Page Tables

When and why do we access a page table?

- On every instruction to translate virtual to physical addresses?

Page Tables

When does the OS access a page table?

- On every instruction to translate virtual to physical addresses?

No, in real machines it is only accessed

- *On TLB miss faults to refill the TLB*
- *During process creation and destruction*
- *When a process allocates or frees memory?*

Translation Lookaside Buffer

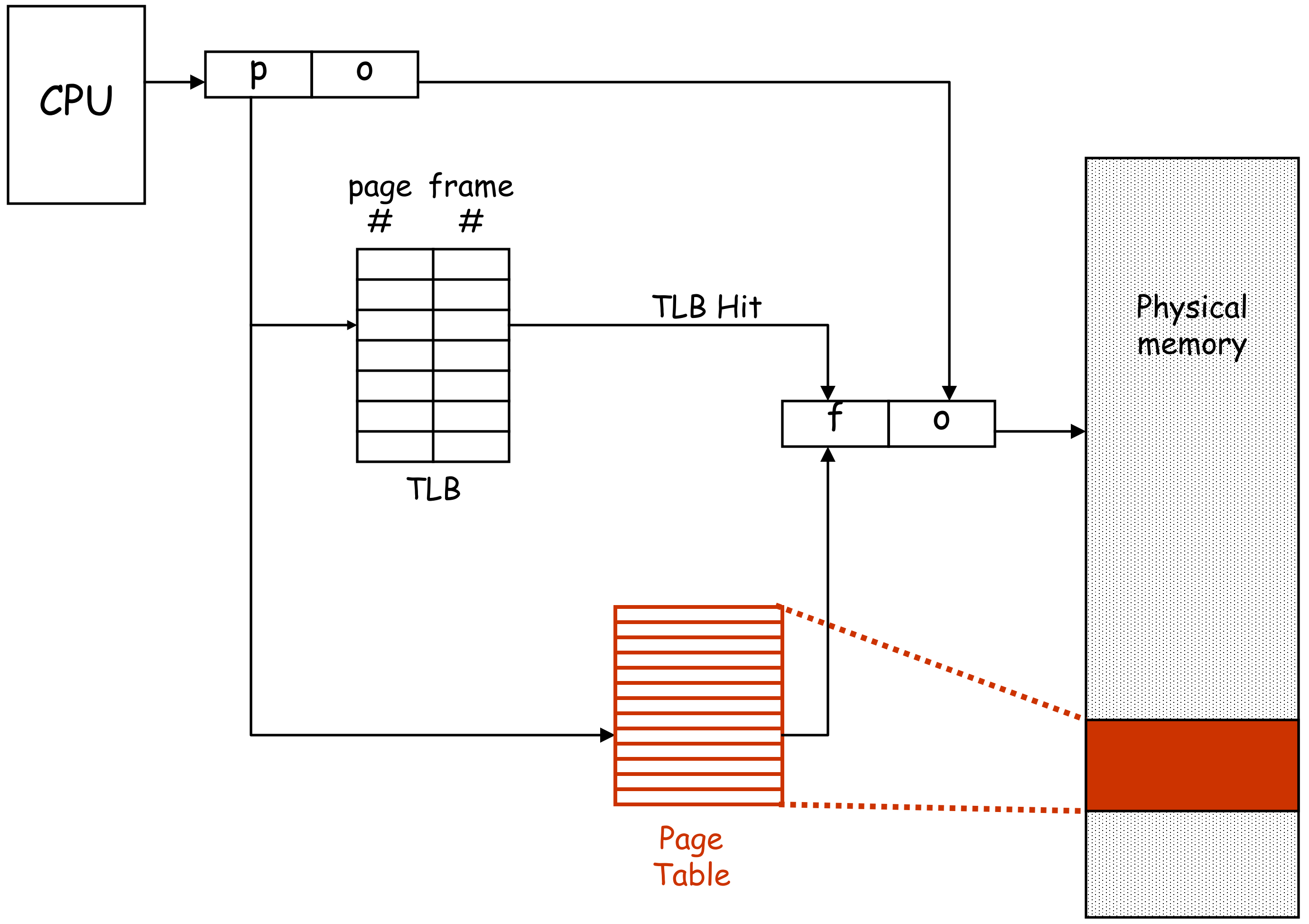
Problem: MMU can't keep up with the CPU if it goes to the page table on every memory access!

Translation Lookaside Buffer

Solution:

- Cache the page table entries in a hardware cache
- Small number of entries (e.g., 64)
- Each entry contains page number and other stuff from page table entry
- Fast and Fully associative:
  - indexed on page number
  - You can do a lookup in a single cycle
  - No conflict misses, only compulsory and capacity misses

Translation Lookaside Buffer

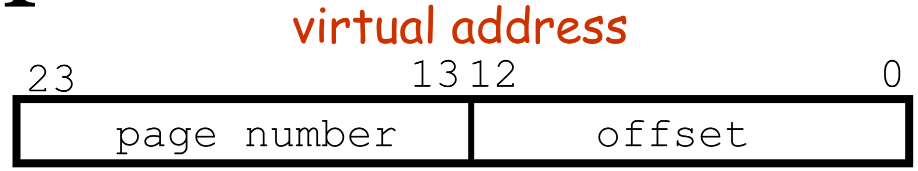


Hardware Operation of TLB

Key

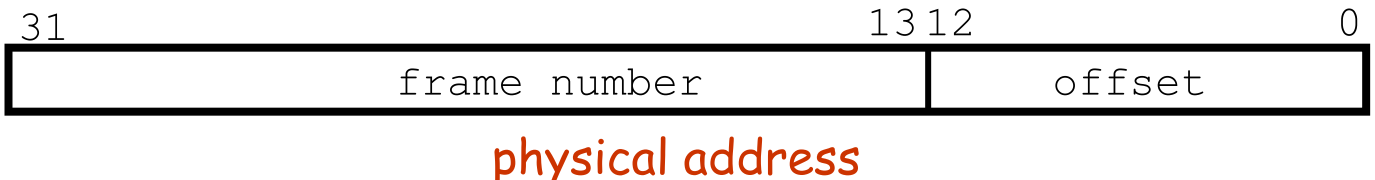
Page Number	Frame Number	Other				
23	37	unused	D	R	W	V
17	50	unused	D	R	W	V
92	24	unused	D	R	W	V
5	19	unused	D	R	W	V
12	6	unused	D	R	W	V

