chapter 5: The Memory

Memory Technology

Latency of access ∝ Capacitance × Resistance on wires.

- Volatile
  - static → SRam: latches/FlipFlop/Registers
    - Fastest
    - costly, lots of gates
  - dynamic → DRam: used for main memory
    - slower than SRam
    - needs "refreshing" because data is stored in capacitors, which drain out.

data lost on power off

Random access: not restricted to sequential

- non-volatile
  - disk: magnetic disk
    very slow
    very capacity, at low cost
  - SSD: Nand Tech
    "solid state" drive

main feature of memory Tech

larger "capacity" is slower to access

smaller "capacity" is faster to access

objective: How can we fake having large "capacity" with fast access.

solution: heuristic program tend access a small amount of memory at one point in time.

- Temporal locality: Items of data accessed recently tend to be accessed again soon. "reuse"
- spatial locality: items of data near by (address) tend to be accessed soon.

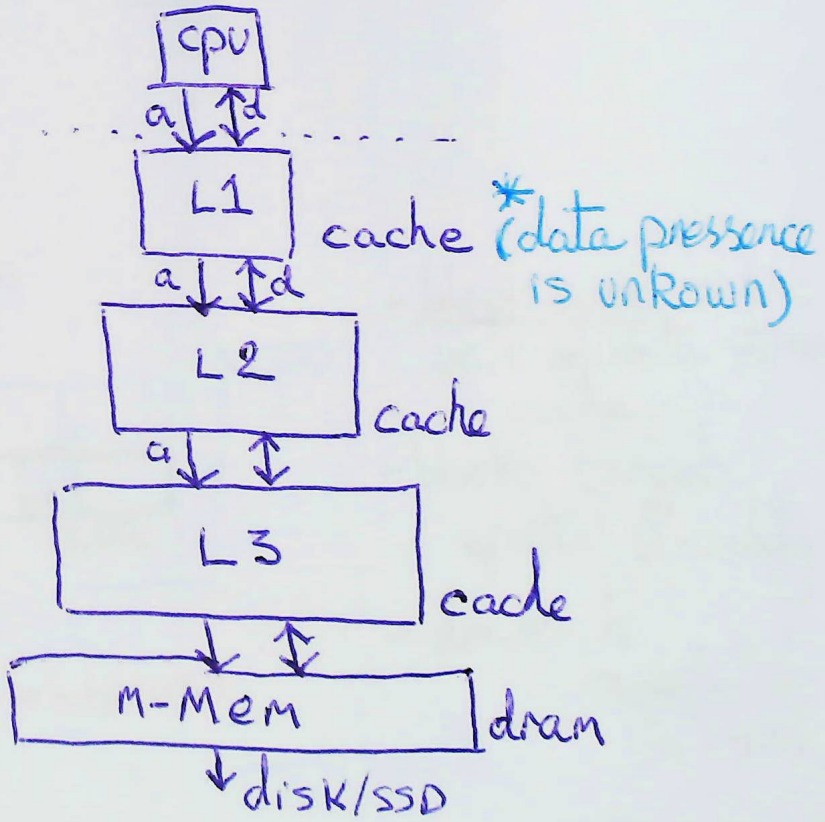
How to take advantage of data locality?

Memory Hierarchy

- Copy and keep recently accessed and nearby data in memory that is smaller - closer to the processor.

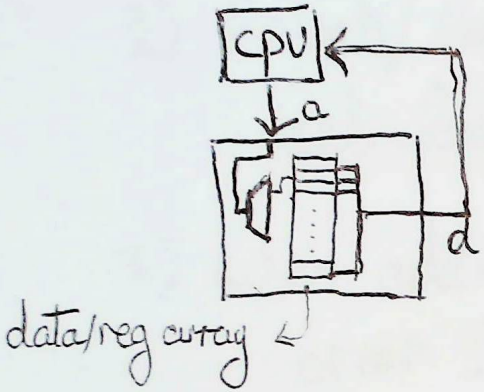
on average data appears to be accessed fast, but with a high total capacity to the memory.

