

CSE 120

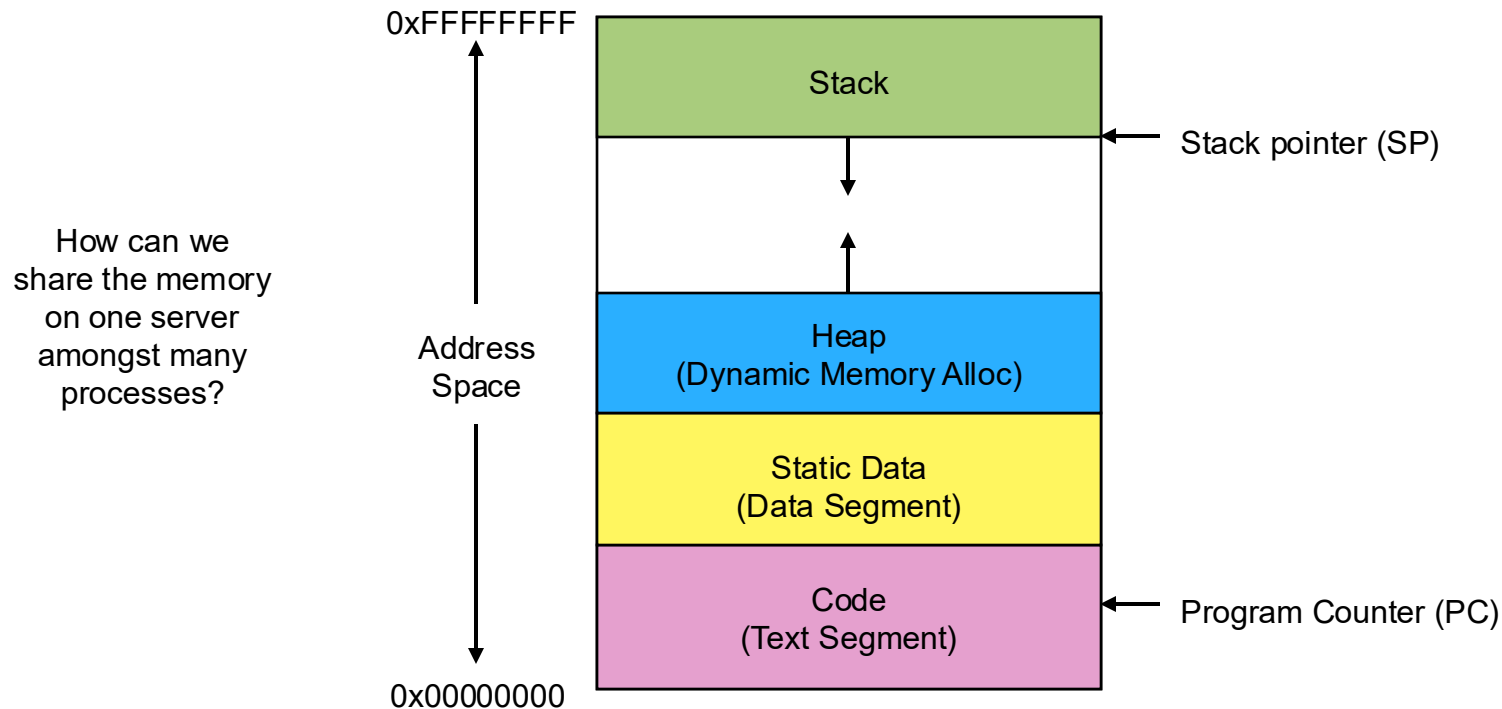
Operating Systems Principles

Spring 2025

Lecture 10: Memory Management Overview

Amy Ousterhout

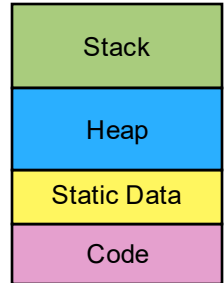
Basic Process Address Space



Memory Management Challenges

- Finite memory capacity
 - My process' data might not fit in physical memory
 - Might run many processes at once
- Locating data in memory
 - Where is each processes' data located in memory?
- Protection
 - Processes should not be able to read or write each others' memory
 - Processes should not be able to corrupt OS memory
- Efficiency
 - Should support many processes at once
 - Should keep CPU overheads low

No more room to
allocate memory!



