CPSC313: Computer Hardware and Operating Systems

Unit 5: Virtual Memory

(5.1) Virtual Memory: Achieving Process Isolation

Admin

- Quiz 4 viewings and retake this week
- You should already have a final exam time!
- Labs 9 and 10 are “in progress”. Check PrairieLearn for deadlines!

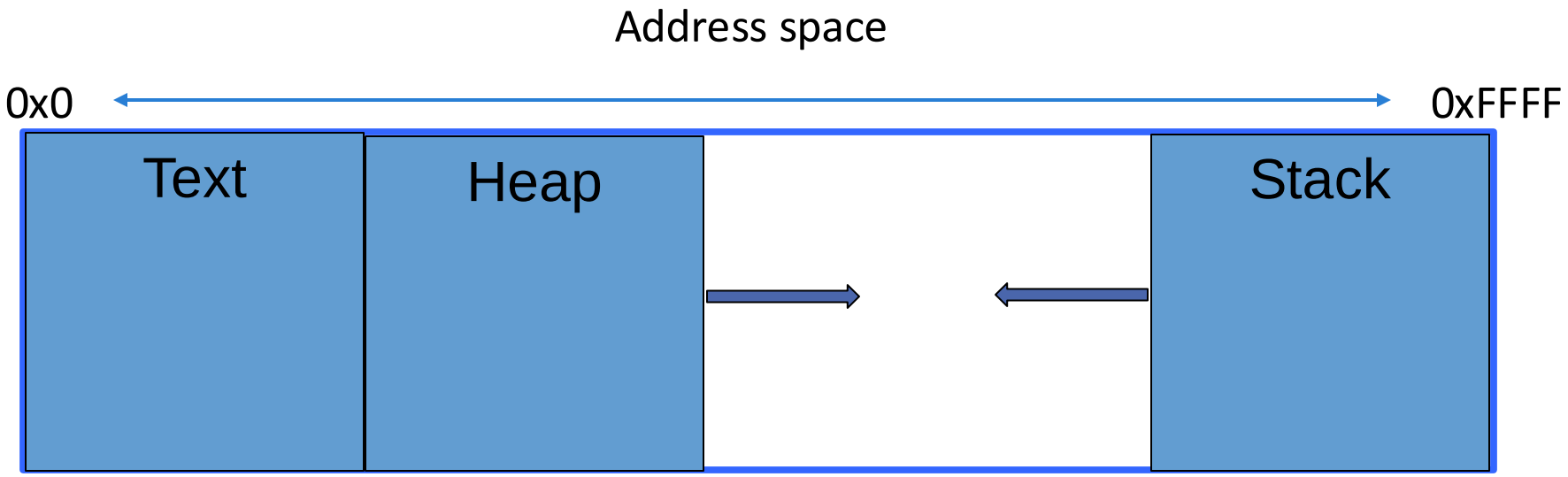
Today: The start of our last unit, and… MAGIC!

- Learning Outcomes
  - Define:
    - Process
    - Address space
    - Process isolation
    - Virtual memory
  - Explain how virtual memory provides process isolation.
- Reading
  - Section 8.2

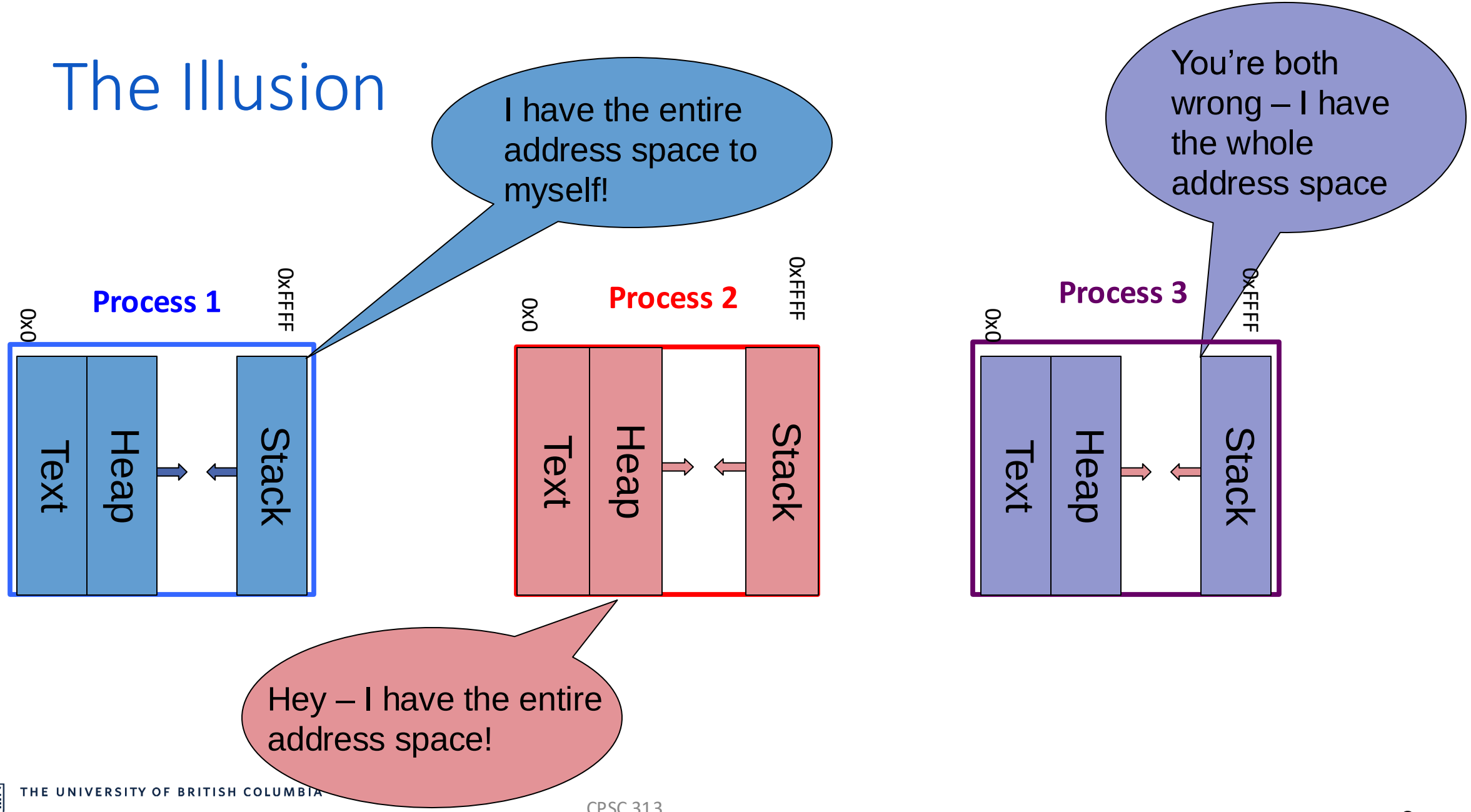
Fundamentals:

- Programs run in different processes.
- Each process has its own address space.
- Address spaces provide process isolation.
- Process isolation means:
  - Anything one process does should not affect what another process does, unless the processes agree (e.g., set up a communication channel).
  - Each process behaves as if it controls the entire machine's resources.

Recall way back from Unit 1



The Illusion



It's all a lie! (illusion)

There are two different kinds of address:

Physical addresses *

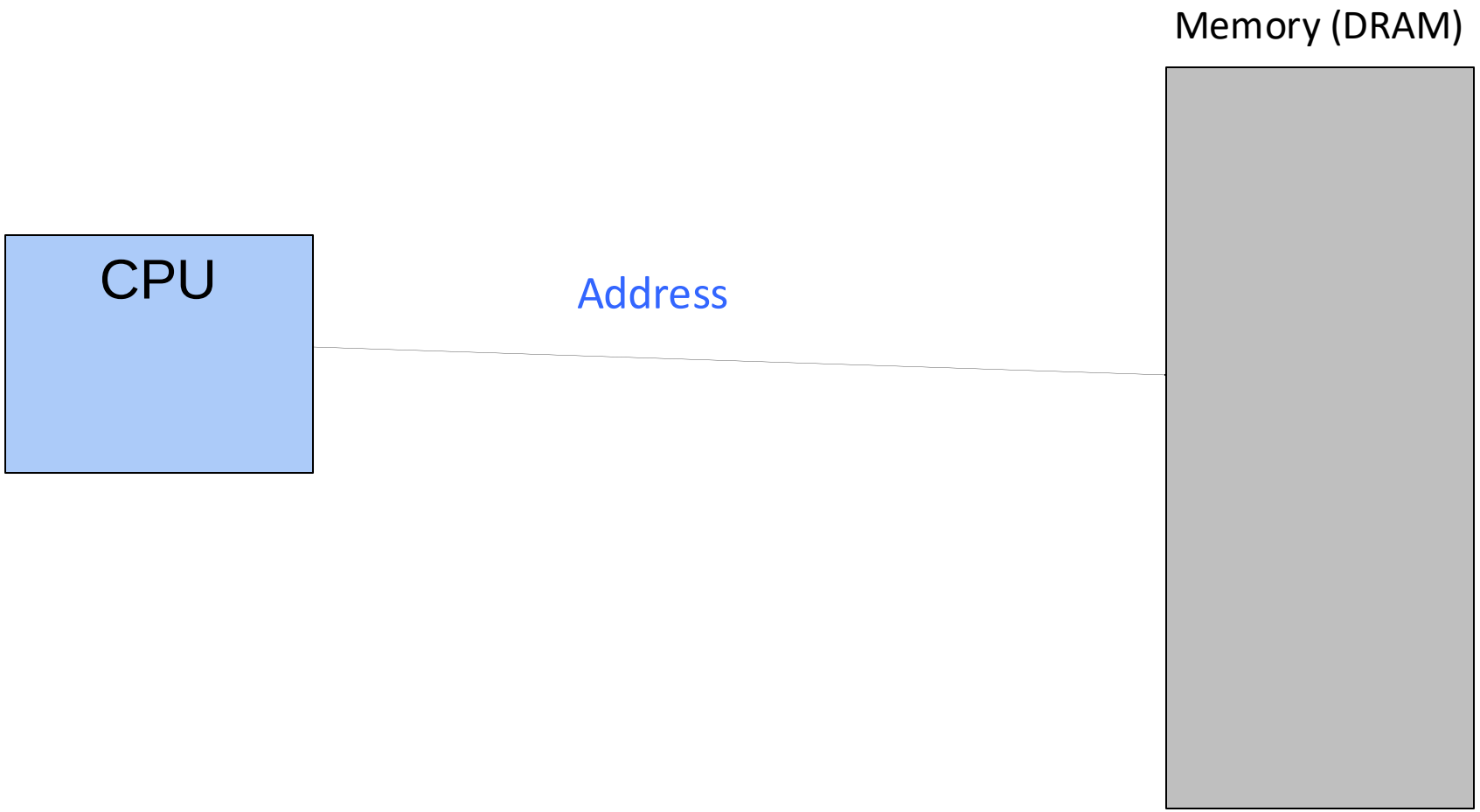
- Refer to specific locations in memory (DRAM).

