CS111, Lecture 21
Virtual Memory Introduction

Optional reading:

Operating Systems: Principles and Practice (2nd Edition): Chapter 8

😷 masks recommended

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

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?*

Why is answering this question important?

- We can understand one of the most “magical” responsibilities of OSes – making one set of memory appear as several!
- Exposes challenges of allowing multiple processes share memory while remaining isolated
- Allows us to understand exactly what happens when a program accesses a memory address

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

CS111 Topic 4: Virtual Memory

Virtual Memory Introduction → Dynamic Address Translation → Paging → Demand Paging and the Clock Algorithm

Virtual Memory Introduction	Dynamic Address Translation	Paging	Demand Paging and the Clock Algorithm
Today	**Lecture 22**	**Lecture 23**	**Lecture 24**

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

Learning Goals

- Understand what impact virtual memory has on our programs
- Learn about the goals of virtual memory
- Reason about the tradeoffs in implementing virtual memory

Plan For Today

- Introducing virtual memory
- Single-tasking
- Goals of sharing memory
- Load-time relocation
- Dynamic address translation

```
cp -r /afs/ir/class/cs111/lecture-code/lect21 .
```

Plan For Today

- **Introducing virtual memory**
- Single-tasking
- Goals of sharing memory
- Load-time relocation
- Dynamic address translation

```
cp -r /afs/ir/class/cs111/lecture-code/lect21 .
```

Introducing Virtual Memory

Virtual memory is a mechanism that allows multiple processes to simultaneously use system memory.

- Program addresses are *virtual* (fake) – the OS maps them to *physical* (real) addresses in memory.
- The OS must keep track of virtual -> physical "translations" and translate every memory access.
- The OS doesn't need to map all virtual addresses unless needed – it can give programs new memory on the fly
- The OS can even temporarily kick memory contents to disk until a program needs it again.
- Example of **virtualization** – making one thing look like another, or many of them

Introducing Virtual Memory

Virtual memory is a mechanism that allows multiple processes to simultaneously use system memory.

Three key questions:

- Why do we even need to have the OS intercepting memory addresses?
- How does the OS translate from virtual to physical addresses?
- What are the tradeoffs in different virtual memory implementations?

Demo: Virtual Memory Implications

memory.c and **htop**

```
cp -r /afs/ir/class/cs111/lecture-code/lect21 .
```

Plan For Today

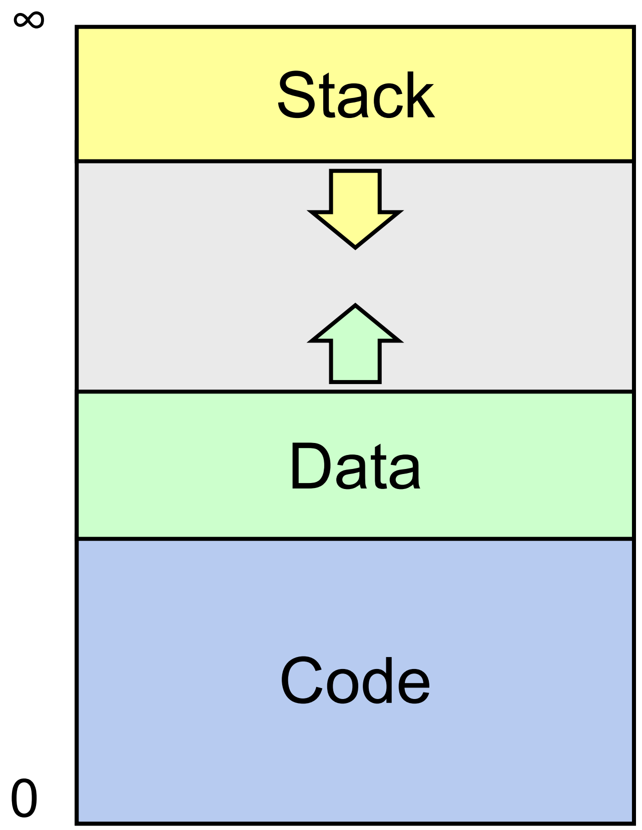
- Introducing virtual memory
- **Single-tasking**
- Goals of sharing memory
- Load-time relocation
- Dynamic address translation

```
cp -r /afs/ir/class/cs111/lecture-code/lect21 .
```

Single-Tasking

Let's start with a system that can just run one program at a time. What does memory look like?

