

CSE 120

Operating Systems Principles

Spring 2025

Lecture 18: Protection

Amy Ousterhout

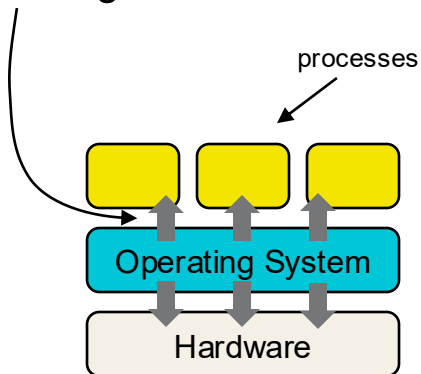
Today's Outline

- Virtual machines (continued)
- Containers
- Protection

Processes vs. Virtual Machines

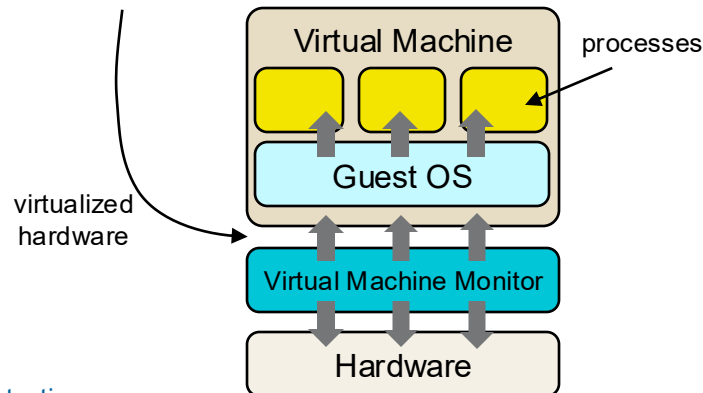
- Abstractions for processes:

- Virtual memory
- System calls
- Most instructions in the ISA
- Most registers



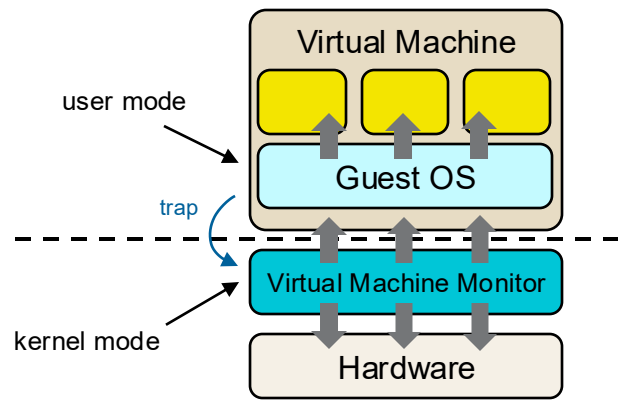
- Abstractions for virtual machines:

- Physical memory
- Interrupts
- All instructions in the ISA
- All registers
- I/O devices



VMM Approach

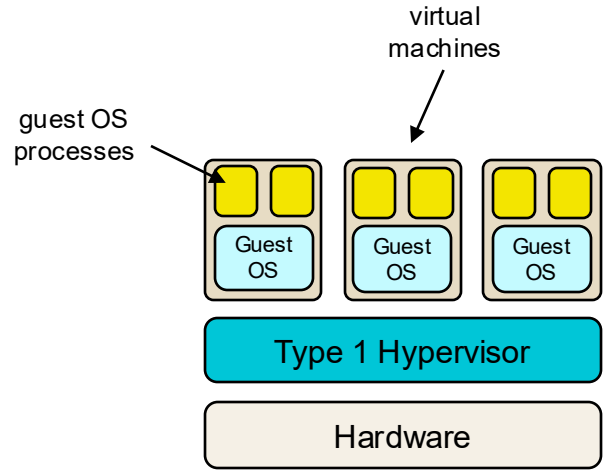
- Run the VMM in kernel mode
- Run the guest OS in user mode
 - Most instructions execute at regular CPU speed
- Anything “unusual” causes a trap to the VMM
- VMM has 2 options:
 - Call back into the guest OS for handling
 - Simulate the appropriate behavior with **trap-and-emulate**



