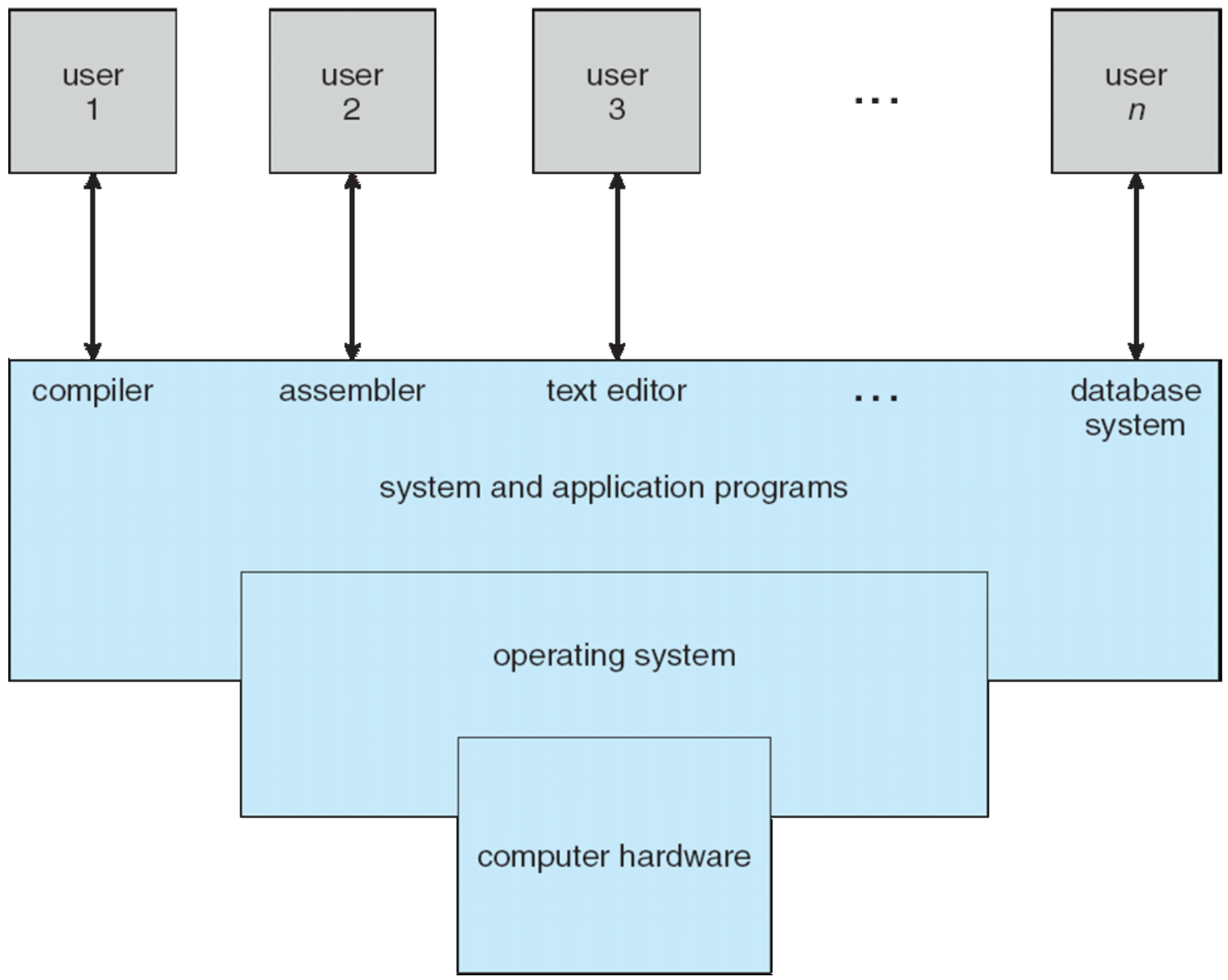
Review

Yajin Zhou (http://yajin.org)

Zhejiang University

Four Components of a Computer System



What Operating Systems Do

- OS is a **resource allocator**
  - it manages all resources
  - it decides between conflicting requests for efficient and fair resource sharing
- OS is a **control program**
  - it controls program execution to prevent **errors** and **improper use** of system

Interrupts and Traps

- Interrupt transfers control to the interrupt service routine
  - **interrupt vector:** a table containing addresses of all the service routines
  - incoming interrupts are disabled while serving another interrupt to prevent a lost interrupt
    - **interrupt handler** must save the (interrupted) execution states
- A **trap** is a software-generated interrupt, caused either by an error or a user request
  - an **interrupt** is asynchronous; a **trap** is synchronous
  - e.g., system call, divided-by-zero exception, general protection exception…
- Operating systems are usually **interrupt-driven**

I/O: from System Call to Devices, and Back

- A program uses a **system call** to access system resources
  - e.g., files, network
- Operating system converts it to device access and issues I/O requests
  - I/O requests are sent to the device driver, then to the controller
    - e.g., read disk blocks, send/receive packets…
- OS puts the program to wait (**synchronous I/O**) or returns to it without waiting (**asynchronous I/O**)
  - OS may switches to another program when the requester is waiting - scheduling
- I/O completes and the controller interrupts the OS
- OS processes the I/O, and then wakes up the program (synchronous I/O) or send its a signal (asynchronous I/O)

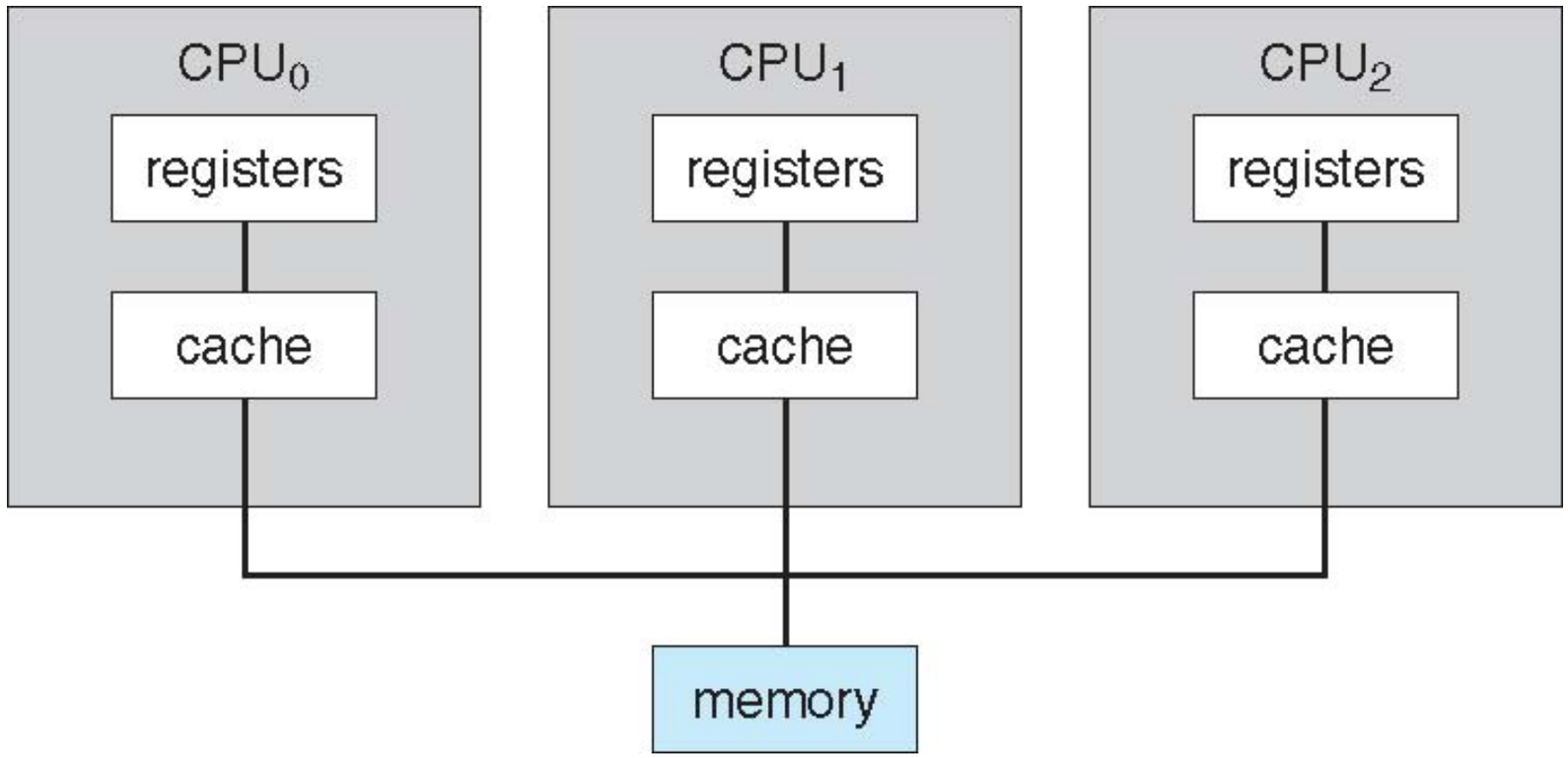
Computer-System Architecture

- Categorized according to the number of general-purpose processors
  - Single-Processor
  - Multiple-Processor

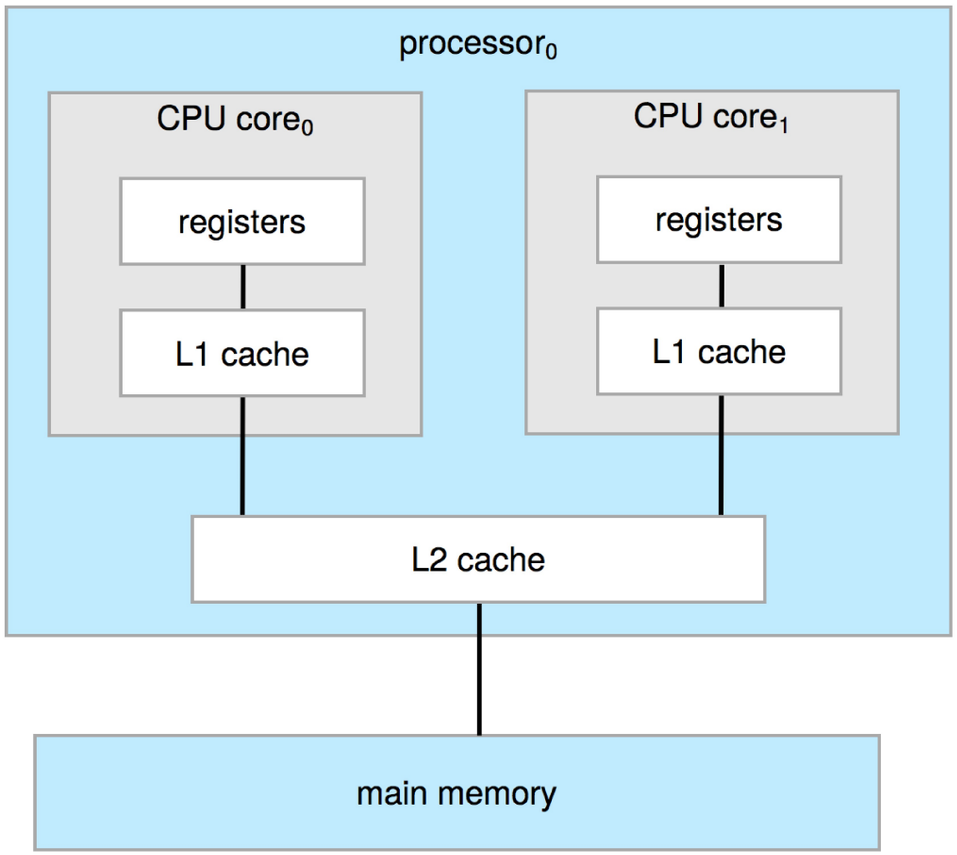
Multiprocessor Systems

- Most **old systems** have one single general-purpose processor
  - e.g., smartphone, PC, server, mainframe
  - most systems also have special-purpose processors as well
- Multiprocessor systems have grown in use and importance
  - also known as parallel systems, tightly-coupled systems
  - advantages: increased throughput, economy of scale, increased reliability -- graceful degradation or fault tolerance
  - two types: **asymmetric multiprocessing** and **symmetric multiprocessing (SMP)**

