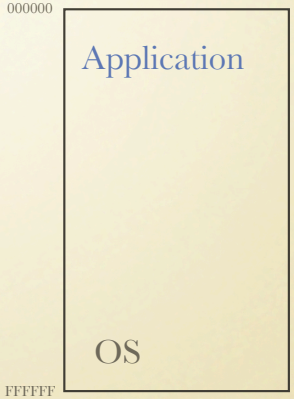
Memory Management

The Virtual Memory Abstraction

Physical Memory	Virtual Memory
Unprotected address space	Programs are isolated
Limited size	Arbitrary size
Shared physical frames	All programs loaded at "0"
Easy to share data	Sharing is possible

Uniprogramming without Protection

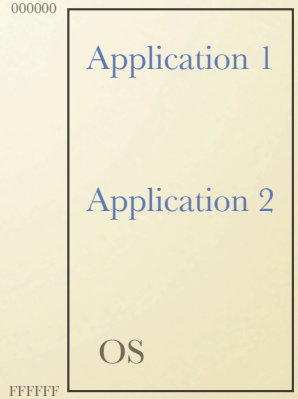
- Old PC OSes
- Only one application at a time
- Trivially achieves illusion of dedicate machine..



Multiprogramming without Protection

Linker-loader

- change addresses of loads, stores jump to refer to where the program lands in memory
  - multiple programs in memory
  - no hardware support
  - no protection
    - applications can trump each other
    - applications can trump OS



Implementing protection

- Use hardware support
- Two key ideas:
  - Address Translation
    - Physical vs. Virtual address spaces
  - Dual Mode Operation
    - Kernel mode vs. User mode

Address spaces: Physical and Virtual

- Physical address space consists of the collection of memory addresses supported by the hardware
- Virtual address space consists of the collection of addresses that the process can "touch"
- Note: CPU generates virtual addresses

Address Translation

- A function that maps $\langle pid, virtual\ address \rangle$ into $physical\ address$

Protection

- At all times, the functions used by different processes map to disjoint ranges

Relocation

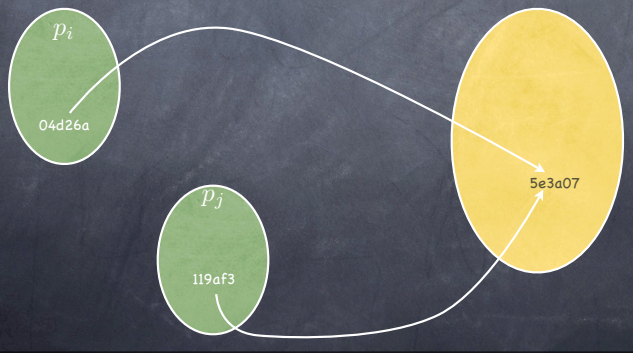
- The range of the function used by a process can change over time

Relocation

- The range of the function used by a process can change over time

Data Sharing

- Map different virtual addresses of different processes to the same physical address



Multiplexing

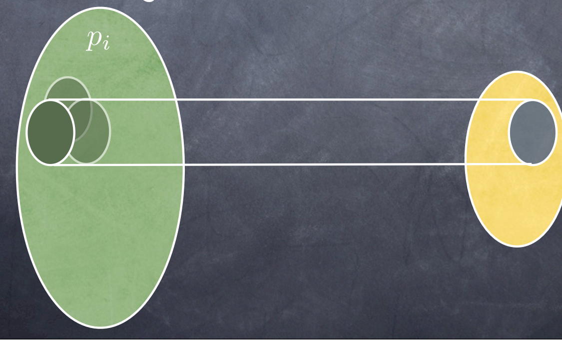
- The domain (set of virtual addresses) that map to a given range of physical addresses can change over time

Multiplexing

- The domain (set of virtual addresses) that map to a given range of physical addresses can change over time

Multiplexing

- The domain (set of virtual addresses) that map to a given range of physical addresses can change over time



Multiplexing

- The domain (set of virtual addresses) that map to a given range of physical addresses can change over time

