A Real Problem

- What if you wanted to run a program that needs more memory than you have?

virtual memory

OFFICE HOURS AFTER CLASS

NO HANDOUT

Virtual Memory (and Indirection)

- Finally, we get to Virtual Memory!
  - We'll talk about the motivations for virtual memory
  - We'll talk about how it is implemented
  - Lastly, we'll talk about how to make virtual memory fast: Translation Lookaside Buffers (TLBs).
- Starting Friday, we'll turn our attention to peripheral devices and I/O.

A Real Problem

- What if you wanted to run a program that needs more memory than you have?
  - You could store the whole program on disk, and use memory as a cache for the data on disk. This is one feature of virtual memory.
  - Before virtual memory, programmers had to manually manage loading "overlays" (chunks of instructions & data) off disk before they were used. This is an incredibly tedious, not to mention error-prone, process.

More Real Problems

- Running multiple programs at the same time brings up more problems.

1. Even if each program fits in memory, running 10 programs might not.

2. Multiple programs may want to store something at the same address.

3. How do we protect one program's data from being read or written by another program?

More Real Problems

- Running multiple programs at the same time brings up more problems.

1. Even if each program fits in memory, running 10 programs might not.
   - This is really the same problem as on the previous slide.

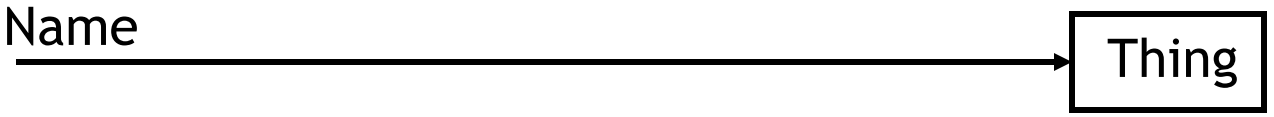
2. Multiple programs may want to store something at the same address.
   - I.e., what if both Program A and B want to use address 0x10000000 as the base of their stack?
   - It is impractical (if not impossible) to compile every pair of programs that could get executed together to use distinct sets of addresses.

3. How do we protect one program's data from being read or written by another program?

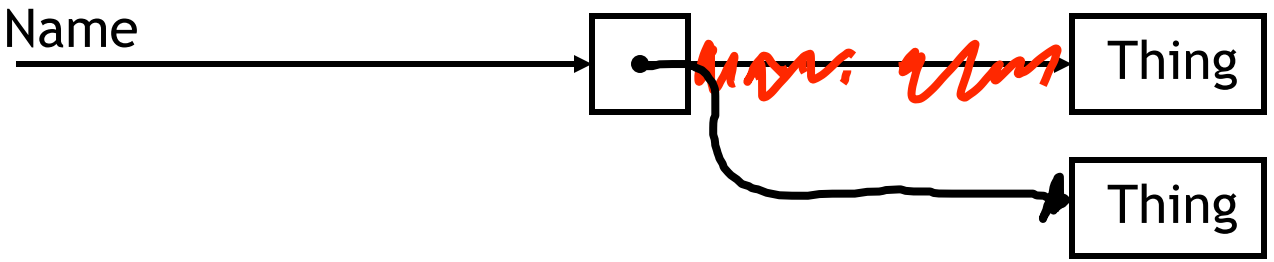
Indirection

- "Any problem in CS can be solved by adding a level of indirection"

