Lecture Note 1.
What is System Programming

August 27, 2021

Jongmoo Choi
Dept. of Software
Dankook University
http://embedded.dankook.ac.kr/~choijm

DKU
DANKOOK UNIVERSITY

Objectives

- **Understand the definition of system program**

- **Describe the types of system program**
  - ✓ Compilation system
  - ✓ Operating system
  - ✓ Runtime system

- **Hardware consideration**

- **Realize the concept of abstraction**

- **Reference: Chapter 1 in the CSAPP**

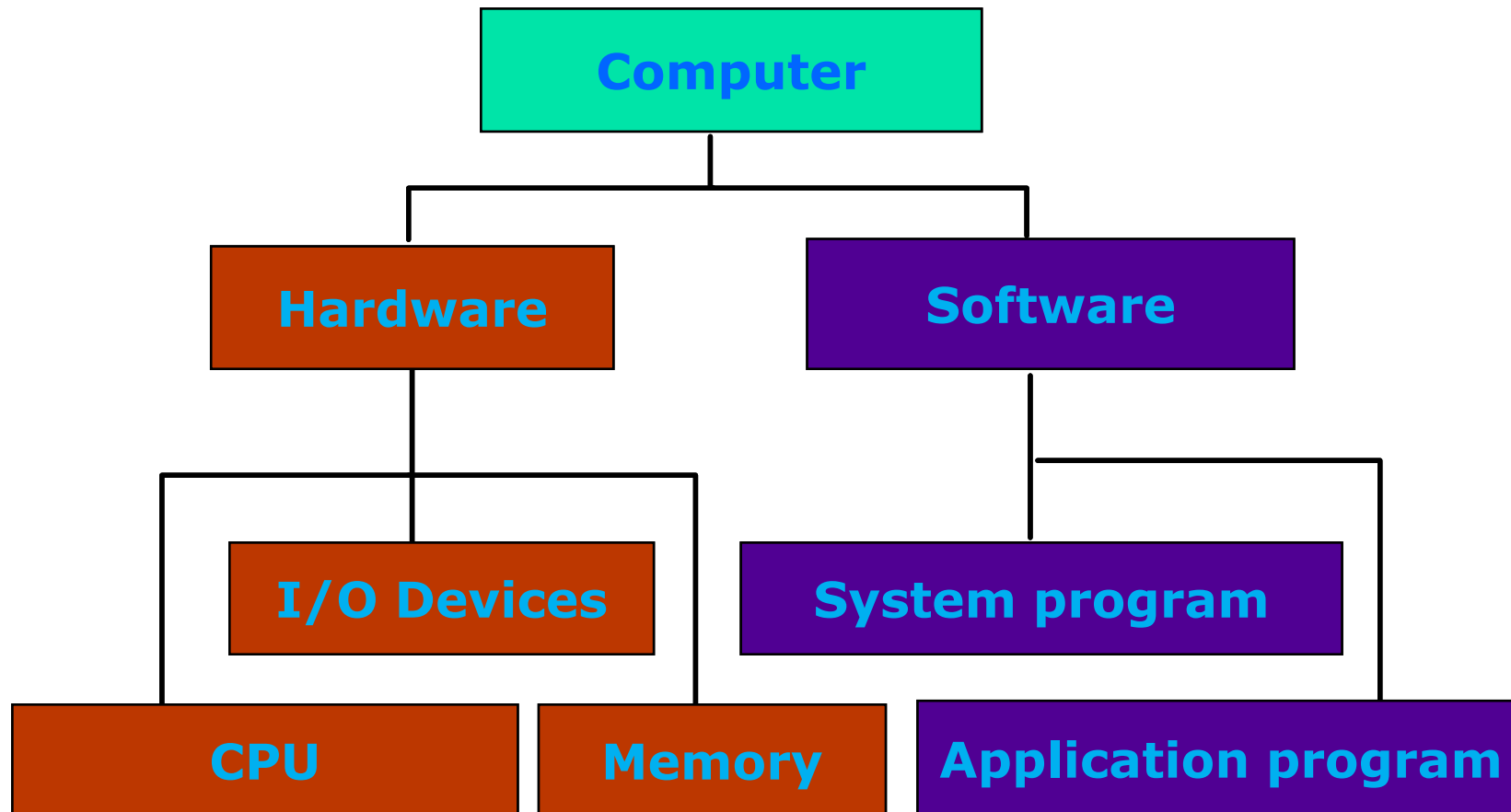CHAPTER **1**

A Tour of Computer Systems

(Source: CSAPP)

■ Computer organization

■ Hardware components: PC

Secondary storage

Main memory

Input device

Communication Device

CPU

Output device

Definition of System Program (3/8)

- **Hardware components: DRAM vs. Disk**
  - ✓ 1. Speed
  - ✓ 2. Capacity
    - ▪ Memory Hierarchy
  - ✓ 3. Volatility: Volatile vs. Non-volatile
    - ▪ Need to write data into disk explicitly for persistency (file I/O)
  - ✓ 4. Interface: Byte-unit interface vs. Sector-unit interface
    - ▪ Need to load a program from disk to RAM before execution (loading)

Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices
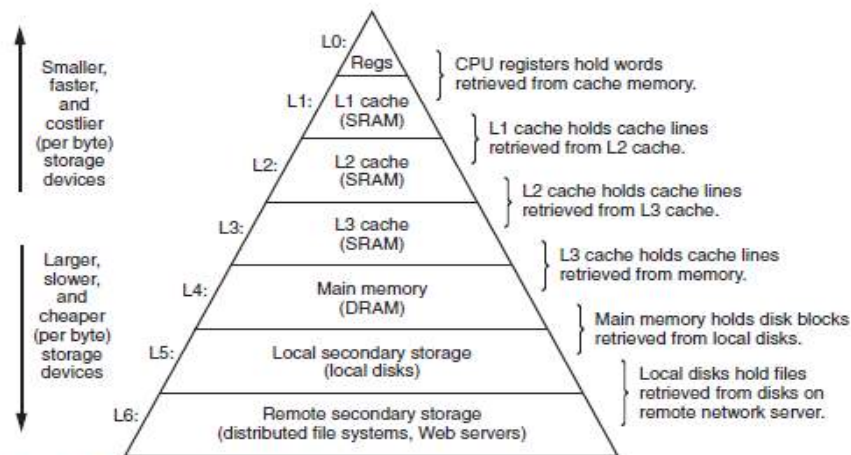
L0: Regs — CPU registers hold words retrieved from cache memory.
L1: L1 cache (SRAM) — L1 cache holds cache lines retrieved from L2 cache.
L2: L2 cache (SRAM) — L2 cache holds cache lines retrieved from L3 cache.
L3: L3 cache (SRAM) — L3 cache holds cache lines retrieved from memory.
L4: Main memory (DRAM) — Main memory holds disk blocks retrieved from local disks.
L5: Local secondary storage (local disks) — Local disks hold files retrieved from disks on remote network server.
L6: Remote secondary storage (distributed file systems, Web servers)

Figure 1.9 An example of a memory hierarchy.

START

HDD

Disk Memory
Secondary Memory
Operating System Loads
Program Into RAM

Primary Memory
Main Memory
RAM

intel CPU
Processor CPU
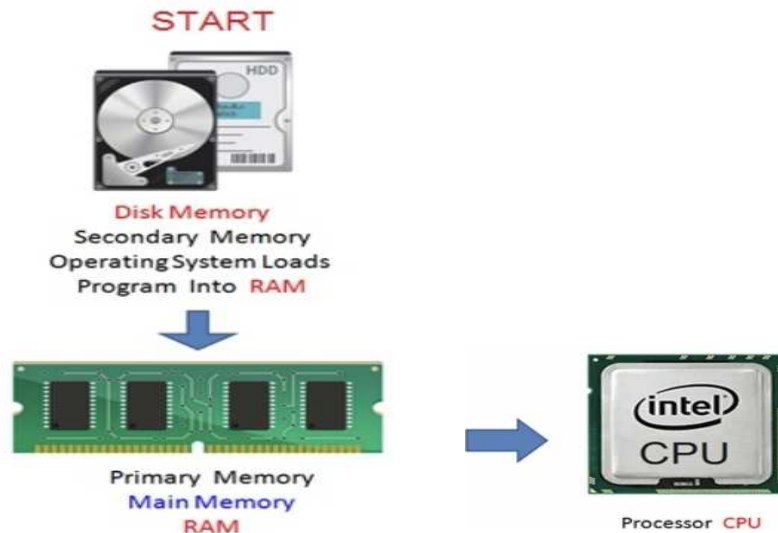
(Source: CSAPP)   **(Source: Google Image)** SYSPROG

Definition of System Program (4/8)

- **Hardware components: Smart Phone**
  - ✓ CPU: ARM based Multicore
  - ✓ Memory: LPDDR, SRAM
  - ✓ Storage: NAND flash
  - ✓ Input: Touch Screen, Sensors, Voice, Iris, ...
  - ✓ Output: LCD, LED, Sound, Buzzer, …
  - ✓ Communication
    - WLAN
    - LTE, CDMA, GSM
    - IrDA, Bluetooth, NFC
    - UART, USB
    - …

(Source: Google Image)

- **Hardware components: Requirements for Mobile devices**
  - ✓ Power Saving
    - ▪ Make use of RICS CPU instead of CISC CPU
      - · RISC: Reduced Instruction Set Computing ➔ Small Instructions ➔ Compact CPU internal ➔ Consume less Power
    - ▪ Make use of LPDDR (Low-Power DDR) instead of General DRAM
      - · LPDDR: Reduce power by using lower voltage and less refreshing
  - ✓ Portability
    - ▪ Make use of Flash memory instead of Disk
      - · Lightweight, Shock resistance
  - ✓ User friendliness
    - ▪ Make use of diverse input, output and communication devices

	DDR3/DDR3L	LPDDR3
전원 전압	1.5V/1.35V	1.2V
Configurations	x4, x8, x16	x16, x32
Address/Command 신호	SDR Command 와 Address pin이 분리되어 있음.	DDR Command/Address pin을 공유
Data 1 pin당 최대 전송 속도 (Mbps)	2133	1866* (spec.은 2133까지 정의)
메모리 내부 온도 센서	없음	있음
Refresh를 각 bank 에 개별적으로 적용 (PASR)	지원가능(optional)	지원
Deep Power Down 모드	없음	있음

(Source: http://egloos.zum.com/donghyun53/v/4125772)

Definition of System Program (6/8)

- **Software components**
  - ✓ Application program vs. System program
    - ▪ Application program: how to do a specific job

    ```
    #include <stdio.h>

    int main()
    {
        printf("hello, world₩n");
    }
    ```

    - ▪ System program: address the following issues
      - · How to run this application program on CPU?
      - · What is the role of printf()?
      - · How the string is displayed on Monitor?
      - · How this program can be executed with other programs concurrently?
      - · What are the differences between local and global variables?
      - · What kinds of techniques can be applied to enhance the performance of this program?

- **Software components: System program**
  - ✓ How to run a program on CPU?
    - object, binary, compiler, assembler, loader, ...
  - ✓ What is the role of printf()?
    - library, linker, ...
  - ✓ How the string is displayed on Monitor?
    - device driver, file system, ...
  - ✓ How a program can be executed with other programs concurrently?
    - process, scheduler, context switch, IPC (Inter process communication), ...
  - ✓ What are the differences between local and global memory?
    - data, stack, heap, virtual memory, buddy system, ...
  - ✓ What kind of techniques can be applied to enhance the performance of a program?
    - compiler optimization (loop unrolling, reordering), CPU optimization (pipeline, superscalar, out-of-order execution), …
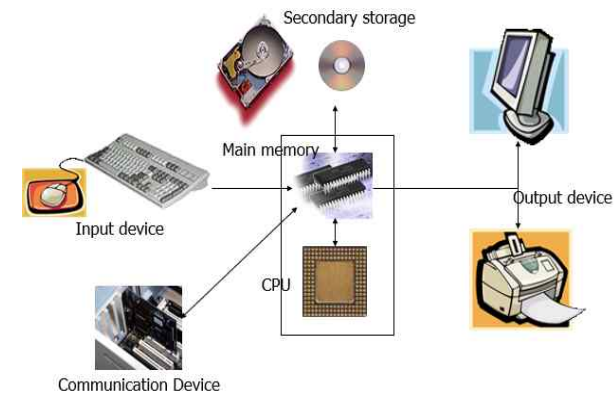
Definition of System Program (8/8)

- **Software components: System program**
  - ✓ Supporting computing environments for application programs (Support Interfaces such as commands, library functions and system calls)
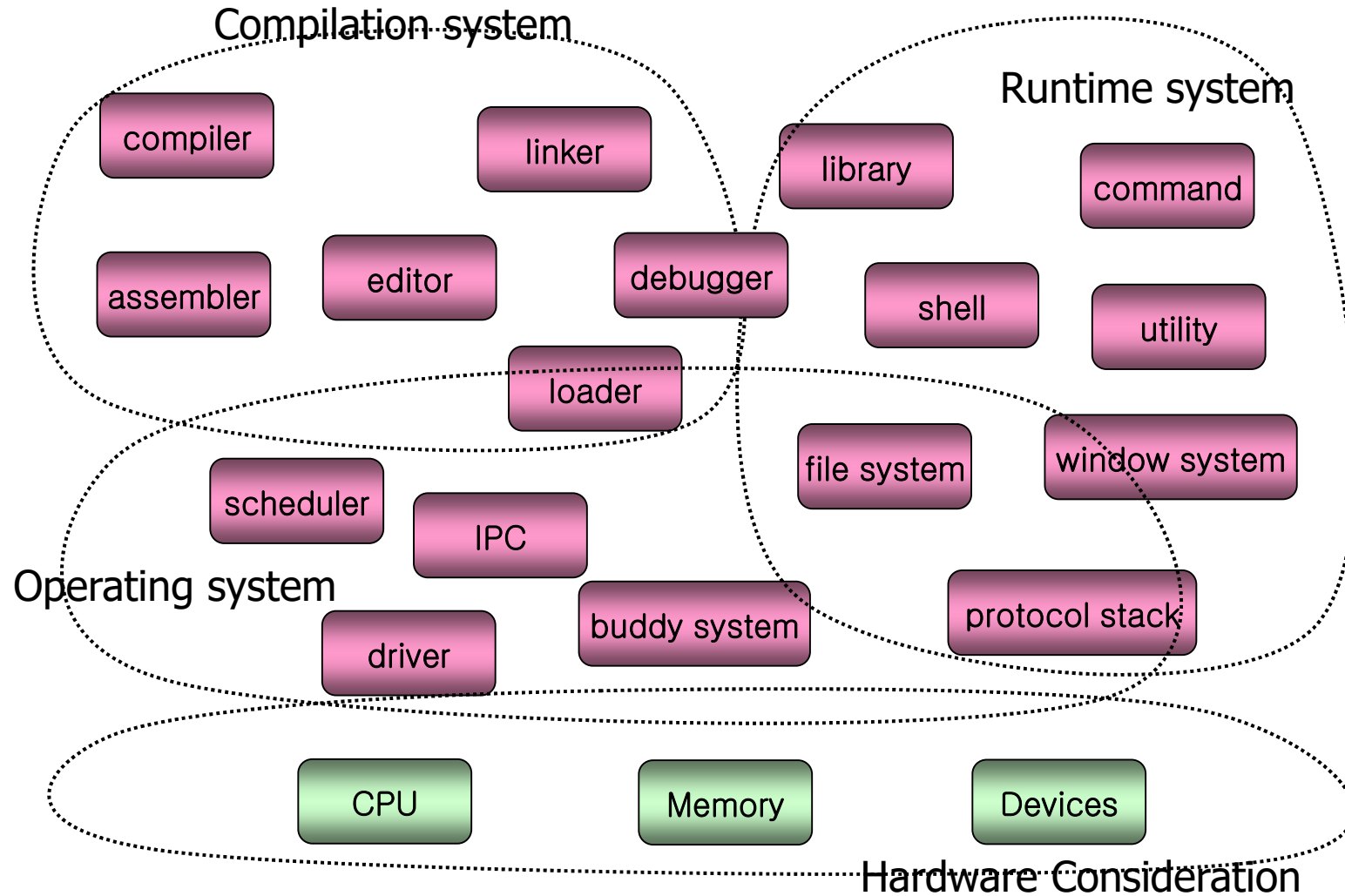 - ✓ Strongly related to hardware (hardware management)

  - ✓ Abstraction
    - CPU and Task (Process)
    - DRAM and Virtual memory
    - Disk and File
    - Device and Driver
    - Machine level language and High level language
    - Untrusted and Trusted Domain
    - ...

Types of System Program

- Classification

Compilation system

compiler	linker	

Runtime system

library

command

assembler	editor	debugger

shell

utility

loader

file system

window system

scheduler

IPC

Operating system

buddy system

protocol stack

driver

CPU	Memory	Devices

Hardware Consideration

Compilation System (1/5)

- Concept: Language Hierarchy

High-level Language

C = A + B;

Assembly Language

```
...
movl    0x8049388, %eax
addl    0x8049384, %eax
movl    %eax, 0x804946c
...
```
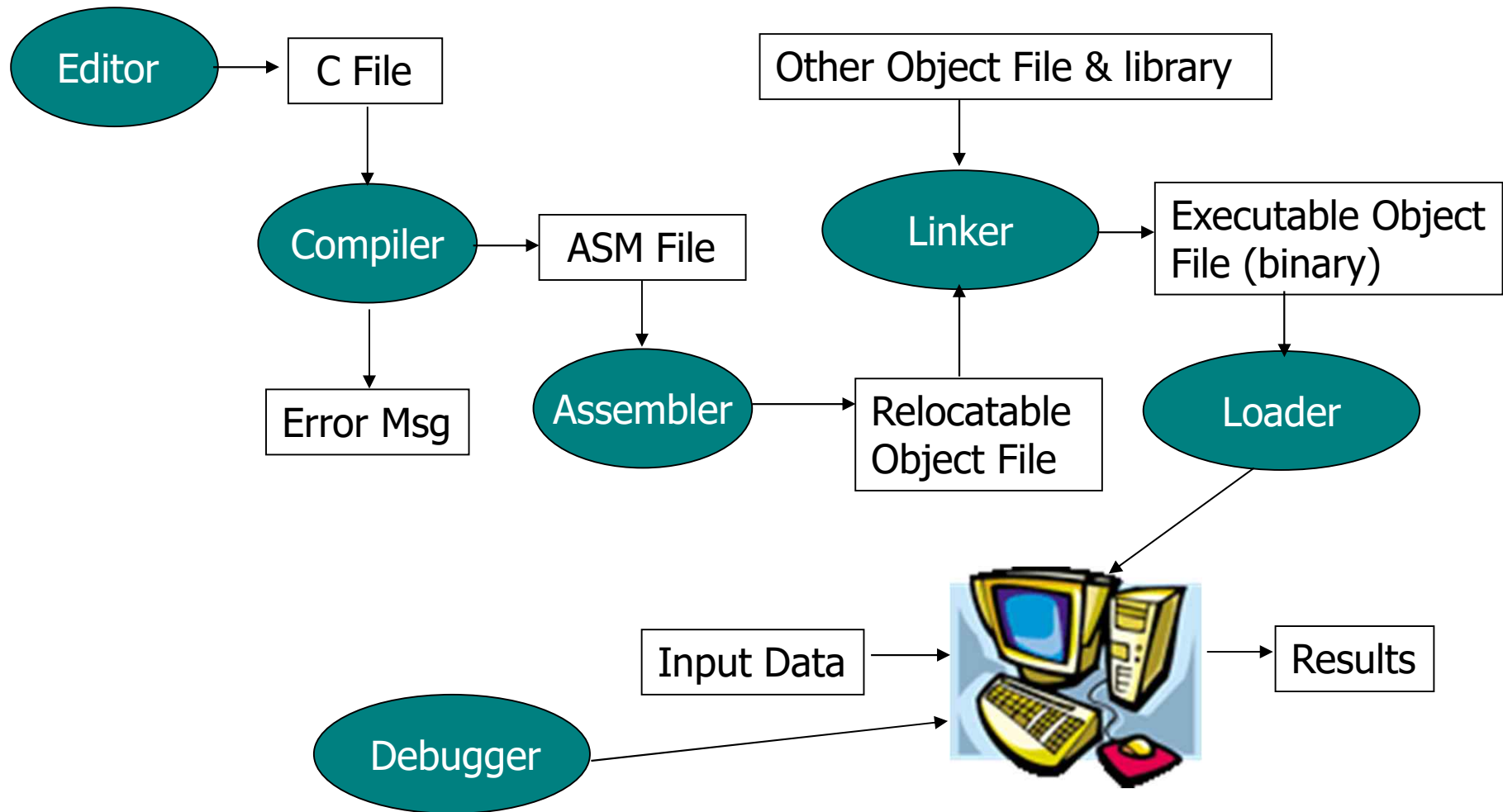
Machine Language (Binary code)

```
...
00a1 8893 0408
0305 8493 0408
00a3 6c94 0408
...
```

■ Overall structure and 6 key components

```
Editor ──→ C File                    Other Object File & library
              │                                    │
              ↓                                    ↓
          Compiler ──→ ASM File       Linker ──→ Executable Object
              │              │           ↑         File (binary)
              ↓              ↓           │              │
          Error Msg     Assembler ──→ Relocatable   Loader
                                      Object File
```

Input Data ──→ [computer] ──→ Results

Debugger

■ Quiz

- ✓ 1. Explain why loader is required in a computer system. (hint: using the difference between Disk and DRAM).

