OS Security

EECS 195

Spring 2019

Zhou Li

Objectives

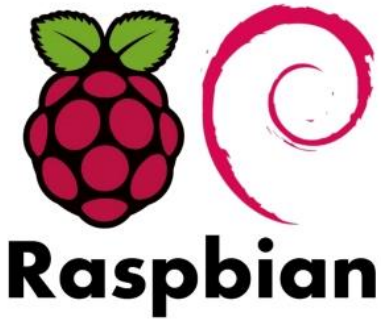
- Basic security functions provided by operating systems
- System resources that require operating system protection
- Operating system design principles
- How operating systems control access to resources
- Contemporary OS protections on memory
- The history of trusted computing
- Characteristics of operating system rootkits
- *Formally verified kernel: seL4*

Operating Systems (OS)

- OS: An executive or supervisor for a piece of computing machinery.



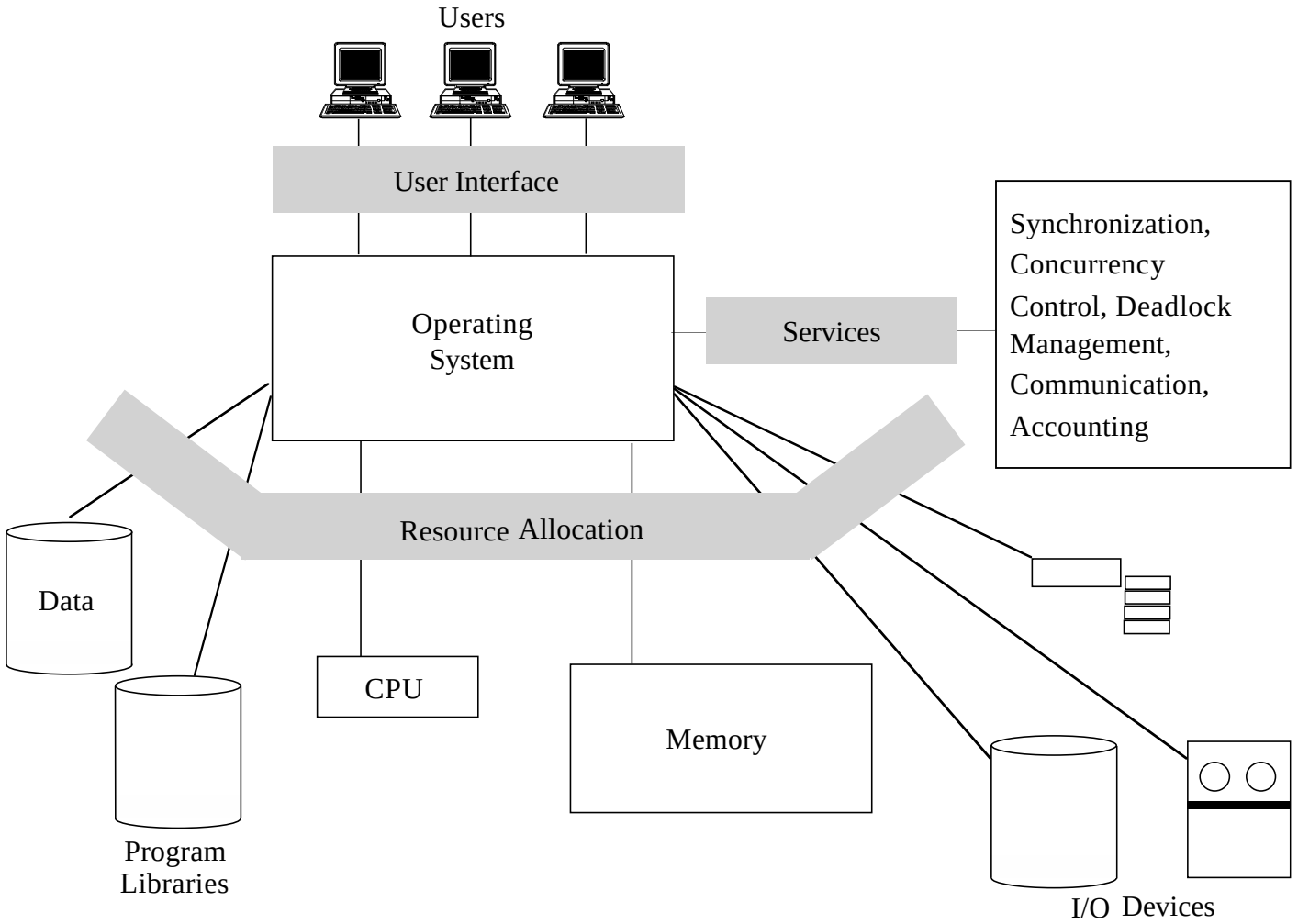





Operating System Functions

Security-relevant features:

