Virtual Memory II

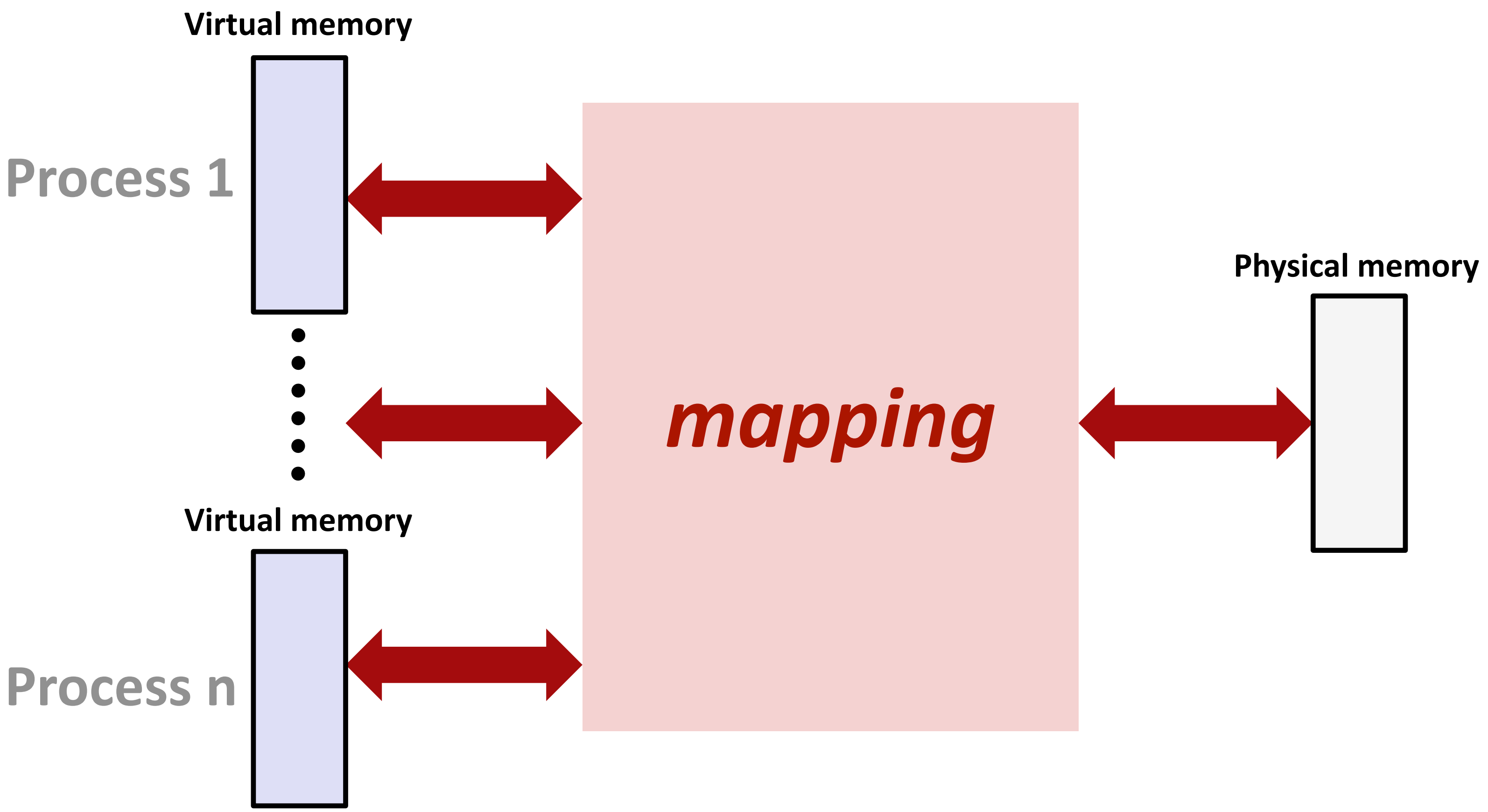
15-213/18-243: Introduction to Computer Systems

16th Lecture, 18 March 2010

Instructors:

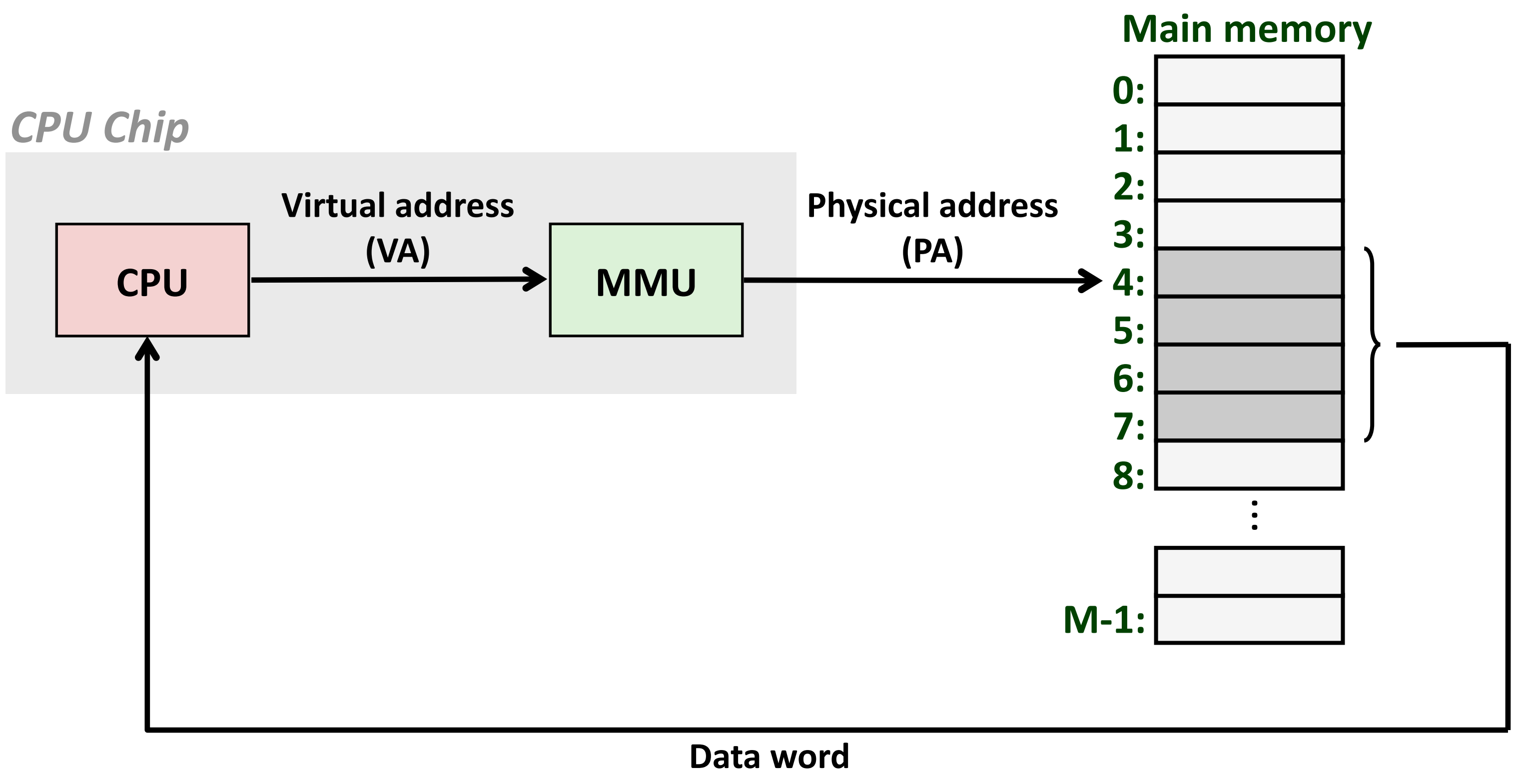
Bill Nace and Gregory Kesden

Last Time: Virtual Memory



- Each process gets its own private memory space
- Mapping function is general: 1 ↔ 1, n ↔ 1, 1 ↔ m

A System Using Virtual Addressing



- Used in all modern desktops, laptops, workstations
- One of the great ideas in computer science
- *MMU checks the cache*

