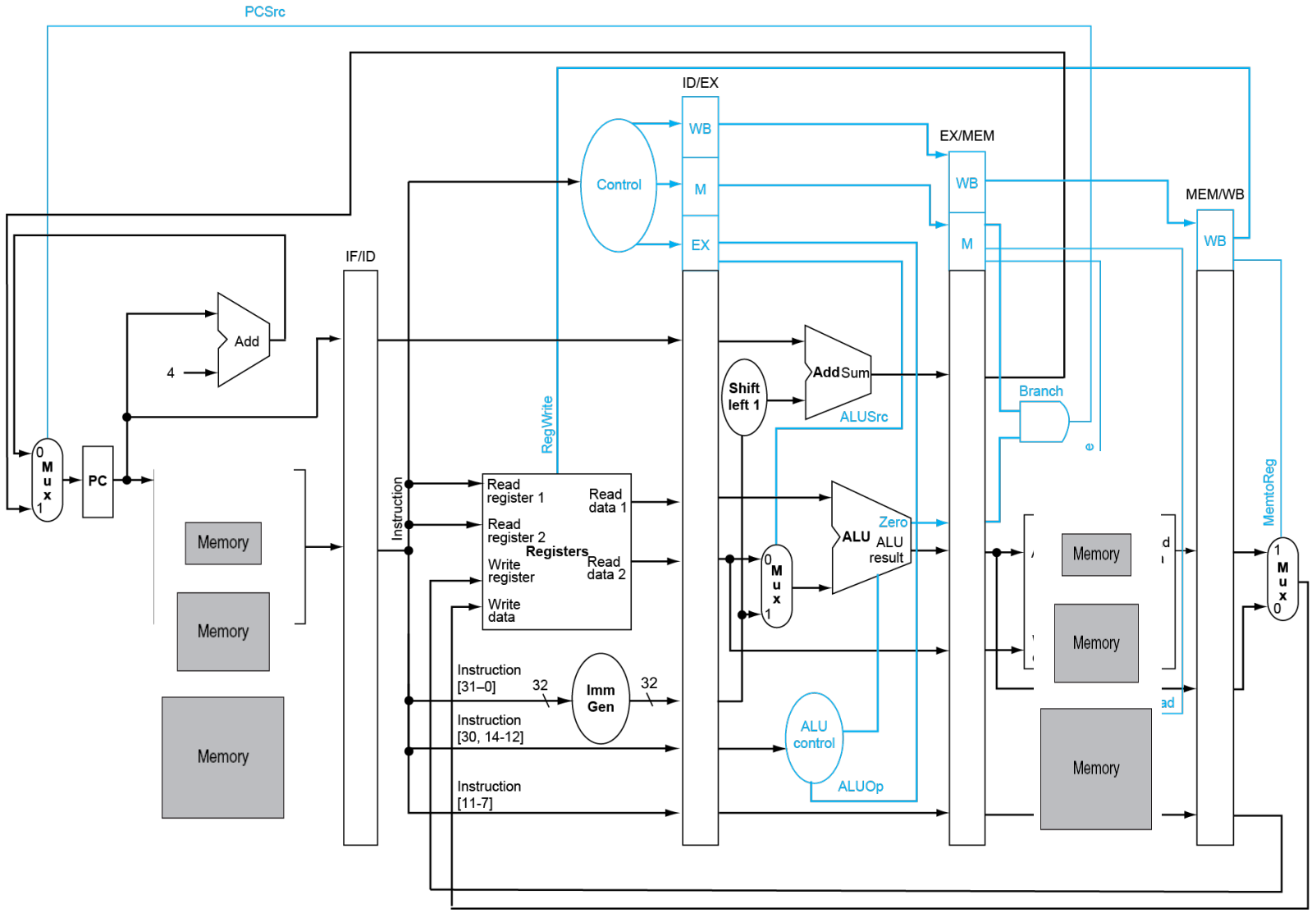
Topic 12

Memory Hierarchy

- Virtual Memory (1)

RISC-V Pipeline Architecture



Issues with Memory

- Computer may have a huge program (GByte)
  - Stored on a hard drive of tera bytes (TByte) – slow
  - But has to run on smaller cache/main memory – fast
- Computer may run multiple programs
  - Sharing the same main memory
  - We might not want them to talk to each other
- CPU interacts with memory (through cache)
  - CPU already has many other issues, doesn't want to know the complications caused by memory

