

Lecture Note 1.

What is System Programming

September 3, 2025

Jongmoo Choi
Dept. of Software
Dankook University

<http://embedded.dankook.ac.kr/~choijm>

Contents

- Understand what is system program
- Identify three types of system program
 - ✓ Compilation system
 - ✓ Operating system
 - ✓ Runtime system
- Discuss Hardware consideration
- Grasp the abstraction concept
- Reference: Chapter 1 in the CSAPP

CHAPTER

1

A Tour of Computer Systems

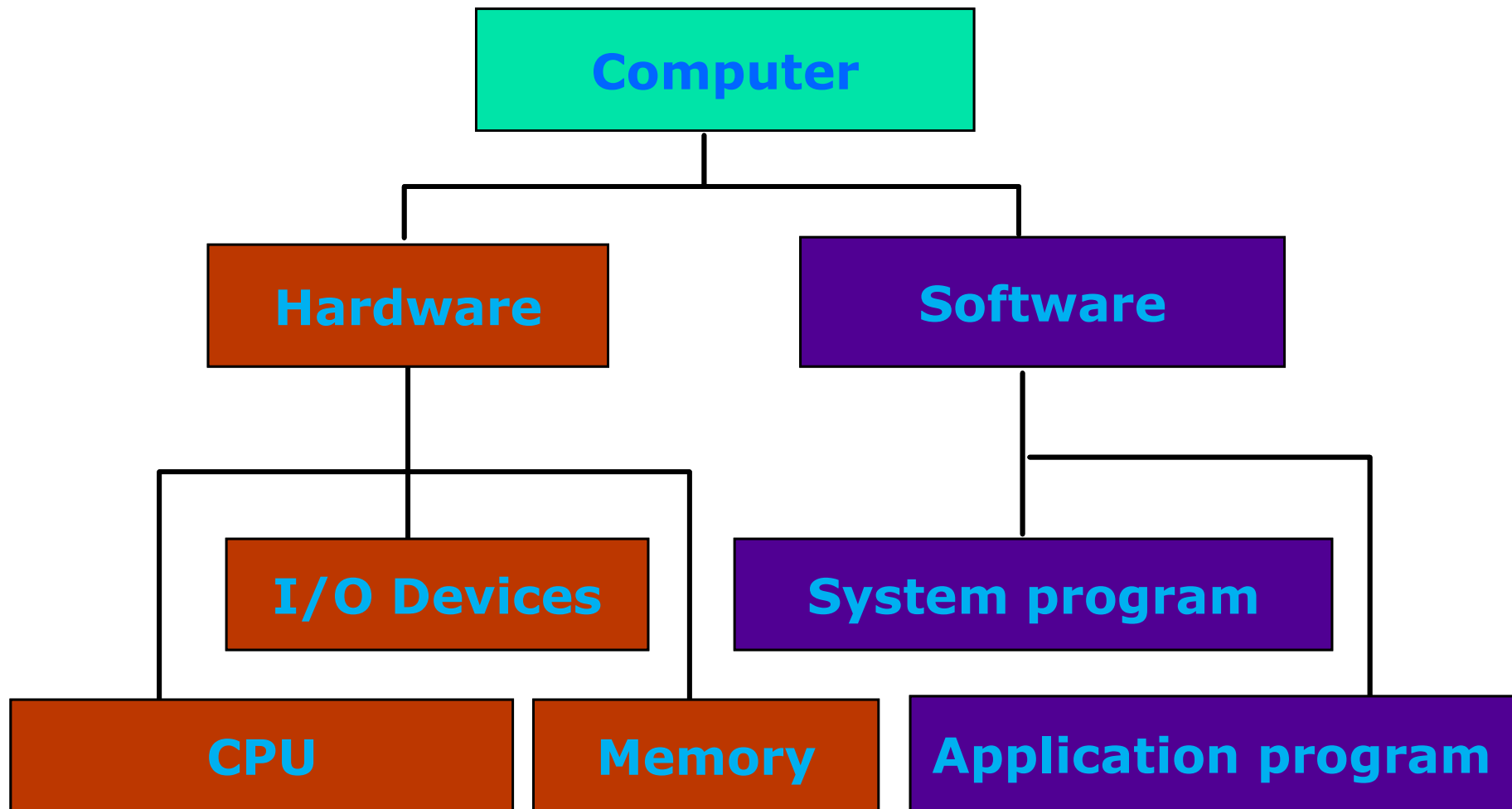
1.1	Information Is Bits + Context	3
1.2	Programs Are Translated by Other Programs into Different Forms	4
1.3	It Pays to Understand How Compilation Systems Work	6
1.4	Processors Read and Interpret Instructions Stored in Memory	7
1.5	Caches Matter	12
1.6	Storage Devices Form a Hierarchy	13
1.7	The Operating System Manages the Hardware	14
1.8	Systems Communicate with Other Systems Using Networks	20
1.9	Important Themes	21
1.10	Summary	25
	Bibliographic Notes	26

(Source: CSAPP)



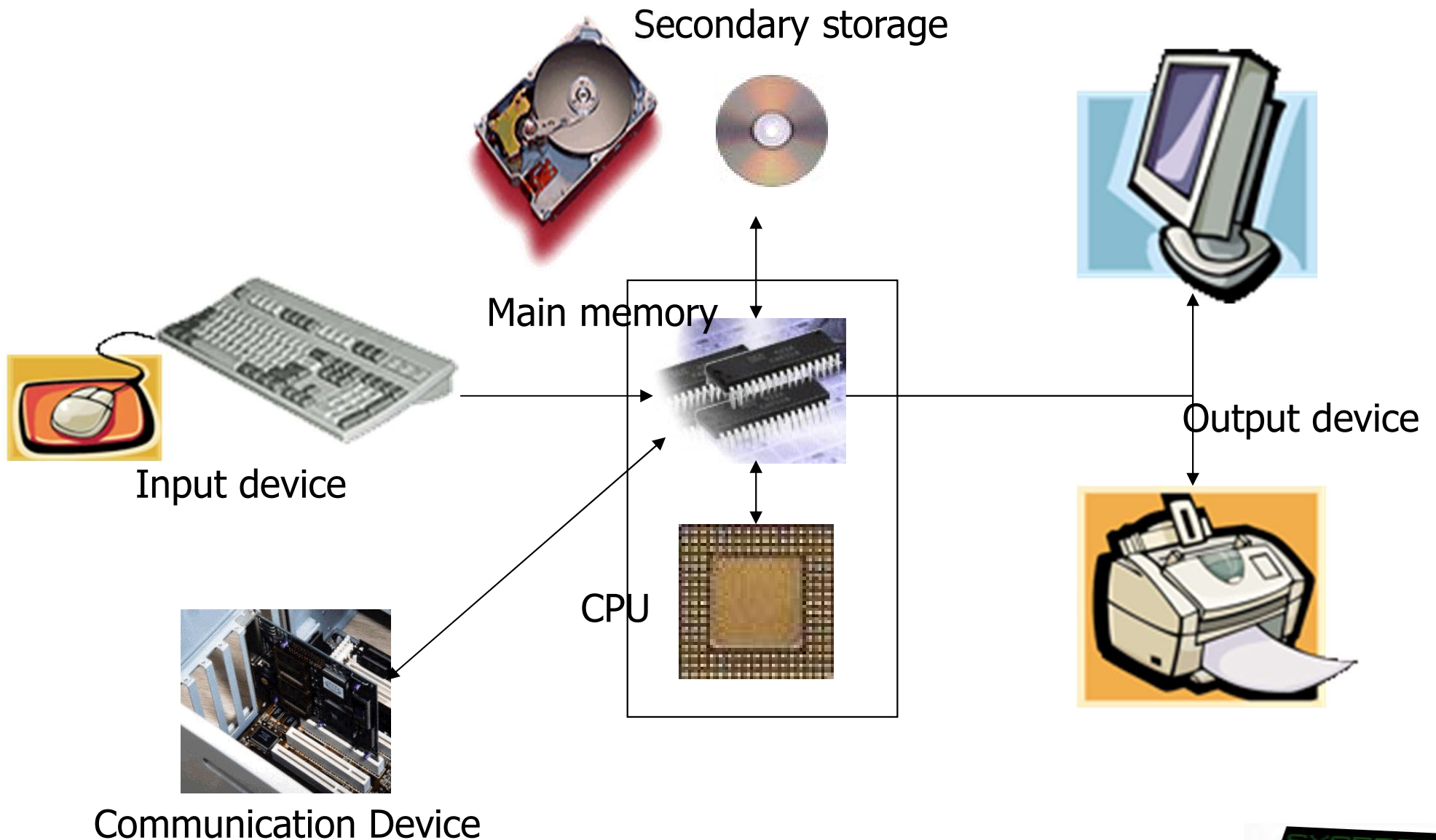
Definition of System Program (1/8)

- Computer organization



Definition of System Program (2/8)

■ Hardware components: PC



Definition of System Program (3/8)

■ Hardware components: DRAM vs. Disk

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

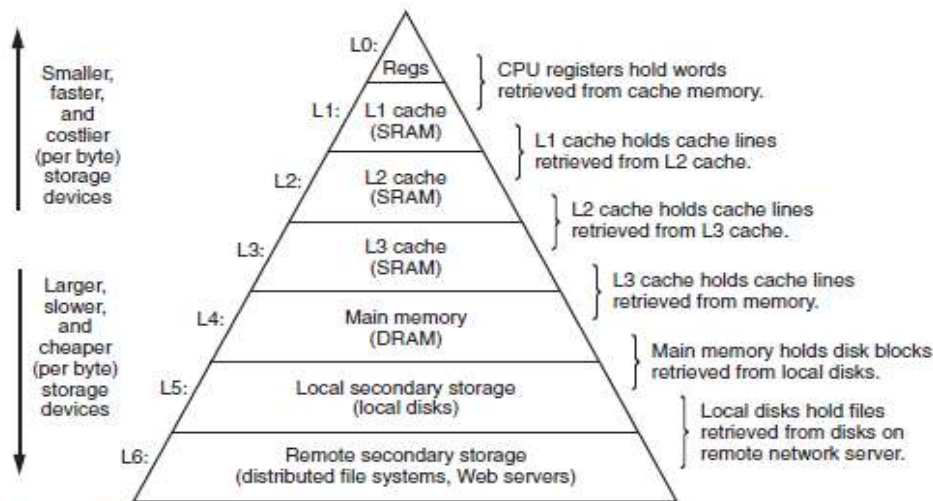
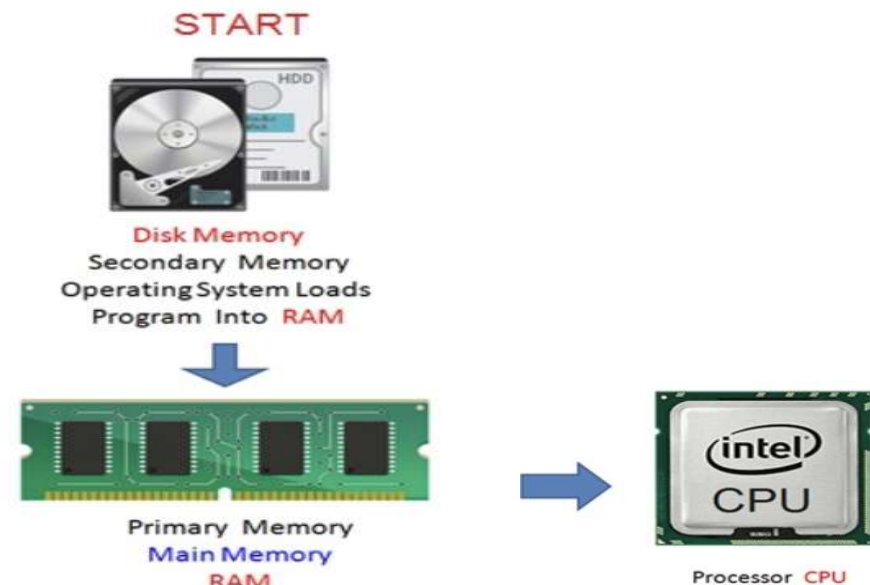


Figure 1.9 An example of a memory hierarchy.

(Source: CSAPP)



(Source: Google Image)



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)

Definition of System Program (5/8)

■ Hardware components: PC vs. Mobile

- ✓ Differ according to the requirements for Mobile devices
- ✓ Power Saving
 - Make use of RISC 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 bank에 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?



Definition of System Program (7/8)

■ 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)

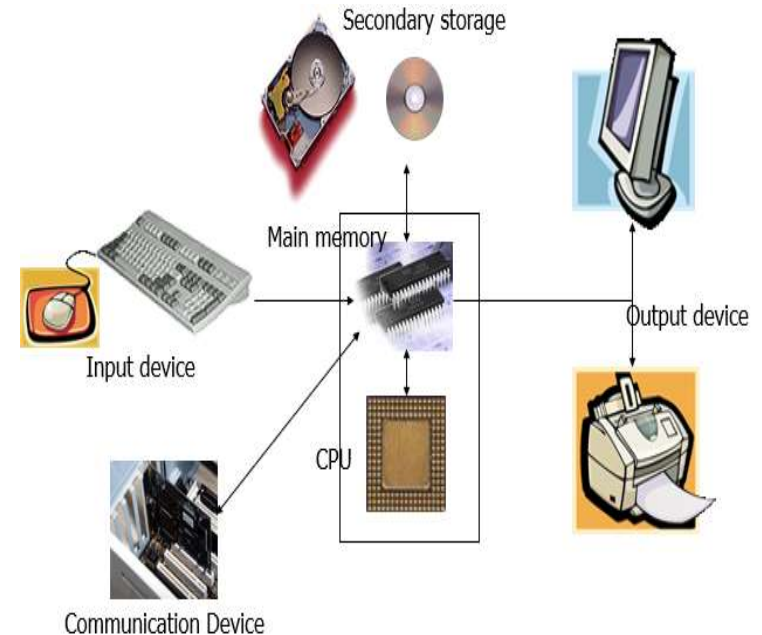
■ Software components: System program

✓ Definition

- Supporting computing environments for application programs (Supporting interfaces such as commands, library functions and system calls)
- Strongly related to hardware (hardware management)

✓ Support **abstraction**

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



Aside (Optional)

■ RISC vs. CISC

✓ assembly language example: look RISC takes longer

■ $a = b + c$;

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

VS

```
add     b, c, a
```

✓ Instruction execution: but, they can be pipelined

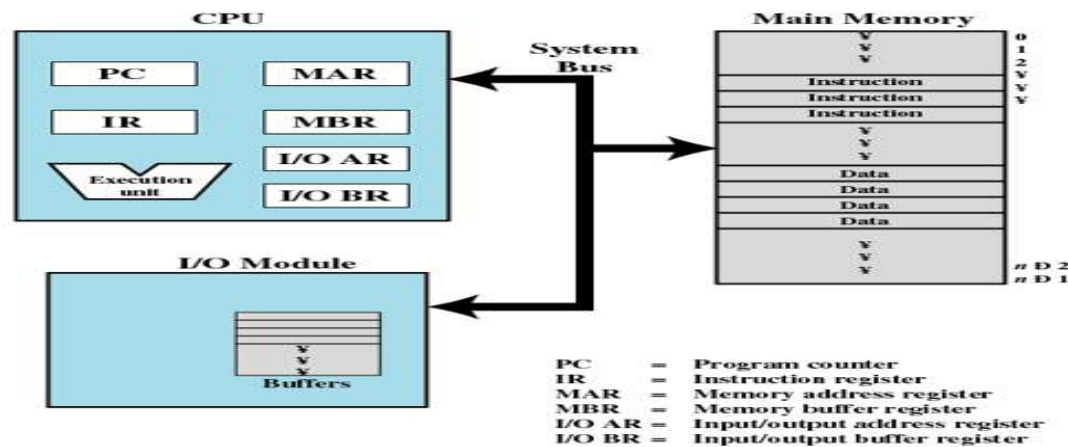


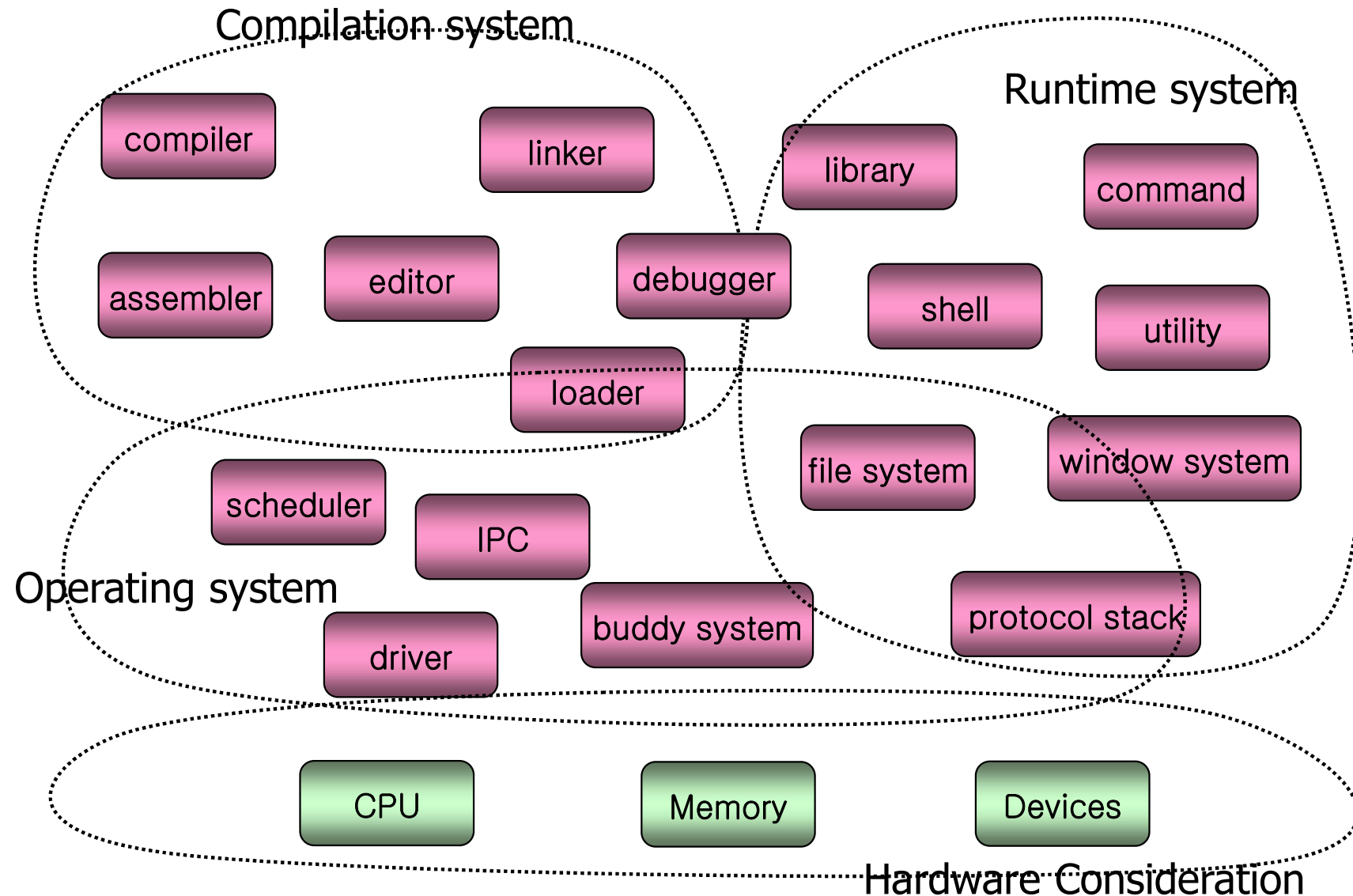
Figure 1.1 Computer Components: Top-Level View

