Today

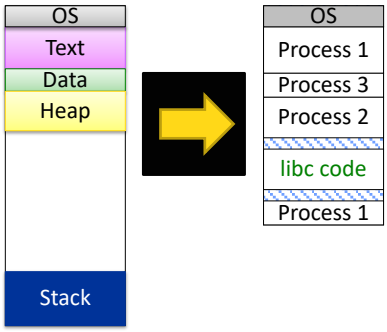
- Memory Management
  - Virtual Memory
  - Segmentation, Paging
- Project 5

Review

- What abstraction does virtual memory provide?
- What requirements do we have for the VM, from the various stakeholders, so far?

Aaron Bauer, University of Washington
"Understanding Human Problem Solving in Complex Digital Environments"
Talk at 4 p.m. (Reception at 3:45 p.m.)

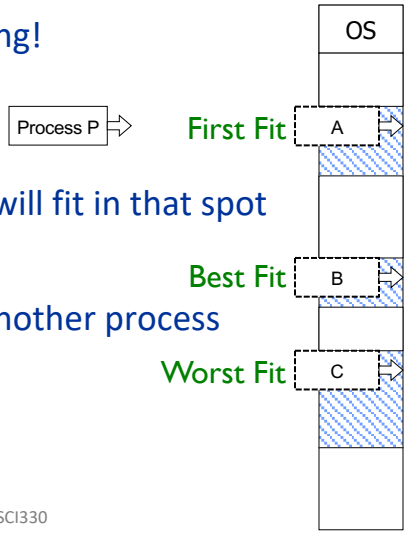
Address Translation: Wish List



- Map virtual addresses to physical addresses
- Allow multiple processes to be in memory at once, but isolate them from each other
- Determine which subset of data to keep in memory/move to disk
- Allow the same physical memory to be mapped in multiple process VASes

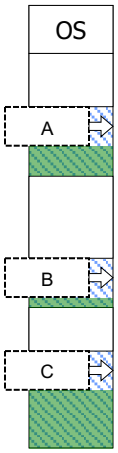
Review:
Where should process P be placed?

- First fit
  - Don't spend time searching!
- Best fit
  - It fits tightly!
  - Maybe no other process will fit in that spot
- Worst fit
  - Leaves lots of space for another process



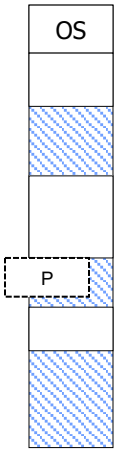
(External) Fragmentation

- No matter where it ends up, the remaining gaps get smaller
- Large gaps are probably still usable, small ones likely aren't
- Fragmentation: over time, we end up with small gaps that become more difficult to use (eventually, wasted)
- "External" because the gaps are *between allocated* pieces



(External) Fragmentation

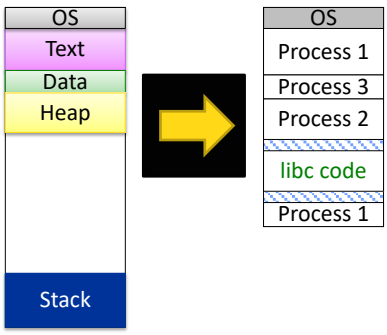
- Suppose we put it here, and later, P asks for more memory?
- What if there isn't enough space…
  - Move P?
  - Move everybody to compact the address space?
- This seems bad. Lots of tough problems (placement, fragmentation) with no clear solutions.



Alternative Organization of PAS

- Divide PAS into *fixed*-size pieces
- Use memory translation to assign virtual addresses to physical locations
- Every physical location is an equally good choice!

