CS 547: Foundation of Computer Security

S. Tripathy
IIT Patna

Previous *Class*

- Protection in General-Purpose Operating Systems
  - Separation vs. Sharing
  - *Memory protection*

Present Class

- Protection in General-Purpose Operating Systems
  - Segmentation and Paging
  - Access Control Matrix
  - Access Control Lists vs. Capabilities

Operating System

- An operating system allows different users to access different resources in a shared way

- The operating system needs to control this sharing and provide an interface to allow this access

- Protection:
  - Mechanism that prevent accident or intentional misuse of a system
- Aspects of Protection
  - Identification and authentication are required for the access control

Security Methods of Operating Systems

- Want to be able to share resources without compromising security
  - Isolate different processes
  - Share all or nothing
  - Share via access limitation **(granularity)**
  - Share by capabilities
  - Limit use of an object

Communication Between Address Spaces

Processes communicate among address spaces via ***interprocess communication***

- Byte stream (e.g., **`pipe`**)
- Message passing (send/receive)
- File system (e.g., read and write files)
- Shared memory

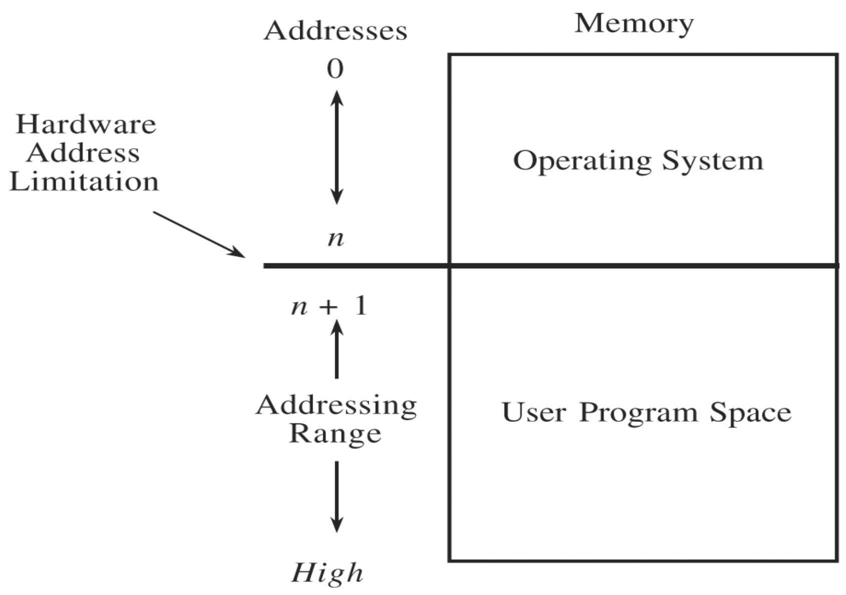
Bugs can propagate from one process to another