Multiplexing

- The domain (set of virtual addresses) that map to a given range of physical addresses can change over time

One idea, many implementations

- base & limit
- segment table
- page table
- paged segmentation
- multi-level page table
- inverted page table

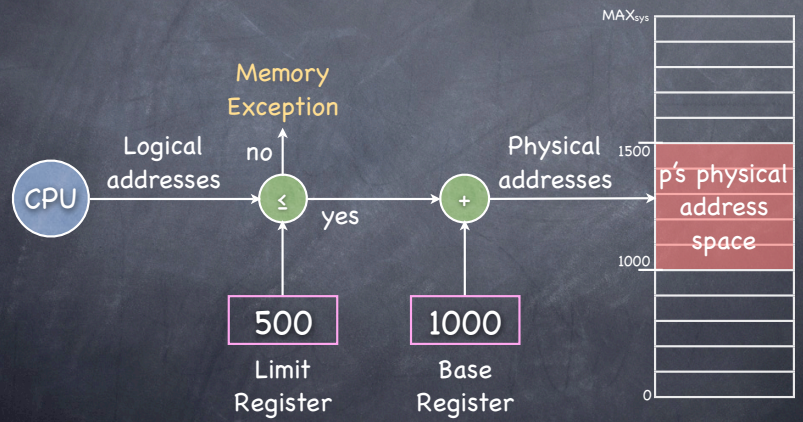
One idea, many implementations

- base & limit
- segment table
- page table
- paged segmentation
- multi-level page table
- inverted page table

It's all just a lookup...

Virtual Address	Physical Address
000000	a30940
000001	56bb03
000010	240421
ffffff	d82a04

Base & Limit

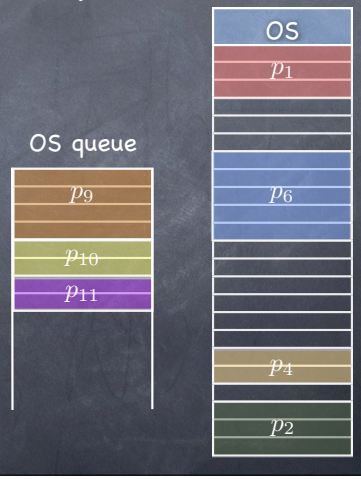


On Base & Limit

- Contiguous Allocation: contiguous virtual addresses are mapped to contiguous physical addresses
- Protection is easy, but sharing is hard
  - Two copies of emacs: want to share code, but have data and stack distinct...
- Managing heap and stack dynamically is hard
  - We want them as far as as possible in virtual address space, but...

Contiguous allocation: multiple variable partitions

- OS keeps track of empty blocks ("holes")
- Initially, one big hole!
- Over time, a queue of processes (with their memory requirements) and a list of holes
- OS decides which process to load in memory next
- Once process is done, it releases memory



Strategies for Contiguous Memory Allocation

- First Fit
  - Allocate first big-enough hole
- Best Fit
  - Allocate smallest big-enough hole
- Worst Fit
  - Allocate largest big-enough hole

Strategies for Contiguous Memory Allocation

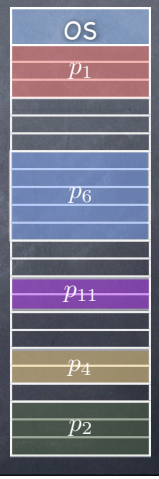
- ✓ First Fit
  - Allocate first big-enough hole
- ✓ Best Fit
  - Allocate smallest big-enough hole
- Worst Fit
  - Allocate largest big-enough hole

Fragmentation

- External fragmentation
  - Unusable memory between units of allocation

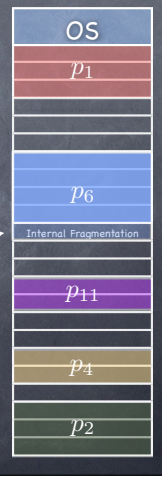
Fragmentation

- External fragmentation
  - Unusable memory between units of allocation
- Internal fragmentation
  - Unusable memory within a unit of allocation