- Without Indirection



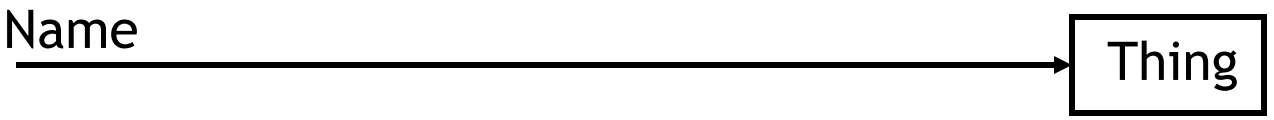
- With Indirection



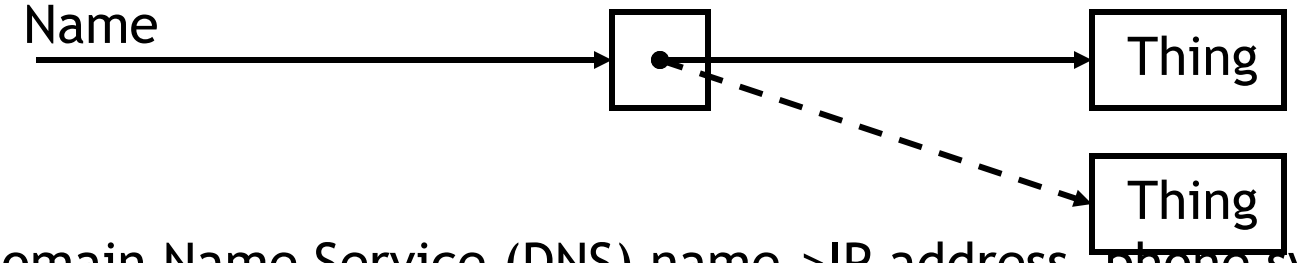
Indirection

- **Indirection:** Indirection is the ability to reference something using a name, reference, or container instead the value itself. A flexible mapping between a name and a thing allows changing the thing without notifying holders of the name.

- Without Indirection



- With Indirection

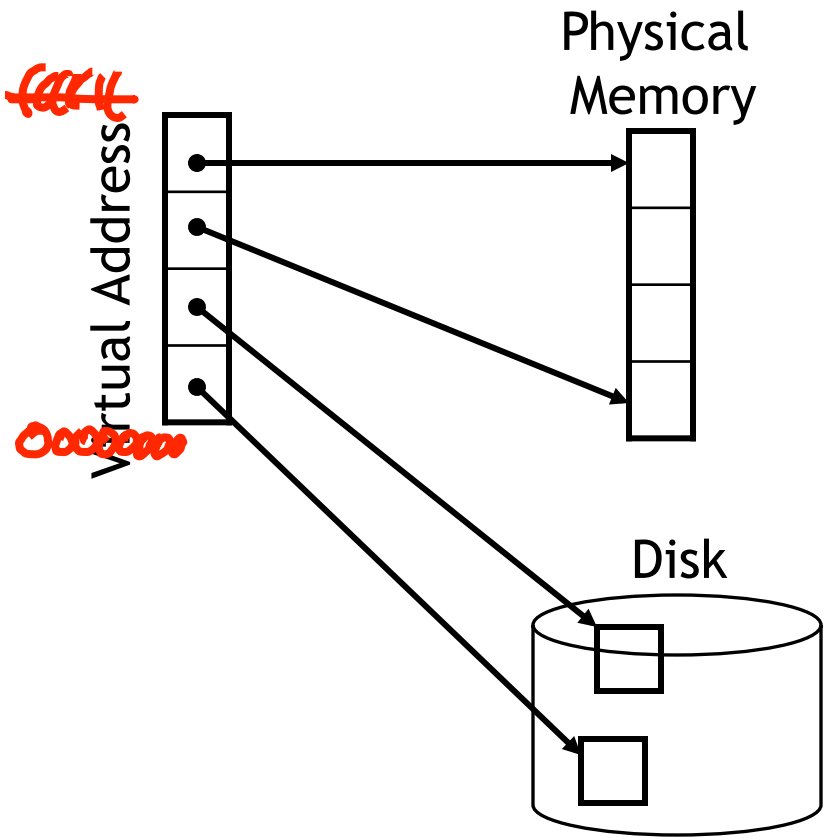


- **Examples:**

  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.

