Virtual Memory

Aravind Prakash

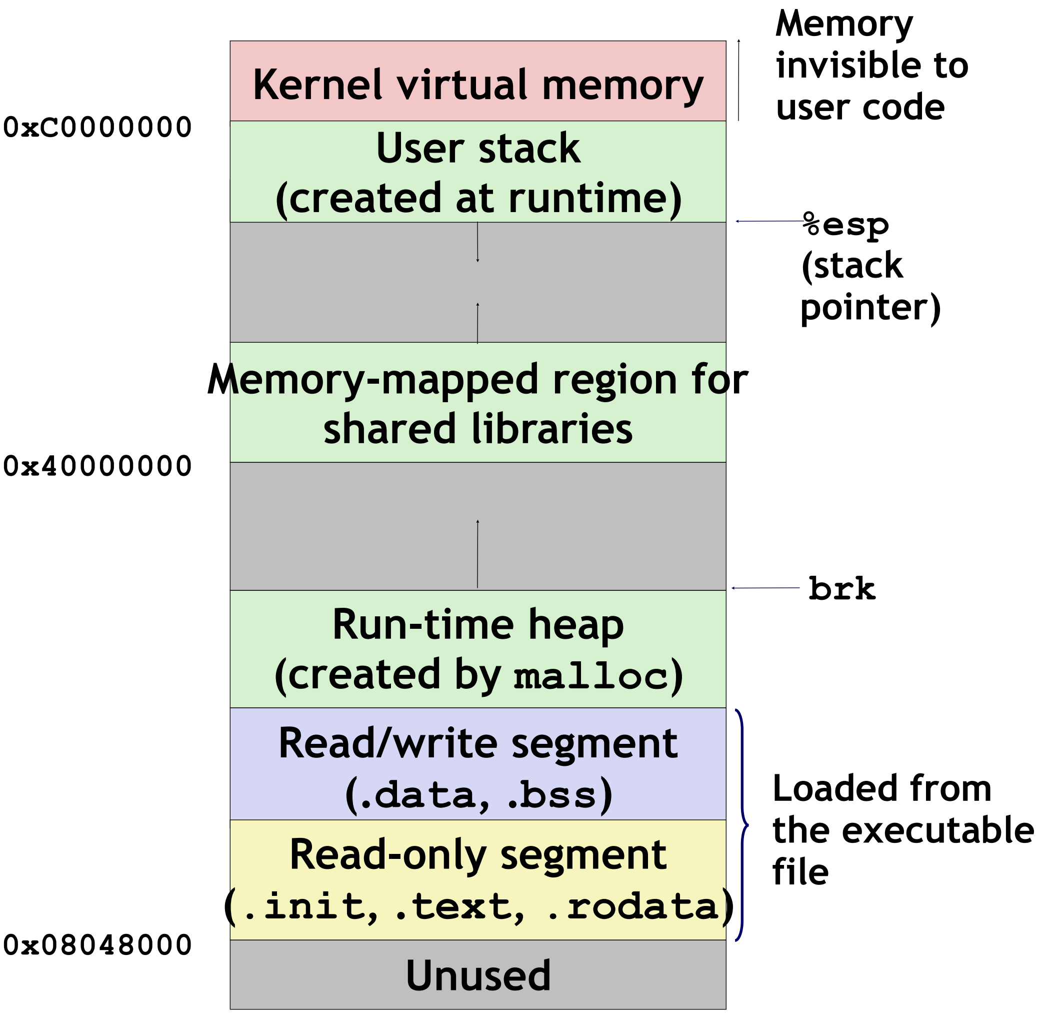
Modified version of slides by:
Dr. Karthic Palaniappan

Agenda

- 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???


Memory invisible to user code
0xC0000000
Kernel virtual memory
User stack (created at runtime)
%esp (stack pointer)
Memory-mapped region for shared libraries
0x40000000
brk
Run-time heap (created by malloc)
Read/write segment (.data, .bss)
Read-only segment (.init, .text, .rodata)
Loaded from the executable file
0x08048000
Unused

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.

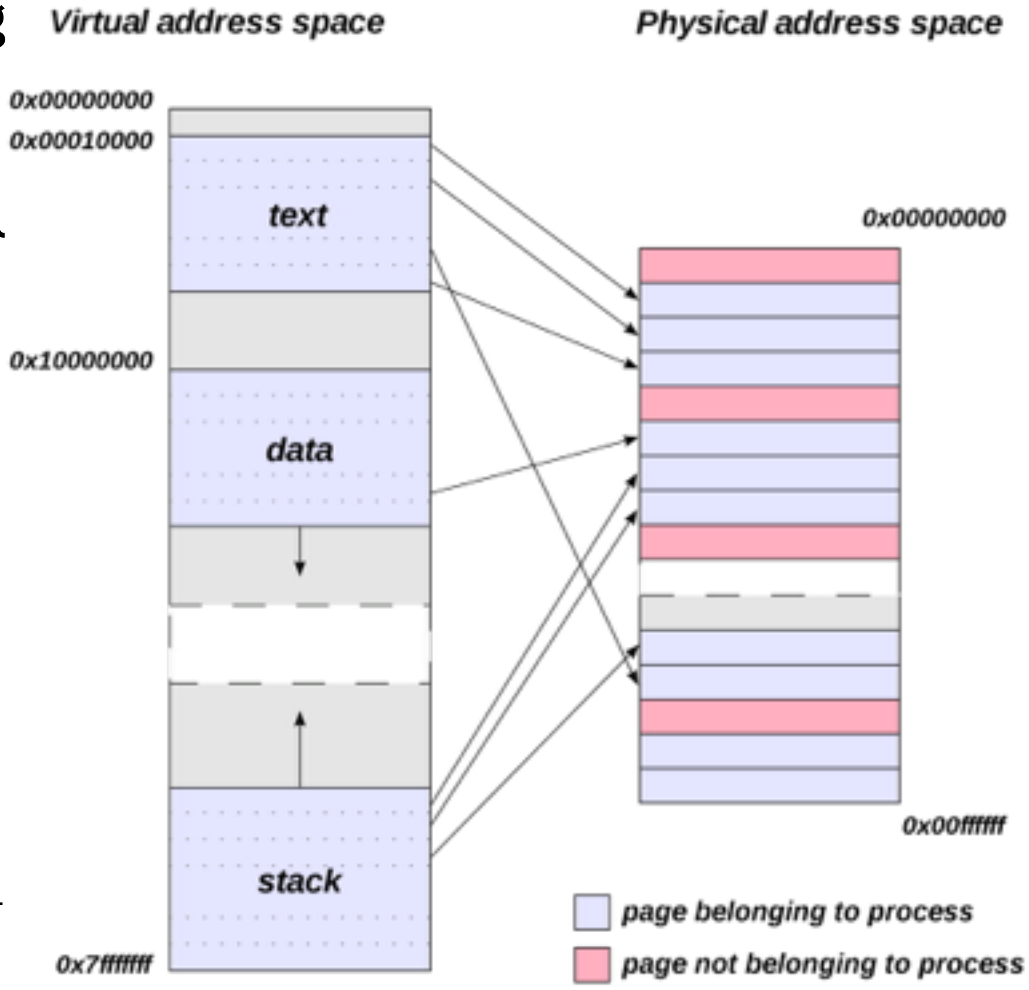
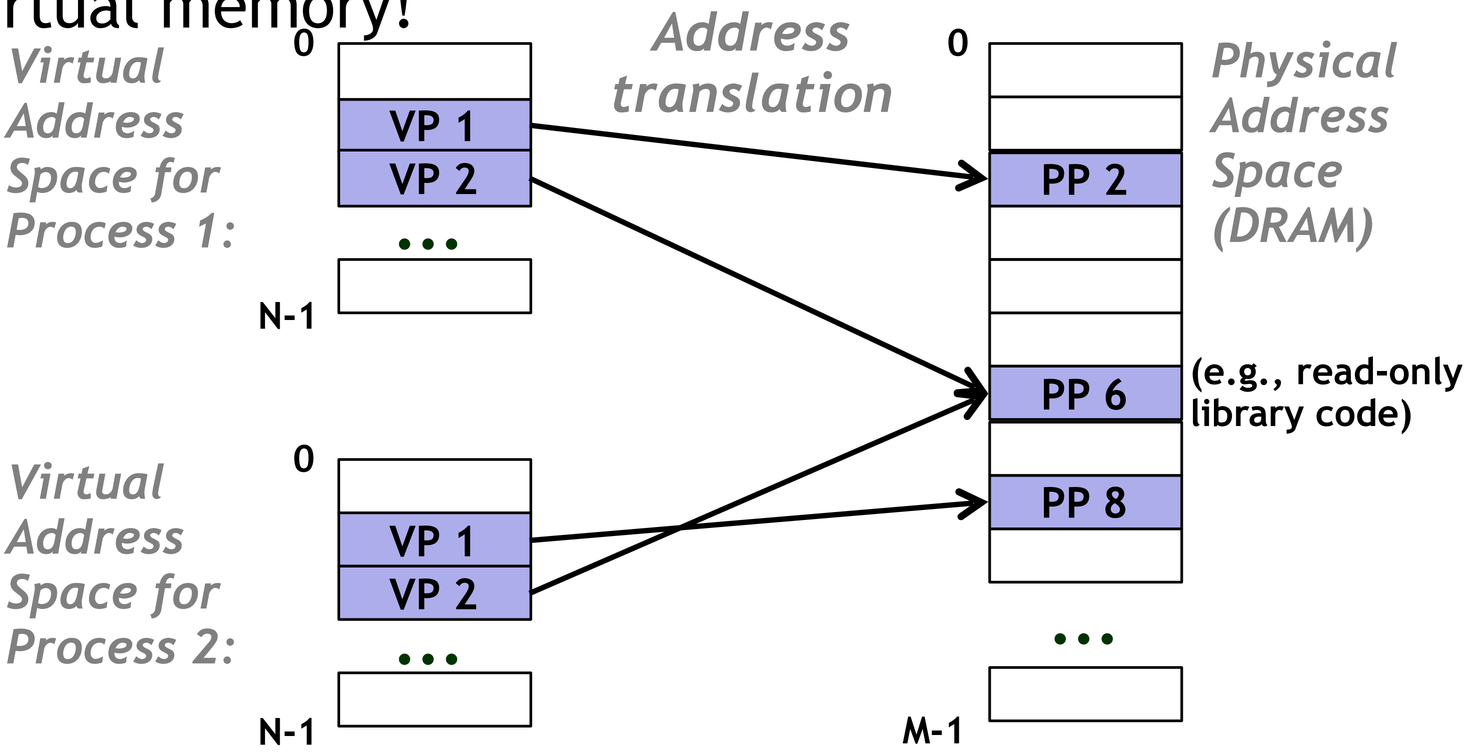


