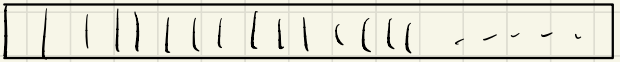
11/3/23

Cost of Address Translation (Paging)

- → single array : page # ⇒ frame #
- → large page : internal fragmentation
- → inverted page table : frame # ⇒ page #
- → multilevel page table

Page Table

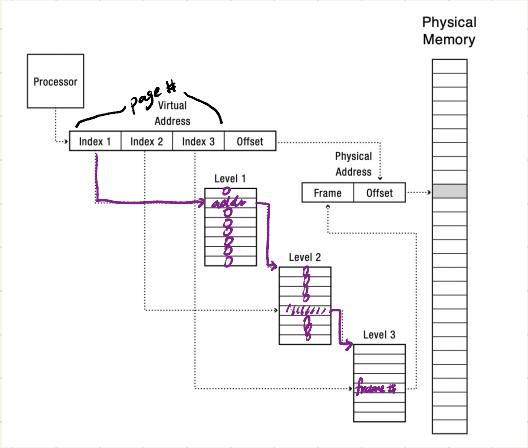
needs to allocate entry for each page



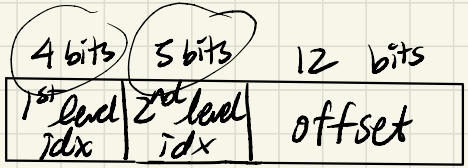
page (only a couple bytes used)

hash collision

page sharing the same frame



21 bit virtual address



[0x1000 - 0x1fff] code 0000 00001

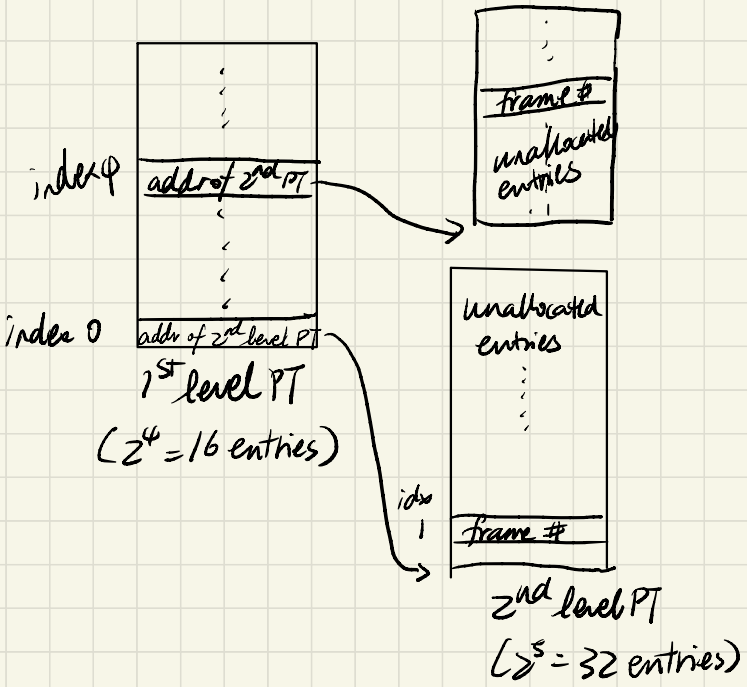
[0x8f000 - 0x8ffff] stack 0100 01111

4KB page size

9 bits for page #

2^9 = 512 pages max

Single level PT always allocate 512 entries



2 page tables needed to map just the code page (one 1st level + one 2nd level)

16 + 32 = 48 entries

3 page tables total when we map the stack page (one 1st level + two 2nd level)

