Some Real Problem

- What if a program needs more memory than the machine has?
  - even if individual programs fit in memory, how can we run multiple programs?

- How do we protect one program's data from being read or written by another program?
  - multiple programs may want to store something at the same address
  - in particular, consider multiple *copies* of the same program

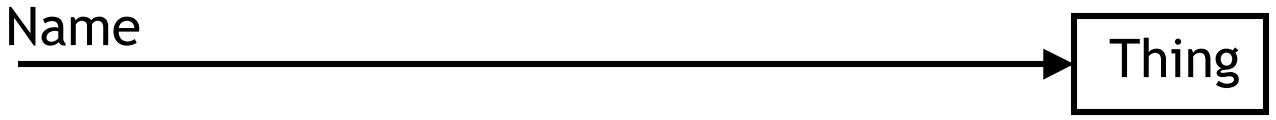
- There are two key ideas used to solve these problems:
  1. Treat the disk as an extended source of memory
     - swap programs between disk and memory as required
  2. Programs use "fake" or "virtual" memory addresses
     - these translate to "real" addresses, but the translation is hidden to the programmer

Indirection

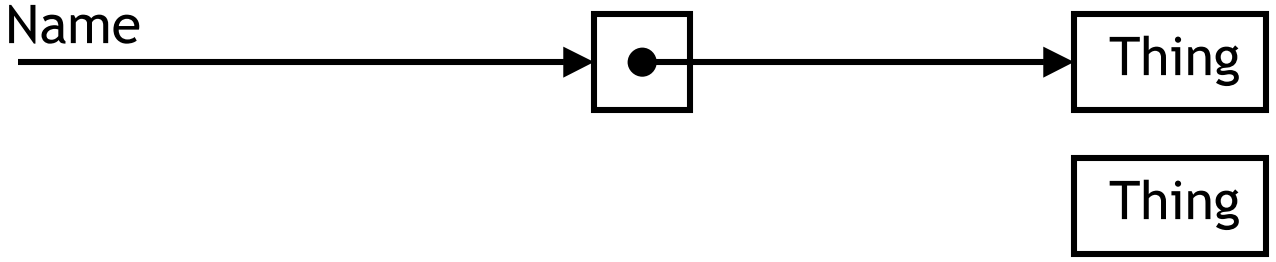
- Many problems can be solved by adding a level of indirection

- Example: “Which bus will take me from here to IMPE?”
- Without indirection: hard to say, because I don’t know which (physical) buses are running right now
- With indirection: “26 Pack”; this (virtual) name identifies one of several (physical) buses

Without Indirection



With Indirection



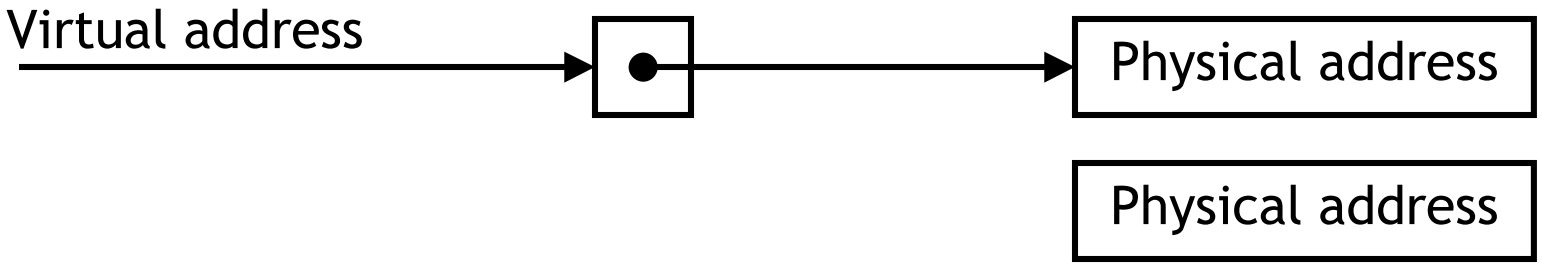
Indirection

- Indirection is the ability to reference something using a name or reference instead the value itself.
  - A mapping between names and things allows changing the thing without notifying holders of the name.