Image: http://en.wikipedia.org/wiki/File:Virtual_address_space_and_physical_address_space_relationship.svg

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 Address Space for Process 1:
0
VP 1
VP 2
...
N-1
Address translation
0
PP 2
PP 6
PP 8
...
M-1
Physical Address Space (DRAM)
(e.g., read-only library code)
Virtual Address Space for Process 2:
0
VP 1
VP 2
...
N-1

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.

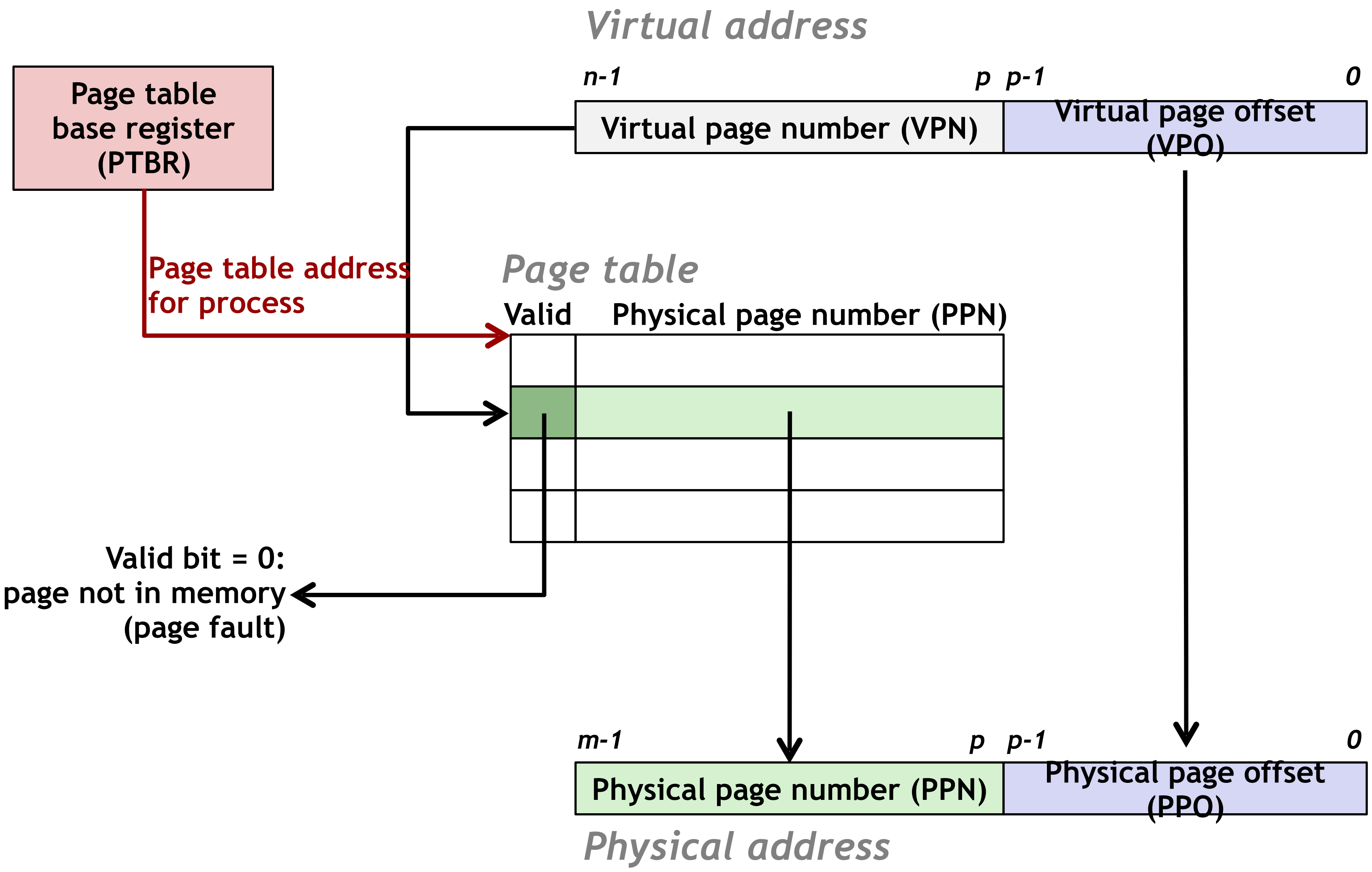
Agenda

- Virtual Memory Concepts
- Address Translation
  - Basic
  - TLB
  - Multilevel

VM Address Translation

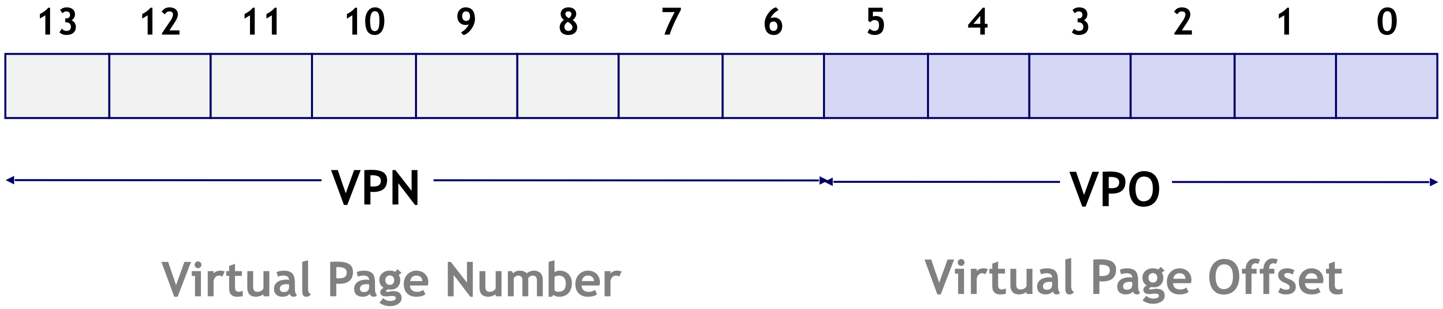
- Virtual Address Space
  - *V = {0, 1, …, N–1}*
  - There are N possible virtual addresses.
  - Virtual addresses are n bits long; $2^n = N$.
- Physical Address Space
  - *P = {0, 1, …, M–1}*
  - There are M possible physical addresses.
  - Virtual addresses are m bits long; $2^m = M$.
- Memory is grouped 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


