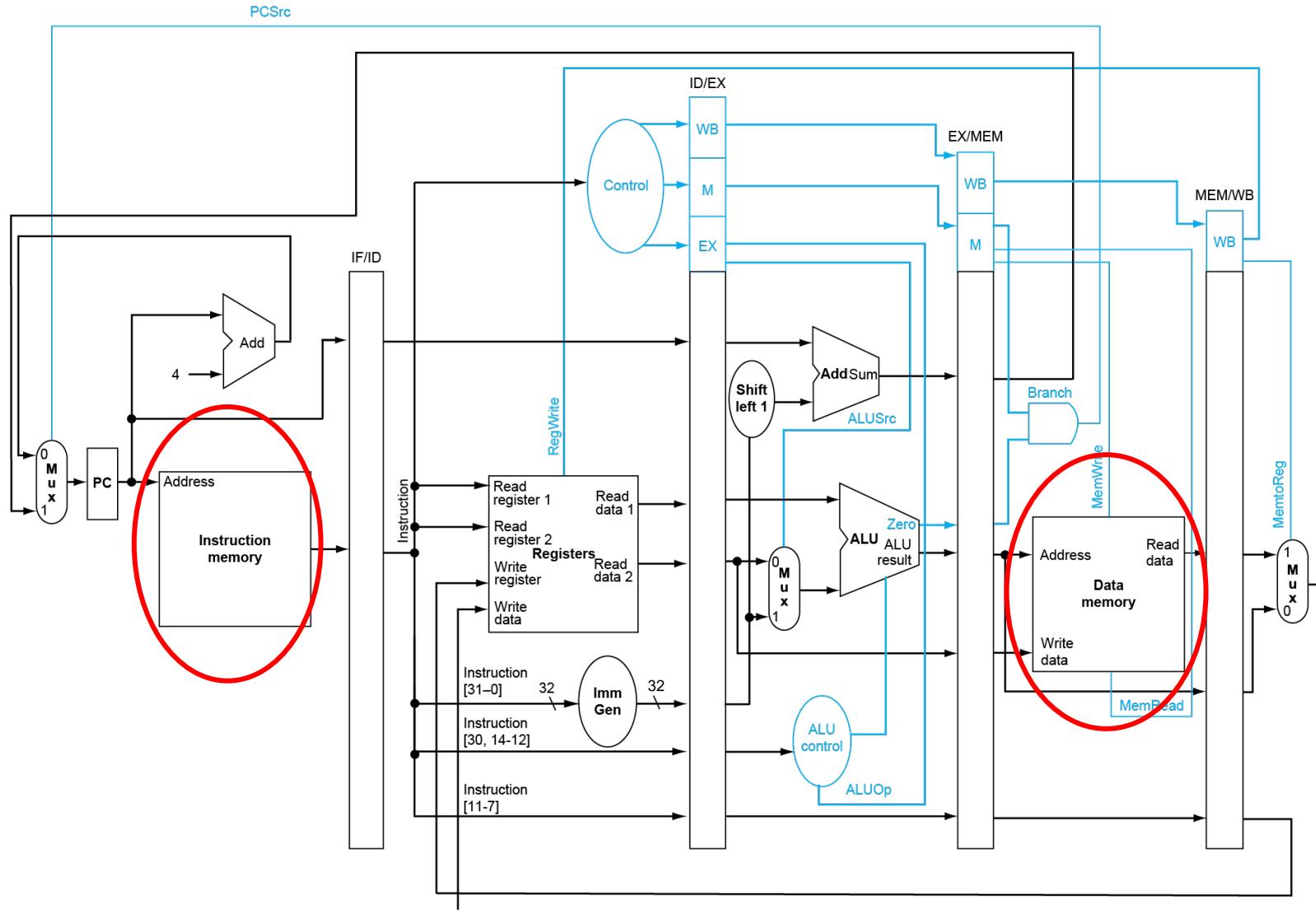


Topic 9

Memory Hierarchy

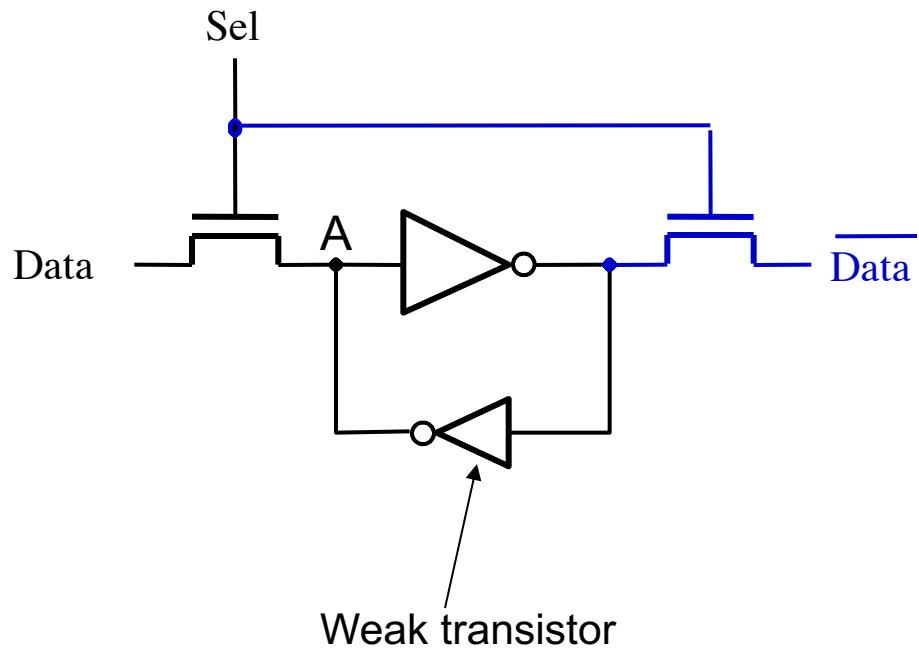
- Cache (1)

RISC-V Pipeline Architecture



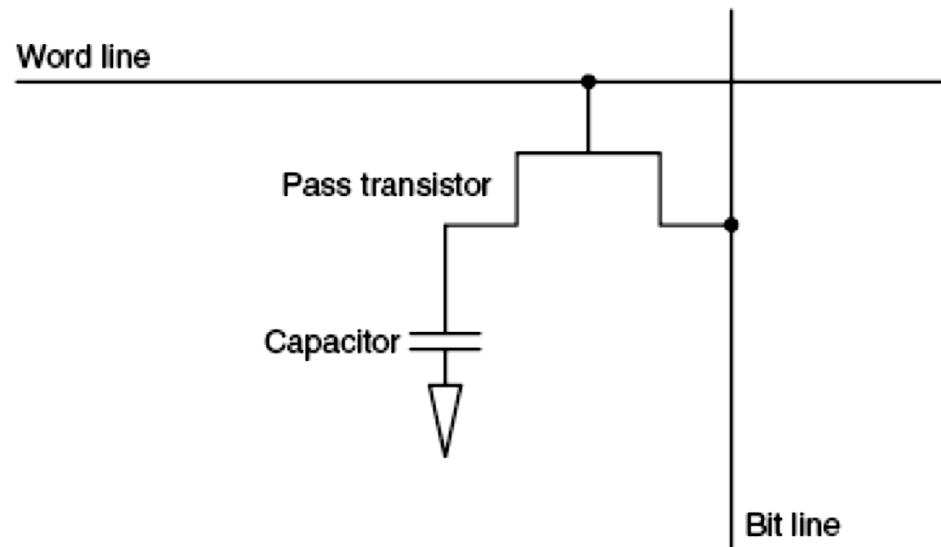
Static RAM (SRAM)

- When $\text{Sel} = 1$, Data is stored and retained in the SRAM cell by the feedback loop
- When $\text{Sel} = 1$, Data can be read out on the same port
- Point A is driven by both the Data transistor and the smaller inverter, but the Data transistor wins because the inverter is implemented using a weak transistor



Dynamic RAM (DRAM)

- Write: turn on word line, charge capacitor through pass transistor by bit line
- Read: charge bit line halfway between high and low, turn on word line, then sense the voltage change on bit line
 - 1 if voltage increases
 - 0 if voltage decreases



Examples of Memory

- USB
- Disk
- Hard drive
- Flash/SSD
- ...



- FASTER PERFORMANCE
- NO VIBRATIONS OR NOISE
- MORE ENERGY EFFICIENT

- CHEAPER PER GB
- AVAILABLE IN LARGE VERSIONS



Source: Internet

Volatile vs. Non-volatile memory (NVM)

- Non-volatile memory (NVM) is a type of memory that retains stored data after the power is turned off.

