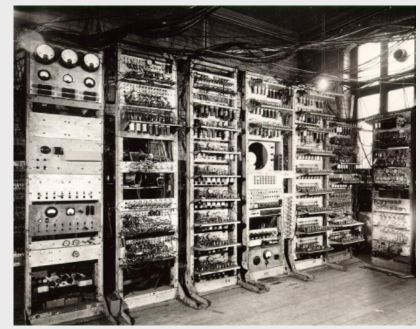
Virtual Memory Introduction

- Topics
  - An historical perspective on Virtual Memory – figure out what we want out of a virtual memory system.
  - What are the various approaches to implementing VM?
- Learning Objectives:
  - Explain what VM provides and why it is (usually) necessary.
  - Identify places where you may not need VM.
  - Explain different models of virtual memory and be able to compare and contrast them.

With thanks to Geoffrey Challen for the slides on which this is based.

Batch Processing (1)

The Computer



Program 2

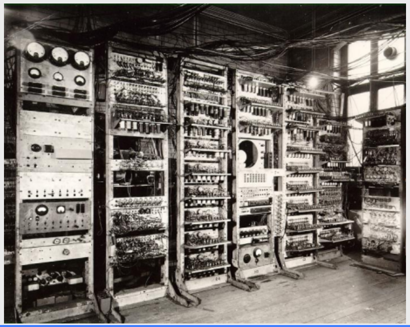
2.7182818284590
452353602874713
526624977572470
9369995

Program 1

3.1415926535897
932384626433832
795028841971693
993751058209749
445923078164062

Batch Processing (2)

The Computer



Program 1

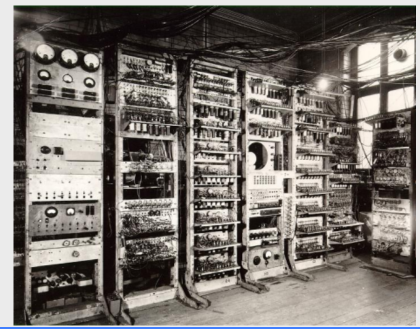
3.1415926535897
932384626433832
795028841971693
993751058209749
445923078164062

Program 2

2.7182818284590
452353602874713
526624977572470
9369995

Batch Processing (3)

The Computer



Program 1

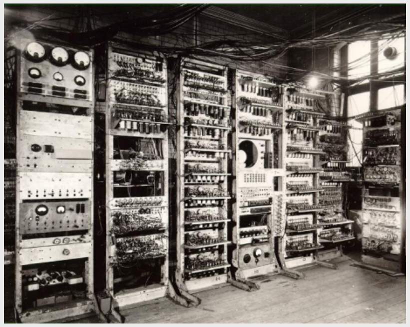
3

Program 2

2.7182818284590
452353602874713
526624977572470
9369995

Batch Processing (4)

The Computer



Program 2

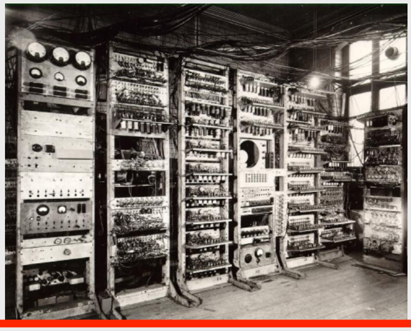
2.7182818284590
452353602874713
526624977572470
9369995

Program 1

3

Batch Processing (4)

The Computer



Program 2

2.7182818284590
45235360287471352662497757247093699959

Program 1

3

Exercise 1: List the advantages and disadvantages of this scheme

- What advantages does this scheme provide?

- What are its disadvantages?

Exercise 2: List some goals you might want to achieve when sharing a computer among multiple processes.

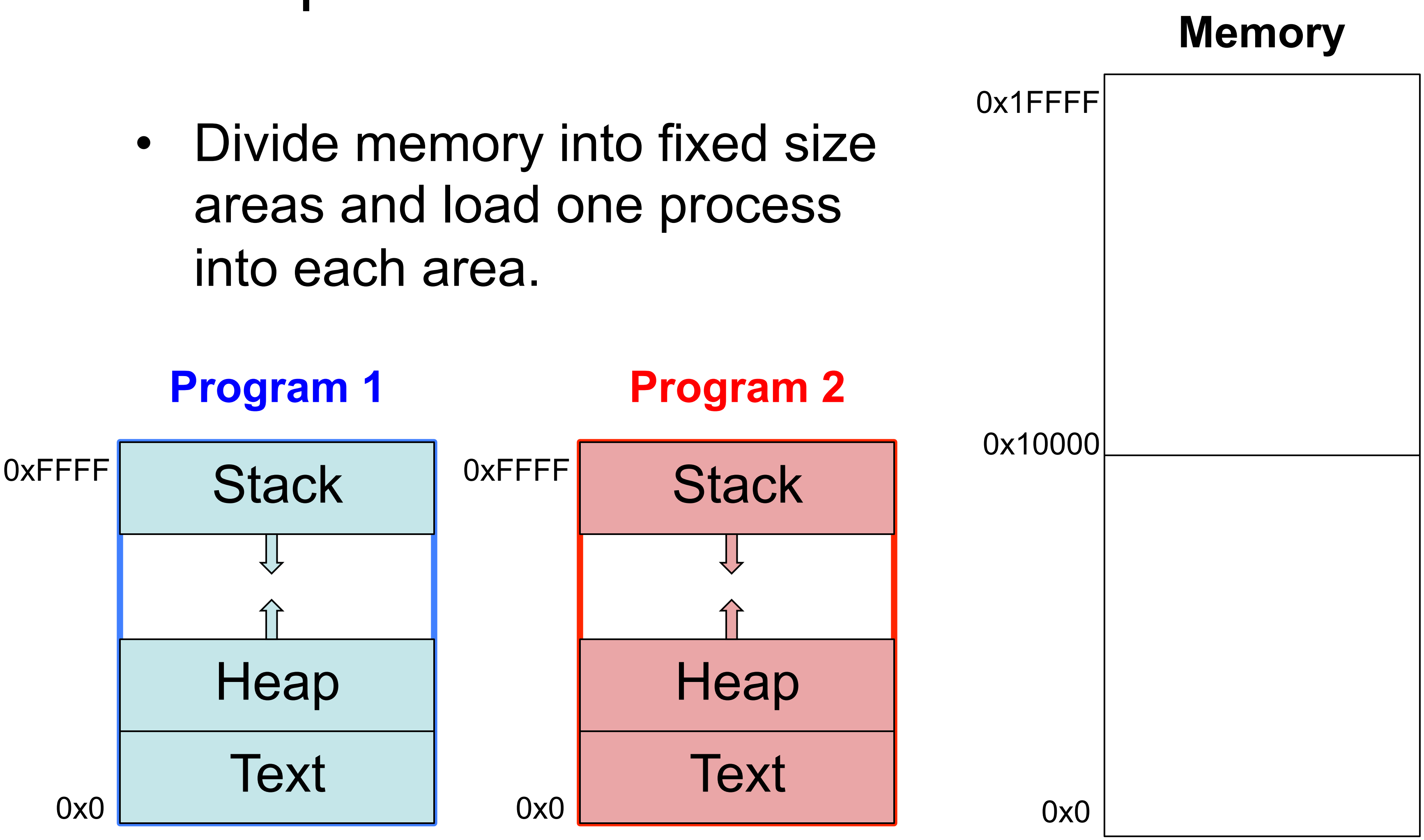
Our Goals

- Isolation
  - Processes should be unaware of other processes.
- Protection
  - Processes should not be able to interfere with each other.
  - I.e., Process A should not scribble on Process B's data.
- Performance
  - We want to use the processor efficiently.
  - No long waits while we change from one process to another.

