☑ 1. Last time

☑ 2. Page faults: intro + mechanics

☑ 3. Page faults: uses (of paging)

☑ 4. Page faults: costs

□ 5. Page replacement policies

□ 6. Thrashing

---

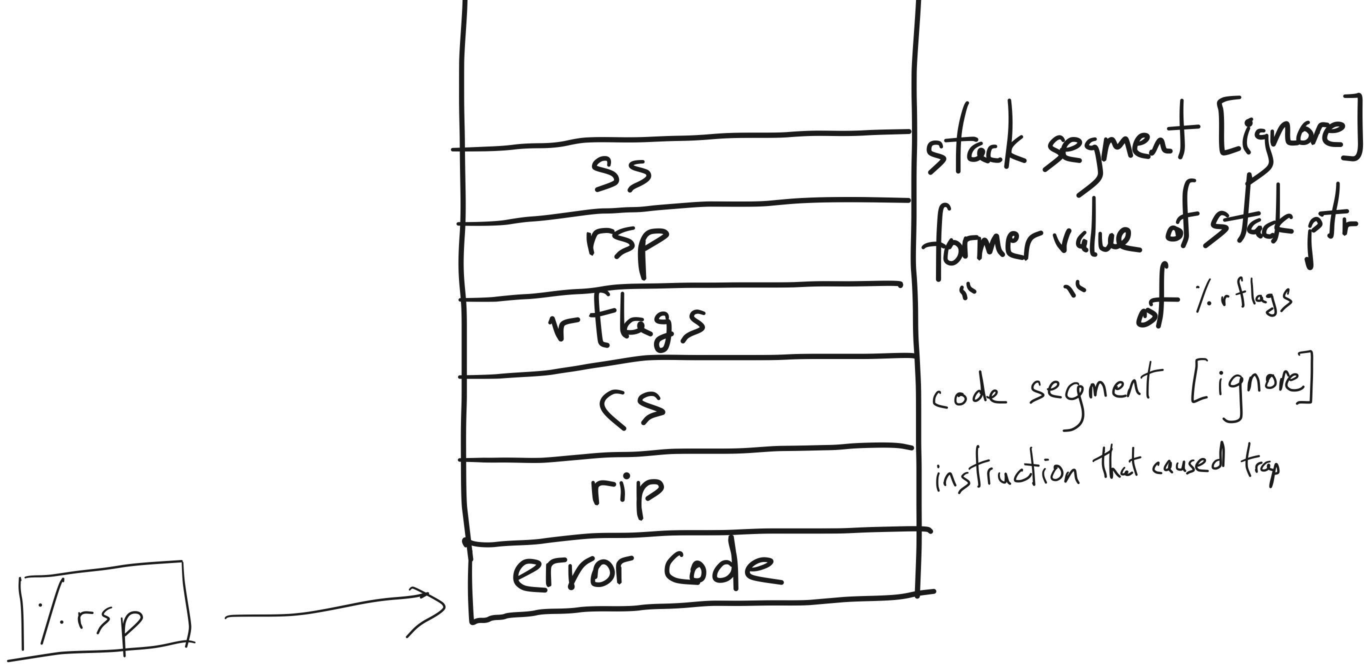
2. Page faults: intro + mechanics

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



$\boxed{\%rip}$ → code to handle the trap

error code : see last pg of handout

[_ _ _ _ I/D _ U/S W/R P]

