MEMORY VIRTUALIZATION

Shivaram Venkataraman

CS 537, Fall 2024

ADMINISTRIVIA

- Project 2 is due Sept 24th Tuesday
- Project 1 grading in progress (soon?)

- Midterm1: Oct 15th at 5.45pm
- Conflict form

AGENDA / LEARNING OUTCOMES

- Memory virtualization
  - What are main techniques to virtualize memory?
  - What are their benefits and shortcomings?

RECAP

MEMORY VIRTUALIZATION

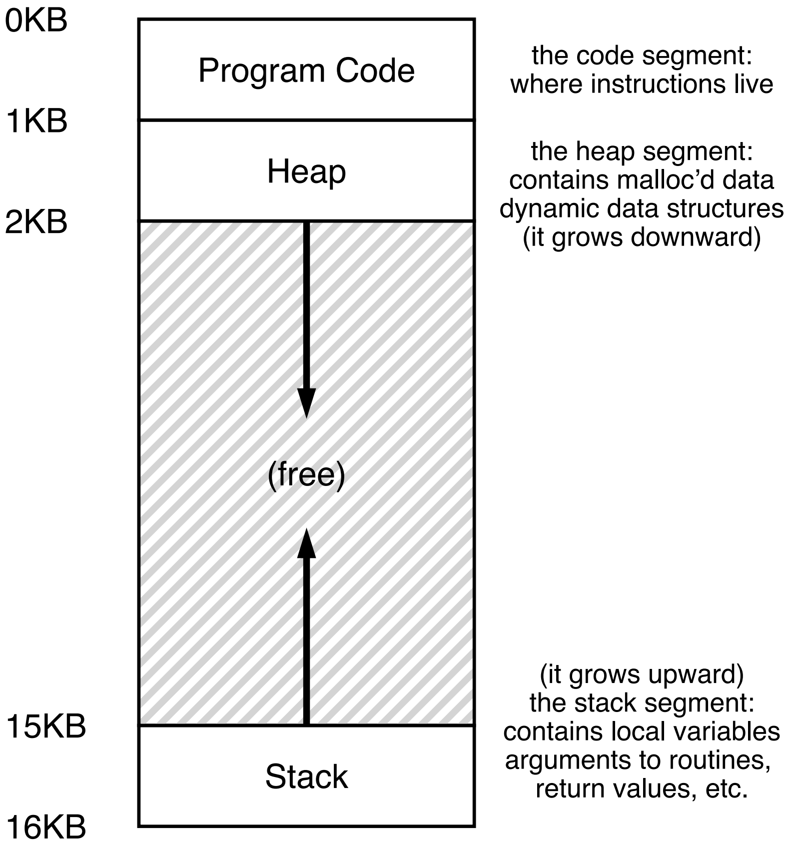
Transparency: Process is unaware of sharing

Protection: Cannot corrupt OS or other process memory

Efficiency: Do not waste memory or slow down processes

Sharing: Enable sharing between cooperating processes

RECAP: WHAT IS IN ADDRESS SPACE?



