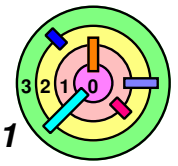
Ch 1: Introduction

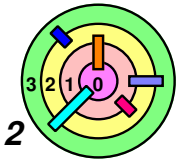
Bill Cheng

http://merlot.usc.edu/william/usc/



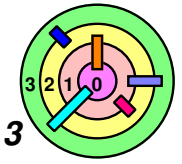
What are Operating Systems?

- **Possible definitions:**
  - **the code that {Microsoft, Apple, Linus, Google} provides**
  - **the code that you didn't write**
  - **the code that runs in privileged mode**
  - **the code that makes things work**
  - **the code that makes things crash**
  - **etc.**



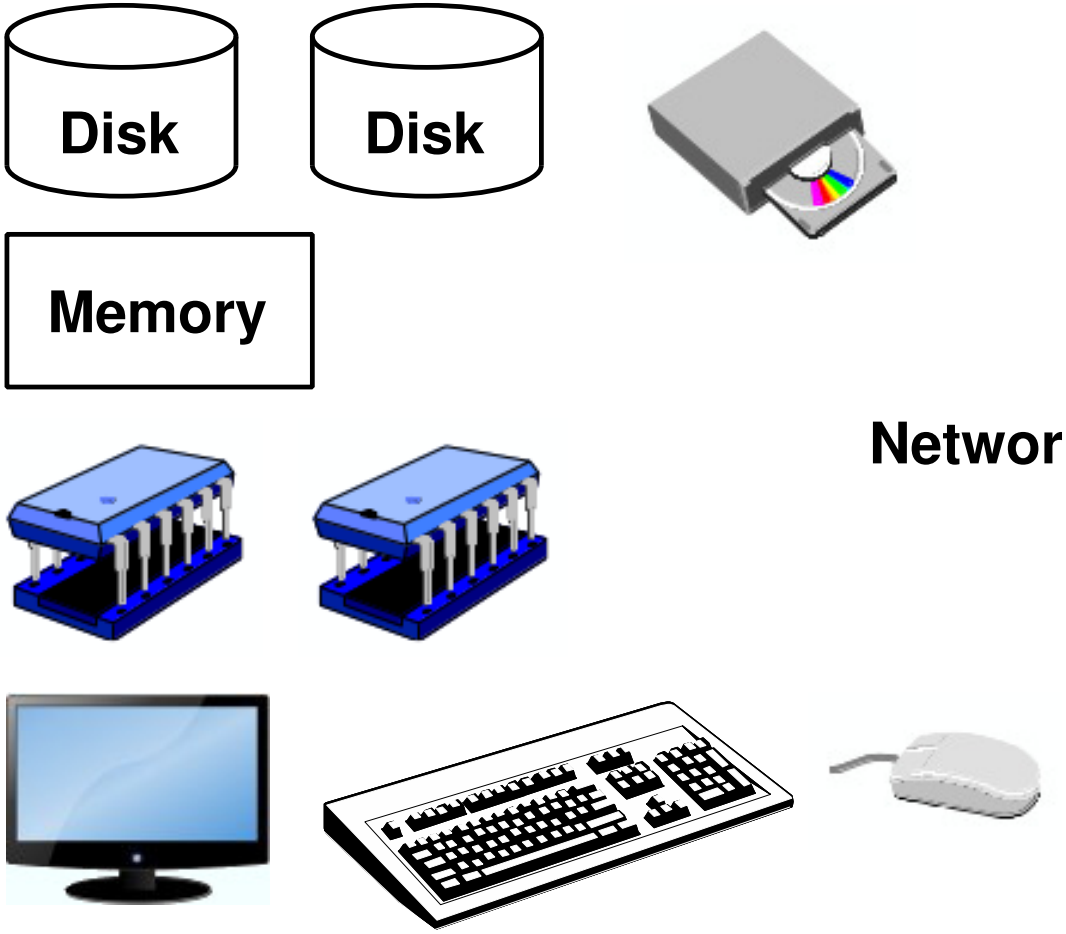
Operating Systems

- **Abstraction**
  - providing an "appropriate" interface for applications
  - but abstraction to what exactly? (next slide)
- **What's an "abstraction" anyway?**
  - think about "abstract data types" in a data structures class
    - it's data structures and associated functions to make something looks like it has some behavior
- **A list object has a `sort()` function**
  - *"objects"* is the word we use to mean *any* data types (primitive, data structures, pointers)
  - can a list really sort itself?
    - of course not
    - we need to put the list under some sort of an *"execution context"* in order to execute the `sort()` function
      - umm... what's a "context"?
      - well, it's hard to say exactly what it is at this time

