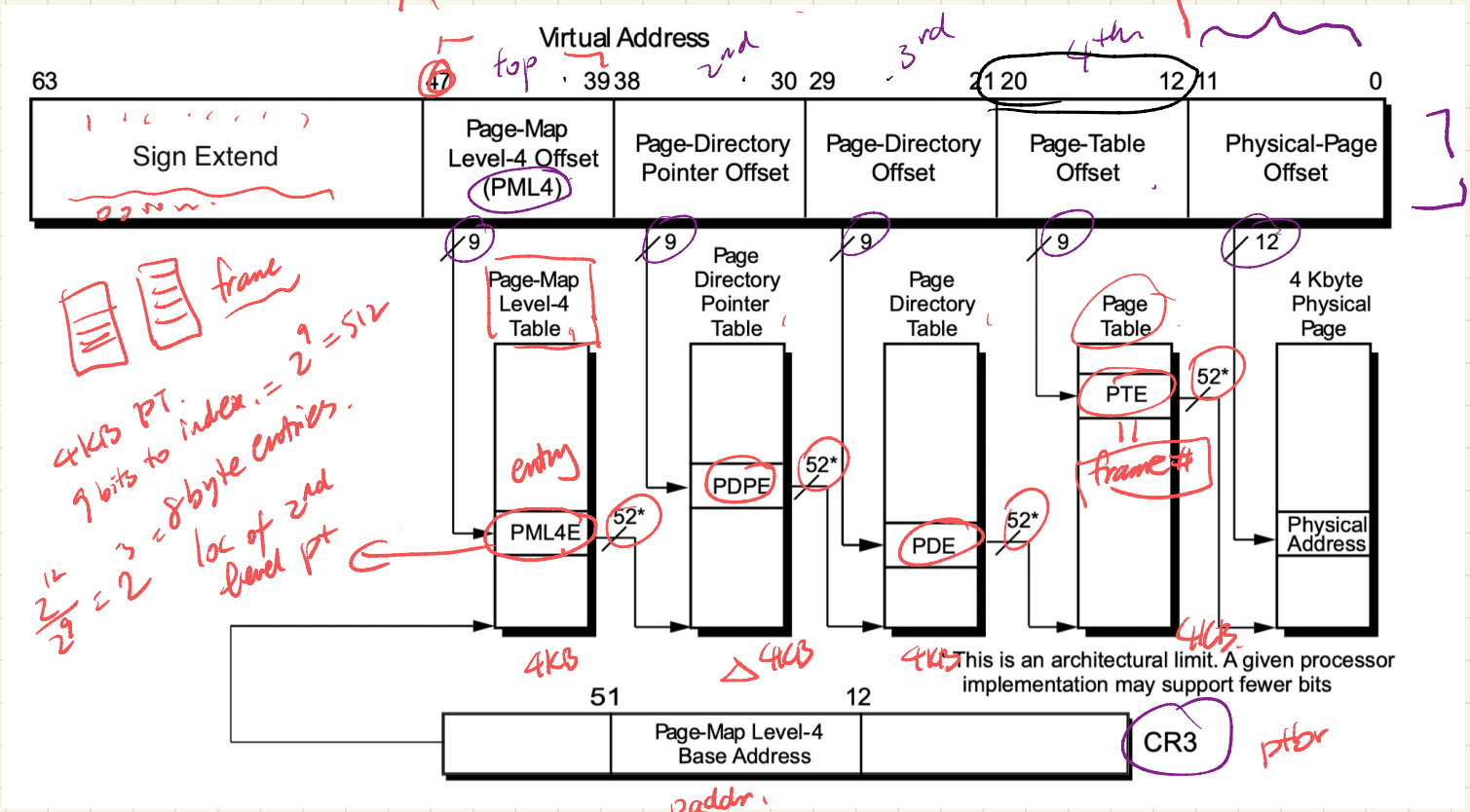
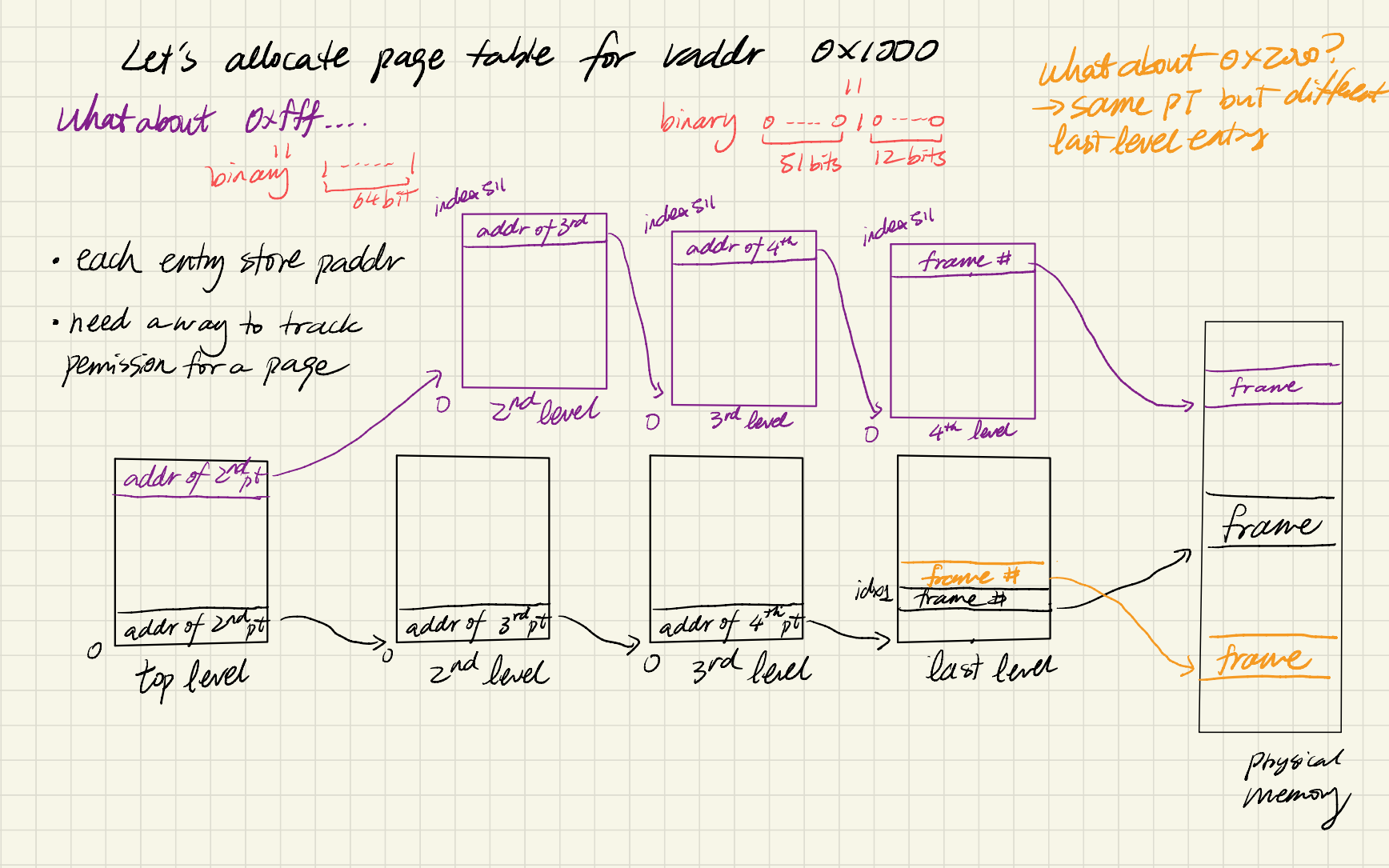
11/4 Endless Paging