- ✓ 2. Discuss why the hardware components of Smartphone are different from those of PC even though they are same with the viewpoint of computer architecture (3 reasons).
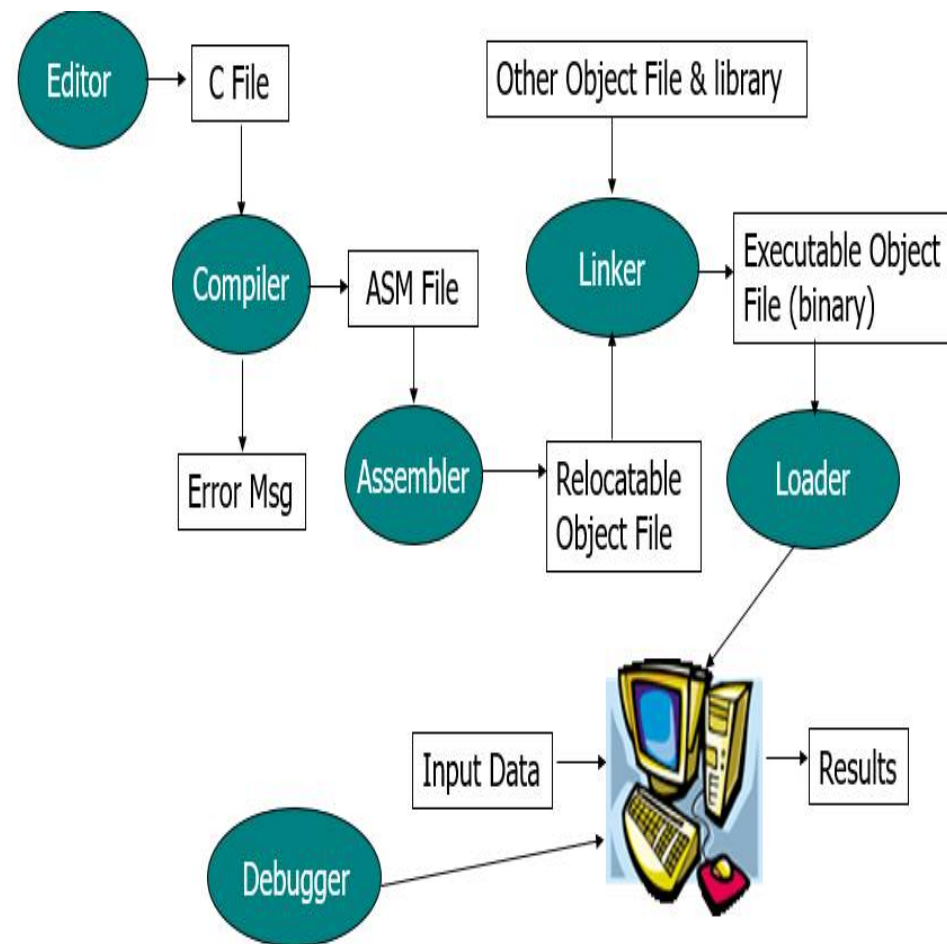
- ✓ Due: until 6 PM Friday of this week(3ʳᵈ, September)
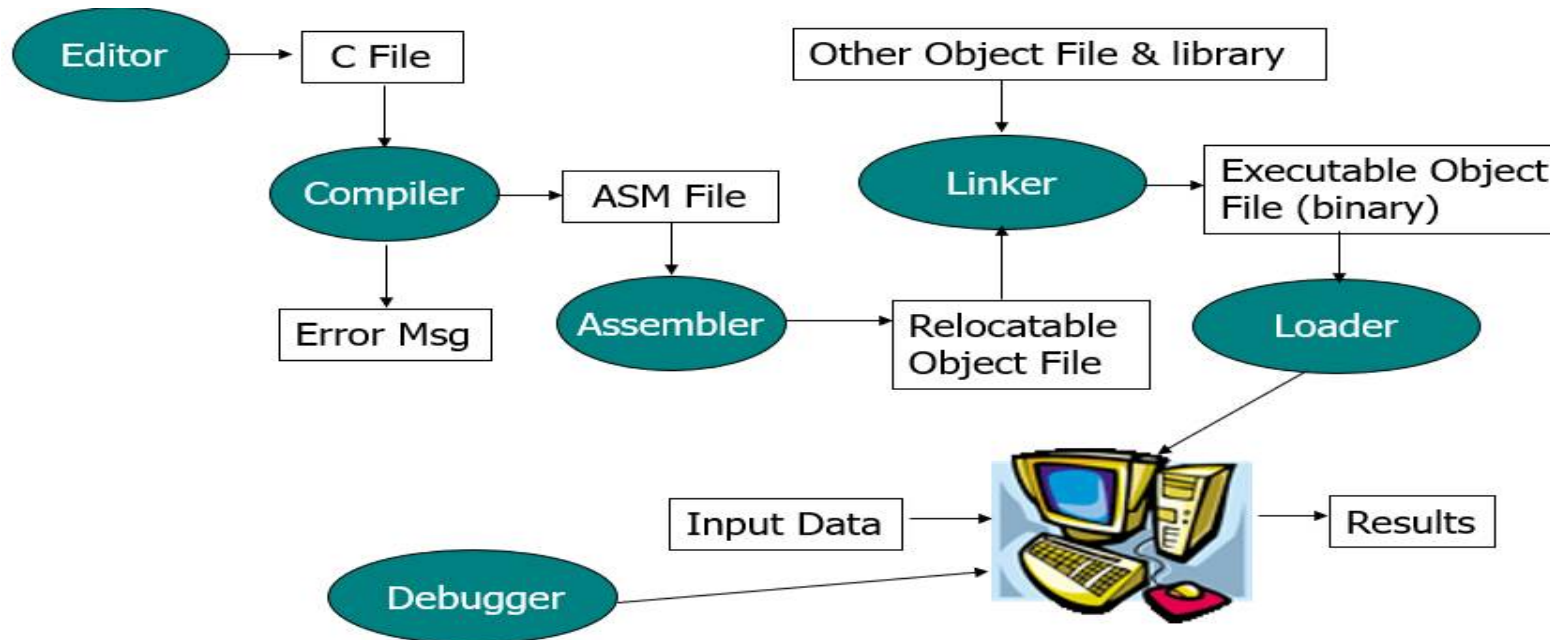
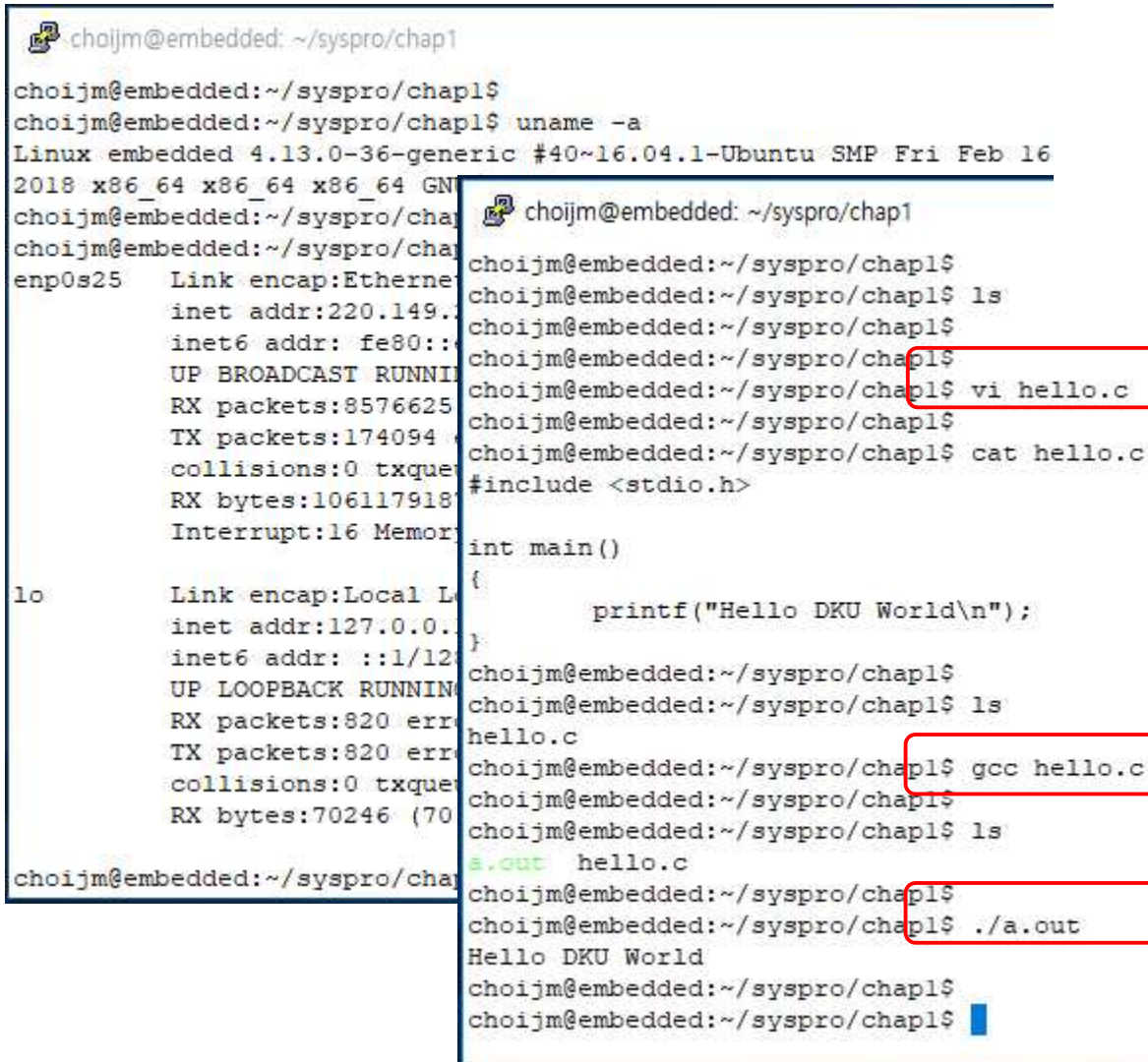- **Relation between Language Hierarchy and Overall Structure**
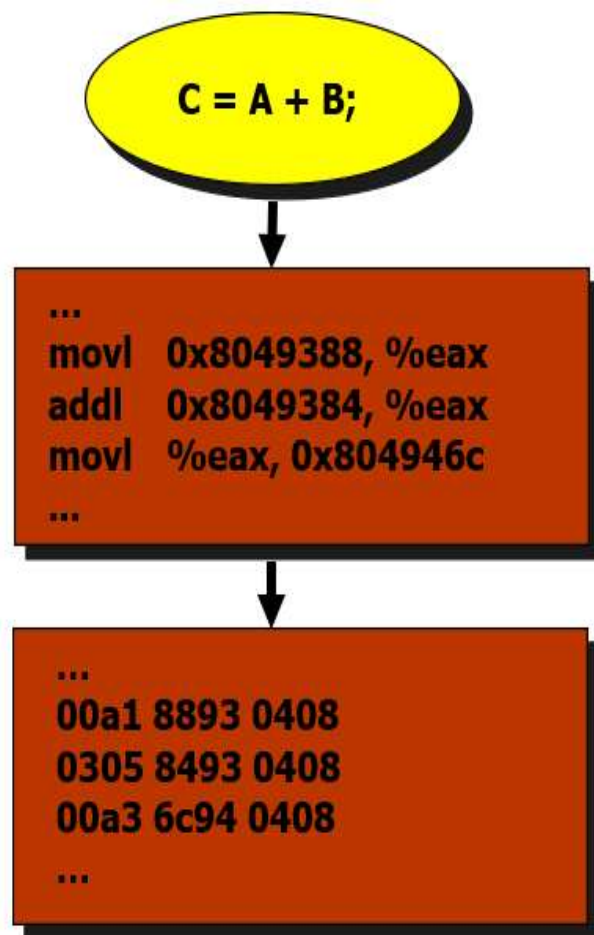
(Language hierarchy)

(Compilation system)

Compilation System (4/5)

- **Example in Linux**

## ■ Example in Linux: details

- ✓ Location of collect2, crt1.o, … depend on gcc


```
choijm@embedded: ~/sypro/chap2
choijm@embedded:~/sypro/chap2$
choijm@embedded:~/sypro/chap2$ vi hello.c
choijm@embedded:~/sypro/chap2$ ls
hello.c
choijm@embedded:~/sypro/chap2$
choijm@embedded:~/sypro/chap2$ gcc -S hello.c
choijm@embedded:~/sypro/chap2$ ls
hello.c  hello.s
choijm@embedded:~/sypro/chap2$
choijm@embedded:~/sypro/chap2$ as -o hello.o hello.s
choijm@embedded:~/sypro/chap2$ ls
hello.c  hello.o  hello.s
choijm@embedded:~/sypro/chap2$
choijm@embedded:~/sypro/chap2$ /usr/lib/gcc/i486-linux-gnu/3.4.6/collect2 /usr/lib/i
86-linux-gnu/crt1.o /usr/lib/i386-linux-gnu/crti.o /usr/lib/i386-linux-gnu/crtn.o /us
/lib/gcc/i486-linux-gnu/3.4.6/crtbegin.o /usr/lib/gcc/i486-linux-gnu/3.4.6/crtend.o h
llo.o -lc -dynamic-linker /lib/ld-linux.so.2
choijm@embedded:~/sypro/chap2$ ls
a.out  hello.c  hello.o  hello.s
choijm@embedded:~/sypro/chap2$
choijm@embedded:~/sypro/chap2$ ./a.out
Hello DKU World
choijm@embedded:~/sypro/chap2$
choijm@embedded:~/sypro/chap2$
```
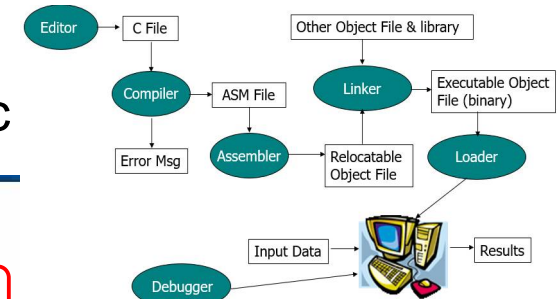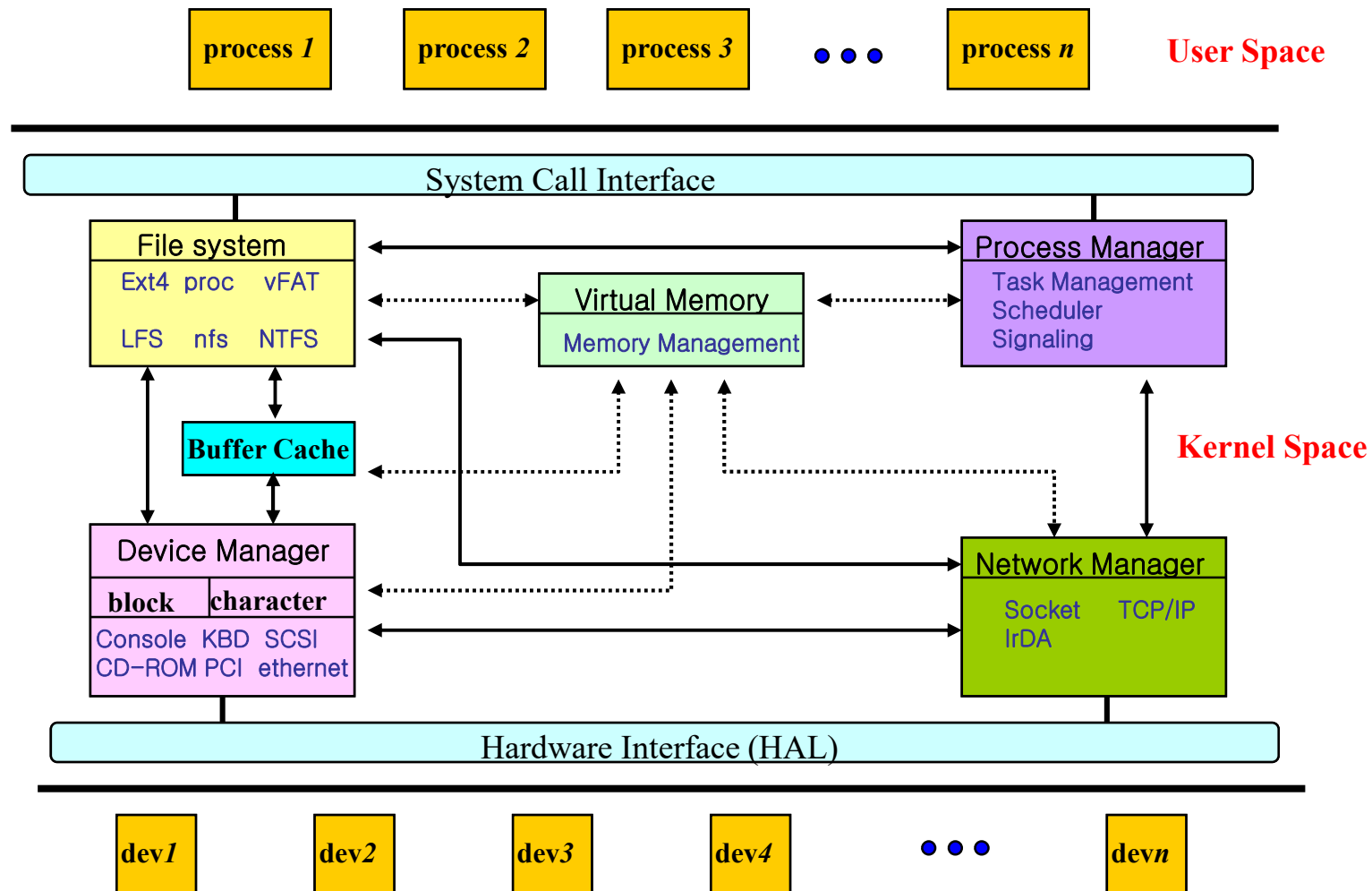
☞ **What are the differences btw hello.c and hello.s?**
☞ **What are the differences btw hello.o and a.out?**

