Operating Systems Practice

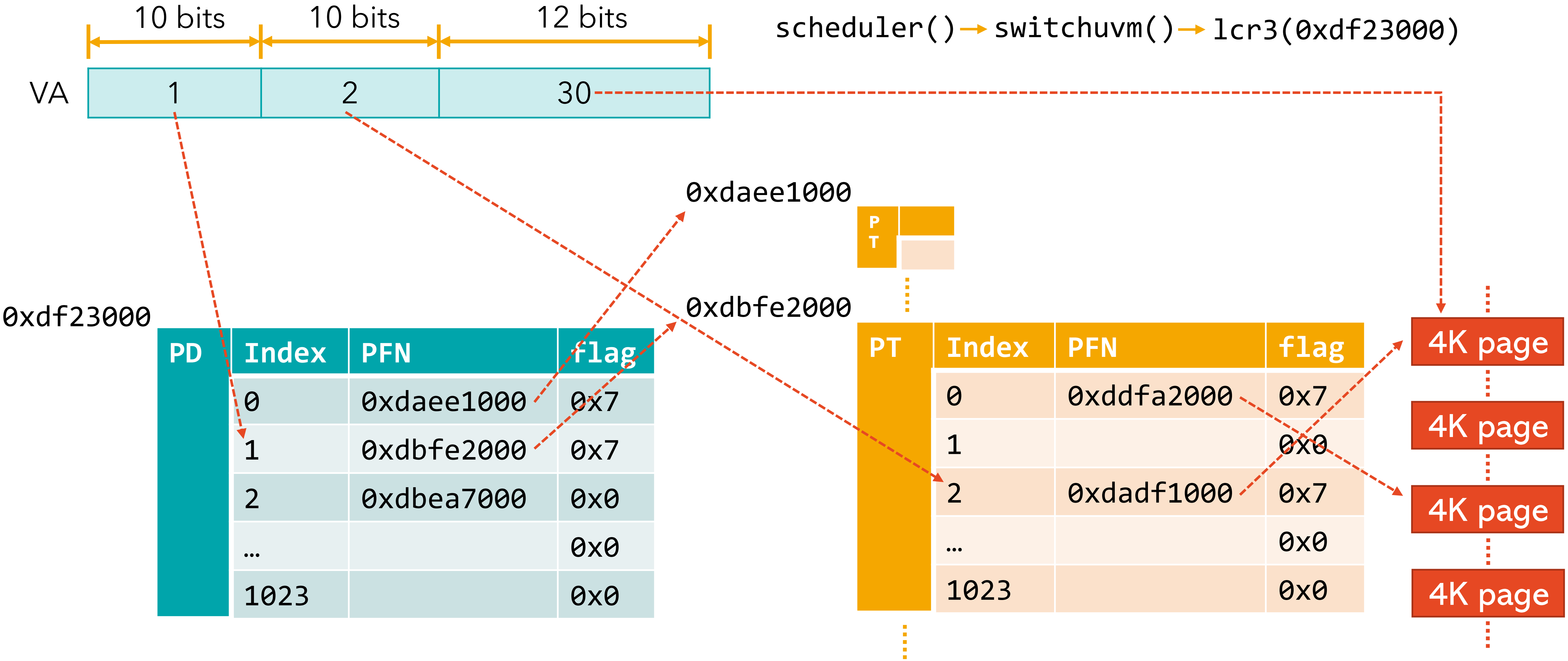
Project #2 - 3-level paging (Part 1)

Juhyung Park

arter97@dgist.ac.kr

Recall how VA is translated to PA (2-level page table)

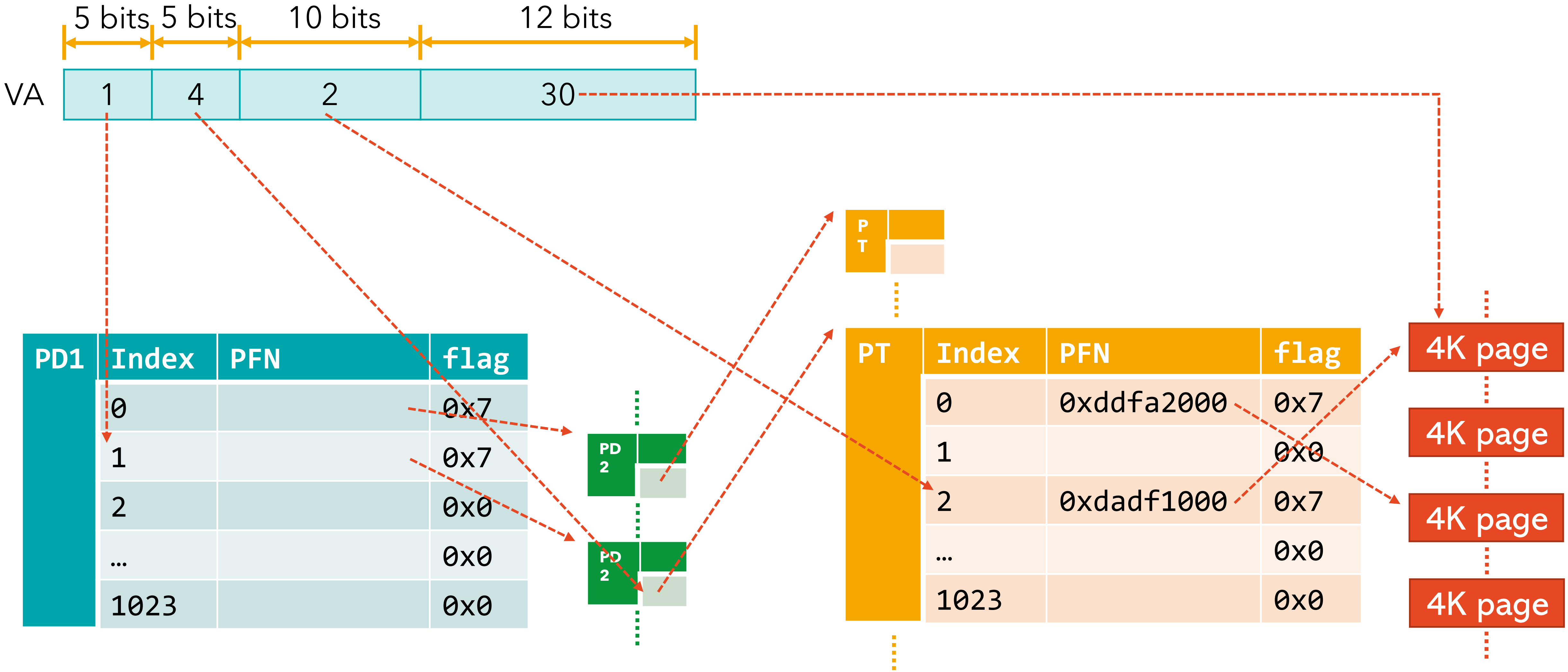
```
scheduler() → switchuvm() → lcr3(0xdf23000)
```