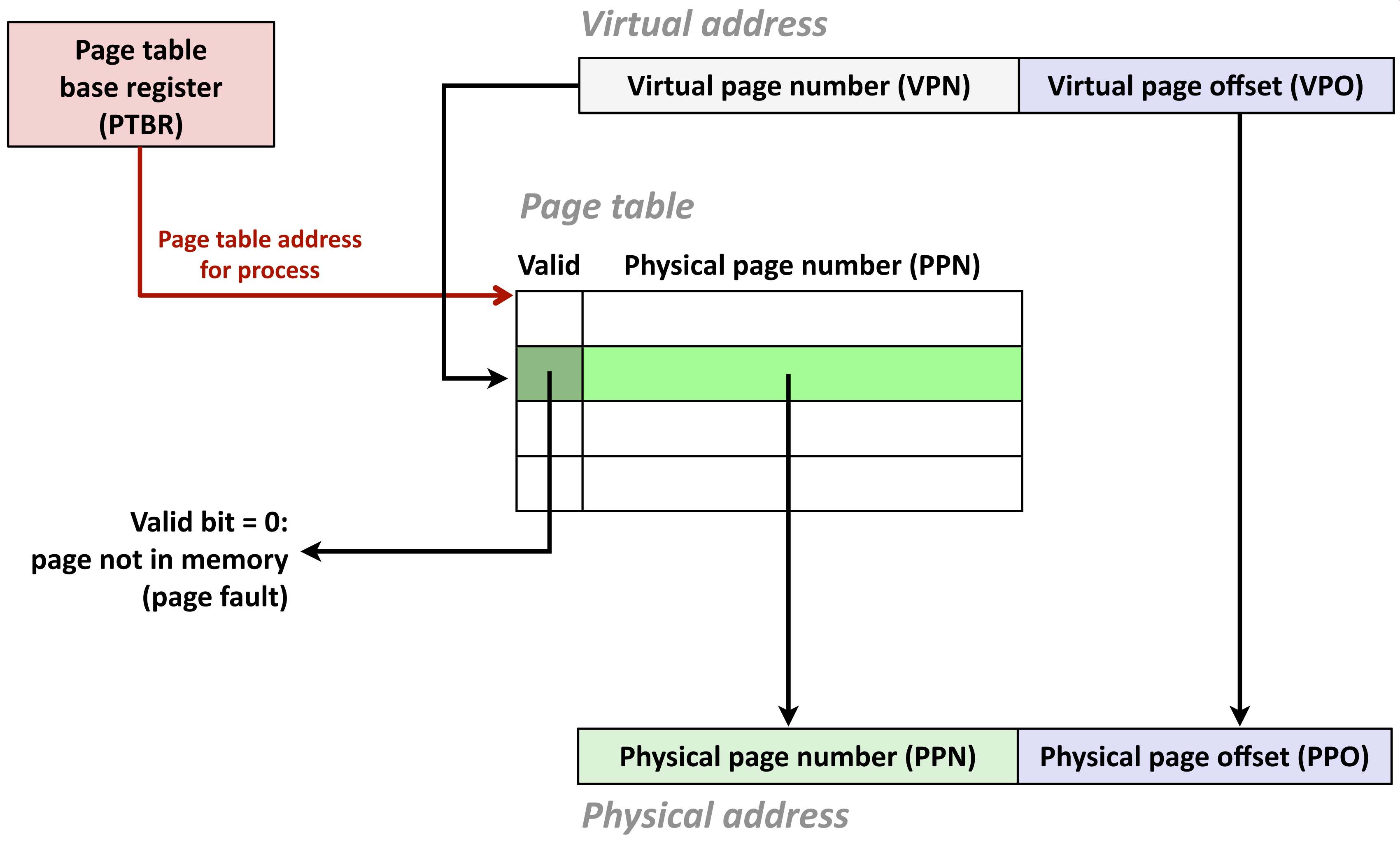
VM Solves Many Problems

- **VM as a tool for caching**
  - Dodges huge miss penalty associated with disks
  - Disk latency is millions of cycles (3 GHz / 5mSec = 600 million cycles)
- **VM as a tool for memory management**
  - Each virtual page can be mapped to any physical page
  - Different pages at different times
  - Vastly simplifies Linking and Loading
- **VM as a tool for memory protection**
  - Add a few access bits to the page table (SUP, READ, WRITE)
  - Now, processes are incapable of overwriting each other's memory
  - User processes can't access kernel data structures/code/etc

Today

- Address Translation
- Multi-level page tables
- Intel Core i7 Address Translation

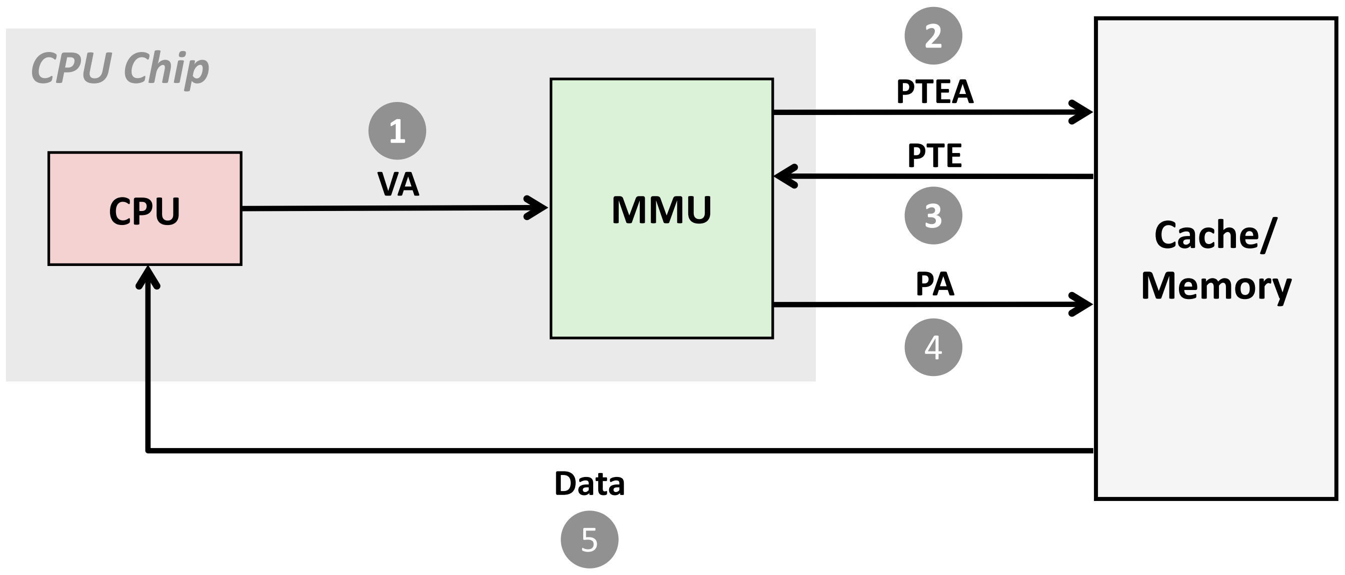
Address Translation With a Page Table


Virtual address
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)
Physical page number (PPN)
Physical page offset (PPO)
Physical address

Concrete Examples

- **VAX 11/780 (from 1978)**
  - 32-bit virtual address, 32-bit physical address, page size of 512 bytes
  - How big is VPN / VPO?
  - PPN? PPO?
- **Alpha 21264 (~2002)**
  - 43-bit virtual address, 41-bit physical address, page size of 8KB
  - How big is VPN / VPO?
  - PPN? PPO?
- **Intel Core i7 (2008)**
  - 48-bit virtual address, 52-bit physical address
  - If page size is 4MB, how many PTEs in page table? How big is PPN?

