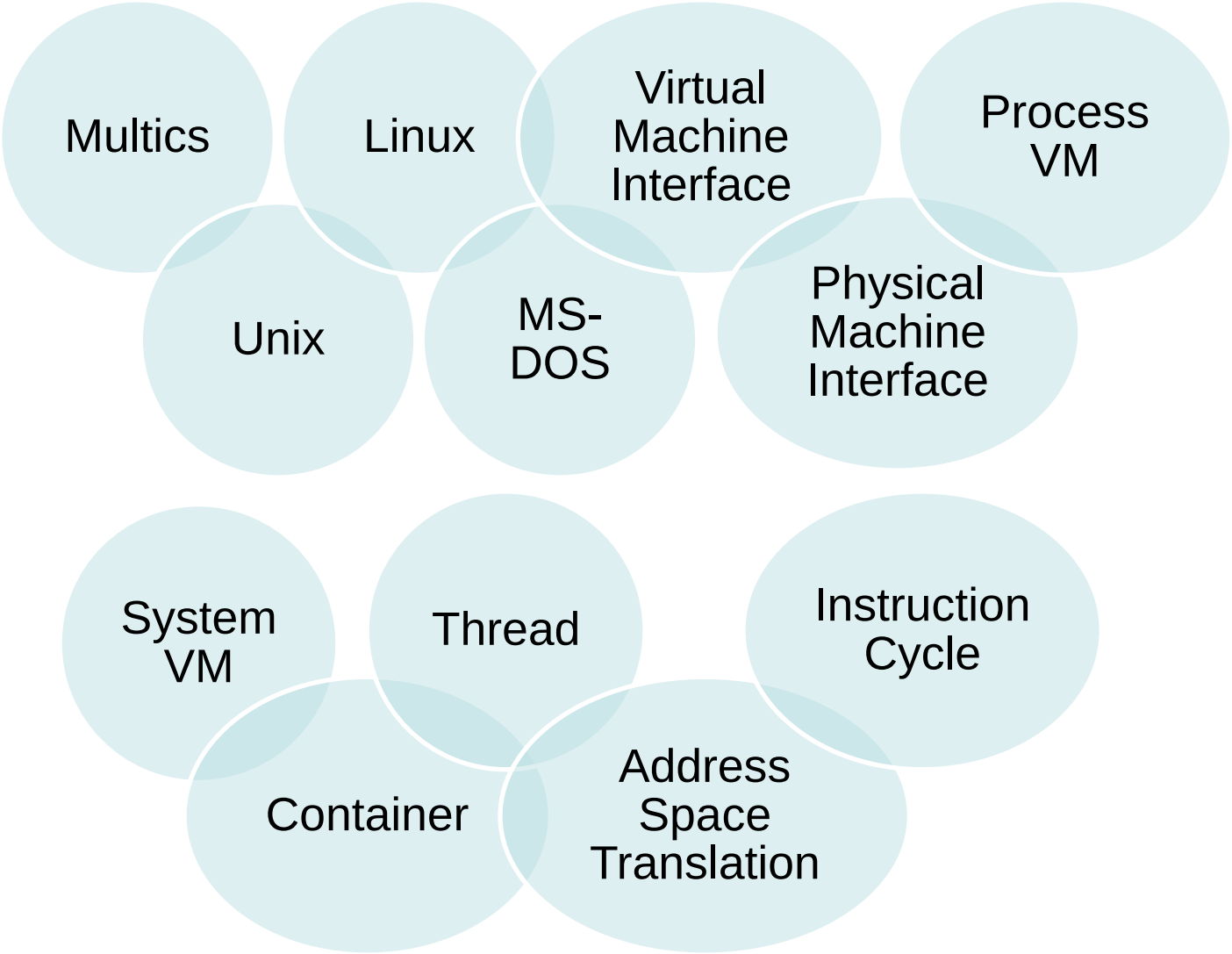
Lecturer: Dr. Michael J. May
Kinneret College

4 Concepts, Base & Bound

21 November 2024
Lecture 3

Slides adapted from John Kubiatowicz (UC Berkeley)

Concept Review



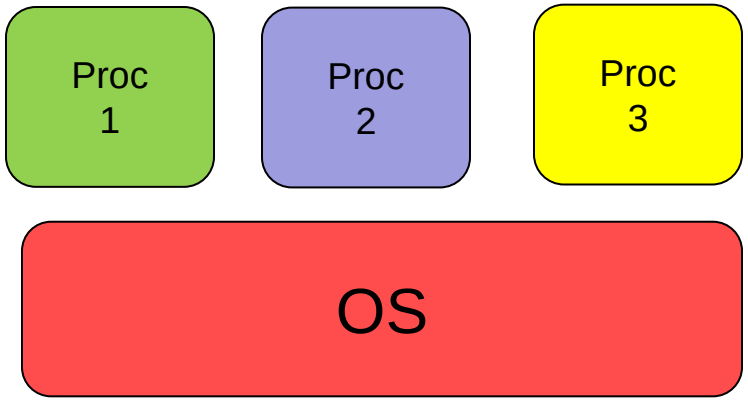
Today’s ideas

Topics for Today

- Four Concepts
  - Threads (last week)
  - Address space (continued)
  - Processes
  - Dual Mode Operation

- Protection Mechanisms
  - Address ranges: Base and Bound
- Interrupt and Process Context

Multiprogramming – Multiple Threads of Control



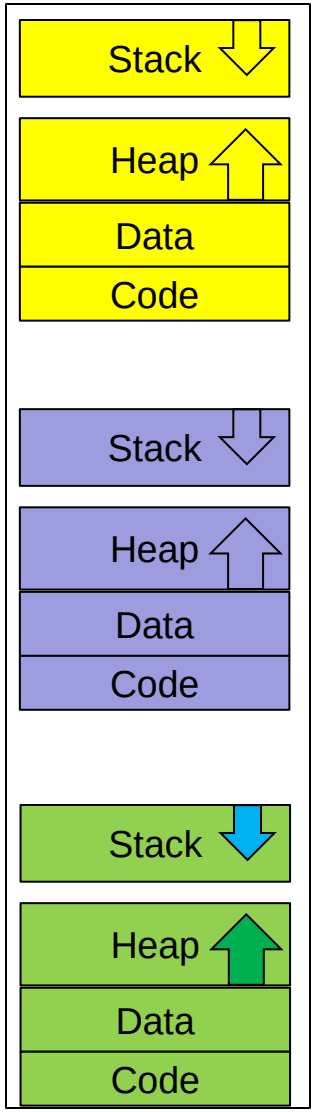


Image source: http://moziru.com/images/game-clipart-musical-chair-5.gif

Basic Problems of Concurrency

Basic problem of concurrency is resources:

- Hardware: single CPU, single DRAM, single I/O devices
- Multiprogramming API: processes think they have exclusive access to shared resources

OS must coordinate all activity

- Multiple processes, I/O interrupts, …
- How can it keep all these things straight?

Basic Idea: Use Virtual Machine abstraction

- Simple machine abstraction for processes
- Multiplex these abstract machines

Dijkstra did this for the "THE system"

- Few thousand lines vs 1 million lines in OS 360 (1K bugs)

Simple Multiprogramming (little protection)

All virtual CPUs share same non-CPU resources	Consequence of sharing	The (unprotected) model found in
• I/O devices the same • Memory the same	• Each thread can access the data of every other thread (good for sharing, bad for protection) • Threads can share instructions (good for sharing, bad for protection) • Can threads overwrite OS functions?	• Embedded applications • Windows 3.1/Early Macintosh (switch only with *yield*) • Windows 95-ME (switch with both *yield* and *timer*)

Clear divisions between entities

https://www.flickr.com/photos/tabor-roeder/5404004415/in/photostream/

Protection (stronger)

- OS protects itself from user programs
  - **Reliability**: Compromising the OS generally causes it to crash
  - **Security**: Limit the scope of what processes can do
  - **Privacy**: limit each process to the data it is permitted to access
  - **Fairness**: each should be limited to its appropriate share of system resources (CPU time, memory, I/O, etc)

Image source: http://denetimservices.com/wp-content/uploads/2013/05/bouncer.png

- OS protects User programs from one another

Protection

- Primary Mechanism: limit the translation from program address space to physical memory space
  - Can only touch what is mapped into process *address space*

Image source: https://media.istockphoto.com/

- Additional Mechanisms:
  - Privileged instructions, in/out instructions, special registers
  - syscall processing, subsystem implementation
    - (e.g., file access rights, etc)

So Far

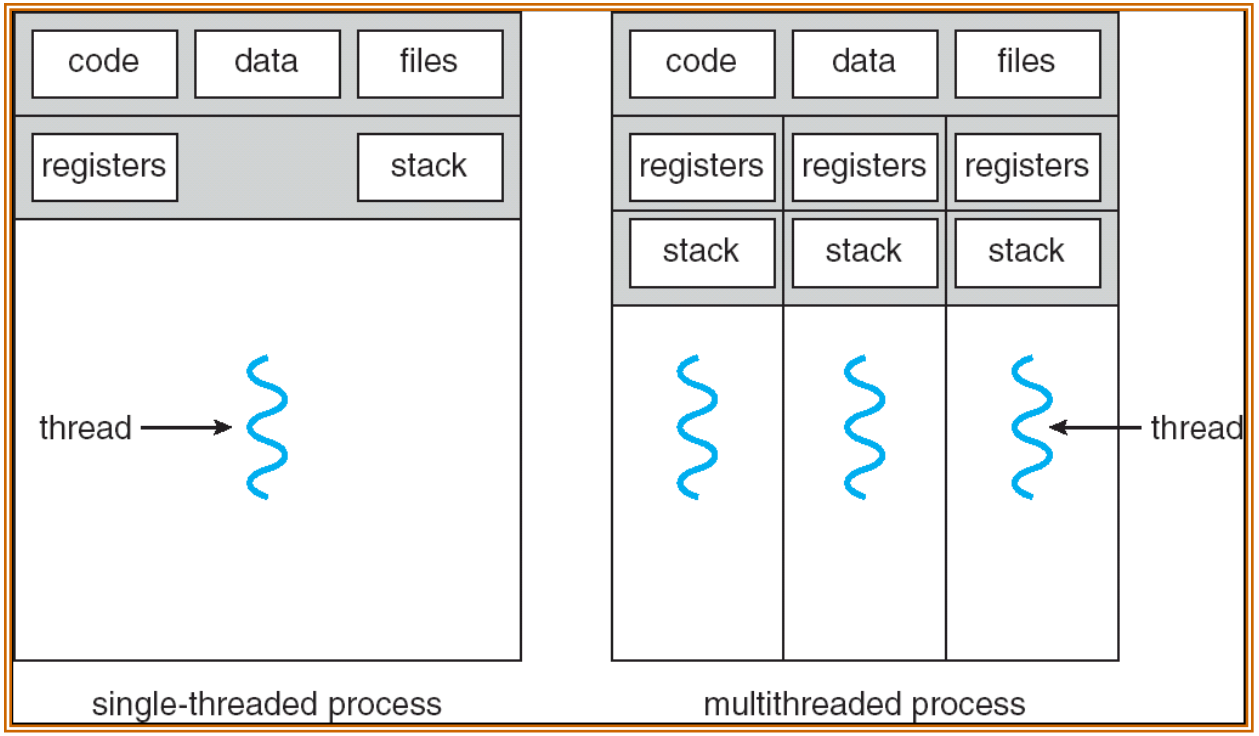
- Four Concepts
  - Threads (last week)
  - Address space (continued)
  - Processes
  - Dual Mode Operation