- **More Examples:**

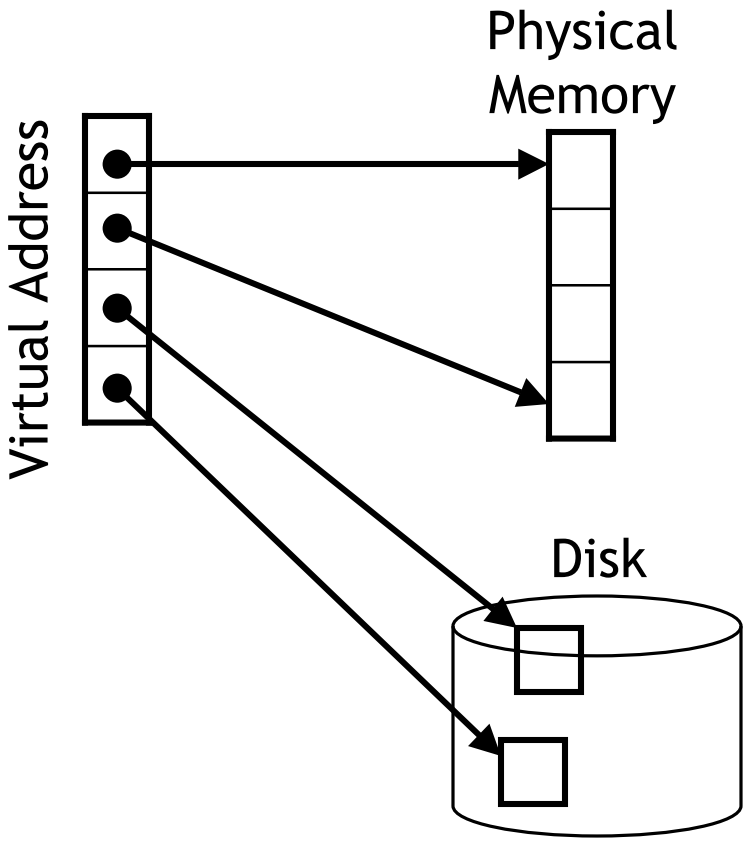
  Pointers, Domain Name Service (DNS) name->IP address, phone system (e.g., cell phone number portability), snail mail (e.g., mail forwarding), 911 (routed to local office), DHCP, color maps, call centers that route calls to available operators, etc.

- In the context of memory:



