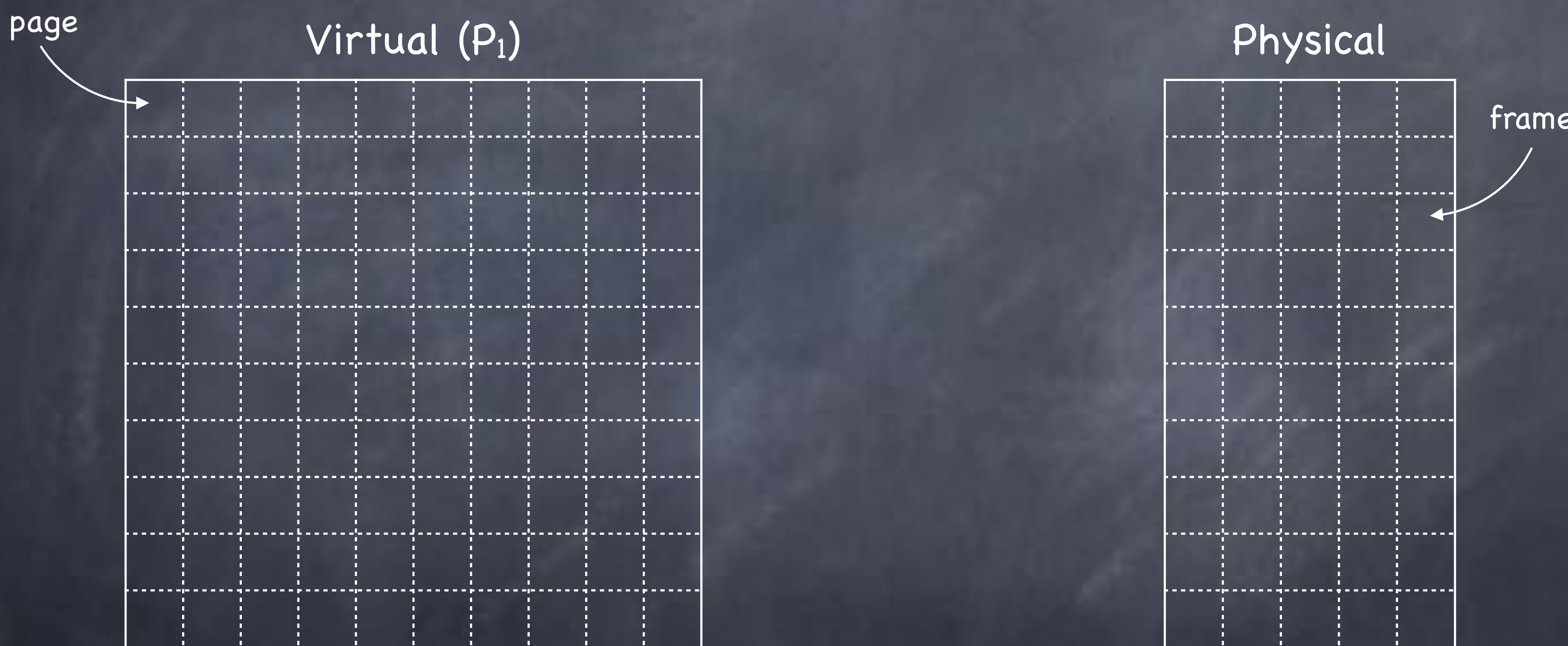
Eliminating External Fragmentation: Tiling Memory

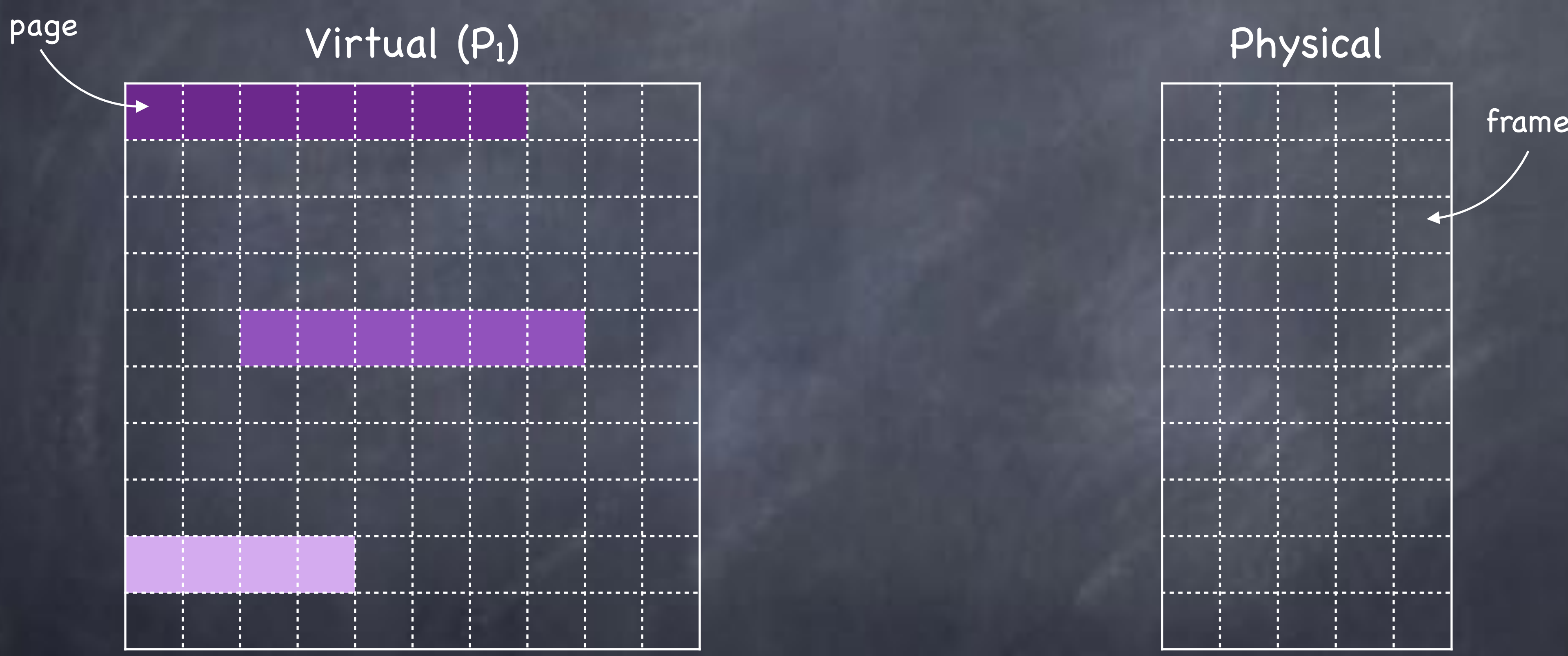
Virtual (P_1)