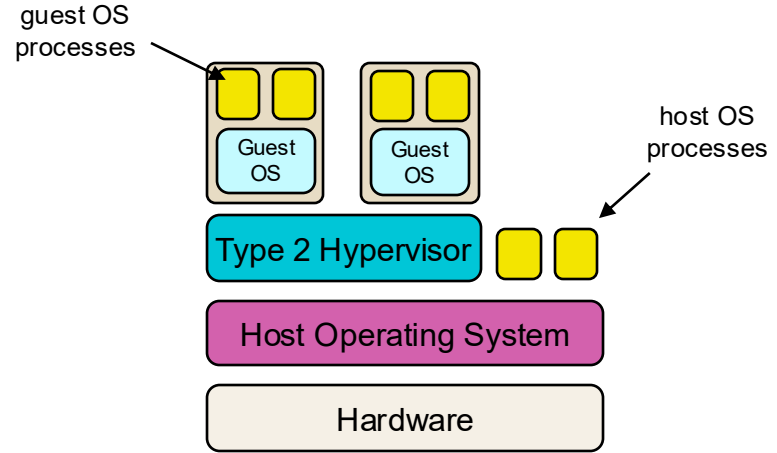
Virtualizing the x86 Architecture

- **Paravirtualization**
 - Change the guest OS to better cooperate with the VMM
- **Binary translation**
 - Run guest OS code under control of a **binary translator**
 - Rewrites privileged instructions with emulation at runtime (may trap to VMM)
- **Hardware support**
 - Intel and AMD added virtualization support in 2005 (Intel VT-x, AMD-V)

Type 1 and Type 2 Hypervisors



Type 1



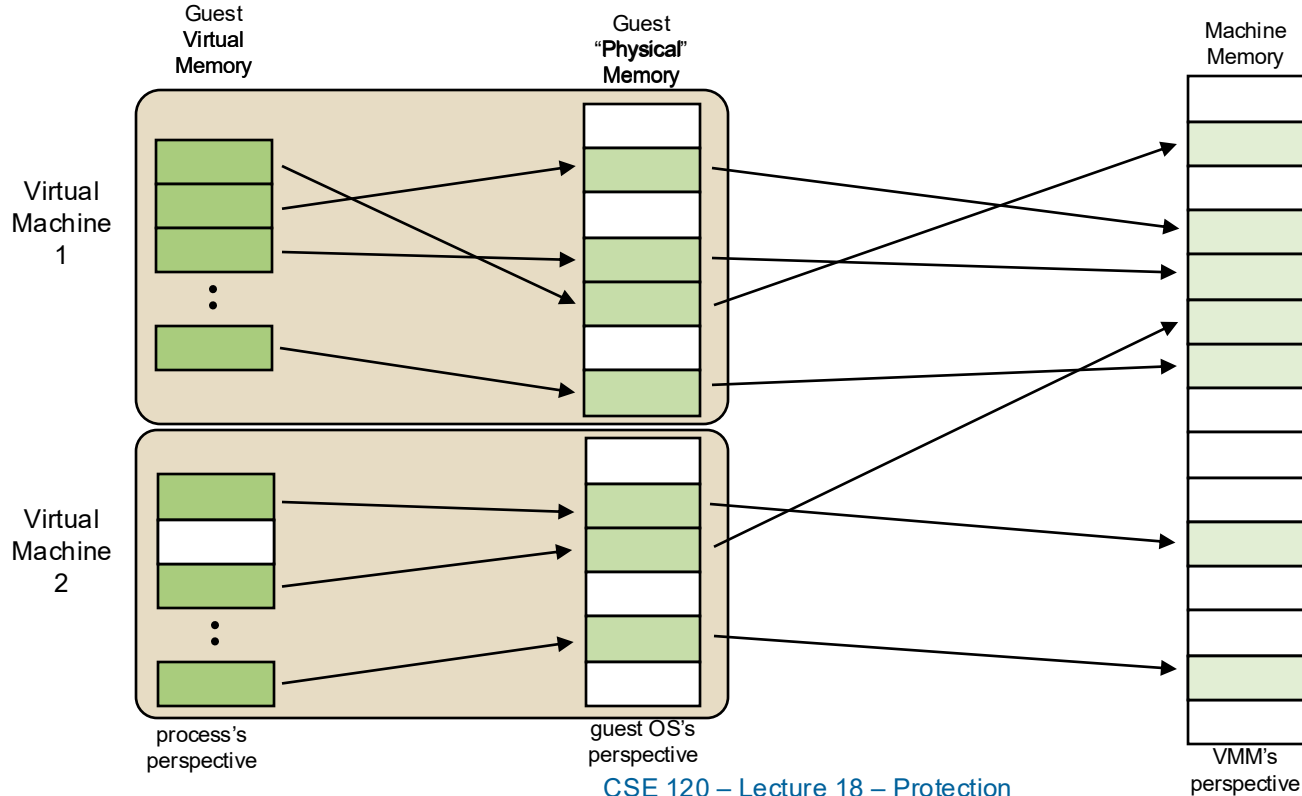
Type 2
(hosted hypervisor)



