Last Class

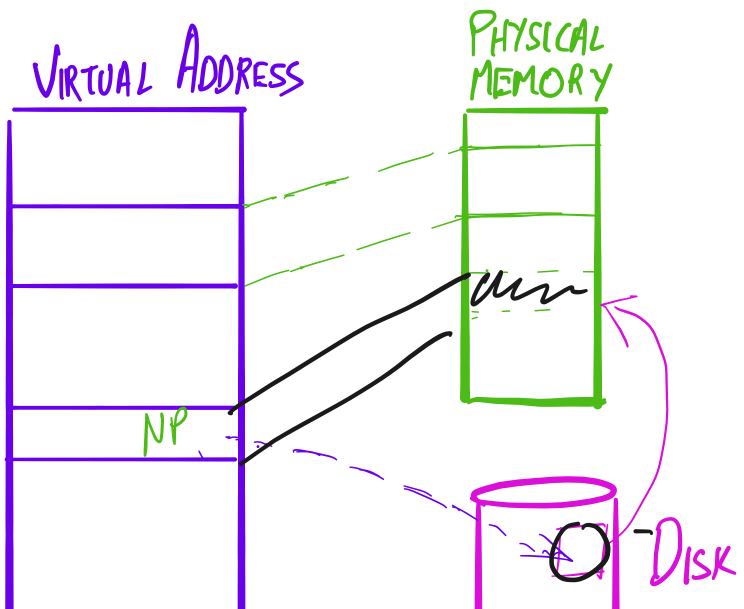
① Page Faults
- ↳ What They Are
- → How They Occur
- → Information They Provide
  - ↳ Faulting Address (%CR2)
  - → Why The Fault Occurred

② Uses
- ① Copy-On-Write

Today

Uses: Demand Paging

Goal



make run

Why?

1. Loading Executables

   $\Uparrow$

2. mmap(2)

3. Using More Memory Than Physically Availaible

How?

1. PF Occurs

   ↳ Check If Page Should Be Loaded From Disk

2. Allocate Physical Page

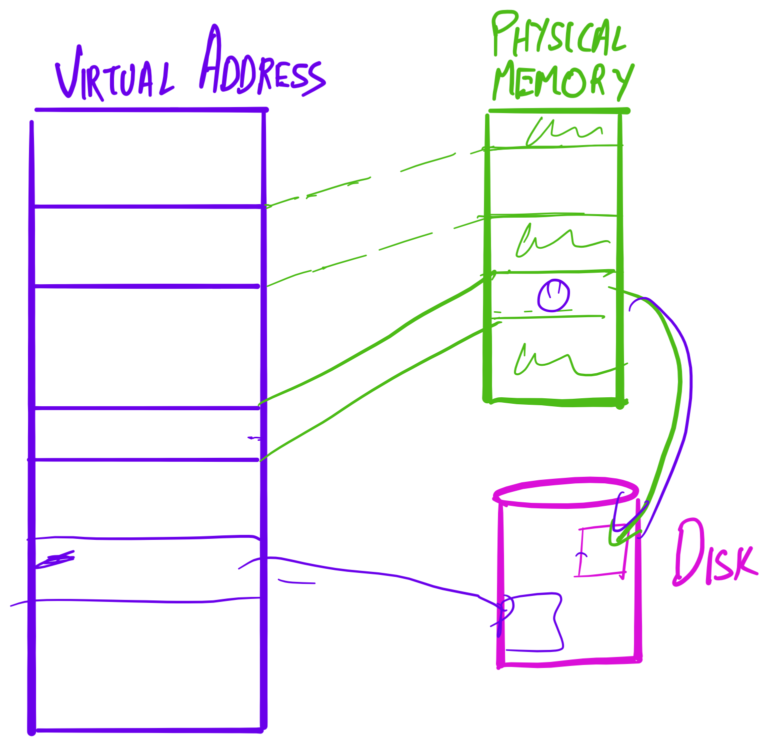
   Check If Free Physical Pages Are Available

   ↳ No → Evict To Free Physical Pages

Yes → Allocate Page ←

③ Load & Map /dev/zero

Eviction



Today's Question: What Page To Evict?

Page Replacement Policy

Aside: Does This Even Matter Anymore?