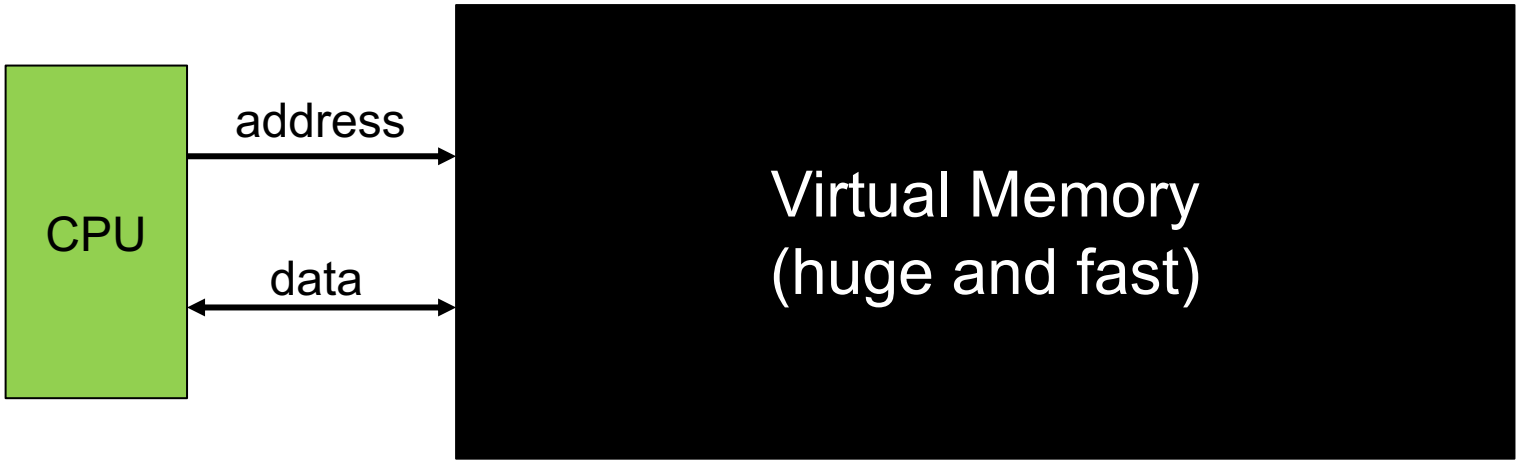
Solutions to the Issues

- Make the programmers aware of the issues
  - Write smaller program
  - Carefully allocate different main memory sections to different programs

- Well, maybe a solution decades ago!

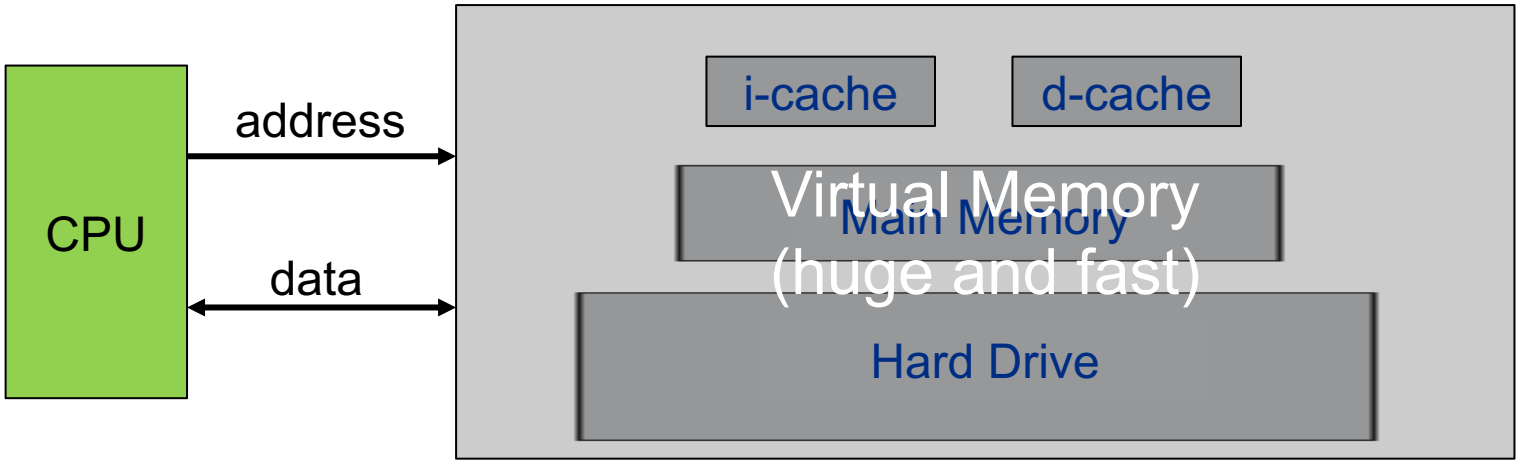
Solutions to the Issues

- Virtual Memory (VM)
  - A **huge and fast** memory from CPU's perspective, like a black box that hides details of memory
  - **Virtual (imaginary)**, but supported by **physical** memory hierarchy.

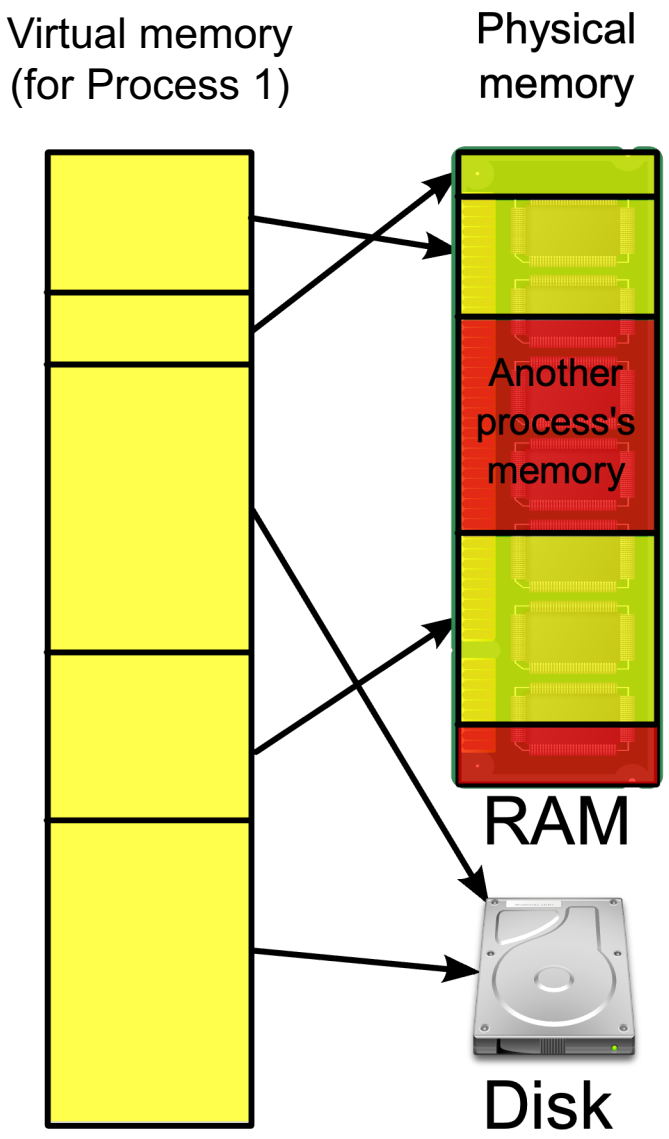


Solutions to the Issues

- Virtual Memory (VM)
  - A **huge and fast** memory from CPU's perspective, like a black box that hides details of memory
  - **Virtual (imaginary)**, but supported by **physical** memory hierarchy.



