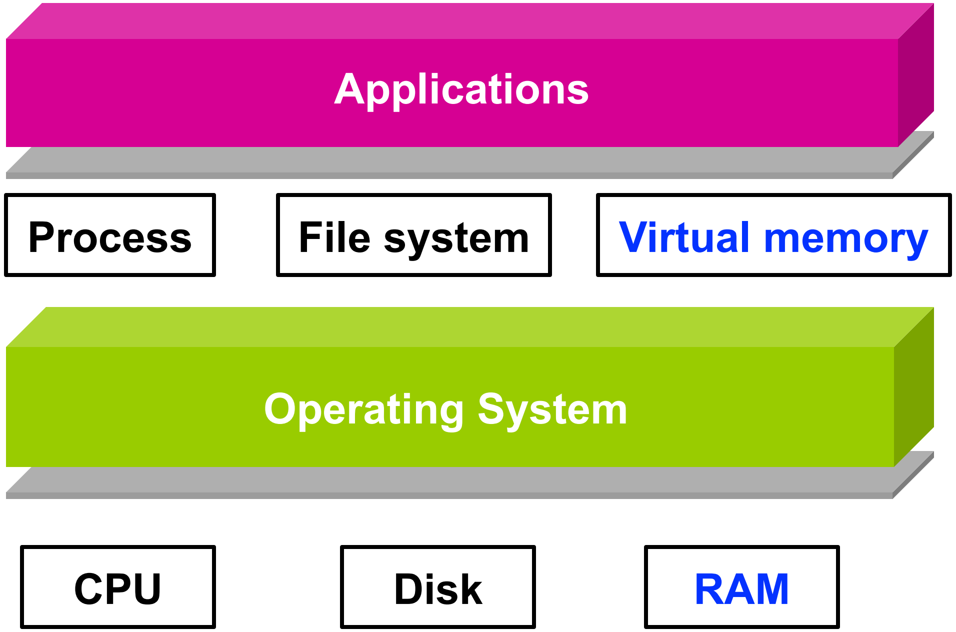
CS 153
Design of Operating Systems

Fall 19

Lecture 9: Virtual Address Space

Instructor: Chengyu Song

OS Abstractions



What is Memory?

- From programmers' perspective
  - A "place" to store data
- How to access data in memory?
  - Variables?
  - Names?
  - Addresses?
- Memory can be viewed as a big array
  - **content** = **memory[address]**

Minimal addressable data size

Need for Virtual Address Space

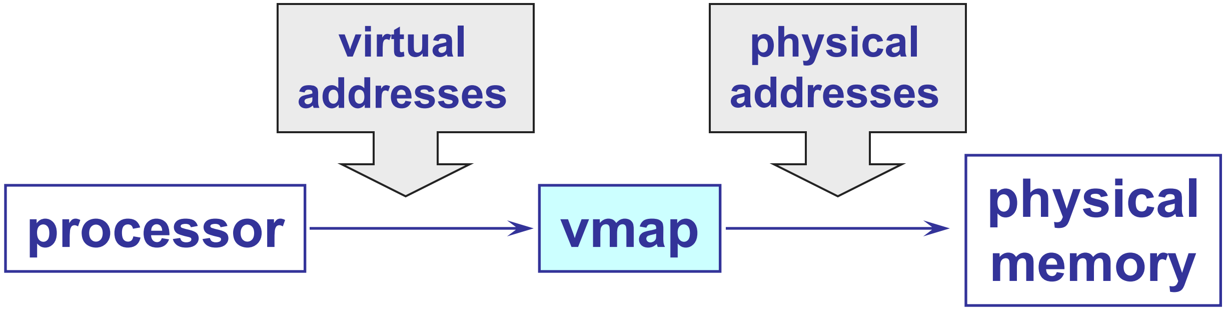
- Rewind to the days of “second-generation” computers
  - Programs use physical addresses directly
  - OS loads job, runs it, unloads it
- Multiprogramming changes all of this
  - Want multiple processes in memory at once
    - Overlap I/O and CPU of multiple jobs
  - How to share physical memory across multiple processes?
    - Programmers cannot predict where the program will be loaded (data access)
    - Many programs do not need all of their code and data at once (or ever) – no need to allocate memory for it (memory management)