- Enforced sharing
- Inter-process communication and synchronization
- Protection of critical data
- Guaranteed fair service
- User authentication
- Memory protection
- File and I/O device access control
- Allocation and access control to general objects

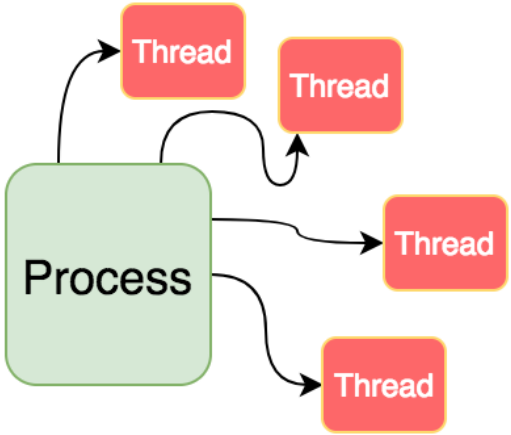


A Brief History of OS

- Single users, no OS, aka *executives*
  - Program entered in binary by “switches” physically
  - User schedules blocks of time for running the machine exclusively
  - Security issues: physical protection of computer, programs and data
- Multiprogramming and shared use, aka *monitors*
  - Waste of resources if only for one user’s tasks
  - Concepts like scheduling, sharing and concurrent use are developed
  - Executives provide service, monitors control the resources
  - Security issues: protecting one user’s program & data from others
- Personal computers, changeover from multiuser mainframes
  - Many security features like controlled sharing are forsaken

Multitasking

- Process
  - Created when running an application
  - Assigned with system resources (called *domain*): files, access to devices & communications, memory and execution time.
  - OS switches control between processes, allocating, deallocating and reallocating resources for processes.
- Thread
  - A process consists of one or more threads, separate streams of execution.
  - A thread executes *in the same domain* as other threads.
  - Much less OS cost during thread switching.



Protected Objects

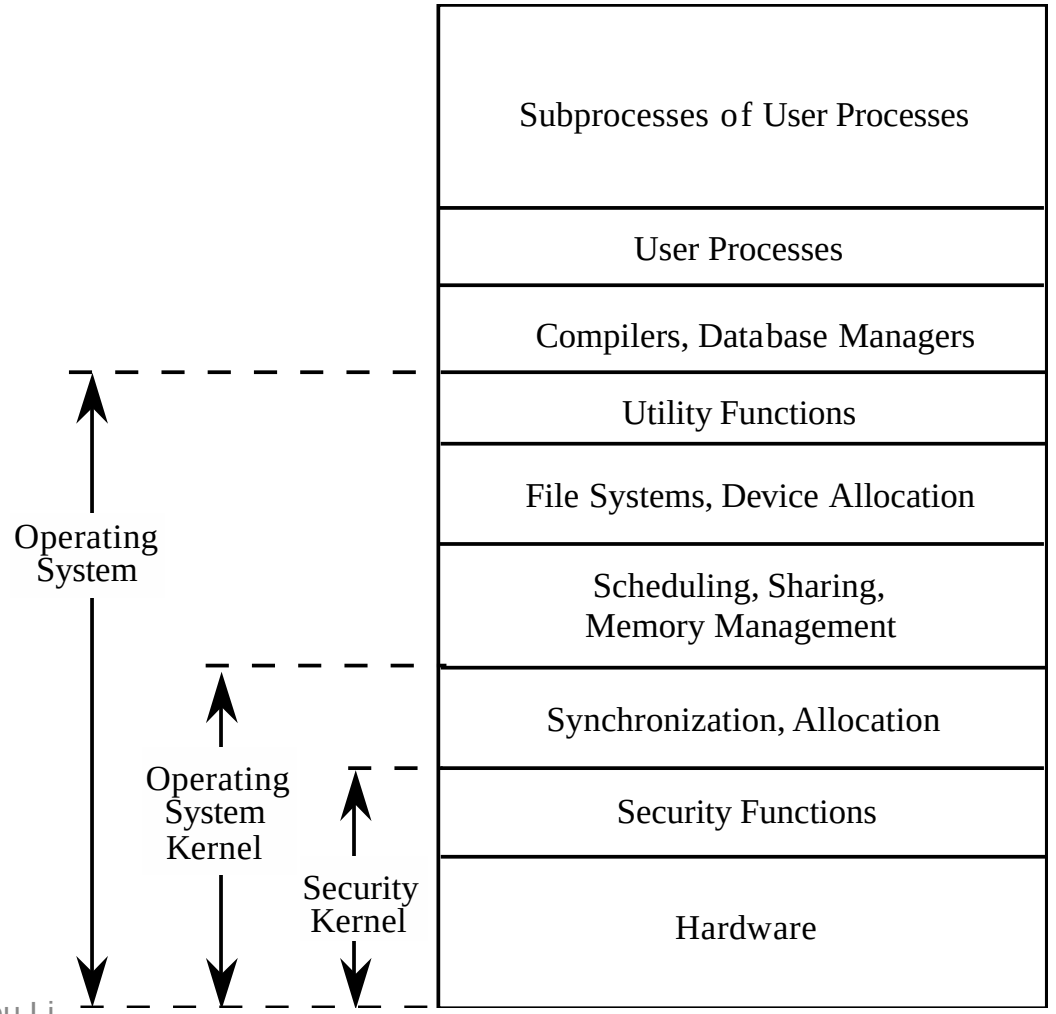
- Memory
- Sharable I/O devices, such as disks
- Serially reusable I/O devices, such as printers
- Sharable programs and subprocedures
- Networks
- Sharable data

