# {Dynamic, Virtual} Memory

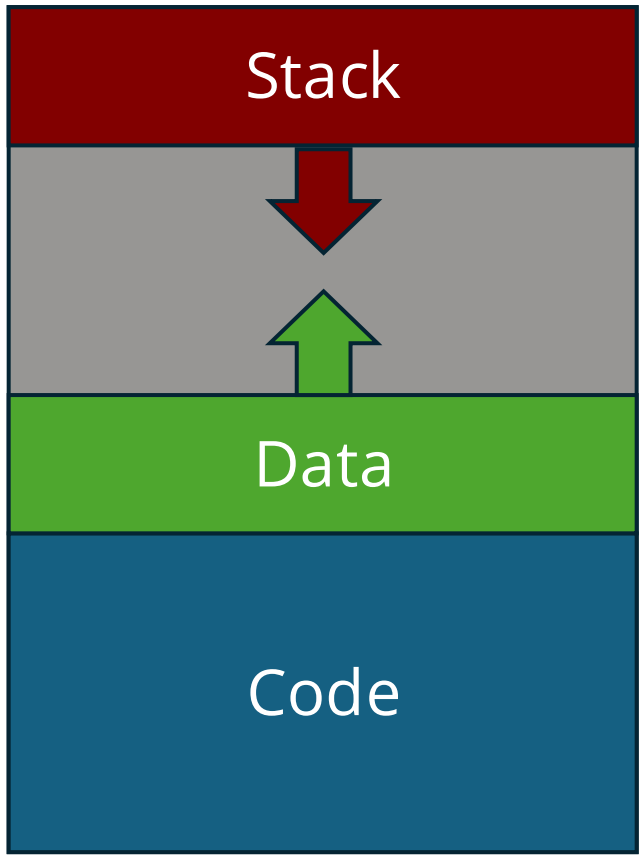
**Problem Solving Lab for CS111**

**Stanford CS111ACE, Spring 2026**

# Today

- Dynamic Memory
- Goals of Virtual Memory
- Dynamic Address Translation
  - Base and Bound
  - Multiple Segments
  - Paging

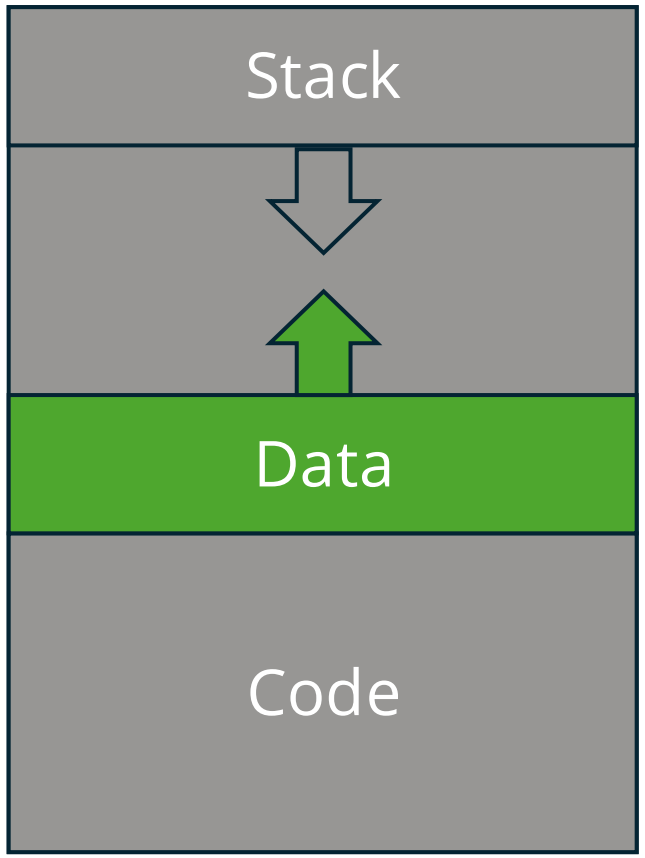
# Dynamic Memory




The **stack** contains stack frames

The **Data** section contains the heap and constants

The **Code** section, otherwise known as .text contains the program code




The **Data** section contains the ***heap*** and constants

The heap is where we do dynamic allocations

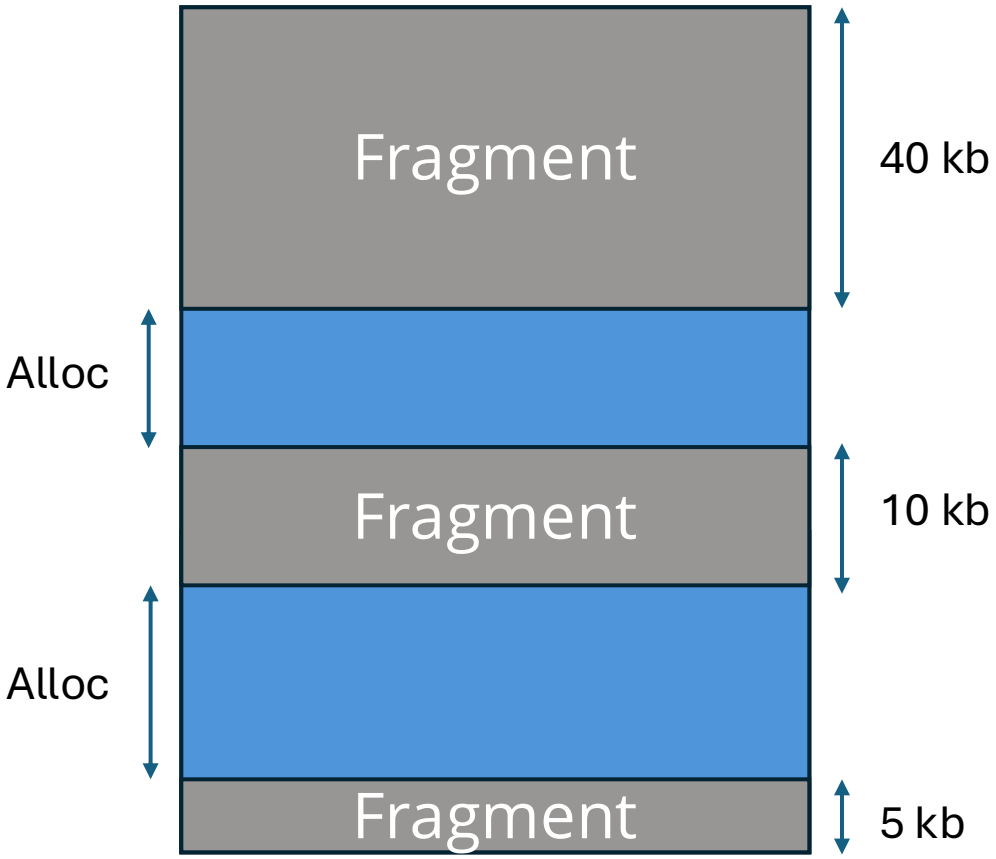
# What is a dynamic allocation?

# Dynamic Allocation

- We want to use memory outside of the scope in which it is defined. I will show you an example.

- We want to have control over the allocation, for example the size, etc.

# How do we do it?

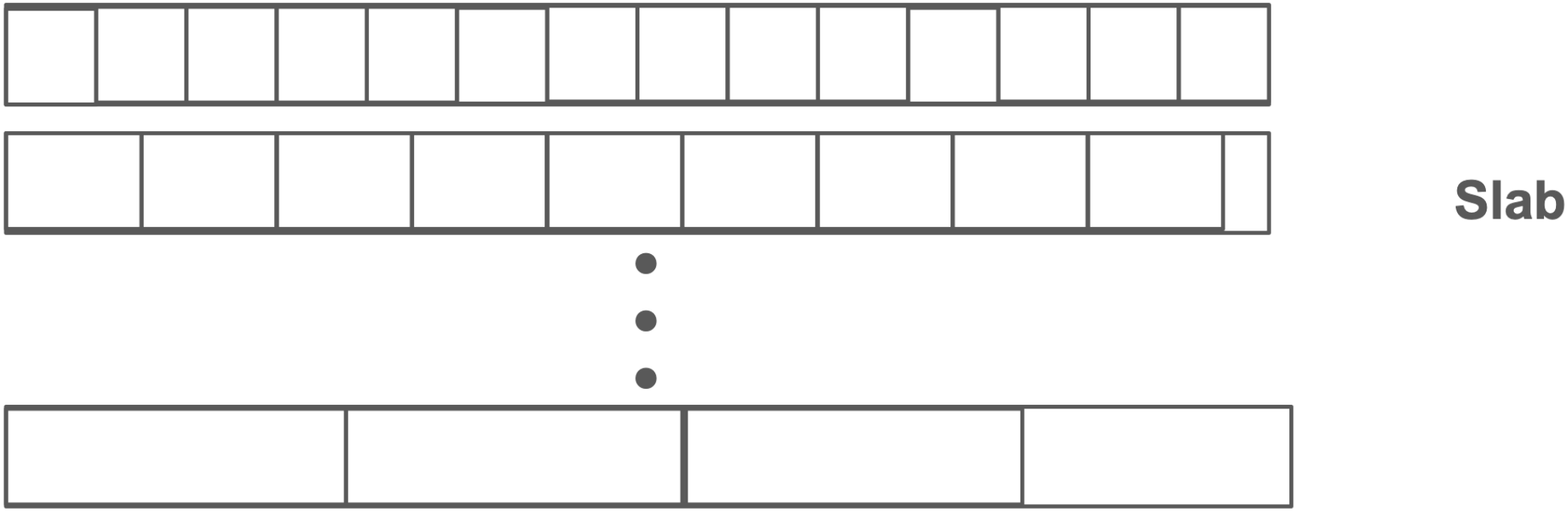



- If we think about the heap as a segment of memory like this
- We must figure out a regime to allocate memory.

