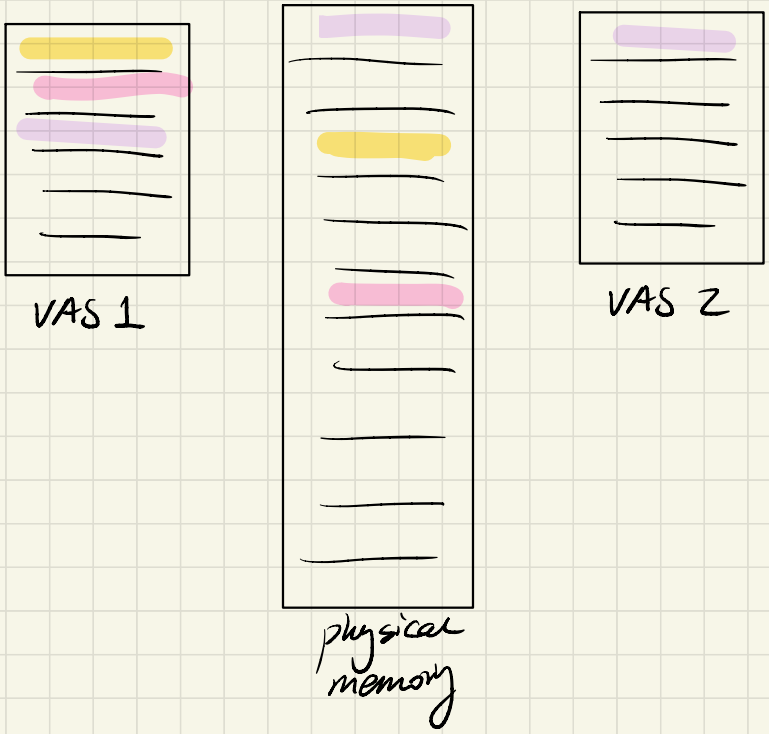
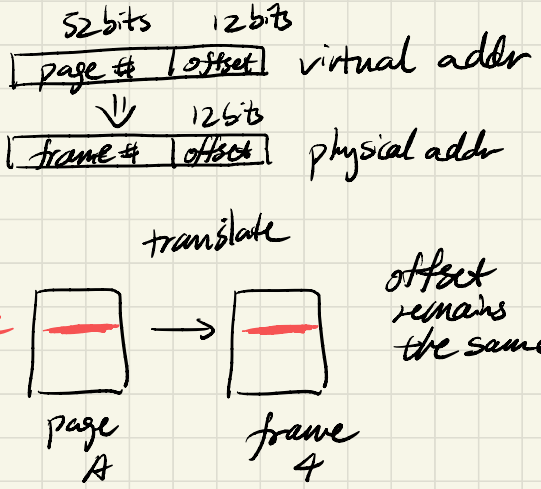
4/24/24

Paging

→ divide virtual & physical memory into fixed size pages

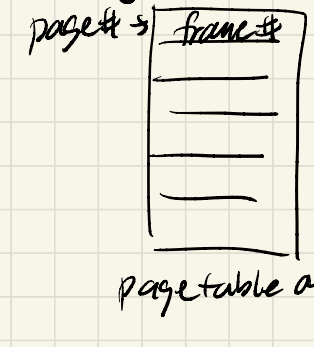


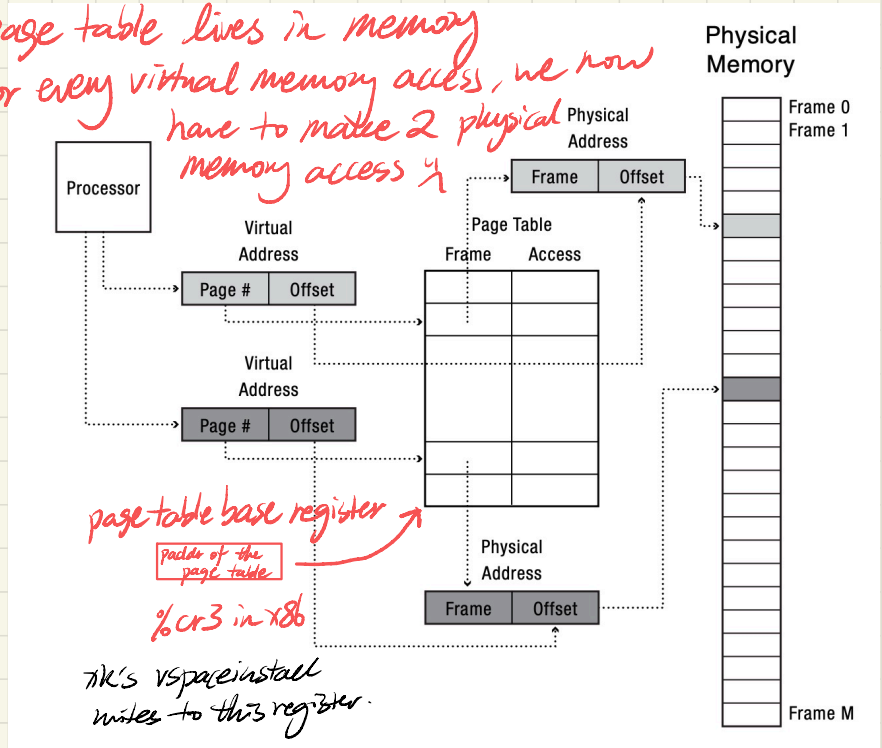
→ page level translation



Page Table: stores translations for every page (hw page table walk)