Virtual Memory

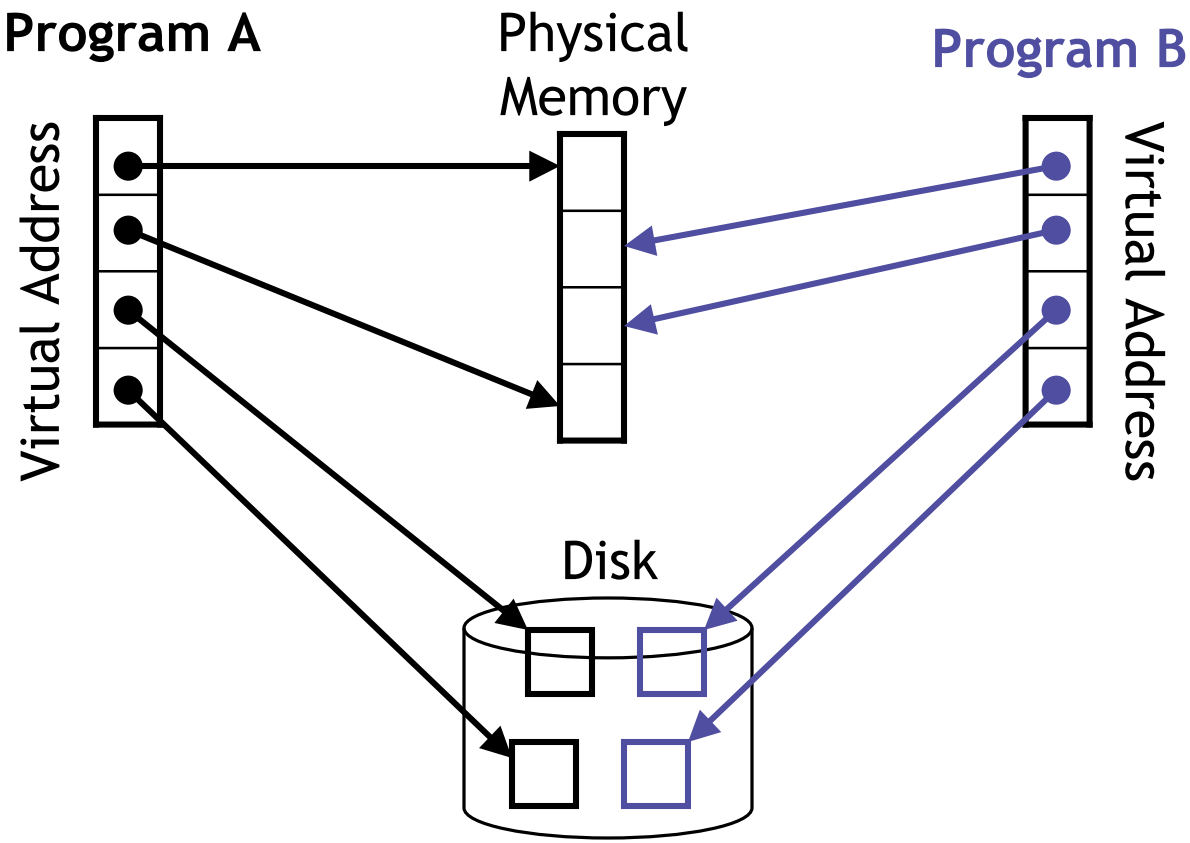
- We translate “virtual addresses” used by the program to “physical addresses” that represent places in the machine’s “physical” memory
  - The word “translate” denotes a level of indirection



A virtual address can be mapped to either physical memory or disk.

Virtual Memory

- Because different processes will have different mappings from virtual to physical addresses, two programs can freely use the same virtual address
- By allocating distinct regions of physical memory to A and B, they are prevented from reading/writing each others data



Caching revisited

- Once the translation infrastructure is in place, the problem boils down to caching.
  - We want the size of disk, but the performance of memory

- The design of virtual memory systems is really motivated by the high cost of accessing disk
  - While memory latency is ~100 times that of cache, disk latency is ~100,000 times that of memory
    - i.e., the miss penalty is HUGE

- Hence, we try to minimize the miss rate:
  - VM "pages" are much larger than cache blocks (why?)
  - A fully associative policy (with approximate LRU) is used

