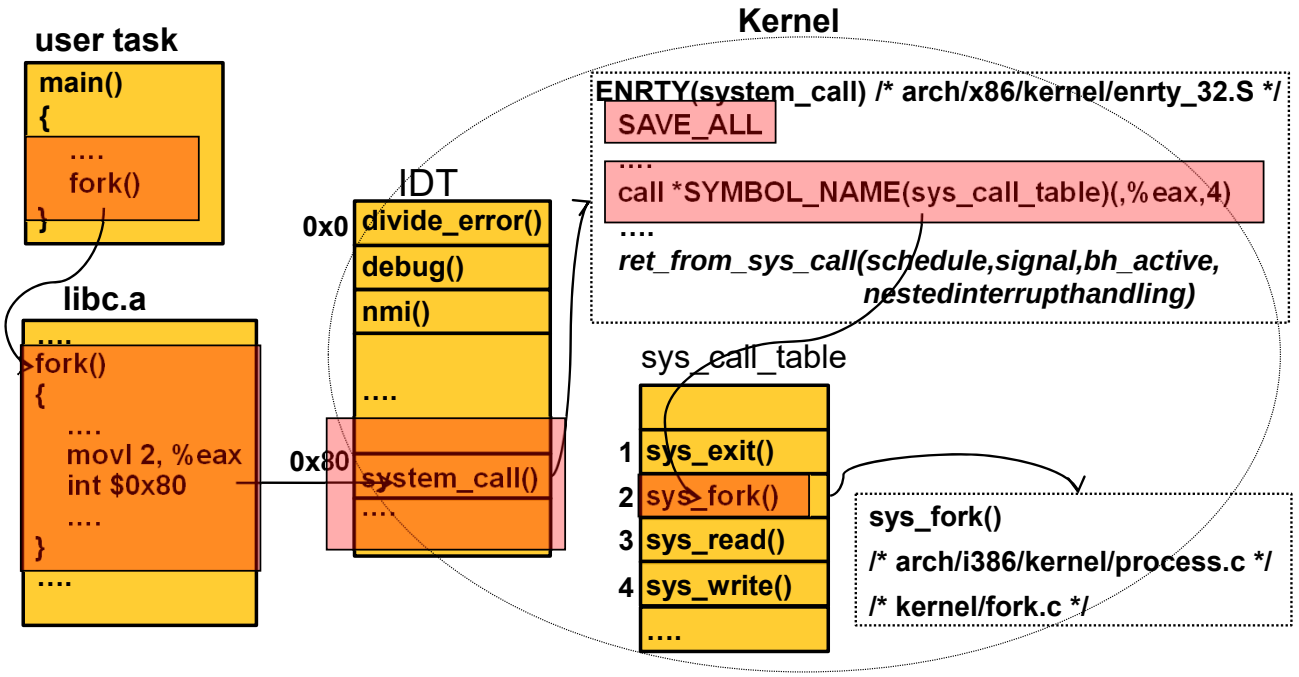
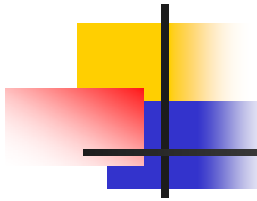
<Linux system call example>



- System program
  - System programs provide a convenient environment for program development and execution.
    - File manipulation
    - Status information
    - File modification
    - Programming language support
    - Program loading and execution
    - Communications
  - Most users' view of the operating system is defined by system programs, not the actual system calls

- Types of system calls
  - Process control
    - End, abort, create, terminate, wait event, etc.
  - File management
    - Create, open, read, write, etc.
  - Device management
    - Read, write, get device attributes
  - Information maintenance
    - Get time, date, process id, etc.
  - Communications
    - Create, delete communication connection
    - Send or receive messages



OS Design & Implementation

Design and Implementation of OS is not “solvable”,
but some approaches have proven successful

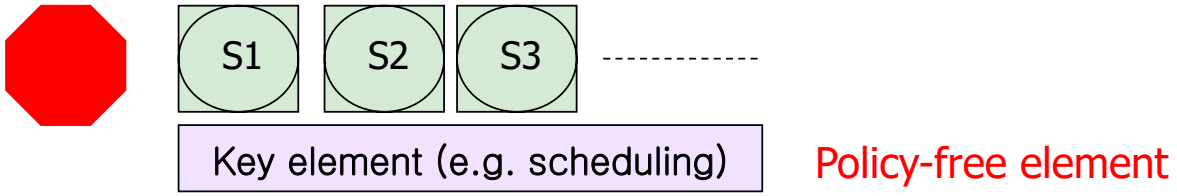
So, what’s the goals of the approaches?