Address Translation: Page Hit



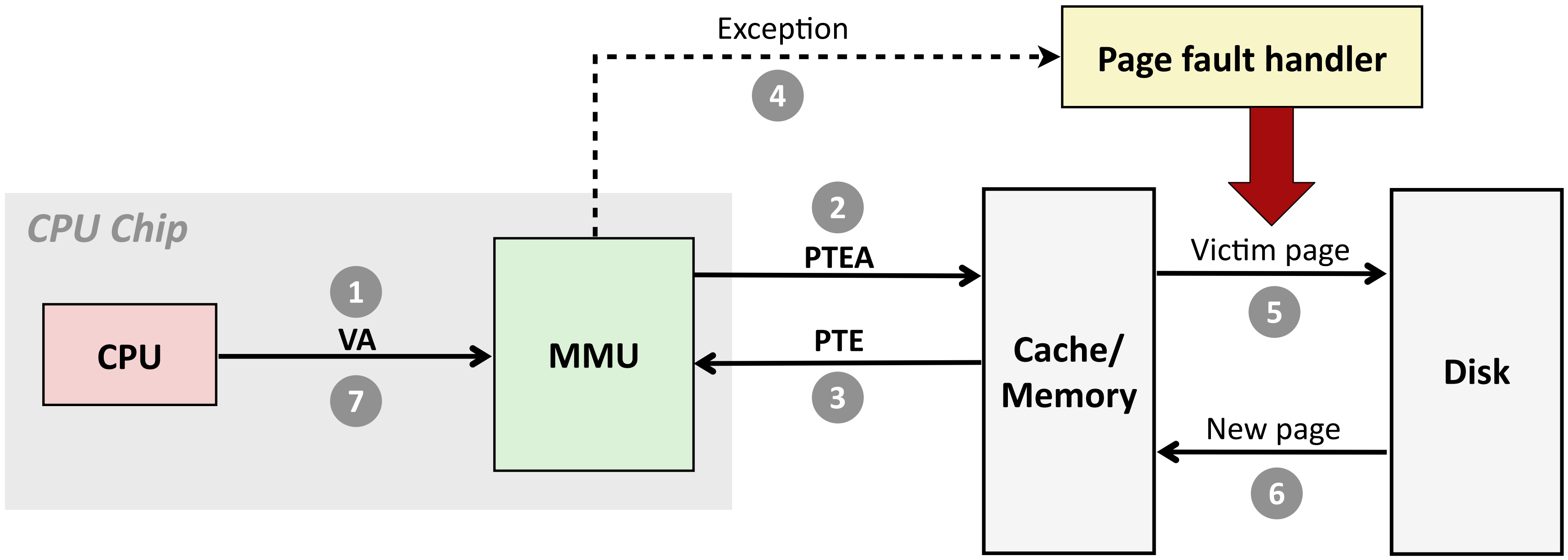
1) Processor sends virtual address to MMU

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

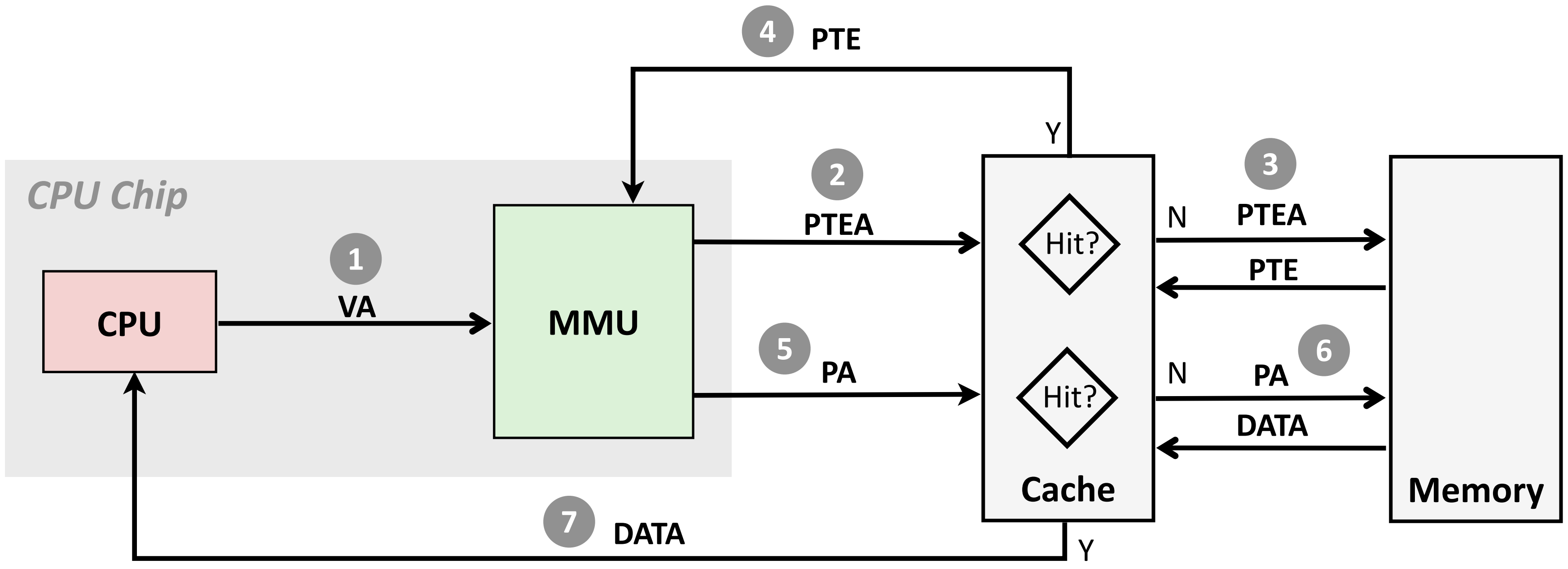
4) Valid bit is zero, so MMU triggers page fault exception

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

Aside: Integrating a Cache



1) Processor sends virtual address to MMU

2) MMU fetches PTE from page table in memory via cache

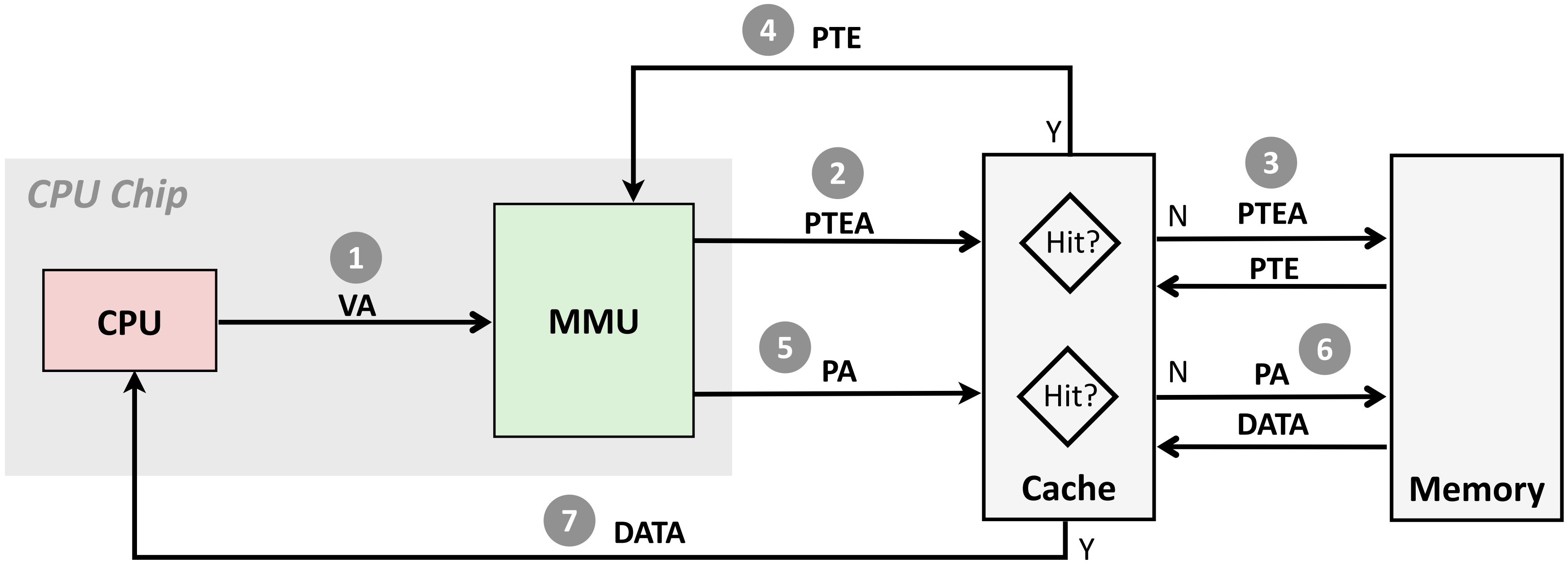
3) If PTE not in cache, it is fetched from memory

4) PTE returned to MMU

5) MMU sends physical address to cache

6) If physical address not in cache, it is fetched from memory

Aside: Integrating a Cache (2)