Examples of Non-Volatile Memory



floppy disk



compact disc



internal hard drive



external hard drive



flash drive



SD card

Memory Technology

- Static RAM (SRAM)
 - 0.5ns – 2.5ns, \$500 – \$1000 per GB
- Dynamic RAM (DRAM)
 - 50ns – 70ns, \$3 – \$6 per GB
- Magnetic disk
 - 5ms – 20ms, \$0.01 – \$0.02 per GB
- Ideal memory
 - Access time of SRAM
 - Capacity and cost/GB of disk

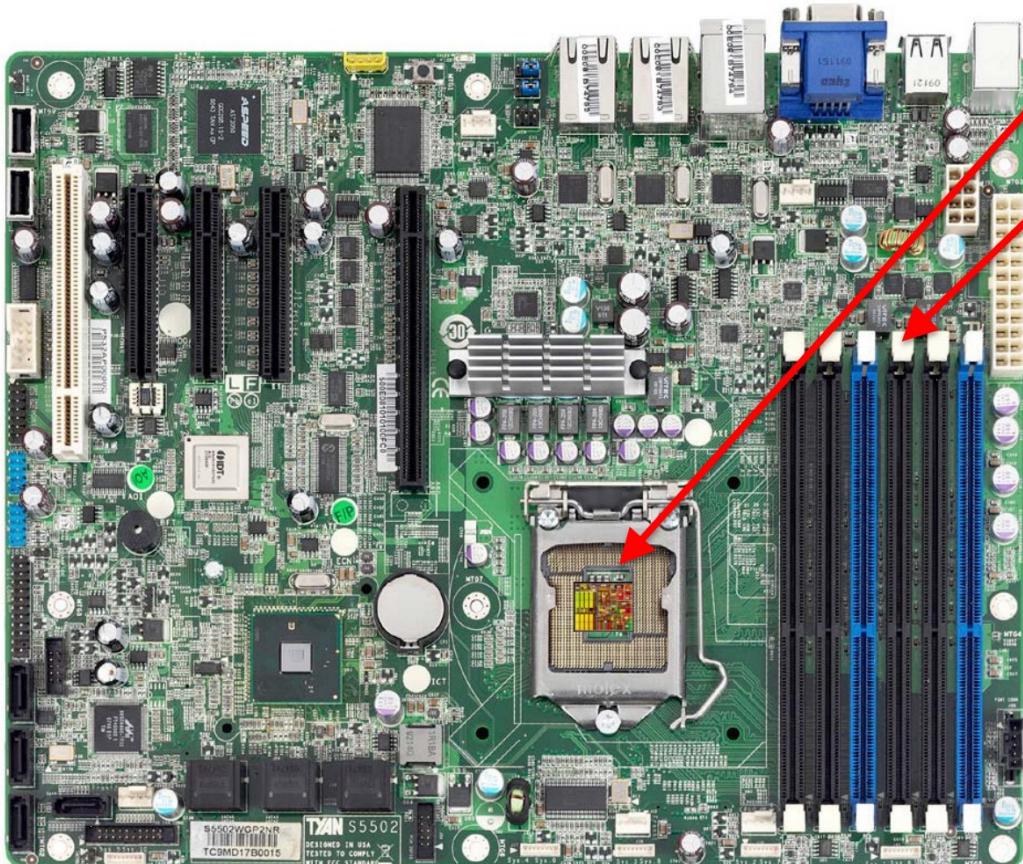
Wishful Memory

- So far we **imagined/assumed**
 - a program owns one big instruction memory and one data memory
 - One cycle per stage...

4.1. Ideally one would desire an indefinitely large memory capacity such that any particular aggregate of 40 binary digits, *word* (cf. 2.3), would be immediately available—i.e. in a tin

----- Burks, Goldstein, von Neumann, 1946

What is the problem?



CPU

Main Memory

+ big

- slow

- far away

SandyBridge Motherboard, 2011
<http://news.softpedia.com>

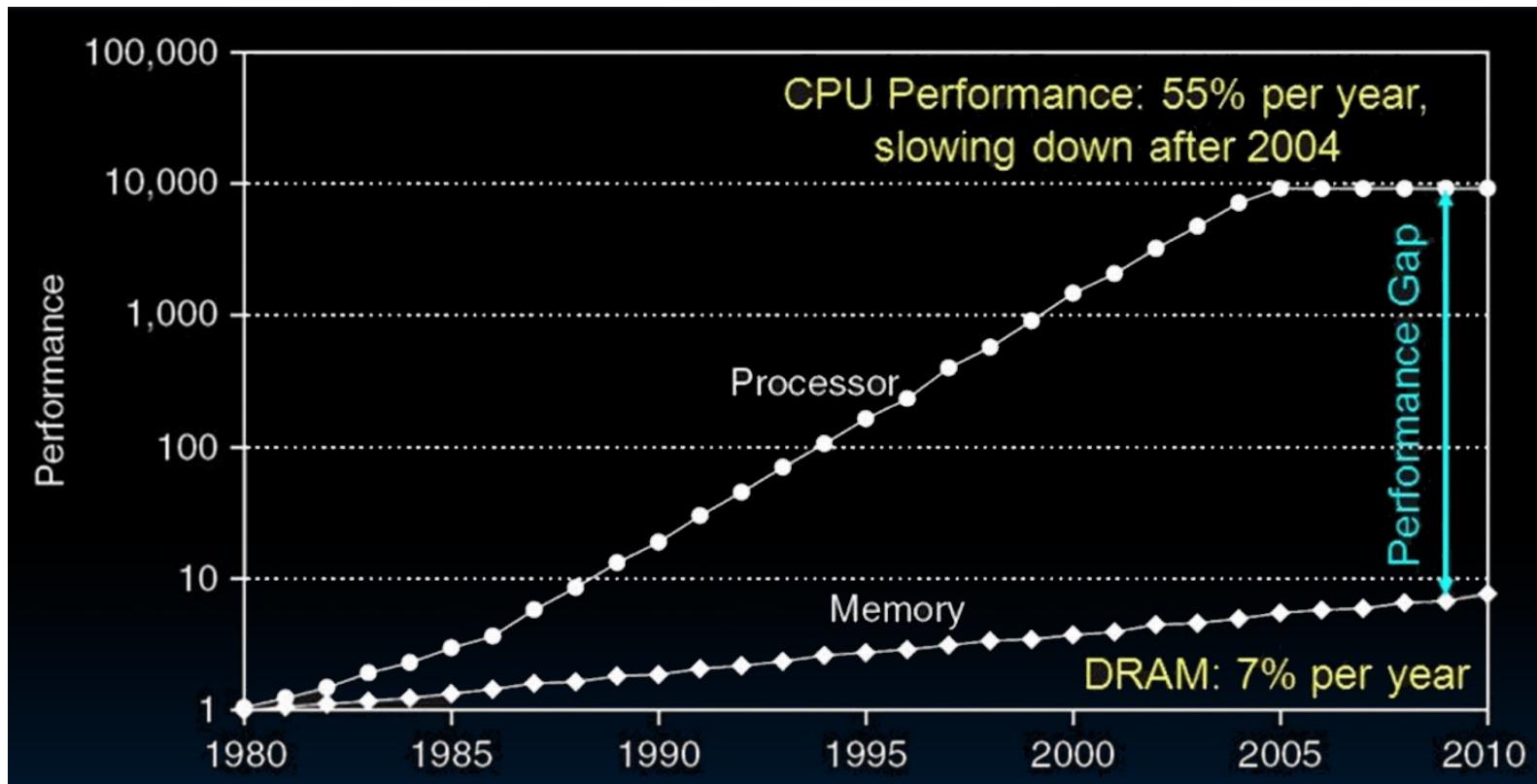
The Need for Speed

- Instruction speeds:
 - add,sub,shift: 1 cycle
 - mult: 3 cycles
 - load/store: 100 cycles
 - off-chip 50(-70)ns
 - 2(-3) GHz processor ~ 0.5 ns clock cycle

