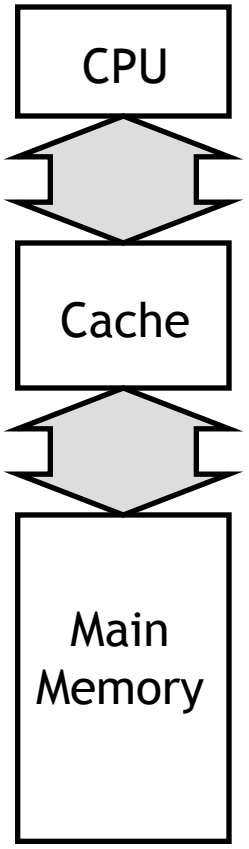
Lecture 18

- Midterm discussion
- Reducing miss penalty

Basic main memory design

- There are some ways the main memory can be organized to reduce miss penalties and help with caching.
- For some concrete examples, let's assume the following three steps are taken when a cache needs to load data from the main memory.
  1. It takes 1 cycle to send an address to the RAM.
  2. There is a 15-cycle latency for each RAM access.
  3. It takes 1 cycle to return data from the RAM.
- In the setup shown here, the buses from the CPU to the cache and from the cache to RAM are all one word wide.
- If the cache has one-word blocks, then filling a block from RAM (*i.e.*, the miss penalty) would take 17 cycles.



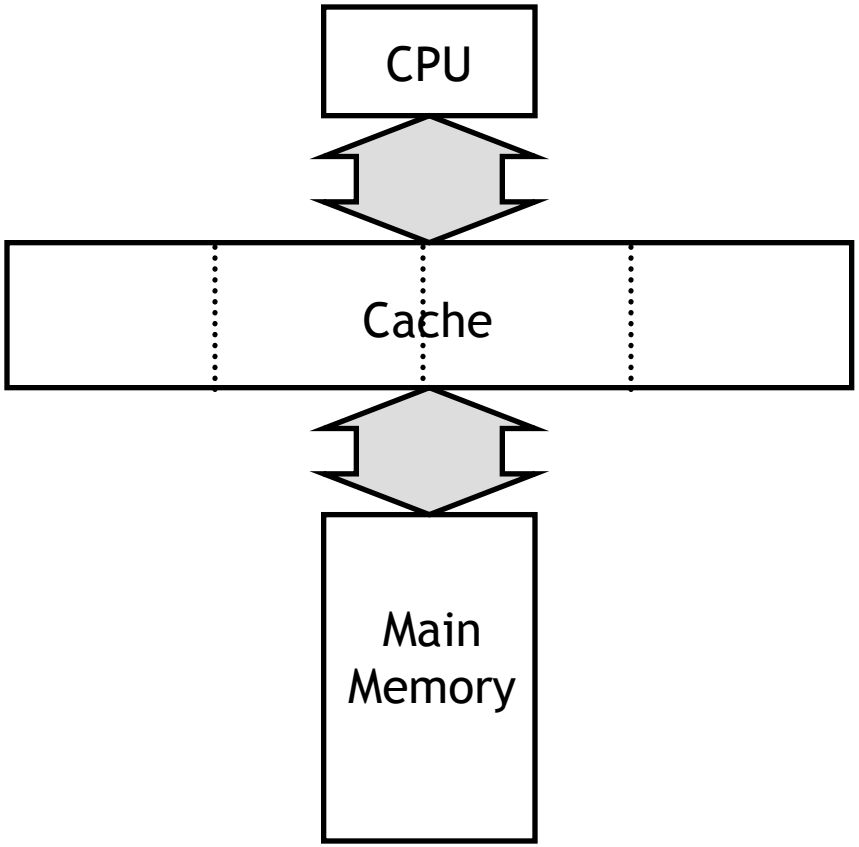
1 + 15 + 1 = 17 clock cycles

- The cache controller has to send the desired address to the RAM, wait and receive the data.

Miss penalties for larger cache blocks

- If the cache has four-word blocks, then loading a single block would need four individual main memory accesses, and a miss penalty of 68 cycles!