Physical

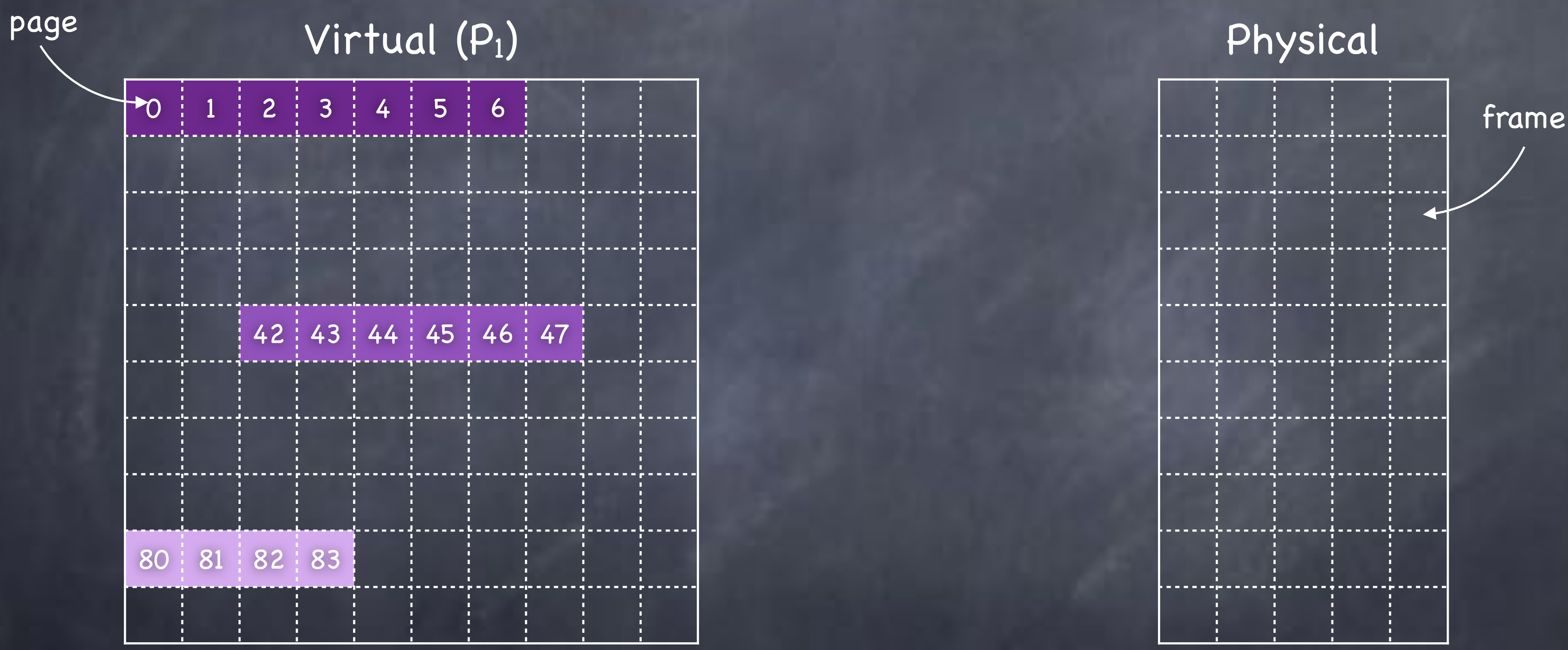
Tiling Memory



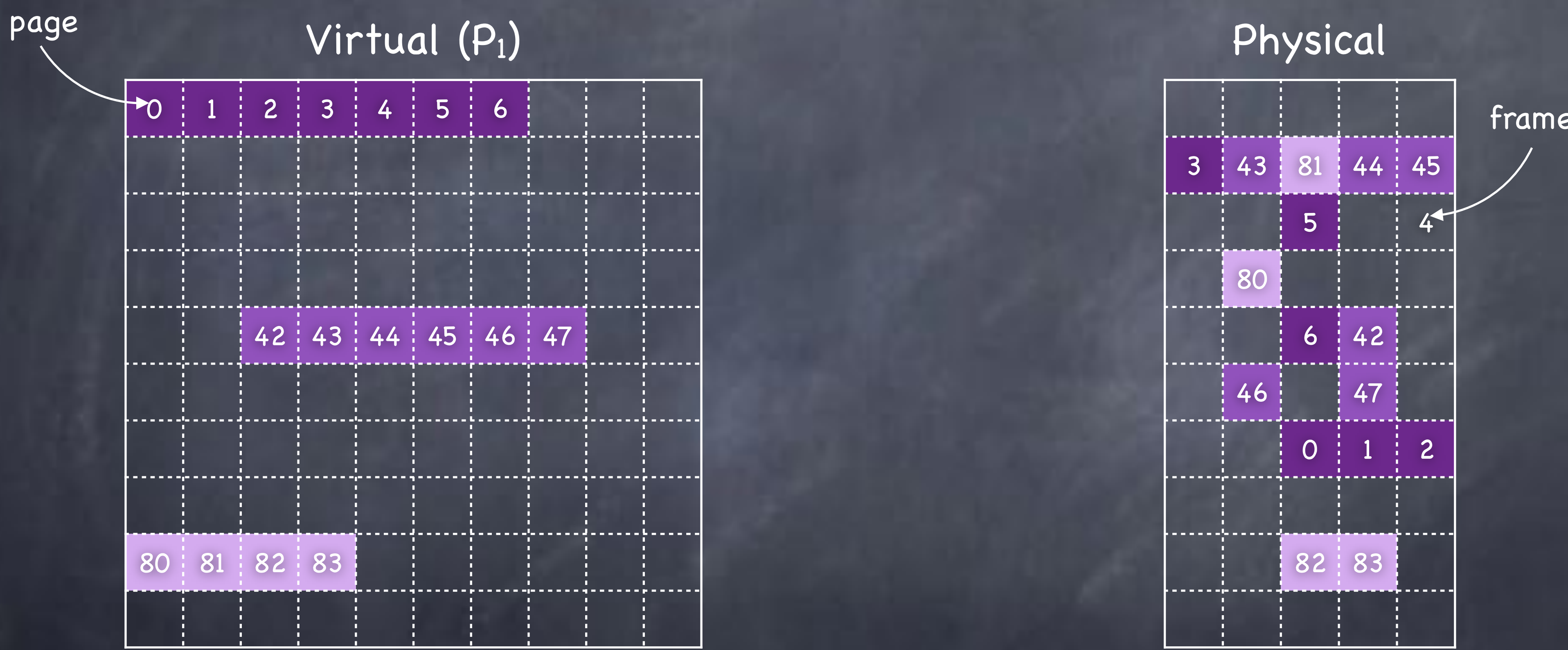
Tiling Memory



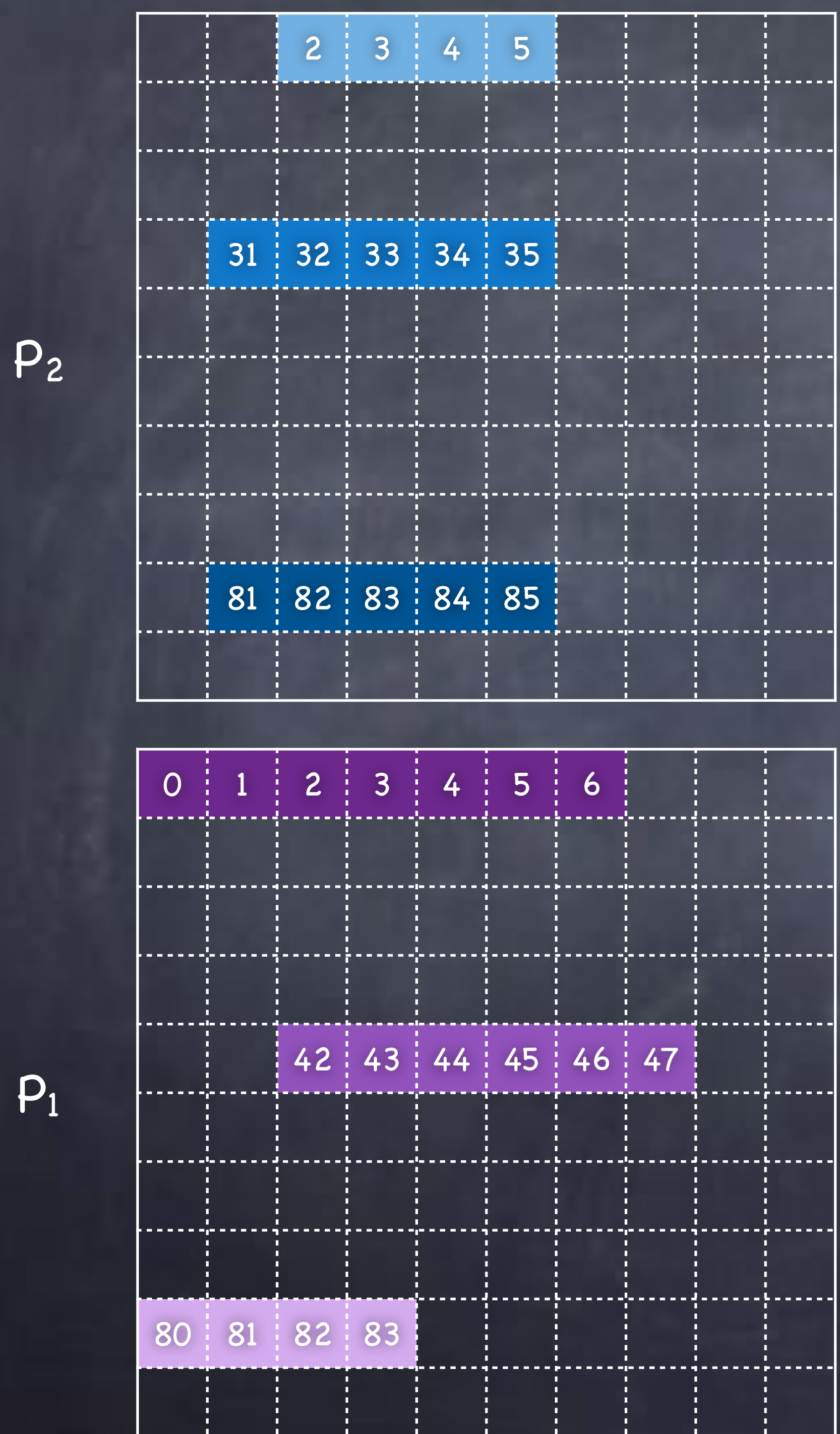
Tiling Memory

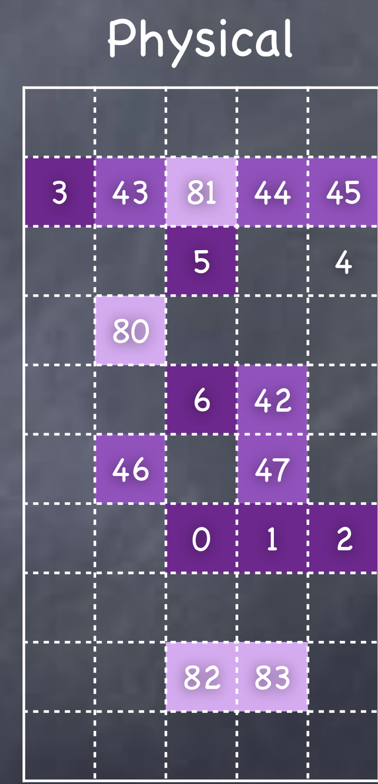


Tiling Memory

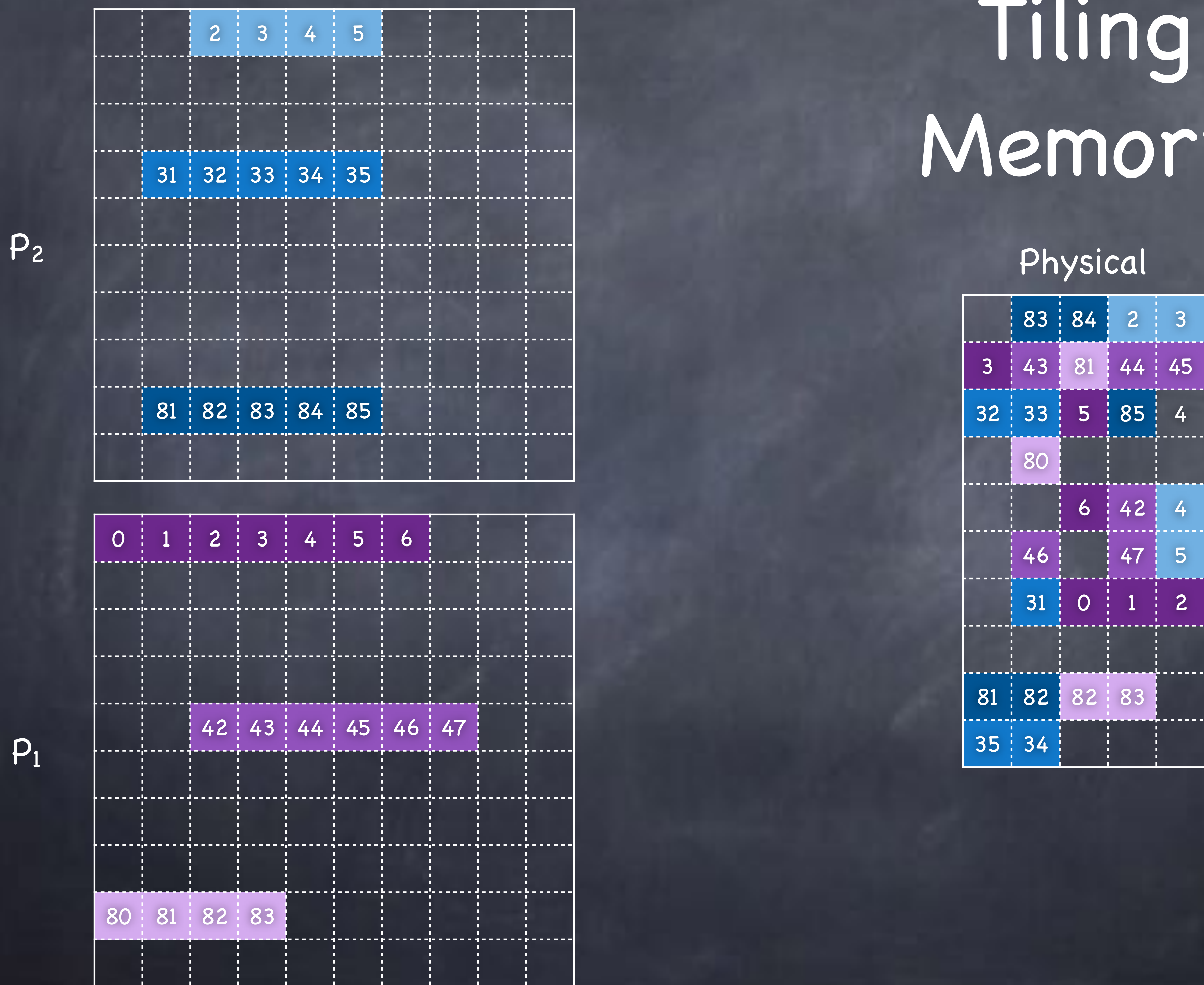


Tiling Memory





Tiling Memory

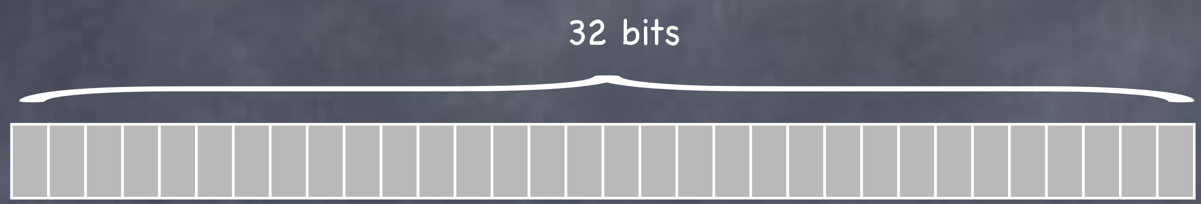


Eliminating External Fragmentation: Paging

- Allocate VA & PA memory in chunks of the same, fixed size (pages and frames, respectively)
- Adjacent pages in VA (say, within the stack) need not map to contiguous frames in PA!
  - 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

How can I reference a byte in VA space?

