CS 153
Design of Operating Systems

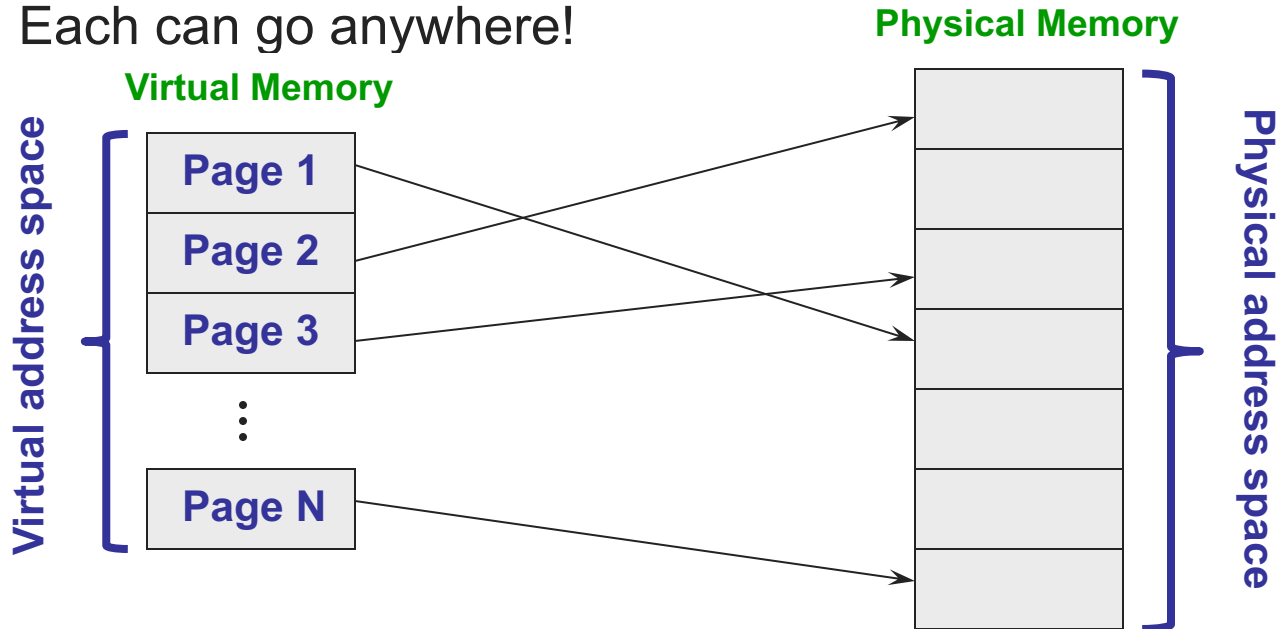## Fall 19

Lecture 10: Paging

Instructor: Chengyu Song

Paging

- New Idea: split virtual address space into multiple fixed size partitions
  - Each can go anywhere!