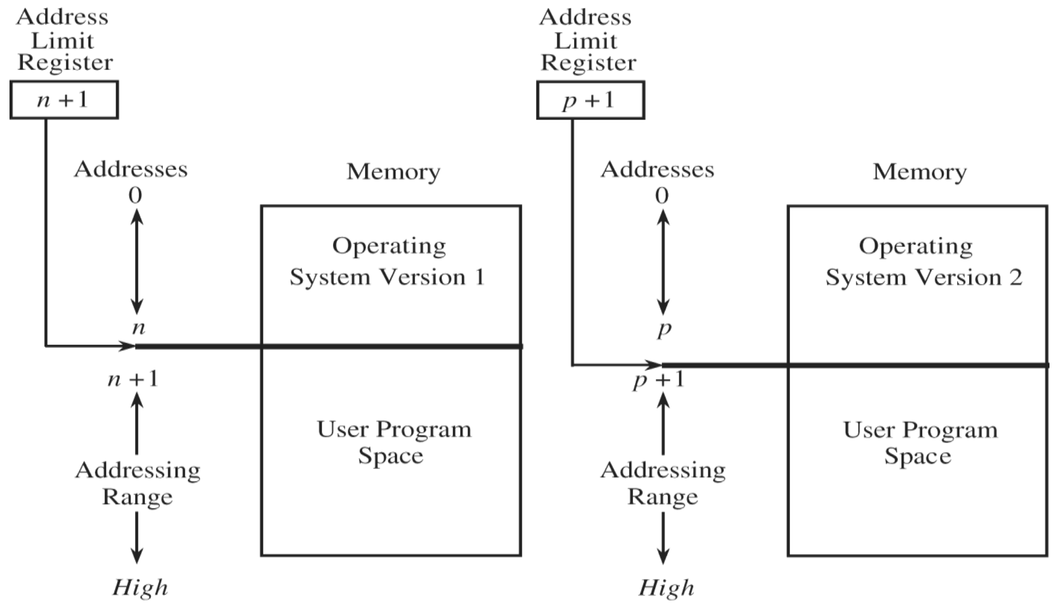
Memory and Address Protection

- Prevent program from corrupting other programs or data, maybe operating system also
- Often, the OS can exploit hardware support for this protection, so it's cheap
  -
- Memory protection is part of translation from virtual to physical addresses
  - Memory management unit (MMU) generates exception if something is wrong with virtual address or associated request
  - OS maintains mapping tables used by MMU and deals with raised exceptions

Fence Register – Memory Protection

- Fence register
  - Simplest of all protection
  - Confine the user to one side of a boundary
  - Used to separate OS and Program (wasteful use of space)
  - Protects a user from an OS but not a user from another user
  - Single user only
  - Exception if memory access below address in fence register
  - Protects operating system from user programs

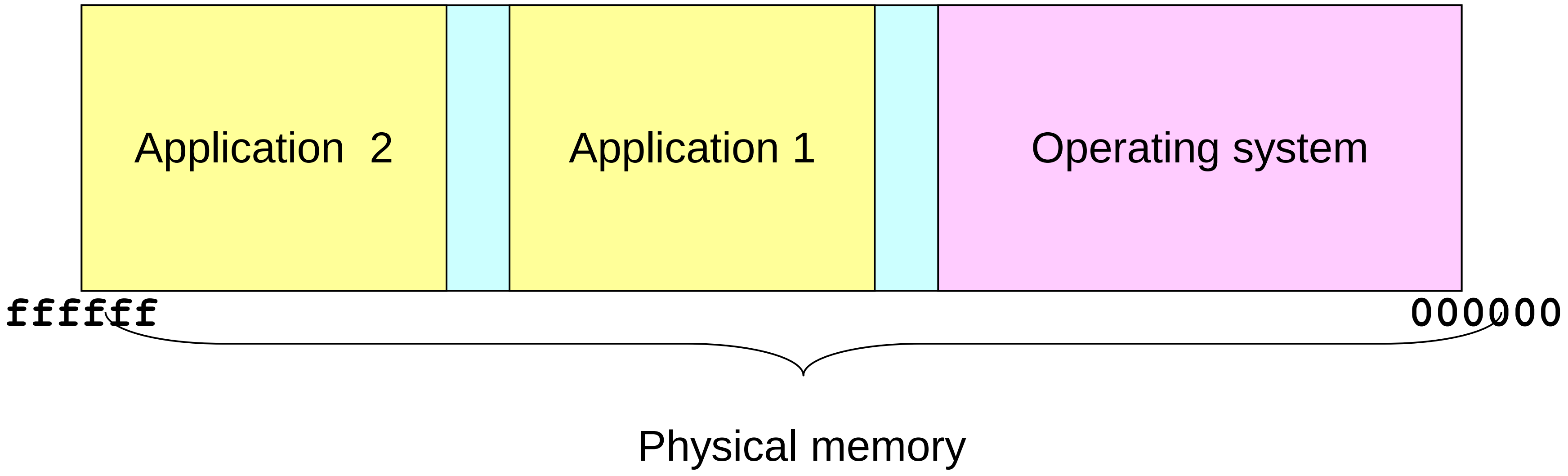




Multiprogramming Without Memory Protection

When a program is copied into memory, a ***linker-loader*** alters the code of the program (e.g., loads, stores, and jumps)

