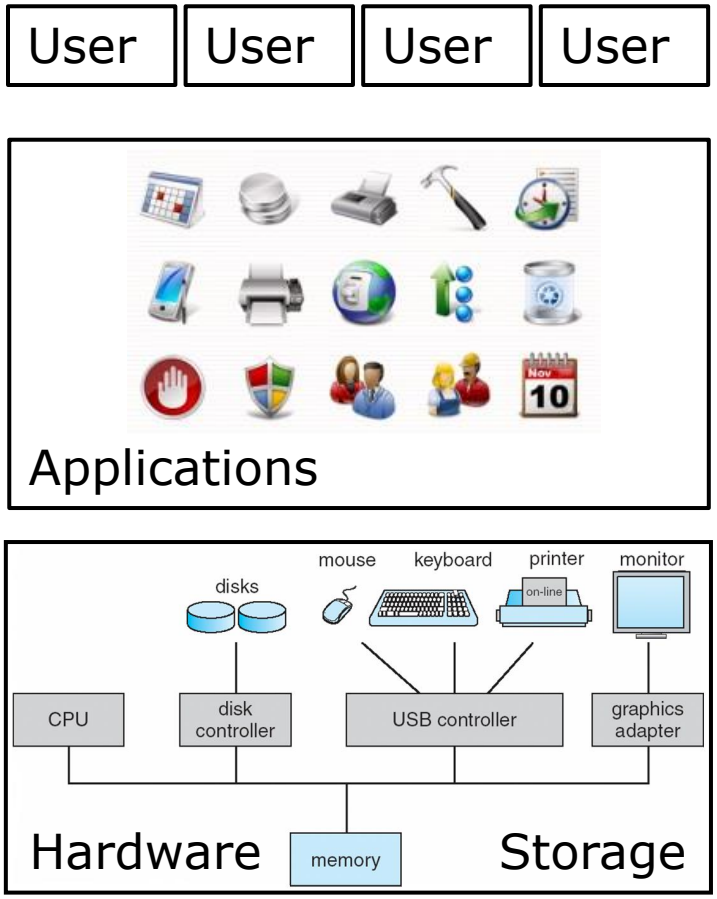
Operating Systems
kernel mode and interrupts

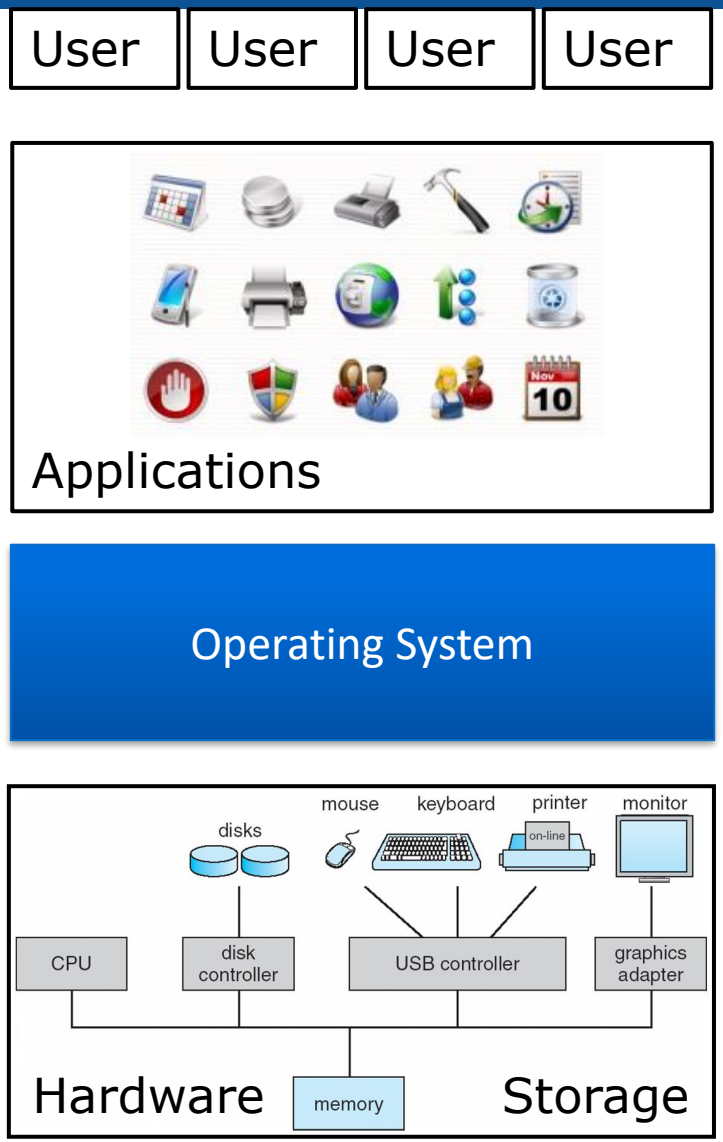
David Hay

Dror Feitelson

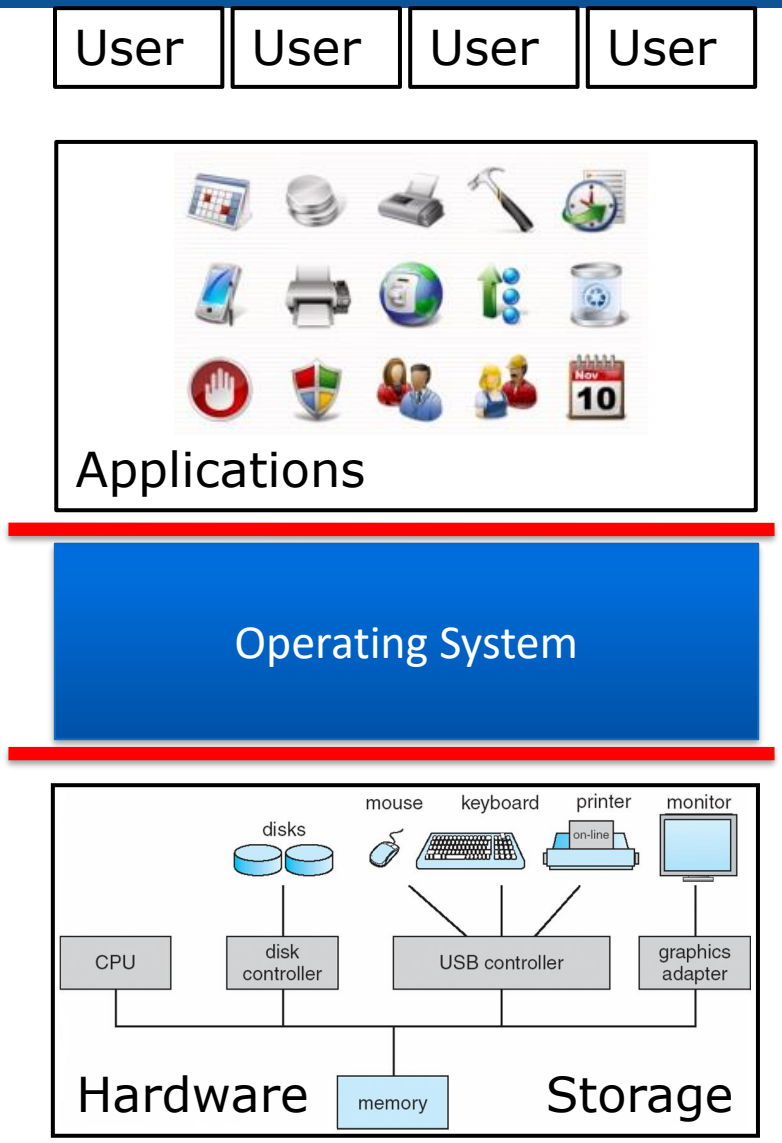
Simplified View of the OS



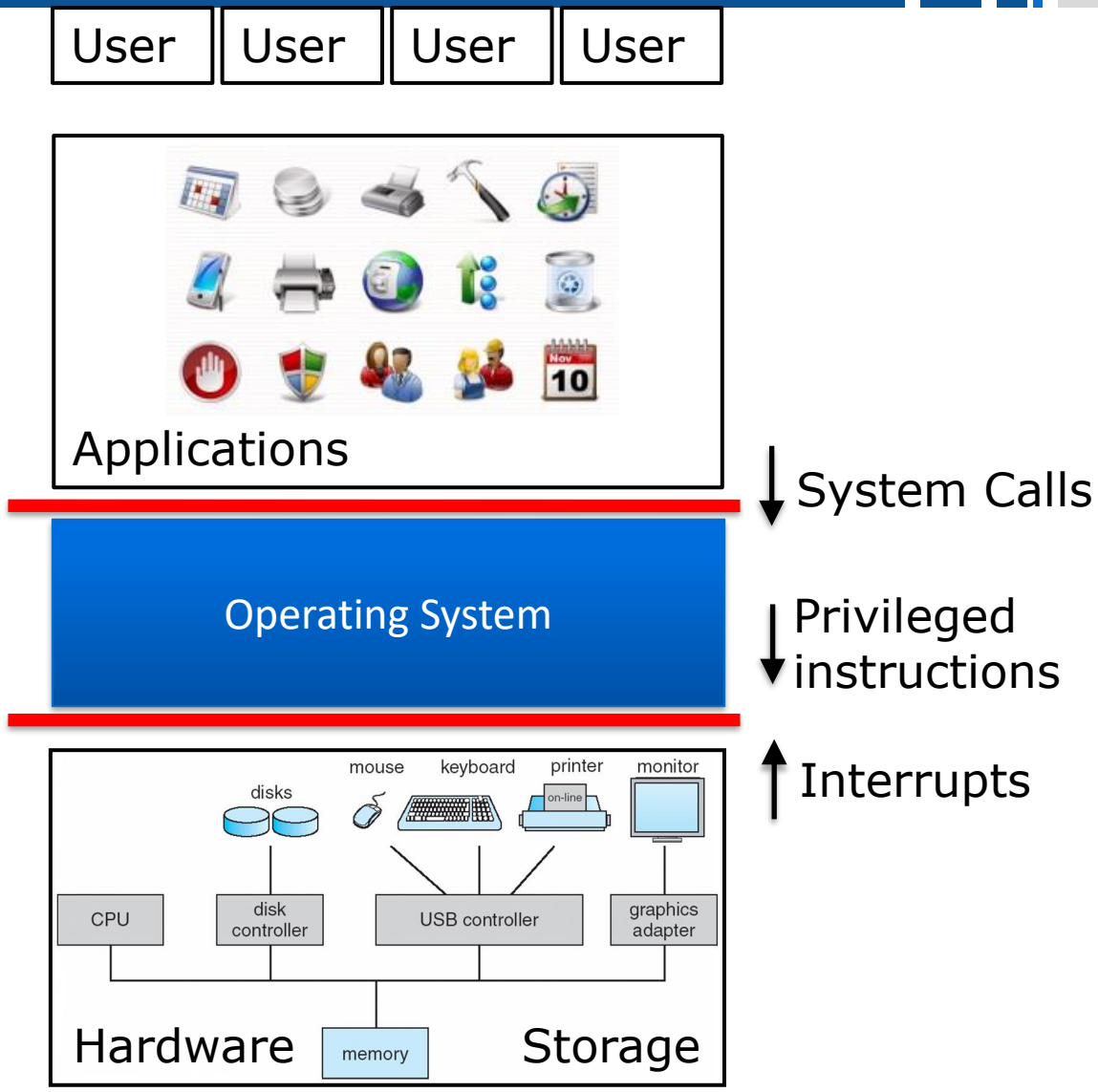
Simplified View of the OS



Simplified View of the OS



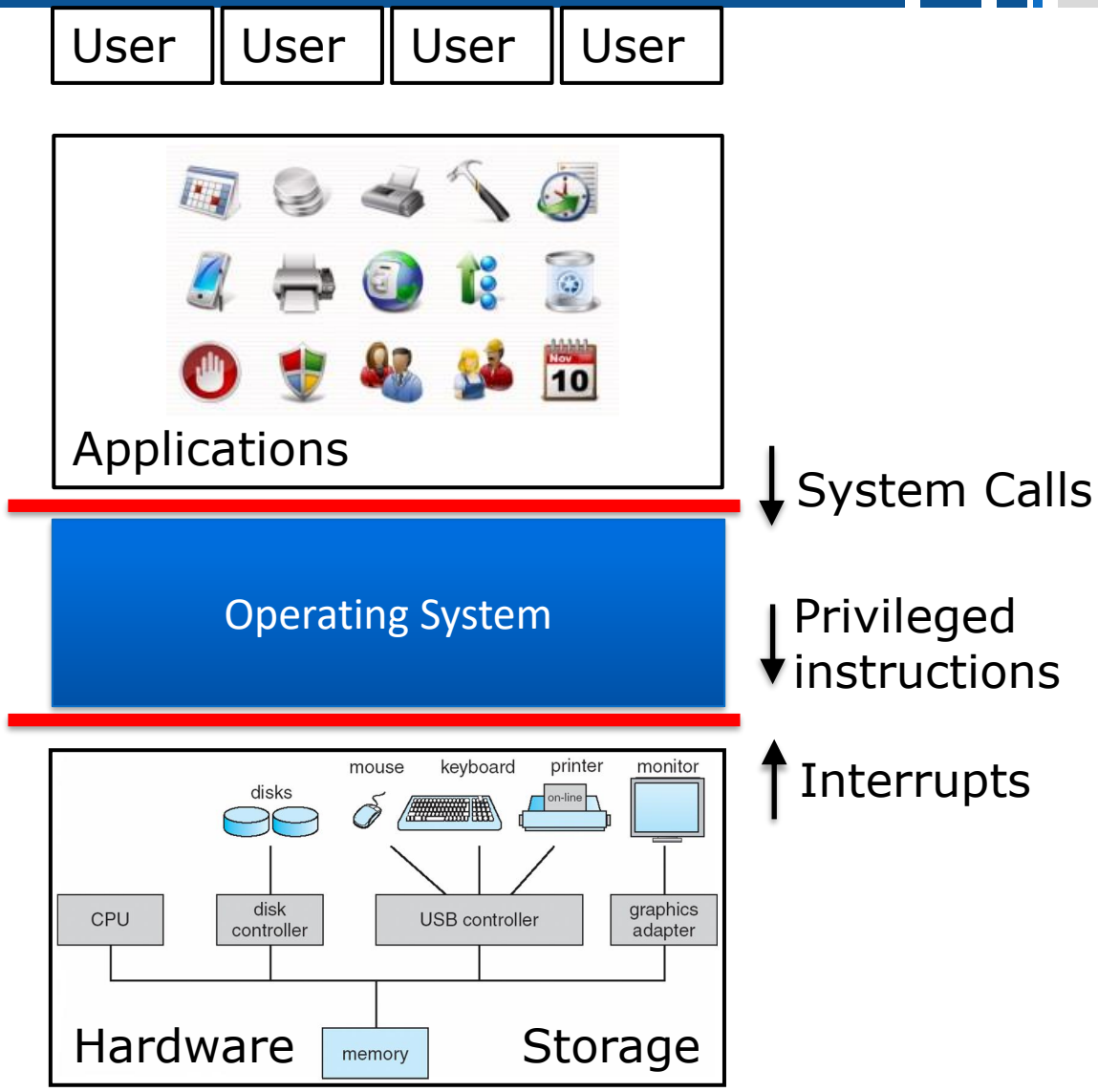
Simplified View of the OS



Two Classes of Instructions

- Non-Privileged Instructions
  - All the usual instructions you expect
  - Add, multiply, jump, …
- Privileged Instructions
  - Special instructions for the OS
  - We don't want user applications to use them
  - Example: activate the disk to get a block of data
    - OS uses this to support the abstraction of files
    - OS prevents access to other users' data (protection)

