University of Central Florida

Department of Electrical Engineering
and Computer Science

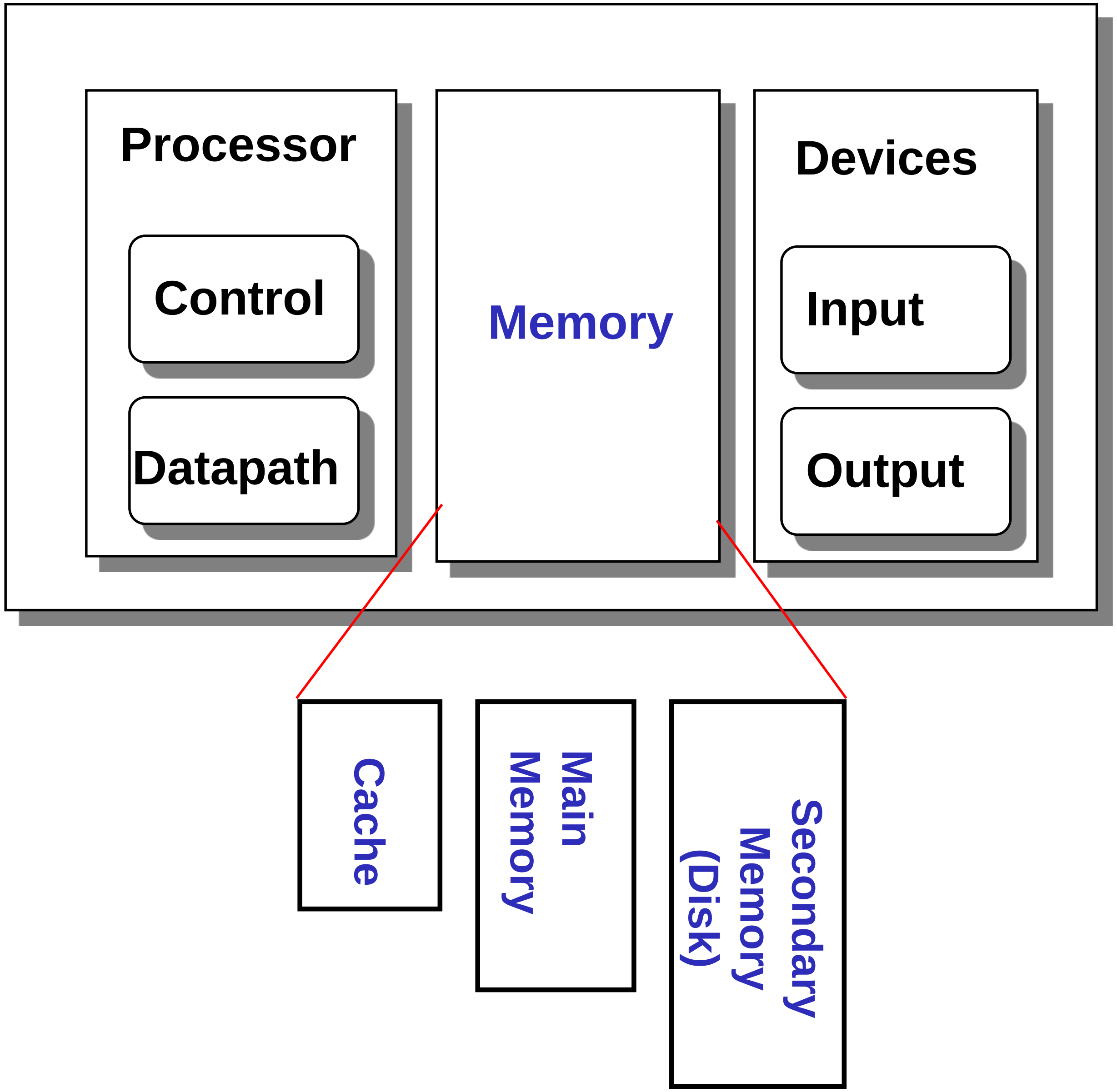
EEL 4768C: Computer Architecture

Memory Hierarchy

Instructor: Suboh A. Suboh

Source: Computer Organization and Design: The Hardware/Software Interface”, Patterson & Hennessy, Third Edition, 4th Rev Edition

Major Components of a Computer



How is the Hierarchy Managed?

