ECE 550D
Fundamentals of Computer Systems and Engineering

Fall 2022

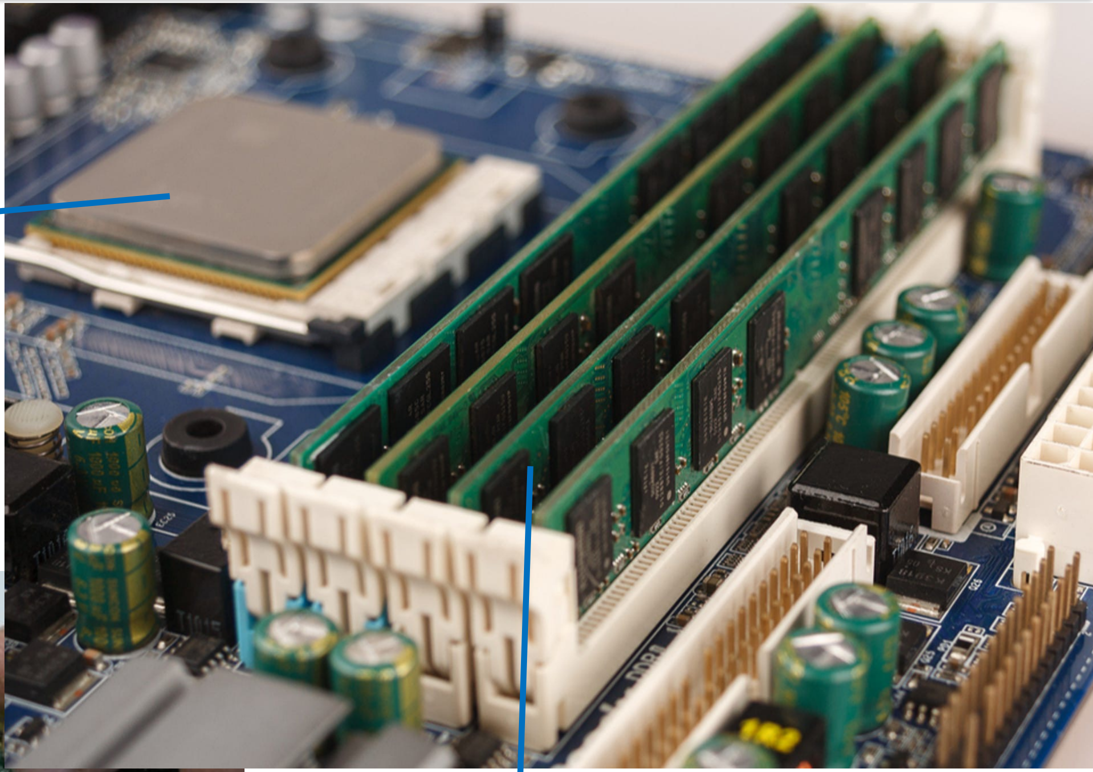
Virtual Memory

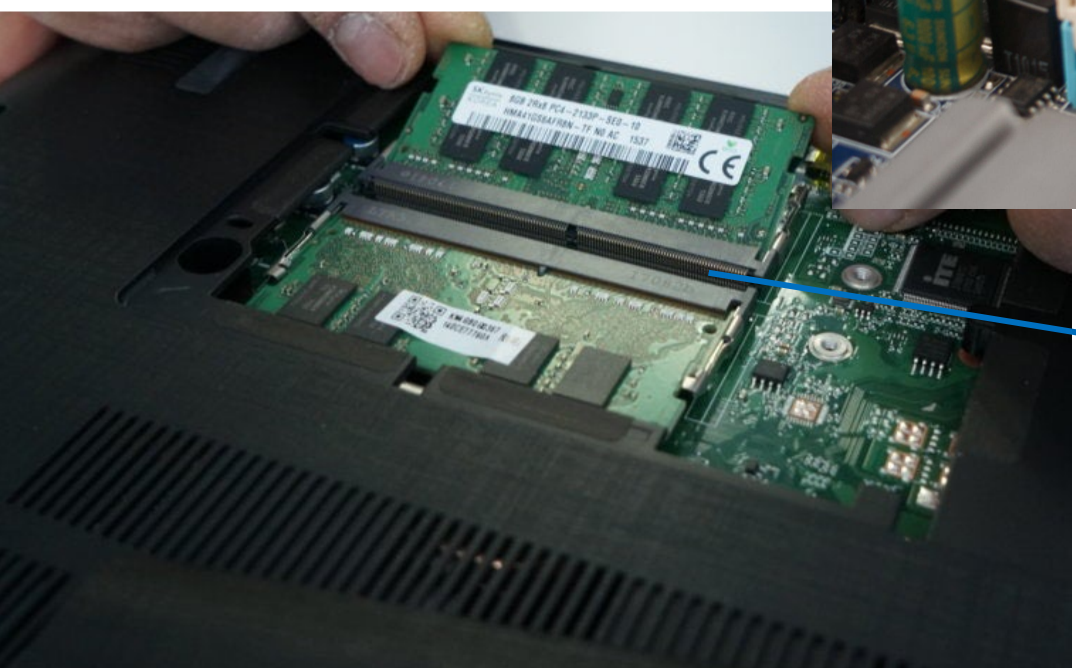
Rabih Younes

Duke University

Slides are derived from work by
Andrew Hilton and Tyler Bletsch (Duke)

Main Memory (DRAM)





Problems With Our Current Approach to Memory?

- Reasonable (main) memory: 4GB—64GB?
  - In 32-bit systems:
    - Program can address 2^32 bytes = **4GB**/program
  - In 64-bit systems:
    - Program can address 2^64 bytes = **16EB**/program!
- What if we're running many programs, not just 1?
  - Impossible using what we know for now
- → We need an approach called: **virtual memory**
  - Gives every program **the illusion** of having access to the entire address space
  - Hardware and OS (operating system) move things around behind the scenes
- How?
  - Good rule to know: when we have a functionality problem
    → we can usually solve it by adding a level of indirection