Operating System (1/15)

- **Overall structure and 7 key components**

process _1_ process _2_ process _3_ • • • process _n_   **User Space**

System Call Interface

File system
Ext4 proc   vFAT
LFS   nfs   NTFS

Virtual Memory
Memory Management

Process Manager
Task Management
Scheduler
Signaling

Buffer Cache

Kernel Space

Device Manager
block | **character**
Console KBD SCSI
CD−ROM PCI ethernet

Network Manager
Socket   TCP/IP
IrDA

Hardware Interface (HAL)

dev_1_ **dev_2_** **dev_3_** **dev_4_** • • • **dev_n_**

(Source: Linux Kernel Internals)
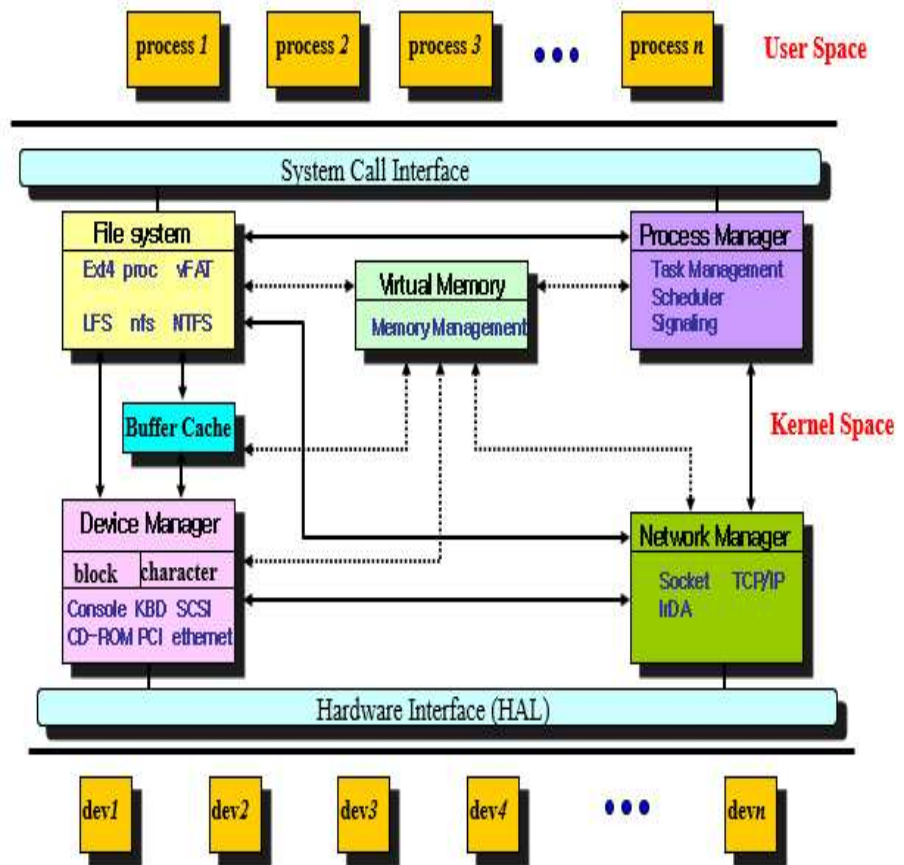
- **Relation between hardware component and overall structure**
  - ✓ OS: a resource manager ➔ abstract HW resources into logical ones
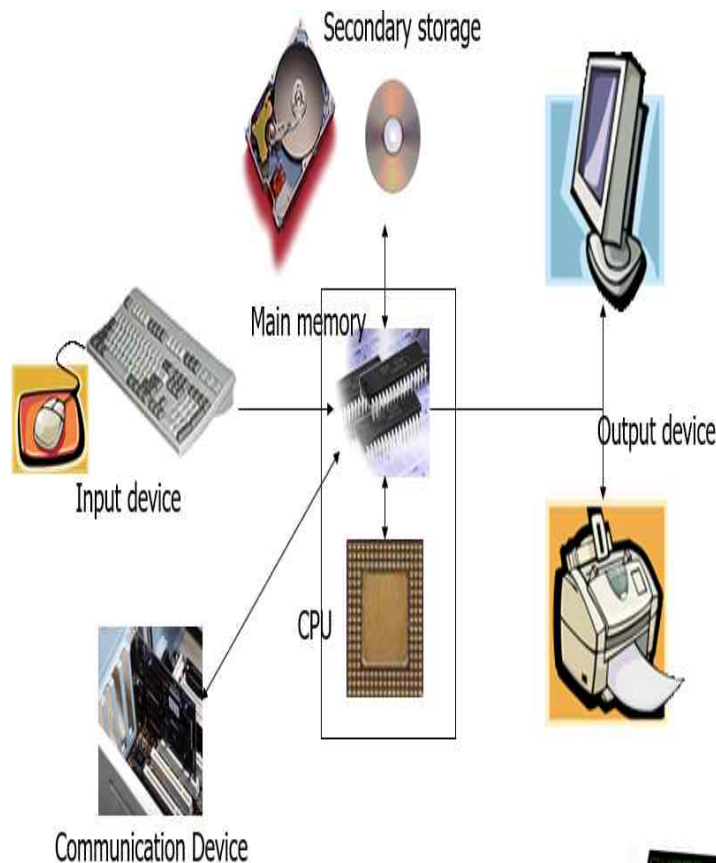
(Physical resources)

(Logical resources)

- **Behaviors: 1) initial state**
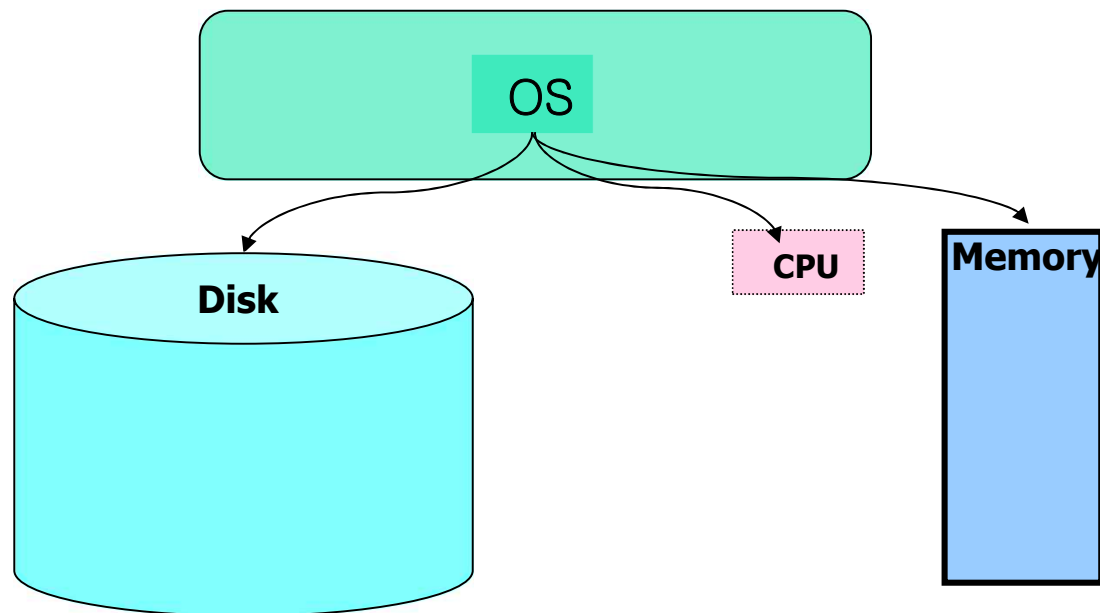
- Behaviors: 2) create a file (user's viewpoint)
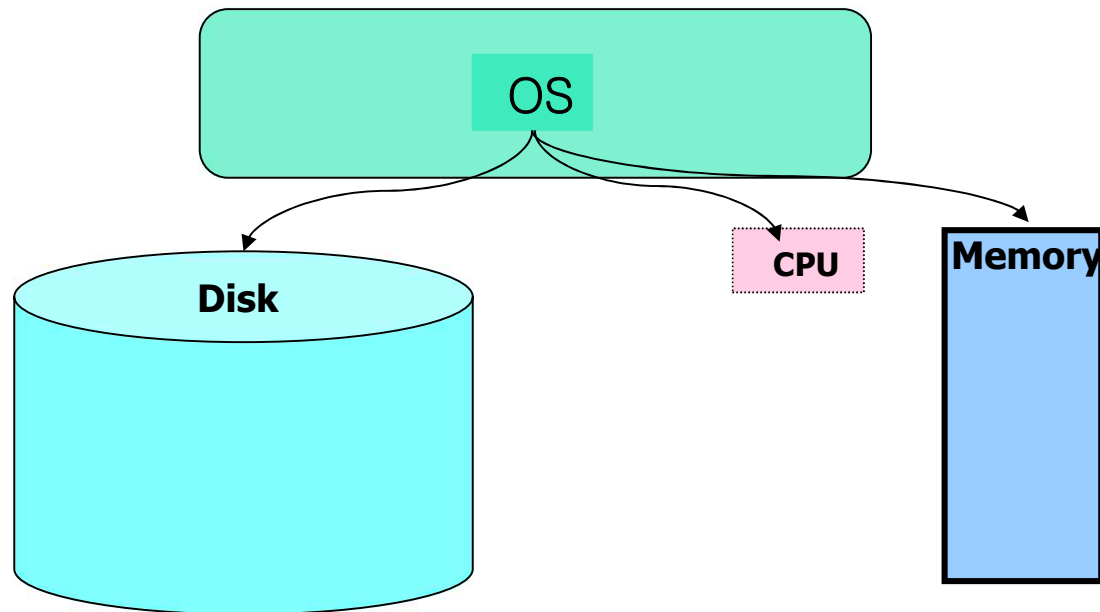
vi test.c

```
int sum = 0;

int main()
{
        int i;

        for (i=0; i<10;i++)
            sum += i;

        printf("%d", sum);
}
```
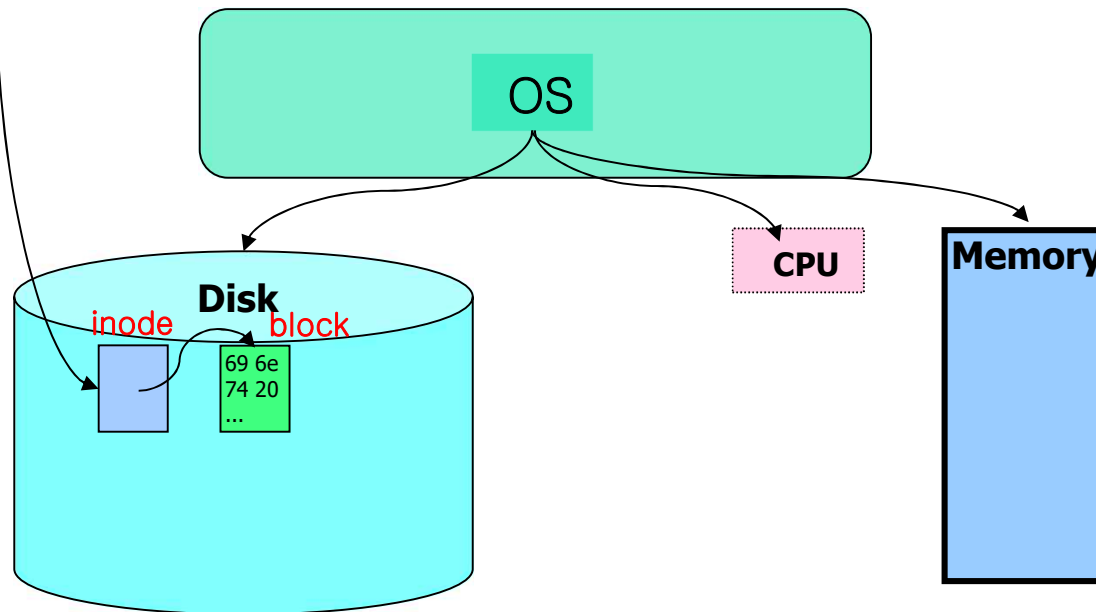
OS

Disk

CPU

Memory

■ Behaviors: 2) create a file (system's viewpoint)

vi test.c

```
int sum = 0;

int main()
{
    int i;

    for (i=0; i<10;i++)
        sum += i;

    printf("%d", sum);
}
```

#	i	n	c	l	u	d	e	\<sp\>	\<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46

h	\>	\n	\n	i	n	t	\<sp\>	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123

\n	\<sp\>	\<sp\>	\<sp\>	\<sp\>	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108

l	o	,	\<sp\>	w	o	r	l	d	\	n	"	)	;	\n	}
108	111	44	32	119	111	114	108	100	92	110	34	41	59	10	125

Figure 1.2   The ASCII text representation of hello.c.

(Source: CSAPP)

OS

CPU

Memory

Disk

inode   block

```
69 6e
74 20
...
```

- **Behaviors: 3) compile the file (user's viewpoint)**

vi test.c

```
int sum = 0;

int main()
{
        int i;

        for (i=0; i<10;i++)
            sum += i;

        printf("%d", sum);
}
```

compile →

a.out

```
.data
    .align 4
    .type   sum,@object
    .size    sum,4
.text
.global main
    .type   main, @func
main:
    pushl    %ebp
    ...
    movl    −4(%ebp), %eax
    addl     %eax, sum
    ...
```

OS

Disk

```
69 6e
74 20
...
```

CPU

Memory
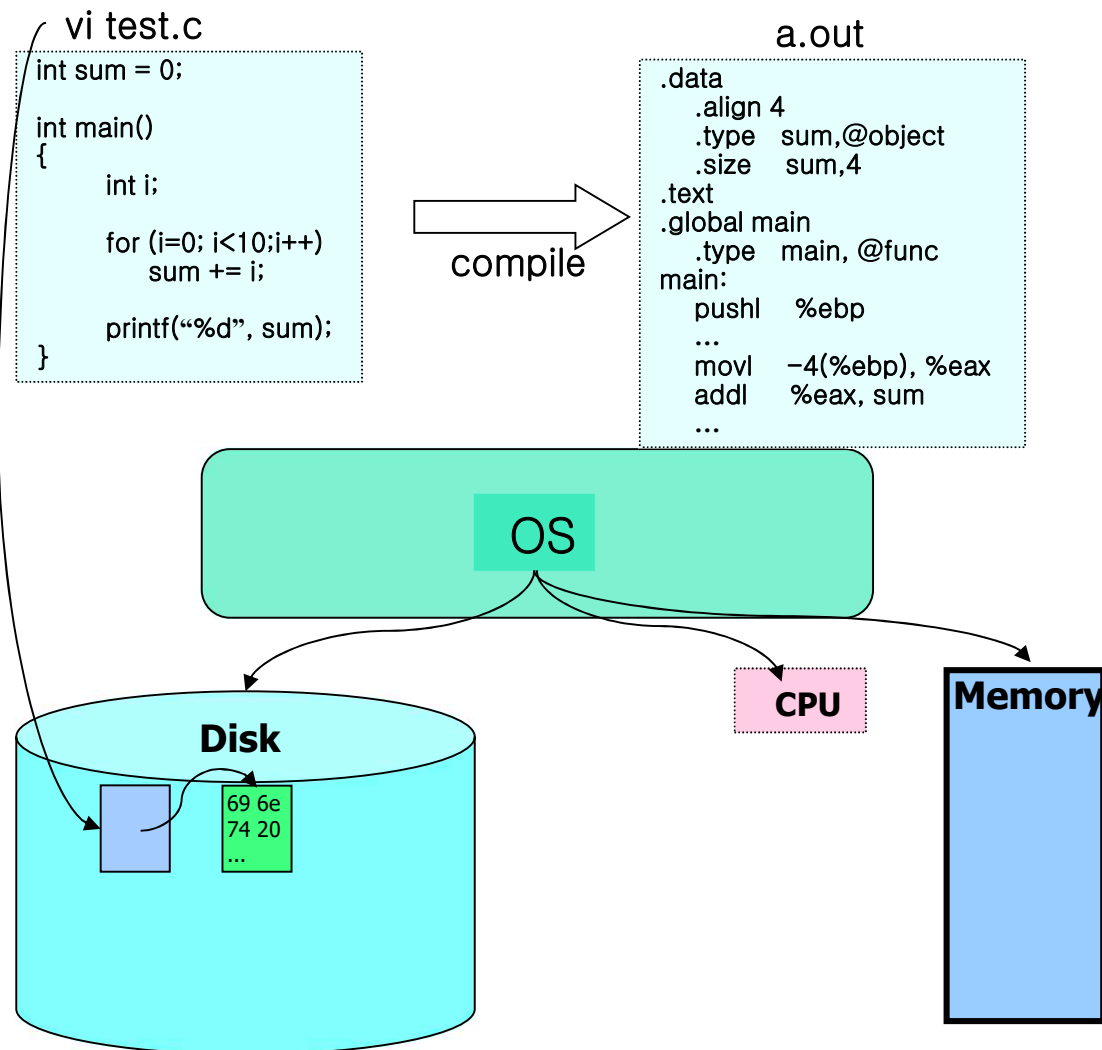
- **Behaviors: 3) compile the file (system's viewpoint)**

vi test.c

```
int sum = 0;

int main()
{
    int i;

    for (i=0; i<10;i++)
        sum += i;

    printf("%d", sum);
}
```

compile

a.out

```
.data
    .align 4
    .type   sum,@object
    .size    sum,4
.text
.global main
    .type   main, @func
main:
    pushl    %ebp
    ...
    movl    −4(%ebp), %eax
    addl     %eax, sum
    ...
```

OS

CPU

Memory

Disk

69 6e
74 20
...