Q: How do levels of memory tell if they have a certain addres?

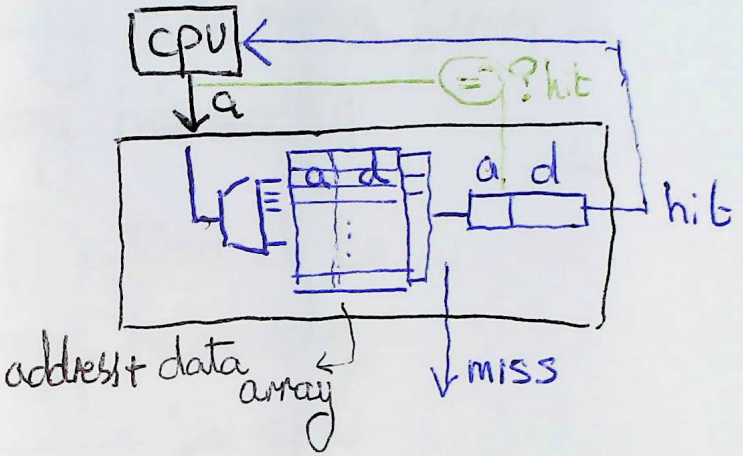


Inside memory design

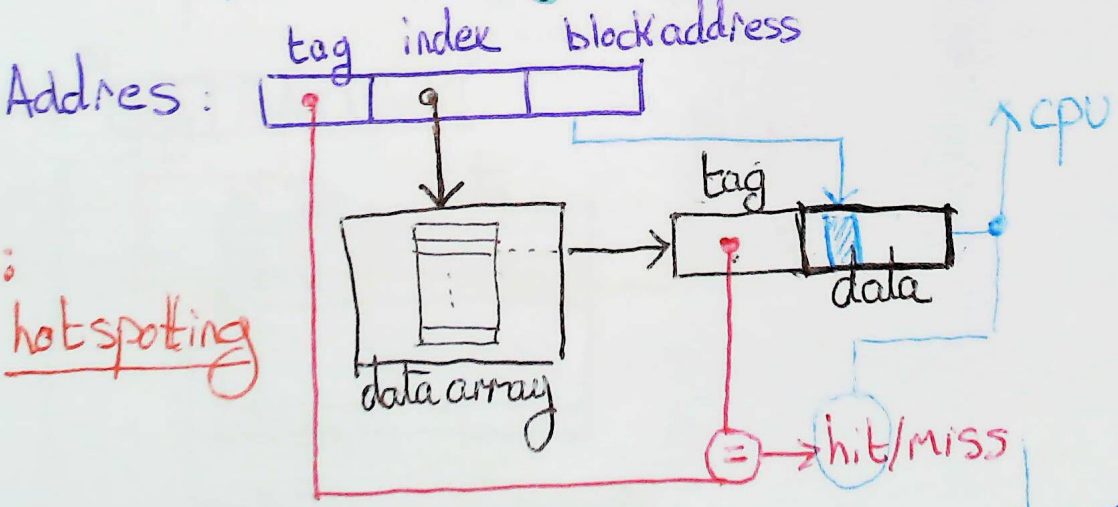
Simple Ram



Simple cache



Direct Map Caching