- CDNs
- Databases
- [illegible]

- ML Training & Inference

Policies

- FIFO: First In First Out

- MIN (Optimal): Clairvoyant (Knows The Future)
  Evict Page That Won't Be Accessed
  For The Longest Time

Example

3 Physical Pages

Page

Access: ABC ABD ADB CB

FIFO

	A	B	C	A	B	D	A	D	B	C	B
P0	A			h		**D**		h		**C**	
P1		B			h		**A**				
P2			C						**B**		h

# Of Page-Ins / Misses 7 } 7/11

# Of Hits 4

OF HITS

MIN

	A	B	C	A	B	D	A	D	B	C	B
P0	A			h			h			C	
P1		B			h				h		h
P2			C			D		h			

OF PAGE-INS / MISSES 5

OF HITS 6

5/11

WHY IT MATTERS

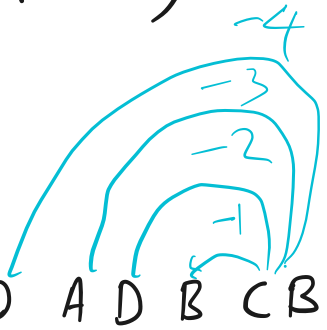
$$\text{Avg Memory Access Time} = \boxed{p} \cdot \text{page fault time} + \boxed{(1-p)} \cdot \text{memory access latency}$$