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



Types of System Program

■ Classification



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)**

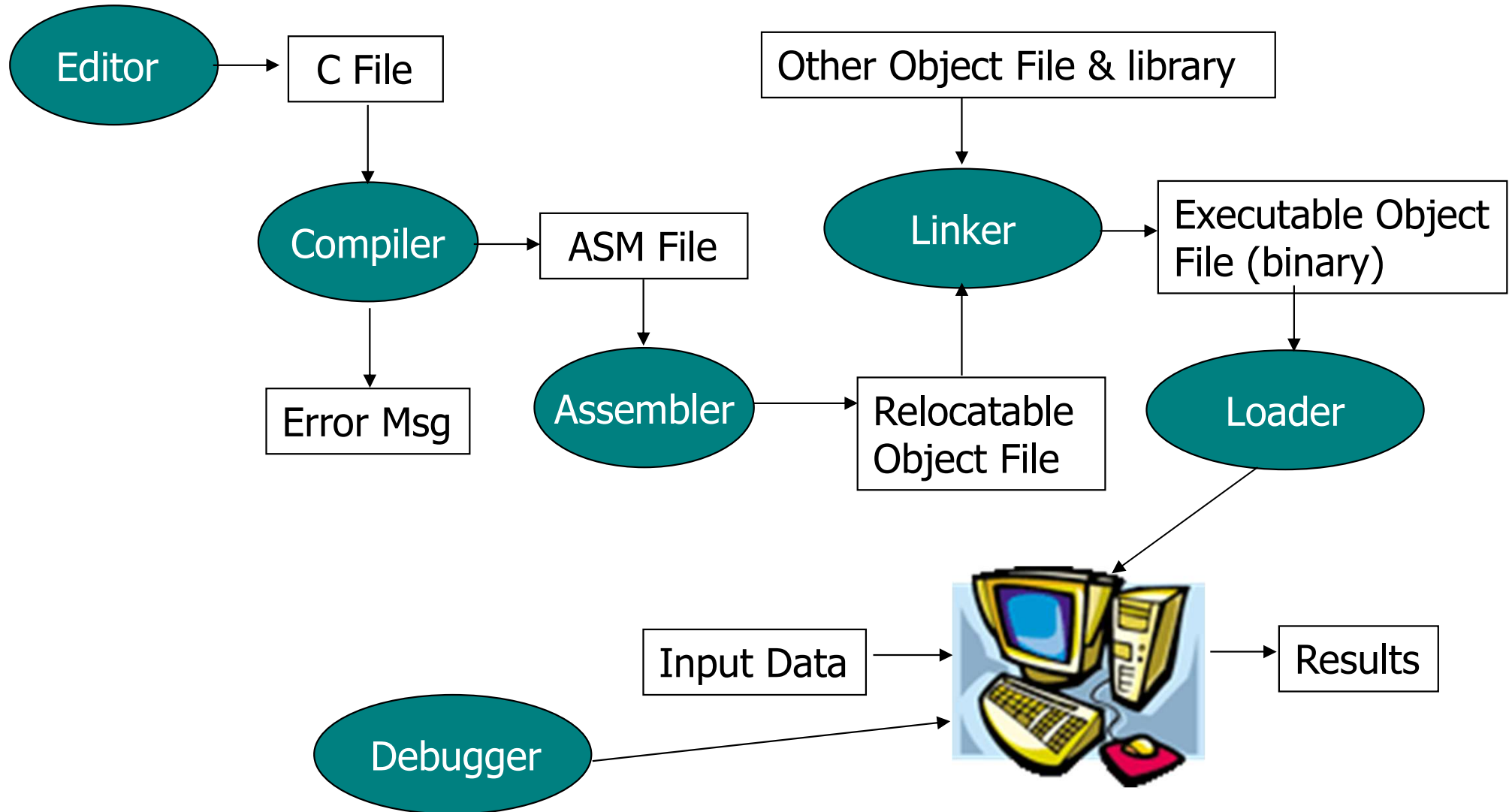
```
...  
00a1 8893 0408  
0305 8493 0408  
00a3 6c94 0408  
...
```



Compilation System (2/5)

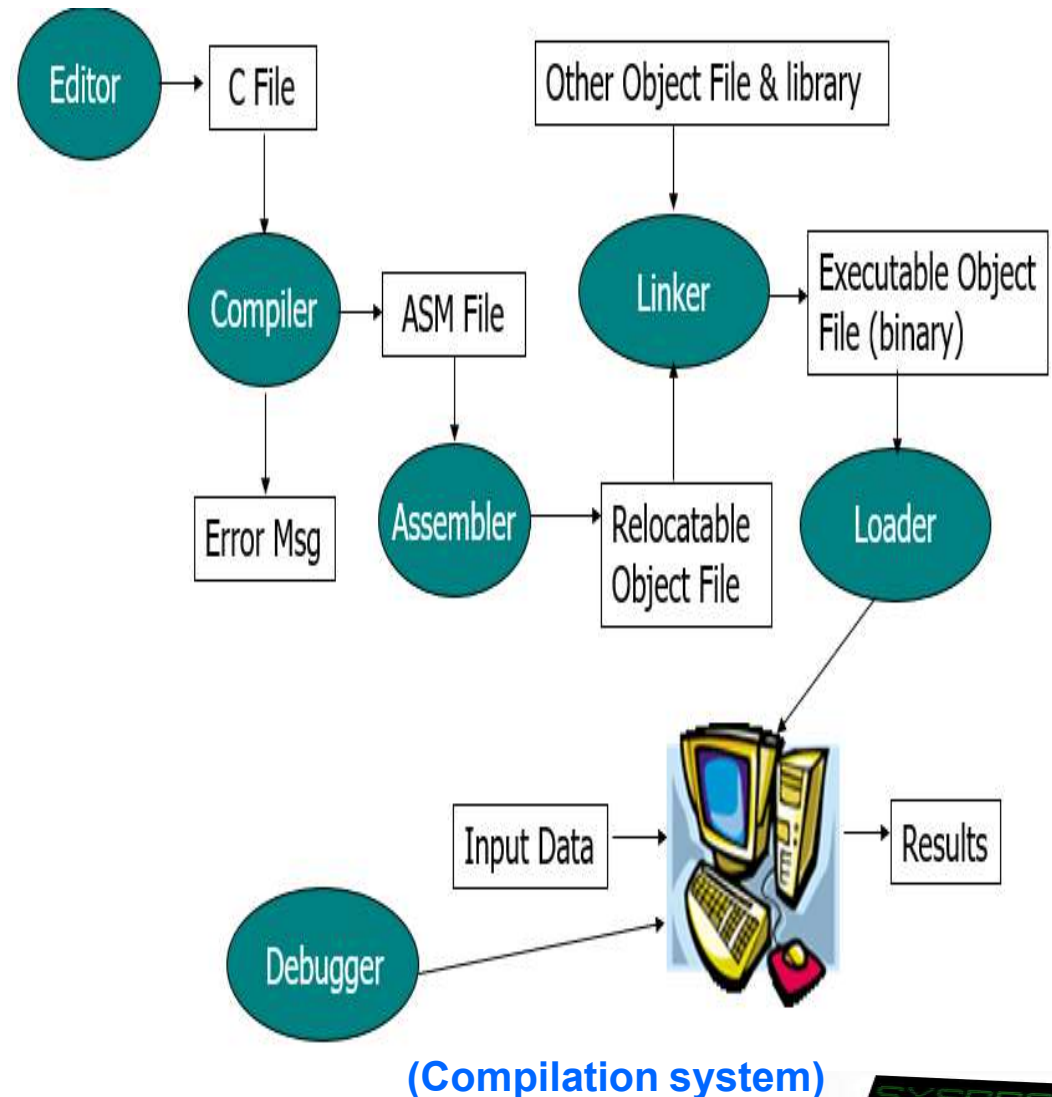
■ Overall structure

- ✓ 6 key components

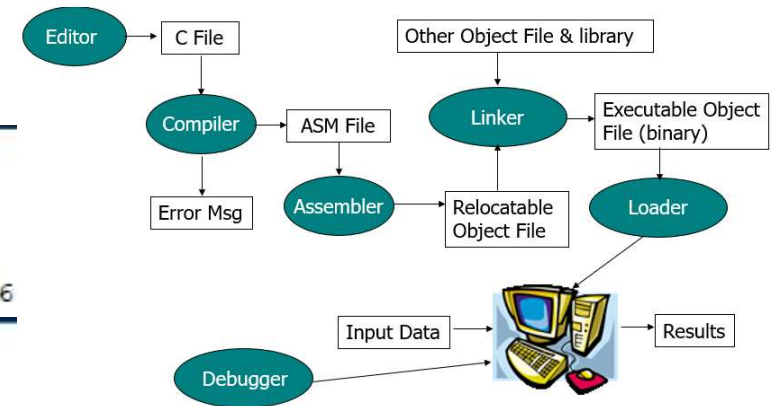


Compilation System (3/5)

■ Relation between Language Hierarchy and Overall Structure



■ Example in Linux



```

choijm@embedded: ~/syspro/chap1
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ uname -a
Linux embedded 4.13.0-36-generic #40~16.04.1-Ubuntu SMP Fri Feb 16
2018 x86_64 x86_64 x86_64 GNU/Linux
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ ifconfig
enp0s25    Link encap:Ethernet  HWaddr 82:55:c2:00:00:00
            inet addr:220.149.1.1  Bcast:220.149.1.255  Mask:255.255.255.0
            inet6 addr: fe80::205:5c2:0000:0000/64 ScopeLink
            UP BROADCAST RUNNING SLAVE  MTU:1500  Metric:0
            RX packets:8576625  TX packets:174094
            collisions:0 txqueuelen:1000
            RX bytes:1061179184 (101.5 MiB)  Interrupt:16 Memory:0
lo          Link encap:Local Loopback
            inet addr:127.0.0.1  Bcast:127.0.0.1  Mask:255.255.255.0
            inet6 addr: ::1/128 ScopeHost
            UP LOOPBACK RUNNING  MTU:65536  Metric:0
            RX packets:820  TX packets:820
            collisions:0 txqueuelen:1000
            RX bytes:70246 (70.2 KiB)
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ vi hello.c
choijm@embedded:~/syspro/chap1$ cat hello.c
#include <stdio.h>

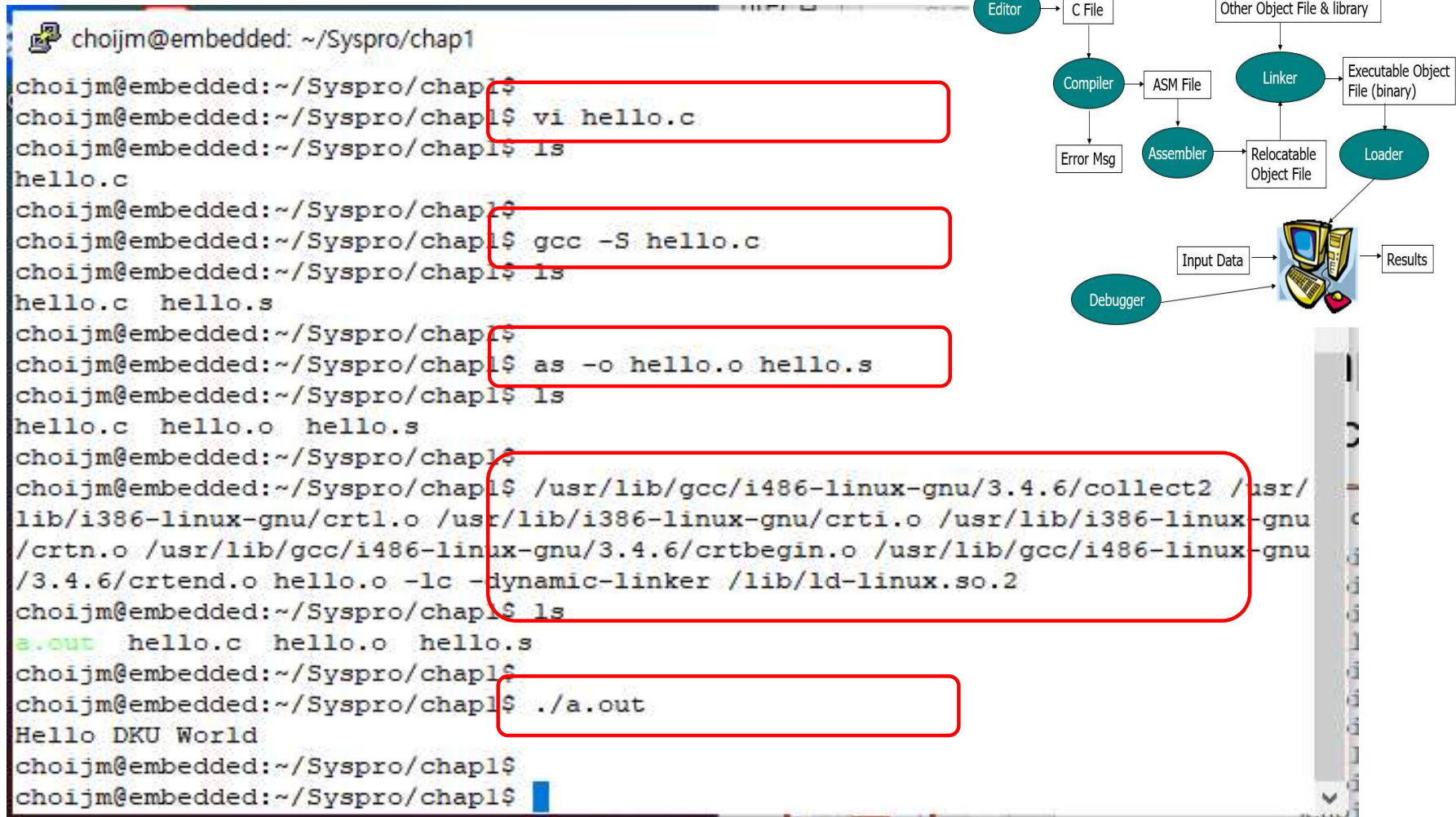
int main()
{
    printf("Hello DKU World\n");
}
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ ls
hello.c
choijm@embedded:~/syspro/chap1$ gcc hello.c
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ ls
a.out hello.c
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ ./a.out
Hello DKU World
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$

```


Compilation System (5/5)

■ Example in Linux: details

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



☞ What are the differences btw hello.c and hello.s?

☞ What are the differences btw hello.o and a.out?



Aside (Optional)

■ Compilation system in CSAPP

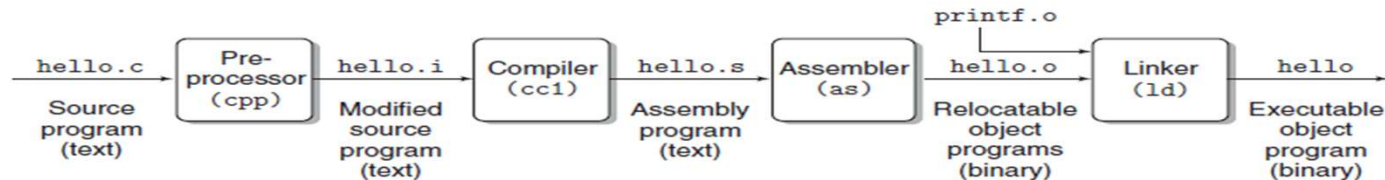


Figure 1.3 The compilation system.

Here, the `gcc` compiler driver reads the source file `hello.c` and translates it into an executable object file `hello`. The translation is performed in the sequence of four phases shown in Figure 1.3. The programs that perform the four phases (*preprocessor*, *compiler*, *assembler*, and *linker*) are known collectively as the *compilation system*.

