Paging review

- VM $\Rightarrow$ pages, PM $\Rightarrow$ frames
- Single level vs. multilevel page table
- TLB + HW page table walk
  $\Rightarrow$ architecture spec defines page table format
X86-64 Page Table Format

Virtual Address

<table>
<thead>
<tr>
<th>Sign Extend</th>
<th>Page-Map Level-4 Offset (PML4)</th>
<th>Page-Directory Pointer Offset</th>
<th>Page-Directory Offset</th>
<th>Page-Table Offset</th>
<th>Physical-Page Offset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>63</td>
<td>47</td>
<td>39 38</td>
<td>30 29</td>
<td>12 11</td>
</tr>
</tbody>
</table>

Page-Map Level-4 Table

Page Directory Pointer Table

Page Directory Table

Page Table

4 Kbyte Physical Page

This is an architectural limit. A given processor implementation may support fewer bits.

4K8

Page-Map Level-4 Base Address

CR3

This is an architectural limit. A given processor implementation may support fewer bits.
Let's allocate page table for vaddr 0x1000

What about 0xffff ....

binary 0 ---- 010 ---- 0
81 bits 12 bits

let's see about ox0....

→ same PT but different last level entry

• each entry store paddr
• need a way to track permission for a page

<table>
<thead>
<tr>
<th>frame #</th>
<th>frame #</th>
</tr>
</thead>
</table>

level 0

level 2

level 3

Physical Memory
### Format of the 8-byte page table entry (last level)

<table>
<thead>
<tr>
<th>Bit Position(s)</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 (P)</td>
<td>Present; must be 1 to map a 4-KByte page</td>
</tr>
<tr>
<td>1 (R/W)</td>
<td>Read/write; if 0, writes may not be allowed to the 4-KByte page referenced by this entry (see Section 4.6)</td>
</tr>
<tr>
<td>2 (US)</td>
<td>User/supervisor; if 0, user-mode accesses are not allowed to the 4-KByte page referenced by this entry (see Section 4.6)</td>
</tr>
<tr>
<td>3 (PWT)</td>
<td>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)</td>
</tr>
<tr>
<td>4 (PCD)</td>
<td>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)</td>
</tr>
<tr>
<td>5 (A)</td>
<td>Accessed; indicates whether software has accessed the 4-KByte page referenced by this entry (see Section 4.8)</td>
</tr>
<tr>
<td>6 (D)</td>
<td>Dirty; indicates whether software has written to the 4-KByte page referenced by this entry (see Section 4.8)</td>
</tr>
<tr>
<td>7 (PAT)</td>
<td>Indirectly determines the memory type used to access the 4-KByte page referenced by this entry (see Section 4.9.2)</td>
</tr>
<tr>
<td>8 (G)</td>
<td>Global; if CR4.PGE = 1, determines whether the translation is global (see Section 4.10); ignored otherwise</td>
</tr>
<tr>
<td>11:9</td>
<td>Ignored</td>
</tr>
<tr>
<td>(M-1):12</td>
<td>Physical address of the 4-KByte page referenced by this entry</td>
</tr>
<tr>
<td>51:48</td>
<td>Reserved (must be 0)</td>
</tr>
<tr>
<td>58:52</td>
<td>Ignored</td>
</tr>
<tr>
<td>62:59</td>
<td>Protection key; if CR4.PKE = 1, determines the protection key of the page (see Section 4.6.2); ignored otherwise</td>
</tr>
<tr>
<td>63 (XD)</td>
<td>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)</td>
</tr>
</tbody>
</table>
What happens when hw encounters error during a page table walk?

→ invalid page table entry (present bit = 0)
→ permission mismatch (write to a readonly 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)
  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 add)
(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 (x86-64 pt):

- info for address translation

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

```
struct vpage_info {
    information about each page.
    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
};
```