What Needs to be Virtualized?

- Events (exceptions and interrupts)
- CPU
- Memory
- I/O devices

Virtualizing Memory

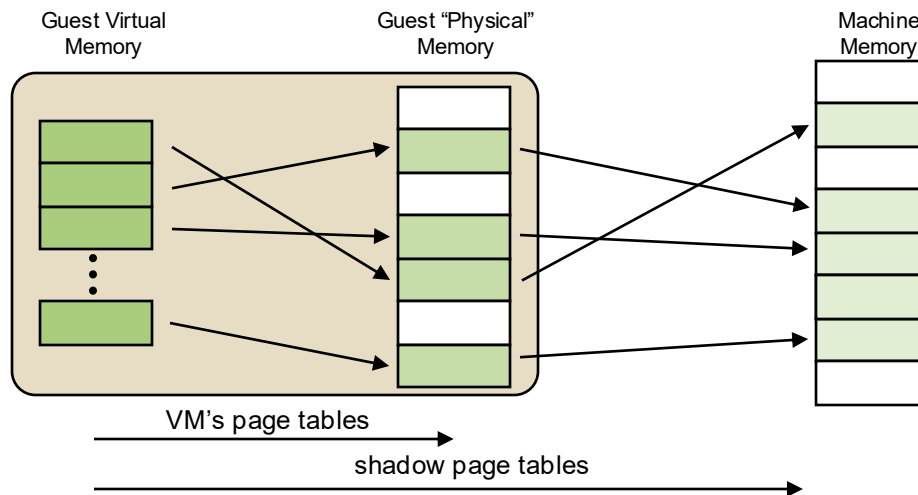


Virtualizing Memory

- Challenges:
 - VMM needs to assign hardware pages to VMs
 - Hardware-managed TLBs – hardware will walk page tables with no opportunity for the VMM to run

Shadow Page Tables

- One approach – shadow page tables
 - VMM maintains shadow page tables for each VM
 - Shadow page tables map from virtual pages in the VM to physical pages allocated by the VMM

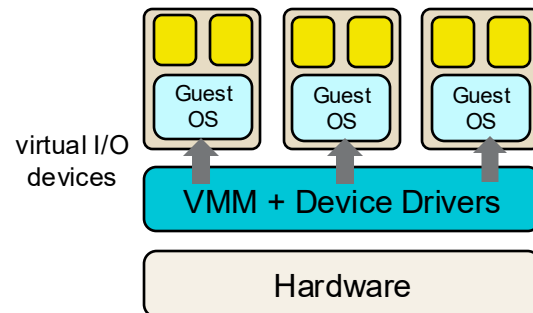


Shadow Page Tables

- MMU points to **shadow page tables**
- When VM tries to change MMU to point to a different page table:
 - Traps to VMM which updates MMU to point to the shadow page table
- Keeping shadow page tables in sync with guest page tables:
 - Mark pages of guest OS page tables as read only
 - When guest OS writes to page table, trap to VMM, VMM updates shadow page table
- VMM can also swap out pages and indicate this in the shadow page tables
- Cons:
 - Lots of overhead to trap to the VMM

