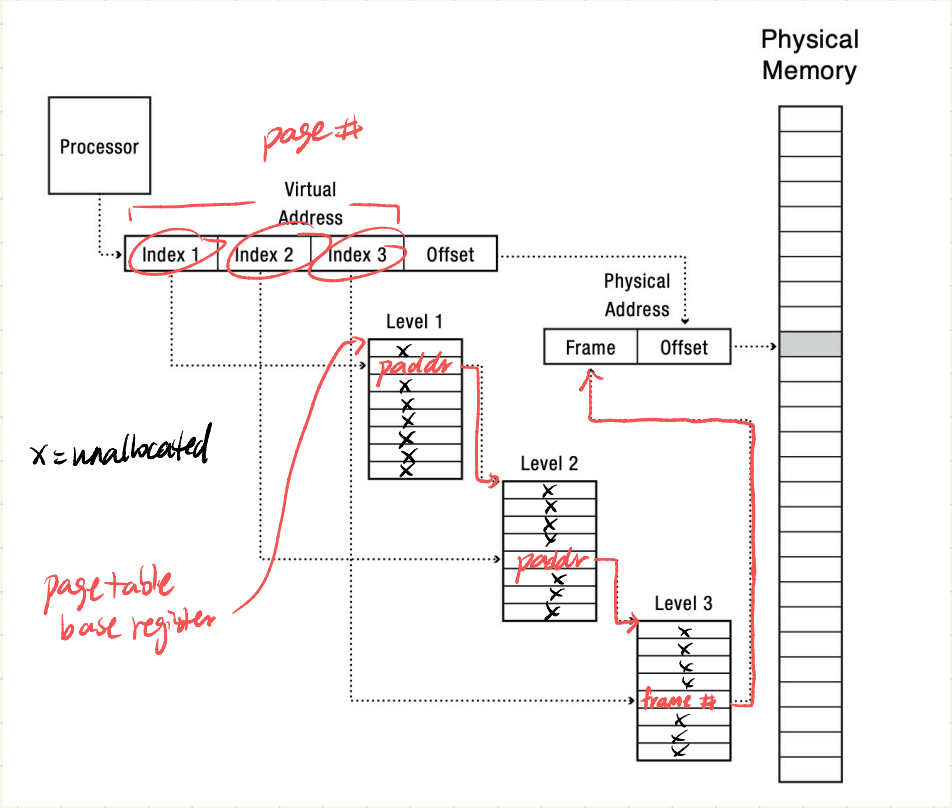
4/26/24

Multilevel Paging



→ each last level page table is a shard of the single array page table

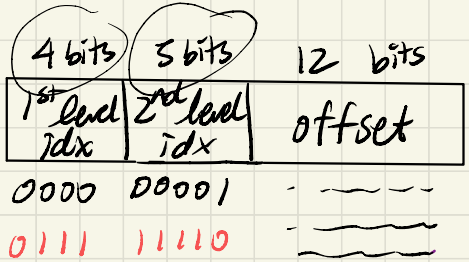
→ indirection takes up more space if most pages are used.

⊠ TLB even more important!