We can get a good start if each process thinks it is the only process running on the machine!

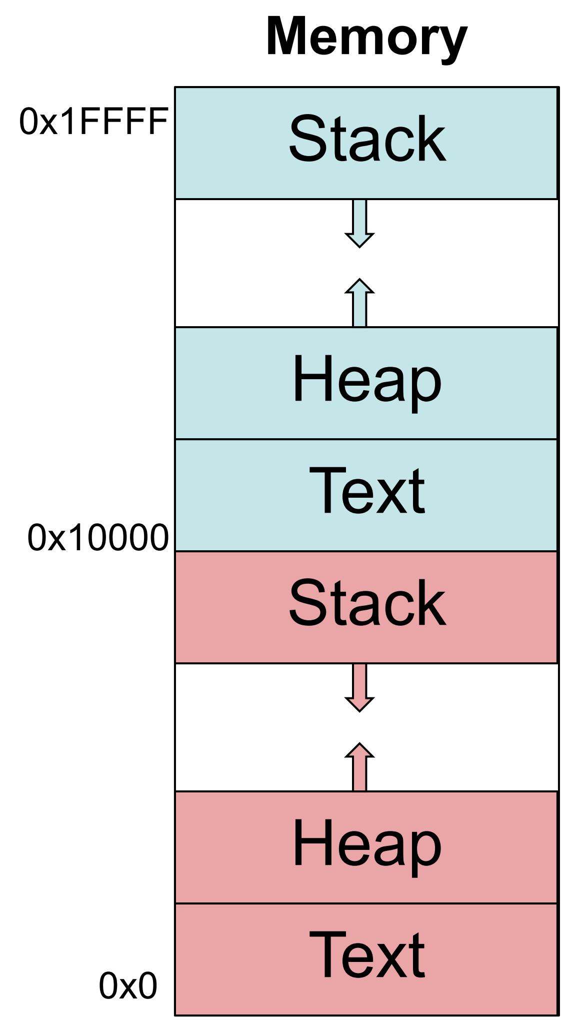
Simple Solution 1: Fixed Size Partitions

- Divide memory into fixed size areas and load one process into each area.



Simple Solution 1: Fixed Size Partitions

- Divide memory into fixed size areas and load one process into each area.
- As you load the program, translate addresses.
- Any problems?



Exercise 3: Critique Fixed partitioning

- List as many problems with this approach as you can:

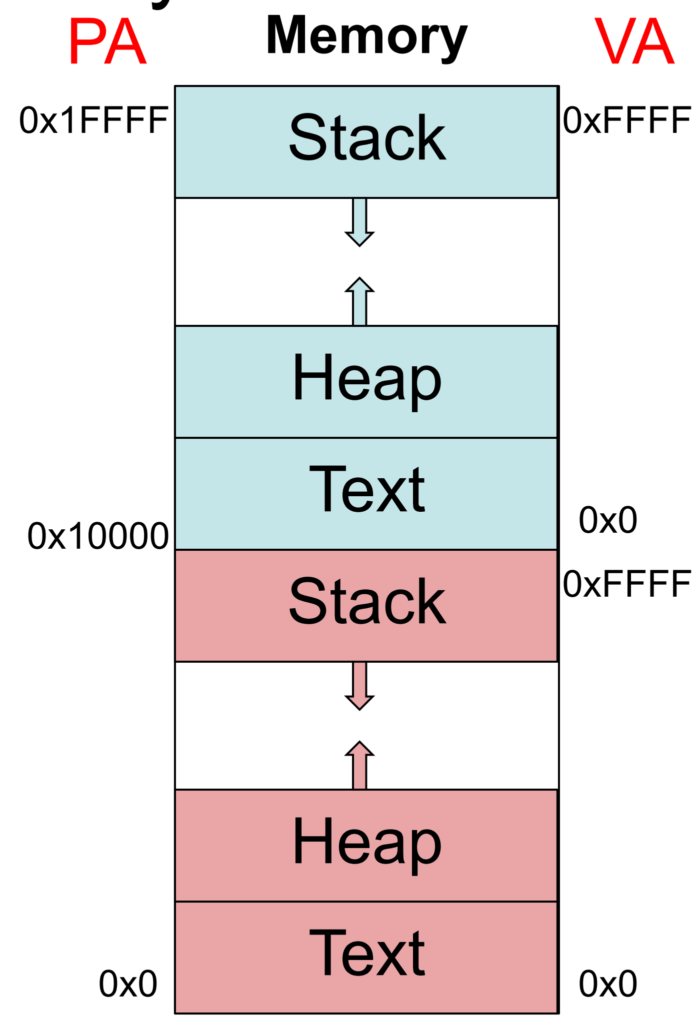
- Propose at least three different ways to cope with the problems:

The fundamental problem

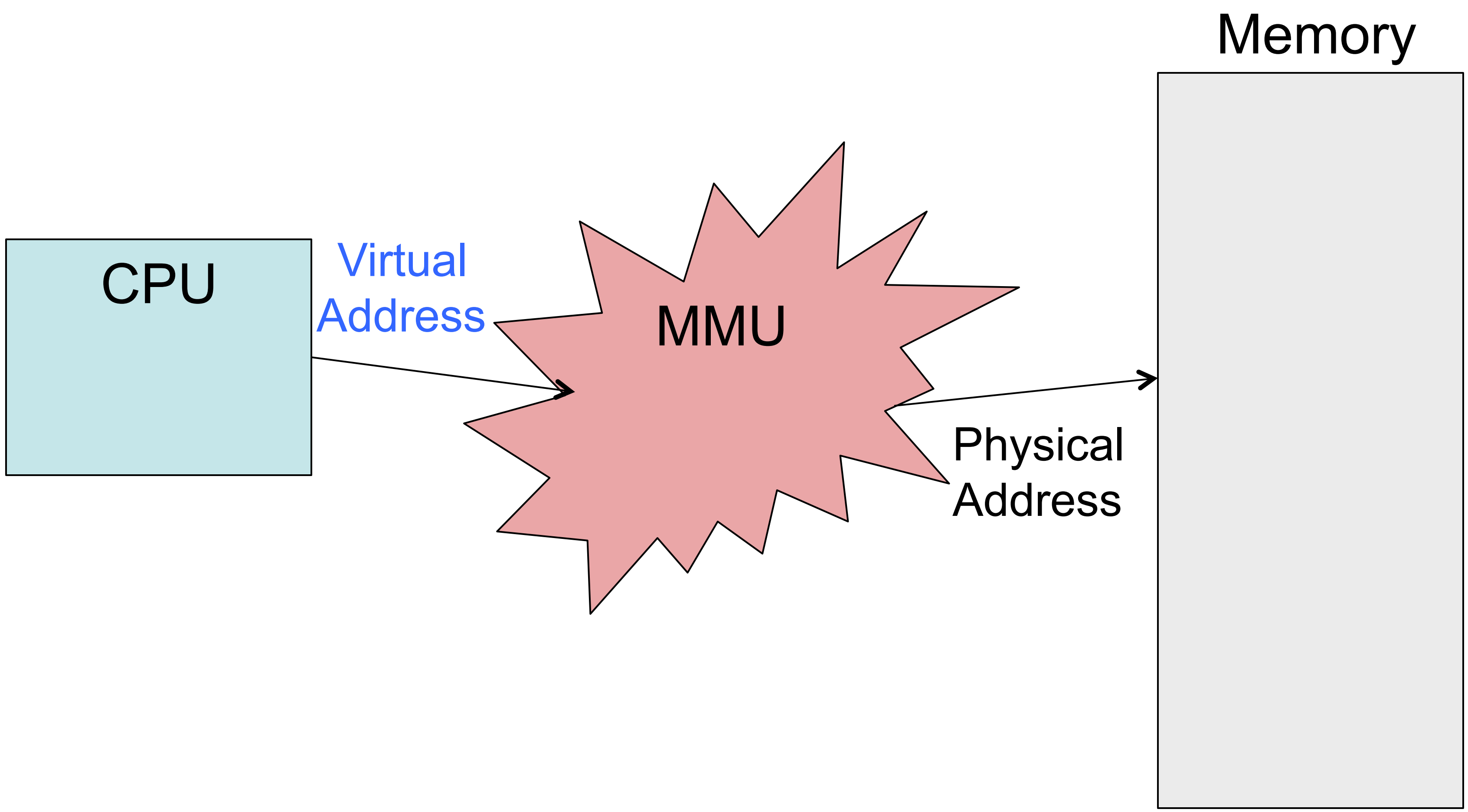
- We can think of the previous approach as static relocation.
- Static relocation is limited:
  - Processes had to be fixed size.
  - We were limited to the number of processes available.
  - Each time a process moves, there is a lot of work to be done.
- Isn't there a better way?
  - Could we make the relocation dynamic?
  - How?

