Page Table Issue (2) – Size

- 32-bit address space
- 4 kB page size

* Linear page tables are too big

|offset| = $\log_2$(4kB) = 12

|VPN| = 32 - 12 = 20 bits

entries in the page table = 2^{20}

5-bytes = |pte|

|page table| = 5 bytes * 2^{20} = 5MB per process

~100 processes active at once in a system

=> 500 MB for all page tables

Solving Size: Solution #1

Simple Solution: .. why not inc↑ the sz of a page?

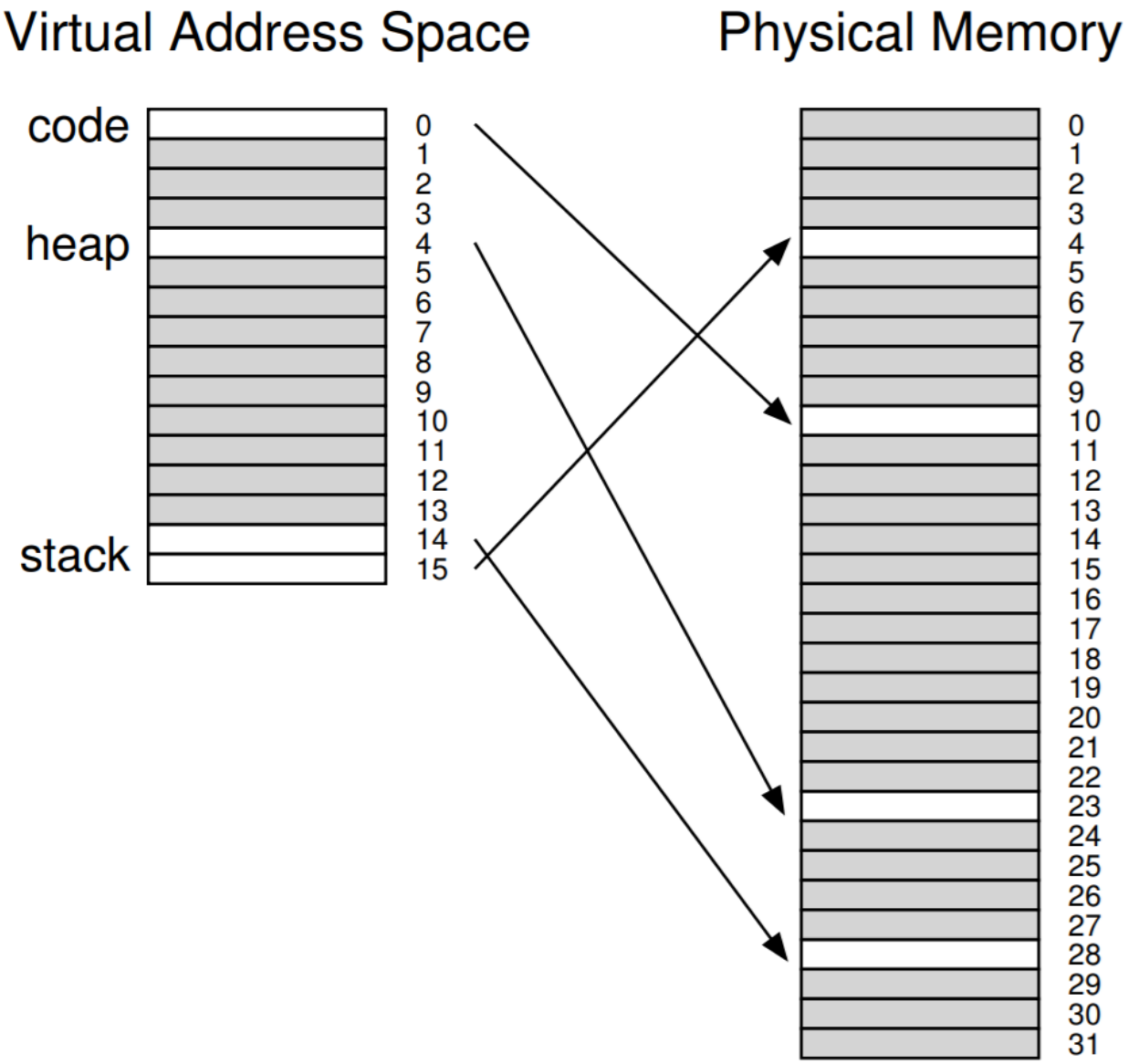
- Internal fragmentation

Solving Size: Solution #2 Motivation

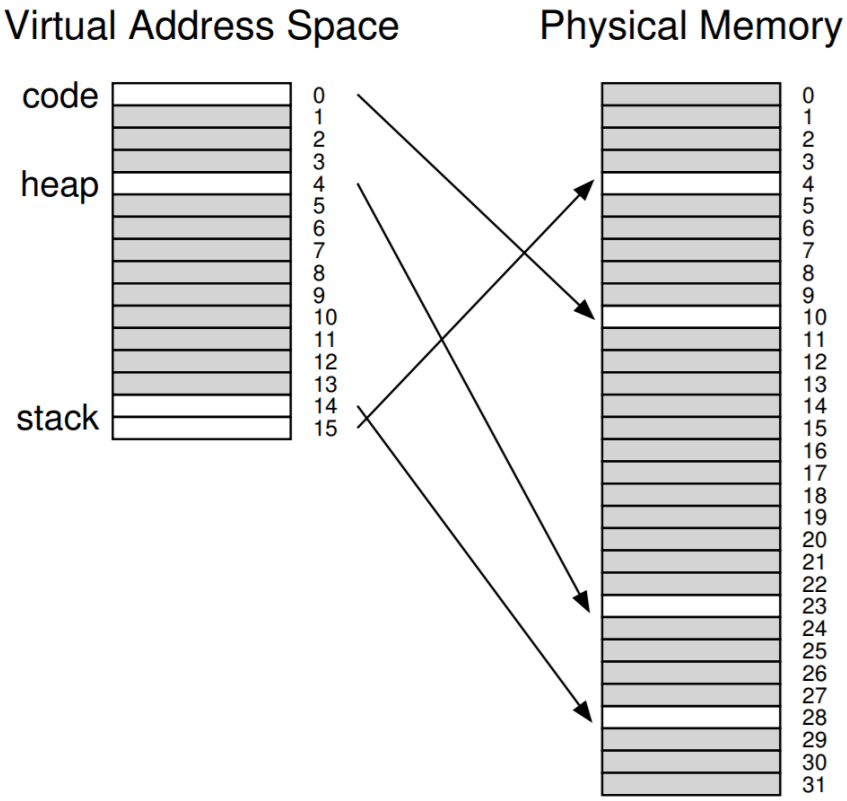
Consider:

- 16 kB address space
- 1 kB pages
- the stack and heap small

Solving Size: Solution #2 Motivation



Solving Size: Solution #2 Motivation



PFN	valid	prot
10	1	r-x
-	0	—
-	0	—
-	0	—
23	1	rw-
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
28	1	rw-
4	1	rw-

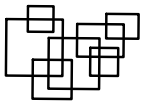
a sparse address space leads to lots of invalid entries in the page table

wasted space

Solving Size: Solution #2

Combine paging *with* segmentation (segmented paging)

1. Break up a process into different sized segments
2. Break up segments into fixed sized pages
   - e/a seg has its own page table
   - all entries in a page table are valid b/c we hv variable length page tables



