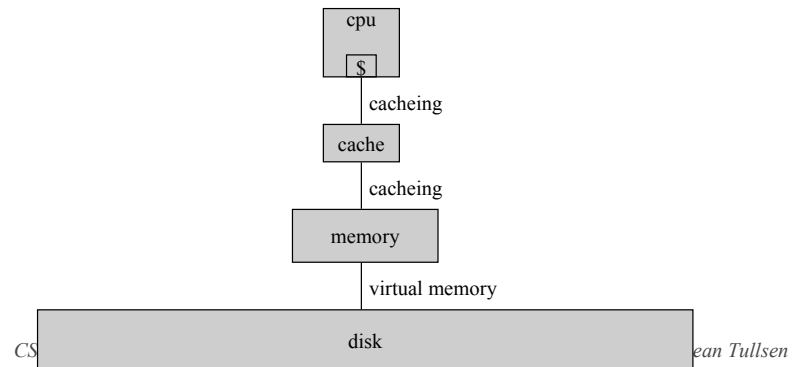


## Virtual Memory

*umm, umm, hang on, it's coming to me...*

## Virtual Memory

- It's just another level in the cache/memory hierarchy
- *Virtual memory* is the name of the technique that allows us to view main memory as a cache of a larger memory space (on disk).



## Virtual Memory

- is just cacheing, but uses different terminology

| <u>cache</u> | <u>VM</u>                  |
|--------------|----------------------------|
| block        | page                       |
| cache miss   | page fault                 |
| address      | virtual address            |
| index        | physical address (sort of) |

## Virtual Memory

- What happens if another program in the processor uses the same addresses that yours does?
- What happens if your program uses addresses that don't exist in the machine?
- What happens to "holes" in the address space your program uses?
- So, virtual memory provides
  - performance (through the cacheing effect)
  - protection
  - ease of programming/compilation
  - efficient use of memory

## Virtual Memory

- is just a mapping function from virtual memory addresses to physical memory locations, which allows cacheing of virtual pages in physical memory.

CSE 240A

Dean Tullsen

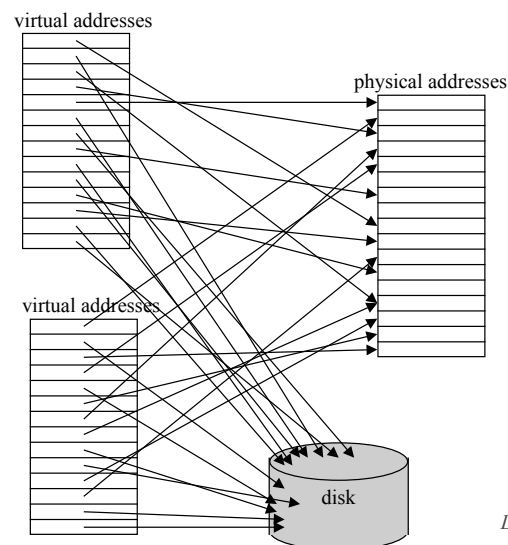
## What makes VM different than memory caches

- **MUCH** higher miss penalty (millions of cycles)!
- Therefore
  - large pages [equivalent of cache line] (4 KB to MBs)
  - associative mapping of pages (typically fully associative)
  - software handling of misses (but not hits!!)
  - *write-through* not an option, only write-back

CSE 240A

Dean Tullsen

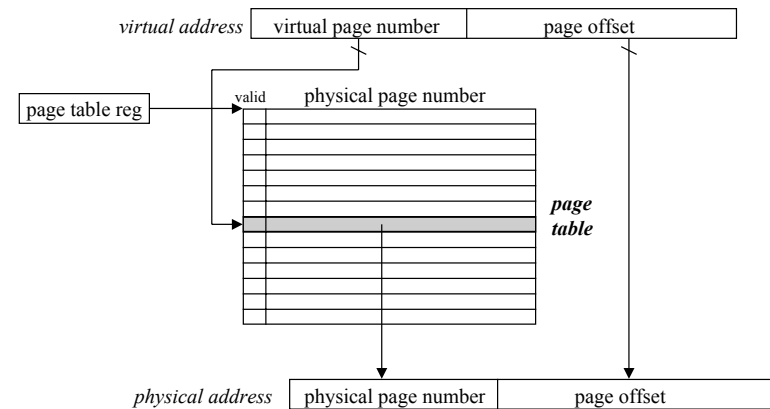
## Virtual Memory mapping



CSE 240A

Dean Tullsen

## Address translation via the page table



- all page mappings are in the page table, so hit/miss is determined solely by the valid bit (i.e., no tag)
- so why is this fully associative???