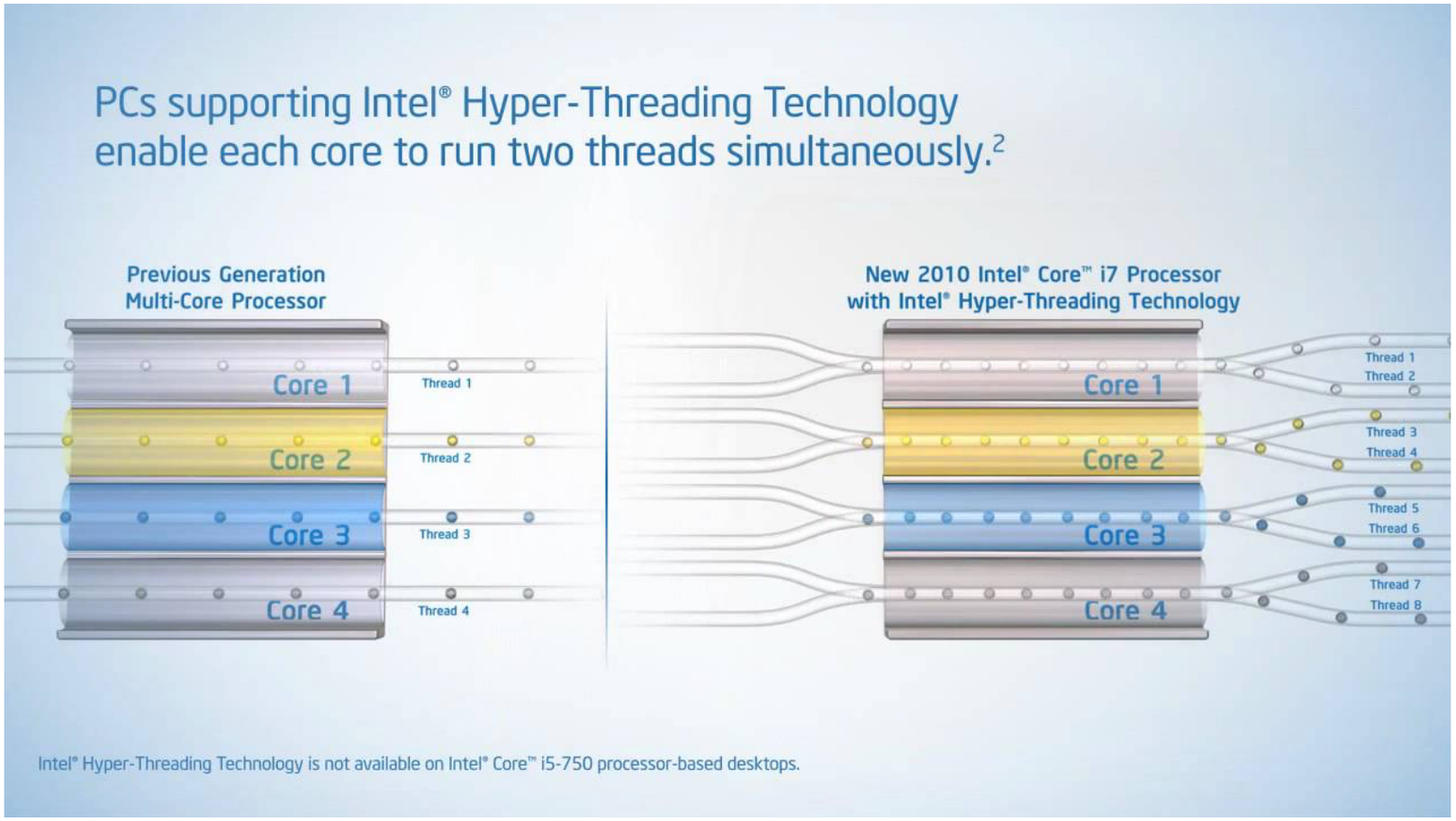
Symmetric Multiprocessing Architecture



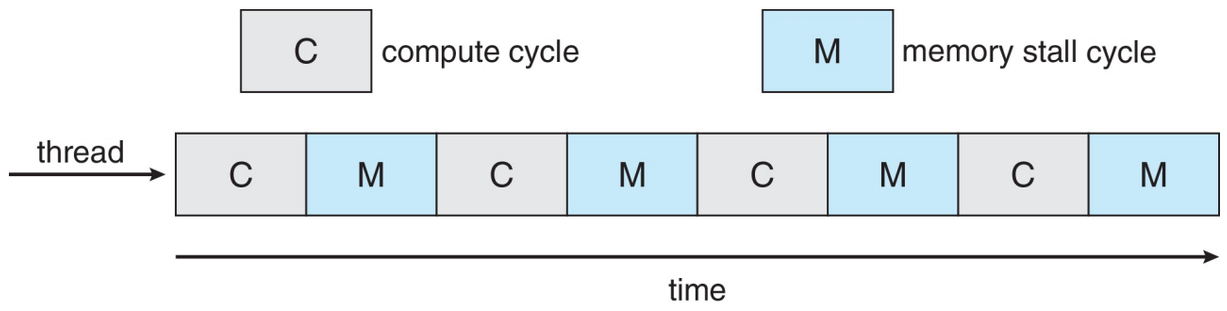
Dual Core

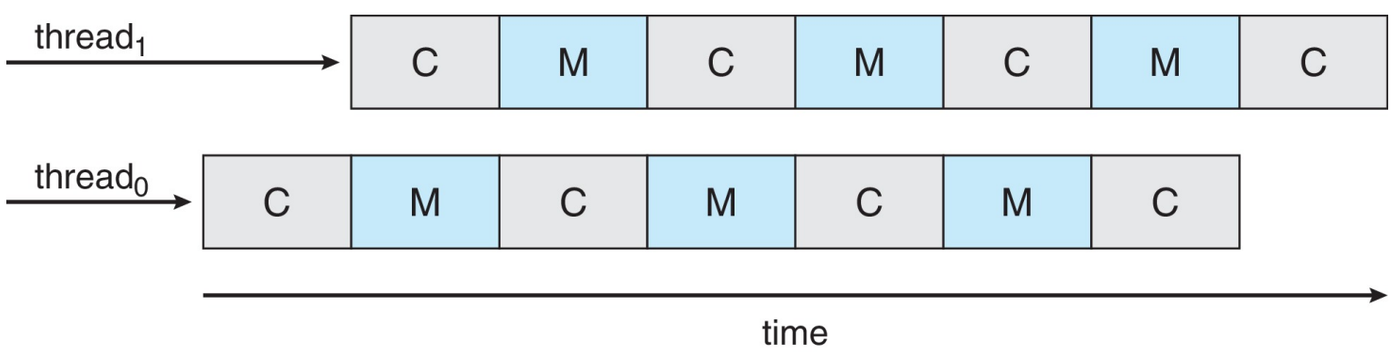

processor0
CPU core0
registers
L1 cache
CPU core1
registers
L1 cache
L2 cache
main memory

Chip Multithreading (CMT)



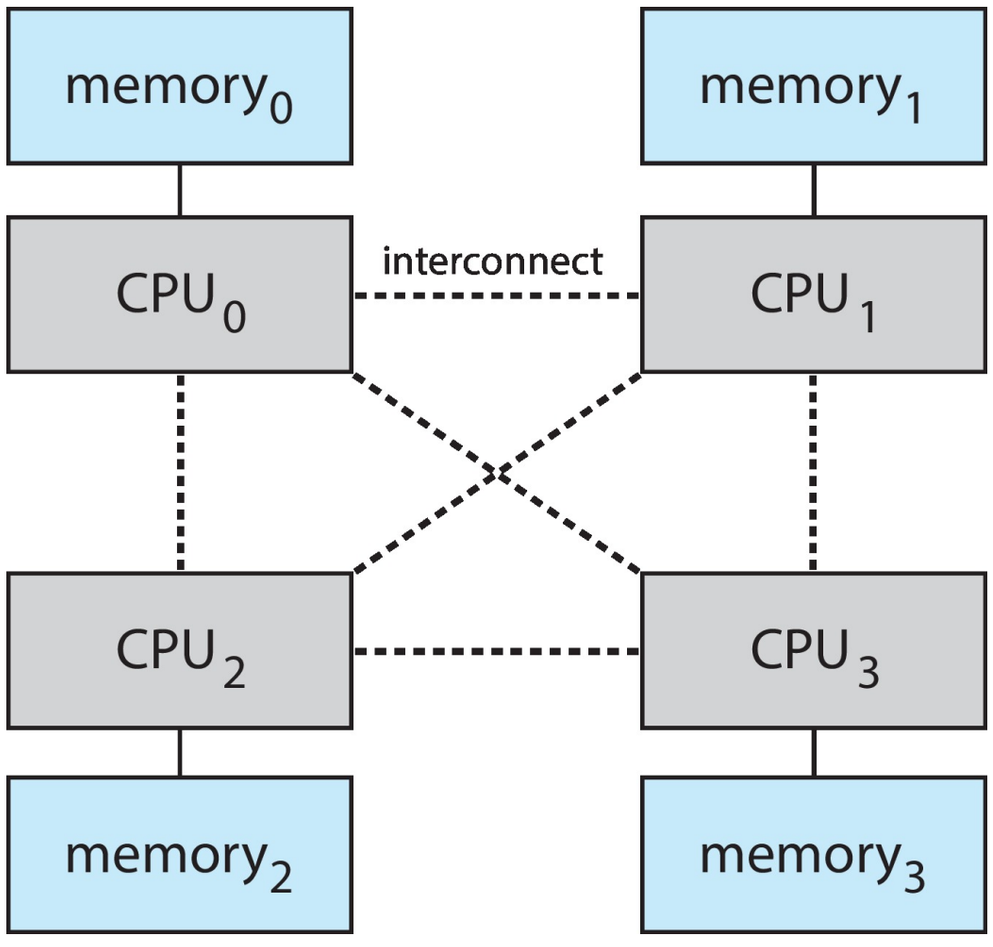
Multiple-Processor Scheduling: CMT





NUMA

- Non-Uniform Memory Access System
  - Access local memory is fast, scale well



Clustered Systems

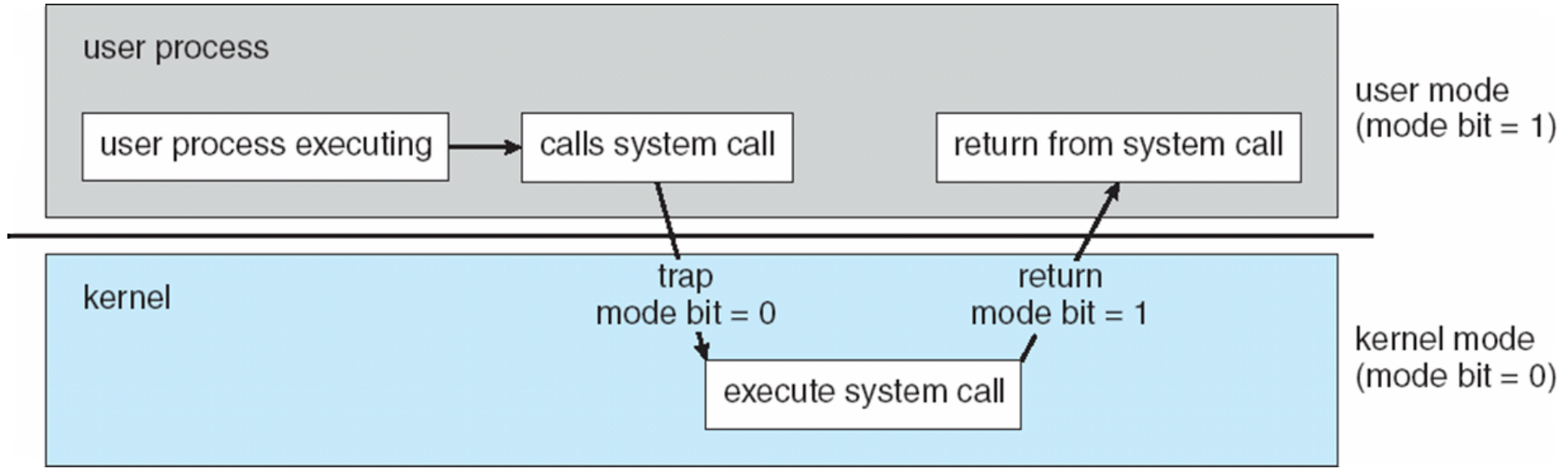
- Multiple systems work together ***through high-speed network***
  - usually sharing storage via a storage-area network (SAN)
- Clusters provide a high-availability service that can survive failures
  - **asymmetric** clustering has one machine in hot-standby mode
  - **symmetric** clustering has multiple nodes running applications, monitoring each other
- Some clusters are designed for high-performance computing (HPC)
  - applications must be written to use parallelization

Dual-mode operation

- Operating system is usually interrupt-driven (why?)
  - Efficiency, regain control (timer interrupt)
- **Dual-mode operation** allows OS to protect itself and other system components
  - **user mode** and **kernel mode (or other names)**
  - a **mode** bit distinguishes when CPU is running user code or kernel code
  - some instructions designated as **privileged**, only executable in kernel
  - **system call** changes mode to kernel, return from call resets it to user

Transition between Modes

- **System calls, exceptions, interrupts** cause transitions between kernel/user modes



Resource Management: Process Management

- A process is **a program in execution**
  - program is a ***passive*** entity, process is an ***active*** entity
  - a system has many processes running concurrently
- Process needs resources to accomplish its task
  - OS reclaims all reusable resources upon process termination
  - e.g., CPU, memory, I/O, files, initialization data

Resource Management: Memory Management

- Memory is the main storage directly accessible to CPU
  - data needs to be kept in memory before and after processing
  - all instructions should be in memory in order to execute
- Memory management determines what is in memory to **optimize CPU utilization** and **response time, provides a virtual view of memory for programmer**
- Memory management activities:
  - keeping track of which parts of memory are being used and by whom
  - deciding which processes and data to move into and out of memory
  - allocating and deallocating memory space as needed

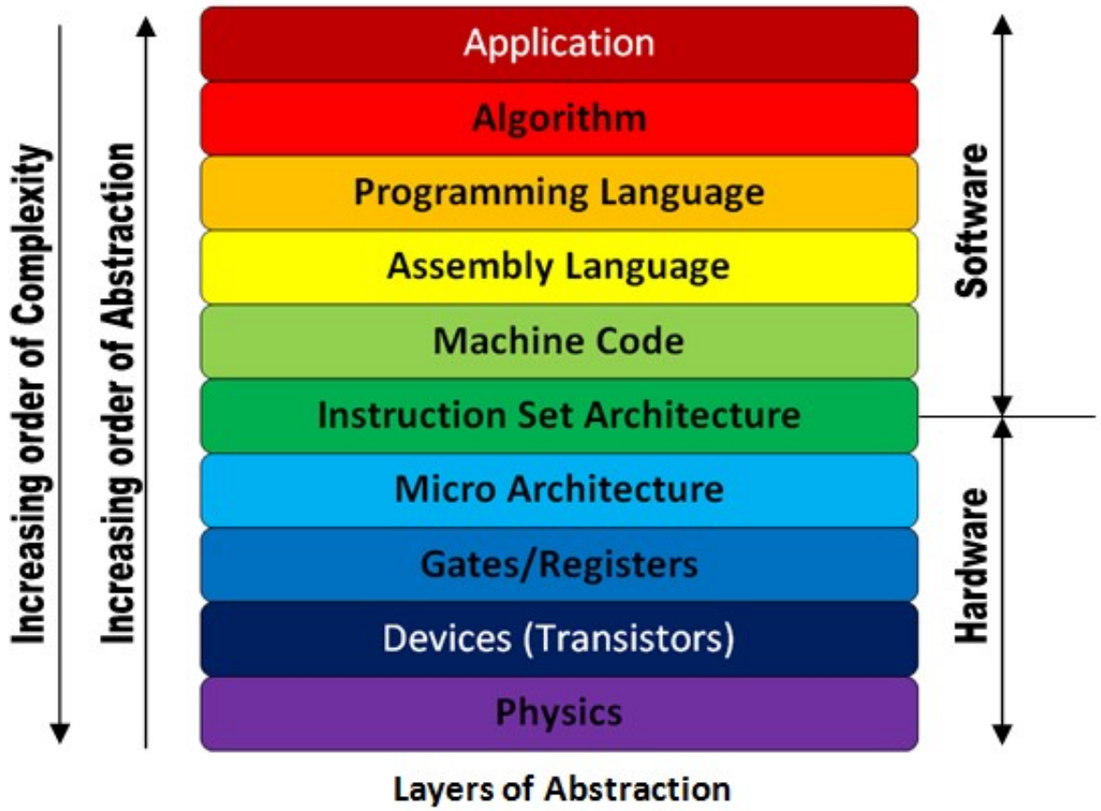
Resource Management

- File systems
  - OS provides a uniform, logical view of data storage
    - **file** is a logical storage unit that abstracts physical properties
      - files are usually organized into **directories**
        - **access control** determines who can access the file
- Mass-Storage Management
  - free-space management
  - storage allocation
  - disk scheduling

Separate Policy and Mechanism

- Mechanism: **how** question about a system
  - How does an operating system performs a context switch
- Policy: **which** question
  - Which process should the process to be switched
- Advantages & Disadvantages

Abstraction



Abstractions are fundamental to everything we do in computer science. Abstraction makes it possible to write a large program by dividing it into small and understandable pieces, to write such a program in a high-level language like C without thinking about assembly, to write code in assembly without thinking about logic gates, and to build a processor out of gates without thinking too much about transistors.

