Virtual Memory

15-213: Introduction to Computer Systems
Recitation 10: Oct. 28, 2013

Marjorie Carlson
Recitation A

Agenda

- **Shell Lab FAQs**
- **Malloc Lab Preview**
- **Virtual Memory Concepts**
- **Address Translation**
  - Basic
  - TLB
  - Multilevel

Updates

- **Shell Lab is due Tuesday (tomorrow), 11:59 p.m.**
- **Malloc Lab is out Tuesday (tomorrow), 11:59 p.m.**
  - Due Thursday, Nov. 14
  - Start early!!
  - "The total time you spend designing and debugging can easily eclipse the time you spend coding."

Shell Lab FAQ

- **"The traces behave differently from command-line input!"**
  - Some people are confused to find /bin/echo on their jobs list after running some trace files.
  - Some traces (e.g. trace05) print what they're running before they run them. They do this by using /bin/echo.
  - So if you see a mysterious /bin/echo show up on your jobs list, you shouldn't wonder *why it got on your jobs list*, you should wonder *why it never got deleted.*
  - Moral of the story: open the trace file and see what it does!

Shell Lab FAQ

- **Sigsuspend???**
  - You can only use waitpid() once, but there are probably two places you probably need to reap children (one for foreground jobs, one for background jobs).
  - Temptation: use waitpid() for background jobs; use `sleep()` or a tight loop (i.e., `while(1) {}`).
  - *Correct* solution: use sigsuspend to block your process until a signal arrives.
- **`int sigsuspend(const sigset_t *mask)`**
  - *Temporarily* replaces the process's signal mask with `mask`, which should be the signals you **don't** want to be interrupted by.
  - `sigsuspend` will return after an **un**blocked signal is received and its handler run. When it returns, it automatically reverts the process signal mask to its old value.

Shell Lab FAQ: sigsuspend example

```
int main() {
    sigset_t waitmask, newmask, oldmask;

    /* set waitmask with everything except SIGINT */
    sigfillset(&waitmask);
    sigdelset(&waitmask, SIGINT);

    /* set newmask with only SIGINT */
    sigemptyset(&newmask);
    sigaddset(&newmask, SIGINT);

    if (sigprocmask(SIG_BLOCK, &newmask, &oldmask) < 0) //oldmask now stores prev mask
        unix_error("SIG_BLOCK error");
    /* CRITICAL REGION OF CODE (SIGINT blocked) */
    /* pause, allowing ONLY SIGINT */
    if (sigsuspend(&waitmask) != -1)
        unix_error("sigsuspend error");

    /* RETURN FROM SIGSUSPEND (returns to signal state from before sigsuspend) */
    /* Reset signal mask which unblocks SIGINT */
    if (sigprocmask(SIG_SETMASK, &oldmask, NULL) < 0)
        unix_error("SIG_SETMASK error");
}
```

Malloc Lab Sneak Preview

- **You will write your own dynamic storage allocator – i.e., your own `malloc`, `free`, `realloc`, `calloc`.**
- **This week in class, you will learn about different ways to keep track of free and allocated blocks of memory.**
  - Implicit linked list of blocks.
  - Explicit linked list of *free* blocks.
  - Segregated lists of different *size* free blocks.
- **Other design decisions:**
  - How will you look for free blocks? (First fit, next fit, best fit…)
  - Should the linked lists be doubly linked?
  - When do you coalesce blocks?
- **This is exactly what you'll do in this lab, so pay lots of attention in class. ☺**

Malloc Lab Sneak Preview

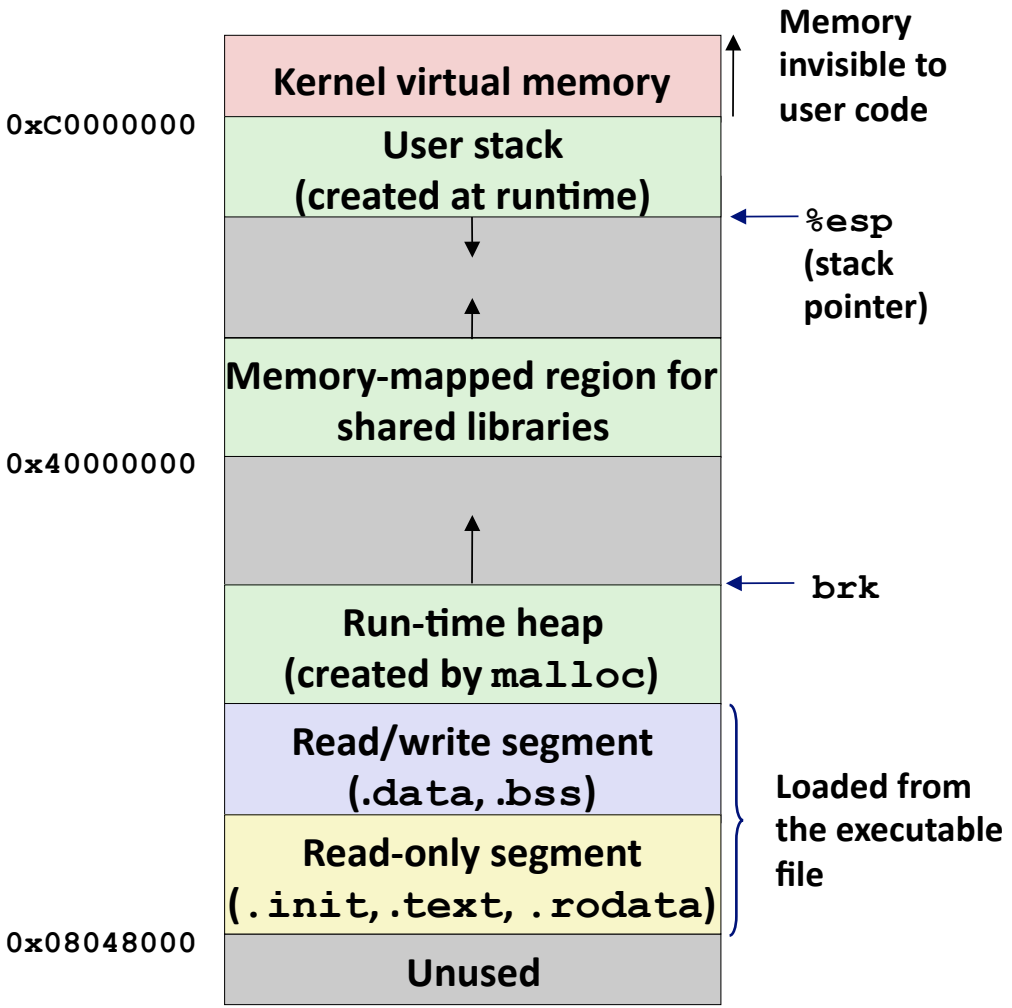
- **If you haven't been using version control so far, this is a good time to start.**
- **Workflow:**
  - Implement indirect linked lists. Make sure it works.
  - Implement explicit linked lists. Make sure it still works.
  - Implement segregated lists. Make sure it still works.
  - You WILL break things and need to revert.
- **Barebones guide to using git on the Shark Machines:**
  - `git init` starts a local repository.
  - `git add foo.c` adds foo.c to that repository.
  - `git commit -a –m 'Describe changes here'` updates your repository with the current state of all files you've added.

Agenda

- Shell Lab FAQs
- Malloc Lab Sneak Preview
- **Virtual Memory Concepts**
- Address Translation
  - Basic
  - TLB
  - Multilevel

Virtual Memory Concepts

- We've been viewing memory as a linear array.
- But wait! If you're running 5 processes with stacks at `0xC0000000`, don't their addresses conflict?
- Nope! Each process has its own address space.
- How???



Virtual memory concepts

- We define a mapping from the **virtual** address used by the process to the actual **physical** address of the data in memory.

