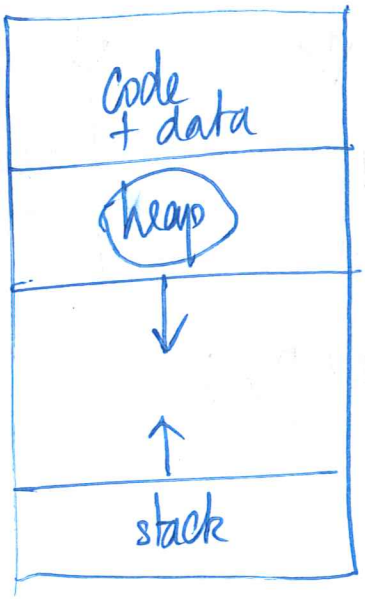
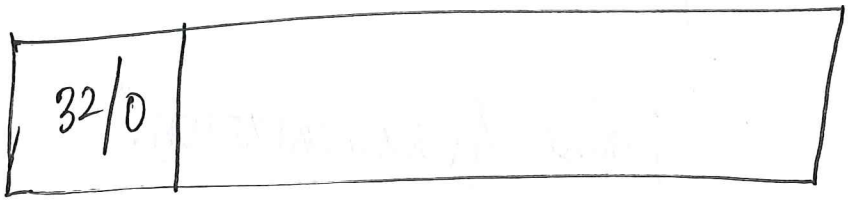
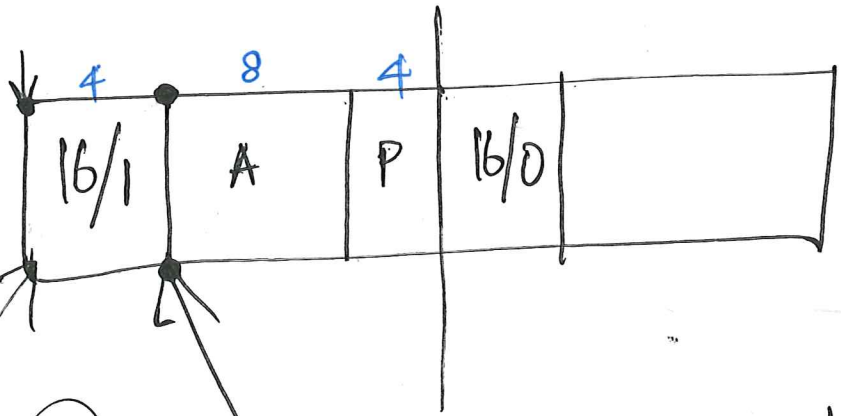
CS 354 - Lecture 11

