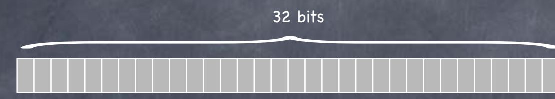
Paging

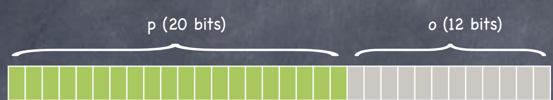
- Allocate VA & PA memory in fixed-sized chunks (pages and frames, respectively)
  - free frames can be tracked using a simple bitmap
    - 0011111001111011110000 one bit/frame
  - no more external fragmentation!
  - but now internal fragmentation (you just can't win...)
    - when memory needs are not a multiple of a page
    - typical size of page/frame: 4KB to 16KB
- Adjacent pages in VA (say, within the stack) need not map to contiguous frames in PA!

Virtual address



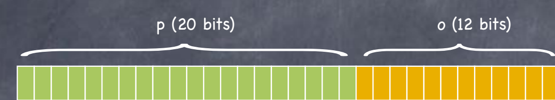
- Interpret VA as comprised of two components
  - page: which page?
  - offset: which byte within that page?

Virtual address



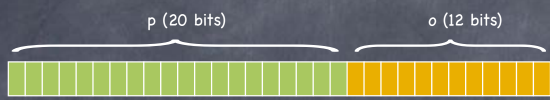
- Interpret VA as comprised of two components
  - page: which page?
    - no. of bits specifies no. of pages are in the VA space
  - offset: which byte within that page?

Virtual address



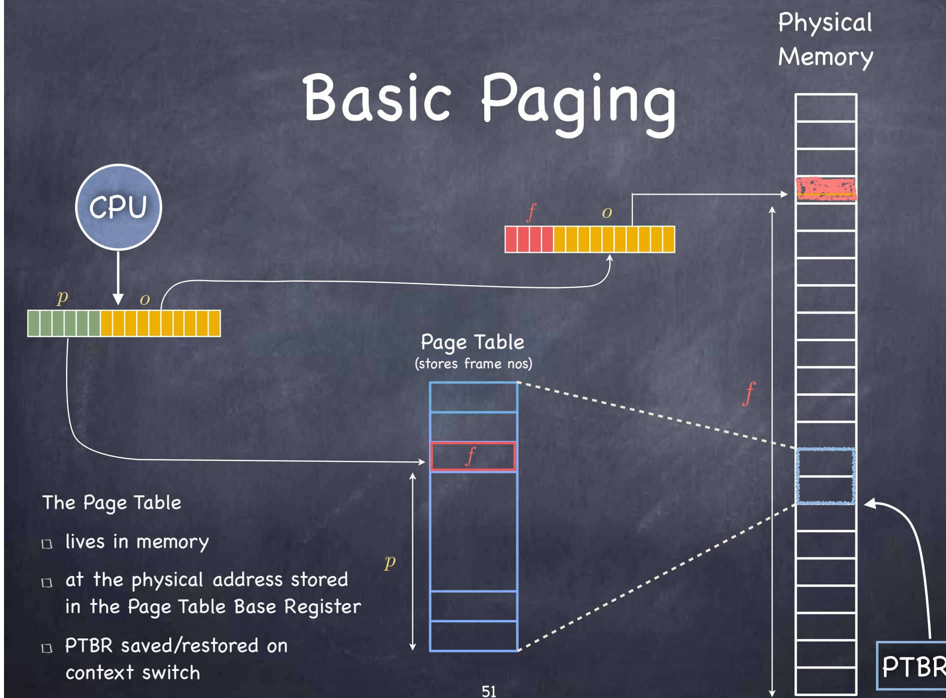
- Interpret VA as comprised of two components
  - page: which page?
    - no. of bits specifies no. of pages are in the VA space
  - offset: which byte within that page?
    - no. of bits specifies size of page/frame

Virtual address



- To access a byte
  - extract page number
  - map that page number into a frame number using a page table
  - extract offset
  - access byte at offset in frame

Basic Paging

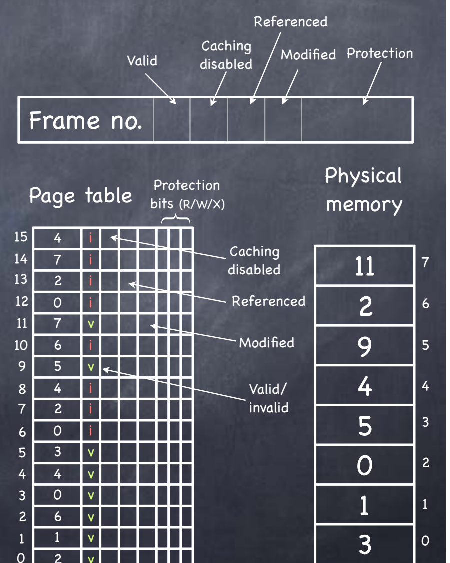


The Page Table

- lives in memory
- at the physical address stored in the Page Table Base Register
- PTBR saved/restored on context switch