Address Translation: Wish List

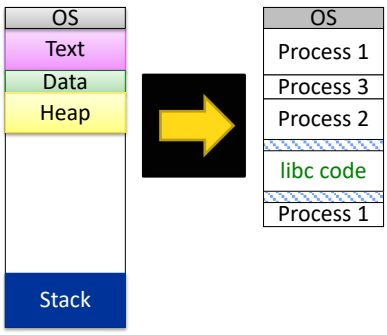


- Map virtual addresses to physical addresses
- Allow multiple processes to be in memory at once, but isolate them from each other
- Determine which subset of data to keep in memory/move to disk
- Allow the same physical memory to be mapped in multiple process VASes
- **Make it easier to perform placement in a way that reduces fragmentation**

OS Perspective

- Primary challenge: Which physical memory do we give to processes?
- Other important considerations:
  - **Protection**: OS is resource gatekeeper, must isolate itself (and processes)
  - **Performance**: OS should map memory for best performance, as long as it doesn't violate protection

Address Translation: Wish List

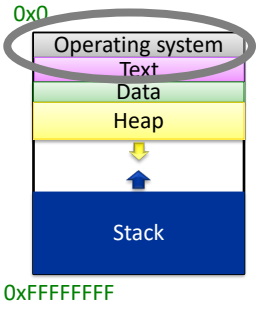


- Map virtual addresses to physical addresses
- **Allow multiple processes to be in memory at once, but isolate them from each other**
- Determine which subset of data to keep in memory/move to disk
- **Allow the same physical memory to be mapped in multiple process VASes**
- Make it easier to perform placement in a way that reduces fragmentation

- **Protection**: OS is resource gatekeeper, must isolate itself (and processes)
- **Performance**: OS should map memory for best performance, as long as it doesn't violate protection

Recall: Context Switching Performance

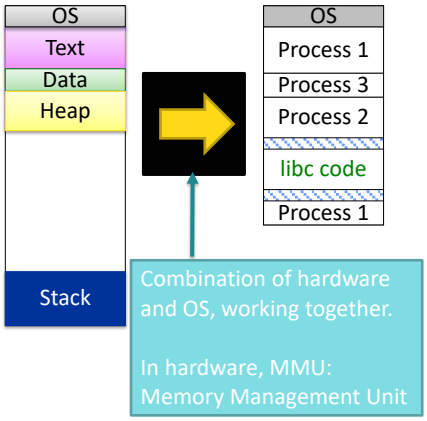
- Even though it's fast, context switching is expensive:
  1. time spent is 100% overhead
  2. must invalidate other processes' resources (caches, memory mappings)
  3. kernel must execute – it must be accessible in memory
- Solution to #3:
  - keep kernel mapped in every process VAS
  - protect it to be inaccessible



Hardware

- Hardware and OS are symbiotic, often influence each other
  - Example: atomic instructions
- Memory management is another important area of collaboration
- Hardware goals:
  - Make translation fast
  - Give OS storage for and control over mappings

Address Translation: Wish List



- Map virtual addresses to physical addresses
- Allow multiple processes to be in memory at once, but isolate them from each other
- Determine which subset of data to keep in memory/move to disk
- Allow the same physical memory to be mapped in multiple process VASes
- Make it easier to perform placement in a way that reduces fragmentation
- **Map addresses quickly with a little HW help**

How does each benefit from having a logical memory abstraction?

- The user
  - Memory size, protection
- The programmer
  - Shared libraries
- The compiler
  - Placement of data
- The OS / OS designer
  - Memory placement, fragmentation
- The hardware / hardware designer
  - Just makes it more complex BUT can help OS

Awkward transition

PROJECT 5

Project 5

- Combines processes and memory management
- proc.c
  - Implementations of process management and memory management

proc.h Data Structures: memoryMap

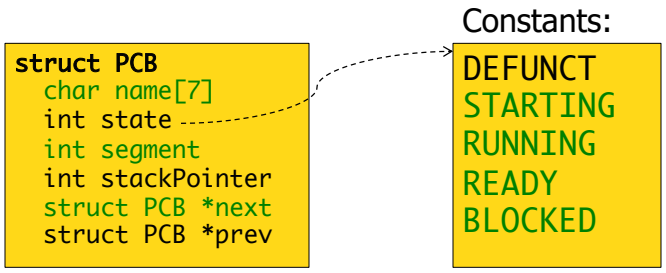
- Allow one process to be loaded into each segment
  - Valid Segments: 0x2000, … 0x9000
  - (0x0000 reserved for interrupt vector, 0x1000 reserved for kernel)
- Memory segment map:
  - Each index corresponds to one memory segment.
    - segment = (index+2)*0x1000
    - index = (segment/0x1000)-2
  - Marked as:
    - FREE or USED