Virtual Memory is NOT Real



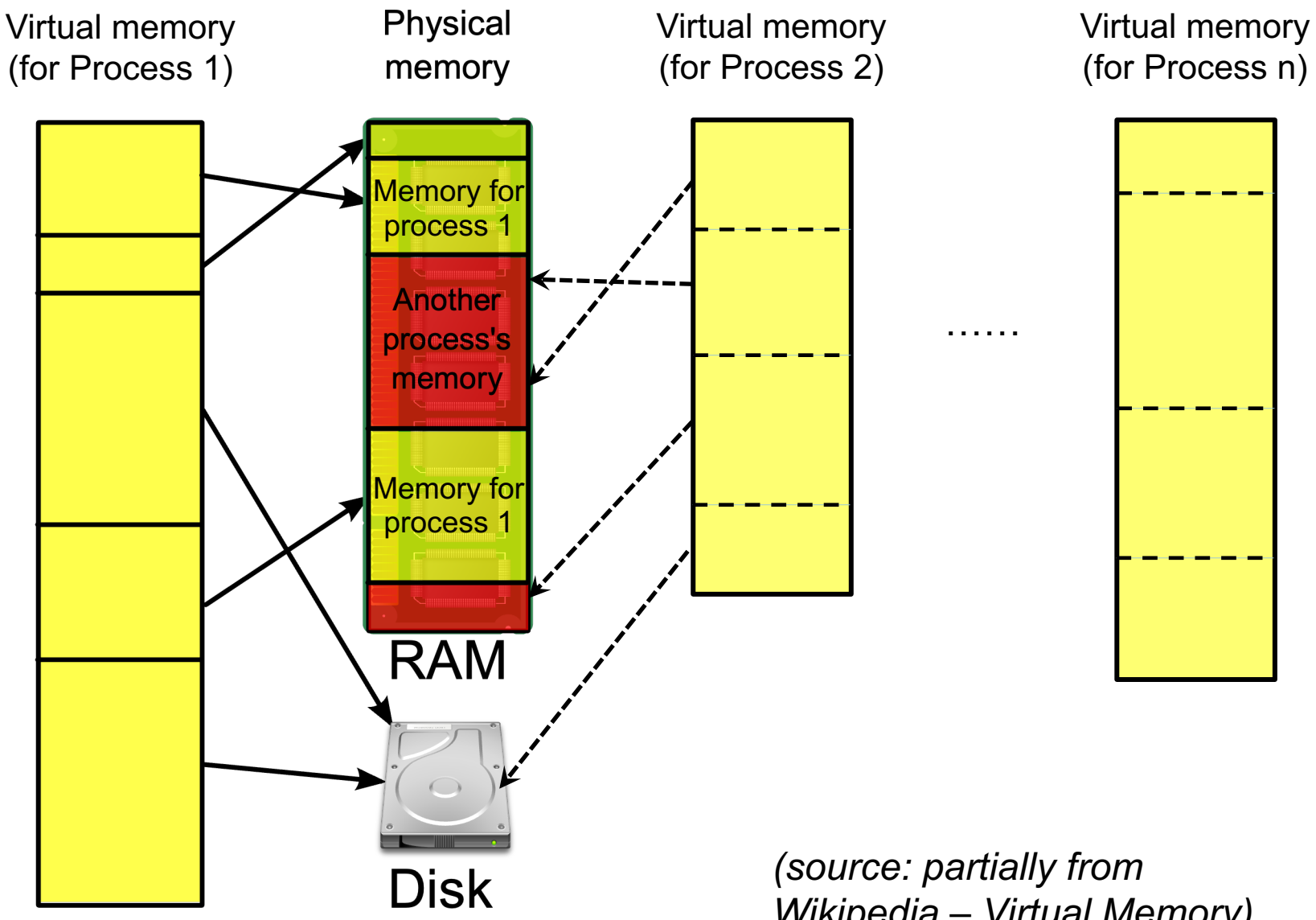
- Virtual Memory is an illusion from CPU's perspective, it's not real
- The imaginary memory has to be realized and supported by physical memory (cache, main memory, and hard drive)

(source: Wikipedia – Virtual Memory)

Solutions to the Issues

- Virtual Memory (VM)
  - Each program has a virtual (memory) space corresponding to a section of physical memory on hard drive
  - Mapping between virtual and physical is done by CPU or OS, translating specific **virtual addresses** to specific **physical addresses**

