CSC 252: Computer Organization
Spring 2018: Lecture 22

Instructor: Yuhao Zhu

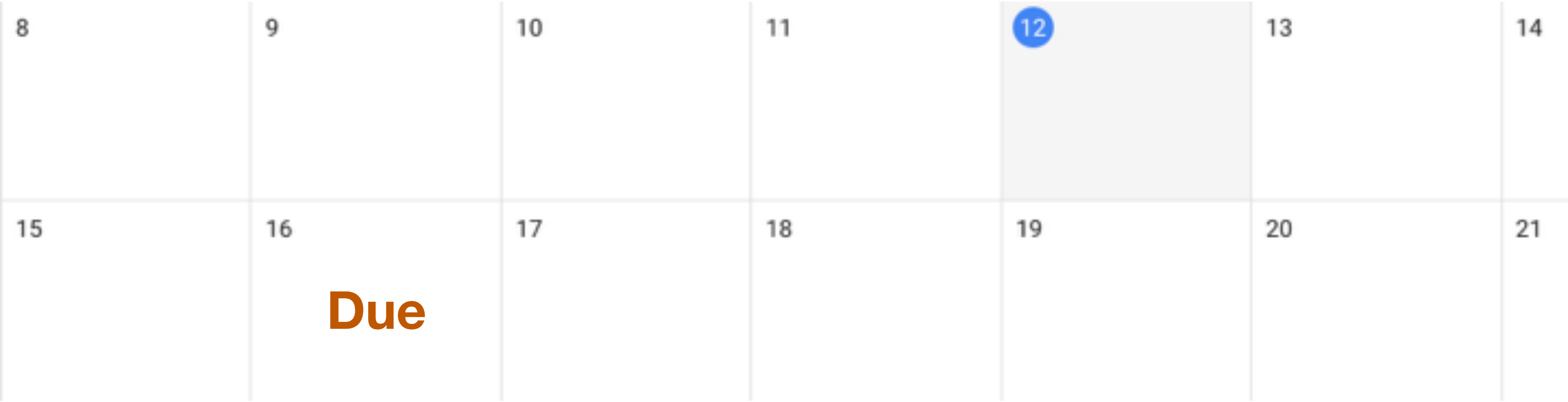
Department of Computer Science
University of Rochester

Action Items:

- **Programming Assignment 5 is due Monday**

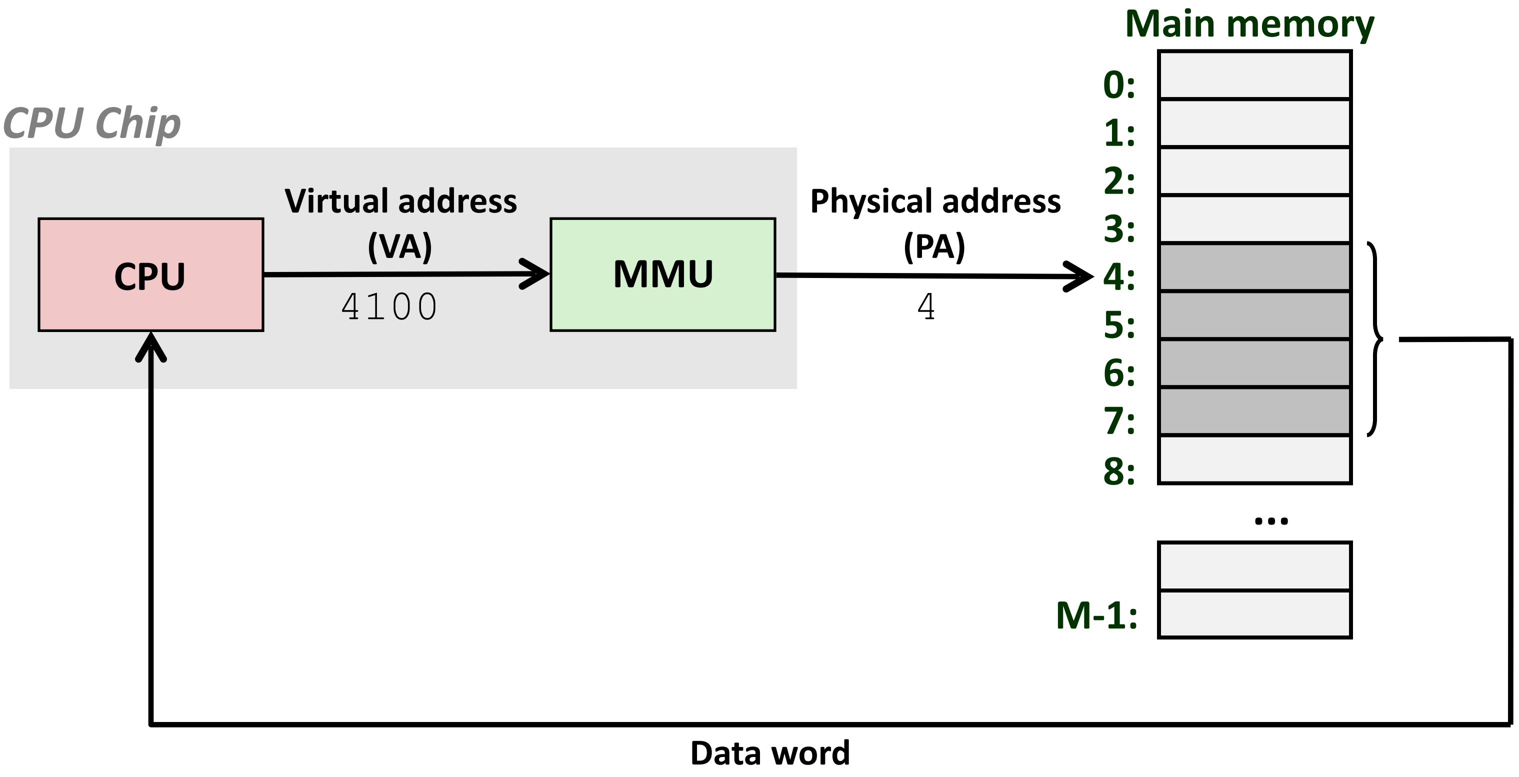
Announcement

- Programming Assignment 5 is out
  - Main assignment: 11:59pm, **Monday, April 16**.
- No office hours today. Had already moved to this Tuesday.

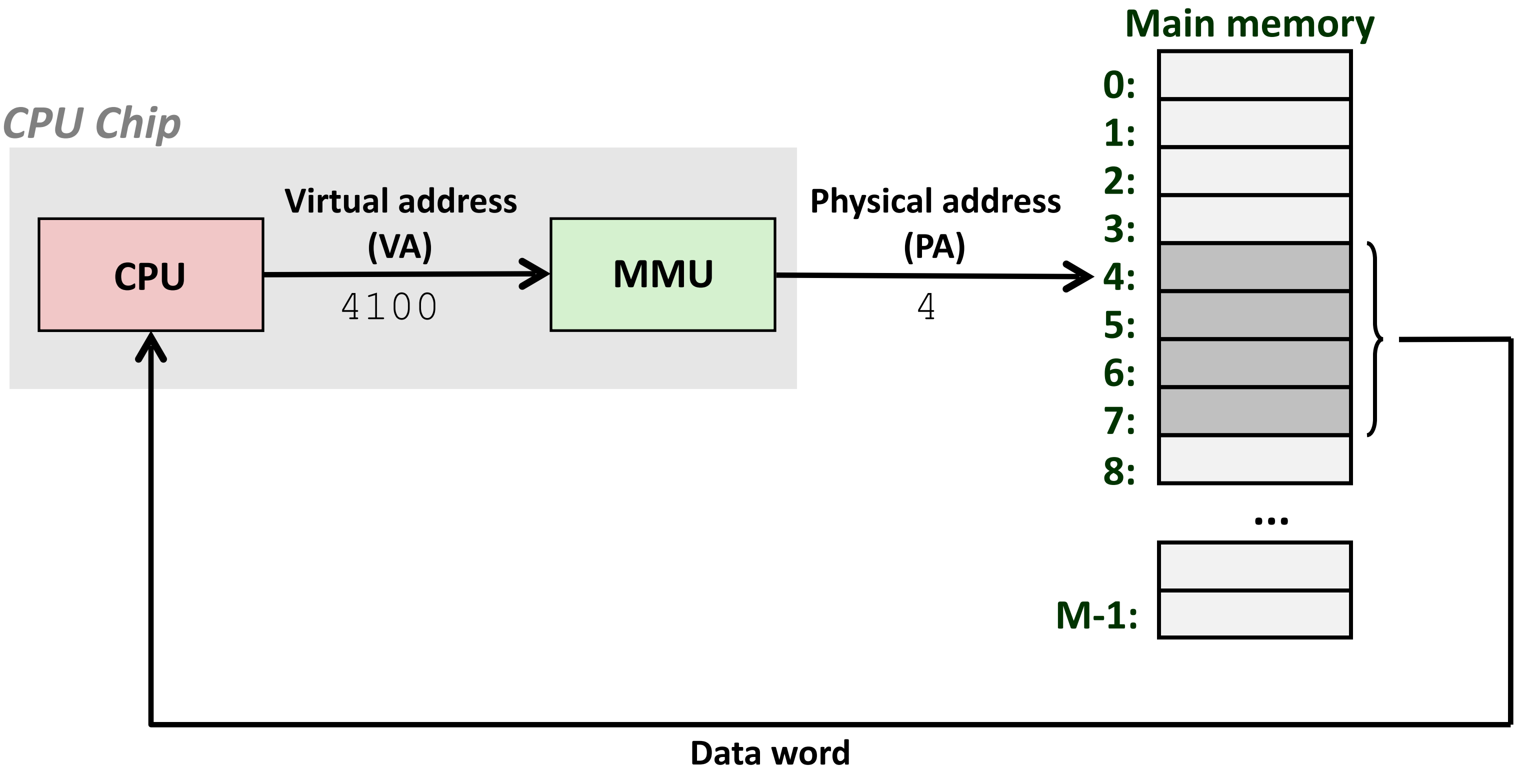


8	9	10	11	12	13	14
15	16 **Due**	17	18	19	20	21

A System Using Virtual Addressing



A System Using Virtual Addressing



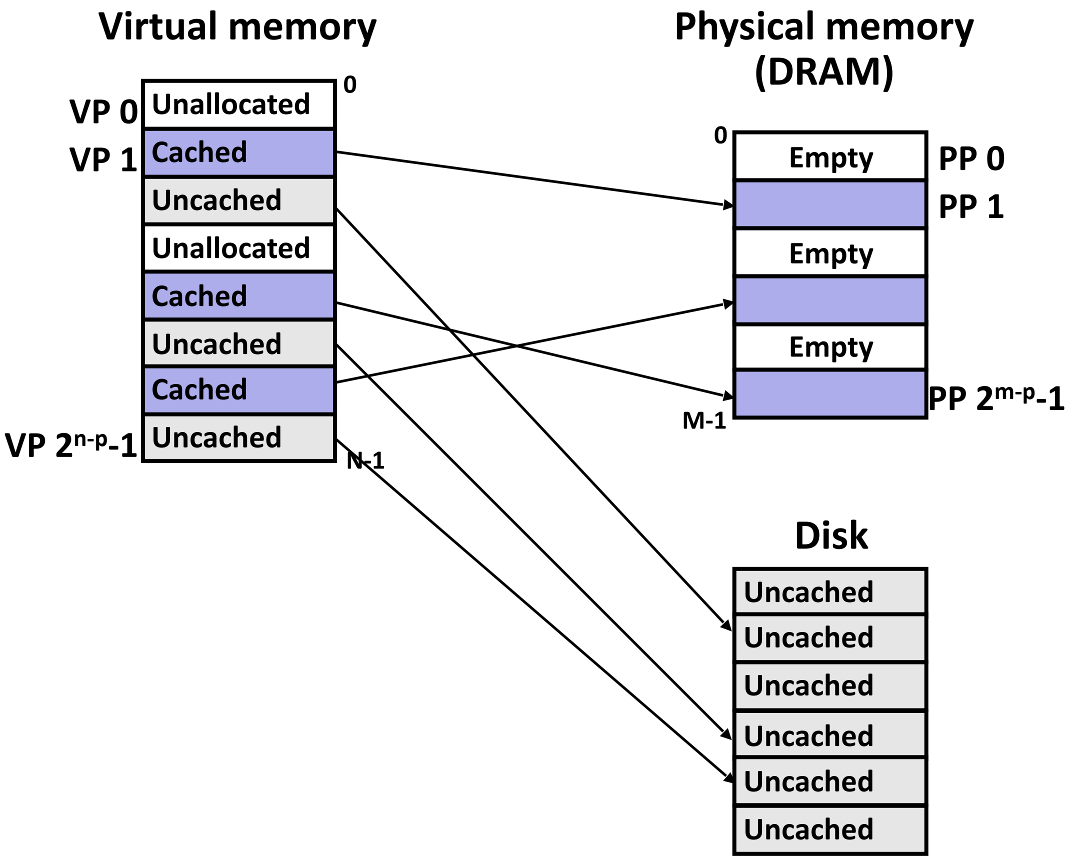
- On a 64-bit machine, virtual memory size = 2^{64}
- Physical memory size is much much smaller:
    - iPhone 8: 2 GB (2^{31})
    - 15-inch Macbook Pro 2017: 16 GB (2^{34})

VM Concepts

- Conceptually, *virtual memory* is an array of N *pages* stored on disk. The contents of the array on disk are cached in *physical memory*, which is an array of M pages (M << N).

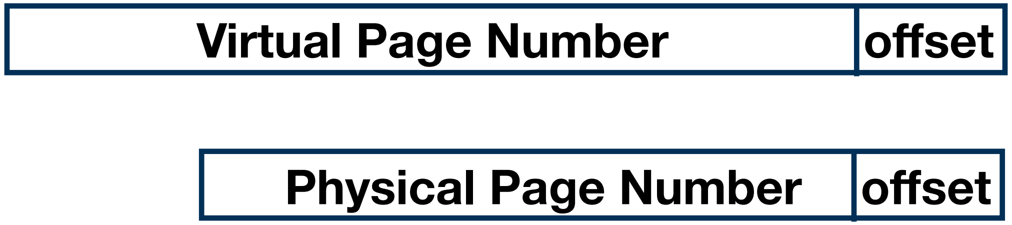
VM Concepts

- Conceptually, *virtual memory* is an array of N *pages* stored on disk. The contents of the array on disk are cached in *physical memory*, which is an array of M pages (M << N).



VM Concepts

- Page size is the same in VM and PM

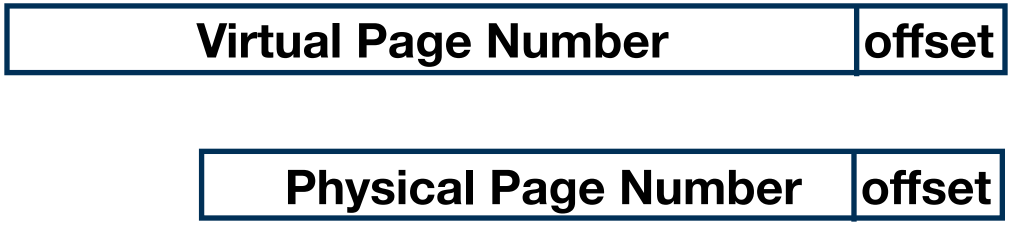


Virtual Page Number | offset

Physical Page Number | offset

VM Concepts

- Page size is the same in VM and PM
- In a 64-bit machine, VA is 64-bit long. Assuming PM is 4 GB. Assuming 4KB page size.

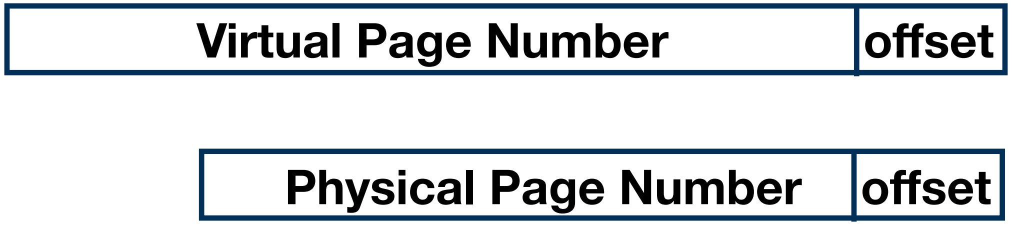


Virtual Page Number | offset

Physical Page Number | offset

VM Concepts

- Page size is the same in VM and PM
- In a 64-bit machine, VA is 64-bit long. Assuming PM is 4 GB. Assuming 4KB page size.
- How many bits for offset?
    - 12. Same for VM and PM

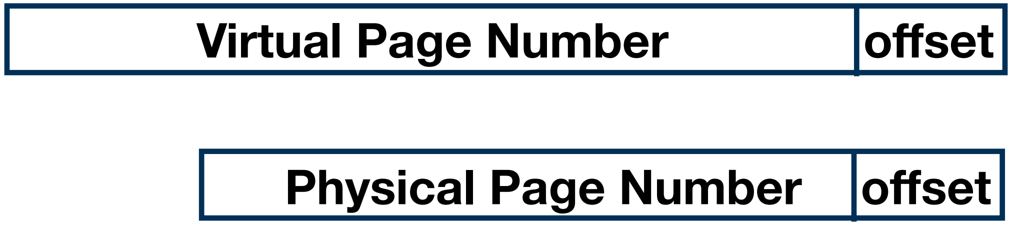


Virtual Page Number | offset

Physical Page Number | offset

VM Concepts

- Page size is the same in VM and PM
- In a 64-bit machine, VA is 64-bit long. Assuming PM is 4 GB. Assuming 4KB page size.
- How many bits for offset?
    - 12. Same for VM and PM
- How many bits for Virtual Page Number?
    - 52

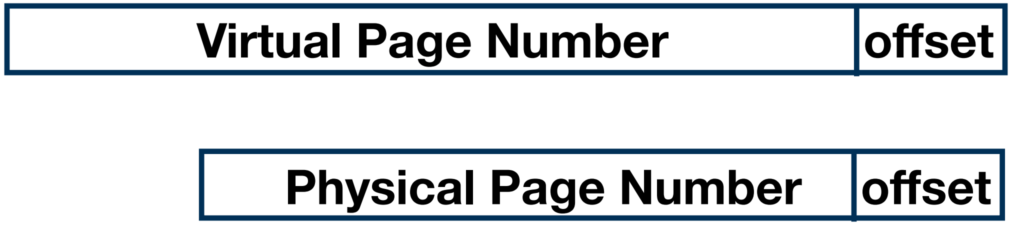


Virtual Page Number | offset

Physical Page Number | offset

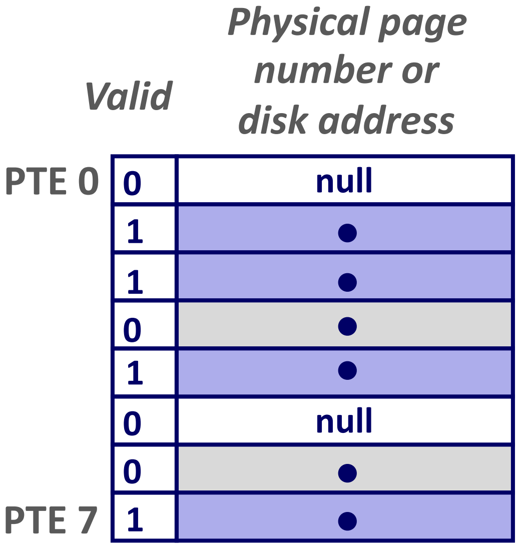
VM Concepts

- Page size is the same in VM and PM
- In a 64-bit machine, VA is 64-bit long. Assuming PM is 4 GB. Assuming 4KB page size.
- How many bits for offset?
  - 12. Same for VM and PM
- How many bits for Virtual Page Number?
  - 52
- How many bits for Physical Page Number?
  - 20



Page Table: Enabling VA to PA Translation

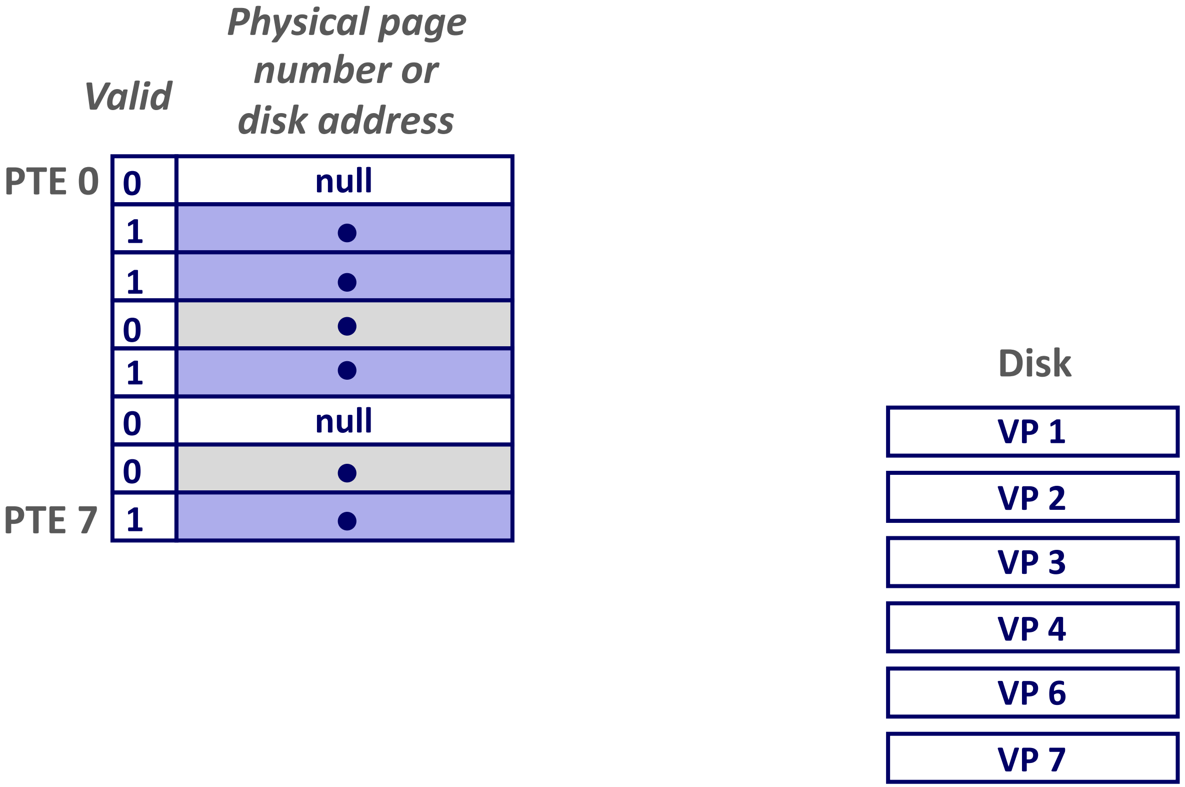
- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.



	Valid	*Physical page number or disk address*
PTE 0	0	null
	1	●
	1	●
	0	●
	1	●
	0	null
	0	●
PTE 7	1	●

Page Table: Enabling VA to PA Translation

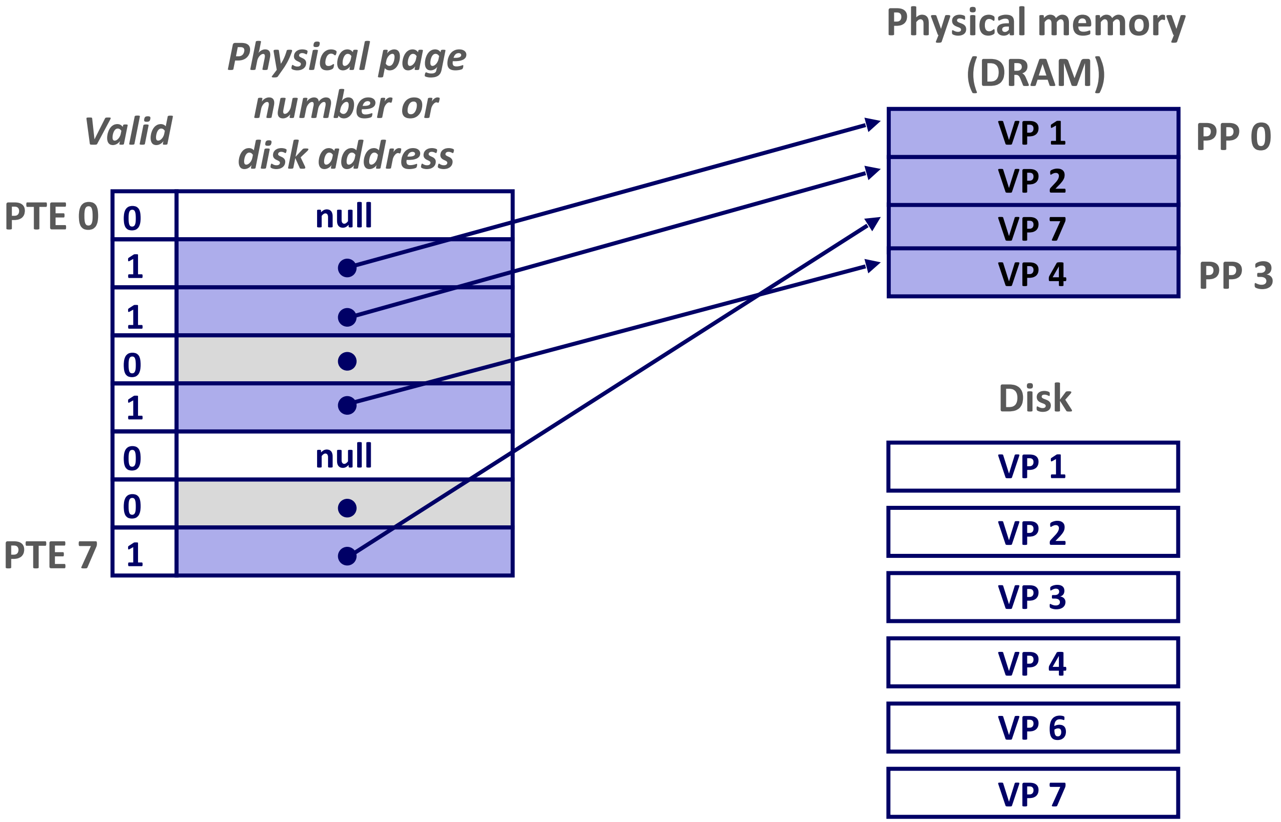
- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.



	Valid	Physical page number or disk address
PTE 0	0	null
	1	●
	1	●
	0	●
	1	●
	0	null
	0	●
PTE 7	1	●

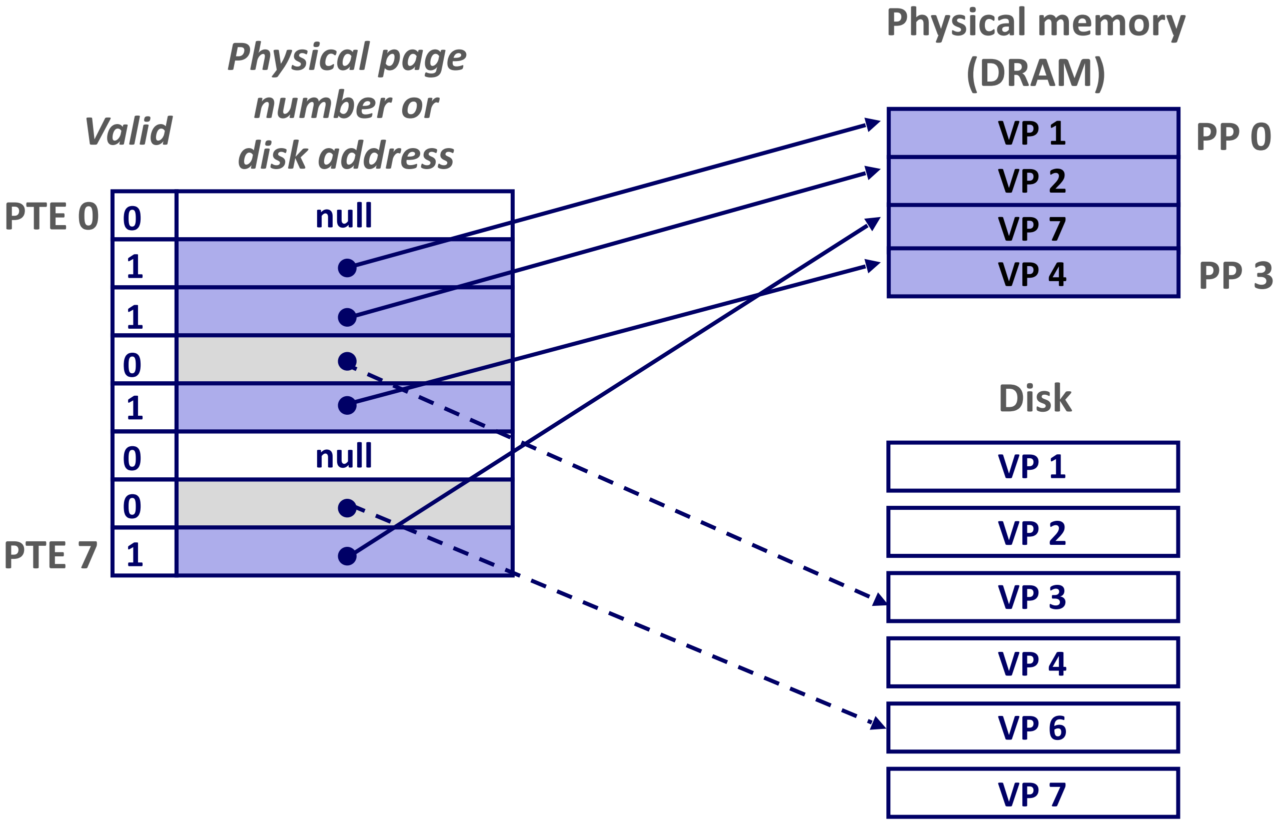
Page Table: Enabling VA to PA Translation

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.



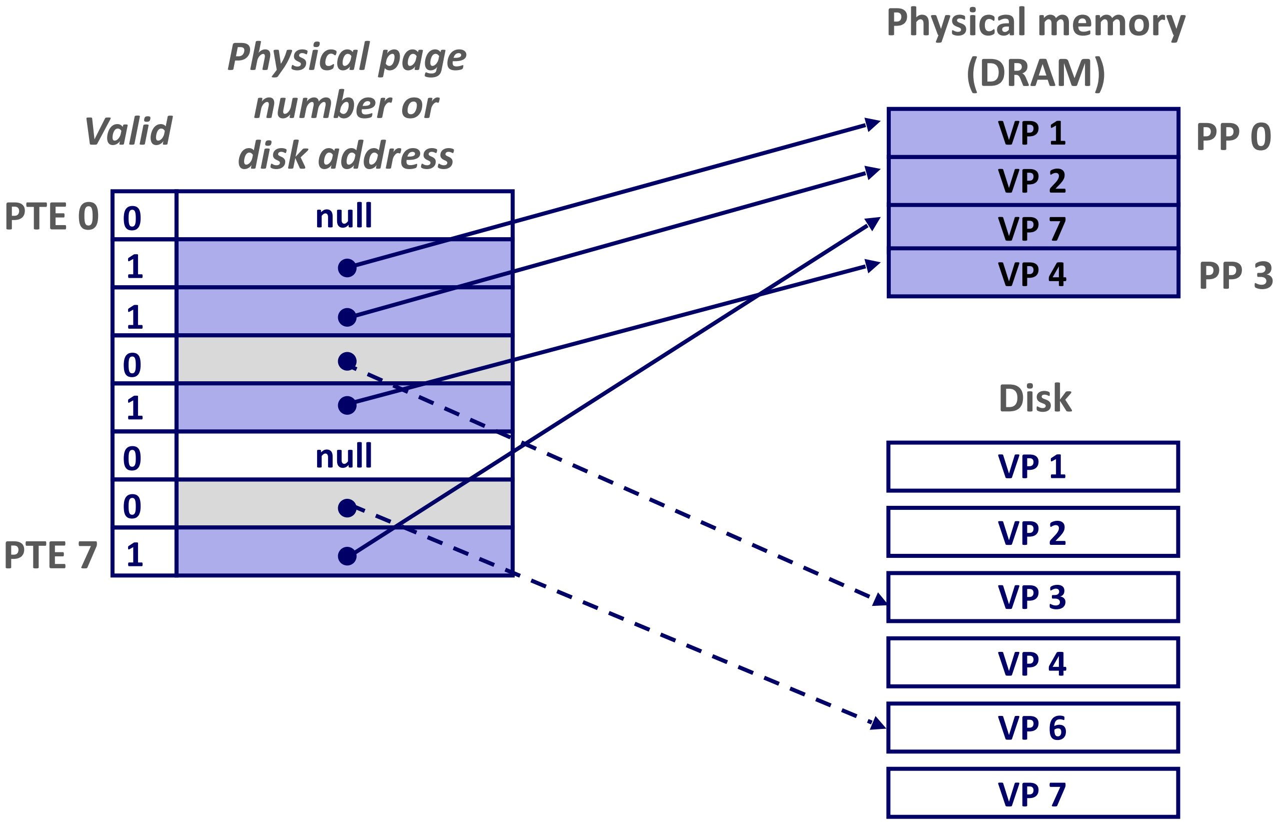
Page Table: Enabling VA to PA Translation

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.

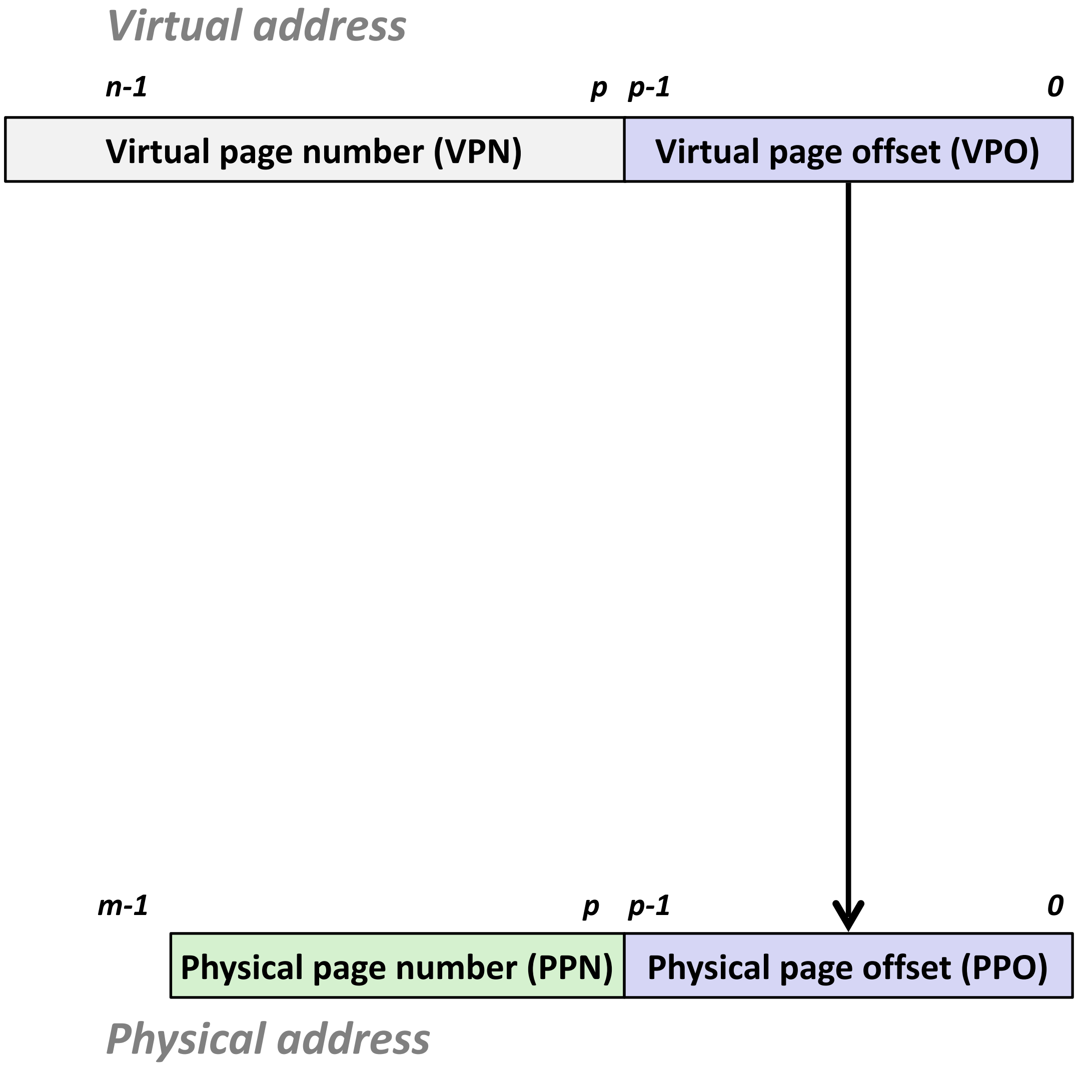


Page Table: Enabling VA to PA Translation

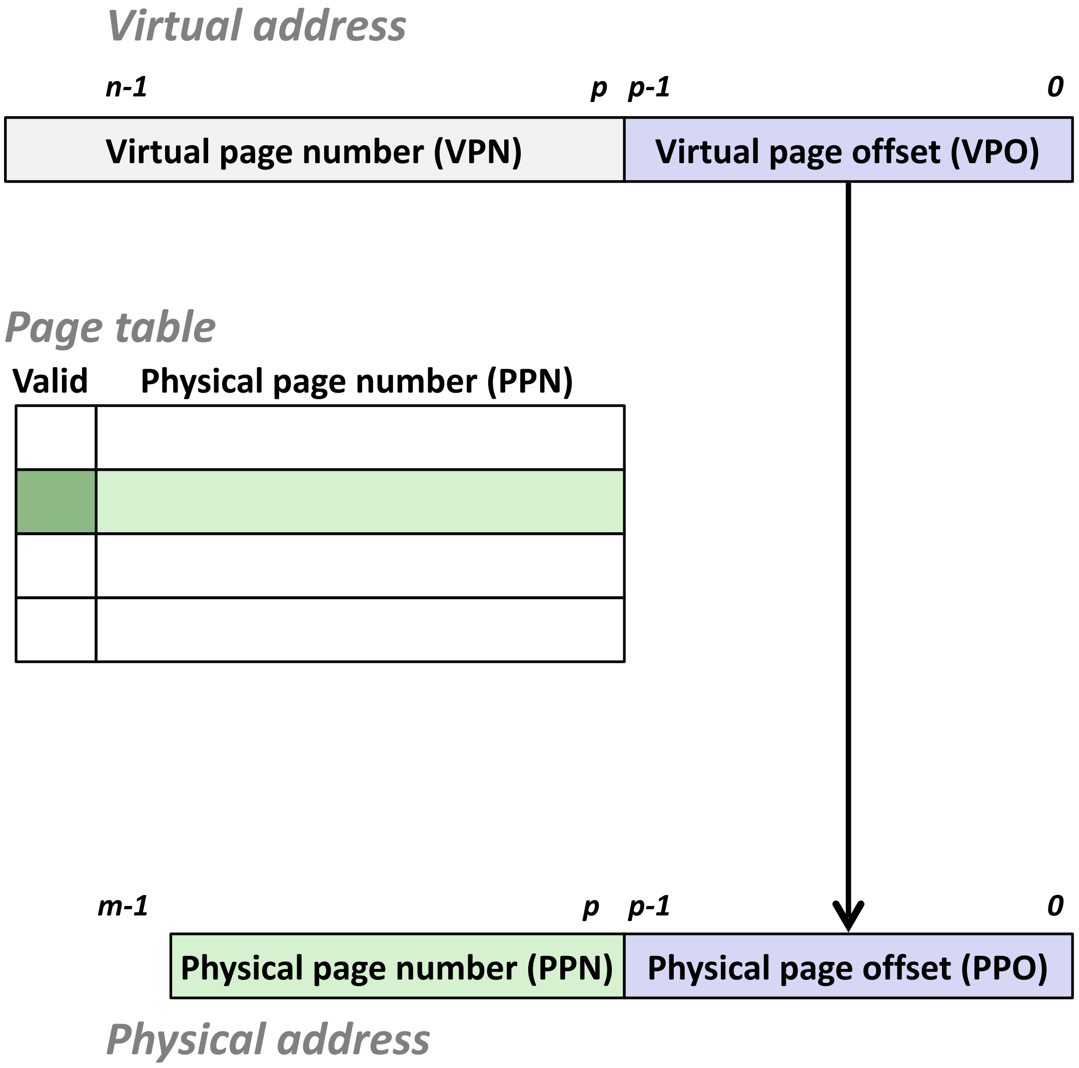
- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.
- 64-bit machine, 4KB page size, how many PTEs?
  - Every page has a PTE, so 2^{52} PTEs.