- *Preprocessing phase.* The preprocessor (`cpp`) modifies the original C program according to directives that begin with the `#` character. For example, the `#include <stdio.h>` command in line 1 of `hello.c` tells the preprocessor to read the contents of the system header file `stdio.h` and insert it directly into the program text. The result is another C program, typically with the `.i` suffix.
- *Compilation phase.* The compiler (`cc1`) translates the text file `hello.i` into the text file `hello.s`, which contains an *assembly-language program*. Each statement in an assembly-language program exactly describes one low-level machine-language instruction in a standard text form. Assembly language is useful because it provides a common output language for different compilers for different high-level languages. For example, C compilers and Fortran compilers both generate output files in the same assembly language.
- *Assembly phase.* Next, the assembler (`as`) translates `hello.s` into machine-language instructions, packages them in a form known as a *relocatable object program*, and stores the result in the object file `hello.o`. The `hello.o` file is a binary file whose bytes encode machine language instructions rather than characters. If we were to view `hello.o` with a text editor, it would appear to be gibberish.
- *Linking phase.* Notice that our `hello` program calls the `printf` function, which is part of the *standard C library* provided by every C compiler. The `printf` function resides in a separate precompiled object file called `printf.o`, which must somehow be merged with our `hello.o` program. The linker (`ld`) handles this merging. The result is the `hello` file, which is an *executable object file* (or simply *executable*) that is ready to be loaded into memory and executed by the system.

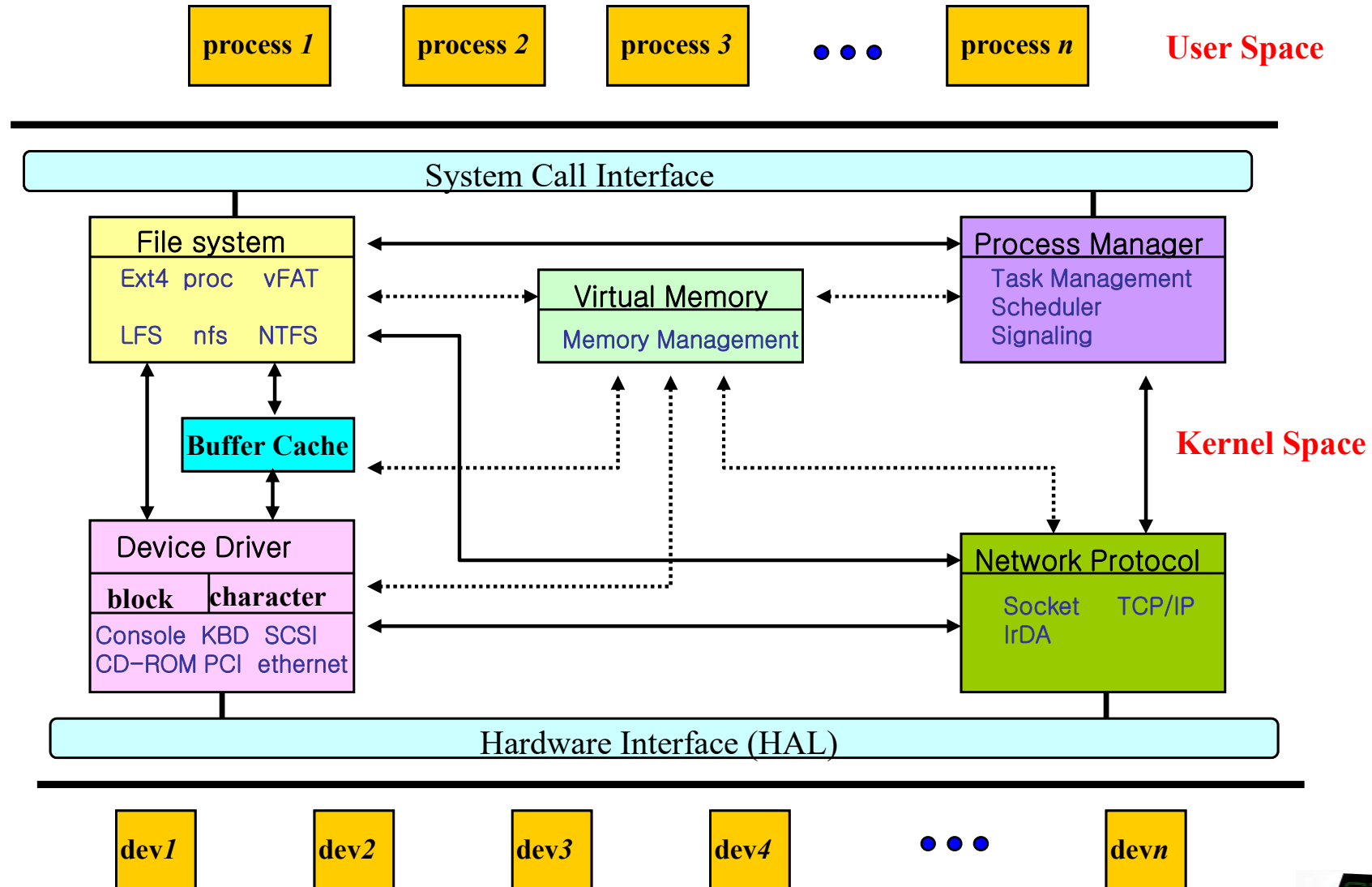
(Source: CSAPP, Chapter 1)



Operating System (1/15)

■ Overall structure

- ✓ 7 key components

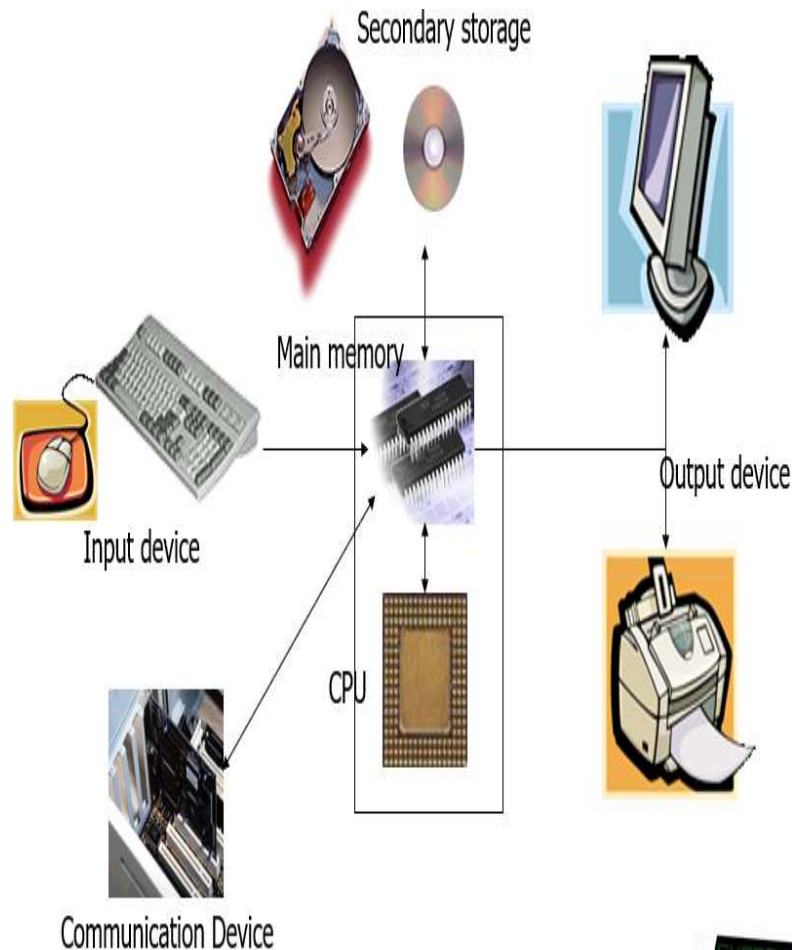


(Source: Linux Kernel Internals)

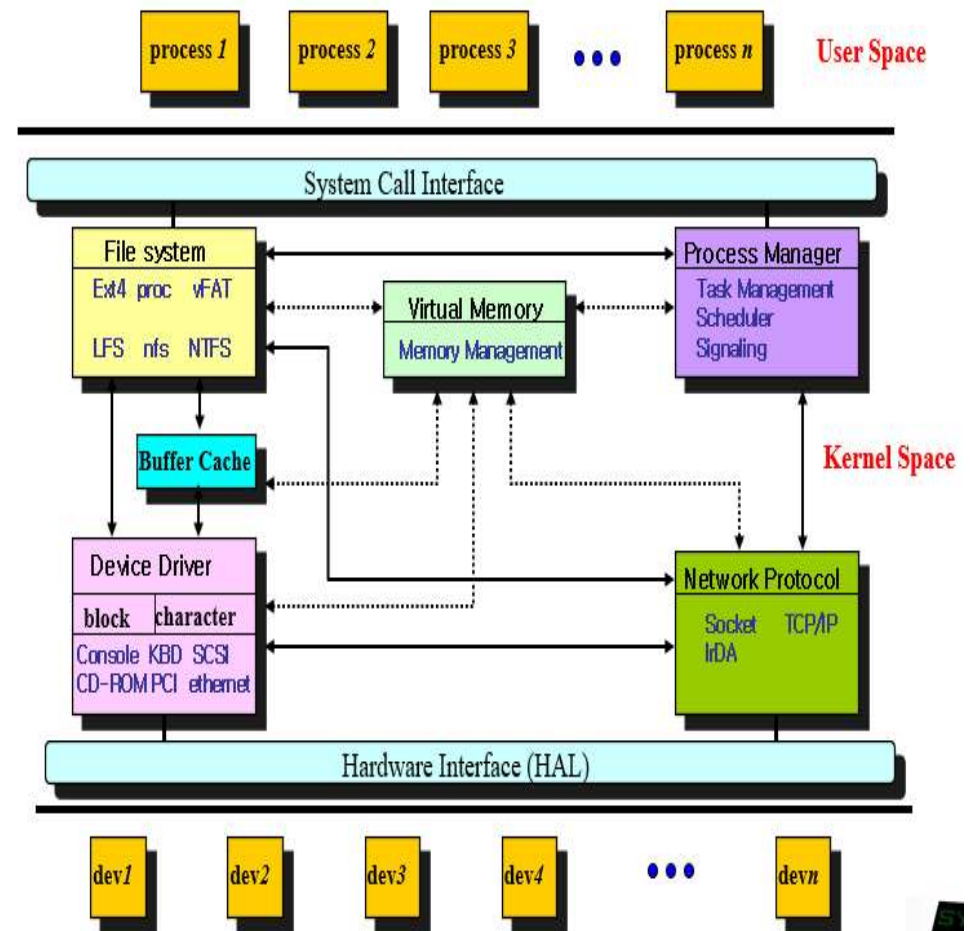


Operating System (2/15)

- Relation between hardware component and overall structure
 - ✓ OS: resource manager → abstract HW resources into logical ones



(Physical resources)

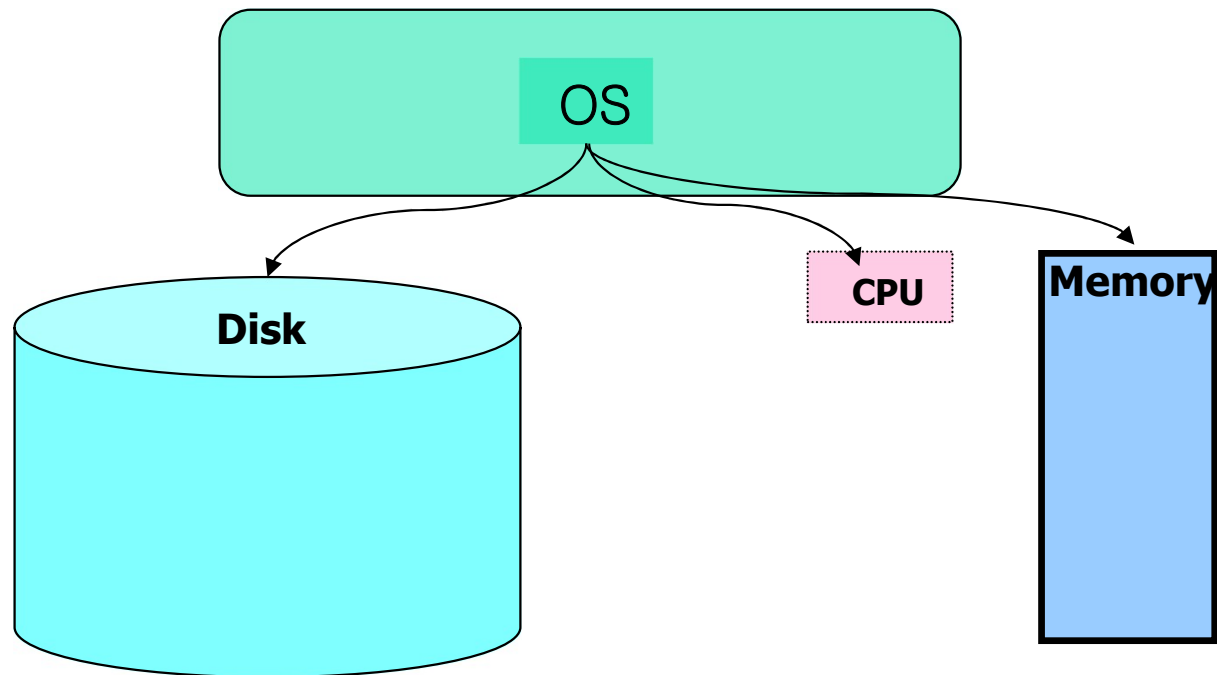


(Logical resources)



Operating System (3/15)

- Behaviors: 1) initial state



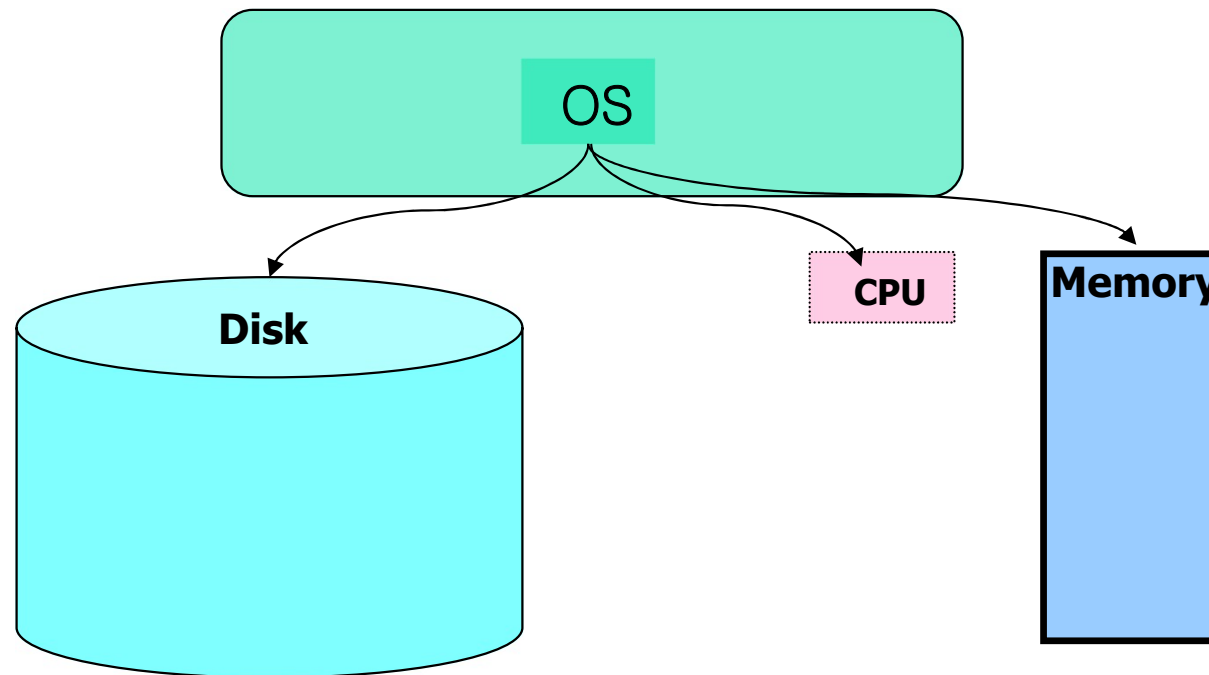
Operating System (4/15)

- Behaviors: 2) create a file (user's viewpoint)

vi hello.c

```
#include <stdio.h>

int main()
{
    printf("Hello world\n");
}
```



Operating System (5/15)

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

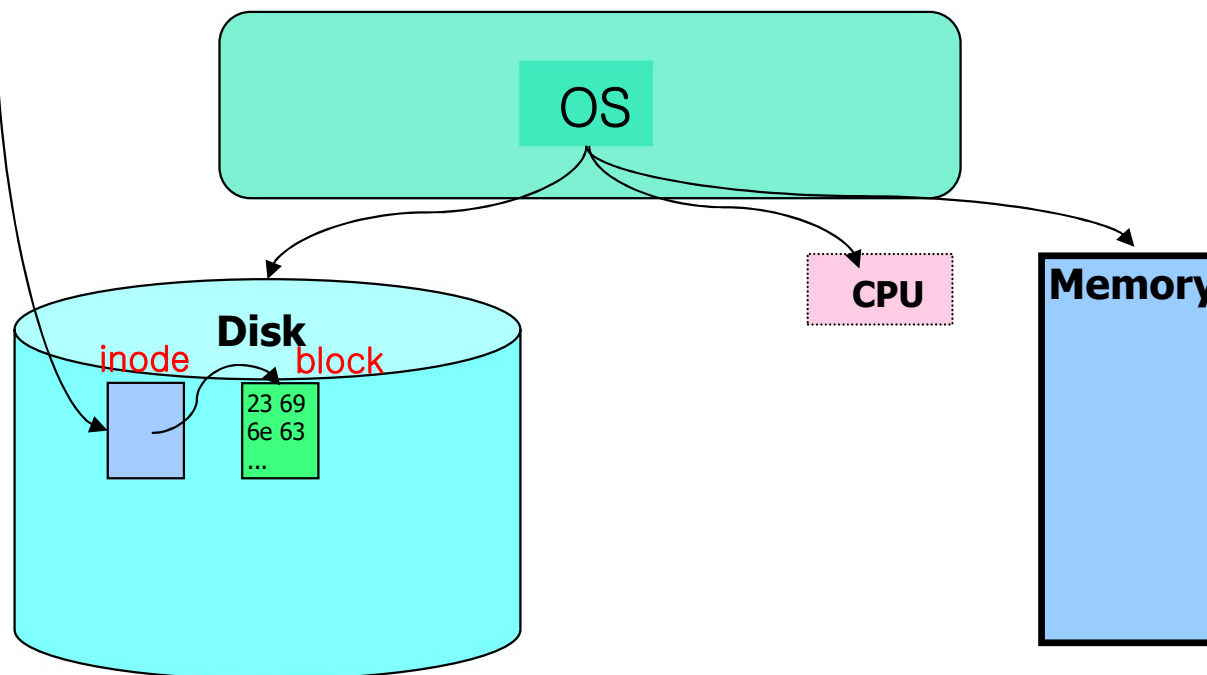
```
vi hello.c
#include <stdio.h>

int main()
{
    printf("Hello world\n");
}
```

