Board Notes on Virtual Memory

Why Virtual Memory?

- Let's user program size exceed the size of the physical address space
- Supports protection
    - Don't know which program might share memory at compile time.
- Usually, virtual address space is *much* greater than physical address space
    - (Mapping allows code with virtual address to run on any machine.)

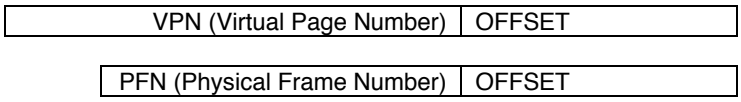
How do we translate a Virtual Address to a Physical Address
(or alternatively, "How do we know where to start looking in memory?")

- Good analogy: It's like finding what cache block a physical address maps to.

Example:

- What if 32-bit virtual address (2^{32} virtual addresses), 4KB pages (like above), 64 MB of main memory (2^{26} physical addresses)

How is this mapping done?

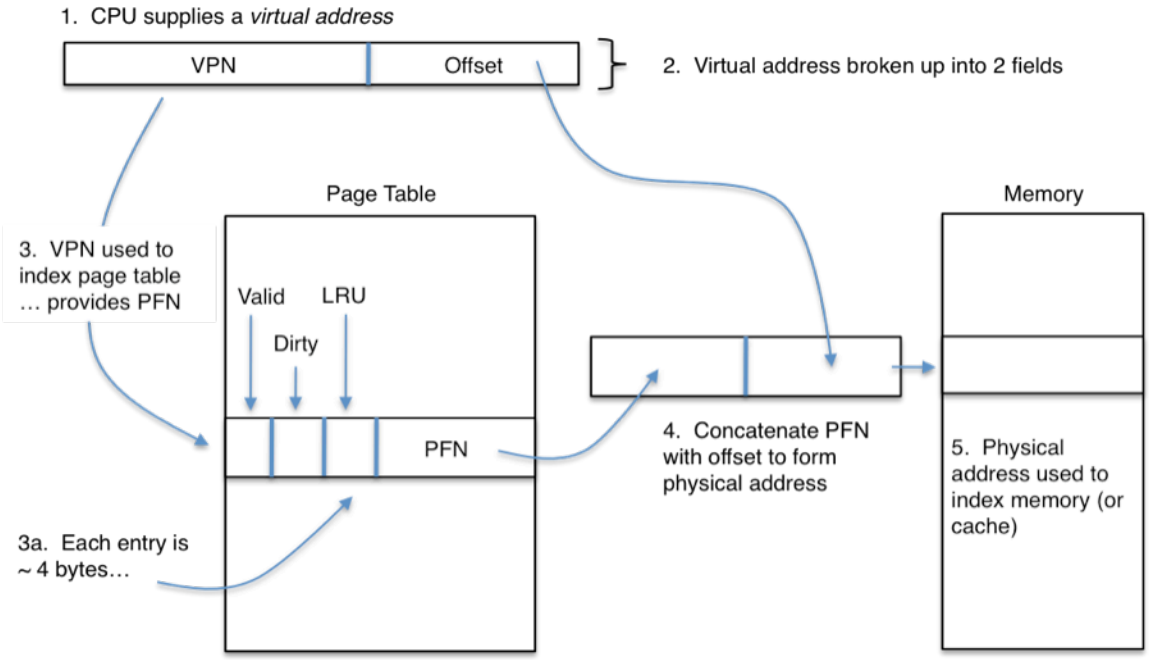


How do we do VPN → PFN mapping?

- Leverage structure called page table
- To make analogy to cache, "data" = PFN
- To make analogy to cache, also have valid, dirty bits
- If no valid mapping, get page fault:
    - Try to avoid
    - Involves lots of disk traffic
    - Placement in memory done fully associative, LRU to minimize
    - Placement = some extra overhead, but small percent – and worth it to avoid M CC penalty