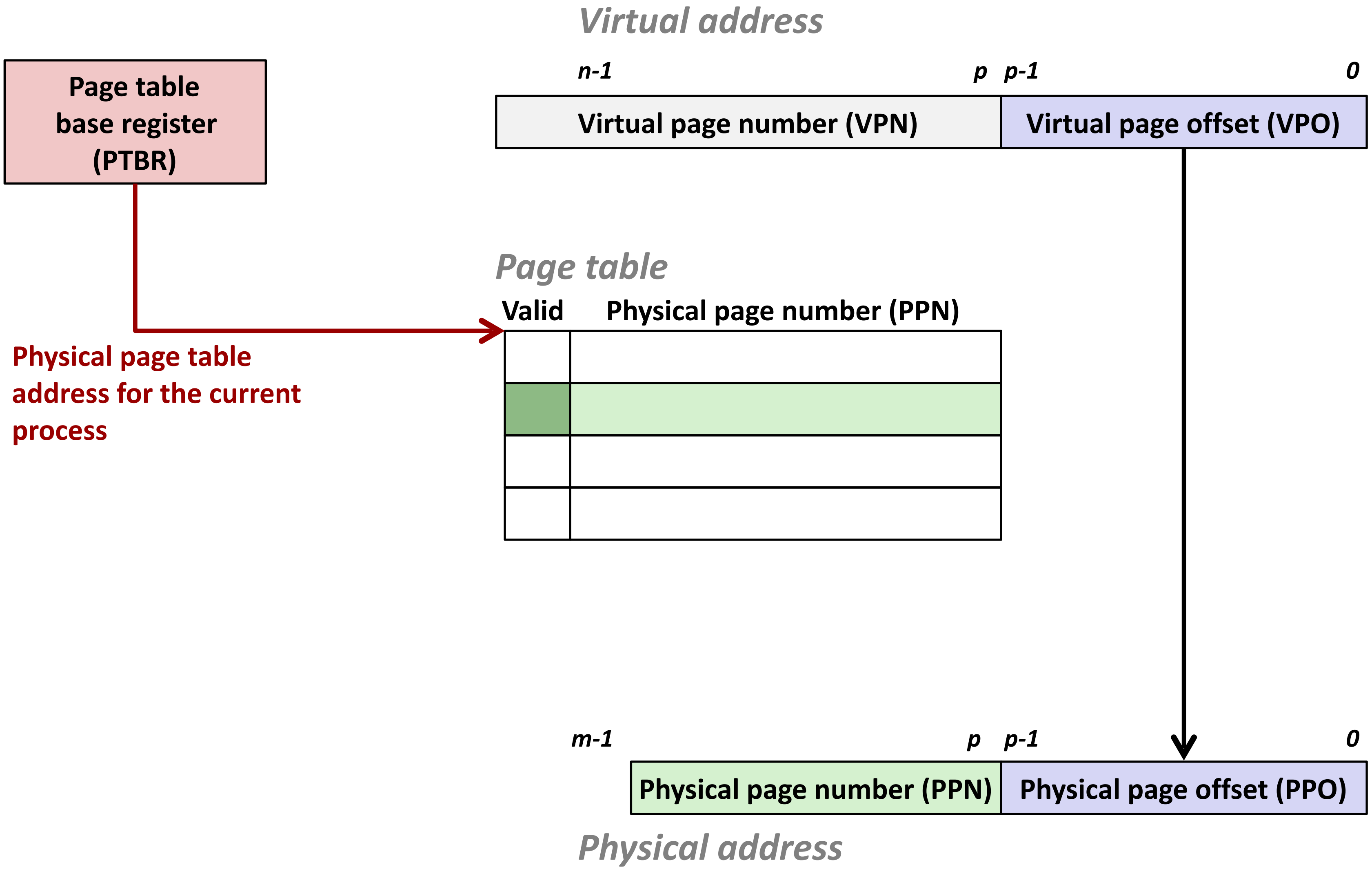
Address Translation With a Page Table


Virtual address
n-1
p
p-1
0
Virtual page number (VPN)
Virtual page offset (VPO)
m-1
p
p-1
0
Physical page number (PPN)
Physical page offset (PPO)
Physical address

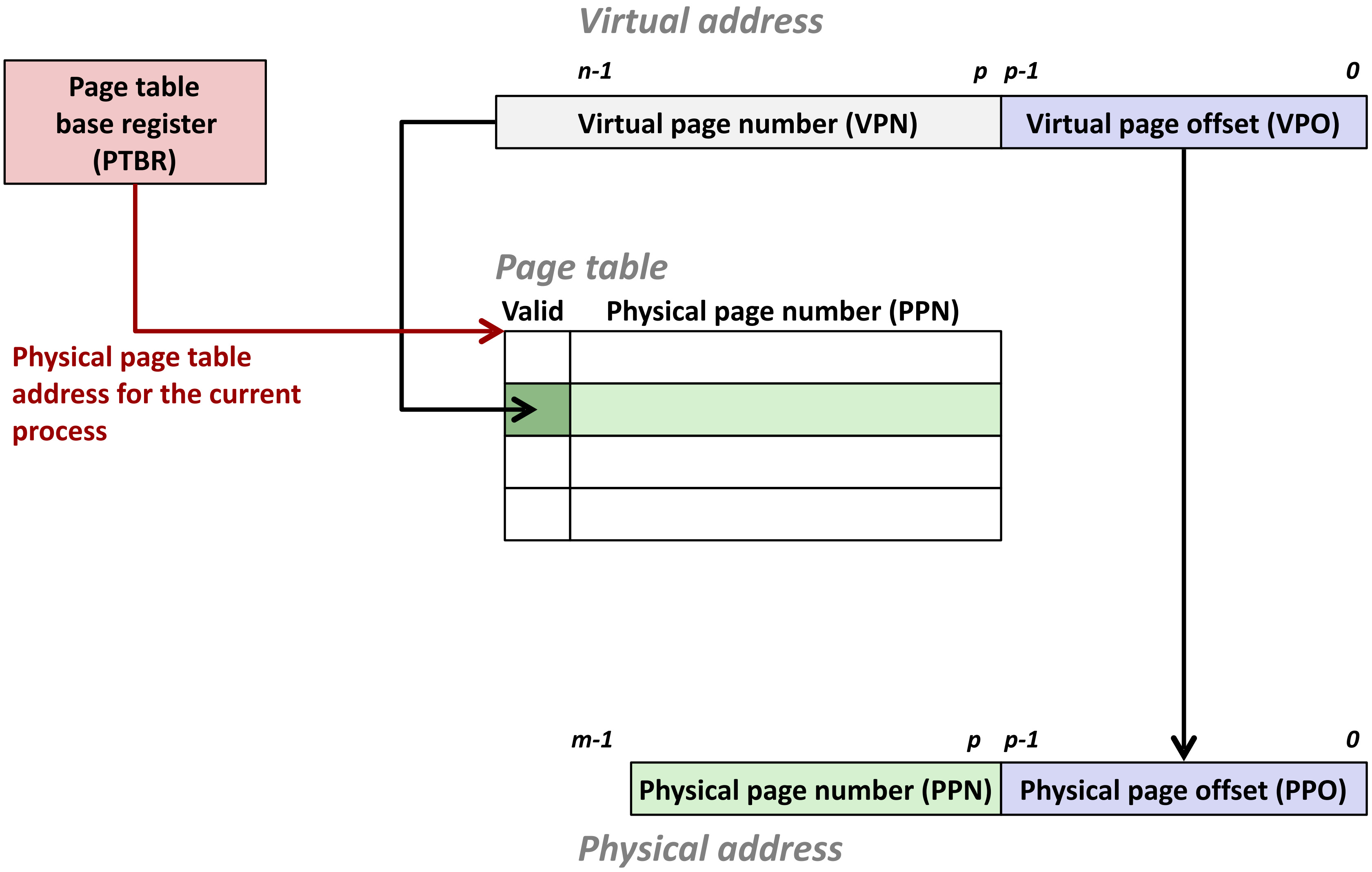
Address Translation With a Page Table


Virtual address
n-1
p
p-1
0
Virtual page number (VPN)
Virtual page offset (VPO)
Page table
Valid
Physical page number (PPN)
m-1
p
p-1
0
Physical page number (PPN)
Physical page offset (PPO)
Physical address

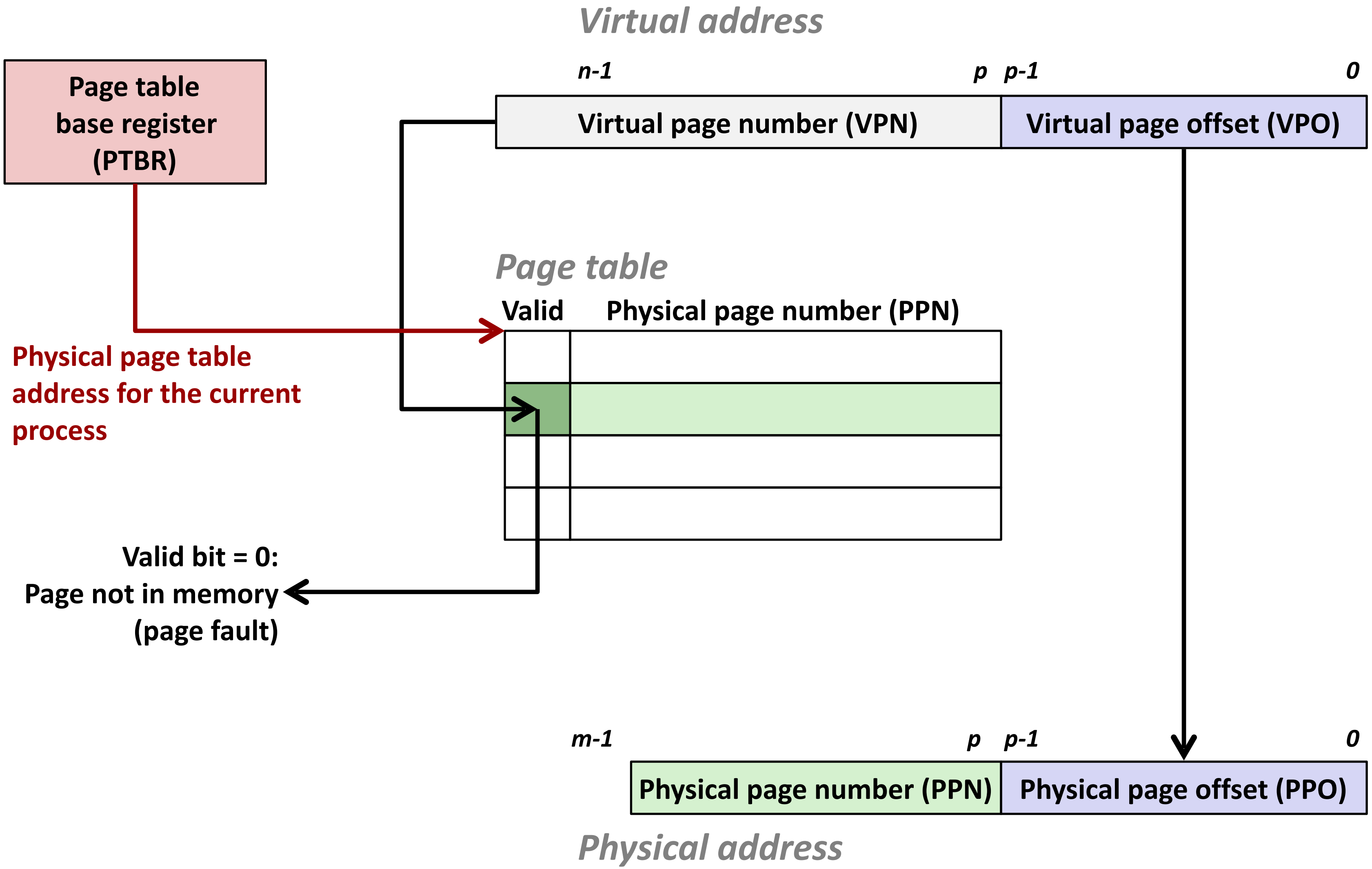
Address Translation With a Page Table



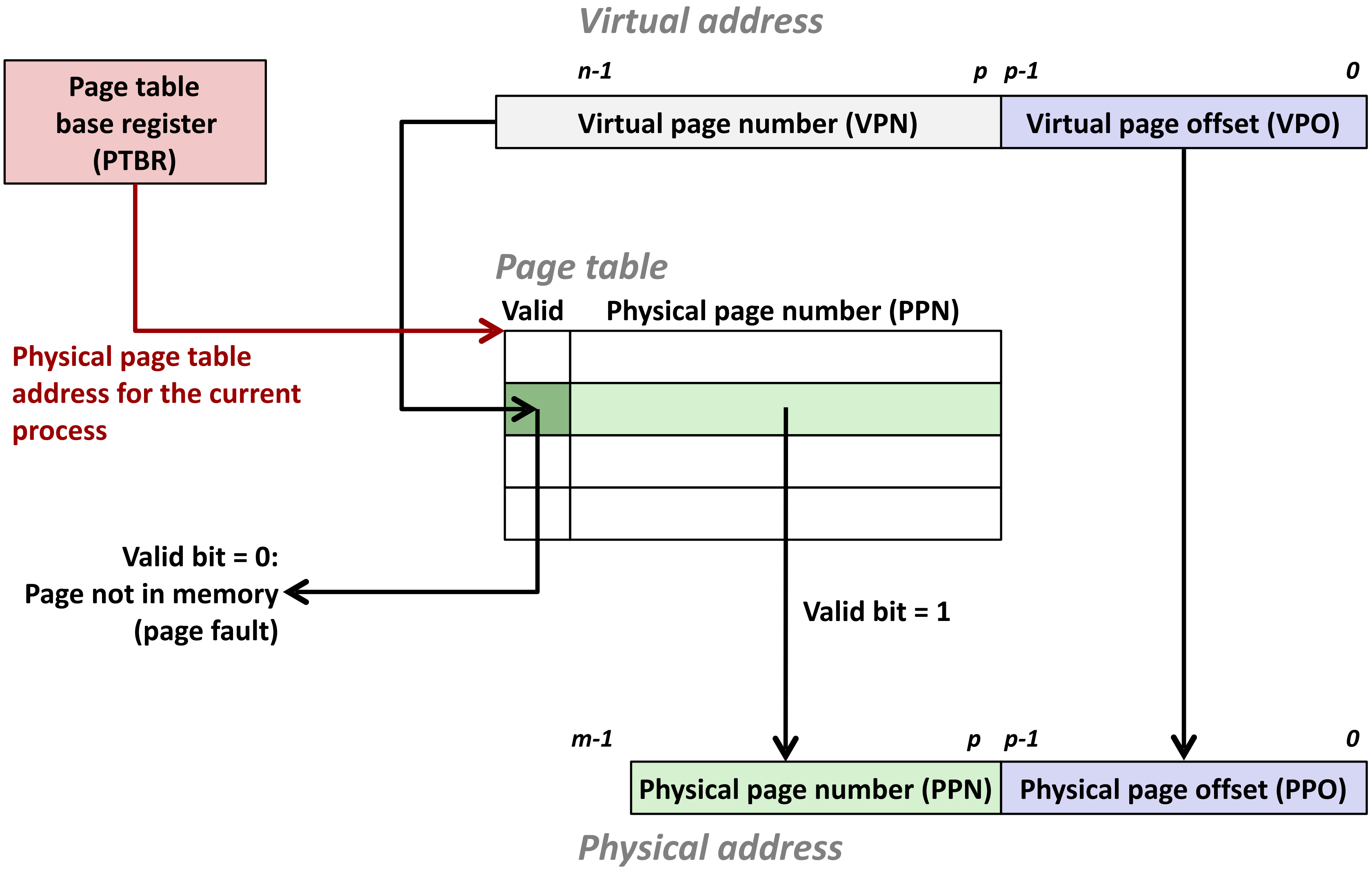
Address Translation With a Page Table


Virtual address
n-1
p
p-1
0
Page table base register (PTBR)
Virtual page number (VPN)
Virtual page offset (VPO)
Page table
Valid
Physical page number (PPN)
Physical page table address for the current process
m-1
p
p-1
0
Physical page number (PPN)
Physical page offset (PPO)
Physical address

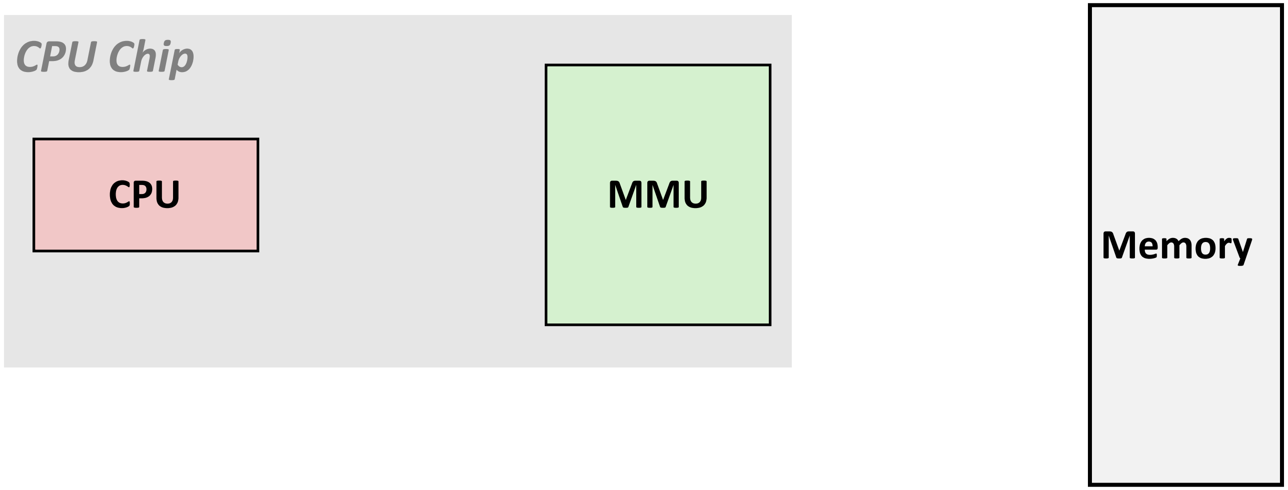
Address Translation With a Page Table


Virtual address
n-1
p
p-1
0
Page table base register (PTBR)
Virtual page number (VPN)
Virtual page offset (VPO)
Page table
Valid
Physical page number (PPN)
Physical page table address for the current process
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

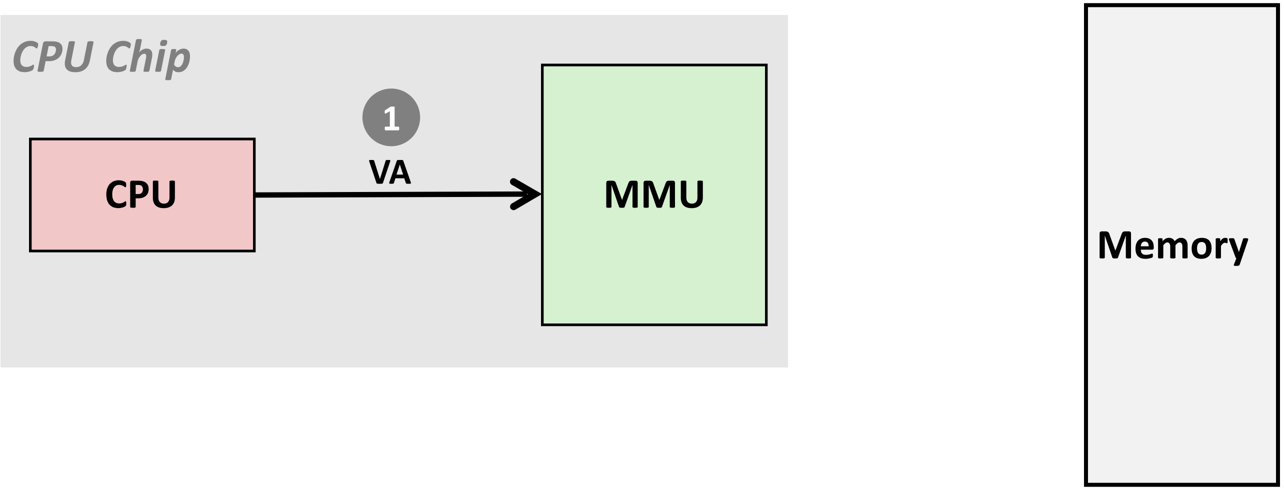
Address Translation With a Page Table


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

Address Translation: Page Hit

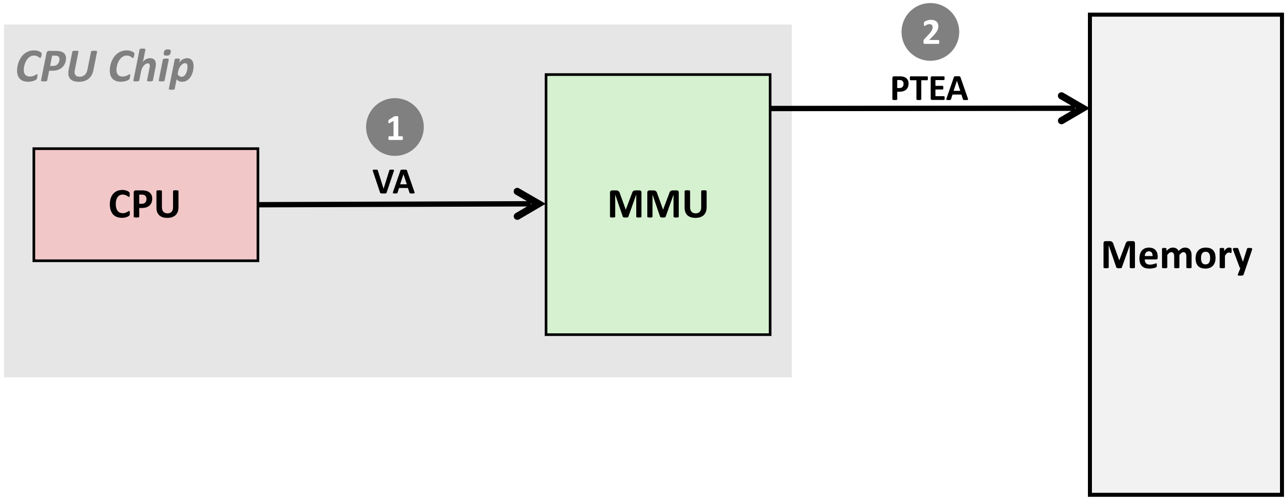


Address Translation: Page Hit



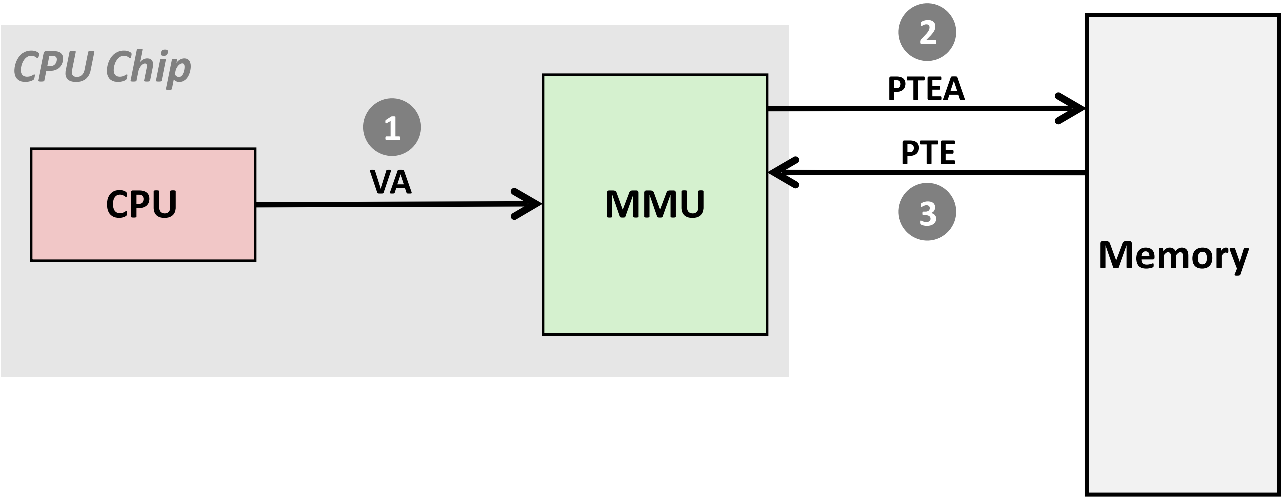
1) Processor sends virtual address to MMU

Address Translation: Page Hit



1) Processor sends virtual address to MMU

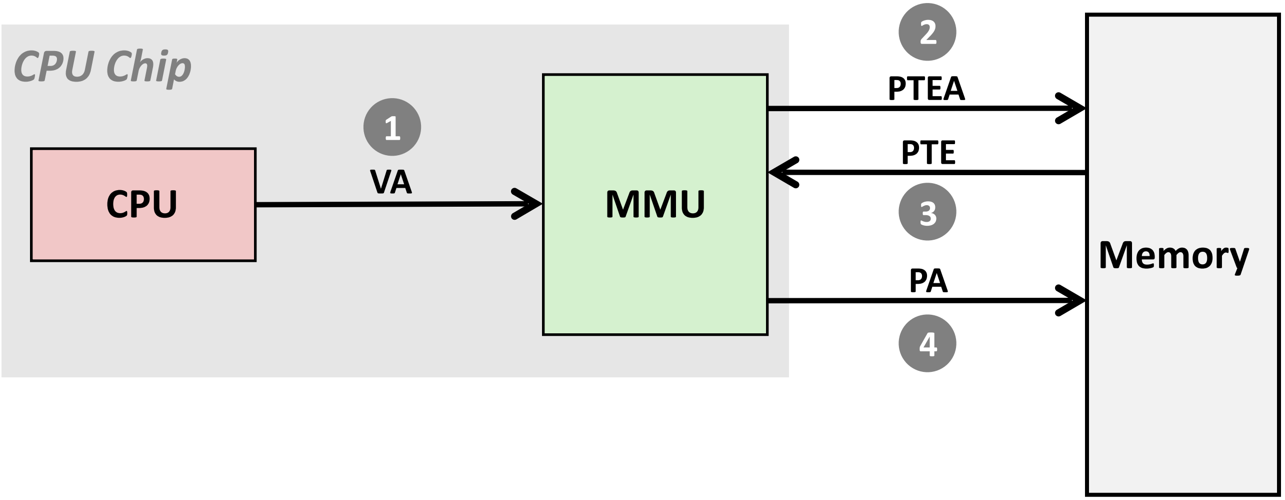
Address Translation: Page Hit



1) Processor sends virtual address to MMU

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

Address Translation: Page Hit

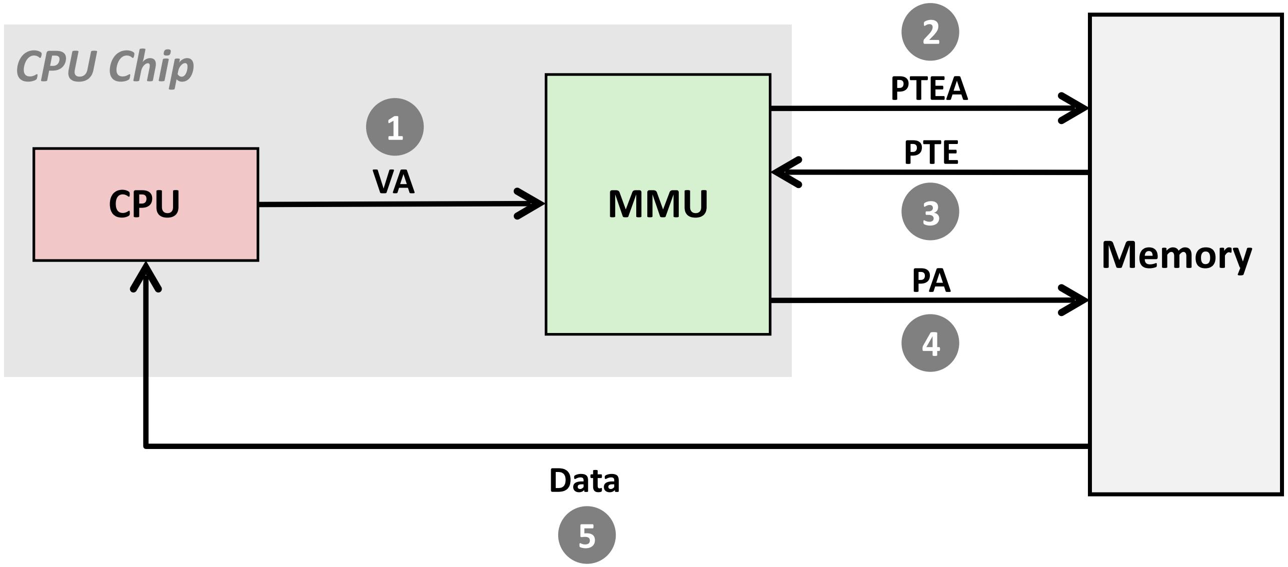


1) Processor sends virtual address to MMU

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

4) MMU sends physical address to cache/memory

Address Translation: Page Hit



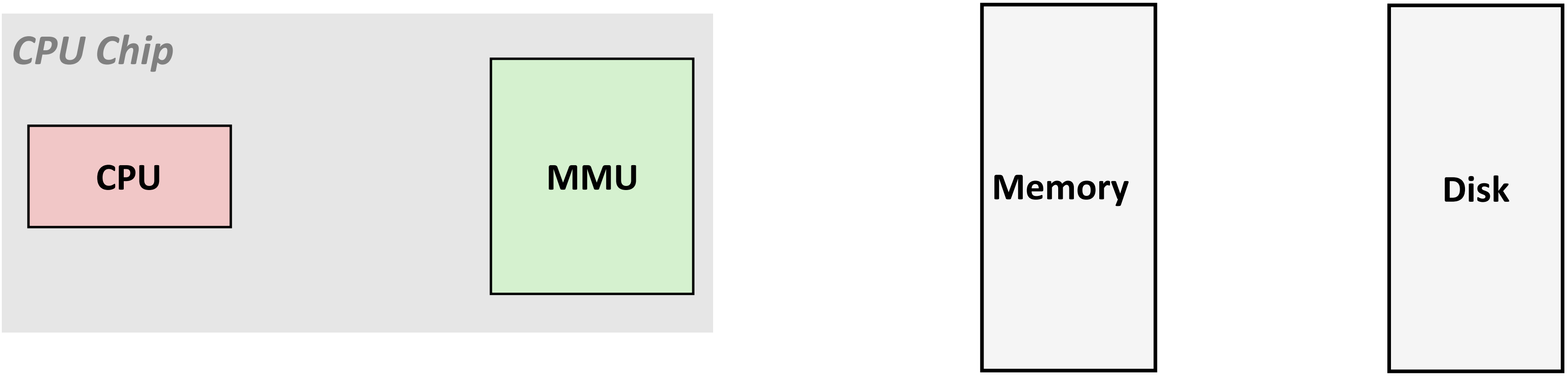
1) Processor sends virtual address to MMU

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

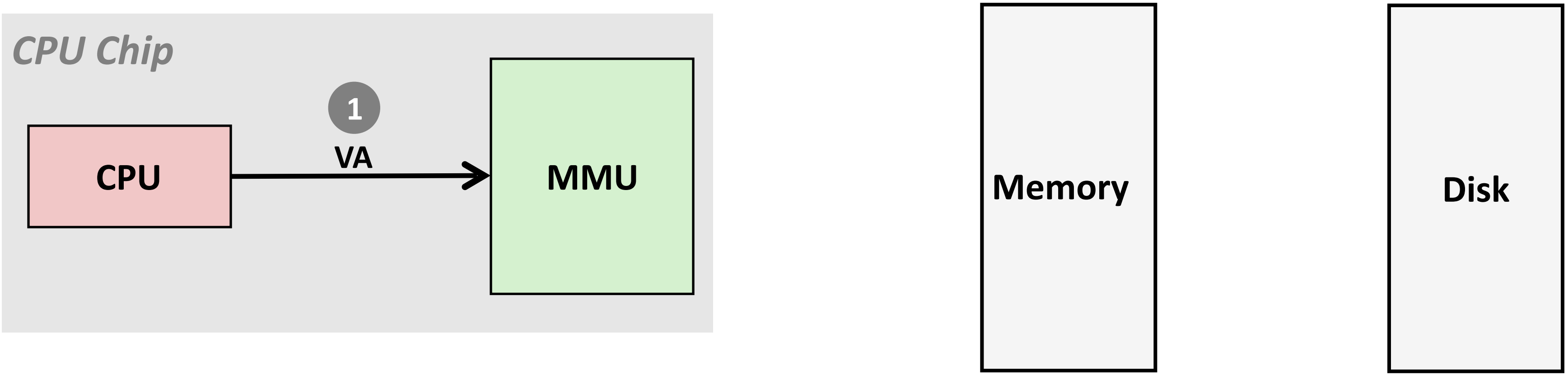
4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

Address Translation: Page Fault

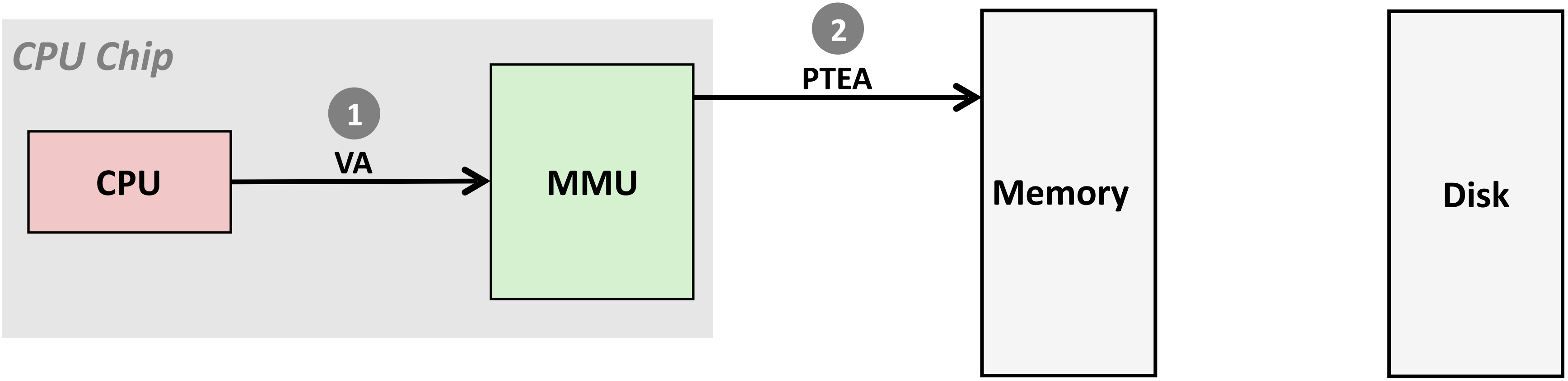


Address Translation: Page Fault



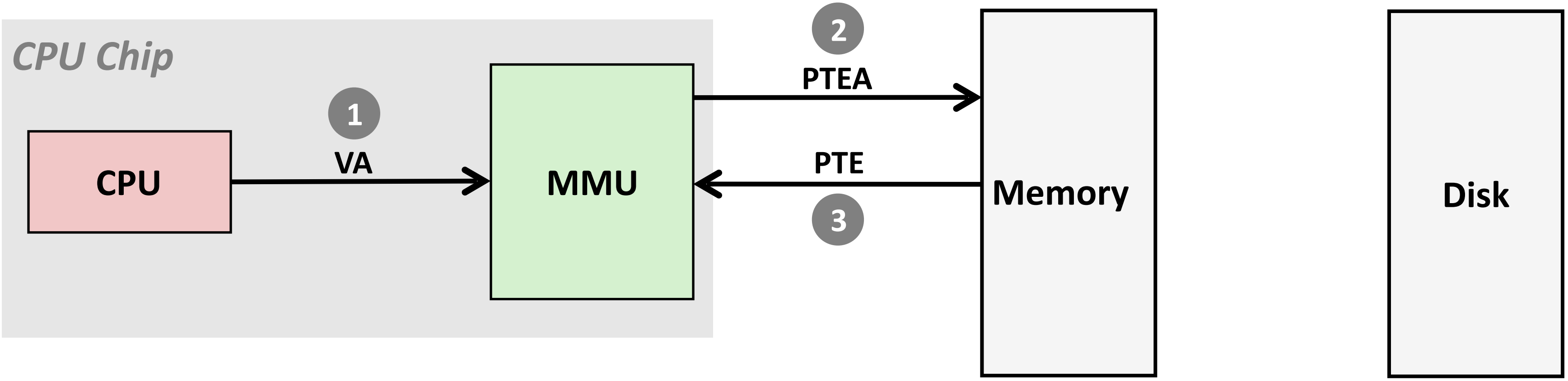
1) Processor sends virtual address to MMU

Address Translation: Page Fault



1) Processor sends virtual address to MMU

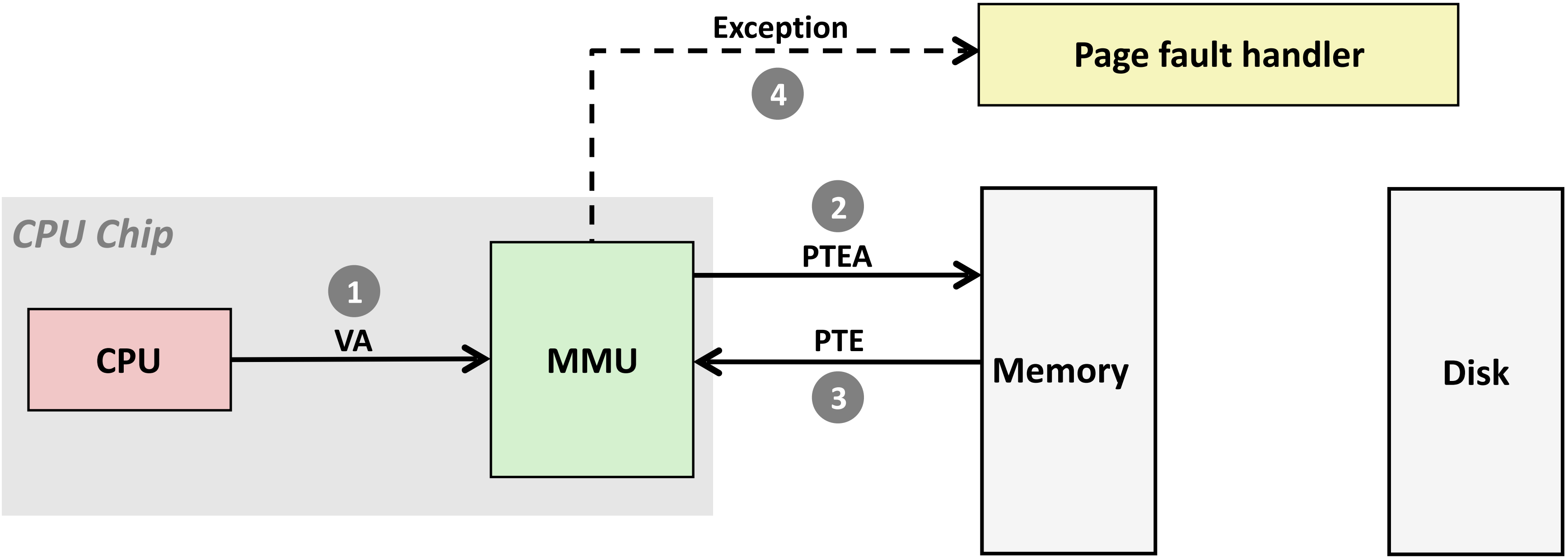
Address Translation: Page Fault



1) Processor sends virtual address to MMU

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

Address Translation: Page Fault

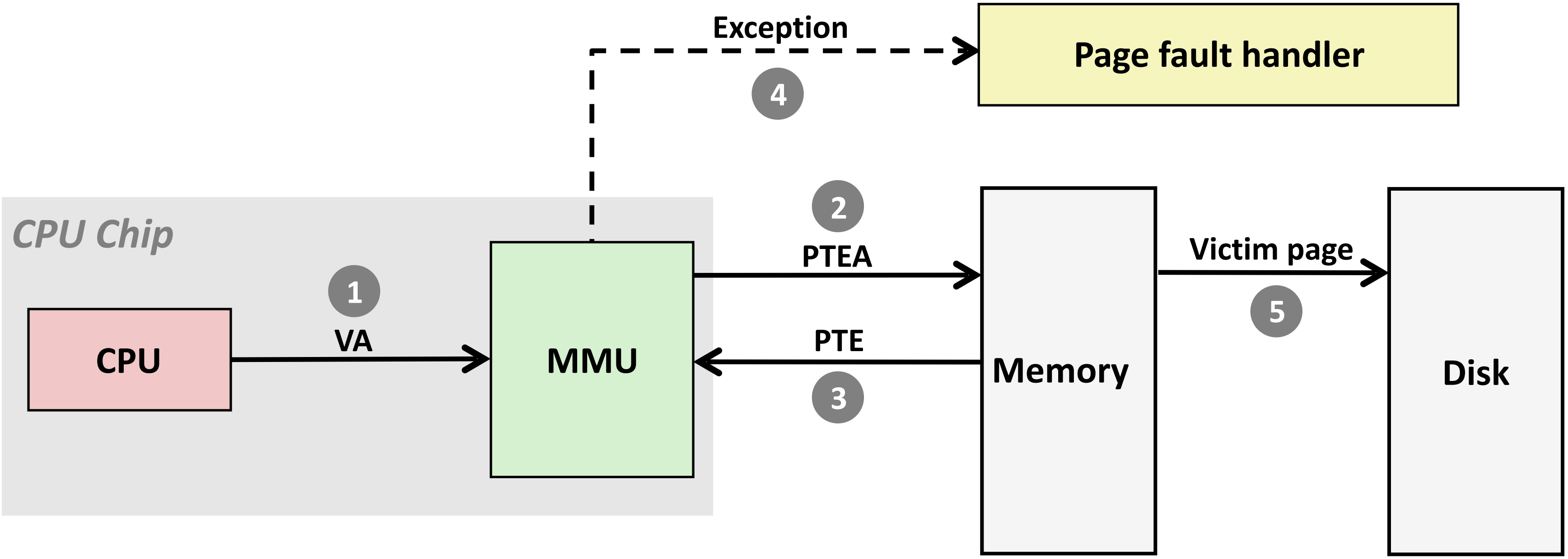


1) Processor sends virtual address to MMU

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

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

Address Translation: Page Fault



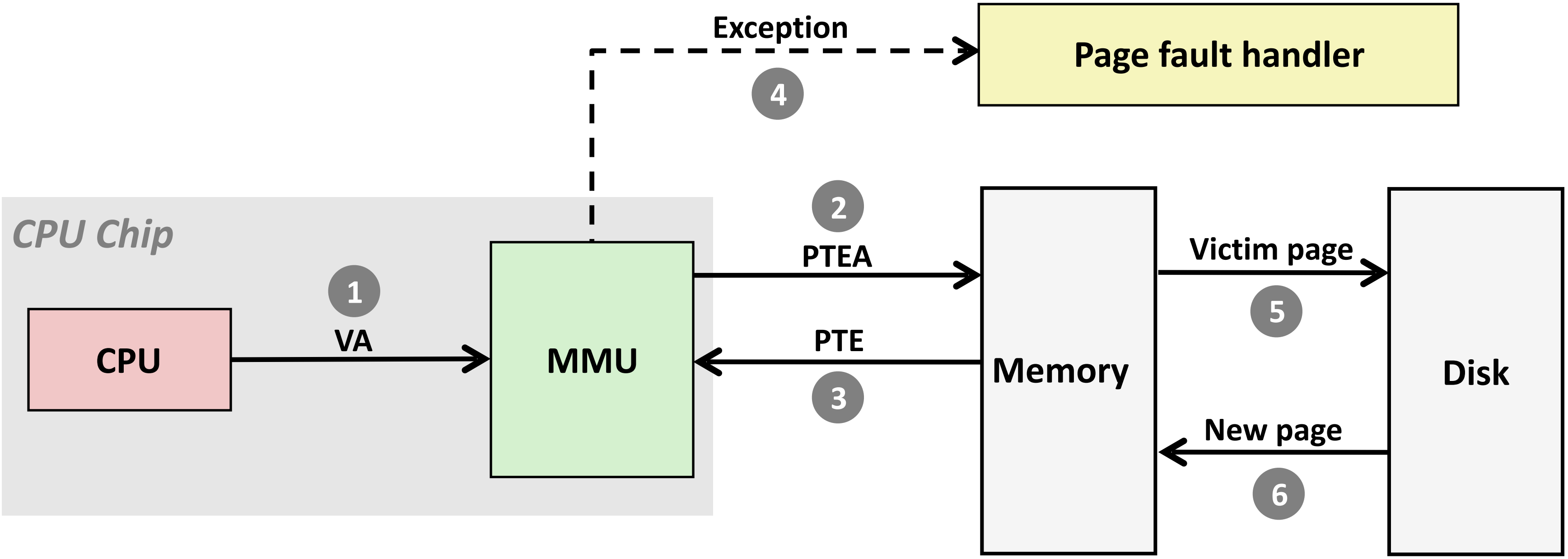
1) Processor sends virtual address to MMU