21 bit virtual address

[0x1000 - 0x1fff] code

[0xfeaa - 0xfeffff] stack

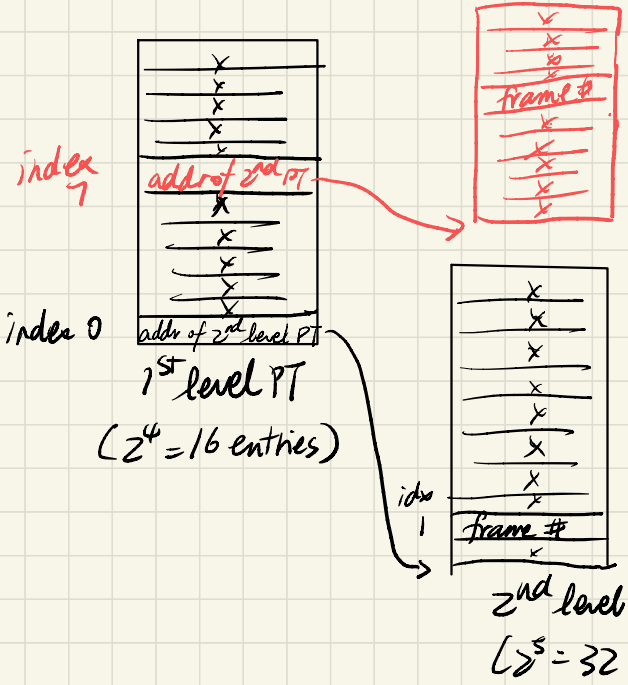


4KB page size

9 bits for page #

2^9 = 512 pages max

Single level PT always allocate 512 entries



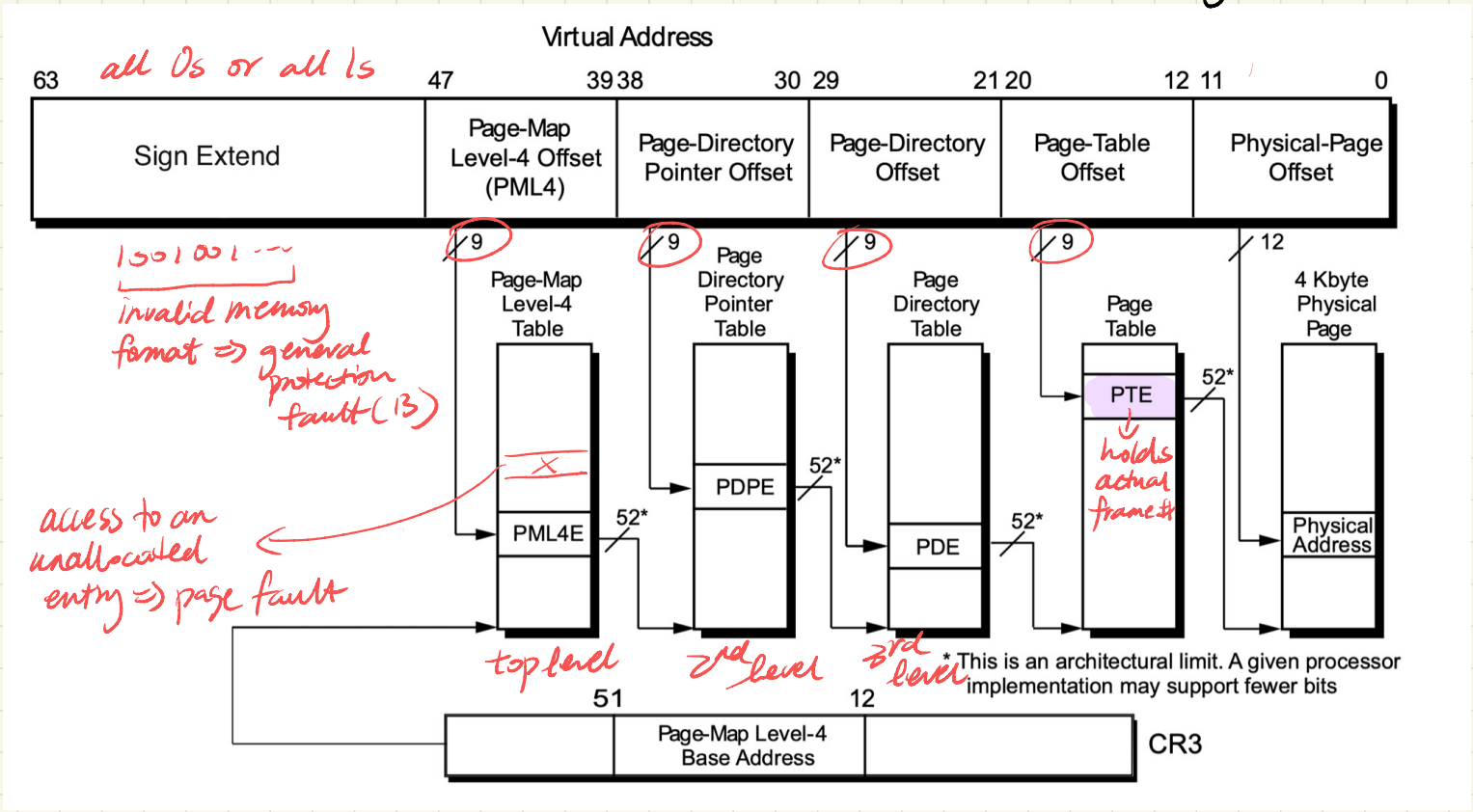
2 page tables needed to map just the codepage (one 1st level + one 2nd level)