Indirection: The Answer to Any Question

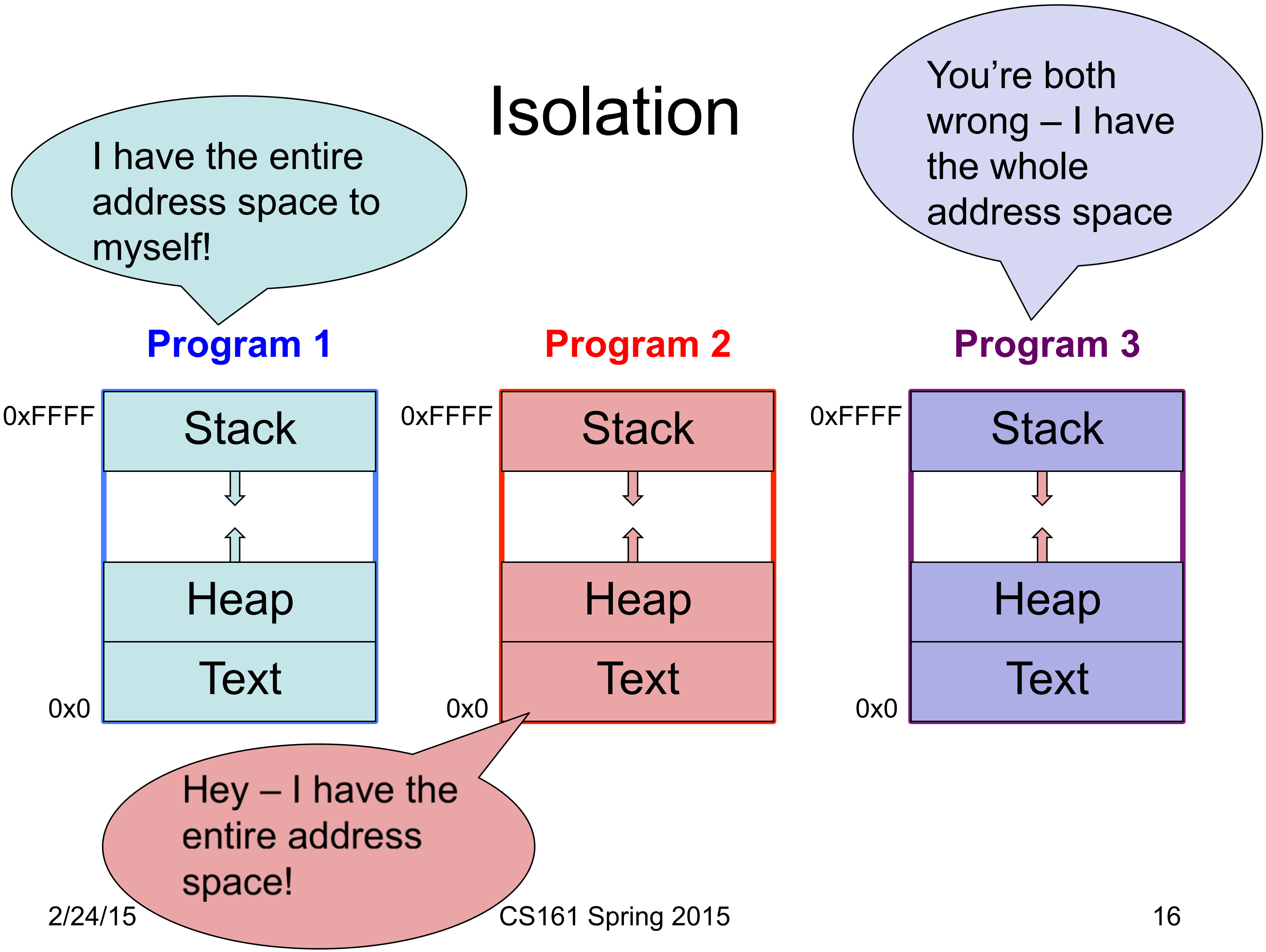
- Let's introduce make-believe addresses (virtual addresses).
  - Processes can use whatever make-believe addresses they want.
  - "We" (the OS, the hardware, the Wizard of Oz, someone) will translate those make-believe addresses into physical addresses.