# Dynamic Allocation Policies

- Best fit
  - Scan the free list and find a free block whose size is closest to the requested
- First fit
  - Iterate through free blocks and return the *first* one that is free and is big enough!
- Bit maps
  - Each bit represents whether a block in memory is free

# Slabs




Each slab has a free list

*From CS111 SPR26 lecture 11, slide 13*

# What questions can I answer?

# Where we're at

Processes

Threads

Virtual Memory

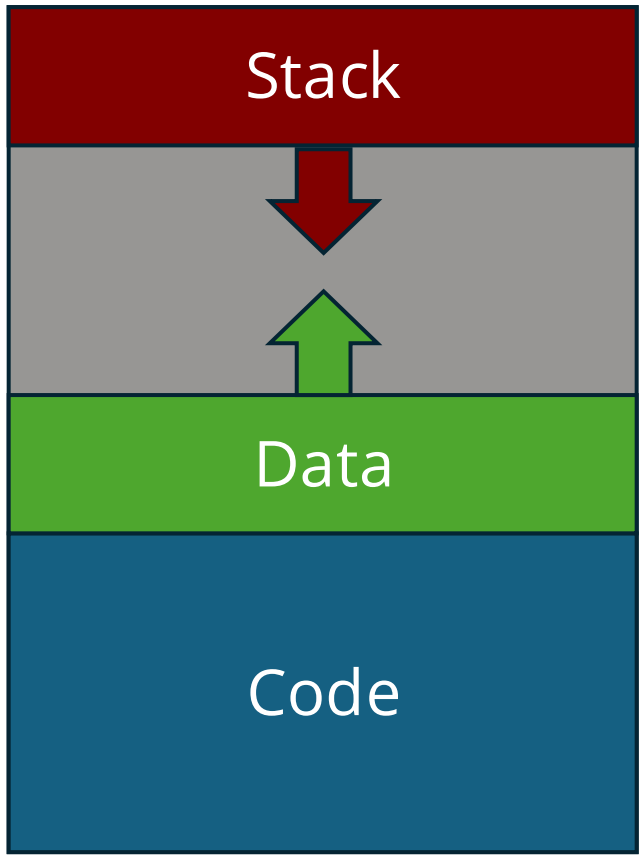
File systems

# Goals of Virtual Memory

# Remember this?

```
int main(void)
{
    int* stuff = malloc(sizeof(int)*1);
    *stuff = 5;
    pid_t pid = fork();
    printf("The last digit of pi is %d\n", *stuff);
    if (pid == 0)
        *stuff = 6
}
```

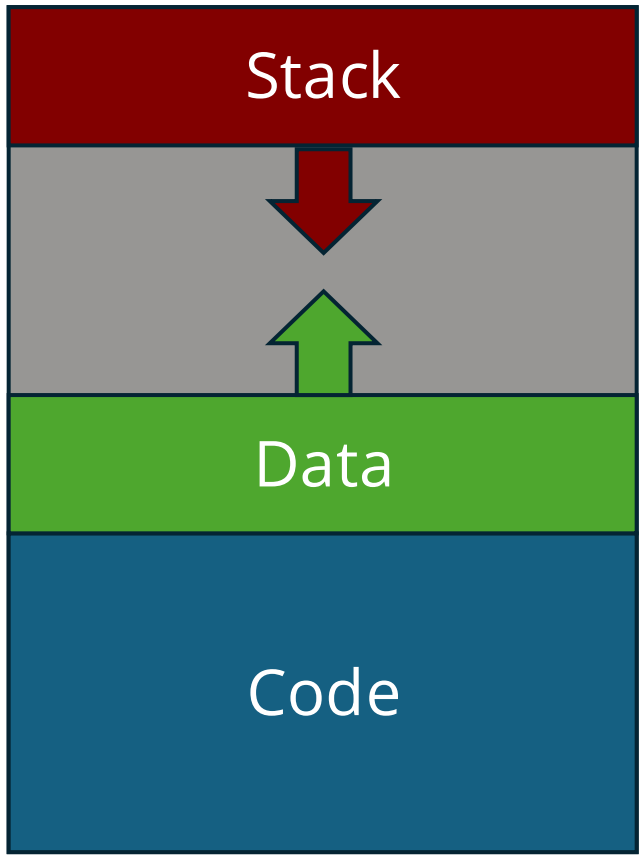
**We want each process to have *isolated* memory**



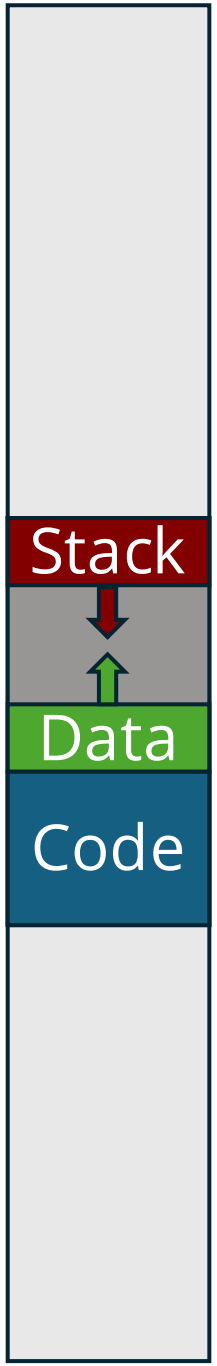

The **stack** contains stack frames

The **Data** section contains the heap and constants

The **Code** section, otherwise known as .text contains the program code




**Each process has one of these!**

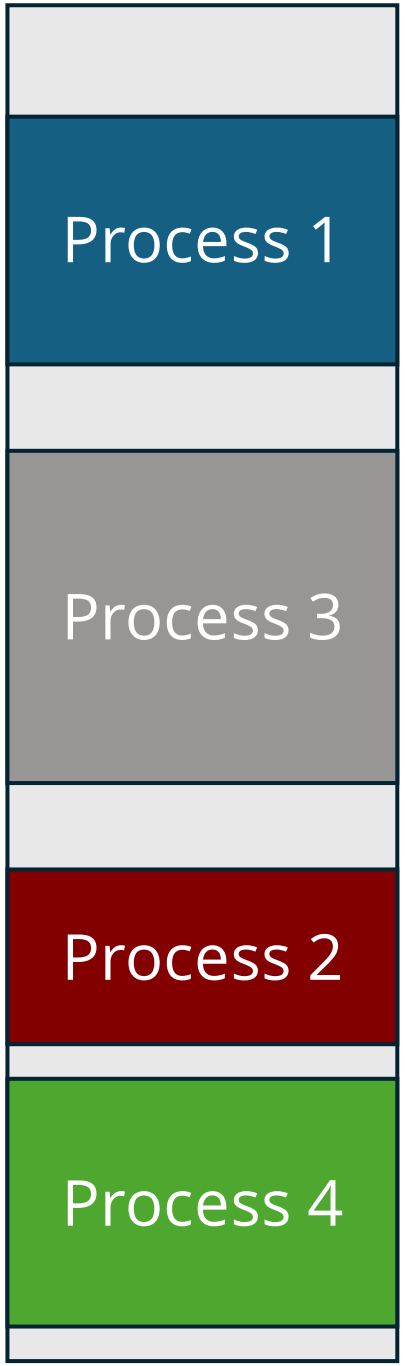



This is one process!


If you run multiple processes, then you must have some space for each process in your memory!

Image source

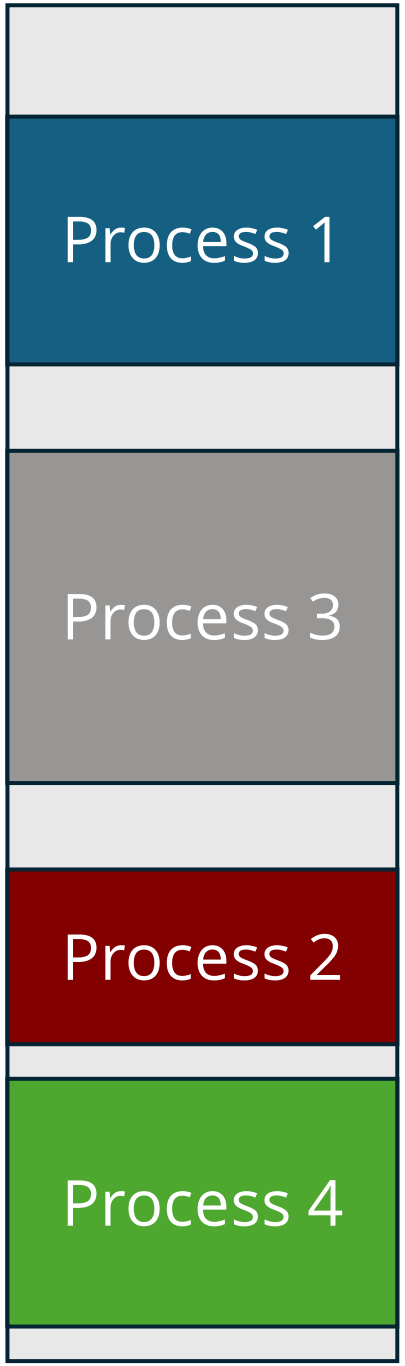
# Load-Time Relocation




# Load-Time Relocation

In **Load-Time Relocation** we assign some memory once a process is loaded!