Virtual Addresses

- To make it easier to program and manage the memory, 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

Virtual Addresses

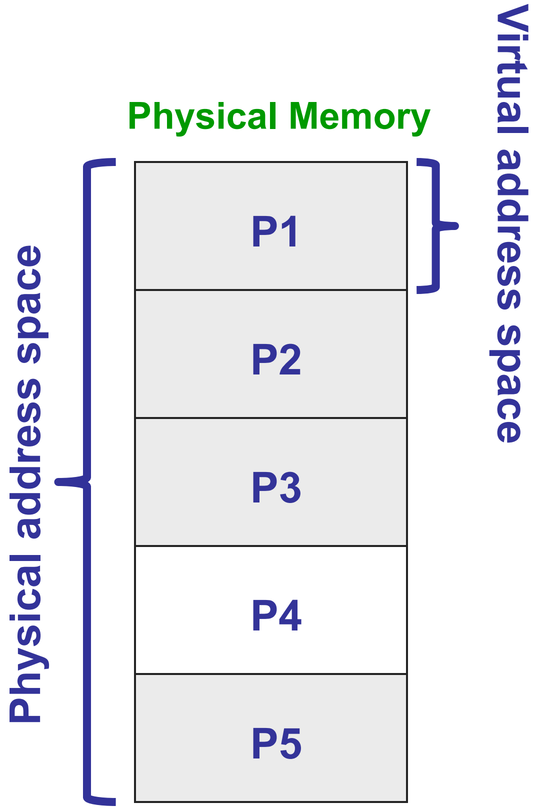
What is the virtualization/illusion we created?



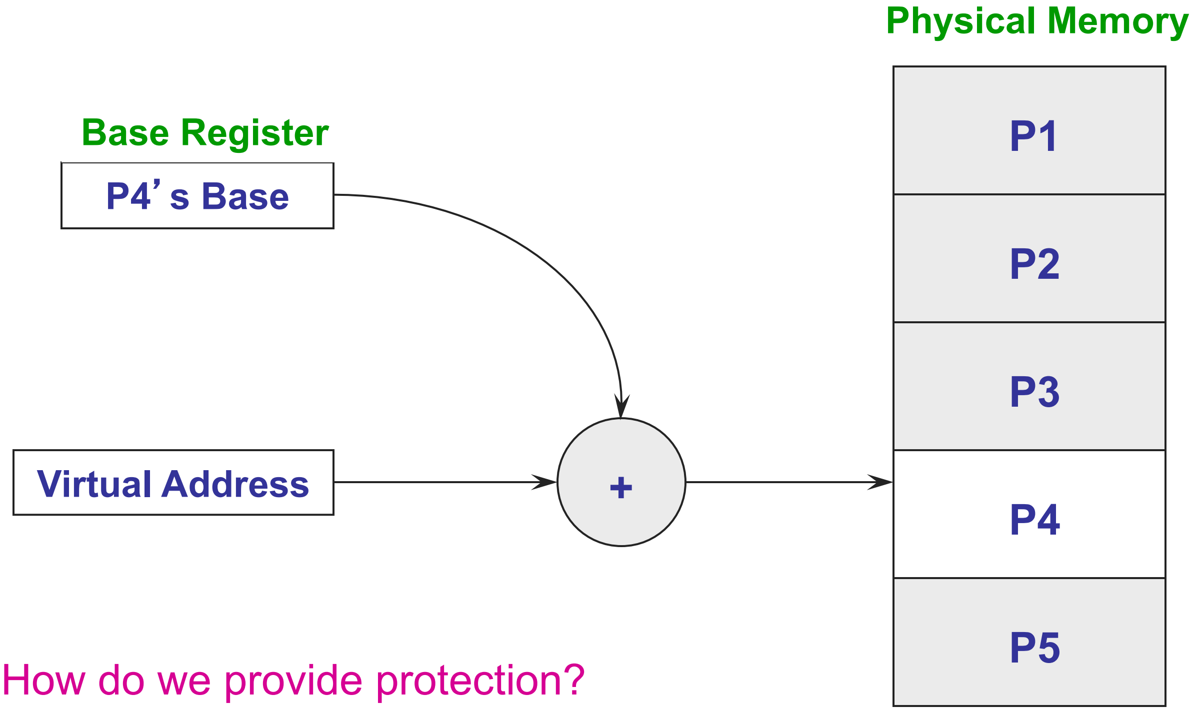
- Many ways to do this translation…
  - Need hardware support and OS management algorithms
- Requirements
  - Need protection – restrict which addresses jobs can use
  - Fast translation – lookups need to be fast
  - Fast change – updating memory hardware on context switch