Static: Code and some global variables

Dynamic: Stack and Heap

HOW TO VIRTUALIZE MEMORY

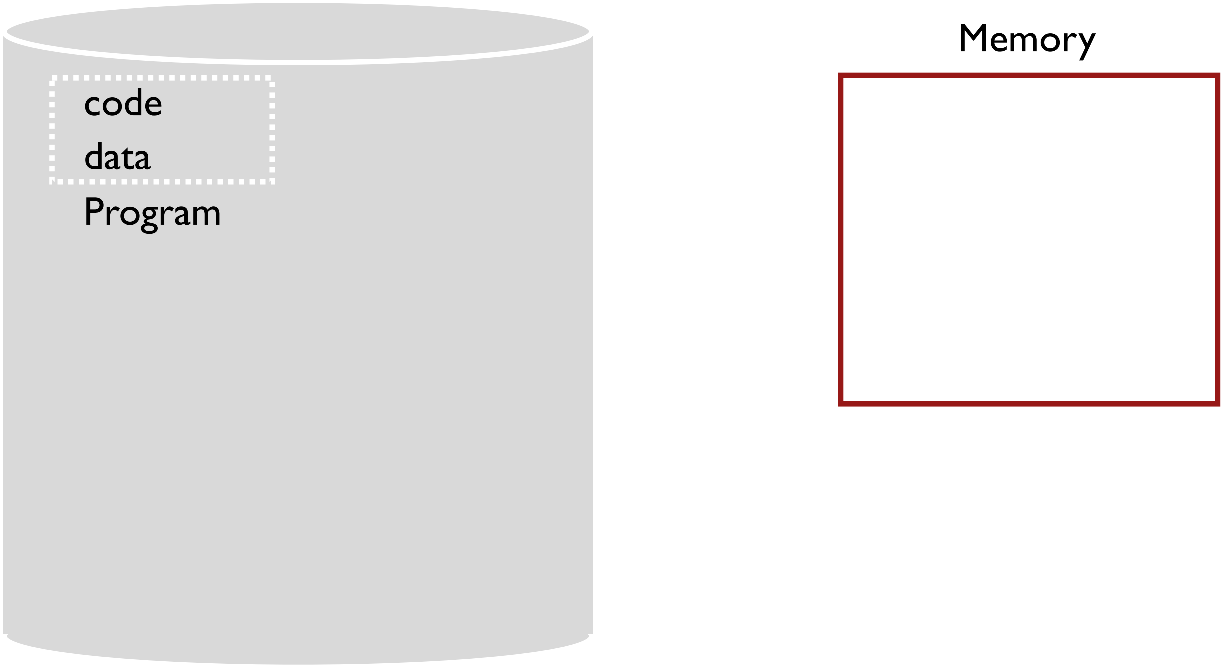
Problem: How to run multiple processes simultaneously?

Addresses are "hardcoded" into process binaries

How to avoid collisions?

Possible Solutions for Mechanisms:

1. Time Sharing
2. Static Relocation
3. Base
4. Base+Bounds

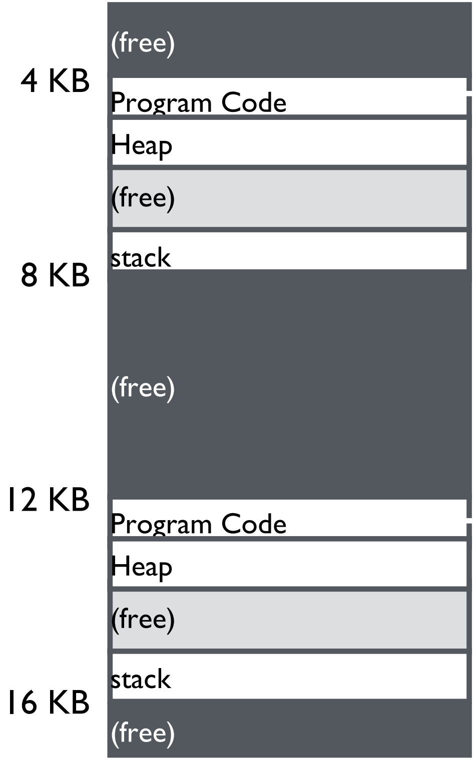


1) TIME SHARING MEMORY

2) STATIC: LAYOUT IN MEMORY

process 1

process 2




```
0x1010: movl  0x8(%rbp), %edi
0x1013: addl  $0x3, %edi
0x1019: movl  %edi, 0x8(%rbp)
```

```
0x3010: movl  0x8(%rbp), %edi
0x3013: addl  $0x3, %edi
0x3019: movl  %edi, 0x8(%rbp)
```

3) DYNAMIC RELOCATION

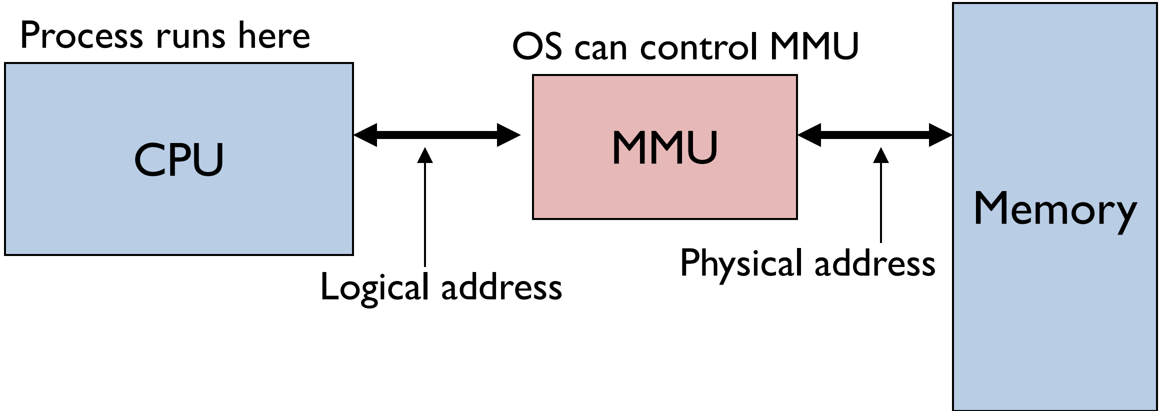
Goal: Protect processes from one another