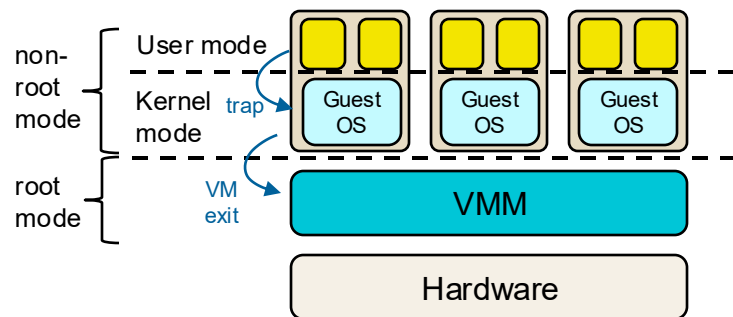
Virtualizing I/O

- Challenge:
 - Lots of I/O devices
 - We don't want to write virtualized device drivers for all possible I/O devices
- One approach:
 - Run real device drivers in the VMM
 - VMM presents simple **virtual I/O devices** to guest VMs
- Can optimize using paravirtualization or specialized hardware



Hardware Support for Virtualization

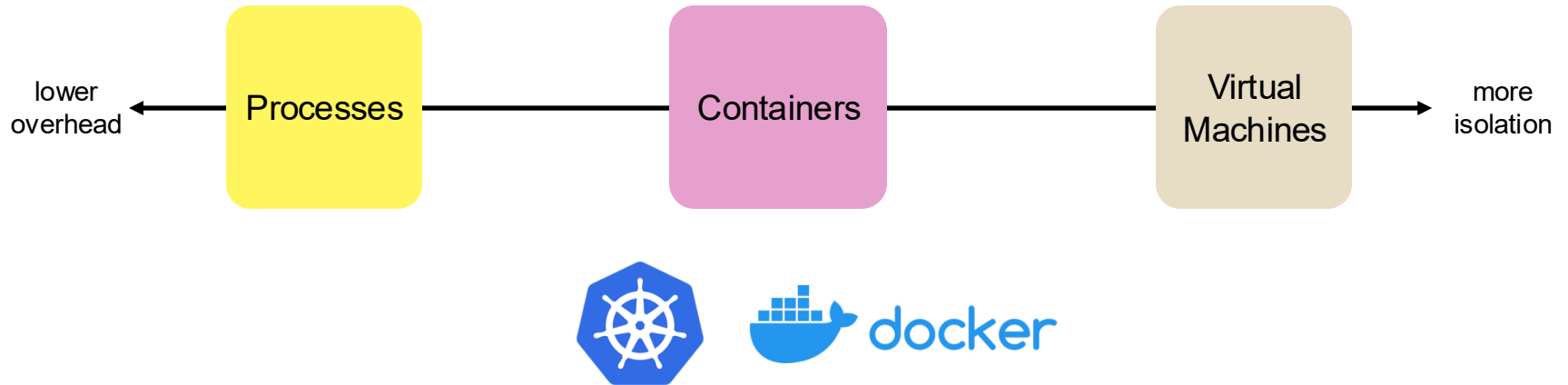
- Most modern CPUs provide virtualization support in CPUs in hardware
 - Intel VT-x, AMD-V, RISC-V H-extension
- Privileged instructions
 - New execution mode: non-root mode
 - Traps from VM processes go to the Guest OS
- Interrupts
 - Hardware delivers them directly to the right VM
- I/O
 - SR-IOV virtualizes I/O devices (e.g., NIC, storage device)
- Memory
 - Intel Extended Page Tables (EPT) virtualize page tables