Paging review

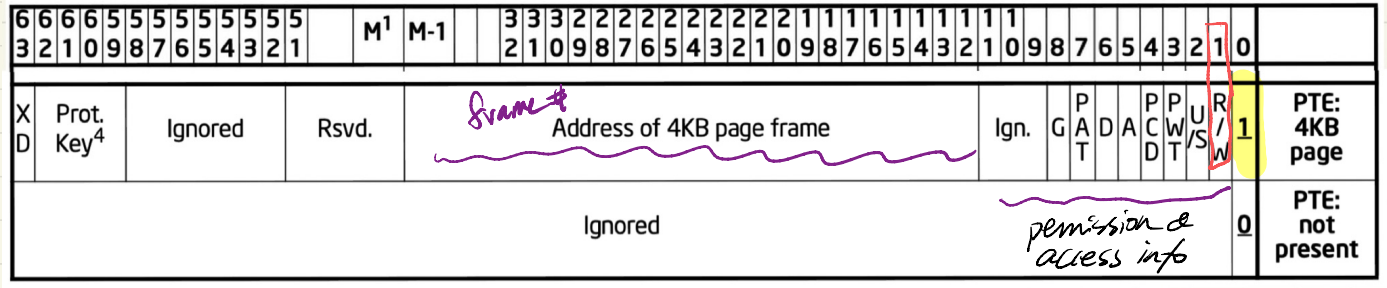
- VM => pages, PM => frames
- single level vs. multilevel page table
- TLB + HW page table walk
  -> architecture spec defines page table format

X86-64 Page Table Format




Let's allocate page table for vaddr 0x1000
what about 0xfff....
binary
64 bit
binary 0 ---- 0 1 0 ---- 0
51 bits
12 bits
what about 0x2000?
→ same PT but different last level entry
• each entry store paddr
• need a way to track permission for a page
index $1l
addr of 3rd
addr of 4th
frame #
addr of 2nd pt
addr of 3rd pt
addr of 4th pt
index $1
frame #
frame #
top level
2nd level
3rd level
last level
2nd level
3rd level
4th level
frame
frame
frame
Physical memory

Format of the 8 byte page table entry (last level)



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)

What happens when hw encounters error during a page table walk?

→ invalid page table entry (present bit == 0)

→ permission mismatch (write to a read only page)

☆ page fault (exception 14)

→ kernel handles page fault (see trap.c)

→ How?

→ cases

- valid
  - ① stack, heap growth (allocate a new frame & create the mapping) → initialize to 0s (why?)
  - ② cow write access (allocate a new frame, copy content, update mapping)
  - ③ memory mapped file, swapped pages
- invalid = bad address, actual permission mismatch (user access kernel addr) (terminate the process)

Remember TLB?

☆ whenever you are changing the translation mapping, you need to flush TLB to get rid of cached, stale permission

xk: vspaceinstall

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 loc.

* vspace invalidate → generates a new x86-64 page table from vspace data.

```
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  };
```

information about each page.