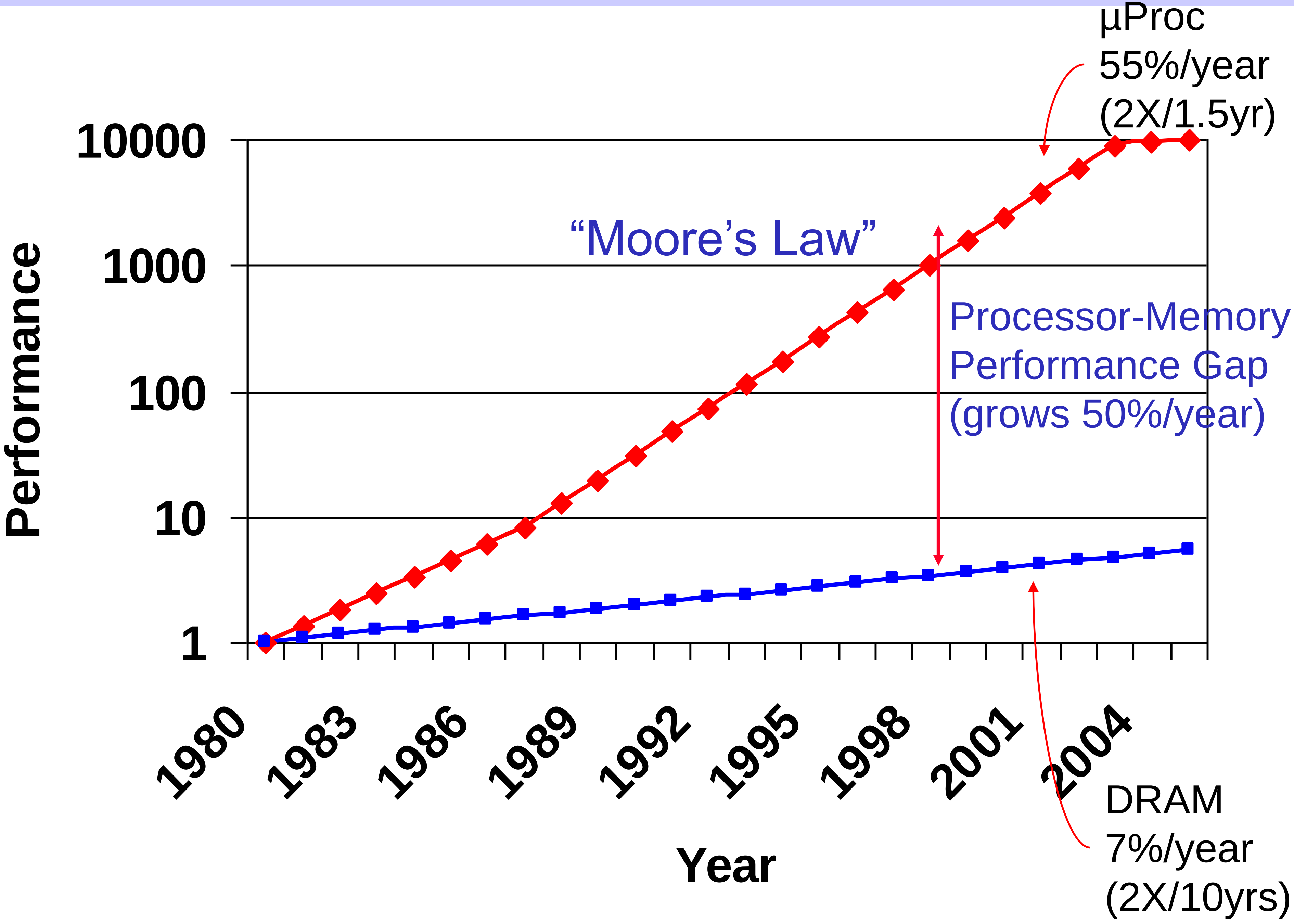
- CPU and RAM are located on the motherboard
- Program Counter register contains address of next instruction to fetch [or for data access, use offset + base from **lw $rd, offset(base)**]
- CPU outputs bits of PC [or base+offset] over the address bus to RAM and asserts control signals (R/W or enable)
- RAM chip receives address and control, looks up the data at that address, and sends the data bits over the data bus to the CPU
- Animation from 2:20 to 5:30

http://www.youtube.com/watch?v=cNN_tTXABUA&t=2m20sec

How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler (programmer?)
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB)
  - by the programmer (files)

Processor-Memory Performance Gap



µProc
55%/year
(2X/1.5yr)

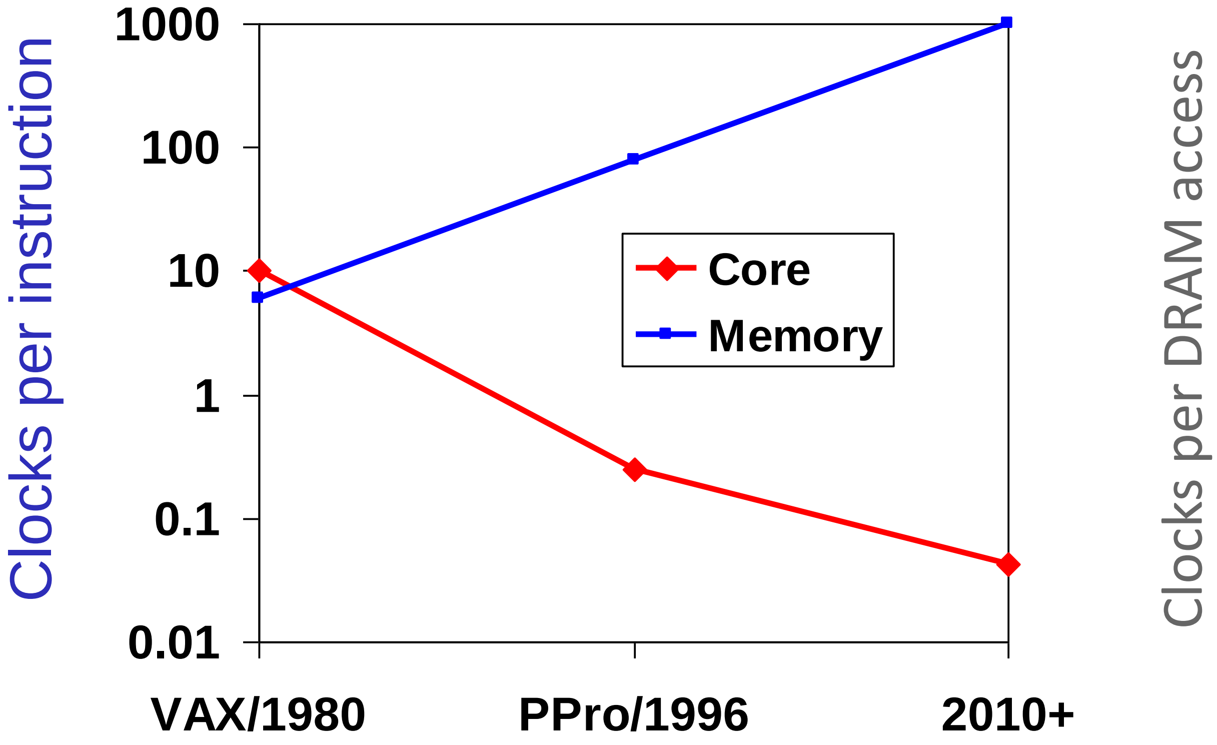
“Moore’s Law”

