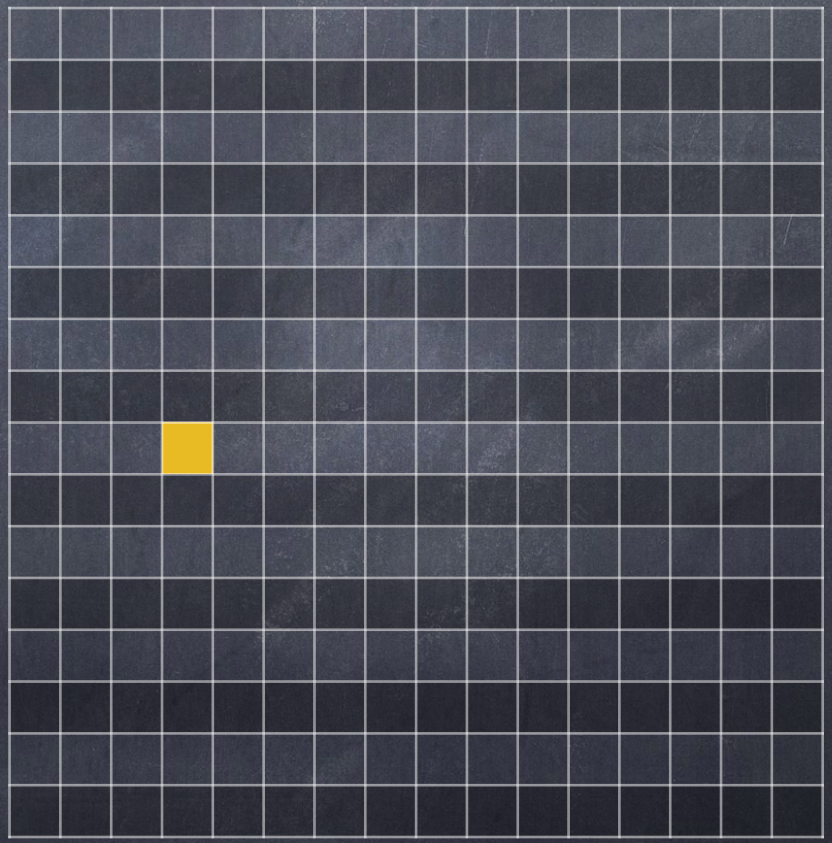
Previously, on CS4410...

A really small VA space

- 8 bits! How large is the address space?
- Consider the following "flat" address: 10000011 (131 in decimal)

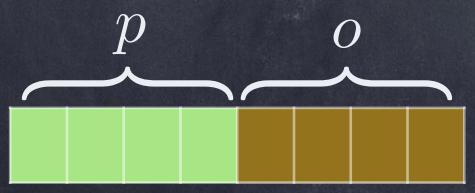
A really small VA space

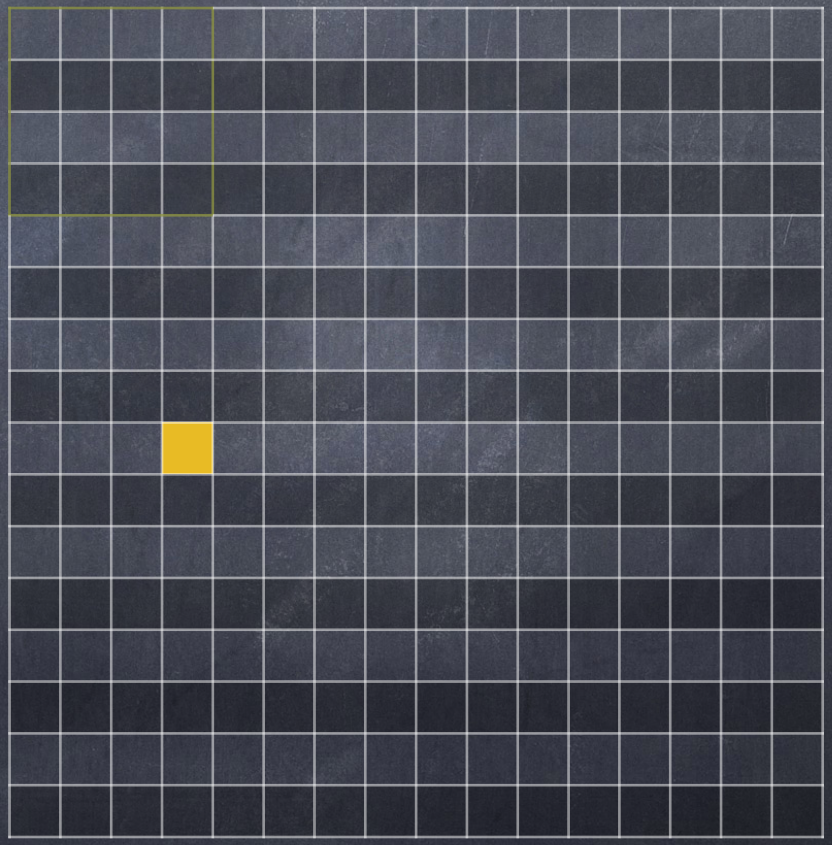
- 8 bits! How large is the address space?
- Consider the following "flat" address: 10000011 (131 in decimal)



A really small VA space

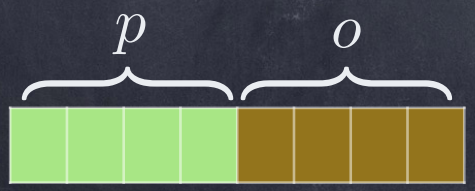
- 8 bits! How large is the address space?
- Consider the following "flat" address: 10000011 (131 in decimal)
- Now, let's consider the address through the lens of paging





A really small VA space

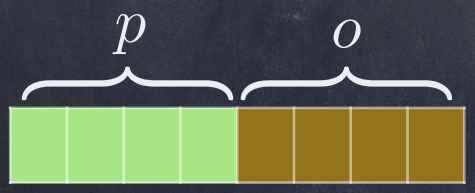
- 8 bits! How large is the address space?
- Consider the following "flat" address: 10000011 (131 in decimal)
- Now, let's consider the address through the lens of paging

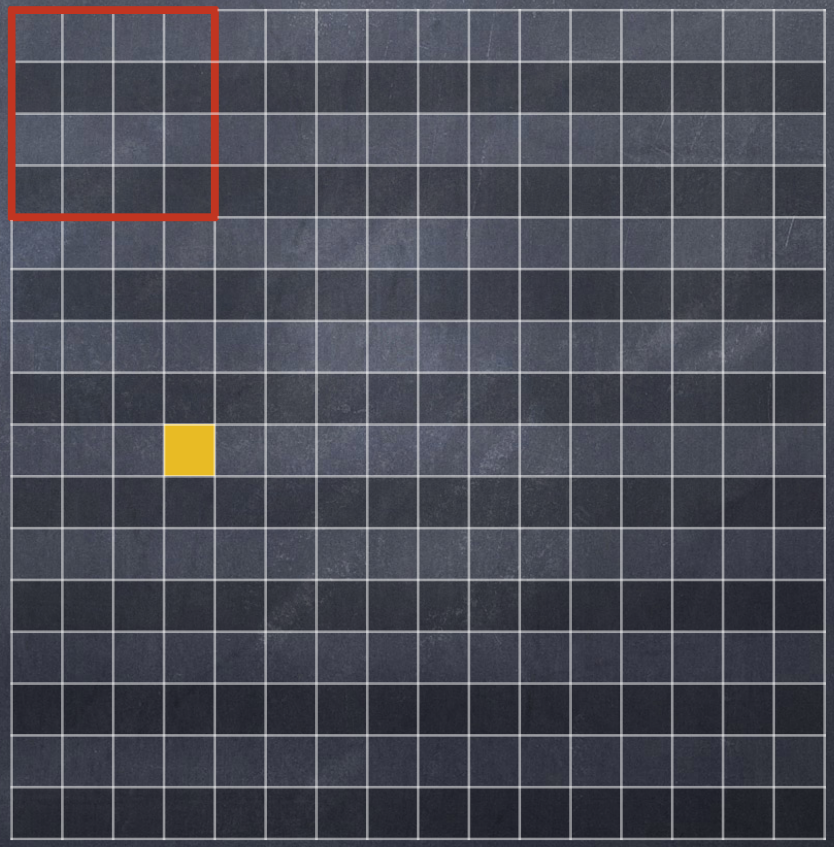


How many pages? How large is each page?

A really small VA space

- 8 bits! How large is the address space?
- Consider the following "flat" address: 10000011 (131 in decimal)
- Now, let's consider the address through the lens of paging

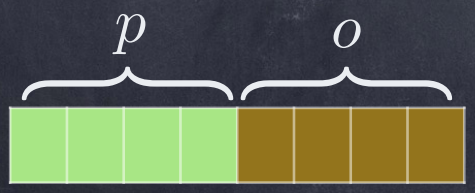


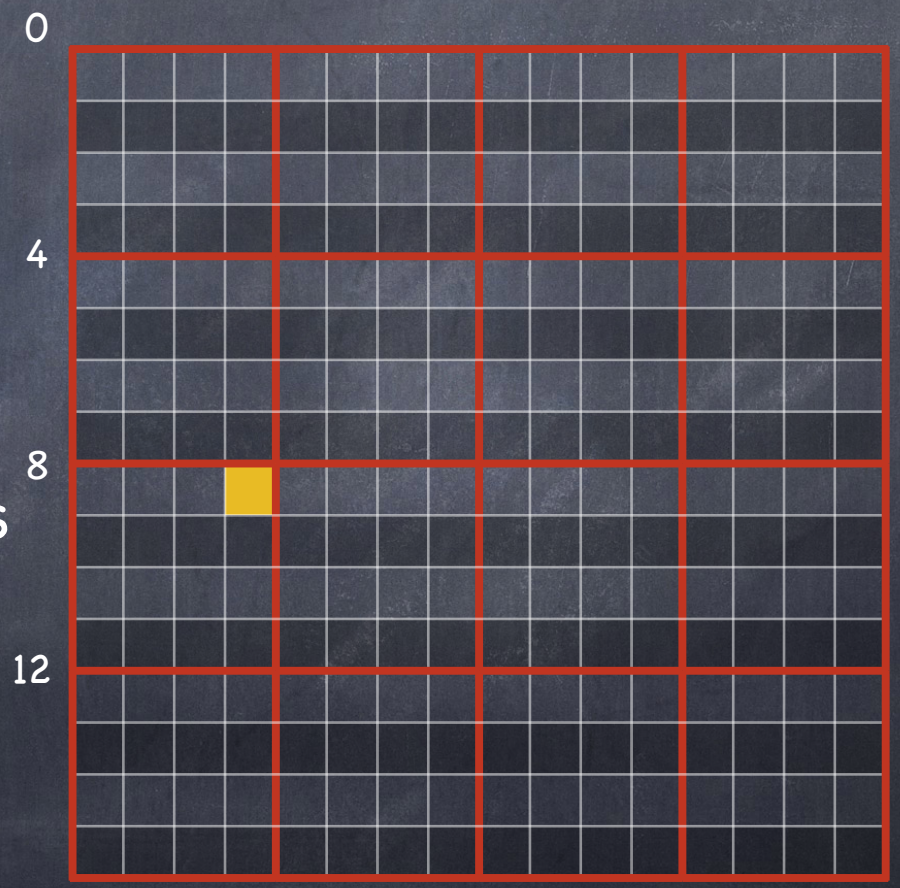


How many pages? How large is each page?