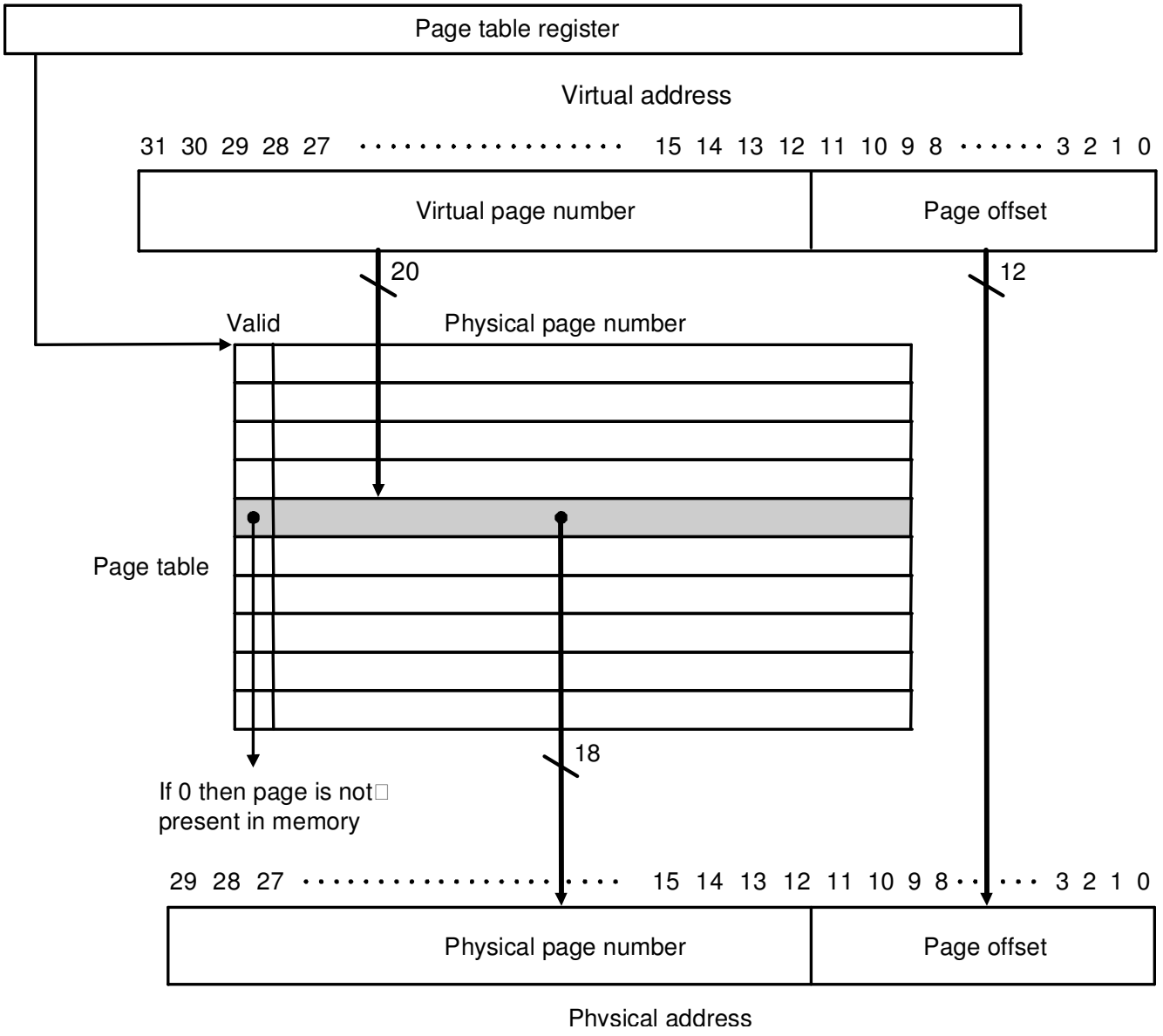
- Should a write-through or write-back policy be used?

Finding the right page

- If it is fully associative, how do we find the right page without scanning all of memory?
  - Use an index (similar to an index in a book)

- The index is called the page table:
  - Each process has a separate page table
    - A "page table register" points to the current process's page table
  - The page table is indexed with the virtual page number (VPN)
    - The VPN is all of the bits that aren't part of the page offset
  - Each entry contains a valid bit, and a physical page number (PPN)
    - The PPN is concatenated with the page offset to get the physical address
  - No tag is needed because the index is the full VPN