#	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

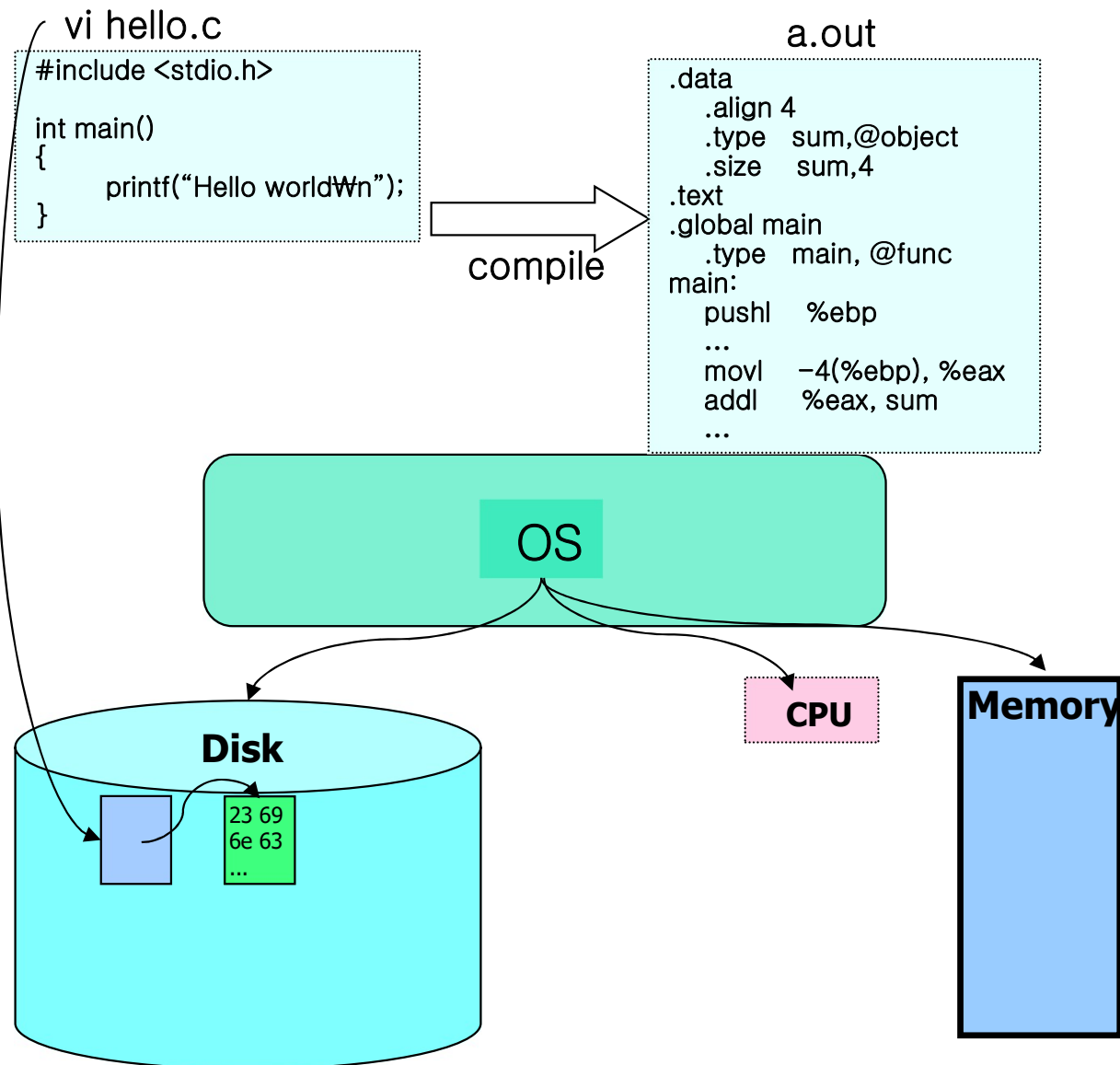
Figure 1.2 The ASCII text representation of hello.c.

(Source: CSAPP)



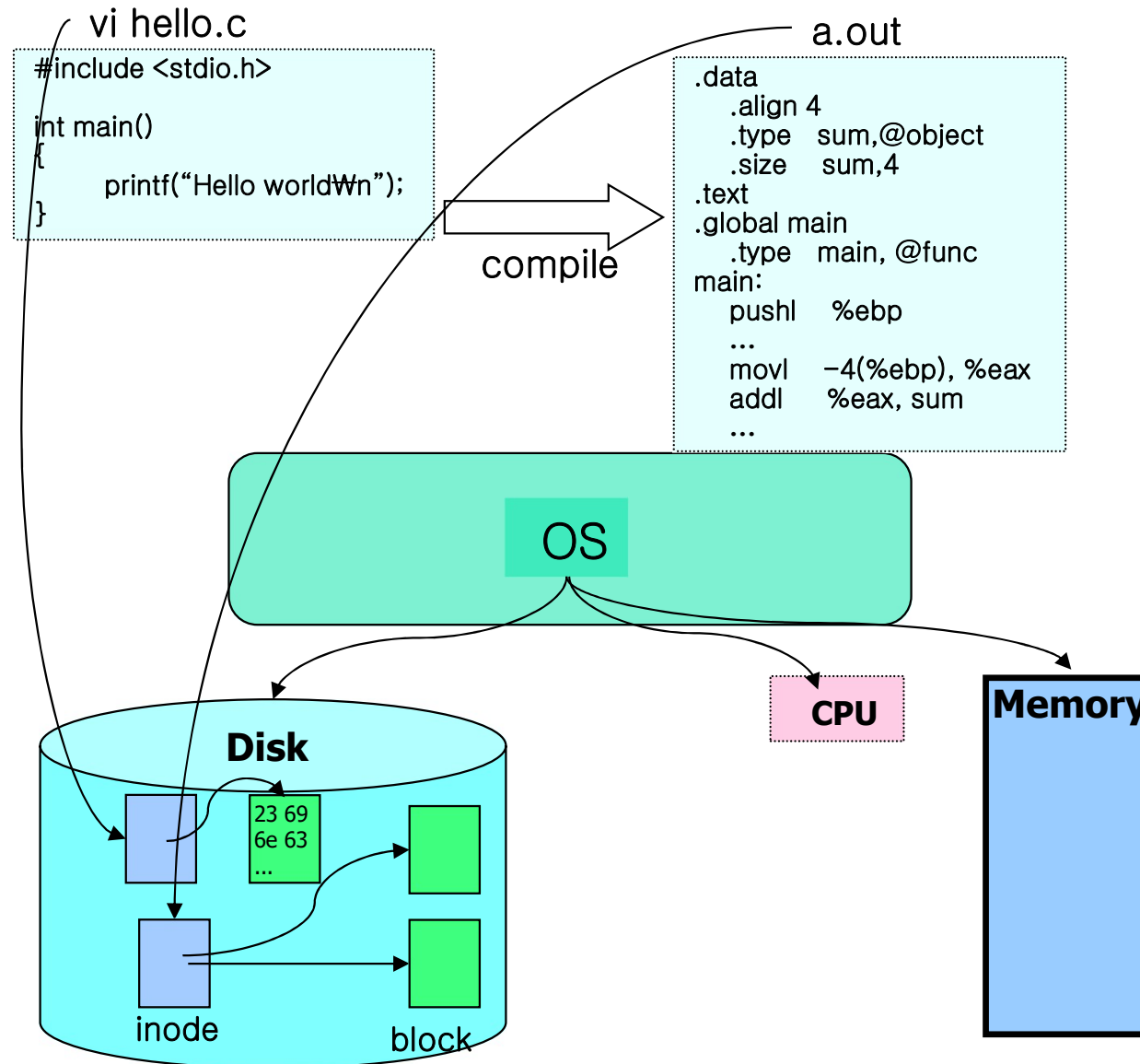
Operating System (6/15)

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



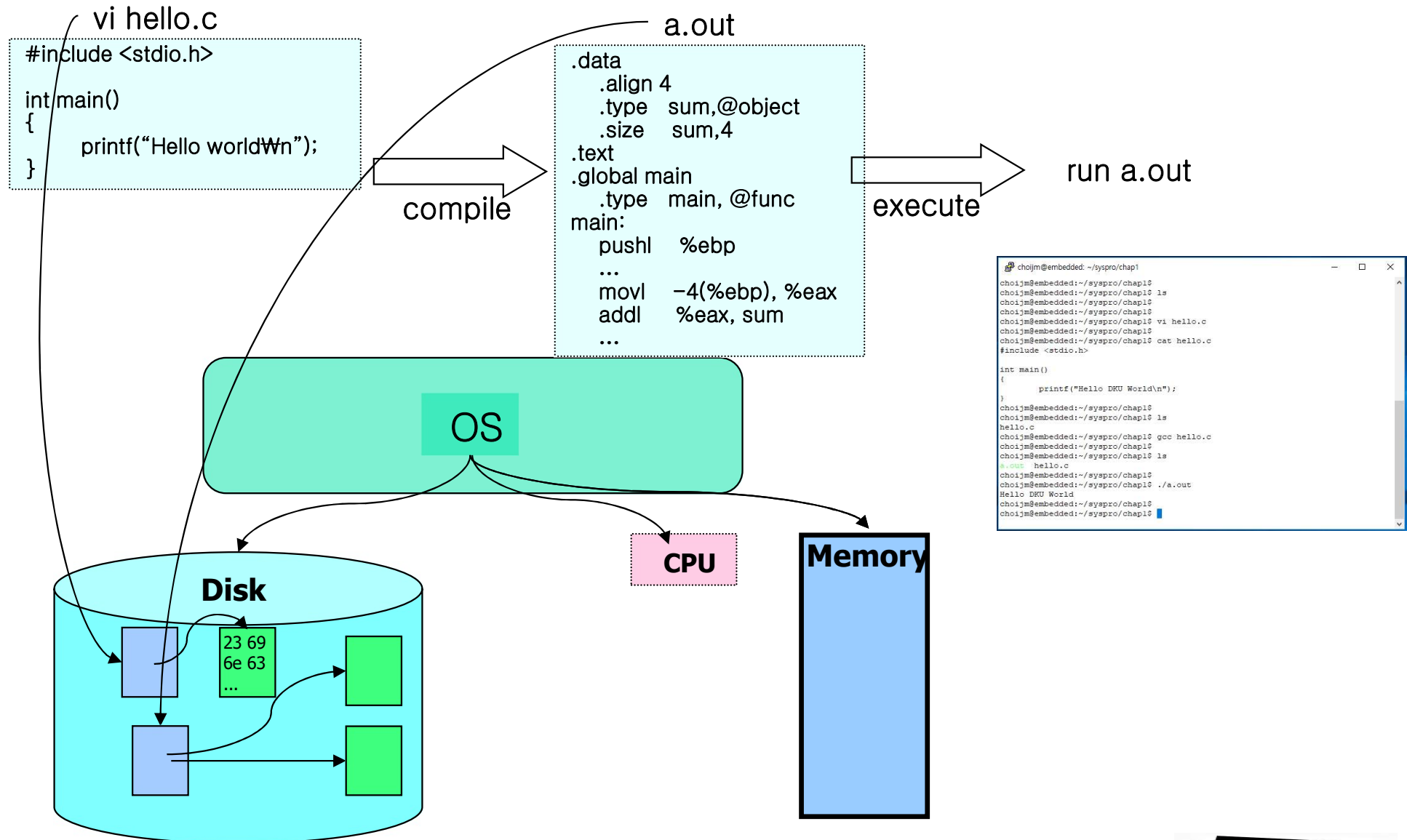
Operating System (7/15)

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



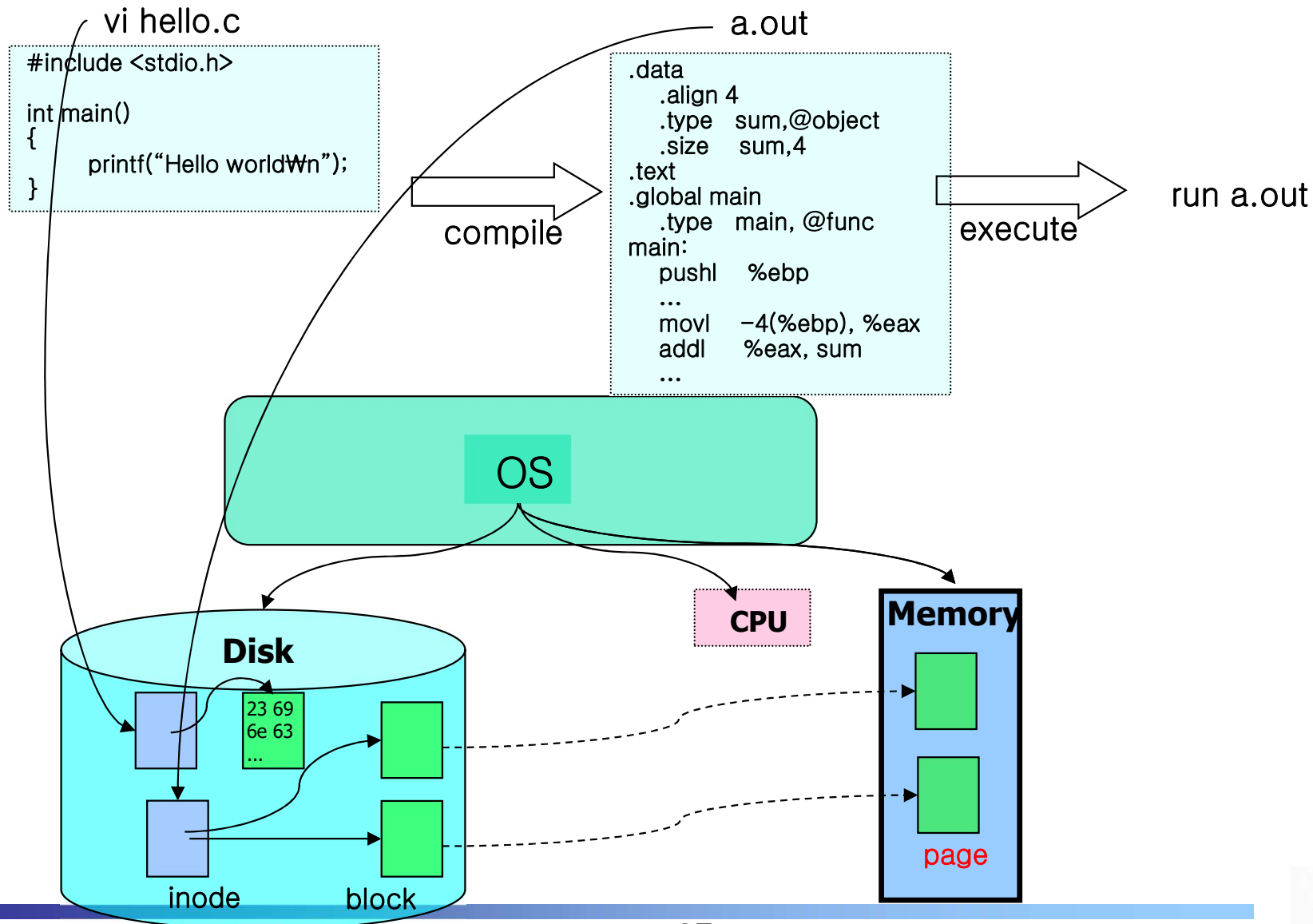
Operating System (8/15)