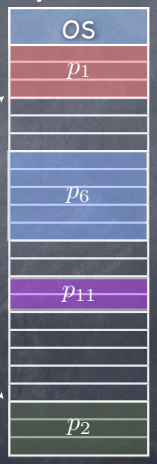
Fragmentation

- External fragmentation
  - Unusable memory between units of allocation
- Internal fragmentation
  - Unusable memory within a unit of allocation



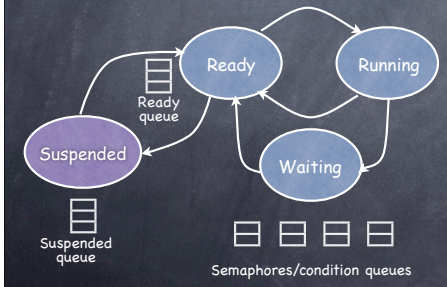
Eliminating External Fragmentation: Compaction

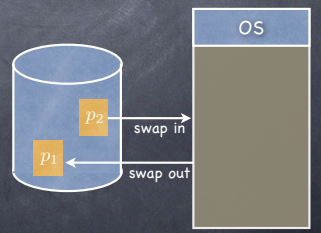
- Relocate programs to coalesce holes
- Problem with I/O
  - Pin job in memory while it is performing I/O
  - Do I/O in OS buffers



Eliminating External Fragmentation: Swapping

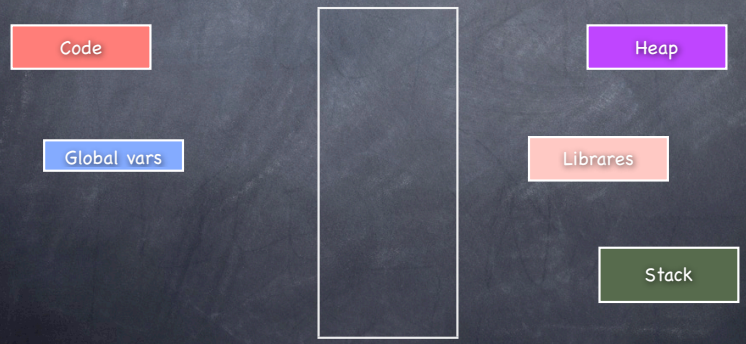
- Preempt processes and reclaim their memory
- Move images of suspended processes to backing store





E Pluribus Unum

- From a user's perspective, a process is a collection of distinct logical address spaces



E Pluribus Unum

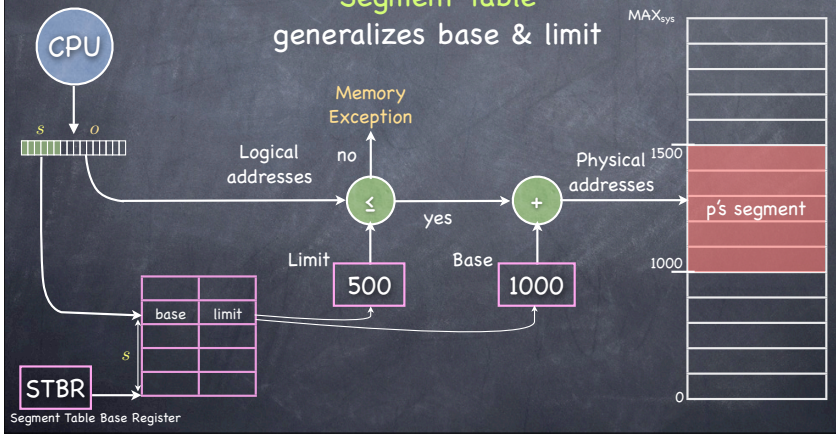
- From a user's perspective, a process is a collection of distinct logical address spaces

We call these logical address spaces segments

- Contiguous mapping of addresses within segment
- Holes in Virtual address space: a problem?
- Think of address as (s, o)
  - s is the segment number
  - o is the offset within the segment