The truth

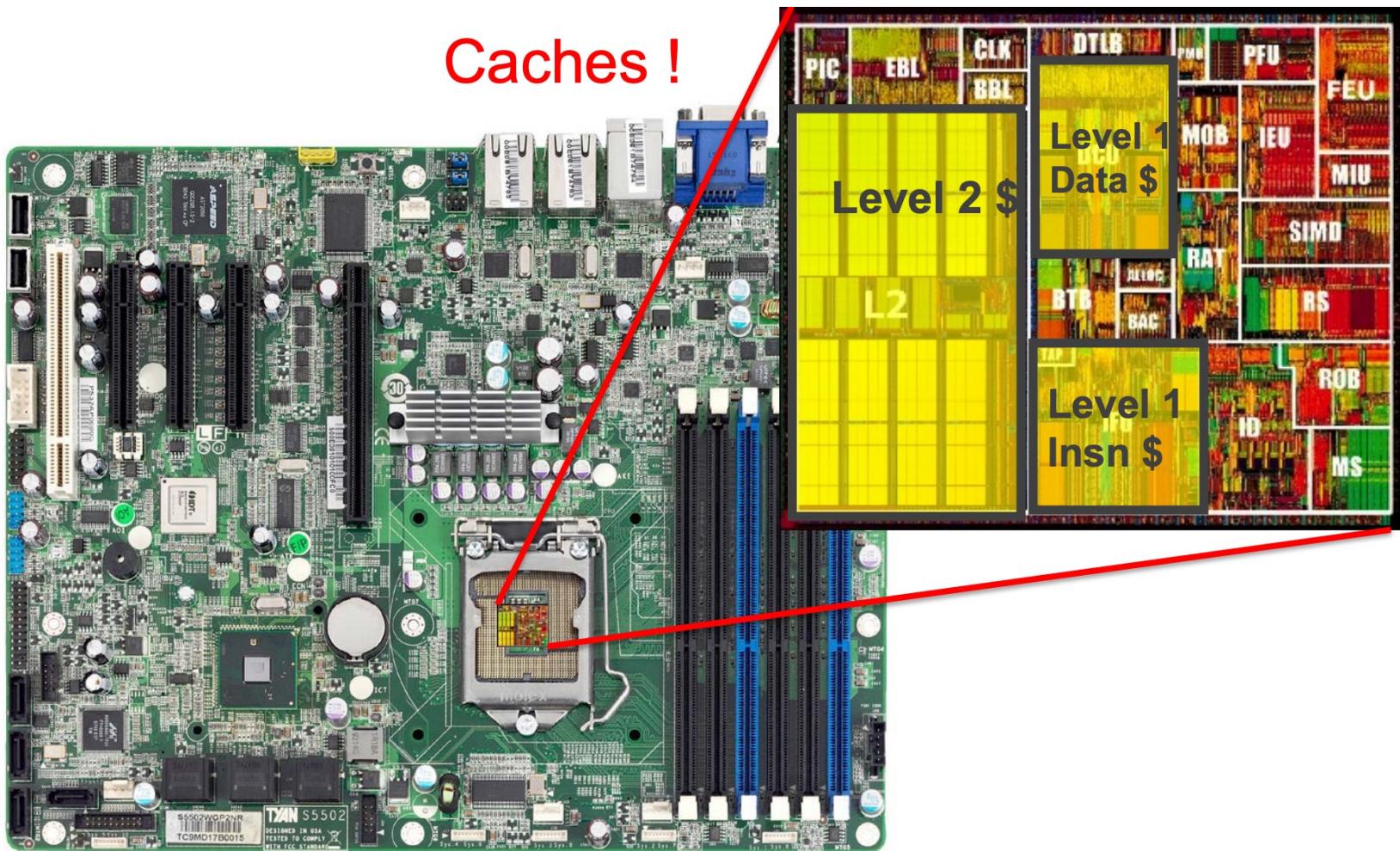
- Can't afford/don't need as much memory as size of address space
- Can't find memory technology that is affordable in GByte and also cycle in GHz

Processor-DRAM Gap (Latency)



- 1980 microprocessor executes ~one instruction in same time as DRAM access
- 2020 microprocessor executes ~1000 instructions in same time as DRAM access
- Slow DRAM access has disastrous impact on CPU performance!

What is the solution here?



Intel Pentium 3, 1999

Cache

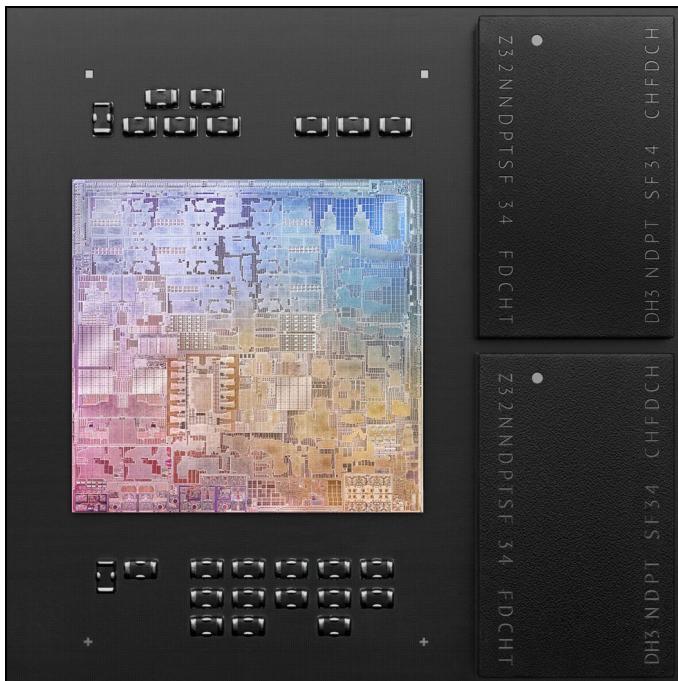
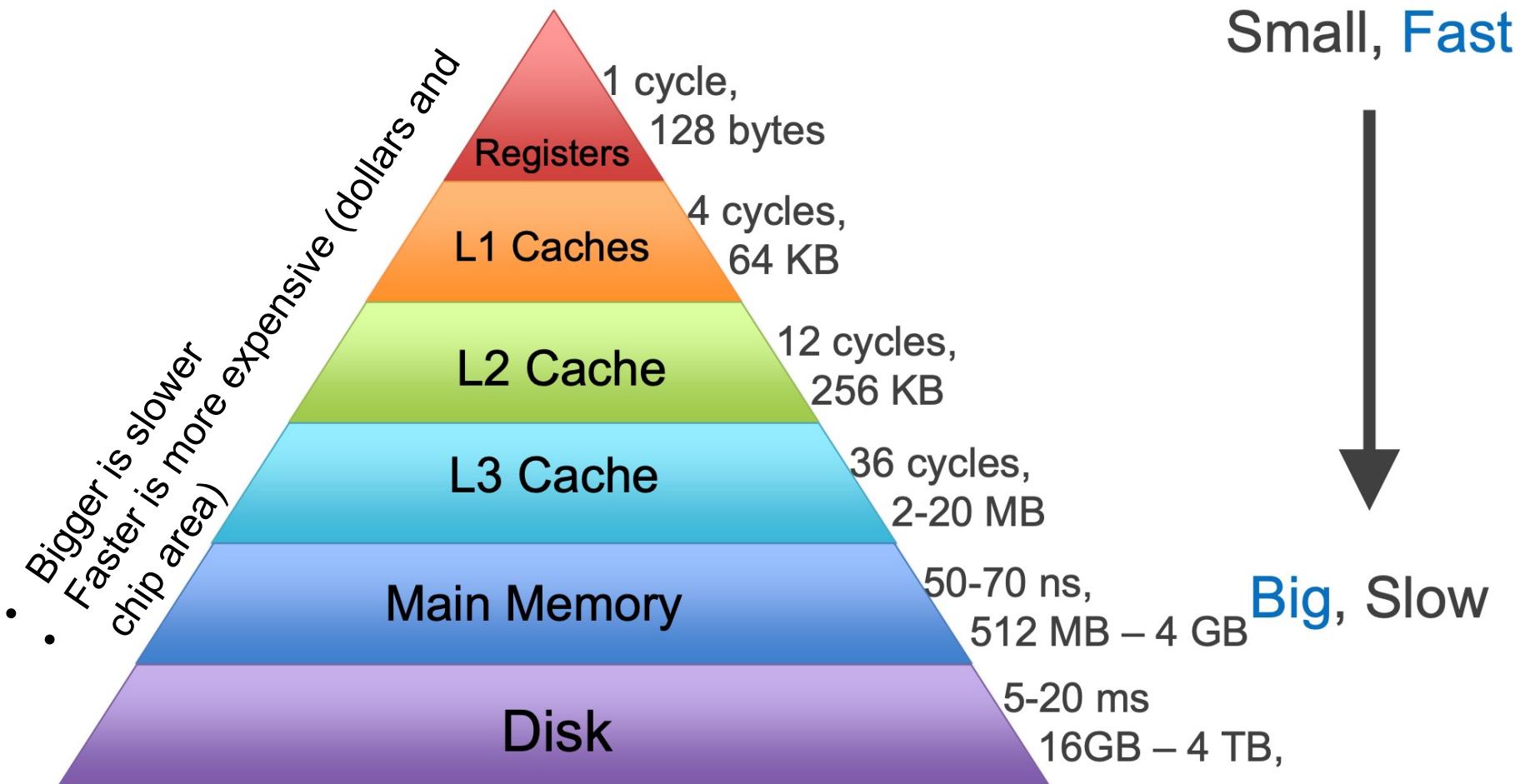


Image: Apple, Inc.