inode

block
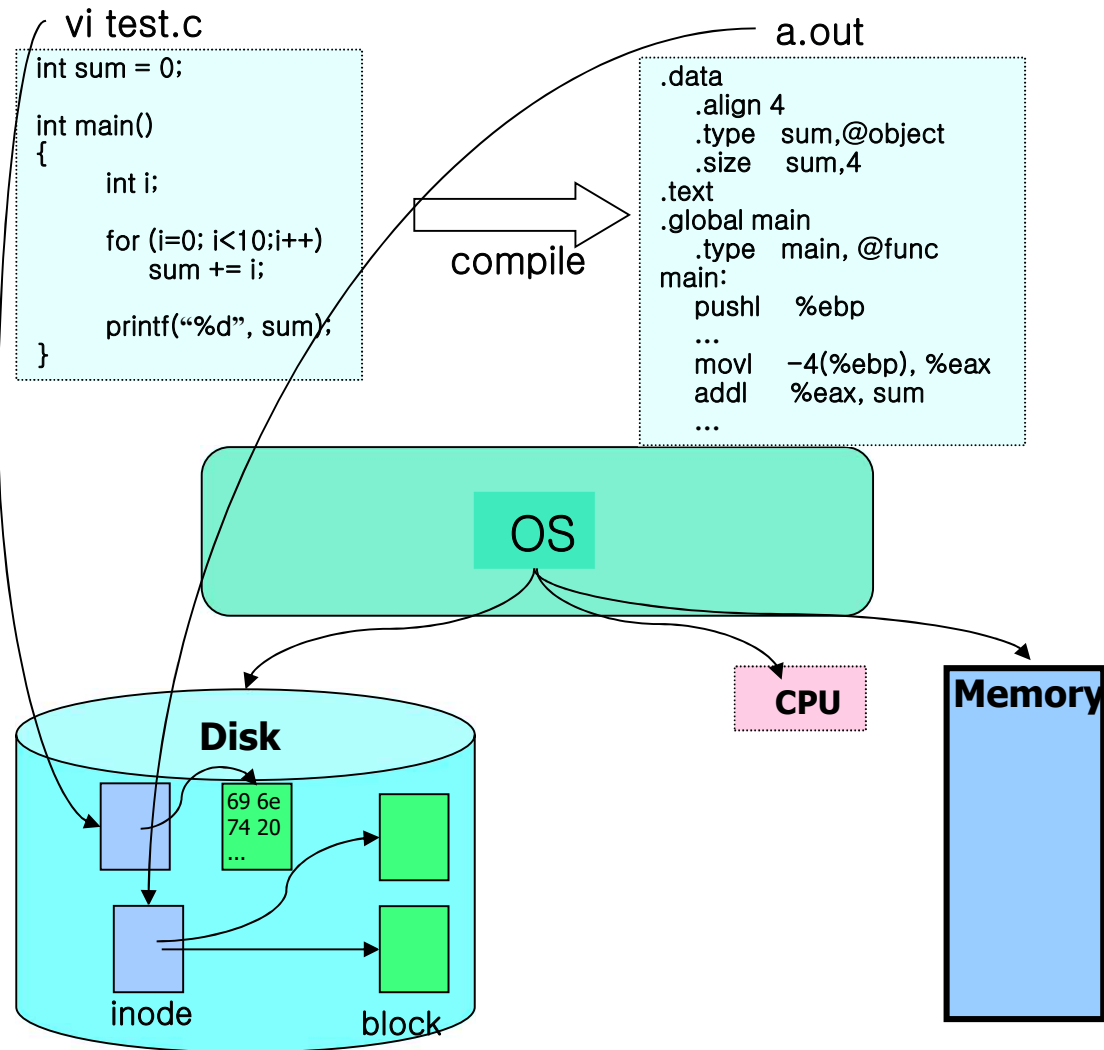
■ **Behaviors: 4) execute the a.out (user's viewpoint)**

vi test.c

```
int sum = 0;

int main()
{
        int i;

        for (i=0; i<10;i++)
            sum += i;

        printf("%d", sum);

}
```

compile ⟹

a.out

```
.data
    .align 4
    .type   sum,@object
    .size    sum,4
.text
.global main
    .type    main, @func
main:
    pushl    %ebp
    ...
    movl    −4(%ebp), %eax
    addl     %eax, sum
    ...
```

execute ⟹

run a.out ➔
We can see 45
on the Monitor

OS

CPU

Memory

Disk

```
69 6e
74 20
...
```
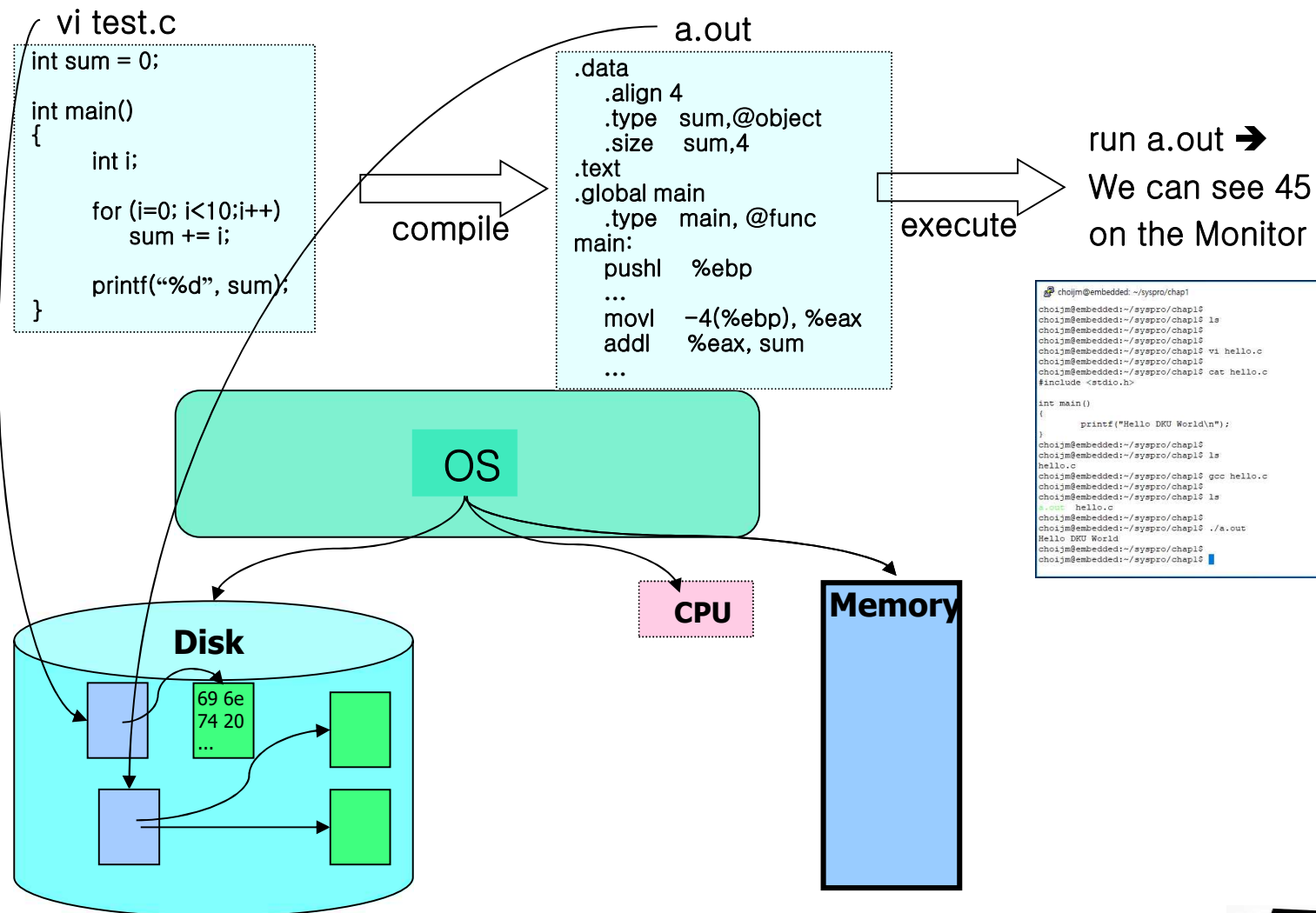
- **Behaviors: 4) execute the a.out (system's viewpoint)**
  - ✓ To run a.out, OS first loads it into memory

vi test.c

```
int sum = 0;

int main()
{
        int i;

        for (i=0; i<10;i++)
            sum += i;

        printf("%d", sum);

}
```

compile ⟹

a.out

```
.data
    .align 4
    .type    sum,@object
    .size    sum,4
.text
.global main
    .type    main, @func
main:
    pushl    %ebp
    ...
    movl    −4(%ebp), %eax
    addl    %eax, sum
    ...
```

execute ⟹

run a.out ➔
We can see 45
on the Monitor

OS

CPU

Memory

Disk

69 6e
74 20
...

inode   block

page
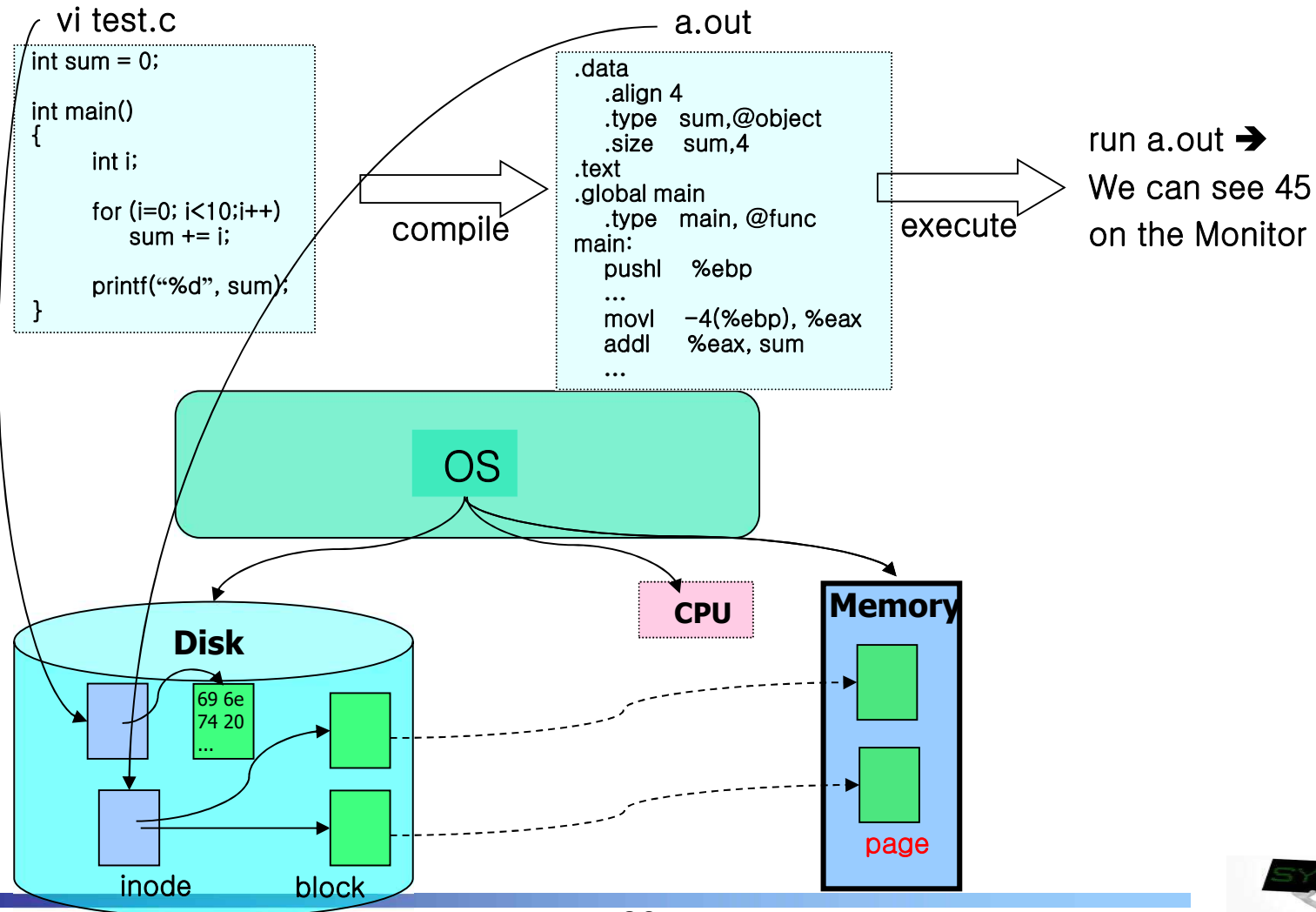
- **Behaviors: 4) execute the a.out (system's viewpoint)**
  - ✓ Then, OS makes a new process (active object)

vi test.c

```
int sum = 0;

int main()
{
        int i;

        for (i=0; i<10;i++)
            sum += i;

        printf("%d", sum);

}
```

compile

a.out

```
.data
    .align 4
    .type    sum,@object
    .size    sum,4
.text
.global main
    .type    main, @func
main:
    pushl     %ebp
    ...
    movl     −4(%ebp), %eax
    addl      %eax, sum
    ...
```

execute

run a.out ➔
We can see 45
on the Monitor

OS

CPU

Memory

new process

Disk

69 6e
74 20
...

inode   block

segment/page table

page

27
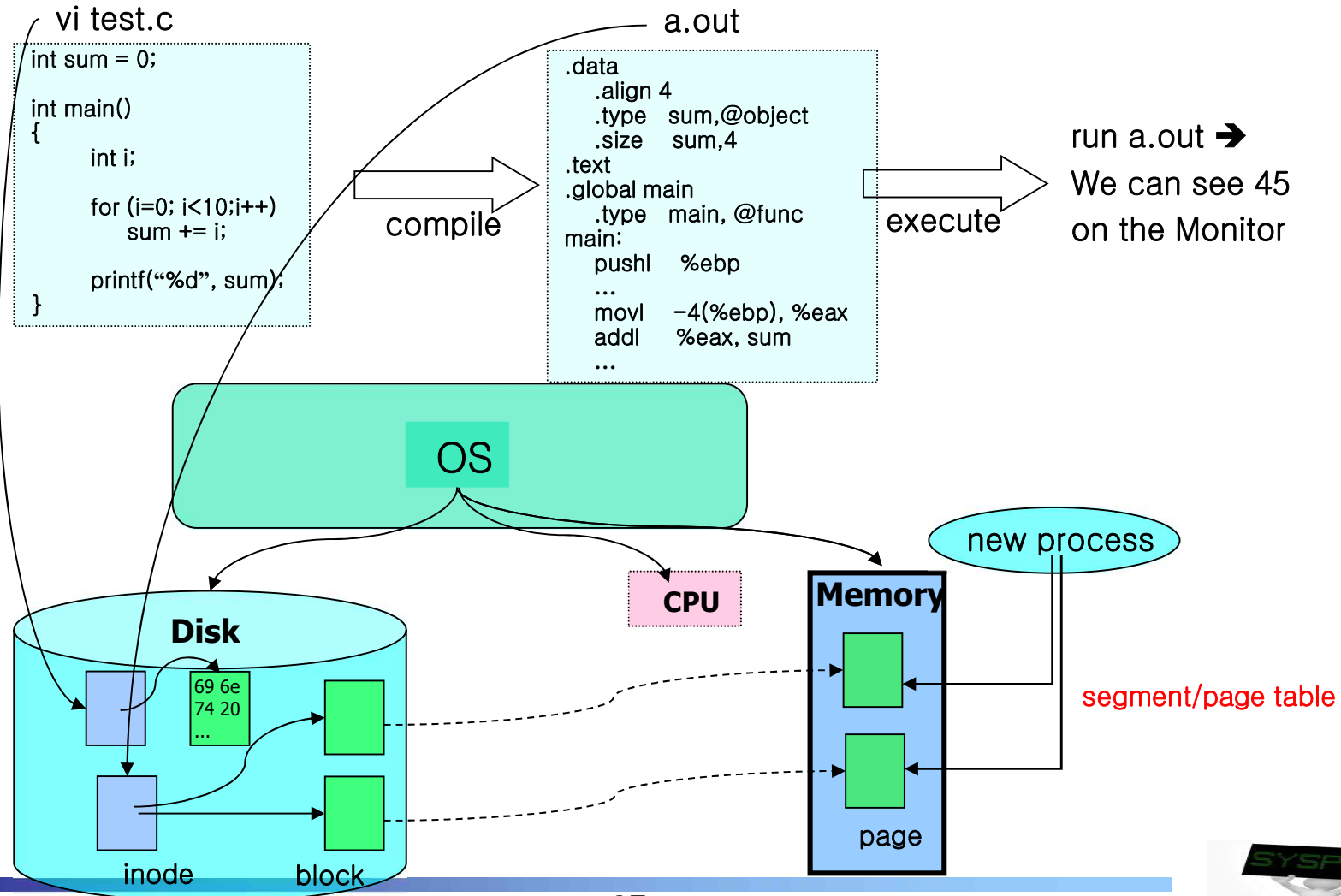
- **Behaviors: 4) execute the a.out (system's viewpoint)**
  - ✓ Then, OS schedule the process

vi test.c

```
int sum = 0;

int main()
{
        int i;

        for (i=0; i<10;i++)
            sum += i;

        printf("%d", sum);

}
```

compile

a.out

```
.data
    .align 4
    .type    sum,@object
    .size    sum,4
.text
.global main
    .type    main, @func
main:
    pushl    %ebp
    ...
    movl    −4(%ebp), %eax
    addl    %eax, sum
    ...
```

execute

run a.out ➔
We can see 45
on the Monitor

OS

CPU

new process

CPU

scheduling

Memory

Disk

inode

```
69 6e
74 20
...
```

segment/page table
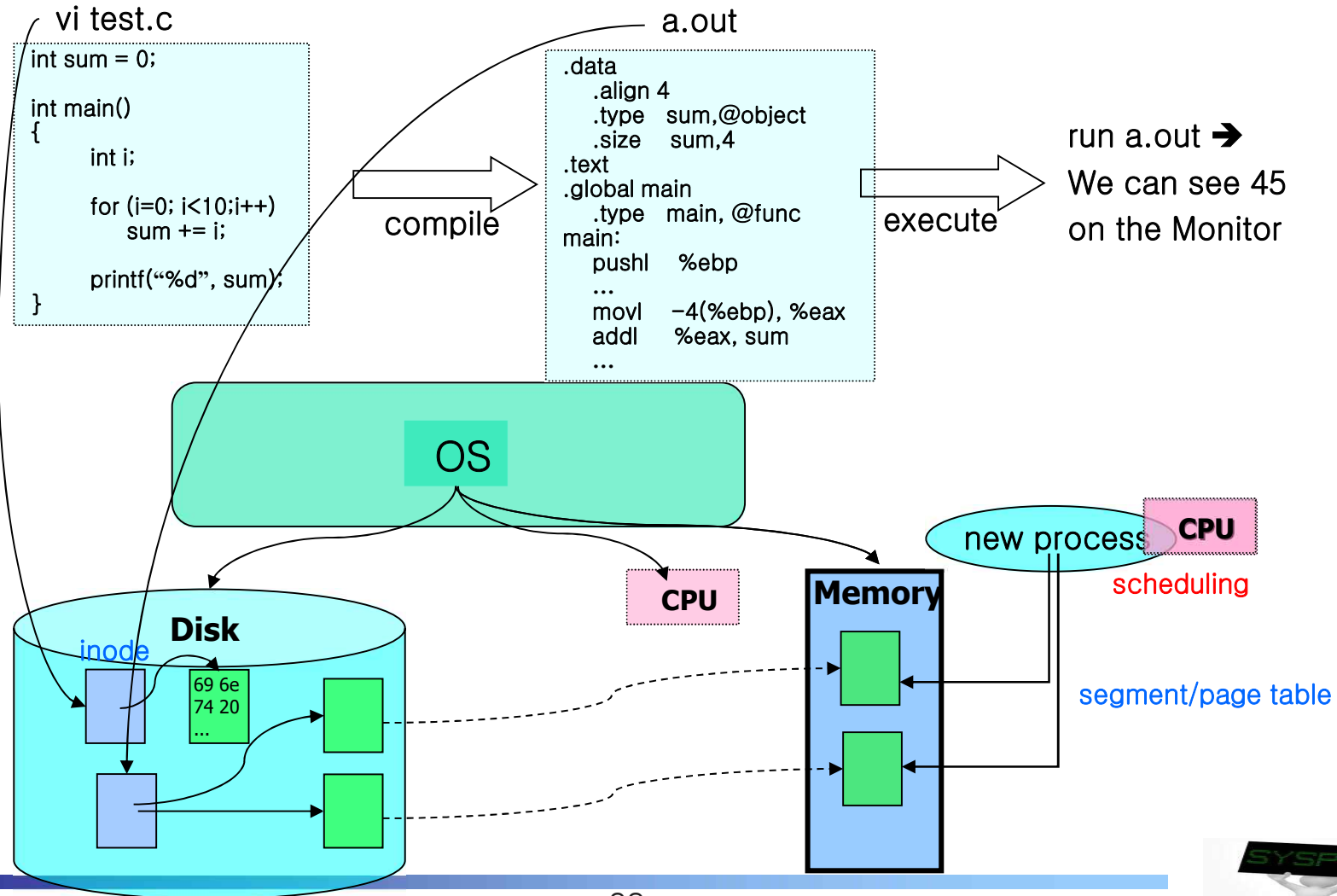
- **Behaviors: 4) execute the a.out (system's viewpoint)**
  - ✓ Actually there are multiple processes managed by scheduler

