EE488: System Software Design
Fall 2022

Operating System Organization

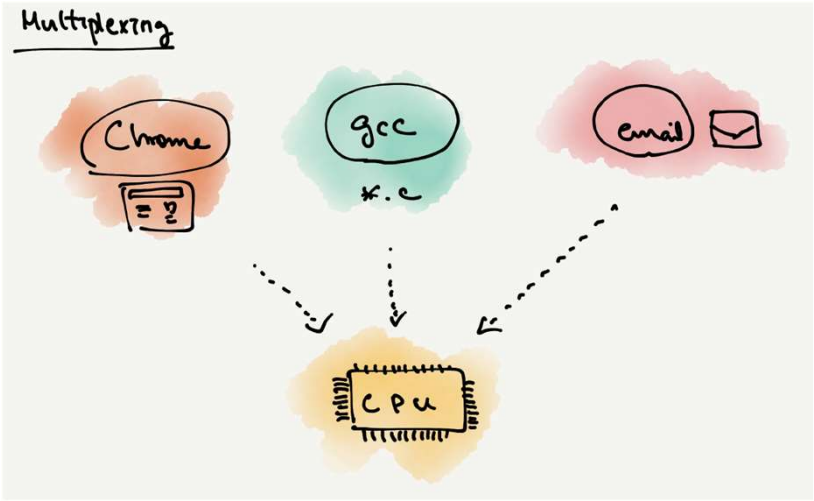
Youjip Won

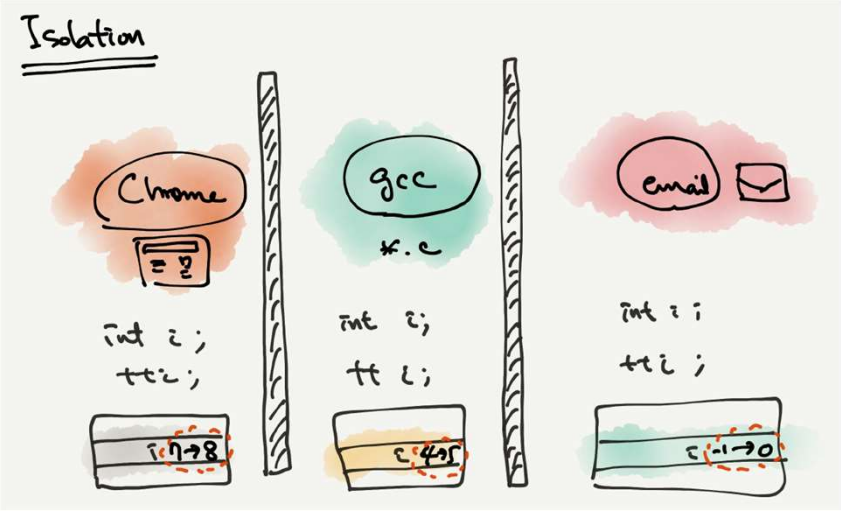
KAIST EE

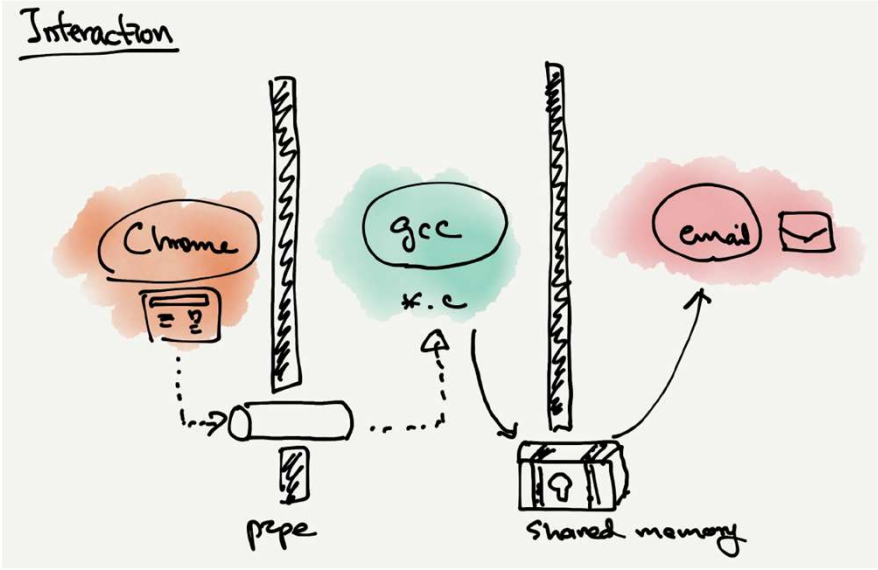
Operating system organization

- The three requirements of operating system
  - Multiplexing
    - OS transparently switches the hardware resources from one process to another. e.g. context switch or virtual memory
  - Isolation
    - One process should not affect the execution of the other process.
    - OS forbids the application to directly access the hardware.
  - Interaction
    - Provides a mechanism for the processes to communication
    - e.g. via pipe or via signal.