Apple M1 chip	
General information	
Launched	November 10, 2020 ^[1]
Designed by	Apple Inc.
Common manufacturer(s)	TSMC
Product code	APL1102 ^[2]
Performance	
Max. CPU clock rate	3.2 GHz ^[1]
Cache	
L1 cache	320 KB per core (performance cores, 192 instructions + 128 data) 192 KB per core (efficient cores, 128 instructions + 64 data)
L2 cache	12 MB (performance cores) 4 MB (efficient cores)
Architecture and classification	
Application	Desktop (Mac Mini, iMac), Notebook (MacBook family), Tablet (iPad Pro)
Min. feature size	5 nm
Microarchitecture	"Firestorm" and "IceStorm" ^[1]
Instruction set	ARMv8.4-A
Physical specifications	
Transistors	16 billion
Cores	8 (4× high-performance + 4× high-efficiency)
GPU(s)	Apple-designed integrated graphics (up to 8 cores)
History	
Predecessor	Intel Core and Apple T2 chip (Mac) Apple A12Z Bionic (iPad Pro)
Architecture:	Arm-based
CPU Cores:	8-core CPU
Nm Process:	5nm
Graphics:	Integrated 8-core GPU with 2.6 teraflops of throughput
Memory:	8GB or 16GB of LPDDR4X-4266 MHz SDRAM

Source: wiki

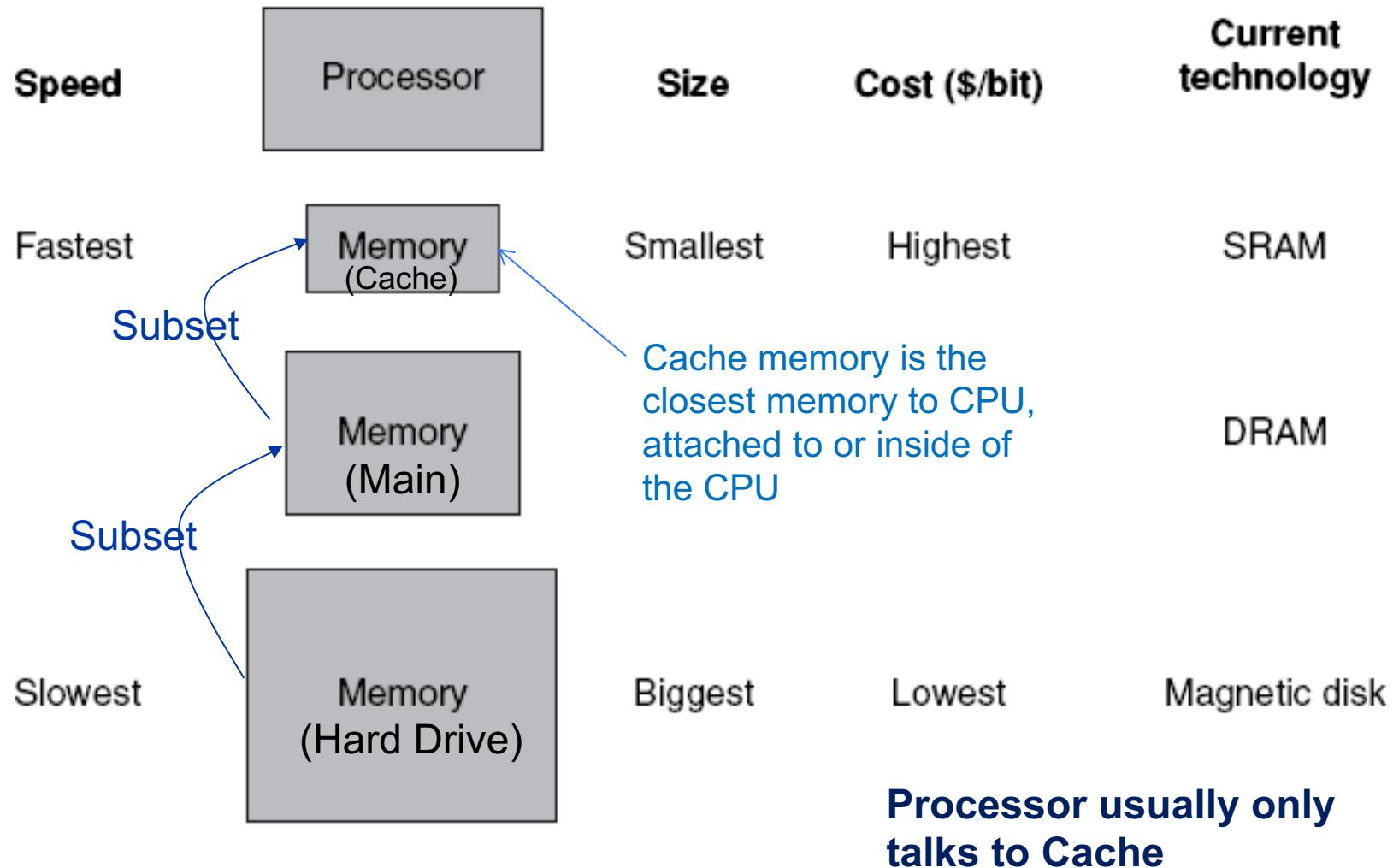
Memory Hierarchy Levels



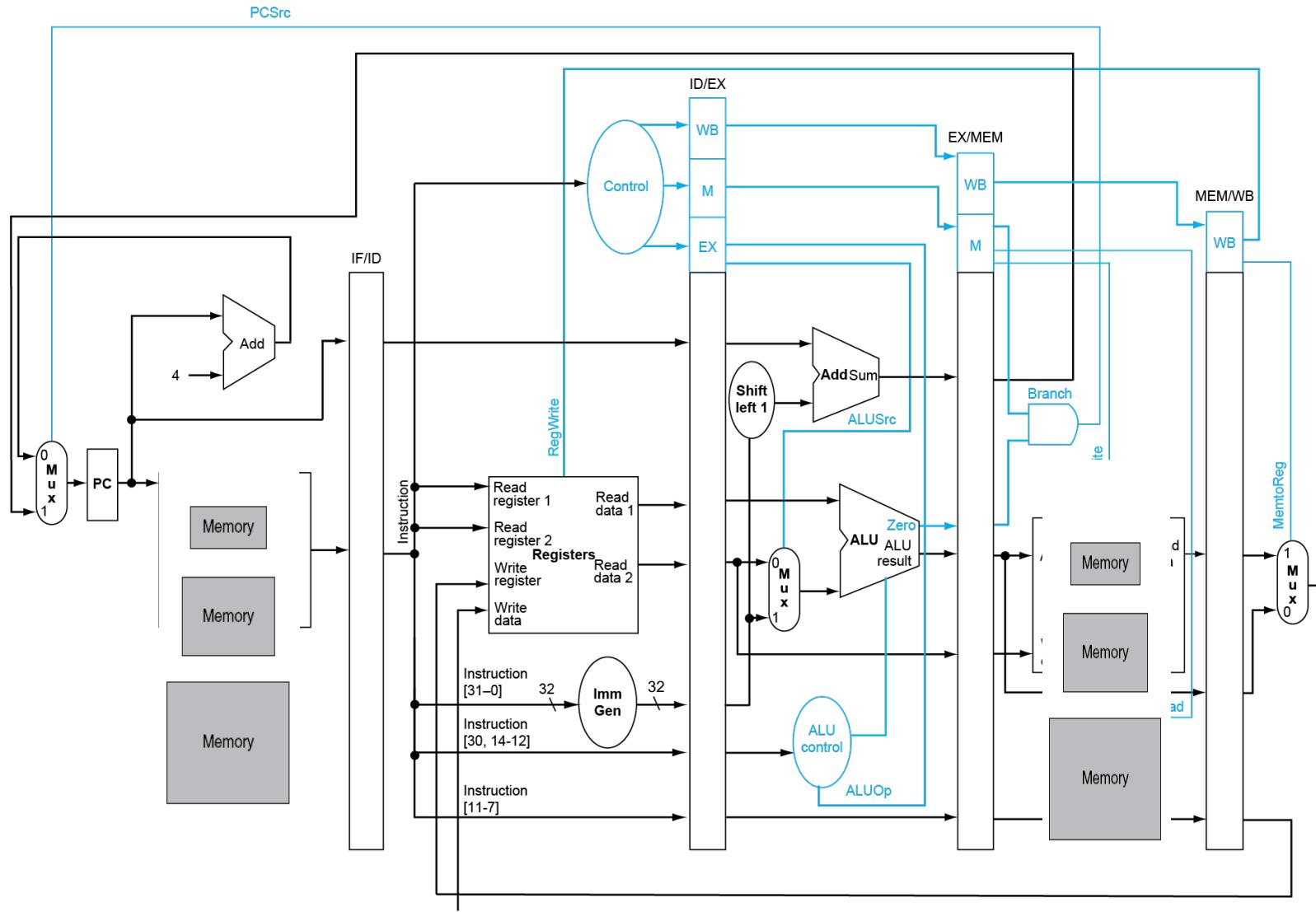
Intel Haswell Processor, 2013

Image: cs.cornell.edu/courses/cs3410/

Memory Hierarchy



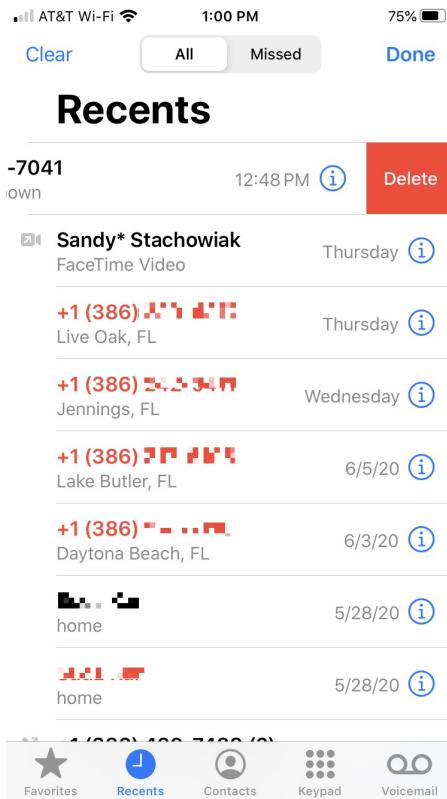
Memory Hierarchy in RISC-V



Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
 - Items that are accessed recently are likely to be accessed again soon
 - e.g., instructions in a loop
- Spatial locality
 - Items near those that are accessed recently are likely to be accessed soon
 - E.g., sequential instruction access, array data

