

TLBs

Jinkyu Jeong (jinkyu@skku.edu)

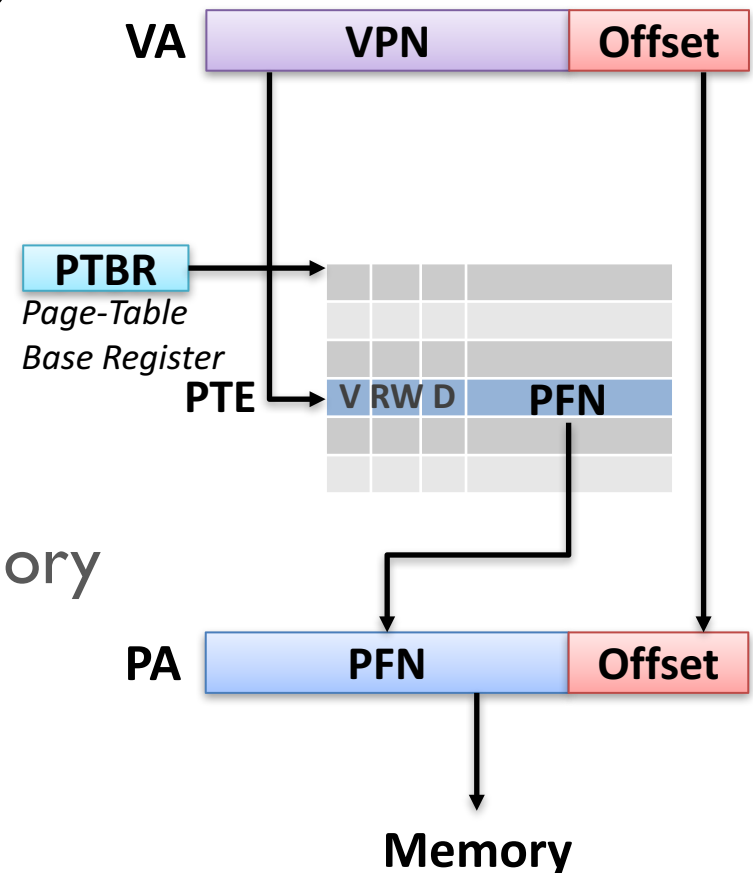
Computer Systems Laboratory

Sungkyunkwan University

<http://csl.skku.edu>

Address Translation Steps

- For each memory reference,
 - Extract VPN from VA
 - Calculate the address of PTE
 - Read the PTE from memory
 - Extract PFN from PTE
 - Build PA
 - Read contents of PA from memory into register
- Which steps are expensive?



The Problem

- **Address translation is too slow**
 - A simple linear page table doubles the cost of memory lookups
 - One for the page table, another to fetch the data
 - Multi-level page tables increase the cost further (discussed later)
- **Goal: make address translation fast**
 - Make fetching from a virtual address about as efficient as fetching from a physical address