Virtual addresses

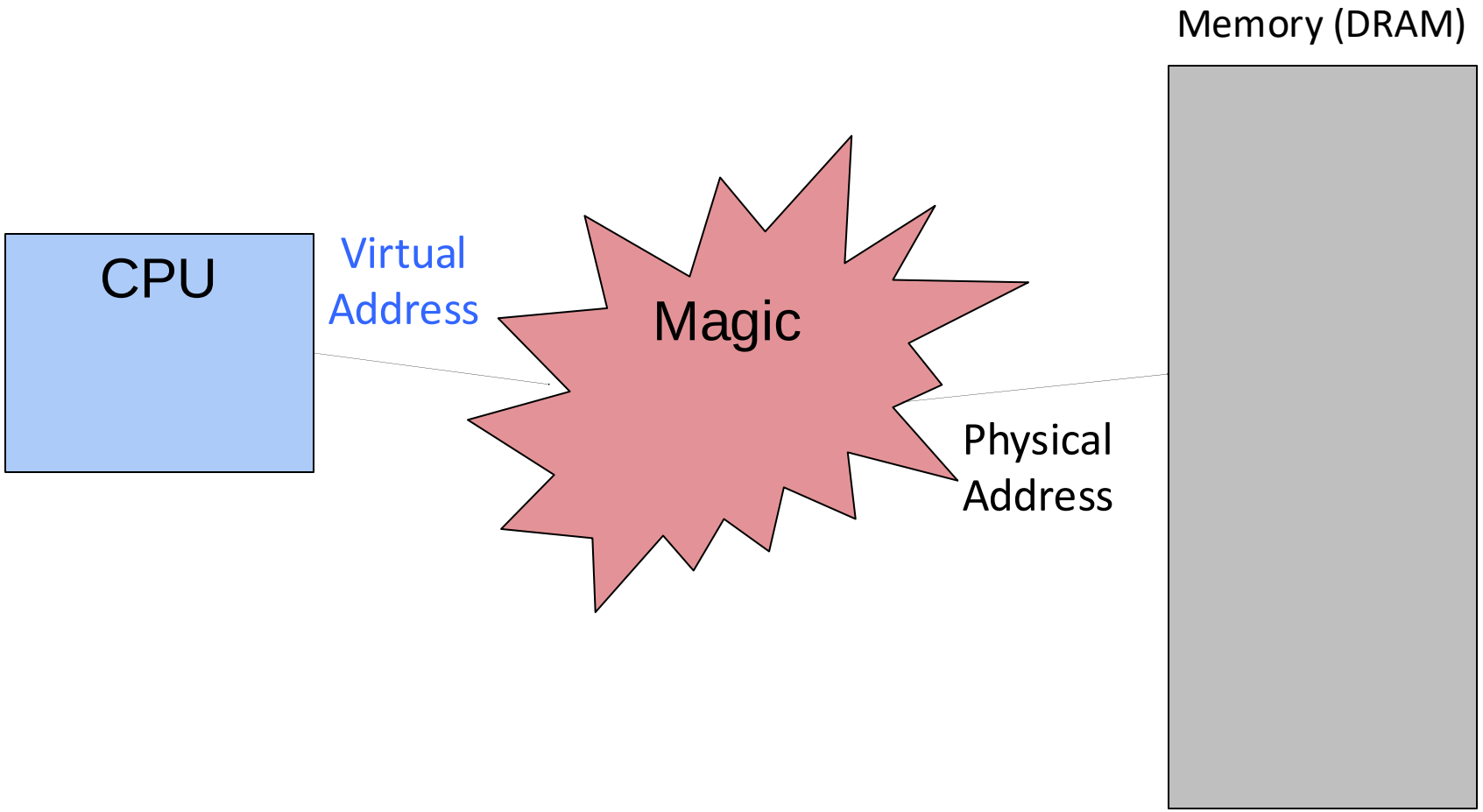
- These are the addresses that programs use.
- The hardware (HW) & software (SW) collaborate to map from virtual addresses to physical address.
- A machine can have less physical memory than the size of the virtual address space!
- You can have a machine with a 64-bit architecture supporting program with virtual addresses measuring in Petabytes, without having a machine with that much memory.

* It turns out that this is all a lie too: out of scope for this course, but if you're interested, check out https://www.usenix.org/system/files/conference/hotos15/hotos15-paper-gerber.pdf

How?

