Memory Management

PSU CS 532

Prof. Karen L. Karavanic

Portland State
UNIVERSITY

Practice Problem revisited (multilevel cache)

- Suppose that in 1000 memory references there are 40 misses in the first-level cache and 20 misses in the second-level cache. What are the various miss rates?
  - One standard calculation:
  - *Local Miss rate* is the number of misses in a cache divided by the total number of memory accesses to this cache.
  - *Global miss rate* is the number of misses in the cache divided by the total number of memory accesses generated by the processor

Memory Management

Memory – a linear array of bits, bytes, words, pages ...

- Each *byte* is named by a unique memory address
- Holds instructions and data for OS and user processes

Each process has an *address space* containing its instructions, data, heap and stack regions

When processes execute, they use addresses to refer to things in their memory (instructions, variables etc)

... But how do they know which addresses to use?

Addressing Memory

Cannot know ahead of time where in memory instructions and data will be loaded!

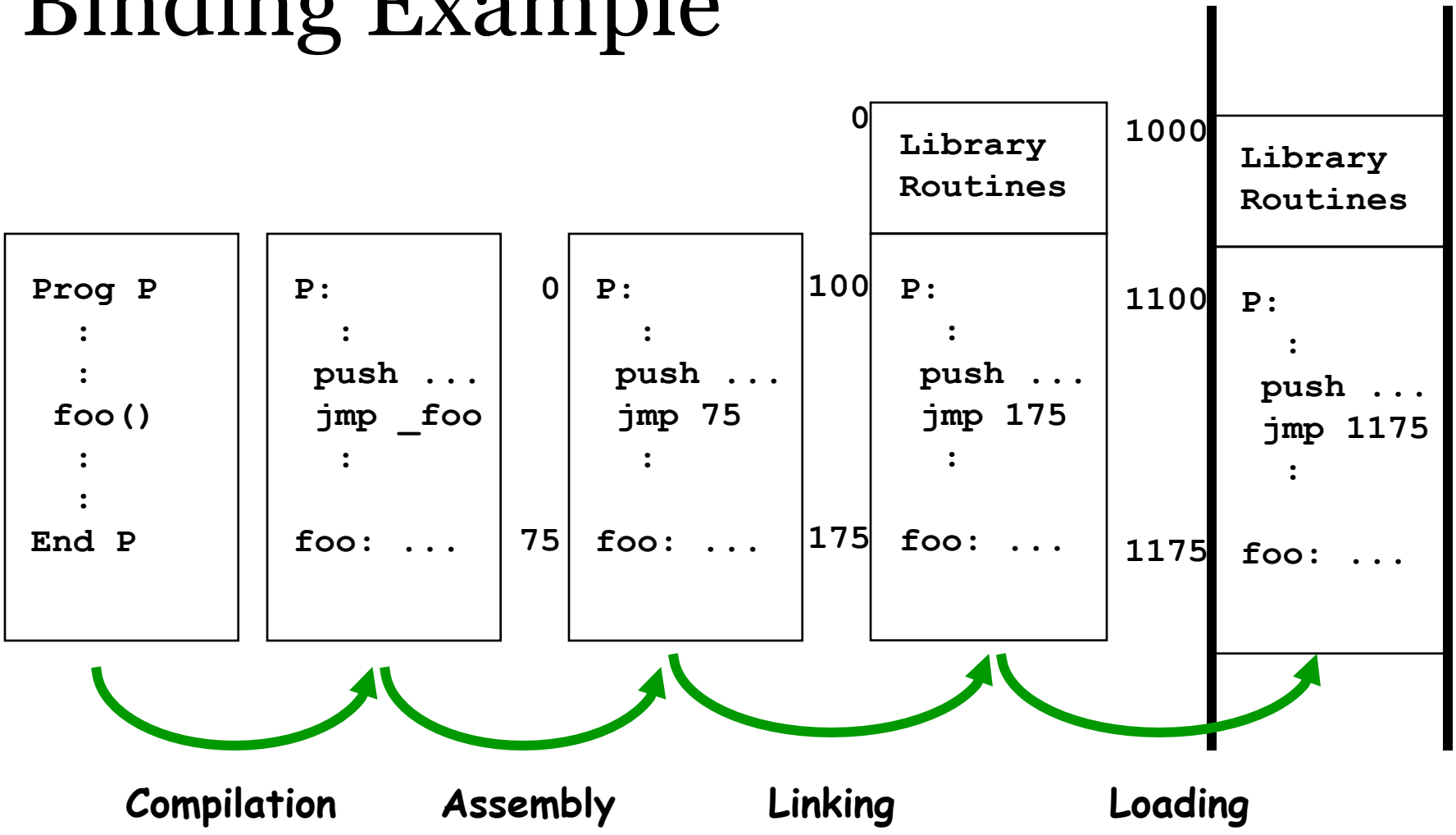
- so we can't hard code the addresses in the program code

Compiler produces code containing names for things, but these names can't be physical memory addresses

Linker combines pieces of the program from different files, and must resolve names, but still can't encode addresses

We need to bind the compiler/linker generated names to the actual memory locations before, or during execution

Binding Example



Relocatable Addresses

How can we execute the same processes in different locations in memory without changing memory addresses?

How can we move processes around in memory during execution without breaking their addresses?

Key Concept: Virtual Addresses

Addresses issued by running processes are "virtual". they must be translated.

Hardware translates the virtual addresses at runtime. Always. Every address. On every reference.

Simple Idea: Base/Limit Registers

Simple runtime relocation scheme

- Use 2 registers to describe a processes memory partition
- Do memory addressing indirectly via these registers