PD	Index	PFN	flag
	0	0xdaee1000	0x7
	1	0xdbfe2000	0x7
	2	0xdbea7000	0x0
	…		0x0
	1023		0x0

PT	Index	PFN	flag
	0	0xddfa2000	0x7
	1		0x0
	2	0xdadf1000	0x7
	…		0x0
	1023		0x0

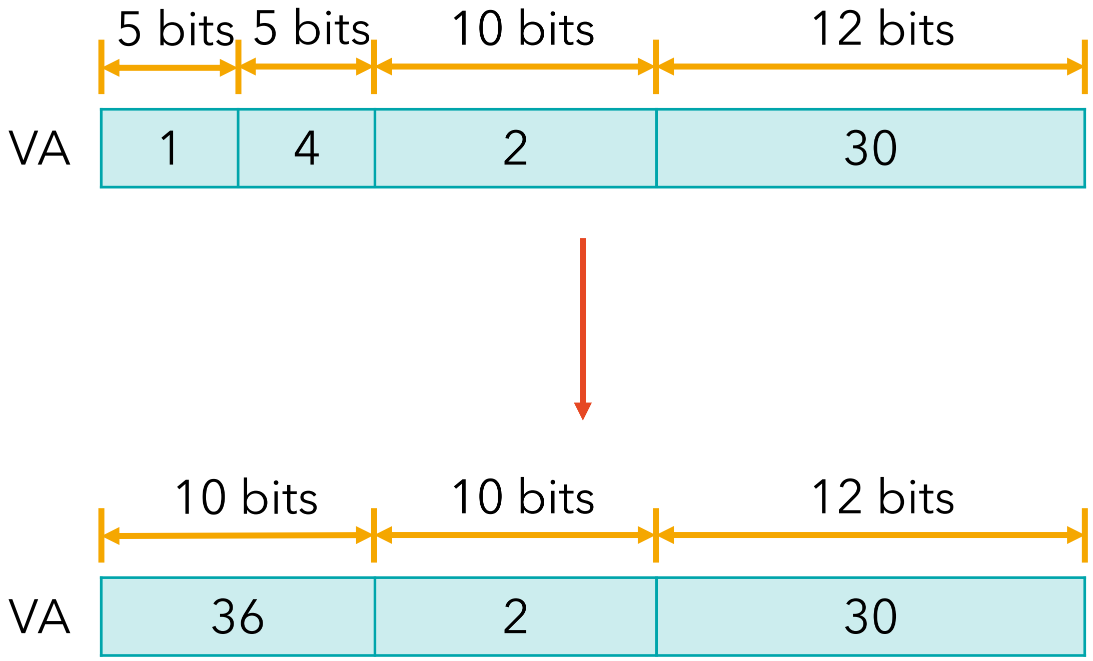
How can we customize VA to PA translation? (e.g., 3-level)



PD1	Index	PFN	flag
	0		0x7
	1		0x7
	2		0x0
	…		0x0
	1023		0x0

PT	Index	PFN	flag
	0	0xddfa2000	0x7
	1		0x0
	2	0xdadf1000	0x7
	…		0x0
	1023		0x0

x86-32 MMU still expects 2-level VA



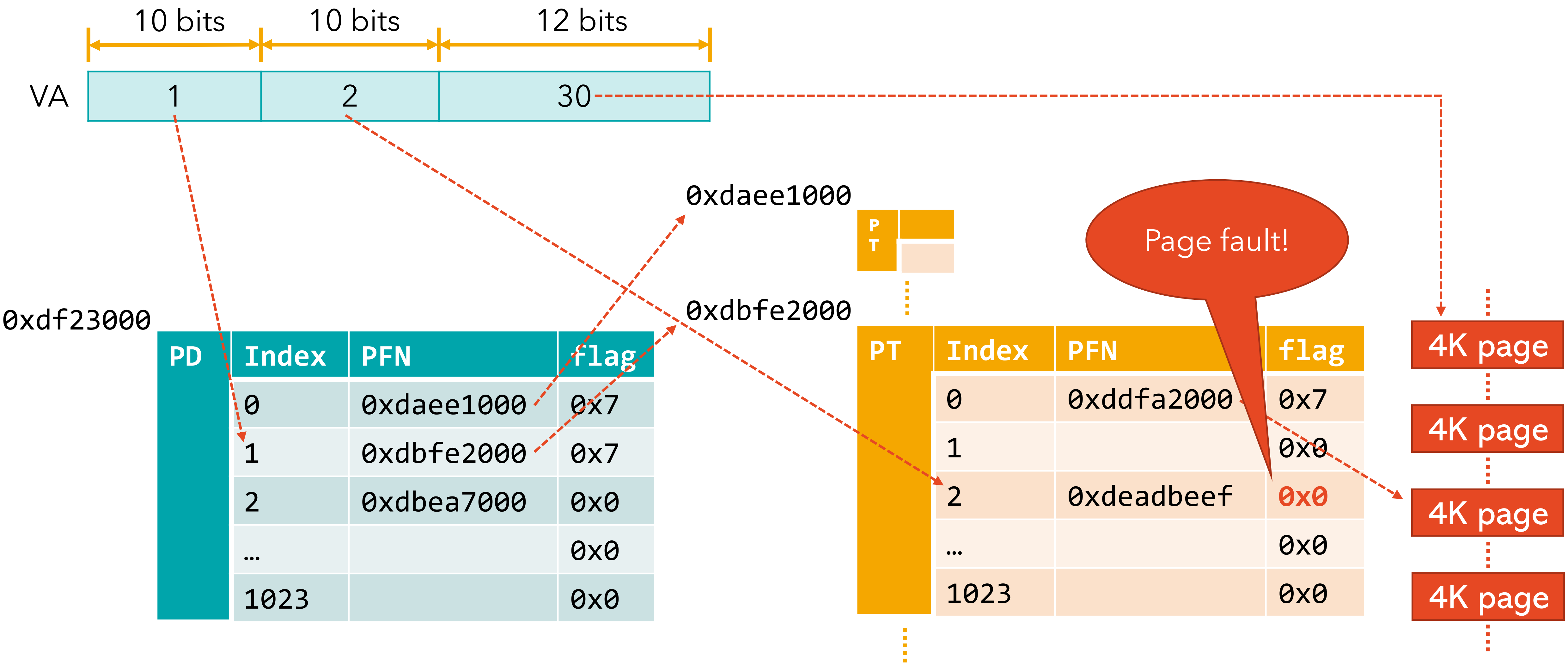
	5 bits	5 bits	10 bits	12 bits
VA	1	4	2	30