TLB

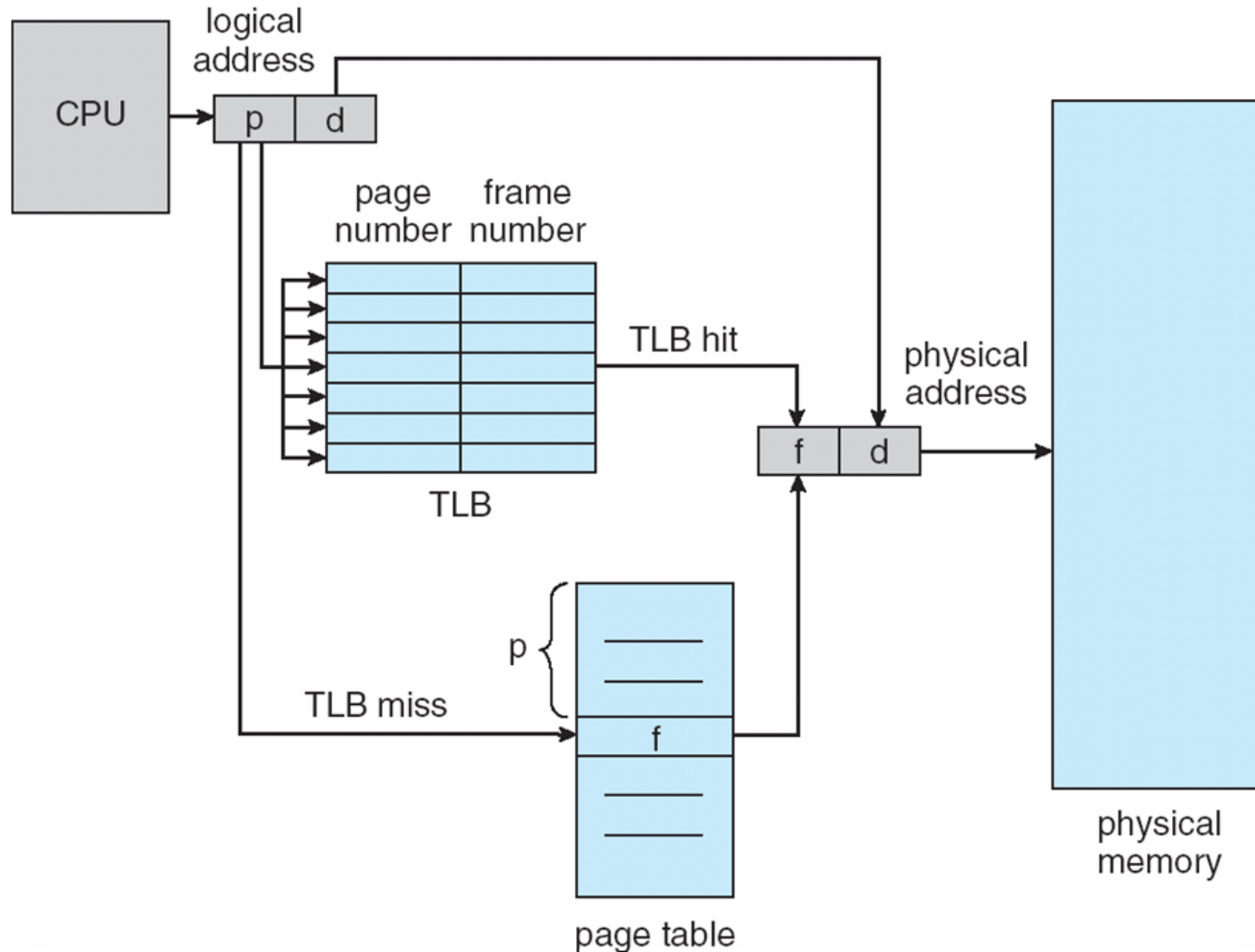
- Translation Lookaside Buffer
 - A hardware cache of popular virtual-to-physical address translations
 - Essential component which makes virtual memory possible
- TLB exploits locality
 - **Temporal locality**: an instruction or data item that has been recently accessed will likely be re-accessed soon
 - Instructions and data accesses in loops, ...
 - **Spatial locality**: if a program accesses memory at address x , it will likely soon access memory near x
 - Code execution, array traversal, stack accesses, ...

TLB Organization

- TLB is implemented in hardware
 - Processes only use a handful of pages at a time
 - 16~256 entries in TLB is typical
 - Usually fully associative
 - All entries looked up in parallel
 - But may be set associative to reduce latency
 - Replacement policy: LRU (Least Recently Used)
 - TLB actually caches the whole PTEs, not just PFNs

Valid	Tag (VPN)	Value (PTE)					
1	0x1000	V	R	M	Prot	PFN	0x1234
1	0x2400	V	R	M	Prot	PFN	0x8800
0	-	-					

Address Translation with TLB



Handling TLB Misses

- **Software-managed TLB**
 - CPU traps into OS upon TLB miss
 - OS finds right PTE and loads it into TLB
 - CPU ISA has (privileged) instructions for TLB manipulation
 - Page tables can be in any format convenient for OS (flexible)
- **Hardware-managed TLB**
 - CPU knows where page tables are in memory (PTBR)
 - e.g. CR3 (or PDBR) register in IA-32
 - OS maintains page tables
 - CPU “walks” the page table and fills TLB
 - Page tables have to be in hardware-defined format