Virtual Memory

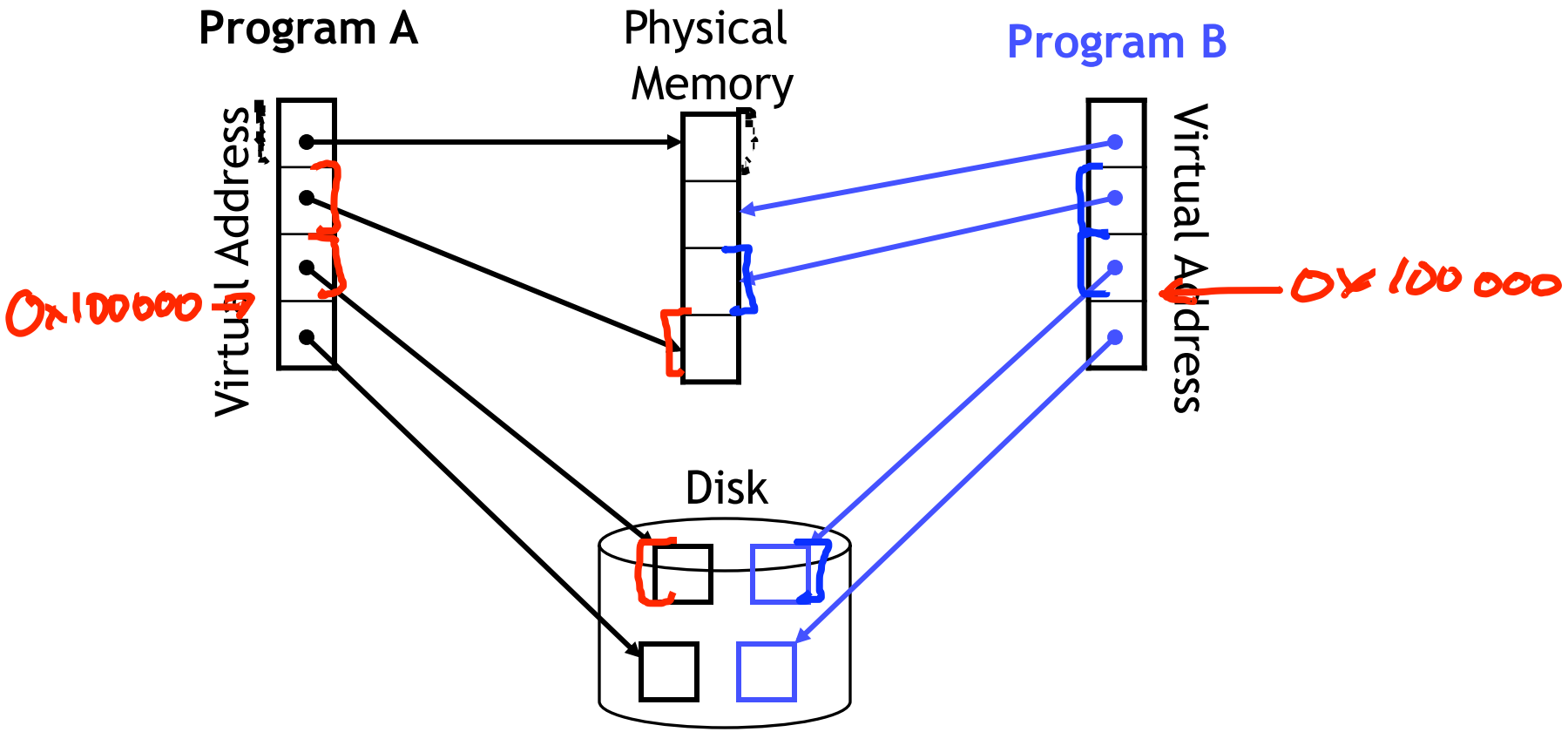
- We translate "virtual addresses" used by the program to "physical addresses" that represent places in the machine's "physical" memory.
  - The word "translate" denotes a level of indirection



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 a real whopper.

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

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

≥4KB
1) disk access cost
2) spatial locality

Finding the right page

- If it is fully associative, how do we find the right page without scanning all of memory?

