

# Computer Architecture

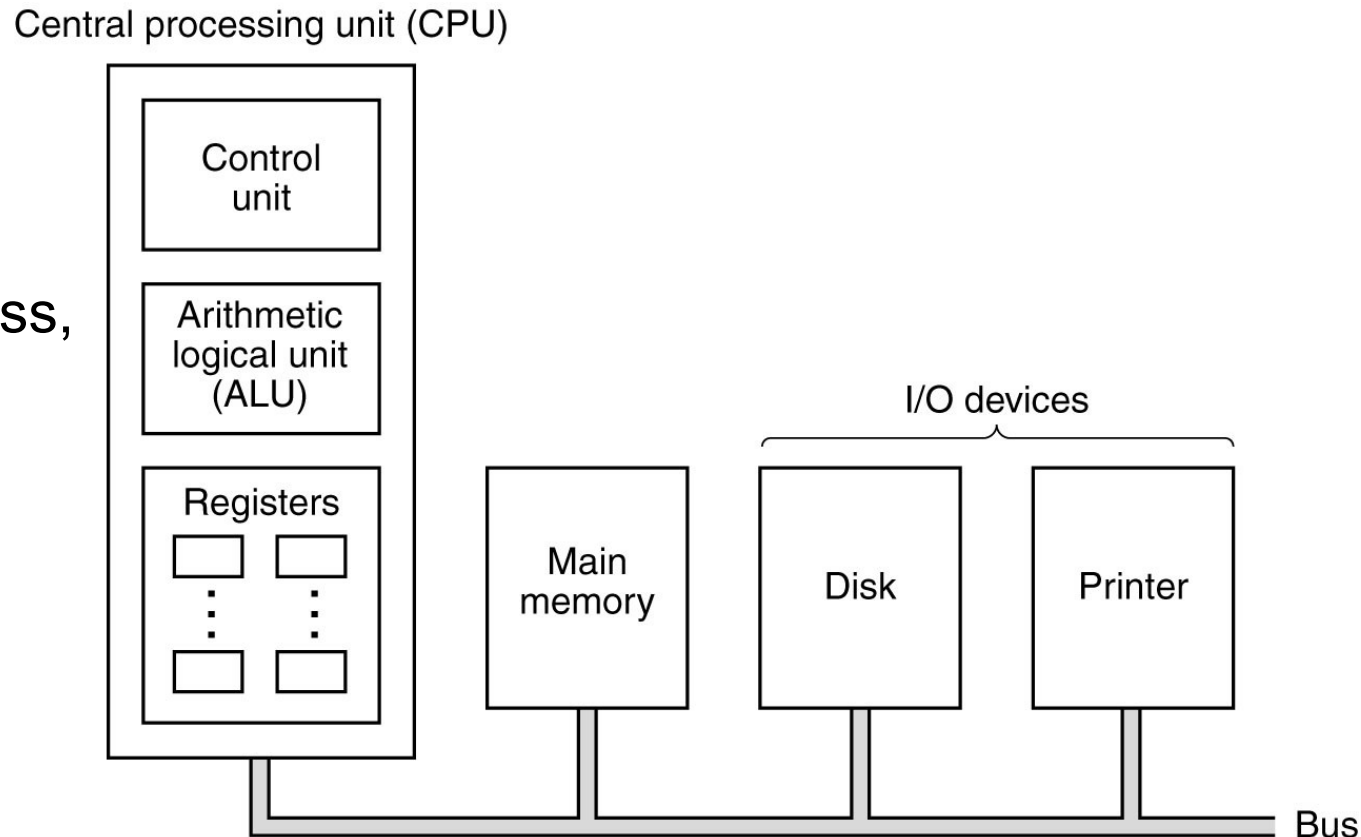
## Ch2 – Computer System Organization

Nguyễn Quốc Đính, FIT – IUH

HCMC, Aug 2015

# Computer System Organization

- Processor
- Memory (bit, byte, address, ECC, cache, RAM, ROM, RAID, CD ...)
- I/O (terminal, printer, modems, ASCII)
- Buses



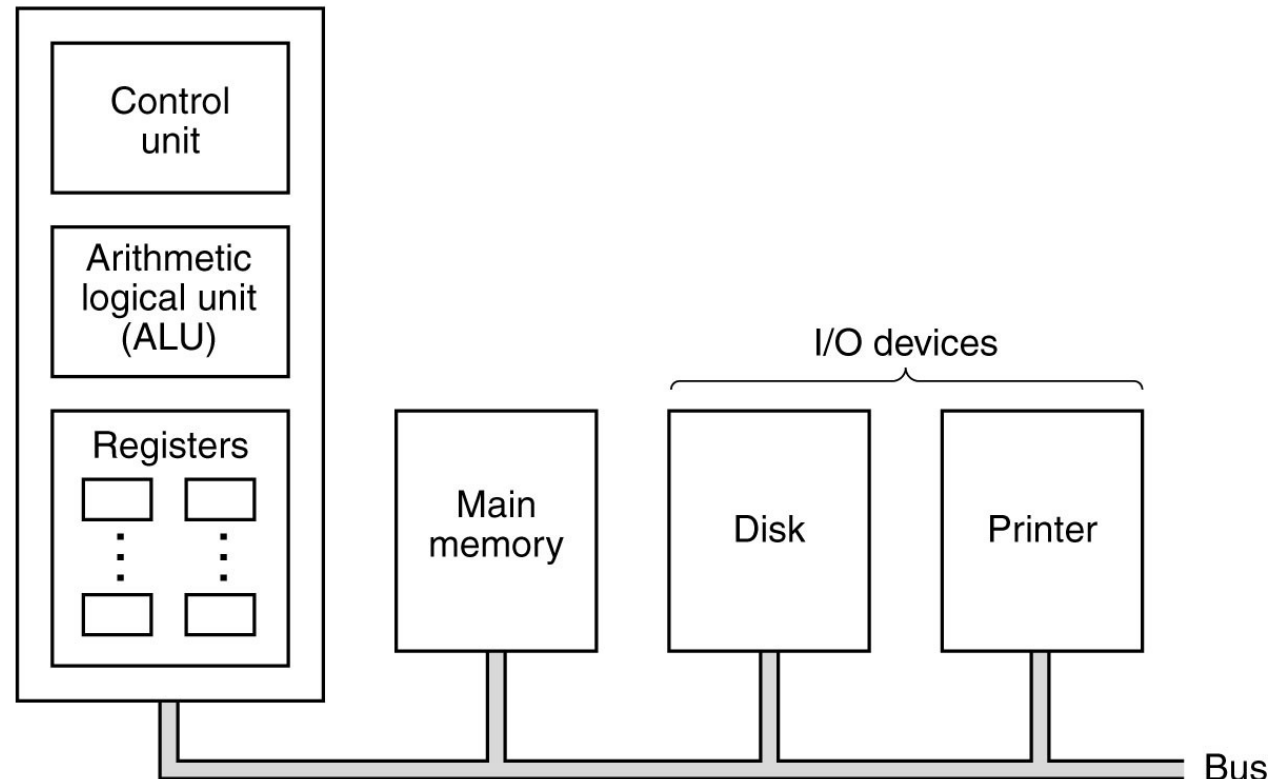
The organization of a simple computer with one CPU and two I/O devices

# Central Processing Unit (CPU)

CPU function is to execute programs stored in the main memory

- fetching instructions
- examining instructions
- executing them one after another

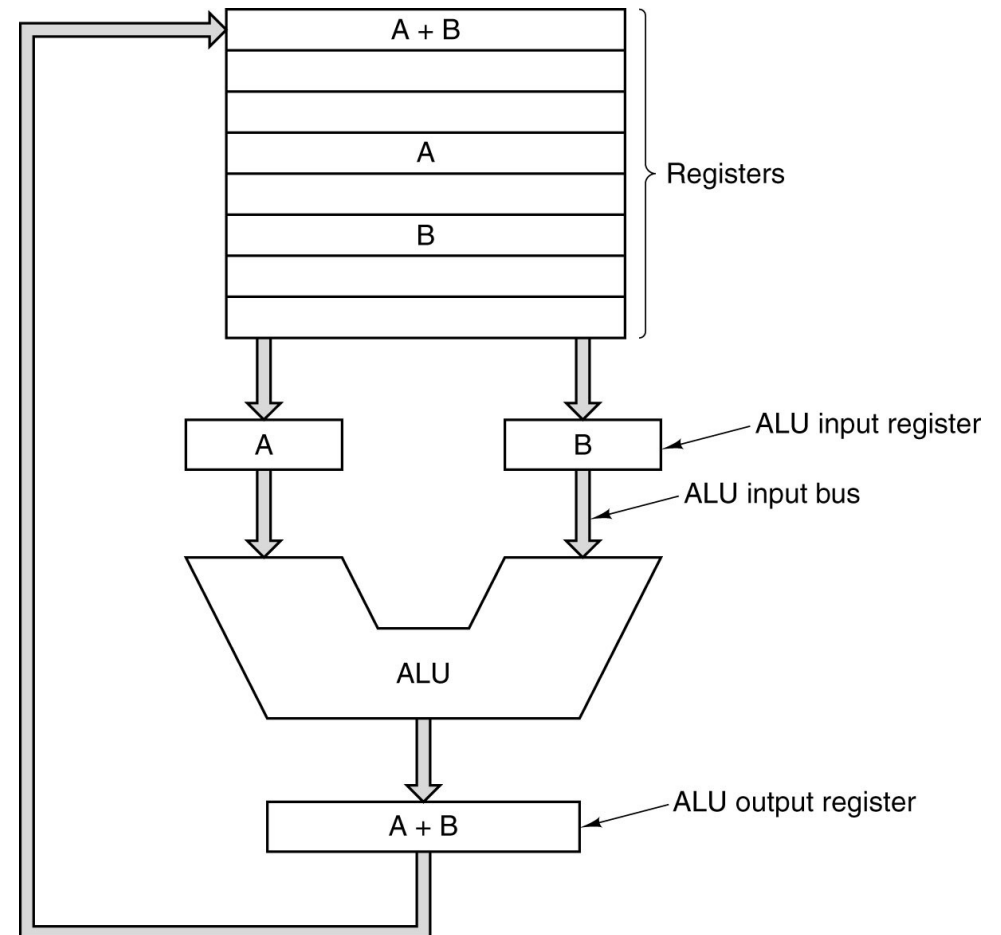
Central processing unit (CPU)



The organization of a simple computer with one CPU and two I/O devices

# CPU Organization

- **Data path** consists of:
  - Registers (No 1 – 32)
  - Arithmetic Logic Unit
  - Buses
- Data path cycle is the heart of most CPU



The data path of a typical Von Neumann machine.

# Instruction Execution Steps

1. Fetch next instruction from memory into instruction register
2. Change program counter to point to next instruction
3. Determine type of instruction just fetched
4. If instructions uses word in memory, determine where it is
5. Fetch the word, if needed, into CPU register
6. Execute the instruction
7. Go to step 1 to begin executing following instruction

*Fetch - decode - execute cycle*

# Interpreter (1)

```
public class Interp {  
    static int PC;           // program counter holds address of next instr  
    static int AC;           // the accumulator, a register for doing arithmetic  
    static int instr;        // a holding register for the current instruction  
    static int instr_type;   // the instruction type (opcode)  
    static int data_loc;     // the address of the data, or -1 if none  
    static int data;         // holds the current operand  
    static boolean run_bit = true; // a bit that can be turned off to halt the machine  
  
    public static void interpret(int memory[], int starting_address) {  
        // This procedure interprets programs for a simple machine with instructions having  
        // one memory operand. The machine has a register AC (accumulator), used for  
        // arithmetic. The ADD instruction adds an integer in memory to the AC, for example.  
        // The interpreter keeps running until the run bit is turned off by the HALT instruction.  
        // The state of a process running on this machine consists of the memory, the  
        // program counter, the run bit, and the AC. The input parameters consist of  
        // of the memory image and the starting address.  
    }  
}
```

...

An interpreter for a simple computer (written in Java).

# Interpreter (2)

```
PC = starting_address;
while (run_bit) {
    instr = memory[PC];           // fetch next instruction into instr
    PC = PC + 1;                 // increment program counter
    instr_type = get_instr_type(instr); // determine instruction type
    data_loc = find_data(instr, instr_type); // locate data (-1 if none)
    if (data_loc >= 0)           // if data_loc is -1, there is no operand
        data = memory[data_loc]; // fetch the data
    execute(instr_type, data);    // execute instruction
}

}

private static int get_instr_type(int addr) { ... }
private static int find_data(int instr, int type) { ... }
private static void execute(int type, int data) { ... }
}
```

An interpreter for a simple computer (written in Java).

# CISC

(Complex Instruction Set Computer)

- Origin:
  - Initial instruction set and computer architecture
  - Hardware designed to facilitate software languages and programming
- Instruction
  - Complex, defined based on software and/or application requests
- CPI (cycles per instruction)
  - Increase with instruction complexity



# CISC

(Complex Instruction Set Computer) – cont'd

- Classic CISC processor
  - Motorola 68000
  - Intel 8088, 8086, 80286
- Early CISC got out of hand
  - How to get more accomplished faster? One way is to reduce CPI
  - $< 10\%$  of instruction execute 90% of the time ... results in wasted hardware and large size

# The architecture fix to early CISC

- To *reduce the CPI*, the instructions had to get easier and faster
  - CPI = 1 goal
  - Instructions should be easily decoded (fewer, simpler instruction)
- *Memory accesses* are slow and a major processor speed impediment
  - Add more register for fewer memory access
  - Force to use register by limiting memory access to load and store

Result in a "Reduced Instruction Set Computer" (RISC)

# RISC

Reduced Instruction Set Computer

- Origin:
  - 1980 Berkeley: David Patterson and Carlo Sequin
  - 1981 Stanford: John Hennessy
- Addressing
  - Load/Store
- Instruction:
  - Fixed length (typically word length), minimal format

# Design Principle for Modern Computers

(RISC design principle)

- All instructions directly executed by hardware
  - Eliminate a level of interpretation provides high speed for most instructions
- Maximize rate at which instructions are issued
  - Increase number of instructions per second (MIPS)
  - Parallelism can play a major role in improving performance

# Design Principle for Modern Computers

(RISC design principle) – cont'd

- Instructions should be easy to decode
  - Making instruction regular, fixed length.
  - Fewer different formats for instruction, the better
- Only loads, stores should reference memory
  - Most instruction come from/return to CPU registers
  - Only LOAD, STORE instructions should reference memory
- Provide plenty of registers
  - At least 32 registers

# If RISC was so great, why did CISC still exist?

- Backward compatibility
- CISC instructions set with RISC micro-operation execution cores

# Speed up CPU computing capacity?

- Increasing clock speed.
  - Up-bound?
- Parallelism
  - Instruction-level parallelism (ILP)
  - Processor-level parallelism (PLP)

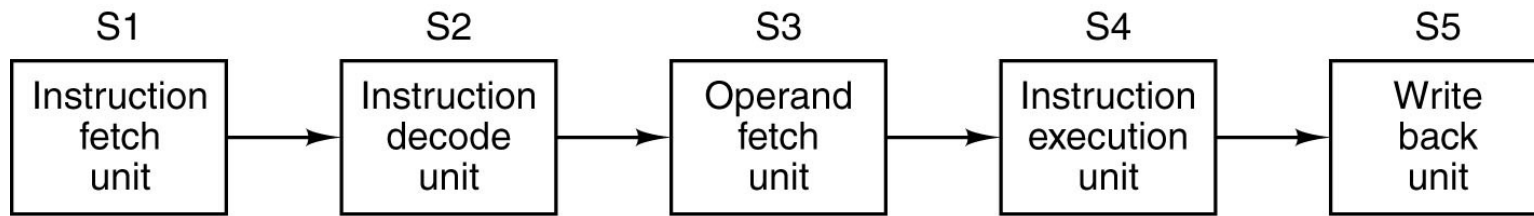
# Speed up CPU computing capacity?

- Increasing clock speed. Up-bound?
- Parallelism
  - Instruction-level parallelism (ILP)
  - Processor-level parallelism (PLP)

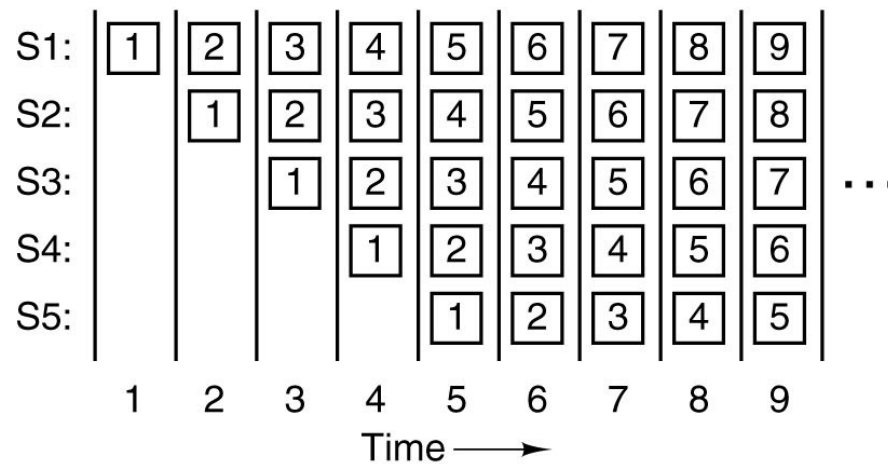


# Instruction-Level Parallelism

- a) A five-stage pipeline
- b) The state of each stage as a function of time. Nine clock cycles are illustrated



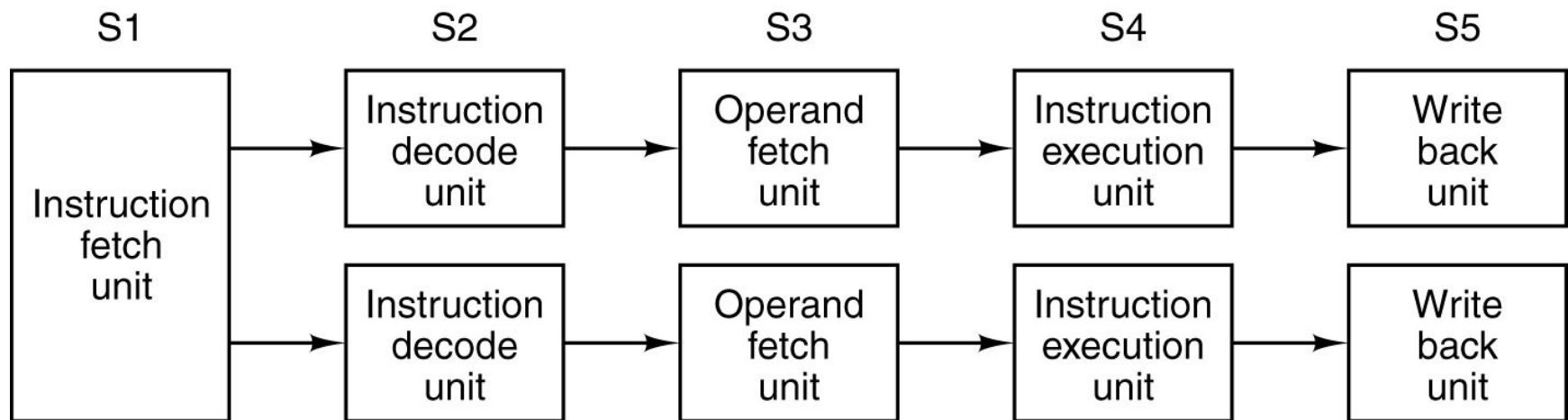
(a)



(b)

# Dual Pipeline

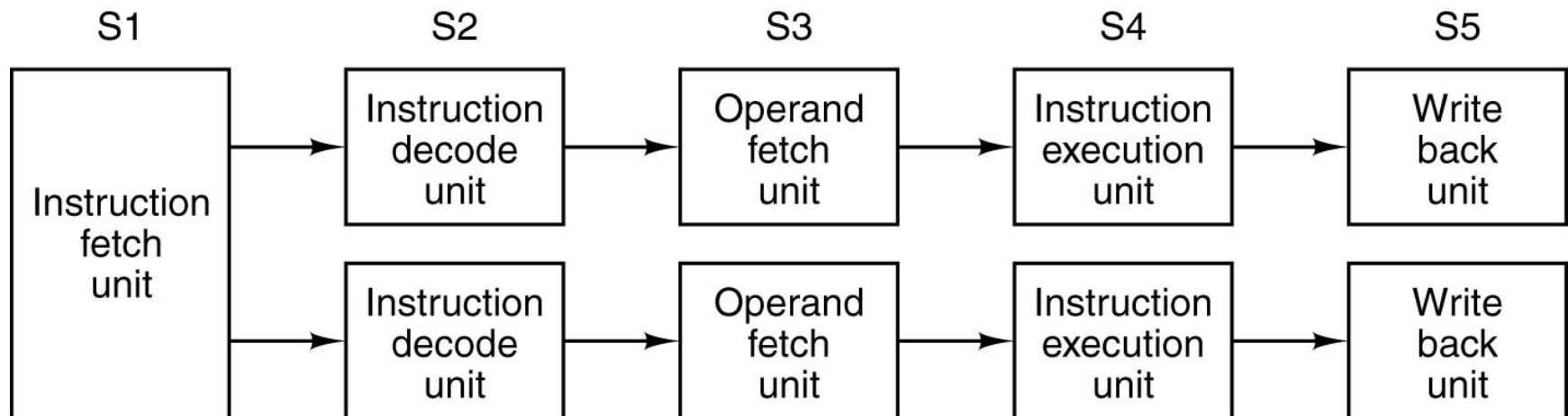
- If one pipeline is good, then surely two pipelines are better
- Pentium processor has two five-stage pipelines



Dual five-stage pipelines with a common instruction fetch unit

# Dual Pipeline

- Pentium two-issue superscalar
  - U-pipeline (main): execute all Pentium instructions
  - V-pipeline (sub): execute simple Pentium instr. when there are no conflicts



