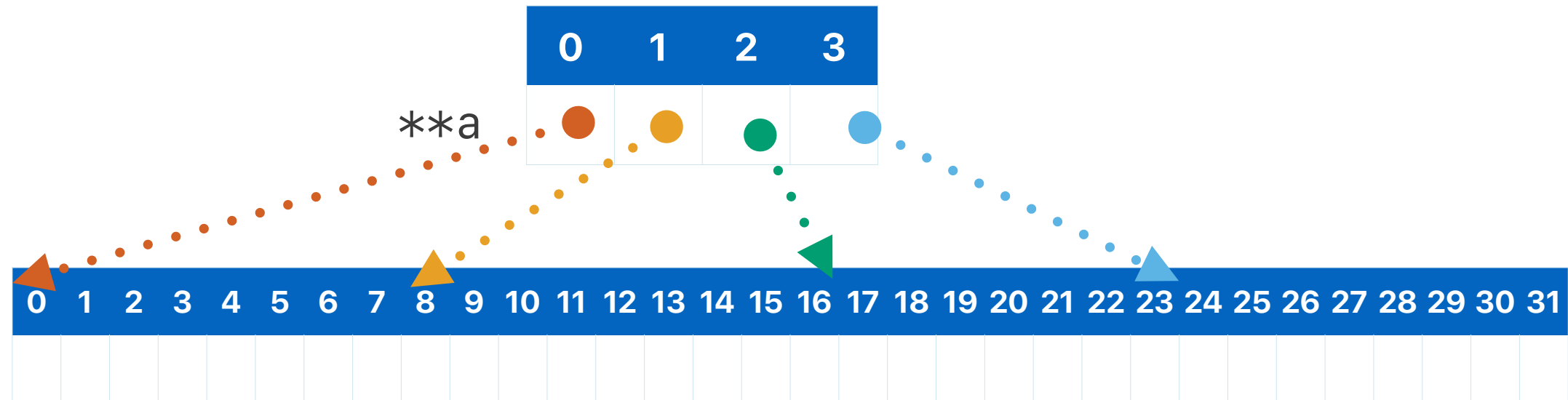
What is an M by N "2-D" array in C?

```
a = (double **)malloc(M*sizeof(double *));  
for(i = 0; i < N; i++)  
{  
    a[i] = (double *)malloc(ARRAY_SIZE*sizeof(double));  
}
```

abstraction

	0	1	2	3	4	5	6	7
0								
1								
2								
3								

physical implementation



Q2: Where does your code access data memory?

- gcc -S can generate the assembly



Q2: From C to Assembly

x86 instruction: op src, dst

```
for(i = 0; i < ARRAY_SIZE; i++)  
{  
    result = 0.0;  
    for(j = 0; j < ARRAY_SIZE; j++)  
    {  
        result += a[i][j]*b[j];  
    }  
    c[i] = result;  
}
```

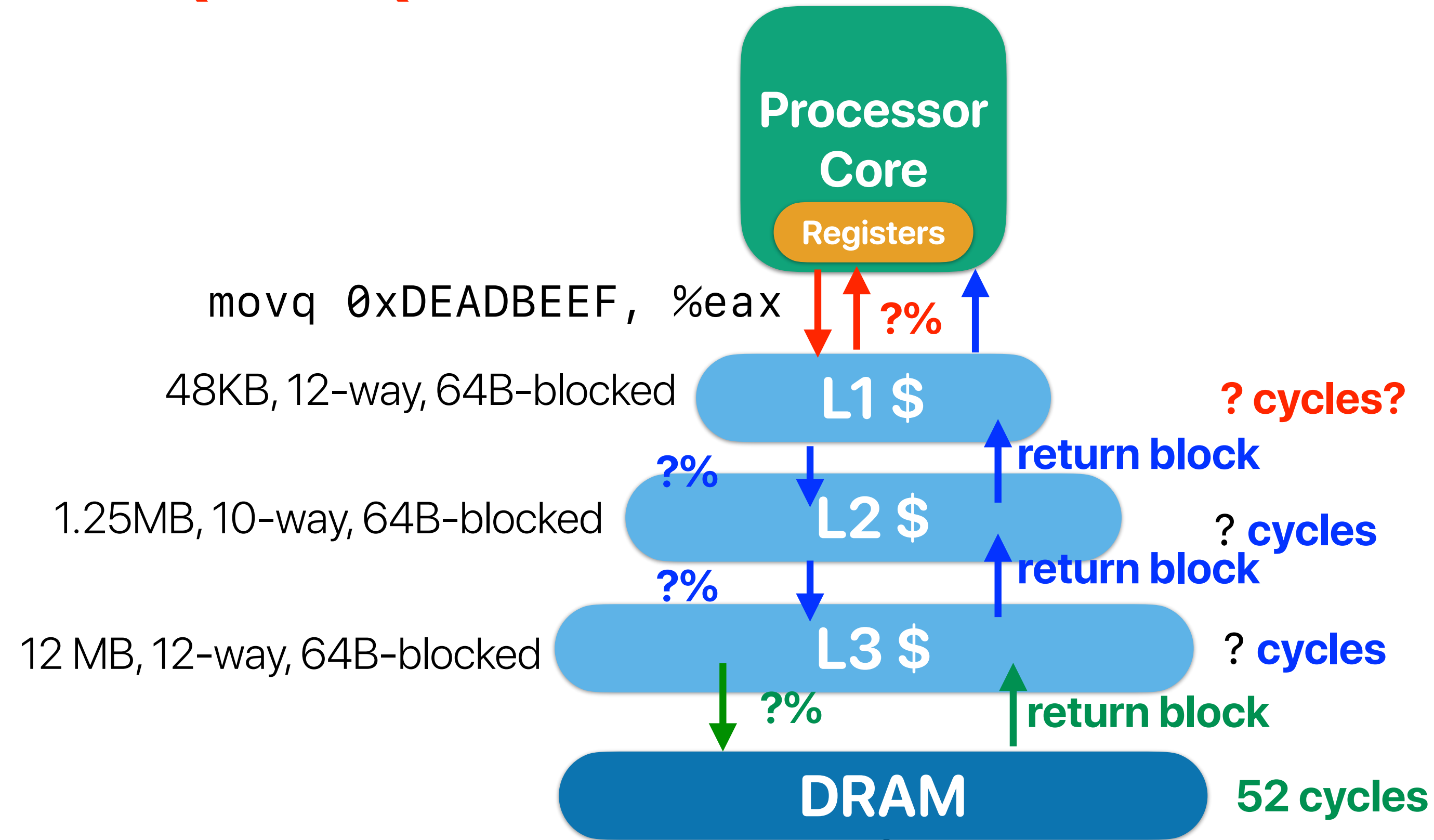
```
.L26:      store memory op src->mem  
movsd %xmm1, 0(%rbp,%rdi,8)  
leaq 1(%rdi), %rax  
cmpq %rdi, %rcx  
je .L15  
movq %rax, %rdi  
.L13:      load memory op mem->dst  
movq 0(%r13,%rdi,8), %rsi  
movl $0, %eax  
pxor %xmm1, %xmm1  
.L14:      load memory op mem->dst  
movsd (%rsi,%rax,8), %xmm0  
mulsd (%rbx,%rax,8), %xmm0  
addsd %xmm0, %xmm1  
movq %rax, %rdx  
addq $1, %rax  
cmpq %rdx, %rcx  
jne .L14  
jmp .L26
```

