Virtual memory

What causes cache misses? 3 Cs

Compulsory – bringing the first blocks into a cache ("warming up" the cache) - **increase the block size**

Capacity – cache not big enough to contain all of the blocks it needs - **increase the cache size**

Conflict – blocks constantly evicted due to cache collisions - **hw3 (associativity)**

Advanced optimization: prefetching

Instead of fetching one block on a miss, fetch two (required + next)

    Store next block in a buffer (**why?**)

Used often with I-cache (**why?**)

Possible to have compiler support with special prefetch instructions

Downsides?

What is an operating system?

Resource management

In an OS, multiple processes at multiple privilege levels might be running “at the same time”

The OS manages how they share/access:

Physical memory

The memory physically available on a computer

Modelled by how we've thought about memory so far

Allows random access to bytes using addresses

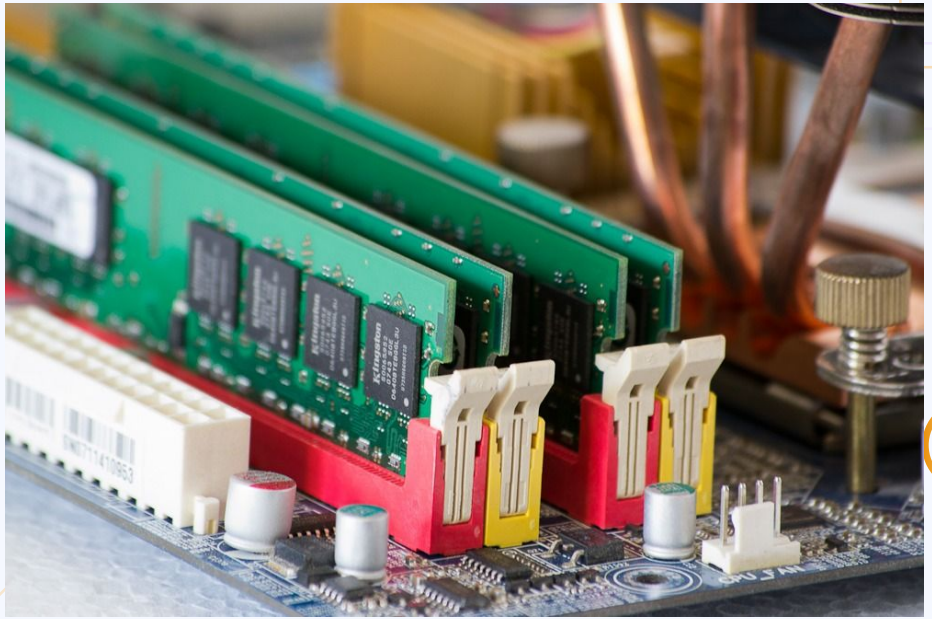


image source

Multiprocessing and memory

Problems with naive interpretation:

A nicer picture

Virtual memory

A way of using main memory as a “cache” for disk storage

OS-managed, hardware supported

Advantages:

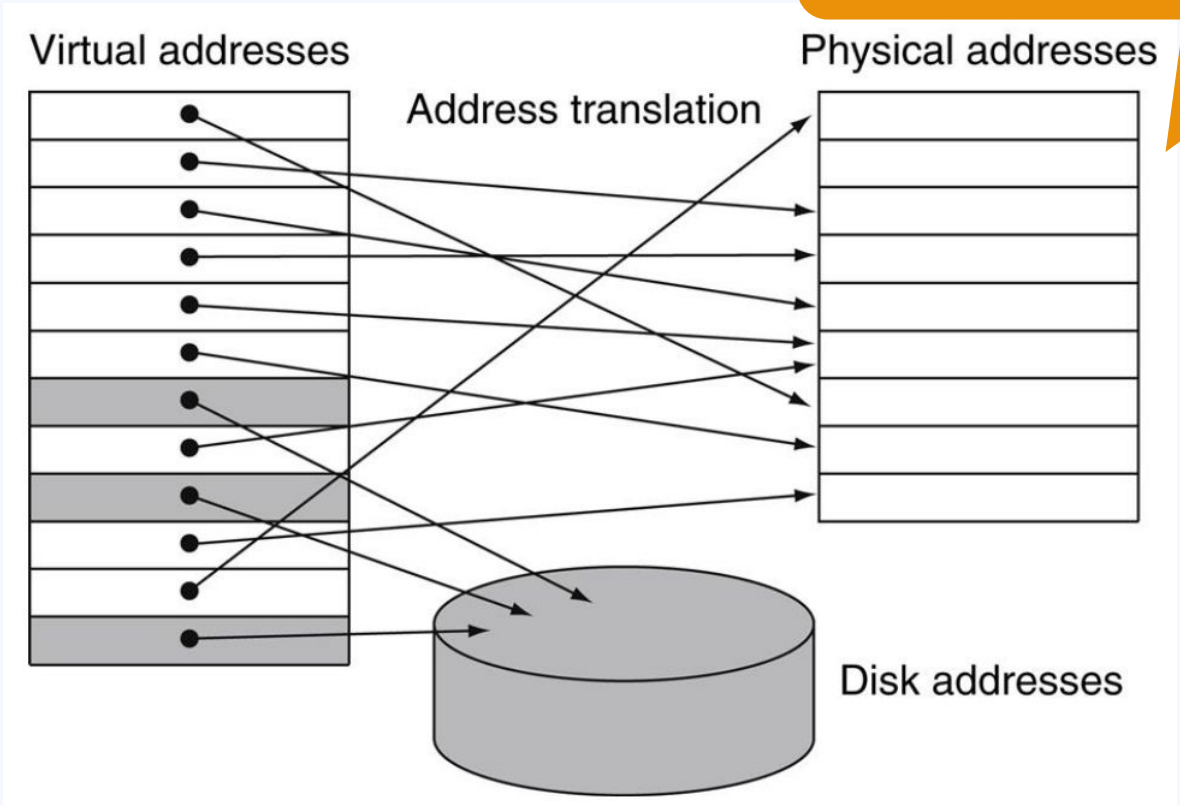
- Per-process memory access management
- Allows process to see larger address space than fits in main memory