Shared Physical Memory

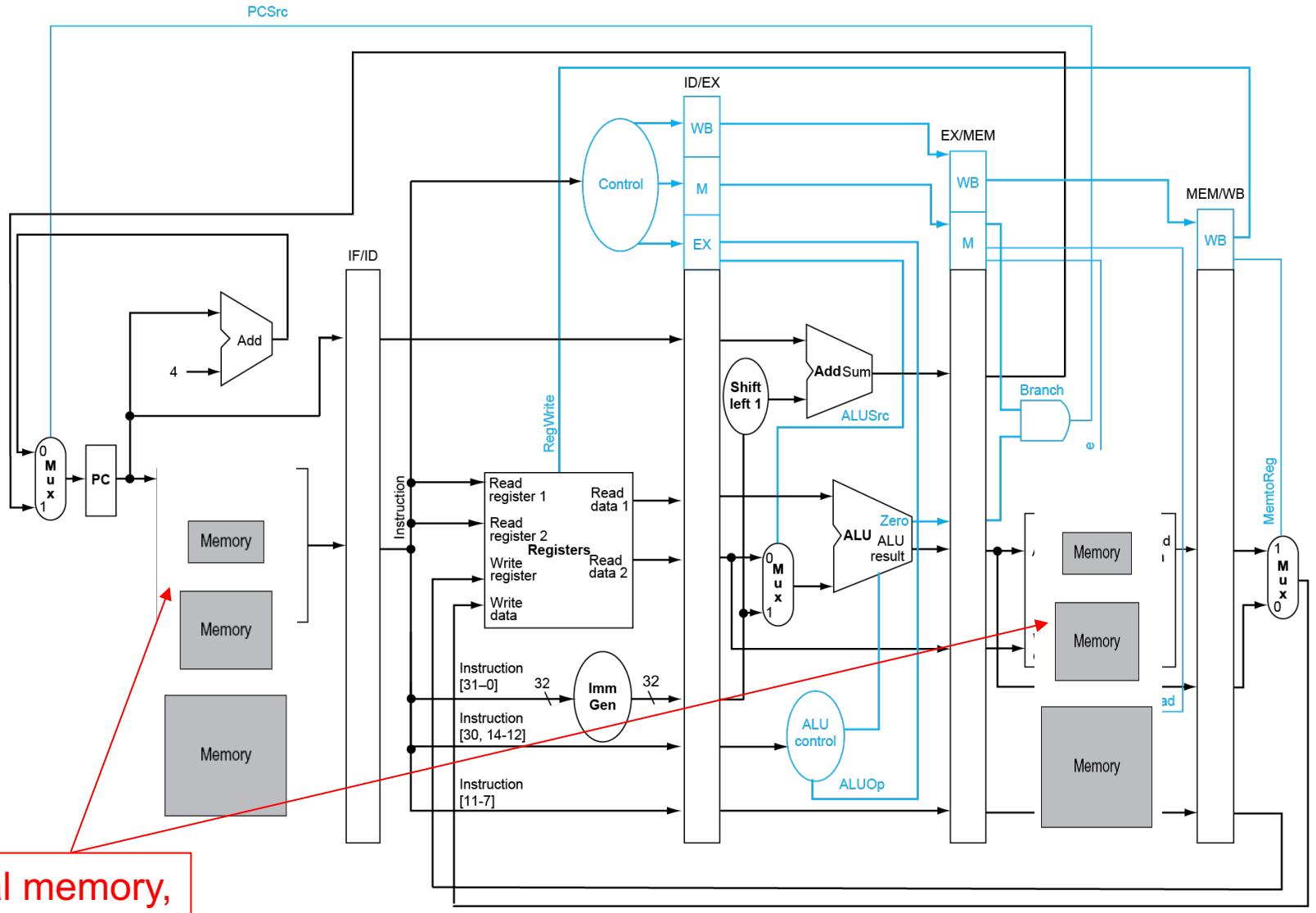


(source: partially from Wikipedia – Virtual Memory)

Summary: What is VM?

- Big
  - It can be as big as needed
  - It's an illusion of a process
- Private
  - VM is a memory that a process owns entirely
  - Each process has a separate and private VM space holding its data and instructions
- A Cover
  - It hides constrains and complications of memory from the CPU and programmer

VM in Pipeline



Virtual memory,
Huge and fast

Virtual Memory

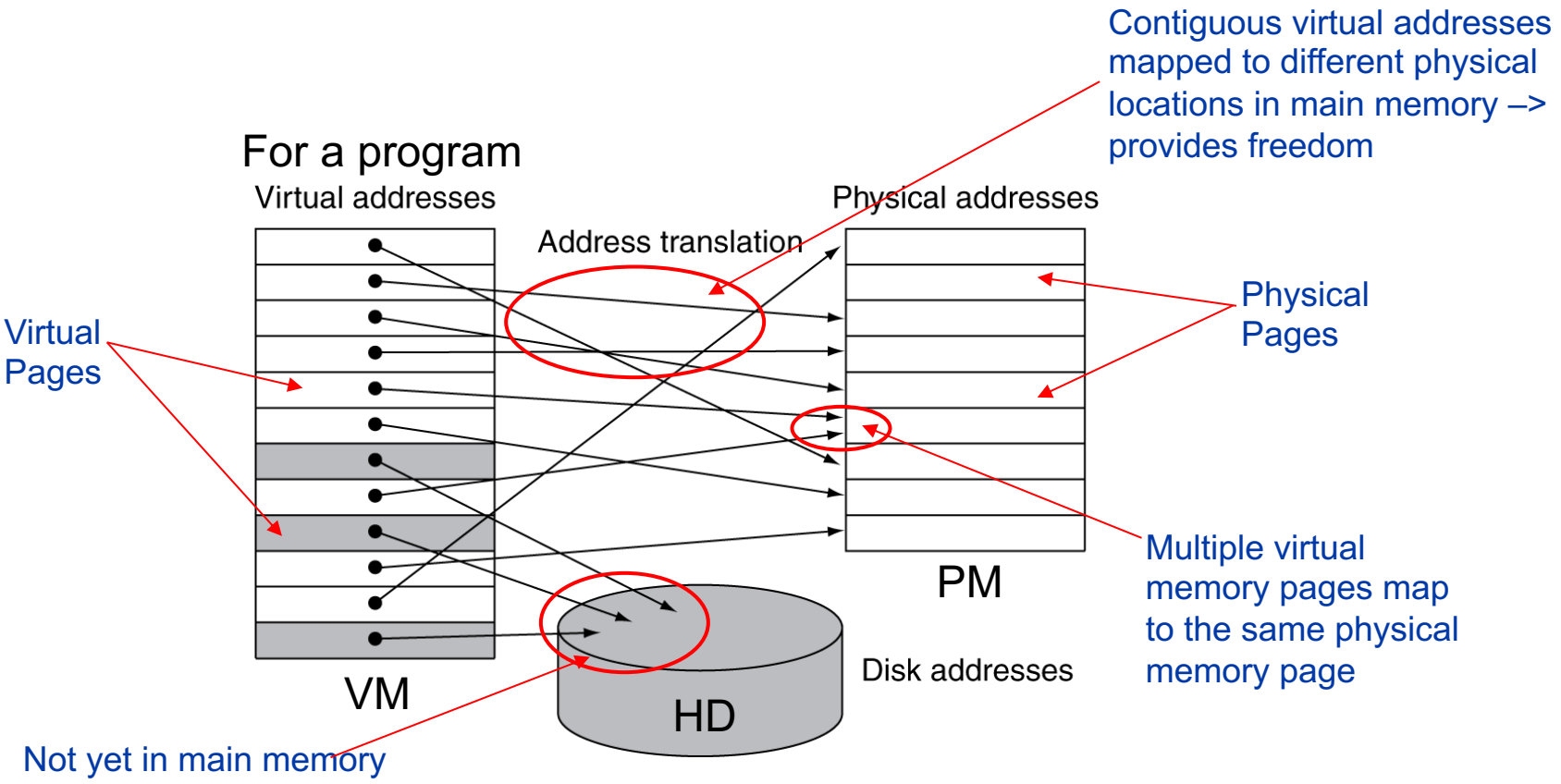
We will focus on the relationship between main memory and hard drive now