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



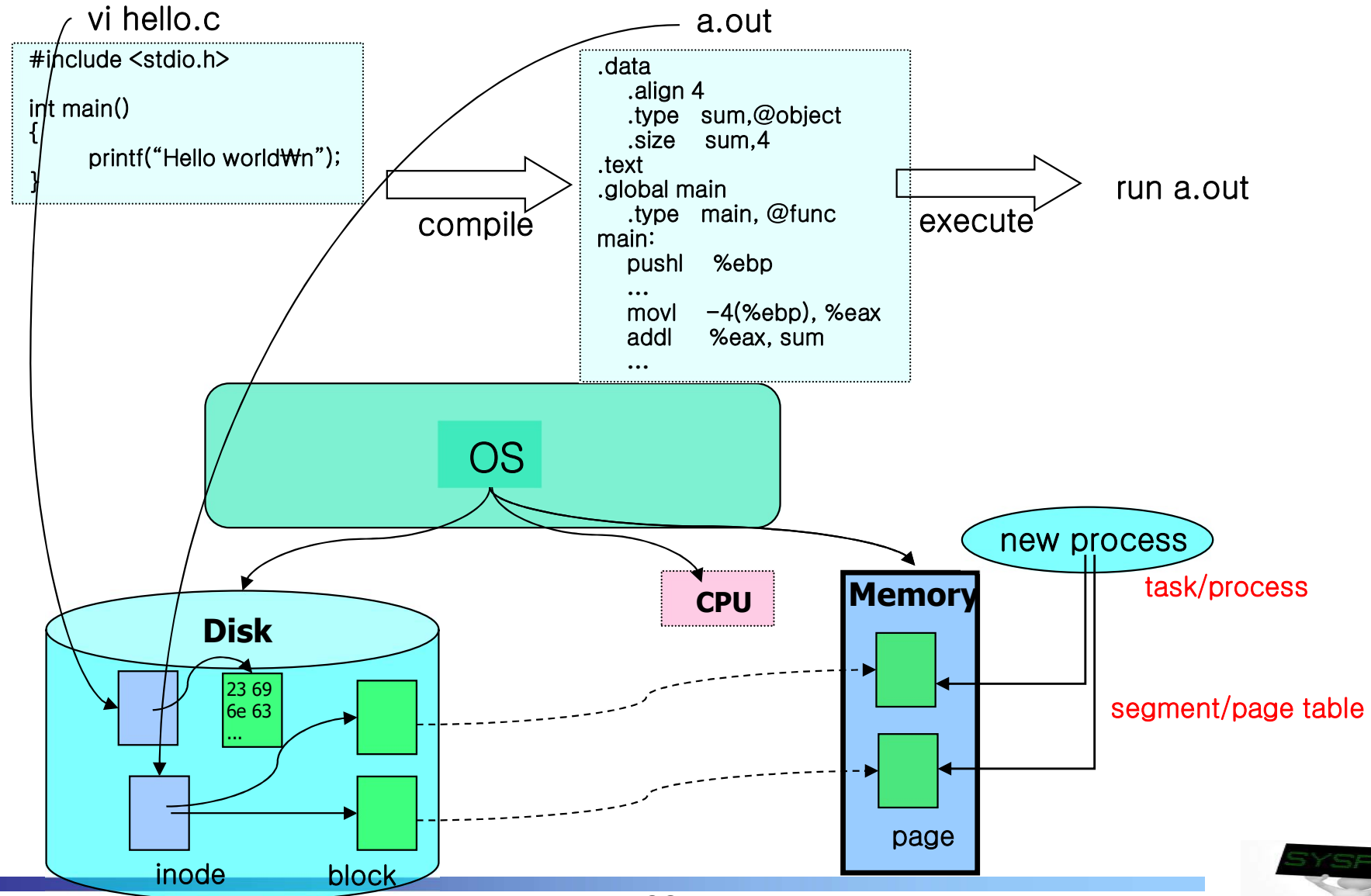
Operating System (9/13)

- Behaviors: 4) execute the a.out (system's viewpoint)
 - ✓ To run a.out, OS first loads it into memory



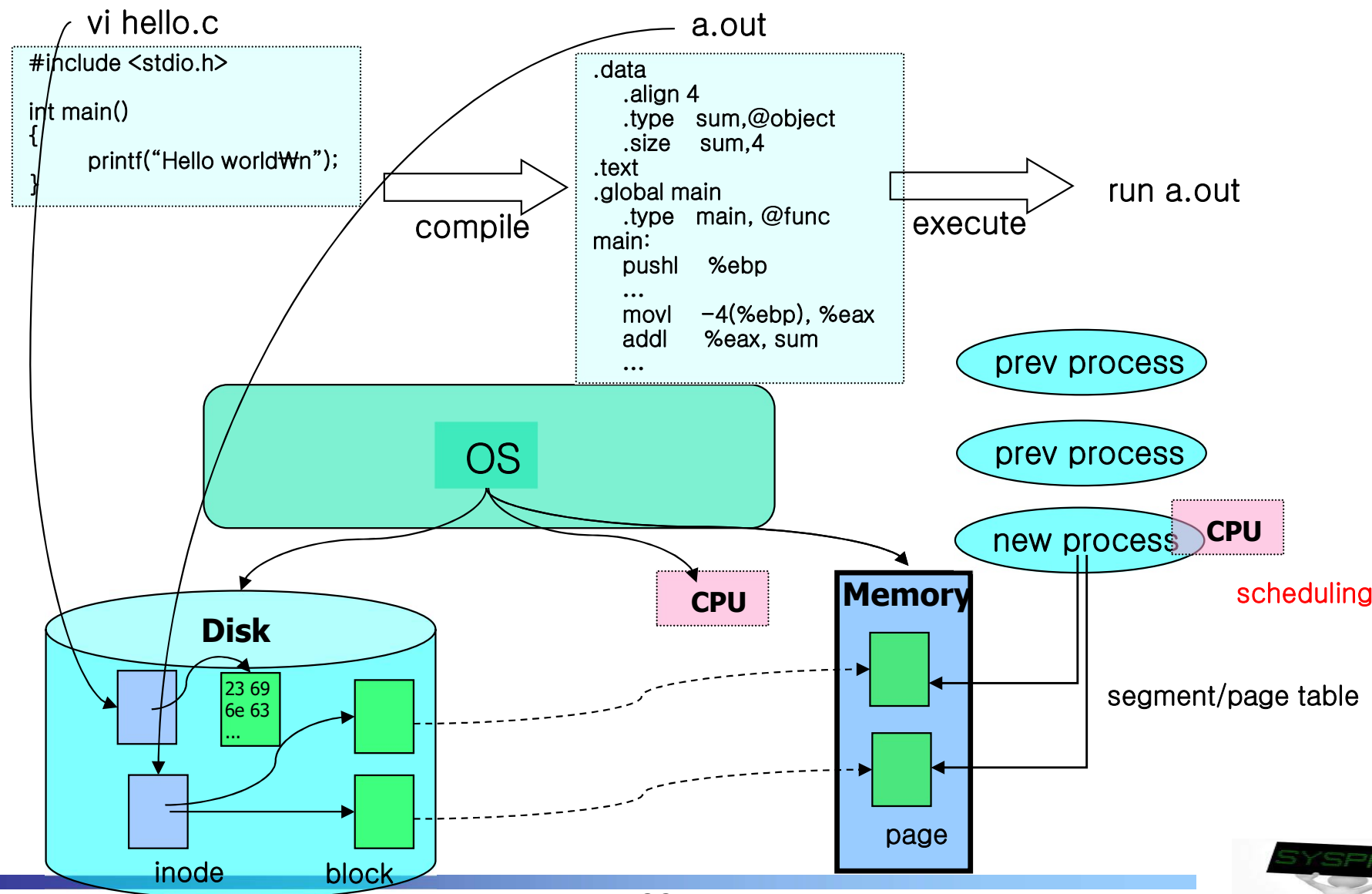
Operating System (10/13)

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



Operating System (11/13)

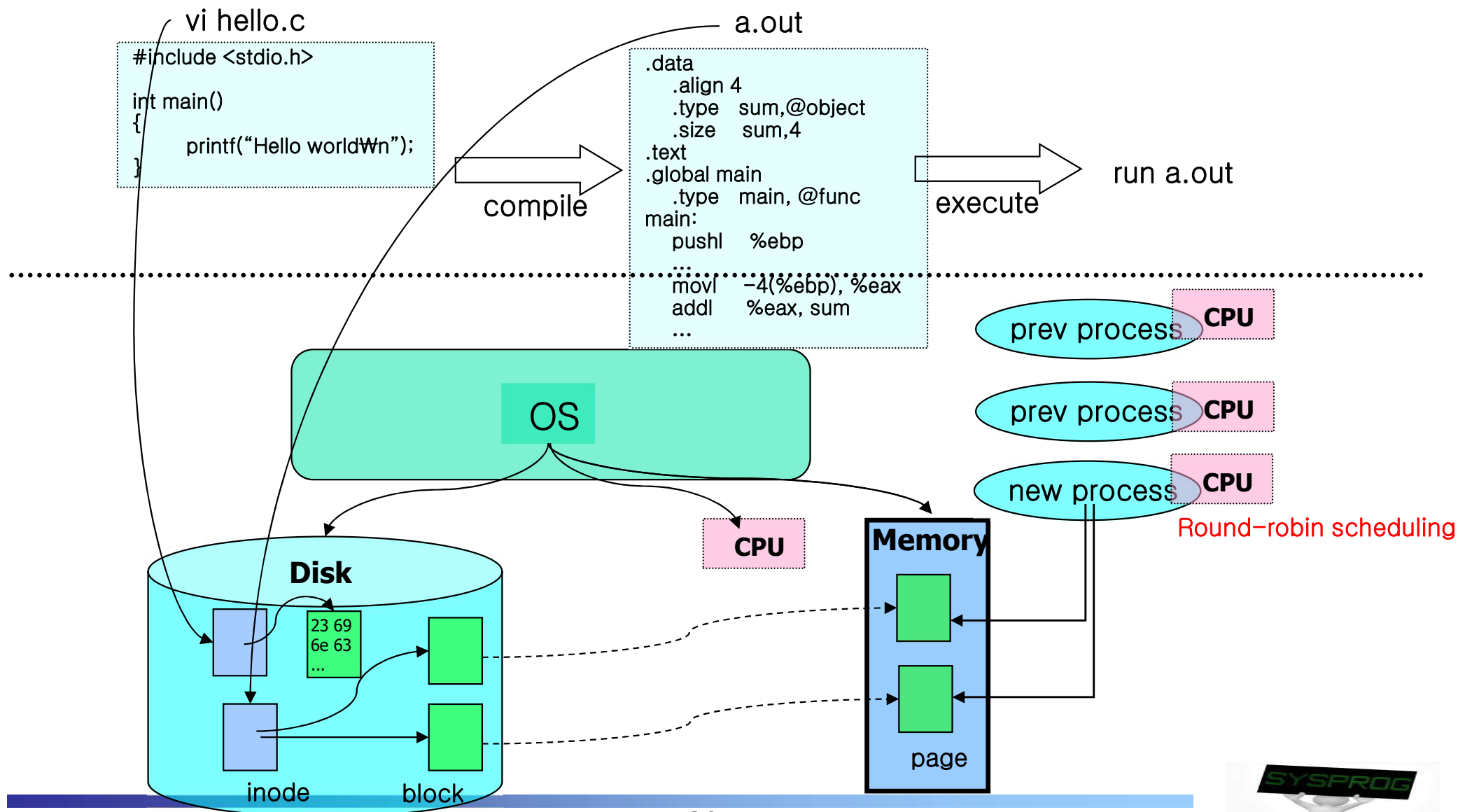
- Behaviors: 4) execute the a.out (system's viewpoint)
 - Then, OS makes a new process & **schedule** it



Operating System (12/13)

■ Behaviors: 4) execute the a.out (system's viewpoint)

- ✓ Then, OS makes a new process & schedule it with **time-sharing**

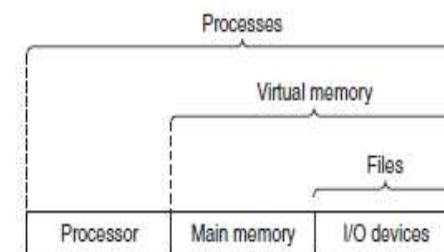


Operating System (13/13)

■ 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.



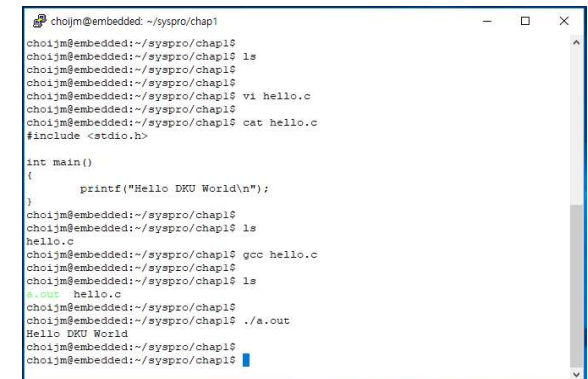
Runtime System (1/5)

■ 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, ...

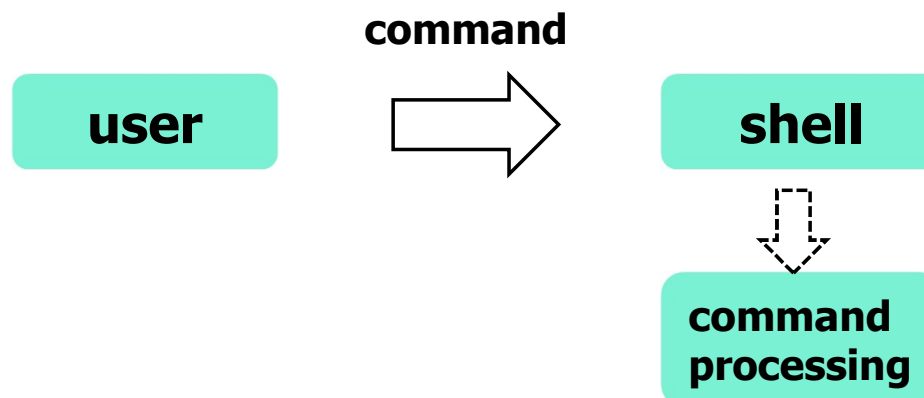
■ shell

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



```
choijm@embedded: ~/syspro/chap1
choijm@embedded:~/syspro/chap1$ ls
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ vi hello.c
choijm@embedded:~/syspro/chap1$ cat hello.c
#include <stdio.h>

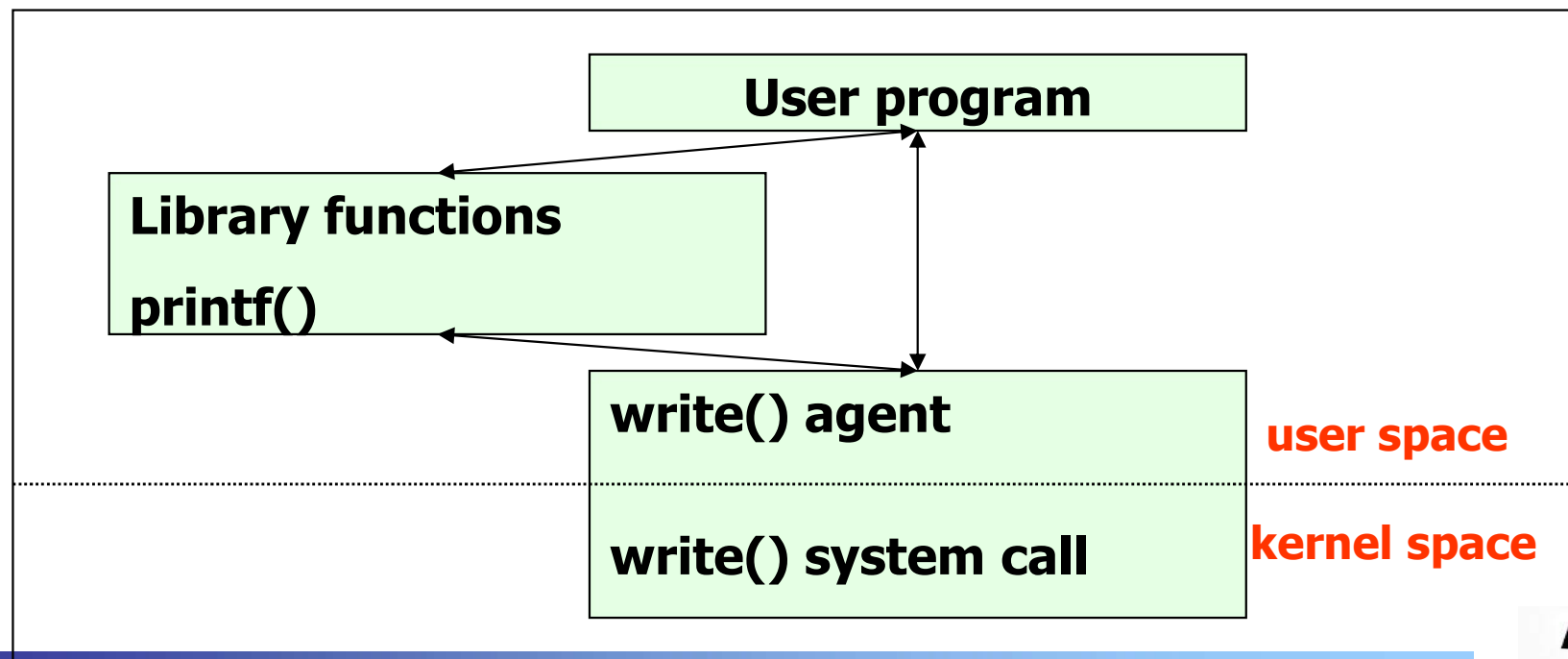
int main()
{
    printf("Hello DKU World\n");
}
choijm@embedded:~/syspro/chap1$ ls
hello.c
choijm@embedded:~/syspro/chap1$ gcc hello.c
choijm@embedded:~/syspro/chap1$ ls
a.out  hello.c
choijm@embedded:~/syspro/chap1$ ./a.out
Hello DKU World
choijm@embedded:~/syspro/chap1$
```



Runtime System (2/5)

■ library

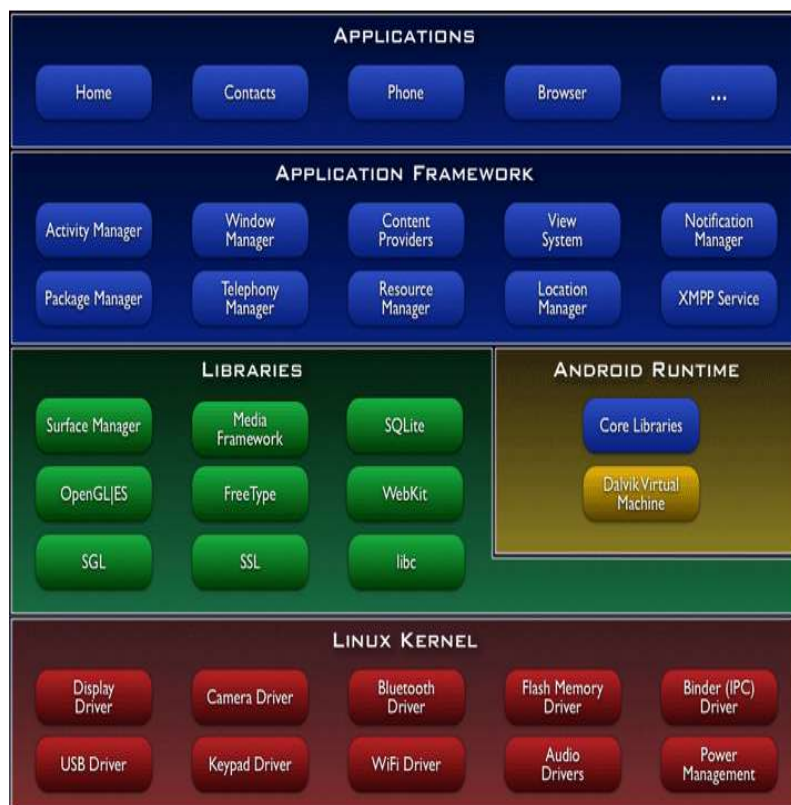
- ✓ A collection of functions, invoked frequently by a lot of users
 - Relocatable objects
 - Most languages have standard libraries (also programmers can make their own custom libraries using ar, ranlib and libtool.)
- ✓ Type
 - Static: 1).a, 2) statically linked (compile time), 3) simple
 - Shared: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