Page Table picture



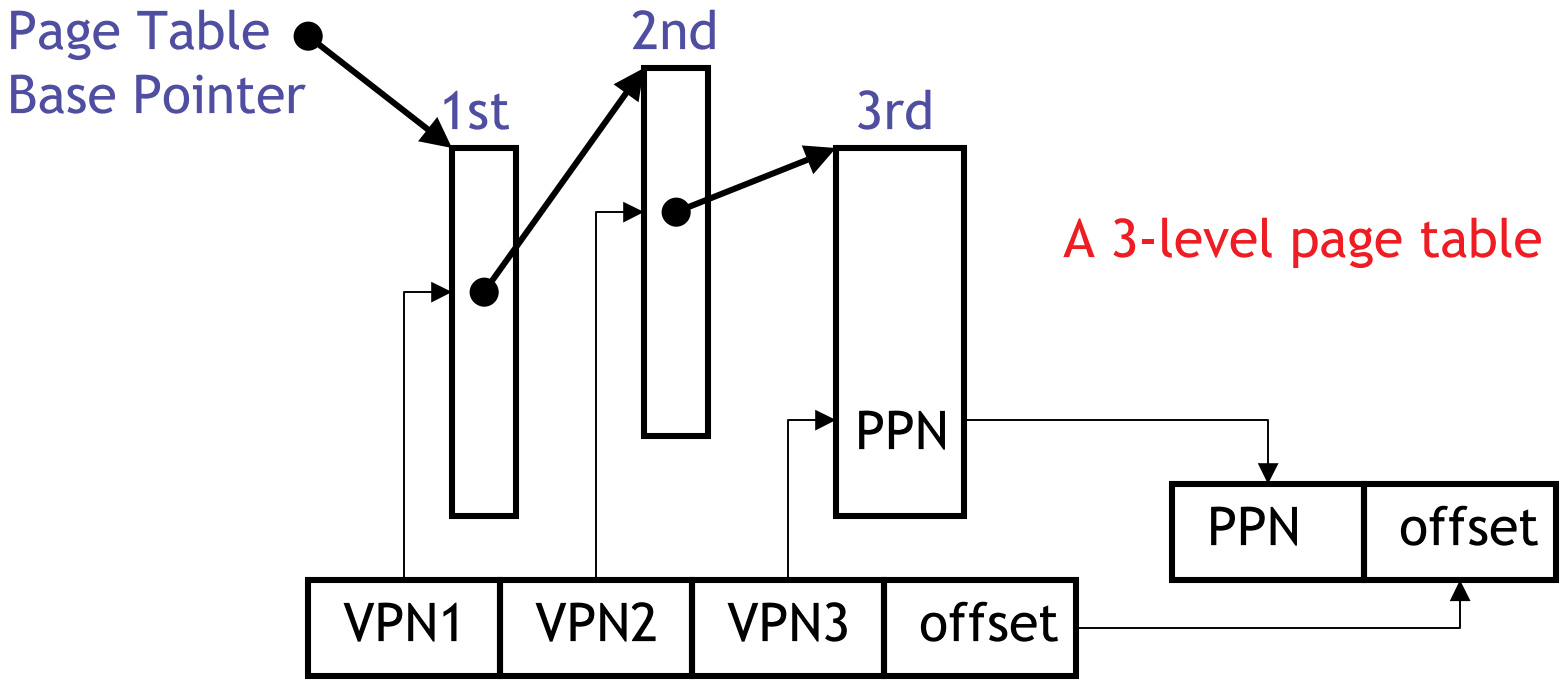
How big is the page table?

- From the previous slide:
  - Virtual page number is 20 bits.
  - Physical page number is 18 bits + valid bit → round up to 32 bits.

- How about for a 64b architecture?

Dealing with large page tables

- Multi-level page tables



- Since most processes don't use the whole address space, you don't allocate the tables that aren't needed
  - Also, the 2nd and 3rd level page tables can be "paged" to disk

Waitaminute!

- We've just replaced every memory access MEM[addr] with:

  MEM[MEM[MEM[MEM[PTBR + VPN1<<2] + VPN2<<2] + VPN3<<2] + offset]

