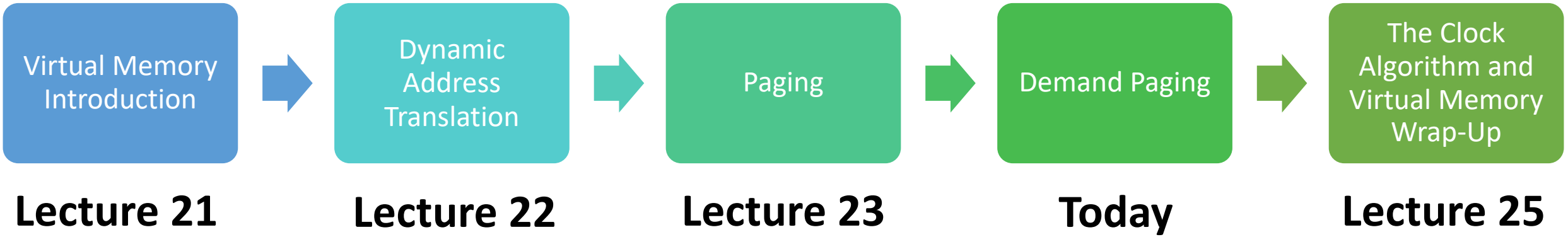
CS111, Lecture 24
Demand Paging

Optional reading:

Operating Systems: Principles and Practice (2nd Edition): Chapter 9

Topic 4: Virtual Memory - How can one set of memory be shared among several processes? How can the operating system manage access to a limited amount of system memory?

CS111 Topic 4: Virtual Memory



assign6: implement *demand paging* system to translate addresses and load/store memory contents for programs as needed.

Learning Goals

- Learn about page maps and how they help translate virtual addresses to physical addresses
- Understand how paging allows us to swap memory contents to disk when we need more physical pages.
- Learn about the benefits of demand paging in making memory look larger than it really is

Plan For Today

- **Recap:** Paging so far
- Page Map Size
- Demand Paging

Plan For Today

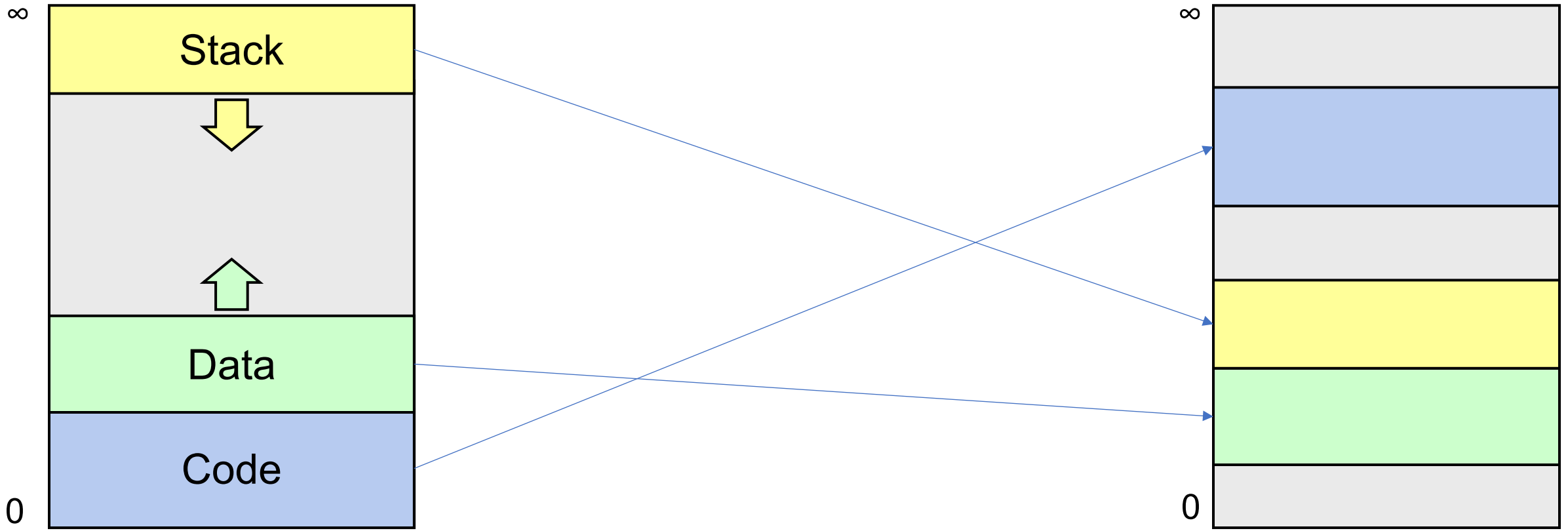
- **Recap: Paging so far**
- Page Map Size
- Demand Paging

Approach #3: Paging

Key Idea: Each process's virtual (and physical) memory is divided into fixed-size chunks called *pages*. (Common size is 4KB pages).

- A "page" of virtual memory maps to a "page" of physical memory. No partial pages
- The **page number** is a numerical ID for a page. We have virtual page numbers and physical page numbers.
- Each process has a *page map ("page table")* with an entry for each virtual page, mapping it to a physical page number and other info such as a protection bit (read-only or read-write).