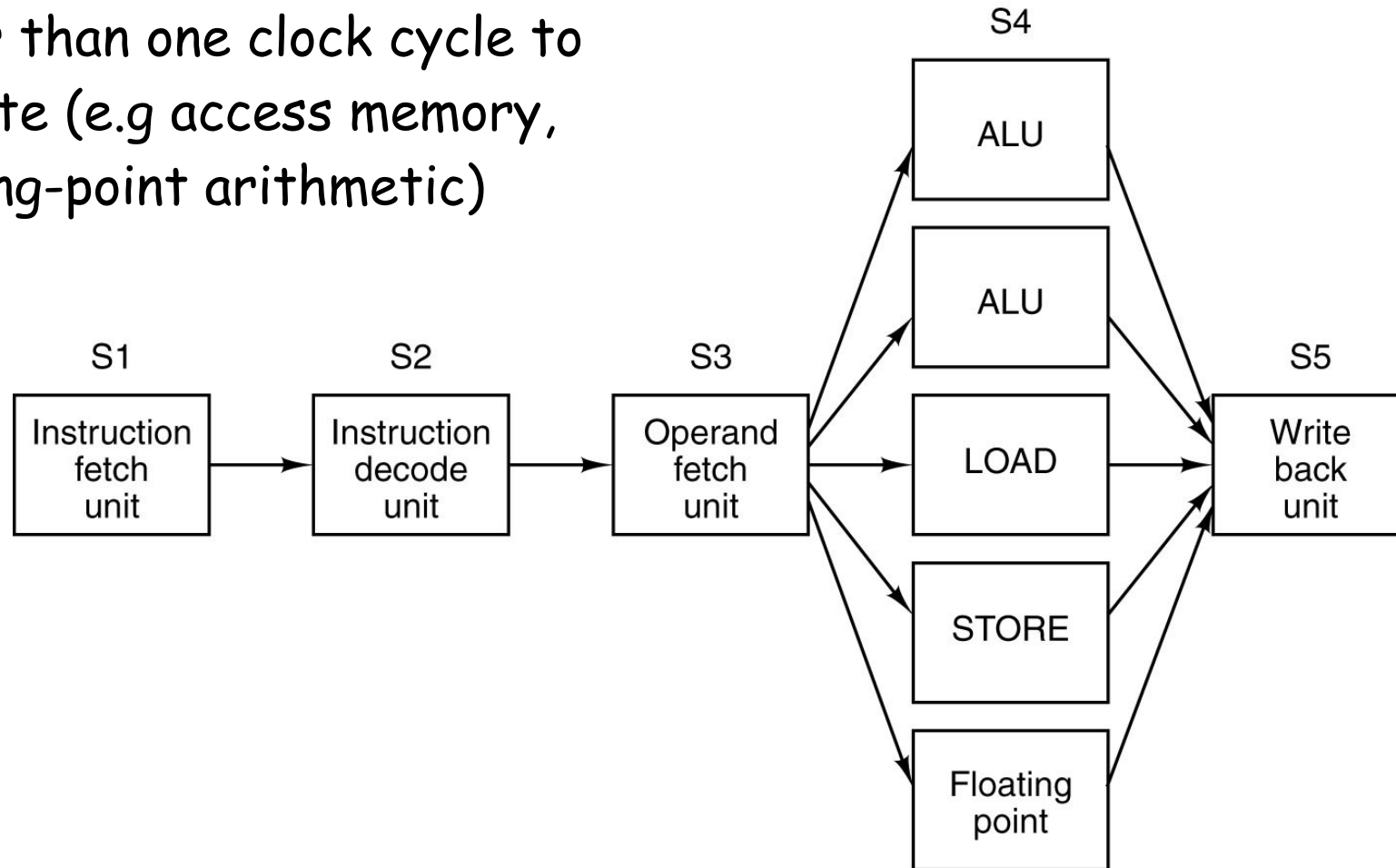
Dual five-stage pipelines with a common instruction fetch unit

# Multi-Pipeline?

- More than 2 pipeline is so complicated
- Different approach:
  - Instruction issue rate is much higher than the execution rate, then the workload spread across a collection of functional units
  - CPU with 100ns clock  $\leftrightarrow$  CPU with 400ns clock that issues 4 instructions per cycle.

# Superscalar Processor

Stage 4 have functional units take longer than one clock cycle to execute (e.g access memory, floating-point arithmetic)

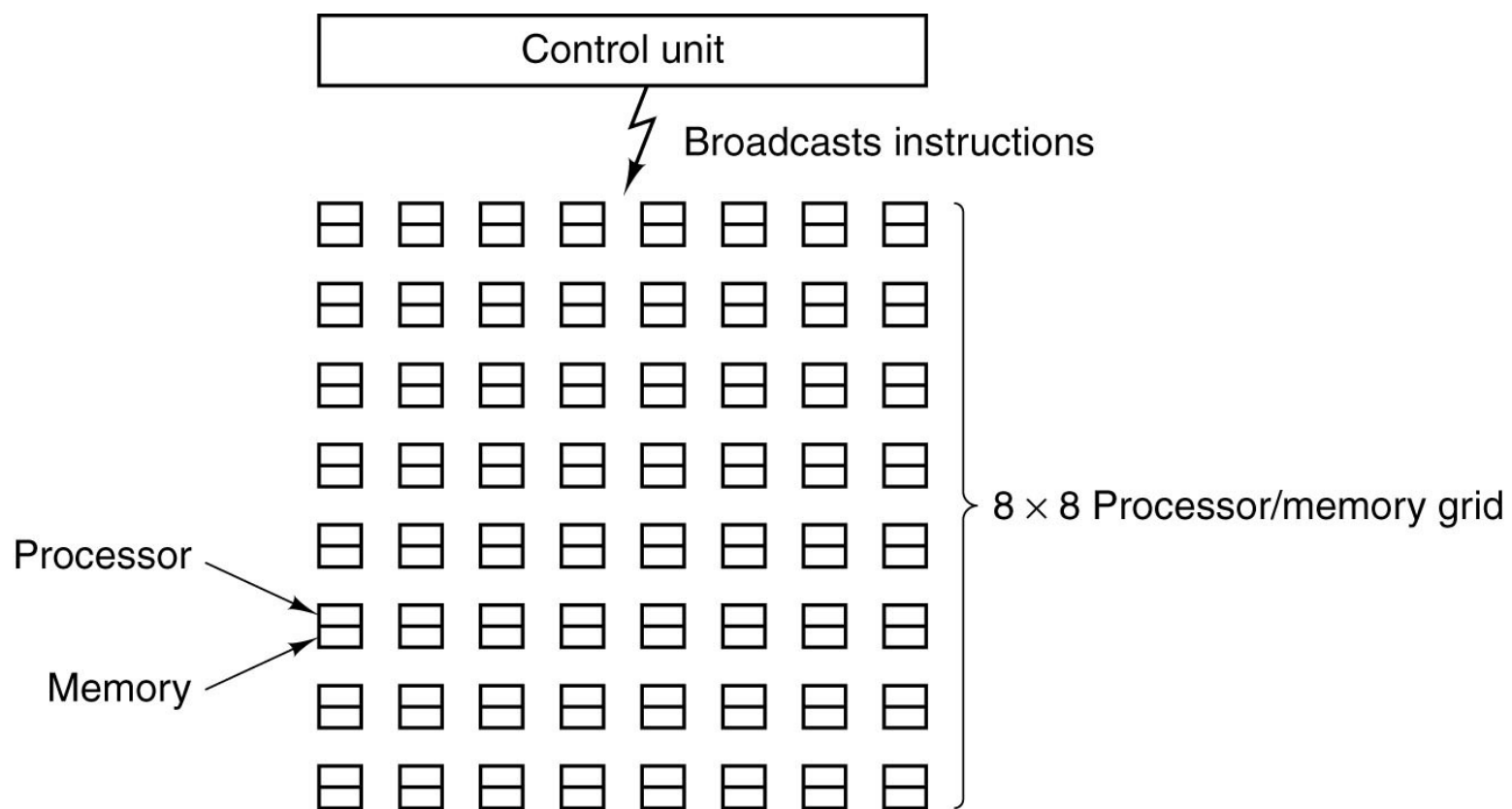


A superscalar processor with five functional units  
Pentium II concept

# Speed up CPU computing capacity?

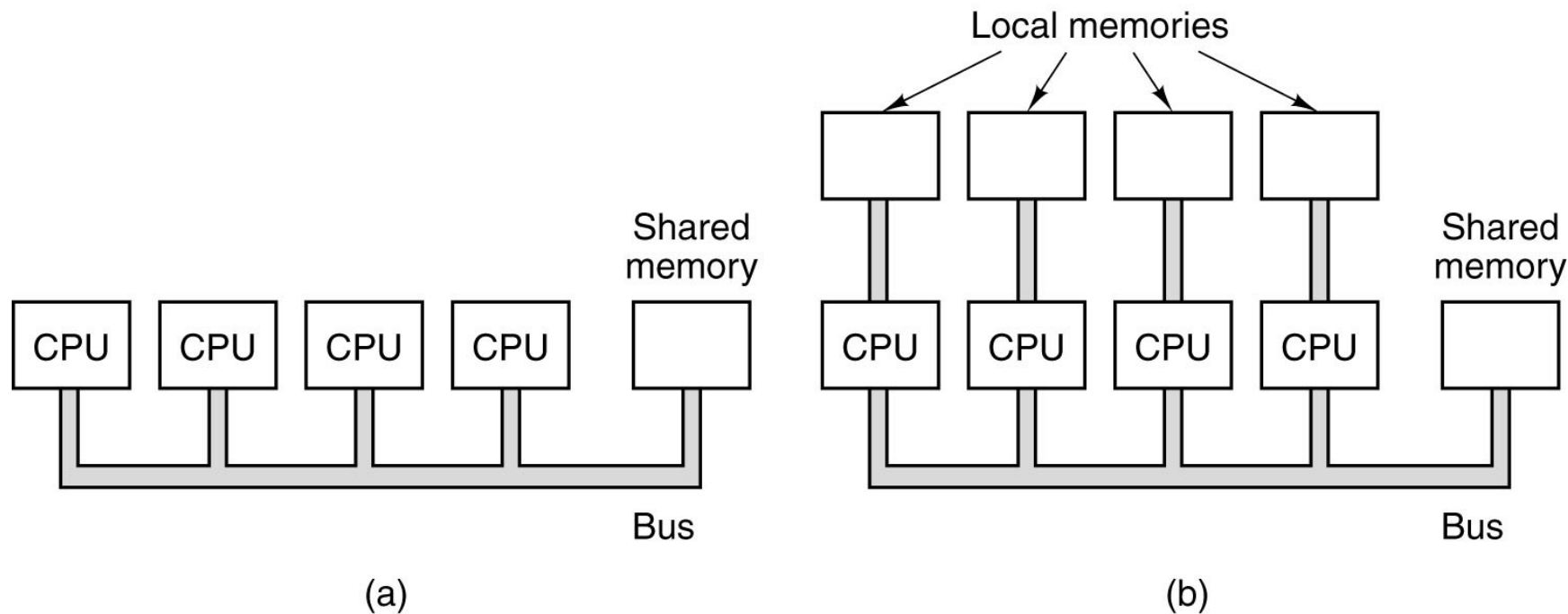
- Increasing clock speed. Up-bound?
- Parallelism
  - Instruction-level parallelism (ILP)
  - Processor-level parallelism (PLP)

# Processor-level Parallelism (1)



An array of processor of the ILLIAC IV type  
(first array processor at UIUC 1972)

# Processor-level Parallelism (2)



a) A single-bus multiprocessor.

b) A multicomputer with local memories.



# Primary Memory

## Memory Addresses

Given  **$2^{10}$  bits**, how is the memory organized:

1024 1-bit cells

128 8-bit bytes

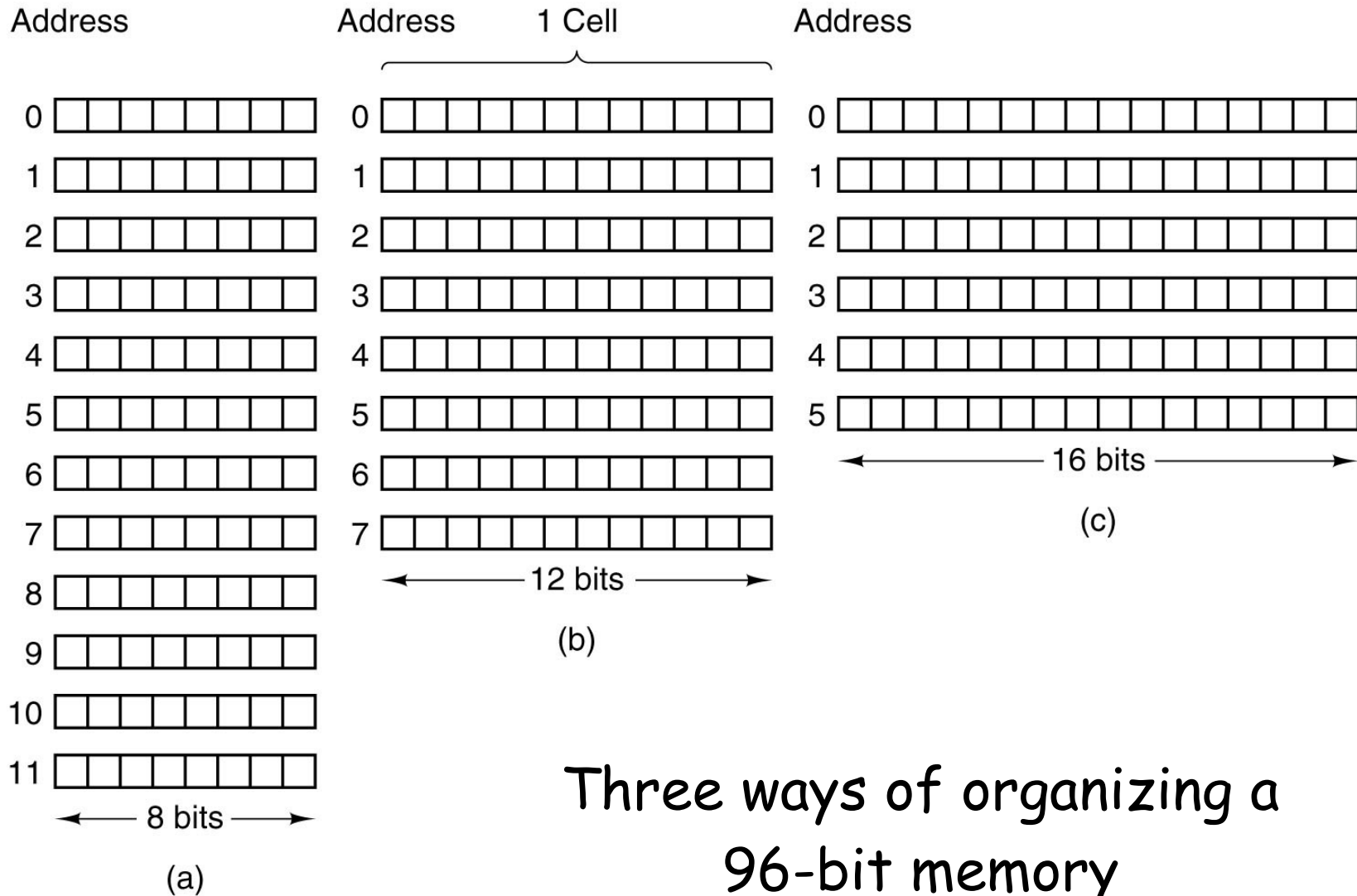
64 16-bit words

32 32-bit words

Each could have a different means of addressing:  
bit-level, byte-level, word(?) -level ...


# Primary Memory

## Memory Addresses



# Primary Memory

## Memory Addresses



Computer	Bits/cell
Burroughs B1700	1
IBM PC	8
DEC PDP-8	12
IBM 1130	16
DEC PDP-15	18
XDS 940	24
Electrologica X8	27
XDS Sigma 9	32
Honeywell 6180	36
CDC 3600	48
CDC Cyber	60

A memory with  $2^{12}$  cells of 8 bits each and a memory with  $2^{12}$  cells of 64 bits each need 12-bit addresses

Number of bits per cell for some historically interesting commercial computers

# Primary Memory

## Memory Addresses

- Note:
  - Most PC based processors use byte-addressing
  - Others use word-addressing, particularly DSP and microcontroller

A byte-addressable 32-bit (memory addresses) computer can address  $2^{32} = 4\text{GB}$

Intel 8086 (16-bit) supported 20-bit addressing, allowing it to access 1 MiB.

# Primary Memory

## Memory Addresses

- When a modern computer reads from or writes to a memory address, it will do this in word sized chunks
- Byte aligned storage is typically required
  - 16-bit words start at multiple of 2
  - 32-bit words start at multiple of 4
  - **If not?**

# Alignment

```
struct mydata {  
    int a;        //4  
    char b;       //1  
    int c;        //4  
    char d;       //1  
    char e;       //1  
    float t;      //4  
};
```

```
cout<<"Size = " << sizeof(mydata) <<endl;?
```

```
struct mydata2 {  
    double x;     //4  
    char n[1];    //  
};
```

```
cout<<"Size = " << sizeof(mydata2) <<endl; ?
```

- Padding

`padding = (align - (offset mod align)) mod align`

`new offset = offset + padding`

- Typical padding

A char (one byte) will be 1-byte aligned.

A short (two bytes) will be 2-byte aligned.

An int (four bytes) will be 4-byte aligned.

A long (four bytes) will be 4-byte aligned.

A float (four bytes) will be 4-byte aligned.

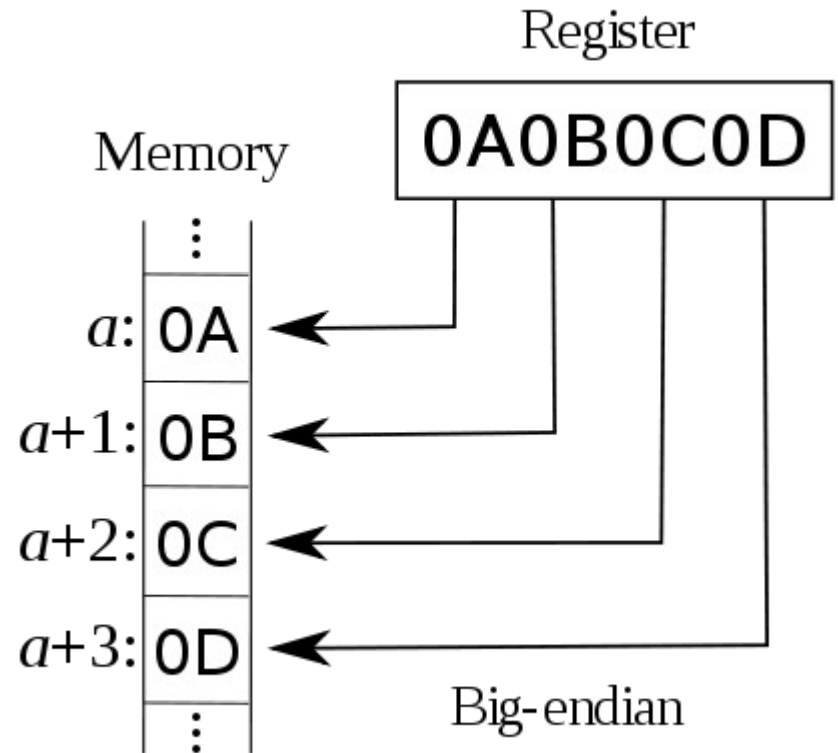
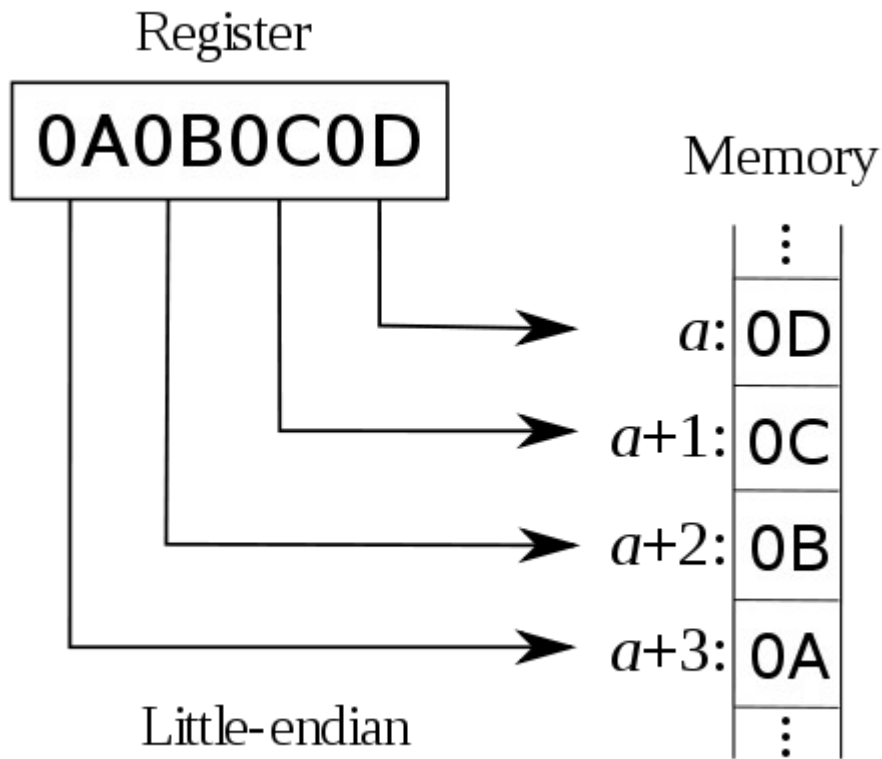
A double (eight bytes) will be 8-byte aligned on Windows and 4-byte aligned on Linux (8-byte with `-malign-double` compile time option).

