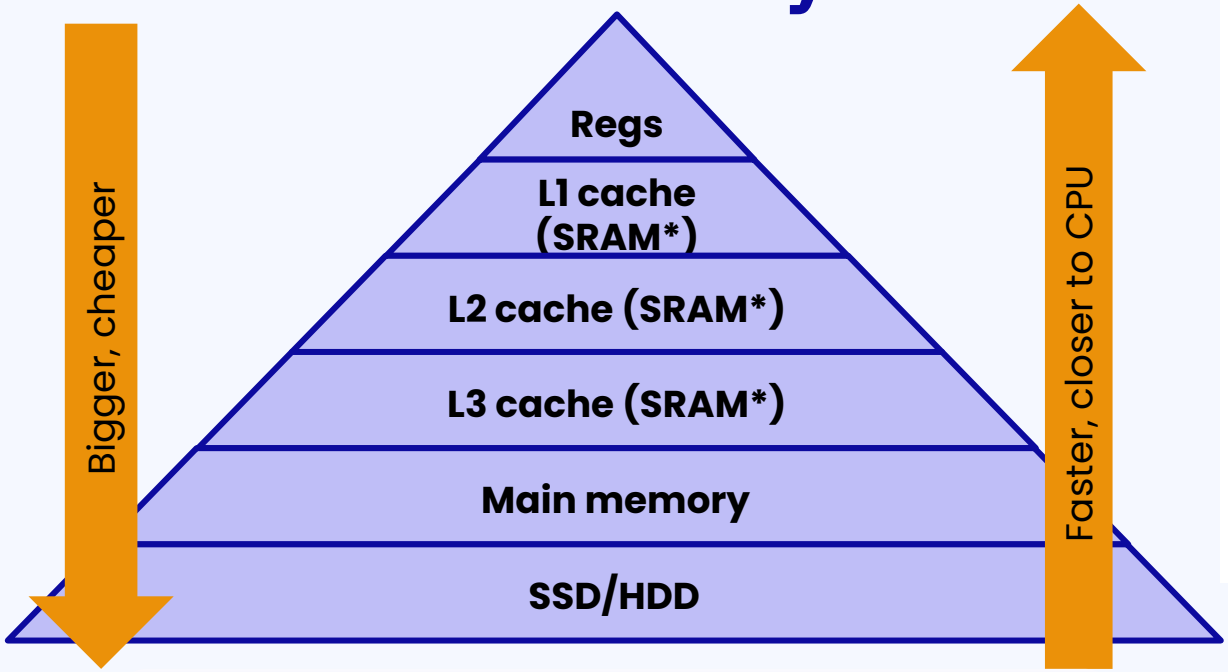
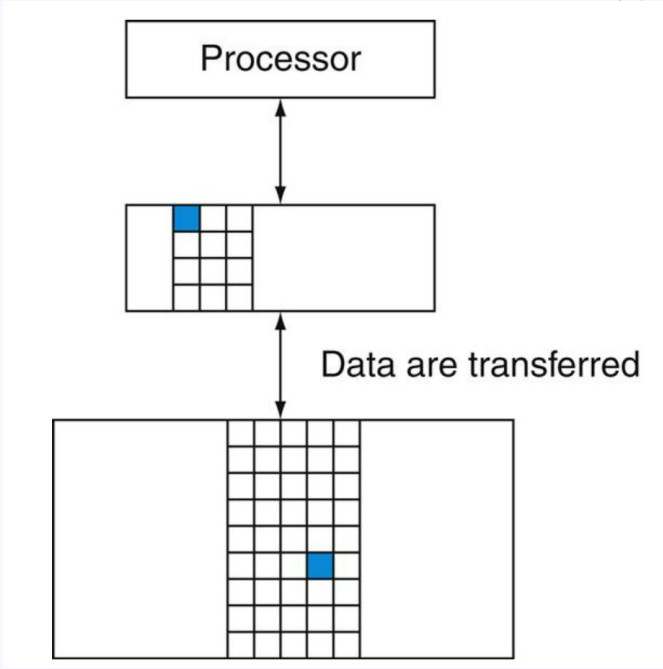
Cache controllers

Mem. hierarchy review





P&H 5.2

Each level stores (**caches**) a subset of the one below it, for faster access of specific data

Cache design questions

- How do we decide what goes where in a cache?
- How do we decide what data to evict when our cache gets full?
- How do we maintain consistent copies of data across the hierarchy?
- What does the cache controller need to keep track of?

Later:

- How do we build an efficient* memory hierarchy?
- How do we manage shared caches?
- Do caches expose security vulnerabilities?