Page-in/miss

ASSUMPTION Memory Access Lat. $\ll$ Page Fault Time

$(\sim 100\text{ns})$ $(\sim 10\text{ms} = 10^7\text{ns})$

LRU: Least Recently Used



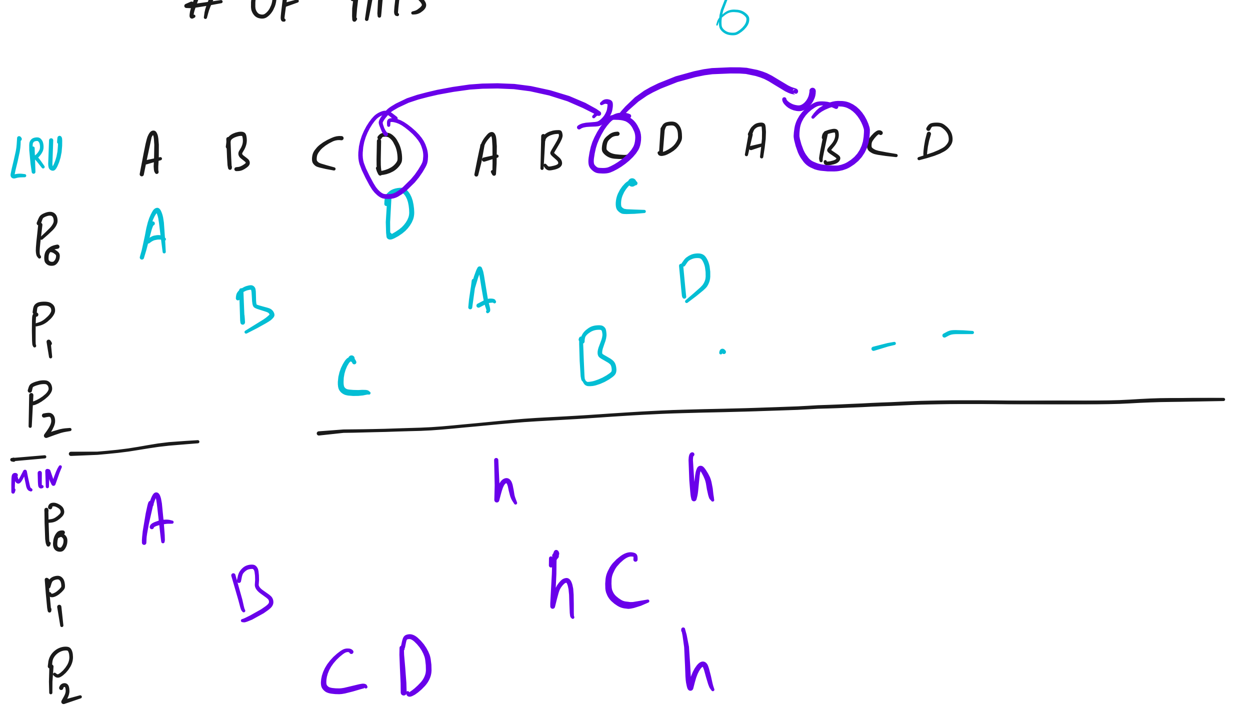
LRU

	A	B	C	A	B	D	A	D	B	C	B
P0	A			h			h			C	
P1		B			h				h		h
P2			C			D		h			

OF PAGE-INS / MISSES 5

OF HITS

# of hits

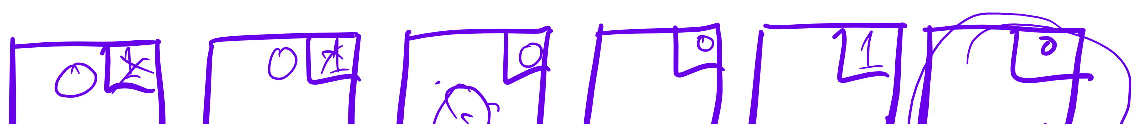


Trade-Offs

- FIFO + Simple
  - High miss rates
- ~~OPT~~ Min + Optimal
  - know the future
- LRU + Empirically close to Min
  -

Approximating LRU - Clock

Reference Bit (A)



MISCELLANEOUS

- BELADI'S ANOMALY
  - ↳ FIFO

	A	B	C	D	A	B	E	A	B	C	D	E
P_0												
P_1												
P_2												
P_0												
P_1												
P_2												
P_3												