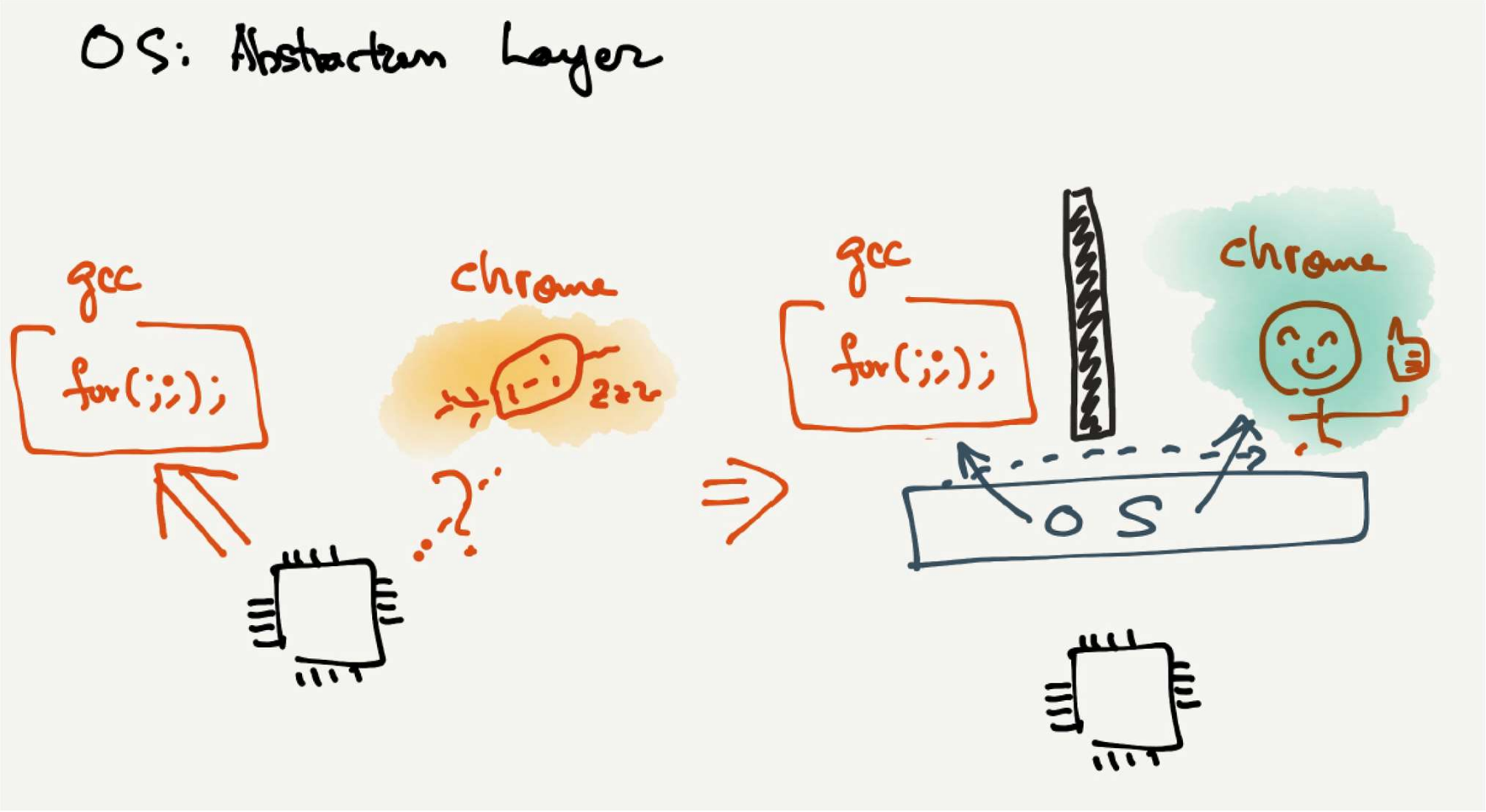
Multiplexing, isolation, interaction



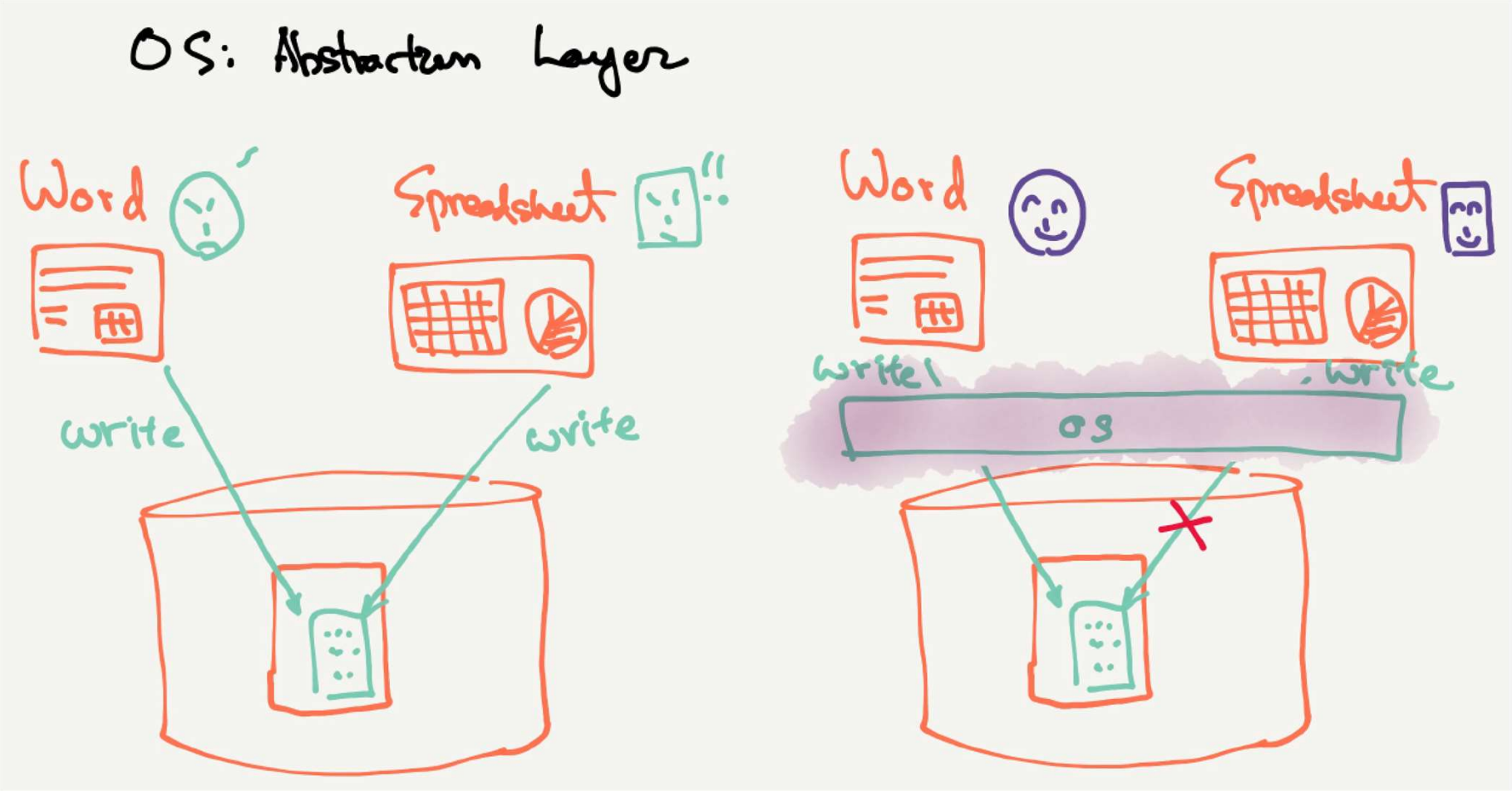