The MMU: A Translation Unit



Isolation

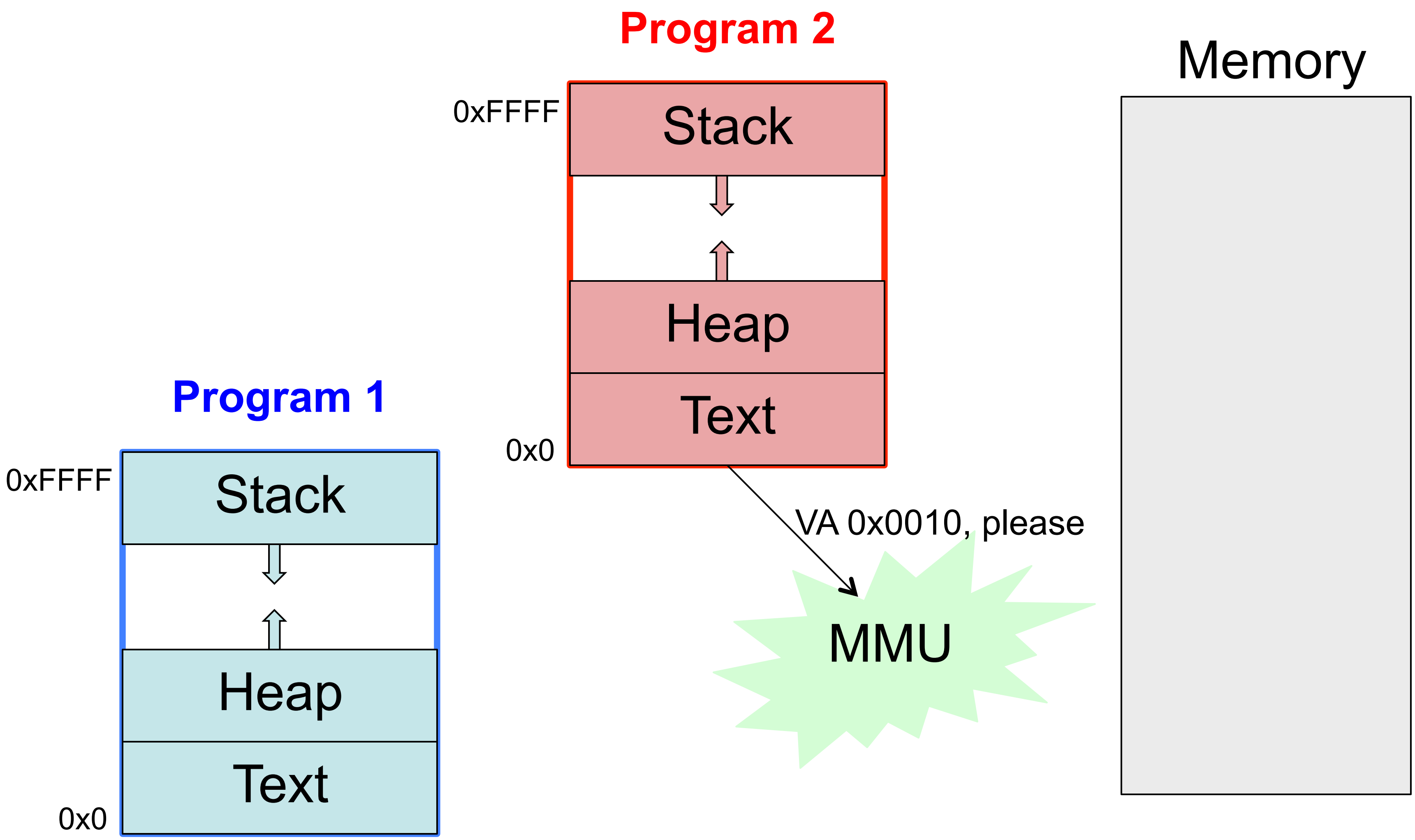

I have the entire address space to myself!
Program 1
0xFFFF
Stack
Heap
Text
0x0
Program 2
0xFFFF
Stack
Heap
Text
0x0
Program 3
0xFFFF
Stack
Heap
Text
0x0
You're both wrong – I have the whole address space
Hey – I have the entire address space!

It’s all a big lie

- In effect, virtual memory is all a big lie, an illusion.
- Just because your code is loaded at **Virtual Address (VA) 0** and your stack starts at **VA 0x80000000** does NOT mean that:
  - **a)** the machine **has** 2GB of memory or
  - **b)** you can use all of it (or even a lot of it) at once.
- Just because your virtual addresses are contiguous doesn't mean the physical ones are.

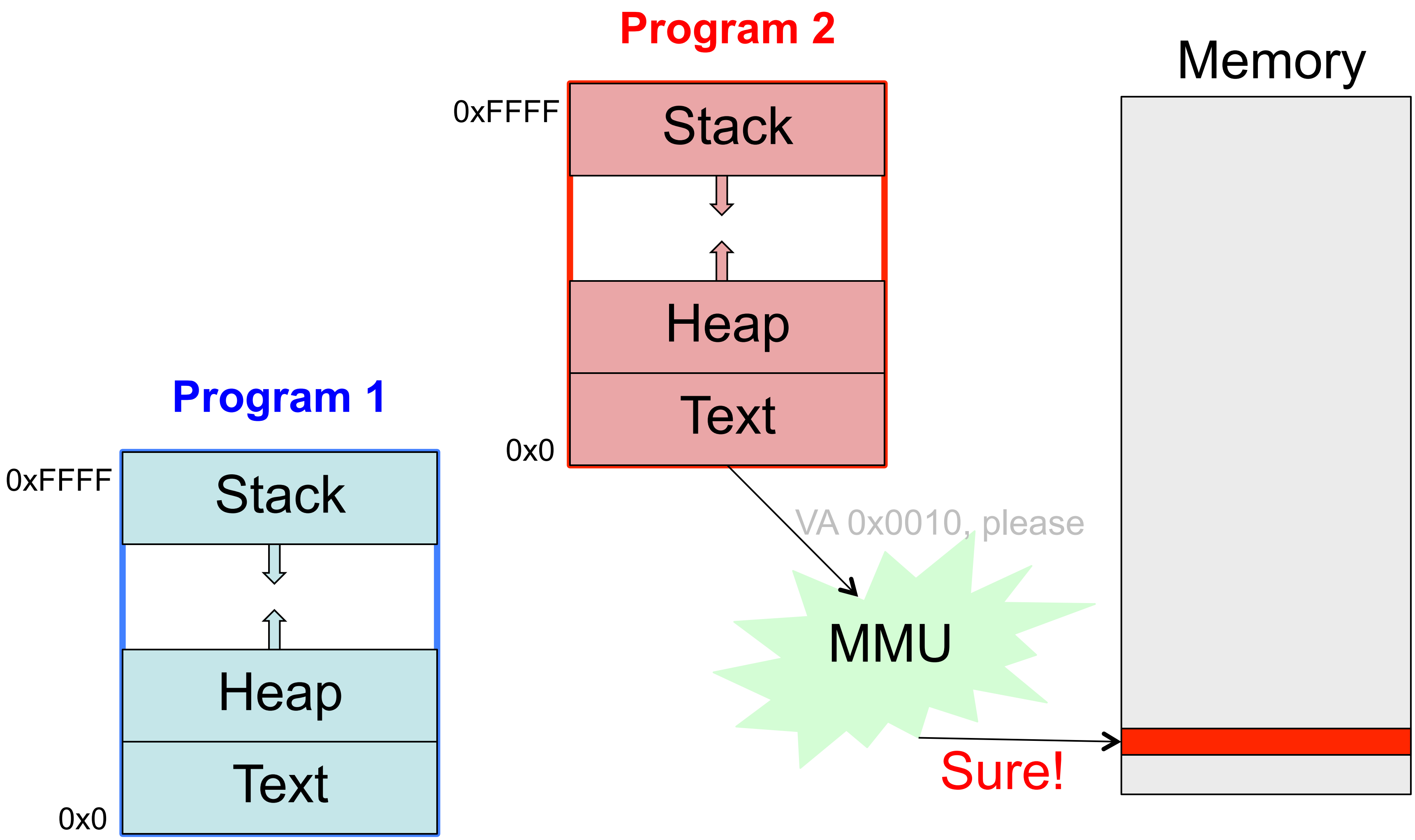
But it is a very useful lie!