Virtual address
n-1
p
p-1
0
Virtual page number (VPN)
Virtual page offset (VPO)
Page table base register (PTBR)
Page table address for process
Page table
Valid
Physical page number (PPN)
Valid bit = 0:
page not in memory
(page fault)
m-1
p
p-1
0
Physical page number (PPN)
Physical page offset (PPO)
Physical address

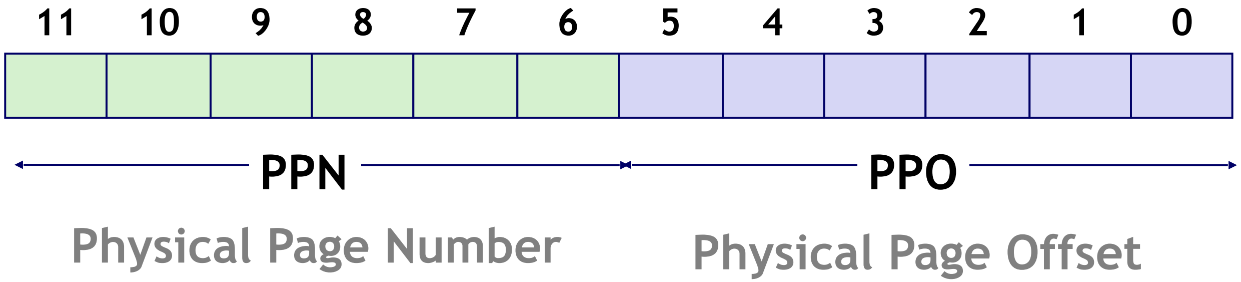
VM Address Translation

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



Virtual Page Number

Virtual Page Offset

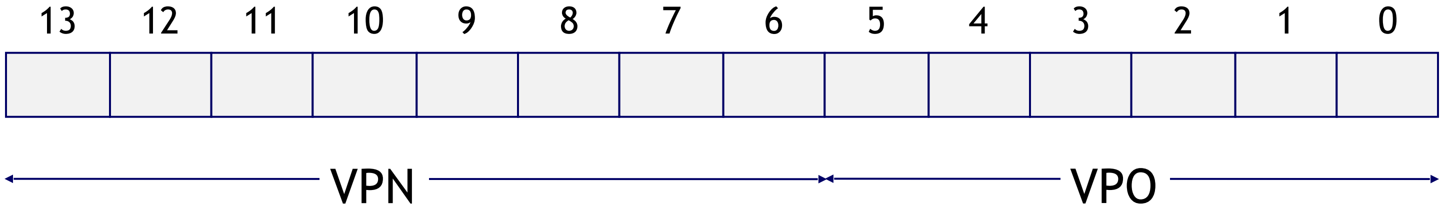


Physical Page Number

Physical Page Offset

Example 1: Address Translation

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

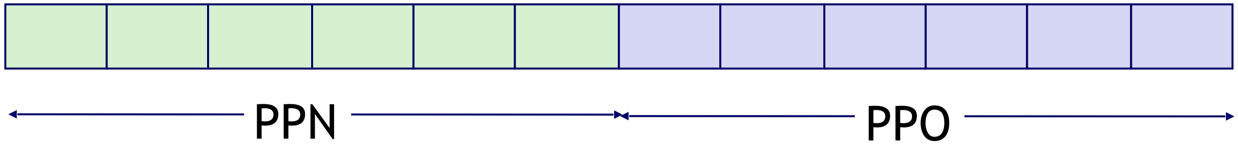


- VPN: ______
- PPN: ______
- Physical address:

____

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`.