Operating System Services (User/Programmer-Visible)

- **User interface**
  - most operating systems have a user interface (UI).
  - e.g., command-Line (CLI), graphics user interface (GUI), or batch
- **Program execution: from program to process**
  - load and execute an program in the memory
  - end execution, either normally or abnormally
- **I/O operations**
  - a running program may require I/O such as file or I/O device
- **File-system manipulation**
  - read, write, create and delete files and directories
  - search or list files and directories
  - permission management

```
  13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
  14 root      rt   0       0      0      0 S   0.0  0.0   0:00.13 watchdog/1
  15 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/1
  16 root      20   0       0      0      0 S   0.0  0.0   0:00.11 ksoftirqd/1
  18 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:+
  19 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kdevtmpfs
  20 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 netns
os@os:~$
os@os:~$
os@os:~$
os@os:~$
os@os:~$
os@os:~$
os@os:~$
os@os:~$
os@os:~$
os@os:~$
os@os:~$
os@os:~$
os@os:~$ ls
Desktop    Downloads  Pictures  Templates  examples.desktop
Documents  Music      Public    Videos     os2018fall
os@os:~$ pwd
/home/os
os@os:~$
```

Operating System Services (User-Visible)

- **Communications**
  - processes exchange information, on the same system or over a network
  - via shared memory or through message passing

- **Error detection**
  - OS needs to be constantly aware of possible errors
  - errors in CPU, memory, I/O devices, programs
  - it should take appropriate actions to ensure correctness and consistency

Operating System Services (System View)

- **Resource allocation**
  - allocate resources for multiple users or multiple jobs running concurrently
  - many types of resources: CPU, memory, file, I/O devices
- **Accounting/Logging**
  - to keep track of which users use how much and what kinds of resources
- **Protection and security**
  - protection provides a mechanism to control access to system resources
    - access control: control access to resources
    - isolation: processes should not interfere with each other
  - security authenticates users and prevent invalid access to I/O devices
    - a chain is only as strong as its weakest link
  - protection is the **mechanism**, security towards the **policy**

```
top - 01:25:31 up 14:16,  3 users,  load average: 0.00, 0.00, 0.00
Tasks:  98 total,   1 running,  97 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.2 us,  0.0 sy,  0.0 ni, 99.8 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  1003772 total,   217452 free,    69248 used,   717072 buff/cache
KiB Swap:  1046524 total,  1046012 free,      512 used.   749832 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
26713 root      20   0       0      0      0 S   0.3  0.0   0:00.01 kworker/u6+
    1 root      20   0  119600   5076   3336 S   0.0  0.5   0:03.39 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.01 kthreadd
    4 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:+
    6 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 mm_percpu_+
    7 root      20   0       0      0      0 S   0.0  0.0   0:00.04 ksoftirqd/0
    8 root      20   0       0      0      0 S   0.0  0.0   0:01.02 rcu_sched
    9 root      20   0       0      0      0 S   0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/0
   11 root      rt   0       0      0      0 S   0.0  0.0   0:00.13 watchdog/0
   12 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/0
   13 root      20   0       0      0      0 S   0.0  0.0   0:00.00 cpuhp/1
   14 root      rt   0       0      0      0 S   0.0  0.0   0:00.13 watchdog/1
   15 root      rt   0       0      0      0 S   0.0  0.0   0:00.01 migration/1
   16 root      20   0       0      0      0 S   0.0  0.0   0:00.11 ksoftirqd/1
   18 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/1:+
   19 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kdevtmpfs
   20 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 netns
```

System Calls And API

- System call is **a programming interface to access the OS services**
  - Typically written in a high-level language (C or C++)
- Certain low level tasks are in assembly languages

EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t    read(int fd, void *buf, size_t count)
```

return value | function name | parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

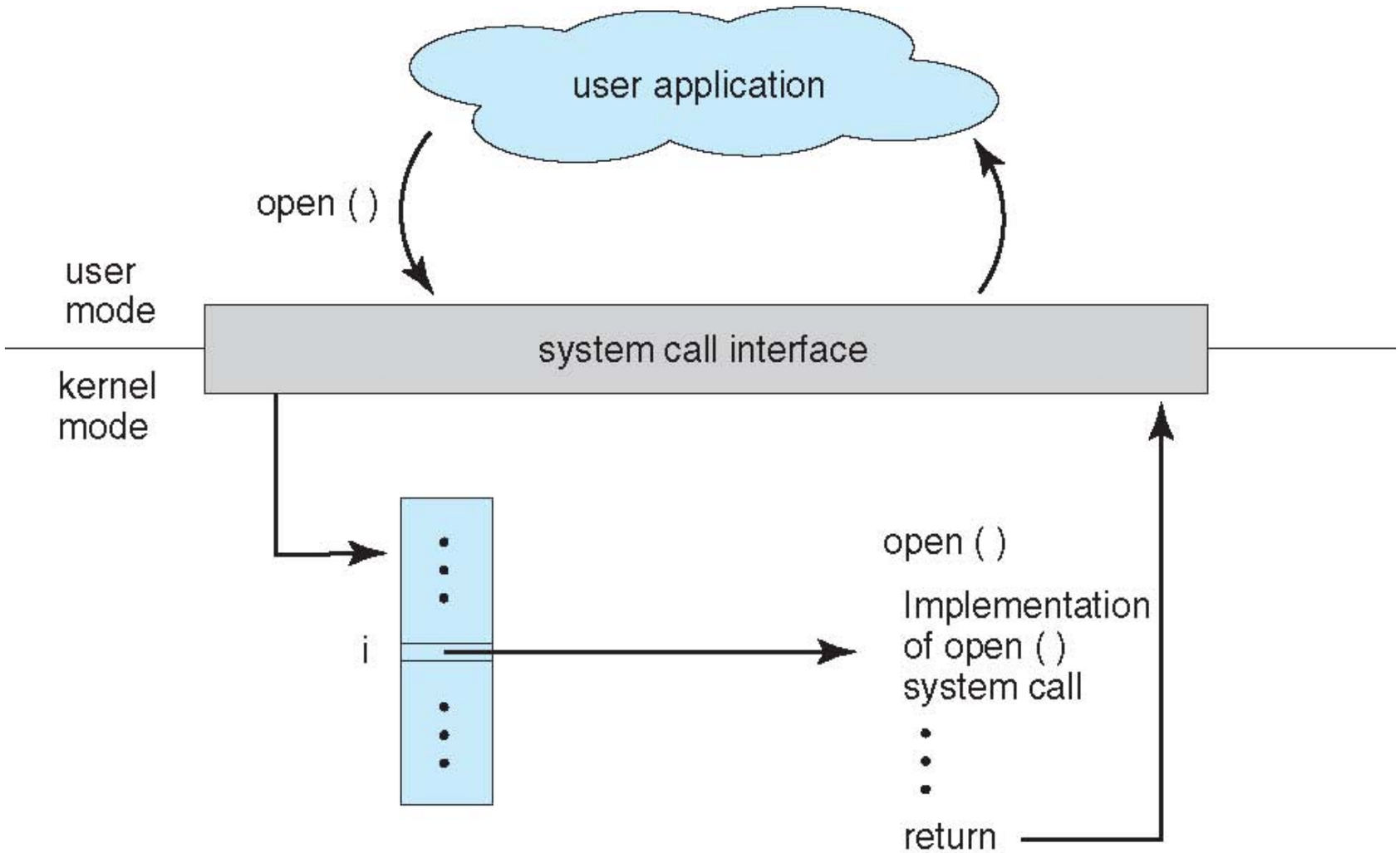
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

System Calls Implementation

- Typically, a **number** is associated with each system call
  - system-call interface maintains a table indexed by these numbers
  - e.g., Linux has around 340 system call (x86: 349, arm: 345)
- Kernel invokes intended system call and returns results
- User program needs to know nothing about syscall details
  - it just needs to use API (e.g., in libc) and understand what the API will do
  - most details of OS interface hidden from programmers by the API

API – System Call – OS Relationship



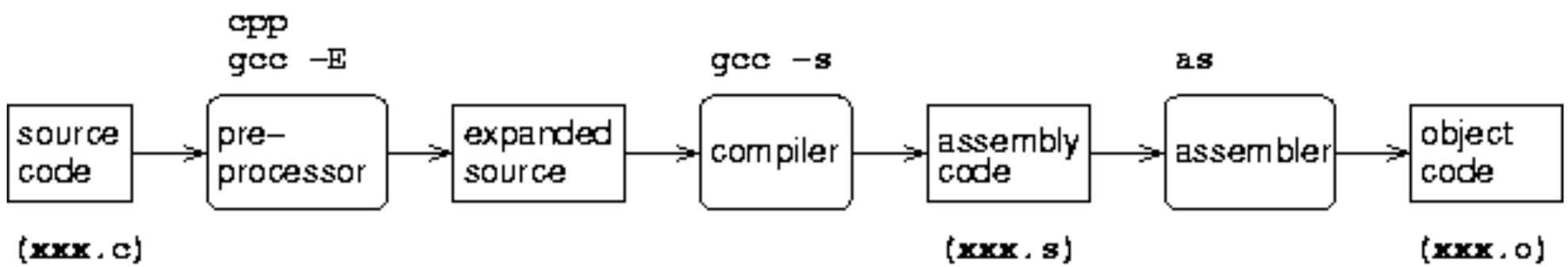
System Call Parameter Passing

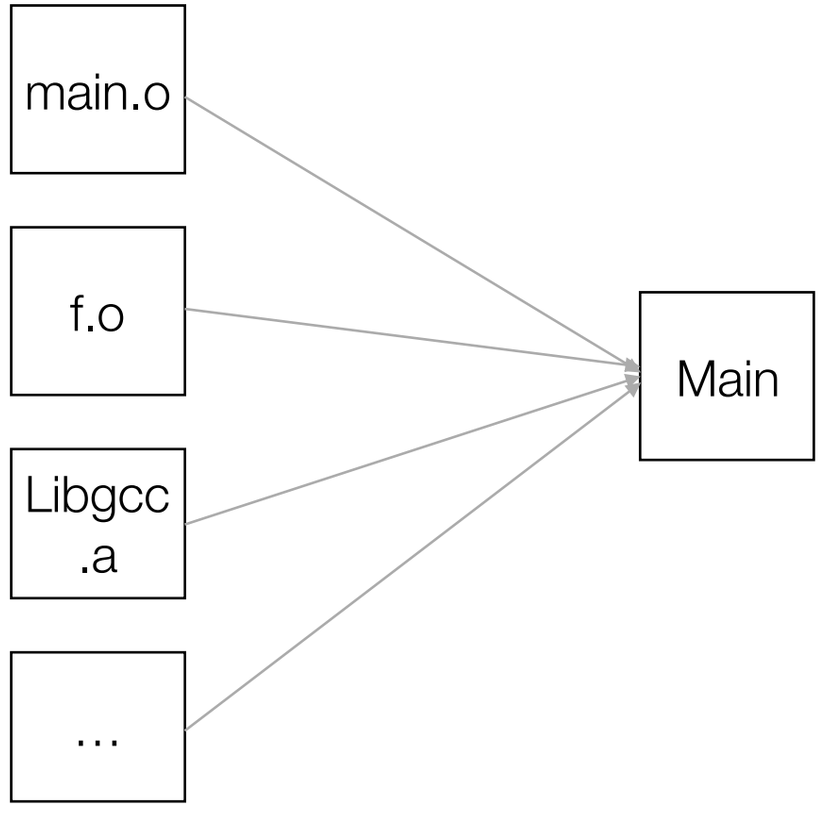
- Parameters are required besides the **system call number**
  - exact type and amount of information vary according to OS and call
- Three general methods to pass parameters to the OS
  - **Register**:
    - pass the parameters in registers
    - simple, but there may be more parameters than registers
  - **Block**:
    - parameters stored in a memory block (or table)
    - address of the block passed as a parameter in a register
    - taken by Linux and Solaris
  - **Stack**:
    - parameters placed, or pushed, onto the stack by the program
    - popped off the stack by the operating system
- Block and stack methods don’t limit number of parameters being passed

Types of System Calls

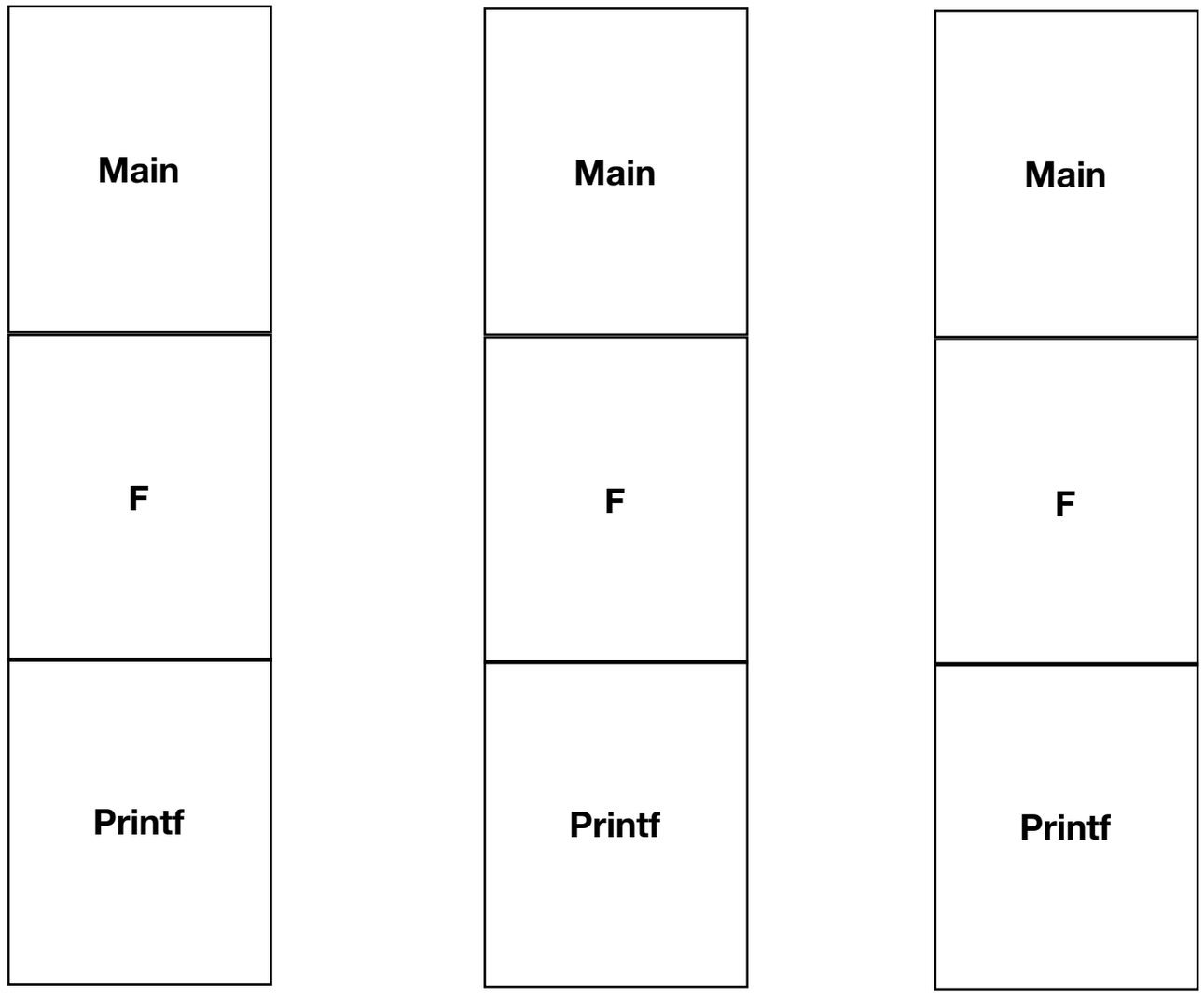
- Process control
- File management
- Device management
- Information maintenance
- Communications

Linkers & Loaders



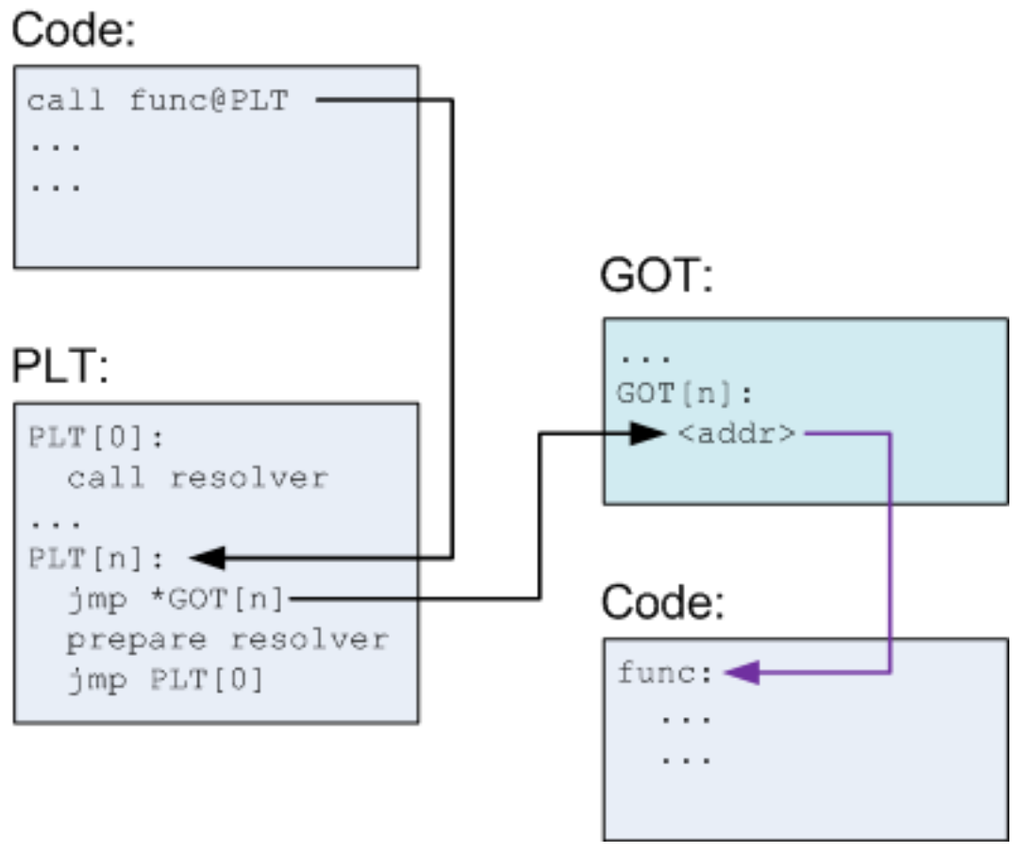


Static Linking: in memory



Dynamic linking

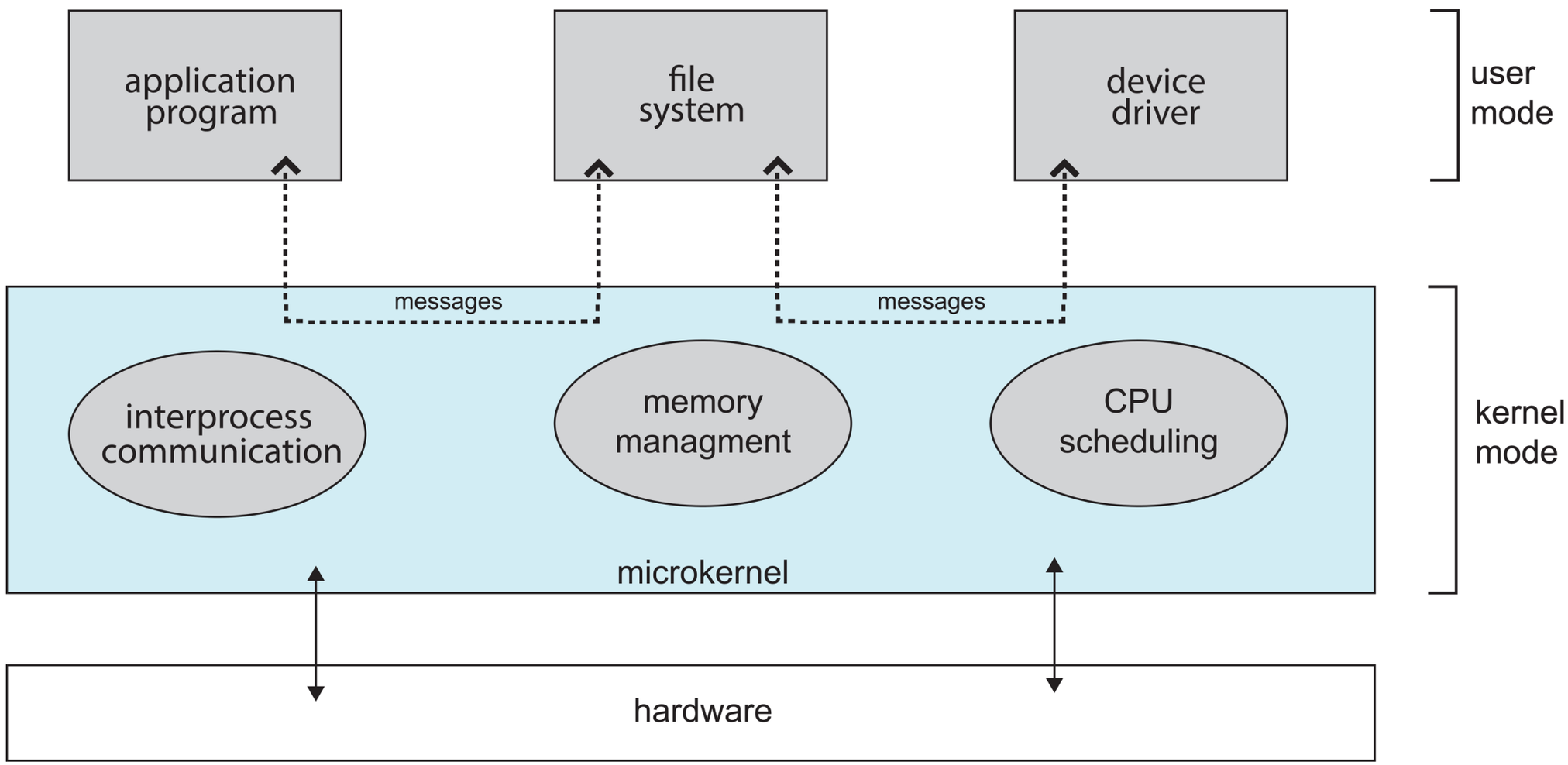
- Code -> PLT -> GOT
- Lazy binding



Operating System Structure

- Many structures:
  - **simple structure - MS-DOS**
  - **more complex -- UNIX**
  - **layered structure - an abstraction**
  - **microkernel system structure - L4**
  - **hybrid: Mach, Minix**
  - research system: **exokernel**

Microkernel System Structure



Linux kernel map

functionalities
layers

human interface | system | processing | memory | storage | networking

user space interfaces

system calls and system files

HI char devices

cdev_add
input_fops
video_fops
snd_fops
console_fops
sys_capset
fb_fops
sys_syslog

interfaces core

kernel/
System Call Interface
system files
linux/syscalls.h
linux/uaccess.h
/proc /sysfs /dev
copy_from_user
sysfs_ops
register_chrdev
sys_ioctl
cdev_add
/sys/class/
cdev_map
sys_epoll_create
cdev
sys_reboot
sys_init_module
kvm_dev_ioctl

processes

kernel/
fs/exec.c
kernel/signal.c
sys_fork
sys_kill
sys_vfork
sys_execve
sys_signal
sys_clone
do_sigaction
linux_binfmt
sys_gettimeofday
sys_futex
sys_time
sys_times
sys_nanosleep

memory access

mm/
sys_brk
sys_mmap
shm_vm_ops
sys_shmctl
sys_sysinfo
sys_shmat
si_meminfo
sys_mprotect
sys_mincore
/dev/mem
mem_fops
mmap_mem
/proc/meminfo
/proc/self/maps
sys_msync

files & directories access

fs/
sys_chroot
sys_pivot_root
sys_mkdir
sys_chdir
sys_open
sys_read
sys_tee
sys_poll
sys_write
sys_pipe
sys_select
do_path_lookup
sys_flock
sys_readv
sys_splice
sys_sendfile
sys_getdents
iovec
sys_swapon
alloc_file
sys_chown
sys_chmod
fd
sys_newfstat
notify_change
sys_fsync
fsnotify_change
fget
sys_mount
inode_setattr
sys_sync
sys_sysfs

sockets access

net/
sys_socketcall
sys_connect
sys_accept
sys_bind
sys_listen
sys_sendmsg
sys_recvmsg
sys_setsockopt
sys_socket
/proc/net/
tcp4_seq_show
sg_proc_seq_show_dev
rt_cache_seq_show
sock_ioctl

virtual

security

security/
security_capset
may_open
linux/security.h
security_socket_create
inode_permission
security_inode_create
security_ops
selinux_ops

Device Model

drivers/base/
driver_init
kobject
linux/kobject.h
bus_register
kset
linux/device.h
bus_type
device_create
device
class
device_type
driver_register
device_driver
probe
kvm

threads

schedule_work
create_workqueue
INIT_WORK
queue_work
work_struct
workqueue_struct
kthread_create
kernel_thread
current
thread_info
do_fork

virtual memory

vmalloc_init
find_vma_prepare
vmalloc
vfree
vmlist
vm_struct
virt_to_page

Virtual File System

file
vfs_fsync
file_systems
vfs_fstat
vfs_read
vfs_write
vfs_getattr
vfs_create
fsnotify
inode
inode_operations
file_operations
file_system_type
get_sb
super_block
do_splice_direct
generic_file_aio_read
ramfs_fs_type

protocol families

inet_init
__sock_create
socket
inet_family_ops
inet_create
unix_family_ops
proto_ops
inet_dgram_ops
inet_stream_ops
socket_file_ops

bridges

cross-functional modules

debugging

sys_ptrace
log_buf
register_kprobe
printk
handle_sysrq
oprofile_start
kgdb_breakpoint
oprofile_init

synchronization

lock_kernel
mutex_lock_interruptible
mutex_unlock
kernel_flag
wait_for_completion
complete
mutex
owner
add_timer
timer_list
run_timer_softirq
down_interruptible
up
semaphore
wait_event
wake_up
spin_lock_irqsave
msleep
spin_unlock_irqrestore

memory mapping

mm/mmap.c
do_mmap
do_mmap_pgoff
kmem_cache_alloc
vma_link
mm_struct
vm_area_struct

page cache

address_space
bdi_writeback_thread
do_writepages

swap

si_swapinfo
swap_info
kswapd
do_swap_page
wakeup_kswapd

networking storage

nfs_file_operations
smb_fs_type
cifs_file_ops
iscsi_tcp_transport

socket splice

sock_sendpage
tcp_sendpage
udp_sendpage
sock_splice_read
tcp_splice_read

logical

functions implementations

HI subsystems

oss
alsa
video_device
mousedev_handler
tty

system run

init/
kernel/
boot, shutdown
power management
init/main.c
start_kernel
mount_root
do_initcalls
run_init_process
kernel_restart
kernel_power_off
hibernate
machine_ops

Scheduler

kernel/sched.c
task_struct
schedule_timeout
schedule
setup_timer
process_timeout
activate_task
rq
context_switch

logical memory

physically mapped memory
mm_init
kmalloc
kfree
pgd_t
pmd_t
pte_t

logical file systems

ext4_file_operations
ext4_get_sb
ext4_readdir

protocols

proto
/proc/net/protocols
udp_prot
tcp_prot
udp_recvmsg
udp_sendmsg
tcp_recvmsg
tcp_sendmsg
tcp_v4_rcv
udp_rcv
tcp_transmit_skb
NF_HOOK
nf_hooks
ip_queue_xmit
ip_local_deliver
ip_push_pending_frames
ip_route_input
alloc_skb
ip_rcv
sk_buff
ip_output

device control

drivers/

abstract devices and HID class drivers

drivers/input/
drivers/media/
sound/
console
kbd
fb_ops
mousedev
drm_driver

generic HW access

request_region
pci_driver
request_mem_region
pci_register_driver
pci_request_regions
usb_driver
usb_submit_urb
usb_hcd_giveback_urb
ioremap
usb_hcd

interrupts core

request_irq
tasklet_struct
jiffies_64++
tasklet_action
do_timer
setup_irq
tick_periodic
timer_interrupt
do_softirq
do_IRQ
irq_desc
softirq_init

Page Allocator

/proc/slabinfo
mm/slob.c
mm/slub.c
mm/slab.c
__free_pages
kmem_cache
__free_one_page
kmem_cache_init
kmem_cache_alloc
__get_free_pages
page
__alloc_pages
vm_stat
totalram_pages
try_to_free_pages

block devices

block/
gendisk
block_device_operations
request_queue
init_scsi
scsi_device
scsi_driver
sd_fops
usb_storage_driver
usb_stor_host_template

network interface

linux/netdevice.h
netif_receive_skb
dev_queue_xmit
netif_rx
register_netdev
net_device
dev_ioctl
alloc_netdev_mq
ieee80211_alloc_hw
ether_setup
ieee80211_rx
netif_carrier_on
ieee80211_xmit
drivers/net/

hardware interfaces

drivers, registers and interrupts

HI peripherals device drivers

uvc_driver
vga_con
ac97_driver
atkbd_drv
i8042_driver
psmouse
drivers/media/video/

device access and bus drivers

writew
ehci_irq
readw
ehci_urb_enqueue
outw
inw
usb_hcd_irq
pci_read
pci_write

CPU specific

include/asm/
arch/x86/
setup_arch
x86_init
trap_init
early_trap_init
start_thread
native_init_IRQ
switch_to
/proc/interrupts
set_intr_gate
system_call
show_regs
interrupt
pt_regs
atomic_t
cli
sti

physical memory operations

arch/x86/mm/
mem_init
get_page_from_freelist
zonelist
zone
free_area
free_list
out_of_memory
die
num_physpages
do_page_fault

disk controller drivers

scsi_host_alloc
libata
Scsi_Host
ahci_pci_driver
aic94xx_init

network device drivers

usbnet_probe
ipw2100_pci_init_one
zd1201_probe
e1000_xmit_frame
e1000_intr

electronics

user peripherals

keyboard
camera
mouse
graphics card
audio

I/O

I/O mem
I/O
PCI controller
I/O ports
ACPI
USB controller

CPU

registers
APIC
interrupt controller

memory

RAM
DMA
MMU

disk controllers

SCSI
SATA

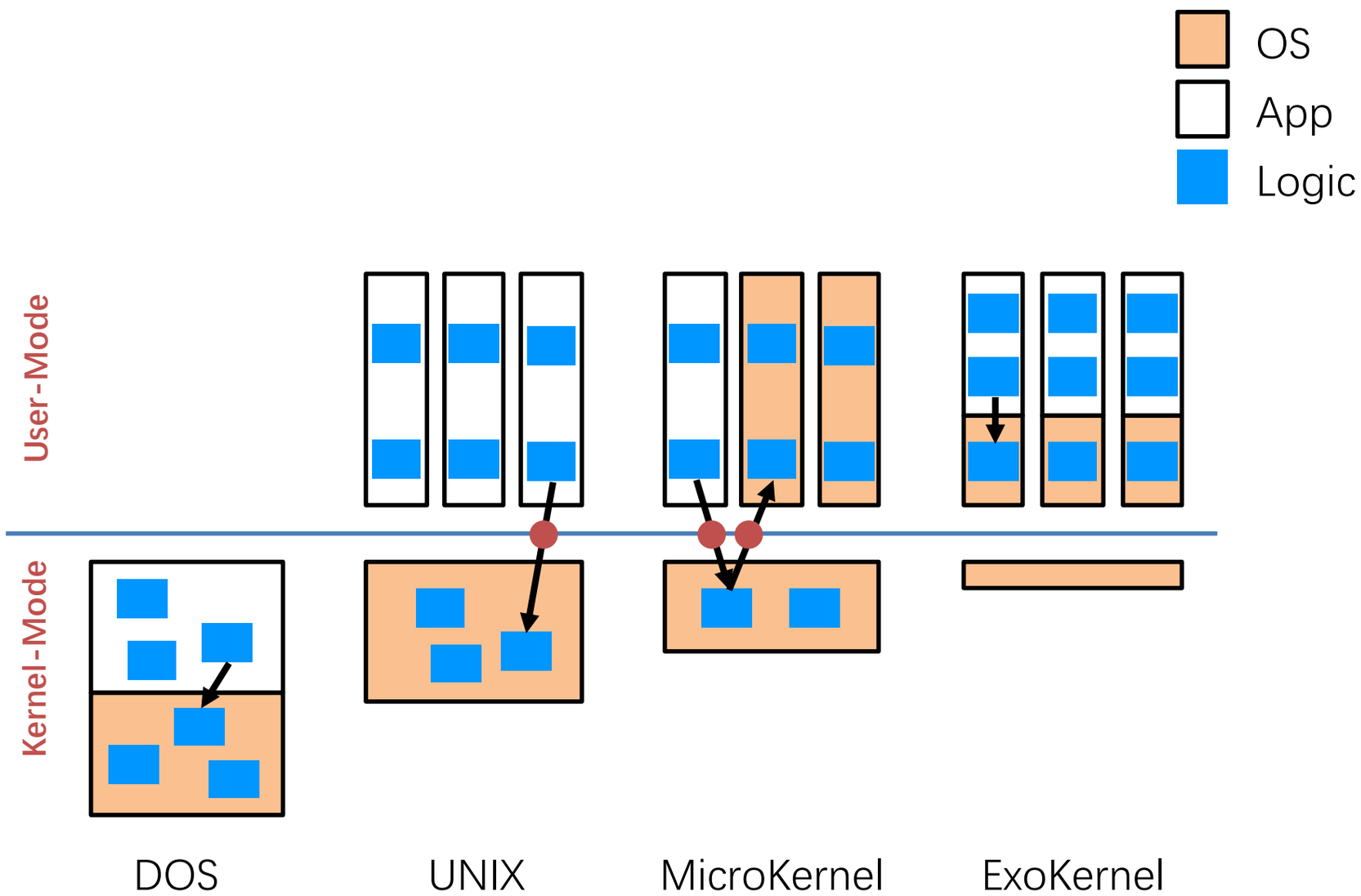
network controllers

Ethernet
WiFi

Modules

- Most modern operating systems implement **kernel modules**
  - uses **object-oriented** design pattern
  - each core component is separate, and has **clearly defined interfaces**
    - some are loadable as needed
- Overall, similar to layers but with more flexible
- Example: Linux, BSD, Solaris
  - http://www.makelinux.net/kernel_map/

Comparison


OS
App
Logic
User-Mode
Kernel-Mode
DOS
UNIX
MicroKernel
ExoKernel

Tracing

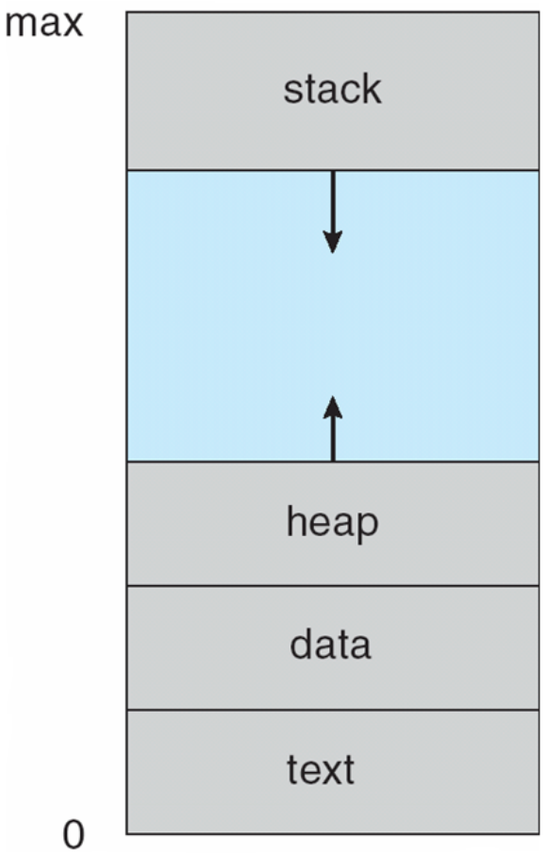
```
os@os:~/os2018fall/code/1_cpu$ strace ./cpu 'A'
execve("./cpu", ["./cpu", "A"], [/* 32 vars */]) = 0
```