- Granularity for access control
  - Larger the level, easier to implement, but could lead to over-privilege

OS Layered Design to Protect Objects

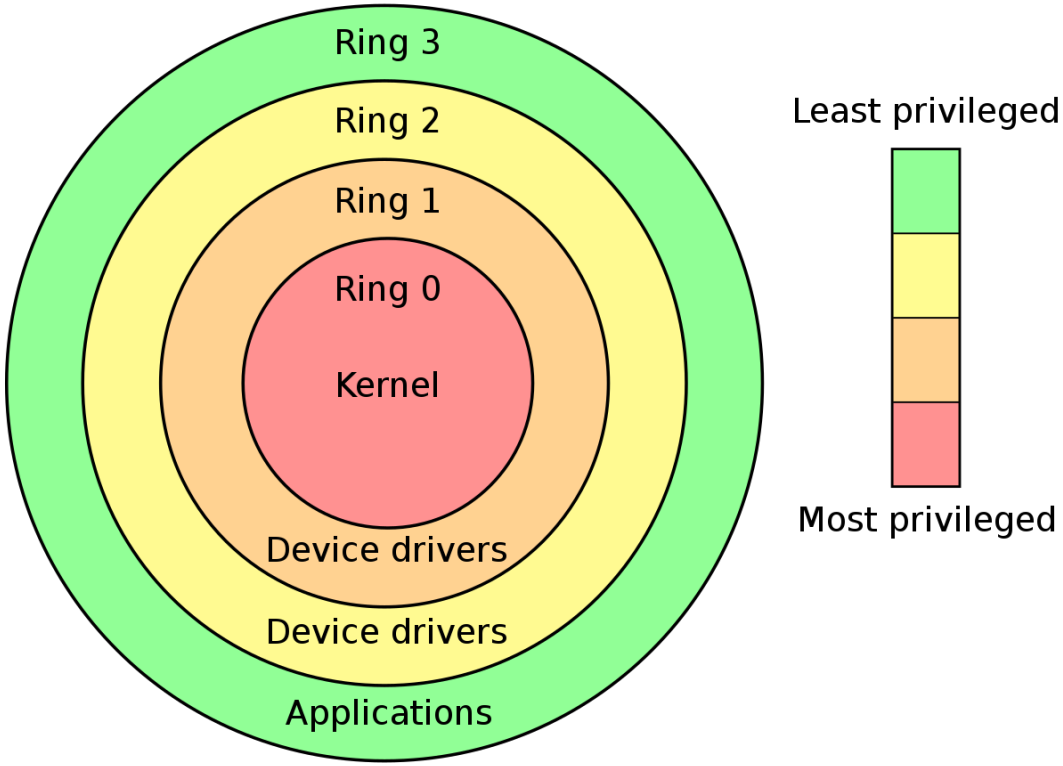
OS layers

- Security kernel: enforce security
- OS kernel: allocate primitive resources such as time or access to hardware devices
- Other OS functions: implement user's interface to hardware



Protection Rings (Hardware)

