IOWA STATE UNIVERSITY
Department of Electrical and Computer Engineering

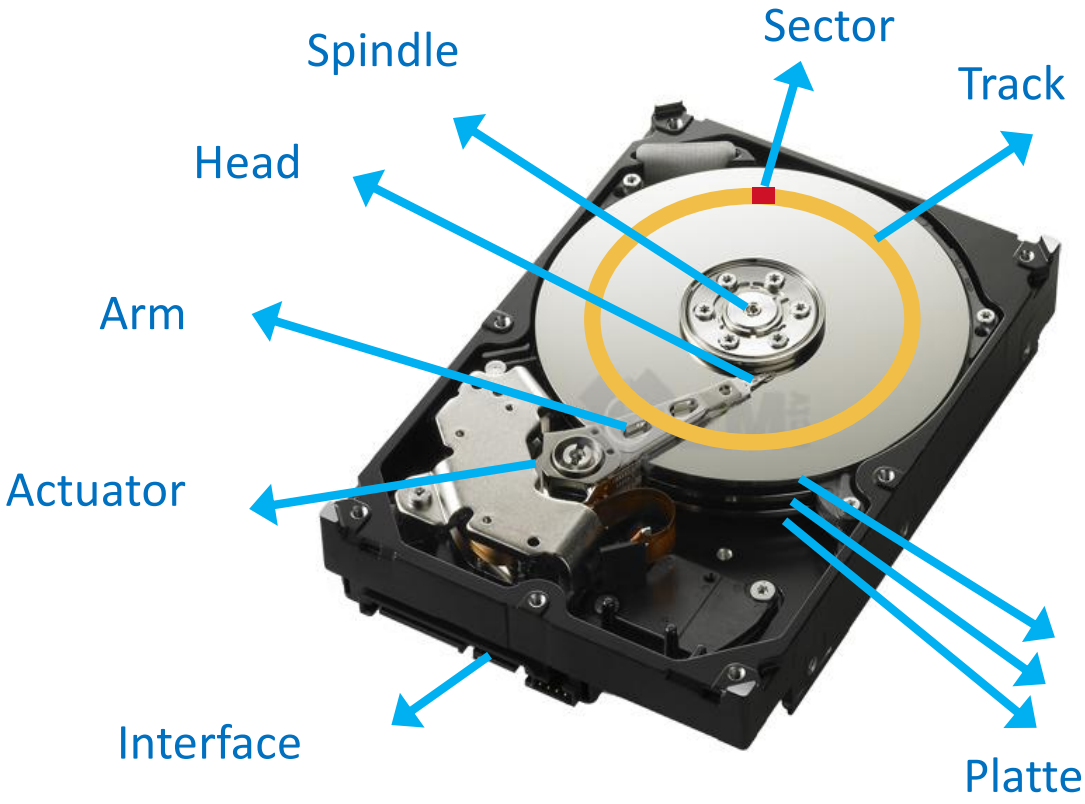
Lecture 04:
OS Introduction III

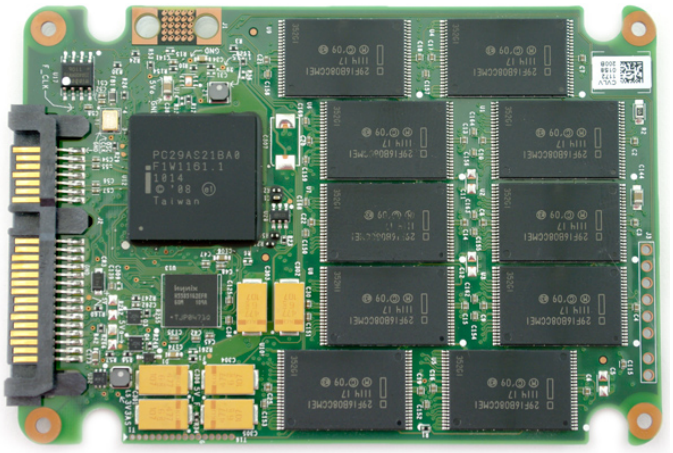
Agenda

- Recap
- OS Introduction III
  - OS Interface: System Calls
  - OS Internal Structures

Recap

- I/O device examples
  - hard disk drives (HDDs) & solid-state drives (SSDs)