```
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
brk(NULL)                               = 0xedd000
brk(0xefe000)                           = 0xefe000
write(1, "A\n", 2A
)                       = 2
write(1, "A\n", 2A
)                       = 2
write(1, "A\n", 2A
)                       = 2
^C--- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
strace: Process 26654 detached
```

Process Concept

- A process has multiple parts:
  - the program **code**, also called **text section**
  - runtime **CPU states**, including program counter, registers, etc
  - various types of memory:
    - **stack**: temporary data
      - e.g., function parameters, local variables, and *return addresses*
    - **data** section: global variables
    - **heap**: memory dynamically allocated during runtime
      - security: heap feng shui -> how to provide randomness
      - Further reading: FreeGuard: A Faster Secure Heap Allocator (CCS 17), Guarder: A Tunable Secure Allocator (USENIX Sec 18)

Process in Memory

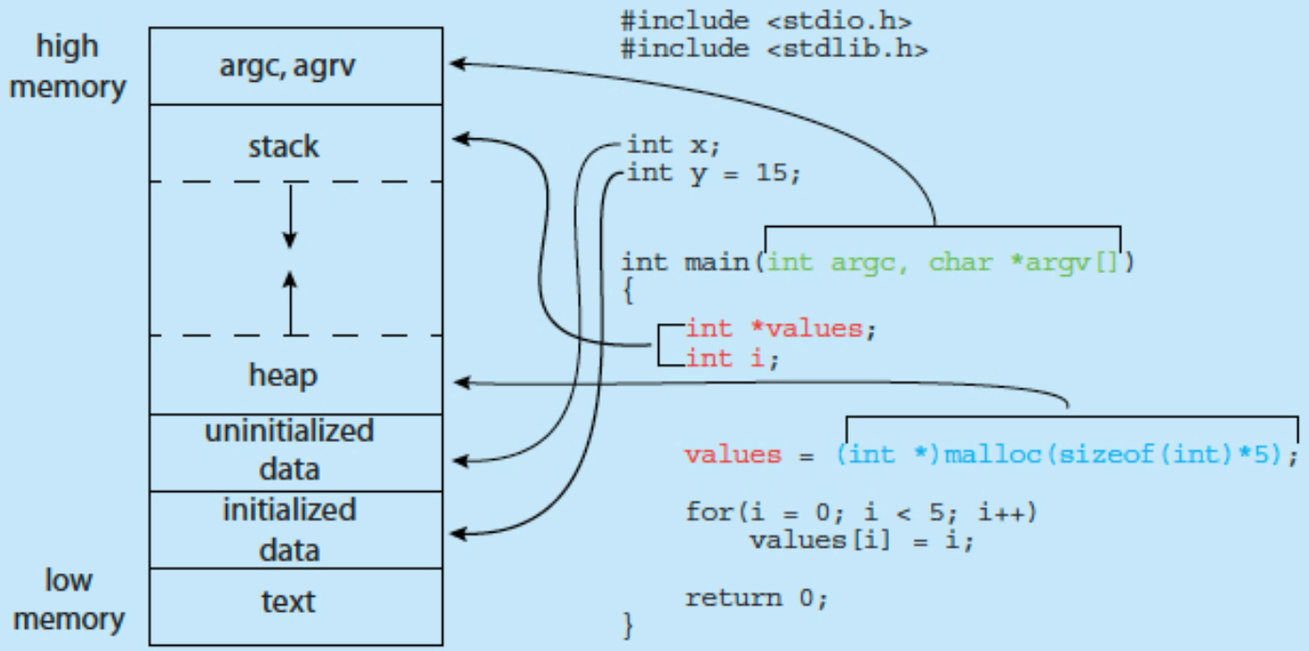



```
os@os:~$ cat /proc/self/maps
00400000-0040c000 r-xp 00000000 08:01 2752536                            /bin/cat
0060b000-0060c000 r--p 0000b000 08:01 2752536                            /bin/cat
0060c000-0060d000 rw-p 0000c000 08:01 2752536                            /bin/cat
0108d000-010ae000 rw-p 00000000 00:00 0                                  [heap]
7f3b4c98d000-7f3b4d34c000 r--p 00000000 08:01 3284766                    /usr/lib/locale/locale-archive
7f3b4d34c000-7f3b4d50c000 r-xp 00000000 08:01 2102132                    /lib/x86_64-linux-gnu/libc-2.23.so
7f3b4d50c000-7f3b4d70c000 ---p 001c0000 08:01 2102132                    /lib/x86_64-linux-gnu/libc-2.23.so
7f3b4d70c000-7f3b4d710000 r--p 001c0000 08:01 2102132                    /lib/x86_64-linux-gnu/libc-2.23.so
7f3b4d710000-7f3b4d712000 rw-p 001c4000 08:01 2102132                    /lib/x86_64-linux-gnu/libc-2.23.so
7f3b4d712000-7f3b4d716000 rw-p 00000000 00:00 0
7f3b4d716000-7f3b4d73c000 r-xp 00000000 08:01 2102104                    /lib/x86_64-linux-gnu/ld-2.23.so
7f3b4d900000-7f3b4d925000 rw-p 00000000 00:00 0
7f3b4d93b000-7f3b4d93c000 r--p 00025000 08:01 2102104                    /lib/x86_64-linux-gnu/ld-2.23.so
7f3b4d93c000-7f3b4d93d000 rw-p 00026000 08:01 2102104                    /lib/x86_64-linux-gnu/ld-2.23.so
7f3b4d93d000-7f3b4d93e000 rw-p 00000000 00:00 0
7ffff3ba3000-7ffff3bc4000 rw-p 00000000 00:00 0                          [stack]
7ffff3bcd000-7ffff3bd0000 r--p 00000000 00:00 0                          [vvar]
7ffff3bd0000-7ffff3bd2000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

MEMORY LAYOUT OF A C PROGRAM

The figure shown below illustrates the layout of a C program in memory, highlighting how the different sections of a process relate to an actual C program. This figure is similar to the general concept of a process in memory as shown in Figure 3.1, with a few differences:

- The global data section is divided into different sections for (a) initialized data and (b) uninitialized data.
- A separate section is provided for the `argc` and `argv` parameters passed to the `main()` function.

