Virtual Memory

COE 501

Computer Architecture

Prof. Muhamed Mudawar

Computer Engineering Department

King Fahd University of Petroleum and Minerals

Presentation Outline

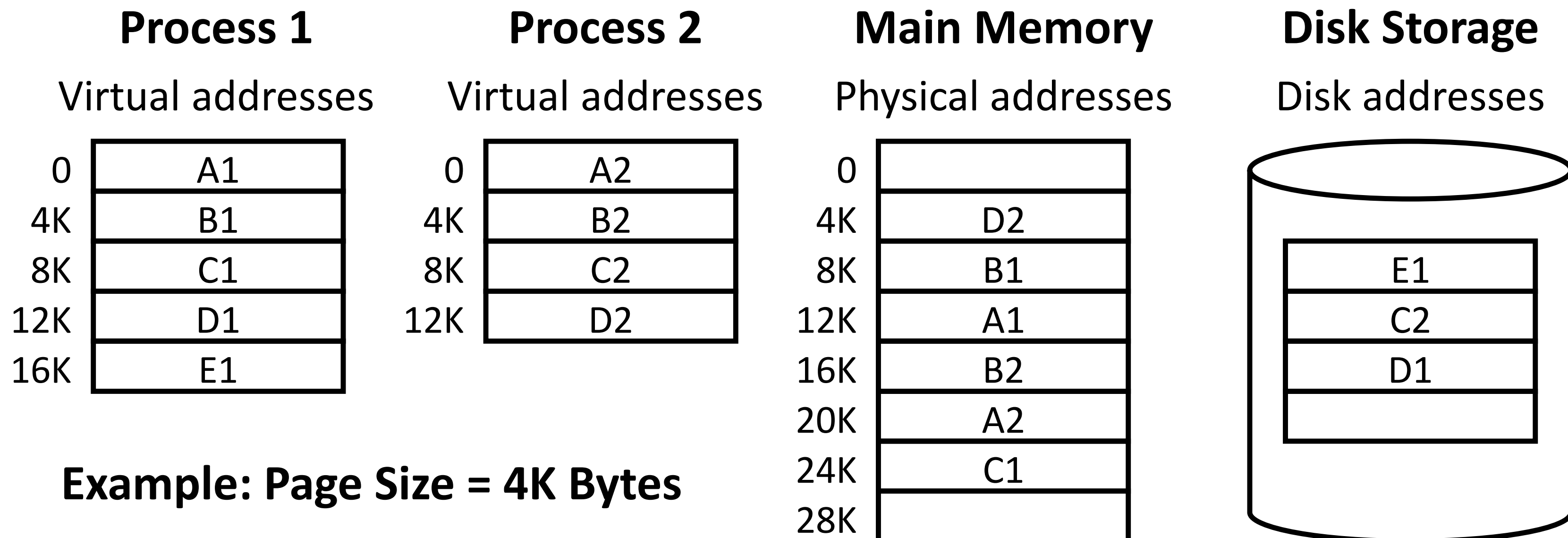
- **What is Virtual Memory?**
- **Fast Address Translation**

What is Virtual Memory?

- Extends main memory into disk storage
  - In early computers, main memory was small and expensive
  - Virtual memory enables a program to exceed the memory size
- Defines a **virtual address space** for a running program
  - A running program is called a **process**
  - Multiple processes can exist in memory at the same time
- Simplifies **Memory Management** done by the OS
  - Operating system software allocates memory to each process
- Provides **protection** to processes and operating system
  - Main memory is **shared** by processes and operating system
  - Virtual memory enforces protection through **address translation**