Fixed Partitions

- Physical memory is broken up into fixed partitions
  - Size of each partition is the same and fixed
  - Hardware requirements: base register
  - Physical address = virtual address + base register
  - Base register loaded by OS when it switches to a process



Fixed Partitions



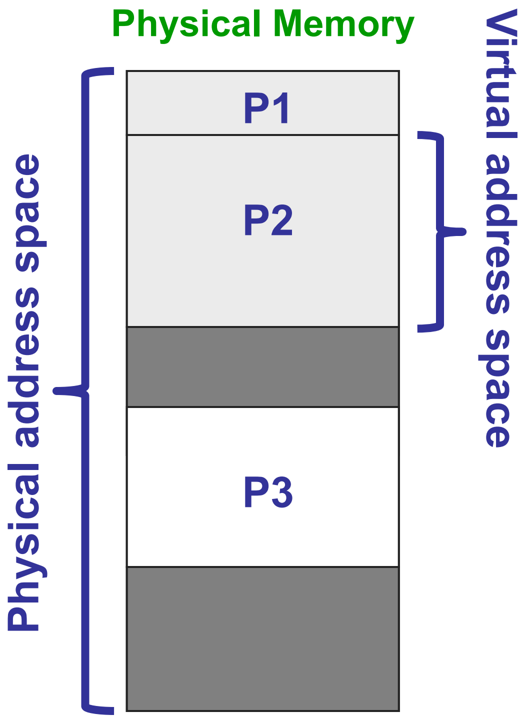
How do we provide protection?

