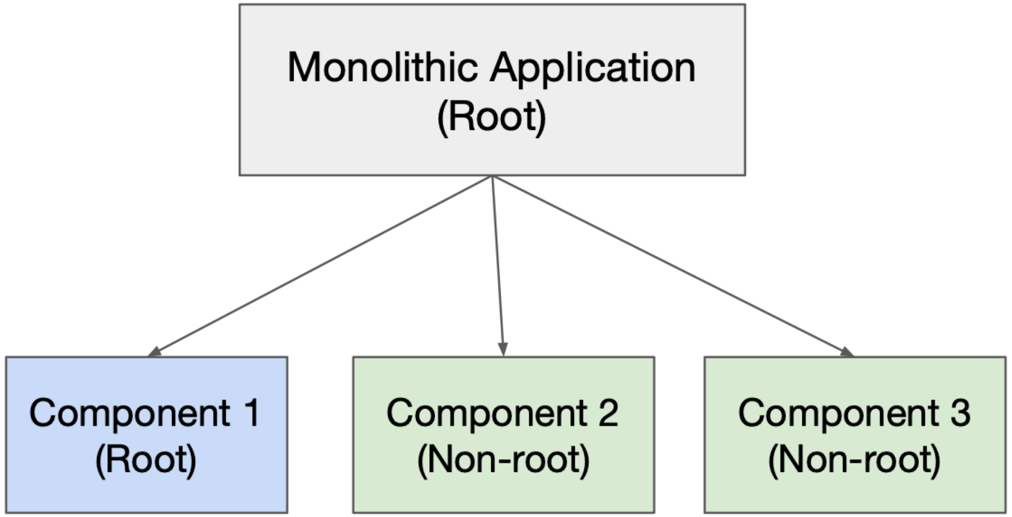
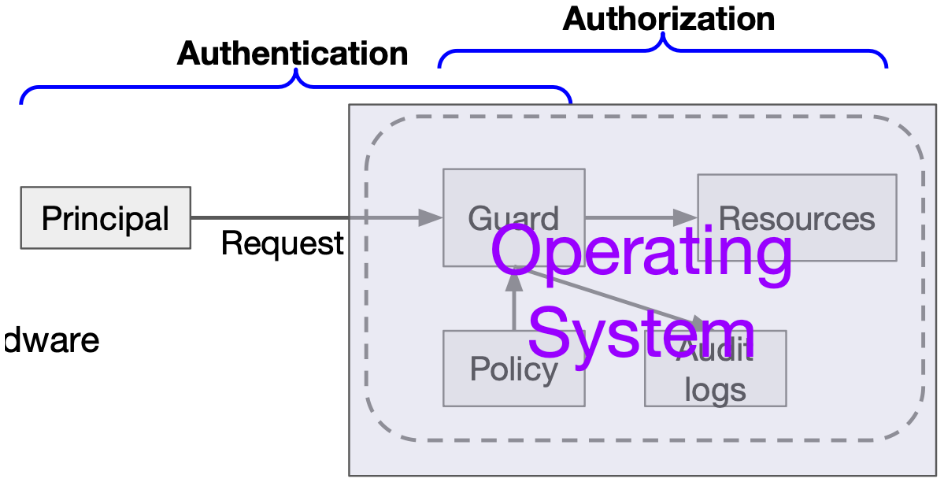
Isolation

Dileepa Fernando

Story (Privilege Separation)

- Modularize Monolithic Applications
  - Different folders?
  - *chroot*?
- For each module
  - Escalate privilege (setuid-binary, sudo, ..)
    - Sensitive operations
  - Drop privilege (setuid(u))
    - After sensitive operations
  - Least privilege principle
- Assumptions
  - We know what privilege to assign
  - OS enforces



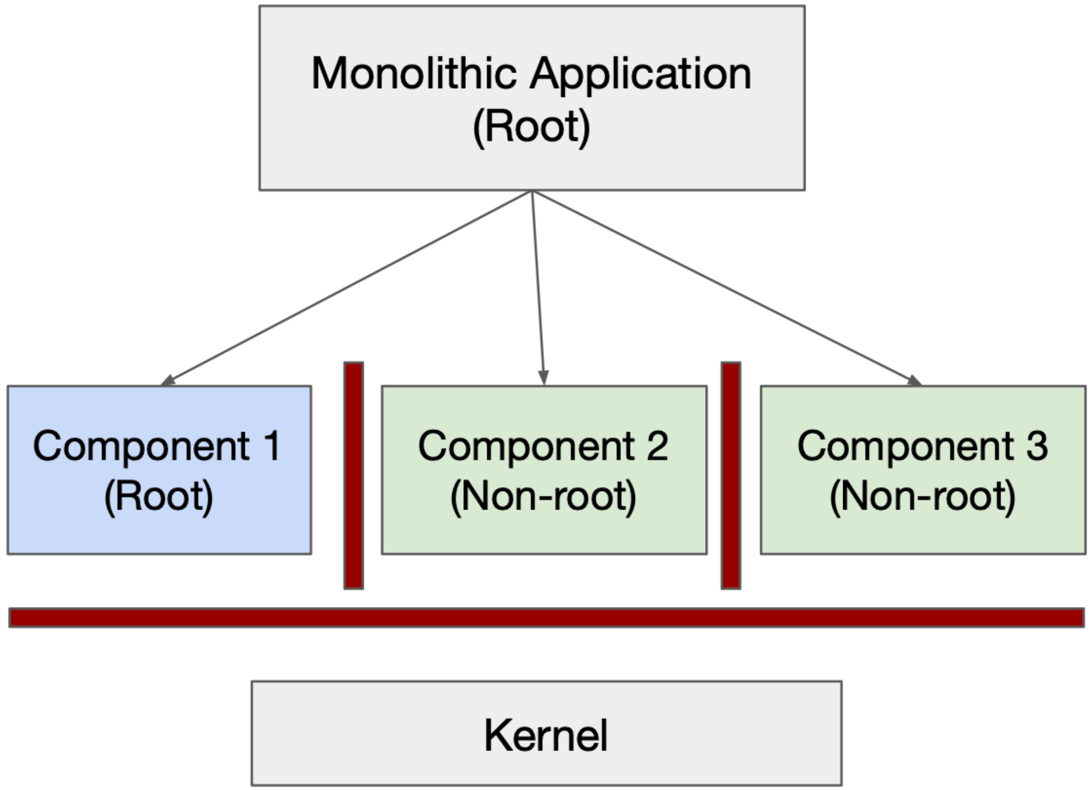


Isolation Needed for Priv. Separation

- The OS isolated
  - The processes

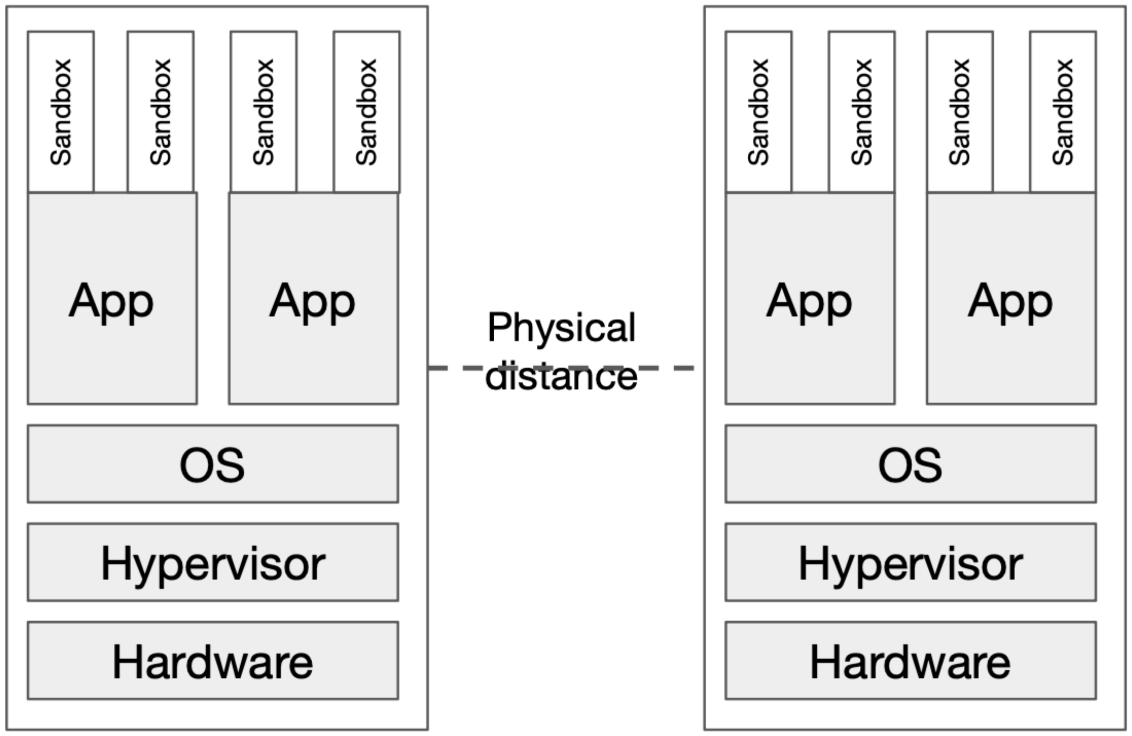
- A cannot affect B directly
- Errors are contained

- 100% Isolation is not desirable
- Controlled communication needed



Isolation (General)

- Lower Level Enforces
- TCB is the lower level
- Example:
  - App separation?
  - OS separation?
  - Hypervisor separation?

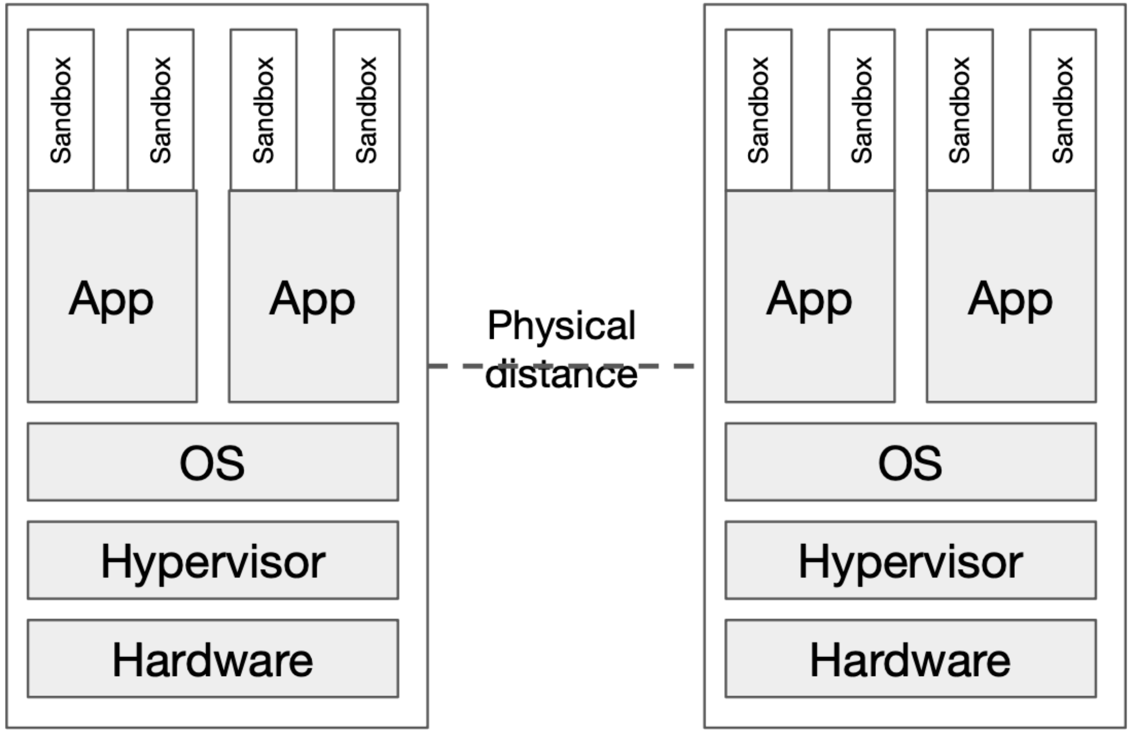


Isolation (General)

- Different Isolation levels

- Different security requirements
  - Keep Attackers out
  - Keep attackers in

- Different performance requirements
  - Container vs VM vs in app

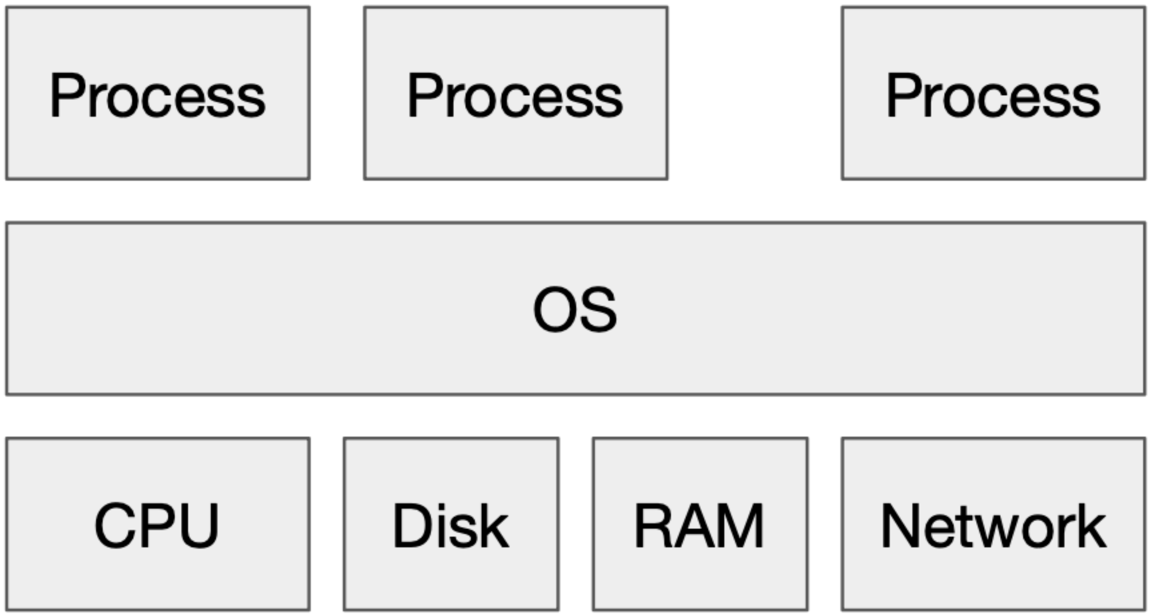


Overview

- Introduction
- Kernel Isolation
- [Application Isolation – Virtual Machine]
- [Application Isolation - Container]