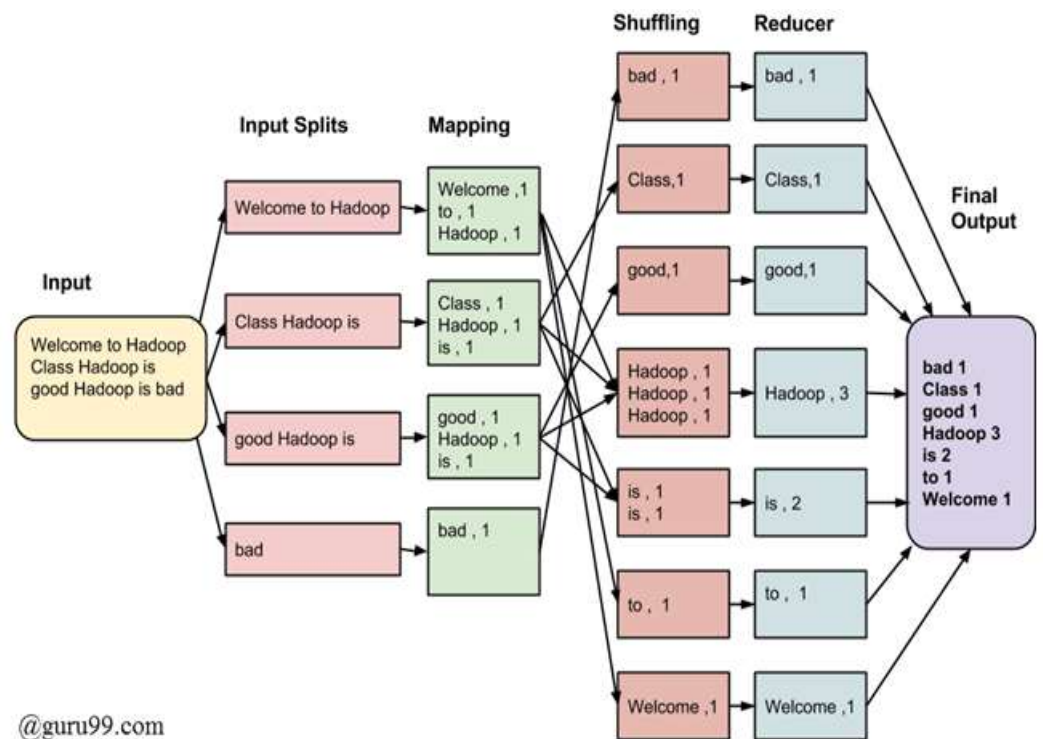
Runtime System (3/5)

■ 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)



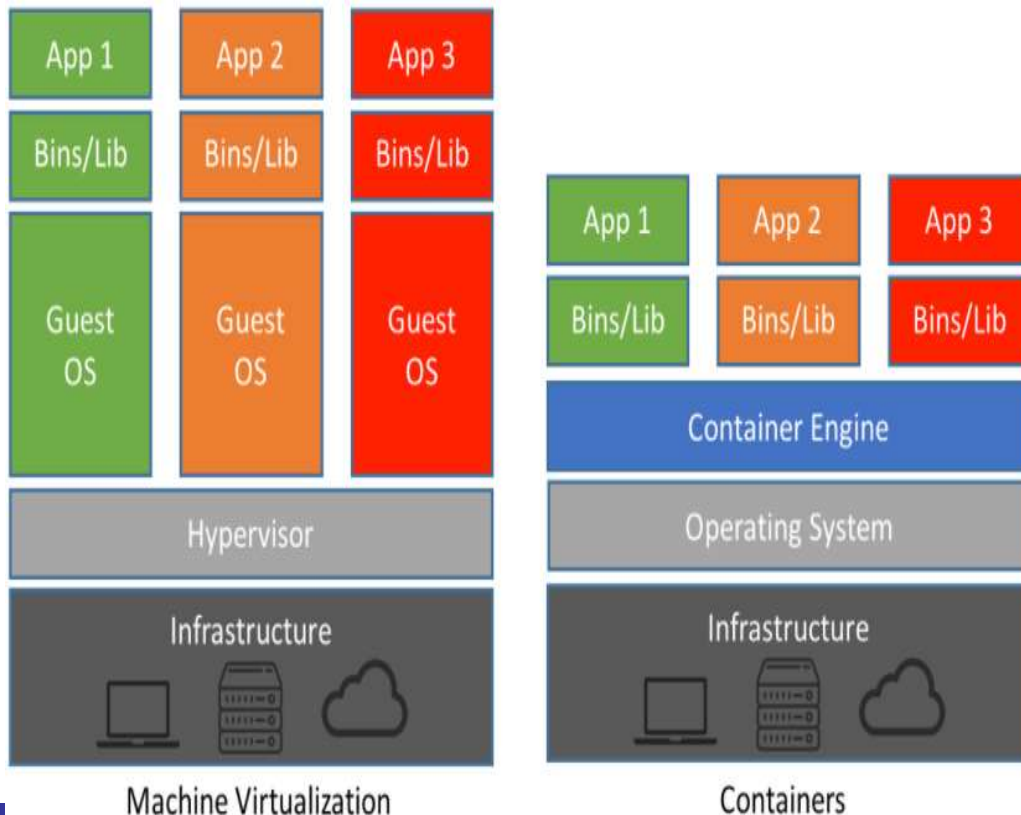
MapReduce Architecture

(Source: <https://www.guru99.com/introduction-to-mapreduce.html>)

Runtime System (4/5)

■ 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



```
[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:/#
```

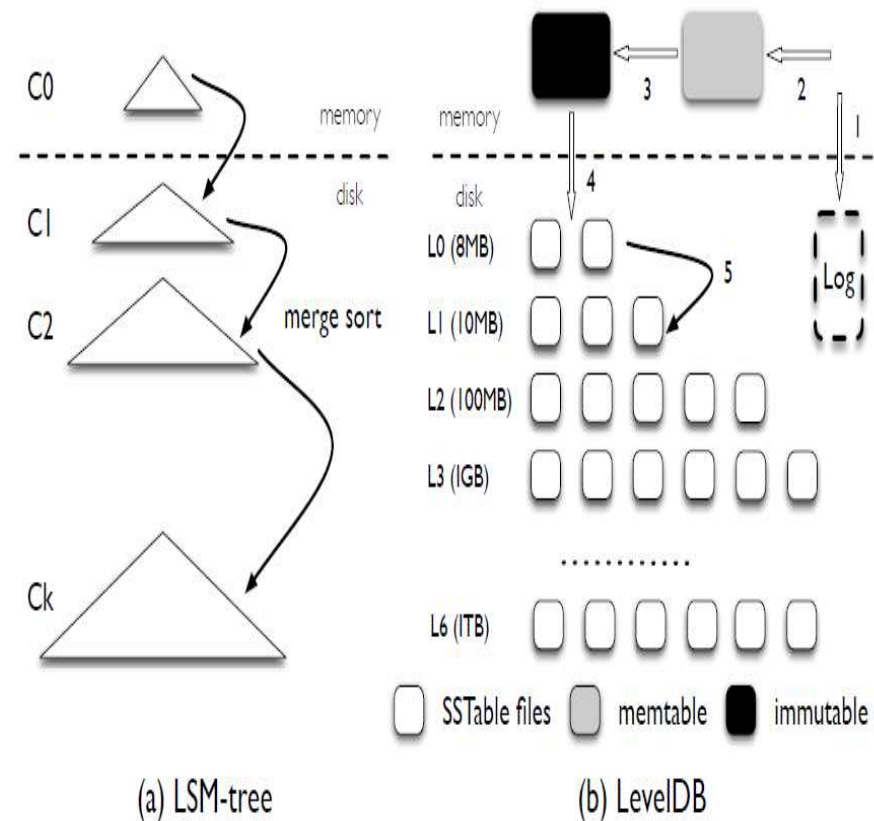
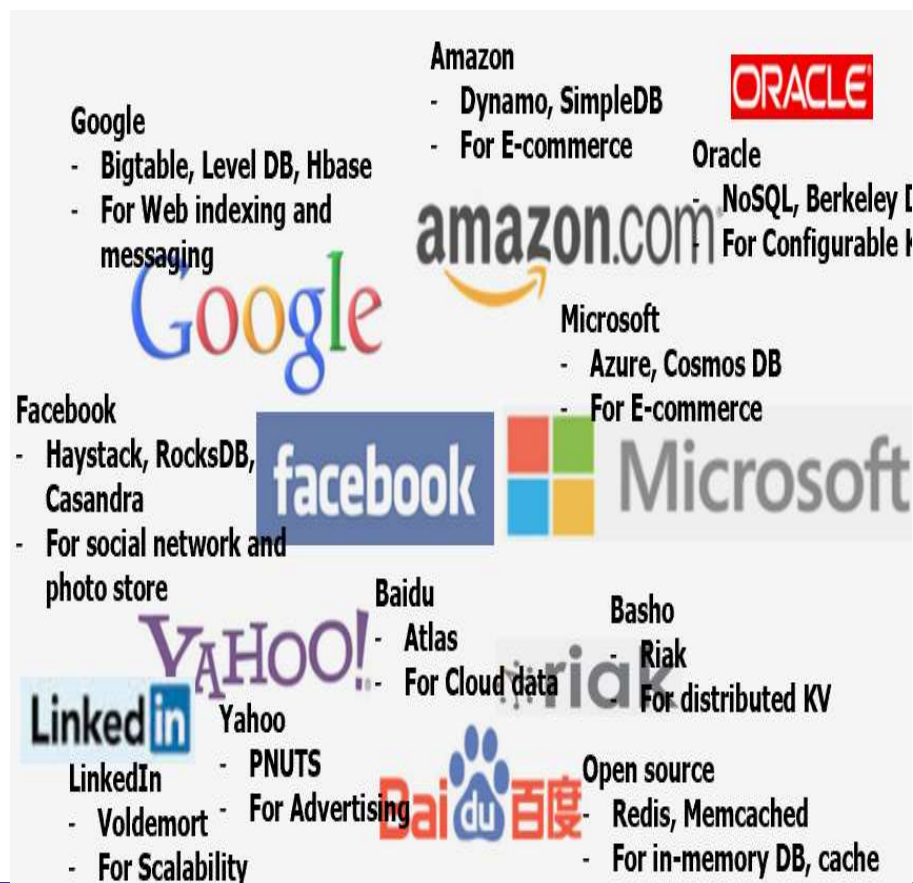
```
[root@docker ~]# docker ps
```



Runtime System (5/5)

■ 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, ...



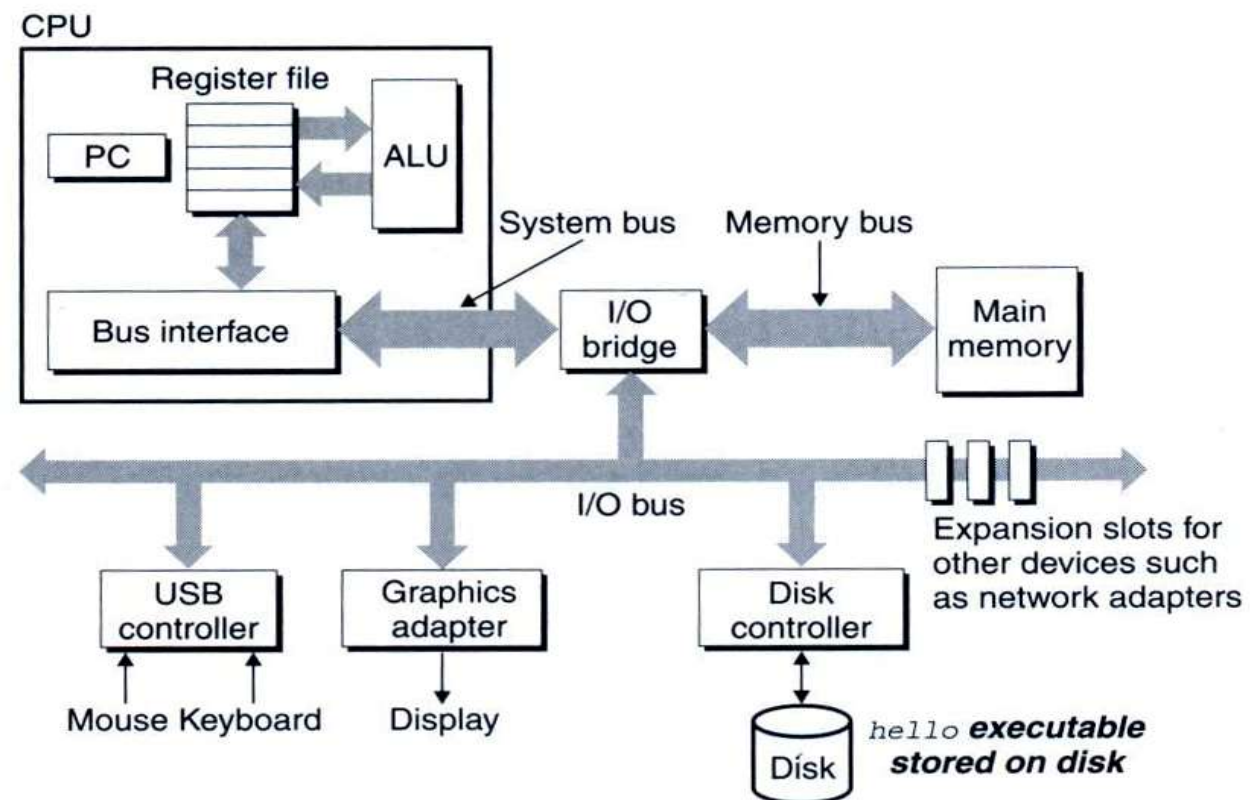
Hardware consideration (1/6)

■ 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)



Hardware consideration (2/6)

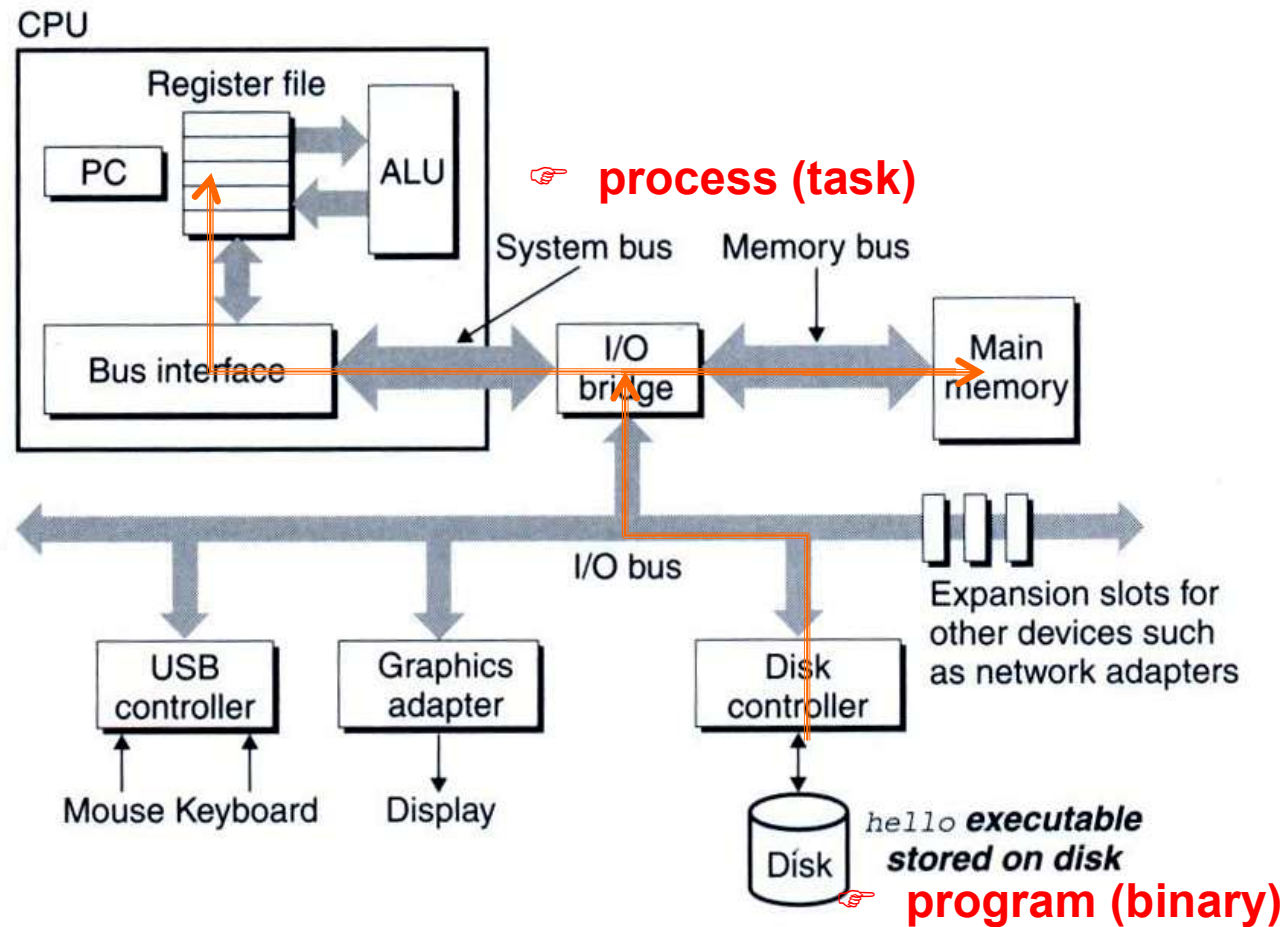
■ Computer organization

- ✓ 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.



