VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
INTERNATIONAL UNIVERSITY

# Operating Systems

## W1 - Introduction

Le Hai Duong, PhD. (lhduong@hcmiu.edu.vn) - 08/09/2021

# Materials

- Textbook:
    - Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2018). *Operating systems: Three easy pieces*. Arpaci-Dusseau Books LLC. (Online book: https://pages.cs.wisc.edu/~remzi/OSTEP/)
- Other useful books:
    - Galvin, P. B., Gagne, G., & Silberschatz, A. (2018). *Operating system concepts* (10th ed.). John Wiley & Sons.
    - Tanenbaum, A. S., & Bos, H. (2015). *Modern operating systems* (4th ed.). Pearson.

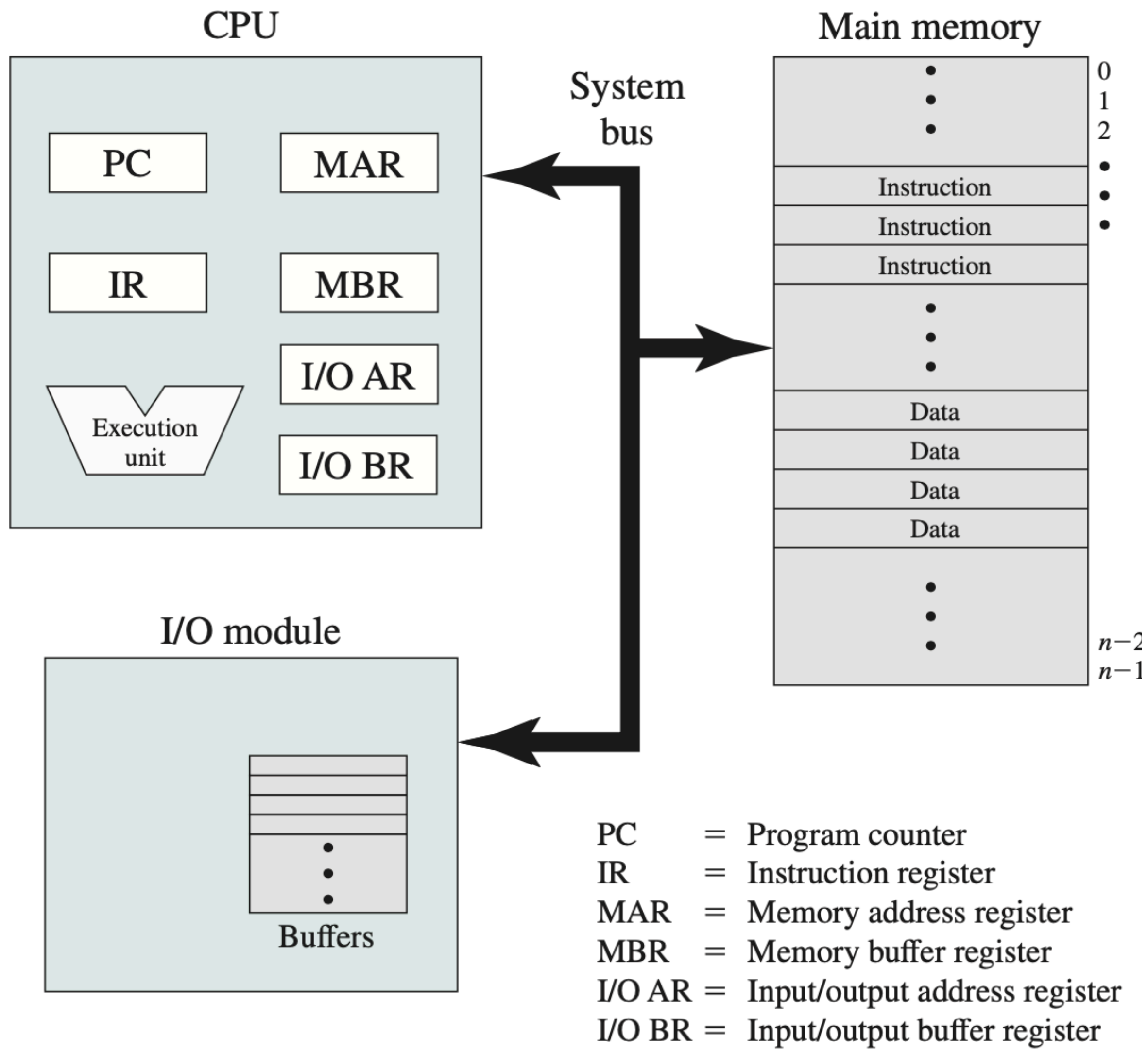
**Tentative Schedule**

| Session | Date | Title | Reading |
|---|---|---|---|
| | | **VIRTUALIZATION** | |
| 1 | Sep 8, 2021 | - Introduction | - Intro: pre, Ch1, Ch2 |
| 2 | Sep 15, 2021 | - Processes | Ch3, Ch4, Ch5, Ch6 |
| 3 | Sep 22, 2021 | - Schedule | Ch7, Ch8, Ch11 |
| 4 | Sep 29, 2021 | - Memory management<br>- Paging | - Mem Mgmt: Ch13, Ch15, Ch16<br>- Paging: Ch18 |