Kernel Isolation

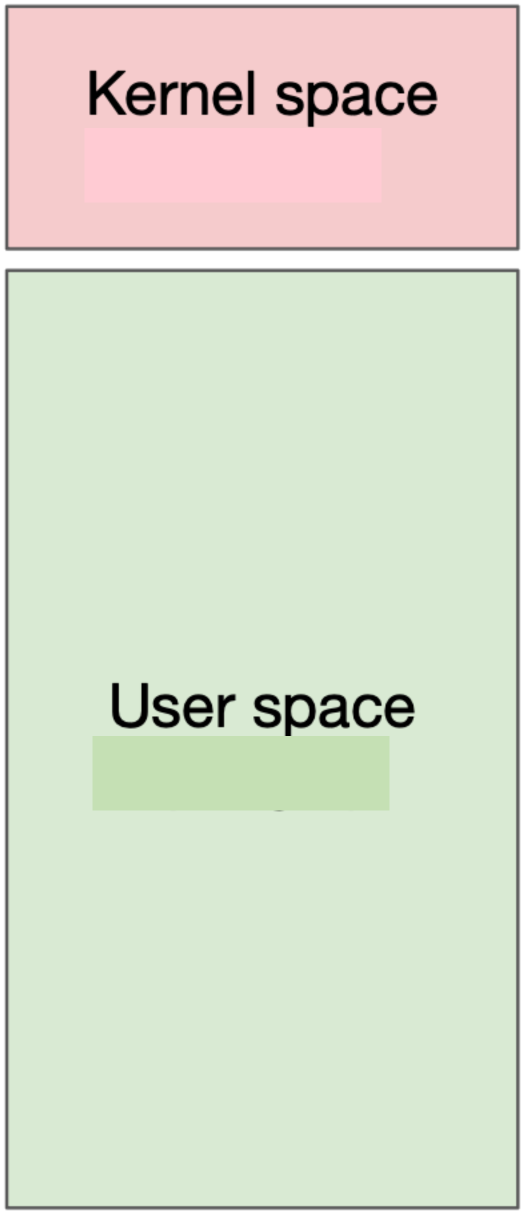
- Role
  - Manage resources
  - Abstract resources
- Security Requirements
  - Isolate itself
  - Isolate processes among each other
- TCB
  - CPU + Kernel Code
- Threat Model
  - Applications
  - Administrator
  - Hardware vendor?
  - Physical Tampering?



Kernel Isolating Itself

- Memory Isolation (Virtual)
  - In 32 bit, 1GB dedicated for kernel
  - User process cannot access kernel space
  Directly

- Instruction Isolation
  - CPU runs in different privilege levels
  - Instructions are restricted at some levels
  - Not to be confused with **process privilege**
    - But idea is similar



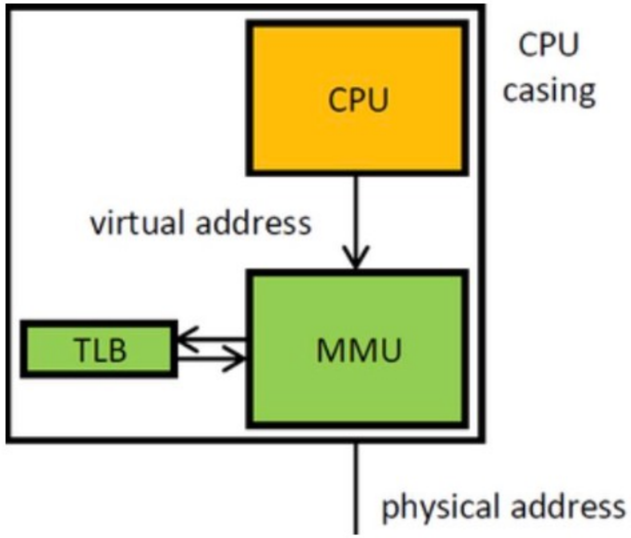
0xffffffff

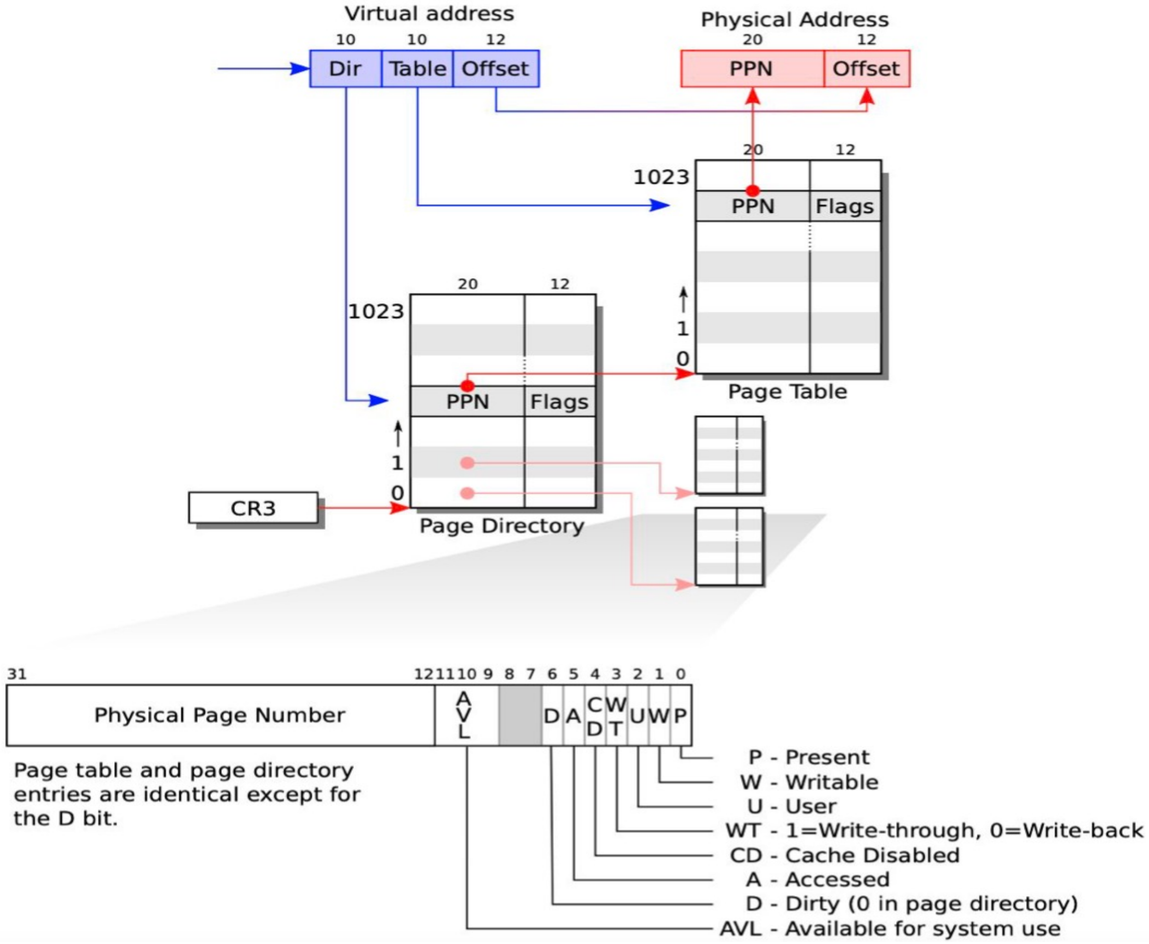
Process Layout

0x00000000

Kernel Isolating Itself

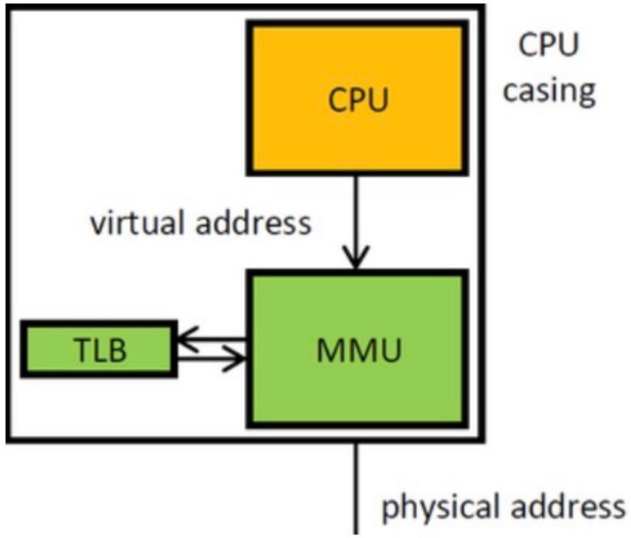
- **Memory Isolation**
  - Virtual -> Physical Translation
    - Page tables reside in kernel
    - Referred from CPU (CR3 register)
  - Cannot write directly to physical mem.

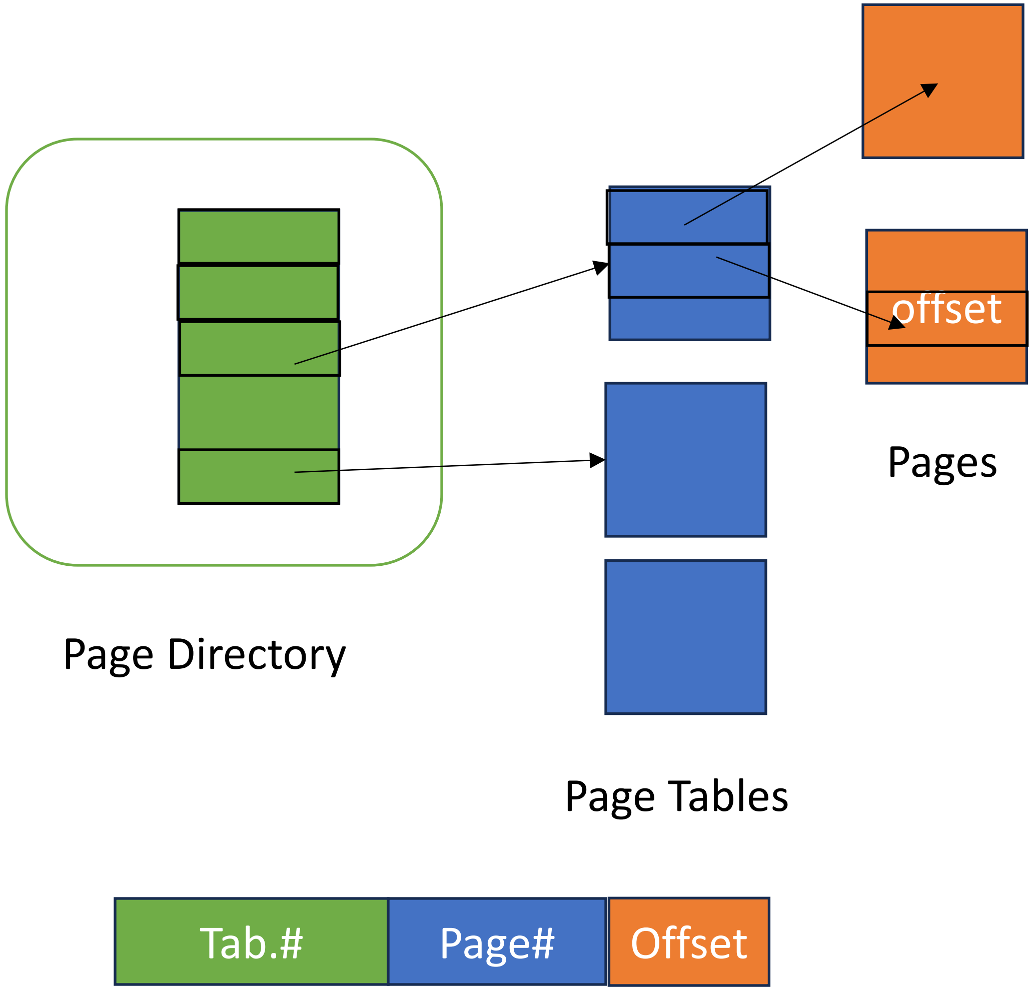




Kernel Isolating Itself

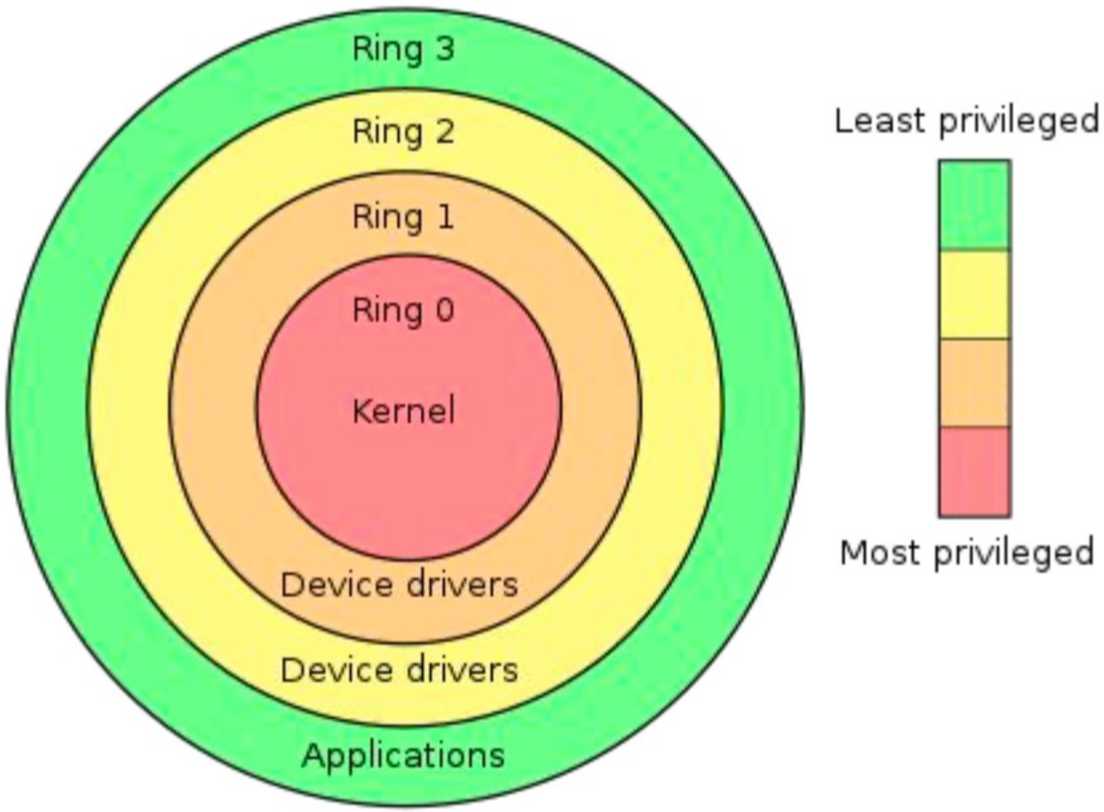
- **Memory Isolation**
  - Virtual -> Physical Translation
  - Happens inside CPU
  - User never knows





Kernel Isolating Itself

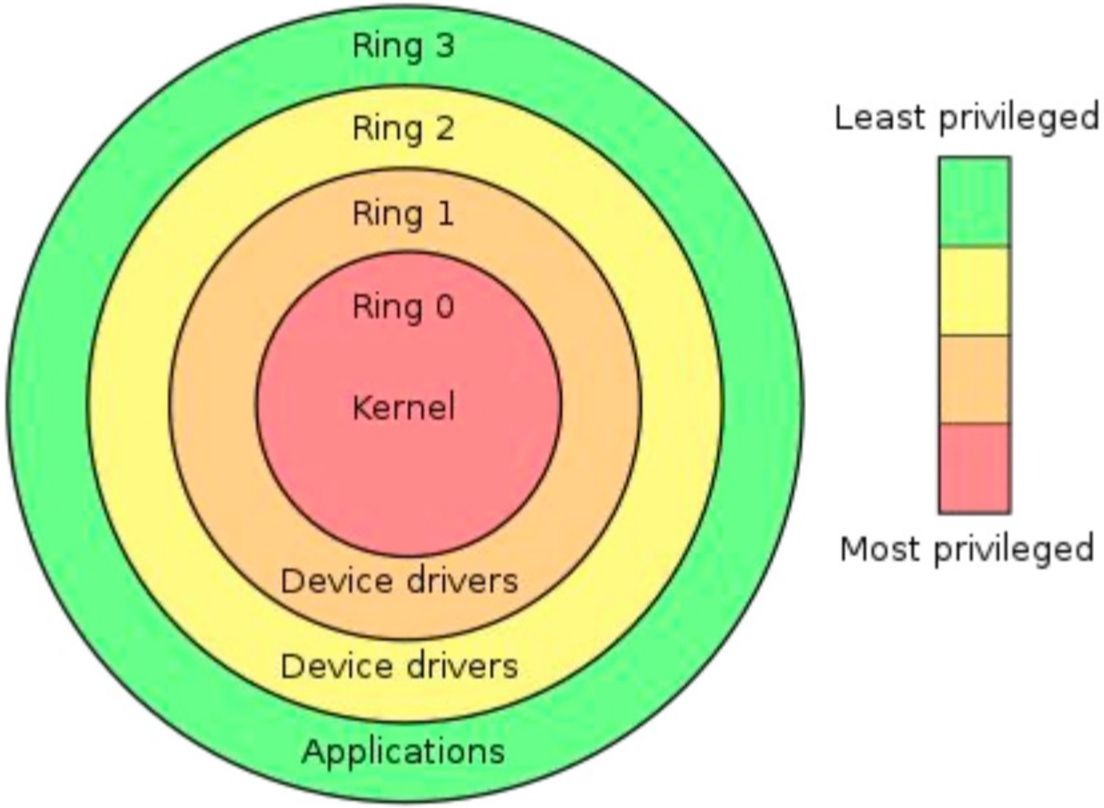
- **Instruction Isolation**
  - CPU executes instructions according to privilege levels (Ring)
  - Ring 0 (Highest Privilege)
    - Any instruction executed
  - Ring 3 (Lowest Privilege)
    - Any user process (including uid 0)
    - **Root user can attack user resources**
      - **But not the OS**



Kernel Isolating Itself

- **Instruction Isolation**
  - Saved in Code Segment Register
  - *info reg*
  - CPL checked for each instruction
    - What is CPL?

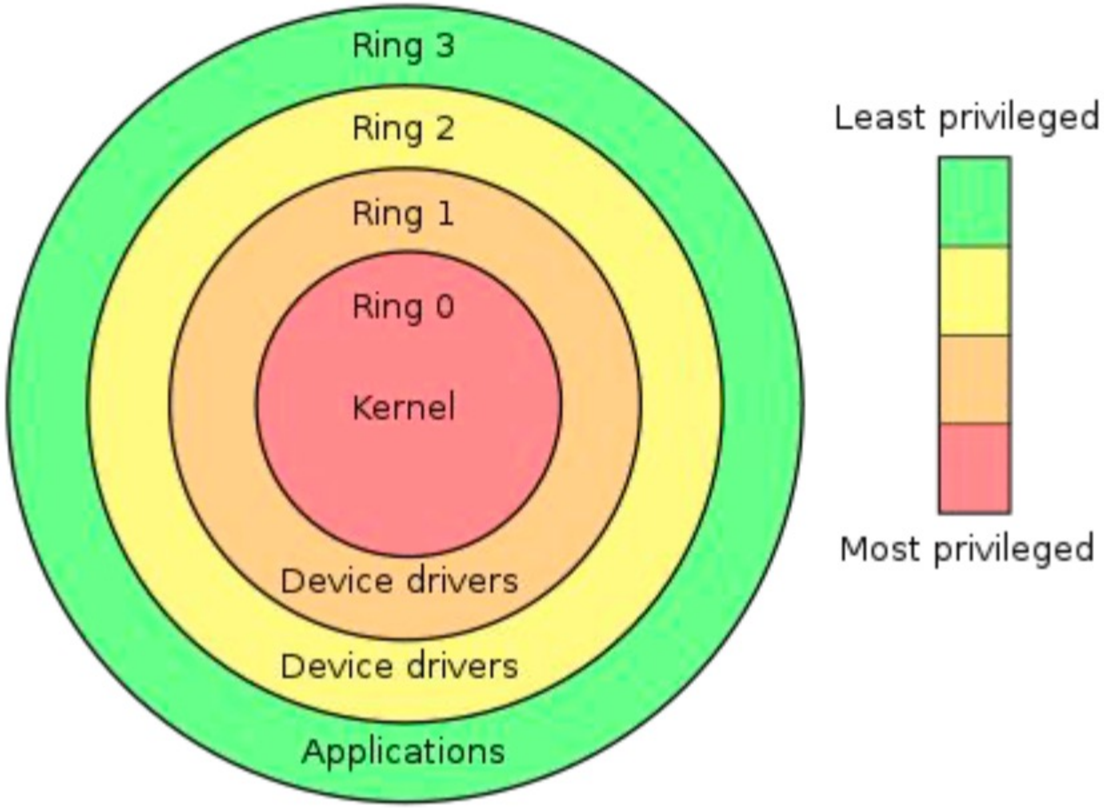
CPL = 0: ring 0
CPL = 3: ring 3



Kernel Isolating Itself

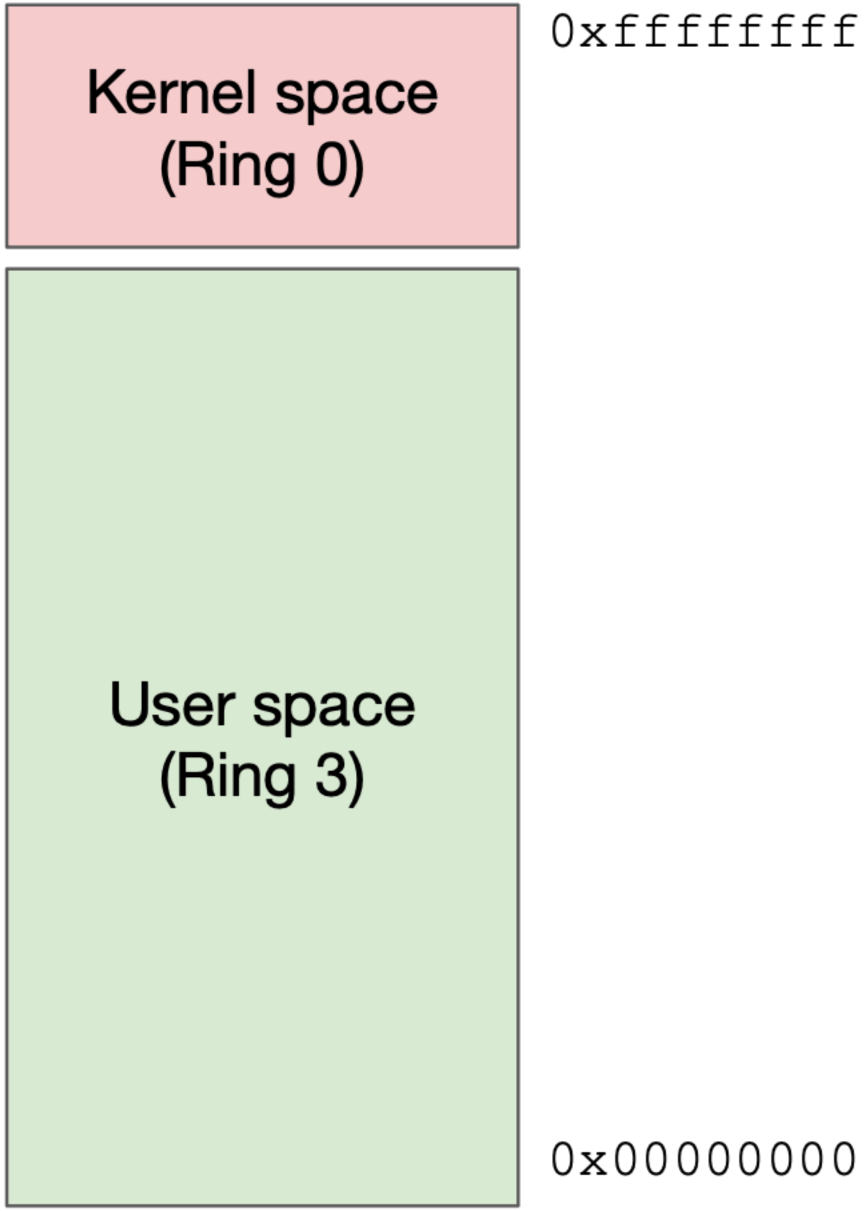
- **Instruction Isolation**
  - Ring 0 at boot
  - Only changes with interrupts
    - System calls
    - Hardware interrupts

CPL = 0: ring 0
CPL = 3: ring 3



Kernel Isolating Itself

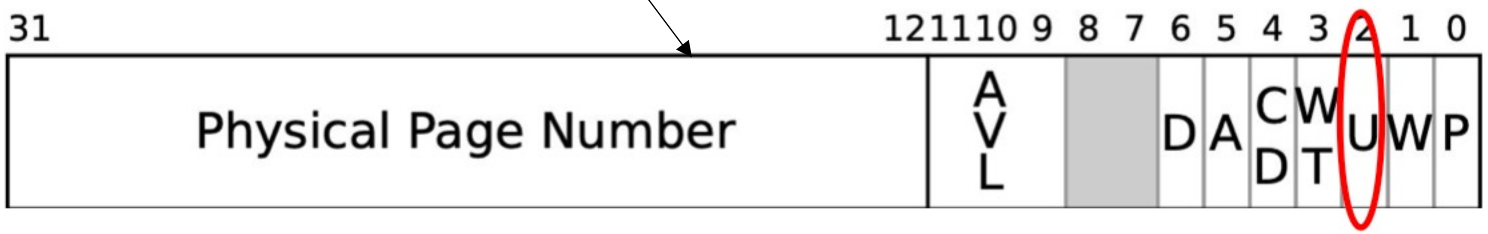
- General Isolation Idea
  - Kernel/ User Space separation
  - User space process -> Ring 0
    - CPL=0
  - Kernel space process -> Ring 3
    - CPL=3
  - Data Isolation
    - Kernel Space data access prohibited for Ring 3
      - MOV EAX [0xffffffff]
  - Code Isolation
    - Kernel instructions are prohibited for Ring 3
      - CLI

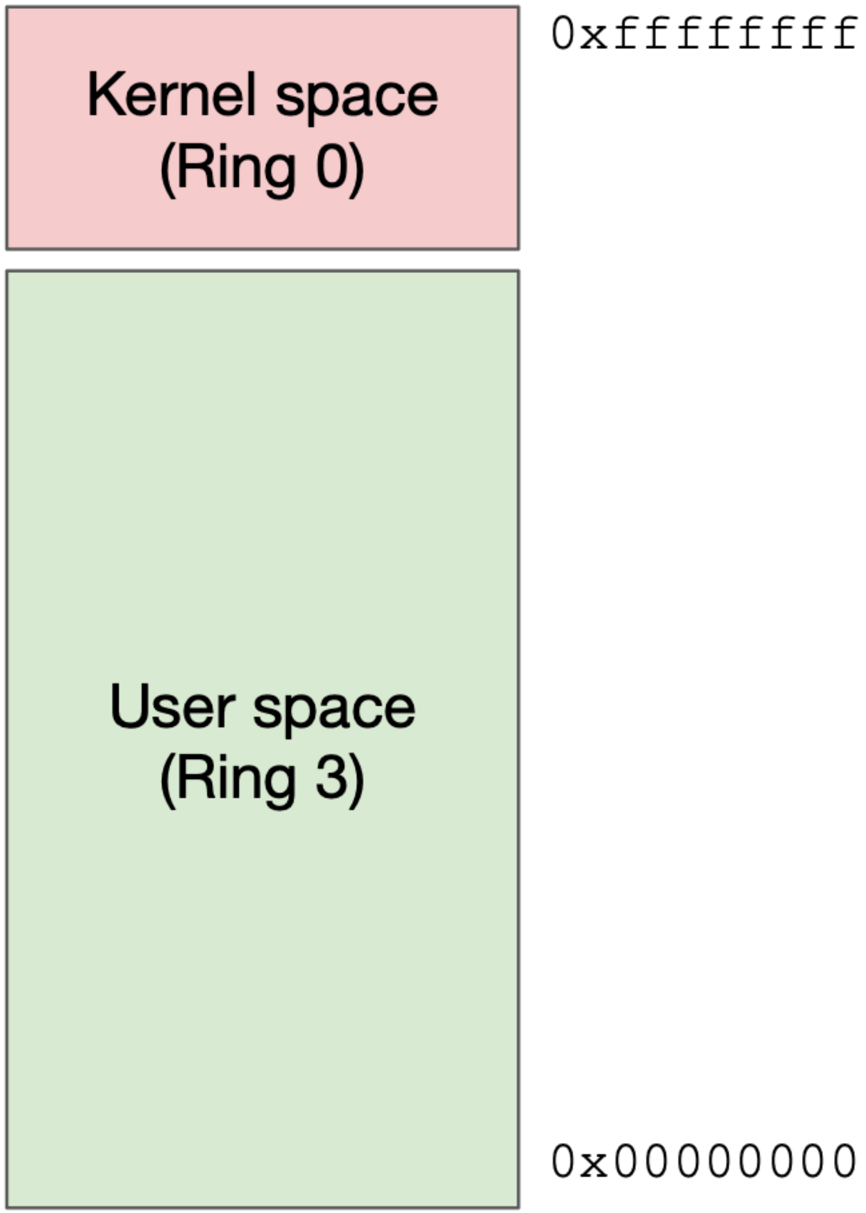


Kernel Isolating Itself

- Data Isolation
  - Read/Write kernel space not allowed for Ring 3
  - Can we jump to a kernel address?
    - JMP ADDR
  - Can we overwrite page table directly?
  - Can we point CR3 to somewhere else?