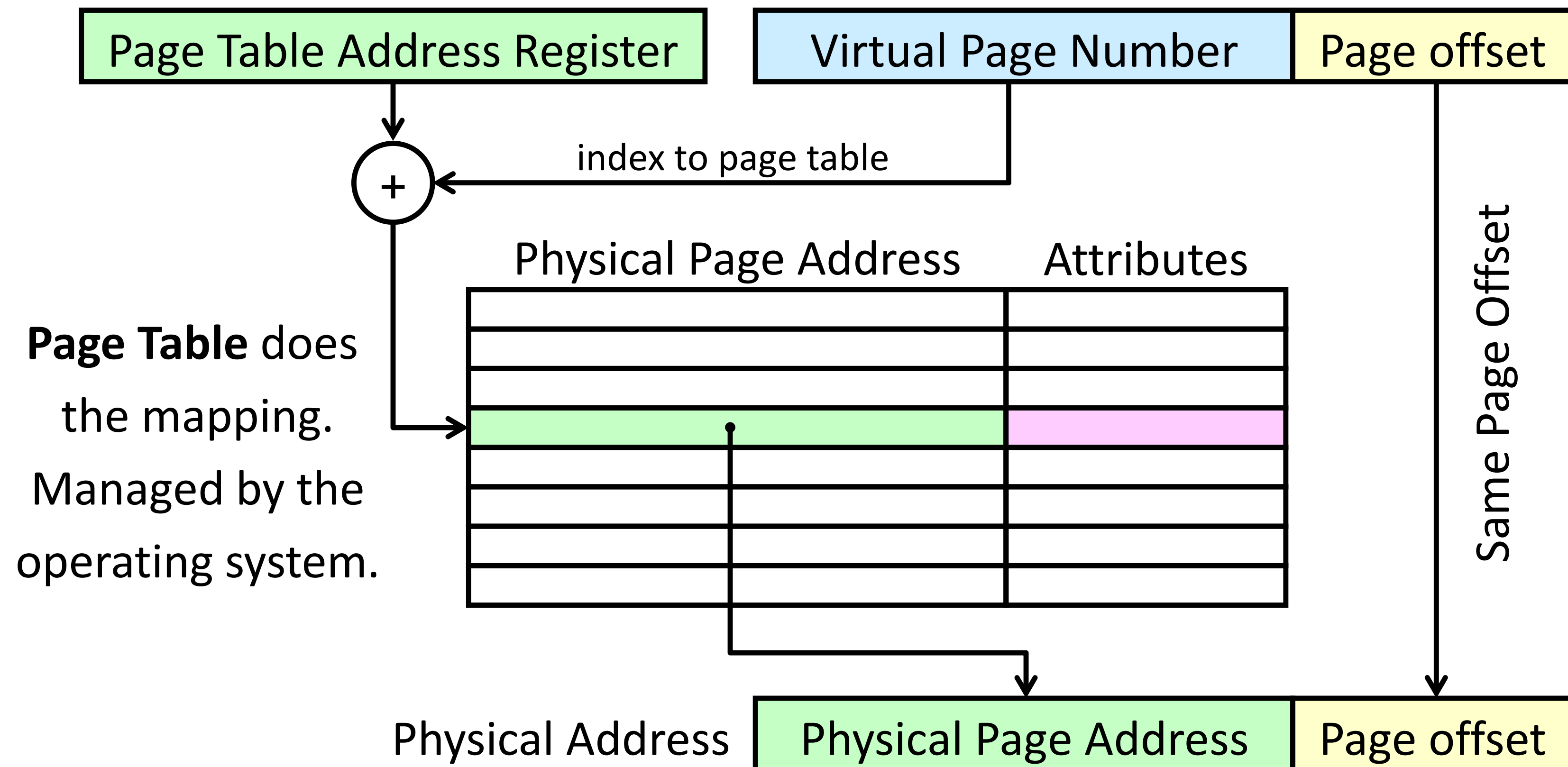
What is Paging?

- A process virtual address space is divided into **pages**
  - All pages have the same fixed size (simplifies their allocation)
- Operating system allocates and maps pages
  - Either in main memory or on disk (if no sufficient memory space)
  - **Page table** does the mapping from virtual to physical addresses



What is a Page Table?

- Pages appear contiguous in the virtual address space
  - However, they can be scattered in main memory (and disk)



Page Table

- Each process has a page table
  - The page table defines the physical address space of a process
  - Physical address space = physical pages that can be accessed
  - Page table is stored in main memory
  - Managed only by the operating system
- Page table address register
  - Contains the physical address of the page table in memory
  - Processor uses this register to locate page table in memory
- Page table entry
  - Contains physical page number and attributes of a single page
  - Attributes specify page presence, protection and use

Page Table Entry Format

- Each Page Table Entry (PTE) stores:
  - Physical address of a page in memory
  - Page attributes (vary according to architecture)
  - PTE is 4 bytes for 32-bit, and 8 bytes for 64-bit architectures
- Typical page attributes
  - Presence bit: indicates whether page is present in memory
  - Read/Write: whether page is read-only or can be written
  - User/Supervisor: whether page is for operating system use only
  - Accessed: whether page has been accessed recently
  - Dirty: whether page has been modified
  - Cache disable: whether page can be cached or not
  - Page size: whether page is small or large

Analogy to Caches

Cache Concept	Virtual Memory Concept
Cache Block = 32, 64, or 128 bytes	Small Page = 4 KB, Large Page = 4 MB
Cache Miss: Block not found in Cache	Page Fault: Page not present in memory
Transfer block from memory to cache	Transfer page from disk to memory
Miss Rate: 0.1% to 10%	Page Fault Rate: 0% to 0.001%
Miss Penalty: 8 to 200 clock cycles	Page Fault Penalty: 10^6 to 10^7 cycles
Cache Miss is handled in hardware	Page Fault is handled in software
Placement: Direct mapped, set Associative	Page placed anywhere in memory
Tags identify cache blocks (hit or miss)	Page table indicates page presence
Block replacement done is hardware	Page replacement done in software
Write policy: Write-through or Write-back	Write-back only (Modified bit)

Advantages of Virtual Memory

- Memory Management
  - Programs are given contiguous view of virtual memory
  - Physical pages are simple to allocate and can be scattered
  - Only the working set of a program must be in main memory
  - Heap and stack can grow (use as many pages as needed)
- Protection
  - Different processes are protected from each other
  - Pages are given special attributes (read only, write, execute)
  - Operating system data protected from user programs
- Sharing
  - Can map same physical page to multiple processes
  - Processes can share library code and can also share data

Issues in Virtual Memory

- Page Size
  - Small page sizes ranging from 4KB to 16KB are typical today
  - Large page size can be 2MB or 4MB (reduces page table size)
  - Recent processors support multiple page sizes
- Fully associative placement to reduce page faults
- Handling Page Faults and Replacement Policy
  - Page faults are handled in software by the operating system
  - Reference bit per page: which page is referenced recently
  - Modified bit per page: which page is modified
- Reducing the page table size
- Supporting fast address translation