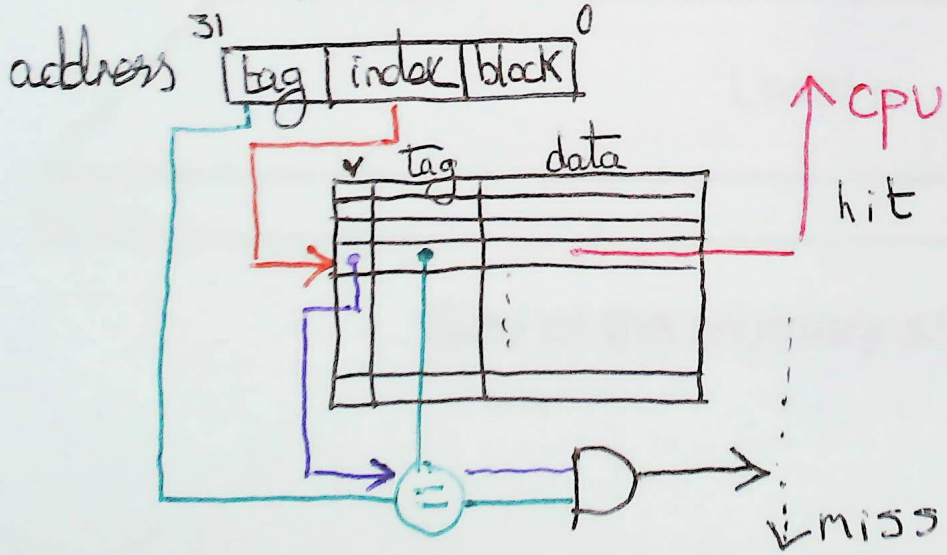
problem:
index hotspotting

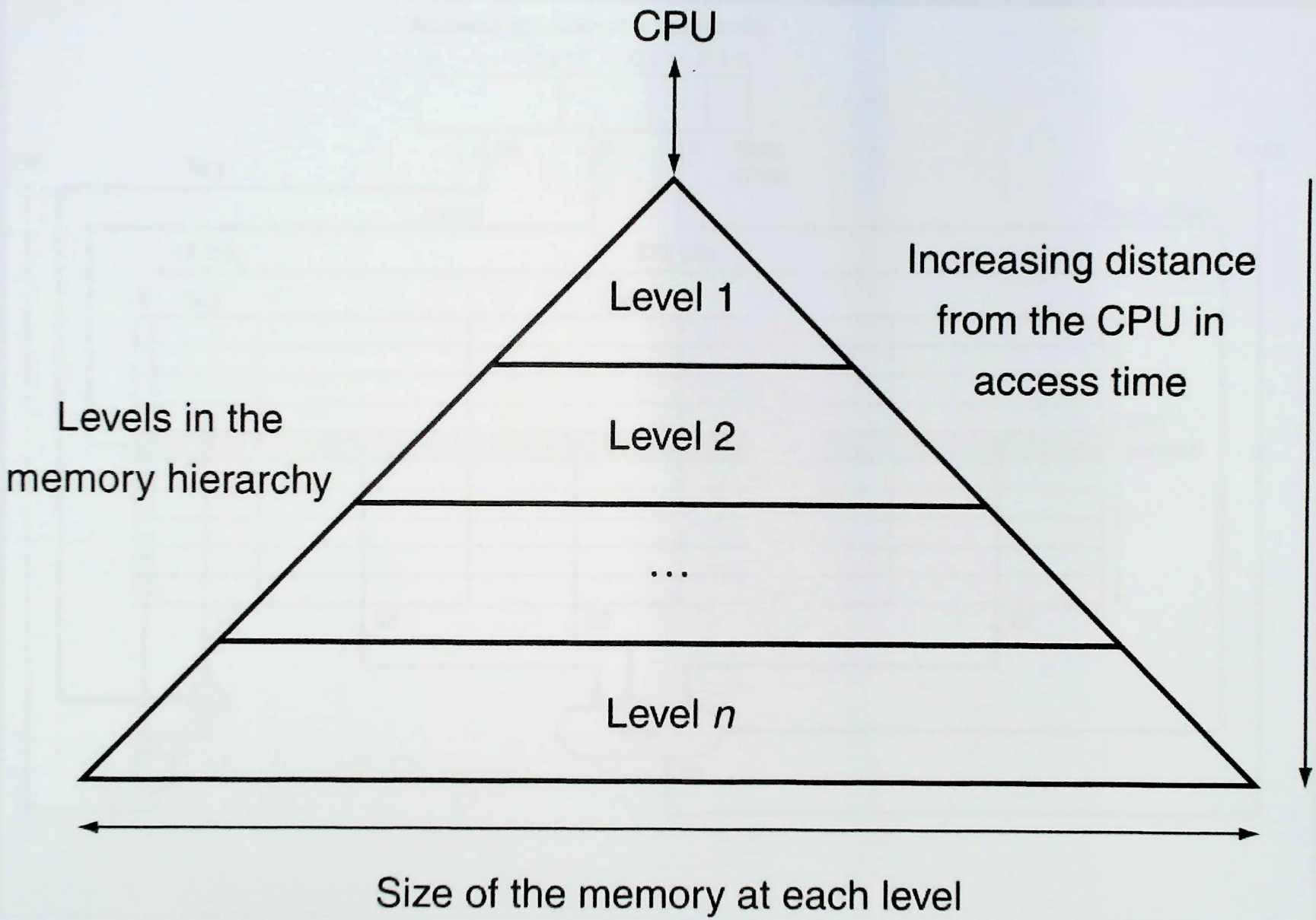
- block size is the unit of data stored in cache.
- blocks target spatial locality.
- downside: overfetch on miss