	10 bits	10 bits	12 bits
VA	36	2	30

Error!

How can we implement custom page addressing within software?

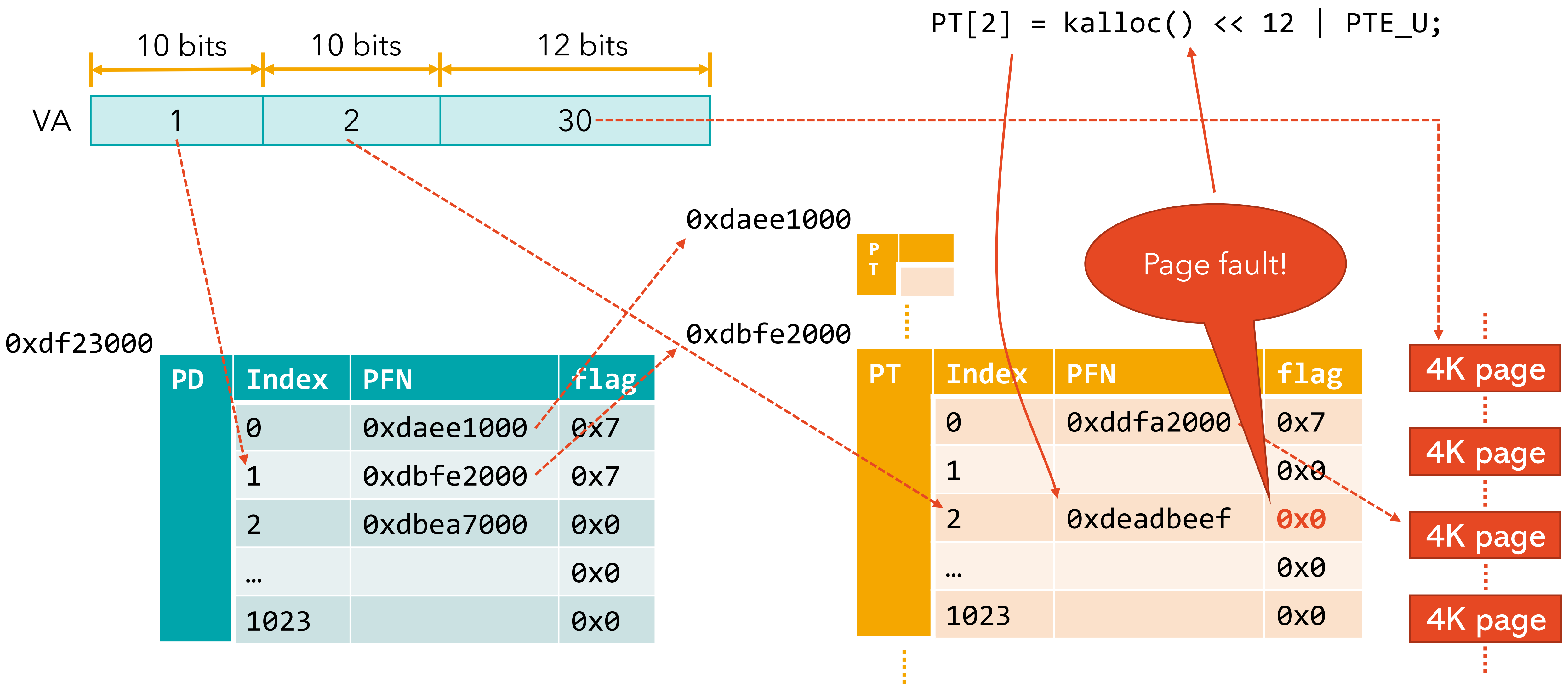
Page fault (1 - occurs when PTE_P flag is not set)



PD	Index	PFN	flag
	0	0xdaee1000	0x7
	1	0xdbfe2000	0x7
	2	0xdbea7000	0x0
	…		0x0
	1023		0x0

PT	Index	PFN	flag
	0	0xddfa2000	0x7
	1		0x0
	2	0xdeadbeef	0x0
	…		0x0
	1023		0x0

Page fault (2 - kernel's page fault handler is executed)