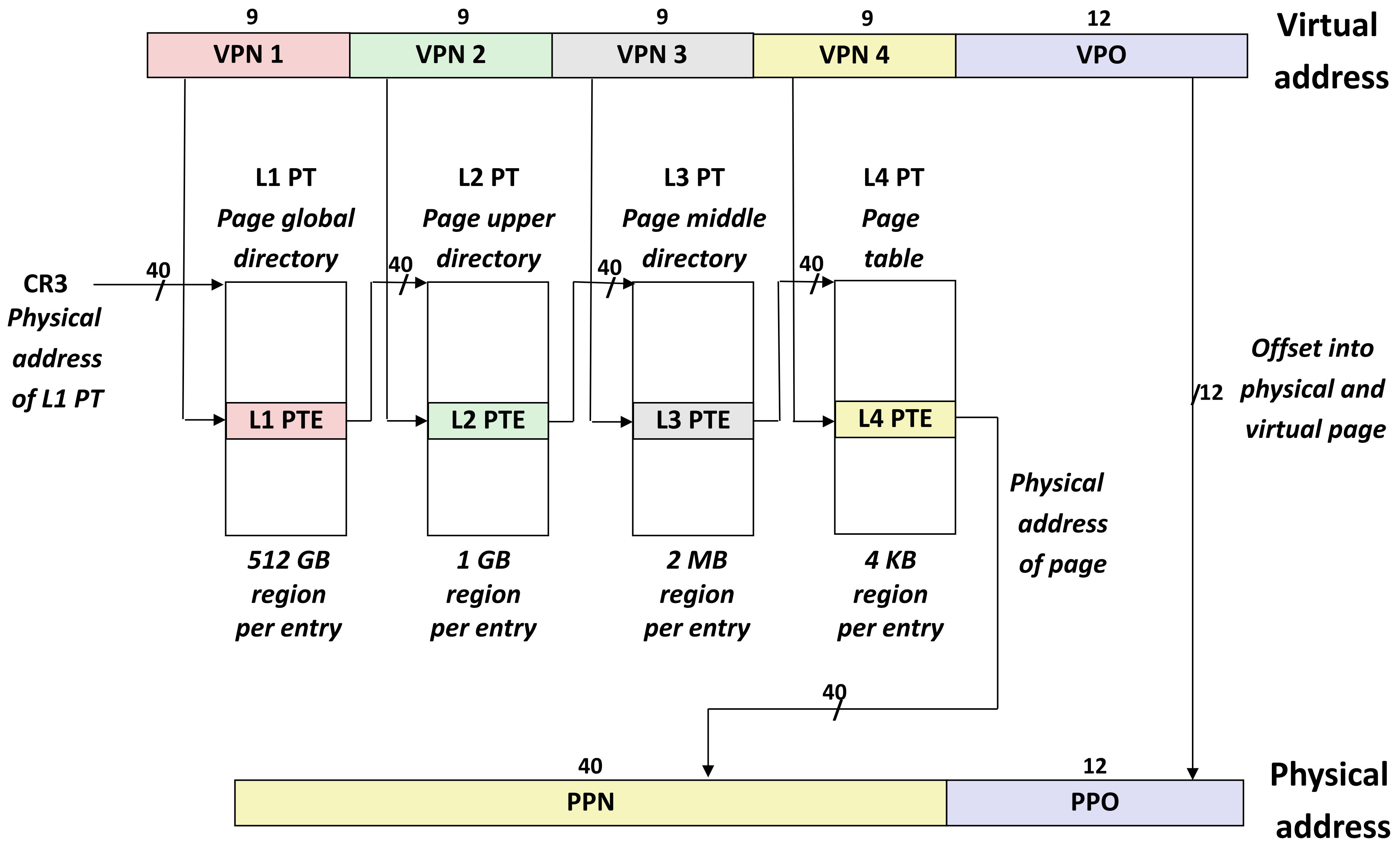
3 Hit

4 Hit

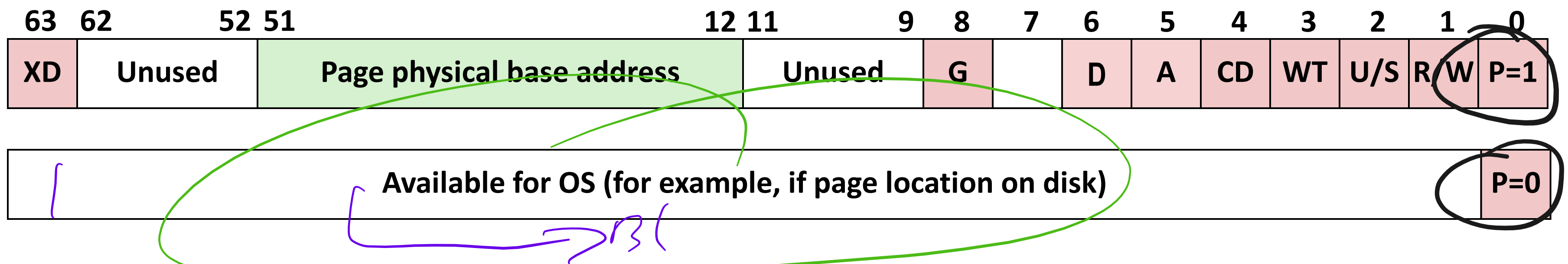
THRASHING

PAGE FAULTS FOR ACCOUNTING

Core i7 Page Table Translation



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.

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.

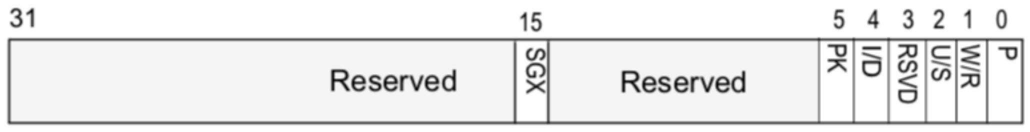
A: Reference bit (set by MMU on reads and writes, cleared by software).

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.