Virtual address space — Virtual Memory — Page 1, Page 2, Page 3, ⋮, Page N

Physical Memory — **Physical address space**

Paging solves the external fragmentation problem by using fixed sized units in both physical and virtual memory
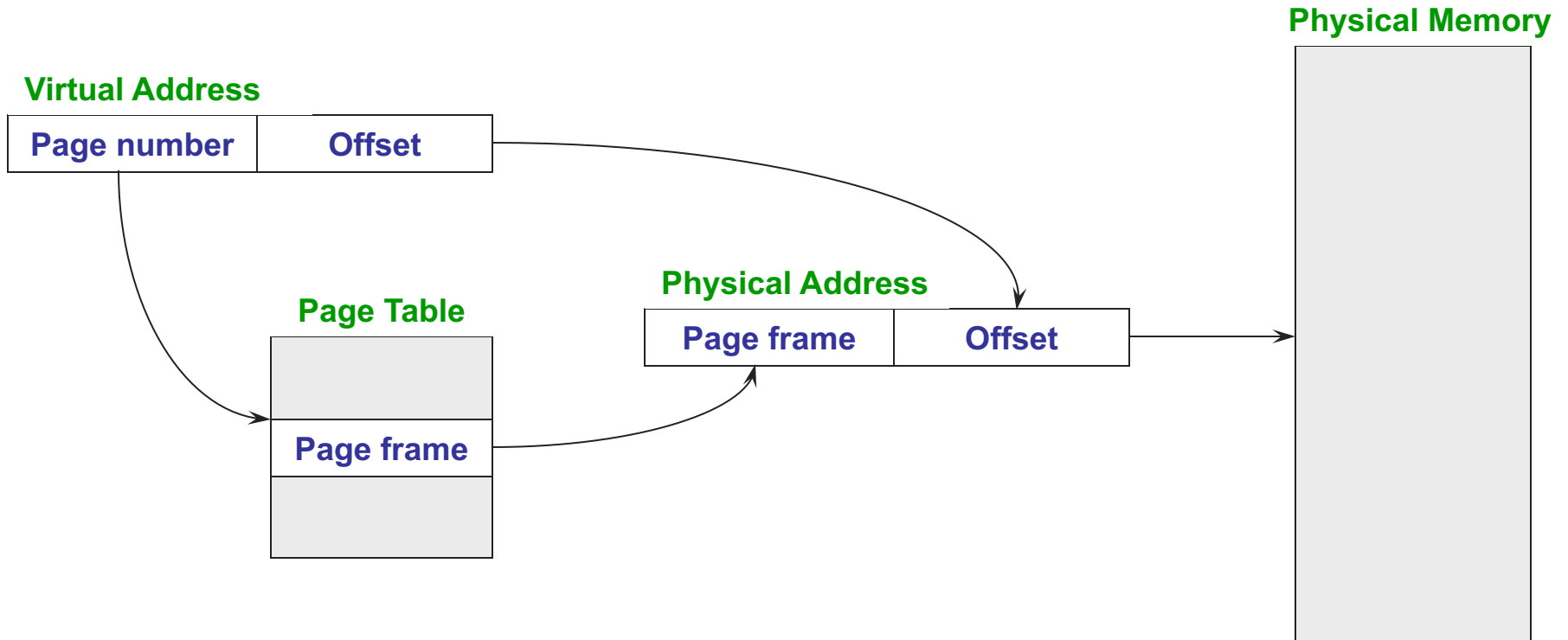
But need to keep track of where things are!

Process Perspective

- Processes view memory as one contiguous address space from 0 through N

  - Virtual address space (VAS)

- In reality, pages are scattered throughout physical storage

- The mapping is invisible to the program

- Protection is provided because a program cannot reference memory outside of its VAS

  - The address "0x1000" maps to different physical addresses in different processes

Paging

- Translating addresses

  - Virtual address has two parts: virtual page number and offset

  - Virtual page number (VPN) is an index into a page table

  - Page table determines page frame number (PFN)

  - Physical address is PFN::offset

- Page tables

  - Map virtual page number (VPN) to page frame number (PFN)