TLB on Context Switches

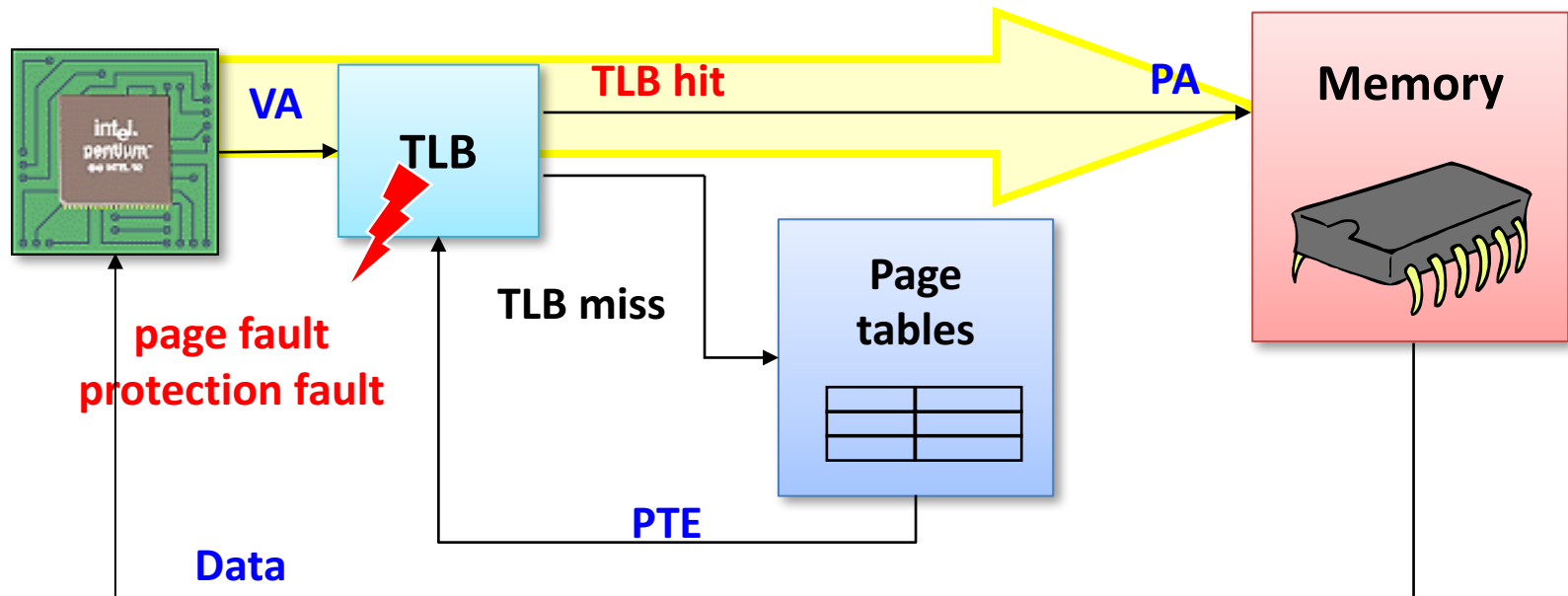
- Flush TLB on each context switch
 - TLB is flushed automatically when PTBR is changed in a hardware-managed TLB
 - Some architectures support the pinning of pages into TLB
 - For pages that are globally-shared among processes (e.g. kernel pages)
 - MIPS, Intel, etc.
- Track which entries are for which process
 - Tag each TLB entry with an ASID (Address Space ID)
 - A privileged register holds the ASID of the current process
 - MIPS supports 8-bit ASID
 - Why not use PID?
 - What if there are more than 256 processes running?

TLB Performance

- TLB is the source of many performance problems
 - Performance metric: hit rate, lookup latency, ...
- Increase TLB reach (= # TLB entries * Page size)
 - Increase the page size: e.g. 2MB, 1GB page support in Intel 64
 - Increase the TLB size
- Use multi-level TLBs
 - e.g. Intel Haswell (4KB pages): L1 ITLB 128 entries (4-way), L1 DTLB 64-entries (4-way), L2 STLB 1024 entries (8-way)
- Change your algorithms and data structures to be TLB-friendly