A really small VA space

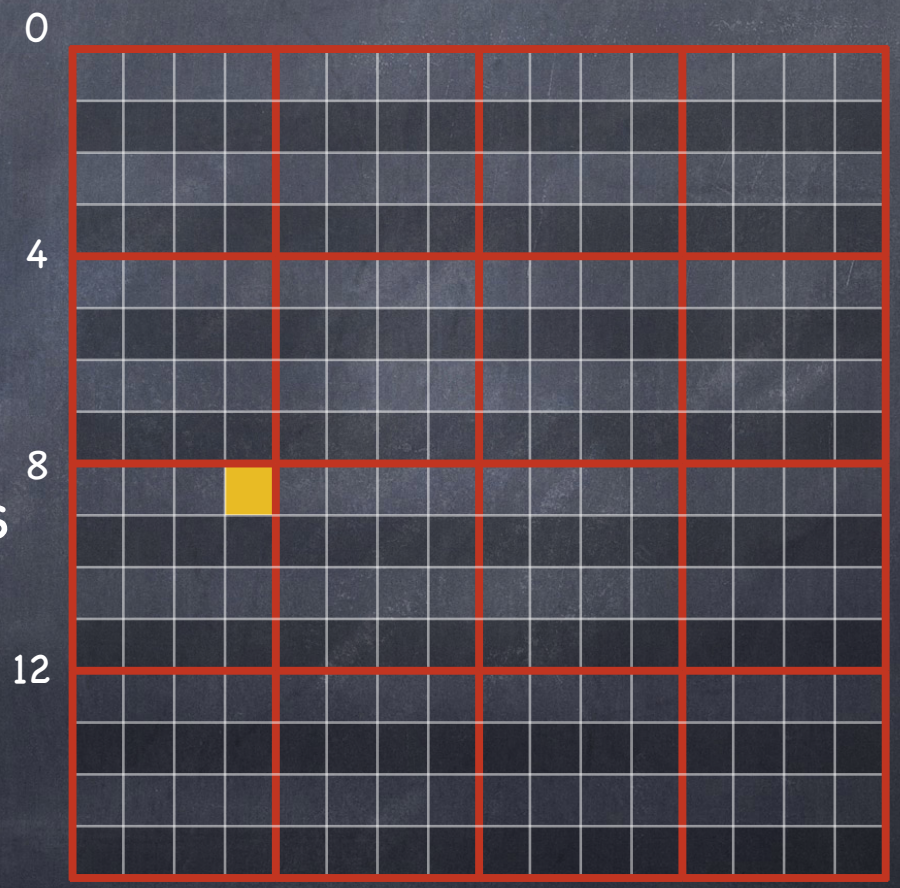
- 8 bits! How large is the address space?
- Consider the following "flat" address: 10000011 (131 in decimal)
- Now, let's consider the address through the lens of paging





A really small VA space

- 8 bits! How large is the address space?
- Consider the following "flat" address: 10000011 (131 in decimal)
- Now, let's consider the address through the lens of paging



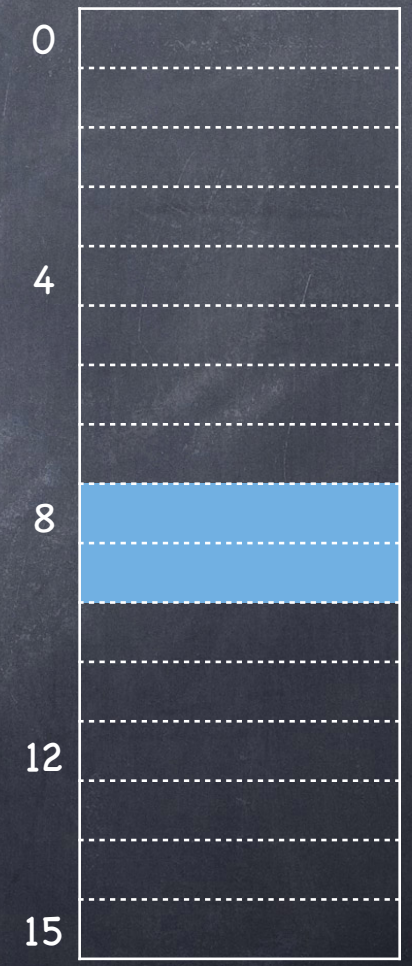
Still finding the same byte! (pheeeewww....)

A really small VA space

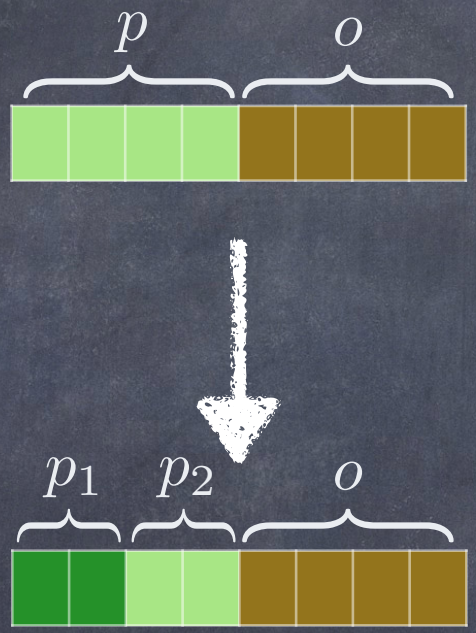
- What about the page table?
  - 16 entries
- Say each table entry takes 4 bytes
  - page table occupies 64 bytes (2 pages)
- Suppose process uses only pages 8 and 9

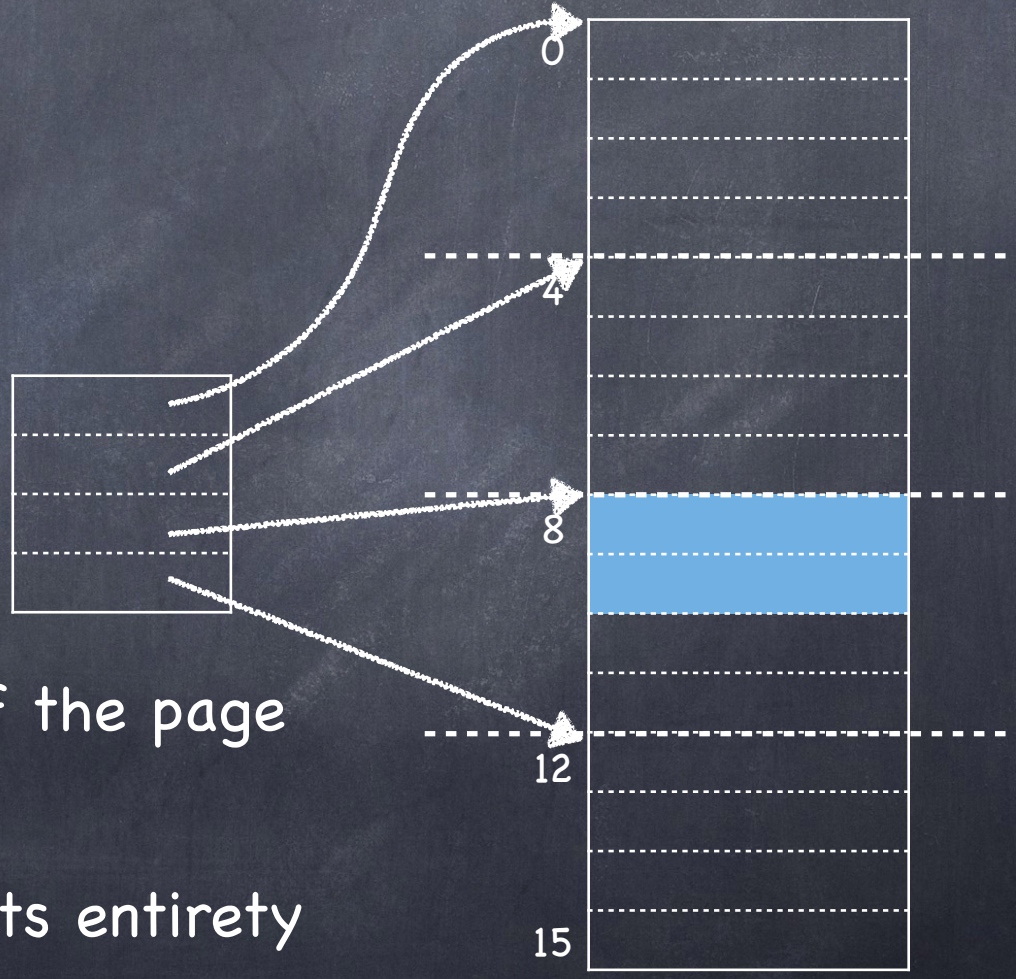
A really small VA space

- What about the page table?
  - 16 entries
- Say each table entry takes 4 bytes
  - page table occupies 64 bytes (2 pages)
- Suppose process uses only pages 8 and 9
  - can we be more efficient?



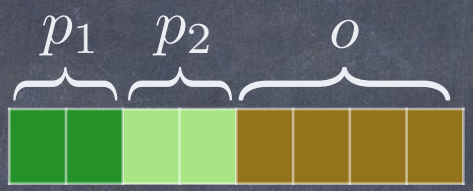
A really small VA space

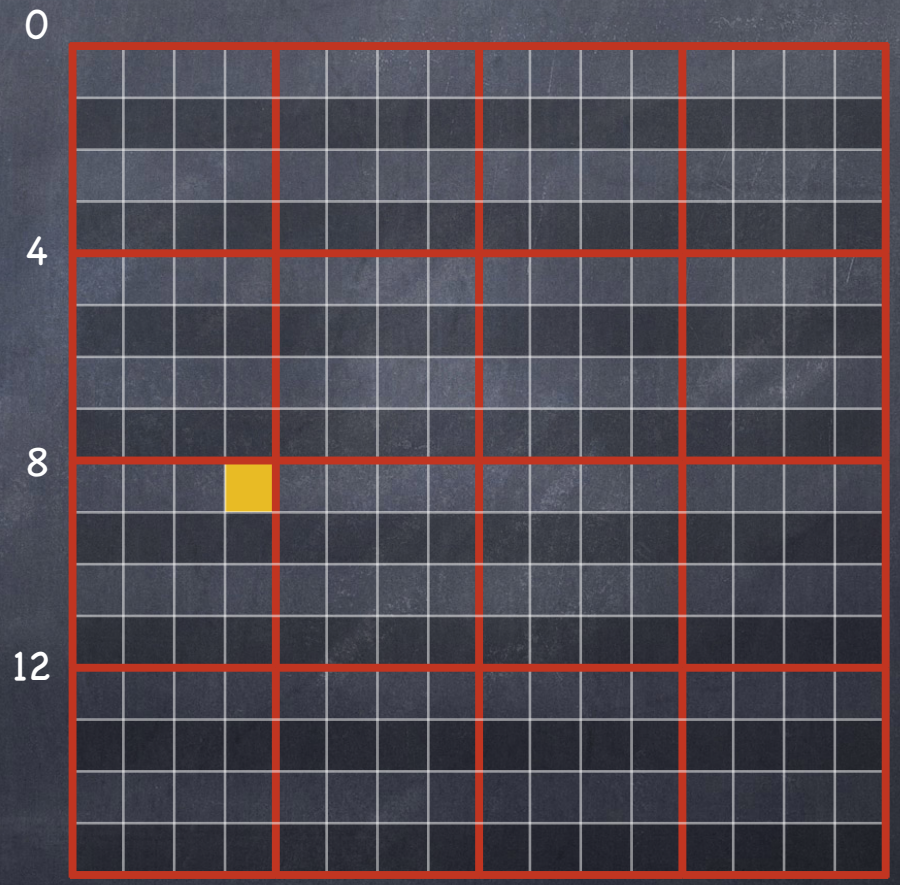




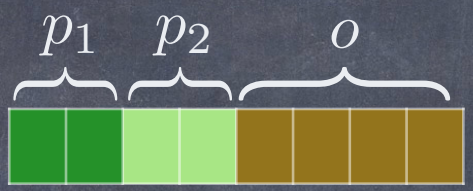
- Materialize only the portion of the page table that is needed
- Out index must be present in its entirety
  - in case the process needs more pages!