Requires hardware support

- Memory Management Unit (MMU)

MMU dynamically changes process address at every memory reference

- Process generates logical or virtual addresses (in their address space)
- Memory hardware uses physical or real addresses



HARDWARE SUPPORT FOR DYNAMIC RELOCATION

Privileged (protected, kernel) mode: OS runs

- When enter OS (trap, system calls, interrupts, exceptions)
- Allows certain instructions to be executed
  (Can manipulate contents of MMU)
- Allows OS to access all of physical memory

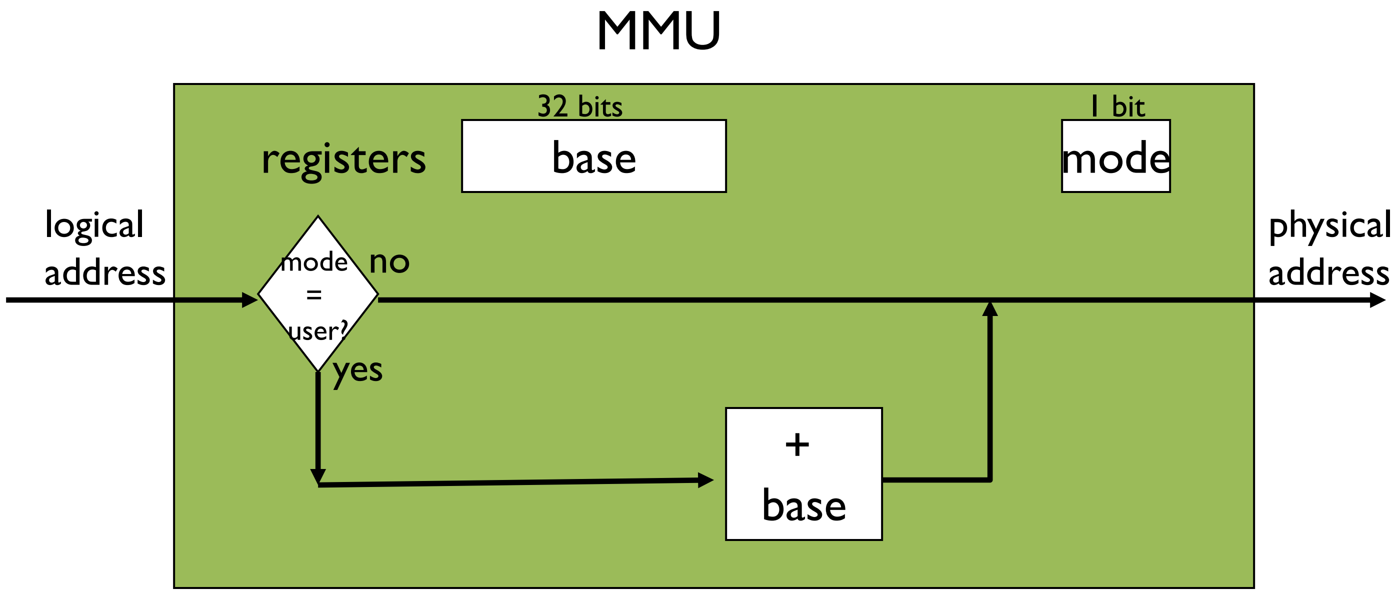
User mode: User processes run

- Perform translation of logical address to physical address

IMPLEMENTATION OF DYNAMIC RELOCATION: BASE REG

Translation on every memory access of user process

MMU adds base register to logical address to form physical address



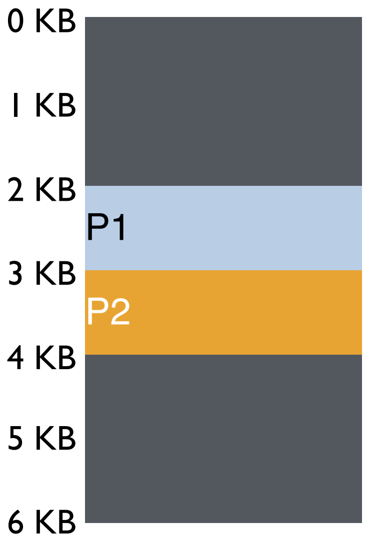
DYNAMIC RELOCATION WITH BASE REGISTER

Translate virtual addresses to physical by adding a fixed offset each time.