Virtual address



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

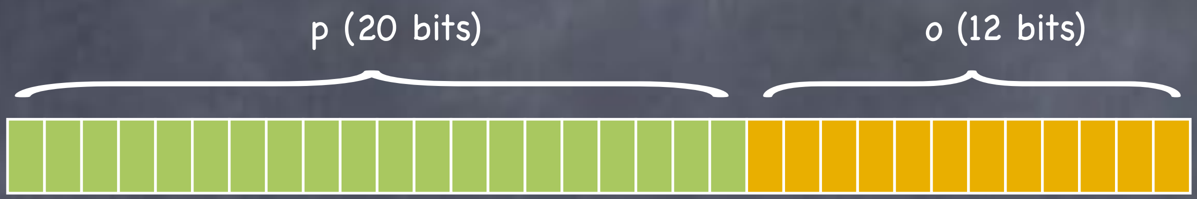
Virtual address

- 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?

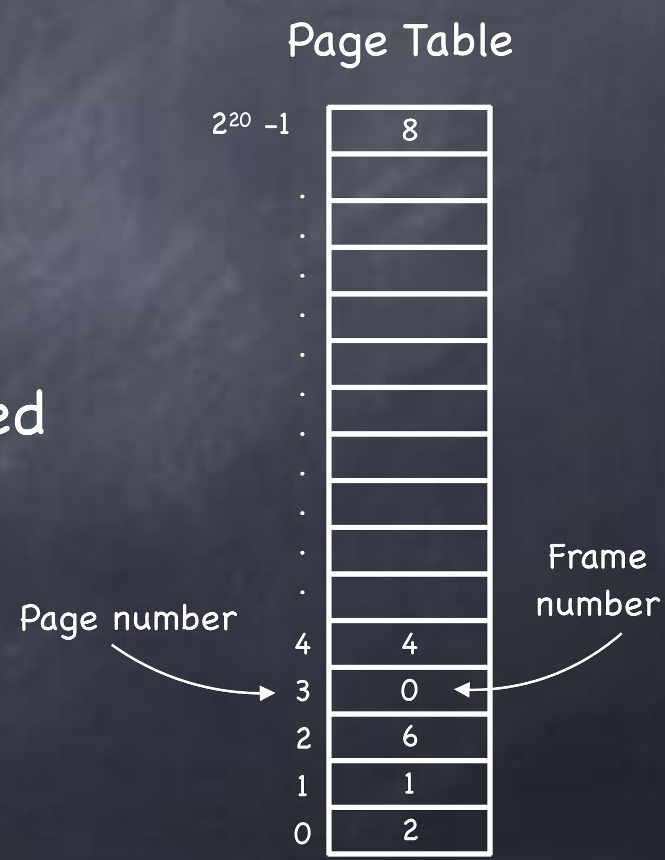
Virtual address

- 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

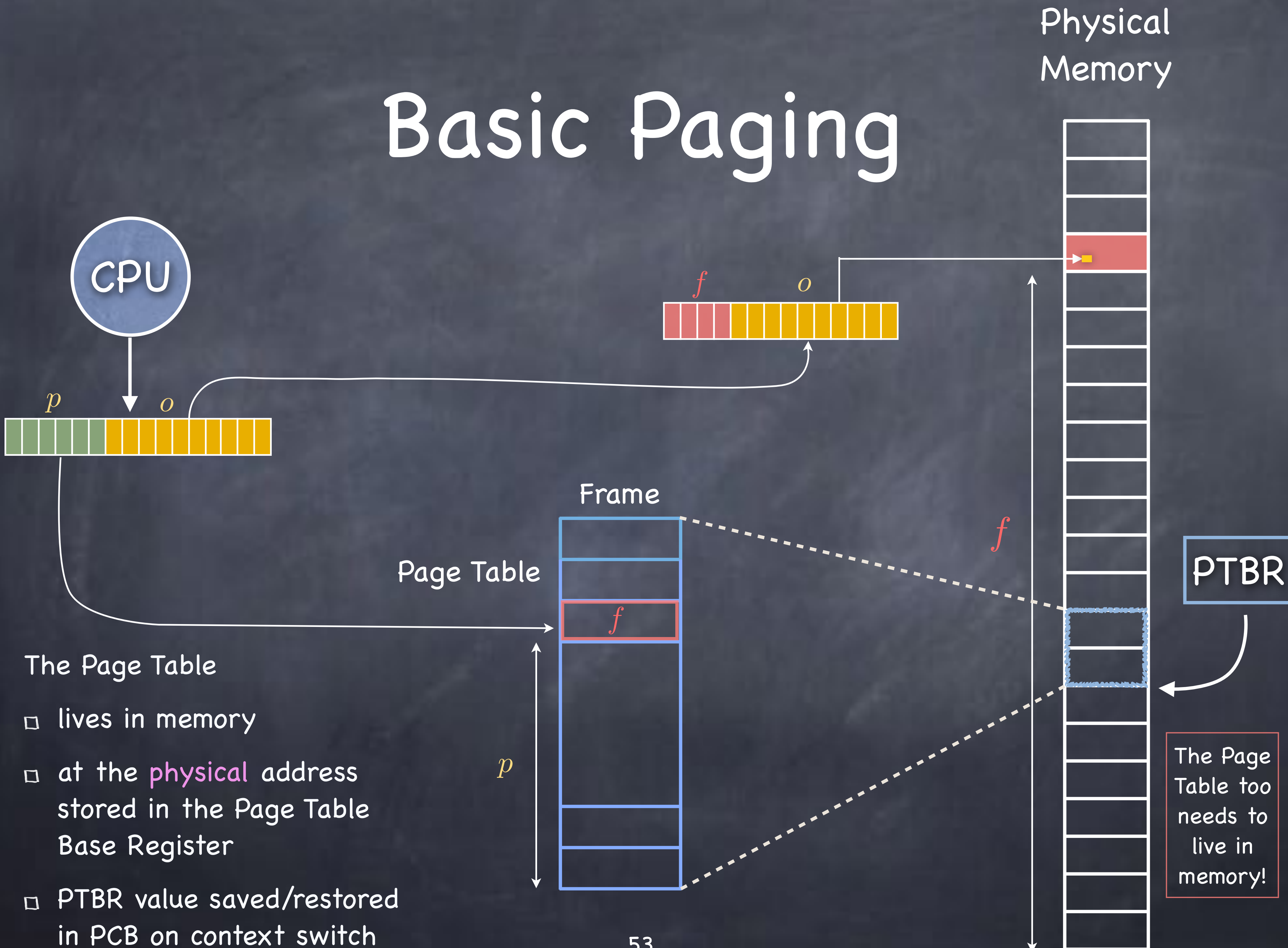
Virtual address



- To access a byte
  - extract page number
  - map that page number into a frame number using a page table
    - Note: not all pages may be mapped to frames
  - extract offset
  - access byte at offset in frame