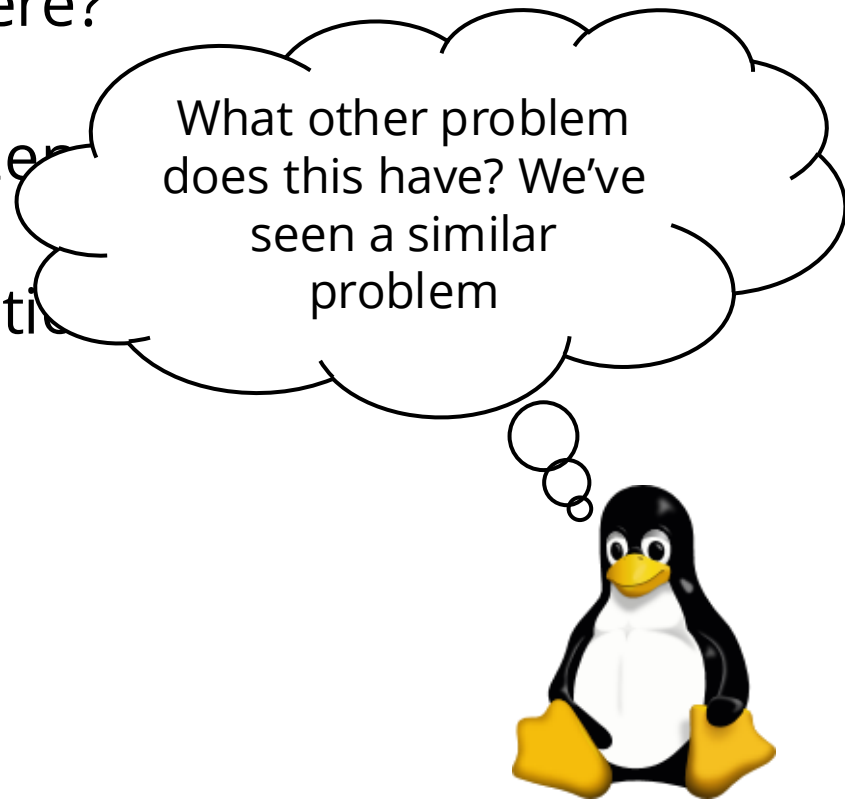
Each memory space contains the parts (Data, Stack, Code) we saw before, that each process has.

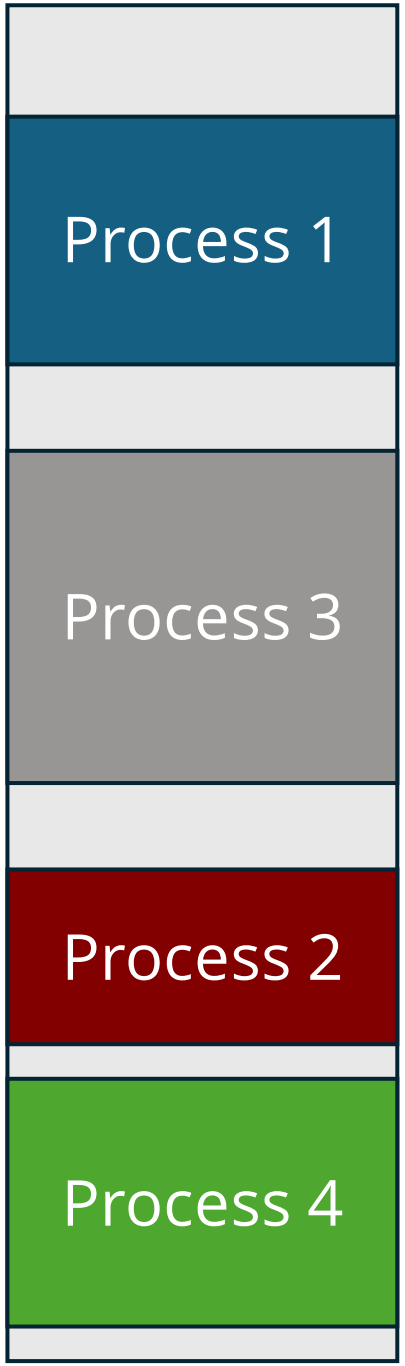



# Load-Time Relocation

**Cons:**

1. Is there isolation here?
2. Process data is froze
3. External fragmentati






# Load-Time Relocation

**Cons:**

1. Is there isolation here?

2. Process data is frozen in place

3. External fragmentation

4. Must predict the future (how big a stack frame will be)

This is pretty buns

Many chefs (processes) in the kitchen (memory)

# What questions can I answer?

# Virtual Memory


ILLUSION 100

# What is virtual memory

- Allows many processes to make use of the memory
- Gives each process the *illusion* that it has *all* the memory
- Maps virtual addresses to physical addresses