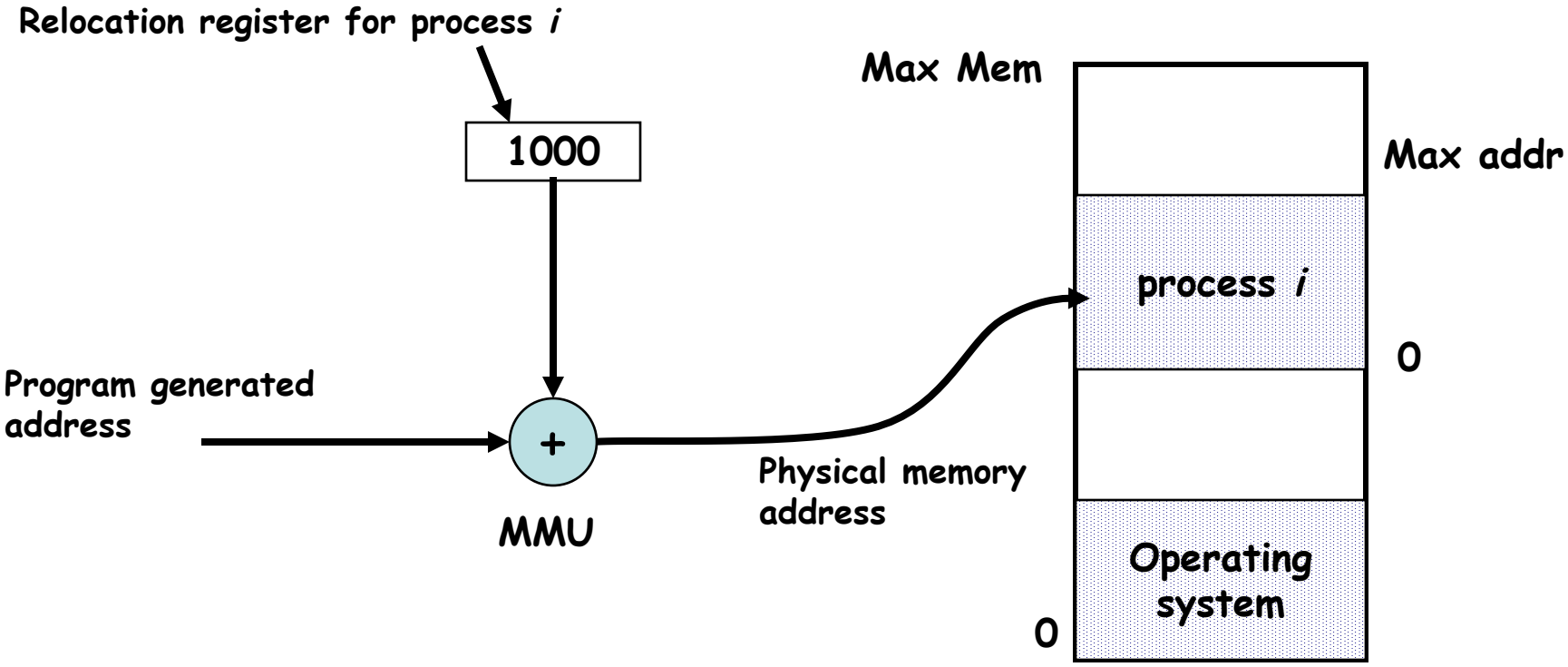
For every address, before going to memory ...

- Add to the base register to given virtual address
- Compare result to the limit register (& abort if larger)

Dynamic Relocation via Base Register

Memory Management Unit (MMU)

- Dynamically converts re-locatable logical addresses to physical addresses

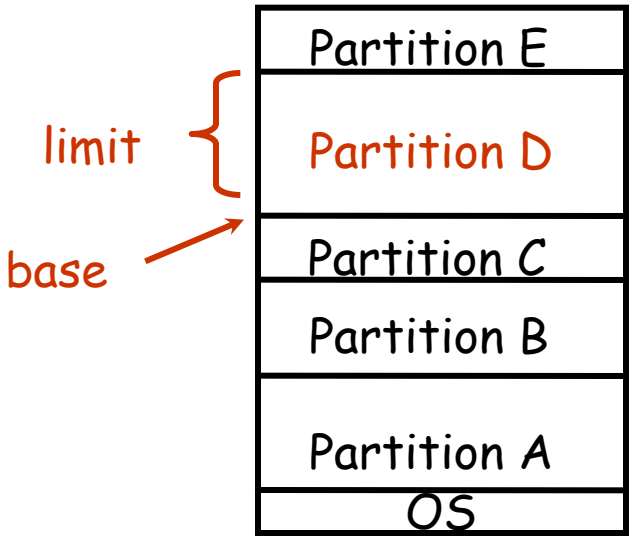


Multiprogramming

Multiprogramming: a separate partition per process

What happens on a context switch?

- Store process base and limit register values
- Load new values into base and limit registers



Quick Quiz

Is it possible for two different processes to emit the same virtual address?

Swapping

When a program is running...

- The entire program must be in memory
- Each program is put into a single partition

When the program is not running why keep it in memory?

- Could swap it out to disk to make room for other processes

Over time...

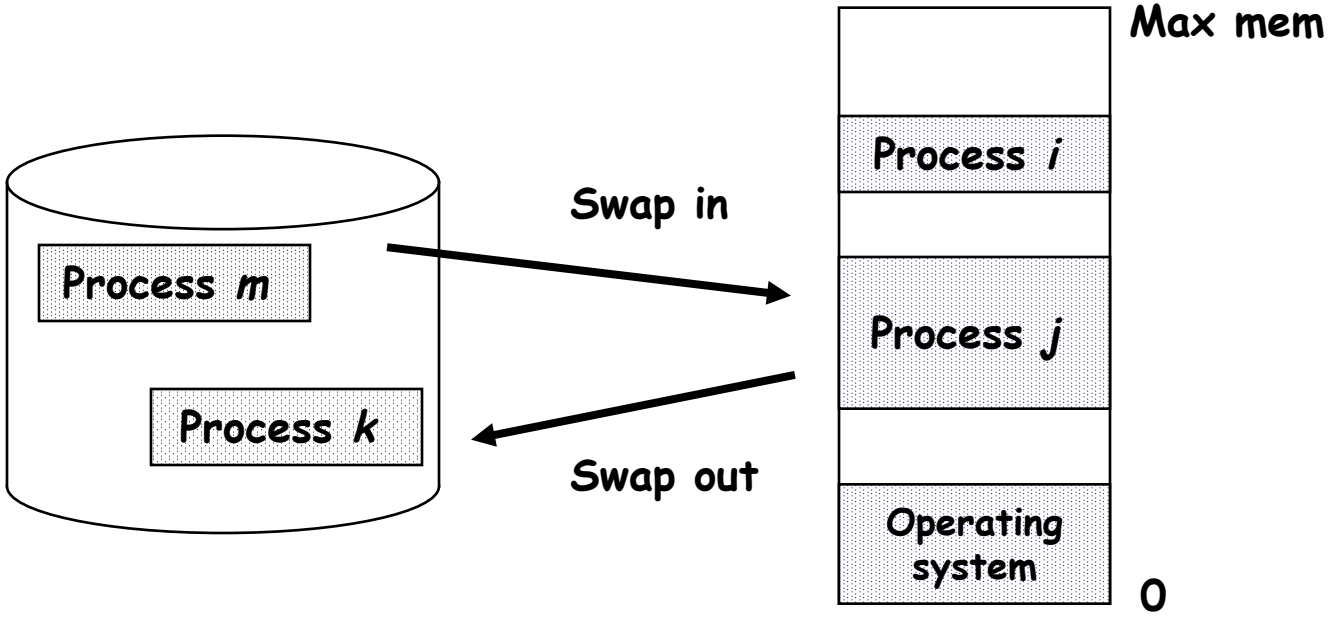
- Programs come into memory when they get swapped in
- Programs leave memory when they get swapped out

