

# **CSE 120**

# **Principles of Operating Systems**

**Fall 2023**

**Lecture 9: Memory Management**

Geoffrey M. Voelker

# Administrivia

---

- Project 2 released
  - ◆ This lecture gets you jumpstarted
- Homework 3 released
  - ◆ Do VM worksheet before part 2 of project 2

# Memory Management

---

Next few lectures are going to cover memory management

- Goals of memory management
  - ◆ To provide a convenient abstraction for programming
  - ◆ To allocate scarce memory resources among competing processes to maximize performance with minimal overhead
- Mechanisms
  - ◆ Physical and virtual addressing (1)
  - ◆ Techniques: partitioning, paging, segmentation (1)
  - ◆ Page table management, TLBs, VM tricks (2)
- Policies
  - ◆ Page replacement algorithms (3)

# Virtual Memory

---

- The abstraction that the OS provides for managing memory is **virtual memory** (VM)
  - ♦ Virtual memory enables a program to execute with less than its complete data in physical memory
    - » A program can run on a machine with less memory than it “needs”
    - » Can also run on a machine with “too much” physical memory
  - ♦ Many programs do not need all of their code and data at once (or ever) → no need to allocate physical memory for it
  - ♦ OS will adjust amount of physical memory allocated to a process based upon its behavior
  - ♦ VM requires hardware support and OS management algorithms to pull it off
- Let’s go back to the beginning...

# In the beginning...

---

- With early computers...
  - ♦ Programs used **physical addresses** directly (e.g., early PCs)
  - ♦ OS loads job, runs it, unloads it
- Multiprogramming changes all of this
  - ♦ Want multiple processes in memory at once
    - » Overlap I/O and CPU of multiple jobs
  - ♦ Can do it a number of ways
    - » Fixed and variable partitioning, paging, segmentation
  - ♦ Requirements
    - » Need protection – restrict which addresses jobs can use
    - » Fast translation – lookups need to be fast
    - » Fast change – updating memory hardware on context switch

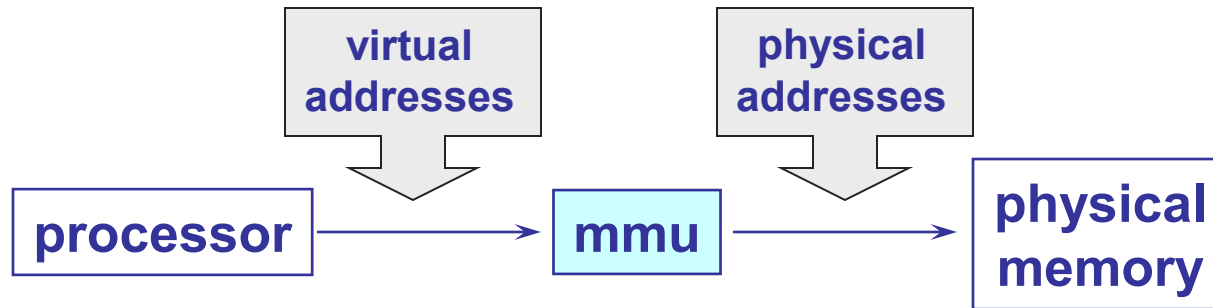
# Virtual Addresses

---

- To make it easier to manage the memory of processes running in the system, we're going to make them use **virtual addresses** (logical addresses)
  - ◆ Virtual addresses are independent of the actual physical location of the data referenced
  - ◆ OS determines location of data in physical memory
  - ◆ Instructions executed by the CPU issue virtual addresses
  - ◆ Virtual addresses are translated by hardware into physical addresses (with help from OS)
- The set of virtual addresses that can be used by a process comprises its **virtual address space (VAS)**
  - ◆ VAS often larger than physical memory (64-bit addresses)
  - ◆ But can also be smaller (32-bit VAS with 8 GB of DRAM)

# Virtual Addresses

---



- Many ways to do this translation...
  - ◆ Start with old, simple ways, progress to current techniques