Processor-Memory
Performance Gap
(grows 50%/year)

DRAM
7%/year
(2X/10yrs)

The “Memory Wall”

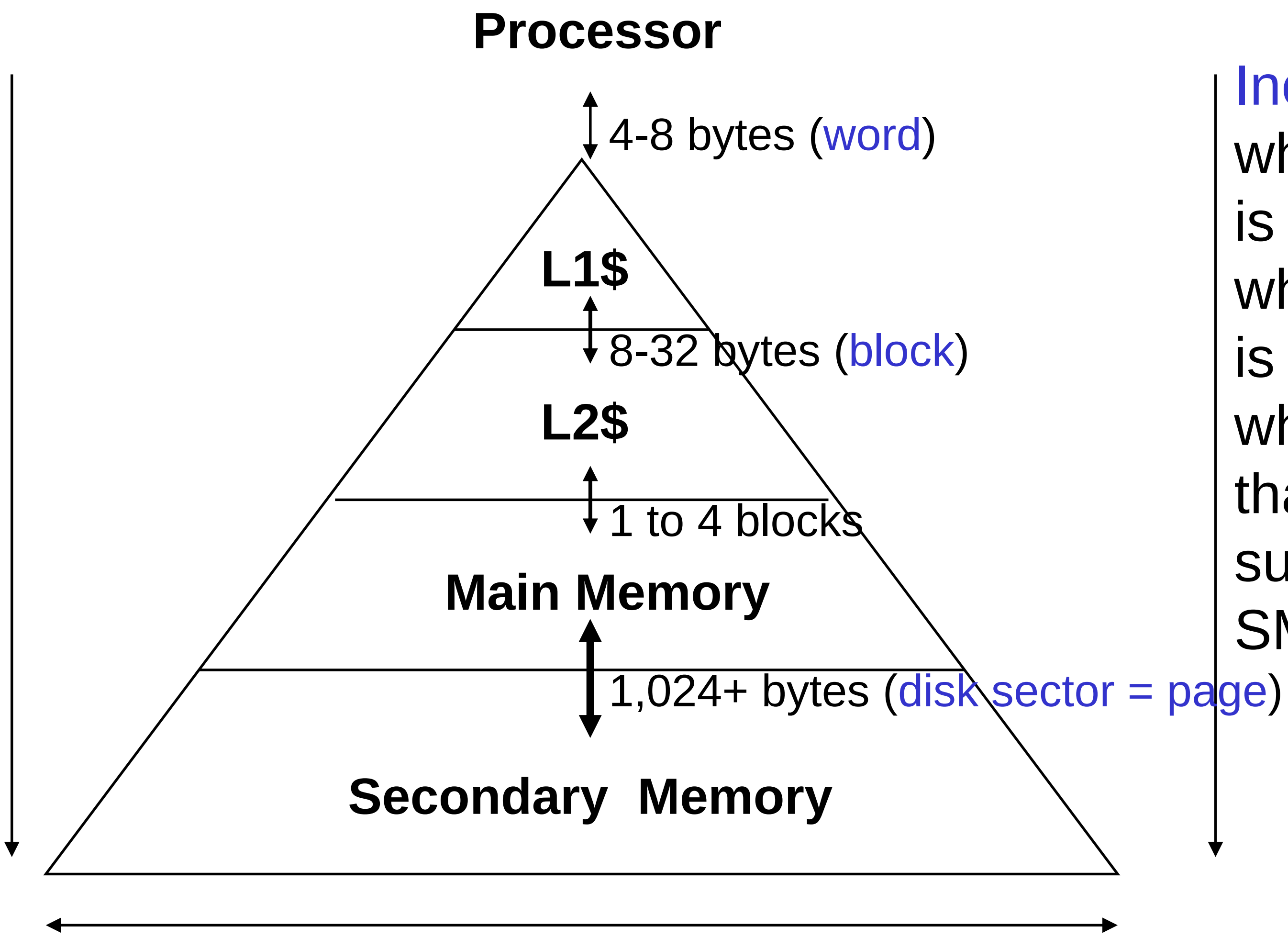
- Processor vs DRAM speed disparity continues to grow



- Good memory hierarchy (cache) design is increasingly important to overall performance

Memory Hierarchy

Increasing distance from the processor in access time



Inclusive – what is in L1$ is a subset of what is in L2$ is a subset of what is in MM that is a subset of is in SM

(Relative) size of the memory at each level

The Memory Hierarchy Goal

- Opposing properties of capacity vs speed:

  “Large memories are slow and fast memories are small.”