Review



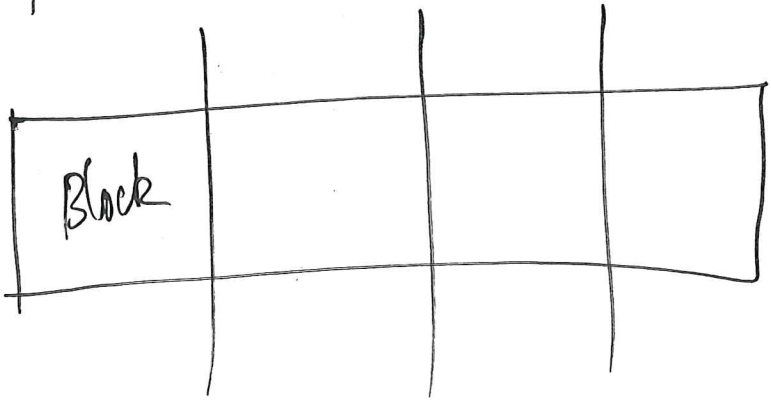


8
+4
12
+4
16

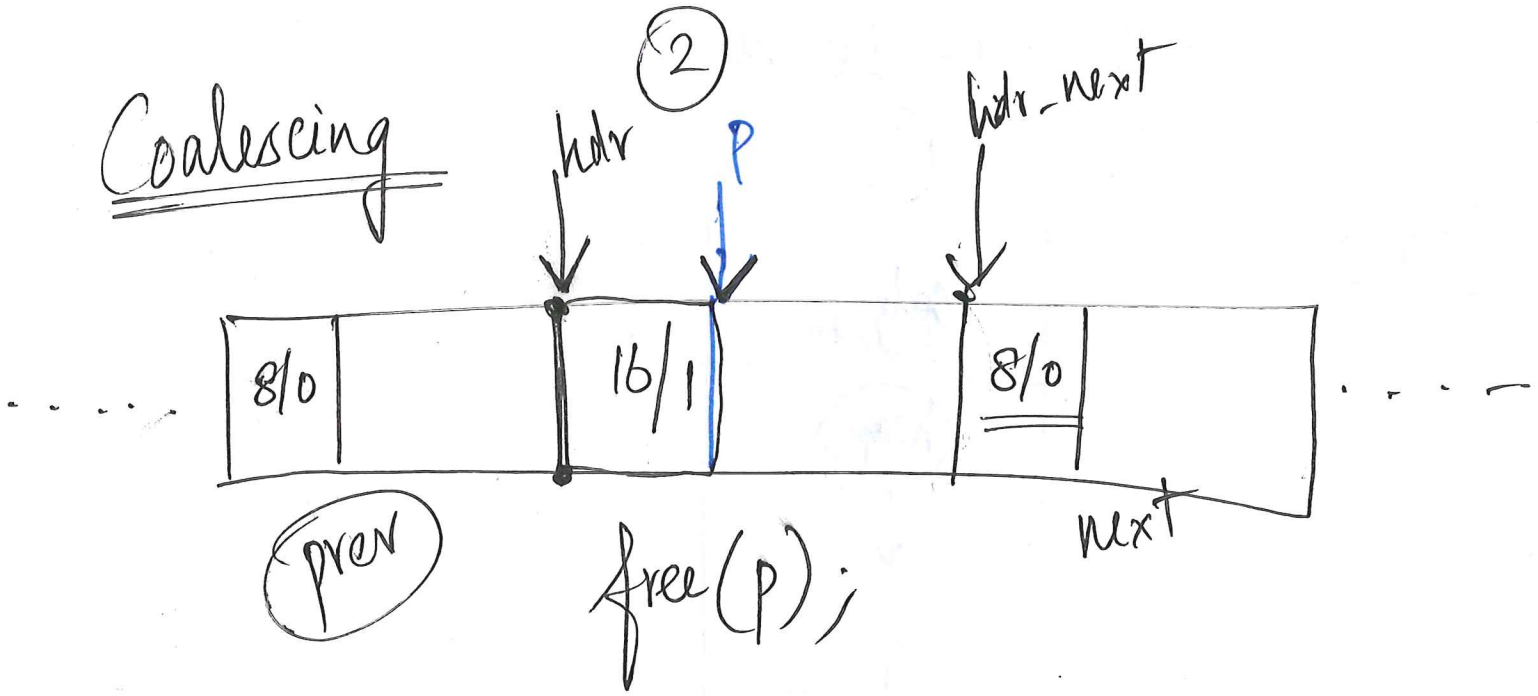


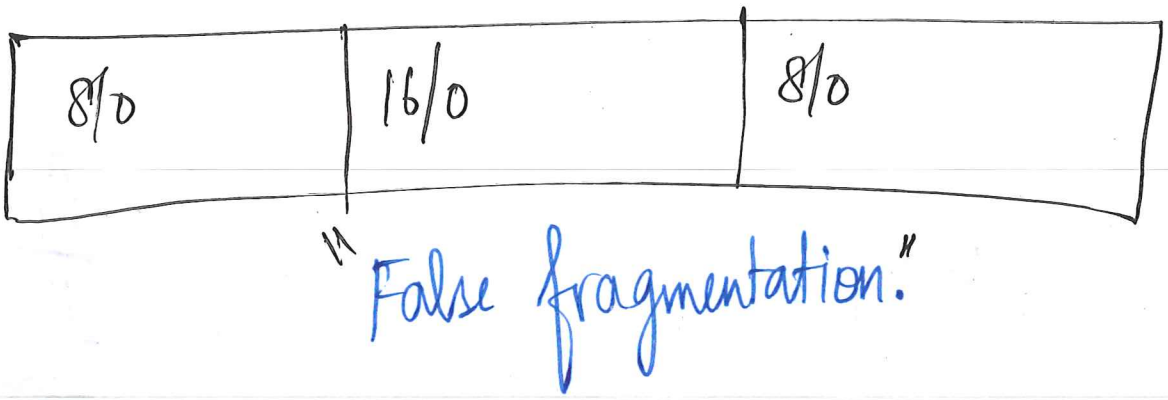
0x8090A0B4

B8

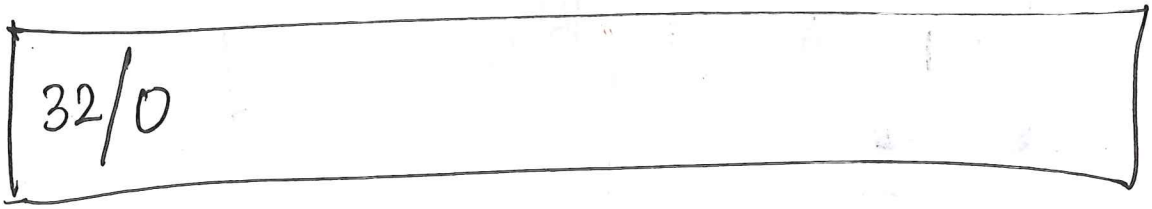


Coalescing



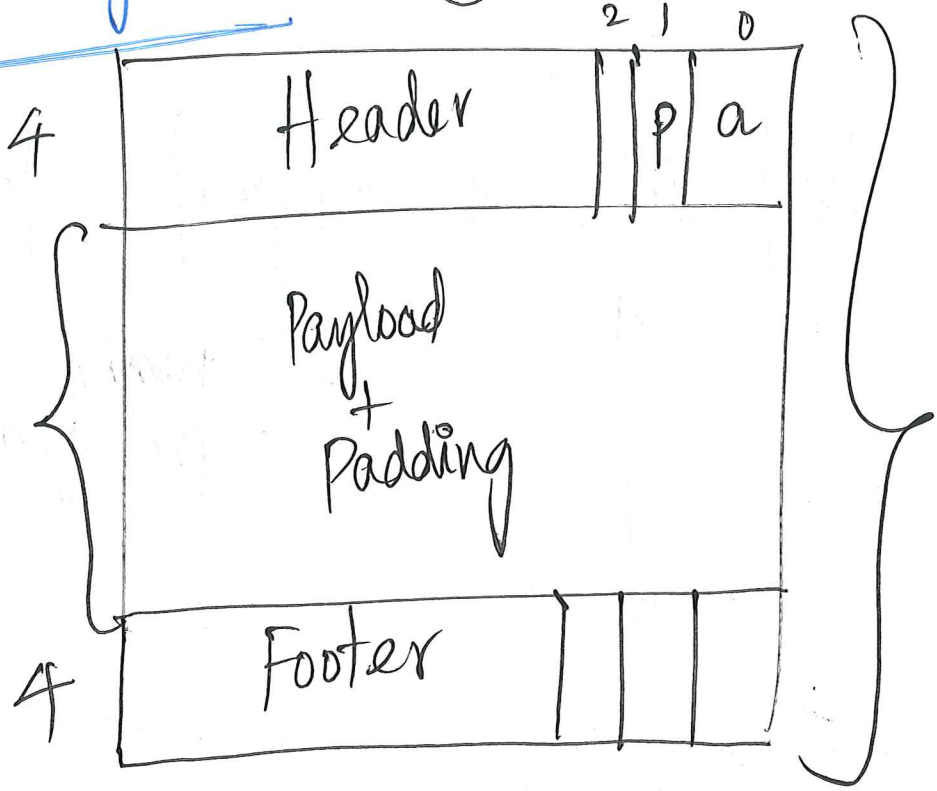


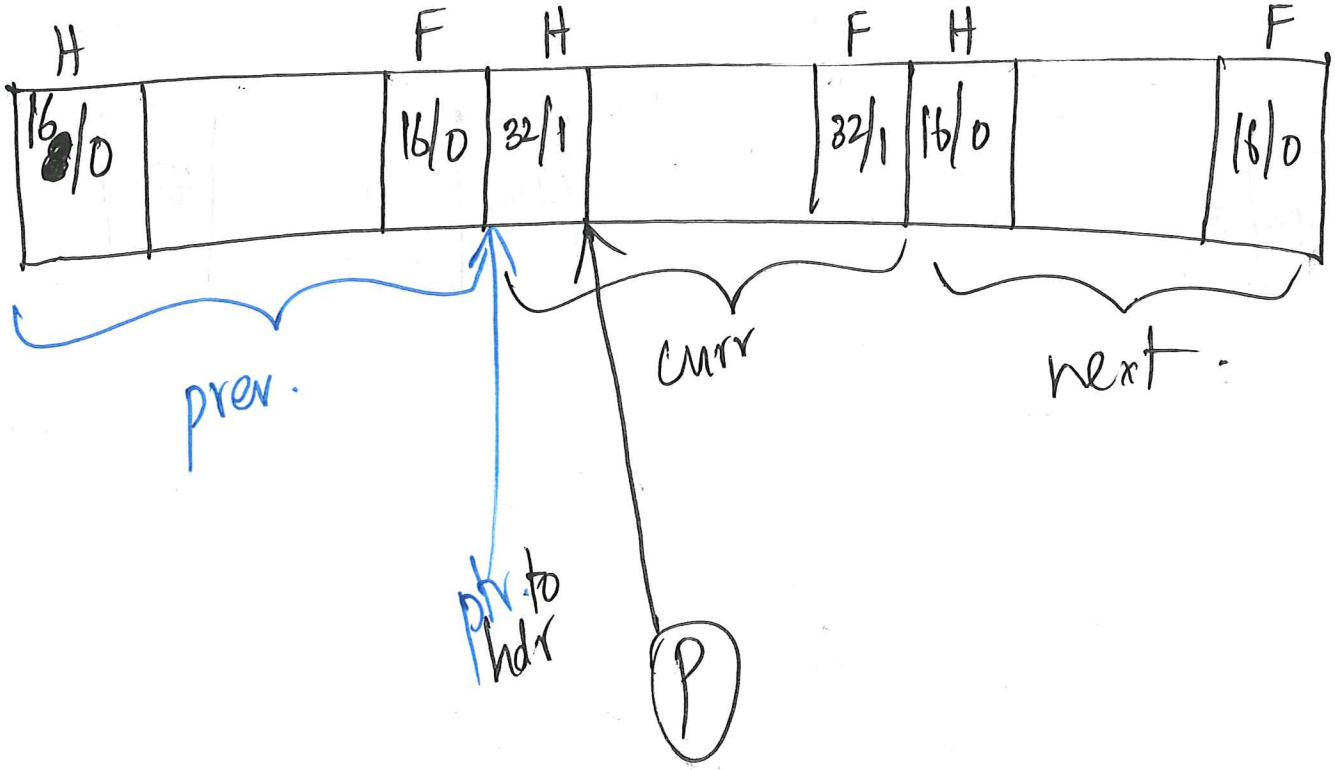
"False fragmentation."



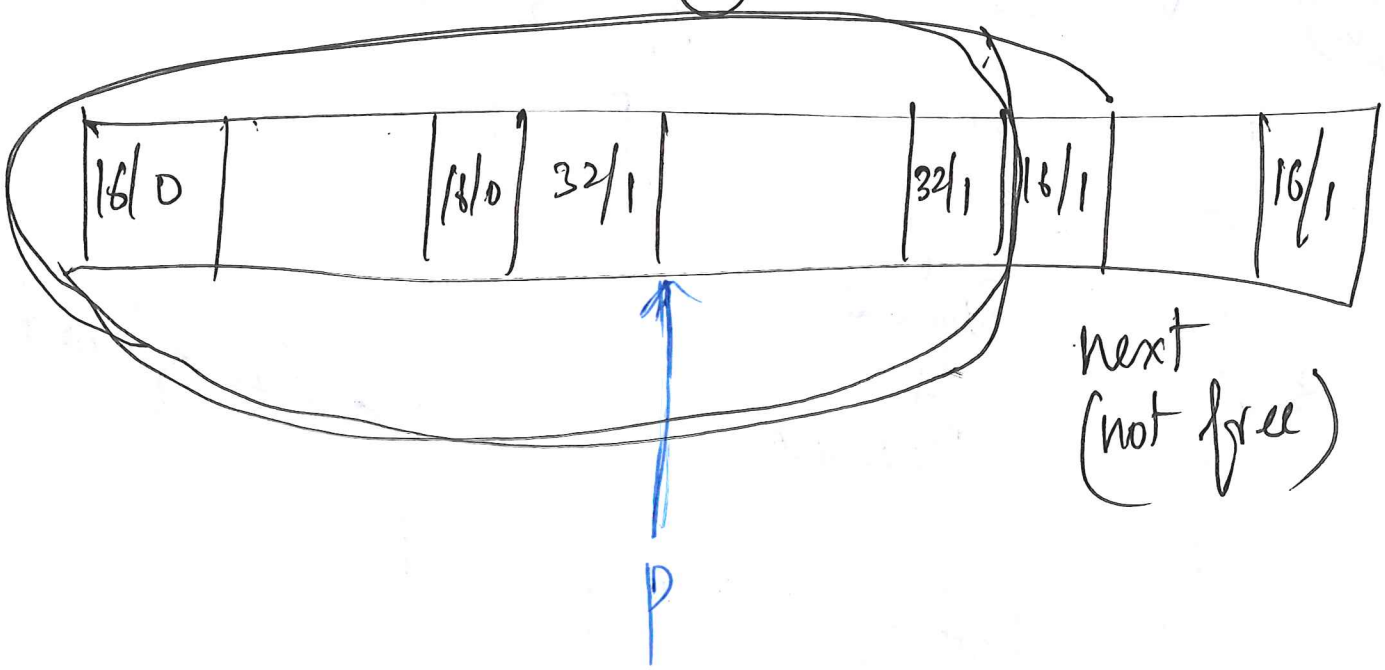
2 Word alignment (8 bytes)