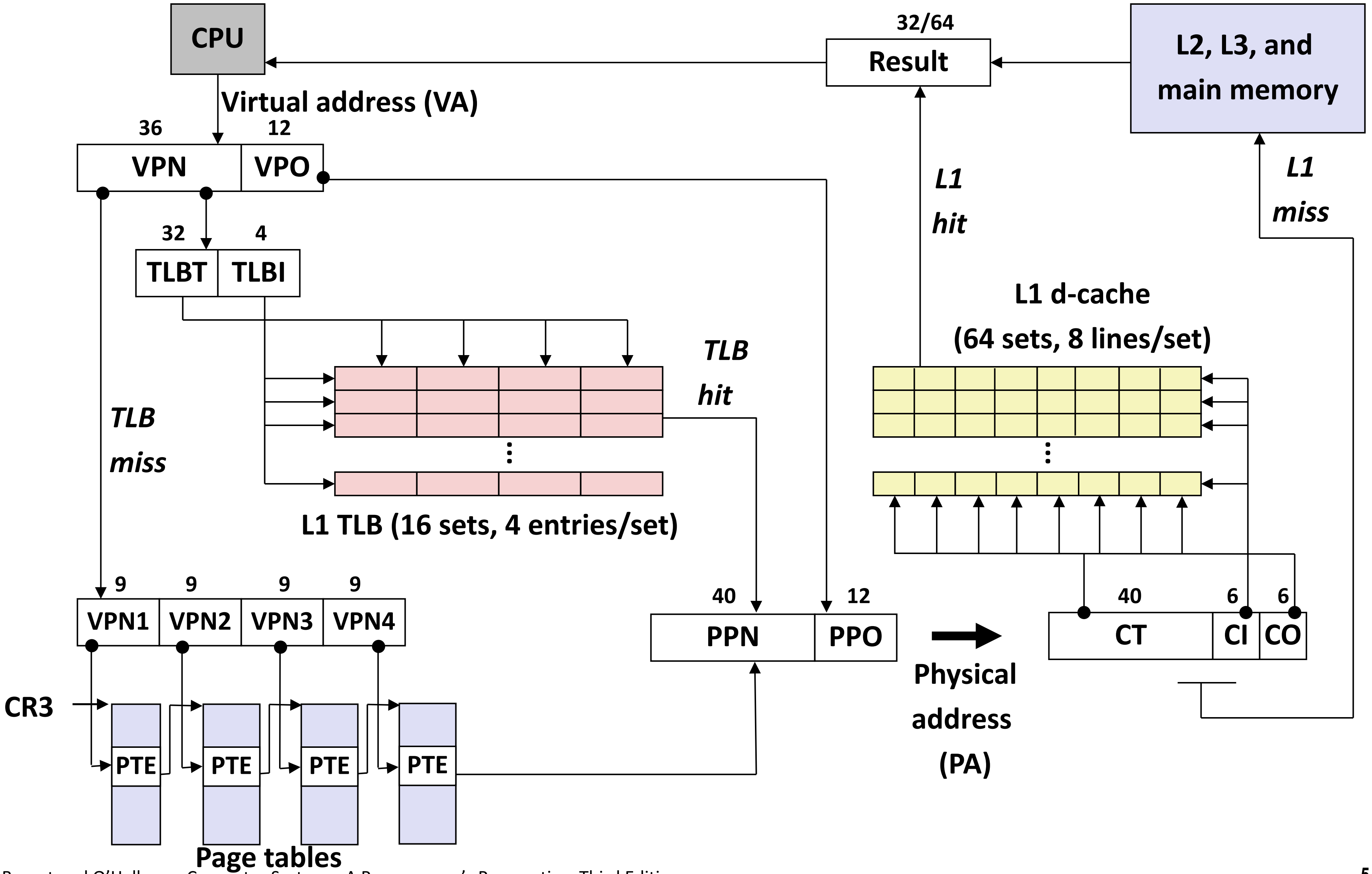
s explain how the bits in the error code describe the nature of the page-fault exception.