$\boxed{\%cr2}$ holds faulting virtual address

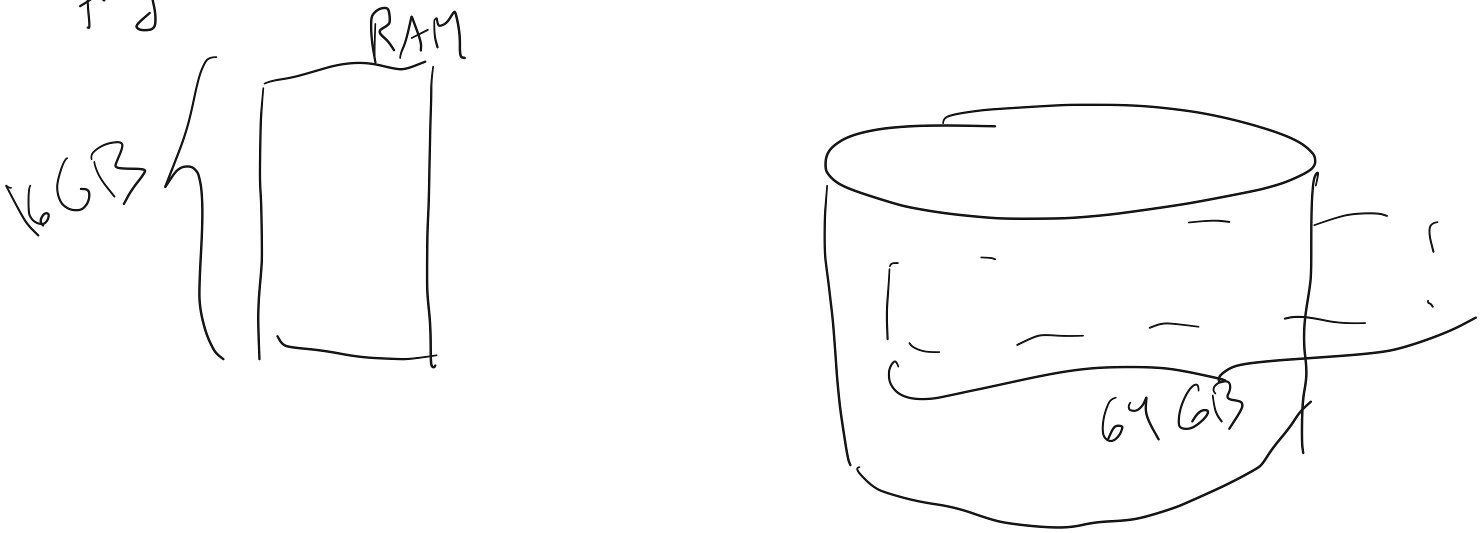
intent: on pg fault, kernel sets up the process's page entries properly, or kills process.

---

3. Uses of page faults

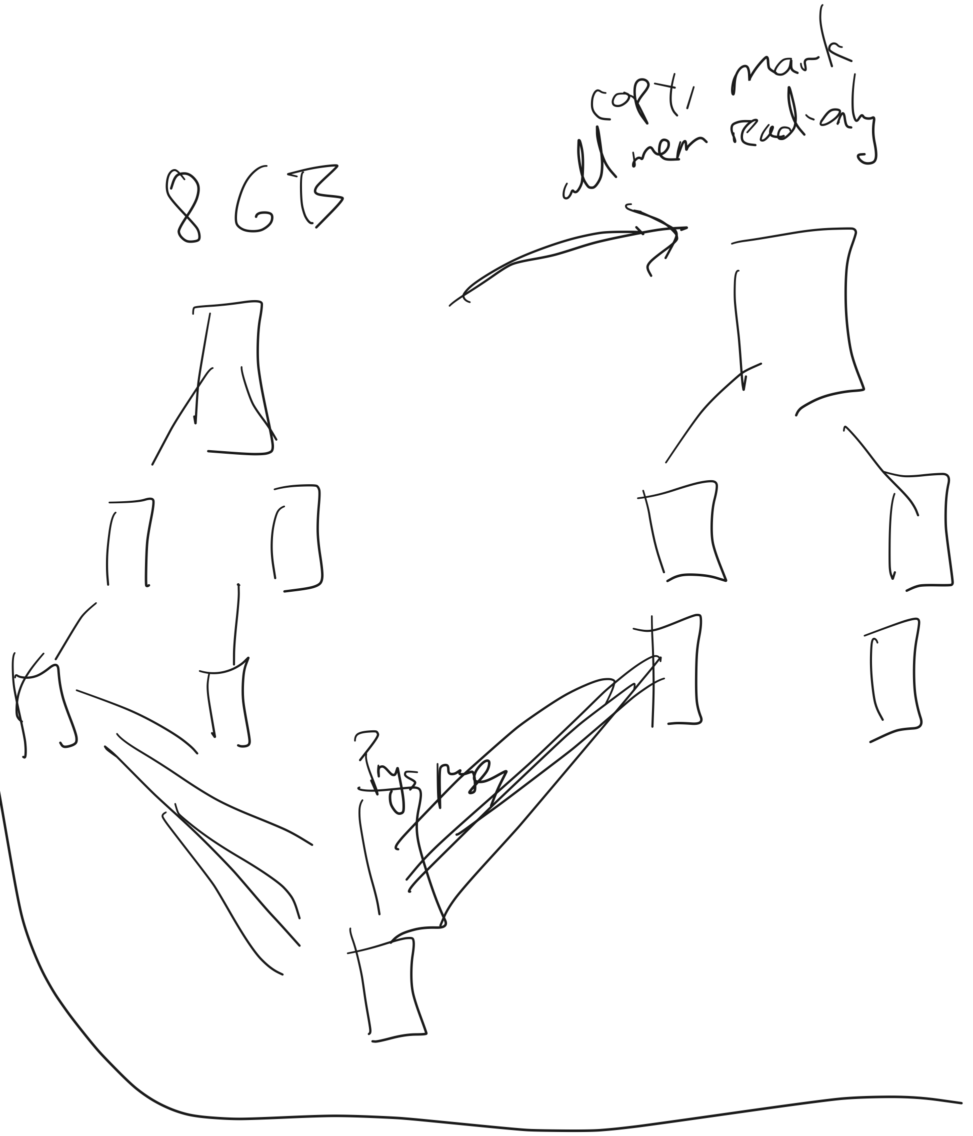
- Classic example: overcommitting physical memory prog: 64 GB