(3)

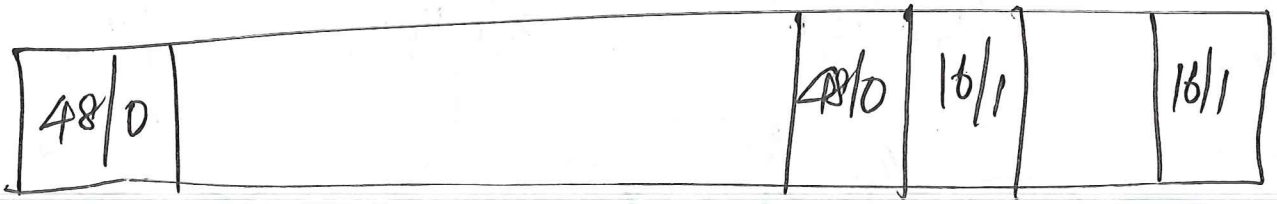




(4)

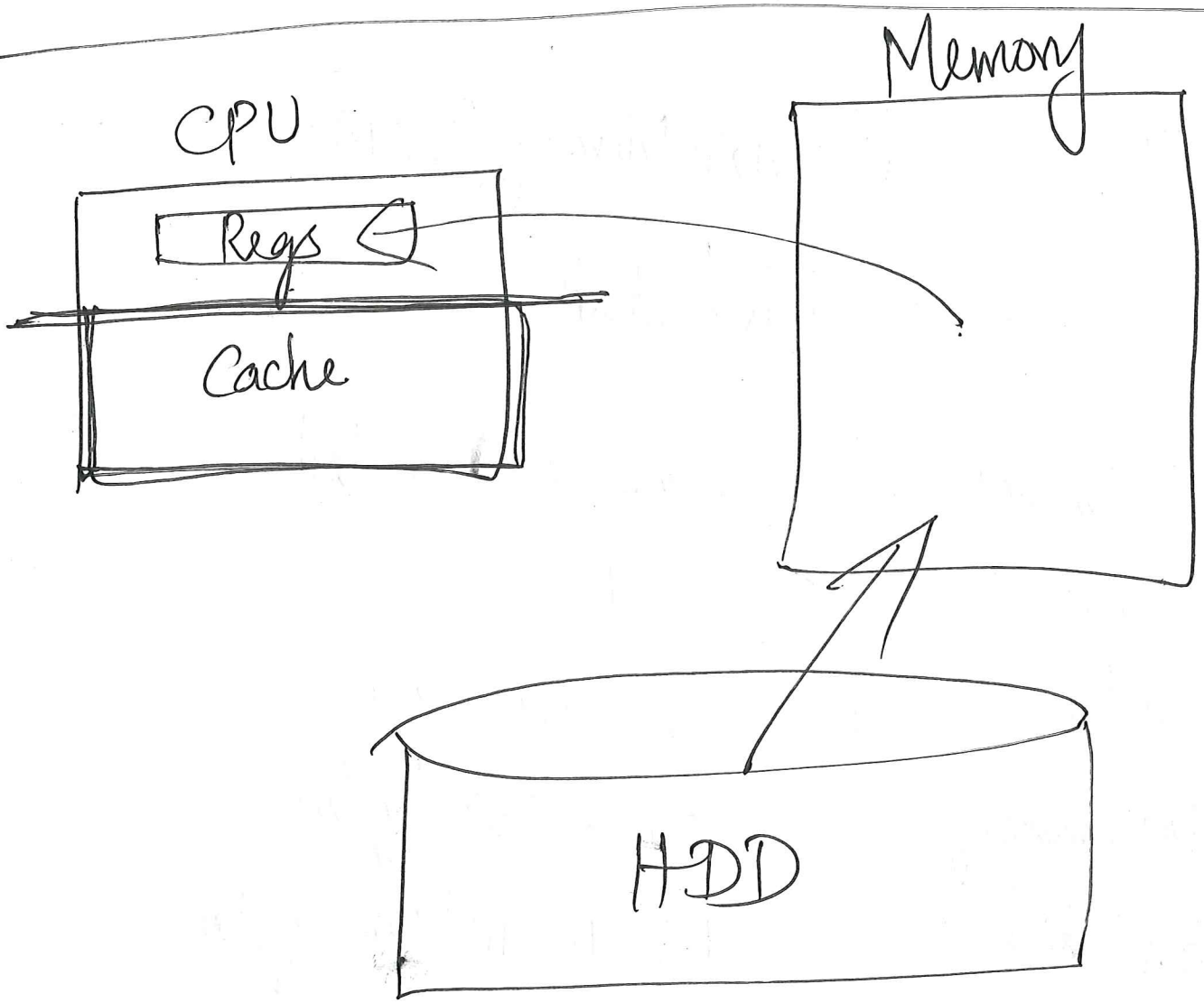


free(p)

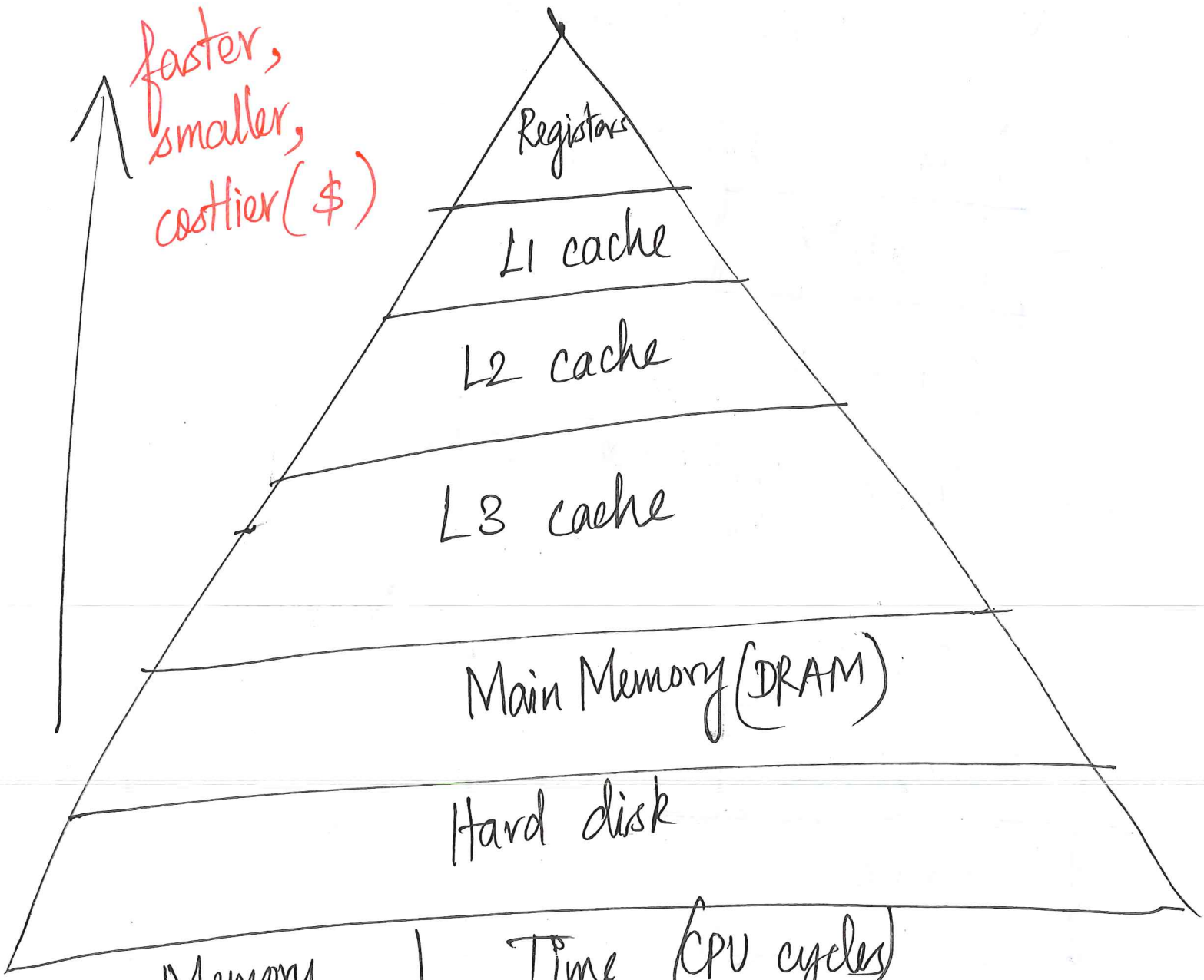


Today: "Cache Memories"