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)

Address Translation: Page Fault



1) Processor sends virtual address to MMU

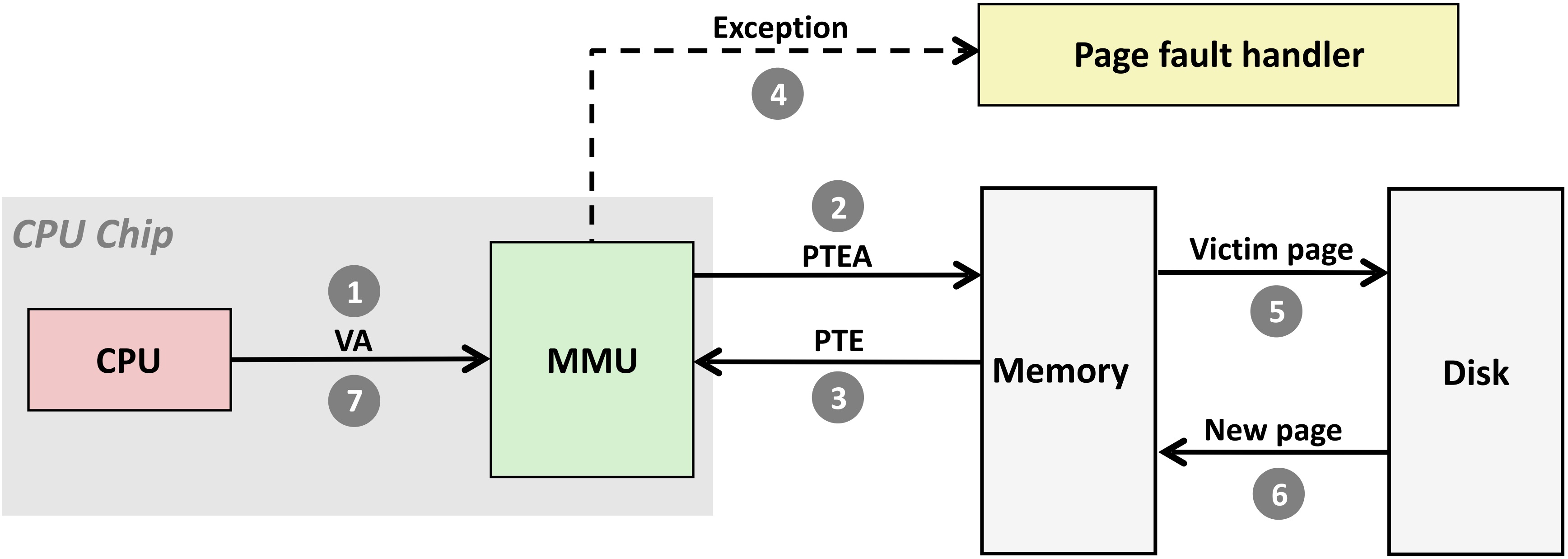
2-3) MMU fetches PTE from page table in memory

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

Address Translation: Page Fault



1) Processor sends virtual address to MMU

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

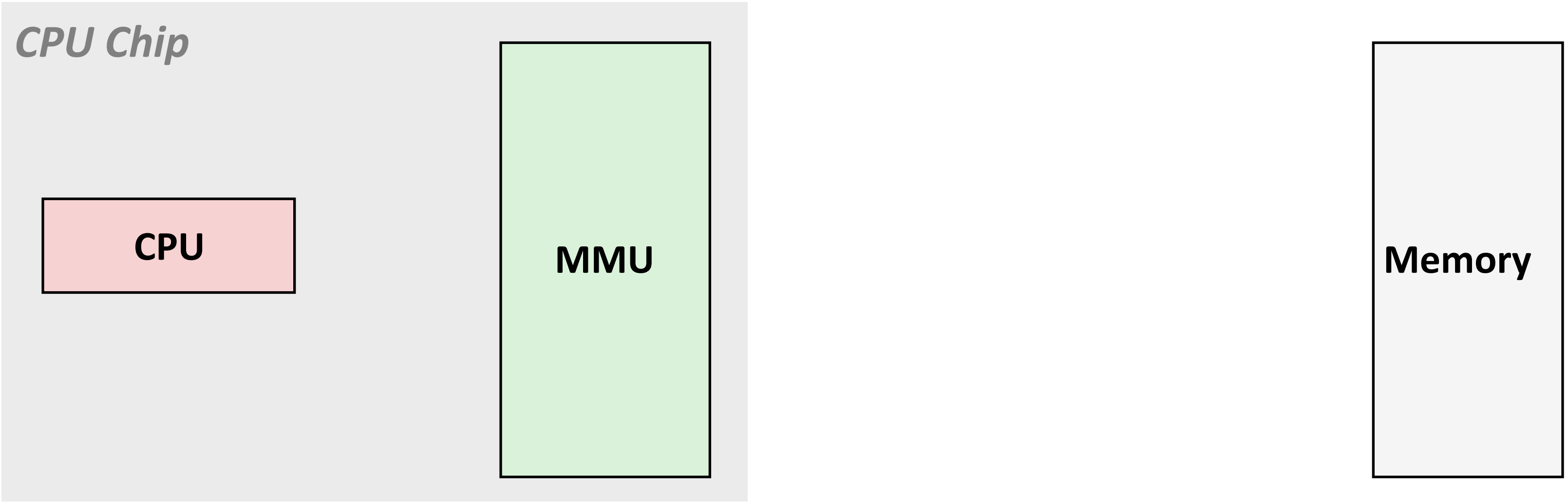
4) Valid bit is zero, so MMU triggers page fault exception

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

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

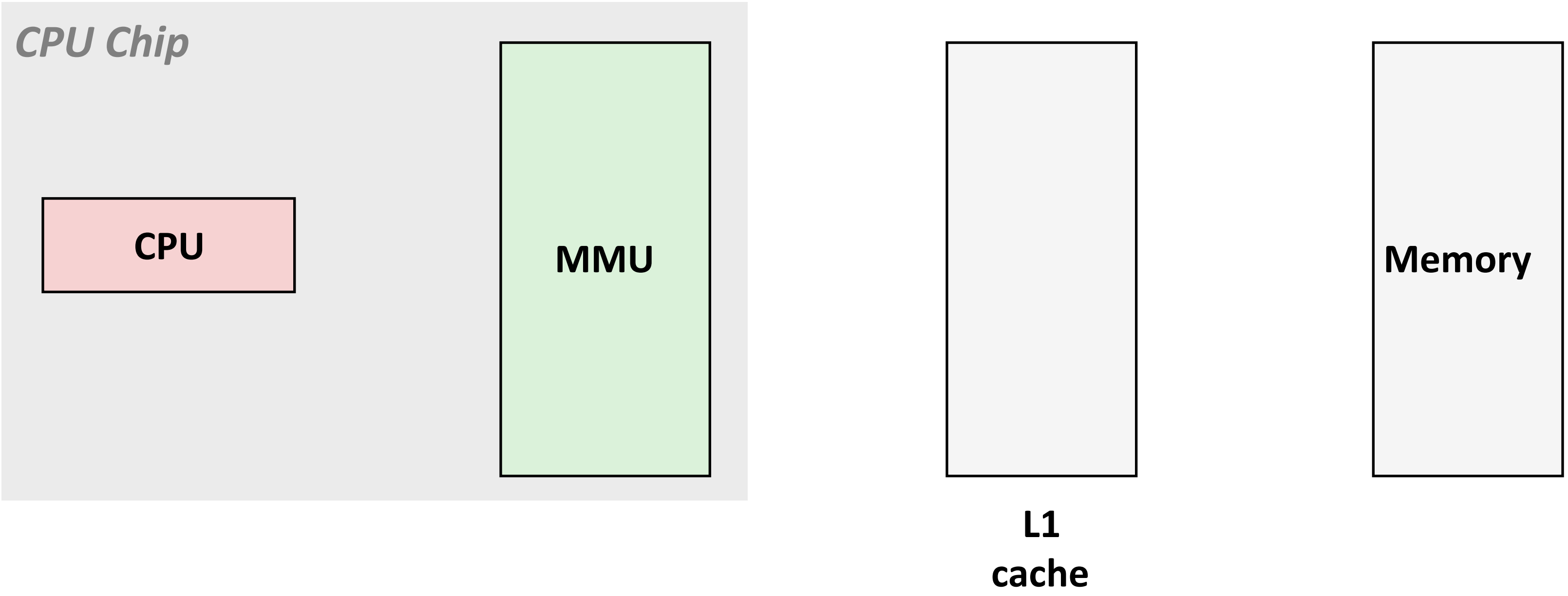
7) Handler returns to original process, restarting faulting instruction

Integrating VM and Cache



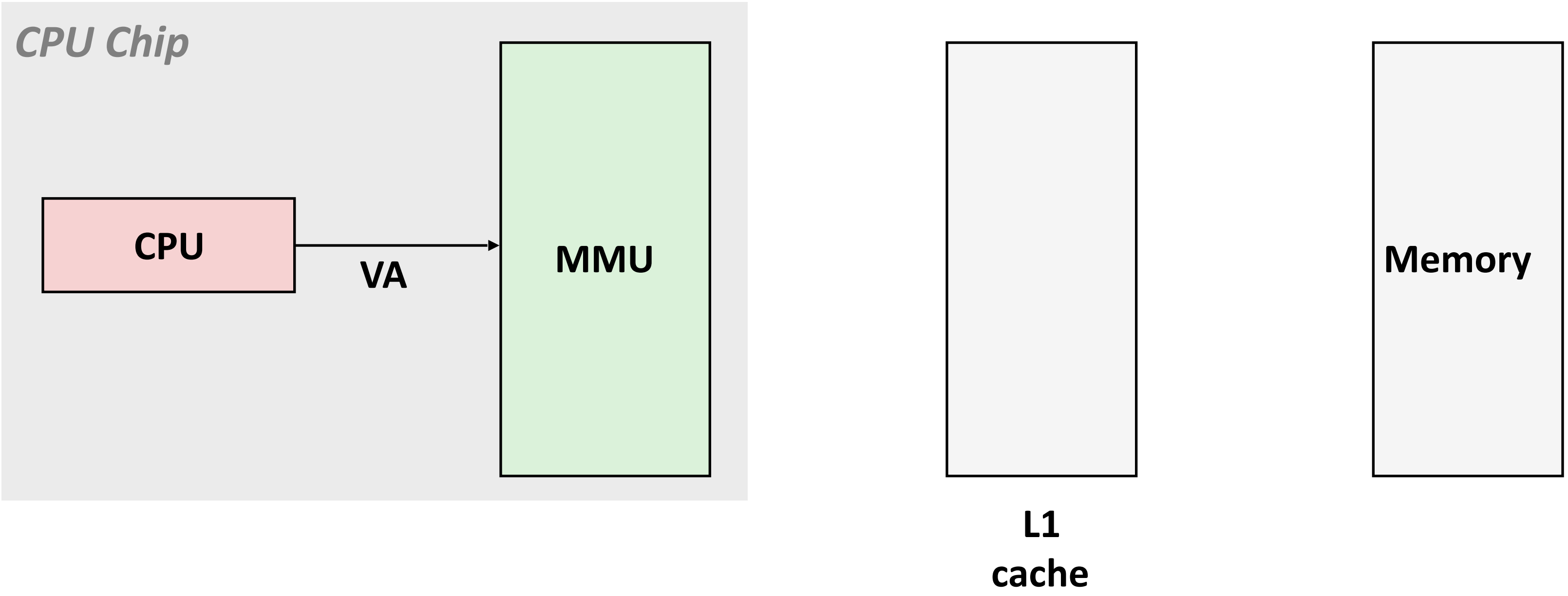
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



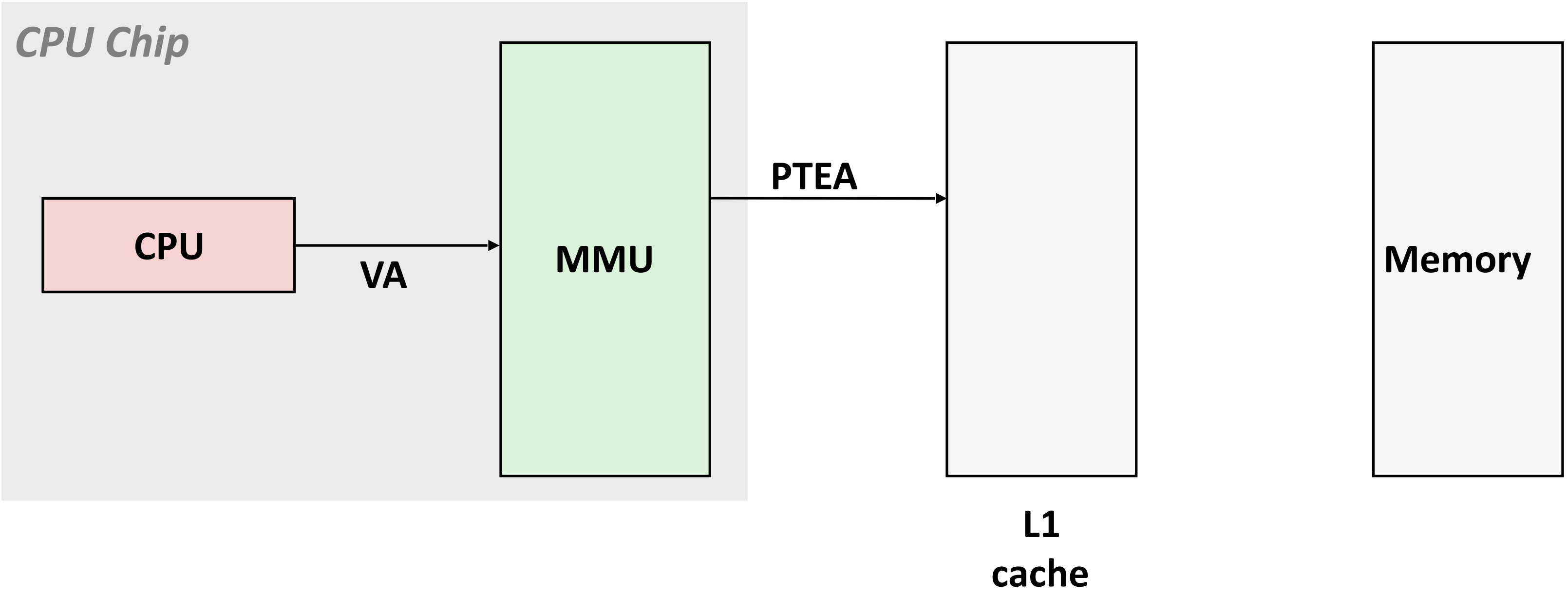
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



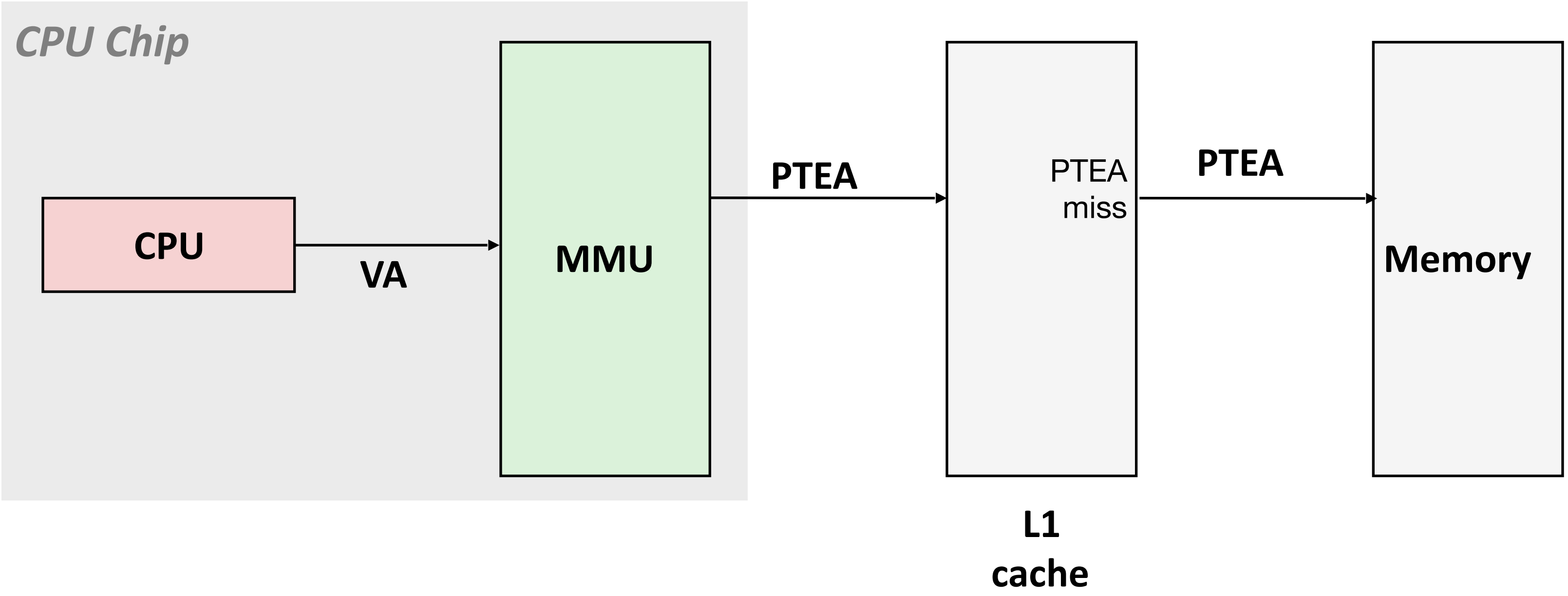
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



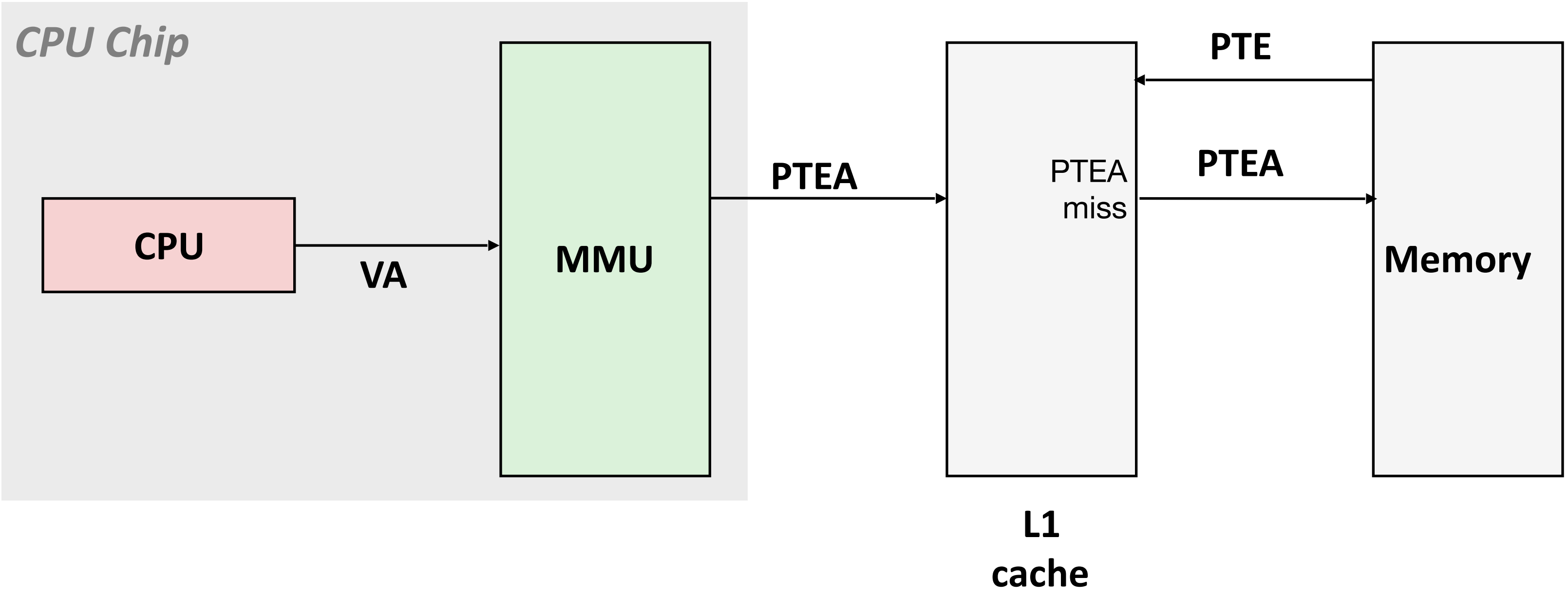
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



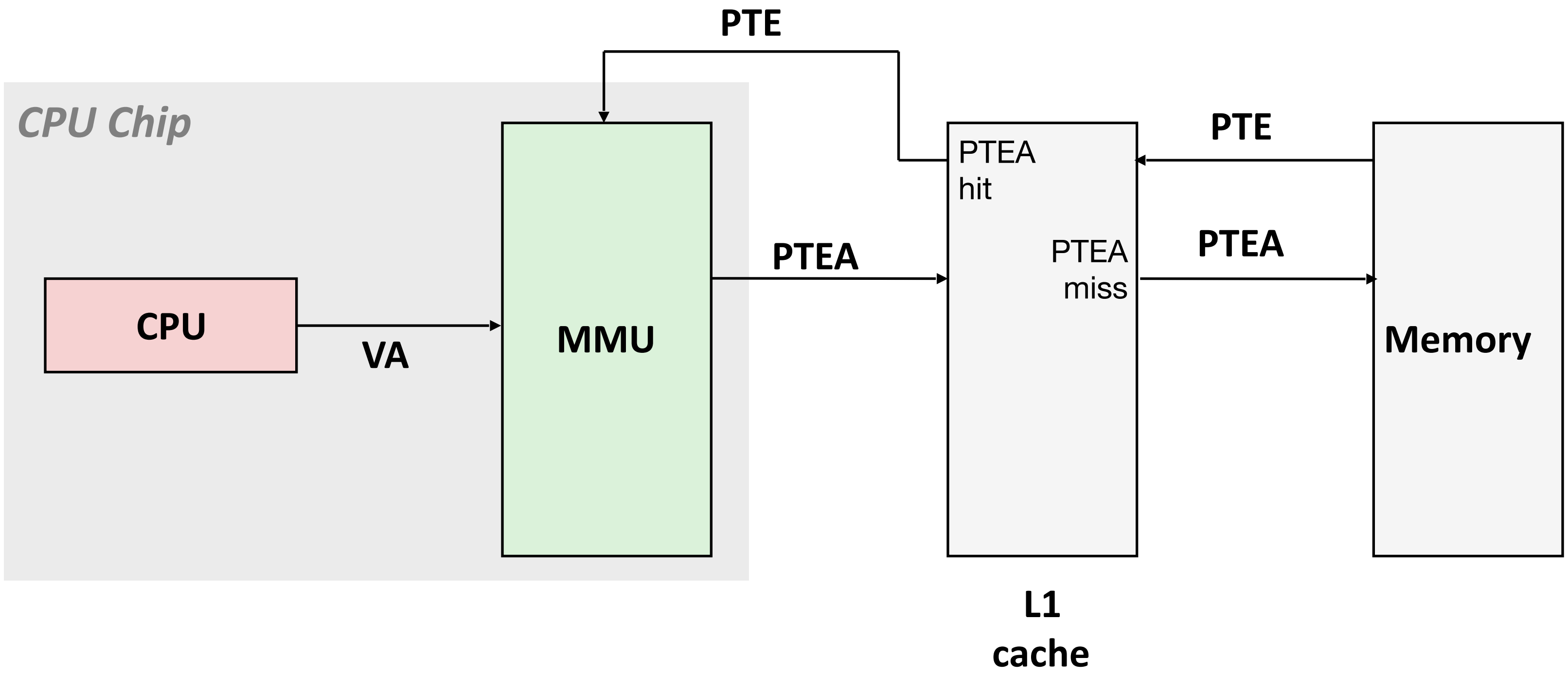
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



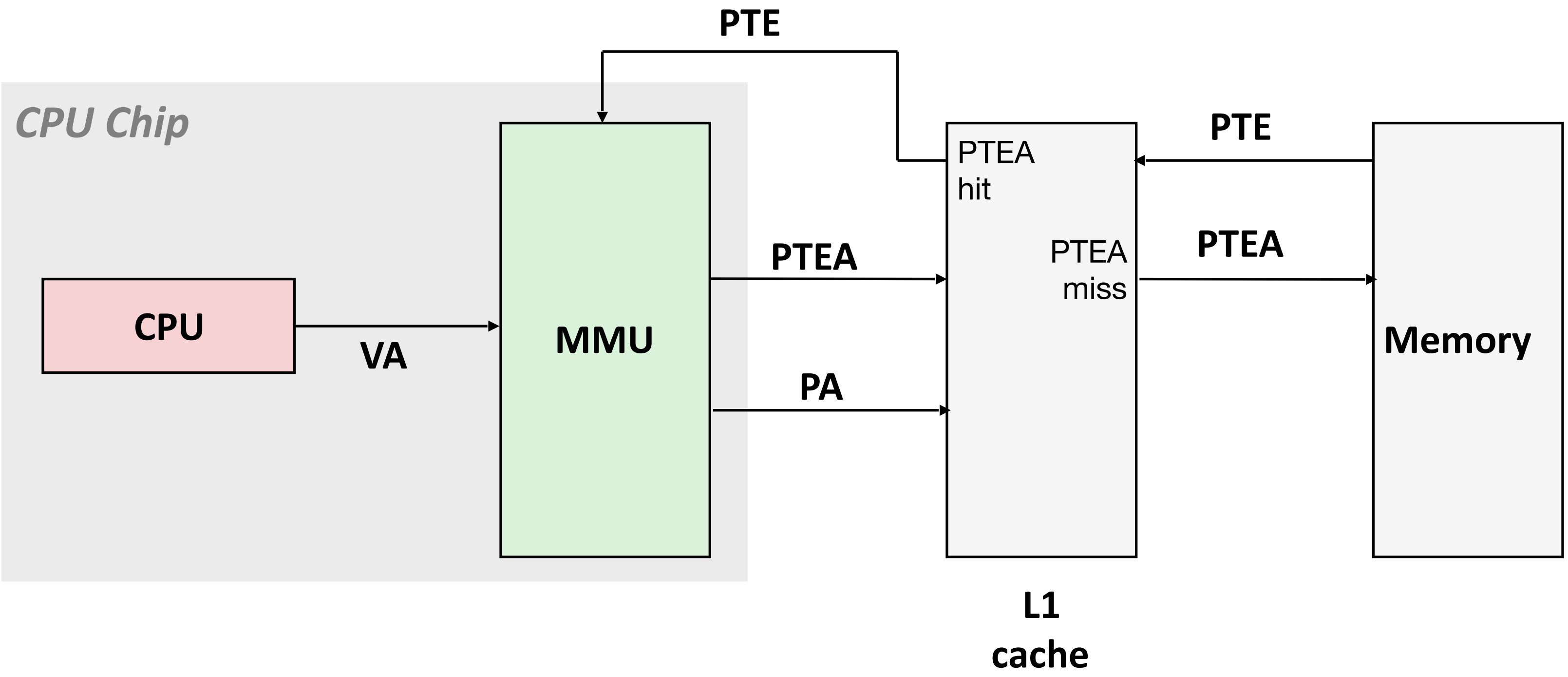
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



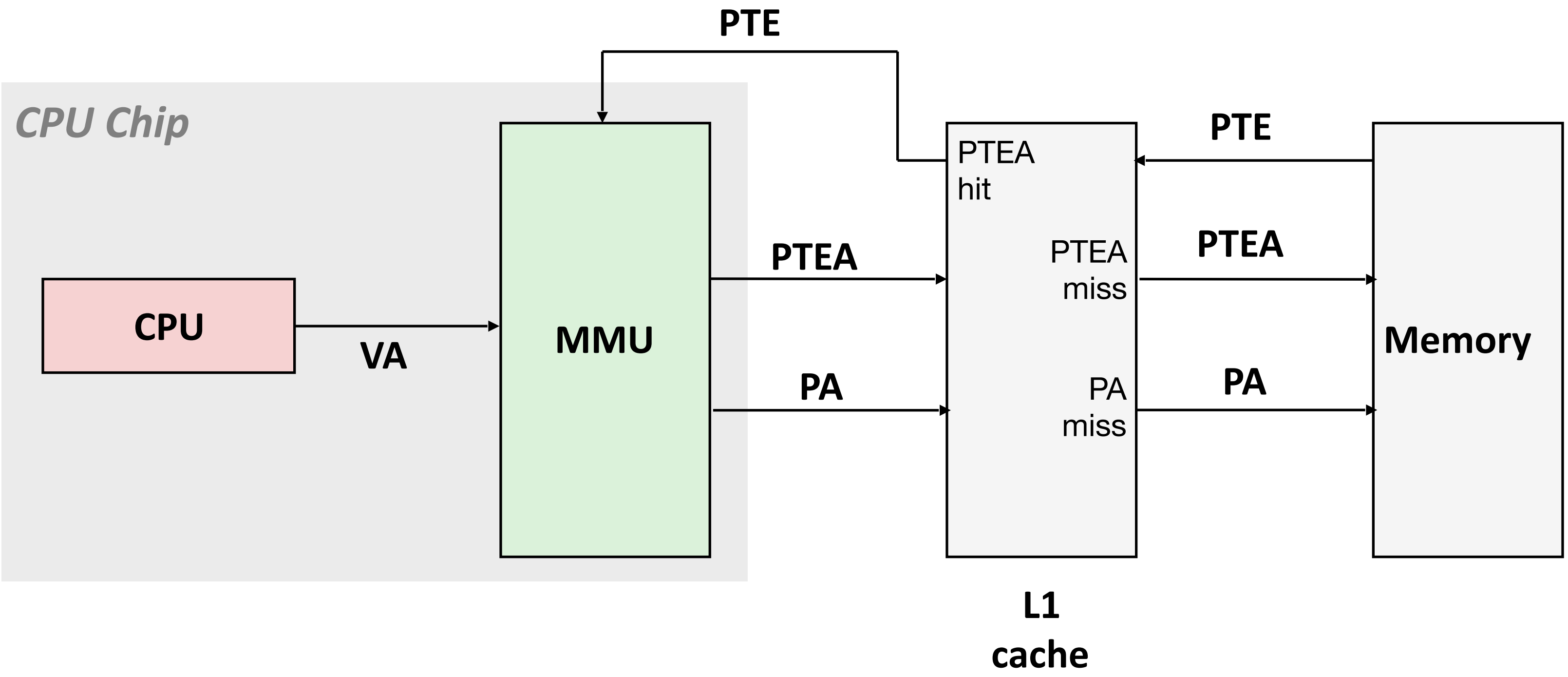
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



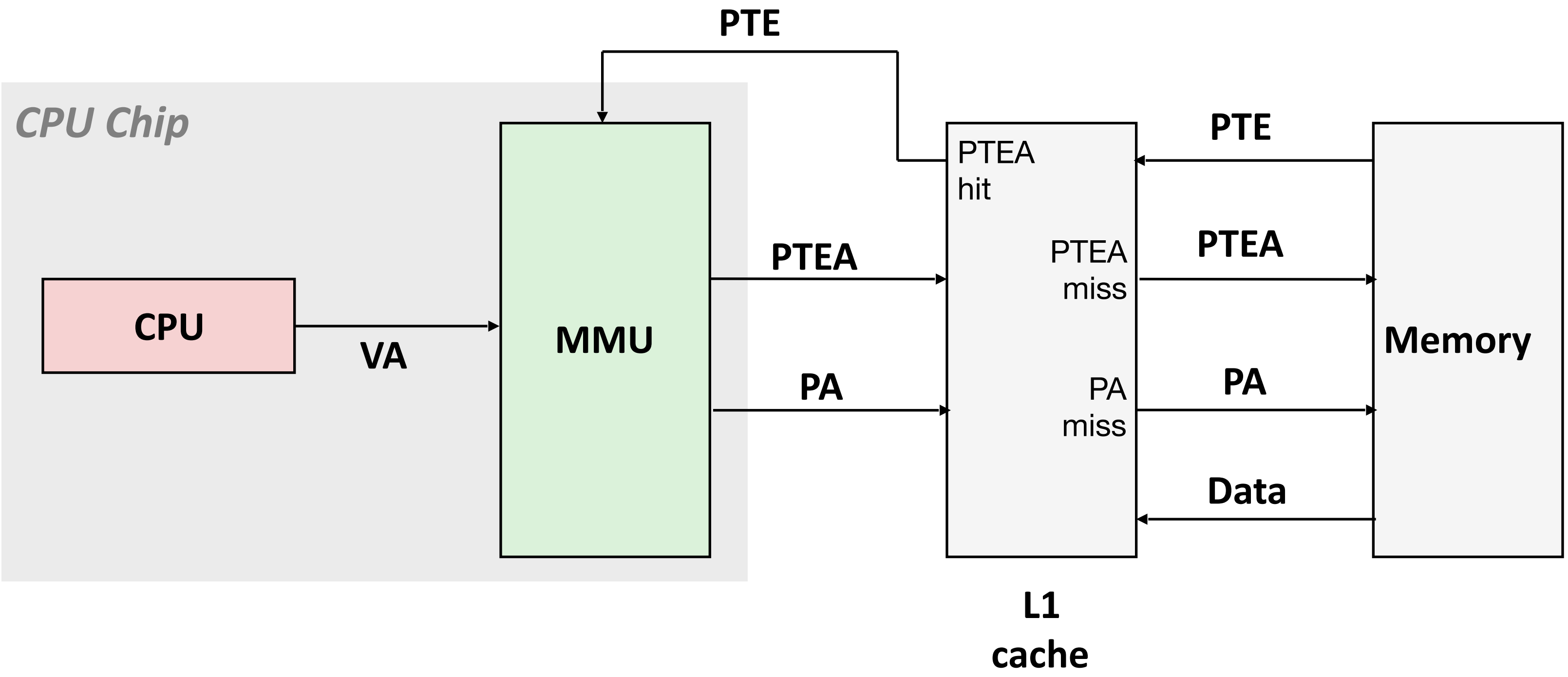
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



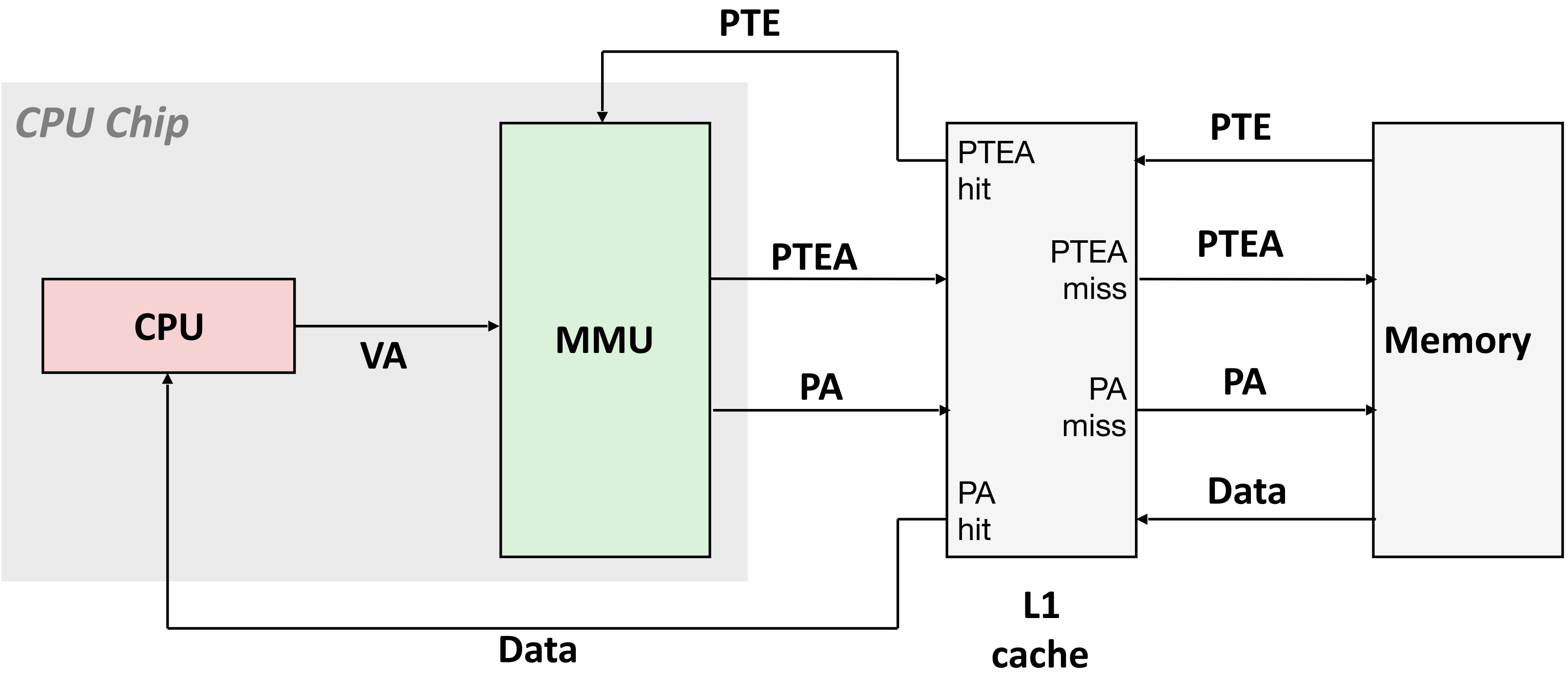
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Today

- Three Virtual Memory Optimizations
  - TLB
  - Page the page table (a.k.a., multi-level page table)
  - Virtually-indexed, physically-tagged cache
- Case-study: Intel Core i7/Linux example

Speeding up Address Translation

Speeding up Address Translation

- Problem: Every memory load/store requires two memory accesses: one for PTE, another for real
  - The PTE access is kind of an overhead
  - Can we speed it up?