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	1	0	1	0	0

VPN VPO

- 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

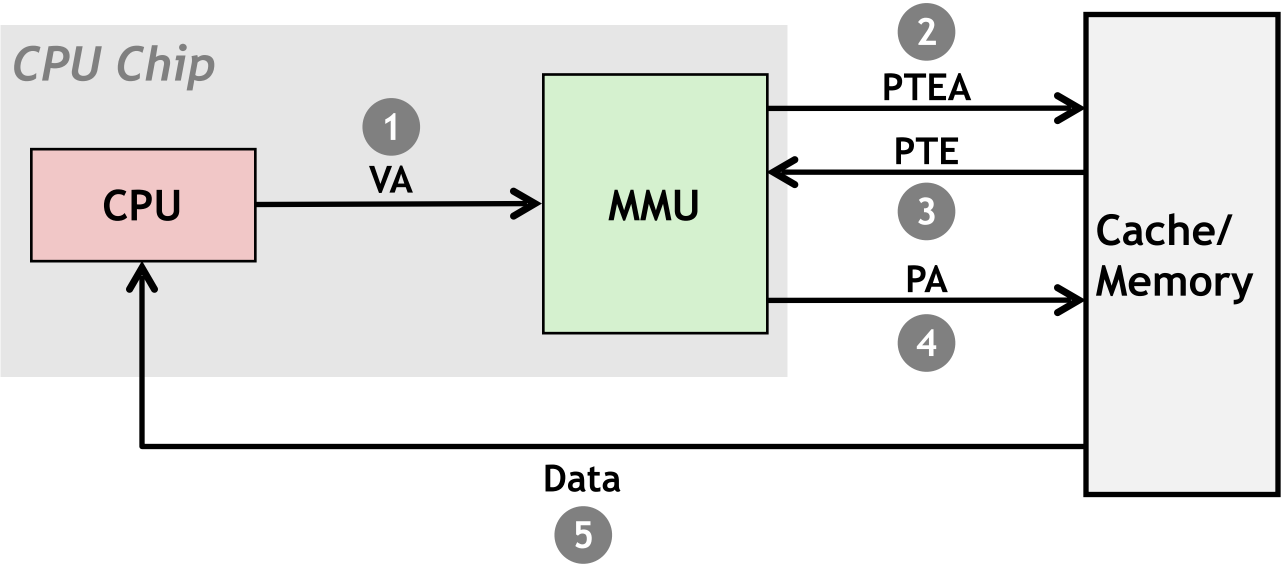
0	0	1	1	0	1	0	1	0	1	0	0

PPN PPO

Agenda

- Virtual Memory Concepts
- Address Translation
  - Basic
  - TLB
  - Multilevel

Address Translation: Page Hit



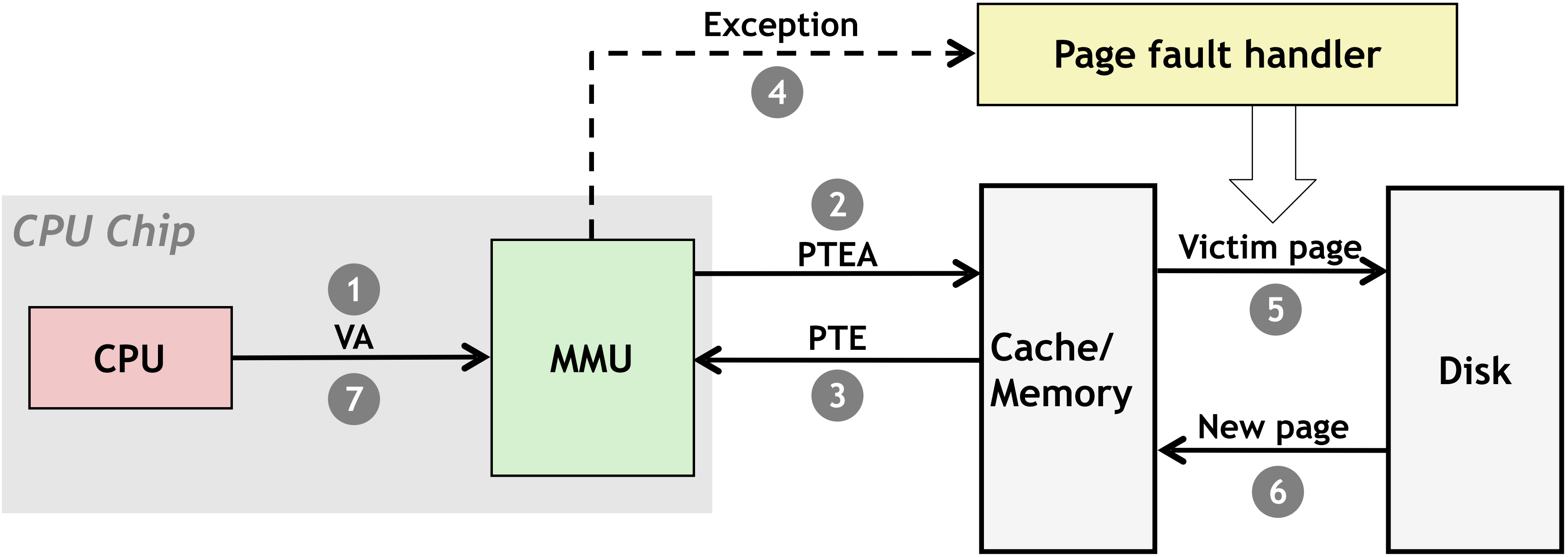
1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

Address Translation: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) Valid bit is zero, so MMU triggers page fault exception

5) Handler identifies victim (and, if dirty, pages it out to disk)

6) Handler pages in new page and updates PTE in memory

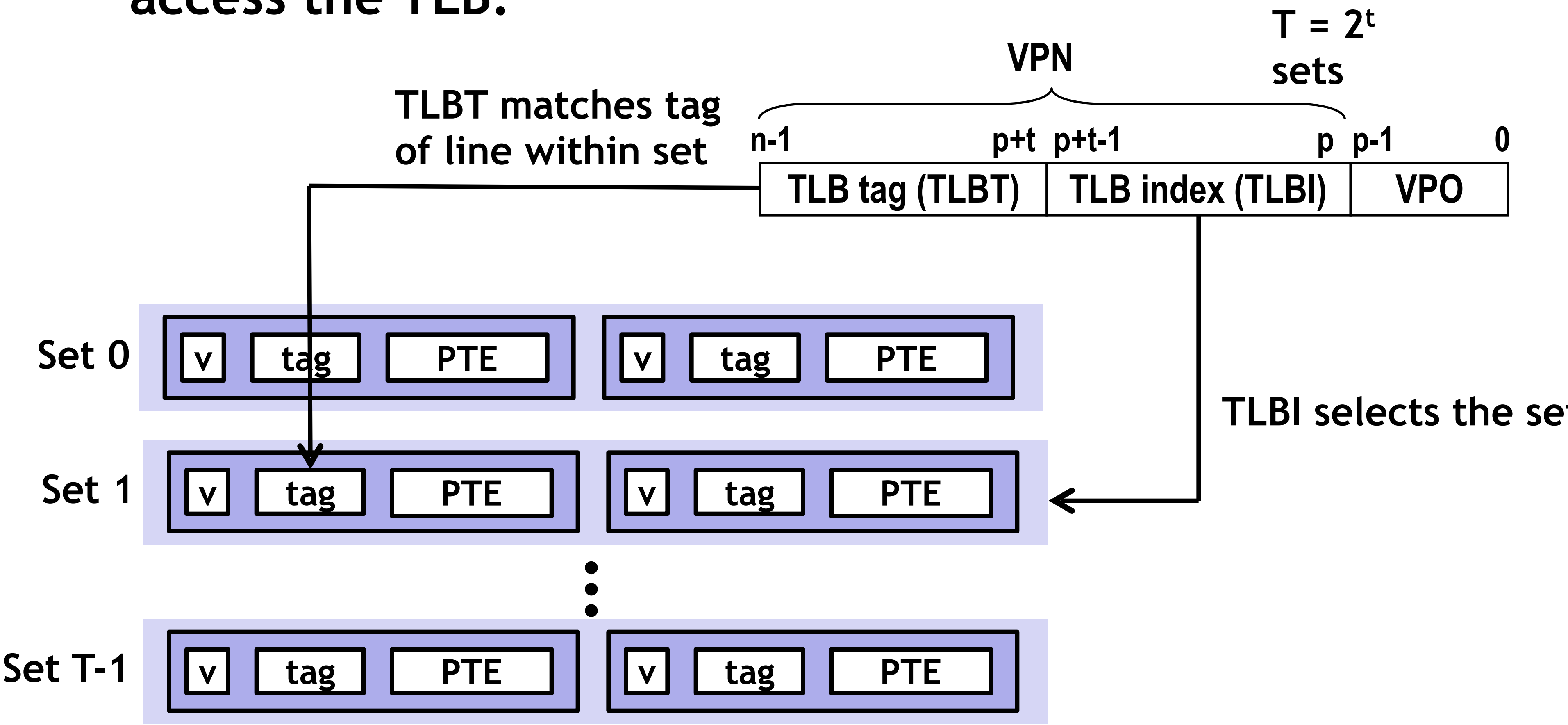
7) Handler returns to original process, restarting faulting instruction

Speeding up Translation with a TLB

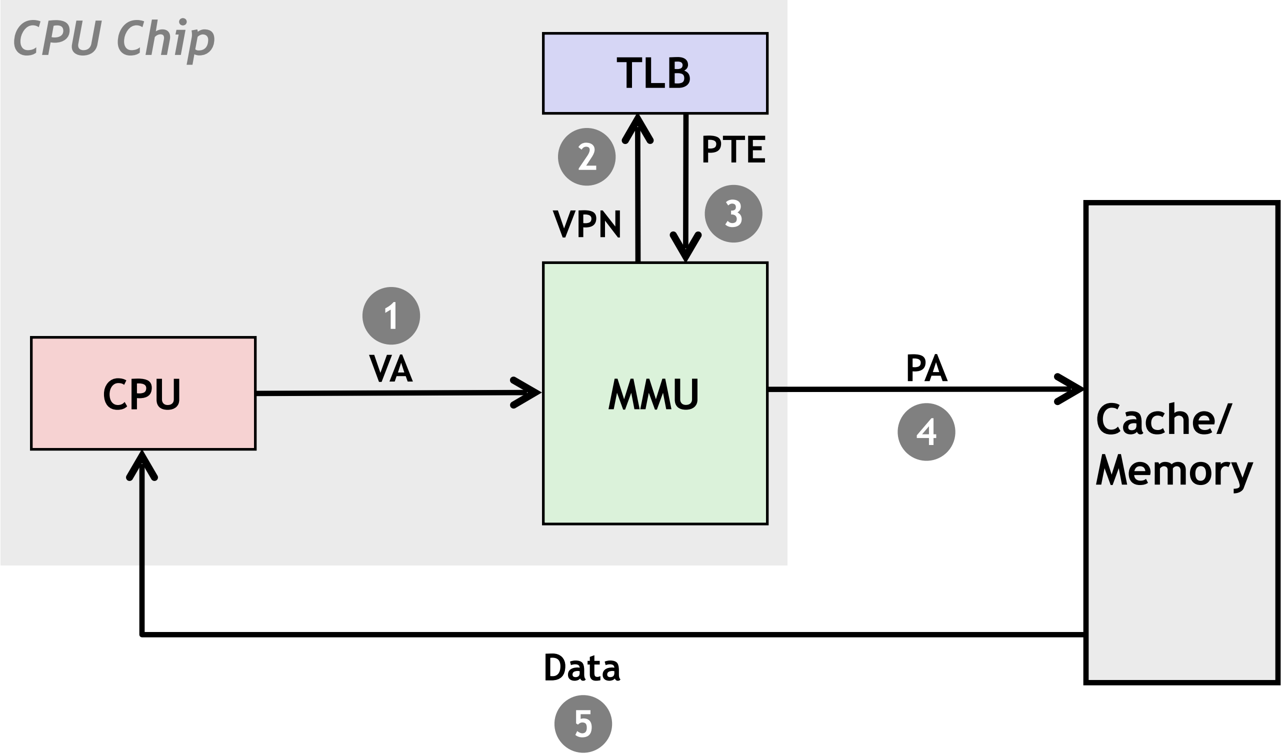
- **Page table entries (PTEs) are cached in L1 like any other memory word**
  - PTEs may be evicted by other data references
  - PTE hit still requires a small L1 delay
- **Solution: *Translation Lookaside Buffer* (TLB)**
  - Small set-associative hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages

Accessing the TLB

- MMU uses the VPN portion of the virtual address to access the TLB:

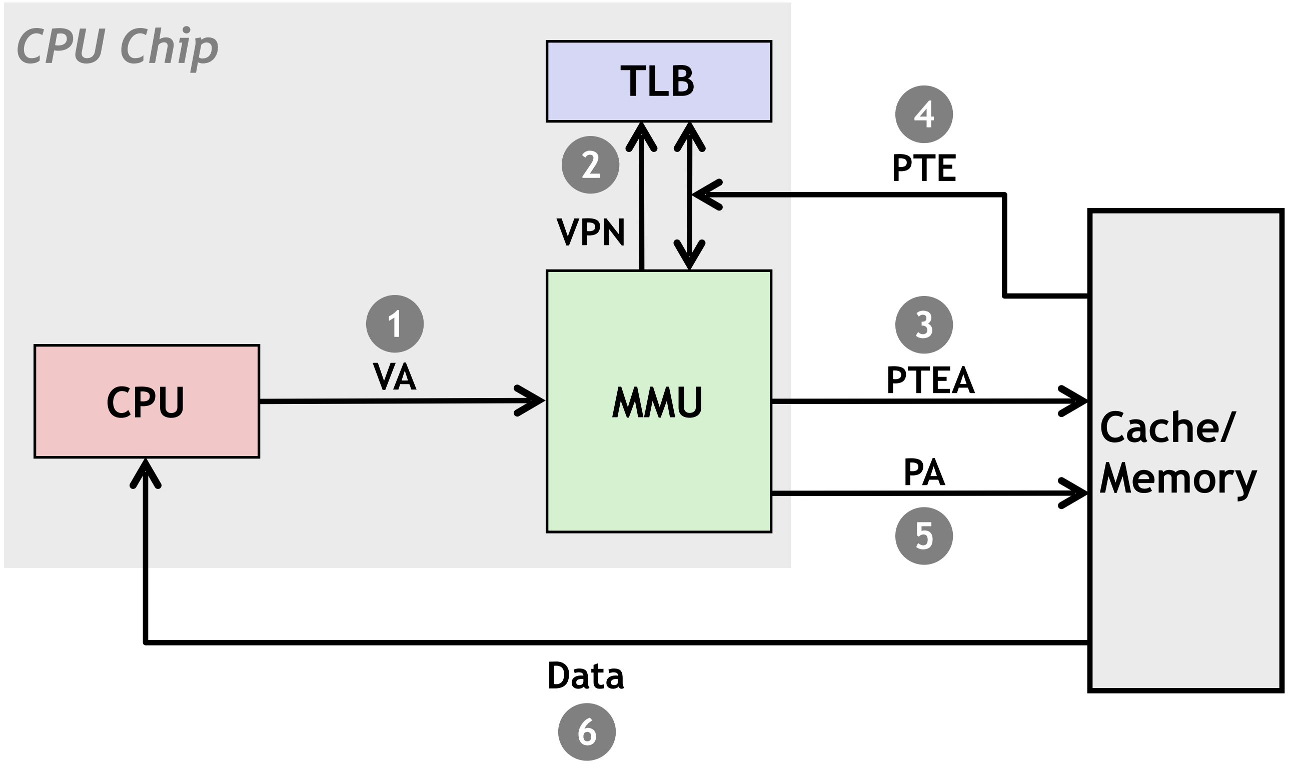


TLB Hit



A TLB hit eliminates a memory access

TLB Miss

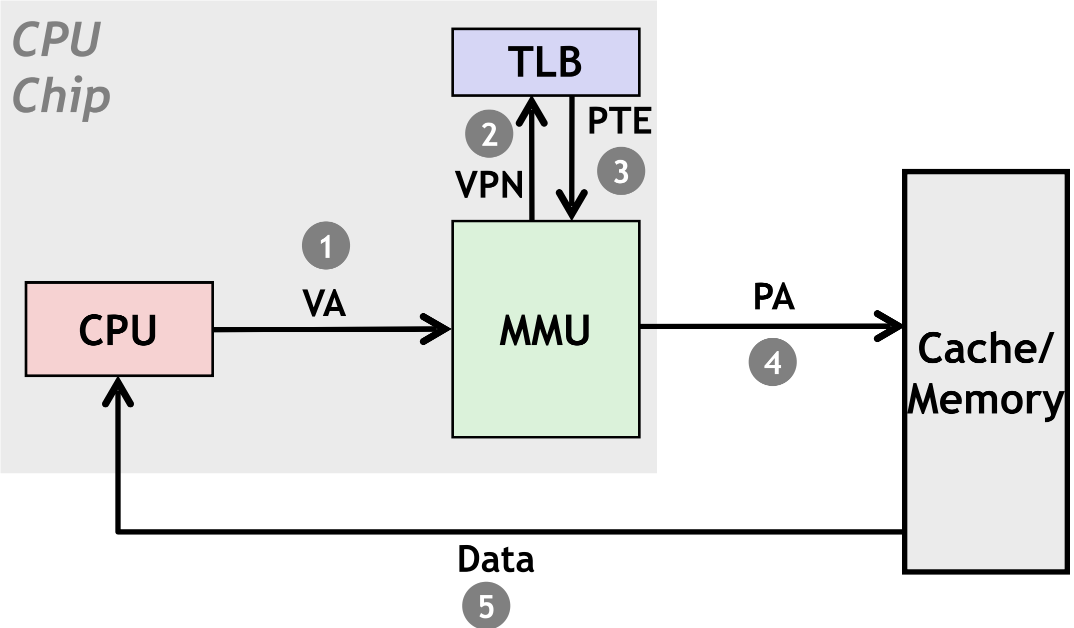


A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare. Why?

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	4 KB page size
256 KB of physical memory	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

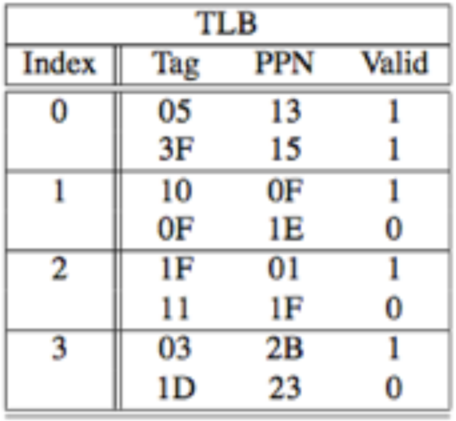
1 MB of virtual memory	4 KB page size
256 KB of physical memory	TLB: 8 entries, 2-way set associative

- 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?