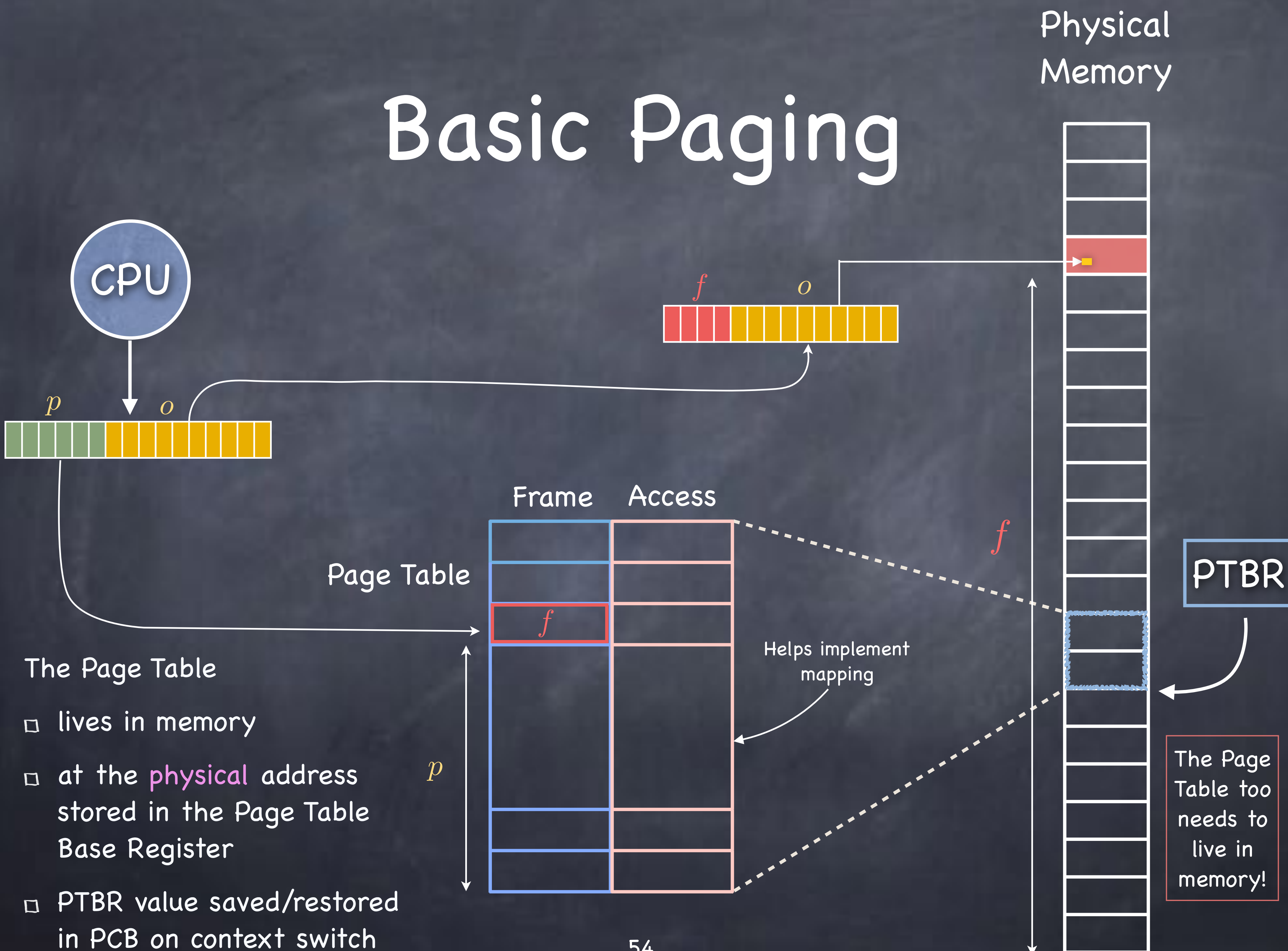
Basic Paging



The Page Table

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

Basic Paging

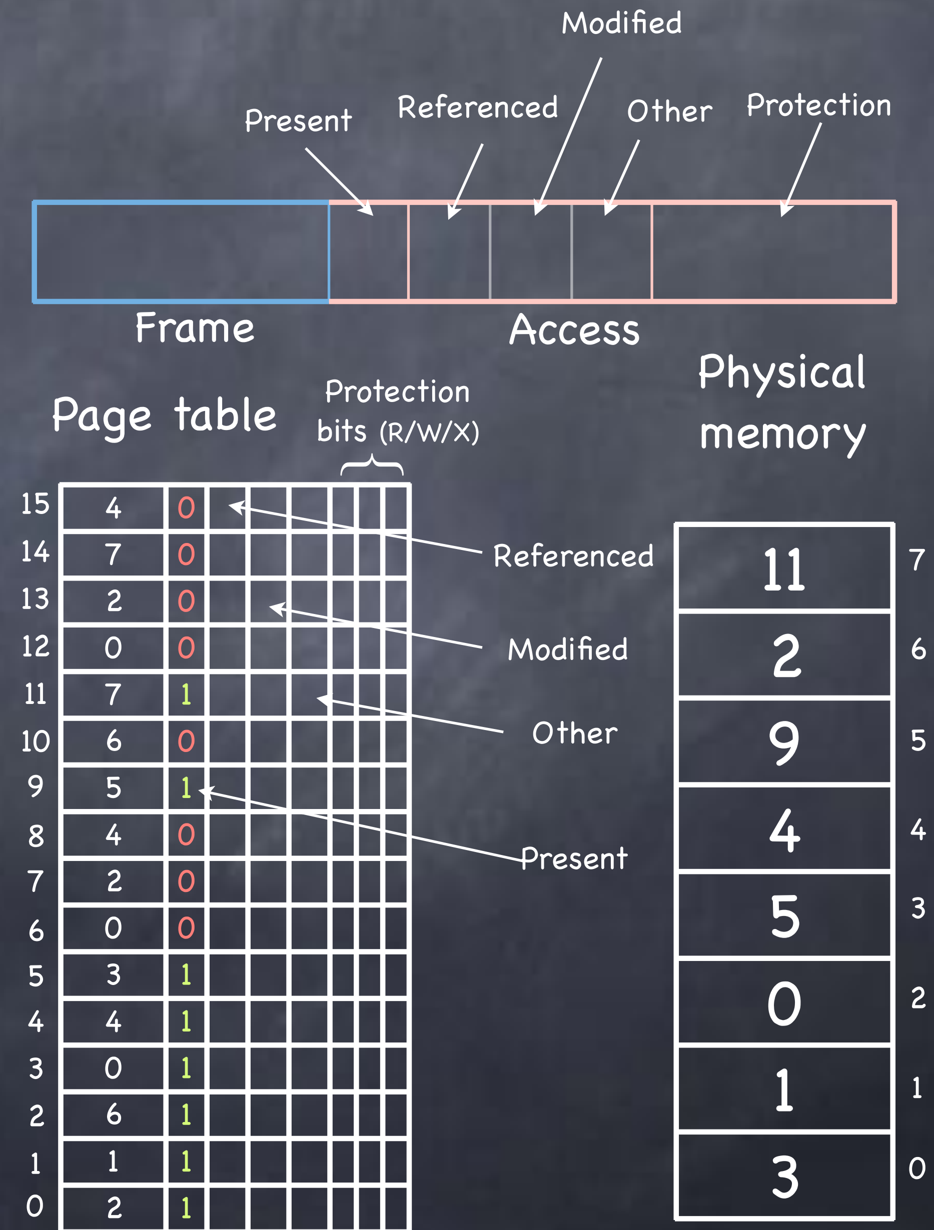


The Page Table

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

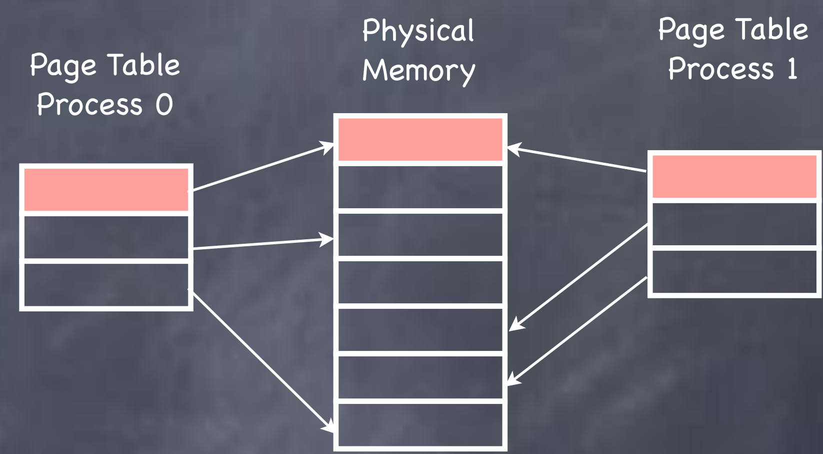
Page Table Entries

- Frame number
- Present (Valid/Invalid) bit
  - Set if entry stores a valid mapping. If not, and accessed, page fault
- Referenced bit
  - Set if page has been referenced
- Modified (dirty) bit
  - Set if page has been modified
- Protection bits (R/W/X)



Sharing

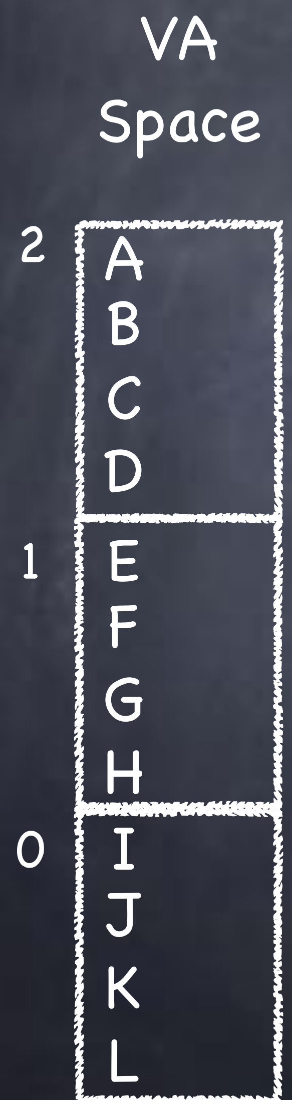
- Processes share a page by each mapping a page of their own virtual address space to the same frame
  - Fine tuning using protection bits (RWX)

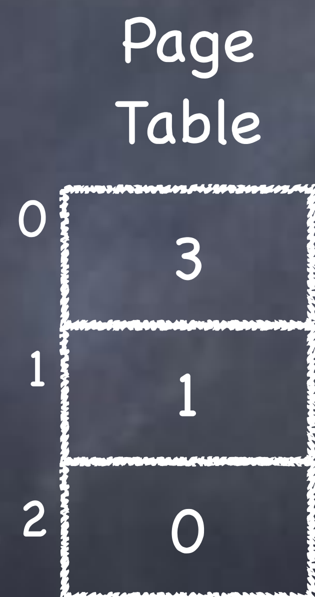


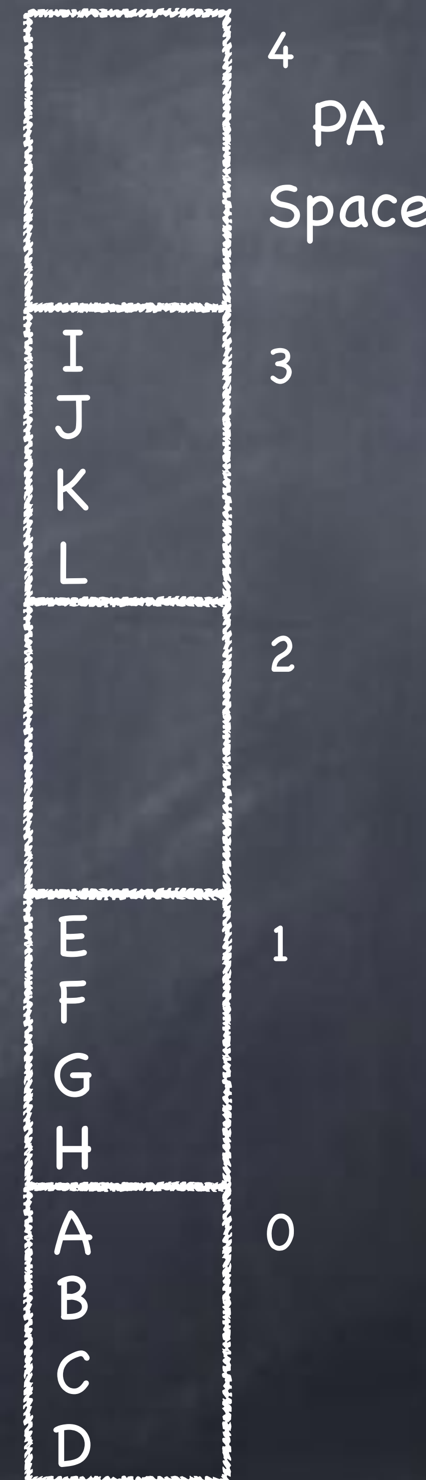
- We can refine COW to operate at the granularity of pages
  - on fork(), mark all pages in page table Read only
  - on write:
    - page fault
    - allocate new frame
    - copy page
    - mark both pages R/W

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:

sets of contiguous pages

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

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

  - What determines sizeofEntry?
    - enough bits to identify physical page ($\log_2$ (PA_Size / page size))
    - should include control bits (present, dirty, referenced, etc)
    - usually word or byte aligned

Computing paging overhead

- 1 MB maximum VA, 1 KB page, 3 segments (program, stack, heap)
  - $((2^{20} / 2^{10}) \times \text{sizeofEntry}) + (3 \times 2^{9})$
  - If I know PA is 64 KB then sizeofEntry = sizeofFrameNo + #ofAccessBits = 6 (since we have 2^6 frames) + #ofAccessBits
    - if 7 access 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

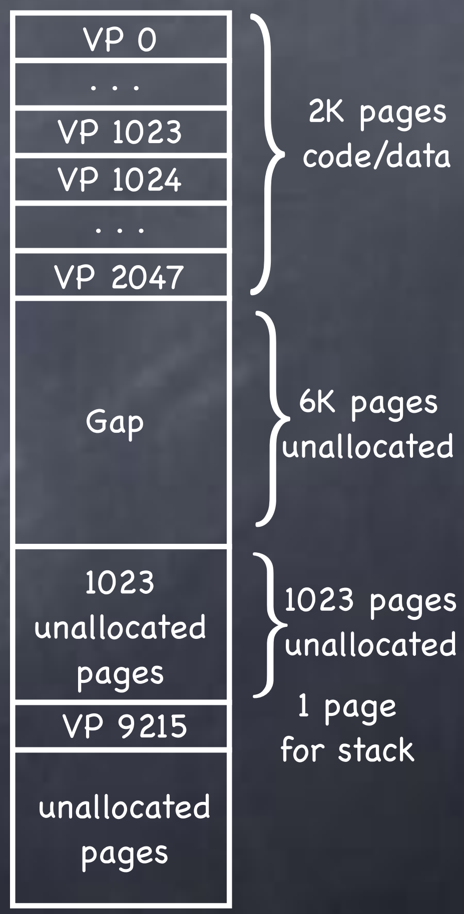
...and, like most times, space and time are in tension...