Implementing Segmentation

Segment table generalizes base & limit



On Segmentation

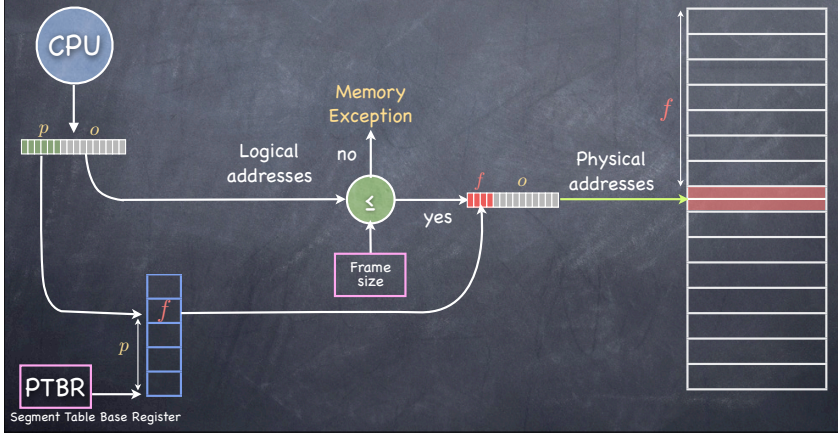
- Sharing a segment is easy!
- Protection bits control access to shared segments
- External fragmentation...
- Each process maintains a segment table, which is saved to PCB on a context switch
- Fast?
- Part of a segment in memory?
- How do we enlarge a segment?

base	limit
400	600
2900	200
2500	200
3200	500
1300	1000

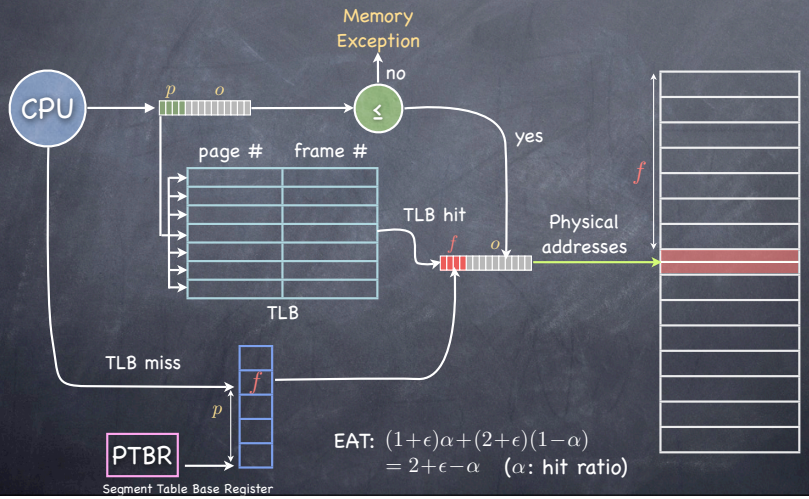
How to avoid external fragmentation?

- Allocate memory in fixed-sized chunks (frames)
  - memory allocation can use a bitmap
- Divide virtual address space in equally-sized chunks (pages)
  - typical size of page/frame: 512B to 16MB
- Contiguous pages must not map to contiguous frames
- Alas, now we face internal fragmentation...

Basic Paging Implementation



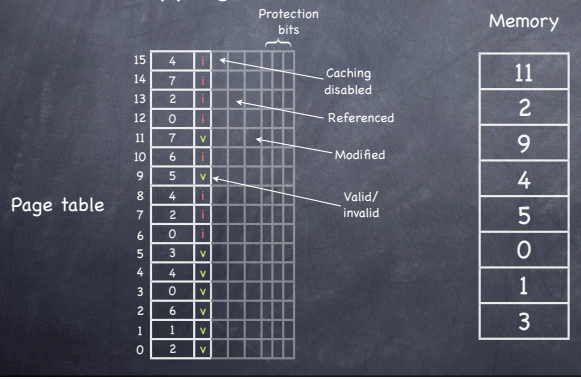
Speeding things up



EAT: $(1+\epsilon)\alpha+(2+\epsilon)(1-\alpha)$
$= 2+\epsilon-\alpha$ (α: hit ratio)