    » VPN is the index into the table that determines PFN

  - One page table entry (PTE) per page in virtual address space

    » Or, one PTE per VPN

Page Lookups

Physical Memory

Virtual Address

Page number	Offset

Page Table

Page frame

Physical Address

Page frame	Offset

Paging Example
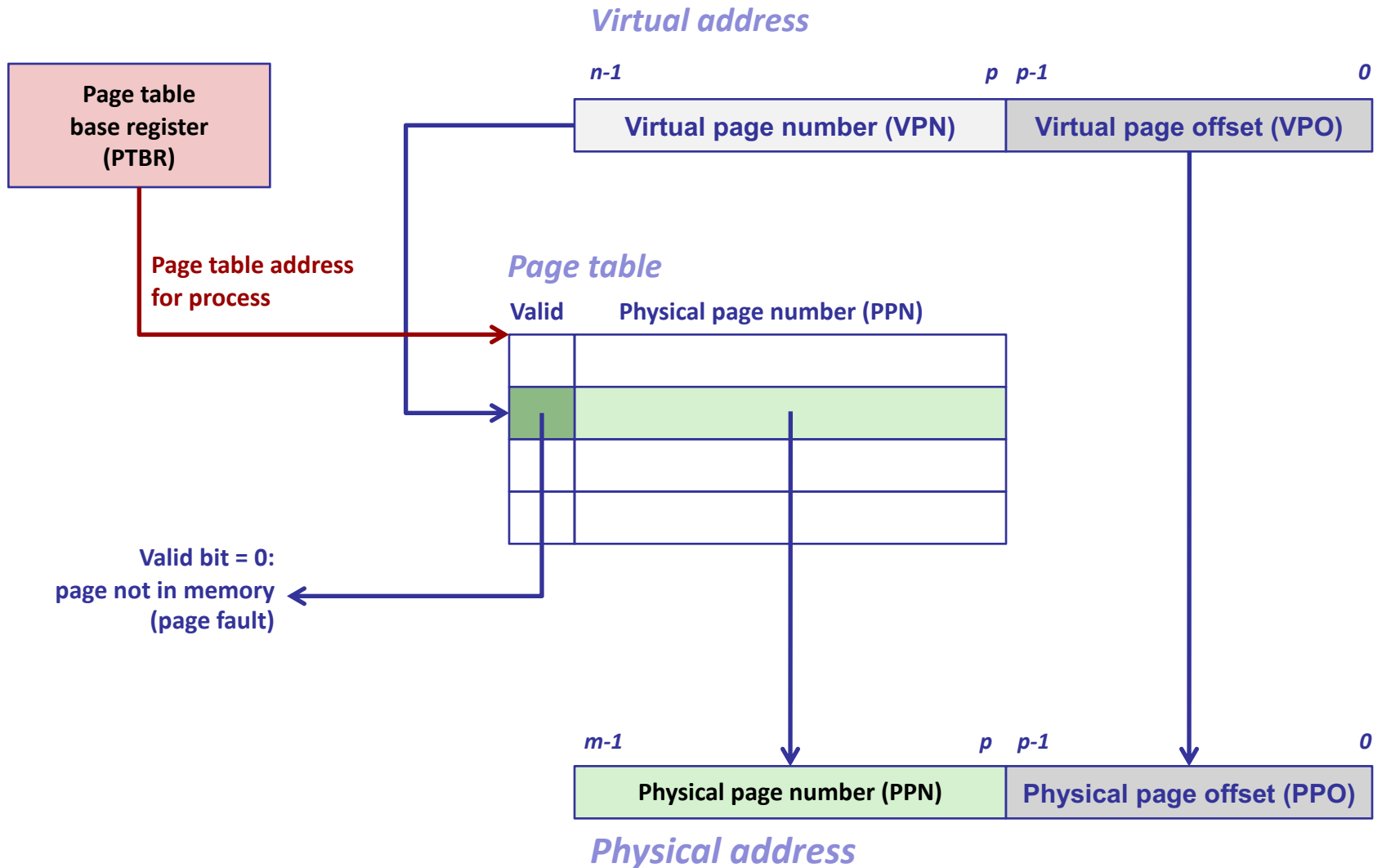
- Pages are 4KB

  - Offset is 12 bits (because 4KB = 2^{12} bytes)

  - VPN is 20 bits (32 bits is the length of every virtual address)

- Virtual address is 0x7468

  - Virtual page is 0x7, offset is 0x468

- Page table entry 0x7 contains 0x2000

  - Page frame number is 0x2000

  - Seventh virtual page is at address 0x2000 (2nd physical page)

- Physical address = 0x2000 + 0x468 = 0x2468

Page Table Entries (PTEs)

1	1	1	2	20
M	R	V	Prot	Page Frame Number

- Page table entries control mapping
  - The Modify bit says whether or not the page has been written
    - » It is set when a write to the page occurs (for caching)
  - The Reference bit says whether the page has been accessed
    - » It is set when a read or write to the page occurs (for eviction)
  - The Valid bit says whether or not the PTE can be used
    - » It is checked each time the virtual address is used (Why?)
  - The Protection bits say what operations are allowed on page
    - » Read, write, execute (Why do we need these?)
  - The page frame number (PFN) determines physical page

Address Translation With a Page Table

Page table base register (PTBR)

Page table address for process

Virtual address

n-1	p	p-1	0
Virtual page number (VPN)		Virtual page offset (VPO)	

Page table

Valid   Physical page number (PPN)

Valid bit = 0:
page not in memory (page fault)

Physical address

m-1	p	p-1	0
Physical page number (PPN)		Physical page offset (PPO)	

Paging Advantages

- **Easy to allocate memory**

  - Memory comes from a free list of fixed size chunks

  - Allocating a page is just removing it from the list

  - External fragmentation not a problem

    - » All pages of the same size

- **Simplifies protection**

  - All chunks are the same size

  - Like fixed partitions, don't need a limit register

- **Simplifies virtual memory – later**

Paging Limitations

- Can still have internal fragmentation

  - Process may not use memory in multiples of a page

- Memory reference overhead

  - 2 references per address lookup (page table, then memory)

  - What can we do?