Any pointer (four bytes) will be 4-byte aligned. (e.g.: `char*`, `int*`)

See [http://en.wikipedia.org/wiki/Data\\_structure\\_alignment](http://en.wikipedia.org/wiki/Data_structure_alignment)

# Byte Ordering

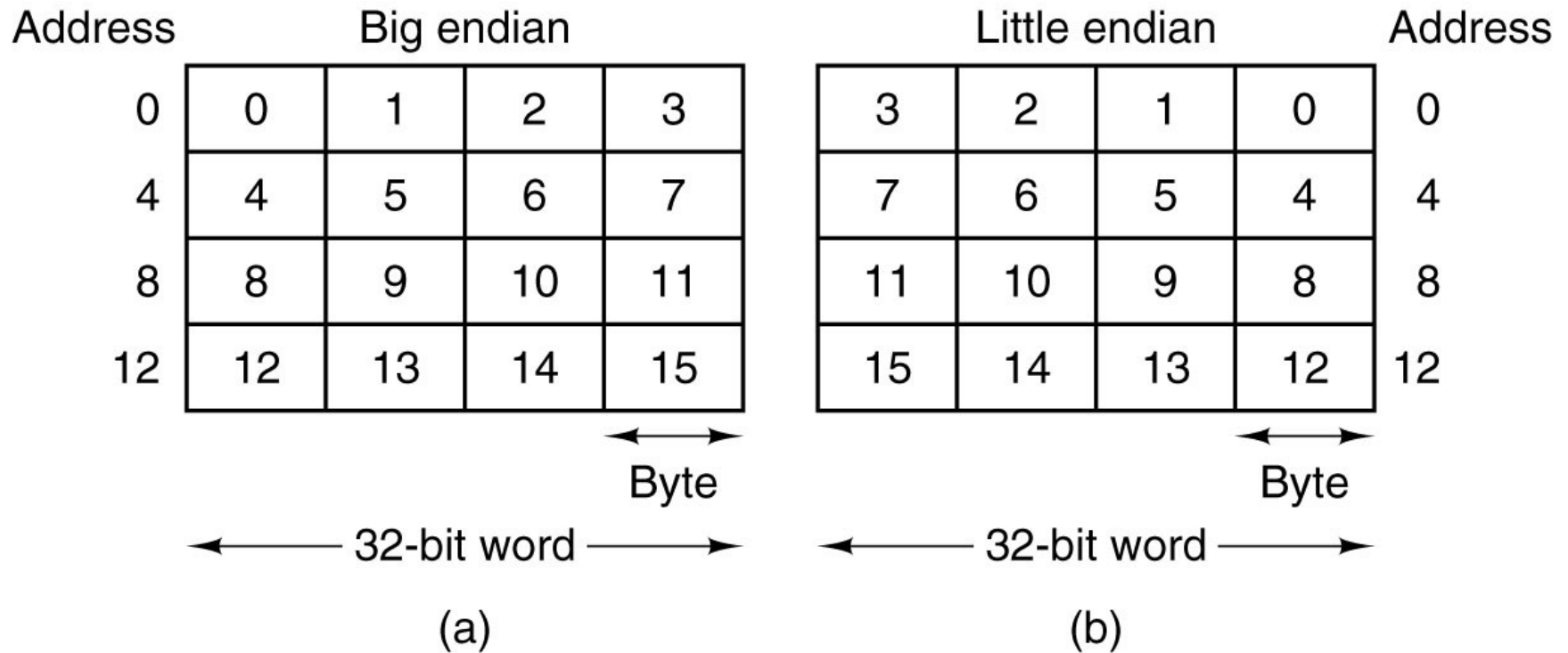
## Big endian vs Little endian





# Byte Ordering

## Big endian vs Little endian



(a) Big endian memory    (b) Little endian memory

# Byte Ordering

Big endian vs Little endian

Question: illustrate following struct memory organization under big-endian and little-endian machine

```
struct {  
    char c    // ("A")  
    int i     // (21)  
}
```

# Byte Ordering

Big endian vs Little endian

Question: illustrate following struct memory organization under big-endian and little-endian machine

```
(char*) JIM SMITH  
(int) 21  
(int) 260
```

# Byte Ordering

## Big endian vs Little endian

Name: (Char\*) JIM SMITH  
Age: (int) 21  
Department: (int) 260

Big endian					Little endian				
0	J	I	M			M	I	J	0
4	S	M	I	T	T	I	M	S	4
8	H	0	0	0	0	0	0	H	8
12	0	0	0	21	0	0	0	21	12
16	0	0	1	4	0	0	1	4	16
(a)					(b)				

(a) Big endian memory  
(b) Little endian memory

# Problem

Write a C/C++ program to determine the current running computer is big-endian or little-endian

# Why is this really importance?

Big endian	Little endian
SPARC (Sun Micro) Motorola 68000 PowerPC	Intel IBM(typically)

“Well-known processor architectures that use the little-endian format include: x86 (including x86-64), 6502 (including 65802, 65C816), Z80 (including Z180, eZ80 etc.), MCS-48, 8051, DEC Alpha, Altera Nios, Atmel AVR, SuperH, VAX, and, largely, PDP-11.”

“Well-known processors that use the big-endian format include: Motorola 6800 and 68k, Xilinx Microblaze, IBM POWER, and System/360 and its successors such as System/370, ESA/390, and z/Architecture.”

Bi-endian: ARM versions 3 and above, PowerPC, Alpha, SPARC V9, MIPS, PA-RISC and IA-64

# Error Correcting Codes (1)

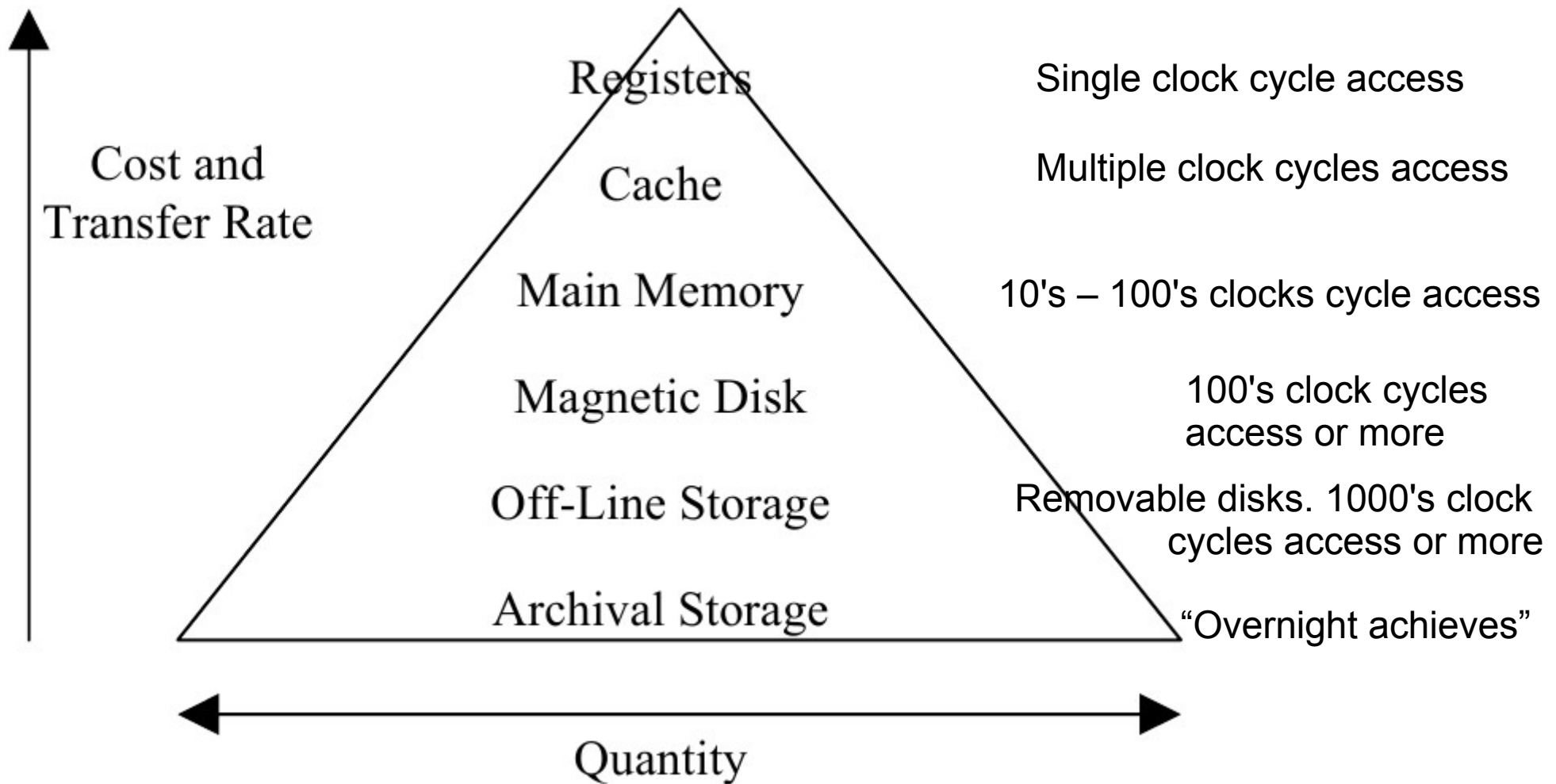
- The bits in a data transfer can become corrupted.
  - More likely with modem or wireless transmission
  - Bus can have bit errors!
- If you expect there to be a high probability that bits in a byte/word will be corrupted, encode!

# Error Correcting Codes (2)

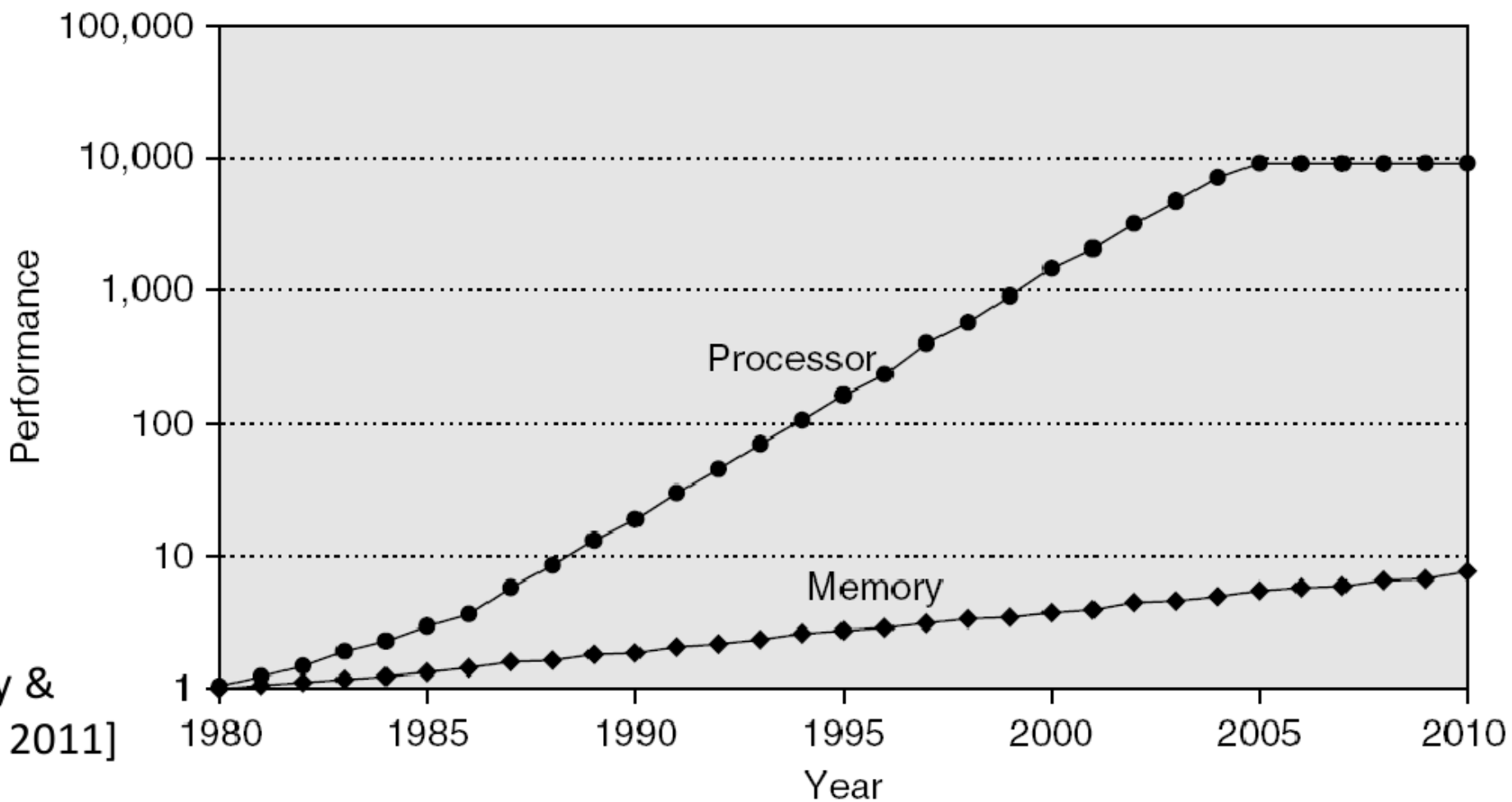
- Parity Bit
  - For EVEN parity, the sum of all the bits must be even ( $0110 = 0+1+1+0=2$ )
  - For ODD parity, the sum of all the bits must be odd ( $1011 = 1+0+1+1=3$ )
  - One more bit is added to the binary data to make the “bit field” EVEN or ODD
- Notes
  - It works really well with 7-bit ASCII to create an 8-bit word !
  - Also 9-bit memory ICs, used for 8-bits and parity



# Memory Hierarchy Pyramid



# Processor-DRAM Latency Gap



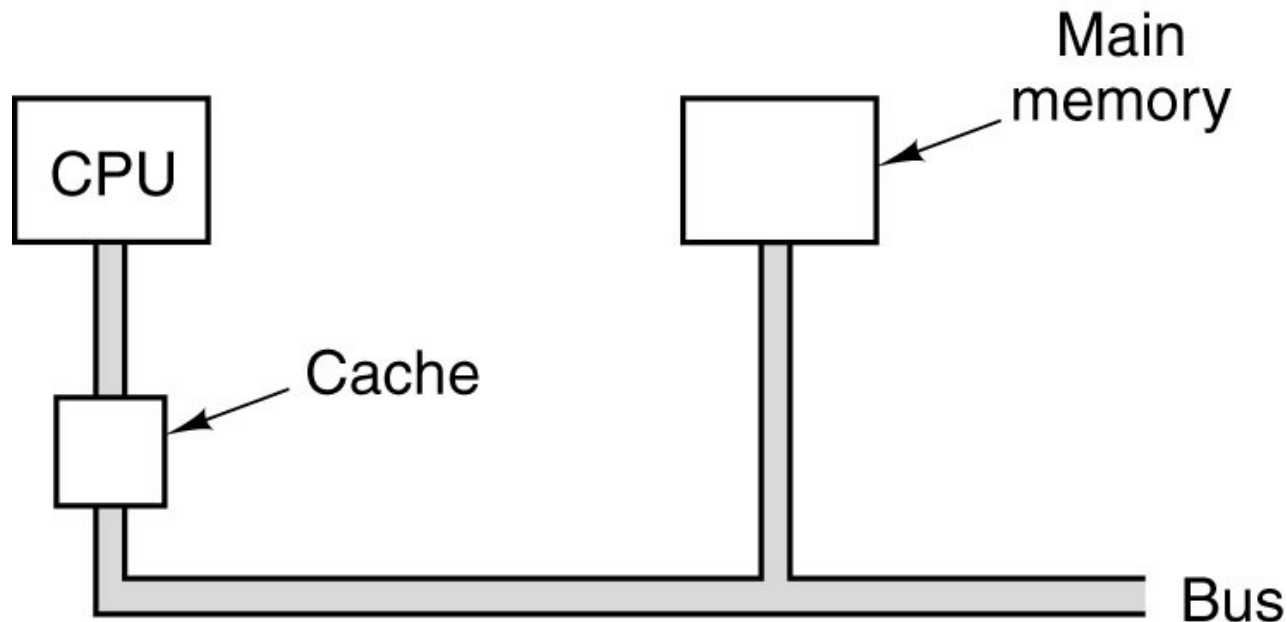
[Hennessy &  
Patterson 2011]

# Cache Memory

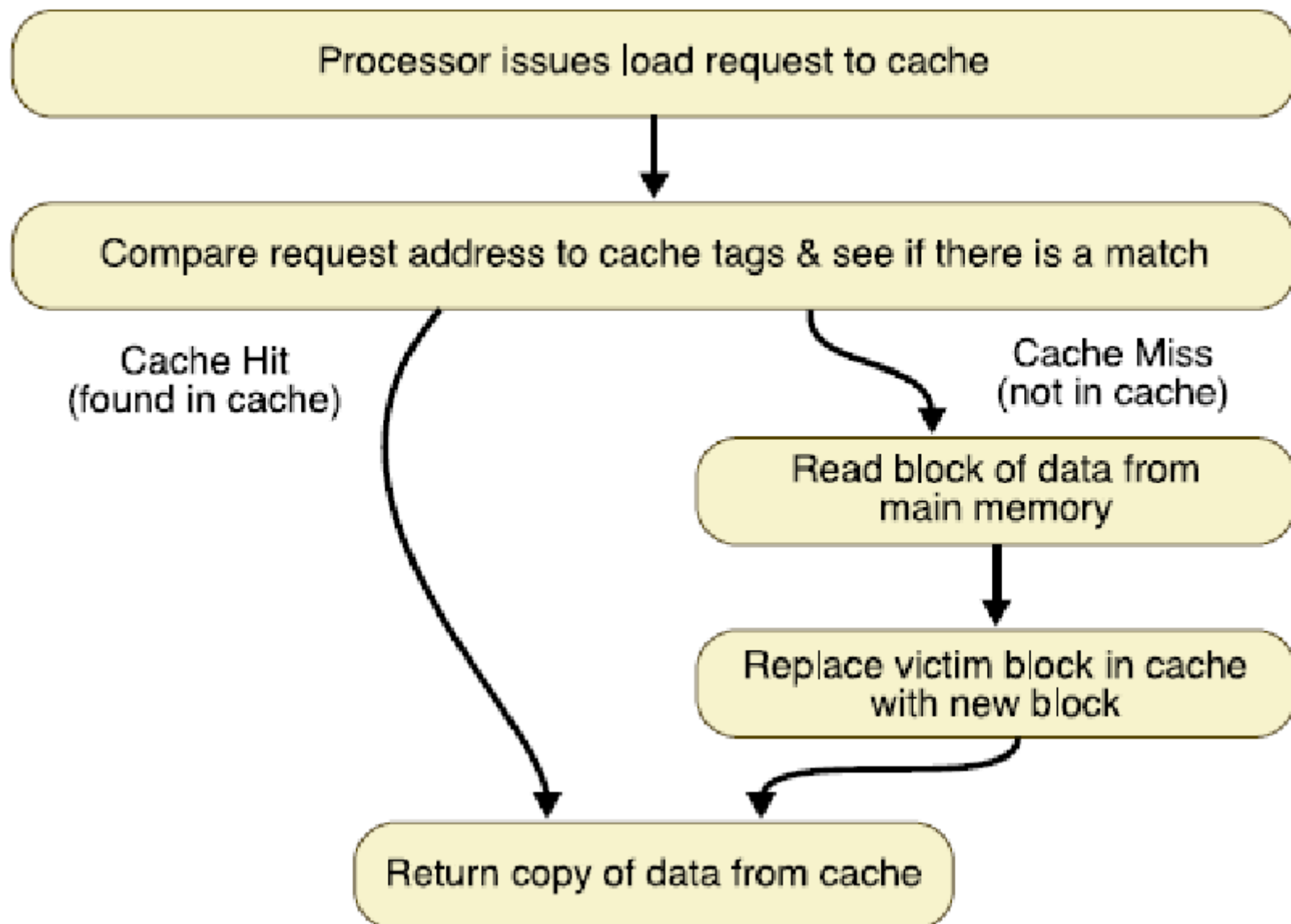
- Fact:
  - CPU goes in vertical
  - Memory goes in horizon
- Build memory inside CPU chip?
  - Cost
- Combination?
  - Small amount of fast memory
  - Large amount of slow memory

# Cache Memory

- Observation: 90% time to execute 10% instructions overall
- Idea: the most heavily used memory words are kept in the cache. When the CPU need a word, it first looks in the cache

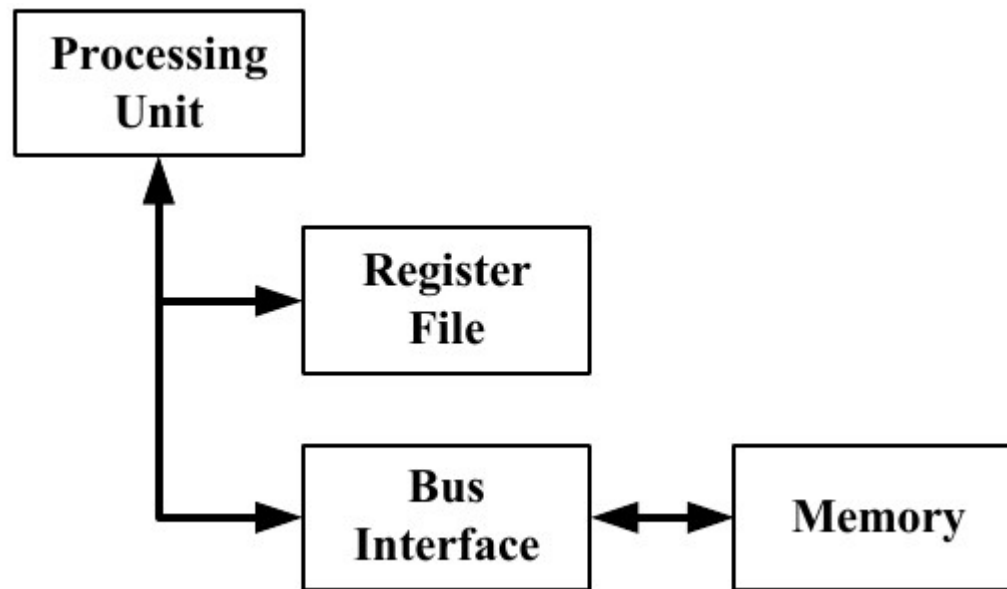


# Basic Cache Algorithm for Load



# Average Memory Access Time

## Registers and Main Memory (1)



# Average Memory Access Time

## Registers and Main Memory (2)

Average memory access time

$$t_{aa} = t_{reg} \cdot p_{reg} + t_{mm} \cdot p_{mm}$$

Where:

$t_{reg}$  Register Access Time

$p_{reg}$  Probability that the data is in the register

$t_{mm}$  Main Memory Access Time

$p_{mm}$  Probability that the data is in the register

Defined:

$p_{reg} = h_{reg}$  The **hit rate** for data in the register

$p_{mm} = (1 - h_{reg})$  The **miss rate** for data in the register

# Average Memory Access Time

## Registers and Main Memory (3)

$$t_{reg} = 1ns, h_{reg} = 50\%, \text{ and } t_{mm} = 100ns$$

$$t_{aa} = ?$$

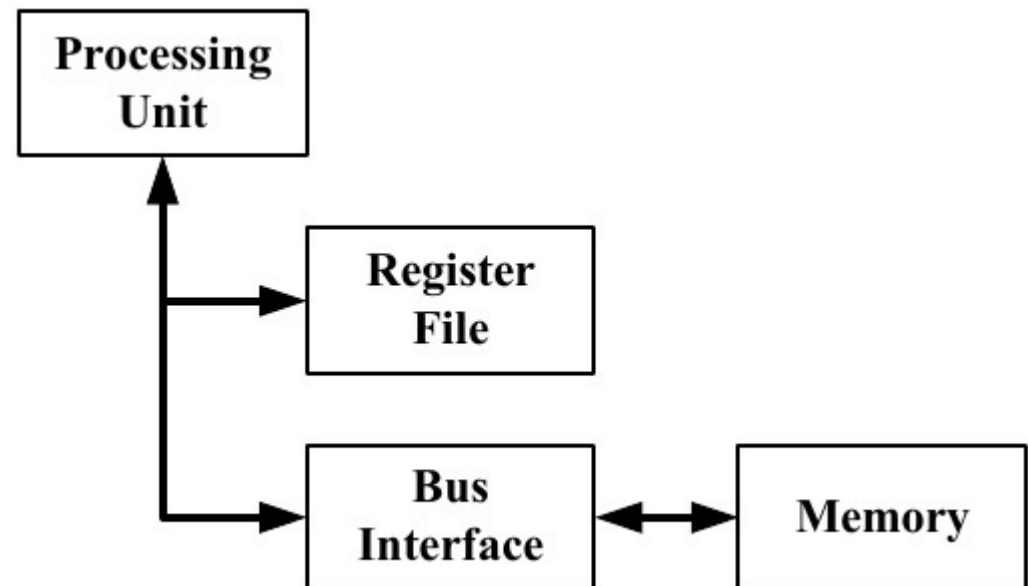
Defined:

$$p_{reg} = h_{reg}$$

The **hit rate** for data in the register

$$p_{mm} = (1 - h_{reg})$$

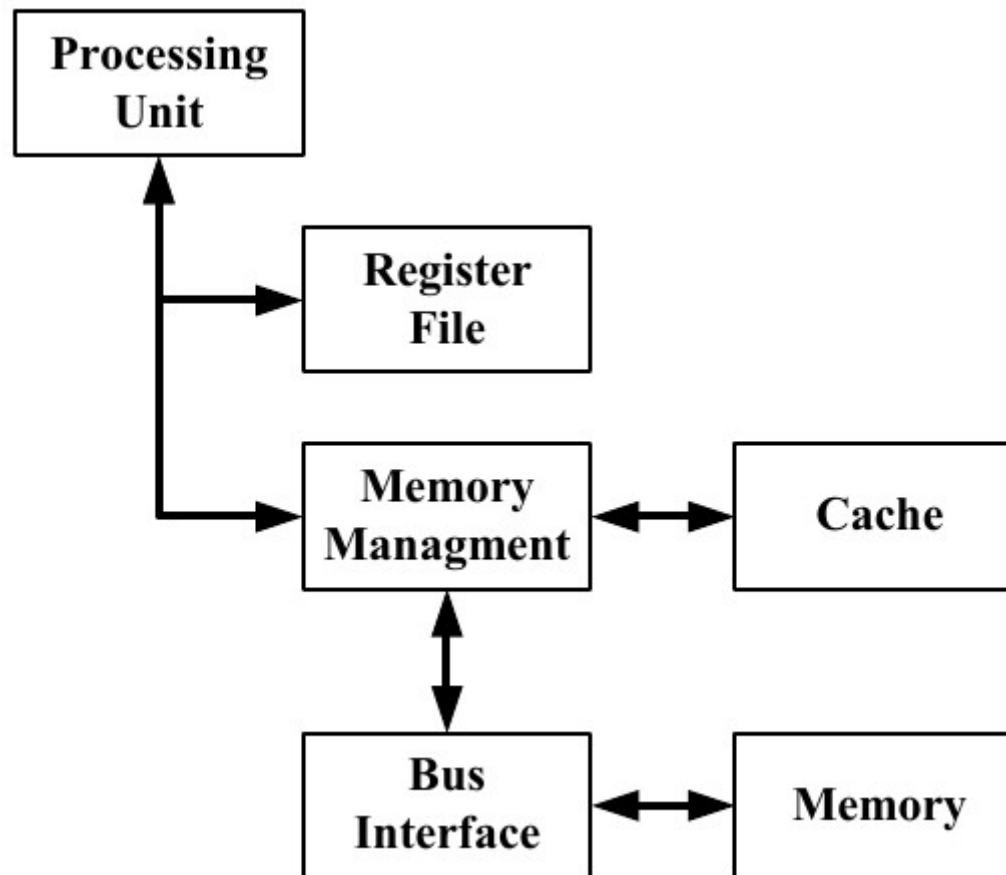
The **miss rate** for data in the register





# Average Memory Access Time

Registers, Cache, and Main Memory (1)



# Average Memory Access Time

## Registers, Cache, and Main Memory (2)

Average memory access time

$$t_{aa} = t_{reg} \cdot p_{reg} + t_{cache} \cdot p_{cache} + t_{mm} \cdot p_{mm}$$

Where:

$t_{reg}$  Register Access Time

$p_{reg}$  Probability that the data is in the register

$t_{cache}$  Cache Access Time

$p_{cache}$  Probability that the data is in the cache

$t_{mm}$  Main Memory Access Time **plus Cache Miss Time (MMU)**

$p_{mm}$  Probability that the data is in the register

# Average Memory Access Time

## Registers, Cache, and Main Memory (3)

Defined:

$$p_{reg} = h_{reg}$$

The **hit rate** for data in the register

$$p_{cache} = (1 - h_{reg}) \cdot h_{cache}$$

The cache **hit rate** for data in the cache

$$p_{mm} = (1 - h_{reg}) \cdot (1 - h_{cache})$$

The **miss rate** for data not in the register or cache

$$t_{reg} = 1 ns, h_{reg} = 50\%$$

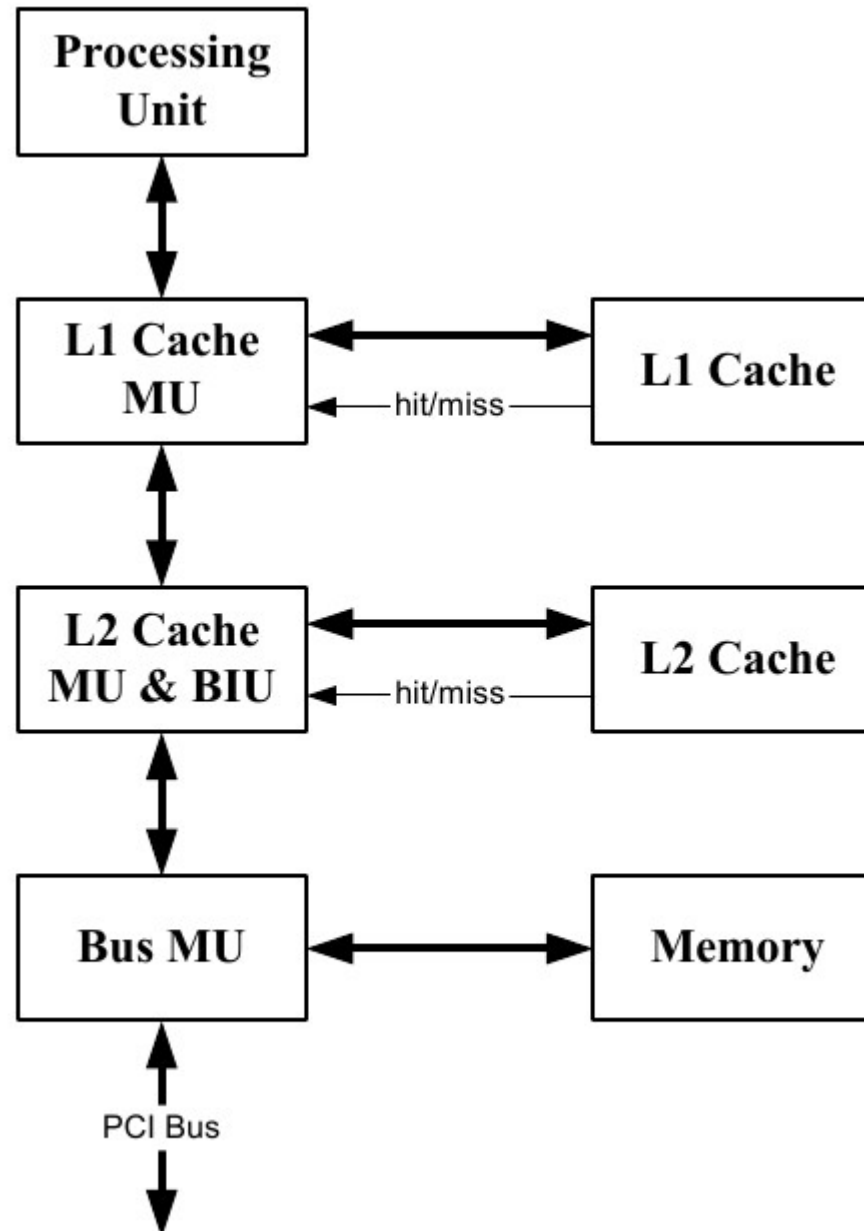
$$t_{cache} = 20 ns, h_{cache} = 90\%$$

$$t_{mm} = 100 ns$$

$$t_{aa} = ?$$

# Average Memory Access Time

## L1, L2 Cache, and Main Memory (1)



# Average Memory Access Time

## L1, L2 Cache, and Main Memory (2)

$$t_{aa} = t_{L1cache} \cdot p_{L1cache} + t_{L2cache} \cdot p_{L2cache} + t_{mm} \cdot p_{mm} + t_{other} \cdot p_{other}$$

$t_{L1cache}$  L1 cache Access Time

$p_{L1cache}$  Probability that the data is in the L1 cache

$t_{L2cache}$  L2 Cache Access Time **plus L1 Cache Miss Time (L1CMU)**

$p_{L2cache}$  Probability that the data is in the L2 cache

$t_{mm}$  Main Memory Access Time **plus L1 and L2 Cache Miss Time**

$t_{mm}$  Main Memory Access Time

$p_{mm}$  Probability that the data is in the register

$t_{other}$  Data Access Time for other storage media or I/O **plus previous Time**

$p_{other}$  Probability that the data is in other storage media or I/O

$$t_{aa} = t_{L1cache} \cdot h_{L1cache} + t_{L2cache} \cdot (1 - h_{L1cache}) \cdot h_{L2cache} + t_{mm} \cdot (1 - h_{L1cache}) \cdot (1 - h_{L2cache}) \cdot h_{mm} + \dots$$

# Cache Memory

- Modern machines typically have split L1 caches.
  - **Level 1 Instruction Cache**: Optimized for threads of execution, support for branching, integrated with intelligent instruction fetch and preprocessing units. Focus on filling from higher level caches and main memory. It does not typically need to write-back instruction to main memory.
  - **Level 1 Data Cache**: Optimized for data handling. Facilitate data fetching for computations and data write back of results.
- Therefore, the cache controllers are concerned about different “cache policies and procedures”.

# Mapping Functions

- Cache of 64kByte
- Cache block of 4 bytes
  - i.e. cache is 16k ( $2^{14}$ ) lines of 4 bytes
- 16MBytes main memory
  - 24 bit address
  - ( $2^{24}=16\text{M}$ )

# Direct Mapping

- Each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place
- Address is in two parts
- Least Significant  $w$  bits identify unique word
- Most Significant  $s$  bits specify one memory block
- The MSBs are split into a cache line field  $r$  and a tag of  $s-r$  (most significant)

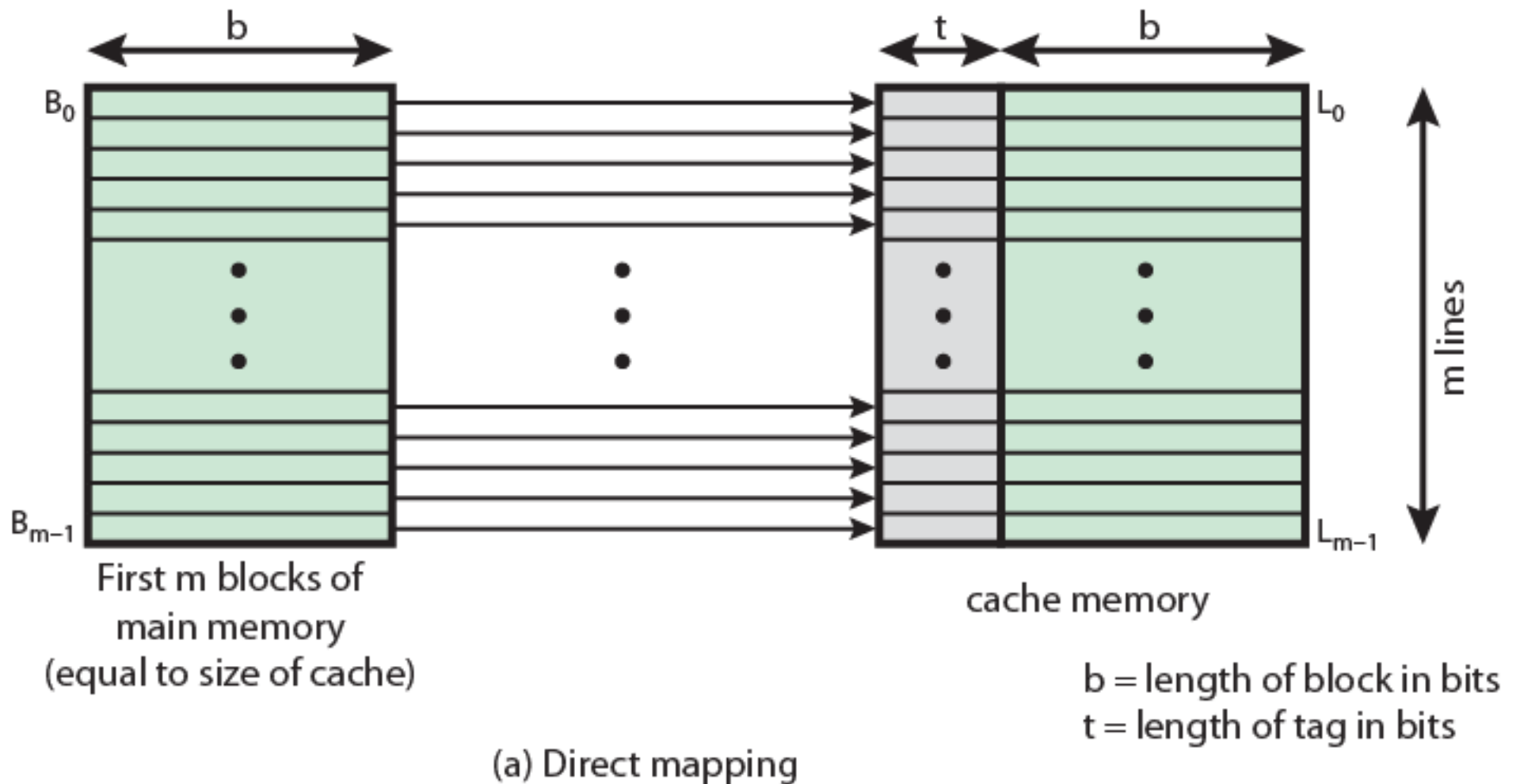


# Direct Mapping: Address Structure

- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
  - 8 bit tag (=22-14)
  - 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

Tag s-r	Line or Slot r	Word w
8	14	2

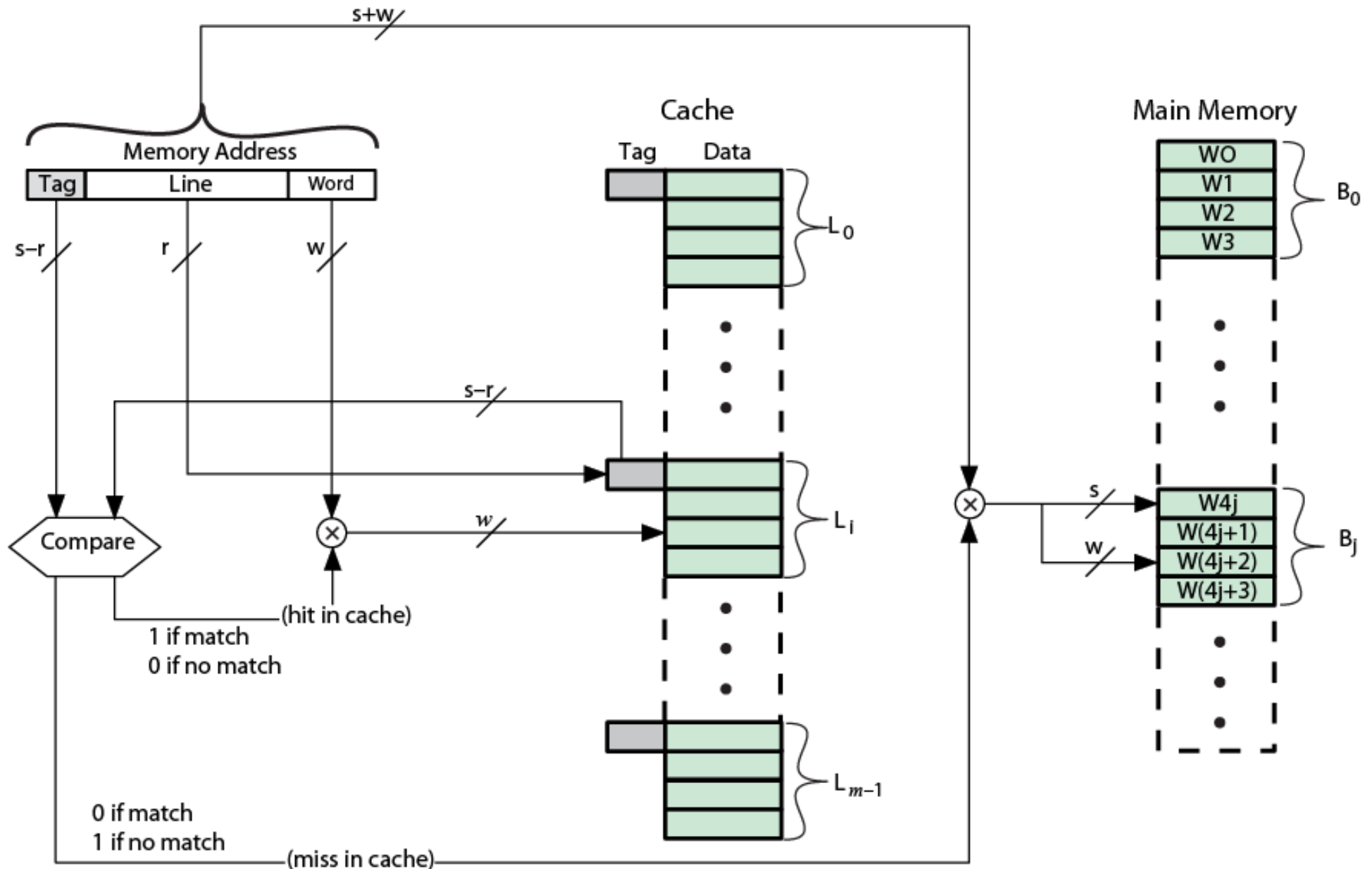
# Direct Mapping: From Cache to Memory

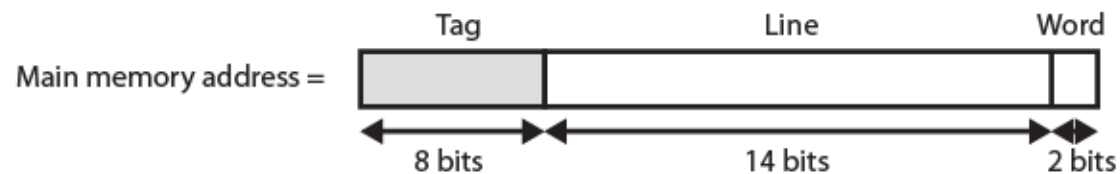
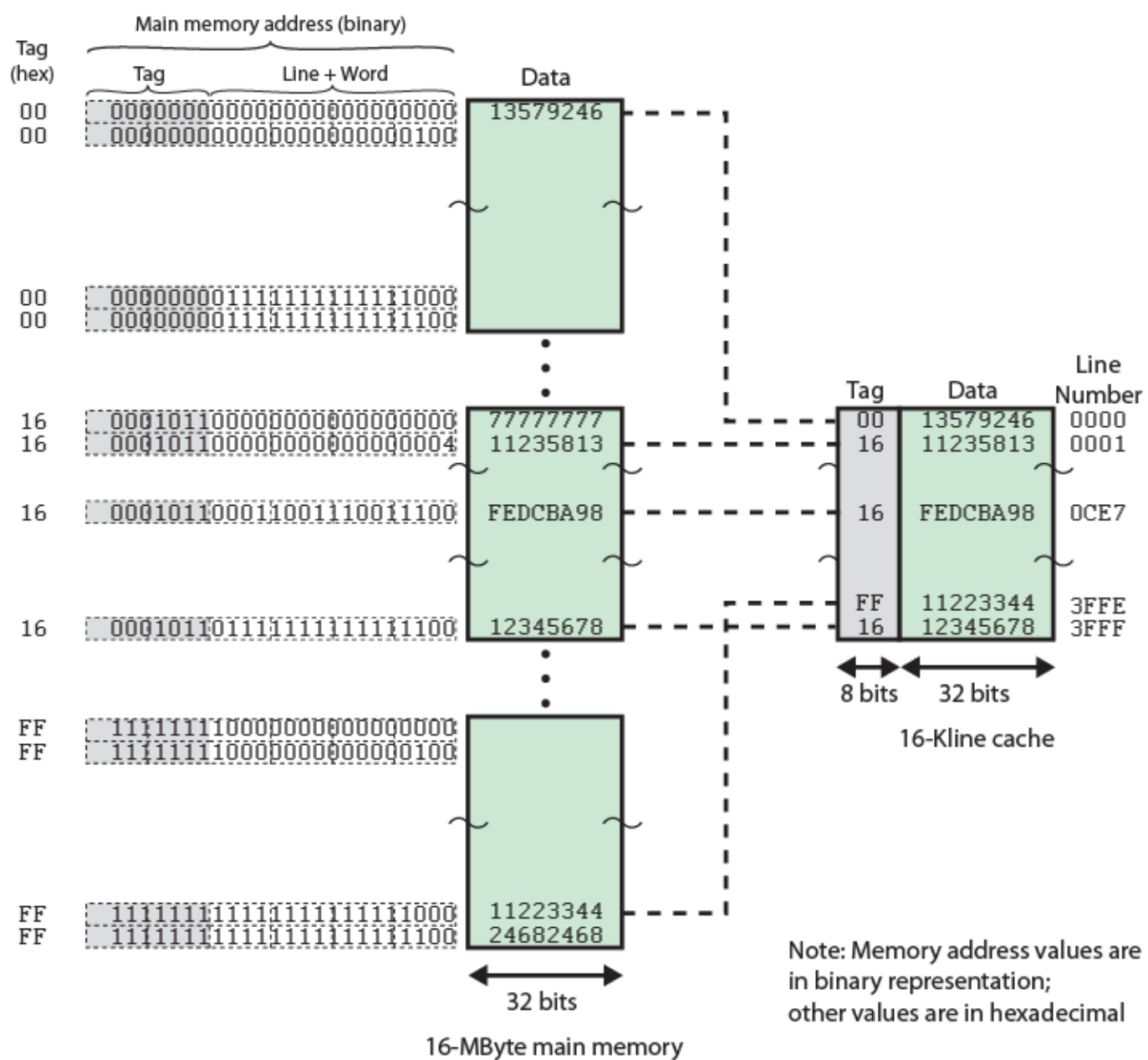