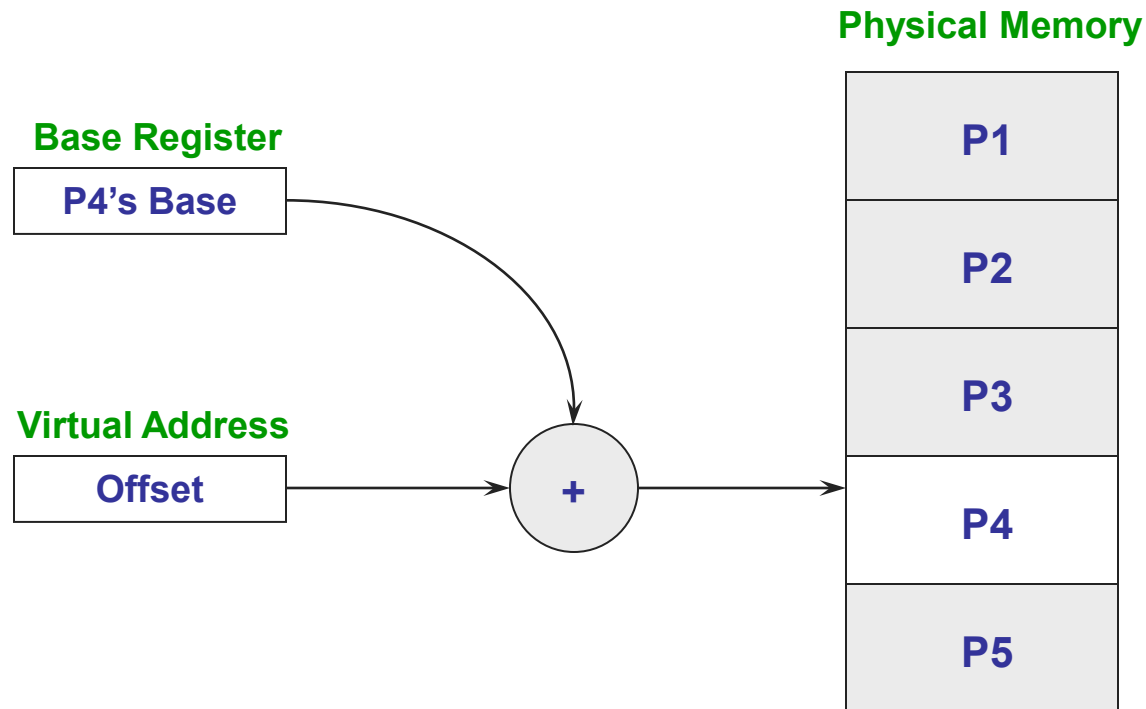
# Fixed Partitions

---

- Physical memory is broken up into fixed partitions
  - ◆ Hardware requirements: **base register**
  - ◆ Physical address = virtual address + base register
  - ◆ Base register loaded by OS when it switches to a process
  - ◆ Size of each partition is the same and fixed
  - ◆ **How do we provide protection?**
- Advantages
  - ◆ **Easy to implement, fast context switch**
- Problems
  - ◆ **Internal fragmentation**: memory in a partition not used by a process is not available to other processes
  - ◆ **Partition size**: one size does not fit all (very large processes)

