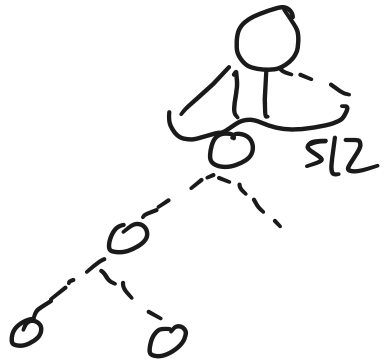


- 1. Last time
- 2. WeensyOS
- 3. Page faults: intro + mechanics
- 4. Page faults and paging: uses
- 5. Page faults' costs
- 6. Page replacement policies



---

### 3. Page faults: intro + mechanics

Concept: illegal virtual memory reference:

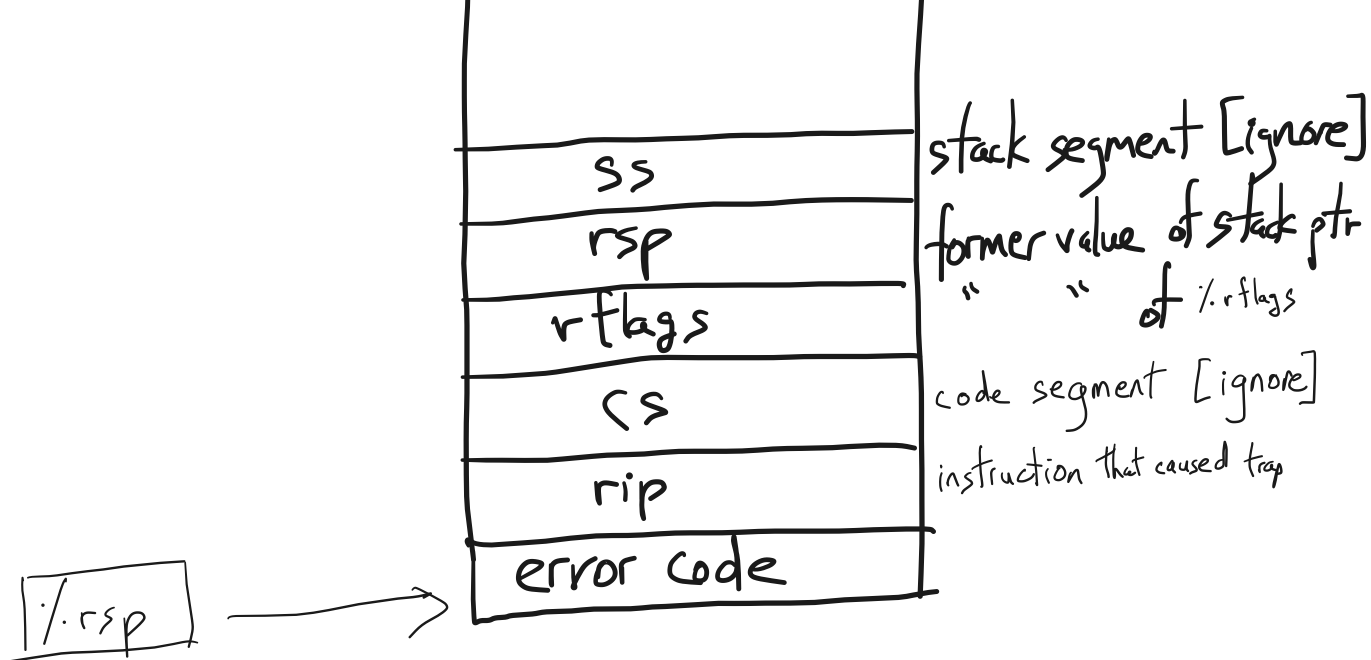
hardware thinks it's illegal (though it might be valid for the process)

OS has to get involved

Mechanics:

- processor constructs trap frame and transfers execution to an interrupt or trap handler





%rip → code to handle the trap

error code: see last pg of handout

[ - - - -  $\frac{I}{D}$  - u/s w/R P ]

%cr2 holds faulting virtual address

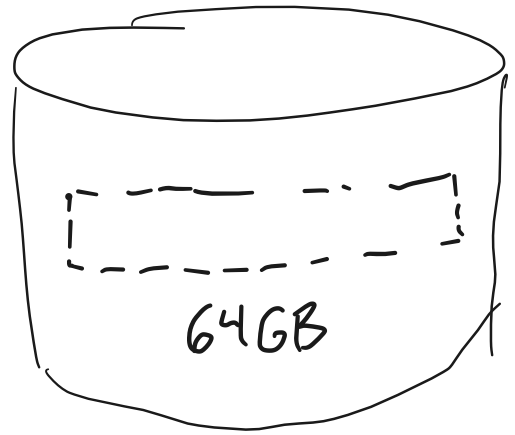
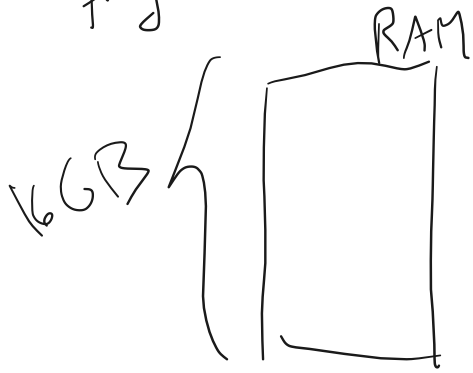
intent: on pg fault, kernel sets up the process's page entries properly, or kills process.

#### 4. Uses of page faults

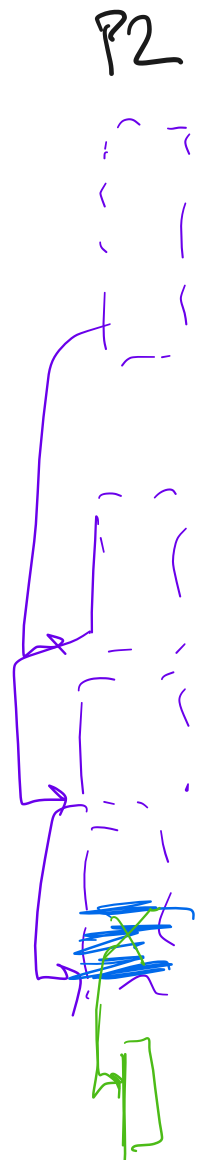
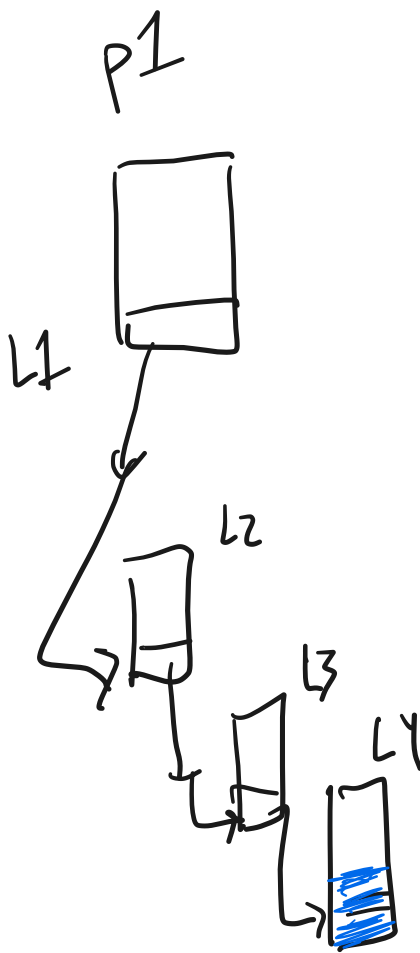
- Classic example: overcommitting physical memory

prog: 64 GB

4/w: 16 GB



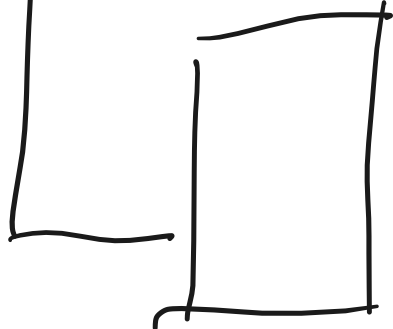
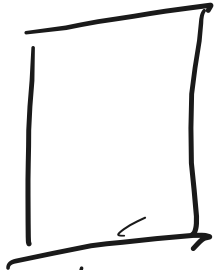
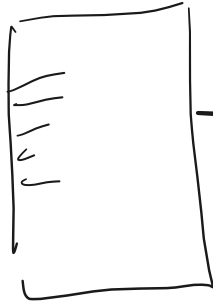
- Copy on write  
fork();



- Accounting

- Store memory across the network (DSM)

machine



- Paging in day to day use

- demand paging

- growing the stack

- BSS page allocation

- Shared text (code)

- Shared libraries

- Shared memory

---

## 5. Page faults: costs

look at AMAT (avg. memory access time)

$$\text{AMAT} = (1-p) \cdot \underbrace{(\text{mem access time})}_{t_M} + p \cdot \underbrace{(\text{page fault time})}_{t_D}$$

$p$  is probability (or frequency) of a page fault

mem access time  $\sim 100\text{ns}$   $t_M$

disk access time  $\sim 10\text{ms} = 10^7\text{ns}$   $t_D$

QUESTION: what is  $p$  such that paging hurts performance by less than 10%?

$$1.1 t_M > (1-p) \cdot t_M + p \cdot t_D$$

$$p \cdot t_M > -0.1 t_M + p \cdot t_D$$

$$p(t_M - t_D) > -0.1 t_M$$

$$\Rightarrow p(t_D - t_M) < 0.1 t_M$$

$$\Rightarrow p < \frac{0.1 t_M}{t_D - t_M}$$

$$\rightarrow \frac{0.1 \times 10^2}{10^7 \text{ ns} - 10^2 \text{ ns}}$$

$$\approx \frac{0.1 \times 10^2}{10^7}$$

$$\approx \frac{1}{10^6} = 10^{-6}$$

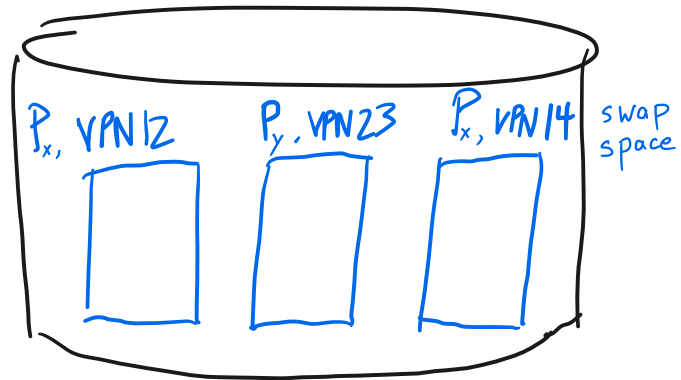
(After break)

## 6. Page replacement policies

PPN=46

RAM





- FIFO: eject oldest
- MIN (OPT): eject entry that won't be referenced for the longest time

input:  
 reference string  
 cache size

output:  
 number of evictions, or more generally: misses

FIFO

A B C A B D A D B C B

phys\_slot

S1

S2

S3

OPTIMAL

A B C A B D A D B C B

phys\_slot

S1

S2

S3

LRU

A B C A B D A D B C B

phys\_slot

S1

S2

S3



A B C D A B C D A B C D

phys-slot

S1

S2

S3

back to FIFO

3 entries A B C D A B E A B C D E

phys-slot

S1

S2

S3

4 entries

A B C D A B E A B C D E

phys-slot

S1

S2

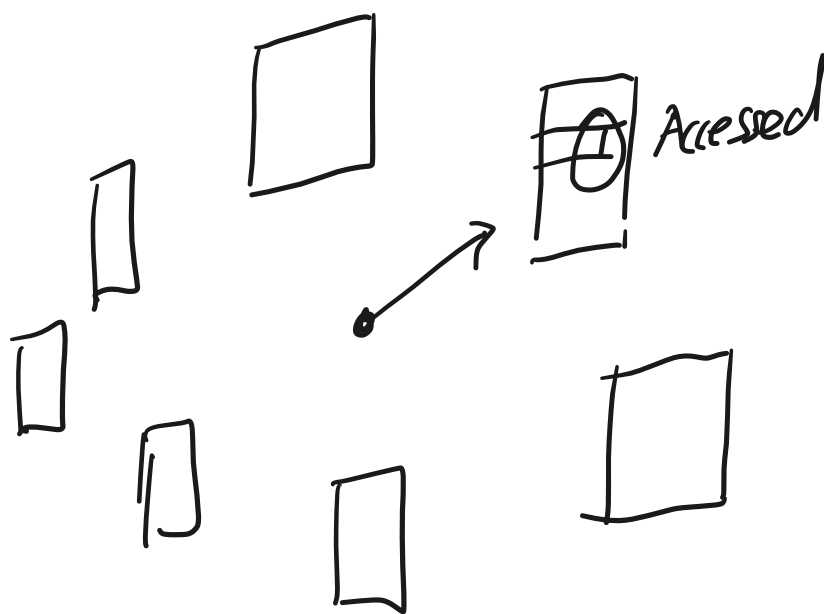
S3

S4

- OPT minimizes misses/swaps/evictions
  - but can't be implemented in general.

- LRU: approximates OPT (assuming what?)

- approximate LRU with CLOCK



H/W sets Accessed + Dirty bits

OS consumes these bits and clears them.

- Generalization of CLOCK:  $N^{\text{th}}$  Chance (see notes).

## 7. Thrashing

ex: program touches 50 pages, equiprobably  
but only 40 phys. frames (or slots)

Thrashing: processes demand more memory for active use than the system has.

3 reasons:

- (a) process has no temporal locality, or
- (b) " " temporal locality but not enough memory, or
- (c) individually all processes fit, but there's not enough memory.

lab 4 work here

↘ : thinks it is interacting with hardware

↘ : uses syscall interface (and possibly virtualization extensions)

↘ : paravirtualized (in between hardware + syscall)

p-forks | p-allocator

WeensyOS

QEMU

Linux

Docker

Your OS (Windows, Mac OS, ...)   
 WSL: Linux

Physical Hardware   
 x86 or ARM

cpu, h/w  
process  
cpu, h/w  
process  
cpu, h/w  
process

### Hints:

- processes: files matching p-\*.c
- kernel code: files matching k-\*.c, k-\*.s
- system calls and returns // (\*) /\* cousin of mmap() \*/

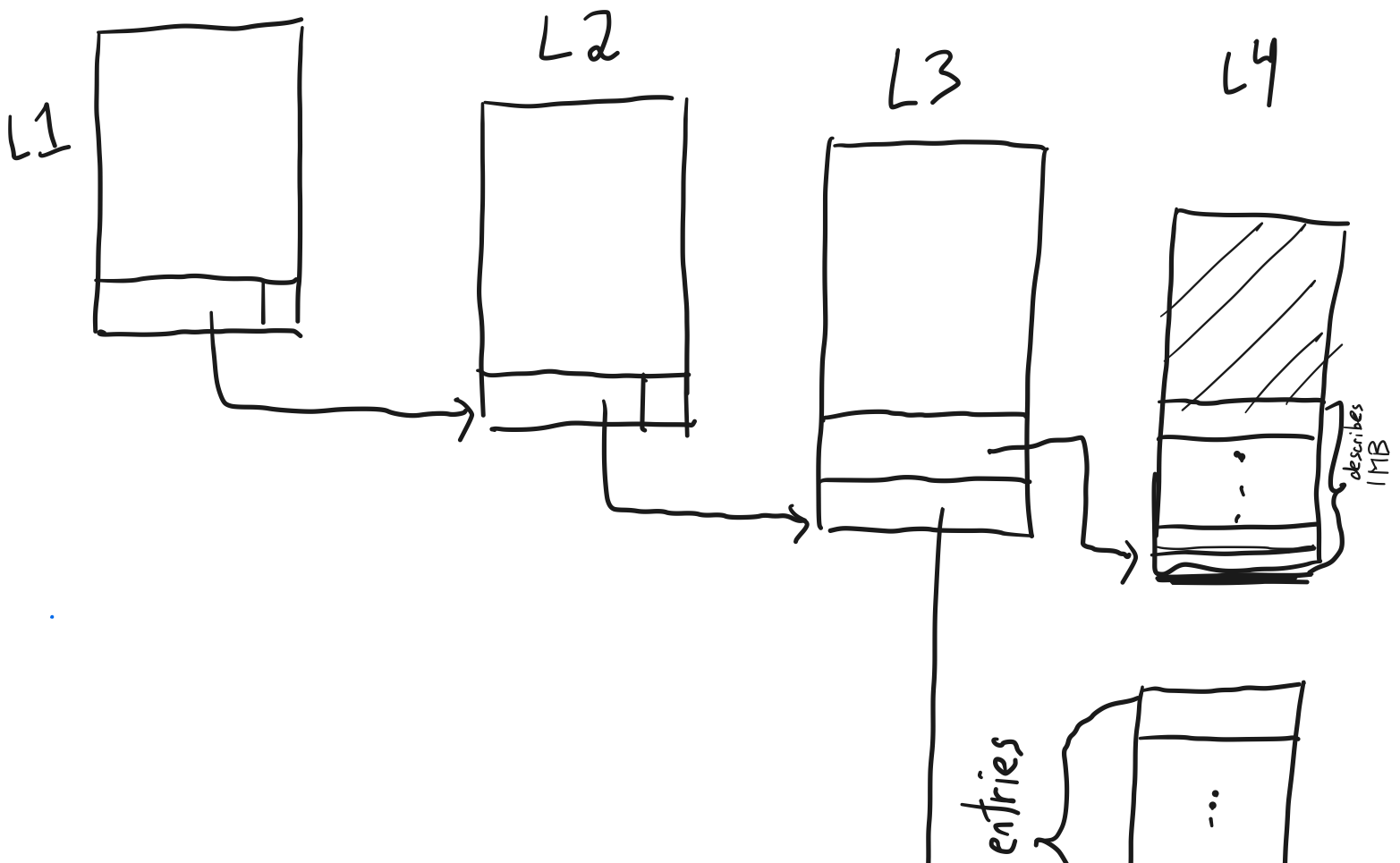
sys\_page\_alloc, lookin process.h

kernel returns: exception\_return();  
%rax contains return value of system call  
(errno, rax)

%rdi:  
arg to  
syscall

- you'll use virtual-memory-map() = (NULL vs. non-NULL)
- pay attention to the allocator argument
- make sure your allocator initializes the page table  
memset(addr, 0, len);

- a process's virtual address space: 3 MB. What's the page table structure?



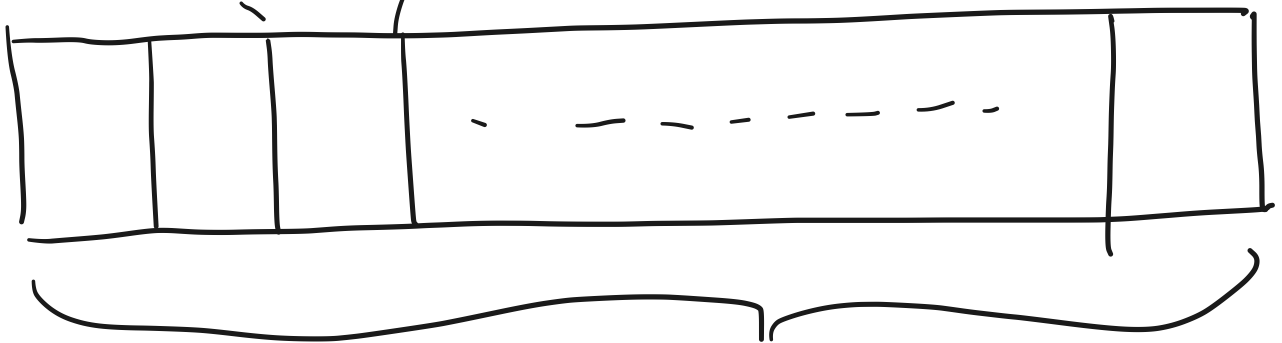


each entry  
describes a mapping  
for one page  
(4KB)

---

PCB  $\equiv$  struct proc (in kernel.h)

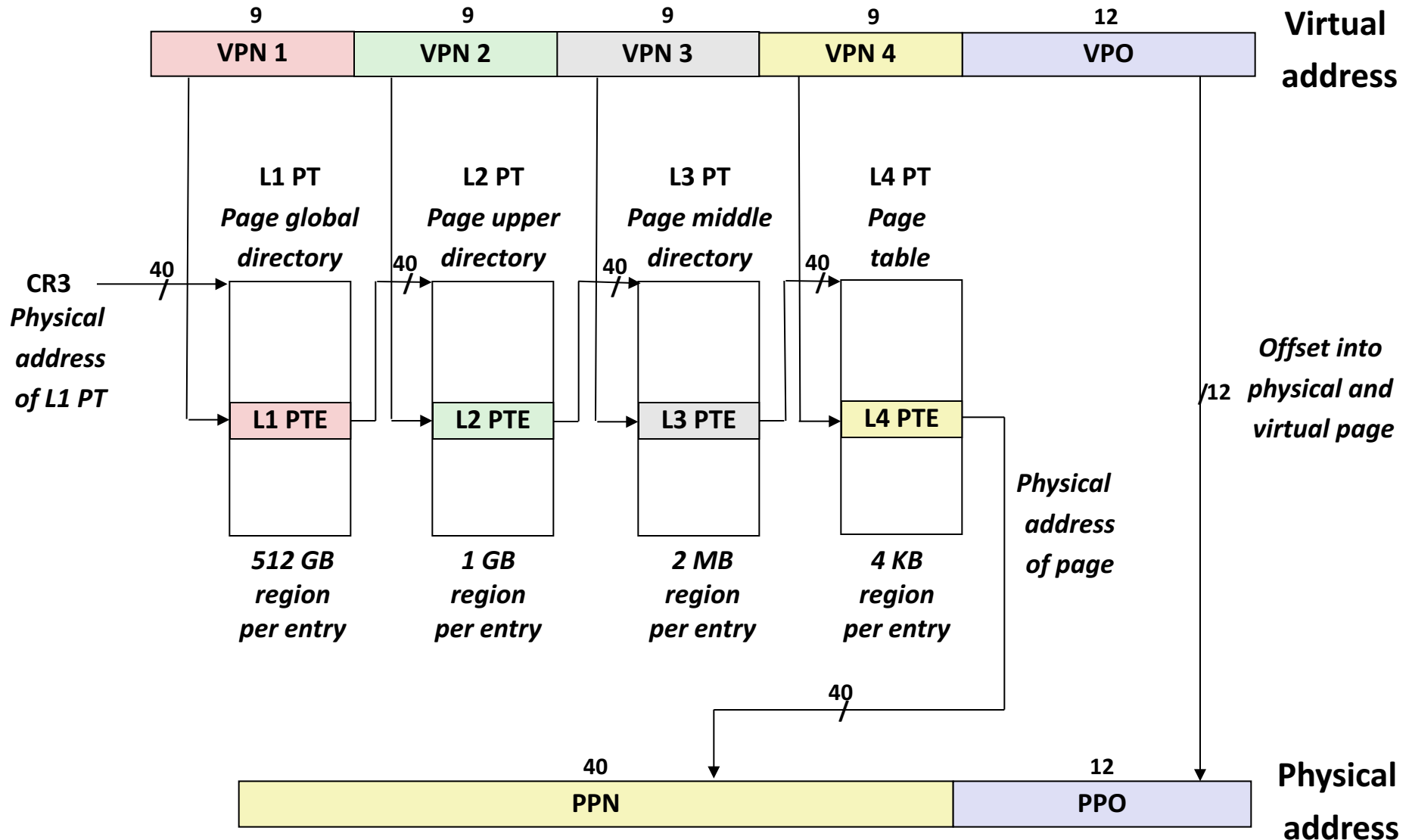
struct physical\_pageinfo {  
int8\_t owner;  
int8\_t refcount;  
}



one per physical page

this is not a page table; it is bookkeeping.

# 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

63	62	52	51	12	11	9	8	7	6	5	4	3	2	1	0	
XD	Unused		Page table physical base address			Unused		G	PS		A	CD	WT	U/S	R/W	P=1
Available for OS															P=0	

**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

63	62	52	51	12	11	9	8	7	6	5	4	3	2	1	0
XD	Unused	Page physical base address				Unused	G		D	A	CD	WT	U/S	R/W	P=1
Available for OS (for example, if page location on disk)														P=0	

**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

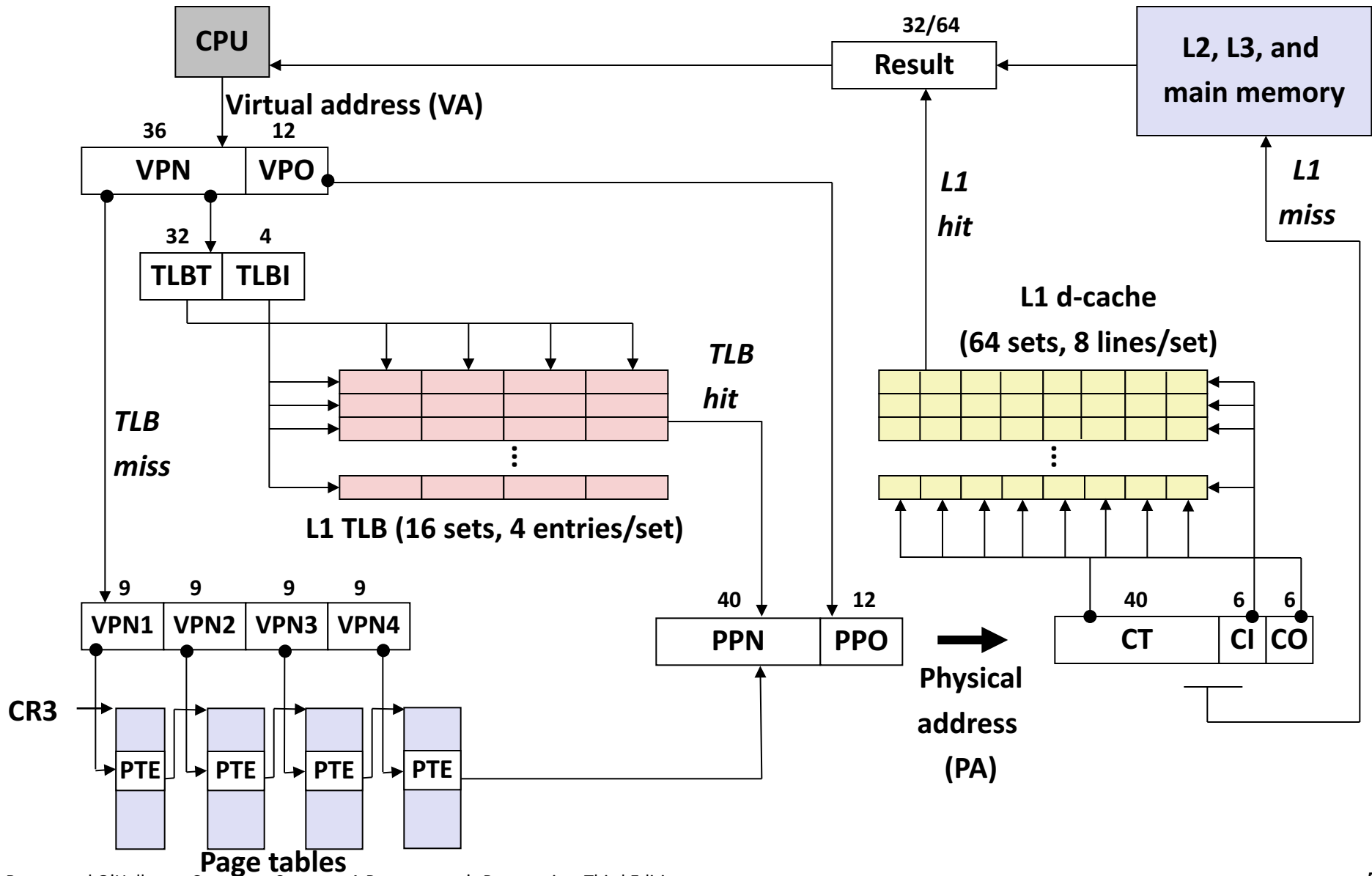
**A:** Reference bit (set by MMU on reads and writes, cleared by software)