Fixed Partitions

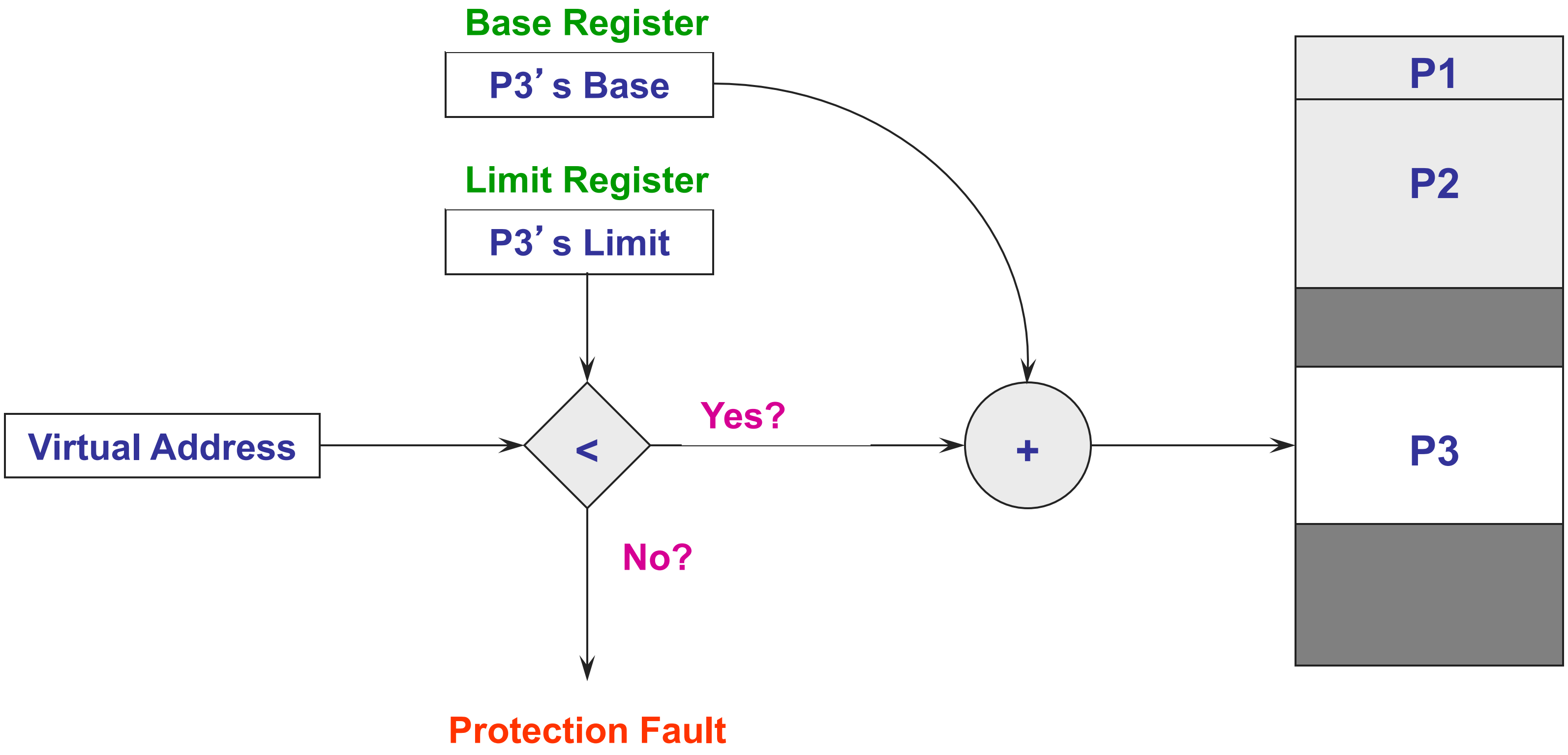
- Advantages
  - Easy to implement
    - Need base register
    - Verify that offset is less than fixed partition size
  - Fast context switch
- Problems?
  - Internal fragmentation: memory in a partition not used by a process is not available to other processes
  - Partition size: one size does not fit all (very large processes?)

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 (virtual address > limit) then fault