- Memory required to hold page table can be significant

  - Need one PTE per page

  - 32-bit address space w/ 4KB pages = 2^{20} PTEs

  - 4 bytes/PTE = 4MB/page table

  - 25 processes = 100MB just for page tables!

  - What can we do?

Managing Page Tables

- Last lecture we computed the size of the page table for a 32-bit address space w/ 4K pages to be 4MB

  - This is far too much overhead for each process

- How can we reduce this overhead?

  - Observation: process don't use all the addresses, so we only need to map the portion of the address space this is actually being used (tiny fraction of entire address space)

- How do we only map what is being used?

  - Can dynamically extend page table

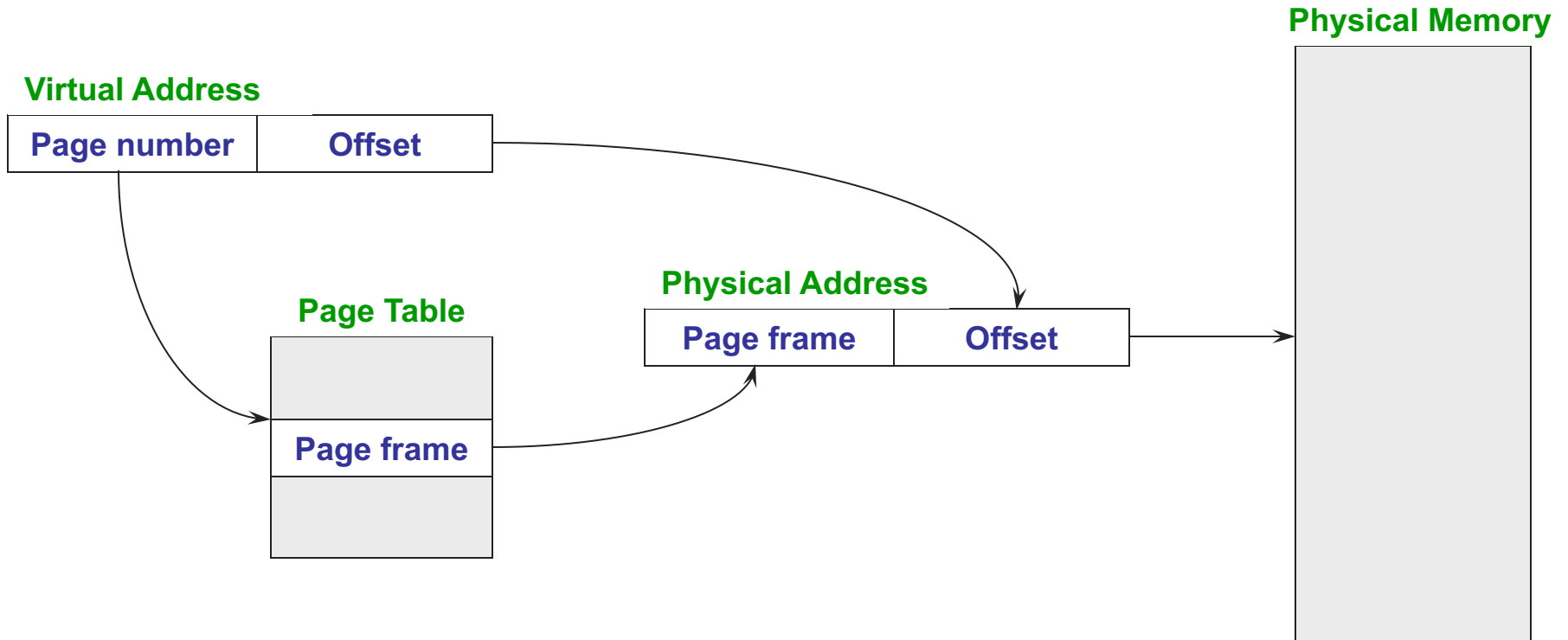- Use another level of indirection: two-level page tables

Two-Level Page Tables

- Two-level page tables
  - Each virtual address (VA) has three parts:
    - » Master page number, secondary page number, and offset