Hardware Operation of TLB



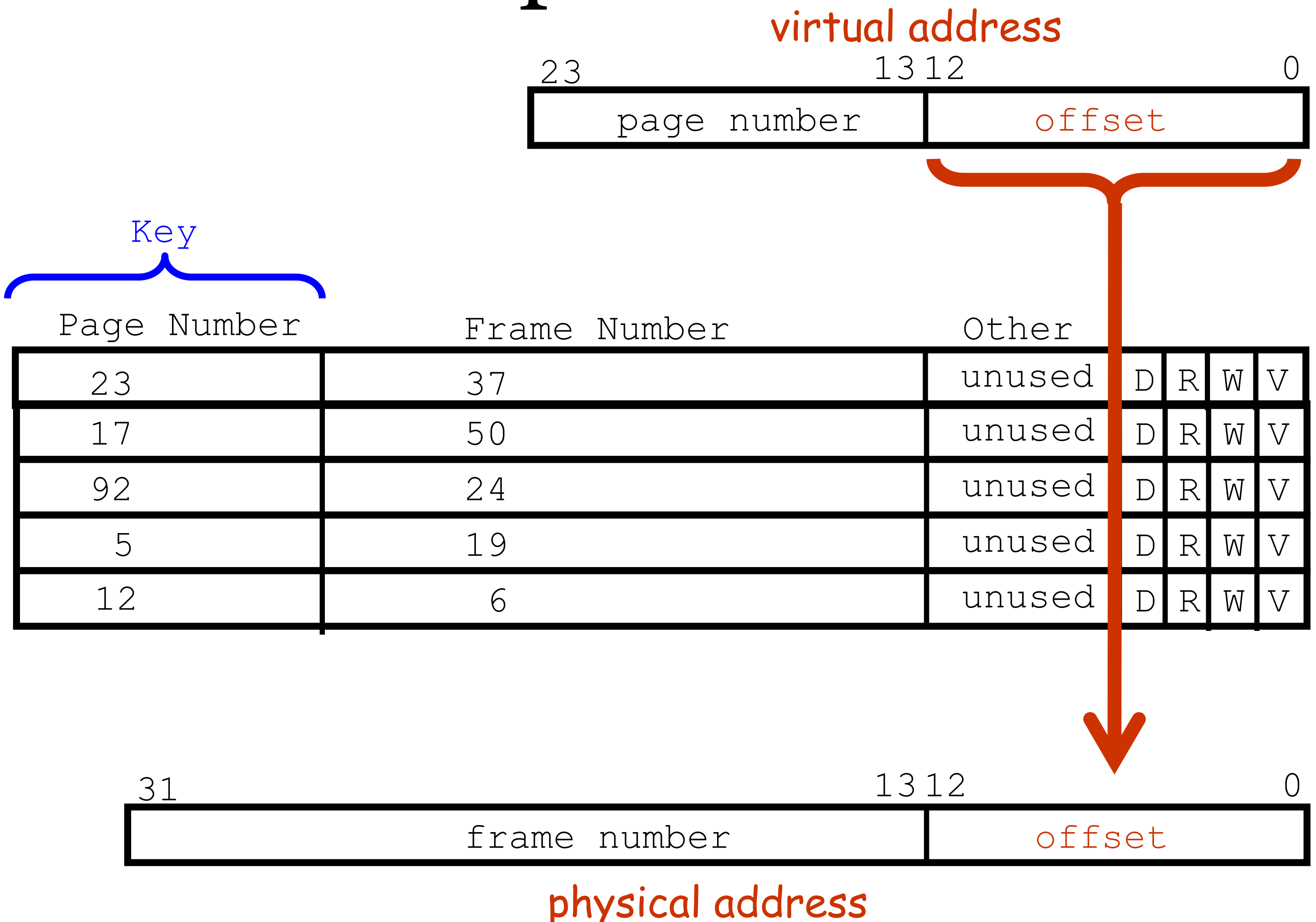
Key

Page Number	Frame Number	Other				
23	37	unused	D	R	W	V
17	50	unused	D	R	W	V
92	24	unused	D	R	W	V
5	19	unused	D	R	W	V
12	6	unused	D	R	W	V



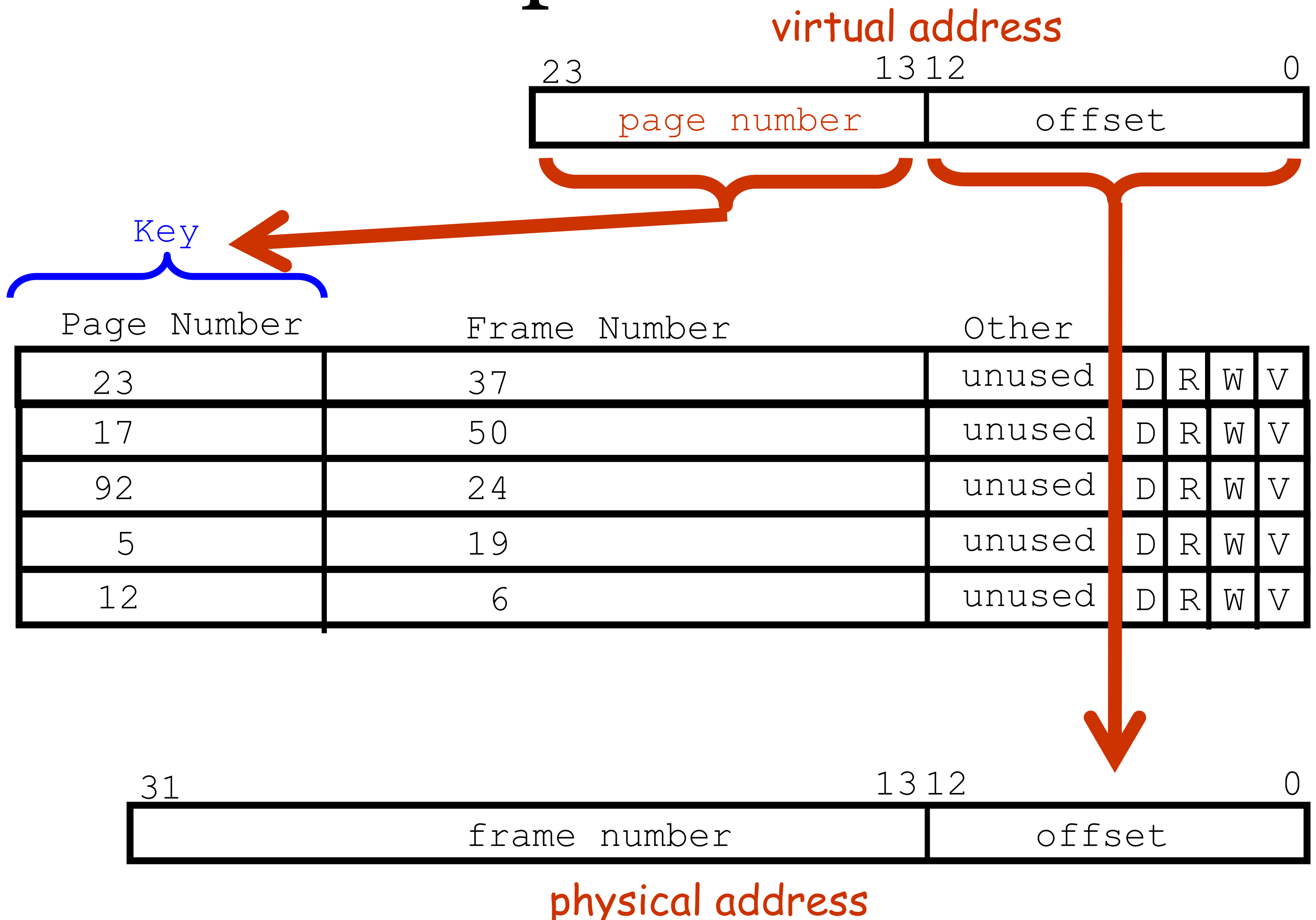
Portland State University

Hardware Operation of TLB



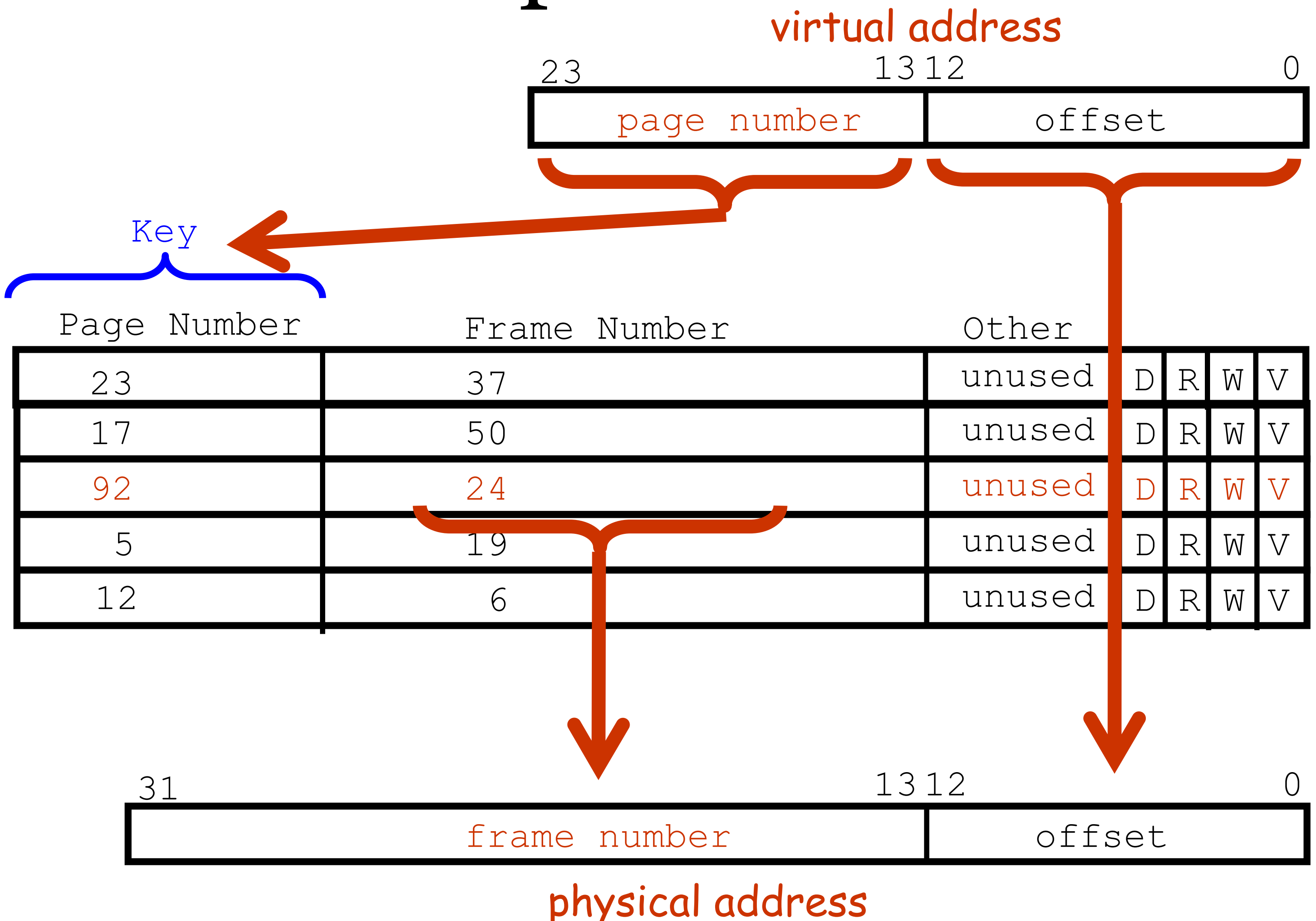
Page Number	Frame Number	Other				
23	37	unused	D	R	W	V
17	50	unused	D	R	W	V
92	24	unused	D	R	W	V
5	19	unused	D	R	W	V
12	6	unused	D	R	W	V

Hardware Operation of TLB



Page Number	Frame Number	Other				
23	37	unused	D	R	W	V
17	50	unused	D	R	W	V
92	24	unused	D	R	W	V
5	19	unused	D	R	W	V
12	6	unused	D	R	W	V

Hardware Operation of TLB



Page Number	Frame Number	Other				
23	37	unused	D	R	W	V
17	50	unused	D	R	W	V
92	24	unused	D	R	W	V
5	19	unused	D	R	W	V
12	6	unused	D	R	W	V

Software Operation of TLB

What if the entry is not in the TLB?

- Go look in the page table in memory
- Find the right entry
- Move it into the TLB
- But which TLB entry should be replaced?

Software Operation of TLB

Hardware TLB refill

- Page tables in specific location and format
- TLB hardware handles its own misses
- Replacement policy fixed by hardware

Software refill

- Hardware generates trap (TLB miss fault)
- Lets the OS deal with the problem
- Page tables become entirely a OS data structure!
- Replacement policy managed in software

Software Operation of TLB

How can we prevent the next process from using the last process's address mappings?

- Option 1: empty the TLB on context switch

  New process will generate faults until its pulls enough of its own entries into the TLB

- Option 2: just clear the "Valid Bit" on context switch

  New process will generate faults until its pulls enough of its own entries into the TLB

- Option 3: the hardware maintains a process id tag on each TLB entry

  Hardware compares this to a process id held in a specific register … on every translation

Page Tables

Do we access a page table when a process allocates or frees memory?

Page Table Usage

Do we access a page table when a process allocates or frees memory?

- Not necessarily

Library routines (malloc) can service small requests from a pool of free memory already allocated within a process address space

When these routines run out of space a new page must be allocated and its entry inserted into the page table