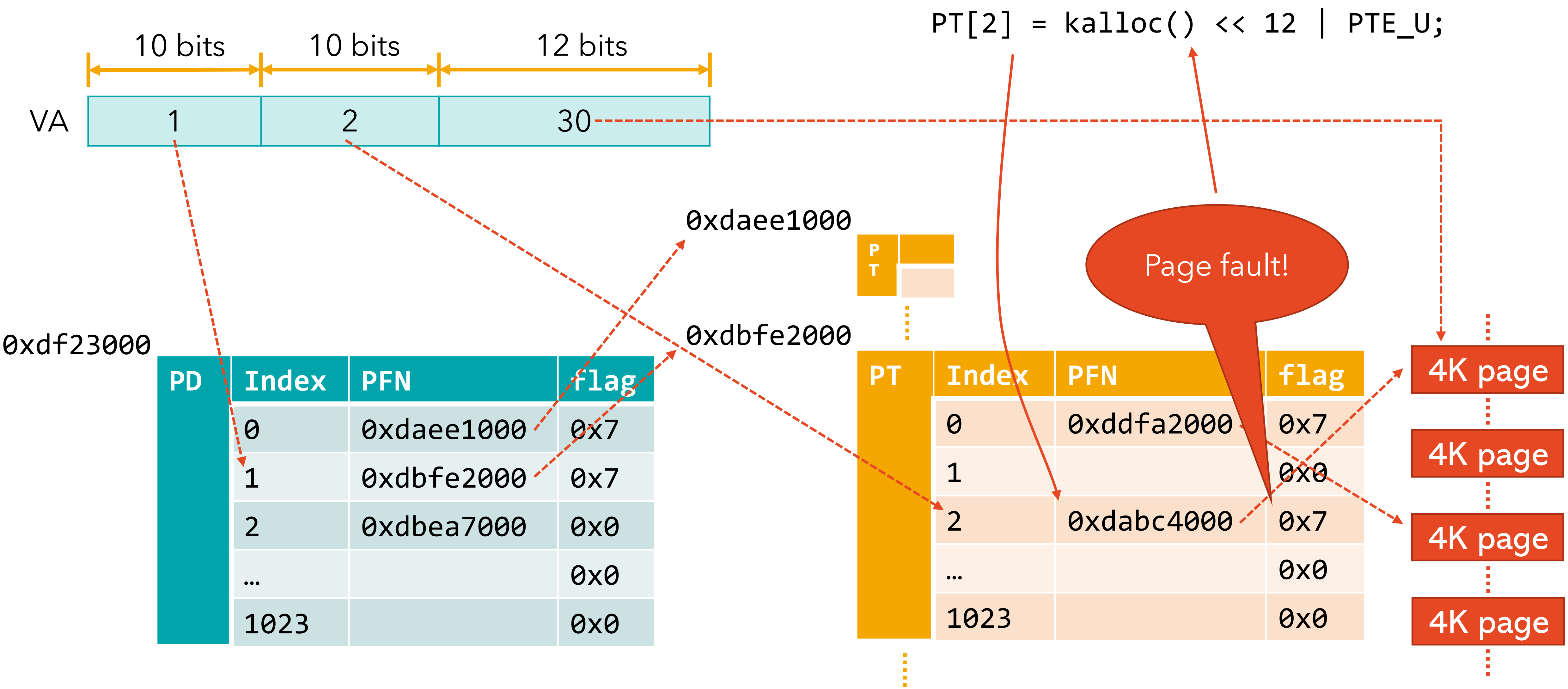


```
PT[2] = kalloc() << 12 | PTE_U;
```

PD	Index	PFN	flag
	0	0xdaee1000	0x7
	1	0xdbfe2000	0x7
	2	0xdbea7000	0x0
	…		0x0
	1023		0x0

PT	Index	PFN	flag
	0	0xddfa2000	0x7
	1		0x0
	2	0xdeadbeef	0x0
	…		0x0
	1023		0x0

Page fault (3 - code execution is resumed from page fault handler)

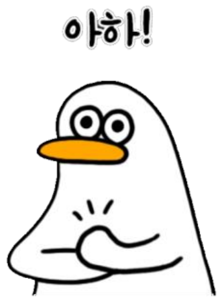


```
PT[2] = kalloc() << 12 | PTE_U;
```

PD	Index	PFN	flag
	0	0xdaee1000	0x7
	1	0xdbfe2000	0x7
	2	0xdbea7000	0x0
	…		0x0
	1023		0x0