Simplified View of the OS



Simplified View of the OS

User Mode

Kernel Mode

User | User | User | User

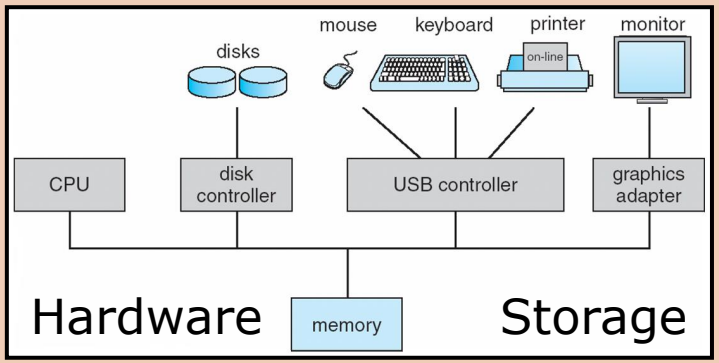
Applications

System Calls

Operating System

Privileged instructions

Interrupts

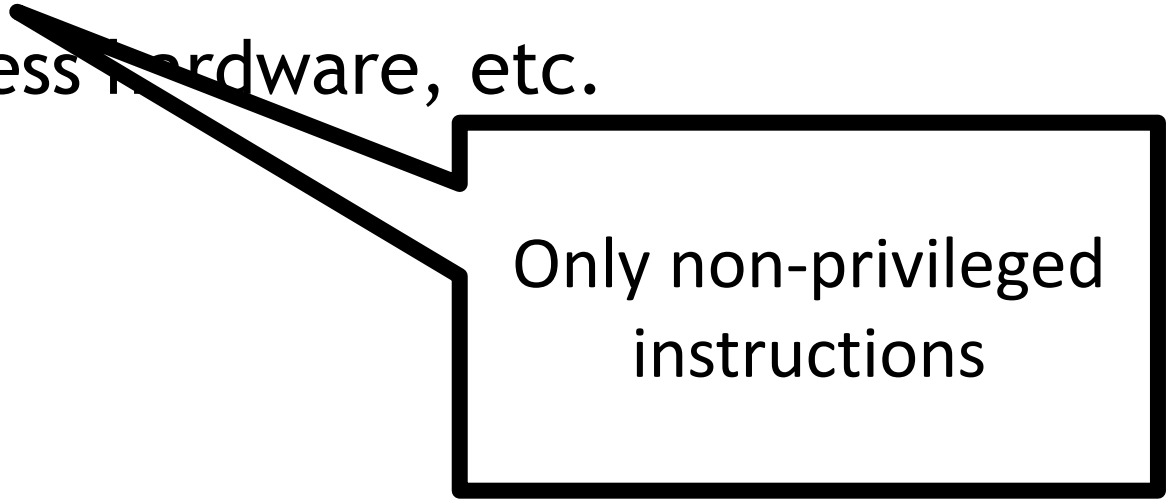


Two Modes of Operation

- Kernel mode (also called “privileged mode”)
  - The running mode of the OS
  - Can run hardware assembly commands
- User mode
  - Can not access hardware, etc.

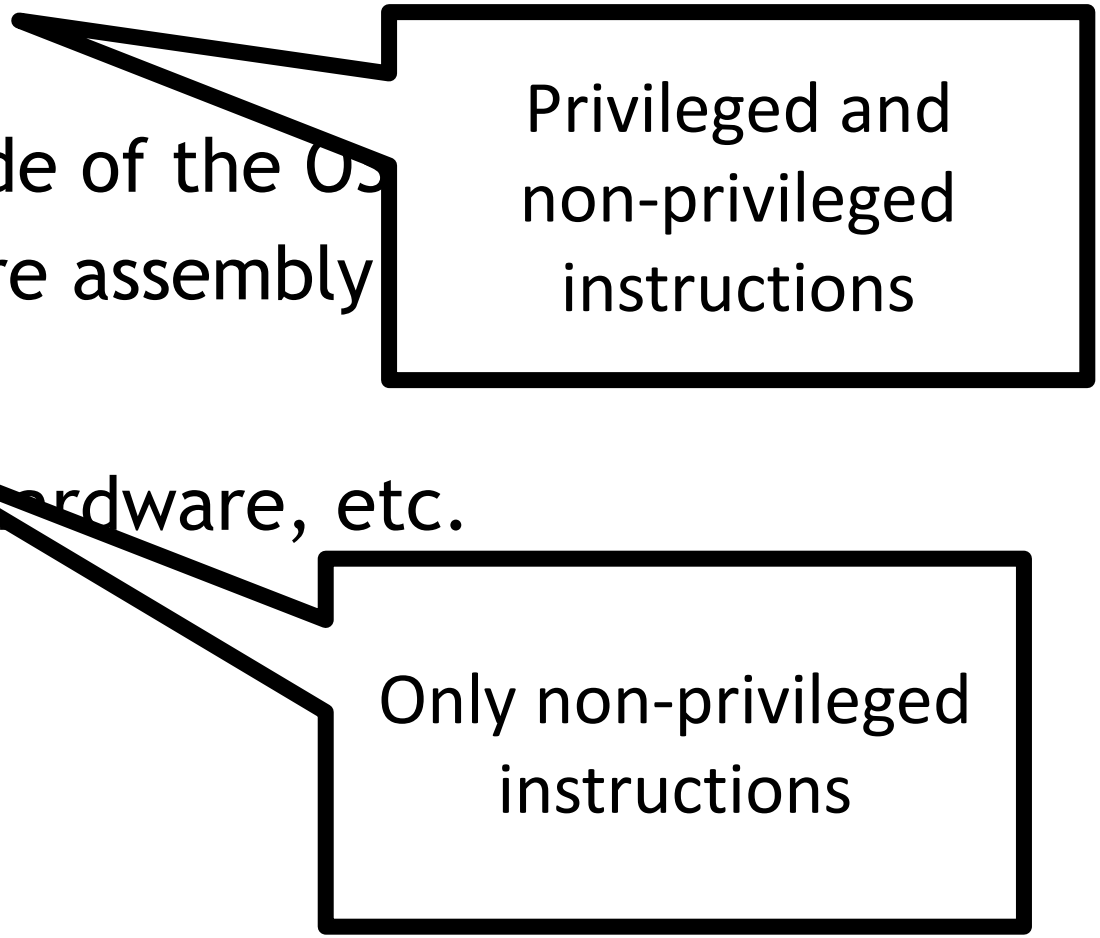
Two Modes of Operation

- Kernel mode (also called “privileged mode”)
  - The running mode of the OS
  - Can run hardware assembly commands
- User mode
  - Can not access hardware, etc.



Two Modes of Operation

- Kernel mode (also called “privileged mode”)
  - The running mode of the OS
  - Can run hardware assembly
- User mode
  - Can not access hardware, etc.

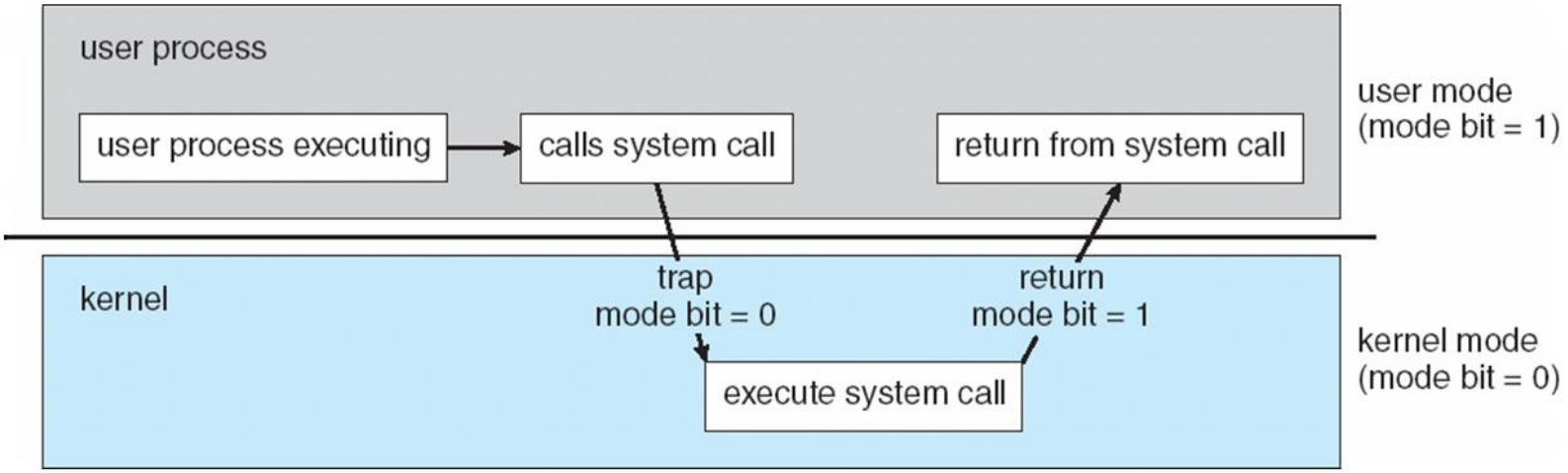


System Calls

- The user process gets the OS services thru system calls
- System calls are functions that can be called from the user programs
  - Not like a regular function calls!
- User programs cannot access/alter internal OS data structures, only thru system calls (protection!)
- System calls check the validity of the parameters (unlike other internal OS functions)
- System calls = portable interface and protection

Transition from User to Kernel Mode

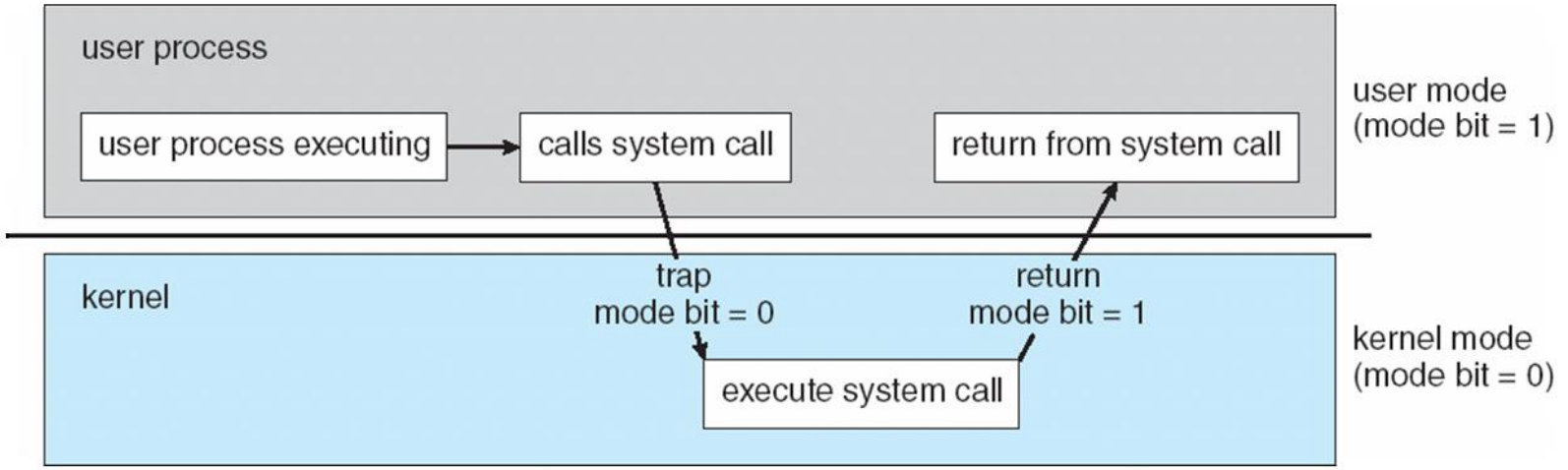
- The process of switching between user and kernel mode is called a **trap**
- **Mode bit** provided and protected by hardware
  - Provides ability to distinguish when system is running user code or kernel code
  - Some instructions designated as **privileged**, only executable in kernel mode
  - System call changes mode to kernel, return from call resets it to user



Transition from User to Kernel Mode

Stored in a specific register (PSW)

- The process of switching between u
  is called a **trap**
- **Mode bit** provided and protected by hardware
  - Provides ability to distinguish when system is running user code or kernel code
  - Some instructions designated as **privileged**, only executable in kernel mode
  - System call changes mode to kernel, return from call resets it to user

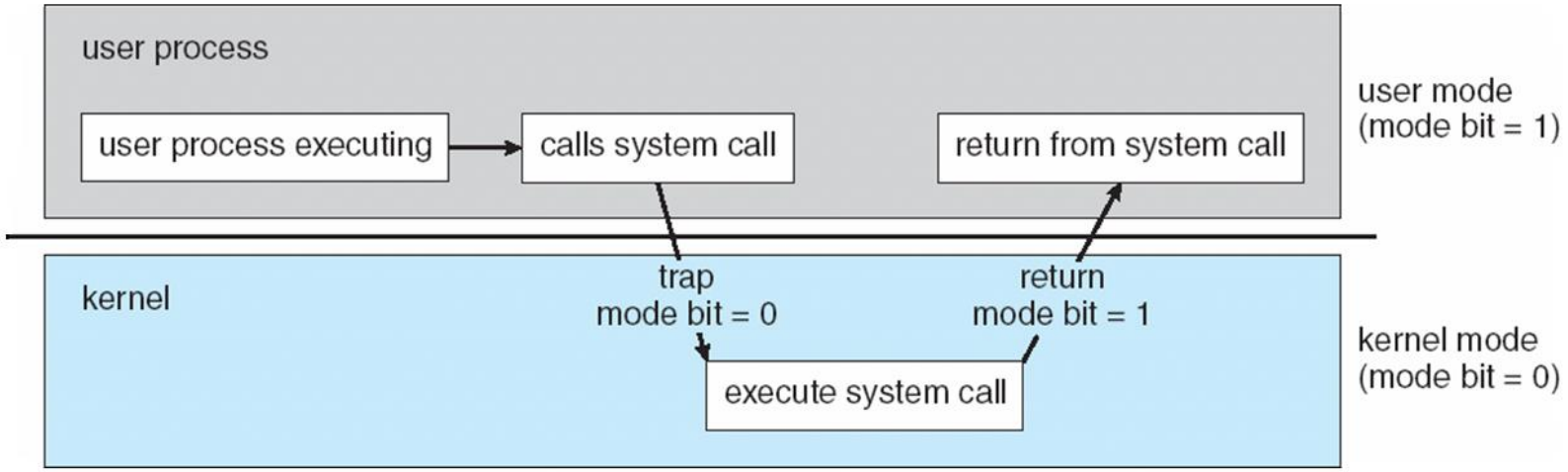


Transition from User to Kernel Mo

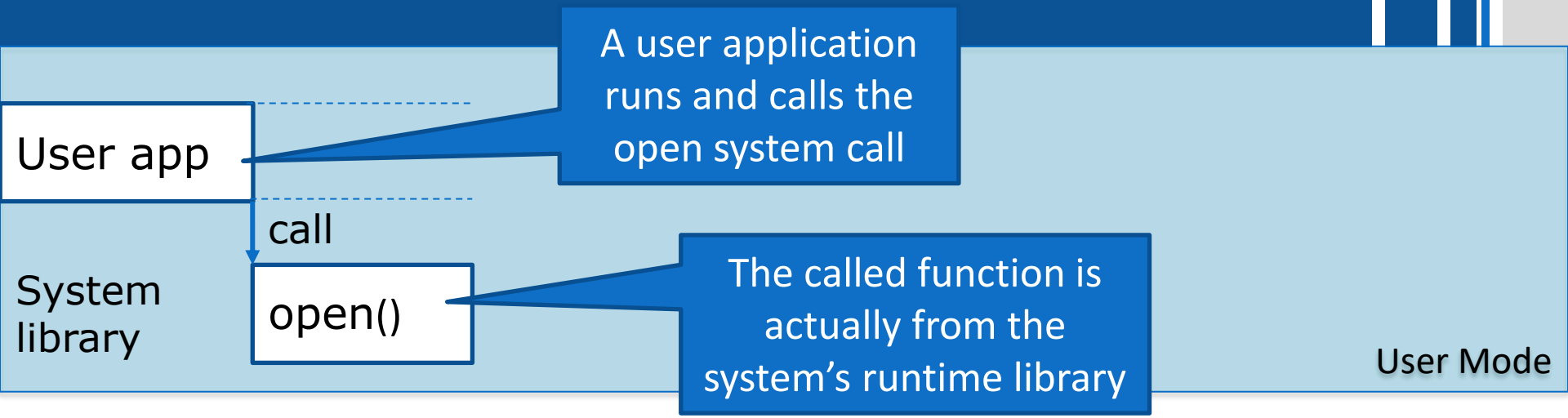
Modern OS has more than one bit, allowing more than 2 "modes" (a.k.a. rings)