Speeding up Address Translation

- Problem: Every memory load/store requires two memory accesses: one for PTE, another for real
  - The PTE access is kind of an overhead
  - Can we speed it up?
- Page table entries (PTEs) are already cached in L1 data cache like any other memory data. But:
  - PTEs may be evicted by other data references
  - PTE hit still requires a small L1 delay

Speeding up Translation with a TLB

- Solution: *Translation Lookaside Buffer* (TLB)
  - Think of it as a dedicated cache for page table
  - Small set-associative hardware cache in MMU
  - Contains complete page table entries for a small number of pages

Speeding up Translation with a TLB

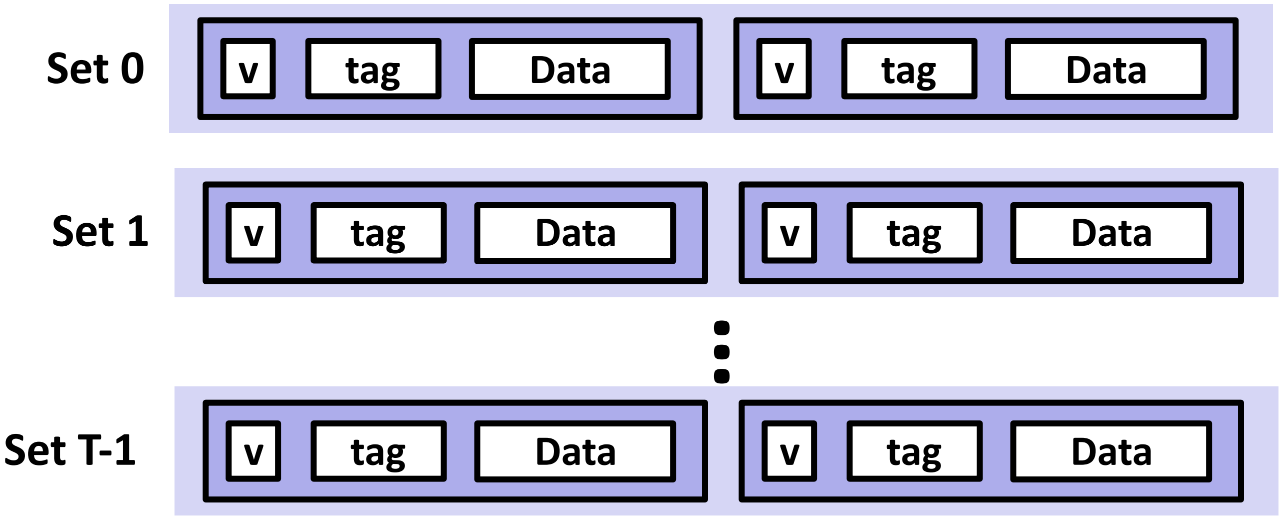
- Solution: *Translation Lookaside Buffer* (TLB)
  - Think of it as a dedicated cache for page table
  - Small set-associative hardware cache in MMU
  - Contains complete page table entries for a small number of pages

Tag	Set Index

Speeding up Translation with a TLB

- Solution: *Translation Lookaside Buffer* (TLB)
  - Think of it as a dedicated cache for page table
  - Small set-associative hardware cache in MMU
  - Contains complete page table entries for a small number of pages

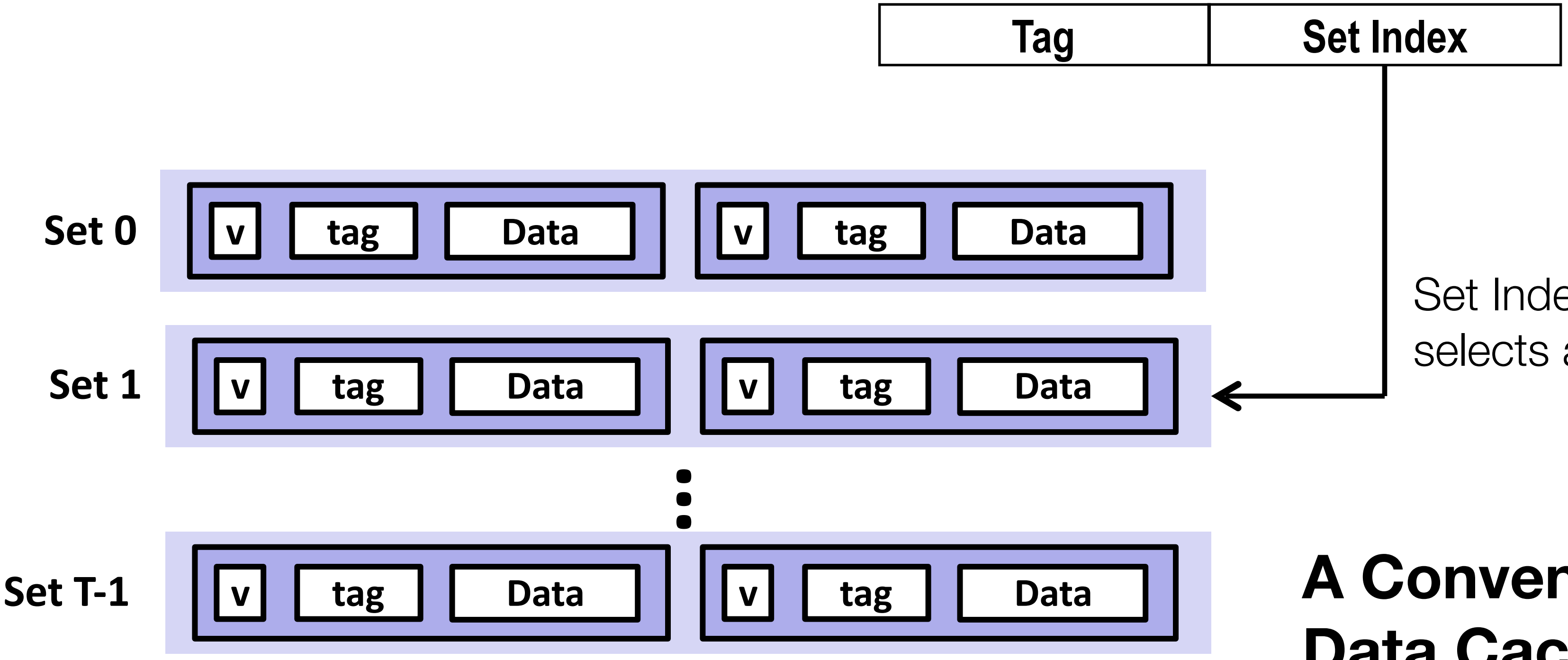
Tag	Set Index



A Conventional Data Cache

Speeding up Translation with a TLB

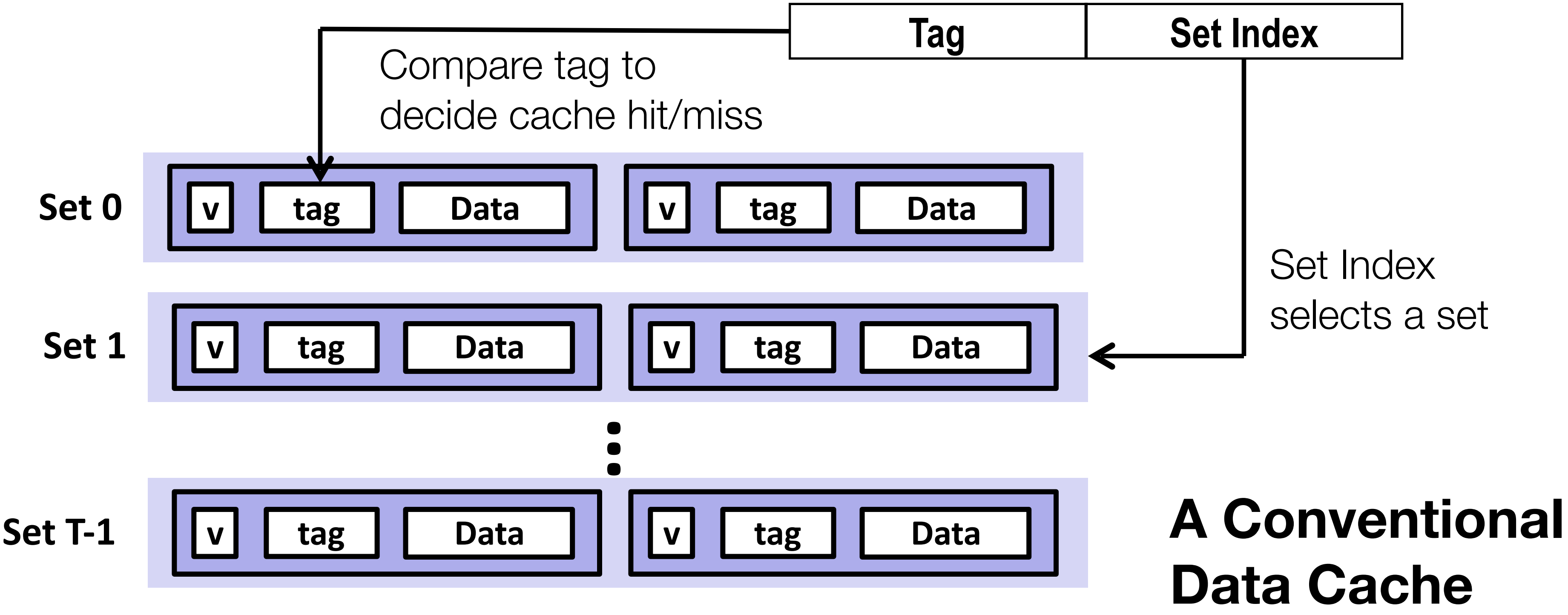
- Solution: *Translation Lookaside Buffer* (TLB)
  - Think of it as a dedicated cache for page table
  - Small set-associative hardware cache in MMU
  - Contains complete page table entries for a small number of pages



A Conventional Data Cache

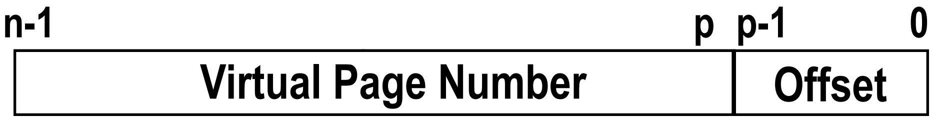
Speeding up Translation with a TLB

- Solution: *Translation Lookaside Buffer* (TLB)
  - Think of it as a dedicated cache for page table
  - Small set-associative hardware cache in MMU
  - Contains complete page table entries for a small number of pages



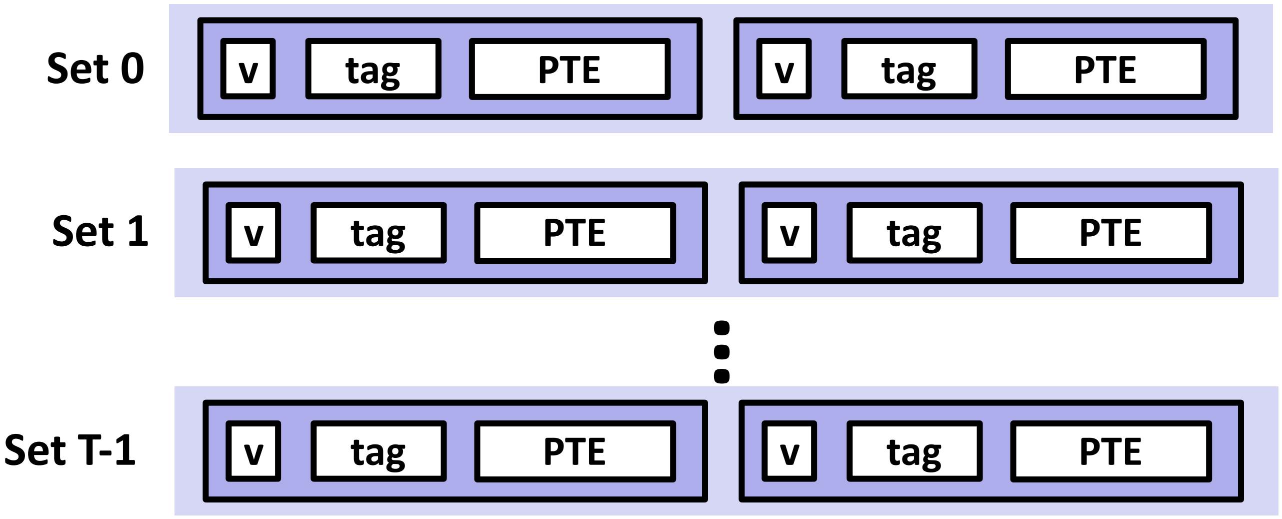
Accessing the TLB

- MMU uses the Virtual Page Number portion of the virtual address to access the TLB:



n-1 p p-1 0

Virtual Page Number | Offset



Set 0: v tag PTE | v tag PTE

Set 1: v tag PTE | v tag PTE

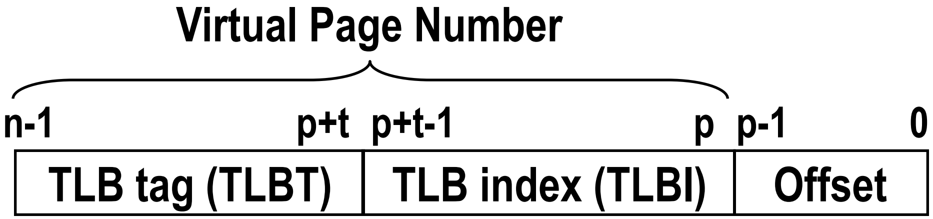
⋮

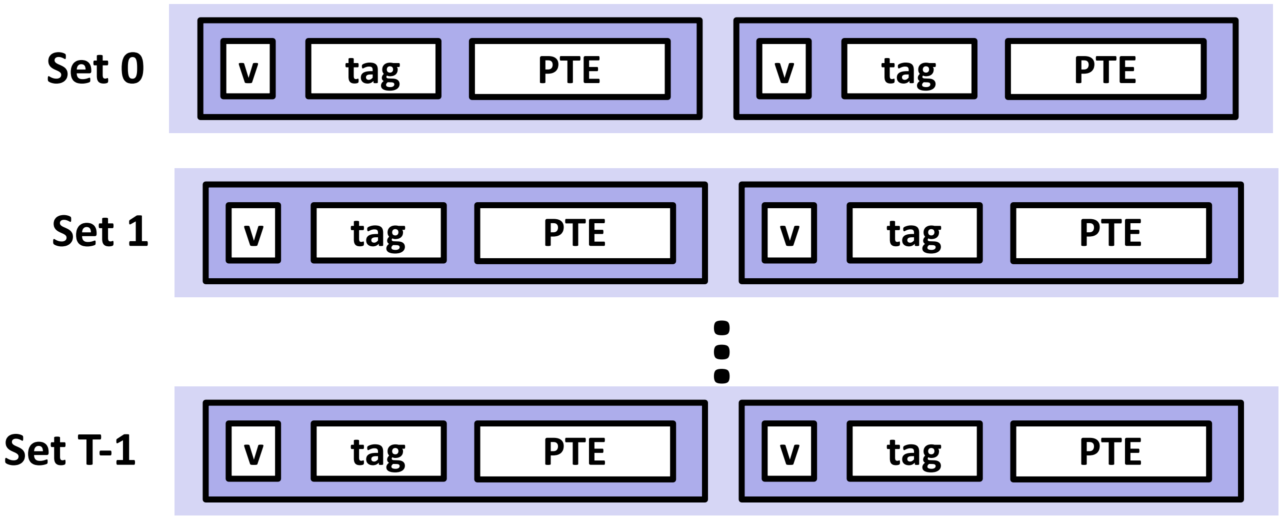
Set T-1: v tag PTE | v tag PTE

A Page Table Cache

Accessing the TLB

- MMU uses the Virtual Page Number portion of the virtual address to access the TLB:

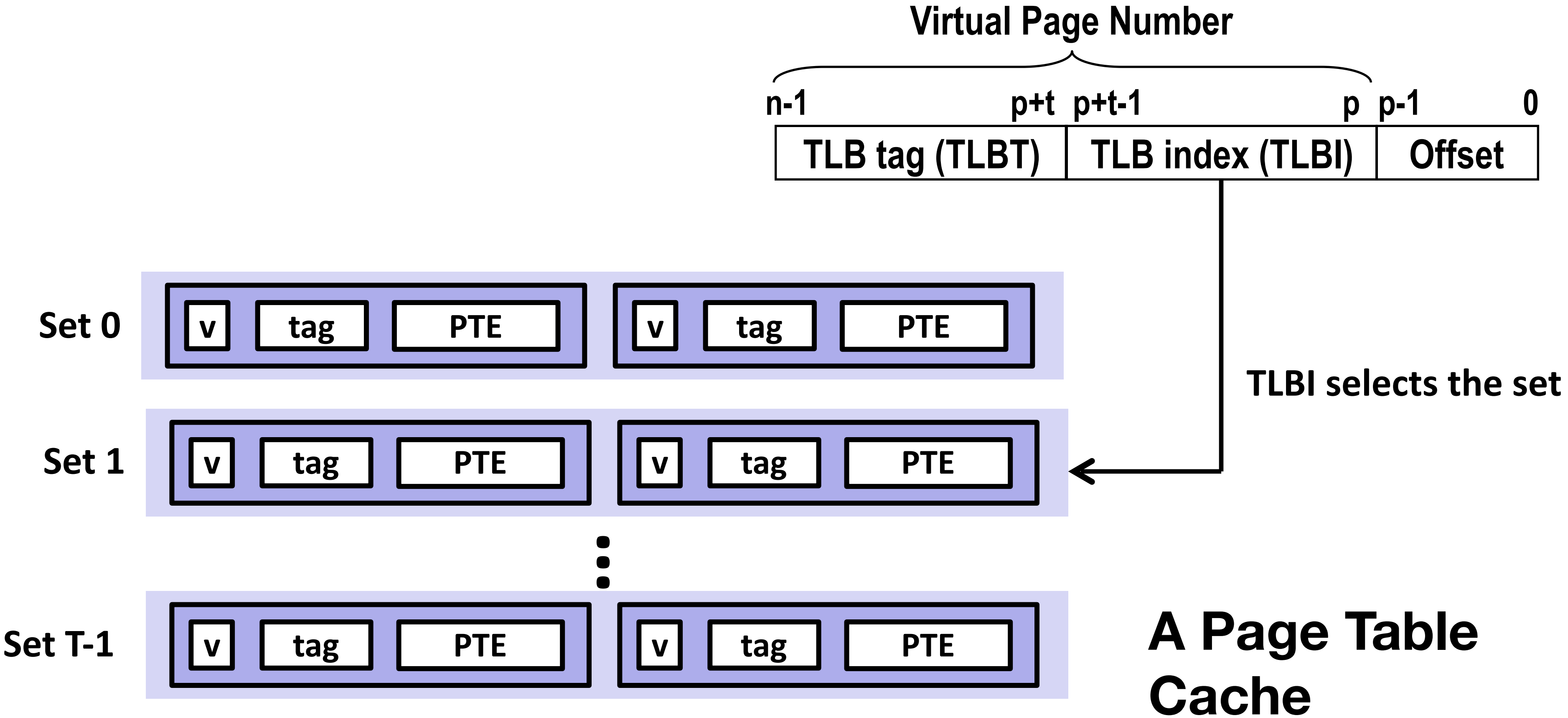




A Page Table Cache

Accessing the TLB

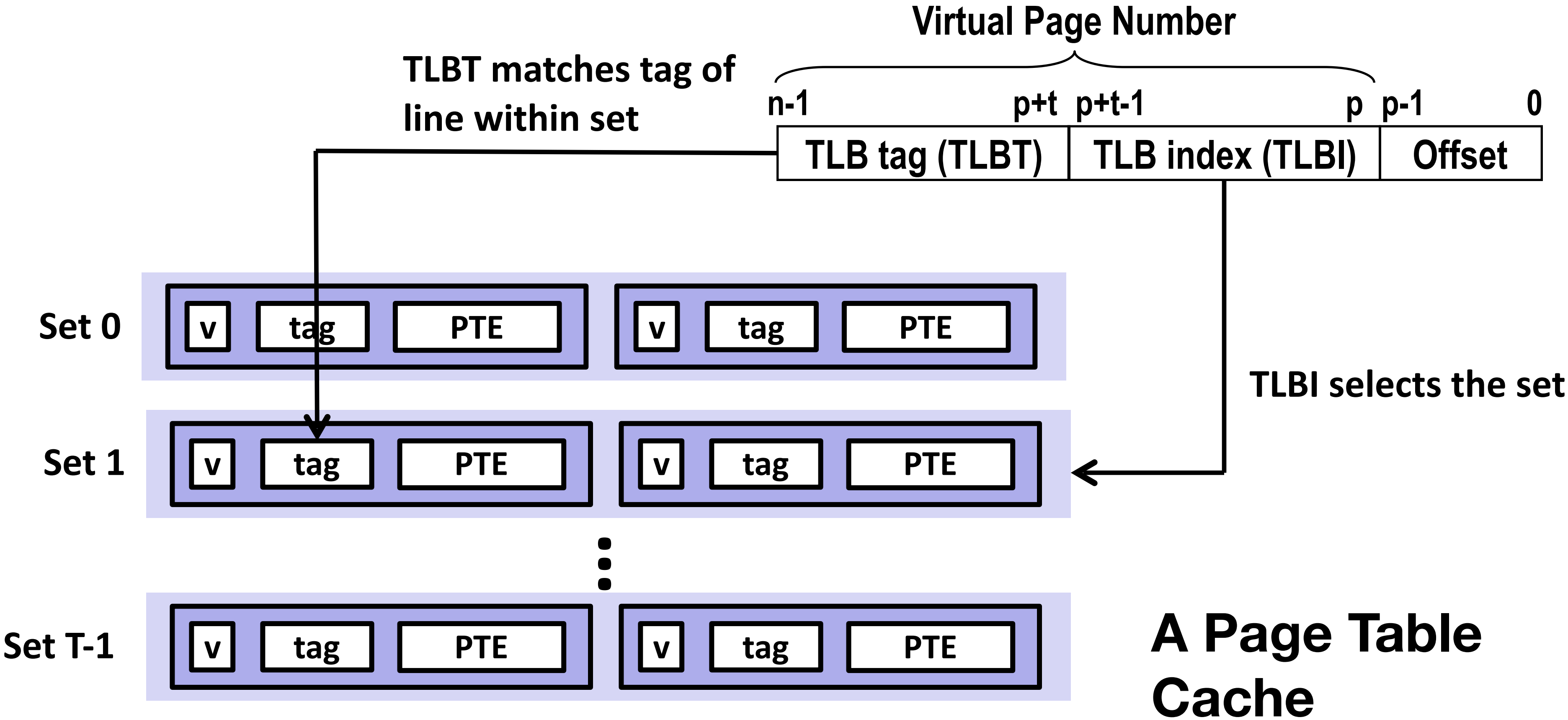
- MMU uses the Virtual Page Number portion of the virtual address to access the TLB:



A Page Table Cache

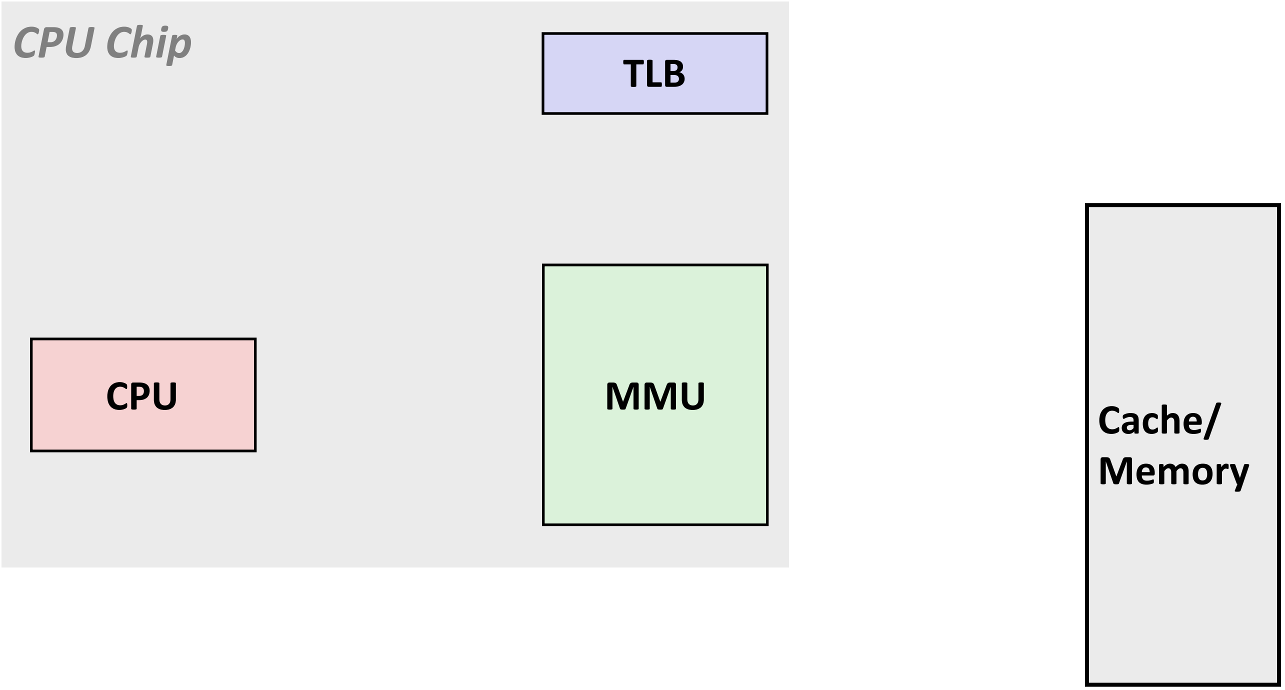
Accessing the TLB

- MMU uses the Virtual Page Number portion of the virtual address to access the TLB:

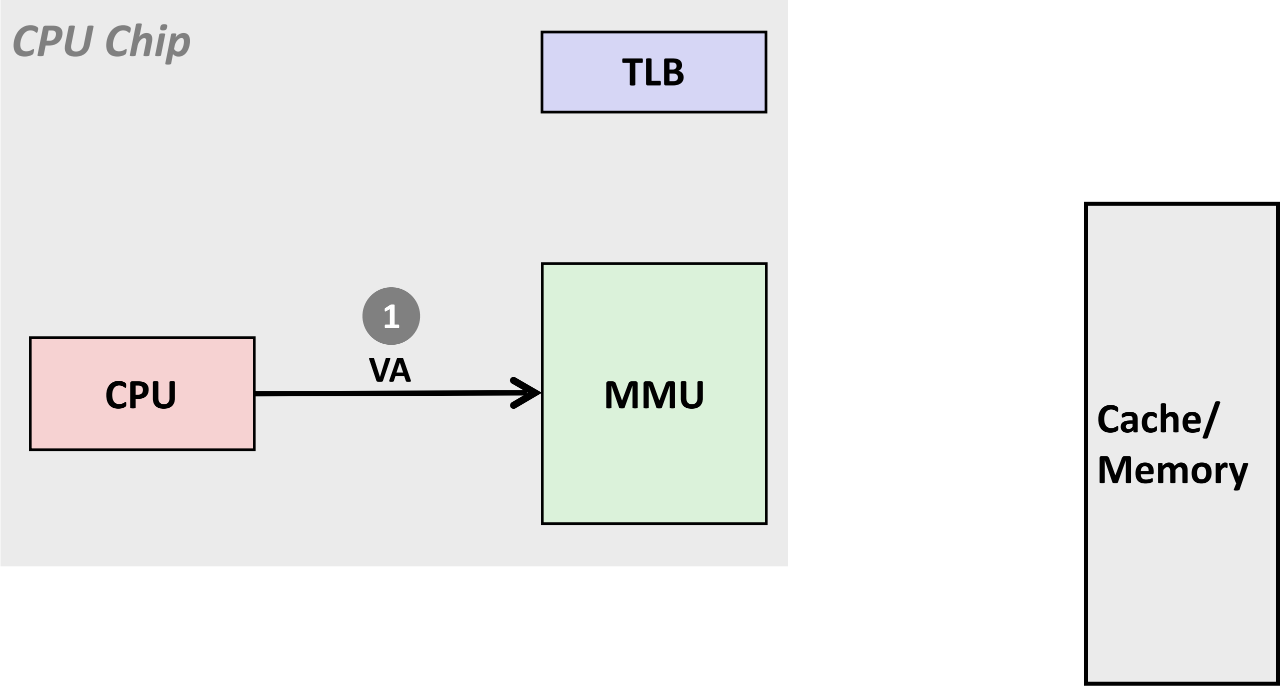


A Page Table Cache

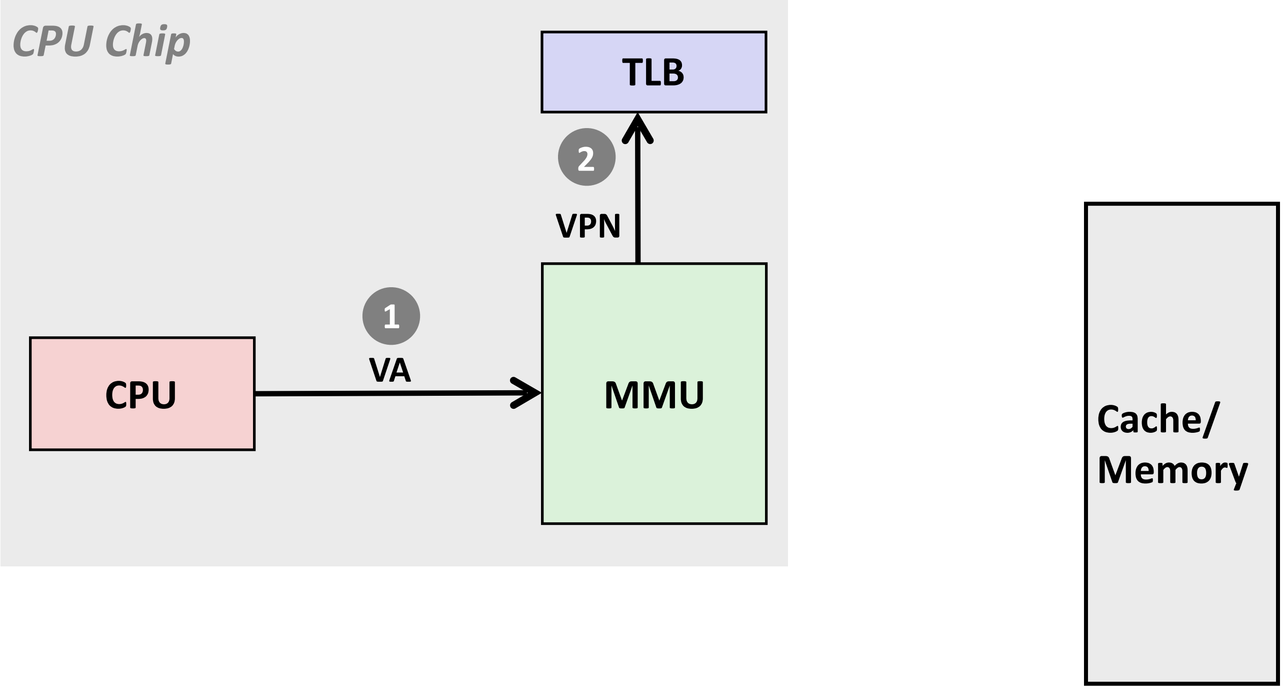
TLB Hit



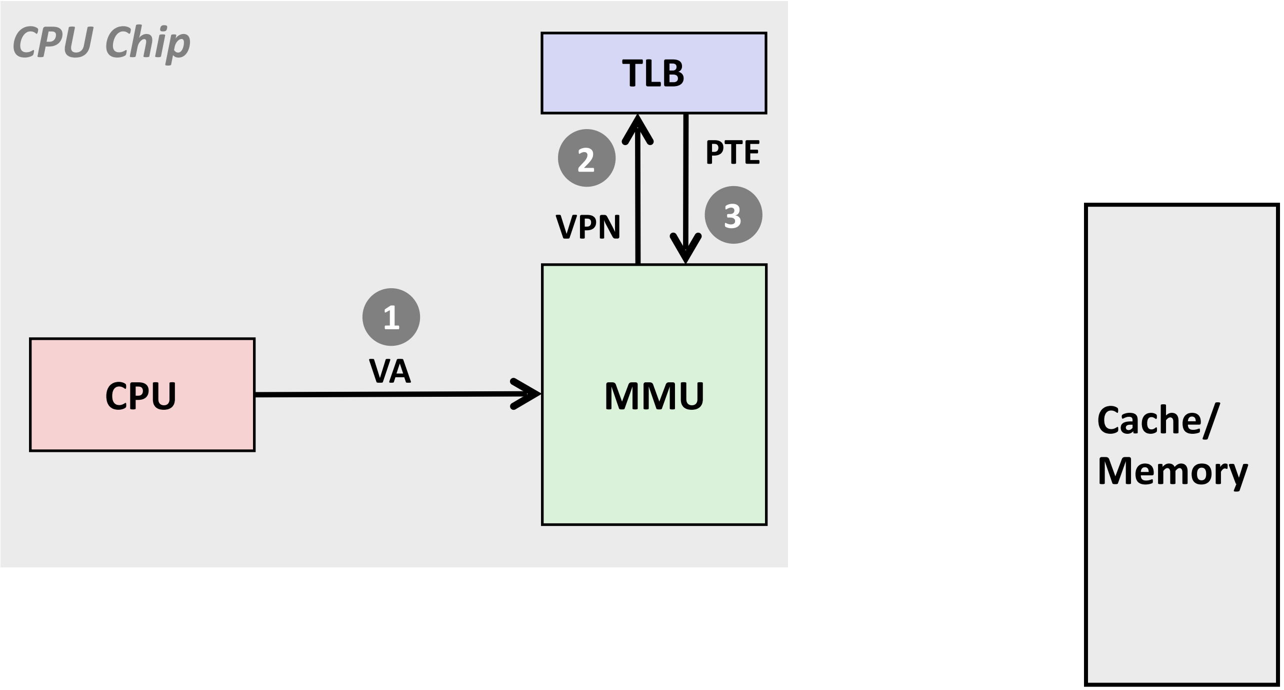
TLB Hit



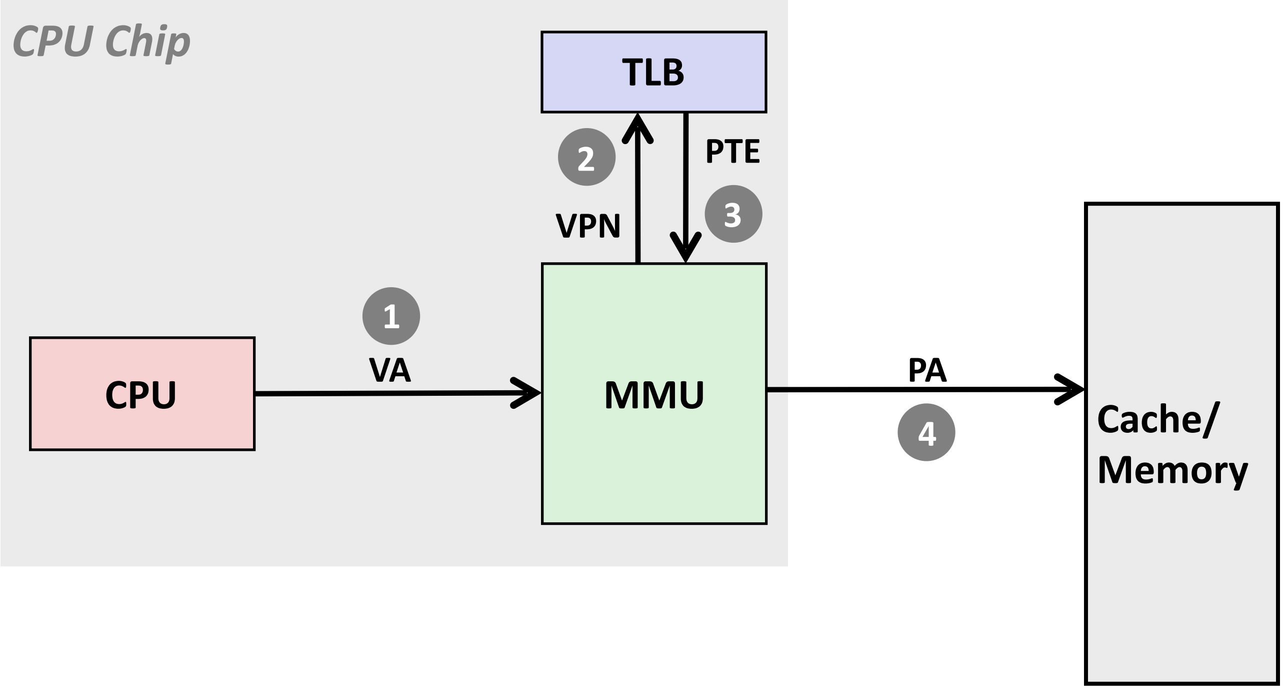
TLB Hit



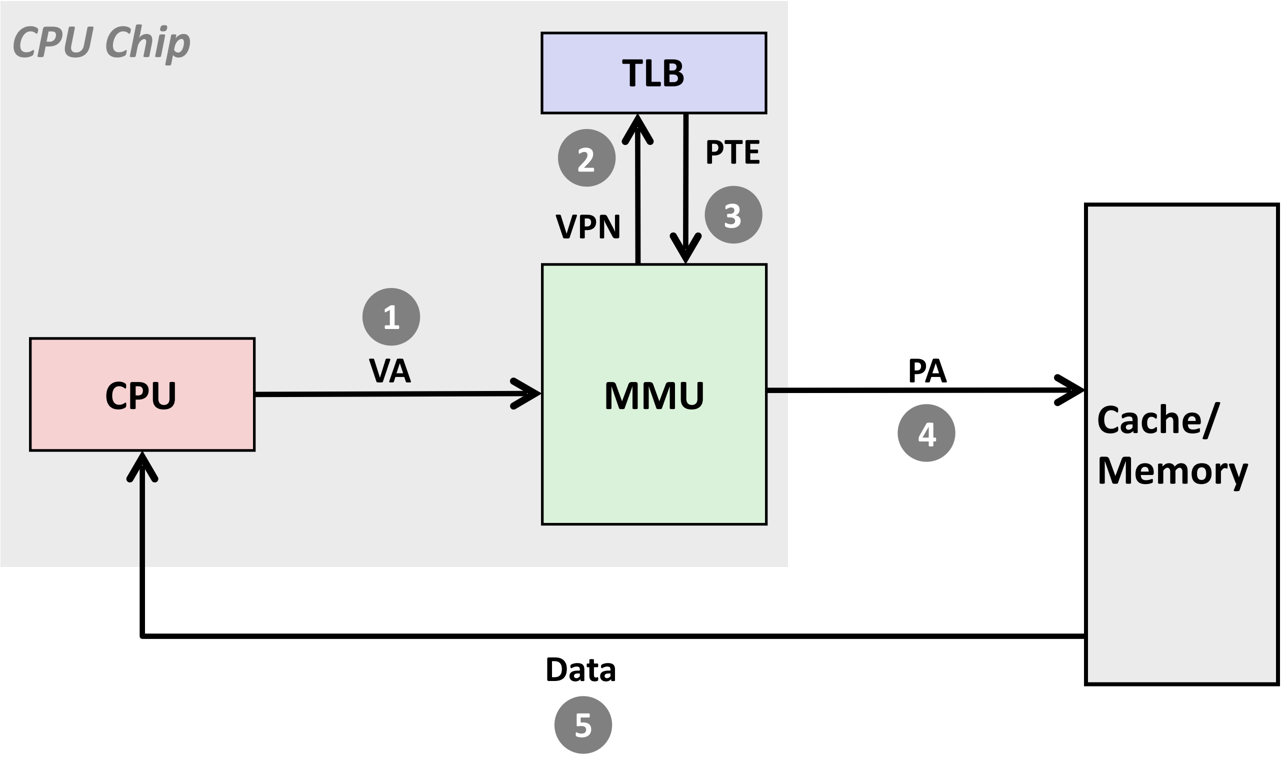
TLB Hit



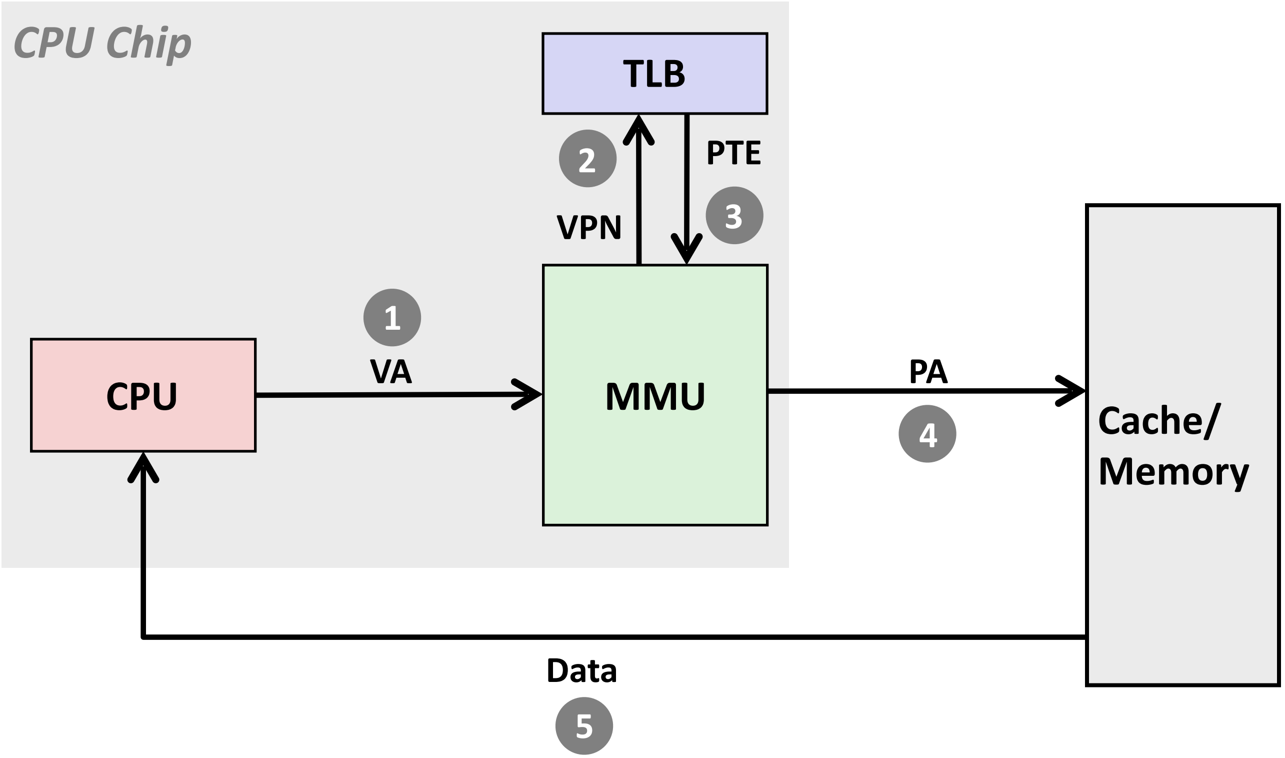
TLB Hit



TLB Hit

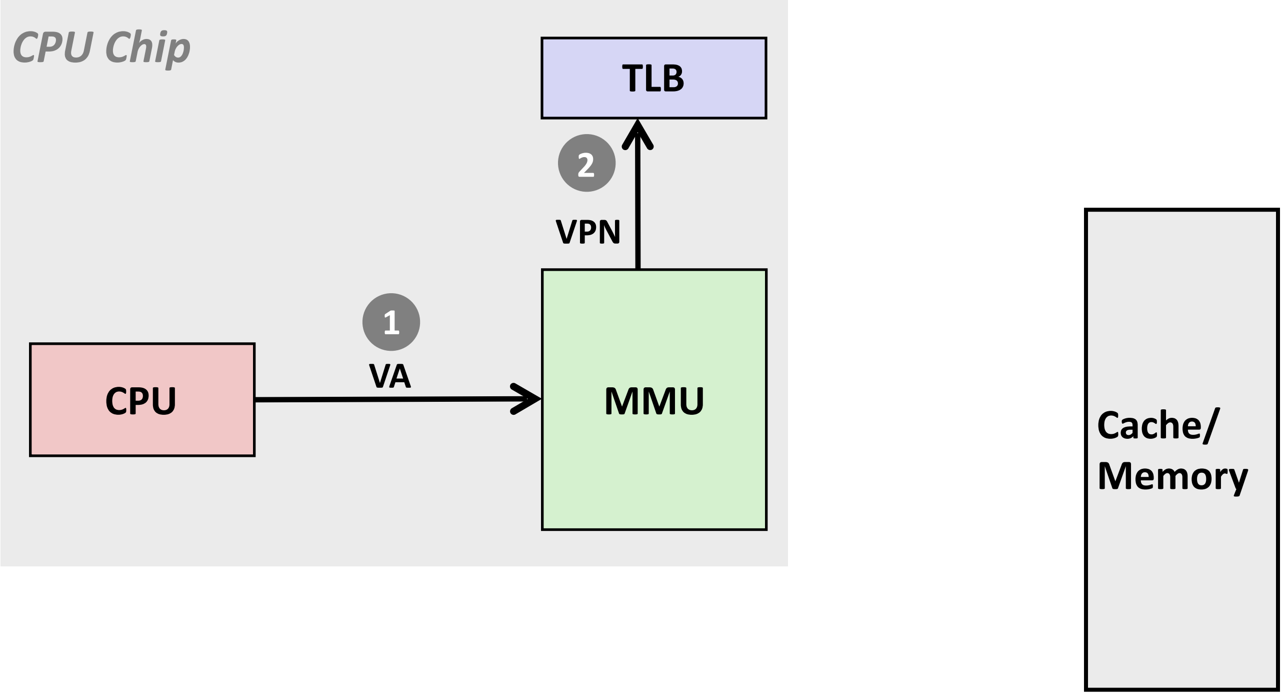


TLB Hit

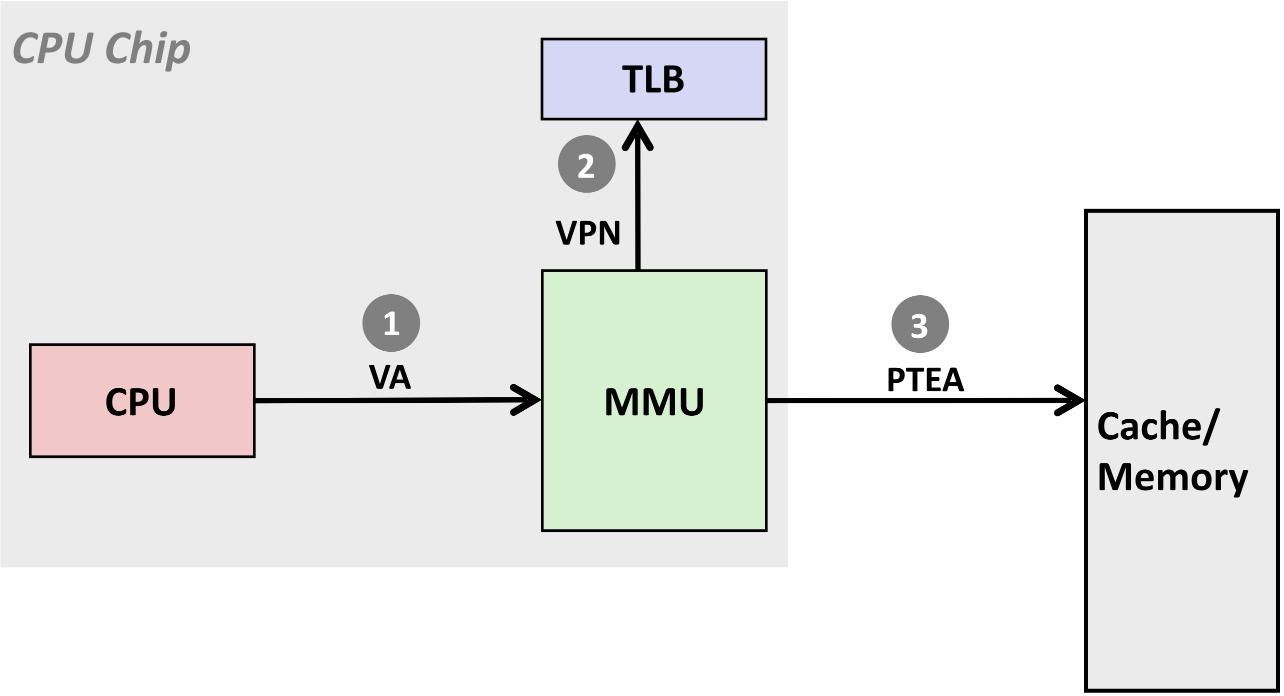


A TLB hit eliminates a memory access

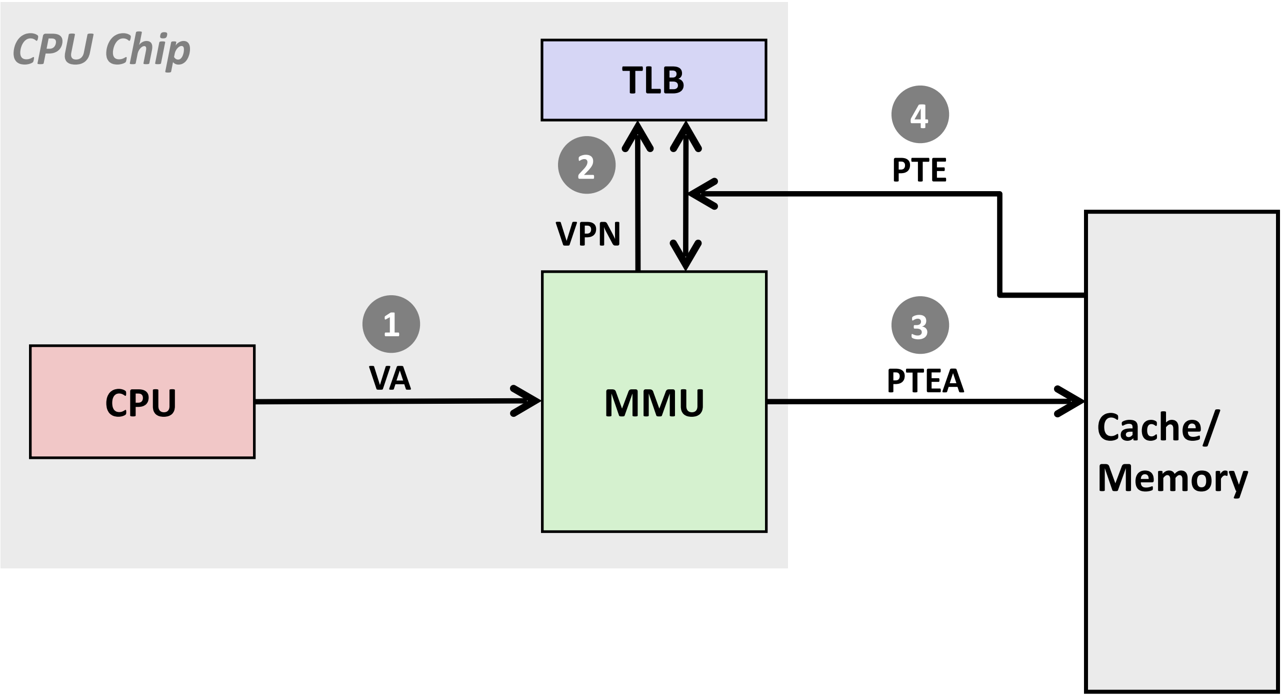
TLB Miss



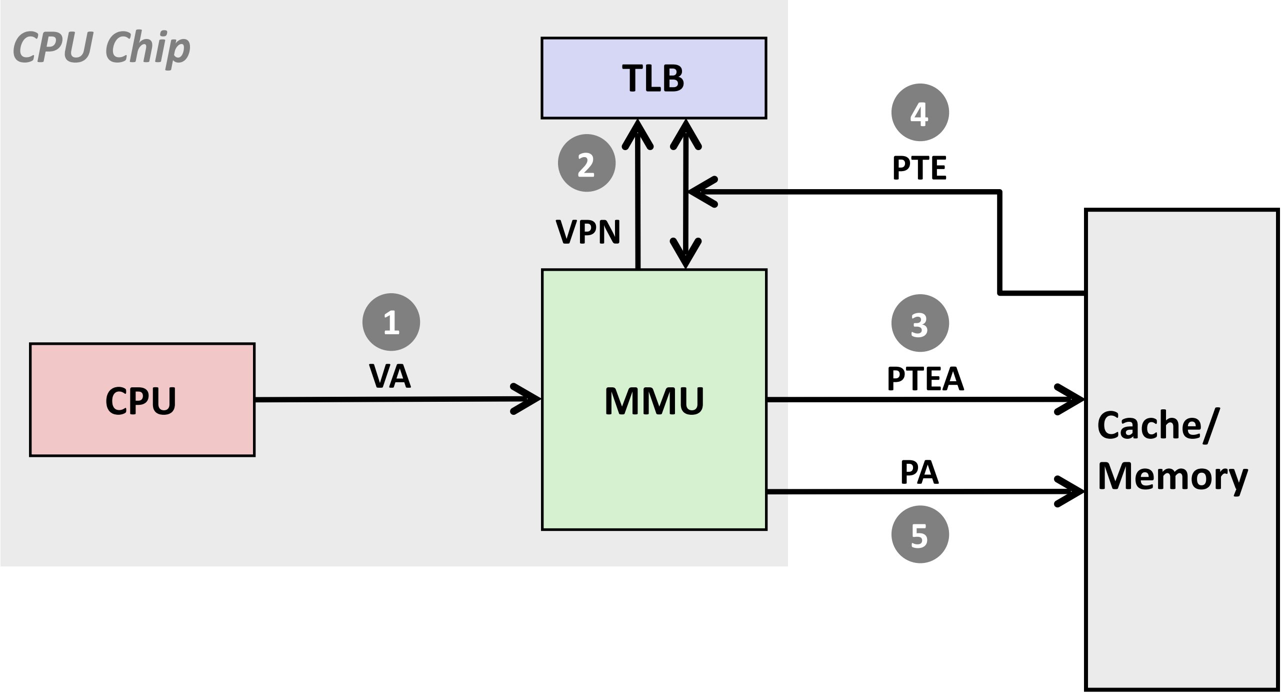
TLB Miss



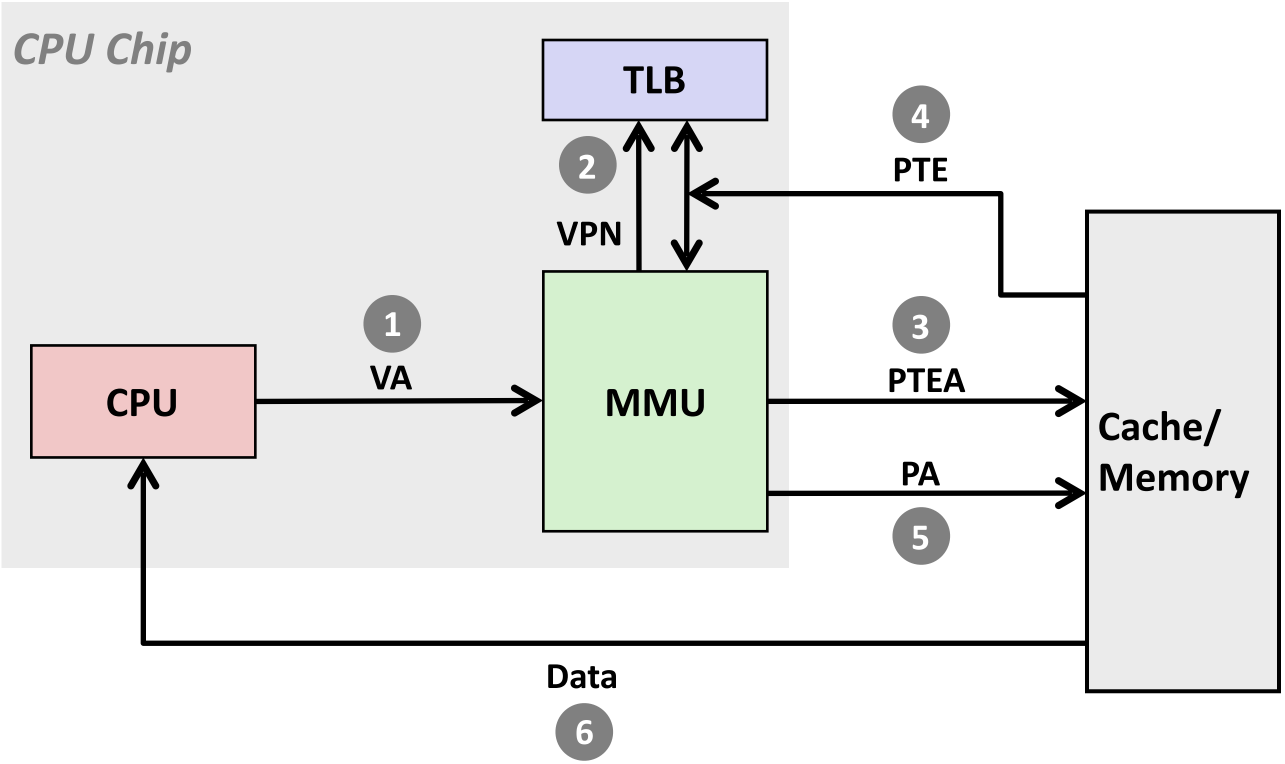
TLB Miss



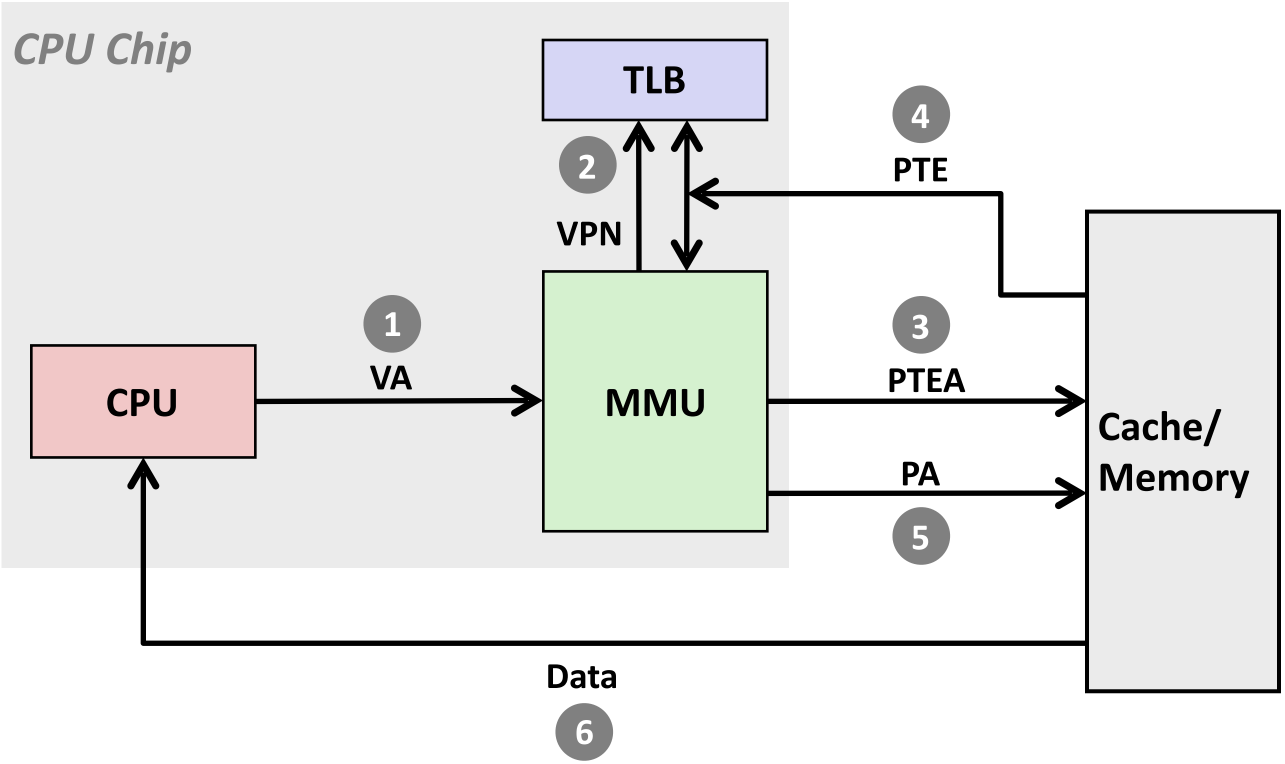
TLB Miss



TLB Miss


CPU Chip
TLB
4
PTE
2
VPN
1
VA
CPU
MMU
3
PTEA
PA
5
Cache/
Memory
Data
6

TLB Miss



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

Fortunately, TLB misses are rare.

Today

- Three Virtual Memory Optimizations
  - TLB
  - Page the page table (a.k.a., multi-level page table)
  - Virtually-indexed, physically-tagged cache
- Case-study: Intel Core i7/Linux example

Where Does Page Table Live?

Where Does Page Table Live?

- It needs to be at a specific location where we can find it
  - In main memory, with its start address stored in a special register (PTBR)

Where Does Page Table Live?

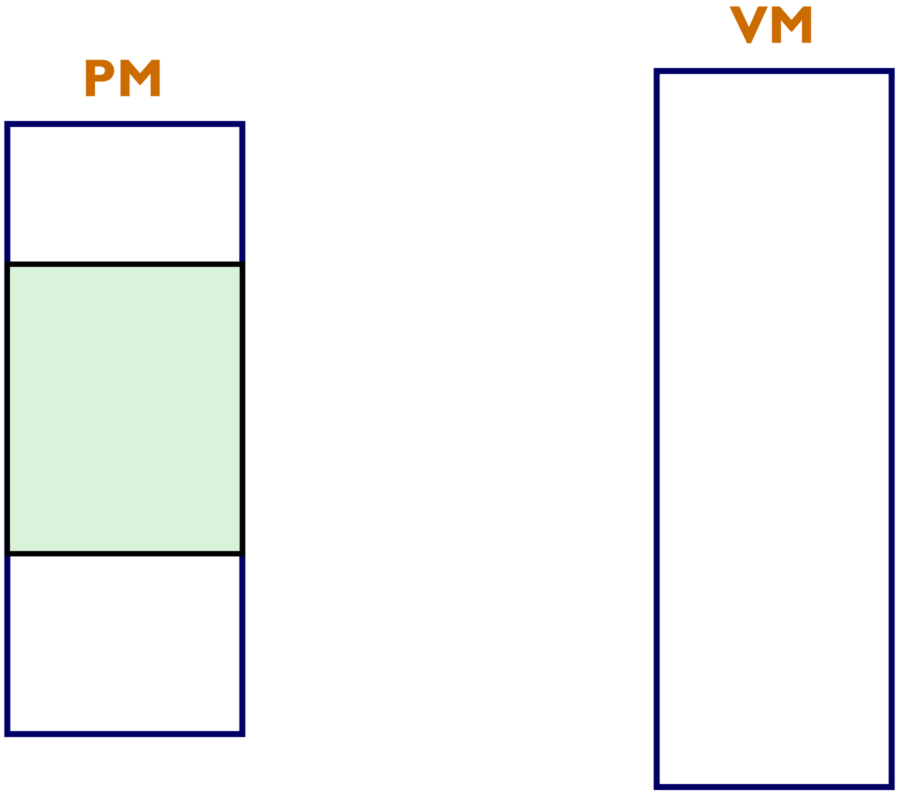
- It needs to be at a specific location where we can find it
    - In main memory, with its start address stored in a special register (PTBR)
- Assume 4KB page, 48-bit virtual memory, each PTE is 8 Bytes
    - 2^{36} PTEs in a page table
    - 512 GB total size per page table??!!

Where Does Page Table Live?

- It needs to be at a specific location where we can find it
    - In main memory, with its start address stored in a special register (PTBR)
- Assume 4KB page, 48-bit virtual memory, each PTE is 8 Bytes
    - 2^{36} PTEs in a page table
    - 512 GB total size per page table??!!
- Problem: Page tables are huge
    - One table per process!
    - Storing them all in main memory wastes space

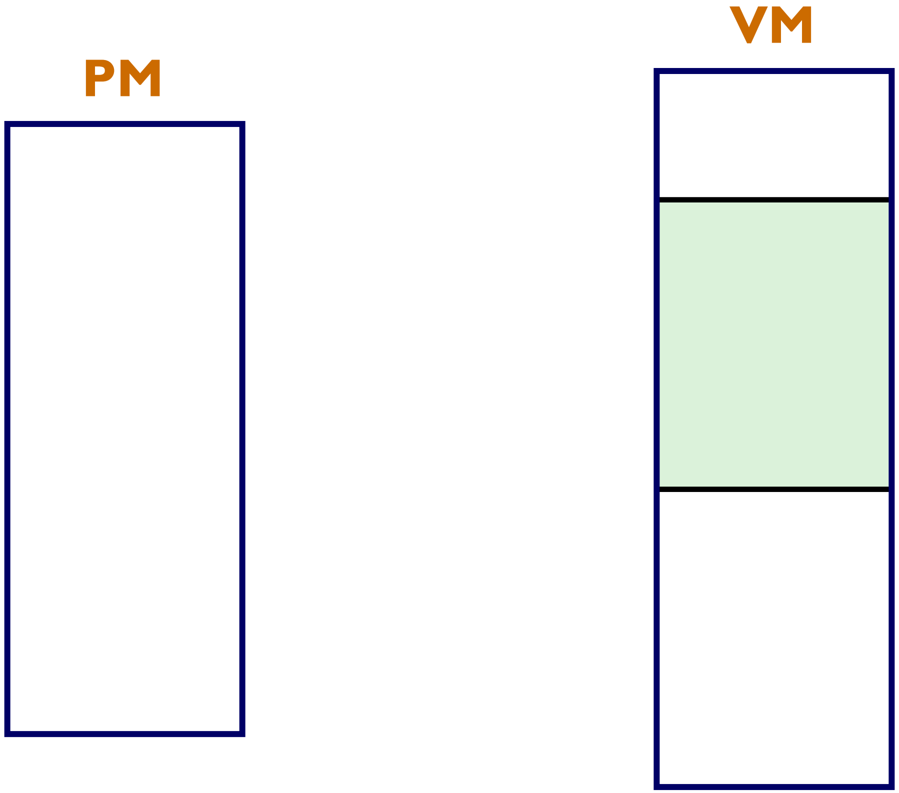
Solution: Page the Page Table

- Observation: Only a small number of pages (working set) are accessed during a certain period of time, due to locality
- Put only the relevant page table entires in main memory
- Idea: Put Page Table in Virtual Memory and swap it just like data



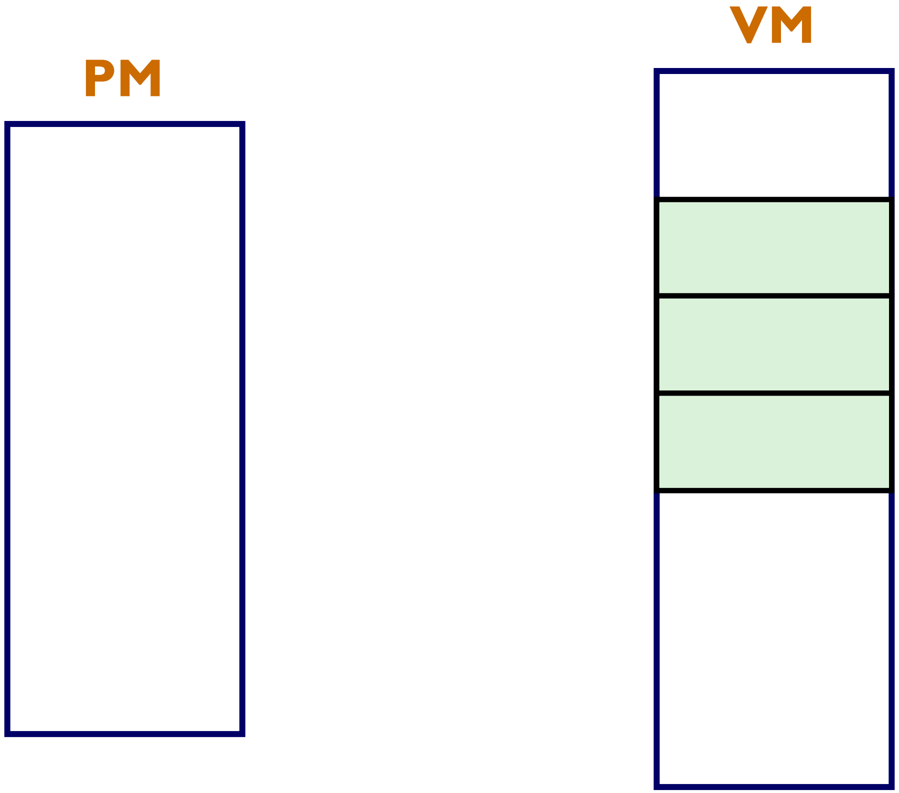
Solution: Page the Page Table

- Observation: Only a small number of pages (working set) are accessed during a certain period of time, due to locality
- Put only the relevant page table entires in main memory
- Idea: Put Page Table in Virtual Memory and swap it just like data



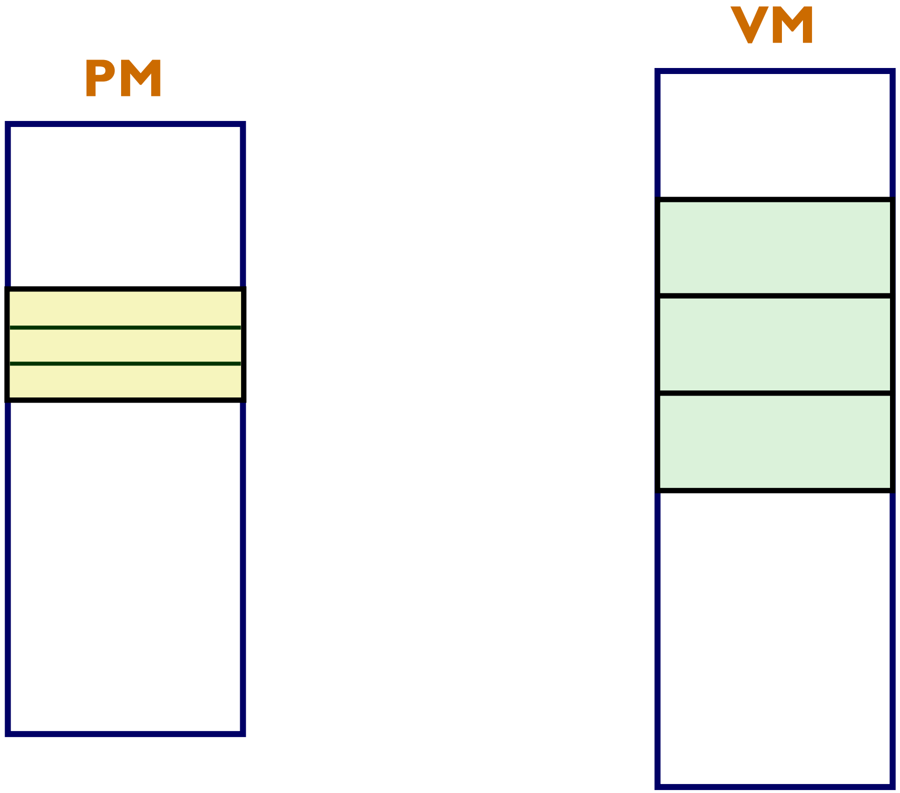
Solution: Page the Page Table

- Observation: Only a small number of pages (working set) are accessed during a certain period of time, due to locality
- Put only the relevant page table entires in main memory
- Idea: Put Page Table in Virtual Memory and swap it just like data



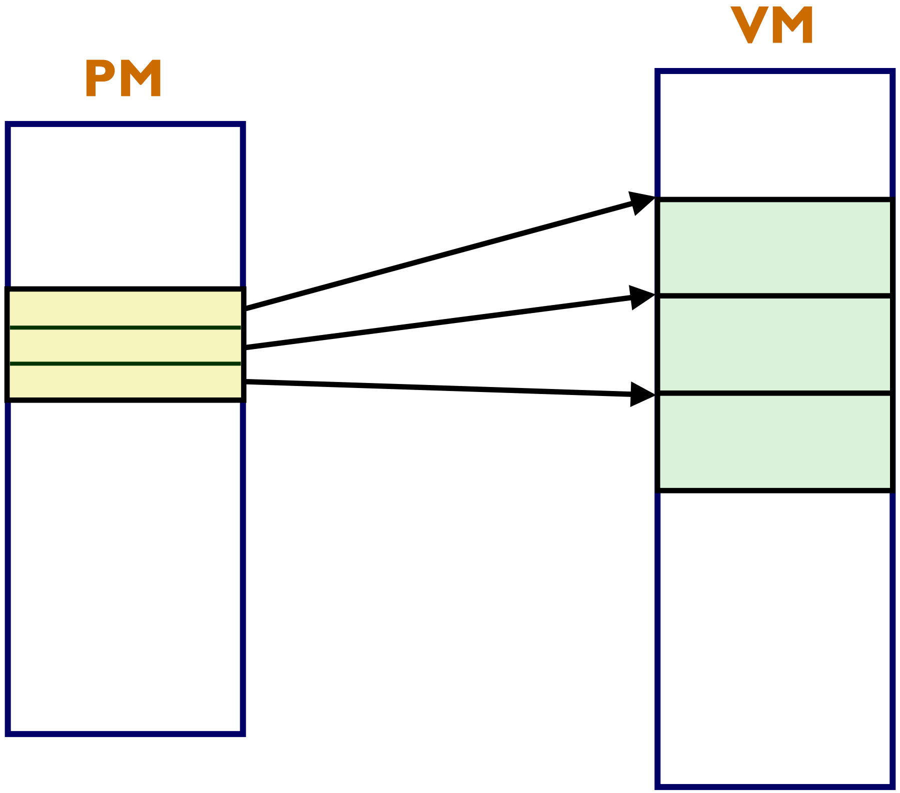
Solution: Page the Page Table

- Observation: Only a small number of pages (working set) are accessed during a certain period of time, due to locality
- Put only the relevant page table entires in main memory
- Idea: Put Page Table in Virtual Memory and swap it just like data



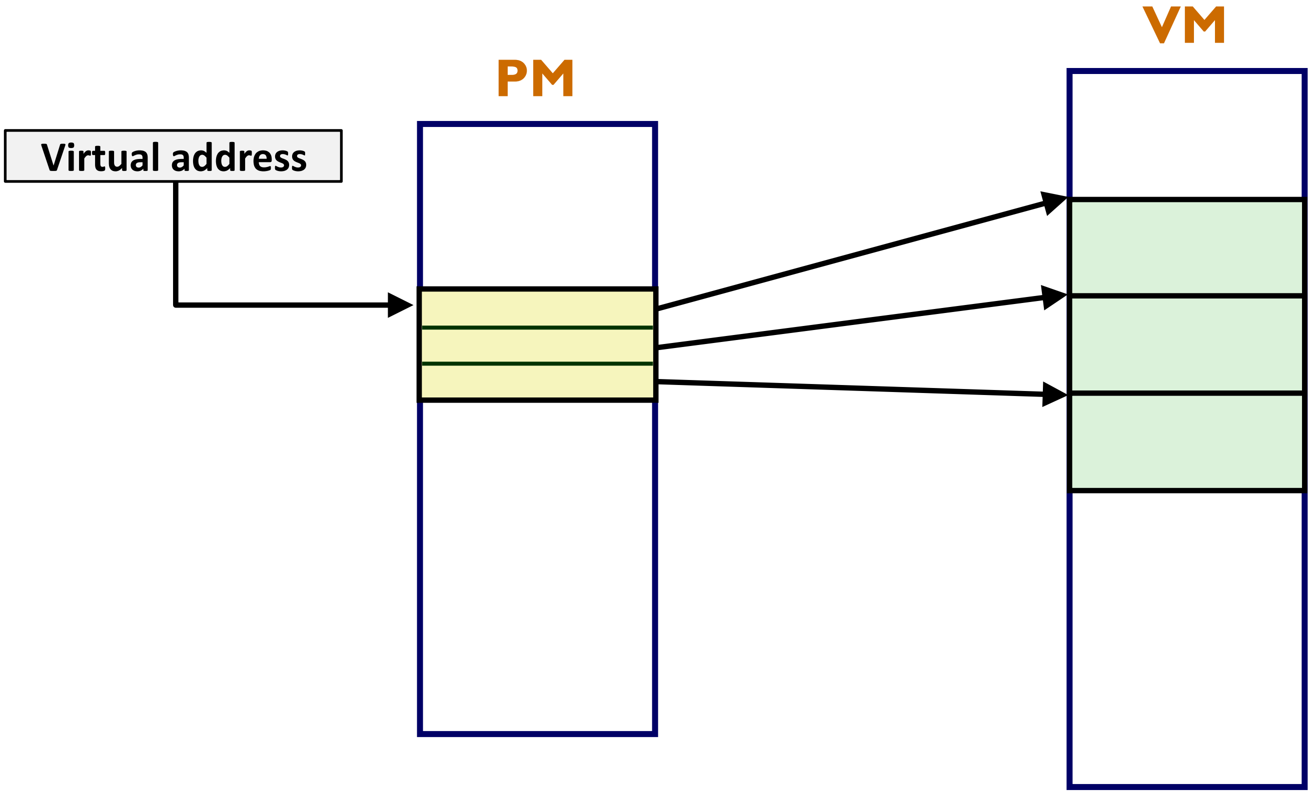
Solution: Page the Page Table

- Observation: Only a small number of pages (working set) are accessed during a certain period of time, due to locality
- Put only the relevant page table entires in main memory
- Idea: Put Page Table in Virtual Memory and swap it just like data



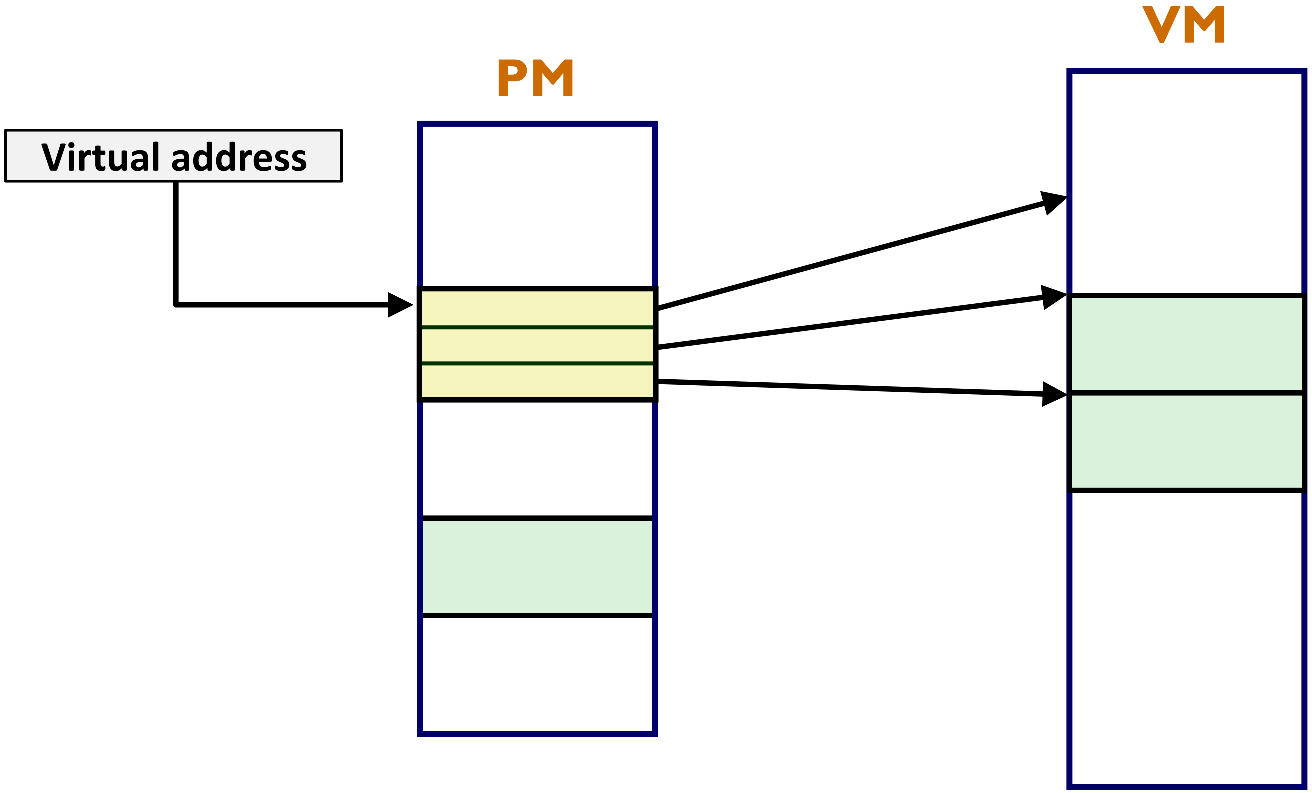
Solution: Page the Page Table

- Observation: Only a small number of pages (working set) are accessed during a certain period of time, due to locality
- Put only the relevant page table entires in main memory
- Idea: Put Page Table in Virtual Memory and swap it just like data



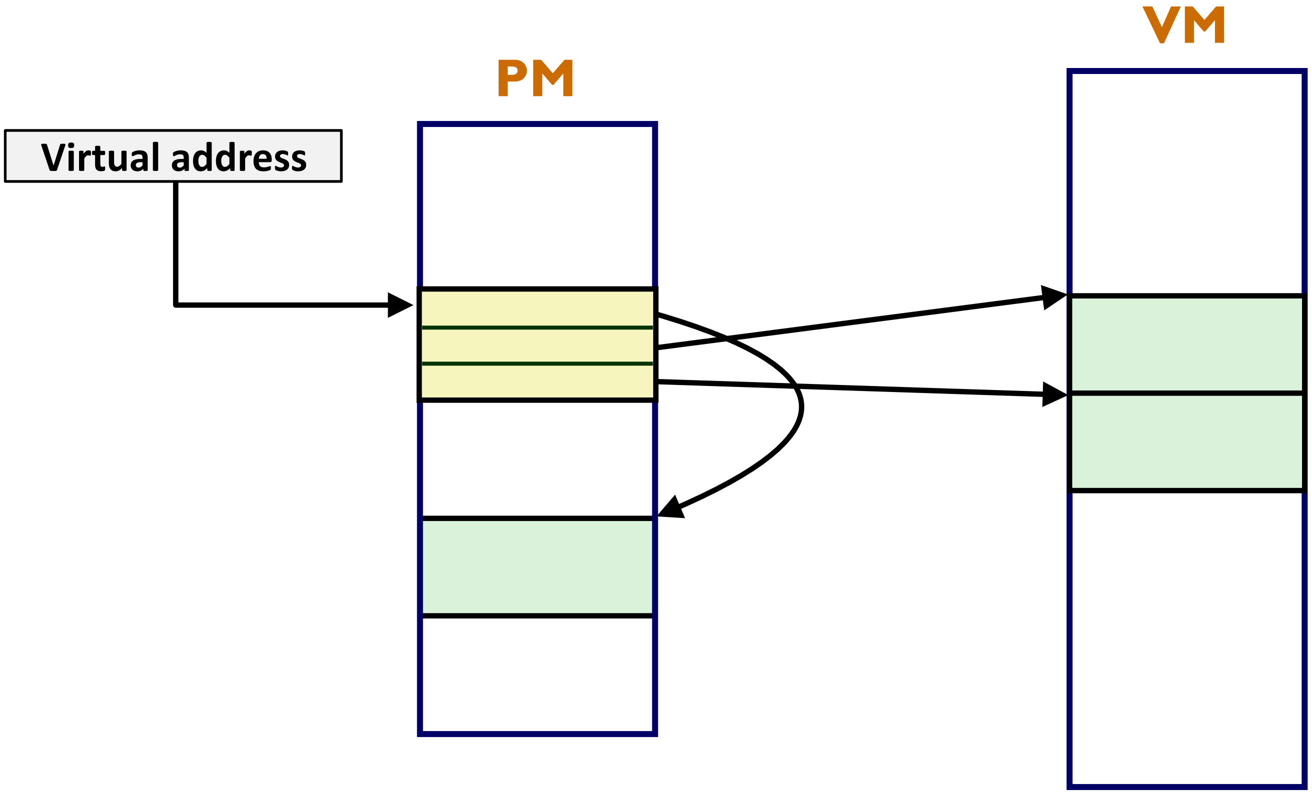
Solution: Page the Page Table

- Observation: Only a small number of pages (working set) are accessed during a certain period of time, due to locality
- Put only the relevant page table entires in main memory
- Idea: Put Page Table in Virtual Memory and swap it just like data



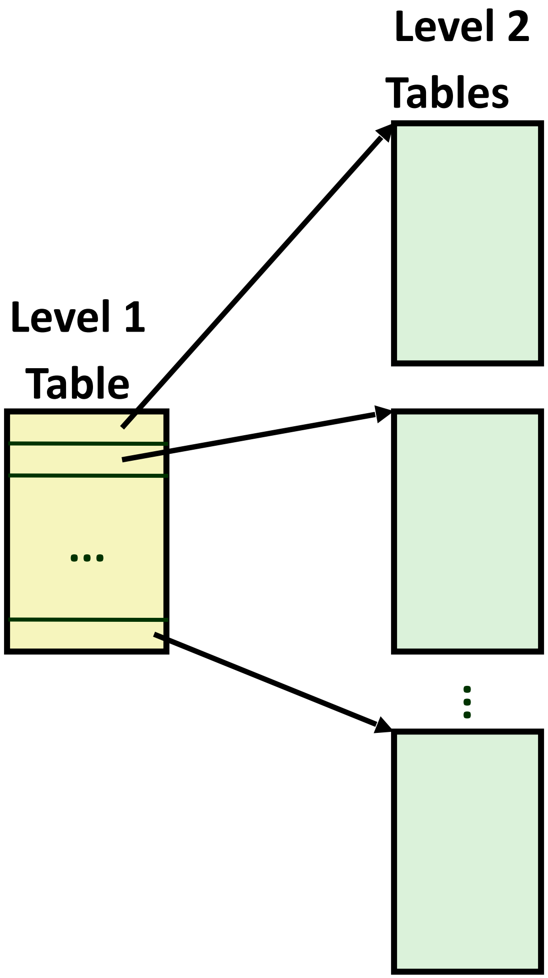
Solution: Page the Page Table

- Observation: Only a small number of pages (working set) are accessed during a certain period of time, due to locality
- Put only the relevant page table entires in main memory
- Idea: Put Page Table in Virtual Memory and swap it just like data

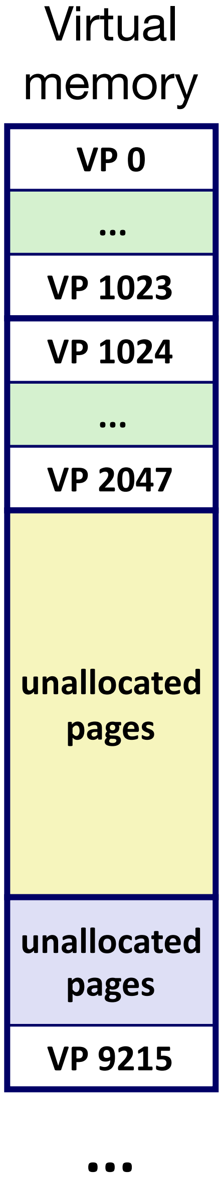


Effectively: A 2-Level Page Table

- Level 1 table:
  - Always in memory at a known location.
  - Each L1 PTE points to the start address of a L2 page table.
  - Bring that table to memory on-demand.
- Level 2 table:
  - Each PTE points to an actual data page