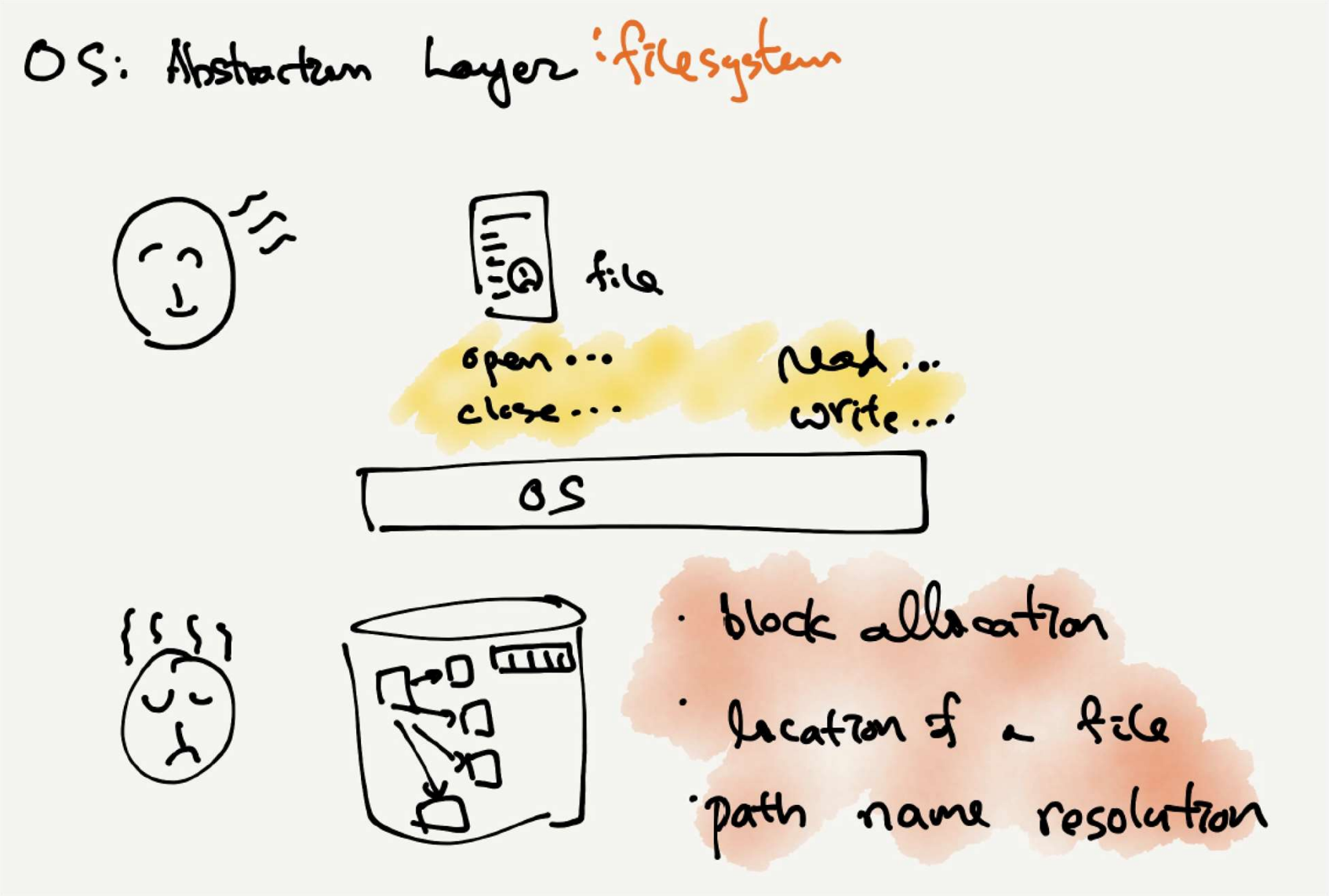
Abstraction layer



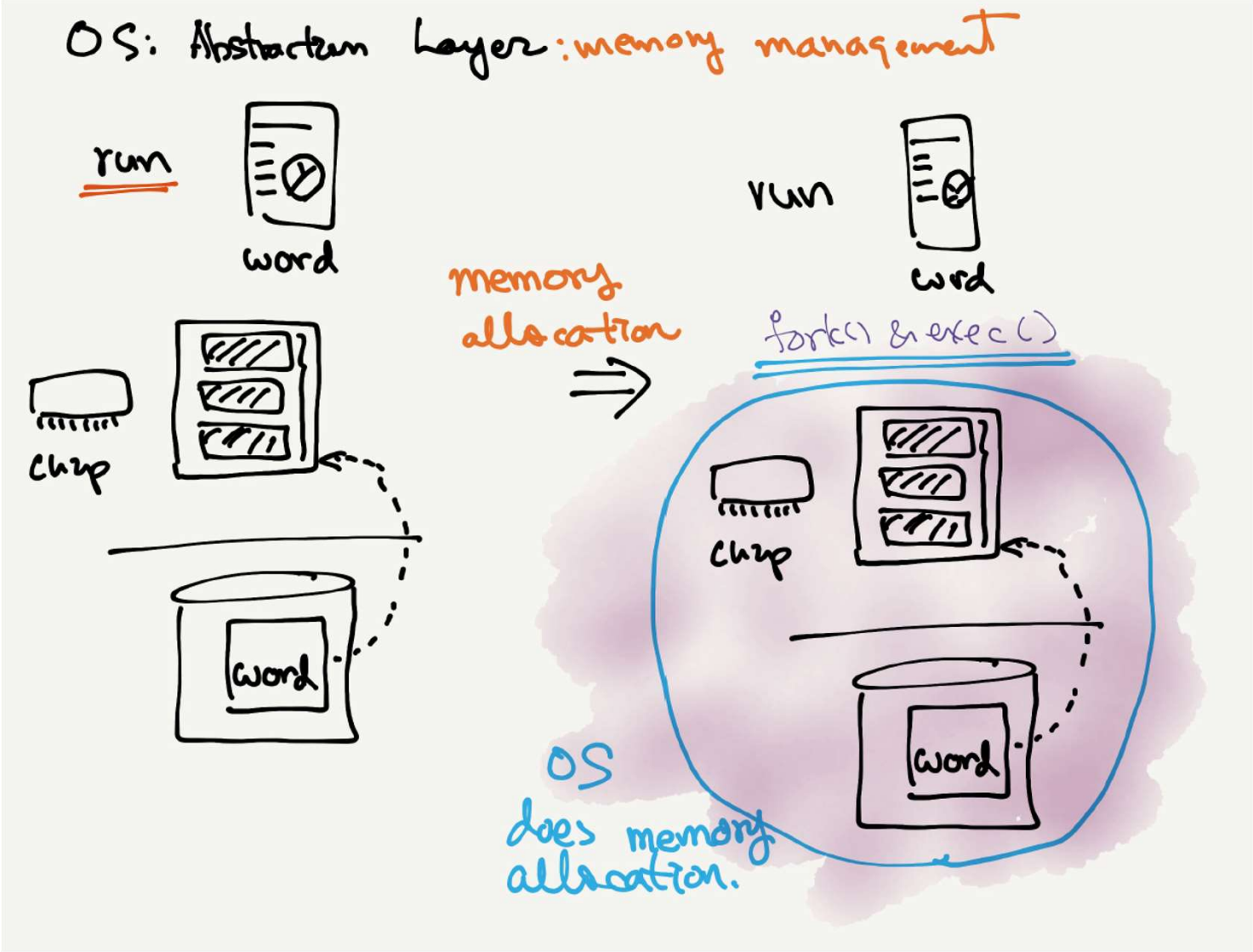
Abstraction layer (Cont.)



