Lecture Note 1.
What is System Programming

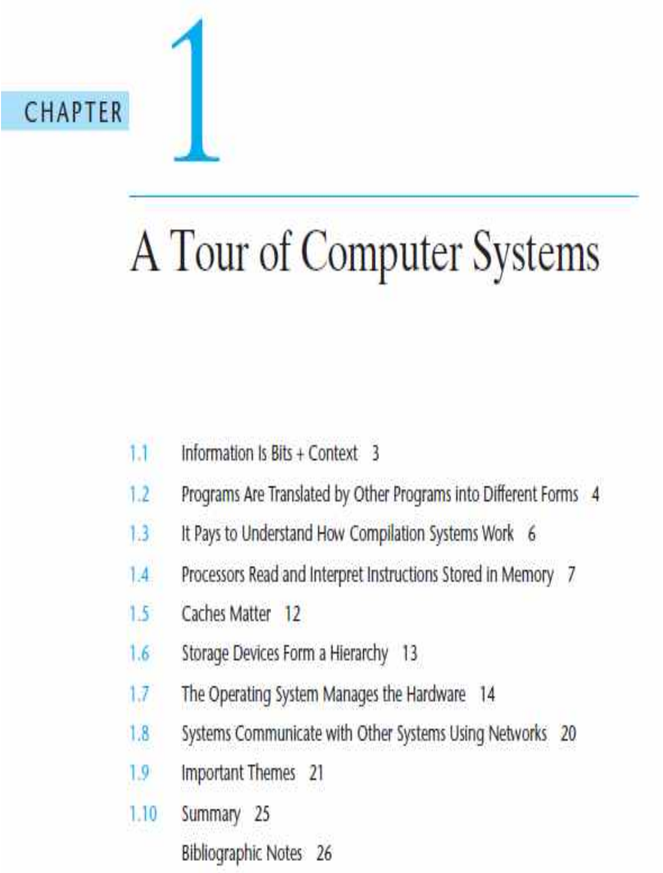
August 26, 2020

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

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

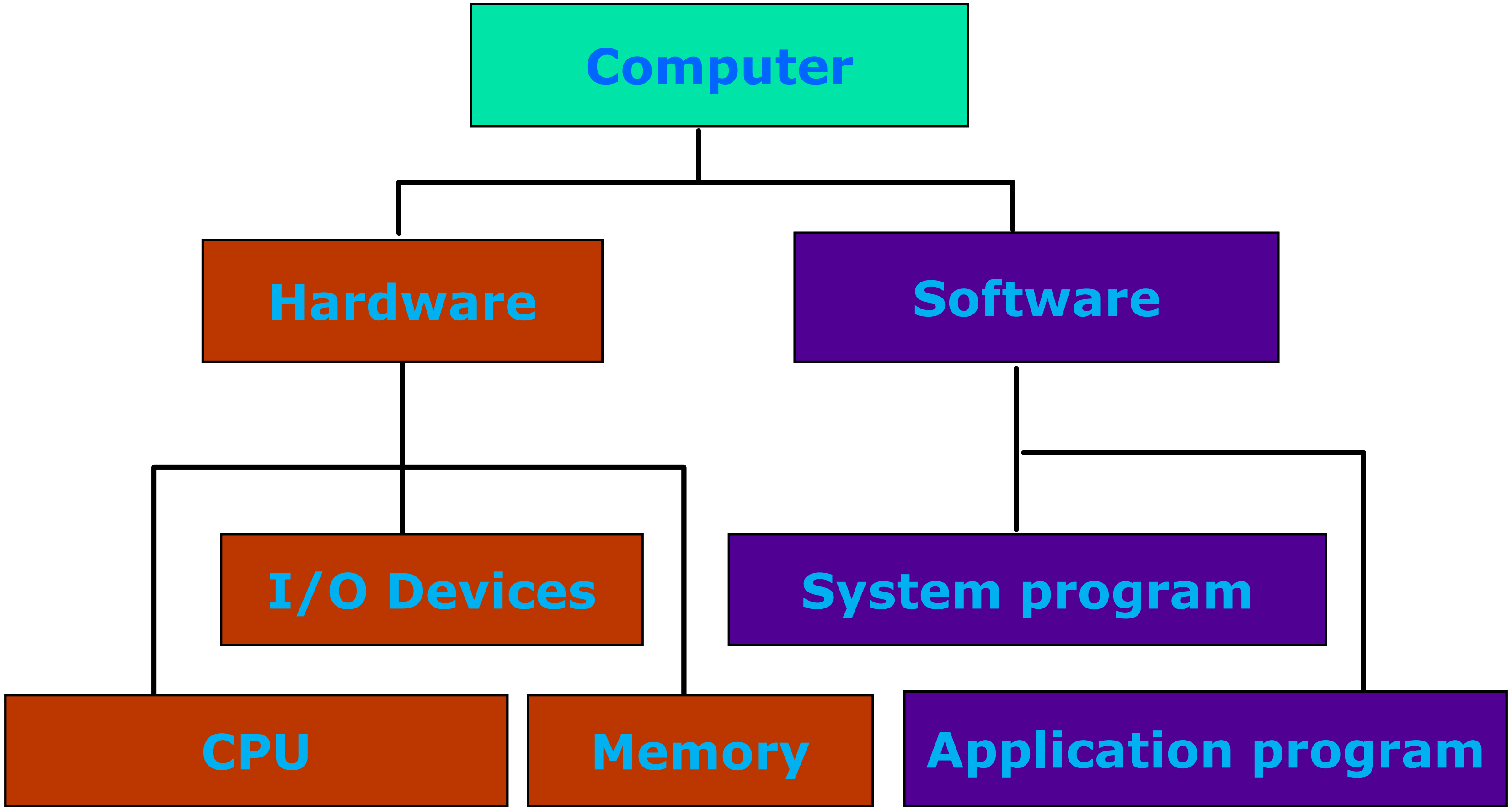
A Tour of Computer Systems

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



Computer

Hardware

Software

I/O Devices

System program

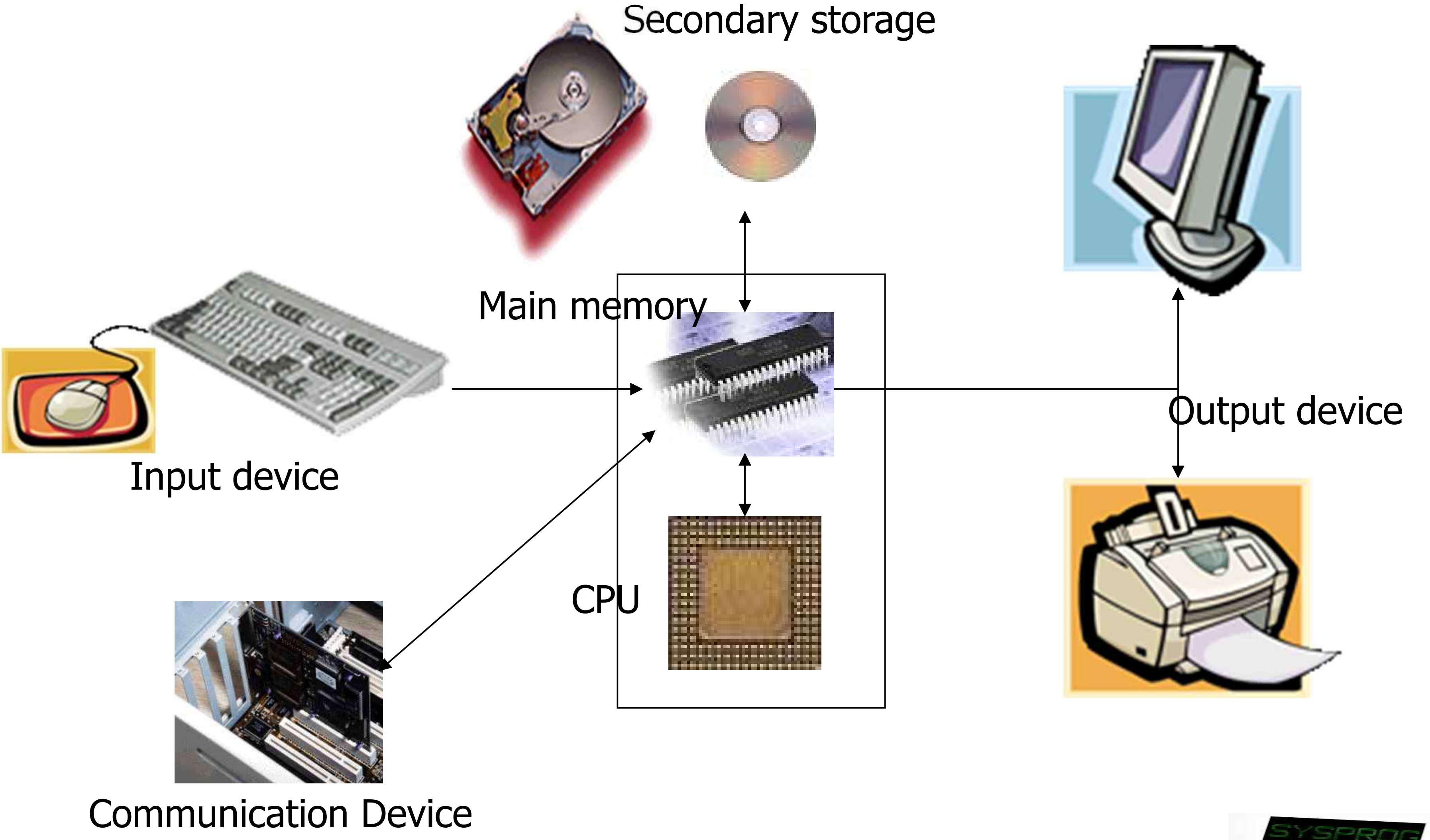
CPU

Memory

Application program

Definition of System Program (2/8)

- Hardware components: PC



Secondary storage

Main memory

Input device

Output device

CPU

Communication 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)

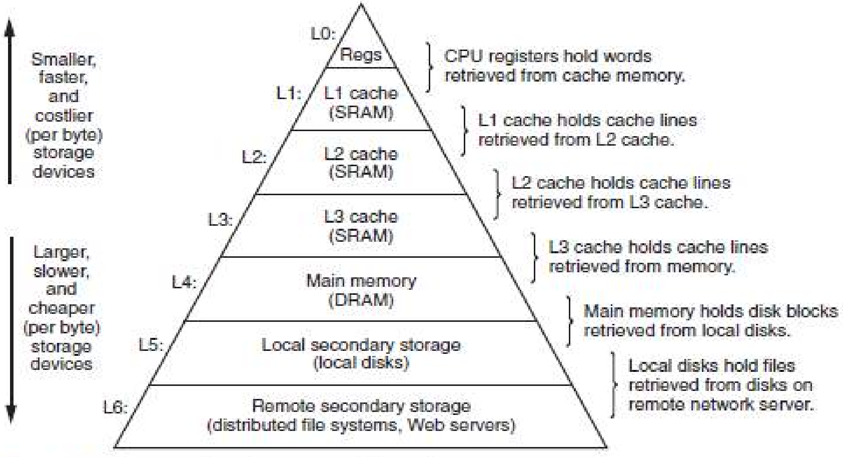
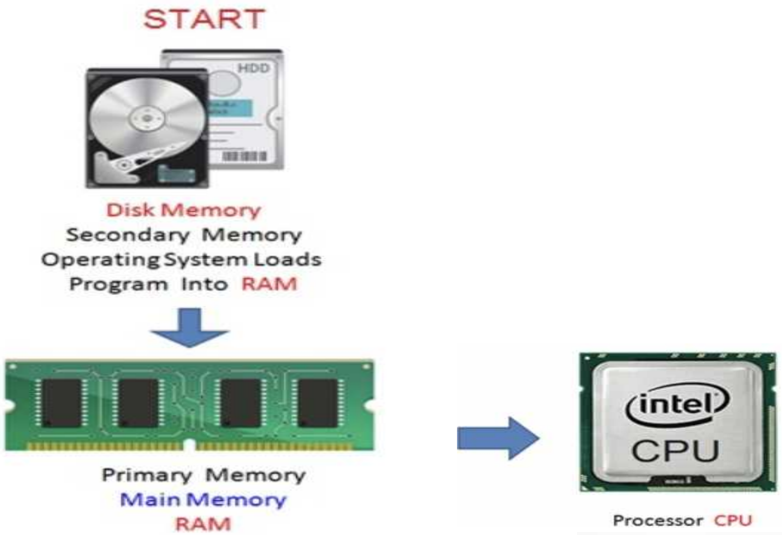

Smaller, faster, and costlier (per byte) storage devices
Larger, slower, and cheaper (per byte) storage devices
L0:
Regs
L1:
L1 cache (SRAM)
L2:
L2 cache (SRAM)
L3:
L3 cache (SRAM)
L4:
Main memory (DRAM)
L5:
Local secondary storage (local disks)
L6:
Remote secondary storage (distributed file systems,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

Figure 1.9 An example of a memory hierarchy.



(Source: CSAPP)

(Source: Google Image)

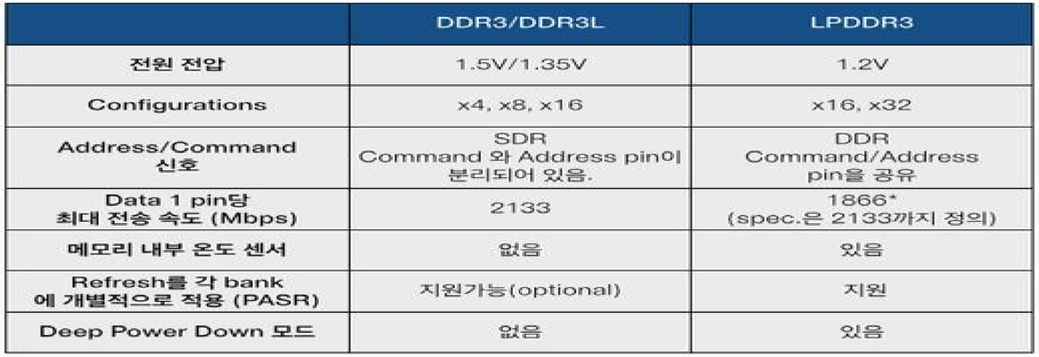
Definition of System Program (4/8)

- 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: 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?

Definition of System Program (7/8)

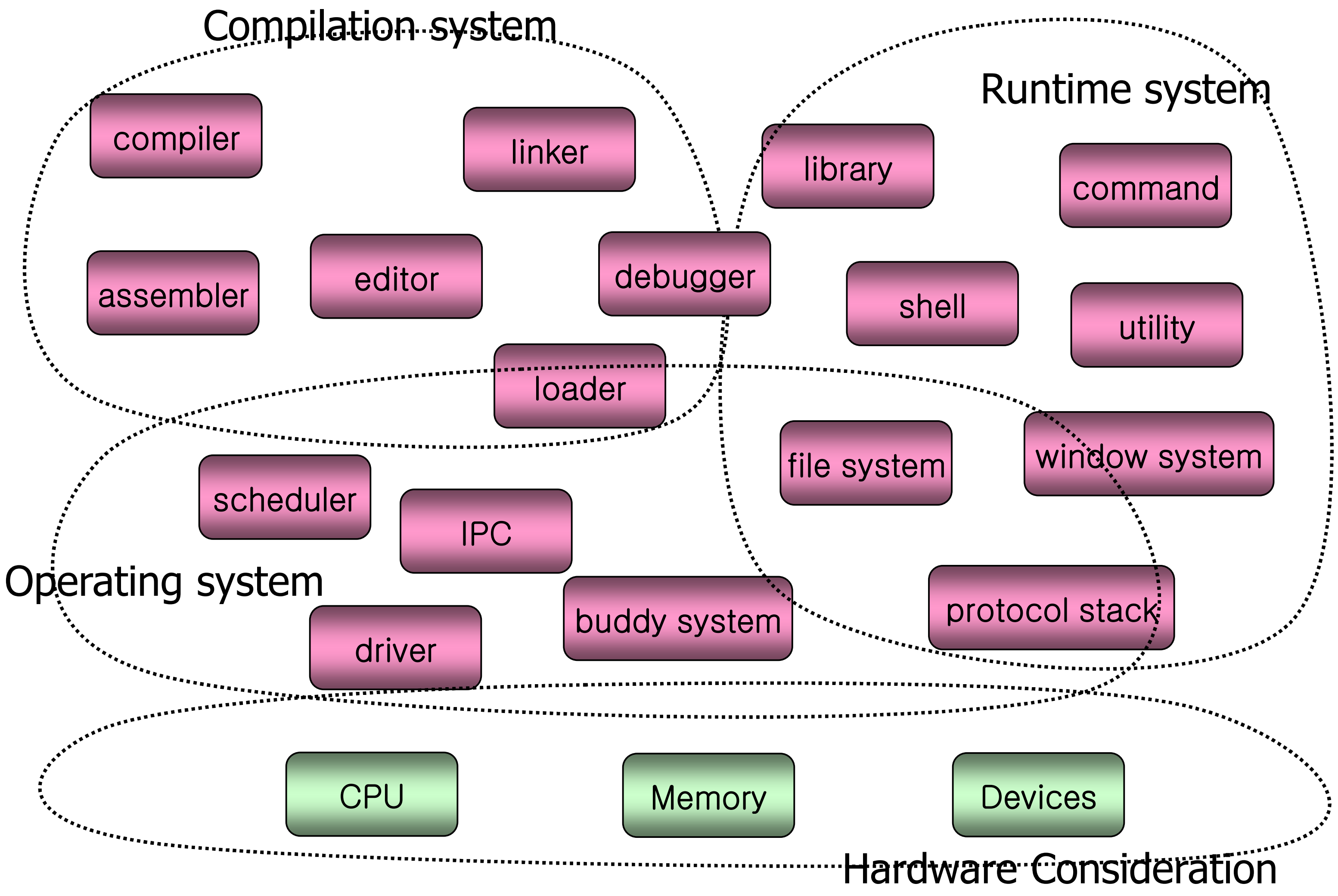
- Software components: System program
  - How to run a program on CPU?
    - object, binary, compiler, assembler, loader, ...
  - What is the role of printf()?
    - library, linker, ...
  - How the string is displayed on Monitor?
    - device driver, file system, ...
  - How a program can be executed 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
  - ✓ 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



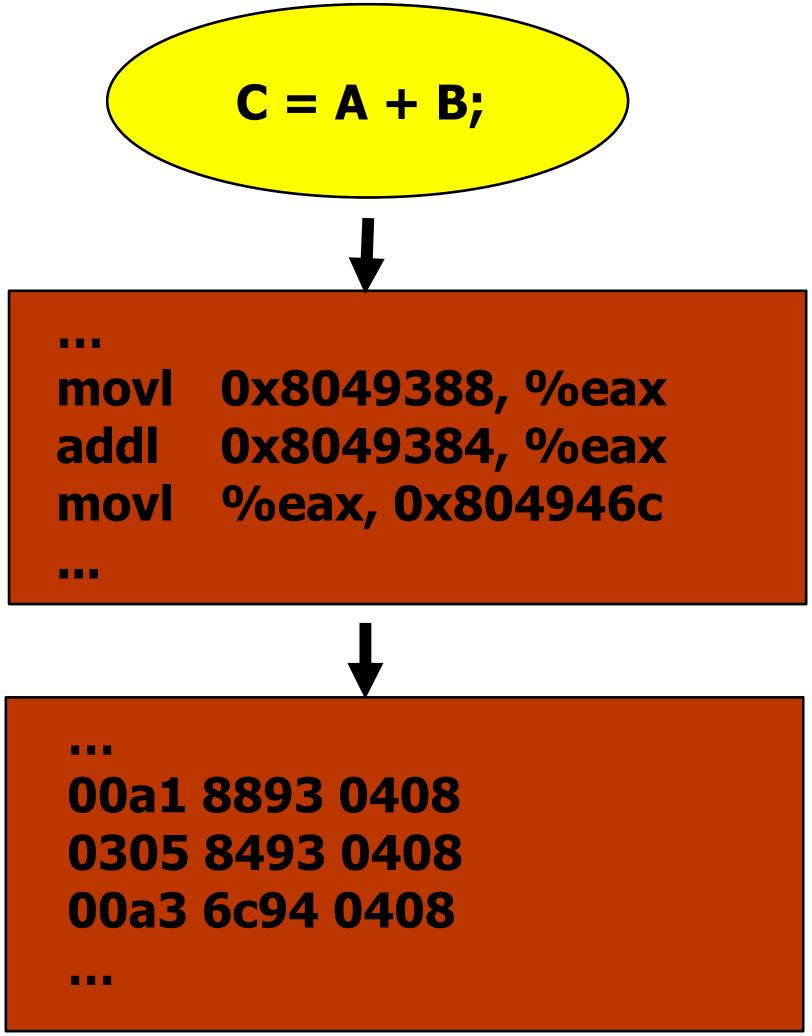
Compilation System (1/5)