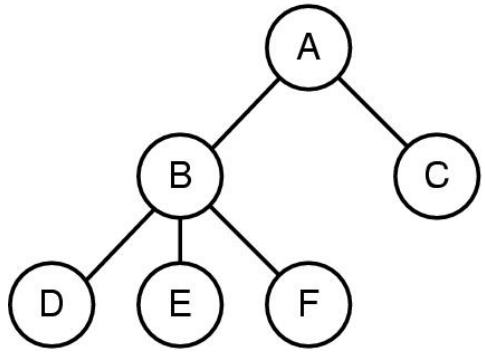


Recap

- Common OS Abstractions for HW
  - CPU
    - process and/or thread
  - Memory
    - address space
  - Disks
    - files

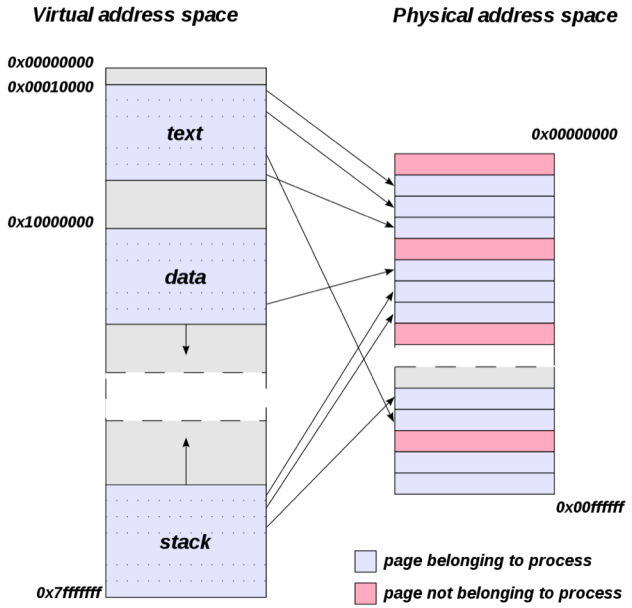
Recap

- Process
  - A program *in execution*
  - holds all the info needed to run a program
    - Address space
      - Program (text), data, stack
    - Register values
      - Program counter, stack pointer, etc
- Process Tree
  - a hierarchy of related processes
- Process management
  - create, schedule, suspend, resume, kill, communication, synchronization, ...



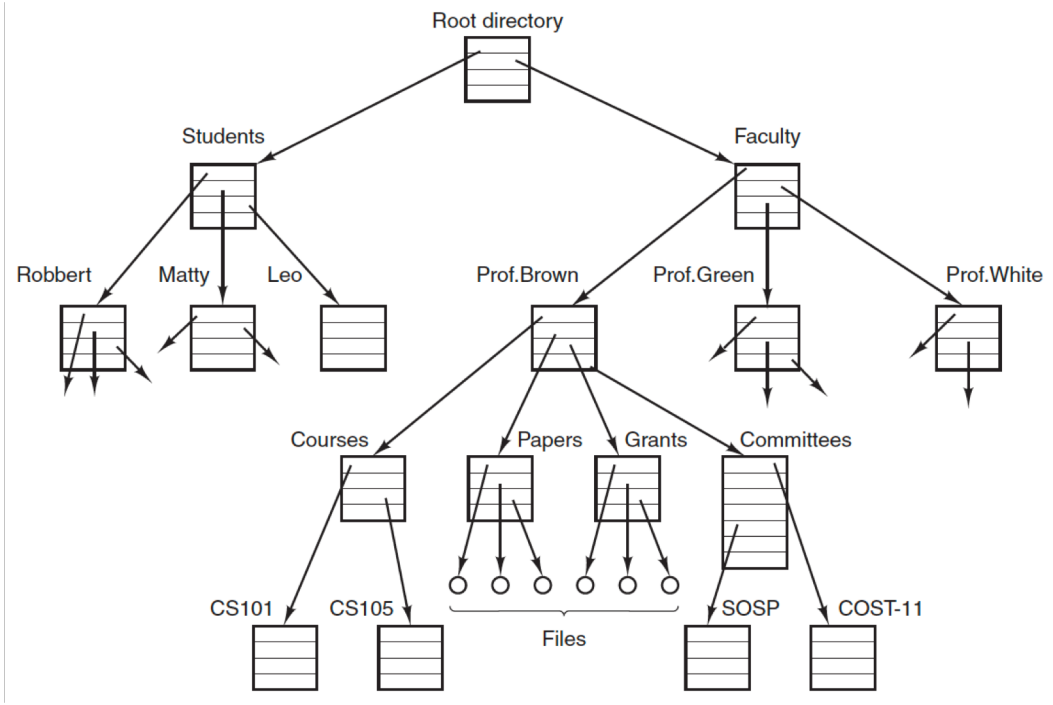
Recap

- Address space
  - A range of memory addresses accessed by the process
    - each process has its own **virtual address space**.
    - OS maps virtual address space onto the physical memory