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

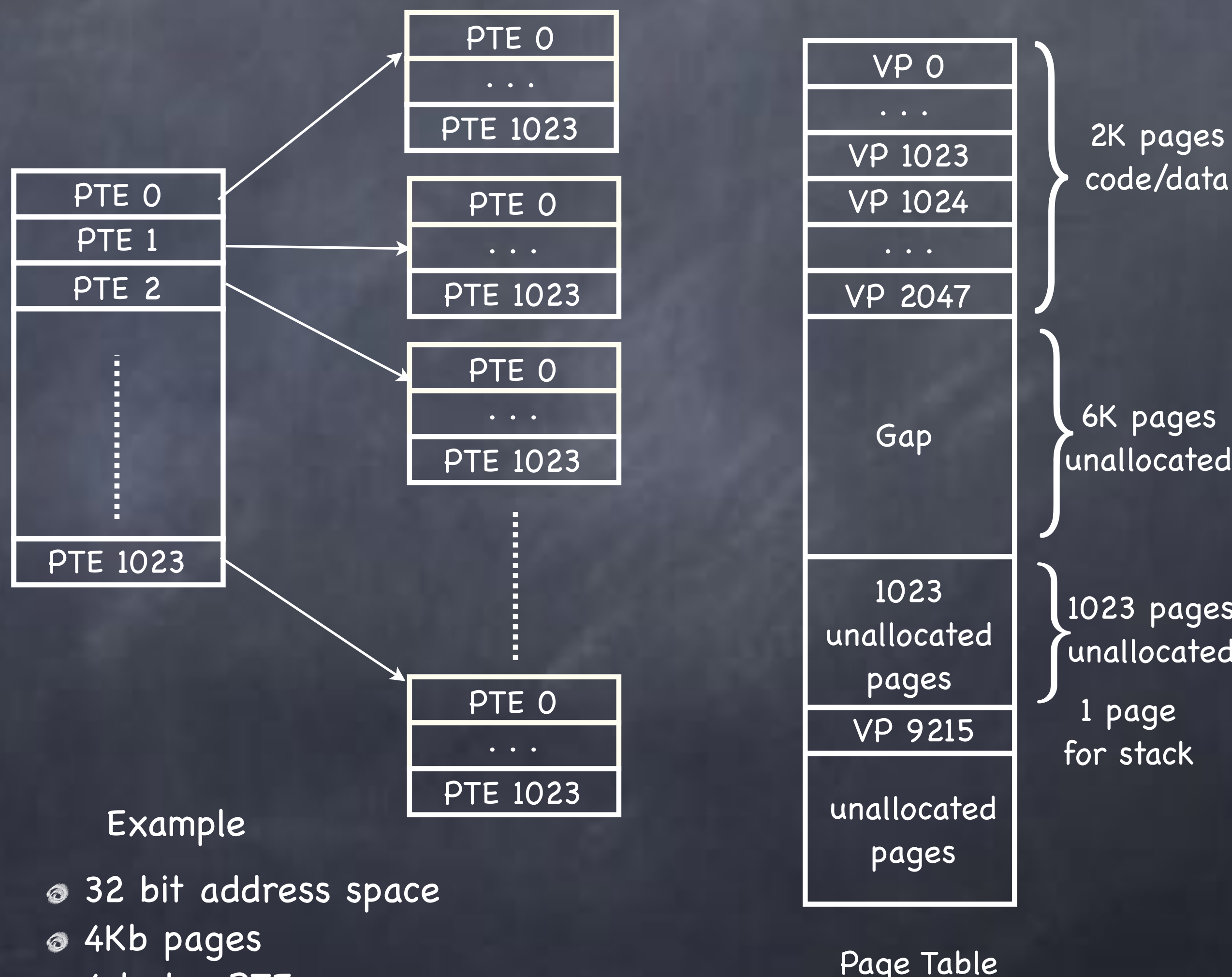
- 32 bit address space
- 4Kb pages
- 4 bytes PTE



Page Table

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!

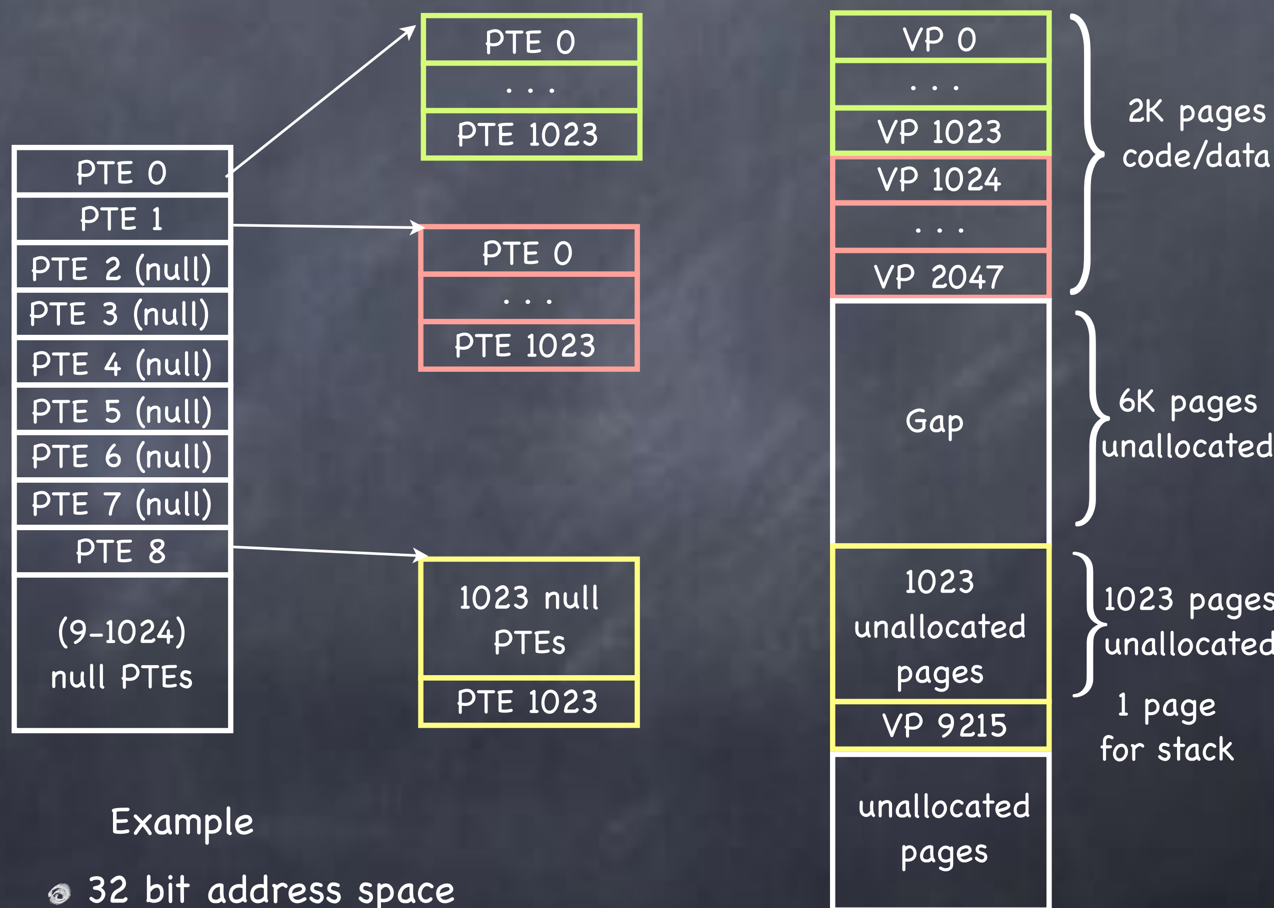


Example

- 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

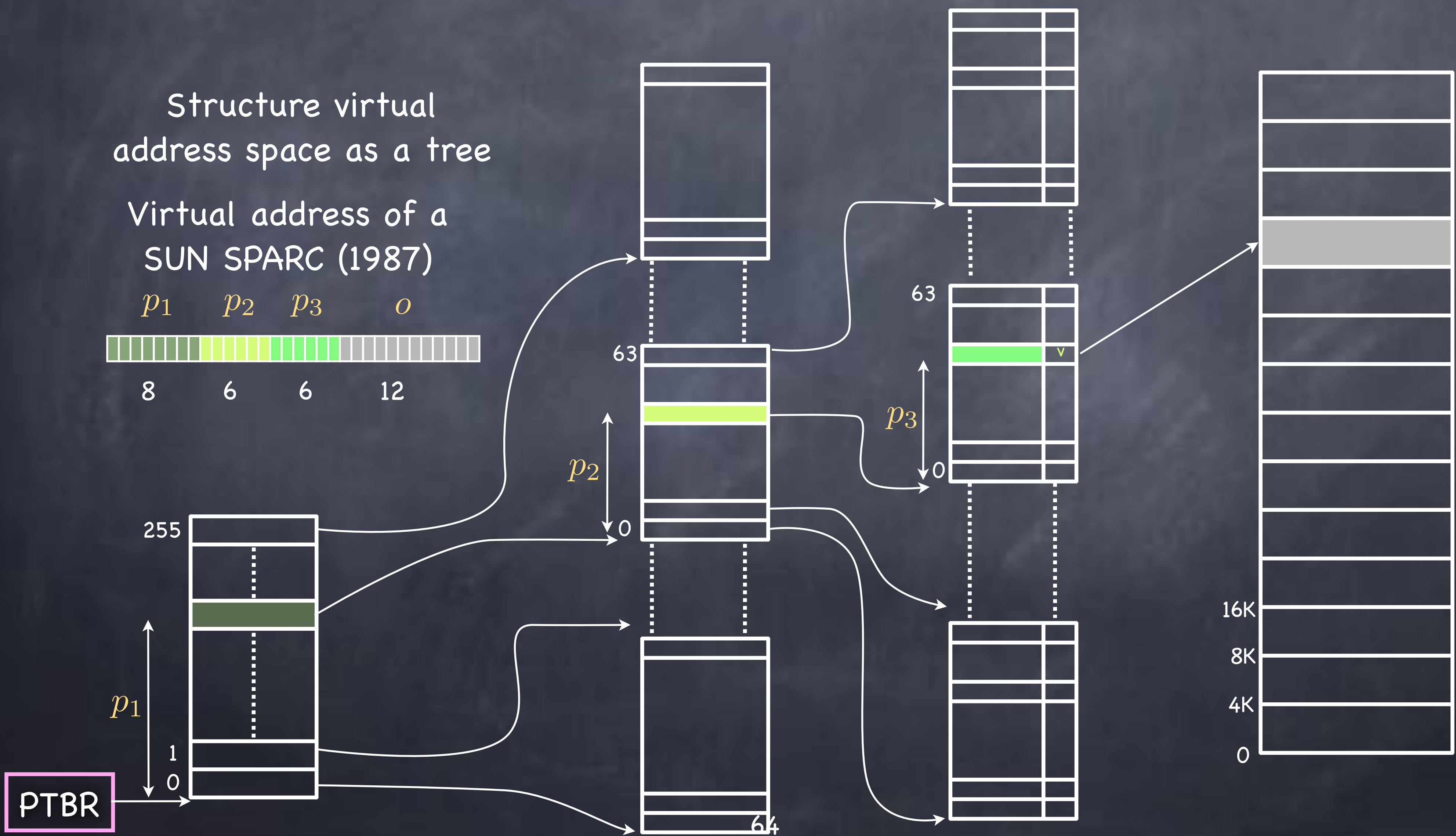
- 32 bit address space
- 4Kb pages
- 4 bytes PTE