Hardware consideration (3/6)

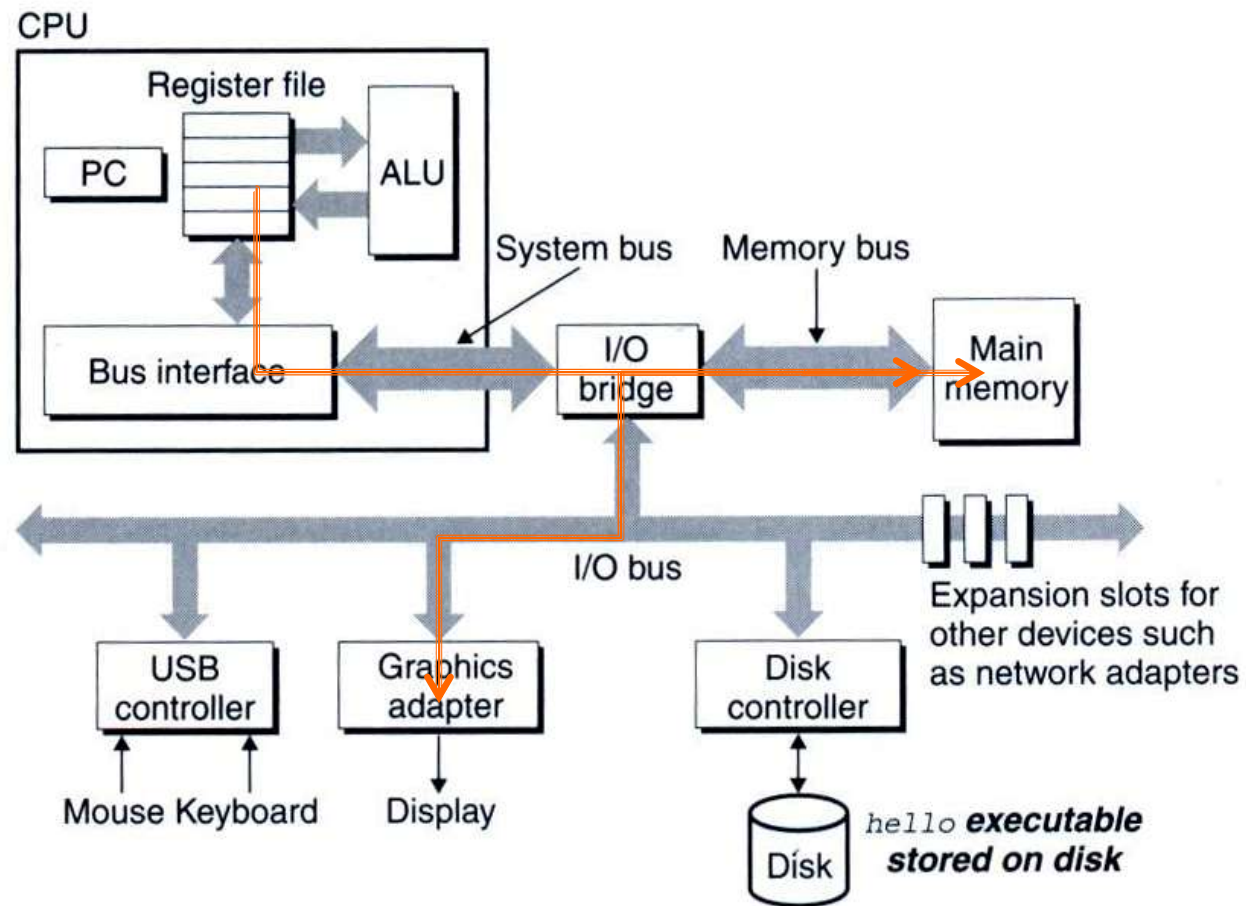
■ Computer organization

- ✓ 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.



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

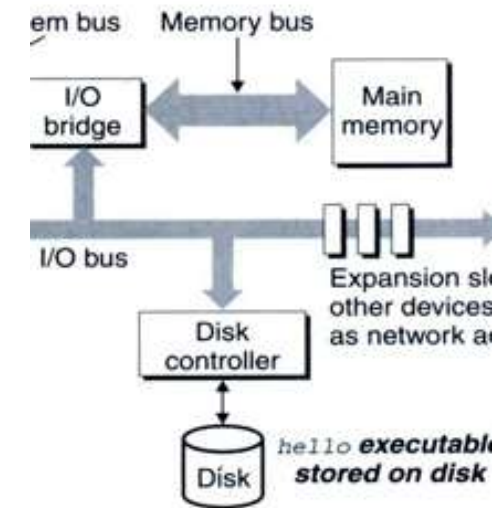
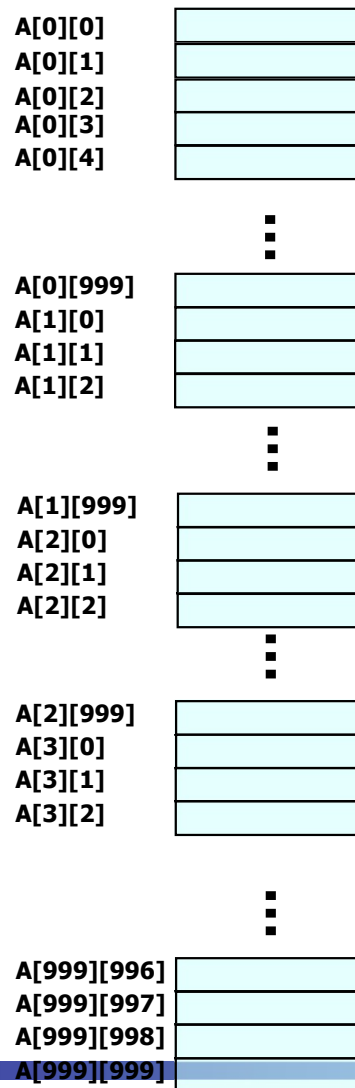
```
/* program B */  
int a[1000][1000];  
int i, j;  
....  
  
for (i=0; i<1000; i++)  
    for (j=0; j<1000; j++)  
        a[j][i] ++;
```



Hardware consideration (5/6)

■ Memory matters

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



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)



Abstraction (1/9)

■ 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.



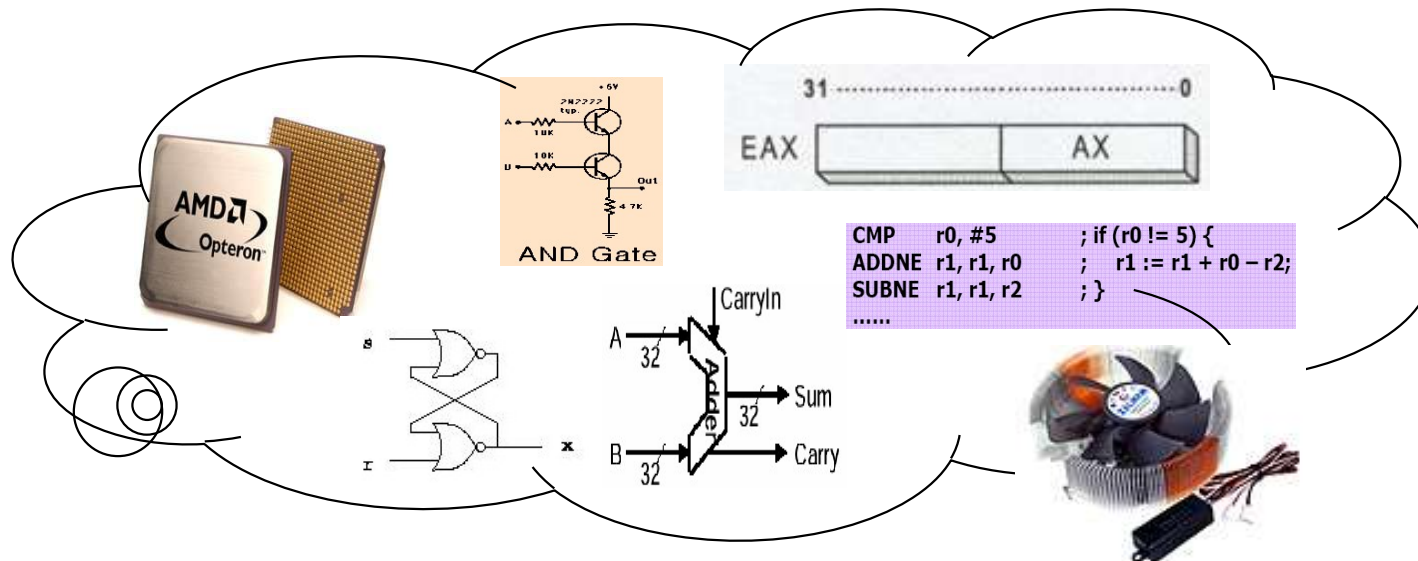
Abstraction (2/9)

■ CPU

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

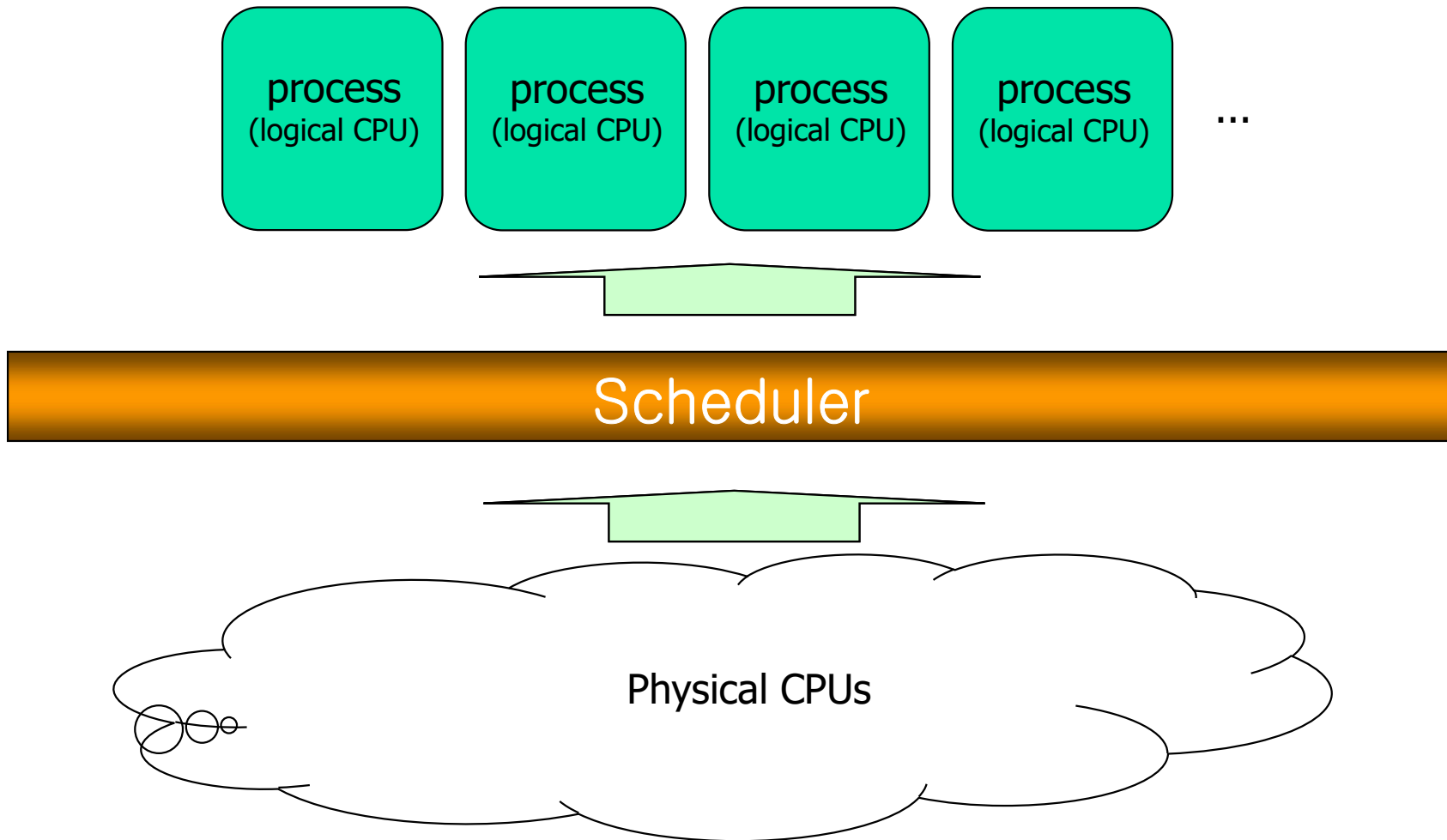


Compilation system



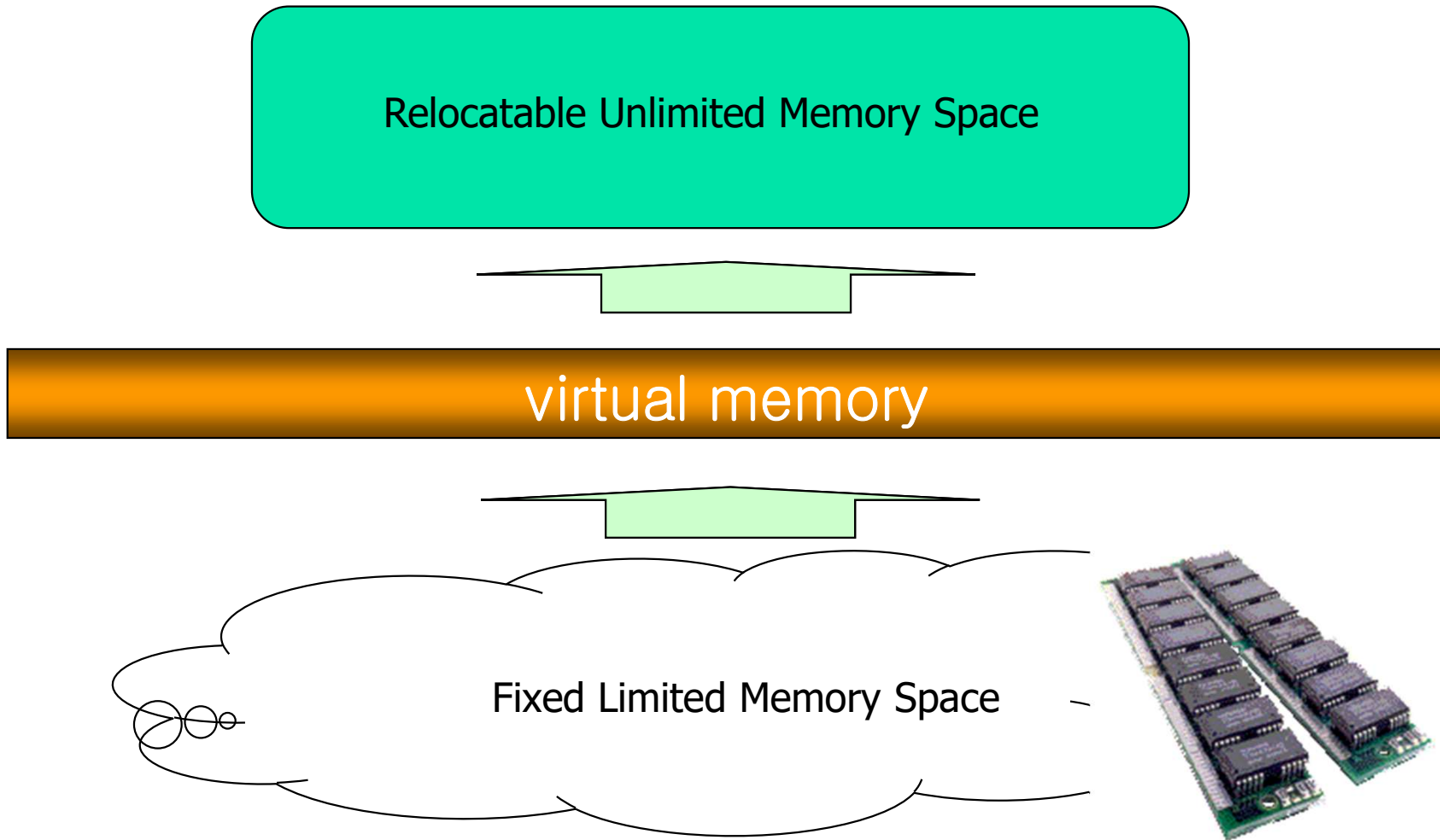
Abstraction (3/9)

■ Multitasking



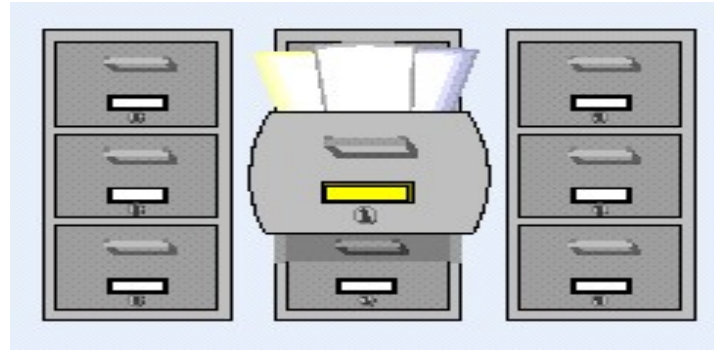
Abstraction (4/9)