- Hardware-enforced by some CPU architectures that provide different CPU modes at the hardware or microcode level.
- Process and resources are in different rings (privilege levels).
- When a lesser privileged process tries to access a higher privileged process / resource, a general protection fault exception is reported by the OS.
- x86 implementations (Ring 0-3)
  - Windows, Linux, macOS uses Ring 0 (kernel / privileged mode) and Ring 3 (user mode)
  - DOS runs everything at Ring 3



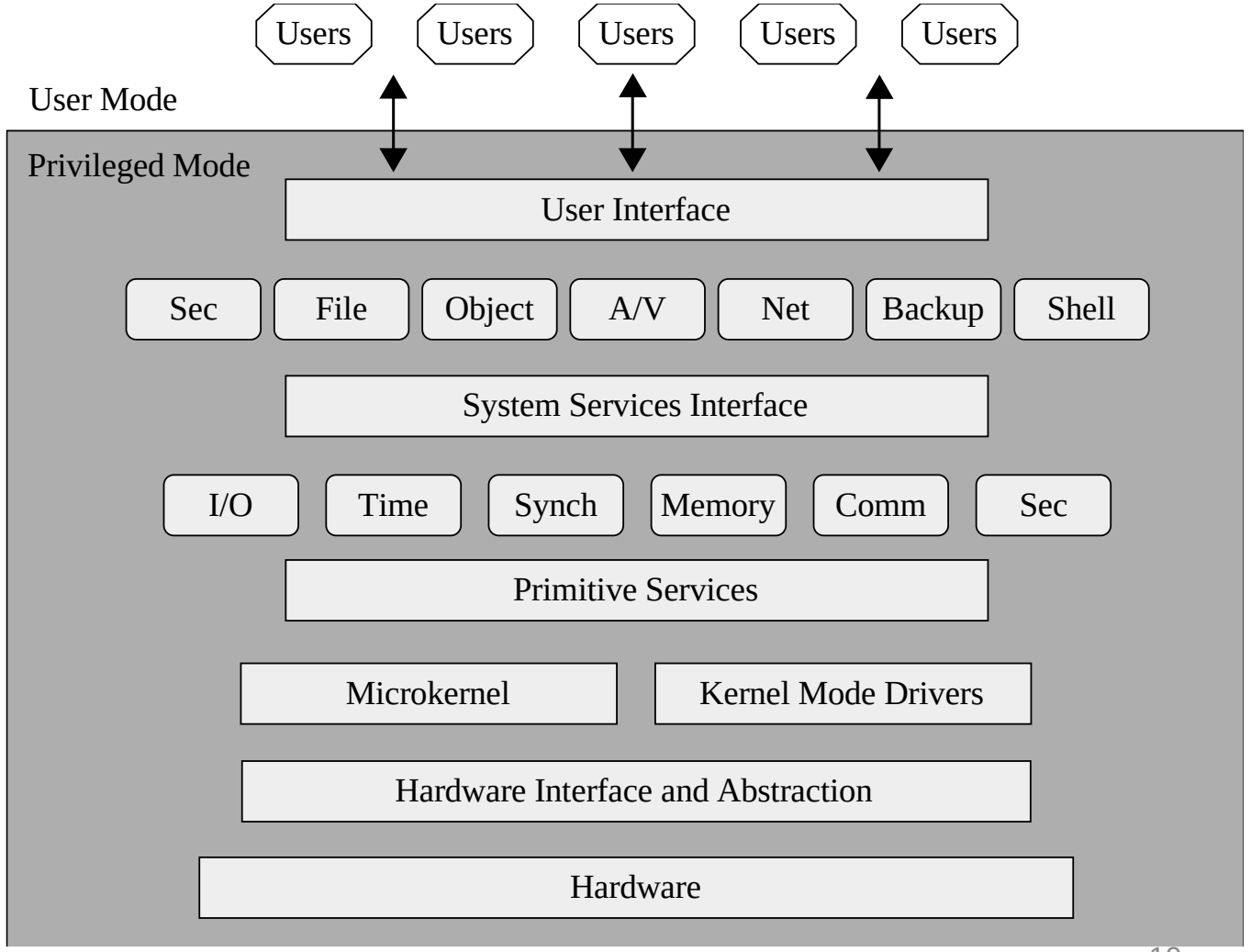
Privilege rings for the x86 available in protected mode