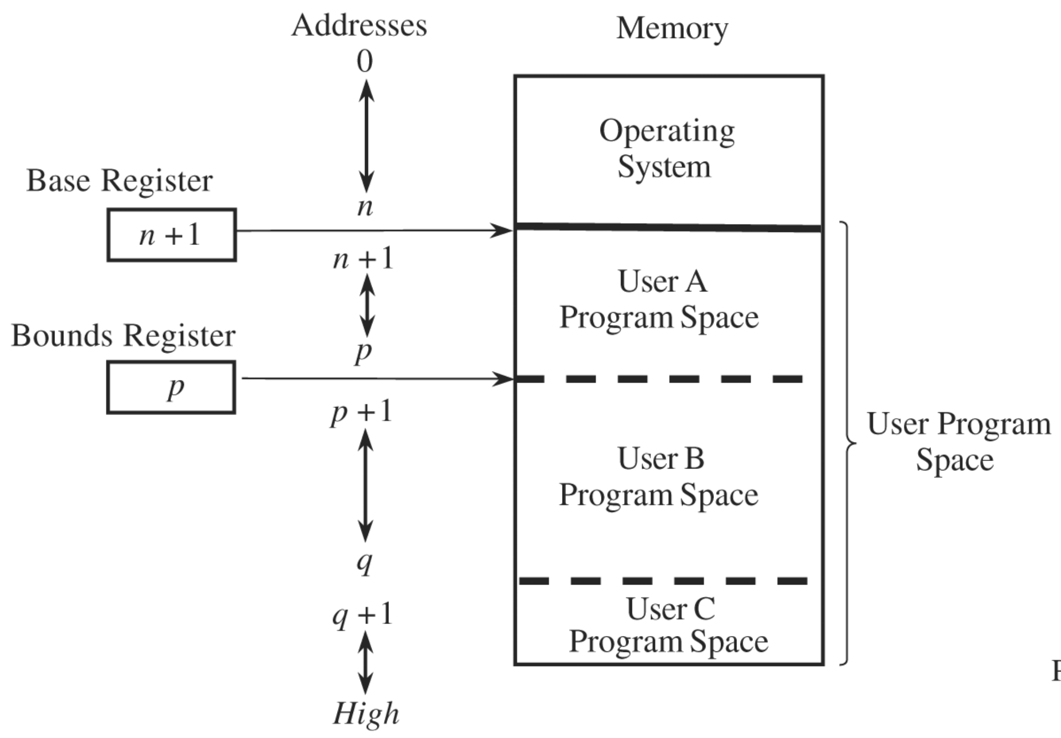
- To use the address of where the program lands in memory