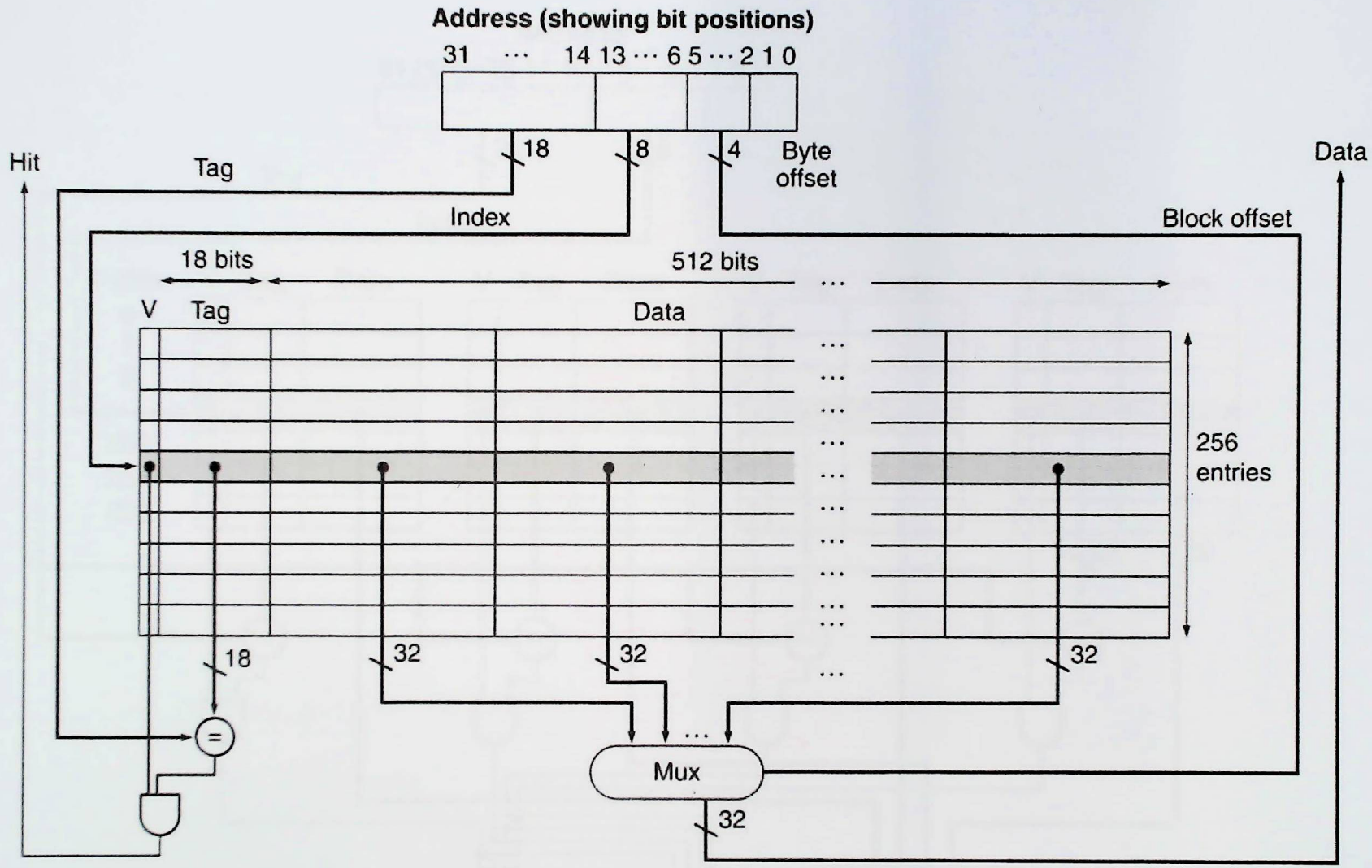
What happens on a cache miss?