Protection

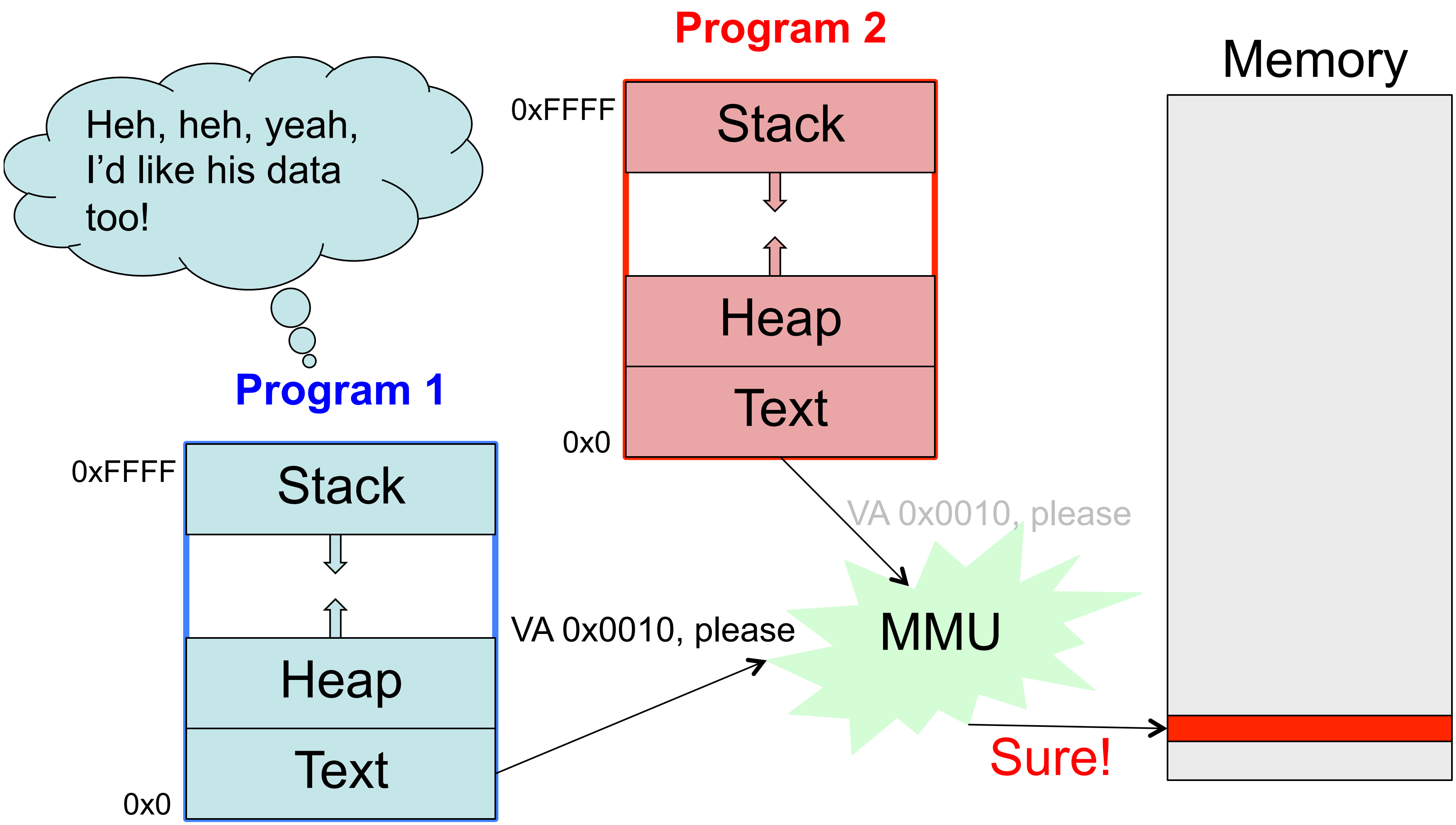


Program 2
0xFFFF
Stack
Heap
Text
0x0
Memory
Program 1
0xFFFF
Stack
Heap
Text
0x0
VA 0x0010, please
MMU

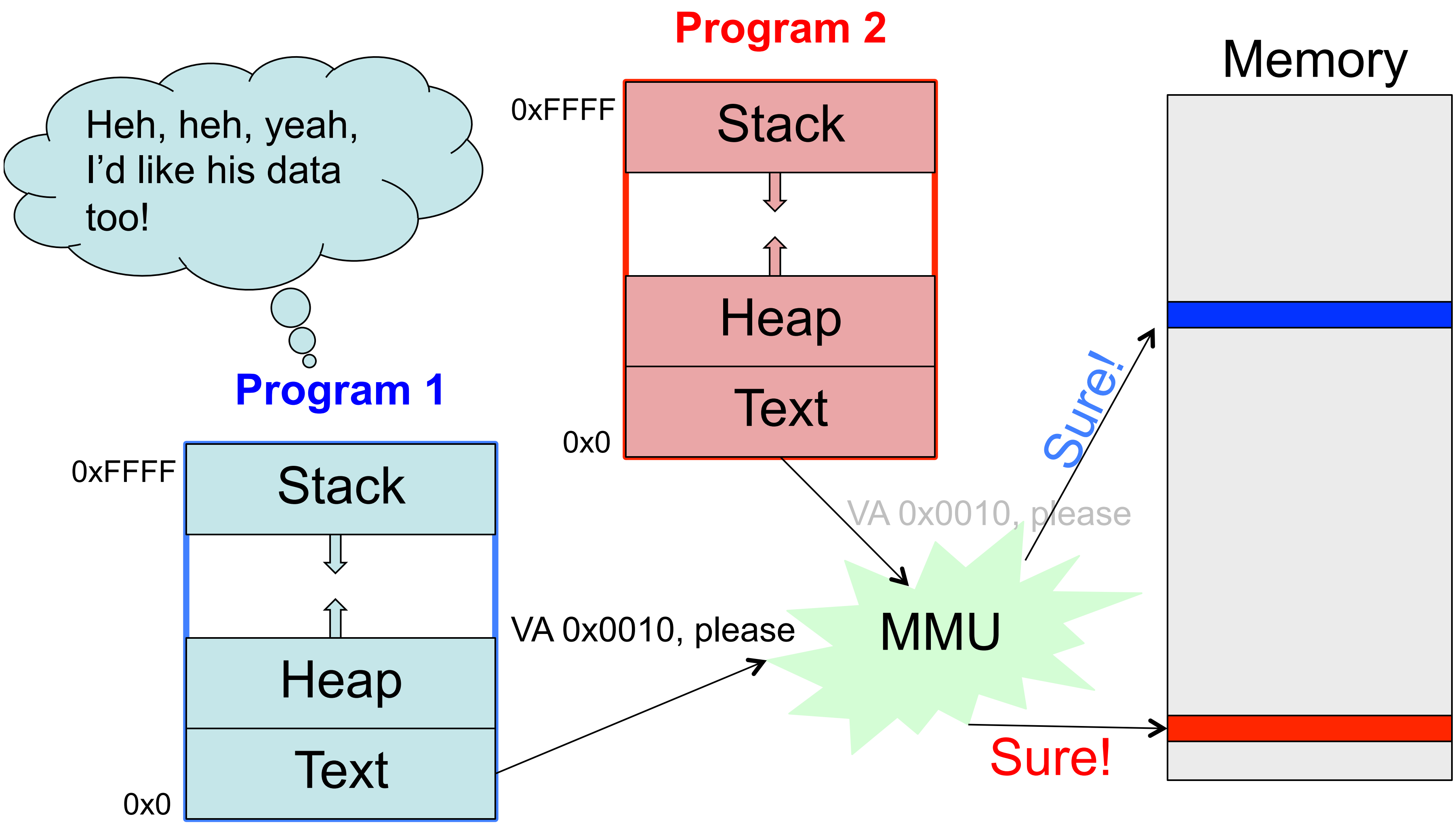
Protection


Program 2
0xFFFF
Stack
Heap
Text
0x0
Memory
Program 1
0xFFFF
Stack
Heap
Text
0x0
VA 0x0010, please
MMU
Sure!