Abstractions for Virtualization

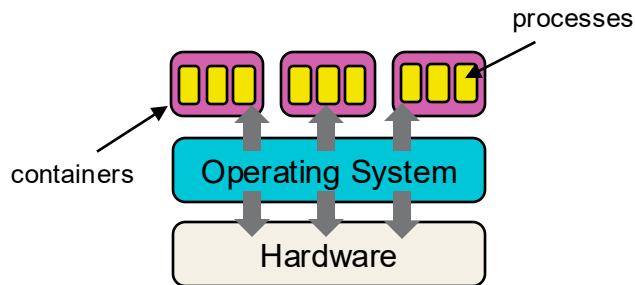
- Processes
- Virtual machines
 - Types of Virtual Machine Monitors
 - Virtualization components
 - » Processor
 - » I/O
 - » Memory
- Containers
 - Namespace isolation
 - Resource management

Virtualization Tradeoffs



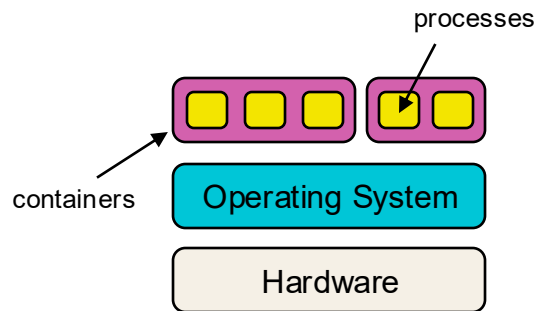
Container-Based OS Virtualization

- Containers virtualize OS abstractions
 - Group of processes that appear to have an entire OS to themselves
 - Manage isolated sets of OS objects (using **namespaces**)
 - Manage isolated sets of resources (using **resource management**)
- Lightweight virtualization
 - Containers on the same machine share an OS



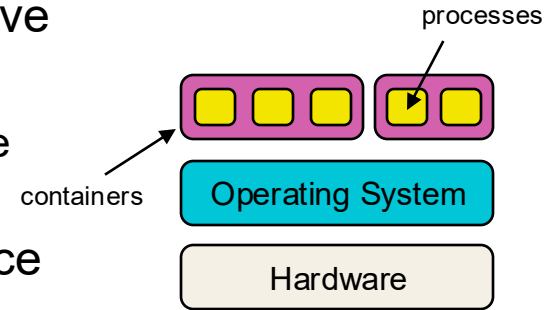
Namespace Isolation

- **Namespace:** set of names for a kind of object
- OS implements a collection of namespaces
 - Process IDs (PIDs): processes
 - User IDs: users and groups
 - Network: IP address, network ports, routing
 - File system: files and directories
- Namespace Isolation
 - Each container has its own namespaces
 - **Principle:** if you cannot name it, you are isolated from it
 - » Similar to virtual memory



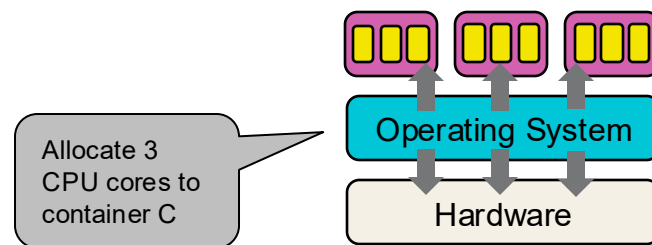
Implementing Namespace Isolation

- OS tags objects with the namespace that they belong to
 - A process has both a PID and a namespace identifier
- **Mappings** map container-local IDs to global perspective
 - OS tracks processes on global lists
 - OS maps from PID within a container to a process on the global list
- **Filters** restrict which objects are visible in a namespace
 - E.g., running `ps` in a namespace only shows processes in the namespace



Resource Management in Containers

- Goal: control resources assigned to containers
 - How much CPU, memory, disk, and network
- OS manages resources at a process granularity by default
- Containers manage resources among a set of processes
 - Which cores can be used, how much overall CPU utilization
 - How much memory (e.g., paging among processes in a group)
 - How much disk and network I/O
- In Linux: control groups (**cgroups**)



Today's Outline

- Virtual machines (continued)
- Containers
- Protection

Protection

- **Protection**: mechanisms that prevent accidental or intentional misuse of a system
- Three components:
 - **Authentication** – identify a responsible party behind each action
 - **Authorization** – determine which parties are allowed to perform which actions
 - **Enforcement** – control access using authentication and authorization