- *User* goals and *System* goals
  - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
  - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient

A general mechanism insensitive to changes in policy may be more desirable

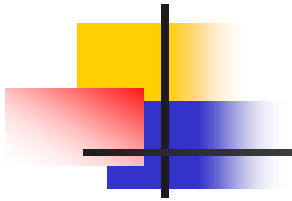
One extreme case: Micro-kernel-based approach

Policy A -> Policy B



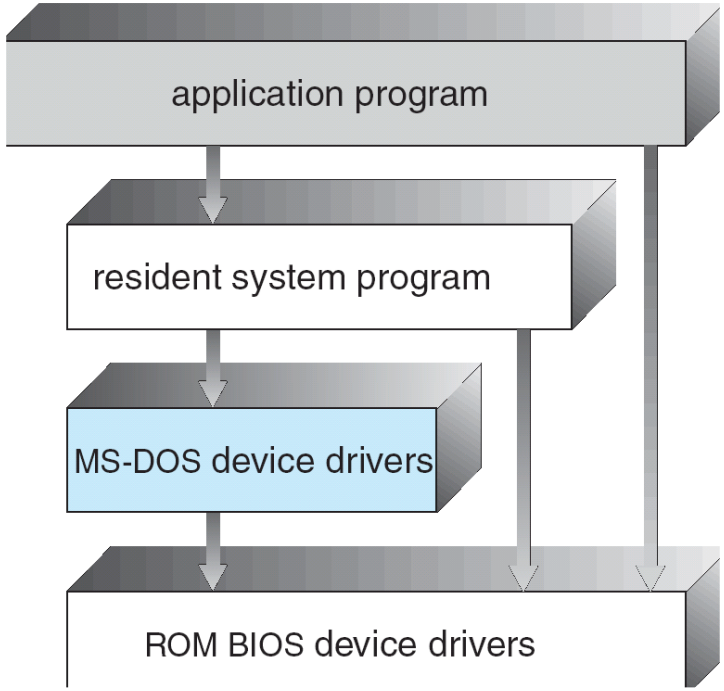
Operating system structure

- Simple structure
- Layered approach
- Microkernel
- Module
- Virtual machine



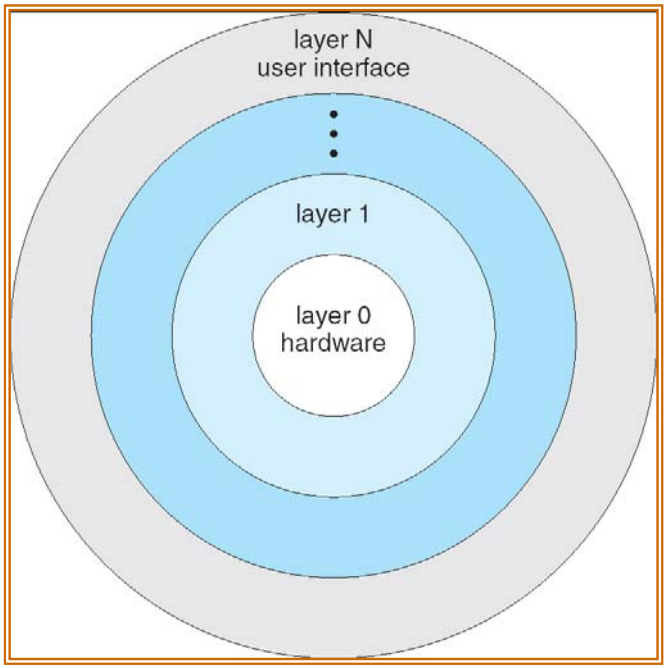
- Simple structure
  - Features
    - Not divided into modules or layers
      - No dual-mode and no H/W protection
    - Interface and levels of functionality are not well separated
  - e.g. MS-DOS
  - Application programs are able to access the basic I/O routines to write directly to the devices