- A process's memory is a collection of *segments* (sections)
- **Code** ("text") – program code
- **Data** – constants, heap
- **Stack** – stack frames for functions
- Stack grows down, heap grows up as more space is needed

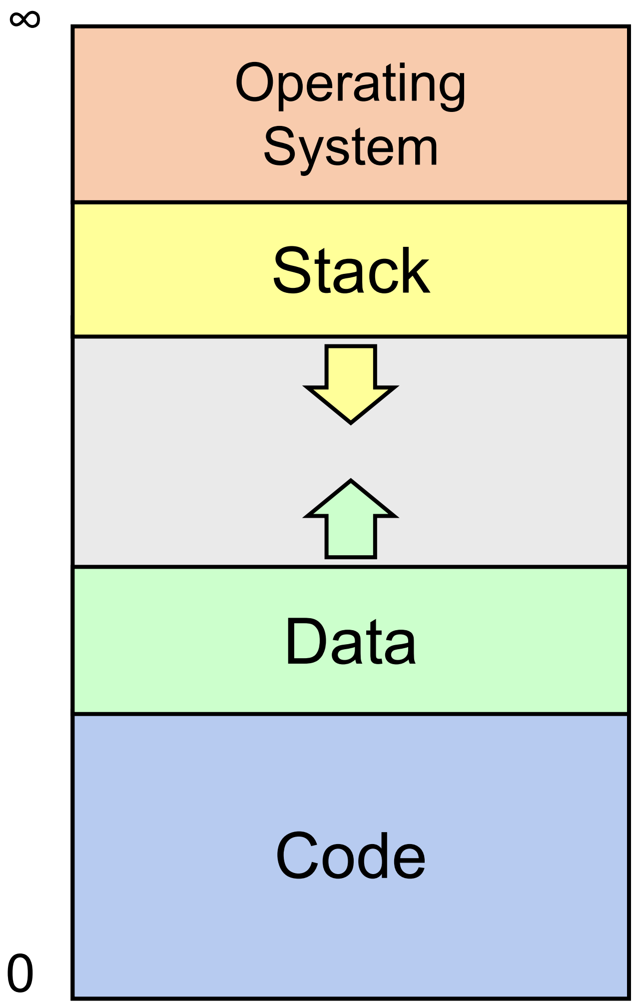


(for Unix/Linux – Windows essentially the same)

Single-Tasking

Let's start with a system that can just run one program at a time. What does memory look like?

- The OS also needs memory space!
- Reserve highest memory addresses for OS
- **Problem:** rogue programs could mess with OS memory, corrupt the system



Challenge: how can we split up memory to give each process space?

Plan For Today

- Introducing virtual memory
- Single-tasking
- **Goals of sharing memory**
- Load-time relocation
- Dynamic address translation

```
cp -r /afs/ir/class/cs111/lecture-code/lect21 .
```

Sharing Memory

What are our goals for sharing memory?

- **Multitasking** – allow multiple processes to be memory-resident at once
- **Transparency** – no process should need to know memory is shared. Each must run regardless of the number and/or locations of processes in memory.
- **Isolation** – processes must not be able to corrupt each other
- **Efficiency** (both of CPU and memory) – shouldn't be degraded badly by sharing

Idea #1: Let’s reserve contiguous blocks in memory for each process.