1. Memory hierarchy
2. Locality
3. Caches
4. Cache-efficient code



Memory hierarchy

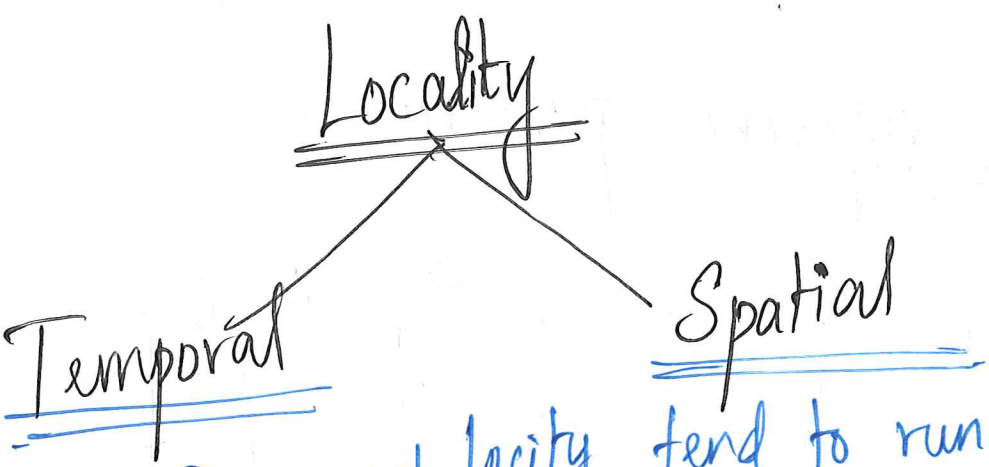


Memory	Time (CPU cycles)
Register	1
Cache	1 – 30 cycles
Main memory	50 – 200 cyles.
Hard disk.	1 – 10 million cyles

Locality

Program - accesses the same data item again & again

- accesses sets of nearby data items.

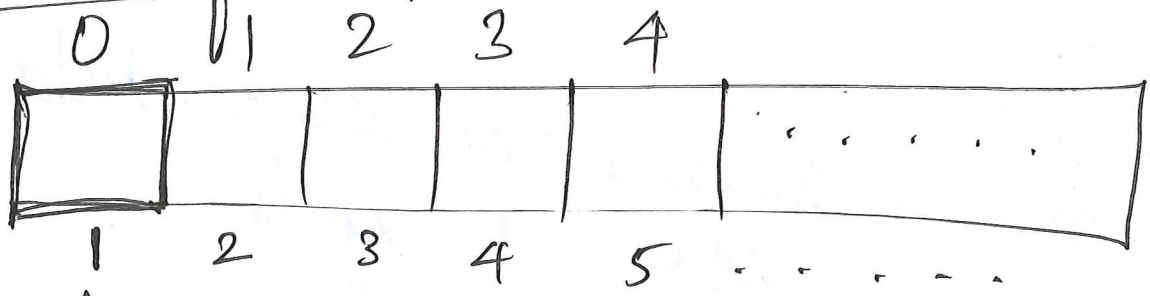


Why? ⟹ Pgms w/ locity tend to run faster.

```
for ( int i = 0 ; i < n ; i++ )
    sum += a[i] ;
```

Principle of locality
Locality of reference.

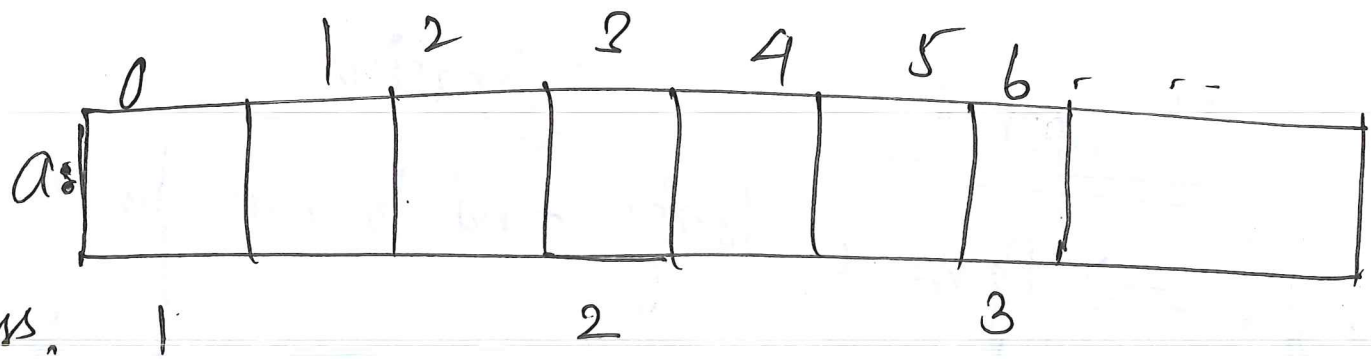
Sequential Reference Pattern (S)



Access order: stride-1 reference pattern.

---

stride-k reference pattern



Access order: 1 2 3

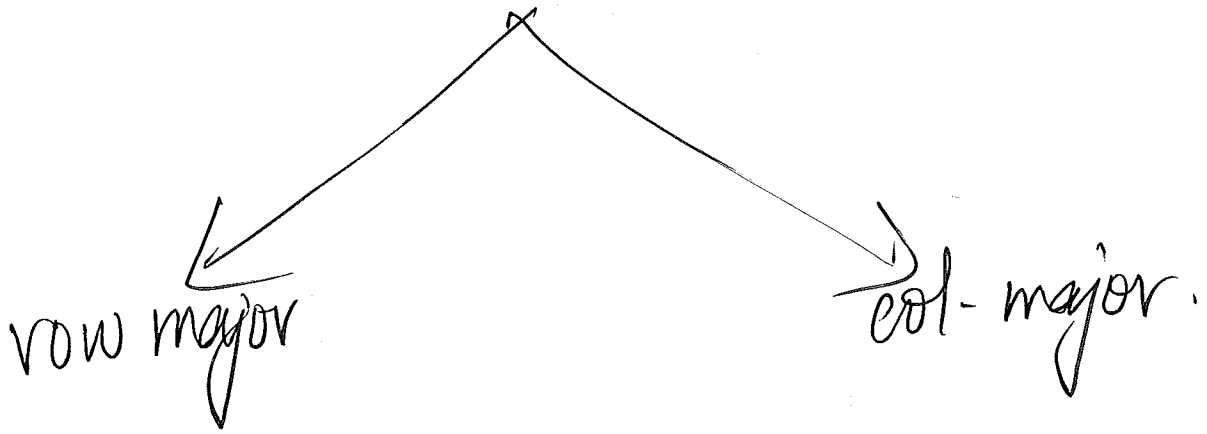
stride-3 reference pattern.

$$k \propto \frac{1}{\text{locality}}$$

2d array

a:

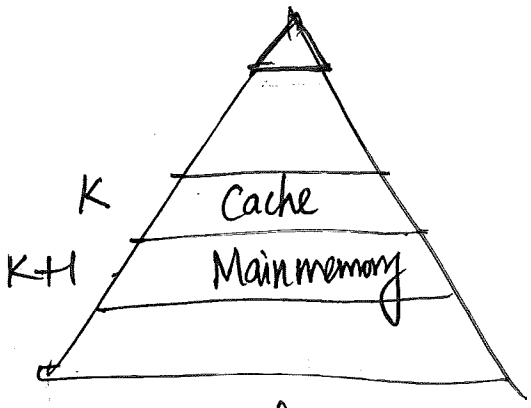
a_{00}	a_{01}	a_{02}
a_{10}	a_{11}	a_{12}