- Concept: Language Hierarchy

High-level Language

Assembly Language

Machine Language
(Binary code)



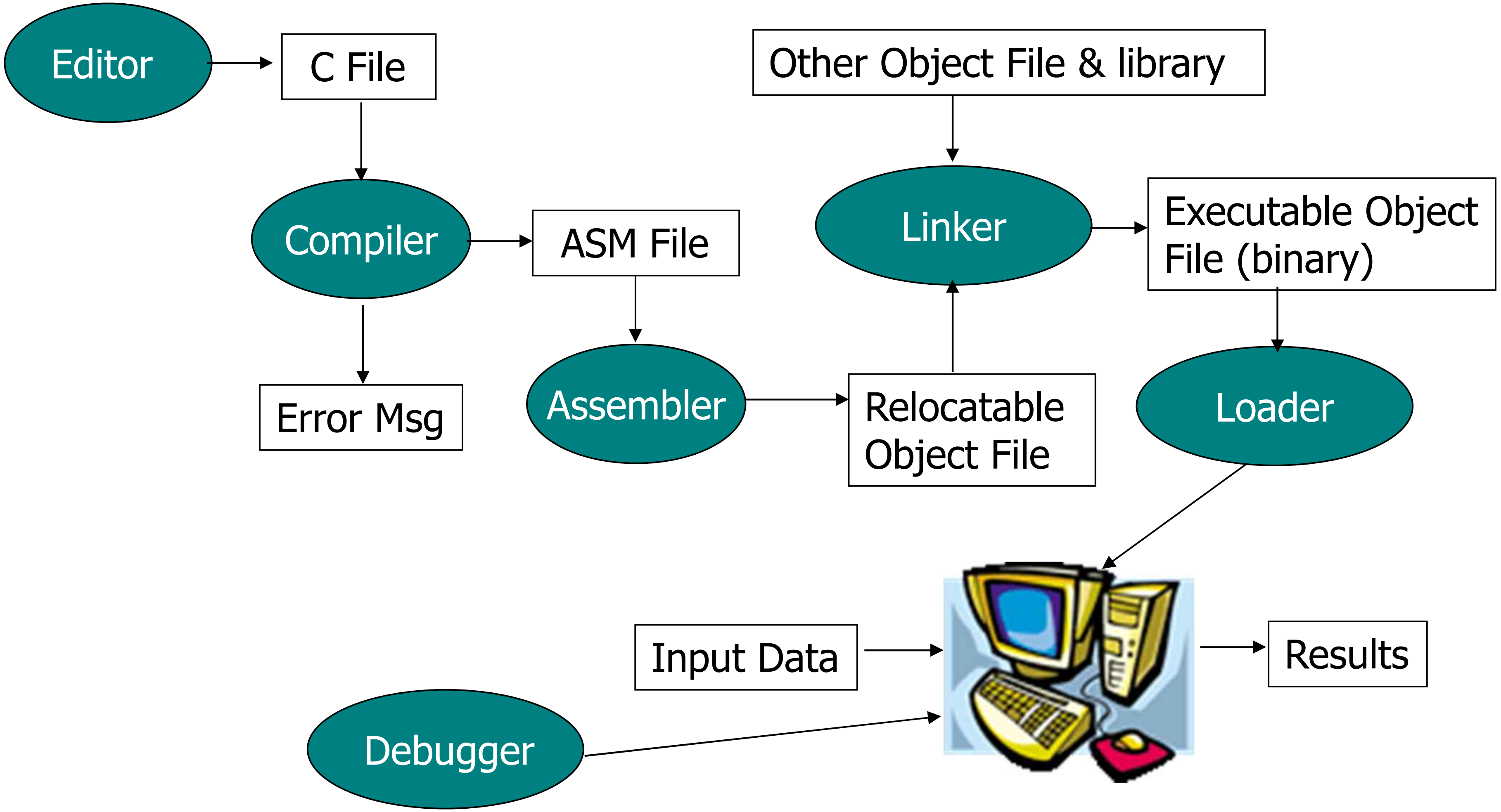
```
C = A + B;
```

```
...
movl   0x8049388, %eax
addl   0x8049384, %eax
movl   %eax, 0x804946c
...
```

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

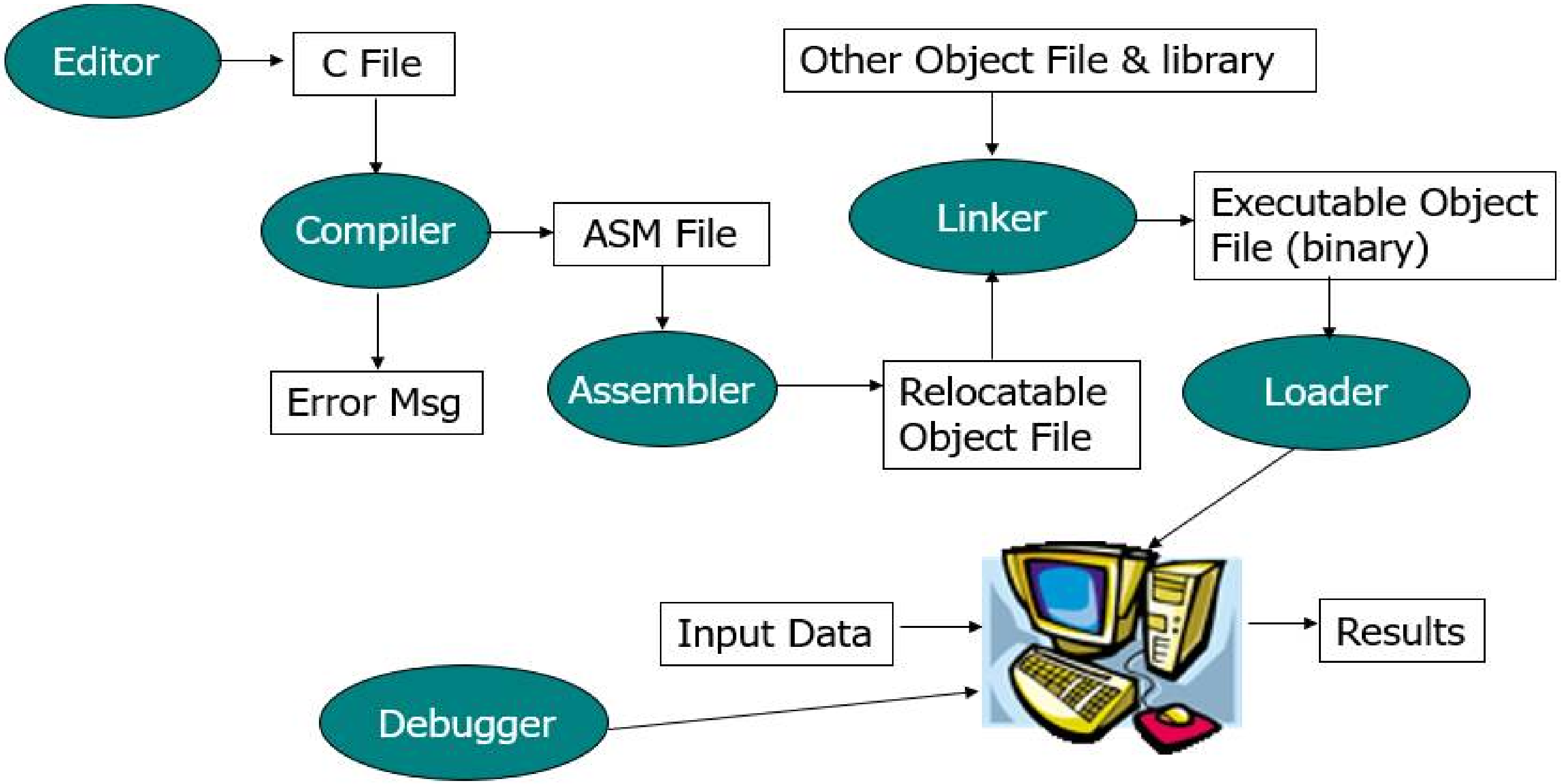
Compilation System (2/5)

- Overall structure and 6 key components



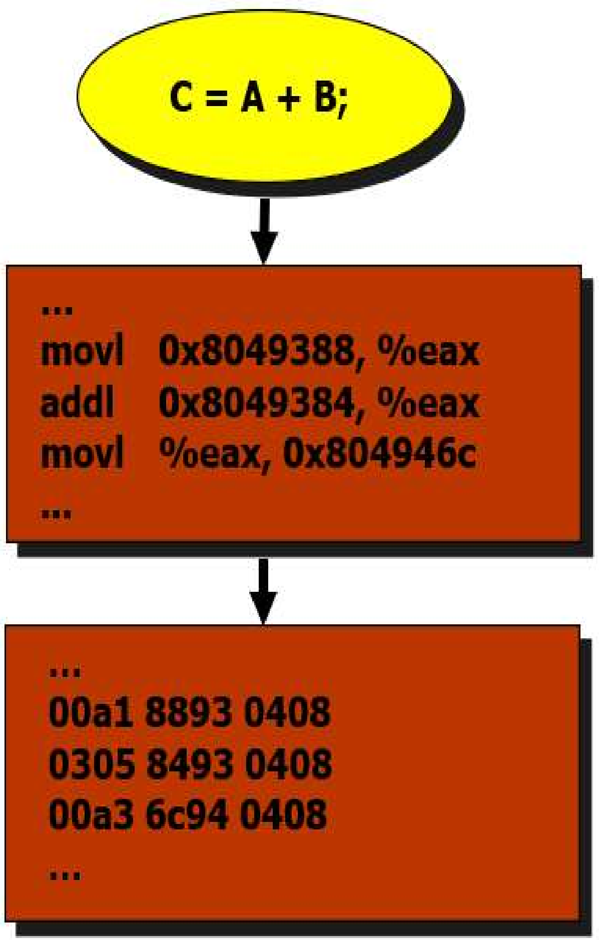
Quiz for 1st-Week 2nd-Lesson

- Quiz
  - ✓ Explain why loader is required in a computer system. (hint: using the difference between Disk and DRAM)
  - ✓ Due: until 6 PM Friday of this week(4th, September)



Compilation System (3/5)

- Relation between Language Hierarchy and Overall Structure

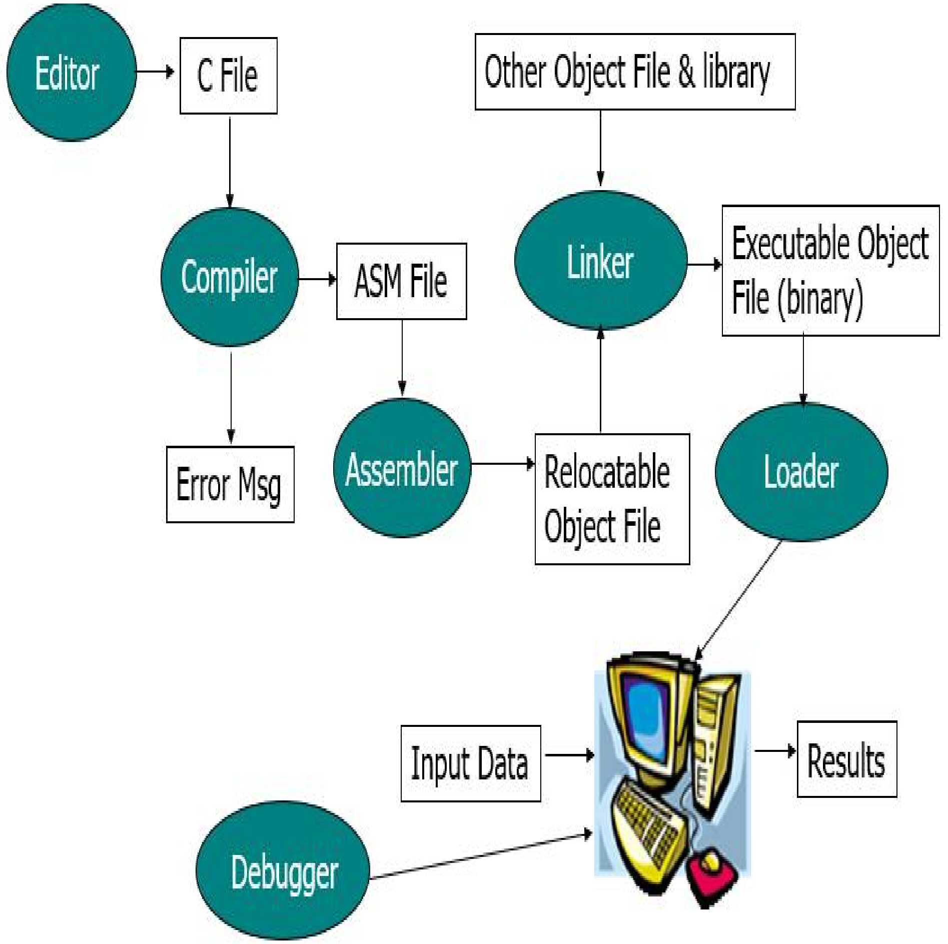


```
C = A + B;
```

```
...
movl   0x8049388, %eax
addl   0x8049384, %eax
movl   %eax, 0x804946c
...
```

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

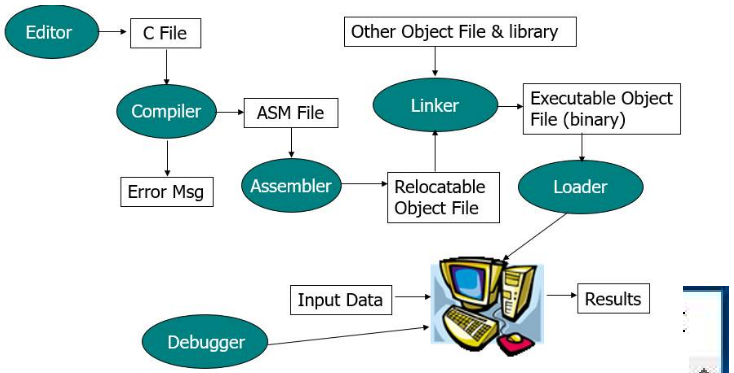
(Language hierarchy)

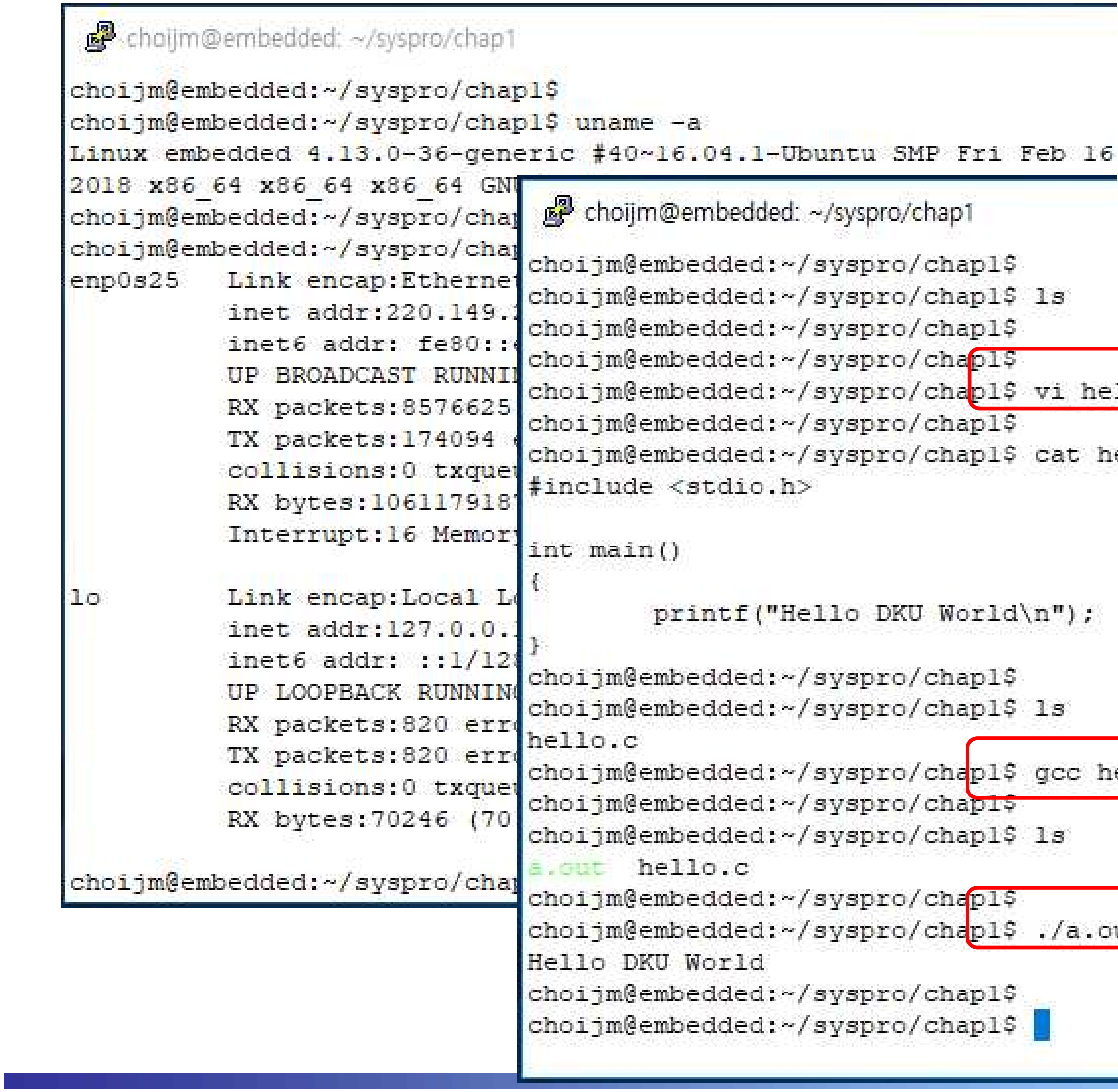


(Compilation system)

Compilation System (4/5)

- Example in Linux





choijm@embedded: ~/syspro/chap1

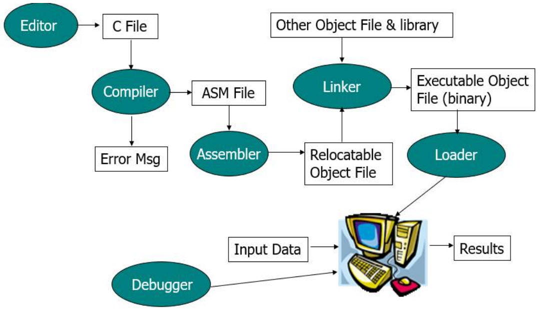
```
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ uname -a
Linux embedded 4.13.0-36-generic #40~16.04.1-Ubuntu SMP Fri Feb 16
2018 x86_64 x86_64 x86_64 GNU
choijm@embedded:~/syspro/cha
choijm@embedded:~/syspro/cha
enp0s25   Link encap:Etherne
          inet addr:220.149.
          inet6 addr: fe80::
          UP BROADCAST RUNNI
          RX packets:8576625
          TX packets:174094
          collisions:0 txque
          RX bytes:106117918
          Interrupt:16 Memory

lo        Link encap:Local L
          inet addr:127.0.0.
          inet6 addr: ::1/12
          UP LOOPBACK RUNNIN
          RX packets:820 err
          TX packets:820 err
          collisions:0 txque
          RX bytes:70246 (70

choijm@embedded:~/syspro/cha
```