H/w: 16 GB



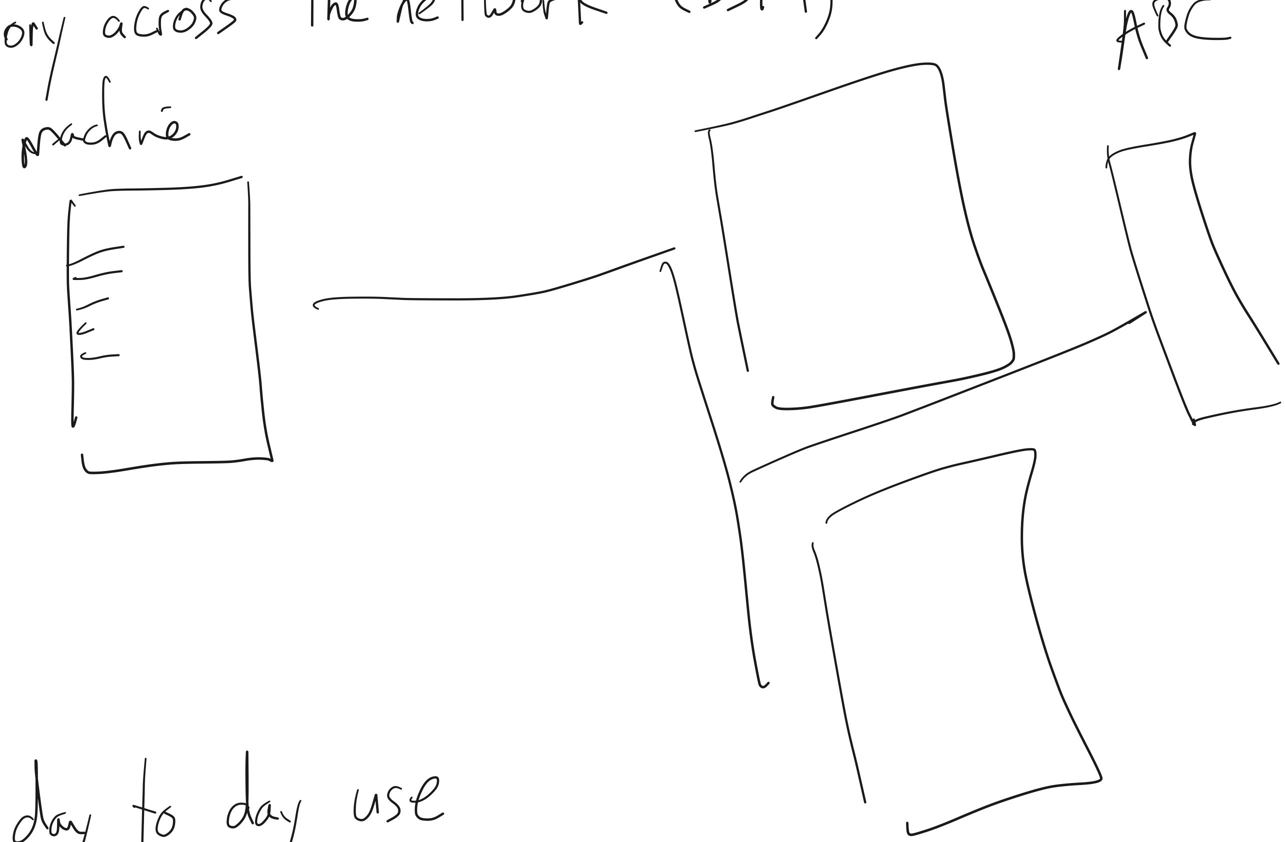
- Copy on write

fork()



- Accounting

- Store memory across the network (DSM)