Page Fault / Invalid Memory Access

- Page Fault: requested page is not present in memory
  - The missing page is located on disk and transferred to memory
  - It takes milliseconds to transfer a page from disk to memory
  - Another process may run while first process is waiting
  - A free page is allocated in memory and page table is updated
  - Program is restarted at the instruction that caused page fault
- Invalid memory access: several cases such as
  - Reference to a page not part of the virtual address space
  - Writing to a read-only page
  - Accessing a supervisor (operating system) page in user mode
  - Operating system terminates program (segmentation fault)

Page Replacement Algorithm

- Working Set of a Process
  - Set of pages expected to be used during some time interval
  - Page faults occur when the working set is not in memory
- Page replacement algorithm
  - Decide which memory pages to swap out (write to disk)
  - To free pages in memory, when number of free pages is low
- Many replacement algorithms: FIFO, NRU, Clock
- Clock replacement algorithm
  - Keep a circular list of pages in memory (circular FIFO)
  - Point to next page to replace and examine the access bit
  - Skip pages that have been accessed (replace if Access bit = 0)
  - Operating system clears the access bits periodically

Size of the Page Table

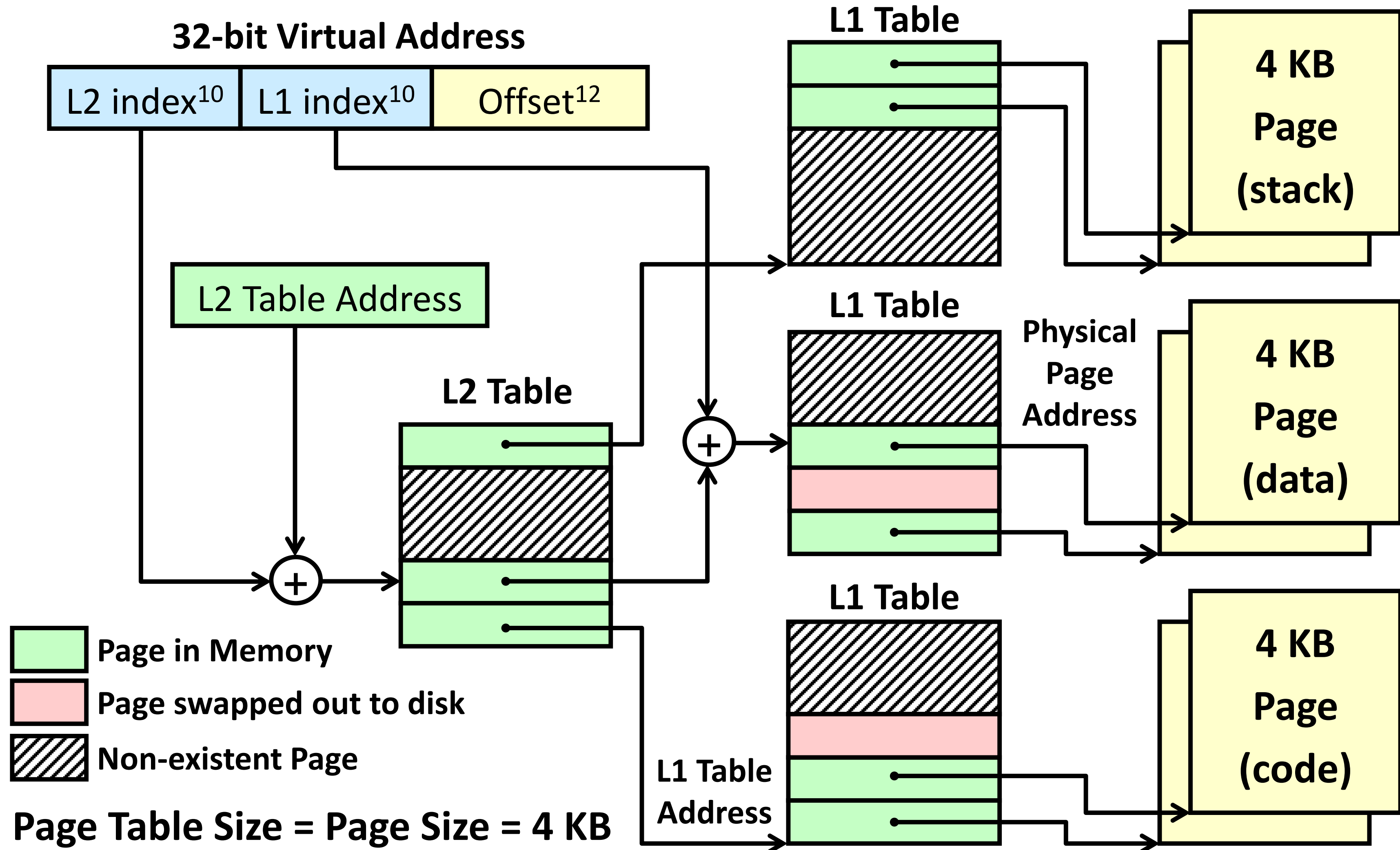
- One-level page table is the simplest to implement
- Consider: 32-bit virtual address space with 4KB pages
- Page offset = 12 bits and Virtual Page Number = 20 bits

- What is the size of the page table if entry = 4 bytes?
- Answer: $2^{32}/2^{12} = 2^{20}$ entries × 4 bytes = 4 MB
- Now consider: 48-bit virtual address with 4KB pages
- What is the size of the page table if entry = 8 bytes?
- Answer: $2^{48}/2^{12} = 2^{36}$ entries × 8 bytes = 512 GB !