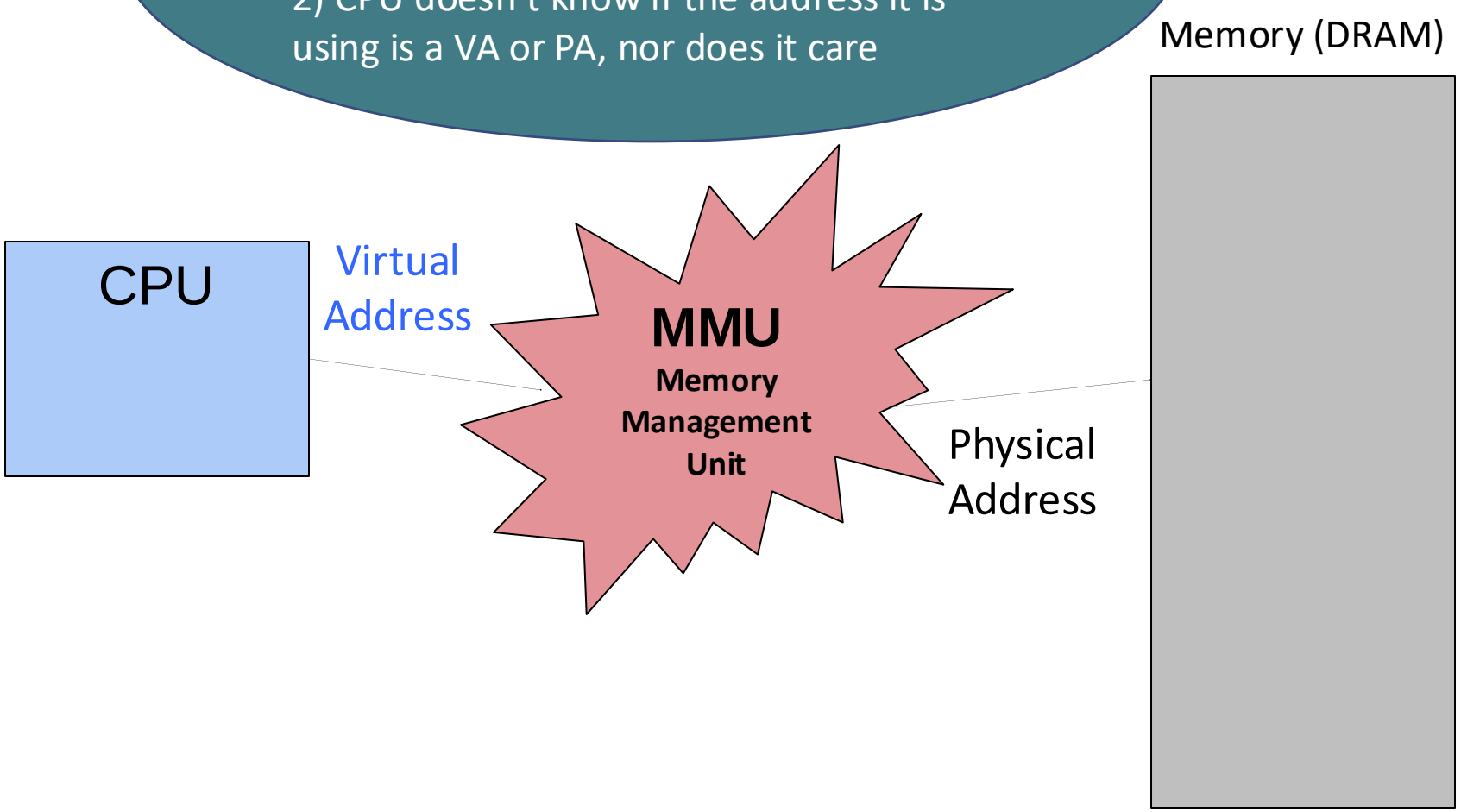


How? – Insert some “magic”



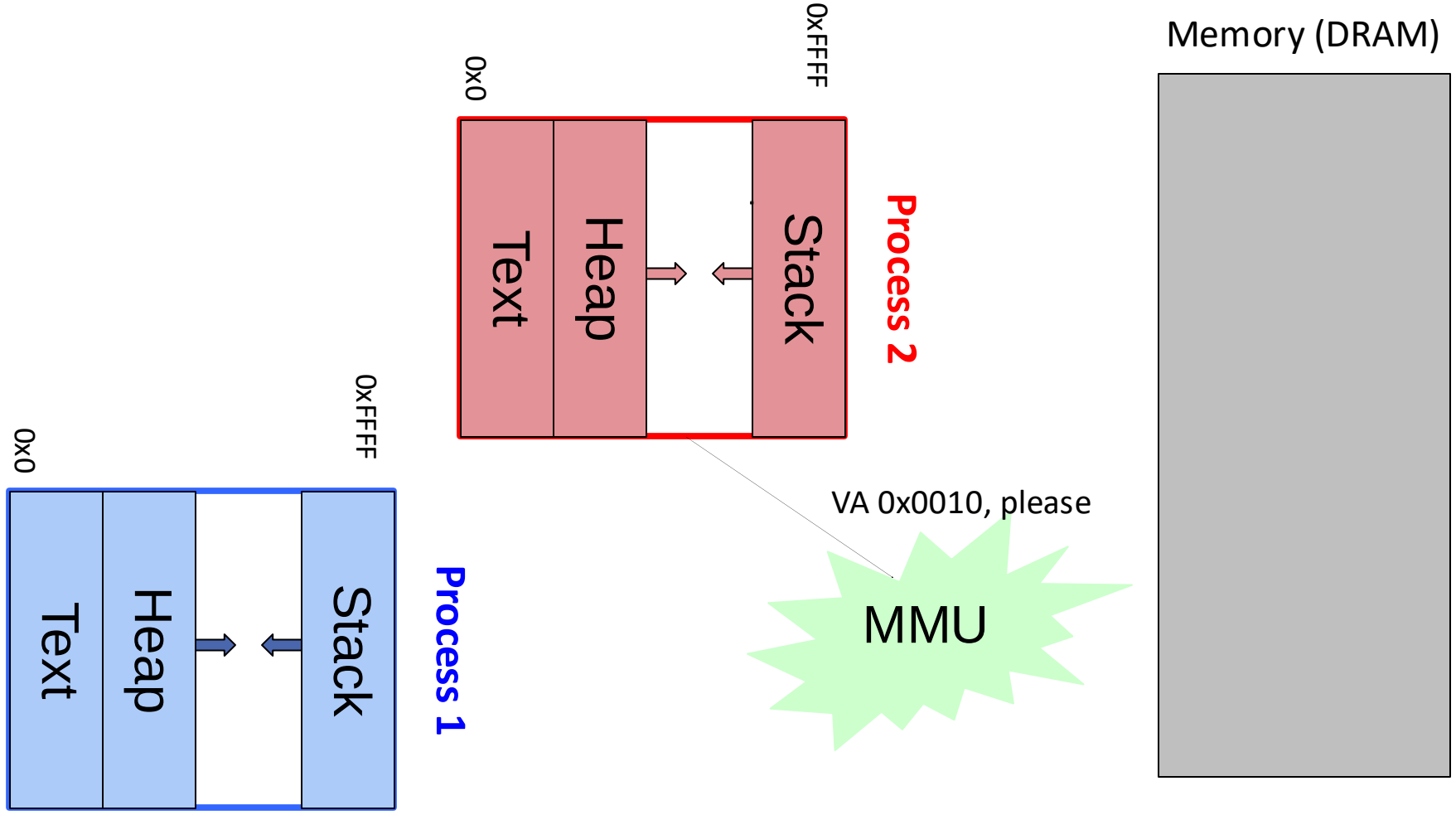
How?

