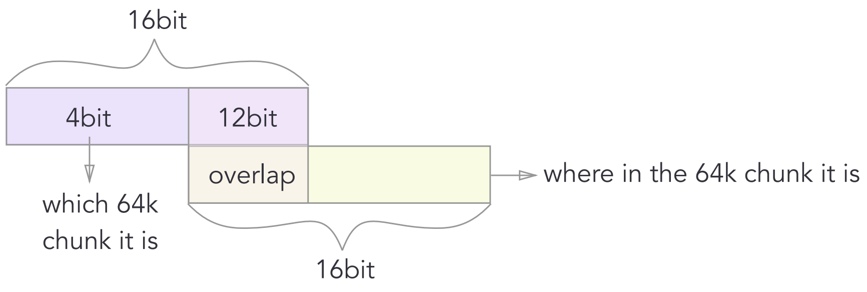
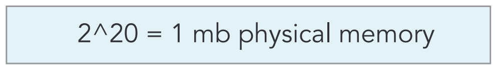
PAGING.md

16-bit

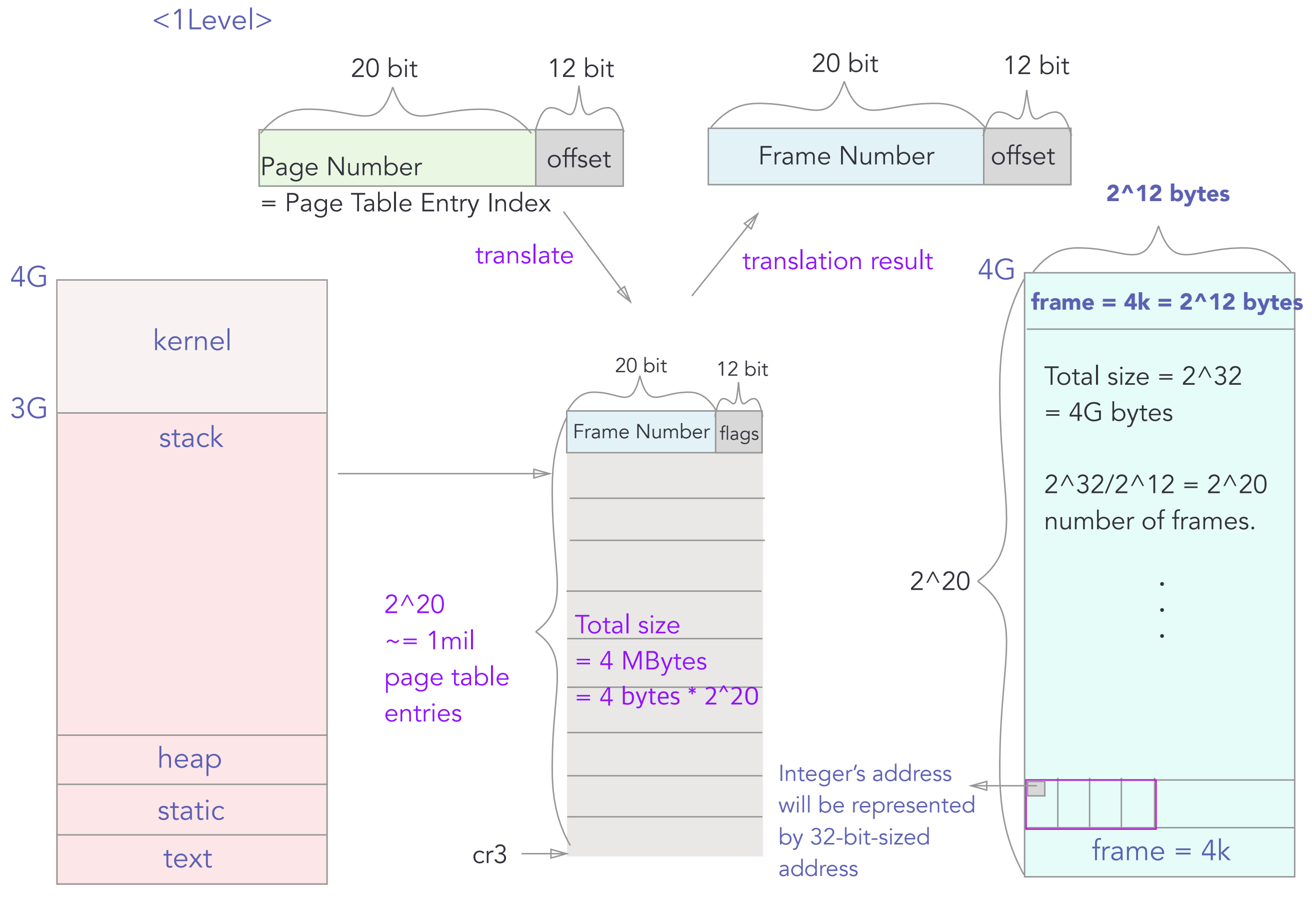




- When the first processor came out, it was 16 bit-cpu. Each word, register and everything was 16 bits = 2 bytes.
- With 16 bits = you can address 2^16 ~= 64k addresses. Extremely small. Back then, full capacity of motherboard is 1mega bytes = 2^20 = 1,000k. 4 bit short to address all the memory. So they divided 1 mb memory into 64k chunks.
- NOTE: the physical memory doesn't have to be 2^16. It's just the maximum possible assuming nothing extra was done in the meory. So the physical memory can exceed 2^16 or can be less than that.
- In order to address 2^20 address, two 16 bits were used and the 12 bits were overlapped.
- 16-bit segment registers were used to denote the starting point of where your code is. -> So the code can't be more than 2^16 = 64k.