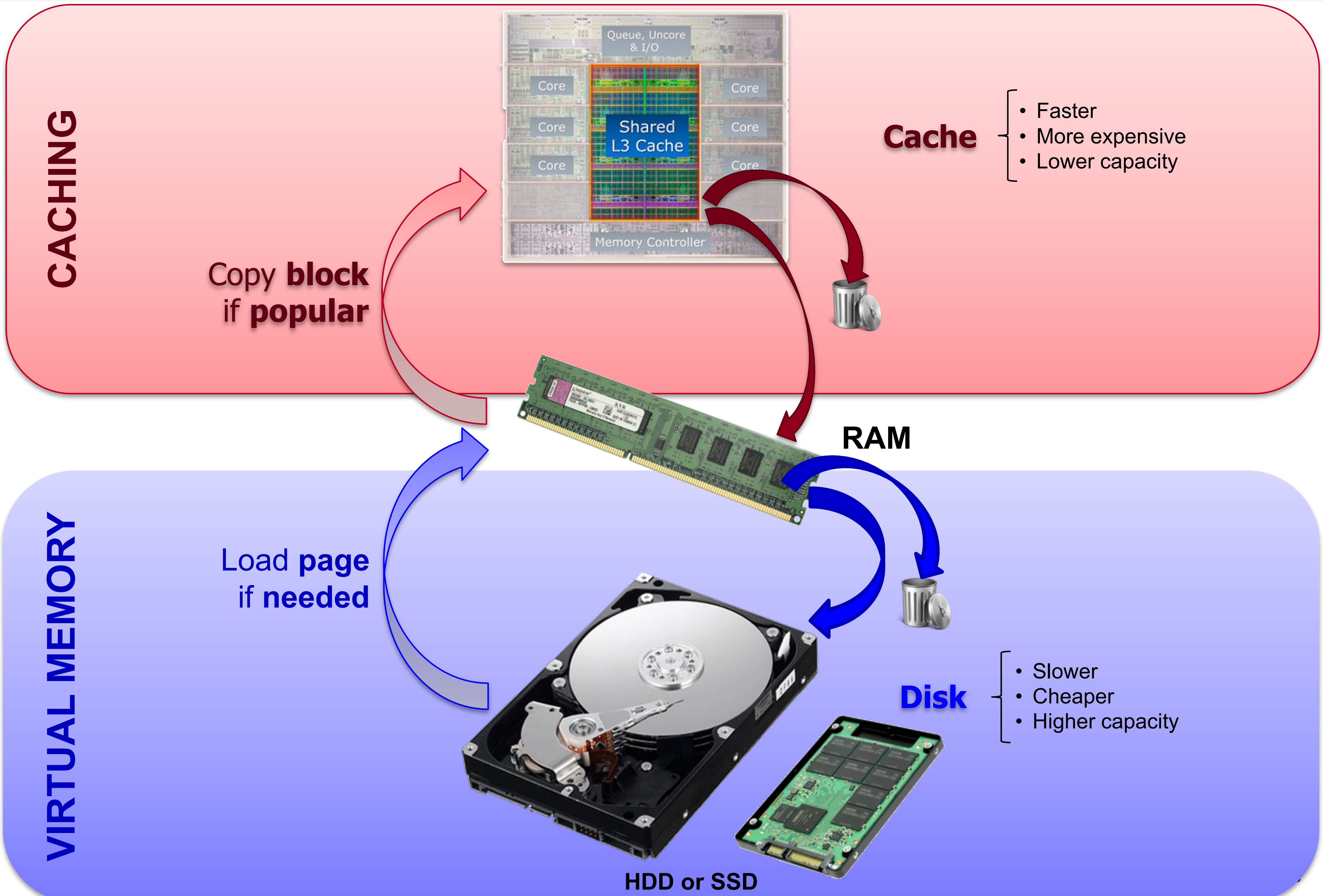
Virtual Memory

- Predates "caches" (by a little)

- Original motivation: **compatibility**
  - Ability to run the same program on machines with different main memory sizes
  - Prior to virtual memory, programmers needed to explicitly account for memory size

- **Virtual memory:**
  - Treats memory like a cache for disks (or other secondary storage)
    - Disks should be able to contain all our data