- This allocation is requested using a system call

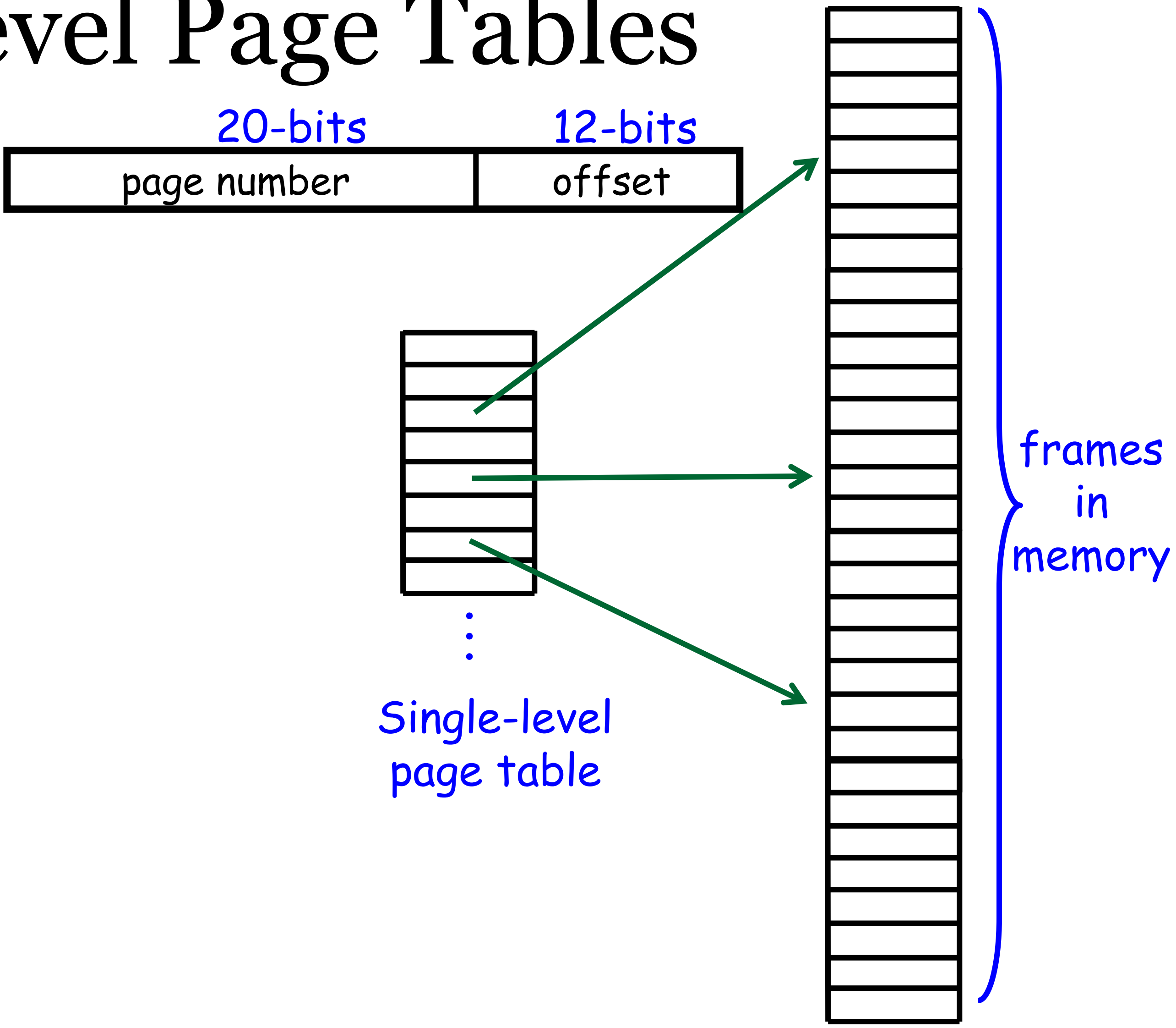
Page Table Design

Page table size depends on
- Page size
- Virtual address length

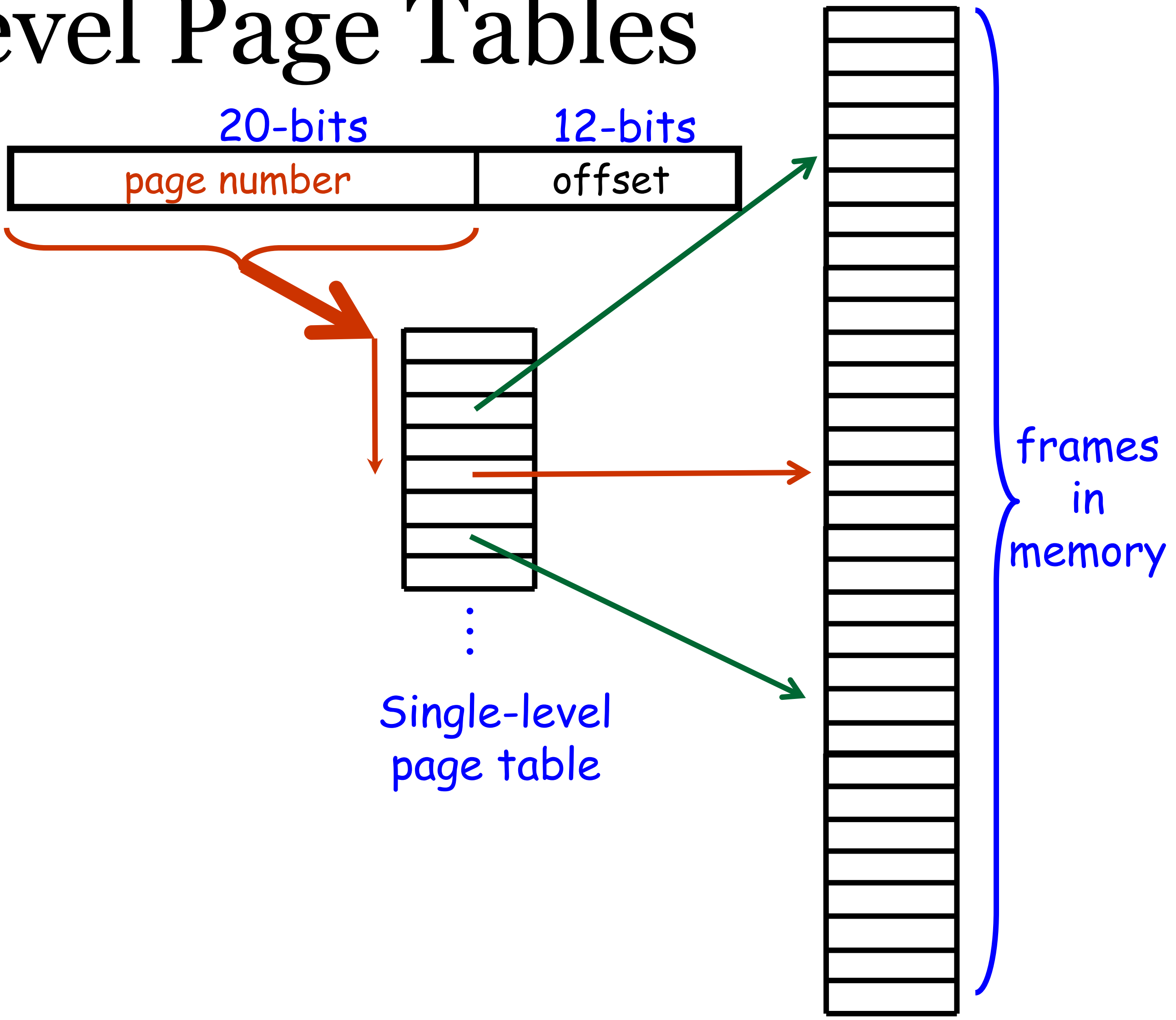
Memory used for page tables is overhead!
- How can we save space? … and still find entries quickly?

Three options
- Single-level page tables
- Multi-level page tables
- Inverted page tables

Single-Level Page Tables

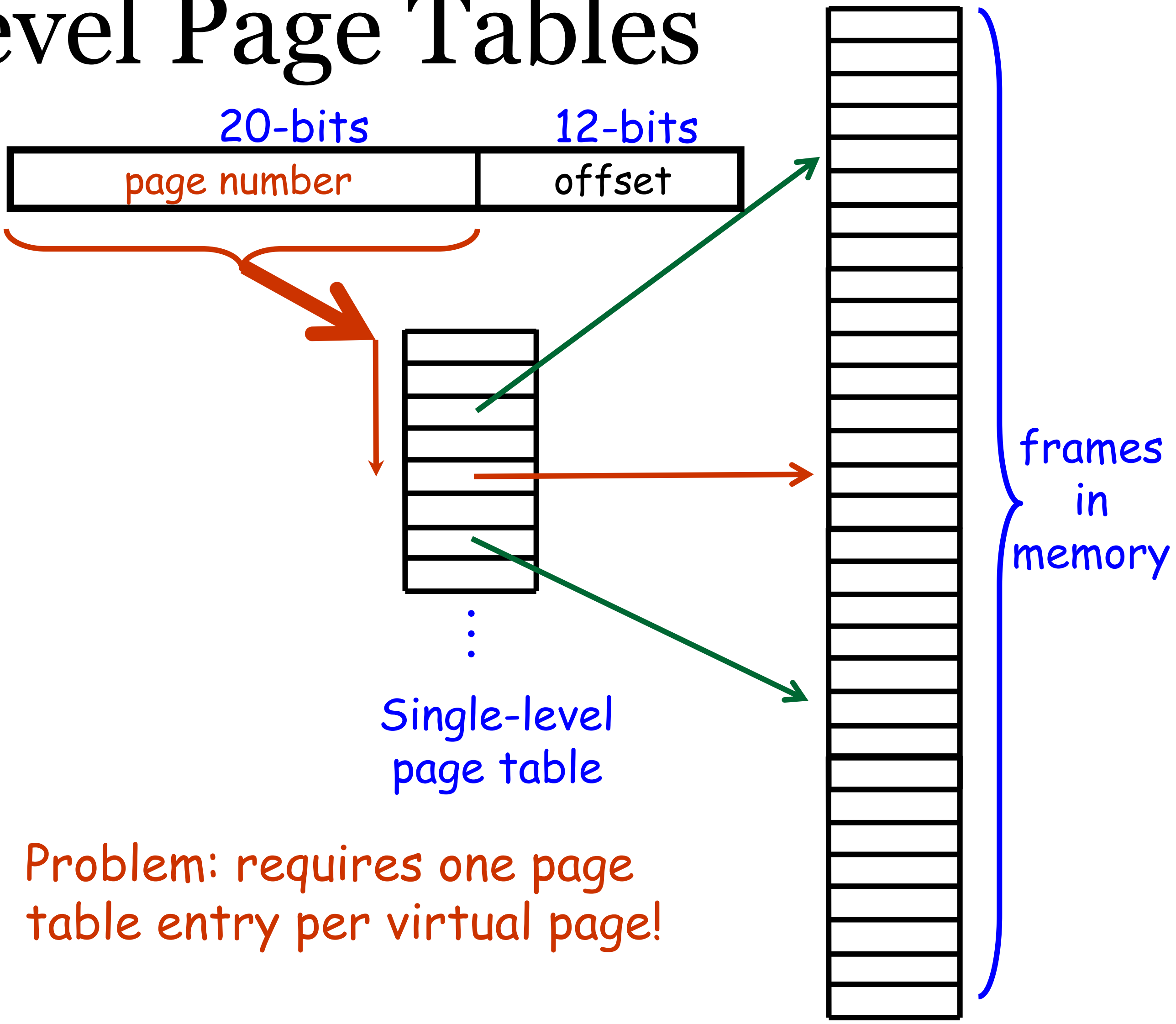


Single-Level Page Tables



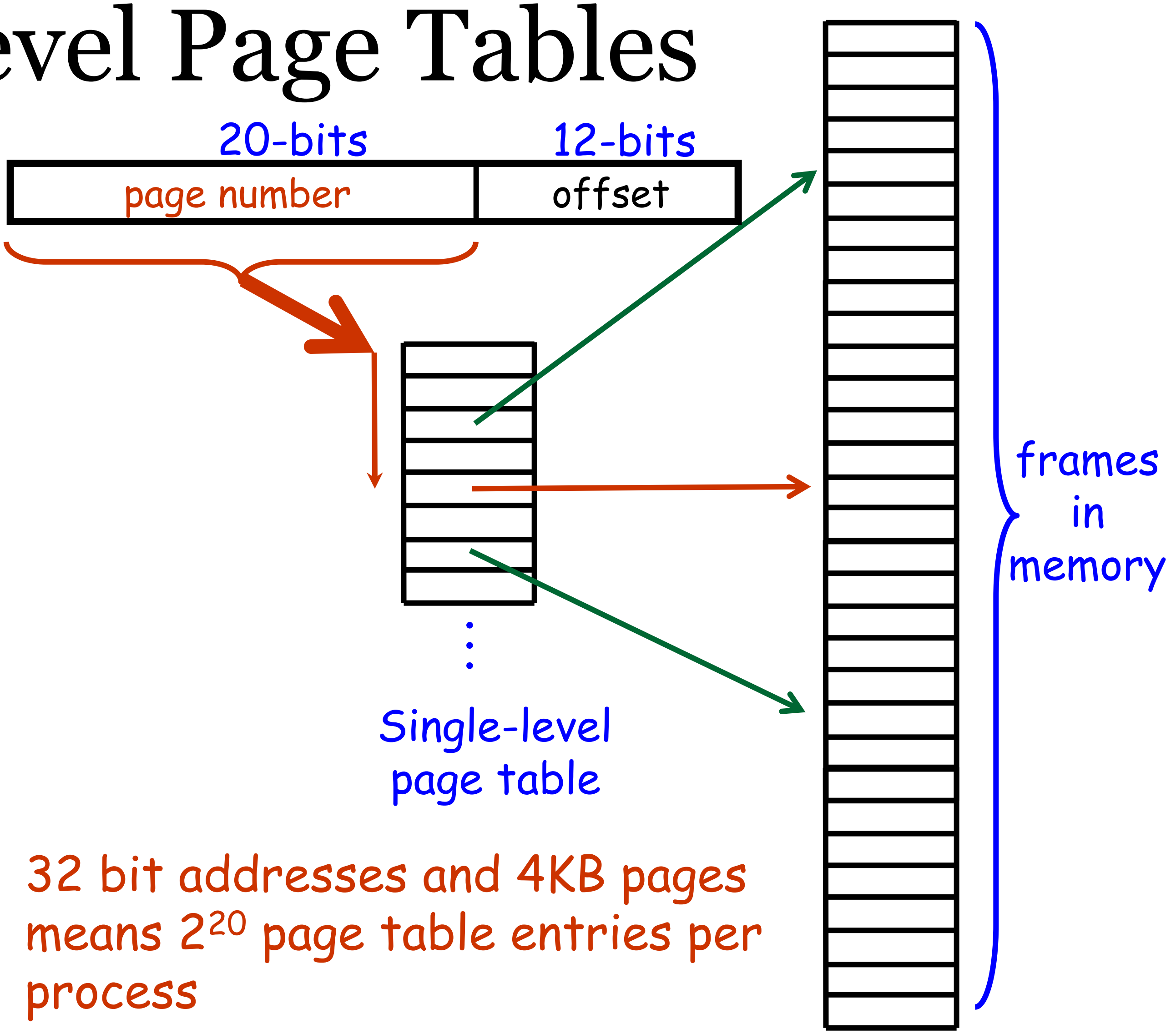
Portland State UNIVERSITY

Single-Level Page Tables



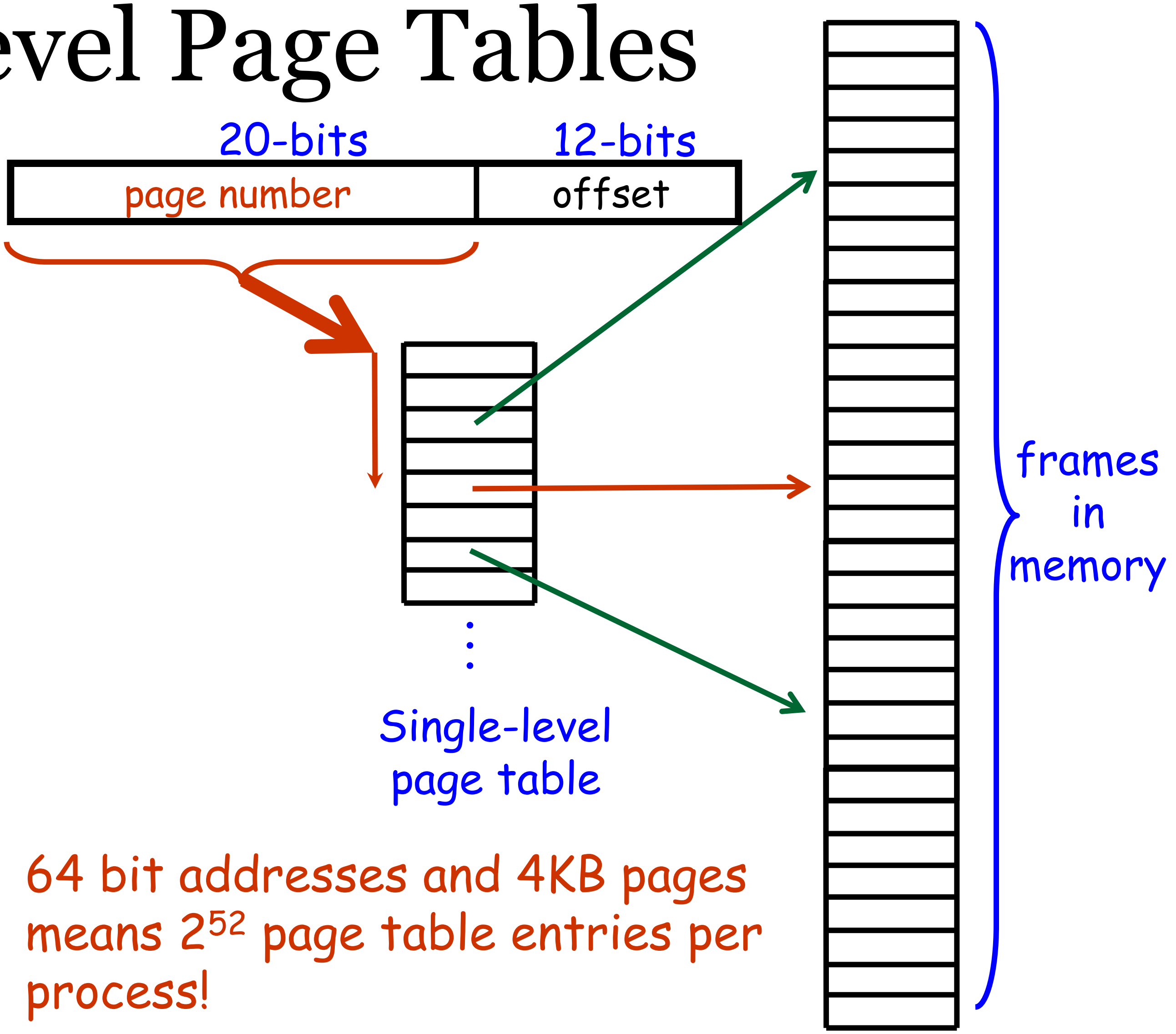
Problem: requires one page table entry per virtual page!

Single-Level Page Tables



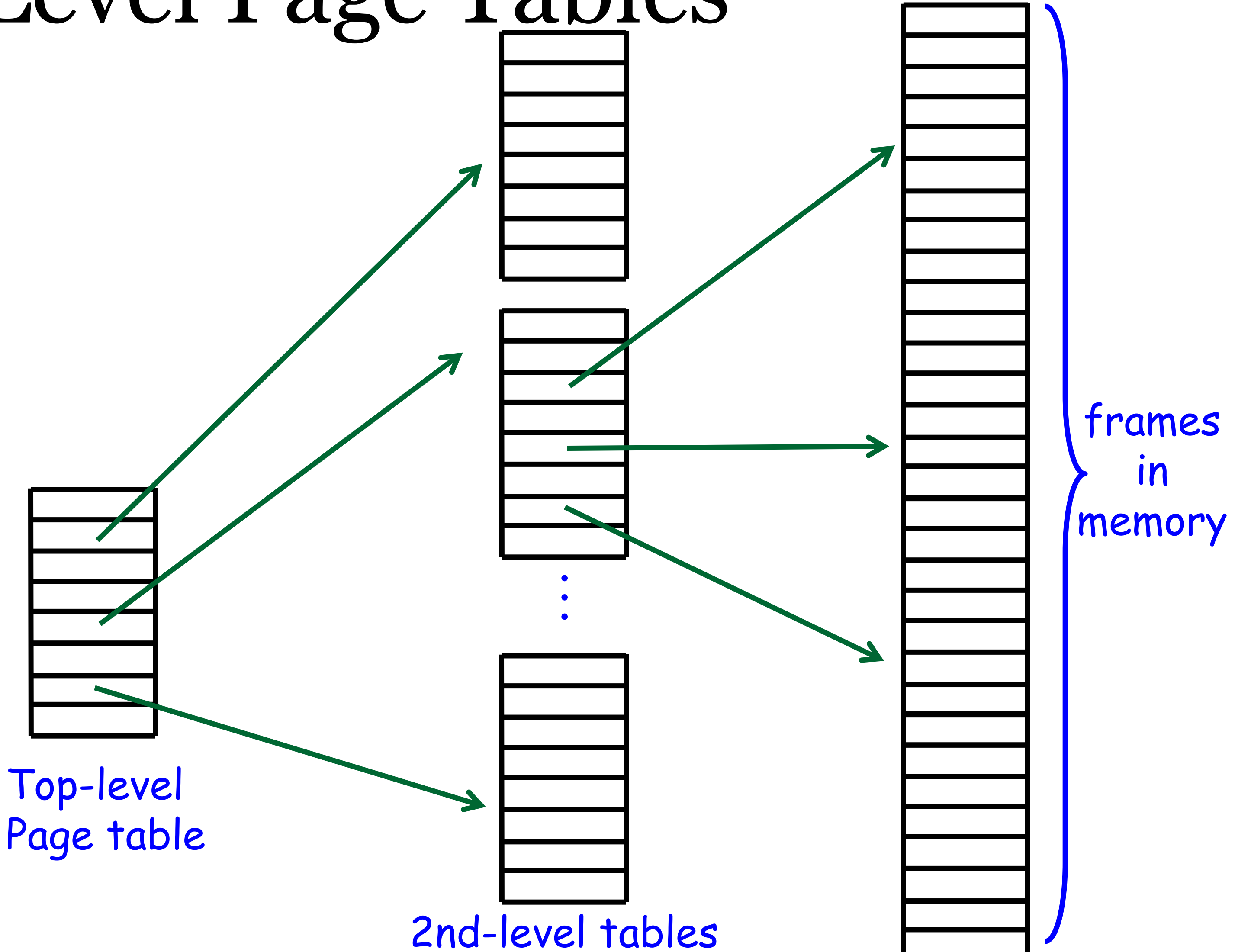
32 bit addresses and 4KB pages means 2^{20} page table entries per process