Page Table

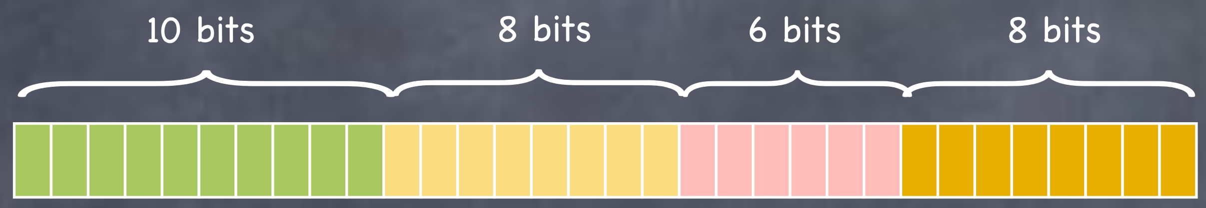
Multi-level Paging

Structure virtual address space as a tree

Virtual address of a SUN SPARC (1987)

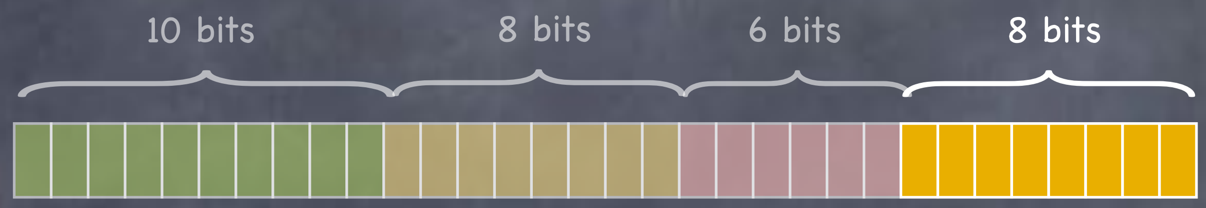


Example



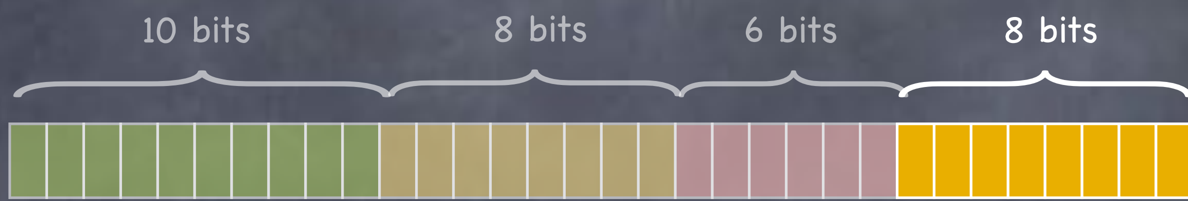
- What is the page size?

Example



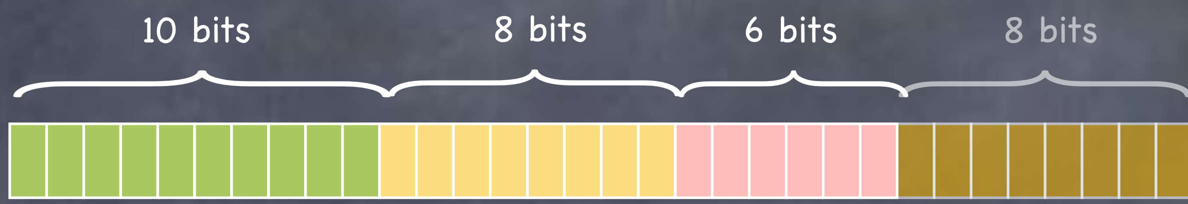
- What is the page size? Page size is 256 bytes (2^8)
- What is the Page Table size for a process that uses 256 contiguous KB of its VA space starting at address 0? [Assume each PTE is 2 bytes]
  - if we used a linear representation of the page table:

Example



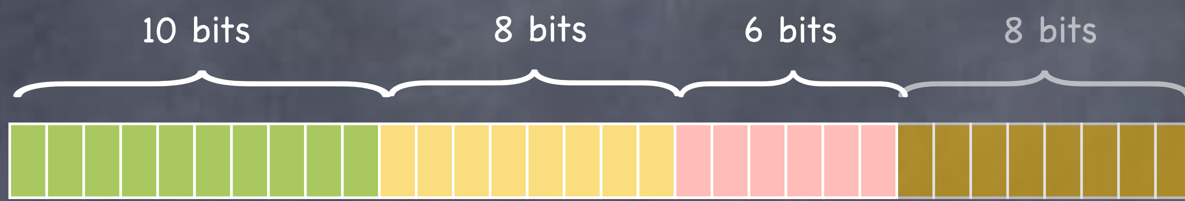
- What is the page size? Page size is 256 bytes (2^8)
- What is the Page Table size for a process that uses 256 contiguous KB of its VA space starting at address 0? [Assume each PTE is 2 bytes]
  - if we used a linear representation of the page table:
    - Page Table has 2^{24} entries

Example



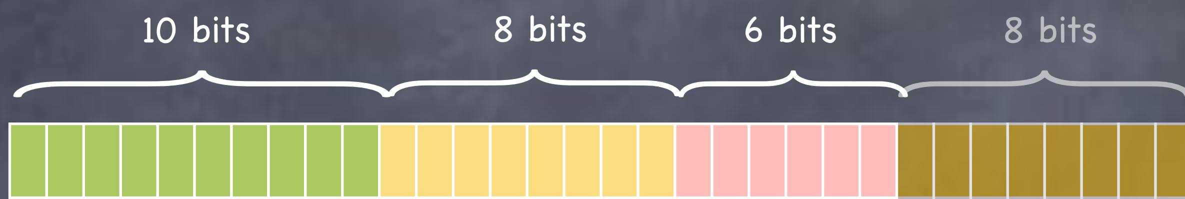
- What is the page size? Page size is 256 bytes (2^8)
- What is the Page Table size for a process that uses 256 contiguous KB of its VAS starting at address 0? [Assume each PTE is 2 bytes]
  - if we used a linear representation of the page table:
    - Page Table has 2^{24} entries
    - PT Size: 2^{24} x 2 bytes = 2^{25} bytes = 32MB

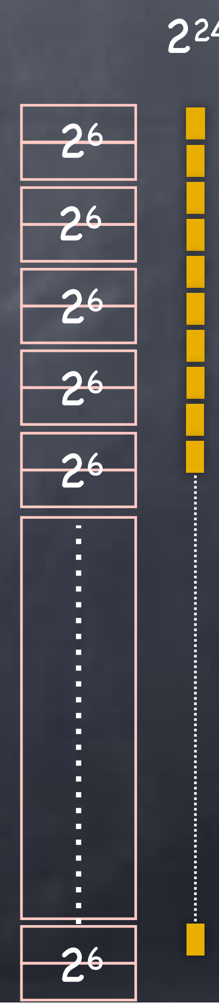
Example



- What if we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries

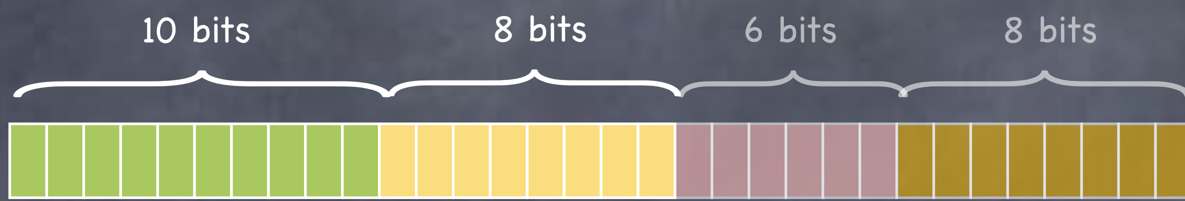
Example

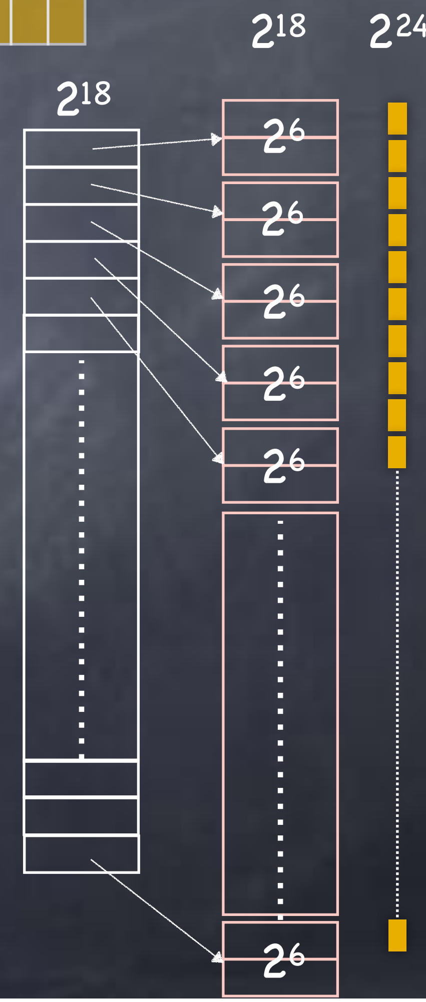




- What is we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries. How many such chunks?

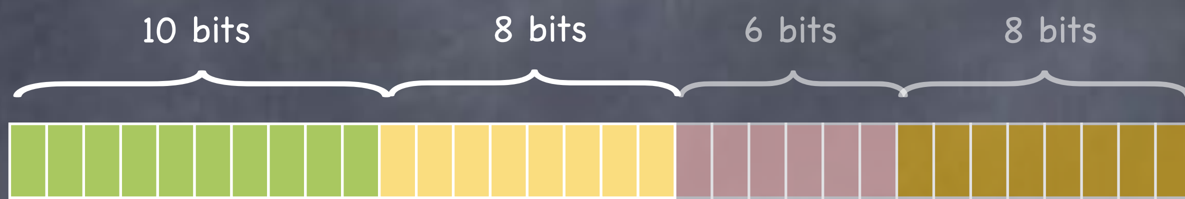
Example

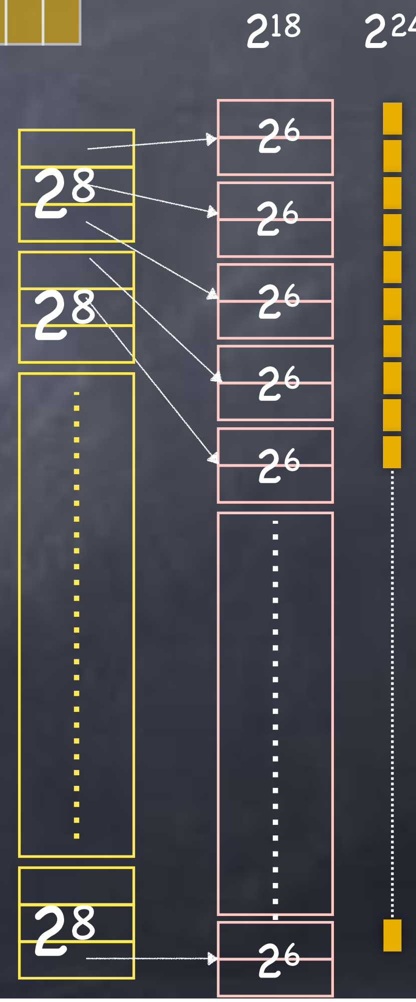




- What is we use a tree?
  - We still need to account for 2^{24} pages…
  - …but we are going to partition the PT in a sequence of chunks, each with 2^6 entries
  - we'll need an index with 2^{18} entries…
  - …which we'll partition in chunks of 2^8 entries