Memory Management Overview

- New few lectures are going to cover memory management
- What **goals** are we trying to achieve?
 - Multitasking, transparency, isolation, and efficiency
- What **mechanisms** can we use to achieve those goals?
 - Physical and virtual addressing
 - Segmentation and paging
 - Page table management, TLBs, VM tricks
- What **policies**?
 - Page replacement algorithms

(Live Demo of Memory in Processes)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    if (argc < 2) {
        return 1;
    }

    // convert string to integer
    int value = atoi(argv[1]);

    for (int i = 0; i < 10; i++) {
        // print address and value
        printf("addr of value: %p. value: %d\n", &value, value);
        sleep(1);
    }

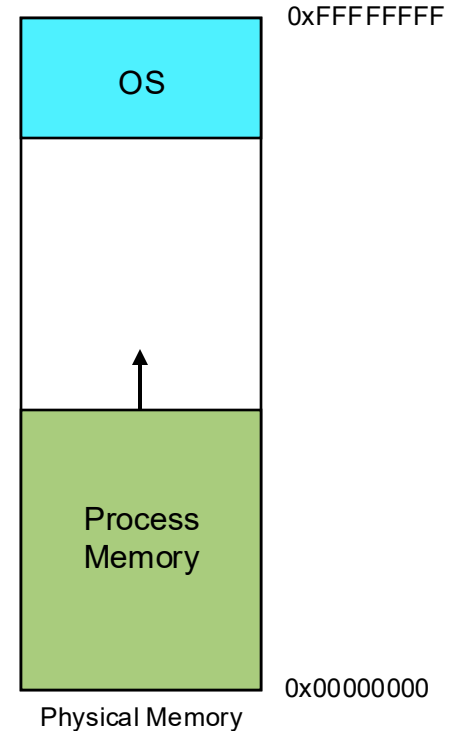
    return 0;
}
```

Today's Outline

- Virtual memory
 - The abstraction that the OS provides for managing memory
- Evolution of memory-management techniques
 - From fixed segments to paging today

Single-tasking

- In the early days: **run one process at a time**
 - OS loads a process, runs it, unloads it, loads the next process...
- Highest memory holds the OS
- Allocate process memory starting at 0
 - Includes code, data, stack, heap
- Programs **use physical addresses directly**
- What are the limitations of this approach?
 - Only supports one process at a time
 - The process can read or write to the OS's memory

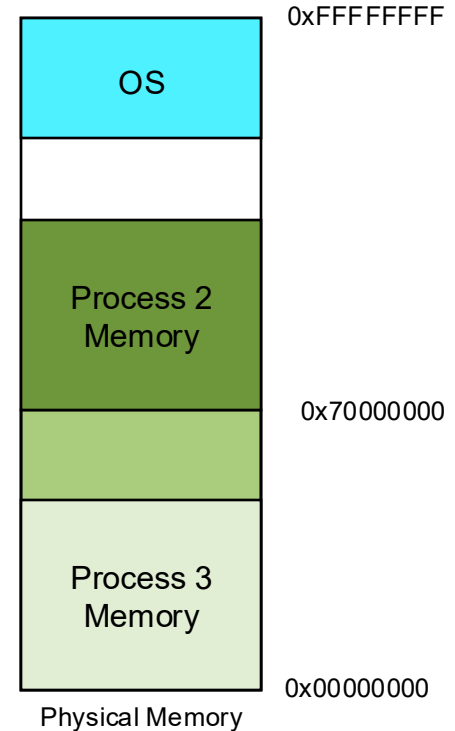


Memory Management Goals

- **Multitasking**
 - Allow multiple processes to be in memory at once
- **Transparency**
 - Convenient abstraction for programming
 - Processes should not know that memory is shared
 - Processes should run regardless of the number/locations of processes
- **Isolation/protection**
 - Processes shouldn't be able to corrupt each other (or the OS)
- **Efficiency**
 - CPU and memory utilization shouldn't be significantly degraded by sharing memory

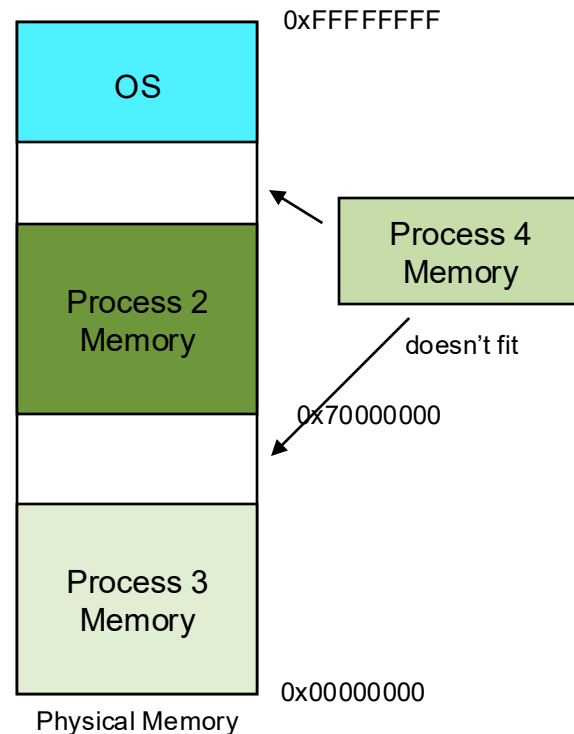
Multi-tasking with Static Relocation

- Support multiple processes by **relocating** once at load time