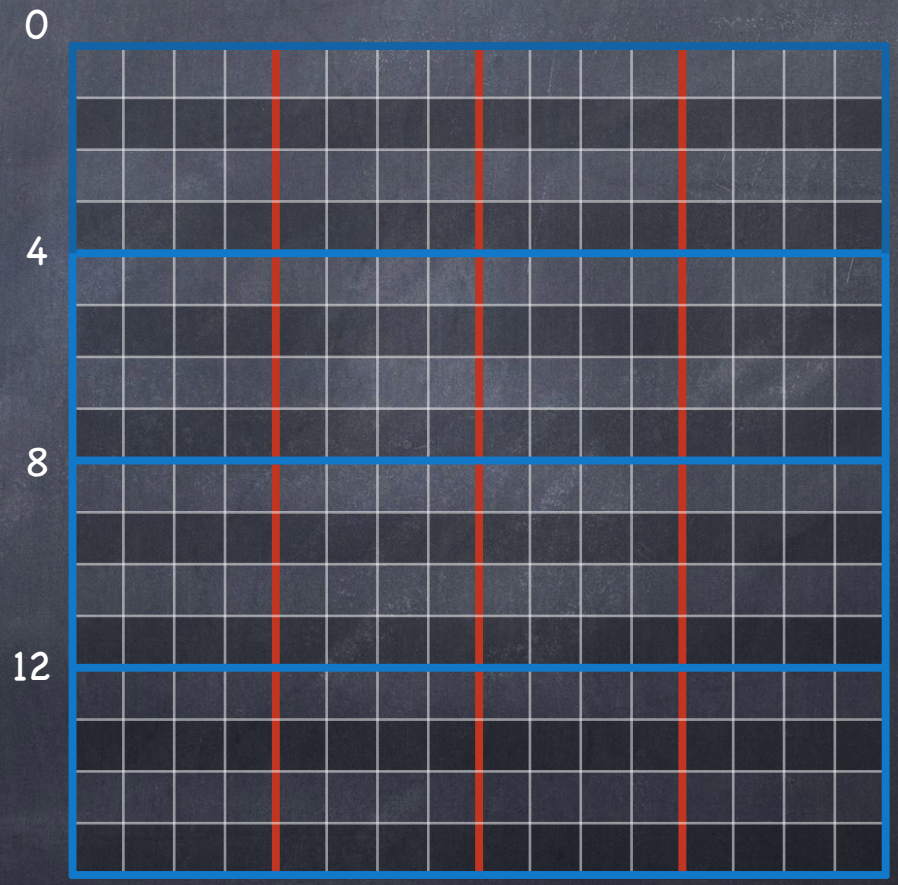
A really small VA space



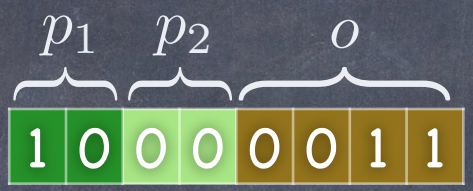


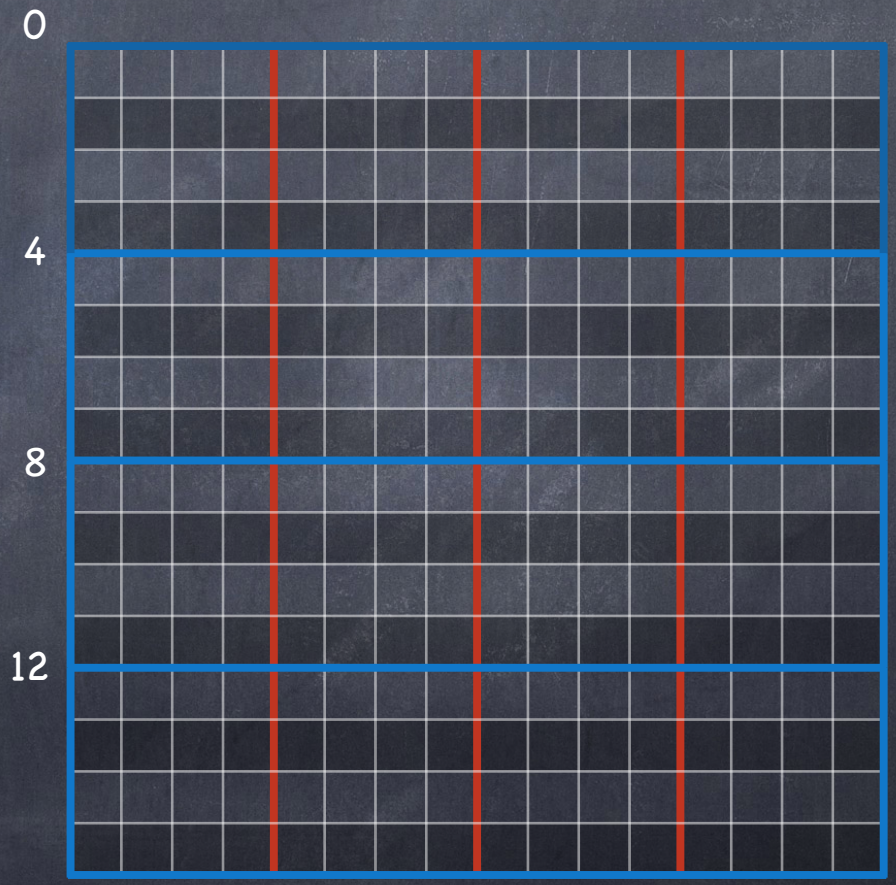
A really small VA space



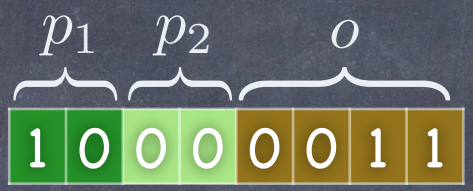


A really small VA space

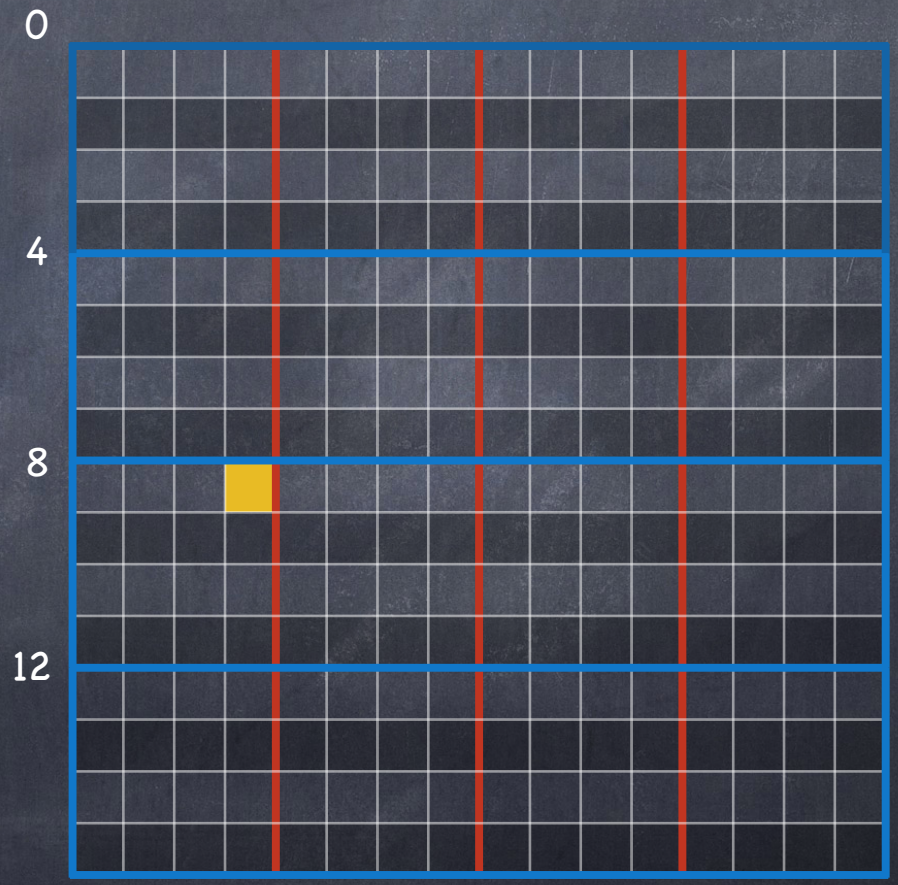




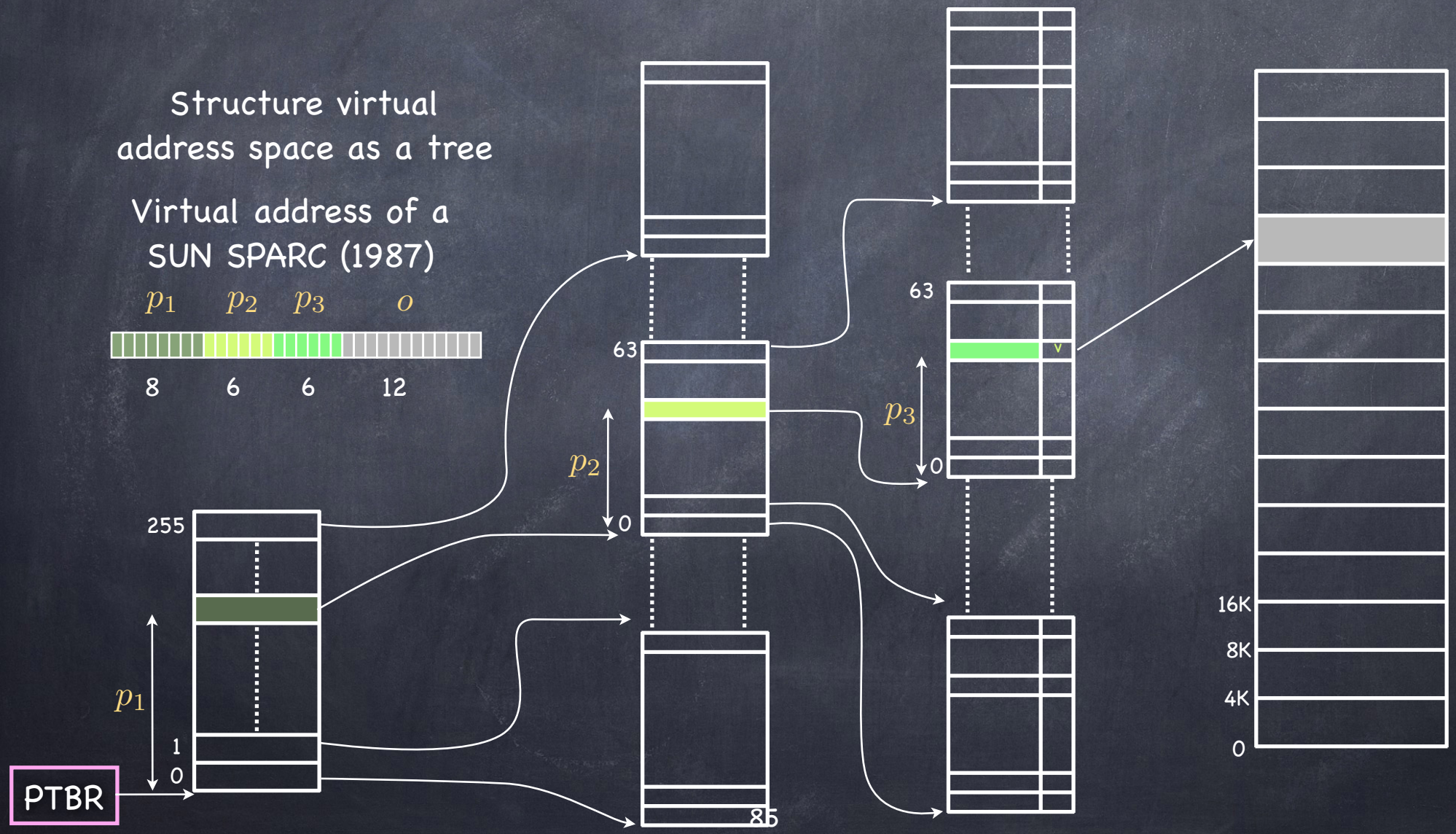
A really small VA space



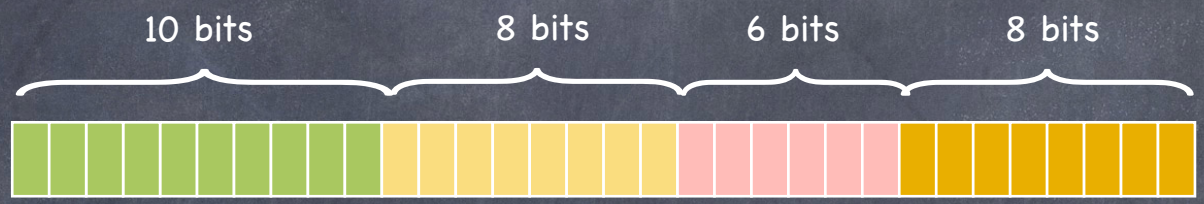
Still finding the same byte!
(pheeeewww….)



Multi-level Paging

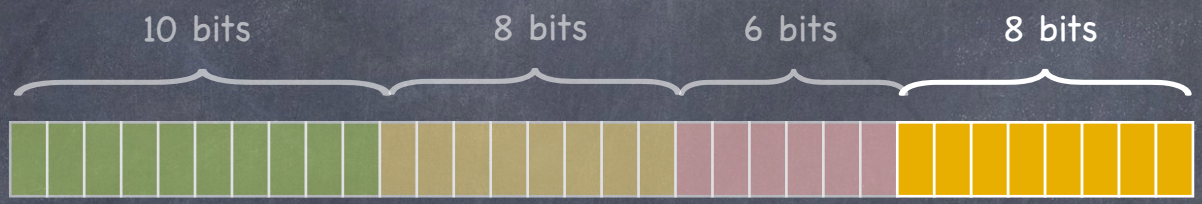


Example



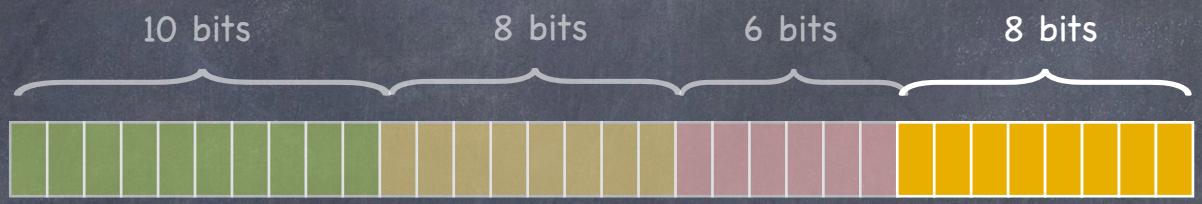
- What is the page size?

Example



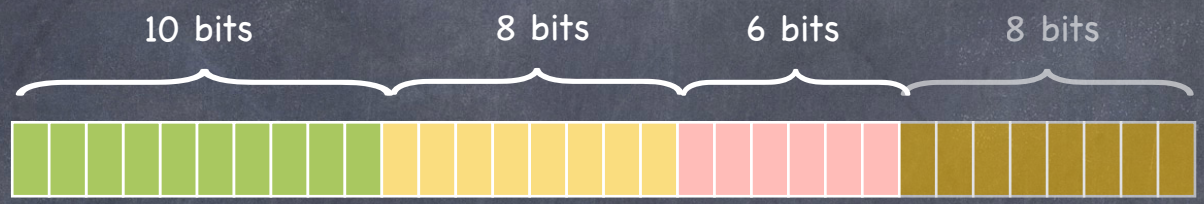
- What is the page size? Page size is 256 bytes (2^8)
- What is the Page Table size for a process that uses 256 contiguous KB of its VA space starting at address 0? [Assume each PTE is 2 bytes]
  - if we used a linear representation of the page table:

Example



