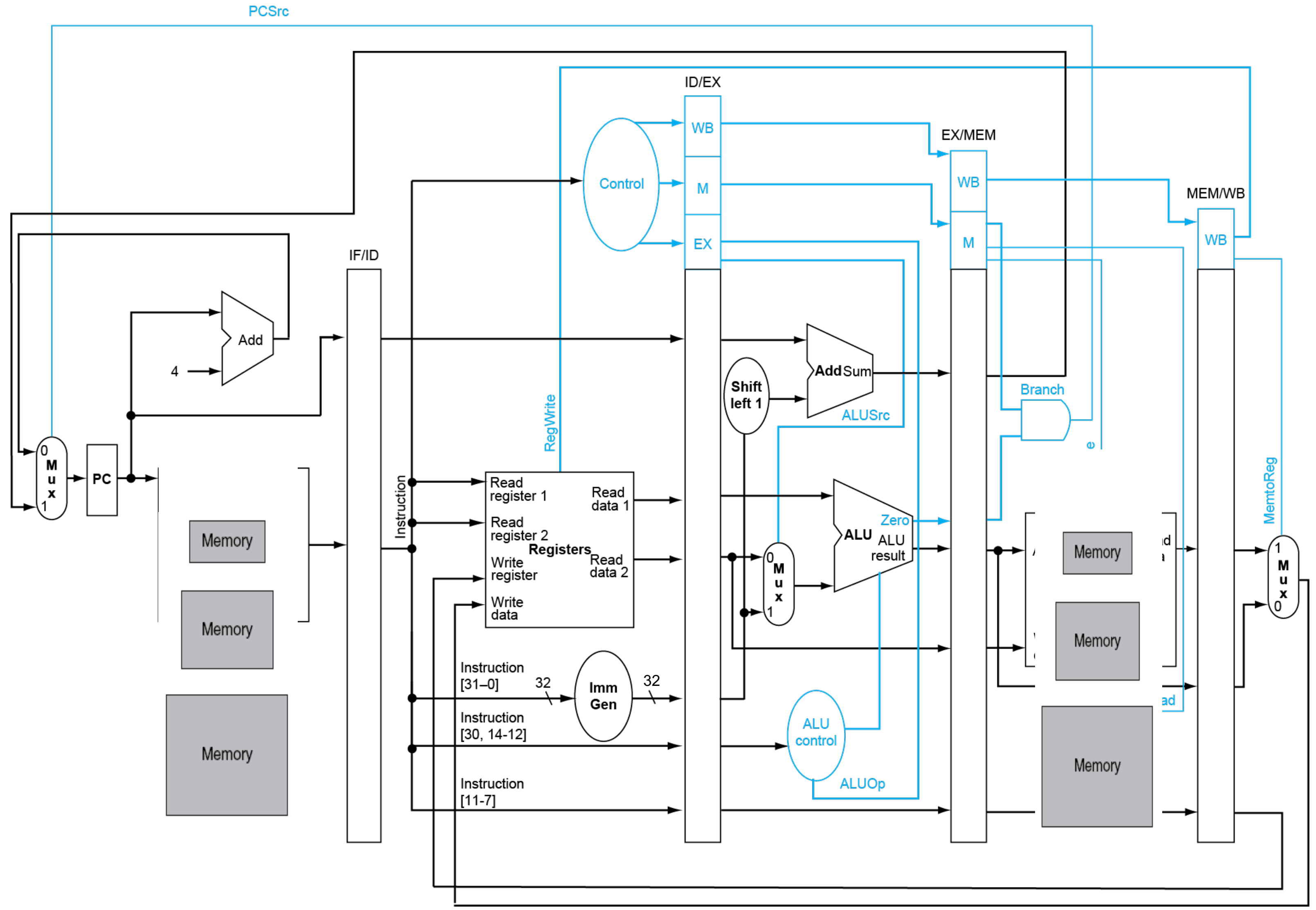
Topic 12

Memory Hierarchy

- Virtual Memory (1)

RISC-V Pipeline Architecture



Issues with Memory

- Computer may have a huge program (GByte)
  - Stored on a tera bytes of hard drive (TByte) – slow
  - But has to run on a Mbyte main memory – fast
- Computer may run multiple programs
  - Sharing the same main memory
  - We don’t want them to talk to each other
- CPU interacts with main memory (through cache)
  - CPU already has many other issues, doesn’t want to be bothered by issues brought by memory

Solutions to the Issues

- Make the programmer 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)
  - An **imaginary, huge and fast** memory from CPU’s perspective – mapped to **physical** memory
  - Each program has a virtual (memory) space corresponding to a section of physical memory on hard drive
  - Mapping is done by CPU or OS translating specific **virtual addresses** to specific **physical addresses**