Observe:
1) If MMU takes the VA and simply passes it through then the VA ==PA
2) CPU doesn't know if the address it is using is a VA or PA, nor does it care

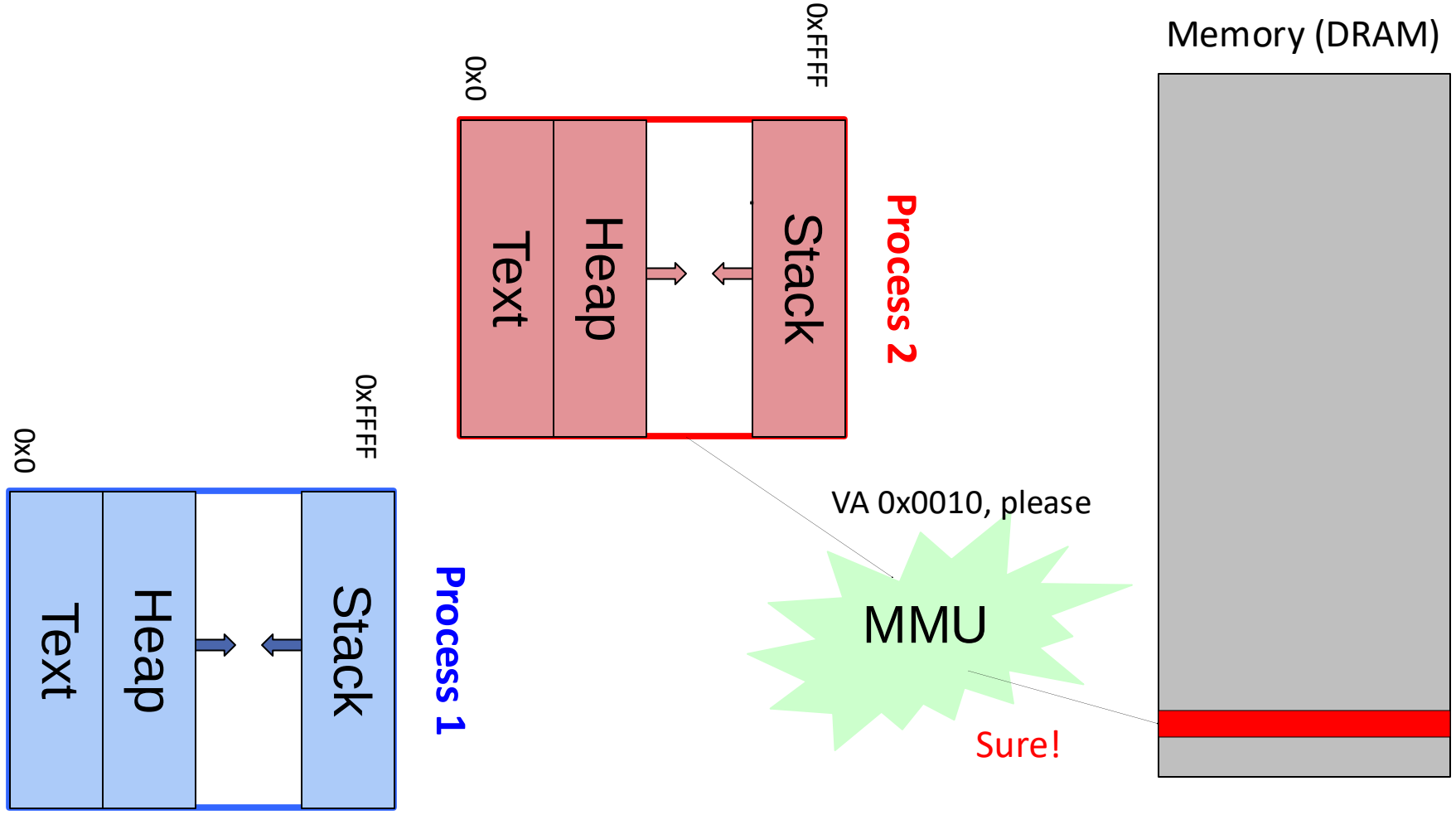


How does this help us?