4 x (1 + 15 + 1) = 68 clock cycles

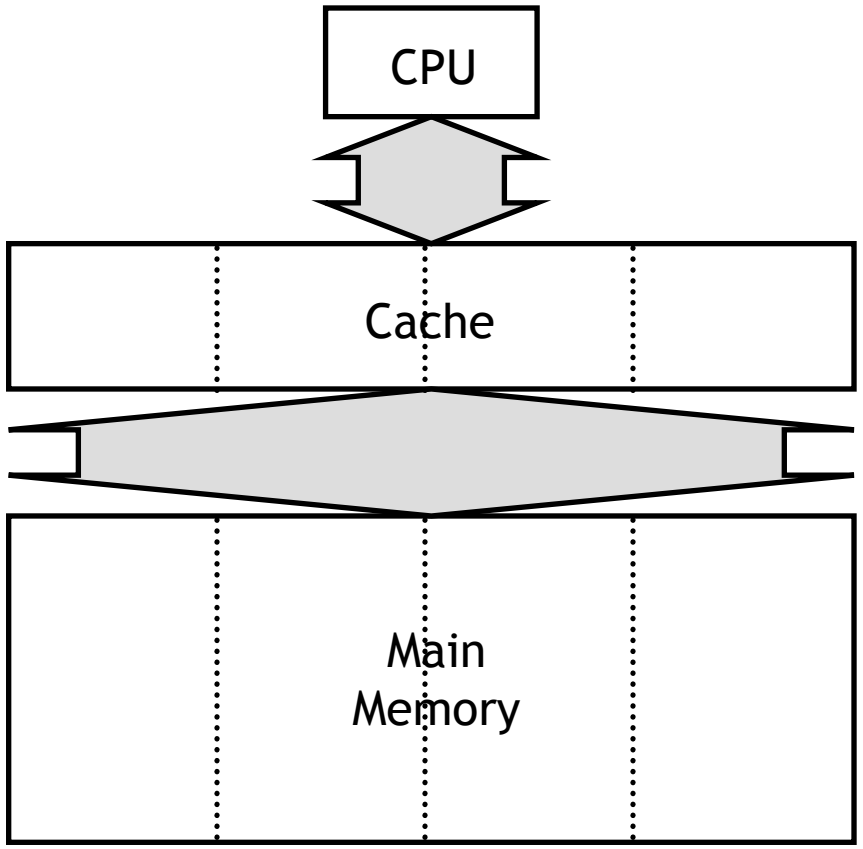


A wider memory

- A simple way to decrease the miss penalty is to widen the memory and its interface to the cache, so we can read multiple words from RAM in one shot.
- If we could read four words from the memory at once, a four-word cache load would need just 17 cycles.

  1 + 15 + 1 = 17 cycles

- The disadvantage is the cost of the wider buses—each additional bit of memory width requires another connection to the cache.

