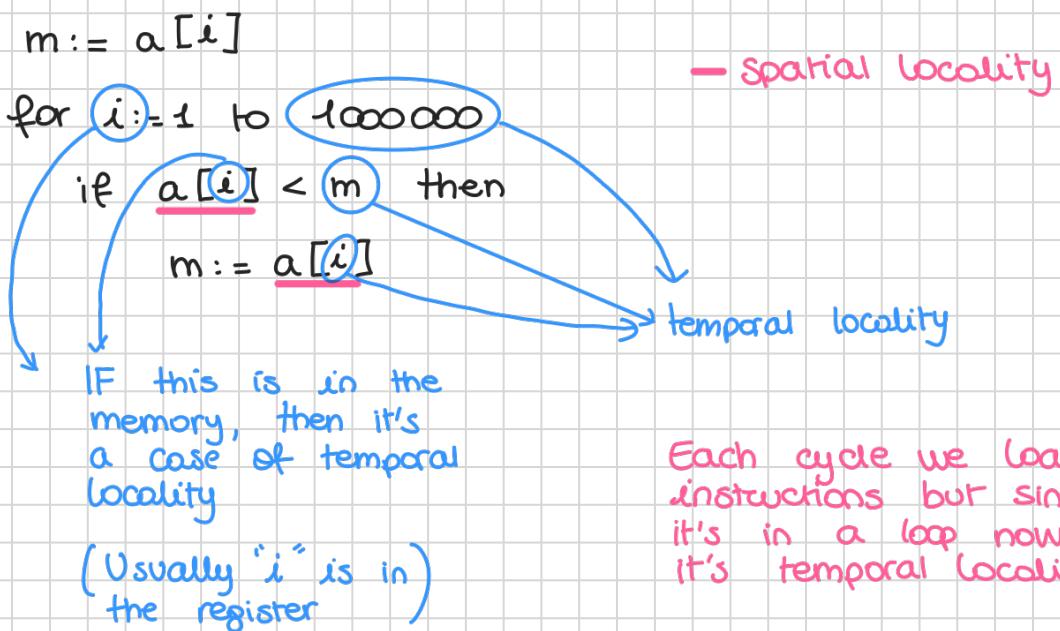


MEMORY

- Spatial Locality
- Temporal =

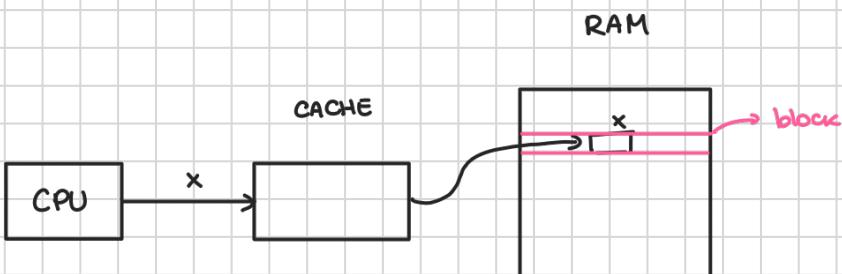
nearby access

Programs have both of these



Linked lists, binary trees and hash tables don't have locality properties.

- We exploit locality properties to make memory instr. faster.
- We can build small memories → expensive but very fast

HIERARCHY OF MEMORY

READ

CPU asks the content of an address "x"

- ① the cache may already have the content (HIT)
- ② cache asks the RAM for the content, saves it in cache and gives to CPU (MISS)

GOAL : to reduce "MISS" rates

when cache doesn't have the content asked from CPU

We are exploiting temporal locality properties.

- ②.1 when we take the content from RAM, the cache takes a block (chunk) of data

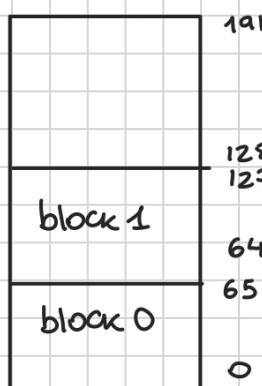
WRITE

CPU wants to write into the memory at the address "x".