What is Virtual Memory (VM)

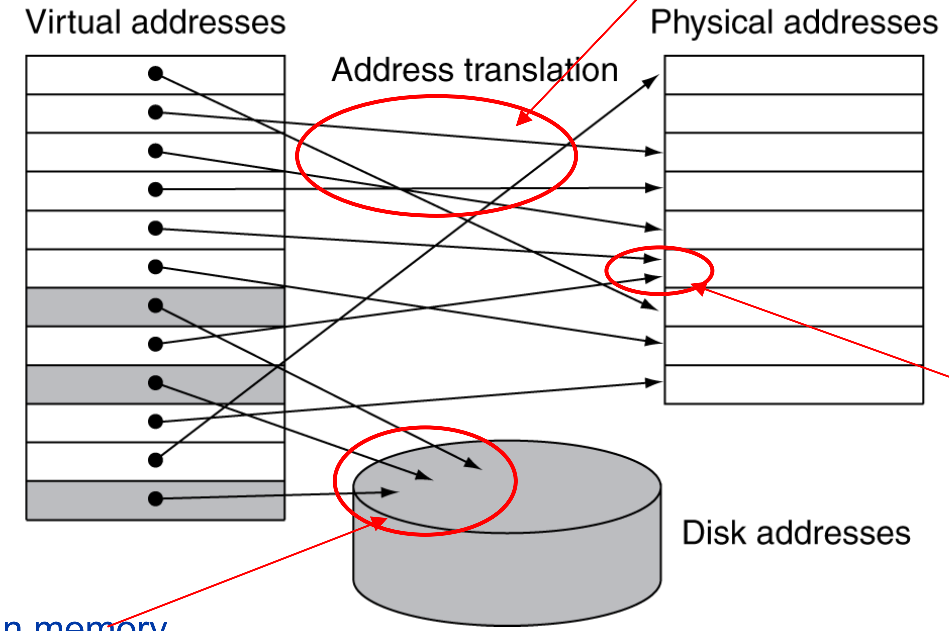
- Big
  - It can be as big as needed
  - It’s an illusion of a process
- Private
  - VM is a memory that a process owns entirely
  - Each process has a separate and private VM space holding its data and instructions
- A Cover
  - It hides constrains and complications of memory from the CPU and programmer

VM Terminology

- Use main memory as a “cache” for hard disk
- Concepts in VM
  - VM “block” is called a **page** (bigger)
  - VM “miss” is called a **page fault**

Contiguous virtual addresses mapped to different physical locations in main memory –> provides freedom