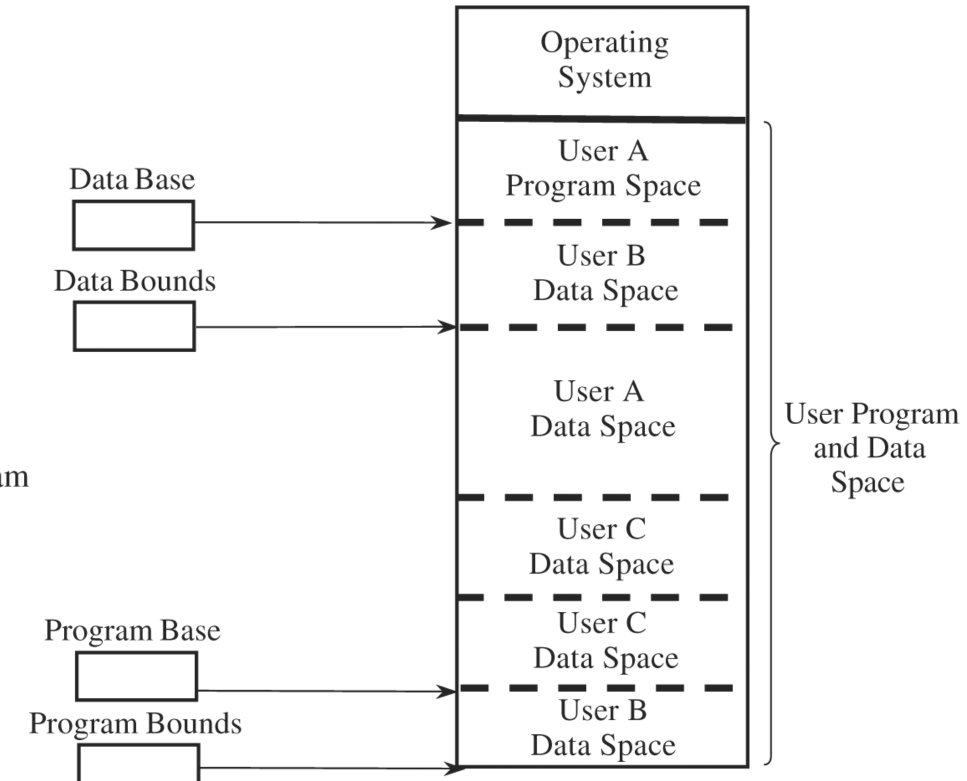


Bugs in any program can cause other programs to crash, even the OS

Base/Bound Register

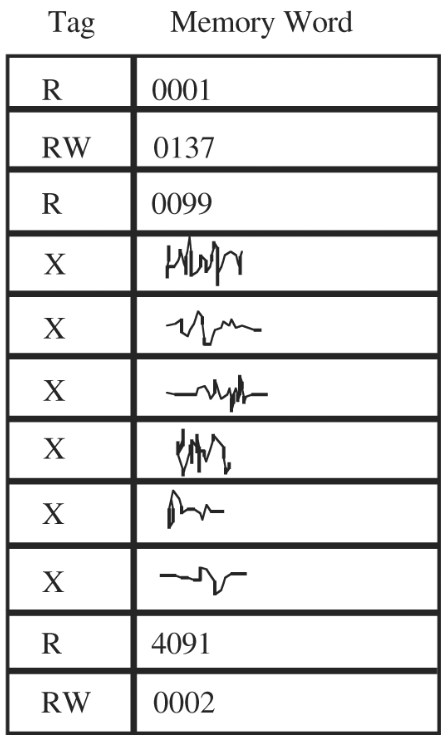
- Base/bounds register pair
  - Created for a multiuser environment
  - Exception if memory access below/above address in base/bounds register
  - Different values for each user program
  - Maintained by operating system during context switch
  - Limited flexibility





Tagged Architecture

In base/bound, it is an all or nothing on the sharing of data. It is hard to manage because of it contiguous data space.
Tagged Architecture- every word of machine memory has one extra bits to identify access right



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

Multiprogrammed OS With Memory Protection

Multiprogrammed OS With Memory Protection:

keeps user programs from crashing one another and the OS

Two hardware-supported mechanisms

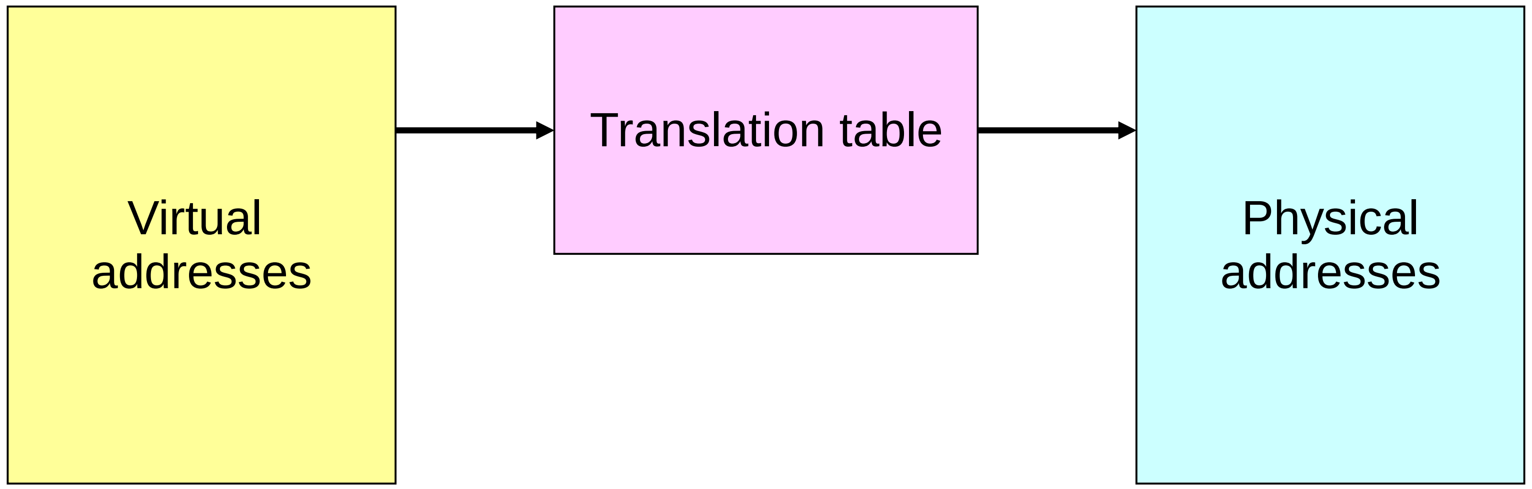
- Address translation
- Dual-mode operation