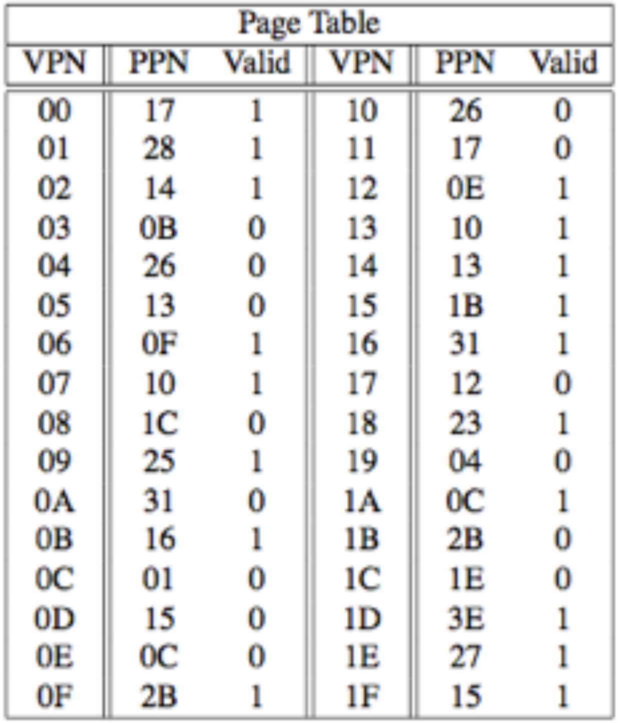
  12. (4 KB = 2^{12} bytes.)
- How many bits are needed to represent VPN? **8. (20-12.)**
- How many bits are in the TLB index? **2. (4 sets = 2^2 set bits.)**
- How many bits are in the TLB tag? **6. (8-2.)**

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.)



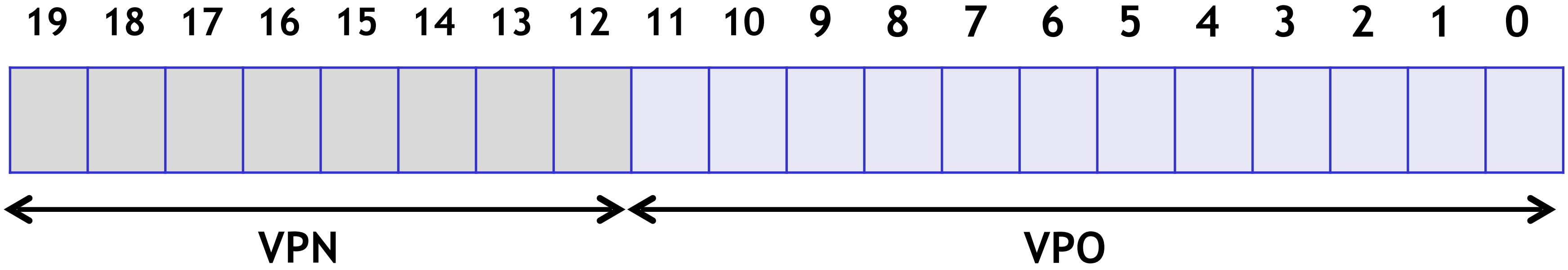
TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0



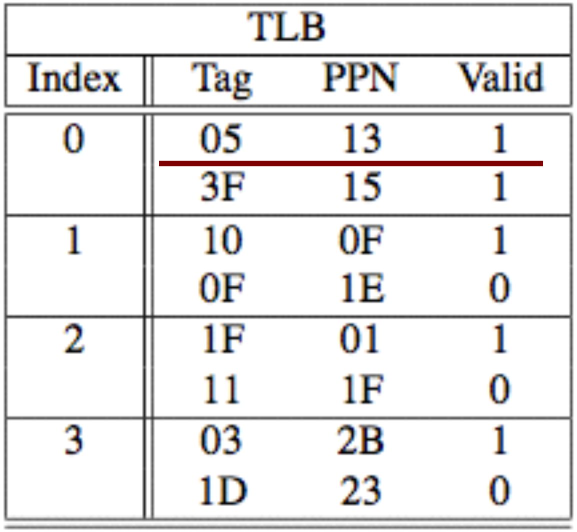
Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

Example 2a: Address Translation with TLB

0x14213



VPN:
TLBI:
TLBT:



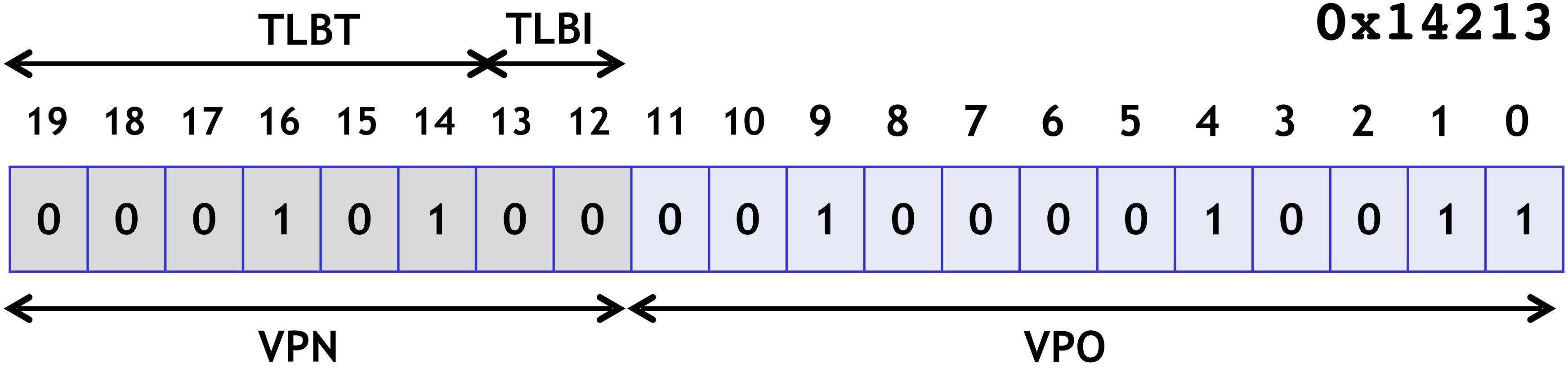
TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

TLB Hit!
PPN:
Offset:

Physical address:

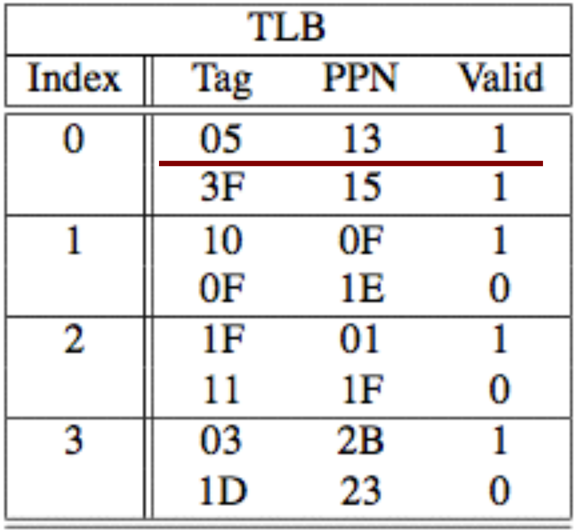
Example 2a: Address Translation with TLB

0x14213



19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	1	0	0	1	1

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



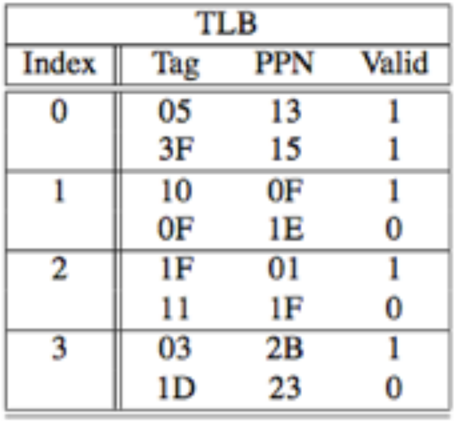
TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

TLB Hit!
PPN: 0x13
Offset: 0x213

Physical address:
0x13213

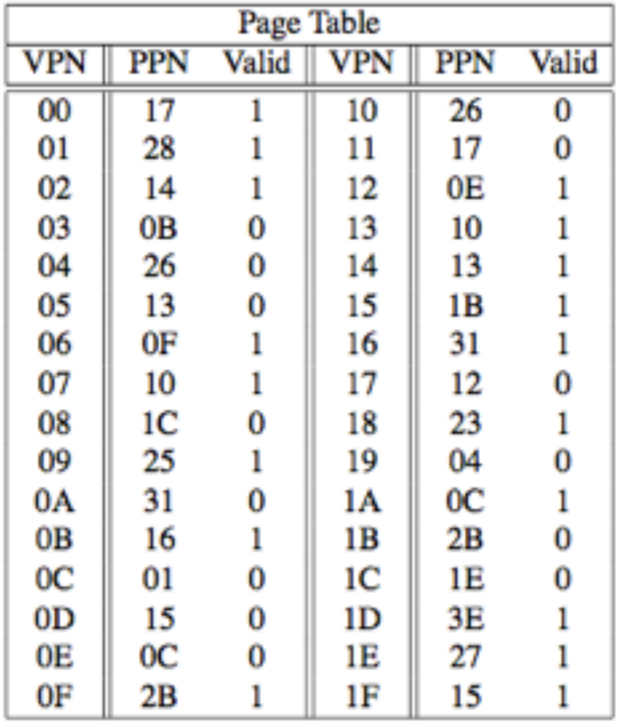
Example 2b: Address Translation with TLB