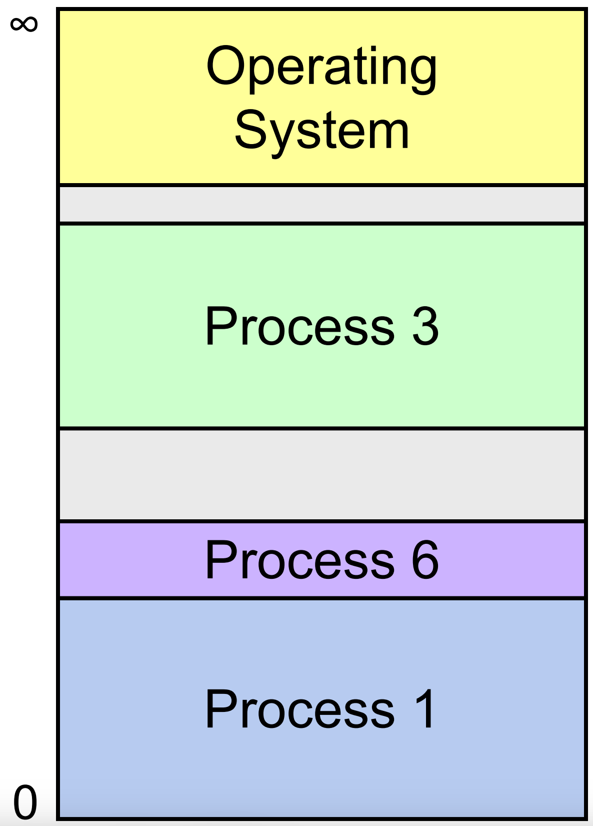
Plan For Today

- Introducing virtual memory
- Single-tasking
- Goals of sharing memory
- **Load-time relocation**
- Dynamic address translation

```
cp -r /afs/ir/class/cs111/lecture-code/lect21 .
```

Load-Time Relocation

- When a process is loaded to run, place it in a designated memory space.
- That memory space is for everything for that process – stack/data/code
- Interesting fact – when a program is compiled, it is compiled assuming its memory starts at address 0. Therefore, we must update its addresses when we load it to match its real starting address.
- Use first-fit or best-fit allocation to manage available memory.



What are the problems with this approach?

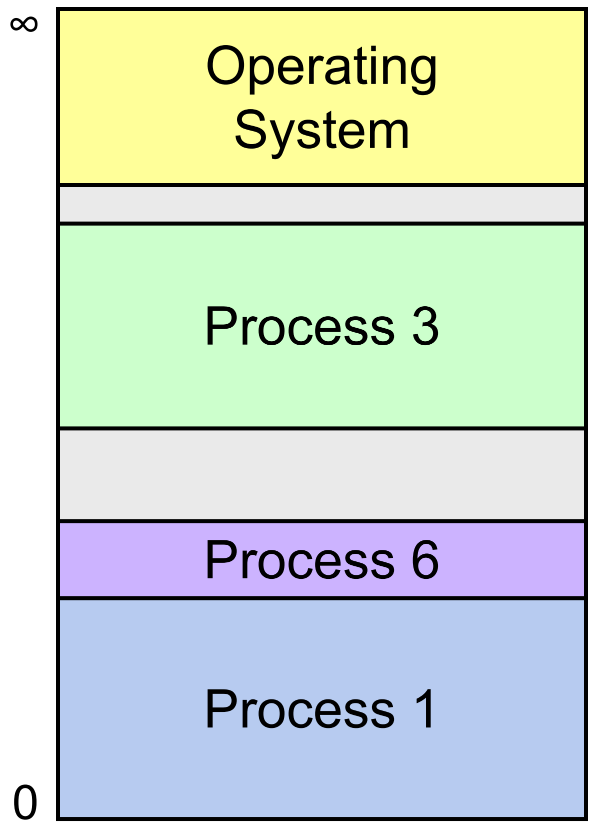
Respond with your thoughts on PollEv:
pollev.com/cs111 or text CS111 to 22333 once to join.

What are some problems with load-time relocation?

Load-Time Relocation

What are the problems with this approach?