- Highest memory holds the OS
- When a process is loaded
 - Allocate a region of memory
 - Loader rewrites all memory addresses to **relocate** the process
- What are the drawbacks of this approach?



Multi-tasking with Static Relocation - Limitations

- No protection (between processes or of OS)
- Low memory utilization
 - Addresses are fixed after loading
 - Cannot relocate at runtime to fill holes
- No sharing
 - One segment per process
 - Cannot share parts of the process address space
- Entire address space needs to fit in memory



Dynamic Memory Relocation

- Change addresses **dynamically** as a process executes
- Virtual addresses
 - Processes use **virtual addresses** to refer to memory locations
 - These addresses are translated to **physical addresses** during every memory reference
 - Virtual addresses are independent of the physical location of the referenced data
- OS decides where to place data in physical memory
- Address translations
 - How can we translate from virtual to physical addresses?
 - Using hardware – the memory management unit (MMU)

Virtual Memory

- The abstraction that the OS provides for managing memory is **virtual memory**
- Two views of memory, called address spaces:
 - **Virtual address space** (seen by program)
 - **Physical address space** (actual allocation of memory)
- Virtual address space often much larger than physical address space
 - 64-bit addresses
- Benefits:
 - Flexible – OS can move processes around in memory as they execute
 - Transparent – hardware handles address translation
 - Protection – can check for isolation during address translation
 - Efficiency – use memory efficiently

Revisiting the Demo of Memory in Processes

- Each process has its own virtual address space

Process 1

```
$ ./a.out 47
```

```
addr of value: 0x7fff2cb61ad0. value: 47
```

```
addr of value: 0x7fff2cb61ad0. value: 47
```

```
addr of value: 0x7fff2cb61ad0. value: 47
```

Are these virtual or
physical addresses?

Process 2

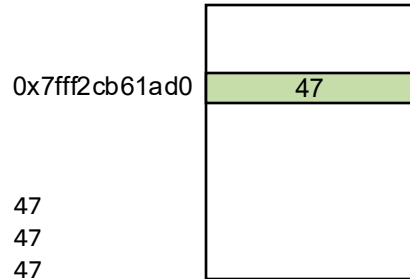
```
$ ./a.out 22
```