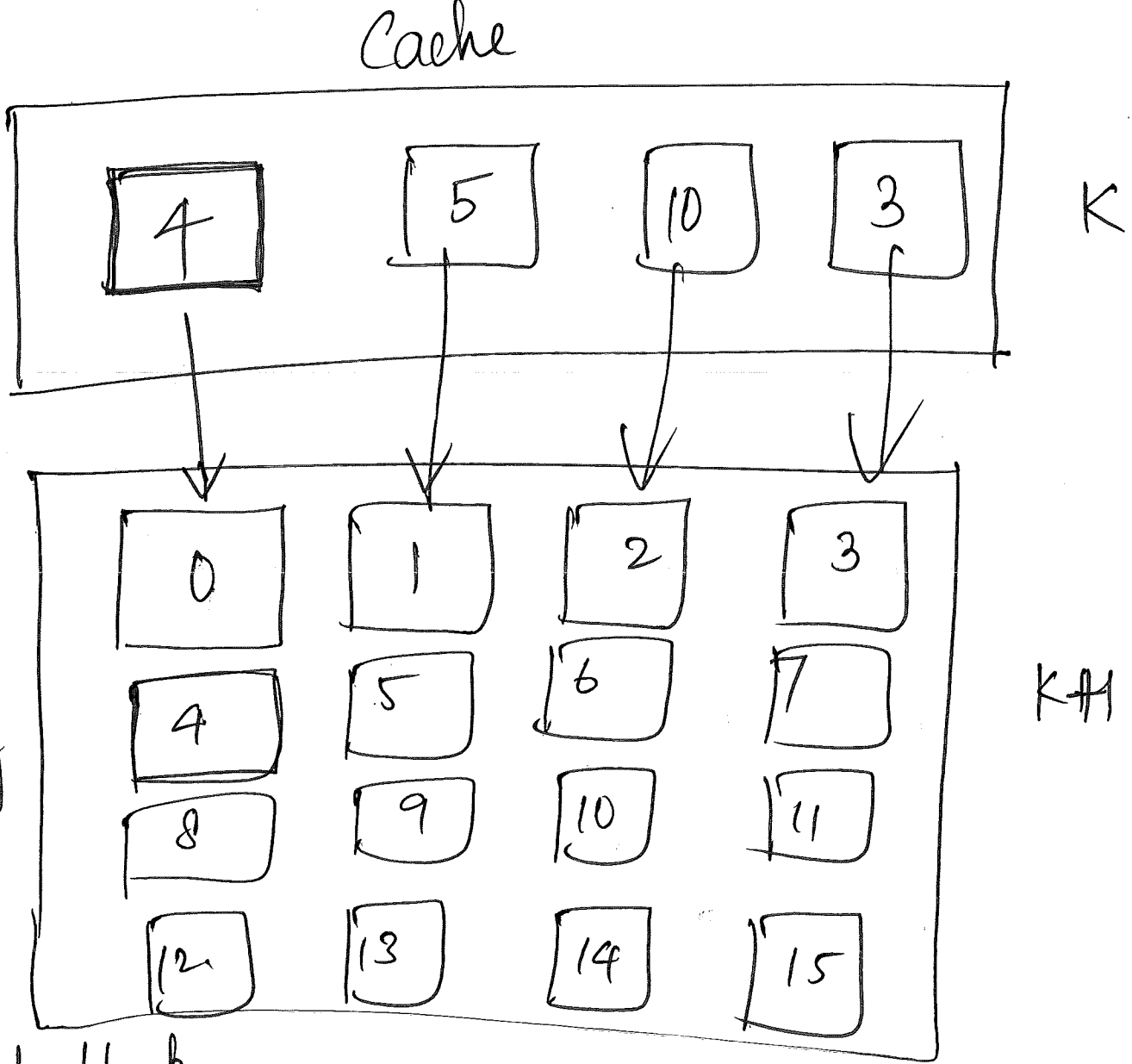


a:

a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}

Caching

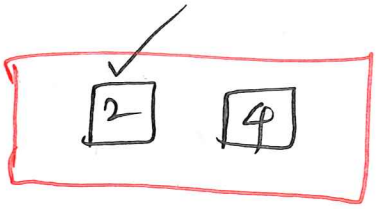




1. Read block 4.
2. Read block 5
   " 10
   " 3

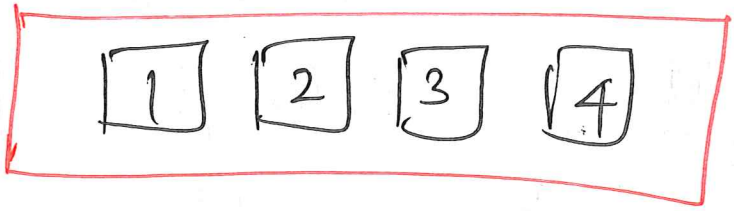
Terminology

1. Cache hit



K (cache)

read blk 2 : hit

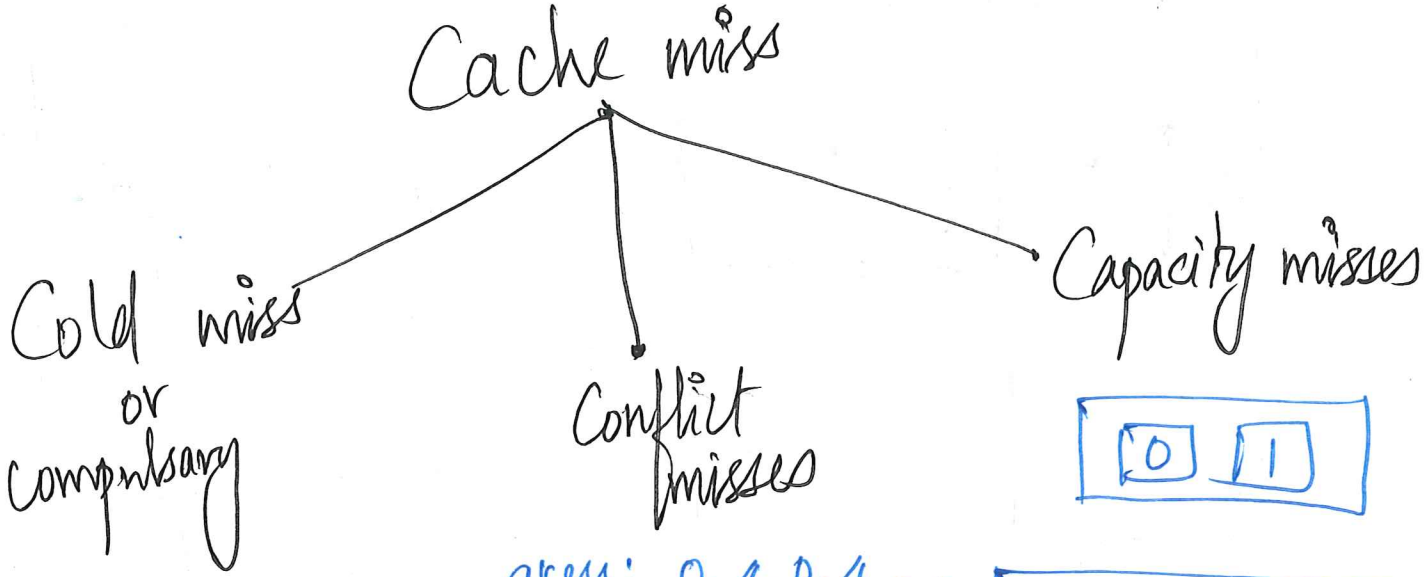


K+1

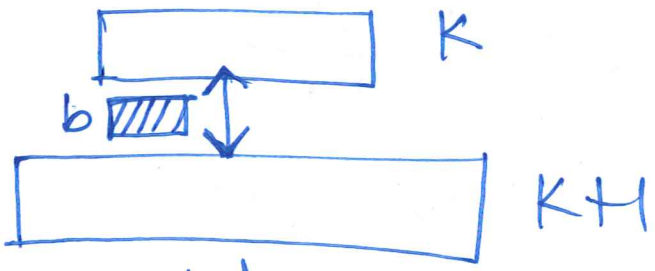
—

2. Cache miss

read blk 3 : "cache miss" at the memory K.



Cache organization.