Stored in a specific register (PSW)

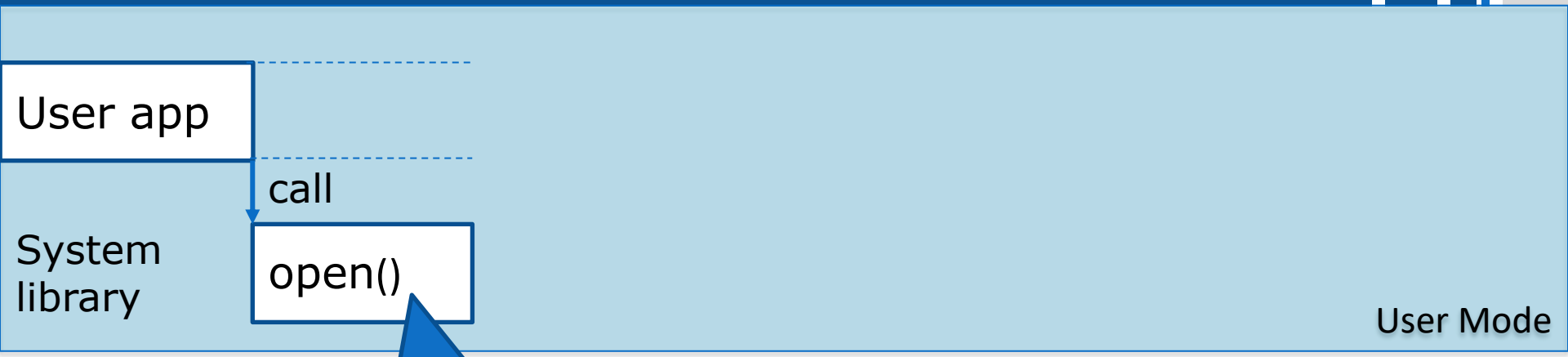
- Th
  is calle **trap**
- **Mode bit** provided and protected by hardware
  - Provides ability to distinguish when system is running user code or kernel code
  - Some instructions designated as **privileged**, only executable in kernel mode
  - System call changes mode to kernel, return from call resets it to user



Trap in Detail



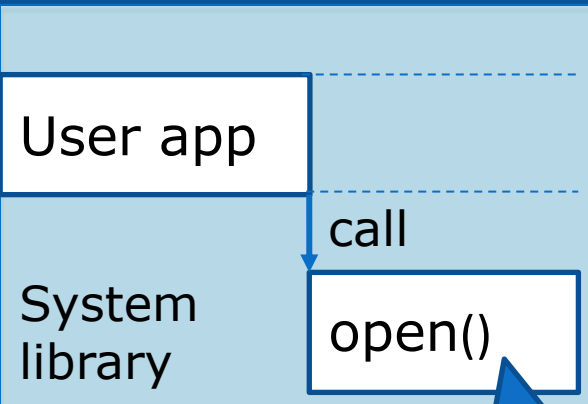
Trap in Detail



This function sets things up for entering the kernel.
Use registers to:

1. Store the parameters
2. Store the system call number for “open” (2)

Trap in Detail

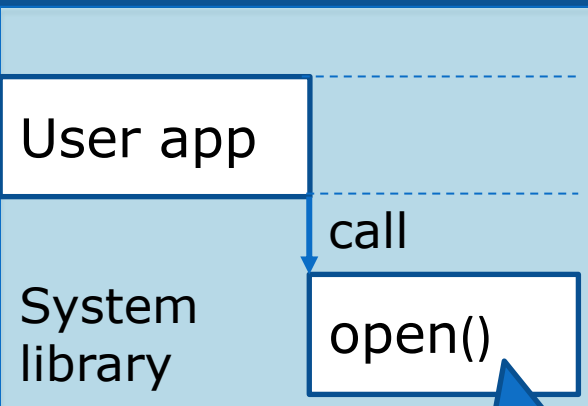


This function sets things up for entering the kernel.
Use registers to:

1. Store the parameters
2. Store the system call number for “open” (2)

0	read
1	write
2	open
3	close
4	stat
5	fstat
6	lstat
7	poll
8	lseek
9	mmap
10	mprotect
11	munmap
12	brk
13	rt_sigaction
14	rt_sigprocmask

Trap in Detail



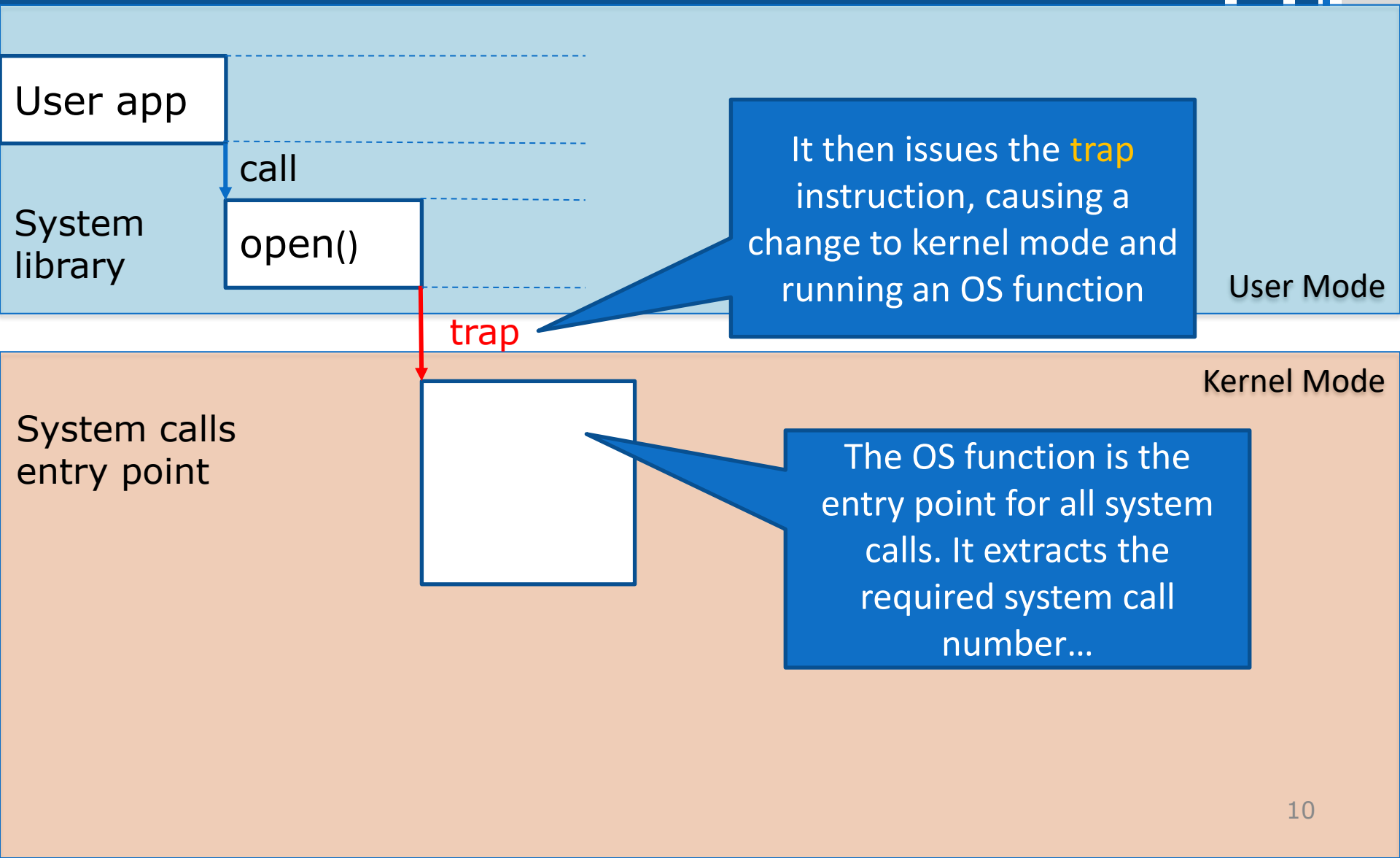
This function sets things up for entering the kernel.
Use registers to:

1. Store the parameters
2. Store the system call number for “open” (2)

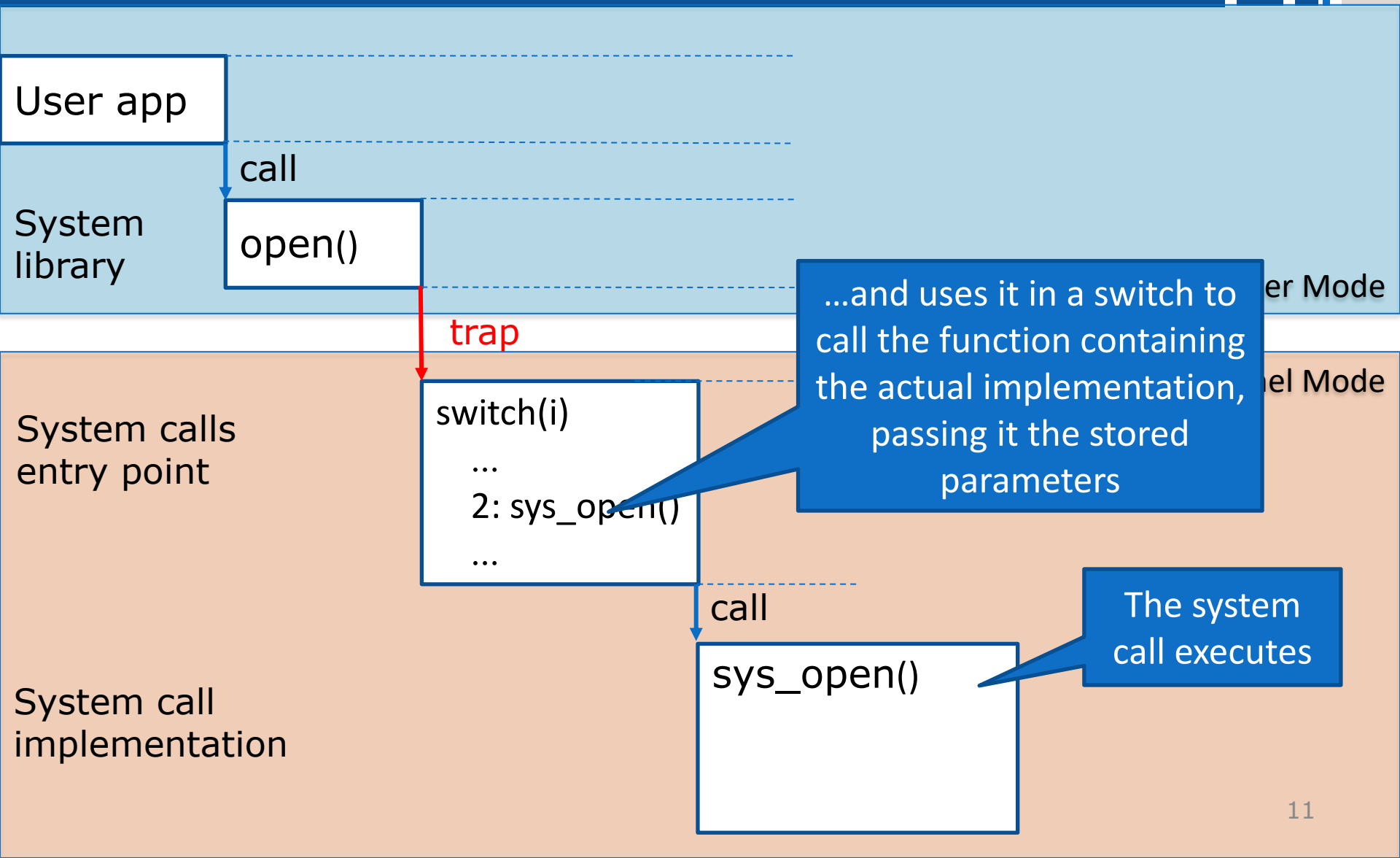
Linux has 300-400 different system calls…

0	read
1	write
2	open
3	close
4	stat
5	fstat
6	lstat
7	poll
8	lseek
9	mmap
10	mprotect
11	munmap
12	brk
13	rt_sigaction
14	rt_sigprocmask