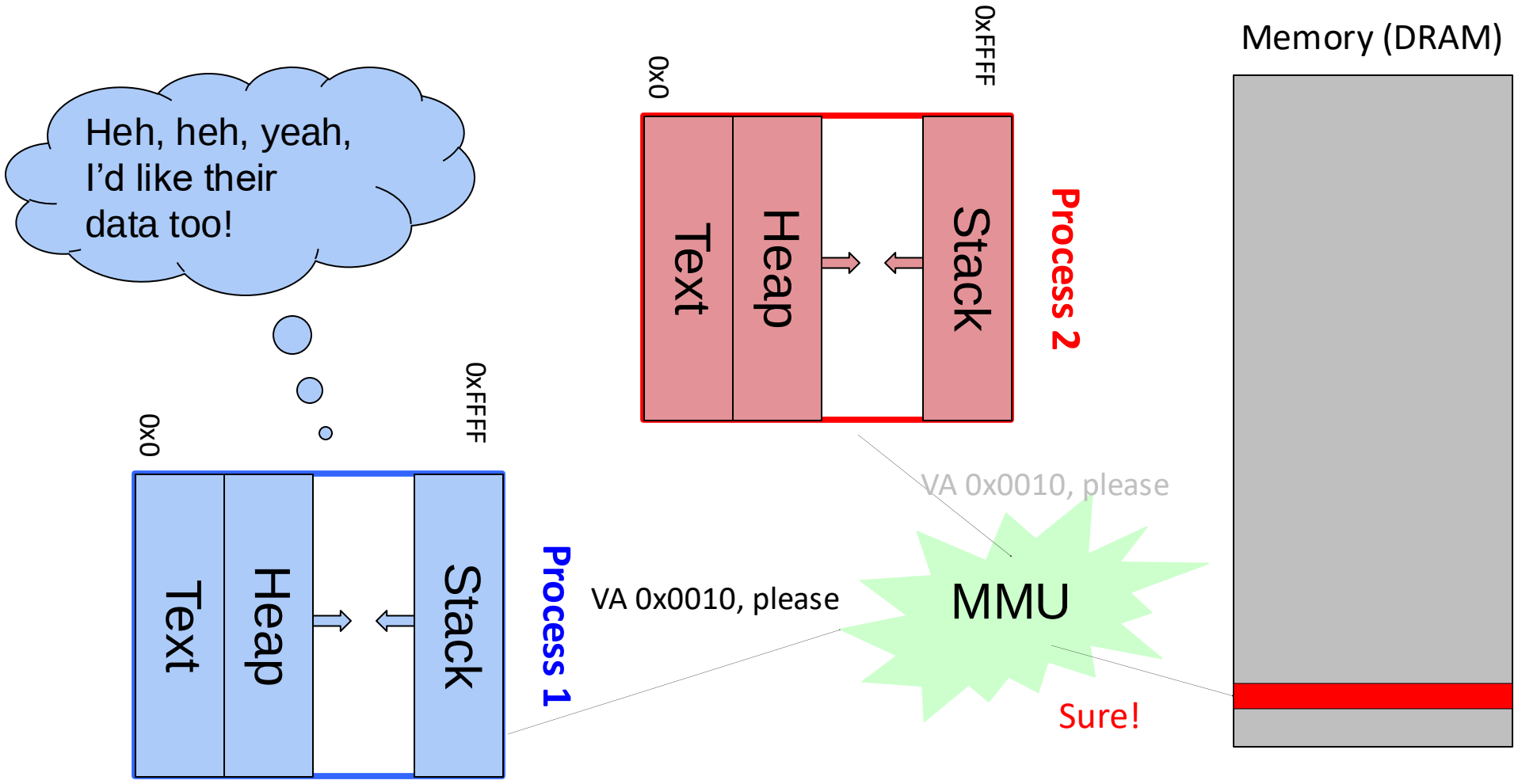
How VM Provides Process Isolation/Protection



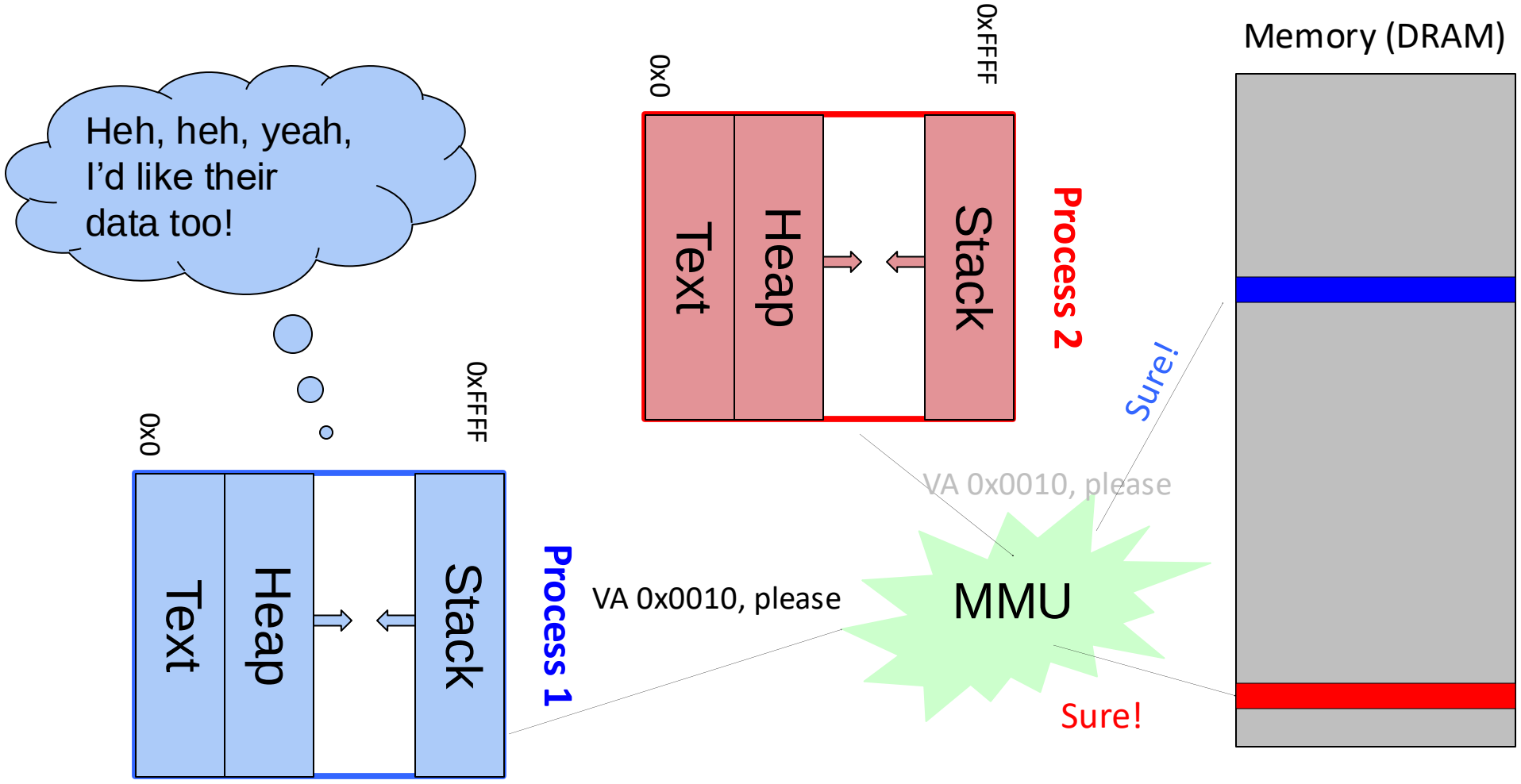
How VM Provides Process Isolation/Protection