Protection Principles

- **Permission rather than exclusion**
 - Default is no access (will quickly discover if wrong)
- **Check every access to every object**
 - Including every instruction and memory reference
- **The design should not be secret**
 - E.g., Linux is open source and that should not make it insecure
- **Principle of least privilege**
 - Only execute with the privileges you need (avoids mistakes)
- **User interface to protection must be easy to use**
 - If it is hard to use, users will find ways around it

Users

- Protection starts with the concept of a **user**
- Which user you are defines:
 - What programs you can run (execute)
 - Which files you can access and how (read, write)
- Cannot do anything on the system until you log in (**authentication**)
- Once you log in, everything you do on the system is performed under your user ID
 - Every process runs under a user ID
 - The user ID is the basis for protection checks



Root and Administrator

- The user “root” is special on Unix
 - It bypasses all protection checks in the kernel
 - Administrator is the equivalent on Windows
- Running as root can be dangerous
 - A mistake (or exploit) can harm the system
 - This is why we create user accounts even if you have root access
 - » Only run as root when you need to modify the system
 - » An example of the [principle of least privilege](#)

sudo and su

- The **sudo** command runs a process with root privileges
 - Authenticate using the **user's password**
 - User must be in the sudo group (/etc/group)
- The **su** command runs a shell with root privileges
 - Authenticate using the **password for the root user**
 - Effectively logging in as root
 - Less precise than sudo, more risky

Authorization

- **Authorization**: determines who is allowed to perform which actions
- More formally:
 - **Subjects** – who is performing the action (e.g., user, process)
 - **Objects** – what the action is being performed on (e.g., a file)
 - **Actions** – what the subject is allowed to do to the object
- Can a given subject perform a given action on a given object?

		Objects		
		/one	/two	/three
Subjects	Alice	rw	-	rw
	Bob	w	-	r
	Charlie	w	r	rw

Access Control Lists vs. Capability Lists

- **Access control lists:** organize by columns
 - For each object, maintain a list of which users are allowed to perform which actions
- **Capability lists:** organize by rows
 - For each subject, maintain a list of objects and their permitted actions

	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

Access Control Lists vs. Capability Lists

- Approaches differ only in how the table is “represented”
 - Different tradeoffs so we use them in different ways
- Capabilities are faster to check and easier to transfer
 - They are like keys, easy to hand off
 - Very fast to check
- ACLs are slower but easier to use
 - Slow to check compared to capabilities
 - Object-centric, easy to grant and revoke
 - Easier for users to express protection goals
 - » Recall: **interface must be easy to use**

access control list

Objects

	/one	/two	/three
Alice	rw	-	rw
Bob	w	-	r
Charlie	w	r	rw

Subjects

capability list

Operating Systems Use Both

- OSes use ACLs on objects in the file system
 - These are what users manipulate to express protection
- OSes use capabilities when checking access frequently
 - Checking every memory reference needs to be fast

File System Protection

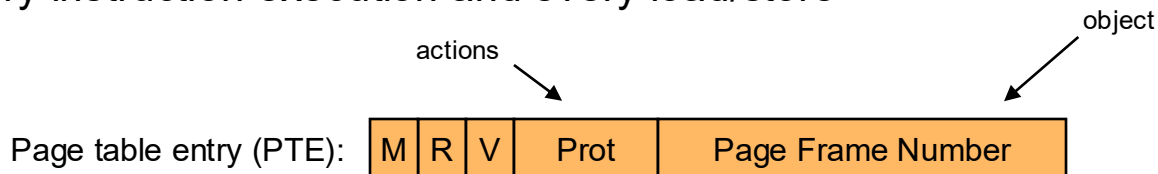
- File systems implement a static protection system
 - Who can access files, directories, devices, etc.
 - How they are allowed to access it
- The mechanism used to represent file system permissions is the **access control list**
 - Recall: **permission, not exclusion**
- For each object (file), which users have access to the object, and what actions can they take?
 - Can be compact: Unix's owner/group/other, read/write/execute
 - Can be flexible: an arbitrary list of user:rights entries