Trap in Detail

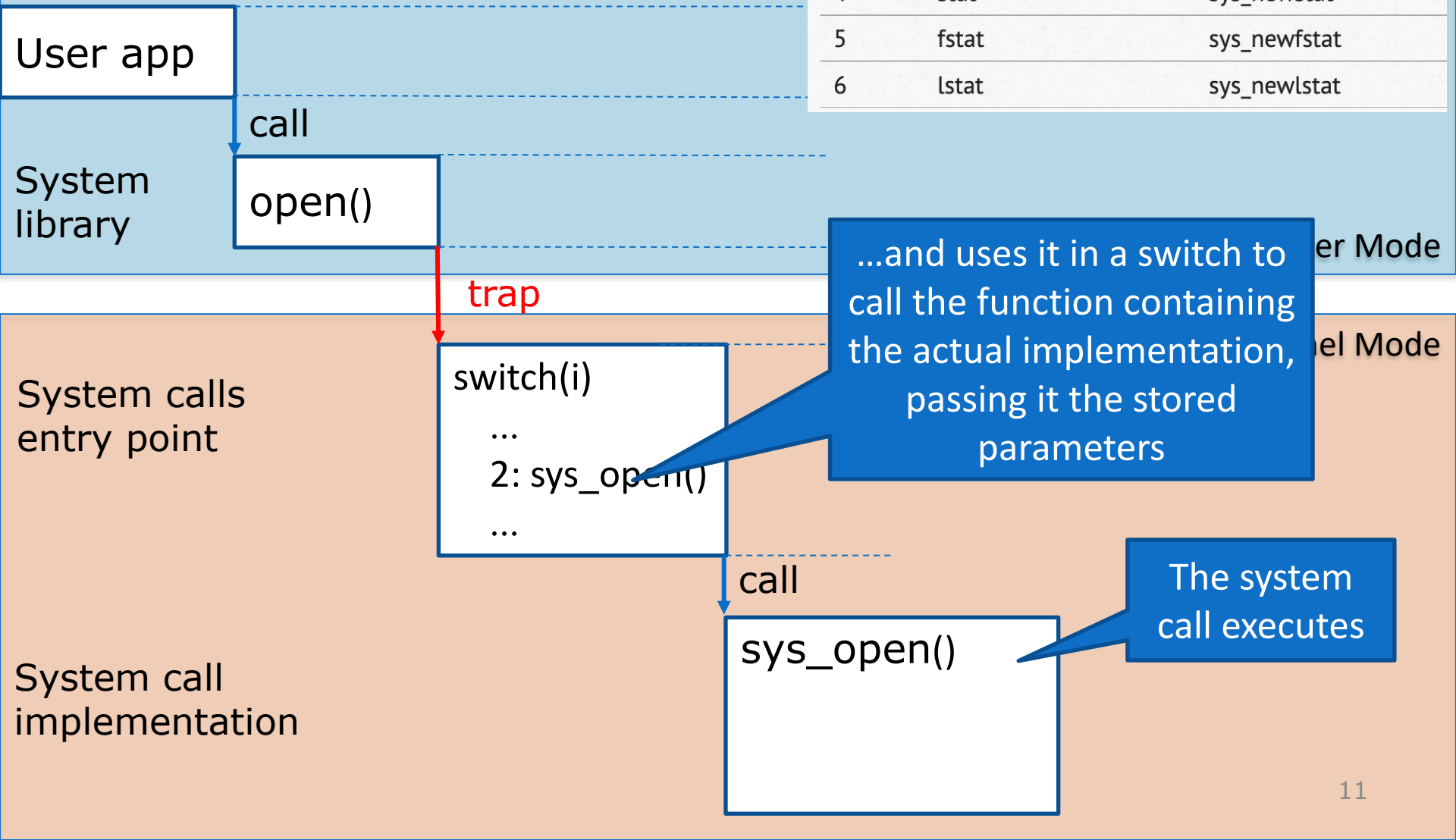


Trap in Detail

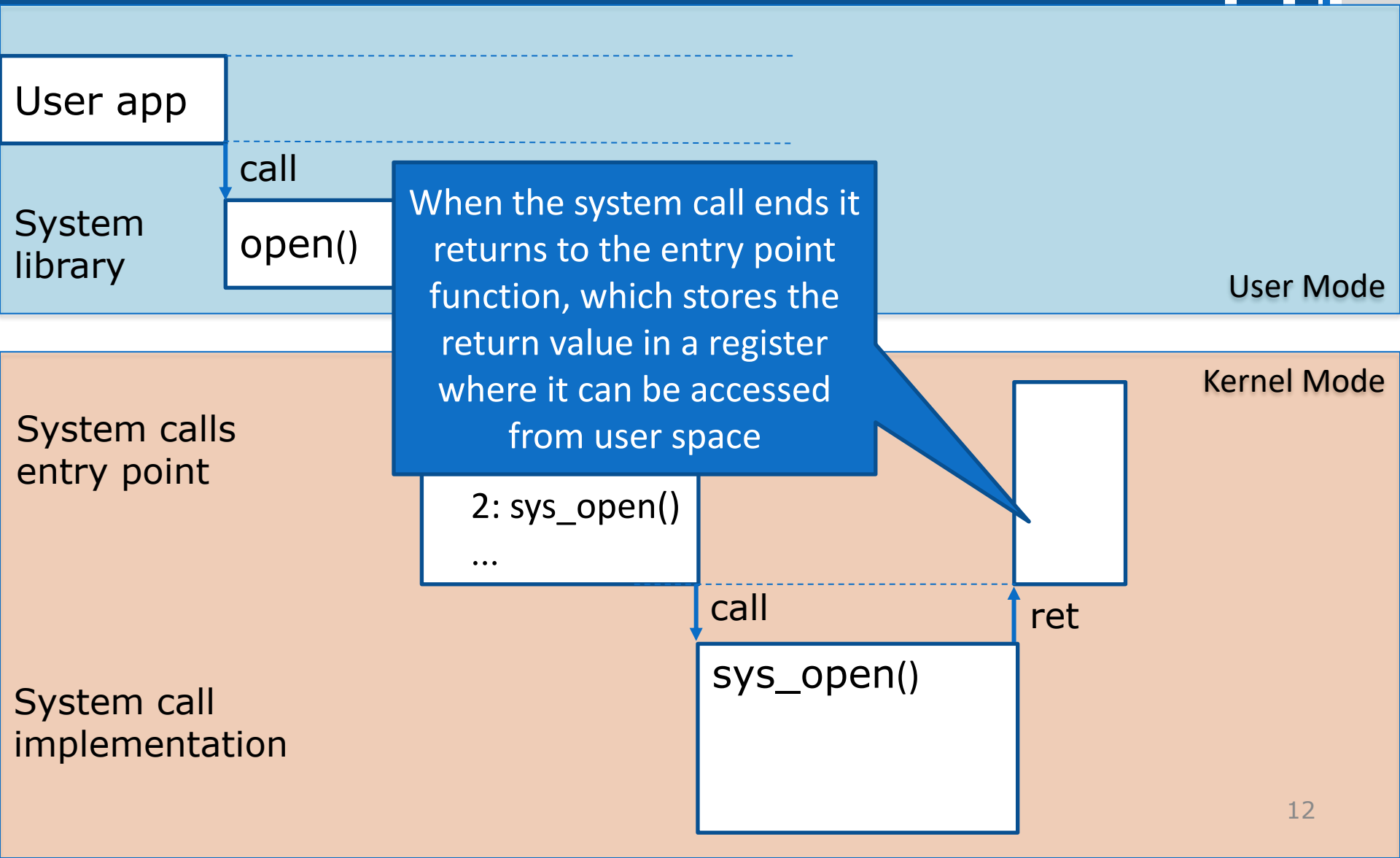


Trap in Detail

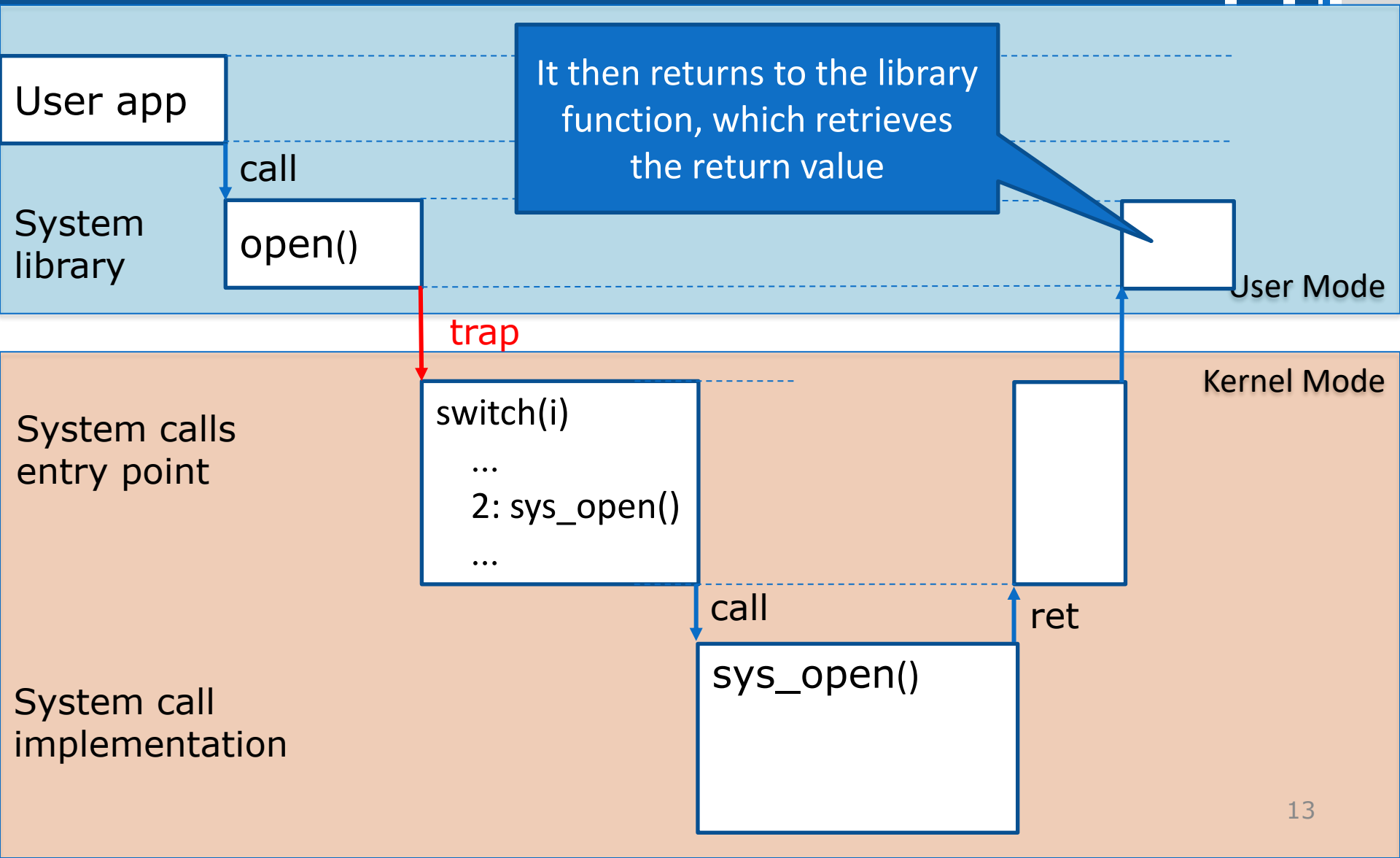
0	read	sys_read
1	write	sys_write
2	open	sys_open
3	close	sys_close
4	stat	sys_newstat
5	fstat	sys_newfstat
6	lstat	sys_newlstat