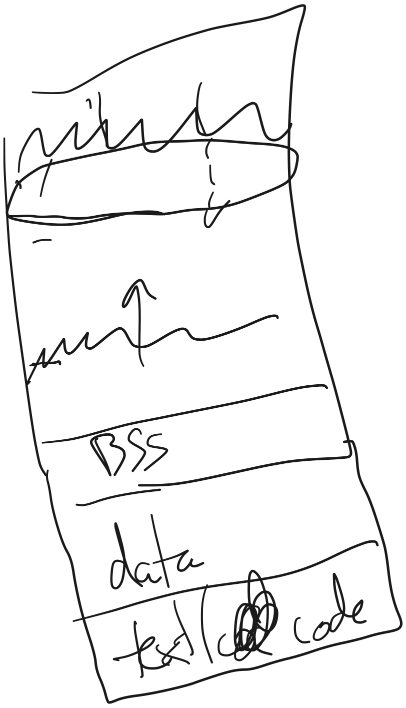


- Paging in day to day use

  - demand paging

  - growing the stack

  - BSS page allocation



- Shared text (code)

- Shared libraries

- Shared memory

Randy Pausch
time management

---

4. Page faults: costs

look at AMAT (avg. memory access time)

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

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

mem access time ~ 100ns t_M

disk access time ~ 10ms $= 10^7$ ns t_D

QUESTION: what is p s.t. paging hurts performance by less than 10%?

w/out paging: t_M

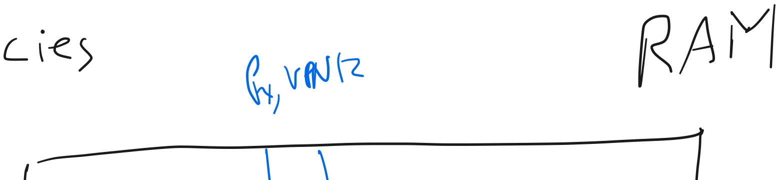
w/ paging: $1.1 * t_M = (1-p) * t_M + p * t_D$

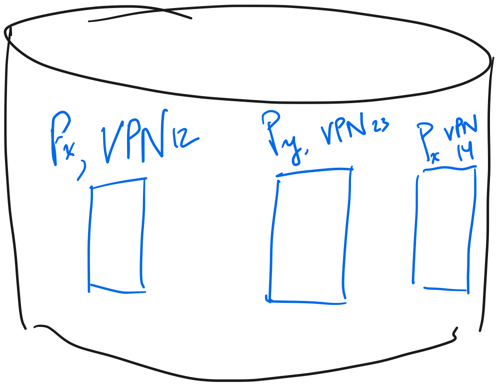
$$.1 * t_M = (p)(-t_M) + p * t_D$$

$$p = \frac{.1 * t_M}{t_D - t_M} = \frac{10\text{ ns}}{10^7\text{ ns}} = \frac{1}{10^6} = 10^{-6}$$

$$p = 10^{-6} \sim \frac{1}{1,000,000}$$

5. Page replacement policies





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

input:
reference string
cache size

output:
number of evictions

FIFO

phys_slot	A	B	C	A	B	D	A	D	B	C	B
S1	A			h		D		h		C	
S2		B			h		A				h
S3			C						B		

7 evictions, 4~~8~~ hits

OPTIMAL

phys_slot	A	B	C	A	B	D	A	D	B	C	B
S1	A			h			h			C	
S2		B			h				h		h
S3			C			D		h			

5 evictions, 6~~8~~ hits

LRU

phys_slot	A	B	C	A	B	D	A	D	B	C	B
S1	A			h			h		h	C	h
S2		B			h						
S3			C			D		h			

5 evictions, 6 hits

	A	B	C	D	A	B	C	D	A	B	C	D
phys-slot												
S1	A			D			C			B		
S2		B			A			D			C	
S3			C			B			A			D