# Direct Mapping: Cache Line Table

Cache line	Main Memory blocks held
0	$0, m, 2m, 3m \dots 2s-m$
1	$1, m+1, 2m+1 \dots 2s-m+1$
...	
$m-1$	$m-1, 2m-1, 3m-1 \dots 2s-1$

# Direct Mapping: Cache Organization





# Direct Mapping: Summary

- Address length =  $(s + w)$  bits
- Number of addressable units =  $2^{(s+w)}$  words or bytes
- Block size = line size =  $2^w$  words or bytes
- Number of blocks in main memory =  $2^{(s+w)}/2^w = 2^s$
- Number of lines in cache =  $m = 2^r$
- Size of tag =  $(s - r)$  bits

# Direct Mapping: Pros & Cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

# Victim Cache

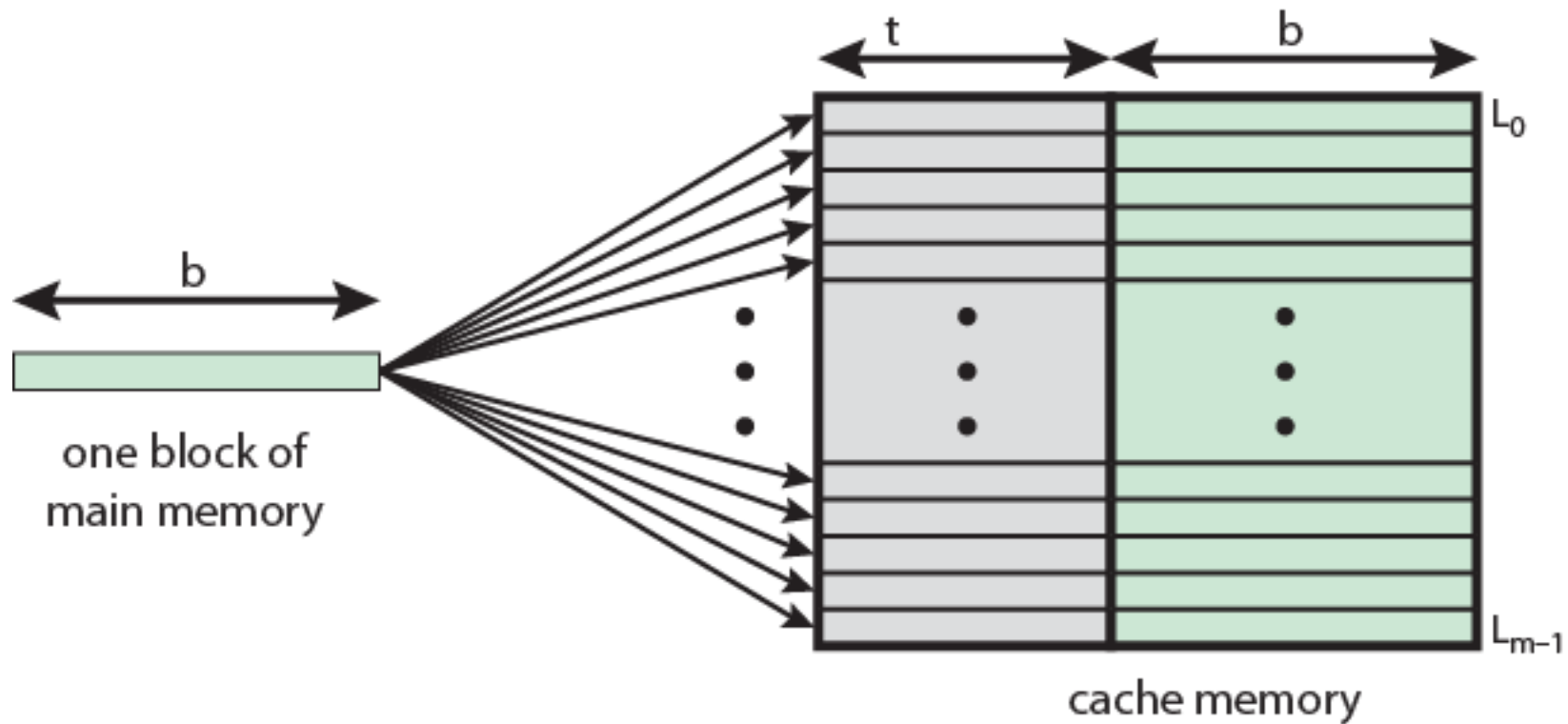
- Lower miss penalty
- Remember what was discarded
  - Already fetched
  - Use again with little penalty
- Fully associative
- 4 to 16 cache lines
- Between direct mapped L1 cache and next memory level



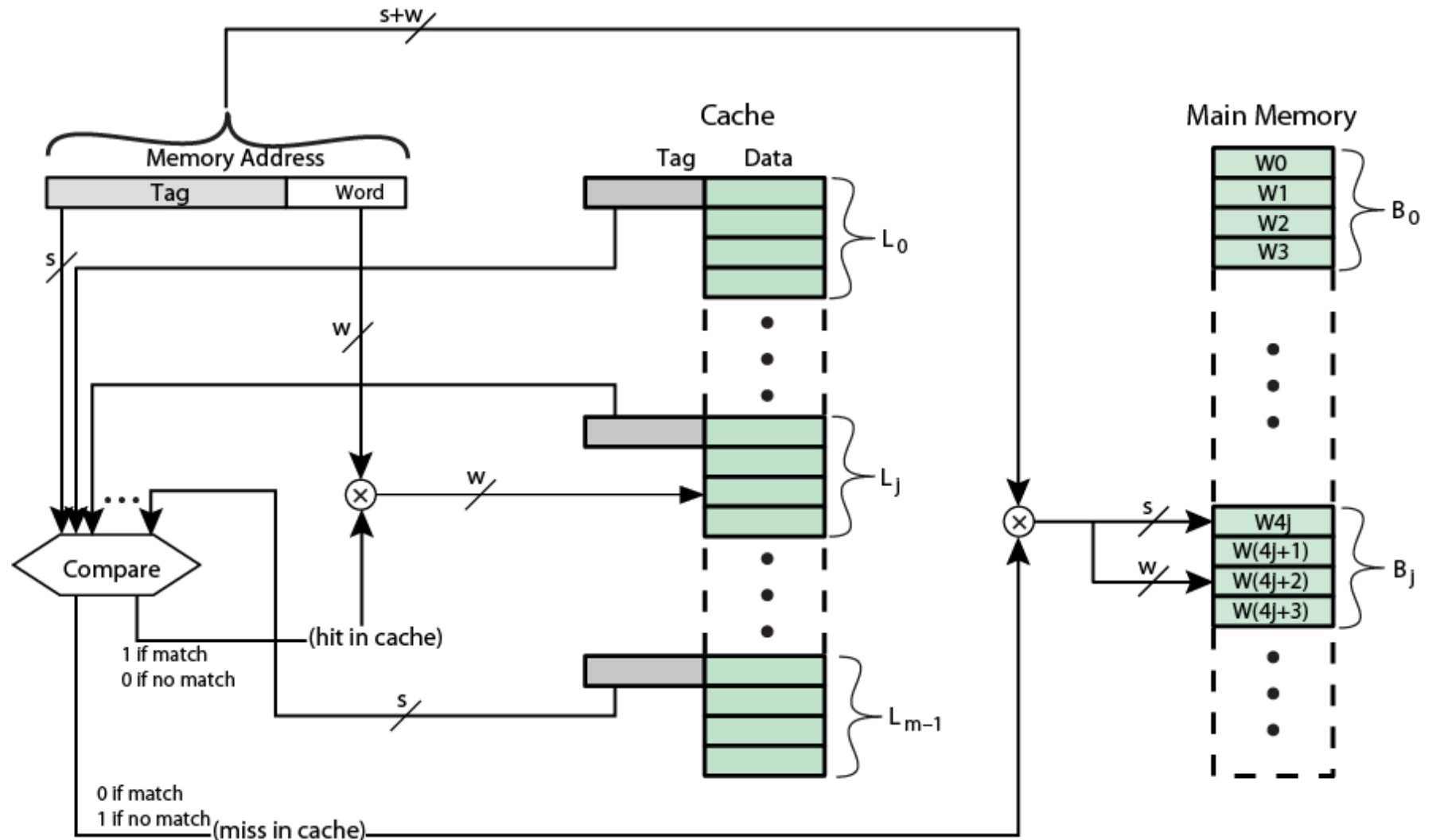
# Associative Mapping (AM)

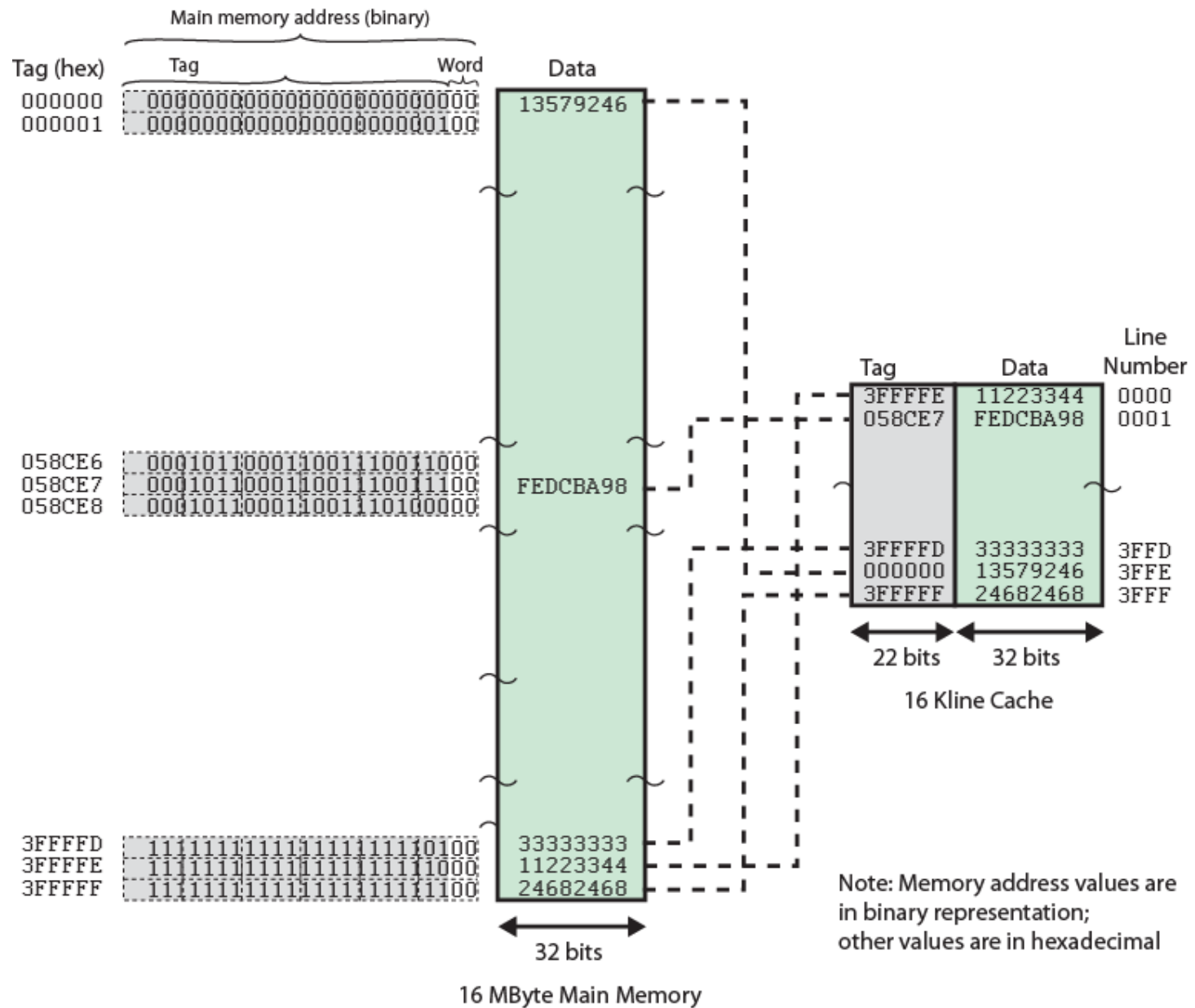
- A main memory block can load into any line of cache
- Memory address is interpreted as tag and word
- Tag uniquely identifies block of memory
- Every line's tag is examined for a match
- Cache searching gets expensive

# AM: Cache to Memory



# Fully Associative Cache Organization





# AM: Address Structure

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which 16 bit word is required from 32 bit data block
- e.g.

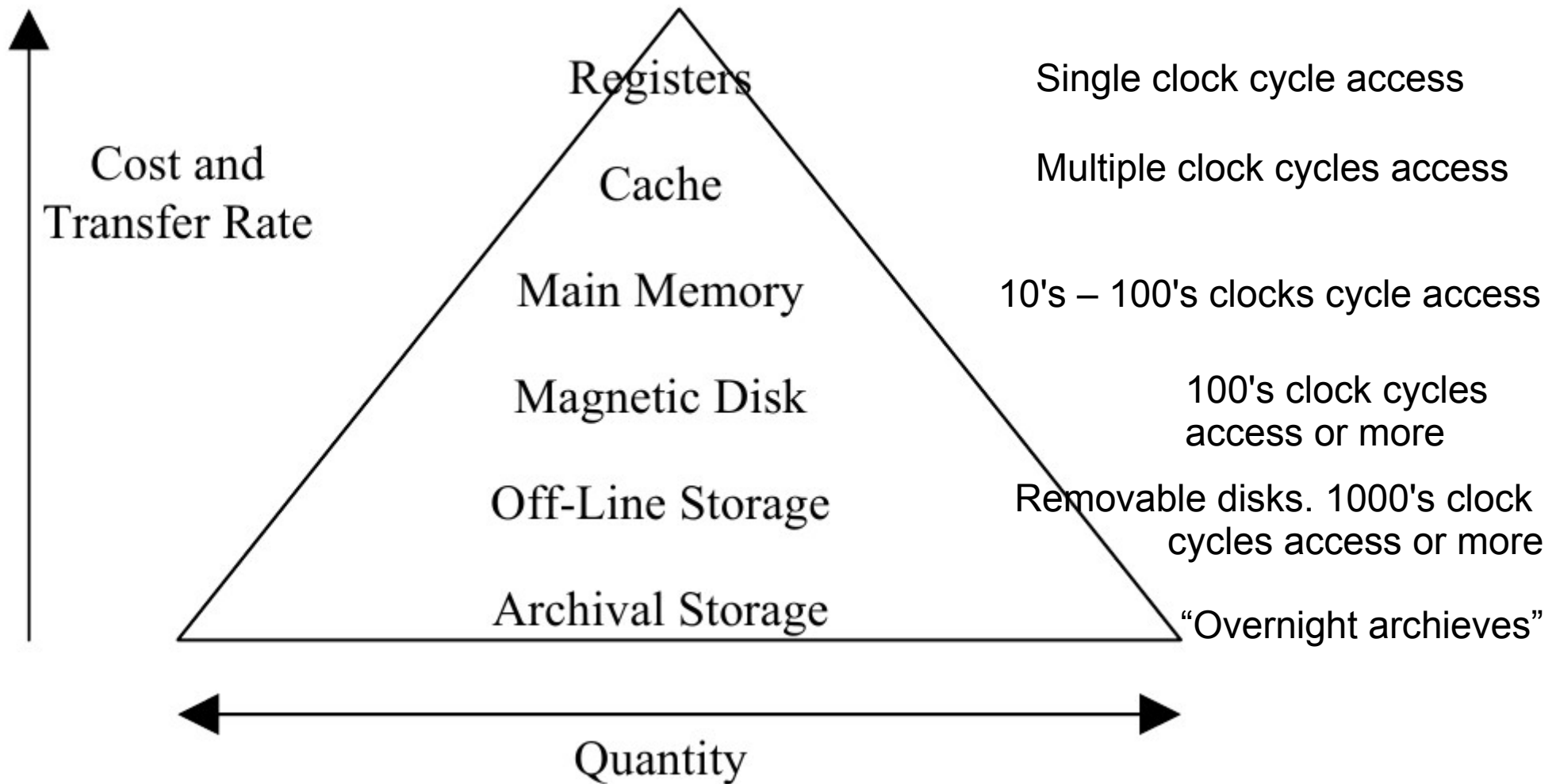
– Address	Tag	Data	Cache line
– FFFFFC	FFFFFC	24682468	3FFF



# AM: Summary

- Address length =  $(s + w)$  bits
- Number of addressable units =  $2^{(s+w)}$  words or bytes
- Block size = line size =  $2^w$  words or bytes
- Number of blocks in main memory =  $2^{(s+w)}/2^w = 2^s$
- Number of lines in cache = undetermined
- Size of tag =  $s$  bits

# Memory Hierarchy Pyramid (again)



# IC Memory Types

- ROM
  - Read-only Memory
  - Permanent storage (boot code, embedded code)
- SRAM
  - Static Random Access Memory: cache and high speed access
- DRAM
  - Dynamic Random Access Memory: Main Memory
- EPROM
  - Electrically programmable read-only memory
  - Replace ROM when reprogramming required



# IC Memory Types

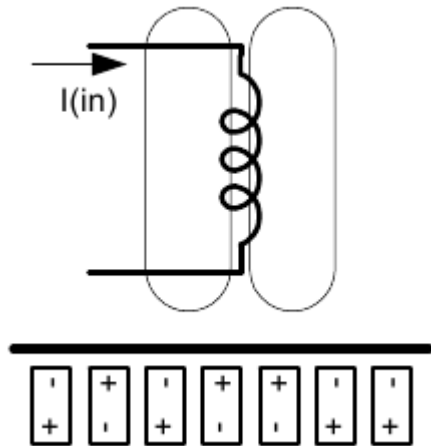
- EEPROM
  - Electrically erasable, programmable read-only memory.
  - Alternative to EPROM, limited but regular reprogramming,
  - Device configuration info during power down
- FLASH
  - An advancement on EEPROM technology allowing blocks of memory location to be written and cleared at one time instead. Found in thumb drives/memory stick or as solid-state hard disks.