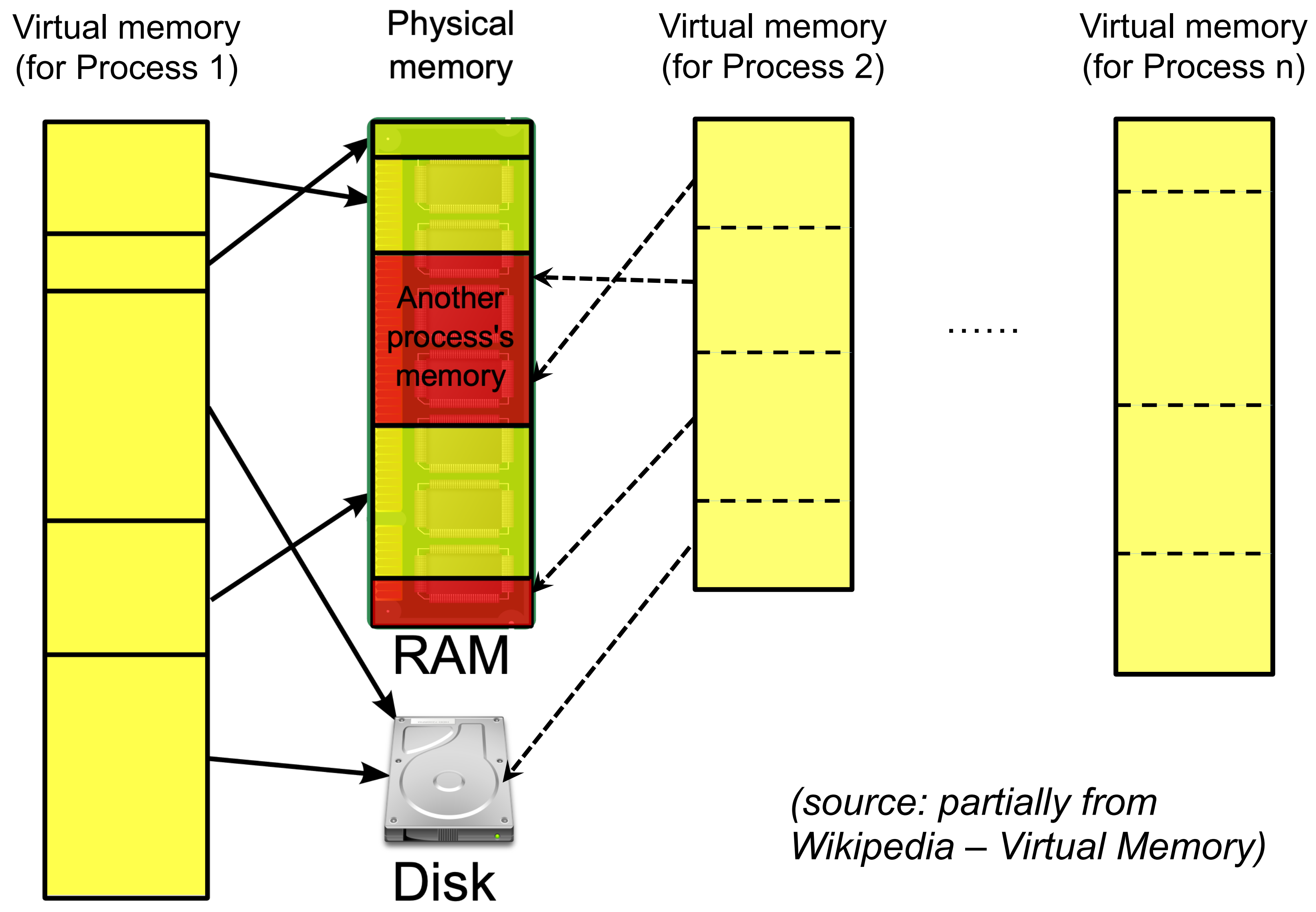
For a program



Multiple virtual memory pages map to the same physical memory page

Not yet in main memory

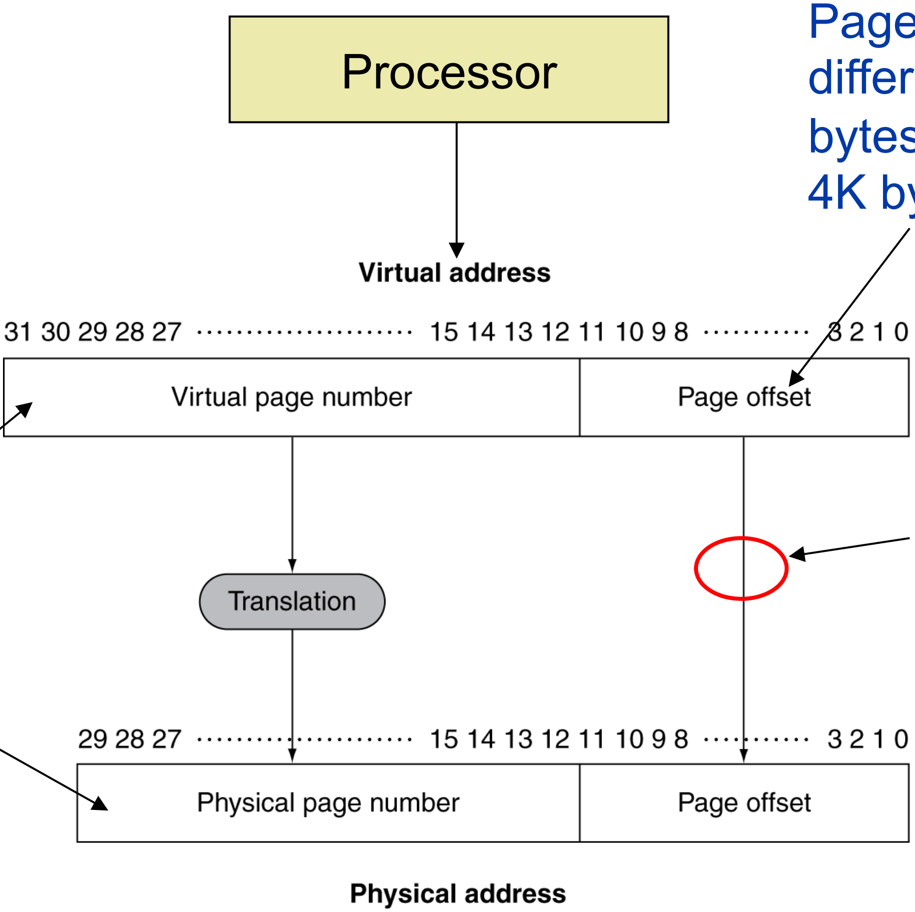
What is Virtual Memory (VM)