11 evictions, 0 hits

FIFO
3entries

	A	B	C	D	A	B	E	A	B	C	D	E
phys-slot												
S1												
S2												
S3												

worse

4entries

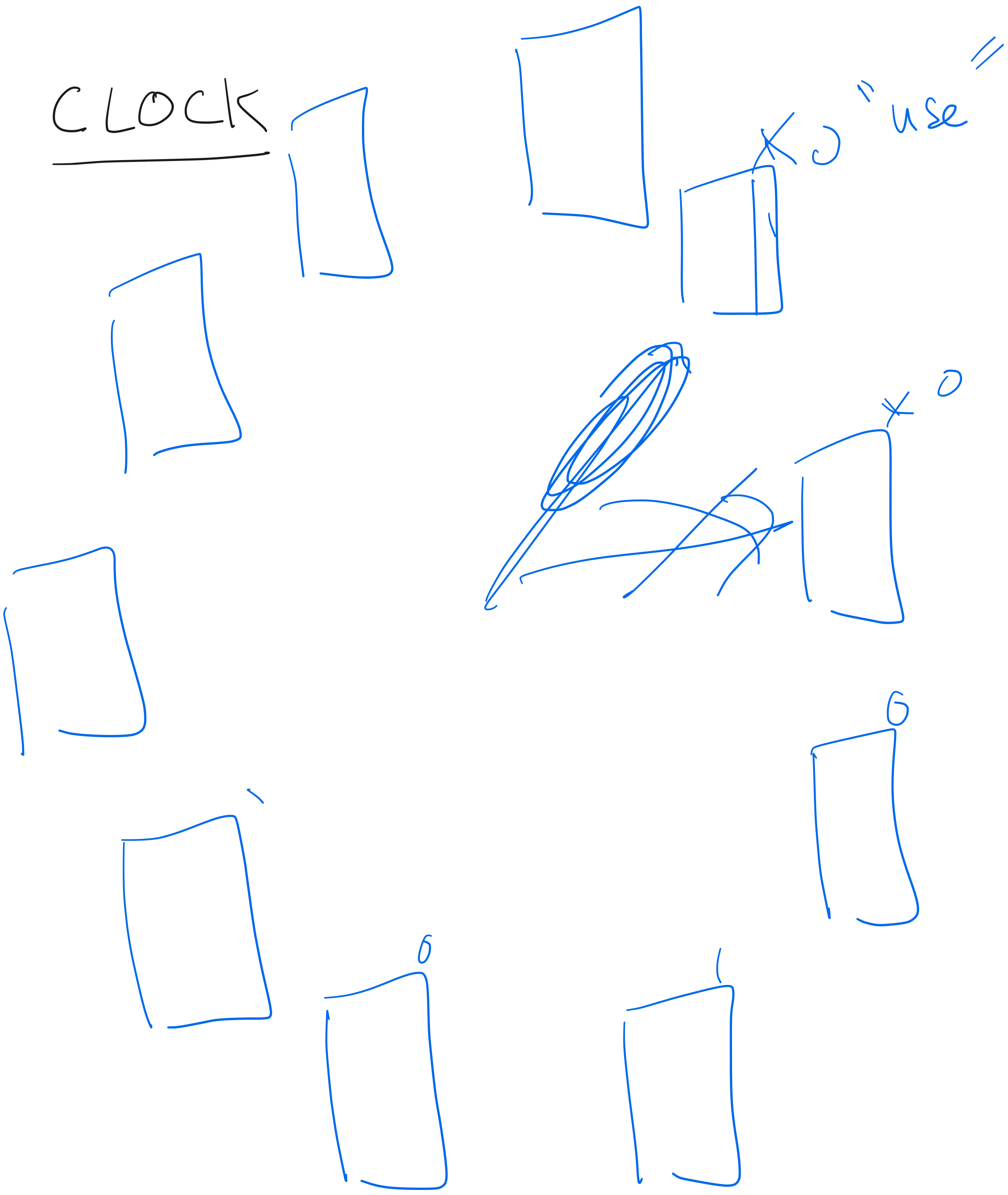
	A	B	C	D	A	B	E	A	B	C	D	E
phys-slot												
S1												
S2												
S3												
S4												