vi test.c

```
int sum = 0;

int main()
{
    int i;

    for (i=0; i<10;i++)
        sum += i;

    printf("%d", sum);

}
```

compile

a.out

```
.data
    .align 4
    .type   sum,@object
    .size   sum,4
.text
.global main
    .type   main, @func
main:
    pushl   %ebp
    ...
    movl    −4(%ebp), %eax
    addl    %eax, sum
    ...
```

execute

run a.out ➔
We can see 45
on the Monitor

OS

CPU

Memory

prev process

prev process

new process

CPU

Time−sharing system

Disk

inode

```
69 6e
74 20
...
```

segment/page table

29
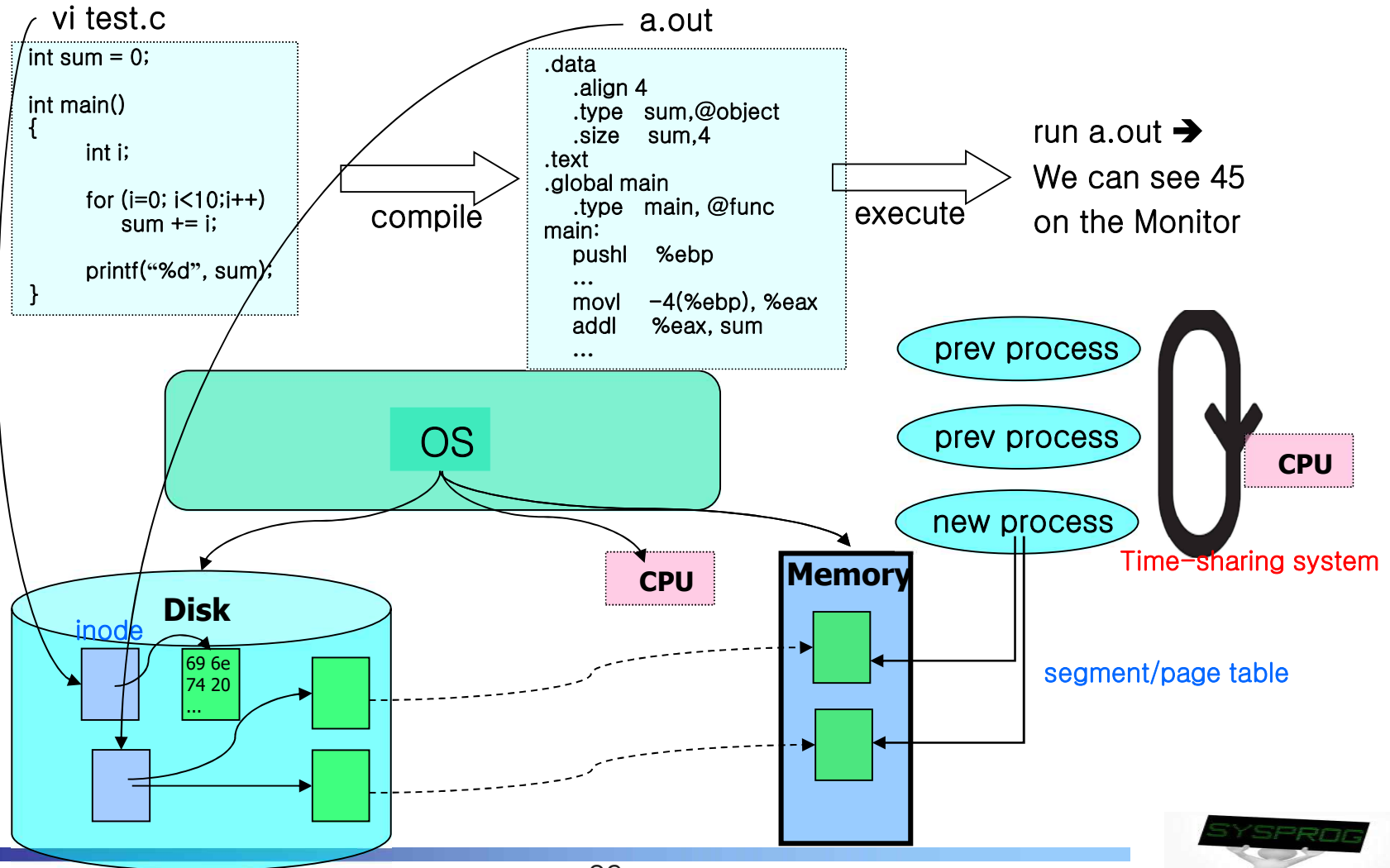
Operating System (13/13)

- Operating system: summary
  - ✓ Process manager (Task manager): CPU
    - process manipulation, schedule, IPC, signal, context switch
    - fork, exec, wait, getpid, (pthread_create) , …
  - ✓ Virtual Memory: Main memory
    - page, segment, address translation, buddy, LRU
    - brk, (malloc, free), …
  - ✓ File system: Storage
    - file, directory, disk scheduling, FAT
    - open, read, write, mknod, pipe, (fopen, fwrite, printf), …
  - ✓ Device driver: Device
    - IO port management, interrupt, DMA
    - open, read, write, ioctl, module, …
  - ✓ Network protocol: Network
    - connection, routing, fragmentation
    - socket, bind, listen, send, receive, …

Figure 1.11
Abstractions provided by
an operating system.

	Processes	
	Virtual memory	
		Files
Processor	Main memory	I/O devices
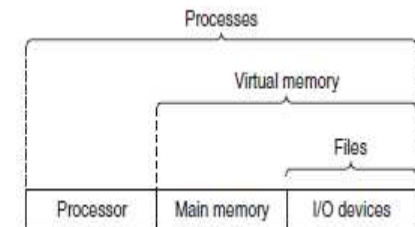
Quiz for 2nd-Week 1st-Lesson

- **Quiz**
  - ✓ 1. Describe the names of Linux command for editor, compiler, assembler, linker and loader (5 names).
  - ✓ 2. Discuss the role of 1) inode, 2) page table and 3) scheduler.
  - ✓ Due: until 6 PM Friday of this week (10th, September)
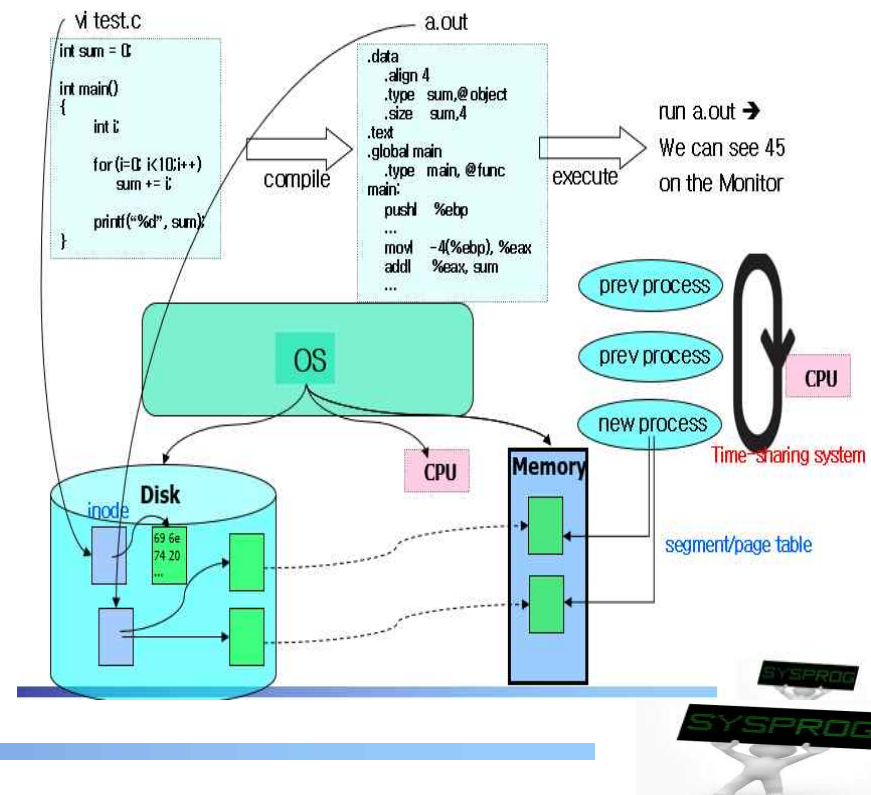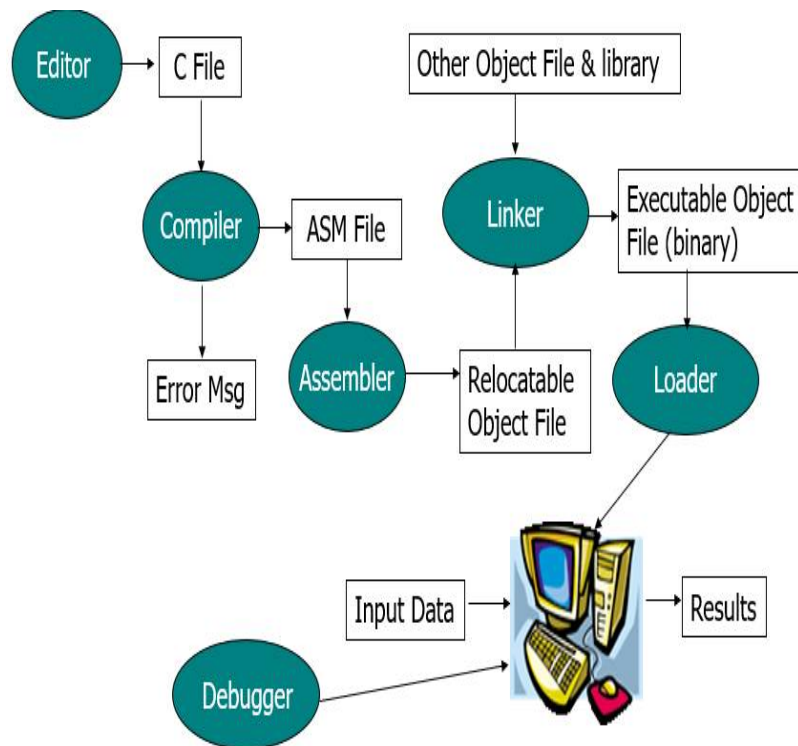
- # Command
  - ✓ file related: ls, cat, more, cp, mkdir, cd, …
  - ✓ task related: ps, kill, jobs, …
  - ✓ utility: vi, gcc, as, make, tar, patch, debugger, ..
  - ✓ management: adduser, passwd, ifconfig, mount, fsck, shutdown, ..
  - ✓ others: man, file, readelf, grep, wc, …

- # shell
  - ✓ command interpreter
  - ✓ pipe, redirection, background processing, ...
  - ✓ shell script programming

command

user ⟹ **shell**

command processing

- ## library
  - ✓ A collection of functions, invoked frequently by a lot of users
    - ▪ Relocatable objects
    - ▪ Most languages have standard libraries (also programmers can make their own custom libraries using ar, ranlib and libtool.)
  - ✓ Type
    - ▪ Static: 1).a, 2) statically linked (compile time), 3) simple
   - ▪ Shared: 1) .so, 2) dynamically linked (runtime), 3) memory efficient

```
┌──────────────────────────────────────────────────────┐
│                    ┌─────────────────────┐            │
│                    │    User program     │            │
│                    └─────────────────────┘            │
│  ┌───────────────────────┐                            │
│  │ Library functions     │                            │
│  │                       │                            │
│  │ printf()              │                            │
│  └───────────────────────┘                            │
│                    ┌─────────────────────┐            │
│                    │  write() agent      │   user space│
│                    ├─────────────────────┤            │
│                    │  write() system call│ kernel space│
│                    └─────────────────────┘            │
└──────────────────────────────────────────────────────┘
```

User program

Library functions

printf()

write() agent

write() system call

user space

kernel space
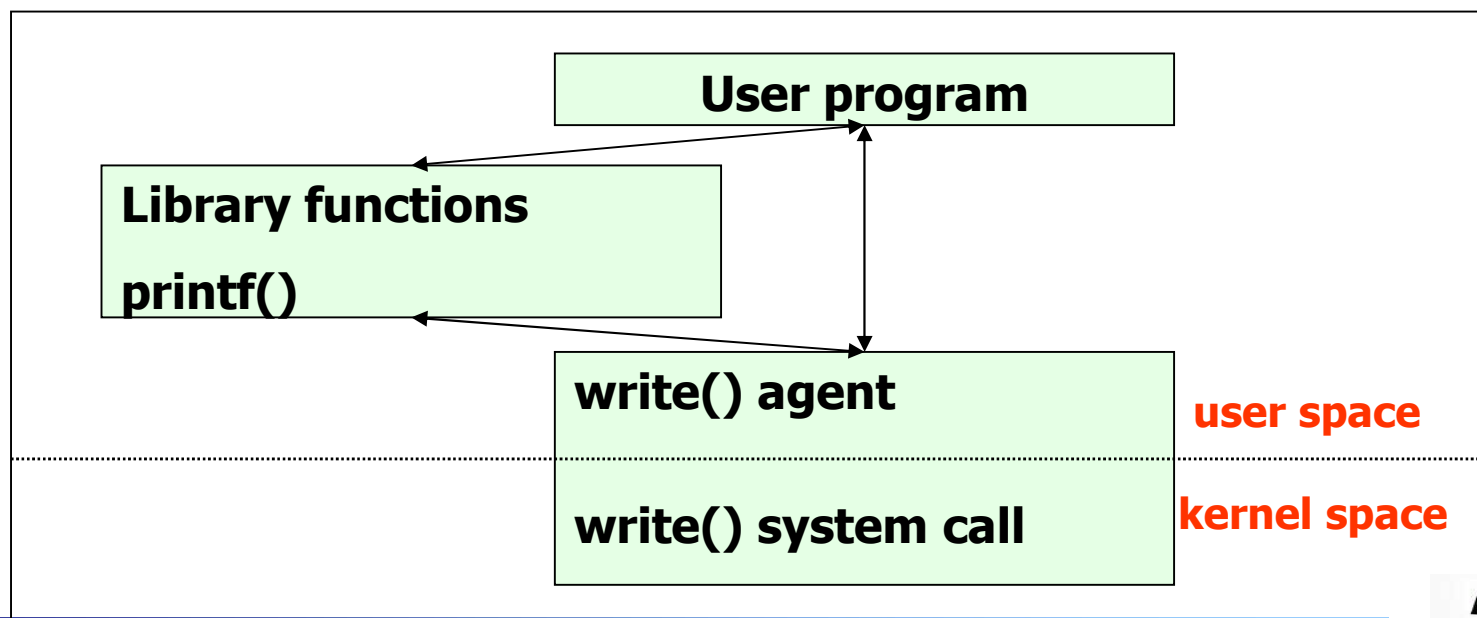
- **Framework (also called as Platform)**
  - ✓ A set of functionalities such as windows, database, graphics, multimedia, web, RPC, protocol, ...
  - ✓ Mobile framework (e.g. Android), Machine learning (e.g. Tensorflow) and Bigdata framework (e.g. MapReduce or Hadoop)


(Source: google image)

(Source: https://www.guru99.com/introduction-to-mapreduce.html)
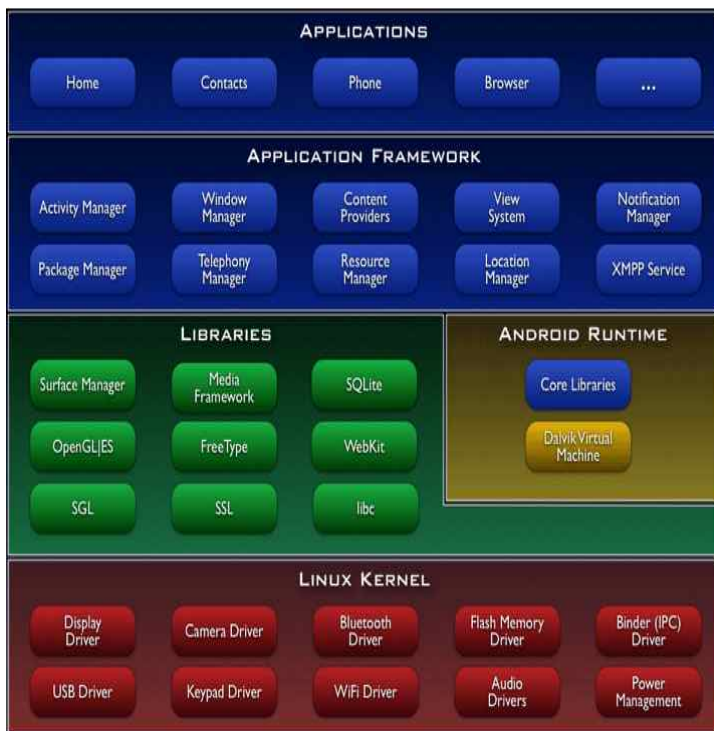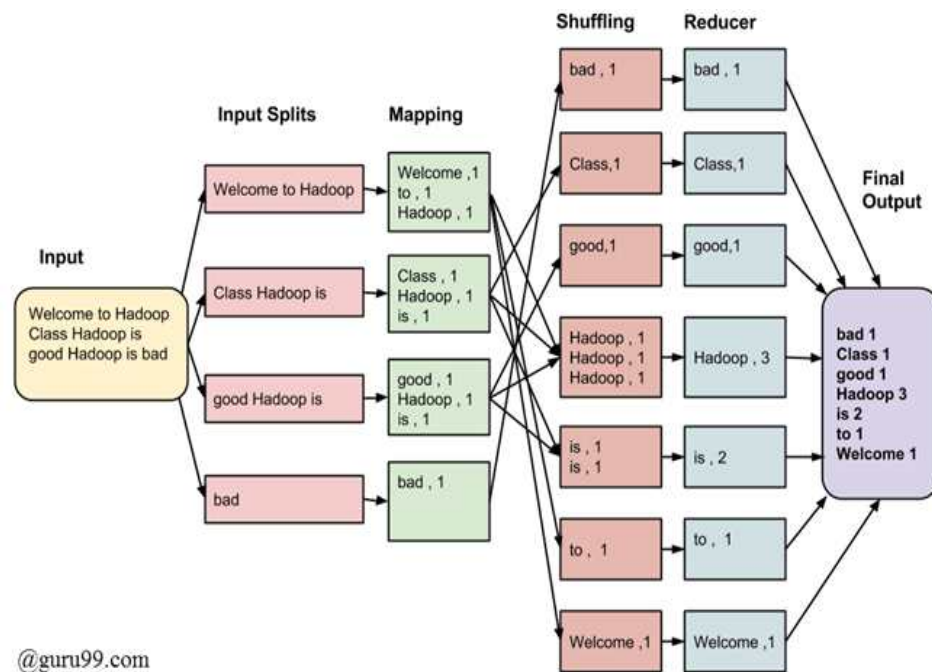
- ## Virtual machine and Docker
  - ✓ Virtual machine: make virtual devices from Hypervisor (or Host OS)
    - ▪ Run GuestOS on the virtual devices
  - ✓ Docker: make a container (an isolated environment) using namespace and cgroup
    - ▪ Docker commands are quite similar to Linux (UNIX) command