4KB 4GB ⇒ 1M pages

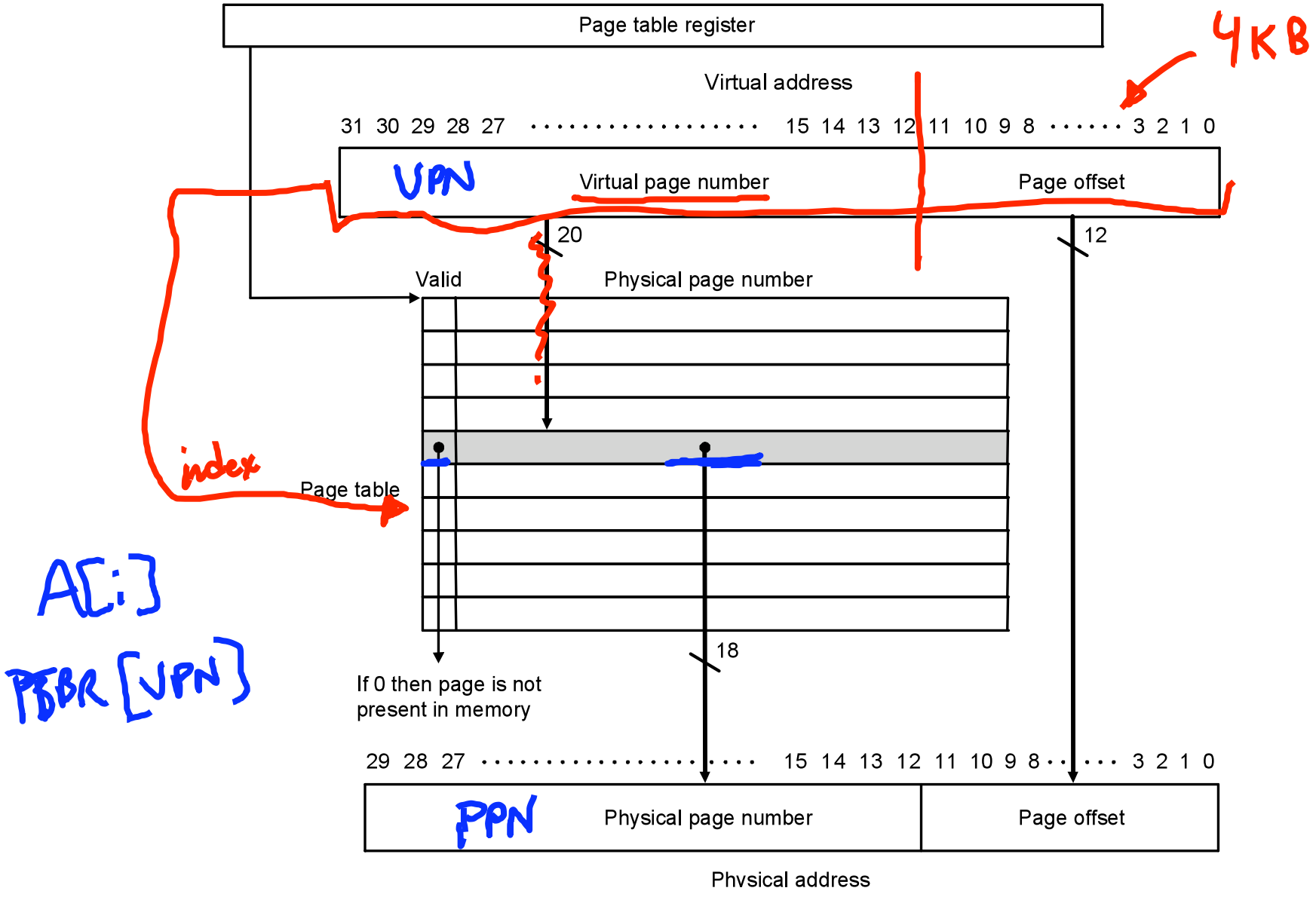
index

Finding the right page

- If it is fully associative, how do we find the right page without scanning all of memory?
  - Use an index, just like you would for a book.

- Our index happens to be 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.

32b

2^{20} pages = ~1M pages

4B per page → 4MB

- How about for a 64b architecture?