VM Terminology

- Page
  - Data transfer unit between main memory and hard drive, like “block”, but bigger than block
- Page Number
  - Index (or address) of a page
  - Virtual Page Number (VPN)
  - Physical Page Number (PPN)
- Page Offset
  - Bits to differentiate all the bytes within a page

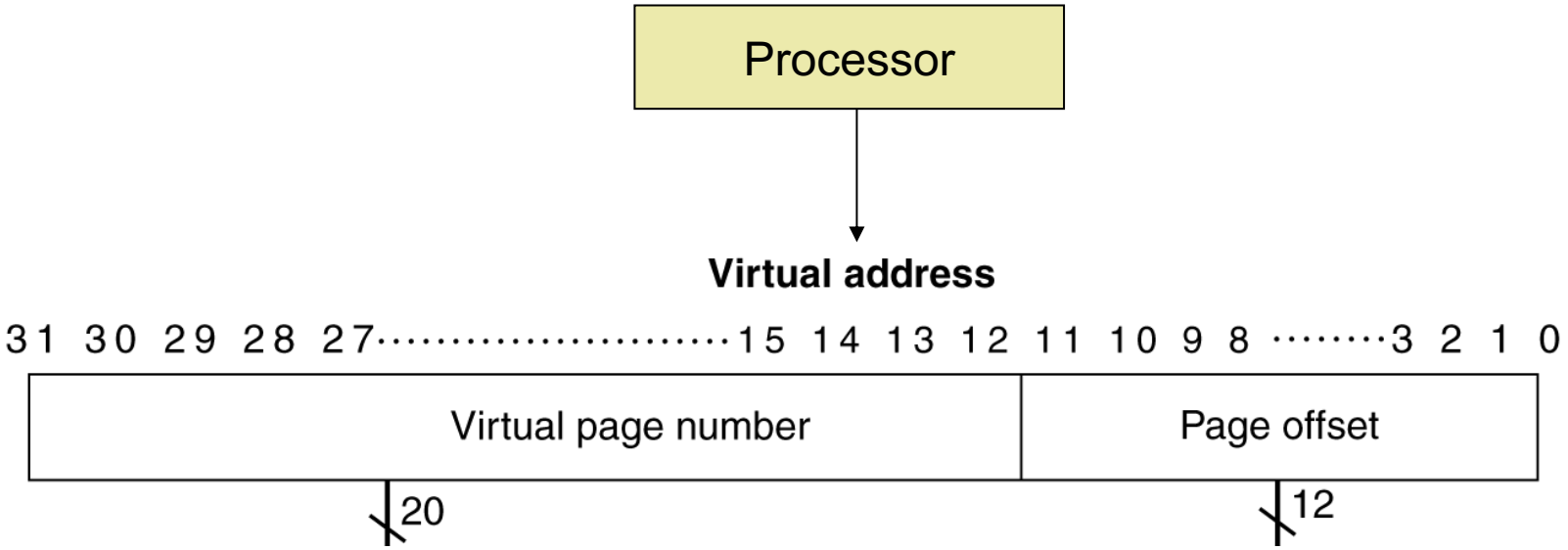
Mapping between VM and PM

- Mapping VM to PM (Physical Mem)



