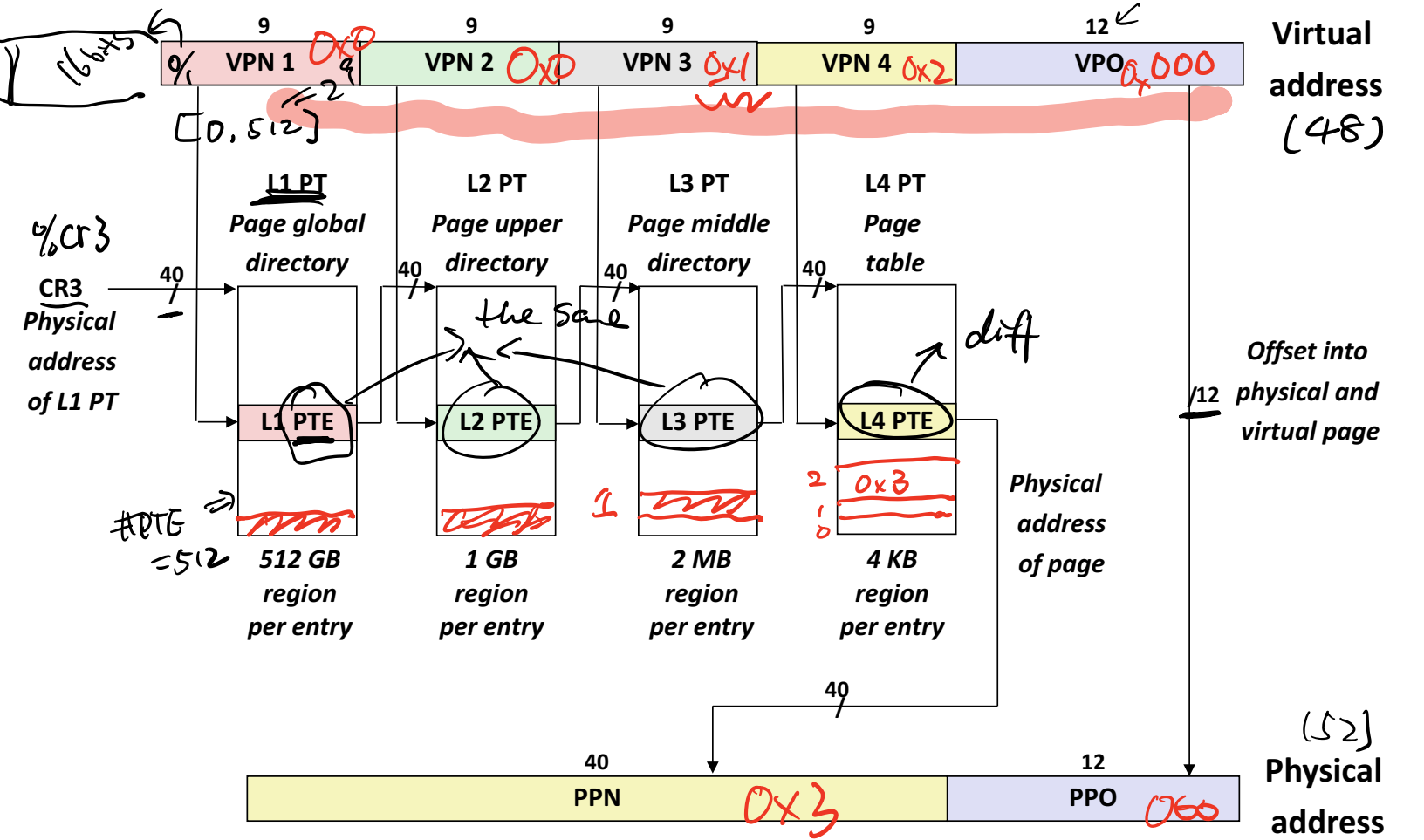
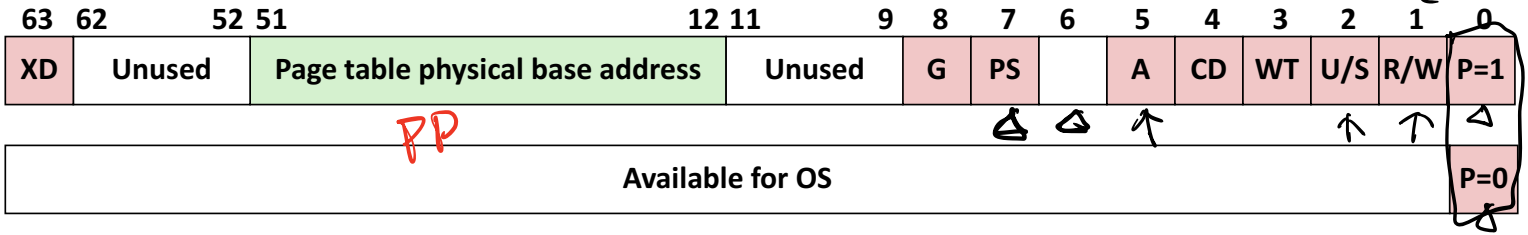
Core i7 Page Table Translation