Machine Virtualization

Containers
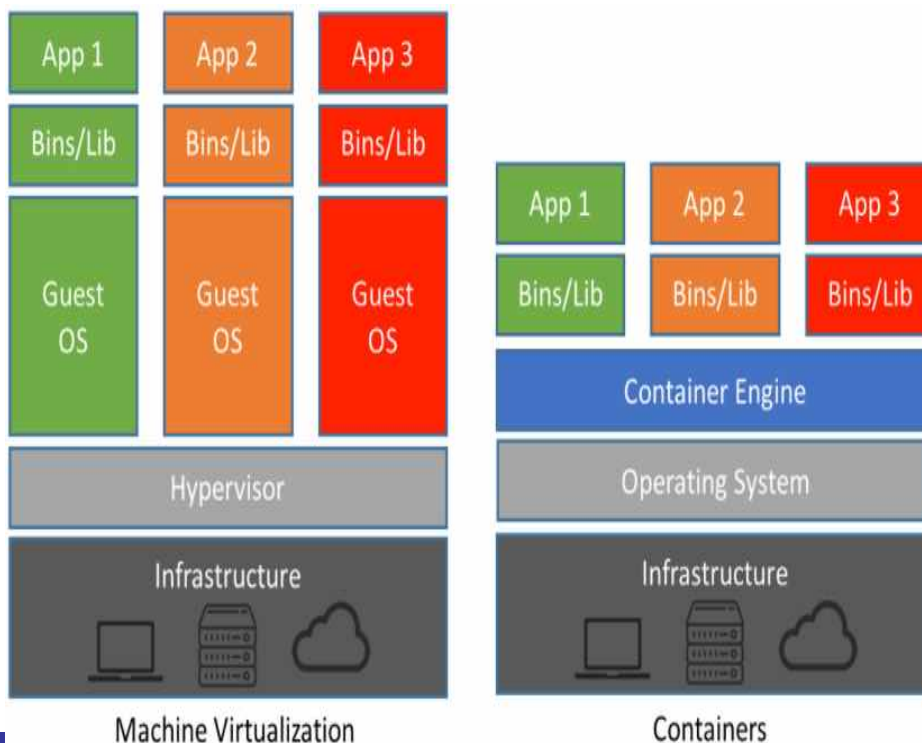
```
[root@docker ~]# docker images
REPOSITORY              TAG             IMAGE ID        CREATED         SIZE
wordpress               latest          ca96afcfa242    2 weeks ago     406 MB
xibosignage/xibo-xmr    release_1.8.1   223afb5ecffe    2 weeks ago     269 MB
ubuntu                  16.04           ebcd9d4fca80    2 weeks ago     118 MB
ubuntu                  14.04           2ff3b426bbaa    2 weeks ago     188 MB
centos                  7               8140d0c64310    2 weeks ago     193 MB
mysql                   5.6             ed7b6c642b9d    3 weeks ago     299 MB
mysql                   5.7             e799c7f9ae9c    3 weeks ago     407 MB
debian                  latest          3e83c23dba6a    3 weeks ago     124 MB
xibosignage/xibo-cms    latest          9678c5299918    5 weeks ago     511 MB
xibosignage/xibo-cms    release_1.8.1   c2767fdc7262    5 weeks ago     511 MB
[root@docker ~]#
```

```
[root@docker ~]# docker run -it -p 9000:80 --name=debian_container1 debian
root@9254e01fadad:/#
```
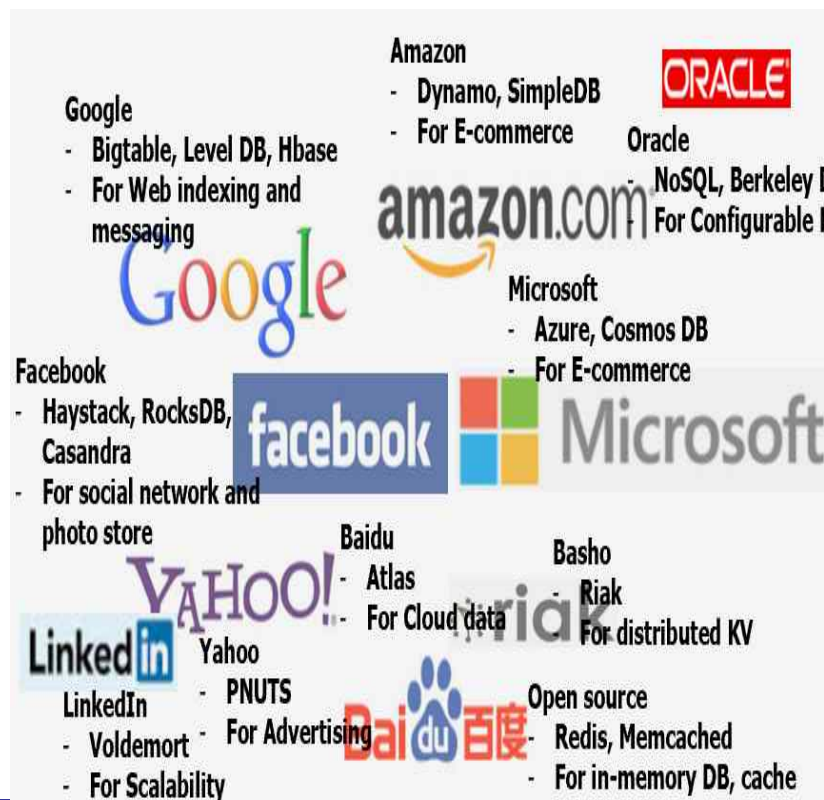
```
[root@docker ~]# docker ps
```

35

- **Key-Value Store**
  - ✓ Bigdata ➔ un-structured ➔ need new database ➔ Key-value store (or Document store or Graph store)
    - ▪ E.g. Google's LevelDB, Facebook's RocksDB, Amazon's Dynamo, …
  - ✓ Key data structure: LSM-tree, Skipped-list, Bloom filter, …
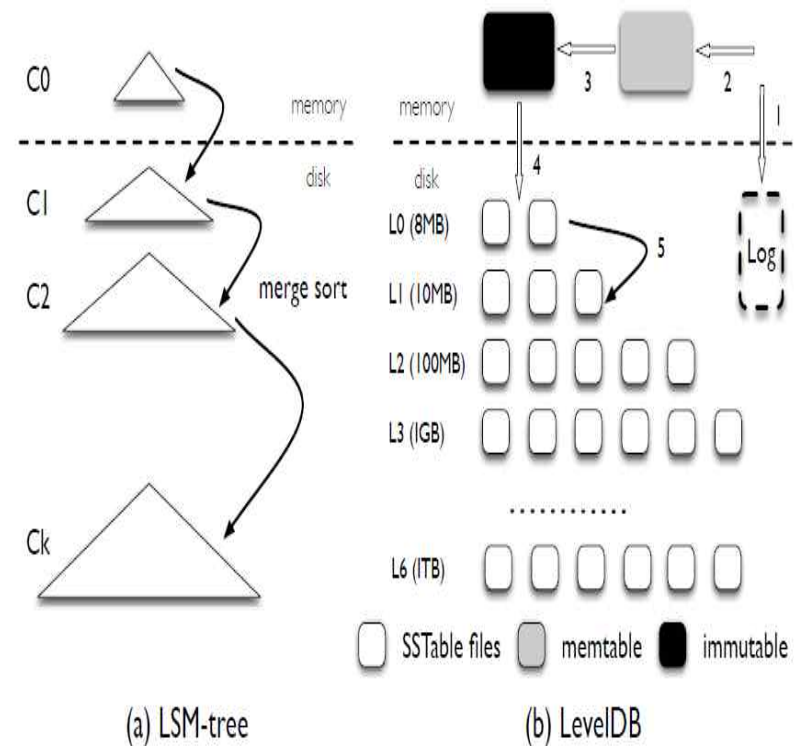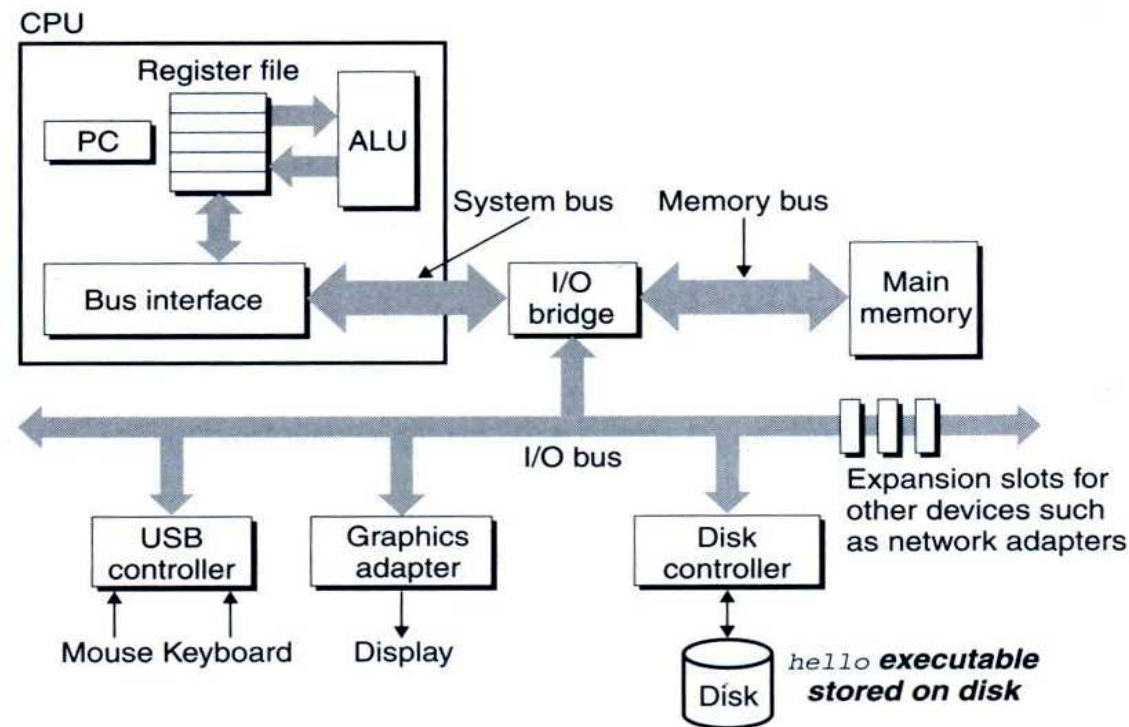
Hardware consideration (1/6)

- **Computer organization**
  - ✓ CPU: registers (include PC), ALU, cache, …
  - ✓ Memory: "address, content" pair
  - ✓ Device: controller + device itself
  - ✓ Bus: hierarchical

Figure 1.4
Hardware organization of a typical system. CPU: Central Processing Unit, ALU: Arithmetic/Logic Unit, PC: Program Counter, USB: Universal Serial Bus.
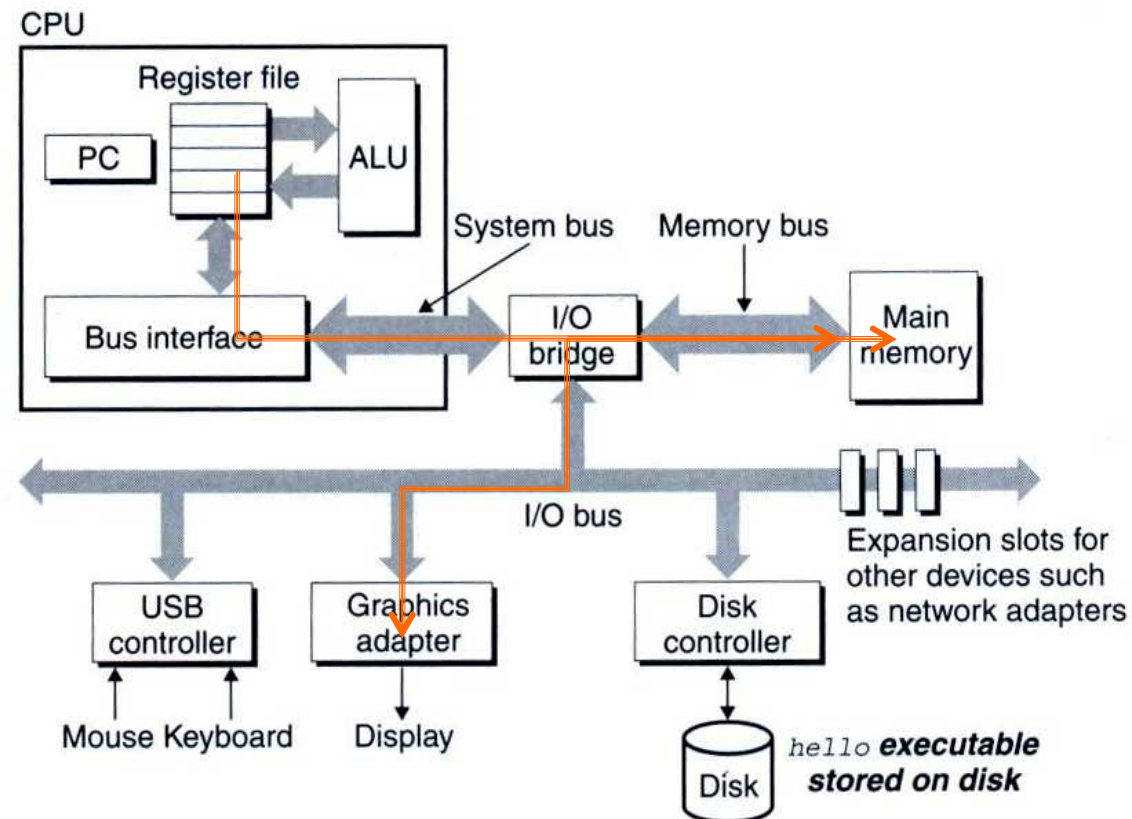

(Source: CSAPP)

Computer organization

- ✓ When a program load

Figure 1.4
Hardware organization of a typical system.
CPU: Central Processing Unit, ALU: Arithmetic/Logic Unit, PC: Program Counter, USB: Universal Serial Bus.

CPU

Register file

PC

ALU

☞ **process (task)**

System bus   Memory bus

Bus interface

I/O bridge

Main memory

I/O bus

USB controller

Graphics adapter

Disk controller

Expansion slots for other devices such as network adapters

Mouse Keyboard

Display

Disk

hello **executable stored on disk**

☞ **program (binary)**
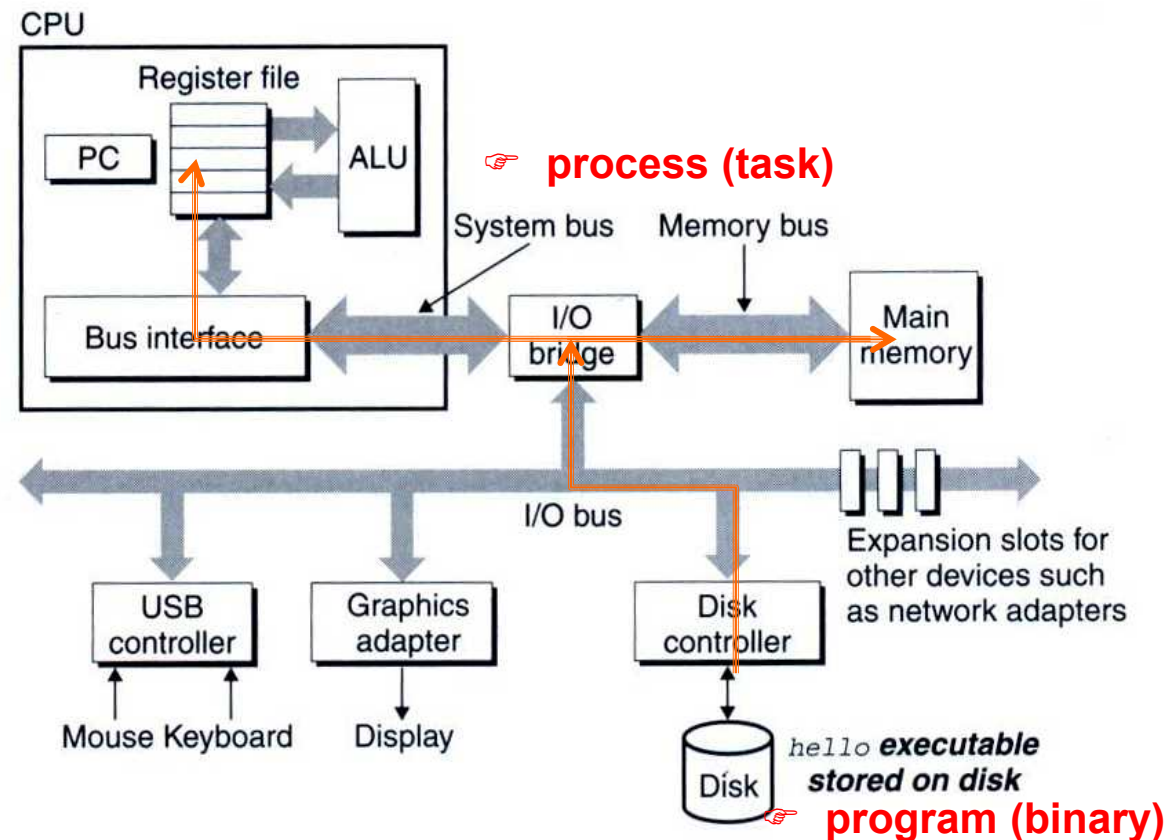
■ Computer organization

- ✓ When printf("Hello World\n") is invoked

Figure 1.4
Hardware organization
of a typical system.
CPU: Central
Processing Unit, ALU:
Arithmetic/Logic Unit, PC:
Program Counter, USB:
Universal Serial Bus.

