11/2 Paging

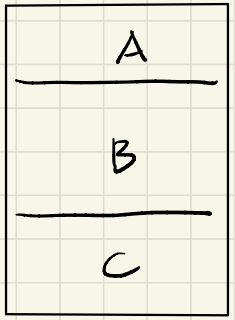
Physical Memory Allocation

→ multiple processes need to share the physical memory

→ last class : base & bound

→ virtual memory for each process (own view, independent from where it is in physical memory)

→ address translation is simple : if(vaddr < bound) { vaddr + bound; }



physical memory

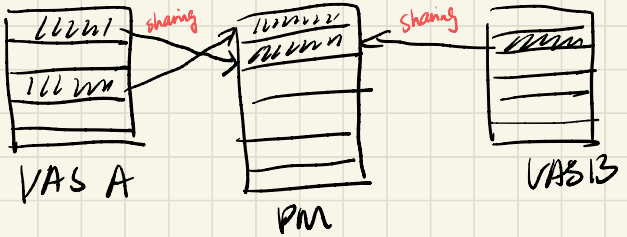
Limitations

→ memory fragmentation (variably sized processes)

→ inefficient use of physical memory (processes don't need all memory at once)

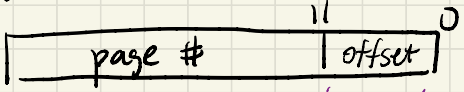
→ hard to implement memory sharing

☆ paging : divide virtual memory into pages, physical memory as frames, and map a page to a frame



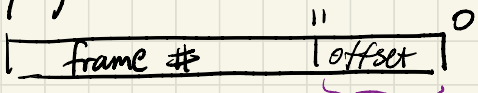
Address Translation For Paging

Virtual address (64bit)



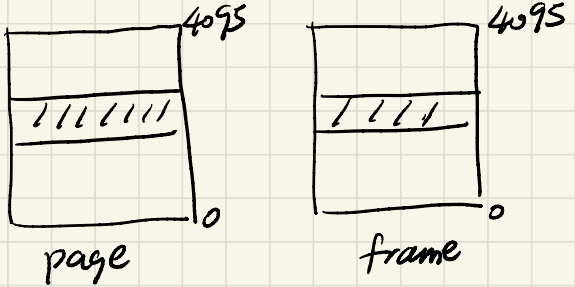
12 bits (2^{12} = 4096 = 4kB page)

physical address



12 bits

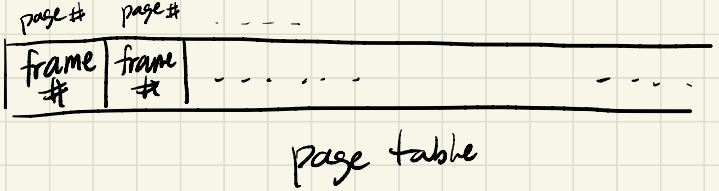
Need a way to translate page # to frame #