Review of Symbols

- **Basic Parameters**
  - **N = 2^n** : Number of addresses in virtual address space
  - **M = 2^m** : Number of addresses in physical address space
  - **P = 2^p** : Page size (bytes)
- **Components of the virtual address (VA)**
  - **TLBI**: TLB index
  - **TLBT**: TLB tag
  - **VPO**: Virtual page offset
  - **VPN**: Virtual page number
- **Components of the physical address (PA)**
  - **PPO**: Physical page offset (same as VPO)
  - **PPN:** Physical page number
  - **CO**: Byte offset within cache line
  - **CI:** Cache index
  - **CT**: Cache tag

Core i7 Level 1-3 Page Table Entries



Each entry references a 4K child page table. Significant fields:

P: Child page table present in physical memory (1) or not (0).

R/W: Read-only or read-write access access permission for all reachable pages.

U/S: user or supervisor (kernel) mode access permission for all reachable pages.

WT: Write-through or write-back cache policy for the child page table.

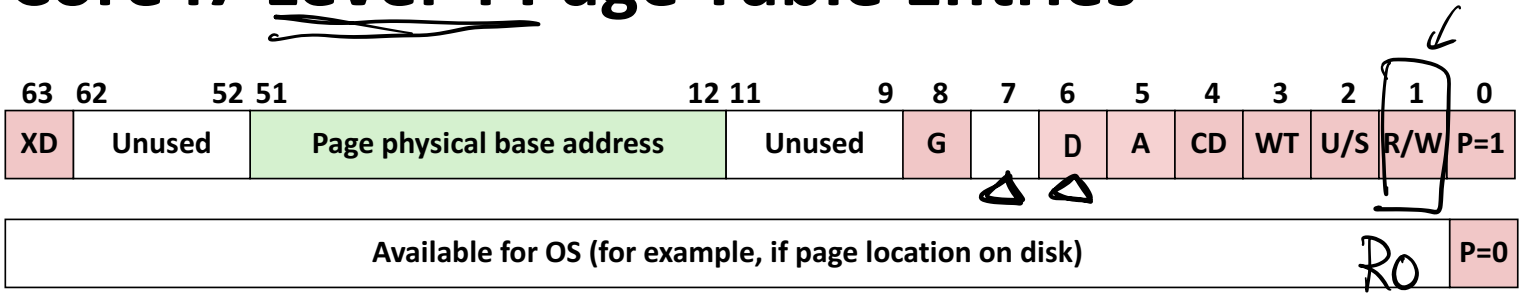
A: Reference bit (set by MMU on reads and writes, cleared by software).

PS: Page size: if bit set, we have 2 MB or 1 GB pages (bit can be set in Level 2 and 3 PTEs only).

Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

XD: Disable or enable instruction fetches from all pages reachable from this PTE.

Core i7 Level 4 Page Table Entries



Each entry references a 4K child page. Significant fields:

P: Child page is present in memory (1) or not (0)

R/W: Read-only or read-write access permission for this page

U/S: User or supervisor mode access

WT: Write-through or write-back cache policy for this page

A: Reference bit (set by MMU on reads and writes, cleared by software)

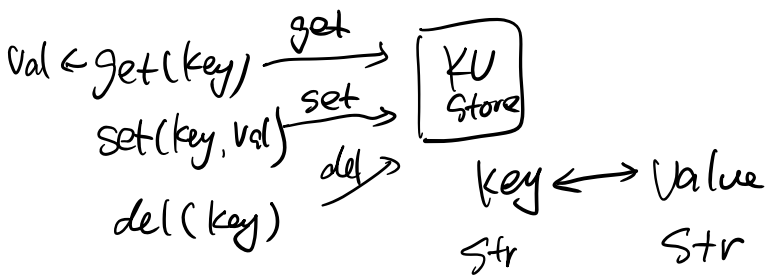
D: Dirty bit (set by MMU on writes, cleared by software)