  - Contents of memory would be a dynamic subset of program's address space
  - Dynamic content management of memory is transparent to program
    - Caching mechanism makes it appear as if memory is 2^N bytes regardless of how much memory there actually is

Caching vs. Virtual Memory

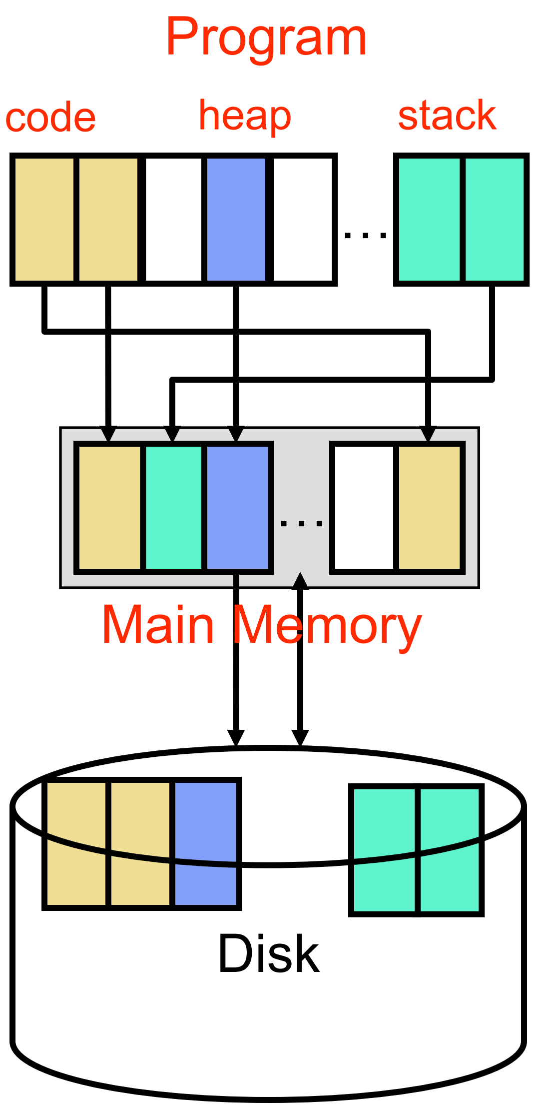


Pages

- What is swapped between disk and memory?
  - A **page** (vs. **block** in caching)
    - Using a process called "demand paging" (or "paging")

- **Page:** A small chunk (~4KB) of memory with its own record in the memory management hardware

How Virtual Memory Solves Our Problem(s)

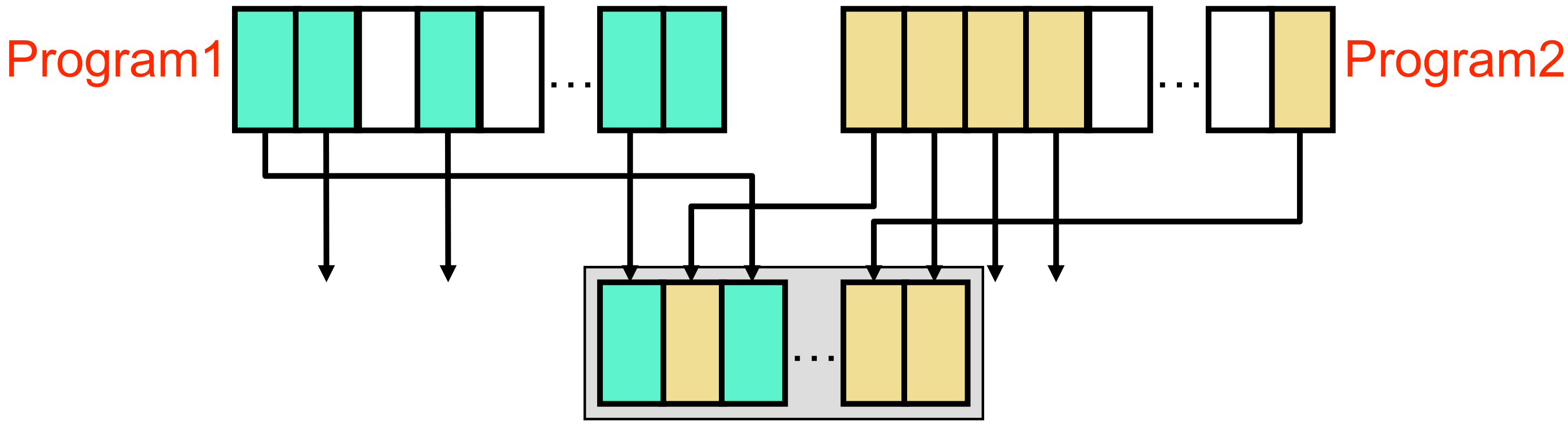


- Programs use **virtual addresses (VA)**
  - 0 to 2^N–1
  - N is the machine/system size (bus width)
    - E.g., Pentium4 is 32-bit, Core i9 is 64-bit
- Memory uses **physical addresses (PA)**
  - 0 to 2^M–1 (M<N, especially if N=64)
  - 2^M is most physical memory machine supports
- VA to PA translation at page granularity
  - → **VP to PP translation**

(Virtual Page to Physical Page)

Other Uses of Virtual Memory

- Virtual memory is quite useful for 1 program, but is also very useful for **multiprogramming** (more than 1 program)
  - Each process thinks it has 2^N bytes of address space
  - Each thinks its stack starts at address 0xFFFFFFFF
  - "System" maps VPs from different processes to different PPs
    - + Prevents processes from reading/writing each other's memory