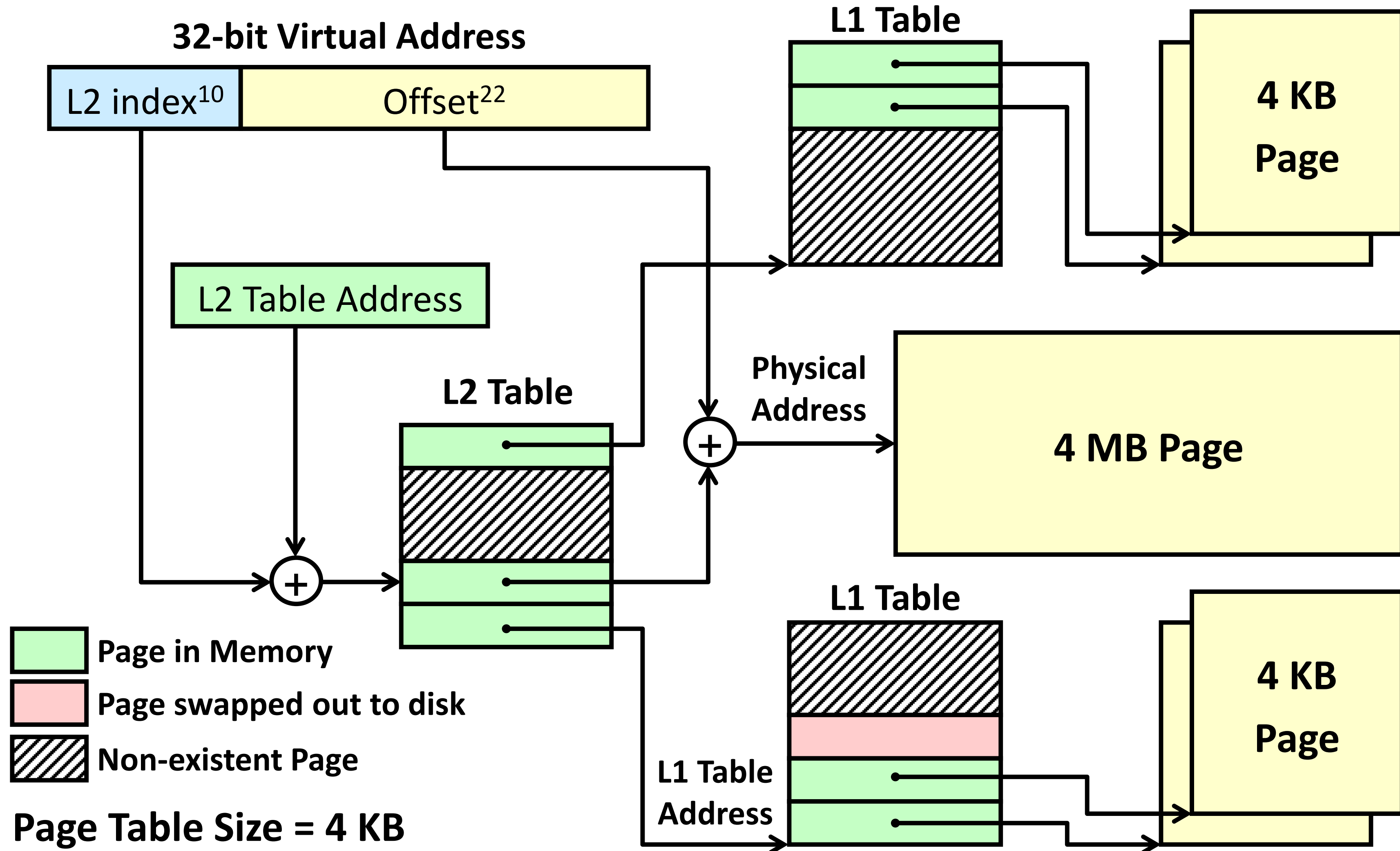
Reducing the Page Table Size

- Many processes have a small virtual address space
  - Might need only few pages for code, data, and stack
- Use Limit Register to restrict the size of the page table
  - Does not work well when the virtual address space is sparse
- Use Hierarchical Page Table for sparse address space
  - Small page tables (size of a page) allocated at different levels
  - Can efficiently map multiple dynamic stacks and heaps
  - Disadvantage: multiple table traversal for address translation
- Use Multiple Page Sizes rather than one page size
  - Large page size works better for large virtual address space
- Use Hashed Page Table shared by all processes

