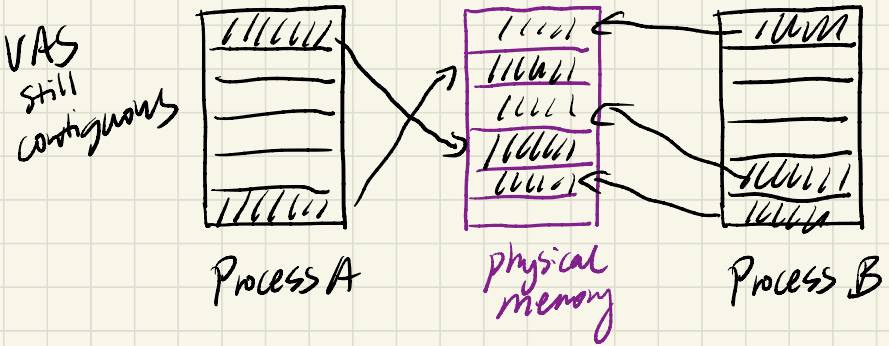
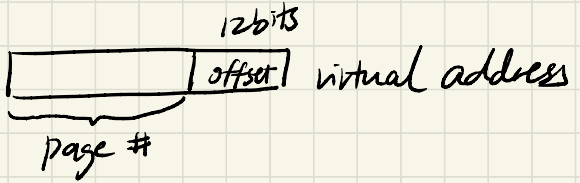
11/1/23

Paging

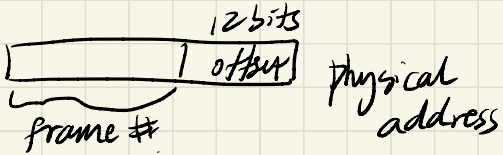


- fixed sized pages (4096) 4KB.
- same for physical memory (frames, physical pages)



0x1000 – 0x1fff page 1

0x2000 – 0x2fff page 2



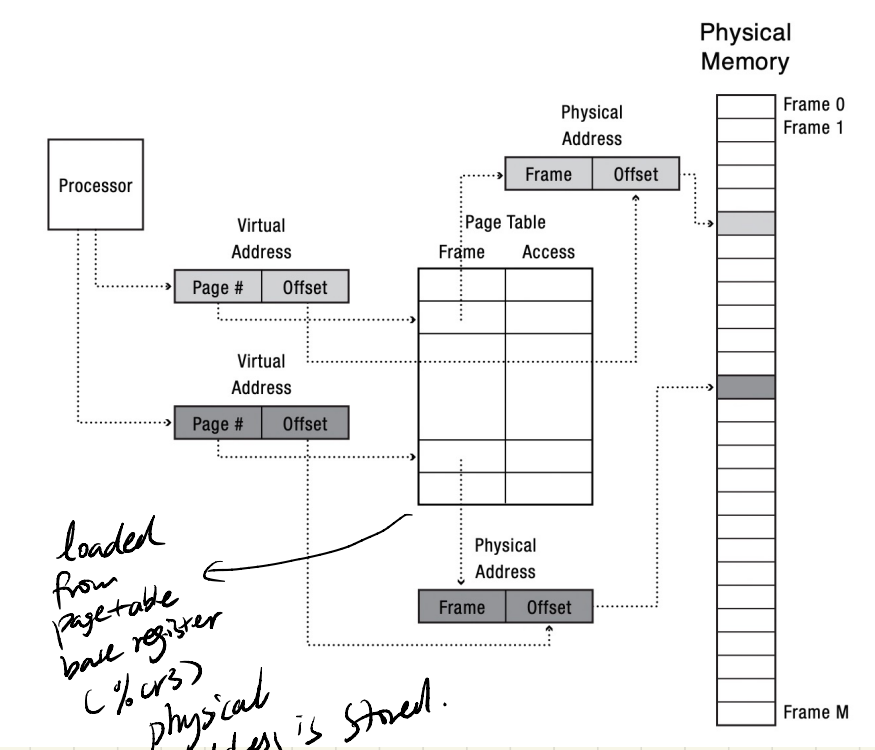
→ base & bound (contiguous translation)