| 5 | Oct 6, 2021 | - Paging: TLBs<br>- Paging: Smaller Tables | - TLBs: Ch19<br>- Smaller: Ch20 |
| | | Beyond Physical: Swapping | Ch21, Ch22 |
| 6 | Oct 13, 2021 | **CONCURRENCY** | |
| 7 | Oct 20, 2021 | - Threads<br>- Locks | - Threads: Ch25, Ch26<br>- Locks: Ch28 |
| 8 | Oct 27, 2021 | - Locks, CVs<br>- Condition variables | - Locks, CVs: Ch29<br>- Condition variables: Ch30 |
| **MIDTERM (1/11/2021 - 14/11/2021)** | | | |
| 9 | Nov 17, 2021 | Semaphores | Ch31 |
| | | Deadlocks | Ch32 |
| 10 | Nov 24, 2021 | **PERSISTENCE** | |
| 11 | Dec 1, 2021 | - I/O + Disks<br>- Disk schedule | - I/O: Ch36<br>- Disk, disk schedule: Ch37 |
| 12 | Dec 8, 2021 | - RAID<br>- File systems | - RAID: Ch38<br>- File systems: Ch39 |
| 13 | Dec 15, 2021 | - File-System Implementation<br>- Fast file system | - FS Impl: Ch40<br>- FFS: Ch41 |
| 14 | Dec 22, 2021 | Journaling | Ch42 |
| 15 | Dec 29, 2021 | - LFS<br>- SSD | - LFS: Ch43<br>- SSD: Ch44 |
| **FINAL (03/01/2021 - 21/01/2021)** | | | |

# Gradings

| | |
|---|---|
| • Labs, homeworks, quizzes | 30% |
| • Midterm exam | 30% |
| • Final exam | 40% |