Belady's anomaly

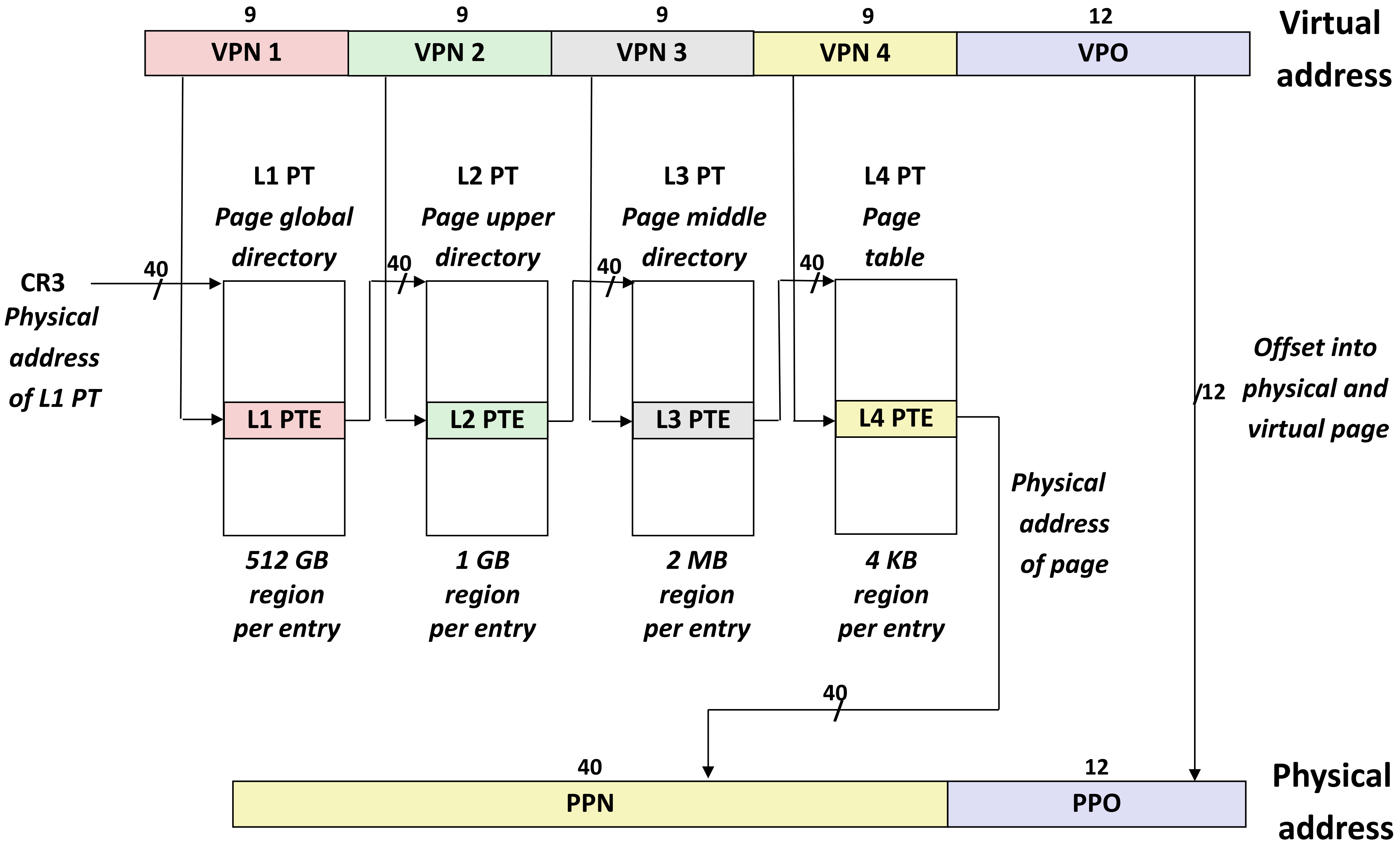
OPT ~ LRU

We are motivated to implement LRU

CLOCK



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

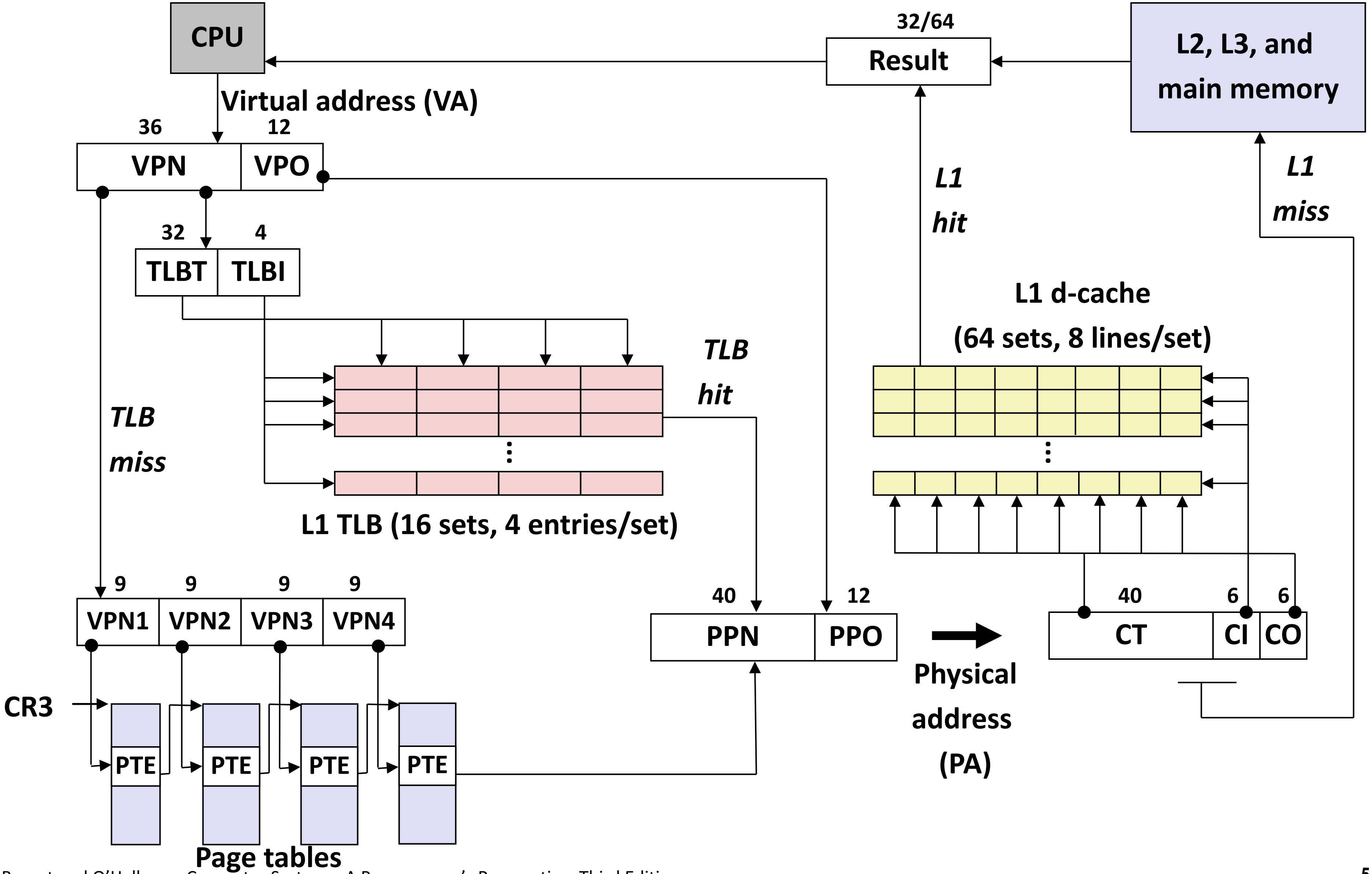
A: Reference bit (set by MMU on reads and writes, cleared by software)