Virtual memory (for Process 1)
Physical memory
Virtual memory (for Process 2)
Virtual memory (for Process n)
Another process's memory
RAM
Disk
……

(source: partially from Wikipedia – Virtual Memory)

Address Translation

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



Page offset bits are to differentiate all the bytes within the page. 4K bytes: 12 bits offset

No change

Virtual address space is bigger than physically available memory – 4GB vs. 1GB

Address Mapping

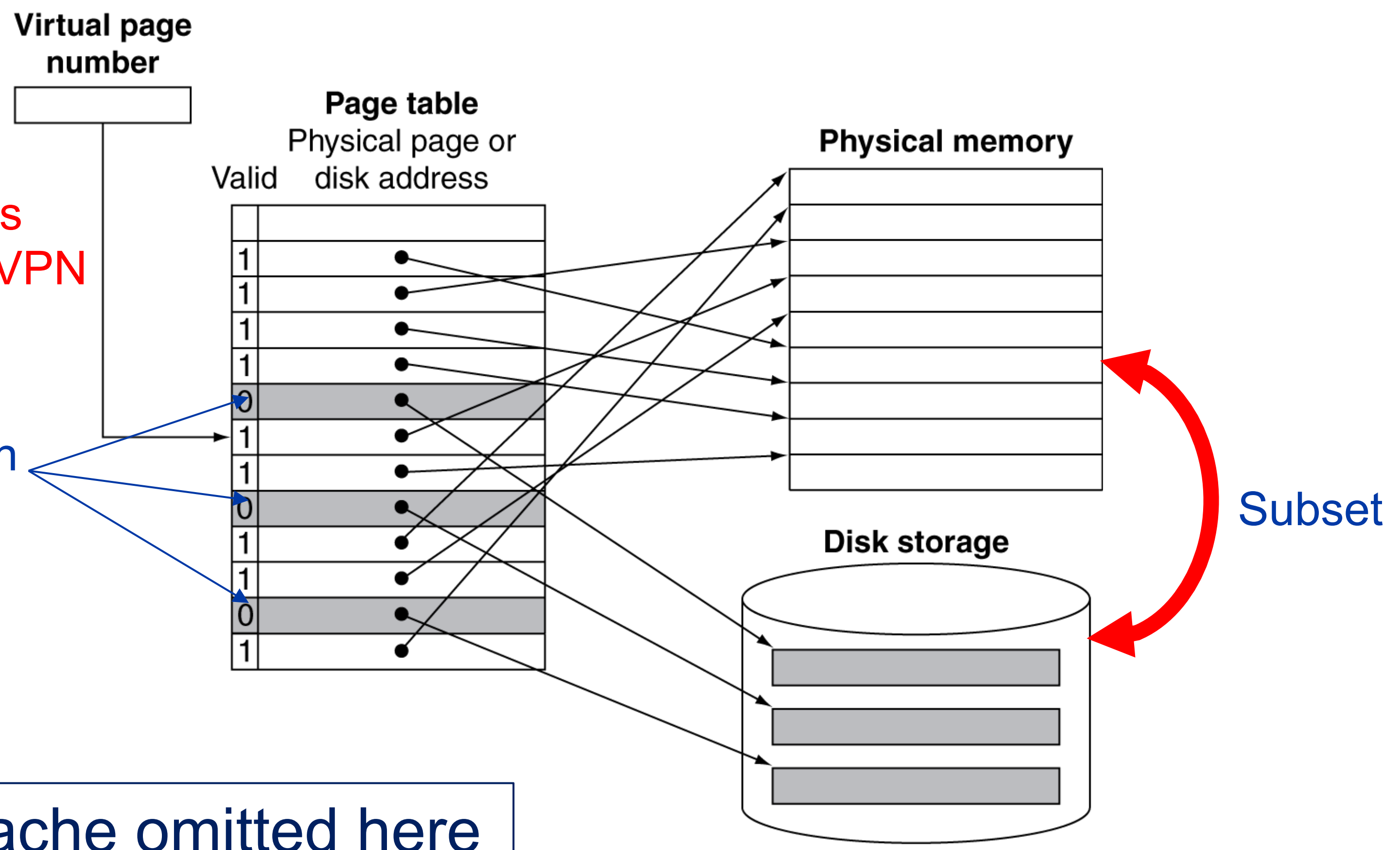
Translator: Page Tables

- Each program has one translator – page table
  - Stores the mapping (translation) of all virtual to physical addresses
  - Indexed by **virtual page numbers (VPN)**
  - Located in main memory
  - A **page table register (PTR) or page table base register (PTBR)** in CPU points to the beginning of page table for the program that is currently running