Swapping

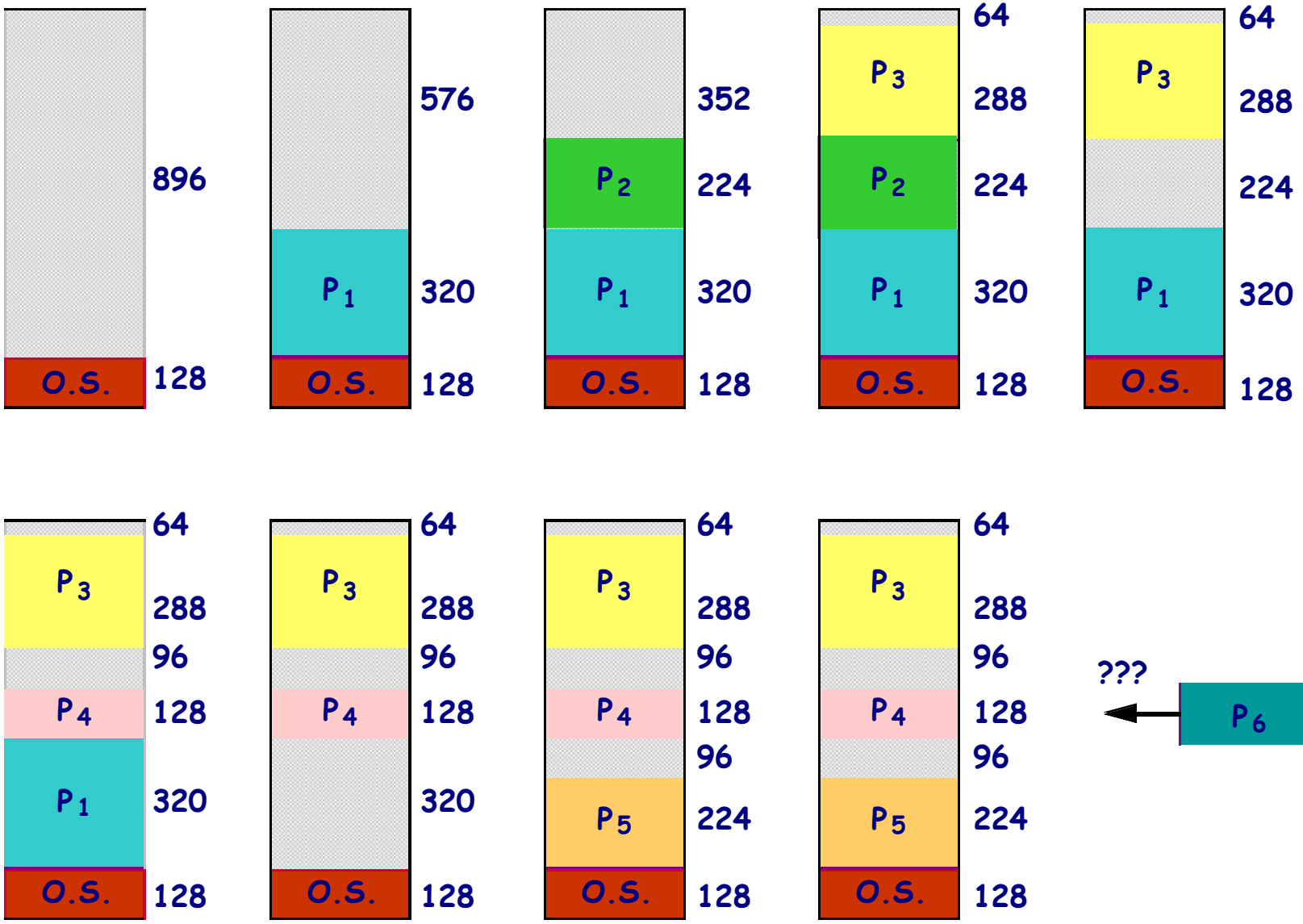
Benefits of swapping:

Allows multiple programs to be run concurrently

… more than will fit in memory at once



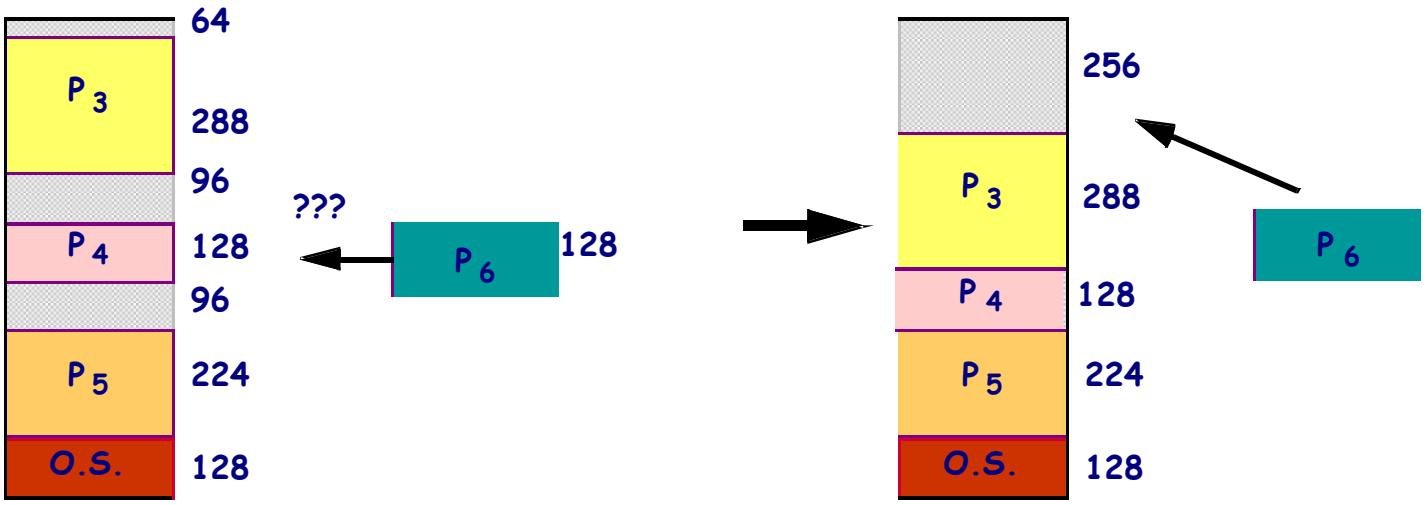
Fragmentation


896
O.S.
128
576
P1
320
O.S.
128
352
P2
224
P1
320
O.S.
128
64
P3
288
P2
224
P1
320
O.S.
128
64
P3
288
224
P1
320
O.S.
128
64
P3
288
96
P4
128
P1
320
O.S.
128
64
P3
288
P4
128
320
O.S.
128
64
P3
288
96
P4
128
96
P5
224
O.S.
128
64
P3
288
96
P4
128
96
P5
224
O.S.
128
???
P6
128

Dealing With Fragmentation

Compaction – from time to time shift processes around to collect all free space into one contiguous block

- Memory to memory copying overhead
- Memory to disk to memory for compaction via swapping!



How Big Should Partitions Be?