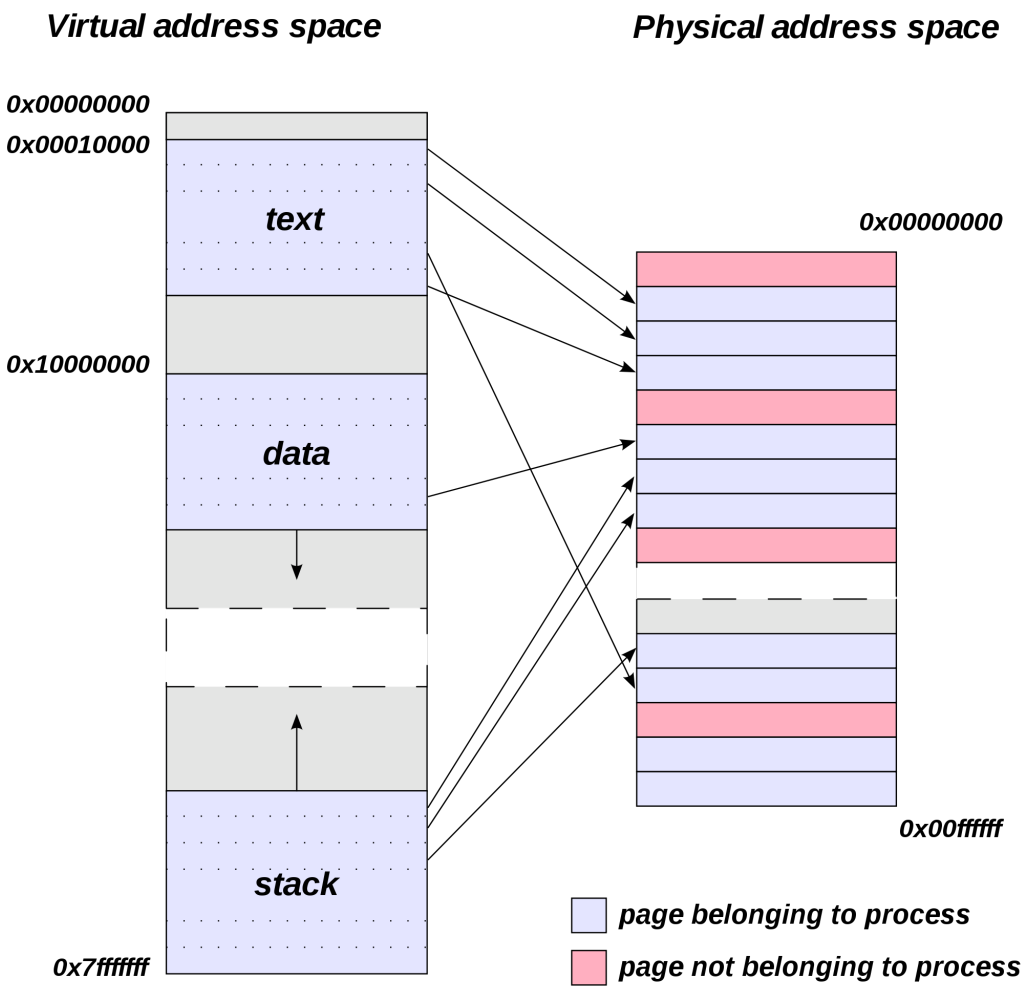
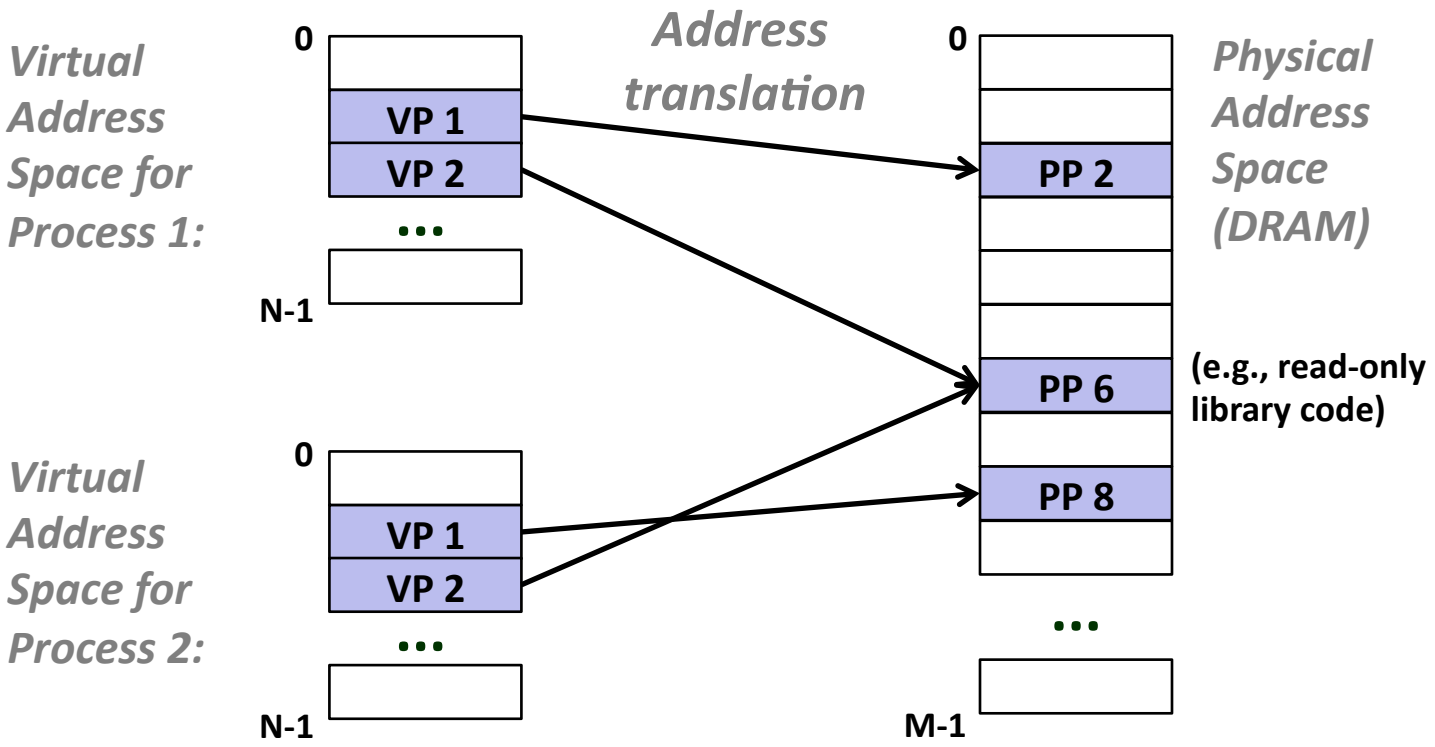


Image: http://en.wikipedia.org/wiki/File:Virtual_address_space_and_physical_address_space_relationship.svg

Virtual memory concepts

This explains why two different processes can use the same address. It also lets them share data *and* protects their data from illegal accesses. Hooray for virtual memory!



Virtual memory concepts

- **Page table**
  - Lets us look up the physical address corresponding to any virtual address. (Array of physical addresses, indexed by virtual address.)
- **TLB (Translation Lookaside Buffer)**
  - A special tiny cache just for page table entries.
  - Speeds up translation.
- **Multi-level page tables**
  - The address space is often sparse.
  - Use page directory to map large chunks of memory to a page table.
  - Mark large unmapped regions as non-present in page directory instead of storing page tables full of invalid entries.