# Computer Organization



| | | |
|---|---|---|
| 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**

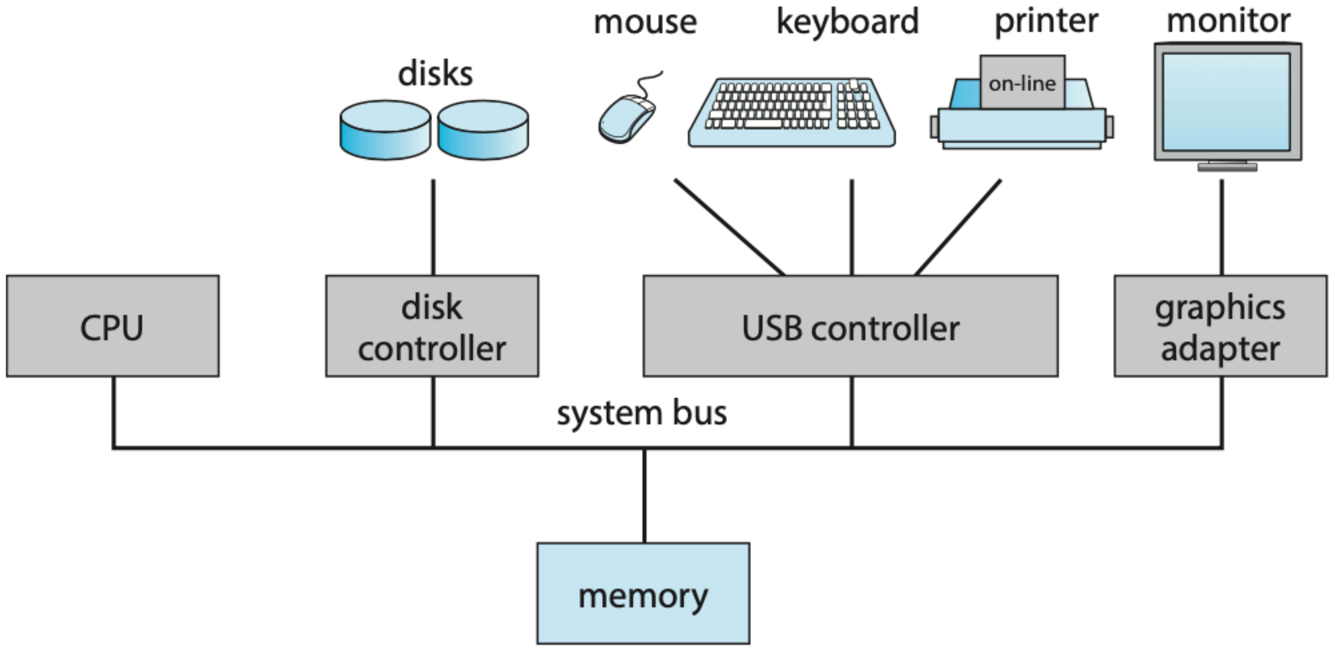
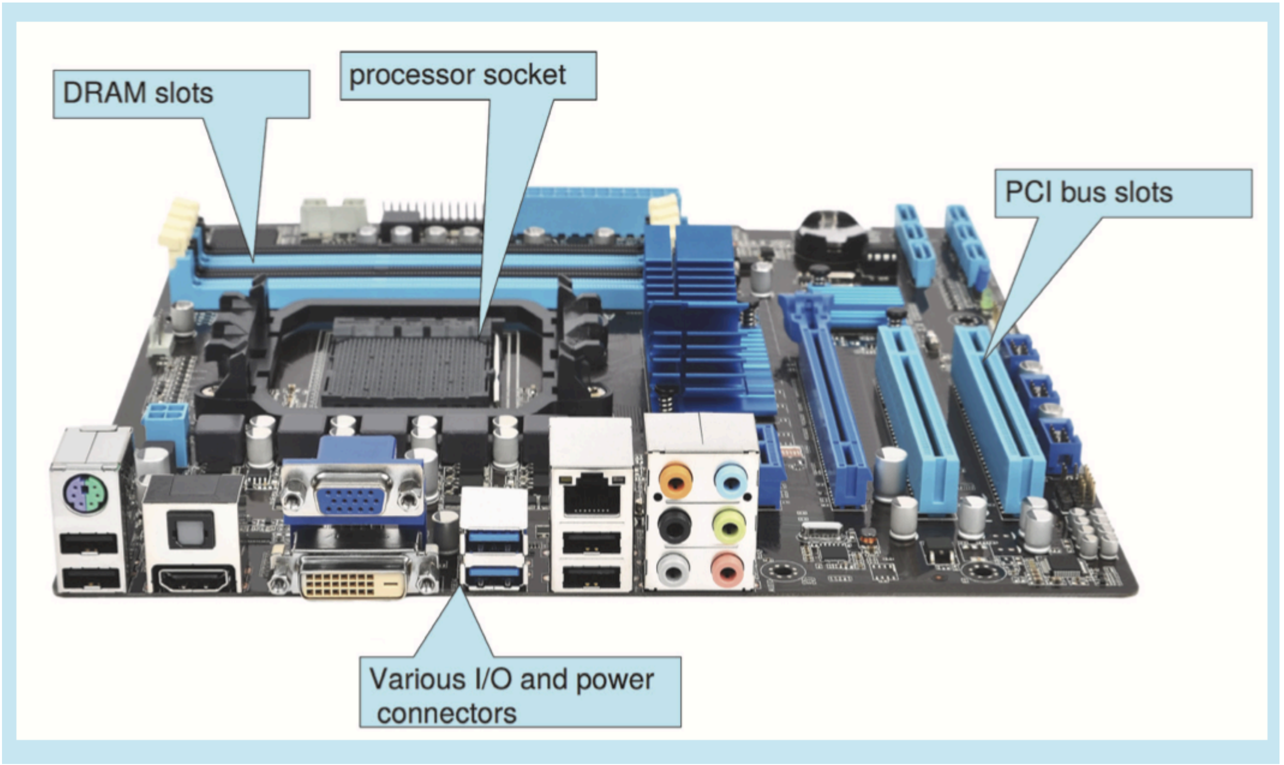