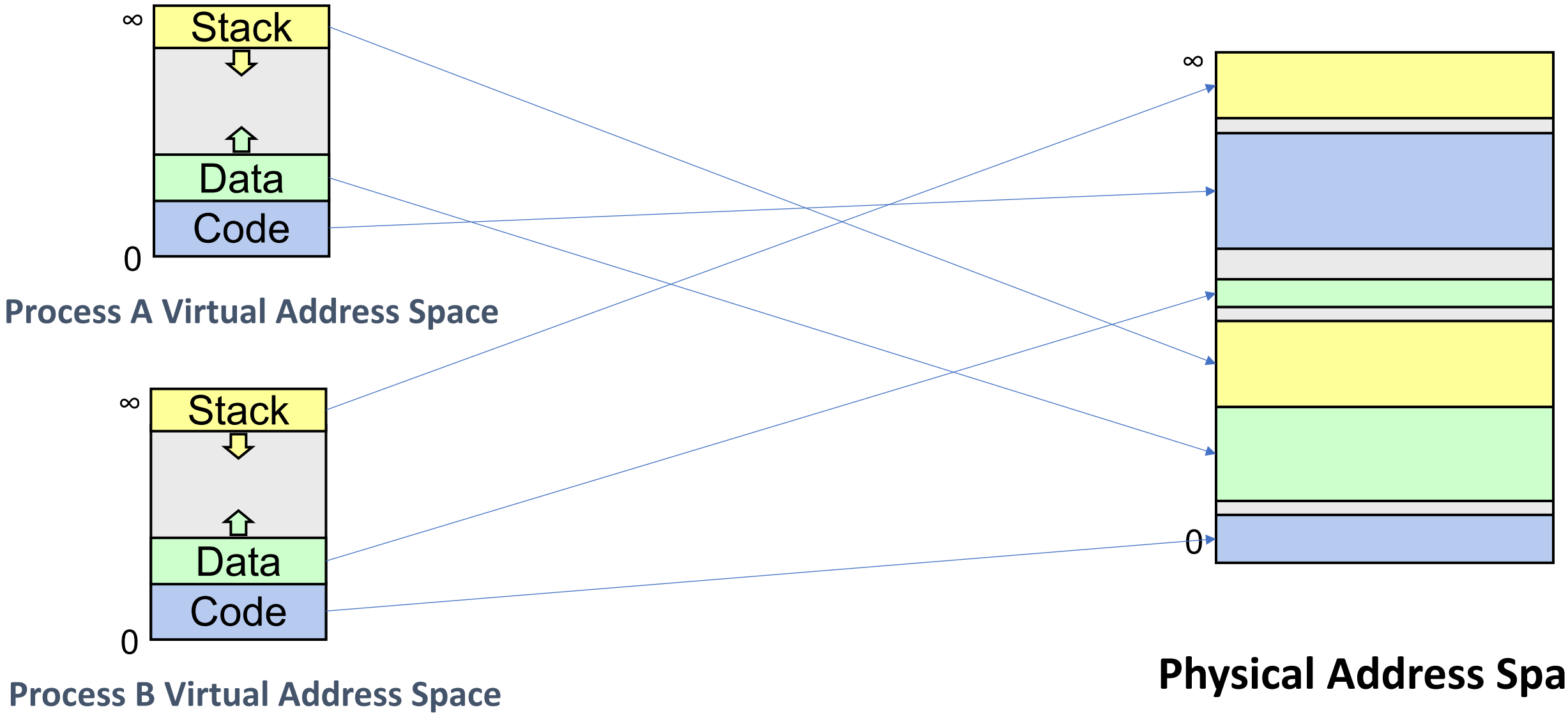
Multiple Segments



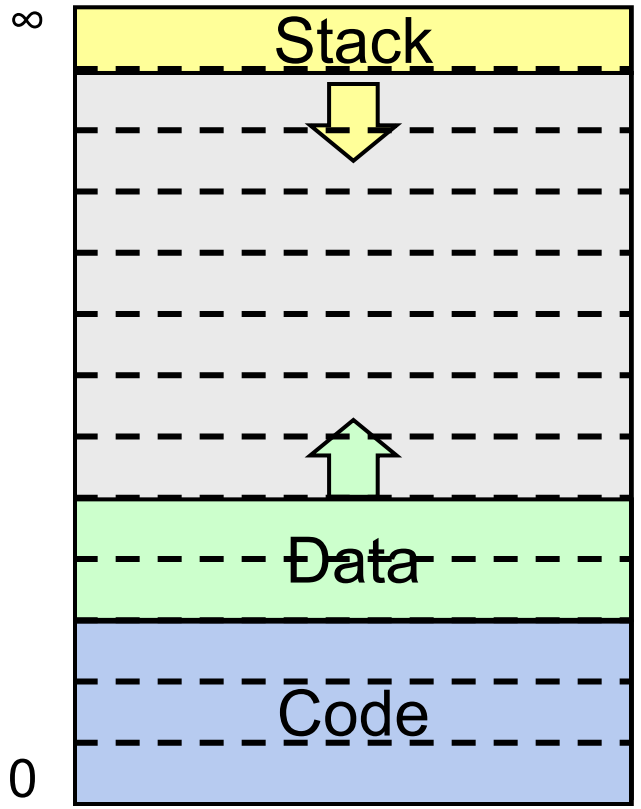
Process A Virtual Address Space

Physical Address Space

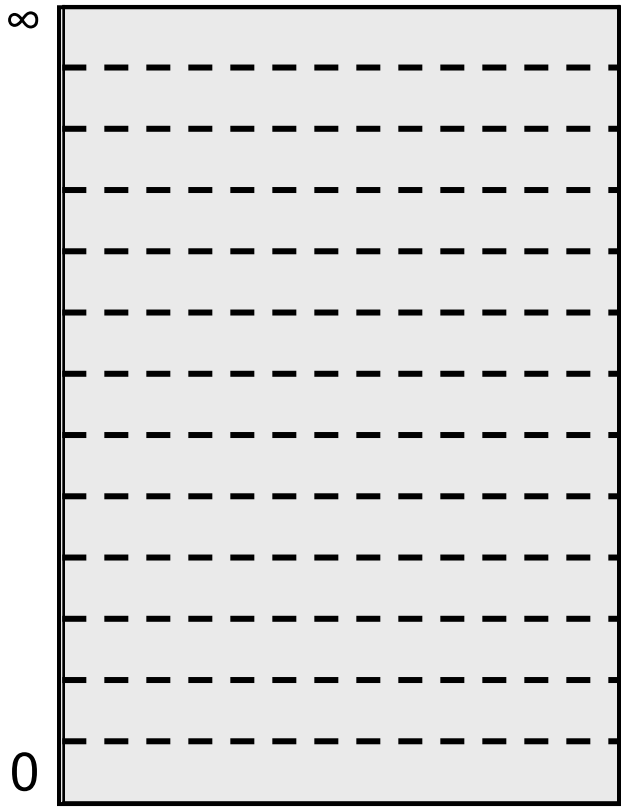
Multiple Segments



Paging

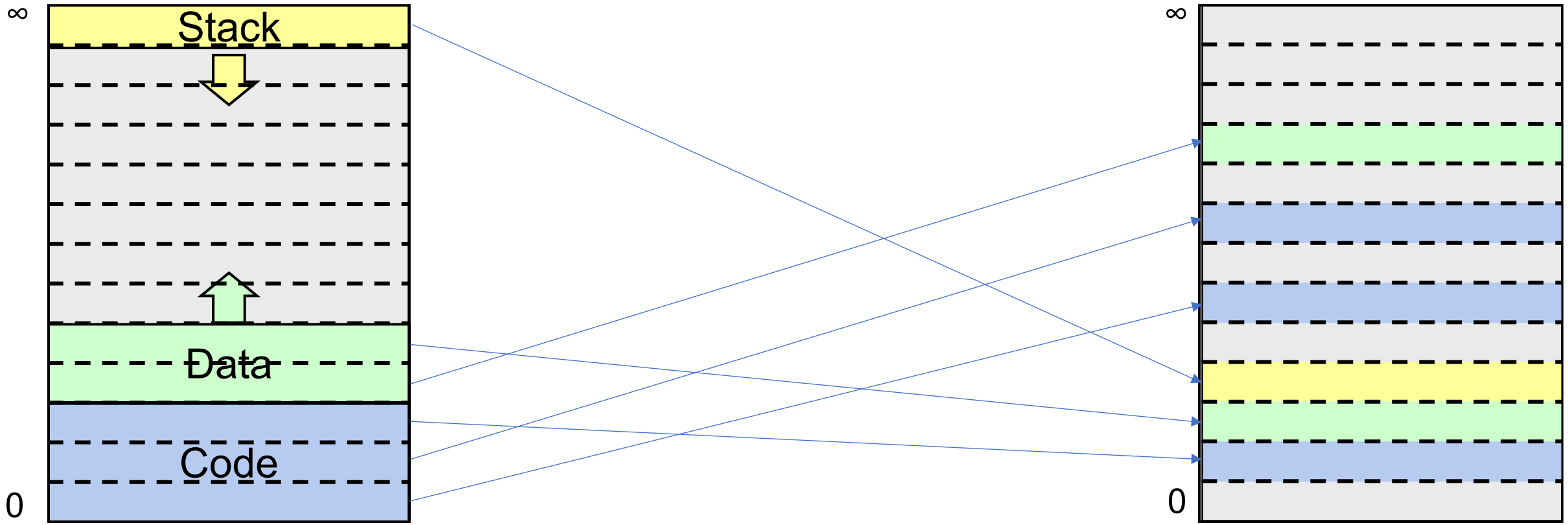


Process A Virtual Address Space



Physical Address Space

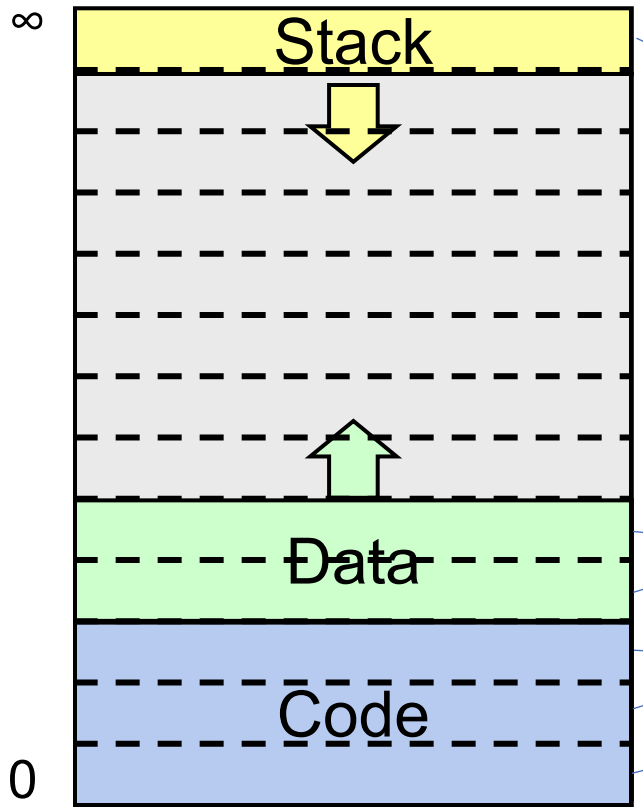
Paging



Process A Virtual Address Space

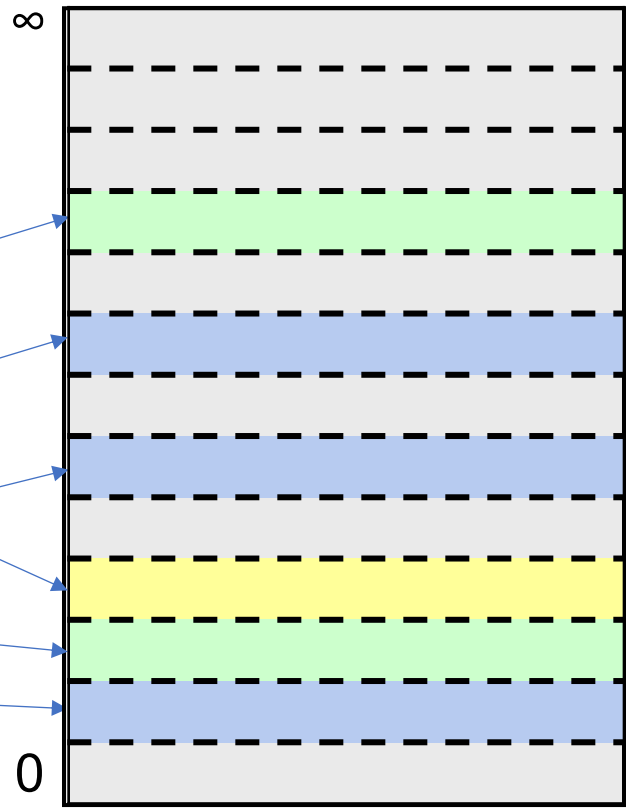
Physical Address Space

Page Map



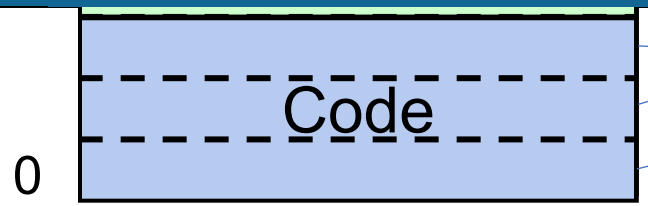
Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	X	X	0
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1



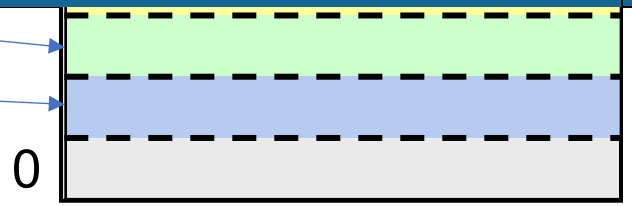
Physical Address Space

Page Map



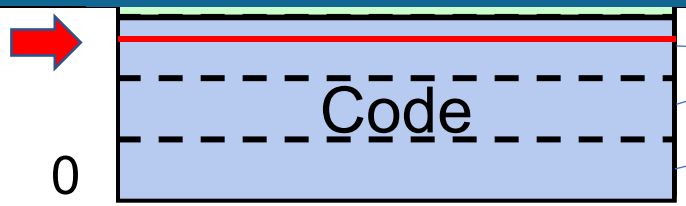
Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	X	X	0
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1



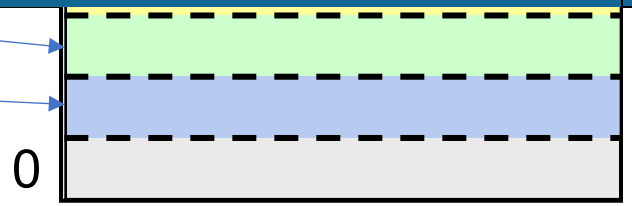
Physical Address Space

Page Map



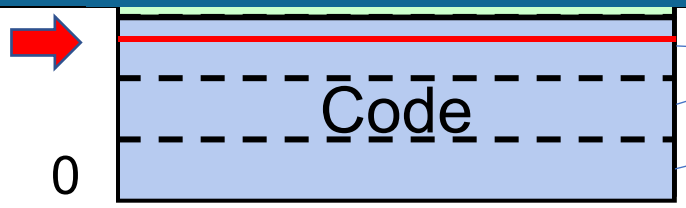
Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	X	X	0
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1



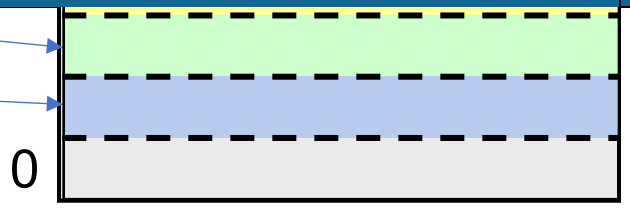
Physical Address Space

Page Map



Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	X	X	0
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1

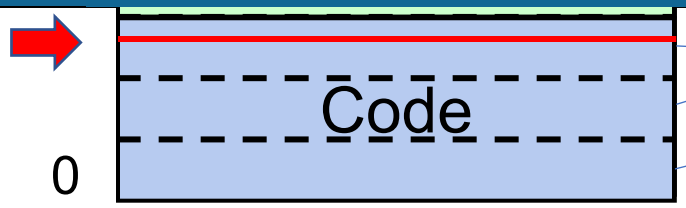


Physical Address Space

Virtual Address

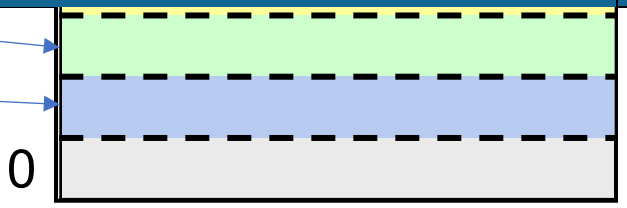
0x2223

Page Map



Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	X	X	0
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1



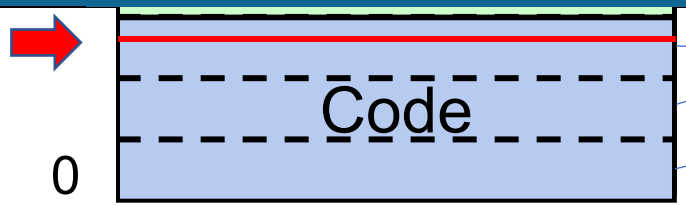
Physical Address Space

Virtual page #	*Offset*
0x2	**0x223**

Virtual Address

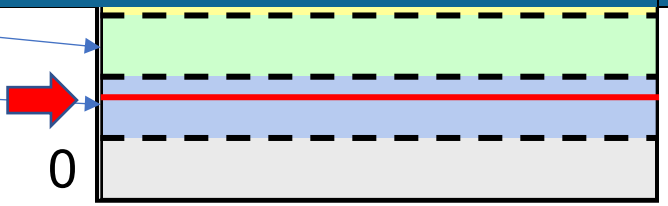
0x2223

Page Map



Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	X	X	0
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1



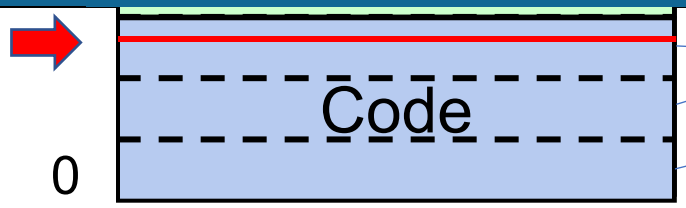
Physical Address Space

Virtual page #	*Offset*
0x2	**0x223**

Virtual Address

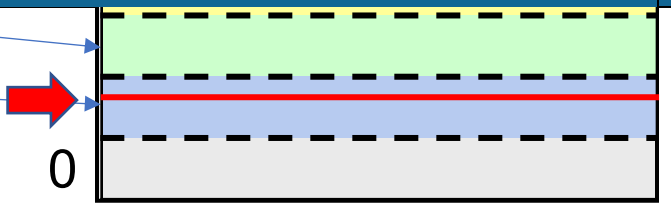
0x2223

Page Map



Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	X	X	0
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1



Physical Address Space

Virtual page #	*Offset*
0x2	**0x223**

Virtual Address

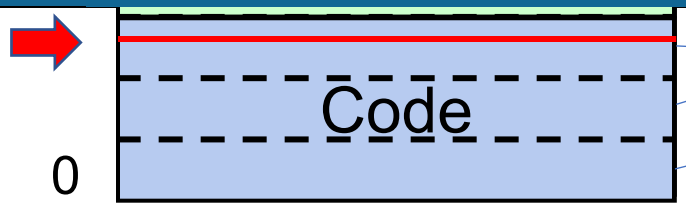
0x2223

Physical page #	*Offset*
???	**0x223**

Physical Address

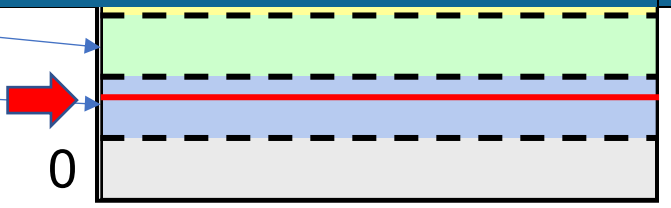
???