https://en.wikipedia.org/wiki/Protection_ring

Modular OS Design

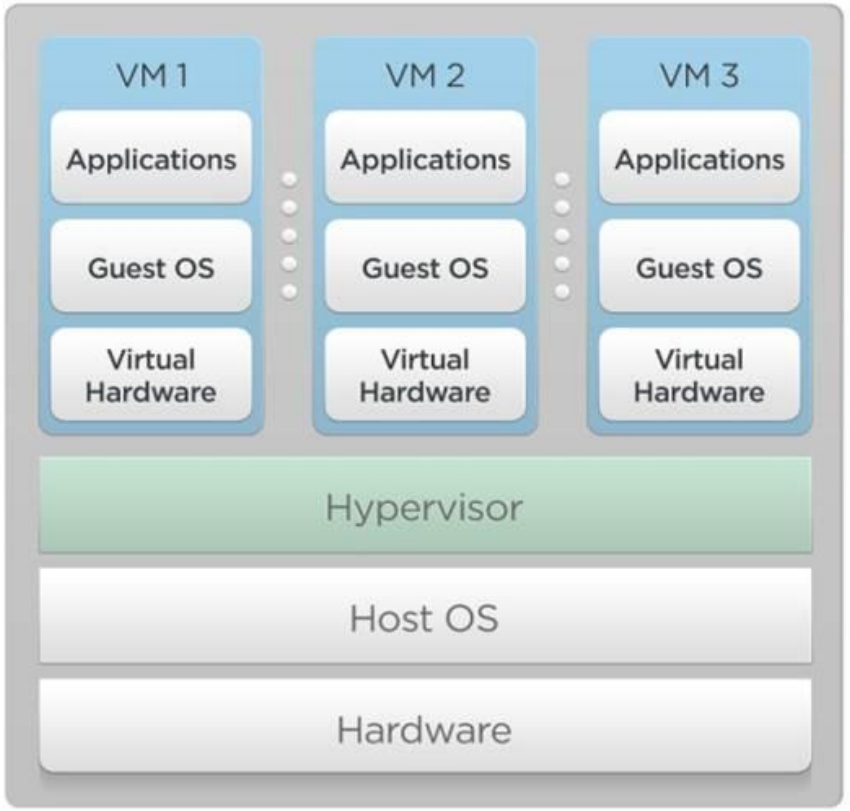
Modules come from different sources

- Hardware and drivers from device manufacturers
- Anti-virus from software vendors “hooks” into OS
- Cannot trust one another by default



Virtualization

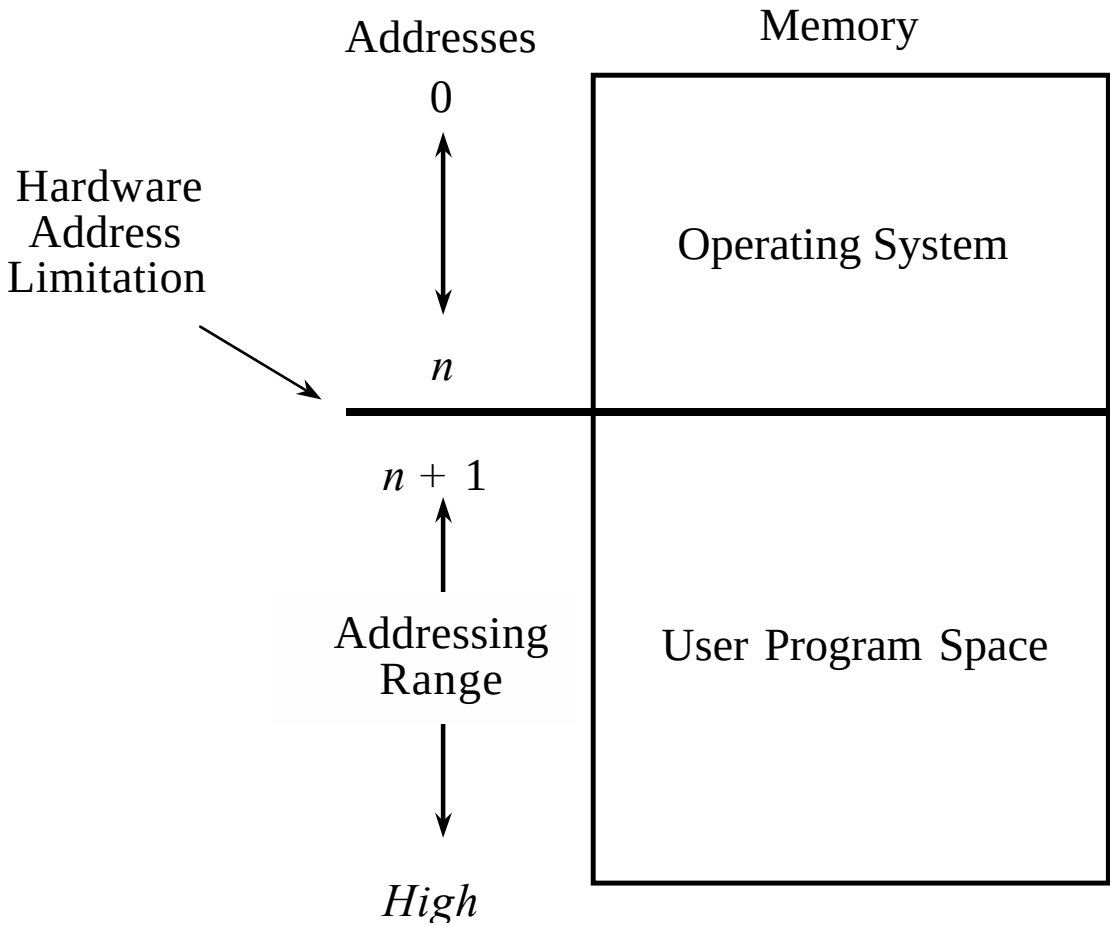
- With virtualization, the OS presents each user with just the resources that user should see
- The user has access to a virtual machine (VM), which contains those resources
- The user cannot directly access resources that are outside the VM (sandbox)
- A hypervisor, or VM monitor, is the software that hosts VM
  - Translates access requests between the VM and the OS
  - Can support multiple OSs in VMs simultaneously