1-LEVEL

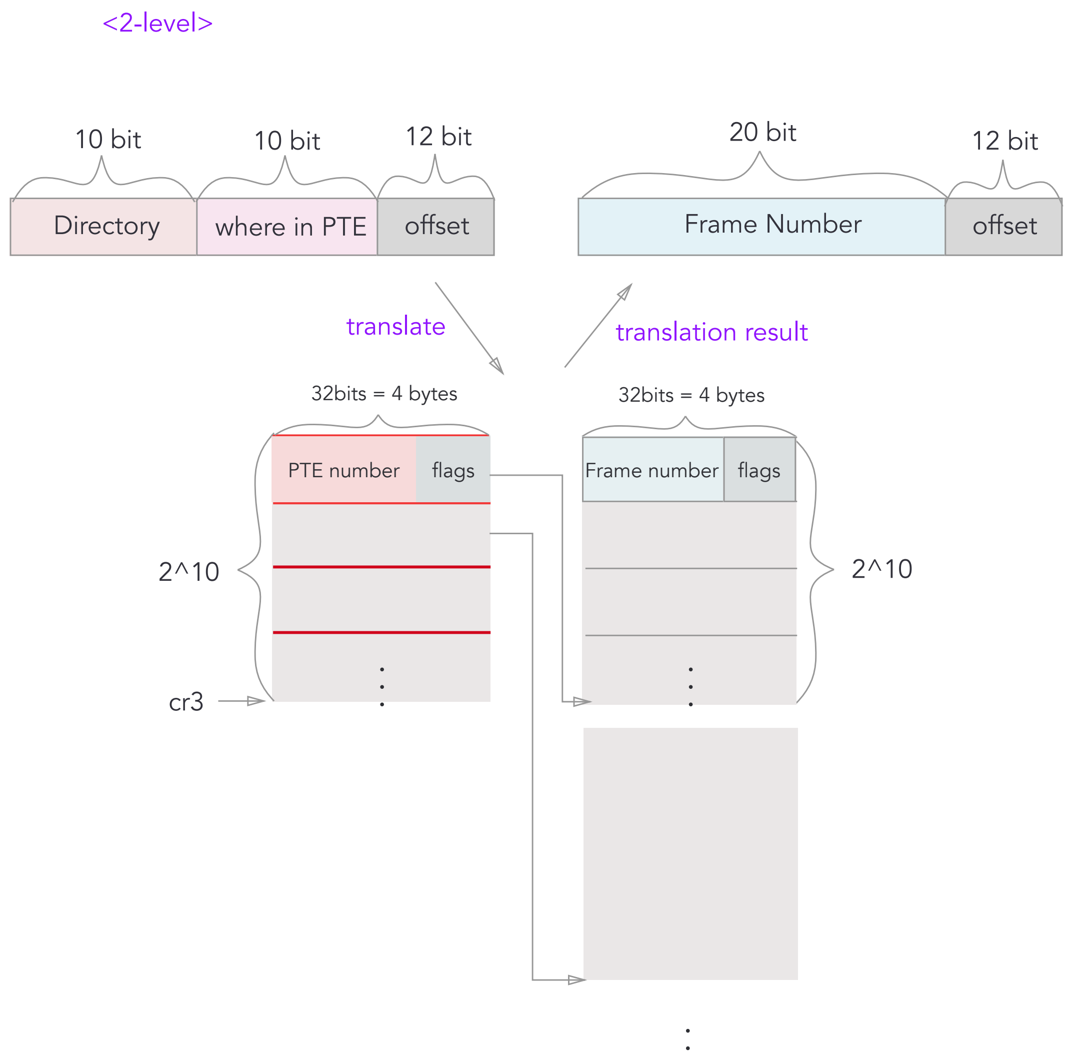
x86-32bit



- Segment registers are no longer needed in 32-bit, so in Linux they are all set to 0 (they are still 16-bit). Overlapping segment for the entire memory.
- offset became a real physical address in 32 bit.
- selector = segment register which remained 16 bit.
- base = starting address of the segment = constant value of 0
- limit = how big it is = constant value of 4g
- perm = whether it is code region.
- PROBLEM: the page table is way too big.
- SOLUTION: 2-level paging