fp mul op src*dst->dst

fp add op src+dst->dst

+1 op src+dst->dst

Q4 & Q5: Performance with multi-level \$

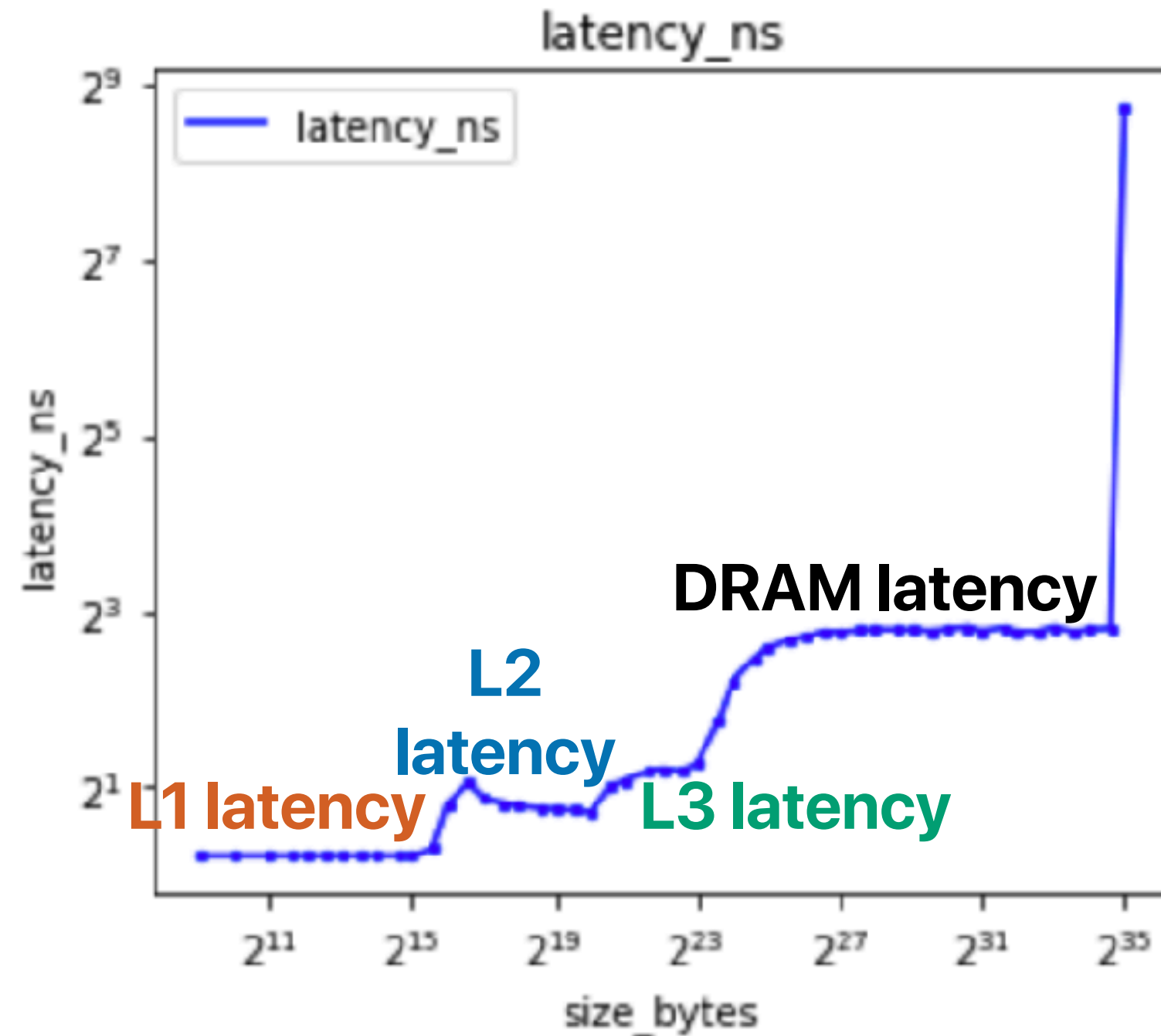


How fast is my memory hierarchy?

The tool

- Memory Latency
 - <https://www.intel.com/content/www/us/en/developer/articles/tool/intelr-memory-latency-checker.html>
- lat_mem_rd
- Spec
 - dmidecode — for DRAM parameters
 - lscpu — for on-CPU caches

Q4: lat_mem_rd



[6]: !cse142 job run ' getconf -a | grep CACHE '

LEVEL1_ICACHE_SIZE	32768
LEVEL1_ICACHE_ASSOC	8
LEVEL1_ICACHE_LINESIZE	64
LEVEL1_DCACHE_SIZE	49152
LEVEL1_DCACHE_ASSOC	12
LEVEL1_DCACHE_LINESIZE	64
LEVEL2_CACHE_SIZE	1310720
LEVEL2_CACHE_ASSOC	10
LEVEL2_CACHE_LINESIZE	64
LEVEL3_CACHE_SIZE	12582912
LEVEL3_CACHE_ASSOC	12
LEVEL3_CACHE_LINESIZE	64
LEVEL4_CACHE_SIZE	0
LEVEL4_CACHE_ASSOC	0
LEVEL4_CACHE_LINESIZE	0

$$1.5 \times 2^{15}$$

$$1.25 \times 2^{20}$$

$$1.5 \times 2^{23}$$

Memory performance when running applications

The steps of solving a cache simulation problem

- Use $C = A \ B \ S$ to figure out how to partition the requesting memory addresses
 - If in the test/assignment, A, B, and C are typically known
 - In real practice, use commands like ``getconf -a|grep CACHE`` to obtain A, B, C or Google the processor's data sheet
- Looking into the code, produce the address sequence
 - If in the test/assignment, we give you the base addresses of data structures
 - In real practice, You may use gdb to investigate the memory addresses
- Partition the addresses into tag, index, offsets
- Use a table to track the current status in the cache

Q5: cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
 - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384];
/* a = 0x20000 */
for(i = 0; i < size; i+=stride) {
    sum += a[i];
    //load a, b, and then store to c
}
```

What's the data cache miss rate for this code when **stride = 1** and size = 8192?

```
double a[16384];
/* a = 0x20000 */
for(i = 0; i < size; i+=stride) {
    sum += a[i];
    //load a, b, and then store to c
}
```

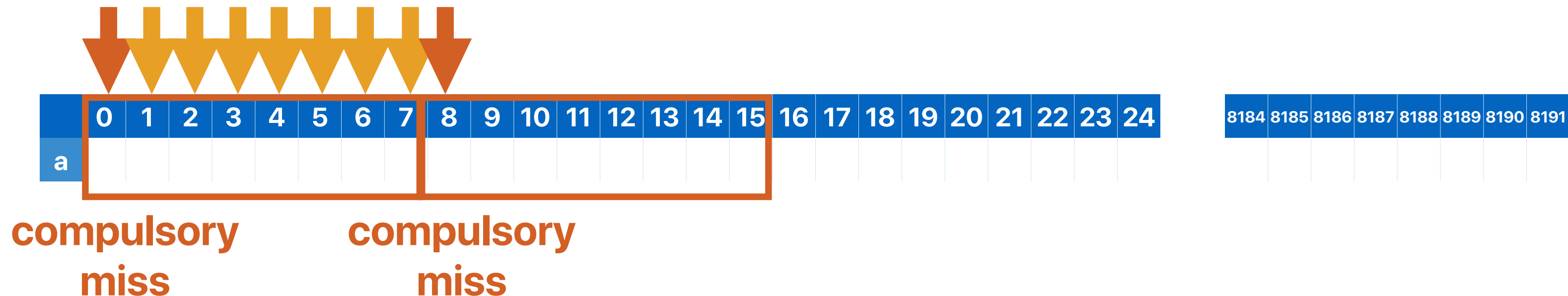
		Addres	Tag	Index	Hit? Miss?	
a[0]		20000	20	0	Miss	a[0]–a[7] in set 0
a[1]		20008	20	0	Hit	
a[2]		20010	20	0	Hit	
a[3]		20018	20	0	Hit	
a[4]		20020	20	0	Hit	
a[5]		20028	20	0	Hit	
a[6]		20030	20	0	Hit	
a[7]		20038	20	0	Hit	
a[8]		20040	20	1	Miss	a[8]–a[15] in set 1
a[9]		20048	20	1	Hit	
a[10]		20050	20	1	Hit	
a[11]		20058	20	1	Hit	
a[12]		20060	20	1	Hit	
a[13]		20068	20	1	Hit	
a[14]		20070	20	1	Hit	
a[15]		20078	20	1	Hit	
a[16]		20080	20	2	Miss	a[16]–a[23] in set 2
a[17]		20088	20	2	Hit	
a[18]		20090	20	2	Hit	
a[19]		20098	20	2	Hit	
a[20]		200A0	20	2	Hit	
a[21]		200A8	20	2	Hit	
a[22]		200B0	20	2	Hit	
a[23]		200B8	20	2	Hit	
a[24]		200C0	20	3	Miss	a[24]–a[31] in set 3
a[25]		200C8	20	3	Hit	

Q5: cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
 - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384];  
/* a = 0x20000 */  
for(i = 0; i < size; i+=stride) {  
    sum += a[i];  
    //load a, b, and then store to c  
}
```