memoryMap

0	USED
1	USED
2	FREE
3	FREE
4	USED
5	FREE
6	FREE
7	FREE

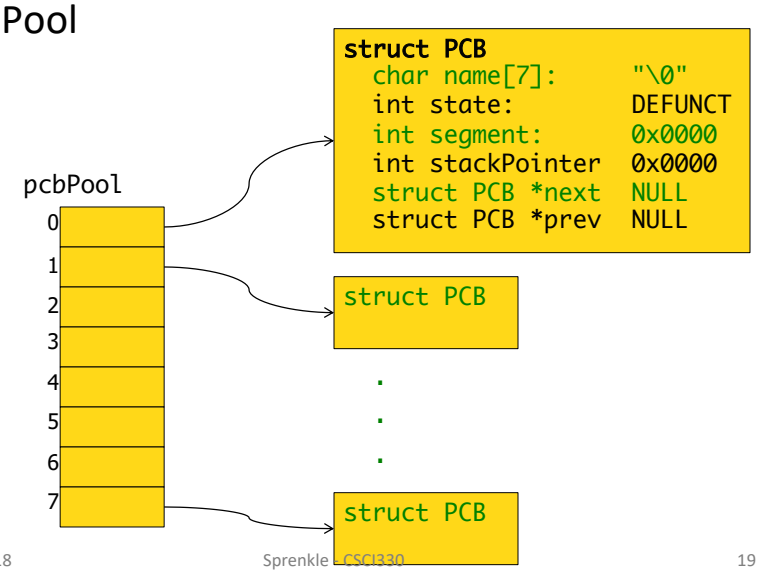
proc.h Data Structures: PCB

- Process Control Block:



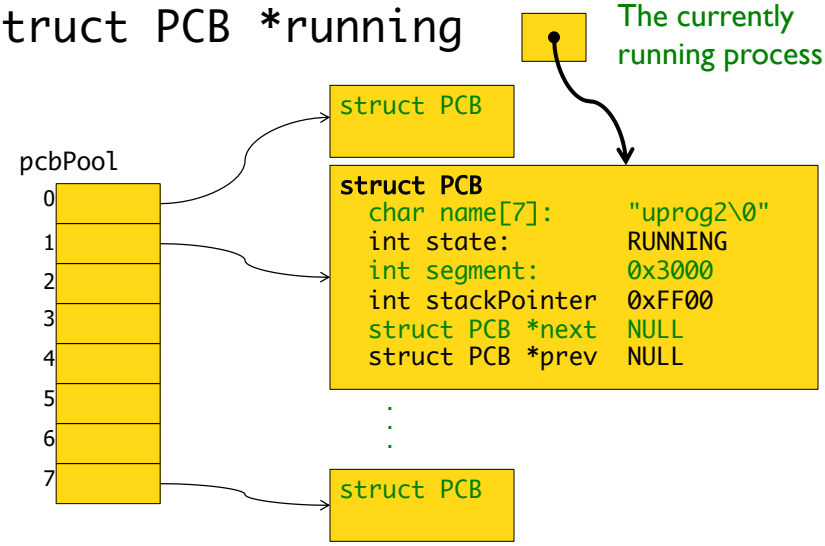
proc.h Data Structures: pcbPool

- PCB Pool



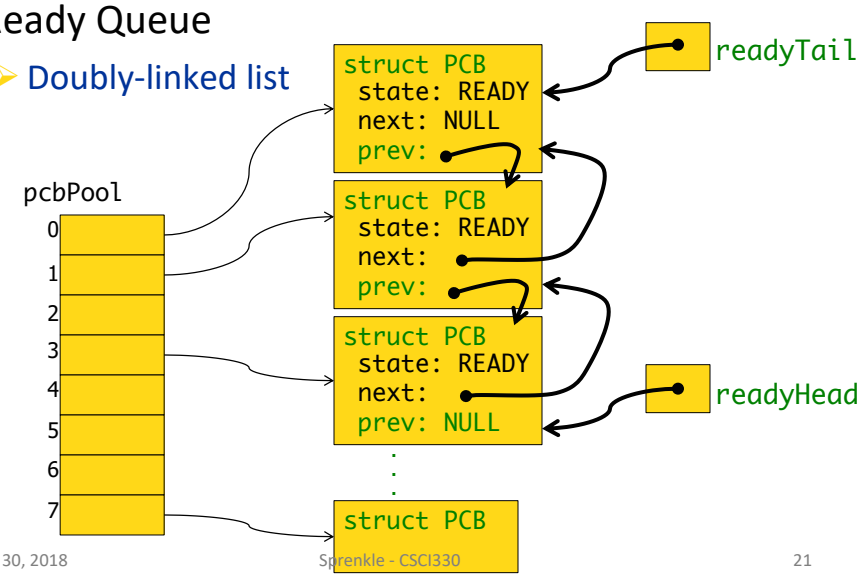
proc.h Data Structures

- struct PCB *running



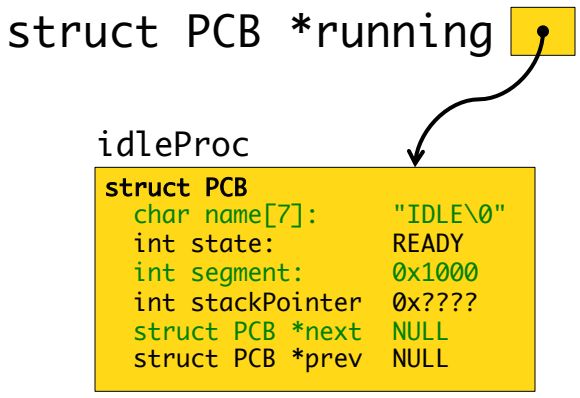
proc.h Data Structures

- Ready Queue
  - Doubly-linked list



proc.h Data Structures: running

- Initially the running process will be the **idle** process

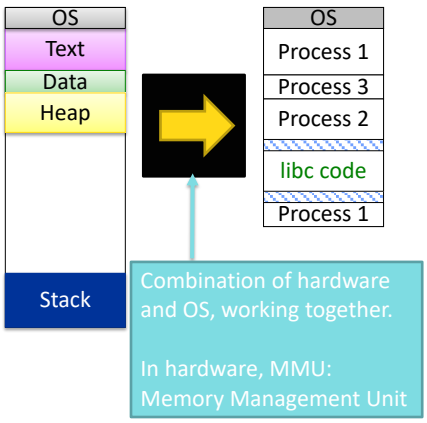


If no processes in the ready queue, run idle process

Equally awkward translation back

VM→PHYSICAL ADDRESS TRANSLATION

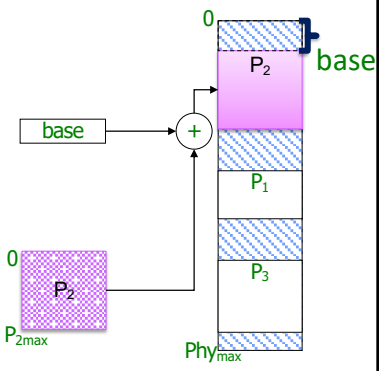
Address Translation: Wish List



- Map virtual addresses to physical addresses
- Allow multiple processes to be in memory at once, but isolate them from each other
- Determine which subset of data to keep in memory/move to disk
- Allow the same physical memory to be mapped in multiple process VASes
- Make it easier to perform placement in a way that reduces fragmentation
- Map addresses quickly with a little HW help

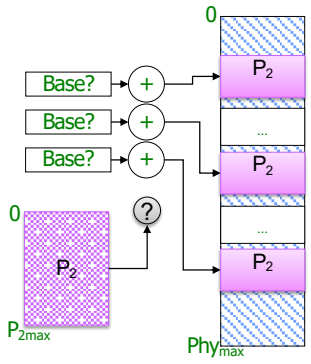
Simple (Unrealistic) Translation Example