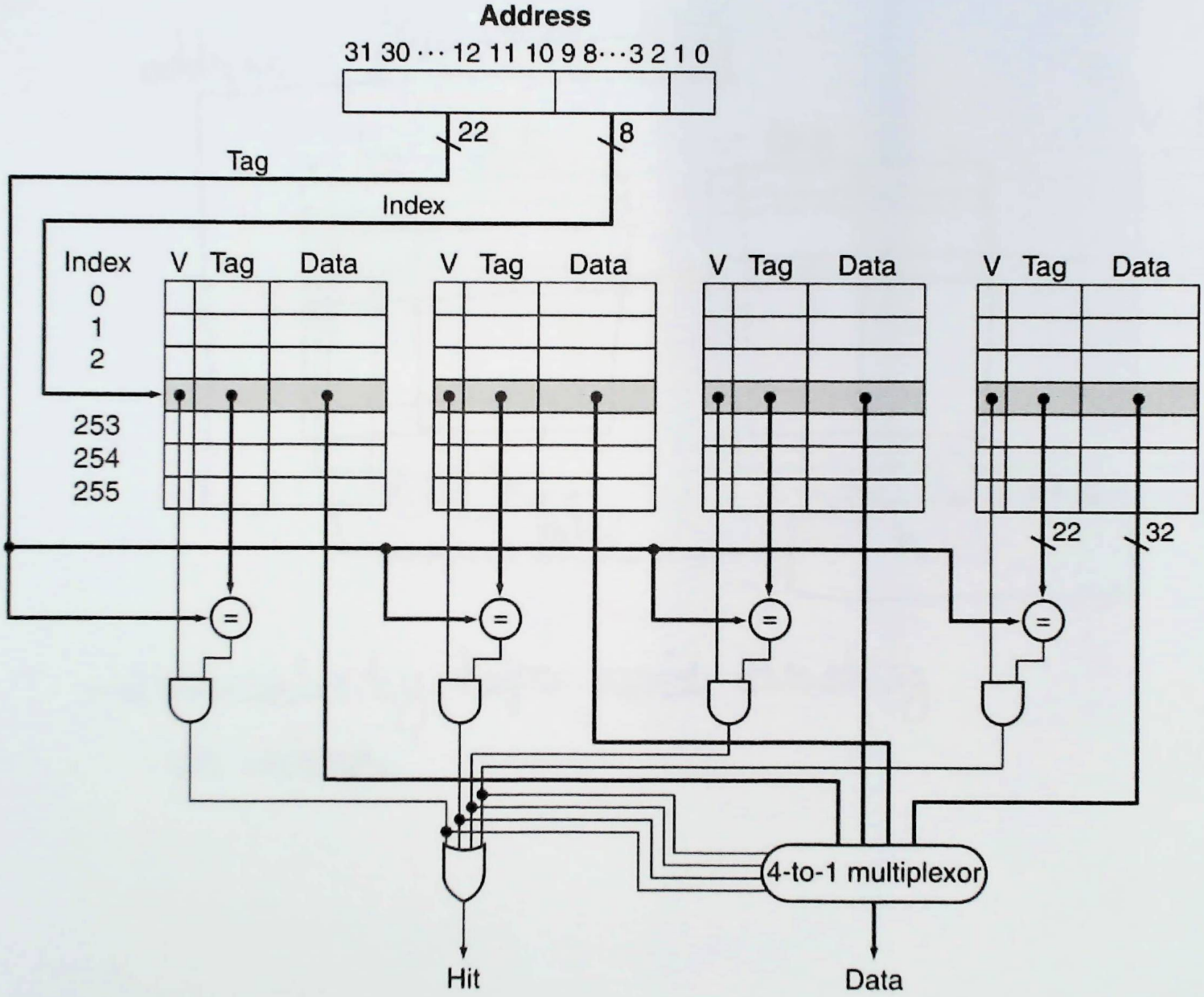
1) stall the cpu
2) fetch/lookup missing address from next level of memory hierarchy.
3) when address is found, return it to prior level of memory hierarchy.
   → to cpu
4) unstall the cpu