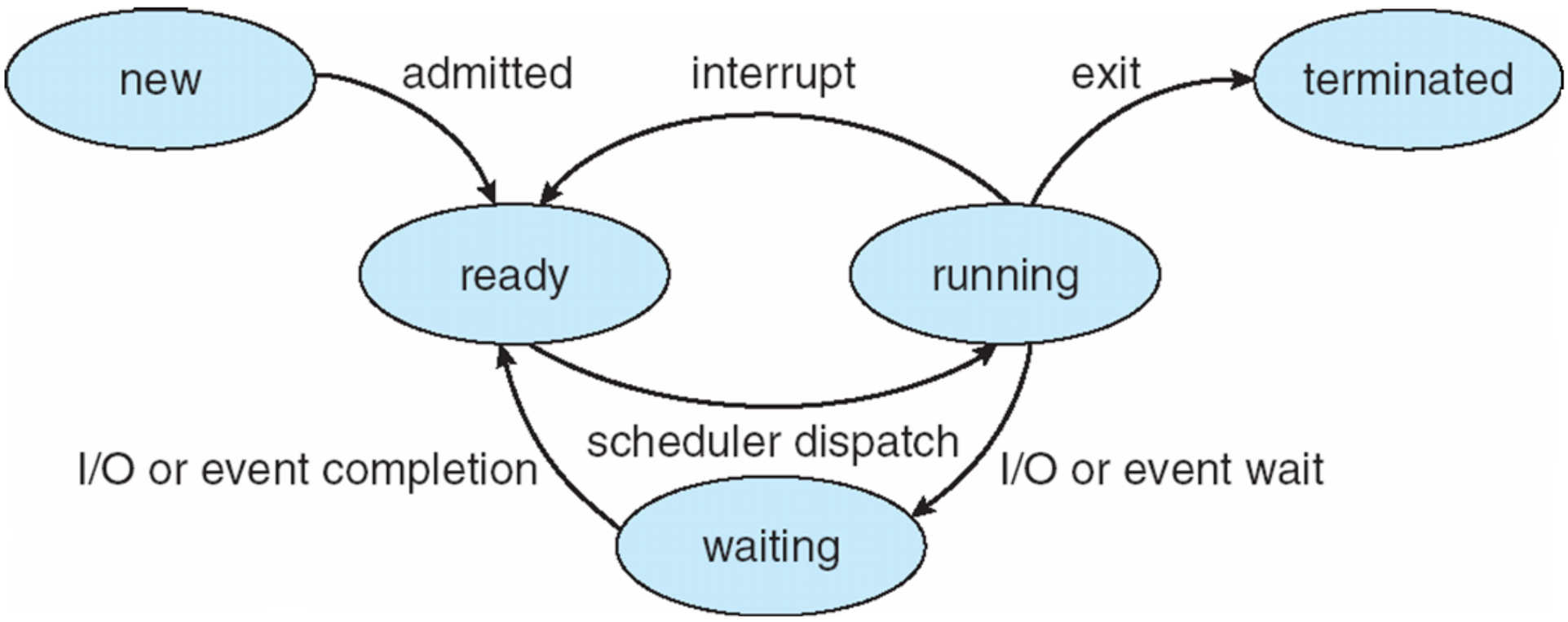


The GNU `size` command can be used to determine the size (in bytes) of some of these sections. Assuming the name of the executable file of the above C program is `memory`, the following is the output generated by entering the command `size memory`:

```
text    data    bss     dec     hex     filename
1158    284     8       1450    5aa     memory
```

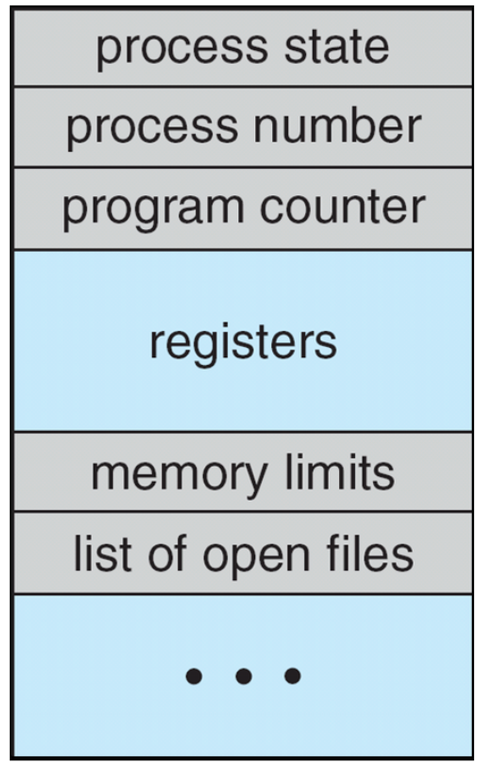
The `data` field refers to uninitialized data, and `bss` refers to initialized data. (`bss` is a historical term referring to *block started by symbol*.) The `dec` and `hex` values are the sum of the three sections represented in decimal and hexadecimal, respectively.

Process State



Process Control Block (PCB)

- In the kernel, each process is associated with a **process control block**
  - process number (pid)
  - process state
  - **program counter (PC)**
  - CPU registers
  - CPU scheduling information
  - memory-management data
  - accounting data
  - I/O status

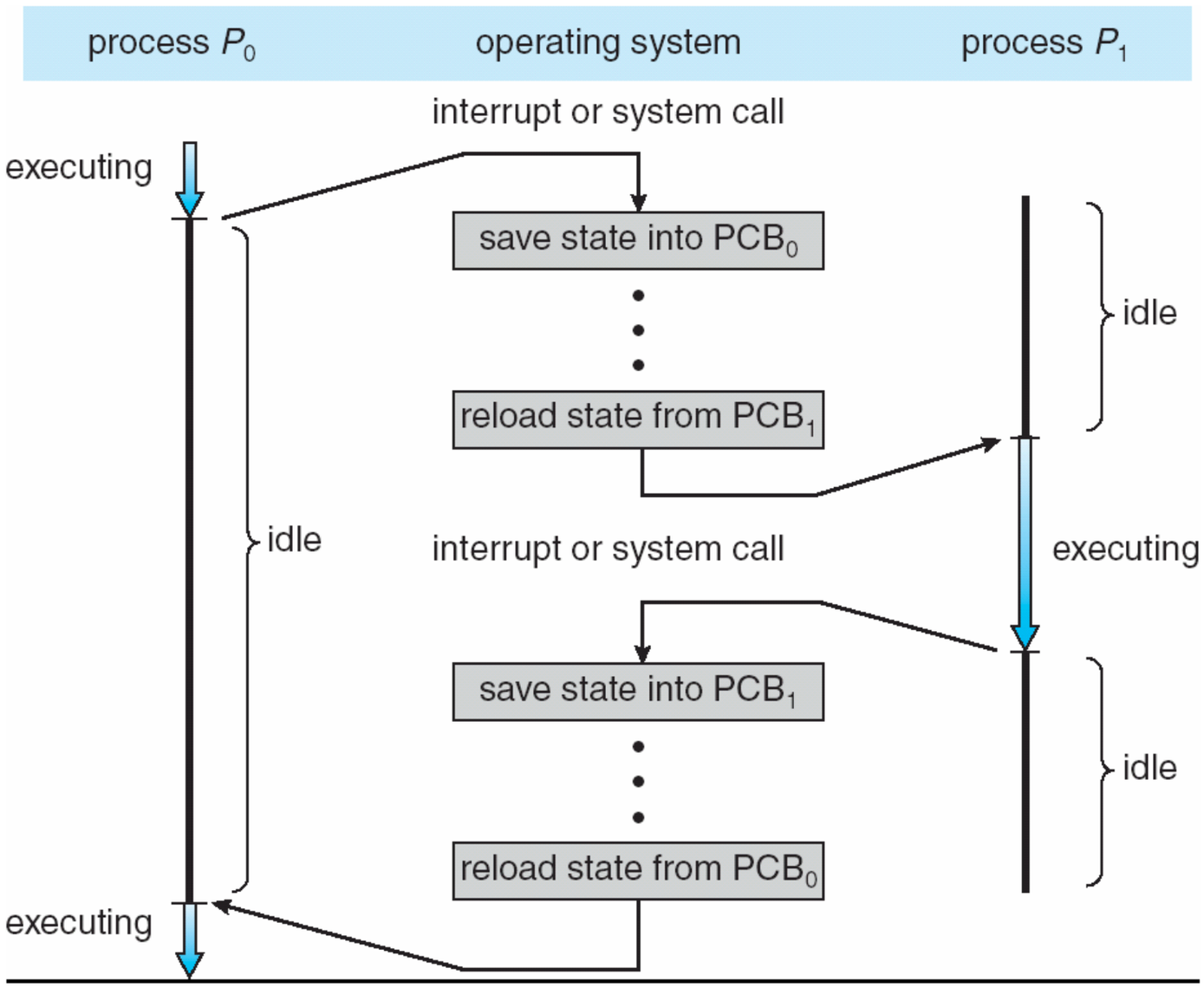


- Linux's PCB is defined in struct task_struct: http://lxr.linux.no/linux+v3.2.35/include/linux/sched.h#L1221

Context Switch

- **Context switch**: the kernel switches to another process for execution
  - save the state of the old process
  - load the saved state for the new process
- **Context-switch is overhead**; CPU does no useful work while switching
  - the more complex the OS and the PCB, longer the context switch
- Context-switch time depends on hardware support
  - some hardware provides multiple sets of registers per CPU: multiple contexts loaded at once

Context Switch



Process Creation

- UNIX/Linux system calls for process creation
  - **fork** creates a new process
  - **exec** overwrites the process' address space with a new program
  - **wait** waits for the child(ren) to terminate

C Program Forking Separate Process


```c
#include <sys/types.h>
#include <studio.h>
#include <unistd.h>
int main()
{
    pid_t  pid;
    pid = fork();                       /* fork another process */
    if (pid < 0) {                      /* error occurred while forking */
        fprintf(stderr, "Fork Failed");
        return -1;
    } else if (pid == 0) {              /* child process */
        execlp("/bin/ls", "ls", NULL);
    } else {                            /* parent process */
        wait (NULL);
        printf ("Child Complete");
    }
    return 0;
}
```

Zombie vs Orphan

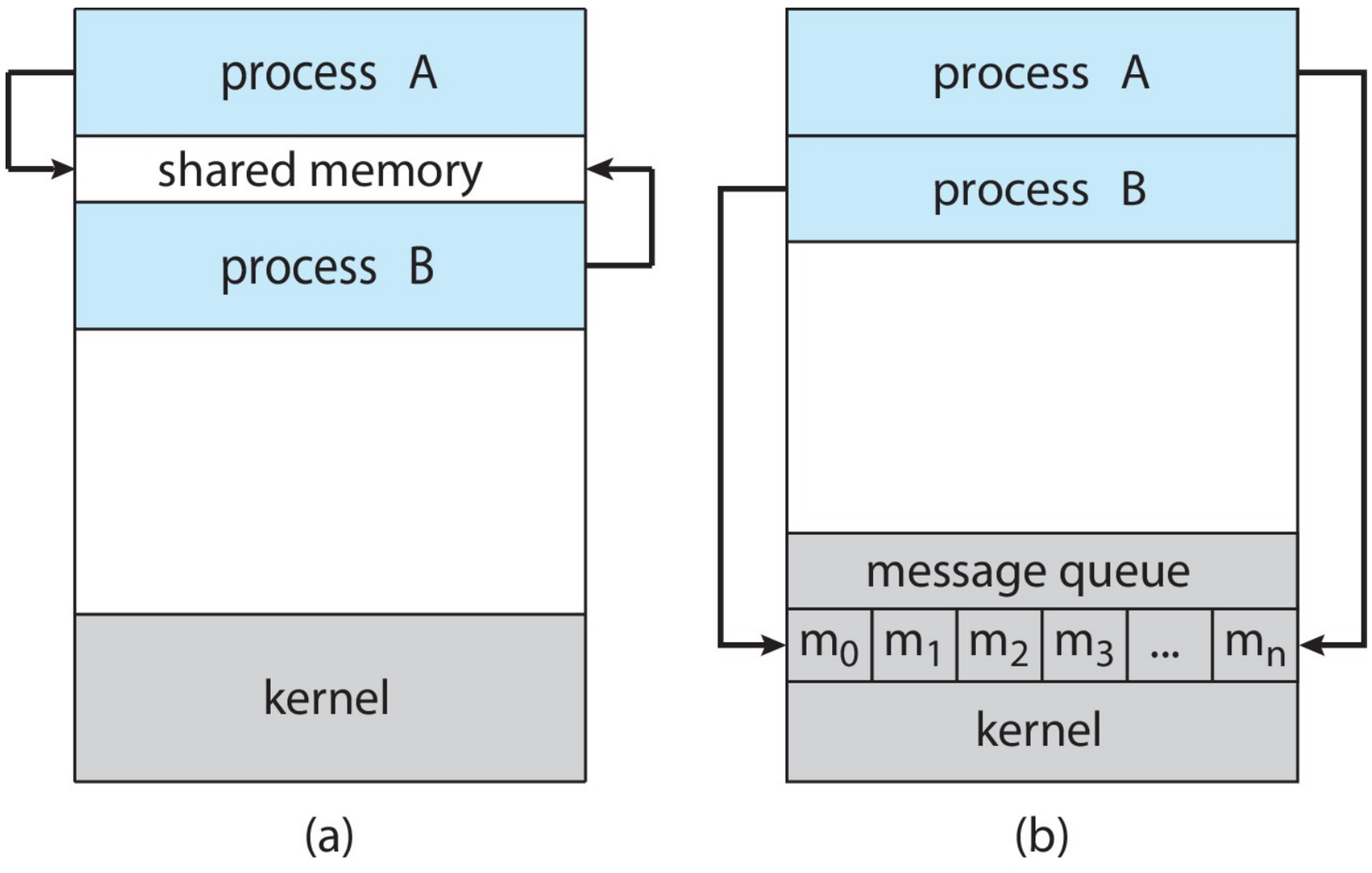
- zombie vs orphan
  - When child process terminates, it is still in the process table until the parent process calls wait()
  - zombie: child has terminated execution, but parent did not invoke wait()
  - orphan: parent terminated without invoking wait - Systemd will take over. Systemd will call wait() periodically

Interprocess Communication

- Two models of IPC
  - Shared memory
  - Message passing

(a) Shared memory.

(b) Message passing.