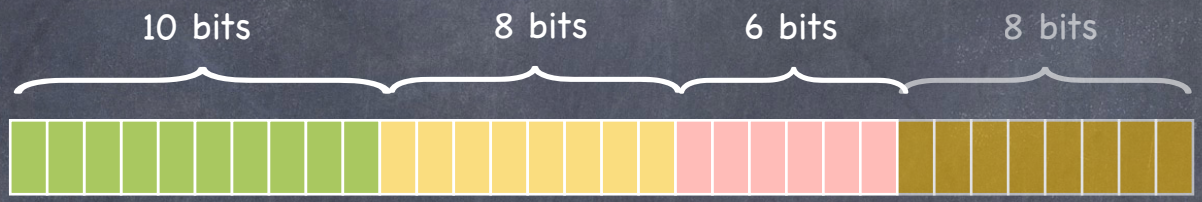
- What is the page size? Page size is 256 bytes (2^8)
- What is the Page Table size for a process that uses 256 contiguous KB of its VA space starting at address 0? [Assume each PTE is 2 bytes]
  - if we used a linear representation of the page table:
    - Page Table has 2^{24} entries

Example

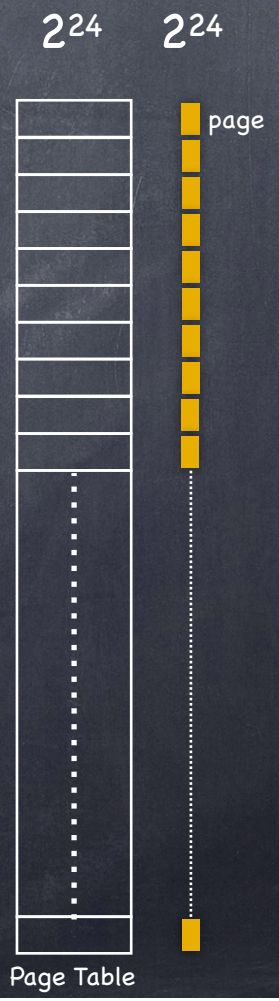


- What is the page size? Page size is 256 bytes (2^8)
- What is the Page Table size for a process that uses 256 contiguous KB of its VAS starting at address 0? [Assume each PTE is 2 bytes]
  - if we used a linear representation of the page table:
    - Page Table has 2^{24} entries
    - PT Size: 2^{24}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

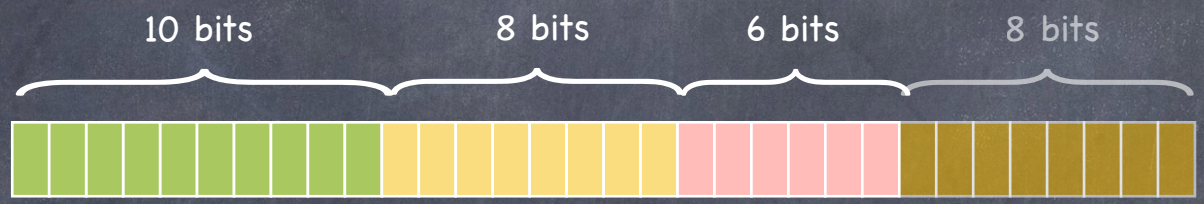
Example



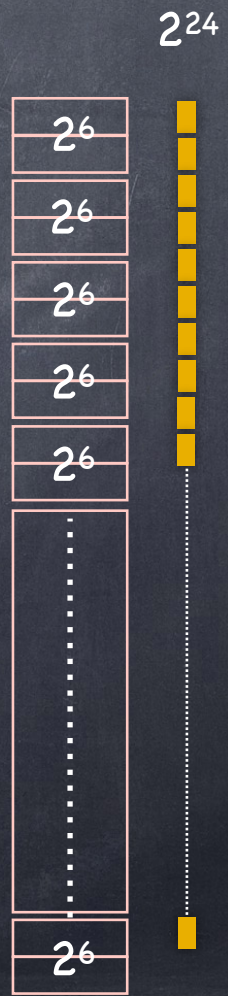
- What if we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries



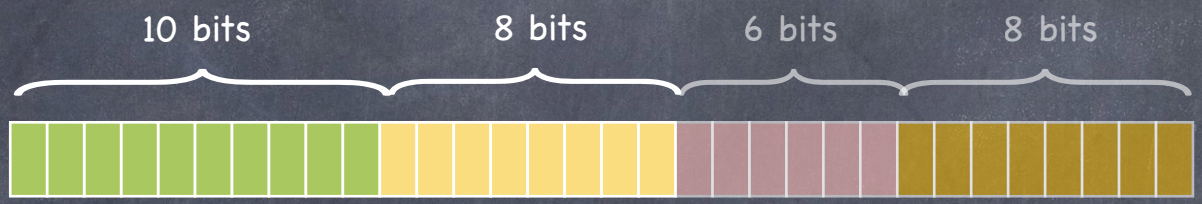
Example



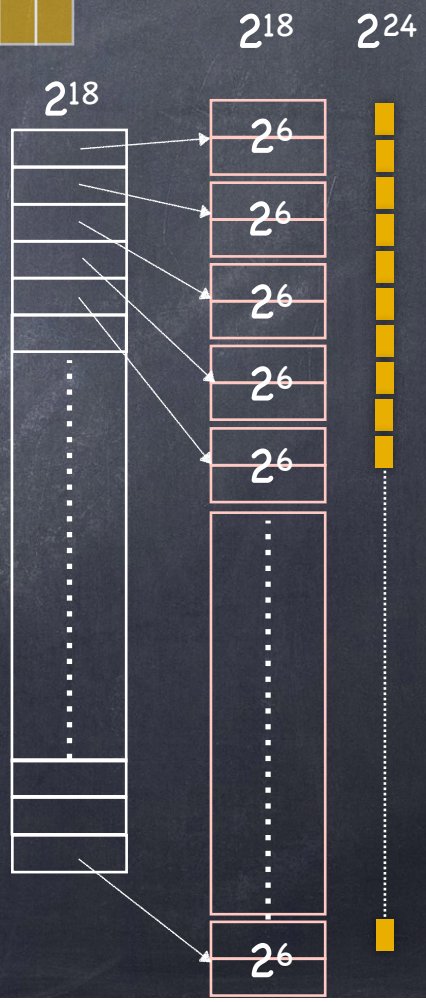
- What is we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries. How many such chunks?



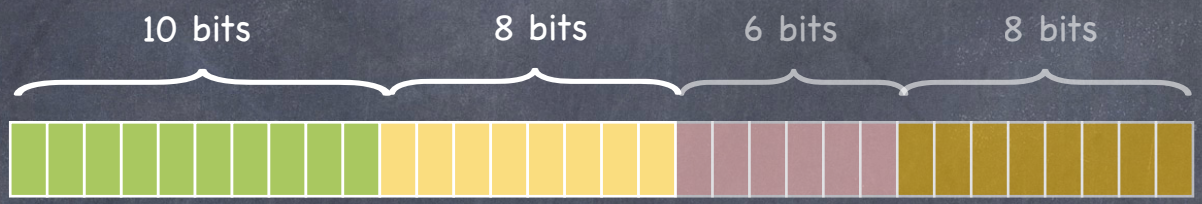
Example



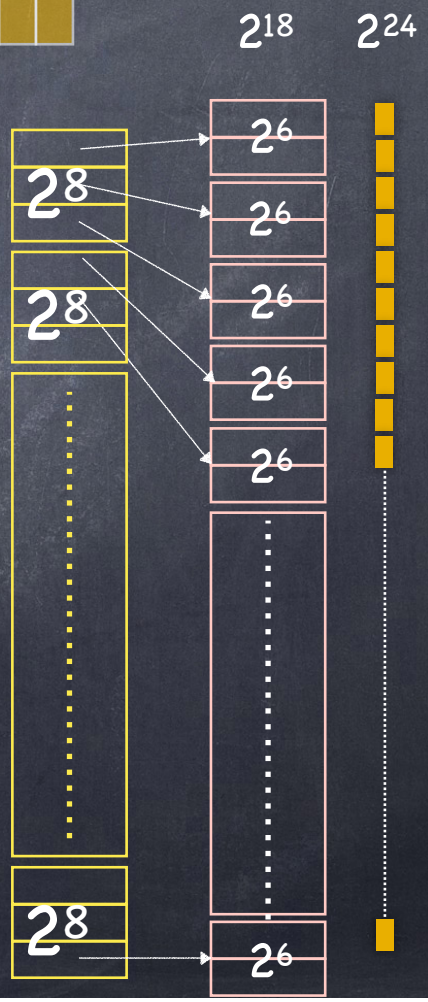
- What is we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries
  - we'll need an index with 2^{18} entries...
  - ...which we'll partition in chunks of 2^8 entries



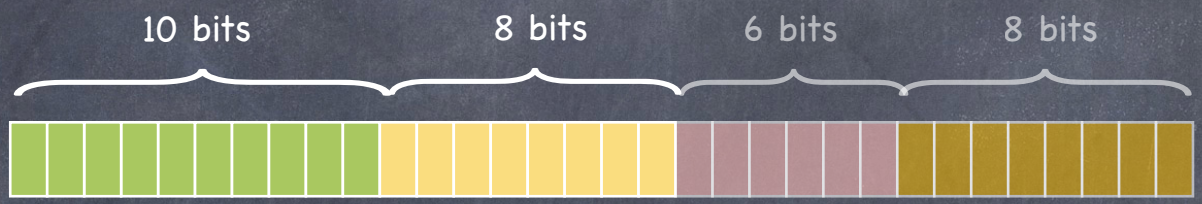
Example



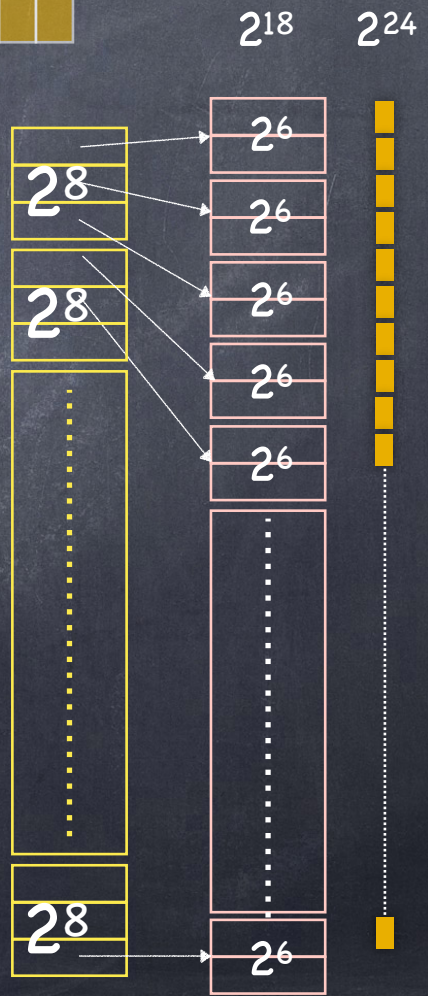
- What is we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries
  - we'll need an index with 2^{18} entries...
  - ...which we'll partition in chunks of 2^8 entries. How many such chunks?



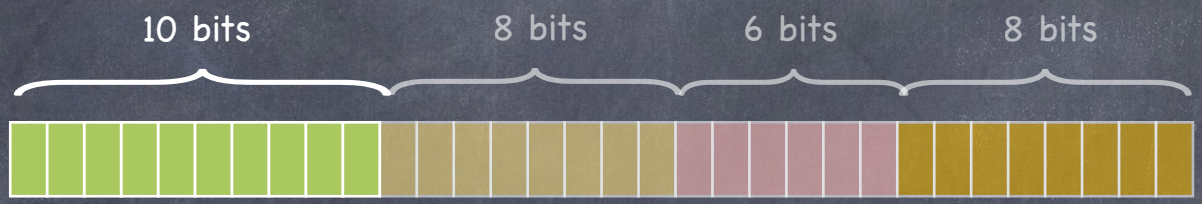
Example



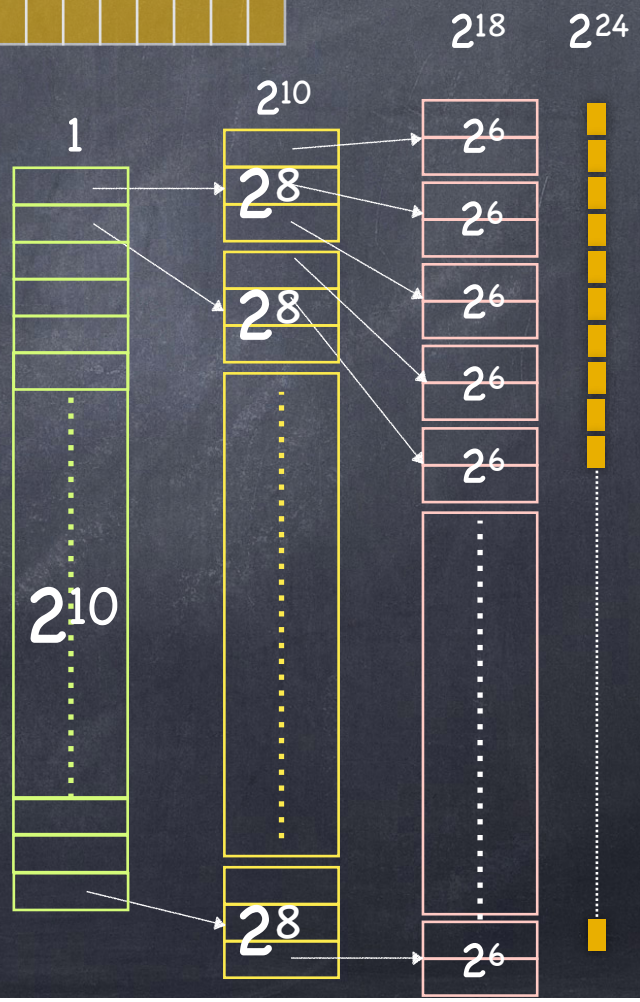
- What is we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries
  - we'll need an index with 2^{18} entries...
  - ...which we'll partition in chunks of 2^8 entries
  - We'll need an index of 2^{10}