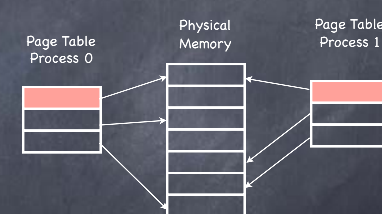
Page Table Entries

- Frame number
- Valid/Invalid bit
  - Set if entry stores a valid mapping If not, page fault
- Cache modified bit
  - Set for memory-mapped I/O
- Referenced bit
  - Set if page has been referenced
- Modified bit
  - Set if page has been modified
- Protection bits (R/W/X)



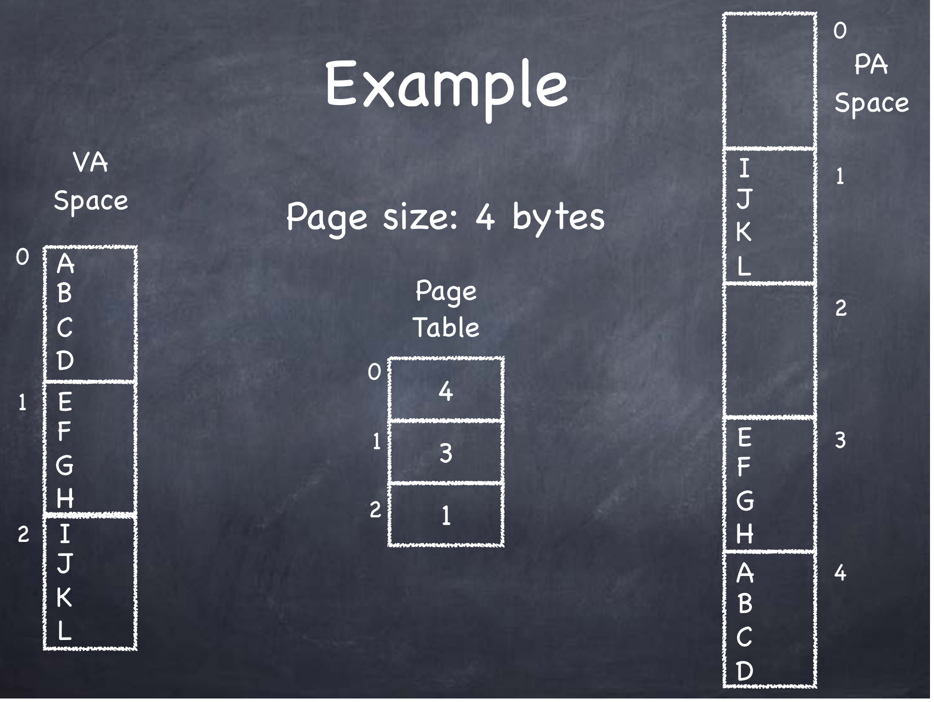
Sharing

- By now, it's old hat:
  - Processes share pages by mapping virtual pages to the same frame
  - Fine tuning using protection bits (RWX)
- We can refine COW to operate at the granularity of pages
  - on fork, mark all pages read only
  - on write, copy only the affected page



Example

Page size: 4 bytes



Space overhead

- Two sources, in tension:
  - data structure overhead (the Page Table itself)
  - fragmentation
    - How large should a page be?

Overhead for paging:

(#entries x sizeofEntry) + (#"segments" x pageSize/2) =

= ((VA_Size/pagesize) x sizeofEntry) + (#"segments" x pageSize/2)

- Size of entry
  - enough bits to identify physical page ($\log_2$ (PA_Size / page size))
  - should include control bits (valid, modified, referenced, etc)
  - usually word or byte aligned

Computing paging overhead

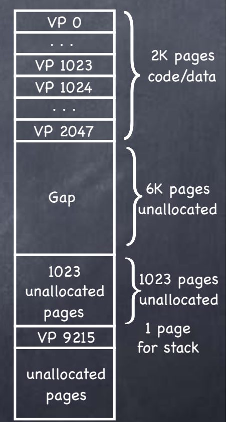
- 1 MB maximum VA, 1 KB page, 3 segments (program, stack, heap)
  - $((2^{20} / 2^{10}) \times$ sizeofEntry$) + (3 \times 2^9)$
  - If I know PA is 64 KB then sizeofEntry = 6 bits (2^6 frames) + control bits
    - if 7 control bits, byte aligned size of entry: 16 bits

What's not to love?