- Protection Mechanisms
  - Address ranges: Base and Bound
- Interrupt and Process Context

Third OS Concept: Process

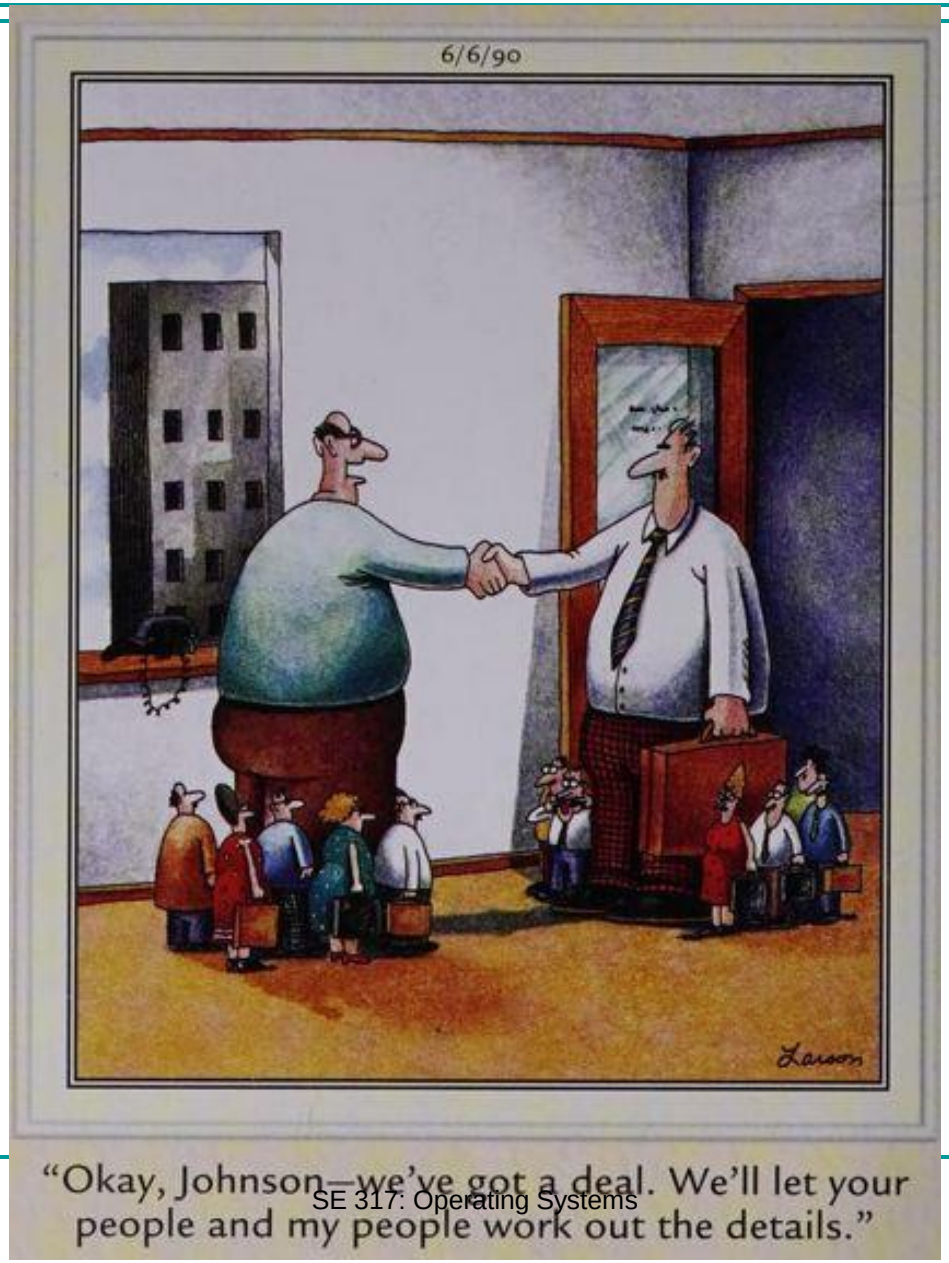
- **Process:** execution environment with Restricted Rights
  - **Address Space with One or More Threads**
  - Owns memory (address space), file descriptors, file system context, …
  - Encapsulate one or more threads sharing process resources
- Why **processes**?
  - Protected from each other! OS Protected from them!
  - Processes provides memory protection, group threads
- Tradeoff between protection (process) and efficiency (thread)
  - Communication easier *within* a process
  - Communication harder *between* processes
- Application instance consists of one or more processes

Single and Multithreaded Processes



- Threads encapsulate concurrency: “Active” component
- Address spaces encapsulate protection: “Passive” part
  - Keeps buggy program from trashing the system
- Why have multiple threads per address space?

©The Far Side by Gary Larson



So Far

- Four Concepts
  - Threads (last week)
  - Address space (continued)
  - Processes
  - Dual Mode Operation

- Protection Mechanisms
  - Address ranges: Base and Bound
- Interrupt and Process Context

Fourth OS Concept: Dual Mode Operation

- Hardware provides at least two modes:
  - "Kernel" mode (or "supervisor" or "protected")
  - "User" mode: Normal programs executed
- What hardware needed to support "dual mode" operation?

Bit of state (user/system mode bit)

Certain operations / actions only permitted in system/kernel mode

- In user mode they fail or trap

User → Kernel transition *sets* system mode and saves the user PC

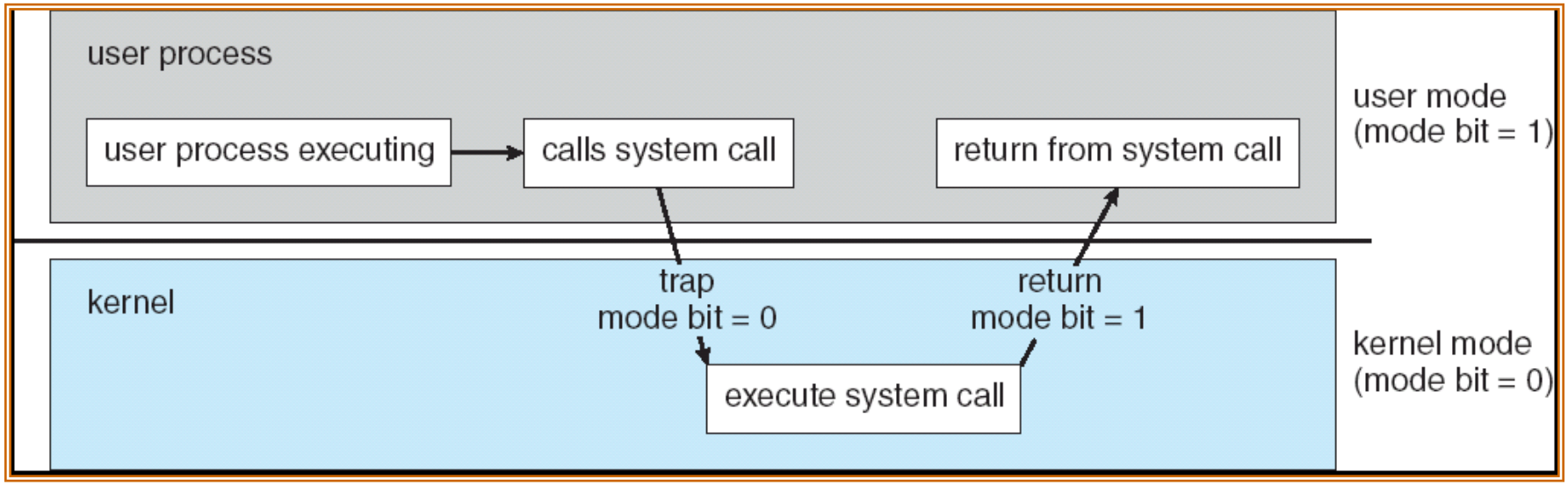
- Operating system code carefully puts aside user state then performs the necessary operations