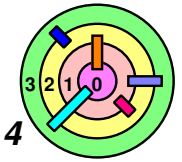


Hardware

- Hardware
  - disks
    - hard drives
    - optical drives
  - memory
  - processors
  - network
    - ethernet
    - modem
  - monitor
  - keyboard
  - mouse



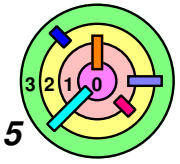
- Application programs are not allowed to use hardware directly
  - that's why we have to provide abstractions



OS Abstractions

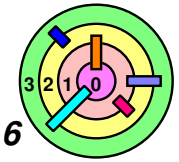
- Hardware
  - disks
  - memory
  - processors
  - network
  - monitor
  - keyboard
  - mouse
- Operating system
  - *files (file system)*
  - *programs (processes)*
  - *threads of control*
  - communication
  - windows, graphics
  - *input*
  - locator
- For those who knows about "processes", we use the word "program" to mean "process" in the introductory material

- Application programs are not allowed to use hardware directly
  - that's why we have to provide abstractions

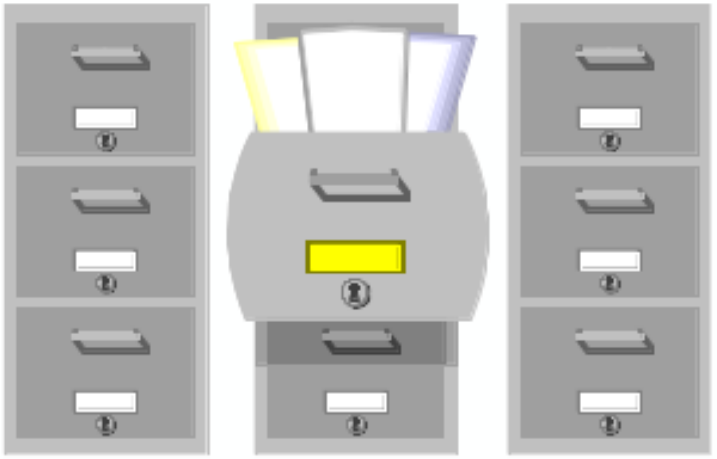


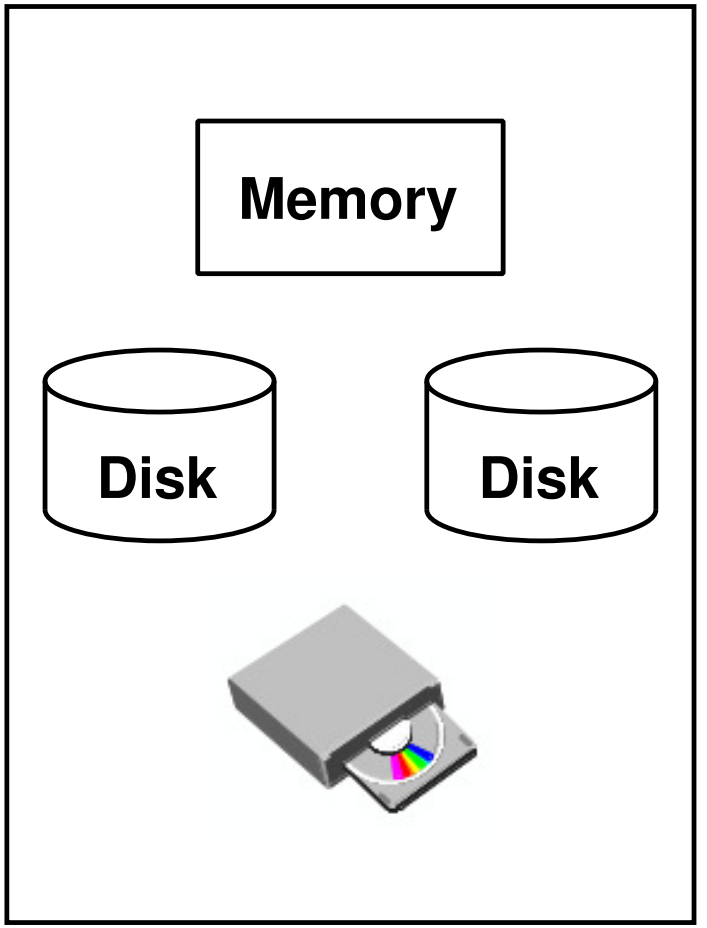
OS Abstractions

- The main focus of this class is about how to provide these abstractions
  - don't just "hack" it until it works
  - we will talk about OS *design principles* and show how these abstractions can be *implemented*
    - this class is more about the *fundamentals* and is not a "tech" class
- Concerns
  - performance
    - time, space, energy
  - sharing and resource management
  - failure tolerance
  - security
  - marketability