seg table ↗? stores location of page table

Solution #2 Visual

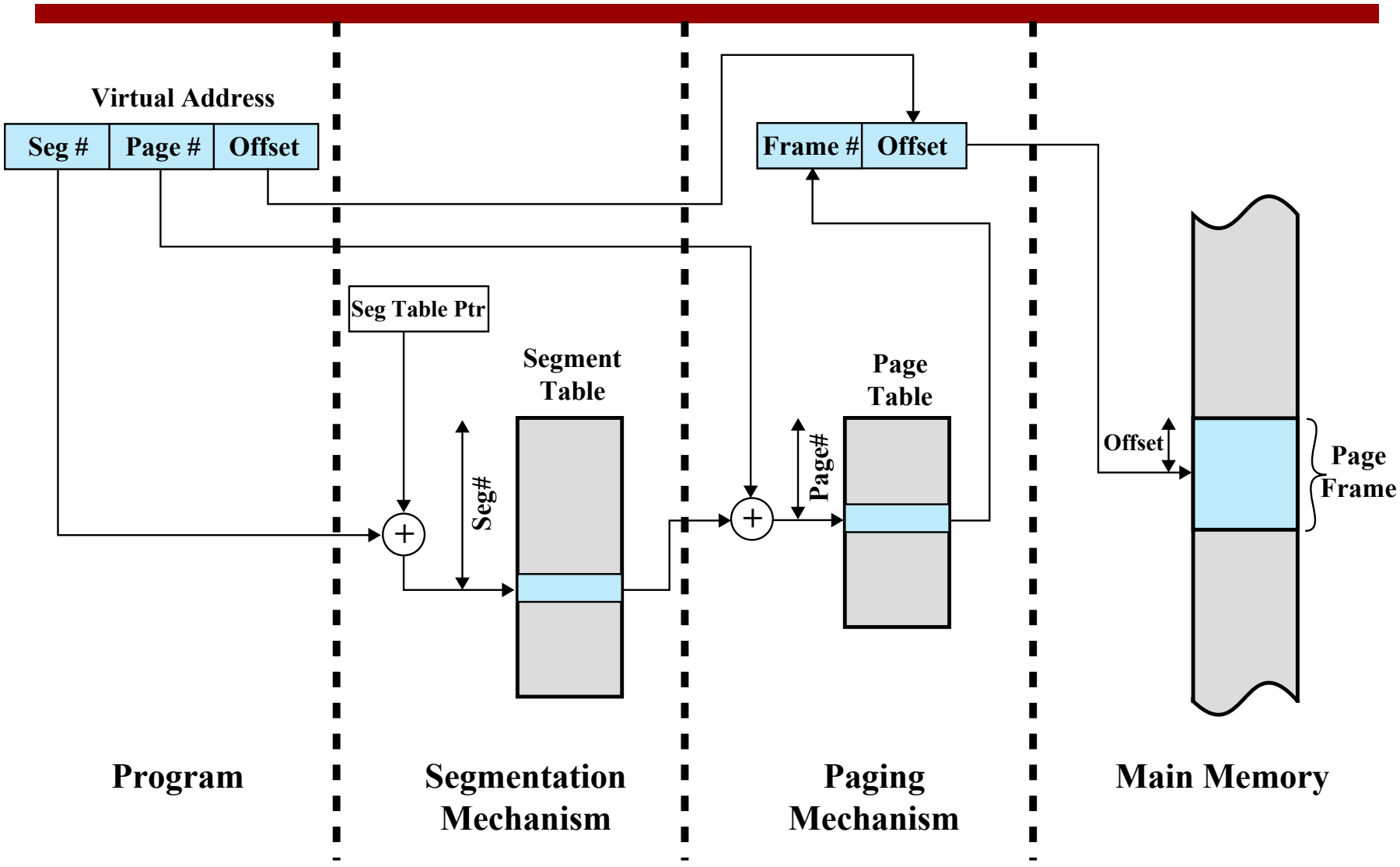


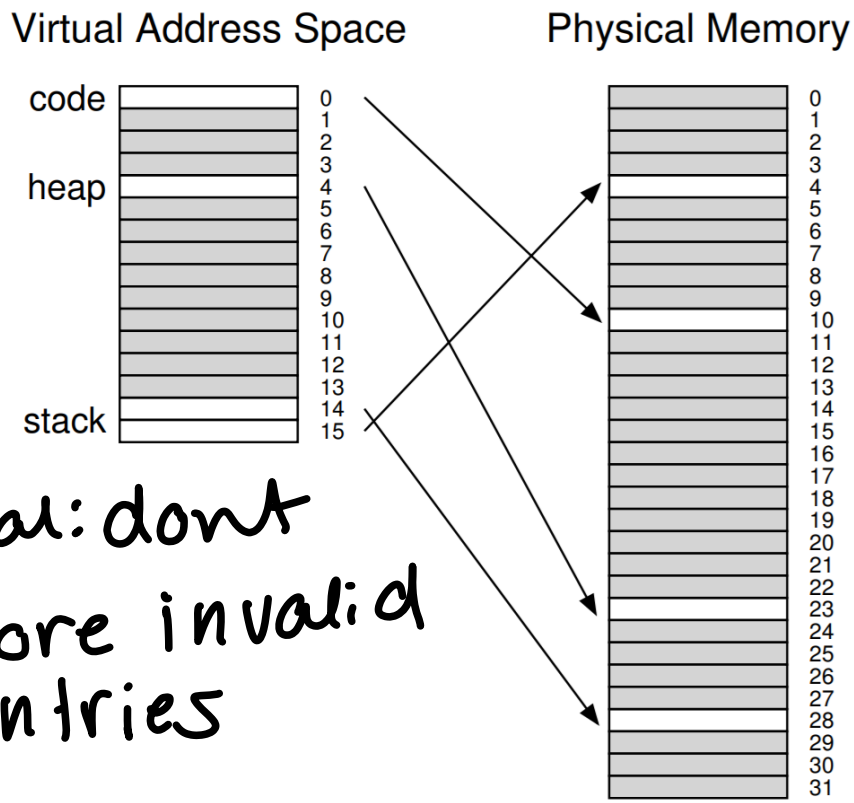
Figure 8.12 Address Translation in a Segmentation/Paging System