Protection


Heh, heh, yeah, I'd like their data too!
0x0
0xFFFF
Text
Heap
Stack
Process 1
0x0
0xFFFF
Text
Heap
Stack
Process 2
VA 0x0010, please
MMU
VA 0x0010, please
Sure!
Memory (DRAM)

Protection


Heh, heh, yeah, I'd like their data too!
0x0
0xFFFF
Text
Heap
Stack
Process 1
0x0
0xFFFF
Text
Heap
Stack
Process 2
VA 0x0010, please
MMU
Sure!
VA 0x0010, please
Sure!
Memory (DRAM)

VM: A Hardware/Software Partnership

- We need hardware support to provide virtual memory. Why?

  Speed! Invoking the operating system (OS) on every address access would be **much** too slow.

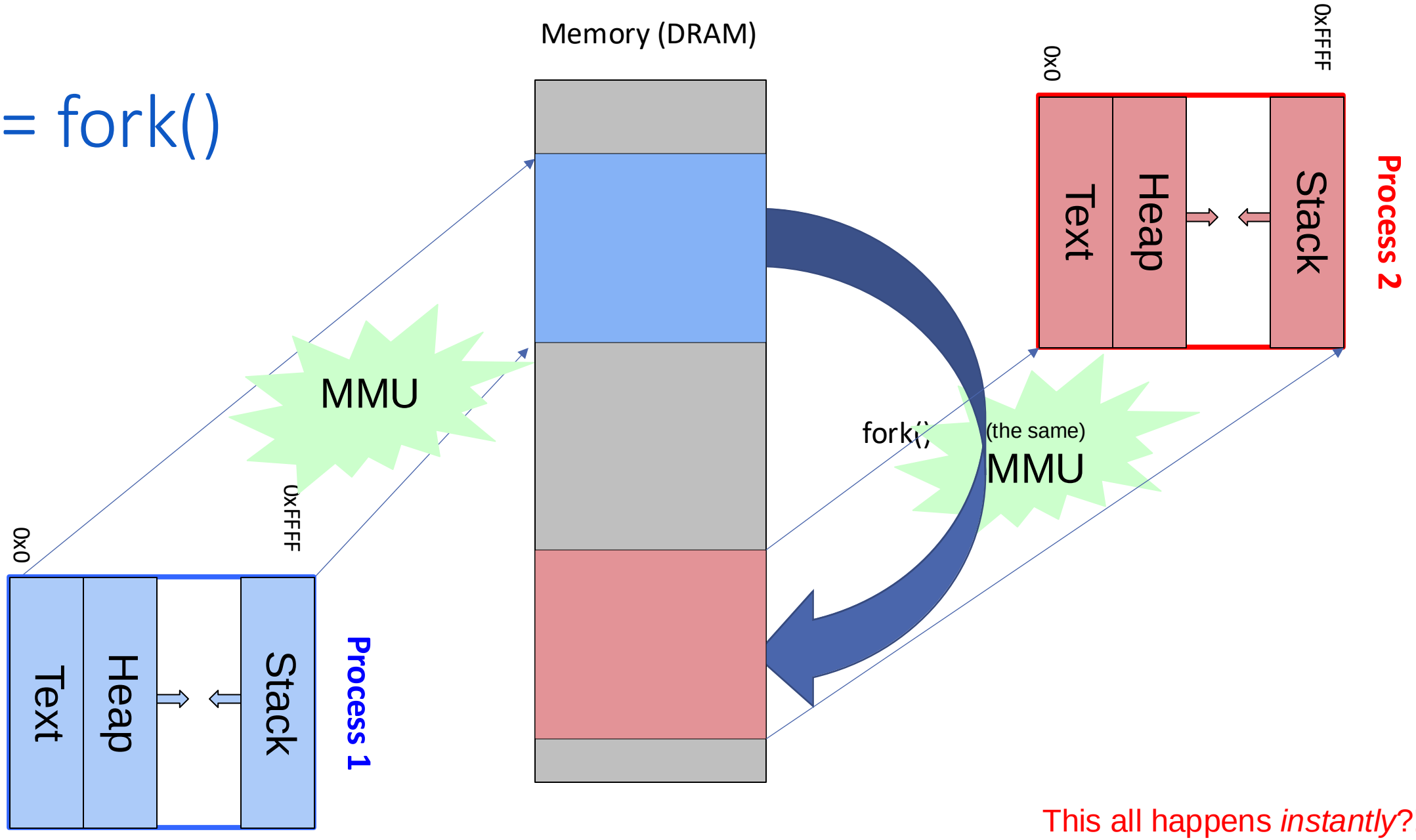
- The Hardware
  - Provides a fast mechanism to map a virtual address to a physical address.
- The Software (OS)
  - Sets up the mappings that the hardware uses.
  - Manages the allocation of physical memory.
  - Implements policies about how memory is shared.