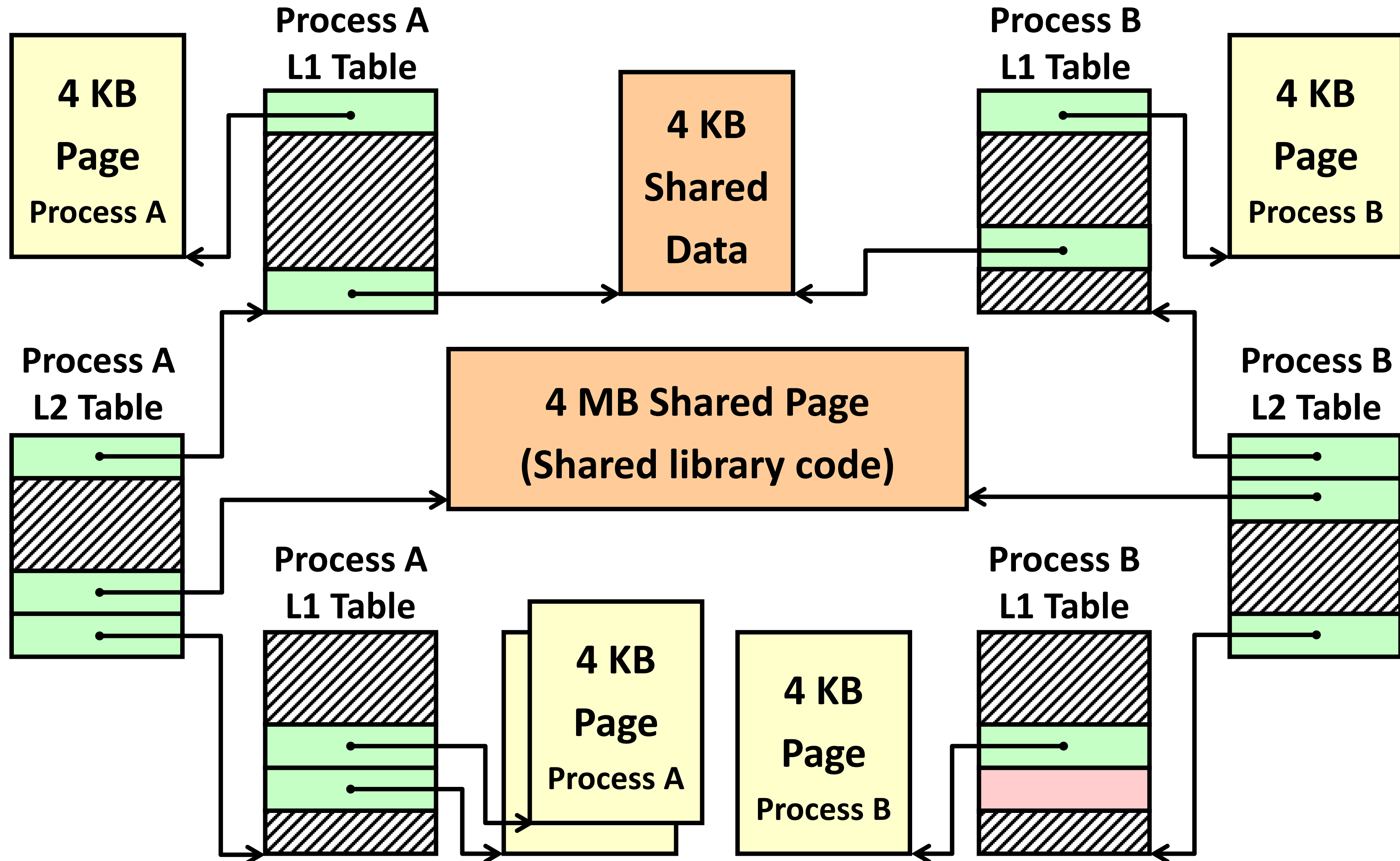
Two-Level Page Tables (Intel 32-bit)