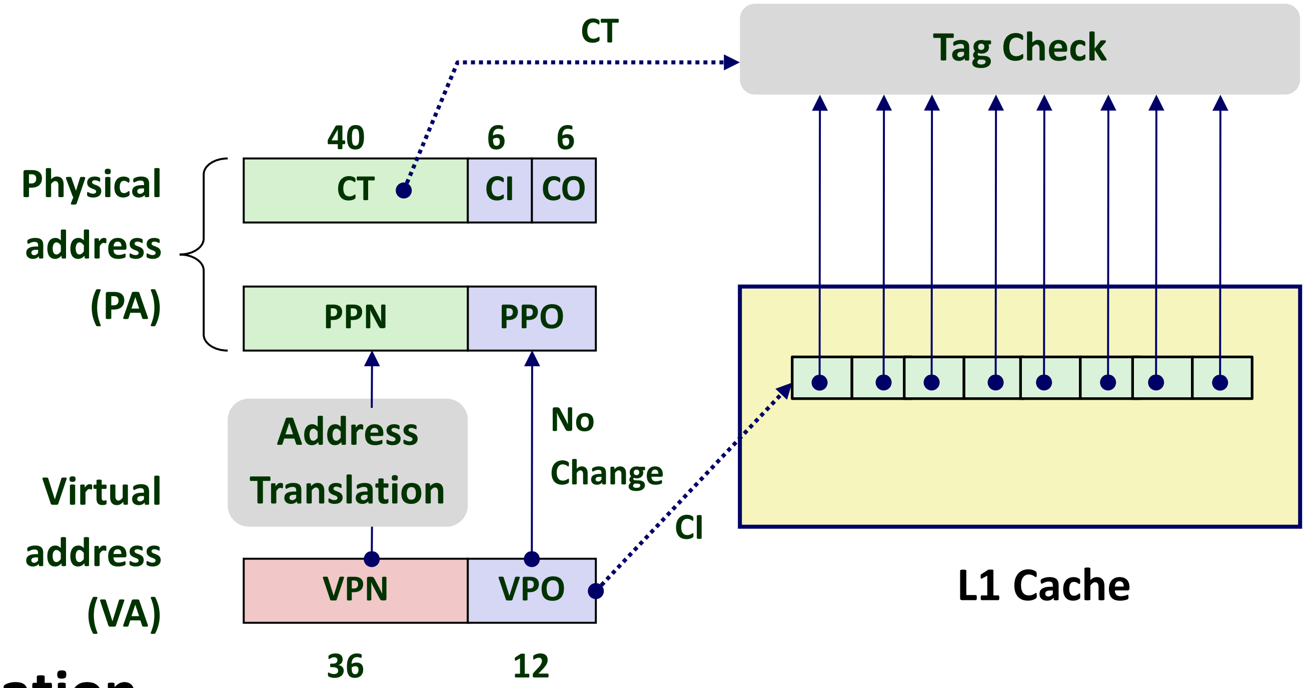
P	0	The fault was caused by a non-present page.
	1	The fault was caused by a page-level protection violation.
W/R	0	The access causing the fault was a read.
	1	The access causing the fault was a write.
U/S	0	A supervisor-mode access caused the fault.
	1	A user-mode access caused the fault.
RSVD	0	The 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

End-to-end Core i7 Address Translation

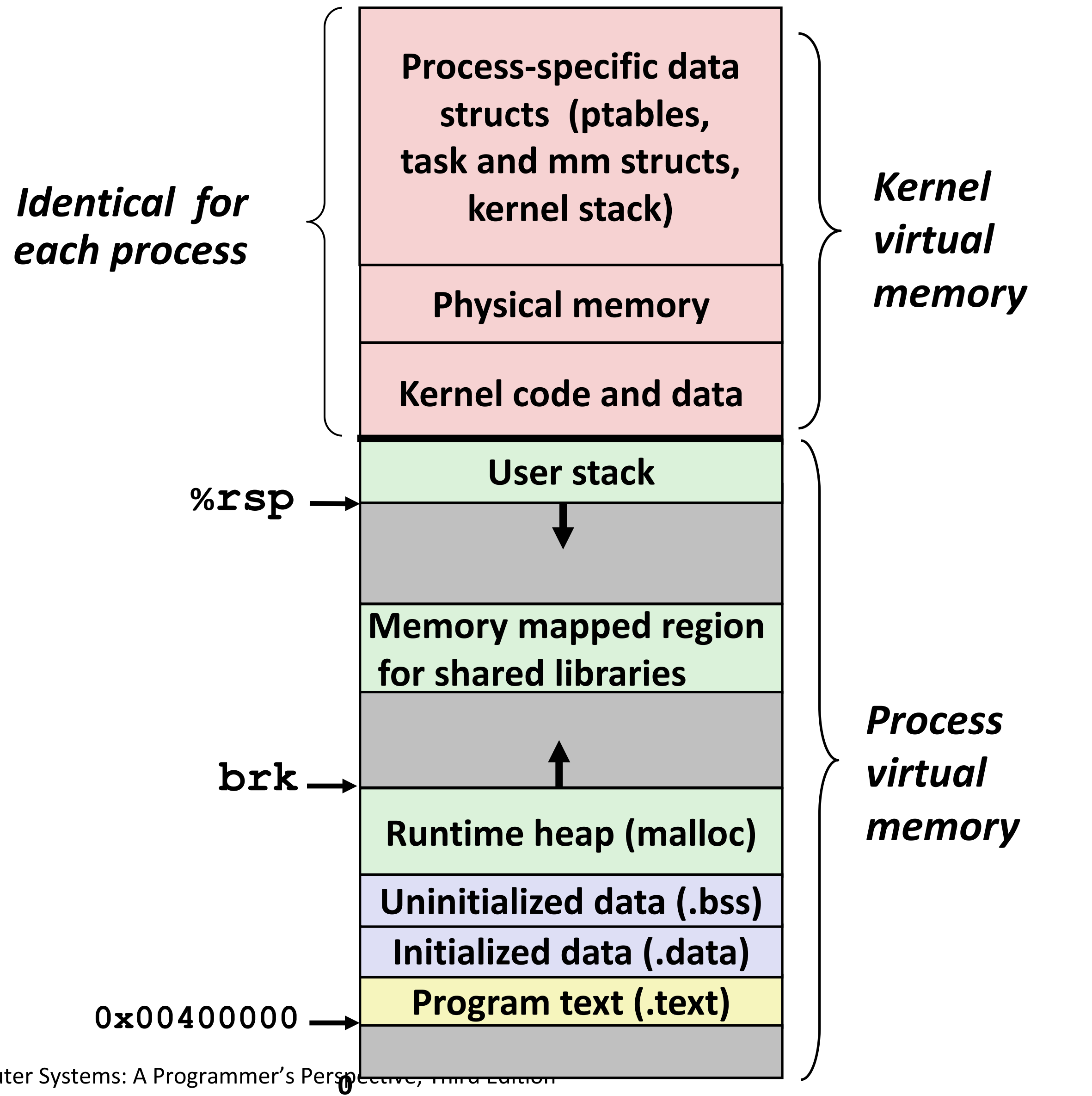


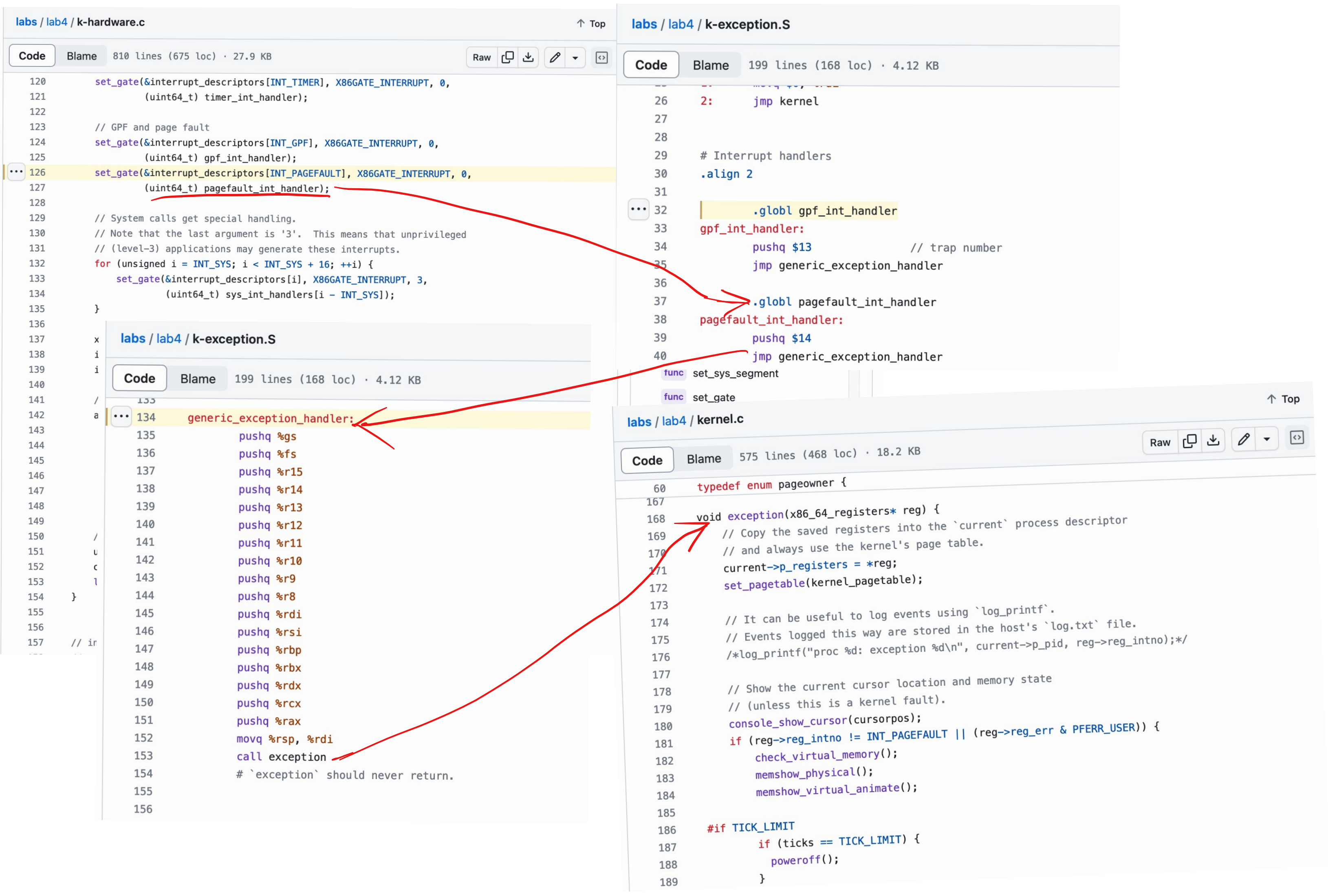
Cute Trick for Speeding Up L1 Access



- **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





```c
    set_gate(&interrupt_descriptors[INT_TIMER], X86GATE_INTERRUPT, 0,
             (uint64_t) timer_int_handler);

    // GPF and page fault
    set_gate(&interrupt_descriptors[INT_GPF], X86GATE_INTERRUPT, 0,
             (uint64_t) gpf_int_handler);
    set_gate(&interrupt_descriptors[INT_PAGEFAULT], X86GATE_INTERRUPT, 0,
             (uint64_t) pagefault_int_handler);

    // System calls get special handling.
    // Note that the last argument is '3'.  This means that unprivileged
    // (level-3) applications may generate these interrupts.
    for (unsigned i = INT_SYS; i < INT_SYS + 16; ++i) {
        set_gate(&interrupt_descriptors[i], X86GATE_INTERRUPT, 3,
                 (uint64_t) sys_int_handlers[i - INT_SYS]);
    }

    x
    i
    i

    /
    a







    /
    u
    c
    l
}


// ir
```

```asm
2:      jmp kernel


# Interrupt handlers
.align 2

        .globl gpf_int_handler
gpf_int_handler:
        pushq $13               // trap number
        jmp generic_exception_handler

        .globl pagefault_int_handler
pagefault_int_handler:
        pushq $14