Abstraction layer : file system



Abstraction layer : memory management



User mode / Kernel mode

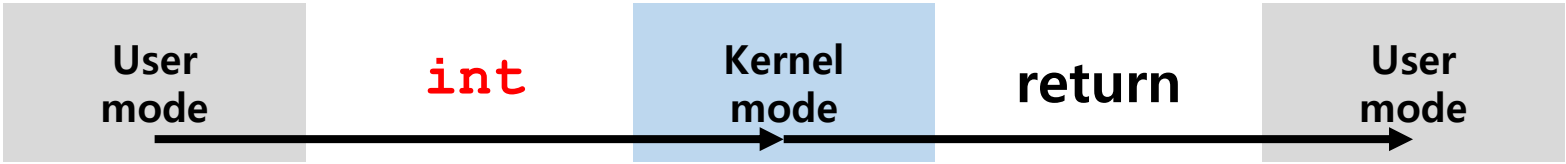
- Strong isolation requires a hard boundary between applications and operating systems. If the application makes a mistake, we do not want the operating system to fail.
- Processors provide hardware support for strong isolation.
  - x86 processor has two execution modes: kernel mode & user mode.
  - An application can execute only user-mode instructions (e.g., adding numbers, etc.) and is said to be running in user space.
  - The software in kernel mode can execute privileged instructions and is said to be running in kernel space. (e.g. reading and writing the disk)
- The software running in kernel space (or in kernel mode) is called the kernel.
- CPU provides a special instruction to switch the mode from the user mode to the kernel mode and enters the kernel at an entry point specified by the kernel.

`int` in x86

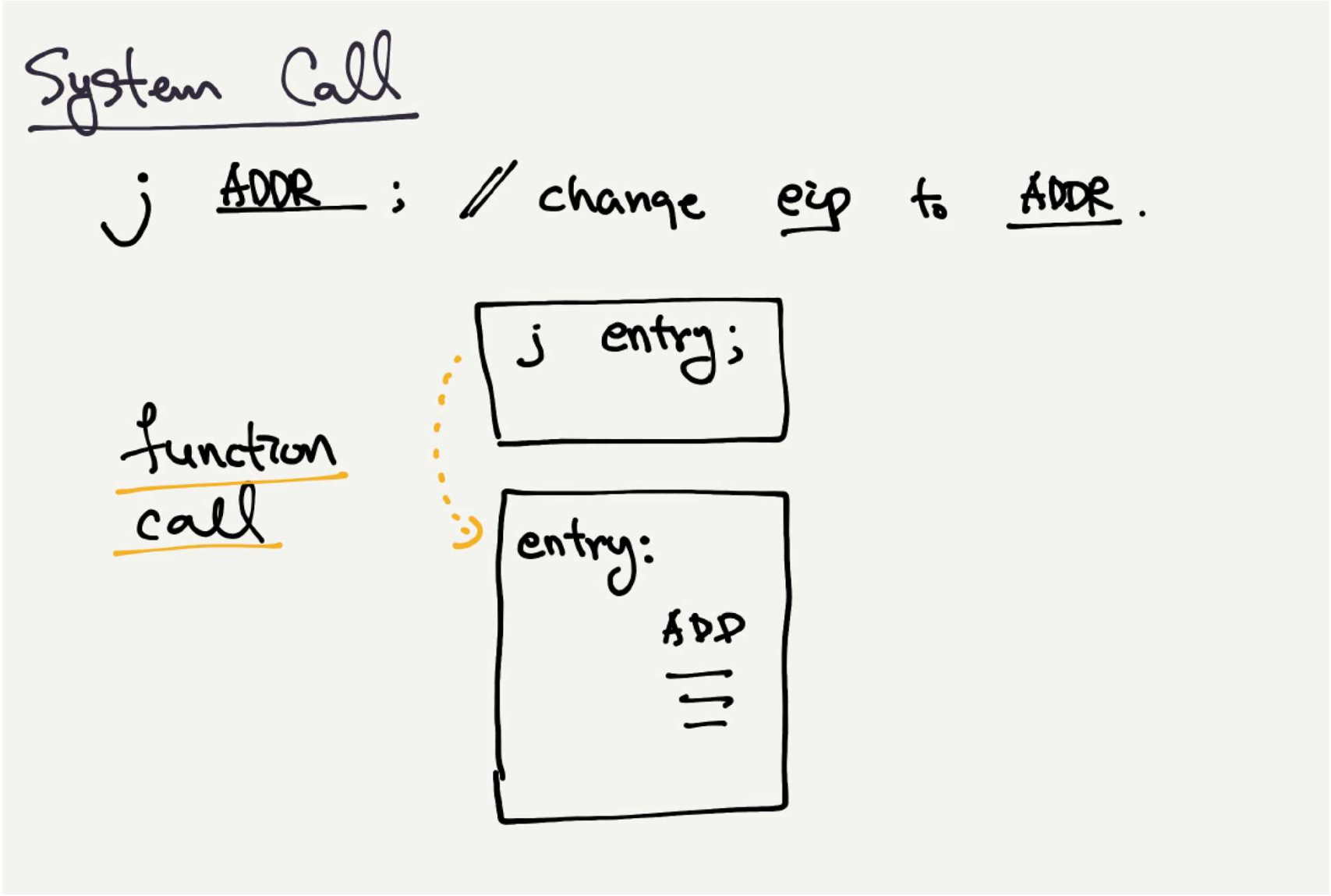
`int` instruction

- Switch the execution mode from the user mode to the kernel mode.
  - e.g. An application that wants to read or write a file on disk must transition to the kernel.