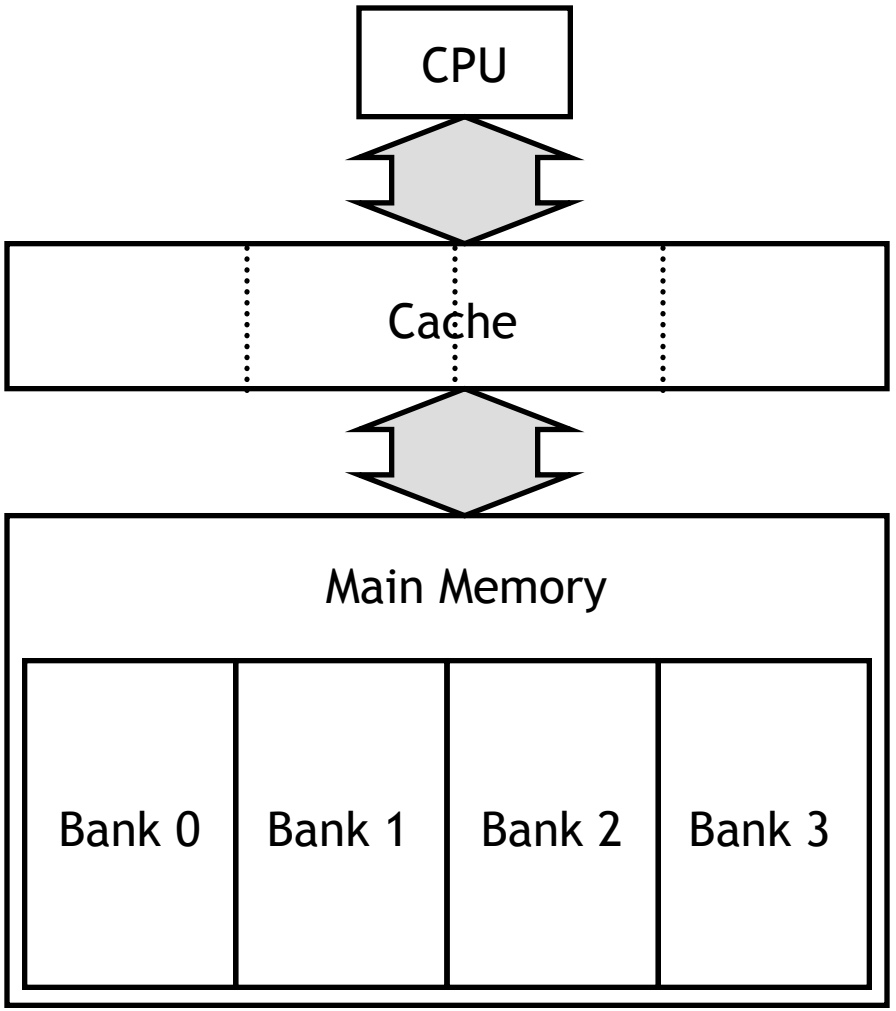


An interleaved memory

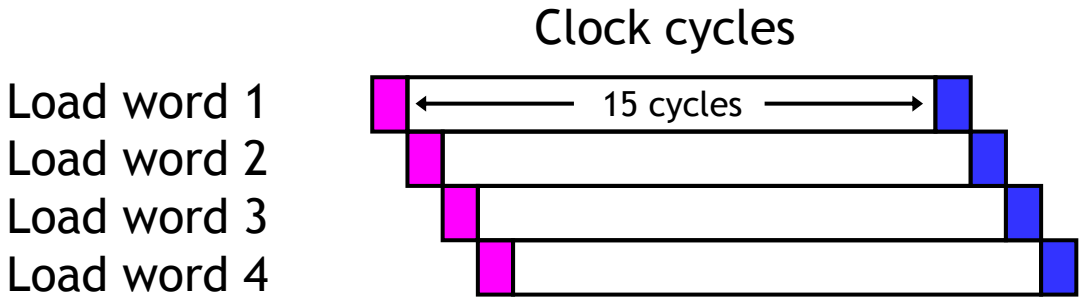
- Another approach is to interleave the memory, or split it into "banks" that can be accessed individually.
- The main benefit is overlapping the latencies of accessing each word.
- For example, if our main memory has four banks, each one byte wide, then we could load four bytes into a cache block in just 20 cycles.

1 + 15 + (4 x 1) = 20 cycles

- Our buses are still one byte wide here, so four cycles are needed to transfer data to the caches.
- This is cheaper than implementing a four-byte bus, but not too much slower.



Interleaved memory accesses



- Here is a diagram to show how the memory accesses can be interleaved.
  - The magenta cycles represent sending an address to a memory bank.
  - Each memory bank has a 15-cycle latency, and it takes another cycle (shown in blue) to return data from the memory.
- This is the same basic idea as pipelining!
  - As soon as we request data from one memory bank, we can go ahead and request data from another bank as well.
  - Each individual load takes 17 clock cycles, but four overlapped loads require just 20 cycles.

Which is better?

- Increasing block size can improve hit rate (due to spatial locality), but transfer time increases. Which cache configuration would be better?

	Cache #1	Cache #2
Block size	32-bytes	64-bytes
Miss rate	5%	4%

- Assume both caches have single cycle hit times. Memory accesses take 15 cycles, and the memory bus is 8-bytes wide:
  - i.e., an 16-byte memory access takes 18 cycles:

    1 (send address) + 15 (memory access) + 2 (two 8-byte transfers)

recall: AMAT = Hit time + (Miss rate x Miss penalty)

Which is better?

- Increasing block size can improve hit rate (due to spatial locality), but transfer time increases. Which cache configuration would be better?

	Cache #1	Cache #2
Block size	32-bytes	64-bytes
Miss rate	5%	4%

- Assume both caches have single cycle hit times. Memory accesses take 15 cycles, and the memory bus is 8-bytes wide:
  - i.e., an 16-byte memory access takes 18 cycles:

    1 (send address) + 15 (memory access) + 2 (two 8-byte transfers)

Cache #1:

Miss Penalty = 1 + 15 + 32B/8B = 20 cycles

AMAT = 1 + (.05 * 20) = 2

Cache #2:

Miss Penalty = 1 + 15 + 64B/8B = 24 cycles

AMAT = 1 + (.04 * 24) = ~1.96

recall: AMAT = Hit time + (Miss rate x Miss penalty)

Summary

- Writing to a cache poses a couple of interesting issues.
  - Write-through and write-back policies keep the cache consistent with main memory in different ways for write hits.
  - Write-around and allocate-on-write are two strategies to handle write misses, differing in whether updated data is loaded into the cache.
- Memory system performance depends upon the cache hit time, miss rate and miss penalty, as well as the actual program being executed.
  - We can use these numbers to find the average memory access time.
  - We can also revise our CPU time formula to include stall cycles.