Solution #2 Analysis

+space efficiency - page tables only store valid entries

- External fragmentation - due to variable length page tables

Solution #3: Motivation



goal: dont store invalid entries

PFN	valid	prot
10	1	r-x
-	0	—
-	0	—
-	0	—
23	1	rw-
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
-	0	—
28	1	rw-
4	1	rw-

Solution #3

Main Idea: **Multi-level paging**

1. Break up the page table into fixed sized chunks
2. If all entries in a chunk are invalid, don't allocate the chunk

$\Longrightarrow$ store in page directory

High-Level Example

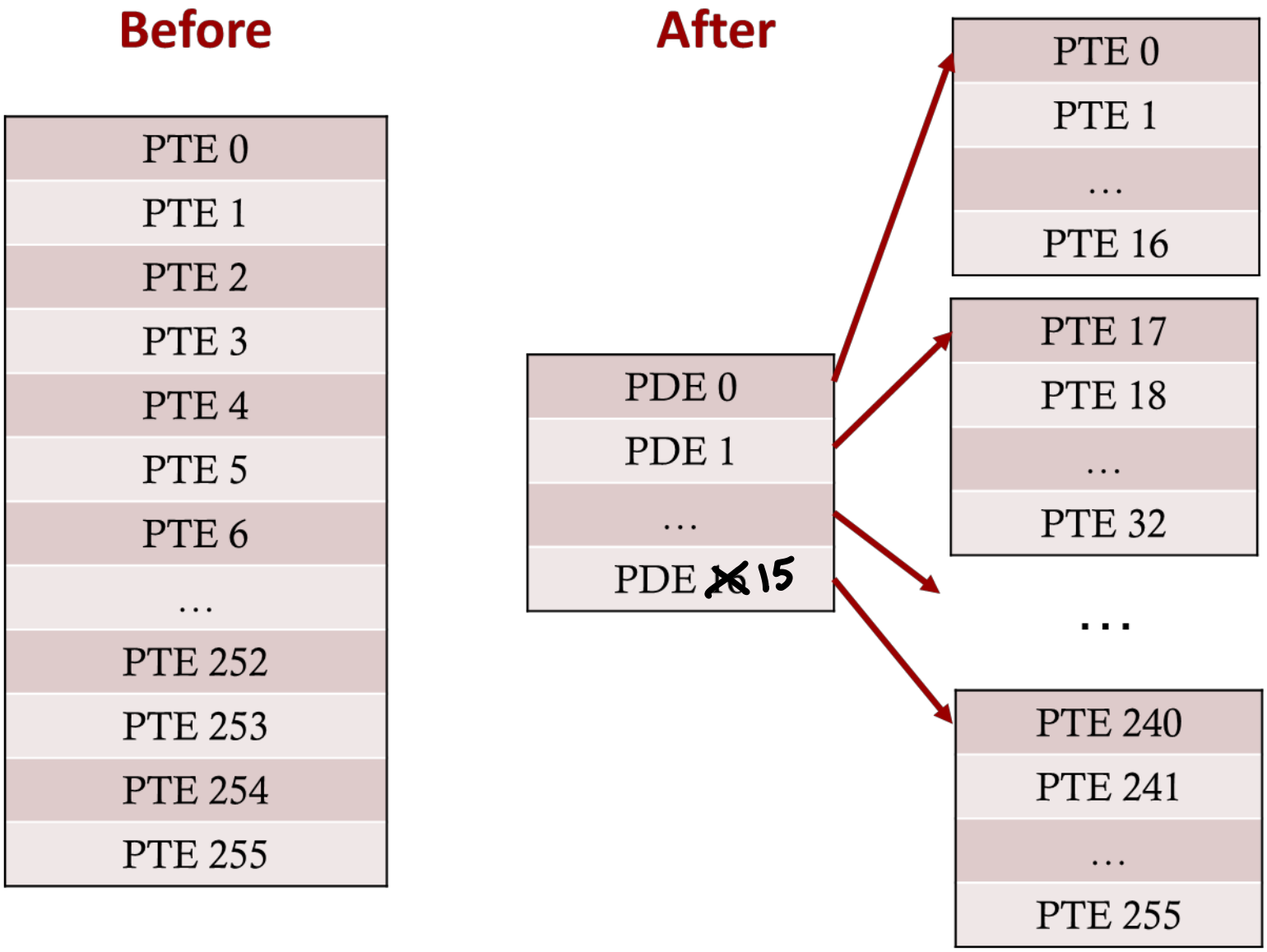
Given:

- |page| = 64 bytes
- |virtual address space| = 16 KB $2^{10} \cdot 16 = 2^{10} \cdot 2^4 = 2^{14}$
- |page table entry| = 4 bytes

Calculate:

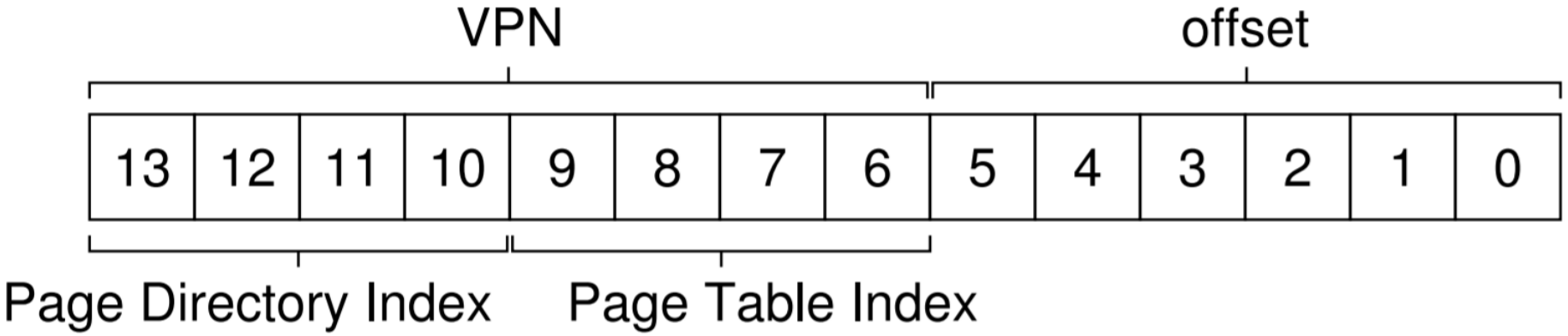
1. |virtual address| = ? $\log_2(16\,KB) = 14$
2. |offset| = ? $\log_2(64B) = 6$
3. |VPN| = ? $14 - 6 = 8$
4. # entries in page table = ? $2^8 = 256$
5. |page table| = ? $= 256 \cdot \frac{4B}{entry} = 1\,KB$
6. $\#\,chunks\ (pages) = \frac{|pg\ table|}{|page|} = 16\ pages =$ # entries in page directory

High-Level Example

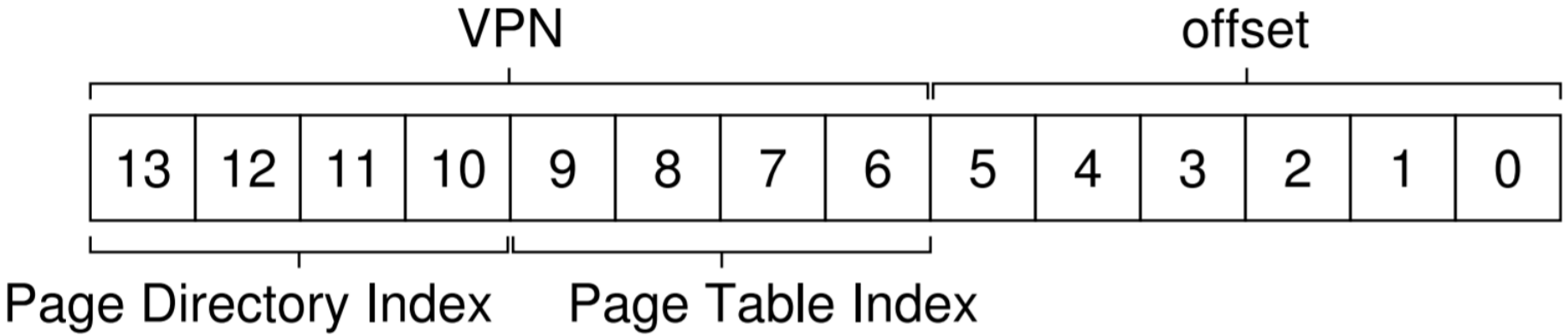


Address Translation #1

- |page directory| = 16 entries $\Rightarrow \log_2(16) = 4$ bits
- |page table pieces| = 16 entries $\Rightarrow$