Page Map



Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	X	X	0
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1



Physical Address Space

Virtual page #	*Offset*
0x2	**0x223**

Virtual Address

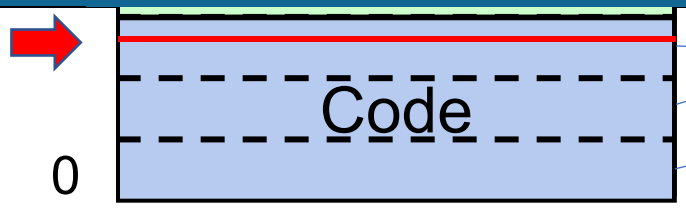
0x2223

Physical page #	*Offset*
0x1	**0x223**

Physical Address

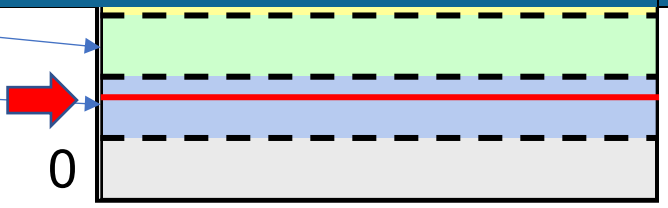
???

Page Map



Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	X	X	0
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1



Physical Address Space

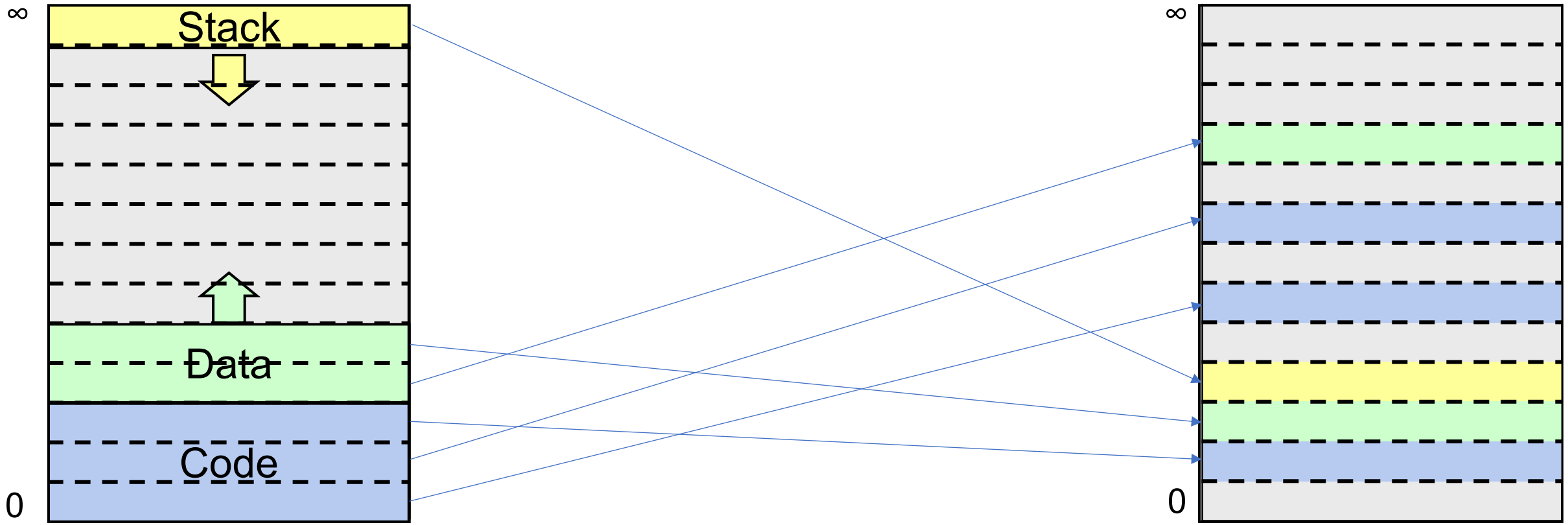
Virtual page #	*Offset*
0x2	**0x223**

Virtual Address
0x2223

Physical page #	*Offset*
0x1	**0x223**

Physical Address
0x1223

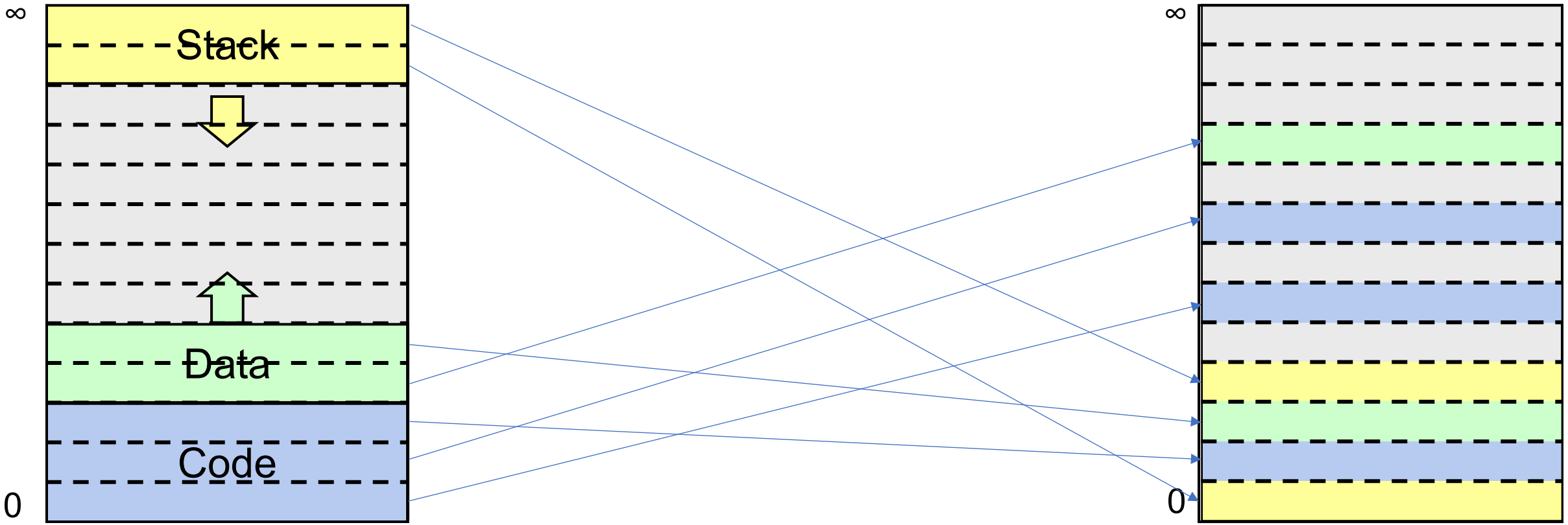
Requesting More Memory



Process A Virtual Address Space

Physical Address Space

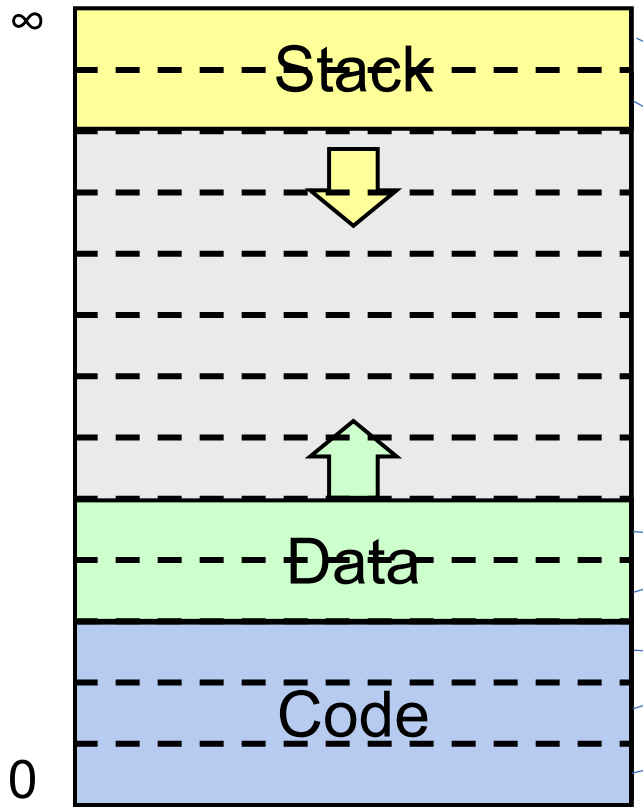