```
addr of value: 0x7fff2cb61ad0. value: 22
```

```
addr of value: 0x7fff2cb61ad0. value: 22
```

```
addr of value: 0x7fff2cb61ad0. value: 22
```

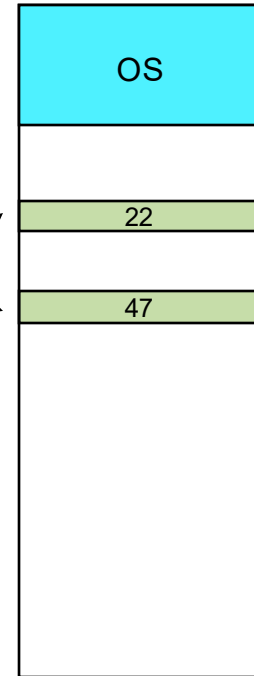
Process 1's Virtual Address Space



Process 2's Virtual Address Space



Physical Address Space

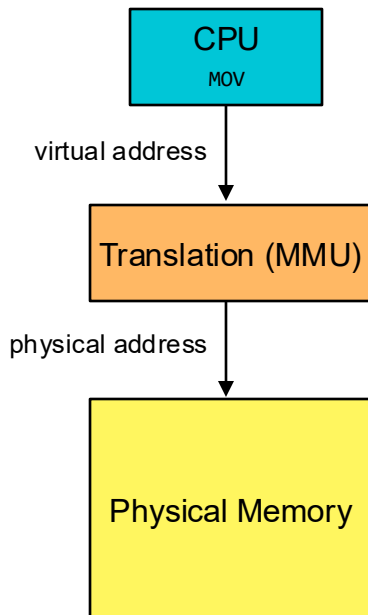


0xFFFFFFFF

0x00000000

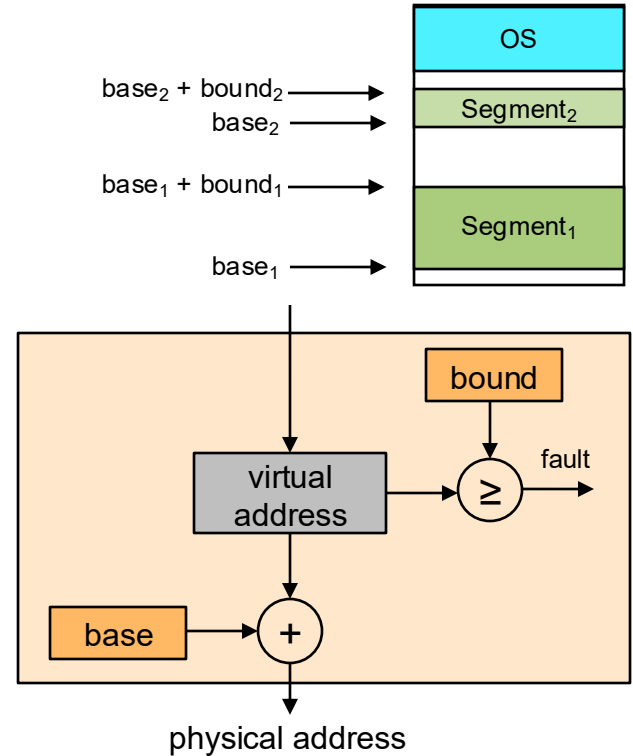
Address Translation

- Translation table set up by the OS in software
 - Mappings from virtual to physical addresses
- Dedicated hardware performs the address translations
 - Memory management unit (MMU)
 - Translates for each load and store
- Many ways to do address translations:
 - Base and bound
 - Segmentation
 - Paging



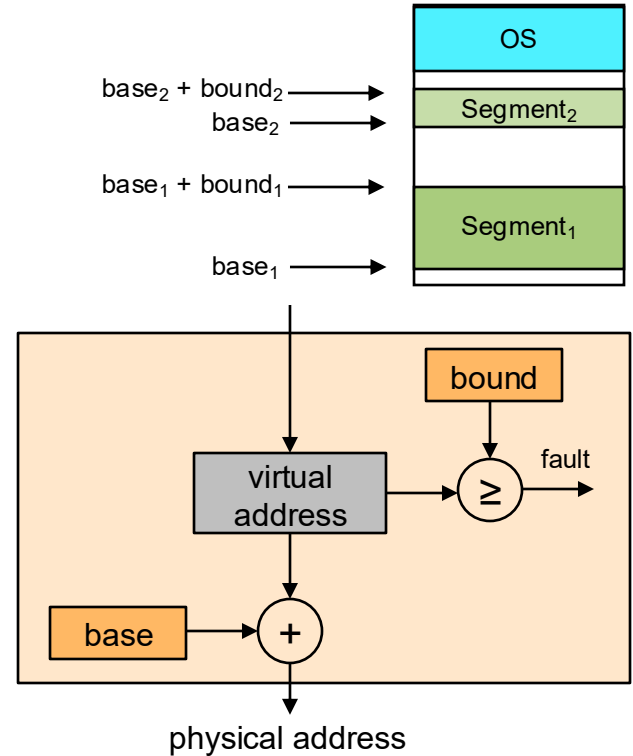
Base and Bound

- 2 hardware registers: base and bound
- A process can only access physical memory in $[base, base+bound)$
- On a context switch:
 - Save/restore base and bound registers
- Built in Cray-1 (1976)
- Benefits:
 - Simple, fast translation, cheap
 - Can relocate a segment at execution time
 - Bound register provides protection