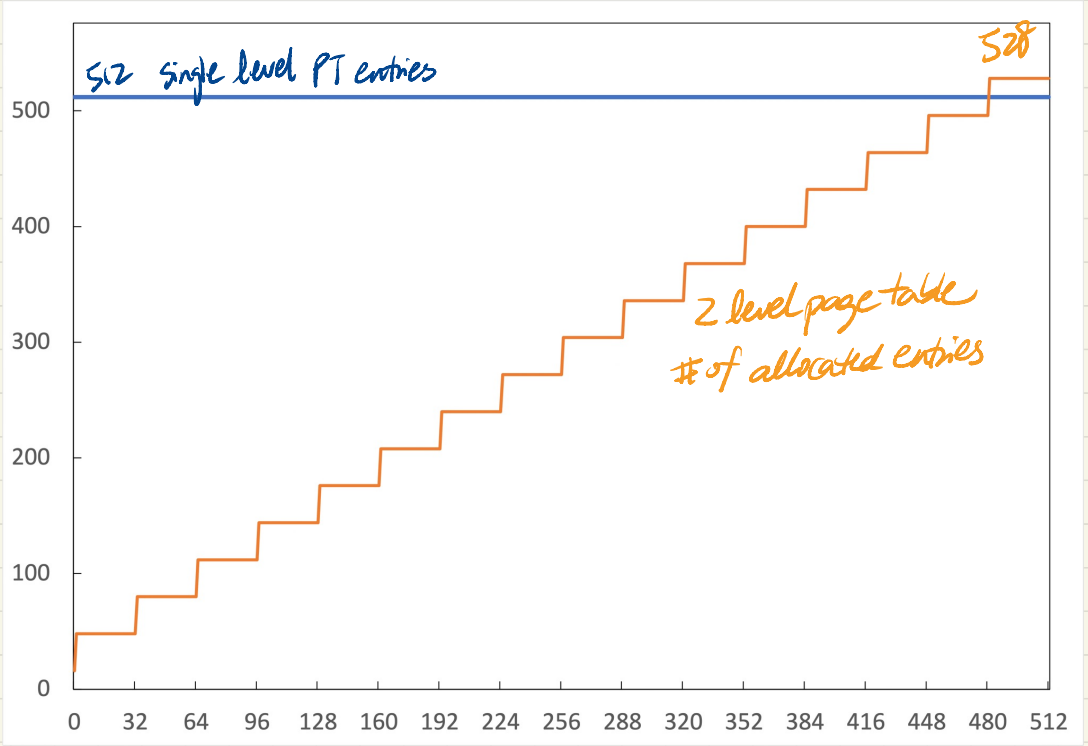
16 + 32×2 = 80 entries

21 bit virtual address

4 bits	5 bits	12 bits
1st level idx	2nd level idx	offset

1st level PT = 2^4 = 16 entries

2nd level PT = 2^5 = 32 entries



# of pages used (contiguous pages)

☆ Not always smaller than single level

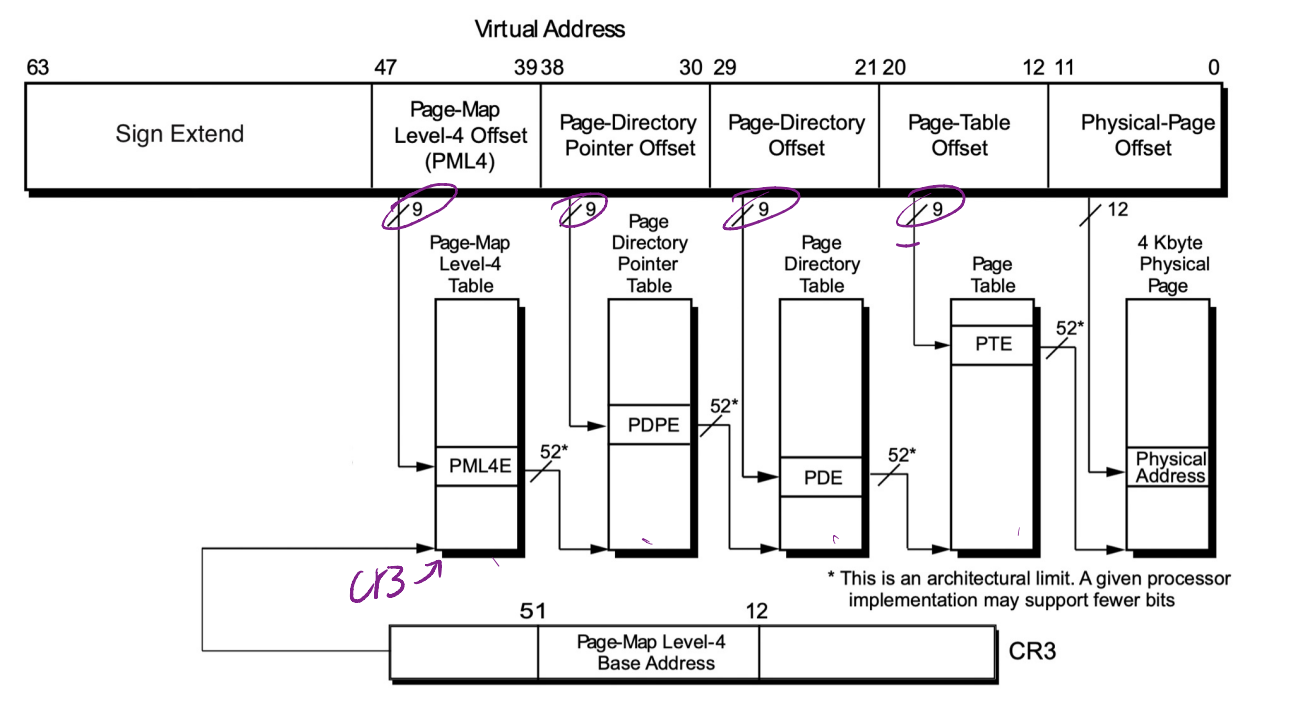
X86-64 Address Translation

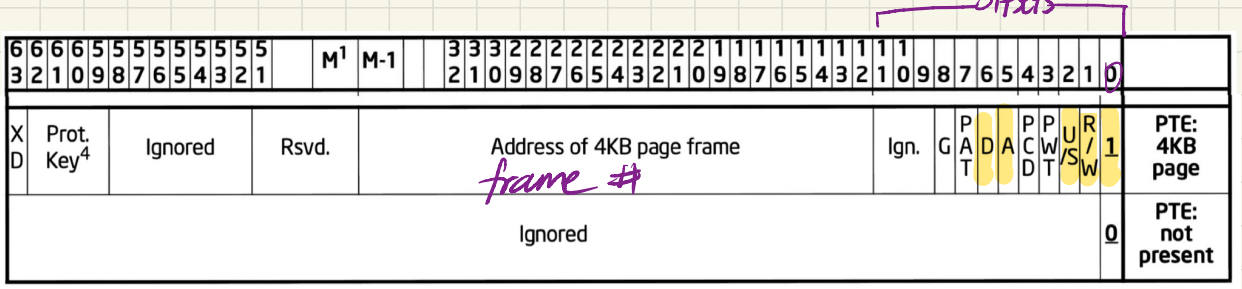
- 4 level page table, each page table takes up a full page in memory

$2^9 = 512$ entries, 8 byte per entry

$512 \times 8 = 4096$

x86_64vm.c (xc)





Bit Position(s)	Contents
0 (P)	Present; must be 1 to map a 4-KByte page
1 (R/W)	Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)
2 (U/S)	User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)
3 (PWT)	Page-level write-through; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
4 (PCD)	Page-level cache disable; indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
5 (A)	Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)
6 (D)	Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)
7 (PAT)	Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)
8 (G)	Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise
11:9	Ignored
(M–1):12	Physical address of the 4-KByte page referenced by this entry
51:M	Reserved (must be 0)
58:52	Ignored
62:59	Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise
63 (XD)	If IA32_EFER.NXE = 1, execute-disable (if 1, instruction fetches are not allowed from the 4-KByte page controlled by this entry; see Section 4.6); otherwise, reserved (must be 0)