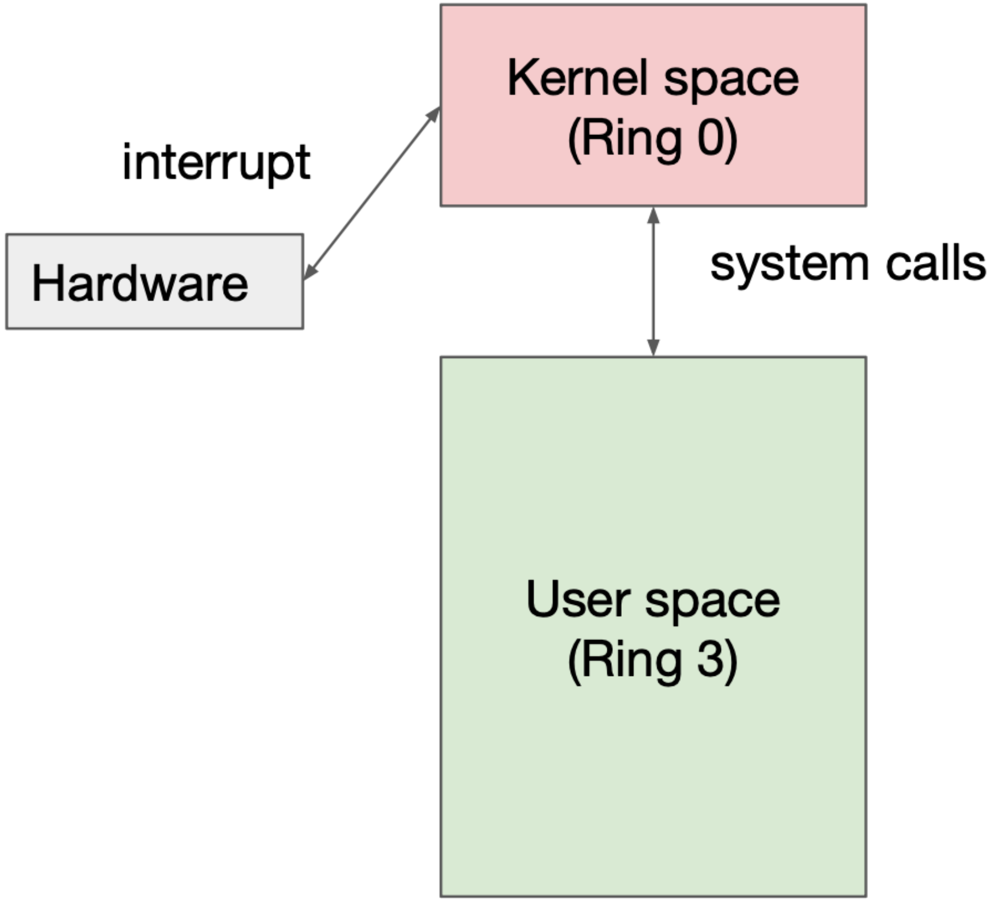
Page Table Entry (U Flag)





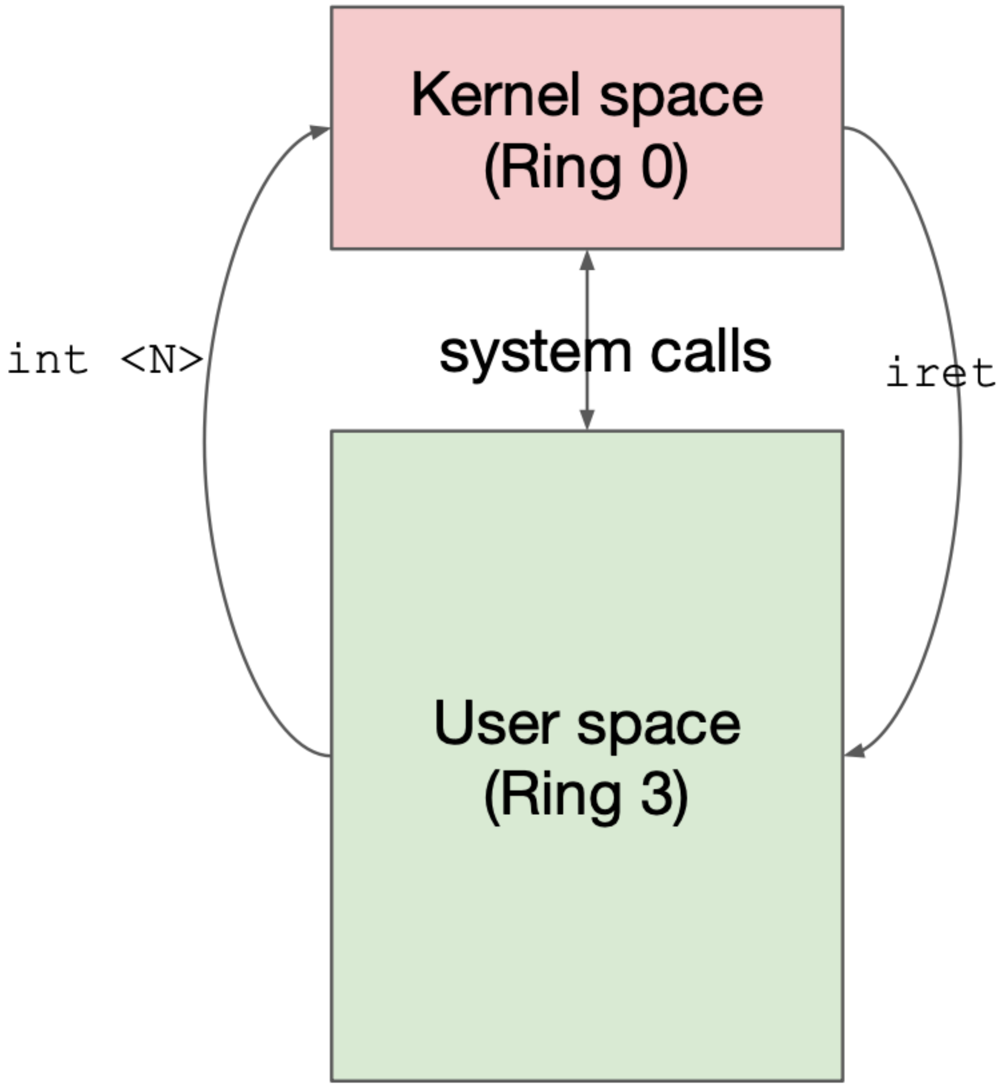
Kernel Isolating Itself

- We cannot jump to a kernel address
- But a user need to access kernel space
- Need safe transition
  - Well defined exit/entry/behavior
  - Safe change of privilege level
  - **Interrupt**
    - Stop normal execution
    - Transition to kernel
  - Software – System Calls
  - Hardware – Timer, Faults



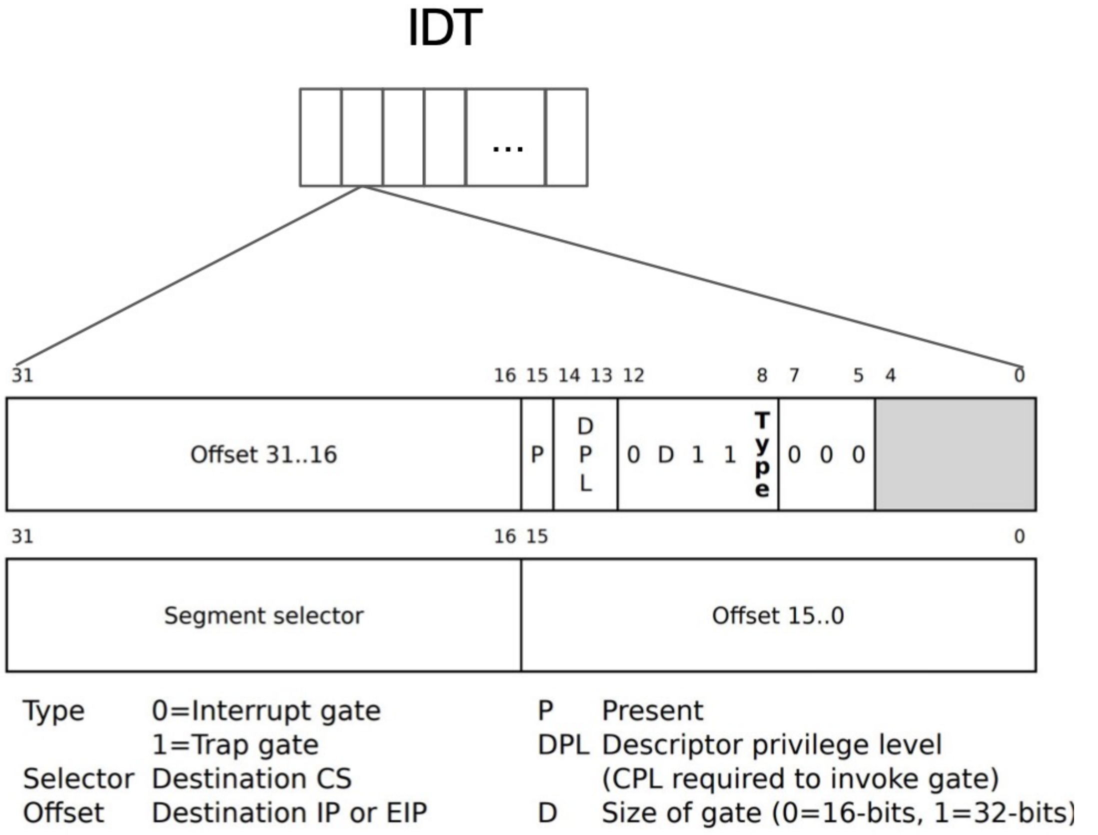
Kernel Isolating Itself

- We cannot jump to a kernel address
- But a user need to access kernel space
- Need safe transition
  - INT N command
    - Control to IDT[N] (kernel space)
    - SET CPL
  - IRET
    - SET CPL
    - Return to user space
  - Security Principle?



Kernel Isolating Itself

- Interrupt Descriptor Table
  - 256 Entries
  - Each Entry 64 bytes
  - Address located in %idtr reg
- When INT N received (Detail)
  - Control to IDT[N]
  - Check CPL<=DPL
  - SET CS, EIP
  - Interrupt Service Routine
  - IRET
    - SET CS, EIP
    - Return



Kernel Isolating Itself

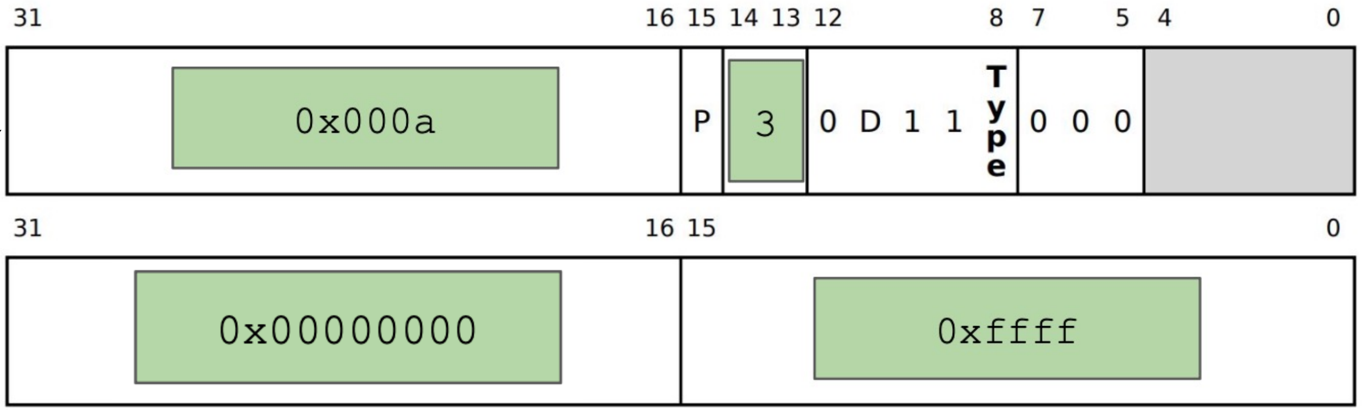
- Question (Comprehension)
  - If SET CPL=0, JMP <addr> is allowed
  - What can happen?
  - Setting CPL in kernel space