offset (location) within a page is the same within a frame, no need to translate

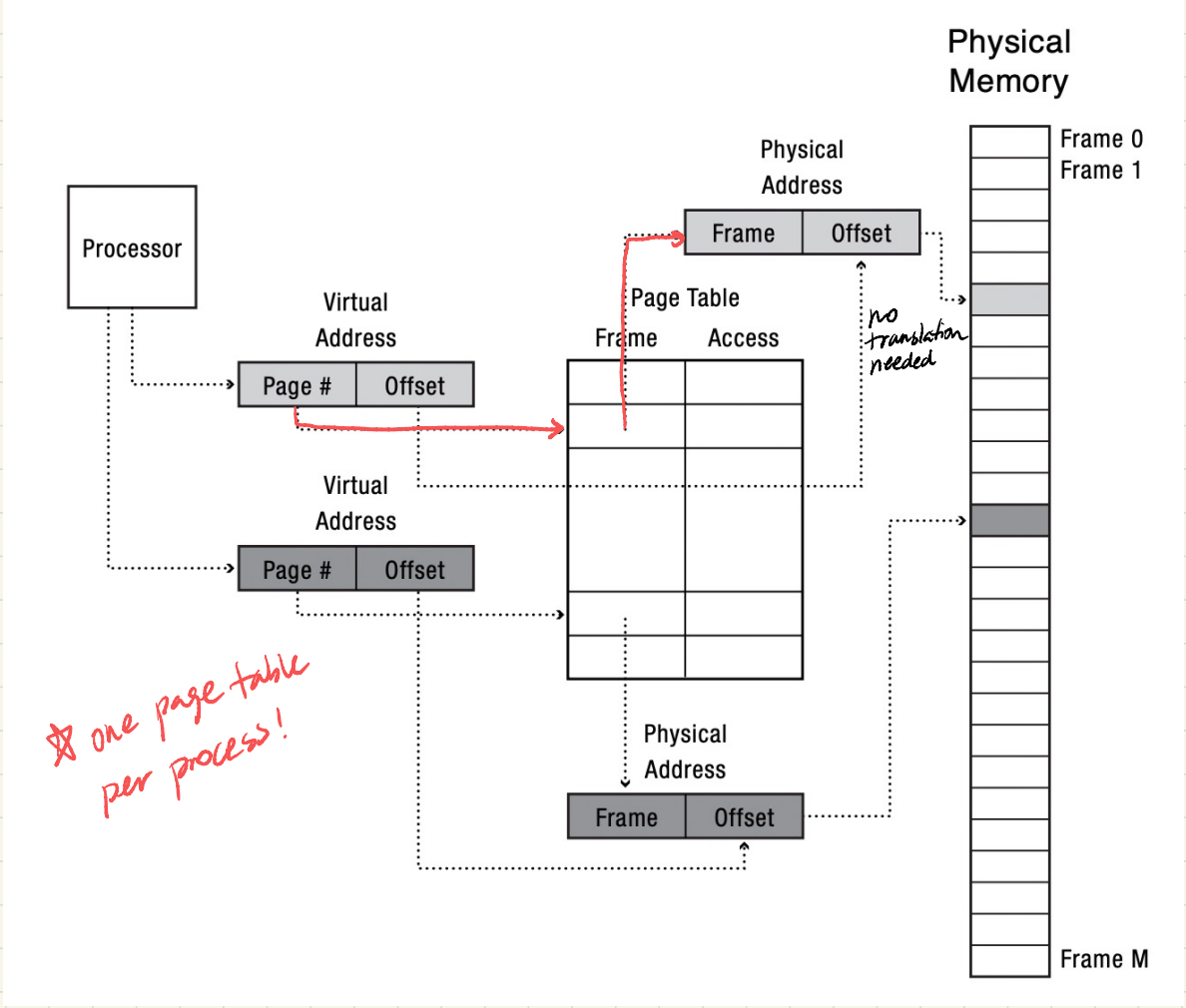
What data structure can we use as the translation (look up) table?

→ simple approach: a single array with an entry for each page



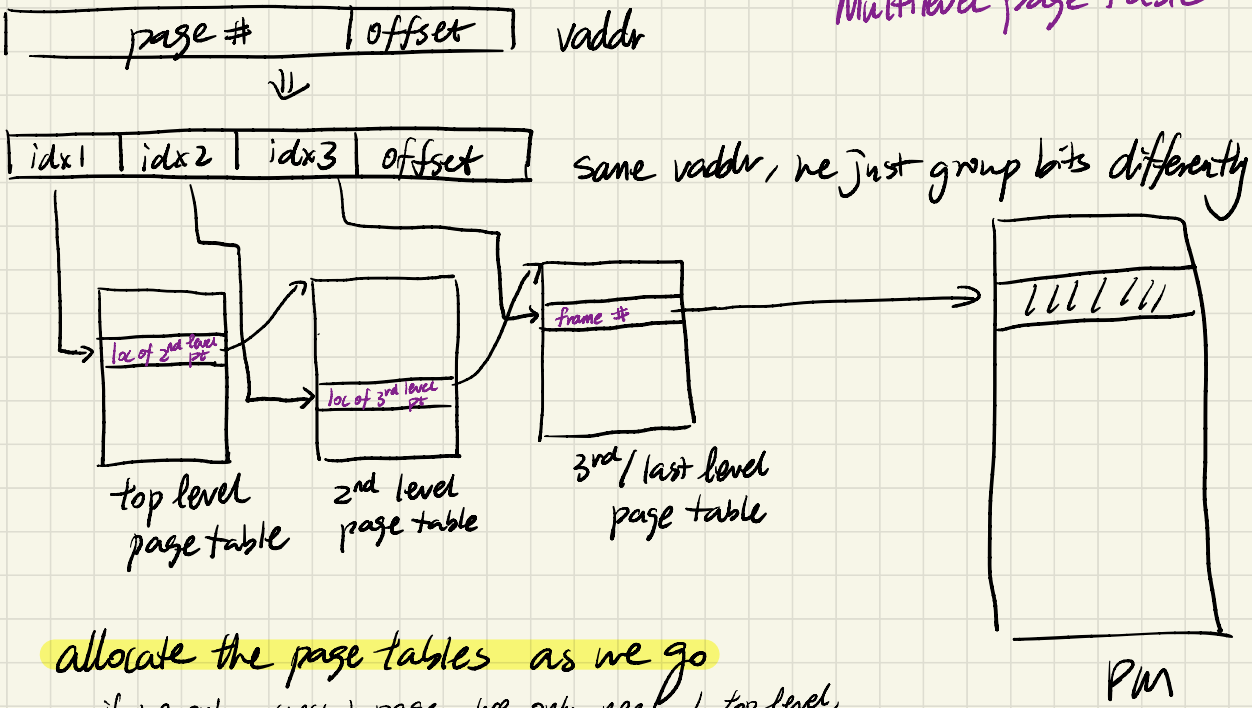
How many entries do we need?