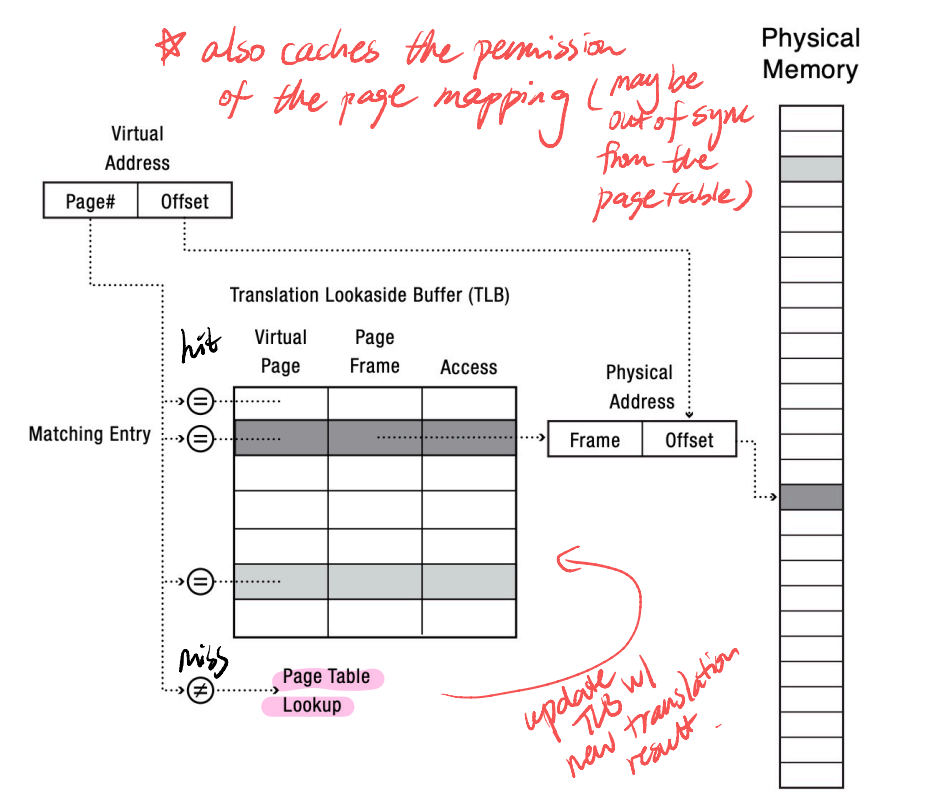
→ Kernel sets up the page table, translation performed by hw





Translation Lookaside Buffer (TLB)

→ caches the result of memory translation



The Cost of single array page table

→ per process data structure

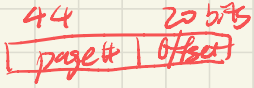
→ 2^{52} entries take up a lot of space

How to reduce the page table size?

→ Larger page size: 2 MB & 1 GB page size (large / super pages)

⇓ less pages, fewer entries

⇓ smaller page tables



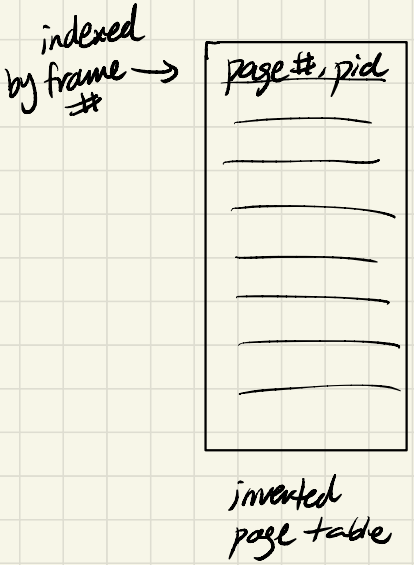
$2^{52} \Rightarrow 2^{44}$ entries

but ... 

internal fragmentation!

Maybe the problem is too many page tables?

→ Inverted page table: tracks frame mapping instead (global)



→ How to look up given a virtual address?

→ search through each entry of the array until we find a matching page # w/ pid.

works but very slow :(

→ Use a hash function to place pages

→ lookup = hash(page #, pid) % frames ⇒ frame #

what about hash collisions?

shared memory?

cost of unmapped page?

Page ⇒ frame mapping still better for look up!

Multilevel Page Tables: use indirection to only allocate entries for pages in use