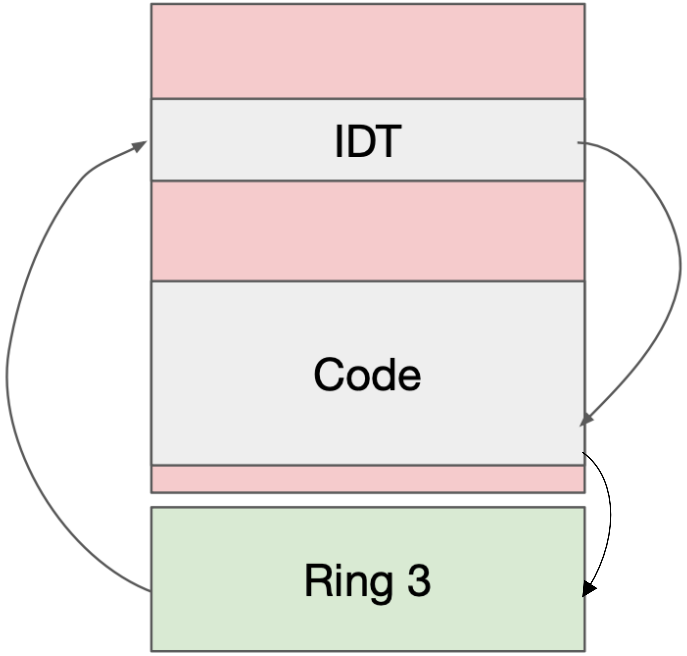
  - If JMP <a kernel addr> SET CPL=0 is allowed
  - What can happen?
  - Not all kernel addresses should be jumped to

  - Changing Privilege level only in Kernel space
  - Only jumping to well defined Kernel functions

Kernel Isolating Itself

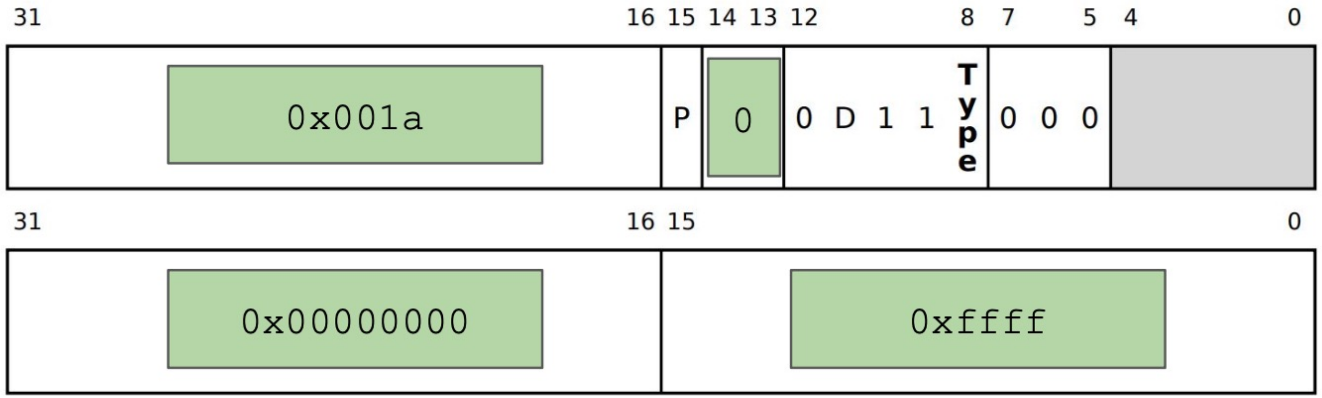
- Example (System Call)
  - INT 0x80
    - IDT entry
    - DPL 3
    - Address of ISR code
      - 0xffff000a
    - Final value of CPL
      - 0x00000000

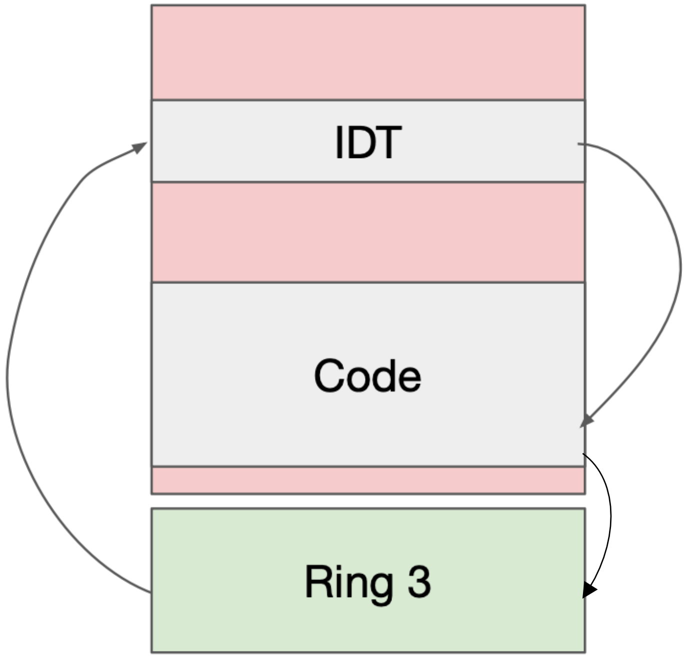




Kernel Isolating Itself

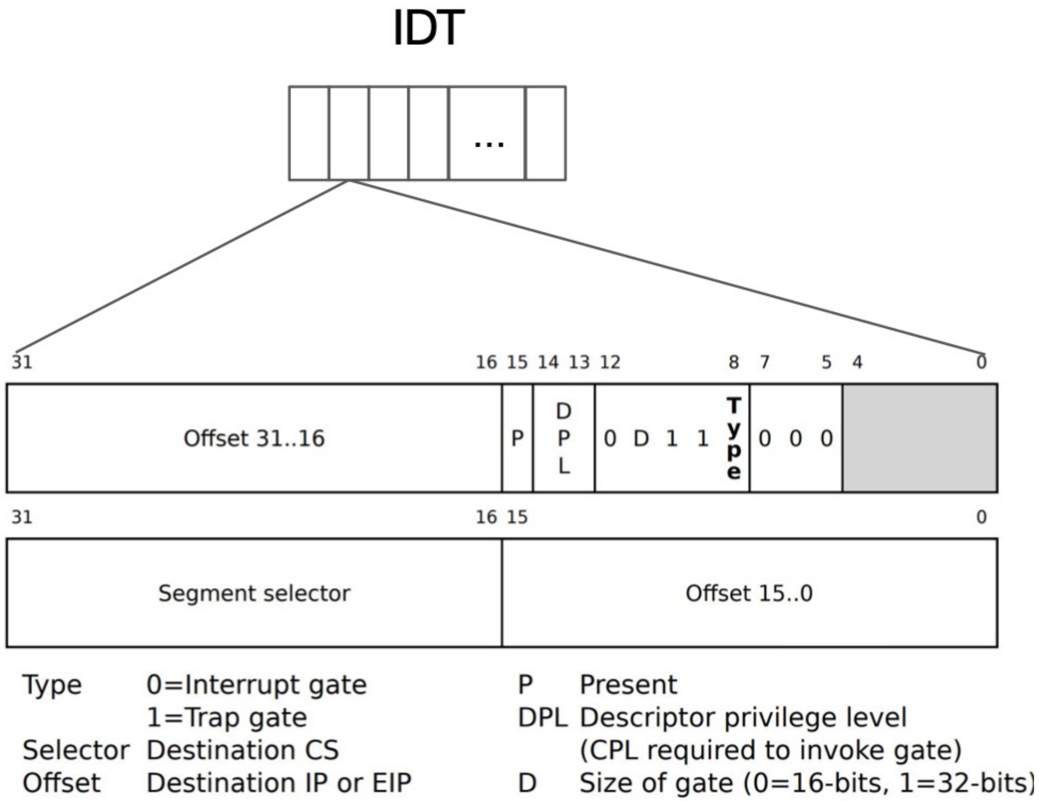
- Example (Page Fault)
    - INT 0xe
        - IDT entry
        - DPL 0
        - Address of ISR code
            - 0xffff001a
        - Final value of CPL
            - 0x00000000





Kernel Isolating Itself

- Question
  - How does hardware stop you if you called INT 0xe?



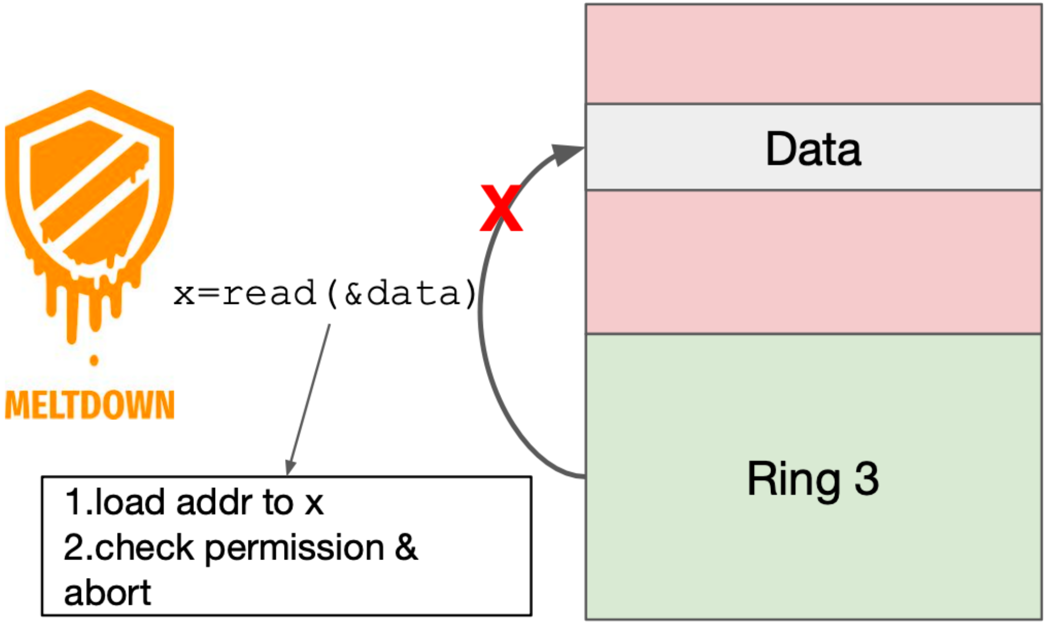
Kernel Isolating Itself

- Summary
  - Data Isolation
    - Virtually Separate kernel and user space
    - Physical separation with U flag
    - User cannot directly access kernel space (MOV, SET, JMP)
  - Instruction Isolation
    - Any instruction in kernel space
    - Restricted instructions in user space
      - SET CR3, SET CS etc. restricted
  - Safe transition (Controlled Interaction)
    - INT 0x80 from user space
    - INT 0xe from hardware
    - SET CPL only in kernel space

Kernel Isolating Itself

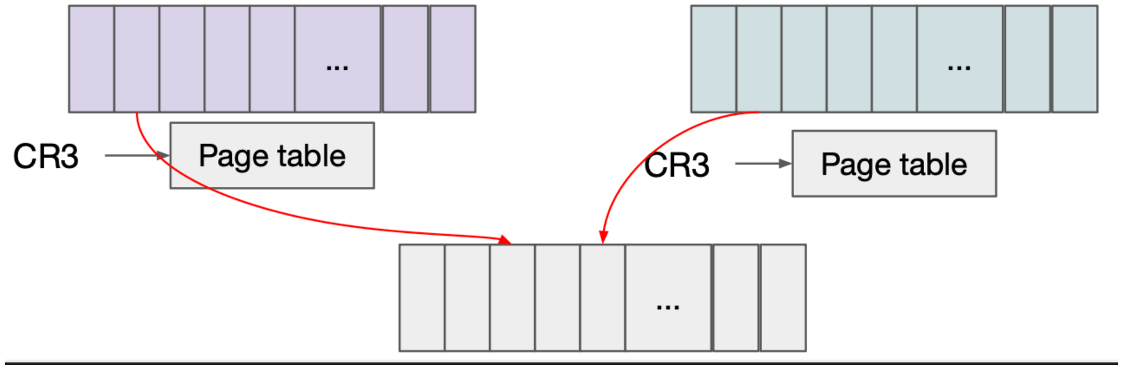
- Can we afford to write sloppy kernel code?

- Meltdown attack
  - Does this work?
  - Works when CPU speculates
    - Cache before check CPL/ U flag



Isolation Among Processes (Separation)

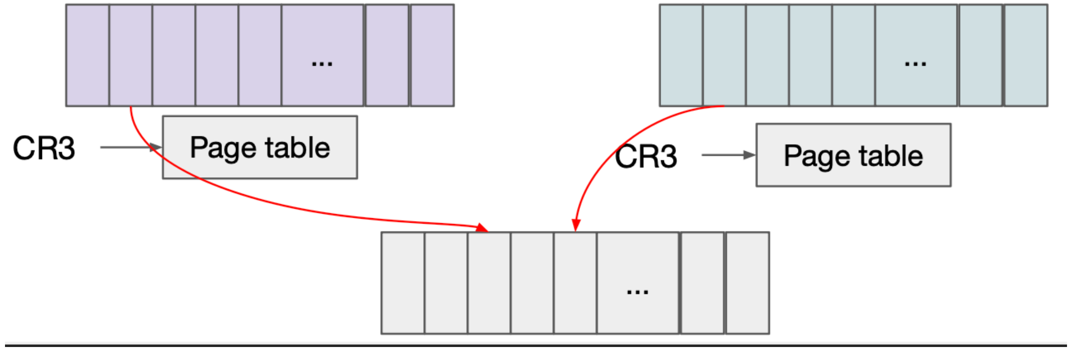
- Each Process has their own Virtual Address Space
  - May overlap some times
    - 0xdeadbeef in Proc. A
    - 0xdeadbeef in Proc. B



- Their Physical Address spaces do not overlap
  - 0xdeadbeed in Proc. A and Proc. B map to different physical addresses
  - Except for explicit shared memory
  - **Enforced by Kernel**