Kernel → User transition *clears* system mode and restores appropriate user PC

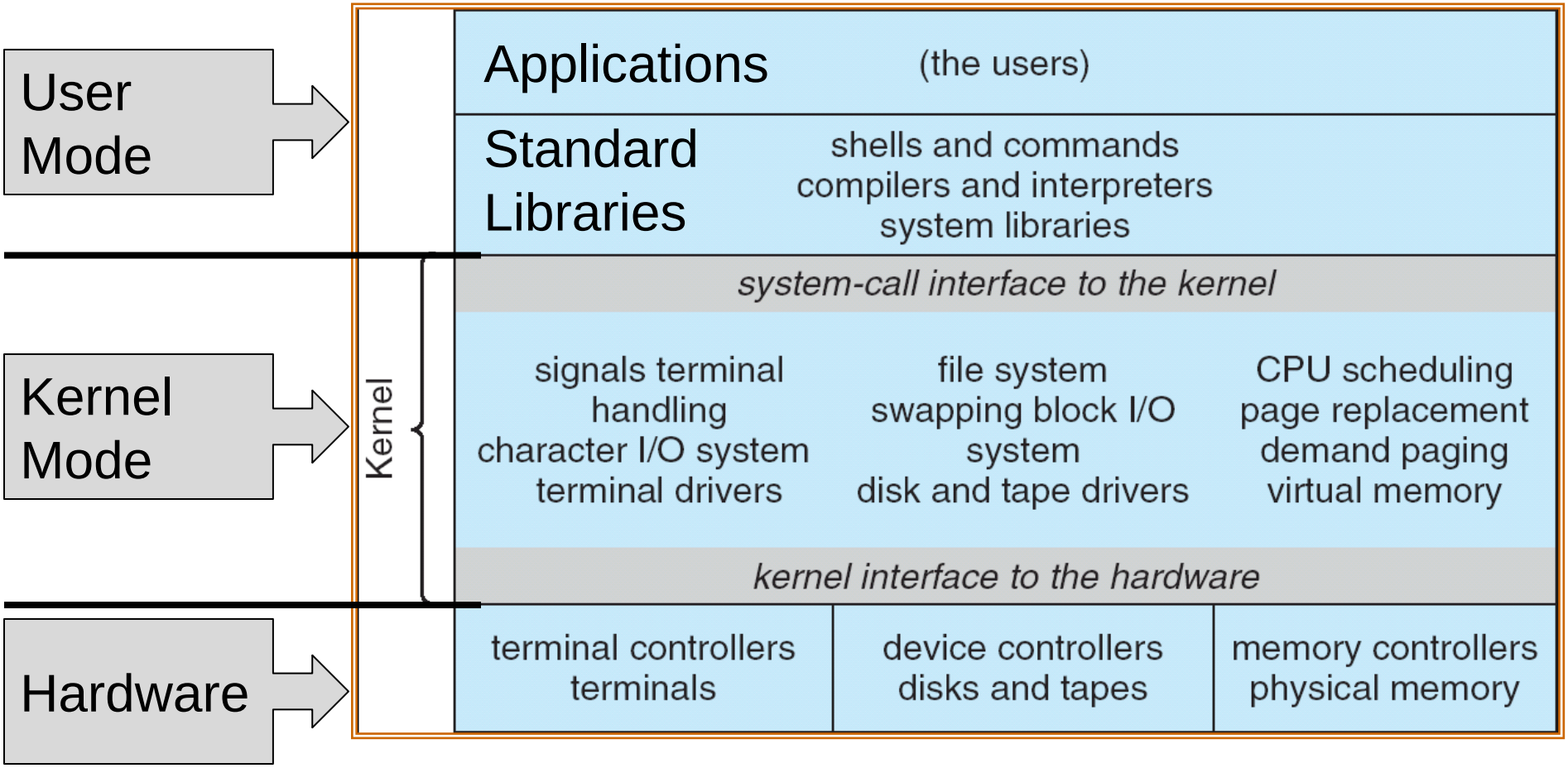
- return-from-interrupt

Dual Mode Operation

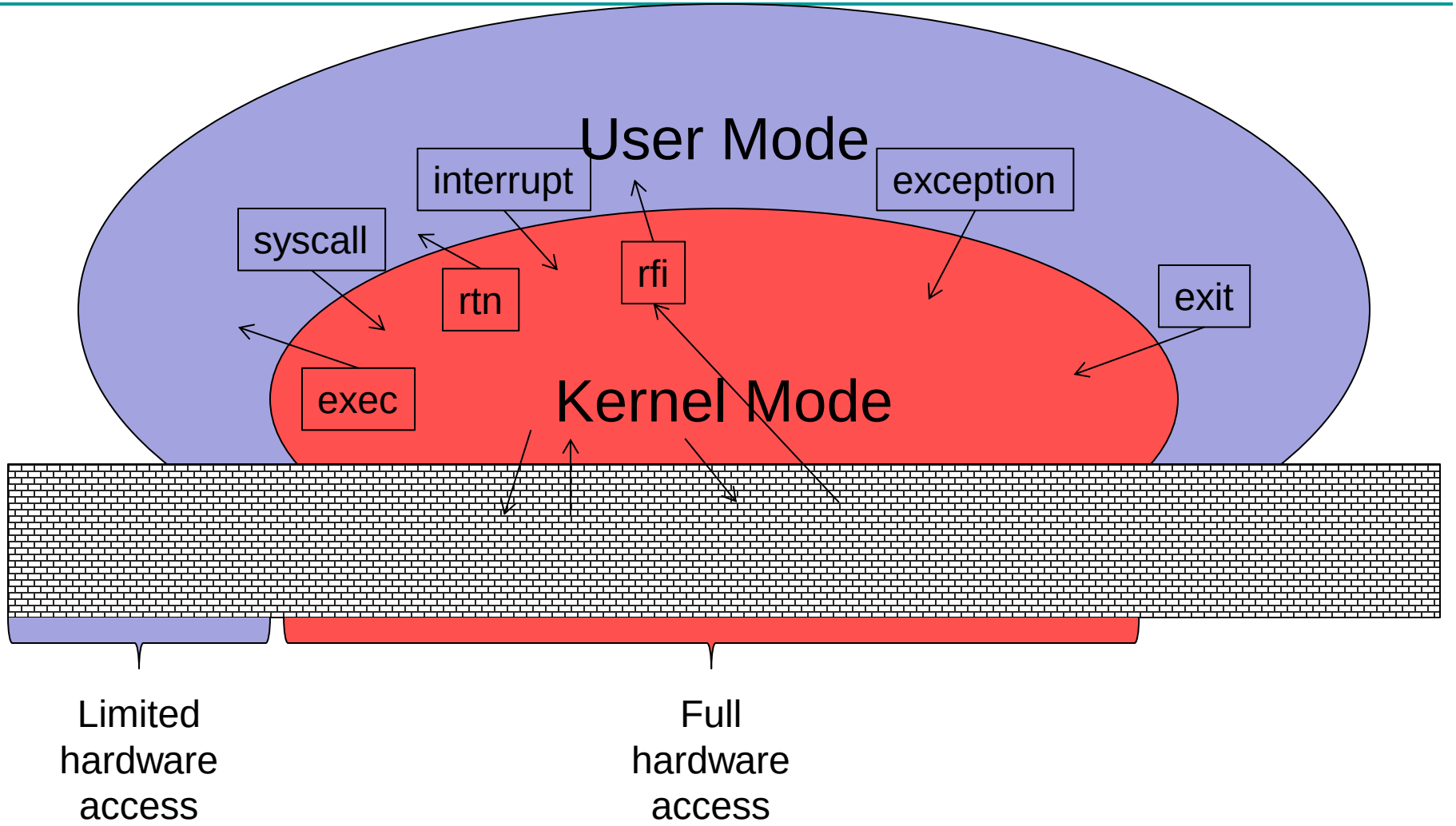
- Some instructions/ops prohibited in user mode:
  - Example: can't modify page tables in user mode
    - Attempt to modify ⇒ Exception generated
- Transitions from user mode to kernel mode:
  - System Calls, Interrupts, Other exceptions



UNIX System Structure



User/Kernel (Privileged) Mode



http://perez.comicbookseries.info/art/superman--art/clark-kent-jan-2017-art-by-perez-at-los-angeles-comic-book-con-2017-from-jahilianaire

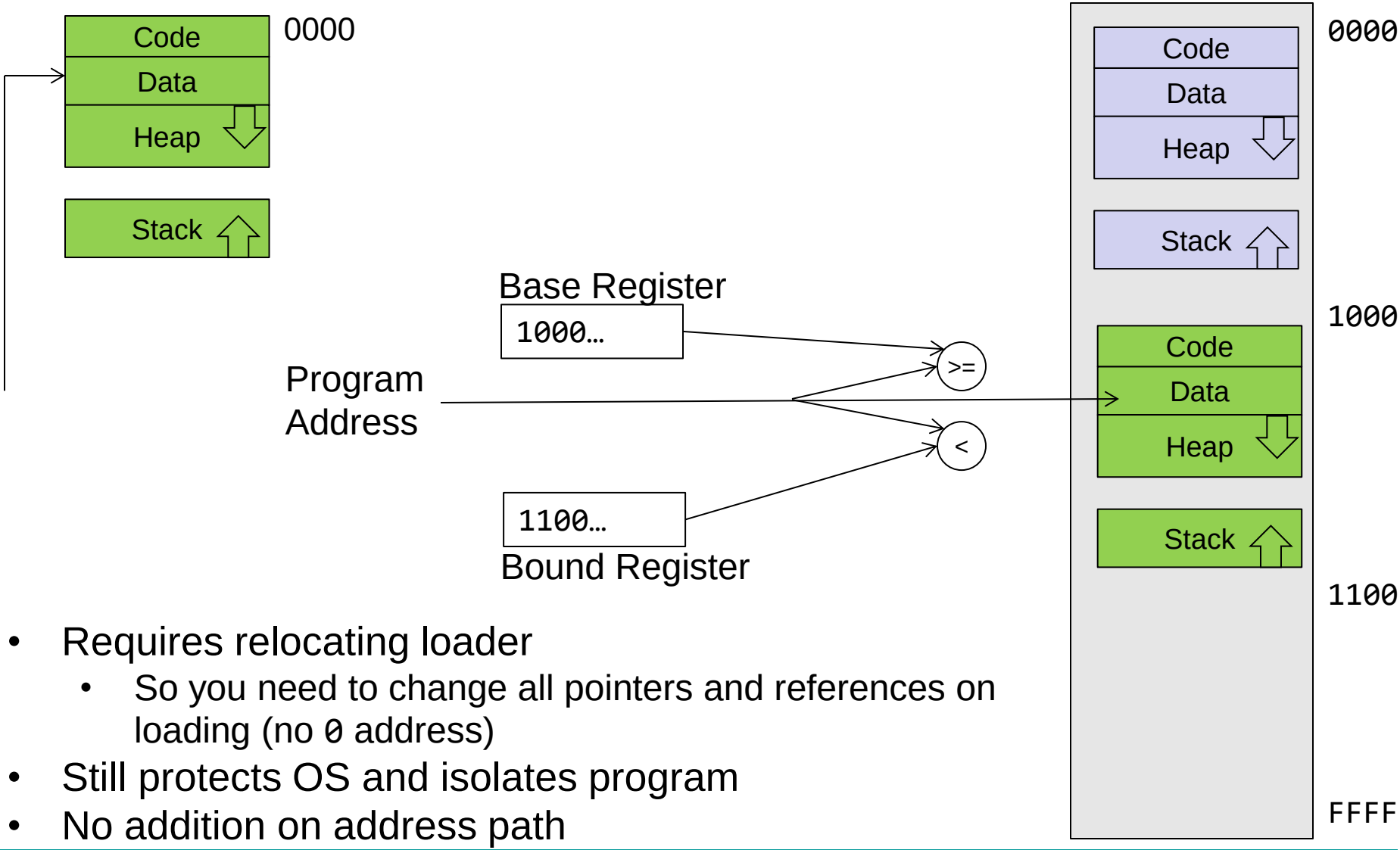
So Far

- Four Concepts
  - Threads (last week)
  - Address space (continued)
  - Processes
  - Dual Mode Operation