AMAT = Hit time + (Miss rate x Miss penalty)

Memory stall cycles = Memory accesses x miss rate x miss penalty

CPU time = (CPU execution cycles + Memory stall cycles) x Cycle time

- The organization of a memory system affects its performance.
  - The cache size, block size, and associativity affect the miss rate.
  - We can organize the main memory to help reduce miss penalties. For example, interleaved memory supports pipelined data accesses.

A Real Problem

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

Virtual Memory (and Indirection)

- 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).

More Real Problems

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

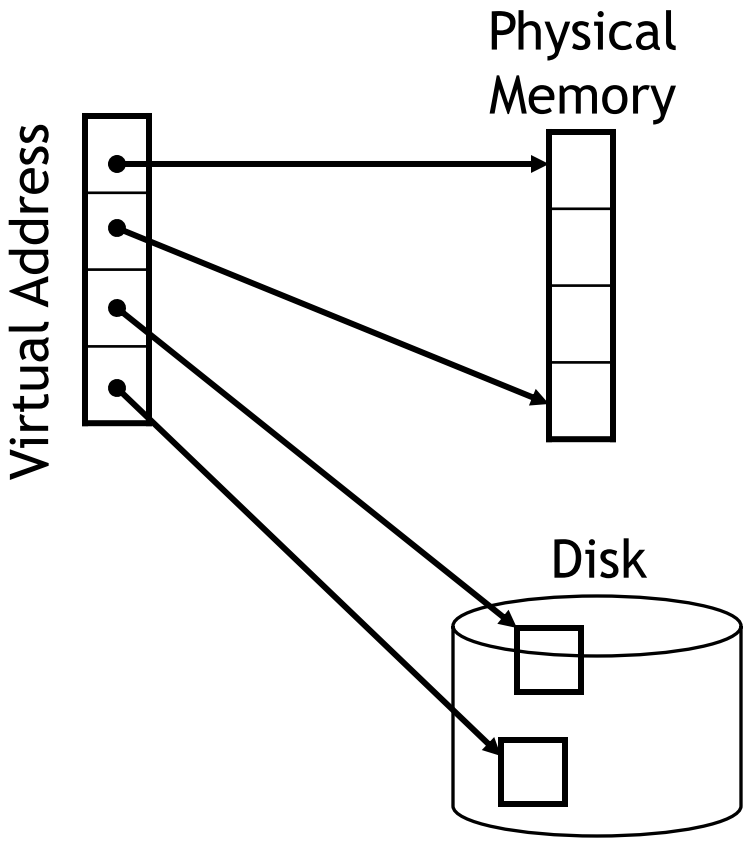
1. Even if each program fits in memory, running 10 programs might not.

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?