- So how do we create a memory that “gives the illusion” of being large, inexpensive, and fast?
  - With hierarchy – leverages locality (explained on next page)
    - Except for recent updates, the data at fastest levels is a subset of data at slower levels
  - With parallelism – multiple bit access between levels
    - word: 32-bits = 4 bytes transferred at a time (Reg ←→ Cache)
    - block: 32 to 128 bytes at a time (Cache ←→ Memory)
- We seek to design hardware such that:
  - taccess = average access time = close to registers
  - $/bit = average cost = close to secondary storage

Principle of Locality

- Temporal Locality (locality in time)
  - If a memory location is referenced then it will tend to be referenced again soon

  ⇒ Keep most recently accessed data items closer to the processor

- Spatial Locality (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

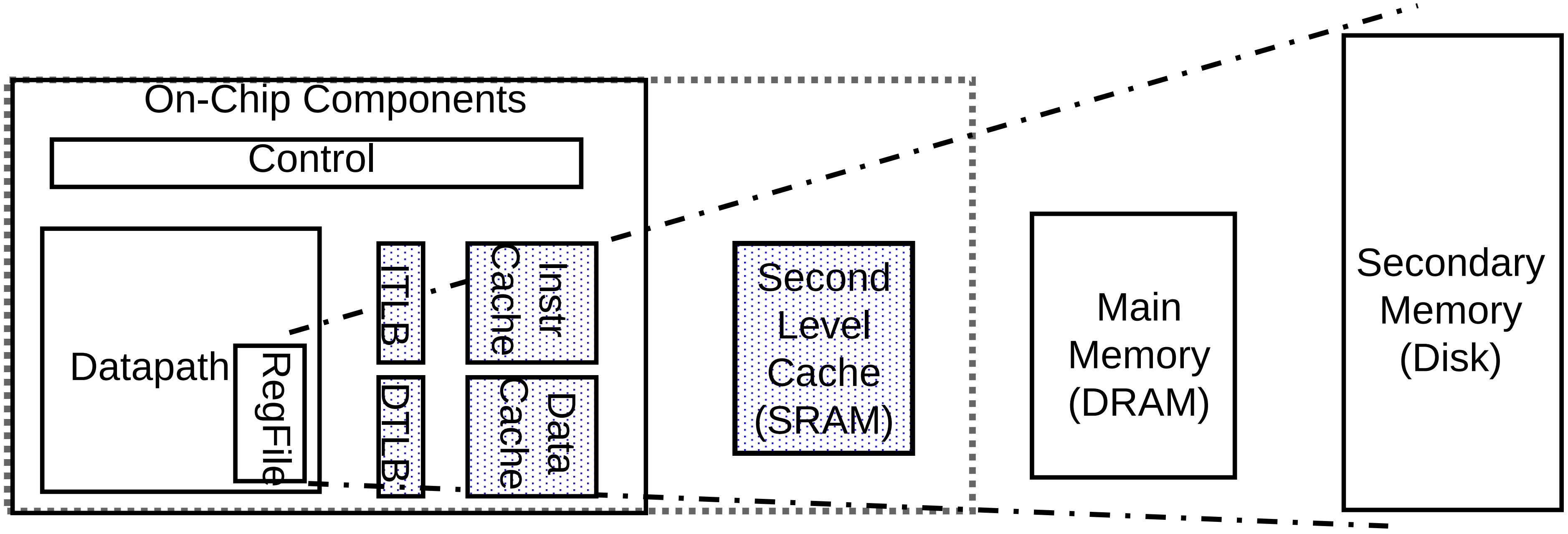
  ⇒ Move blocks consisting of contiguous words closer to the processor

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

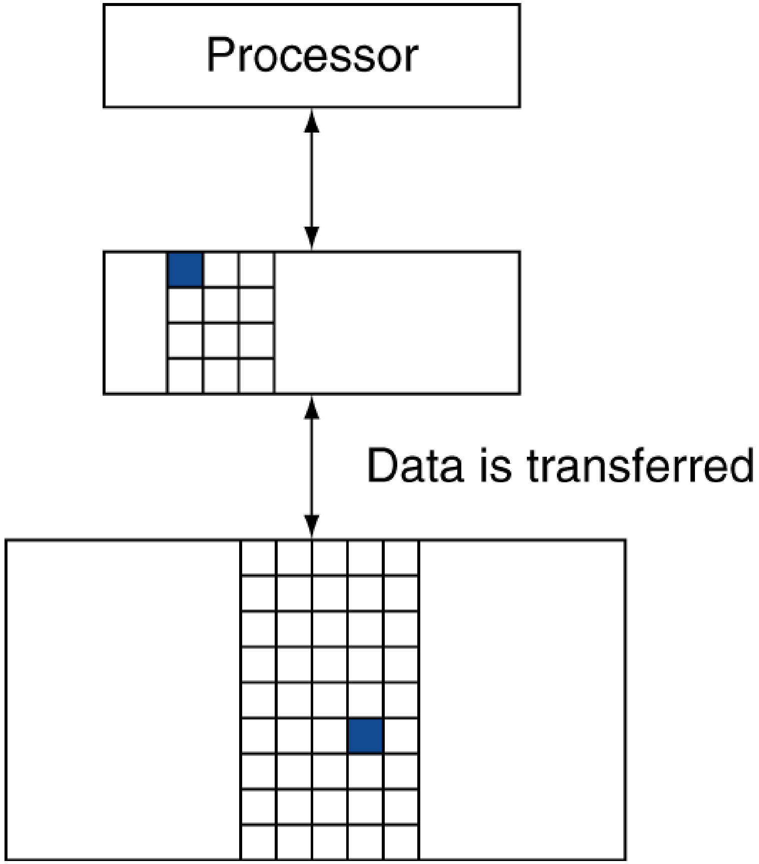
A Typical Memory Hierarchy

Take advantage of the principle of locality to present the user with as much memory as is available in the *cheapest* technology at the speed offered by the *fastest* technology