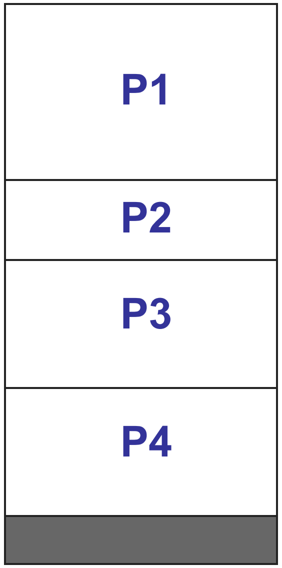


Variable Partitions



Variable Partitions

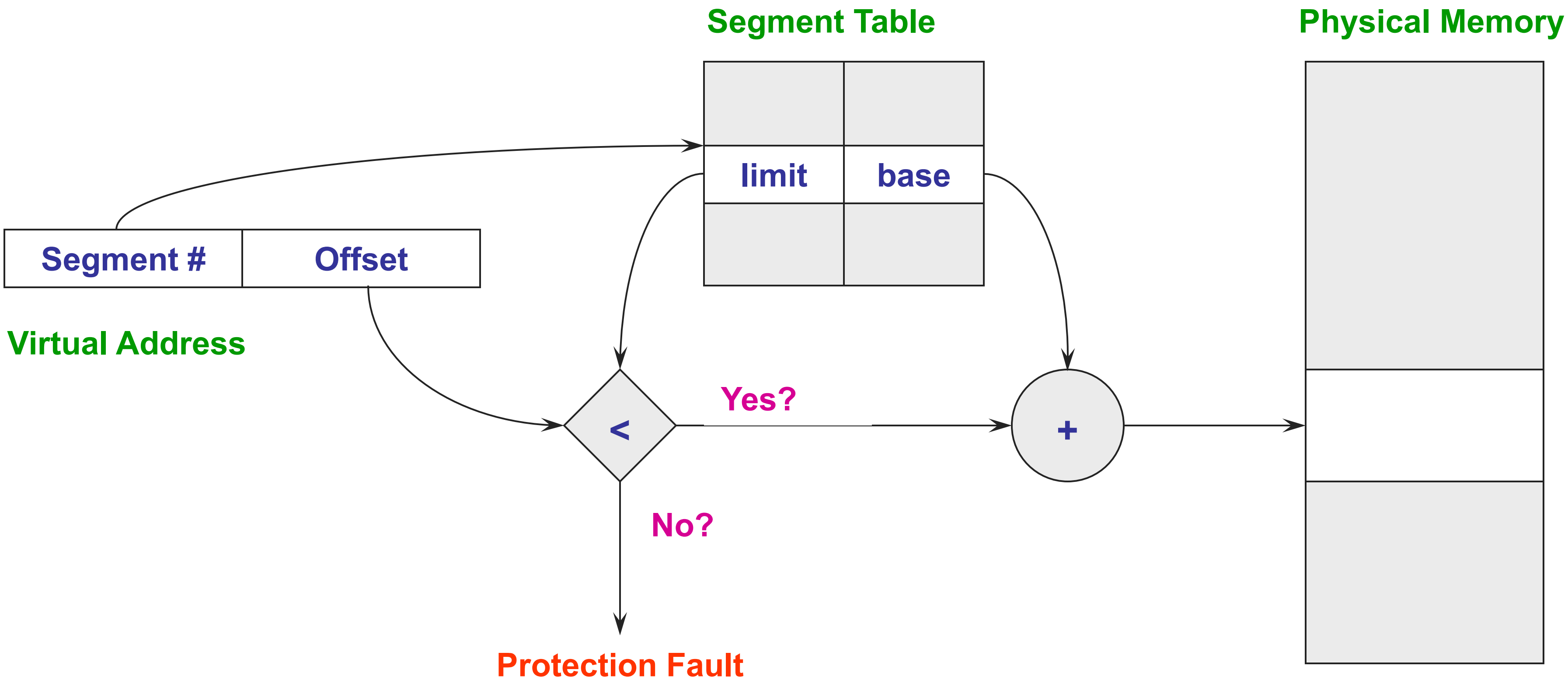
- Advantages
    - No internal fragmentation: allocate just enough for process
- Problems?
    - External fragmentation: job loading and unloading produces empty holes scattered throughout memory