Page Table

- If page is present in memory
  - Page table stores the physical page address
  - Plus other status bits (valid, dirty, …)
- If page is not present in memory – **page fault**
  - All virtual pages for a program are mapped to a unique **swap space** on disk
  - Page table can refer to locations in swap space of a program on disk

Mapping Pages to Storage

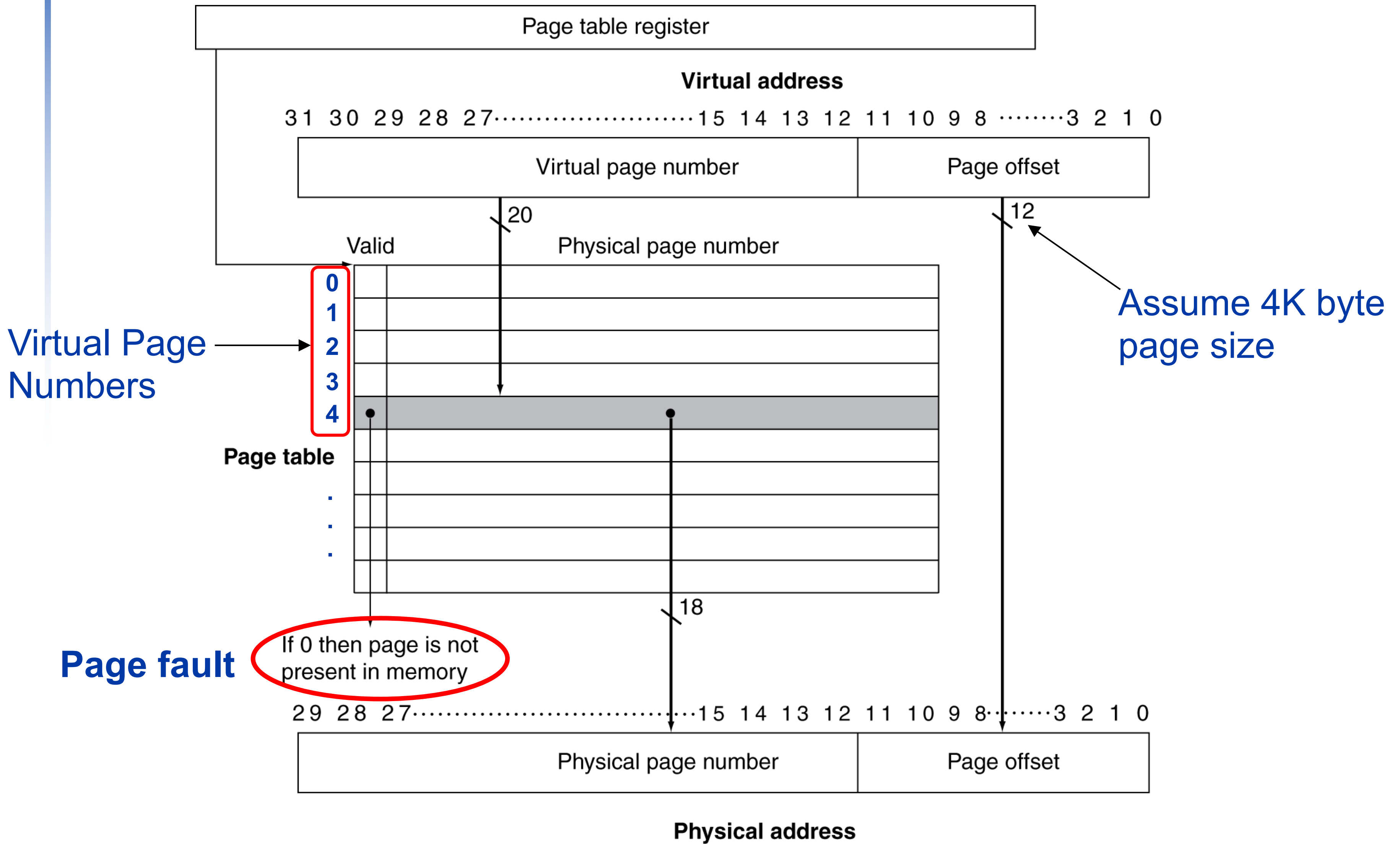


Page table is indexed by VPN

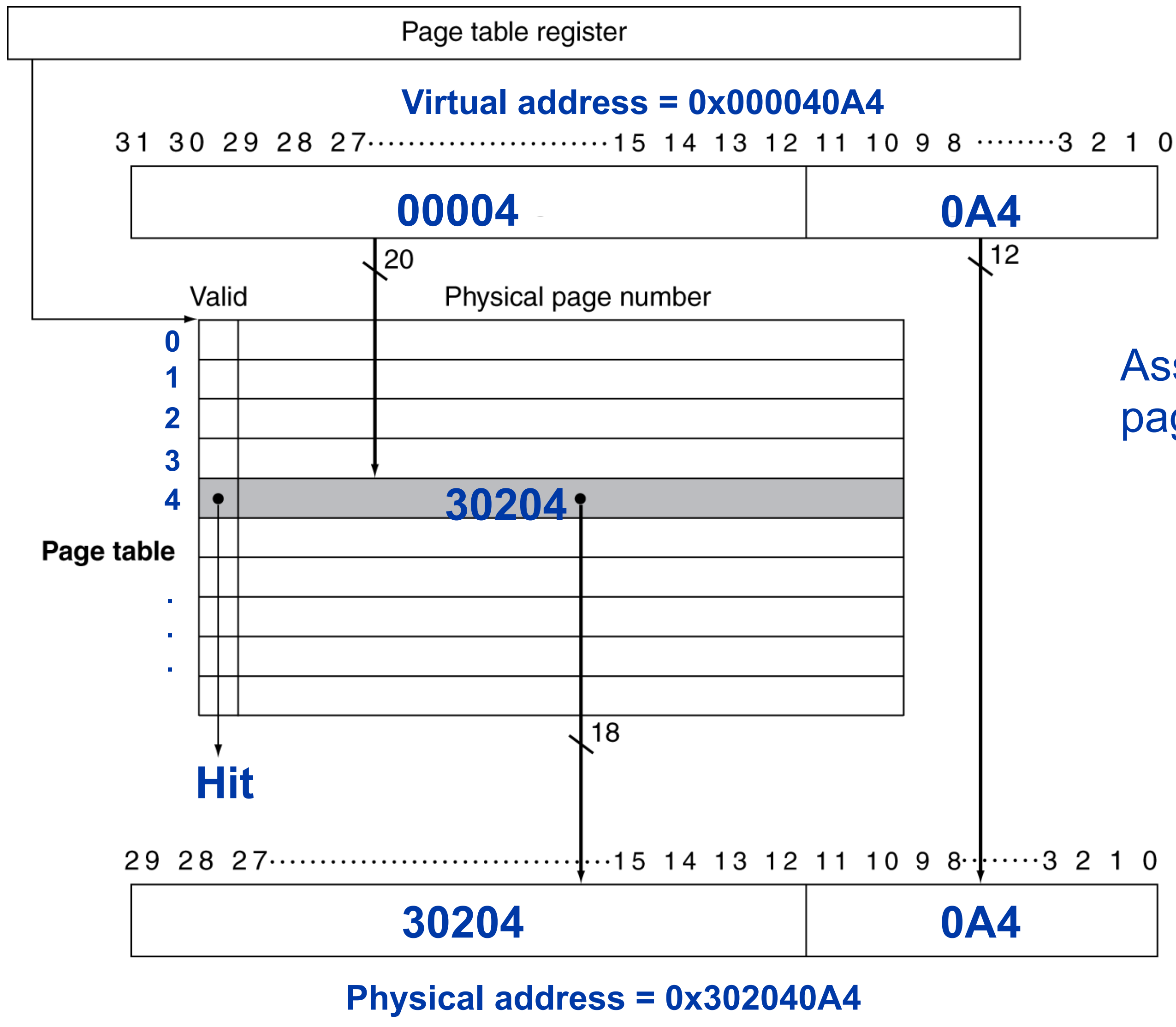
Page not in memory, page fault

Cache omitted here for now

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 = $2^{(32-12)} = 2^{20}$
- Size of page table = number of page table entries x bytes/page table entry
  - Page table size = 2^{20} x 4 = 4 MB