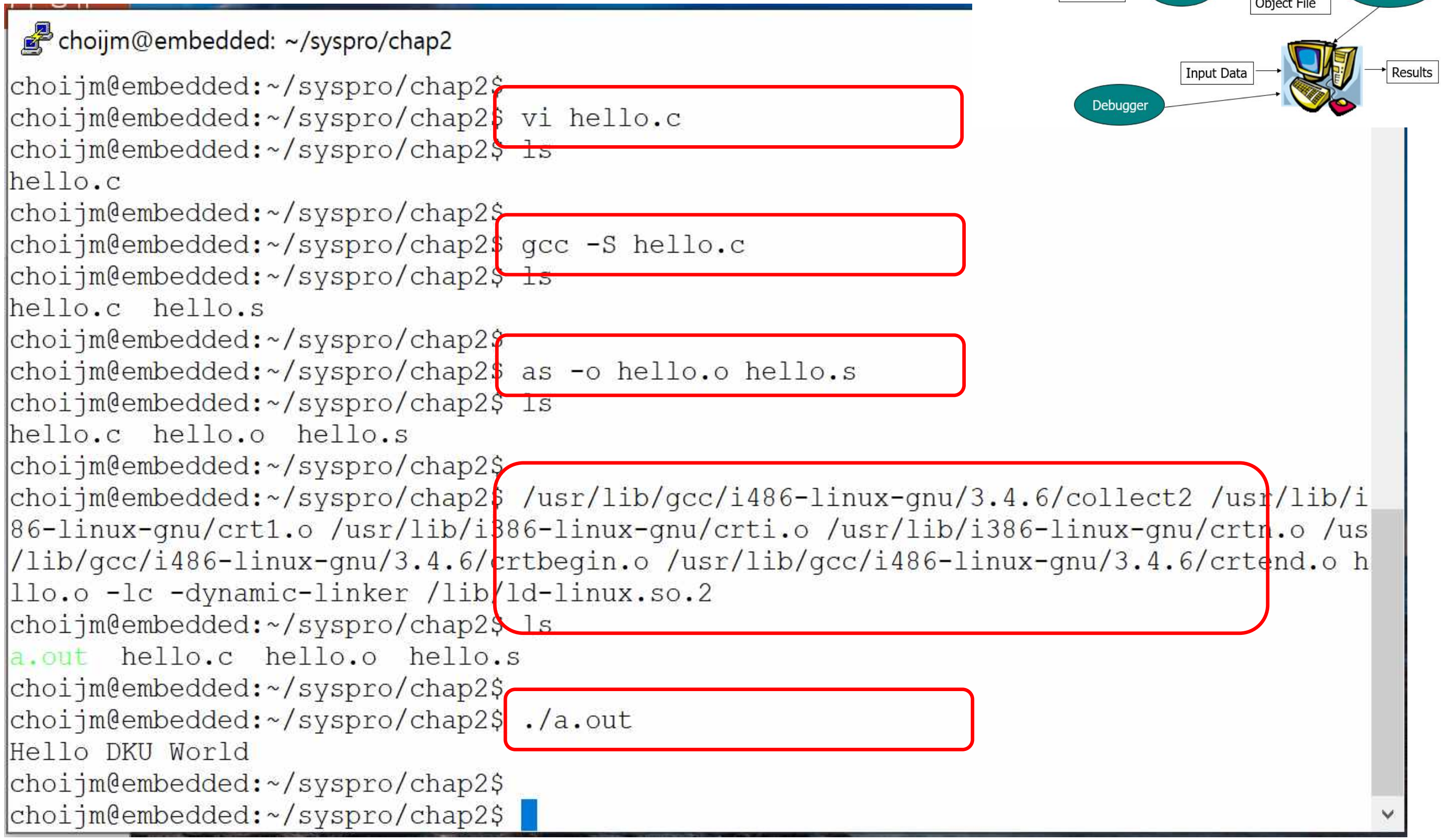
choijm@embedded: ~/syspro/chap1

```
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ ls
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ vi hello.c
choijm@embedded:~/syspro/chap1$
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





choijm@embedded: ~/syspro/chap2

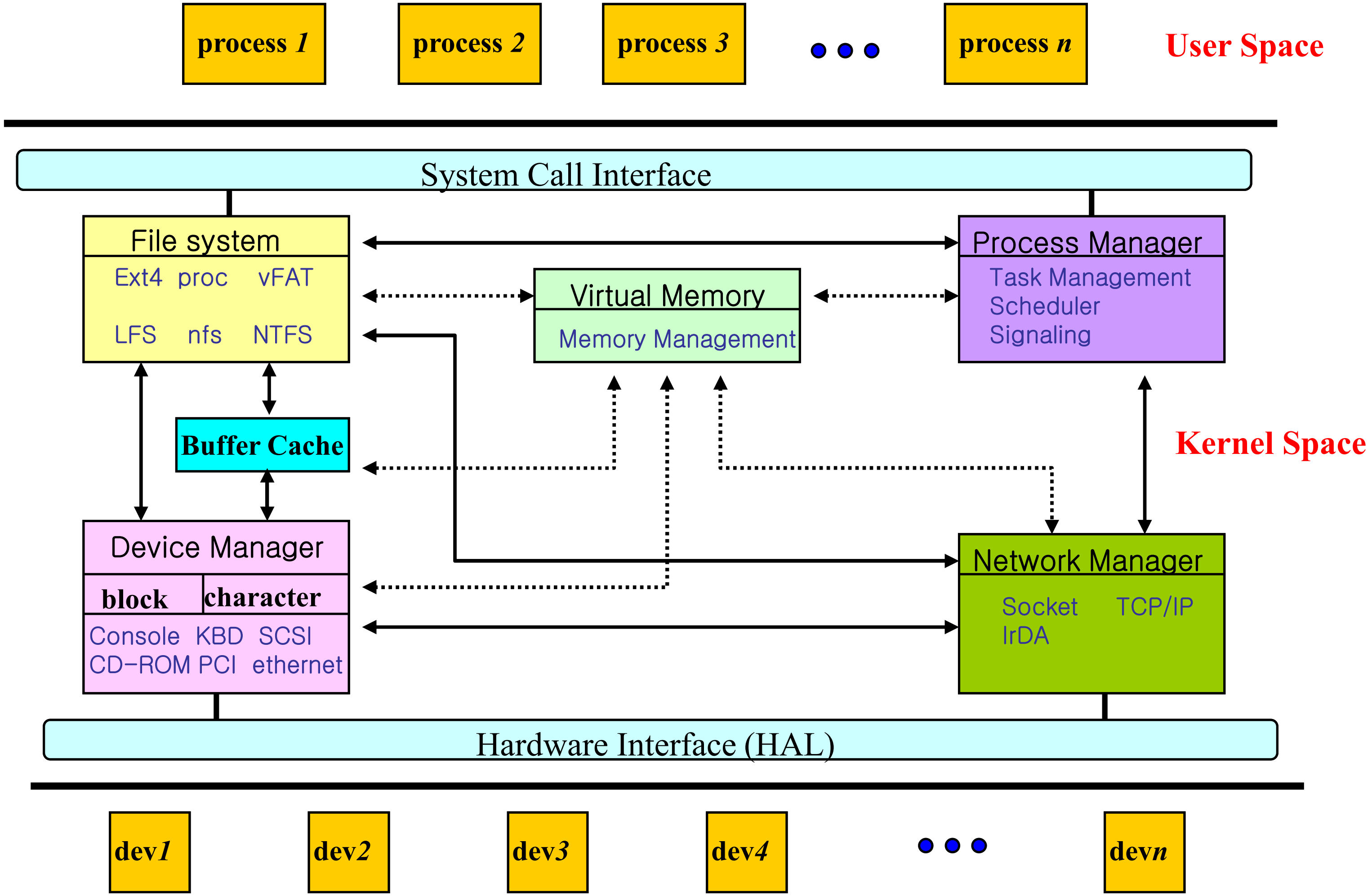
```
choijm@embedded:~/syspro/chap2$
choijm@embedded:~/syspro/chap2$ vi hello.c
choijm@embedded:~/syspro/chap2$ ls
hello.c
choijm@embedded:~/syspro/chap2$
choijm@embedded:~/syspro/chap2$ gcc -S hello.c
choijm@embedded:~/syspro/chap2$ ls
hello.c  hello.s
choijm@embedded:~/syspro/chap2$
choijm@embedded:~/syspro/chap2$ as -o hello.o hello.s
choijm@embedded:~/syspro/chap2$ ls
hello.c  hello.o  hello.s
choijm@embedded:~/syspro/chap2$
choijm@embedded:~/syspro/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:~/syspro/chap2$ ls
a.out  hello.c  hello.o  hello.s
choijm@embedded:~/syspro/chap2$
choijm@embedded:~/syspro/chap2$ ./a.out
Hello DKU World
choijm@embedded:~/syspro/chap2$
choijm@embedded:~/syspro/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)

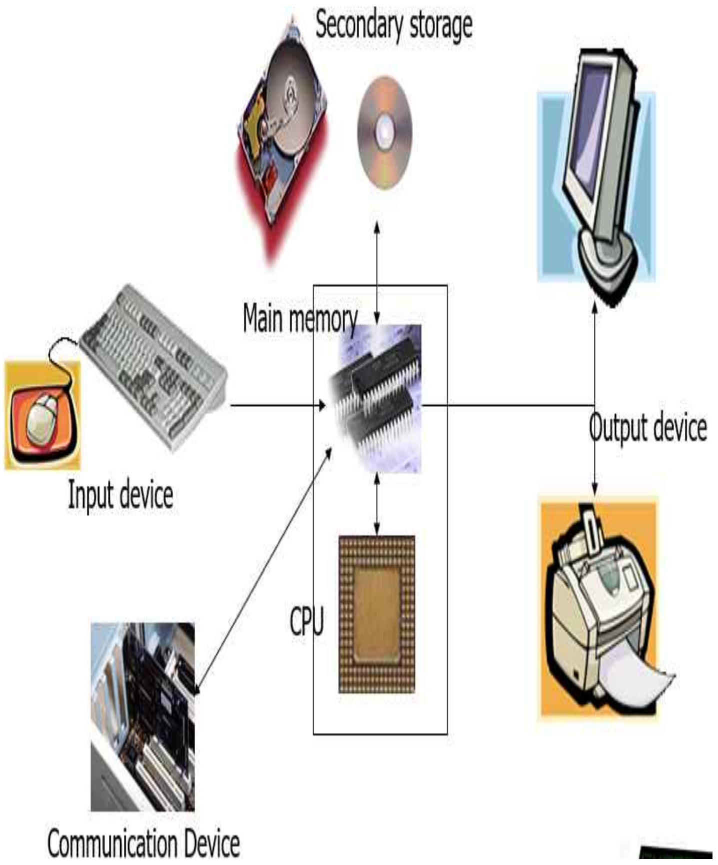
- Overall structure and 7 key components



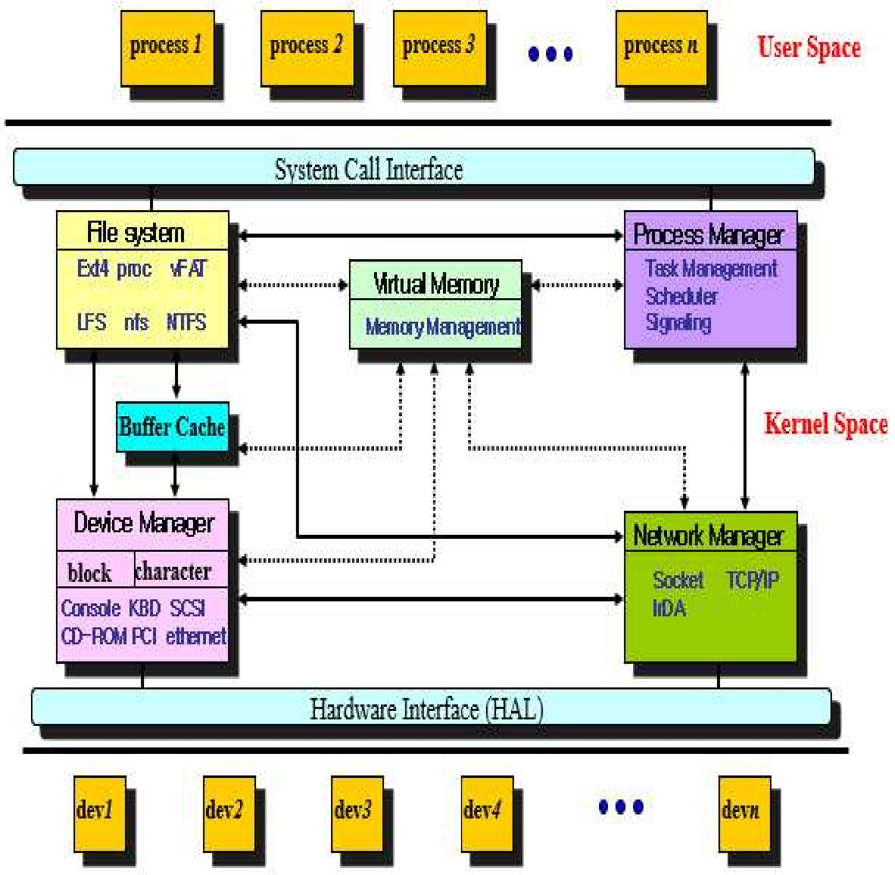
(Source: Linux Kernel Internals)

Operating System (2/15)

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



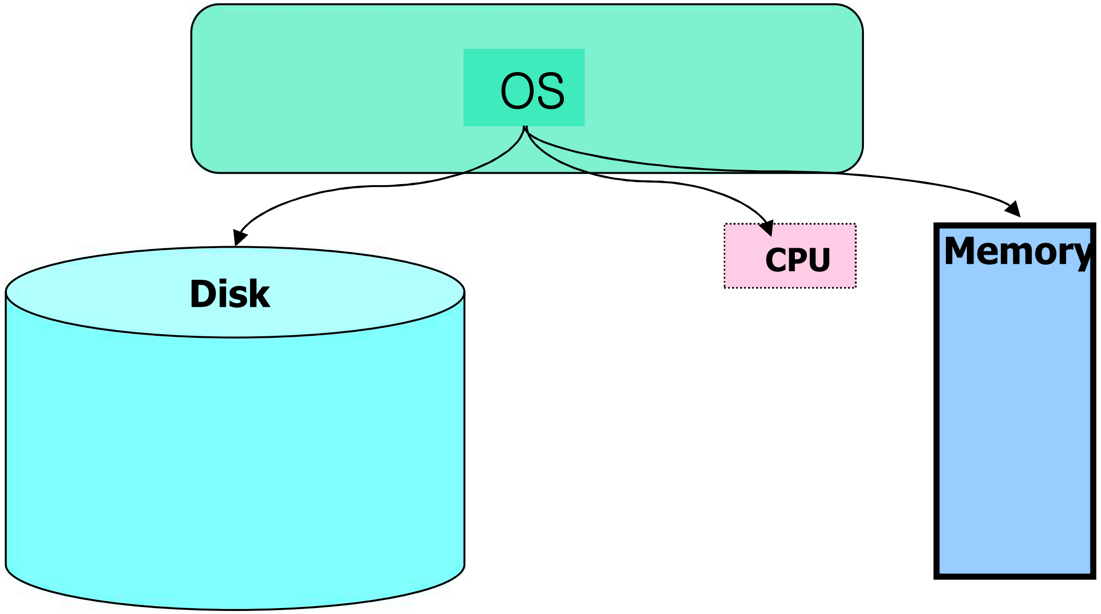
(Physical resources)



(Logical resources)

Operating System (3/15)

- Behaviors: 1) initial state

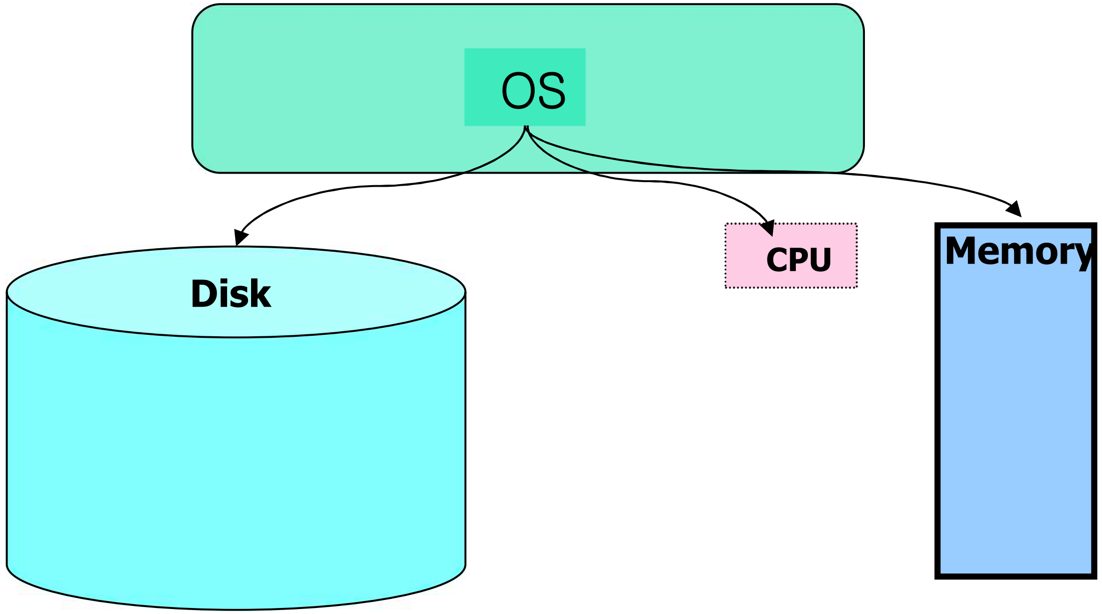


Operating System (4/15)

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

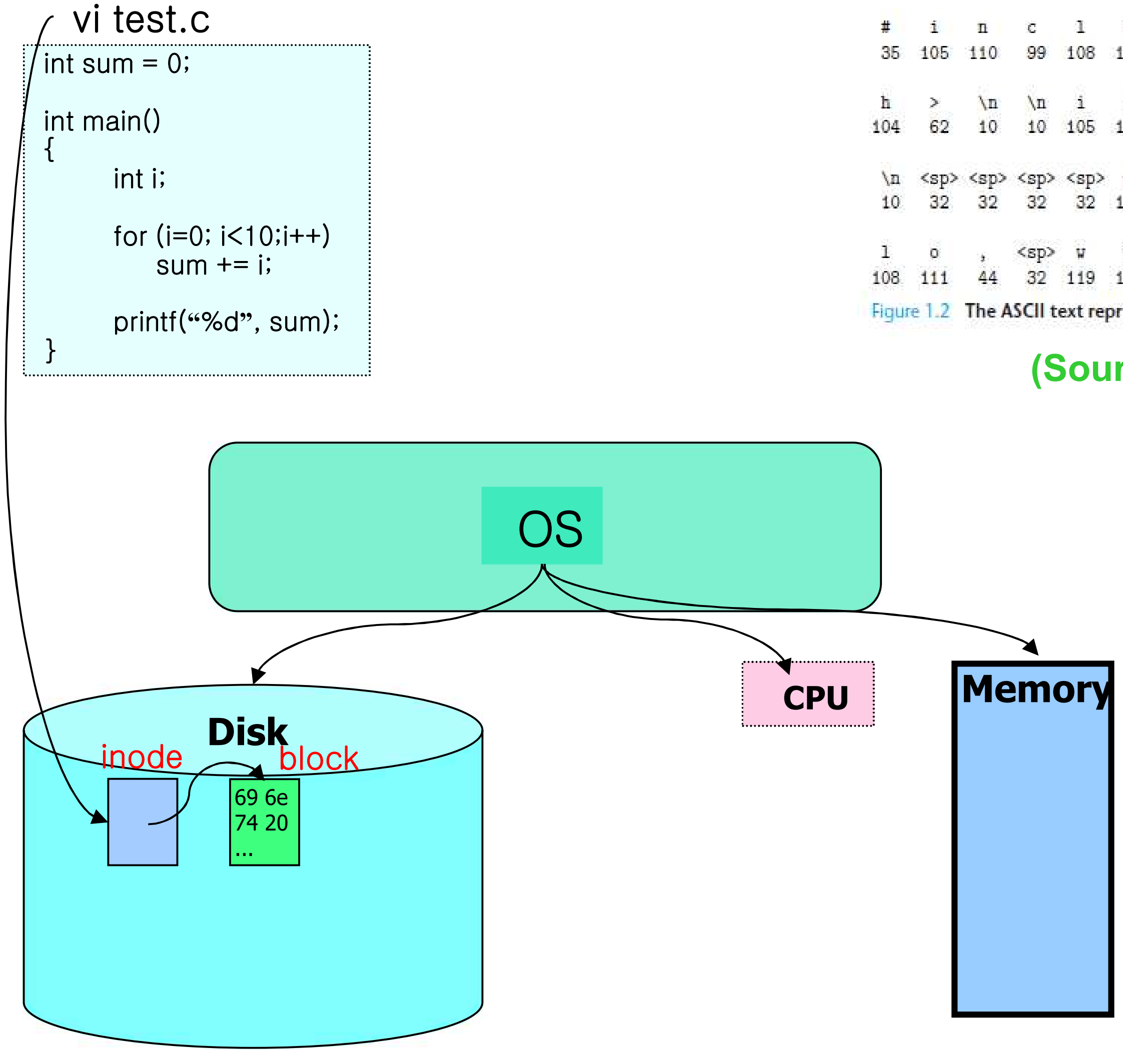
vi test.c

```
int sum = 0;

int main()
{
        int i;

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

        printf(“%d”, sum);
}
```