- 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

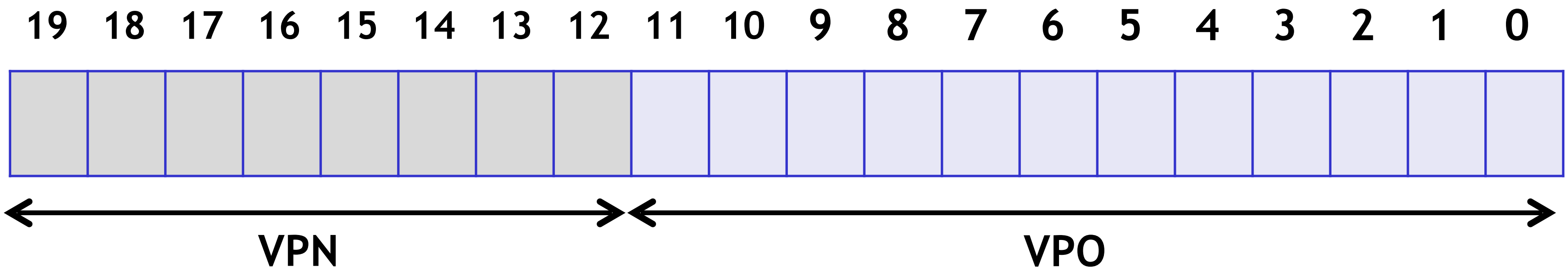


Page Table

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

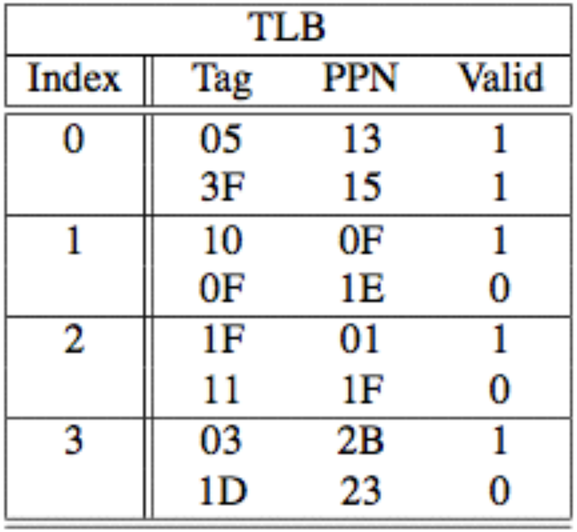


19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

VPN

VPO

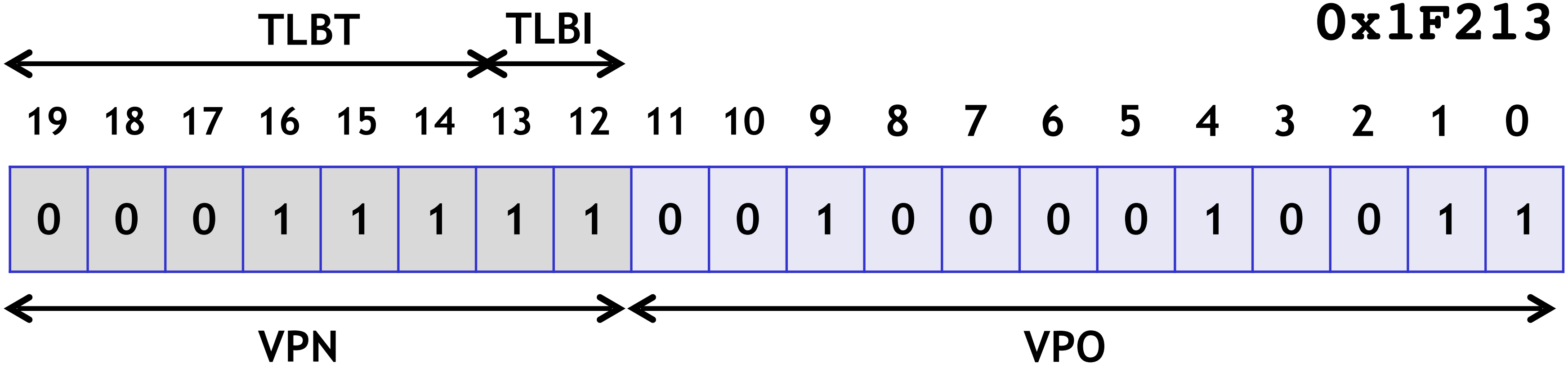
VPN:
TLBI:
TLBT:



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

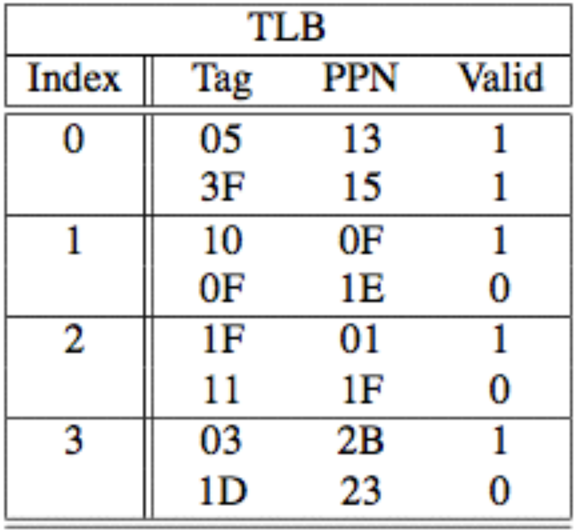
Example 2b: Address Translation with TLB

0x1F213



19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	1	0	0	1	0	0	0	0	1	0	0	1	1

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



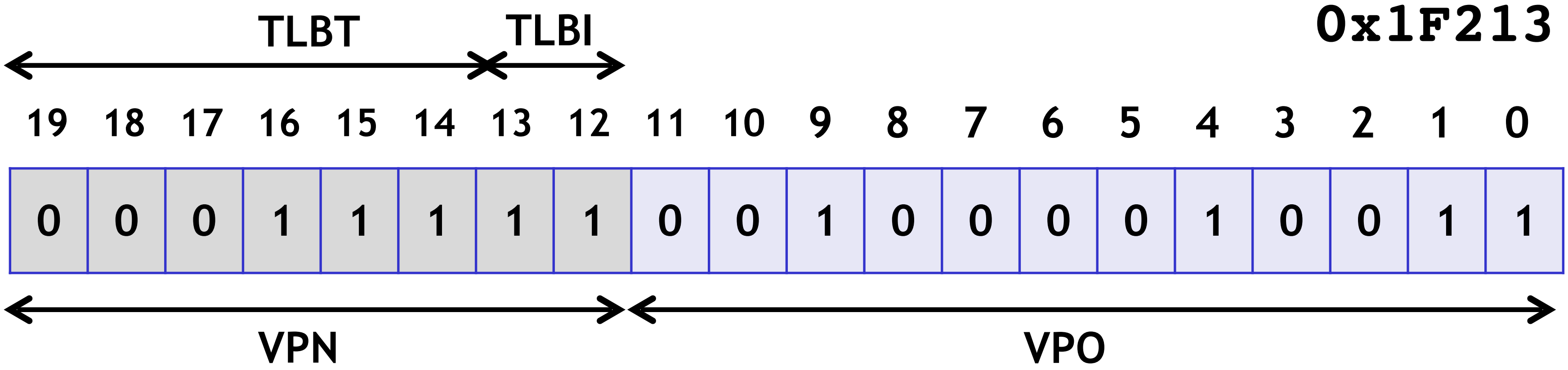
TLB			
Index	Tag	PPN	Valid
0	05	13	1
	3F	15	1
1	10	0F	1
	0F	1E	0
2	1F	01	1
	11	1F	0
3	03	2B	1
	1D	23	0

TLB Miss!