Virtual Address

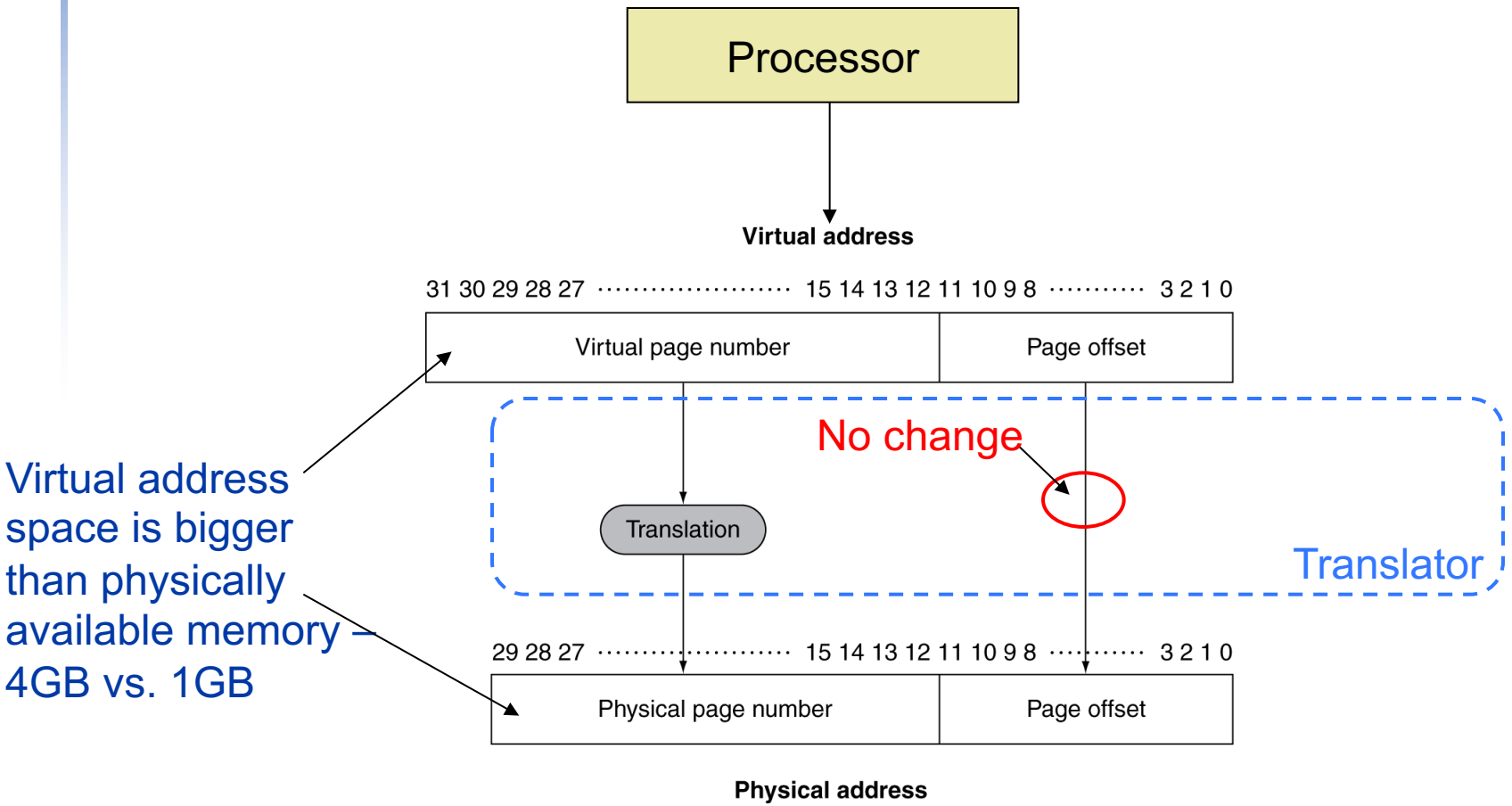
- All addresses provided by the Processor are virtual addresses



- Virtual page number is used to index the virtual pages
- Page offset: bits to differentiate all the **bytes** within a page.
  E.g., 12 bits of offset : 4K bytes

Address Translation

- Assuming fixed-size pages (e.g., 4K Bytes)