- Process P_2's virtual addresses don't align with physical memory's addresses
- Consider: P_2 wants to access address 0x1000
- Determine offset from *physical address* 0 to start of P_2
  - store in *base*



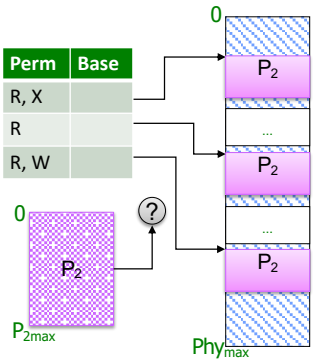
Generalizing

- Problem: process may not fit in one contiguous region



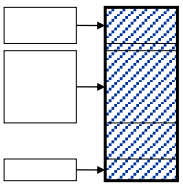
Generalizing

- Problem: process may not fit in one contiguous region
- Solution: keep a table (one per process)
  - Keep details for each region in a row
  - Store additional metadata (ex. permissions)
- Questions:
  - How many regions should there be (and what size)?
  - How to determine which table entry we should use?

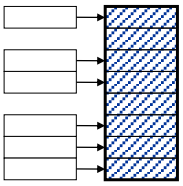


Defining Regions - Two Approaches

- Segmentation:
  - Partition address space and memory into logical segments
  - Segments have *varying* sizes

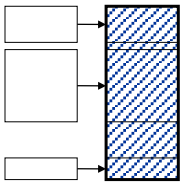


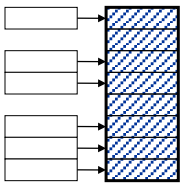
- Paging:
  - Partition address space and memory into pages
  - Pages are a constant, *fixed* size



Why would you use each approach? Pros/Cons

- Segmentation:
  - Partition address space and memory into segments
  - Segments have *varying* sizes
- Paging:
  - Partition address space and memory into pages
  - Pages are a constant, *fixed* size



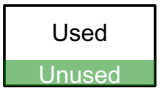


Consider how memory would be requested and used
Do the pros/cons change based on whose perspective?

Fragmentation

Internal

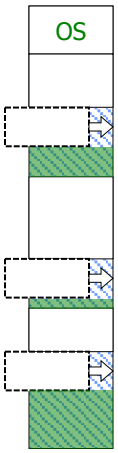
- Process asks for memory, doesn't use it all
- Possible reasons:
  - Process was wrong about needs
  - OS gave it more than it asked for
- ***internal***: within an allocation



Memory allocated to process

External

- Over time, we end up with small gaps that become more difficult to use
  - eventually, wasted
- ***external***: unused memory between allocations



Which scheme is better for *reducing* internal and external fragmentation?

A. Segmentation is better than paging for both forms of fragmentation.

B. Segmentation is better for *internal* fragmentation, while paging is better for *external* fragmentation.

C. Paging is better for *internal* fragmentation, while segmentation is better for *external* fragmentation.

D. Paging is better than segmentation for both forms of fragmentation.

Which scheme is better for reducing internal and external fragmentation?

A. Segmentation is better than paging for both forms of fragmentation.

B. Segmentation is better for *internal* fragmentation, while paging is better for *external* fragmentation.

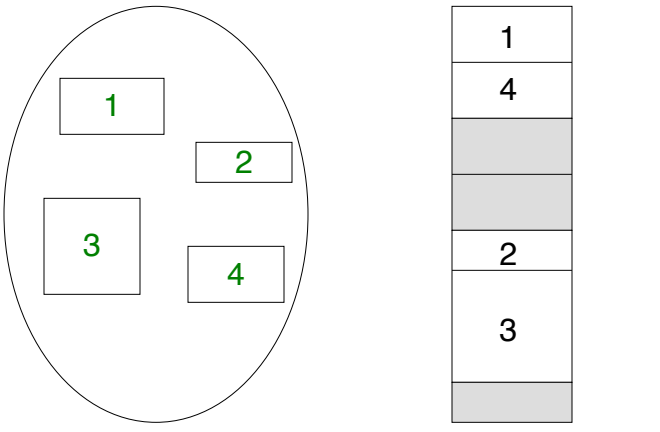
C. Paging is better for *internal* fragmentation, while segmentation is better for *external* fragmentation.