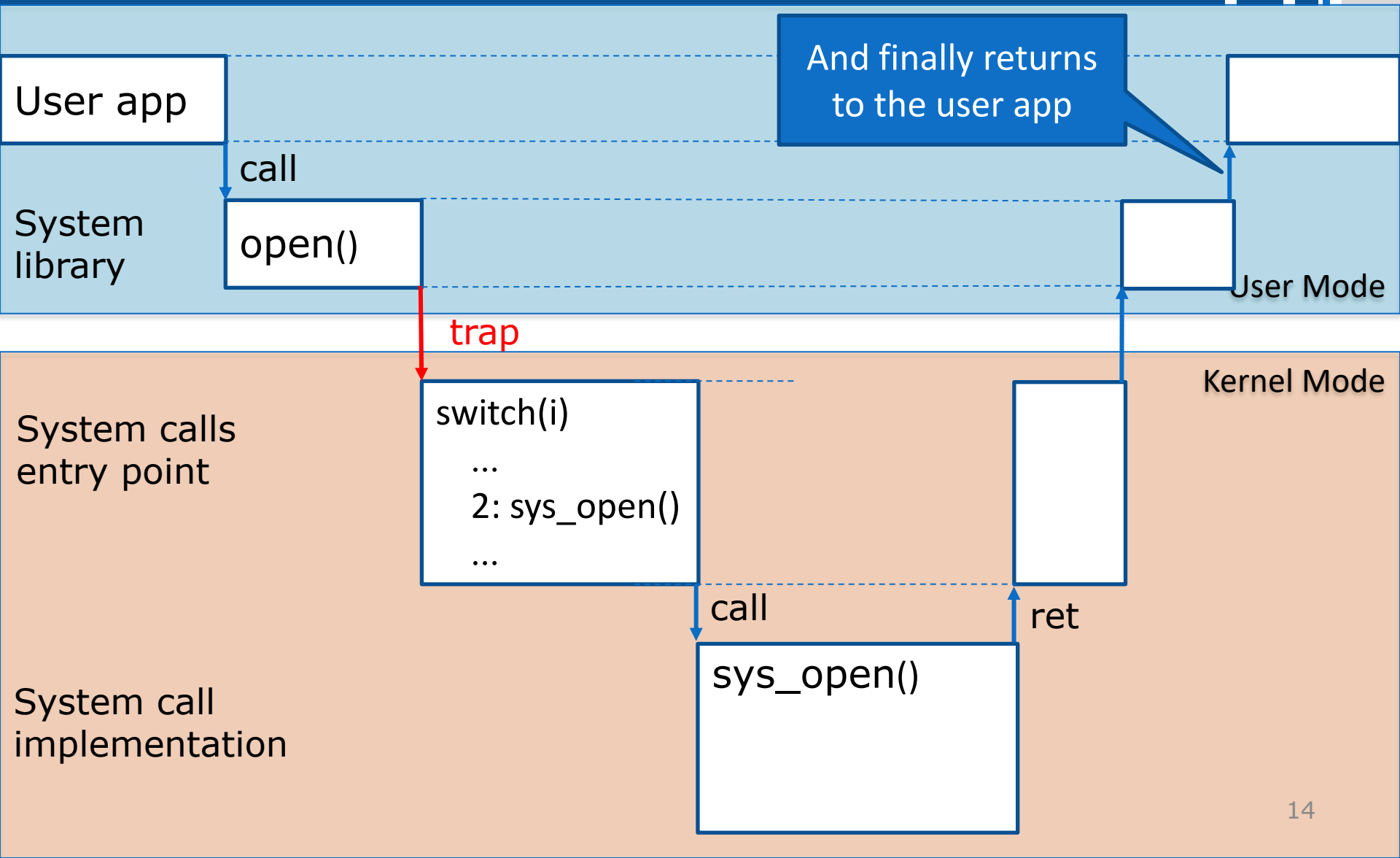
Trap in Detail



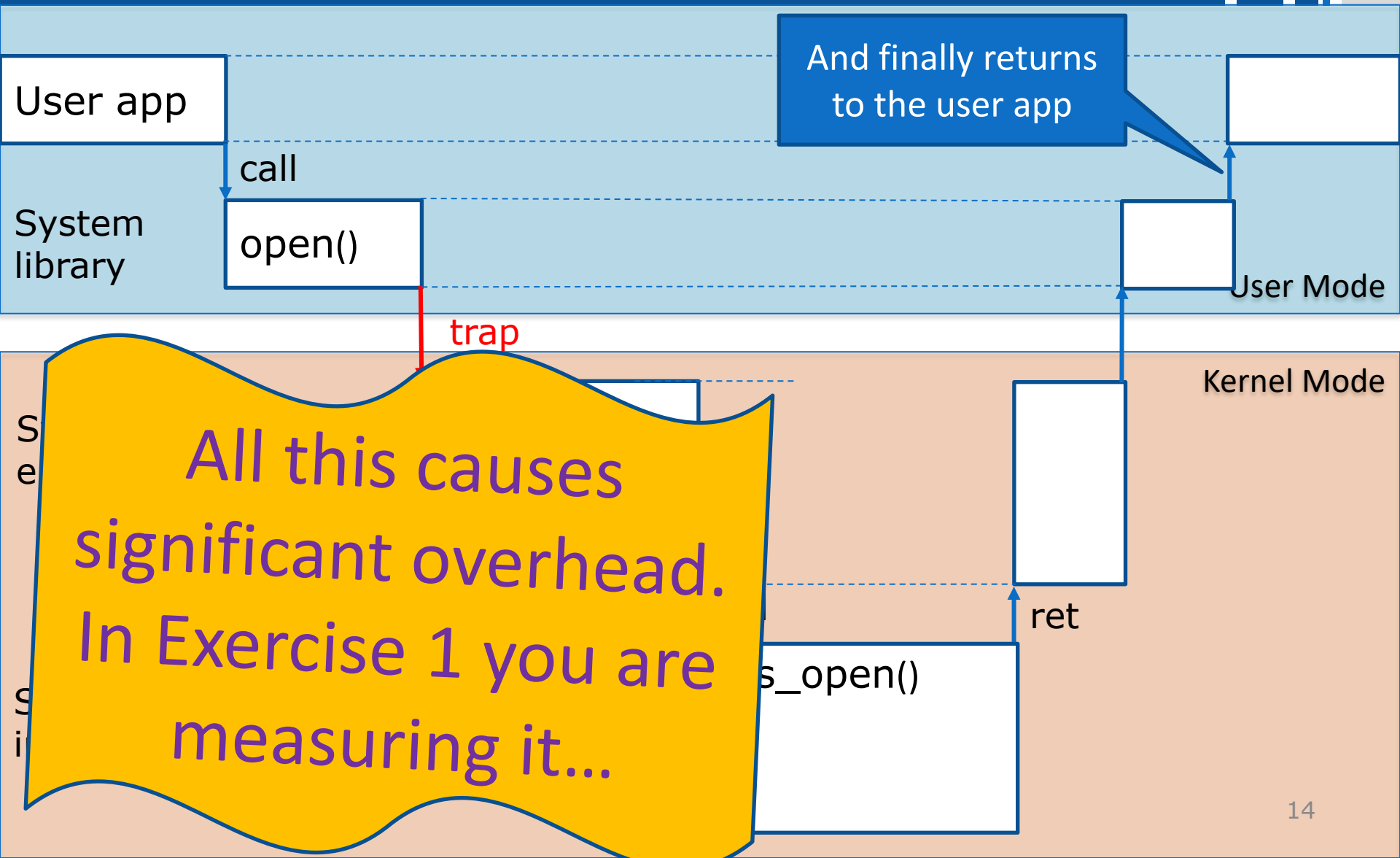
Trap in Detail



Trap in Detail



Trap in Detail



Types of System Calls

- Process Control
- File management
- Device management
- Information maintenance
- Communications
- Protection

Transition from User to Kernel Mode

- Sometimes happens because something wrong happened: exception (e.g., division by 0)

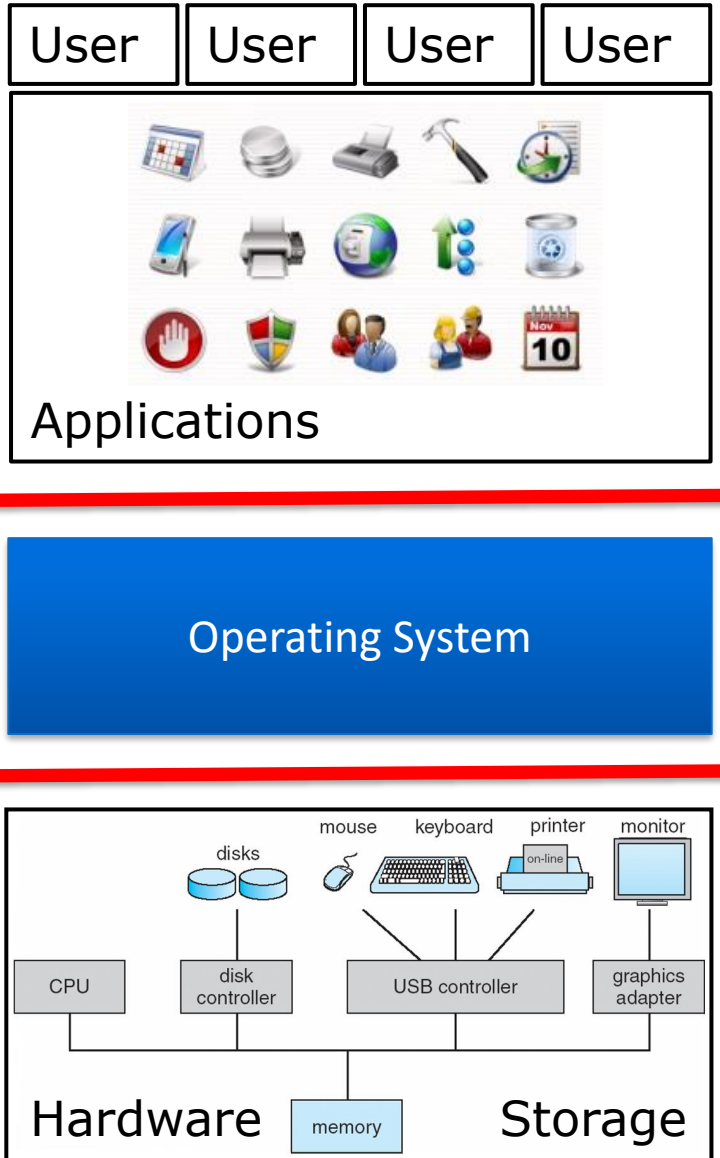
Transition from User to Kernel Mode

- Sometimes happens because something wrong happened: exception (e.g., division by 0)
- It means that the hardware (=CPU) cannot execute the instruction:
  - Runtime error (division by 0, …)
  - Programming error (illegal instruction, privileged instruction, …)
  - Bad memory address (segmentation violation)

Transition from User to Kernel Mode

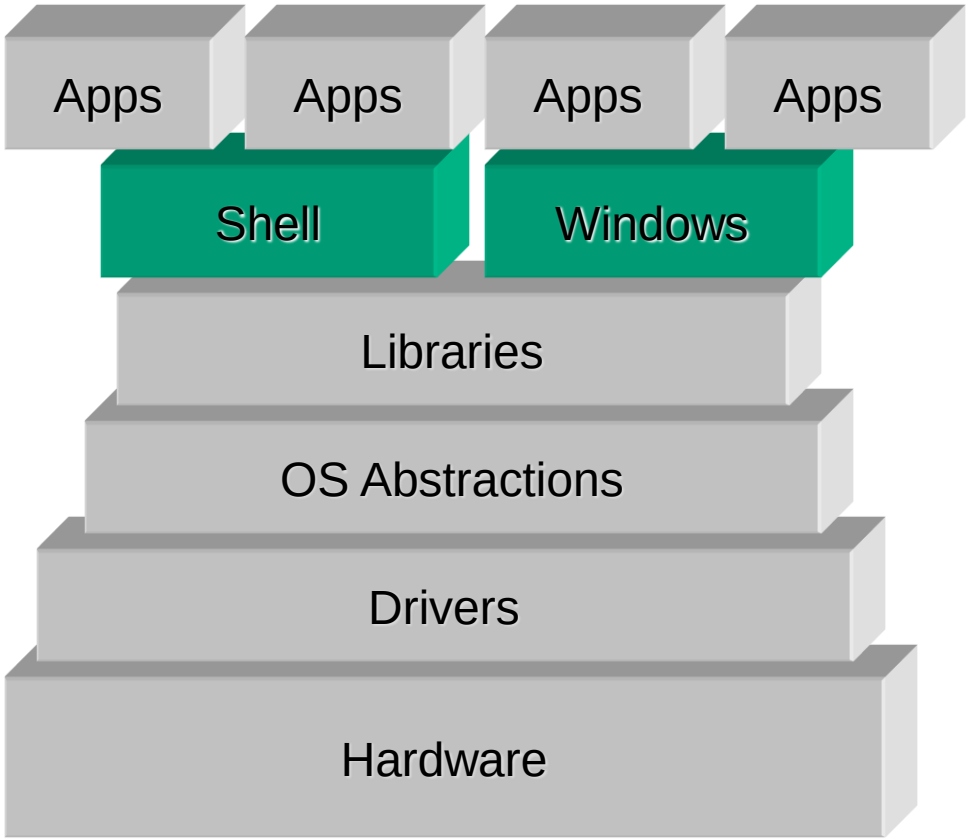
- Sometimes happens because something wrong happened: exception (e.g., division by 0)
- It means that the hardware (=CPU) cannot execute the instruction:
  - Runtime error (division by 0, …)
  - Programming error (illegal instruction, privileged instruction, …)
  - Bad memory address (segmentation violation)
- Exception handling is part of the CPU **architecture** (hardware)
  - Transfer the responsibility to the OS (and the user) to handle the exception condition → transition to kernel mode.

Simplified view of OS



Simplified view of OS

The reality is much more complicated…



User interface

- Make OS mechanisms available to user
- psychological issues are important

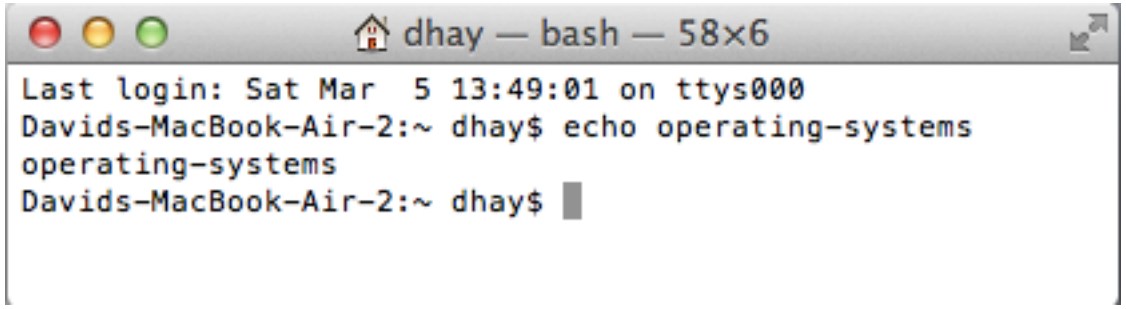
Is a web browser part of an Operating System ?

System Programs (or System Utilities)

- Modern OS comes along with system programs
  - Often an interface to a system call (run in user mode)
  - Sometimes more complex
  - Designed to provide service to other software
- Not to be confused with application programs
  - Sometimes also provided by the OS, but are not considered part of the OS
  - Run by users. Uses system programs to complete their operation
  - Example: Web browser

Example: echo system program

- echo: write argument to the standard output



Example: echo system program

- echo: write argument to the standard output

Example: echo system program

- echo: write argument to the standard output
- Linux implementation: 274 lines in c

Example: echo system program

- echo: write argument to the standard output
- Linux implementation: 274 lines in c
  - Uses fputs, putchar (multiple times)

Example: echo system program

- echo: write argument to the standard output
- Linux implementation: 274 lines in c
  - Uses fputs, putchar (multiple times)
  - … which in turn uses fwrite_unlocked

Example: echo system program

- echo: write argument to the standard output
- Linux implementation: 274 lines in c
  - Uses fputs, putchar (multiple times)
  - … which in turn uses fwrite_unlocked
  - … which in turn uses write – system call 0x80, which only understands bytes

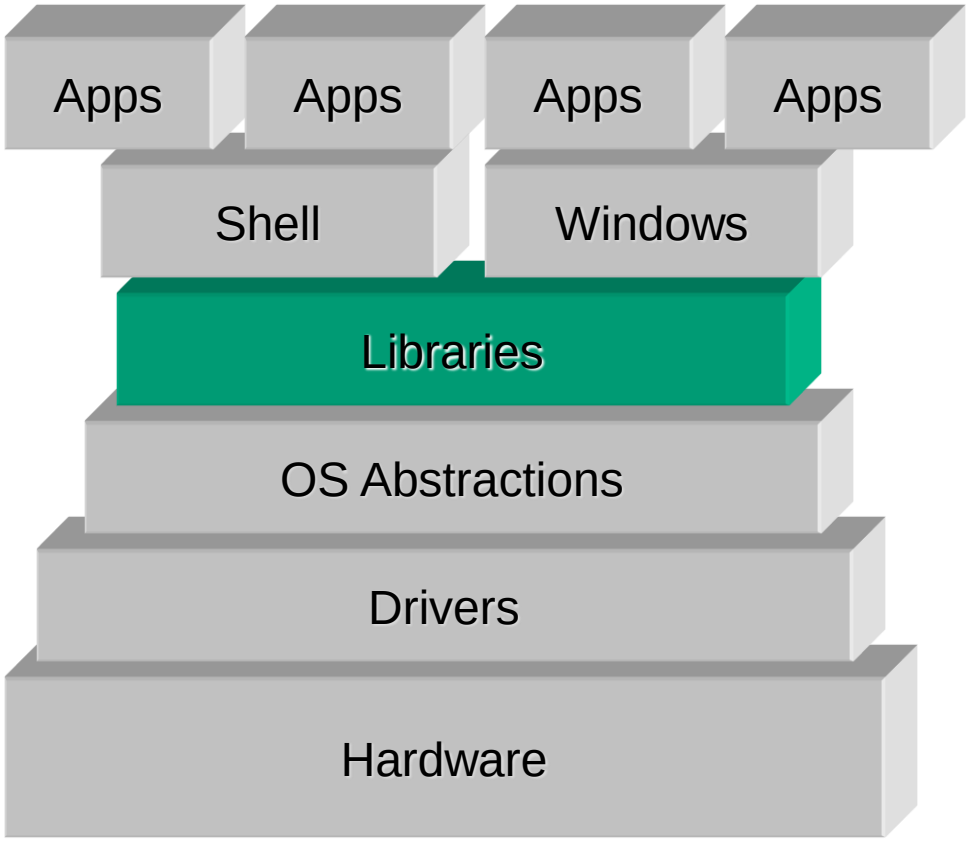
Example: echo system program

- echo: write argument to the standard output
- Linux implementation: 274 lines in c
  - Uses fputs, putchar (multiple times)
  - … which in turn uses fwrite_unlocked
  - … which in turn uses write – system call 0x80, which only understands bytes
- The write system call switches to kernel-mode, the rest is in user-mode

Example: echo system program

- echo: write argument to the standard output
- Linux implementation: 274 lines in c
  - Uses fputs, putchar (multiple times)
  - … which in turn uses fwrite_unlocked
  - … which in turn uses write – system call 0x80, which only understands bytes
- The write system call switches to kernel-mode, the rest is in user-mode
- Possibly several switches between the modes