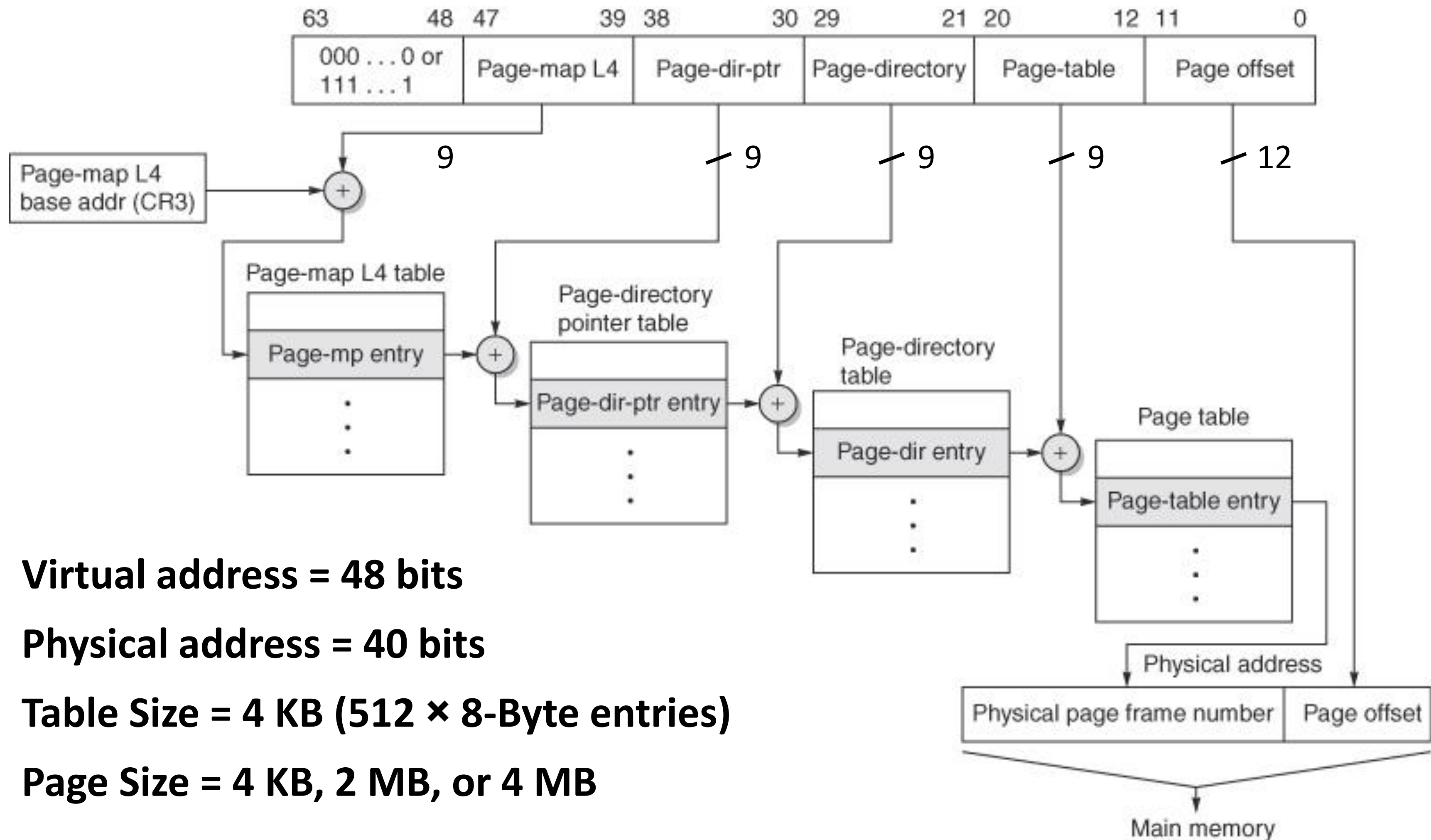
Variable-Size Page Support



Sharing Pages Among Processes



Four-Level Page Tables (AMD64)



Virtual address = 48 bits

Physical address = 40 bits

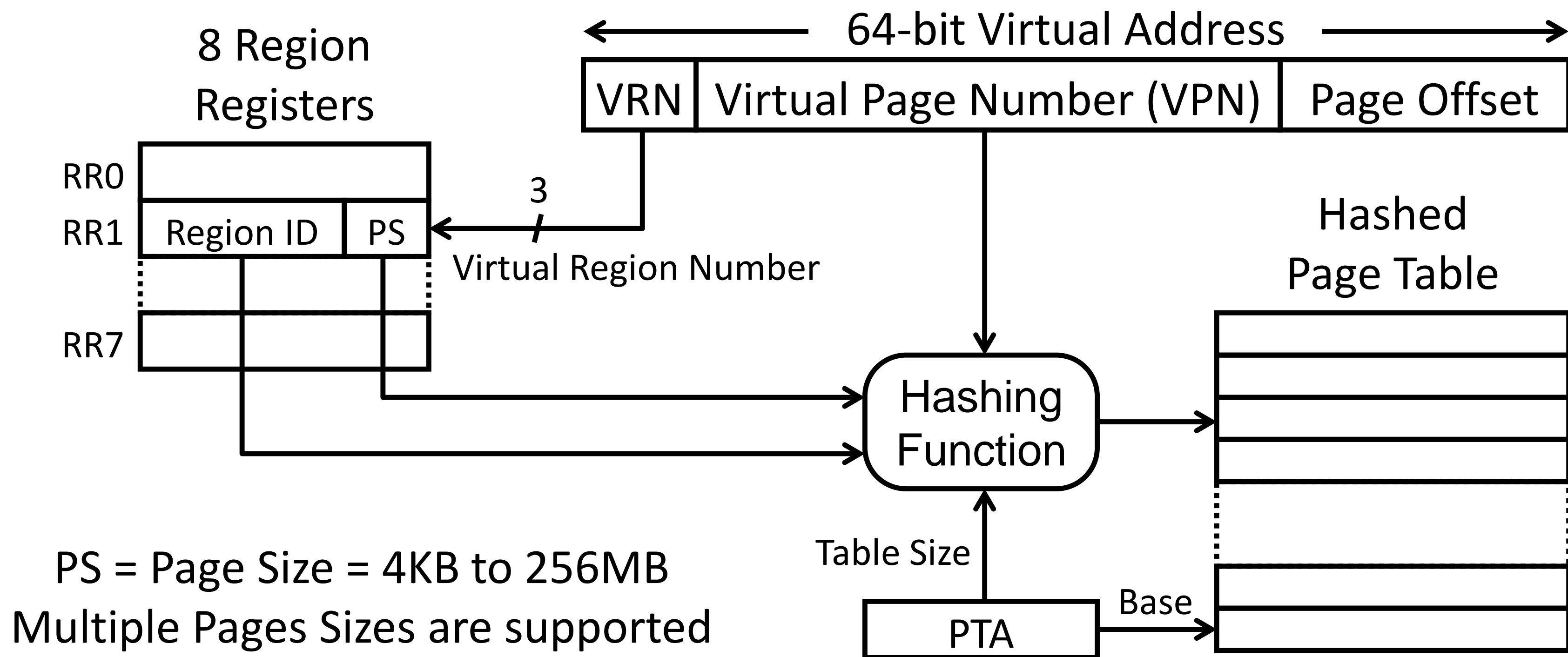
Table Size = 4 KB (512 × 8-Byte entries)

Page Size = 4 KB, 2 MB, or 4 MB

Copyright © 2012, Elsevier Inc. All rights Reserved.