Segmentation

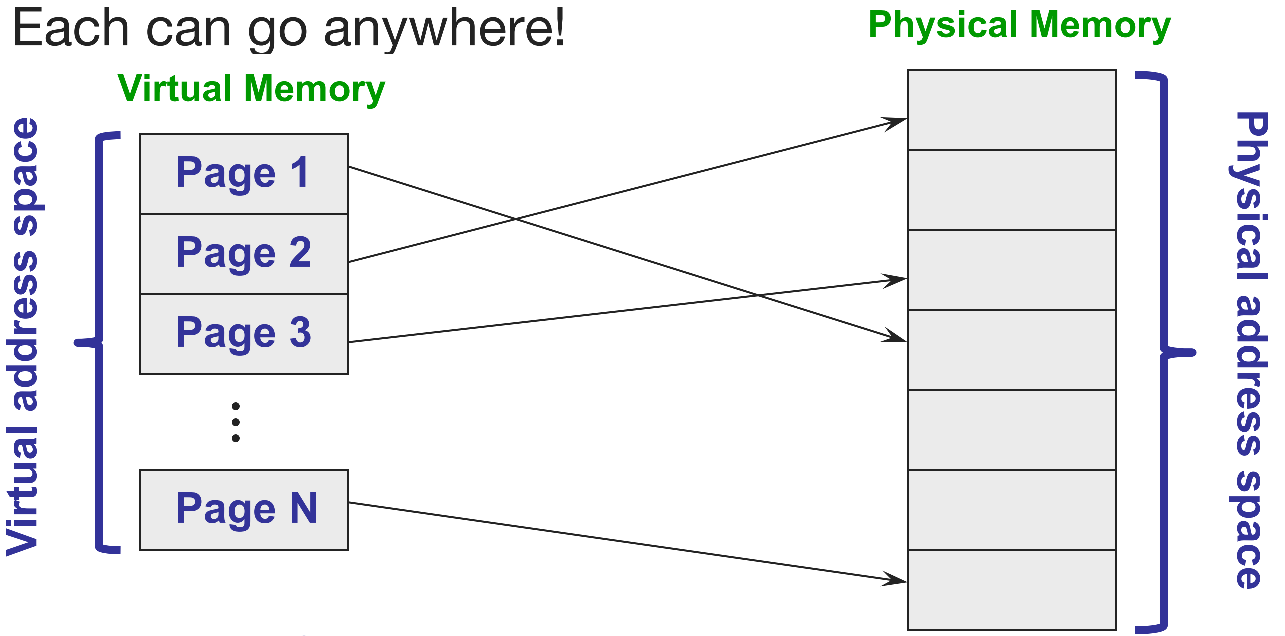
- Segmentation: partition memory into logically related units
  - Module, procedure, stack, data, file, etc.
  - Units of memory from programmer's perspective
- Natural extension of variable-sized partitions
  - Variable-sized partitions = 1 segment per process
  - Segmentation = many segments per process
- Hardware support
  - Multiple base/limit pairs, one per segment (segment table)
  - Segments named by #, used to index into table
  - Virtual addresses become <segment #, offset>
    - content = memory[segment#, offset]

Segment Lookups



Paging

- New Idea: split virtual address space into multiple fixed size partitions
  - Each can go anywhere!



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

But need to keep track of where things are!