Address Translation #2



1. Check page directory
2. if valid = 1, keep looking, check next level
3. form physical address

Address Translation Visual

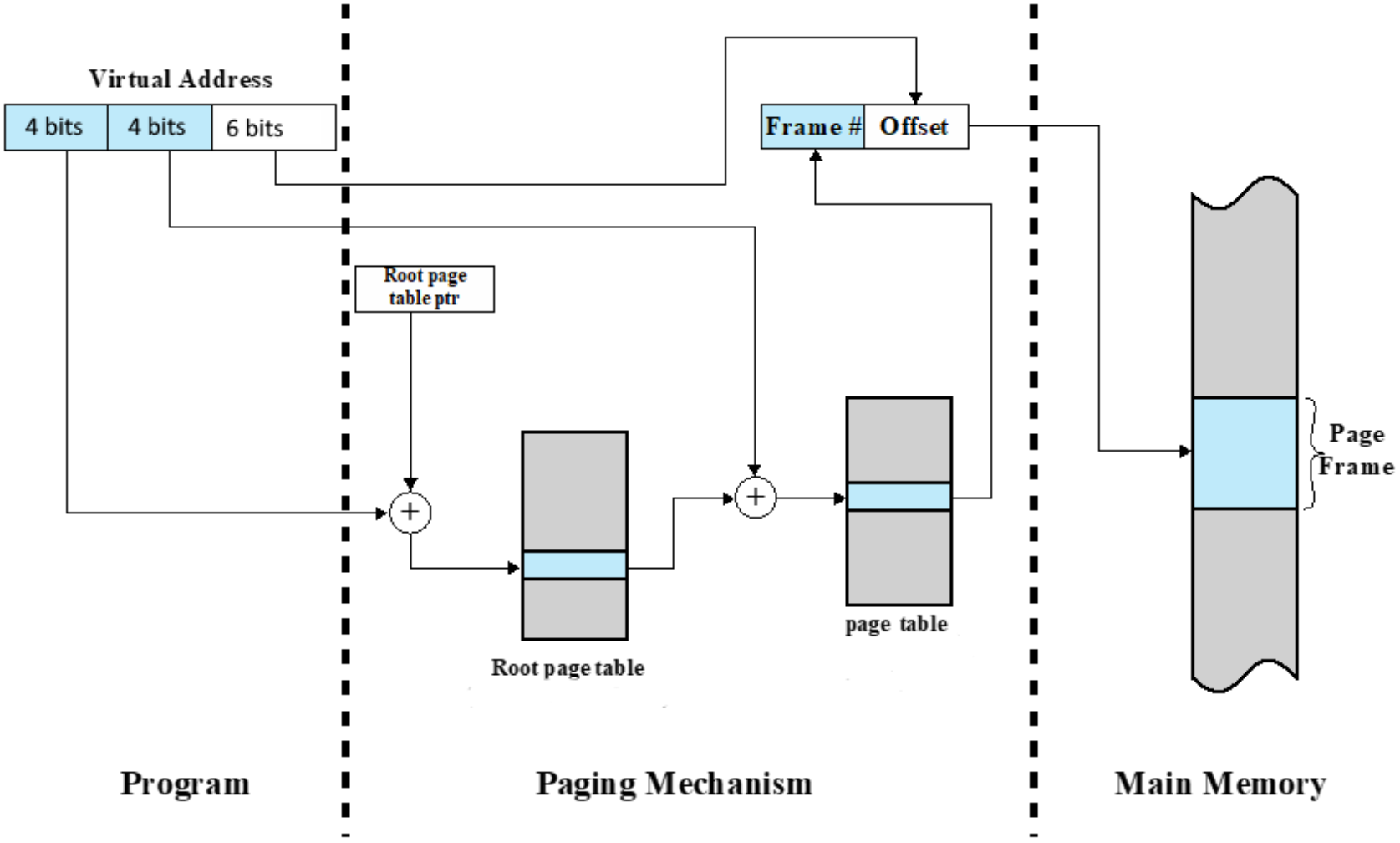


Figure 8.4 Address Translation in a Two-Level Paging System

Address Translation Steps (w/ TLB)