Requesting More Memory



Process A Virtual Address Space

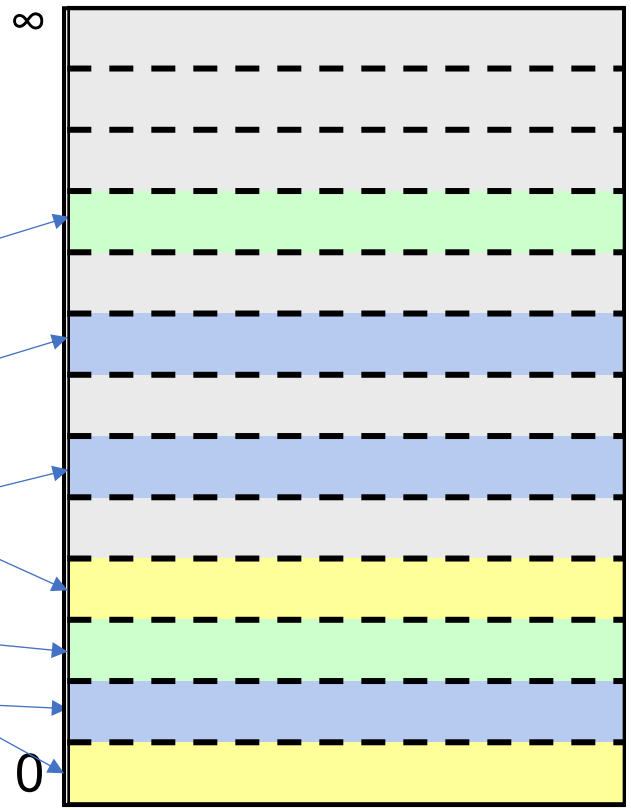
Physical Address Space

Requesting More Memory



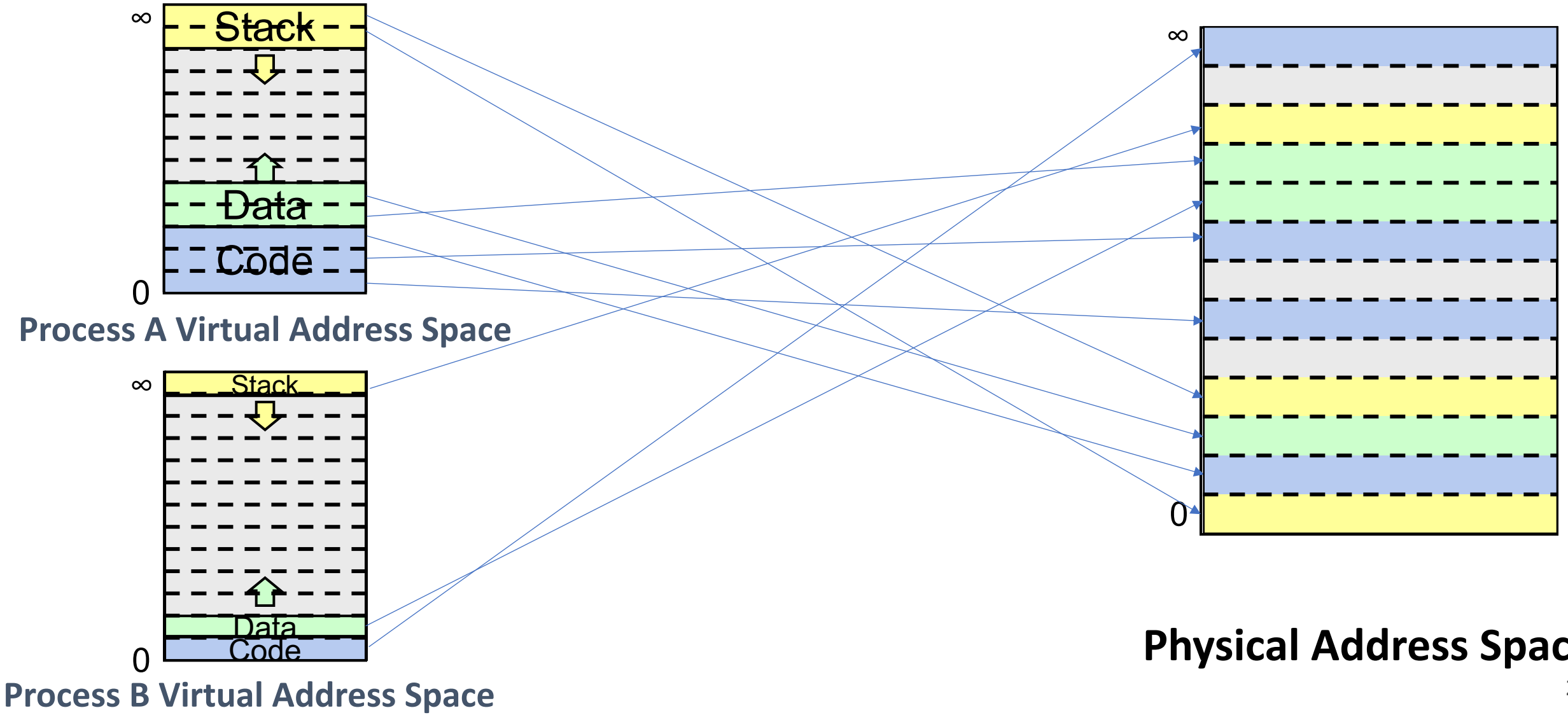
Process A Virtual Address Space

	Physical page #	WR?	PR?
12	3	1	1
11	0	1	1
10	X	X	0
9	X	X	0
8	X	X	0
7	X	X	0
6	X	X	0
5	X	X	0
4	2	1	1
3	9	0	1
2	1	0	1
1	7	0	1
0	5	0	1

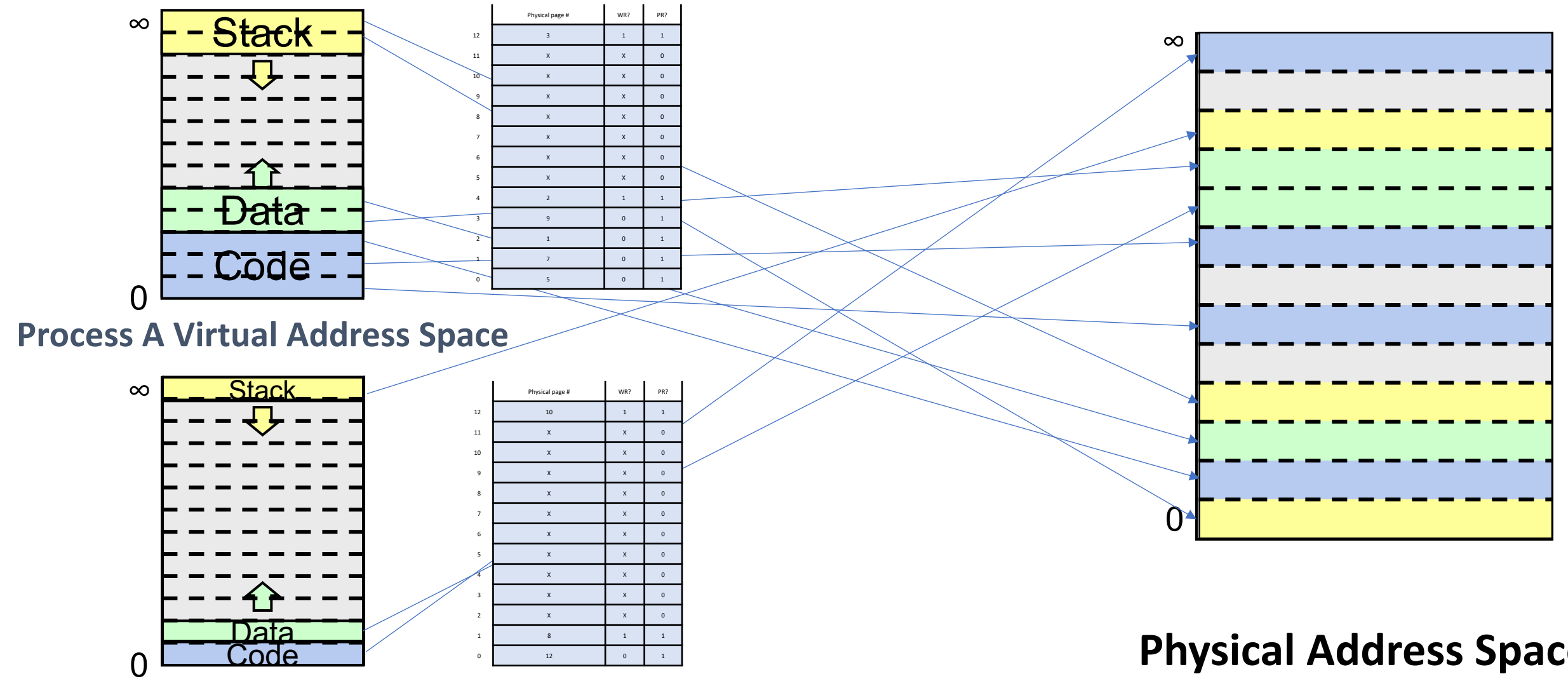


Physical Address Space

Paging


∞
Stack
Data
Code
0
Process A Virtual Address Space
∞
Stack
Data
Code
0
Process B Virtual Address Space
∞
0
Physical Address Space

Each Process Has A Page Map



Paging Summary

Each process has a *page map ("page table")* with an entry for every virtual page, mapping it to a physical page number and other info such as a protection bit (read-only or read-write) and whether it is present.