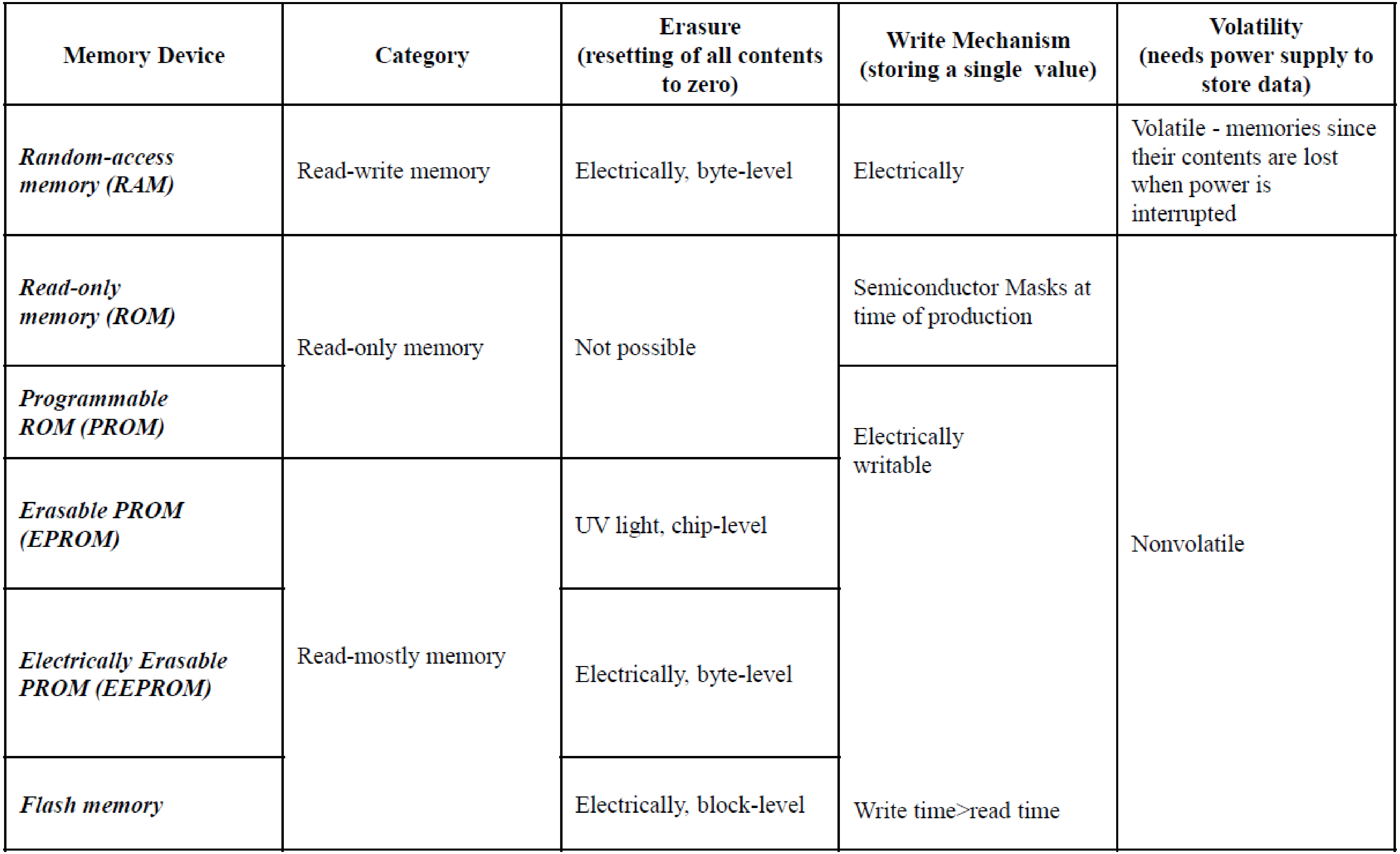
Speed (%cycles):	½’s	1’s	10’s	100’s	>10,000’s
Size (bytes):	100’s	10K’s	M’s	G’s	T’s
Cost:	highest				lowest

Memory Hierarchy Levels



- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses
      = 1 – hit ratio
  - Then accessed data supplied from upper level