        jmp generic_exception_handler
```

```asm
generic_exception_handler:
        pushq %gs
        pushq %fs
        pushq %r15
        pushq %r14
        pushq %r13
        pushq %r12
        pushq %r11
        pushq %r10
        pushq %r9
        pushq %r8
        pushq %rdi
        pushq %rsi
        pushq %rbp
        pushq %rbx
        pushq %rdx
        pushq %rcx
        pushq %rax
        movq %rsp, %rdi
        call exception
        # `exception` should never return.
```

```c
typedef enum pageowner {
void exception(x86_64_registers* reg) {
    // Copy the saved registers into the `current` process descriptor
    // and always use the kernel's page table.
    current->p_registers = *reg;
    set_pagetable(kernel_pagetable);

    // It can be useful to log events using `log_printf`.
    // Events logged this way are stored in the host's `log.txt` file.
    /*log_printf("proc %d: exception %d\n", current->p_pid, reg->reg_intno);*/

    // Show the current cursor location and memory state
    // (unless this is a kernel fault).
    console_show_cursor(cursorpos);
    if (reg->reg_intno != INT_PAGEFAULT || (reg->reg_err & PFERR_USER)) {
        check_virtual_memory();
        memshow_physical();
        memshow_virtual_animate();

#if TICK_LIMIT
        if (ticks == TICK_LIMIT) {
            poweroff();
        }
```