- The page map is stored in contiguous memory
- All pages the same size – no more external fragmentation! (but some internal fragmentation if not all of a page is used)

Problem: how big is a single process's page map? You said an entry for *every* page?

Plan For Today

- **Recap:** Paging so far
- **Page Map Size**
- Demand Paging

Page Map Size

Problem: how big is a single process's page map? An entry for *every* page?

Example with x86-64: 36-bit virtual page numbers, 8-byte map entries

How many possible virtual page #s? 2^{36}

2^{36} virtual pages x 8 bytes per page entry = ???

Page Map Size

Problem: how big is a single process's page map? An entry for *every* page?

Example with x86-64: 36-bit virtual page numbers, 8-byte map entries

How many possible virtual page #s? 2^{36}

2^{36} virtual pages x 8 bytes per page entry = **512GB!!** (2^{39} bytes)

Plus, most processes are small, so most pages will be "not present". And even large processes use their address space sparsely (e.g. code at bottom, stack at top).

We'll see how to mitigate this problem later.

assign6

On assign6, you'll implement your own virtual memory system using paging:

- You'll intercept memory requests
- You'll maintain a page map mapping virtual addresses to physical ones

Plan For Today

- Recap: Paging so far
- Page Map Size
- **Demand Paging**

Demand Paging

Thought: if memory is in high demand, we could fill up all of memory. If a process wants another page, we may not have any more. What do we do?

Idea #1: process just can't have any more memory (not ideal)

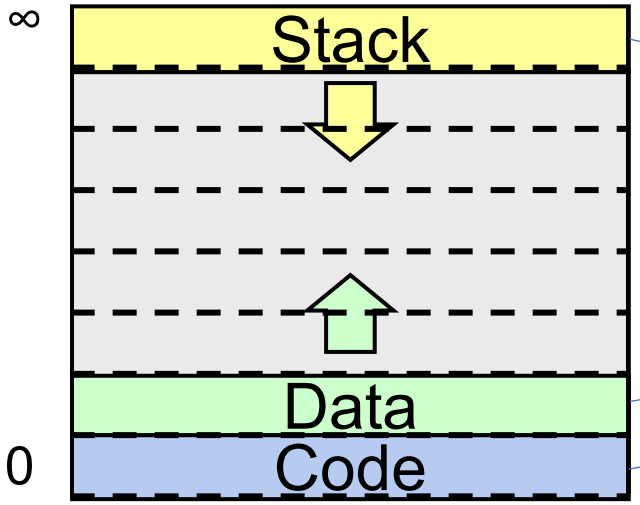
Idea #2: let's "borrow" a used physical page – we'll store its existing contents on disk, and then use the page for this new data. If the old contents are referenced later, we'll load them back into a physical page.

Overall goal: make physical memory look larger than it is.

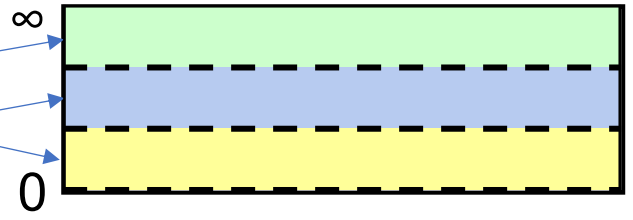
Demand Paging

- Locality – most programs spend most of their time using a small fraction of their code and data
- Keep in memory the information that is being used, and keep unused information on disk, moving info back and forth as needed.
- Ideally: we have a memory system with the performance of main memory and the cost/capacity of disk!

Demand Paging



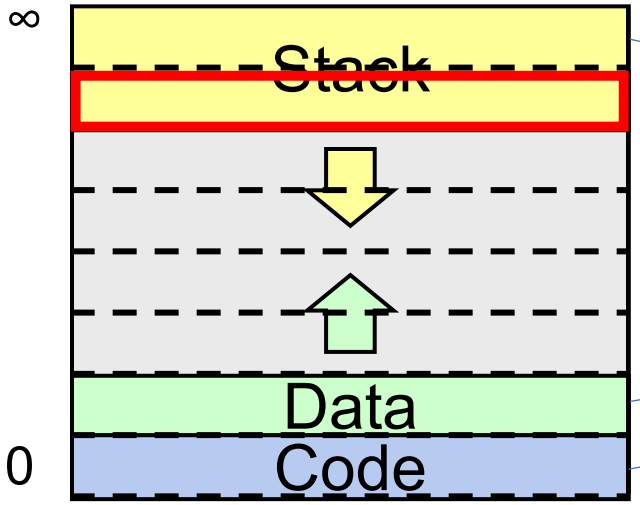
Process A Virtual Address Space



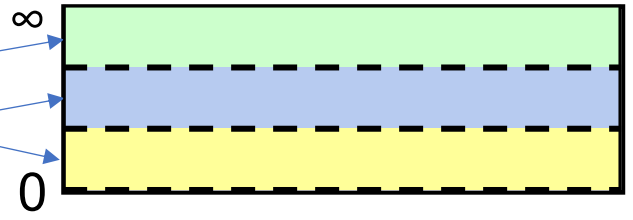
Physical Address Space

	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

Demand Paging



Process A Virtual Address Space

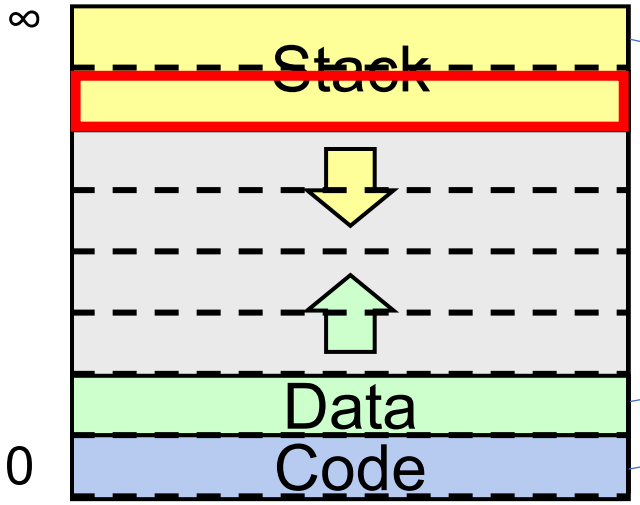


Physical Address Space

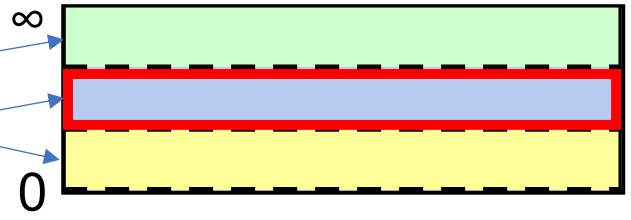
	Physical page #	WR?	PR?
7	0	1	1
6	X	X	0
5	X	X	0
4	X	X	0
3	X	X	0
2	X	X	0
1	2	0	1
0	1	0	1