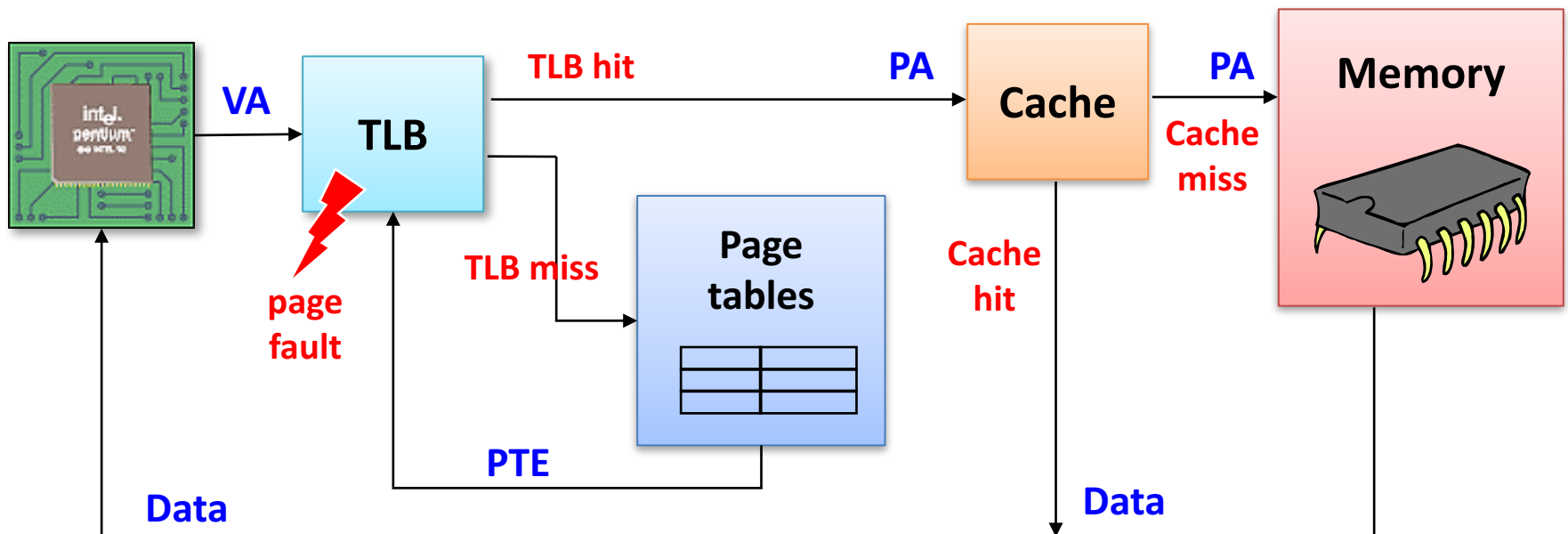
From CPU to Memory

- A process is executing on the CPU, and it issues a read to a virtual address



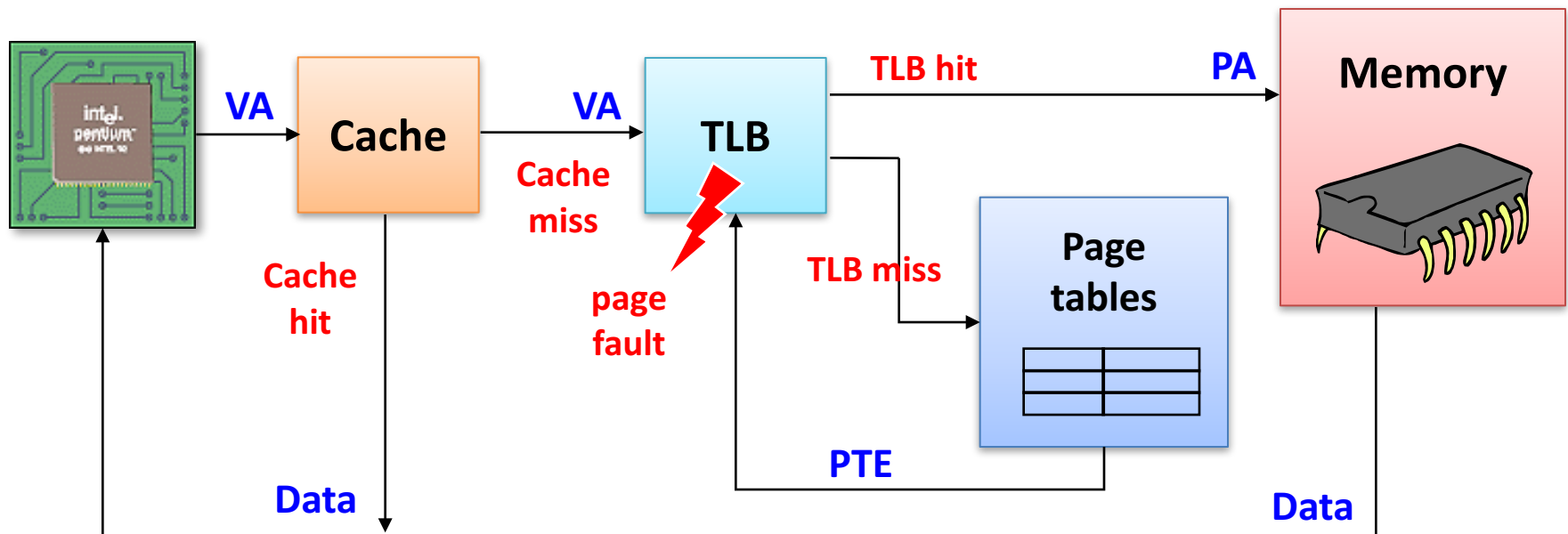
Integrating VM and Cache (I)

- Physically addressed cache
 - Allows multiple processes to have blocks in cache
 - Allows multiple processes to share pages
 - Address translation is on the critical path



Integrating VM and Cache (2)

- Virtually addressed, virtually tagged cache
 - Homonym problem
 - Each process has a different translation of the same virtual address
 - Address synonyms or aliases problem
 - Two different virtual addresses point to the same physical address



Integrating VM and Cache (3)

- Virtually addressed, physically tagged cache
 - Use virtual address to access the TLB and cache in parallel
 - TLB produces the PFN – which must match the physical tag of the accessed cache line for it to be a “hit”