Complexities:

- Managing the mapping of physical to virtual memory
- Slow speed of disk

P&H Fig. 5.25

Each process can have its own view of virtual addresses (0-N)



Each “block” of virtual memory is called a **page**

What determines how many physical pages our system has? How many virtual pages?

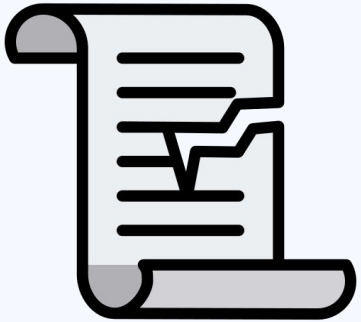
Avoiding page faults

A virtual memory “miss” is called a **page fault**

Virtual page not present in physical memory → have to go to disk

Since disk is even slower (100k* slowdown) than main memory, we want:

- Fairly large page sizes (4KB on Intel chips, 16KB on Apple silicon)
- Flexible placement of pages in memory (virtual page can map to any physical page)



Does it make sense for virtual memory systems to use write-through or write-back?

If virtual addresses can map to *any* physical address, how do we efficiently find a physical page?

Page tables

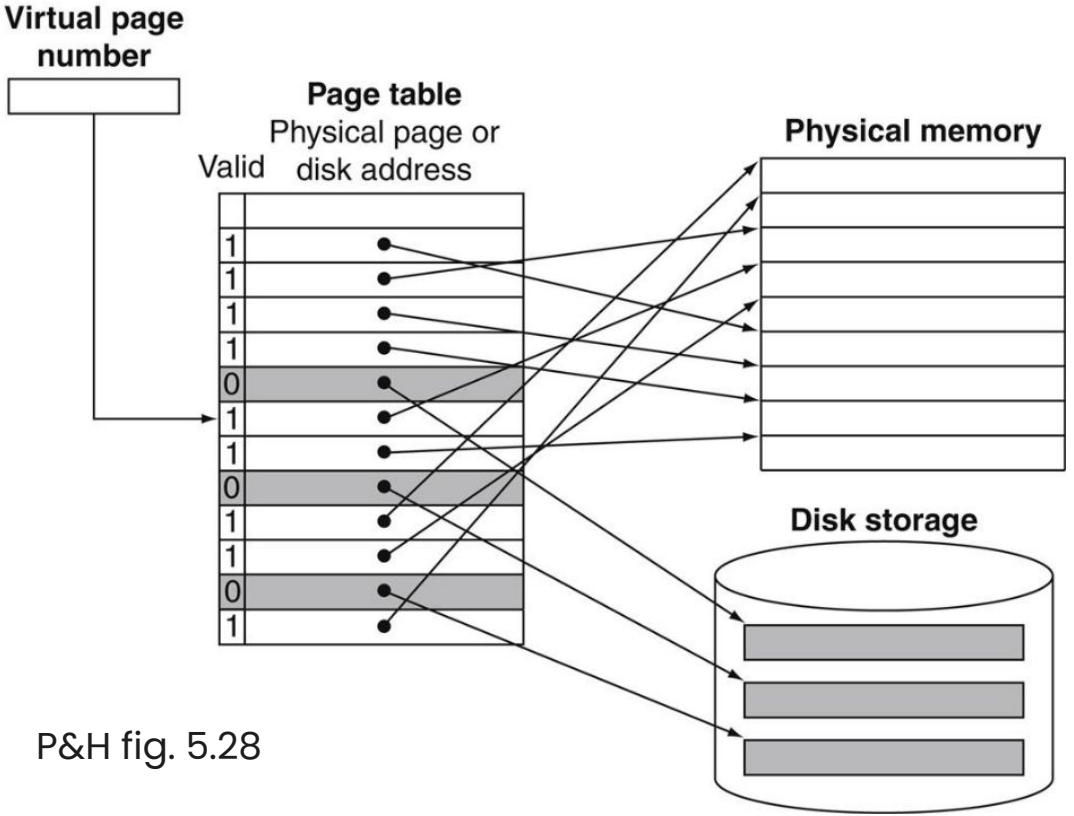
One for each process

Map virtual addresses to physical addresses

Not a cache – **why?**

Page faults managed by OS - **why?**

Where does the page table live?



P&H fig. 5.28

? ? ?

What should happen if our virtual address space is so big that the page table can’t efficiently fit in main memory?