Recap

- Files
  - “Everything is a file” on Linux
    - resources are represented as files in a file system tree
    - retrievable via a path from the root

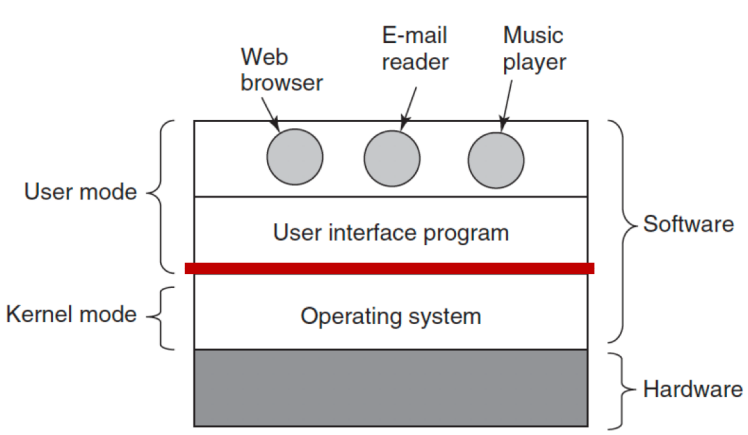


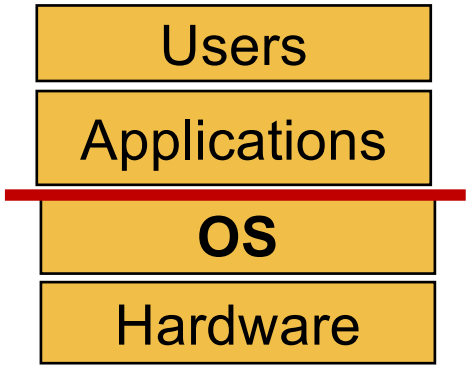
Agenda

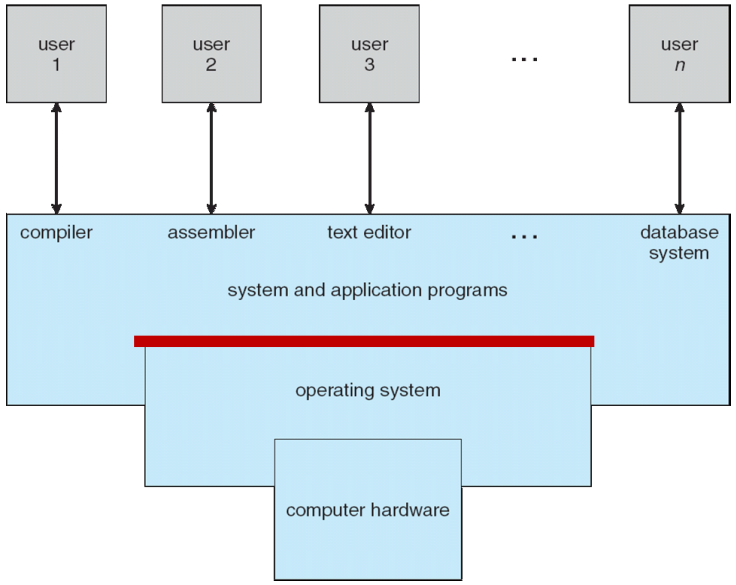
- ~~Recap~~
- OS Introduction III
  - OS Interface: System Calls
  - OS Internal Structures

System Calls

- Function calls implemented by OS
  - interface to apps/users
  - may be wrapped in library calls (e.g., glibc)
  - many follow the POSIX standard