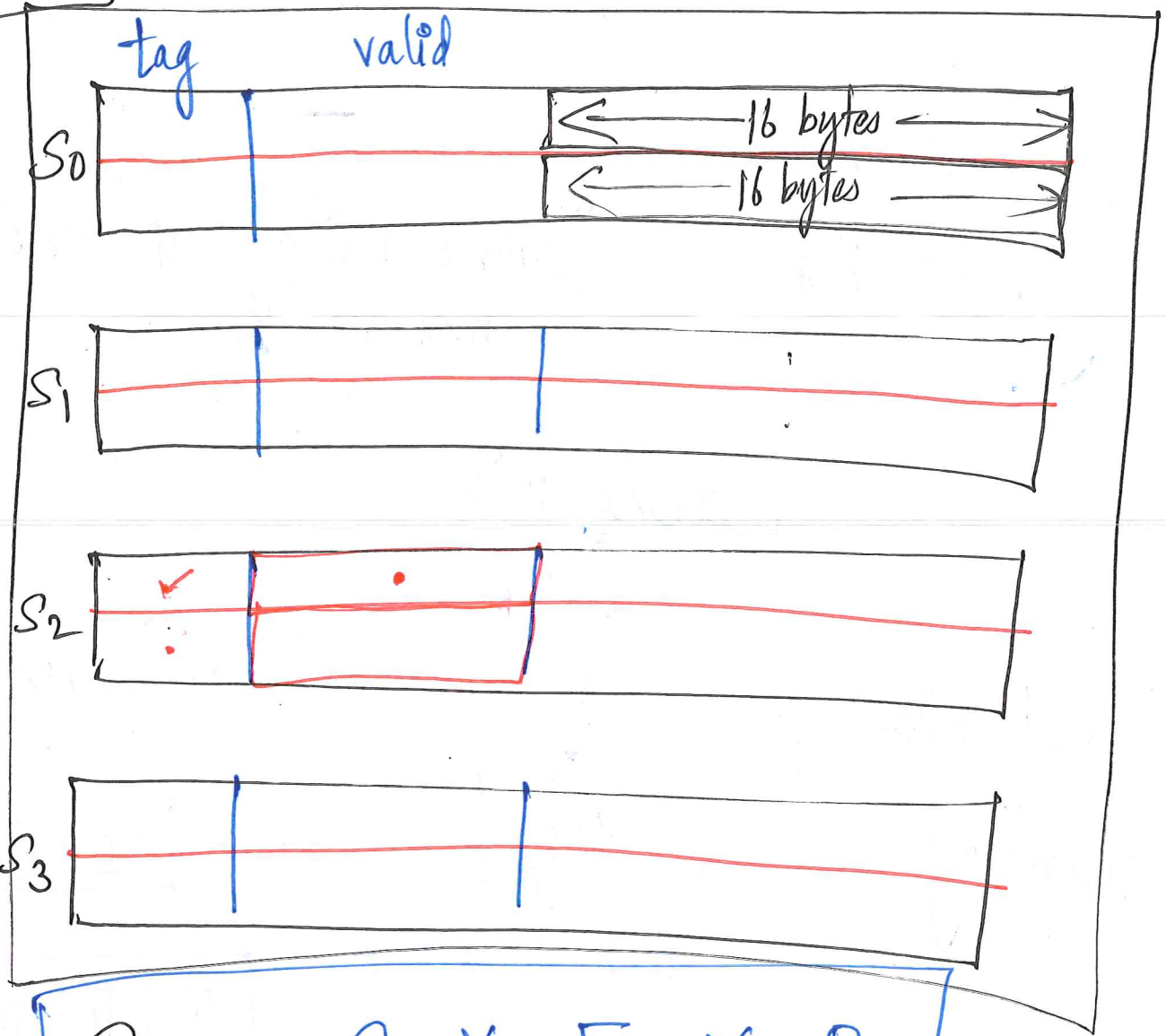


32-bit machine

1 block = 4 words.

= 4 X 4 bytes = 16 bytes

1. Sets S = 4
2. Lines E = 2
3. Block size B = 16 bytes



$$C = S \times E \times B$$

C → size of a cache

Address

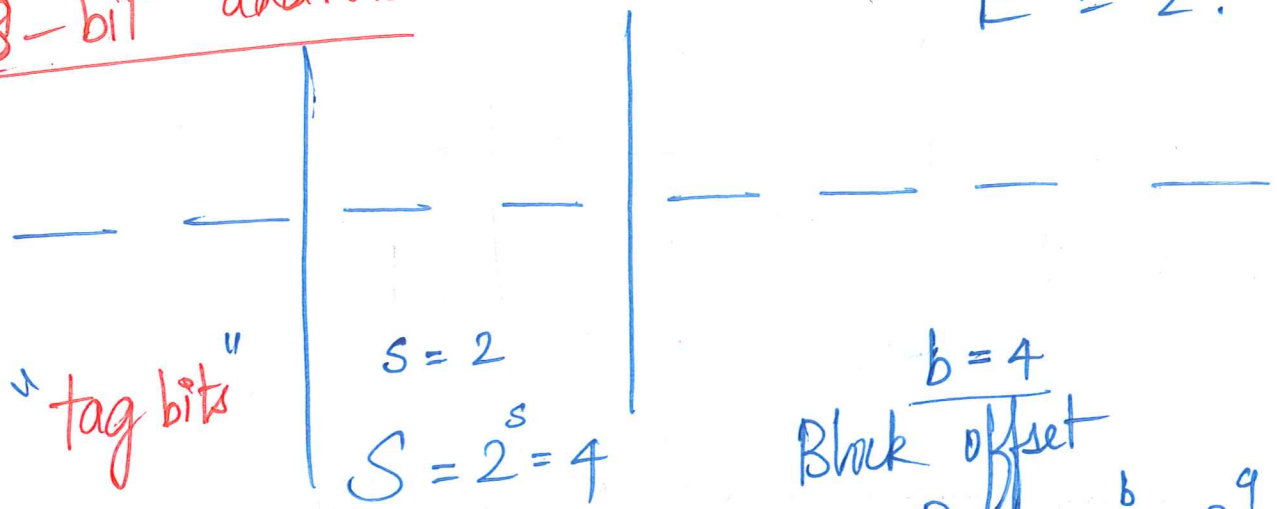
32-bit

8-bit address

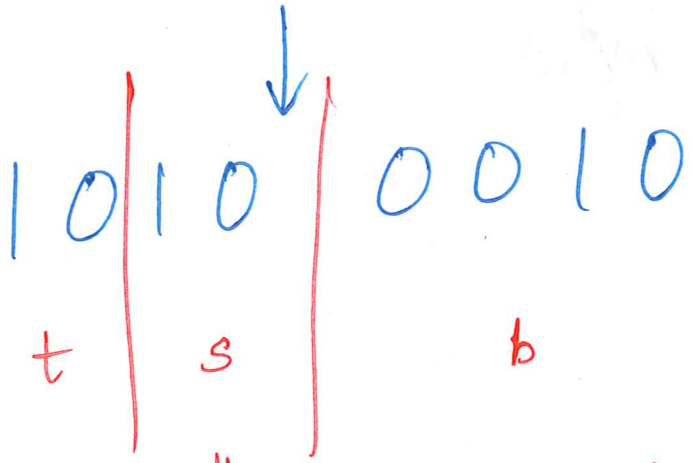
Block size = 16 bytes

sets = 4

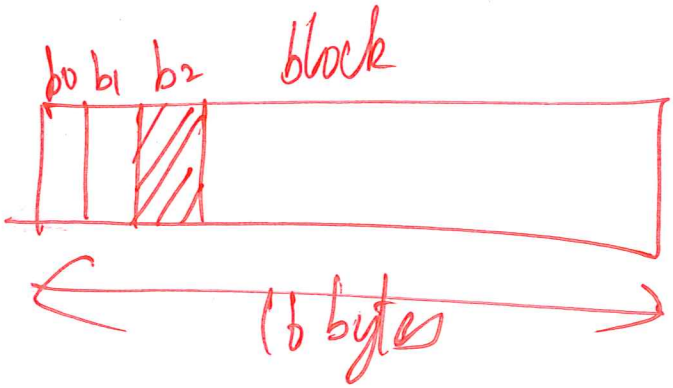
E = 2.