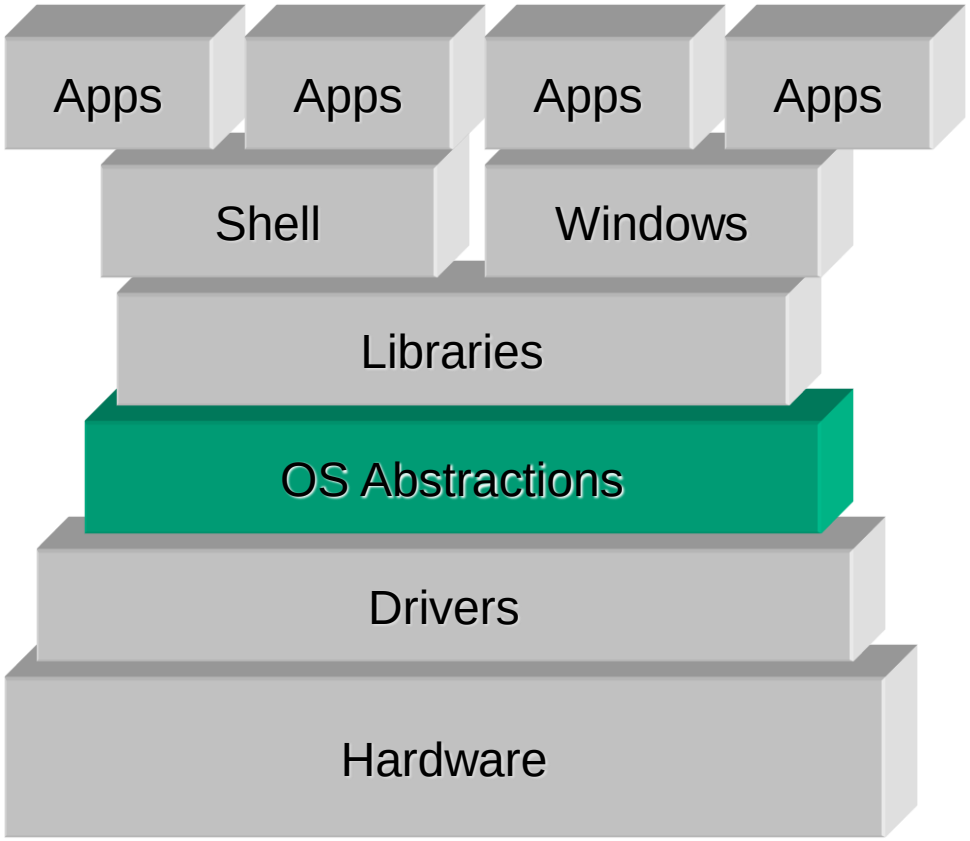
Layers of System



Libraries

- Usually language specific
  - java.io.*, java.net.*
  - stdio.h; stdlib.h
- Often higher level abstractions

Layers of System



OS Abstractions

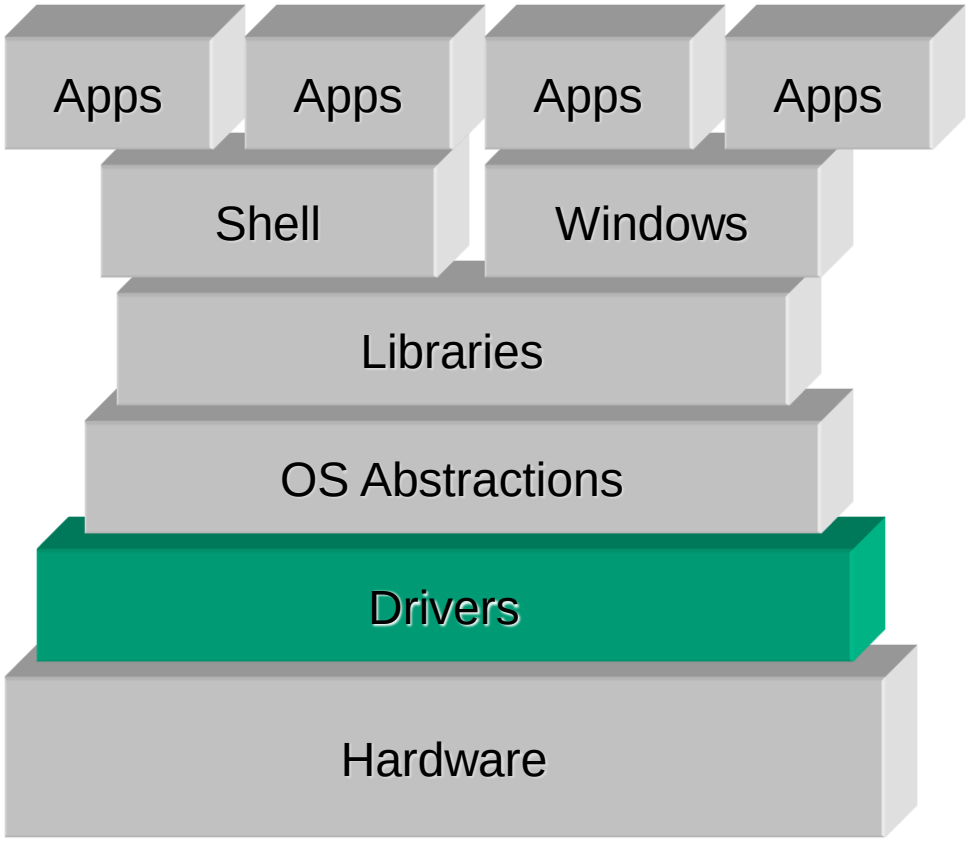
- provide lower level abstractions and mechanisms
- Storage
  - File systems
- Computation
  - processes
- Communications
  - sockets

Layers of System

OS Abstractions

- provide lower level abstractions and mechanisms
- Storage
  - File systems
- Computation
  - processes
- Communications
  - sockets

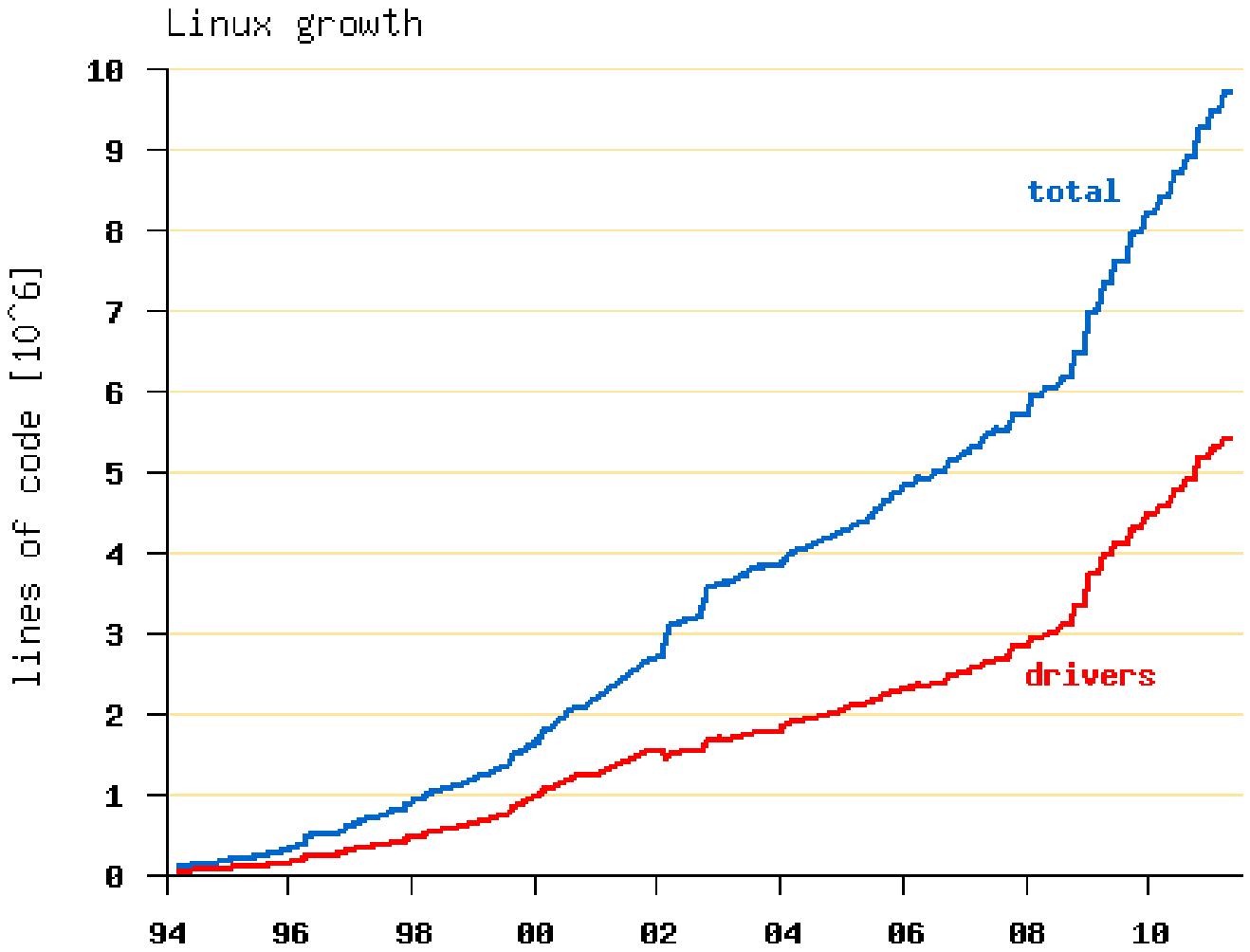
Layers of System



Hardware drivers

- provide usable interface to hardware
- A huge part of the system in terms of code volume
- Often written by device vendors

Drivers in Linux

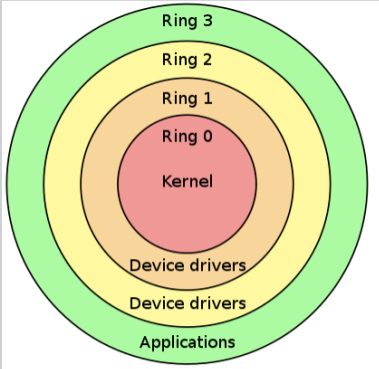


Drivers and Protection

- Drivers are part of the OS
- So they execute in kernel mode
- And have access to all OS data structures
  - Example: driver can change process priority
- Is this a good idea?

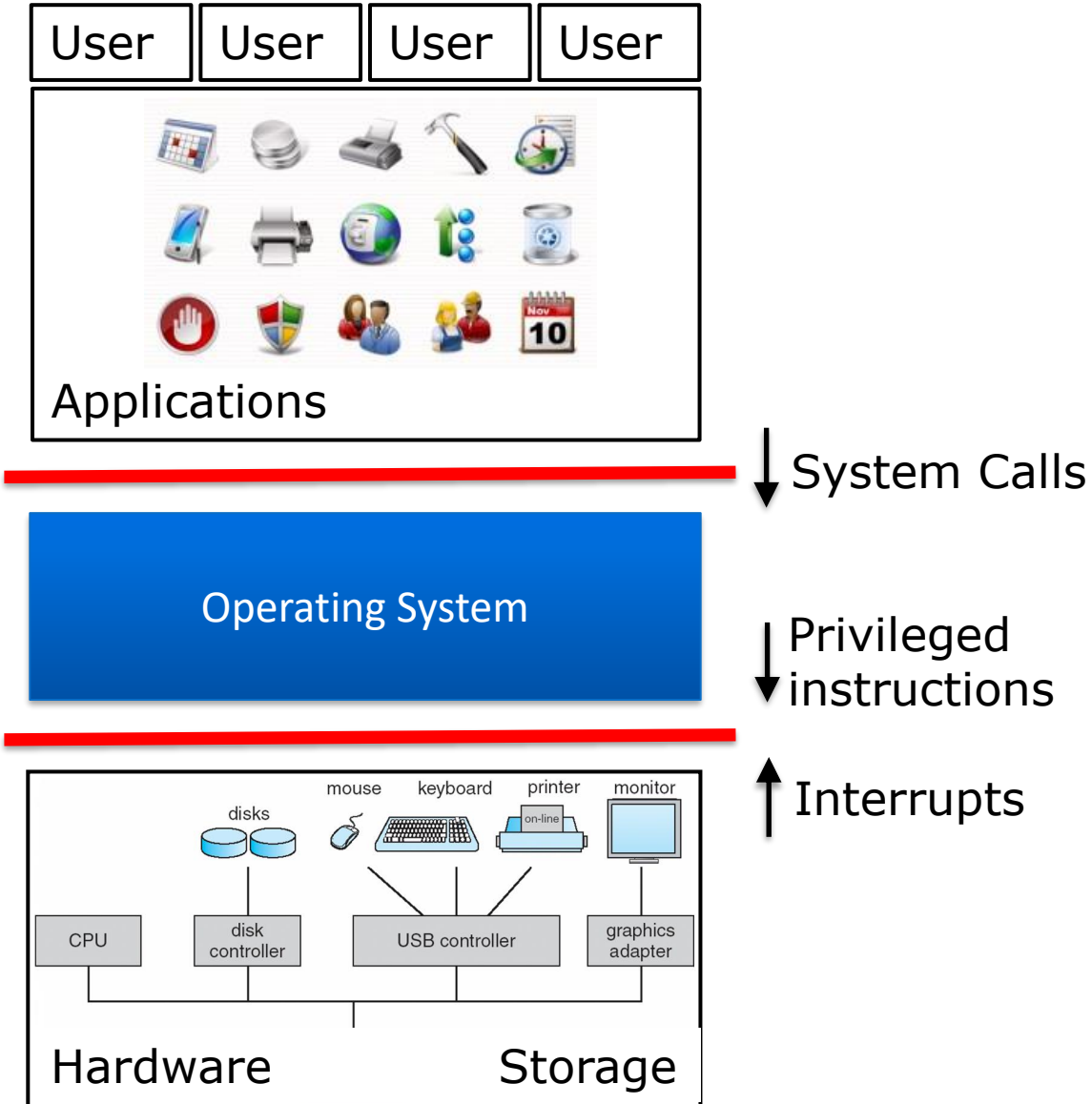
Drivers and Protection

- Drivers are part of the OS
- So they execute in kernel mode
- And have access to all OS data structures
  - Example: driver can change process priority
- Solution: use lower privileges
  - A level (levels) between kernel mode and user mode
  - Exists in most processors
  - Not used by most OS