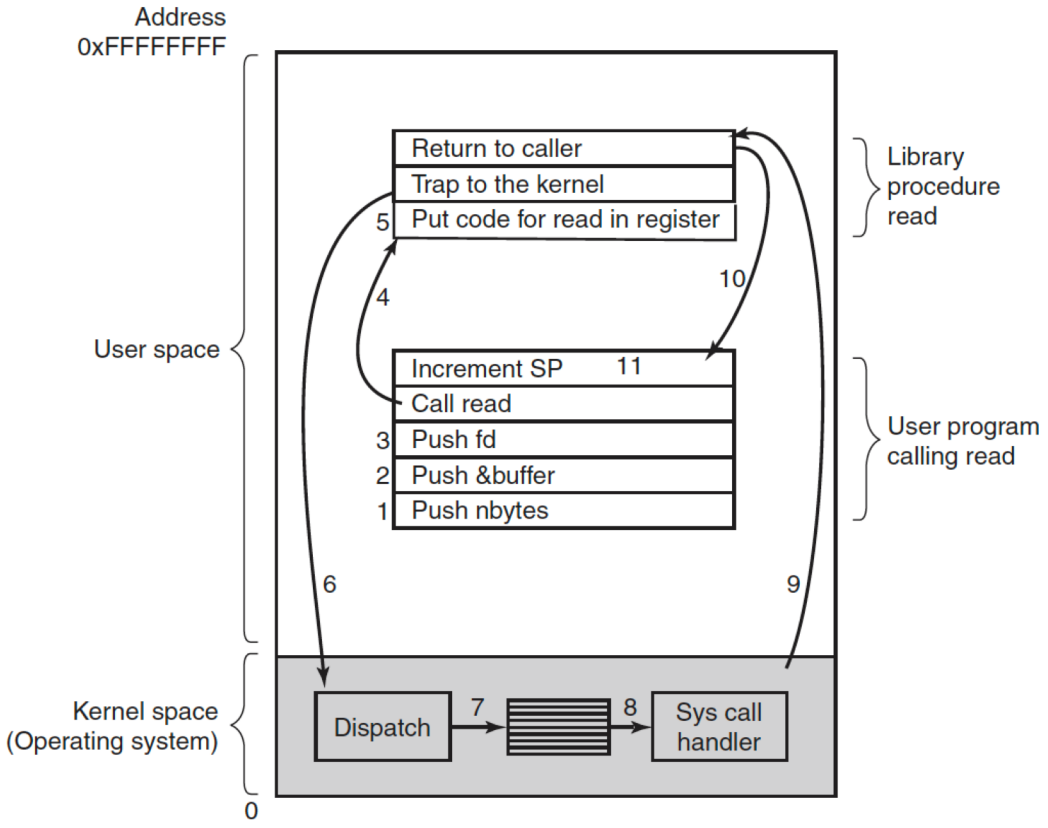




System Calls

- Examples
  - a system call directly invoked by a program

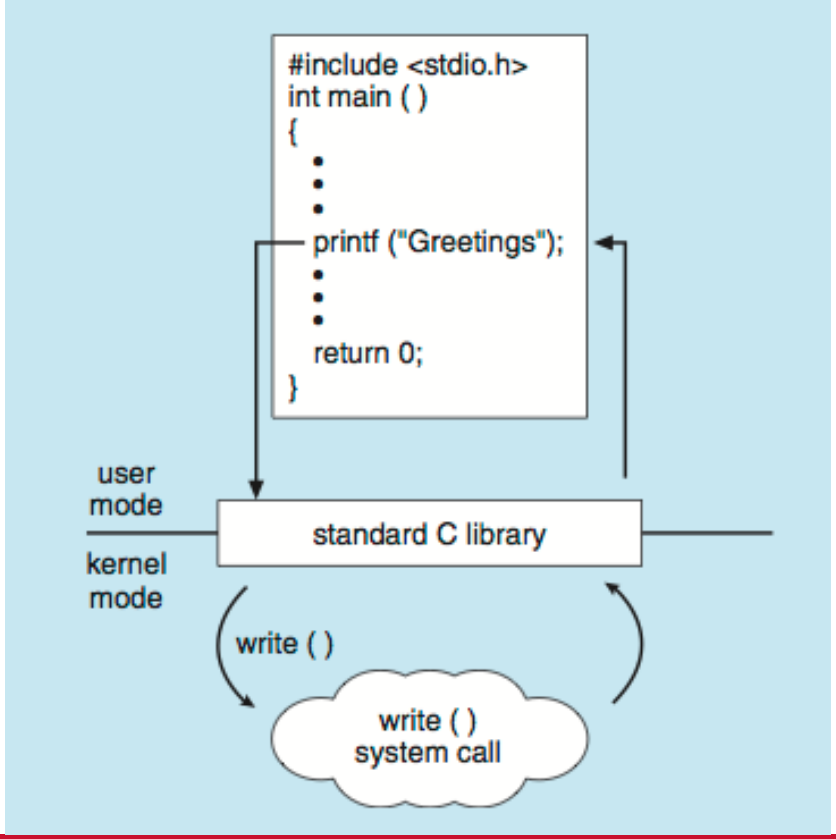
count = read(fd, buffer, nbytes);