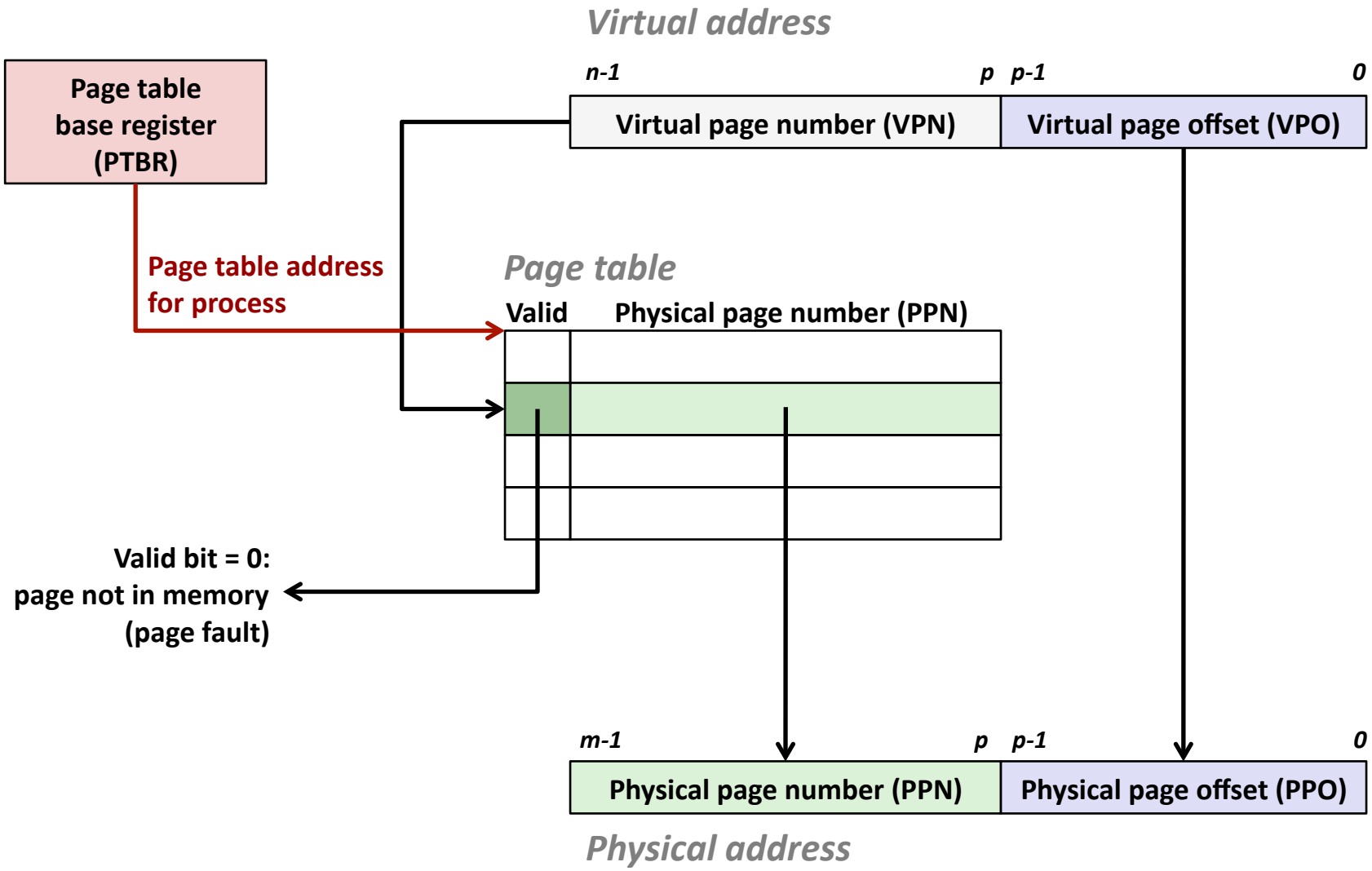
Agenda

- Shell Lab FAQs
- Malloc Lab Sneak Preview
- Virtual Memory Concepts
- **Address Translation**
  - Basic
  - TLB
  - Multilevel

VM Address Translation

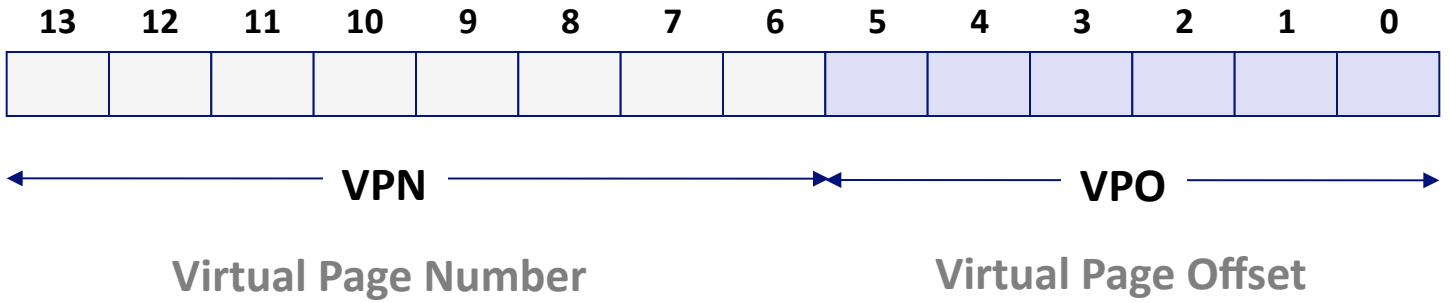
- **Virtual Address Space**
  - *V = {0, 1, …, N–1}*
  - There are N possible virtual addresses.
  - Virtual addresses are n bits long; $2^n = N$.
- **Physical Address Space**
  - *P = {0, 1, …, M–1}*
  - There are M possible physical addresses.
  - Virtual addresses are m bits long; $2^m = M$.
- **Memory is grouped into "pages."**
  - Page size is P bytes.
  - The address offset is p bytes; $2^p = P$.
  - Since the virtual offset (VPO) and physical offset (PPO) are the same, the offset doesn't need to be translated.

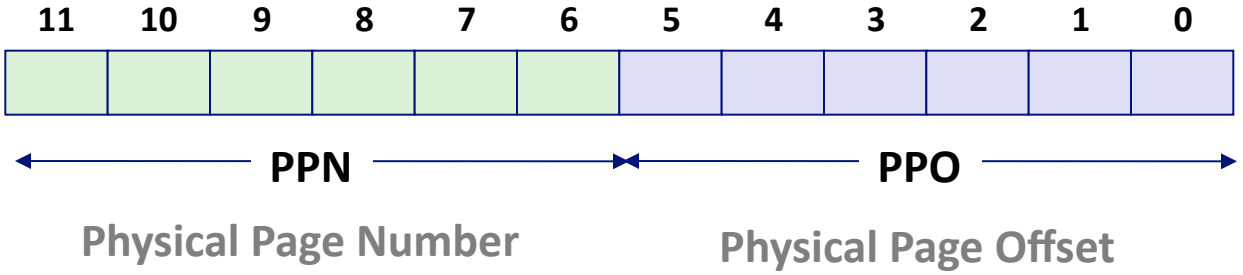
VM Address Translation



VM Address Translation

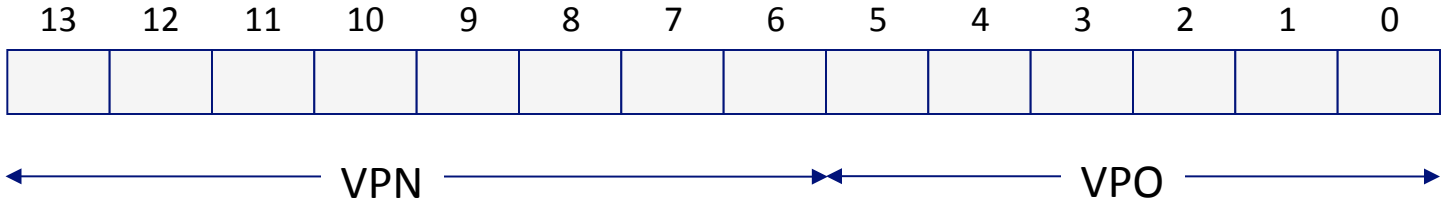
- **Addressing**
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes





Example 1: Address Translation

- Pages are 64 bytes. How many bits is the offset?
- Find `0x03D4`.

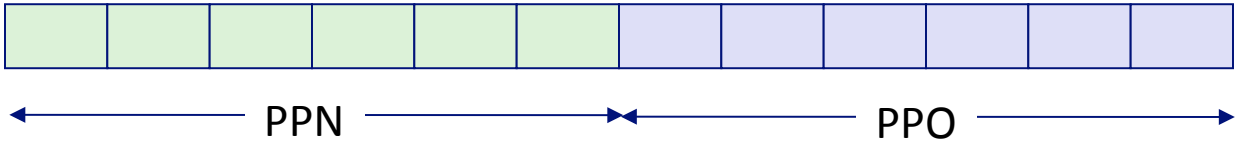


- VPN: _____
- PPN: _____
- Physical address:

_____

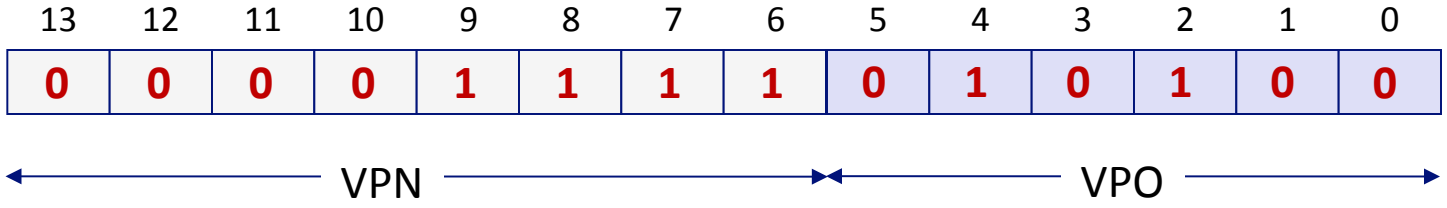
VPN	PPN	Valid
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

VPN	PPN	Valid
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1



Example 1: Address Translation

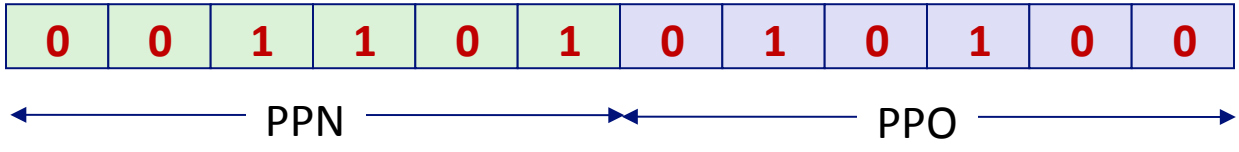
- Pages are 64 bytes. How many bits is the offset? $\log_2 64 = 6$
- Find 0x03D4.