2^{52} ($2^{64}/2^{12}$) entries!


Physical Memory
Processor
Virtual Address
Page # Offset
Virtual Address
Page # Offset
Page Table
Frame Access
Physical Address
Frame Offset
no translation needed
Physical Address
Frame Offset
Frame 0
Frame 1
Frame M
* one page table per process!

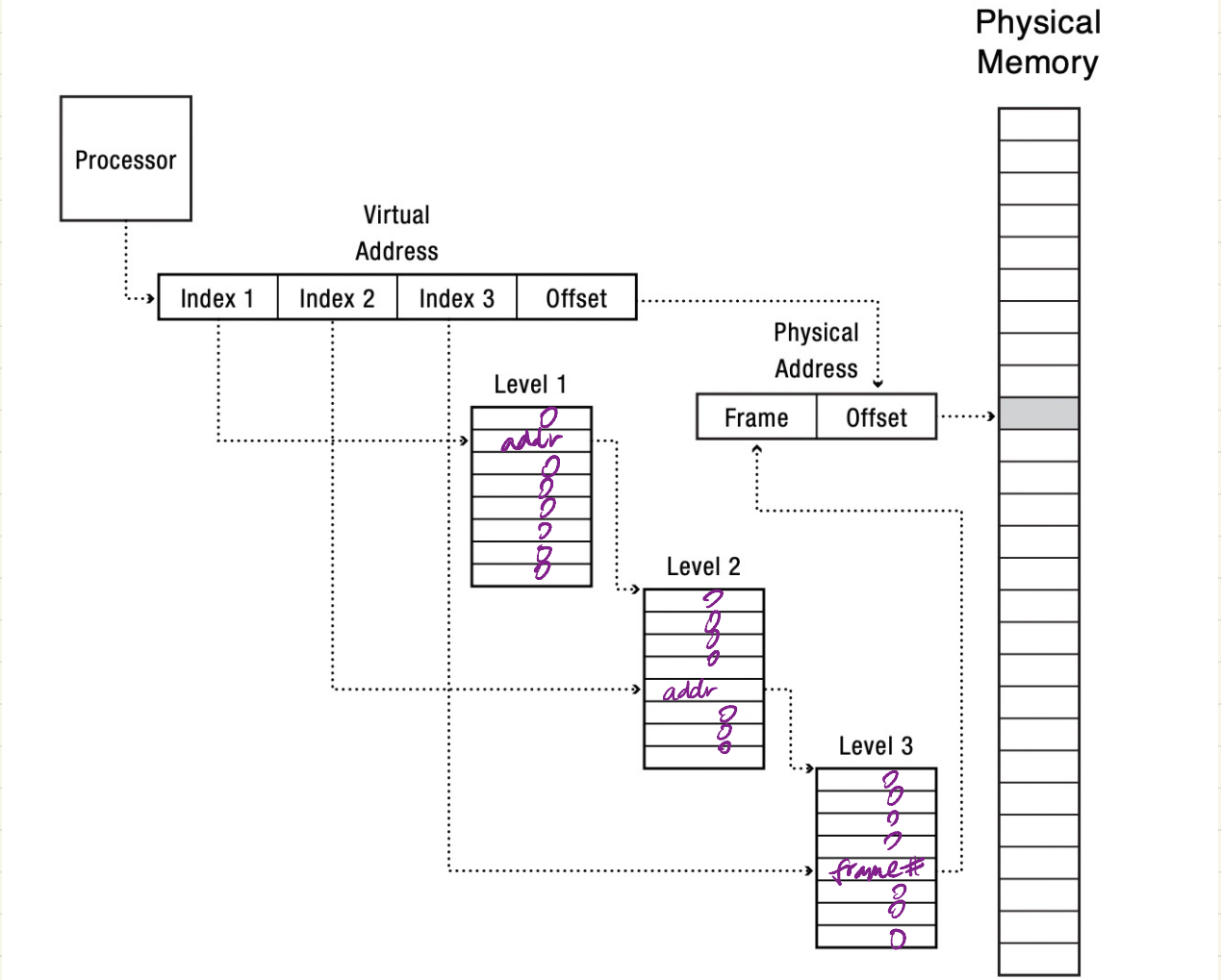
A single page table is too large in size, especially since most programs don't use that much memory! To save space, we can introduce layer