^{hit}
What's the data cache miss rate for this code when **stride = 1** and size = 8192?

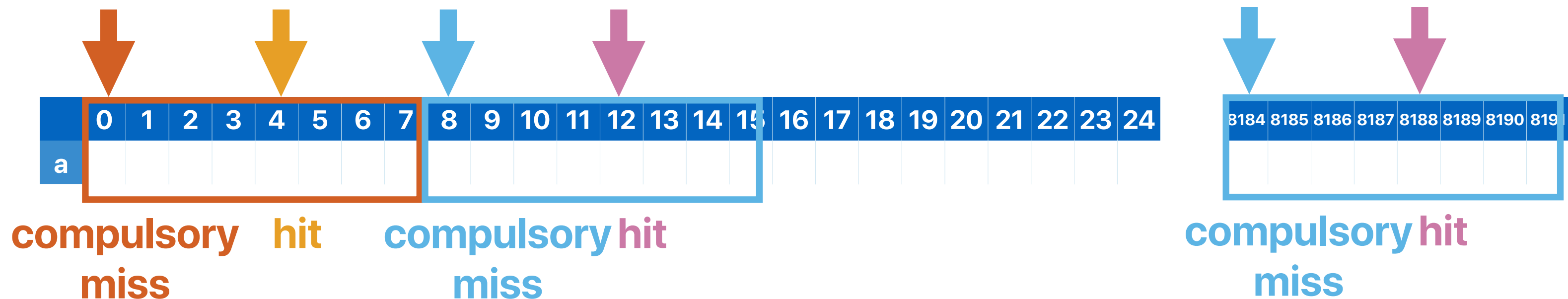


Q5: cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
 - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384];
/* a = 0x20000 */
for(i = 0; i < size; i+=stride) {
    sum += a[i];
    //load a, b, and then store to c
}
```

What's the data cache miss rate for this code when **stride = 4** and size = 8192?

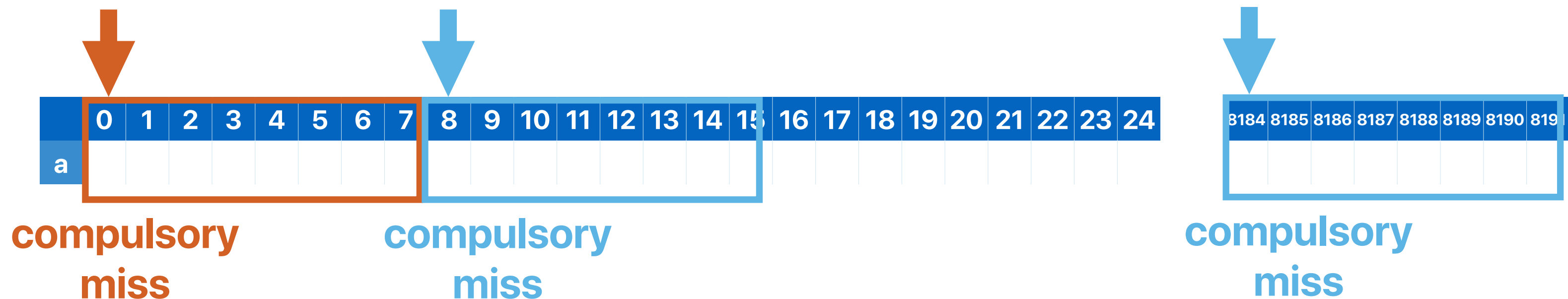


Q5: cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
 - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384];  
/* a = 0x20000 */  
for(i = 0; i < size; i+=stride) {  
    sum += a[i];  
    //load a, b, and then store to c  
}
```

What's the data cache miss rate for this code when **stride = 8** and size = 8192?