1. form VPN
2. check TLB *translation lookaside buffer*
   - Hit ≈ 99%
     a) check protection bits
     b) if good ⟶ form phys address
   - Miss
     a) form address of page directory entry (PDE)
     b) fetch PDE
     c) if valid → form addr of PTE
     d) fetch PTE
     e) check valid & protect bits
     f) if good → form phys addr & update TLB

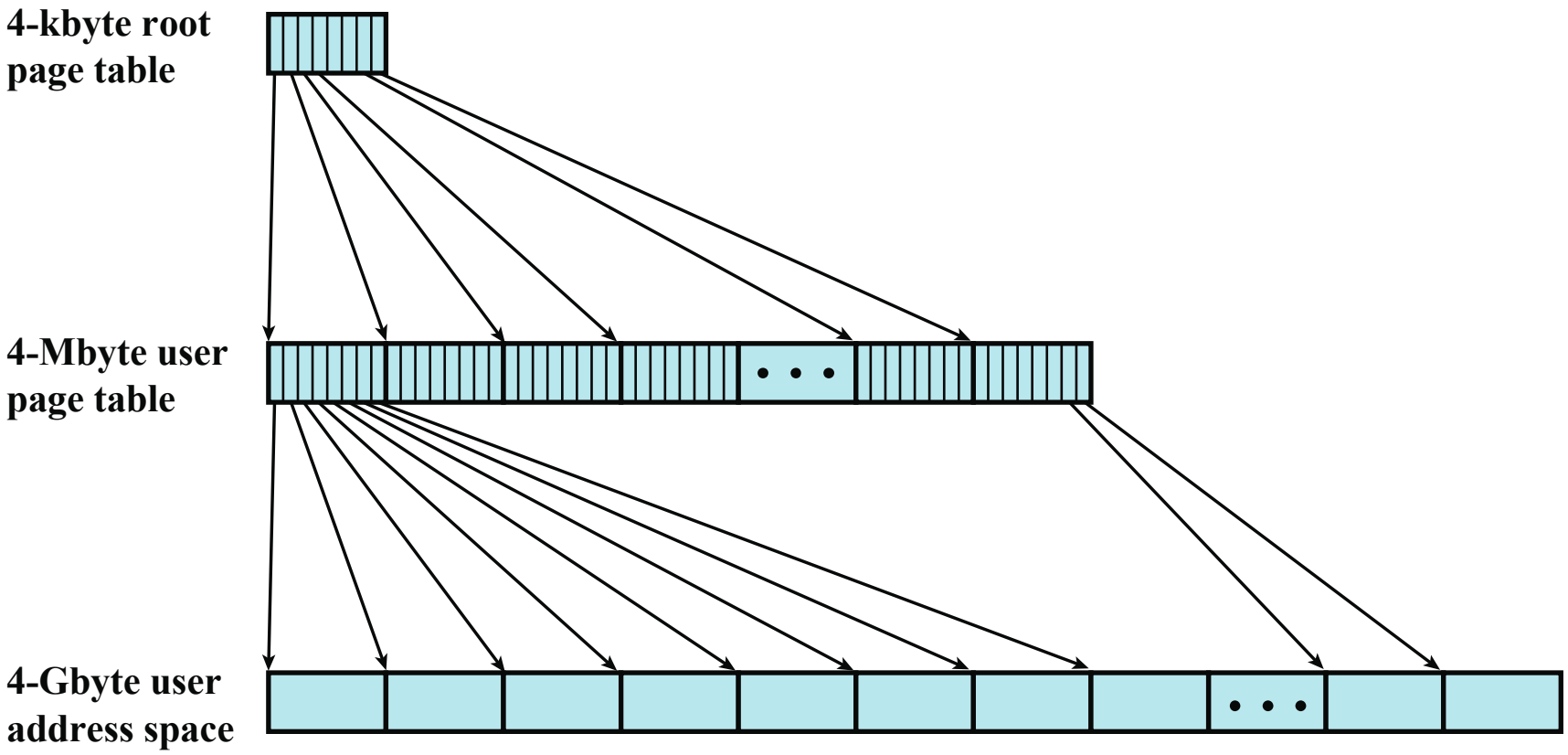
     increased miss penalty
3. fetch phys address

We're Not Done Yet!