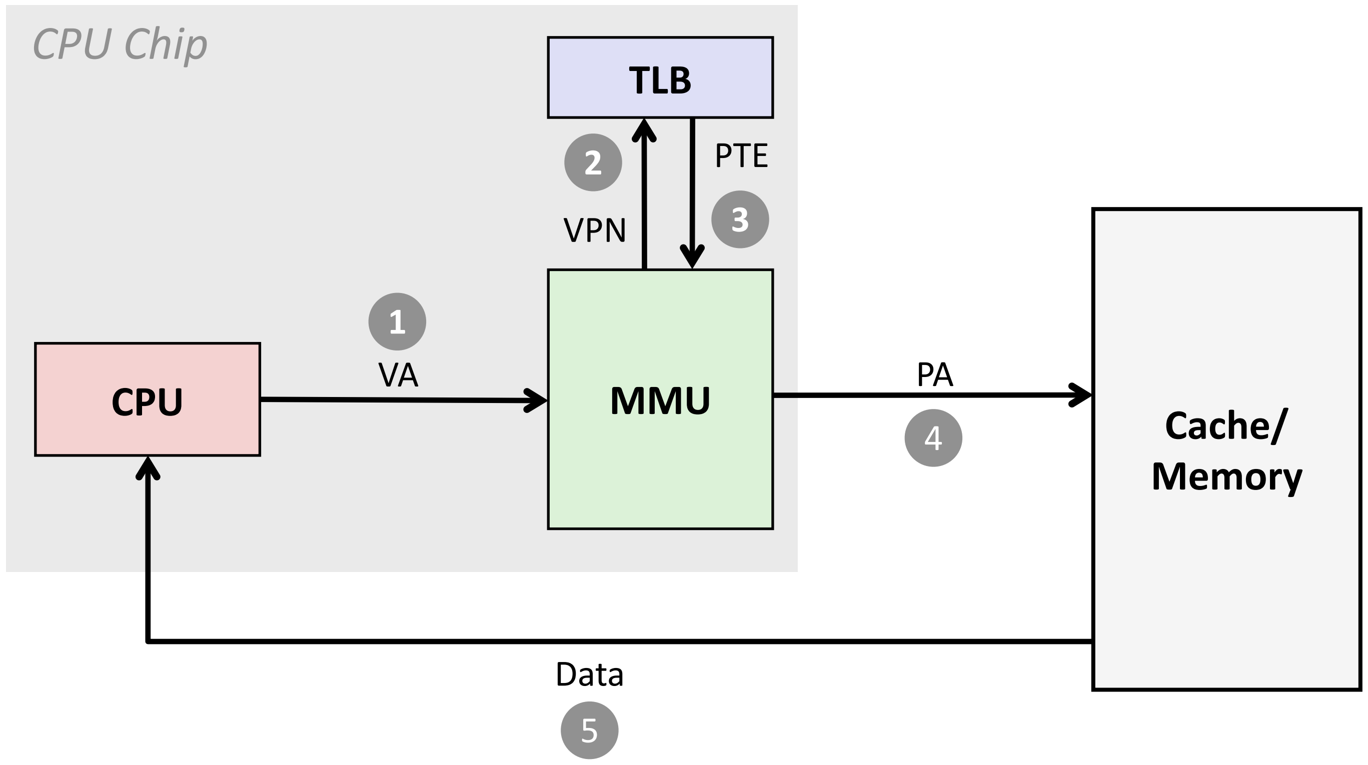


- **Cache usually designed to use physical addresses**
  - Blocks from multiple processes may have the same virtual address
  - Shared page (i.e. same virtual address) will still have same physical address
  - Simplifies protection issues → access rights already checked in MMU

Speeding up Translation with a TLB

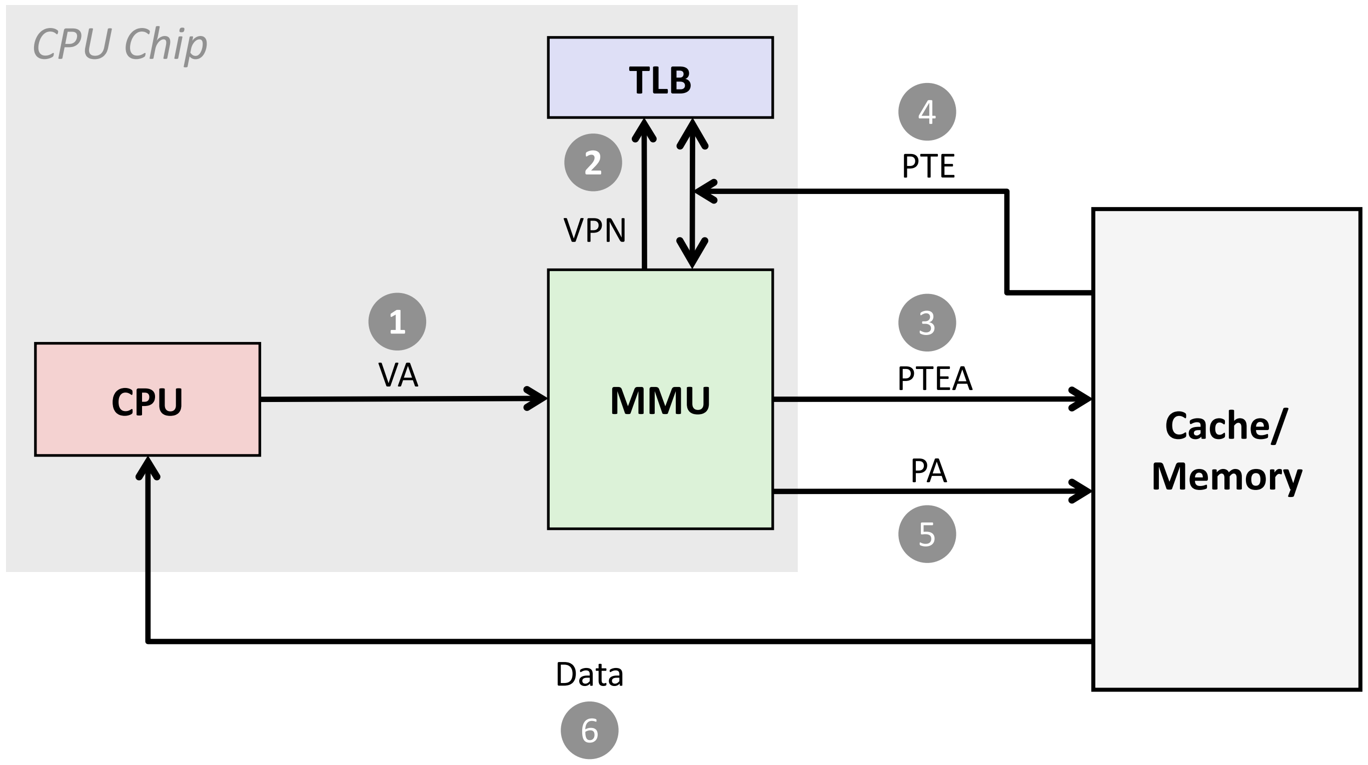
- **Page table entries (PTEs) are cached in L1 like any other memory word**
  - PTEs may be evicted by other data references
  - PTE hit still requires a 1-cycle delay
- **Solution: *Translation Lookaside Buffer* (TLB)**
  - Small hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages

TLB Hit



- **A TLB hit eliminates a memory access**

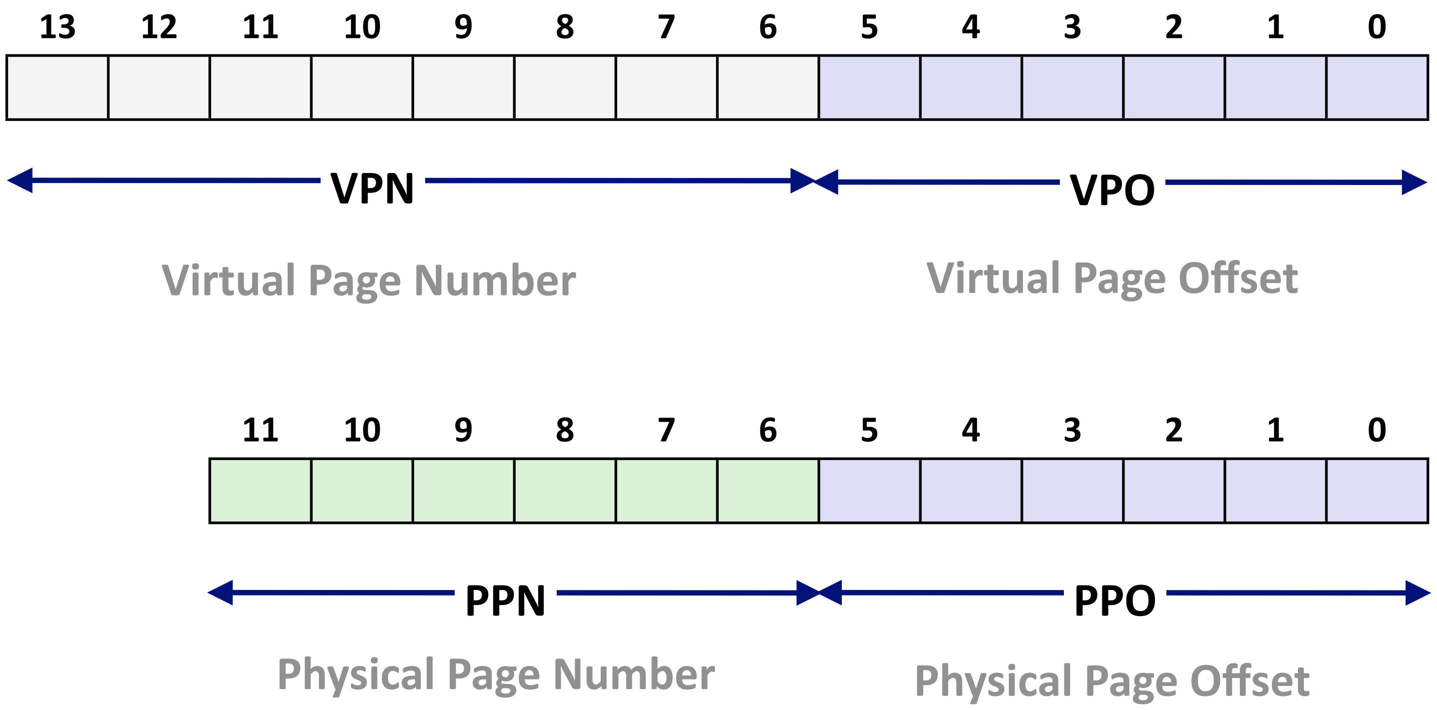
TLB Miss



- **A TLB miss incurs an add’l memory access (the PTE)**
  - Fortunately, TLB misses are rare
  - The PTE may be in cache even then, so 1-2 cycle access

Simple Memory System Example

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



Example: Page Table

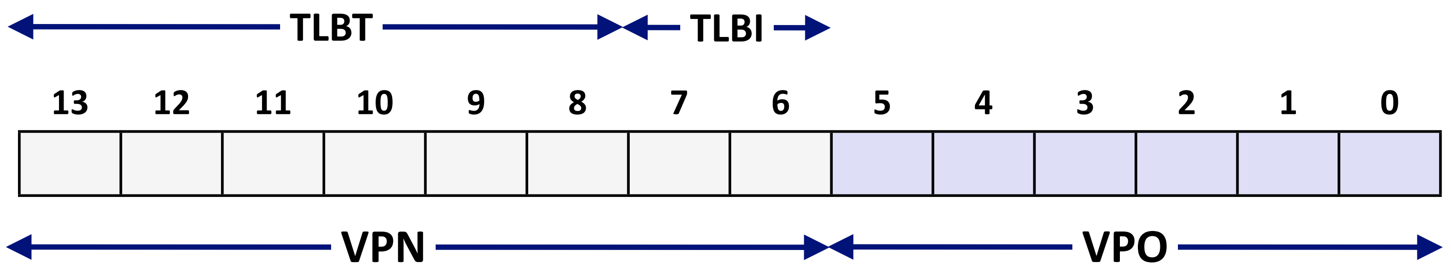
- Only first 16 PTEs are shown (out of 256)

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

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

Example: TLB

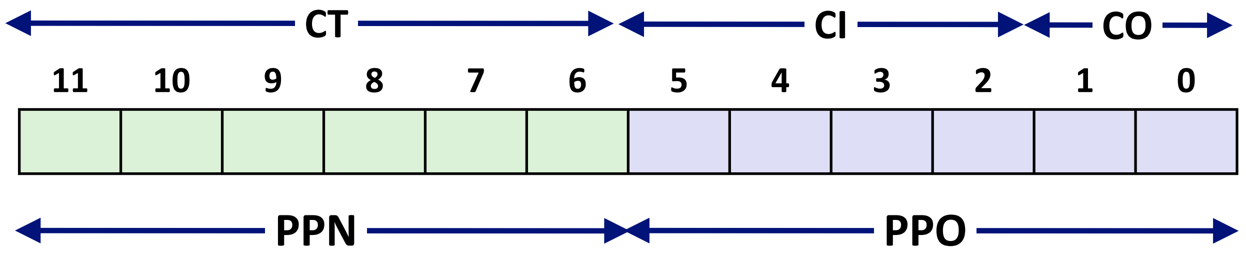
- 16 entries
- 4-way set associative (TLBT = TLB Tag, TLBI = TLB Index)



Set	Tag	Valid	PPN	Tag	Valid	PPN	Tag	Valid	PPN	Tag	Valid	PPN
0	03	0	--	09	1	0D	00	0	--	07	1	02
1	03	1	2D	02	0	--	04	0	--	0A	0	--
2	02	0	--	08	0	--	06	0	--	03	0	--
3	07	0	--	03	1	0D	0A	1	34	02	0	--

Example: Cache

- 16 lines, 4-byte block size
- Physically addressed
- Direct mapped (CT = Cache Tag, CI = Set Index, CO = Block Offset)

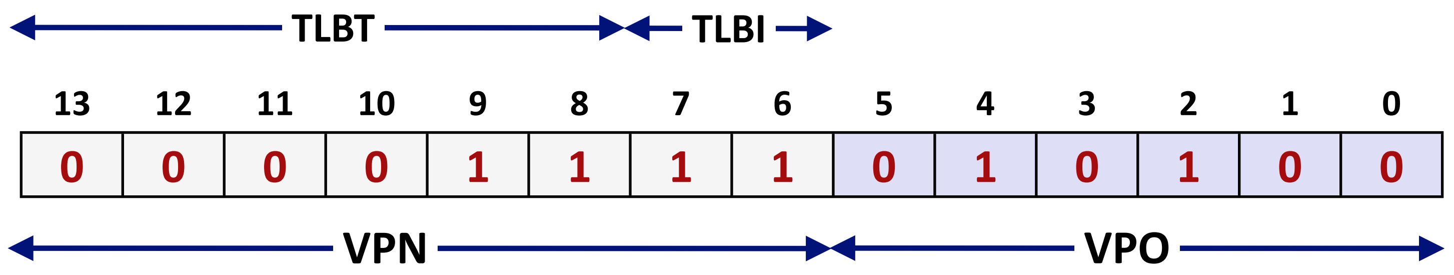


Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11
1	15	0	--	--	--	--
2	1B	1	00	02	04	08
3	36	0	--	--	--	--
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	--	--	--	--
7	16	1	11	C2	DF	03

Idx	Tag	Valid	B0	B1	B2	B3
8	24	1	3A	00	51	89
9	2D	0	--	--	--	--
A	2D	1	93	15	DA	3B
B	0B	0	--	--	--	--
C	12	0	--	--	--	--
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	--	--	--	--

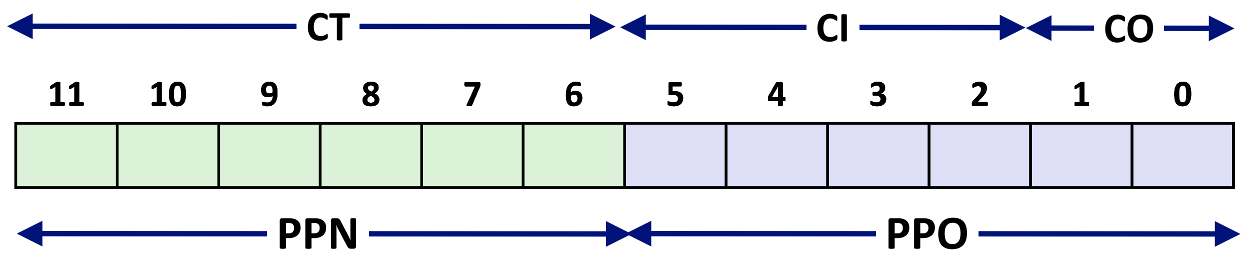
Address Translation Example #1

Virtual Address: **`0x03D4`**



VPN: TLBI: TLBT: TLB Hit? Page Fault? PPN:

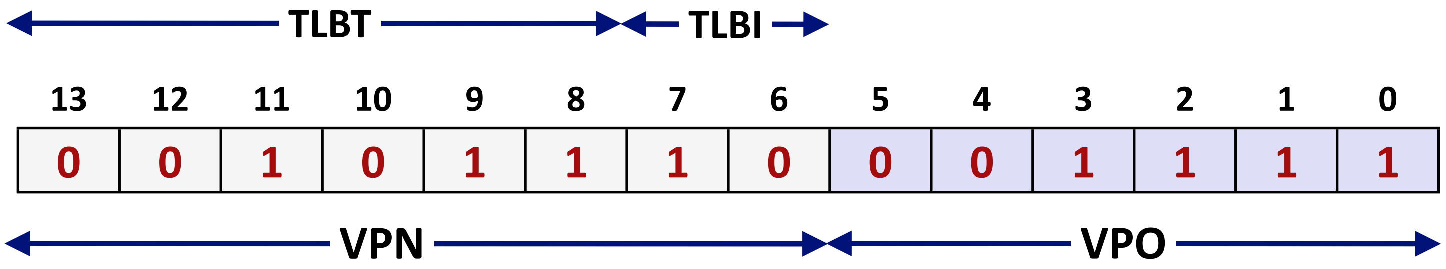
Physical Address



CO: CI: CT: Cache Hit? Data:

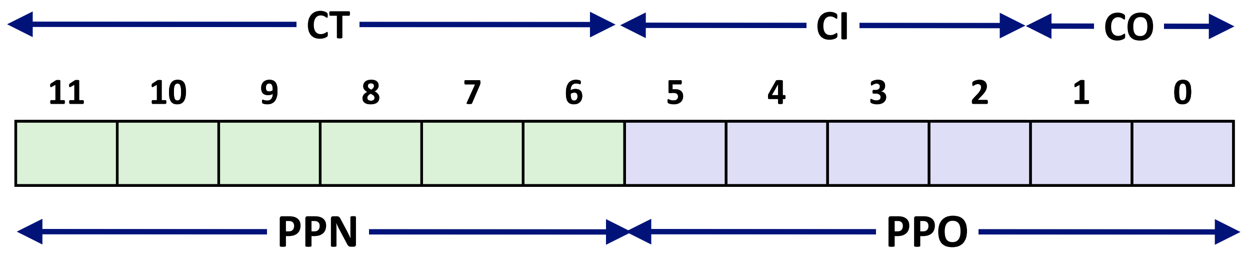
Address Translation Example #2

Virtual Address: **`0x0B8F`**



VPN: ☐ TLBI: ☐ TLBT: ☐ TLB Hit? ☐ Page Fault? ☐ PPN: ☐

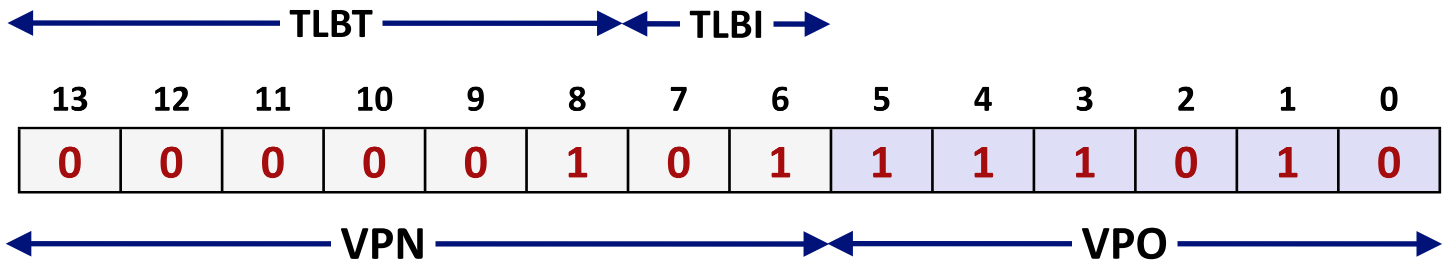
Physical Address



CO: ☐ CI: ☐ CT: ☐ Cache Hit? ☐ Data: ☐

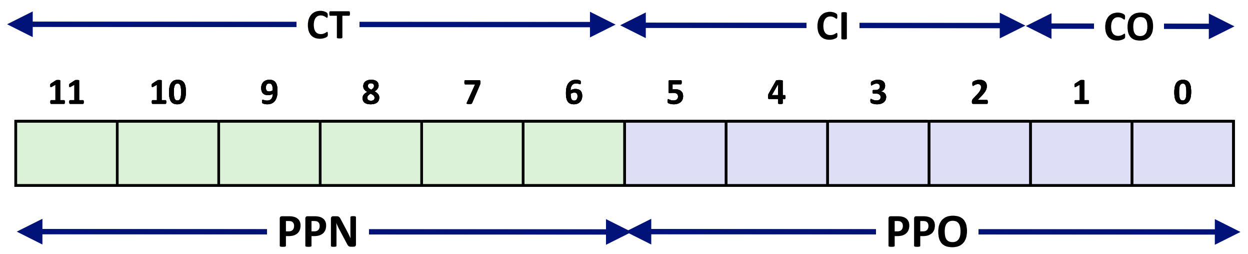
Address Translation Example #3

Virtual Address: **`0x017A`**



VPN: TLBI: TLBT: TLB Hit? Page Fault? PPN:

Physical Address



CO: CI: CT: Cache Hit? Data:

Summary

- **Programmer's view of virtual memory**
  - Each process has its own private linear address space
  - Cannot be corrupted by other processes

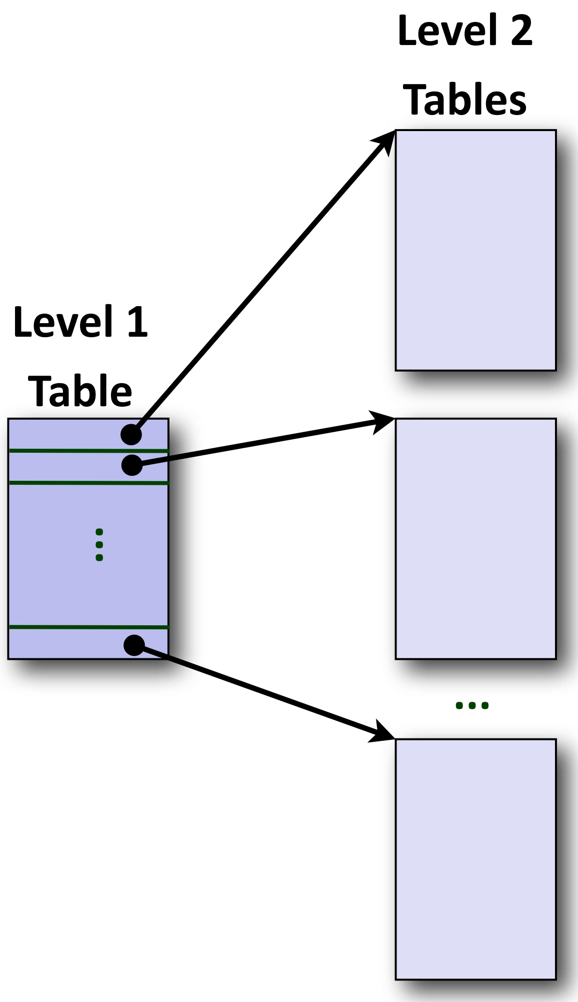
- **System view of virtual memory**
  - Uses memory efficiently by caching virtual memory pages
    - Efficient only because of locality
  - Simplifies memory management and programming
  - Simplifies protection by providing a convenient interpositioning point to check permissions