D. Paging is better than segmentation for both forms of fragmentation.

Segmentation vs. Paging

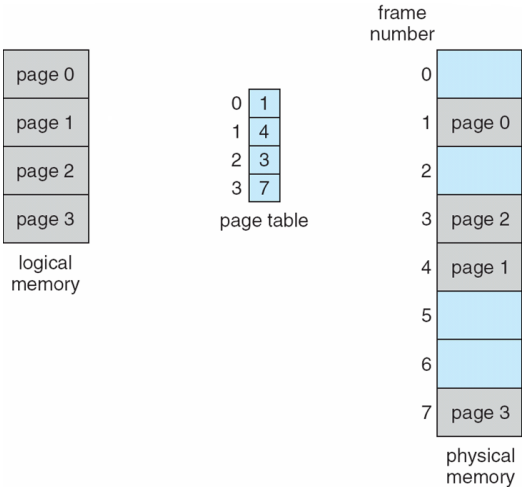
- A ***segment*** is good ***logical*** unit of information
  - Can be sized to fit any contents
  - Easy to share large regions (e.g., code, data)
  - Protection requirements correspond to logical data segment
- A ***page*** is good ***physical*** unit of information
  - Simple physical memory placement
  - No external fragmentation
  - Constant sizes make it easier for hardware to help

Logical View of Segmentation



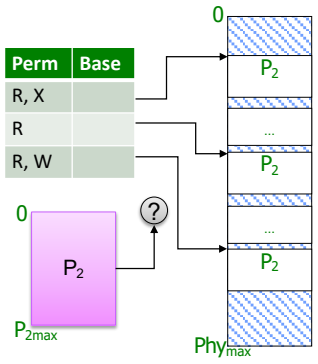
Since segments vary in length,
memory allocation is a dynamic storage allocation problem

Paging Model of Logical and Physical Memory



Generalizing

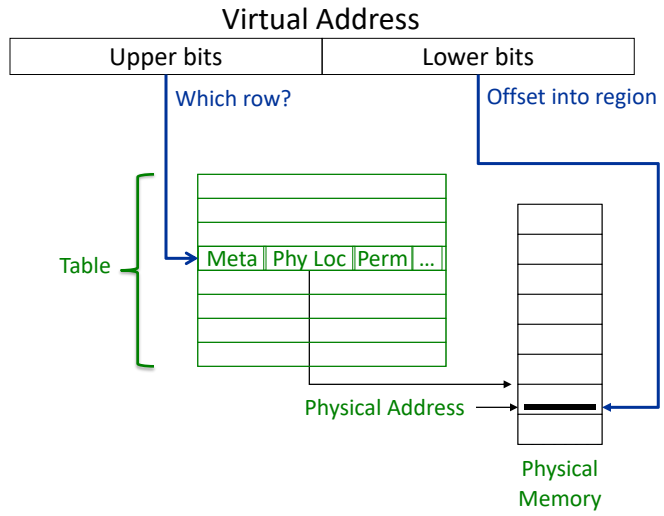
- Problem: process may not fit in one contiguous region
- Solution: keep a table (one per process)
  - Keep details for each region in a row
  - Store additional metadata (ex. permissions)
- Interesting questions:
  - How many regions should there be (and what size)?
  - **How to determine which table entry we should use?**