System Calls

- Examples
  - a system call wrapped in a library function

    *ssize_t write(int fd, const void *buf, size_t count); /*Linux*/*



System Calls

- Examples
  - process management

Call	Description
pid = fork( )	Create a child process identical to the parent
pid = waitpid(pid, &statloc, options)	Wait for a child to terminate
s = execve(name, argv, environp)	Replace a process' core image
exit(status)	Terminate process execution and return status

System Calls

- Examples
  - process management
    - a simplified shell

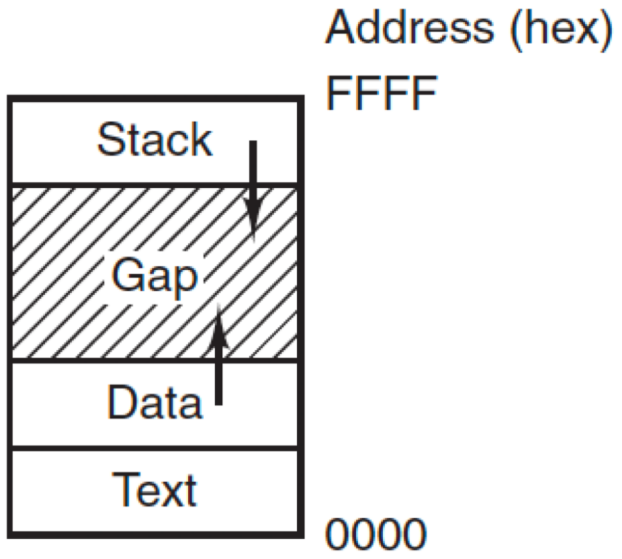
```
#define TRUE 1

while (TRUE) {                                  /* repeat forever */
        type_prompt( );                         /* display prompt on the screen */
        read_command(command, parameters);      /* read input from terminal */

        if (fork( ) != 0) {                     /* fork off child process */
             /* Parent code. */
             waitpid(–1, &status, 0);           /* wait for child to exit */
        } else {
             /* Child code. */
             execve(command, parameters, 0);    /* execute command */
        }
}
```

System Calls

- Examples
  - UNIX processes have their memory divided into 3 segments
    - text: the program code
    - data: variables
      - extend explicitly via brk syscall
    - stack: for function calls



System Calls

- Examples
  - file/directory/file-system management

Call	Description
fd = open(file, how, ...)	Open a file for reading, writing, or both
s = close(fd)	Close an open file
n = read(fd, buffer, nbytes)	Read data from a file into a buffer
n = write(fd, buffer, nbytes)	Write data from a buffer into a file
position = lseek(fd, offset, whence)	Move the file pointer
s = stat(name, &buf)	Get a file's status information