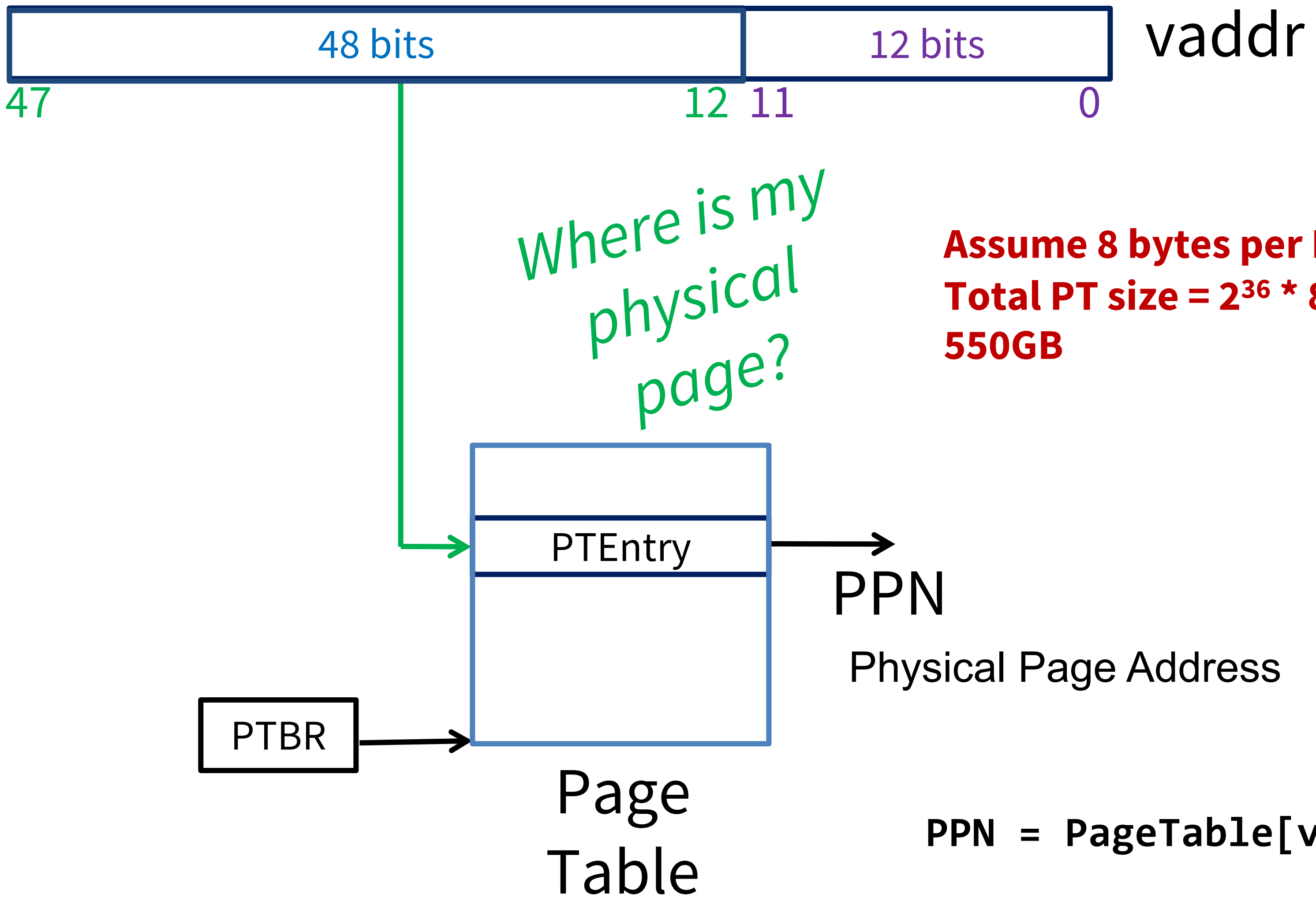
Reducing Page Table Size

- Limit register
  - Restricts number of page table entries
  - Page table expands as needed
- Multiple levels of page table
  - E.g. segment table → page table
  - Total size not smaller, but non-contiguous storage for page table
- Allow page table to go virtual
  - While only having part of the page table in memory
  - The other part in disk

Multi-Level Page Table

- Looking level 0 table, using the highest-order bits of the address -> If the address in this table is valid, the next set of high-order bits is used to index the page table indicated by the segment table entry, and so on.
- Allows the address space to be used in a sparse fashion
- Drawback - Performance: Longer lookups