# Fixed Partitions

---

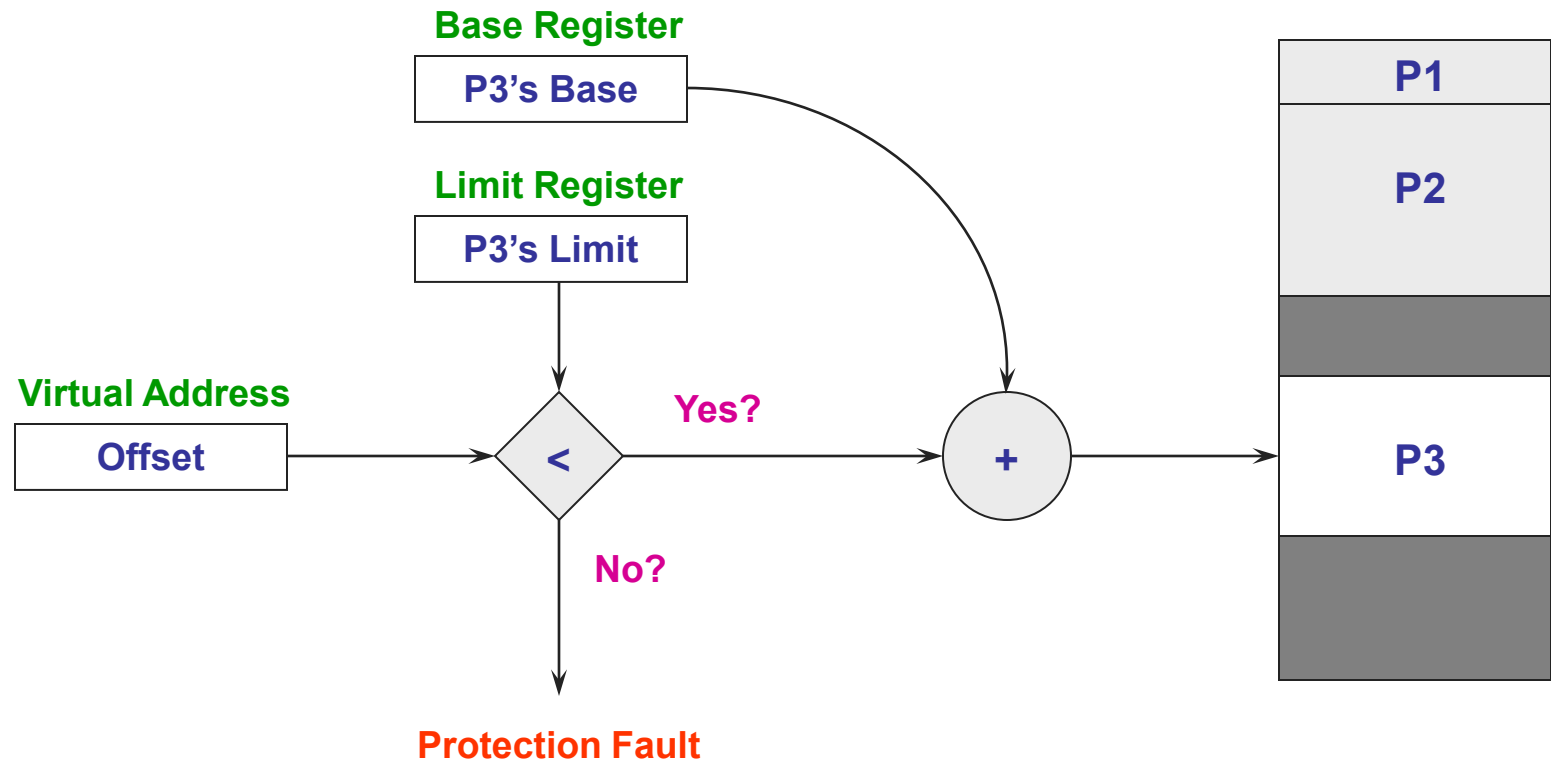


# Variable Partitions

---

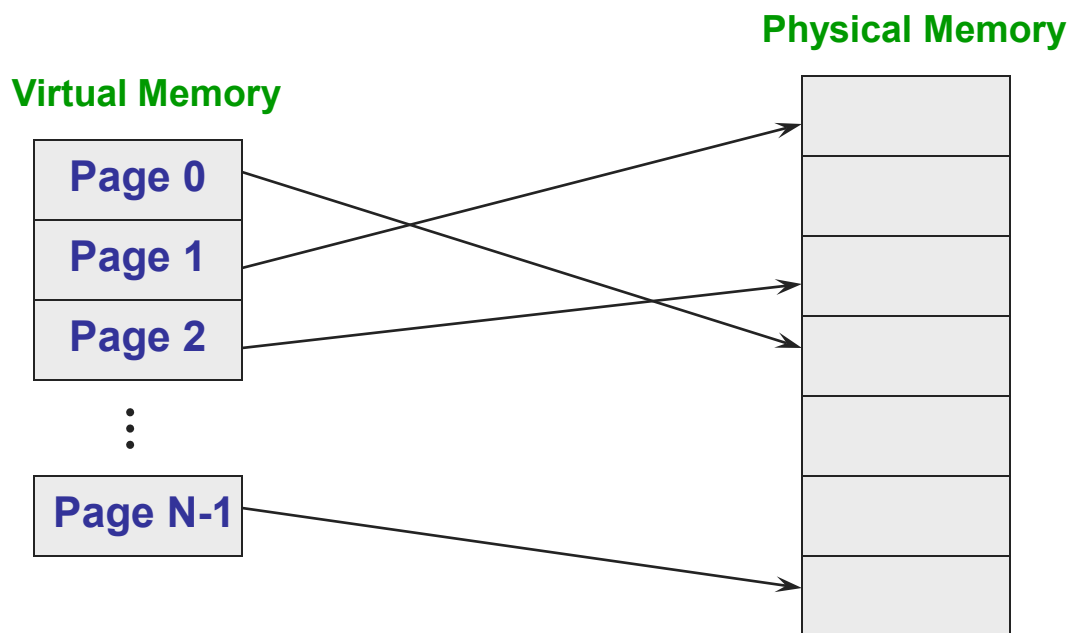
- Natural extension – physical memory is broken up into variable sized partitions
  - ♦ Hardware requirements: **base register** and **limit register**
  - ♦ Physical address = virtual address + base register
  - ♦ Why do we need the limit register? Protection
    - » If (physical address > base + limit) then fault (exception)
- Advantages
  - ♦ **No internal fragmentation**: allocate just enough for process
- Problems
  - ♦ **External fragmentation**: process creation and termination produces empty holes scattered throughout memory

# Variable Partitions



# Paging

- Paging solves the external fragmentation problem by using fixed-sized units in both physical and virtual memory