Protection


Program 2
0xFFFF
Stack
Heap
Text
0x0
Memory
Heh, heh, yeah, I'd like his data too!
Program 1
0xFFFF
Stack
Heap
Text
0x0
VA 0x0010, please
MMU
VA 0x0010, please
Sure!

Protection



Heh, heh, yeah, I'd like his data too!
Program 1
0xFFFF
Stack
Heap
Text
0x0
Program 2
0xFFFF
Stack
Heap
Text
0x0
VA 0x0010, please
VA 0x0010, please
MMU
Sure!
Sure!
Memory

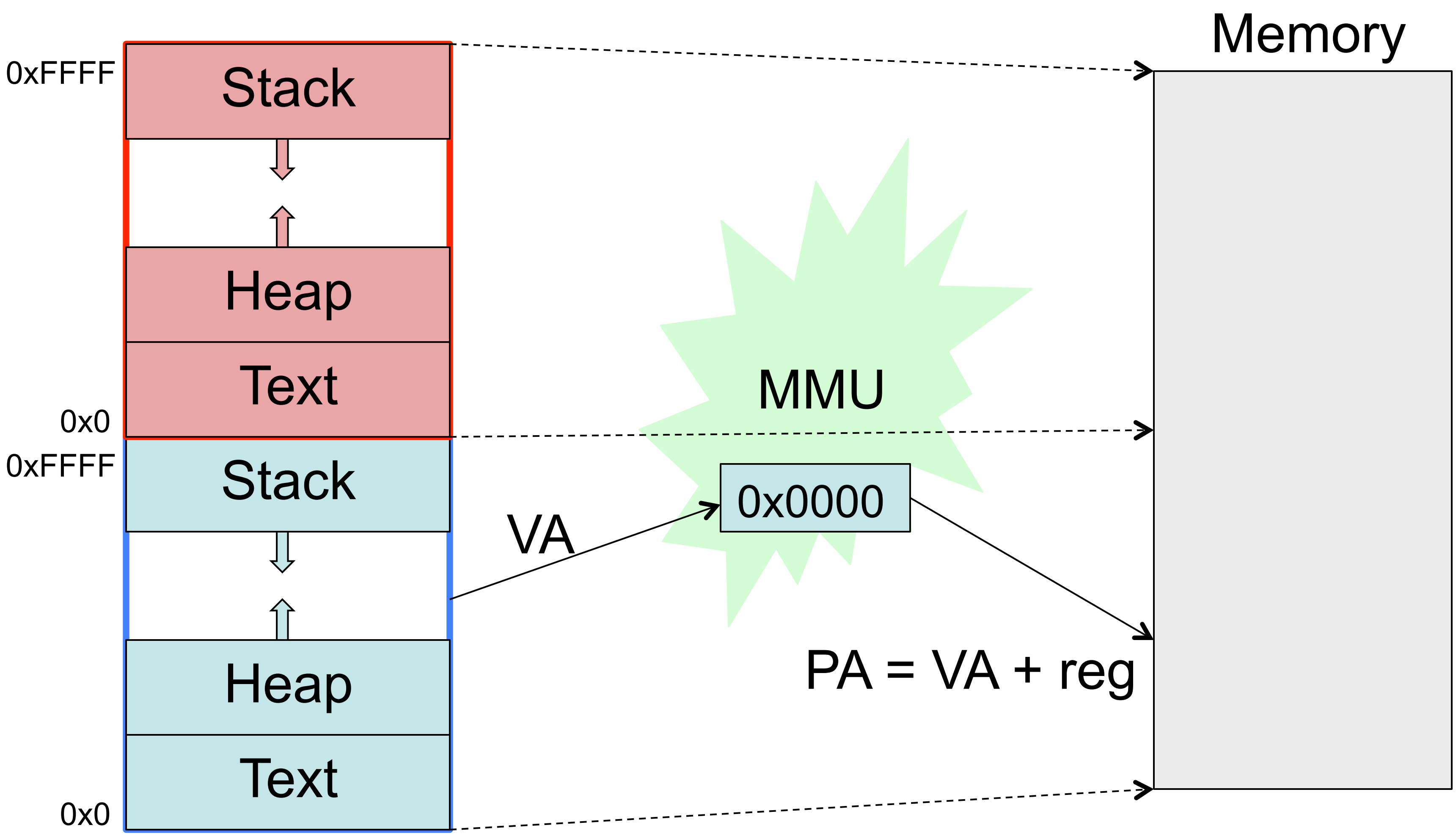
Isolation, Protection, Performance

- We said that we wanted a solution that provides isolation, protection, and performance.
- Adding a level of indirection so that processes can use virtual addresses provide isolation and protection, but what about performance?
  - OS can allocate memory to processes in a manner that provides best performance.
  - Goal is to let processes run as if they had as much memory as we have disk, but with performance of memory.
  - When things go bad, the system can appear to have only as much memory as we have physical memory, but it runs at the speed of disk. That's pretty awful!
- But, to make this work, we have to figure out how to make the translation efficient!