Multilevel page table



allocate the page tables as we go

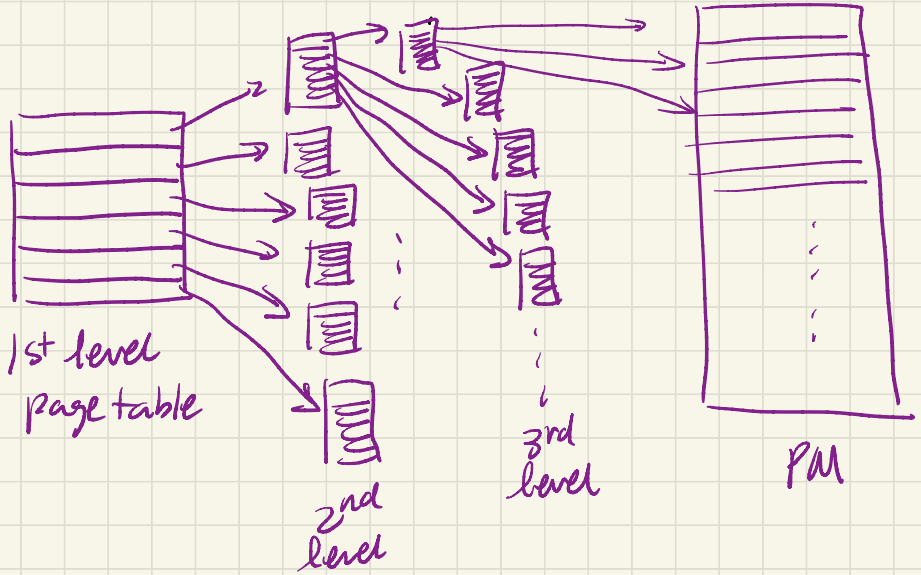
- if we only access 1 page, we only need 1 top level, 1 2nd level and 1 last level page table (each is much smaller than a PT of 2^{52} entries)


Physical Memory
Processor
Virtual Address
Index 1
Index 2
Index 3
Offset
Physical Address
Frame
Offset
Level 1
addr
Level 2
addr
Level 3
frame#

Is multilevel page table always smaller than single level?

→ What if we use all of the virtual memory?

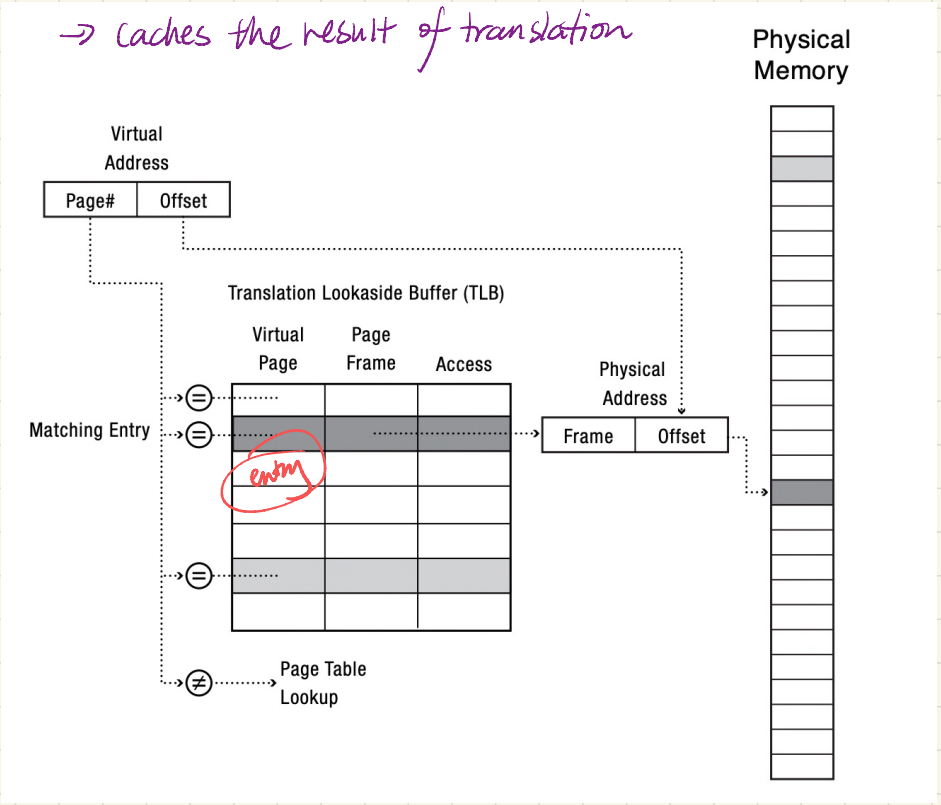
→ need to allocate the first level page table, all of 2nd level page table, and all of 3rd level page table. (last level page tables store the actual page → frame translation, a total of 2^{52} entries)