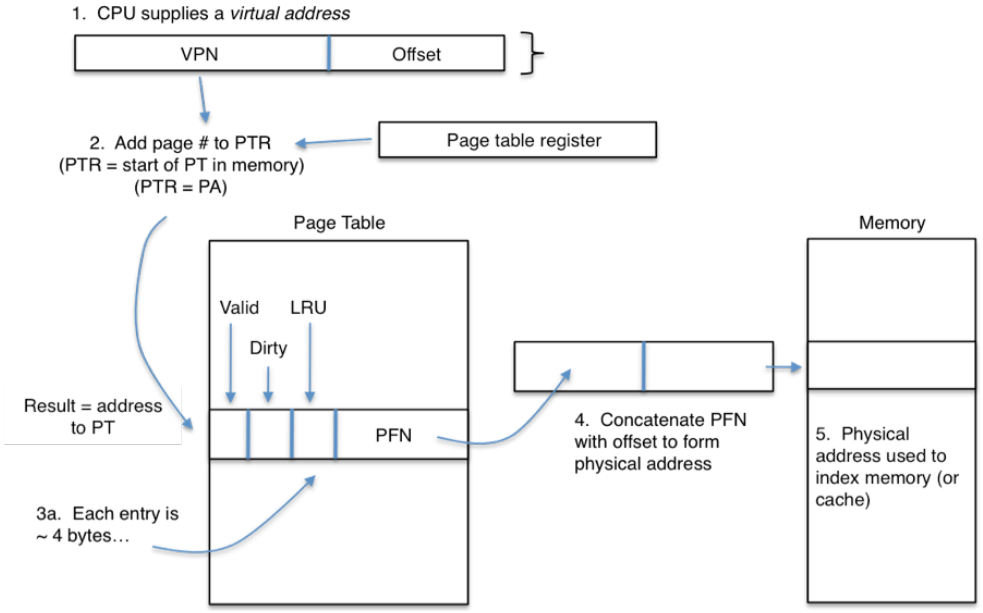
Offset still the same because we go down the same distance

More specifically:
The process works like this…



Even more specifically…

- The page table is stored in memory
- The beginning of the page table is stored in the page table register

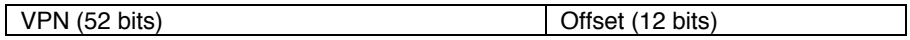


How big is the page table?

- Page table can actually become pretty big…
- Example #1:
  - 4 KB pages
    - Therefore need 2^{12} (or 12 bits of offset)
    - (Offset does same thing that it does in cache block – not just picks page entry)
  - 32-bit virtual address
    - 32 – 12 = 20 bits of VPN
  - 4 Byte / page table entry
    - Holds LRU status, valid, dirty, PFN (~32 bits)
- Therefore, 2^{20} entries in page table, each ~ 4 bytes each → 4 MBytes

Another Example…

- Assume
  - Virtual address = 64 bits
  - 4 KB pages
  - 4 bytes/page



- PT would be:
  - $4.5 \times 10^{15} \times 4 \sim 10^{16}$ bytes → 10 *petabytes!*
- Soluton(s):
  - Multi-level, inverted page tables – you'll learn about in OS

Is page table / virtual address translation slow?

- It can be → have maybe 2 references / translation
- Solution: TLB = "Translation Lookaside Buffer"
  - Fast cache for page table

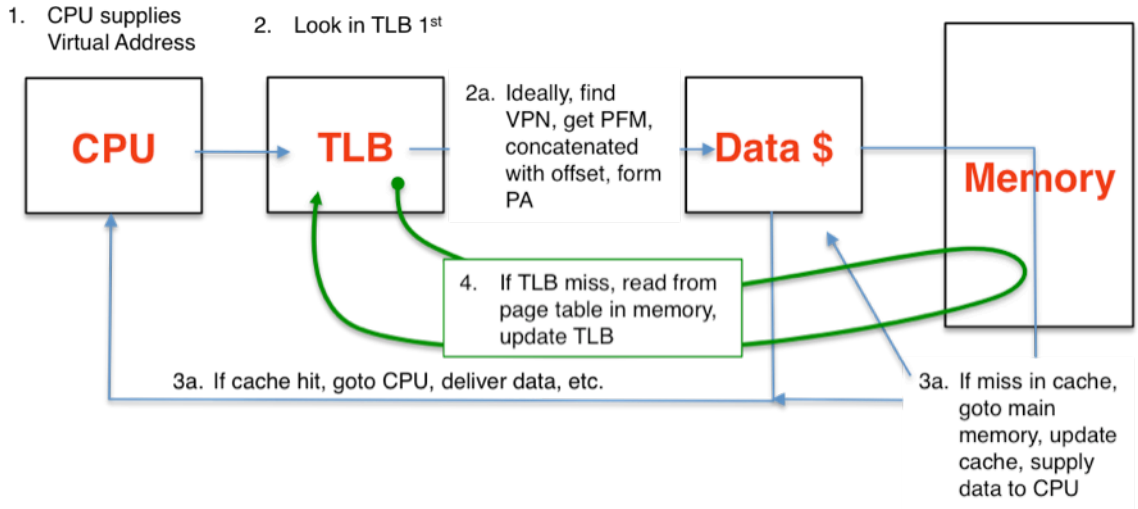
What does the TLB look like?

- It's a really small, fully associative cache

Virtual Page #	Physical Frame #	Dirty	LRU	Valid

1. All of the virtual page numbers would be searched for a match
2. The physical frame number is the data that is supplied
3. The physical frame number is concatenated with an offset to form a physical address

Where is the TLB on the critical path?



See flow chart below / in notes: