CS330: Operating Systems

Limited direct execution

Recap: virtual view of resources

- Process
  - Each running process thinks that it owns the CPU

Recap: virtual view of resources

- Process
  - Each running process thinks that it owns the CPU
- Address space
  - Each process feels like it has a huge address space

Recap: virtual view of resources

- Process
  - Each running process thinks that it owns the CPU
- Address space
  - Each process feels like it has a huge address space
- File system tree
  - The user feels like operating on the files directly

Recap: virtual view of resources

- Process
  - Each running process thinks that it owns the CPU
- Address space
  - Each process feels like it has a huge address space
- File system tree
  - The user feels like operating on the files directly
- What are the OS responsibilities in providing the above virtual notions?

Recap: virtual view of resources

- Process
  - Each running process thinks that it owns the CPU
- Address space
  - Each process feels like it has a huge address space
- File system tree
  - The user feels like operating on the files directly
- What are the OS responsibilities in providing the above virtual notions?
  - The OS performs multiplexing of physical resources efficiently
  - Maintains mapping of virtual view to physical resource

Virtualization: Efficiency/performance

- Resource virtualization should not add *excessive* overheads
- Efficient when programs use the resources directly, no OS mediation
  - Example: when a process is scheduled on a CPU, it should execute without OS intervention
- What is the catch?

Virtualization: Efficiency/performance

- Resource virtualization should not add any overheads
- Efficient when programs use the resource directly, infrequent OS mediation
  - Example: when a process is scheduled on CPU, it should execute without OS intervention
- What is the catch?
  - Loss of control e.g., process running an infinite loop on a CPU
  - Isolation issues e.g., process accessing/changing OS data structures

Virtualization: Efficiency/performance

- Resource virtualization should not add any overheads
- Efficient when programs use the resource directly, infrequent OS mediation
  - Example: when a process is scheduled on CPU, it should execute without OS intervention
- What is the catch?
  - Loss of control e.g., process running an infinite loop on a CPU
  - Isolation issues e.g., process accessing/changing OS data structures

Conclusion: Some limits to direct access must be enforced.

Limited direct execution

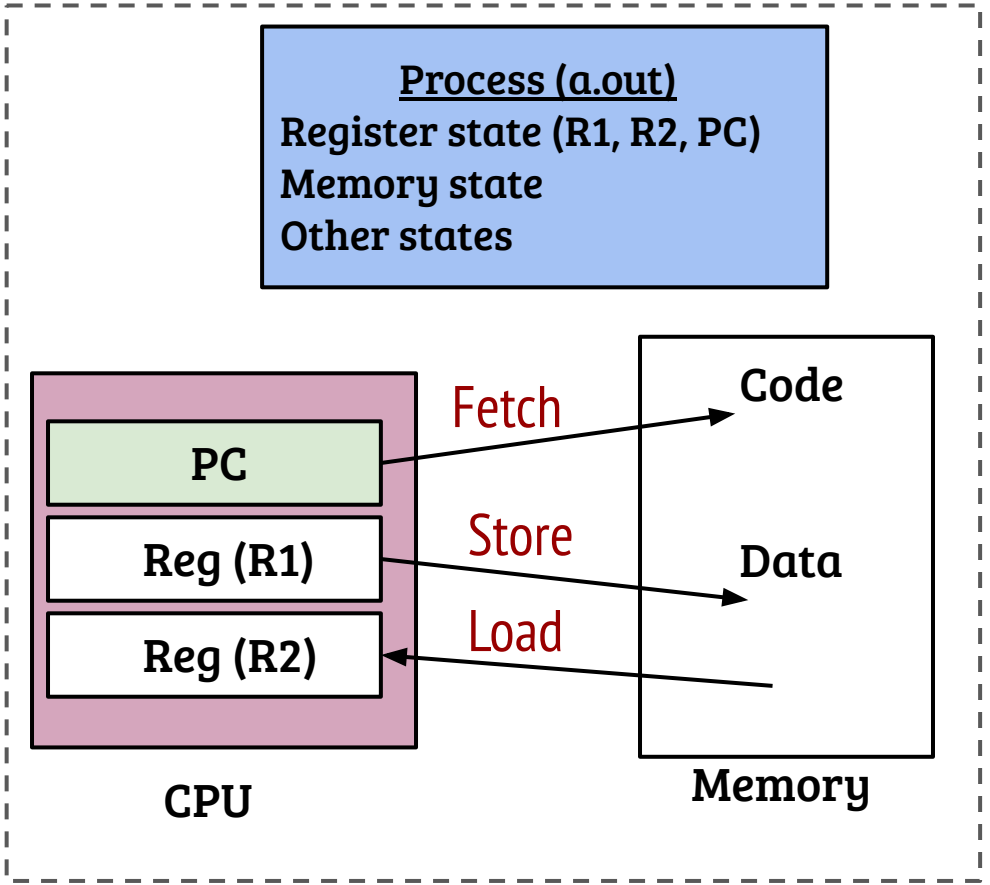
- Can the OS enforce limits to an executing process by itself?