Virtual Memory

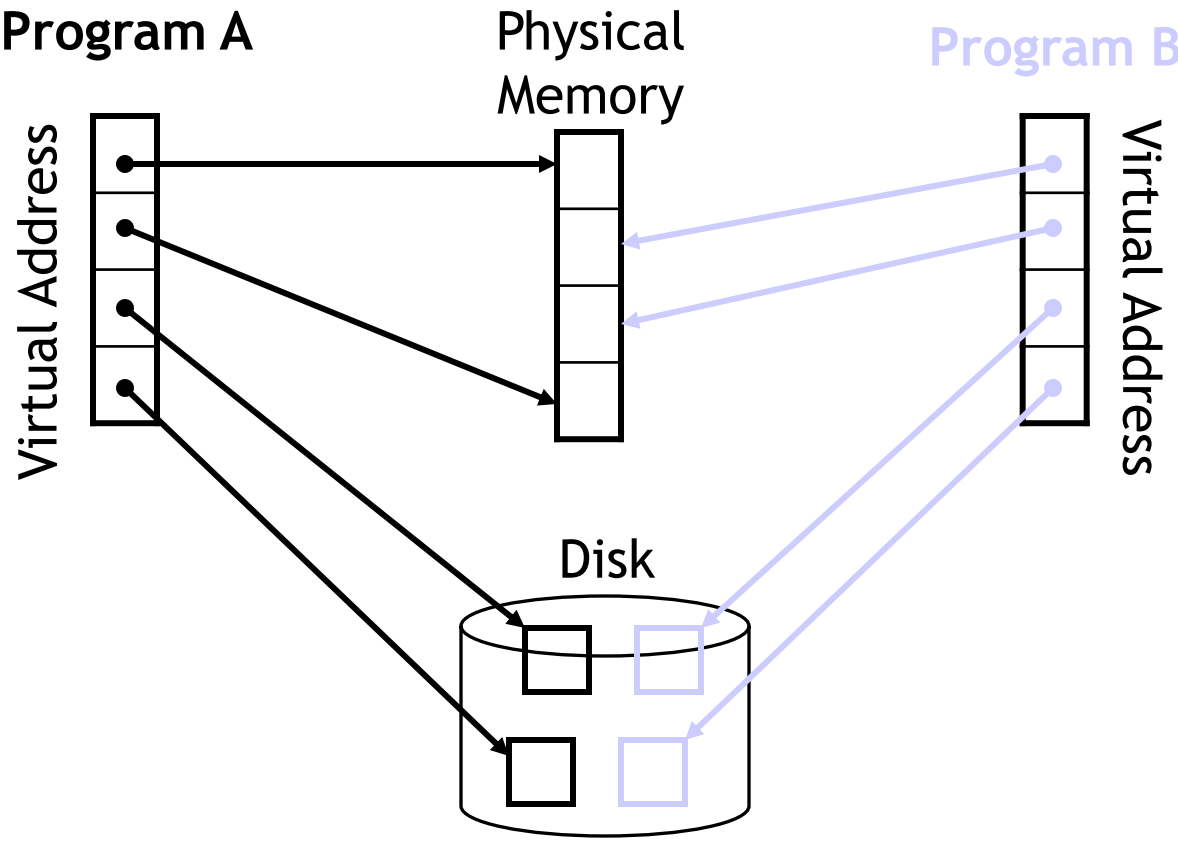
- We translate “virtual addresses” used by the program to “physical addresses” that represent places in the machine’s “physical” memory.
  - The word “translate” denotes a level of indirection



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.

- 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?