- VPN: 0x0F
- PPN: 0x0D
- Physical address: 0x0354

VPN	PPN	Valid
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

VPN	PPN	Valid
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1

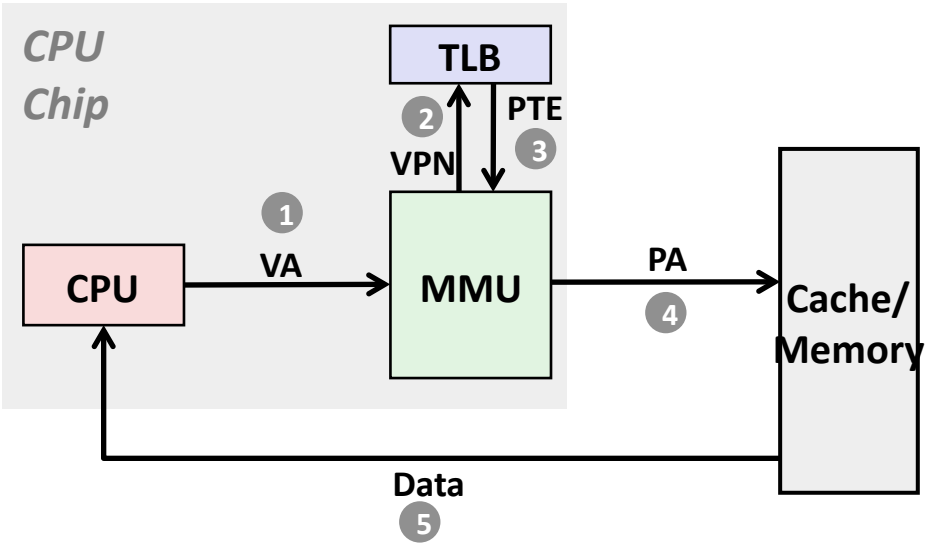


Agenda

- Shell Lab FAQs
- Malloc Lab Sneak Preview
- Virtual Memory Concepts
- **Address Translation**
  - Basic
  - TLB
  - Multilevel

VM Address Translation with TLB

- **That's nice and simple, but it doubles memory usage.**
  - One memory access to look in the page table.
  - One memory access of the actual memory we're looking for.
- **Solution:**
  - Cache the most frequently used page table entries in the TLB.
  - To look up a virtual address in the TLB, split up the VPN (not the whole virtual address!) into a TLB index and a TLB tag.



Example 2: Address Translation with TLB

1 MB of virtual memory

256 KB of physical memory

4 KB page size

TLB: 8 entries, 2-way set associative

- **How many bits are needed to represent the virtual address space?**
- **How many bits are needed to represent the physical address space?**
- **How many bits are needed to represent the offset?**

- **How many bits are needed to represent VPN?**
- **How many bits are in the TLB index?**
- **How many bits are in the TLB tag?**

Example 2: Address Translation with TLB

1 MB of virtual memory

256 KB of physical memory

4 KB page size

TLB: 8 entries, 2-way set associative

- **How many bits are needed to represent the virtual address space? 20. (1 MB = 2^{20} bytes.)**
- **How many bits are needed to represent the physical address space? 18. (256 KB = 2^{18} bytes.)**
- **How many bits are needed to represent the offset? 12. (4 KB = 2^{12} bytes.)**
- **How many bits are needed to represent VPN? 8. (20-12.)**
- **How many bits are in the TLB index? 2. (4 sets = 2^2 set bits.)**
- **How many bits are in the TLB tag? 6. (8-2.)**

Example 2a: Address Translation with TLB

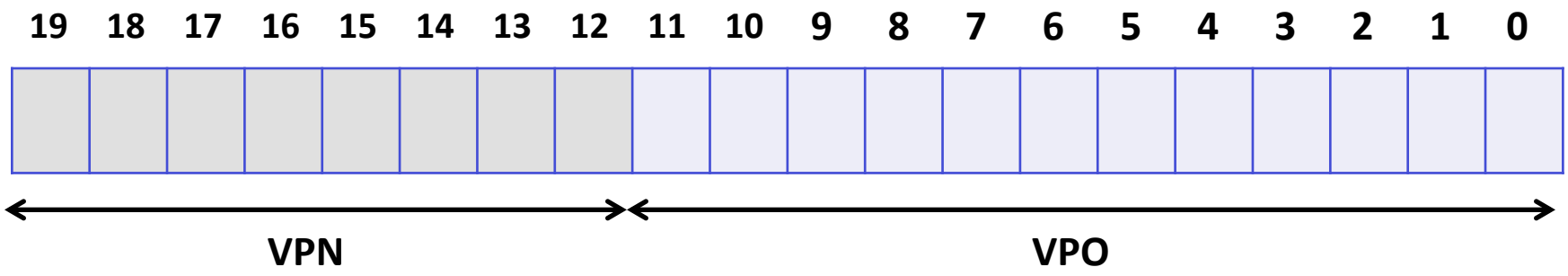
- Translate 0x14213, given the contents of TLB and the first 32 entries of the page table below. (All the numbers are in hexadecimal.)

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

Example 2a: Address Translation with TLB

0x14213



VPN:

TLBI:

TLBT:

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

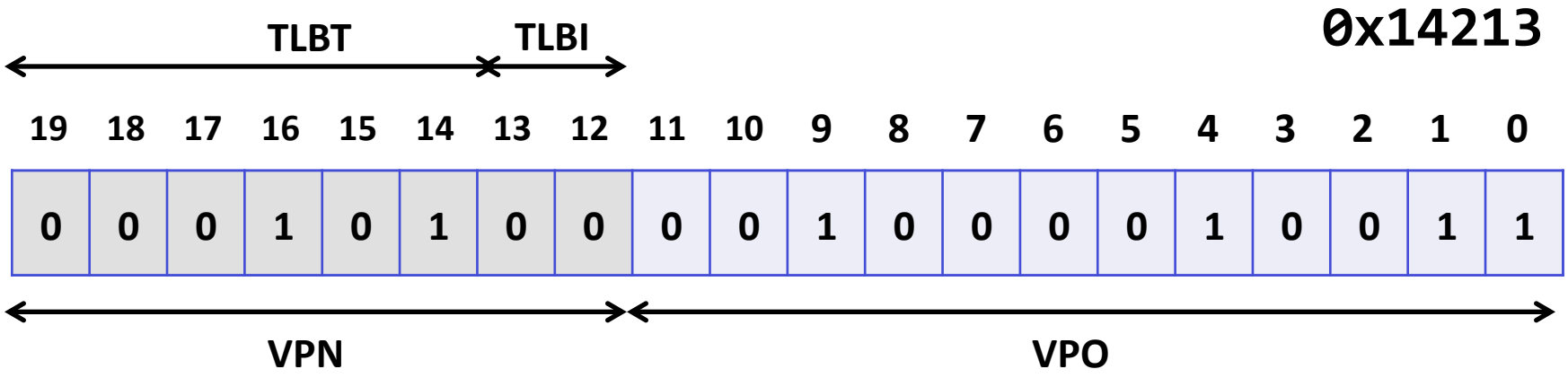
TLB Hit!

PPN:

Offset:

Physical address:

Example 2a: Address Translation with TLB



VPN: 0x14
TLBI: 0x00
TLBT: 0x05

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

TLB Hit!
PPN: 0x13
Offset: 0x213

Physical address: 0x13213

Example 2b: Address Translation with TLB

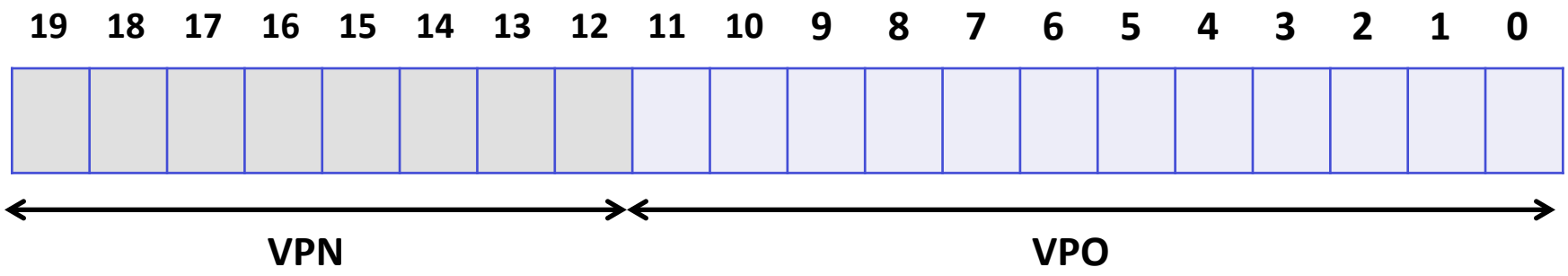
- Translate `0x1F213`, given the contents of TLB and the first 32 entries of the page table below.

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

Example 2b: Address Translation with TLB

0x1F213



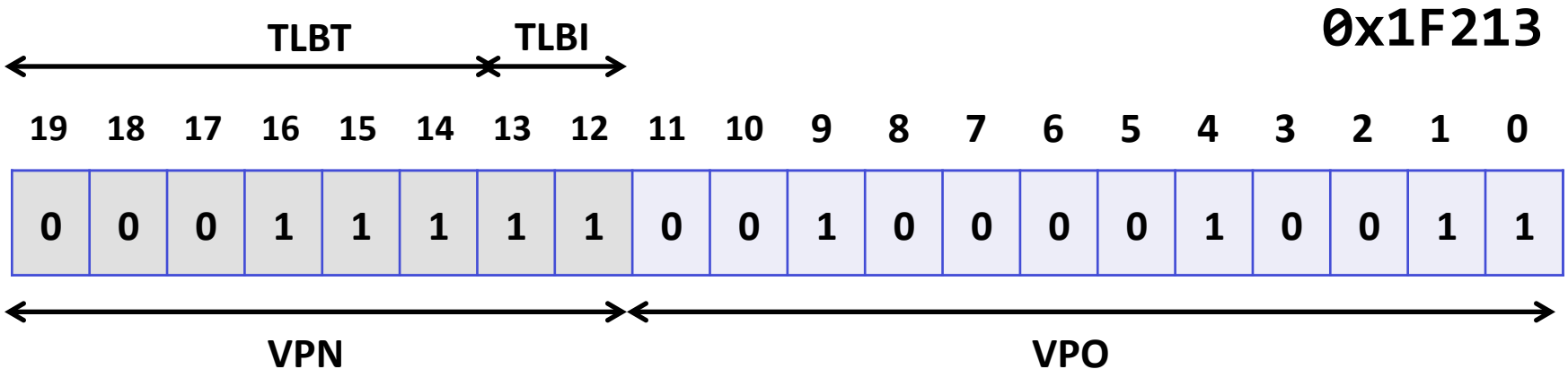
VPN:

TLBI:

TLBT:

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

Example 2b: Address Translation with TLB



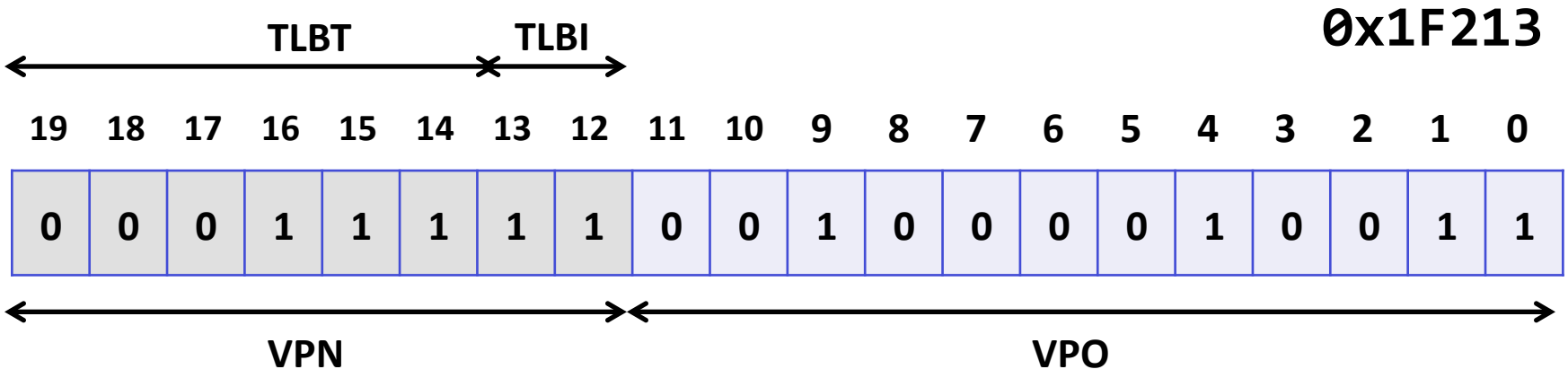
VPN: 0x1F
TLBI: 0x03
TLBT: 0x07

TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

TLB Miss!

Step 2: look it up in the page table. ☹

Example 2b: Address Translation with TLB



VPN: 0x1F
TLBI: 0x03
TLBT: 0x07

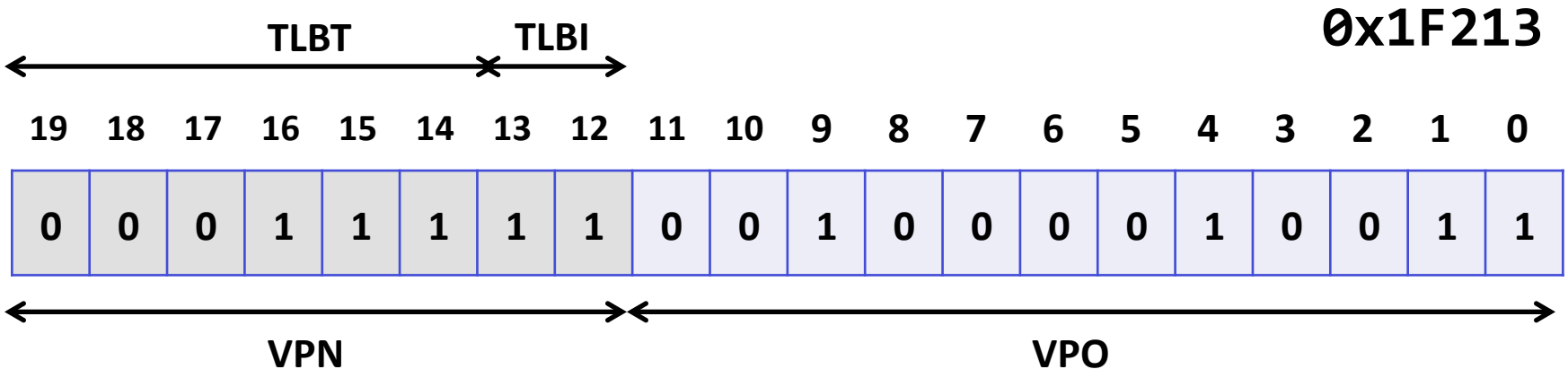
Page Table

VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

Page Table Hit
PPN:
Offset:

Physical address:

Example 2b: Address Translation with TLB



VPN: 0x1F
TLBI: 0x03
TLBT: 0x07

Page Table

VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

Page Table Hit
PPN: 0x15
Offset: 0x213

Physical address:
0x15213

Agenda

- Shell Lab FAQs
- Malloc Lab Sneak Preview
- Virtual Memory Concepts
- **Address Translation**
  - Basic
  - TLB
  - Multilevel

VM Address Translation In Real Life

- Page tables are often multi-level, with the first level called the "page directory."
- This example uses Intel IA-32 conventions.
- To index into *k* levels, divide VPN into *k* parts.

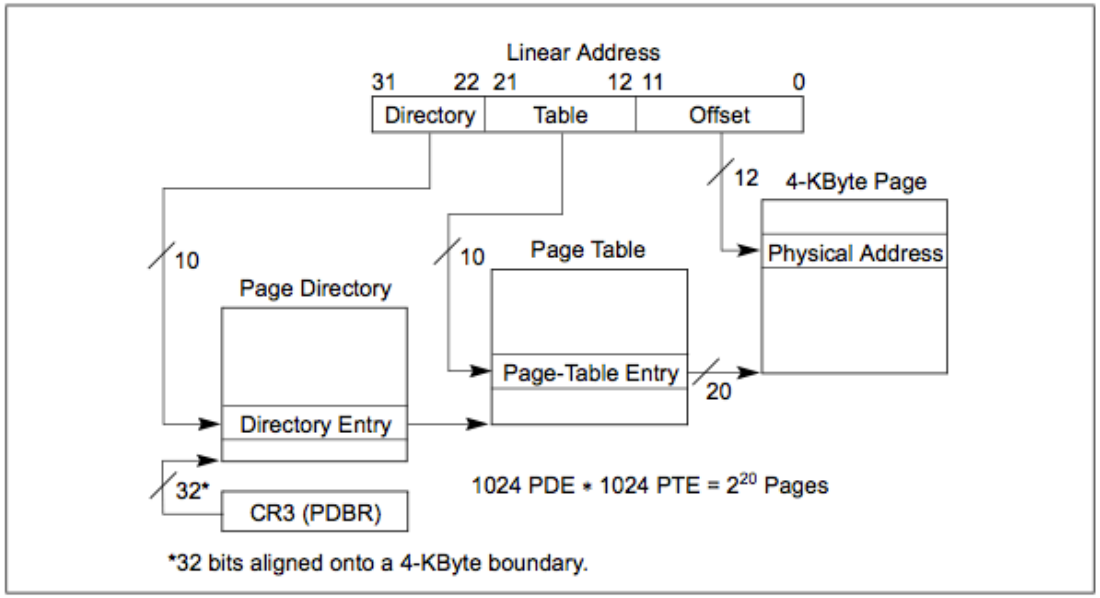
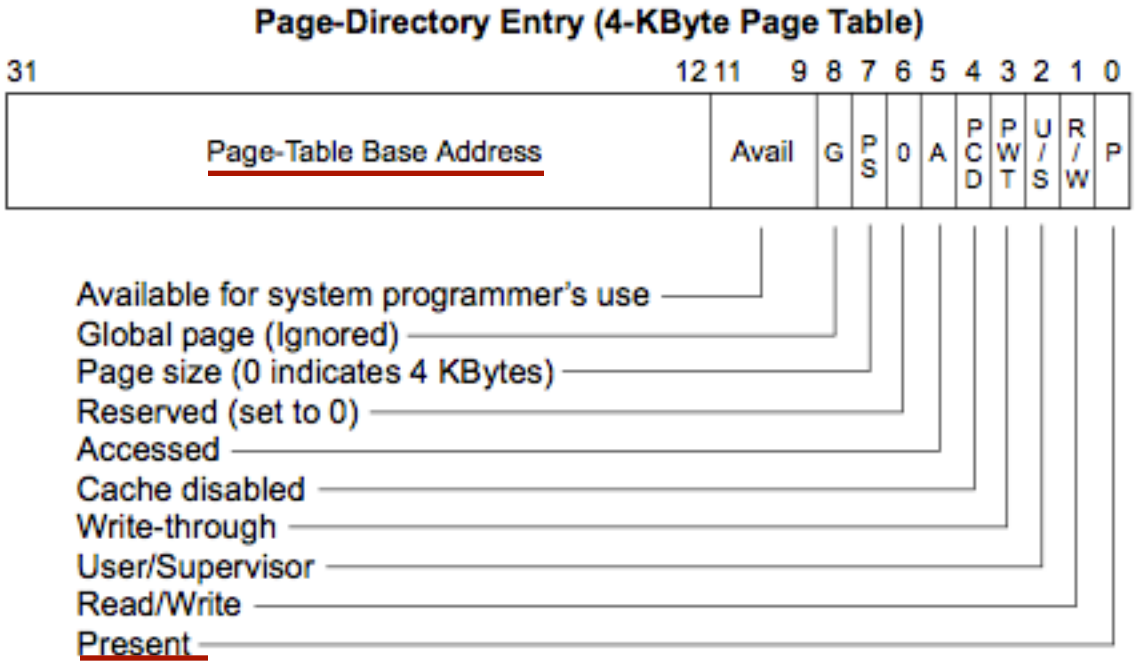


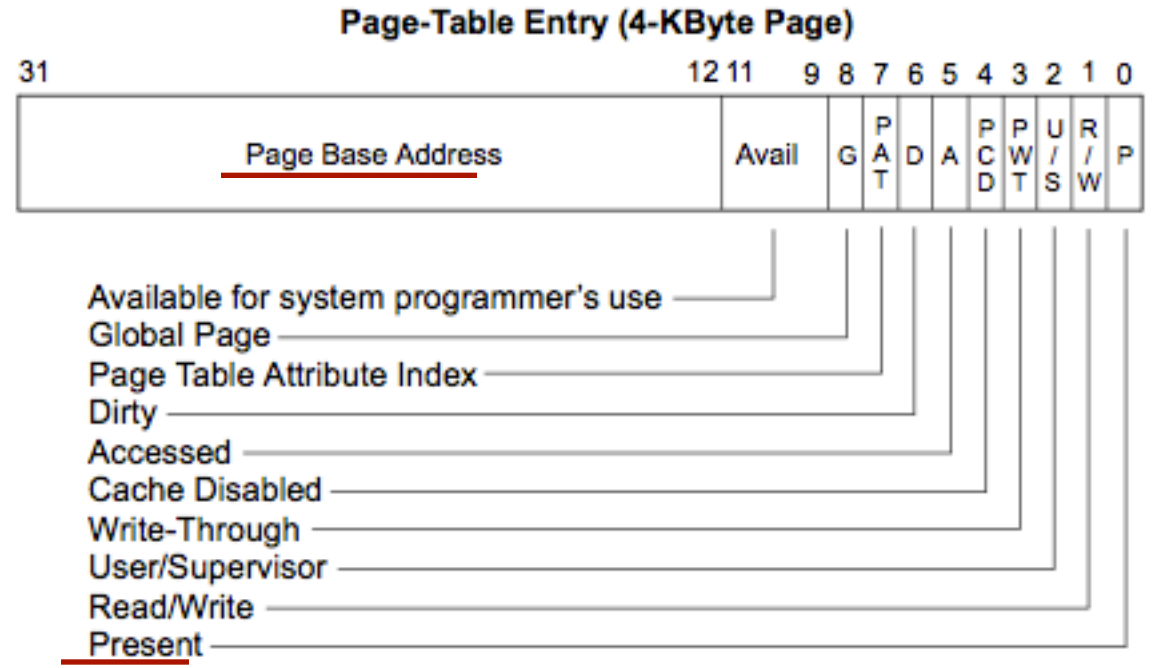
Figure 3-12. Linear Address Translation (4-KByte Pages)

(diagrams in the following slides are from Intel System Programmers Guide)

IA-32 Page-Directory Entry Format

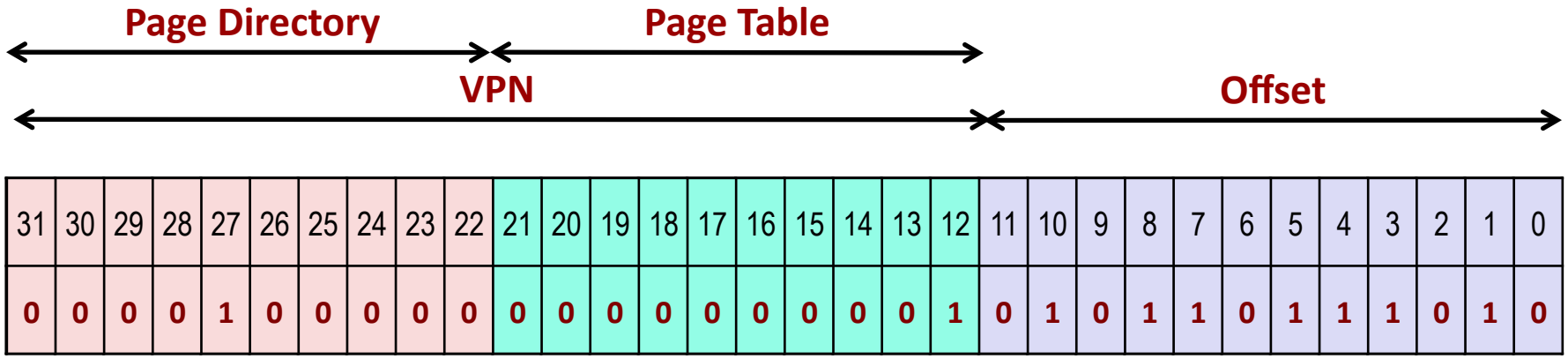


IA-32 Page-Table Entry Format



Example 3a: Intel Address Translation

- Page Directory Base Address = `0x0c23b000`.
- Read from virtual address `0x080016ba`. Pages are 4 KB.

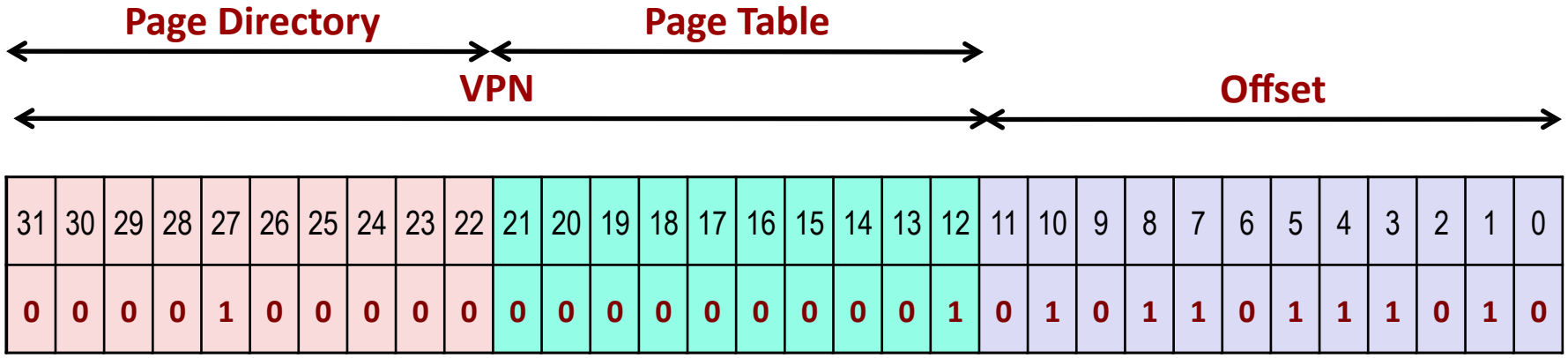


VPN:
PDI:
PTI:

Step 1: Calculate PDE address, find PT.
Page Directory Base Address: `0x0c23b000`
Entry size: 4 bytes
PDE address:

Example 3a: Intel Address Translation

- Page Directory Base Address = 0x0c23b000.
- Read from virtual address 0x080016ba. Pages are 4 KB.



VPN: 0x08001
PDI: 0x020
PTI: 0x001

Step 1: Calculate PDE address, find PT.
Page Directory Base Address: 0x0c23b000
Entry size: 4 bytes
PDE address: 0x0c23b000 + 4 * 0x20
= 0x0c23b080

Address	Content	Address	Content
00023000	beefbee0	00023120	12fdc883
00023200	debcfd23	00023320	d2e52933
00023fff	bcdeff29	00055004	8974d003
0005545c	457bc293	00055460	457bd293
00055464	457be293	0c23b020	01288b53
0c23b020	01288b53	0c23b040	012aab53
0c23b080	00055d01	0c23b09d	0ff2d303
0c23b274	00023d03	0c23b9fc	2314d222
2314d200	0fdc1223	2314d220	D21345a9
2314d4a0	D388bcbd	2314d890	00b32d00
24aee520	B58cdad1	29de2504	56ffad02

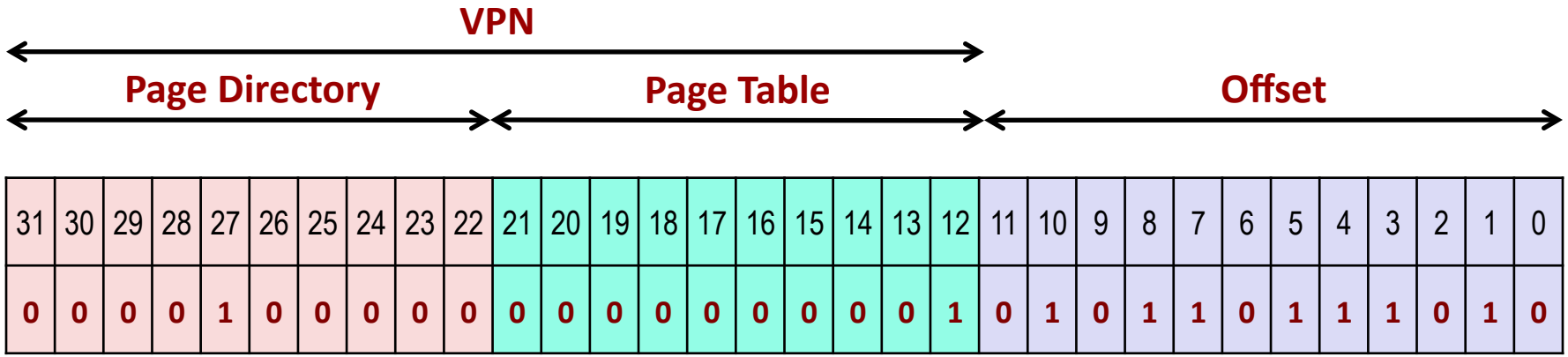
Step 1: Check present bit (last bit according to IA-32 format). Present bit is set.

Step 2: Calculate page base address by ignoring other flags.

`0x00055d01 & 0xFFFFF000 = 0x00055000`

Example 3a: Intel Address Translation

- Read from virtual address `0x080016ba`.

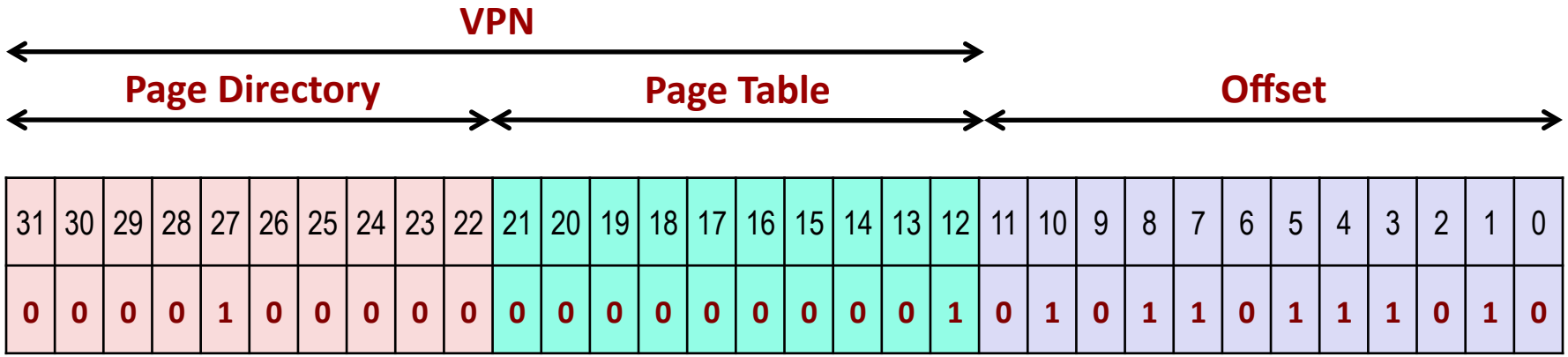


VPN:
PDI:
PTI:

Step 2: Calculate PTE address, find page.
Page Table Base Address: `0x00055000`
Entry size: 4 bytes
PTE address:

Example 3a: Intel Address Translation

- Read from virtual address 0x080016ba.