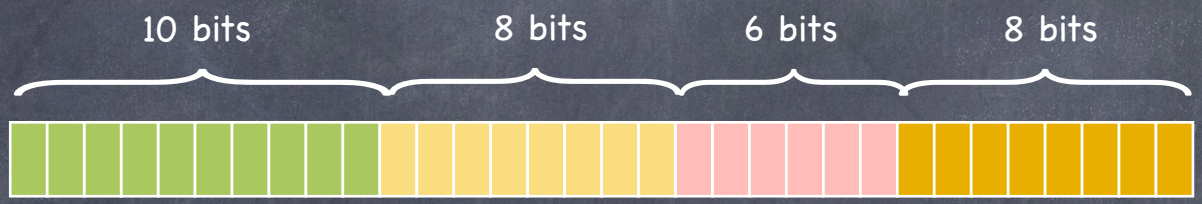
Example



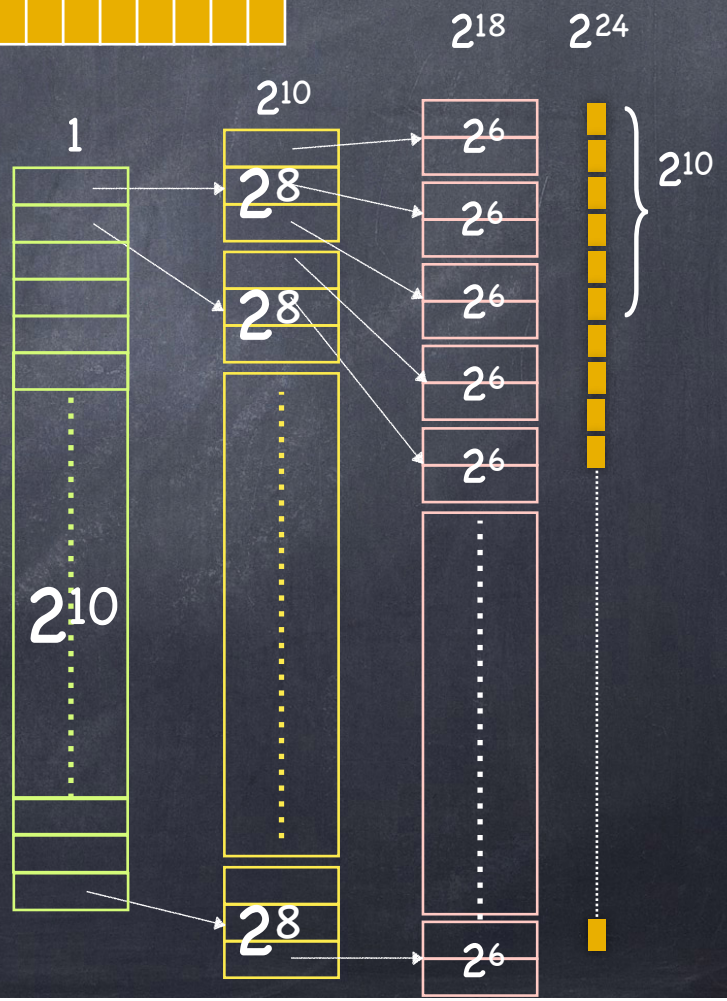
- What is we use a tree?
  - We still need to account for 2^{24} pages...
  - ...but we are going to partition the PT in a sequence of chunks, each with 2^6 entries
  - we'll need an index with 2^{18} entries...
  - ...which we'll partition in chunks of 2^8 entries
  - We'll need an index of 2^{10}



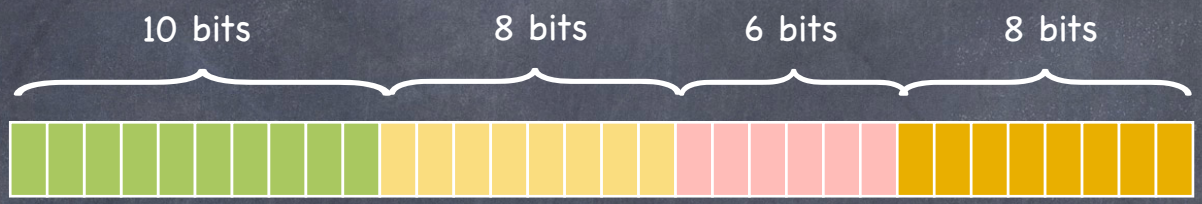
Example



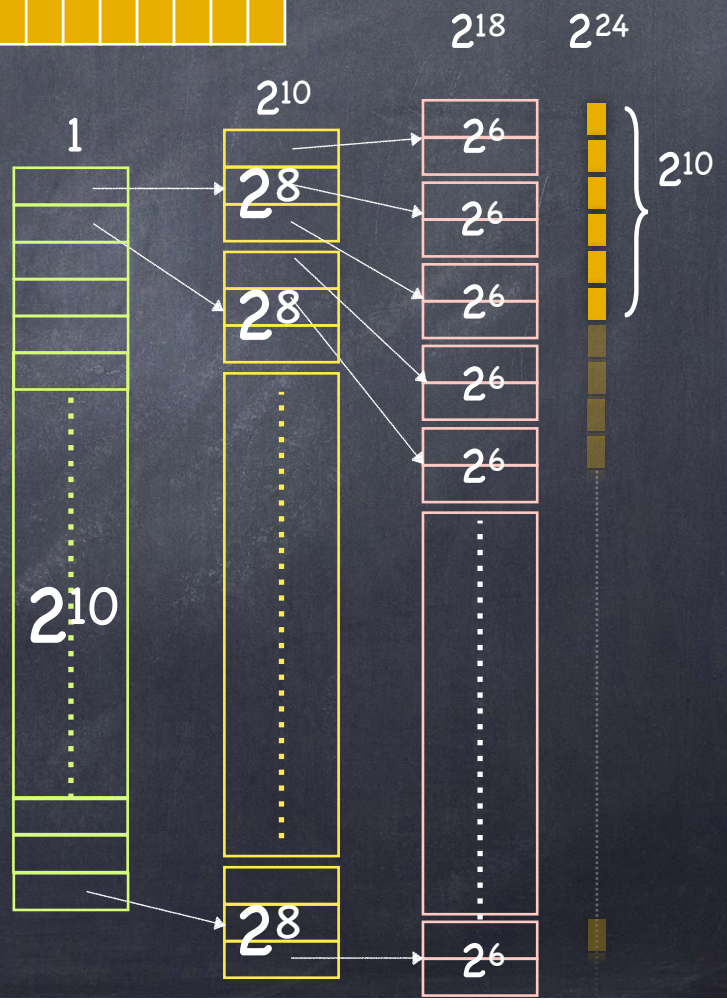
- Are we better off?
  - The number of PT entries now is $(2^6 \times 2^{18}) + (2^{10} \times 2^8) + 2^{10} > 2^{24}$!!
  - But we only need the portion of the tree needed to map the first 1K (2^{10}) pages!



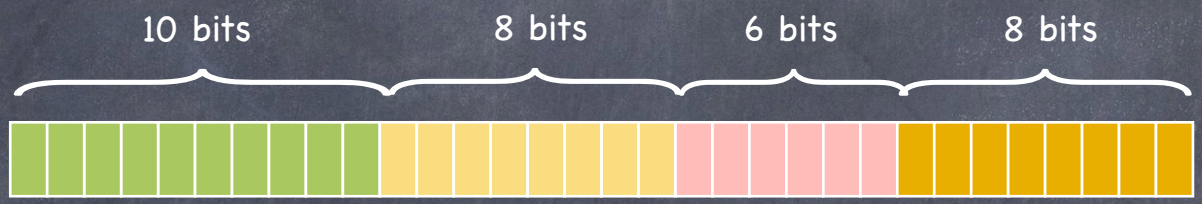
Example



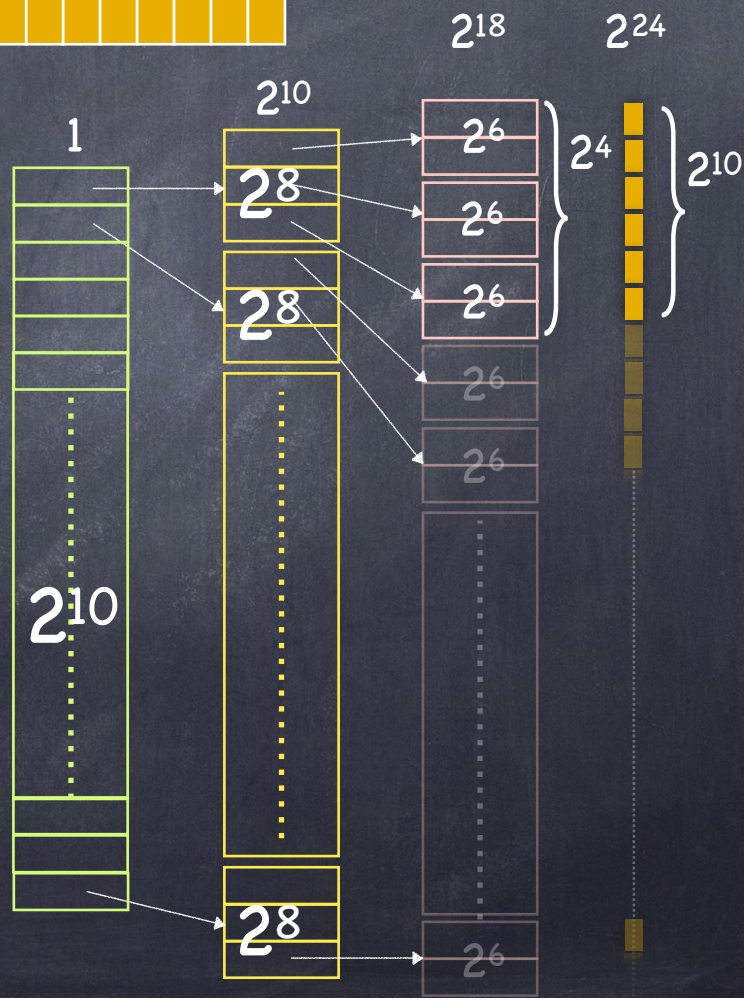
- How many chunks of size 2^6 are needed to hold 2^{10} PTEs of consecutive pages starting at 0?
  - $2^{10}/2^6 = 2^4 = 16$



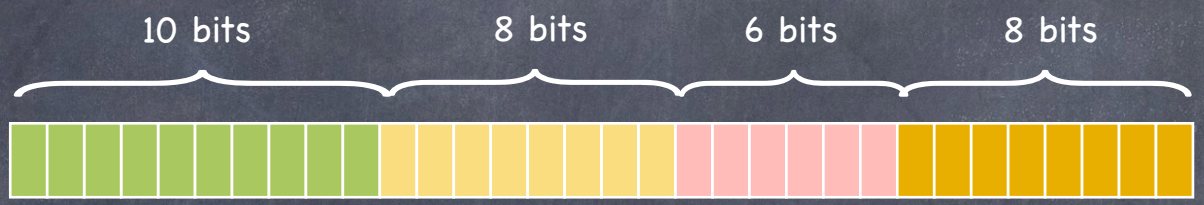
Example



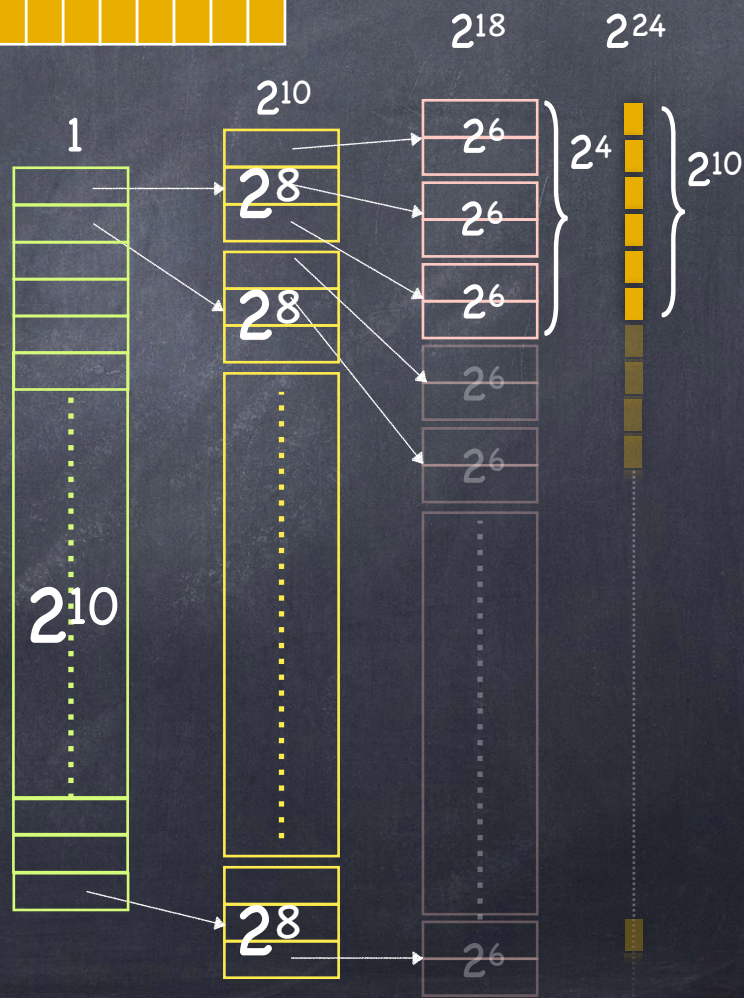
- How many chunks of size 2^6 are needed to hold 2^{10} PTEs of consecutive pages starting at 0?
  - $2^{10}/2^6 = 2^4 = 16$