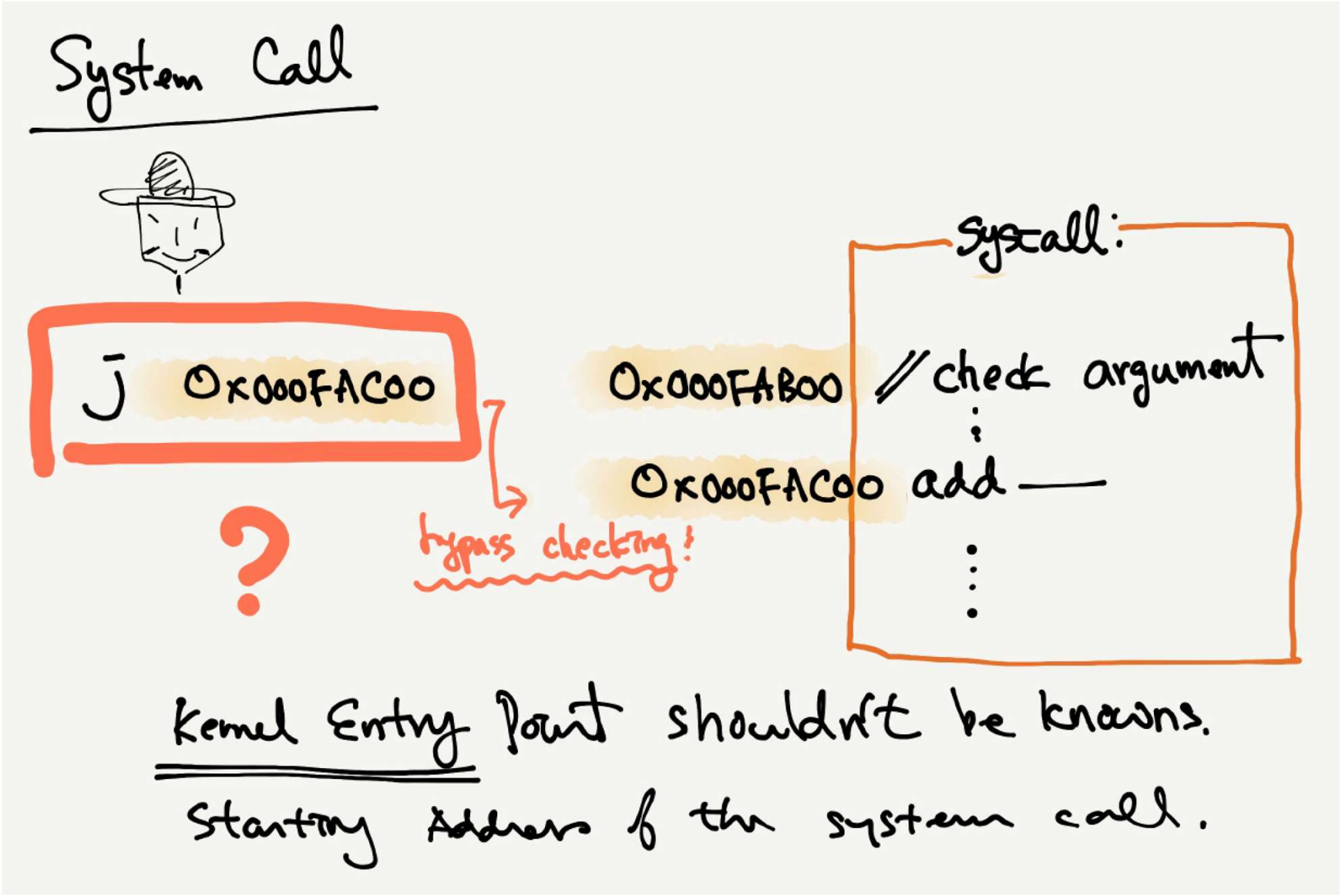
- Once the processor has switched to kernel mode, the kernel can then validate the arguments of the system call, decide whether the application is allowed to perform the requested operation, and then deny it or execute it.



System call


System Call
j ADDR ; // change eip to ADDR.
function call
j entry;
entry:
ADD

System call


System Call
J 0x000FAC00
?
bypass checking!
Syscall:
0x000FAB00 // check argument
0x000FAC00 add
Kernel Entry Point shouldn't be known.
Starting Address of the system call.