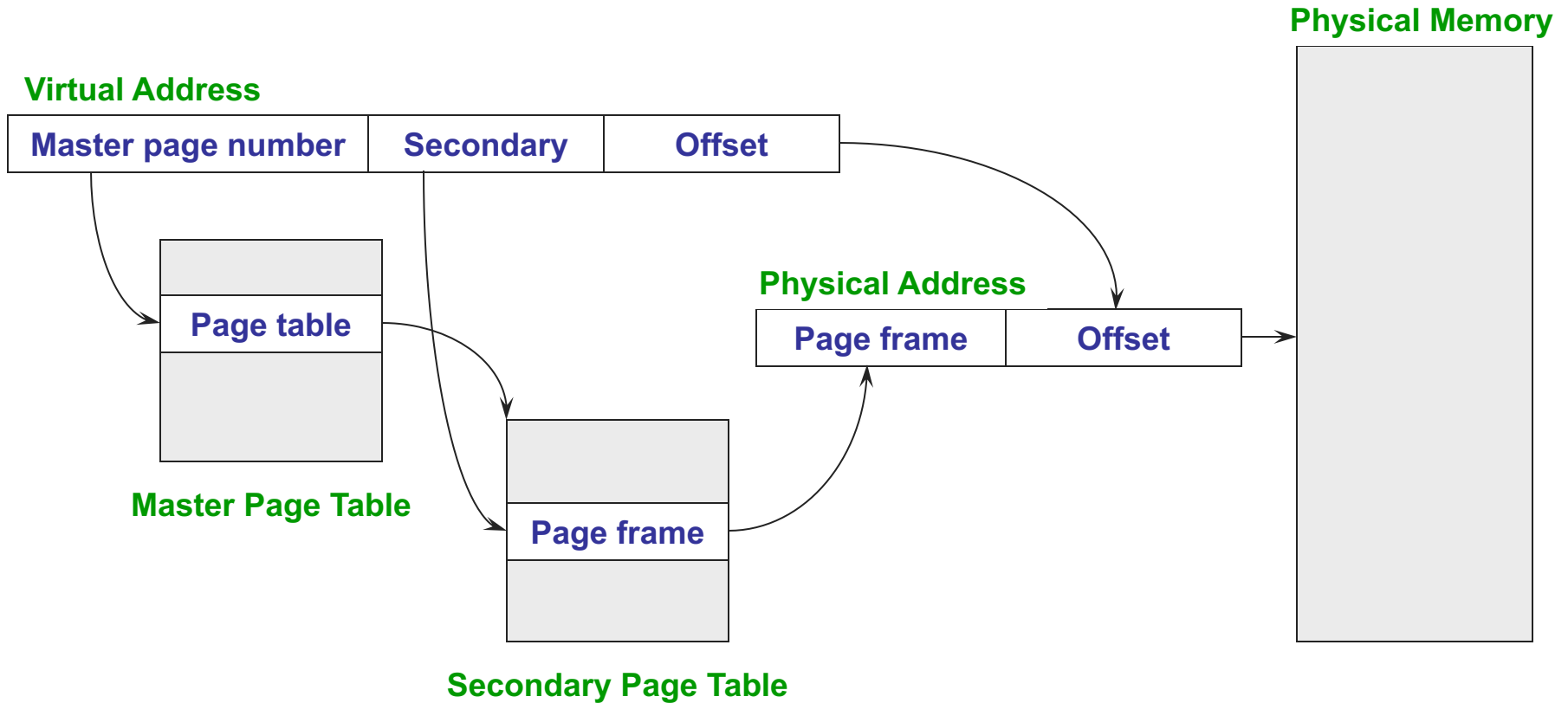 | Master page number | Secondary | Offset |
    | --- | --- | --- |

  - Master page table maps VA to secondary page table
  - Secondary page table maps virtual page number to physical page
  - Offset indicates where in physical page address is located