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

Demand Paging

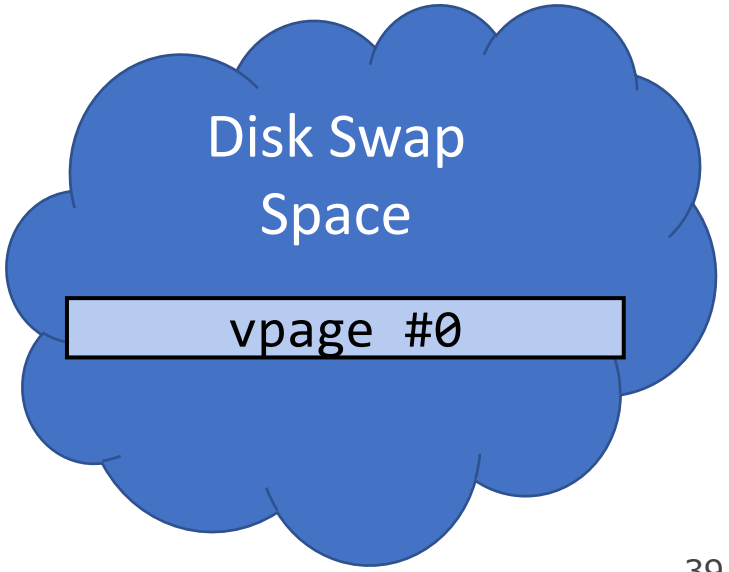


Process A Virtual Address Space



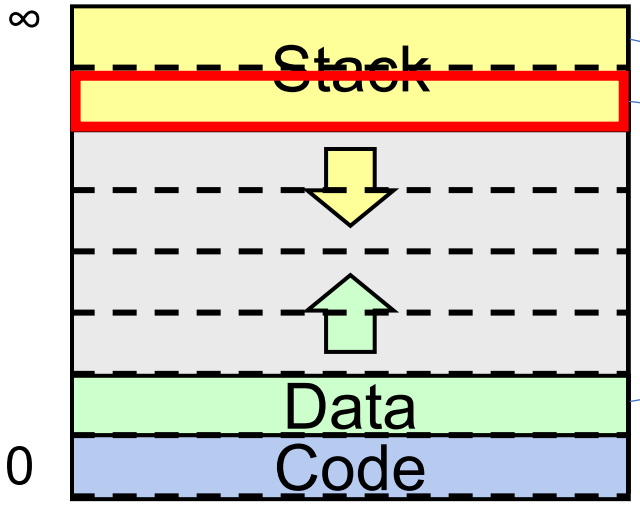
Physical Address Space

	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

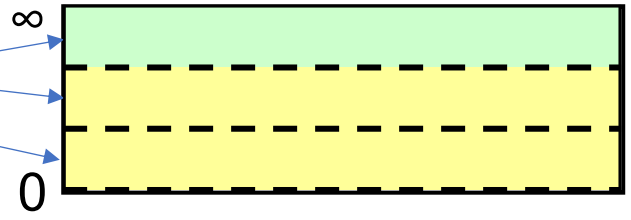


1. Pick an existing physical page and swap it to disk.

Demand Paging

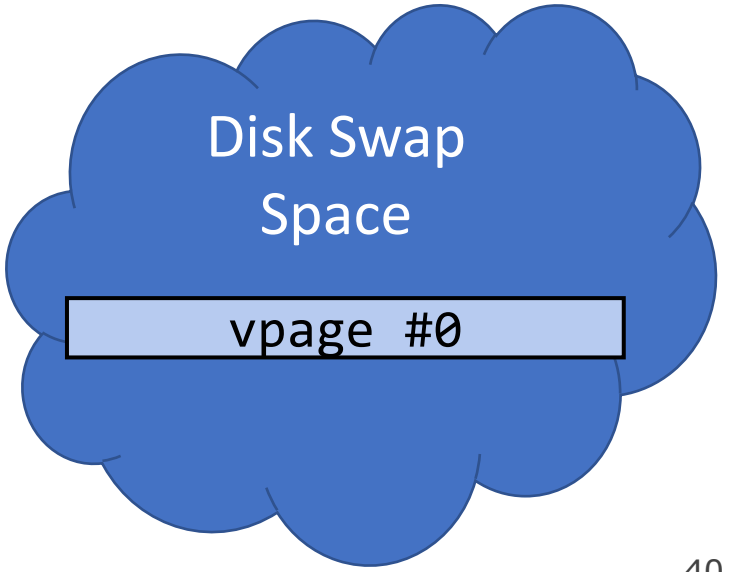


Process A Virtual Address Space



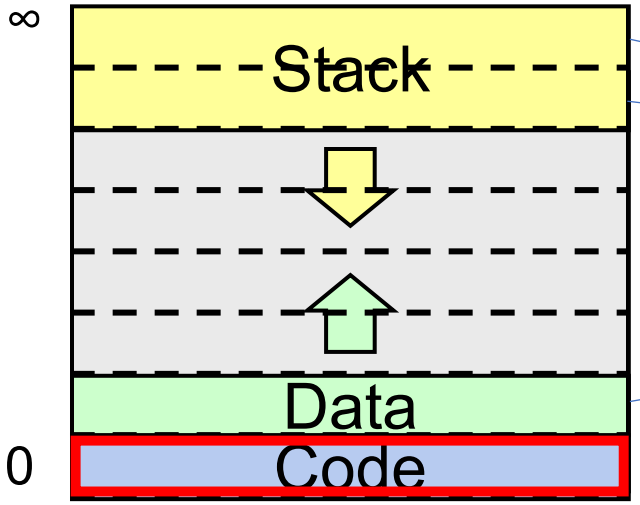
Physical Address Space

	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

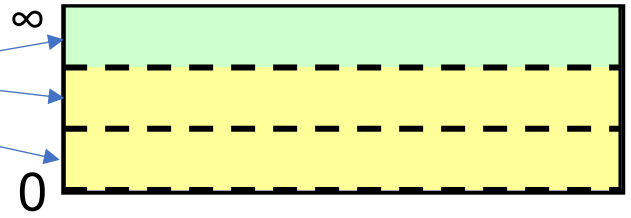


2. Map this physical page to the new virtual page.

Demand Paging

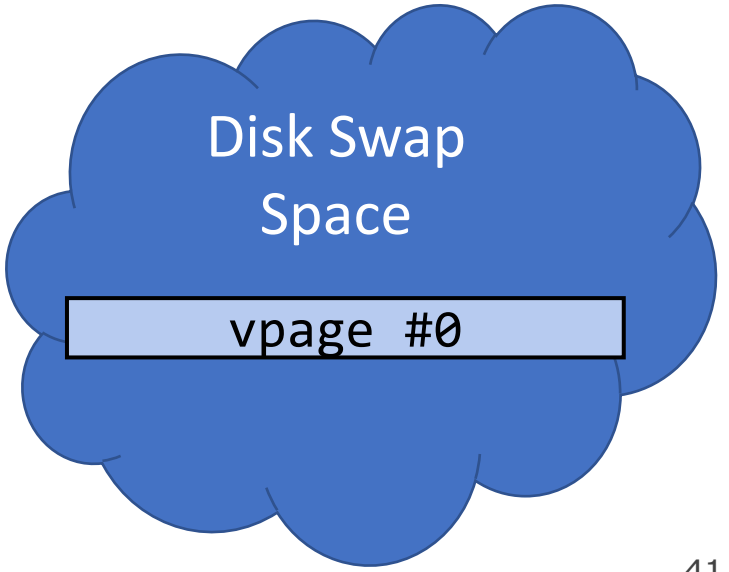


Process A Virtual Address Space



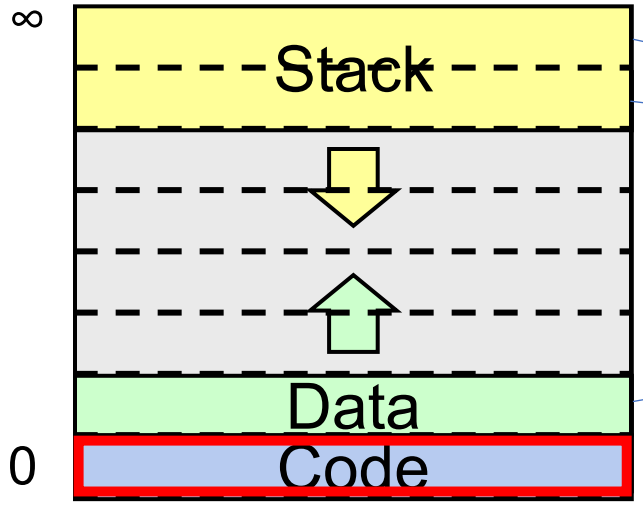
Physical Address Space

	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

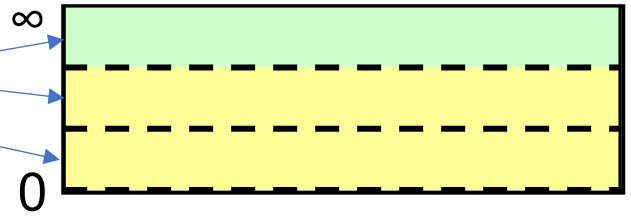


1. We look in the page map and see it's not present.

Demand Paging

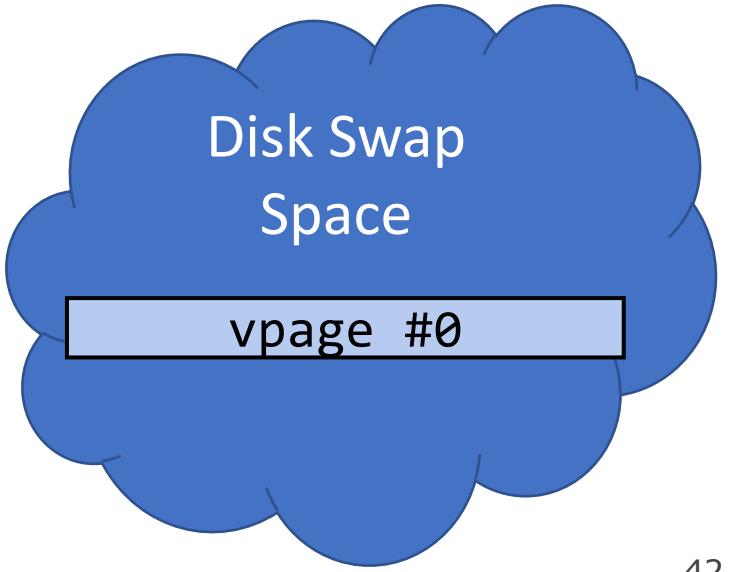


Process A Virtual Address Space



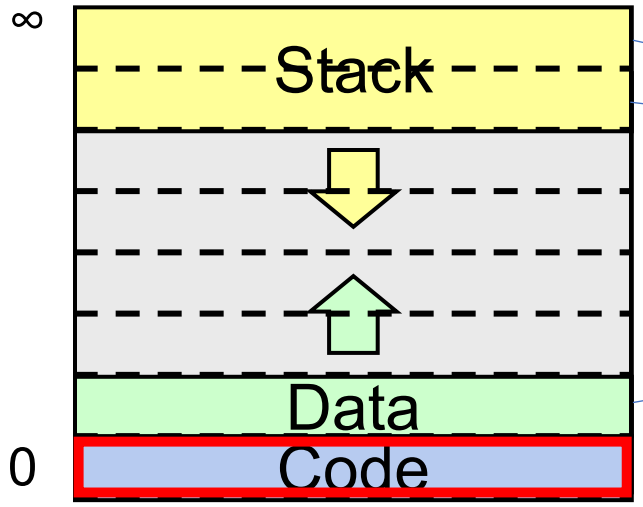
Physical Address Space

	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

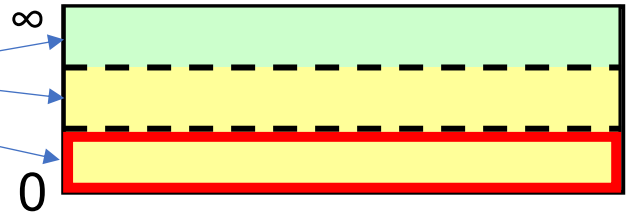


2. But we look in the disk swap and see it is stored there, so we load it back in (kicking another page if needed).

Demand Paging

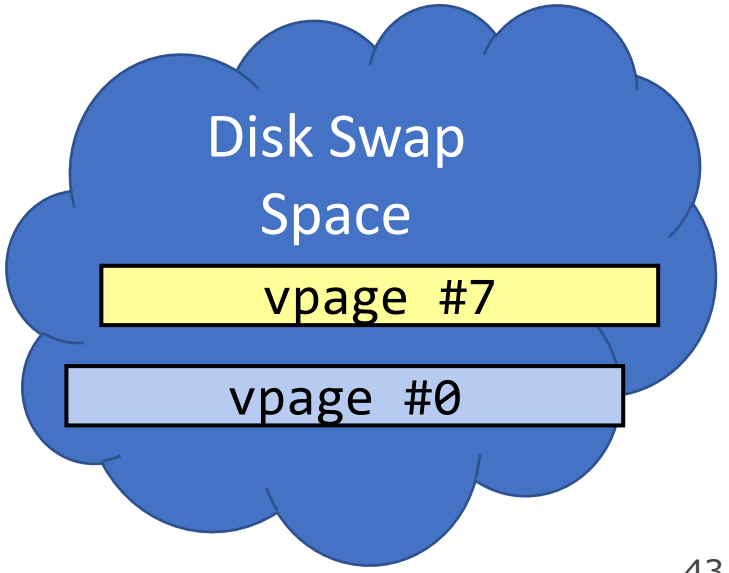


Process A Virtual Address Space



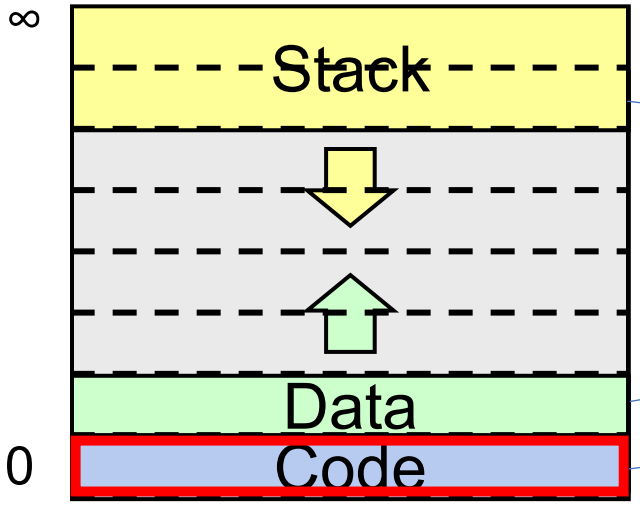
Physical Address Space

	Physical page #	WR?	PR?
7	0	1	0
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

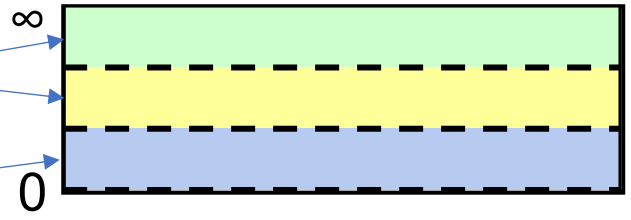


2. But we look in the disk swap and see it is stored there, so we load it back in (kicking another page if needed).

Demand Paging

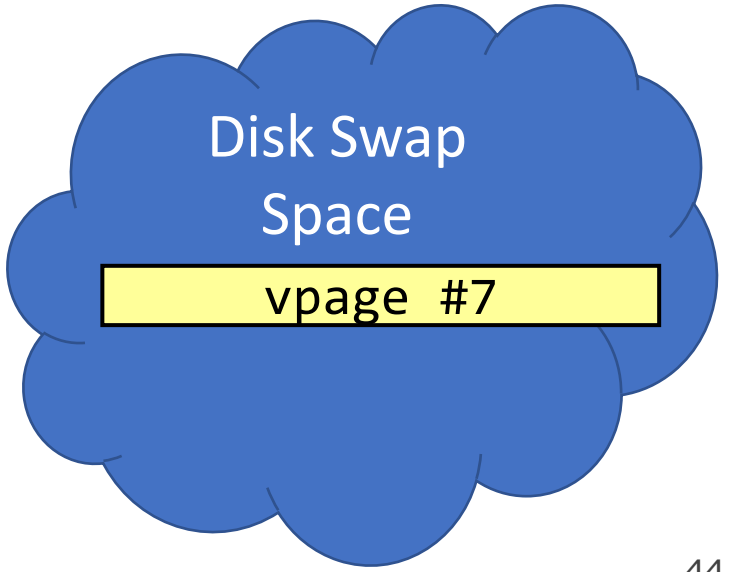


Process A Virtual Address Space



Physical Address Space

	Physical page #	WR?	PR?
7	0	1	0
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	0	0	1