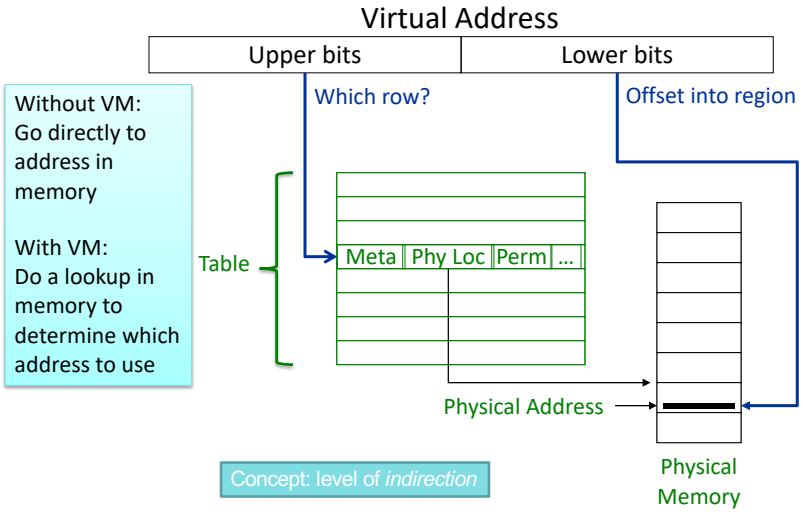
For **Both** Segmentation and Paging…

- Each process has a table to track memory address translations
- When a process attempts to read/write to memory:
  - use *high order* bits of virtual address to determine which row to look at in the table
  - use *low order* bits of virtual address to determine an offset within the physical region

Address Translation

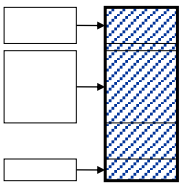


Performance Implications

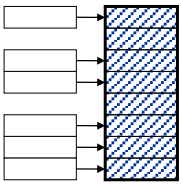


Defining Regions - Two Approaches

- Segmentation:
  - Partition address space and memory into logical segments
  - Segments have *varying* sizes

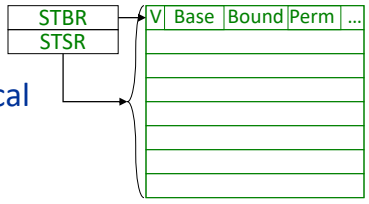


- Paging:
  - Partition address space and memory into pages
  - Pages are a constant, *fixed* size

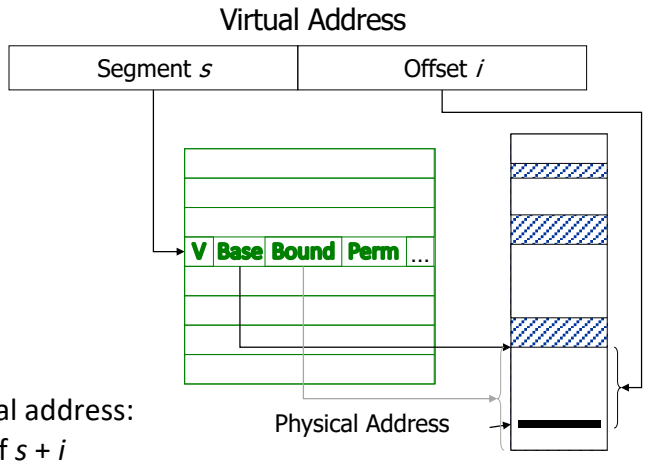


Segment Table

- One table per process
- Where is the *table* located in memory?
  - Segment table base register (STBR)
  - Segment table size register (STSR)
- Table entries: Segment metadata
  - V: valid bit
    - does it contain a mapping?
  - Base: segment location in physical memory
  - Bound: segment size in physical memory
  - Permissions