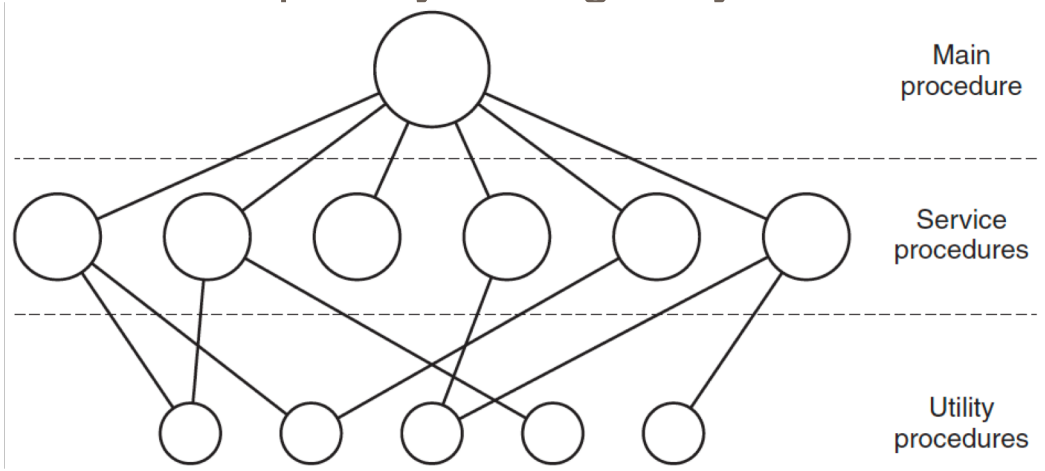
Call	Description
s = mkdir(name, mode)	Create a new directory
s = rmdir(name)	Remove an empty directory
s = link(name1, name2)	Create a new entry, name2, pointing to name1
s = unlink(name)	Remove a directory entry
s = mount(special, name, flag)	Mount a file system
s = umount(special)	Unmount a file system

Agenda

- ~~Recap~~
- OS Introduction III
  - ~~OS Interface: System Calls~~
  - OS Internal Structures

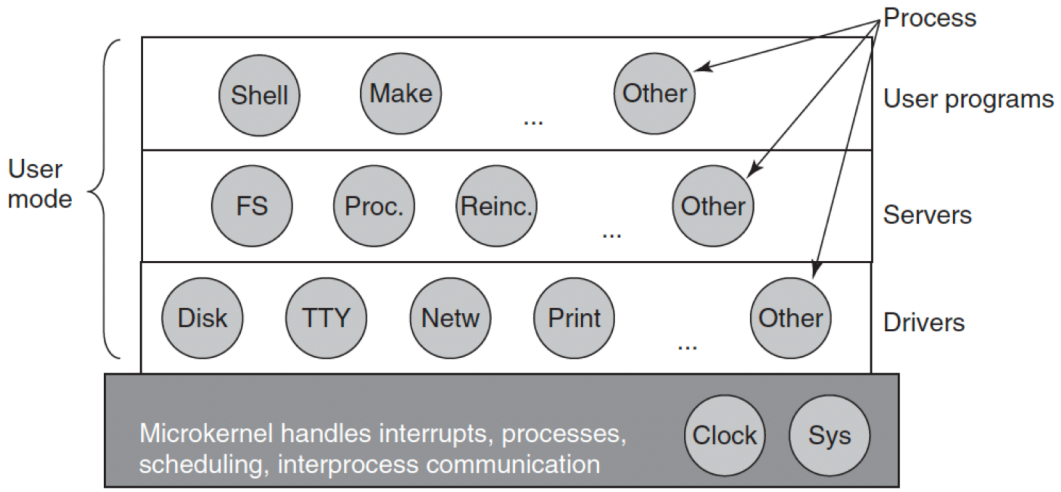
OS Structures

- Monolithic kernel
  - E.g., Linux
  - the entire OS runs as a single program in kernel mode
    - a collection of procedures, linked together into a single large executable
    - may include loadable modules (e.g., drivers, file systems)
    - may include multiple layers logically