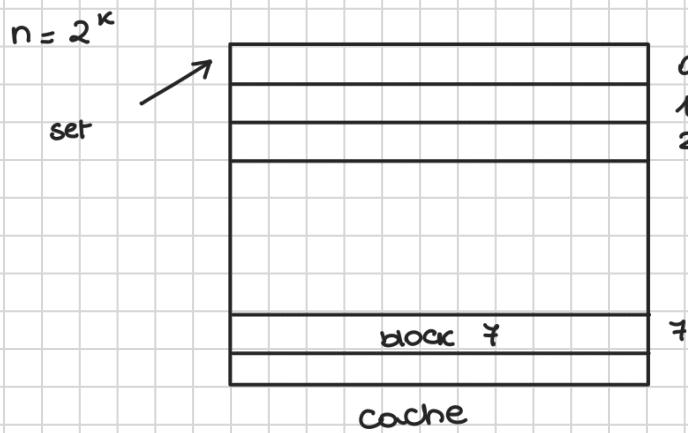
- ① write in cache only
- ② = in memory too

You would want to do the 2nd one, but typically we write only on Cache. If its a lot to write and Cache needs to save blocks, you write on memory (DEALLOCATE BLOCK)

↳ When you remove a block from the cache



Let's say block is 64 bytes
only fixed blocks



Let's say we want to load block 7 in cache

- we load it into set 7

STATIC ALLOCATION

- We load block only if CPU requests it (this happens during MISS situation) and we load it in its according set

$$n = 2^k \text{ # sets}$$

block made of 64 bytes

x (32 bytes)



4 bit
(set#)

6

offset within the block

HOW CAN I COMPUTE THE NUMBER OF BLOCK STARTING FROM "X"?

$$\text{block\#} = \frac{x}{64} \quad (\text{easier in binary})$$

block bytes

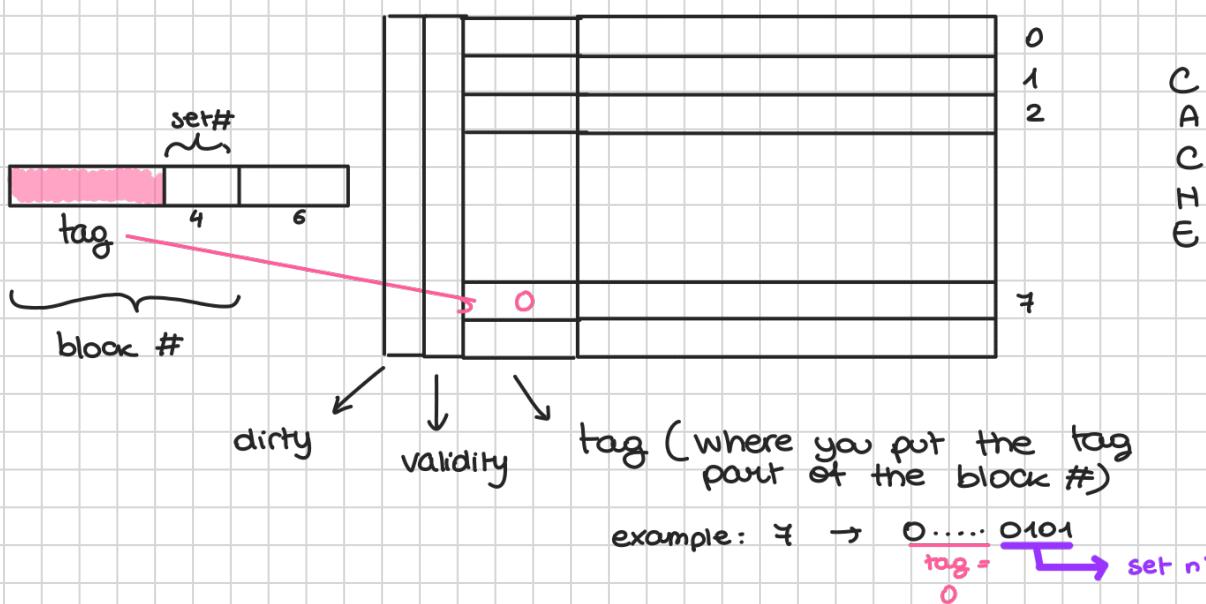
HOW TO KNOW WHICH IS THE SET# OF BLOCK#?

- Let's say we have 16 sets.

$$\text{set\#} = \text{block\# \% } n^2 \text{ sets}$$

HOW TO BE SURE SET 7 CONTAINS BLOCK 7 ?