42b virtual address

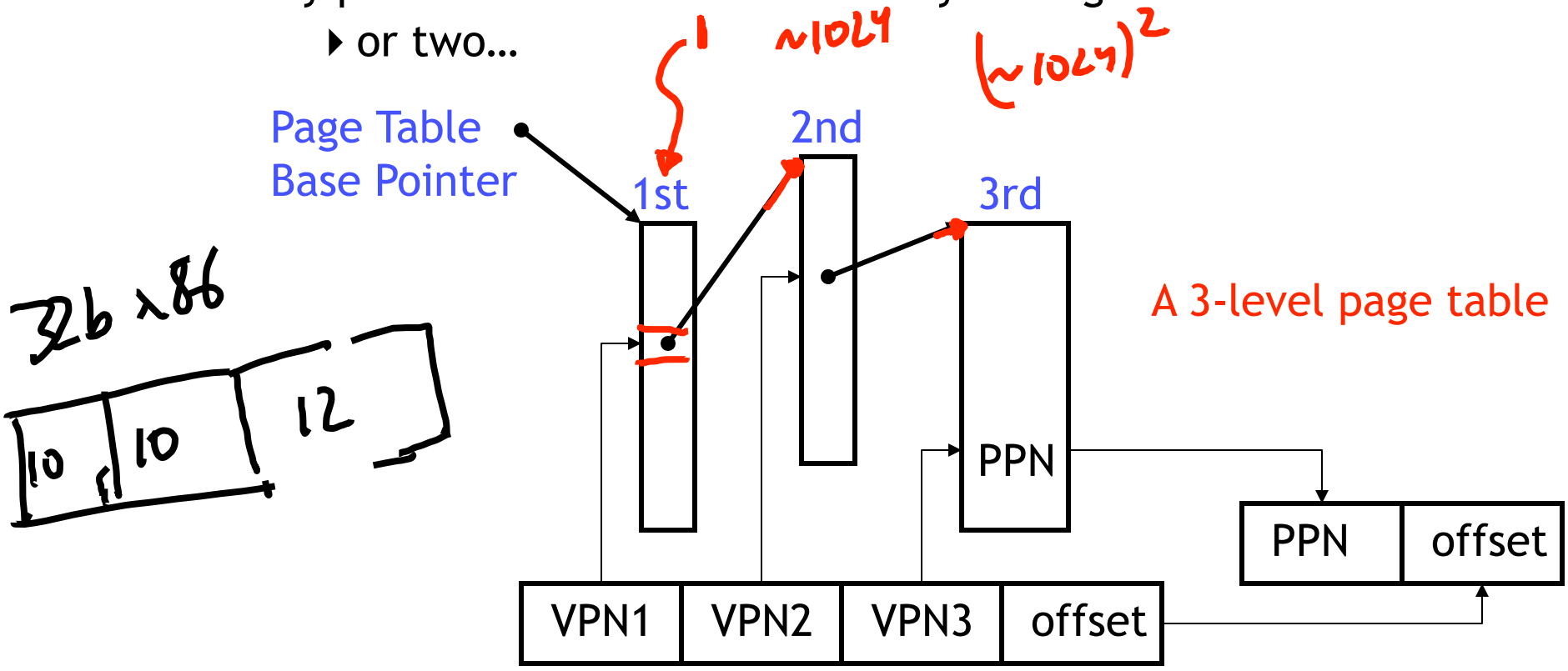
42-12 = 30 → 2^{30} pages

assume 4B entries

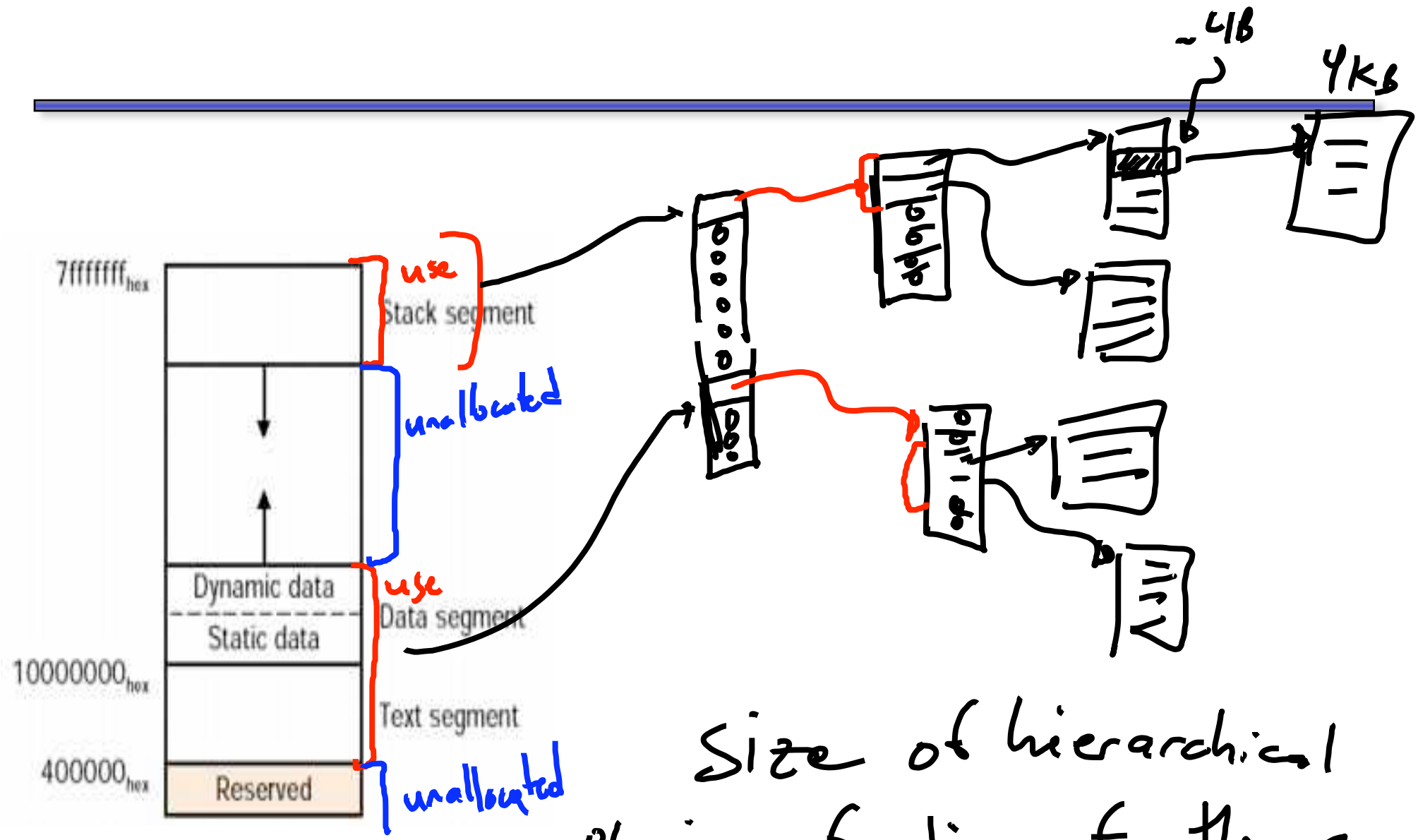
4GB

Dealing with large page tables

- Multi-level page tables
  - "Any problem in CS can be solved by adding a level of indirection"
    - or two…



- 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.