Today

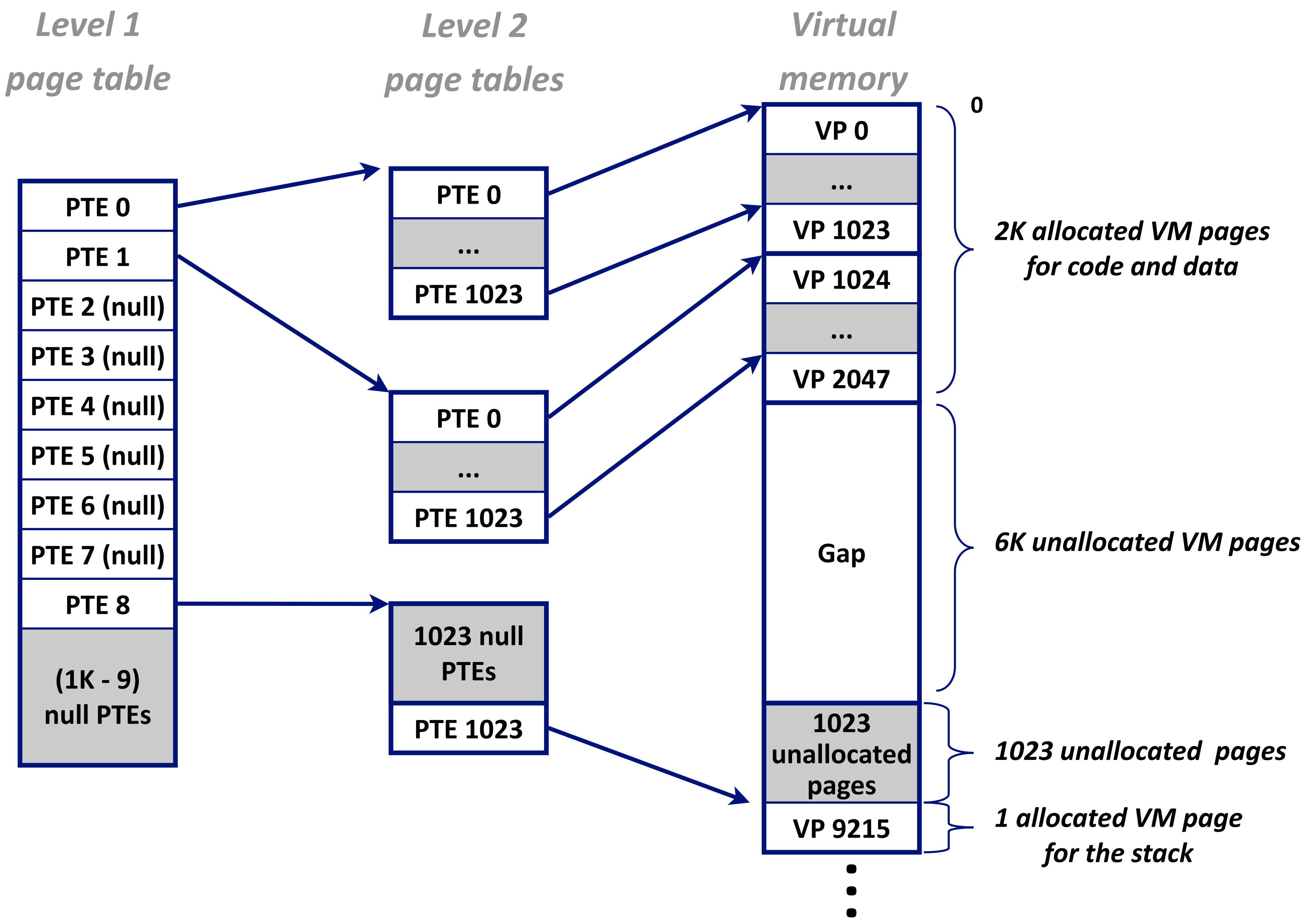
- Address Translation
- **Multi-level page tables**
- Intel Core i7 Address Translation

Multi-Level Page Tables

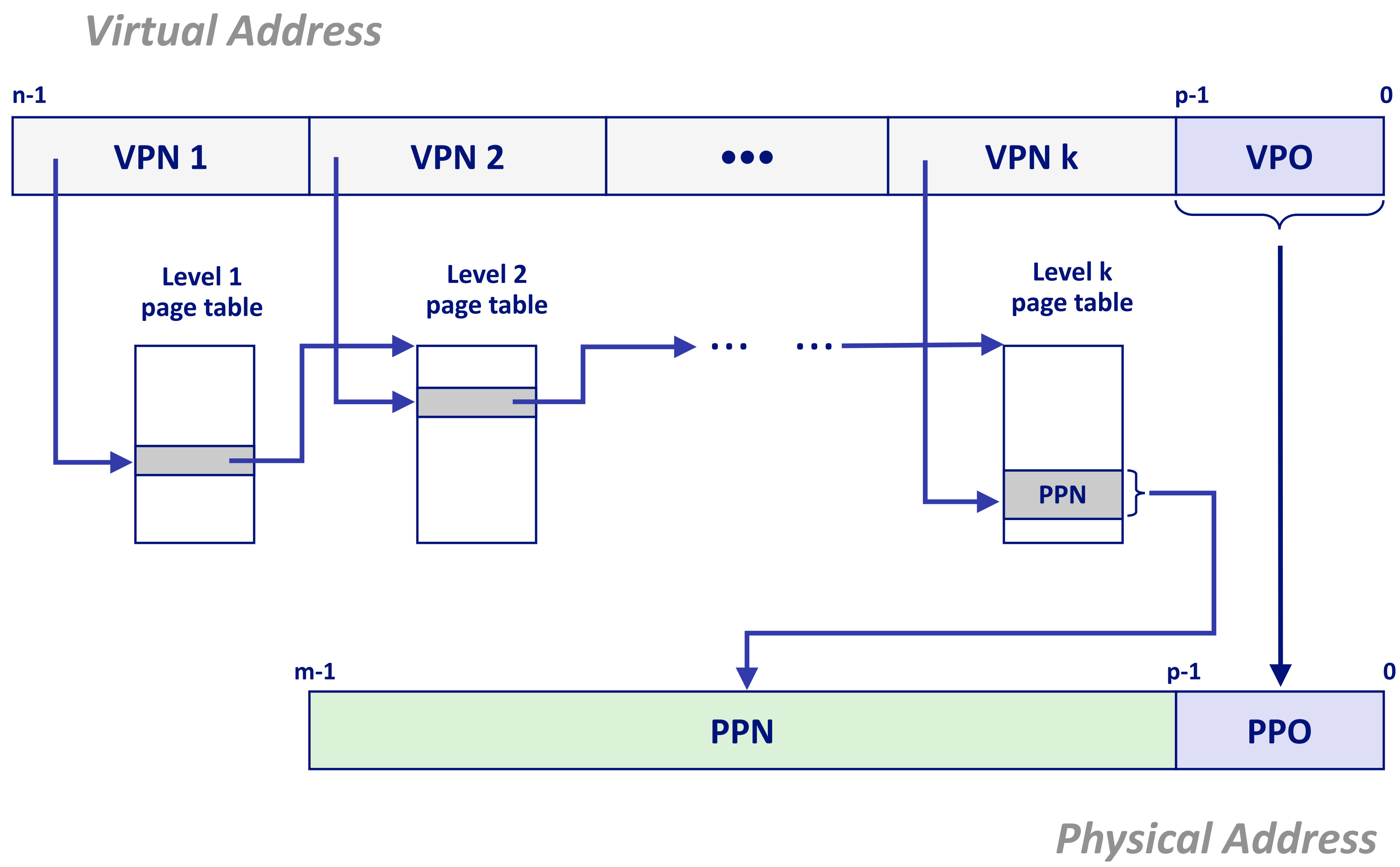
- **Intel Core i7:**
  - 4KB (2^{12}) page size
  - 48-bit address space
  - 4-byte PTE
- **Page table size = 256 GB! Ooops!**
  - $2^{48} * 2^{-12} * 2^{2} = 2^{38}$ bytes
- **Common solution**
  - Multi-level page tables
  - Example: 2-level page table
  - Level 1 table: each PTE points to a page table
  - Level 2 table: each PTE points to a page (paged in and out like other data)
  - Level 1 table stays in memory
  - Level 2 tables paged in and out