VPN: 0x08001
PDI: 0x020
PTI: 0x001

Step 2: Calculate PTE address, find page.
Page Table Base Address: 0x00055000
Entry size: 4 bytes
PTE address: 0x00055000 + 4 * 0x01
= 0x00055004

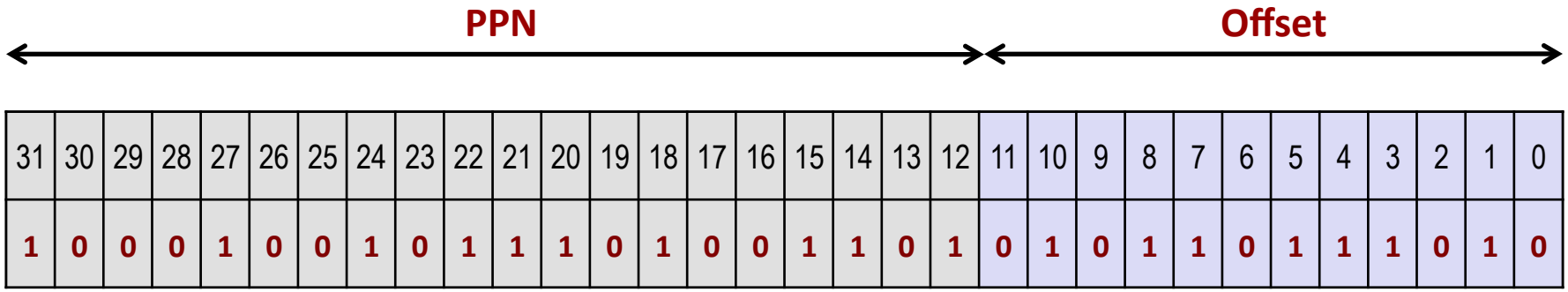
Address	Content	Address	Content
00023000	beefbee0	00023120	12fdc883
00023200	debcfd23	00023320	d2e52933
00023fff	bcdeff29	00055004	8974d003
0005545c	457bc293	00055460	457bd293
00055464	457be293	0c23b020	01288b53
0c23b020	01288b53	0c23b040	012aab53
0c23b080	00055d01	0c23b09d	0ff2d303
0c23b274	00023d03	0c23b9fc	2314d222
2314d200	0fdc1223	2314d220	D21345a9
2314d4a0	D388bcbd	2314d890	00b32d00
24aee520	B58cdad1	29de2504	56ffad02

Step 1: Check present bit (last bit according to IA-32 format).
Present bit is set.

Step 2: Calculate page base address by ignoring other flags.
`0x8974d003 & 0xFFFFF000 = 0x8974d000`

Example 3a: Intel Address Translation

- Read from virtual address 0x080016ba.



VPN: 0x08001
PDI: 0x020
PTI: 0x001

Step 3: combine page address & offset

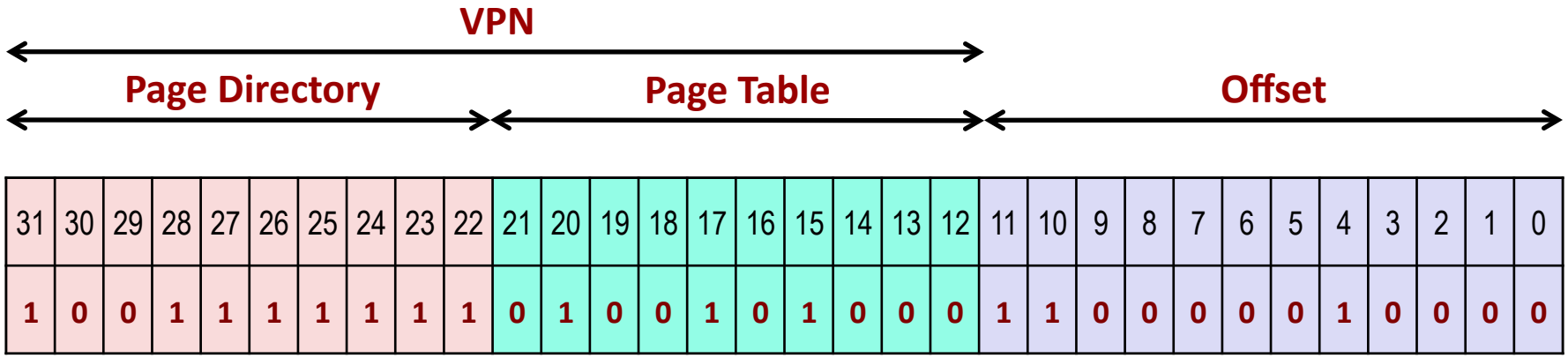
Page Base Address: 0x8974d000
Offset: 0x000006ba
Physical Address: 0x8974d6ba

Example 3a: Recap!

- Divide the virtual address into VPN and offset.
- Divide the VPN into a page *directory* index and a page *table* index. These index into each of those tables.
- Your page directory entry is at the page directory base address + 4 * {your page directory index}. Discard last 3 bytes; that gives you your page table base address.
- The correct page table entry is at the page table base address + 4 * {your page table index}. Discard last 3 bytes; that gives you the physical page base address.
- Add your offset to that physical page base address to get your final physical address.
- (Obviously, check for valid bits at every stage.)

Example 3b: Intel Address Translation

- Read from virtual address `0x9fd28c10`.



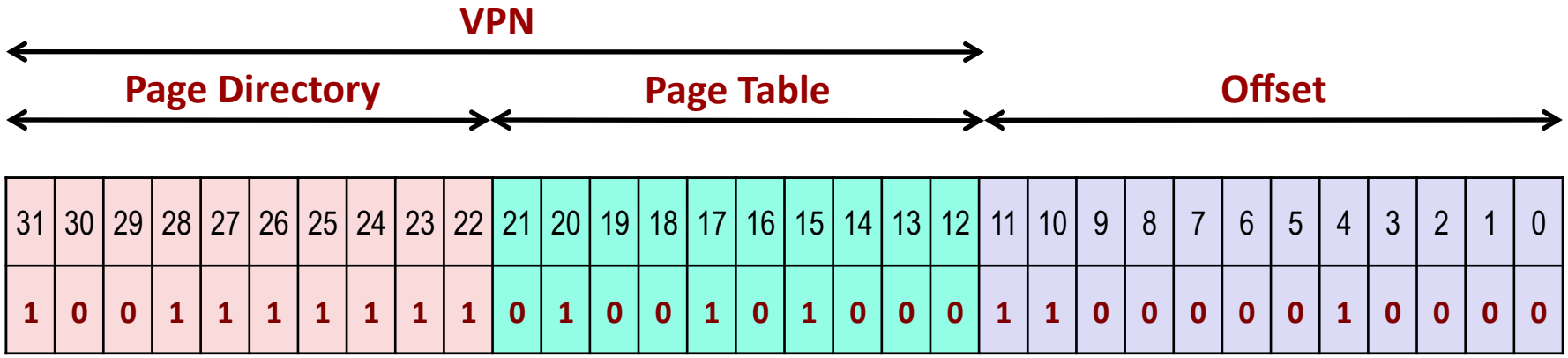
VPN: Page Directory Base Address: `0x0c23b000`

PDI: Entry size: 4 bytes

PTI: PTE Address

Example 3b: Intel Address Translation

- Read from virtual address `0x9fd28c10`.



VPN: `0x9fd28`
PDI: `0x27f`
PTI: `0x128`

Page Directory Base Address: `0x0c23b000`
Entry size: 4 bytes
PTE Address `0x0c23b000 + 4 * 0x27f`
= `0x0c23b9fc`

Address	Content	Address	Content
00023000	beefbee0	00023120	12fdc883
00023200	debcfd23	00023320	d2e52933
00023fff	bcdeff29	00055004	8974d003
0005545c	457bc293	00055460	457bd293
00055464	457be293	0c23b000	01288b53
0c23b020	01288b53	0c23b040	012aab53
0c23b080	00055d01	0c23b09d	0ff2d303
0c23b274	00023d03	0c23b9fc	2314d222
2314d200	0fdc1223	2314d220	D21345a9
2314d4a0	D388bcbd	2314d890	00b32d00
24aee520	B58cdad1	29de2504	56ffad02

Step 1: Check present bit (last bit according to IA-32 format).
PRESENT BIT IS NOT SET.

Page fault! (Trap to kernel.)

Questions?