- Protection Mechanisms
  - Address ranges: Base and Bound
- Interrupt and Process Context

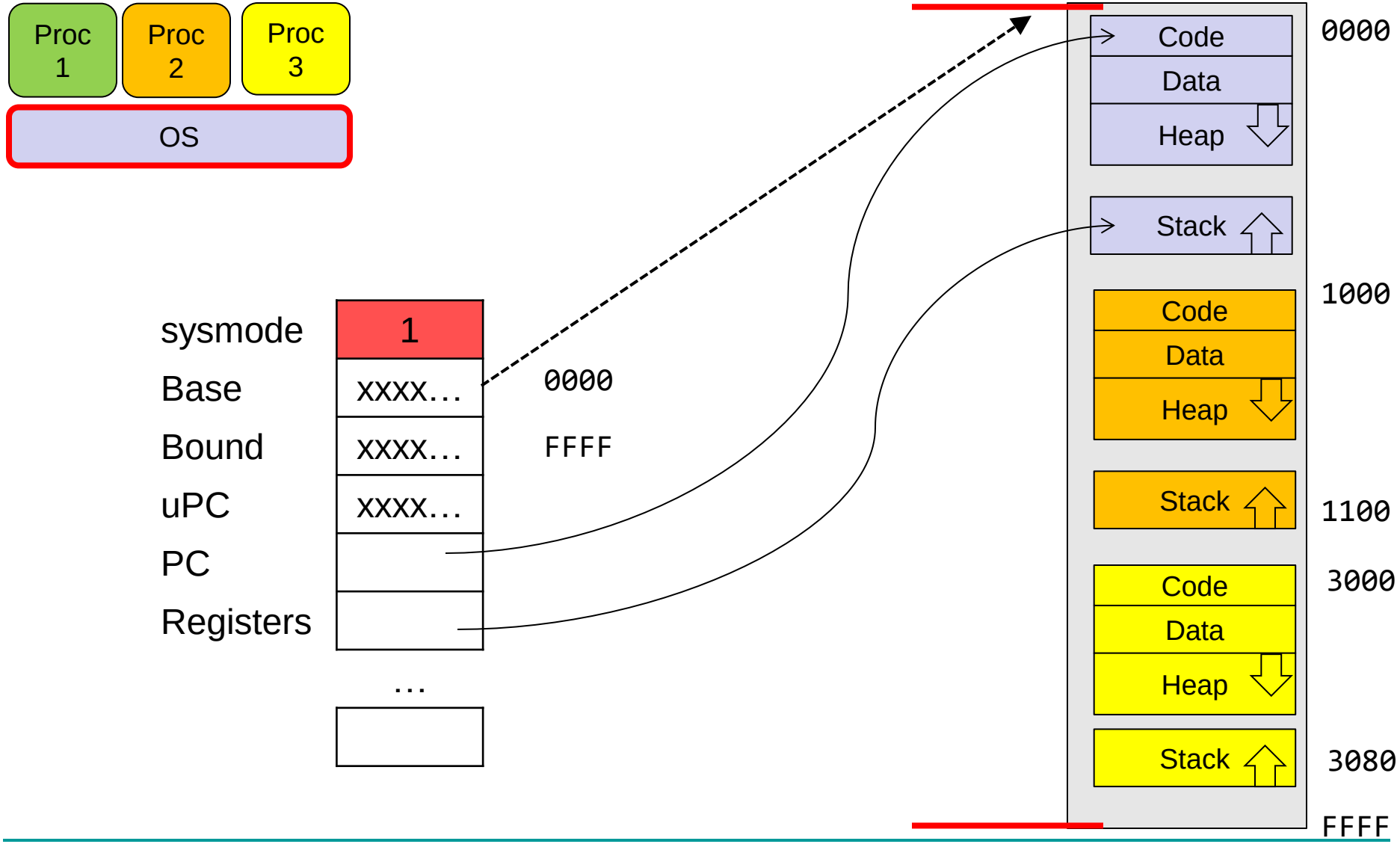
https://i.pinimg.com/originals/f7/00/8b/f7008bd046fd3e7cd6b7f9968de394a6.jpg

Simple Protection: Base and Bound (B&B)

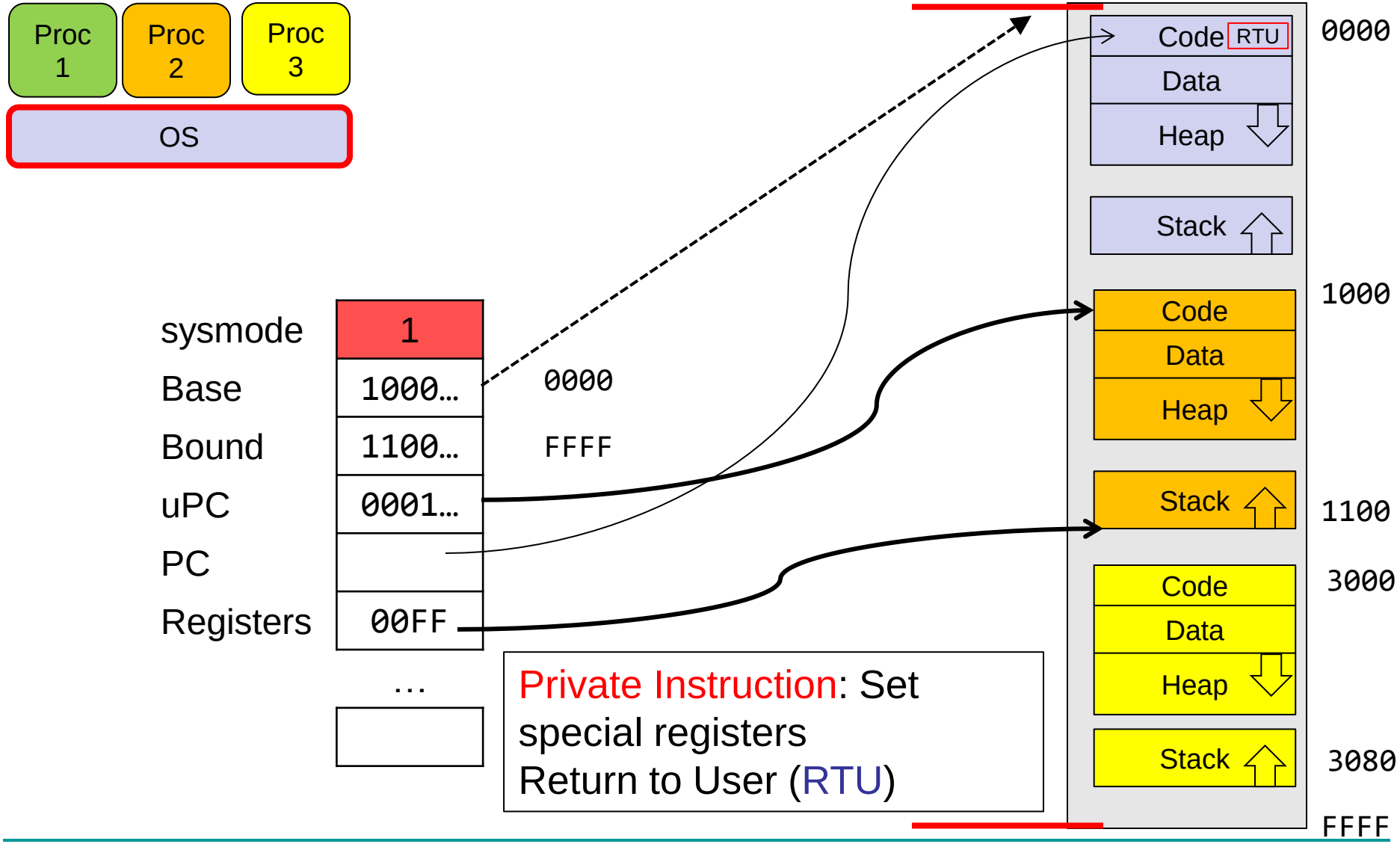


- Requires relocating loader
  - So you need to change all pointers and references on loading (no `0` address)
- Still protects OS and isolates program
- No addition on address path