Abstraction Example: Files

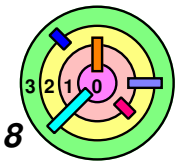




- It's nice to have a simple abstraction
- Abstraction did not come for free
  - it introduces problems that need to be solved and issues to be addressed

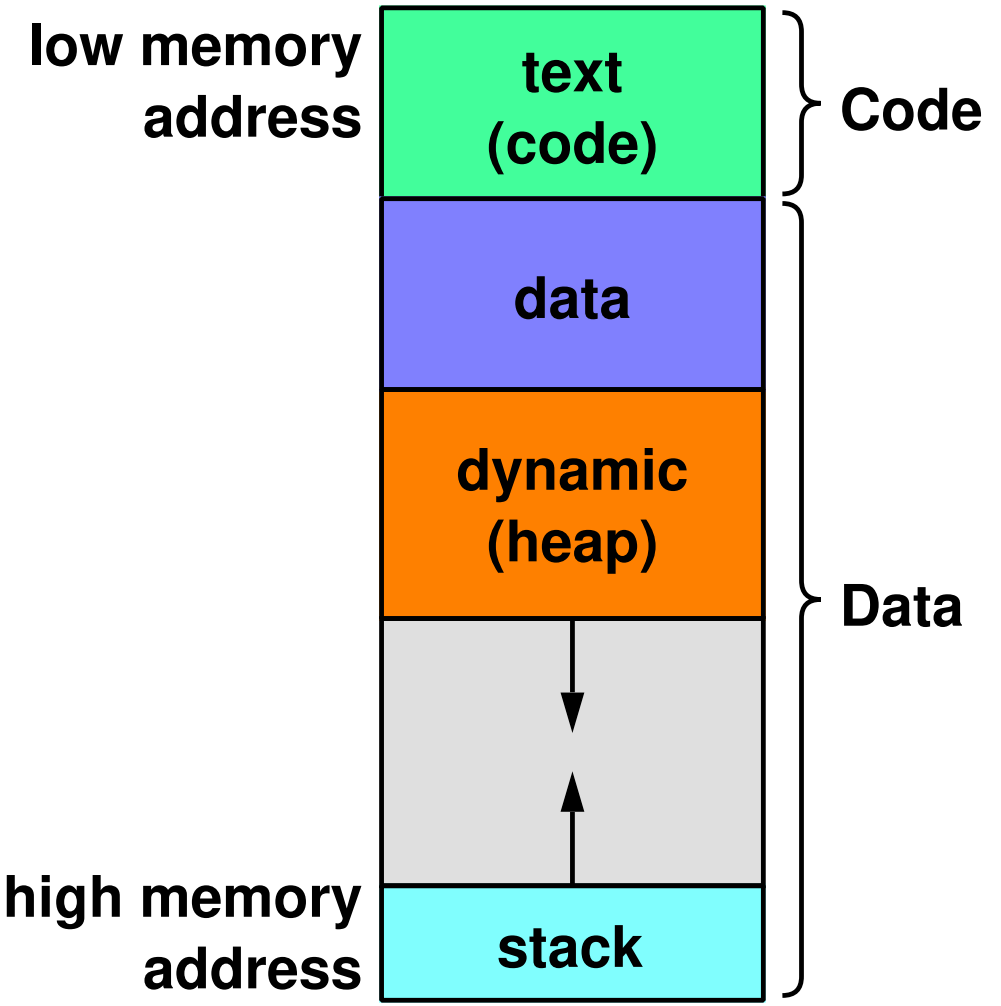
Issues With The Files Abstraction

- Naming
  - device-independence
- Allocating space on disk (permanent storage)
  - organized for fast access
  - minimize waste
- Shuffling data between disk and memory (high-speed temporary storage)
- Coping with crashes