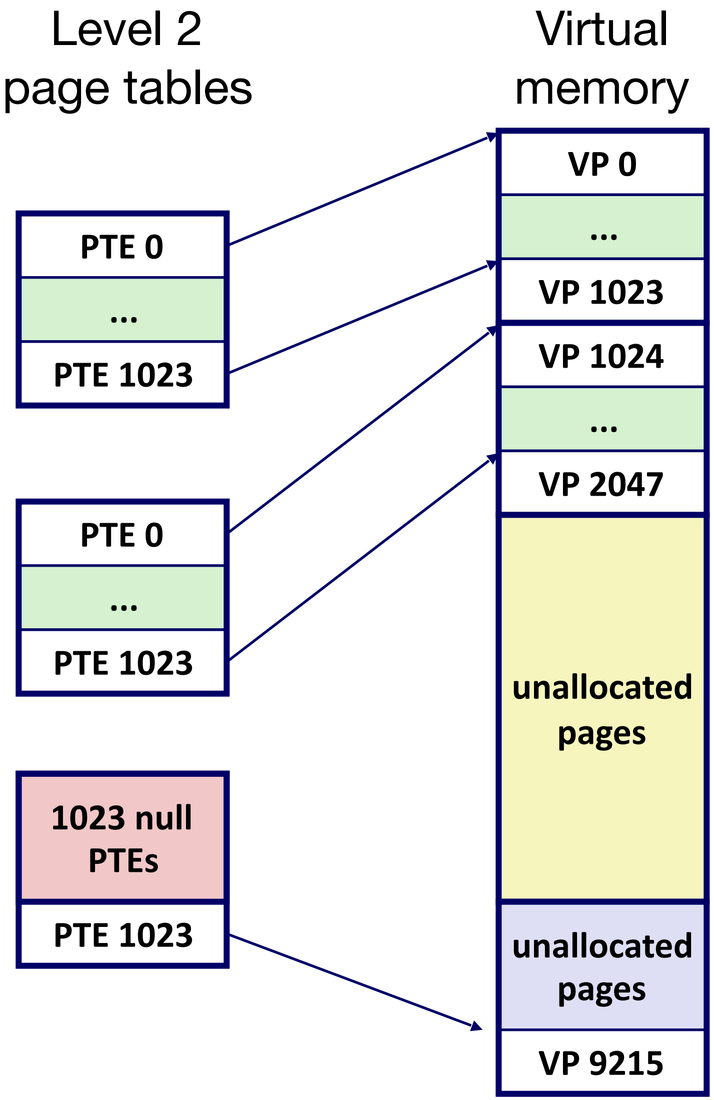


A Two-Level Page Table Hierarchy



32 bit addresses, 4KB pages, 4-byte PTEs

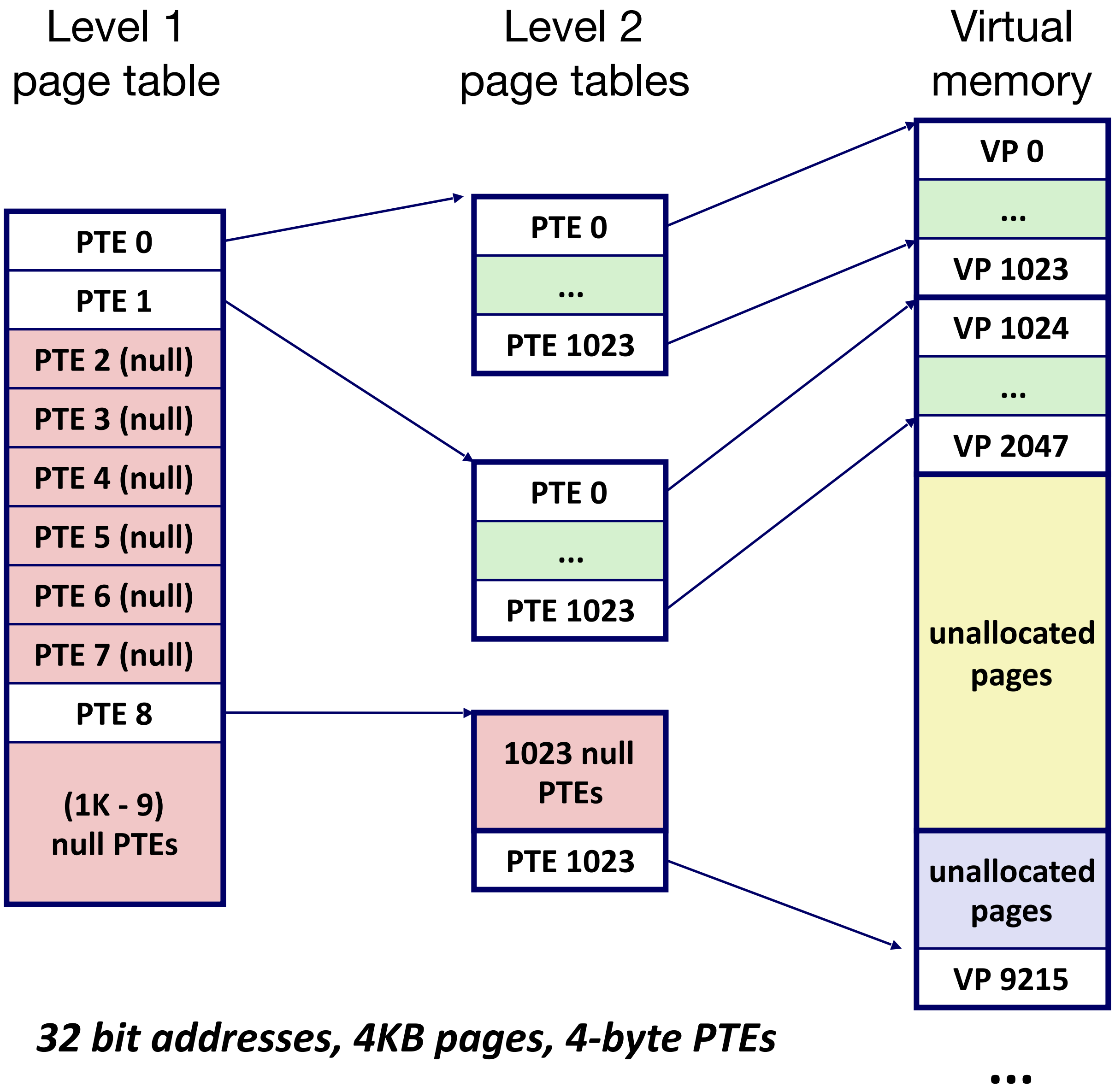
A Two-Level Page Table Hierarchy



32 bit addresses, 4KB pages, 4-byte PTEs

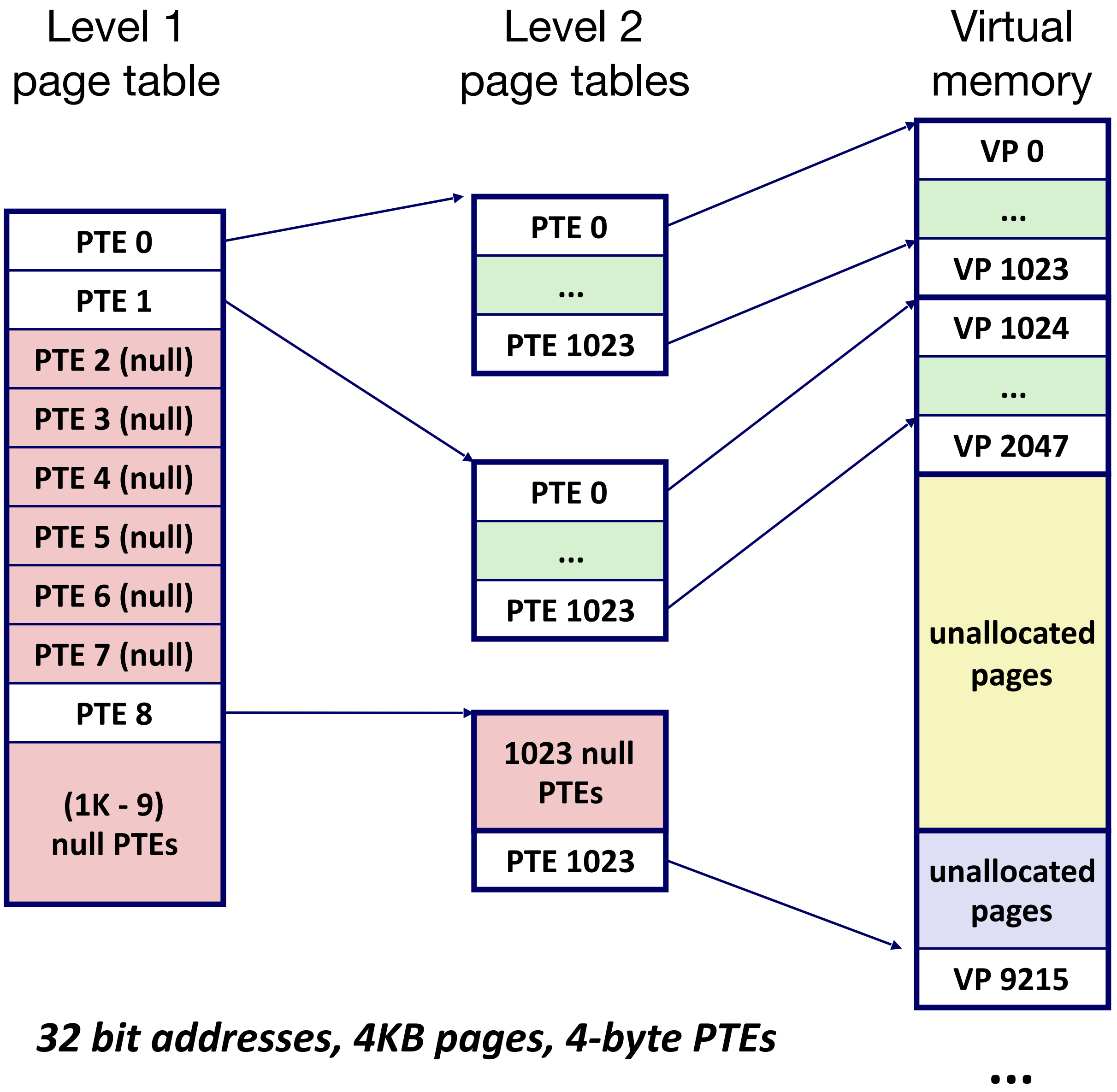
•••

A Two-Level Page Table Hierarchy



32 bit addresses, 4KB pages, 4-byte PTEs

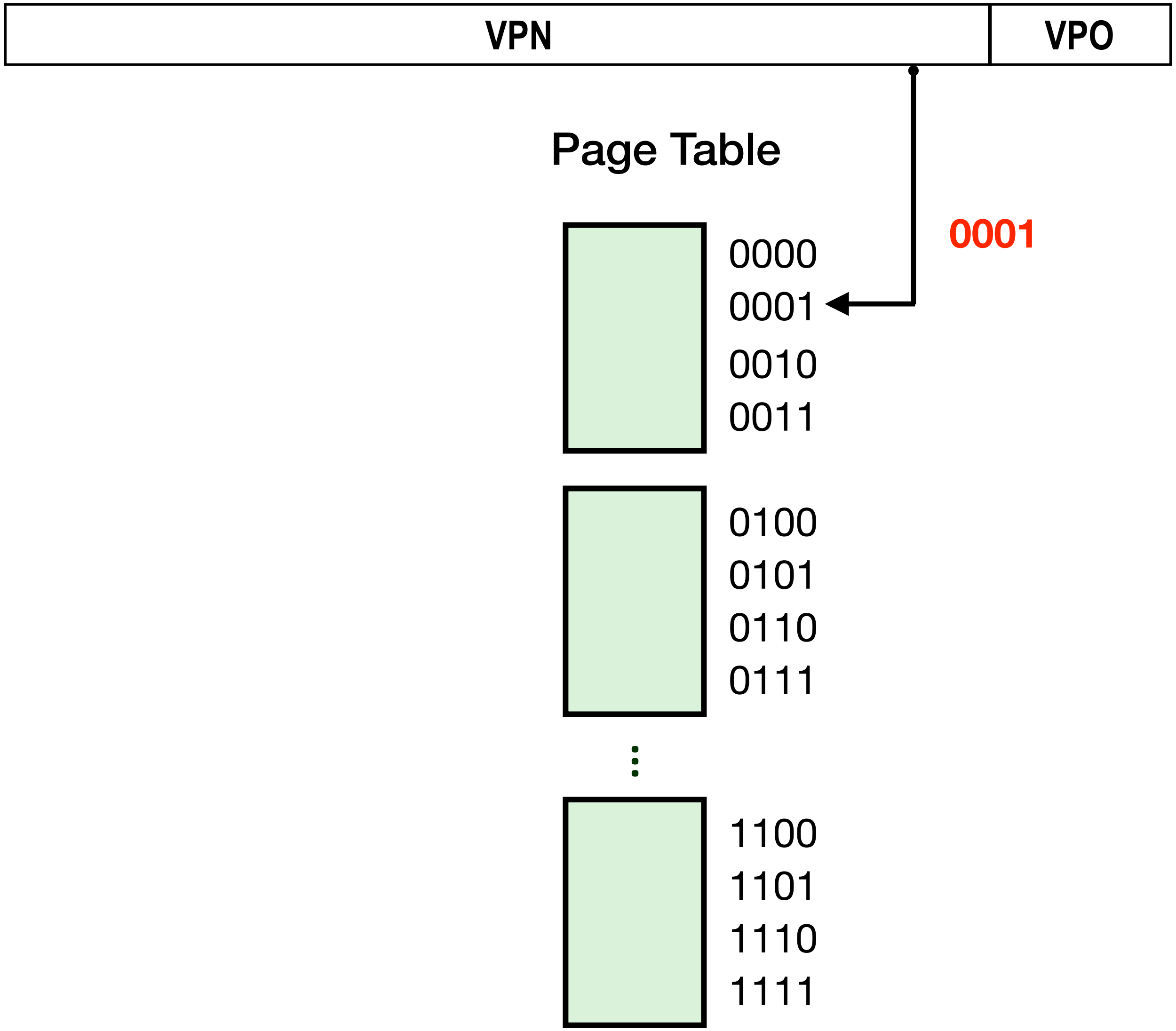
A Two-Level Page Table Hierarchy



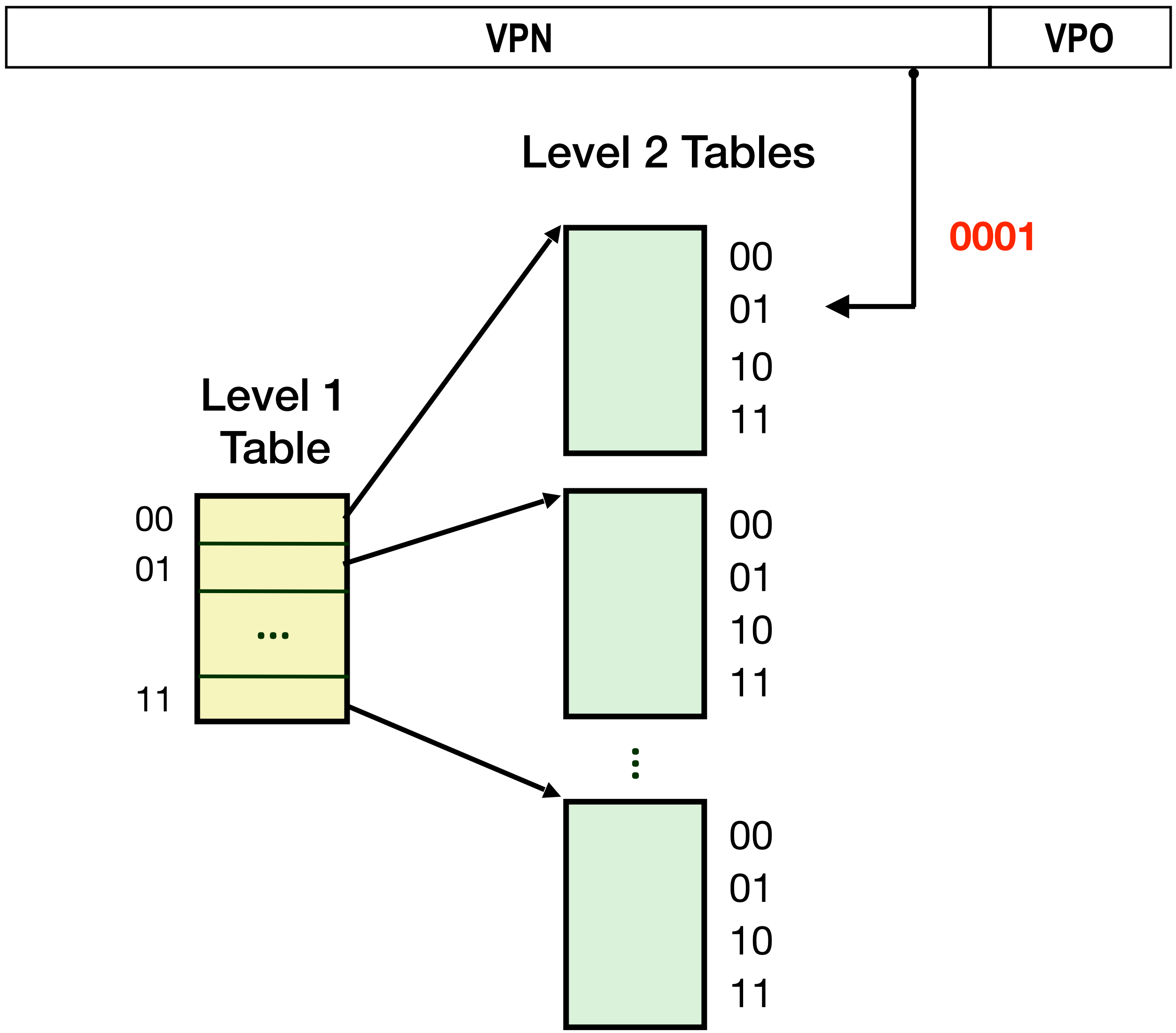
- Level 2 page table size:
  - $2^{32} / 2^{12} * 4 = 4$ MB
- Level 1 page table size:
  - $(2^{32} / 2^{12} * 4) / 2^{12} * 4 = 4$ KB

32 bit addresses, 4KB pages, 4-byte PTEs

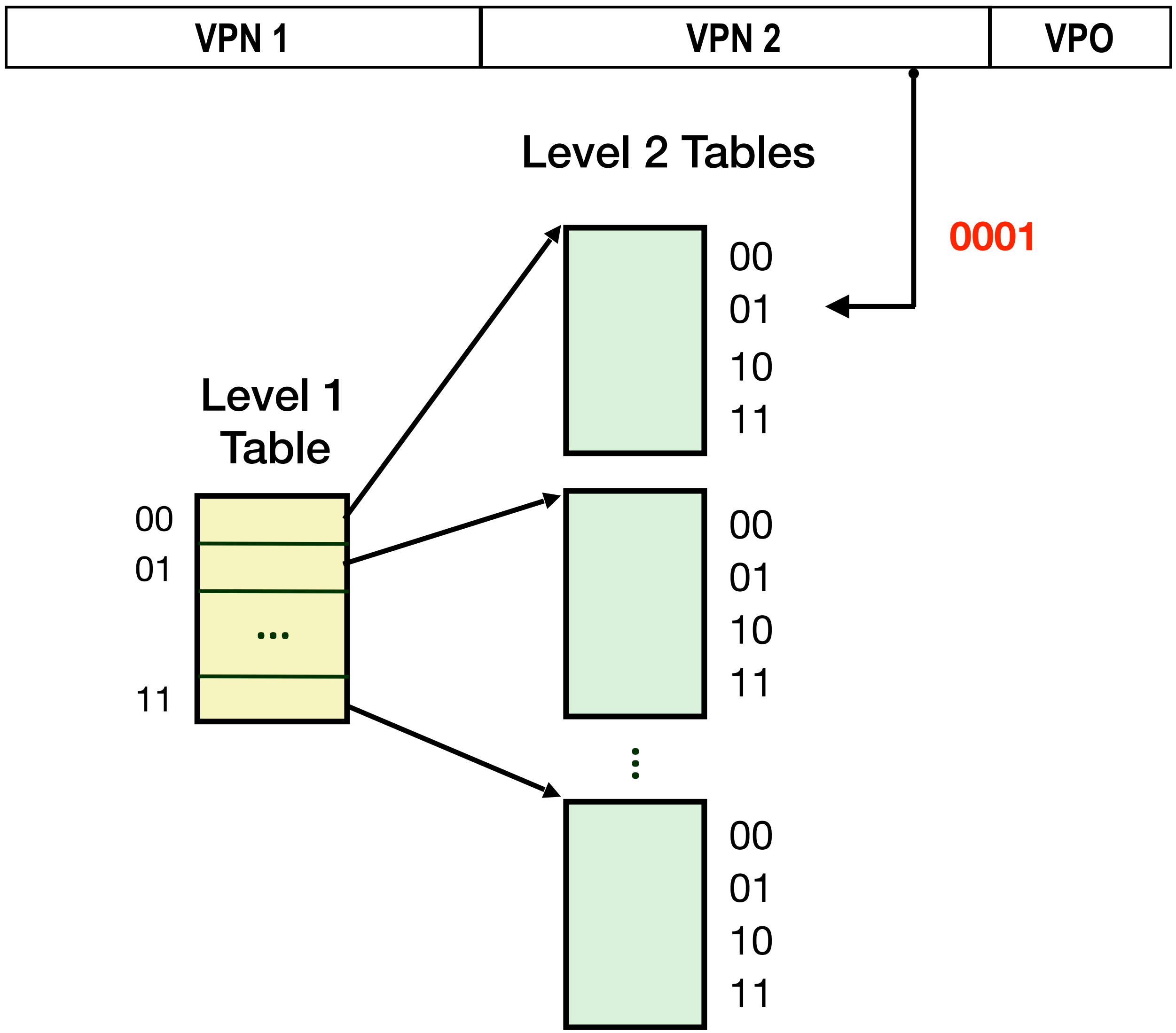
How to Access a 2-Level Page Table?



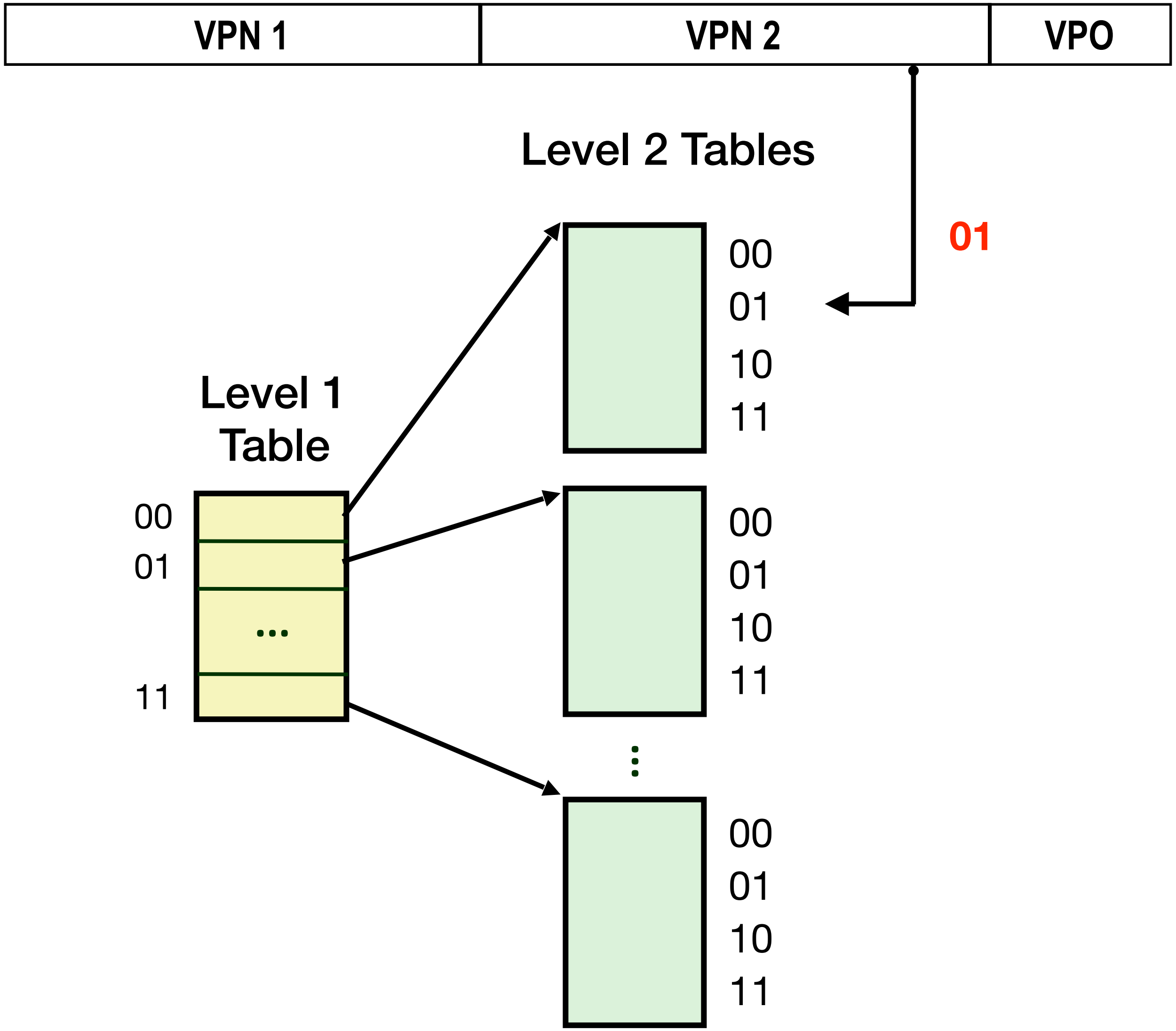
How to Access a 2-Level Page Table?



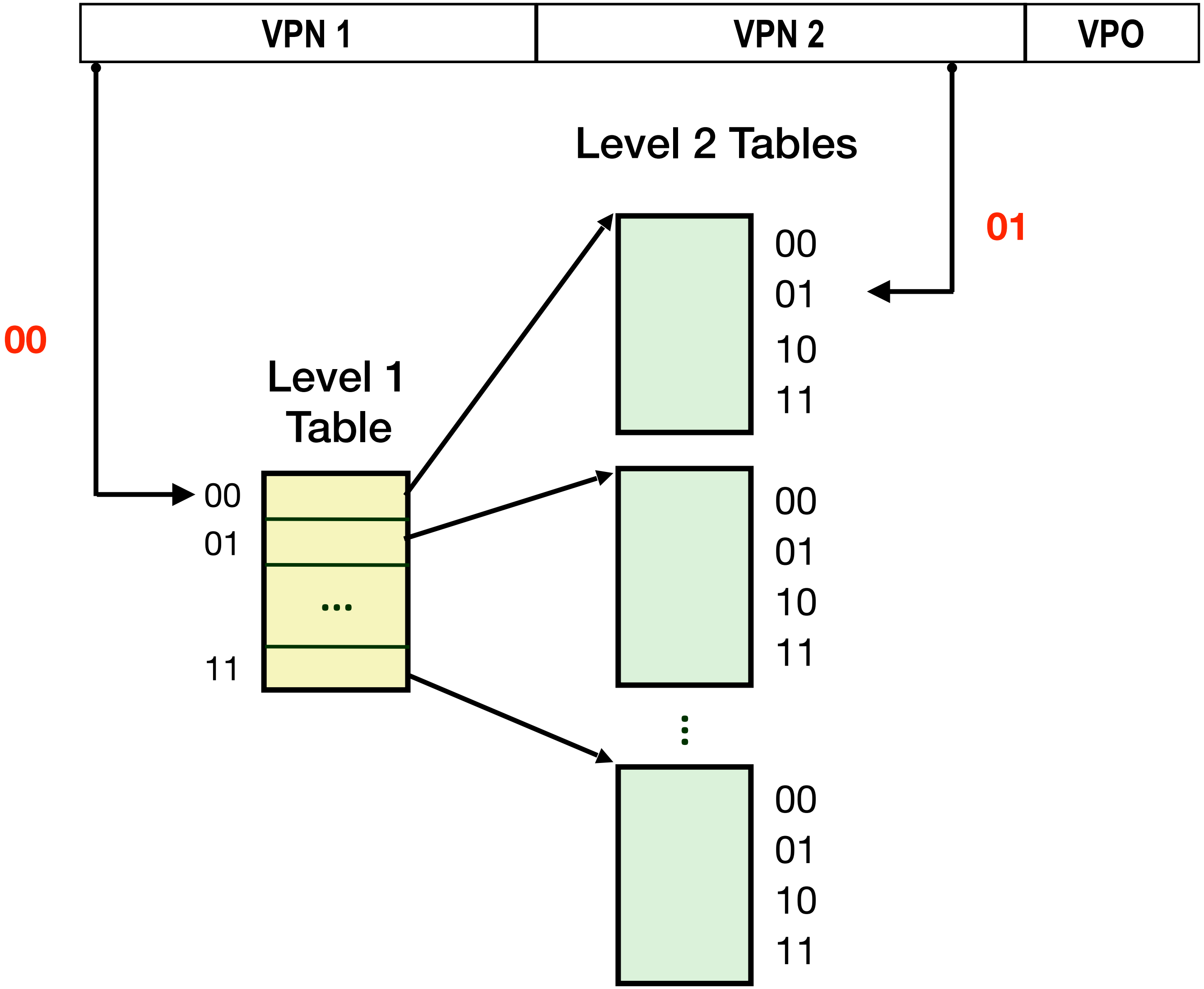
How to Access a 2-Level Page Table?



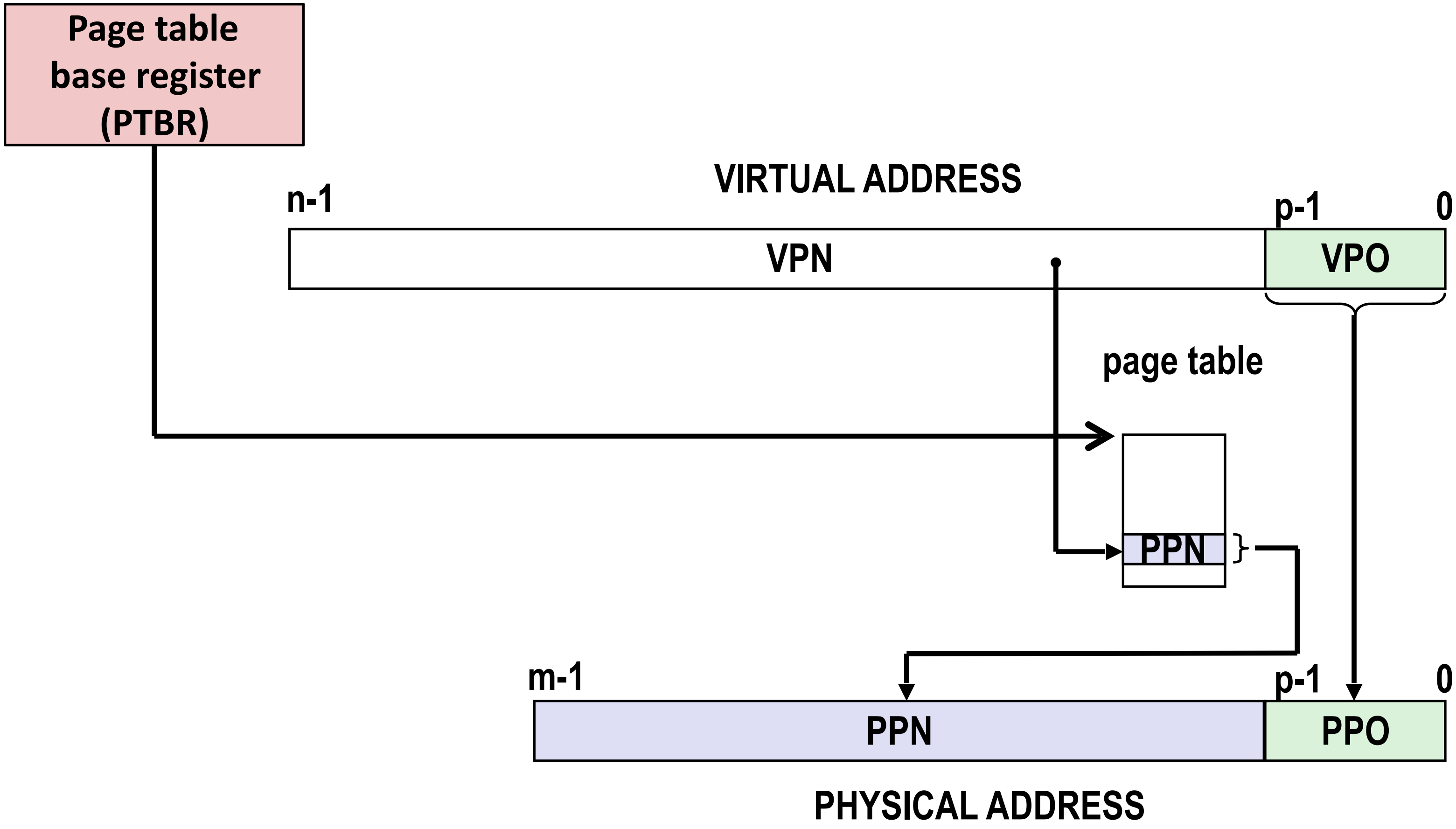
How to Access a 2-Level Page Table?



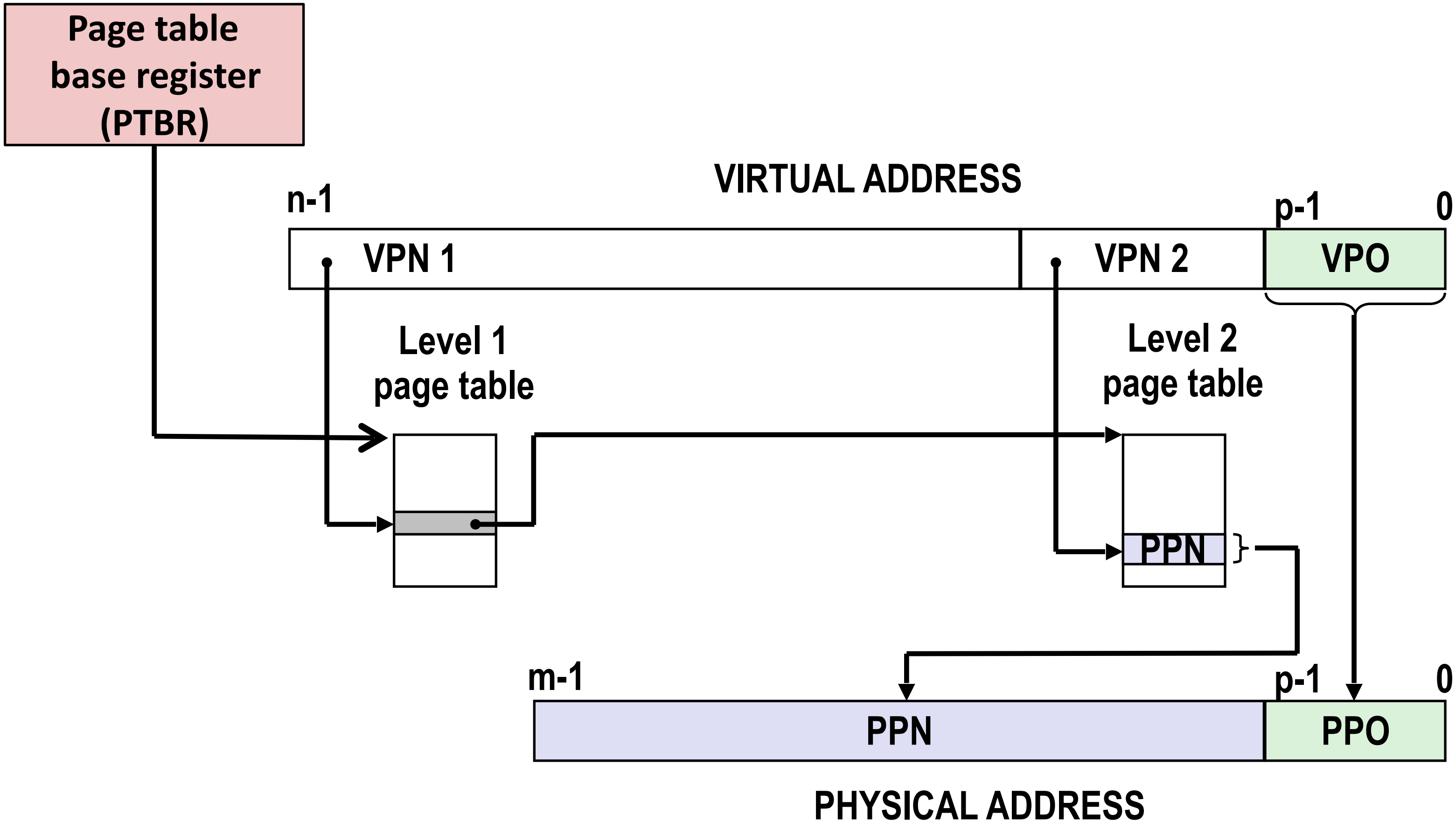
How to Access a 2-Level Page Table?



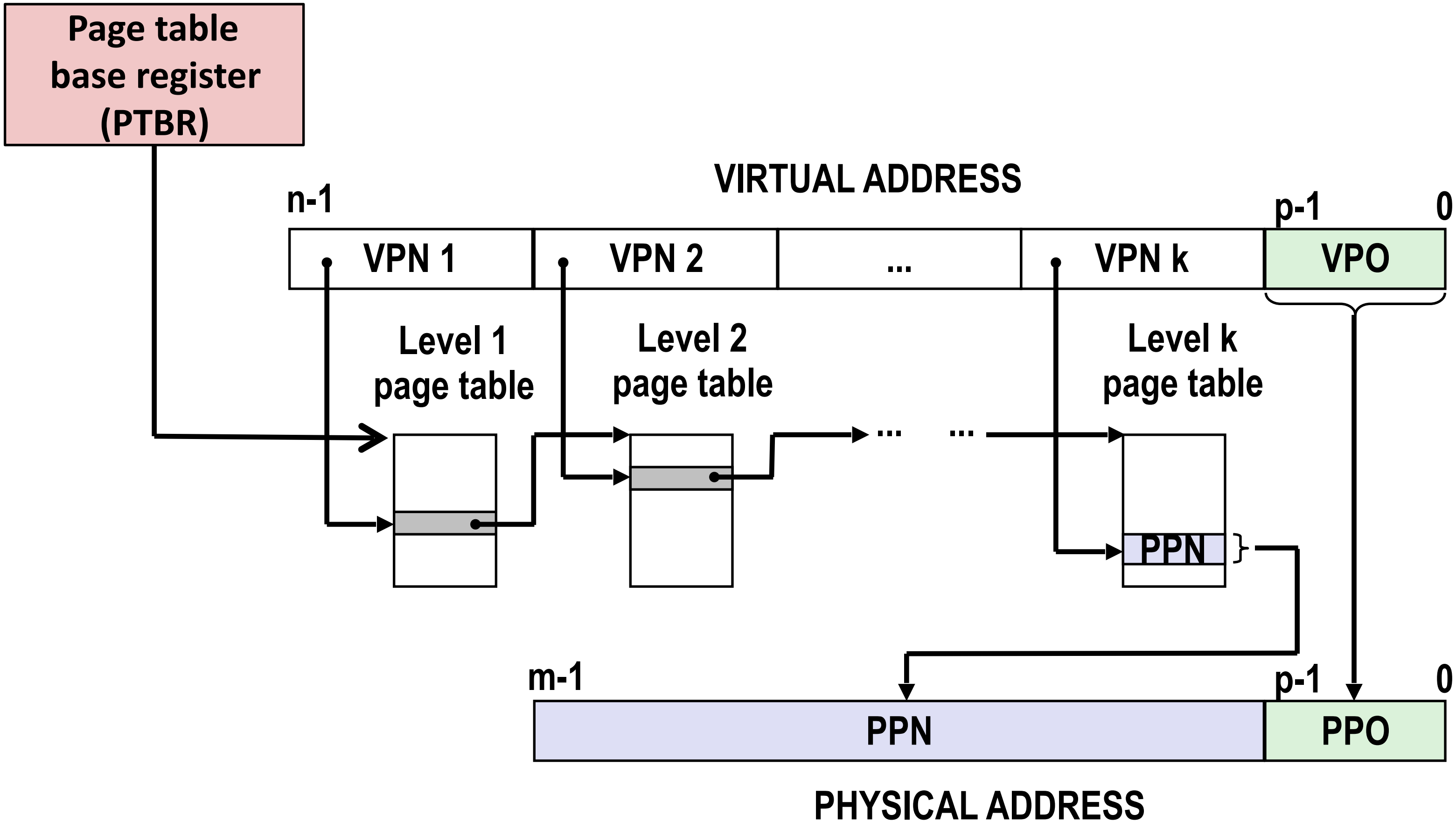
How to Access a 2-Level Page Table?



How to Access a 2-Level Page Table?