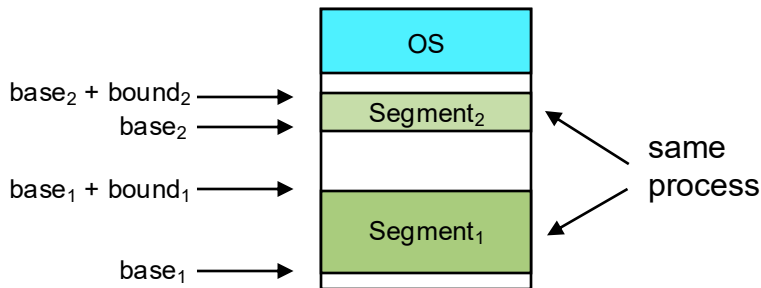
Base and Bound - Limitations

- **Fragmentation** – it's hard to use memory efficiently
 - **External fragmentation**: wasted memory between segments
 - **Internal fragmentation**: wasted memory within a segment
- Cannot share memory between processes



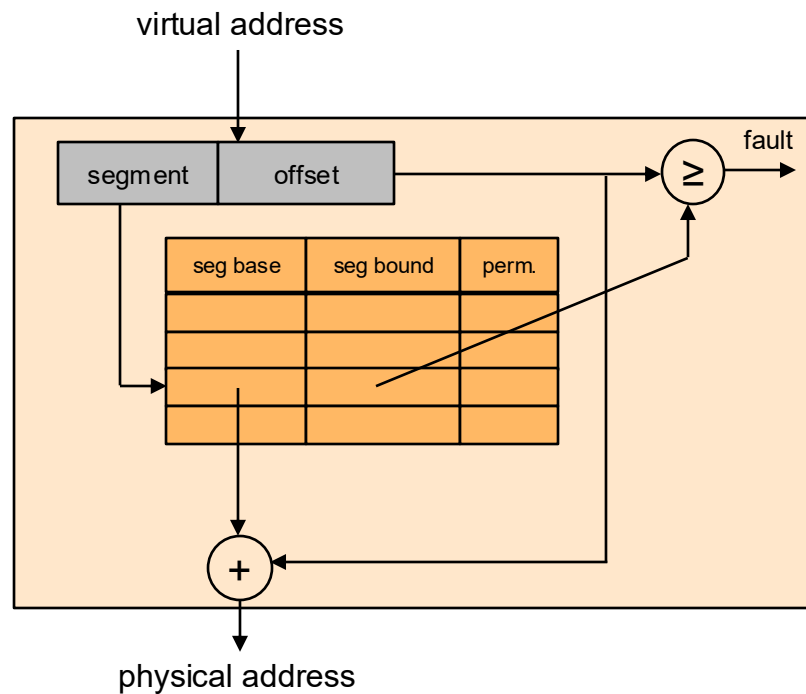
Segmentation

- Split each process' virtual address space into multiple segments
 - **Segment**: variable-sized area of memory
- Natural extension of base and bound
 - Base and bound: 1 segment per process
 - Segmentation: many segments per process



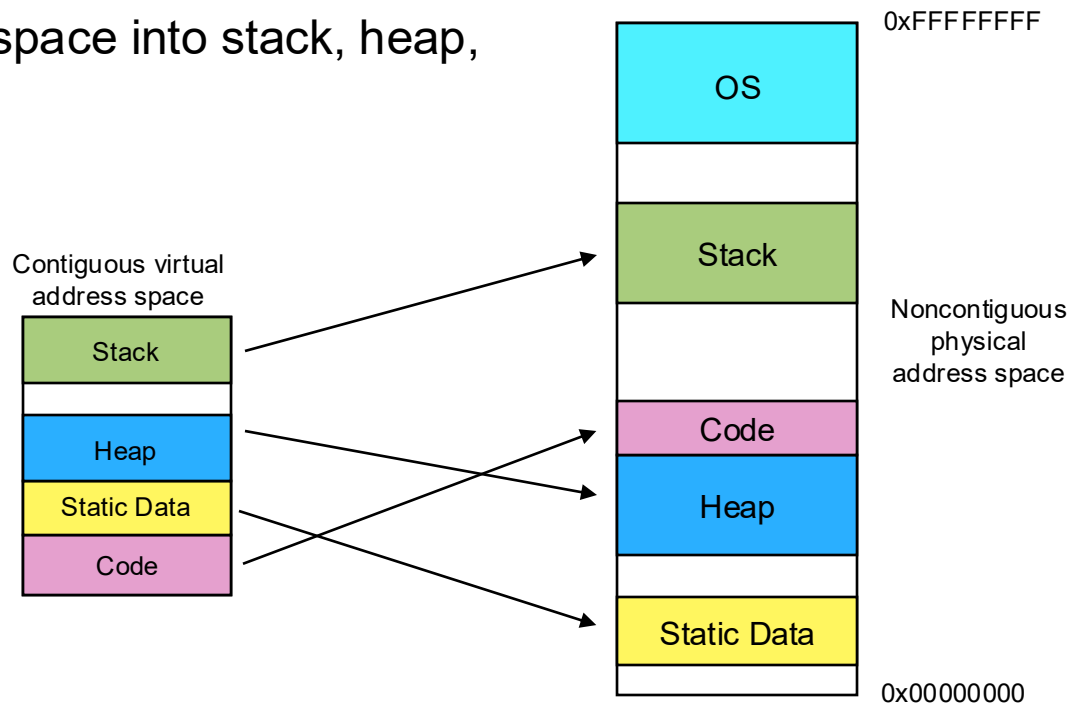
Segmentation – Address Translation

- Segment map
 - One per process
 - Holds base and bounds registers for each segment
 - Also per-segment permissions
- Context switch
 - Save/restore table or pointer to table in kernel memory



Segmentation Example

- Split process's virtual address space into stack, heap, static data, code

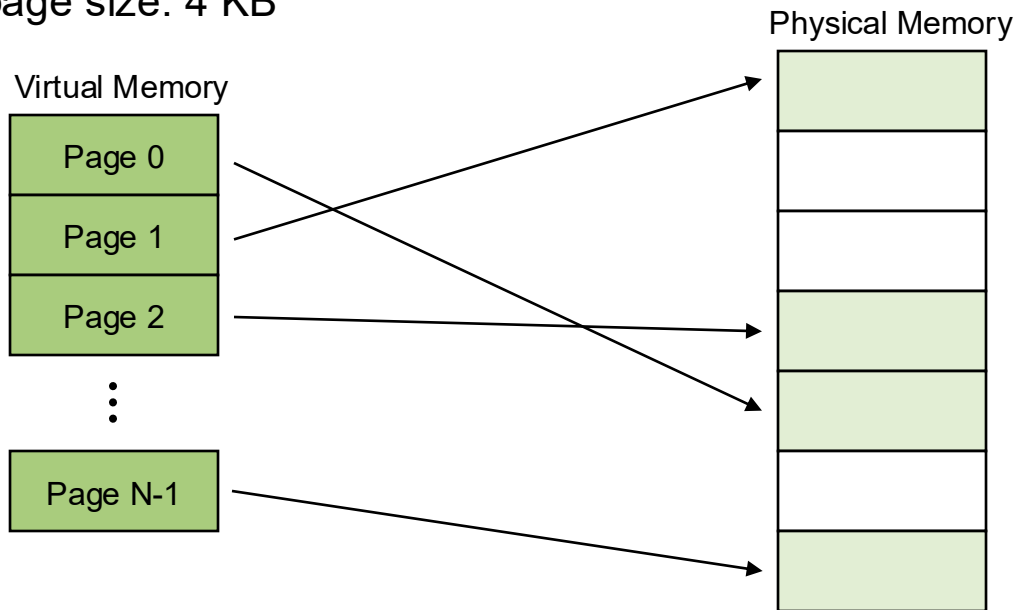


Segmentation - Tradeoffs

- Benefits:
 - Process memory can be split among several segments
 - » Allows sharing of some segments between processes
