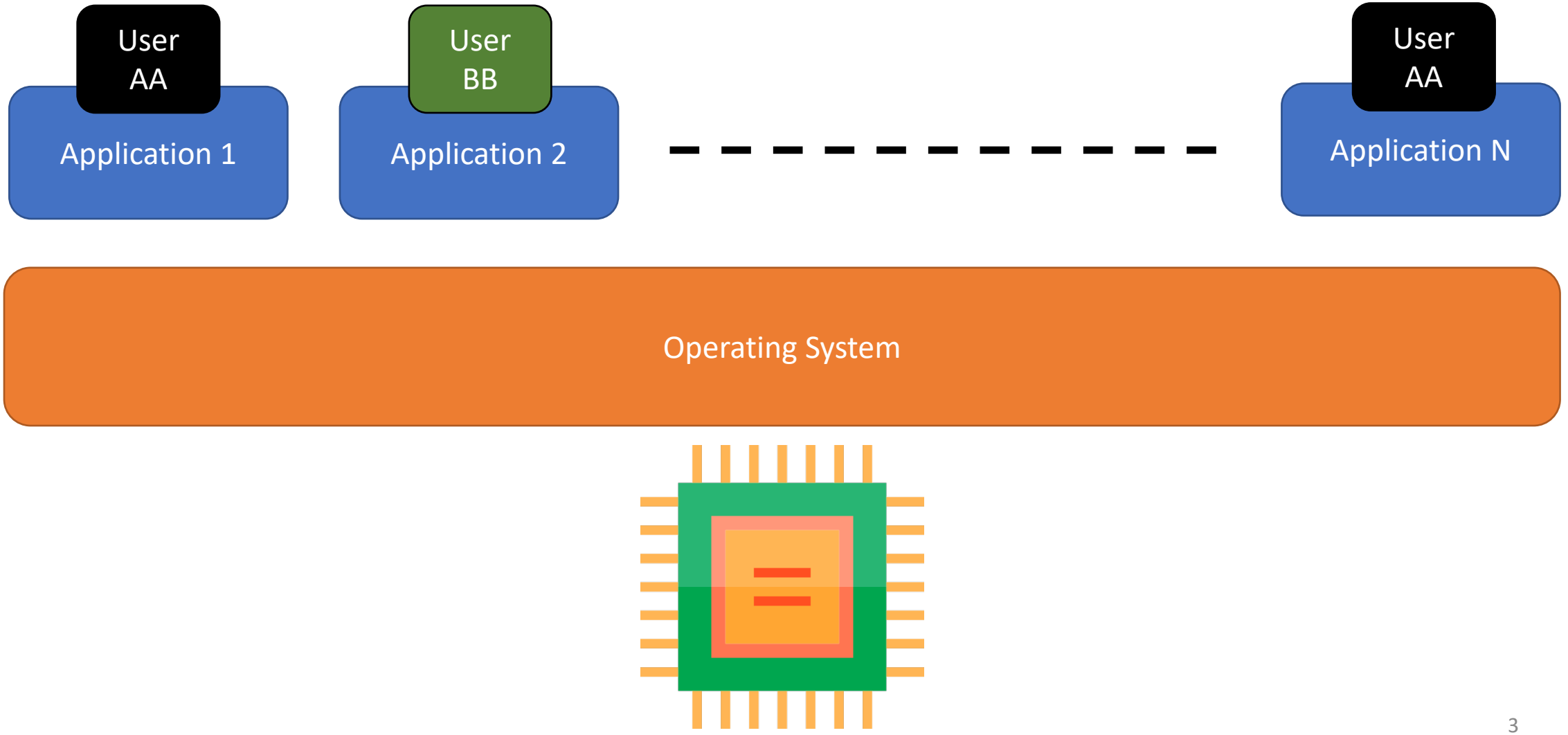
# CS 1217

Lecture 7 : Privilege Levels, Interrupts and Exceptions

# Logistics

- Assignment 3 is out today; will be due Monday
- Start asap!
- Office hours this week for browsing xv6 code

# Virtualizing the CPU



# Virtualizing the CPU

- CPU is a resource that is used by all processes
  - To keep discussion simple, we will assume a single core for now
- The processes could be from multiple users
  - Need to give each user a fair share of the resource
- This resource needs to be shared across multiple processes that are running “concurrently”
- The processes might want the kernel to carry out ***privileged*** operations on their behalf
  - Reading from disk, getting packet from network etc.