- Simple structure

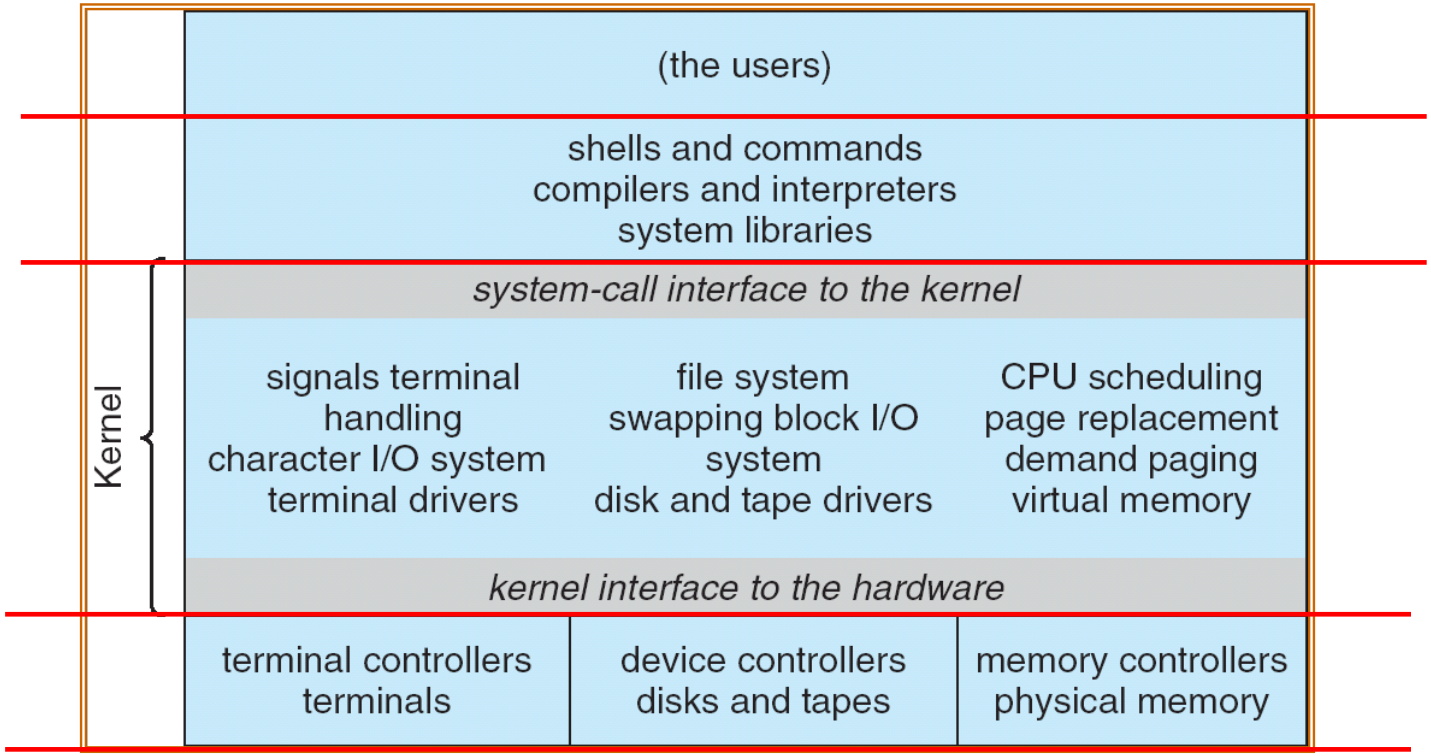


- Layered approach
  - Divided into a number of layers (levels)
    - The bottom layer (layer 0), is the hardware
    - The highest (layer N) is the user interface
  - Each layer is implemented with only those operations provided by lower-level layers
  - e.g. Traditional UNIX
  - Advantages
    - Simplicity of construction and debugging

- Layered approach



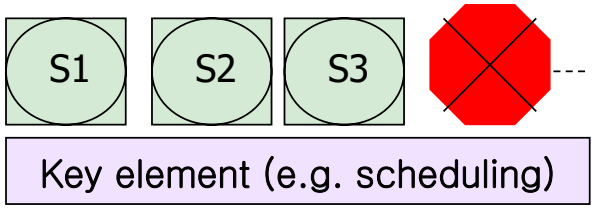
Each layer can be debugged without any concern for the rest of the system



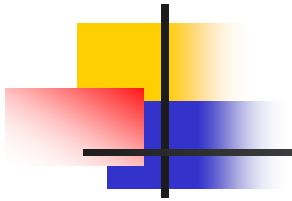
UNIX system architecture

- Microkernels
  - Moves as much from the kernel into "*user*" space
  - Benefits:
    - Easier to extend a microkernel
    - Easier to port the operating system to new architectures
    - More reliable (less code is running in kernel mode)
  - Detriments:
    - Performance overhead for message passing
  - Mach OS, QnX (real-time operation systems)