- Space overhead
  - With a 64-bit address space, size of page table can be huge
- Time overhead
  - Accessing data now requires two memory accesses
    - must also access page table, to find mapped frame

Reducing the Storage Overhead of Page Tables

- Size of the page table for a machine with 64-bit addresses and a page size of 4KB?
  - an array of 2^{52} entries!
- Good news
  - most space is unused
- Use a better data structure to express the Page Table
  - a tree!

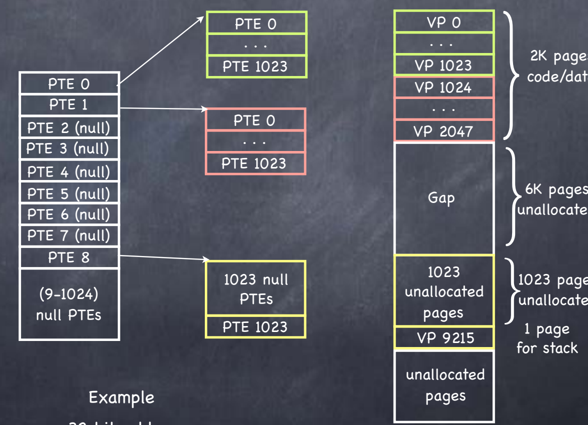


Example

- 32 bit address space
- 4Kb pages
- 4 bytes PTE

Reducing the Storage Overhead of Page Tables

- Size of the page table for a machine with 64-bit addresses and a page size of 4KB?
  - an array of 2^{52} entries!
- Good news
  - most space is unused
- Use a better data structure to express the Page Table
  - a tree!



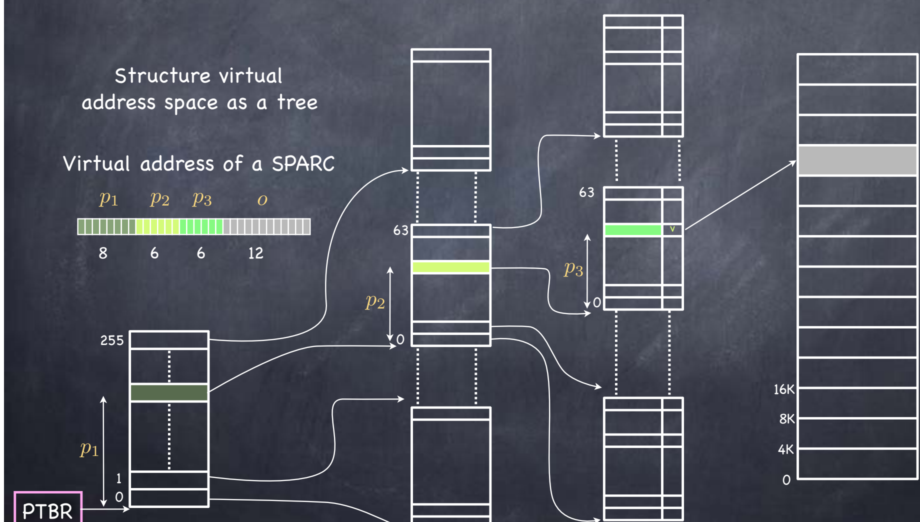
Example

- 32 bit address space
- 4Kb pages
- 4 bytes PTE

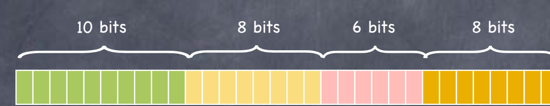
Multi-level Paging

Structure virtual address space as a tree

Virtual address of a SPARC

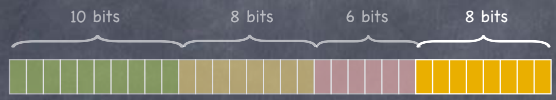


Example



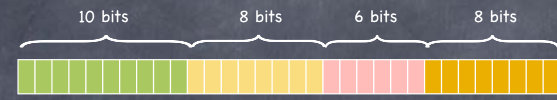
- What is the page size?

Example

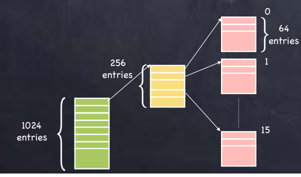


- What is the page size? Page size is 256 bytes
- What is the Page Table size for a process with a 256KB VAS starting at address 0?

Example



- What is the page size? Page size is 256 bytes
- What is the Page Table size for a process with a 256KB VAS starting at address 0?
  - The process needs to map 1024 pages (256KB/256 bytes)
  - To do so, it is sufficient to materialize the following tree since 1024/64 = 16



  - So, assuming each PTE is 2 bytes

1024 x 2 + 256 x 2 + 16 x 64 x 2 = 4608 bytes