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

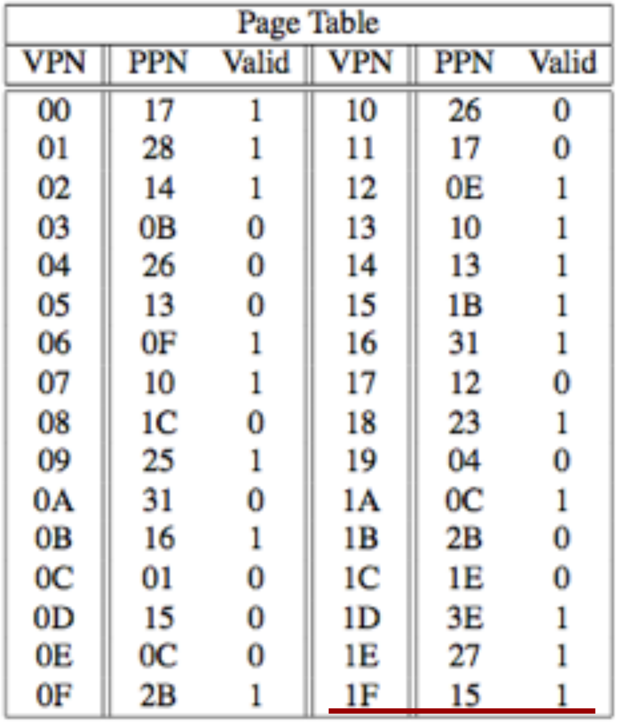
Example 2b: Address Translation with TLB

0x1F213



19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	1	0	0	1	0	0	0	0	1	0	0	1	1

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

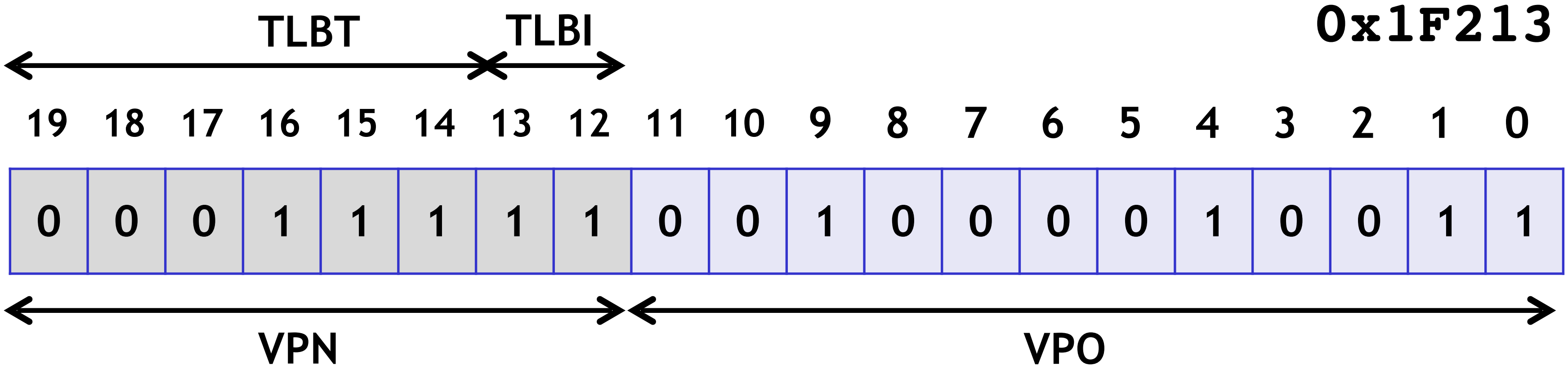


Page Table					
VPN	PPN	Valid	VPN	PPN	Valid
00	17	1	10	26	0
01	28	1	11	17	0
02	14	1	12	0E	1
03	0B	0	13	10	1
04	26	0	14	13	1
05	13	0	15	1B	1
06	0F	1	16	31	1
07	10	1	17	12	0
08	1C	0	18	23	1
09	25	1	19	04	0
0A	31	0	1A	0C	1
0B	16	1	1B	2B	0
0C	01	0	1C	1E	0
0D	15	0	1D	3E	1
0E	0C	0	1E	27	1
0F	2B	1	1F	15	1

Page Table Hit
PPN:
Offset:

Physical address:

Example 2b: Address Translation with TLB



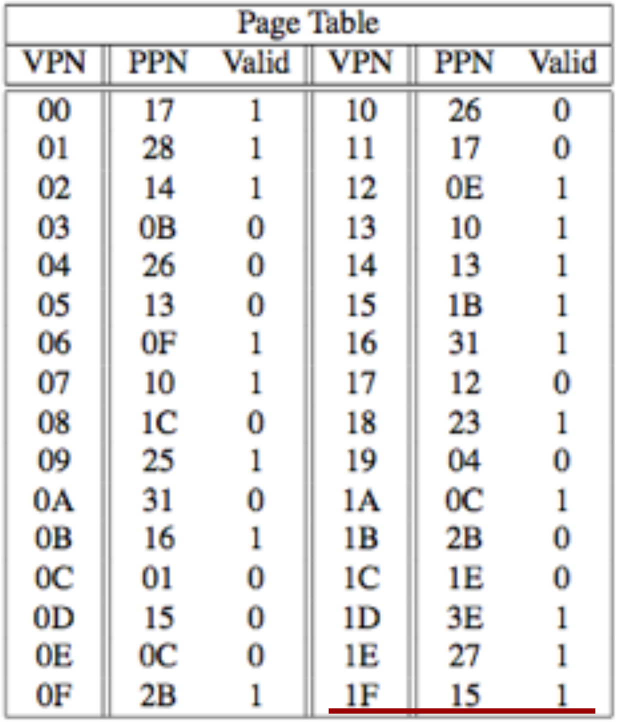
0x1F213

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	1	0	0	1	0	0	0	0	1	0	0	1	1

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:

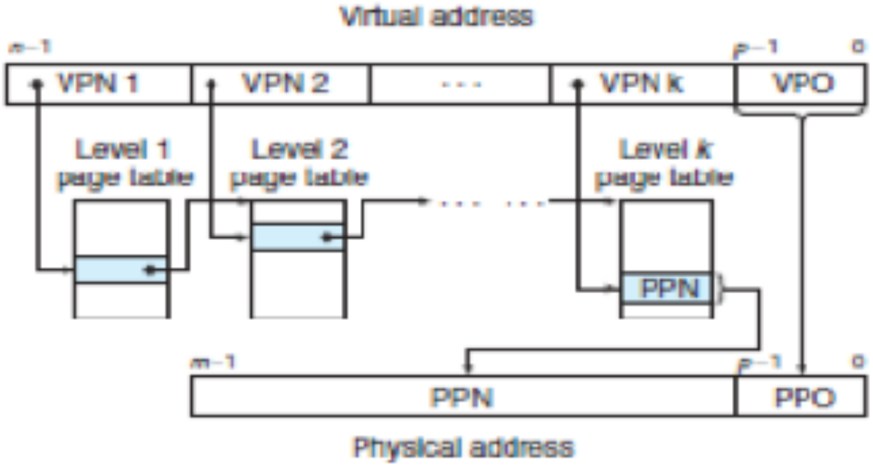
0x15213

Agenda

- Virtual Memory Concepts
- Address Translation
  - Basic
  - TLB
  - Multilevel

Address Translation in Real Life

- Multi level page tables, with the first level often called a “page directory”
- Use first part of the VPN to get to the right directory and then the next part to get the PPN
- K-level page table divides VPN into k parts


Virtual address
n−1
p−1
0
VPN 1
VPN 2
. . .
VPN k
VPO
Level 1
page table
Level 2
page table
. . . . . .
Level k
page table
PPN
m−1
p−1
0
PPN
PPO
Physical address