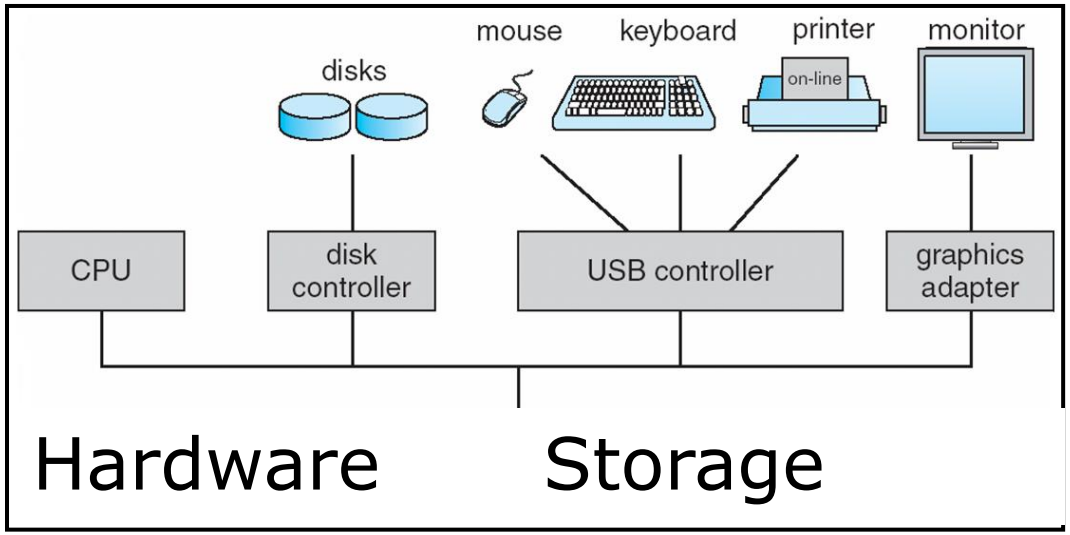


Privilege rings for the x86 available in protected mode

Simplified view of OS

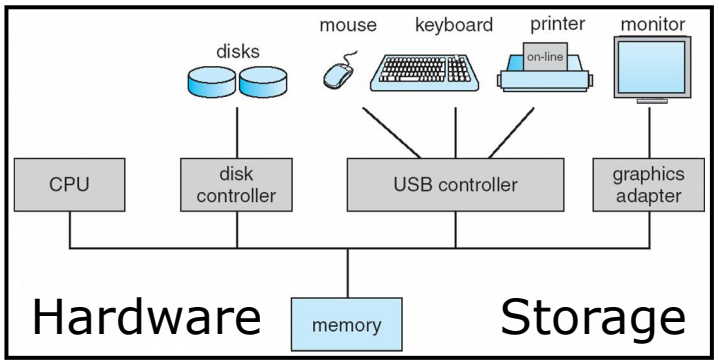


Simplified view of OS



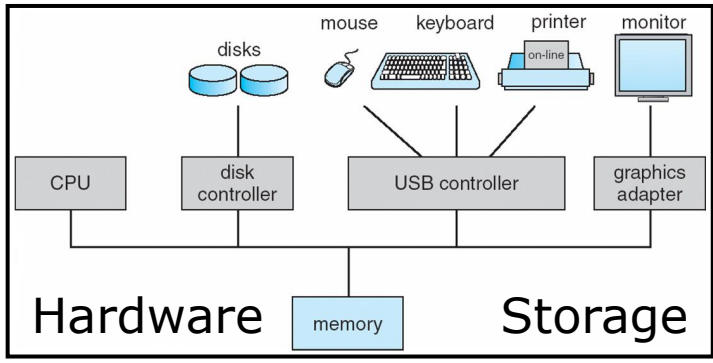
How to connect I/O devices? Bus

- Advantages:
  - Versatility:
    - New devices can be added easily
    - Peripherals can be moved between computer systems that use the same bus standard
  - Low Cost:
    - A single set of wires is shared in multiple ways

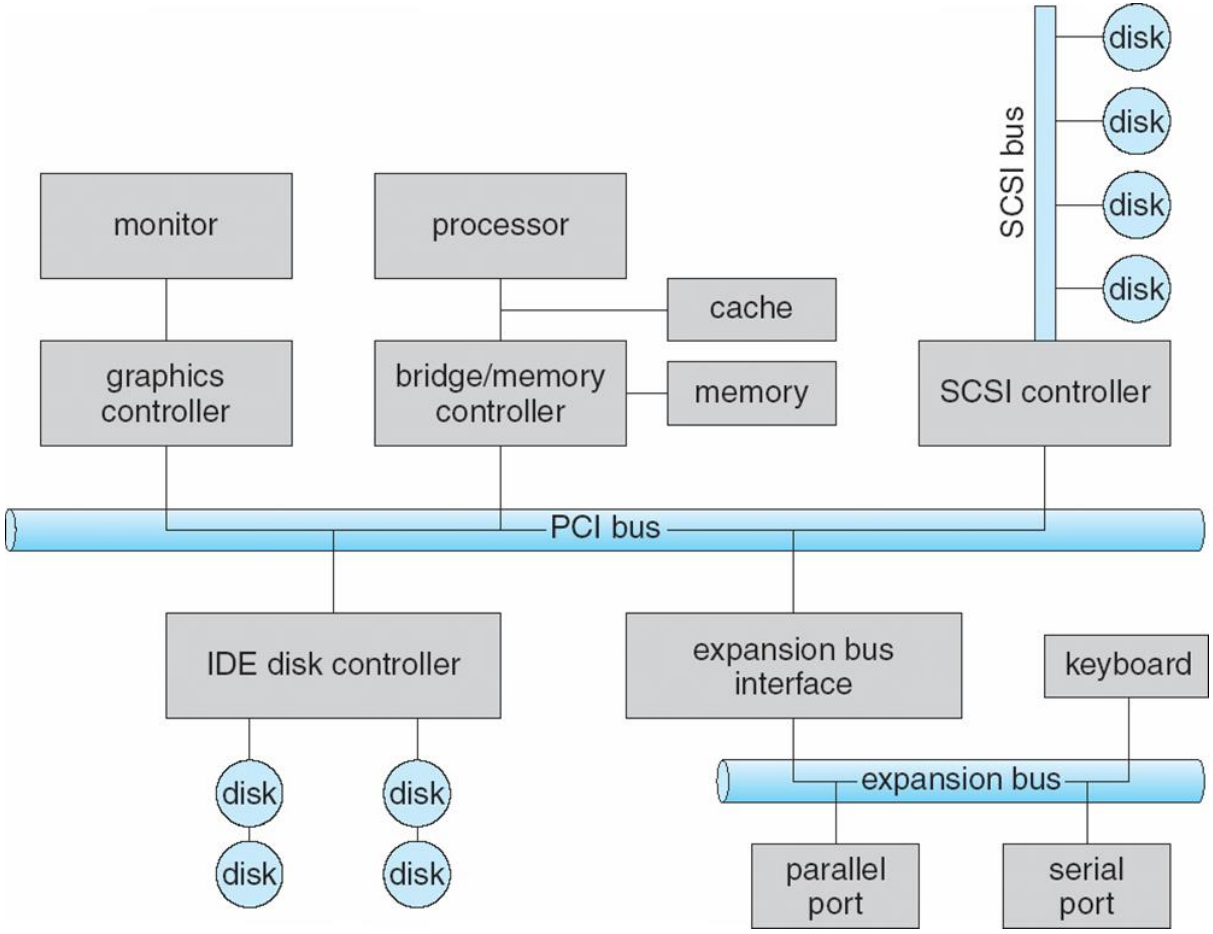


How to connect I/O devices? Bus

- Disadvantages:
  - It creates a communication **bottleneck**
    - The bandwidth of that bus can limit the maximum I/O throughput
  - The maximum bus speed is largely limited by:
    - The length of the bus
    - The number of devices on the bus
    - The need to support a range of devices with:
      - Widely varying latencies
      - Widely varying data transfer rates



Again, the reality is more complicated



How to communicate with I/O devices? Interrupts

- Devices have controllers with firmware
- Writing to certain bus addresses activates the controllers and transfers data to them
  - The driver for a device knows the specific details
- The controllers can then operate in the background (DMA)
- They then use interrupts to signal completion
- Also used by interactive devices
  - Every mouse click causes an interrupt
  - Every keystroke causes an interrupt

How to communicate with I/O devices? Interrupts

- Devices use interrupts to signal that they need service
  - Completion of an operation
  - Input from some external source (user, network)
- Cause the CPU to stop what it was doing
- And start running an OS function to handle the interrupt
- Just like exceptions and system calls

Interrupt Handling

- Upon an interrupt, the CPU needs to start running the correct interrupt handler
- This is part of the OS
- How does the hardware know where this OS function is located?
  - Hardware is hardwired – no flexibility
  - Maybe the OS was not even written when the CPU was manufactured!

The Interrupt Vector

- The solution: indirection via the interrupt vector
- The hardware defines an area in memory where it expects to find pointers to interrupt handlers
  - This is part of the architecture
  - Example: the first 1KB of memory (256 pointers)
- It is the responsibility of the OS to install pointers to the correct interrupt handlers in the cells of the interrupt vector
  - Happens during system boot and when new devices are installed
  - Defined as OS memory inaccessible to users

Interrupt Handling

- When an interrupt happens, the hardware will blindly load the address from the specified vector to the PC
- And atomically also set the mode bit to kernel mode
  - Identifying the device is part of the interrupt mechanism
  - Assigning IDs to devices is part of plug-and-play

Classifications of Interrupts

- Software vs. Hardware

  Software: causes by a software, this includes traps and exceptions.

  Hardware: caused by hardware, this includes interrupts from external devices.

- Internal vs. External

  Generally: Internal=Software, External=Hardware

- Synchronous vs. Asynchronous

  Synchronous: happens with the clock ticks (software, timer/clock interrupts).

- Maskable vs. Non-maskable

  Maskable: hardware interrupts that can be delayed if more important interrupts is currently handled.

- Periodic vs. Aperiodic

Examples of Interrupts

INT_NUM	Short Description PM [*clarification needed*]
0x00	Division by zero
0x01	Single-step interrupt (see trap flag)
0x02	NMI
0x03	Breakpoint (callable by the special 1-byte instruction 0xCC, used by debuggers)
0x04	Overflow
0x05	Bounds
0x06	Invalid Opcode
0x07	Coprocessor not available
0x08	Double fault
0x09	Coprocessor Segment Overrun *(386 or earlier only)*
0x0A	Invalid Task State Segment
0x0B	Segment not present
0x0C	Stack Fault
0x0D	General protection fault
0x0E	Page fault
0x0F	*reserved*
0x10	Math Fault
0x11	Alignment Check
0x12	Machine Check
0x13	SIMD Floating-Point Exception
0x14	Virtualization Exception
0x15	Control Protection Exception

x86 Architecture:

- there are 256 interrupt vectors
- First 32 are reserved

← Some examples

0x80 is used for system calls' traps

Disabling (Masking) Interrupts

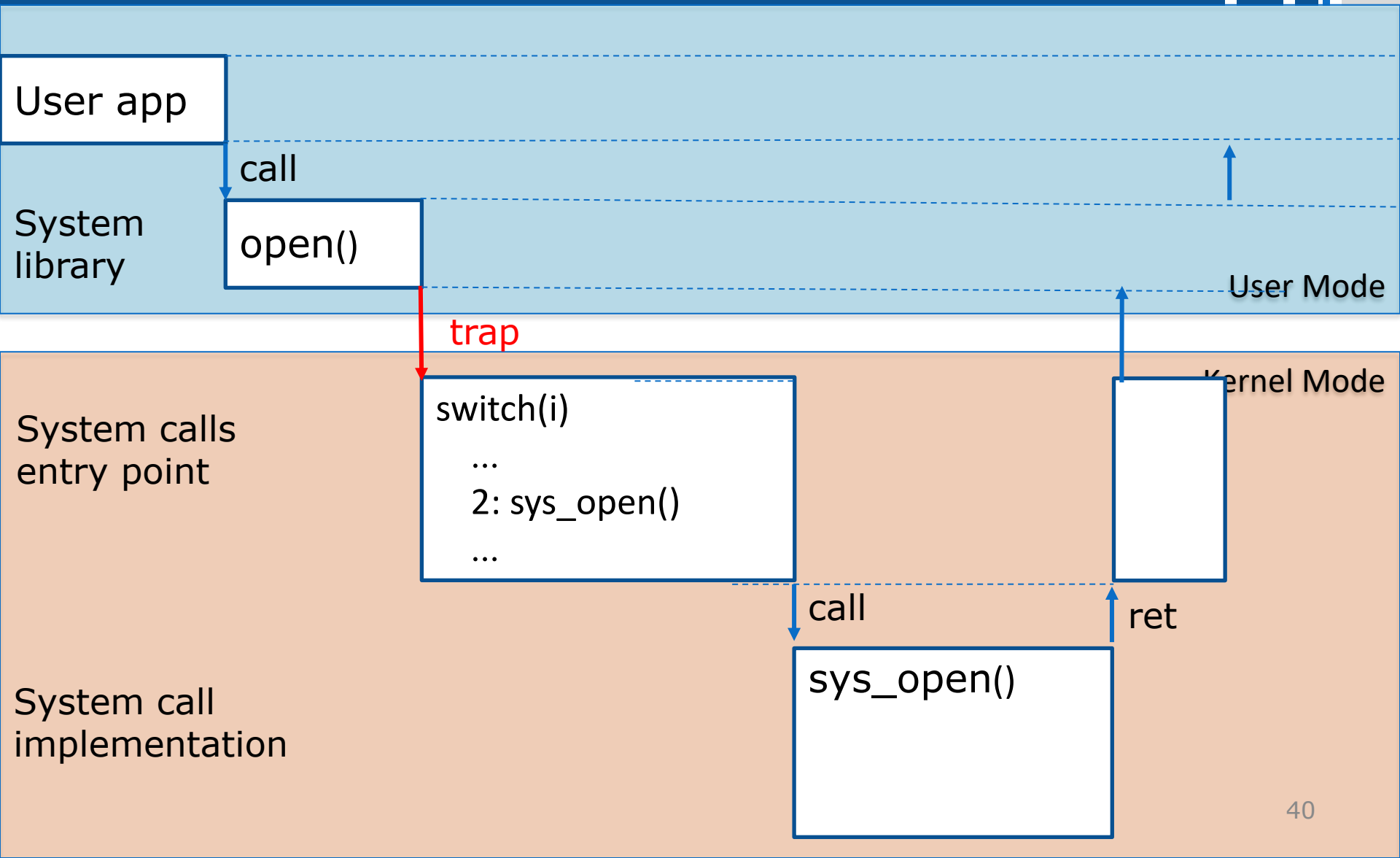
- An operating system is **interrupt driven**
  - Comes with being a reactive program
- Incoming interrupts are generally *disabled* while another interrupt is being processed to prevent a *lost interrupt*
  - But a few interrupts cannot be delayed…
- The OS can also disable interrupts so it won't be interrupted in the middle of critical work