  — i.e. 4 memory accesses

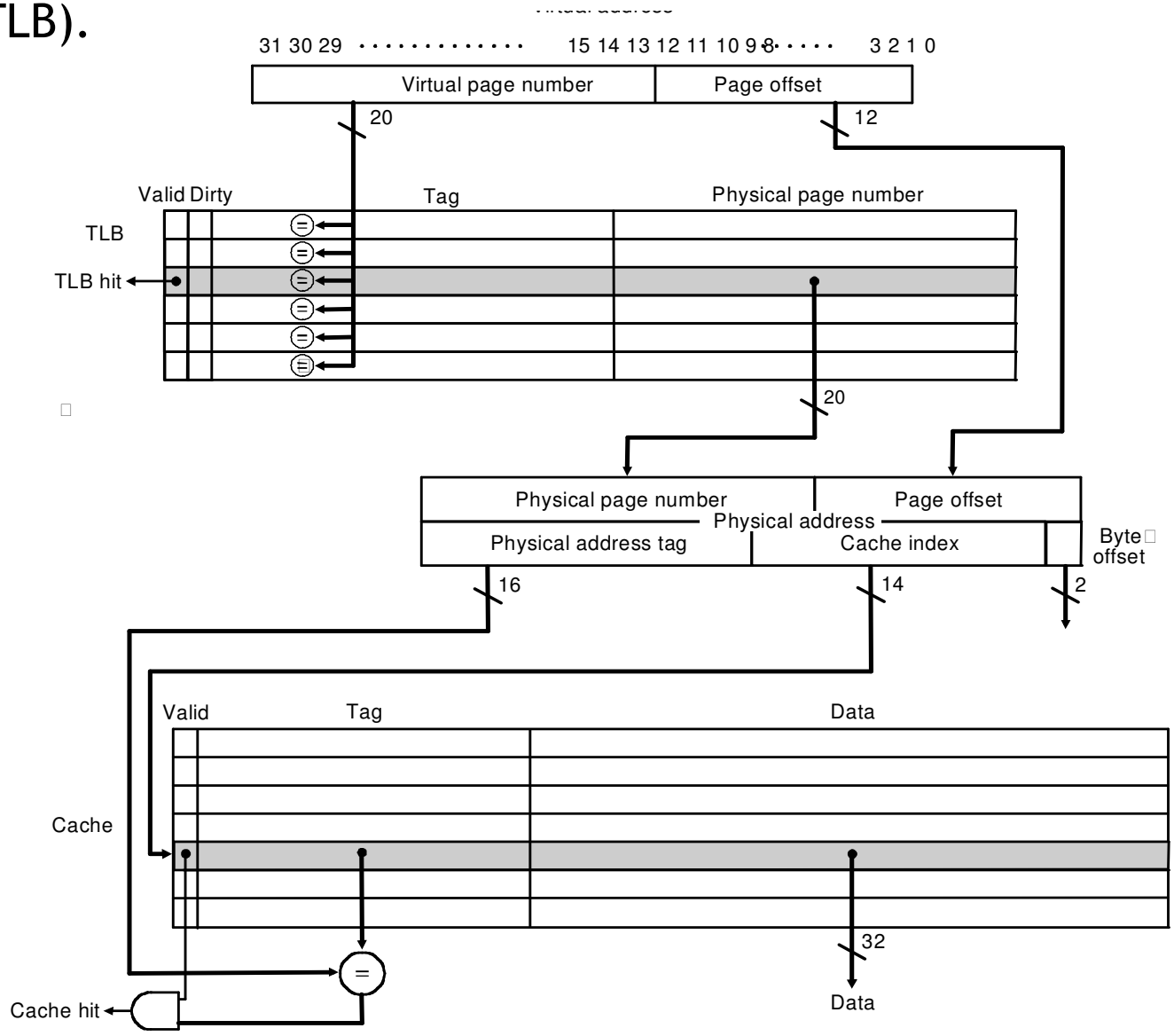
- And we haven't talked about the bad case yet (i.e. page faults)…

- We have too many levels of indirection!

- How do we deal with too many levels of indirection?

Caching Translations

- Virtual to Physical translations are cached in a Translation Lookaside Buffer (TLB).



What about a TLB miss?

- If we miss in the TLB, we need to “walk the page table”
  - In MIPS, an exception is raised and software fills the TLB
  - In x86, a “hardware page table walker” fills the TLB

- What if the page is not in memory?
  - This situation is called a page fault
  - The operating system will have to request the page from disk
  - It will need to select a page to replace
    - The O/S tries to approximate LRU (see CS423)
  - The replaced page will need to be written back if dirty