A process in execution

I want to take control of the CPU from this process which is executing an infinite loop, but how?

OS

I want to restrict this process accessing memory of other processes, but how? Monitoring each memory access is not efficient!



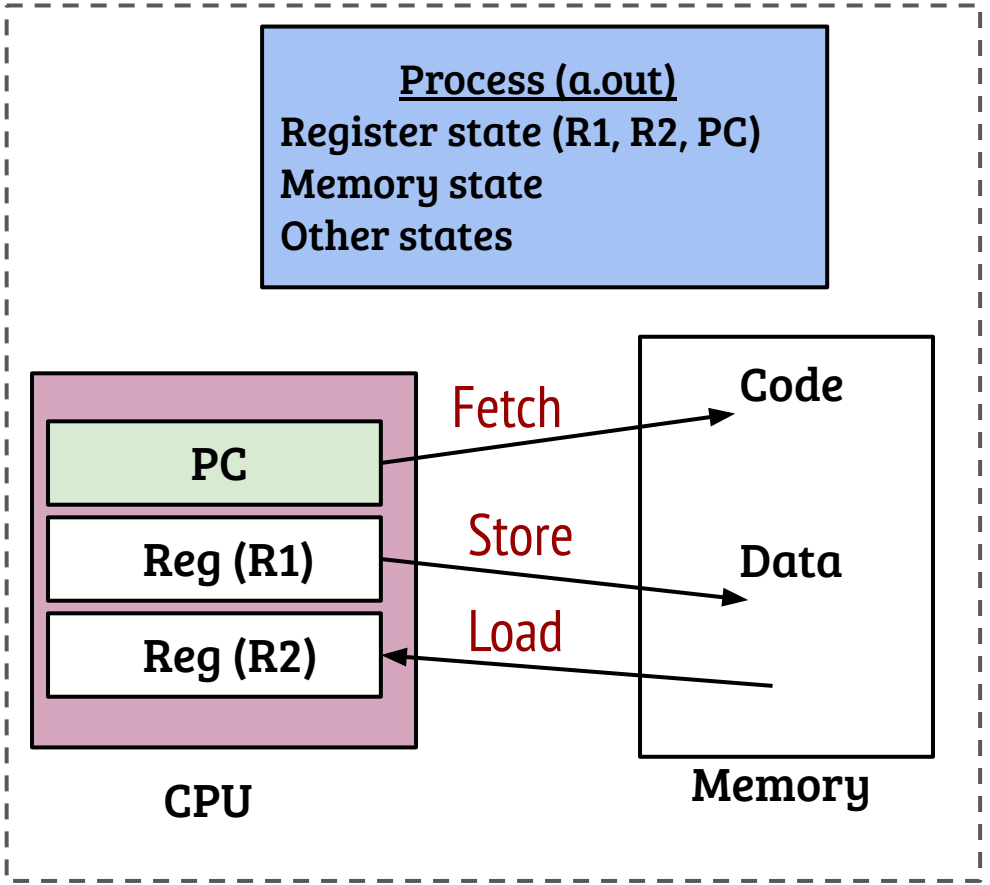
A process in execution

I want to take control of the CPU from this process which is executing an infinite loop, but how?

Help me!

OS

I want to restrict this process accessing memory of other processes, but how? Monitoring each memory access is not efficient!