System-Wide Hashed Page Table

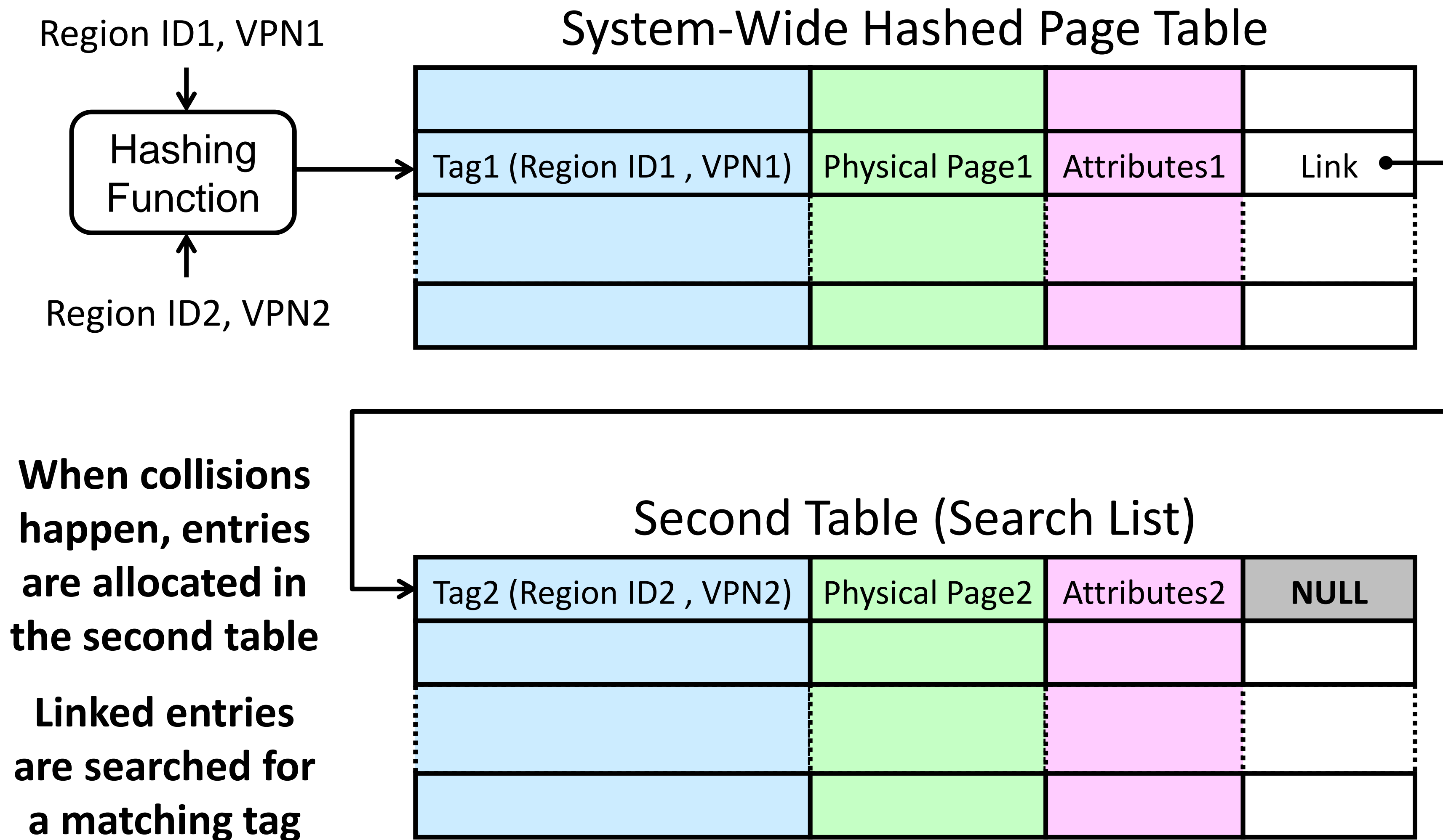
- Shared by all processes and operating system
- Example: Intel Itanium architecture
- Uses the concept of virtual regions (8 per process)



Hashed Page Table Entry

- Hashed Page Table Entry includes the following fields:
  - Tag that matches Region ID and Virtual Page Number
  - Presence bit, Physical Page Address
  - Access bit, Dirty bit, Privilege level, Access rights
  - Link in case of collision (multiple entries are searched)
  - Hashed page table entry can be long (32 bytes on the Itanium)
- Example of a simple hashing function:
  - Hashed Page Table Index = (VPN $\oplus$ Region ID) % Table Size
  - Generates an index between 0 and Table Size – 1
  - Page Size (PS) specifies the number of bits in VPN
  - Hashed Page Table is searched by content