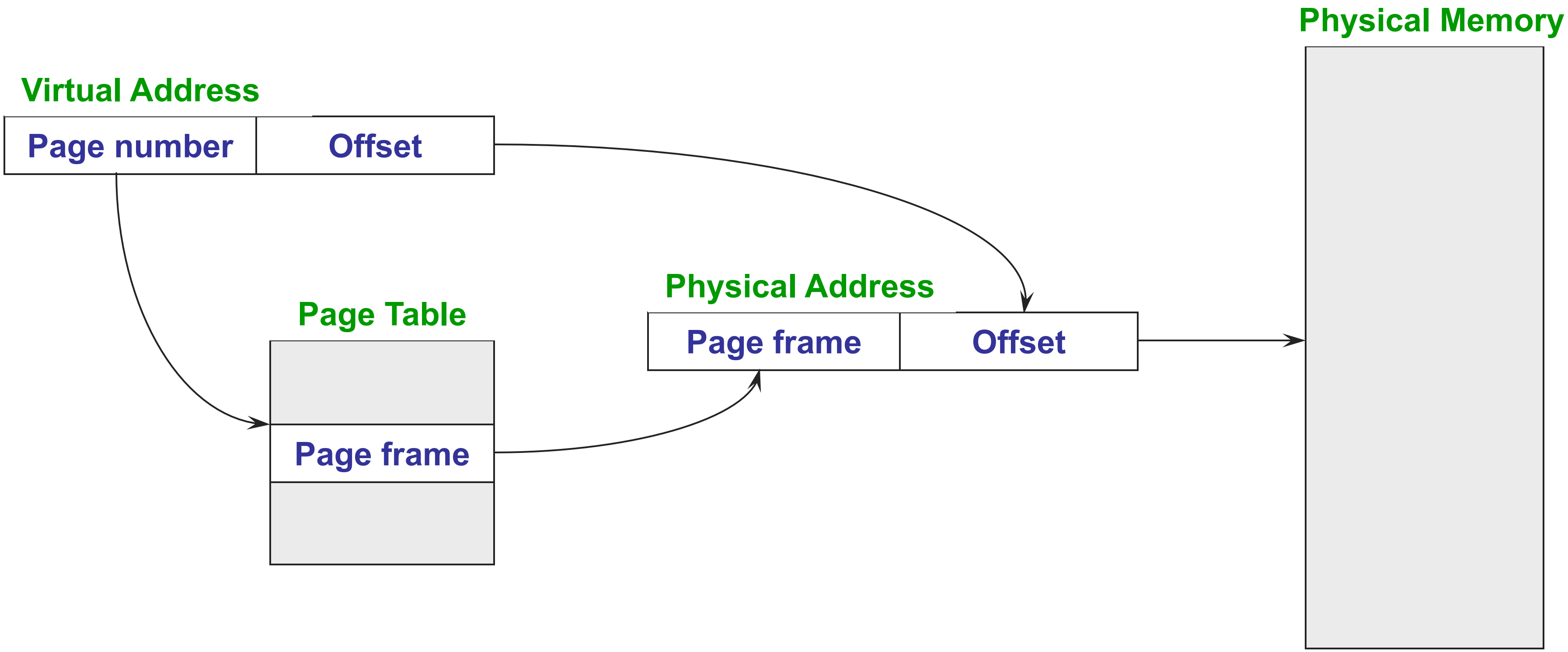
Process Perspective

- Processes view memory as one contiguous address space from 0 through N
    - Virtual address space (VAS)
- In reality, pages are scattered throughout physical storage
- The mapping is invisible to the program
- Protection is provided because a program 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
- Page tables
    - Map virtual page number (VPN) to page frame number (PFN)
        - VPN is the index into the table that determines PFN
    - One page table entry (PTE) per page in virtual address space
        - Or, one PTE per VPN

Page Lookups



Paging Example

- Pages are 4KB
  - Offset is 12 bits (because 4KB = 2^{12} bytes)
  - VPN is 20 bits (32 bits is the length of every virtual address)
- Virtual address is 0x7468
  - Virtual page is 0x7, offset is 0x468
- Page table entry 0x7 contains 0x2000
  - Page frame number is 0x2000
  - Seventh virtual page is at address 0x2000 (2nd physical page)
- Physical address = 0x2000 + 0x468 = 0x2468

Page Table Entries (PTEs)

1	1	1	2	20
M	**R**	**V**	**Prot**	**Page Frame Number**

- 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 (Why?)
  - The Protection bits say what operations are allowed on page
    - Read, write, execute (Why do we need these?)
  - The page frame number (PFN) determines physical page

Paging Advantages

- Easy to allocate memory
  - Memory comes from a free list of fixed size chunks
  - Allocating a page is just removing it from the list
  - External fragmentation not a problem
    - All pages of the same size
- Simplifies protection
  - All chunks are the same size
  - Like fixed partitions, don't need a limit register
- Simplifies virtual memory – later

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)
  - What can we do?
- 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!
  - What can we do?

Segmentation and Paging*

- Can combine segmentation and paging
  - The x86 supports both segments and paging
- Use segments to manage logically related units
  - Code, data, stack, thread-local storage, etc.
  - Segments vary in size, but usually large (multiple 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…

Summary

- Virtual address space
  - Developers use virtual address
  - Processes use virtual address
  - OS + hardware translate VA into PA
- Various techniques
  - Fixed partitions
  - Variable partitions
  - Segmentation
  - Paging