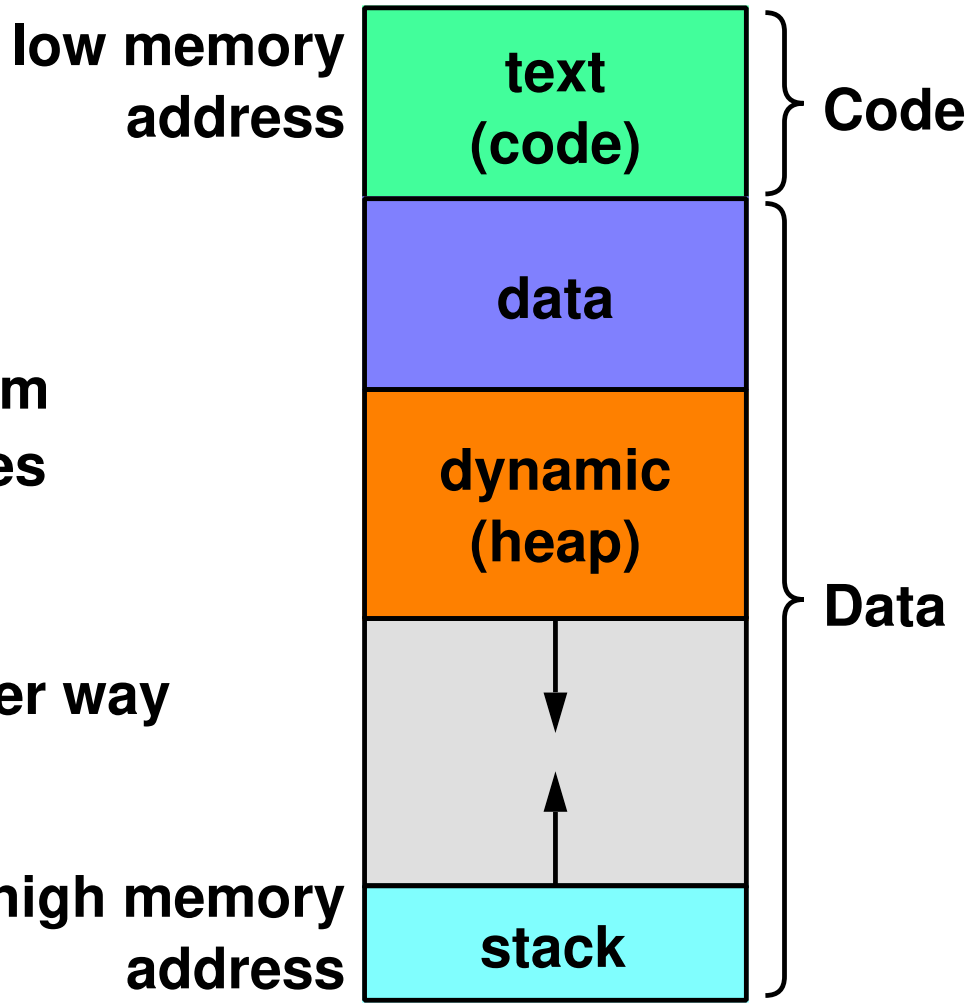
Abstraction Example: Programs

- Application programmers use the ***Address Space*** abstraction:

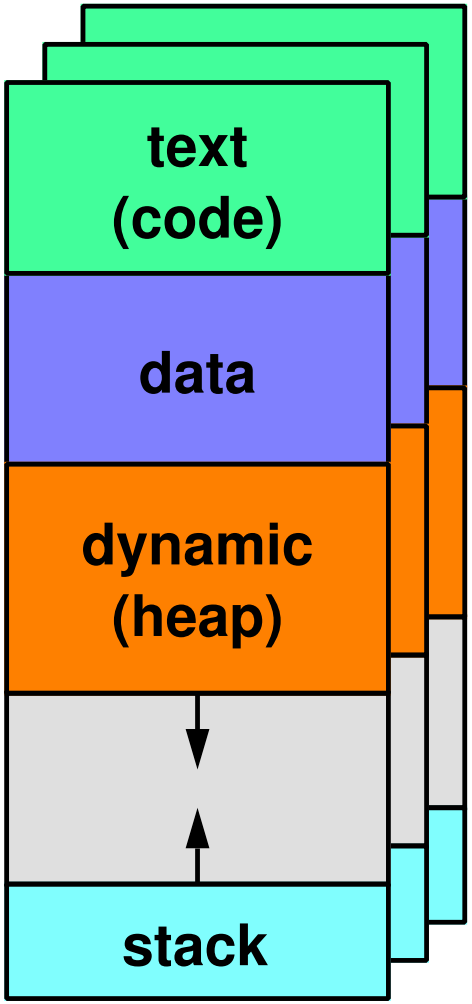


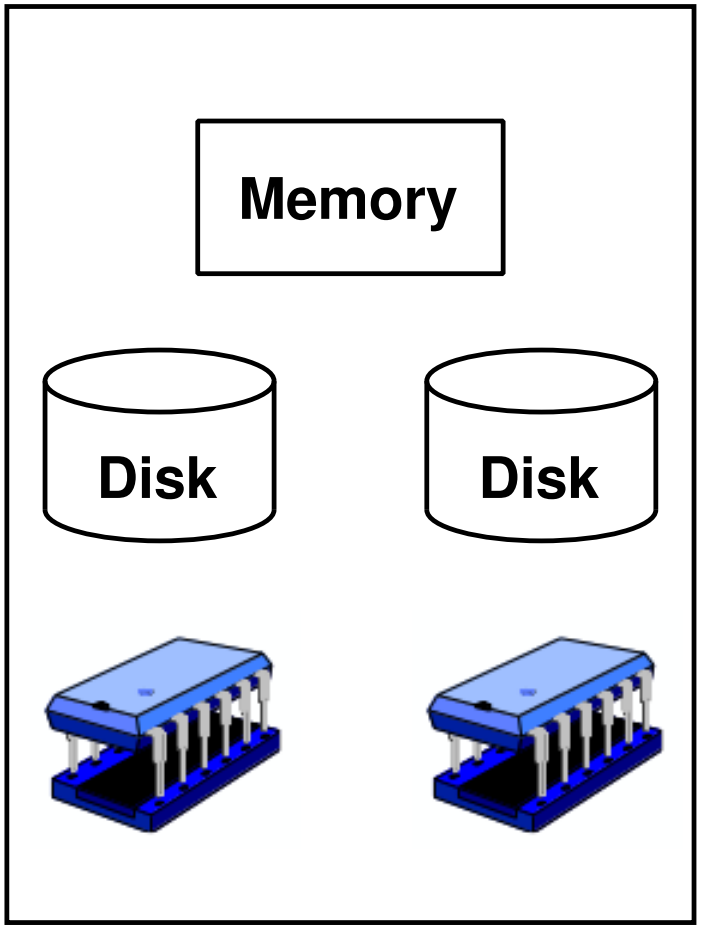
Abstraction Example: Programs

- Application programmers use the *Address Space* abstraction:
- *Very important:*
  - our address space is *up-side-down* (compared with the textbook)
    - low address at the top
    - high address at the bottom
      - memory layout matches an array
    - stack looks like a "stack"
  - our textbook does it the other way
- This is not the only possible memory layout
  - compiler decides!