Address Translation

Recall that each process is associated with an ***address space***, or all the *physical* addresses a process can touch

However, each process believes that it owns the entire memory, starting with the *virtual* address 0
The missing piece is a translation table to translate every memory reference from virtual to physical addresses

Address Translation Visualized



More on Address Translations

- Translation provides protection
  - Processes cannot talk about other processes' addresses, nor about the OS addresses
  - OS uses physical addresses directly
    - No translations

Protection Techniques

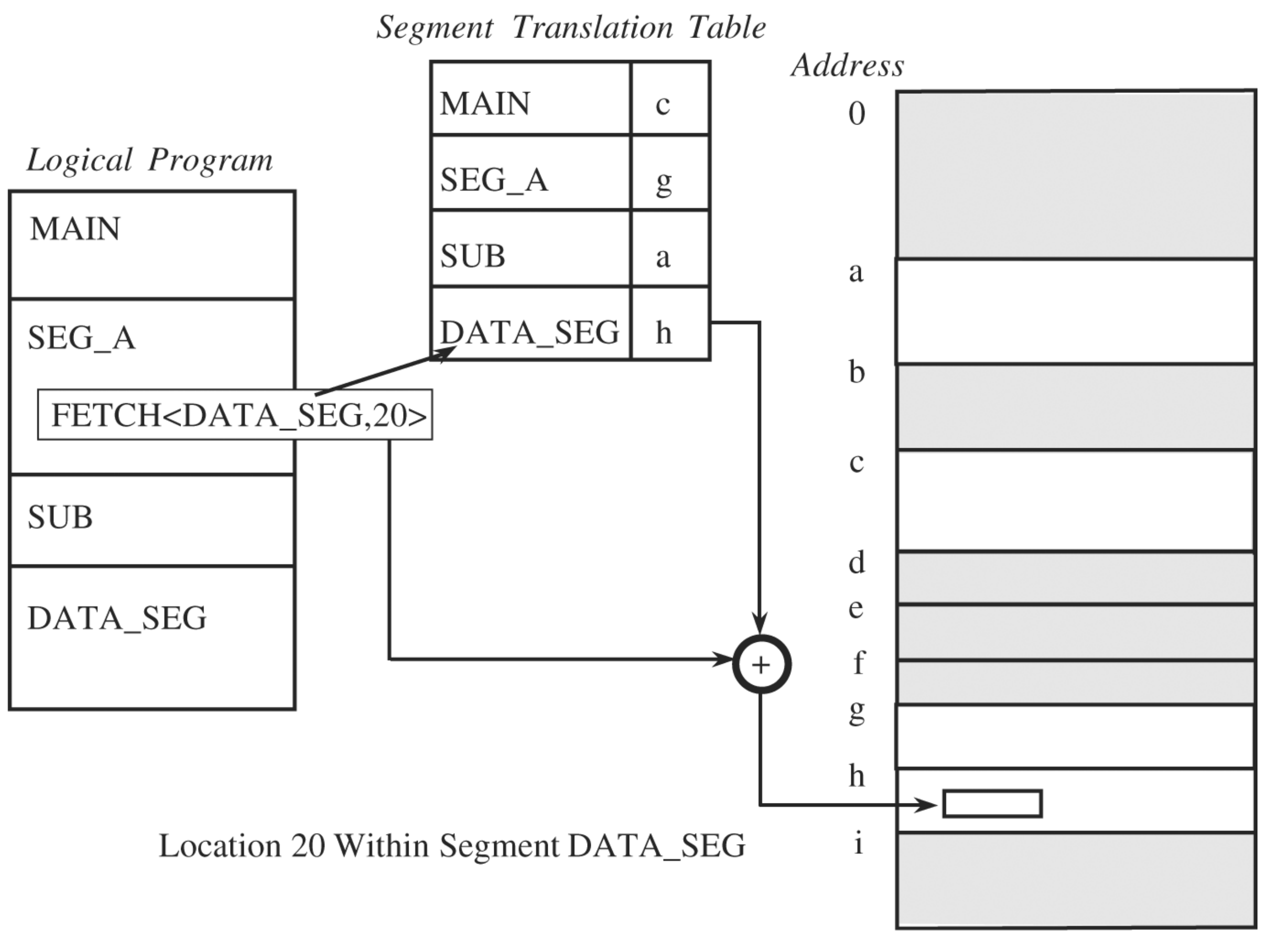
- Tagged architecture
  - Each memory word has one or more extra bits that identify access rights to word
  - Very flexible
  - Large overhead
  - Difficult to port OS from/to other hardware architectures
- Segmentation
- Paging