CPU

Register file

PC

ALU

System bus   Memory bus

Bus interface

I/O bridge

Main memory

I/O bus

USB controller

Graphics adapter

Disk controller

Expansion slots for other devices such as network adapters

Mouse Keyboard

Display

Disk

hello executable stored on disk

Hardware consideration (4/6)

- **Memory matters**
  - ✓ array programming example

```
/* program A */
int a[1000][1000];
int i, j;
....

for (i=0; i<1000; i++)
   for (j=0; j<1000; j++)
     a[i][j] ++;
```

VS

```
/* program B */
int a[1000][1000];
int i, j;
....

for (i=0; i<1000; i++)
   for (j=0; j<1000; j++)
     a[j][i] ++;
```

■ **Memory matters**

  ✓ Memory layout of the array programming example
  ✓ Note that, in limited memory, some data are swapped out and in

A[0][0]
A[0][1]
A[0][2]
A[0][3]
A[0][4]

A[0][999]
A[1][0]
A[1][1]
A[1][2]

A[1][999]
A[2][0]
A[2][1]
A[2][2]

A[2][999]
A[3][0]
A[3][1]
A[3][2]

A[999][996]
A[999][997]
A[999][998]
A[999][999]

em bus   Memory bus

I/O
bridge   Main
                          memory

I/O bus   Expansion sl
                          other devices
                          as network a

Disk
controller

hello **executabl**
Disk   **stored on disk**
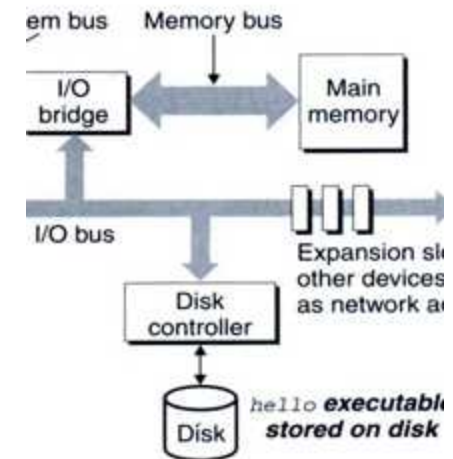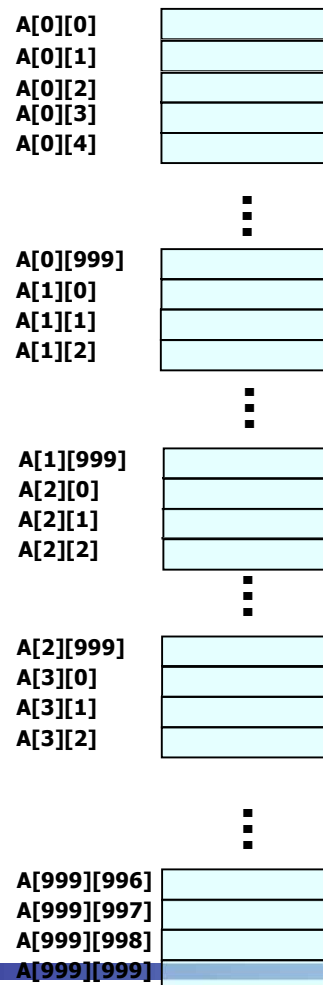
Hardware consideration (6/6)

- **CPU also matters**
  - ✓ Loop unrolling example
    - ▪ Two programs show different resource utilization in CPU (➔ See Chapter 5 in CSAPP)

```
void combine4(vec_ptr v, data_t *dest)
{
    int i;
    int length = vec_length(v);
    data_t *data = get_vec_start(v);
    data_t x = 0;

    for (i = 0; i < length; i++) {
        x = x + data[i];
    }
    *dest = x;
}
```

VS

```
void combine5(vec_ptr v, data_t *dest)
{
    int i;
    int length = vec_length(v);
    data_t *data = get_vec_start(v);
    data_t x = 0;
    int limit = length − 2;

    for (i = 0; i < limit; i += 3) {
        x = x + data[i] + data[i+1] + data[i+2];
    }

    for ( ; i < length; i++) {
        x = x + data[i];
    }
    *dest = x;
}
```

(Source: Chapter 5 in CSAPP)

- **Key of System Program: Abstraction**
  - ✓ **Abstraction** is the process of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose.
  - ✓ In computer science, abstraction tries to reduce and factor out details so that the programmer can focus on a few concepts at a time. A system can have several abstraction layers whereby different meanings and amounts of detail are exposed to the programmer.

- **CPU**

Human-Friendly High Level Language
(ISA: Instruction Set Architecture)

Compilation system
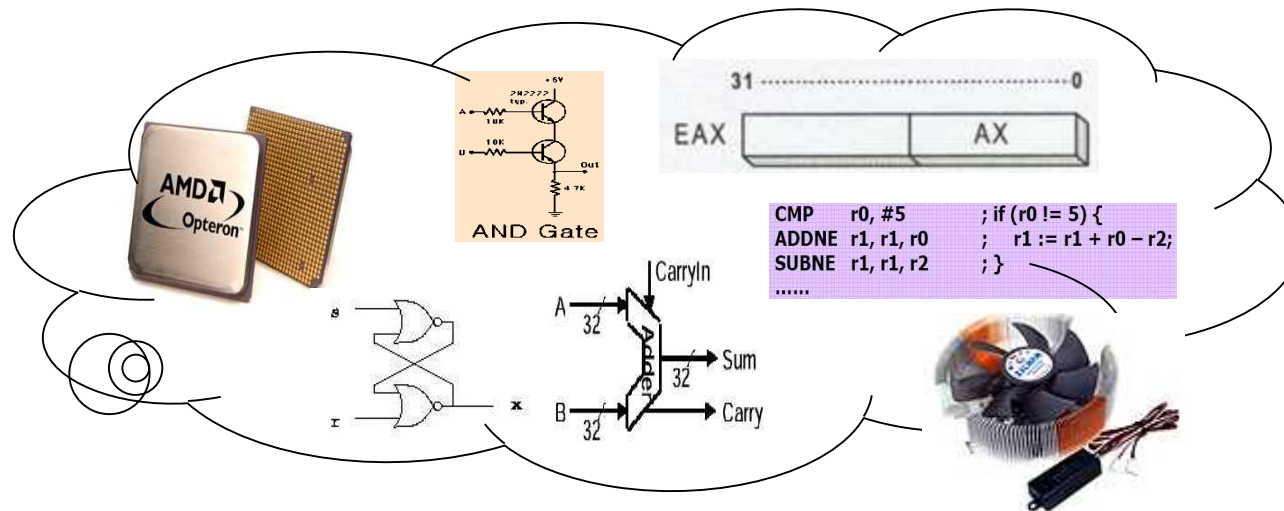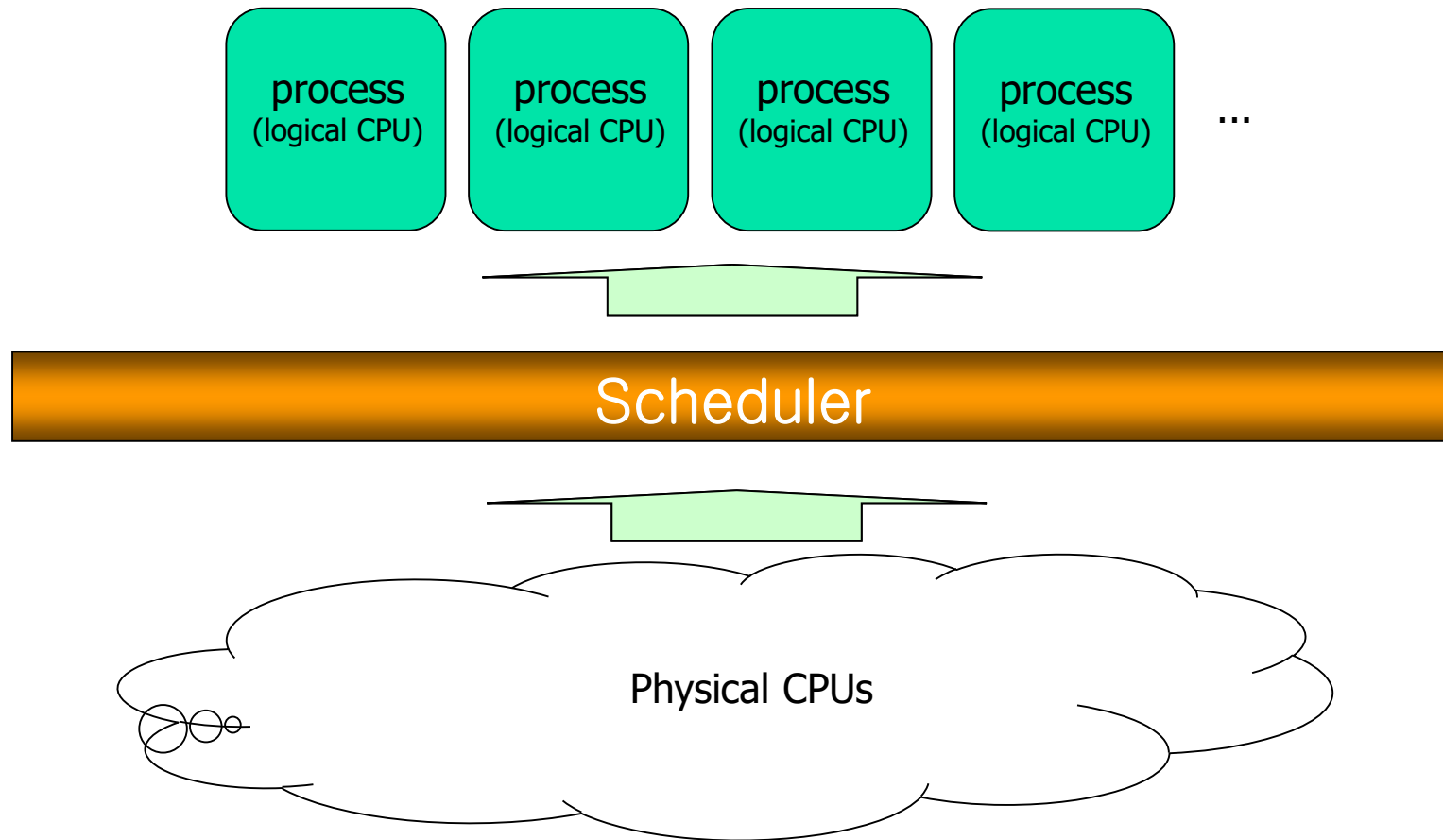
```
CMP    r0, #5        ; if (r0 != 5) {
ADDNE  r1, r1, r0     ;    r1 := r1 + r0 − r2;
SUBNE  r1, r1, r2     ; }
......
```

■ **Multitasking**

- **Memory management**

Relocatable Unlimited Memory Space

virtual memory

Fixed Limited Memory Space

- **File system**

file system

- **Device driver**

Handle and I/O STREAM
(open, read, write, close)
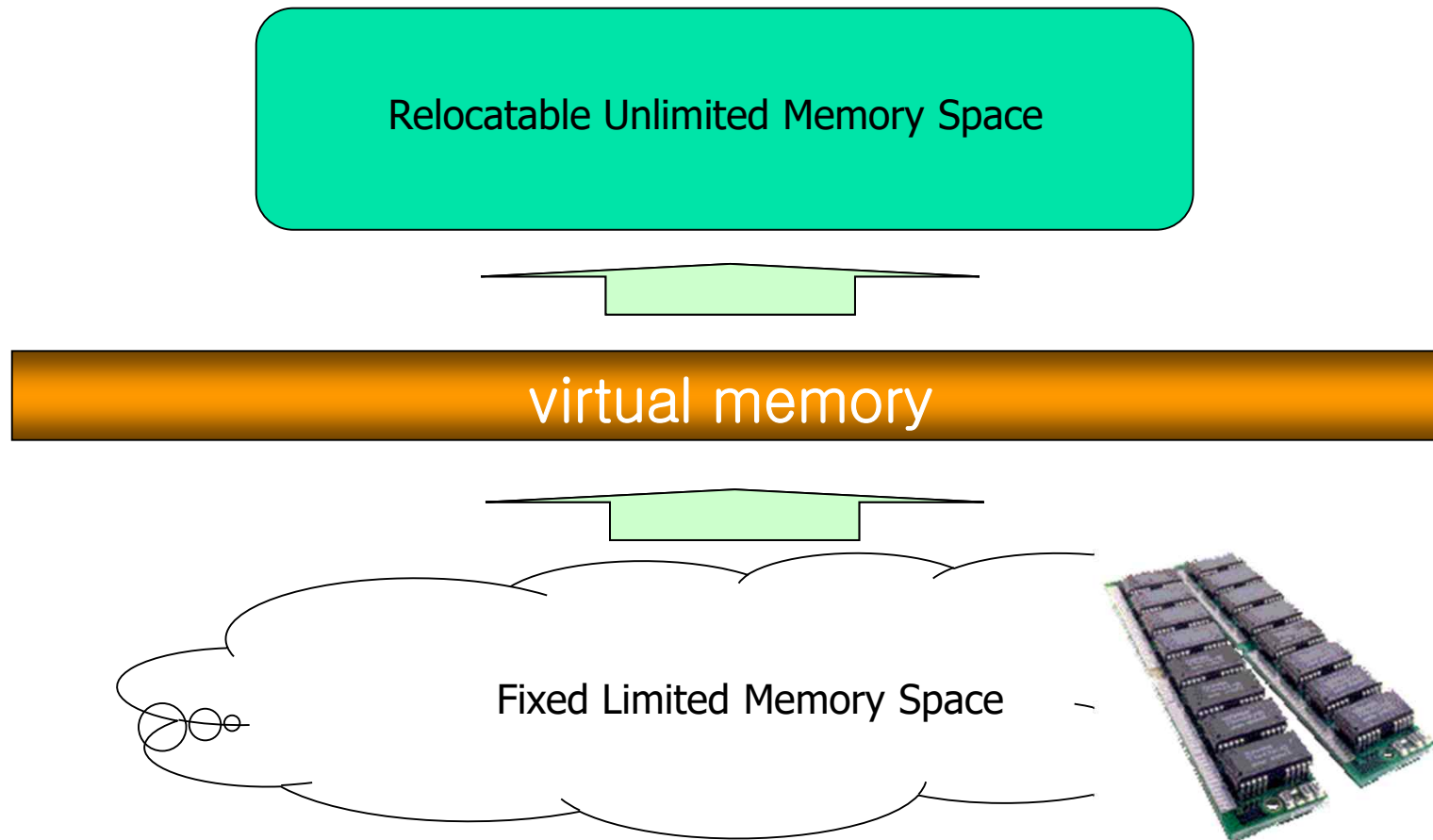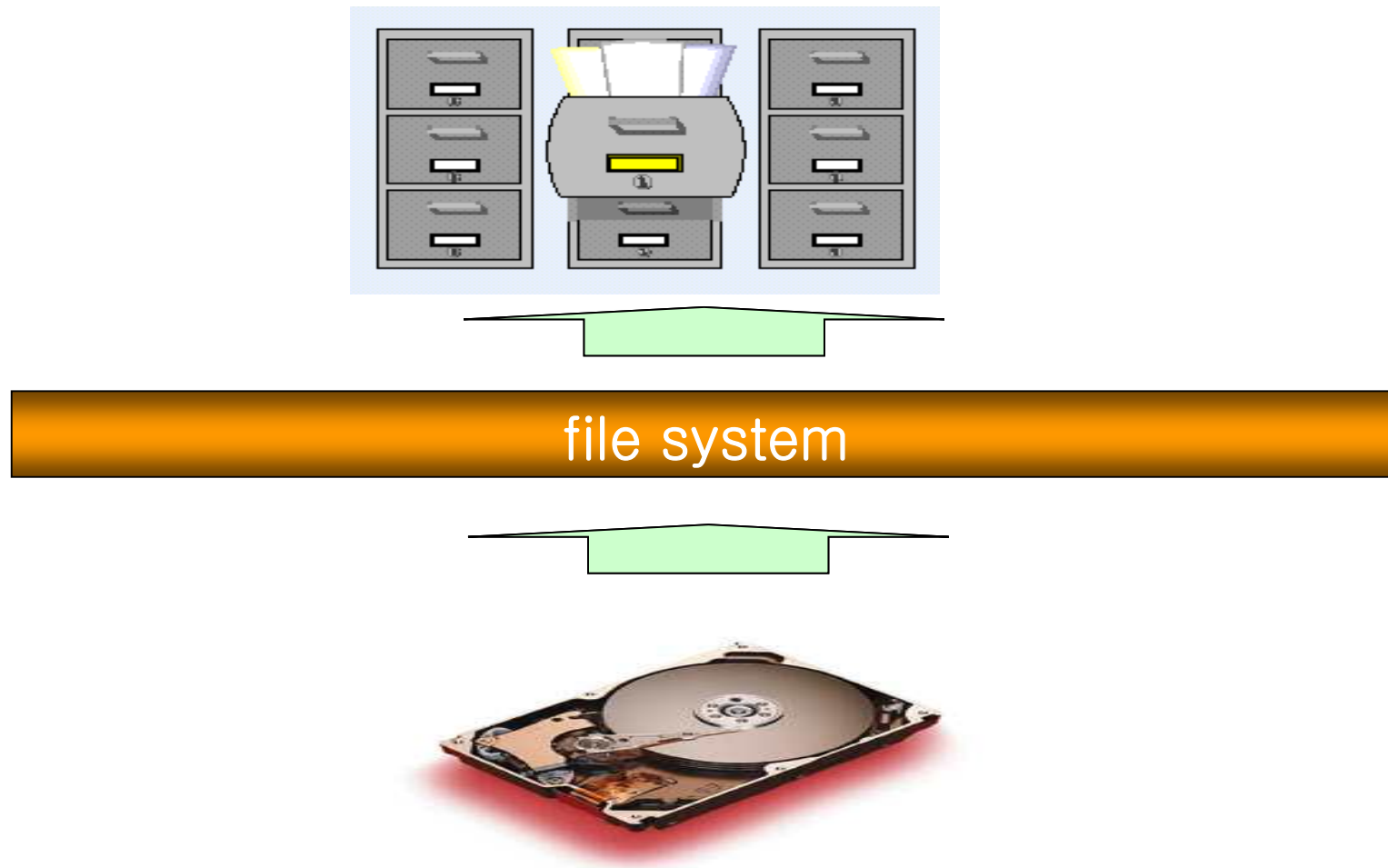
device driver
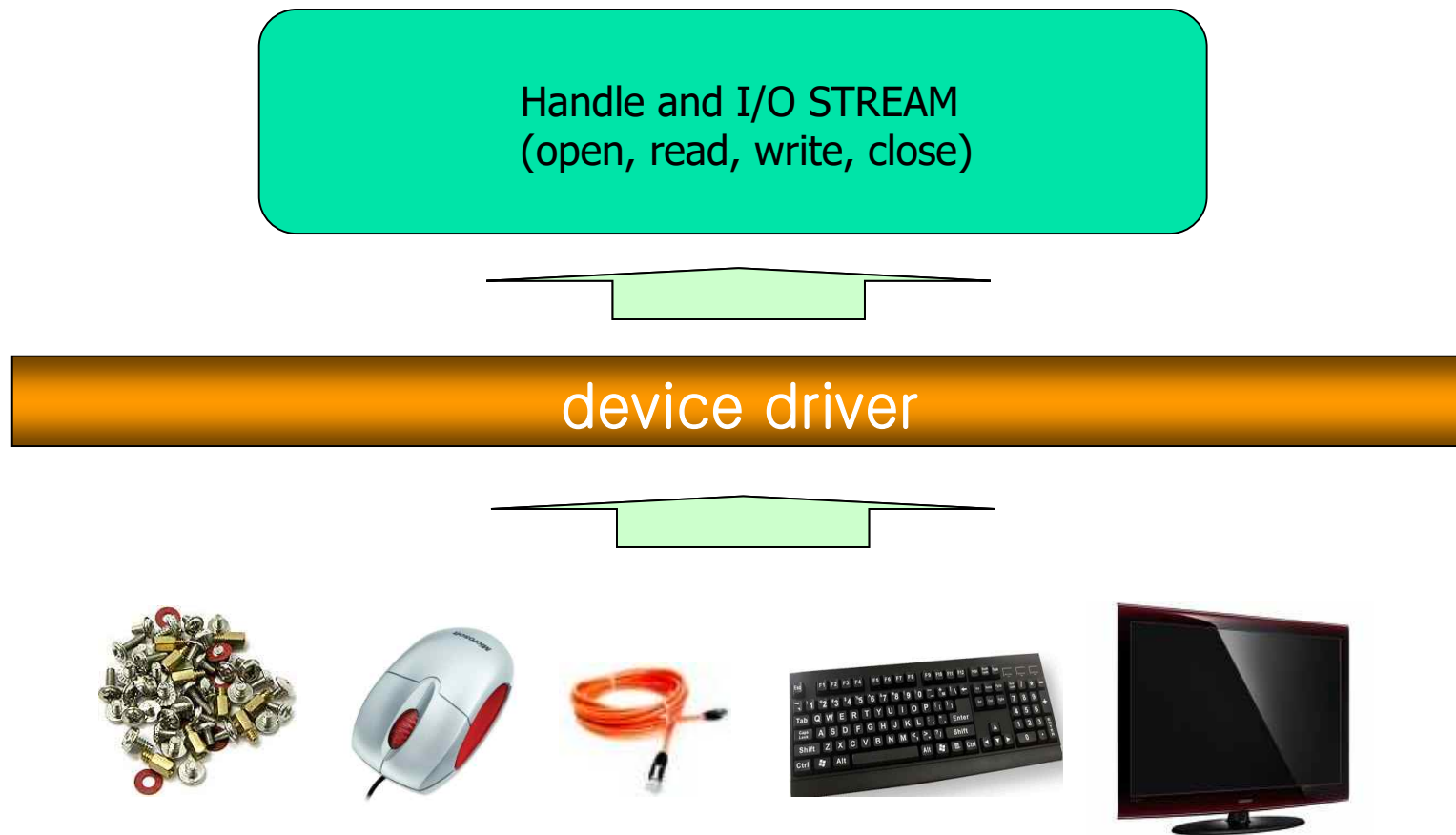
Abstraction (7/9)