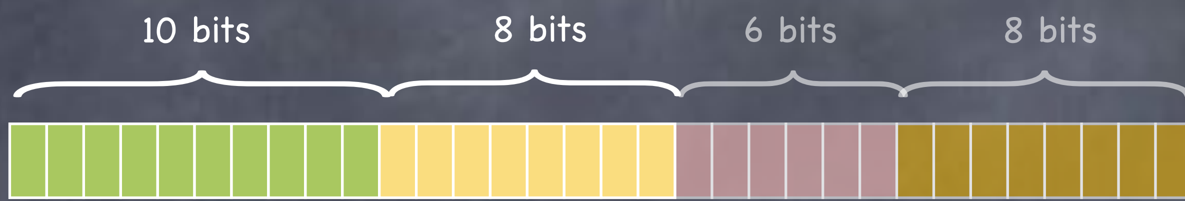
Example



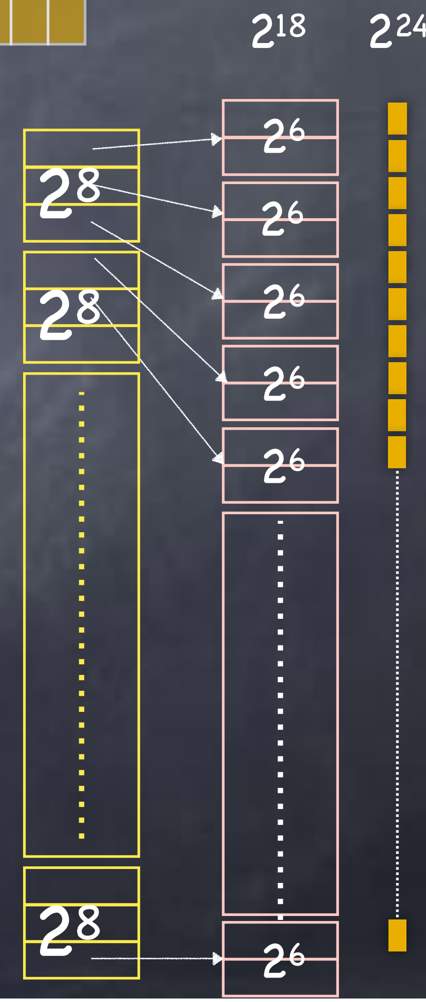


- What is we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries
  - we'll need an index with 2^{18} entries...
  - ...which we'll partition in chunks of 2^8 entries. How many such chunks?

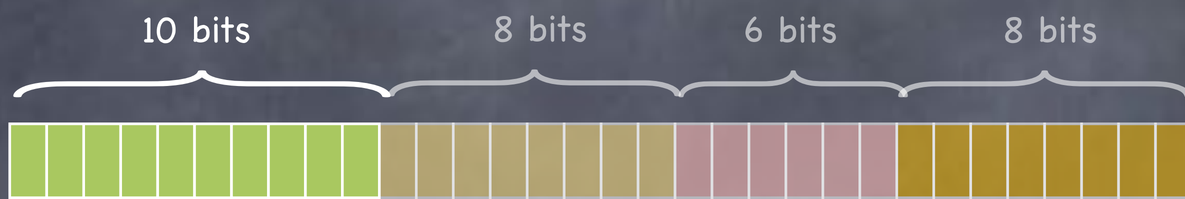
Example



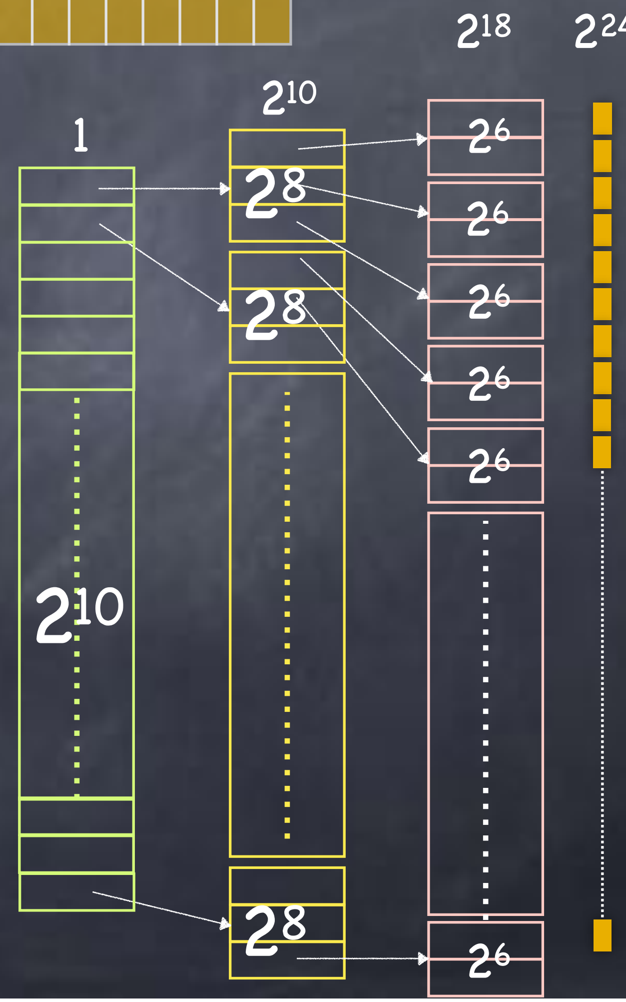
- What is we use a tree?
  - We still need to account for 2^{24} pages…
  - …but we are going to partition the PT in a sequence of chunks, each with 2^6 entries
  - we'll need an index with 2^{18} entries…
  - …which we'll partition in chunks of 2^8 entries
  - We'll need an index of 2^{10}



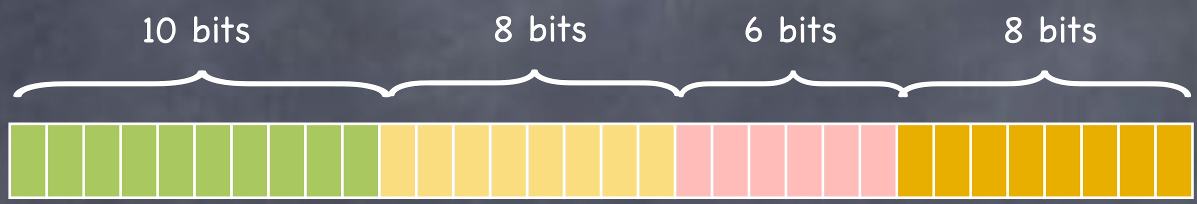
Example



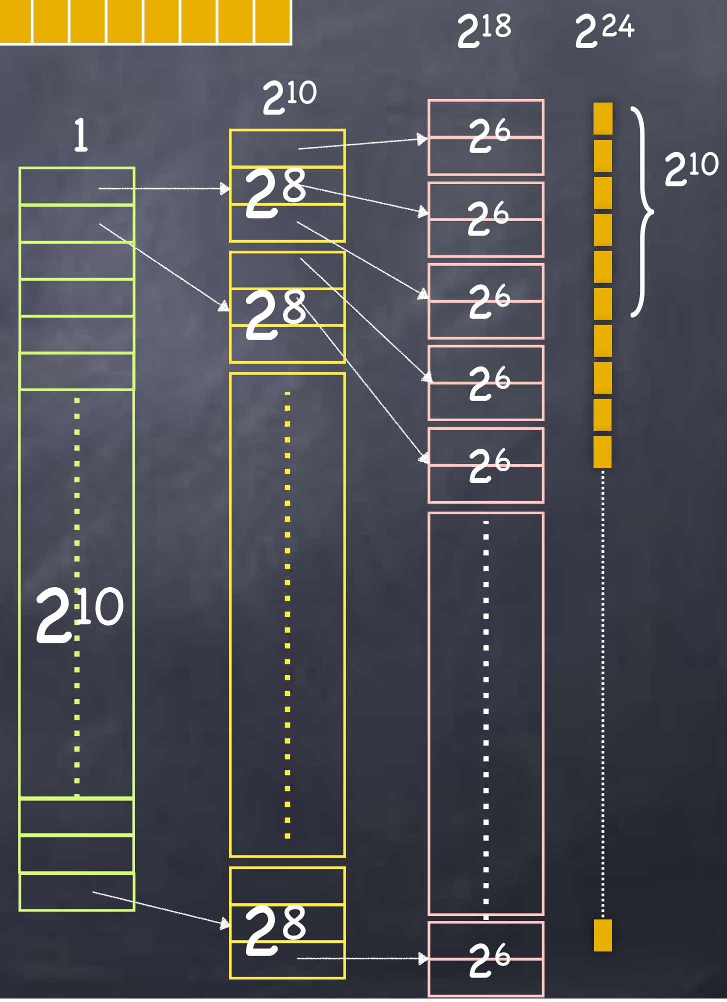
- What is we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries
  - we'll need an index with 2^{18} entries...
  - ...which we'll partition in chunks of 2^8 entries
  - We'll need an index of 2^{10}



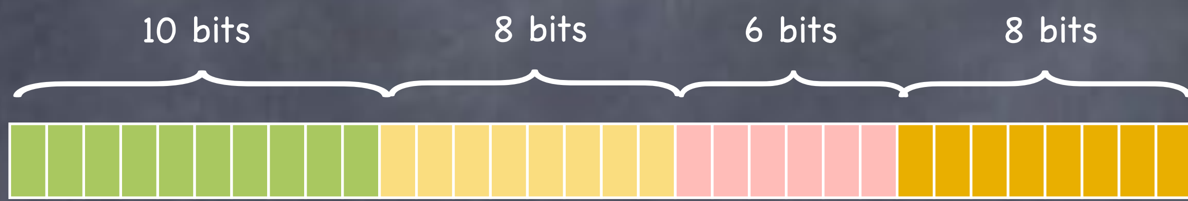
Example



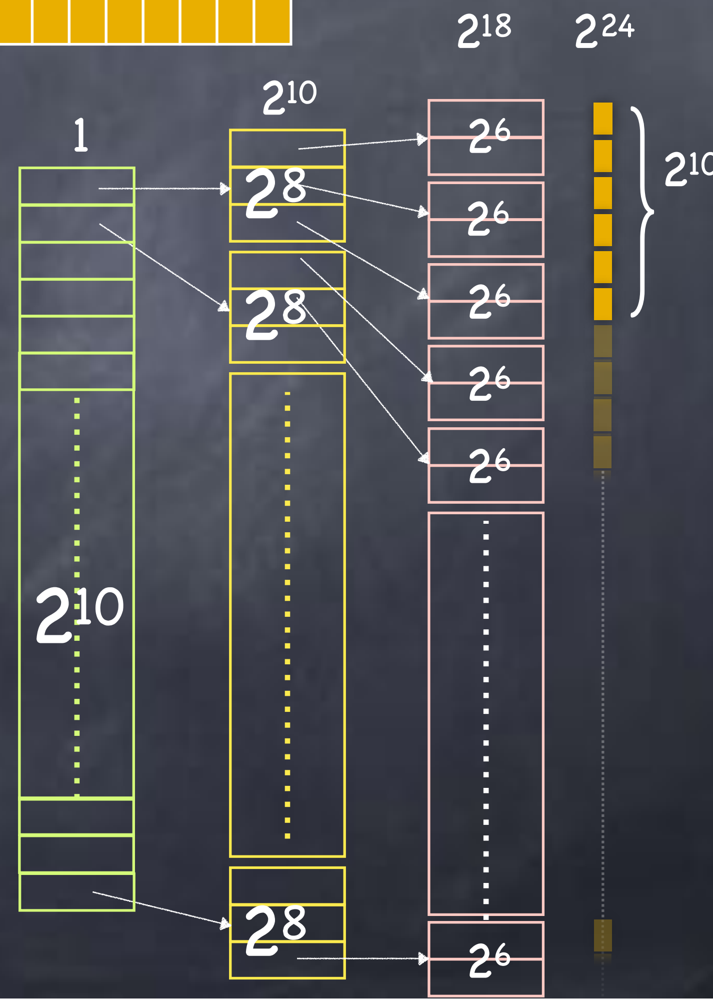
- Are we better off?
  - The number of PT entries now is $(2^6 \times 2^{18}) + (2^{10} \times 2^8) + 2^{10} > 2^{24}$!!
  - But we only need the portion of the tree needed to map the first 1K (2^{10}) pages!