System calls that manipulate address spaces

- **`pid_t fork(void)`**
  - Creates a new process (called a child process)
  - The child is an exact (almost) copy of the parent

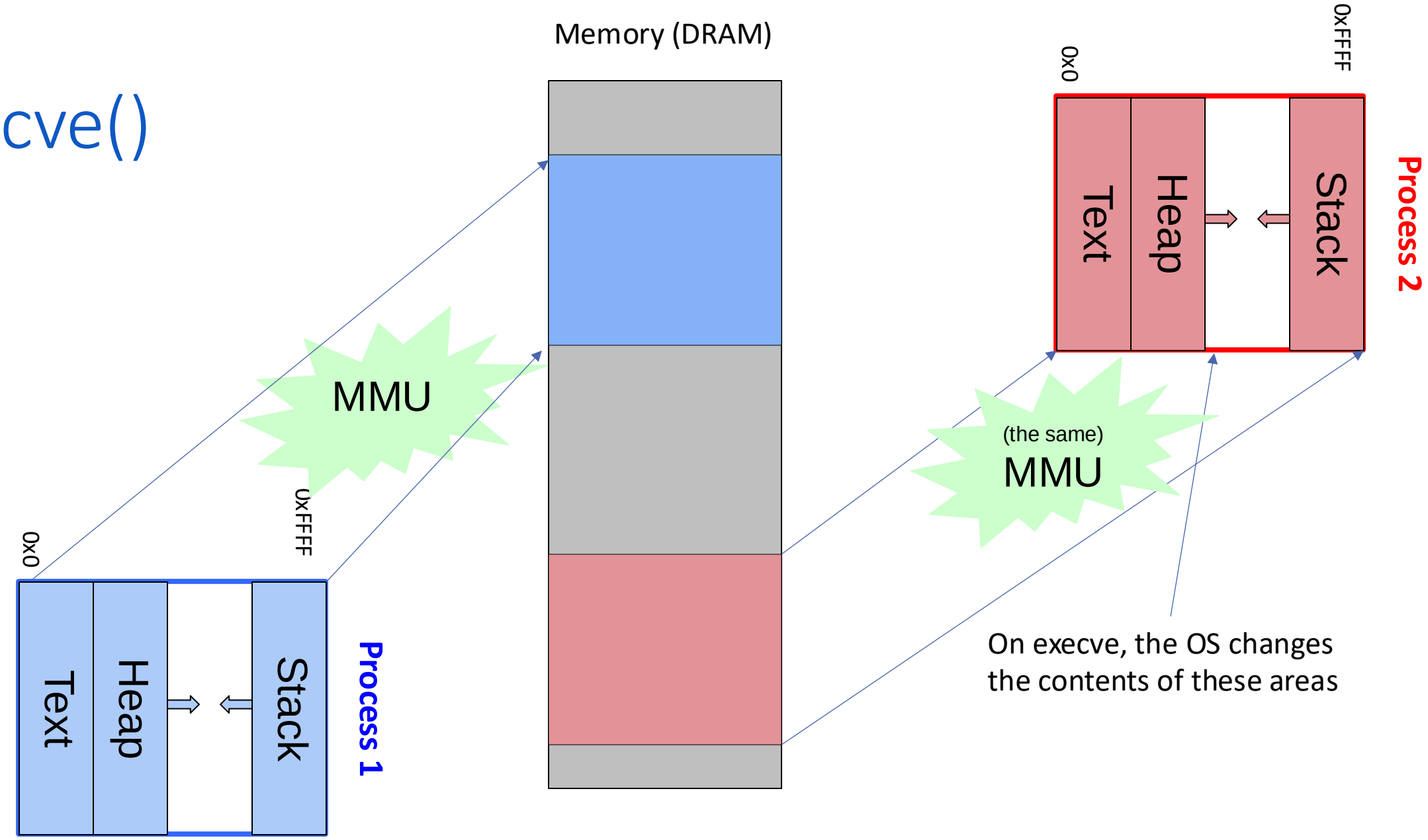
- **`int execve(const char *path, char *const argv[], char *const envp[])`**
  - Replace the current process image with the program image stored in file.
  - There is a whole family of function calls that all ultimately invoke the `execve` system call.
    - `execl, execlp, execle,`
    - `execv, execvp, execvpe (or execvP on a MAC, FreeBSD)`

pid = fork()



This all happens *instantly*?!
Something here is a lie. Start pondering what.
We'll talk more in a few lectures.

execve()


Memory (DRAM)
0x0
0xFFFF
Text
Heap
Stack
Process 2
MMU
(the same)
MMU
0x0
0xFFFF
Text
Heap
Stack
Process 1
On execve, the OS changes
the contents of these areas

Exec and friends

- **`int execve(const char *path, char *const argv[], char *const envp[])`**
  - Typically `path` is the pathname of a command you want to execute, e.g., `./myprog, /bin/ls`
  - `argv` is an argument vector -- it is what is passed to main, e.g.,
    - `int main(int argc, char *argv[])`
  - `envp` is an environment: the environment is a set of name/value pairs that are frequently used to communicate 'environmental' information to processes: where should the process look for commands, what OS are we running, etc.
- **`int execvp(const char *path, char *const argv[])`**
  - If the parameter path does not begin with "/", or ".", or "..", then `execvp` searches for parameter path in each directory listed in the PATH environment variable (just like the shell does).

In-class activity – build a baby shell!