Memory management

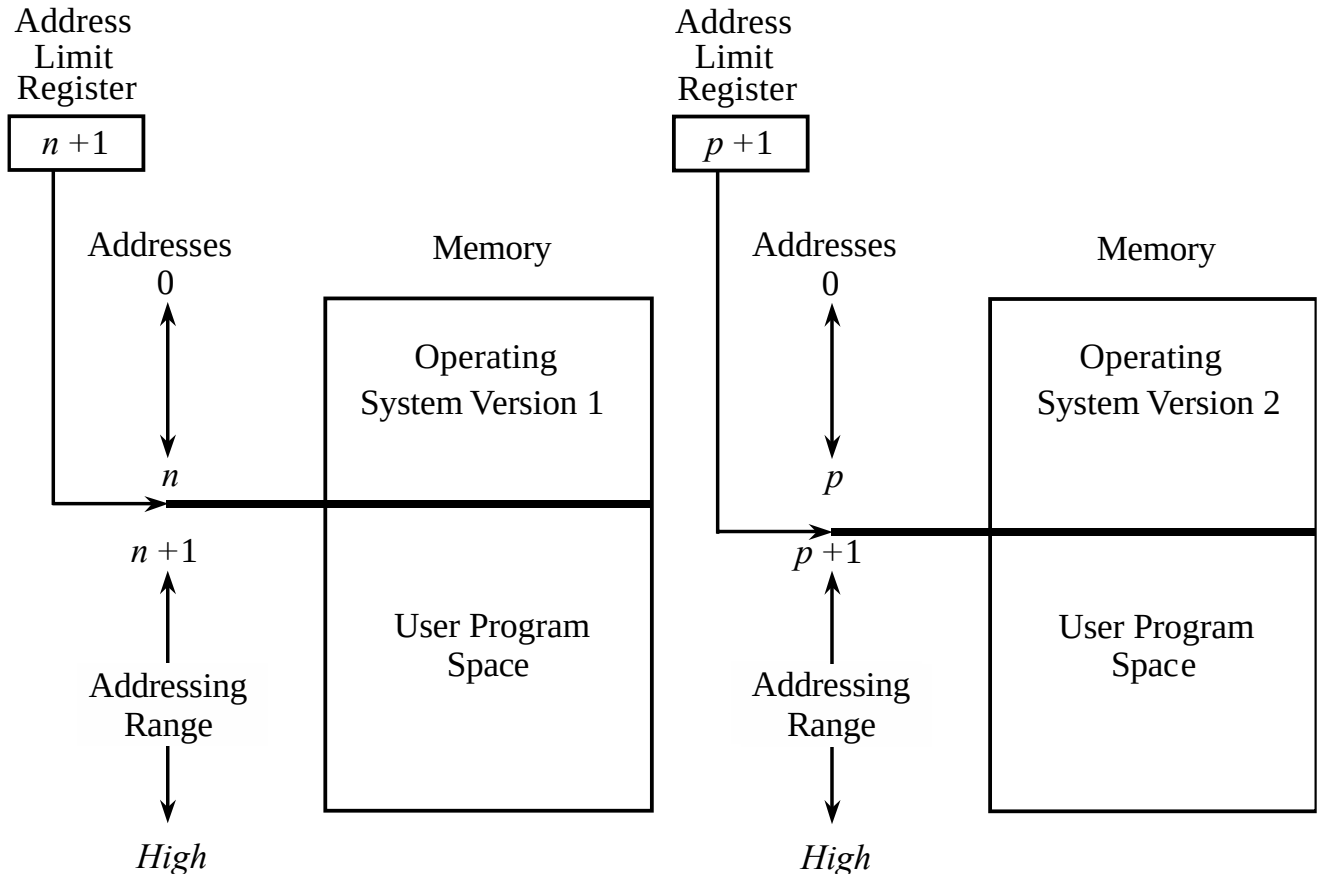
Hardware Protection of Memory

- Goal: program can share selected part of memory with other programs and even other users; OS and user can coexist in memory without interference
- Approach: fence
  - OS and user reside in different sides of memory (fixed)