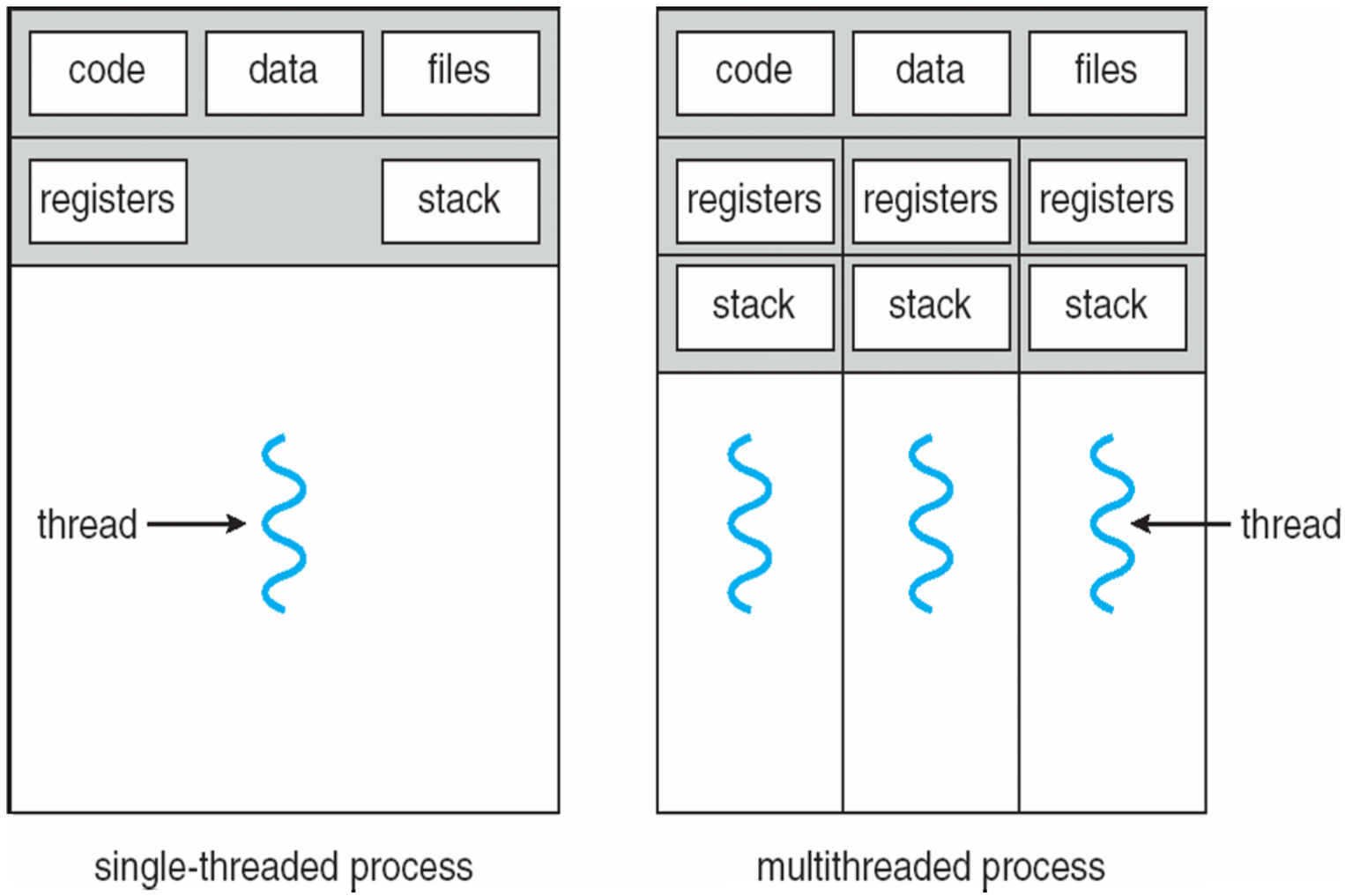
Interprocess Communication – Shared Memory

- An area of memory shared among the processes that wish to communicate
- The communication is under the control of the users processes not the operating system.
- Major issues is to provide mechanism that will allow the user processes to synchronize their actions when they access shared memory.
- Synchronization is an issue

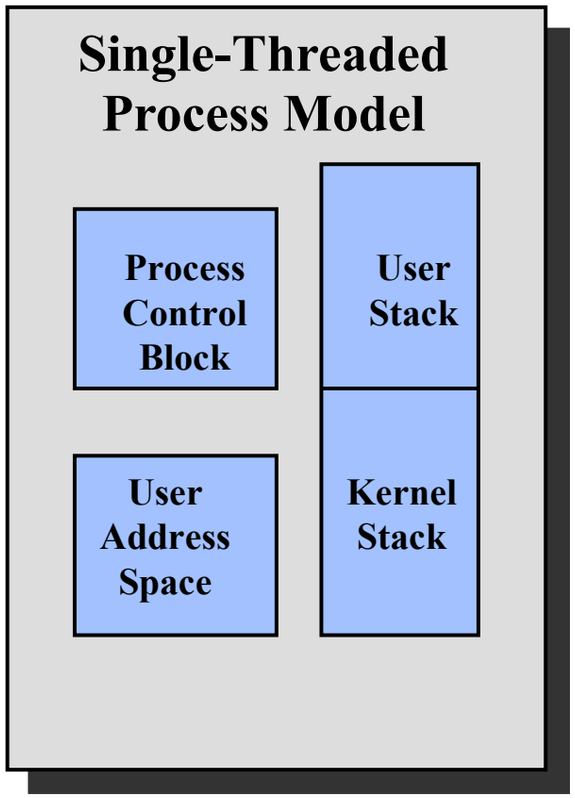
Message Passing

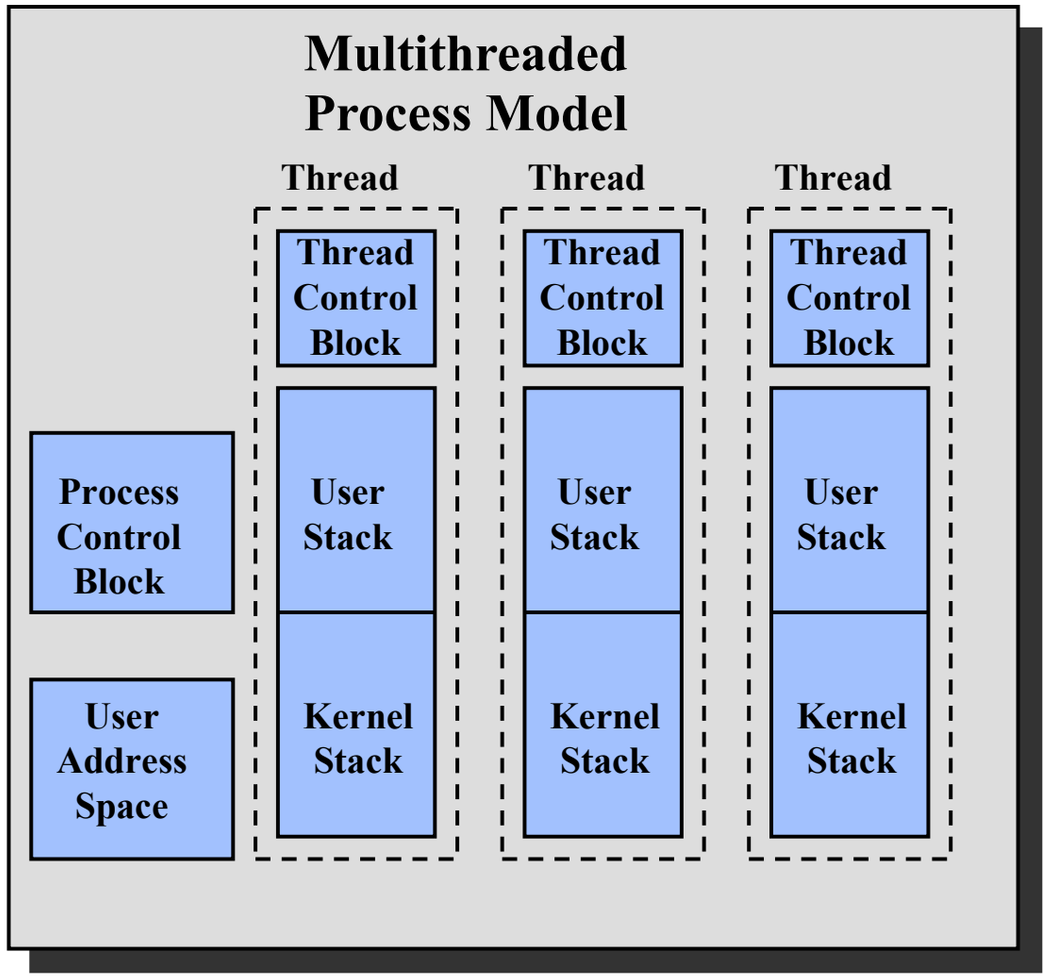
- Processes communicate with each other by exchanging messages
  - without resorting to shared variables
- Message passing provides two operations:
  - **send** (message)
  - **receive** (message)
- Message passing may be either **blocking** or **non-blocking**
- Blocking is considered **synchronous**
  - **blocking send** has the sender block until the message is received
  - **blocking receive** has the receiver block until a message is available
- Non-blocking is considered **asynchronous**
  - **non-blocking send** has the sender send the message and continue
  - **non-blocking receive** has the receiver receive a valid message or null

Single and Multithreaded Processes

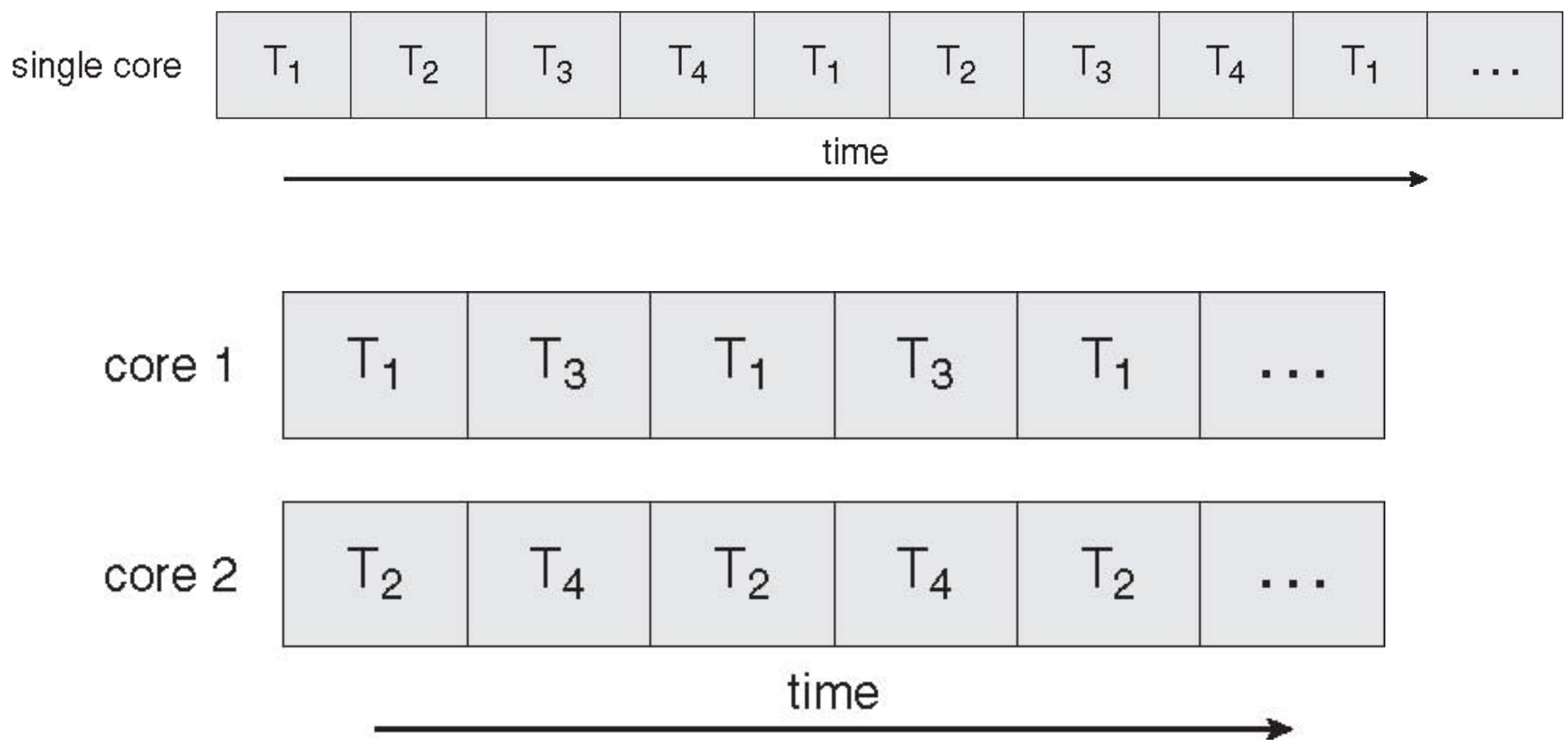


Thread and Process





Concurrency vs Parallelism

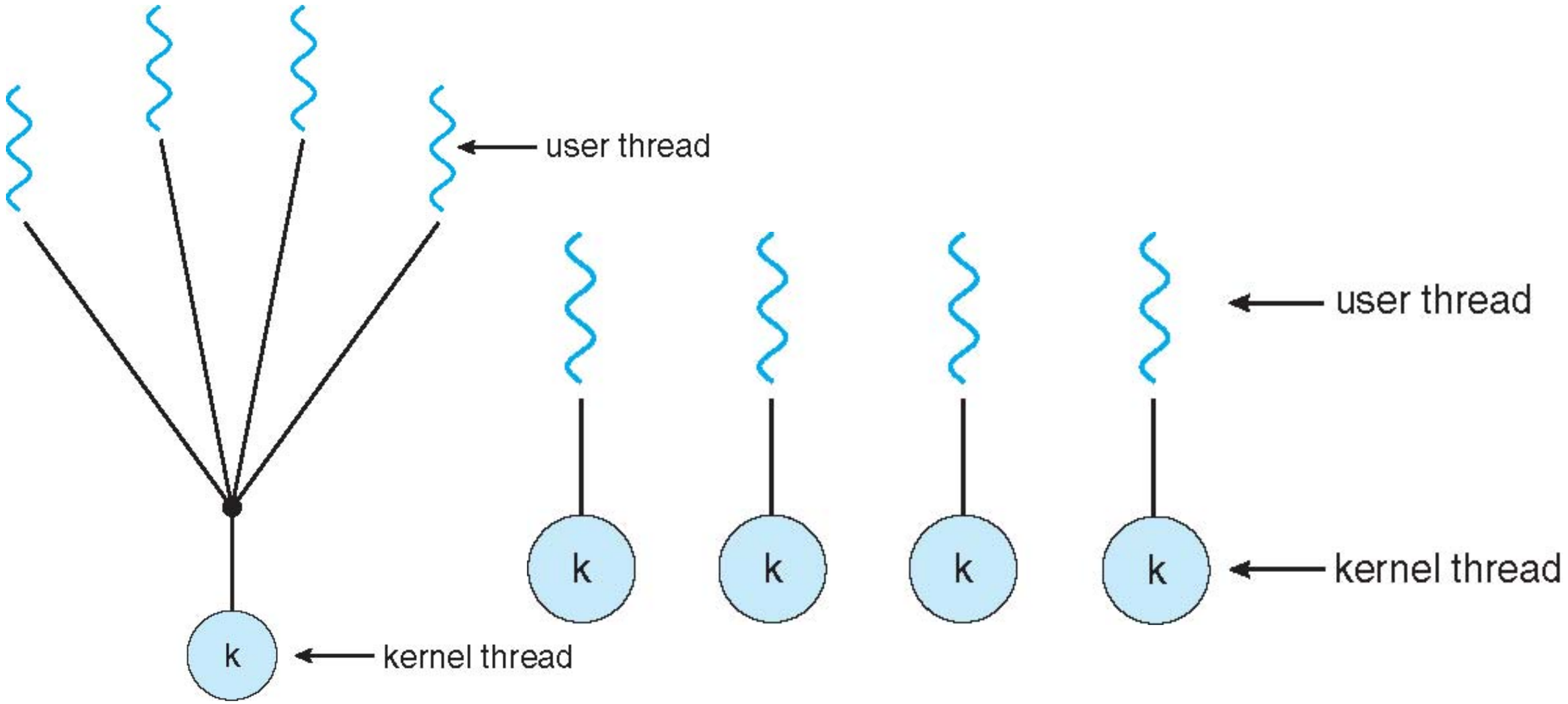


Concurrency is about **structure**, parallelism is about **execution**.
Concurrency provides a way to structure a solution to solve a problem that may (but not necessarily) be parallelizable.