Size of hierarchical

4B / 4KB = ~1/1000 = .1% is a function of the amount of virtual memory used.

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)…

"Any problem in CS can be solved by adding a level of indirection"

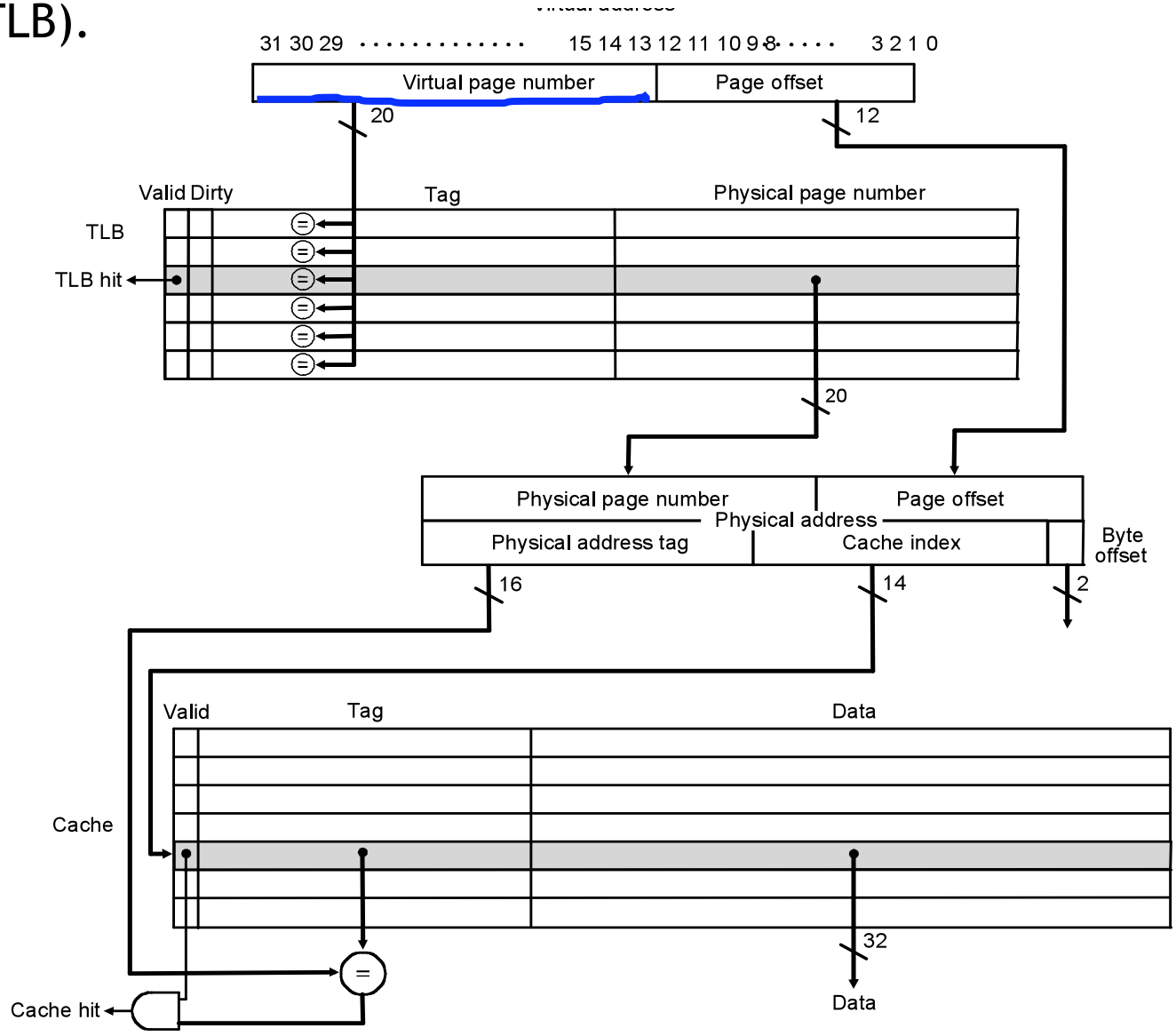
— except too many levels of indirection…