- No isolation – one process can corrupt another or the OS
- Must decide process memory size ahead of time
- Challenges with allocating memory for new processes – memory fragmentation
- Can't grow regions if adjacent space is in use
- Can't move once we load the process
- Need to update pointers in executable before running



Idea #2: What if, instead of translating addresses when a program is loaded, the OS intercepted every memory reference and handled it?

Plan For Today

- Introducing virtual memory
- Single-tasking
- Goals of sharing memory
- Load-time relocation
- **Dynamic address translation**

```
cp -r /afs/ir/class/cs111/lecture-code/lect21 .
```

Dynamic Address Translation

Let's have the OS intercept every memory reference a process makes.

- The OS can prohibit processes from accessing certain addresses (e.g. OS memory or another process's memory)
- Gives the OS lots of flexibility in managing memory

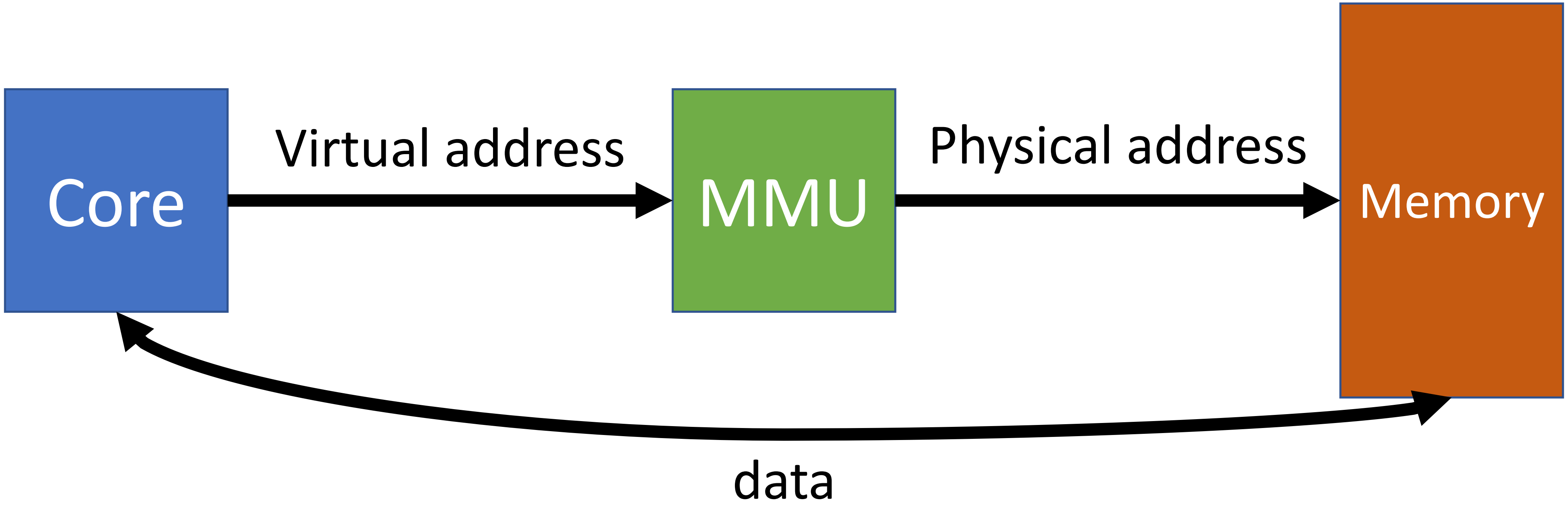
Problem: intercepting and translating *every* memory reference is expensive! How can we do this?

Solution: hardware support

Dynamic Address Translation

We will add a *memory management unit* (MMU) in hardware that changes addresses dynamically during every memory reference.