Single-Level Page Table

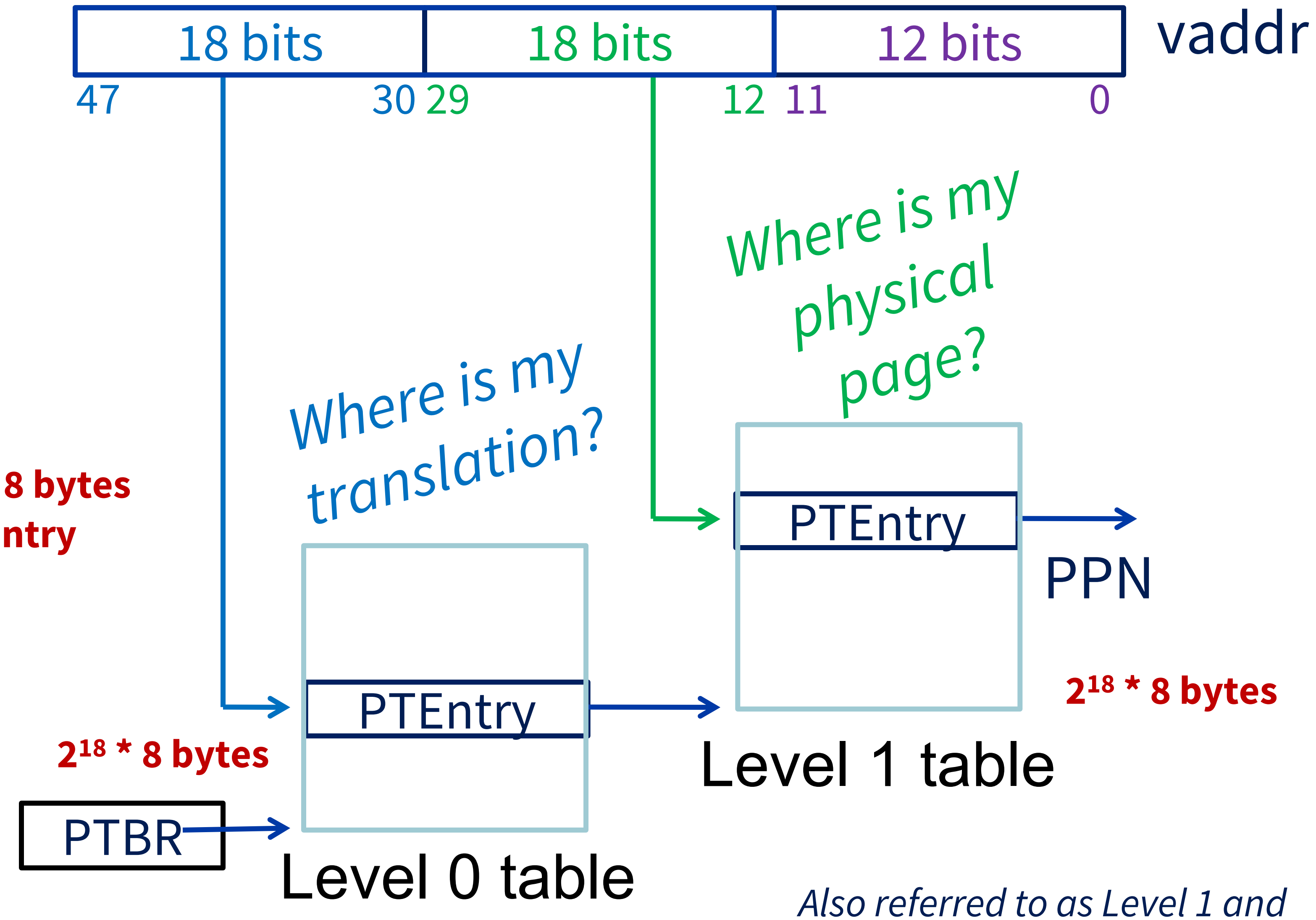


Assume 8 bytes per PT entry
Total PT size = 2^{36} * 8 bytes = 550GB

```
PPN = PageTable[vpn]
```

Image: cs.cornell.edu/courses/cs3410/

Multi-Level Page Table



Assume 8 bytes per PT entry

Also referred to as Level 1 and Level 2 Page Tables

Image: cs.cornell.edu/courses/cs3410/

Page Fault

- Page Fault
  - Requested page in virtual memory is not mapped to a page in main memory
- What should we do on page fault?

Handling Page Fault

- On page fault (page table valid bit is 0)
  - Find the page on disk
  - Fetch and put it in main memory
- Locate a page from disk
  - All virtual pages for a program are in a unique swap space on disk
  - Some page table entries contain disk addresses

Page Fault is Expensive

- On page fault, the page must be fetched from disk and put in main memory
  - Takes millions of clock cycles
  - Handled by OS
- Should try to minimize page fault rate

Reduce Page Fault Rate and Penalty

- Most of the time is for getting the first word in the page – access time – very long
  - Should have **large page size**, so one access fetches more data, also reduces page fault rate
  - Reduce page fault rate by **full associativity**
  - Use **write-back**
  - Handle page fault by software – more sophisticated and less expensive

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

- 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			