So far, we've assumed only *two* levels:

1. Page Directory
2. Pieces of page table
   ⟹ do this again?

Three-Levels Visual



Solution (3) Analysis

+ extremely space efficient

- increased penalty of TLB misses

x86! does this (real world)

Solution (4)

Inverted Page Table: (Hash table)

- One page table, total
- Index by frame number
- Hash (VPN) ⟶ PFN (index of PTE)

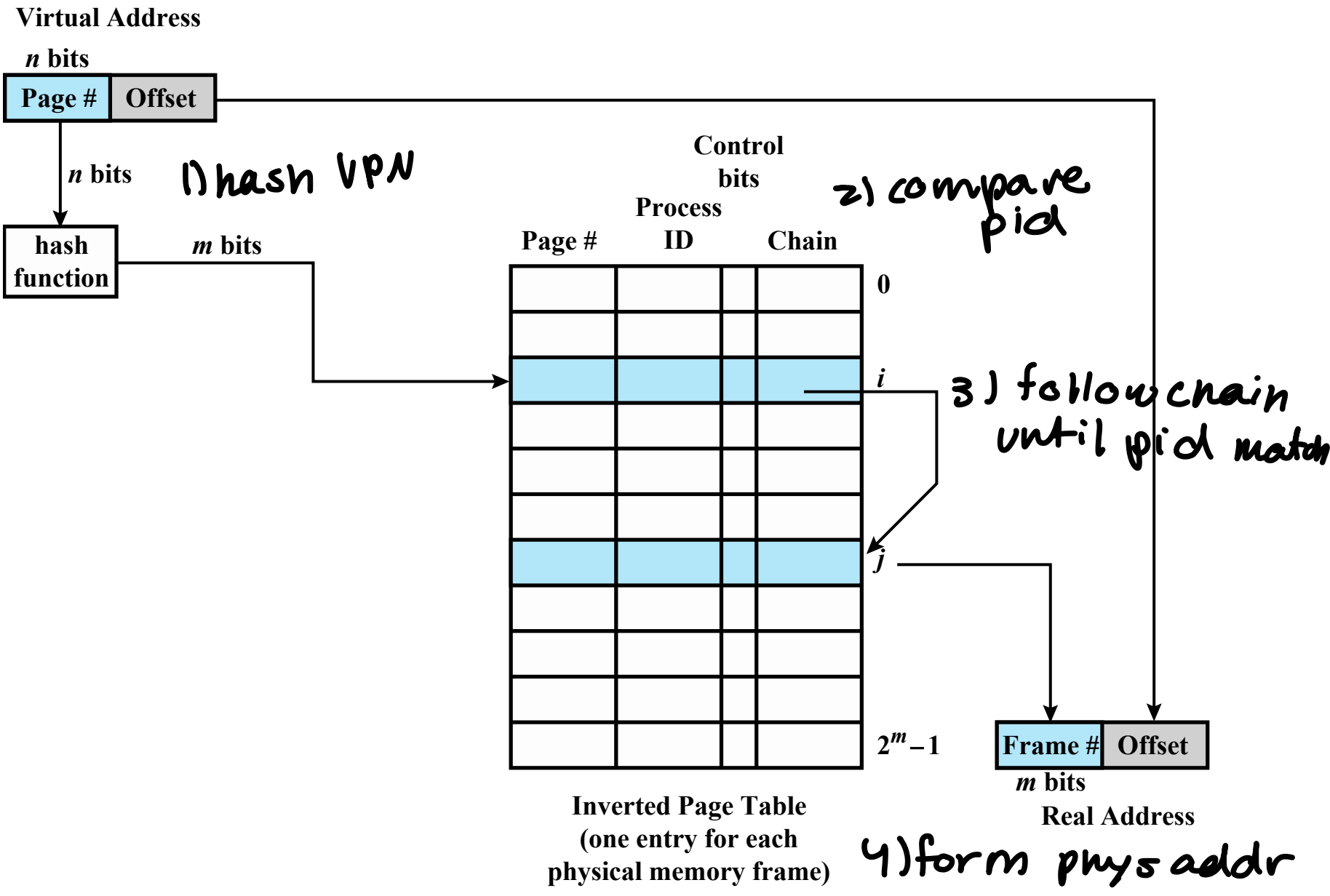
VPNs are not globally unique! unique per proc.

PFN1 PFN2

1) Hash (VPN, pid) ↦ PFN (ind of PTE)

2) store proc id in page table & "chain"

Solution (4) Visual



Solution (4) Analysis

+ page table size is fixed
  → dependent on # of frames,
  dependent on sz of physmem

– Difficult to implement
  "sharing"

powerpc uses this

– Hash funct. must be implemented in hardware

Solving Space: Summary

use a complex data structure

Learning Group Time

(Maybe)