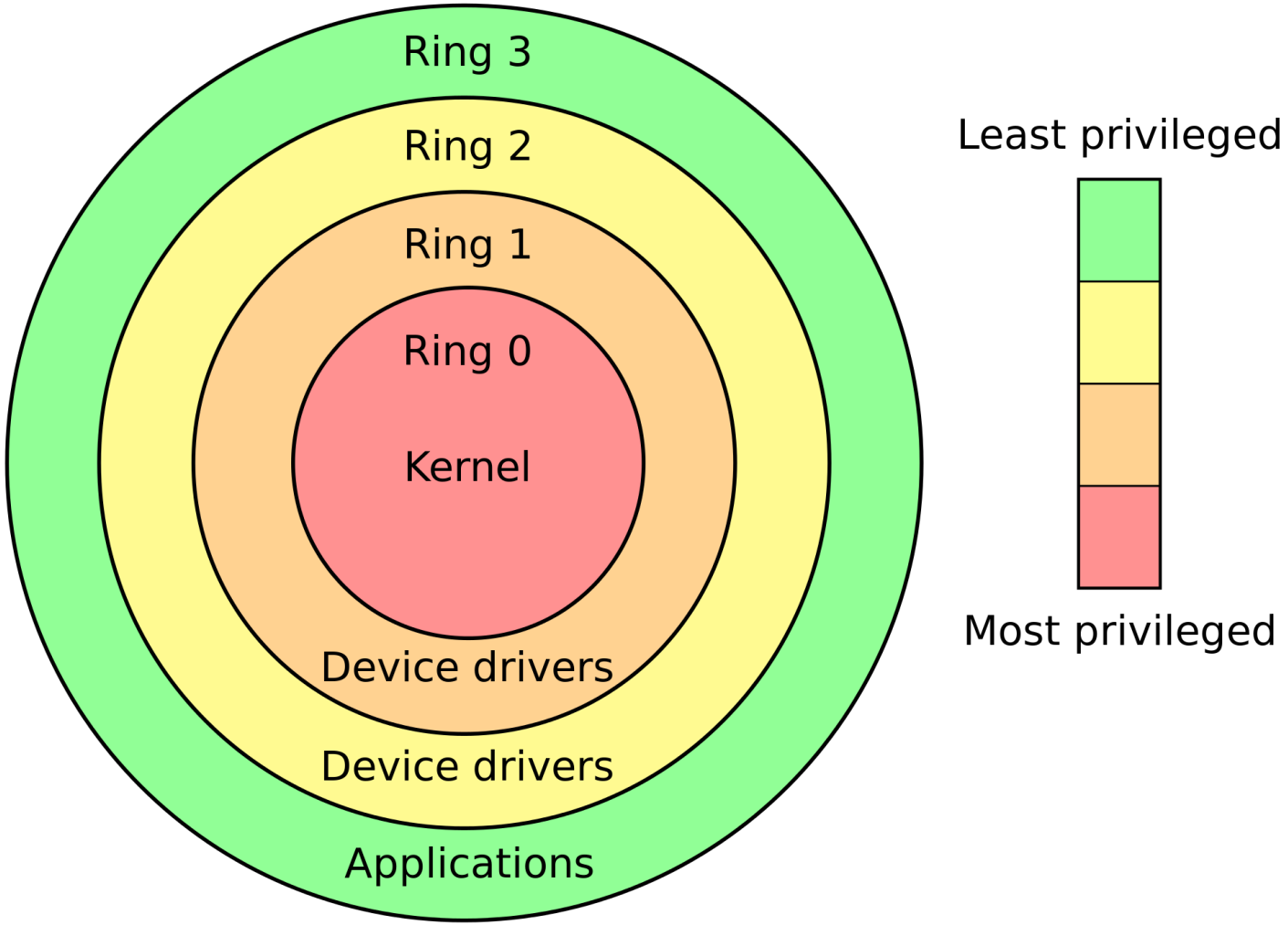
# Kernel Privileges

- Kernel has the job of multiplexing resources across multiple processes
- It needs to have special privileges over the hardware
  - Should be able to do things that “normal” processes cannot
- Kernel is the ultimate arbiter
  - Needs to make sure that ***malicious*** processes are not able to cause harm
  - Needs to make sure that processes running ***buggy programs*** are not able to cause harm

# Kernel Privileges : OS Rings

- CPU provides support for multiple ***privilege*** levels, user and kernel

- Kernel mode allows
  - for execution of special instructions
  - A *different* view of memory



# Example of Special Instructions

- Special instructions allow for change of state and about how resources can be shared
- Putting something in CR3 register
  - CR3 contains the physical address of the base of the paging-structure
- HLT instruction : Stops instruction execution and places the processor in a HALT state, until an interrupt happens
- Could be disastrous if any process was able to execute any of these instructions.

# Kernel Privilege

- The ability of the kernel to be able to execute a much larger set of instructions than a “regular” process
- Guaranteed to work by the processor architecture itself.
- Remember: Kernel itself is a process; has special permissions to execute all instructions : why?

# The OS Privilege

- Special code runs in the kernel mode – code for the OS
  - Trusted for managing system resources
- Everything else runs in the user mode
  - Not trusted; has to rely on kernel code for using system resources
- OS Kernel also implements the functionality for switching between the two states.

# How does the OS get the privilege?

- The OS is installed that way: One program is granted the privileges to execute the special instructions
- How does the OS get this privilege?
- On boot:
  - The CPU starts out in the privileged the state; the OS is executed in this mode
- Then, how do you execute applications in user mode?
  - The kernel needs to **lower** the privilege before executing user level or application code

# Trapping into the Kernel

- When the application wants the kernel to do something on its behalf, it ***traps*** into the kernel