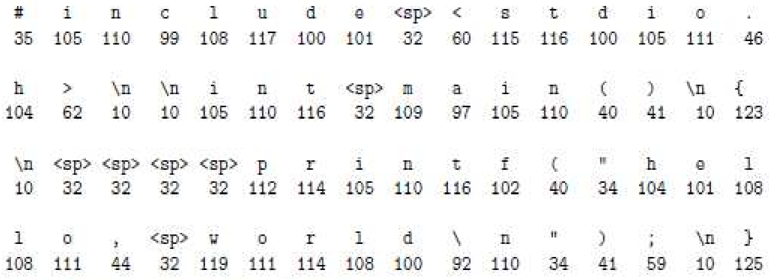
Operating System (5/15)

- 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);
}
```



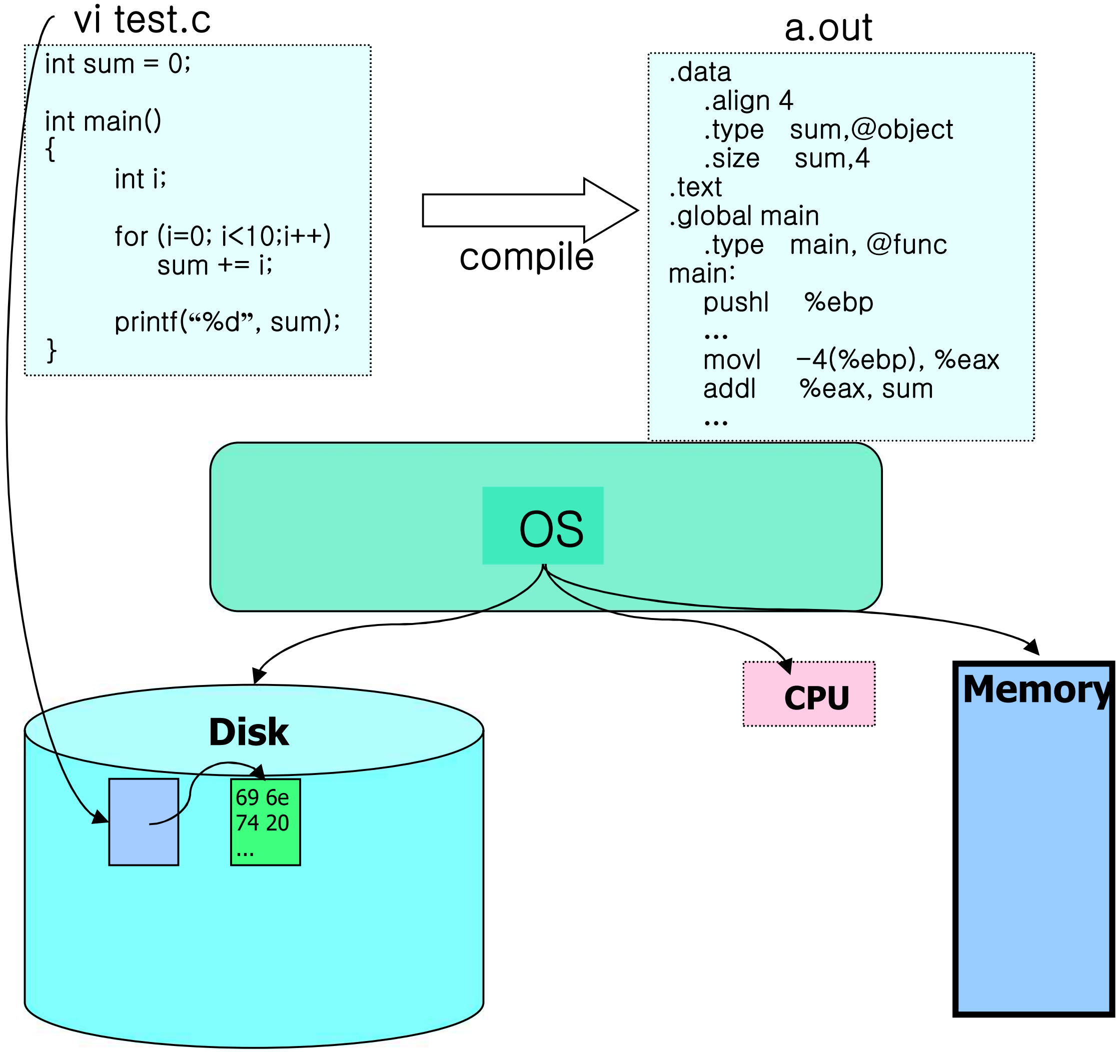
#	i	n	c	l	u	d	e	<sp>	<	s	t	d	i	o	.
35	105	110	99	108	117	100	101	32	60	115	116	100	105	111	46
h	>	\n	\n	i	n	t	<sp>	m	a	i	n	(	)	\n	{
104	62	10	10	105	110	116	32	109	97	105	110	40	41	10	123
\n	<sp>	<sp>	<sp>	<sp>	p	r	i	n	t	f	(	"	h	e	l
10	32	32	32	32	112	114	105	110	116	102	40	34	104	101	108
l	o	,	<sp>	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)

Operating System (6/15)

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



vi test.c

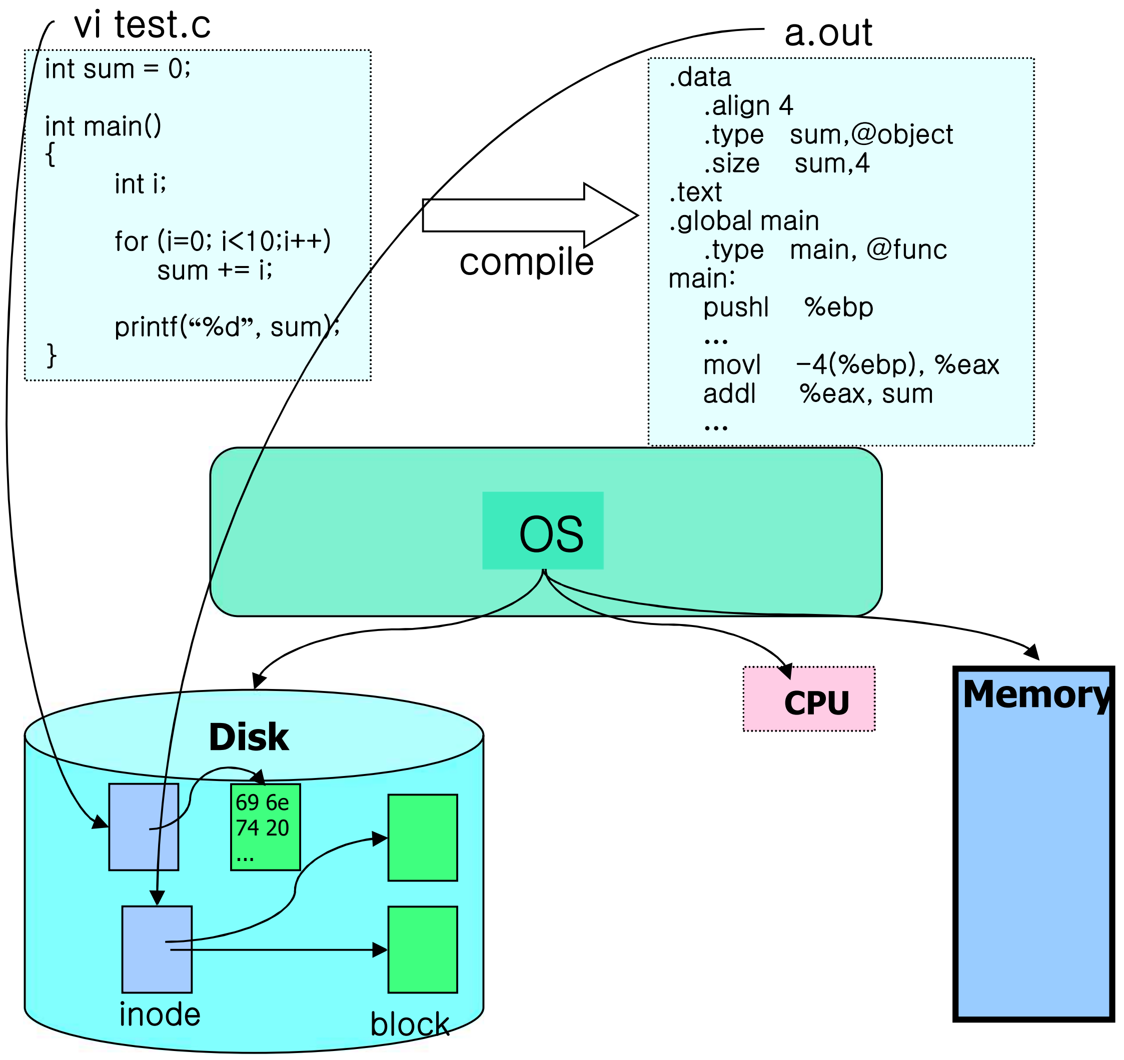
```
int sum = 0;

int main()
{
        int i;

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

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

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
    …
```

Operating System (7/15)

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



vi test.c

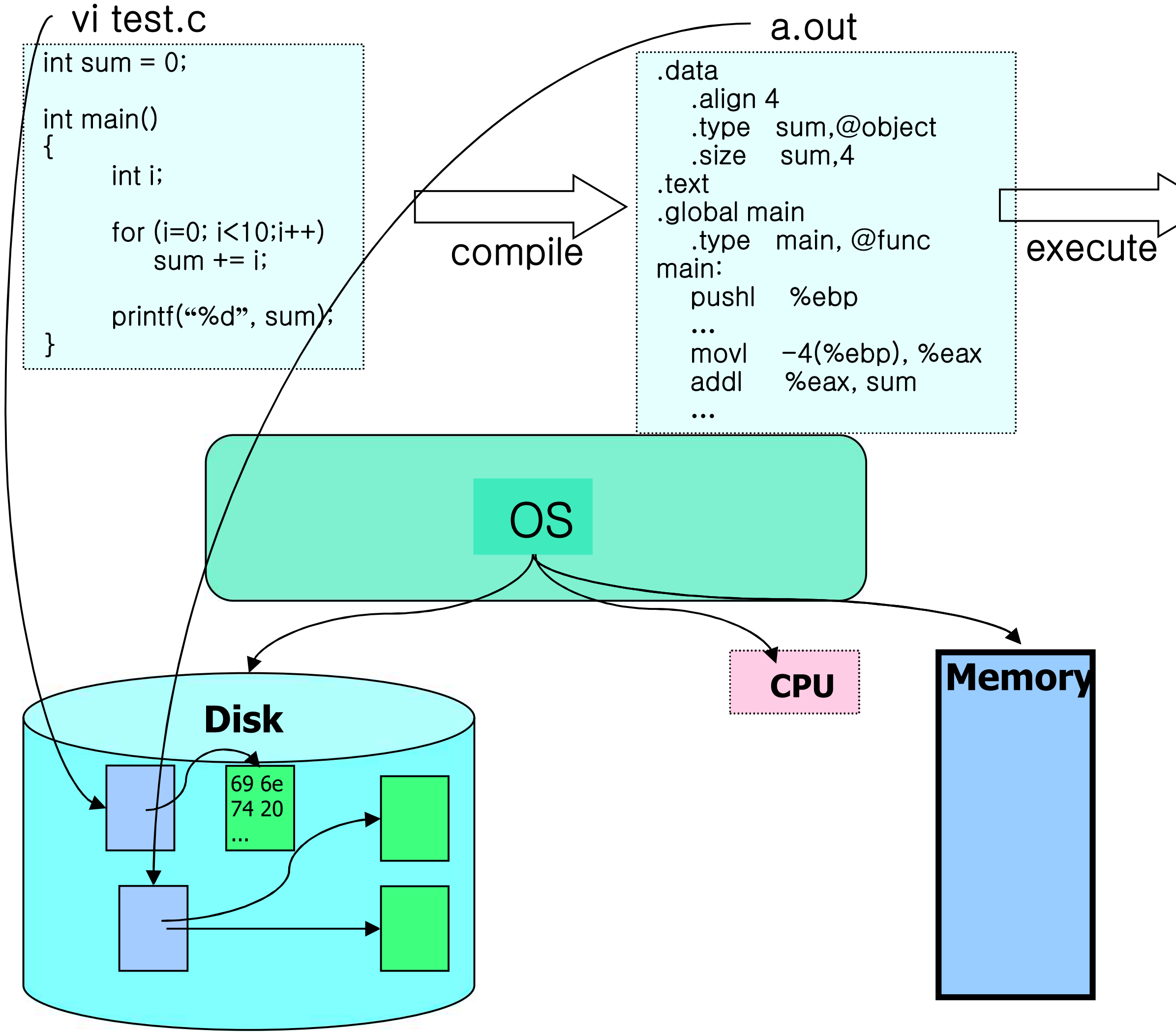
```
int sum = 0;

int main()
{
        int i;

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

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

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
    …
```

Operating System (8/15)

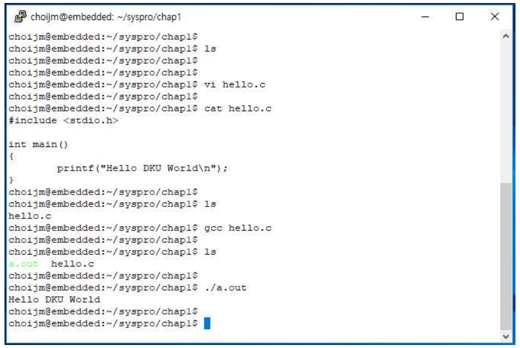
- Behaviors: 4) execute the a.out (user's viewpoint)



```
int sum = 0;

int main()
{
        int i;

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

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

```
.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
    ...
```

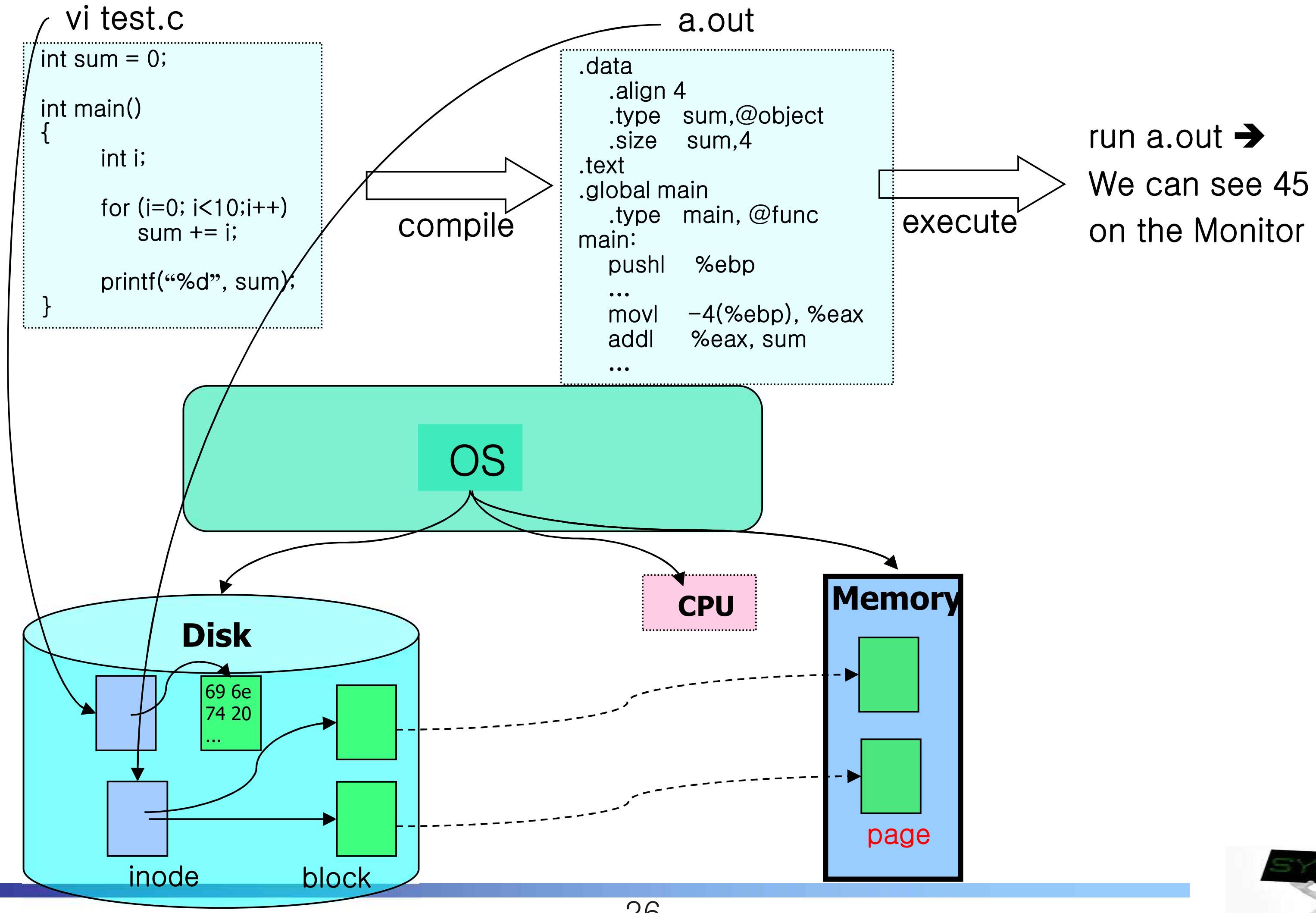
run a.out ➔
We can see 45
on the Monitor



```
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ ls
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ vi hello.c
choijm@embedded:~/syspro/chap1$
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$
```

Operating System (9/13)

- 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);
}
```

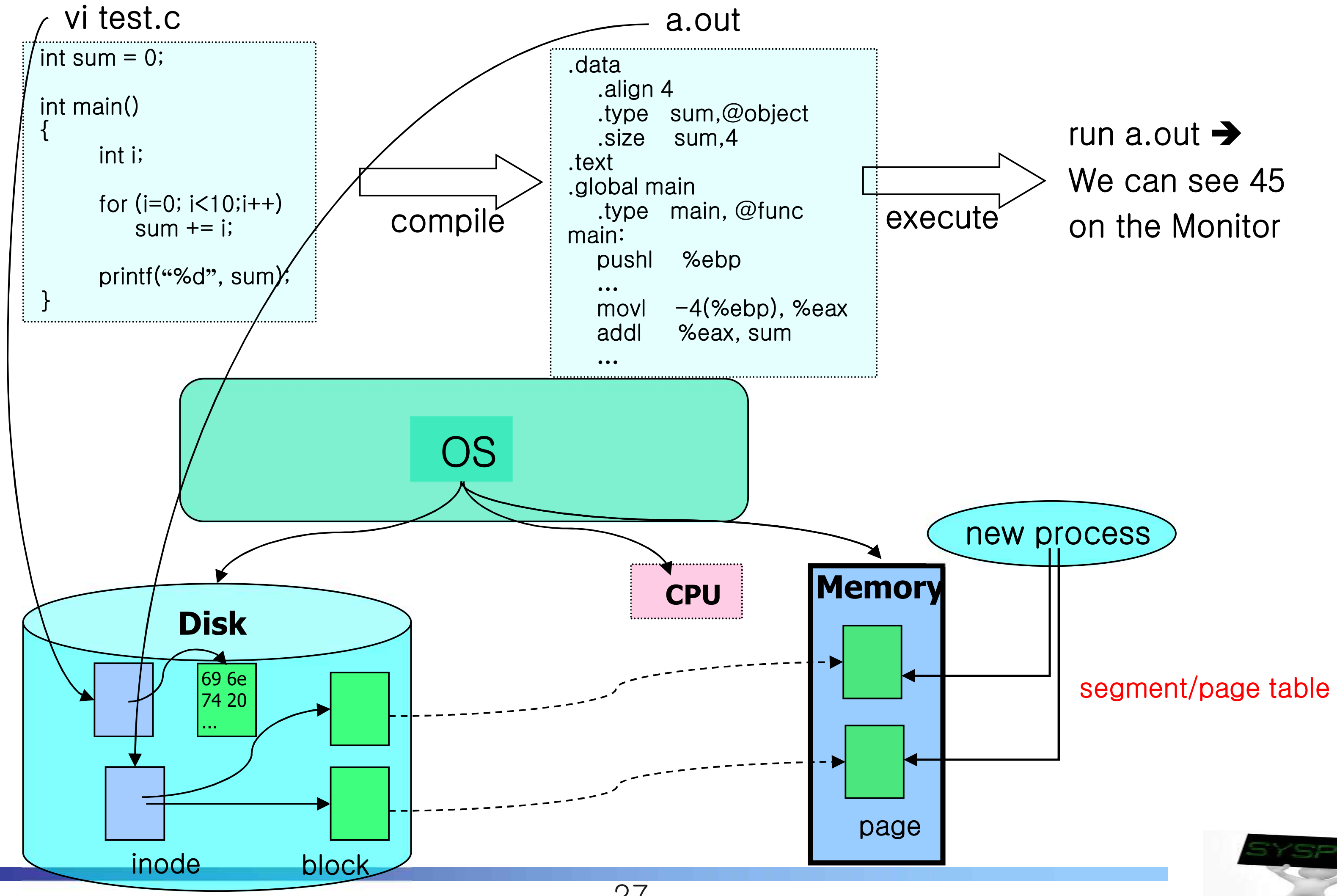
a.out