Life is full of Locality



- Last Called
- Speed Dial
- Favorites Contacts

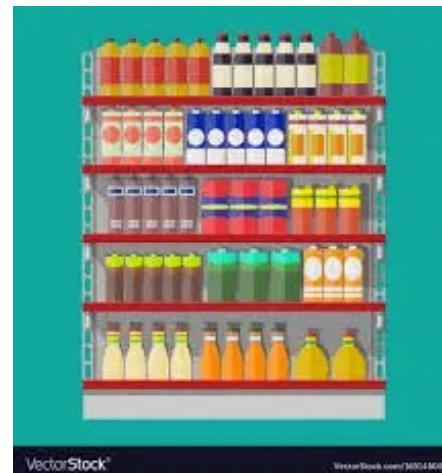
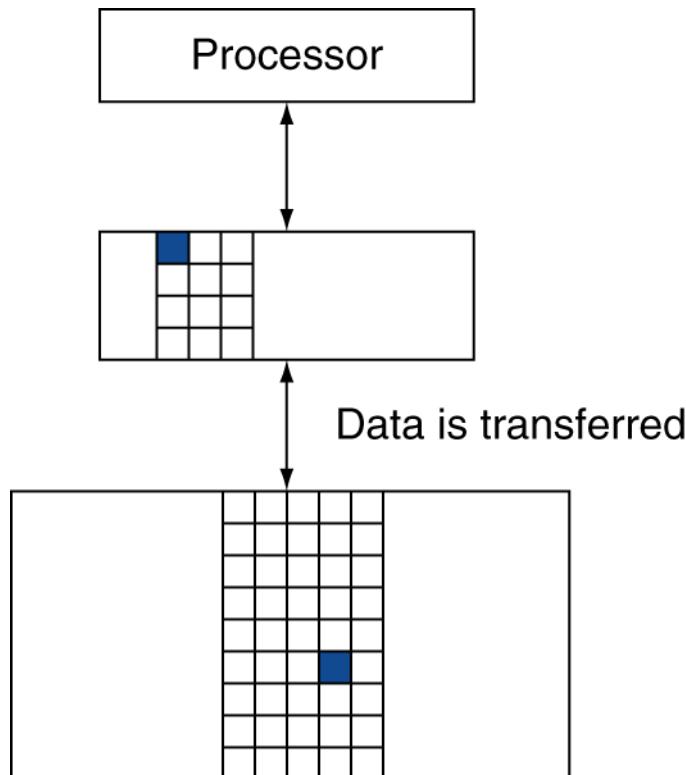


Image: Internet

Principle of Memory Access

- Taking Advantage of Locality
- If a word of data/instruction is referenced
 - Copy this recently accessed word (temporal locality) and nearby items (spatial locality) together as a block from lower level memory to higher level memory (Hard Drive to Main memory or Main memory to Cache)

Access the Memory Hierarchy

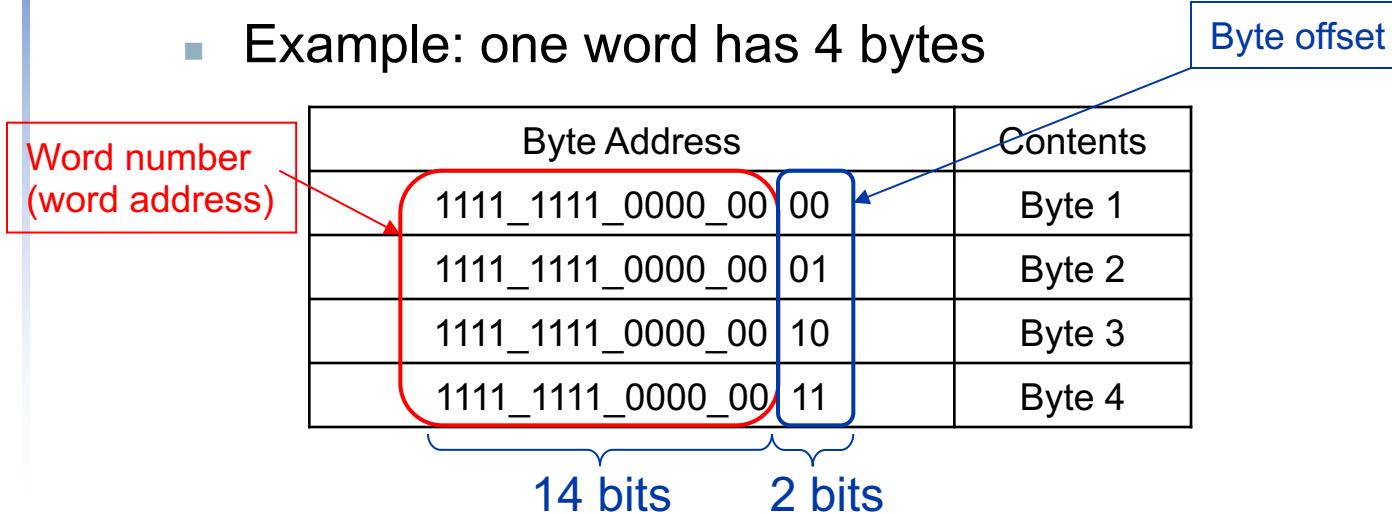


- Concepts:

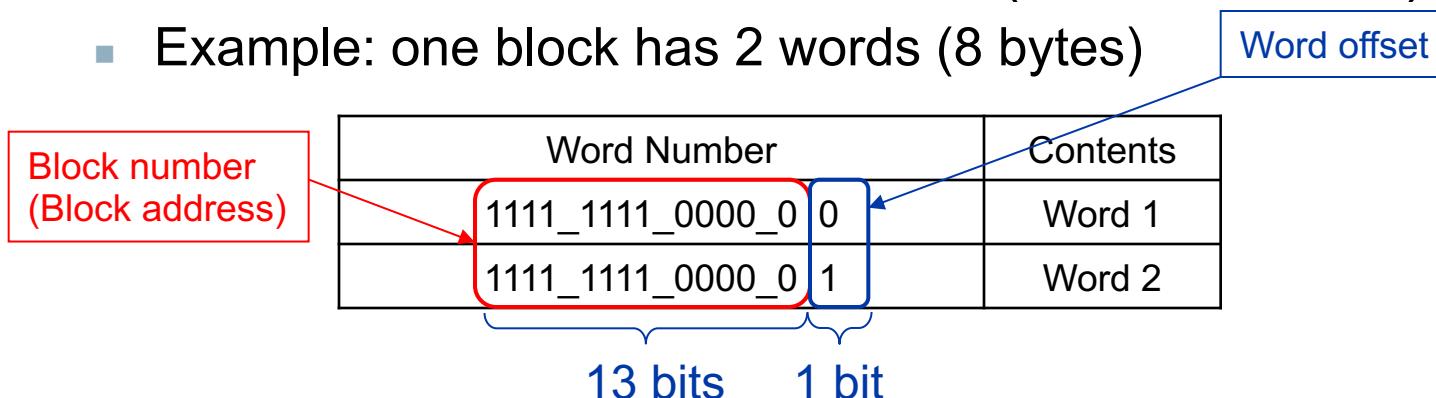
- Block (aka line)
 - Unit of data referencing to take advantage of temporal and spatial locality
 - May be one or multiple words
 - Block also has an address