반대되는 개념:
<Monolithic kernel>

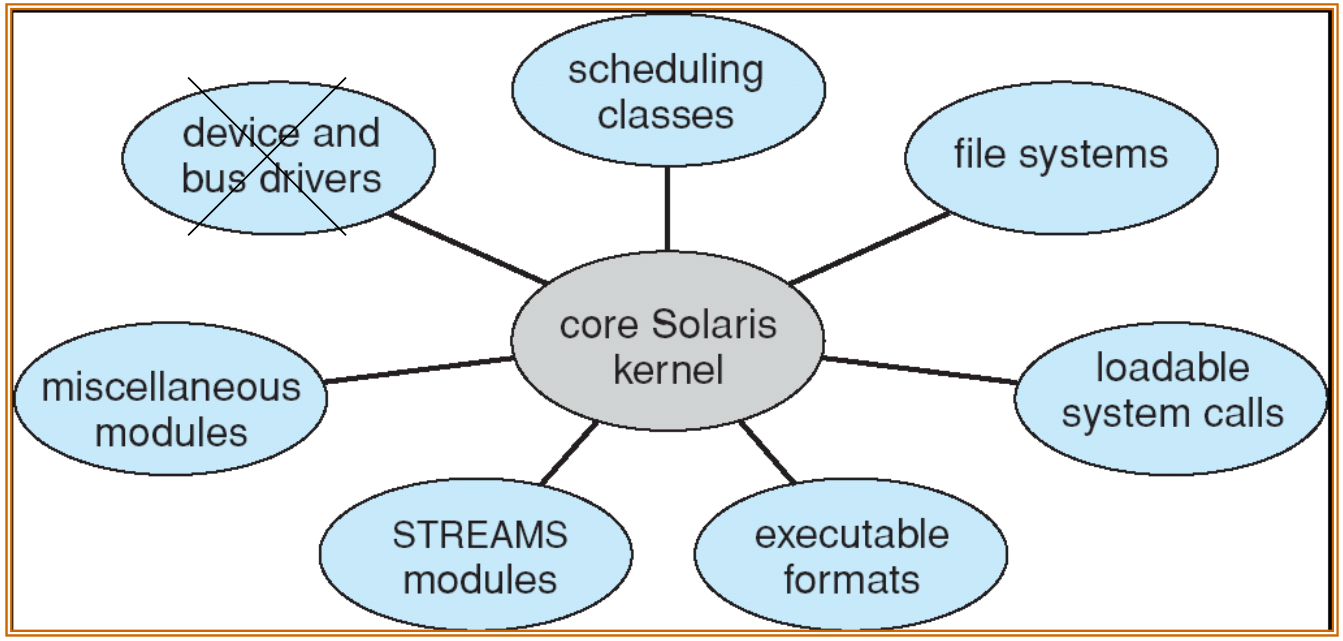


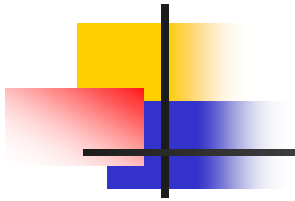
Microkernel



- Modules
  - Features
    - Each core component is separate
    - Each is loadable as needed within the kernel
  - Solaris loadable modules
  - Modern implementation of LINUX ..