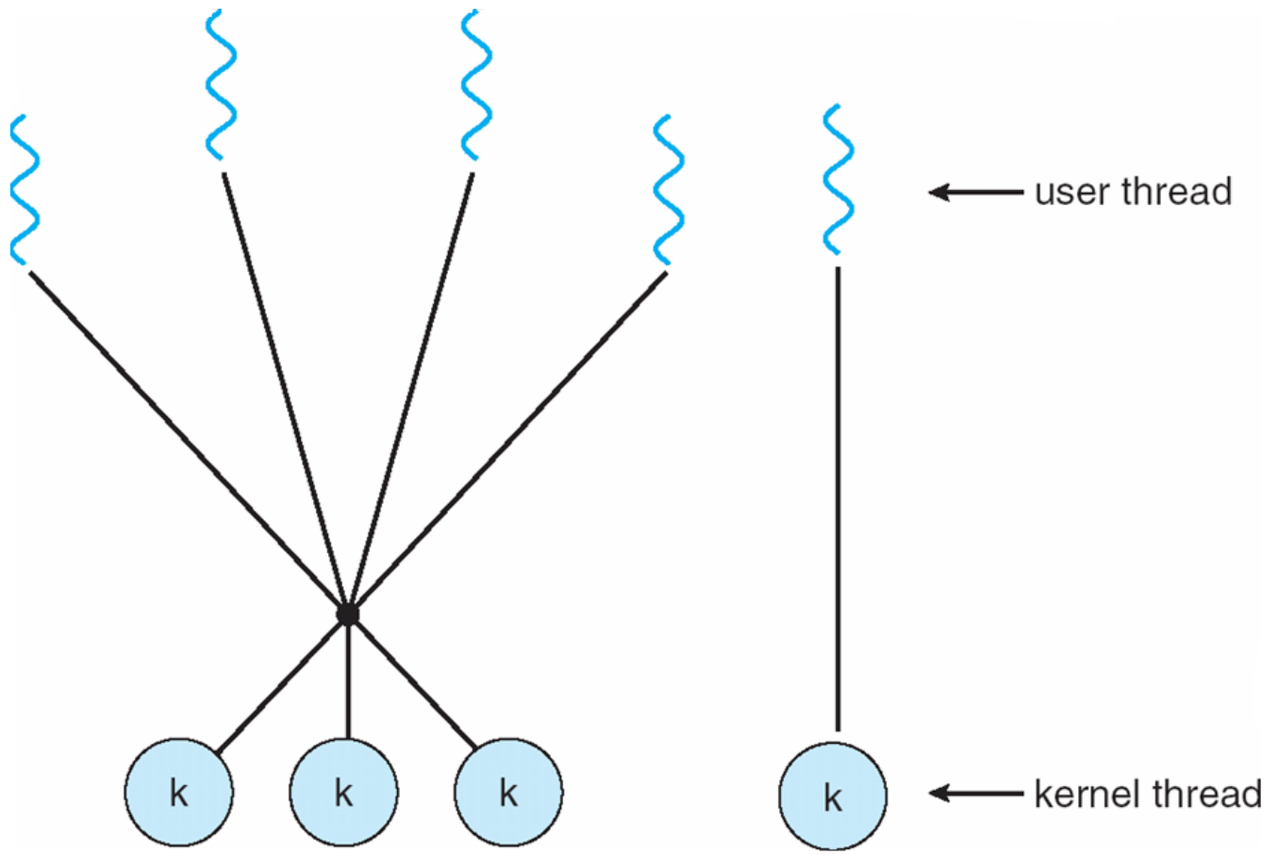
Implementing Threads

- Thread may be provided either at the user level, or by the kernel
  - **user threads** are supported above the kernel and managed without kernel support
    - three thread libraries: **POSIX Pthreads**, **Win32 threads**, and **Java threads**
- **kernel threads** are supported and managed directly by the kernel
  - all contemporary OS supports kernel threads

Thread Model



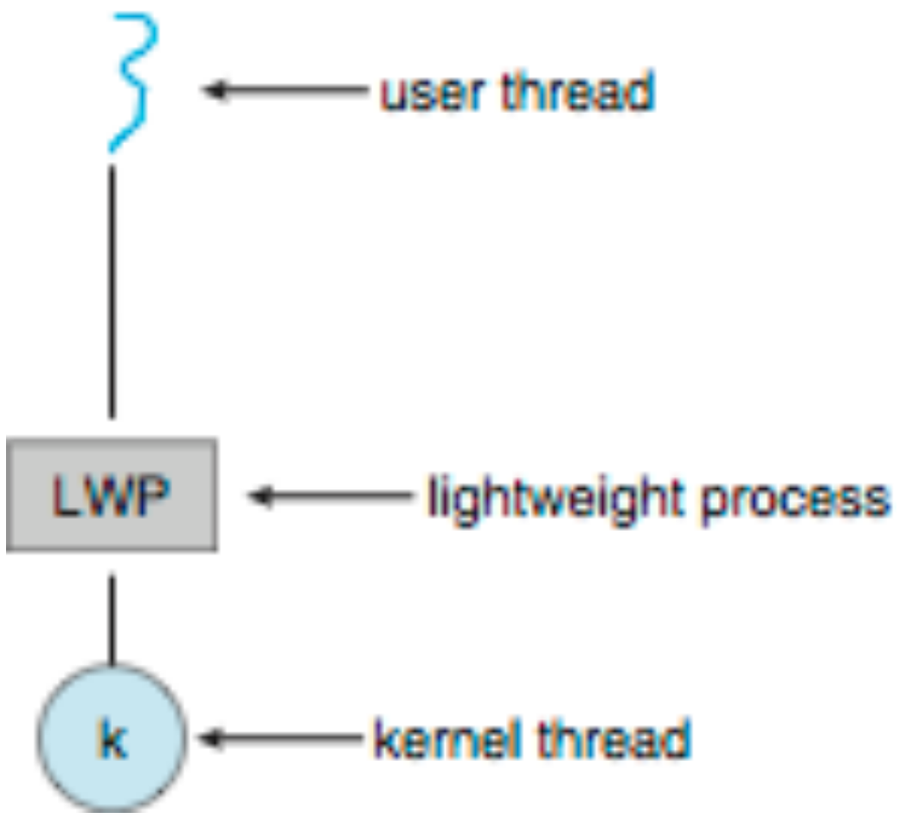
Thread Model



Thread Issues

- Semantics of **fork** and **exec** system calls
- Signal handling
- Thread cancellation of target thread
- Thread-specific data
- Scheduler activations

Lightweight Processes



Linux Threads

```
os@os:~$ ps -efL
UID          PID    PPID     LWP  C NLWP STIME TTY          TIME CMD
root           1       0       1  0    1 Oct13 ?        00:00:05 /sbin/init text
root           2       0       2  0    1 Oct13 ?        00:00:00 [kthreadd]
root           4       2       4  0    1 Oct13 ?        00:00:00 [kworker/0:0H]
root           6       2       6  0    1 Oct13 ?        00:00:00 [mm_percpu_wq]
root           7       2       7  0    1 Oct13 ?        00:00:00 [ksoftirqd/0]
root           8       2       8  0    1 Oct13 ?        00:00:02 [rcu_sched]
root           9       2       9  0    1 Oct13 ?        00:00:00 [rcu_bh]
root          10       2      10  0    1 Oct13 ?        00:00:00 [migration/0]
root          11       2      11  0    1 Oct13 ?        00:00:00 [watchdog/0]
root          12       2      12  0    1 Oct13 ?        00:00:00 [cpuhp/0]
root          13       2      13  0    1 Oct13 ?        00:00:00 [cpuhp/1]
root          14       2      14  0    1 Oct13 ?        00:00:00 [watchdog/1]
root          15       2      15  0    1 Oct13 ?        00:00:00 [migration/1]
root          16       2      16  0    1 Oct13 ?        00:00:00 [ksoftirqd/1]
root          18       2      18  0    1 Oct13 ?        00:00:00 [kworker/1:0H]
root         761       1     761  0    8 Oct13 ?        00:00:00 /usr/lib/snapd/snapd
root         761       1     806  0    8 Oct13 ?        00:00:00 /usr/lib/snapd/snapd
root         761       1     807  0    8 Oct13 ?        00:00:00 /usr/lib/snapd/snapd
root         761       1     808  0    8 Oct13 ?        00:00:00 /usr/lib/snapd/snapd
root         761       1     822  0    8 Oct13 ?        00:00:01 /usr/lib/snapd/snapd
root         761       1     823  0    8 Oct13 ?        00:00:00 /usr/lib/snapd/snapd
root         761       1     824  0    8 Oct13 ?        00:00:00 /usr/lib/snapd/snapd
root         761       1    4293  0    8 Oct13 ?        00:00:00 /usr/lib/snapd/snapd
```

Clone

- `SIGCHLD` : The thread sends a `SIGCHLD` signal to the parent process after completion. It allows the parent to `wait()` for all its threads to complete.
- `CLONE_FS` : Shares the parent's filesystem information with its thread. This includes the root of the filesystem, the current working directory, and the umask.
- `CLONE_FILES` : The calling and caller process share the same file descriptor table. Any change in the table is reflected in the parent process and all its threads.
- `CLONE_SIGHAND` : Parent and threads share the same signal handler table. Again, if the parent or any thread modifies a signal action, it is reflected to both the parties.
- `CLONE_VM` : The parent and threads run in the same memory space. Any memory writes/mapping performed by any of them is visible to other process.

Preemptive vs nonpreemptive

- **Preemptive scheduling**
- **nonpreemptive**

Scheduling Criteria

- **CPU utilization** : percentage of CPU being busy
- **Throughput**: # of processes that complete execution per time unit
- **Turnaround time**: the time to execute a particular process
  - from the time of *submission* to the time of *completion*
- **Waiting time**: the total time spent waiting in the *ready queue*
- **Response time**: the time it takes from when a request was submitted until the first response is produced
  - the time it takes to *start responding*

Scheduling Algorithms

- First-come, first-served scheduling (FCFS)
- Shortest-job-first scheduling (SJF)
- Priority scheduling
- Round-robin scheduling (RR)
- Multilevel queue scheduling
- Multilevel feedback queue scheduling

Problems

- Starving
- Aging

Multiple Processing

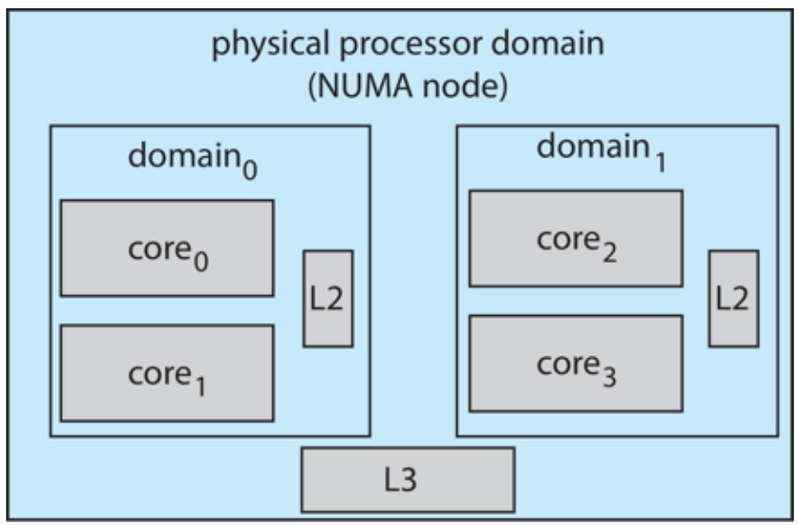
- Load balancing
- Processor Affinity

CFS

- **Scheduling classes**
  - Each has specific priority
  - Scheduler picks **highest priority** task in **highest scheduling class**
  - 2 scheduling classes included, others can be added
    - default
    - real-time
- Quantum calculated based on nice value from -20 to +19
  - Lower value is higher priority
  - Calculates **target latency** – interval of time during which task should run at least once
  - Target latency can increase if number of active tasks increases
- CFS scheduler maintains per task virtual run time in variable vruntime
  - Associated with decay factor based on priority of task – lower priority is higher decay rate
  - Normal default priority yields virtual run time = actual run time

Linux Scheduling

- Linux supports load balancing, but is also NUMA-aware.
- **Scheduling domain** is a set of CPU cores that can be balanced against one another.
- Domains are organized by what they share (i.e. cache memory.) Goal is to **keep threads *from*** migrating between domains.



priority inversion