Memory Protection

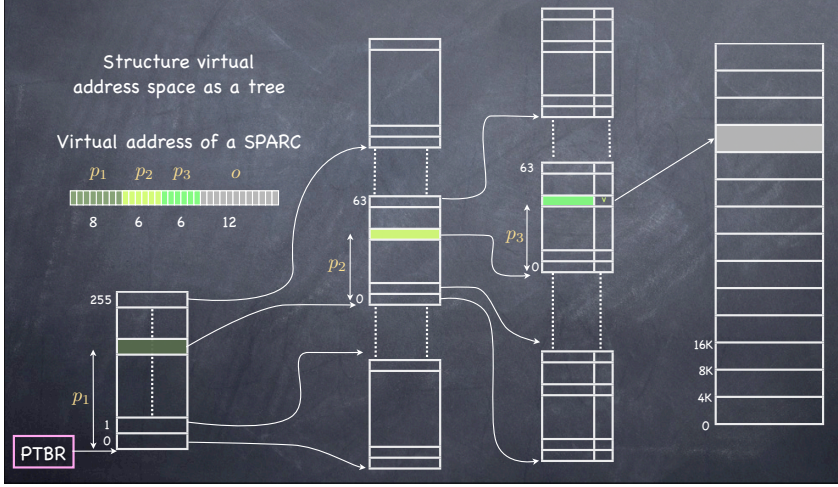
- Used valid/invalid bit to indicate which mappings are active



Oops...

- What is the size of the page table for a machine with 32-bit addresses and a page size of 1KB?

Multi-level Paging



The Challenge of Large Address Spaces

- With large address spaces (64-bits) page tables become cumbersome
  - 5 levels of tables
- A new approach---make tables proportional to the size of the physical, not the virtual, address space
  - virtual address space is growing faster than physical

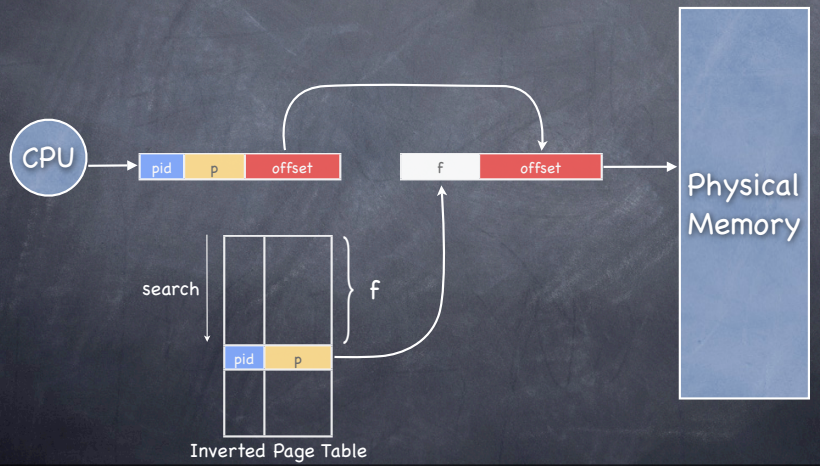
Page Registers (a.k.a. Inverted Page Tables)

- For each frame, a register containing
  - Residence bit
    - is the frame occupied?
  - Page # of the occupying page
  - Protection bits

Catch?

- An example
  - 16 MB of memory
  - Page size: 4k
  - # of frames: 4096
  - Used by page registers (8 bytes/ register): 32 KB
  - Overhead: 0.2%
  - Insensitive to size of virtual memory

Basic Inverted Page Table Architecture

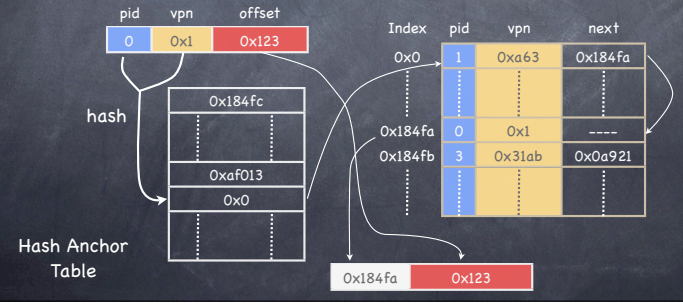


Where have all the pages gone?