Even More Uses of Virtual Memory

- Inter-process communication
  - Map VPs in different processes to same PPs

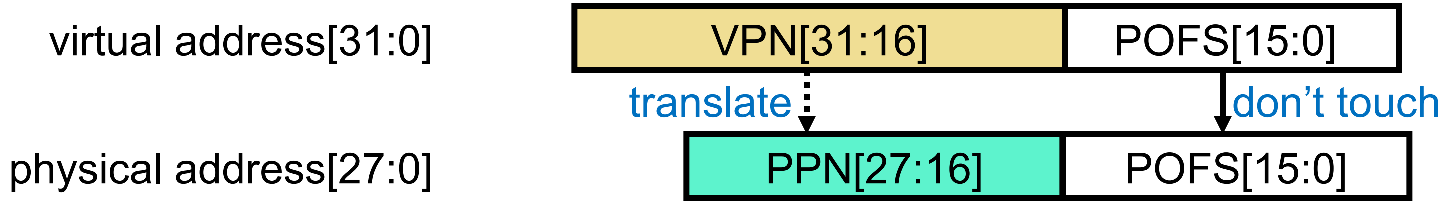
- Direct memory access I/O
  - Think of I/O device as another process
  - Will talk more about I/O in the future

- Protection
  - Piggy-back mechanism to implement page-level protection
  - Map VP to PP … and RWX protection bits
  - Attempt to execute data, or attempt to write insn/read-only data?
    - Exception → OS terminates program

Address Translation (VA→PA or VP→PP)

Address Translation

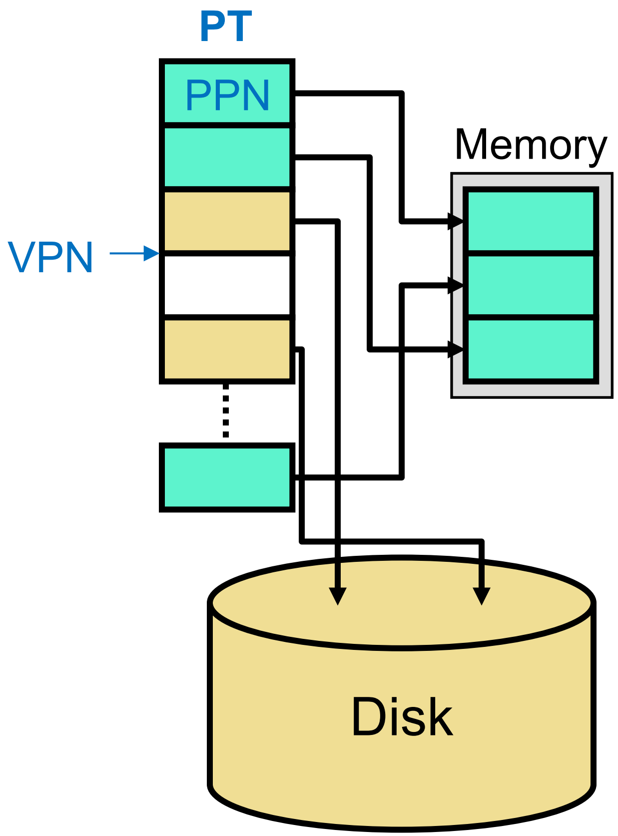
- VA→PA mapping is called **address translation**
  - Split VA into virtual page number (VPN) and page offset (POFS)
  - Translate VPN into physical page number (PPN)
  - POFS is not translated
    - Why? Because it takes us to the desired byte in a page, regardless of where that page is residing
  - VA→PA = [VPN, POFS]→[PPN, POFS]