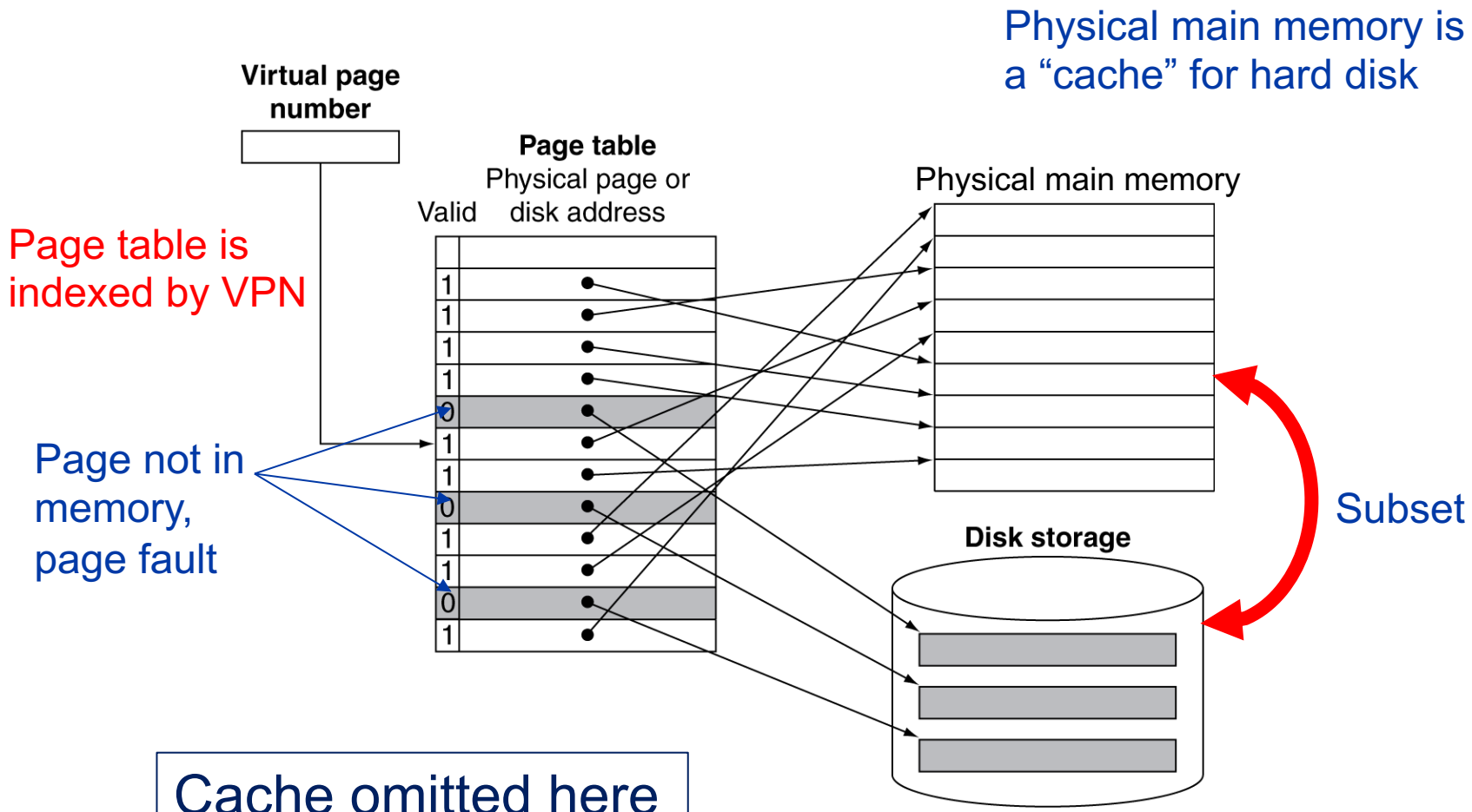


Address Mapping

Translator: Page Tables

- Each program (process) has one translator – called **Page Table**
  - Rows of the table indexed by **VPN**
  - Each row has a valid bit, plus other status bits (dirty, reference…)
  - Each row stores the mapping (translation) of a virtual address to physical addresses
  - Located in main memory
  - A **page table register (PTR) or page table base register (PTBR)** in CPU points to the beginning of the page table for the program that is currently running

Page Table Translation

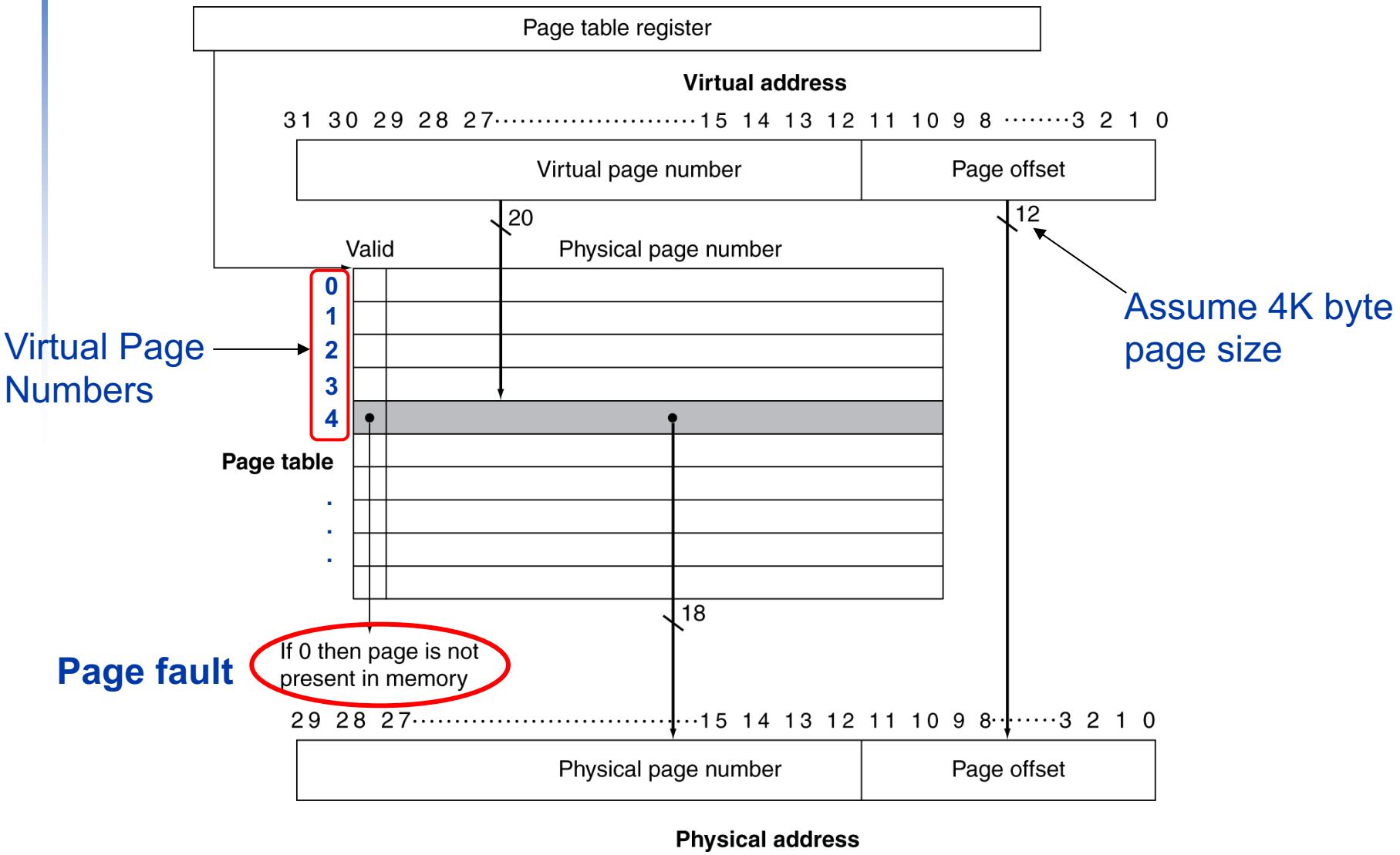


Cache omitted here for now!