Block, Word, Byte Addresses

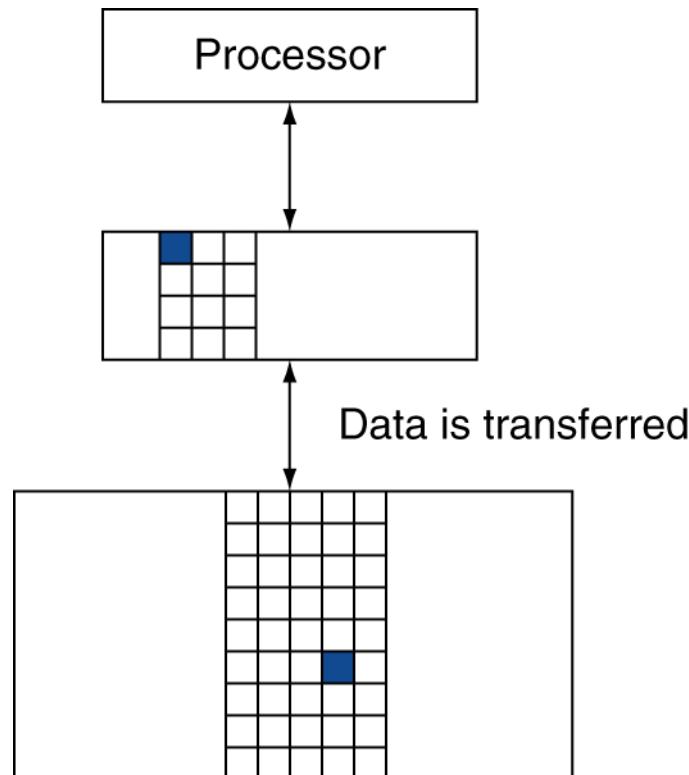
- Byte address vs. word address (assume 16-bit address)
 - Example: one word has 4 bytes



- Word address vs. block address (block number)
 - Example: one block has 2 words (8 bytes)

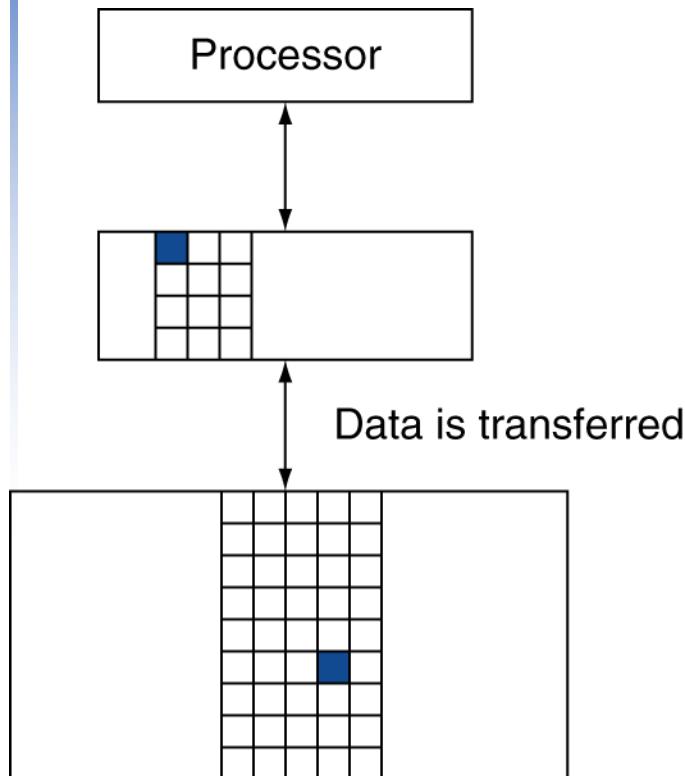


Access the Memory Hierarchy



- Concepts:
 - Hit
 - If accessed data is present in upper level, access satisfied by upper level
 - Hit rate: hits/accesses
 - Hit time: time to access a memory including
 - Time to determine whether a hit or miss
 - Time to pass block to requestor

Access the Memory Hierarchy



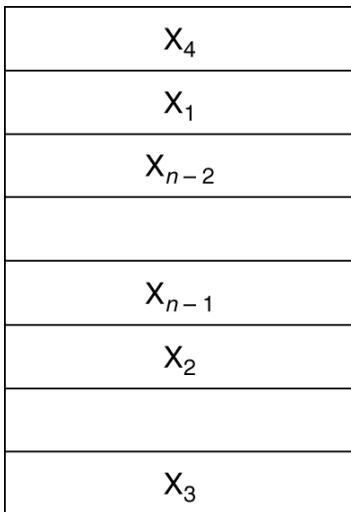
- Concepts:
 - Miss
 - If accessed data is absent
 - Block copied from lower to higher level
 - Then accessed data supplied from upper level
 - Time taken: miss penalty
 - Miss rate: misses/accesses
 $= 1 - \text{hit rate}$

Miss Penalty

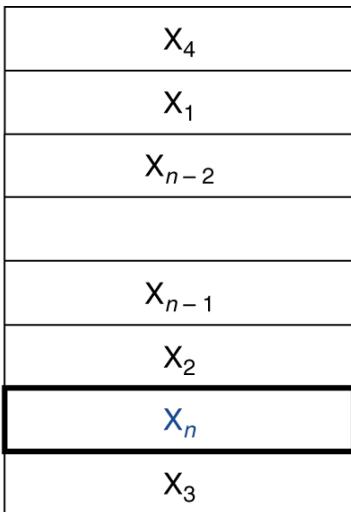
- Miss (time) penalty
 - Time to fetch a block from lower level upon a miss including
 - Time to access the block
 - Time to transfer it between levels
 - Time to overwrite the higher level block
 - Time to pass block to requestor

Cache Miss

- Assume a cache with 1-word blocks X_1, \dots, X_{n-1} .
Now CPU requests X_n



a. Before the reference to X_n



b. After the reference to X_n

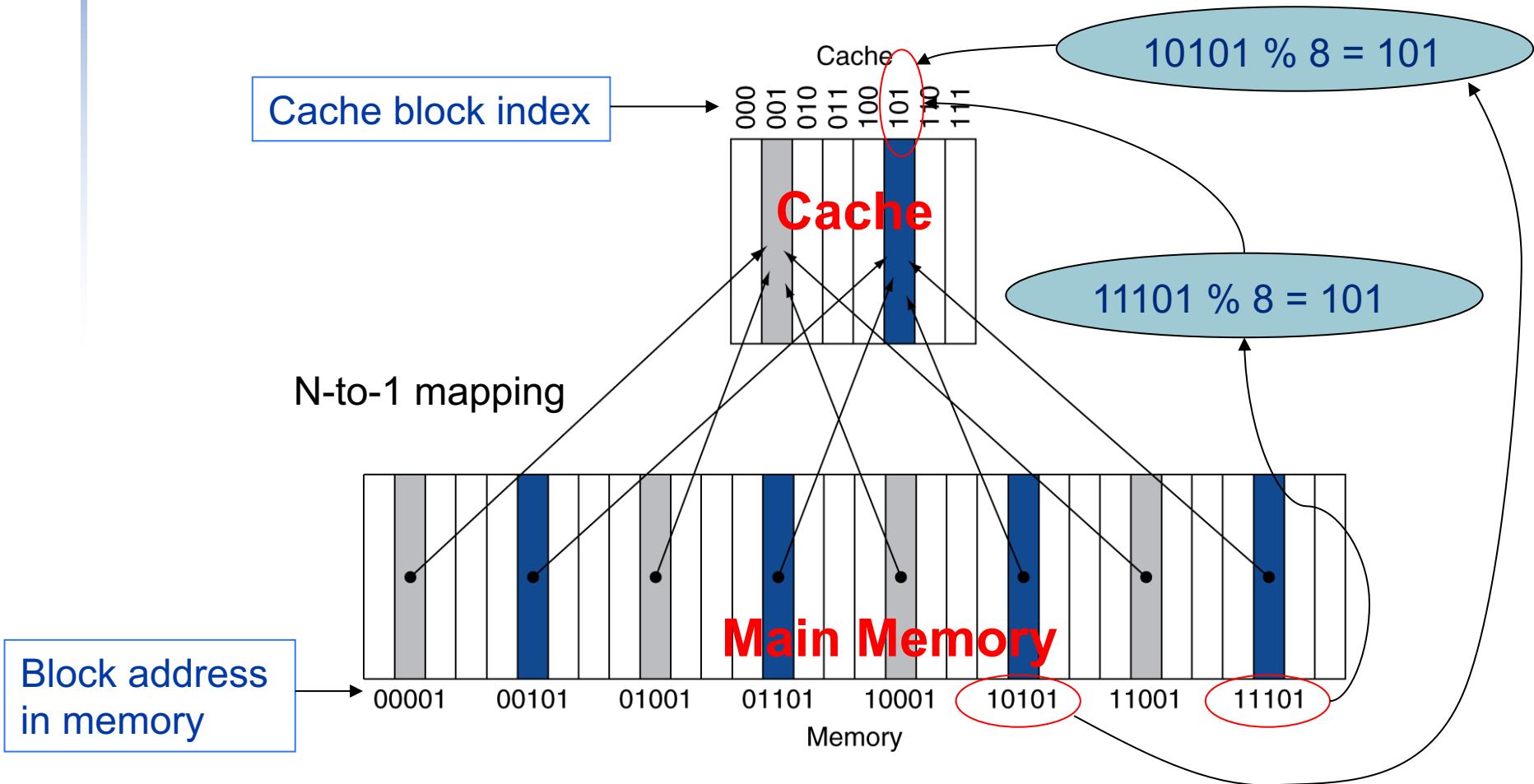
- Miss. Then X_n is brought in from lower level
- But
 - How do we know if there is a hit or miss?
 - Where do we look?