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

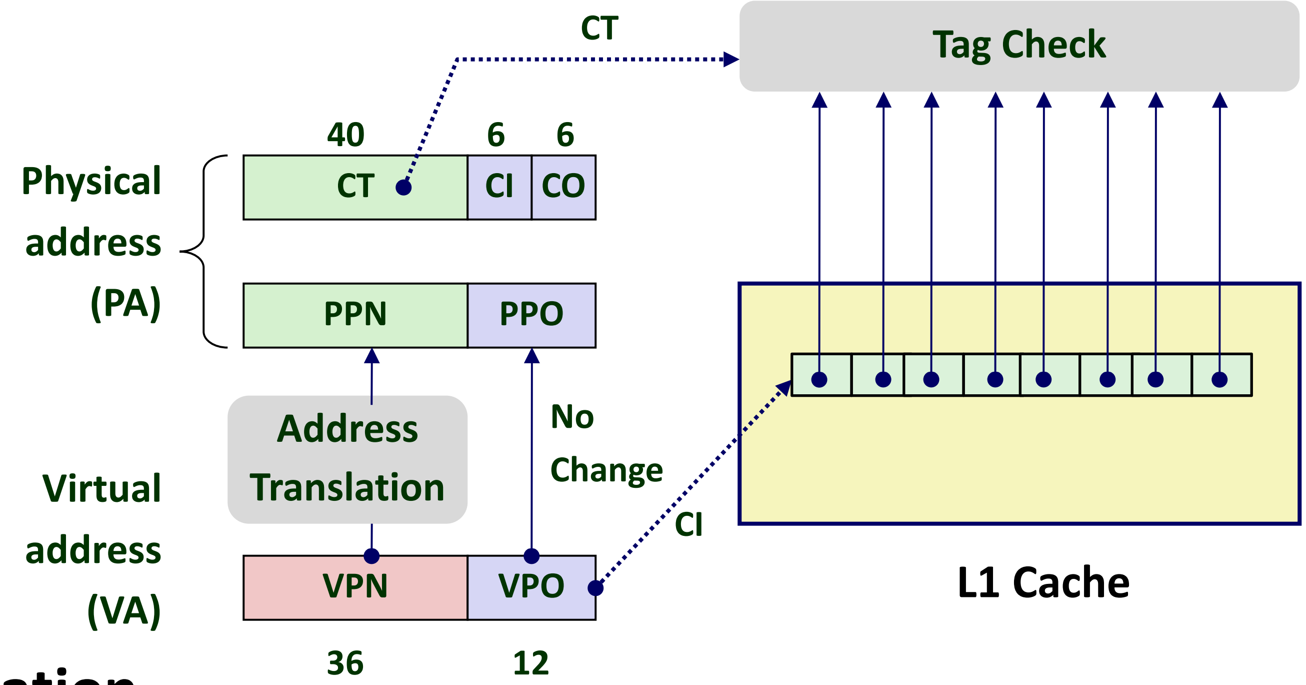
Page physical base address: 40 most significant bits of physical page address
(forces pages to be 4KB aligned)

XD: Disable or enable instruction fetches from this page.