2. But we look in the disk swap and see it is stored there, so we load it back in (kicking another page if needed).

Page Faults

- If a program references an address that's not present, it triggers a *page fault*
- If the page is in the disk swap space, the OS finds a free physical page, reads the page in from disk to that physical page, updates the page map, and resumes the program.

This can provide big benefits – but what potential scenario would lead demand paging to slow the system way down?

Respond on PollEv: pollev.com/cs111
or text CS111 to 22333 once to join.

What potential scenario would lead demand paging to slow the system way down?

Thrashing

If the pages being actively used don't all fit in memory, the system will spend all its time reading and writing pages to/from disk and won't get much work done.

- Called *thrashing*
- The page we kick to disk will be needed very soon, so we will bring it back and kick another page, which will be needed very soon, etc….
- Progress of the program will make it look like access time of memory is as slow as disk, rather than disks being as fast as memory. ☹
- With personal computers, users can notice thrashing and kill some processes

Page Replacement

If we need another physical page but all memory is used, which page should we throw out? How do we pick?

- Random? (works surprisingly well!)
- FIFO? (throw out page that's been in memory the longest) – fairness
- Would be nice if we could pick page whose next access is farthest in the future, but we'd need to predict the future…
- LRU (least-recently-used)? Replace page that was accessed the longest time ago.

Recap

- Recap: Paging so far
- Page Map Size
- Demand Paging

Lecture 24 takeaway: We can make memory appear larger than it is by swapping pages to disk when we need more space, and swapping them back later.

Next time: how to choose which pages to swap to disk (the clock algorithm).