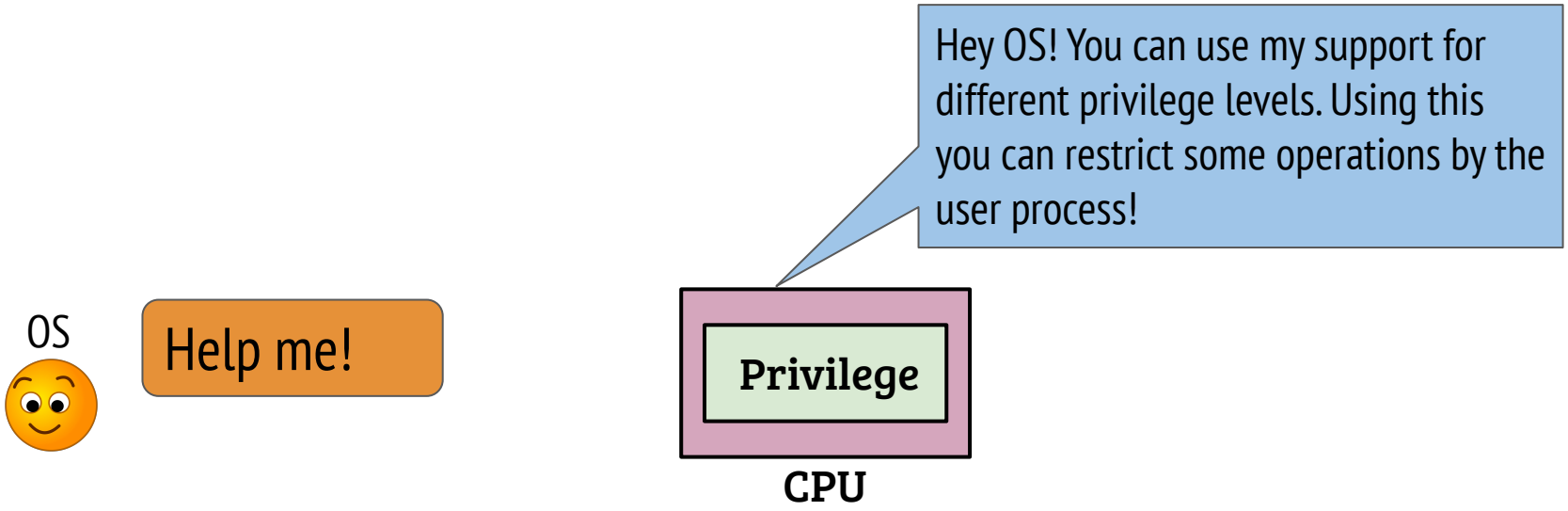
Limited direct execution

- Can the OS enforce limits to an executing process by itself?
- No, the OS can not enforce limits by itself and still achieve efficiency
- OS requires support from hardware!

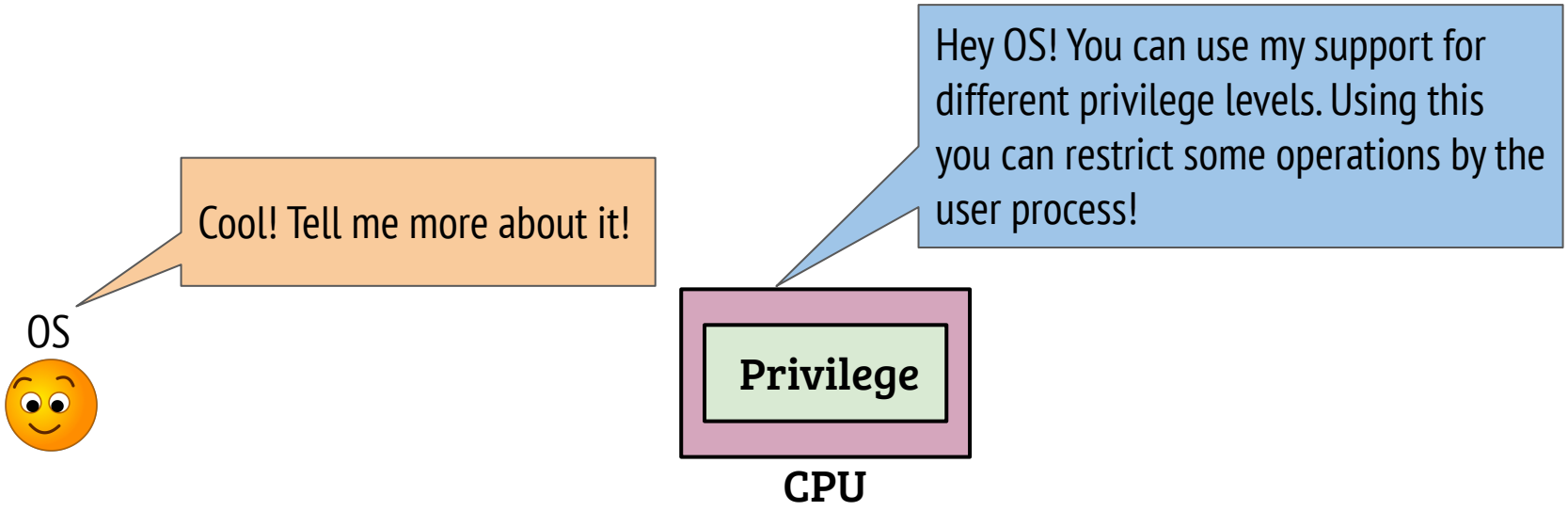
Limited direct execution

- Can the OS enforce limits to an executing process?
- No, the OS can not enforce limits by itself and still achieve efficiency
- OS requires support from hardware!
- What kind of support is needed from the hardware?