Kernel Organization

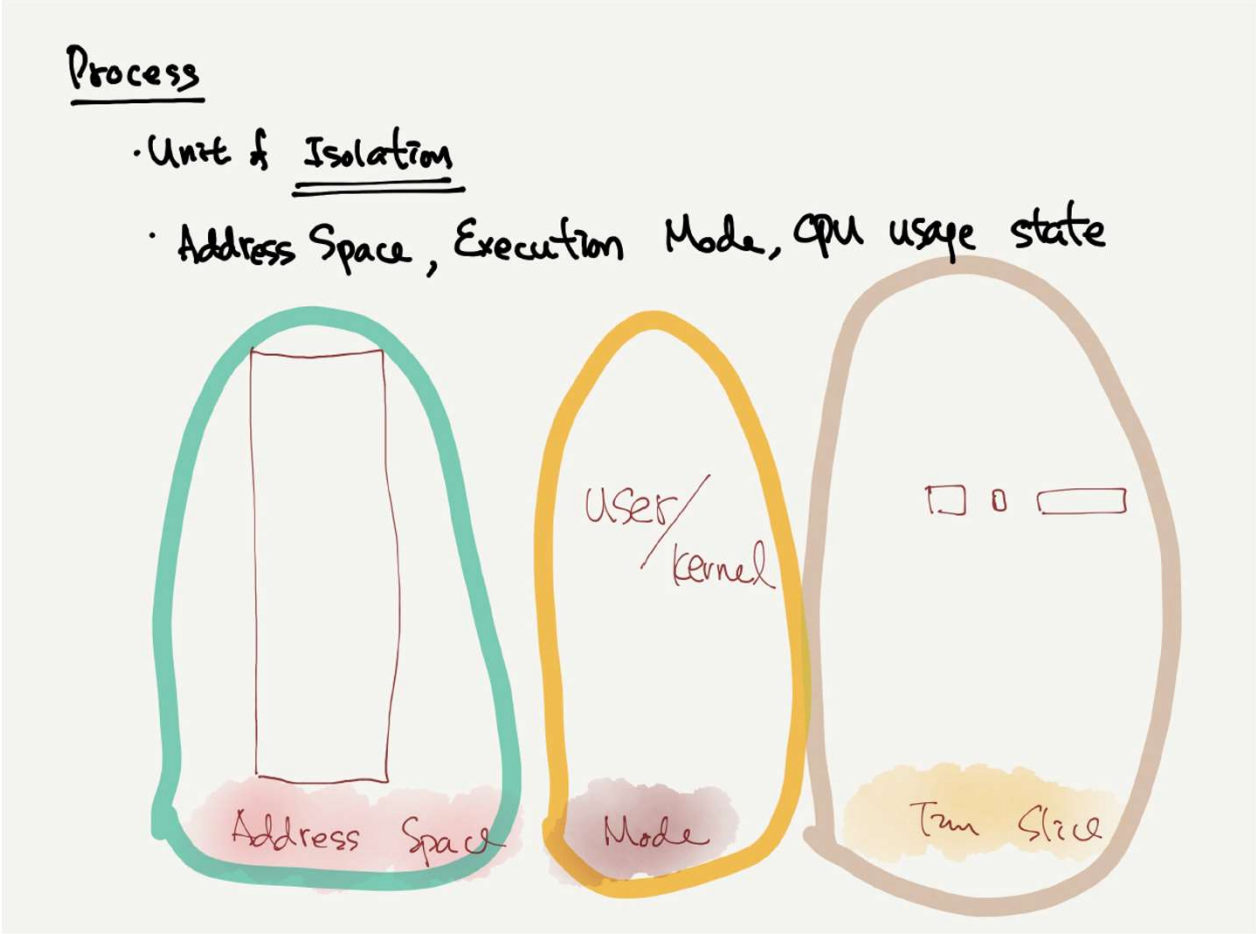
What part of the operating system should run in kernel mode?

- **Monolithic kernel**
  - Entire OS resides in the kernel.
  - This organization is convenient because the OS designer doesn't have to decide which part of the operating system doesn't need full hardware privilege.
  - Different parts of OS can easily cooperate., e.g. virtual memory and filesystem
  - Error prone
- **Micro kernel** :
  - Executing the bulk of the operating system in user mode.
  - The kernel interface consists of a few low-level functions for starting applications, sending messages, accessing device hardware, etc.
  - The kernel is relatively simple, as most of the operating system resides in user-level servers.
- **xv6 is implemented as a monolithic kernel.**