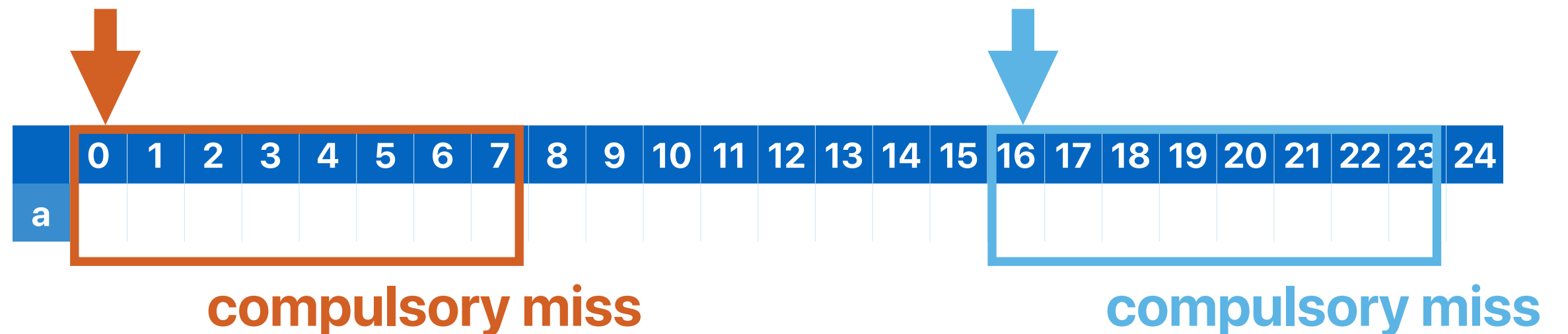


Q5: cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
 - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384];  
/* a = 0x20000 */  
for(i = 0; i < size; i+=stride) {  
    sum += a[i];  
    //load a, b, and then store to c  
}
```

What's the data cache miss rate for this code when **stride = 16** and size = 8192?



Q5 and Q8: cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
 - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

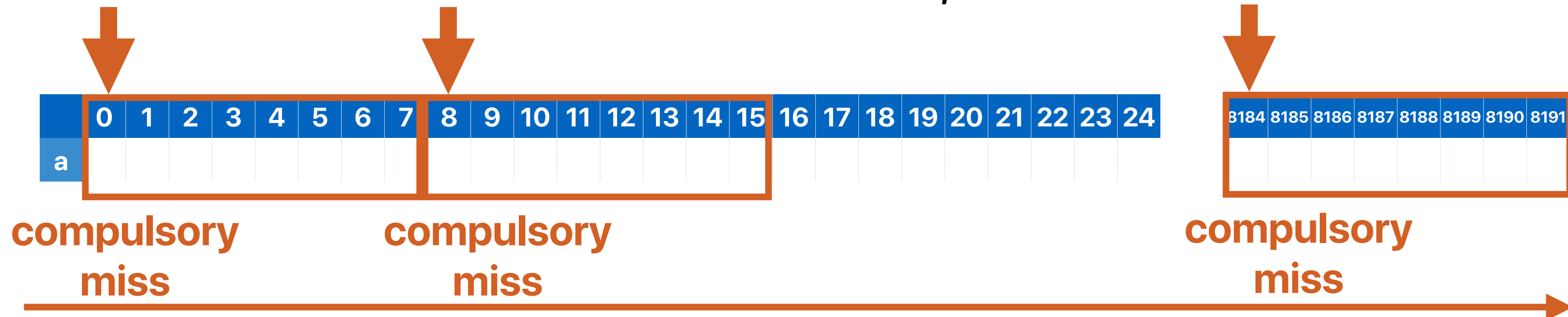
```
double a[16384];
/* a = 0x20000 */
for(j = 0; j < runs; j++)
    for(i = 0; i < size; i+=stride) {
        sum += a[i];
        //load a, b, and then store to c
    }
}
```

We have a total of $\frac{8192}{8} = 1024$ blocks

We need $1024 \times 64 = 64$ KB capacity to keep them

So, it will be miss the next time we revisit a[0]
— capacity miss

What's the data cache miss rate for this code when **stride = 8**, **runs = 4** and **size = 8192**?



Q5 and Q8: cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
 - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

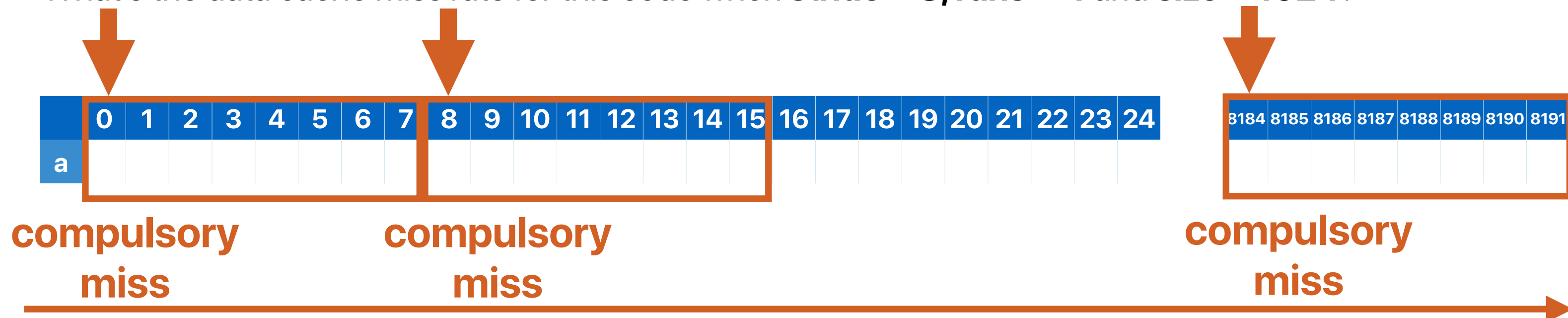
```
double a[16384];
/* a = 0x20000 */
for(j = 0; j < runs; j++)
    for(i = 0; i < size; i+=stride) {
        sum += a[i];
        //load a, b, and then store to c
    }
}
```

We have a total of $\frac{1024}{8} = 128$ blocks

We need $128 \times 64 = 8$ KB capacity < 48KB

We may be able to have them in the cache(?)

What's the data cache miss rate for this code when **stride = 8**, **runs = 4** and **size = 1024**?



Q8: Quick intuition

$$C = A \times B \times S$$

$$48 \times 1024 = 12 \times 64 \times S$$

$$S = 64$$


We have a total of $\frac{1024}{8} = 128$ *blocks*

2 blocks per set

Each set can store 12 blocks! So we're good!