One-Level Page Lookups

Physical Memory

Virtual Address

Page number	Offset

Page Table

Page frame

Physical Address

Page frame	Offset

Two-Level Page Lookups

Physical Memory

Virtual Address

Master page number	Secondary	Offset

Page table

Master Page Table

Physical Address

Page frame	Offset

Page frame

Secondary Page Table
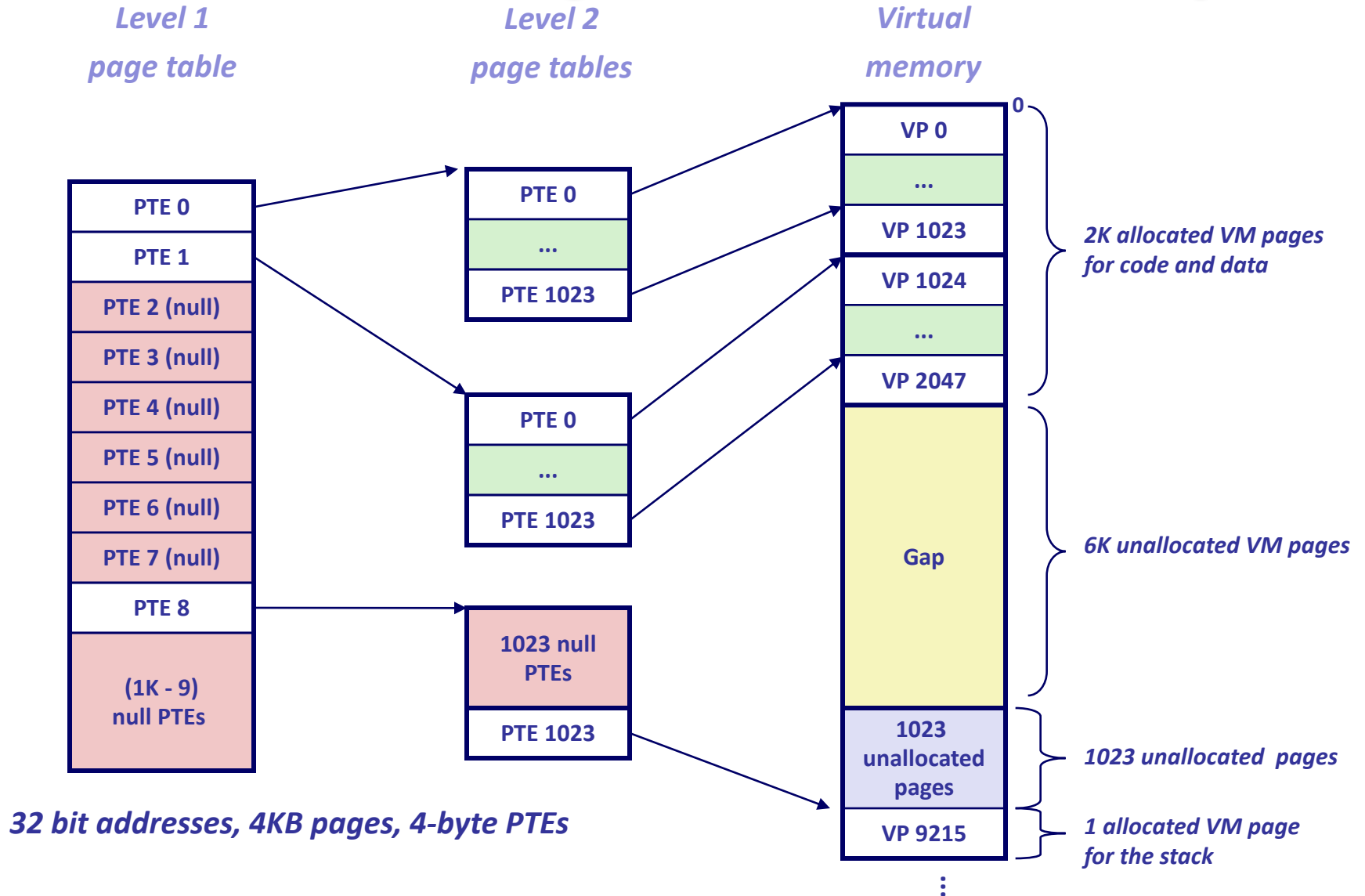
Example

- How many bits in offset? 4K = 12 bits

- 4KB pages, 4 bytes/PTE

- Want master page table in one page: 4K/4 bytes = 1K entries

- Hence, 1K secondary page tables

- How many bits?