Single-Level Page Tables

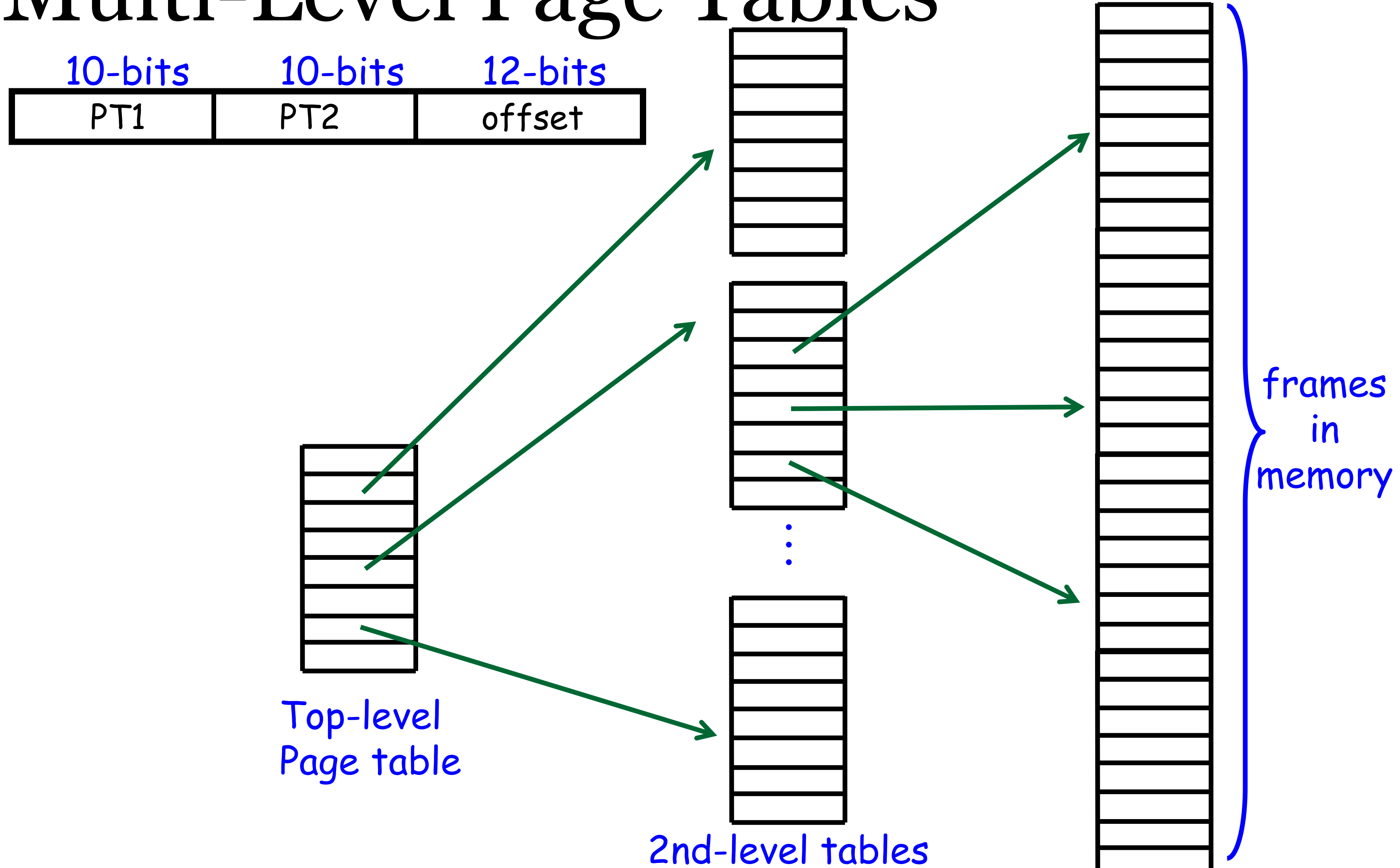


64 bit addresses and 4KB pages means 2^{52} page table entries per process!