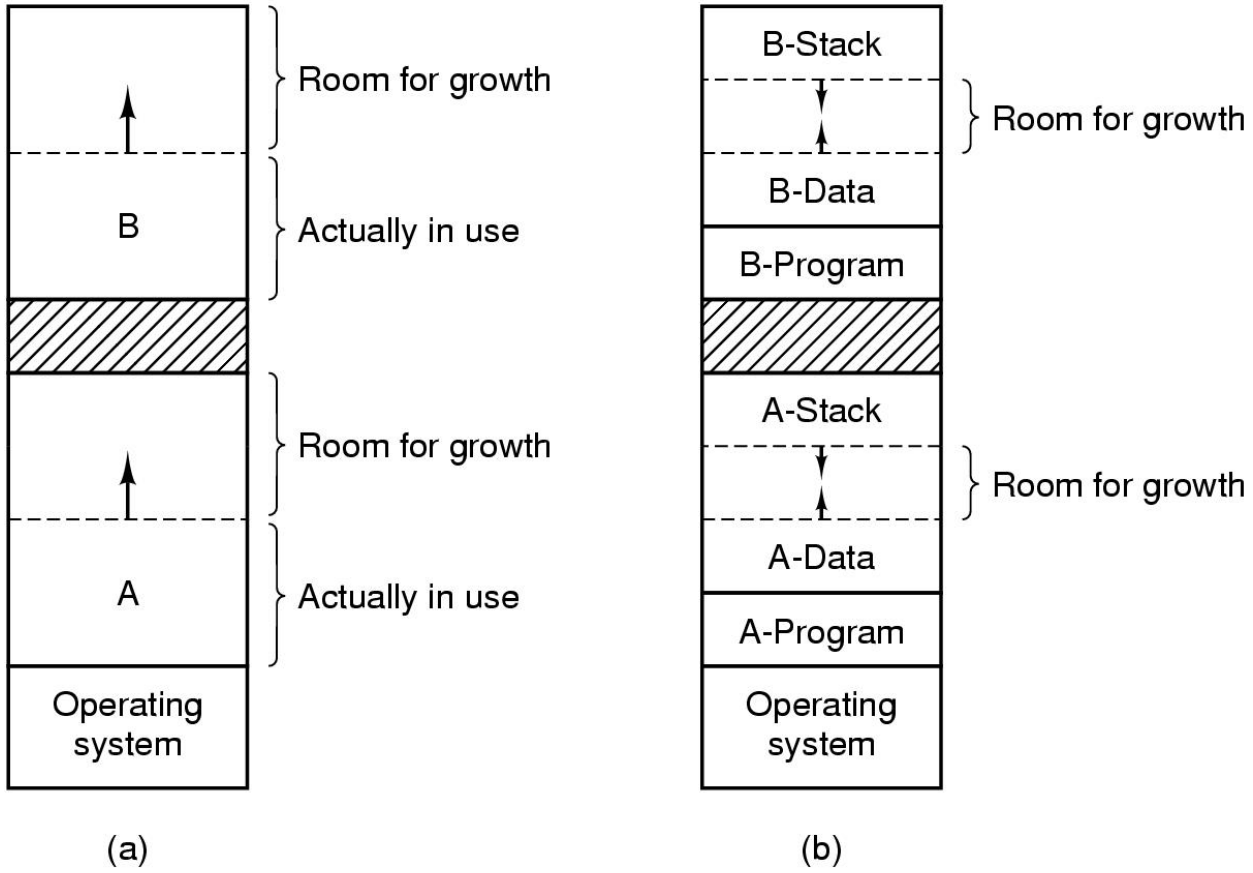
Programs may want to grow during execution

- How much stack memory do we need?
- How much heap memory do we need?

Problem:

- If the partition is too small, programs must be moved
- Requires copying overhead
- Why not make the partitions a little larger than necessary to accommodate “some” cheap growth?
- ... but that is just a different kind of fragmentation

Allocating Extra Space Within



Fragmentation Summary

Memory is divided into partitions

Each partition has a different size

Processes are allocated space and later freed

After a while memory will be full of small holes!

- No free space large enough for a new process even though there is enough free memory in total

If we allow free space within a partition we have fragmentation

External fragmentation = unused space between partitions

Internal fragmentation = unused space within partitions

What Causes These Problems?

Contiguous allocation per process leads to fragmentation, or high compaction costs

Contiguous allocation is necessary if we use a single base register

- ... because it applies the same offset to all memory addresses

Non-Contiguous Allocation

Why not allocate memory in non-contiguous fixed size pages?

- Benefit: no external fragmentation!
- Internal fragmentation < 1 page per process region

How big should the pages be?

- The smaller the better for internal fragmentation
- The larger the better for management overhead (i.e. data structures required to keep track of free pages)

The key challenge for this approach

How can we do secure dynamic address translation?

i.e., how do we keep track of where things are?

Paged Virtual Memory

Memory divided into fixed size page frames

- Page frame size = 2^n bytes
- n low-order bits of address specify byte offset in a page
- remaining bits specify the page number

But how do we associate page frames with processes?

- And how do we map memory addresses within a process to the correct memory byte in a physical page frame?

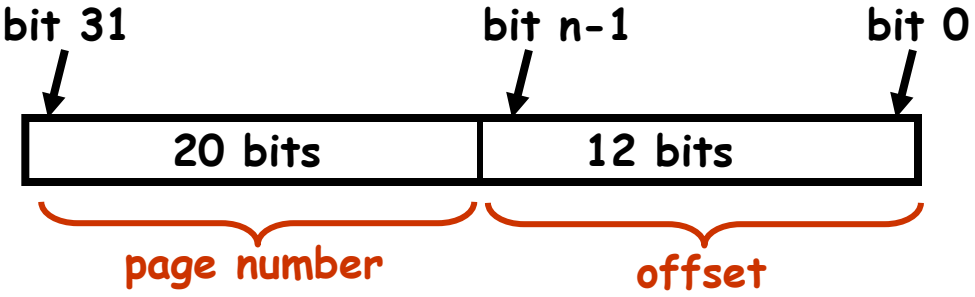
Solution – per-process page table for address translation

- Processes emits virtual addresses
- CPU uses hardware to implement virtual to physical address translation

Virtual Addresses

Virtual memory addresses (what the process uses)

- Page number plus byte offset in page
- Low order n bits are the byte offset
- Remaining high order bits are the page number



Example: 32 bit virtual address

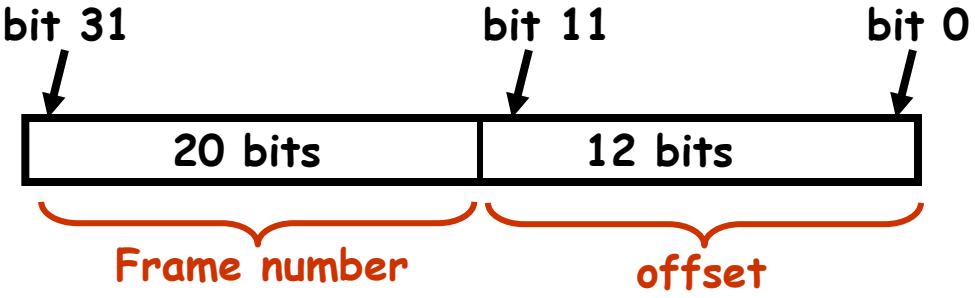
Page size = 2^{12} = 4KB

Address space size = 2^{32} bytes = 4GB

Physical Addresses

Physical memory addresses (what the CPU uses)