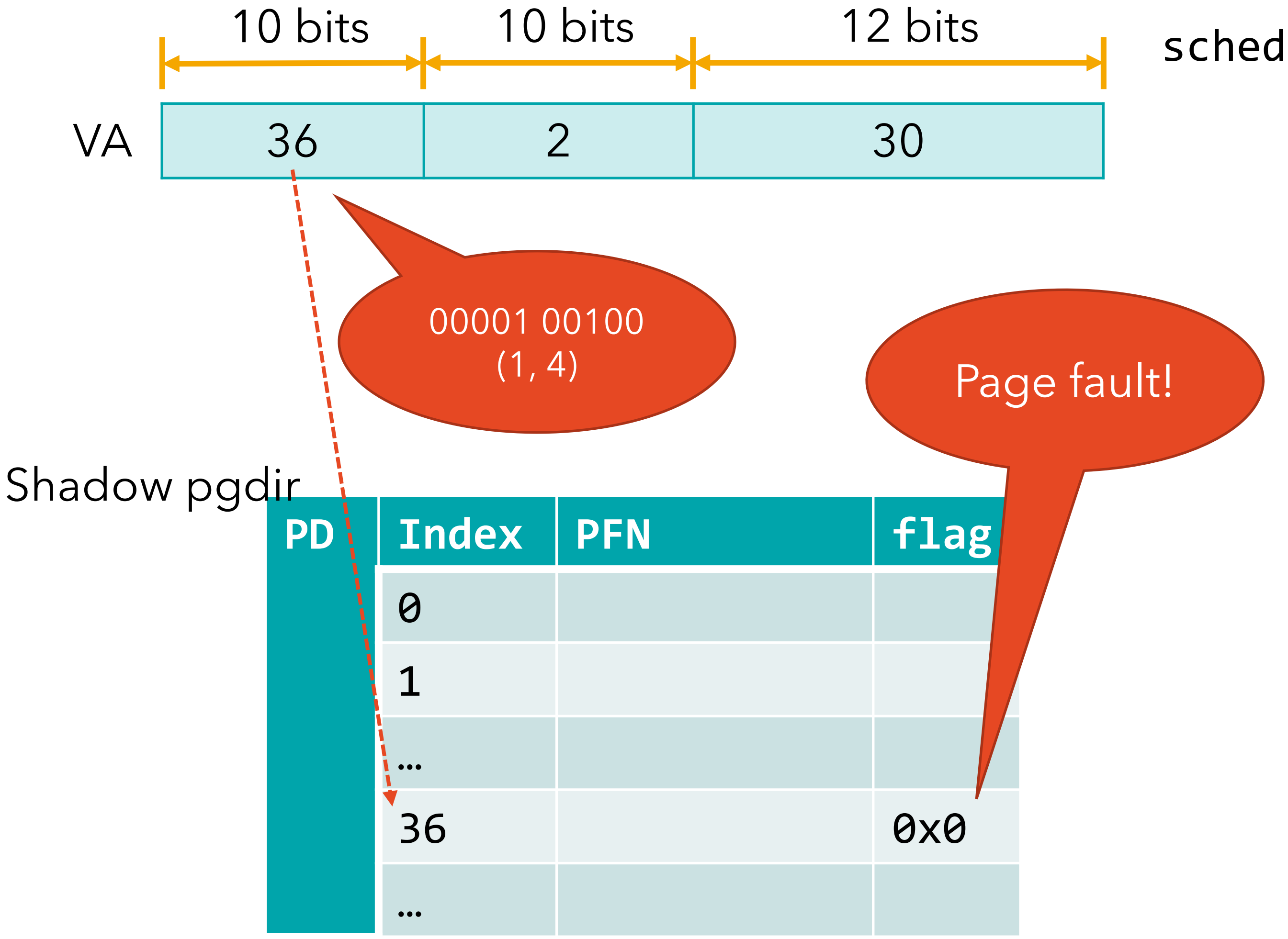
PT	Index	PFN	flag
	0	0xddfa2000	0x7
	1		0x0
	2	0xdabc4000	0x7
	…		0x0
	1023		0x0

Triggering page fault on every page access allows the software to fully redirect VA!



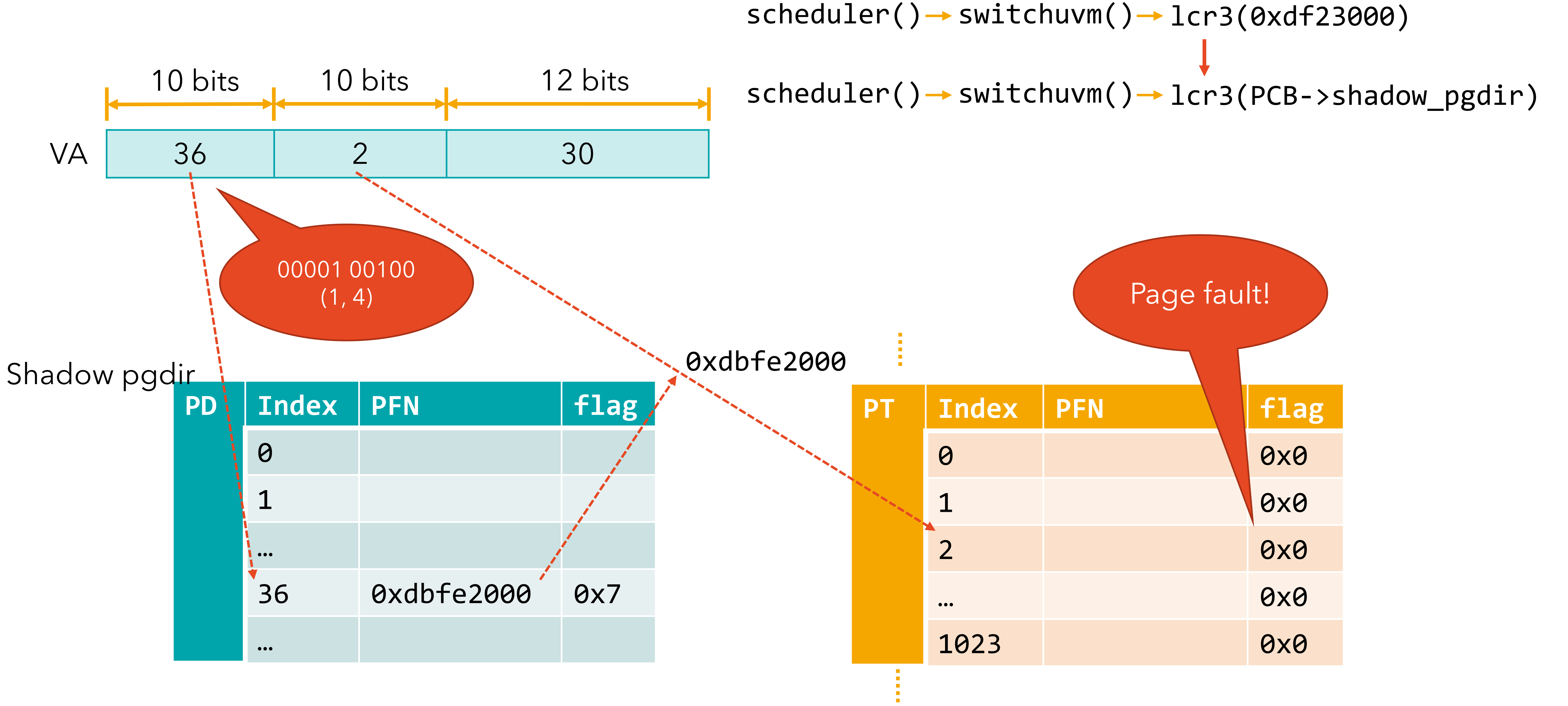
How?

```
scheduler() → switchuvm() → lcr3(0xdf23000)
                                    ↓
scheduler() → switchuvm() → lcr3(PCB->shadow_pgdir)
```



PD	Index	PFN	flag
	0		
	1		
	…		
	36		0x0
	…		

How?