→ paging: translation for each page

page # → frame #

Page Table

→ a translation array, indexed by page #



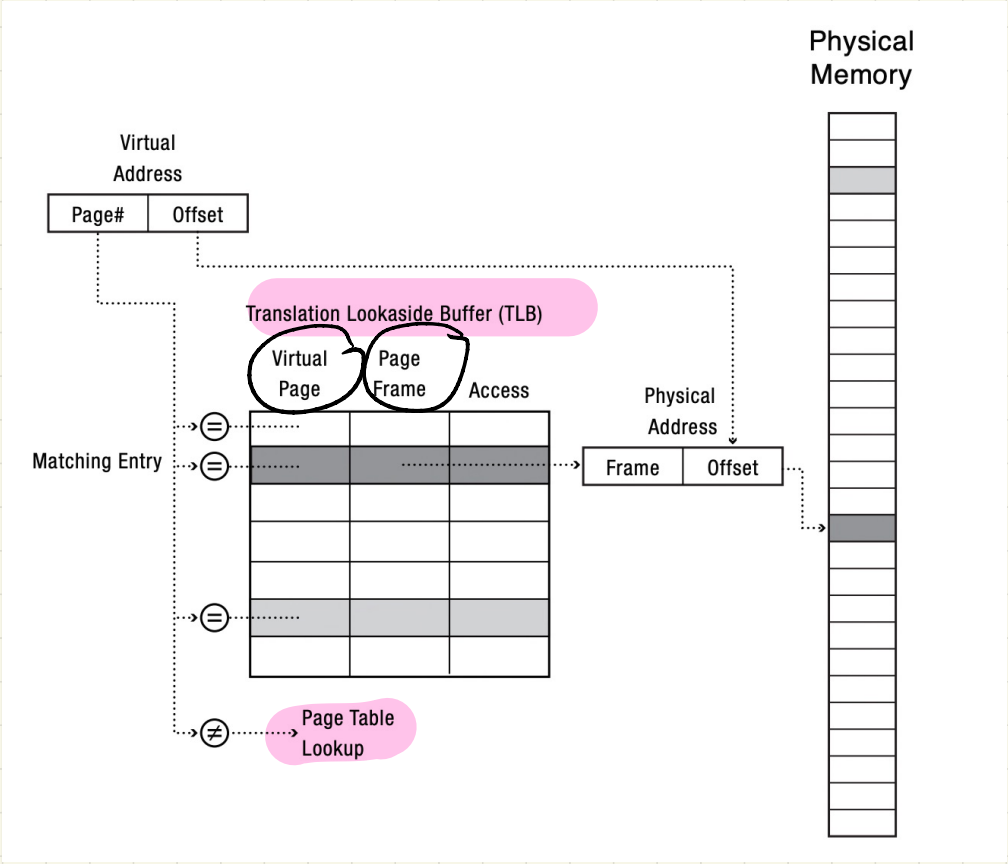
- page table walk (translation) done by the hardware
- page table structure defined by architecture

How to speed up translation?

→ cache the result!

Translation Lookaside Buffer (TLB)

(pid + page #) ⇒ frame #

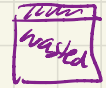

Physical
Memory
Virtual
Address
Page#
Offset
Translation Lookaside Buffer (TLB)
Virtual
Page
Page
Frame
Access
Physical
Address
Matching Entry
Frame
Offset
Page Table
Lookup

→ Single array : many pages, lots of entries (space), one per process

How to reduce cost?

→ Larger Pages : bigger page size ⇒ less pages & less translation overhead
(2MB & 1GB supported)

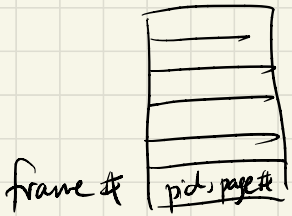
★ problem: internal fragmentation
minimum allocation unit is too large



→ Inverted Page Table : frame # ⇒ page #, global table

hash function (pid + page #)

→ takes time, hash collision



→ Multi-level Page Table