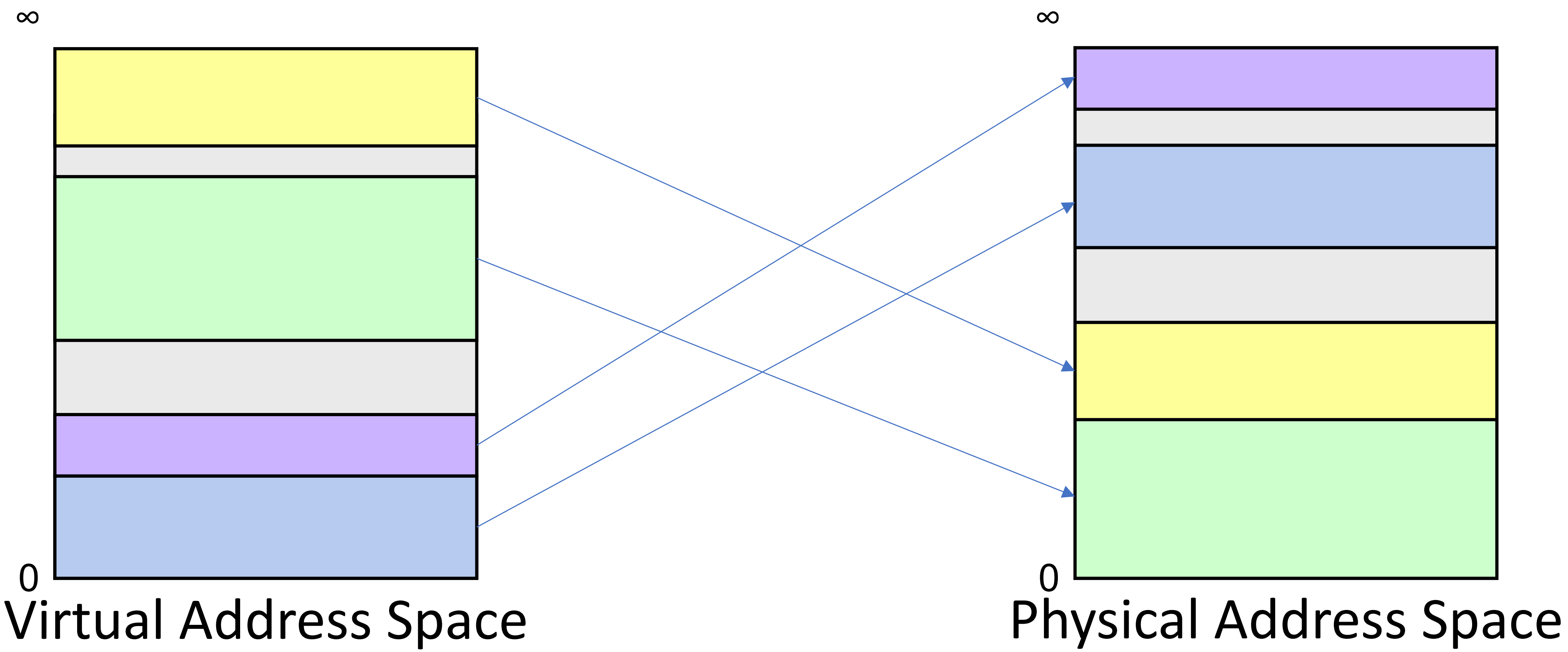
- *Virtual address* is what the program sees
- *Physical address* is the actual location in memory



Dynamic Address Translation

Key Idea: there are now *two views of memory,* and they can look very different:

- **Virtual address space** is what the program sees
- **Physical address space** is the actual allocation of memory



Dynamic Address Translation

- **Transparency** – virtual addresses allow a program's view of memory to be different than the real view; doesn't know its memory is e.g., split up.
- **Isolation** – OS intercepts memory references and can prevent rogue accesses

Key question: how does the MMU translate from a virtual address to a physical address? *We'll see several different approaches over the next few lectures.*

Plan For Today

- Introducing virtual memory
- Single-tasking
- Goals of sharing memory
- Load-time relocation
- Dynamic address translation

Next time: more about dynamic address translation

Lecture 21 takeaway: Virtual memory is a mechanism that allows multiple processes to simultaneously use system memory. There are two views of memory: virtual and physical. The hardware MMU translates from virtual to physical addresses.