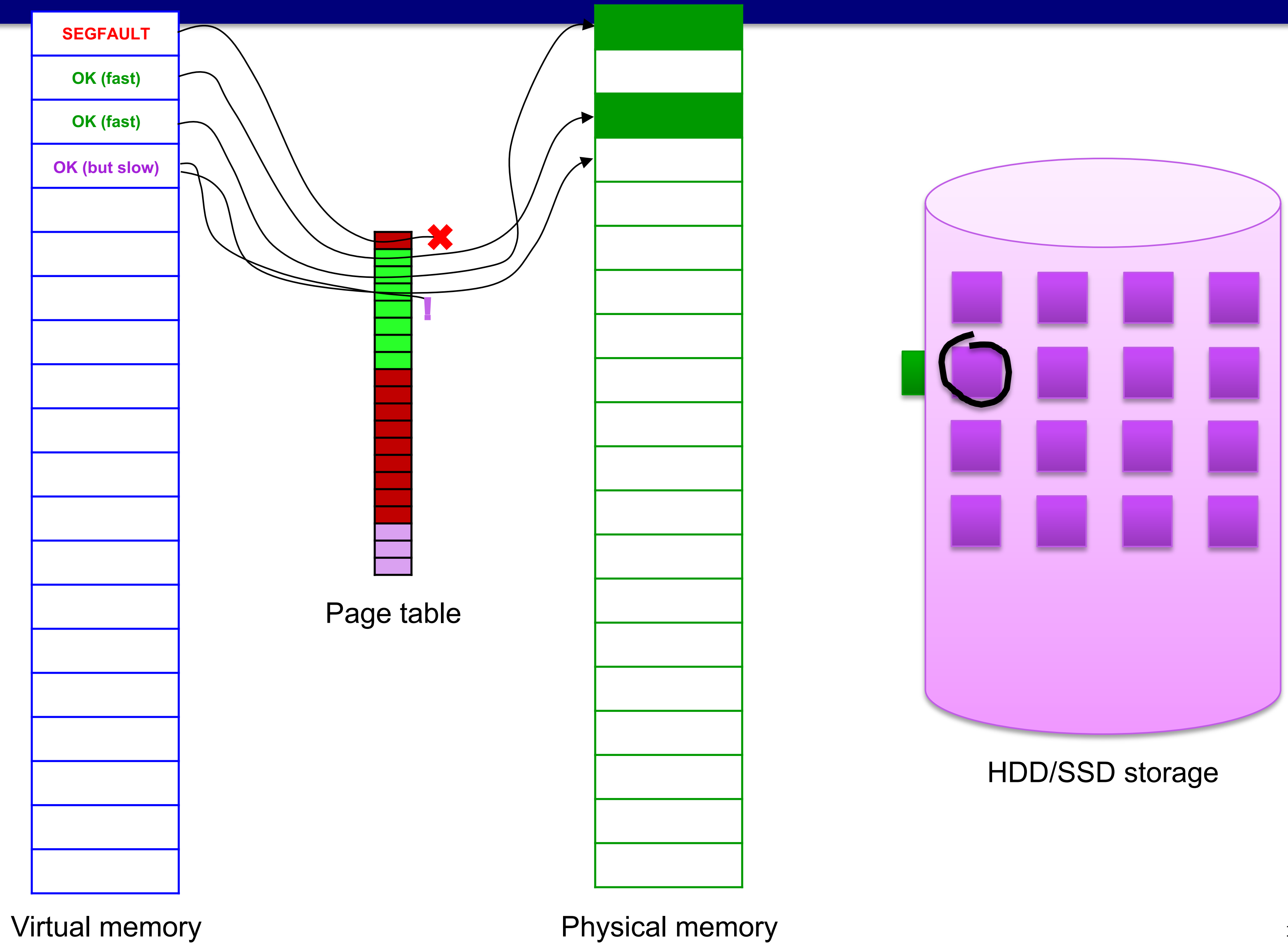
- In the example above:
  - 64KB pages? → 16-bit (= $\log_2 64K = \log_2 2^{16}$) POFS
  - 32-bit machine? → 32-bit VA → 16-bit VPN (= 32b VA – 16b POFS)
  - Maximum 256MB memory? → 28-bit PA → 12-bit PPN (= 28b – 16b)