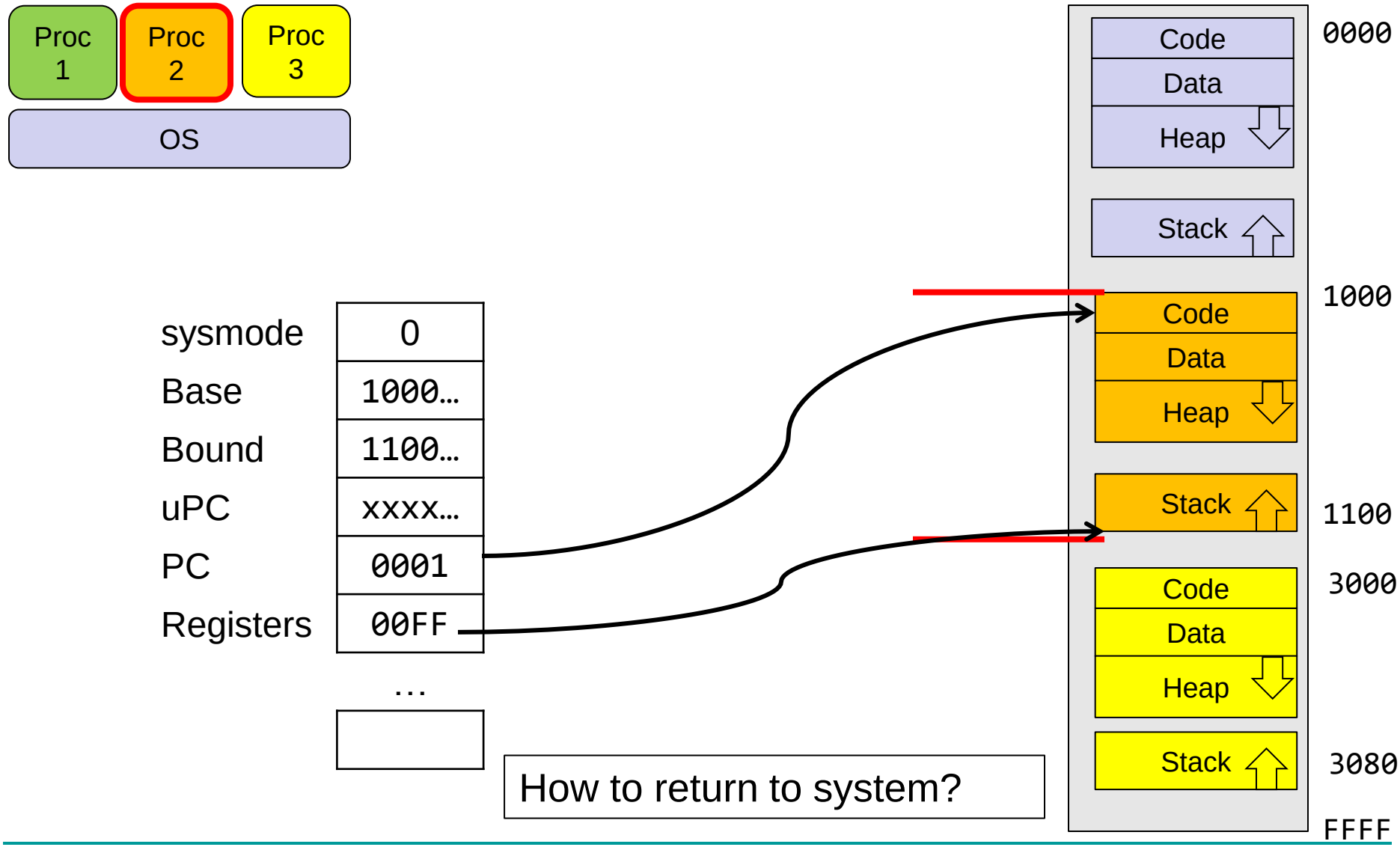
Tying it together: Simple B&B: OS Loads Process



Simple B&B: OS gets ready to switch



Simple B&B: “Return” to User



How to return to system?

3 Types of Mode Transfer

- All 3 are a Unprogrammed Control Transfers
  - Where do they go?

Syscall

- Process requests a system service (e.g., exit())
- Like a function call, but "outside" the process
- Does not have the address of the system function to call
- Like a Remote Procedure Call (RPC) (next year)
- Marshall the syscall id and arguments in registers and exec syscall

Interrupt

- External asynchronous event triggers context switch
- Examples: Timer, I/O device
- Independent of user process

Trap or Exception

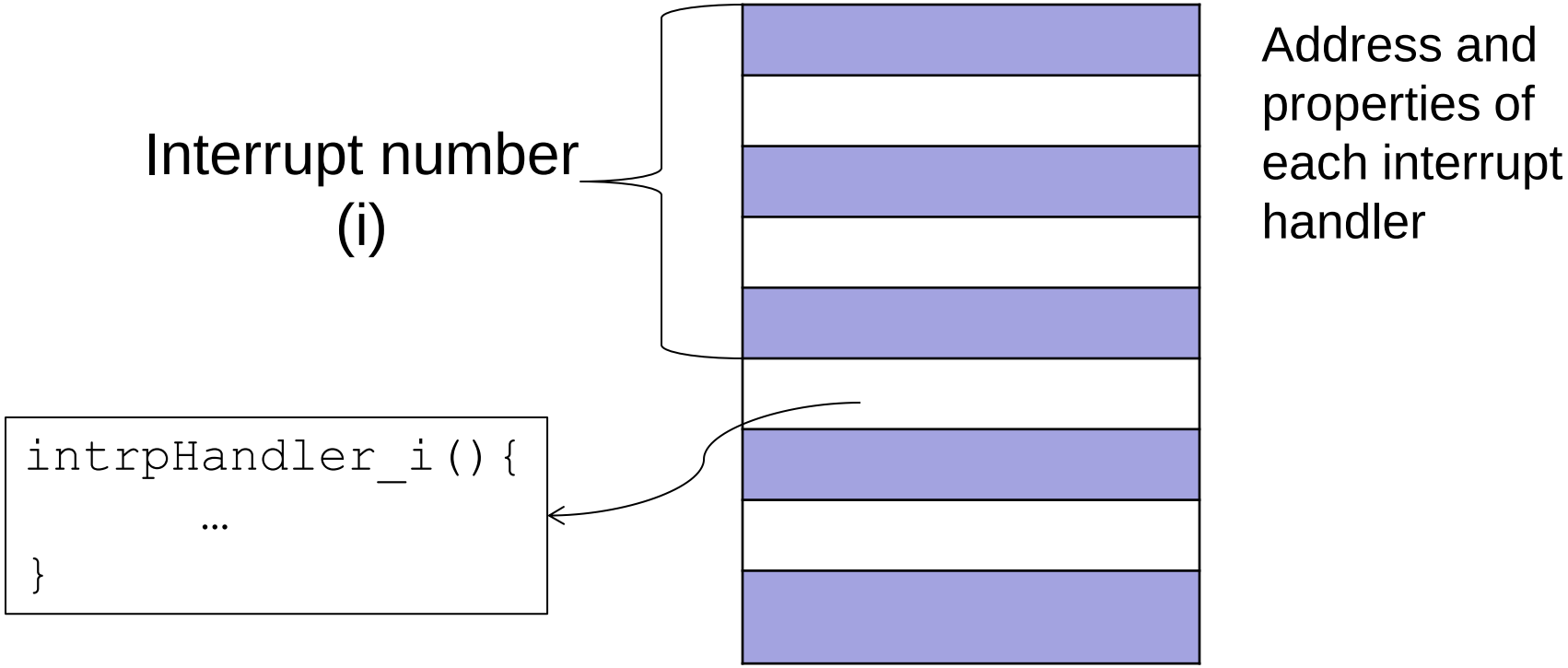
- Internal synchronous event in process triggers context switch
- e.g., Protection violation (segmentation fault), Divide by zero, etc.

How do we get the system target address of the “unprogrammed control transfer?”

El Paso Times

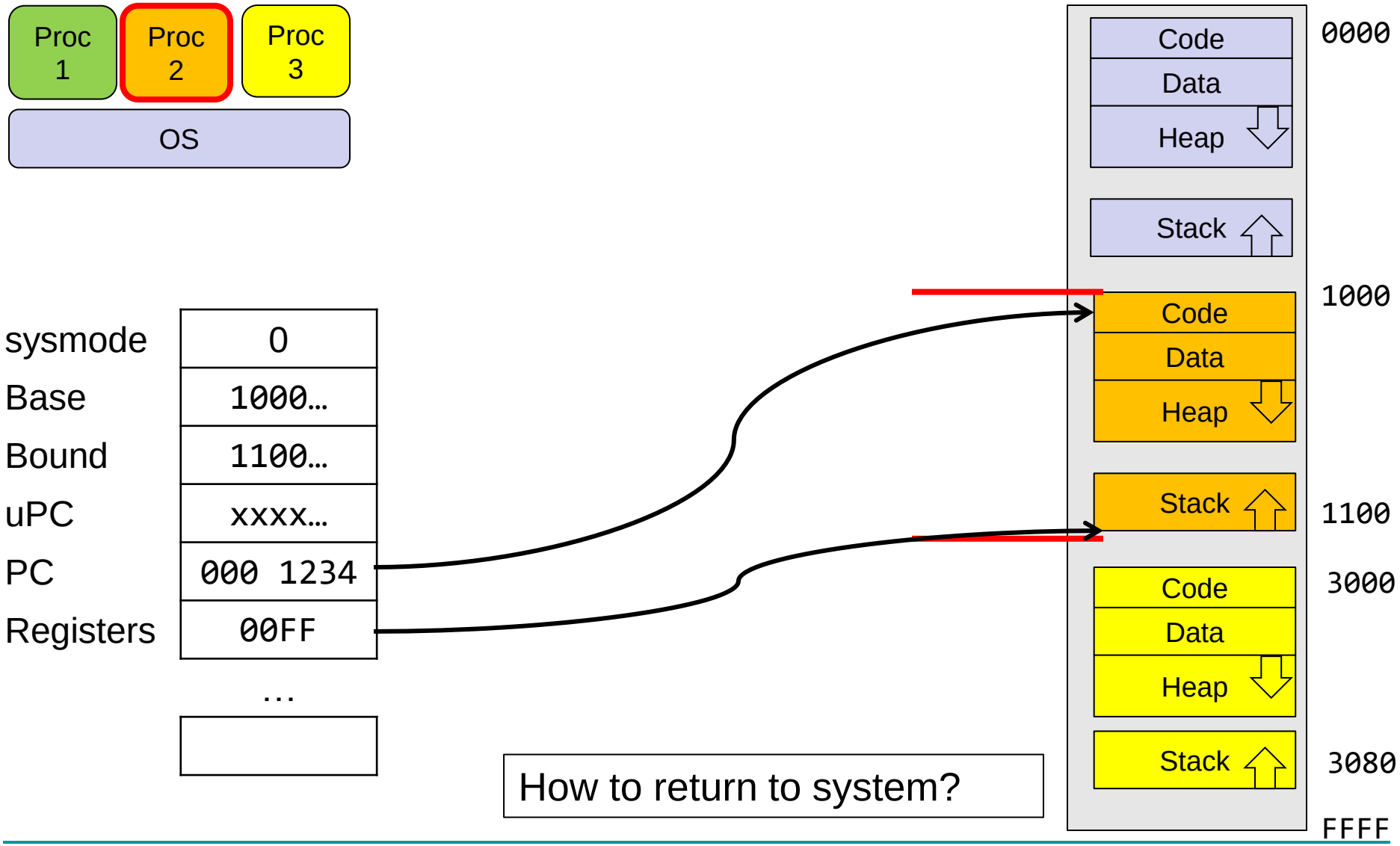
https://www.elpasotimes.com/story/entertainment/events/2016/11/03/flying-cortes-bring-5-generations-coliseum/93177306/