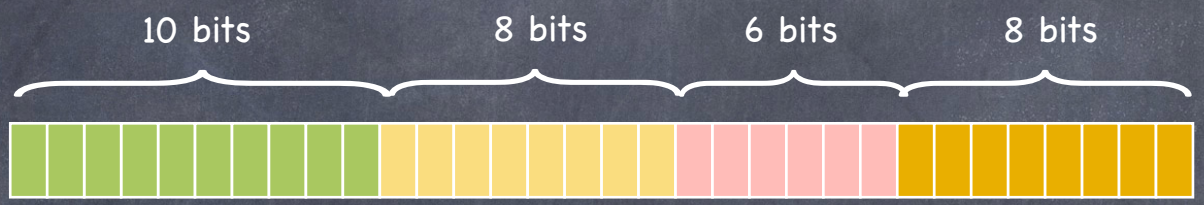
Example



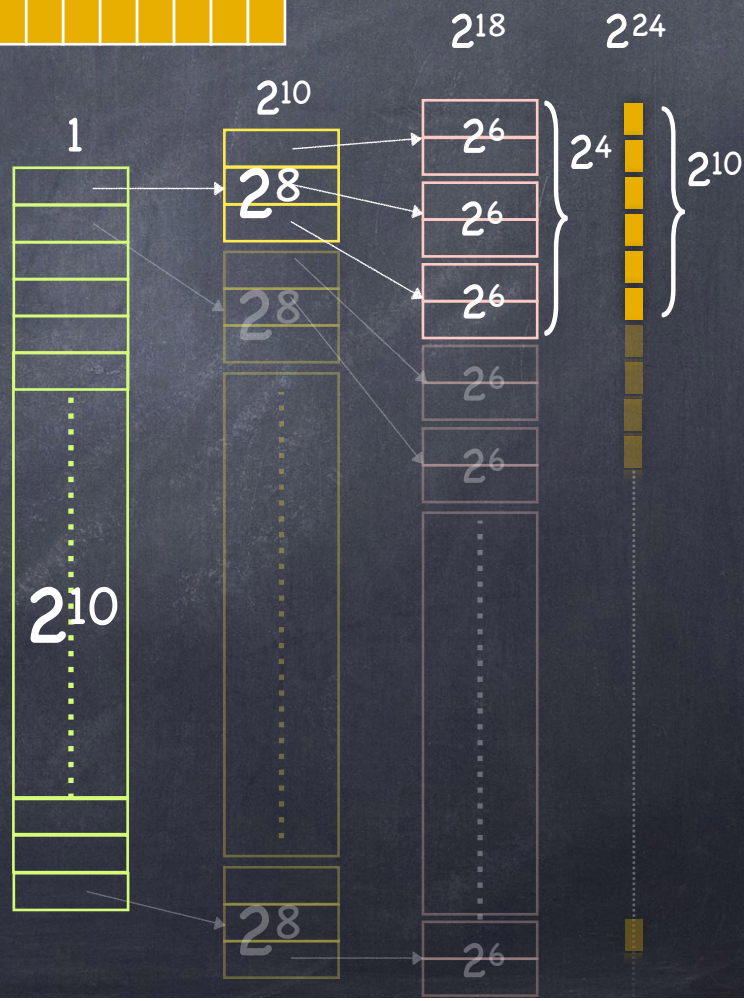
- How many chunks of size 2^6 are needed to hold 2^{10} PTEs of consecutive pages starting at 0?
  - $2^{10}/2^6 = 2^4 = 16$
- How many chunks of size 2^8 are needed to hold pointers to 16 pink chunks?
  - 1



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 x (1 x 1024 + 1 x 256 + 16 x 64) = 4608 bytes



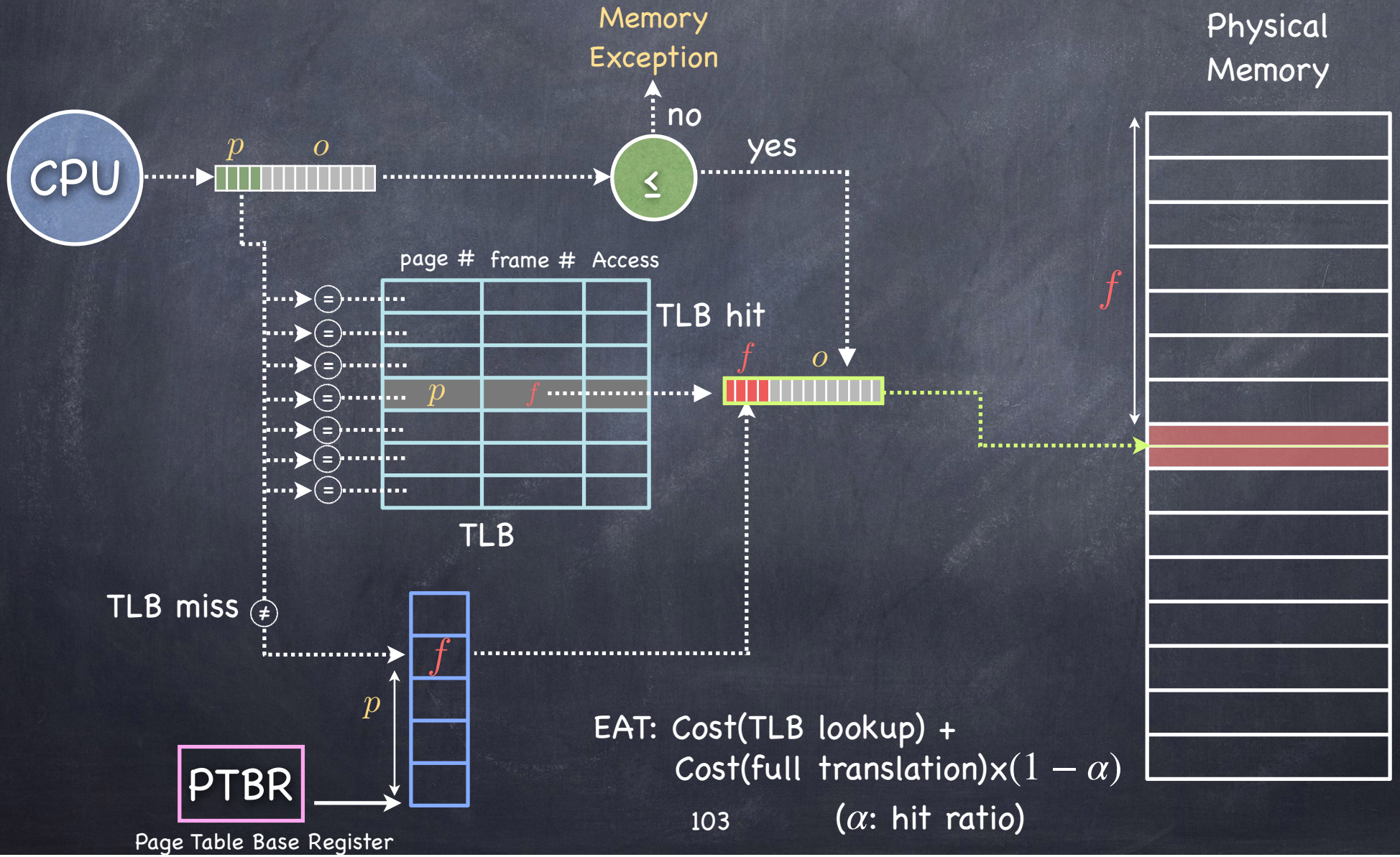
Getting slooooower

- Every new level of paging
  - reduces the memory overhead for computing the mapping function...
  - ... but increases the time necessary to perform the mapping function

Caching!

- Keep the results of recent VA-PA translations in a structure called Translation Lookaside Buffer (TLB)
  - TLB is a cache for page-to-frame mappings