Read 0xA2

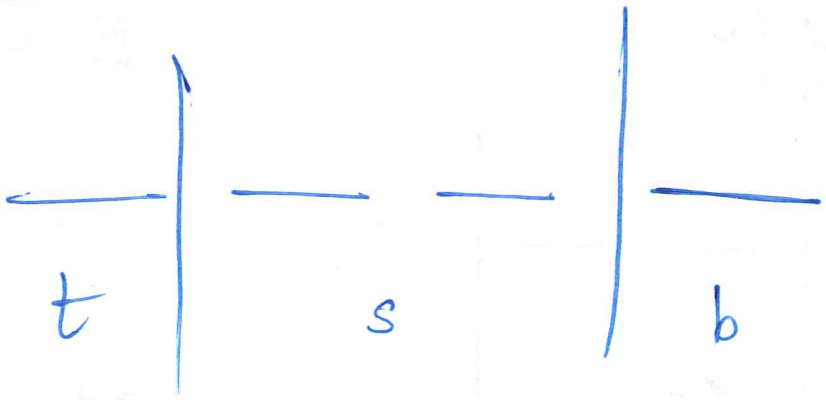


set # = 2



4-bit address

Assume: E = 2



$$C = S \times E \times B$$

$$= 4 \times 2 \times 2$$

$$= 16 \text{ bytes}$$

Caches

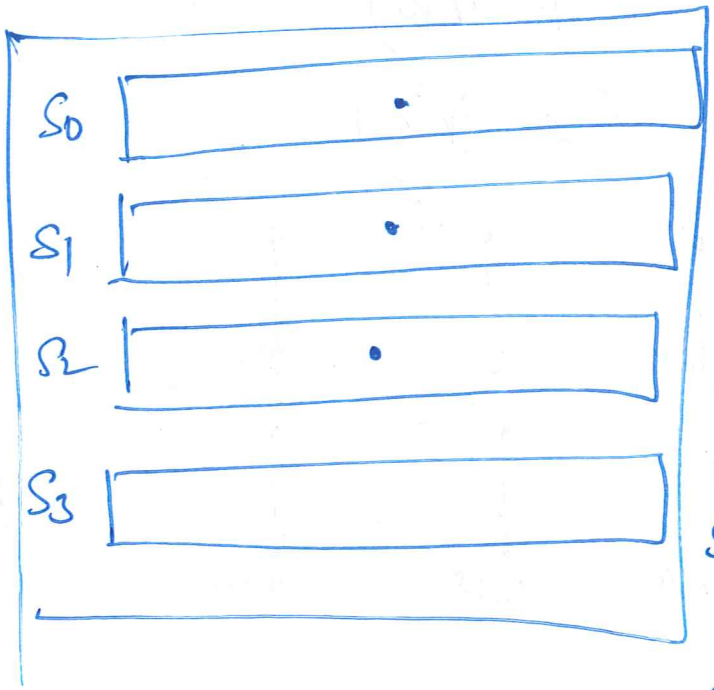
- Direct Mapped
- Set associative
- Fully associative cache

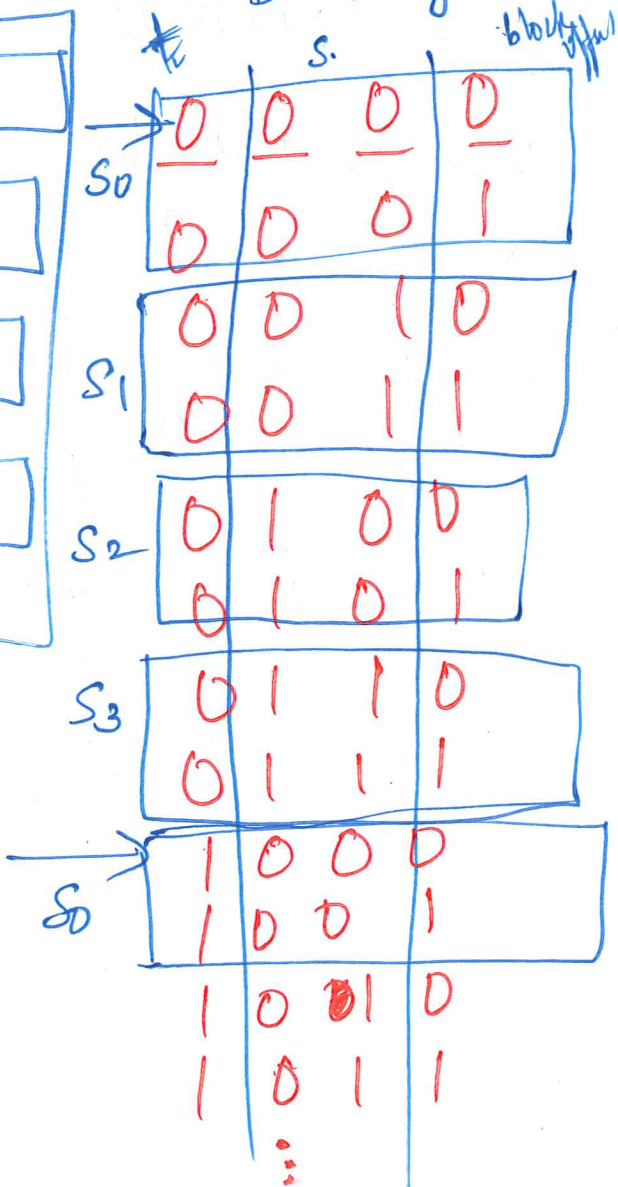
Direct Mapped

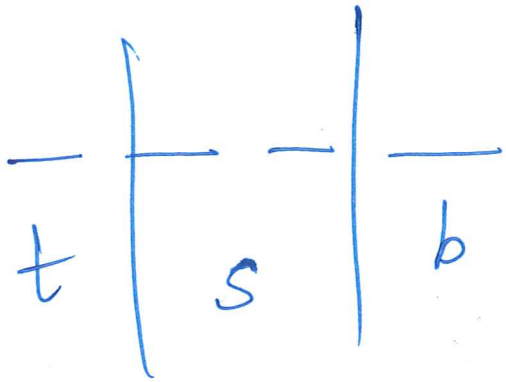
E = 1 (fixed)