Interrupt Vector



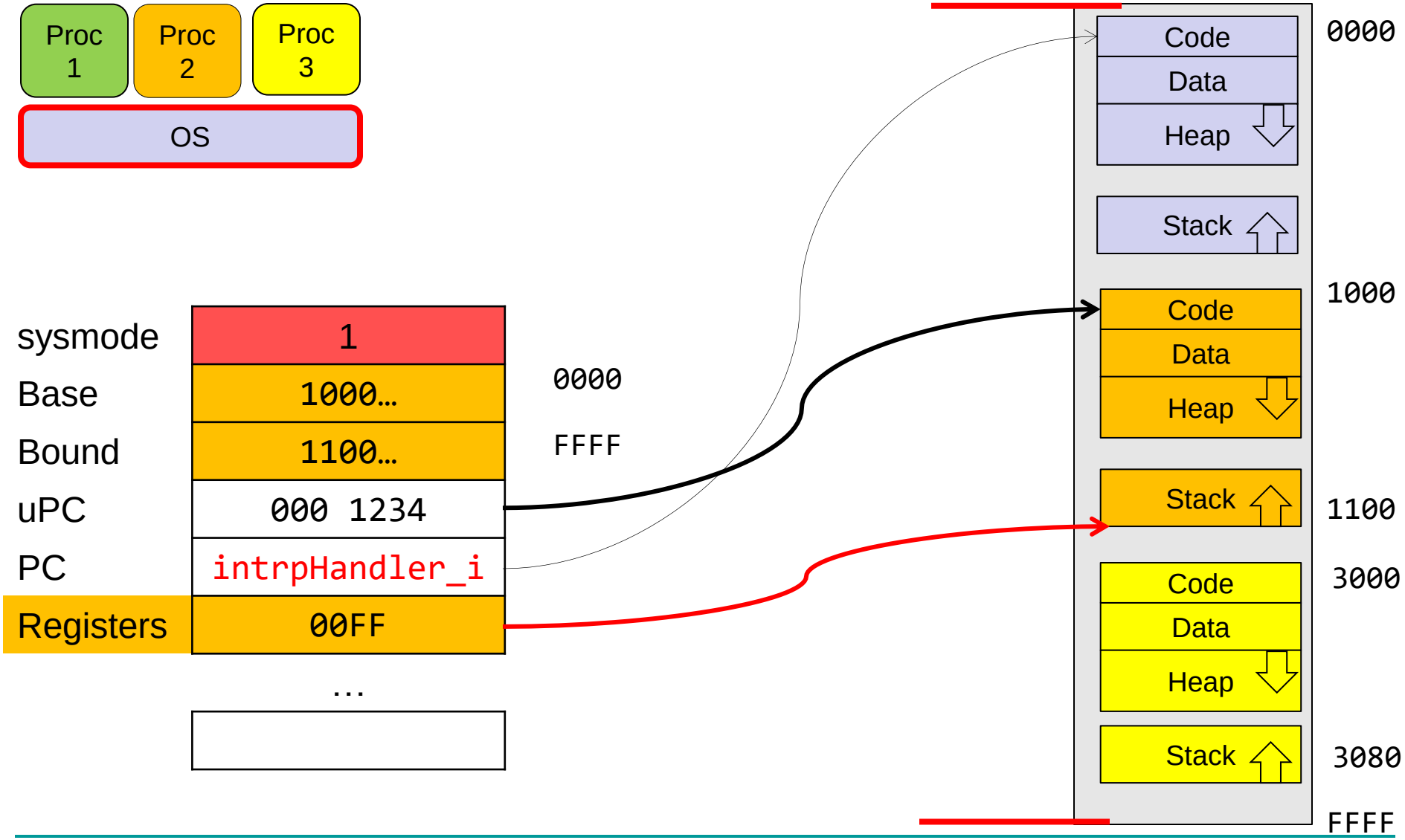
Where else do you see this dispatch pattern?

Simple B&B: User →Kernel



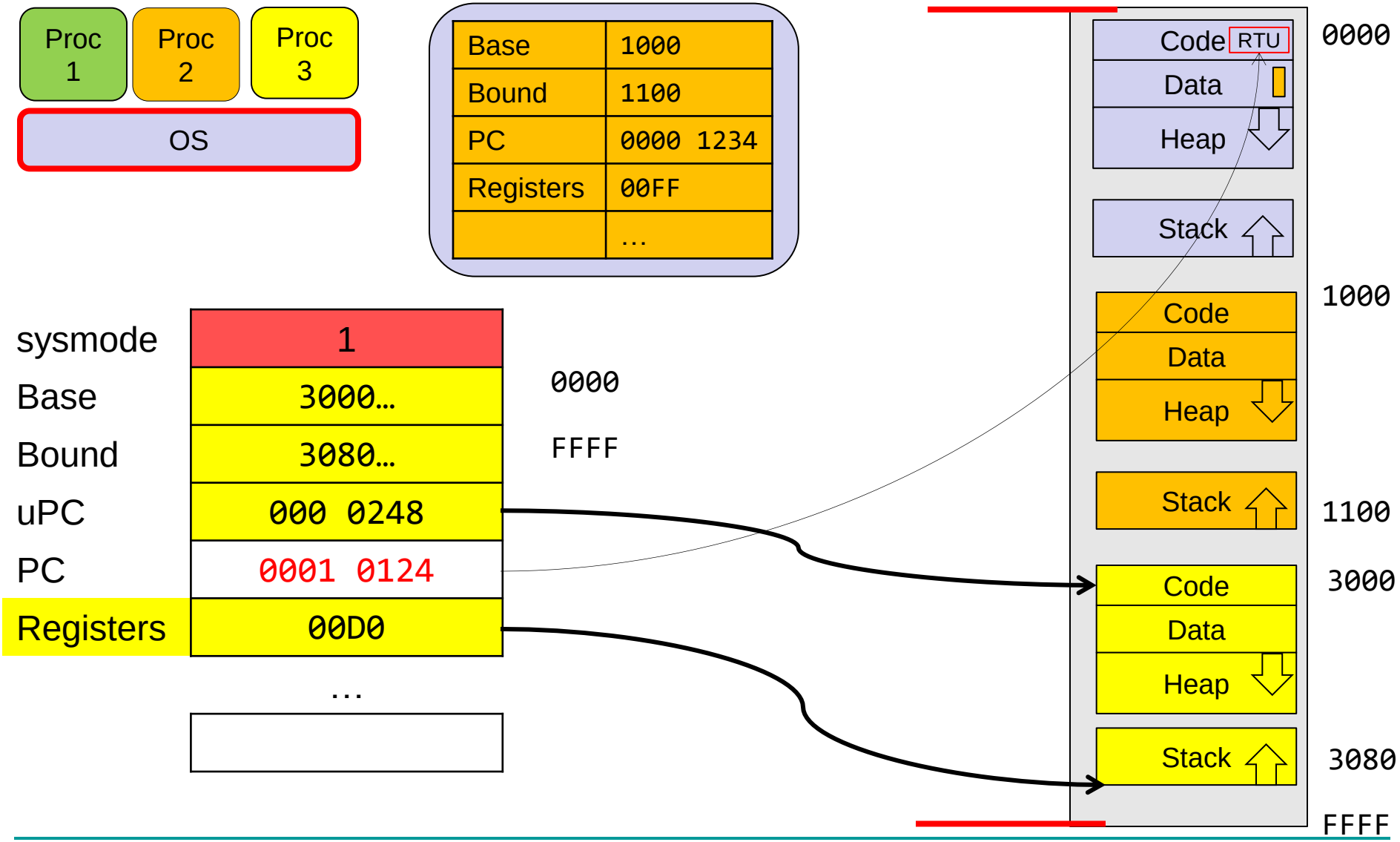
How to return to system?