- Store offset in base register

Each process has different value in base register

- Dynamic relocation by changing value of base register!



VISUAL EXAMPLE OF DYNAMIC RELOCATION: BASE REGISTER

Base Register for P1 = 2048

Base Register for P2 = 3072

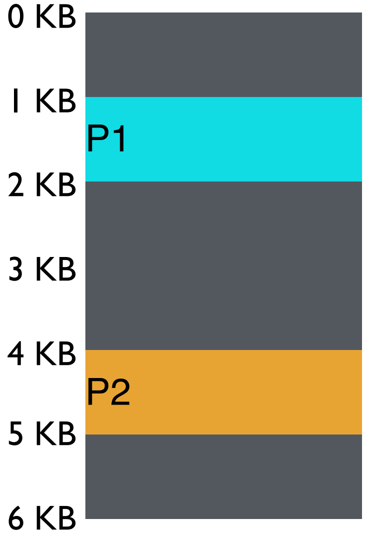
Virtual

P1: load 10, R1

P1: load 200, R1

P2: load 500, R1

Physical



Virtual	Physical
P1: load 100, R1	load 1124, R1
P2: load 100, R1	load 4196, R1
P2: load 1000, R1	load 5196, R1
P1: load 100, R1	load 2024, R1
P1: store 3072, R1	

Can P1 hurt P2?

4) DYNAMIC WITH BASE+BOUNDS

Idea: limit the address space with a bounds register

Base register: smallest physical addr (or starting location)

Bounds register: size of this process's virtual address space

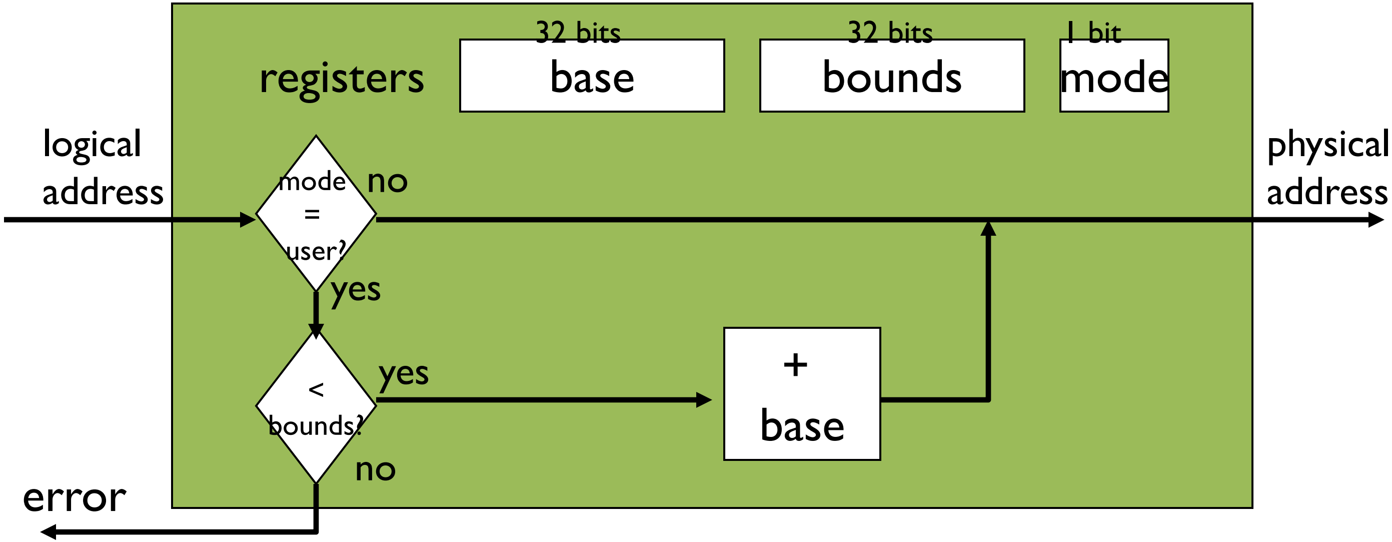
- Sometimes defined as largest physical address (base + size)