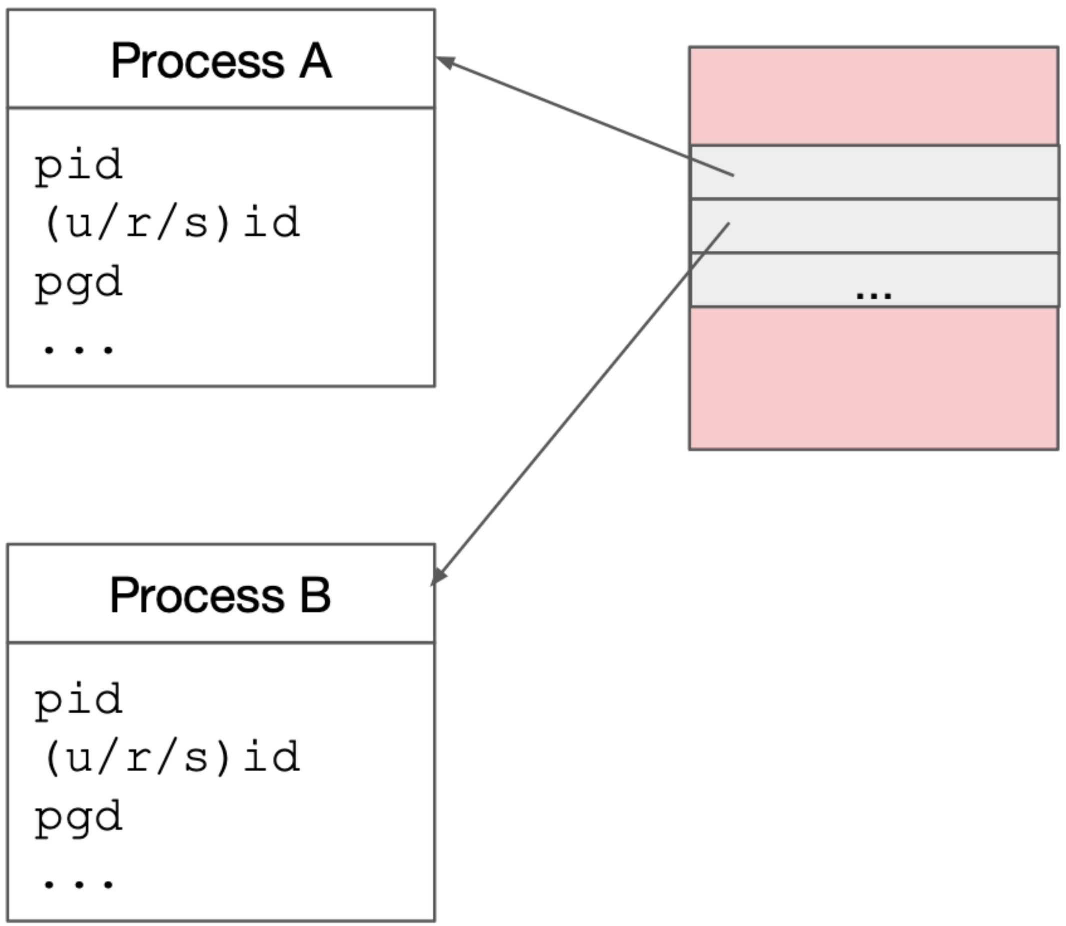
Isolation Among Processes (Enforcement)

- Can Proc. A access page table of Proc. B?
  - When Proc. A running
  - CR3 is page directory of A
  - It has to be changed
    - To access a different page table
    - CR3 cannot be changed from user space
  - Can we have our own page table?



Isolation Among Processes

- Kernel Ensures no overlap
  - Ex: For Malloc
- During Context Switch
  - Kernel takes control
  - SET CR3 to proc->pgd
  - Flush TLB

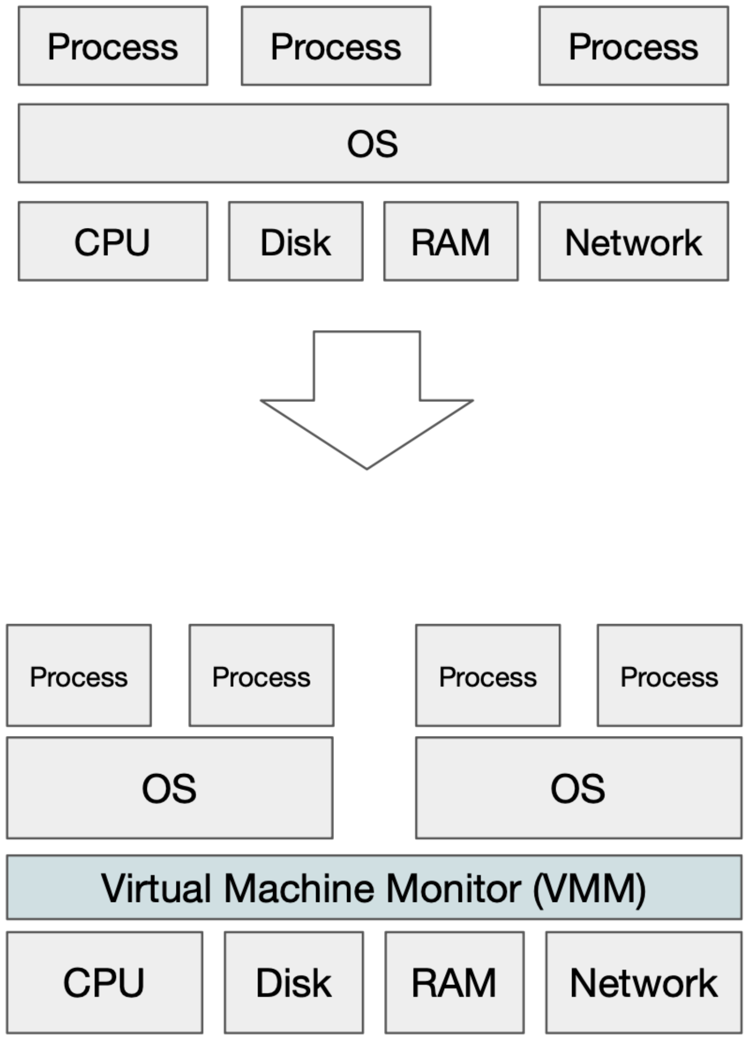


Kernel Isolation

- Summary
  - CPU privilege levels and safe transitions
  - User/ Kernel space separation (Virtual Memory)
  - Separation between Processes (Virtual Memory)
  - Enforcement by Kernel and CPU together
    - Kernel, CPU vulnerabilities lead to attacks

Application Isolation – Virtual Machine

- Old Idea – Popek & Goldberg 1974
    - Host and Guest concept
        - Real Hw (Host Hw)
        - **Simulated** Hw (VMM, hypervisor)
        - First Sw contact with **simulated** Hw (Guest OS)
    - We focus on VMM directly running on Hw
    - Why do we call VMM the hypervisor?



Application Isolation – Virtual Machine

- Simulation
  - Assume you write a program (Virtual Machine)
    - Accepts a binary as input (Ex: ./demo)
    - Keep data structures for CPU, Memory, etc.
    - Update the data structures according to binary instructions

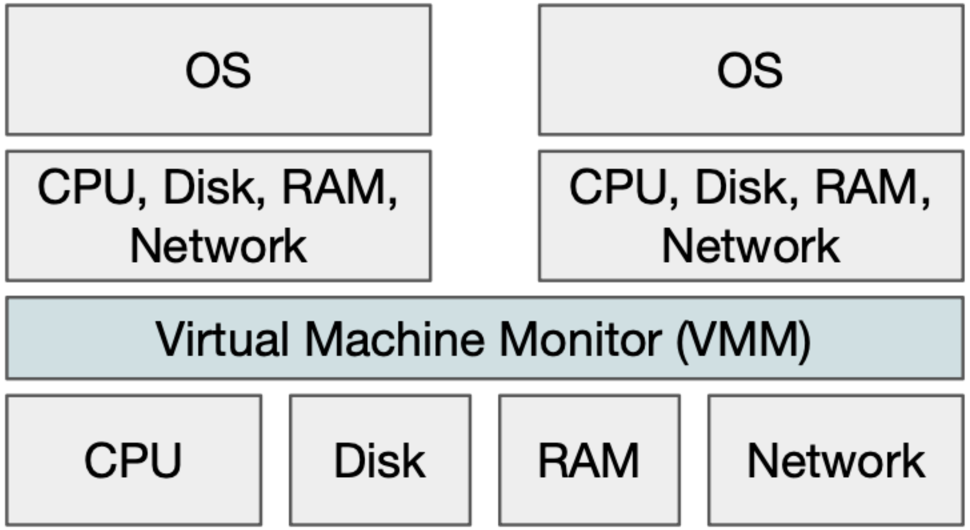
```
while ( ip<code.length ) {
    int opcode = code[ip]; // fetch
    if ( trace ) System.err.printf("%-35s", disInstr());
    ip++;
    switch (opcode) {
        case ICONST:
            int v = code[ip];
            ip++;
            sp++;
            stack[sp] = v;
            break;
        case PRINT:
            v = stack[sp];
            sp--;
            System.out.println(v);
            break;
        case GLOAD :
            int addr = code[ip];
            ip++;
```

- Virtual Machine Monitor
  - Manages/ Runs virtual machines
  - Efficiently distribute Hw resources

Application Isolation – Virtual Machine

- To VMM
  - Guest OS is a user program
  - Guest OS has its own virtual memory space

- To guest OS
  - It sees the Hw simulation as real
  - Virtual Hw executes guest OS

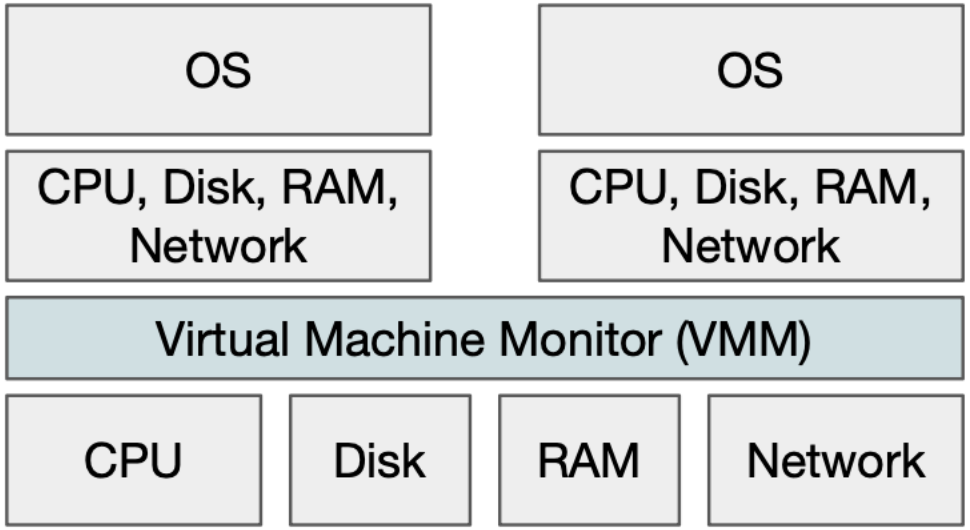


Application Isolation – Virtual Machine

- The Virtual Machine Monitor

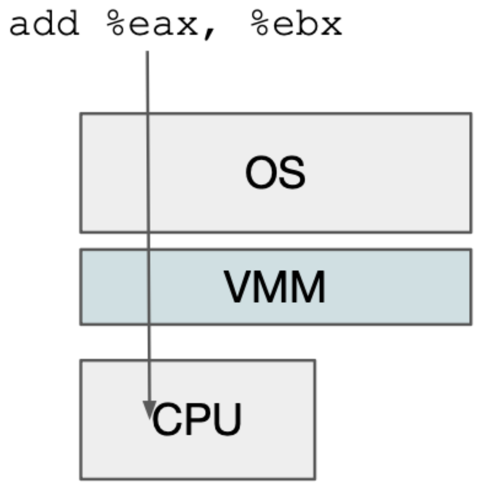
- Requirement 1: Protection
  - VMM protects itself
  - Guest OS kept inside simulation
- Requirement 2: Illusion
  - Guest OS must not realize the simulation
  - No check inside guest OS should reveal this
- Requirement 3: Performance

- Two more security principles
  - Defense in-depth
  - Small TCB

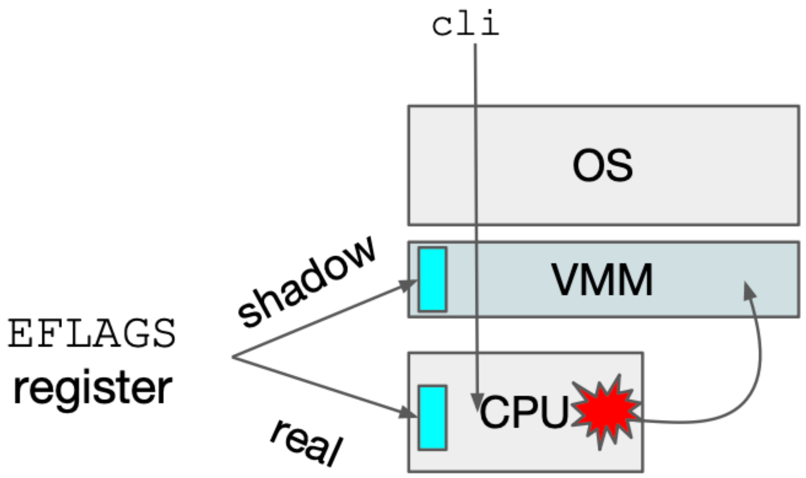


Application Isolation – Virtual Machine

- Approach (Trap and Emulate)
  - VMM runs in Ring 0
  - Guest OS/ processes run in Ring 3
  - CPU executes Guest Instruction
    - ADD, XOR, PUSH etc.