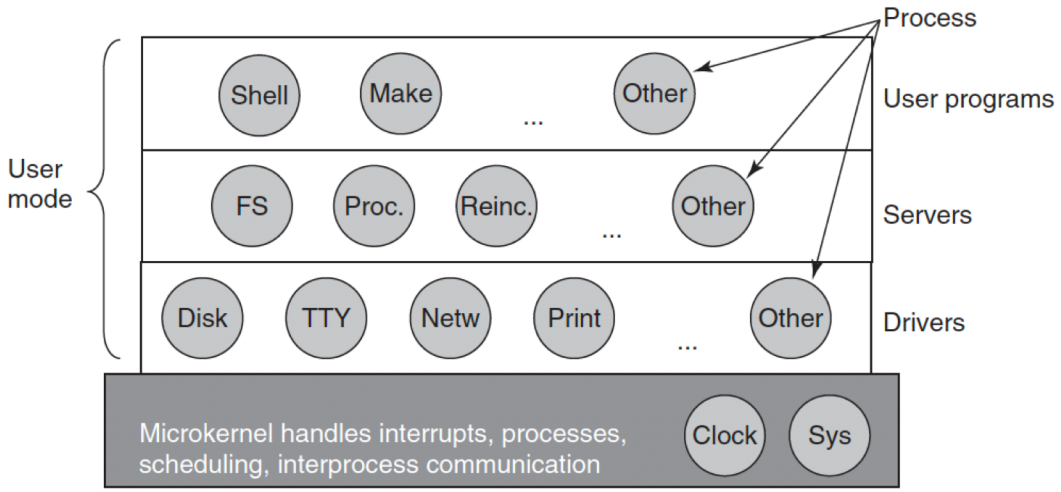
OS Structures

- Microkernel
  - E.g., MINIX 3
  - split the OS up into small, well-defined modules, only one of which—the microkernel—runs in kernel mode
    - the rest run as relatively powerless ordinary user processes



OS Structures

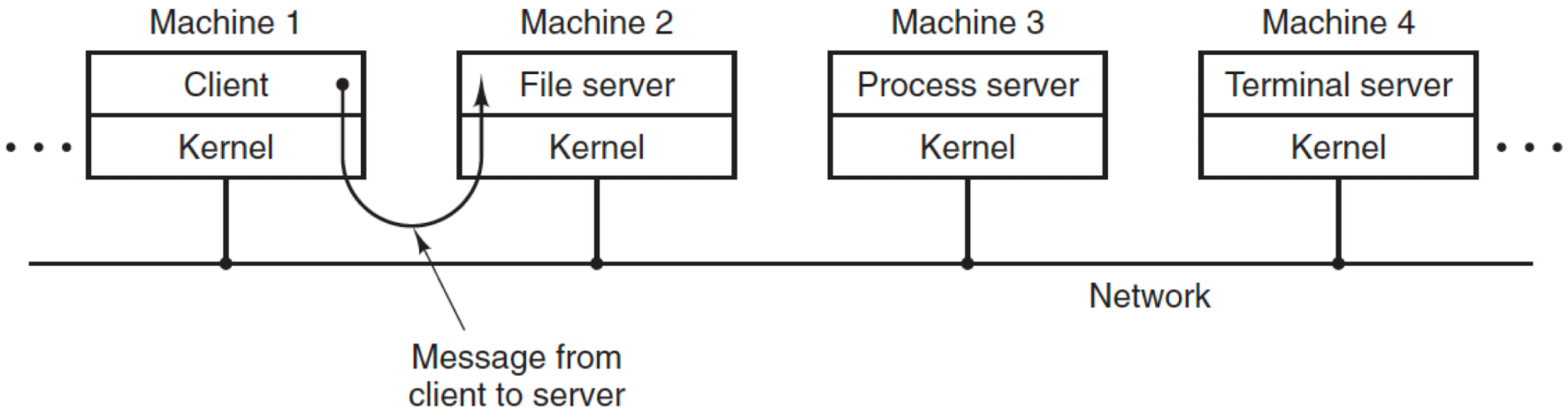
- Microkernel
  - E.g., MINIX 3
  - split the OS up into small, well-defined modules, only one of which—the microkernel—runs in kernel mode
    - the rest run as relatively powerless ordinary user processes



  - Advantage/disadvantage compared to Monolithic?