  - Too restrictive
  - Excess space is wasted



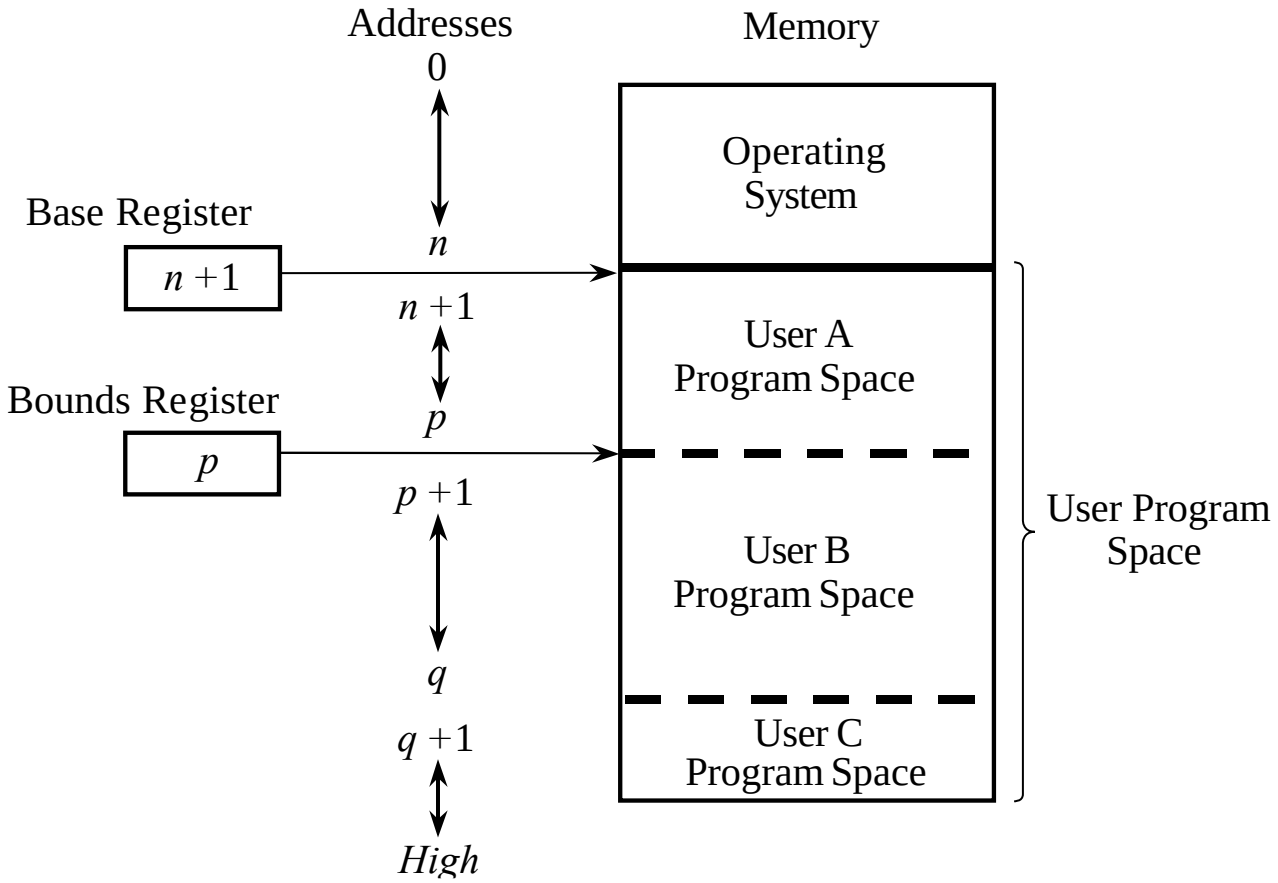
Fence Registers

- Containing the address of the end of the OS
- Boundary can be changed
- Program's instruction can be executed if the data's address is greater than fence address
- Cannot protect one user from another