End-to-end Core i7 Address Translation

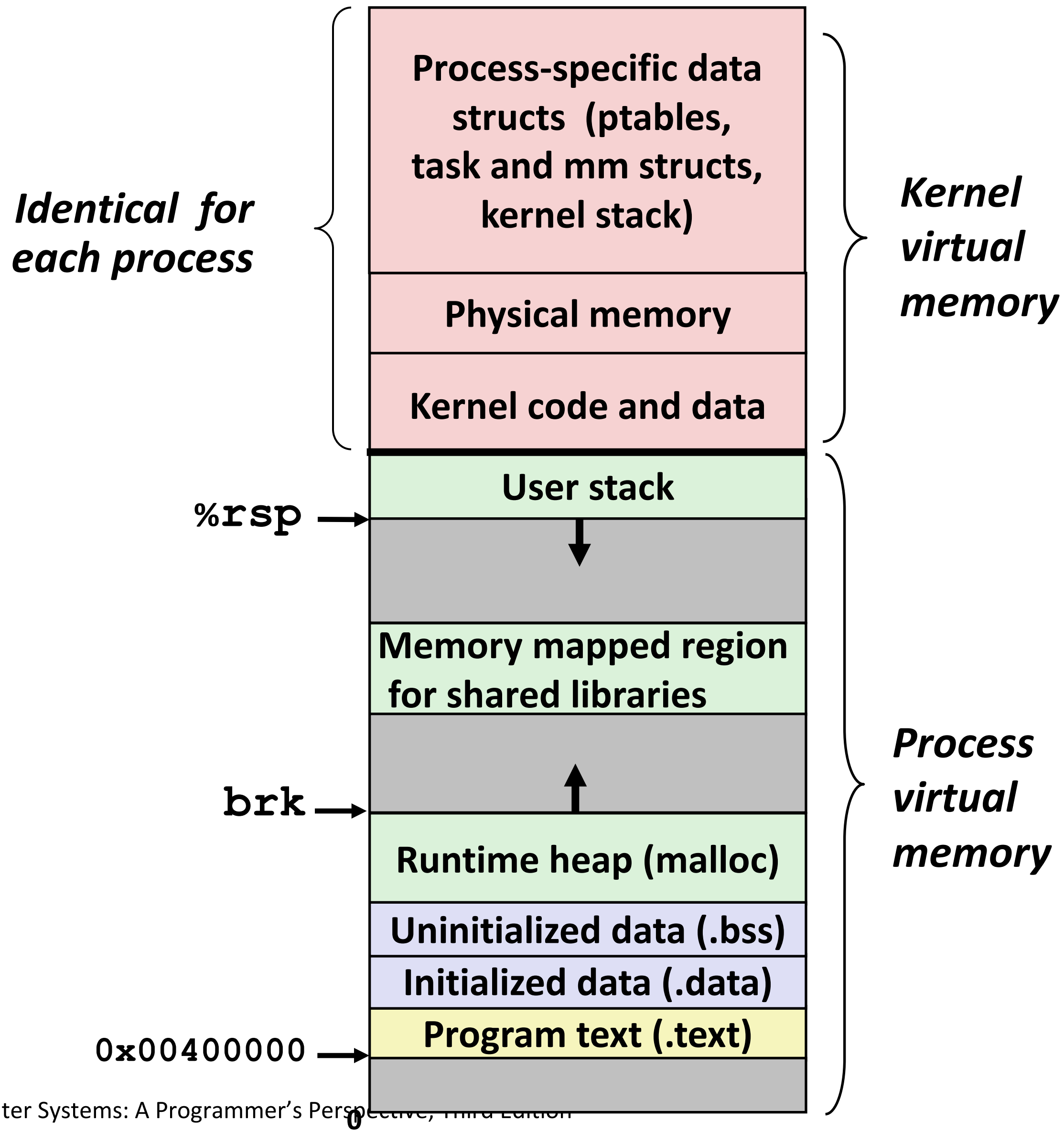


Cute Trick for Speeding Up L1 Access



- **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



Identical for each process

Process-specific data structs (ptables, task and mm structs, kernel stack)

Physical memory

Kernel code and data

Kernel virtual memory

User stack

`%rsp`

Memory mapped region for shared libraries

`brk`

Runtime heap (malloc)

Uninitialized data (.bss)

Initialized data (.data)

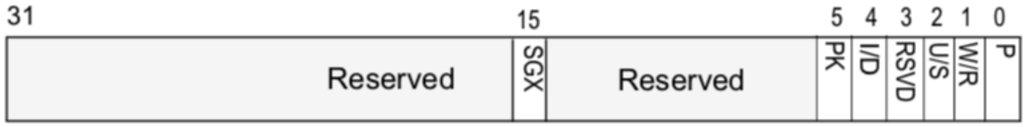
Program text (.text)

`0x00400000`

`0`

Process virtual memory

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