**Note:** EEPROM and FLASH have lifetime write cycle limitations!

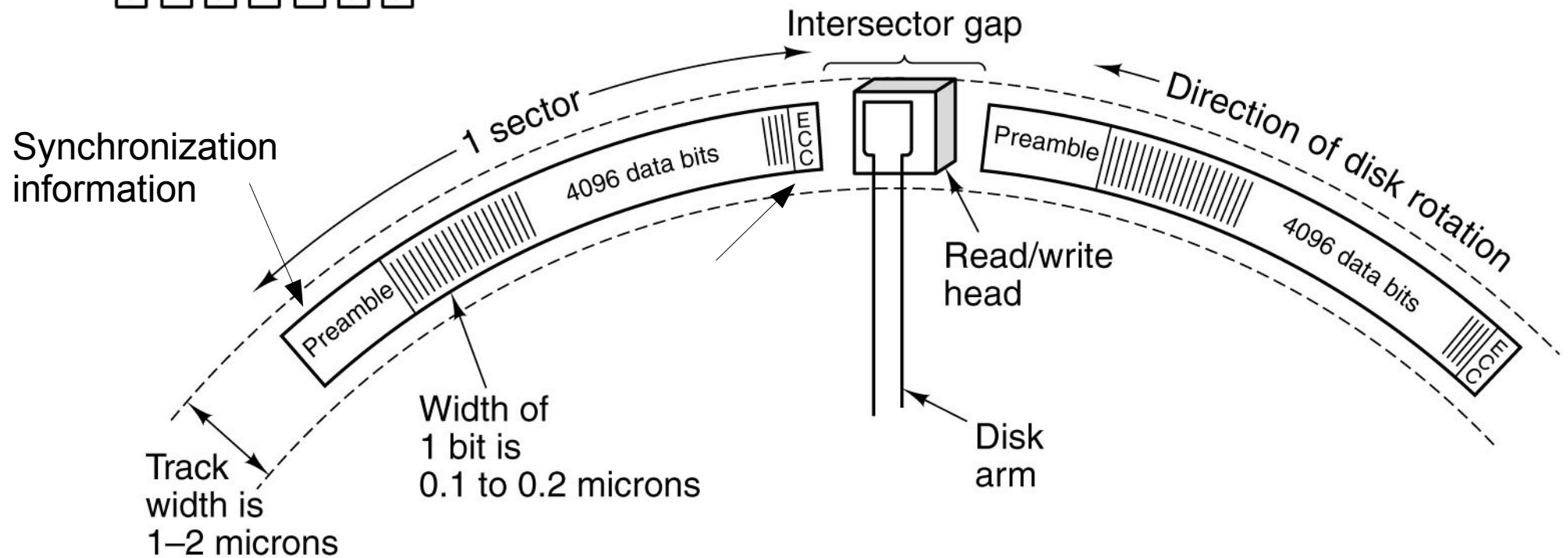
# Secondary Memory

## Magnetic, CD ...

# Magnetic Disks



Electronic Coil Writing/Reading Magnetic Medium

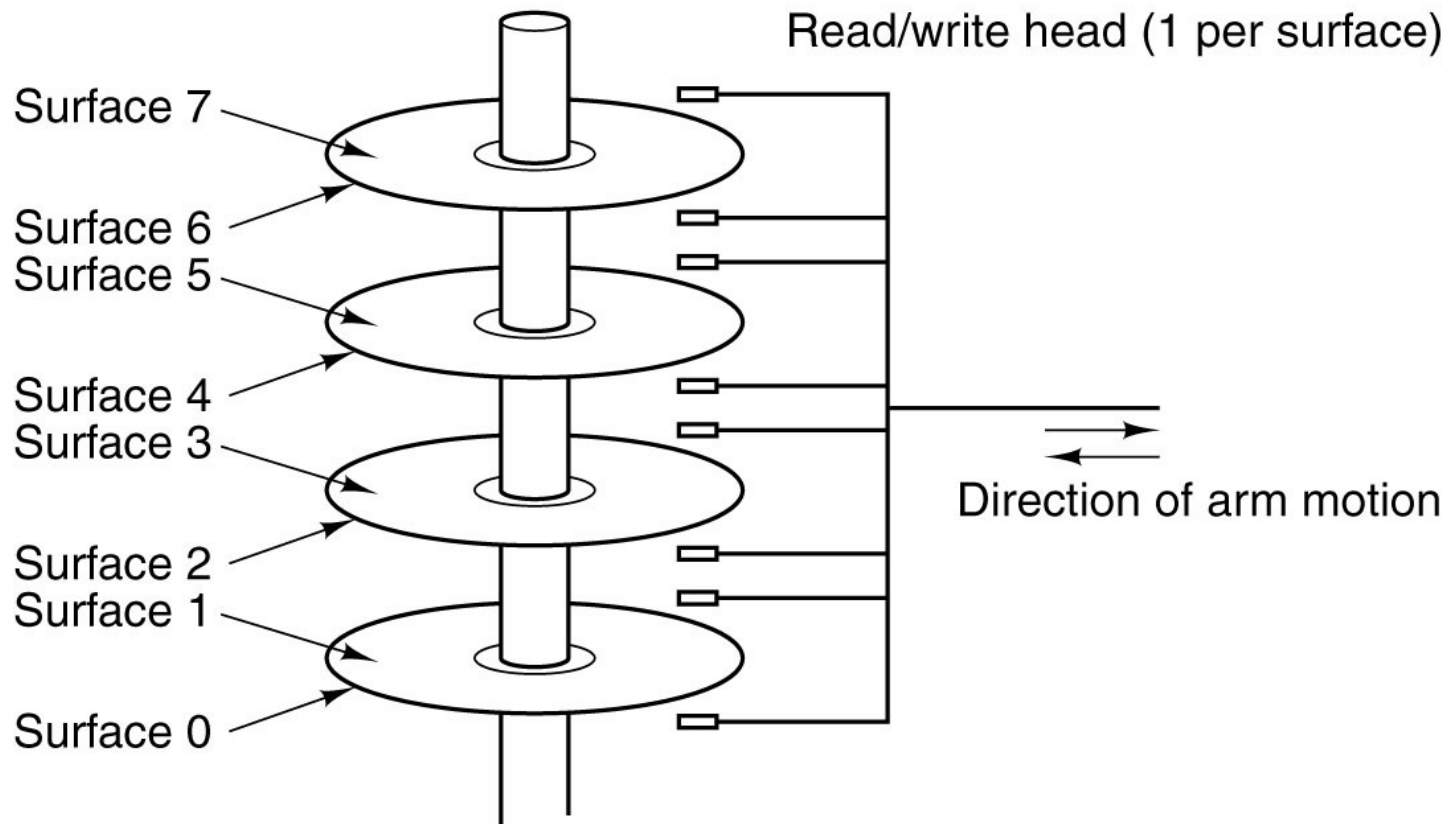


A portion of a disk track. Two sectors are illustrated.

# Magnetic Disks

- Track
  - A radial spaced circle on a platter for storing data.
  - Typically 5,000 to 10,000 tracks per centimeter on a platter or 1-2 micron track widths.
- Sector
  - A fixed bit-length section of a track.
  - There are multiple sectors in a track.
  - A typical sector contains: a preamble, 512 Bytes or 4096 bits, and error correction code bits.
  - Between sectors on a track are inter-sector gaps.

# Magnetic Disks



A disk with four platter

# Speed up the Transfers

## **Disk Controllers:**

- Typically includes buffer memory space for rapid burst transfers.
- May (must) allow simultaneous access to the multiple tracks in a cylinder.
- Performs the ECC generation, testing, and corrections
- Can provide a mapping table of good and bad sectors.

# ATA vs SATA vs SCSI

- IDE → EIDE → ATA → SATA: IDE disks
- SCSI disk
  - Differ from IDE disk in the interface
  - Higher data rate
  - More than one devices in the bus can operate (compare to 1 active device of IDE/EIDE)
- Current trend: SATA

# RAID (1)

Redundant Array of Inexpensive Disks

- Example from Intel Modular, a strong web server

CPU: 2 x Intel® Nehalem-EP Xeon Quad Core  
Gainestown E5520 ( 2.26GHz 8-Threads 2x6.4GT/s  
8MB(L3))

RAM: Kingston DDR3 PC-8400 12GB/1066/Ecc/Reg  
Intel Certifield

HDD: 14 x Seagate 2.5" 300G/SAS/10Krpm Version  
2.0 (**RAID 5**)



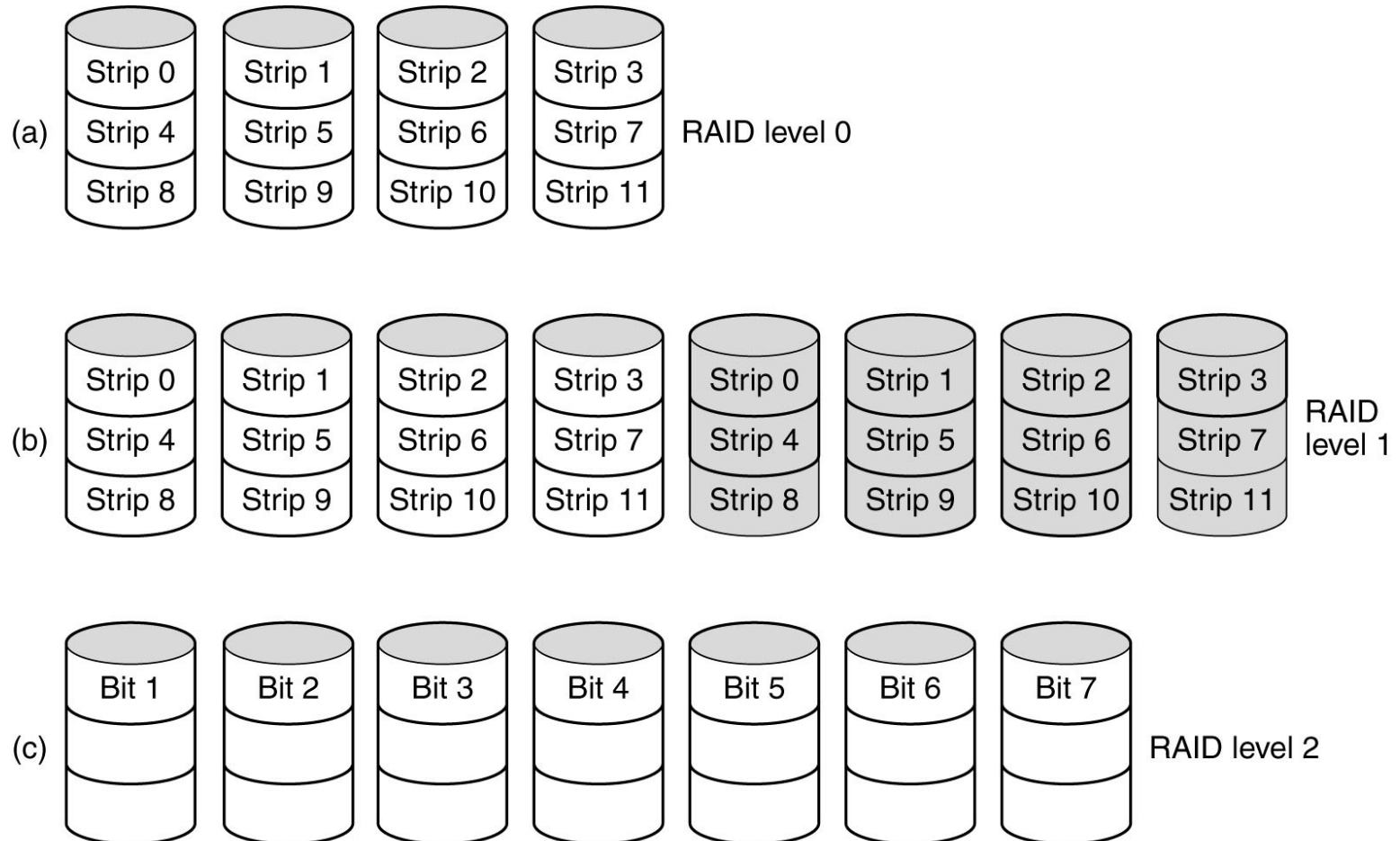
Note: Hard disk provides low bandwidth and is quite easily to fail!

Ideas: Use a box full of hard disks to allow:

- Higher data transfer and storage rates (operate disks in parallel)
- Higher degrees of redundancy (save on multiple disks simultaneously), and
- Higher degree of error detection and correction (Additional ECC)

# RAID (2)

## Redundant Array of Inexpensive Disks



RAID levels 0 through 2.  
Backup and parity disks are shown shaded.

# RAID (3)

Redundant Array of Inexpensive Disks

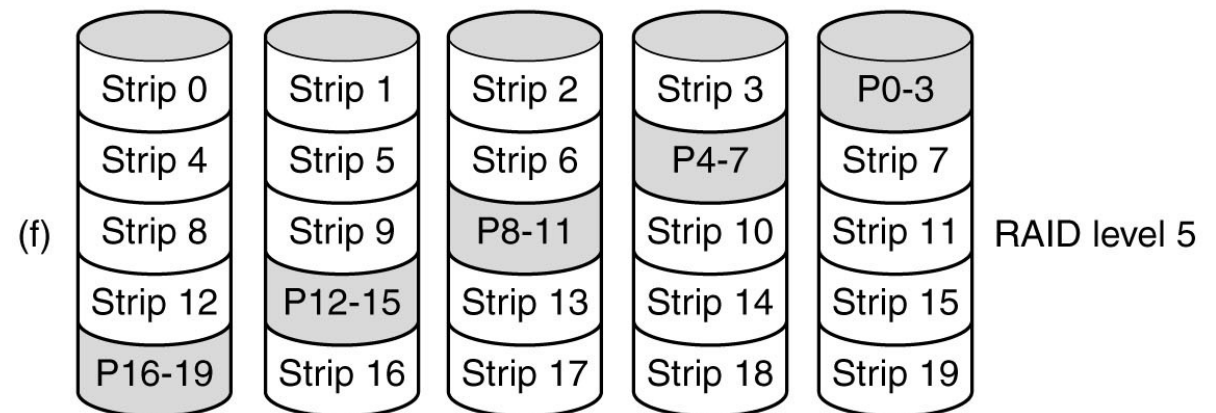
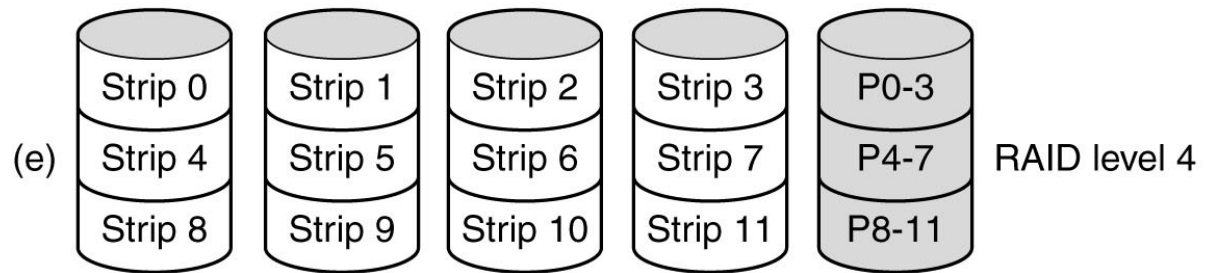
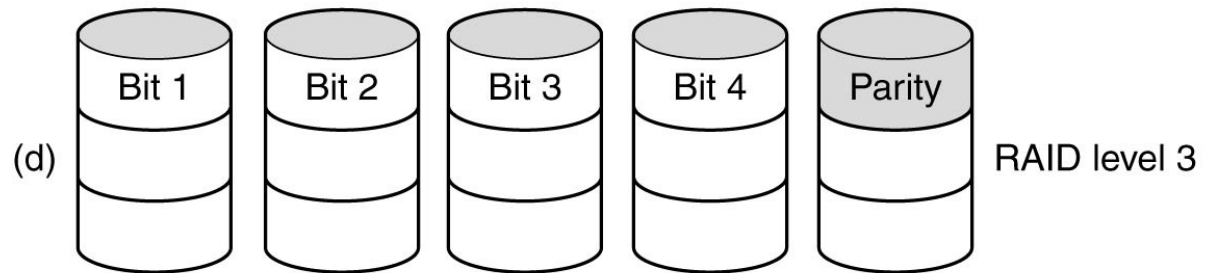
Level	Description	Feature
RAID 0	Block-level striping	Speed Increase
RAID 1	Mirroring	Redundancy
RAID 2	Bit-level striping with error-correcting code  Drives were synchronized	Faster with conditional 1 drive failure recovery

# RAID (4)

Redundant Array of Inexpensive Disks

RAID levels 3 through 5.

Backup and parity disks are shown shaded.



# RAID (5)

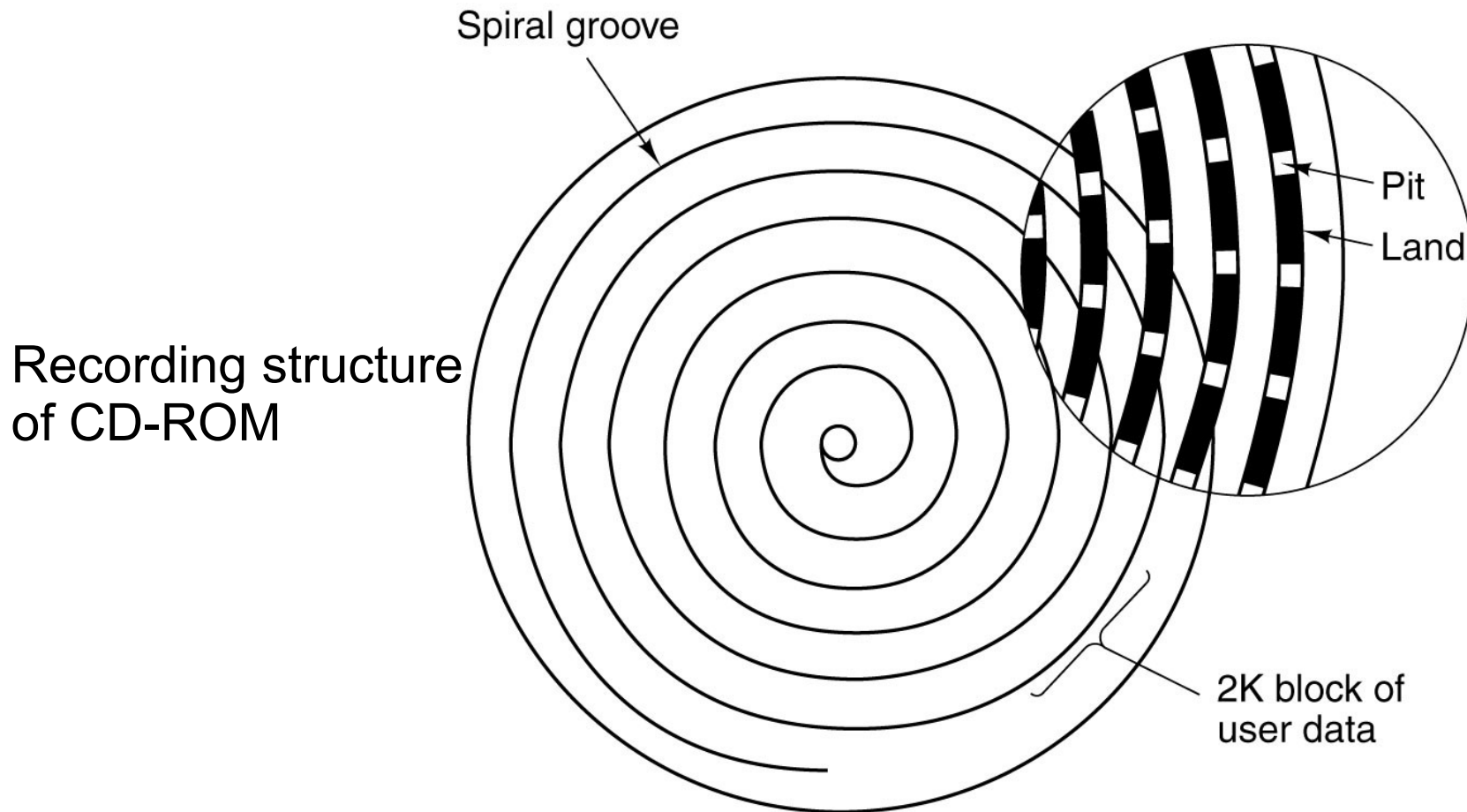
## Redundant Array of Inexpensive Disks

Level	Description	Feature
RAID 3	Byte-level striping with parity (Simple version of level 2)	Faster with 1 drive failure recovery (with parity bit?)
RAID 4	Block-level striping with parity (XOR calculation)	Faster with 1 drive failure recovery
RAID 5	Block-level striping with distributed parity Eliminated the bottleneck of parity disk in level 4	Faster with conditional 1 drive failure recovery

# Question

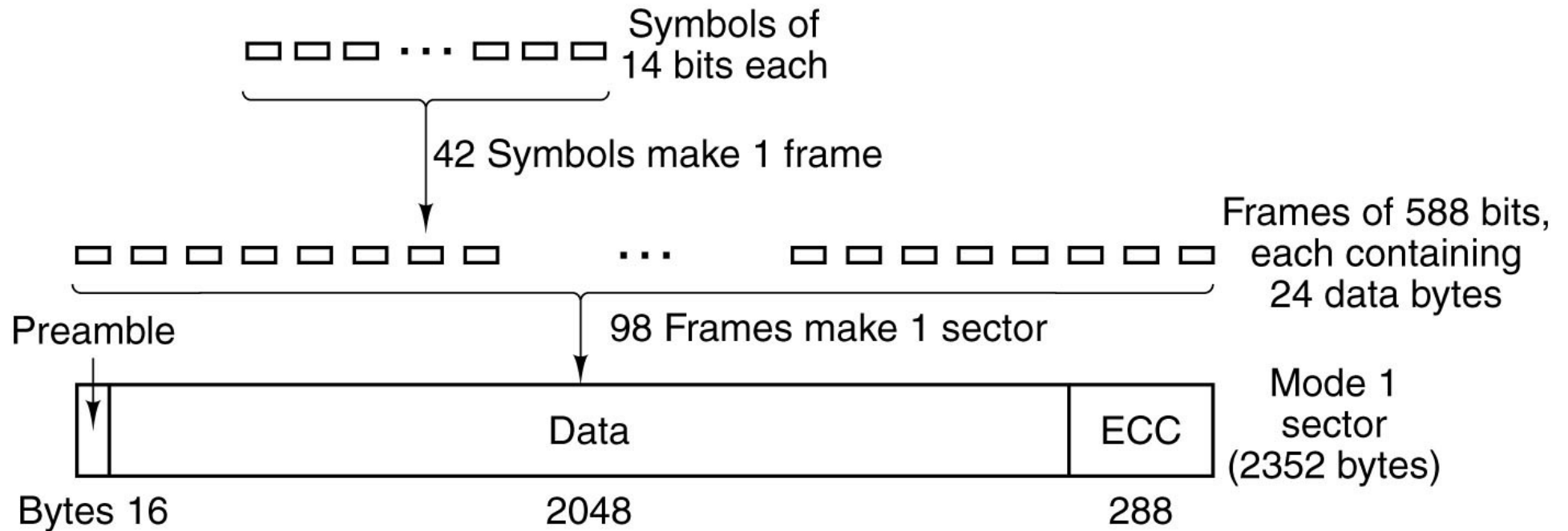
- How ECC with parity bit in RAID level 3 has “recovery” feature?
- Why RAID level 4 and RAID level 5 utilize XOR function but not AND or OR?
- Why RAID level 5 is considered to be more fairly distributing in utilizing among disks than level 4?
- Total storing capacity of n disk in RAID level 0, 1, 2, 3, 4, and 5?

# CD-ROM



Bit=1 Transitions from pit to land OR transitions from land to pit  
Bit=0 No transition of the data.