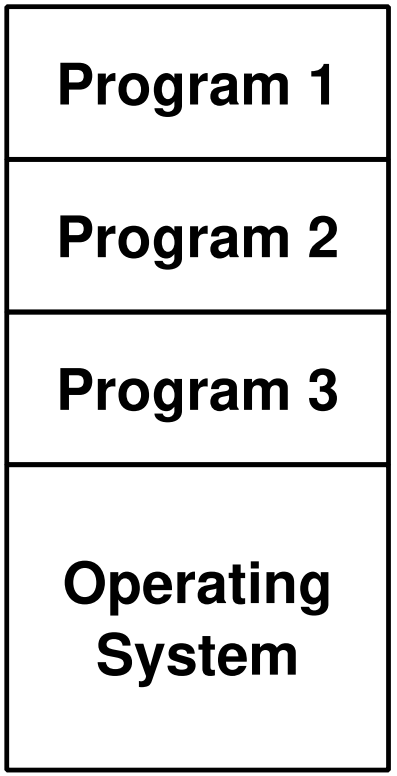
Abstraction Example: Programs



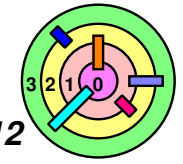


Application programmers do not have to worry about any *sharing* that's going on

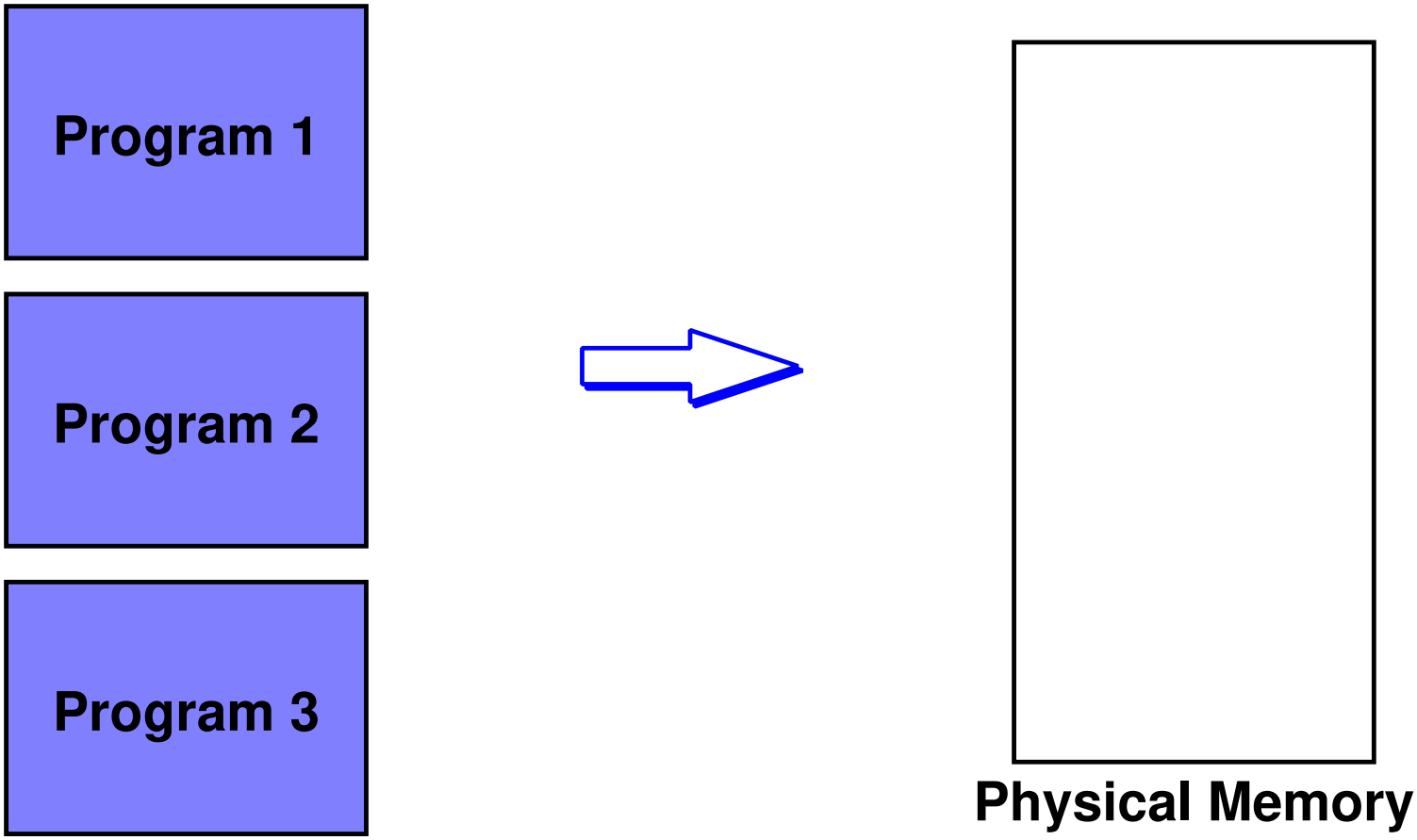