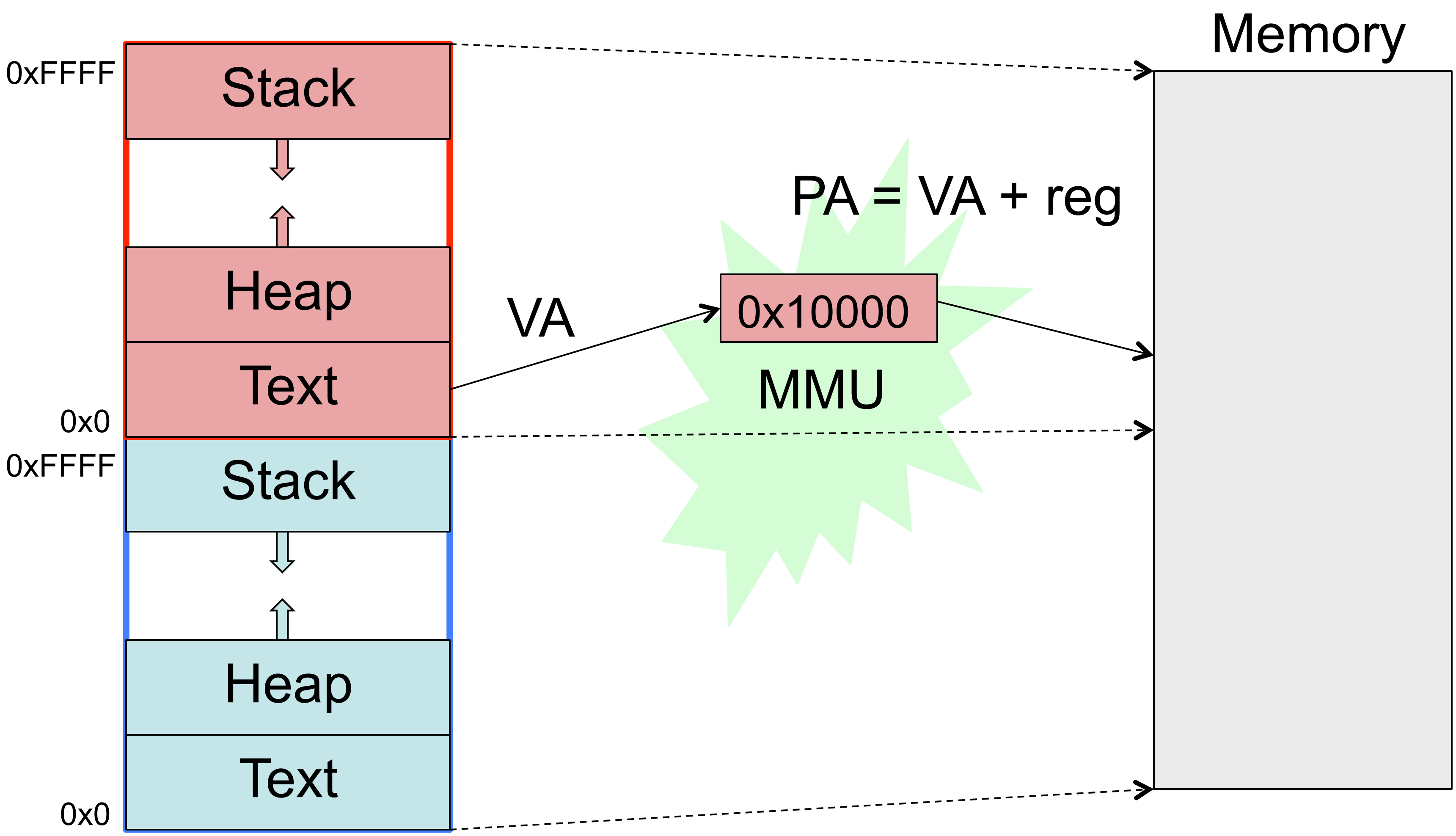
Two Dimensions to Performance

1. How we perform mapping.
   - Actual hardware that does the mapping.
   - Kernel data structure that manage the hardware.
2. How we manage memory and allocate it to processes.
   - Allocate memory to processes that will use it well.
   - Avoid:
     - Internal fragmentation: memory that we allocate to processes, but is unused.
     - External fragmentation: Chunks of memory that the hardware makes us allocate to processes, even if the process cannot use it.

Dynamic Relocation



Dynamic Relocation



Exercise 4: Critique Dynamic Relocation

- Advantages:

- Disadvantages:

- Any other functionality you want?

Extension 1: Base and Bounds

- One of the simplest address translation strategies!
- For each process we maintain a base and bounds.
- Translating a VA:

```
if VA > bounds:
      return error
else:
      return base + VA
```

Process ID	Base	Bounds
1	0	1000
2	INVALID	2000
3	4000	500
4	3000	500
5	5000	1000

```
Translate:
A: PID = 1 VA = 40
B: PID = 4 VA = 750
C: PID = 5 VA = 900
D: PID = 2 VA = 3000
E: PID = 3 VA = 0
```

Exercise 5: Critique Base and Bounds

- Advantages:

- Disadvantages:

- Extensions:

Extension 2: Segmentation

- Notice that processes contain different kinds of data that can be treated differently:
- Code: (usually) Read Only, doesn't change size
- Static data: RW, doesn't change size
- Heap: RW, size increased on request
- Stack: RW, size increased implicitly
- Add multiple base & bound registers:
  - Use one for each segment.
  - Add protection on each segment.

Segmentation: Translation

```
segment = find_segment(VA)
if get_offset(VA) > segment.bounds:
  return error
else:
  return segment.base + get_offset(VA)
```

- Different ways of getting segment and offset:

	find_segment	get_offset
Implicit	data comes from data segment, code from code segment	Use address in instruction.
Explicit	Segment identified in instruction	Use address in instruction
Partitioned VA	High bits of VA	Low bits of VA

Segmentation: Example

Process 1 Segment Table

	Segment ID	Base	Bounds	Protection
Code	1	150	50	RO
Code (Library)	2	500	100	RO
Static Data	3	450	50	RW
Heap	4	600	200	RW
Stack	5	800	200	RW

3	2	1	0
Segment Number	**Offset**		

Read
VA=1040
PA=

Write
VA=2000
PA=

Read
VA=8010
PA=

Write
VA=5180
PA=

Segmentation: Operations

- Grow a segment: Increase bounds (assuming space)
- Move a segment: Change base
- Free memory: Put allocated space on free list
- What do I do if I need a bunch of contiguous space and I have enough free space, but it's chopped up?

Move segments around
To coalesce free space

Exercise 6: Segmentation Pros and Cons

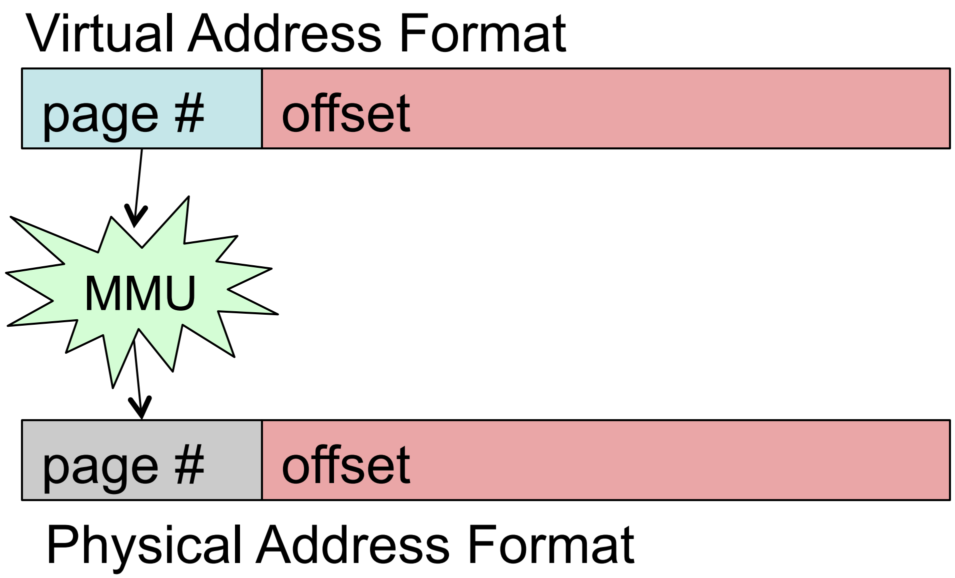
- Advantages:

- Disadvantages:

- Extensions:

Extension 3: Paging

- Let's tackle two problems at once:
  - Make allocation problem trivial: fixed sized units (pages)
    - No more bounds; it is implied with the fixed size
  - Use space efficiently: make that fixed size small
    - The things we used to call segments are now collections of pages.



Exercise 7: Paging: Pros and Cons

- Pros

- Cons

Single Level Page Table

- What happens if we don't do anything clever?
  - 4KB pages => 12 bits of offset
  - That leaves 20 bits for page numbers (even worse in a 64-bit address space!)