**Figure 1.2** A typical PC computer system.



# Storage Structure

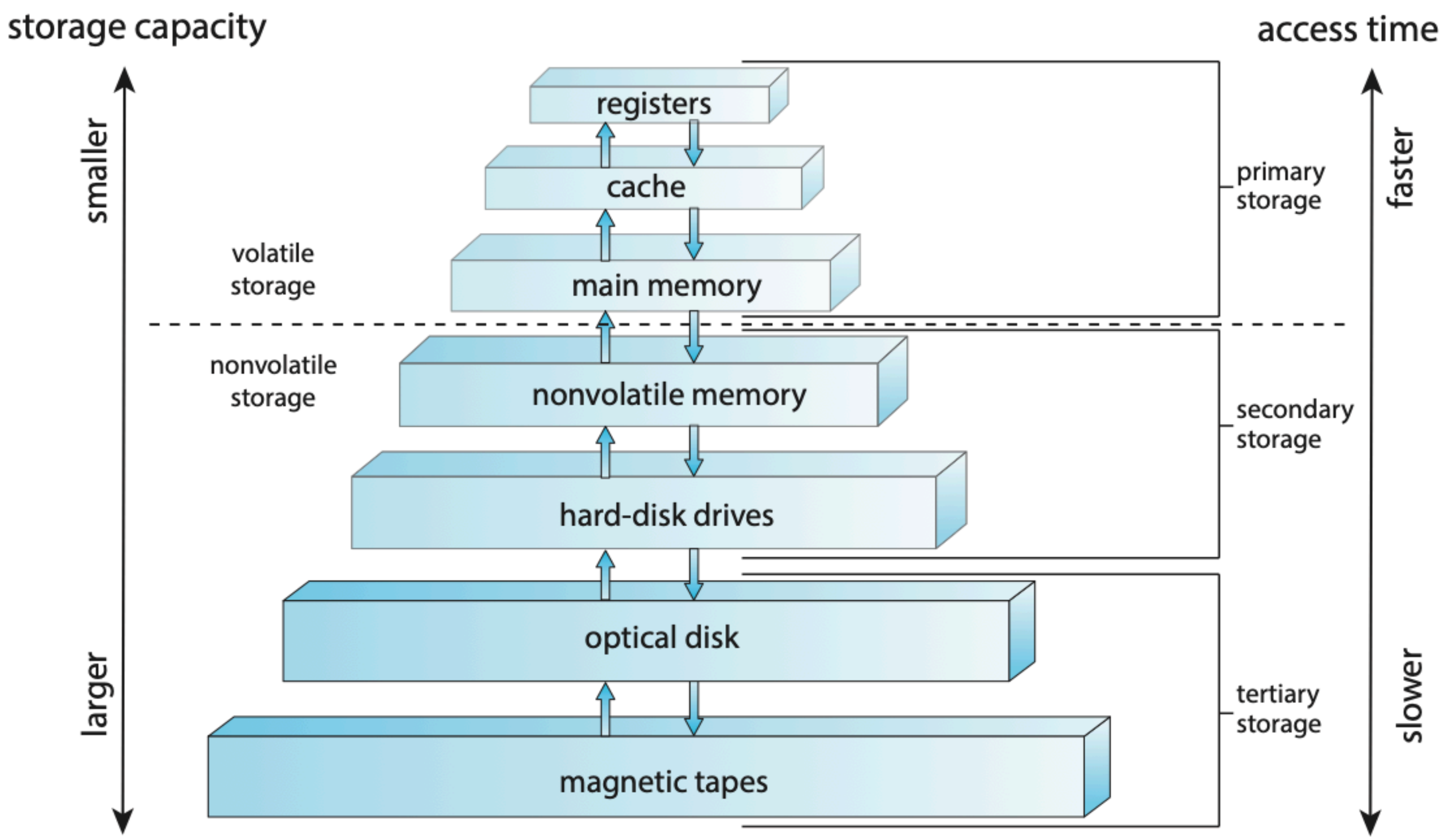


**Figure 1.6** Storage-device hierarchy.

# Computer Architecture

- Single-processor systems
- Multiprocessor systems
  - Symmetric Multiprocessing (SMP)
- Clusters

○ **core** is the component that executes instructions and has registers for storing data locally