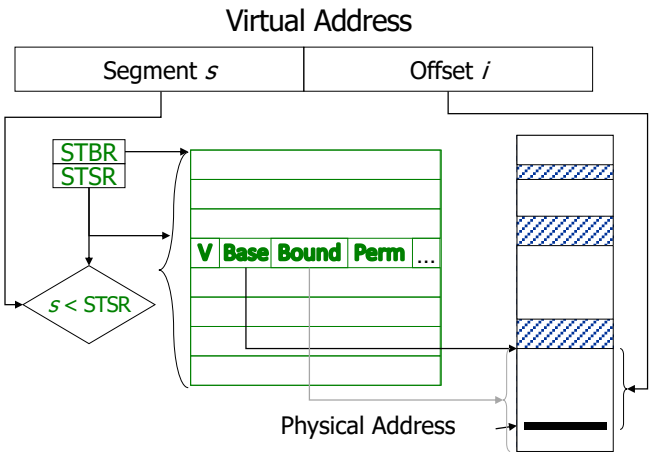


Segment Address Translation

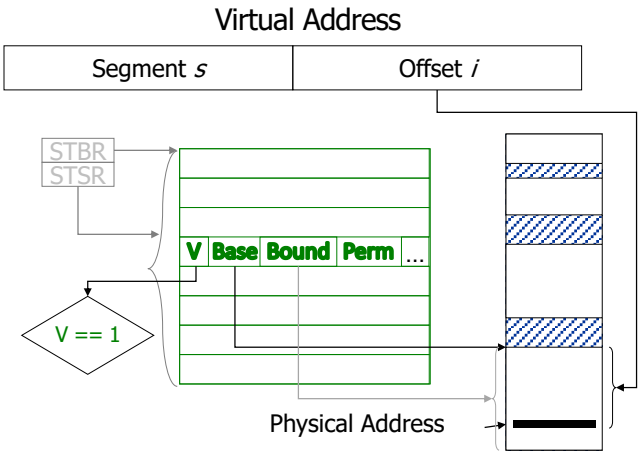


- Physical address: base of *s* + *i*

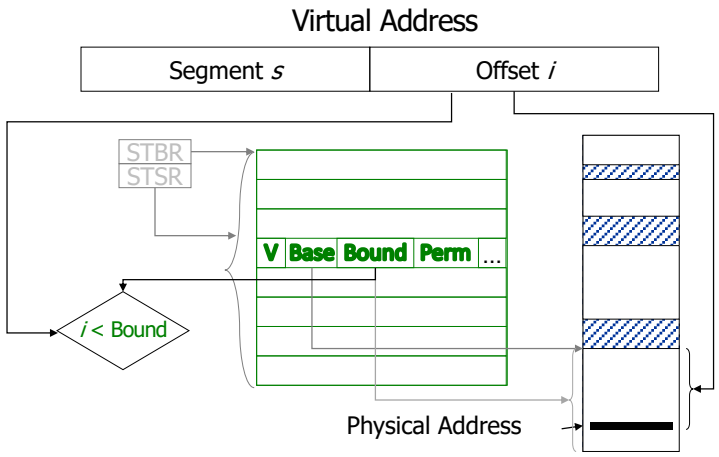
Check if Segment *s* is within Range



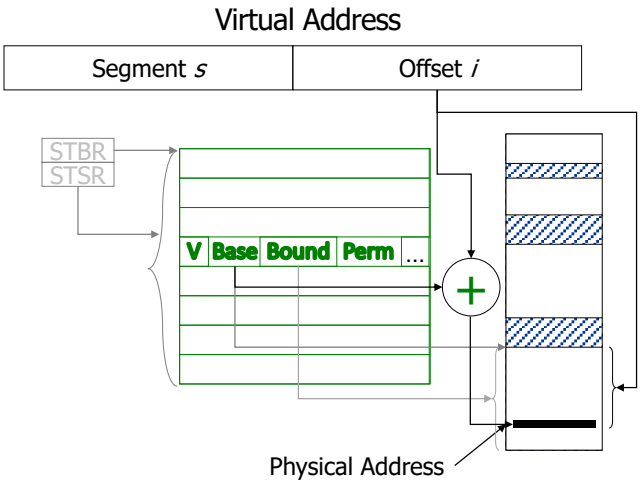
Check if Segment Entry *s* is Valid



Check if Offset *i* is within Bounds



Translate Address



Pros and Cons of Segmentation

Pros

- Each segment can be
  - located independently
  - separately protected
  - grown/shrunk independently
- Small segment table size
  - ~256 Bytes → 1GB memory

Cons

- Variable-size allocation
  - Difficult to find holes in physical memory
  - External fragmentation

Looking Ahead

- Project 5 due next Friday