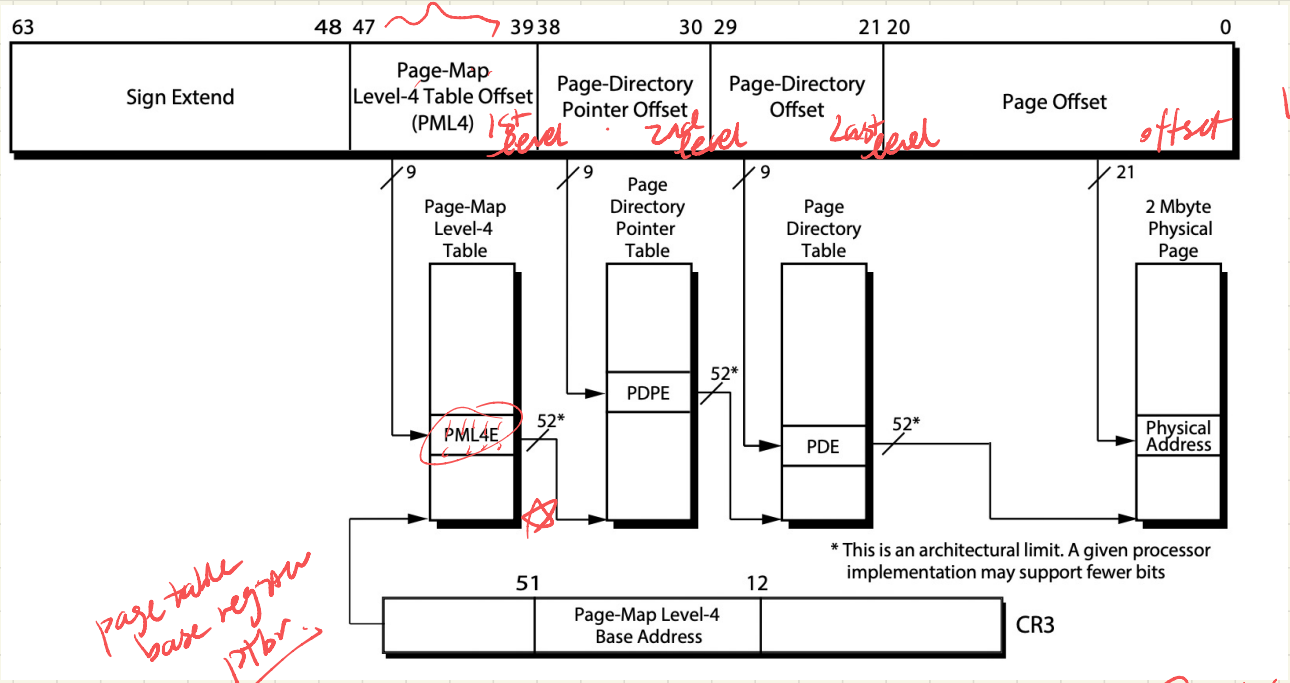
Now we mostly solved the space problem, next is the cost of page table look up.

① Cache : Translation Lookaside Buffer (TLB)

→ caches the result of translation



② Make hardware walk the page table for you
→ hw needs to understand the page table format.



page fault
① missing entry
② mismatch permission