Page physical base address: 40 most significant bits of physical page address (forces pages to be 4KB aligned)

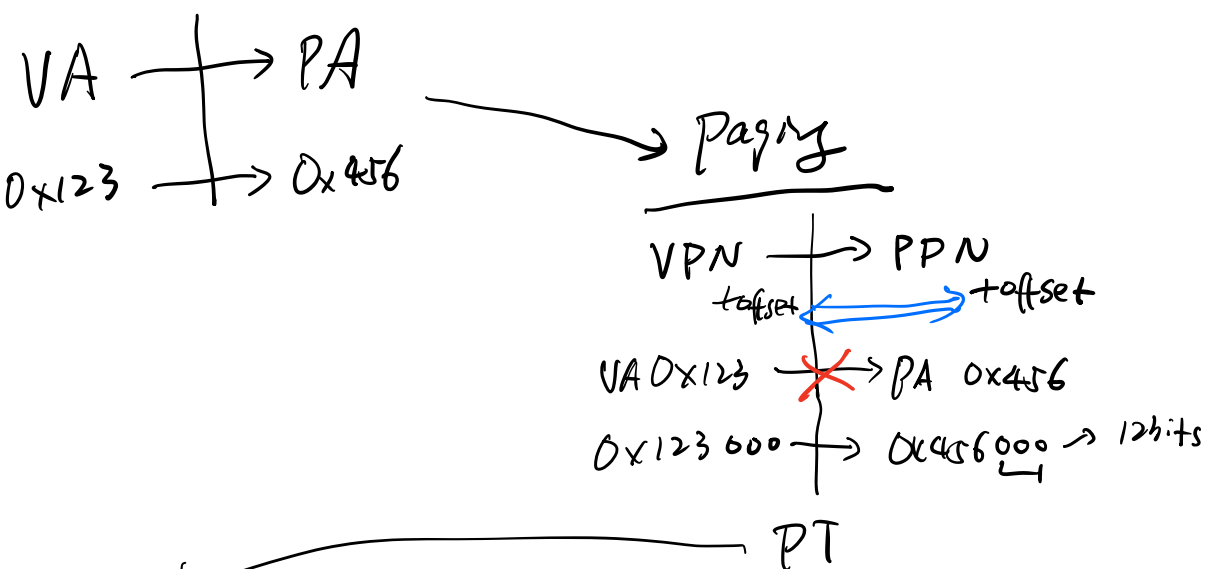
XD: Disable or enable instruction fetches from this page.

1. Last time
2. x86-64: addresses
   - virtual
   - physical
3. x86-64: page table structures
4. Practice

Lab3. Concurrent KV store

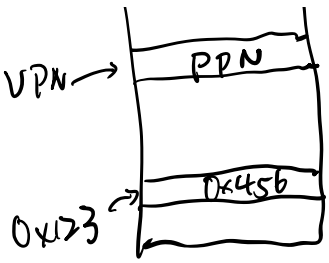


Virtual memory



Naive proposal

Array



Array[0x123] = 0x456

Size:

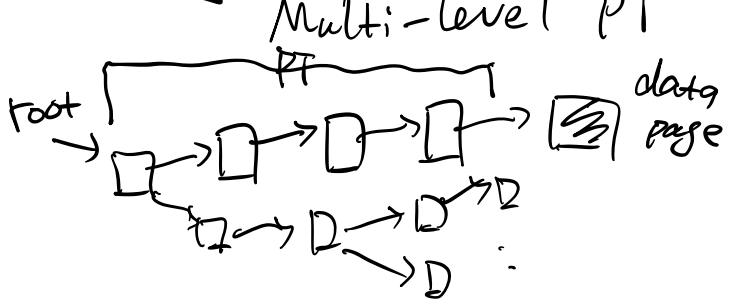
len(Array) = #VPN

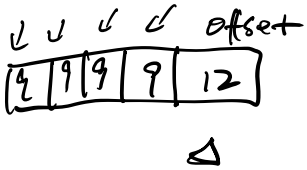
$= 2^{36} \times 8B = 2^{39} B$

$= 512GB$

48bit ~ 12bit

= 36 bit



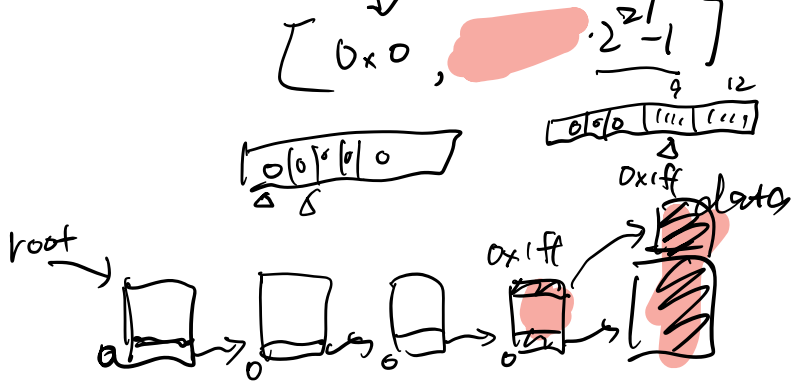


Example: x86-64 VA 48bit

OS maps [0, 2MB-1] ⇒ PM

Q: how many pages in PT?

[0x0, $2^{27}-1$]



1 + 1 + 1 + 1 = 4

Ex: 2GB, how many pages in PT?

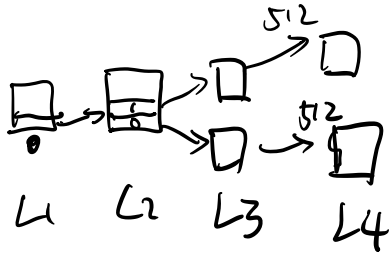
512 WHY? ⇒ HW $\frac{4KB}{?\ 8B} = 512$

144 516

① [0, 2GB − 1]

$2^{31}-1$ → 31 bit — 48bit

0060 1111 1111



② $\frac{2GB}{4KB} = \frac{1}{2}\cdot 2^{20}$

$= 2^{19}$ pages

#L4 PT $= \frac{2^{19}}{2^9}$

$= 2^{10}$

$= 1024$ L4

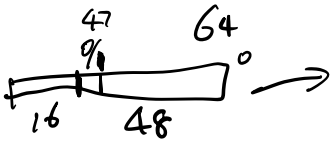
$1 + 1 + 2 + \frac{2\times 512}{1024} = 1028$ ⇒

#L3 PT $= \frac{1024}{512}$

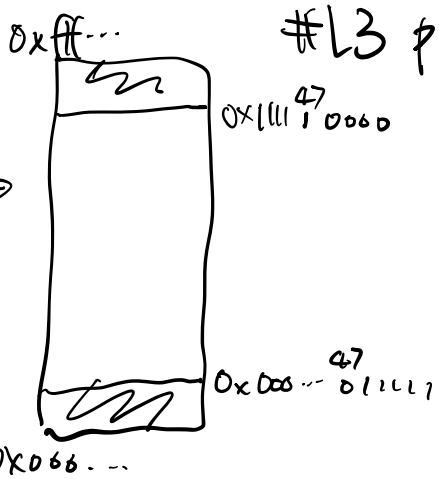
$= 2$

X86-64.

• VA:



↳ all 0s
all 1s



$2^{48} = 2^8 TB = 256 TB$

∴ 5-level paging

- PA: 52bit

  Why ⇒ HW

  4-level ⇒ 5-level

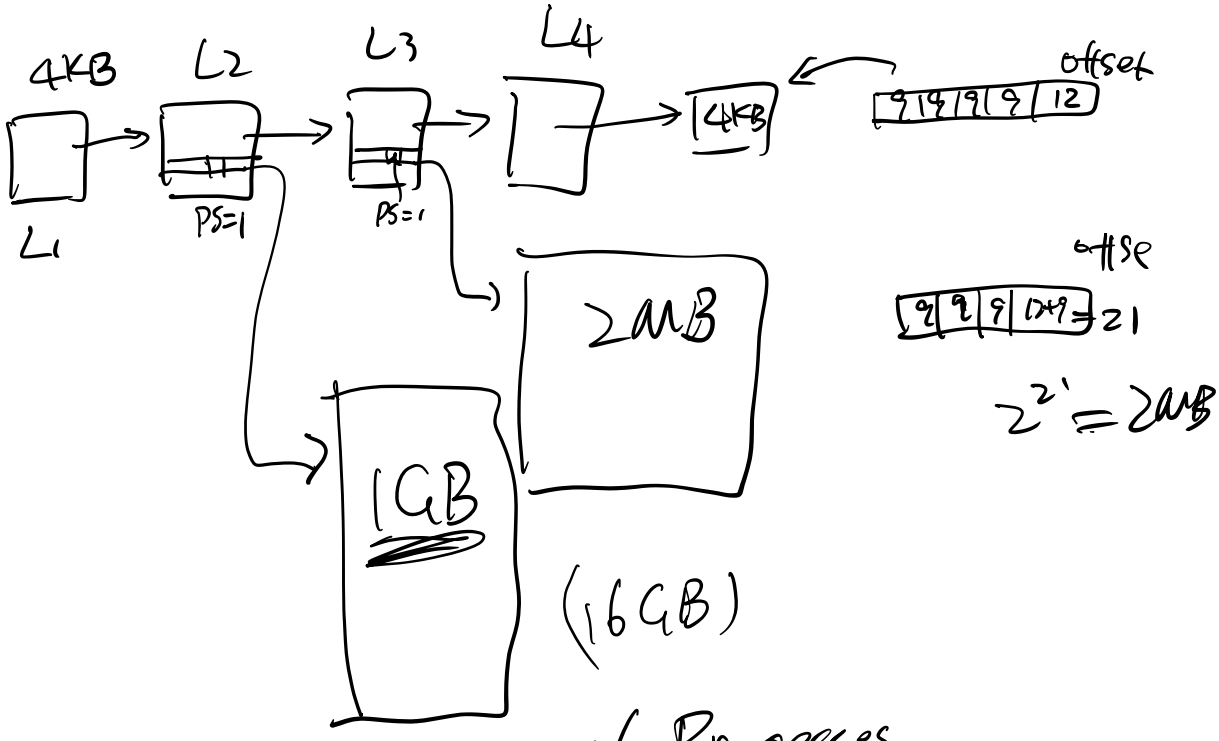
  48 bits ⇒ 57 bits

  $2^{52} = 4PB$ (16GB)

  $2^{57} = 128 PB$

- VA → PA

  (48) ? (52)



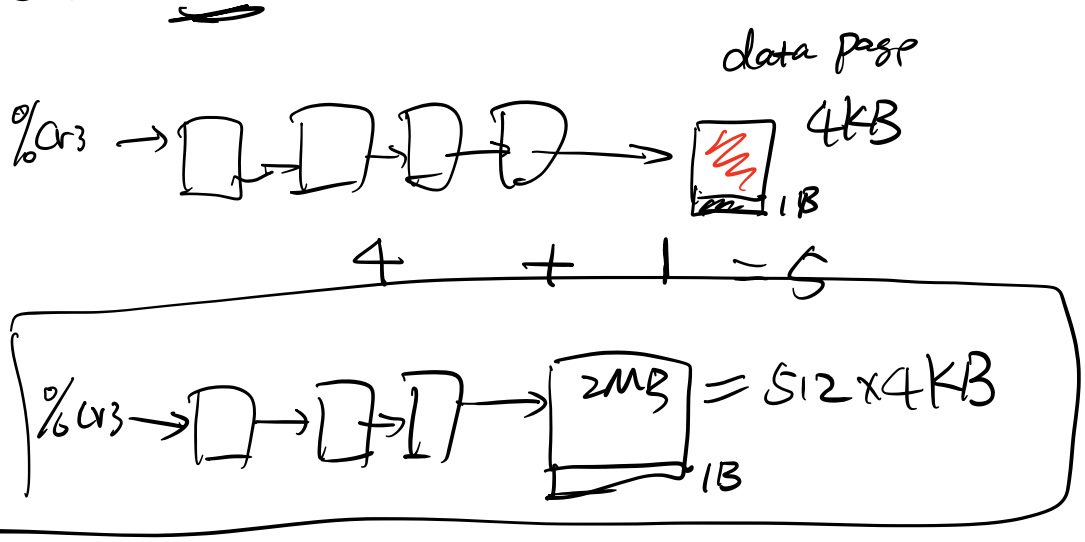
Q: x86-64

OS.

program allocate Memory min

how many pages in memory?