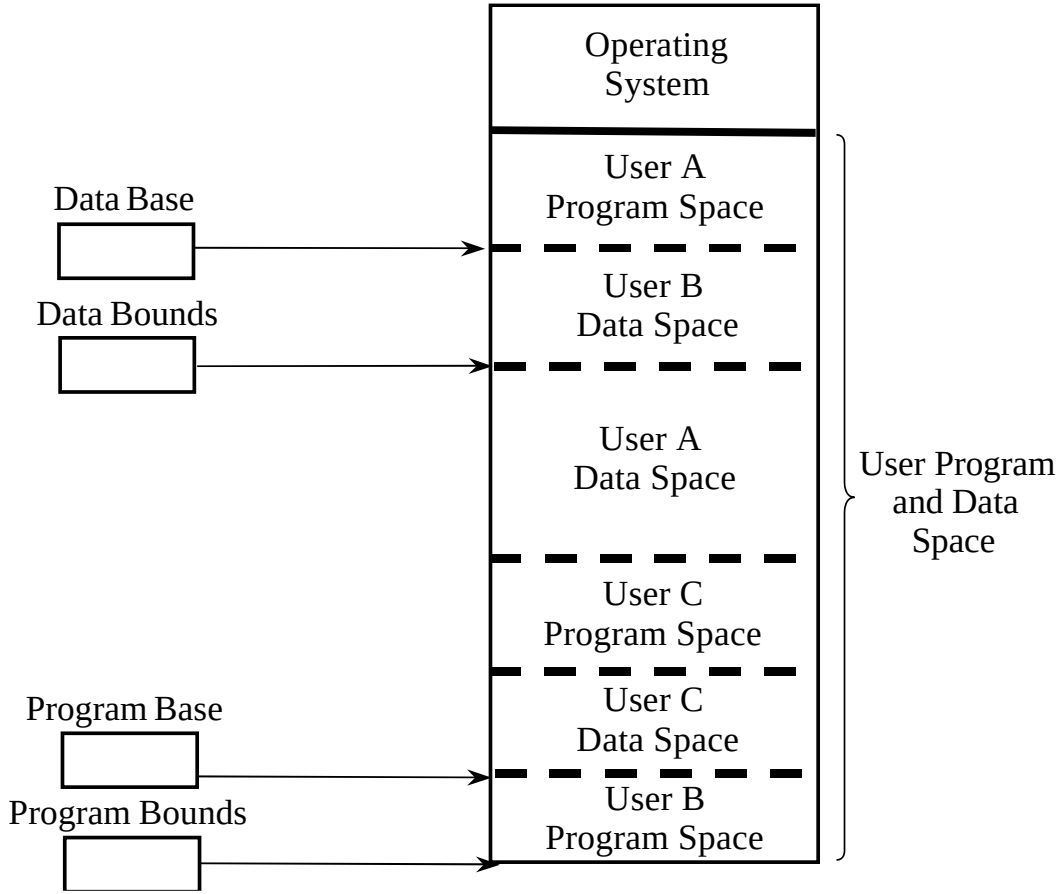
Base/Bounds Registers

- Base register: variable fence register about lower-bound
- Bounds register: upper-bound for a user program
- How to protect program address from modification by another user?
  - Context switch
  - OS changes content of base and bounds registers to reflect true address space of another user



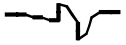
Two Pairs of Base/Bounds Registers

- Erroneous addresses inside a user space can impact program
  - Subscript is out of range
  - Undefined variable reference an address inside program space
- Solution: two pairs of registers
  - Data registers
  - Program registers



Tagged Architecture

- What if we want to protect some data value but not all?
- Solution: tagged architecture
  - Each word of machine memory has one or more extra bits to identify its access rights
- Example:
  - Burroughs B6500-7500 three tag bits to separate data words, descriptors (pointers) and control words (stack pointers & addressing control words)
- Problem: compatibility with OS

Tag	Memory Word
R	0001
RW	0137
R	0099
X	
X	
X	
X	
X	
X	
R	4091
RW	0002

Code: R = Read-only RW = Read/Write
X = Execute-only

Virtual Memory

- Two more approaches for memory protection
  - Segmentation
  - Paging
- Can be implemented on top of a conventional machine structure
- Designed between 1965 and 1975