# Dynamic Address Translation

# Dynamic Address Translation

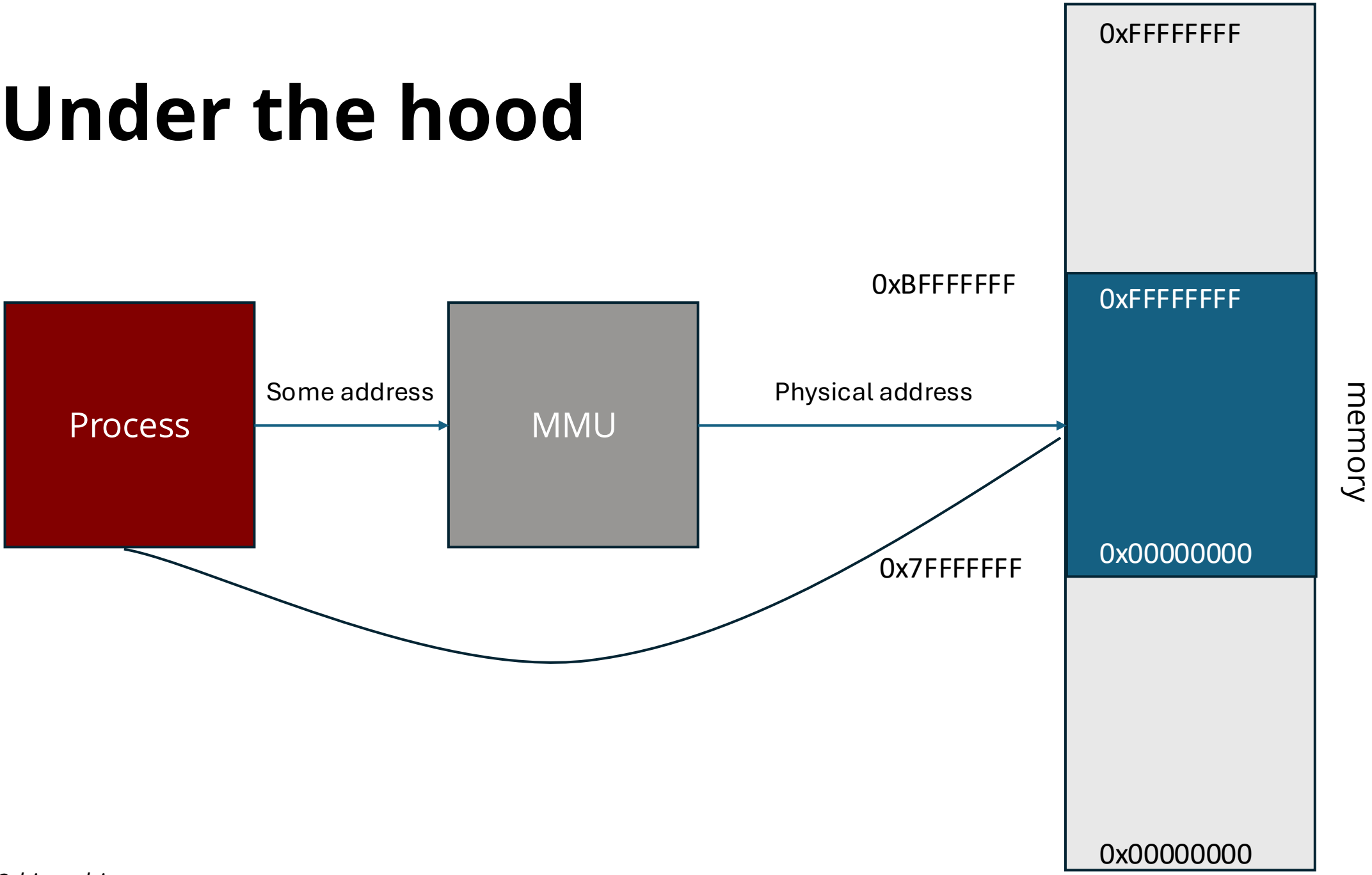
# Dynamic Address Translation

We translate *virtual addresses* into *physical addresses*

# Dynamic Address Translation

We use hardware to do this ☺

# Under the hood



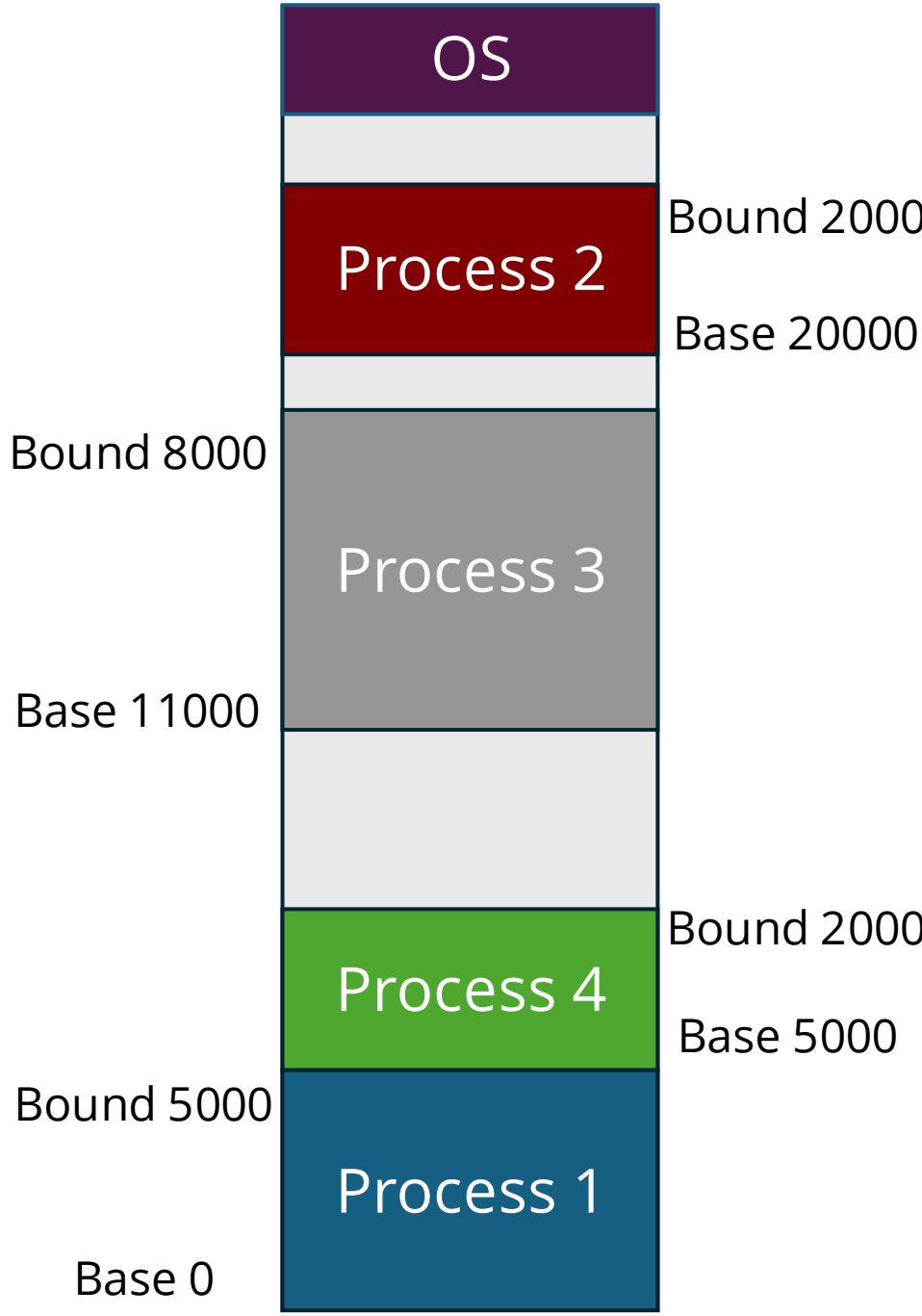

*Assumes 32-bit architecture*

# What questions can I answer?

# Base and Bound

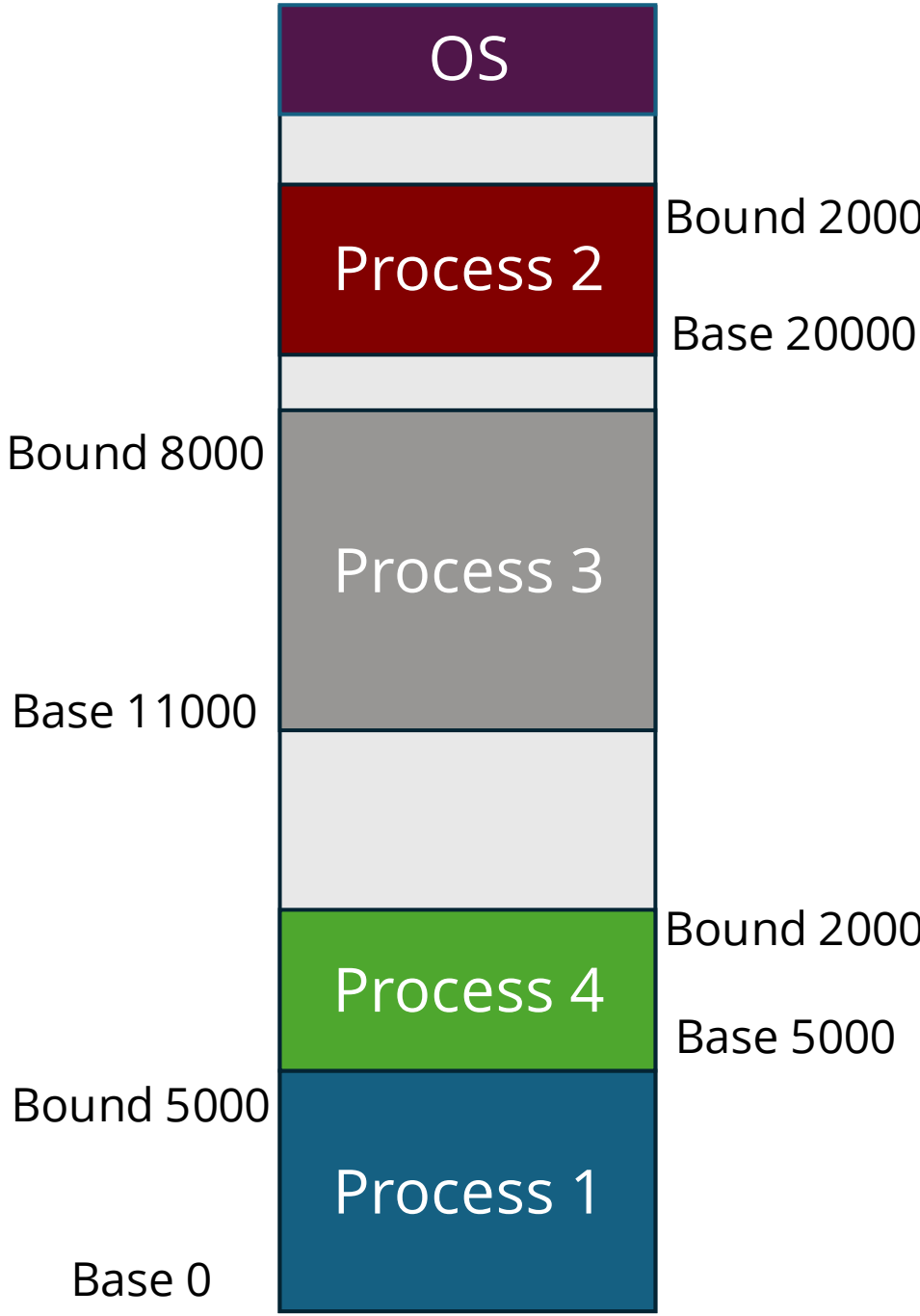
# What is base and bound

- Each process has a base, which is the starting *physical* address for that process
- Each process has a *bound* which is one greater than the highest allowed virtual address



# What is base and bound

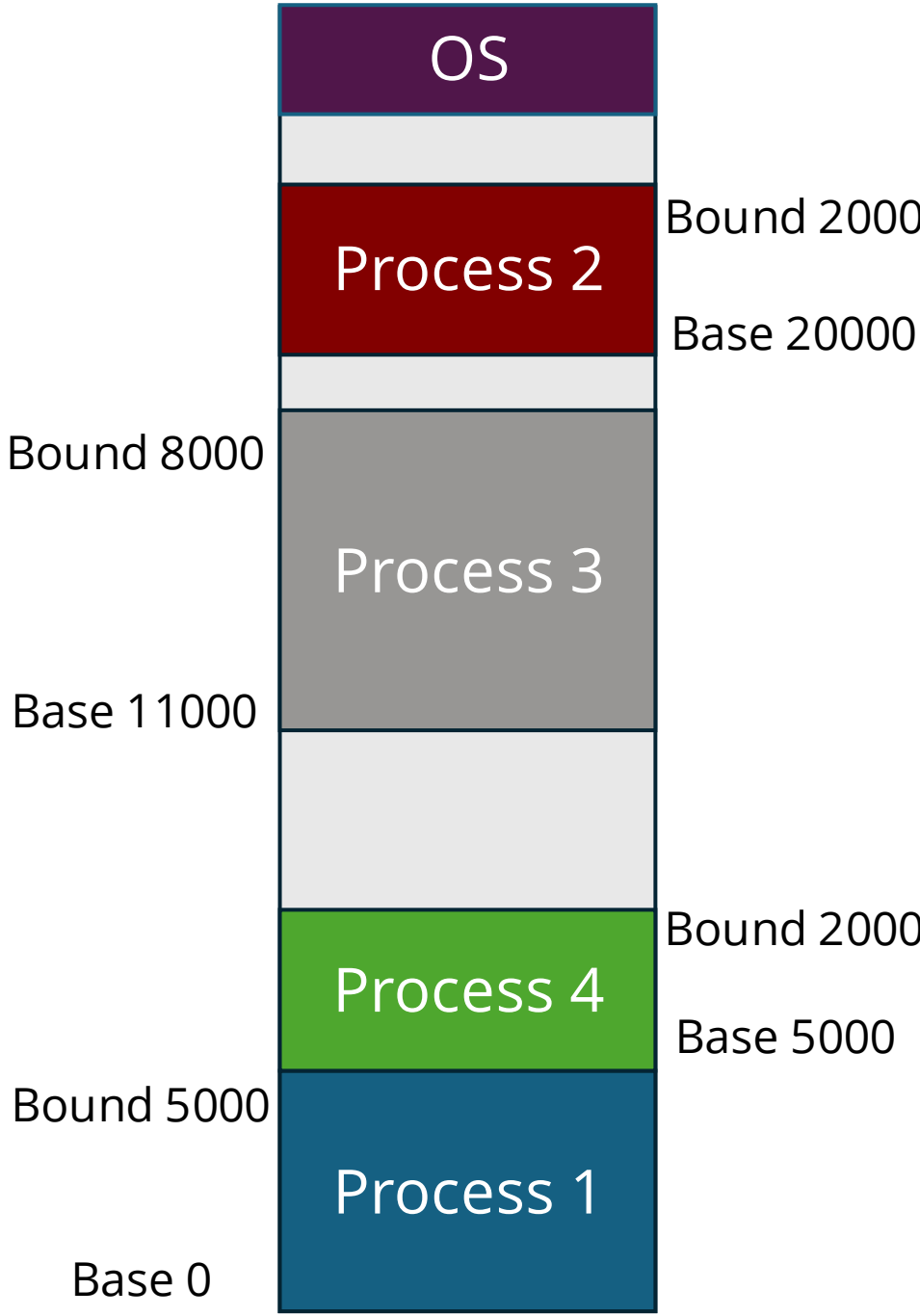
- Physical address =
  - base + requested_addr
- The OS manages the base addresses, and the process has no knowledge of it!