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

Caching!

Caching Translations

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



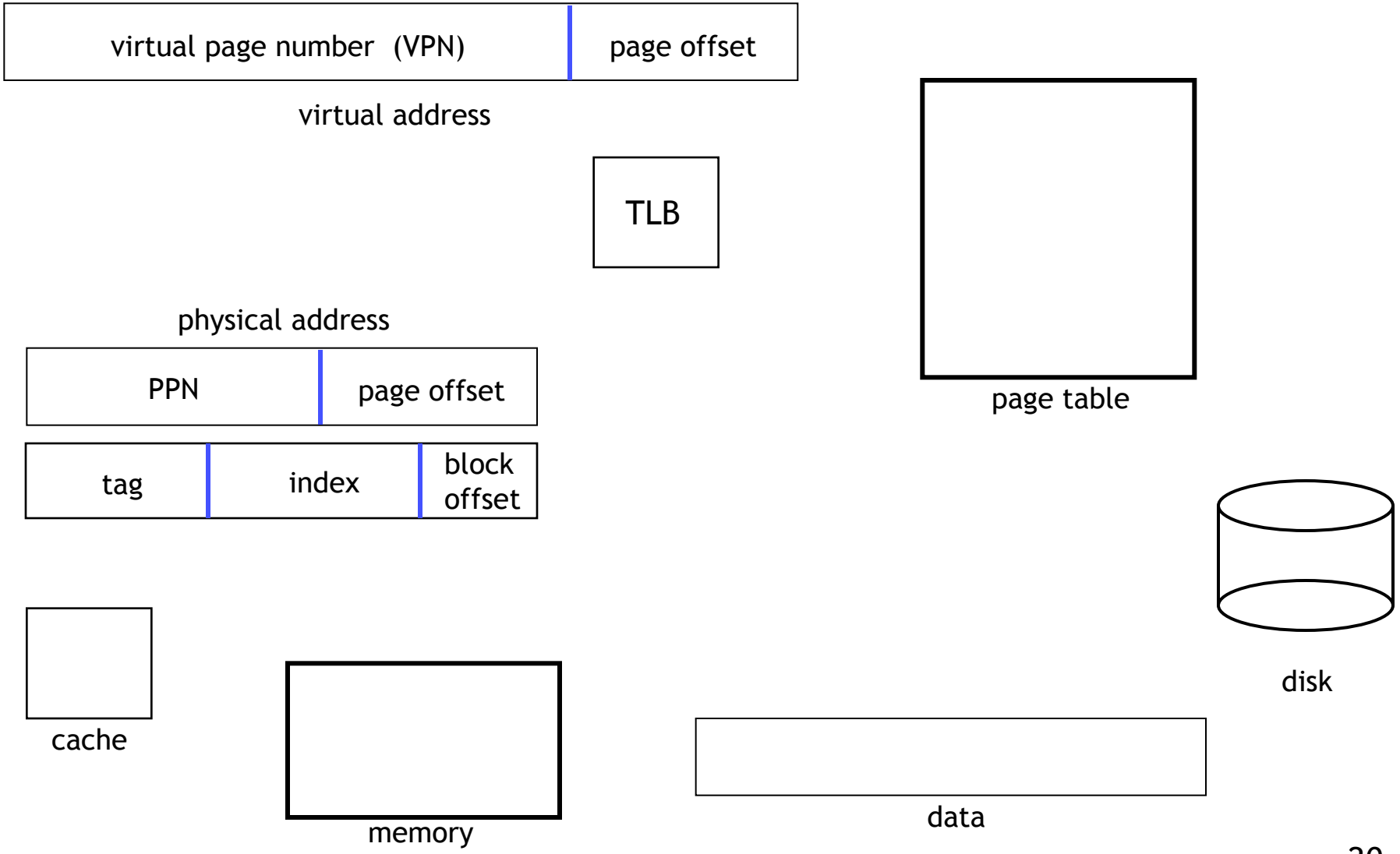
What about a TLB miss?