Q8: in details

	Address	Tag	Index	Offset	Hit/Miss?
a[0]	0x20000	0x20	0x0	0x0	Miss
a[8]	0x20040	0x20	0x1	0x0	Miss
a[16]	0x20080	0x20	0x2	0x0	Miss
a[24]	0x200C0	0x20	0x3	0x0	Miss
a[504]	0x20FC0	0x20	0x3F	0x0	Miss
a[512]	0x21000	0x21	0x0	0x0	Miss
a[520]	0x21040	0x21	0x1	0x0	Miss
a[1016]	0x21FC0	0x21	0x3F	0x0	Miss
a[0]	0x20000	0x20	0x0	0x0	Hit
a[8]	0x20040	0x20	0x1	0x0	Hit
a[16]	0x20080	0x20	0x2	0x0	Hit
a[24]	0x200C0	0x20	0x3	0x0	Hit


$$\frac{1024}{8} = 128 \text{ blocks}$$

Brain storm ... what if ...

	Address	Tag	Index	Offset	Hit/Miss?
a[0]	0x20000	0x20	0x0	0x0	Miss
a[512]	0x21000	0x21	0x0	0x0	Miss
a[1024]	0x22000	0x22	0x0	0x0	Miss
a[1536]	0x23000	0x23	0x0	0x0	Miss
	0x24000	0x24	0x0	0x0	Miss
	0x25000	0x25	0x0	0x0	Miss
	0x26000	0x26	0x0	0x0	Miss
	0x27000	0x27	0x0	0x0	Miss
	0x28000	0x28	0x0	0x0	Miss
	0x29000	0x29	0x0	0x0	Miss
	0x2A000	0x2A	0x0	0x0	Miss
	0x2B000	0x2B	0x0	0x0	Miss
	0x2C000	0x2C	0x0	0x0	Miss
a[0]	0x20000	0x20	0x0	0x0	Miss



Each set can store only 12 blocks! So we cannot hold 13 blocks — kick the oldest one out

Q11: Stride at 2^n v.s. $2^n +/- x$

	Address	Tag	Index	Offset	Hit/Miss?
a[0]	0x20000	0x20	0x0	0x0	Miss
a[504]	0x20FC0	0x20	0x3F	0x0	Miss
a[1008]	0x21F80	0x21	0x3E	0x0	Miss

all in different sets — more likely to hit if we revisit them

	Address	Tag	Index	Offset	Hit/Miss?
a[0]	0x20000	0x20	0x0	0x0	Miss
a[520]	0x21040	0x21	0x1	0x0	Miss
a[1040]	0x22080	0x22	0x2	0x0	Miss

all in different sets — more likely to hit if we revisit them

Q5 and Q8: cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
 - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384];
/* a = 0x20000 */
for(j = 0; j < runs; j++)
    for(i = 0; i < size; i+=stride) {
        sum += a[i];
        //load a, b, and then store to c
    }
}
```