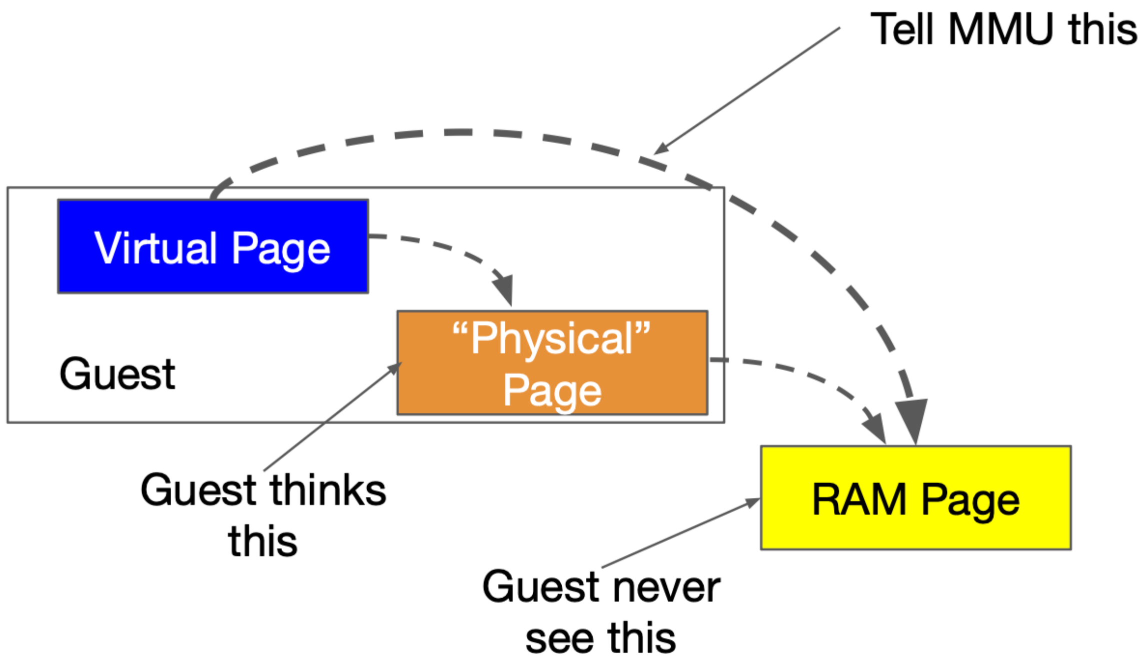


  - Privileged Instructions
    - CLI, LCR3
    - CPU raises exceptions
    - Handled by VMM



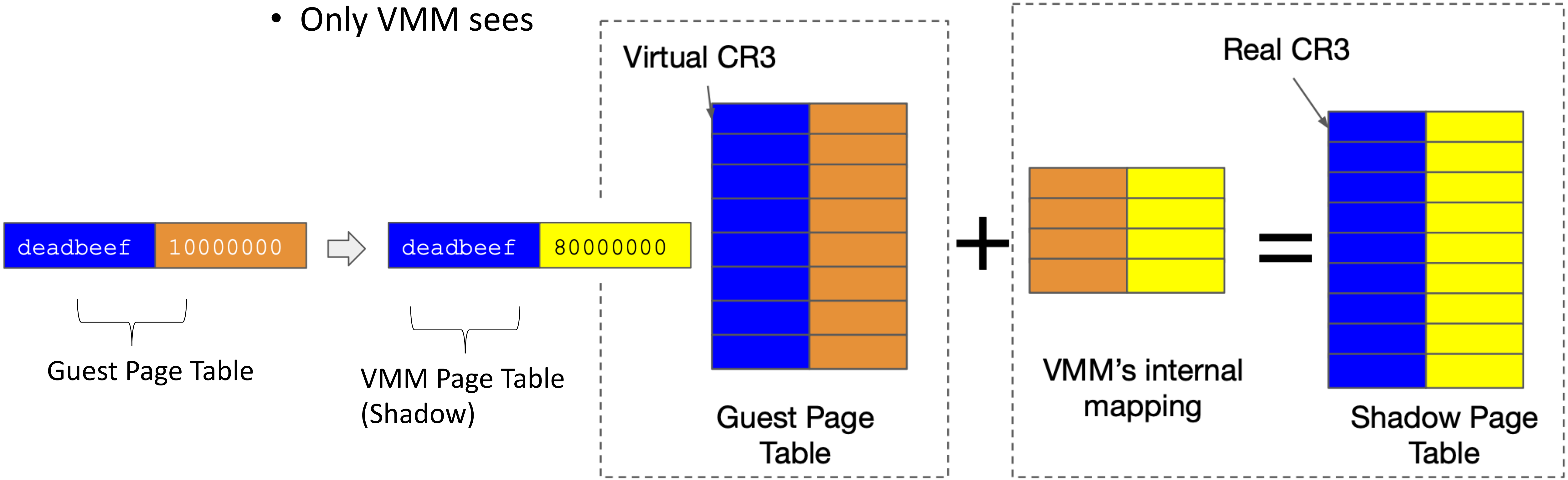
Application Isolation – Virtual Machine

- Approach (Trap and Emulate) – Virtualize Memory
  - Guest
    - Guest Virtual
    - Guest Physical
  - Host
    - Guest Physical
    - Host Physical
  - Modern Implementations
    - Guest Virtual
    - Host Physical
    - Shadow page table



Application Isolation – Virtual Machine

- Approach (Trap and Emulate)
  - Shadow Page Table
    - One per Guest Process
    - Only VMM sees



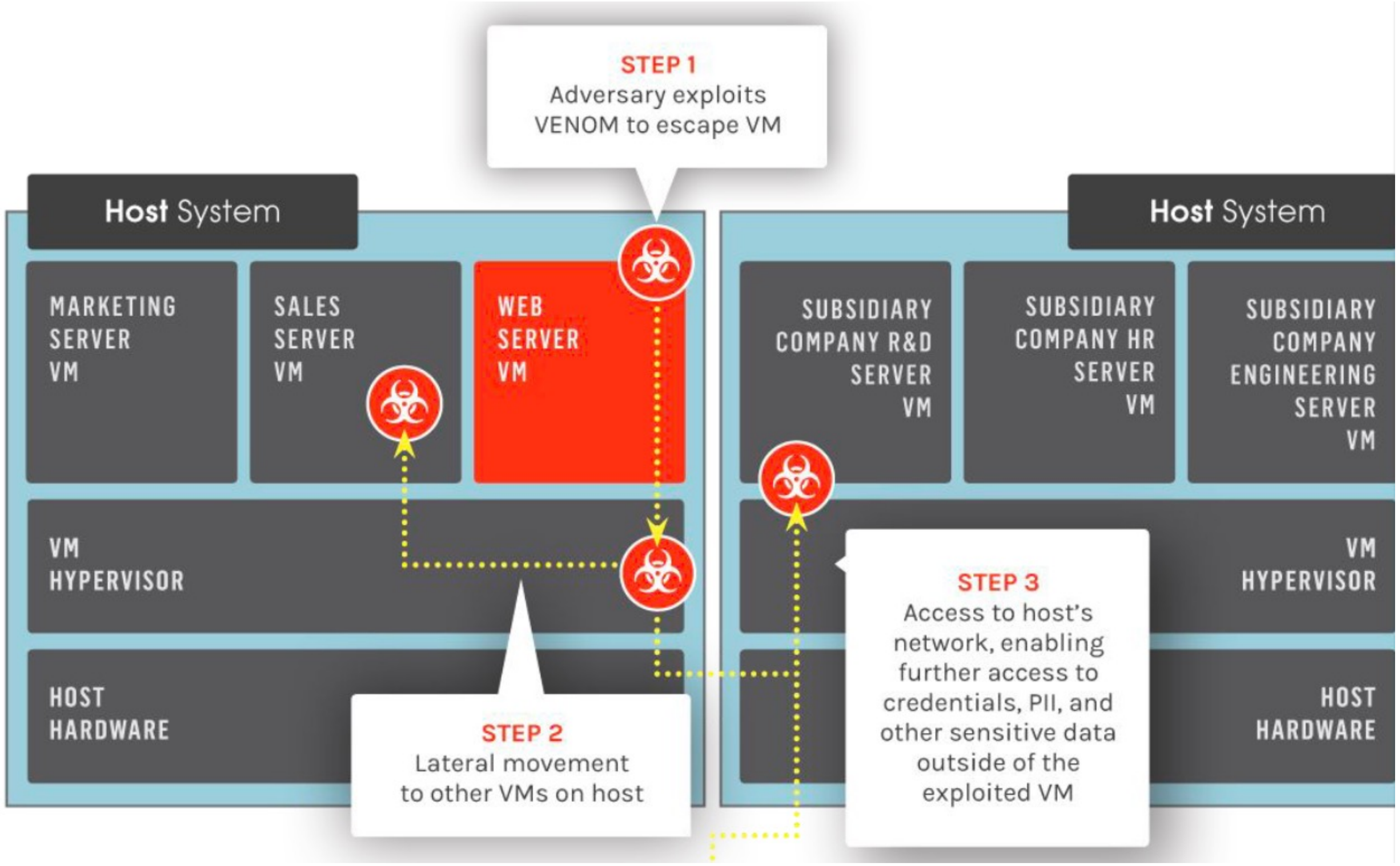
Application Isolation – Virtual Machine

- Does Trap and Emulate Meet Requirements?
  - Protection, Illusion, Performance
- Alternative 1: Binary Translation (VM Ware)
  - Translate Troublesome instructions to safer ones
  - MOV CS EAX -> INT XX
- Alternative 2: Paravirtualization (Xen)
  - VM knows it is in simulation
  - Guest OS is modified to communicate with VMM
- Alternative 3: Hardware Support (Intel-VTX, AMD-SVM)
  - Privilege handling
  - Memory/IO management

Application Isolation – Virtual Machine

- Summary
  - Simulation good for Isolation
  - Another Abstraction Layer
  - Fast Enough?

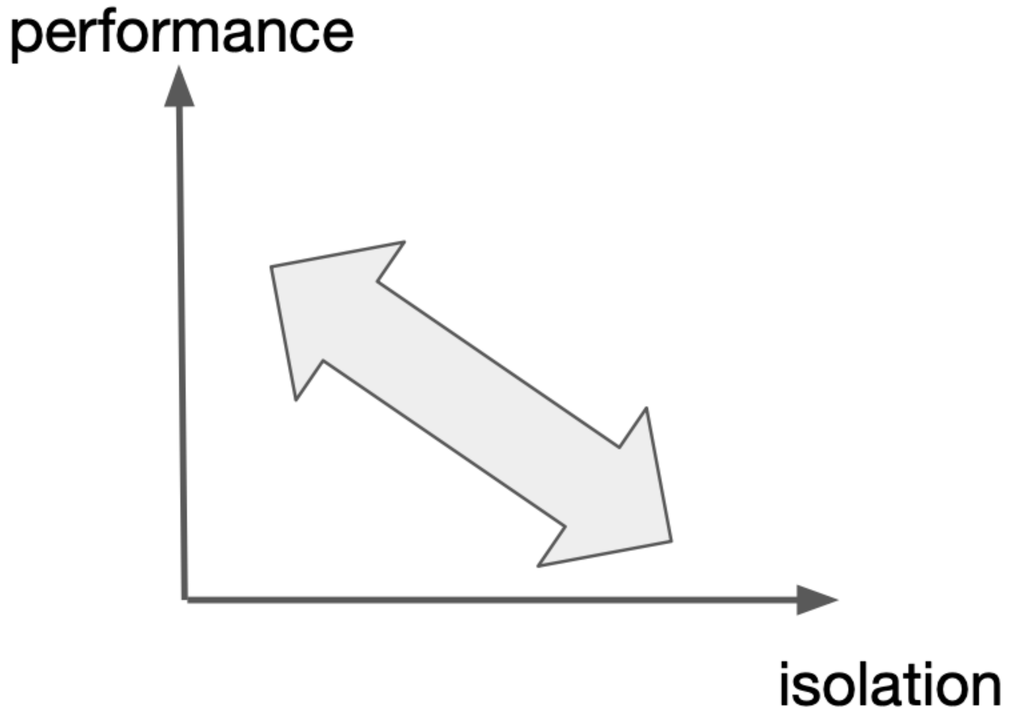
- Attack?
  - Buffer Overflow in VMM
    - VENOM



Application Isolation – Containers

- Isolation So Far
- TCB – Hw, Kernel
- Virtualized Hardware
- Performance, Isolation Trade-off

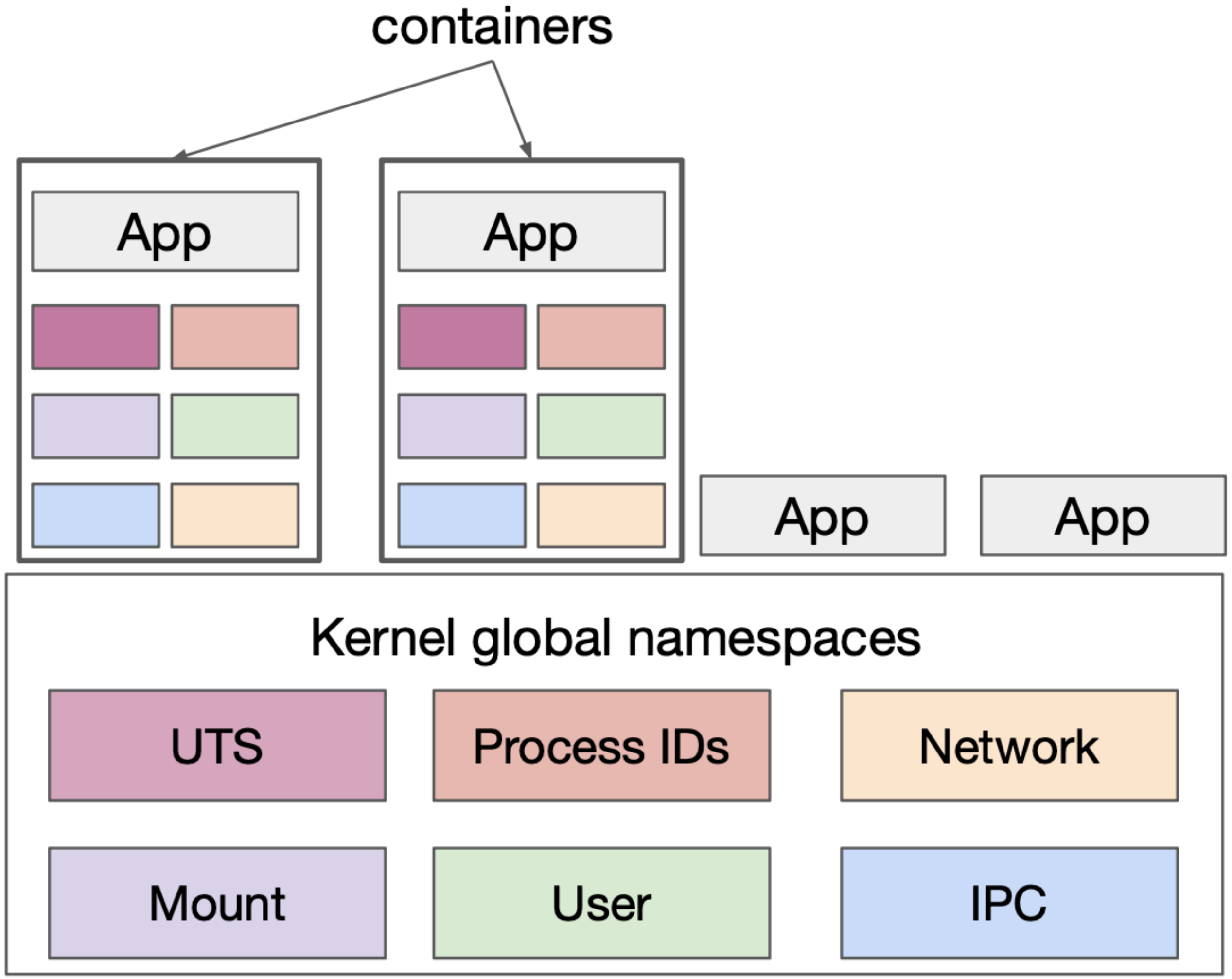
- Container – Somewhere in the middle
- Virtualize OS