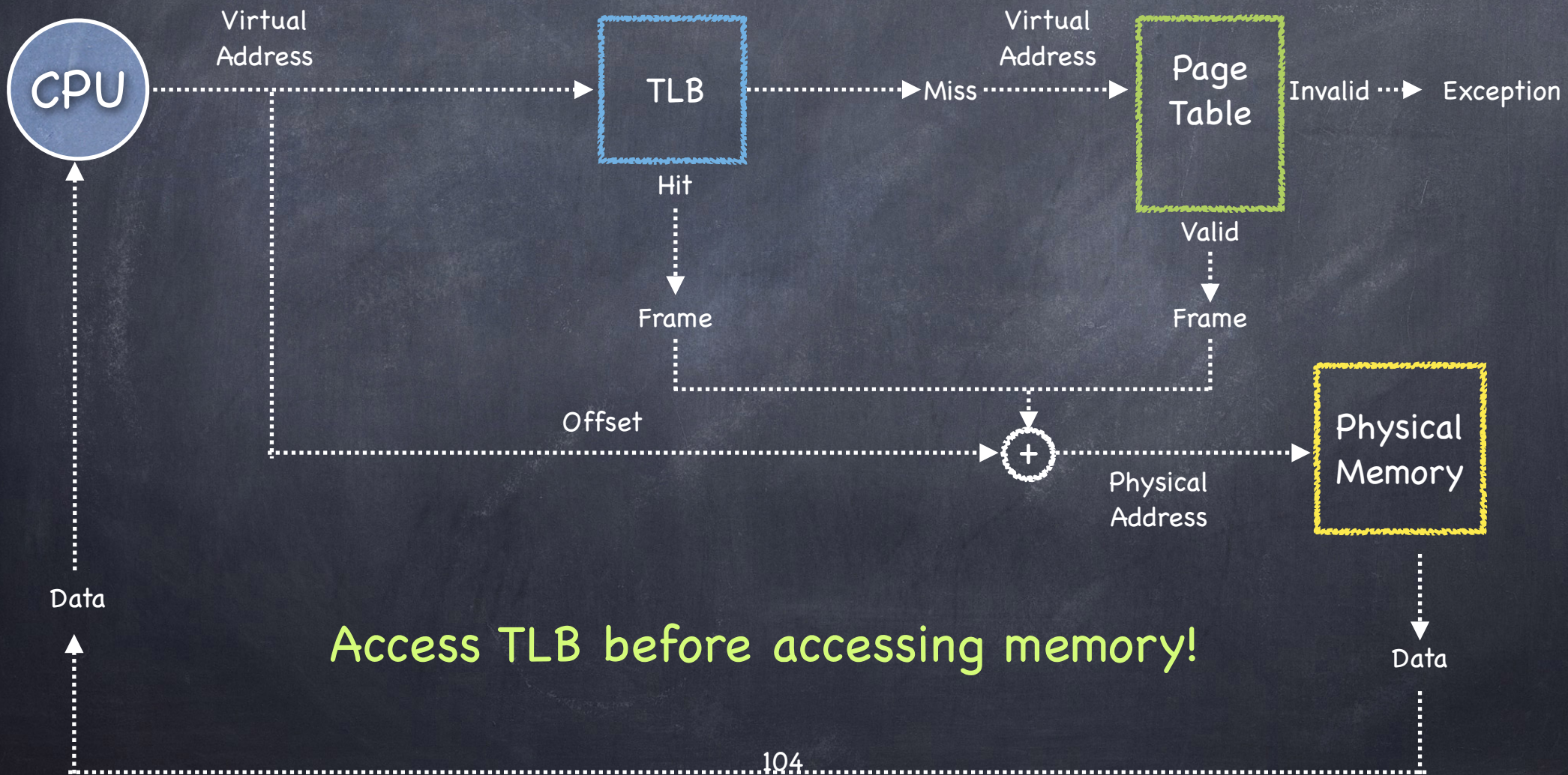
Speeding things up: The TLB



EAT: Cost(TLB lookup) + Cost(full translation)x$(1-\alpha)$

(α: hit ratio)

Address Translation with TLB



Access TLB before accessing memory!

Hit and Miss

- The TLB is small; it cannot hold all PTEs
  - it can be fast only if it is small!
  - Some translations will inevitably miss the TLB
  - Must access memory to find the appropriate PTE
    - called walking the page table
    - incurs large performance penalty

Handling TLB Misses: Hardware

- Hardware-managed (e.g., x86)
  - The hardware does the page walk
  - Hardware fetches PTE and inserts it in TLB
    - If TLB is full, must replace another TLB entry
  - Done transparently to system software

Handling TLB Misses: Software

- Software-managed (e.g., MIPS)
  - Hardware raises an exception, trap handler runs in kernel
  - Handler does the page walk, fetches PTE, and inserts/evicts entries in TLB
  - Handler must return to the same instruction that caused the trap!
  - Careful not to generate a TLB miss while running the handler!

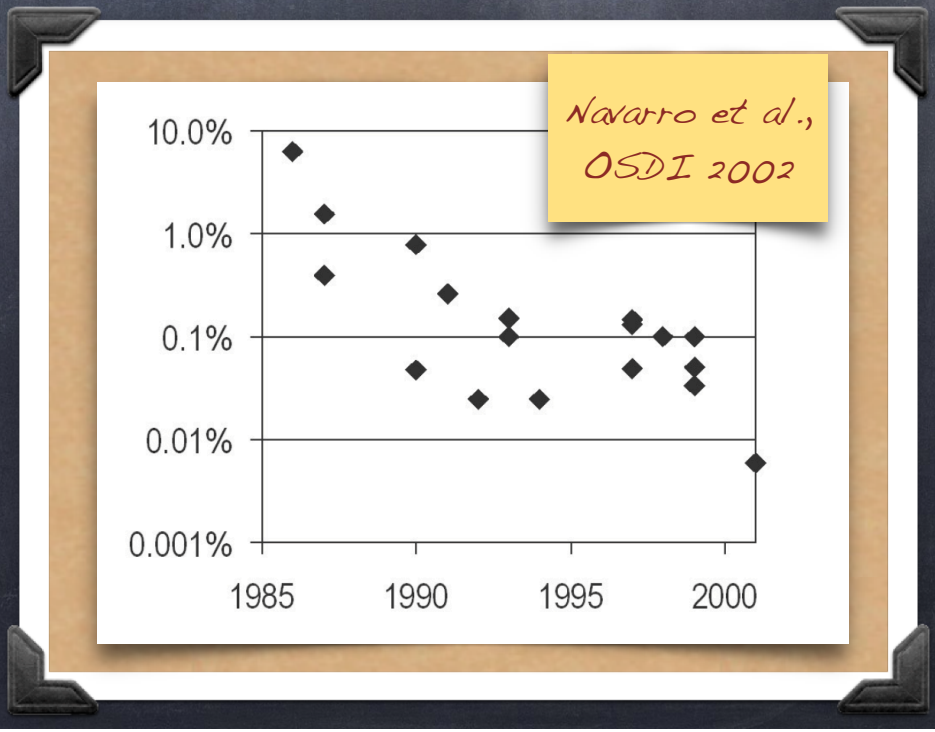
Tradeoffs, Tradeoffs...

- Hardware-managed TLB
  - \+ No exception on TLB miss. Instruction just stalls
  - \+ No extra instruction/data brought into the cache
  - – OS has no flexibility in deciding Page Table: hardware must know location and format of PTEs
- Software-managed TLB
  - \+ OS can define Page Table organization
  - \+ More flexible TLB entry replacement policies
  - – Slower: exception causes to flush pipeline; execute handler; pollute cache

TLB Coverage

- What fraction of memory can be accessed without TLB misses?
  - low TLB coverage can result in a large number of memory references

1000-fold decrease in 15 years!



Superpages

- Wider TLB coverage by supporting page sizes that are multiples of the base page size: superpages
  - Pentium: 4KB base; 4MB Super
  - Itanium: 10 sizes, from 4KB (base) to 256 MB
- A set of contiguous base pages can be promoted to a superpage
- Demotion works the other way around

For more

Navarro et al.,
"Practical, transparent Operating systems support for superpages"

Tradeoffs, Tradeoffs...

+ Improved TLB coverage! but...
- Larger internal fragmentation
- External fragmentation (?)
  - superpage of N base pages
  - N free base frames free, but not contiguous
- Less efficient reading
- Coarser granularity for dirty, reference, and protection bits

TLB Consistency - I

- On context switch
  - VAs of old process should no longer be valid
  - Change PTBR — but what about the TLB?

TLB Consistency - I

- On context switch
  - VAs of old process should no longer be valid
  - Change PTBR — but what about the TLB?
    - Option 1: Flush the TLB
    - Option 2: Add pid tag to each TLB entry

	PID	VirtualPage	PageFrame	Access
TLB Entry	1	0x0053	0x0012	R/W

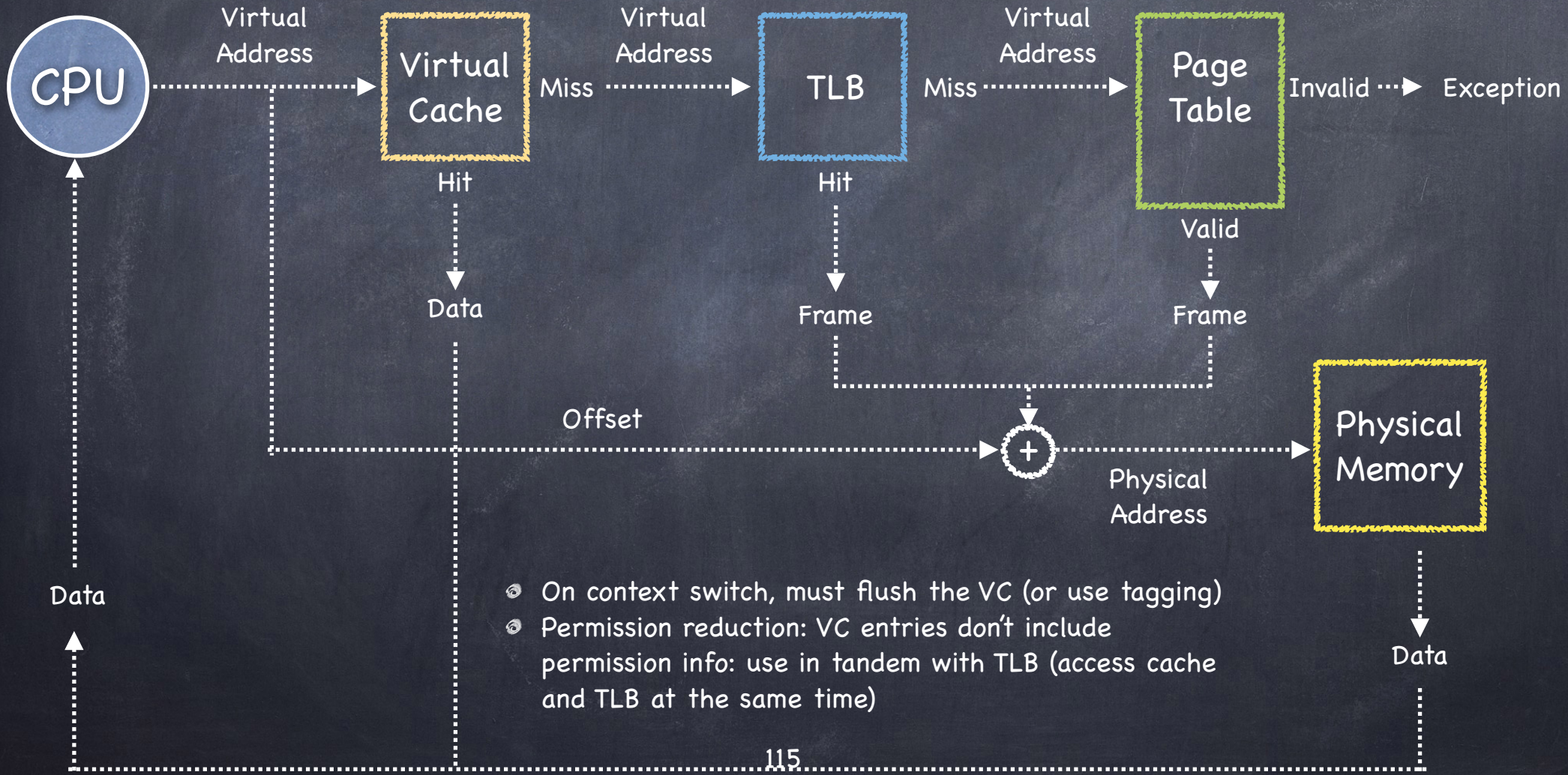
Ignore entries with wrong PIDs

TLB Consistency - II

- What if OS changes permissions on page?
  - If permissions are reduced, OS must ensure affected TLB entries are purged
    - e.g., on copy-on-write
  - If permissions are expanded, no problem
    - new permissions will cause an exception and hardware and OS will restore consistency

Virtually Addressed Caches

A copy of the contents of physical memory, indexed by the virtual address



- On context switch, must flush the VC (or use tagging)
- Permission reduction: VC entries don't include permission info: use in tandem with TLB (access cache and TLB at the same time)

Physically Addressed Caches