- Page frame number plus byte offset in page
- Low order n bits are the byte offset
- Remaining high order bits are the frame number



Example: 32 bit physical address

Frame size = 2^{12} = 4KB

Number of frames = 2^{20} ~ 1M frames

Max physical memory size = 2^{32} bytes = 4GB

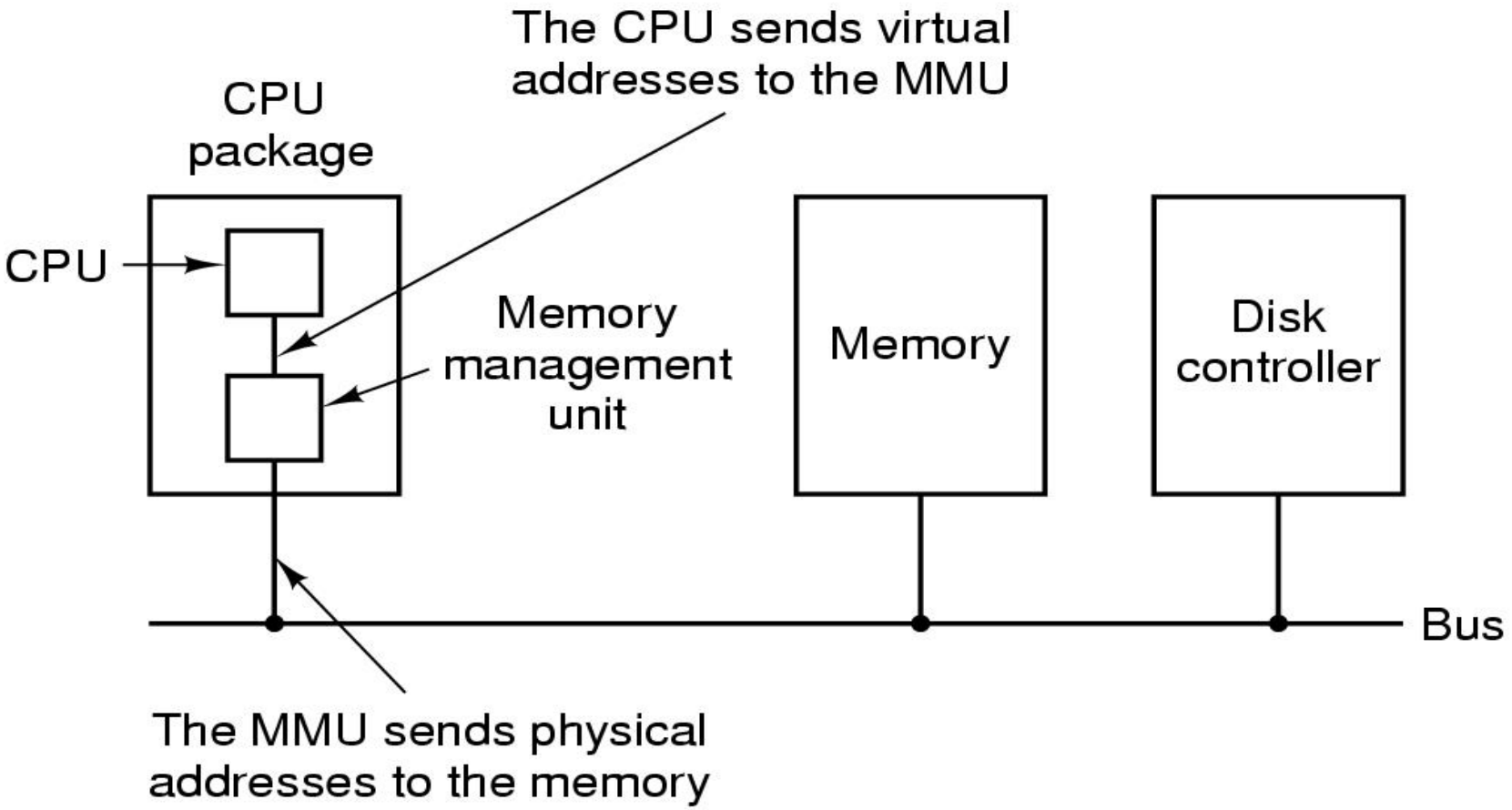
Address Translation

Hardware addresses to frame numbers

Memory management unit (MMU) has multiple offsets for multiple pages, i.e., a page-to-frame lookup table, known as a page table

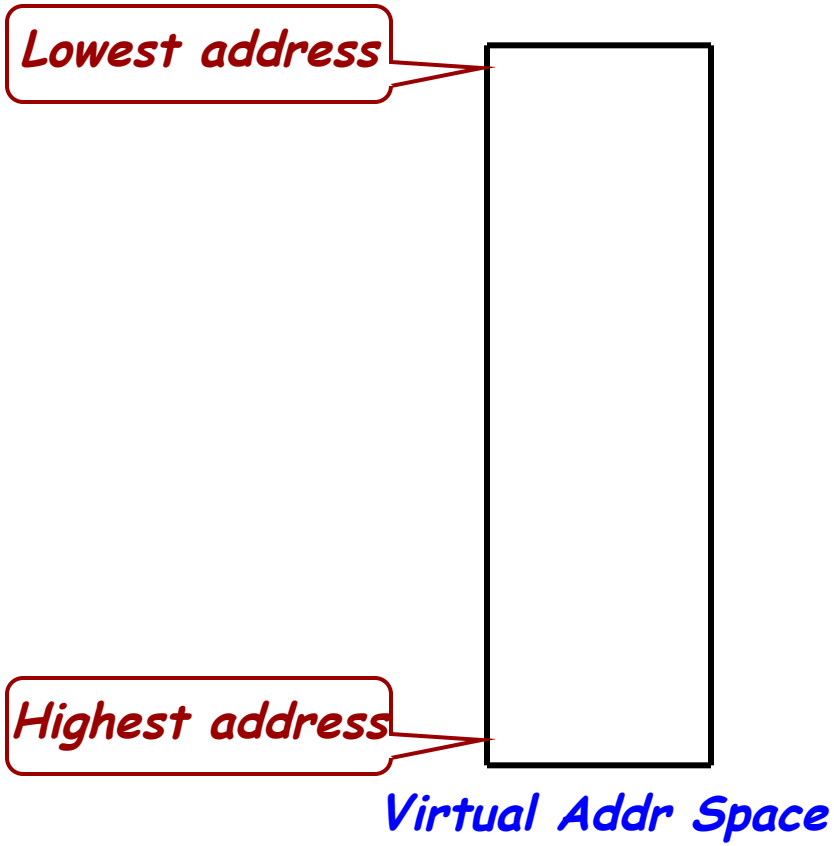
- Like a base register except each entries value is substituted for the page number rather than added to it
- Quiz: Why don't we need a limit register for each page?