- Solaris loadable modules

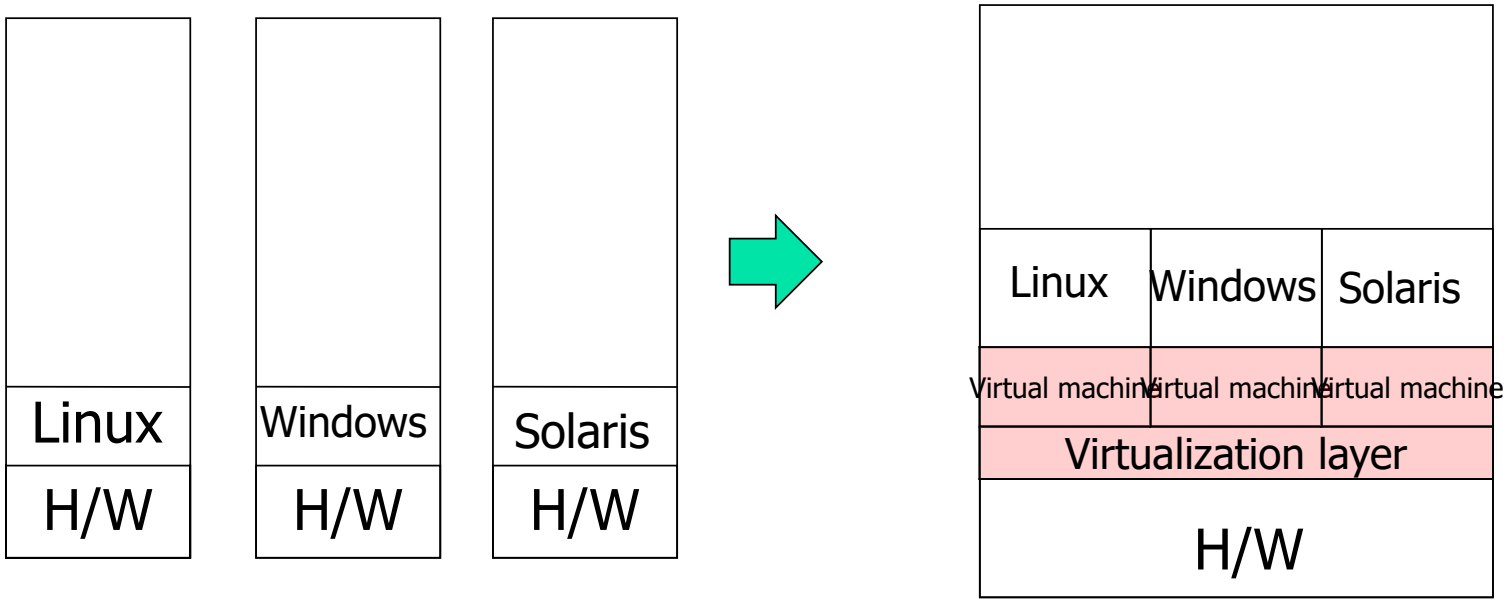




- Linux supports dynamic loading or unloading of modules
  - insmod
    - Inserting a module into the kernel
      - ex) insmod driver.o
    - Hear I am, and this is what I can do
  - rmmod
    - Removing a module from the kernel
      - ex> rmmod driver
    - I am not there anymore, don't ask me to do anything else