```
.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
    …
```

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

Operating System (10/13)

- 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);
}
```

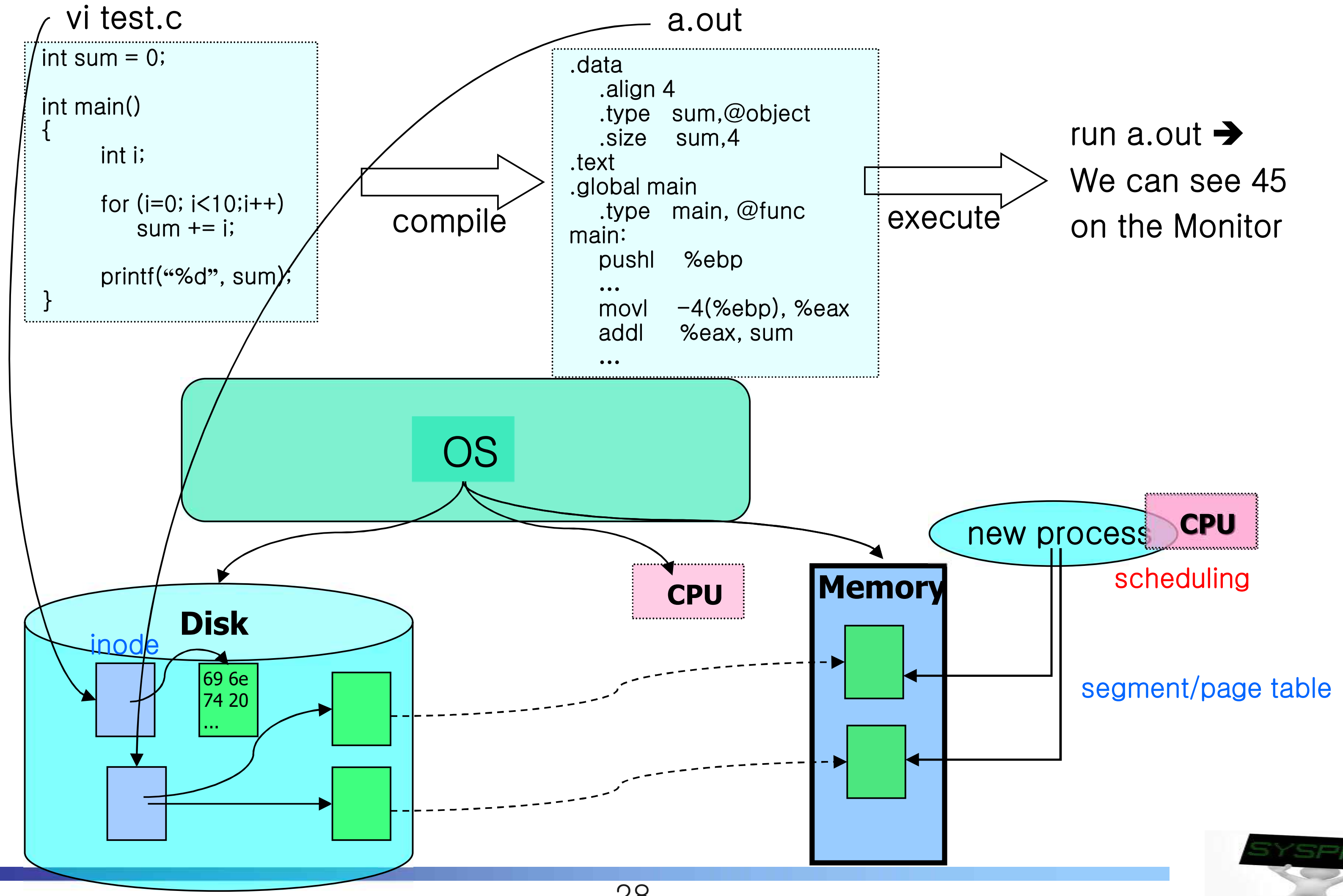
a.out

```
.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
    …
```

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

Operating System (11/13)

- 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);
}
```

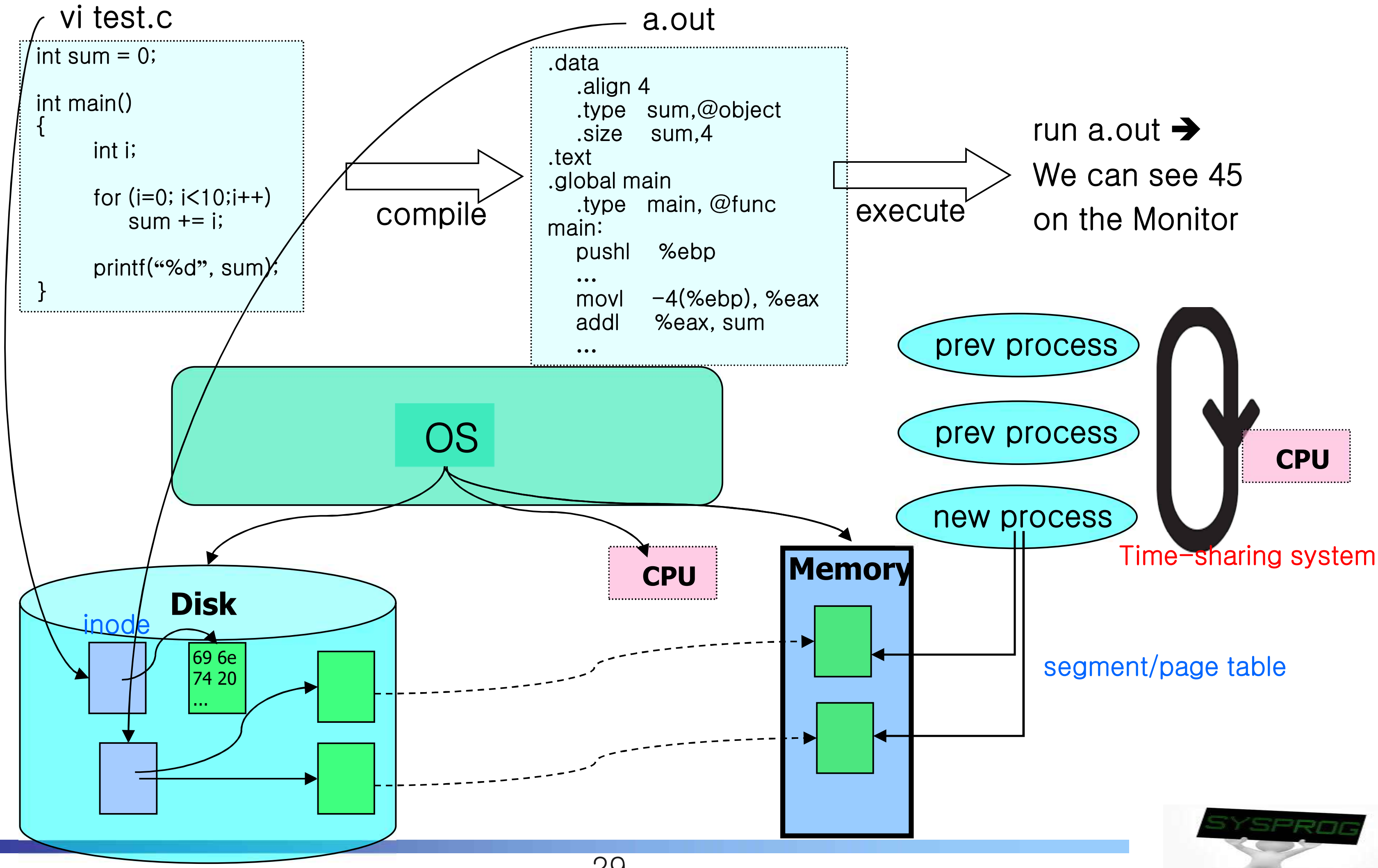
a.out

```
.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
    …
```

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

Operating System (12/13)

- Behaviors: 4) execute the a.out (system’s viewpoint)
  - ✓ Actually there are multiple processes



vi test.c

```
int sum = 0;

int main()
{
        int i;

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

        printf(“%d”, sum);
}
```

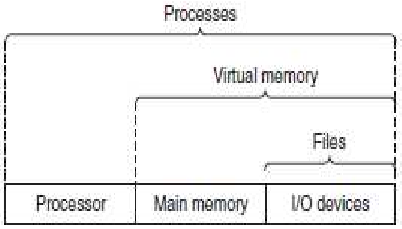
a.out

```
.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
    …
```

Operating System (13/13)

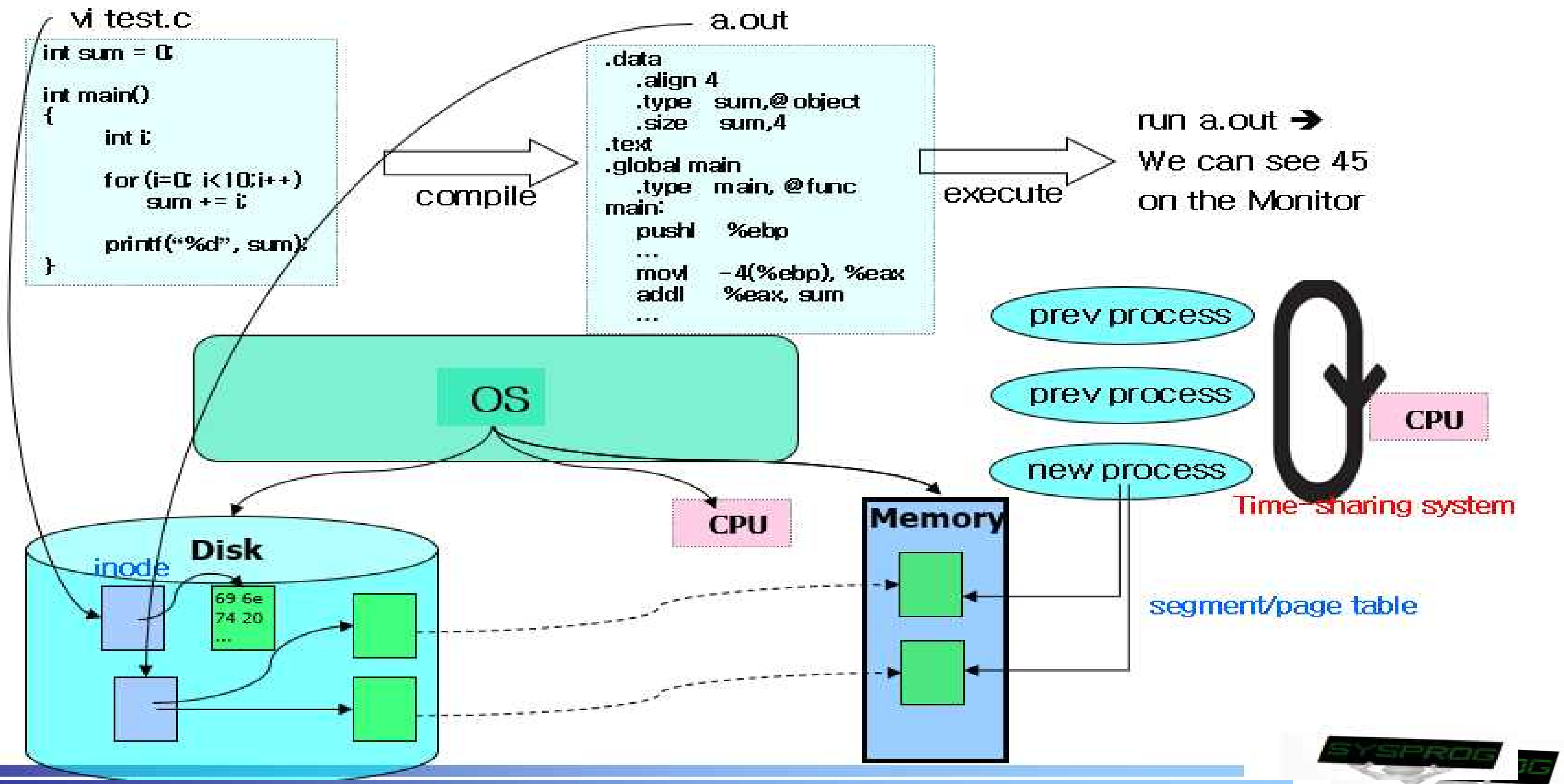
- Operating system: summary
  - ✓ Process manager (Task manager): CPU
    - process manipulation, schedule, IPC, signal, context switch
    - fork, exec, wait, getpid, (pthread_create) , …
  - ✓ Virtual Memory: Main memory
    - page, segment, address translation, buddy, LRU
    - brk, (malloc, free), …
  - ✓ 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.



Quiz for 2nd-Week 1st-Lesson

- Quiz
  - ✓ Discuss the role of inode and page table. What are the similarity and differences between page table and inode?
  - ✓ Due: until 6 PM Friday of this week(11th, September)

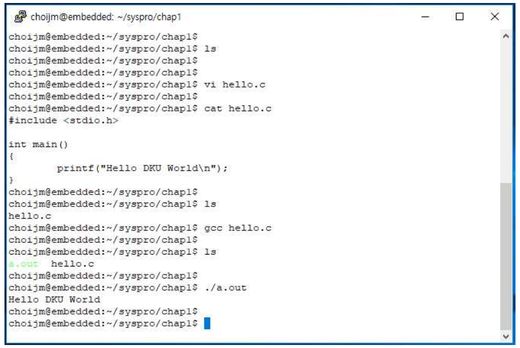


```
int sum = 0;

int main()
{
        int i;

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

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

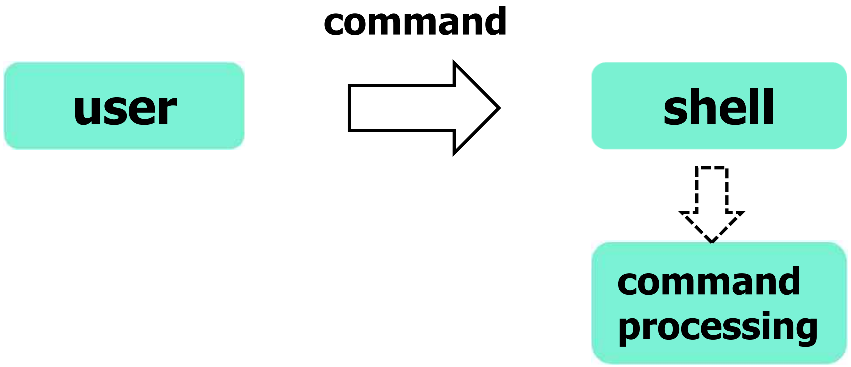
```
.data
    .align 4
    .type   sum,@object
    .size   sum,4
.text
.global main
    .type   main, @func
main:
    pushl   %ebp
    ...
    movl    -4(%ebp), %eax
    addl    %eax, sum
    ...
```

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

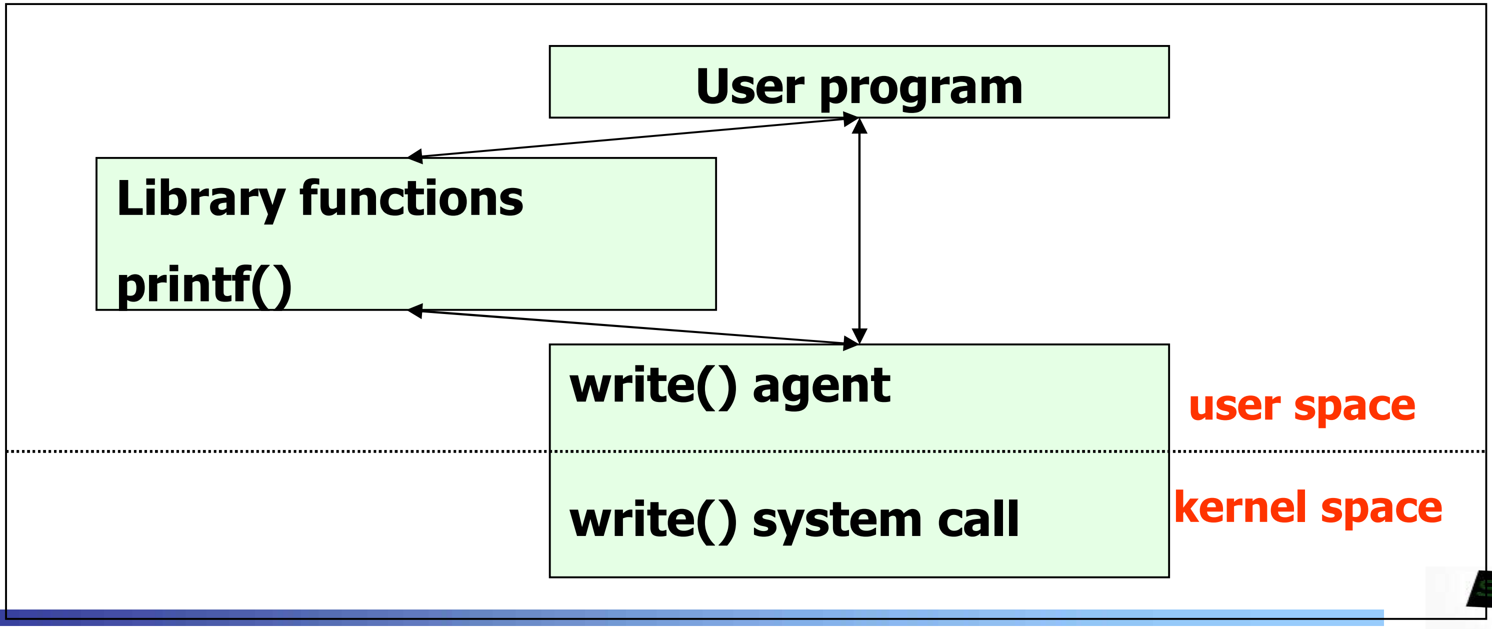


```
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ ls
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ vi hello.c
choijm@embedded:~/syspro/chap1$
choijm@embedded:~/syspro/chap1$ cat hello.c
#include <stdio.h>

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$
```



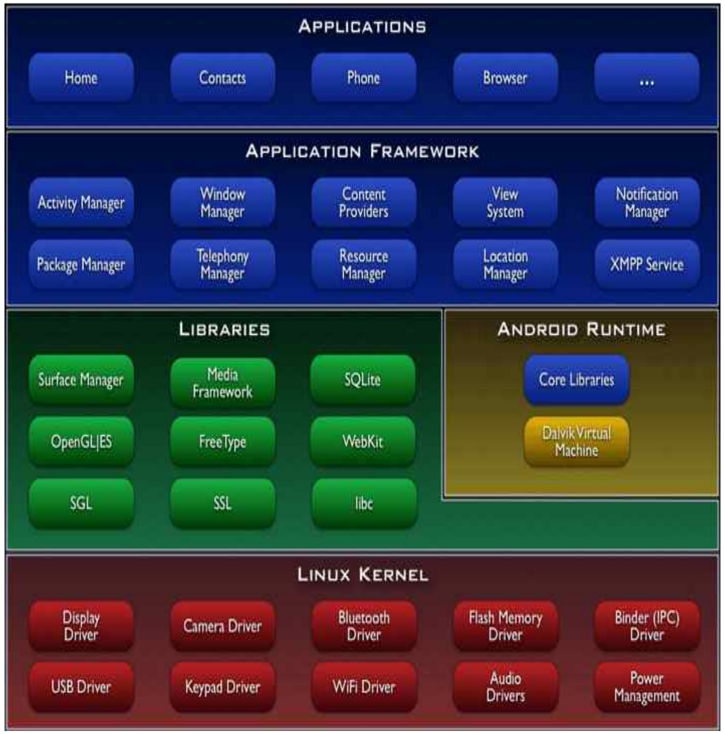
Runtime System (2/5)

- 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

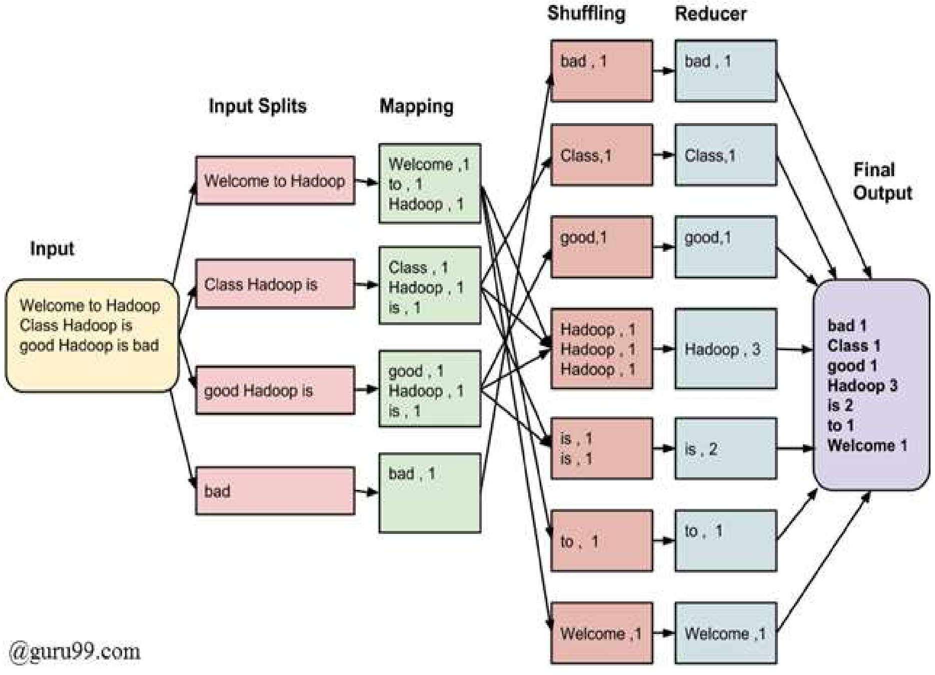


Runtime System (3/5)

- 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)

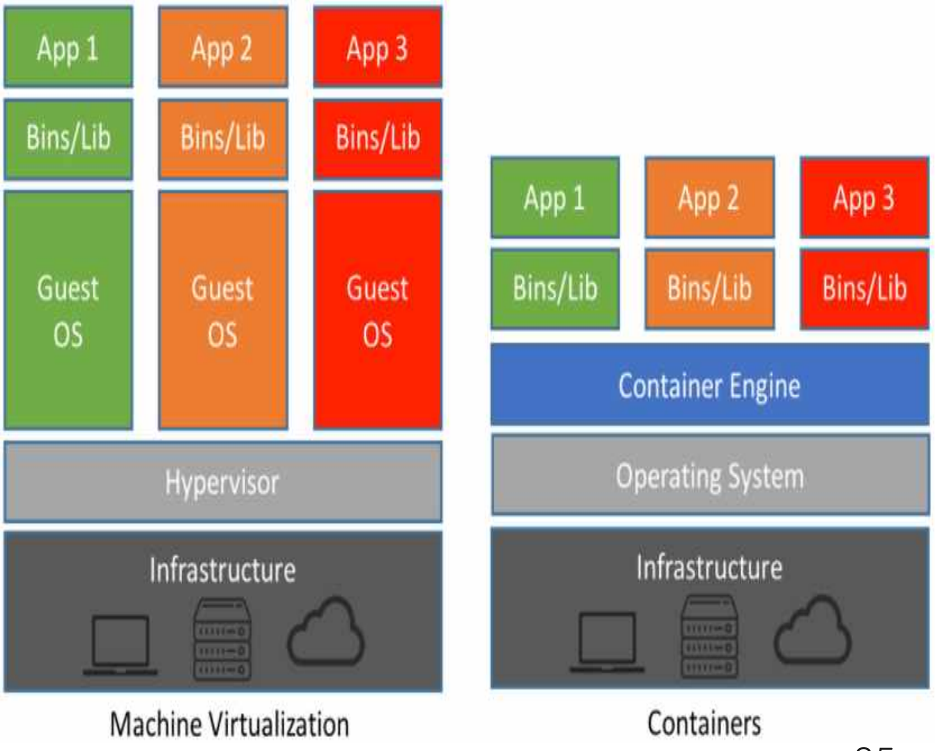


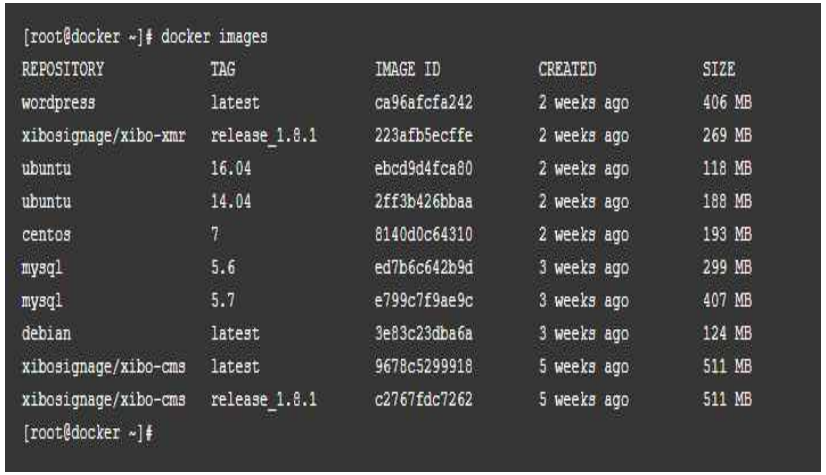
MapReduce Architecture

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

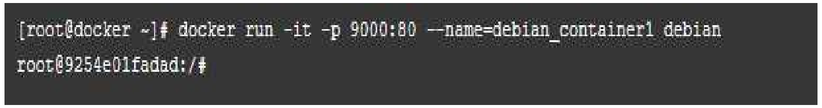
Runtime System (4/5)

- 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

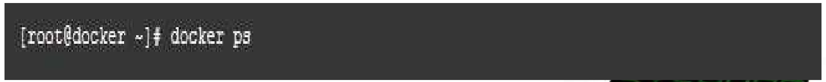




```
[root@docker ~]# docker images
REPOSITORY             TAG             IMAGE ID       CREATED       SIZE
wordpress              latest          ca96afcfa242   2 weeks ago   406 MB
xibosignage/xibo-xmr   release_1.8.1   223afb5ecffe   2 weeks ago   269 MB
ubuntu                 16.04           ebcd9d4fca80   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 ~]#
```



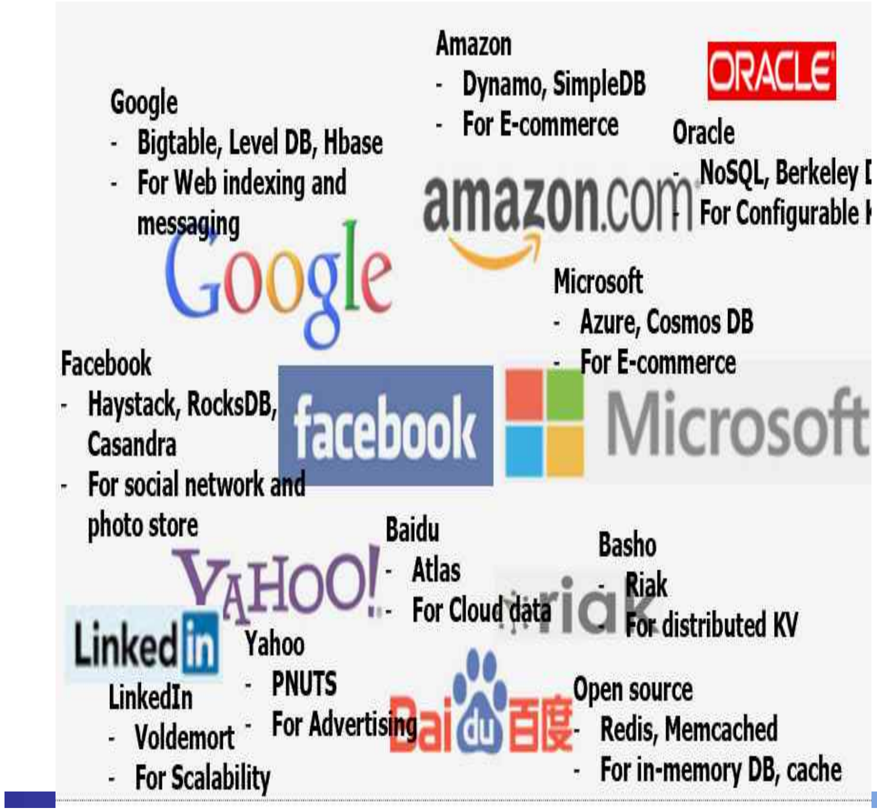
```
[root@docker ~]# docker run -it -p 9000:80 --name=debian_container1 debian
root@9254e01fadad:/#
```



```
[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, …



Google
- Bigtable, Level DB, Hbase
- For Web indexing and messaging

Facebook
- Haystack, RocksDB, Casandra
- For social network and photo store

LinkedIn
- Voldemort
- For Scalability

Yahoo
- PNUTS
- For Advertising

Amazon
- Dynamo, SimpleDB
- For E-commerce

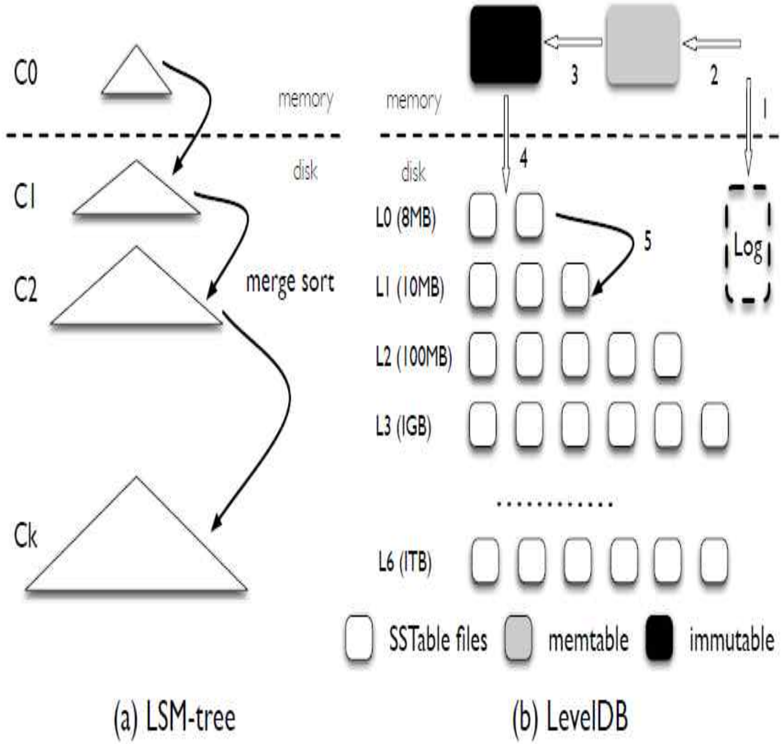
Oracle
- NoSQL, Berkeley D
- For Configurable K

Microsoft
- Azure, Cosmos DB
- For E-commerce

Baidu
- Atlas
- For Cloud data

Basho
- Riak
- For distributed KV

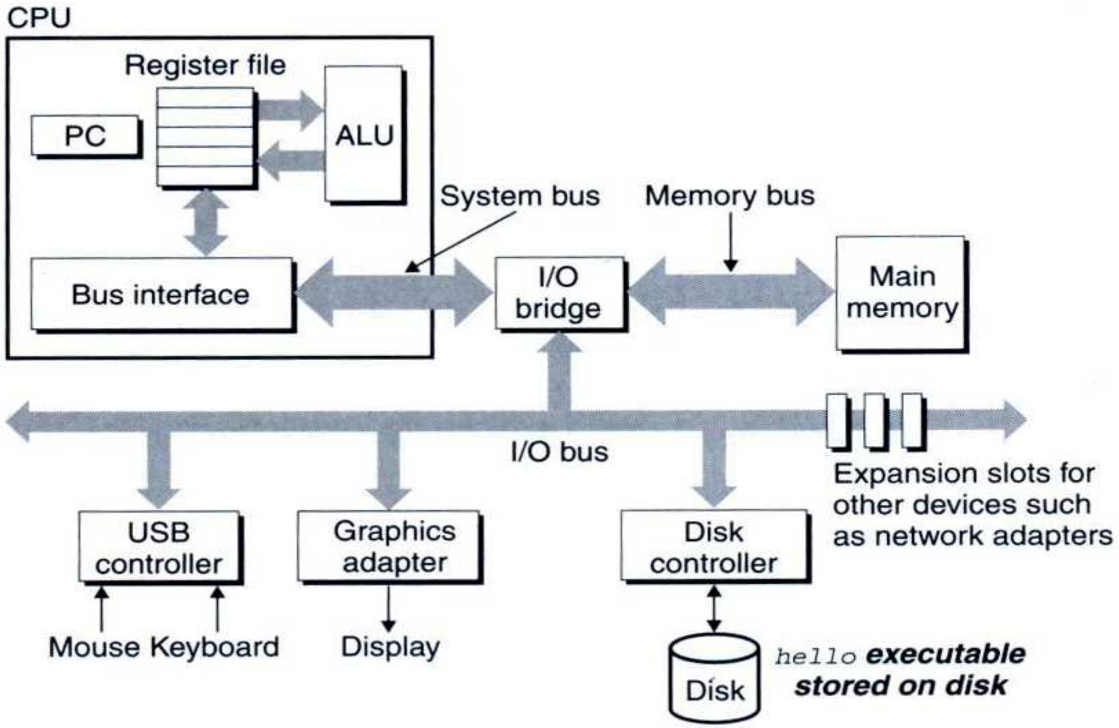
Open source
- Redis, Memcached
- For in-memory DB, cache



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.

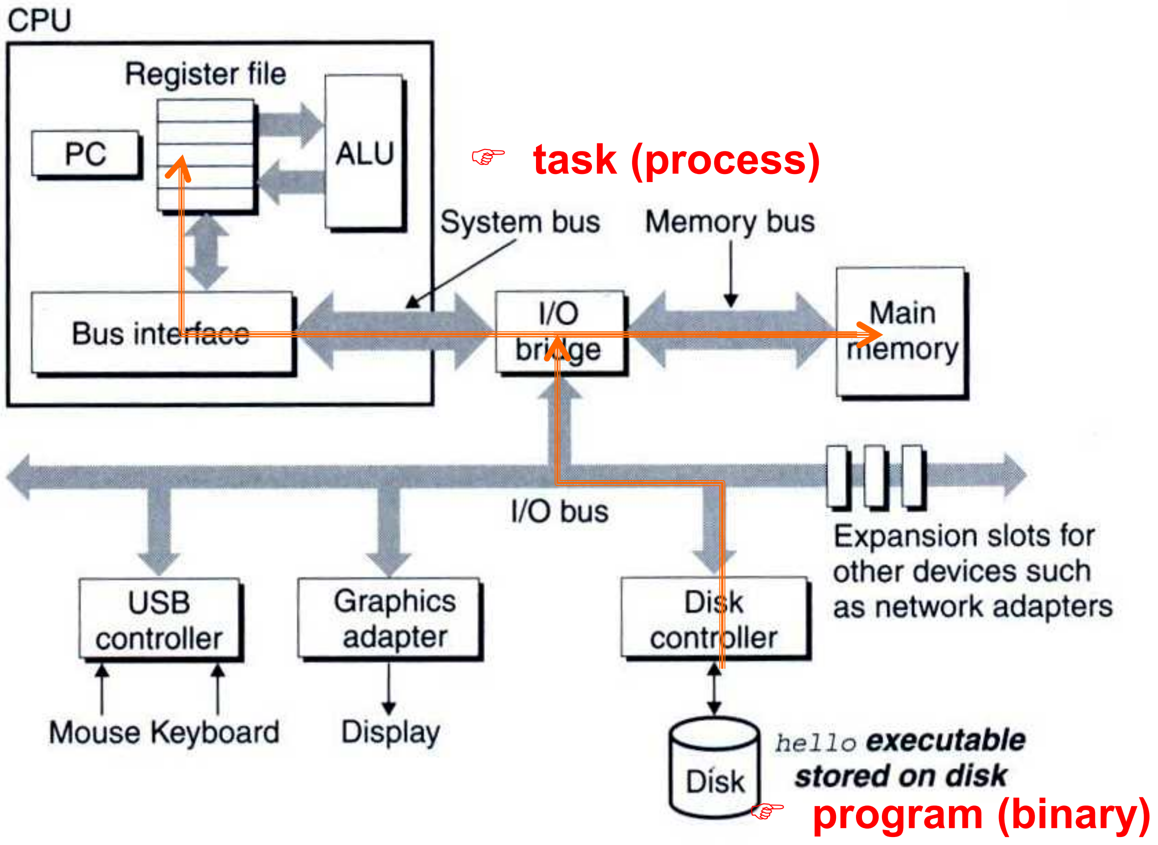


(Source: CSAPP)

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.



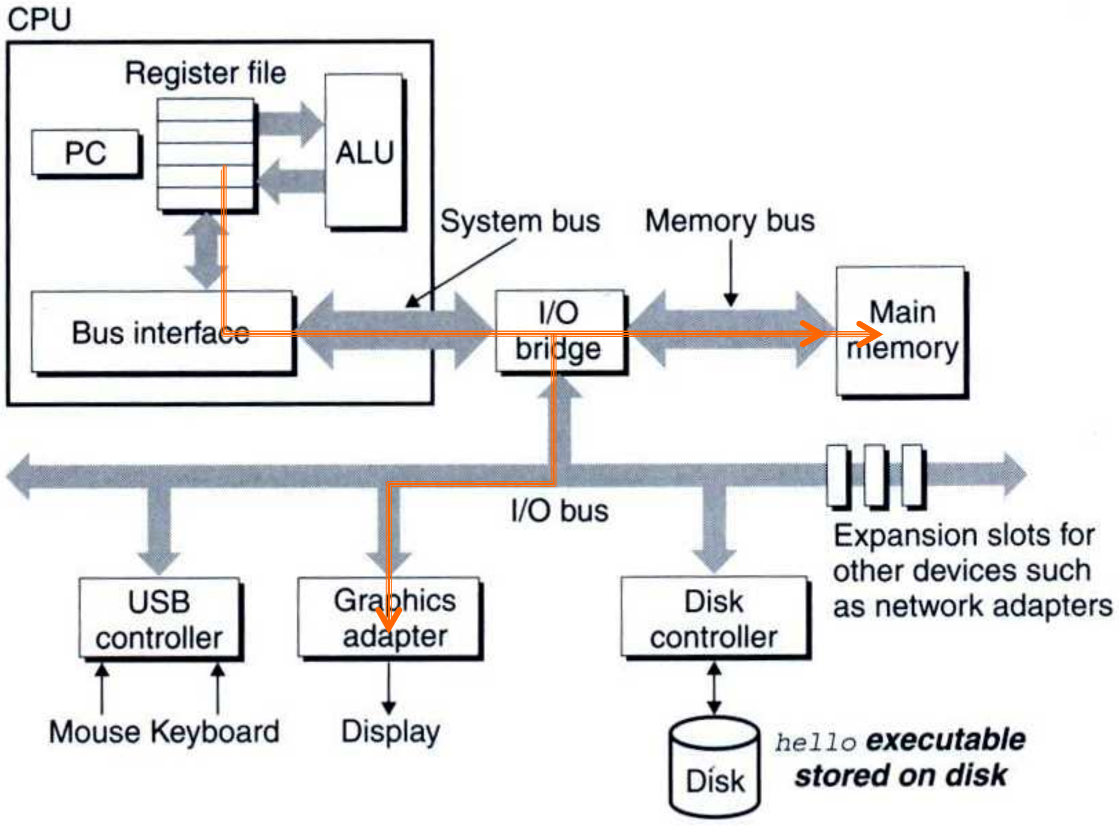
☞ **task (process)**

☞ **program (binary)**

Hardware consideration (3/6)

- Computer organization
  - ✓ When printf(“Hello World\n”) is conducted

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

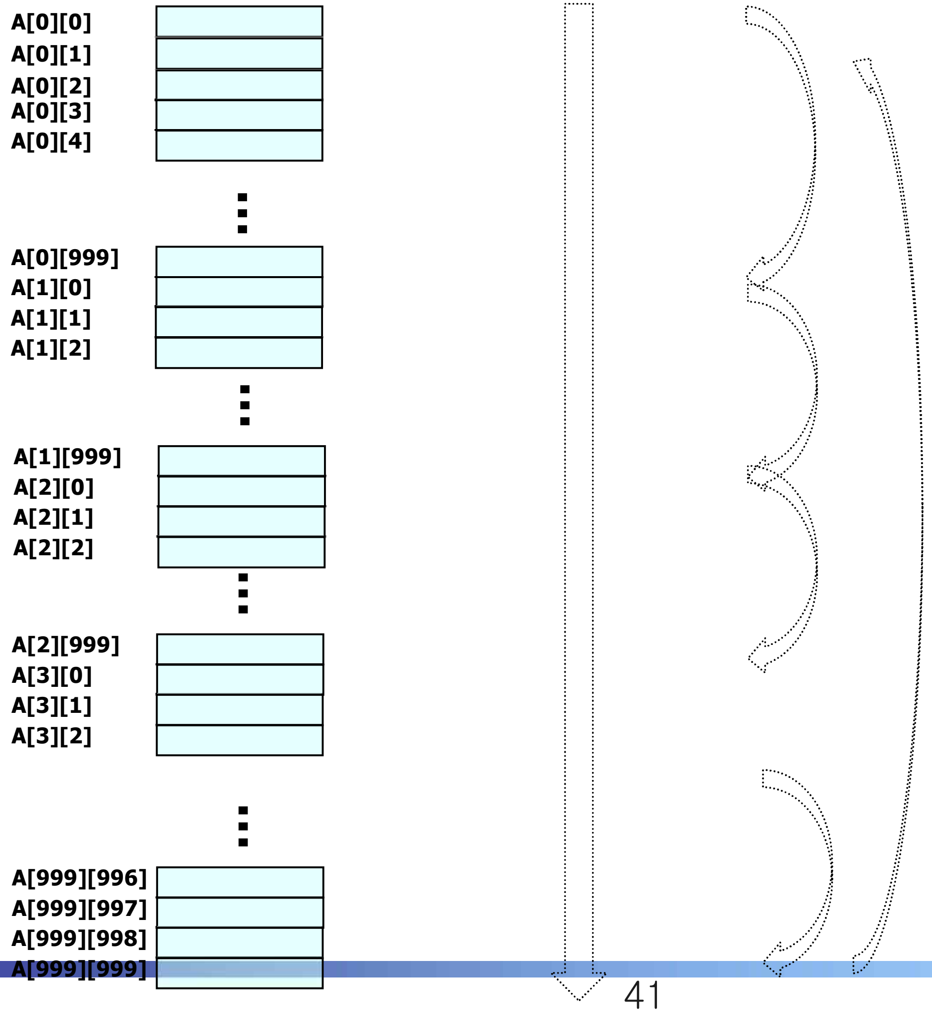
```
/* program A */
int a[1000][1000];
int i, j;
....

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

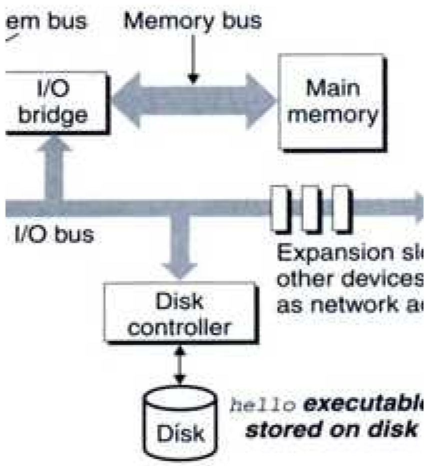
VS

```
/* 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)

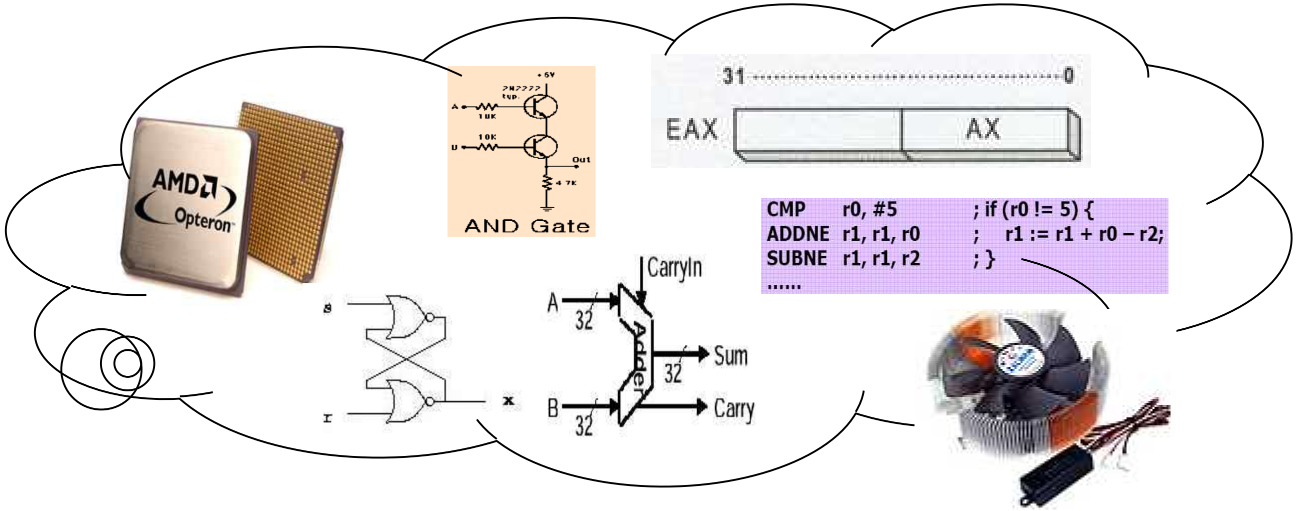
- Key of System Program: Abstraction
  - ✓ **Abstraction** is the process of generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose.
  - ✓ In computer science, abstraction tries 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)

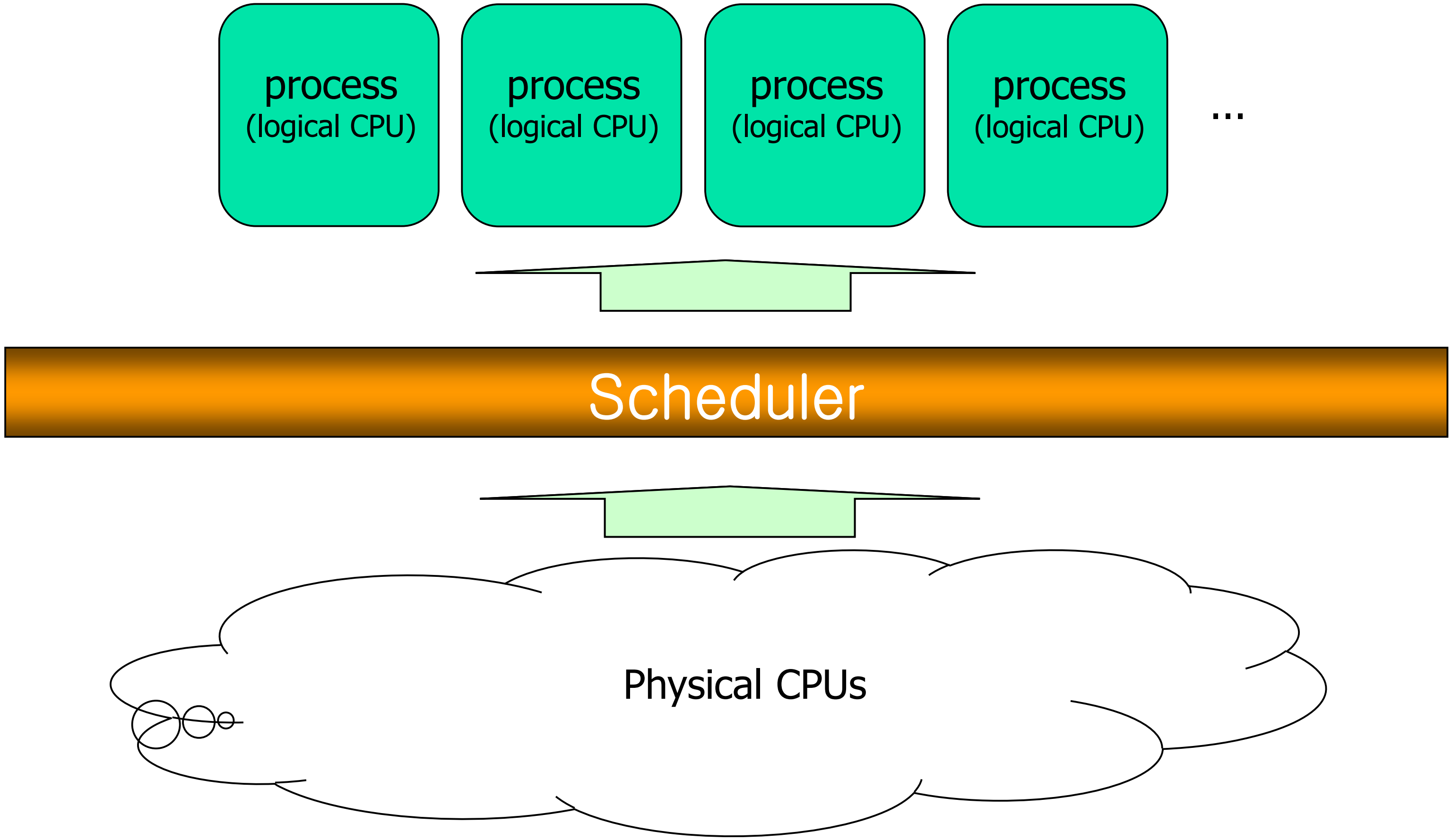
Compilation system



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

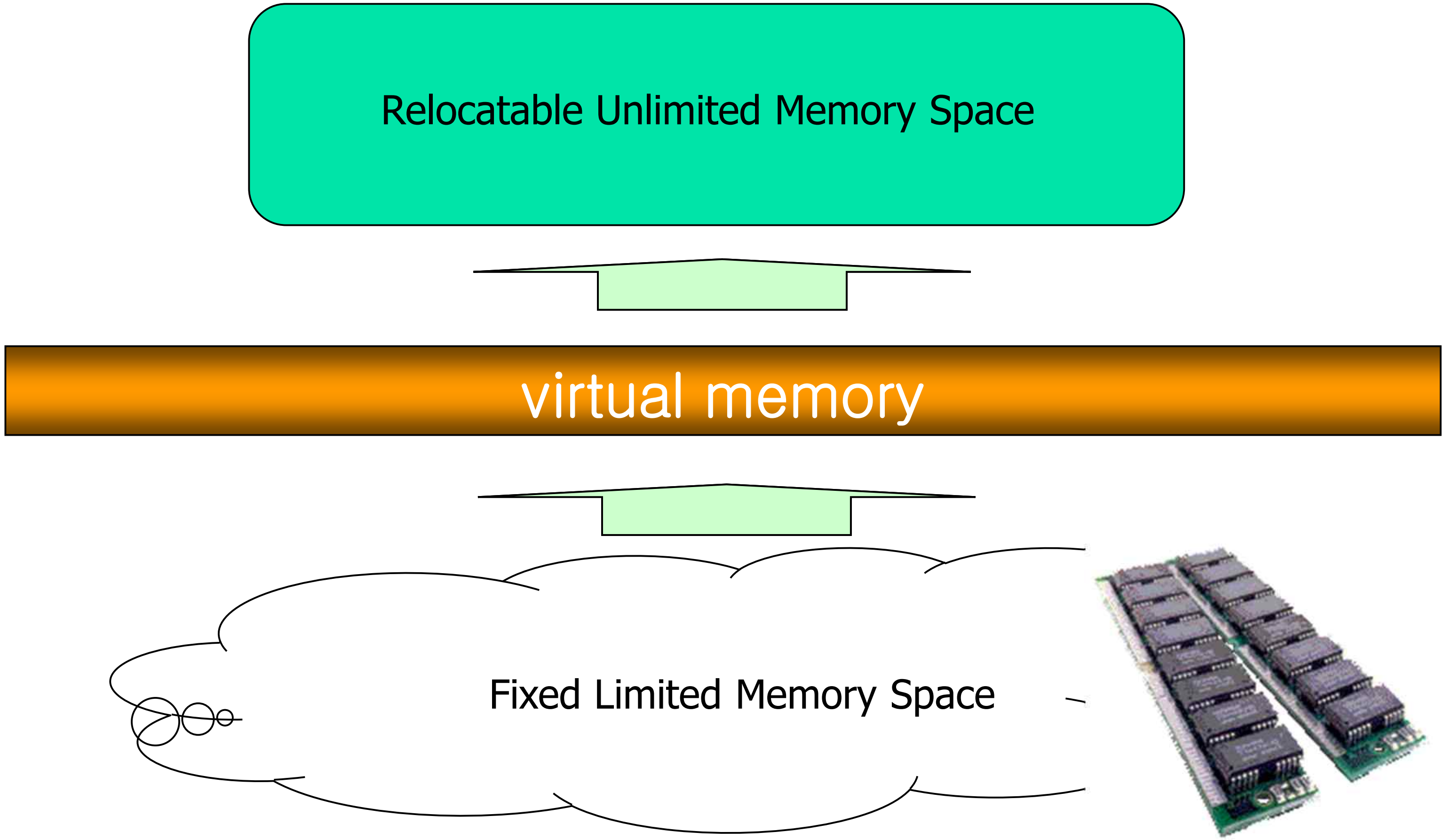
Abstraction (3/9)

- Multitasking



Abstraction (4/9)

- Memory management



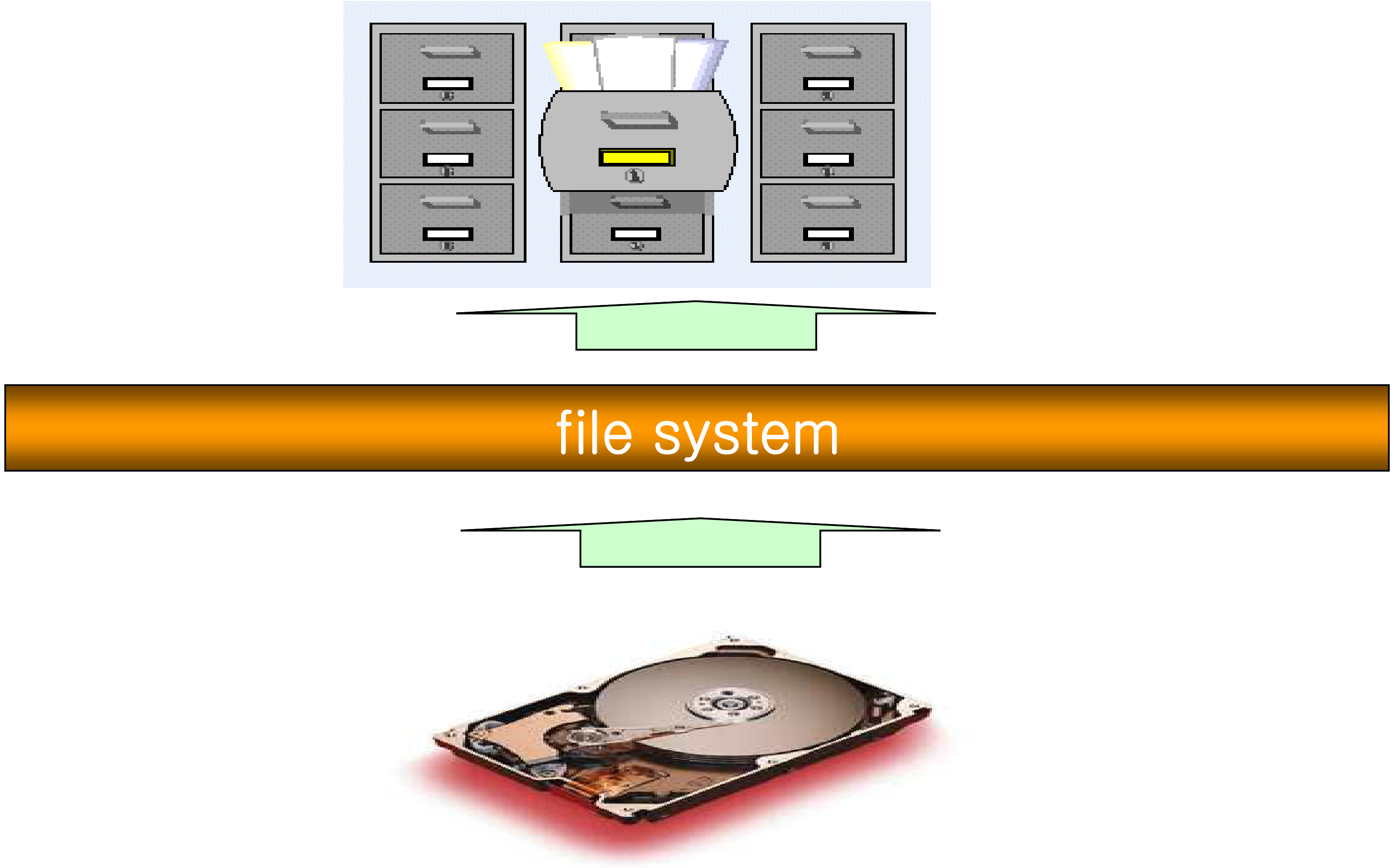
Relocatable Unlimited Memory Space

virtual memory

Fixed Limited Memory Space

Abstraction (5/9)

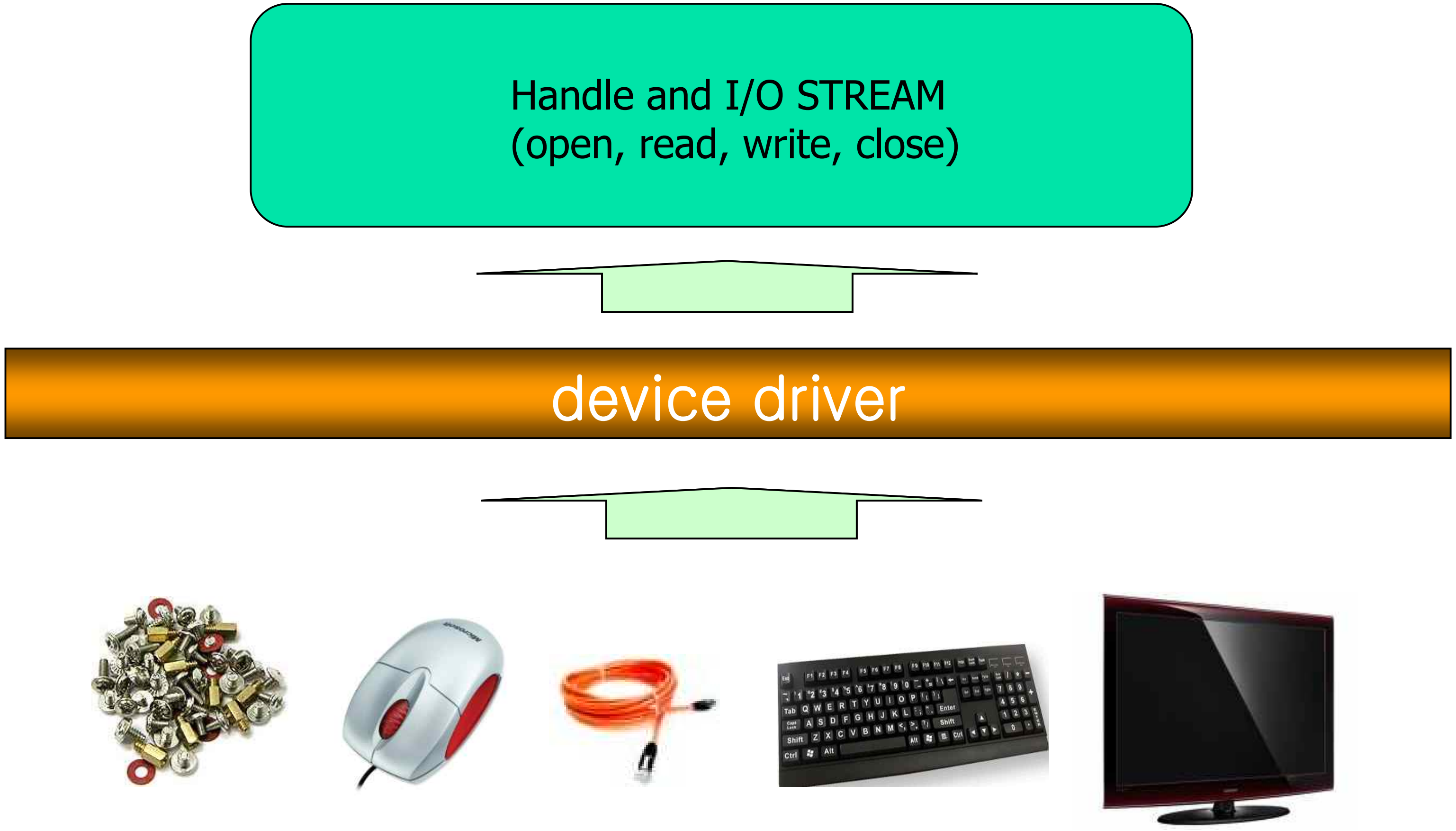
- File system



file system

Abstraction (6/9)

- Device driver

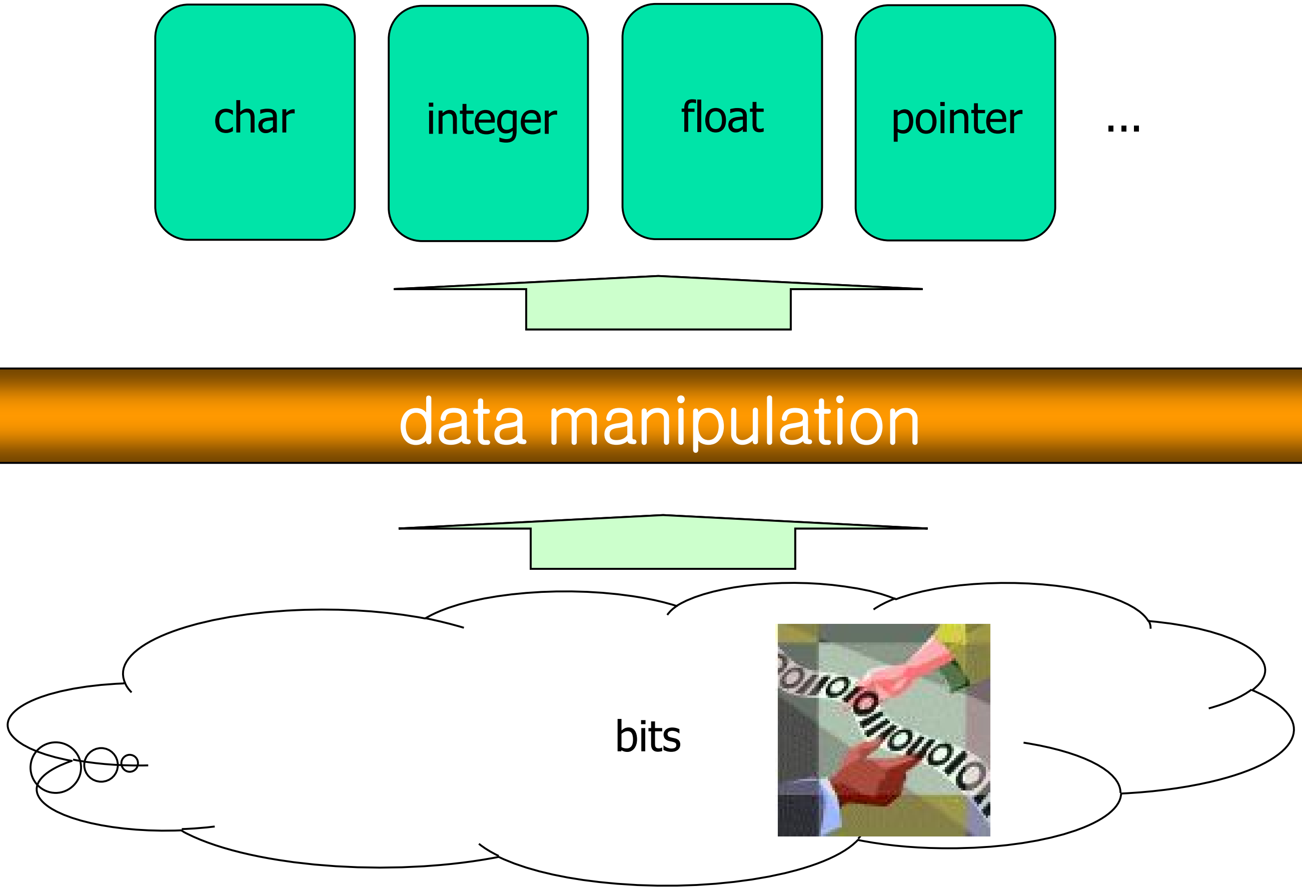


Handle and I/O STREAM
(open, read, write, close)

device driver

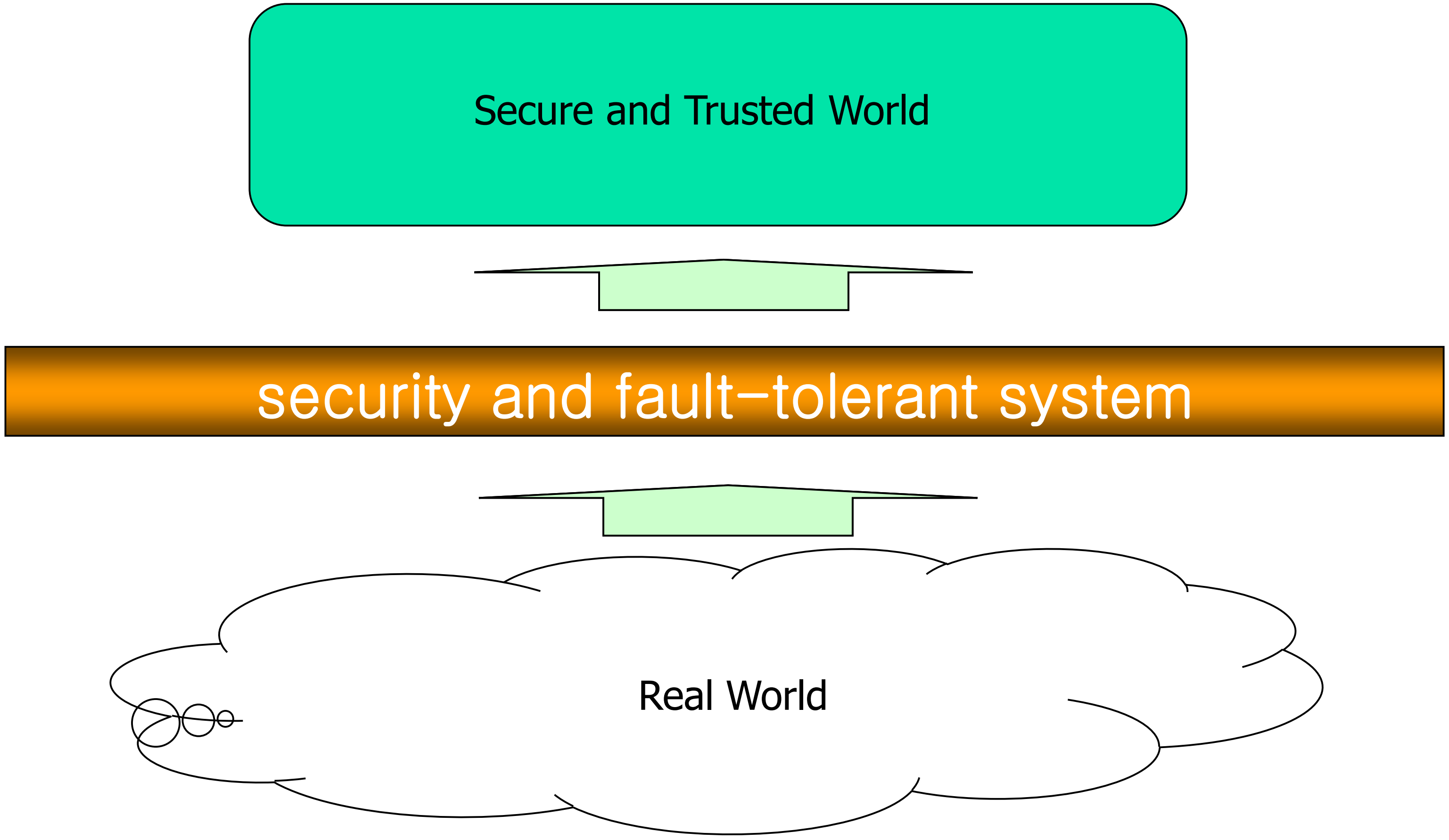
Abstraction (7/9)

- Data representation



Abstraction (8/9)

- Security and reliability



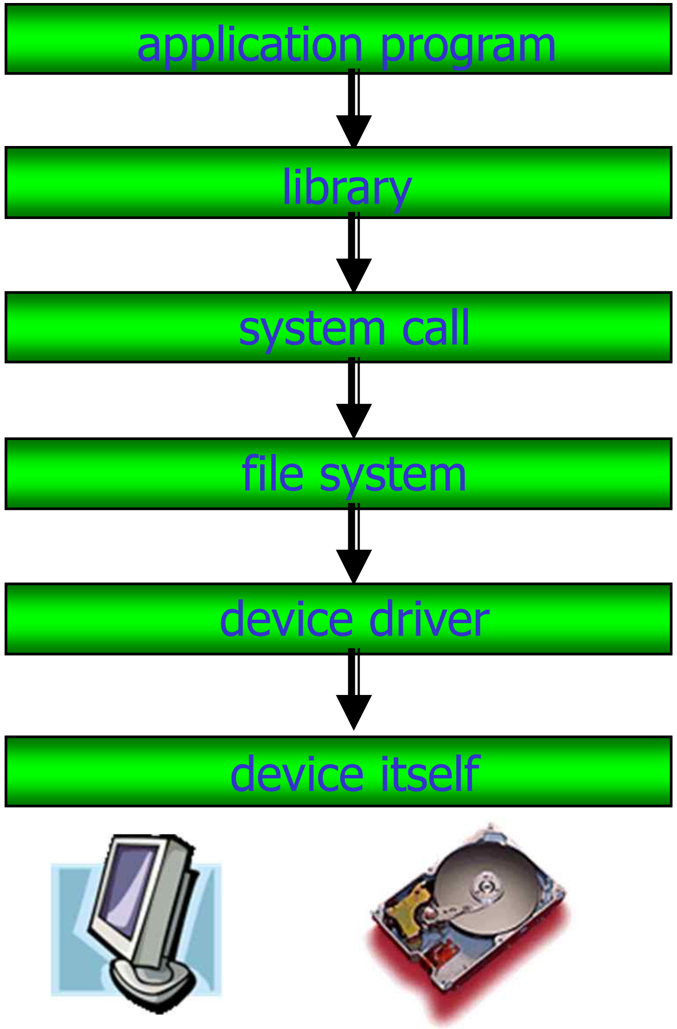
Secure and Trusted World

security and fault-tolerant system

Real World

Abstraction (9/9)

- Software layers (Layered architecture)



application program

library

system call

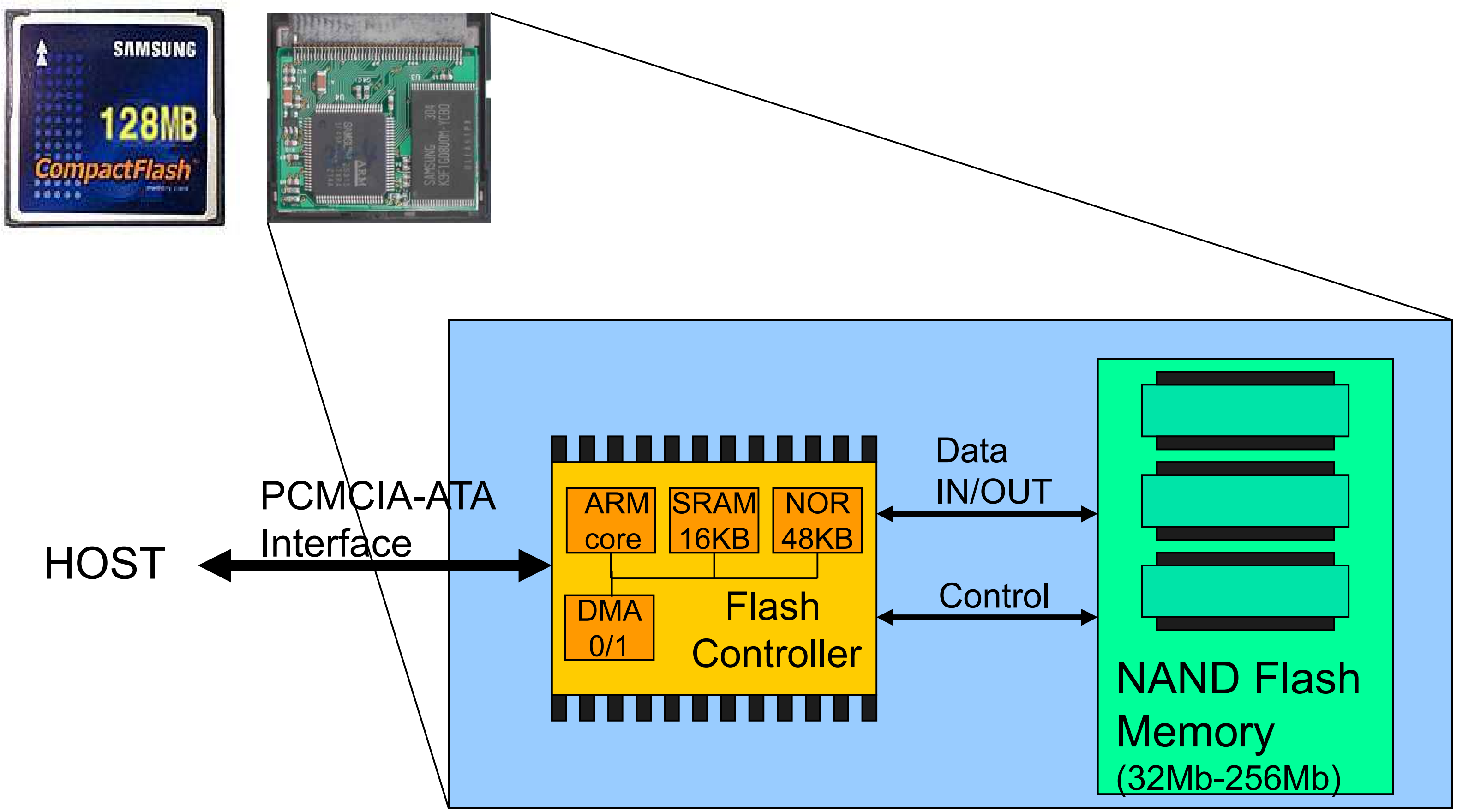
file system

device driver

device itself

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

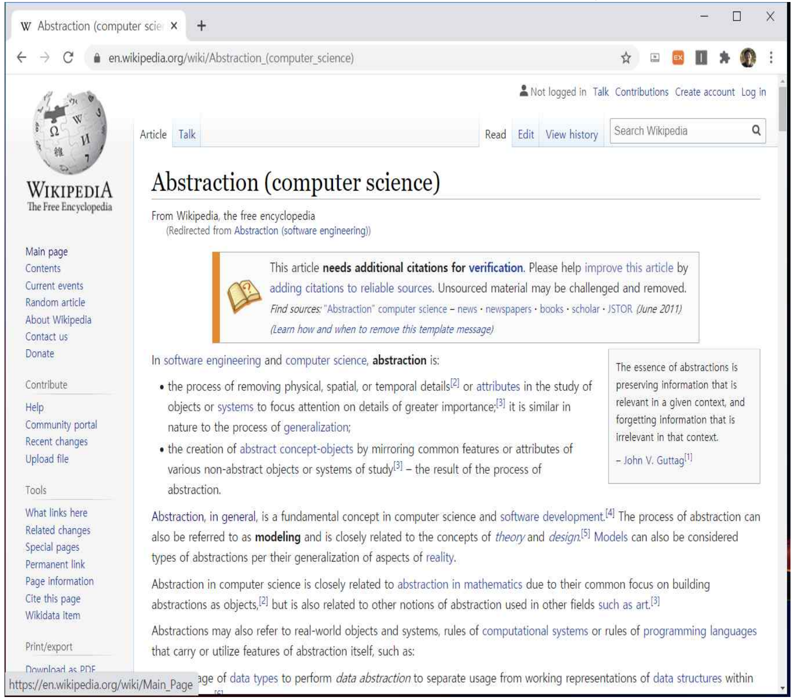
- Definition of System Program
  - ✓ Supporting computing environments
  - ✓ Managing hardware directly
- 3 Types of System Program
  - ✓ Compilation system, operating system, runtime system
  - ✓ Hardware consideration
- Concept of Abstraction
  - ✓ Information hiding
  - ✓ Layered architecture

☞ **Homework 1: Read the Chapter 1, “A Tour of Computer Systems” and summarize it.**

- ✓ **Requirement: 1) From the beginning to the Section 1.4 (at most 4 pages, 10 font (can be larger for section or subsection title)), 2) What is the purpose of studying System Programming? (1 page), 3) Section 1.7 (2 pages, discuss about “process, thread, virtual memory and file”)**
- ✓ **Deadline: 6 PM Friday of the next week (18th, September)**
- ✓ **Caution: Do not copy!! Send your report as a “pdf” file!!**

Quiz for 2nd-Week 2nd-Lesson

- Quiz
  - ✓ Describe an example of abstraction in your life and discuss the features of abstraction in the 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 (11th, September)



Abstraction (computer science)