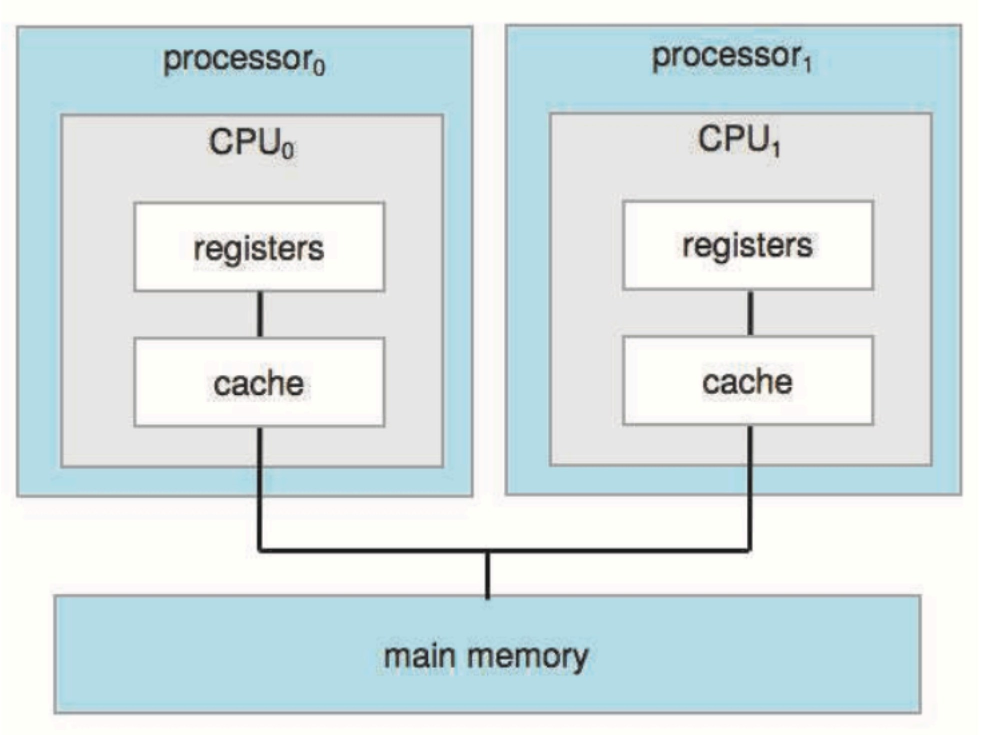


**Figure 1.8** Symmetric multiprocessing architecture.

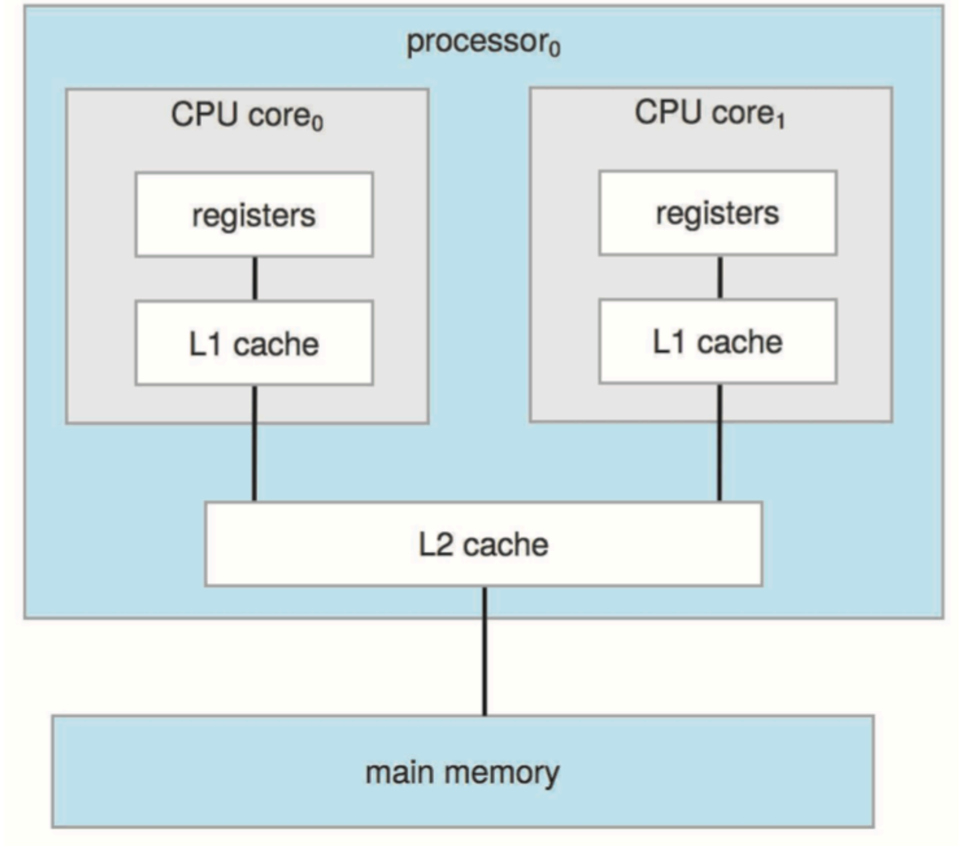


**Figure 1.9** A dual-core design with two cores on the same chip.

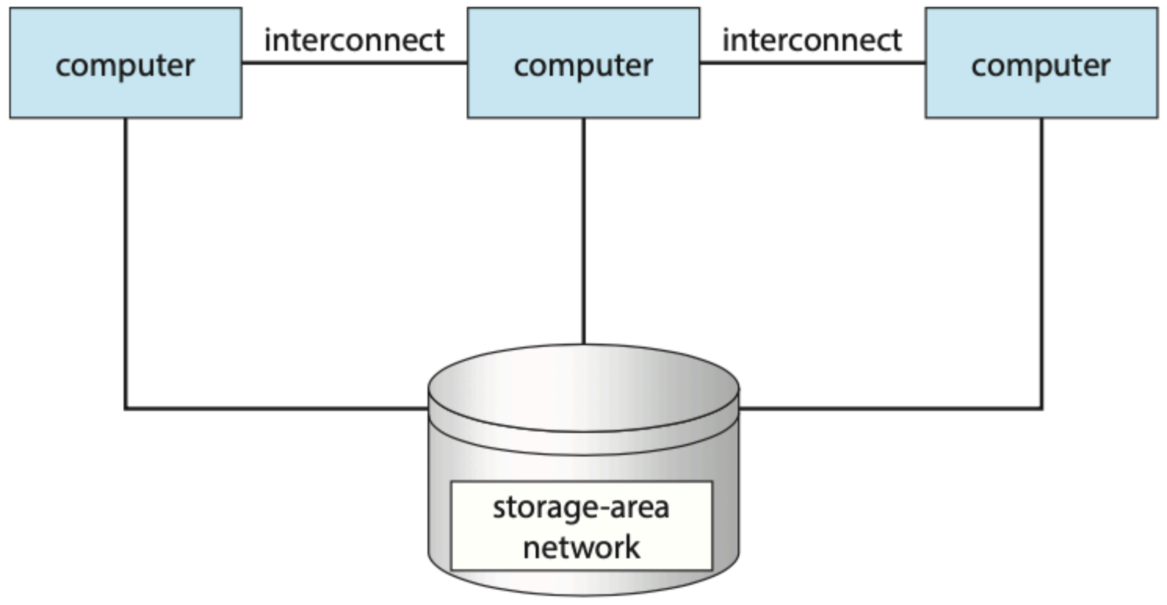
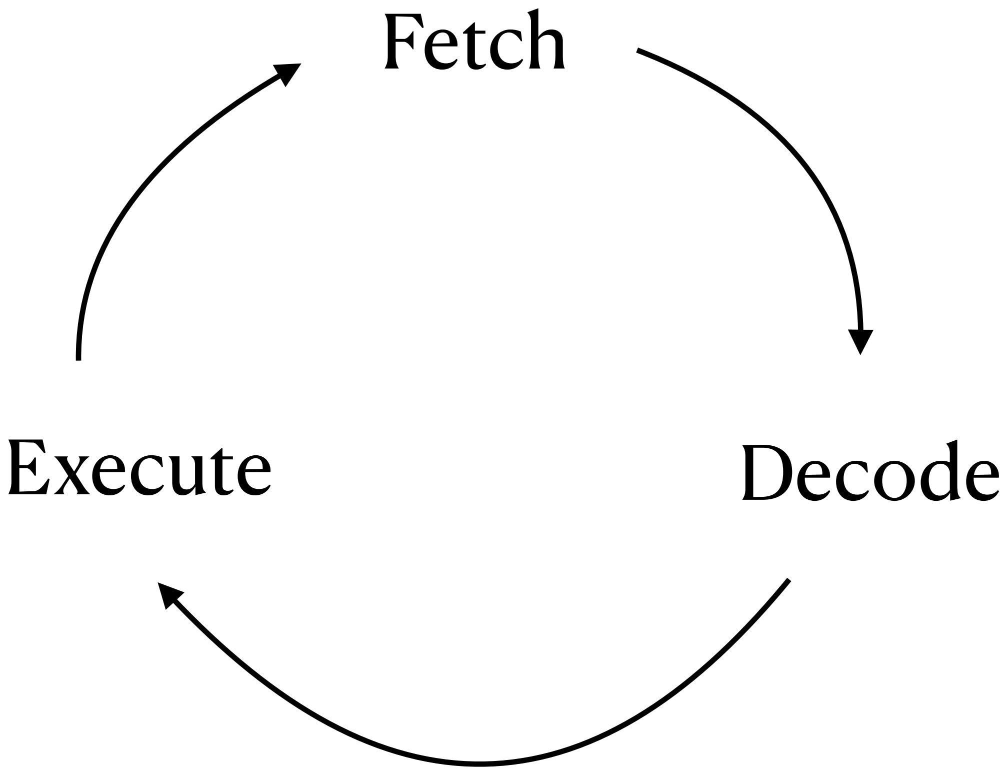


**Figure 1.11** General structure of a clustered system.

# What happens when a program runs?



Fetch

Decode

Execute

# What is OS?

- User view: making sure the system operates correctly and efficiently in an easy-to-use manner.
- System view:
  - resource manager/allocator
  - control program
- OS as a kernel (the one program running at all times on the computer)

# Three key ideas

- Virtualization
- Concurrency
- Persistence

# Virtualization

# Virtualization

- OS takes a physical resource (such as the processor, or memory, or a disk) and transforms it into a more general, powerful, and easy-to-use virtual form of itself
- provides some interfaces (APIs)
  - System calls (standard library to applications)