 - Flexible: segments can be assigned, moved, grown, shrunk, or swapped independently
- Limitations:
 - Fragmentation
 - » **External fragmentation** – can still have holes in physical memory due to different sized segments
 - » **Internal fragmentation** – each segment can be large and have regions that are unused
 - Large segment tables can be complex to manage

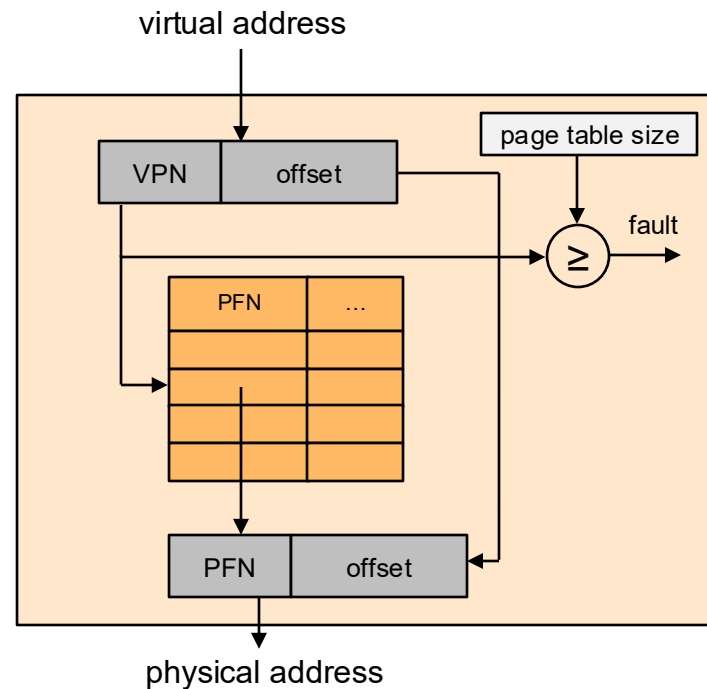
Paging

- Divide physical and virtual memory into fixed-sized chunks called **pages**
 - Common page size: 4 KB



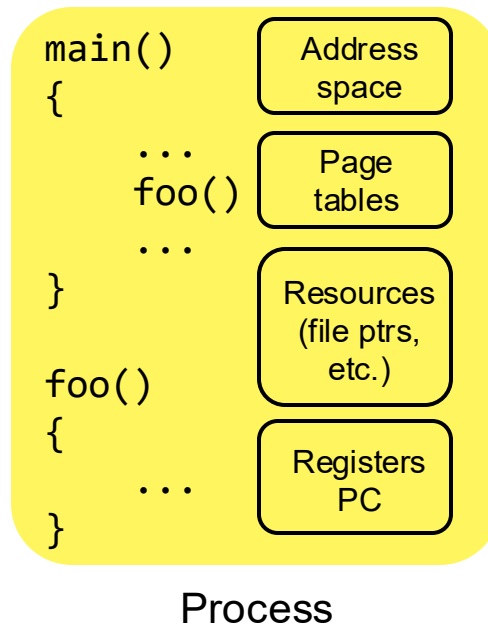
Paging – Address Translation

- Virtual addresses
 - Two parts: **virtual page number (VPN)** and **offset**
- **Page tables**
 - Map virtual page number to **page frame number (PFN)**
 - » The physical page number
 - Permissions, etc.
 - Use the VPN as an index into the page table
- Use concatenation instead of addition
 - Possible due to fixed power-of-2-sized pages



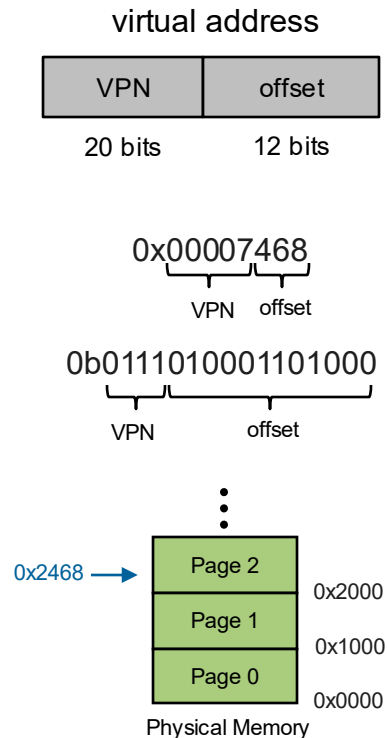
Page Tables

- Page tables are per process
 - Each process maintains its own page tables
 - On a context switch:
 - » Switch which page tables are used for translation



Address Translation Example

- Pages are 4 KB
 - Offset is 12 bits ($2^{12} = 4096 = 0x1000$)
 - Assume 32 bit system
 - Leaves 20 bits for virtual page number (2^{20} VPNs)
- Example virtual address: 0x00007468
 - Offset is 0x468 (lowest 12 bits of address)
 - Virtual page is 0x7
- Suppose page table entry 0x7 contains 0x2
 - Virtual page 0x7 is located in physical page 0x2
 - Physical page 0x2 is at address 0x2000
 - Physical address = $0x2000 + 0x468 = 0x2468$



Paging - Advantages

- Easy to allocate memory
 - Memory comes from a free list of fixed-size chunks (pages)
 - Allocating a page is just removing it from the list
 - External fragmentation is 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

Paging - Limitations

- Can still have internal fragmentation
 - Process may not use memory in multiples of pages
- Memory reference overhead
 - 2 references per address lookup (page table, then memory)
 - Solution: use a hardware cache of lookups (more later)
- May need a lot of memory to hold the page tables
 - Need one page table entry per page
 - 32-bit address space with 4 KB pages = 2^{20} page table entries
 - 4 bytes per entry = 4 MB per page table
 - 25 processes = 100 MB just for page tables
 - Solution: hierarchical pages tables (more later)

Academic Integrity and Getting Help

- Academic integrity in CSE 120
 - You (and your teammates) must implement the projects yourselves
 - Copying code from others or the internet is not permitted
 - Using generative AI or LLMs is not permitted
- The tutors and TAs are here to help!
 - Visit lab hours in person
 - Attend lab hours remotely via Zoom
- Attend office hours

Project 2

- Project 2 released
 - Focuses on user-level processes
- Getting started
 - You have already learned enough to do tasks 0 and 1
 - Do the VM worksheet (HW 3) before starting on task 2



- Code reviews (task 4)
 - Will occur in week 8 (after project 2 submission)
 - Meet with a TA and answer basic questions about your submission
 - Designed to check your understanding of your code

Upcoming Tasks

- Read chapters 18-20
- Project 2
 - Due Friday 5/16 at 11:59 pm



- Homework 3
 - Due Tuesday 5/20 at 11:59 pm