# Programmer/Process View

---

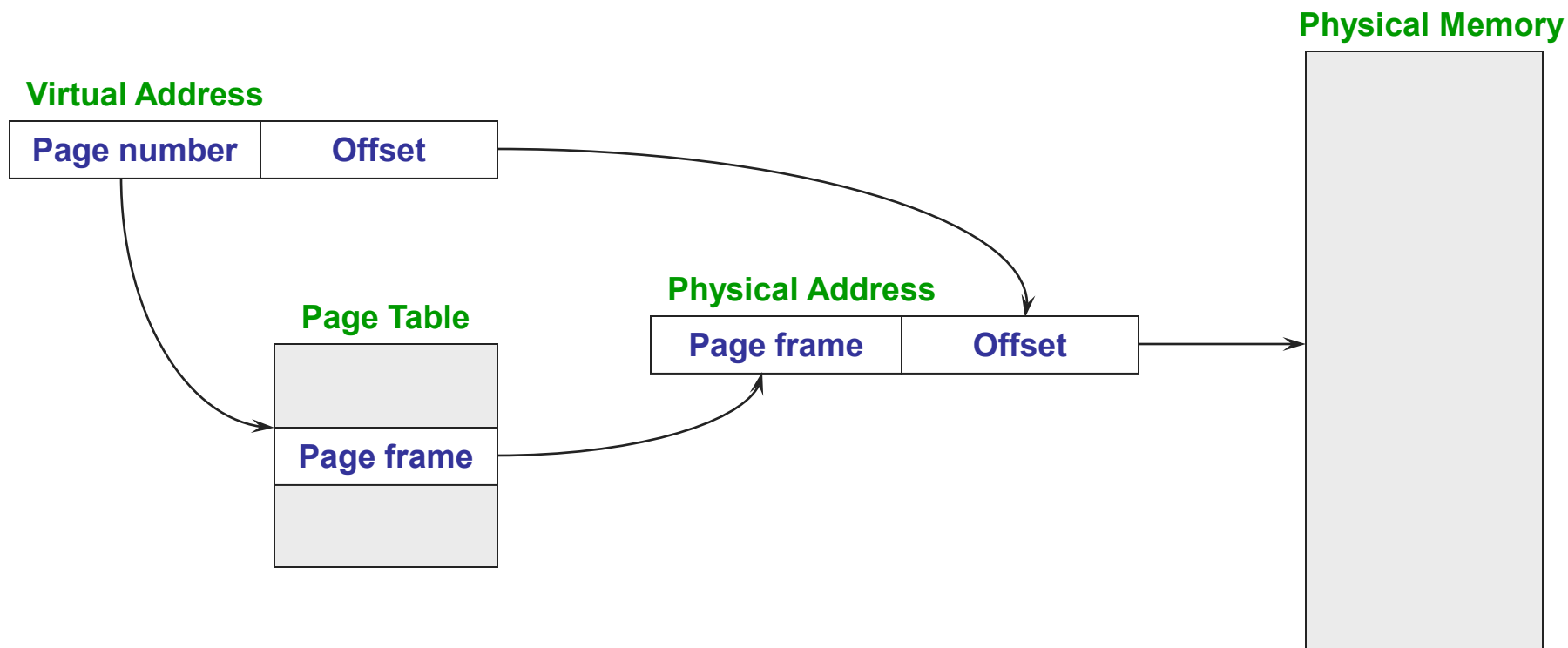
- Programmers (and processes) view memory as one contiguous address space from 0 through N
  - ◆ Virtual address space (VAS)
- In reality, pages are scattered throughout physical memory
  - ◆ The mapping is invisible to the process
- Protection is provided because a process cannot reference memory outside of its VAS
  - ◆ The address “0x1000” maps to different physical addresses in different processes

# Paging

---

- Translating addresses
  - ◆ Virtual address has two parts: **virtual page number** and **offset**
  - ◆ Virtual page number (VPN) is an index into a page table
  - ◆ Page table determines page frame number (PFN)
  - ◆ Physical address is PFN::offset (“::” means concatenate)
- Page tables
  - ◆ Map **virtual page number** (VPN) to **page frame number** (PFN)
    - » VPN is the index into the table that determines PFN
    - » Will also refer to PFN as “physical page number”
  - ◆ One page table entry (PTE) per page in virtual address space
    - » Or, one PTE per VPN