Translating with a k-level Page Table

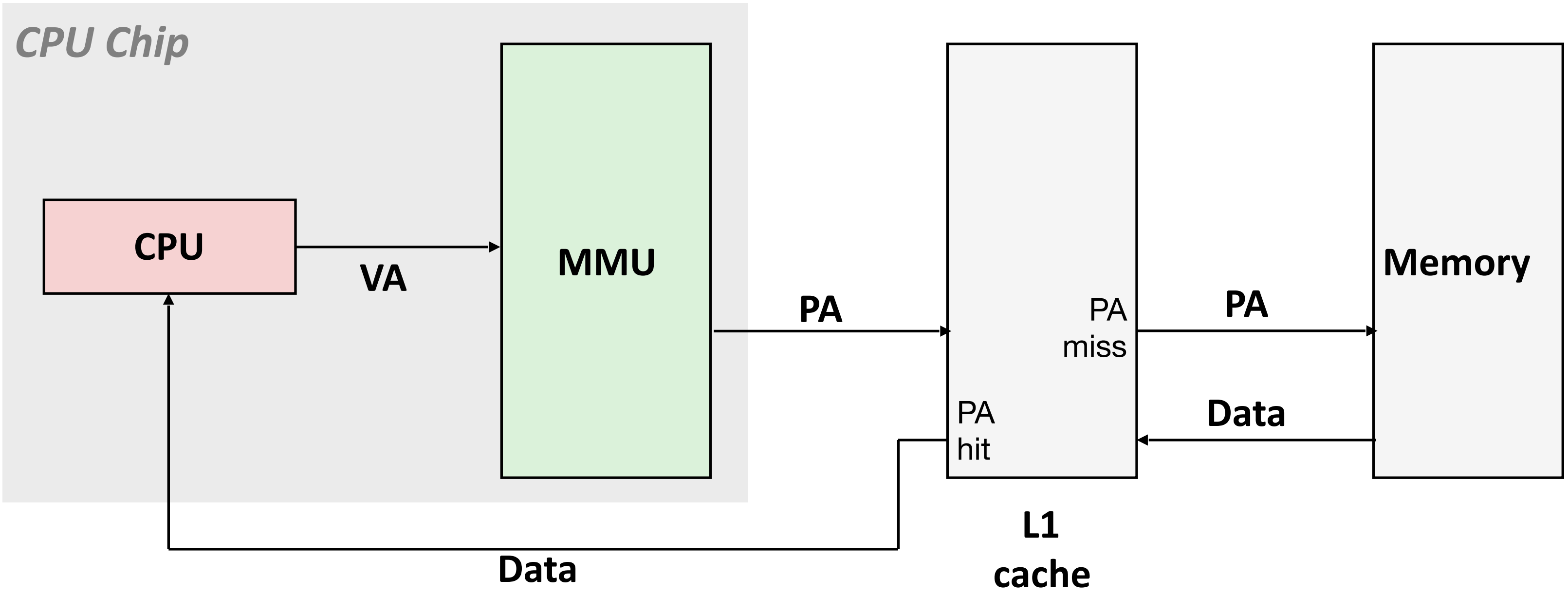


Today

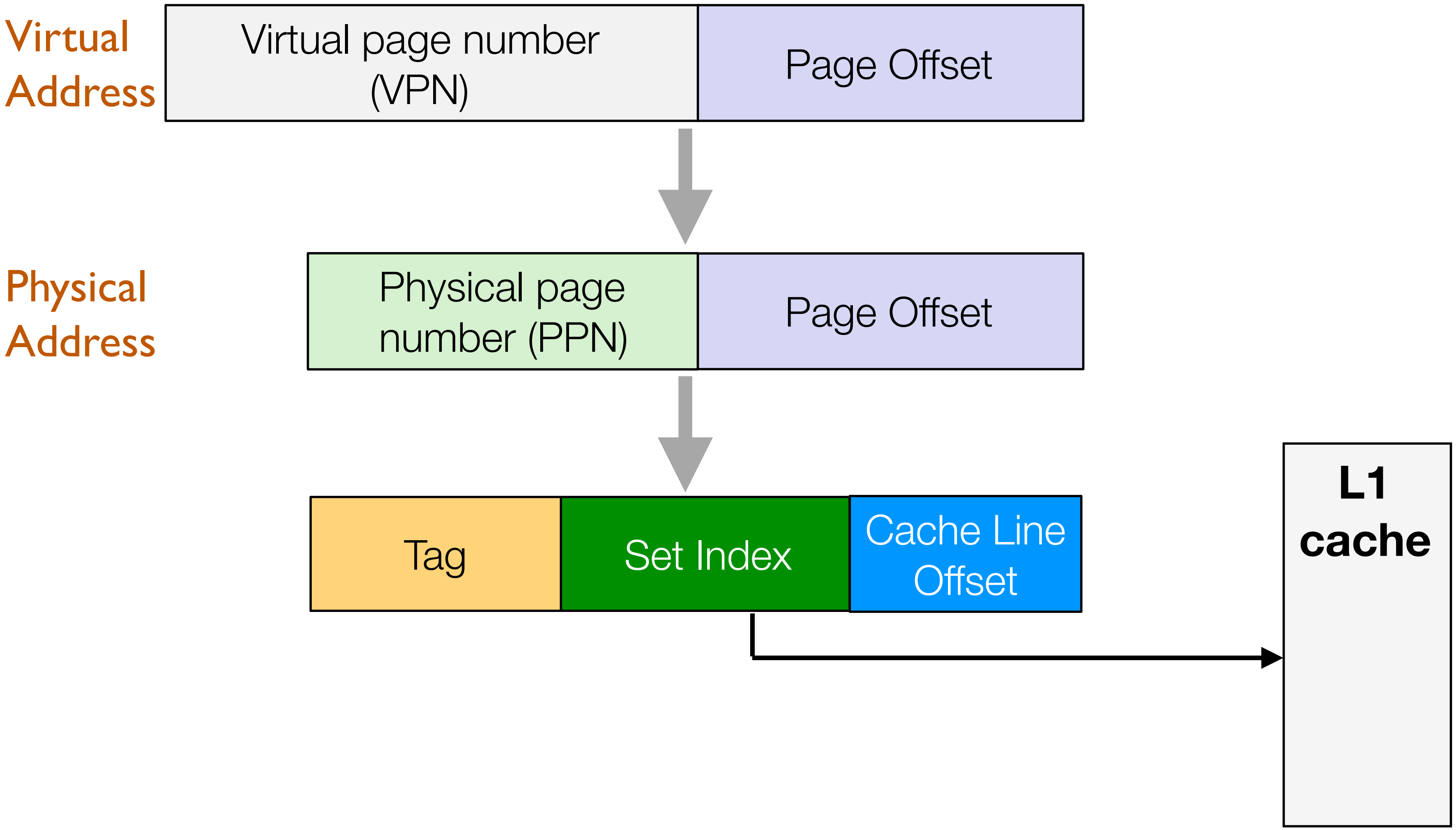
- Three Virtual Memory Optimizations
    - TLB
    - Page the page table (a.k.a., multi-level page table)
    - Virtually-indexed, physically-tagged cache
- Case-study: Intel Core i7/Linux example

Performance Issue in VM

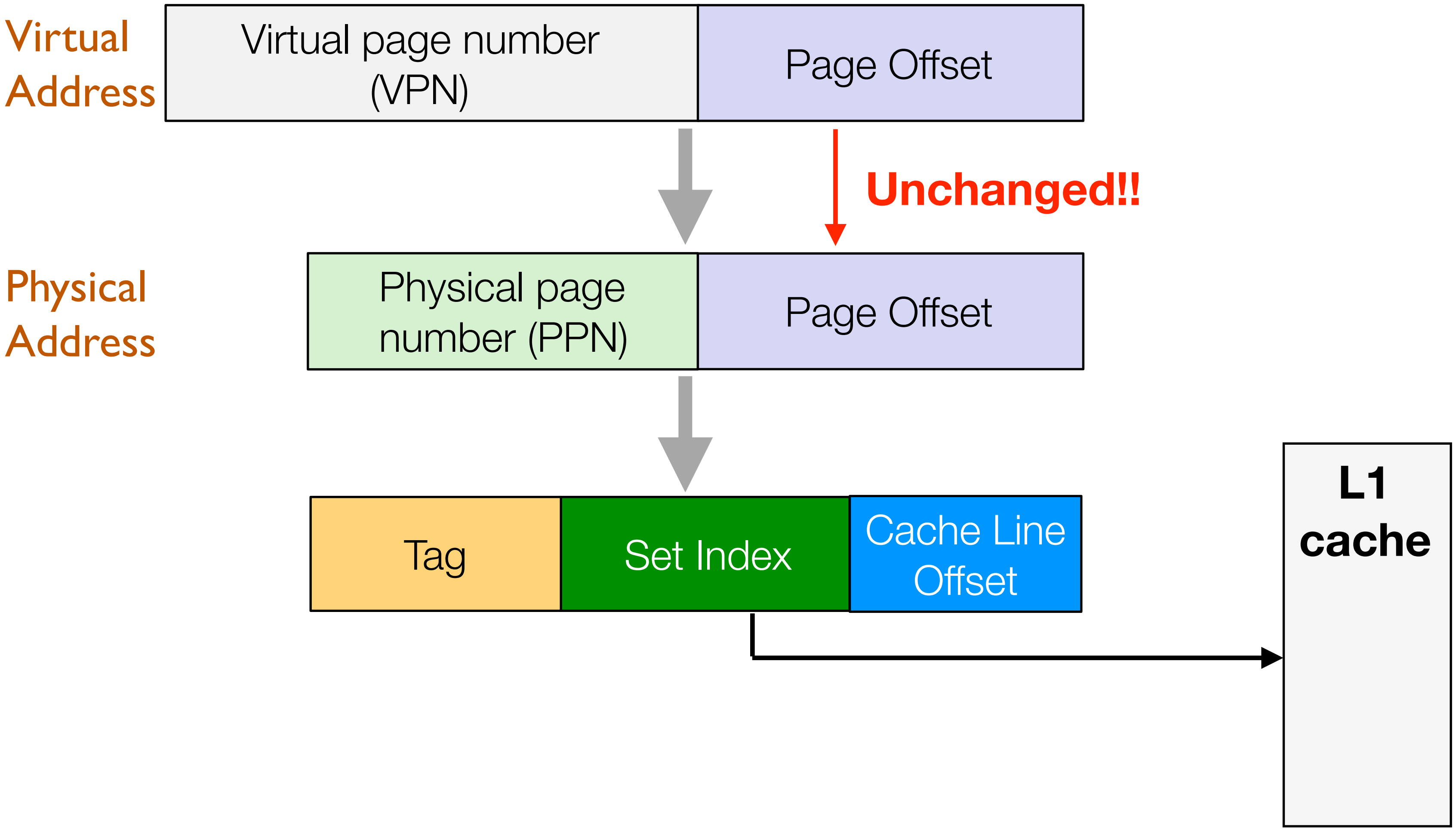
- Address translation and cache accesses are serialized
  - First translate from VA to PA
  - Then use PA to access cache
  - Slow! Can we speed it up?



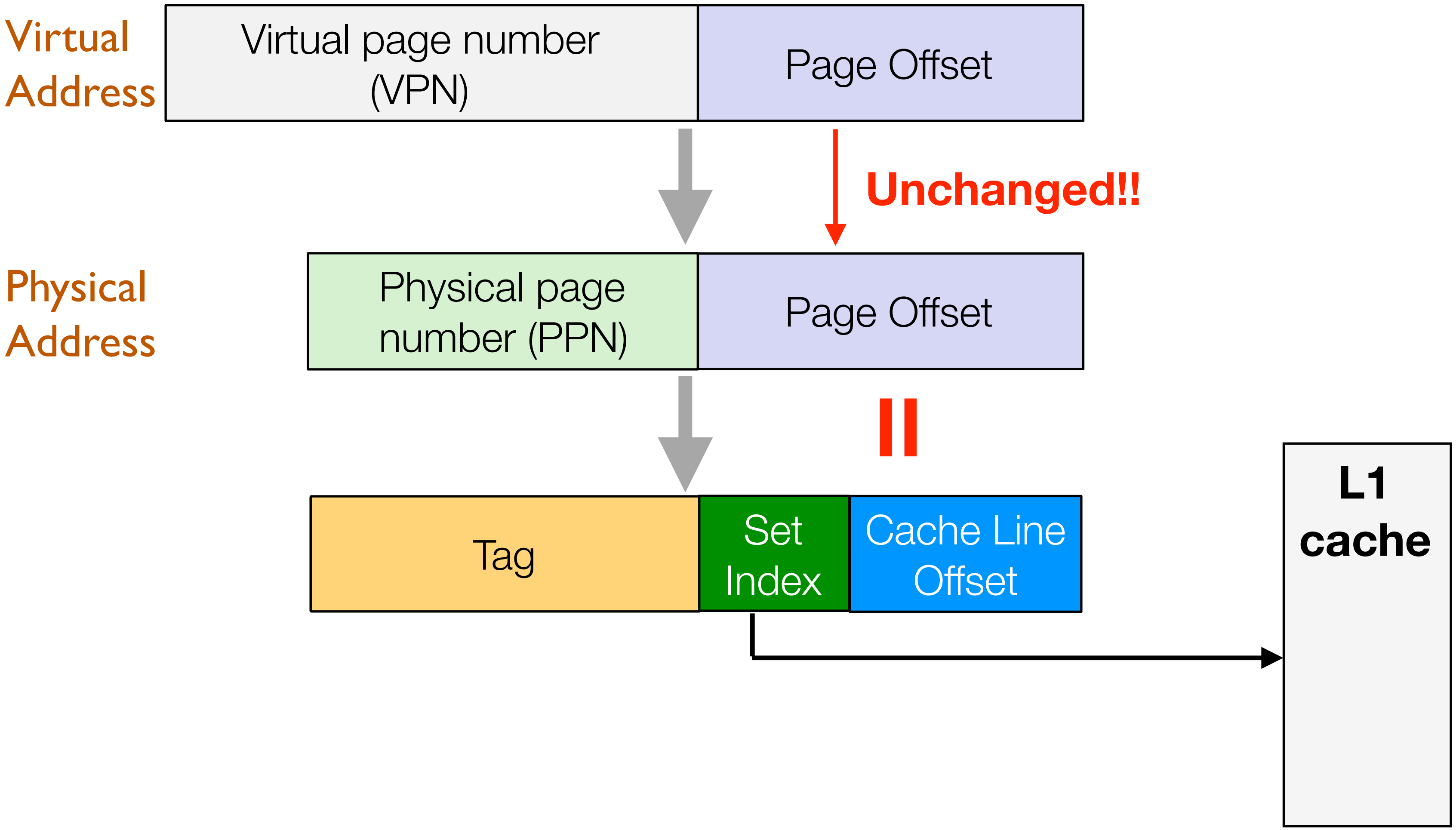
Performance Issue in VM



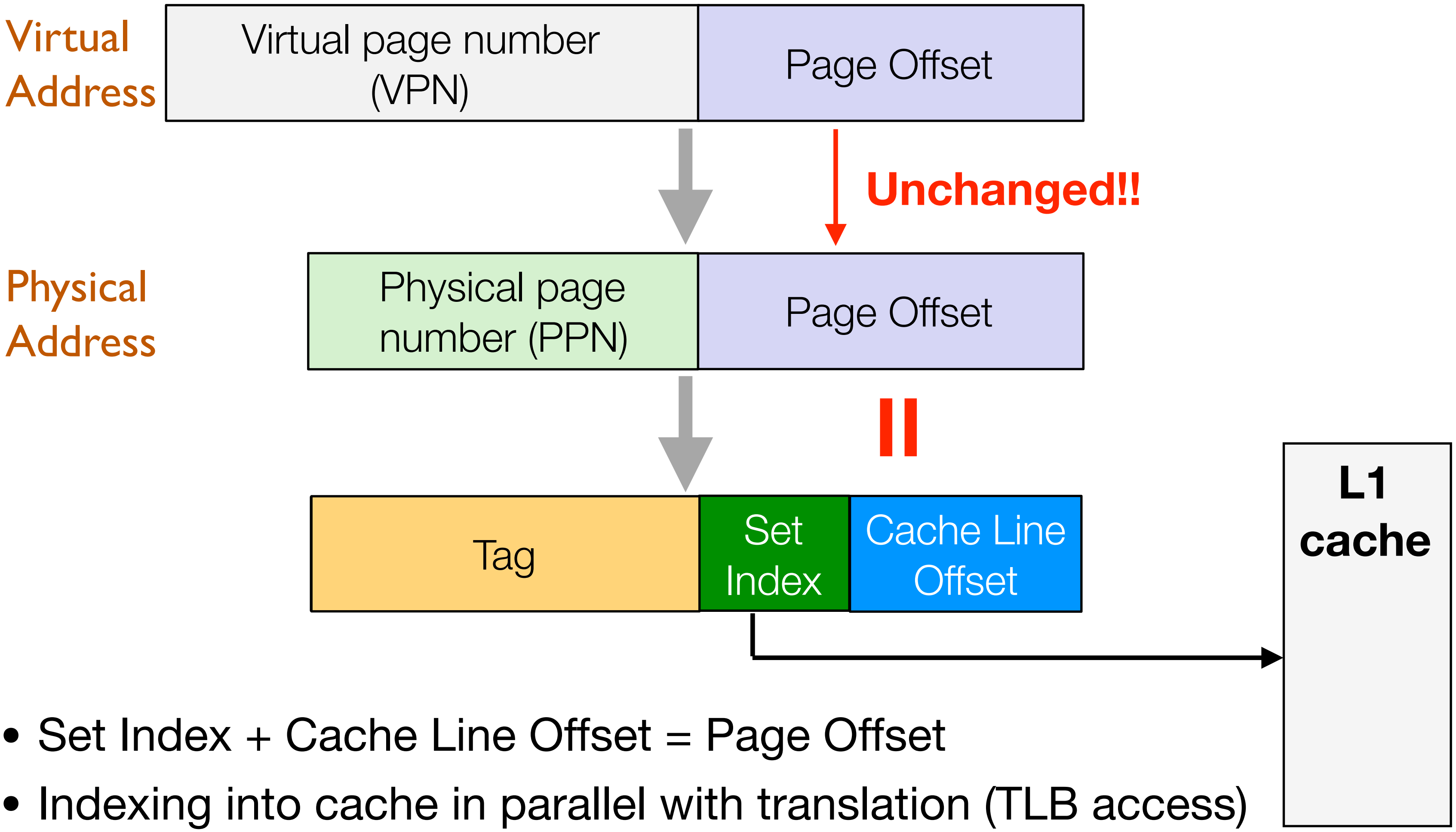
Performance Issue in VM



Performance Issue in VM

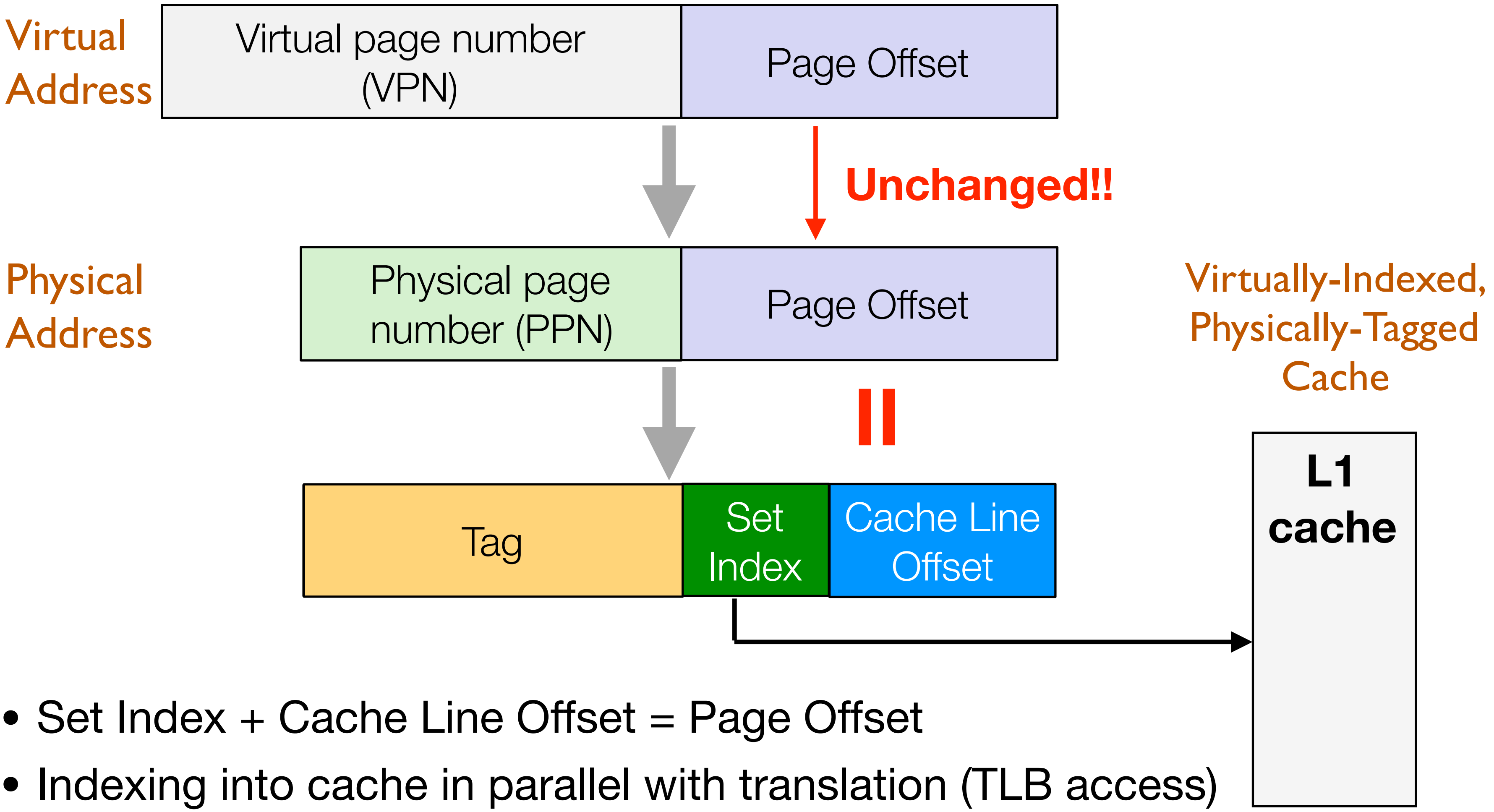


Performance Issue in VM



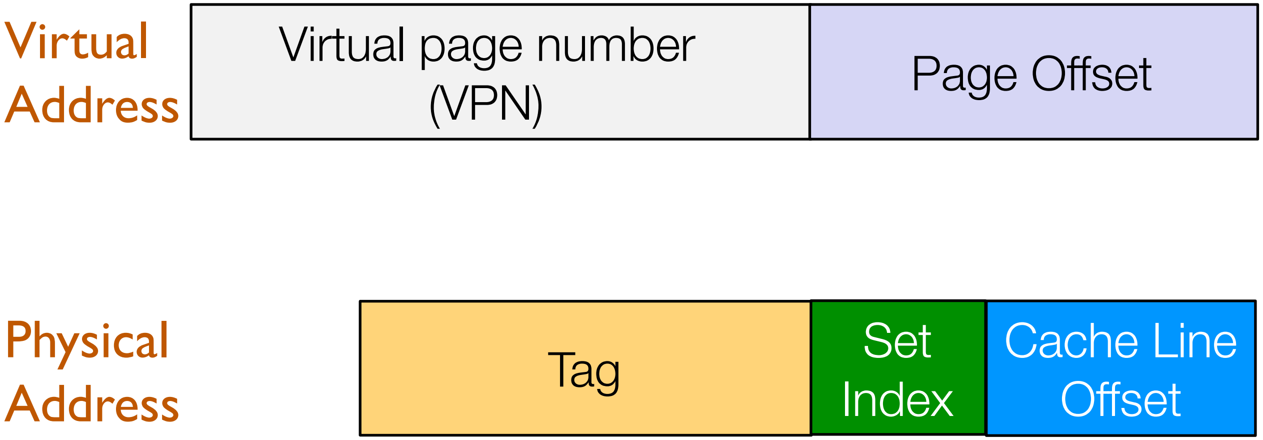
- Set Index + Cache Line Offset = Page Offset
- Indexing into cache in parallel with translation (TLB access)
- If TLB hits, can get the data back in one cycle

Performance Issue in VM



- Set Index + Cache Line Offset = Page Offset
- Indexing into cache in parallel with translation (TLB access)
- If TLB hits, can get the data back in one cycle

Any Implications?



Virtual Address

Virtual page number (VPN)

Page Offset

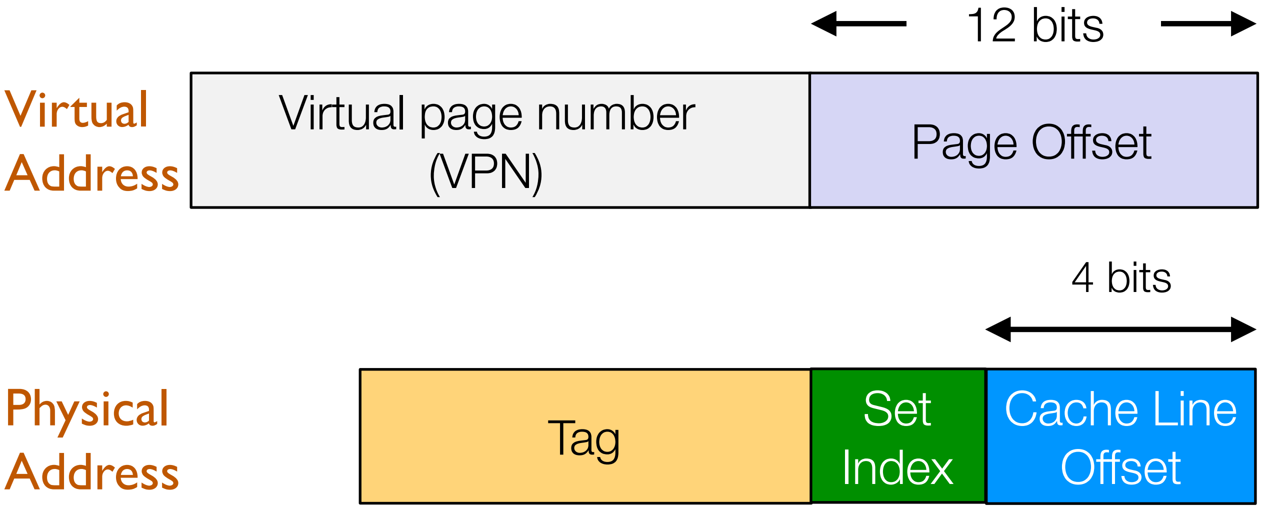
Physical Address

Tag

Set Index

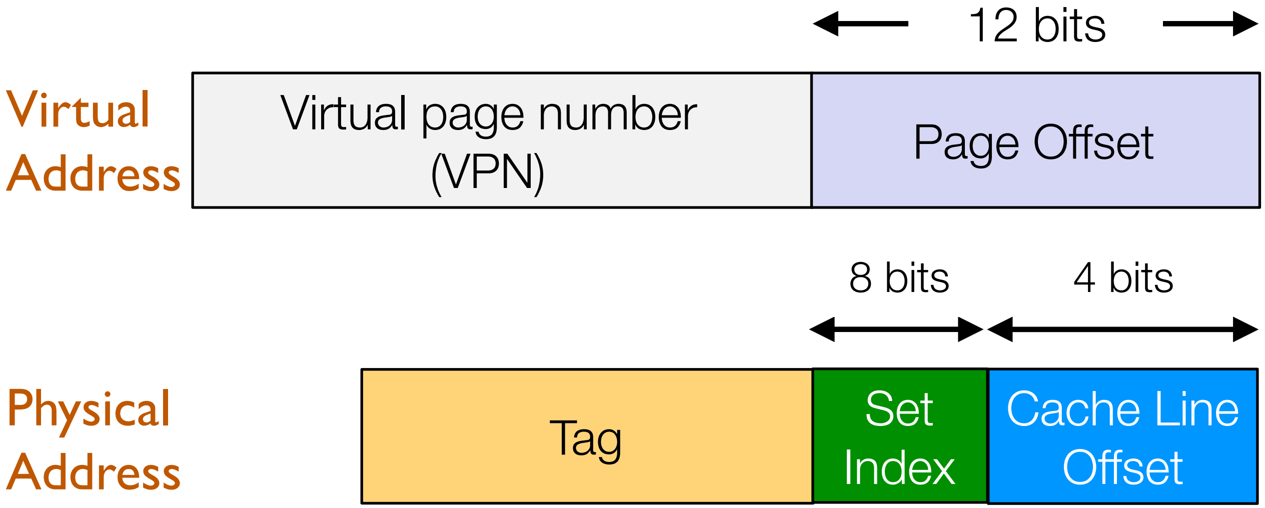
Cache Line Offset

Any Implications?



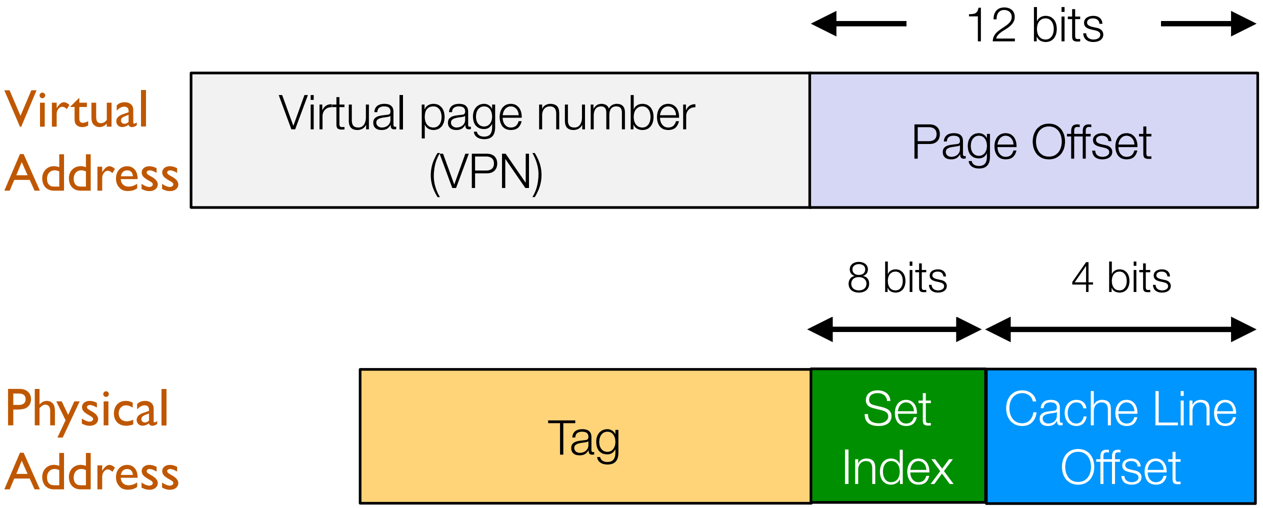
- Assuming 4K page size, cache line size is 16 bytes.

Any Implications?



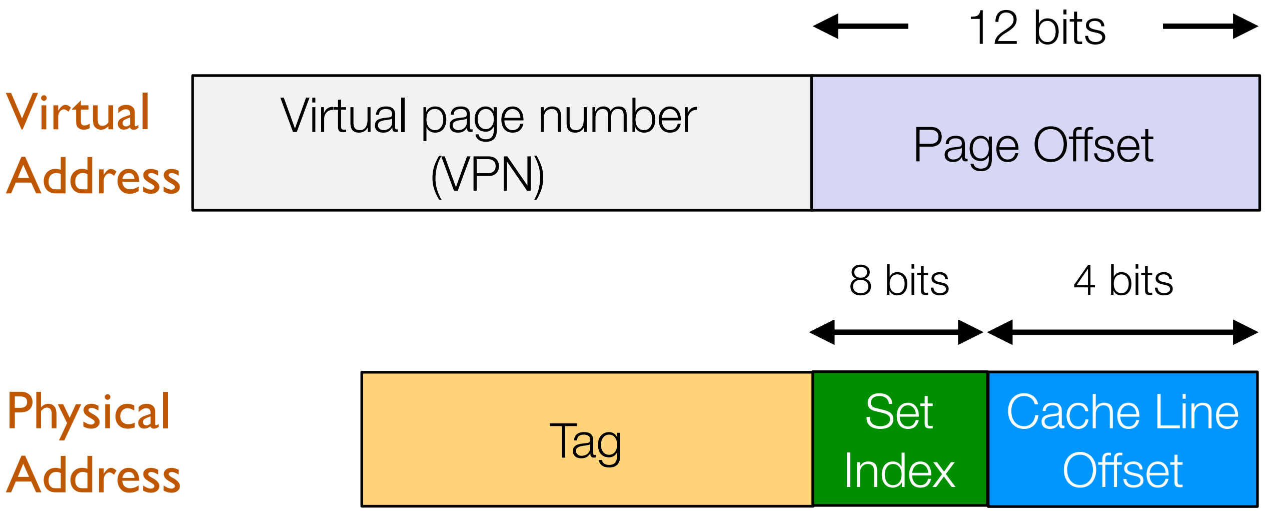
- Assuming 4K page size, cache line size is 16 bytes.
- Set Index = 8 bits. Can only have 256 Sets => Limit cache size

Any Implications?



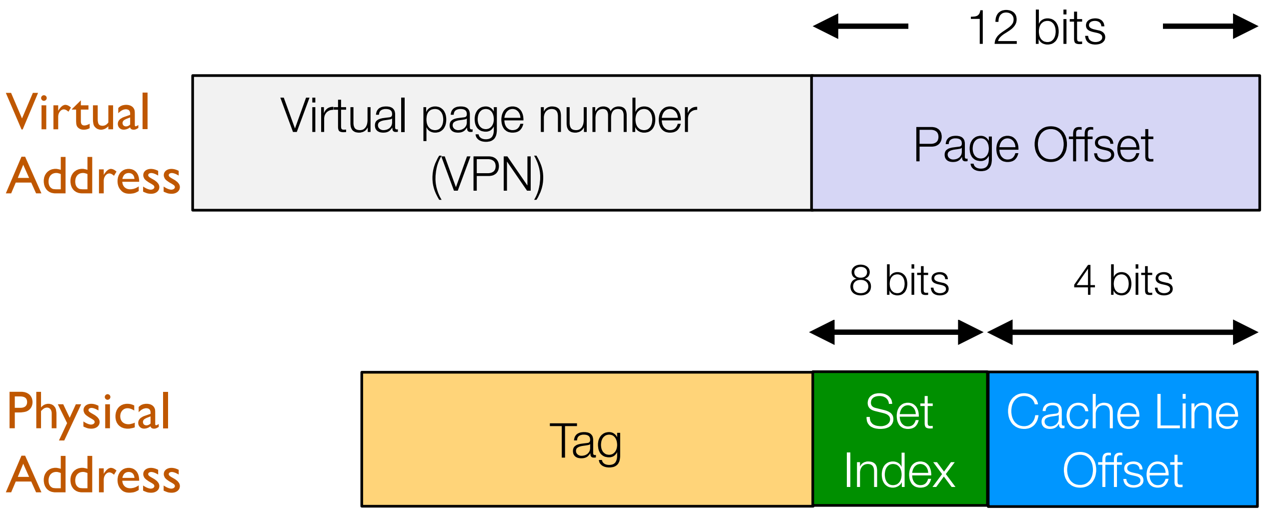
- Assuming 4K page size, cache line size is 16 bytes.
- Set Index = 8 bits. Can only have 256 Sets => Limit cache size
- Increasing cache size then requires increasing associativity

Any Implications?



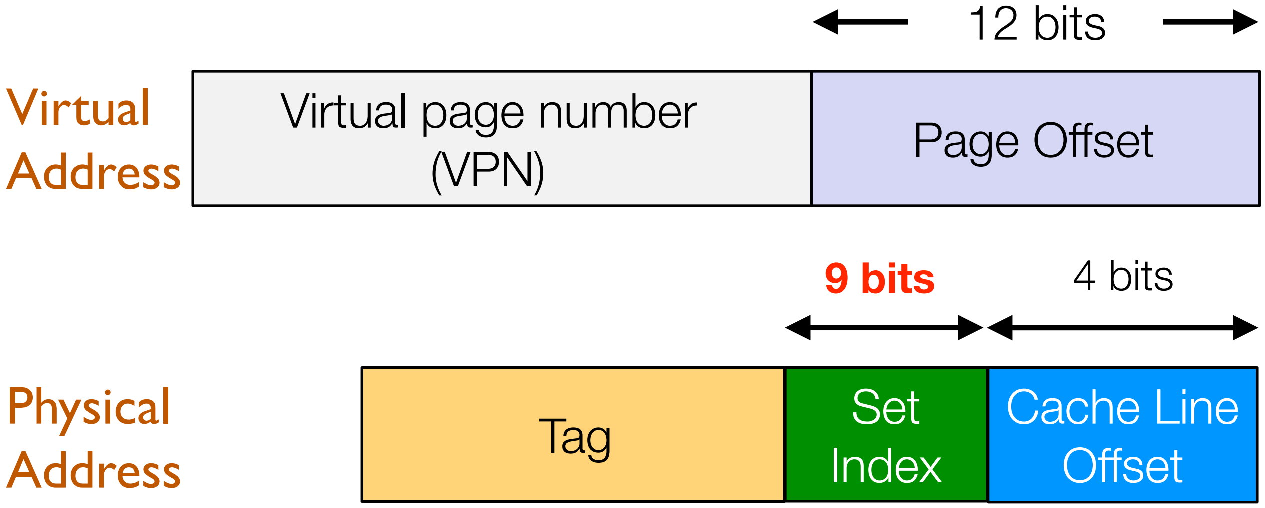
- Assuming 4K page size, cache line size is 16 bytes.
- Set Index = 8 bits. Can only have 256 Sets => Limit cache size
- Increasing cache size then requires increasing associativity
    - Not ideal because that requires comparing more tags

Any Implications?



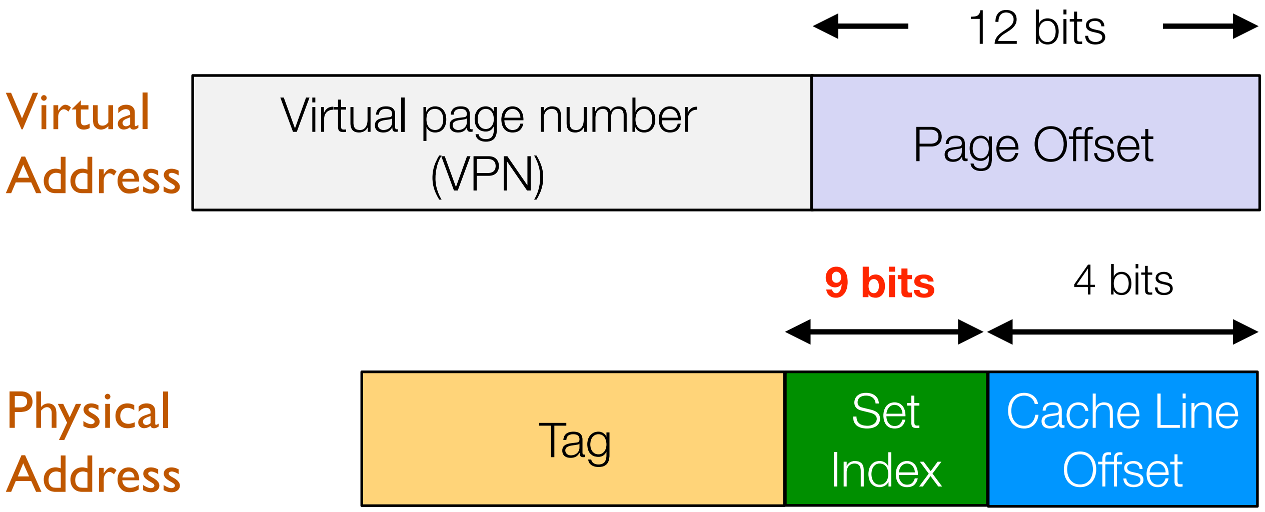
- Assuming 4K page size, cache line size is 16 bytes.
- Set Index = 8 bits. Can only have 256 Sets => Limit cache size
- Increasing cache size then requires increasing associativity
    - Not ideal because that requires comparing more tags
- Solutions?

Any Implications?



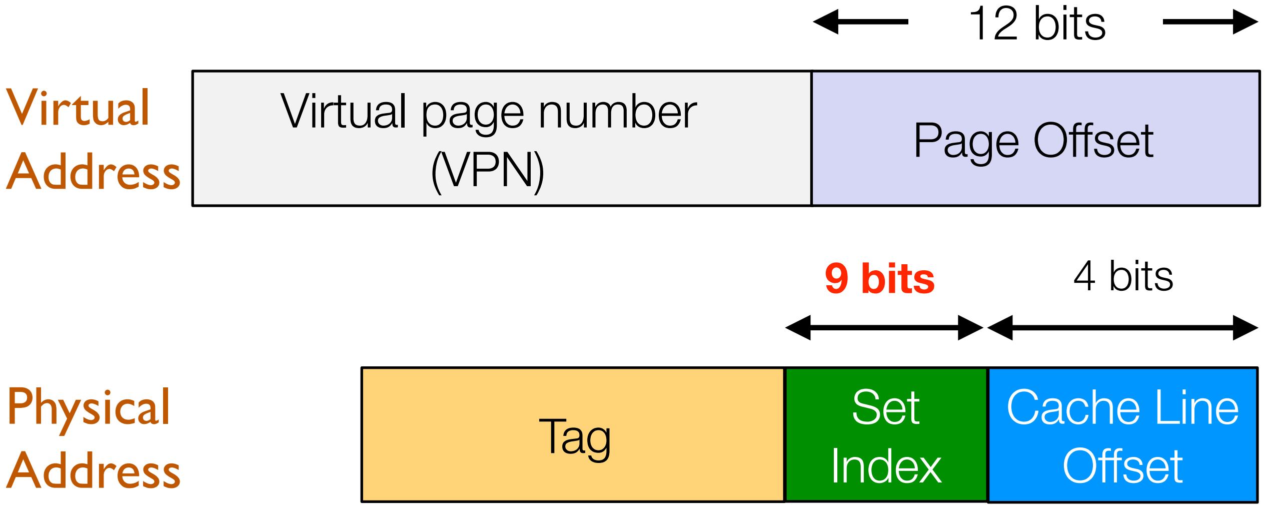
- What if we use 9 bits for Set Index? More Sets now.

Any Implications?



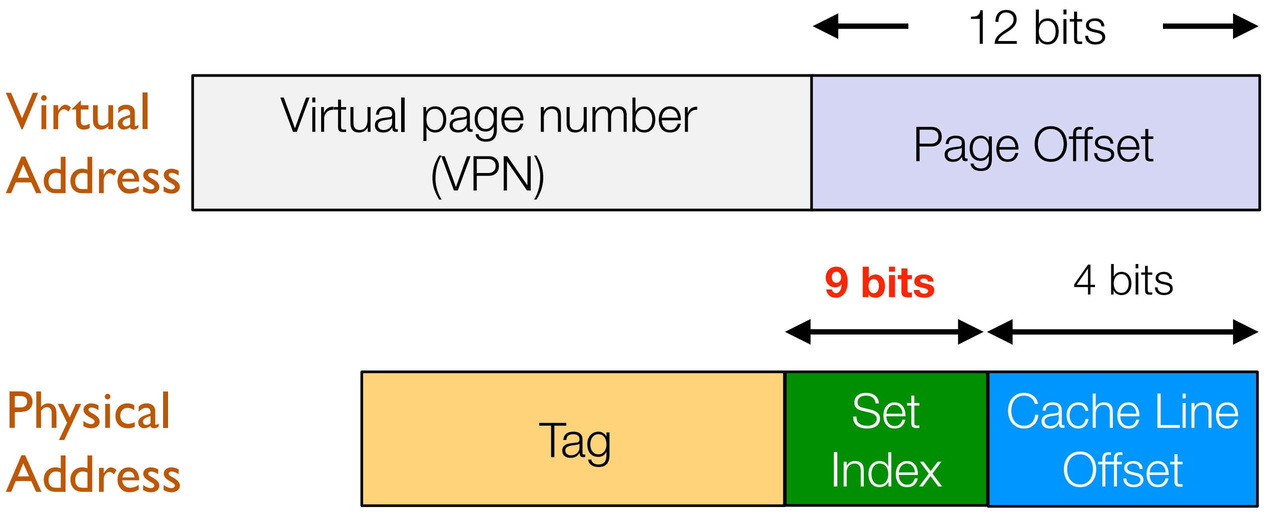
- What if we use 9 bits for Set Index? More Sets now.
- How can this still work???