Direct Mapped Cache

- Direct mapped cache: each memory location corresponds to **one** choice in cache
- How to map?
- Location of a block in cache is determined by address of the requested word
 - Given word address, we can get block address (block number)
 - *Cache location (or cache block index) =
(Block address in memory) % (Number of blocks in cache) =
lower $\log_2(\text{Number of blocks in cache})$ bits of block address*
- **Multiple** memory locations correspond to **one** block in cache – n-to-1 mapping

Direct Mapped Cache

- Number of blocks in a cache is a power of 2
- Low-order bits of block address in memory = cache index

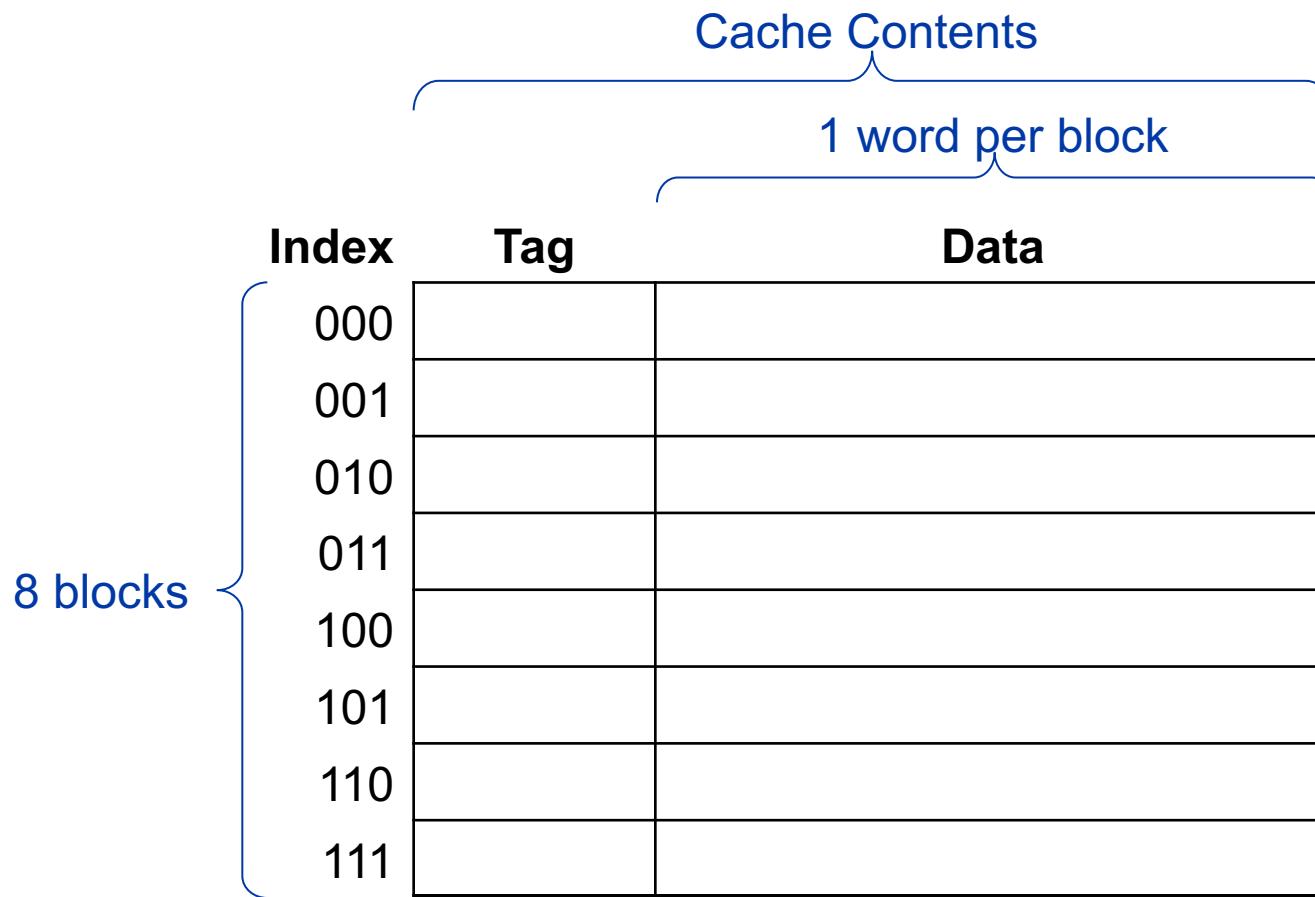


Tags in Cache

- How do we know which block of data is present in cache since it's n-to-1 mapping?
 - Store part of block address together with the data when the data is copied to higher level
 - Only need to store the high-order bits of memory block address, excluding the *lower $\log_2(\text{Number of blocks in cache})$ bits of block address*
 - Called **tag**

Direct Mapped Cache Example

- 8-block cache
- 1 word per block (word address = block address)