- Memory management



Abstraction (5/9)

- File system



file system



Abstraction (6/9)

- Device driver

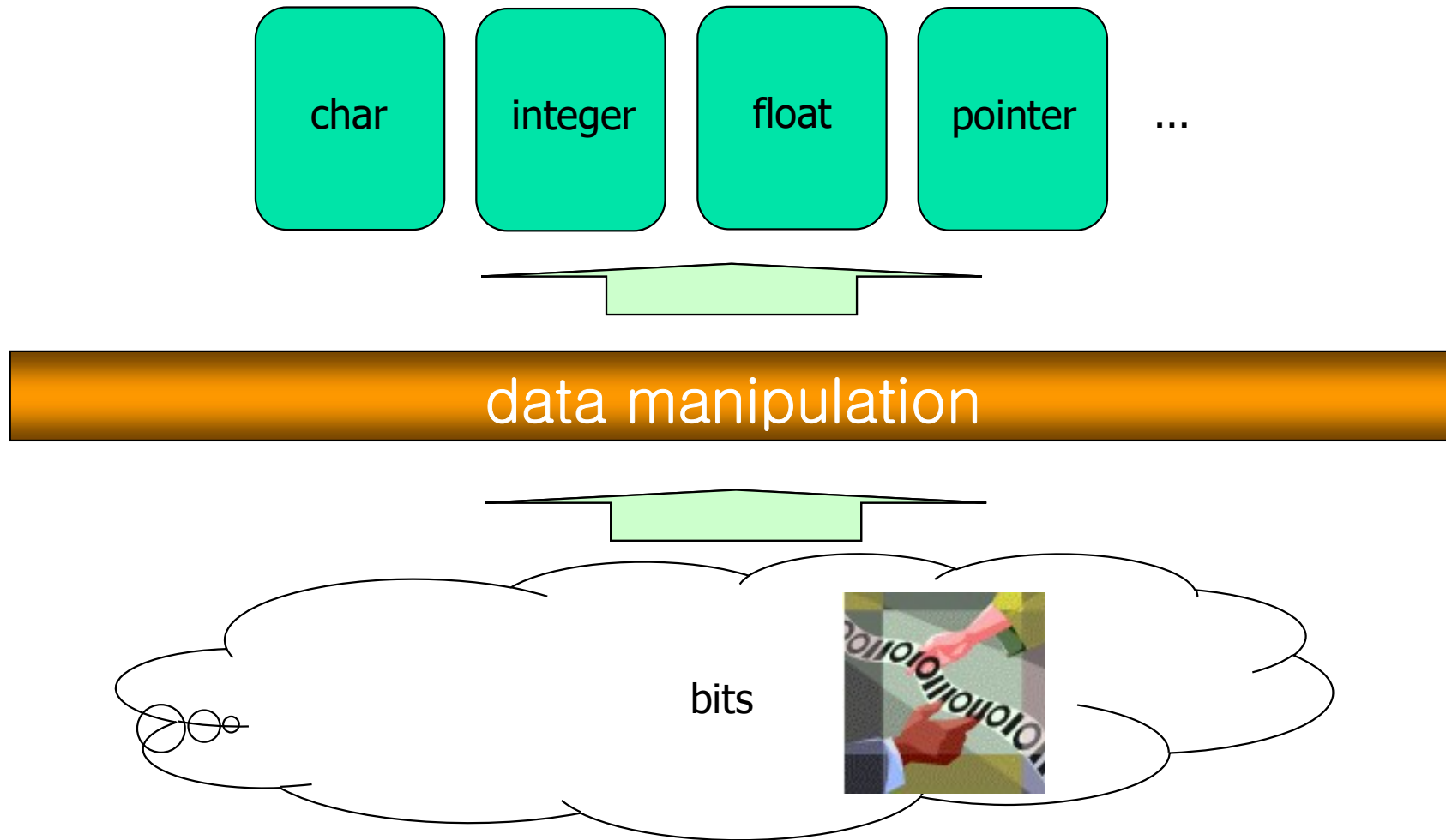
Handle and I/O STREAM
(open, read, write, close)

device driver



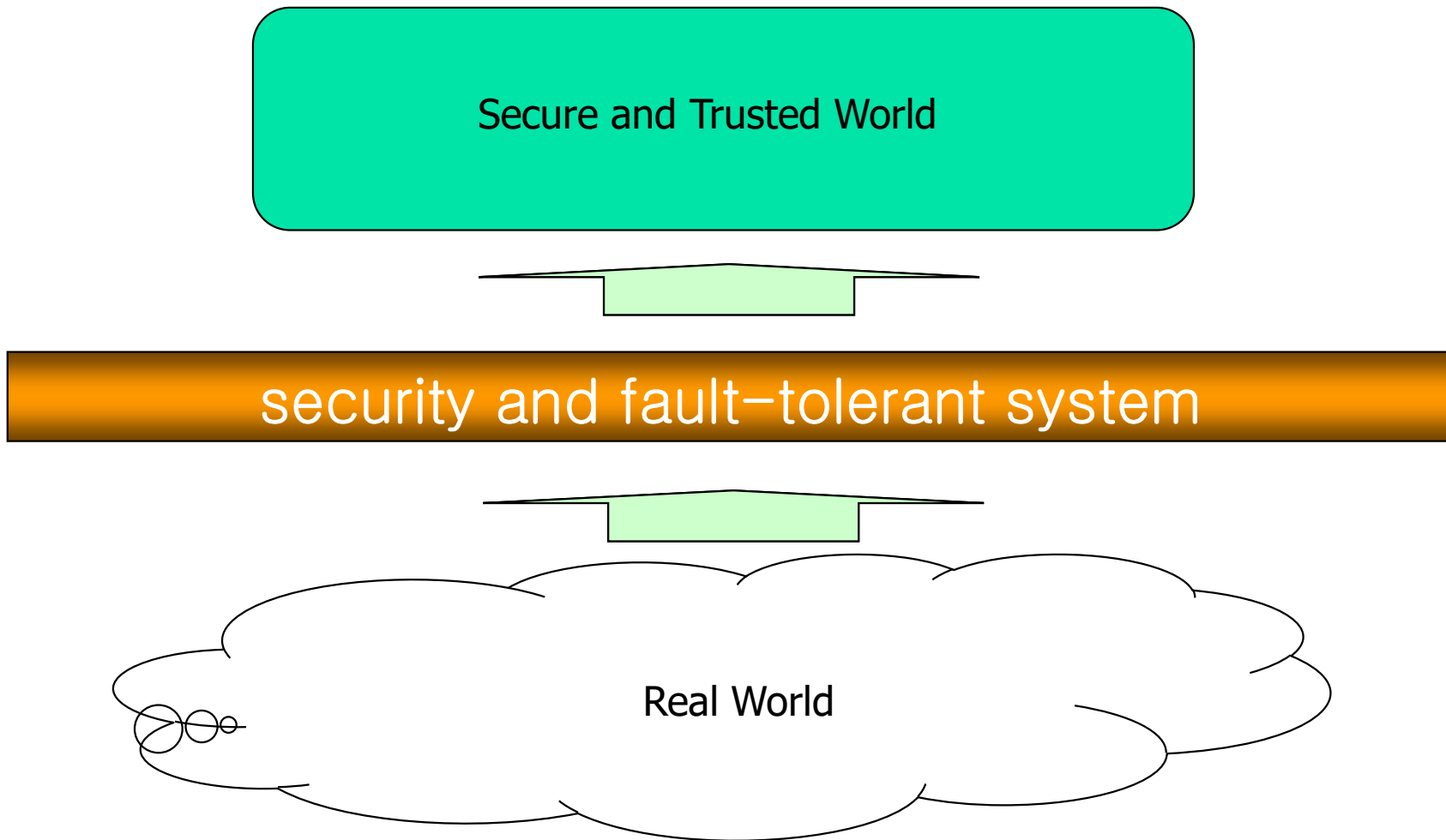
Abstraction (7/9)

■ Data representation



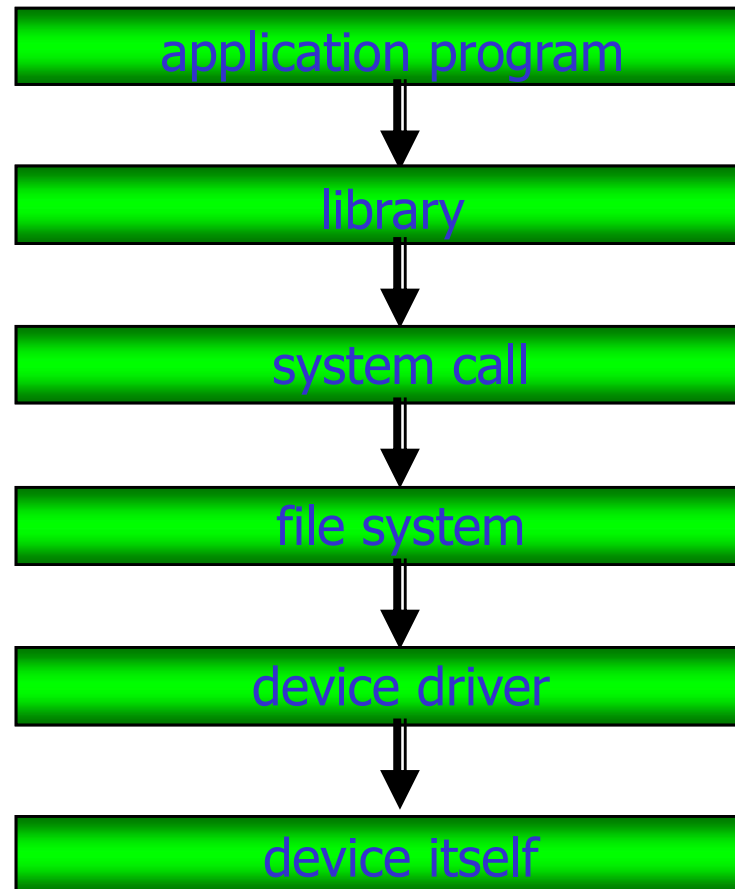
Abstraction (8/9)

- Security and reliability



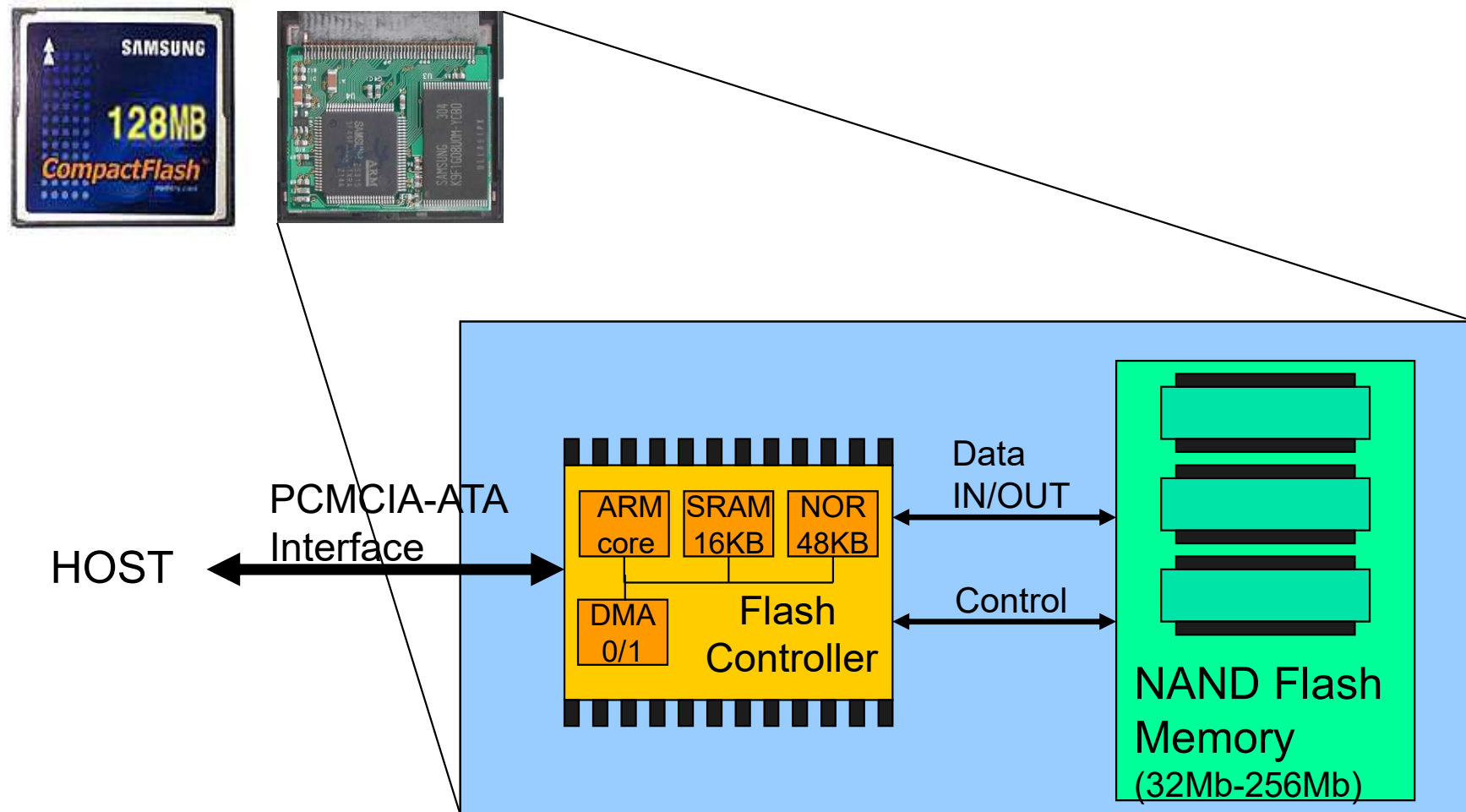
Abstraction (9/9)

- 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: Summarize Chapter 1, “A Tour of Computer Systems” in CSAPP.

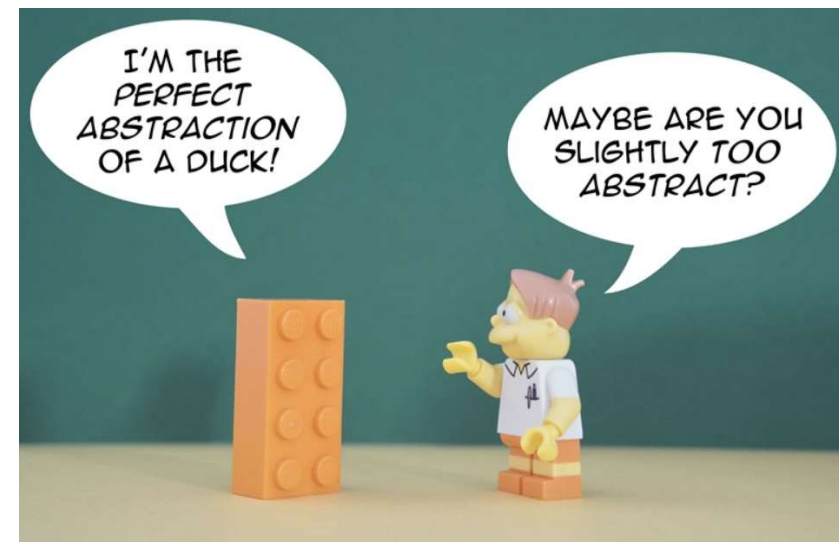
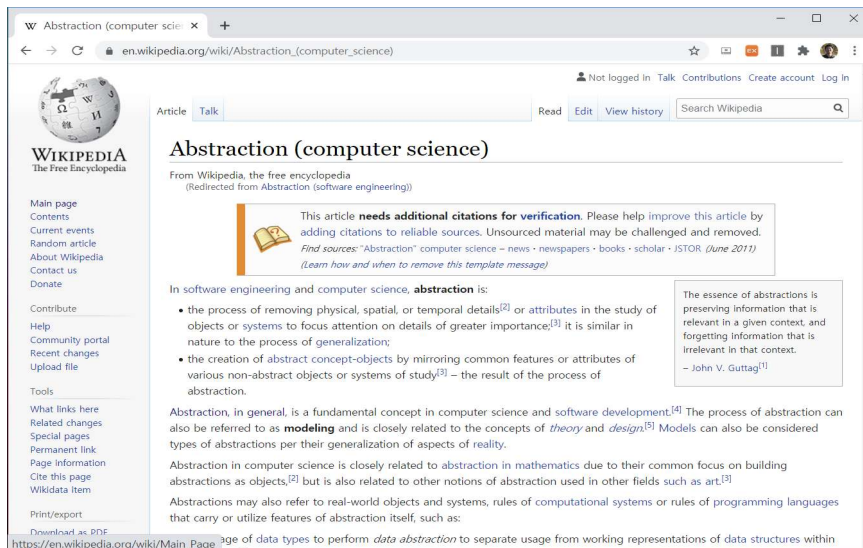
- ✓ Requirement: 1) From the beginning to the Section 1.7 (not include 1.8, 1.9 and 1.10), 2) What is the purpose of studying System Programming?
- ✓ Pages: 1) 1~10 pages, 2) 1~2 pages
- ✓ Deadline: Two weeks later
- ✓ How to submit? Email to choiyg@dankook.ac.kr
- ✓ Caution: Do not copy!!





Quiz for this Lecture

- 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).
- 3. What are the names of Linux command for editor, compiler, assembler, linker and loader (5 names).
- 4. 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).



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



- 본 교재는 2025년도 과학기술정보통신부 및 정보통신기획평가원의 ‘SW중심대학사업’ 지원을 받아 제작 되었습니다.
- 본 결과물의 내용을 전재할 수 없으며, 인용(재사용)할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원이 지원한 ‘SW중심대학’의 결과물이라는 출처를 밝혀야 합니다.

IITP 정보통신기획평가원
디지털인재양성단 SW인재팀