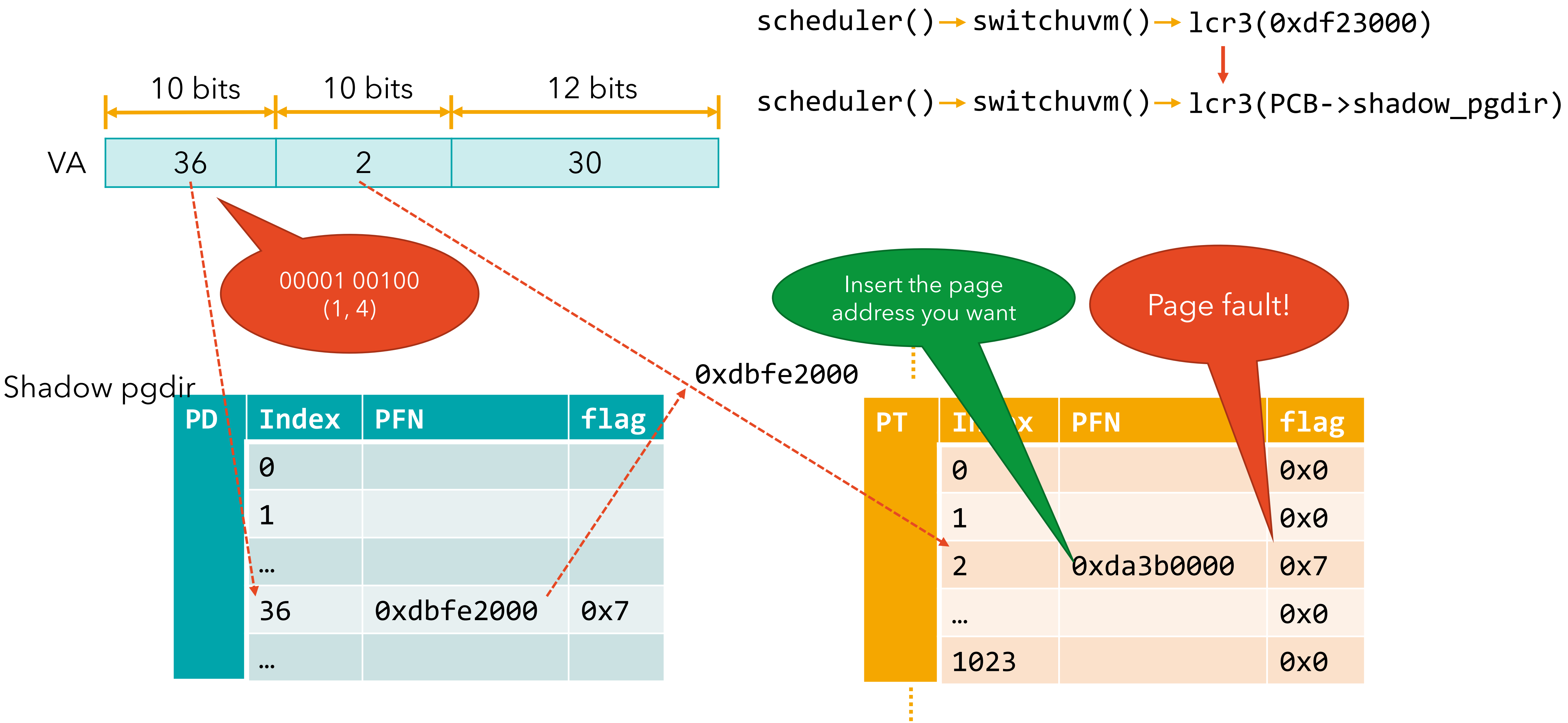
scheduler() → switchuvm() → lcr3(0xdf23000)

scheduler() → switchuvm() → lcr3(PCB->shadow_pgdir)

PD	Index	PFN	flag
	0		
	1		
	…		
	36	0xdbfe2000	0x7
	…		

PT	Index	PFN	flag
	0		0x0
	1		0x0
	2		0x0
	…		0x0
	1023		0x0

How?



scheduler()→switchuvm()→lcr3(0xdf23000)

scheduler()→switchuvm()→lcr3(PCB->shadow_pgdir)

PD	Index	PFN	flag
	0		
	1		
	…		
	36	0xdbfe2000	0x7
	…		

PT	Index	PFN	flag
	0		0x0
	1		0x0
	2	0xda3b0000	0x7
	…		0x0
	1023		0x0

Software TLB

- Trigger a page fault on every new page access
- Reference the vanilla pgdir and make a copy to shadow_pgdir

- We maintain 2-level paging in this project but the CPU's MMU must never access PCB->pgdir
  - PCB->shadow_pgdir must return the appropriate PA instead

Trigger a page fault on every new page access (1/2)

- Setting the CR3 register tells the hardware to use it as the current page directory address
  - When is switchuvm() called?
  - Can we abuse this to force page faults?
  - Hint: a new empty pgdir can be allocated with setupkvm();

```
vm.c:

// Switch TSS and h/w page table to correspond to process p.
void
switchuvm(struct proc *p)
{

  ...

  ltr(SEG_TSS << 3);
  lcr3(V2P(p->pgdir));  // switch to process's address space
  popcli();
}
```

Reference the vanilla pgdir and make a copy to shadow_pgdir (2/2)

- The CPU's MMU must never access PCB->pgdir
  - It should use PCB->shadow_pgdir
- You must copy PA from PCB->pgdir and set it up to PCB->shadow_pgdir
  - Recall what walkpgdir() function did in vm.c