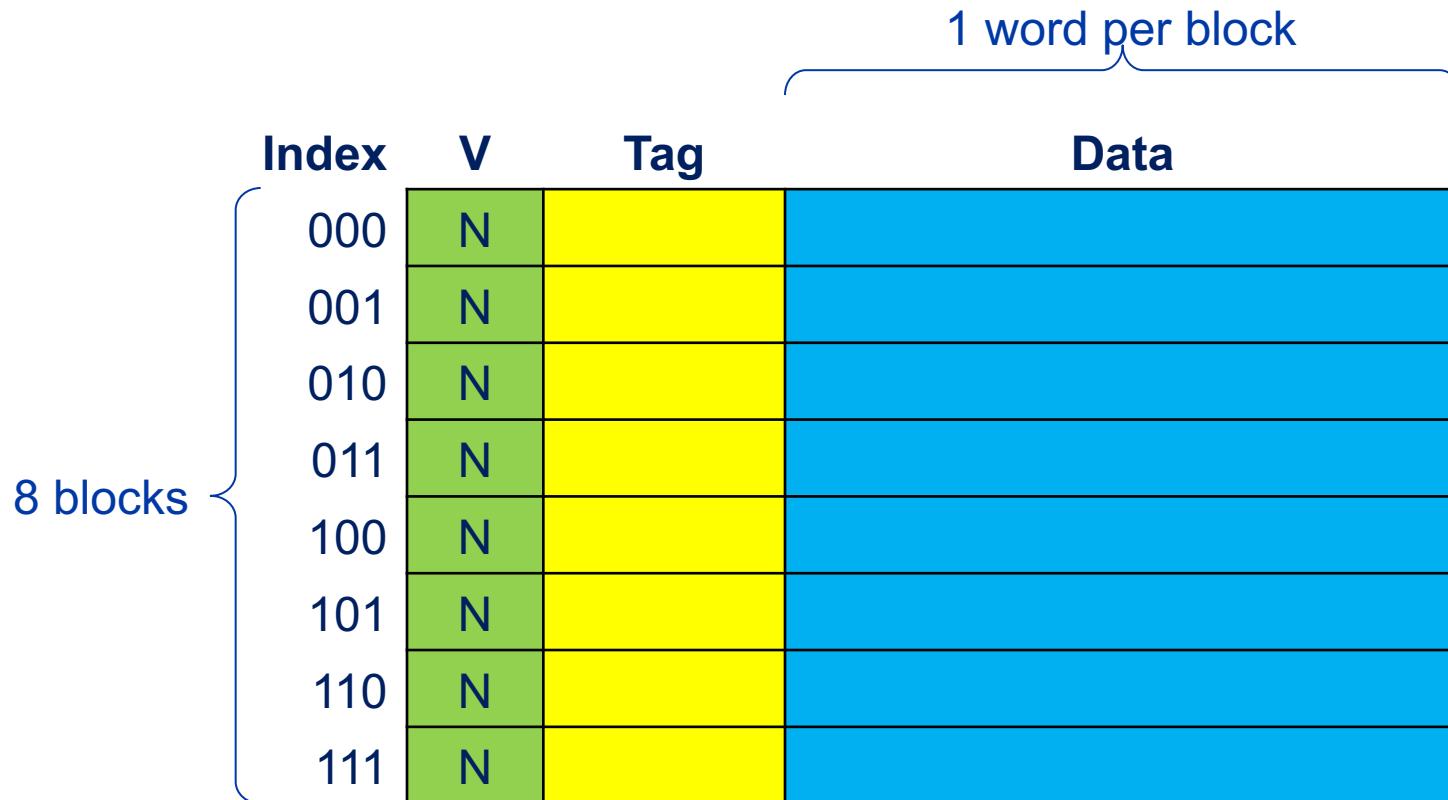


Valid Bits in Cache

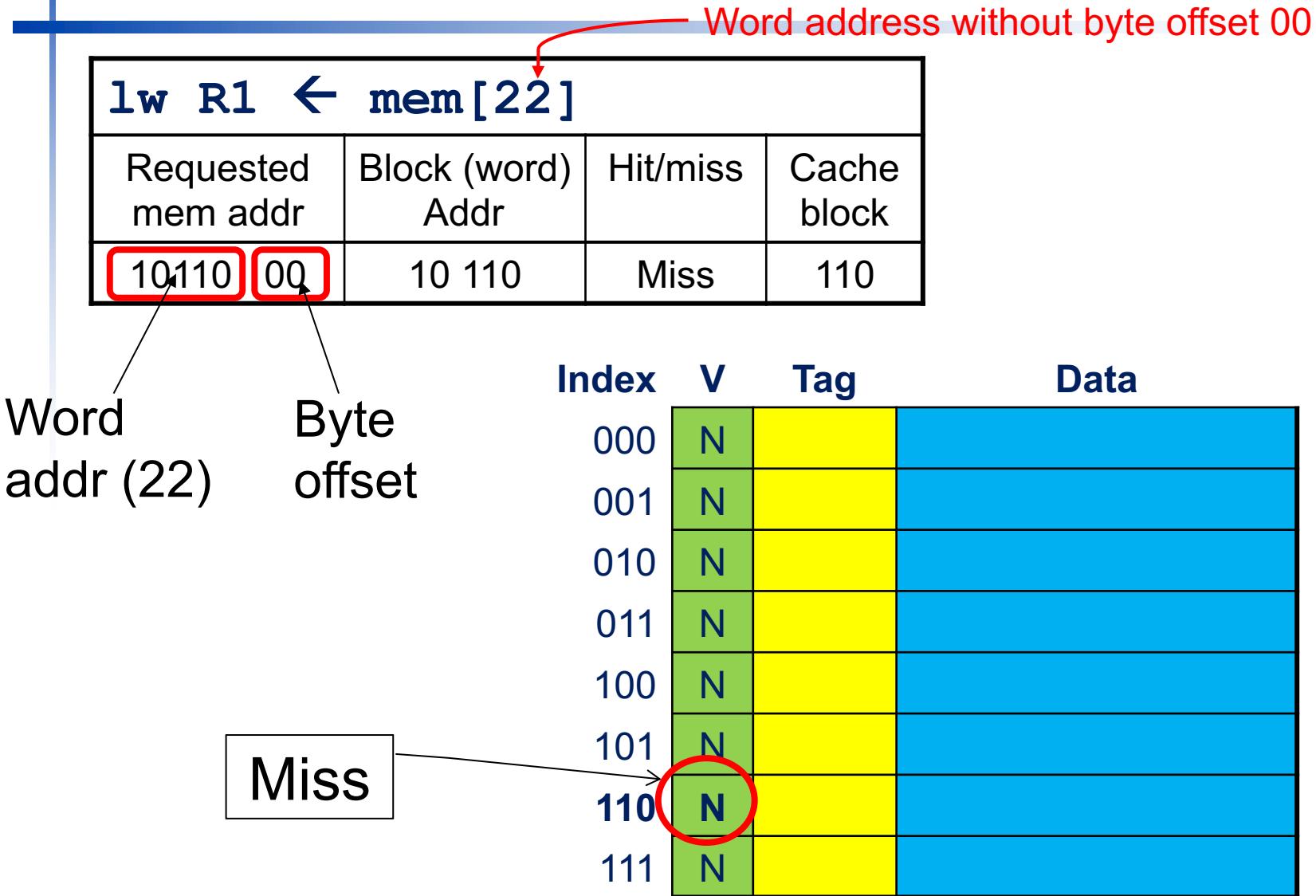
- Waste searching time if there is no valid data in a location
 - Valid bit: 1 = present, 0 = not present
 - Initially 0 (N), to improve search speed

Cache Example 1

- 8-block cache
- 1 word per block (word address = block address)
- direct mapped
- Assuming 7-bit byte addresses



Cache Example 1 (cont.)



Cache Miss

- On cache hit, CPU proceeds normally
- On cache miss
 - Stall the CPU pipeline
 - Using processor control unit and a cache controller
 - Freezes registers and waits for memory
 - Fetch block from next level of hierarchy

Cache Example 1 (cont.)

lw R1 ← mem[22]			
Requested mem addr	Block (word) Addr	Hit/miss	Cache block
10110 00	10 110	Miss	110

byte offset 00
are omitted

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	0x05ACF011
111	N		

Cache memory

Word Addr	Data
00000(0)	0x81230431
00001(1)	0xABCD3305
...	...
10101(21)	0x12345678
10110(22)	0x05ACF011
...	...
11110(30)	0x00000000
11111(31)	0x00000000

Main memory

Cache Example 1 (cont.)

lw R2 ← mem [26]			
Requested mem addr	Block (word) Addr	Hit/miss	Cache block
11010 00	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	0x5DC60007
011	N		
100	N		
101	N		
110	Y	10	0x05ACF011
111	N		

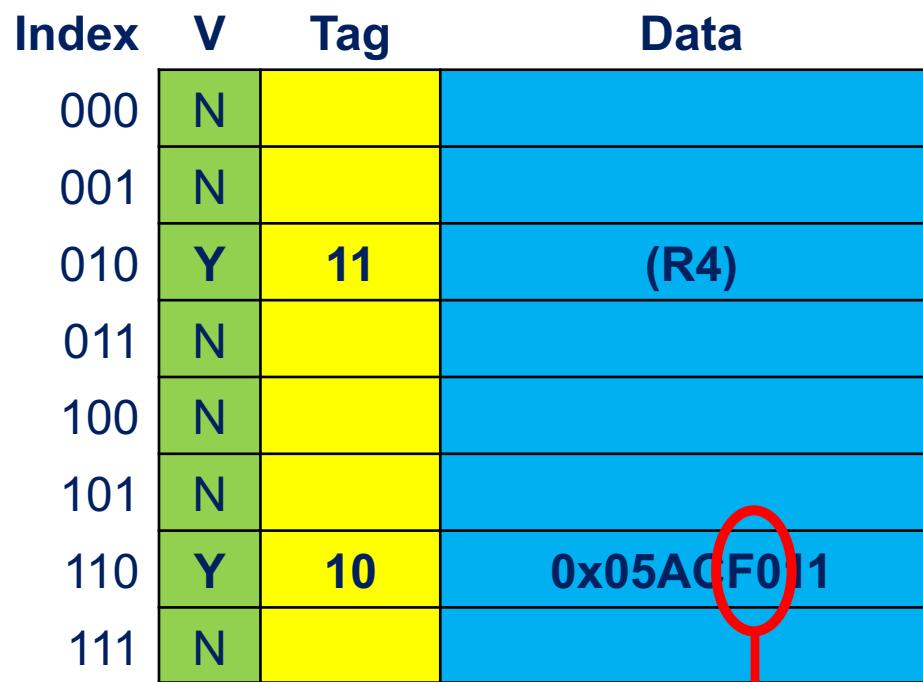
Cache memory

Word Addr	Data
00000 (0)	0x81230431
00001 (1)	0xABCD3305
...	...
10101 (21)	0x12345678
10110 (22)	0x05ACF011
...	...
11010 (26)	0x5DC60007
...	...
11110 (30)	0x00000000
11111 (31)	0x00000000

Main memory

Cache Example 1 (cont.)

	Requested mem addr	Block (word) Addr	Hit/miss	Cache block
lb R3 ← mem[22]byte1	10110 01	10 110	Hit	110
sw R4 → mem[26]	11010 00	11 010	Hit	010



Hit due to temporal locality

R3 FFFFFFFFFF ←

Cache Example 1 (cont.)

Requested mem Addr	Block (word) Addr	Hit/miss	Cache block
1000000(lw)	10 000	Miss	000
0001100(sw R5)	00 011	Miss	011
1000000(lw)	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	0x8765ABCD
001	N		
010	Y	11	(R4)
011	Y	00	0xFFFF0002 → (R5)
100	N		
101	N		
110	Y	10	0x05ACF011
111	N		

Word Addr	Data
00000(0)	0x81230431
00001(1)	0xABCD3305
00010(2)	0xFFFF0001
00011(3)	0xFFFF0002
...	...
10000(16)	0x8765ABCD
...	...
10101(21)	0x12345678
10110(22)	0x05ACF011
...	...
11010(26)	0x5DC60007
...	...
11110(30)	0x00000000
11111(31)	0x00000000

Cache Example 1 (cont.)

Requested mem Addr	Block (word) Addr	Hit/miss	Cache block
1001000(lw)	10 010	Miss	010

Currently occupied by Mem[26]

Index	V	Tag	Data
000	Y	10	0x8765ABCD
001	N		
010	Y	10	(R4)→0x0000FF00
011	Y	00	0xFFFF0002→(R5)
100	N		
101	N		
110	Y	10	0x05ACF011
111	N		

Word Addr	Data
00000(0)	0x81230431
00001(1)	0xABCD3305
00010(2)	0xFFFF0001
00011(3)	0xFFFF0002
...	...
10000(16)	0x8765ABCD
10001(17)	0x12345678
10010(18)	0x0000FF00
...	...
11010(26)	0x5DC60007
...	...
11110(30)	0x00000000
11111(31)	0x00000000