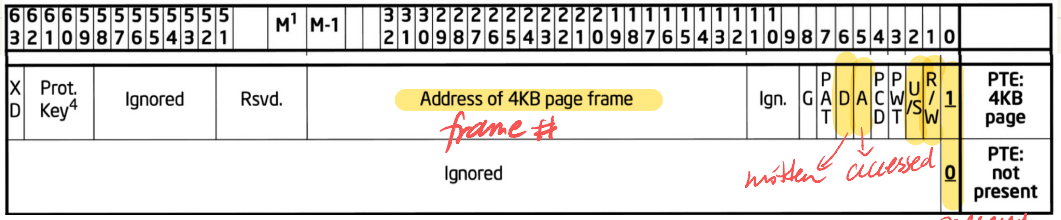
16 + 32 = 48 entries

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

16 + 32×2 = 80 entries

x86-64 Page Table : each level of page table has 2^9 entries & lives in a page, 8 byte entry





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 (page table walk or TLB)

→ 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)

→ types of page faults

→ demand paging (valid)

- → stack growth
- → heap growth
- → memory mapped files

allocate a frame, clear out the contents before mapping it.

load from disk

→ Permission Mismatch

- → cow access
- → actual violation (user access kernel addr) write to read only page.

allocate new frame, update PT & flush TLB ☆

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

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

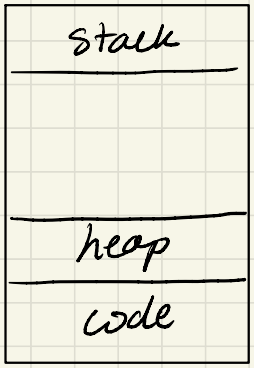
7K VM structures

machine dependent

machine independent

```
struct vspace {
  struct vregion regions[NREGIONS]; // the regions for a process' virtual space
  pml4e_t* pgtbl;                   // process' page table
};
```

vspace_update() updates the machine dependent PT based on machine independent vregion content

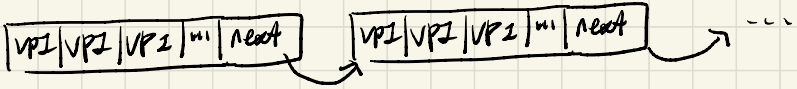


process VAS

```
struct vregion {
  enum vr_direction dir;  // direction of growth
  uint64_t va_base;       // base of the region
  uint64_t size;          // size of region in bytes
  struct vpi_page *pages;  // pointer to array of page_infos
};
```

```
struct vpi_page {
  struct vpage_info infos[VPIPPAGE];  // info struct for the given page
  struct vpi_page *next;              // the next page
};
```

linked list of vpage info array



xk VM structures

```
struct vpage_info {
  short used;     // whether the page is in use
  uint64_t ppn;   // physical page number
  short present;  // whether the page is in physical memory
  short writable; // does the page have write permissions
  // user defined fields

};
```

per page metadata