Memory Sharing Option 1



- Does not appear to be very flexible

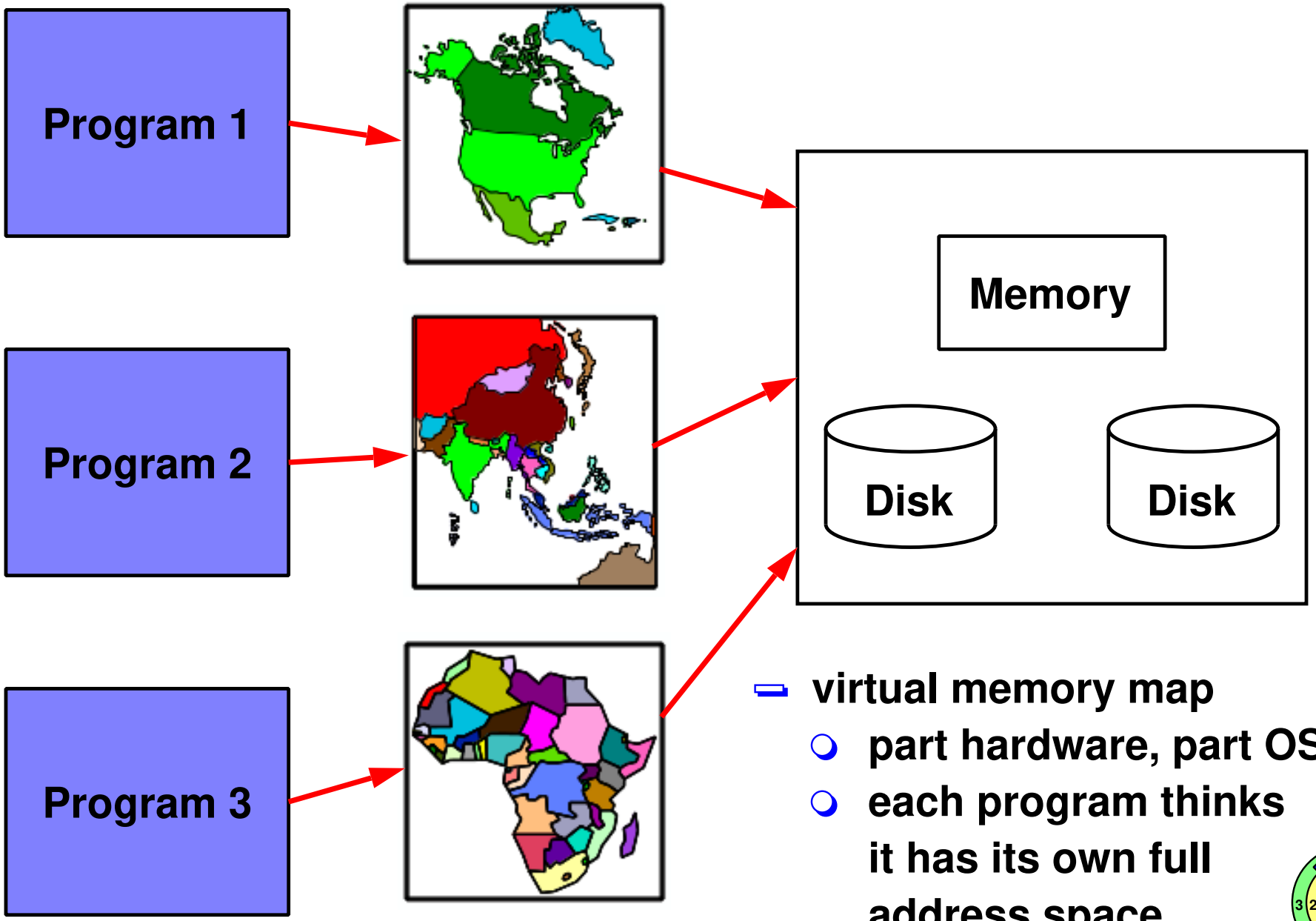


Memory Sharing Option 2



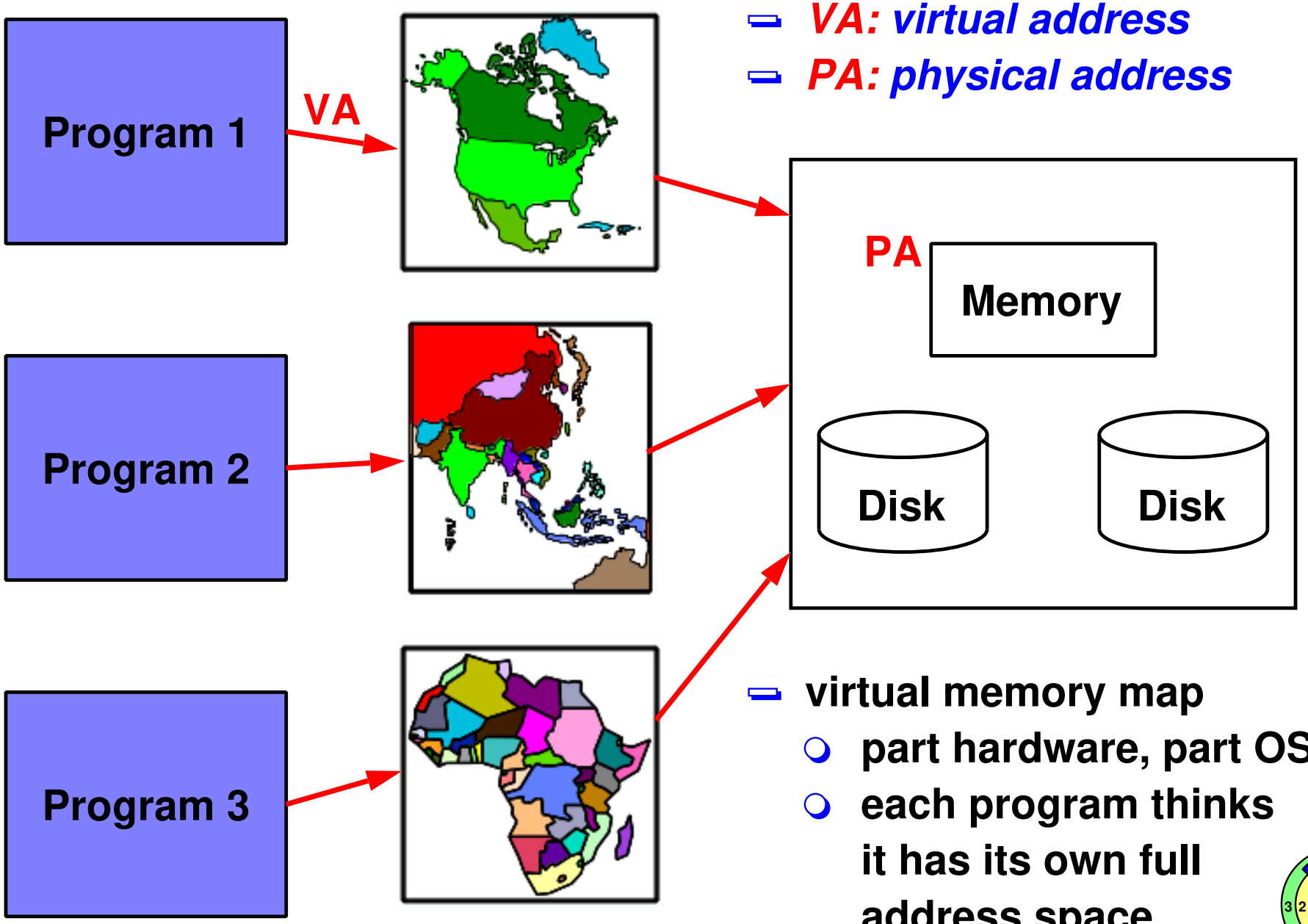
What if programs take up too much space (more than physical memory)?