Direct Map cache (detailed)

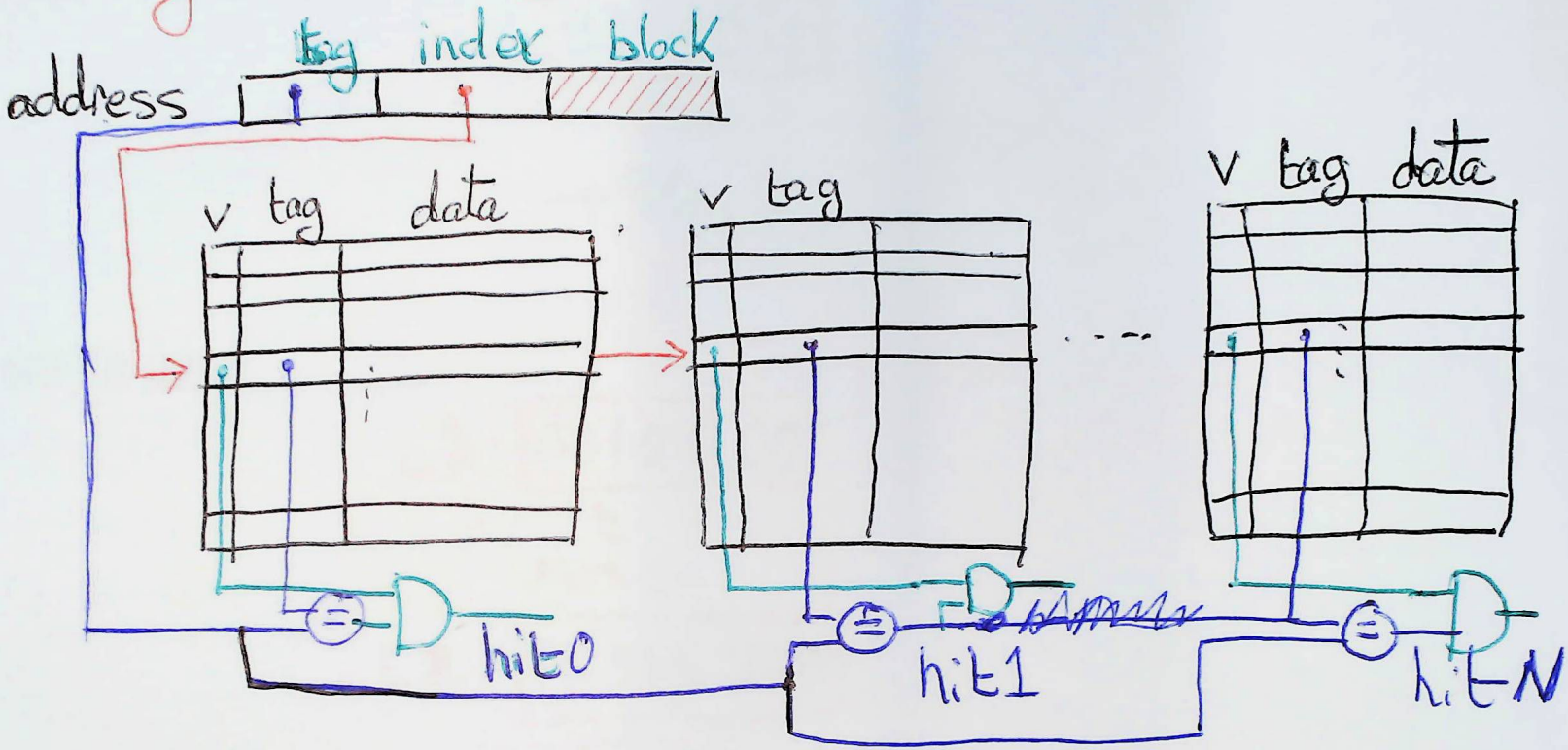



CPU
Level 1
Level 2
...
Level n
Increasing distance from the CPU in access time
Levels in the memory hierarchy
Size of the memory at each level


Address (showing bit positions)
31 ··· 14 13 ··· 6 5 ··· 2 1 0
18
8
4
Byte offset
Hit
Tag
Index
Block offset
Data
18 bits
512 bits
V
Tag
Data
256 entries