Virtual Address Format

page # (20 bits)	Offset (12 bits)

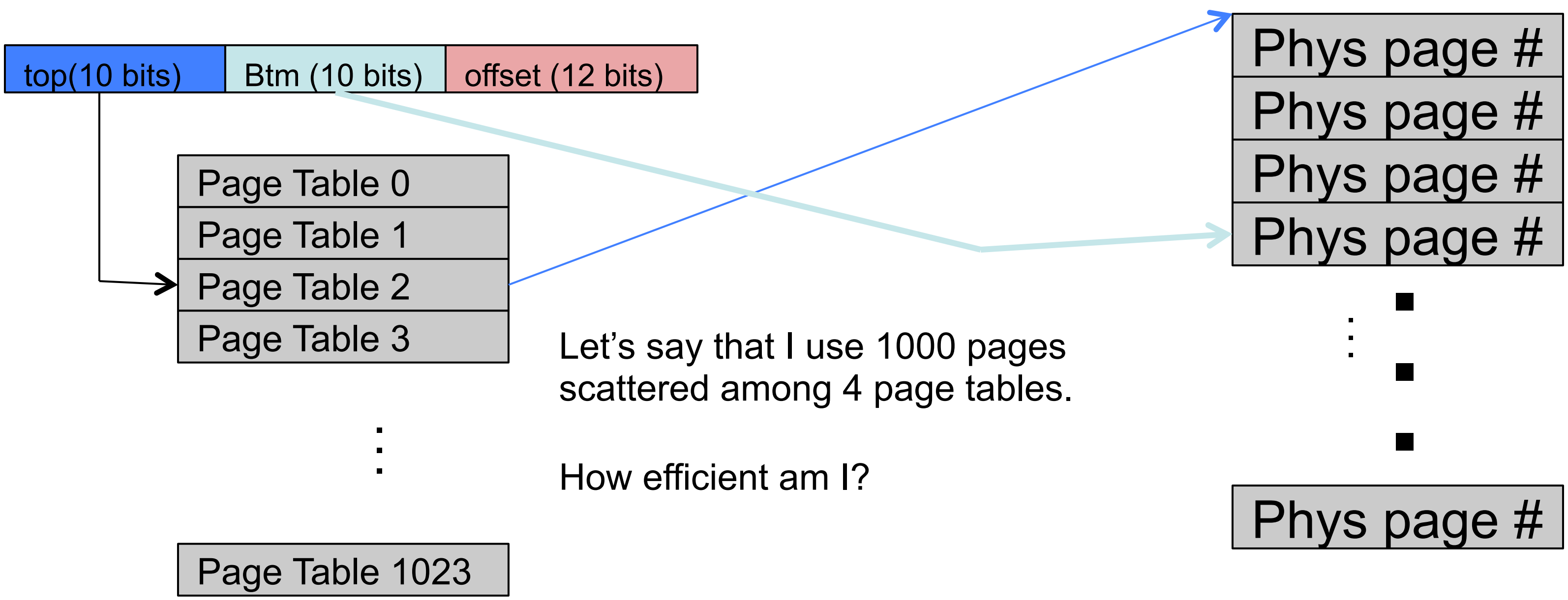
20 bits => 1M entries
Let's say my program only needs 1000 pages
Utilization is < 0.01 %

What do we do?????

Phys page #
Phys page #
Phys page #
Phys page #
▪
▪
▪
Phys page #

Two-Layer Page Table

- Let's cut our 20-bit page number into 2 parts, each of 10 bits.
- The top 10-bits select a page table; the bottom10 bits select a page within that page table.

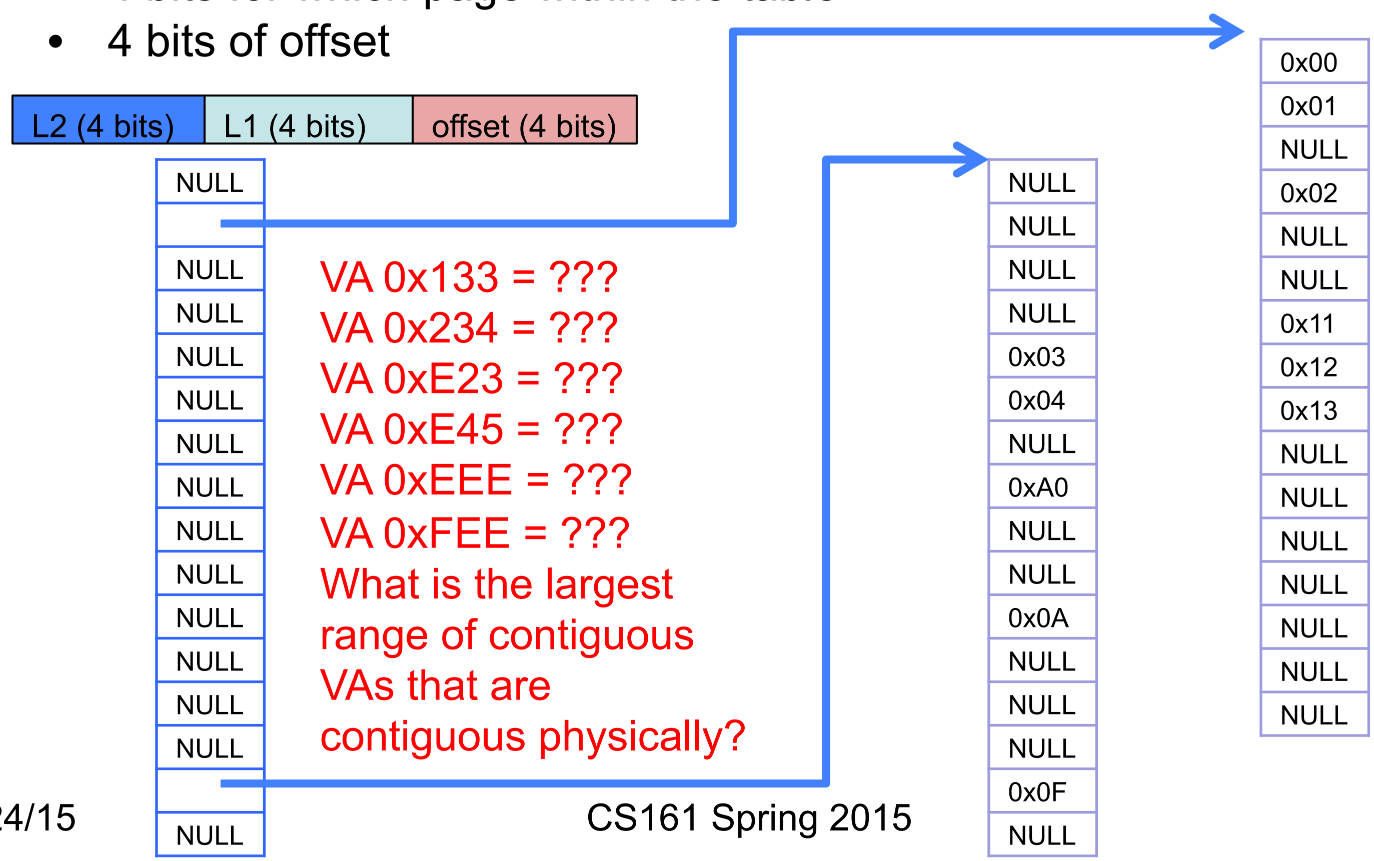


Let's say that I use 1000 pages scattered among 4 page tables.

How efficient am I?

Page Table Translations

- To make this tractable, let's assume I have 12 bit addressing:
  - 4 bits to select a page table
  - 4 bits for which page within the table
  - 4 bits of offset



Exercise 8: Practice Questions

- Using the tables on the previous page:
    1. What is the maximum size of physical memory?
    2. What is the size of the virtual address space?
    3. How many pages are in use?
    4. What is the size of an entry in the L1 pages tables?
    5. Starting at physical address 0x0 – how many contiguous pages are in use? What are their virtual addresses?
    6. Let’s say that we wanted to leave the virtual address the same size, but support twice the amount of physical memory. How could we do that?