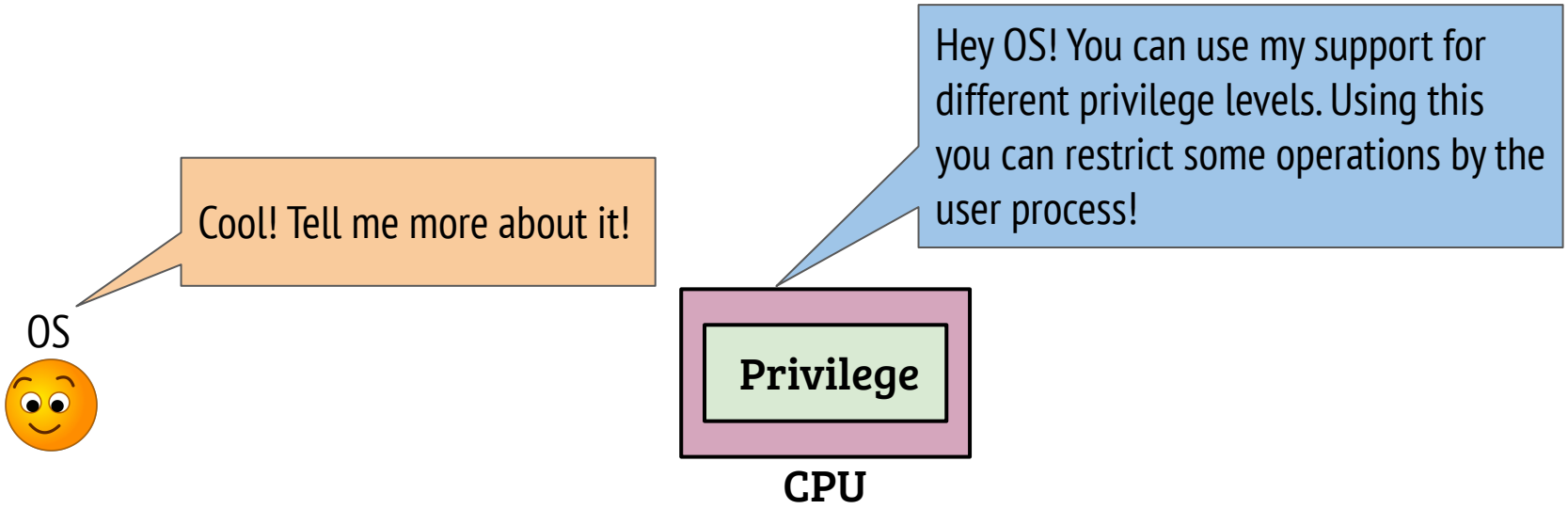
Hardware support: Privilege levels



Hardware support: Privilege levels

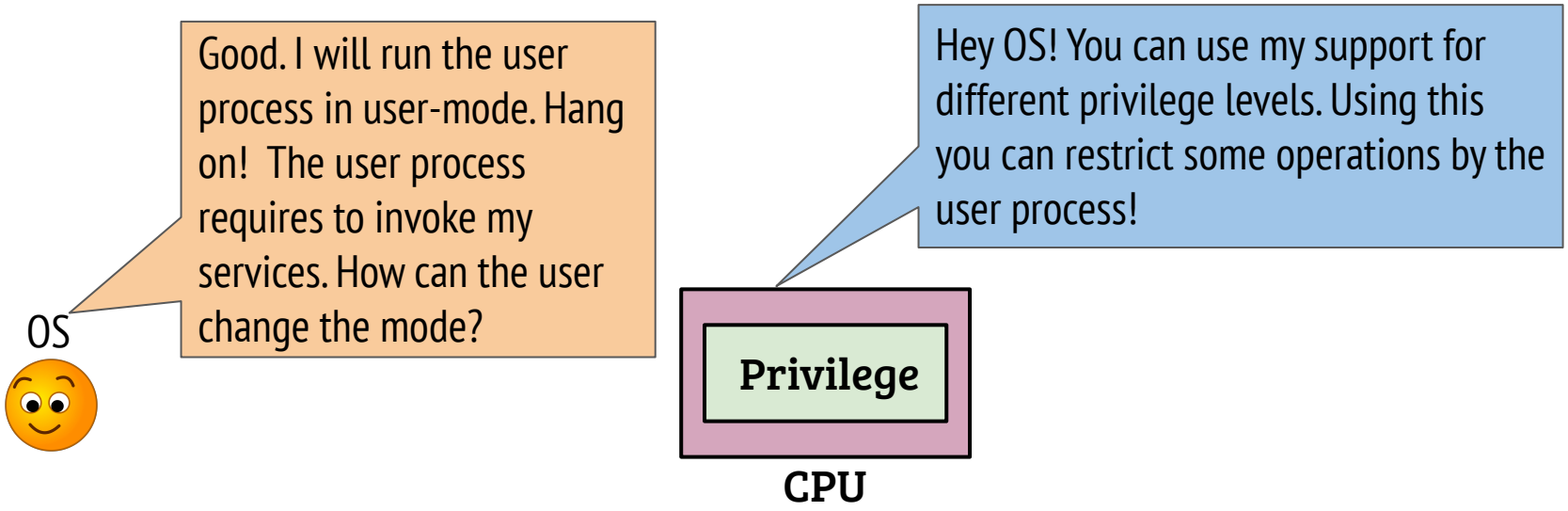


Hardware support: Privilege levels

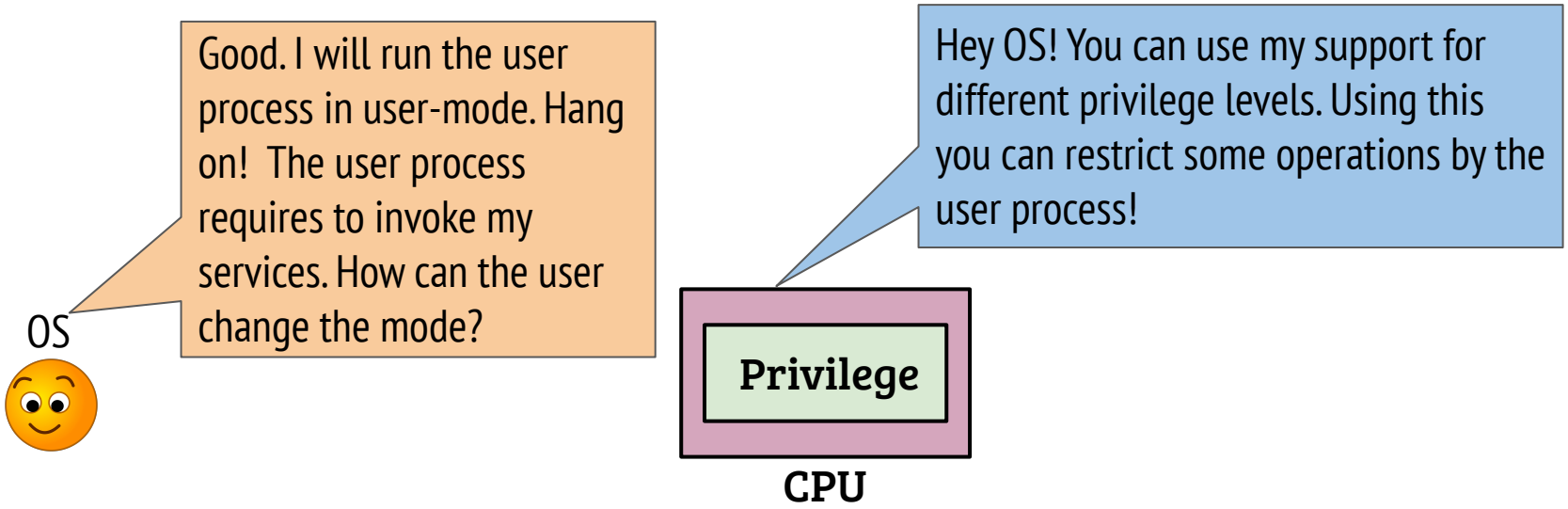


- CPU can execute in two modes: *user-mode* and *kernel-mode*
- Some operations are allowed only from kernel-mode (privileged OPs)
  - If executed from user mode, hardware will notify the OS by raising a fault/trap