18
32
32
32
=
Mux
32


Address
31 30 ··· 12 11 10 9 8···3 2 1 0
22
8
Tag
Index
Index
V
Tag
Data
0
1
2
253
254
255
22
32
=
4-to-1 multiplexor
Hit
Data

Associativity in caches



- associativity helps avoid thrashing on indexes.

what happens to stores?

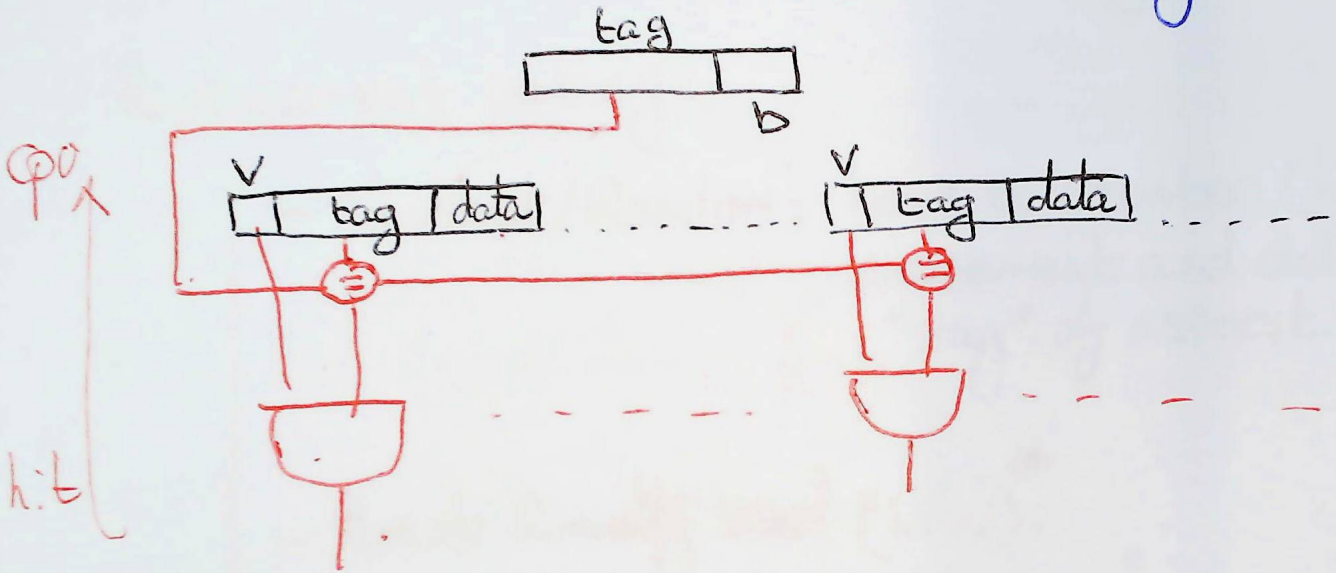
data comes from cpu.
nothing to fetch from further down hierarchy on miss.