MMU


The CPU sends virtual addresses to the MMU
CPU package
CPU
Memory management unit
Memory
Disk controller
Bus
The MMU sends physical addresses to the memory

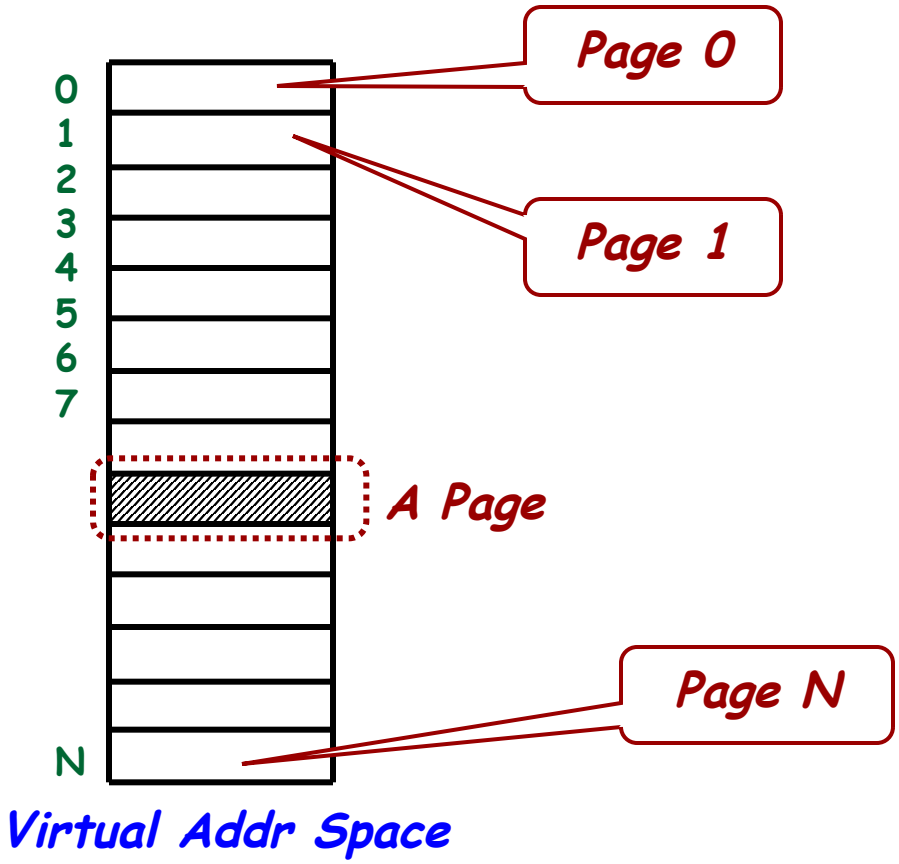
Virtual Address Spaces

Here is the virtual address space (as seen by the process)



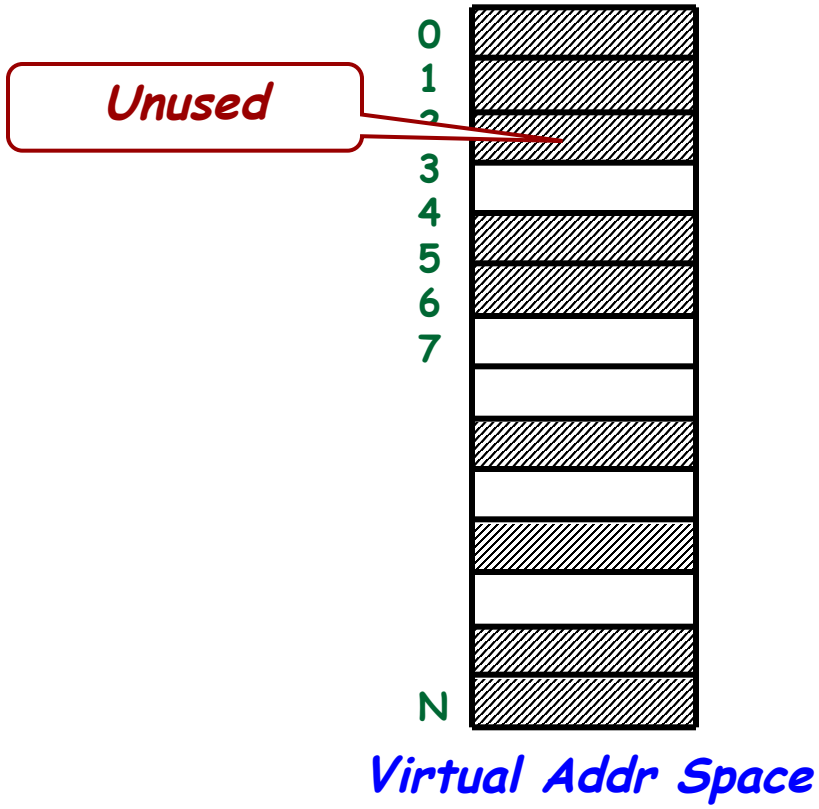
Virtual Address Spaces

The address space is divided into "pages"



Virtual Address Spaces

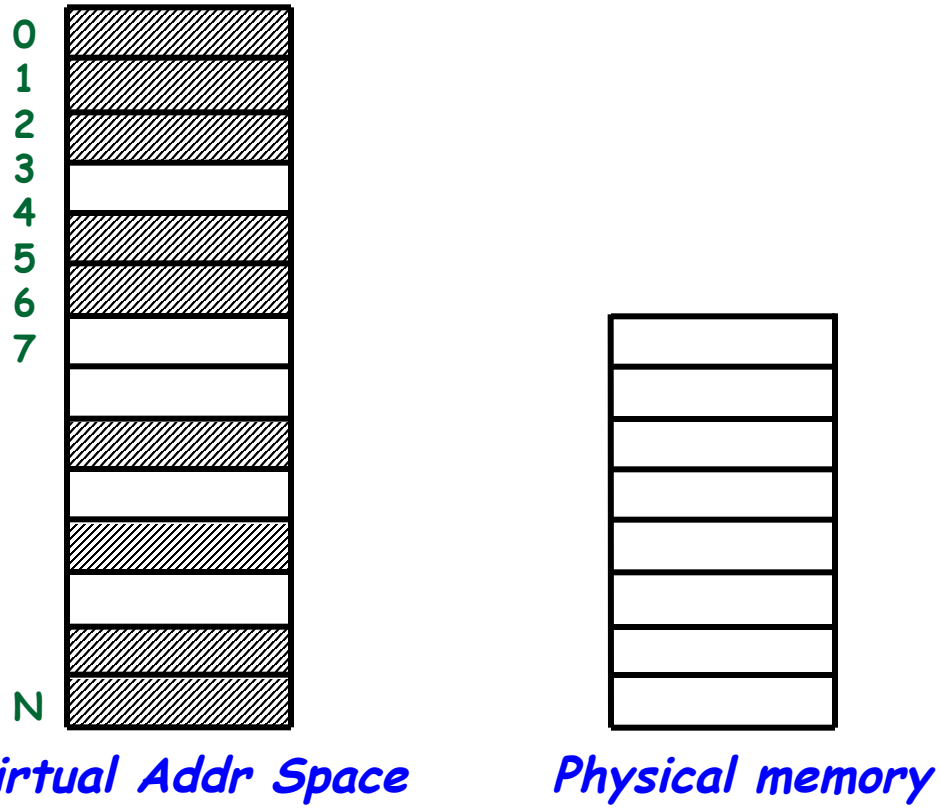
In reality, only some of the pages are used