# Virtualizing the CPU

cpu.c

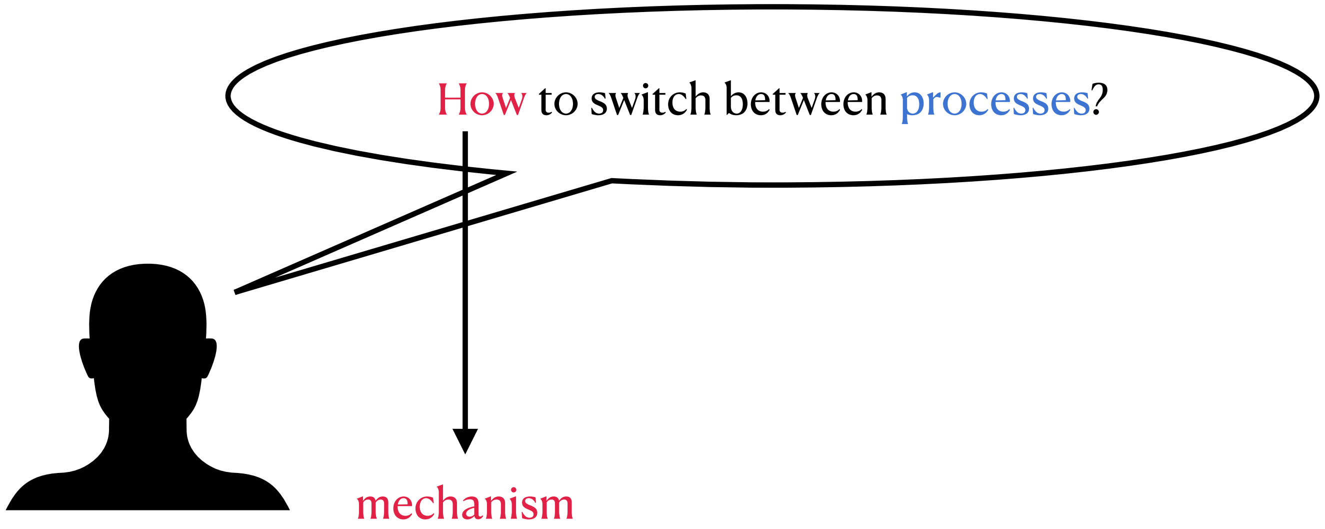
```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[])
{
    if (argc != 2) {
        fprintf(stderr, "usage: cpu <string>\n");
        exit(1);
    }
    char *str = argv[1];

    while (1) {
        printf("%s\n", str);
        Spin(1);
    }
    return 0;
}
```

```
os@VM$ gcc -o cpu cpu.c -Wall
os@VM$ ./cpu
usage: cpu <string>
os@VM$ ./cpu testing
testing
testing
testing
testing
^C
```

# Run multiple programs at once

```
os@VM$ ./cpu "A" & ./cpu "B" & ./cpu "C" & ./cpu "D" &
[1] 6596
[2] 6597
[3] 6598
[4] 6599
A
os@VM$ D
C
B
C
D
A
B
C
D
A
B
C
A
D
B
kill $(jobs -p)
[1]   Terminated                 ./cpu "A"
[2]   Terminated                 ./cpu "B"
[3]-  Terminated                 ./cpu "C"
[4]+  Terminated                 ./cpu "D"
```

Illusion that the system has a very large number of virtual CPUs

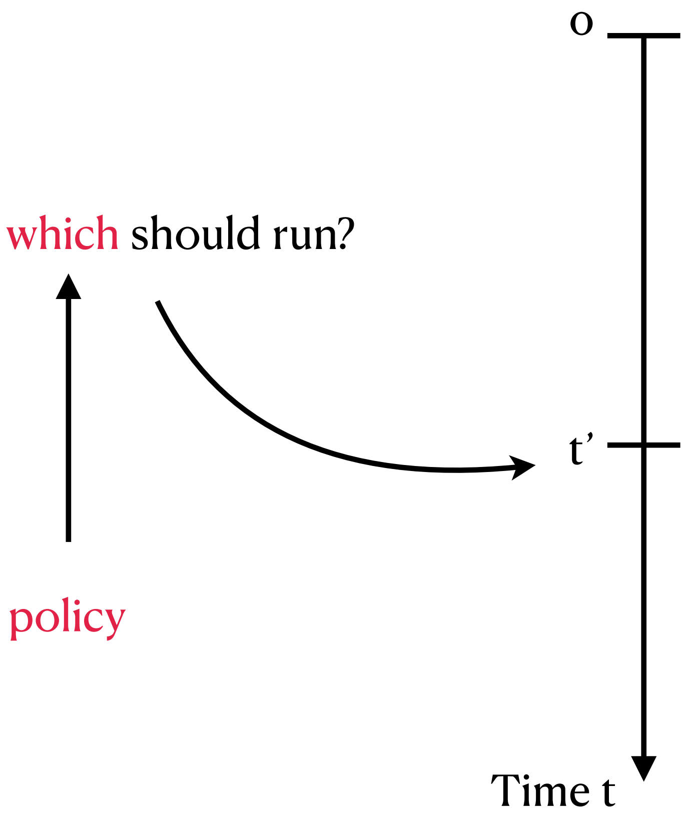


How to switch between processes?