Checking File Permissions

- Recall: **check every access**
- For reading/writing a file, the OS needs to verify on every `read()/write()` that the process has permission to perform the syscall
- But, checking file permissions is expensive
 - Scanning ACLs on every read/write is slow
- How do we optimize the permissions check?
 - open syscall
- **Use file descriptors as capabilities**
 - The process passes this descriptor to every call to `read()/write()`
 - OS checks that the descriptor is valid and the action is allowed

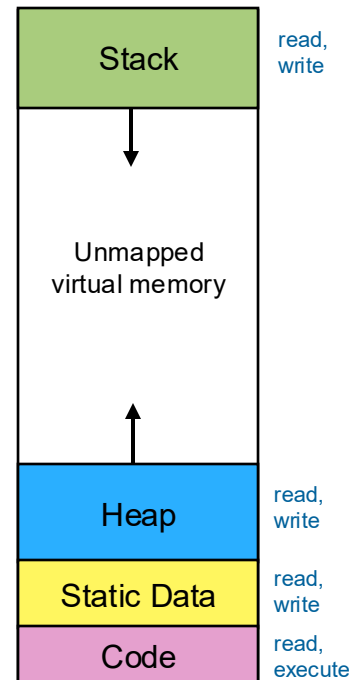
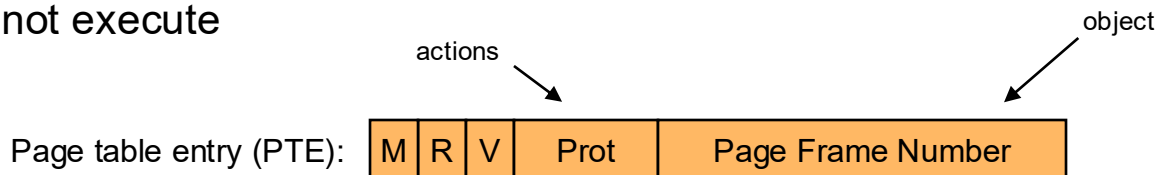
Virtual Memory Protection

- The address space defines permissions for a process under execution
 - It is a dynamic representation of permissions
- The mechanism used to represent virtual memory protection is **capabilities**
- Page table entries are our virtual memory capabilities
 - Every PTE refers to a page of memory
 - Specifies what the process is allowed to do with that page
- Recall: **check every access**
 - Every instruction execution and every load/store



Origins of PTEs

- PTEs are capabilities
 - Where are they derived from?
- Loading a process and creating the address space
 - Code pages: set the PTE protection bits to read-only and execute
 - Data pages: set the PTE protection bits to read/write but not execute
- As a process executes
 - Stack and heap pages: set the PTE protection bits to read/write but not execute

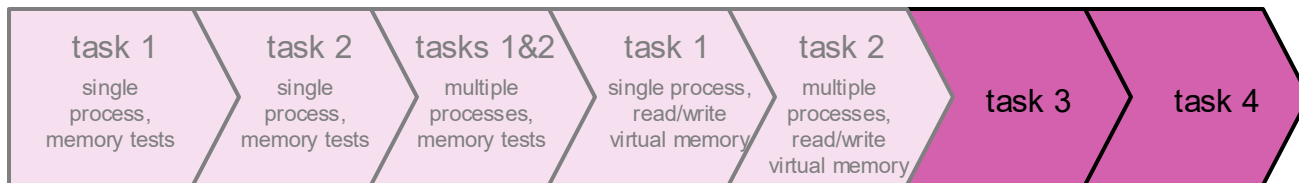


Course Evaluations

- Student Evaluation of Teaching (SET)
 - Evaluate and provide feedback to instructors, TAs, and tutors
 - » I use your feedback to improve the course for future students
 - Your feedback helps other students choose classes and instructors
 - » Feedback is anonymized and summarized

Upcoming Tasks

- Homework 4
 - Due Tuesday 6/3 at 11:59 pm
- Project 3
 - Due Friday 6/6 at 11:59 pm, no option to submit late



- Final Exam
 - Monday, June 9th, 3:00-6:00 pm in the REC gym
 - We will review in class on Thursday
 - My Thursday office hours will be on Friday (6/6), 1-2 pm instead