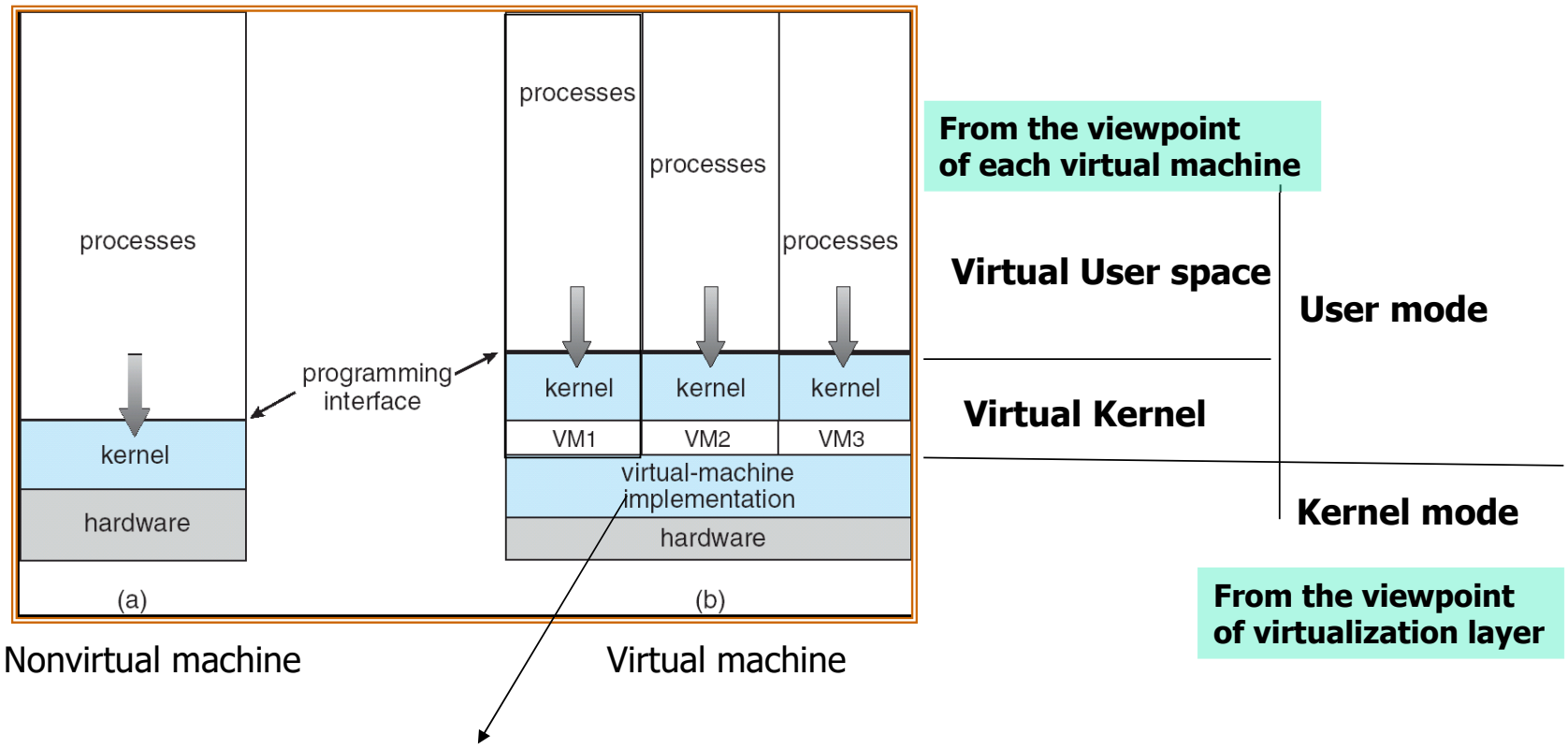
Linux => Layered approach + Modules

- Virtual machine
  - Why ?
    - Share the same hardware yet run several different operating system concurrently
  - Each process is provided with a virtual copy of the underlying computer
    - To abstract the hardware of a single computer into several different execution environments
  - That is, create illusions that each separate execution environment (OS) is running its own private computer
  - Advantages
    - Very useful concept because concurrent tests are possible for different OS's
  - Shortcomings
    - It is difficult to implement



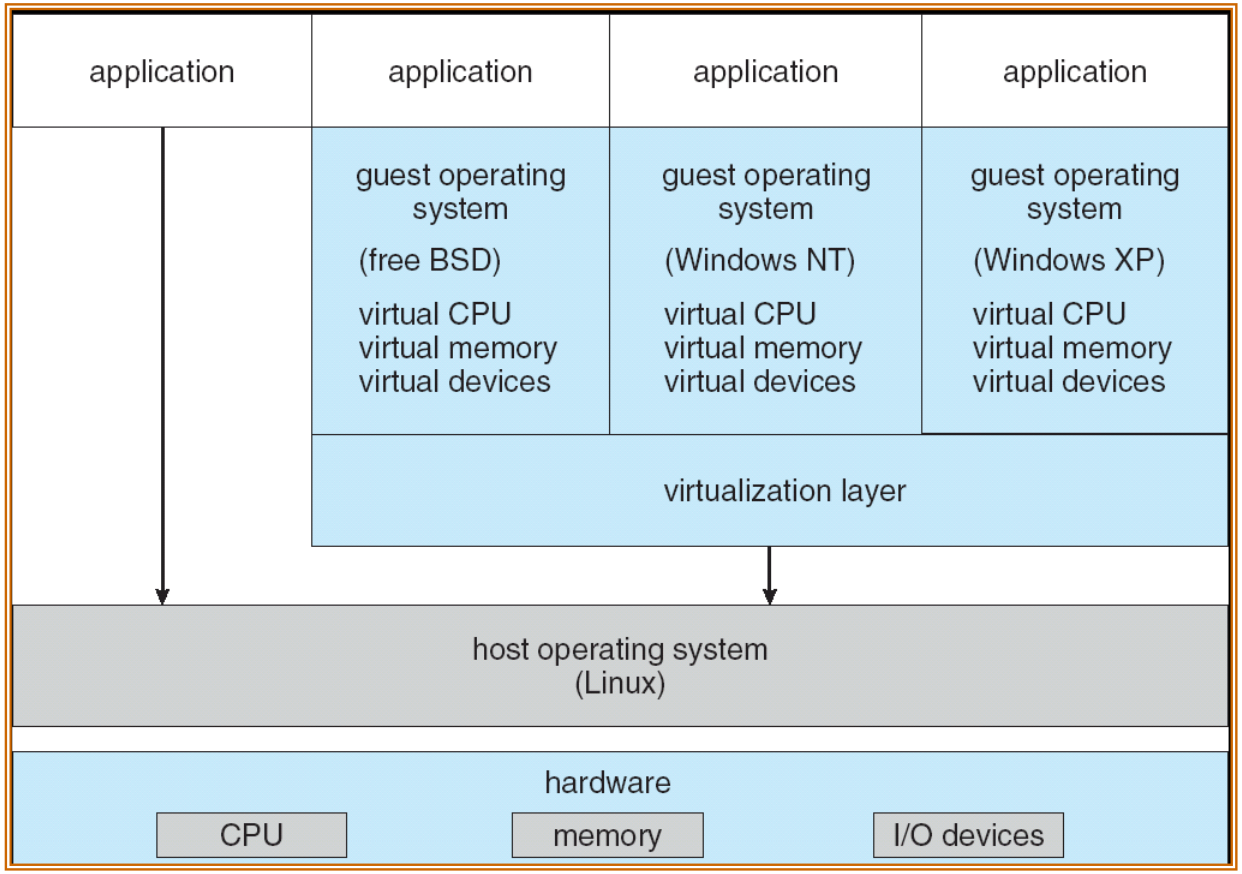
Reduce the maintenance cost greatly !

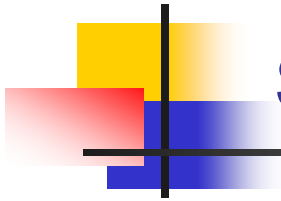
Reduce the system development time greatly !