Segmentation

- Each program has multiple address spaces (segments)
- Could use different segments for code, data, and stack
  - Or maybe even more fine-grained, e.g., different segments for data with different access restrictions
- Virtual addresses consist of two parts:

  <segment name, offset within segment>
- OS keeps mapping from segment name to its base physical address in Segment Table
- OS can (transparently) relocate or resize segments and share them between processes
- Each segment has its own memory protection attributes

Segment Table



Review of Segmentation

- Advantages:
  - Each address reference is checked for protection by hardware
  - Many different classes of data items can be assigned different levels of protection
  - Users can share access to a segment, with potentially different access rights
  - Users cannot access an unpermitted segment
- Disadvantages:
  - External fragmentation
  - Dynamic length of segments requires costly out-of-bounds check for generated physical addresses
  - Segment names are difficult to implement efficiently

Paging

- Program (i.e., virtual address space) is divided into equal-sized chunks (pages)
- Physical memory is divided into equal-sized chunks (frames)
- Frame size equals page size
- Virtual addresses consist of two parts:

  <page #, offset within page>

  - # bits for offset = $\log_2$(page size), no out-of-bounds possible for offset
- OS keeps mapping from page # to its base physical address in Page Table
- Each page has its own memory protection attributes

Review of Paging

- Advantages:
  - Each address reference is checked for protection by hardware
  - Users can share access to a page, with potentially different access rights
  - Users cannot access an unpermitted page
- Disadvantages:
  - Internal fragmentation
  - Assigning different levels of protection to different classes of data items not feasible

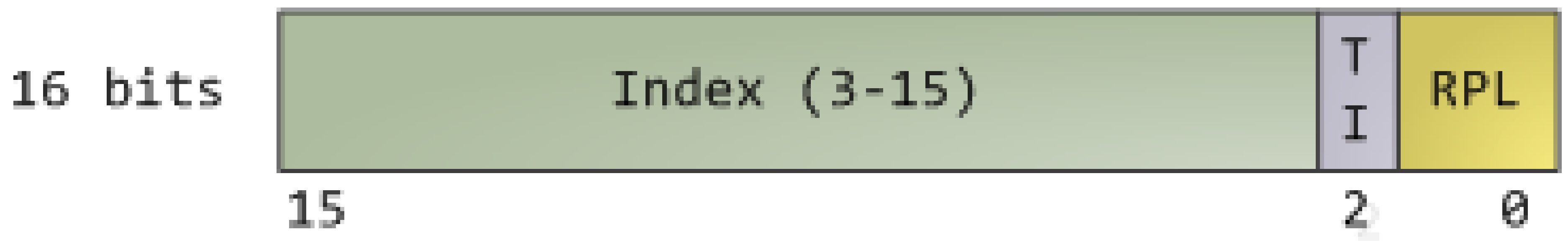
Physical vs. Virtual Memory

Physical memory	Virtual memory
No protection	Each process isolated from all others and from the OS
Limited size	Illusion of infinite memory
Sharing visible to processes	Each process cannot tell if memory is shared