# What is base and bound

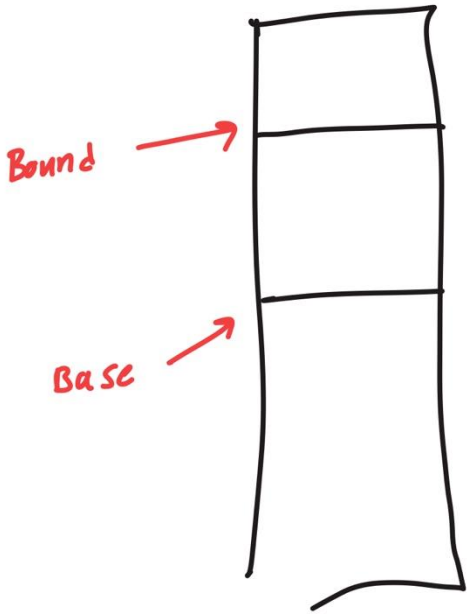
**Cons:**

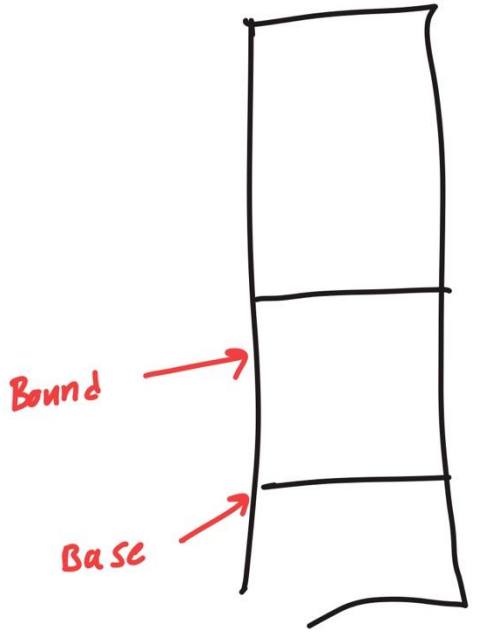
1. Each process must be contiguous
2. Fragmentation
3. Can only grow bound upward
4. No access policy