Finding the right page

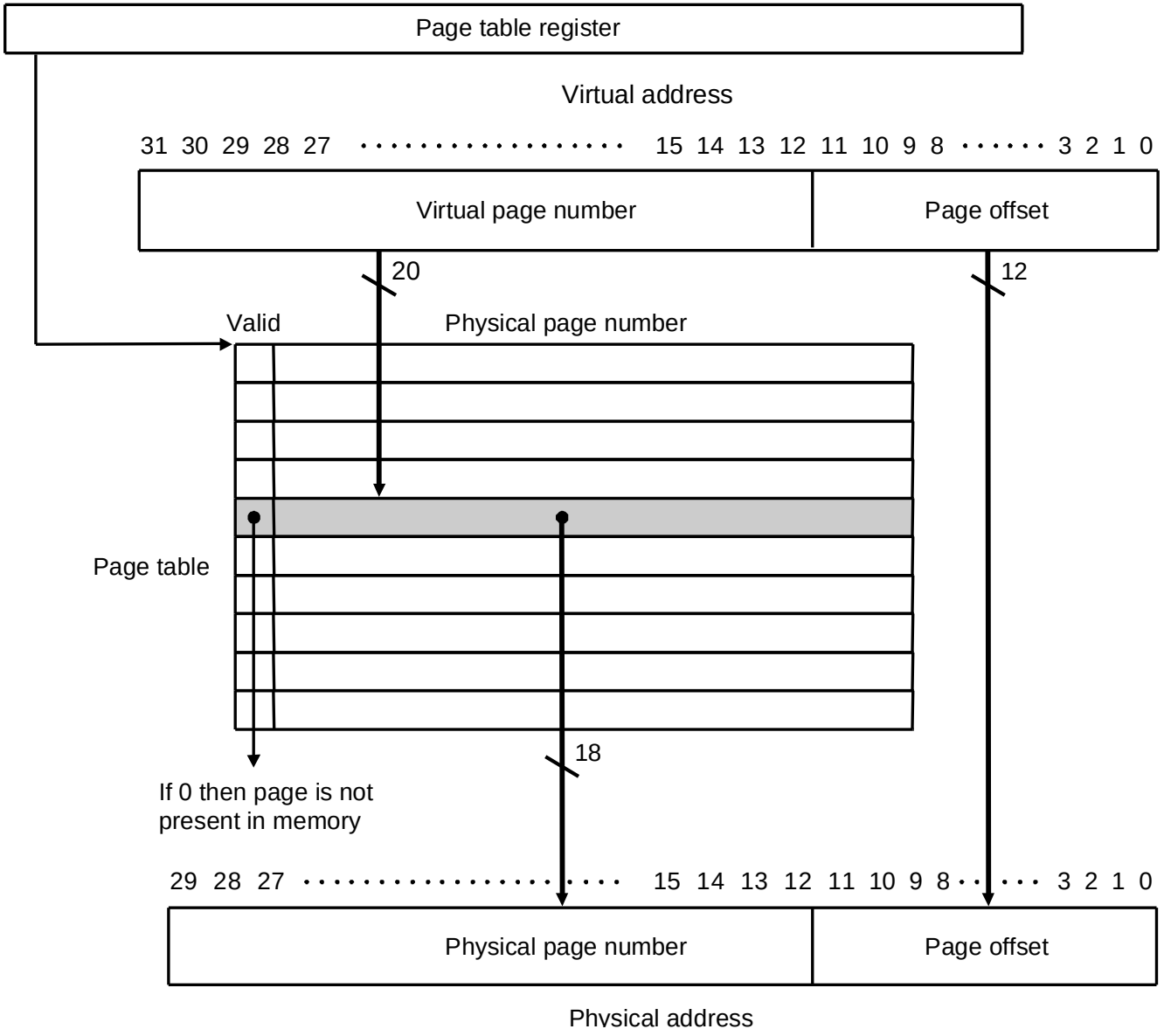
- If it is fully associative, how to we find the right page without scanning all of memory?

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



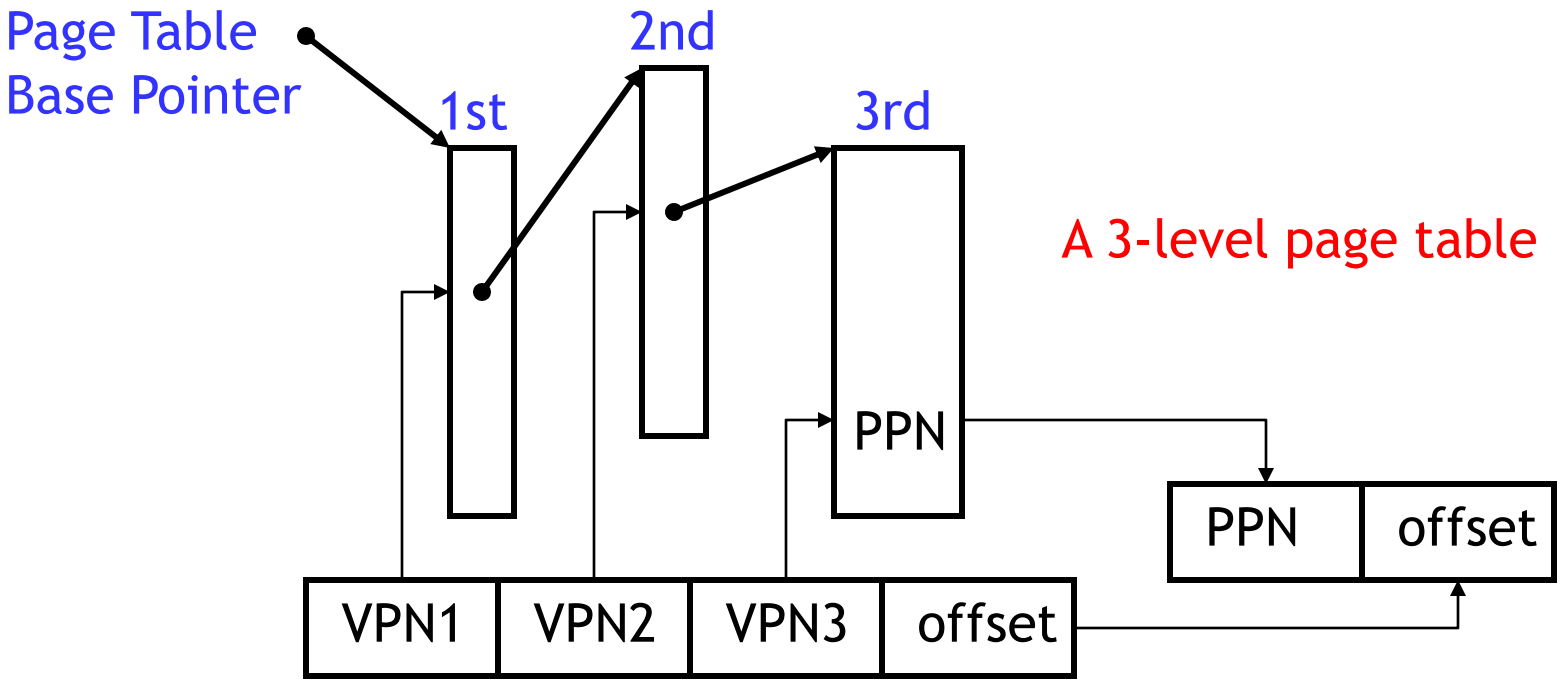
How big is the page table?

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

- How about for a 64b architecture?

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.

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)…

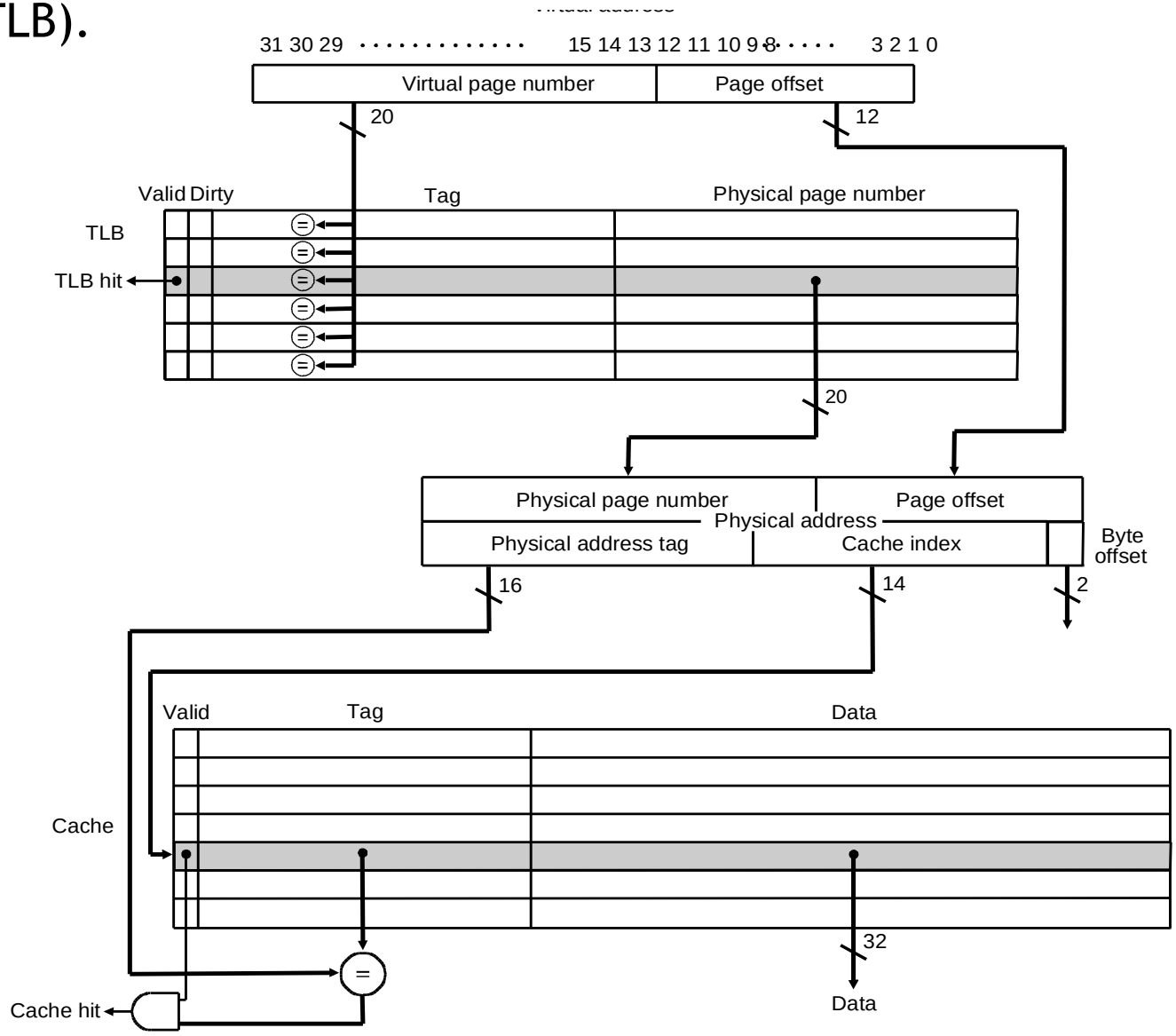
  "Any problem in CS can be solved by adding a level of indirection"