Simple B&B: Interrupt

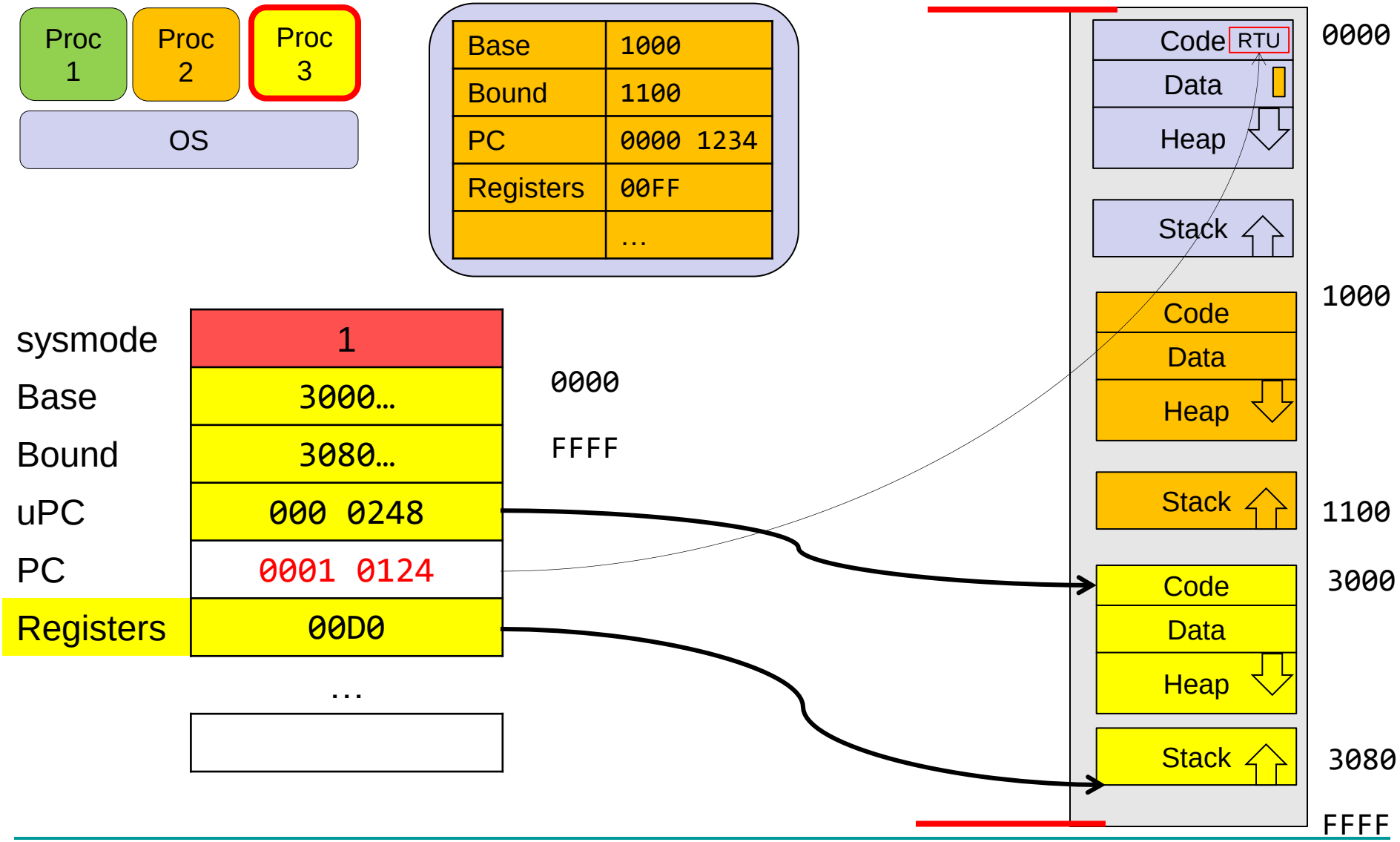


Hostal icons created by Freepik - Flaticon

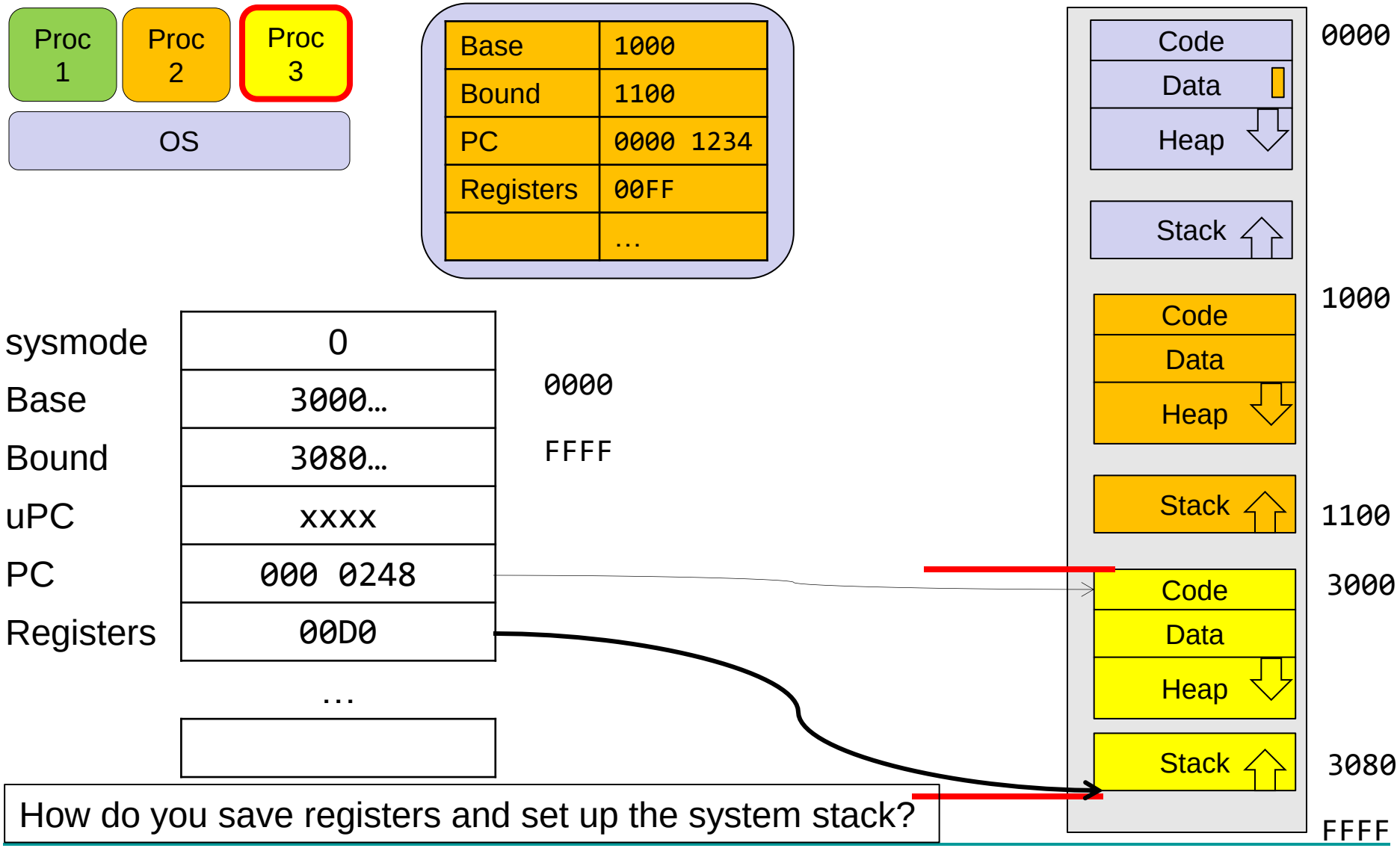
Simple B&B: Switch User Process



Simple B&B: “Resume”



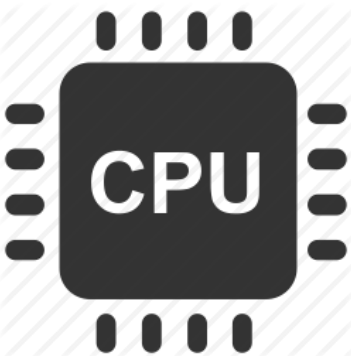
Simple B&B: Proc 3 is running



How do you save registers and set up the system stack?

Another Idea: Address Space Translator

- Program operates in an address space that is distinct from the physical memory space of the machine



Virtual Addresses

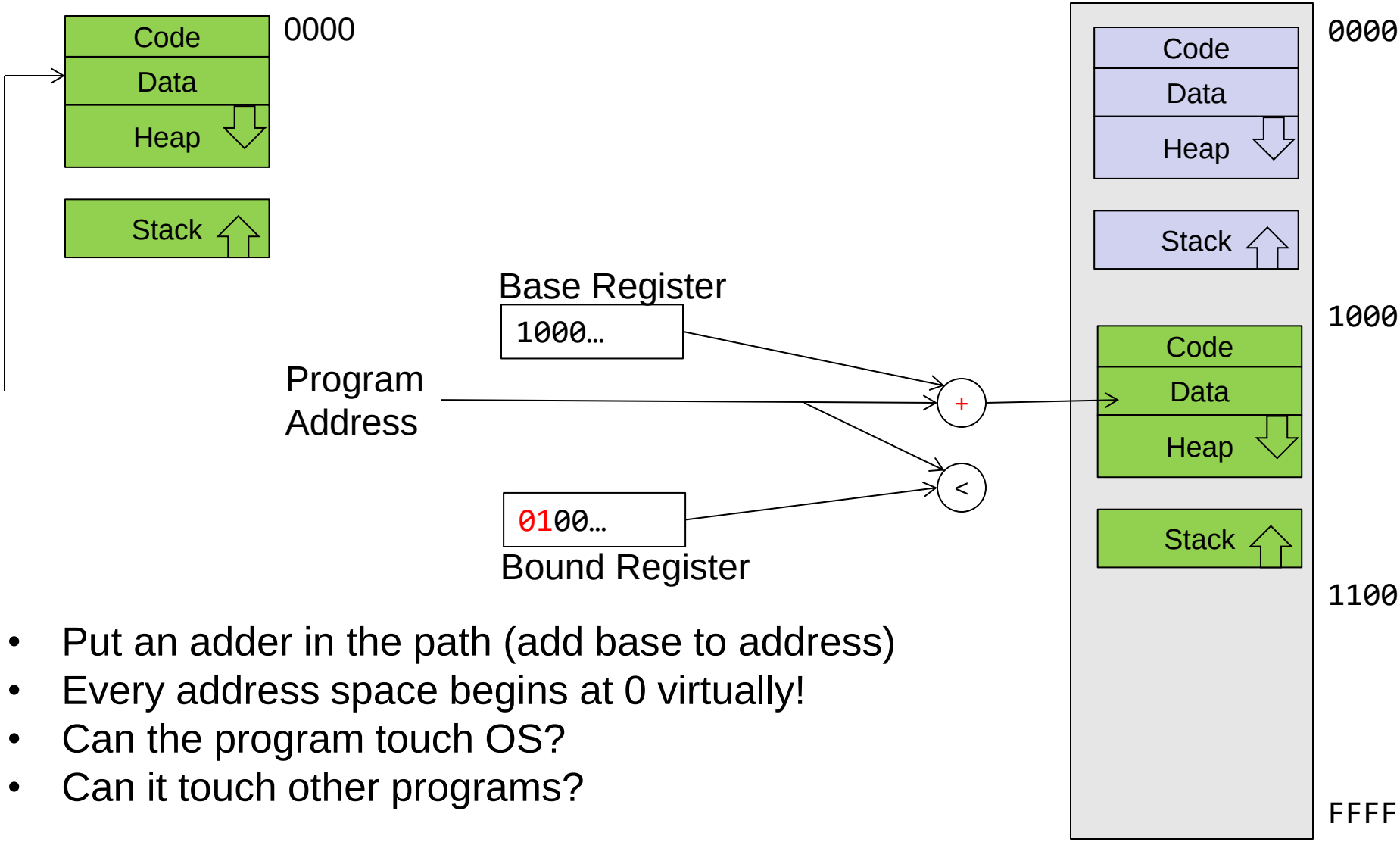
Give me 1000

MMU

Physical Addresses

Give me 3000

Add an Adder to Base and Bound (B&B)