Application Isolation - Containers

- Idea1: Virtualize and Isolate Kernel Resources
- Idea2: Limit Resources for each container
- Idea3: Limit Kernel Access for each container (System Call Filter)

- All available in linux distribution

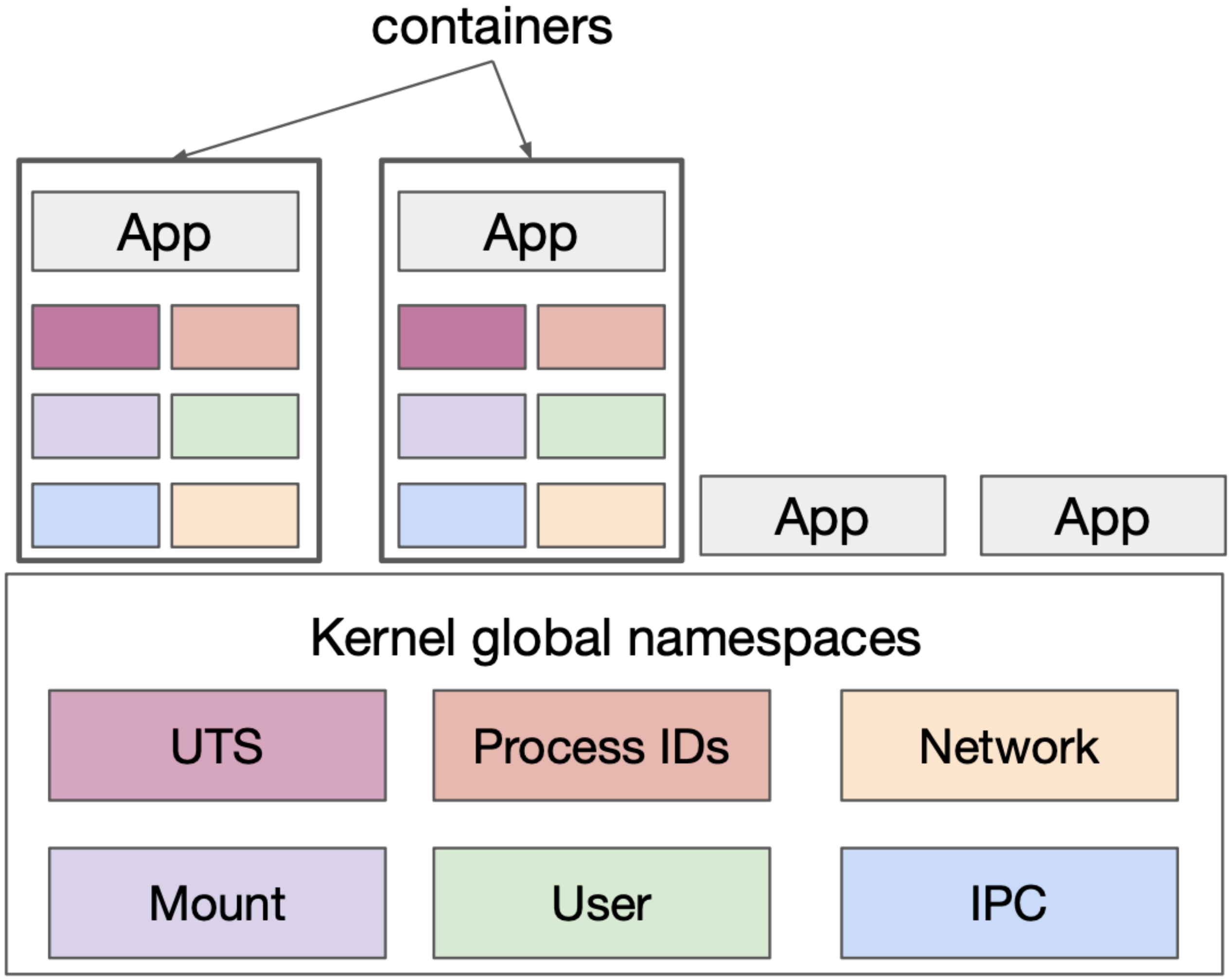


Application Isolation - Containers

- **Idea1: Virtualize and Isolate Kernel Resources**

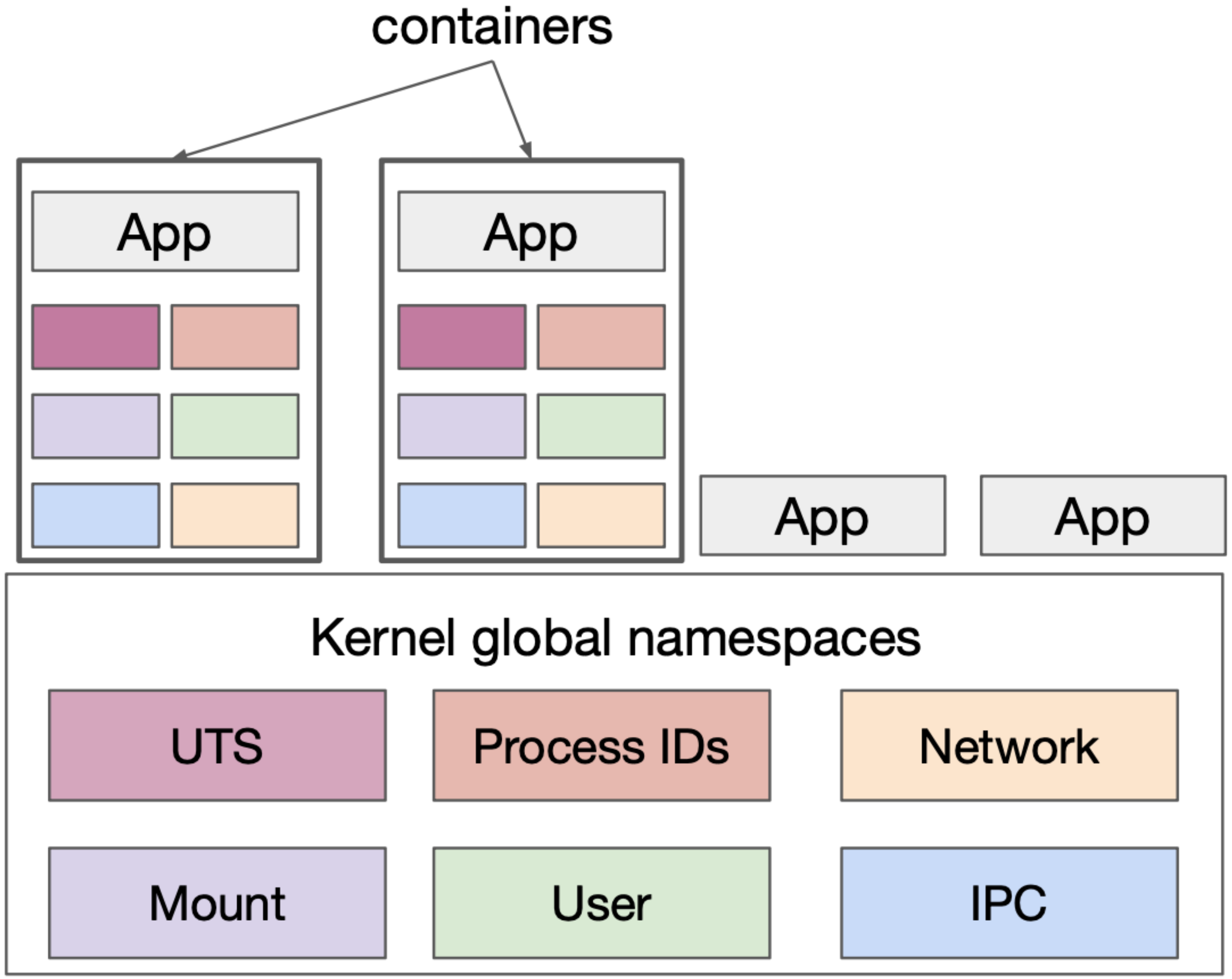
- Resources: Name Spaces
  - Process A in name space X != Process A in name space Y
  - *man namespace*

- Analogies
  - Path in Chroot
  - Address A in two virtual mem. spaces



Application Isolation - Containers

- **Idea2: Limit Resources for each container (CPU, RAM, IO)**
- cgroups
  - Set Resource limit to a group of processes
  - *man cgroups*
- namespaces + cgroups
  - Start a new process in namespace
  - Add it and children to a new cgroup

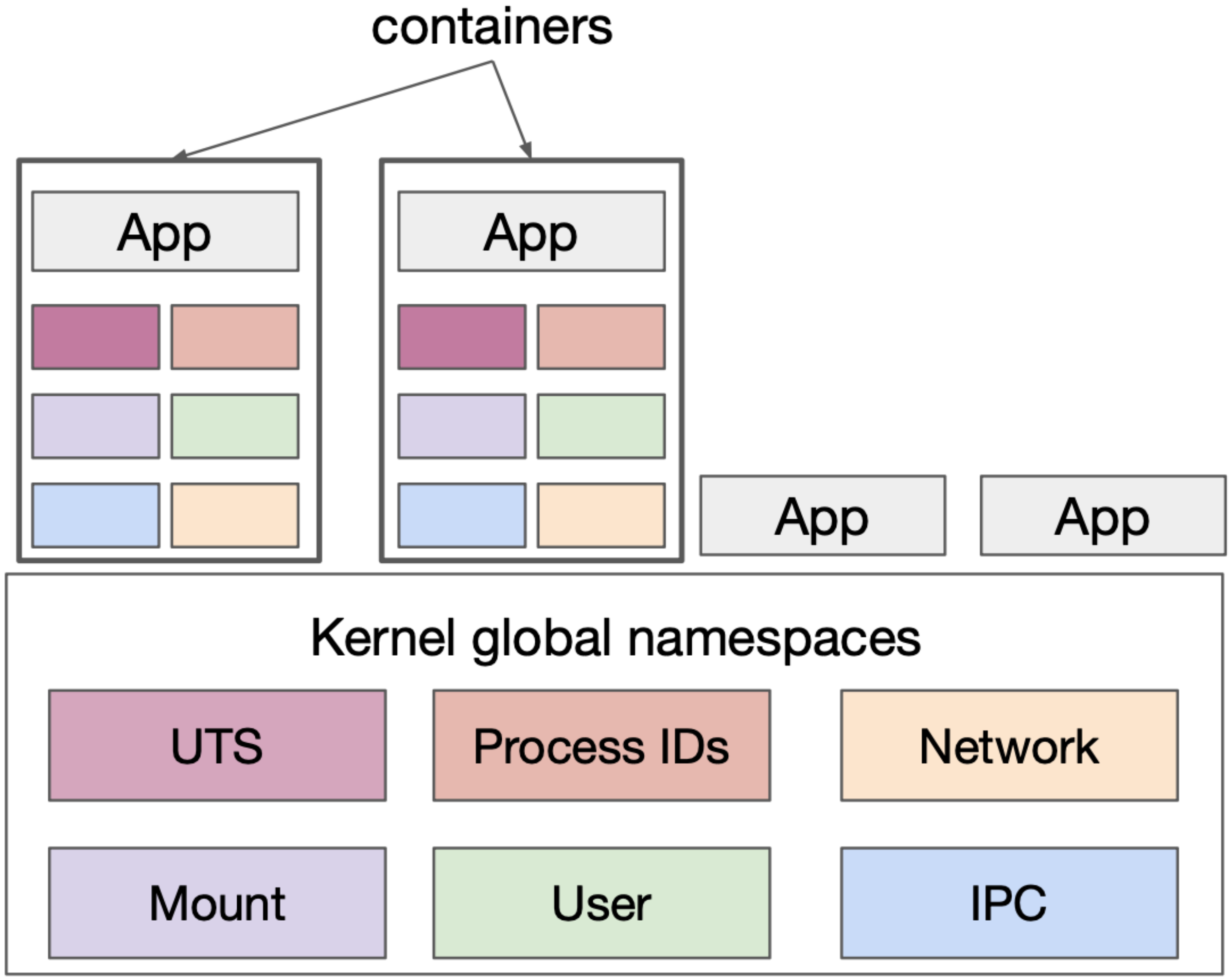


Application Isolation - Containers

- **Idea3: Limit Kernel Access for each container (System Call Filter)**

- Linux has >300 system calls
- Restrict accessible system calls to container
- Goal: Reduce TCB

- Seccom:
  - A file of black/white list
  - Include the file as running option



Application Isolation - Containers

- **Idea3: Limit Kernel Access for each container (System Call Filter)**

- Linux has >300 system calls
- Restrict accessible system calls to container
- Goal: Reduce TCB

- Seccom:
  - Pass a filter to the kernel
  - Process can add filters


```
"defaultAction": "SCMP_ACT_ALLOW",
"architectures": [
    "SCMP_ARCH_X86_64",
    "SCMP_ARCH_X86",
    "SCMP_ARCH_X32"
],
"syscalls": [
    {
        "name": "chmod",
        "action": "SCMP_ACT_ERRNO",
        "args": []
    },
      {
        "name": "fchmod",
        "action": "SCMP_ACT_ERRNO",
        "args": [

        ]
      },
      {
        "name": "fchmodat",
        "action": "SCMP_ACT_ERRNO",
        "args": [

        ]
      },
    {
        "name": "chown",
        "action": "SCMP_ACT_ERRNO",
        "args": []
```

Application Isolation - Containers

- Namespace + cgroups + seccomp = LXC
- LXC + Management Tools = Docker

- Docker vs chroot?
- Docker vs VMM?

Summary

- Isolation necessary for security
- Isolation at different levels
  - Kernel/ Hypervisor
  - Process
  - VM
  - Container
- Know your threat model
- Other goals
  - Portability
  - Performance