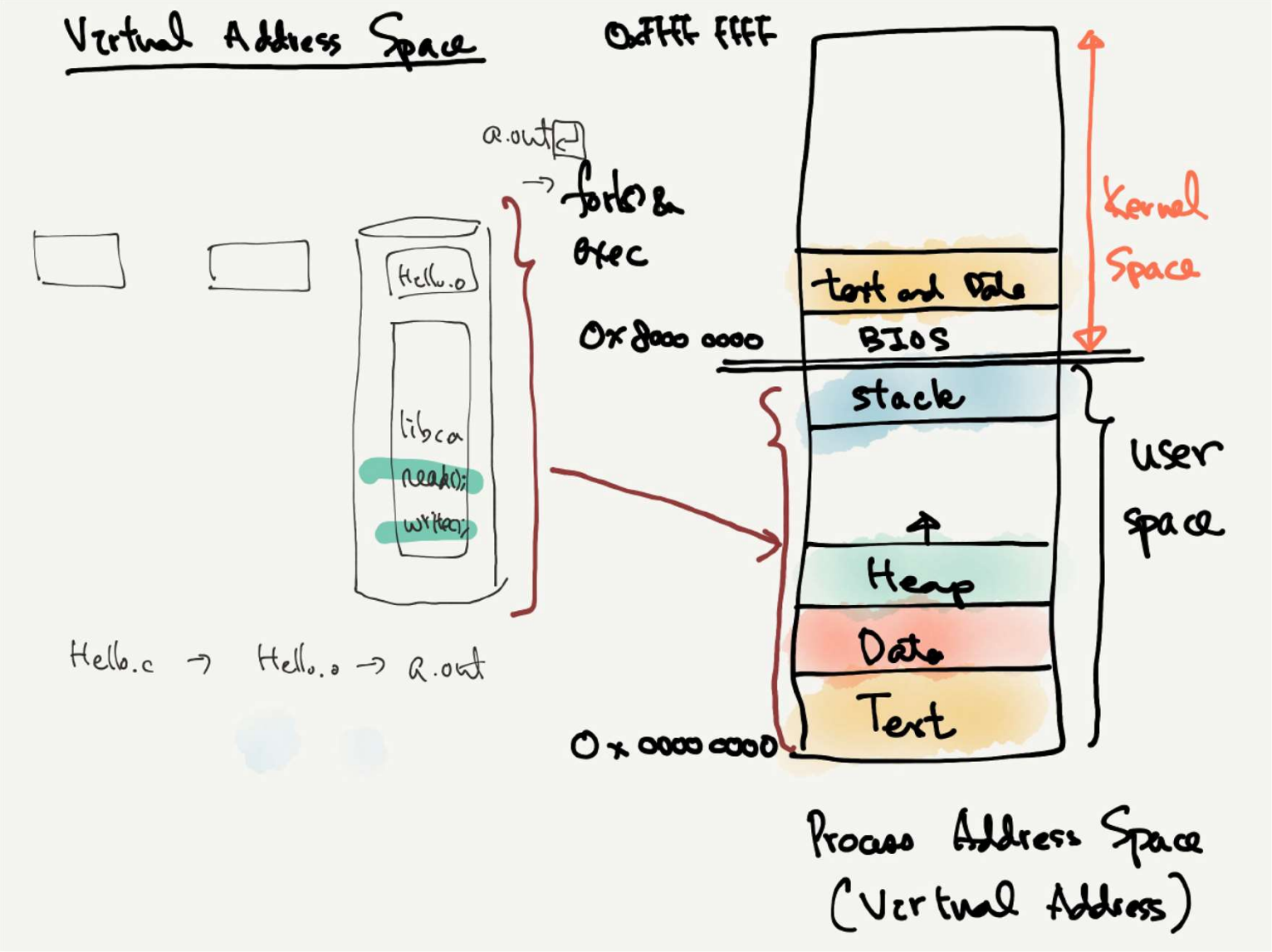
Process

Process

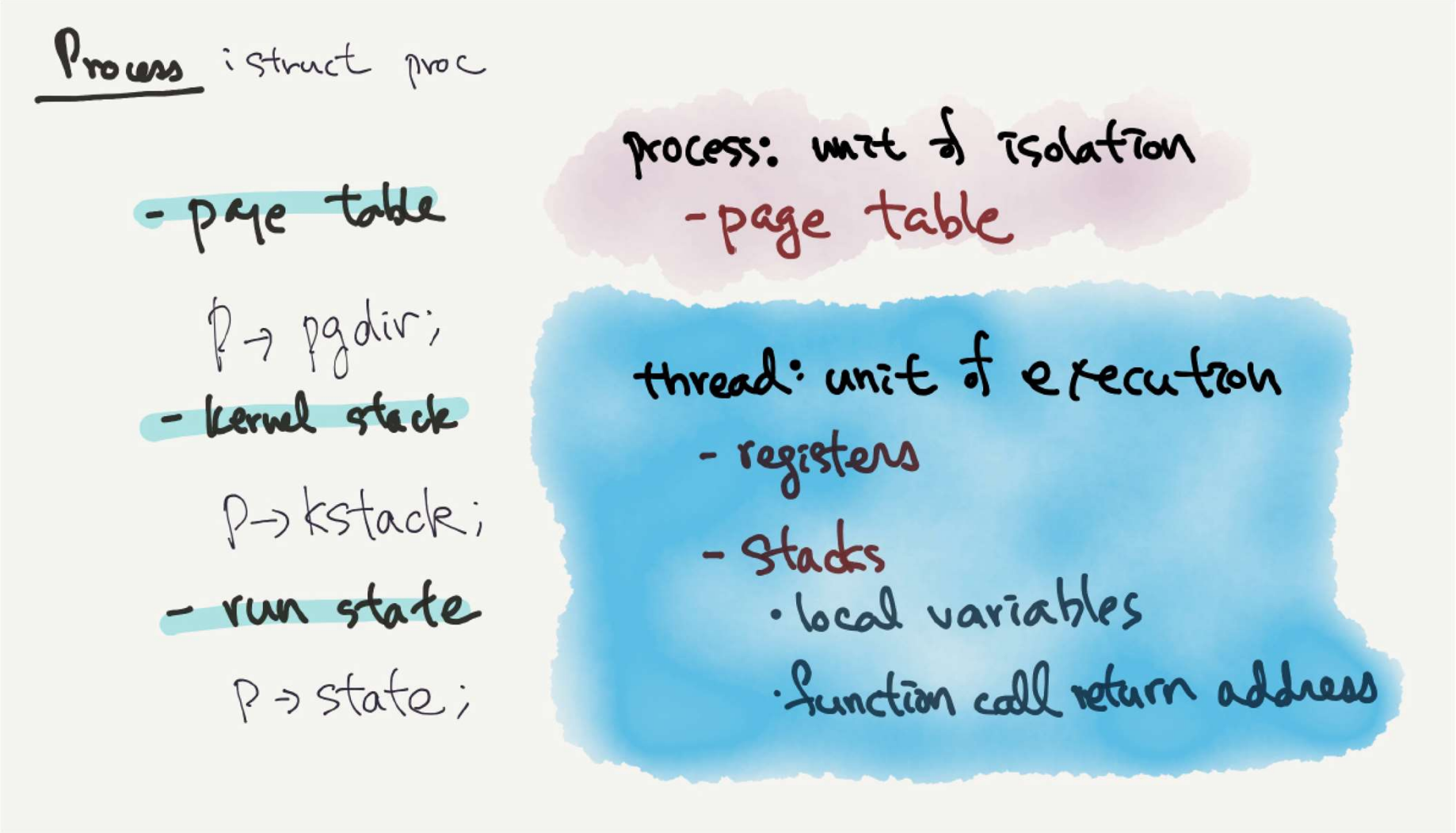
- Unit of isolation
- Has its own state: address space, execution mode, CPU time quantum