Hardware support: Privilege levels

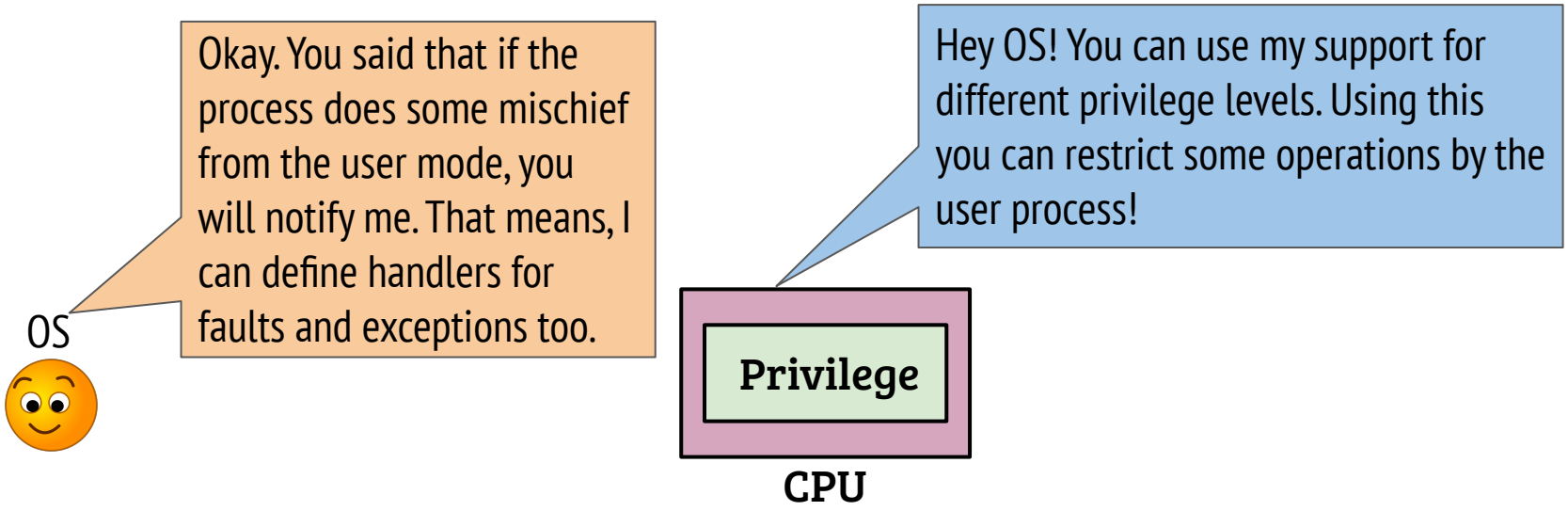


Hardware support: Privilege levels

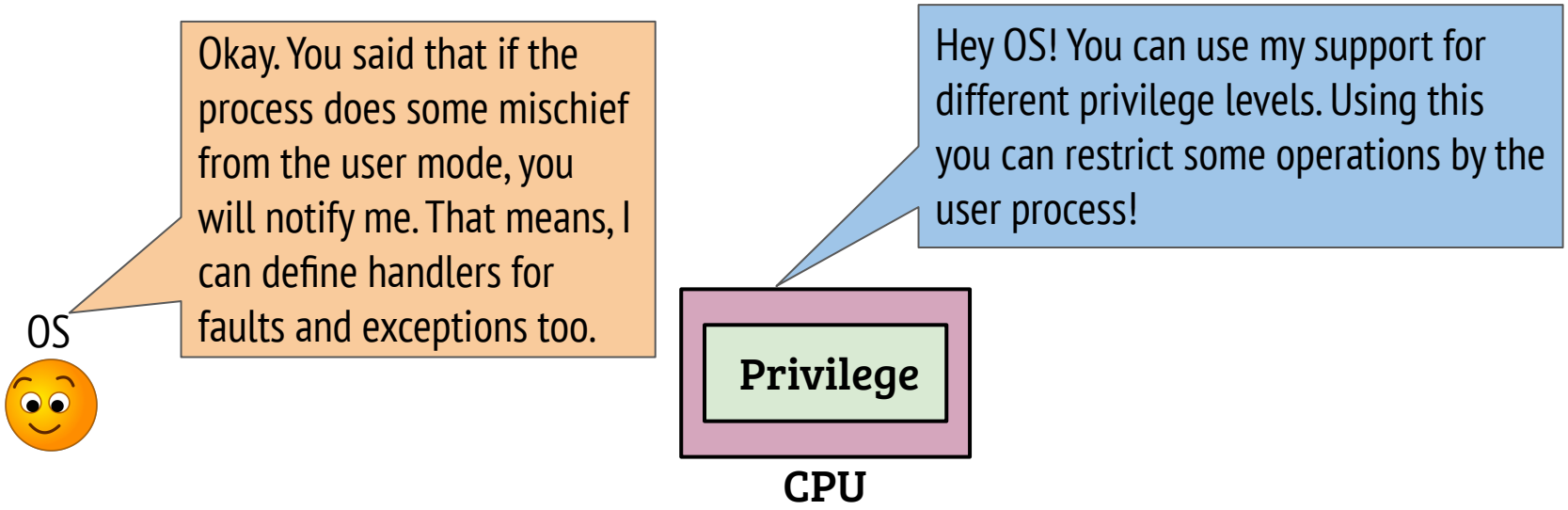


- From user-mode, privilege level of CPU can not be changed directly
- The hardware provides entry instructions from the user-mode which causes a mode switch
- The OS can define the handler for different entry gates