OS kills process if process loads/stores beyond bounds

IMPLEMENTATION OF BASE+BOUNDS

Translation on every memory access of user process

- MMU compares logical address to bounds register
  - if logical address is greater, then generate error
- MMU adds base register to logical address to form physical address



MANAGING PROCESSES WITH BASE AND BOUNDS

Context-switch: Add base and bounds registers to proc struct

Steps

- Change to privileged mode
- Save base and bounds registers of old process
- Load base and bounds registers of new process
- Change to user mode and jump to new process

Protection requirement

- User process cannot change base and bounds registers
- User process cannot change to privileged mode

BASE AND BOUNDS

Advantages

- Provides protection (both read and write) across address spaces
- Supports dynamic relocation
  - Can place process at different locations initially and move address spaces

- Simple, inexpensive implementation: Few registers, little logic in MMU

Disadvantages

- Each process must be allocated contiguously in physical memory
- Must allocate memory that may not be used by process
- No partial sharing: Cannot share parts of address space

QUIZ 4

https://tinyurl.com/cs537-fa24-q4

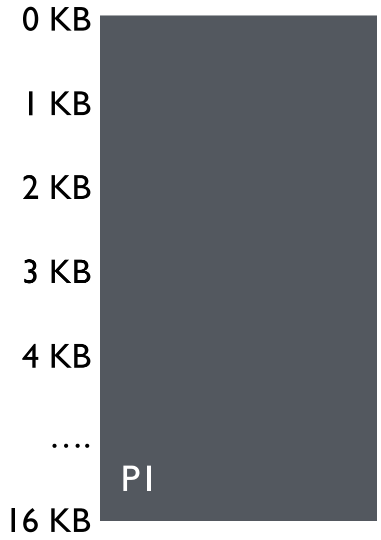
```
unsigned long A = 3;
int main(int argc, char *argv[]) {
    int B = 7;
    short *P = malloc(5 * sizeof(short));
}
```

QUIZ 4

https://tinyurl.com/cs537-fa24-q4

Address space size 1K
Physical memory of size 16K

Base: 0x00003cb5 (decimal 15541)
Limit: 492

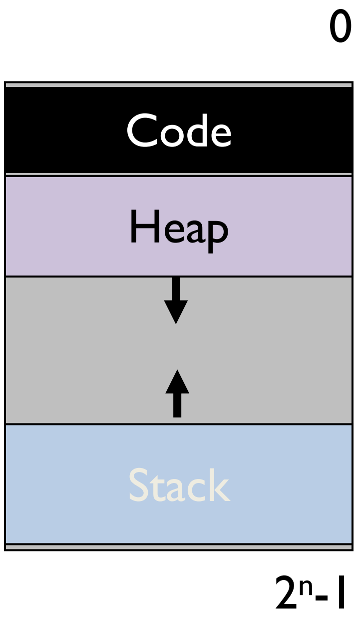


5) SEGMENTATION

Divide address space into logical segments

- Each segment corresponds to logical entity in address space (code, stack, heap)

Each segment has separate base + bounds register



SEGMENTED ADDRESSING

Process now specifies segment and offset within segment

How does process designate a particular segment?

- Use part of logical address
  - Top bits of logical address select segment
  - Low bits of logical address select offset within segment

What if small address space, not enough bits?

- Implicitly by type of memory reference
- Special registers

SEGMENTATION IMPLEMENTATION

MMU contains Segment Table (per process)

- Each segment has own base and bounds, protection bits
- Example: 14 bit logical address, 4 segments;

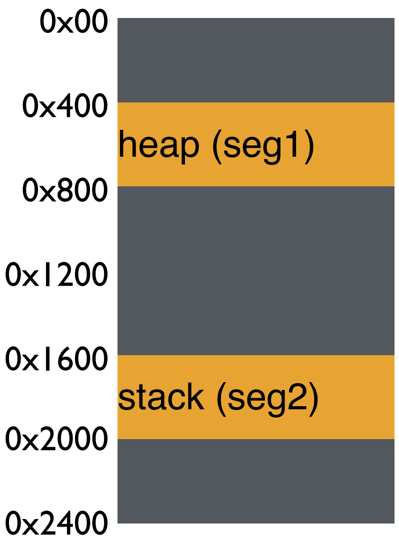
How many bits for segment?

How many bits for offset?