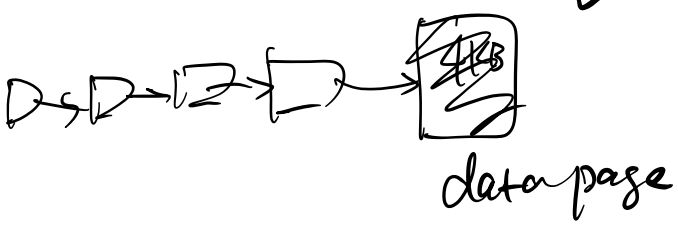
- allocate 1B :



- alloc 2^{12} B ?

"4KB

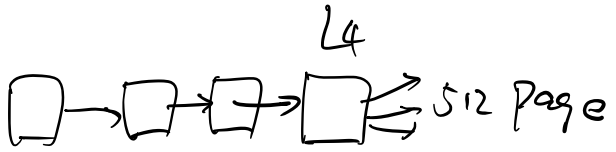
5, 6, 17



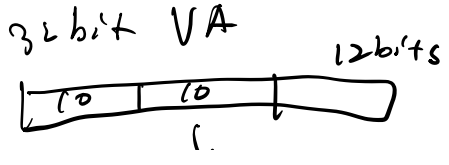
- alloc $2^{9} \times 2^{12}$ Bytes.