# Think-Pair-Share (2-min)

**Why can memory only grow upwards?**

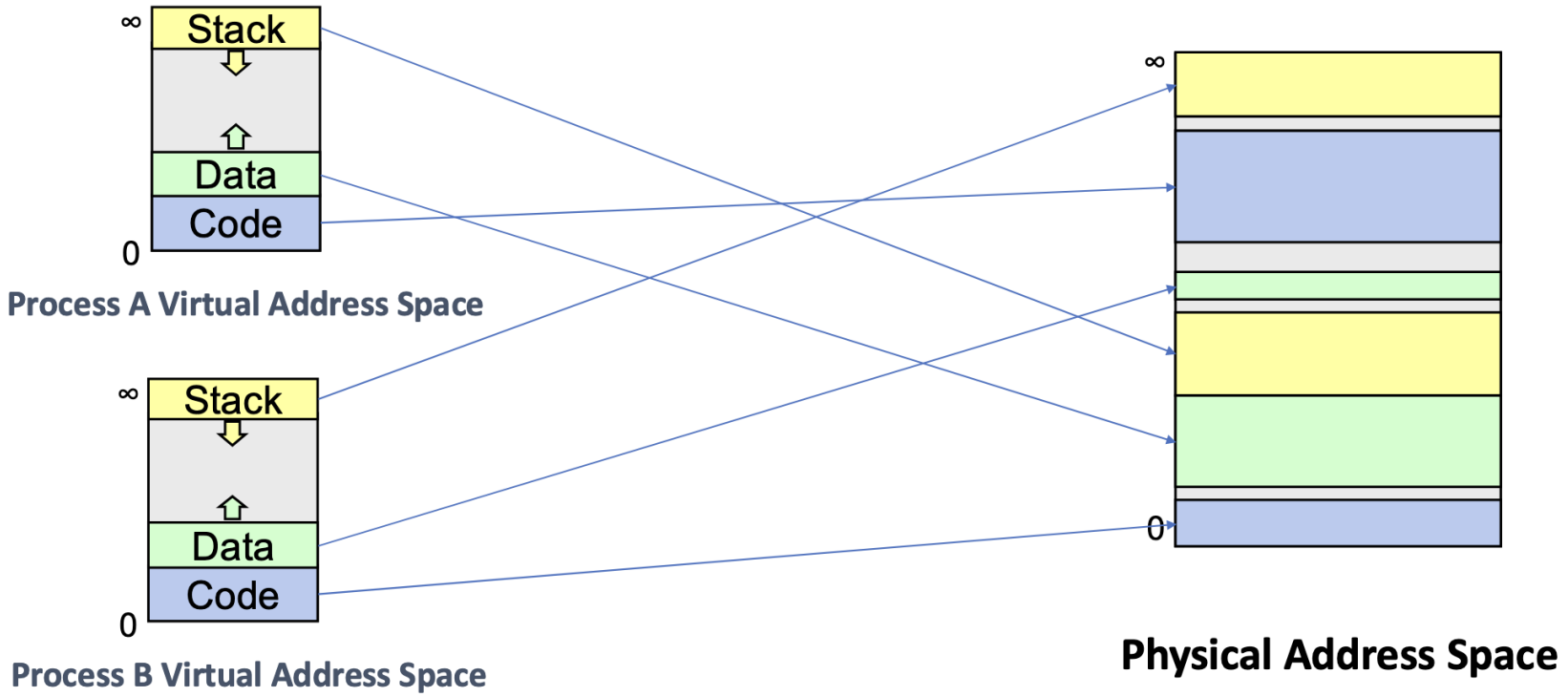


# What questions can I answer?

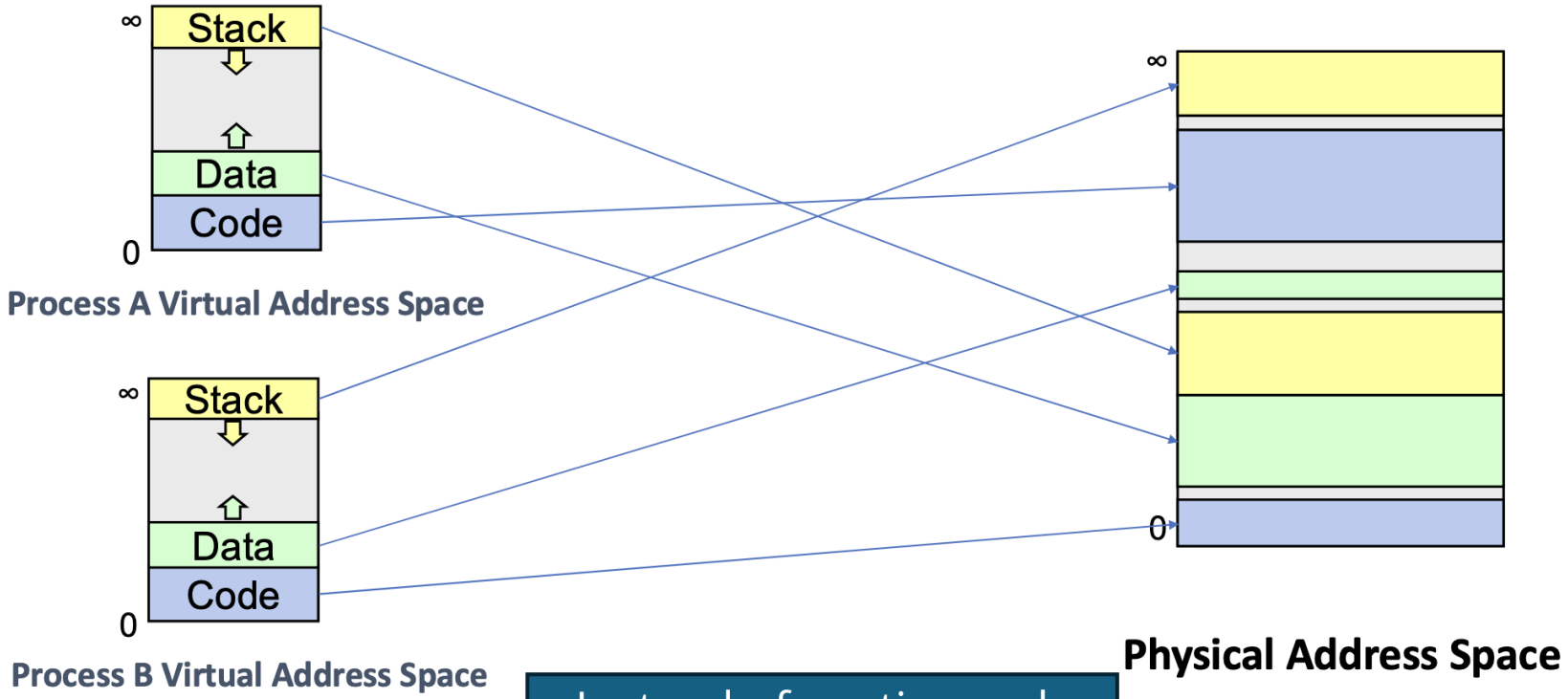
# Multiple Segments

# What are multiple segments?




*From CS111 W26 lecture 22, slide 20*

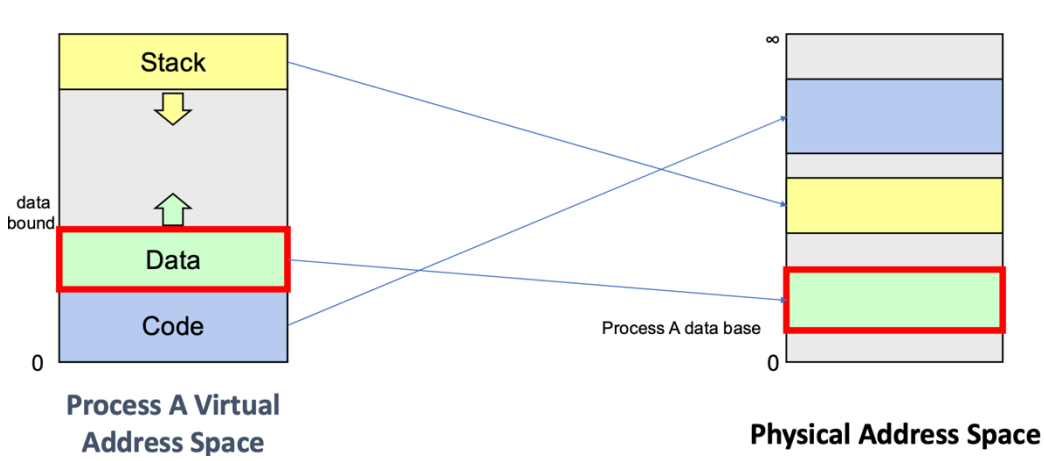
# What are multiple segments?

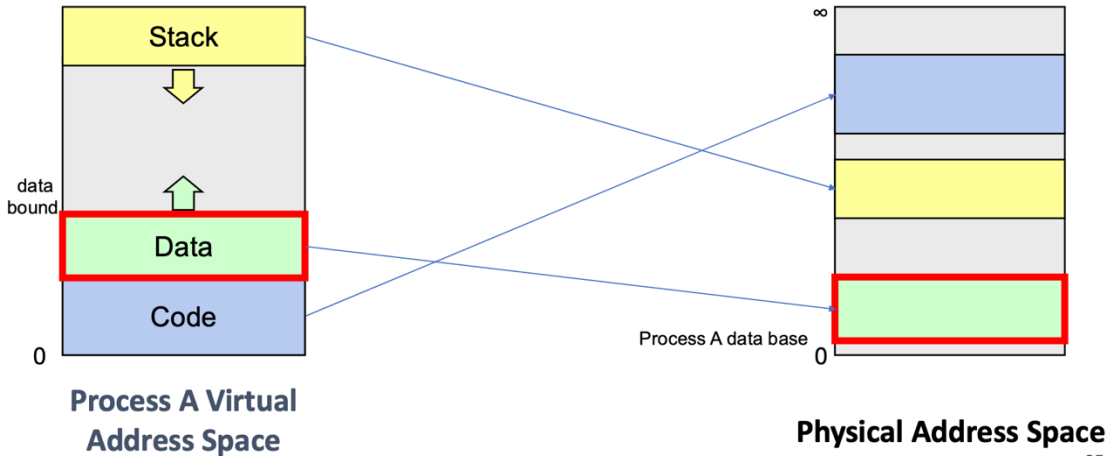



Instead of contiguously mapping processes, contiguously map the segments within processes

*From CS111 W26 lecture 22, slide 20*

# Changing bounds with multiple segments




*From CS111 W26 lecture 22, slide 24 & 25*

# Tradeoffs of multiple segments

**Pros:**

1. Segments are more easily moved to avoid segmentation

2. Can share segments
   - Why might this be useful?

**Cons:**

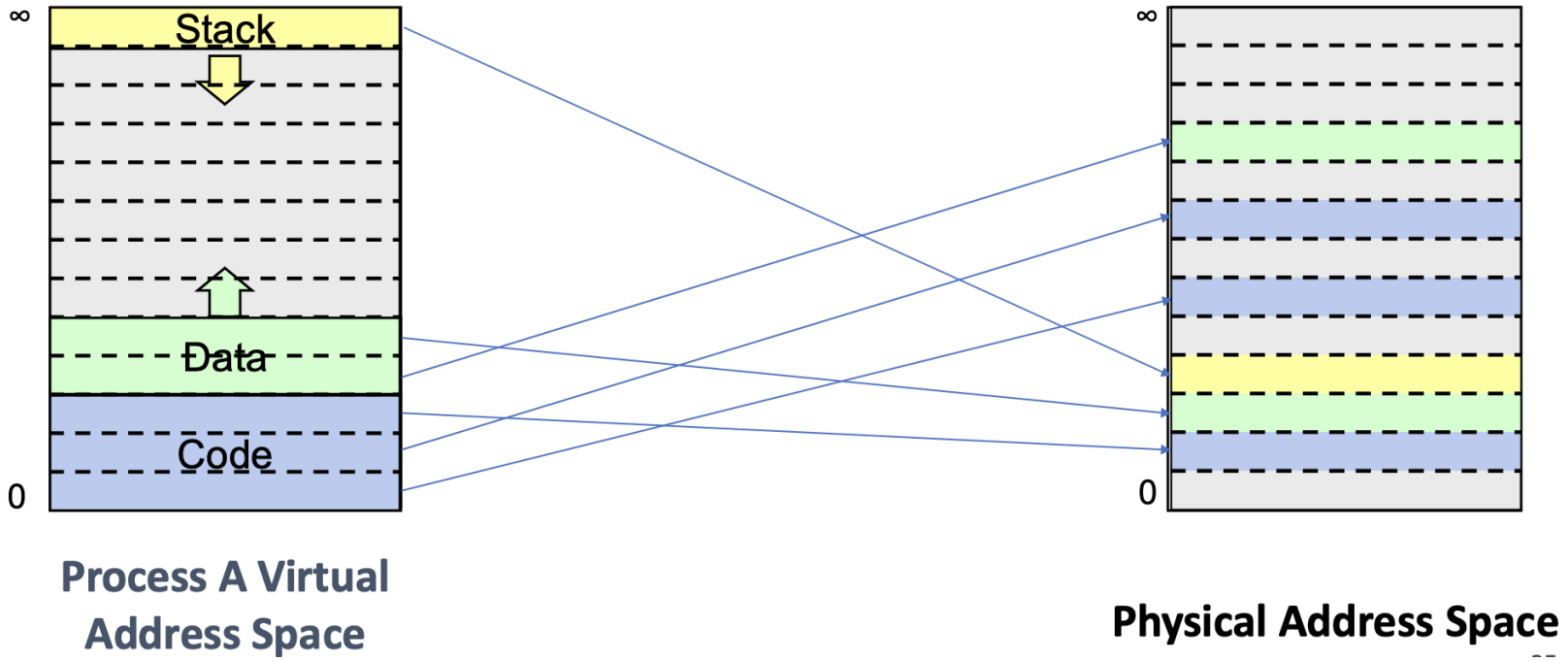
1. Can still only grow upwards with the bound