A Two-Level Page Table Hierarchy



Translating with a k-level Page Table

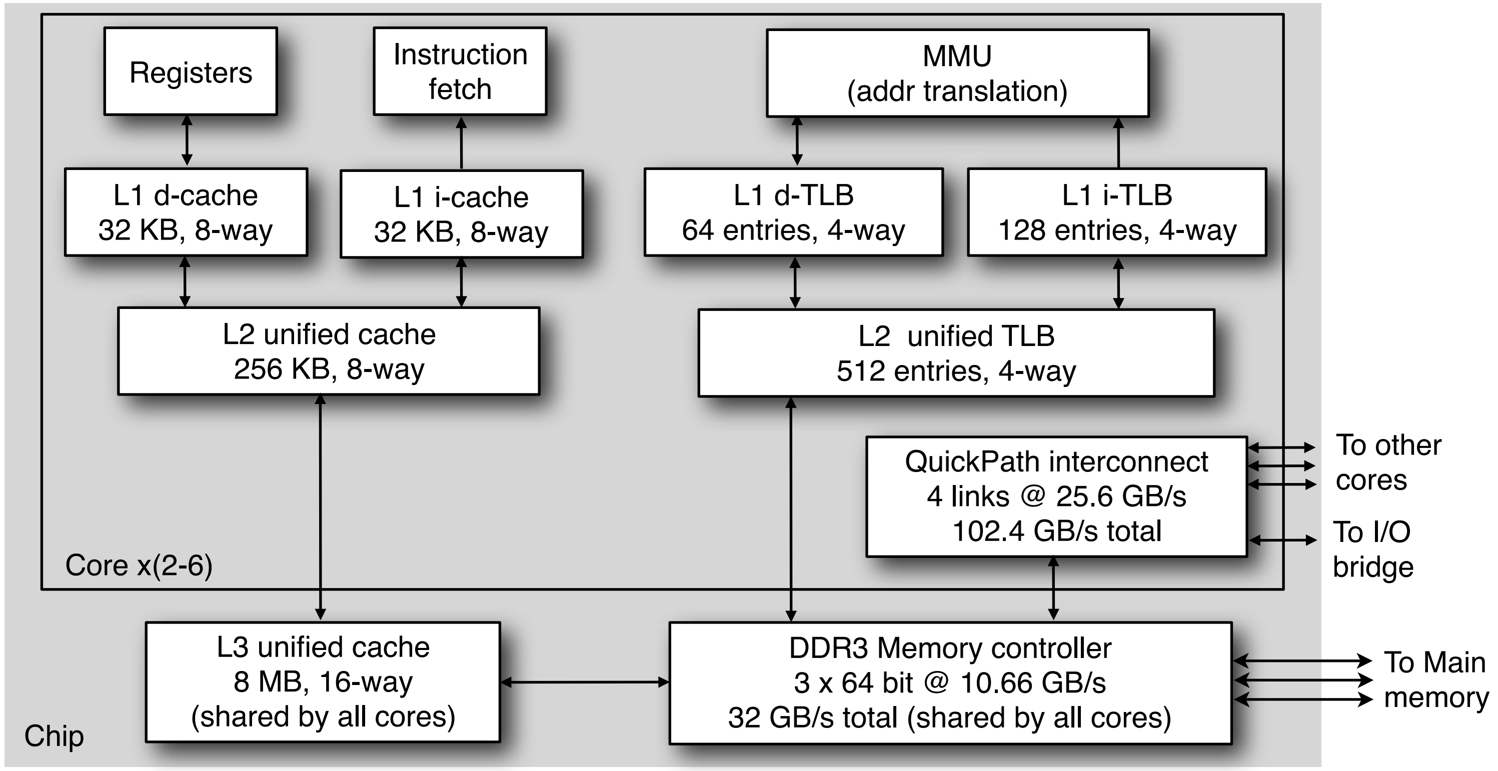


Today

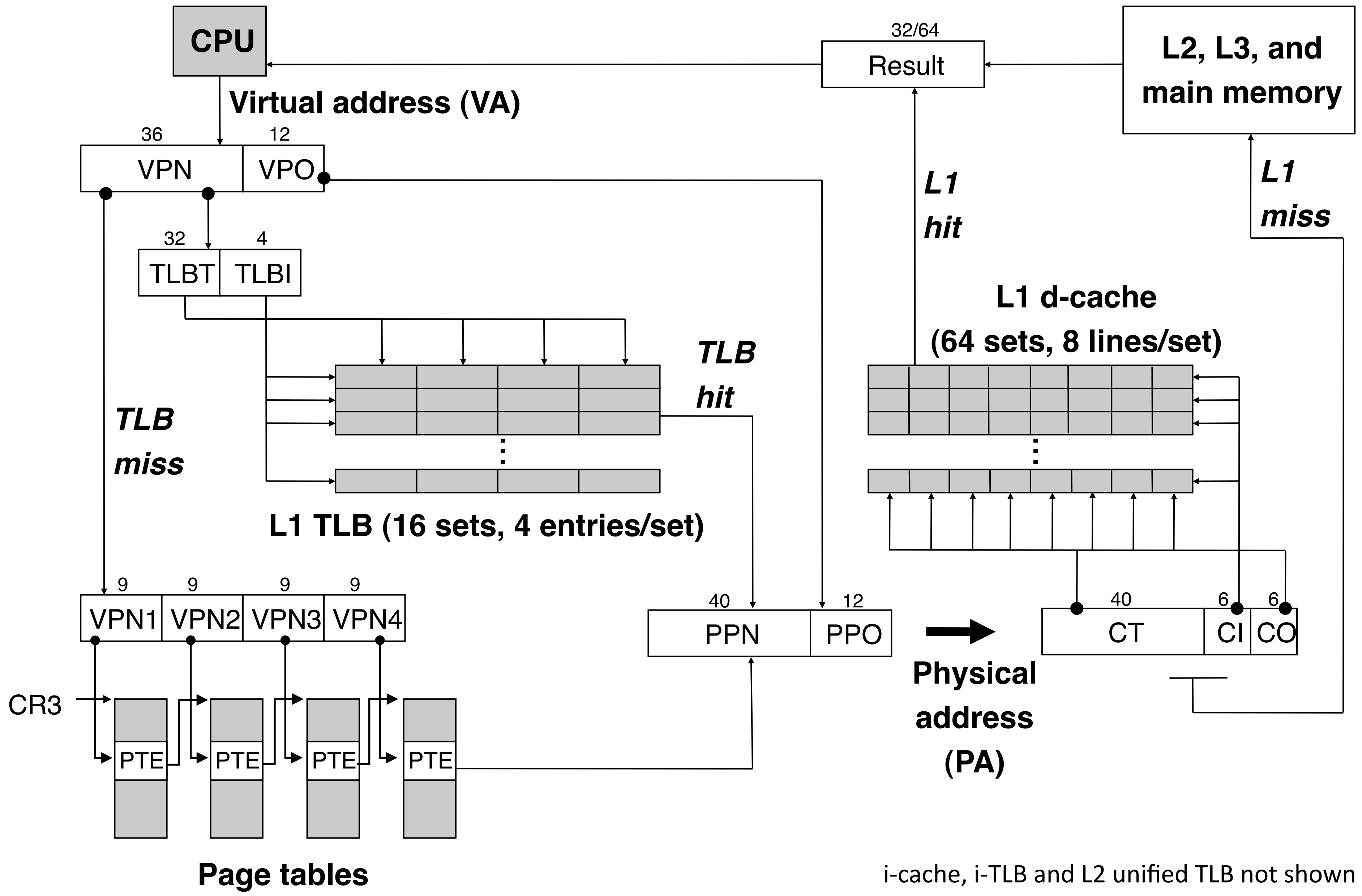
- Address Translation
- Multi-level page tables
- **Intel Core i7 Address Translation**

Intel Core i7

- **Nehalem microarchitecture**
  - Allows for full 64-bit virtual and physical address
  - Current implementations (Bloomfield, Lynnfield, etc):
    - 48-bit virtual address (256 TB)
    - 52-bit physical address (4 PB)
  - Page size is either 4KB or 4MB (Linux uses 4KB)


Registers
Instruction fetch
MMU (addr translation)
L1 d-cache 32 KB, 8-way
L1 i-cache 32 KB, 8-way
L1 d-TLB 64 entries, 4-way
L1 i-TLB 128 entries, 4-way
L2 unified cache 256 KB, 8-way
L2 unified TLB 512 entries, 4-way
QuickPath interconnect 4 links @ 25.6 GB/s 102.4 GB/s total
To other cores
To I/O bridge
Core x(2-6)
L3 unified cache 8 MB, 16-way (shared by all cores)
DDR3 Memory controller 3 x 64 bit @ 10.66 GB/s 32 GB/s total (shared by all cores)
To Main memory
Chip

Intel Core i7 Address Translation



i-cache, i-TLB and L2 unified TLB not shown

Further Reading

- Intel TLBs:
  - Application Note:
    “TLBs, Paging-Structure Caches, and Their Invalidation”, April 2007

Summary

- **Address Translation**
- **Multi-level Page Tables**
- **Intel Core i7 Address Translation**

- **Next Time: Virtual Memory III**
  - Linux memory management