Page Fault

→ triggered on memory translation error : exception 14

↳ page table walk encounters unallocated page table

↳ page not mapped (page table entry's present bit not set)

↳ mismatch permission (write to read only page, kernel page accessed in user mode)

→ need to identify valid vs. invalid page faults

demand paging { stack growth, cow fork, memory mapped files

access to unallocated user address range, permission violation

How does the kernel determine what kind of fault it is?

→ kernel has bookkeeping structures (machine independent) that track information about each page.

* Machine independent vs. machine dependent page table

(vspace, vregion, vpage_info) (x86-64 pt)

info for page fault

info for address translation

```
42  struct vregion {
43    enum vr_direction dir;  // direction of growth
44    uint64_t va_base;       // base of the region
45    uint64_t size;          // size of region in bytes
46    struct vpi_page *pages;  // pointer to array of page_infos
47  };
48
49  struct vspace {
50    struct vregion regions[NREGIONS]; // the regions for a process' virtual space
51    pml4e_t* pgtbl;                   // process' page table
52  };
```

first level page table

info about each page

```
17  struct vpage_info {
18    short used;     // whether the page is in use
19    uint64_t ppn;   // physical page number
20    short present;  // whether the page is in physical memory
21    short writable; // does the page have write permissions
22    // user defined fields
23
24  };
```