mechanism

# Run multiple programs at once

```
os@VM$ ./cpu "A" & ./cpu "B" & ./cpu "C" & ./cpu "D" &
[1] 6596
[2] 6597
[3] 6598
[4] 6599
A
os@VM$ D
C
B
C
D
A
B
C
D
A
B
C
A
D
B
kill $(jobs -p)
[1]   Terminated              ./cpu "A"
[2]   Terminated              ./cpu "B"
[3]-  Terminated              ./cpu "C"
[4]+  Terminated              ./cpu "D"
```



# Virtualizing Memory

mem.c

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include "common.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: mem <value>\n");
        exit(1);
    }
    int *p;
    p = malloc(sizeof(int));
    assert(p != NULL);
    printf("(%d) addr pointed to by p: %p\n", (int) getpid(), p);
    *p = atoi(argv[1]); // assign value to addr stored in p
    while (1) {
        Spin(1);
        *p = *p + 1;
        printf("(%d) value of p: %d\n", getpid(), *p);
    }
    return 0;
}
```

```
os@VM$ gcc -o mem mem.c -Wall
os@VM$ ./mem 5
(6985) addr pointed to by p: 0x55a3492fa2a0
(6985) value of p: 6
(6985) value of p: 7
(6985) value of p: 8
(6985) value of p: 9
(6985) value of p: 10
(6985) value of p: 11
^C
```

```
os@VM$ ./mem 0 & ./mem 0 &
[5] 3927
[6] 3928
(3927) addr pointed to by p: 0x5586b2a432a0
(3928) addr pointed to by p: 0x5653bb05d2a0
(3927) value of p: 1
(3928) value of p: 1
(3927) value of p: 2
(3928) value of p: 2
(3927) value of p: 3
(3928) value of p: 3
kill $(jobs -p)
```

# Virtualization Summary

- OS as a resource manager
    - Share CPU —> virtualizing CPU
    - Share memory —> virtualizing memory

# Concurrency

# Concurrency

- Problems arise when working on many things at once (i.e., concurrently)
- Multi-threaded programs exhibit the same problems

threads.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "common.h"
#include "common_threads.h"

volatile int counter = 0;
int loops;

void *worker(void *arg) {
    int i;
    for (i = 0; i < loops; i++) {
        counter++;
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: threads <loops>\n");
        exit(1);
    }
    loops = atoi(argv[1]);
    pthread_t p1, p2;
    printf("Initial value : %d\n", counter);
    Pthread_create(&p1, NULL, worker, NULL);
    Pthread_create(&p2, NULL, worker, NULL);
    Pthread_join(p1, NULL);
    Pthread_join(p2, NULL);
    printf("Final value   : %d\n", counter);
    return 0;
}
```

```
os@VM$ gcc -o 2thread threads.c -Wall -pthread
os@VM$ ./thread 1000
Initial value : 0
Final value   : 1000
os@VM$ ./1thread 1000
Initial value : 0
Final value   : 1000
os@VM$ ./2thread 1000
Initial value : 0
Final value   : 2000
os@VM$ ./2thread 10000
Initial value : 0
Final value   : 20000
os@VM$ ./2thread 100000
Initial value : 0
Final value   : 193888
```

# Persistence

# Persistence

- No virtualized disk for each application
- Handling problems with journaling or copy-on-write
- Topics: I/O devices, disks, RAIDs, file systems

io.c

```c
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <string.h>

int main(int argc, char *argv[]) {
    int fd = open("/tmp/file", O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    assert(fd >= 0);
    char buffer[20];
    sprintf(buffer, "hello world\n");
    int rc = write(fd, buffer, strlen(buffer));
    assert(rc == (strlen(buffer)));
    fsync(fd);
    close(fd);
    return 0;
}
```

```
os@VM$ gcc -o io io.c -Wall
os@VM$ cat /tmp/file
hello world
```

# Design Goals

- Build up some abstraction
- Performance: minimize the overheads of the OS
- Provide protection between applications, & between OS and applications
- Reliability
- Security
- Mobility