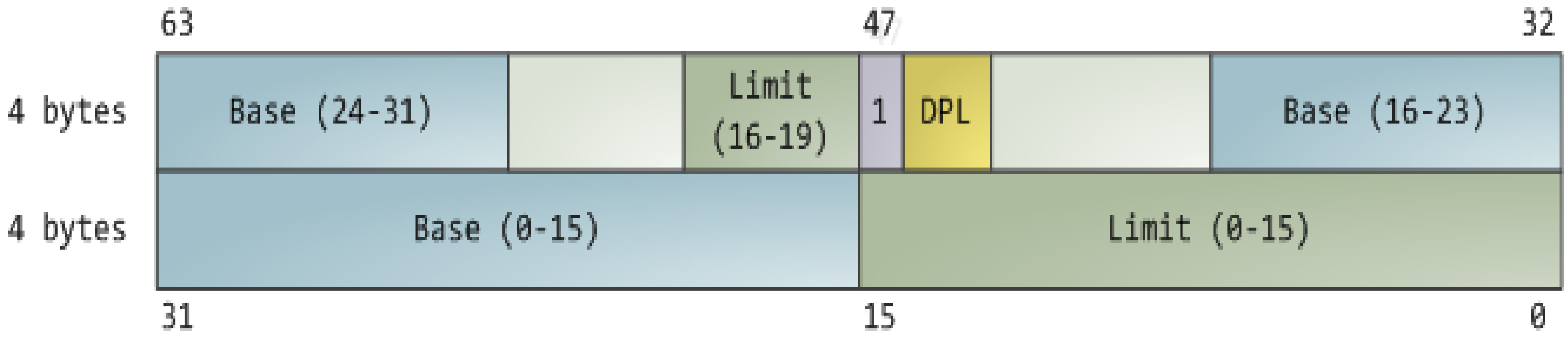
x86 Architecture

- x86 architecture provides both segmentation and paging
  - Linux uses a combination of segmentation and paging
    - Only simple form of segmentation to avoid portability issues
    - Segmentation cannot be turned off on x86
  - Same for Windows
- Memory protection bits indicate no access, read/write access or read-only access
- Recent x86 processors also include NX (No eXecute) bit, forbidding execution of instructions stored in page
  - Enabled in Windows XP SP 2 and some Linux distros
  - Helps against some buffer overflows

Segment Selector

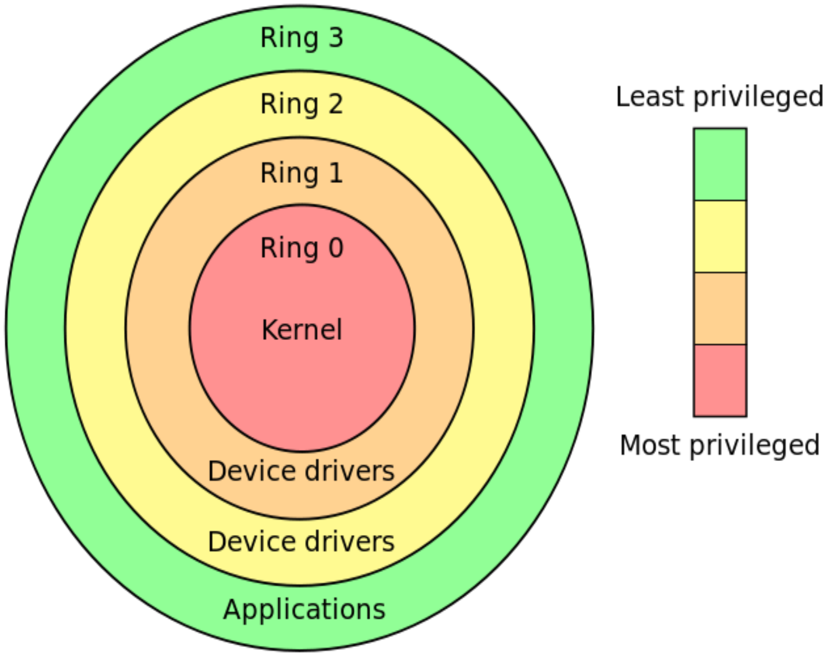


Segment Descriptor



Ring security model

Ring 0 – Kernel Level
Kernel provides the abstraction between Device Drivers and Applications software, and the firmware that the hardware uses and marking memory area as executable, writable are done by kernel, GDT table is maintained only by ring 0.



*The **Global Descriptor Table** or *GDT* is a data structure used by Intelx86-family processors starting with the 80286 in order to define the characteristics of the various memory areas used during program execution, including the base address, the size and access privileges like executability and writability.

Ring 1 and 2 (Device driver)