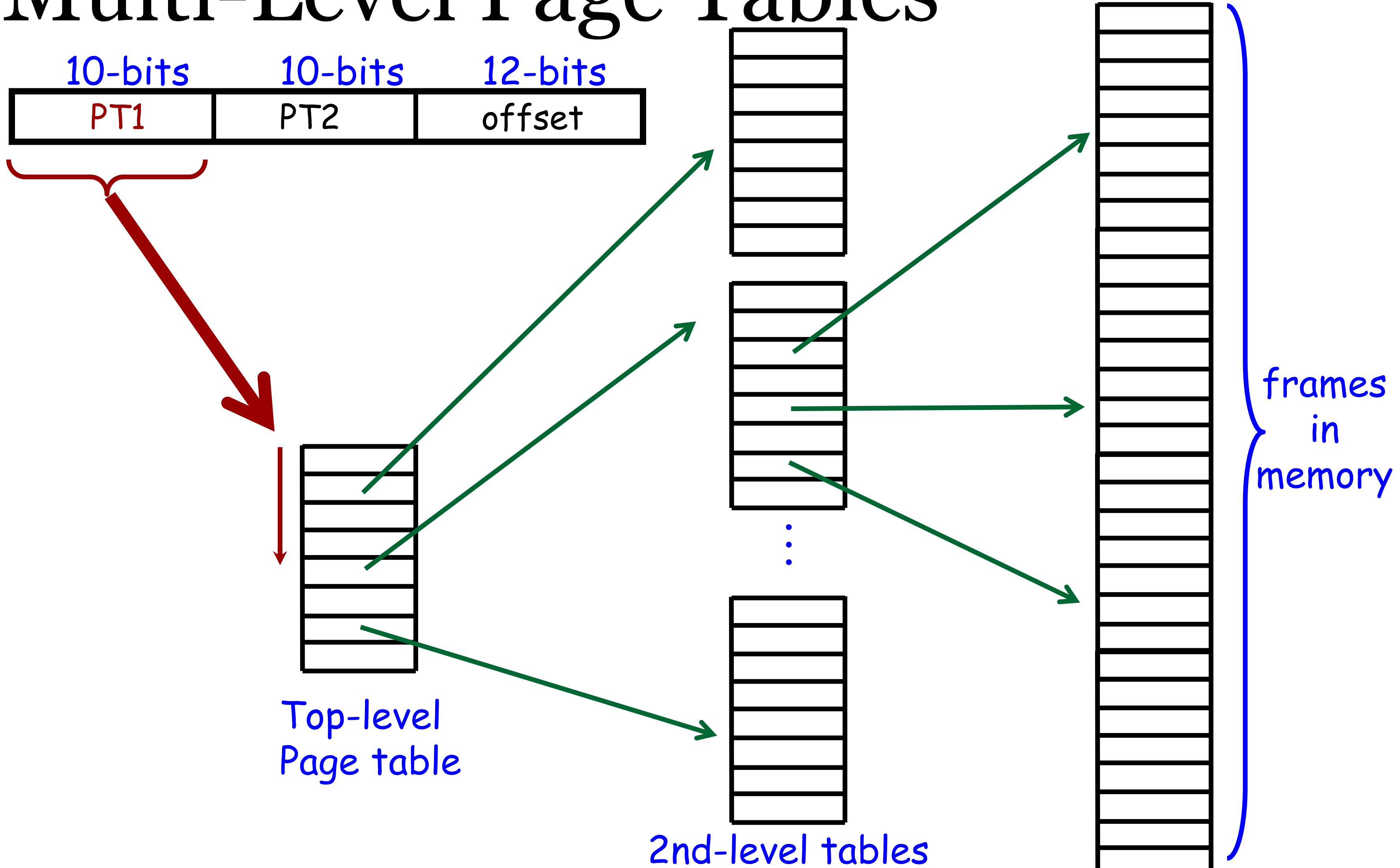
Multi-Level Page Tables



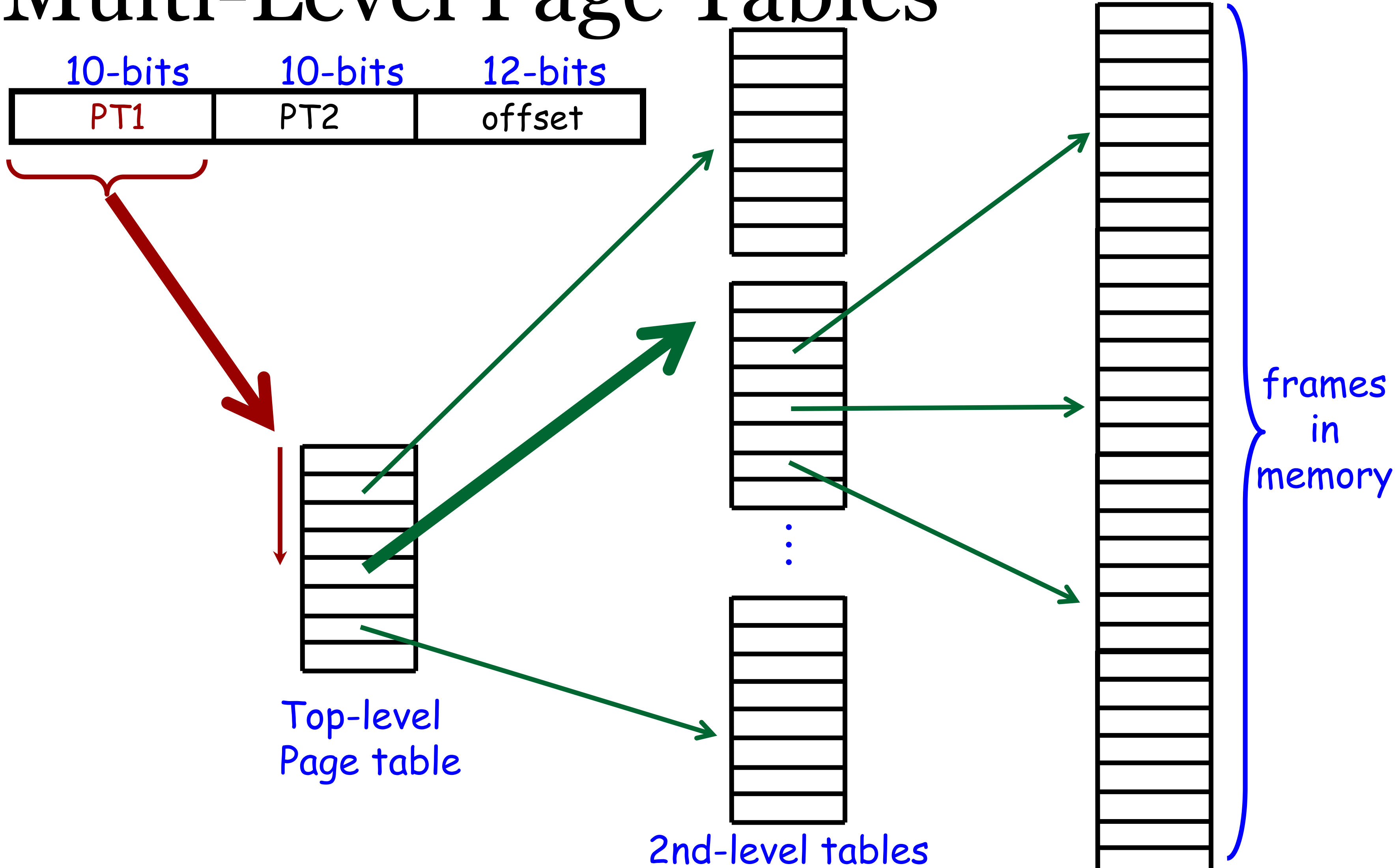
Multi-Level Page Tables



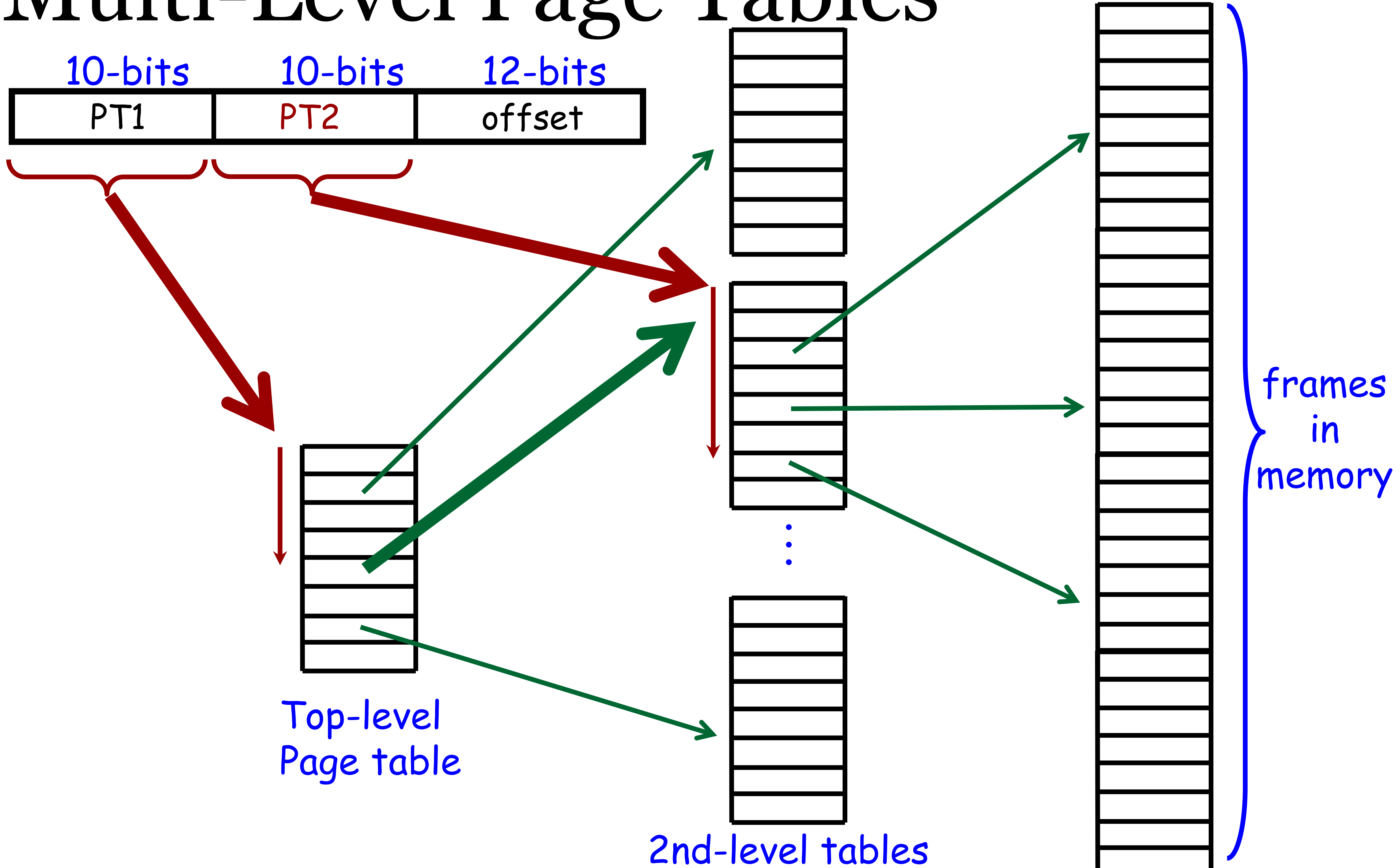
Multi-Level Page Tables



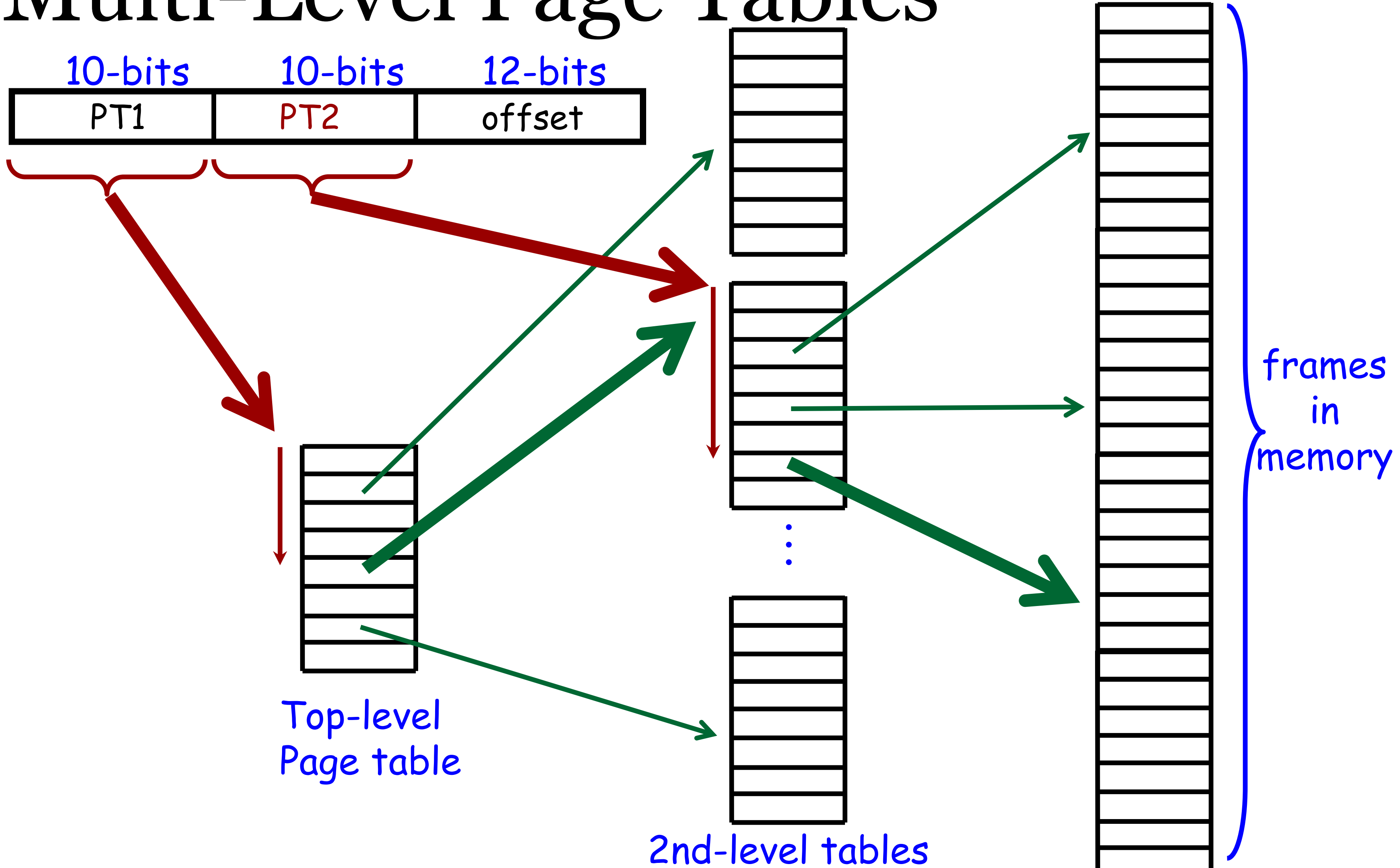
Multi-Level Page Tables



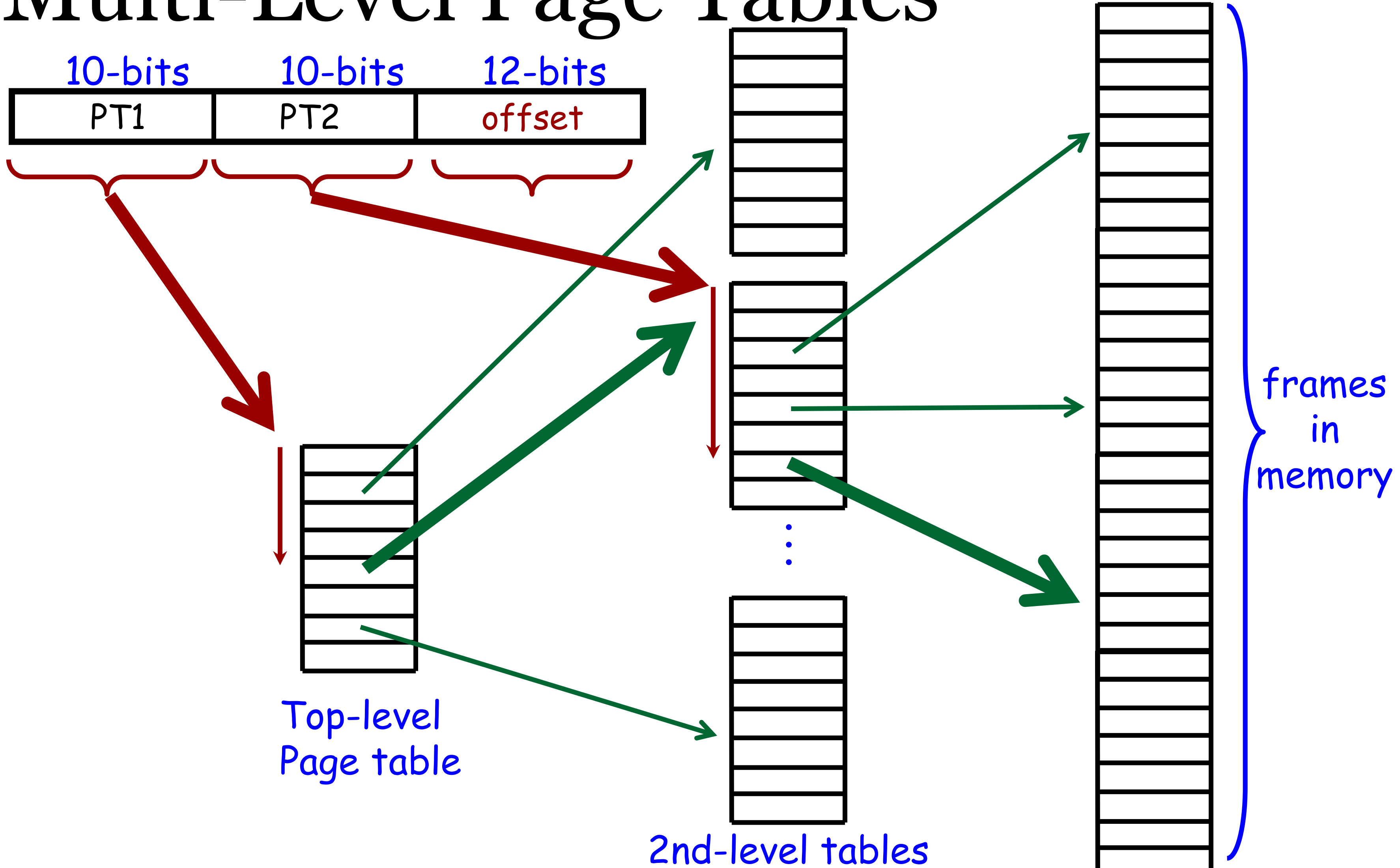
Multi-Level Page Tables



Multi-Level Page Tables



Multi-Level Page Tables



Multi-Level Page Tables

Ok, but how exactly does this save space?

Multi-Level Page Tables

Ok, but how exactly does this save space?

Not all pages within a virtual address space are allocated

- Not only do they not have a page frame, but that range of virtual addresses is not being used
- So no need to maintain complete information about it
- Some intermediate page tables are empty and not needed

We could also page the page table

- This saves space but slows access … a lot!