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

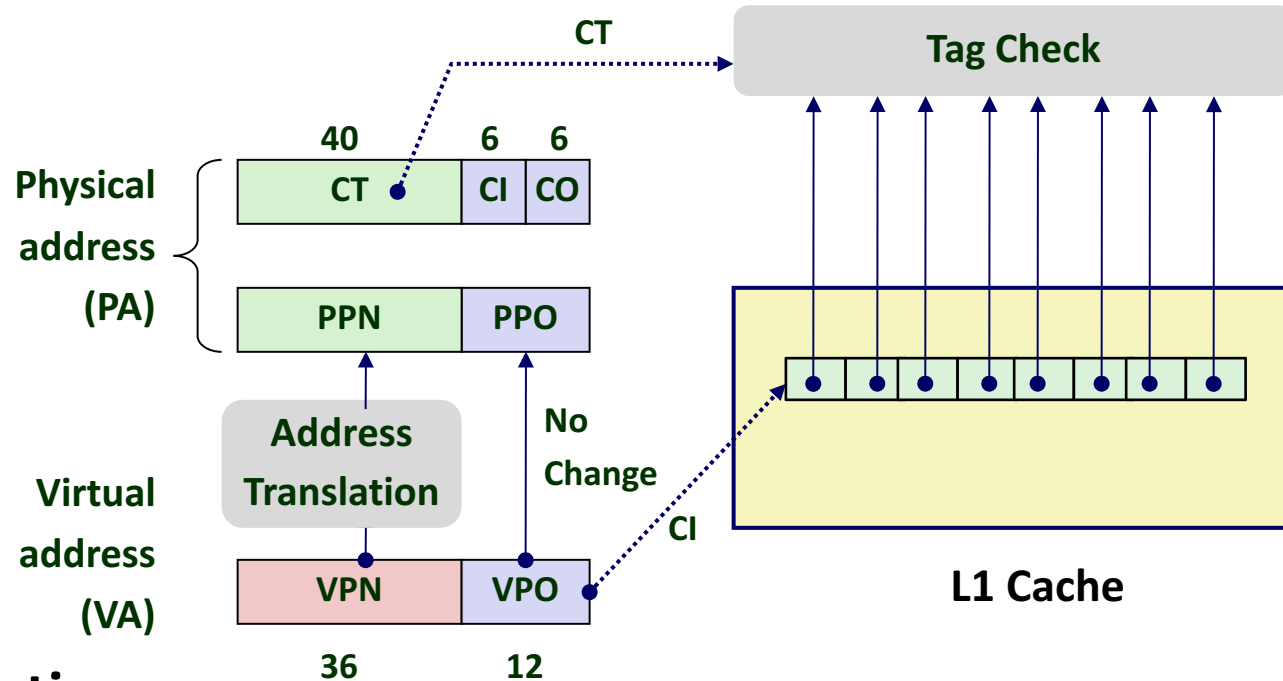
**Page physical base address:** 40 most significant bits of physical page address  
(forces pages to be 4KB aligned)