S = 4

B = 2^2 bytes



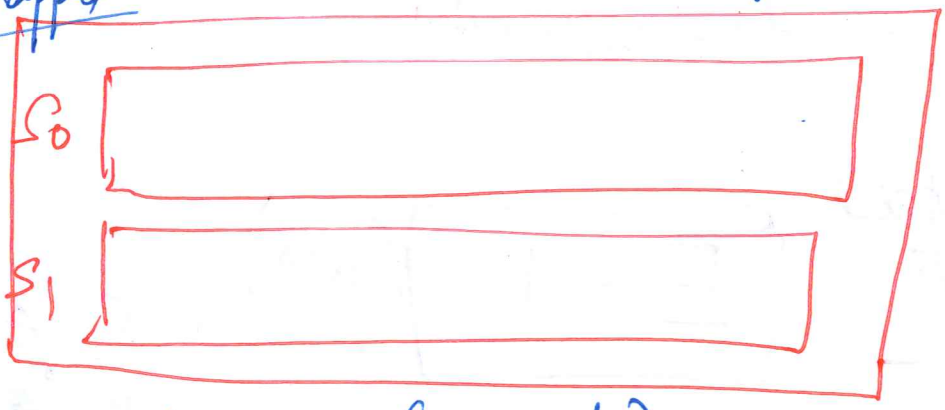




(16)

```
for (i=0; i < 8; i++)
    sum += x[i] * y[i];
```

Direct mapped

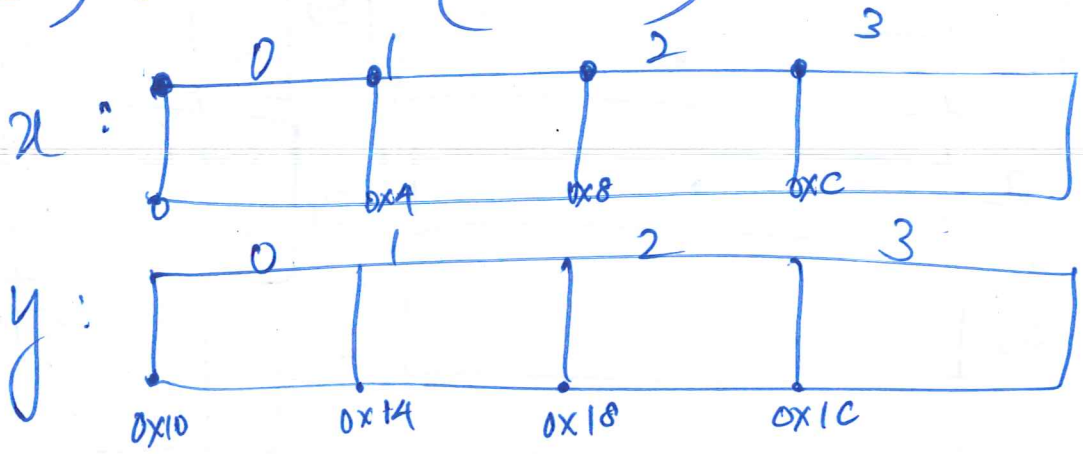


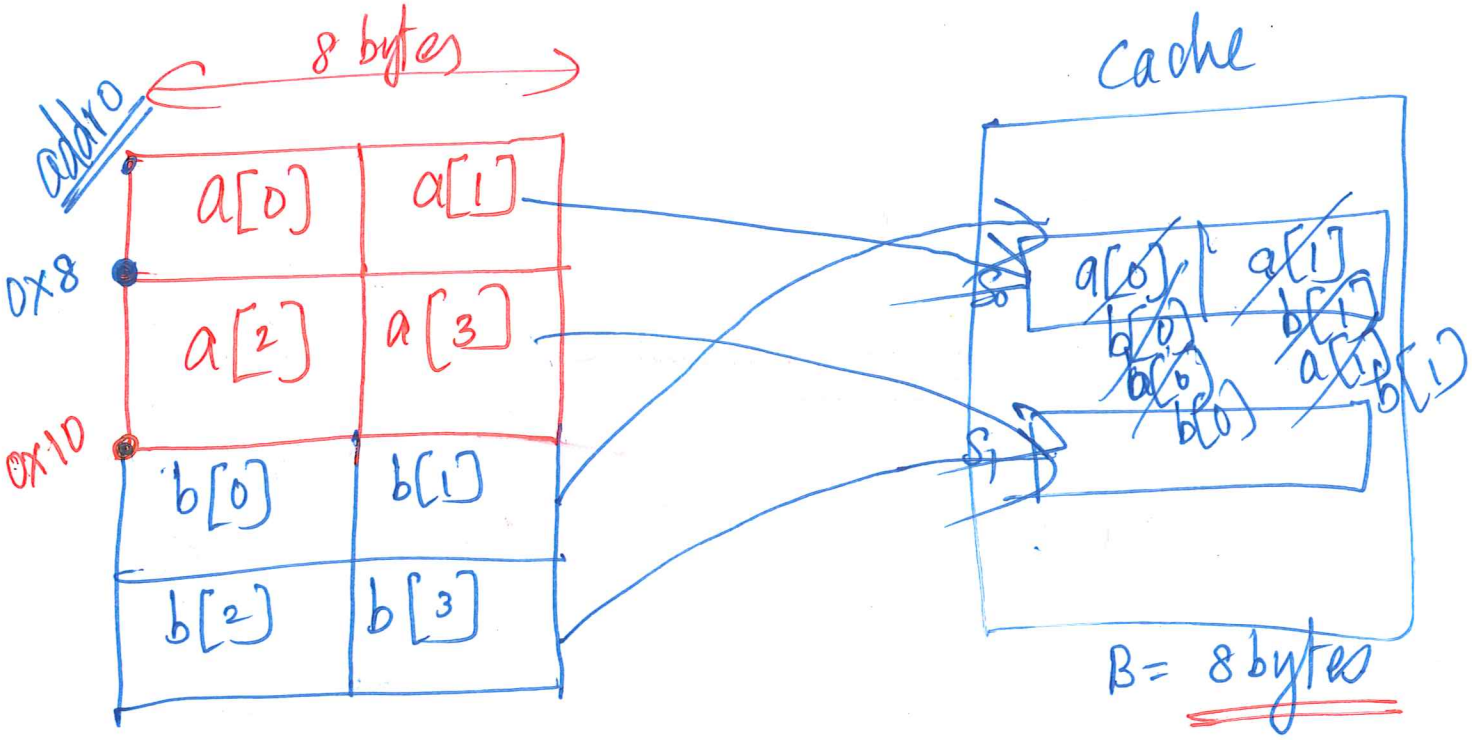
B = 8 bytes

C = 16 bytes

x → addr 0 (4 ints)

y → addr 16 (4 ints)





addr of a[2] = 0x08

= 0000 | 1 | 000

s | b

addr of b[0] = 0x10

= 0001 | 0 | 000

s | b

a[0] → cold miss

b[0] → conflict miss

CS 354: Lecture 12 - Cache Memories

April 11th 2018

1. Assume the following:
   a. The memory is **byte addressable**.
   b. Memory accesses are to **1-byte words** (not to 4-byte words).
   c. Addresses are **13 bits** wide.
   d. The cache is a **direct-mapped cache** (E = 1), with a 4-byte block size (B = 4) and eight sets (S = 8).

The contents of the cache (in **hexadecimal** notation) is shown below..

Set index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	09	1	86	30	3F	10
1	45	1	60	4F	E0	23
2	EB	0	-	-	-	-
3	06	0	-	-	-	-
4	C7	1	06	78	07	C5
5	71	1	0B	DE	18	4B
6	91	1	A0	B7	26	2D
7	46	0	-	-	-	-

The following figure shows the format of an address (one bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

CO - The cache block offset
CI - The cache set index
CT - The cache tag

$t = m - (s + b)$

12	11	10	9	8	7	6	5	4	3	2	1	0
								CI	CI	CI	CO	CO

t s b

CT

m

Suppose a program running on the machine references the 1-byte word at address **0x0E34**. Indicate the cache entry accessed and the cache byte value returned in hex. Indicate whether a cache miss occurs. If there is a cache miss, enter "–" for "Cache byte returned."

tag → set index | Block offset

Address Format

12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	0	0	0	1	1	0	1	0	0

(5)

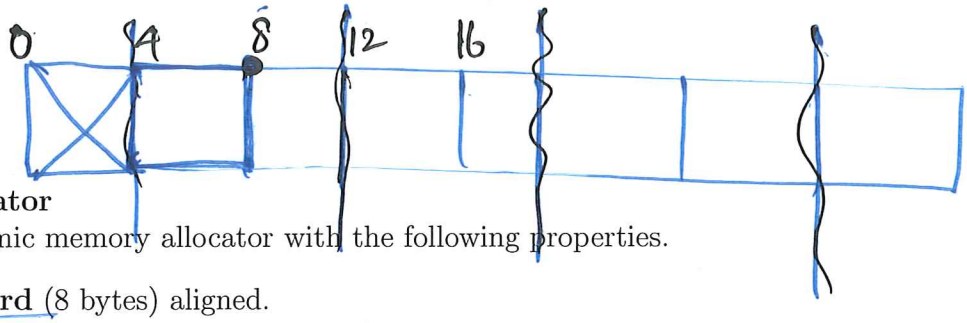
Memory Reference

Parameter	**Value**
Cache block offset (CO)	0x 00 (OR) 0x0
Cache set index (CI)	0x 05 (OR) 0x5
Cache tag (CT)	0x 71
Cache hit? (Y / N)	Y
Cache byte returned	0x 0B

tag = 0 1 1 1 | 0 0 0 1

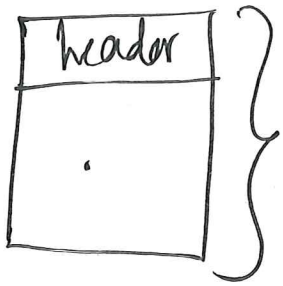
0x 7 1

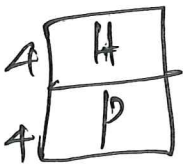
1 0 1



1. **Memory Allocator**
Consider a dynamic memory allocator with the following properties.

- **Double word** (8 bytes) aligned.
- **Implicit** free list is used for free block organization.
- All blocks have a header of size 4 bytes.
- The size of a block (including the header) is stored in the header.
- **bit 0** (least significant bit) in the header indicates the use of the current block: 1 for allocated, 0 for free.





Answer the following questions regarding this allocator:

(a) Minimum block size = 8

(b) Maximum block size = $(2^{32}-1)-7$

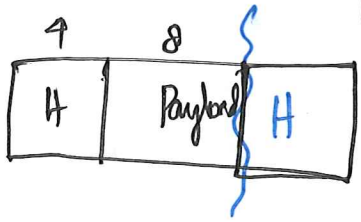
32 bits $2^{32}-1$

(c) For the following memory allocation, what is the size of the payload and padding that will be used in the allocated block? Block splitting might happen based on the size of the request.
You should assume the following: A free block of size **32 bytes** is chosen by the allocator to satisfy the below malloc request. The header of this block is at the memory location **0x8090A0B4**.

```
char *p = malloc(8);
```

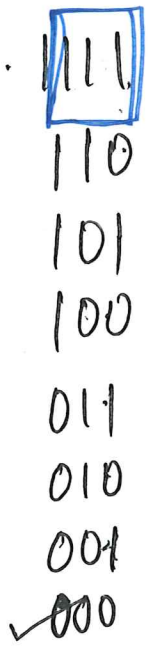
Payload = 8 bytes

Padding = 4 bytes

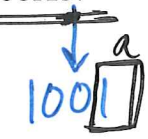


Memory address stored in the pointer p (in hexadecimal):

0x 8090A0B8



(d) The contents of the header of a block in the allocator is **0x000000A9**.



i. Is the block allocated or free?
allocated.

ii. What is the size of the block (in decimal)?
0xA8 = 168

iii. Is the contents of the header of this block valid with respect to this allocator? Remember, for a block to be valid with respect to an allocator, its size should satisfy the alignment requirement of the allocator. Yes OR No

$0xA8 = (10\times16)+(8\times1)_1 = 168$