Virtual address space



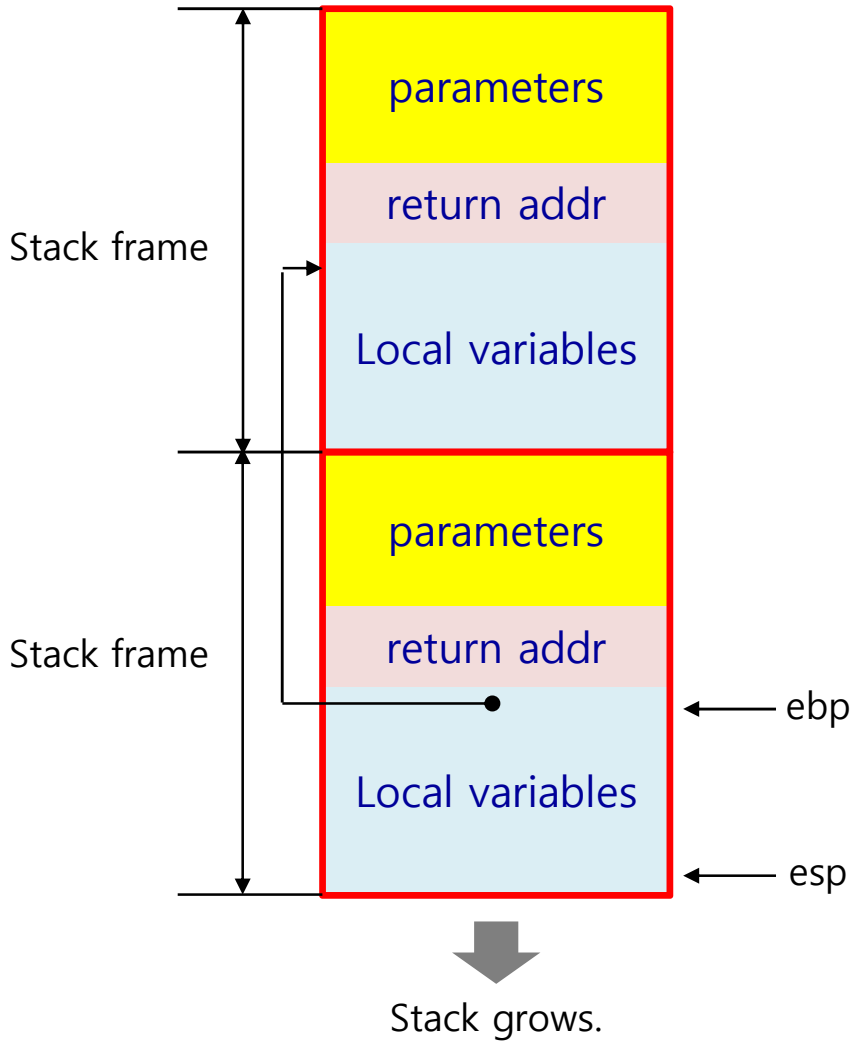
Process : `struct proc`



Process : `struct proc` (Cont.)

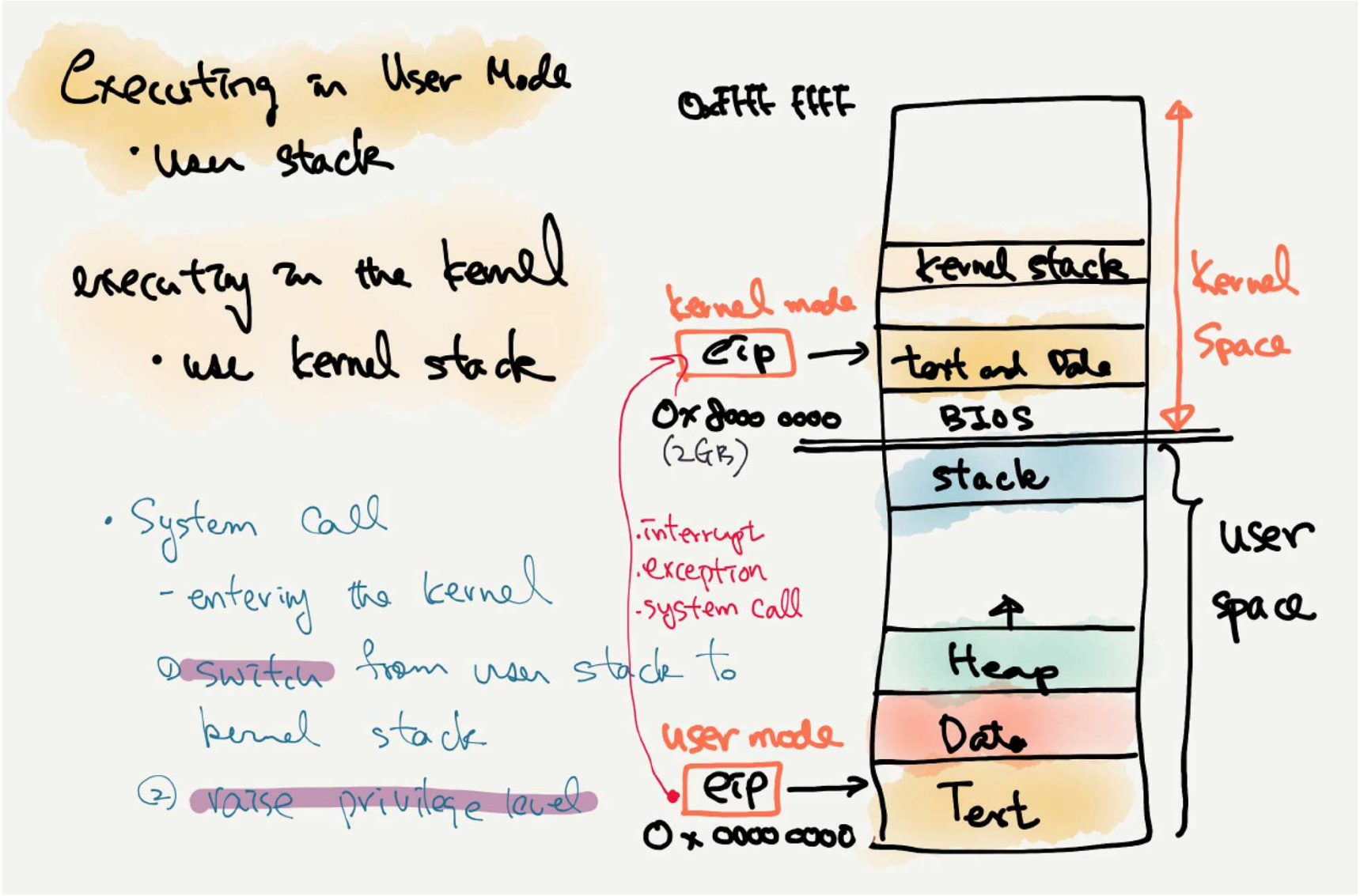
```
// Per-process state
struct proc {
  uint sz;                     // Size of process memory (bytes)
  pde_t* pgdir;                // Page table
  char *kstack;                // Bottom of kernel stack for this process
  enum procstate state;        // Process state
  int pid;                     // Process ID
  struct proc *parent;         // Parent process
  struct trapframe *tf;        // Trap frame for current syscall
  struct context *context;     // swtch() here to run process
  void *chan;                  // If non-zero, sleeping on chan
  int killed;                  // If non-zero, have been killed
  struct file *ofile[NOFILE];  // Open files
  struct inode *cwd;           // Current directory
  char name[16];               // Process name (debugging)
};
```

stack frame for function call



- to save the followings
  - Parameters, Return address, local variables
- ebp (base pointer register)
  - the address of the beginning of the stack frame, remains unchanged while the function executes.
- esp (stack pointer)
  - address of stack top, keeps changing while the function executes.

User stack vs. kernel stack



Summary

- overview of booting

- creating the first address space
- creating the first process
- running the first process

- do not forget: preview and review