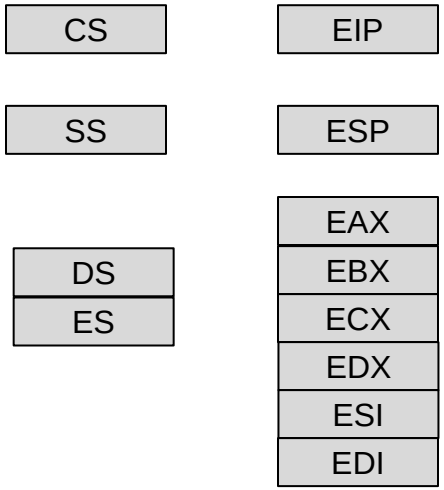


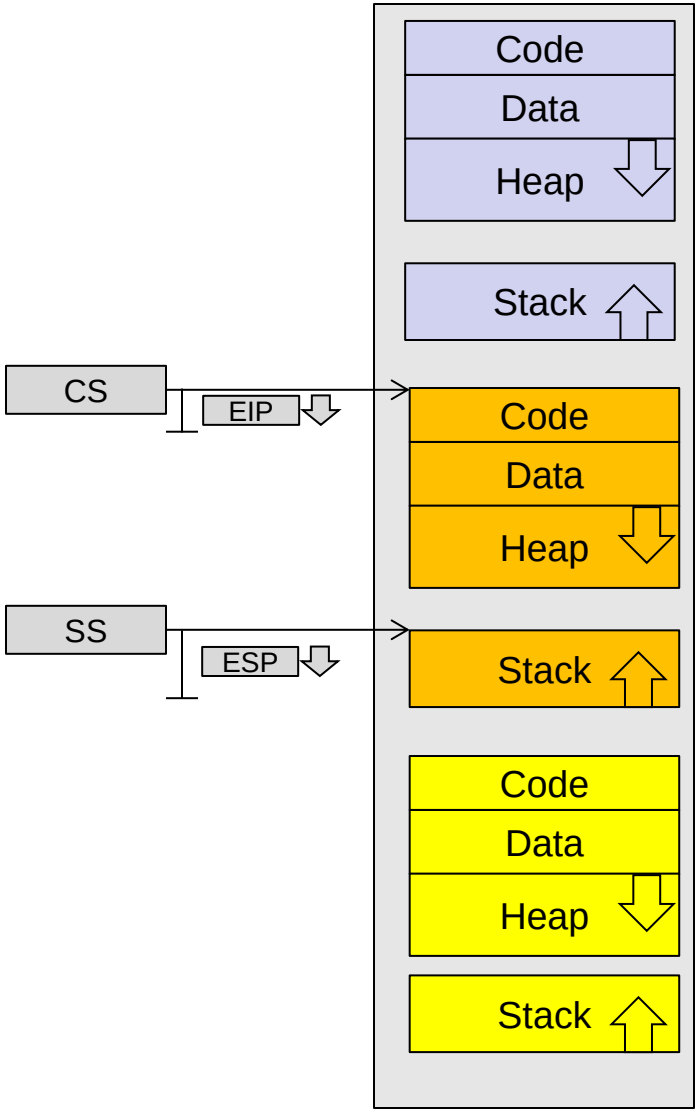
- Put an adder in the path (add base to address)
- Every address space begins at 0 virtually!
- Can the program touch OS?
- Can it touch other programs?

What’s wrong with this simplistic address translation mechanism?

x86 – Segments and Stacks

Processor Registers





Start address, length, and access rights are associated with each segment

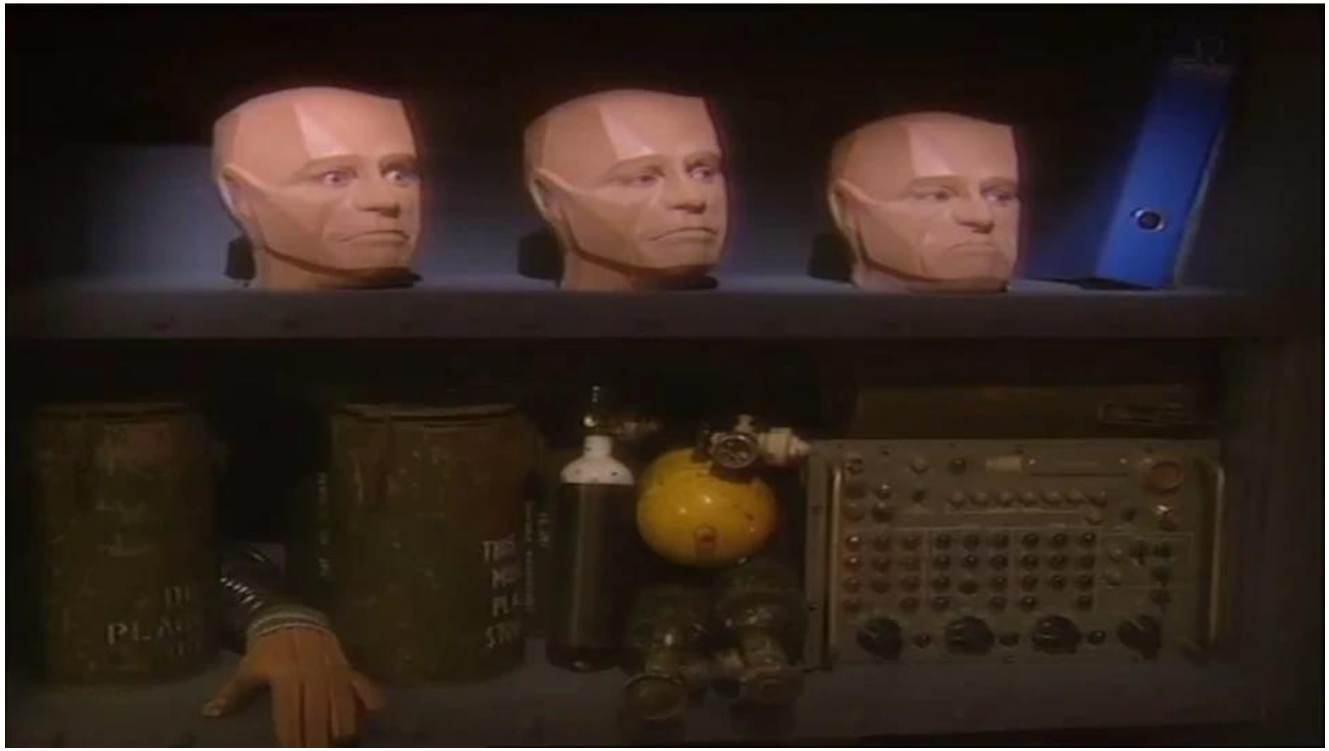
So Far

- Four Concepts
  - Threads (last week)
  - Address space (continued)
  - Processes
  - Dual Mode Operation

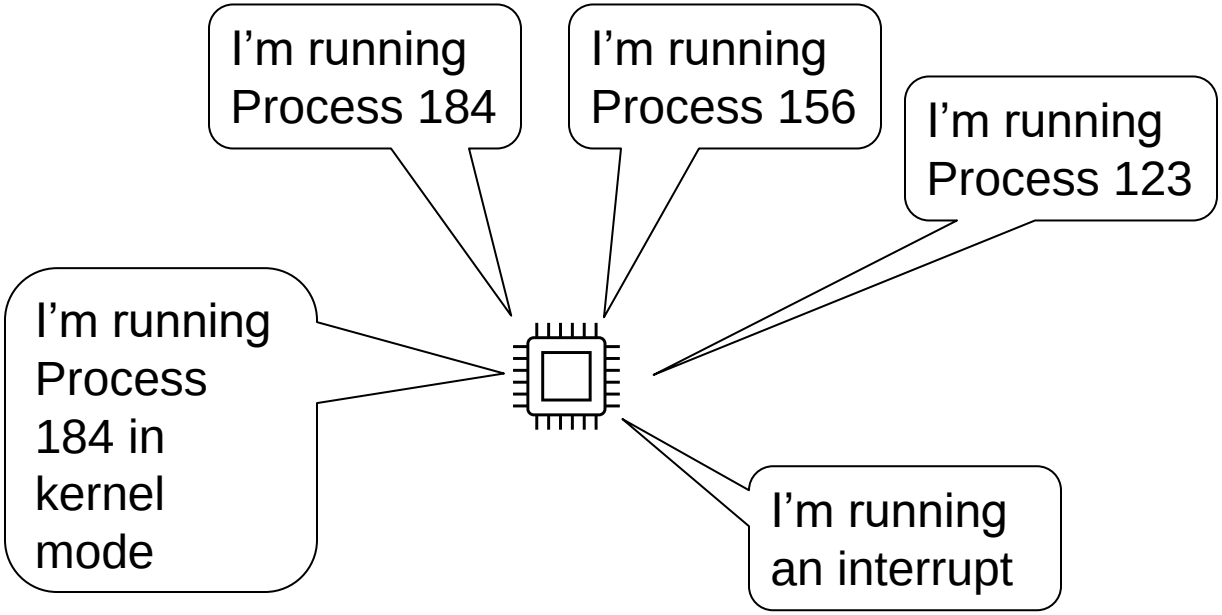
- Protection Mechanisms
  - Address ranges: Base and Bound
- Interrupt and Process Context

Which head to use now?

https://vignette.wikia.nocookie.net/reddwarf/images/c/c9/Spareheads.jpg/revision/latest?cb=20140602034628



Discern: Context



Process context versus Interrupt context

Conclusion

- Four Concepts
  - Threads (last week)
  - Address space (continued)
  - Processes
  - Dual Mode Operation

- Protection Mechanisms
  - Address ranges: Base and Bound
- Interrupt and Process Context