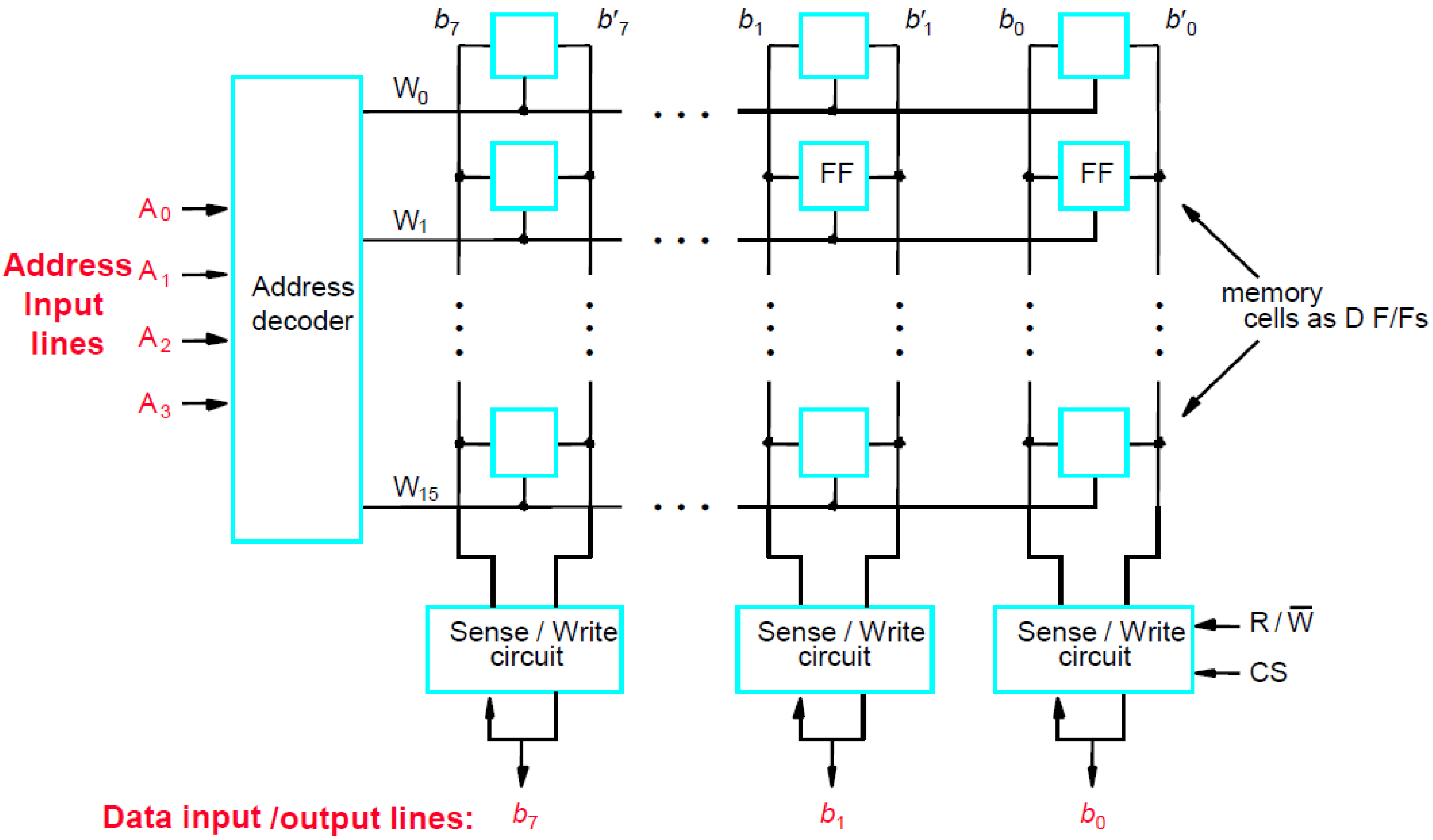
Properties of Memory Devices



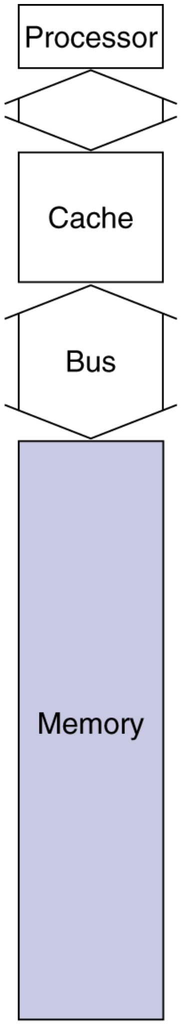
Memory Device	Category	Erasure (resetting of all contents to zero)	Write Mechanism (storing a single value)	Volatility (needs power supply to store data)
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile - memories since their contents are lost when power is interrupted
Read-only memory (ROM)	Read-only memory	Not possible	Semiconductor Masks at time of production	Nonvolatile
Programmable ROM (PROM)			Electrically writable Write time>read time	
Erasable PROM (EPROM)	Read-mostly memory	UV light, chip-level		
Electrically Erasable PROM (EEPROM)		Electrically, byte-level		
Flash memory		Electrically, block-level		

Memory Design – conceptual cell organization

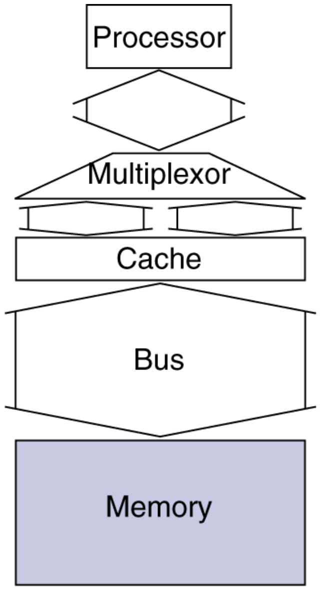
Organization of bit cells in a memory chip.