Virtualization layer:
Multiprogramming multiple virtual machines onto a physical machine

- Vmware

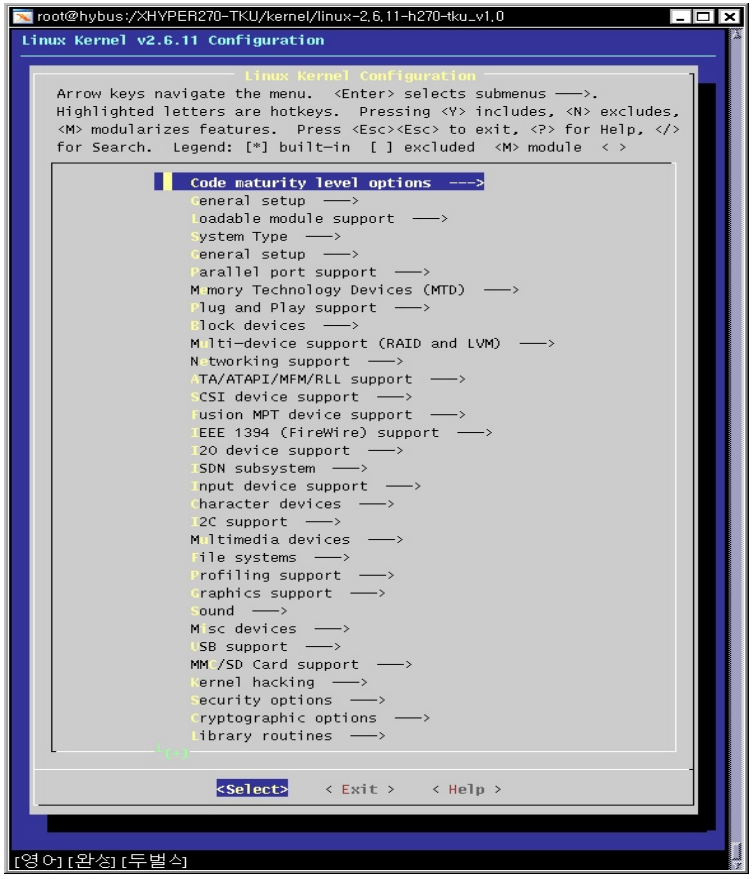

application
application
application
application
guest operating system
(free BSD)
virtual CPU
virtual memory
virtual devices
guest operating system
(Windows NT)
virtual CPU
virtual memory
virtual devices
guest operating system
(Windows XP)
virtual CPU
virtual memory
virtual devices
virtualization layer
host operating system
(Linux)
hardware
CPU
memory
I/O devices

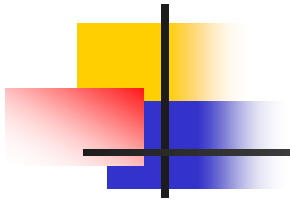


System boot

- Operating systems are designed to run on any of a class of machines;
  - System generation (SYSGEN)
    - The process in which systems are configured for each specific computer site
      - 1. asks the operator of the system
      - 2. probes the hardware directly
      - 3. reads from a given file
  - SYSGEN program
    - Obtains information concerning the specific configuration of the hardware system
      - What CPU is to be used?
      - How much memory is available?
      - What devices are available?
      - What operating system options are desired?