2-LEVEL

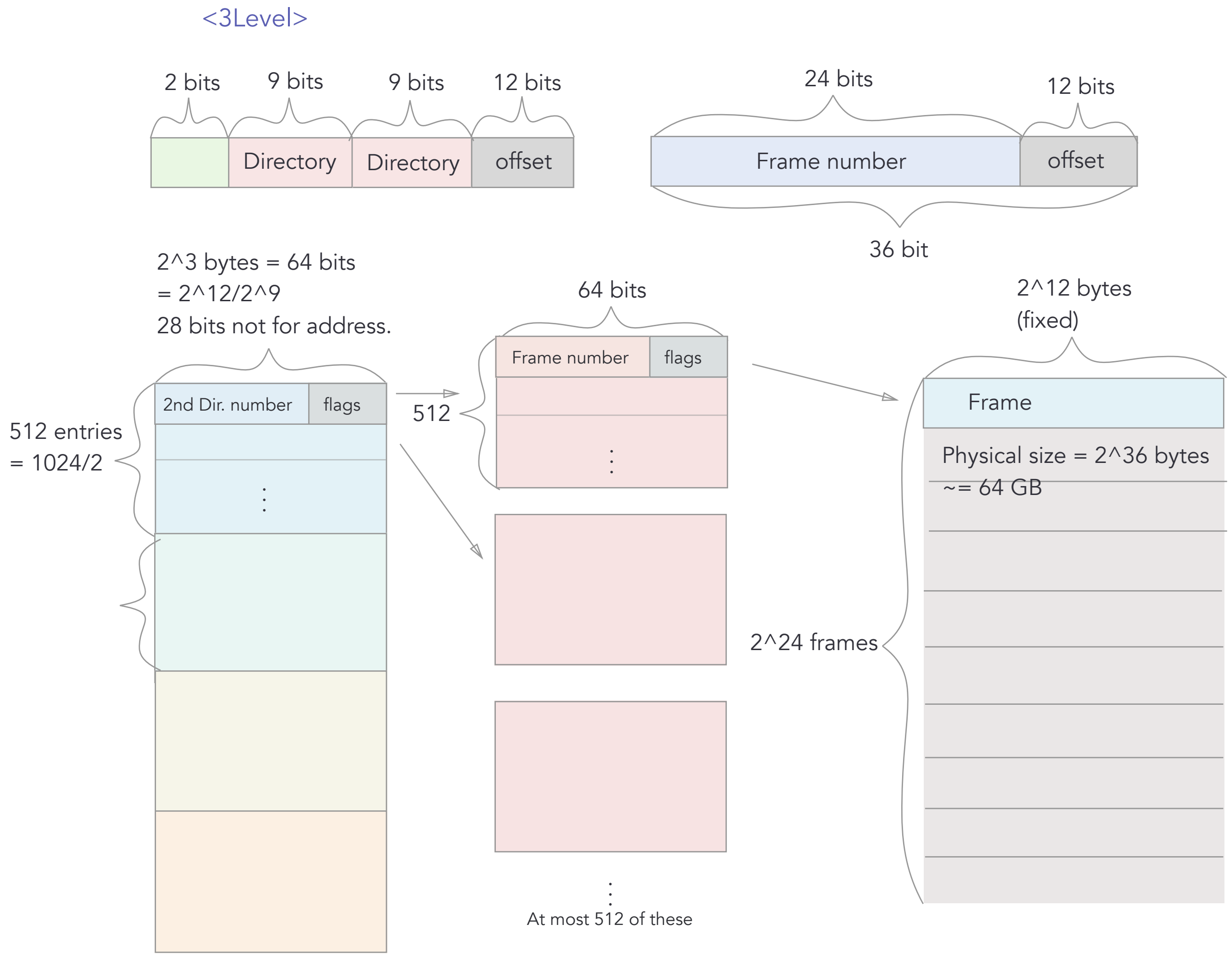
x86-32bit



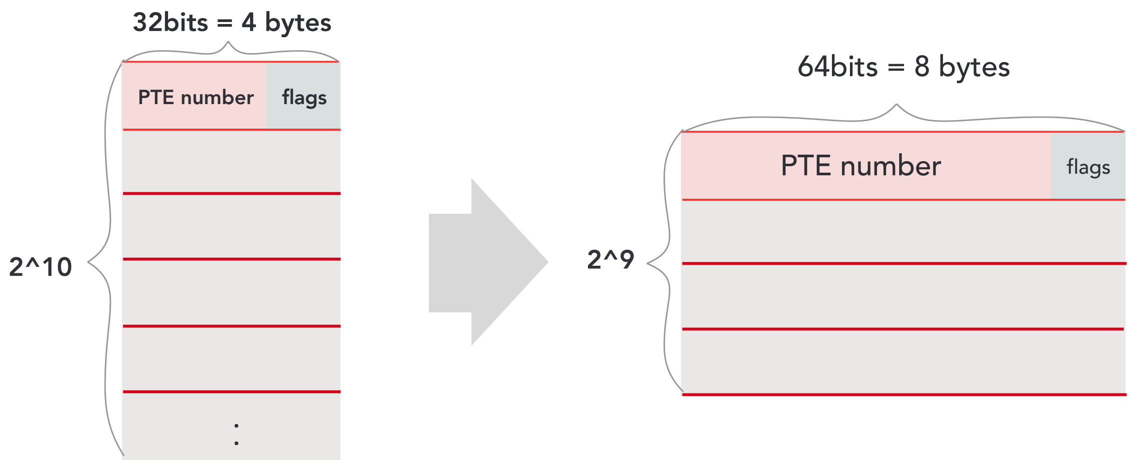
- Directory is already created, but the second level bundle is created on demand.
- Page Table Base Register (PTBR) points to the base of page table. In x86, it's called **cr3**.
- PTBR (cr3) contains the physical address of the bottom of the page table. Or else, the page table location won't be found. NOTE: Everything is physical including the page table
- OS stores PTBR in task struct (PCB), since every process has their own page table.