- Data representation

| char | integer | float | pointer | ... |

data manipulation

bits
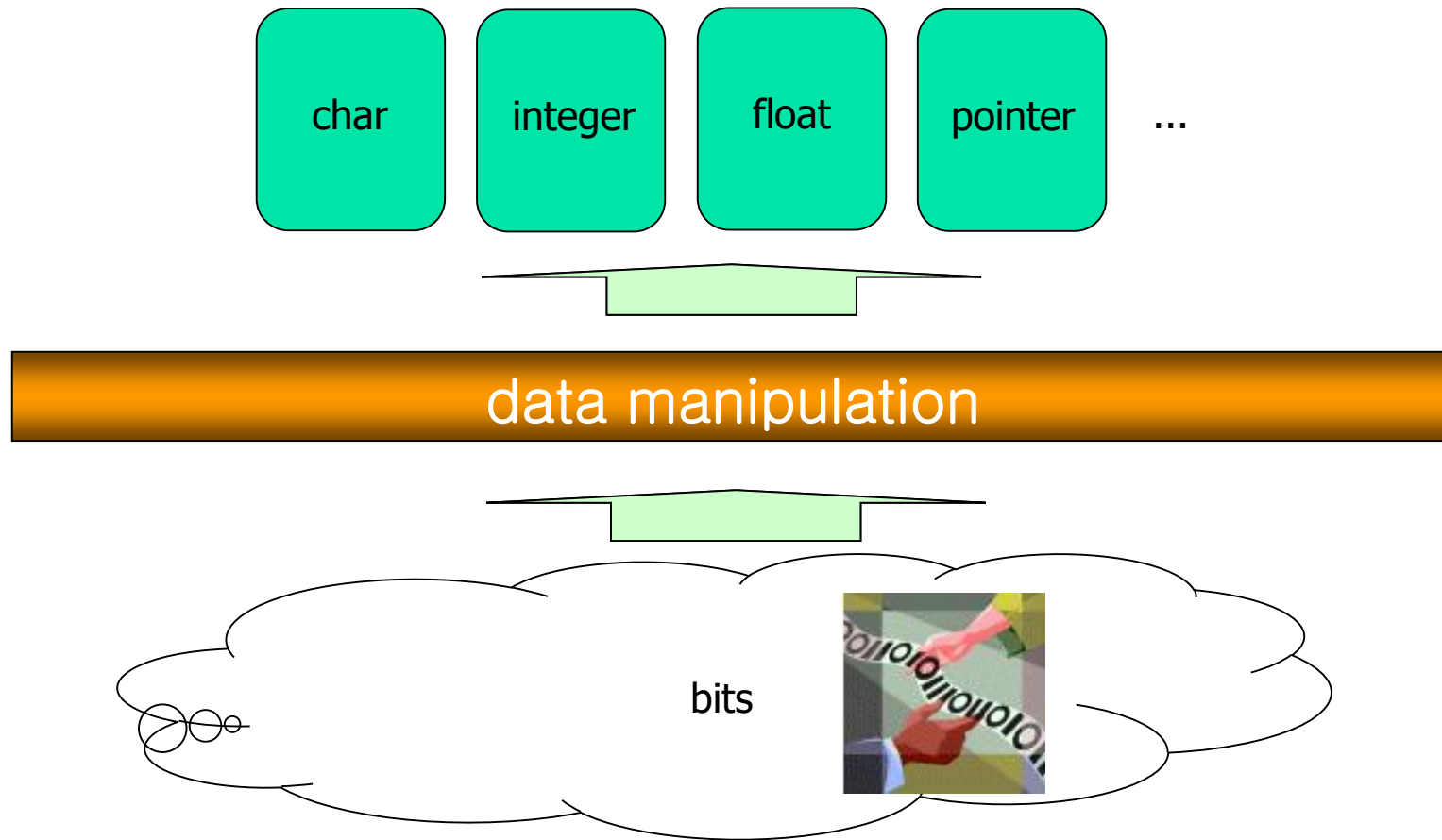
- **Security and reliability**

Secure and Trusted World
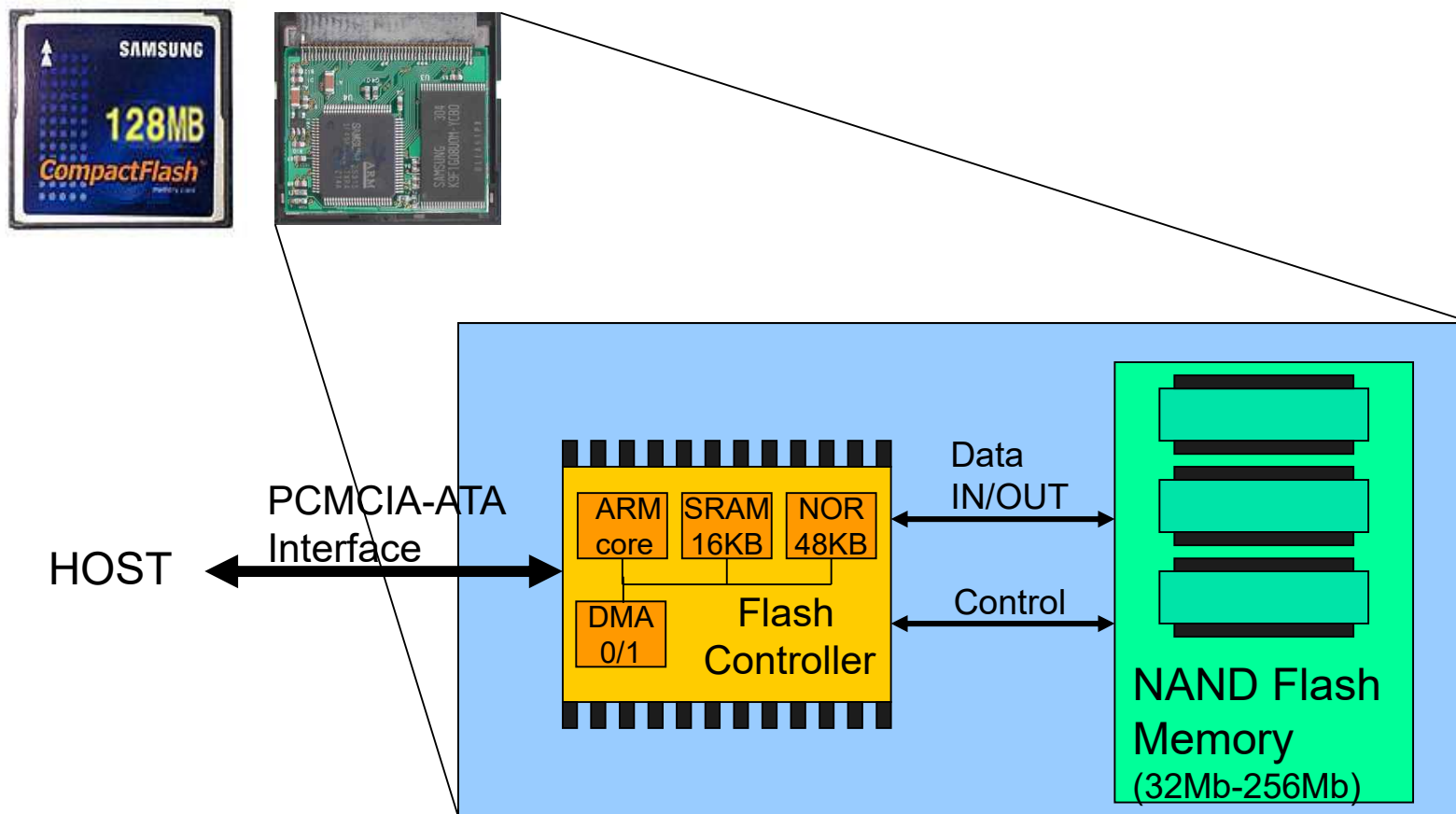
security and fault−tolerant system

Real World
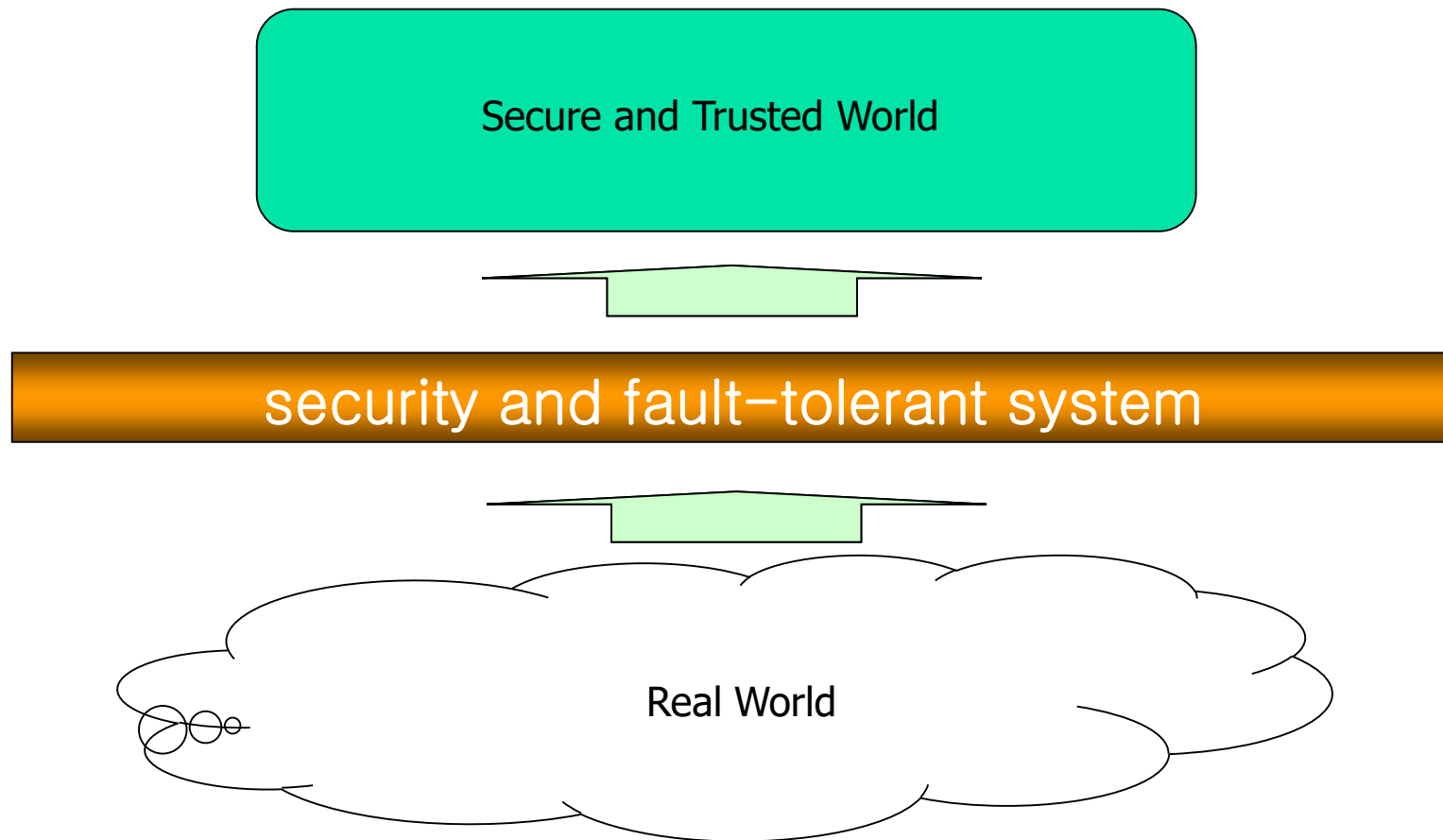
Abstraction (9/9)

- **Software layers (Layered architecture)**
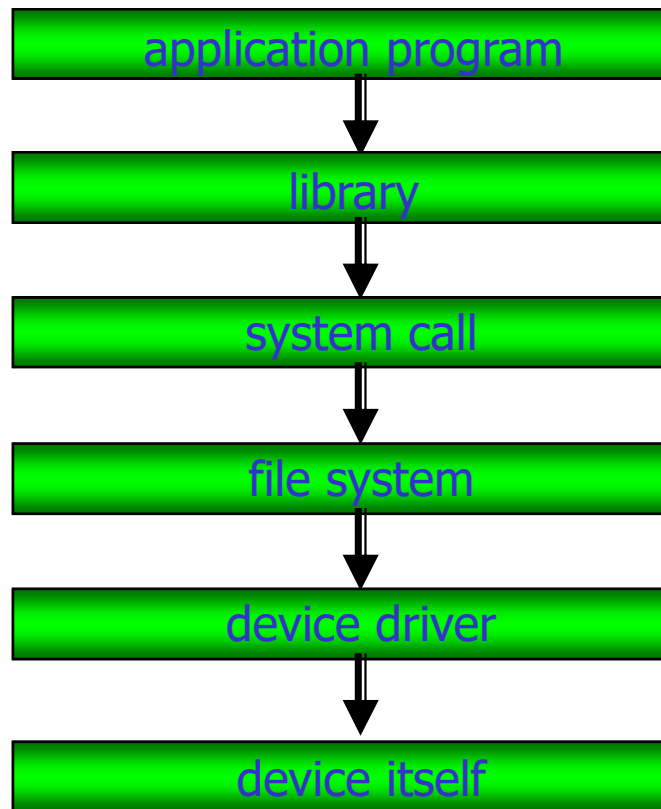
Importance of System Program

■ Compact Flash Storage Card Internals

☞ **Knowledge about how HW and SW are cooperated becomes indispensable in recent computing industry (HW/SW Co-design)**

Summary

- **Definition of System Program**
  - ✓ Supporting computing environments
  - ✓ Managing hardware directly

- **3 Types of System Program**
  - ✓ Compilation system, operating system, runtime system
  - ✓ Hardware consideration

- **Concept of Abstraction**
  - ✓ Information hiding
  - ✓ Layered architecture

☞ **Homework 1: Read the Chapter 1, "A Tour of Computer Systems" and summarize it.**
  - ✓ **Requirement: 1) From the beginning to the Section 1.7 (at most 9 pages, 10 font (can be larger for section or subsection title), discuss about "process, thread, virtual memory and file"), 2) What is the purpose of studying System Programming? (1~2 page)**
  - ✓ **Deadline: 6 PM Friday of the next week (17th, September)**
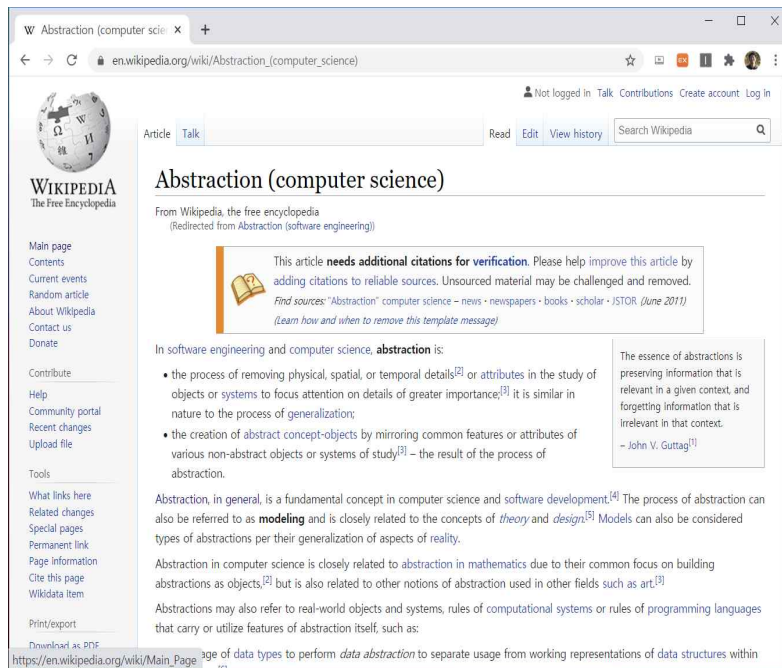 - ✓ **Caution: Do not copy!!**

Quiz for 2nd-Week 2nd-Lesson

- **Quiz**
  - ✓ Describe an example of abstraction in your life and discuss the features of abstraction in your chosen example (e.g. information hiding, focusing on what you are interested in).
    - ▪ If it is funny, better grade :-)
  - ✓ Due: until 6 PM Friday of this week (10th, September)

(Source: https://thevaluable.dev/abstraction-type-software-example/)

Appendix

- **RISC vs. CISC**
  - ✓ assembly language example
    - ▪ a = b + c;

    ```
    load       b, eax
    add        c, eax
    store      eax, a
    ```

    VS

    ```
    add        b, c, a
    ```
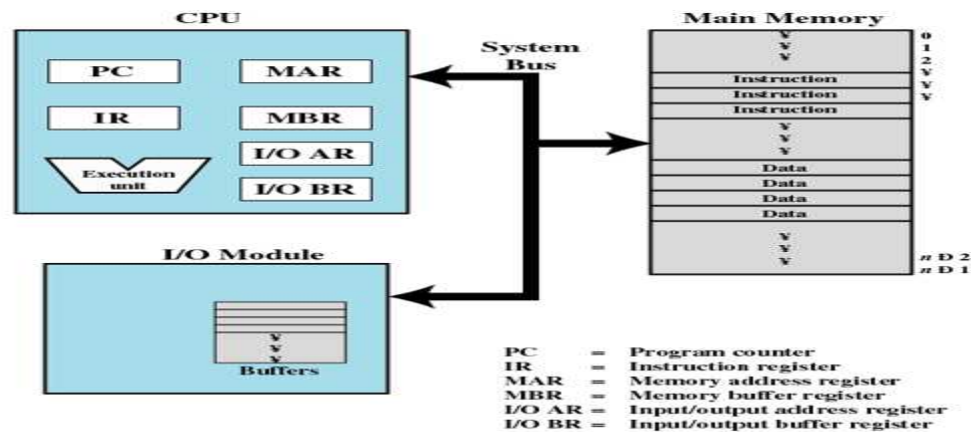
  - ✓ Instruction execution

Figure 1.1 Computer Components: Top-Level View

(Source: W. Stalling, "Operating Systems: Internals and Design Principles")