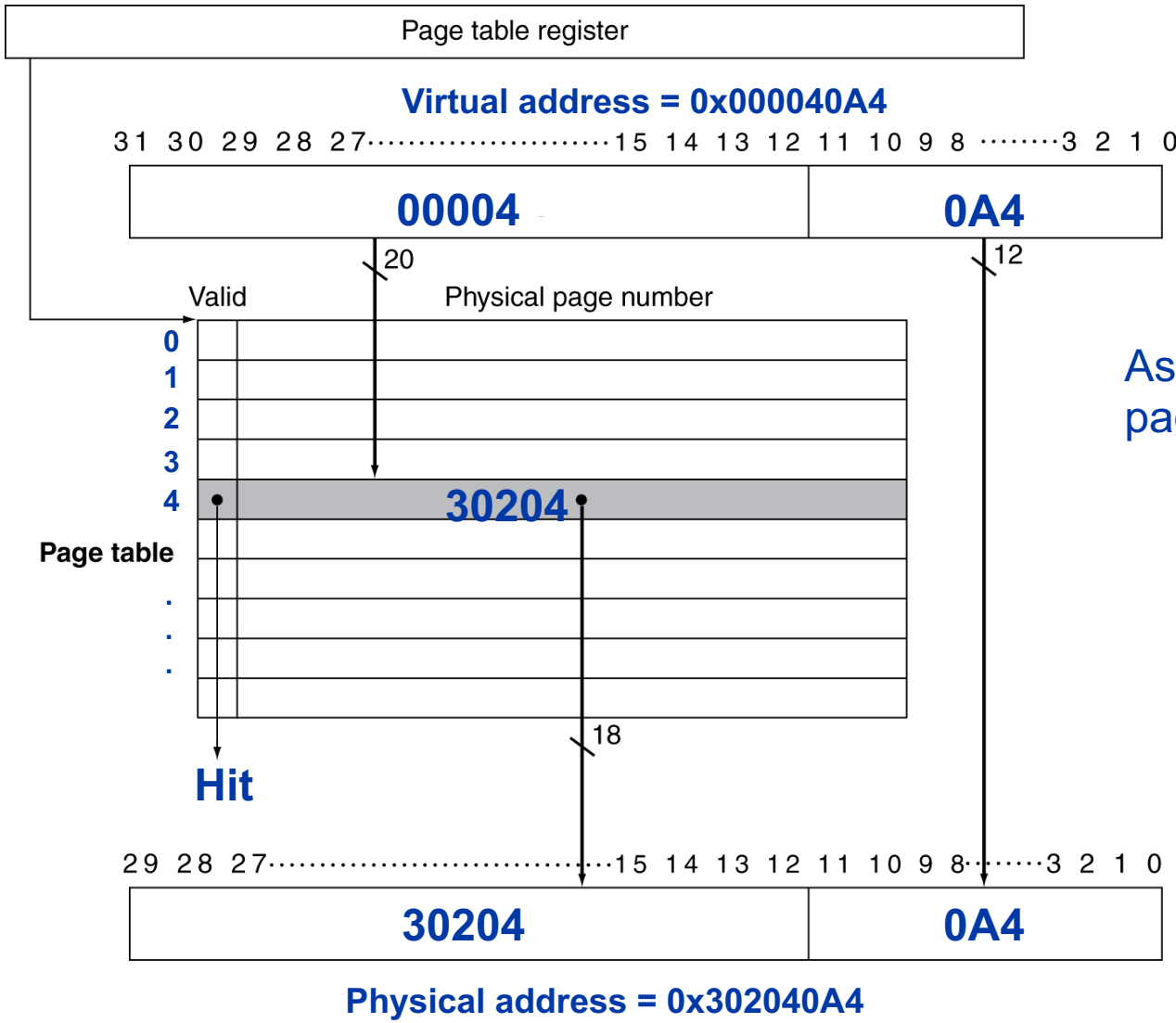
Page Table

- If page is present in main memory
  - Page table stores the physical page address of the main memory
  - Valid bit is set
  - Plus other status bits (dirty, reference…)
- If page is not in main memory – **page fault**
  - It's on hard drive (disk)
  - All virtual pages for a program are stored in a unique **swap space** on disk
  - Page table can refer to locations in the swap space of a program on disk

Translation Using a Page Table



Example: Translate Virtual to Physical



Assume 4K byte page size

Page Table Size

- Example:
  - Page size: 4KB
  - 32-bit virtual byte address (4G Bytes)
  - 4 bytes per page table entry
- Number of page table entries = number of virtual pages = $2^{(32-12)} = 2^{20}$
  - Page table indexed by virtual page numbers
- Size of page table = number of page table entries x bytes/page table entry
  - Page table size = 2^{20} x 4 = 4 MB

Page Fault

- Page Fault
  - Valid bit in a row of the page table is 0
  - Requested page is not available in main memory
    - VPN doesn’t have a translation to PPN
- What should we do on page fault?

Handling Page Fault

- Like a “miss” in cache
- On page fault
  - Find the page on disk
  - Fetch and put it in main memory
- Fetching a page from disk to main memory is very expensive
  - Takes millions of clock cycles
  - Should be handled by OS – more sophisticated and less expensive
- Should try to minimize page fault rate

Reduce Page Fault Rate and Penalty

- Main memory should
  - Have **large page size**, so one access fetches more data, also reduces page fault rate
    - Most of the time is for getting the first word in the page – access time very long
    - May also reduce page fault rate
  - Reduce page fault rate by **full associativity in main memory**
  - Use **write-back** technique

Page Writes

- Disk writes take millions of cycles
  - Write through is impractical, even with write buffer
    - Millions of processor clock cycles
  - Use **write-back**
    - Dirty bit in page table is set when page is written
    - Write-back first if dirty bit is on
    - Writing entire page is more time efficient than writing a word
  - CPU switches to another process/program while waiting – **context switch**

Page Replacement

- A page in main memory need be replaced when the main memory is full
- Least-recently used (LRU) replacement
  - Lower page fault rate – temporal locality
  - **Reference bit** (aka use bit) in page table
    - Set to 1 on access to page
    - Periodically cleared to 0 by OS
    - A page with reference bit = 0, means it has not been used recently – to be replaced

Class Exercise

- Given
  - 4KB page size, 16KB physical memory, LRU replacement
  - Virtual address: byte addressable, 20 bits (how many bytes?)
  - Page table for program A stored in page #0 of physical memory, starting at address 0x0100, assume only 2 valid entries in page table:
    - Virtual page number 0 => physical page number 1
    - Virtual page number 1 => physical page number 2
- Show physical memory including page table
- Complete following table

Virtual Address	Virtual page number	Page fault?	Physical Address
0x00F0C			
0x01F0C			
0x20F0C			
0x00100			
0x00200			
0x30000			
0x01FFF			
0x00200			