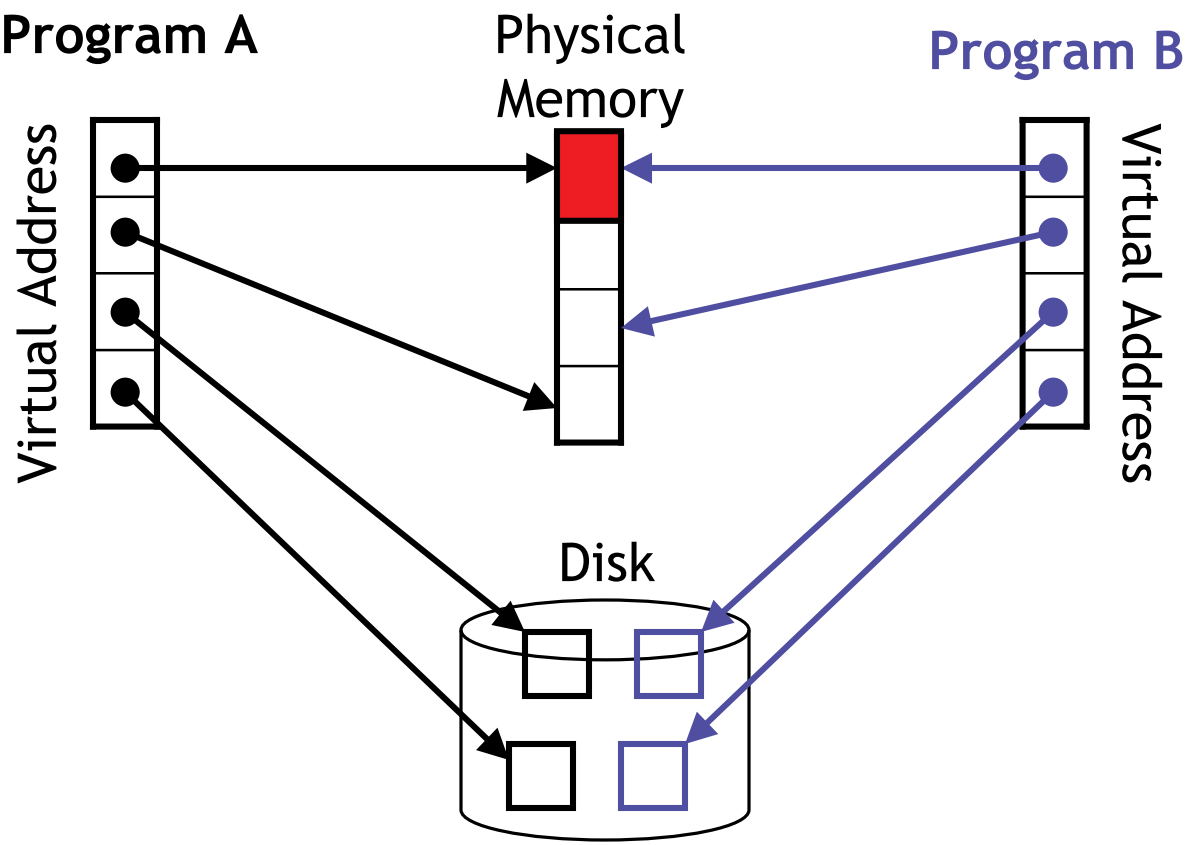
Memory Protection

- In order to prevent one process from reading/writing another process's memory, we must ensure that a process cannot change its virtual-to-physical translations

- Typically, this is done by:
  - Having two processor modes: user & kernel
    - Only the O/S runs in kernel mode
  - Only allowing kernel mode to write to the virtual memory state:
    - The page table
    - The page table base pointer
    - The TLB

Sharing Memory

- Paged virtual memory enables sharing at the granularity of a page, by allowing two page tables to point to the same physical addresses
- For example, if you run two copies of a program, the O/S will share the code pages between the programs



Summary

- Virtual memory is great:
  - It means that we don't have to manage our own memory
  - It allows different programs to use the same memory
  - It provides protect between different processes
  - It allows controlled sharing between processes (albeit somewhat inflexibly)
- The key technique is **indirection**:
  - Yet another classic CS trick you've seen in this class
  - Many problems can be solved with indirection
- Caching made a few cameo appearances, too:
  - Virtual memory enables using physical memory as a cache for disk
  - We used caching (in the form of the Translation Lookaside Buffer) to make Virtual Memory's indirection fast