(you can't put the whole block number)



- We need one bit to determine if the block content is meaningful or if it's empty. This is the "validity" part of the cache.
- We need another bit to know whether the content written in cache has been written inside the memory as well
(bit = 0 if RAM has been written as well)

EXERCISE TIME GUYS

	0
	1
	2
	3

cache → 4 sets
block → 32 bytes

M = Miss

M H M M M H H M H
36, 48, 0, 100, 164, 172, 96, 92, 16

Suppose each one of those are addresses the CPU wants to access

- 36. The first one is a Miss (M) 'cause cache empty

0	
1	1
2	
3	

$$\text{Block\#} = \frac{36}{32} = 1 \text{ so block 1}$$

↑ block bytes

Since it's a M, we load "block 1" from memory to a set:

$$\text{Set\#} = 1 \% 4 = 1 \text{ so set 1}$$

↑ n = sets

- 48. This is a HIT (H)

$$\text{Block\#} = \frac{48}{32} = 1 \text{ so block 1}$$

(inside cache)

0	
1	1
2	
3	

$$\text{Set\#} = 1 \% 4 = 1$$

- 0. This is MISS (M)

$$\text{block\#} = \frac{0}{32} = 0 \quad \text{no block 0 in cache}$$

0	0
1	1
2	
3	

Since it's a M, we load "block 0" from memory to a set:

$$\text{Set\#} = 0 \% 4 = 0 \text{ so set 0}$$

- 100. This is MISS (M)

$$\text{block\#} = \frac{100}{32} = 3 \quad \text{no block 3 in cache}$$

0	0
1	1
2	
3	3

Since it's a M, we load "block 3" from memory to a set:

$$\text{Set\#} = 3 \% 4 = 3 \text{ so set 3}$$

- 164. This is MISS (M)

$$\text{block\#} = \frac{164}{32} = 5 \quad \text{no block 5 in cache}$$

0	0
1	5
2	1
3	3

Since it's a M, we load "block 5" from memory to a set:

$$\text{Set\#} = 5 \% 4 = 1 \text{ so set 1}$$

We have to overwrite set 1 and put block 5 now, no block 1

- 172. This is a HIT (H)

$$\text{Block\#} = \frac{172}{32} = 5 \text{ so block 5 (inside cache)}$$

0	0
1	5
2	
3	3

$$\text{Set\#} = 5 \% 4 = 1$$

- 96. This is a HIT (H)

$$\text{Block\#} = \frac{96}{32} = 3 \text{ so block 3 (inside cache)}$$

0	0
1	5
2	
3	3

$$\text{Set\#} = 3 \% 4 = 3$$

- q2. This is MISS (M)

$$\text{block \#} = \frac{92}{32} = 2 \quad \text{no block 2 in cache}$$

0	0
1	5
2	2
3	3

Since it's a M, we load "block 2" from memory to a set:

Set# = 2 % 4 = 2 so set 2

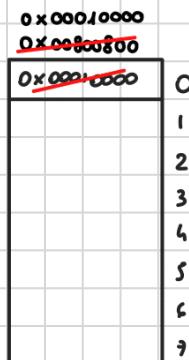
- 16. This is a HIT(H)

$$\text{BLOCK\#} = \frac{16}{32} = 0 \quad \text{so block } 0 \\ (\text{inside cache})$$

0	0
1	5
2	2
3	3

Set # = 0% 4 = 0

Cache → 8 sets
Block → 64 bytes



• data

x: . word 5, 6, 2, 0, -2, 5, ...
n: . word 1024

16 instr. in block 0

.text

→

miss li t3, 0 block 0 $0x00400000 : 2^{(64)}$ = $0x0001\underline{0000} : 8$ = rem 0
block n.0 set n.
last 3 bits

hit lui t0. 0x10010 block 0 0x00400004 : 2^{64} = 0x00010000 //

hit lw t1, 0x01000 (to) b0 0x00400008 : 2⁶(64) = 0x00010000 //

b/m

acid: $\text{H}_2\text{O} + \text{Zn} \rightarrow \text{Zn(OH)}_2$

The `lw` is **hit** because it's in the same block.

To load the data, I have to go to another

address that's part of another block, so it's

2 miss $0x10010000 : 2^{(64)} = 0x00800800 : 8 = \text{rem } 0$ set n.
block n

this block replaces the previous one stored in set 0.

miss add t3, t2, t2 It's a miss because the previous `lw` instr. replaced the block stored into the set 0 that is needed by this instruction.
The block needed by this instr. is the no, so the first one that was stored into the set 0 of the cache memory.

hit add; t0, t0, 4

hit add; t1, t1, -1

Sum of the array.

hit bne t1, zero, ciclo

- loads 1st instruction
- looks for block number