Any Implications?



- What if we use 9 bits for Set Index? More Sets now.
- How can this still work???
- The least significant bit in VPN and PPN must be the same

Any Implications?

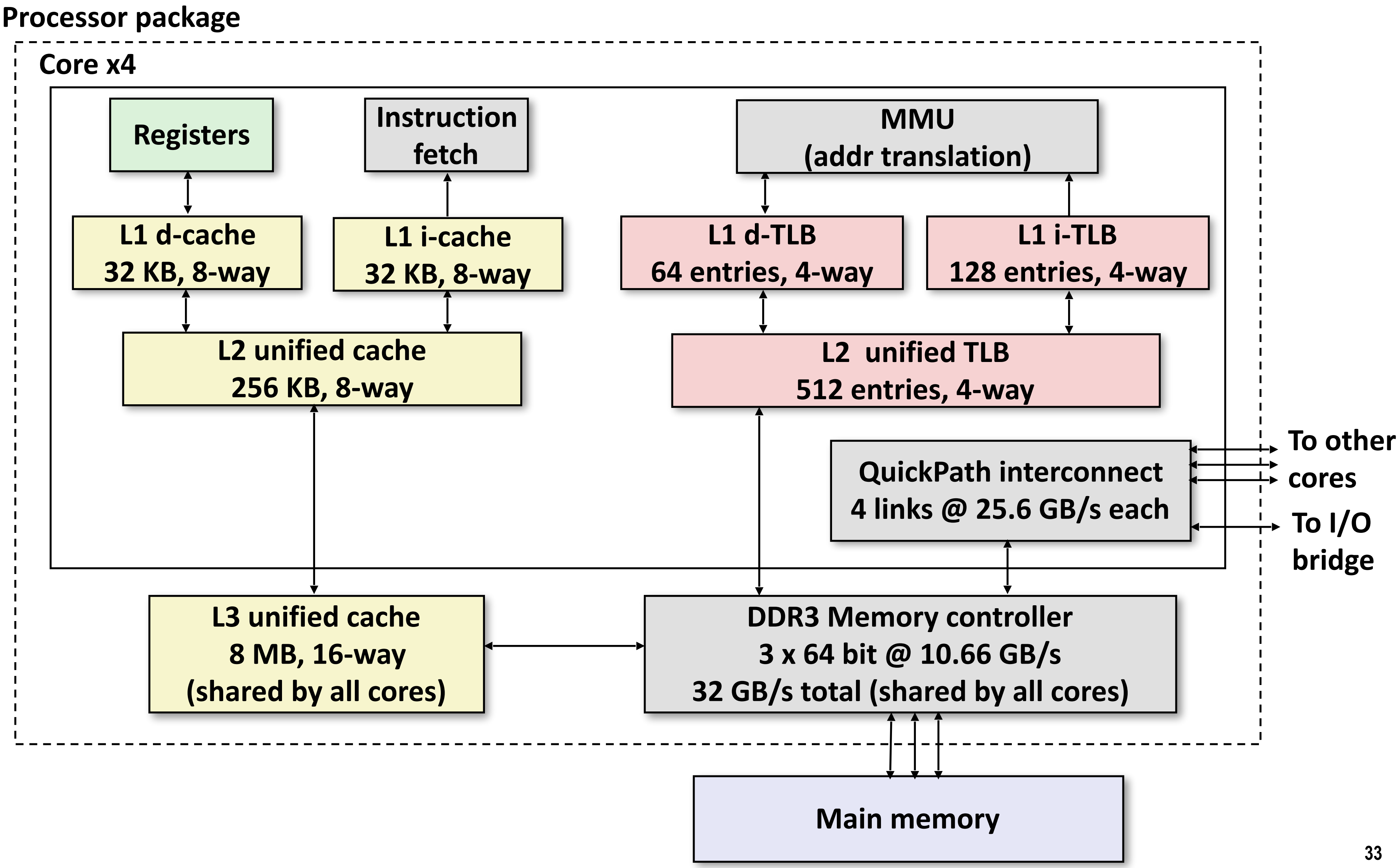


- What if we use 9 bits for Set Index? More Sets now.
- How can this still work???
- The least significant bit in VPN and PPN must be the same
- That is: an even VA must be mapped to an even PA, and an odd VA must be mapped to an odd PA

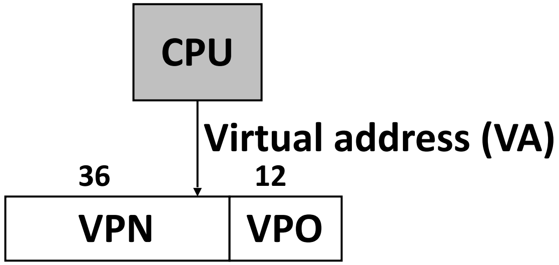
Today

- Three Virtual Memory Optimizations
  - TLB
  - Page the page table (a.k.a., multi-level page table)
  - Virtually-indexed, physically-tagged cache
- Case-study: Intel Core i7/Linux example

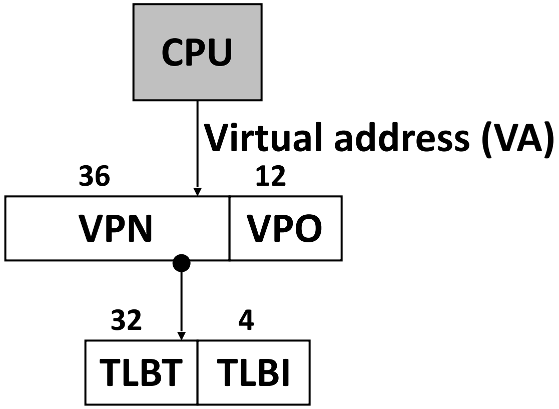
Intel Core i7 Memory System



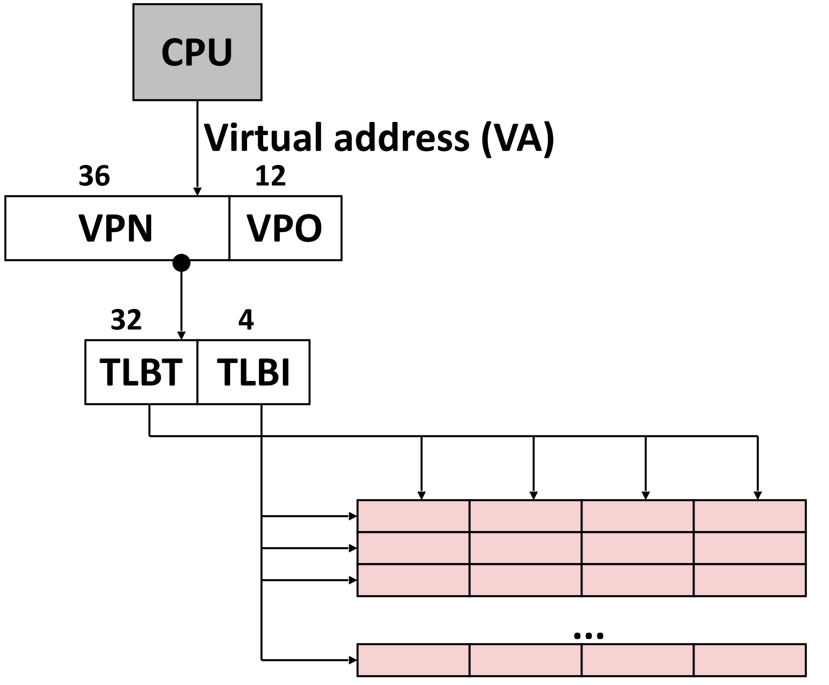
End-to-End Core i7 Address Translation



End-to-End Core i7 Address Translation

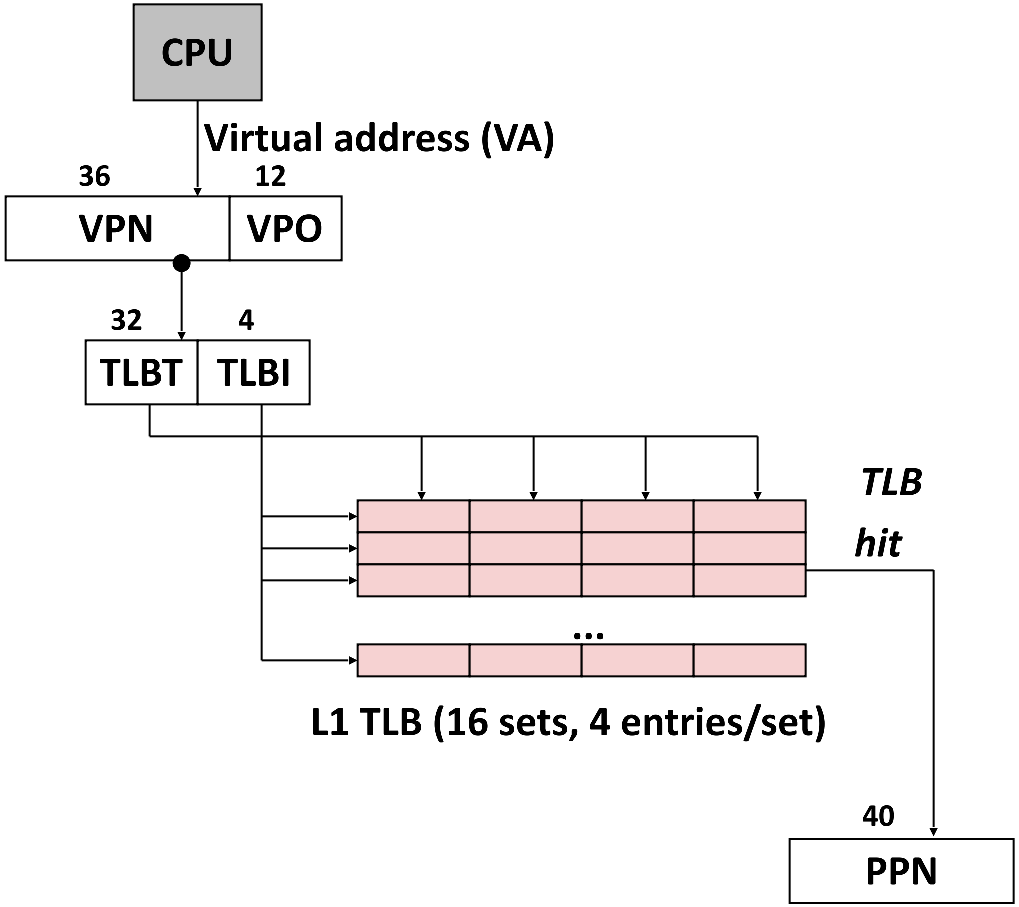


End-to-End Core i7 Address Translation

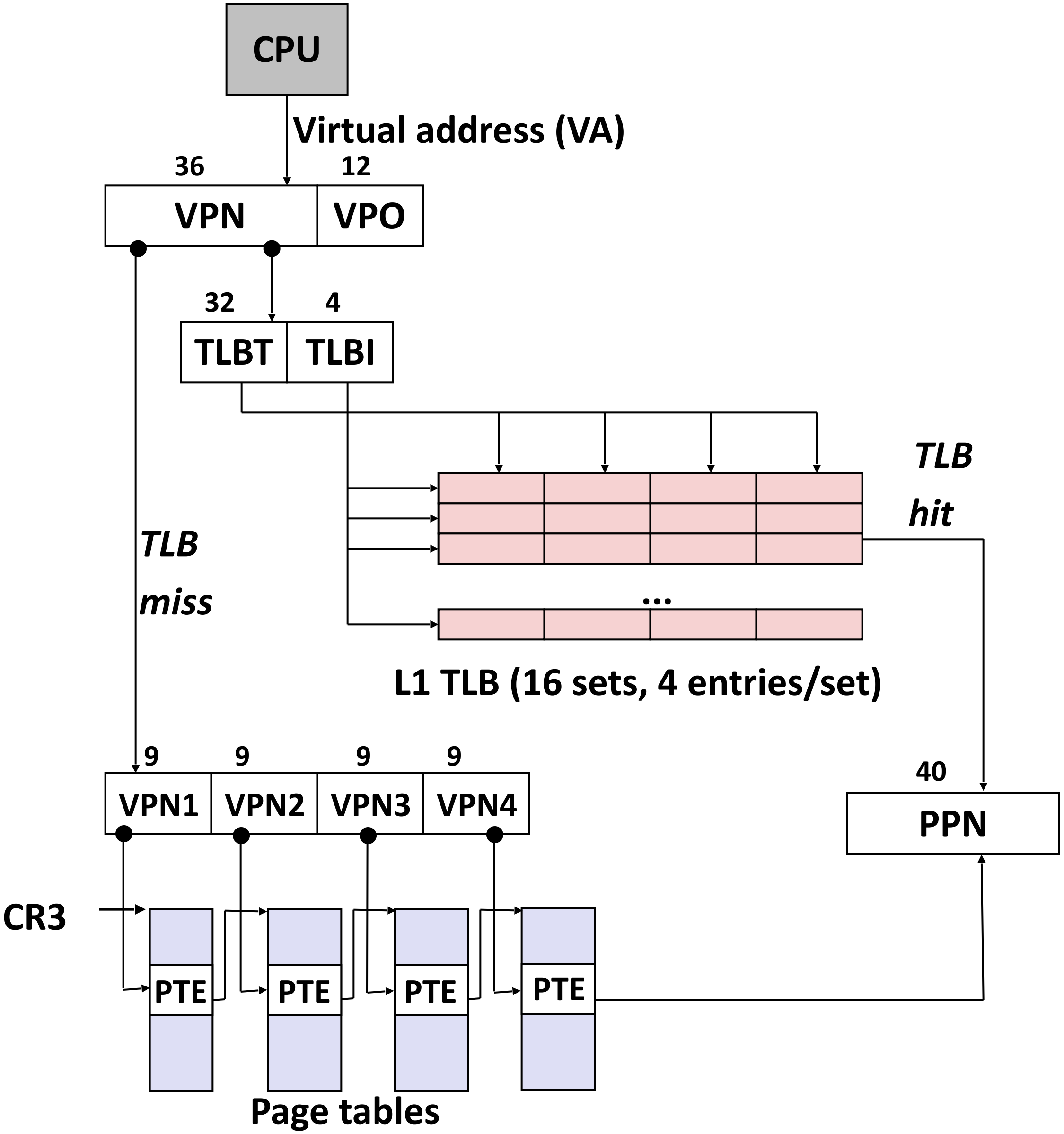


L1 TLB (16 sets, 4 entries/set)

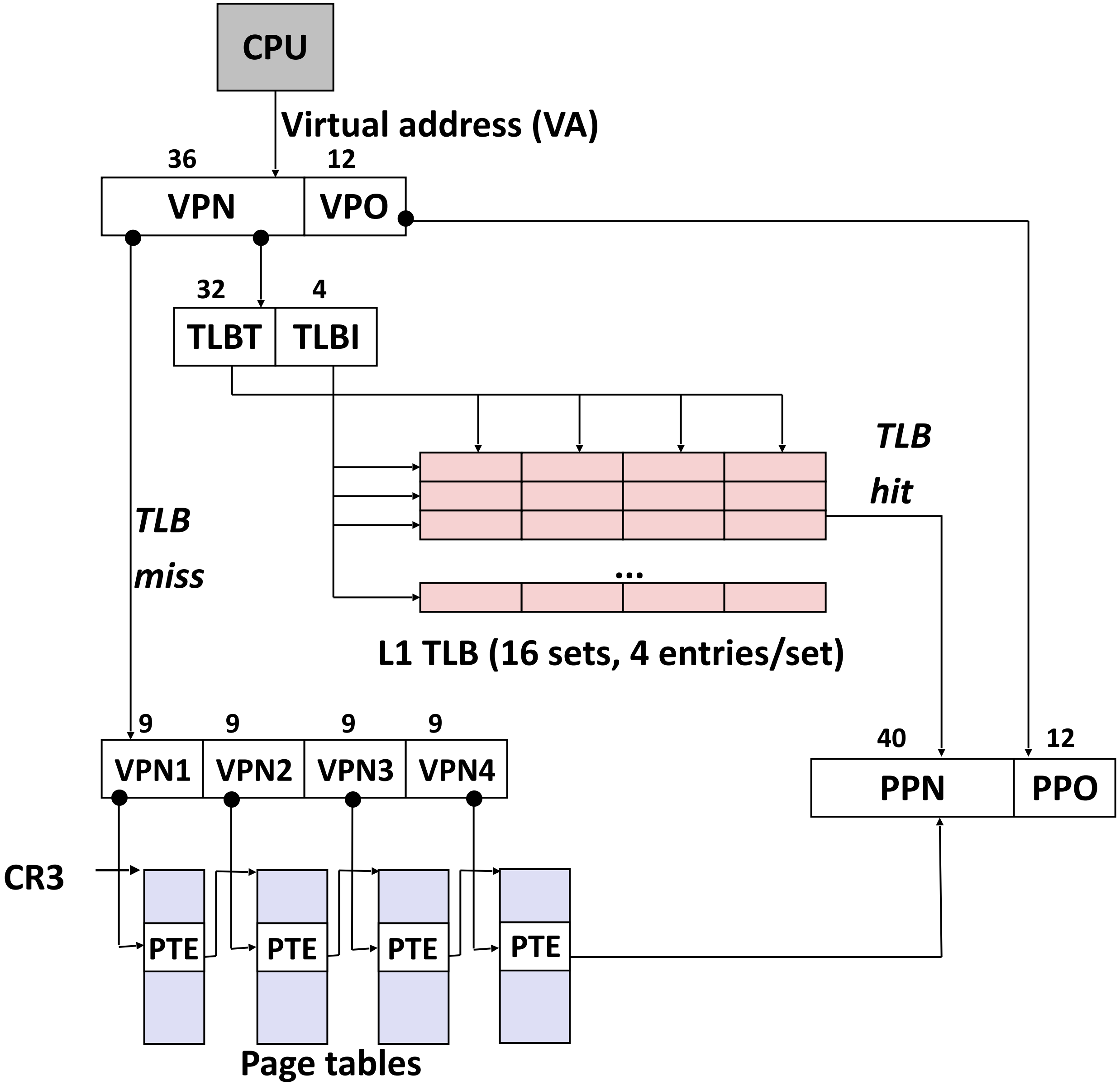
End-to-End Core i7 Address Translation



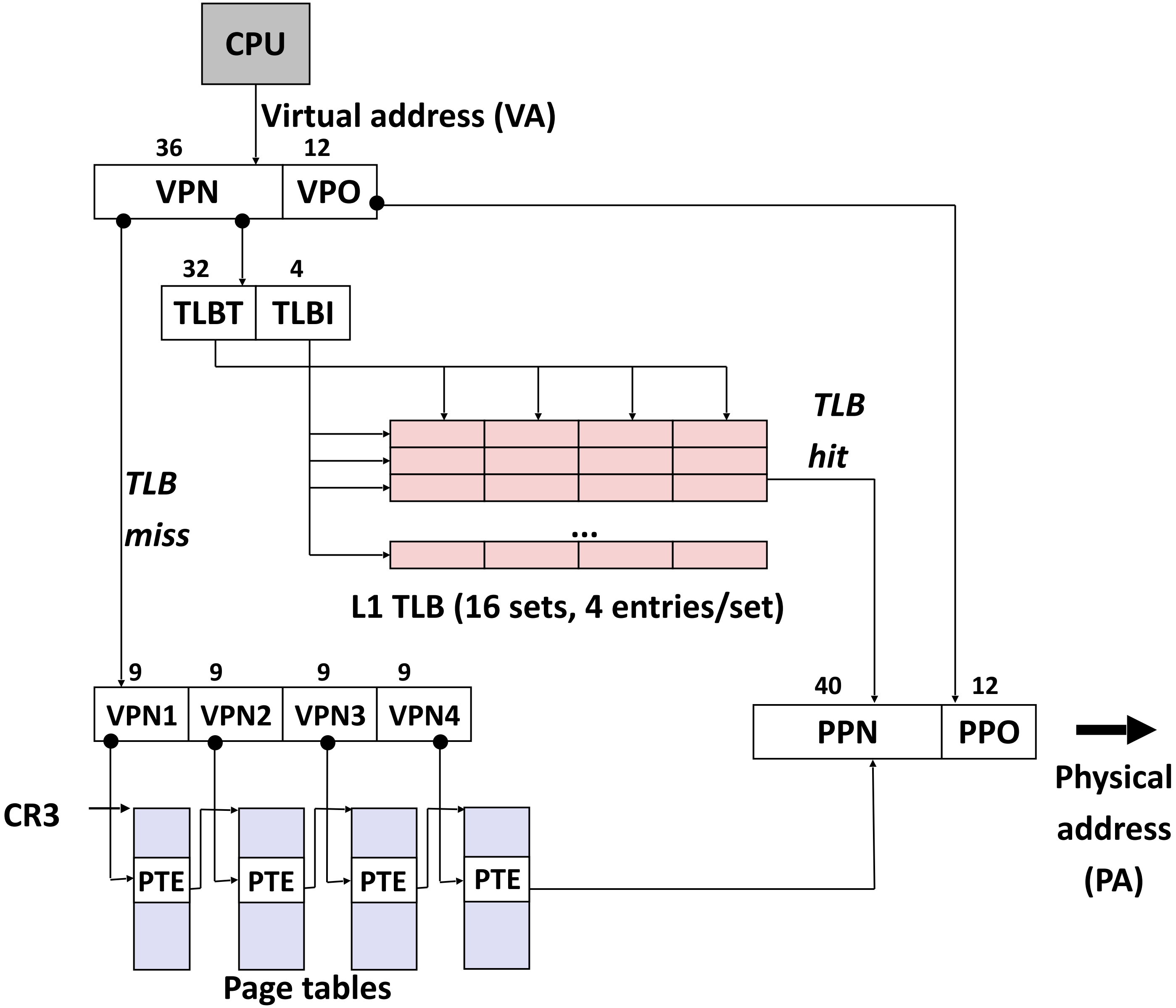
End-to-End Core i7 Address Translation



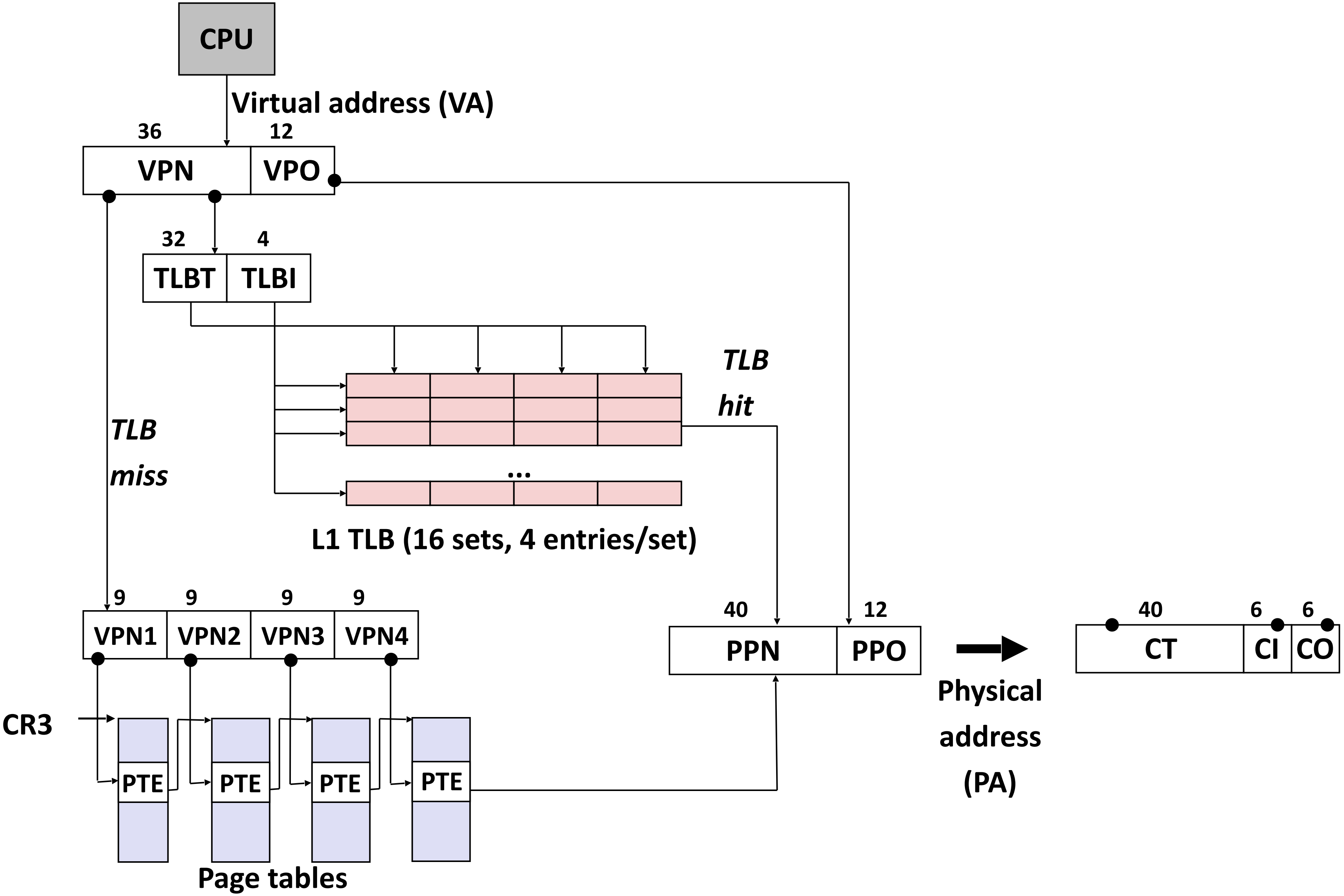
End-to-End Core i7 Address Translation



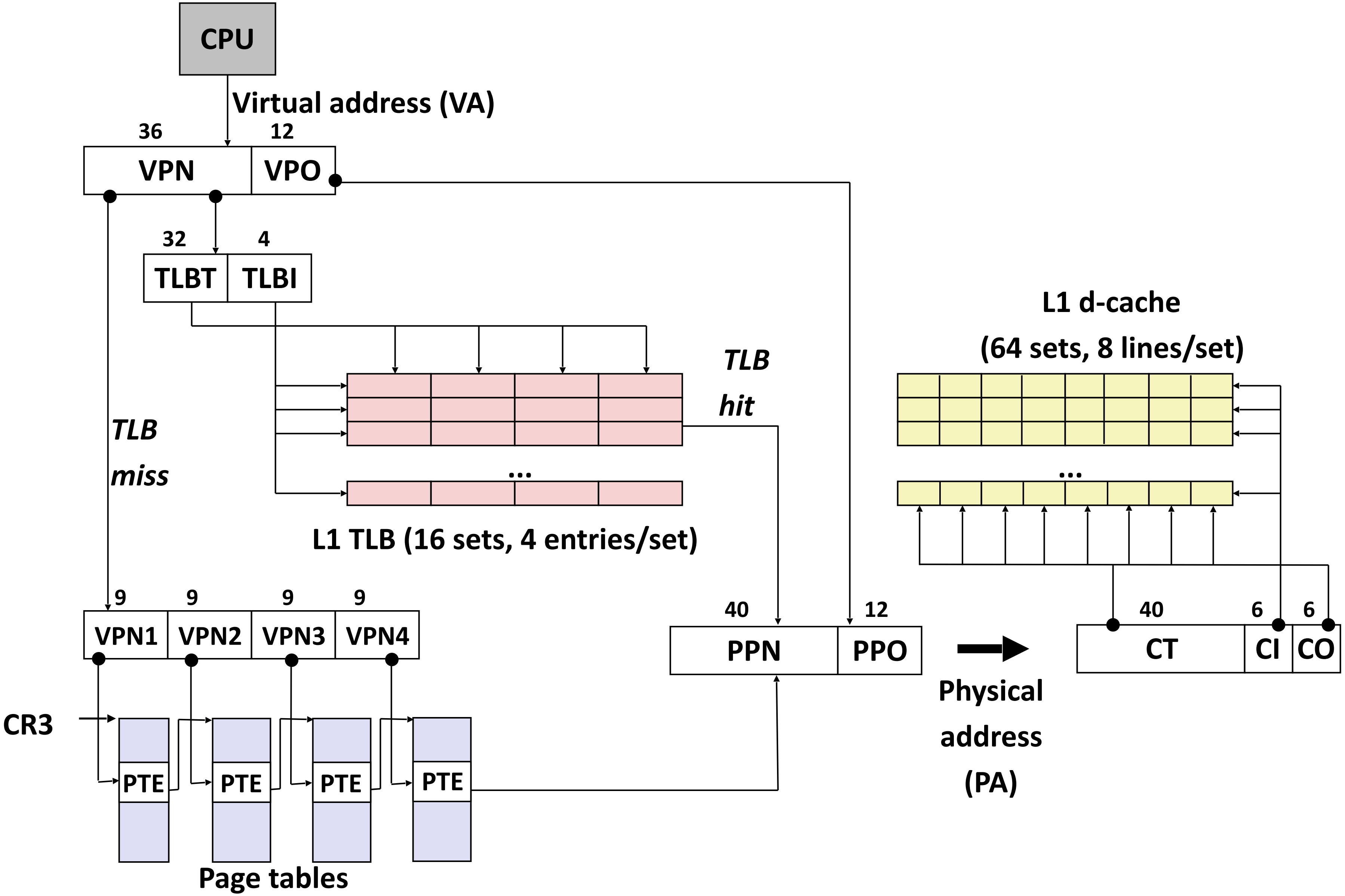
End-to-End Core i7 Address Translation



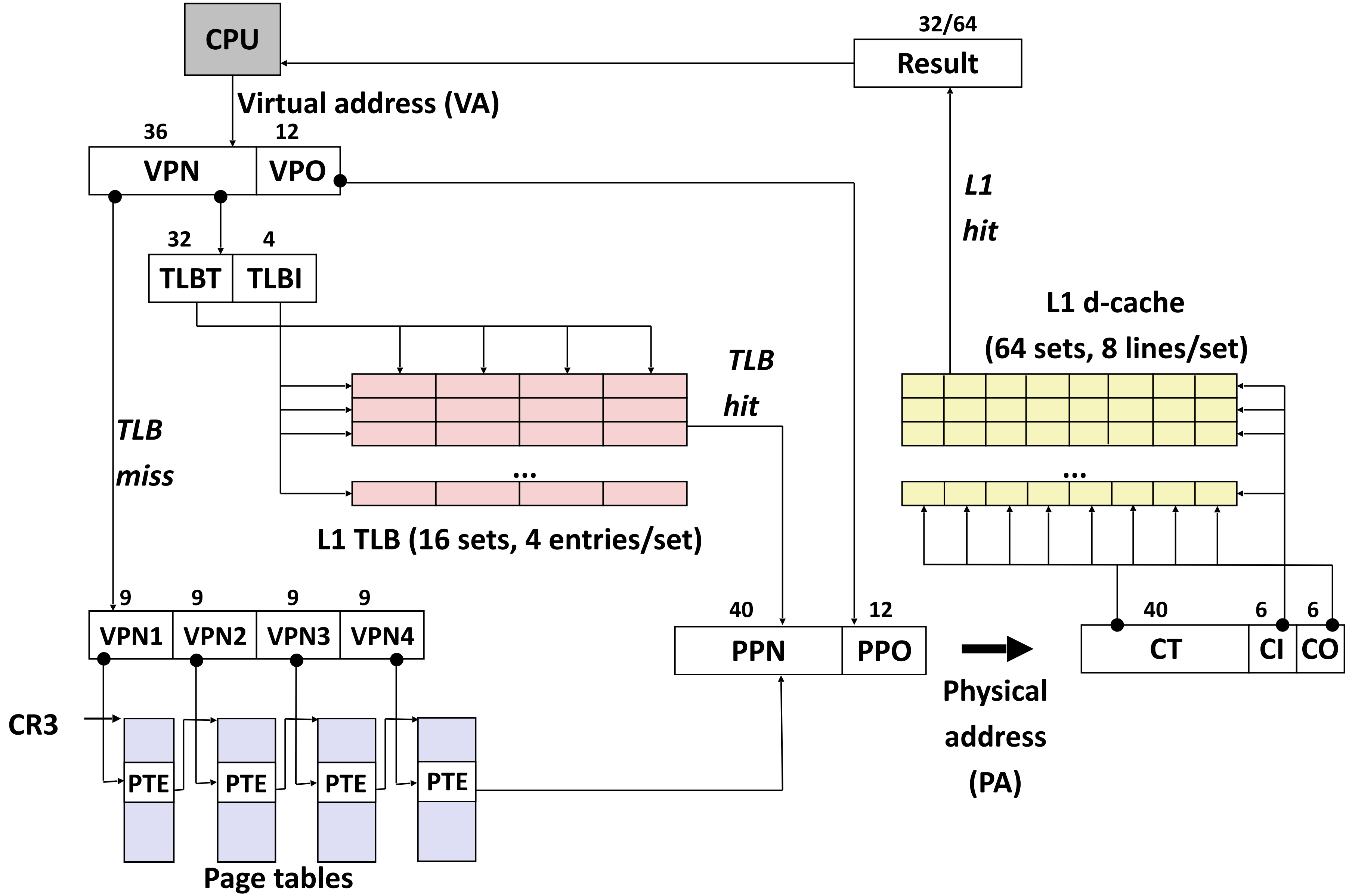
End-to-End Core i7 Address Translation



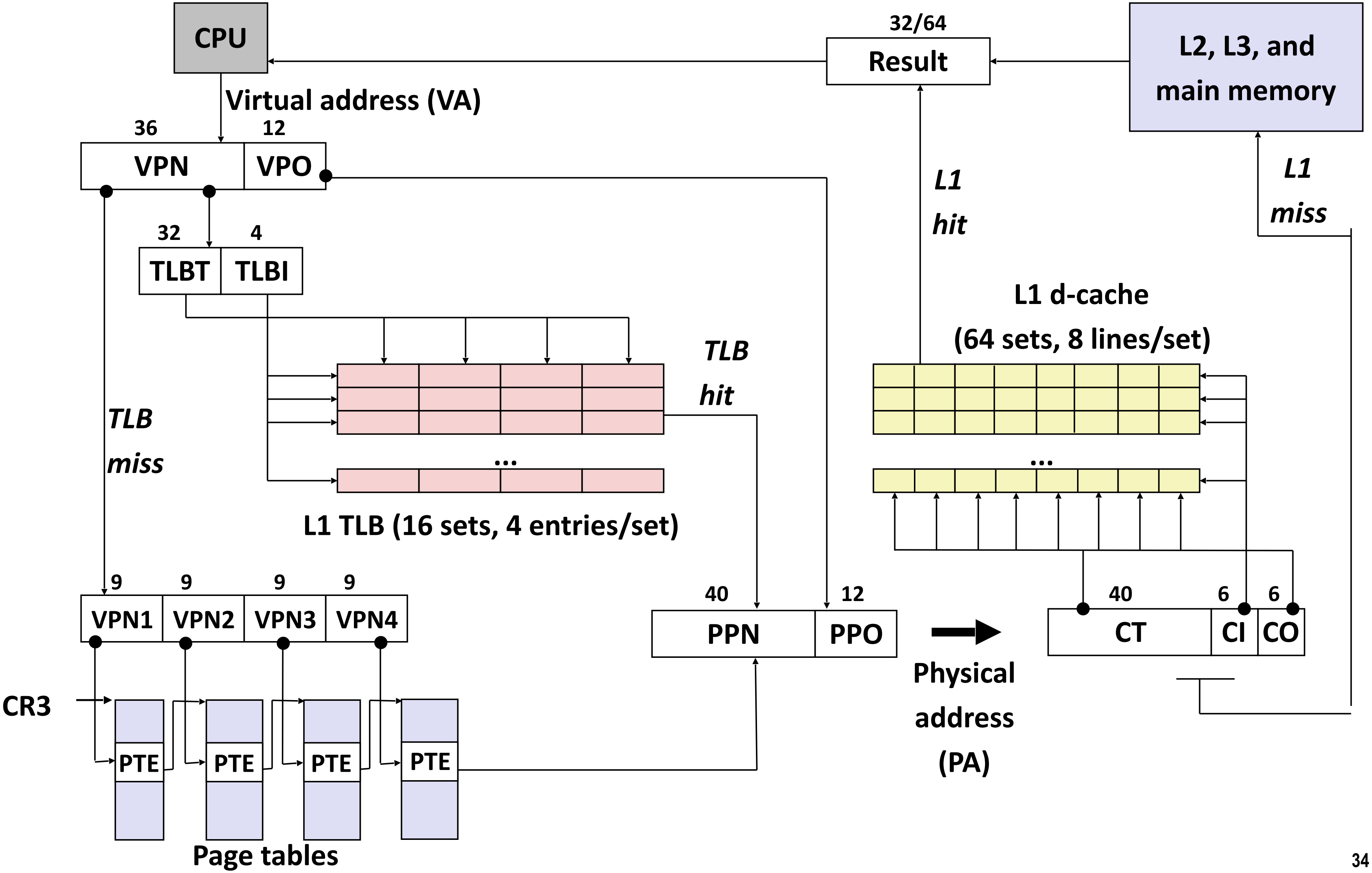
End-to-End Core i7 Address Translation



End-to-End Core i7 Address Translation



End-to-End Core i7 Address 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 (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 child 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 (page table location on disk)	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.

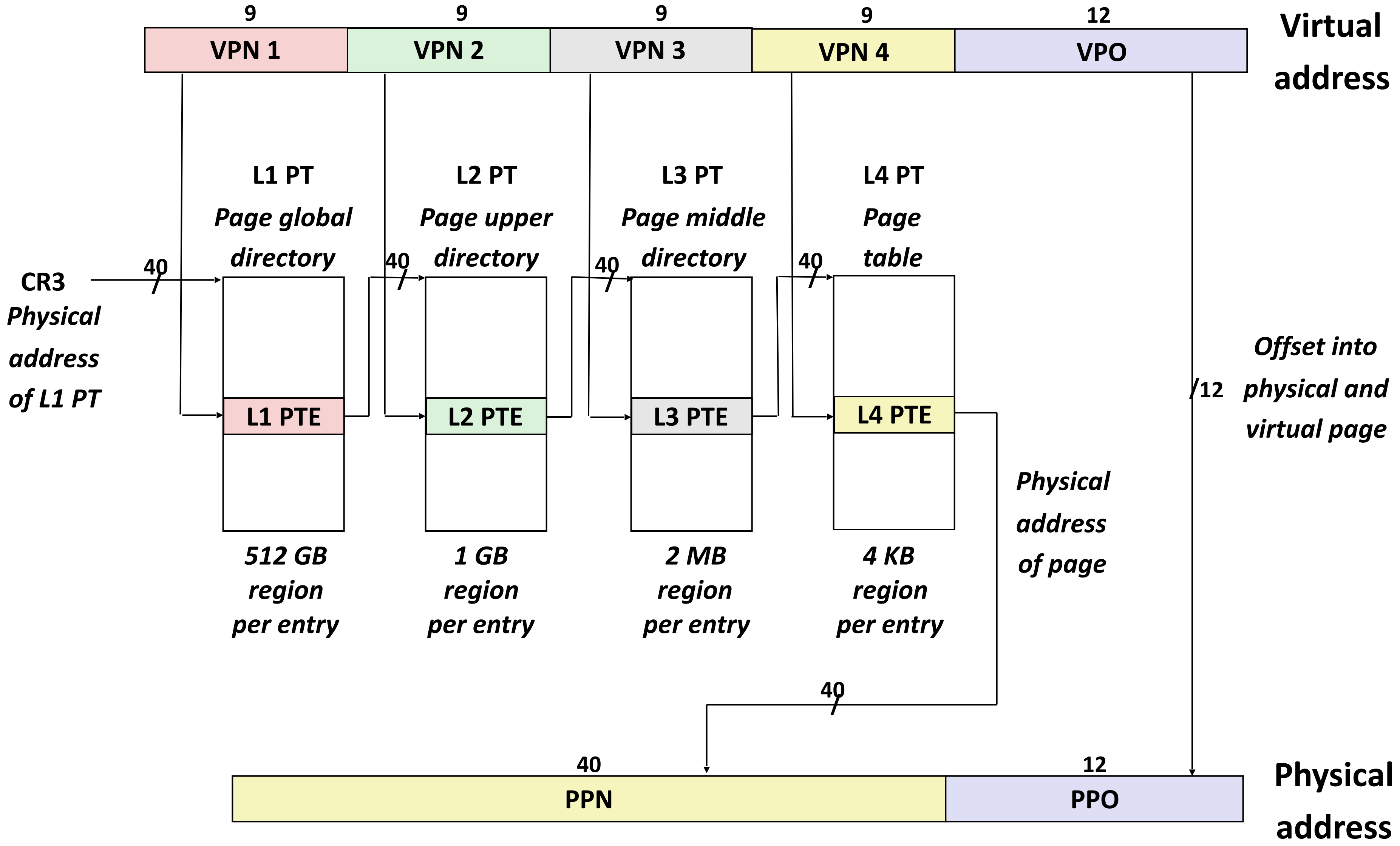
A: Reference bit (set by MMU on reads and writes, cleared by software).

PS: Page size either 4 KB or 4 MB (defined for Level 1 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 Page Table Translation


9
9
9
9
12
VPN 1
VPN 2
VPN 3
VPN 4
VPO
Virtual address
L1 PT
Page global directory
L2 PT
Page upper directory
L3 PT
Page middle directory
L4 PT
Page table
CR3
Physical address of L1 PT
40
40
40
40
L1 PTE
L2 PTE
L3 PTE
L4 PTE
Offset into physical and virtual page
12
512 GB region per entry
1 GB region per entry
2 MB region per entry
4 KB region per entry
Physical address of page
40
40
12
PPN
PPO
Physical address