Summary: Three ways to communicate with OS

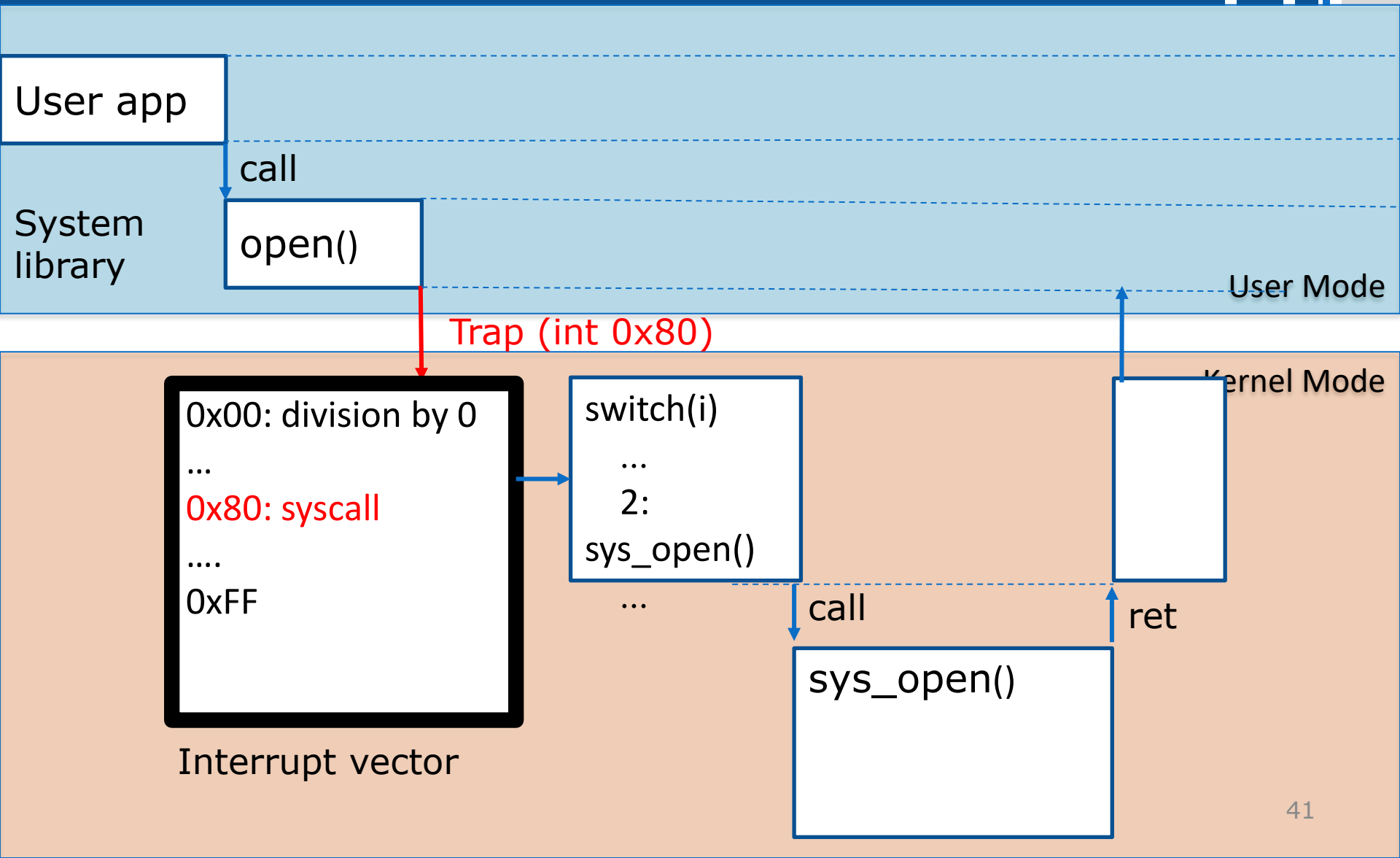
- **Interrupts** (a.k.a. hardware interrupts)
- **System calls**
  - that cause **traps** (a.k.a. software interrupts)
- **Exceptions** (when something wrong happens, also considered software interrupts)

Sometimes hardware interrupts, traps and exceptions are collectively called **interrupts**
Technically, the all use the same mechanism (the interrupt vector)

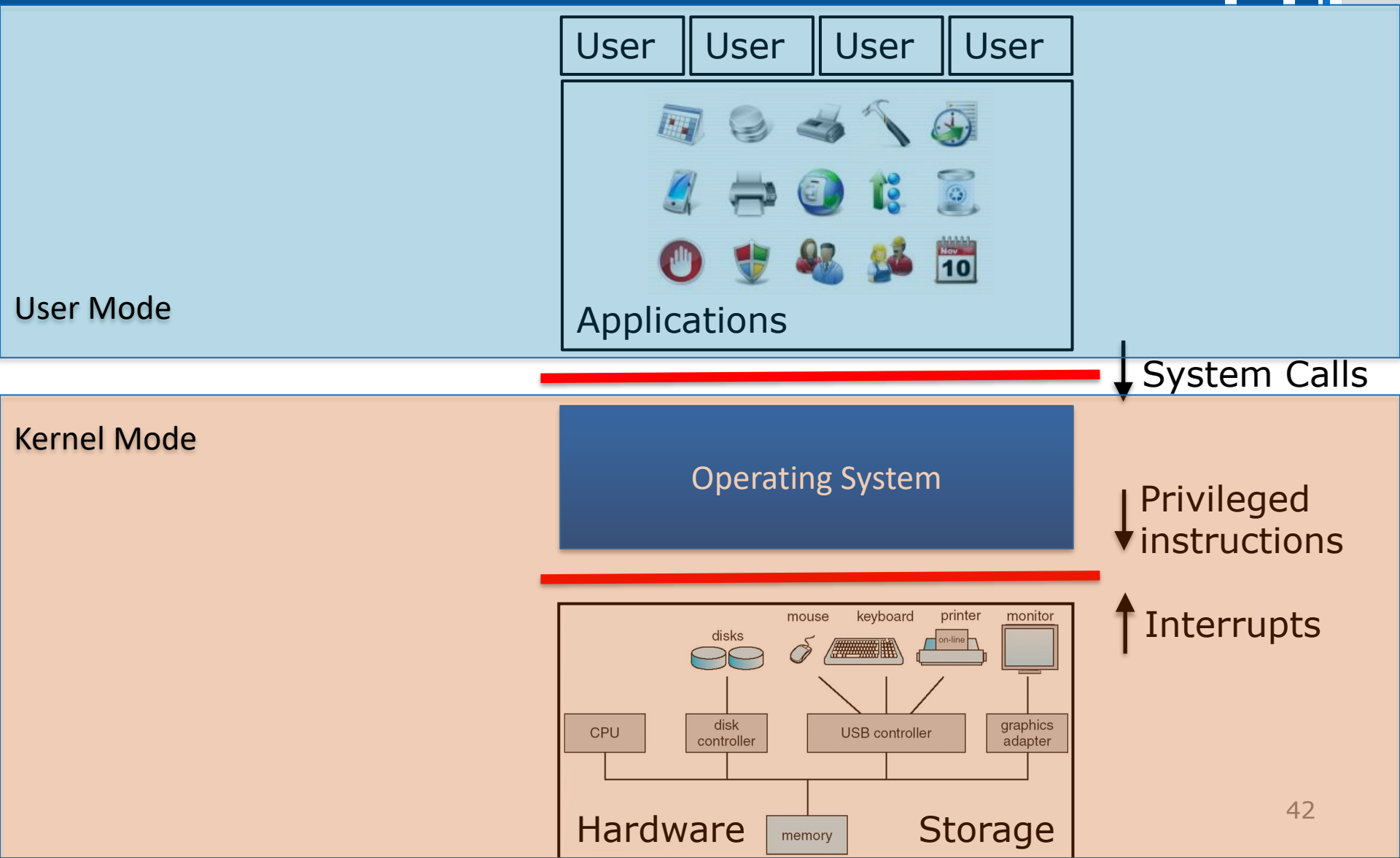
Trap in Detail



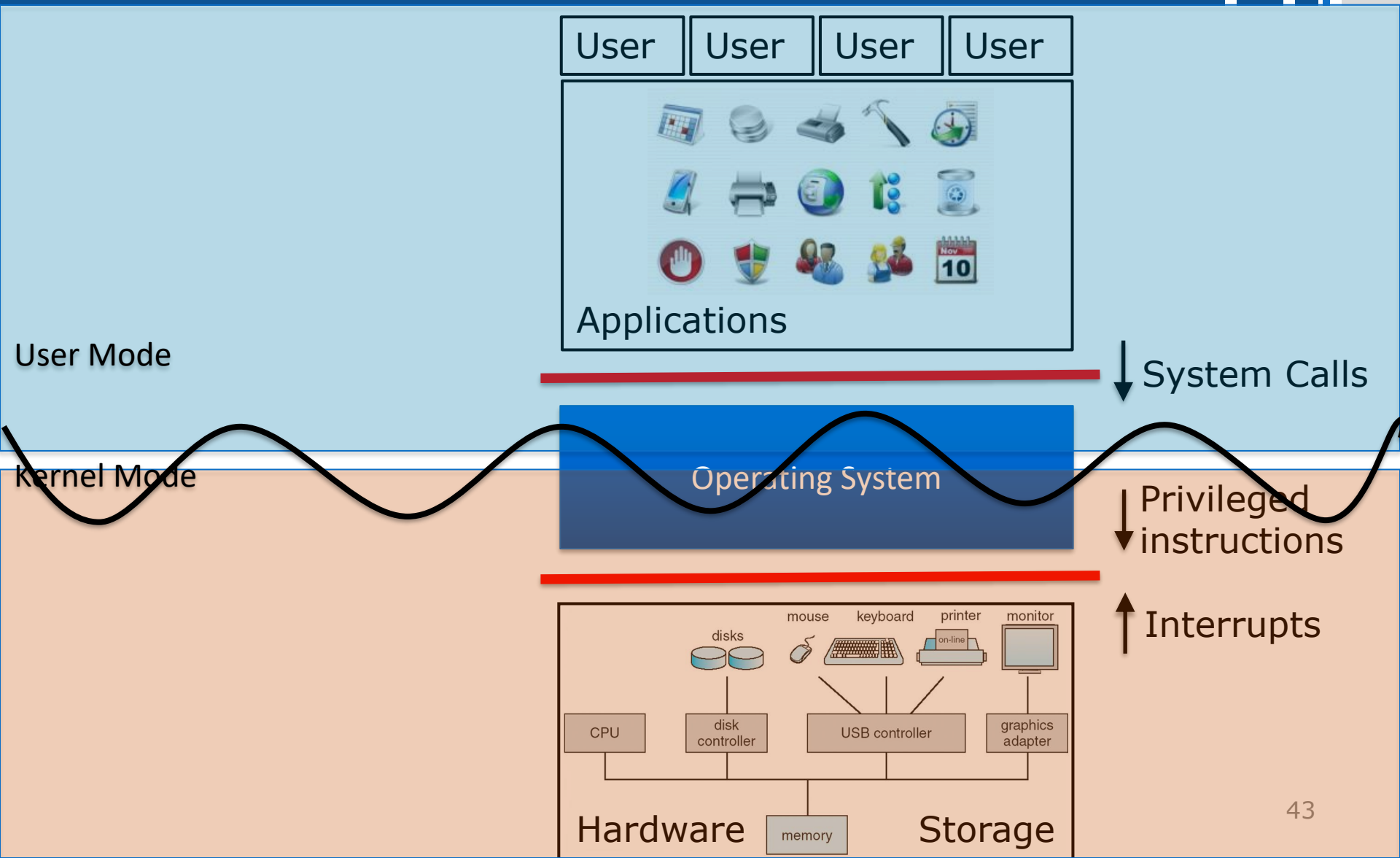
Trap in Detail



Kernel and User Modes revisited



Kernel and User Modes revisited



Trade-offs

- Smaller kernel:
  -  Less traps → less overhead → higher throughput
  -  Less protection

Trade-offs

- Smaller kernel:
  -  Less traps → less overhead → higher throughput
  -  Less protection
- Smaller kernel → smaller memory footprint

Trade-offs

- Smaller kernel:
  -  Less traps → less overhead → higher throughput
  -  Less protection
- Smaller kernel → smaller memory footprint
- Smaller kernel → more functions outside kernel → if they change, no need to recompile the kernel

Type of Kernels

Type of Kernels

- Monolithic:
  - MS-DOS ,(Old) Unix

Type of Kernels

- Monolithic:
  - MS-DOS ,(Old) Unix
- Modular Monolithic (Loadable Kernel Modules):
  - Mac OS X, Windows, Linux, Solaris, (modern) Unix

Type of Kernels

- Monolithic:
  - MS-DOS ,(Old) Unix
- Modular Monolithic (Loadable Kernel Modules):

dhay — bash — 171×50

```
Davids-MacBook-Air-2:~ dhay$ kextstat
Index Refs Address            Size       Wired      Name (Version) <Linked Against>
    1   70 0xffffff7f80757000 0x686c     0x686c     com.apple.kpi.bsd (12.6.0)
    2    6 0xffffff7f80742000 0x46c      0x46c      com.apple.kpi.dsep (12.6.0)
    3   94 0xffffff7f80761000 0x1b7ec    0x1b7ec    com.apple.kpi.iokit (12.6.0)
    4   98 0xffffff7f8074d000 0x99f8     0x99f8     com.apple.kpi.libkern (12.6.0)
    5   86 0xffffff7f80743000 0x88c      0x88c      com.apple.kpi.mach (12.6.0)
    6   38 0xffffff7f80744000 0x502c     0x502c     com.apple.kpi.private (12.6.0)
    7   57 0xffffff7f8074a000 0x23cc     0x23cc     com.apple.kpi.unsupported (12.6.0)
    8    0 0xffffff7f81806000 0x41000    0x41000    com.apple.kec.corecrypto (1.0) <7 6 5 4 3 1>
    9   22 0xffffff7f80d17000 0x9000     0x9000     com.apple.iokit.IOACPIFamily (1.4) <7 6 4 3>
   10   30 0xffffff7f80851000 0x29000    0x29000    com.apple.iokit.IOPCIFamily (2.8) <7 6 5 4 3>
   11    2 0xffffff7f825f6000 0x5a000    0x5a000    com.apple.driver.AppleACPIPlatform (1.8) <10 9 7 6 5 4 3 1>
   12    1 0xffffff7f80a72000 0xe000     0xe000     com.apple.driver.AppleKeyStore (28.21) <7 6 5 4 3 1>
   13    6 0xffffff7f8077d000 0x25000    0x25000    com.apple.iokit.IOStorageFamily (1.8) <7 6 5 4 3 1>
   14    0 0xffffff7f80dd8000 0x19000    0x19000    com.apple.driver.DiskImages (345) <13 7 6 5 4 3 1>
   15    0 0xffffff7f822dd000 0x2e000    0x2e000    com.apple.driver.AppleIntelCPUPowerManagement (214.0.0) <7 6 5 4 3 1>
   16    0 0xffffff7f8075e000 0x3000     0x3000     com.apple.security.TMSafetyNet (7) <7 6 5 4 2 1>
   17    2 0xffffff7f807f3000 0x4000     0x4000     com.apple.kext.AppleMatch (1.0.0d1) <4 1>
   18    1 0xffffff7f807f7000 0x11000    0x11000    com.apple.security.sandbox (220.4) <17 7 6 5 4 3 2 1>
   19    0 0xffffff7f80808000 0x6000     0x6000     com.apple.security.quarantine (2.1) <18 17 7 6 5 4 2 1>
```

Type of Kernels

- Monolithic:
  - MS-DOS ,(Old) Unix
- Modular Monolithic (Loadable Kernel Modules):
  - Mac OS X, Windows, Linux, Solaris, (modern) Unix

Type of Kernels

- Monolithic:
  - MS-DOS ,(Old) Unix
- Modular Monolithic (Loadable Kernel Modules):
  - Mac OS X, Windows, Linux, Solaris, (modern) Unix
- Microkernels :
  - Mach

NEXT SUBJECT: PROCESS MANAGEMENT