- If we miss in the TLB, we need to "walk the page table"
  - In MIPS, an exception is raised and software fills the TLB
    - MIPS has "TLB_write" instructions
  - 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 CS241/CS423)
  - The replaced page will need to be written back if dirty.

Putting it all together

- Add arrows to indicate what happens on a lw



Virtual Memory & Prefetching

- Don’t want to cause page faults by prefetching

- Prefetches typically dropped when they miss in the TLB
  - Don’t want to disrupt program’s execution for a prefetch.
  - May cause a hardware TLB fill on x86 platforms.

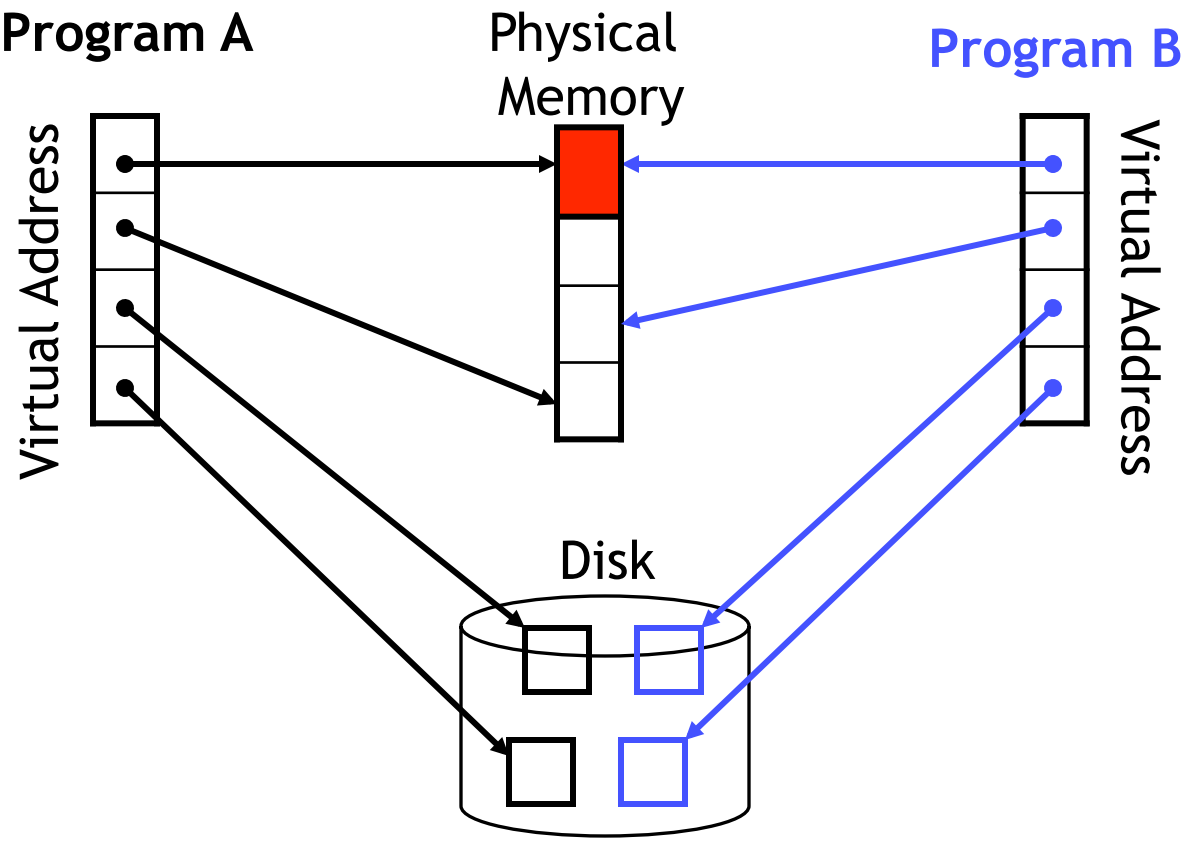
- HW prefetchers don’t cross page boundaries.
  - They use physical addresses
    - Don’t use the TLB.
  - After page boundary don’t know where next page lies
  - Sequential stream will have a few misses @ beginning of each page

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, *e.g.*,
    - 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 pure manna from heaven:
  - It means that we don't have to manage our own memory.
  - It allows different programs to use the same (virtual) addresses.
  - 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.