From Wikipedia, the free encyclopedia
(Redirected from Abstraction (software engineering))

This article **needs additional citations for verification**. Please help improve this article by adding citations to reliable sources. Unsourced material may be challenged and removed.
Find sources: "Abstraction" computer science – news · newspapers · books · scholar · JSTOR *(June 2011)*
(Learn how and when to remove this template message)

In software engineering and computer science, **abstraction** is:

- the process of removing physical, spatial, or temporal details[2] or attributes in the study of objects or systems to focus attention on details of greater importance;[3] it is similar in nature to the process of generalization;
- the creation of abstract concept-objects by mirroring common features or attributes of various non-abstract objects or systems of study[3] – the result of the process of abstraction.

> The essence of abstractions is preserving information that is relevant in a given context, and forgetting information that is irrelevant in that context.
>
> – John V. Guttag[1]

Abstraction, in general, is a fundamental concept in computer science and software development.[4] The process of abstraction can also be referred to as **modeling** and is closely related to the concepts of *theory* and *design*.[5] Models can also be considered types of abstractions per their generalization of aspects of reality.

Abstraction in computer science is closely related to abstraction in mathematics due to their common focus on building abstractions as objects,[2] but is also related to other notions of abstraction used in other fields such as art.[3]

Abstractions may also refer to real-world objects and systems, rules of computational systems or rules of programming languages that carry or utilize features of abstraction itself, such as:

age of data types to perform *data abstraction* to separate usage from working representations of data structures within

I'M THE PERFECT ABSTRACTION OF A DUCK!
MAYBE ARE YOU SLIGHTLY TOO ABSTRACT?

(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

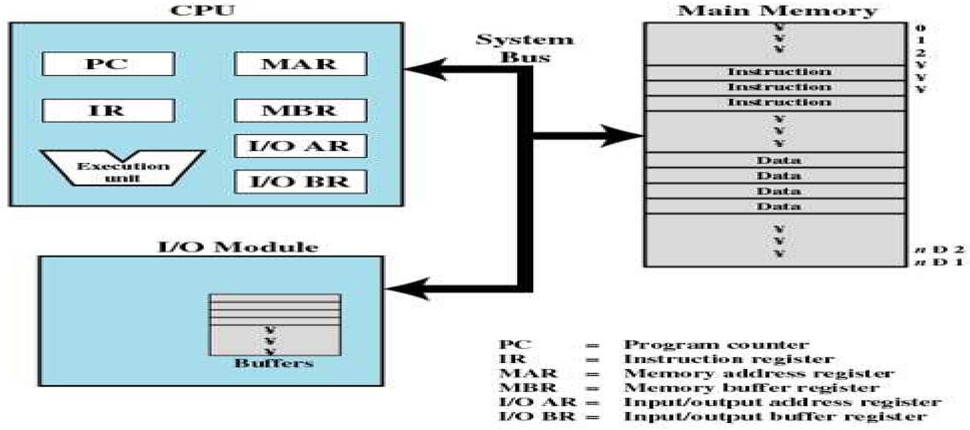

CPU
PC
MAR
IR
MBR
I/O AR
Execution unit
I/O BR
System Bus
Main Memory
0
1
2
Instruction
Instruction
Instruction
Data
Data
Data
Data
n-2
n-1
I/O Module
Buffers
PC = Program counter
IR = Instruction register
MAR = Memory address register
MBR = Memory buffer register
I/O AR = Input/output address register
I/O BR = Input/output buffer register

Figure 1.1 Computer Components: Top-Level View

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