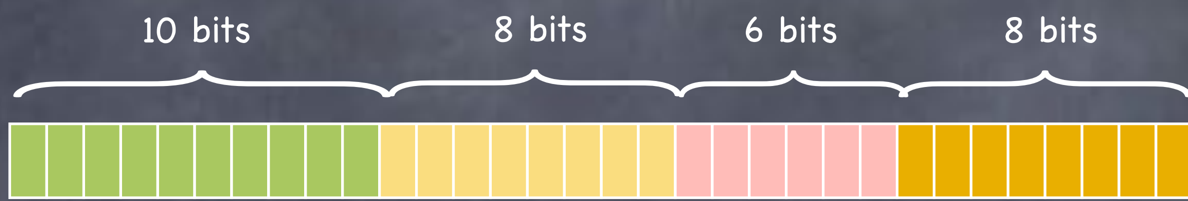
Example

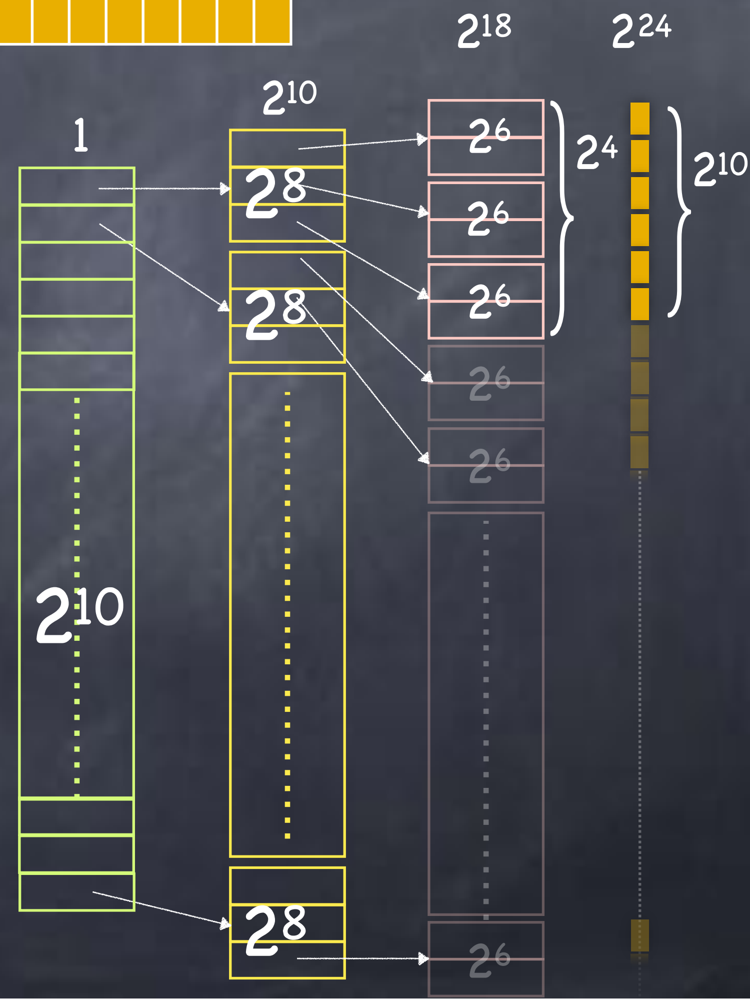


- How many chunks of size 2^6 are needed to hold 2^{10} PTEs of consecutive pages starting at 0?
  - $2^{10}/2^6 = 2^4 = 16$



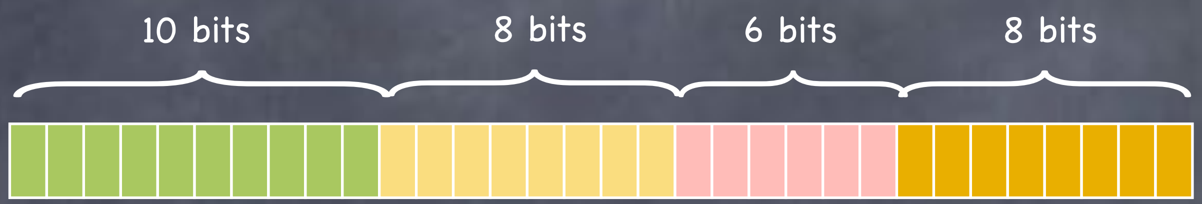
Example



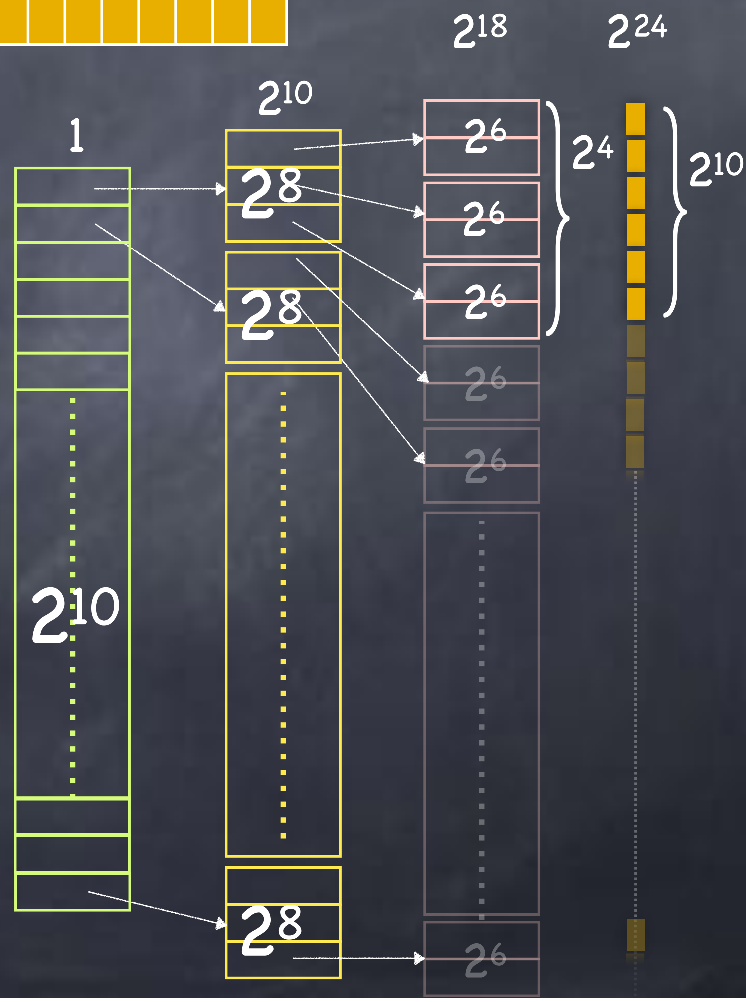


- How many chunks of size 2^6 are needed to hold 2^{10} PTEs of consecutive pages starting at 0?
  - $2^{10}/2^6 = 2^4 = 16$

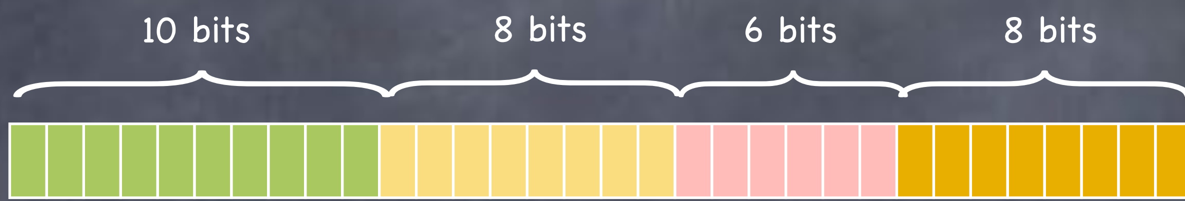
Example

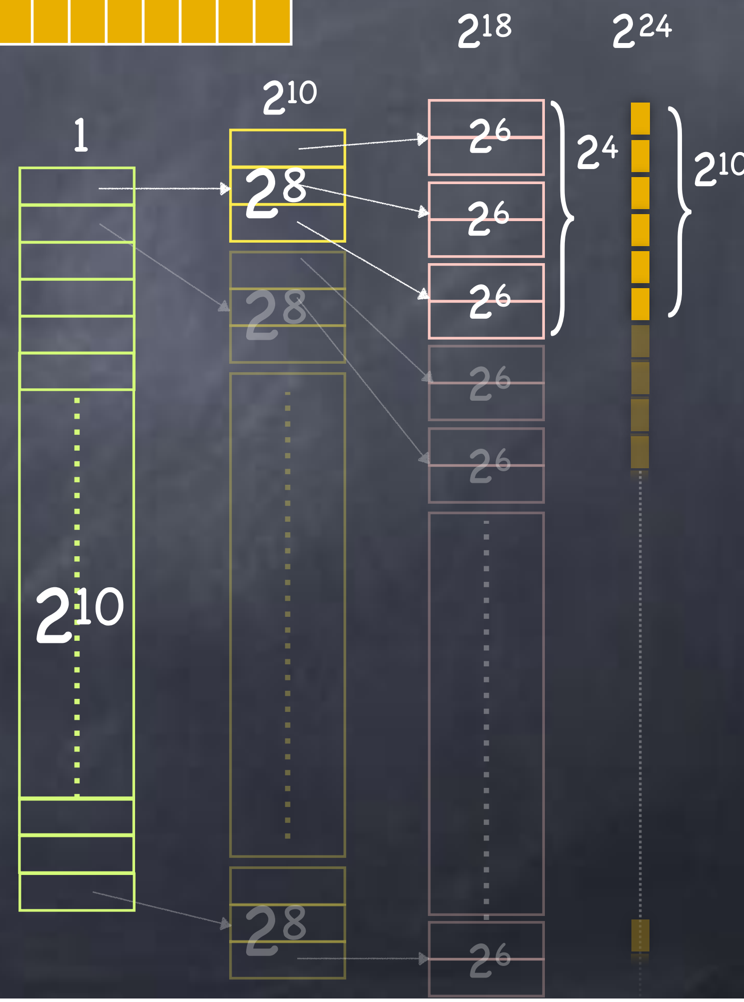


- How many chunks of size 2^6 are needed to hold 2^{10} PTEs of consecutive pages starting at 0?
  - $2^{10}/2^6 = 2^4 = 16$
- How many chunks of size 2^8 are needed to hold pointers to 16 pink chunks?
  - 1



Example





- How many chunks of size 2^6 are needed to hold 2^{10} PTEs of consecutive pages starting at 0?
  - $2^{10}/2^6 = 2^4 = 16$
- How many chunks of size 2^8 are needed to hold pointers to 16 pink chunks?
  - 1
- So, if each PTE is 2 bytes, the PT takes
  - $2 \times (1 \times 1024 + 1 \times 256 + 16 \times 64) = 4608$ bytes