# CD-ROM



Logical data layout on a CD-ROM



# CD Storage Requirement

- Audio Format Storage Requirement:

74 min. Audio:  $44,100 \text{ samples/channel/second} \times 2 \text{ bytes/sample} \times 2 \text{ channels} \times 74 \text{ minutes} \times 60 \text{ seconds/minute} = 783,216,000 \text{ bytes}$

80 min. Audio:  $44,100 \text{ samples/channel/second} \times 2 \text{ bytes/sample} \times 2 \text{ channels} \times 80 \text{ minutes} \times 60 \text{ seconds/minute} = 846,720,000 \text{ bytes}$

# Operating Rate

Medium	1X speed			Capacity (GB)	Full Read Time (min)
	Mbit/s	kB/s	KiB/s		
CD	1.229	153.6	150.0	0.734	80
DVD	11.080	1,385.0	1,352.5	4.7	57
Blu-ray Disc	36.000	4,500.0	4,394.5	25.0	93

# DVD

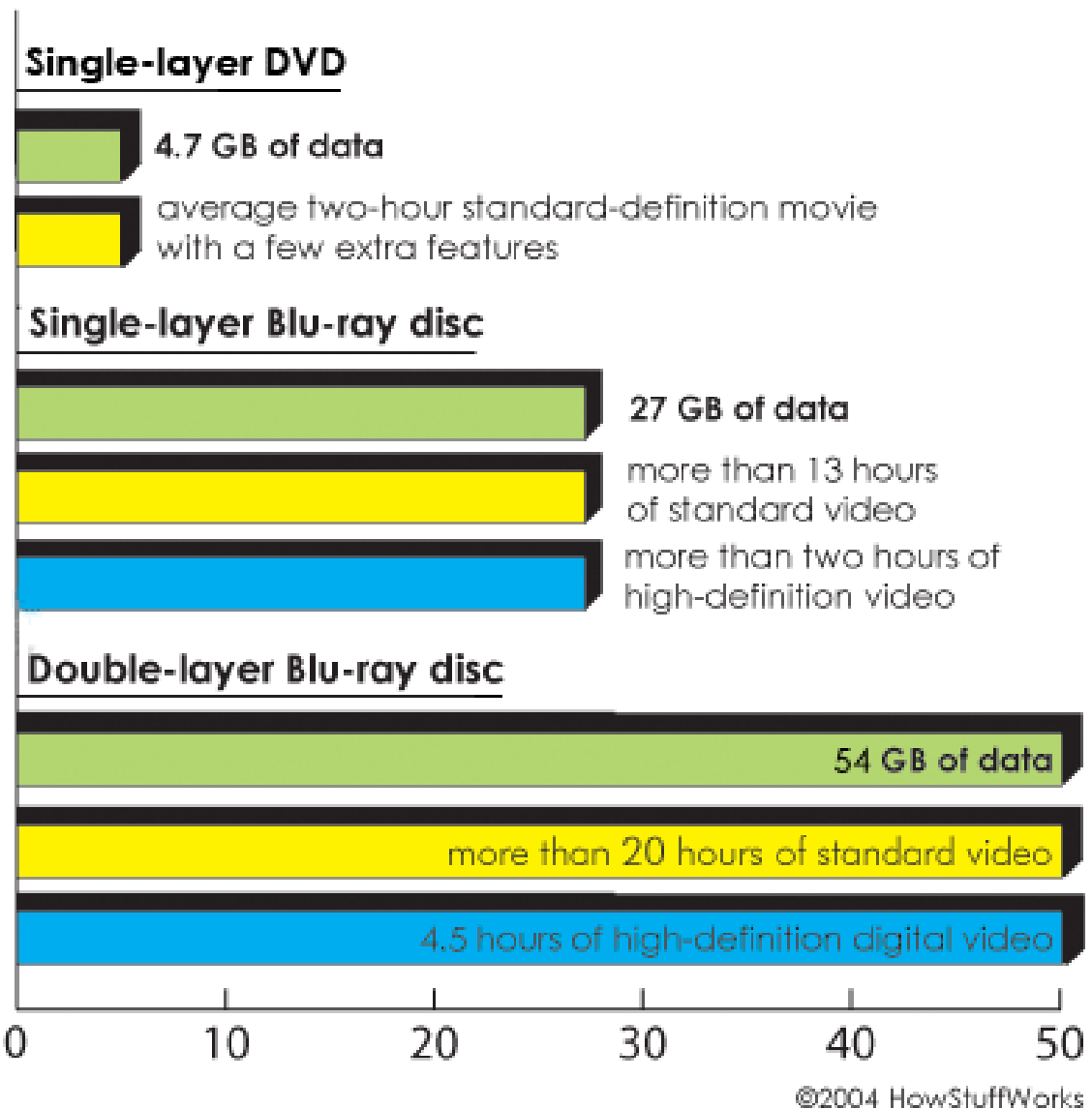
- Name
  - Digital Video Disk (originally)
  - Digital Versatile Disk (now)
- A higher density version of the CD.
  - Smaller pits
  - Tighter spiral
  - Different laser with a reduced spot size (more expensive)
  - Advanced versions with multiple layers (dual layer, top and bottom)

# Operating Rate

- Total Data: 4.7 GB (single-sided, single layer)
  - 133 min of 720x480 high-resolution video, 8-language sound track, and 32 subtitles. (Fit 92% of all Hollywood movies)
- Potential DVD Capacities:
  - Single-sided, single layer      4.7 GB
  - Single-sided, dual layer      8.5 GB
  - Double-sided, single layer      9.4 GB
  - Double-sided, dual layer      17.0 GB

# Blue-Ray

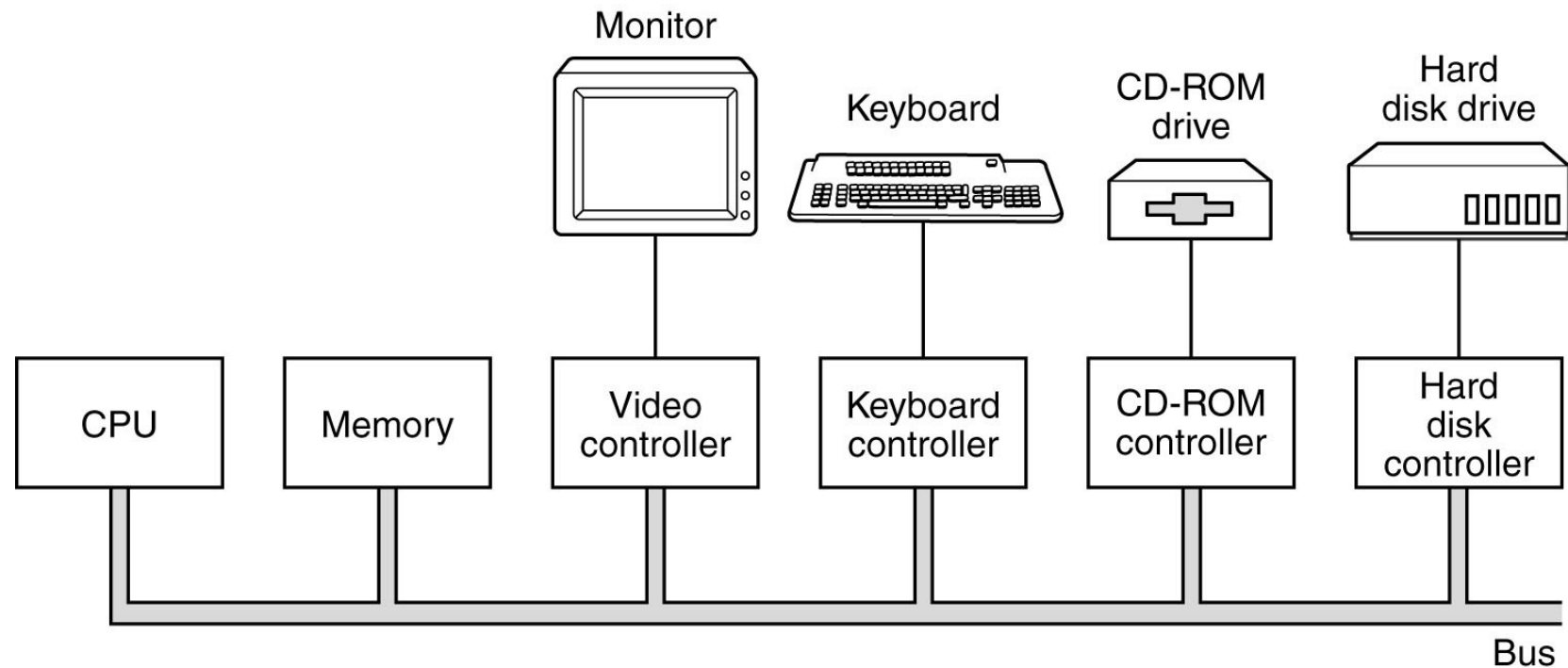
## Blu-ray vs. DVD Capacity



# Input/Output

Bus, Terminal, Mice, Printer, Tel Com

# Single Bus System



Old Omnibus, Unibus, Single-bus System

# Aspects of Bus

- One operation at a time
  - Buses include address/data/control
  - Bus multiplex on one cycle with data on the next
- DMA
  - A controller that reads/write data to/from memory without CPU intervention is said to be performing Direct Memory Access (DMA)
- Bus arbiter
  - What if CPU, I/O controller want to use the bus at the same time?

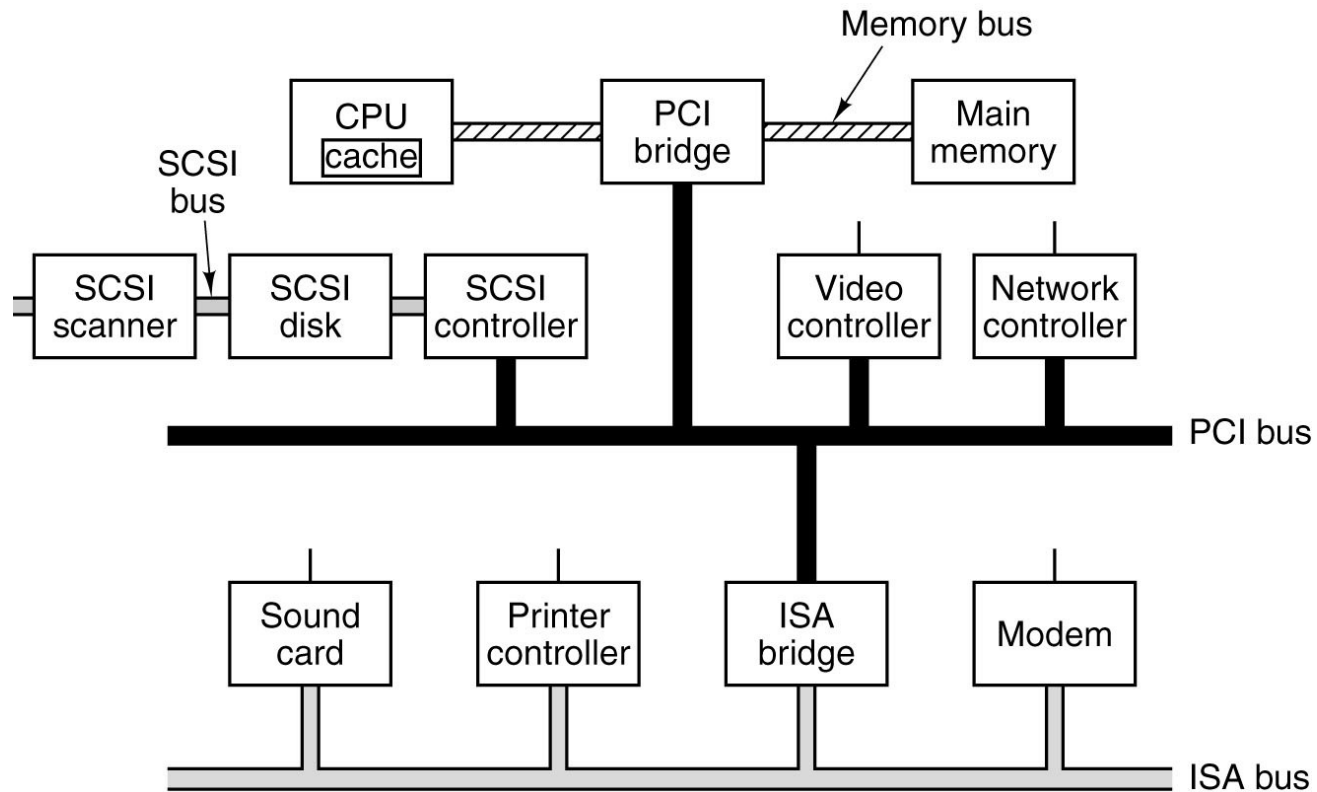


# Problems from Single Bus System

- The simple bus works well when all components were roughly in balance
- Unbalance performance nowadays between components
- People tend to keep printer, scanner, modem when upgrading their PC

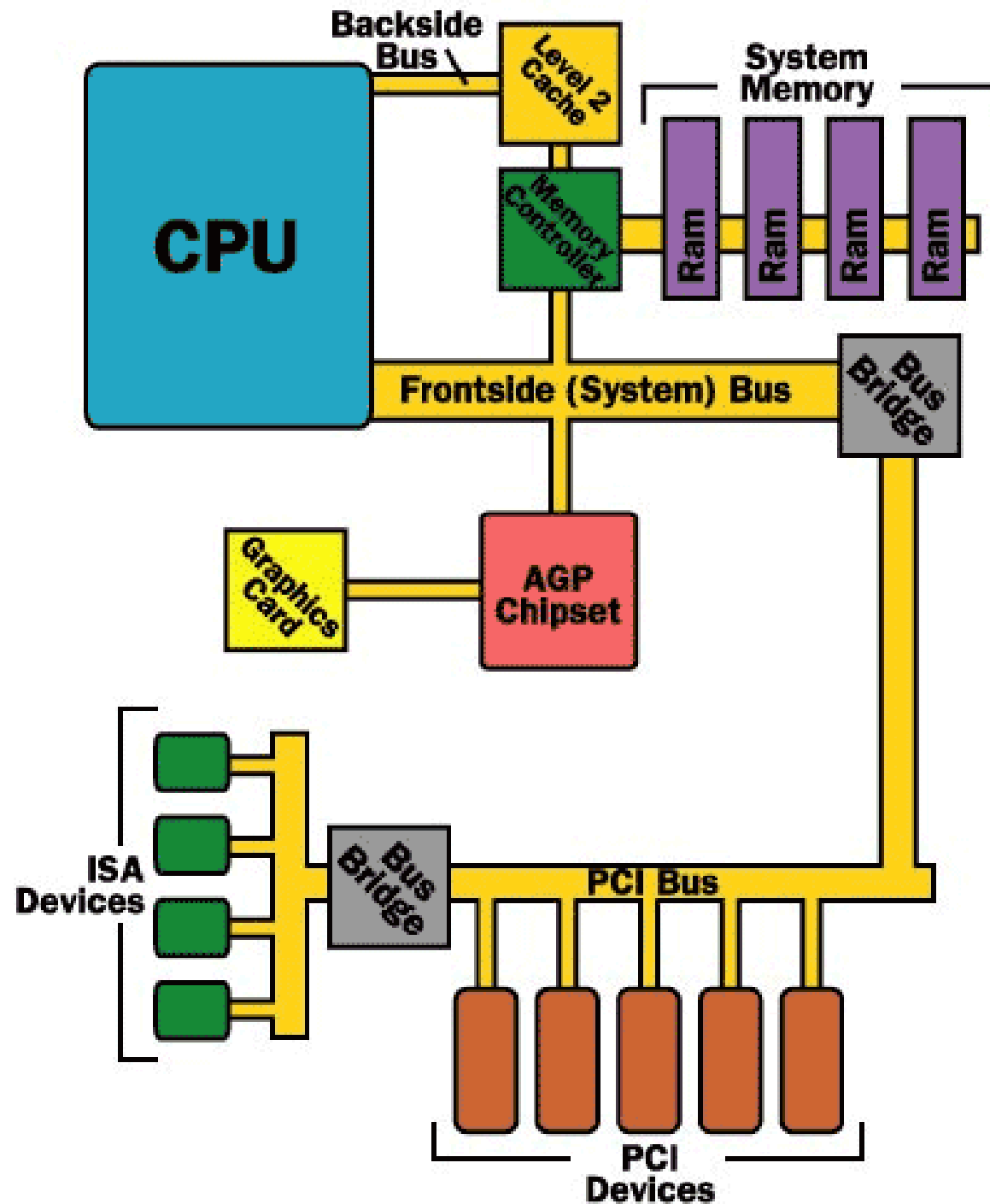
→ Solution: multiple buses

# Multiple Bus System



A typical modern PC with a PCI bus and an ISA bus

# Other version



# Bit-map terminals

Writing pixels to the screen  
(4:3 aspect ratios)

VGA 640 x 480 (480i)

SVGA 800 x 600

XVGA 1024 x 768

1280 x 960

HD 1440 x 1080

1600 x 1200

Writing pixels to a high  
definition screen (16:9  
aspect ratios)

640 x 360

800 x 450

1024 x 576

HD 1280 x 720 (720p)

1600 x 900

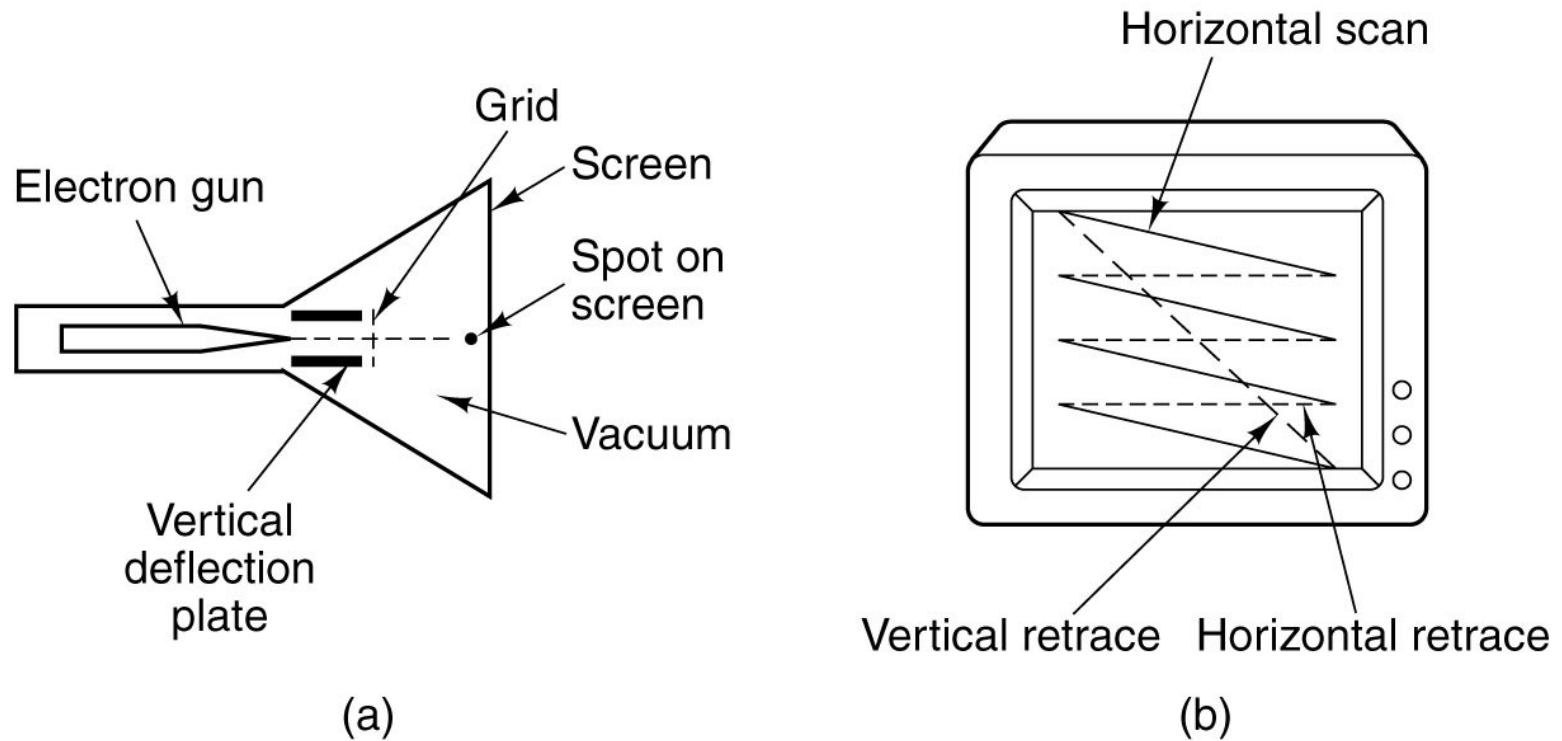
HD 1920 x 1080 (1080i,  
1080p)

# Byte per Pixel

- 1
  - $2^8$  levels or 256 colors (the levels for RGB)
- color palette
  - interpretation of 256 into whatever is fixed for the value "indexed" color
  - reduce video RAM by  $2/3$
- 3
  - one byte for each color -  $2^8$  levels or 256 each 24-bit RGB representation

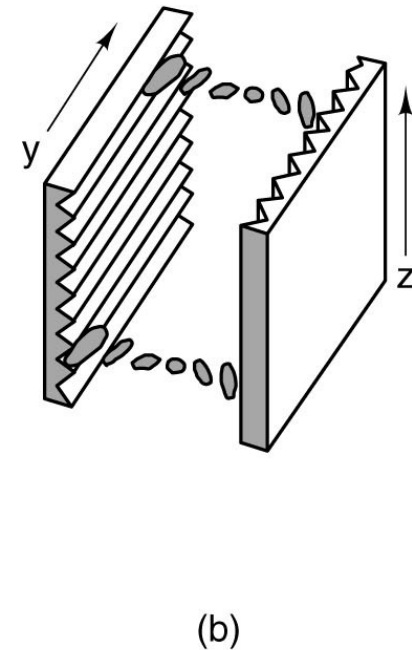
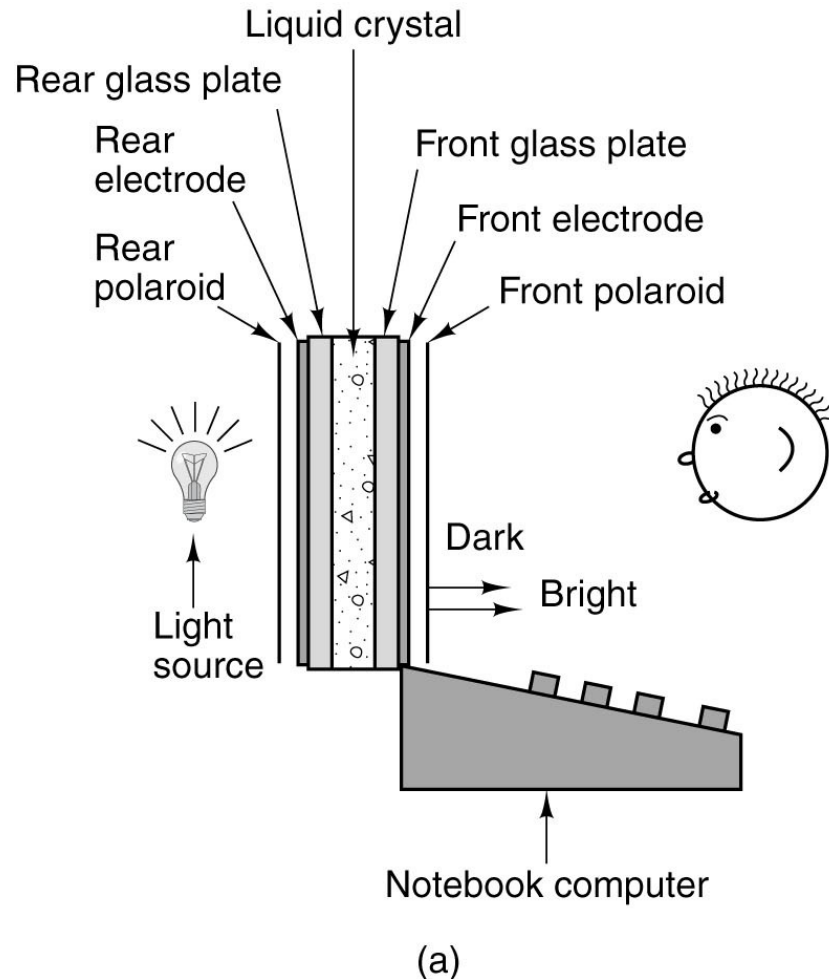
# CRT Monitor

(CRT = Cathode Ray Tube)



- (a) Cross section of a CRT
- (b) CRT scanning pattern

# Flat Panel Displays



- (a) The construction of an LCD screen.
- (b) The grooves on the rear and front plates are perpendicular to one another.