2. Can still have memory fragmentation

# What questions can I answer?

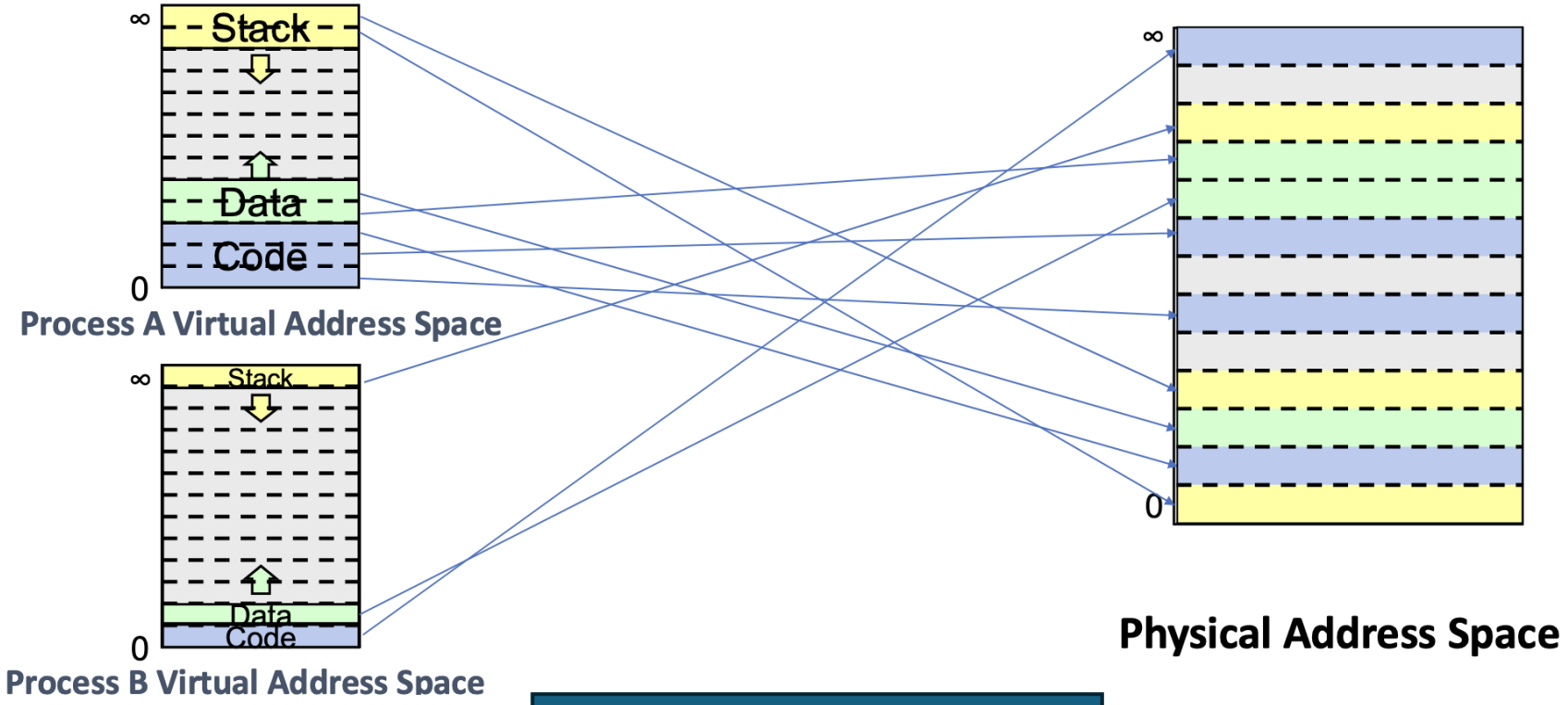
# Paging

# Reminder of *what* paging is




*From CS111 W26 lecture 22, slide 35*

# With multiple processes




Instead of mapping each segment, map pages, typically 4 Kb

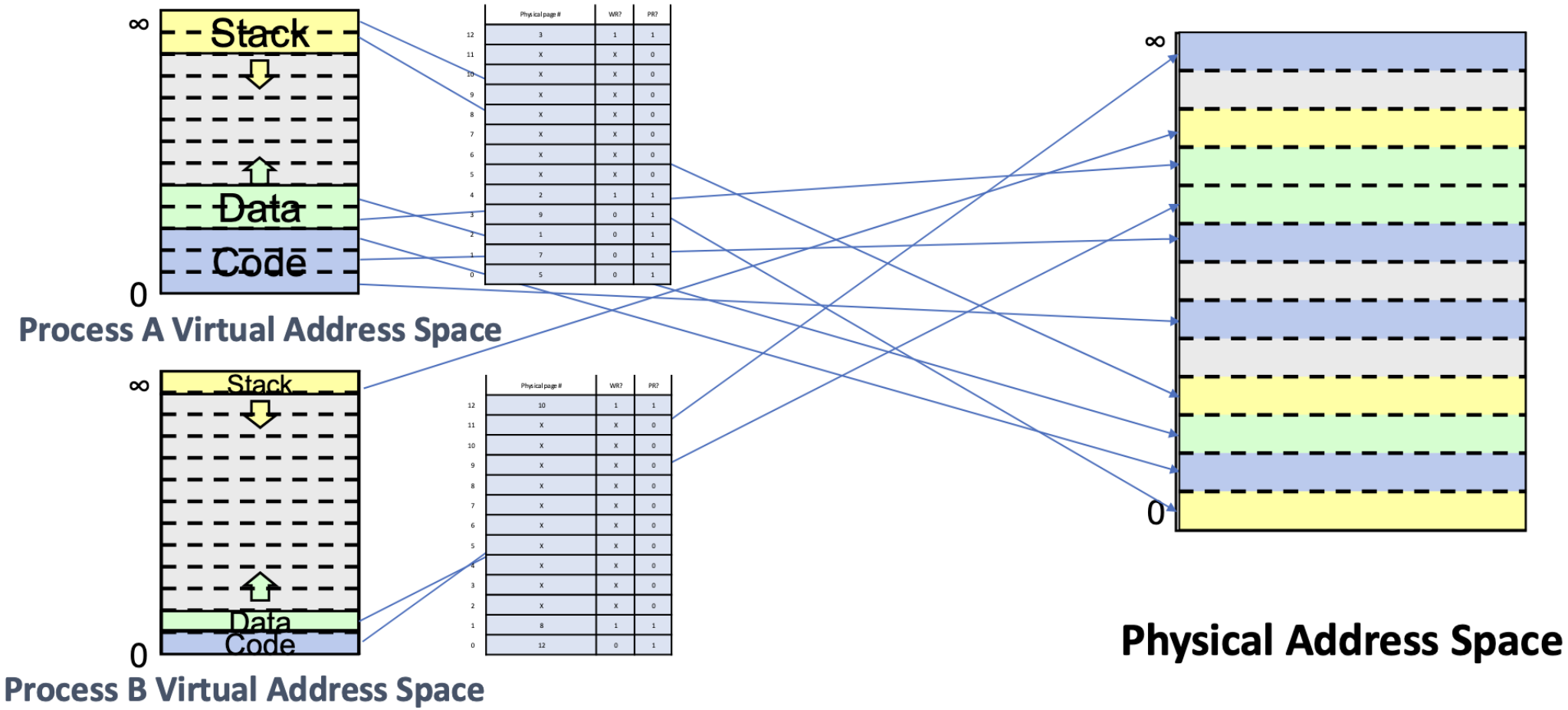
*From CS111 W26 lecture 22, slide 35*

# About paging

- Each page is associated with a *page number*
- The virtual address is the virtual page # and offset in the page
- The physical address is the physical page # and offset in that page

# How do we map virtual addresses to physical addresses?

# Each process has a Page Map




*From CS111 W26 lecture 22, slide 66*

# How do you index into the page map?

- Assume that your pages are 4 KB

- How many bits do you need to represent 4 KB?
  - You need 12 bits to represent 0-4095


x86-64 with 4KB pages has 36-bit virtual page numbers and 40-bit physical page numbers.

*From CS111 W26 lecture 22, slide 65*

# What questions can I answer?

# Demand Paging

# Think-Pair-Share (3-min)

- How many possible virtual pages are there in our scheme?


x86-64 with 4KB pages has 36-bit virtual page numbers and 40-bit physical page numbers.



- How big is the page map?

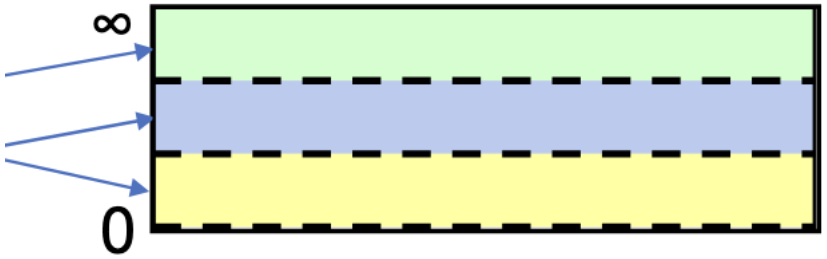
# Keeping track of presence

- Our page map has a permissions bit and also a *present* bit.
- We must modify this every time that we move pages into our physical address space.

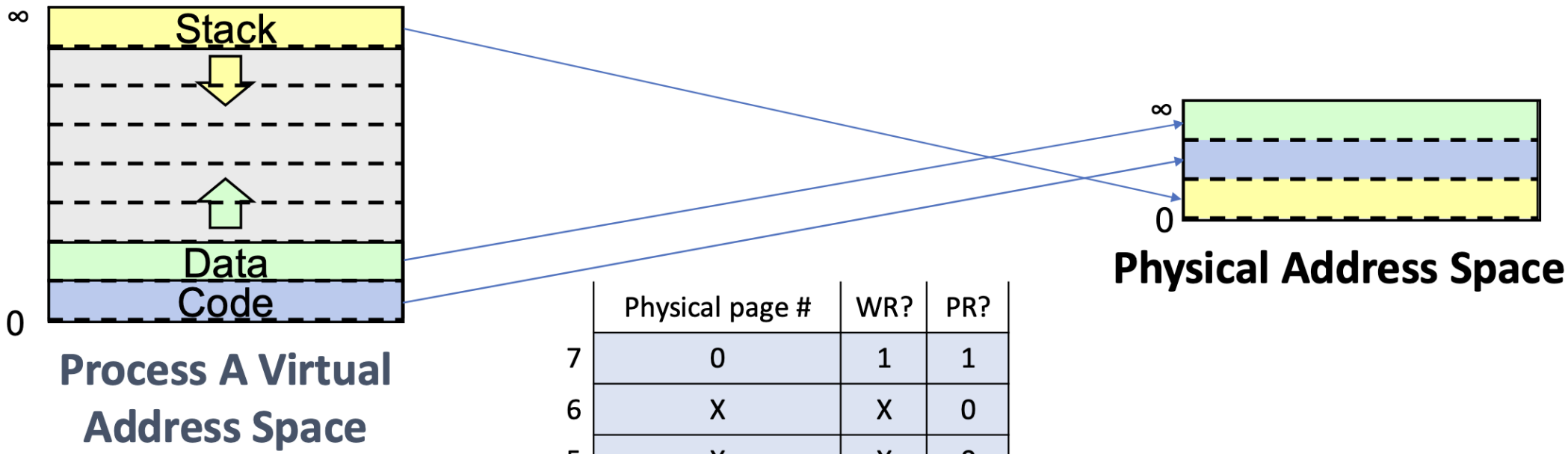
| | Physical page # | WR? | PR? |
|---|---|---|---|
| 7 | 0 | 1 | 1 |
| 6 | X | X | 0 |
| 5 | X | X | 0 |
| 4 | X | X | 0 |
| 3 | X | X | 0 |
| 2 | X | X | 0 |
| 1 | 2 | 0 | 1 |
| 0 | 1 | 0 | 1 |

# In lecture

We only had three physical pages, so we had to use this technique track which pages were present in our page map