Physical Memory

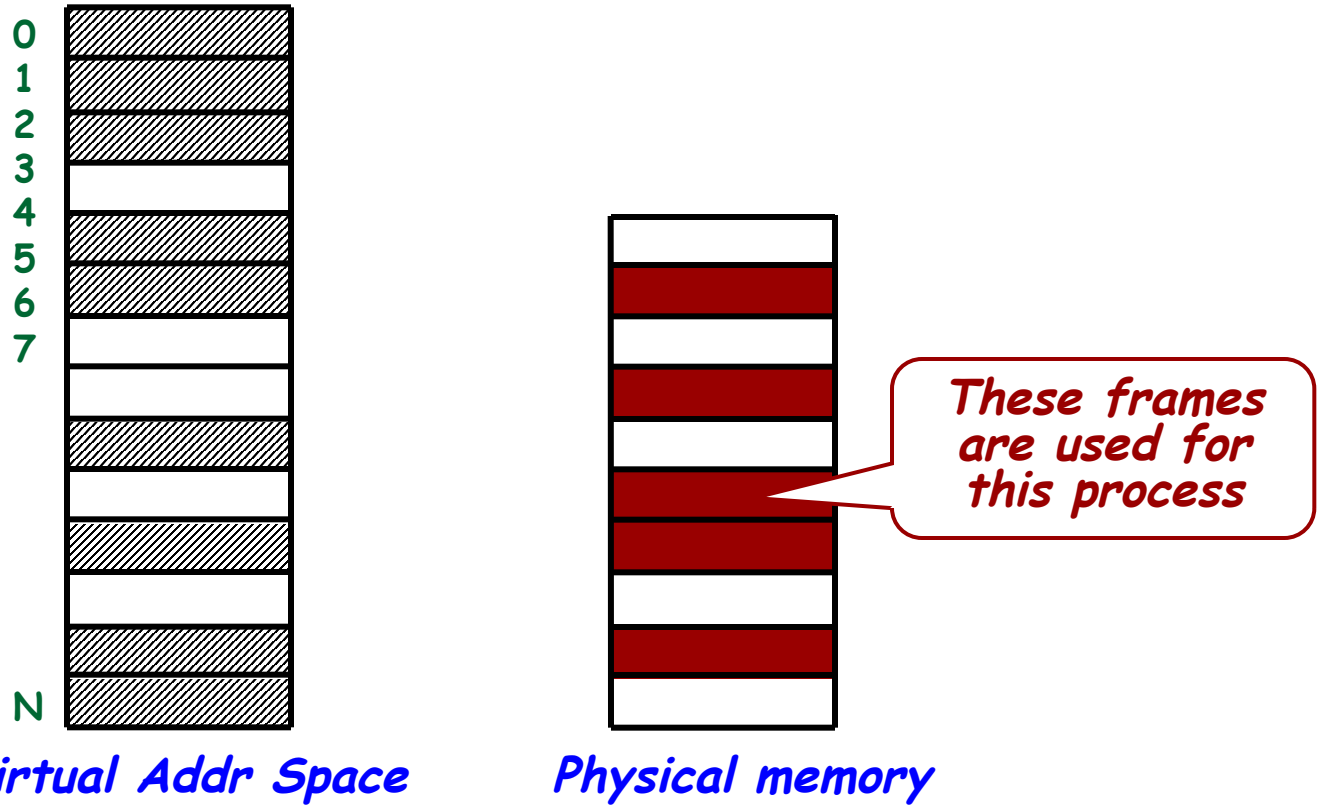
Physical memory is divided into "*page frames*"

(Page size = frame size)