# Page Lookups



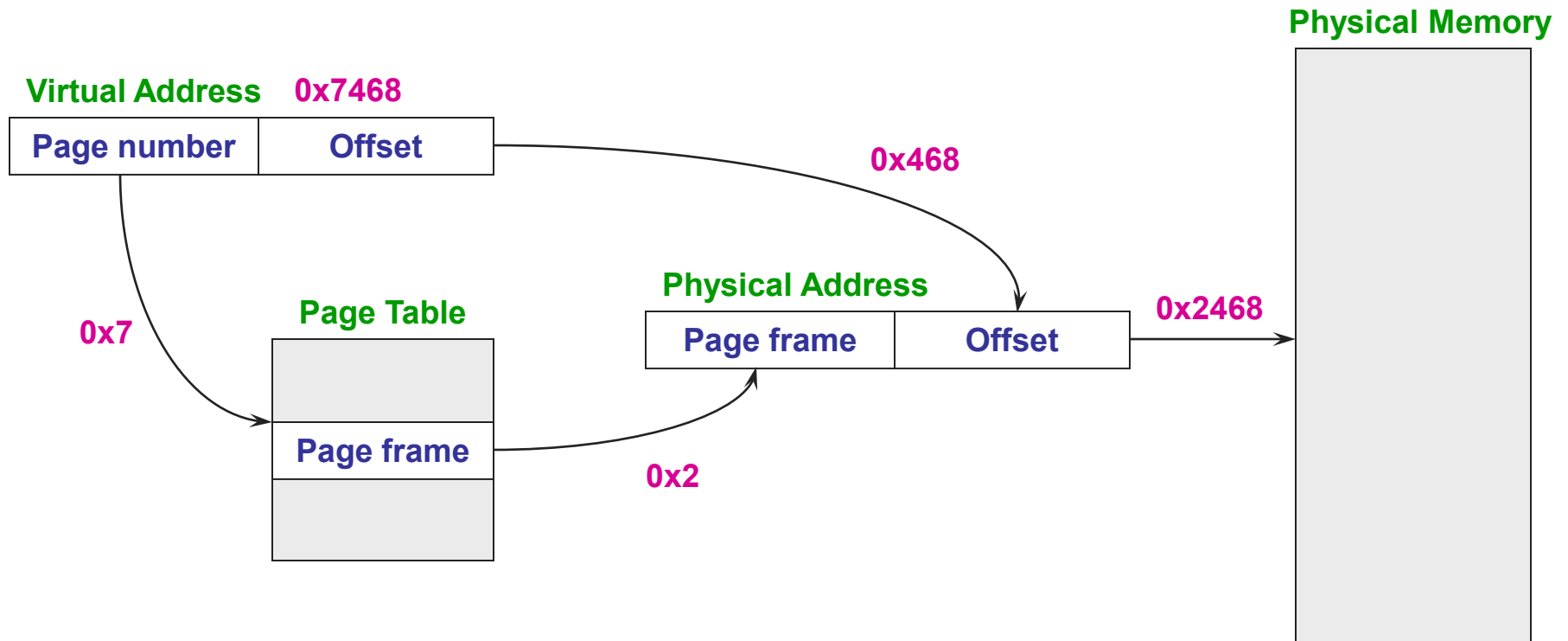
(Simple page table used by Nachos)

# Paging Example

---

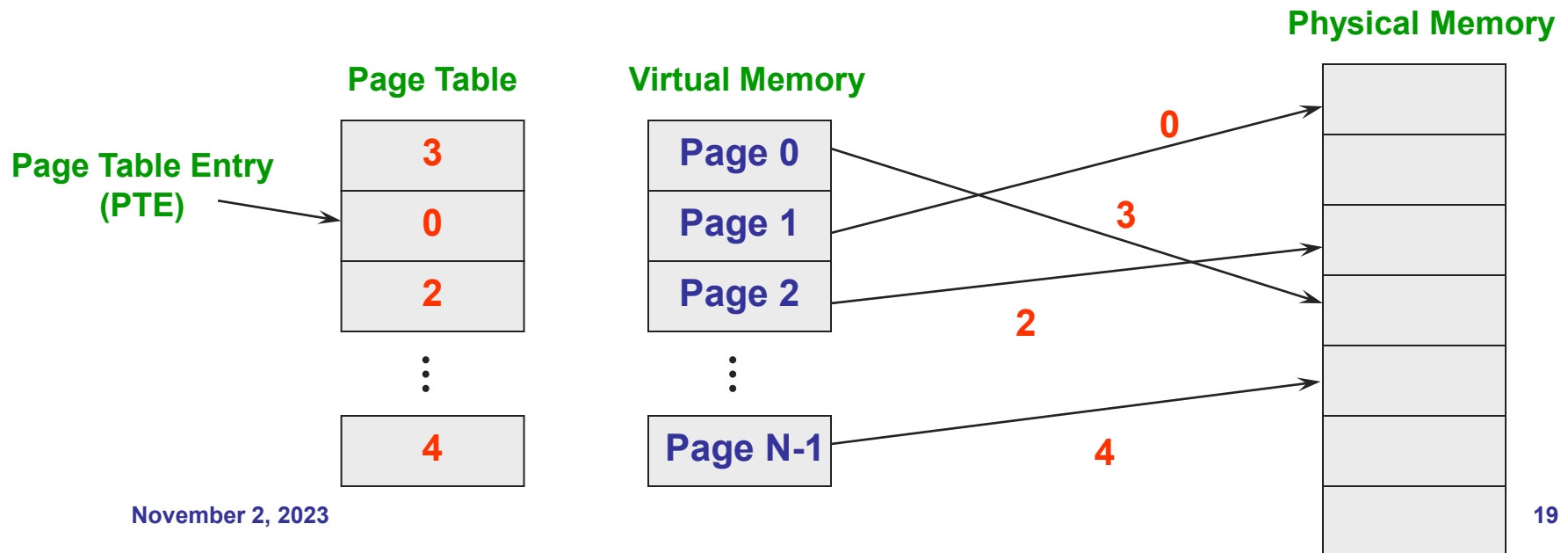
- Pages are 4K
  - ◆ 4K → offset is 12 bits → VPN is 20 bits ( $2^{20}$  VPNs)
- Virtual address is 0x7468
  - ◆ Virtual page is 0x7, offset is 0x468 (lowest 12 bits of address)
- Page table entry 0x7 contains 0x2
  - ◆ Page frame number is 0x2
  - ◆ Virtual page 0x7 is at address 0x2000 (physical page 2)
- Physical address = 0x2 :: 0x468 = 0x2468