Segment	Base	Bounds	R W
0	0x2000	0x6ff	1 0
1	0x0000	0x4ff	1 1
2	0x3000	0xfff	1 1
3	0x0000	0x000	0 0

remember:
1 hex digit → 4 bits

VISUAL INTERPRETATION



Virtual (hex)	Physical
load 0x2010, R1	
load 0x1010, R1	
load 0x1100, R1	

Segment numbers:

0: code+data

1: heap

2: stack

Virtual	Physical
load 0x2010, R1	0x1600 + 0x010 = 0x1610
load 0x1010, R1	0x400 + 0x010 = 0x410
load 0x1100, R1	0x400 + 0x100 = 0x500

Segment numbers:

0: code+data
1: heap
2: stack

HOW DOES THIS LOOK IN X86

Stack Segment (SS): Pointer to the stack

Code Segment (CS): Pointer to the code

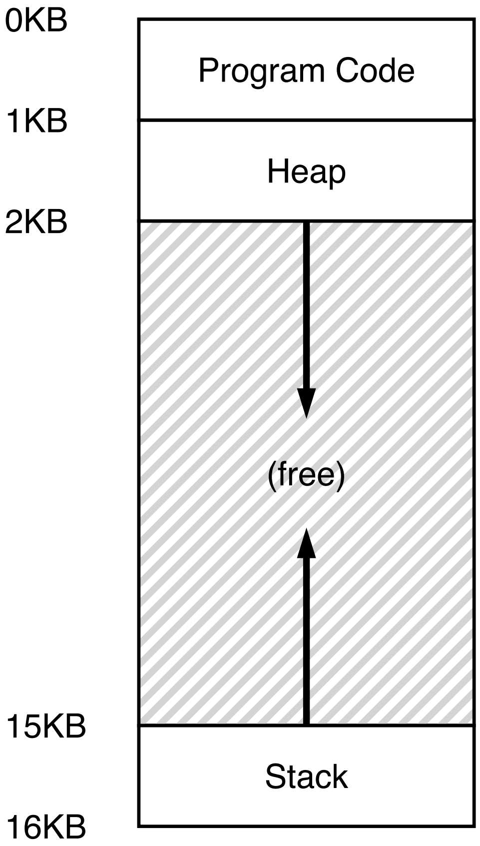
Data Segment (DS): Pointer to the data

Extra Segment (ES): Pointer to extra data

F Segment (FS): Pointer to more extra data

G Segment (GS): Pointer to still more extra data

NOTE: HOW DO STACKS GROW ?



Stack goes 16K → 12K, in physical memory is 28K → 24K

Segment base is at 28K

Virtual address 0x3C00 = 15K

→ top 2 bits (0x3) segment ref, offset is 0xC00 = 3K

How do we make CPU translate that ?

Negative offset = subtract max segment from offset

= 3K – 4K = -1K

Add to base = 28K – 1K = 27K

ADVANTAGES OF SEGMENTATION

Enables sparse allocation of address space

Stack and heap can grow independently

- Heap: If no data on free list, dynamic memory allocator requests more from OS (e.g., UNIX: malloc calls sbrk())
- Stack: OS recognizes reference outside legal segment, extends stack implicitly

Different protection for different segments

- Enables sharing of selected segments
- Read-only status for code

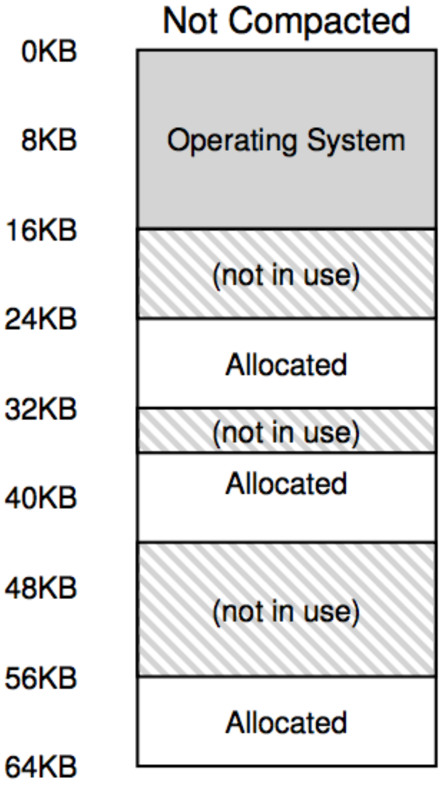
Supports dynamic relocation of each segment

DISADVANTAGES OF SEGMENTATION

Each segment must be allocated contiguously

May not have sufficient physical memory for large segments?

External Fragmentation



NEXT STEPS

Project 2: Due soon!

Next class: Paging, TLBs and more!