  — except too many levels of indirection…

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

Caching 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

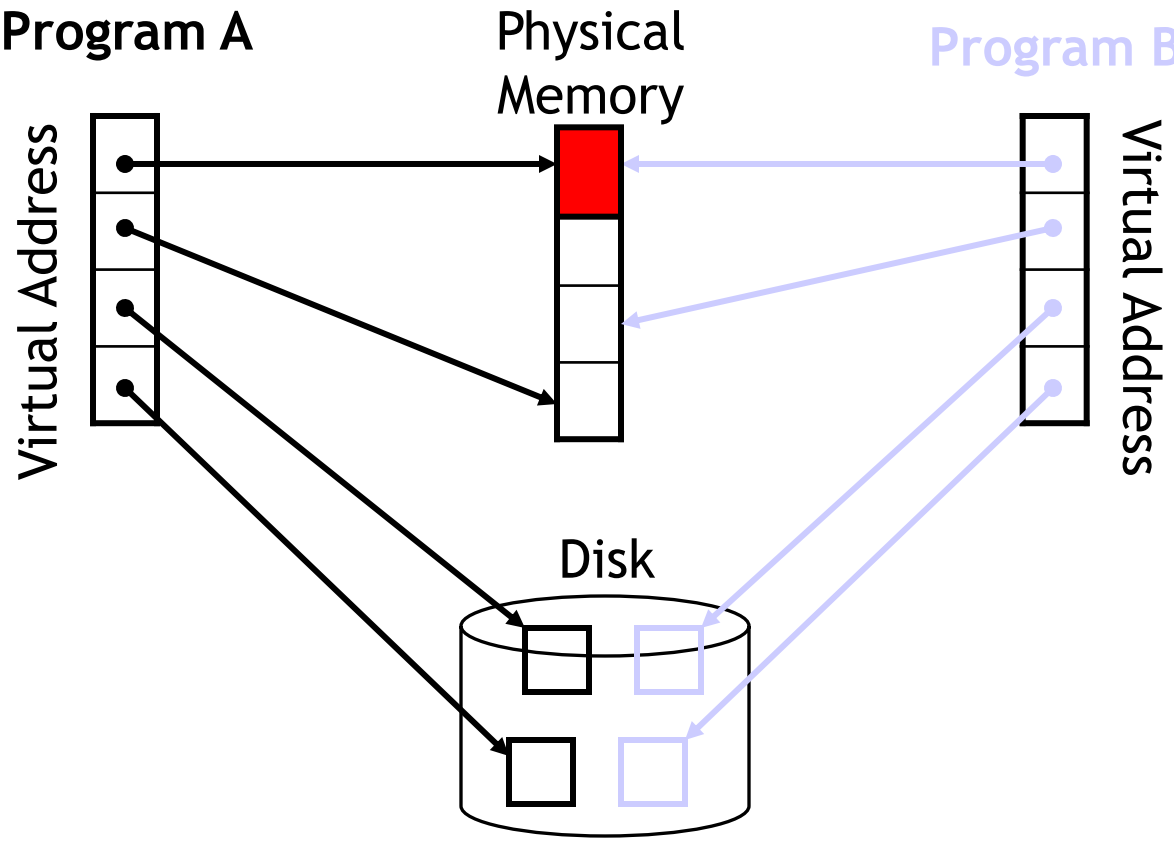
- What if the page is not in memory?
  - This situation is called a **page fault**.
  - The operating system will have to request the page from disk.
  - It will need to select a page to replace.
    - The O/S tries to approximate LRU (see CS423)
  - The replaced page will need to be written back if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