# Page Lookups



# Page Tables

- Page tables completely define the mapping between virtual pages and physical pages for an address space
- Each process has an address space, so each process has a page table
- Page tables are data structures maintained in the OS



# Page Table Entries (PTEs)



- Page table entries control mapping
  - ♦ The **Modify** bit says whether or not the page has been written
    - » It is set when a write to the page occurs
  - ♦ The **Reference** bit says whether the page has been accessed
    - » It is set when a read or write to the page occurs
  - ♦ The **Valid** bit says whether or not the PTE can be used
    - » It is checked each time the virtual address is used
  - ♦ The **Protection** bits say what operations are allowed on page
    - » Read, write, execute
    - » **PTE for page 0 often set no-read, no-write, no-execute**
  - ♦ The **page frame number** (PFN) determines physical page

# Paging Advantages

---

- Easy to allocate memory
  - ◆ Memory comes from a free list of fixed-sized chunks
  - ◆ Allocating a page is just removing it from the list
  - ◆ External fragmentation not a problem
- Easy to swap out chunks of a program
  - ◆ All chunks are the same size
  - ◆ Use valid bit to detect references to swapped pages
  - ◆ Pages are a convenient multiple of the disk block size

# Paging Limitations

---

- Can still have internal fragmentation
  - ◆ Process may not use memory in multiples of a page
- Memory reference overhead
  - ◆ 2 references per address lookup (page table, then memory)
  - ◆ Solution – use a hardware cache of lookups (more later)
- Memory required to hold page table can be significant
  - ◆ Need one PTE per page
  - ◆ 32 bit address space w/ 4KB pages =  $2^{20}$  PTEs
  - ◆ 4 bytes/PTE = 4MB/page table
  - ◆ 25 processes = 100MB just for page tables!
  - ◆ Solution – hierarchical page tables (more later)