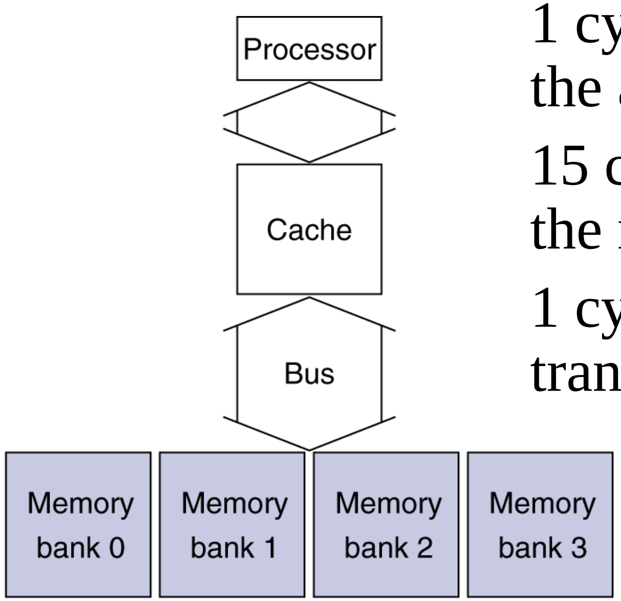
Increasing Memory Bandwidth



a. One-word-wide memory organization



b. Wider memory organization



c. Interleaved memory organization

1 cycle to transmit the address

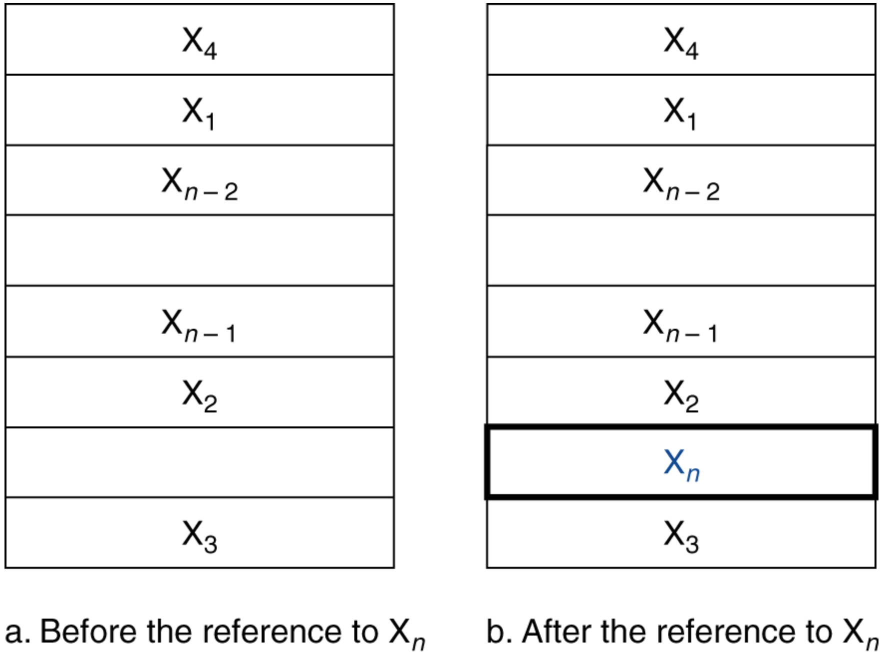
15 cycle to access the memory

1 cycle for data transfer

- 4-word wide memory
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle
- 4-bank interleaved memory
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses $X_1, \ldots, X_{n-1}, X_n$



a. Before the reference to X_n

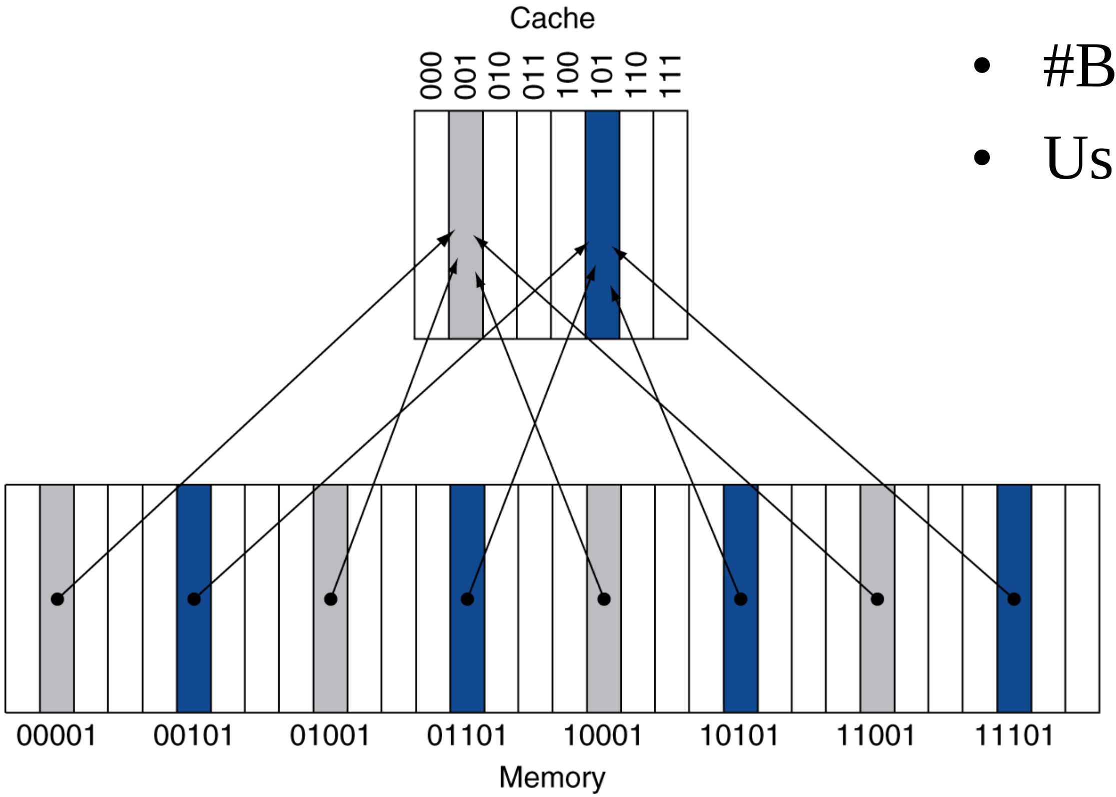
b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?