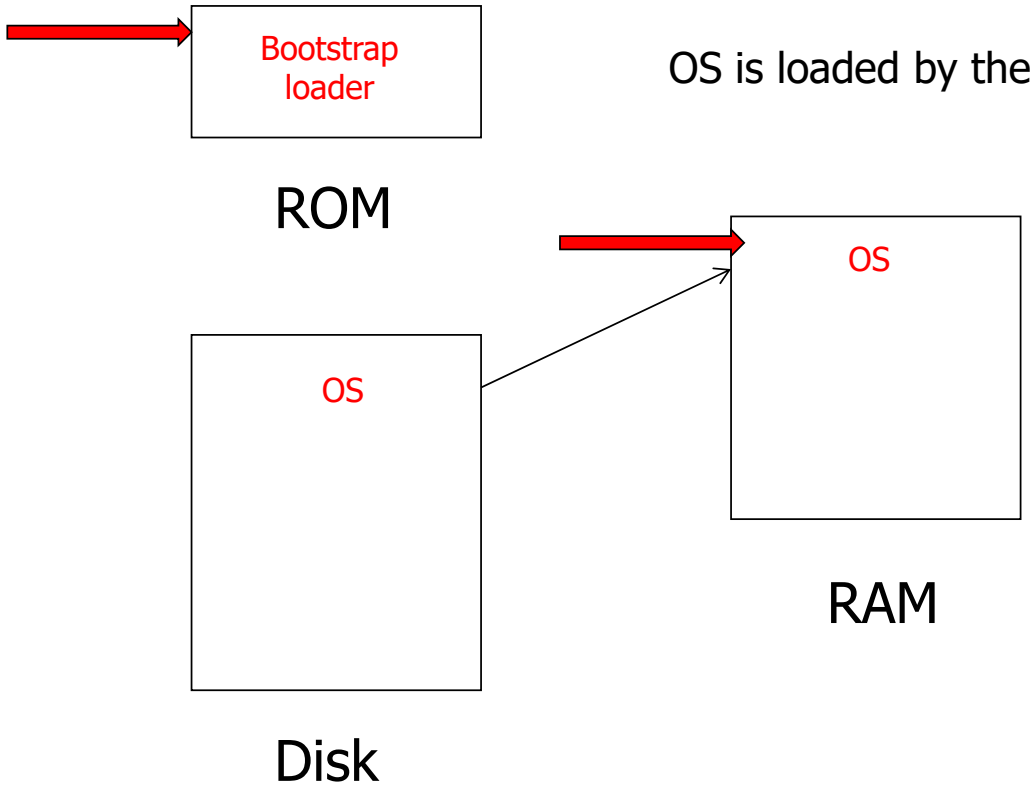
- One example of the configurations (Linux)





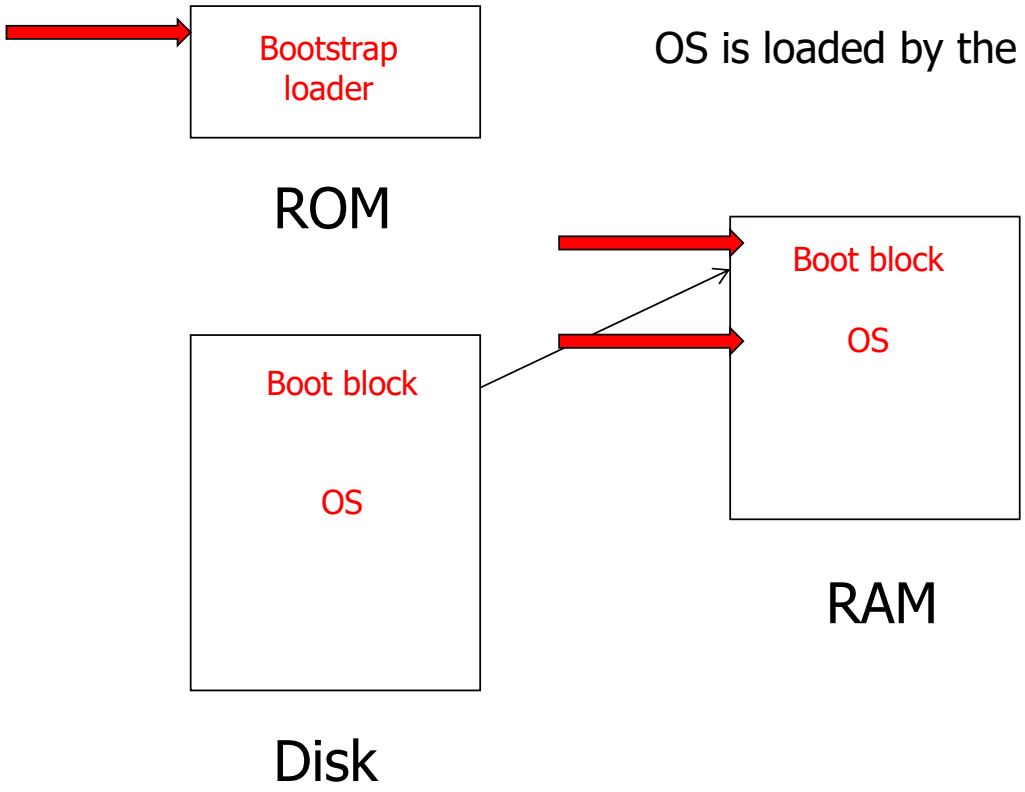
- Operating system must be made available to hardware so hardware can start it;
- Booting;
  - The procedure of starting a computer by loading the kernel
- Bootstrap loader;
  - A small piece of code that locates the kernel, loads it into memory, and starts it
  - Some systems use a two-step process where a simple bootstrap loader fetches a more complex boot program from disk
  - When the computer is powered up, execution starts at a fixed memory location (e.g. ROM)
    - ROM (or firmware) is typically used to hold initial boot code
    - Executing code there is slower than executing code in RAM

- Single-step approach



OS is loaded by the bootstrap loader

- Two-step approach



OS is loaded by the boot program (block)