**XD:** Disable or enable instruction fetches from this page.

# End-to-end Core i7 Address Translation



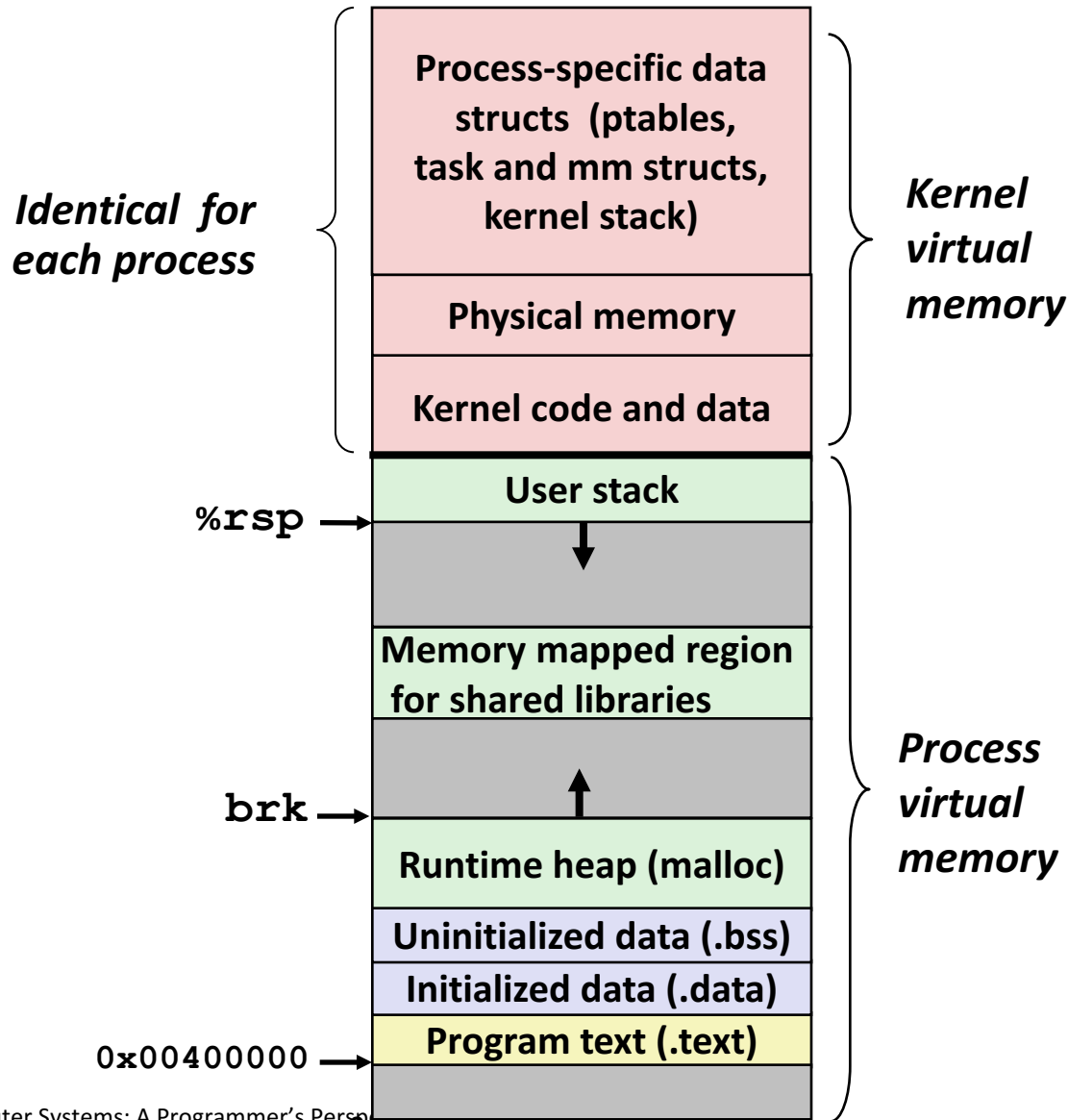
# Cute Trick for Speeding Up L1 Access

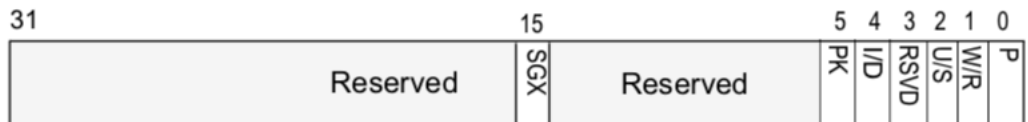


## ■ Observation

- Bits that determine CI identical in virtual and physical address
- Can index into cache while address translation taking place
- Cache carefully sized to make this possible: 64 sets, 64-byte cache blocks
- Means 6 bits for cache index, 6 for *cache* offset
- That's 12 bits; matches *VPO*, *PPO* → One reason pages are  $2^{12}$  bits = 4 KB

# Virtual Address Space of a Linux Process





- P**      0 The fault was caused by a non-present page.  
           1 The fault was caused by a page-level protection violation.
- W/R**    0 The access causing the fault was a read.  
           1 The access causing the fault was a write.
- U/S**    0 A supervisor-mode access caused the fault.  
           1 A user-mode access caused the fault.
- RSVD**   0 The fault was not caused by reserved bit violation.  
           1 The fault was caused by a reserved bit set to 1 in some  
             paging-structure entry.
- I/D**    0 The fault was not caused by an instruction fetch.  
           1 The fault was caused by an instruction fetch.
- PK**     0 The fault was not caused by protection keys.  
           1 There was a protection-key violation.
- SGX**    0 The fault is not related to SGX.  
           1 The fault resulted from violation of SGX-specific access-control  
             requirements.

**Figure 4-12. Page-Fault Error Code**