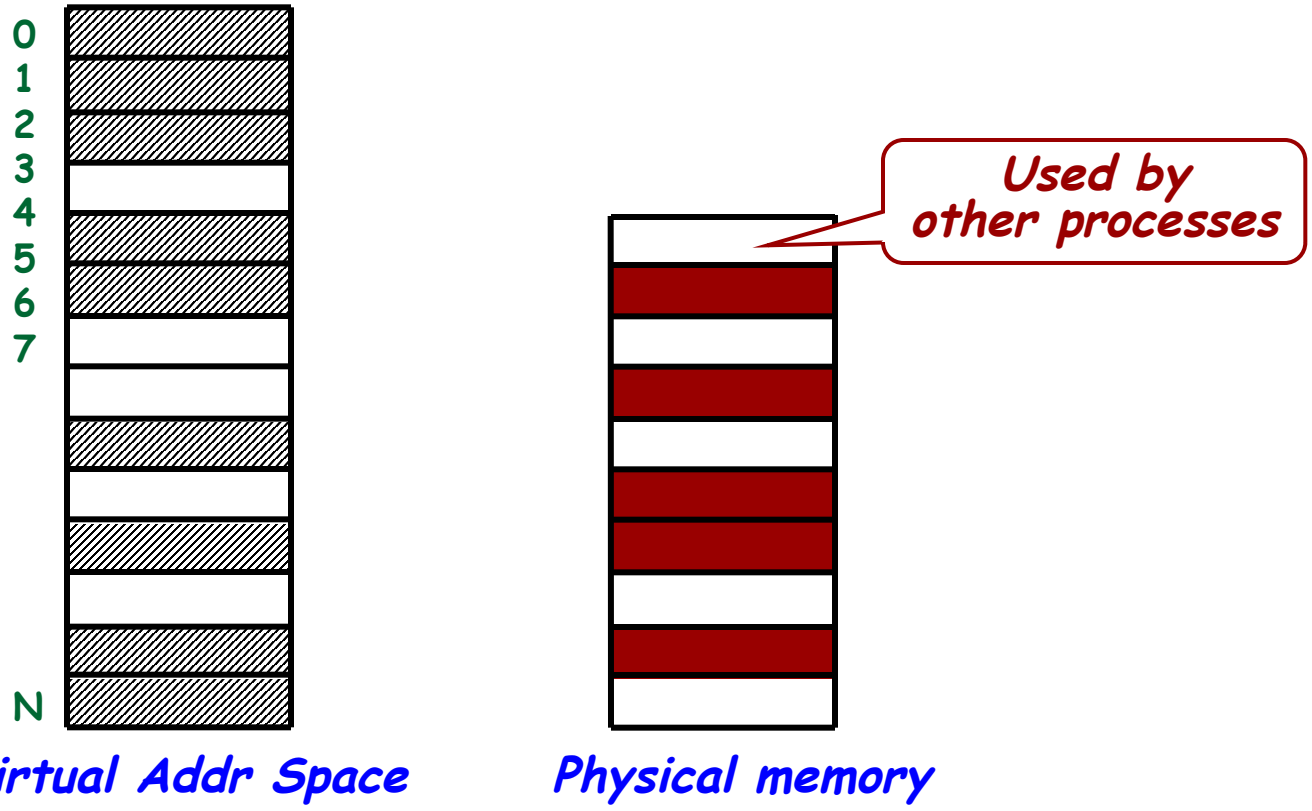
Virtual & Physical Address Spaces

Some frames are used to hold the pages of this process



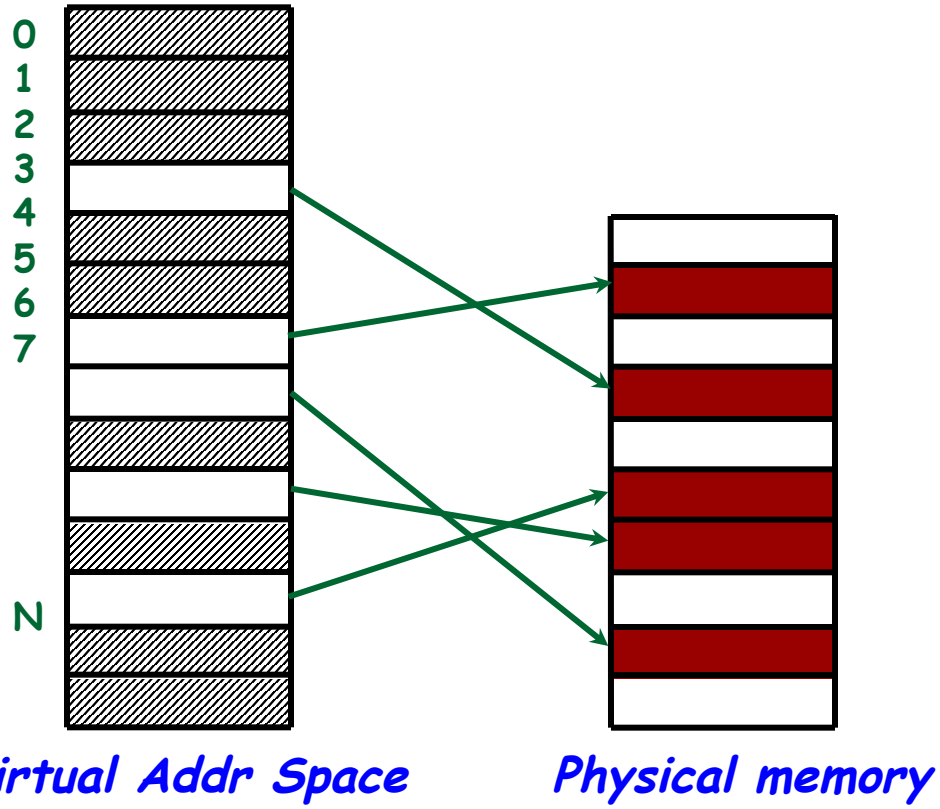
Virtual & Physical Address Spaces

Some frames are used for other processes



Virtual & Physical Address Spaces

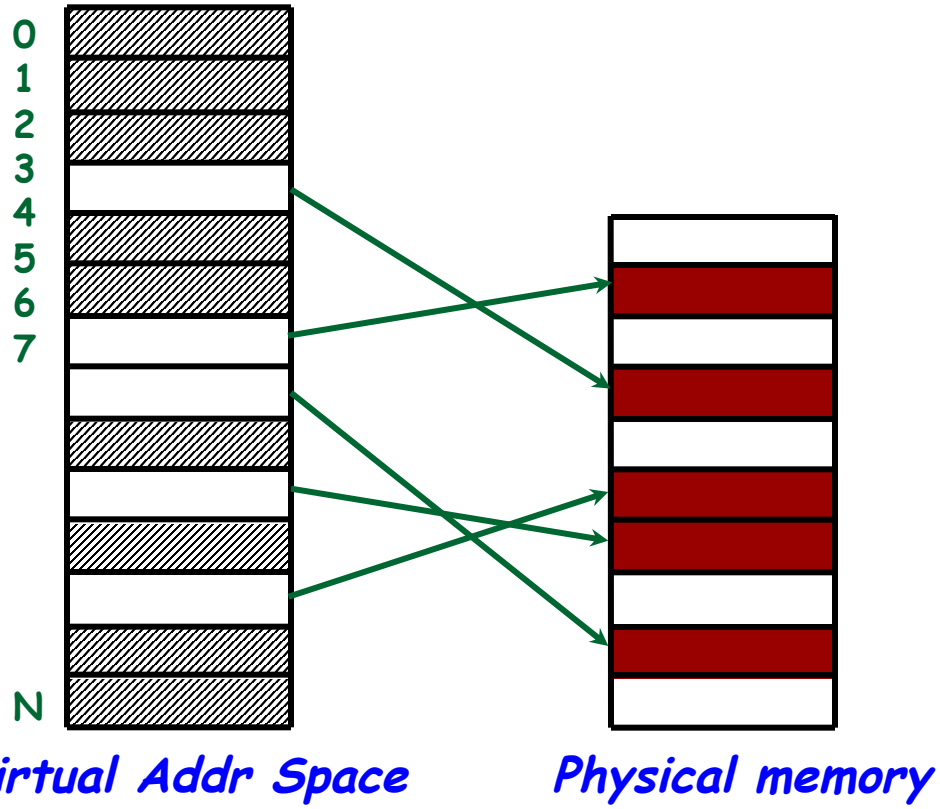
Address mappings say which frame has which page



Page Tables

Address mappings are stored in a *page table* in memory

1 entry/page: is page in memory? If so, which frame is it in?



Key Concept: size

Note that the physical memory could be smaller than the virtual address space or it could be larger.

Note that it is possible to have multiple processes in memory at once, and the total used space might be larger than the size of physical memory.

Address Mappings

Address mappings are stored in a page table in memory

- one page table for each process because each process has its own independent address space

Address translation is done by hardware (ie the TLB ... translation-look-aside buffer)

How does the TLB get the address mappings?

- Either the TLB holds the entire page table (too expensive) or it knows where it is in physical memory and goes there for every translation (too slow)
- Or the TLB holds a portion of the page table and knows how to deal with TLB misses
  - the TLB is a cache of page table entries

Process Context Switch

Every time OS switches processes it must point the MMU at a different, process-specific page table.

And OS must flush the TLB OR the TLB must have extra hardware bits indicating which process's translations are cached in each TLB entry (common)

NOTE: TLBs are often fully associative (so no conflict misses)

Two Types of TLB

What if the TLB needs a mapping it doesn't have?

Software managed TLB

- It generates a TLB-miss fault which is handled by the operating system (like interrupt or trap handling)
- The operating system looks in the page tables, gets the mapping from the right entry, and puts it in the TLB, perhaps replacing an existing entry

Hardware managed TLB (Intel)

- It looks in a pre-specified physical memory location for the appropriate entry in the page table
- The hardware architecture defines where page tables must be stored in physical memory
- OS loads current process page table there on context switch!

Quiz

What is the difference between a virtual and a physical address?

What is address binding?

Why are programs not usually written using physical addresses?

Why is hardware support required for dynamic address translation?

What is a page table used for?

What is a TLB used for?

How many address bits are used for the page offset in a system with 8KB page size?