- Searching 32KB of registers on every memory reference is not fun
- If the number of frames is small, the page registers can be placed in an associative memory---but...
- Large associative memories are expensive
  - hard to access in a single cycle.
  - consume lots of power

Speeding things up

- Use a TLB: if lucky just as fast as regular page tables...
- ...if unlucky, use a hash table!

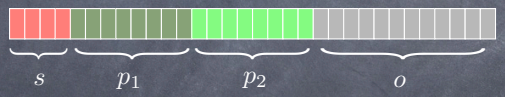


Sharing

- Processes can share the information stored in a memory frame by each having one of their pages mapped to that frame
- How does this sharing compare with segmentation?

Paged segmentation!

- Partition segments into fixed-size pages
- Allocate and deallocate pages



- Virtual address =

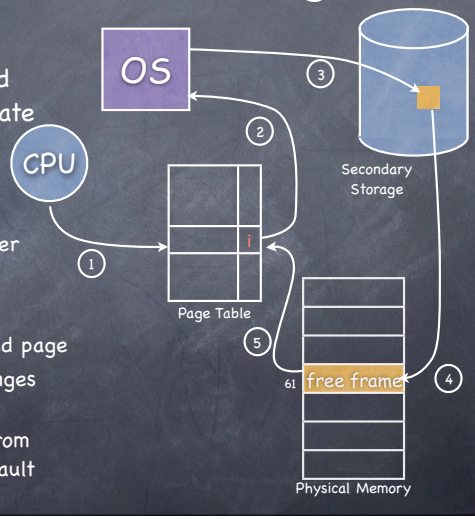
$$((s \times p_1^{max} + p_1) \times p_2^{max} + p_2) \times o^{max} + o$$

Demand Paging

- Code pages are stored in a file on disk
  - some are currently residing in memory–most are not
- Data and stack pages are also stored in a file
- OS determines what portion of VAS is mapped in memory
  - this file is typically invisible to users
  - file only exists while a program is executing
- Creates mapping on demand

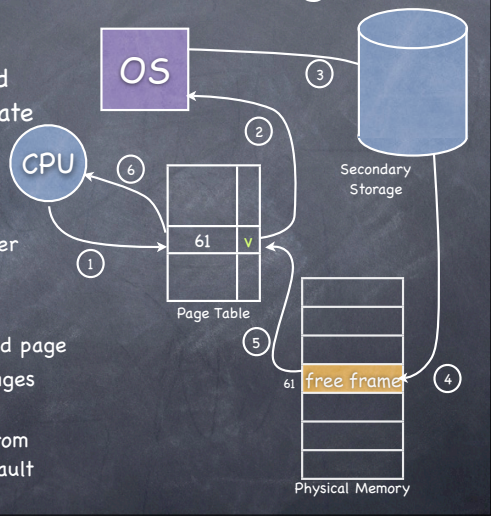
Page-Fault Handling

- References to a non-mapped page (i in page table) generate a page fault
- Handling a page fault:
  - Processor runs interrupt handler
  - OS blocks running process
  - OS finds a free frame
  - OS schedules read of unmapped page
  - When read completes, OS changes page table
  - OS restarts faulting process from instruction that caused page fault



Page-Fault Handling

- References to a non-mapped page (i in page table) generate a page fault
- Handling a page fault:
  - Processor runs interrupt handler
  - OS blocks running process
  - OS finds a free frame
  - OS schedules read of unmapped page
  - When read completes, OS changes page table
  - OS restarts faulting process from instruction that caused page fault



Taking a Step Back

- Physical and virtual memory partitioned into equal-sized units (respectively, frames and pages)
- Size of VAS decoupled to size of physical memory
- No external fragmentation
- Minimizing page faults is key to good performance