3-LEVEL

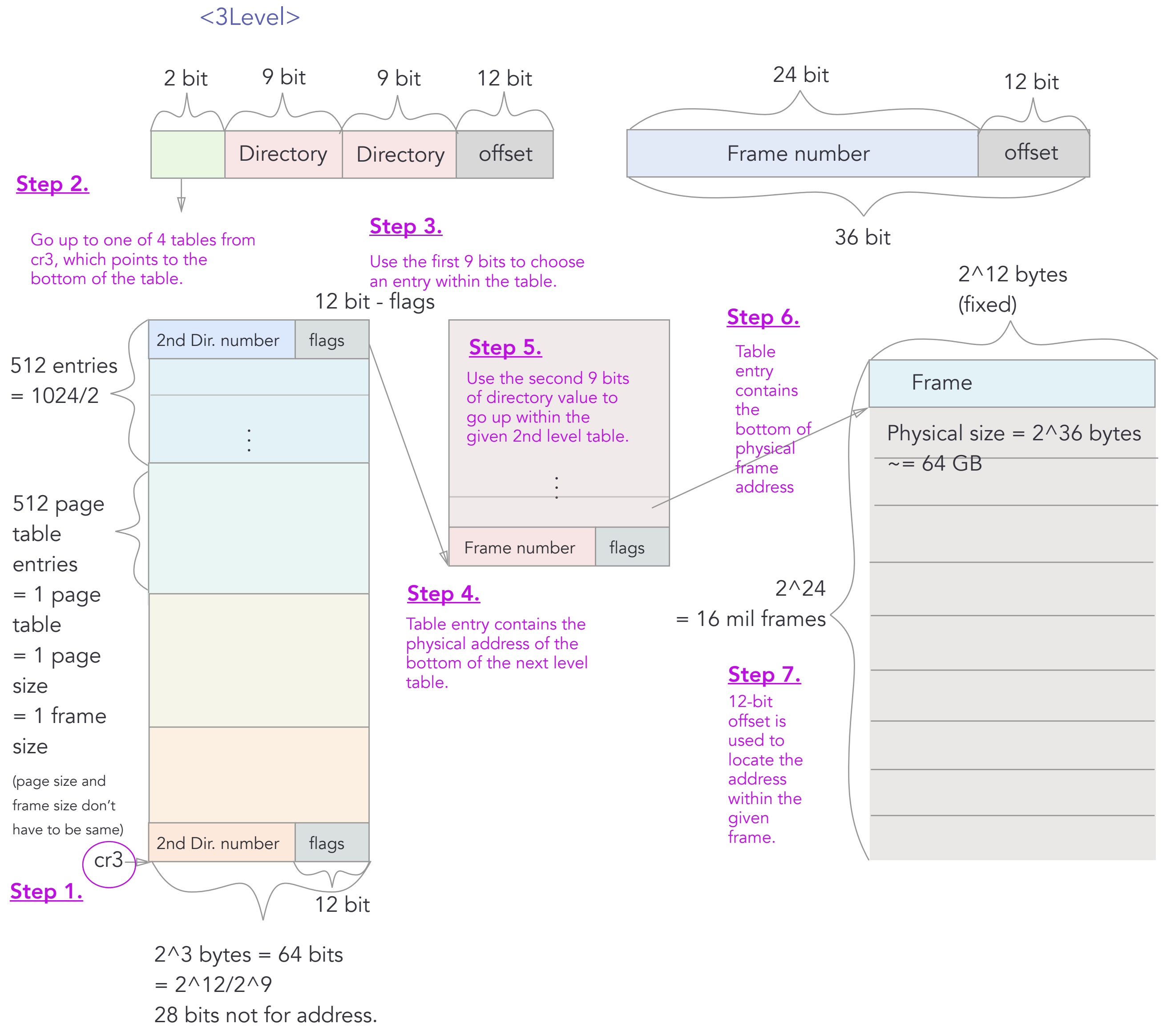
Physical Address Extension (PAE)



- 36 bit physical address
- Increased the number of pins from CPU to memory: 36 bits total
- Internally everything is 32 bits. - virtual address limited to 4k
- But 4 more pins out of CPU, which means 2^32 * 2^4 = 64GB - physical memory - can support physical ram up to 64GBG
- Virtual memory limited but it allows you to learn a lot more processes.
- 12 bit offset is the same since the page size is the same.