write consistency problem:
writen blocks can be clobbered by later requests that land on same index.
then future reads of the data will come from further down hierarchy, which will be stale (data from before cpu store).

solutions
- write through: writes go all the way down the hierarchy, to make sure no stale version of data is stored anywhere.
- write-back: mark data stored by cpu as "dirty". only send writes to next level of memory if its dirty and needs to be kicked out.

Fully associative cache

Extreme case of associativity



the index is shrunk to zero bits (only one index)

What to do on cache misse when we have associtivity?

Replacement policy:

- Counter/Random: have a counter (hashed) to increment and determine next "way" of associtivity to be Replaced.

- Least Recently Used (LRU)*: Keep track of "age" of each "way" of associtivity with a counter that is reset on accesses to that way and incremented on access to other "way" in that index. That "age" tells you who is LRU.

- Data-specific replacement: provide mechanisms for inserting data with high age, make it kicked out soon.

Need to be able to quantify cache quality.

Cache hit rate: measure of cache performance

$$\text{hit rate} = \frac{\text{number of accesses that "hit"}}{\text{number of accesses}}$$

(# hits + # misses)

Different types of cache misses

- Compulsary Miss: First time accessing data/block/address.
- Capacity misse: not first access to the block but got "Ricked out" due to limited capacity.
- Conflict miss: due to competition for same index while total capacity still not exceeded.

Cache hierarchy Average Access Time /Mem

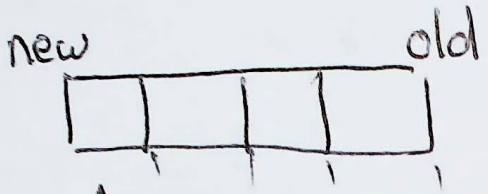
$$ATT = t_1 + m_1 t_2 + m_1 m_2 t_3 + m_1 m_2 m_3 t_4 \ldots$$

t_n = access time to cache level n

m_n = miss rate of cache level n (1 - hit rate)

Design Choices in caches

choice	hit/miss rate	access time
# indexes	capacity ↑ hit rate ↑ capacity miss ↓ conflict .. .. ↓	access time ↑ larger cap. on wires.
associativity	capacity ↑ hit rate ↑ capacity conflict miss ↓	access time ↑↑ slower tag compare
Block size	capacity ↑ hit rate ↑ compulsary miss ↓	access time ↑ (loss) due to slower fill on miss



						Miss/hit
A	A					m
B	B	A				m
C	C	B	A			m
D	D	C	B	A	→	m
E	E	D	C	B	→	m
A	A	E	D	C	→	m
B	B	A	E	D		m
A	A	B	E	D		h
B	B	A	E	D	→	h
C	C	B	A	E	→	m
D	D	C	B	A	→	m
E	E	D	C	B		m

$$\text{hit rate} = \frac{2}{12} = \frac{1}{6}$$