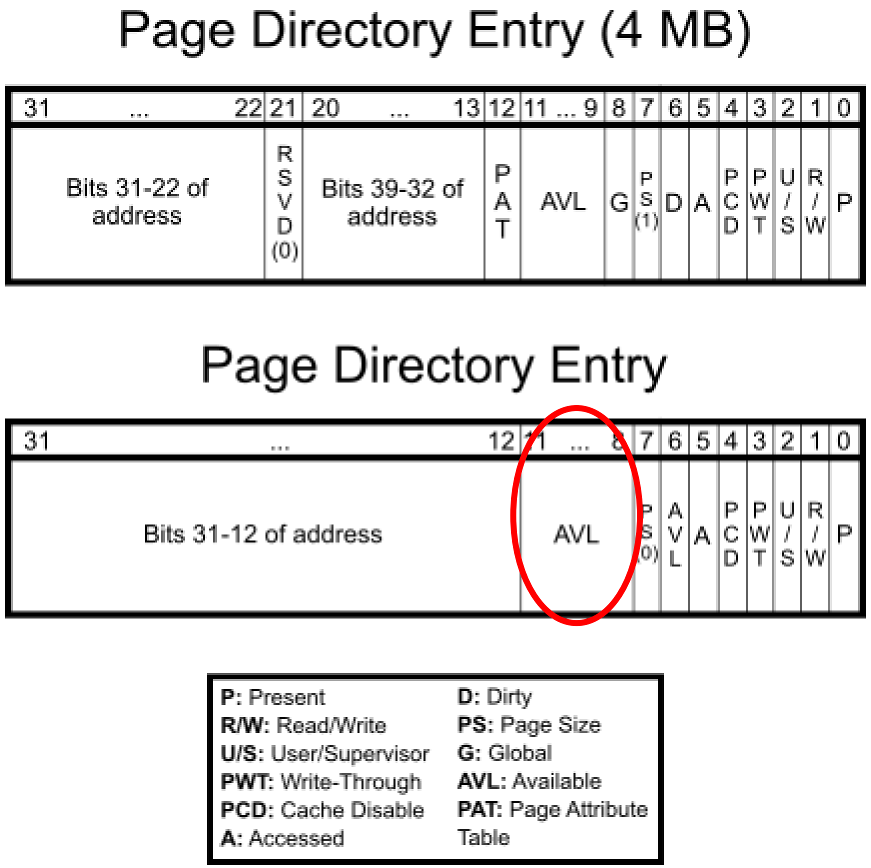
```
void pagefault(void)
{

  ...

  // Map pgdir's page address to shadow_pgdir's page table
  // XXX

  ...
}
```

Counting page accesses

- Once shadow_pgdir is set, page fault handler no longer runs if it was already handled before
- To count page accesses, shadow_pgdir must be cleared
- shadow_pgdir may contain addresses to page table itself
- Clearing all of shadow_pgdir will cause recursion loop during page fault

Handling sbrk'ed memory separately

- We use a reserved bit available on PDE



Page Directory Entry (4 MB)

Page Directory Entry

P: Present
R/W: Read/Write
U/S: User/Supervisor
PWT: Write-Through
PCD: Cache Disable
A: Accessed

D: Dirty
PS: Page Size
G: Global
AVL: Available
PAT: Page Attribute Table

```
#define PTE_SBRK        (1 << 10)  // sbrk'ed memory (malloc)
```

Handling sbrk’ed memory separately

- We use a reserved bit available on PDE
- malloc => sbrk => growproc => PTE_SBRK set!
- pagefault => proc->last_va => if PTE_SBRK is set, clear!

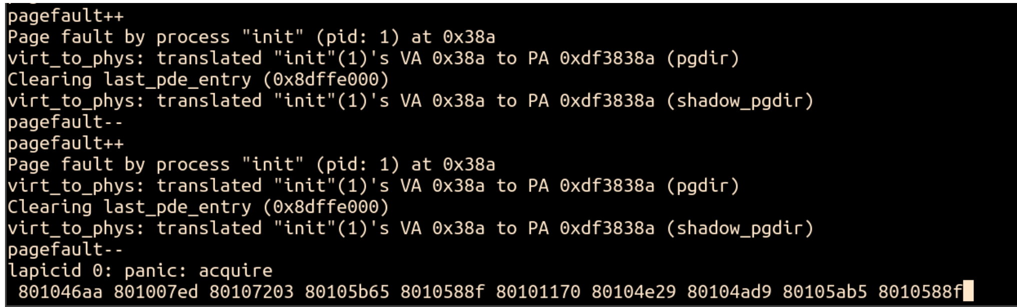
- This allows page fault handler to run again on the same memory address
    - Which in turn allows us to “count” its occurrence

- https://github.com/dgist-datalab/xv6/commit/27ce2ad

Tips

- Try to understand 2-level paging fully before writing actual code
- Be careful of pointer accesses
- Reference mmu.h for helper macros
  - P2V, PDX, PTX, PTE_ADDR, …
- Have a look at skeleton code's commit history
  - https://github.com/dgist-datalab/xv6/commits/page-counter