Mechanics of Address Translation

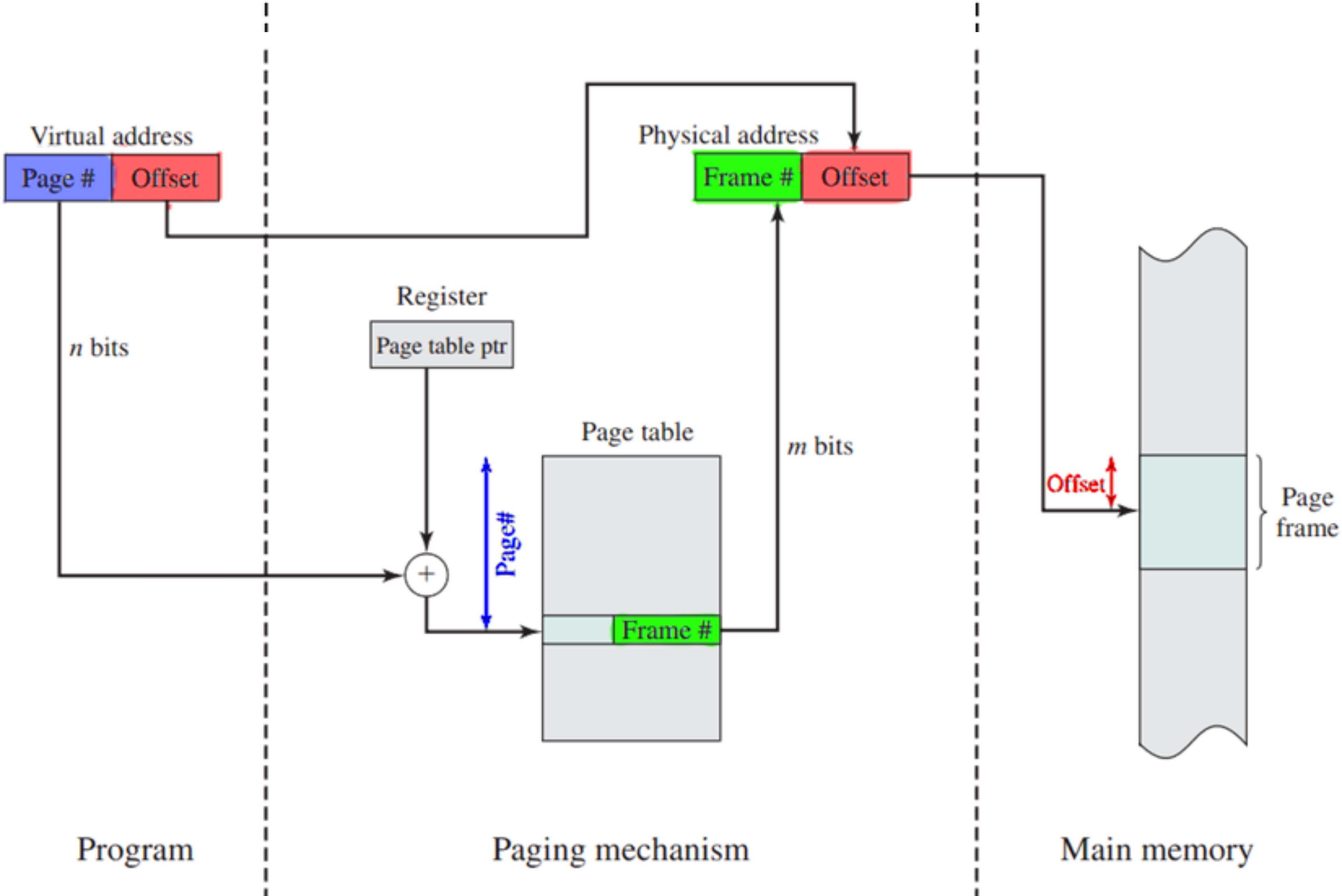
- Each process is allocated a **page table (PT)**
  - PT maps VPs to PPs or to disk addresses
    - VP entries are empty if page is never referenced
  - Translation is called **table lookup**
    - PT here is a lookup table (LUT)



High-Level Operation



Address Translation



Note that PT has to reside in memory

Adapted from Operating System Concepts by Silberschatz, Galvin, and Gagne

Address Translation

Virtual page number | Page offset

Virtual address: $\underbrace{00000000000000000111}_{\text{Virtual page number}}\underbrace{000000000101}_{\text{Page offset}}$

Page table:

Index	Data	Valid?
0	463	0
1	116	1
2	460	1
3	407	1
4	727	0
5	719	1
6	203	0
7	12	1
8	192	1

…

Physical address: $\underbrace{00000000000000001100}_{\text{Physical page number}}\underbrace{000000000101}_{\text{Page offset}}$

Structure of the Page Table

Page Table Size

- How big is a page table on the following machine?
  - 4B page table entries (PTEs)
  - 32-bit machine
  - 4KB pages

- How big would the page table be with 64KB pages?
- How big would it be for a 64-bit machine?