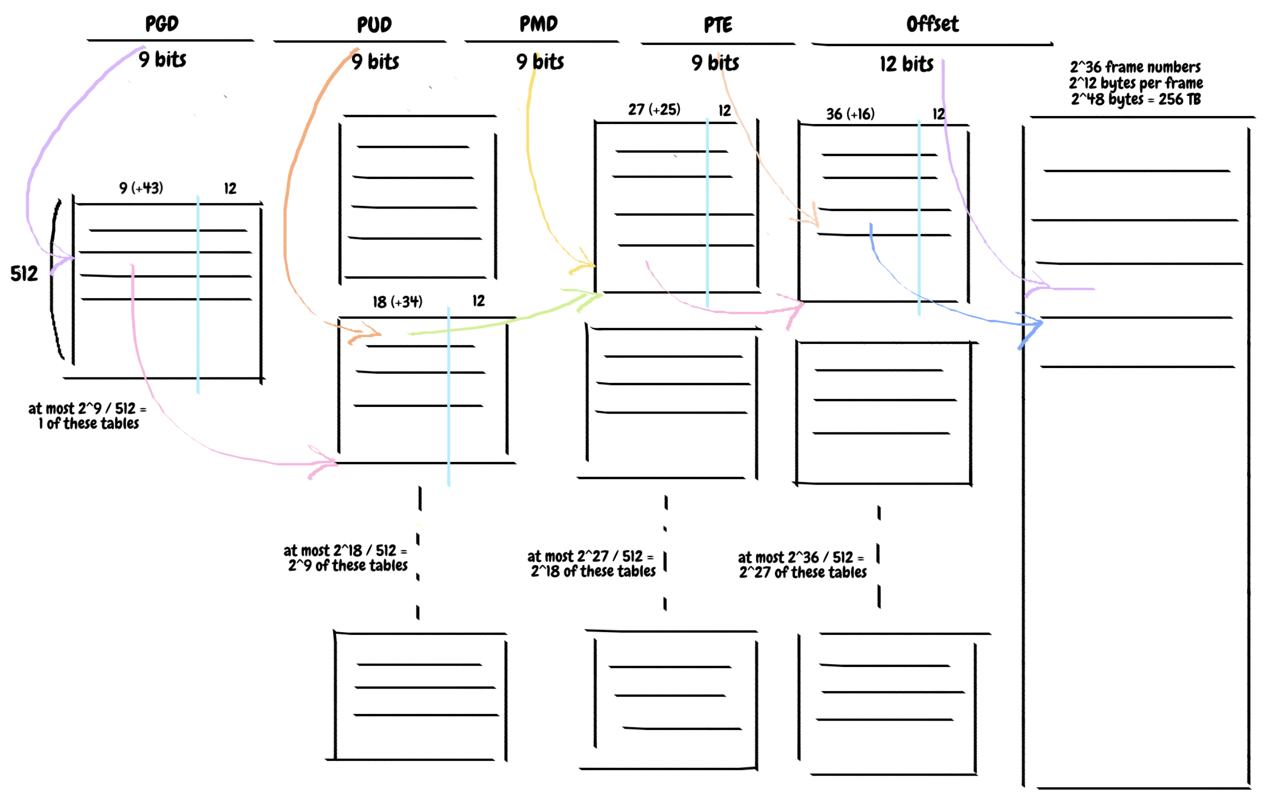
- 24 bit doesn't fit into 32 bit anymore. so each PTE now has to be 64 bits.


<3Level>
2 bit
9 bit
9 bit
12 bit
Directory
Directory
offset
24 bit
12 bit
Frame number
offset
36 bit
Step 2.
Go up to one of 4 tables from cr3, which points to the bottom of the table.
Step 3.
Use the first 9 bits to choose an entry within the table.
12 bit - flags
2^12 bytes (fixed)
Step 6.
Table entry contains the bottom of physical frame address
512 entries = 1024/2
2nd Dir. number
flags
Step 5.
Use the second 9 bits of directory value to go up within the given 2nd level table.
Frame
Physical size = 2^36 bytes ~= 64 GB
512 page table entries = 1 page table = 1 page size = 1 frame size
(page size and frame size don't have to be same)
Frame number
flags
Step 4.
Table entry contains the physical address of the bottom of the next level table.
2^24 = 16 mil frames
Step 7.
12-bit offset is used to locate the address within the given frame.
2nd Dir. number
flags
cr3
Step 1.
12 bit
2^3 bytes = 64 bits = 2^12/2^9
28 bits not for address.

4-LEVEL

x86-64bit

- PGD = page global directory
- PUD = page uppder directory
- PMD = page middle directory
- PTE = pate table entry


PGD
9 bits
PUD
9 bits
PMD
9 bits
PTE
9 bits
Offset
12 bits
2^36 frame numbers
2^12 bytes per frame
2^48 bytes = 256 TB
9 (+43)
12
512
at most 2^9 / 512 =
1 of these tables
18 (+34)
12
at most 2^18 / 512 =
2^9 of these tables
27 (+25)
12
at most 2^27 / 512 =
2^18 of these tables
36 (+16)
12
at most 2^36 / 512 =
2^27 of these tables