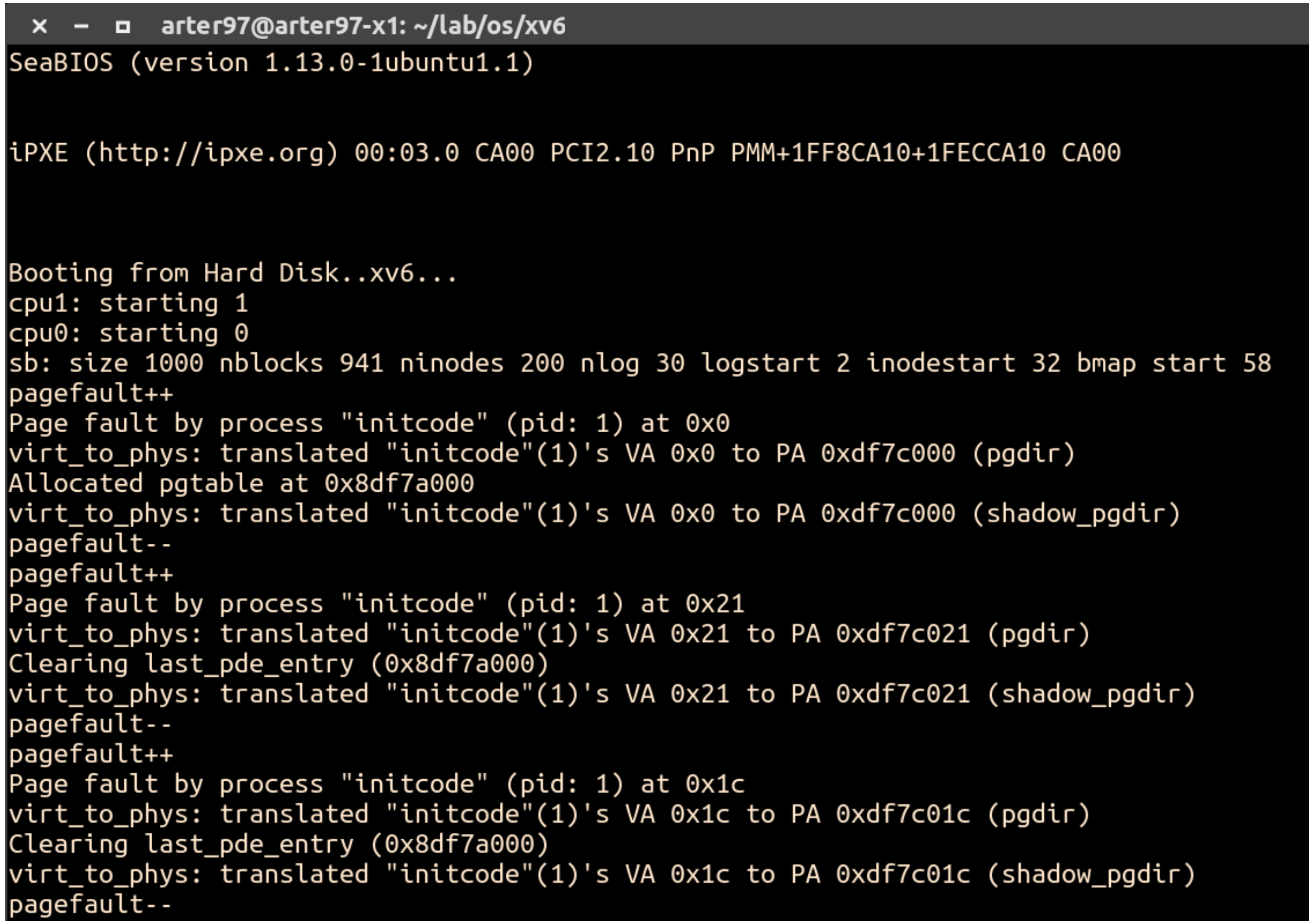
- cprintf() is buggy when called from page fault handler
  - You can set #define LOG 1 from vm.c to forcefully enable clprintf()
  - If a page fault is triggered during cprintf()…
  - It'll cause a crash during normal operations



```
pagefault++
Page fault by process "init" (pid: 1) at 0x38a
virt_to_phys: translated "init"(1)'s VA 0x38a to PA 0xdf3838a (pgdir)
Clearing last_pde_entry (0x8dffe000)
virt_to_phys: translated "init"(1)'s VA 0x38a to PA 0xdf3838a (shadow_pgdir)
pagefault--
pagefault++
Page fault by process "init" (pid: 1) at 0x38a
virt_to_phys: translated "init"(1)'s VA 0x38a to PA 0xdf3838a (pgdir)
Clearing last_pde_entry (0x8dffe000)
virt_to_phys: translated "init"(1)'s VA 0x38a to PA 0xdf3838a (shadow_pgdir)
pagefault--
lapicid 0: panic: acquire
 801046aa 801007ed 80107203 80105b65 8010588f 80101170 80104e29 80104ad9 80105ab5 8010588f
```

In action

- `#define LOG 1` (vm.c)



```
SeaBIOS (version 1.13.0-1ubuntu1.1)


iPXE (http://ipxe.org) 00:03.0 CA00 PCI2.10 PnP PMM+1FF8CA10+1FECCA10 CA00



Booting from Hard Disk..xv6...
cpu1: starting 1
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap start 58
pagefault++
Page fault by process "initcode" (pid: 1) at 0x0
virt_to_phys: translated "initcode"(1)'s VA 0x0 to PA 0xdf7c000 (pgdir)
Allocated pgtable at 0x8df7a000
virt_to_phys: translated "initcode"(1)'s VA 0x0 to PA 0xdf7c000 (shadow_pgdir)
pagefault--
pagefault++
Page fault by process "initcode" (pid: 1) at 0x21
virt_to_phys: translated "initcode"(1)'s VA 0x21 to PA 0xdf7c021 (pgdir)
Clearing last_pde_entry (0x8df7a000)
virt_to_phys: translated "initcode"(1)'s VA 0x21 to PA 0xdf7c021 (shadow_pgdir)
pagefault--
pagefault++
Page fault by process "initcode" (pid: 1) at 0x1c
virt_to_phys: translated "initcode"(1)'s VA 0x1c to PA 0xdf7c01c (pgdir)
Clearing last_pde_entry (0x8df7a000)
virt_to_phys: translated "initcode"(1)'s VA 0x1c to PA 0xdf7c01c (shadow_pgdir)
pagefault--
```

Extra points

- If you found a mistake in the skeleton code
- If you found a different/better method
- Kernel memory leak test has been disabled in usertests
  - https://github.com/dgist-datalab/xv6/commit/63776c9cc
  - If you manage to pass this test (after reverting the commit)
- Write a report for extra points

Finally …

<u>Do NOT hesitate</u> to ask questions!

Thank You!