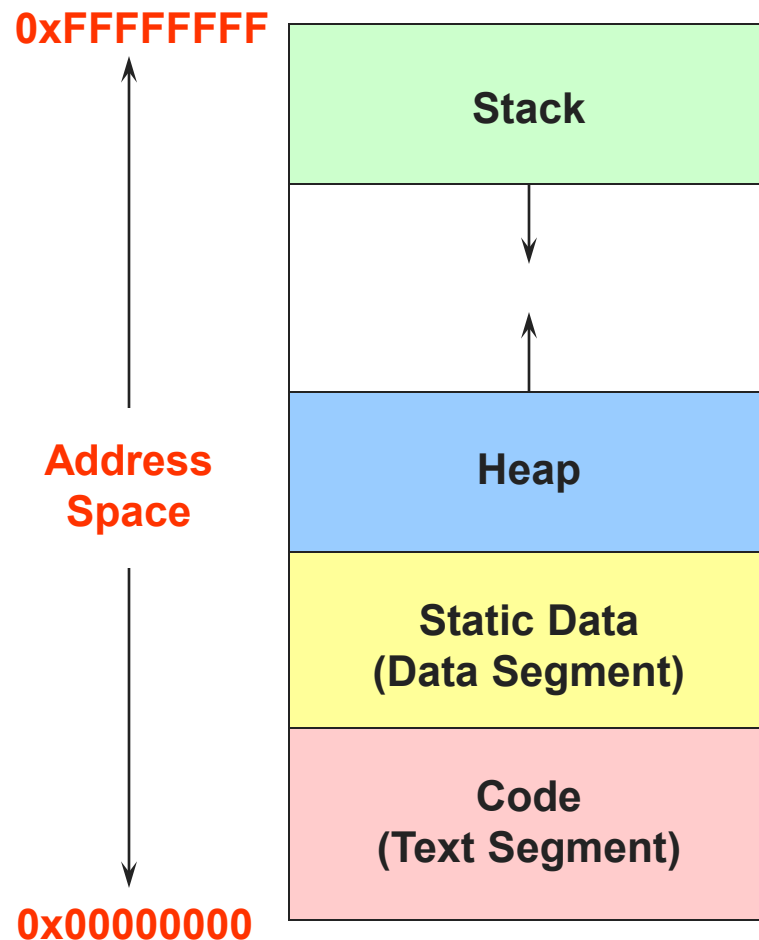
# Segmentation

---

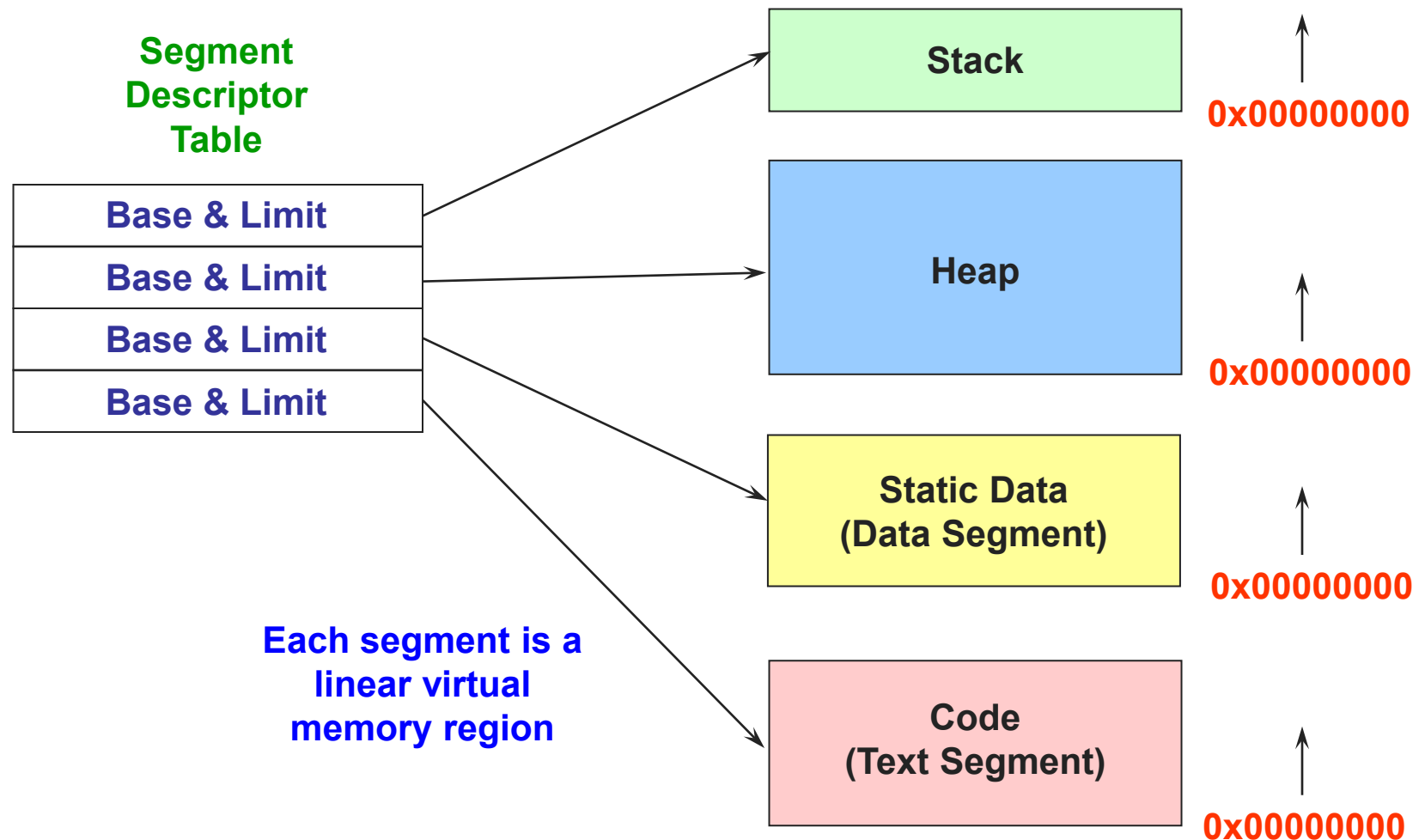
- Segmentation is a technique that partitions memory into logically related data units
  - ◆ Module, procedure, stack, data, file, etc.
  - ◆ Virtual addresses become `<segment #, offset>`
    - » x86 stores segment #s in registers (CS, DS, SS, ES, FS, GS)
  - ◆ Units of memory from programmer's perspective
- Natural extension of variable-sized partitions
  - ◆ Variable-sized partitions = 1 segment/process
  - ◆ Segmentation = many segments/process
- Hardware support
  - ◆ Multiple base/limit pairs, one per segment (segment table)
  - ◆ Segments named by #, used to index into table