Virtual Memory



- virtual memory map
  - part hardware, part OS
  - each program thinks it has its own full address space

Virtual Memory

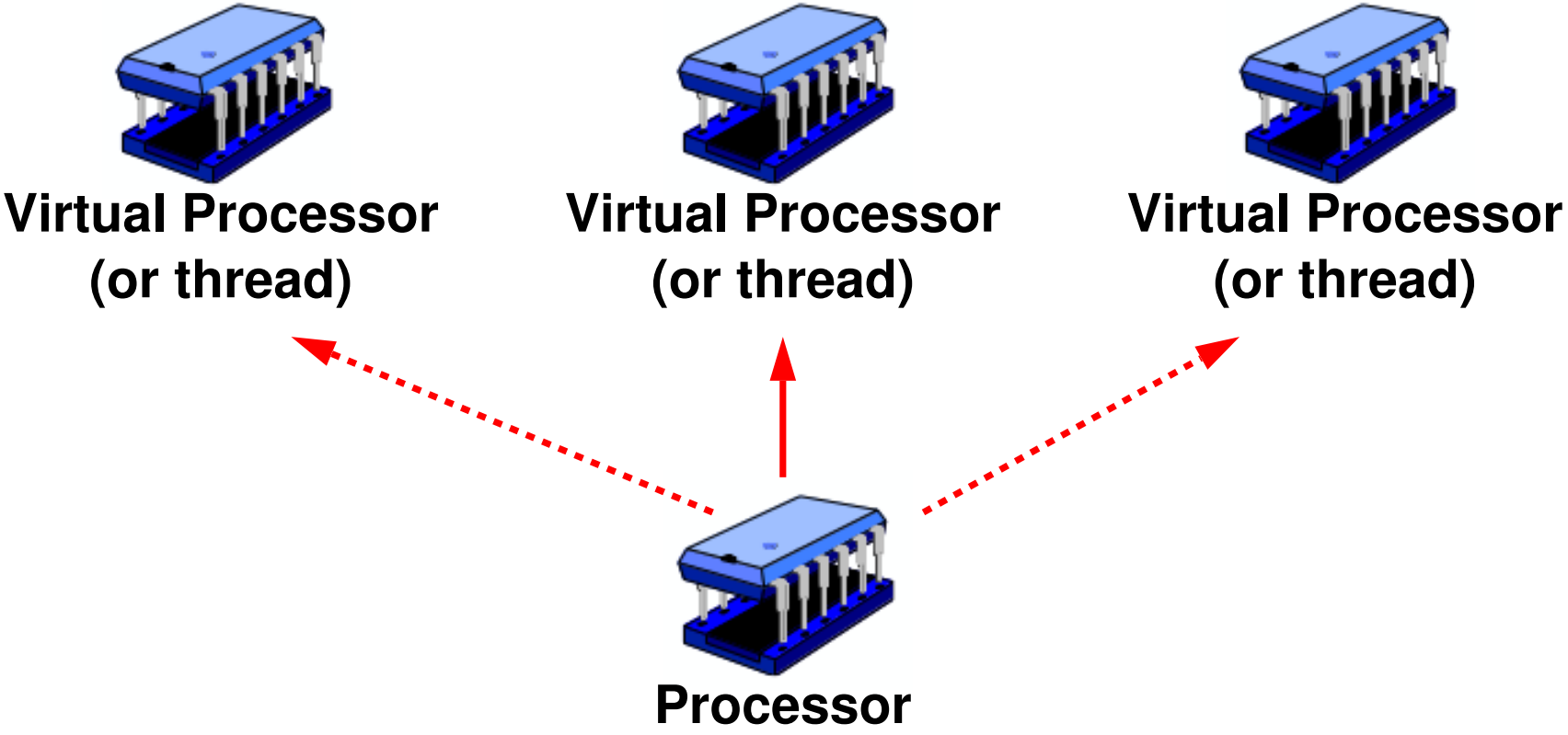


- *VA: virtual address*
- *PA: physical address*

- virtual memory map
  - part hardware, part OS
  - each program thinks it has its own full address space

Sharing of Processor: Concurrency

- If you only have one processor, how do you run multiple "programs" and every program thinks it owns the processor?
  - abstraction: threads (or "threads of execution")



- How do you *suspend* a thread *(save execution context)* so you can *resume* its execution later *(restore execution context)*?

Sharing of Processors: Parallelism

- What if you have a multicore processor or multiple processors?
  - we don't distinguish concurrency and parallelism in this class
  - can still use threads
    - but we need to worry about how well we do resource (processor) management/allocation