The cache just before and just after the reference to a word X_n that is not initially in the cache. This reference caused a miss that forces the cache to fetch X_n from memory and inserted into the cache

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the *tag*
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

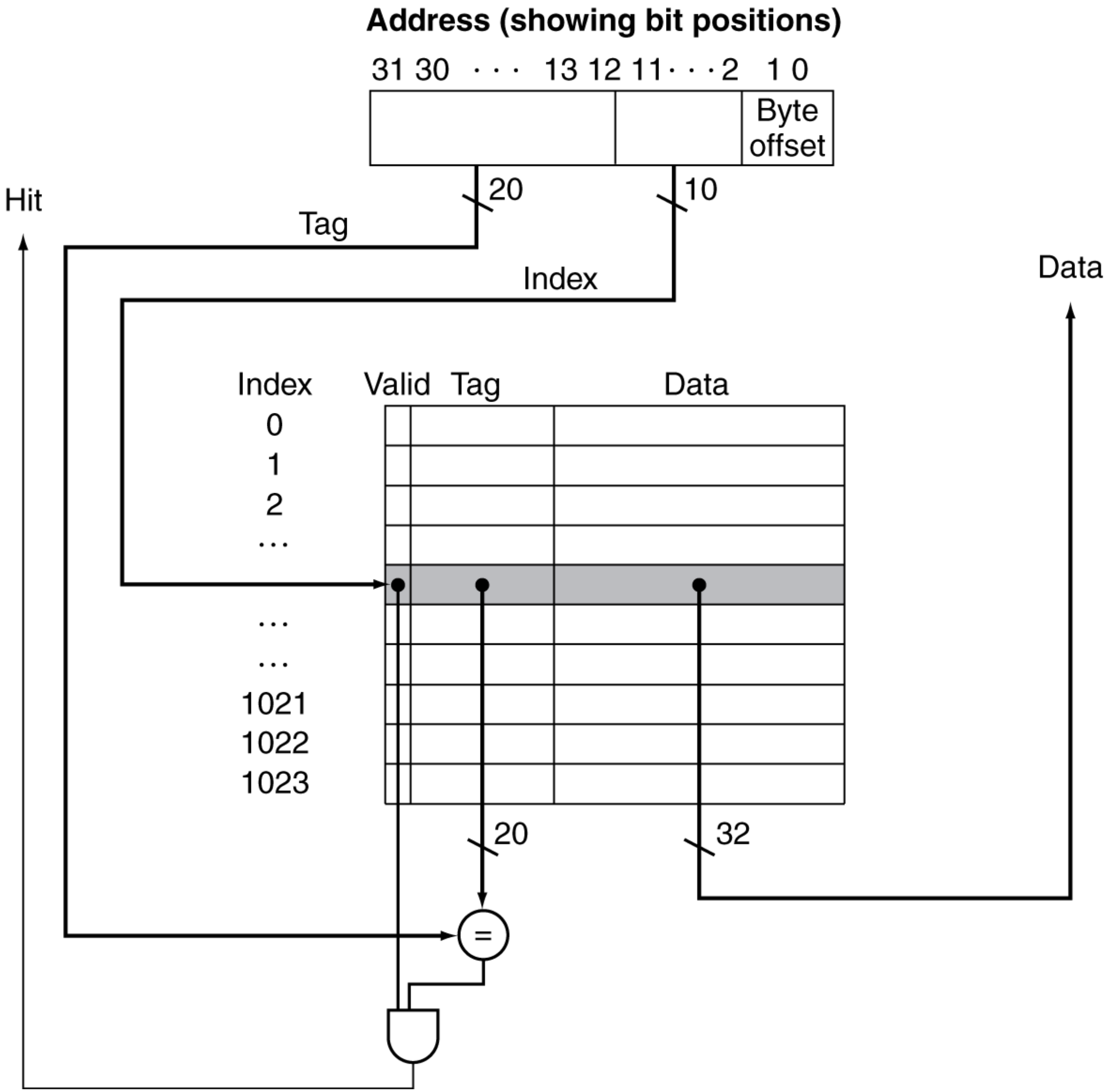
Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Address Subdivision

- 32 bit byte address
- A direct-mapped cache
- The cache size 2^n bocks , n bits used for index
- Block size is 2^m words , (2^{m+2} bytes)



Cache Field Sizes

- The number of bits in a cache includes both the storage for data and for the tags
  - 32-bit byte address
  - For a direct mapped cache with 2^n blocks, *n* bits are used for the index
  - For a block size of 2^m words (2^{m+2} bytes), *m* bits are used to address the word within the block and 2 bits are used to address the byte within the word
- *What is the size of the tag field?*
- The total number of bits in a direct-mapped cache is then

$$2^{n} \text{ x (block size + tag field size + valid field size)}$$

- *Exercise: How many total bits are required for a direct mapped cache with 16KB of data and 4-word blocks assuming a 32-bit address?*