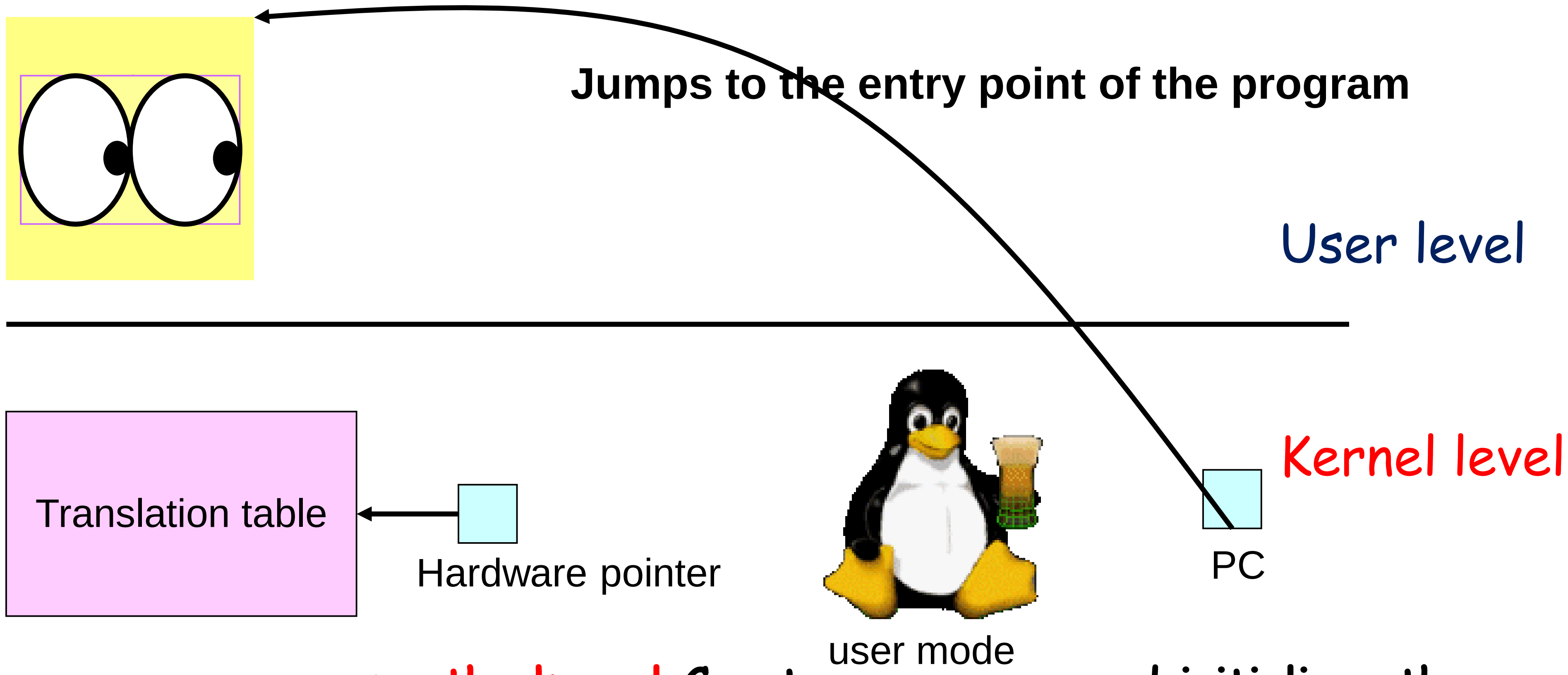
- Device driver needs lots of access to the hardware, and needs lots of freedom to access hardware, but too much freedom can crash system, so modification to GDT not provided to ring 1 and ring 2.
- It may happens that attacker find some vulnerability in driver code so using driver code he can mark any memory area as executable(if GDT modification permission given to ring 1,2).
- Virtual box /VM puts the guest kernel code in ring 1.
- In ring 2 there are driver like (Keyboard device driver, External device driver)

Dual-Mode Operation Revisited

- Translation tables offer protection if they cannot be altered by applications
- An application can only touch its address space under the user mode
- Hardware requires the CPU to be in the kernel mode to modify the address translation tables

- How the CPU is shared between the kernel and user processes
- How processes interact among themselves

Switching from the Kernel to User Mode



To run a user program, the kernel Creates a process and initializes the address space. Loads the program into the memory.

Initializes translation tables

Sets the hardware pointer to the translation table.

Sets the CPU to user mode

Switching from User Mode to Kernel Mode

- Voluntary
  - *System calls*: a user process asks the OS to do something on the process's behalf
- Involuntary
  - Hardware interrupts (e.g., I/O)
  - Program exceptions (e.g., segmentation fault)
- For all cases, hardware atomically performs the following steps
  - Sets the CPU to kernel mode
  - Saves the current program counter
  - Jumps to the handler in the kernel
    - The handler saves old register values

Switching from User Mode to Kernel Mode

- Unlike context switching among threads, to switch among processes
  - Need to save and restore pointers to translation tables
- To resume process execution
  - Kernel reloads old register values
  - Sets CPU to user mode
  - Jumps to the old program counter

- Thanks