# Linear Address Space

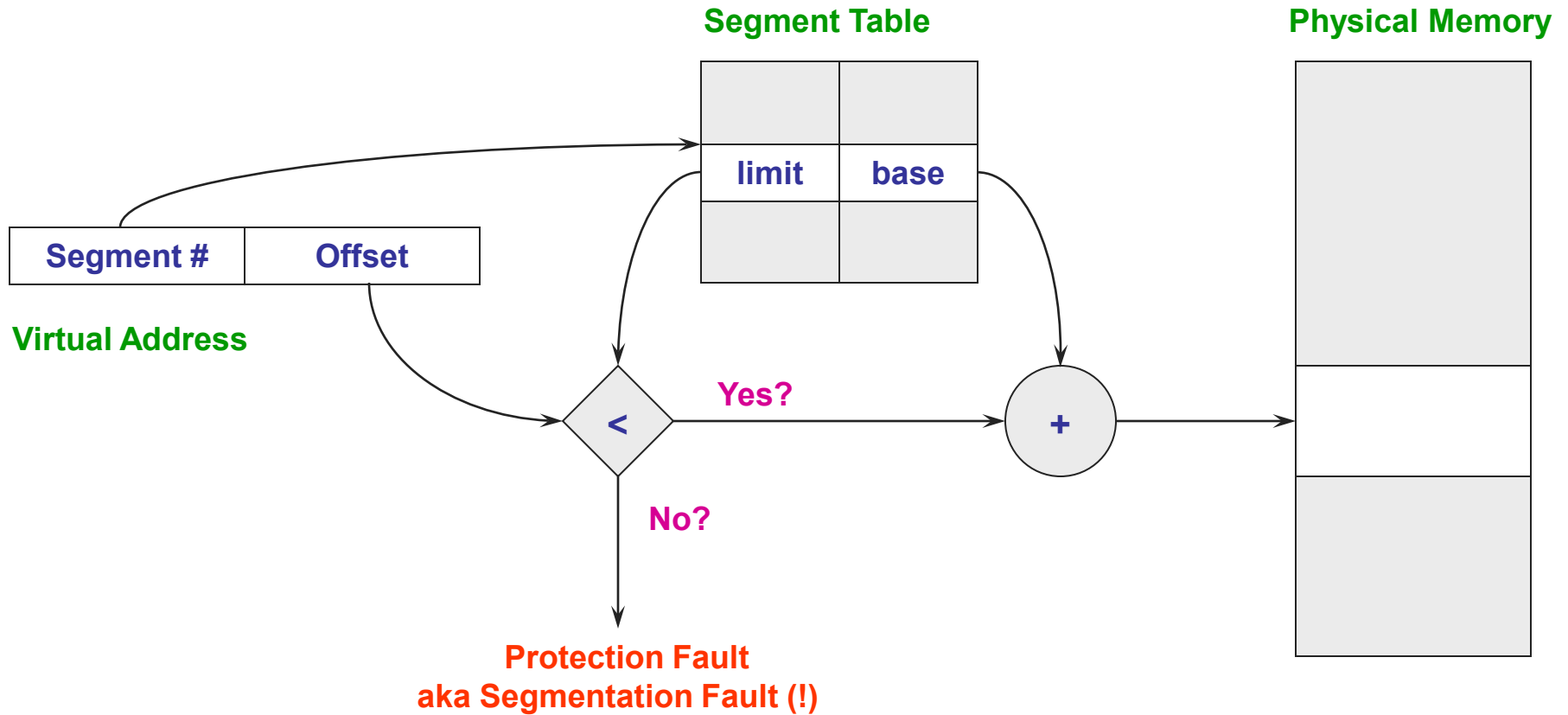
---



# Segmented Address Space



# Segment Lookups



# Segmentation and Paging

---

- Can combine segmentation and paging
  - ◆ x86 in 32-bit mode supports segments and paging
- Use segments to manage logically related units
  - ◆ Segments vary in size, but usually large (many pages)
- Use pages to partition segments into fixed-size chunks
  - ◆ Makes segments easier to manage within physical memory
    - » Segments become “pageable” – rather than moving segments into and out of memory, just move page portions of segment
  - ◆ Need to allocate page table entries only for those pieces of the segments that have themselves been allocated
- Tends to be complex
  - ◆ Need both segment tables *and* page tables

# Summary

---

- Virtual memory
  - ◆ Processes use virtual addresses
  - ◆ Hardware translates virtual address into physical addresses with OS support
- Various techniques
  - ◆ Fixed partitions – easy to use, but internal fragmentation
  - ◆ Variable partitions – more efficient, but external fragmentation
  - ◆ Paging – use small, fixed size chunks, efficient for OS
  - ◆ Segmentation – manage from programmer's perspective
  - ◆ Combine paging and segmentation to get benefits of both

# Next time...

---

- Chapters 19, 20