6Block size: 1 word (4 bytes)

Cache size: 1 KB (256/2^8 blocks)

Our example cache

Our program

Assume register a0 holds the value 0x20000004

```
lb t0, 1(a0)
lw t1, 4(a0)
lhu t2, 2(a0)
lw t0, 1024(a0)
addi t1, t1, 1
sw t1, 4(a0)
sh t1, 8(a0)
```

Terminology

Block: minimum unit of information that can be present/not present in a cache

Index: how we refer to locations in the cache; each block has a unique index and each (block-aligned) memory address maps to a unique index (using the lower bits of the block address)

Valid bit: indicates whether data has been pulled in to that block of the cache

Tag: the upper bits of an address, used to uniquely identify which data is in the cache

0x20...04 in bin: 0b0010...0100
1(a0) = 0x20...05 = 0b0010...0101
4(a0) = 0x20...08 = 0b0010...1000
1024(a0) = 0x20...0404 = 0b0010...0100 0000 0100

32-bit addresses, 256 (2^8) blocks, 4 byte (2^2) block size:

address: [32:10 | 9:2 | 1:0]

tag | index | block offset

block address

```
0x20000004 in reg a0

lb t0, 1(a0)  ← index 1
lw t1, 4(a0)  ← index 2

lhu t2, 2(a0) ← index 1
                (cache hit b/c valid!)
lw t0, 1024(a0)
                ↖ index 1

addi t1, t1, 1
sw t1, 4(a0)  ← index 2 (hit)
sh t1, 8(a0)  ← index 3
```

index	data	valid	tag	dirty
0		0		0
1	~~~	1	~~0b0010...0~~ 0b0010...01	0
2	~~~	1	0b0010...0	1
3	~~~	1	0b0010...0	1
⋮		⋮		⋮
255		0		0

Write through vs write back

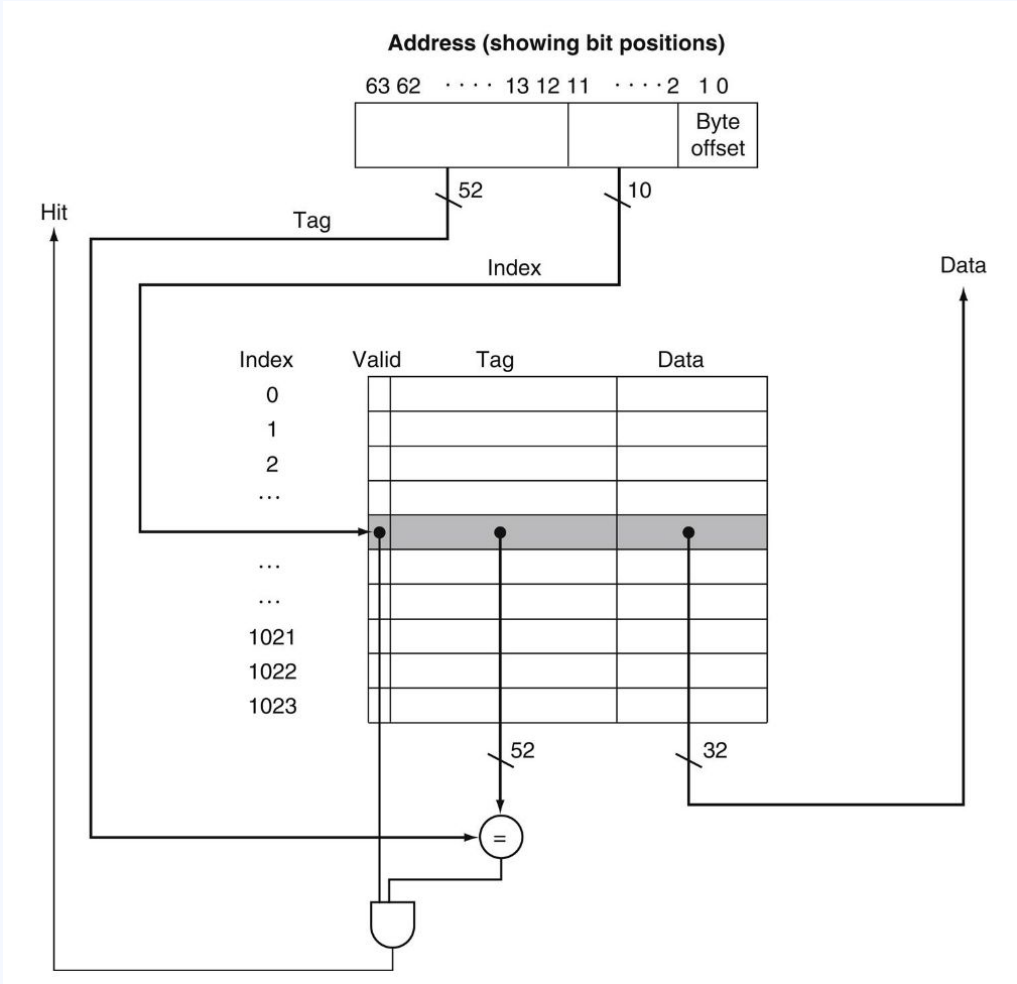
Write through: every time data is changed in cache, change is done to lower level in hierarchy

- Pro: We always have a consistent view of data across levels
- Con: Every write takes a long time

Write back: changes to lower level in hierarchy are only done when data is evicted from cache

- Pro: Fast
- Con: Consistency/needs a dirty bit

Cache controller



P&H fig. 5.10