- The transition into the kernel or into privileged mode occurs for one of three reasons (typically):
- a hardware device requests attention—**hardware interrupt**
- software requests attention—**software interrupt** or system call
- software needs attention—**software exception**
  - illegal instructions, protection fault, divide-by-0, page fault

# Interrupt Handling

- When an interrupt is triggered (interrupt request, IRQ), the processor:

1. **enters privileged mode**,
2. **records state** necessary to process the interrupt,
3. **jumps to a pre-determined memory location** and begins executing instructions.

- The instructions that the processor executes when an interrupt fires are called the **interrupt service routine** (ISR).

# Example: Making Syscalls

- When an application makes a syscall:

- Needs to place the arguments to the system call at an agreed upon location where the kernel can find them
- Pass information to the kernel about which syscall it wants executed
  - Done by writing a number in a predefined register
- Executes a special instruction, which lets the kernel know that a syscall needs to be executed
  - `int` in x86

# Hardware Interrupts

- A device proactively interrupts the CPU because it needs service or attention

- Examples of hardware interrupts
  - Mouse clicks, keyboard presses
  - Disk reads being completed
  - Network card receiving data packet(s)
  - Timer interrupts: will be useful later

# Software Interrupts

- Applications need a mechanism for “asking” the operating system to carry out privileged operations on its behalf
- CPU ISAs provide a **special instruction** (`syscall` on MIPS, `int` on x86) that generates a software (or synthetic) interrupt.
- Rest of the interrupt handling path is unchanged

# Software Interrupts vs. Software Exceptions

- **Interrupts** are **voluntary**.
  - For example, The process wants to read some data from disk that it needs to make forward progress

- **Exceptions** are **non-voluntary**.
  - Buggy code was written that has now resulted in a divide by zero exception; the process didn't request help, but has to now be terminated