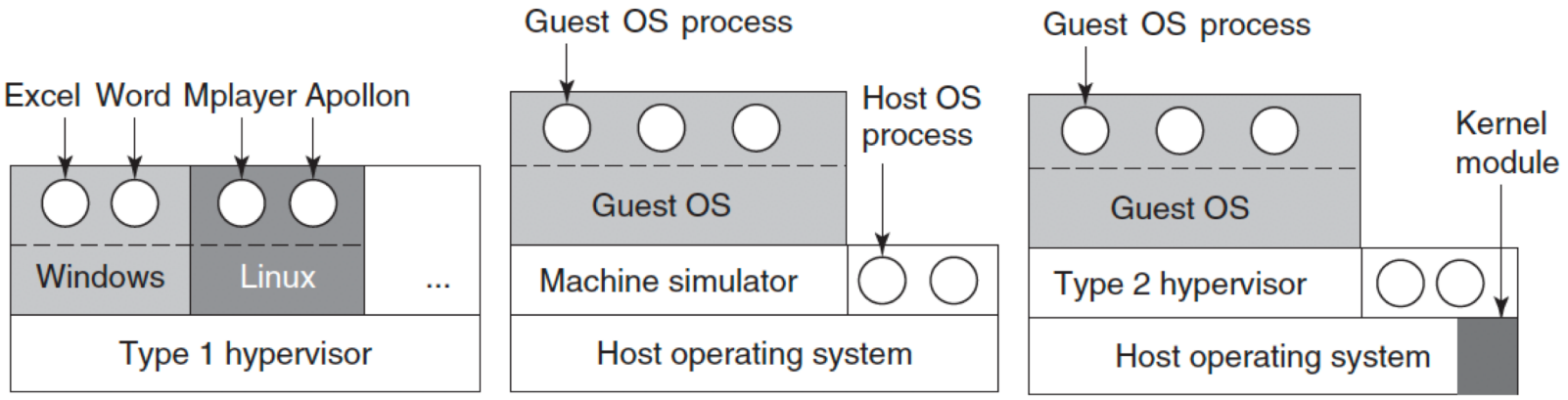
OS Structures

- Client-Server Model
  - distinguish two classes of processes
    - server: each of which provides some service
    - client: uses the services
  - client/server may run on different machines



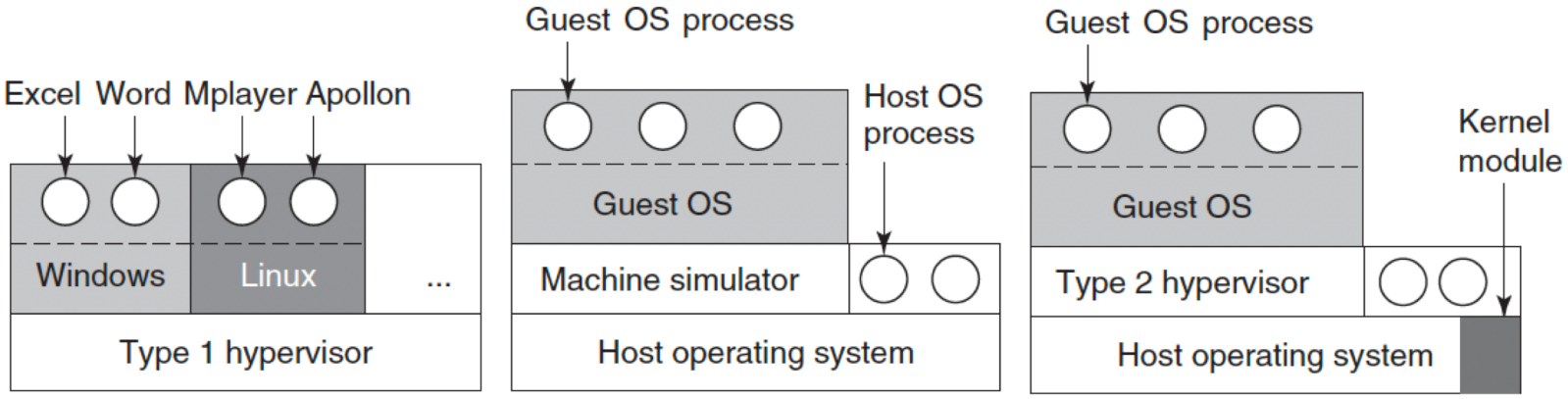
OS Structures

- Virtual Machine (VM)
  - E.g., VMWare, VirtualBox, Xen
  - Another layer of indirection beneath OS
    - Type 1 hypervisor: directly on hardware
    - Type 2 hypervisor: requires support from an underlying OS



OS Structures

- Virtual Machine (VM)
  - E.g., VMWare, VirtualBox, Xen
  - Another layer of indirection beneath OS
    - Type 1 hypervisor: directly on hardware
    - Type 2 hypervisor: requires support from an underlying OS
  - advantage/disadvantage?



Agenda

- ~~Recap~~
- ~~OS Introduction III~~
  - ~~OS Interface: System Calls~~
  - ~~OS Internal Structures~~

Questions?

*acknowledgement: slides include content from “Modern Operating Systems” by A. Tanenbaum, “Operating Systems Concepts” by A. Silberschatz etc., “Operating Systems: Three Easy Pieces” by R. Arpaci-Dusseau etc., and anonymous pictures from internet.