# Printer

- Matrix printer
- Inkjet printer
- Laser printer



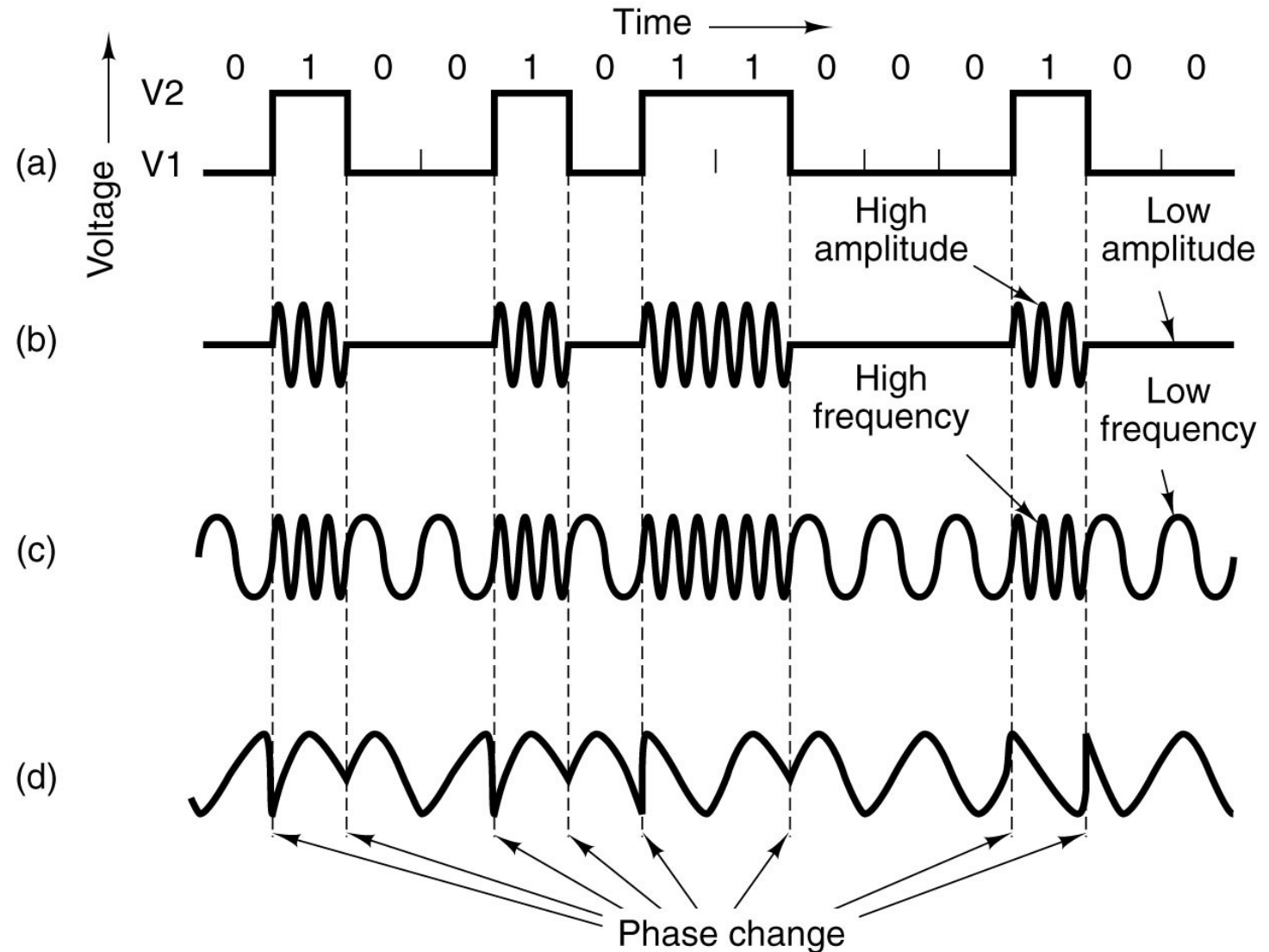
# Telecommunication

- Modem
- DSL
- Ethernet

# MODEM – Modulator/Demodulator

- Convert digital data 0's and 1's to audio tones.
- Allow data transmission on an analog phone line (audio) which allows tones from 50Hz to 3500 Hz
- Full-duplex vs Half-duplex
  - Full-duplex employs different frequencies

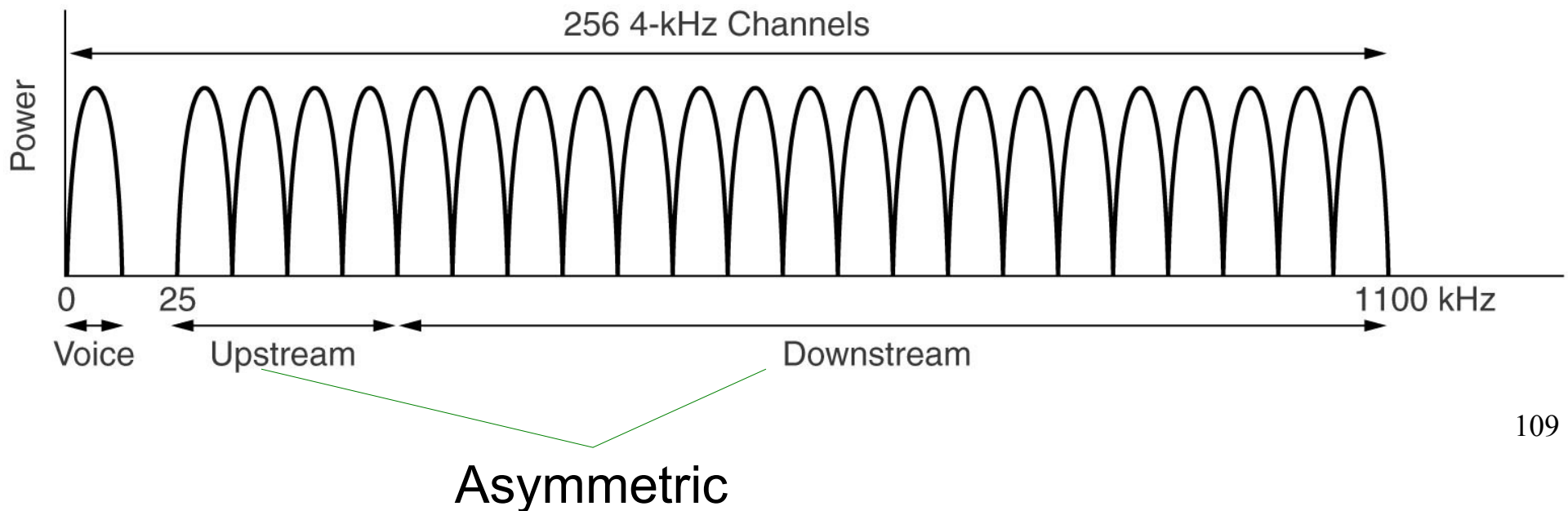
# MODEM



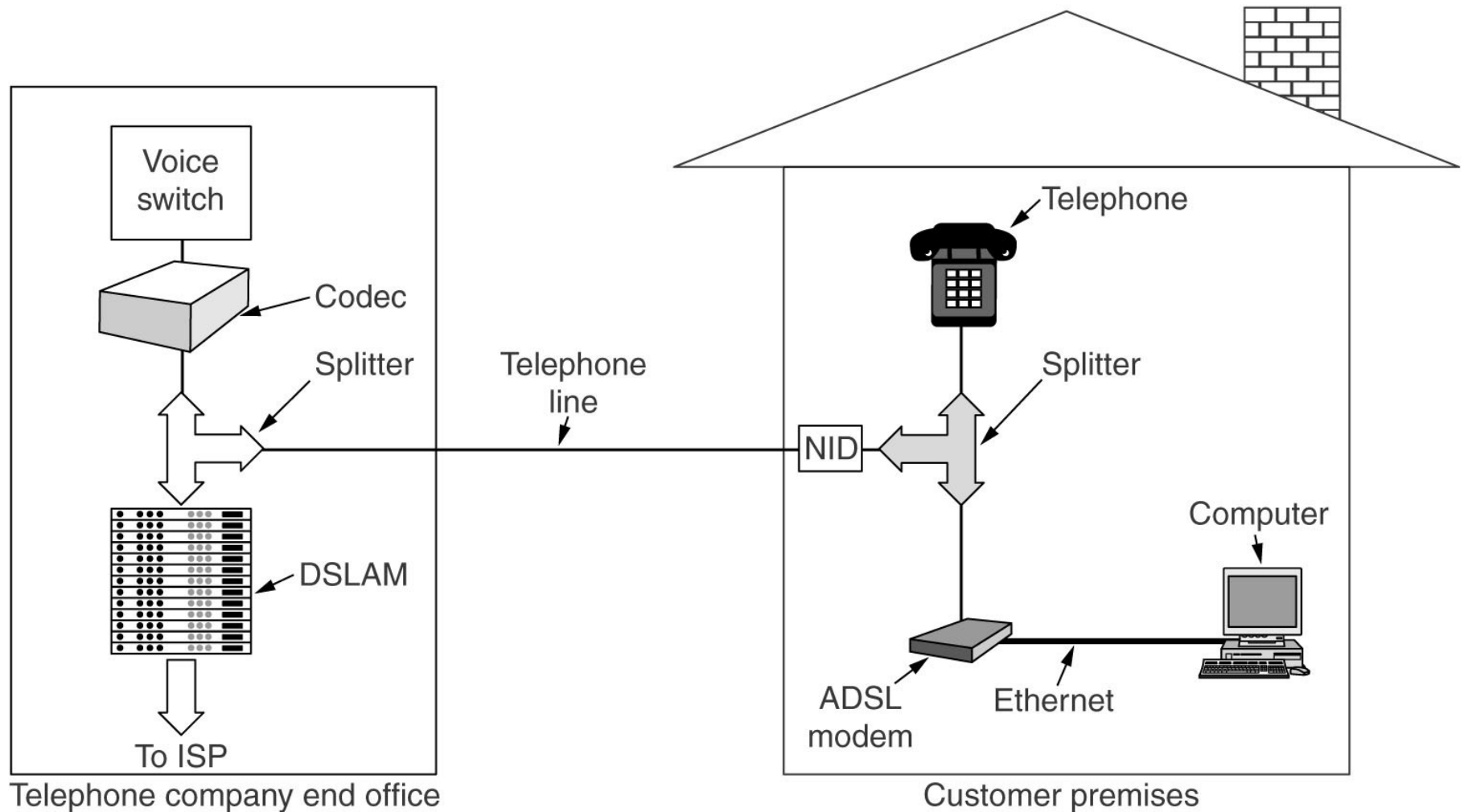
Transmission of the binary number 01001010000100 over a telephone line bit by bit. (a) Two-level signal. (b) Amplitude modulation. (c) Frequency modulation. (d) Phase modulation.

# Digital Subscriber Lines

- Low speed in dial-up connection due to filter in telco office (limit to 3000Hz)
- xDSL
  - Remove the filter
  - ADSL (Asymmetric DSL)

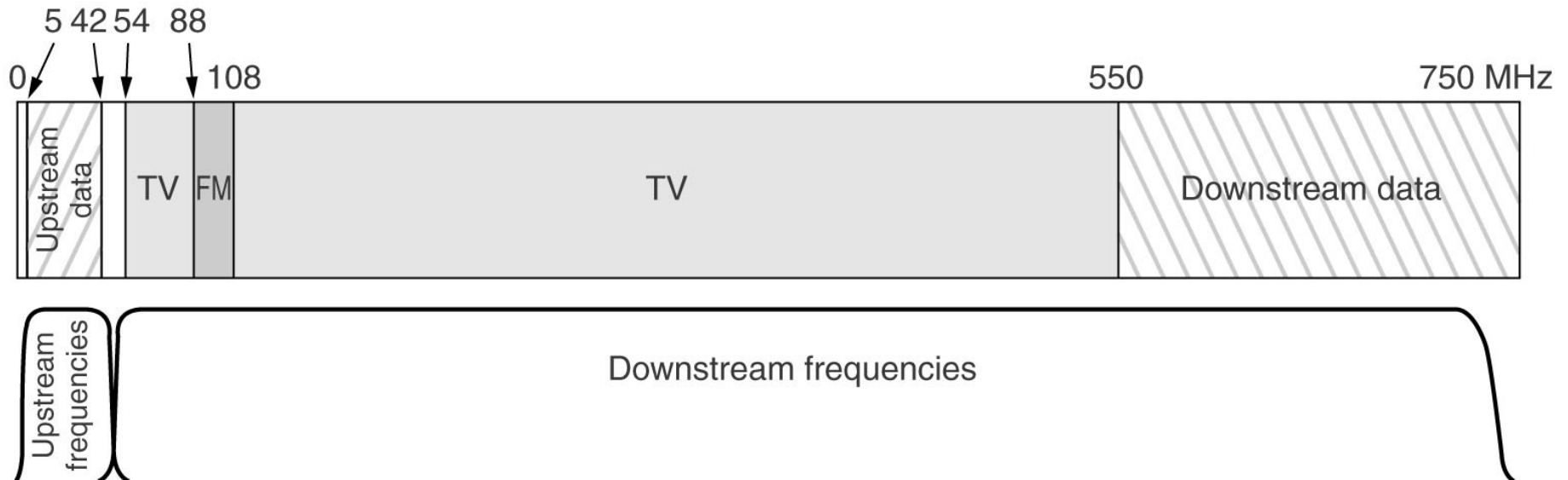


# Digital Subscriber Lines



A typical ADSL equipment configuration.

# Internet over Cable

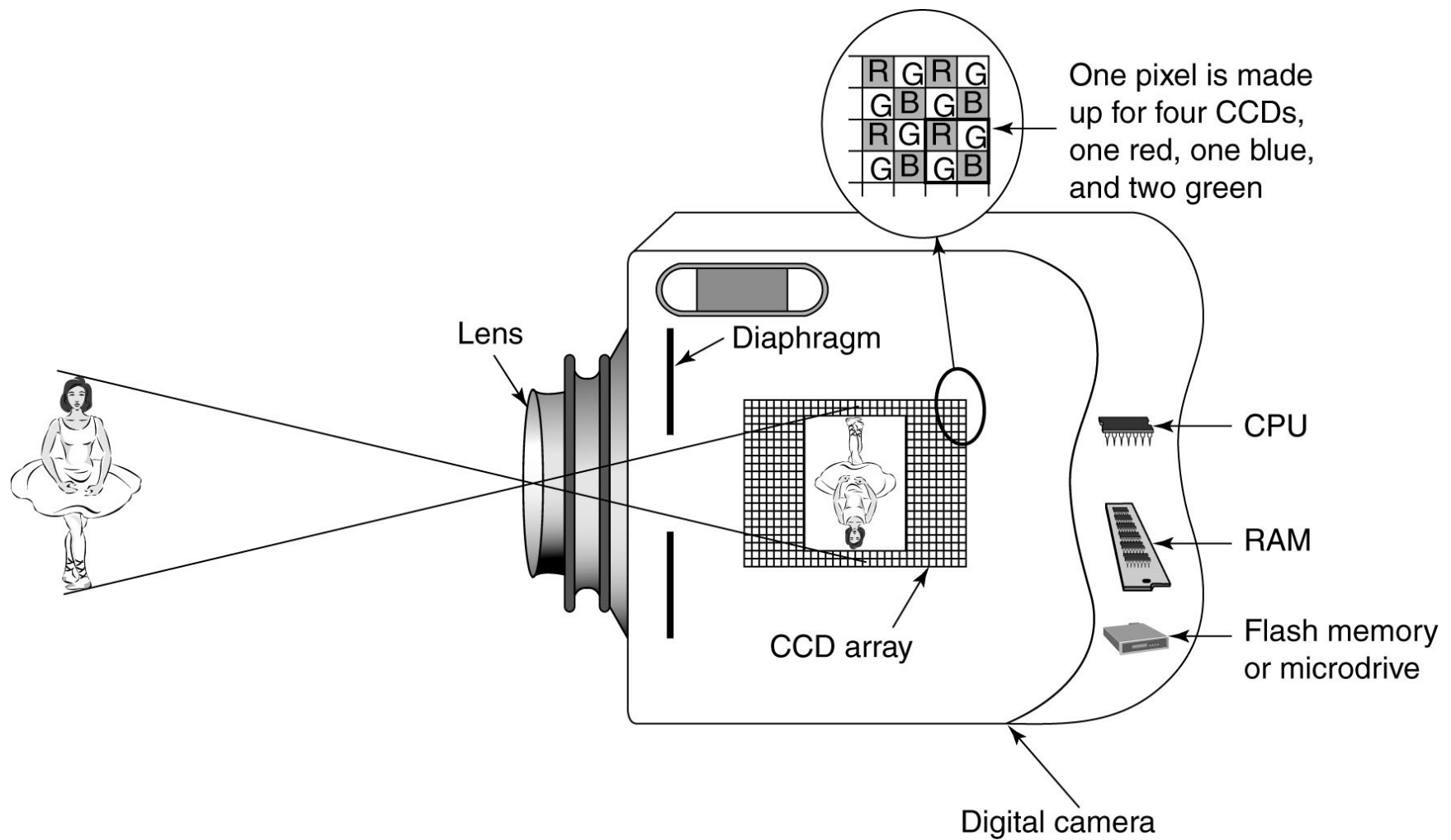


Frequency allocation in a typical cable TV system  
used for Internet access

Properties:

- Hundreds of user share the same cable to the headend
- Asymmetric connection also

# Digital Cameras



# Digital Cameras

- Four CCD make up on pixel
  - 4 is more convenience than 3
  - Eye is more sensitive to green light than to red or blue light
  - Camera maker claim 6 mpixel, means 1.5 mpixel in real



# Bonus

## Hard Disk Access Time Calculation

# Access Time (1)

- Seek time:
  - First the head must be position on the right track (radial distance).
  - Typical numbers: 5-15 msec.
  - “Avg. seek time”: not best case, not worst case, average!
  - Min. time – one track to the next
  - Max. time – outer track to inner track or vice versa

# Access Time (2)

- Rotational Latency:
  - The head must arrive at the correct sector on a track (rotational distance)

<b>RPM</b>	<b>rev/sec.</b>	<b>msec/rev</b>	<b>Avg. Rotation Latency (msec)</b>
3600	60	16.67	8.33
5400	90	11.11	5.56
7200	120	8.33	4.17
10800	180	5.56	2.78

# Access Time (3)

- Access time = Seek time + Rotational time
- Example:
  - A drive with 8.9 msec avg. seek time at 7200 RPM (Western Digital WD800JBRTL drive)
  - Access Time = 8.9 msec + 4.17 msec = 13.07 msec.

# Transfer Time

- Example: If there are 16 or 64 sectors on a track and the rotation rate is 7200 RPM, what is the maximum possible data transfer rate to read one sector on the disk.

7200 RPM  $\rightarrow$  8.33 msec/rev

For 16 sectors  $\rightarrow 8.33/16 = 0.52$  msec/sector

4096 bits/sector  $\rightarrow 4096/0.00052 = 7,876,923$  bits/sec  $\sim 0.98$  MB/sec.

7200 RPM  $\rightarrow$  8.33 msec/rev

For 64 sectors  $\rightarrow 8.33/64 = 0.13$  msec/sector

4096 bits/sector  $\rightarrow 4096/0.00013 = 31,507,692$  bits/sec  $\sim 3.94$  MB/sec.

**Note: this is the maximum burst rate  
doesn't include access time to get data**

# Transfer Time

How does cylinder help?

- Example: If there are 16 or 64 sectors on a track and the rotation rate is 7200 RPM, what is the maximum possible data transfer rate to read one sector on the disk.

0.98 MB/sec & 3.94 MB/sec.

- Note:
  - To support (or even get close to) ATA rates of 100-150 MB/sec and quoted WD rate, multiple tracks in a cylinder must be read simultaneously!
  - Multiply the rates defined by the number of surfaces simultaneously read, and higher rates can be achieved!

**Example:** Time to read one sector given as 10,800 RPM, 8.0 msec avg. seek time and 64 sectors per track.

Avg. Access Time = 8.0 msec + (5.56/2 msec) = 10.78 msec  
Transfer time = 5.56 msec/rev / 64 sectors/rev = 0.0869 msec  
Total Time = 10.78 msec + 0.0869 msec = 10.8669 msec

If this is the average data transfer rate we have,  
 $4096 \text{ bits} / 10.8669 \text{ msec} = 376,924 \text{ bits/sec}$  or *46.01 kB/sec* ... not very fast for “random sectors”

Fortunately, we usually read multiple sectors at a time (64).

Avg. Access Time = 8.0 msec + (2.78 msec) = 10.78 msec  
Transfer time = 5.56 msec/rev / all sectors/rev = 5.56 msec  
Total Time = 10.78 msec + 5.56 msec = 16.34 msec  
 $64 \times 4096 \text{ bits} / 16.34 \text{ msec} = 16,043,084 \text{ bits/sec}$  or *2.005 MB/sec* ... much better

# Copyright note

- Slides are adopted from lecture notes of Tanenbaum, Structured Computer Organization, Fifth Edition, (c) 2006 Pearson Education, Inc.