515, 516,

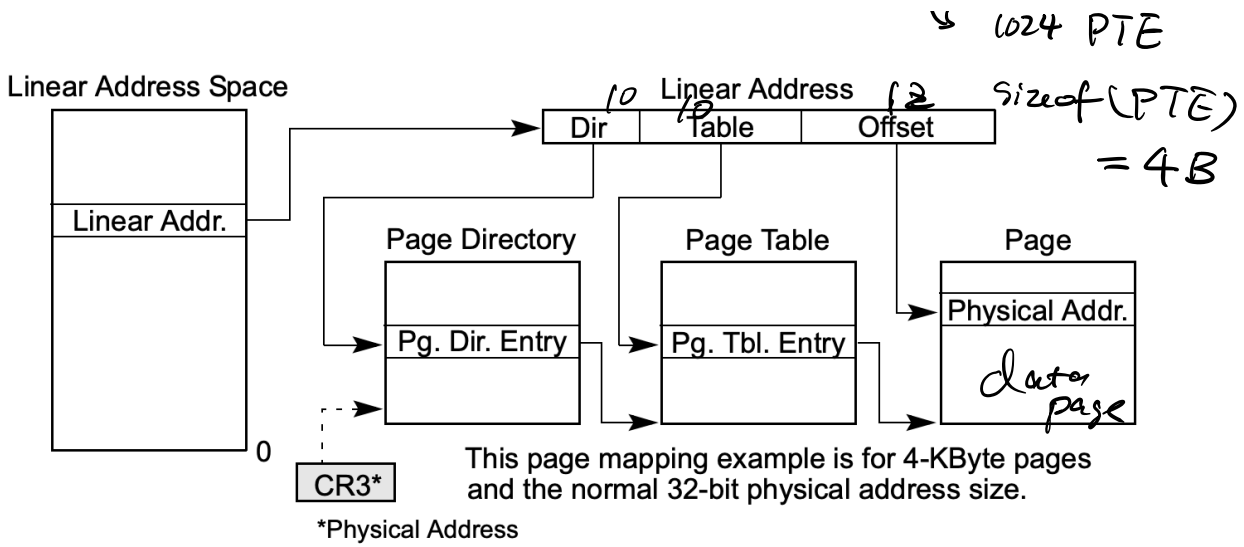
512 x 4KB

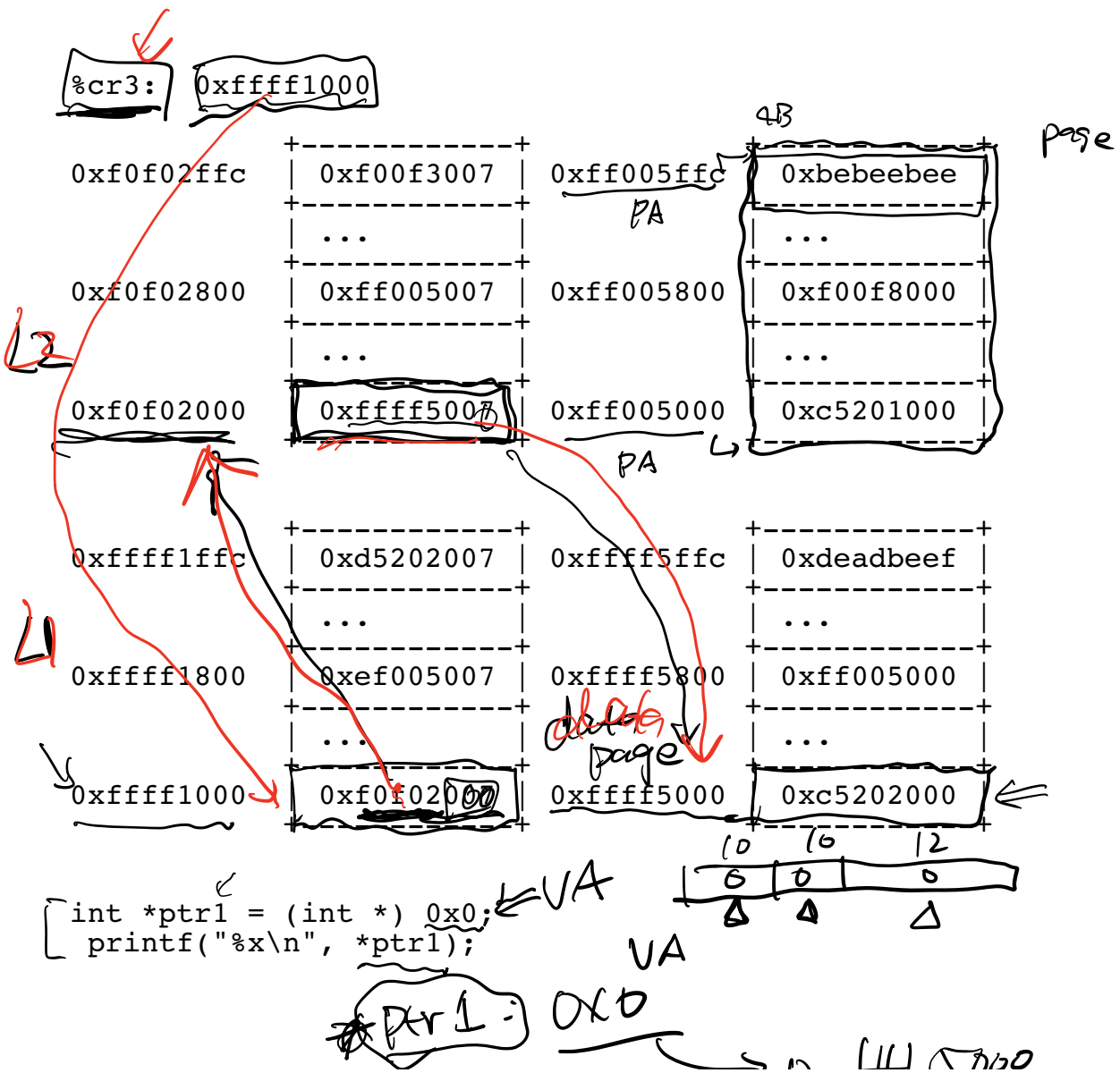


1+1+1+1+512 = 516



x86-32


Linear Address Space
Linear Addr.
0
Linear Address
Dir
Table
Offset
10
12
Page Directory
Pg. Dir. Entry
Page Table
Pg. Tbl. Entry
Page
Physical Addr.
CR3*
*Physical Address
This page mapping example is for 4-KByte pages
and the normal 32-bit physical address size.
1024 PTE
sizeof(PTE)
= 4B


%cr3:
0xffff1000
0xf0f02ffc
0xf00f3007
0xf0f02800
0xff005007
0xf0f02000
0xffff5000
0xffff1ffc
0xd5202007
0xffff1800
0xef005007
0xffff1000
0xf0f02000
0xff005ffc
0xbebeebee
0xff005800
0xf00f8000
0xff005000
0xc5201000
0xffff5ffc
0xdeadbeef
0xffff5800
0xff005000
0xffff5000
0xc5202000
4B
PA
PA
page
10
10
12
int *ptr1 = (int *) 0x0;
printf("%x\n", *ptr1);
VA
VA
ptr1
0x0

print(*ptr1) 0xc5202000