  - Master page number = 10 bits (because 1K entries)

  - Offset = 12 bits

  - Secondary page number = 32 – 10 – 12 = 10 bits

A Two-Level Page Table Hierarchy

Level 1
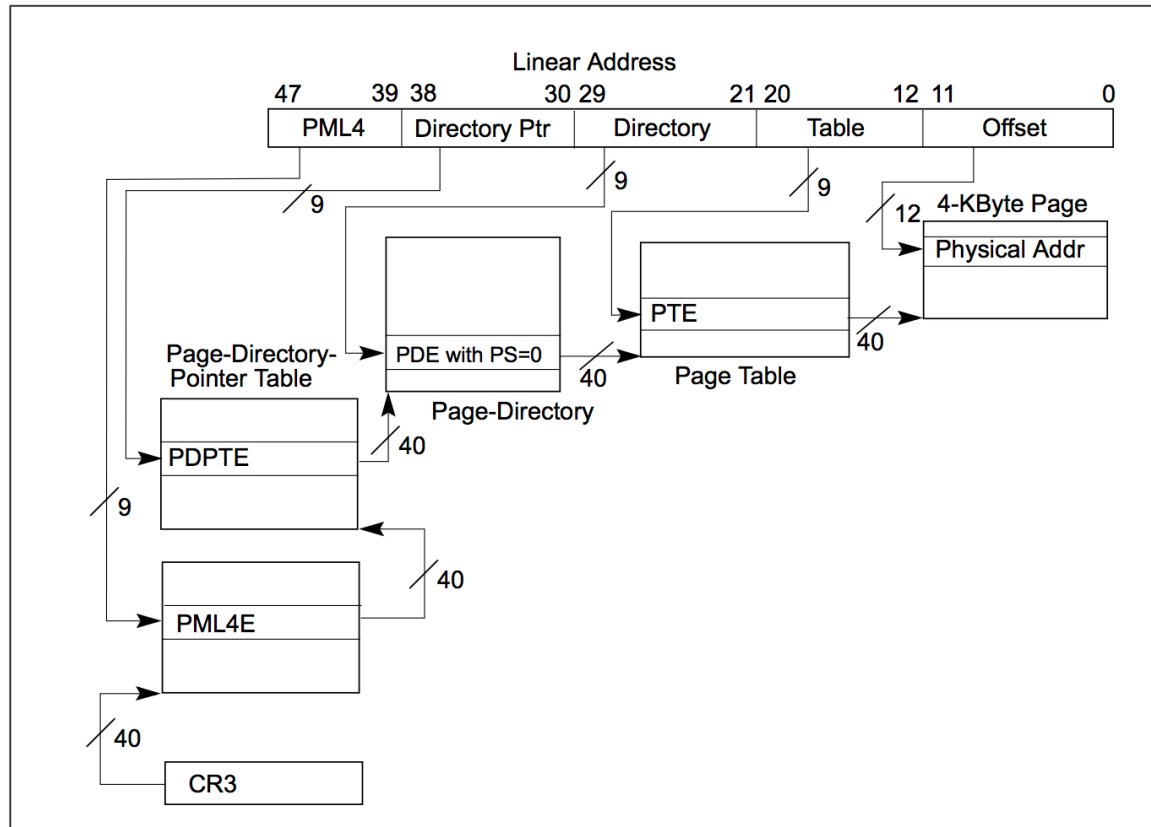**page table**

Level 2
page tables

Virtual
memory

PTE 0
PTE 1
PTE 2 (null)
PTE 3 (null)
PTE 4 (null)
PTE 5 (null)
PTE 6 (null)
PTE 7 (null)
PTE 8
(1K - 9) null PTEs

PTE 0
...
PTE 1023

PTE 0
...
PTE 1023

1023 null PTEs
PTE 1023

0

VP 0
...
VP 1023
VP 1024
...
VP 2047

2K allocated VM pages for code and data

Gap

6K unallocated VM pages

1023 unallocated pages

1023 unallocated pages

VP 9215

1 allocated VM page for the stack

32 bit addresses, 4KB pages, 4-byte PTEs

Intel IA32-e Paging

Figure 4-8. Linear-Address Translation to a 4-KByte Page using IA-32e Paging

Source: Intel Architecture Software Developer Manuals

Multi-level Paging

- Multi-level paging reduces memory overhead of paging
  - Only need one master page table and one secondary page table when a process begins
  - As address space grows, allocate more secondary page tables and add PTEs to master page table

- What problem remains?
  - Hint: what about memory lookups?

Efficient Translations

- Recall that our original page table scheme doubled the latency of doing memory lookups

  - One lookup into the page table, another to fetch the data

- Now two-level page tables triple the latency!

  - Two lookups into the page tables, a third to fetch the data

  - And this assumes the page table is in memory

- How can we use paging but also have lookups cost about the same as fetching from memory?

  - Cache (remember) translations in hardware

  - Translation Lookaside Buffer (TLB)

  - TLB managed by Memory Management Unit (MMU)