↳ there are two different segments for initialized and uninitialized variable in vm, because uninitialized ones do not occupy any space in the executable

VM ⟶ MMU (memory management unit) ⟶ physical memory

*maps the addresses (first checks TLB if not exists)

↳ if you execute two hello word programs concurrently, their virtual memory addresses will be same but they map to different physical memory addresses

↳ CPU works with virtual addresses, does not know physical memory

TLB (translation lookaside buffer) = cache for translation

memory management techniques

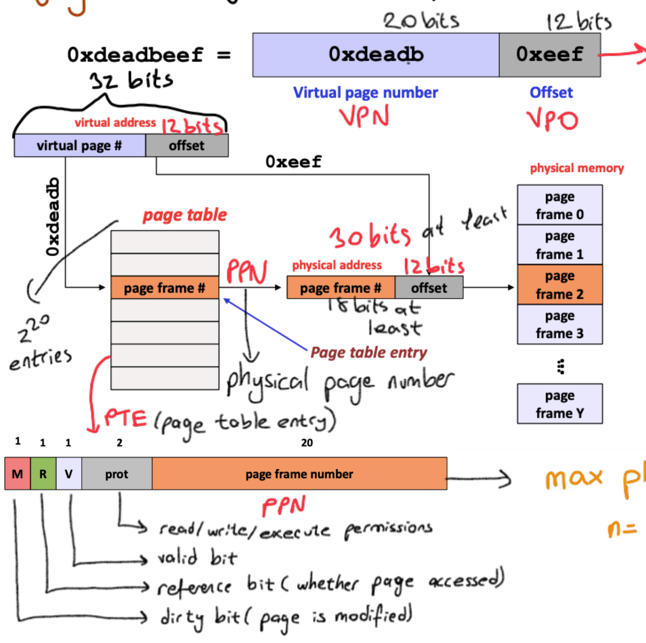
fixed partition = base register (3K) + offset (virtual address) → physical memory

• for every process only change base register to isolate, but partitions into static/single sizes, internal fragmentation, fast

variable partition = limit register (size of the program) + base register + virtual address → physical mem

• external fragmentation

paging = N pages for one process ⟶ N same size frames in physical mem



→ which line to look in page frame

page size = 2^k (2^{12})

k = # of bits used in offsett

VM size = 2^{32} bytes

|VA| = 32 bits

if |Page| = 2^{12} bytes → 12 → page offset

if |PM| = 2^{30} bytes → |PA| → 30 bits

max physical mem supported = 2^n × (1 frame size)

n = page frame number bits

• different page table for different processes (different mapping)

page table base pointer register

PT = VPN → PPN (~~VA → PA~~)

table would be huge

PT size = 2^{20} (entries #) × 4 (one entry) = 4MB

• page tables are also stored in physical memory, costly to reach every time → cache (TLB)

• TLB caches page table entries

page fault = when virtual address translation cannot be done (valid bit = 0)

↳ can also happen due to protection fault (no needed rights)

• when a process starts to execution, its not brought into physical mem from the disk, all valid bit in PT are = 0, if you try to read it, it will be brought into physical mem after page fault → demand paging

temporal locality = memory accessed recently tends to be accessed again soon

spatial locality = memory locations near recently-accessed mem is likely to be referenced soon

page replacement policies

random = lowest performance limit

optimum (min) = highest performance limit, assumed to know future page references

↳ picks the latest to be accessed

FIFO = throw out pages in order what they were allocated - keeps them in a queue, insert at buttom and swich all to top

↳ suffers from Belady's algorithm = performance can get worse when increase the physical mem

LRU (least recently used) = the frame with minimum/oldest timestamp is throwed

↳ keeping the timestamps in memory and sorting to find min is costly

LRU with additional reference bits = keep a 3/4 bit counter per page, periodically scan all pages in physical mem, if shift counter with reference bit, then clear reference bit

• t=1 1000 (counter, page is accessed) t=2 0100 (page is not accessed) t=3 1010 (accessed) throw the page with lowest counter

second chance - clock = scan PTE starting from the clock hand, if ref bit = 1, clear it, and give it a second chance else (=0) throw the page

second chance - queue = iterate over the FIFO queue instead of PTE

enhanced second chance - clock = consider also modify bit

select order ⇒ (0,0) no ref, not modified → (0,1) (no ref, modified) → (1,0) → (1,1)