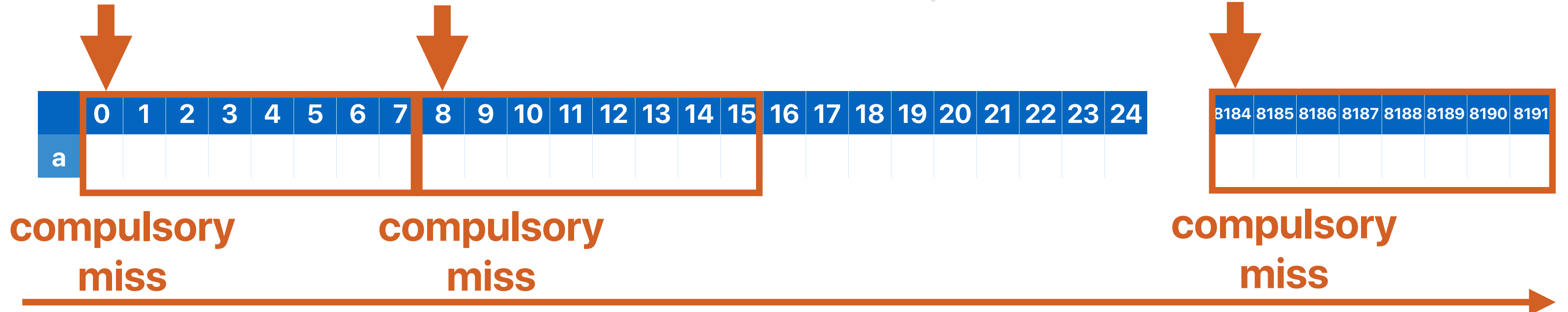
We have a total of $\frac{8192}{8} = 1024$ blocks

We need $1024 \times 64 = 64$ KB capacity to keep them

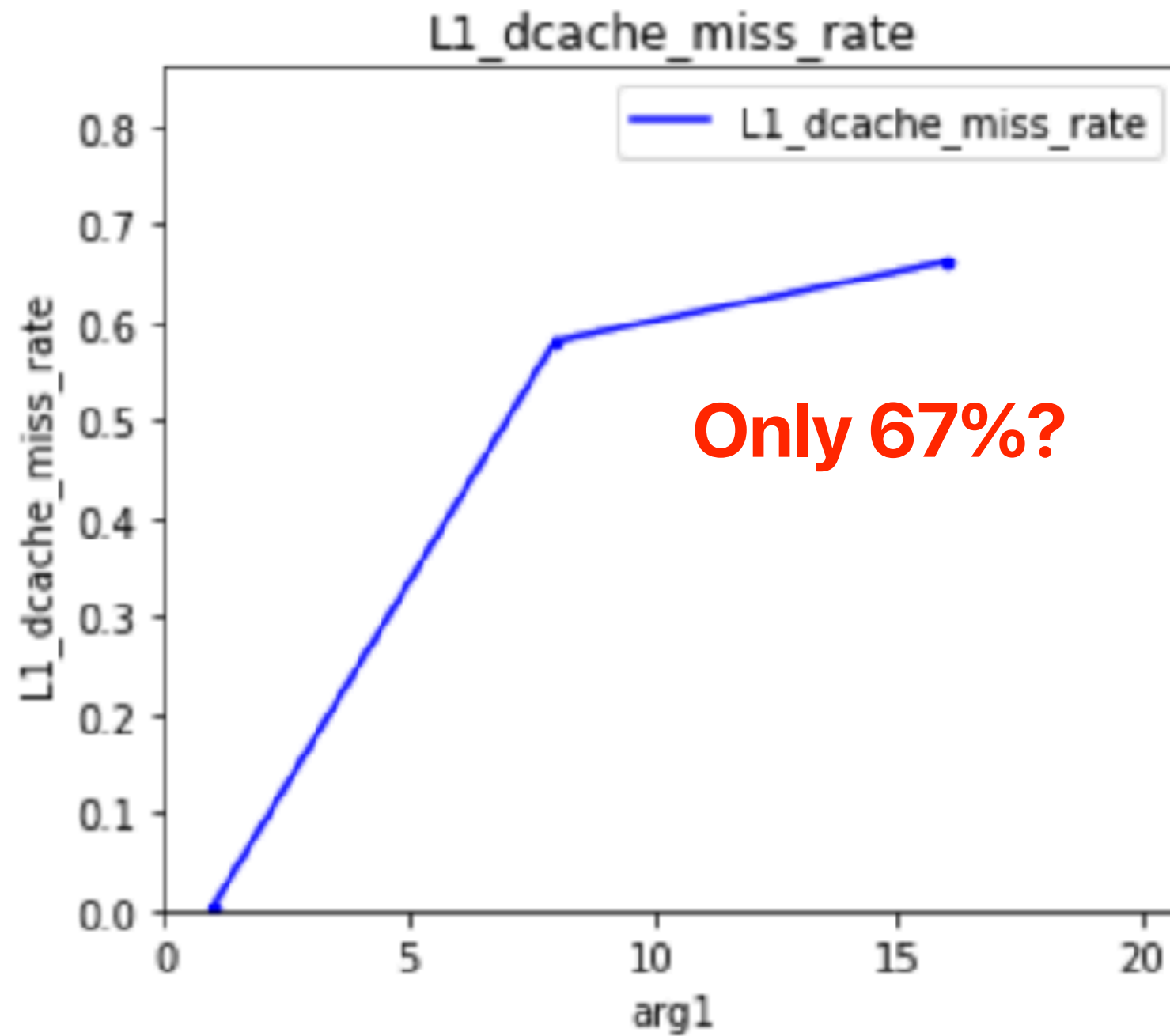
So, it will be miss the next time we revisit a[0] — capacity miss

What's the data cache miss rate for this code when stride = 8, runs = 4 and size = 8192?

I expect 100% miss rate but...

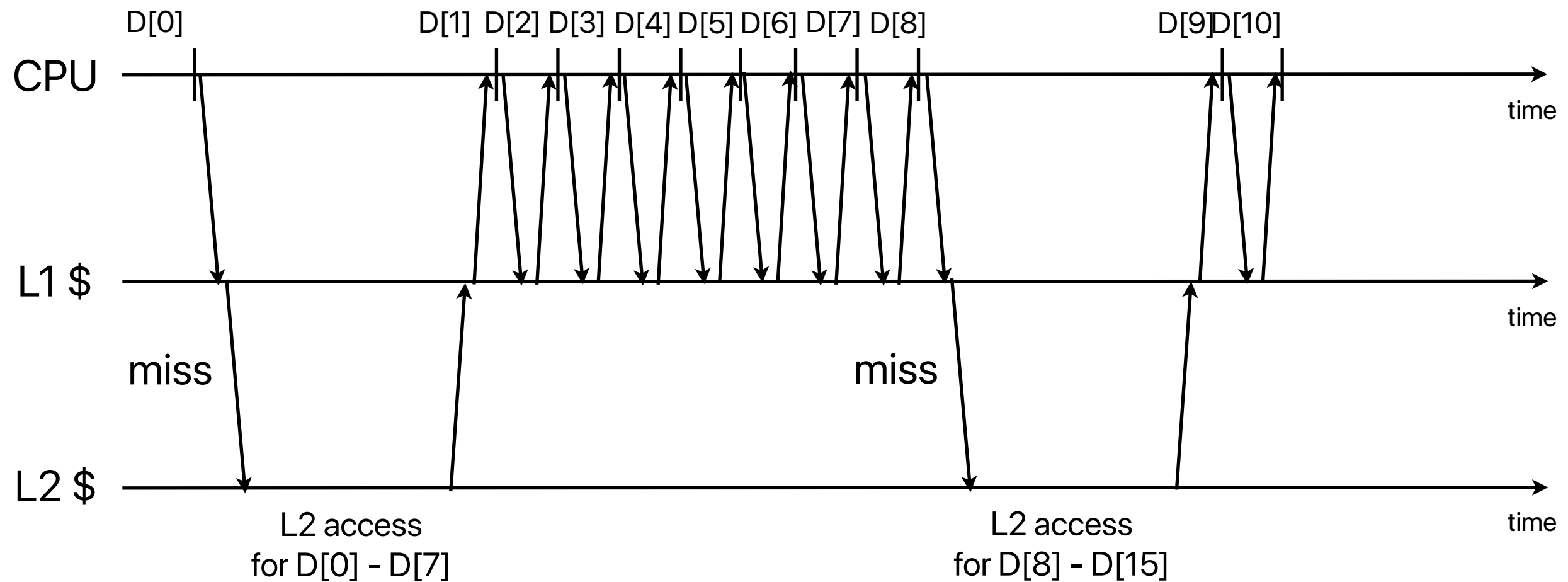


I expected 100%...



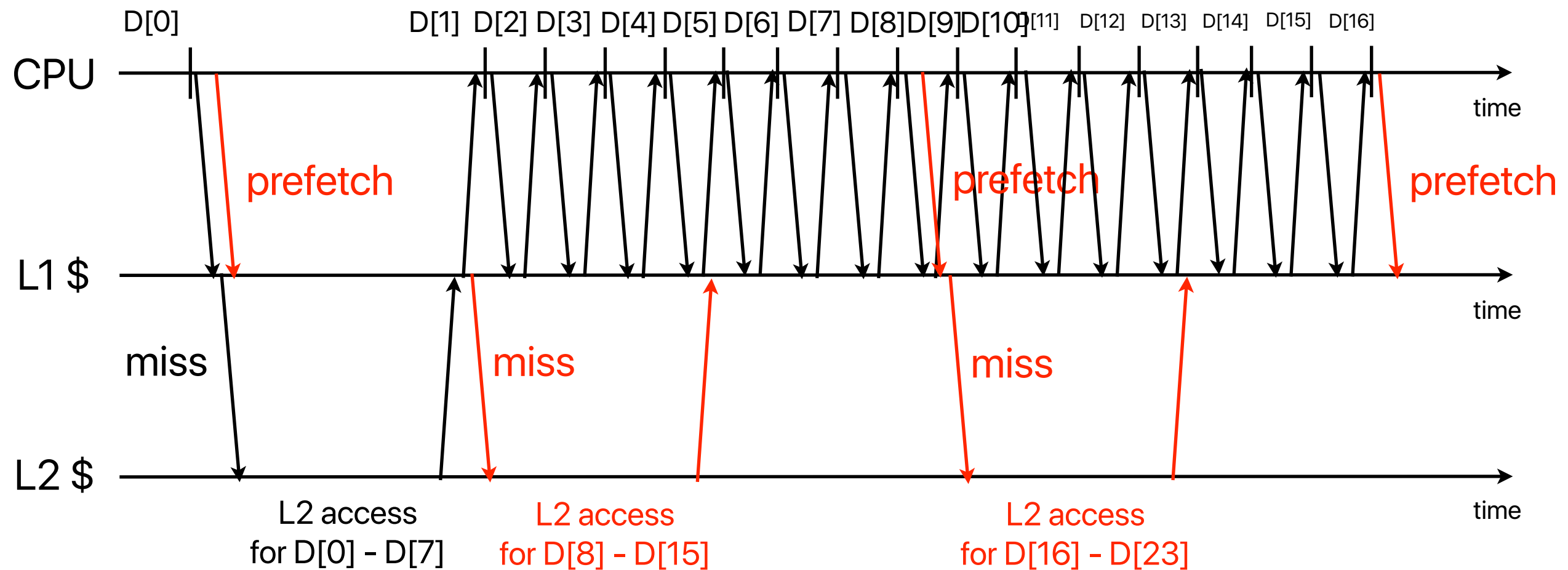
Characteristic of memory accesses

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



Hardware Prefetching

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



Announcement

- Lab Report #2 due this Saturday
- Programming assignment due **next** Saturday
 - 16x speedup on gradescope
 - You may check "leaderboard" of the top performing one
 - Top 3 will have the chance of A+
- No lab lecture this Thursday
 - Hung-Wei will extend the office hour to 6p @ CSE2138 on Thursday
 - If you have questions regarding midterm, that's the best time to ask

Computer
Science &
Engineering

142L

つづく



Locality

- Spatial locality — application tends to visit nearby stuffs in the memory
 - Code — the current instruction, and then $PC + 4$
 - Data — the current element in an array, then the next
- Temporal locality — application revisit the same thing again and again
 - Code — loops, frequently invoked functions
 - Data — the same data can be read/write many times

If we want to improve the memory performance of our code, we have to make our code better exploit "localities"