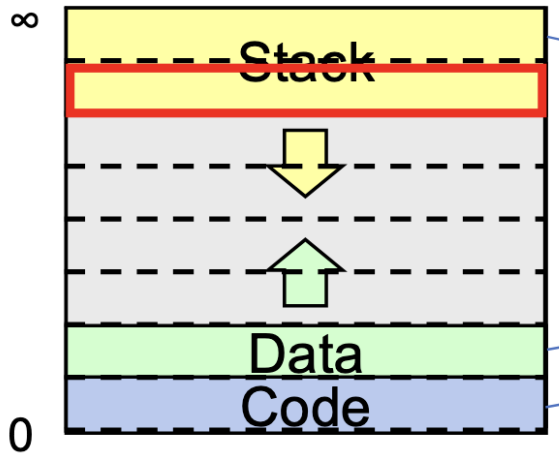


# A quick review of the lecture example

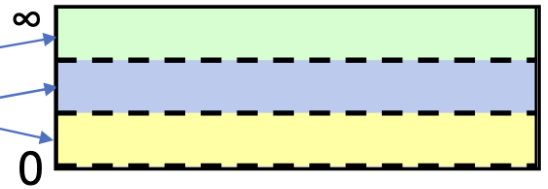



| | Physical page # | WR? | PR? |
|---|---|---|---|
| 7 | 0 | 1 | 1 |
| 6 | X | X | 0 |
| 5 | X | X | 0 |
| 4 | X | X | 0 |
| 3 | X | X | 0 |
| 2 | X | X | 0 |
| 1 | 2 | 0 | 1 |
| 0 | 1 | 0 | 1 |

*From CS111 W26 lecture 23, slide 44*




**Process A Virtual Address Space**

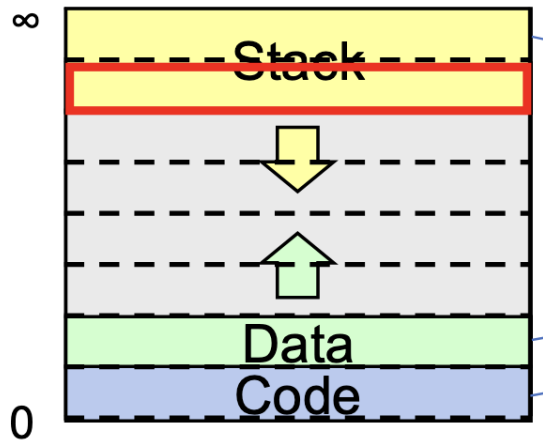



**Physical Address Space**

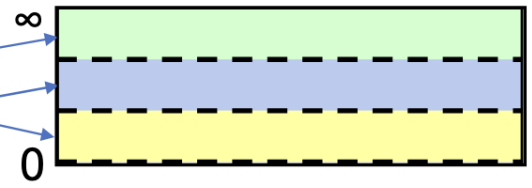
| | Physical page # | WR? | PR? |
|---|---|---|---|
| 7 | 0 | 1 | 1 |
| 6 | X | X | 0 |
| 5 | X | X | 0 |
| 4 | X | X | 0 |
| 3 | X | X | 0 |
| 2 | X | X | 0 |
| 1 | 2 | 0 | 1 |
| 0 | 1 | 0 | 1 |

**1. Pick an existing physical page and swap it to disk.**

*From CS111 W26 lecture 23, slide 45*




**Process A Virtual Address Space**

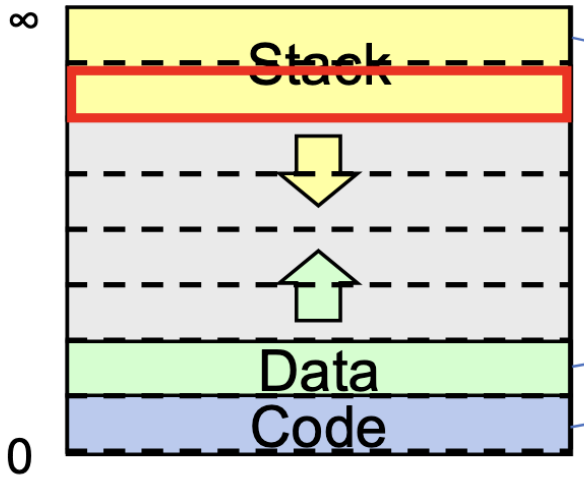



**Physical Address Space**

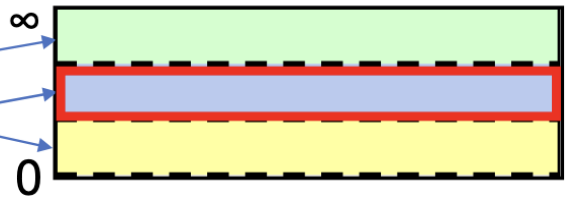
| | Physical page # | WR? | PR? |
|---|---|---|---|
| 7 | 0 | 1 | 1 |
| 6 | X | X | 0 |
| 5 | X | X | 0 |
| 4 | X | X | 0 |
| 3 | X | X | 0 |
| 2 | X | X | 0 |
| 1 | 2 | 0 | 1 |
| 0 | 1 | 0 | 1 |

**1. Pick an existing physical page and swap it to disk.**

*From CS111 W26 lecture 23, slide 46*

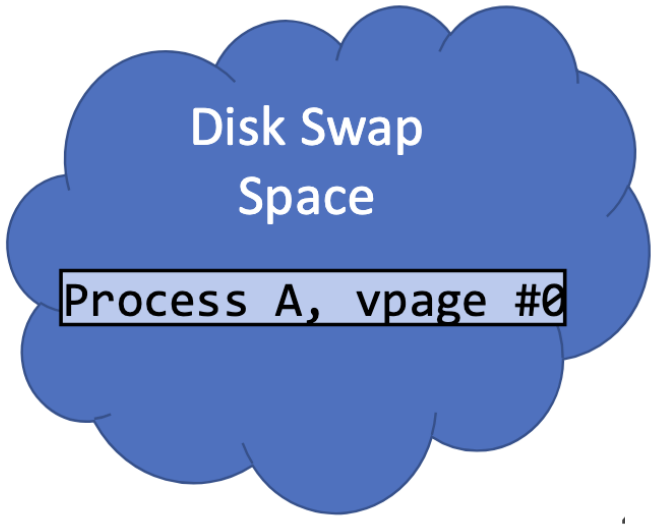



Process A Virtual Address Space



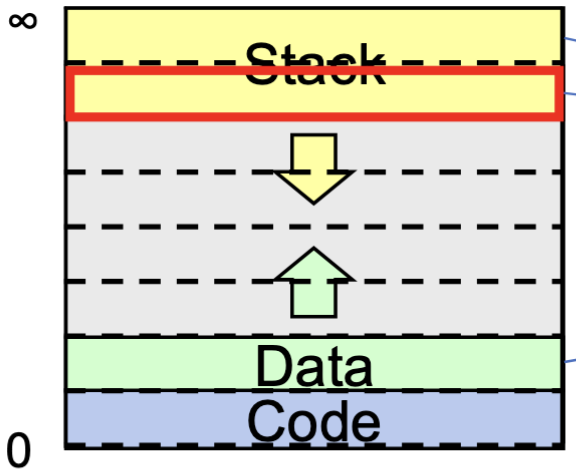

Physical Address Space

| | Physical page # | WR? | PR? |
|---|---|---|---|
| 7 | 0 | 1 | 1 |
| 6 | X | X | 0 |
| 5 | X | X | 0 |
| 4 | X | X | 0 |
| 3 | X | X | 0 |
| 2 | X | X | 0 |
| 1 | 2 | 0 | 1 |
| 0 | 1 | 0 | 0 |

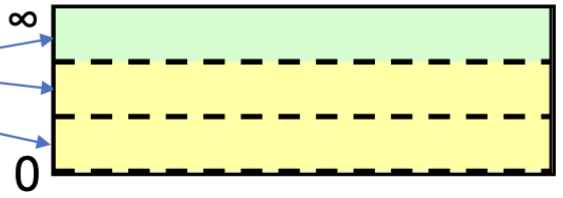



**1. Pick an existing physical page and swap it to disk, mark not present.**

*From CS111 W26 lecture 23, slide 47*

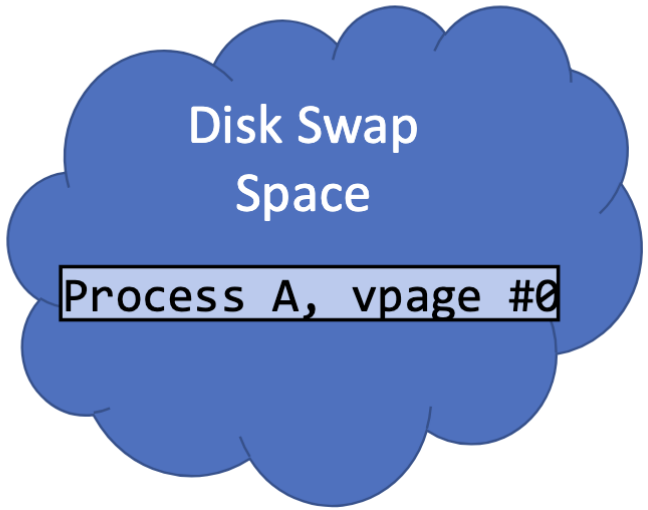



Process A Virtual Address Space




Physical Address Space

| | Physical page # | WR? | PR? |
|---|---|---|---|
| 7 | 0 | 1 | 1 |
| 6 | 1 | 1 | 1 |
| 5 | X | X | 0 |
| 4 | X | X | 0 |
| 3 | X | X | 0 |
| 2 | X | X | 0 |
| 1 | 2 | 0 | 1 |
| 0 | 1 | 0 | 0 |




**2. Map this physical page to the new virtual page.**

*From CS111 W26 lecture 23, slide 48*

# What questions can I answer?