Hardware support: Privilege levels

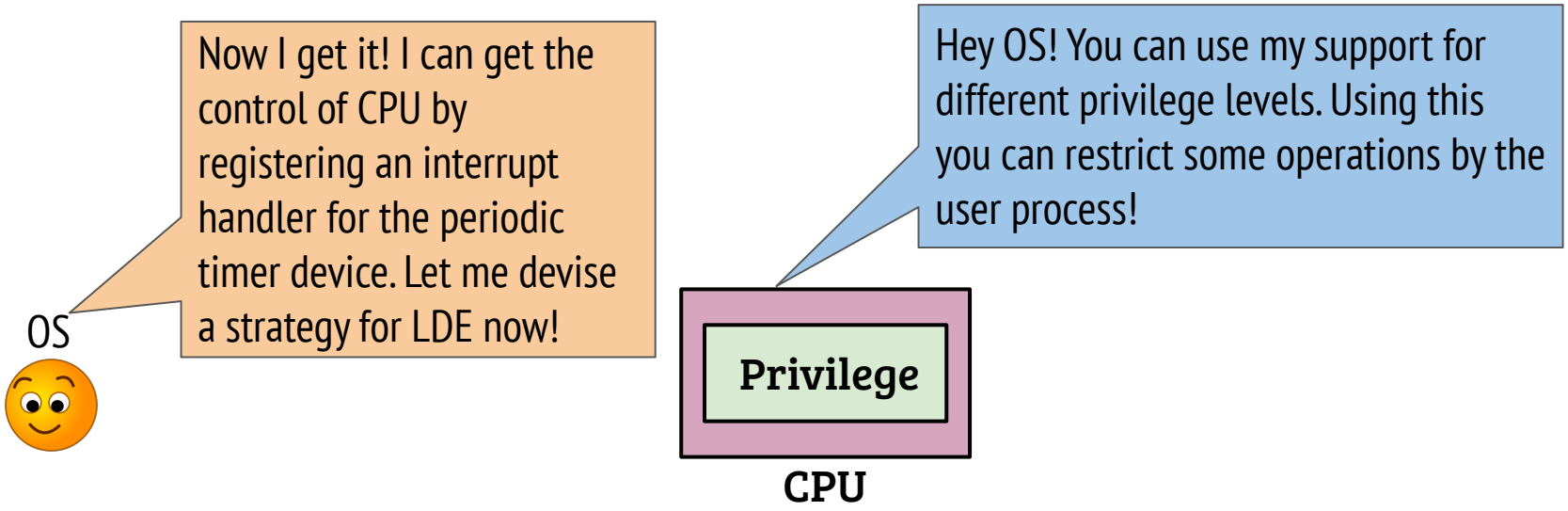


Hardware support: Privilege levels

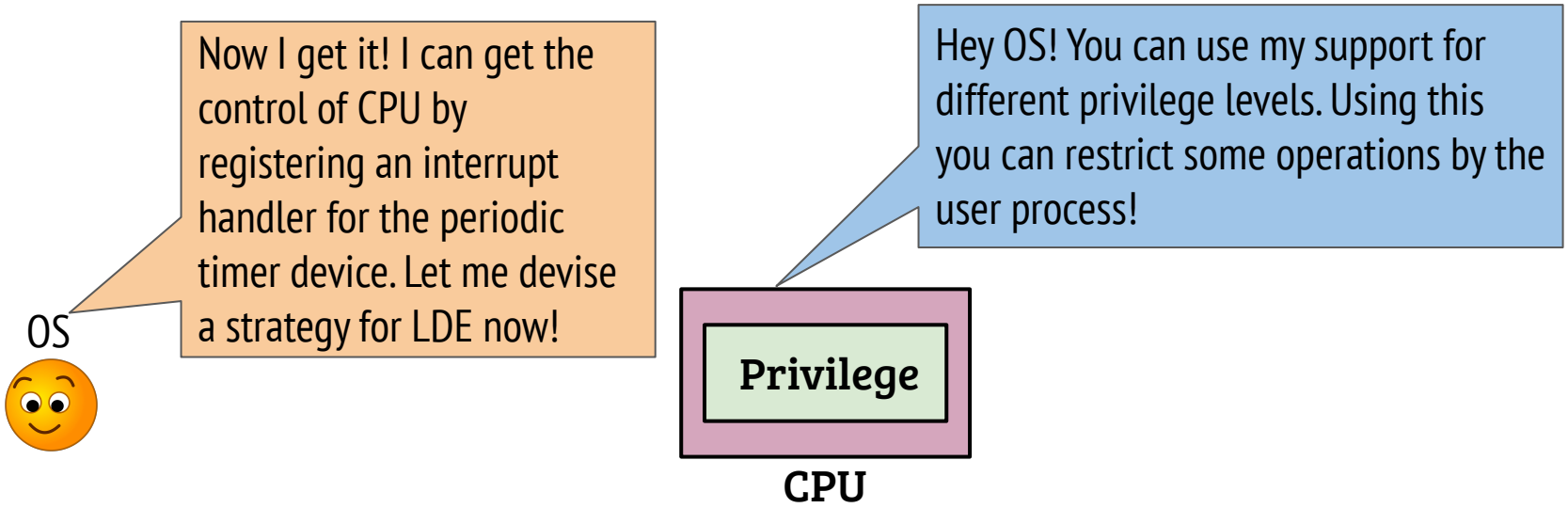


- The OS can register the handlers for faults and exceptions
- The OS can also register handlers for device interrupts
- *Registration of handlers is privileged!*

Hardware support: Privilege levels

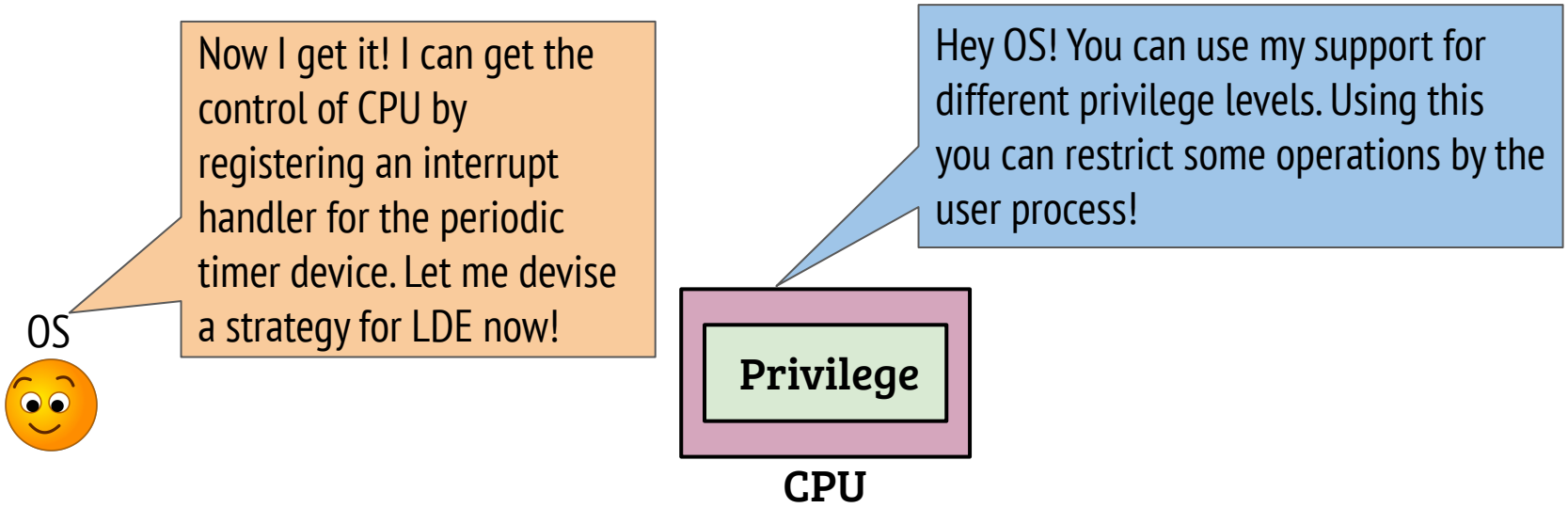


Hardware support: Privilege levels



- After the boot, the OS needs to configure the handlers for system calls, exceptions/faults and interrupts

Hardware support: Privilege levels



- After the boot, the OS needs to configure the handlers for system calls, exceptions/faults and interrupts
- The handler code is invoked by the OS when user-mode process invokes a system call or an exception or an external interrupt

Limited direct execution

- Can the OS enforce limits to an executing process?
- No, the OS can not enforce limits by itself and still achieve efficiency
- OS requires support from hardware!
- What kind of support is needed from the hardware?
- CPU privilege levels: user-mode vs. kernel-mode
- Switching between modes, entry points and handlers