Illustrating Collisions & Search List



Advantages / Disadvantages

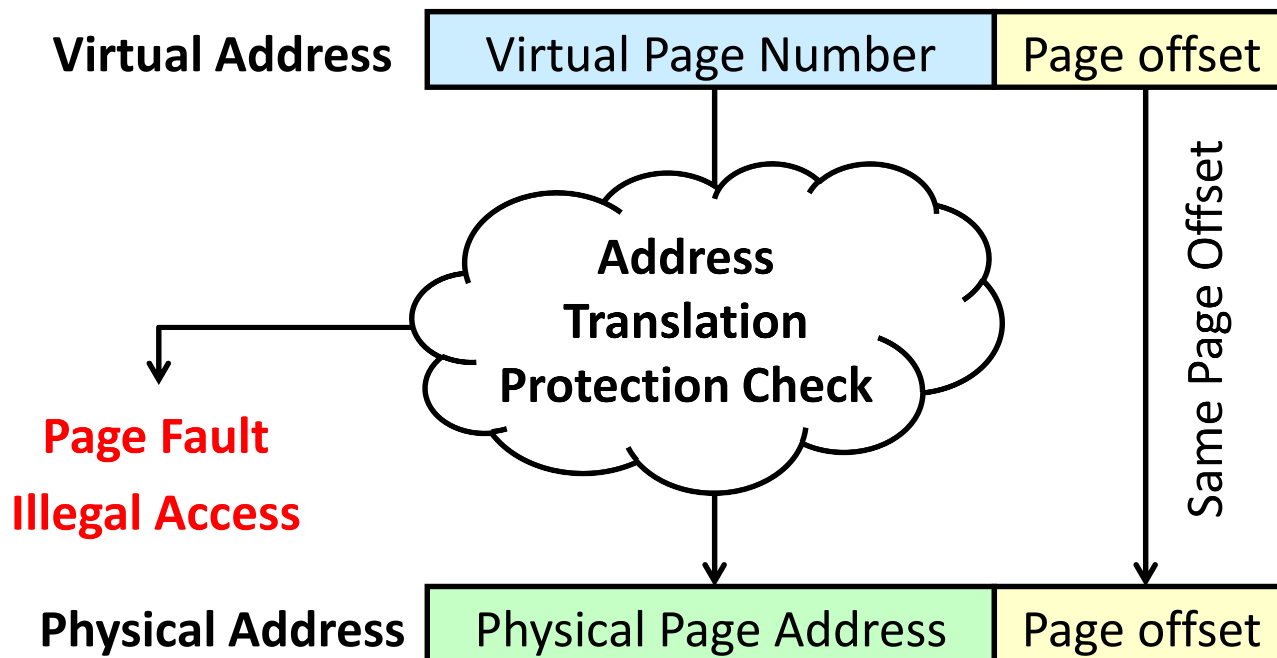
- Advantages of Hashed Page Table
  - Table is only a small fraction of memory
  - Scales with physical memory, not with the virtual address space
  - Sufficient number of entries to reduce collision probability
  - Region ID can be shared by multiple processes
  - No problem with sparse virtual address space
- Disadvantages
  - Hashed page table entry is larger
  - Collisions are unavoidable
  - Must traverse search list when collisions occur (can be long list)

Presentation Outline

- What is Virtual Memory?
- **Fast Address Translation**

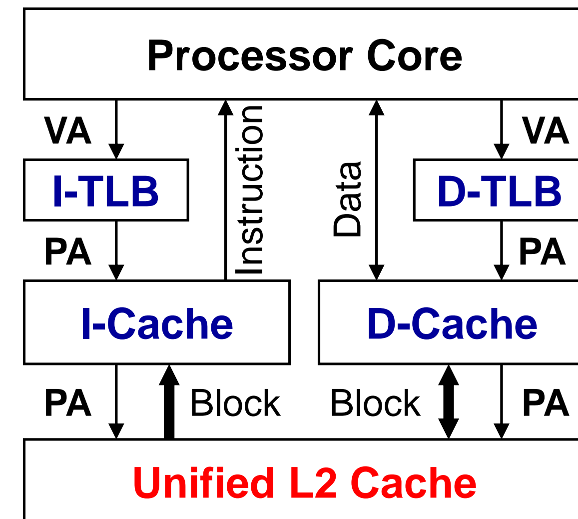
Address Translation and Protection

- Program generates virtual addresses
  - Must translate virtual address on every memory access
  - Must check whether page in memory (page fault)
  - Must check protection (illegal access, invalid page)



Fast Address Translation

- Address translation is expensive
  - Must translate each virtual address on every memory access
  - Multilevel page table ➔ translation is several memory accesses
- Solution: **Translation Lookaside Buffer (TLB)**
  - TLB = Cache for address translation
  - TLB = Fast address translation
  - TLB input = VA = Virtual Address
  - TLB output = PA = Physical Address
  - I-TLB = I-Cache TLB
  - D-TLB = D-Cache TLB



TLB Entries

- TLB keeps track of recently accessed pages
  - Virtual and Physical page numbers (fast address translation)
  - Accessed and Dirty bits (whether page is accessed or modified)
  - Access rights and Privilege level protection bits
- Additional TLB fields
  - Address Space Identifier (or Region ID)
    - Allows multiple processes to be in the TLB at the same time
    - Otherwise, TLB should be flushed on a context switch
  - Global bit (G bit) for global pages that are shared by all processes
  - Page size (PS) for variable page sizes

TLB Organization

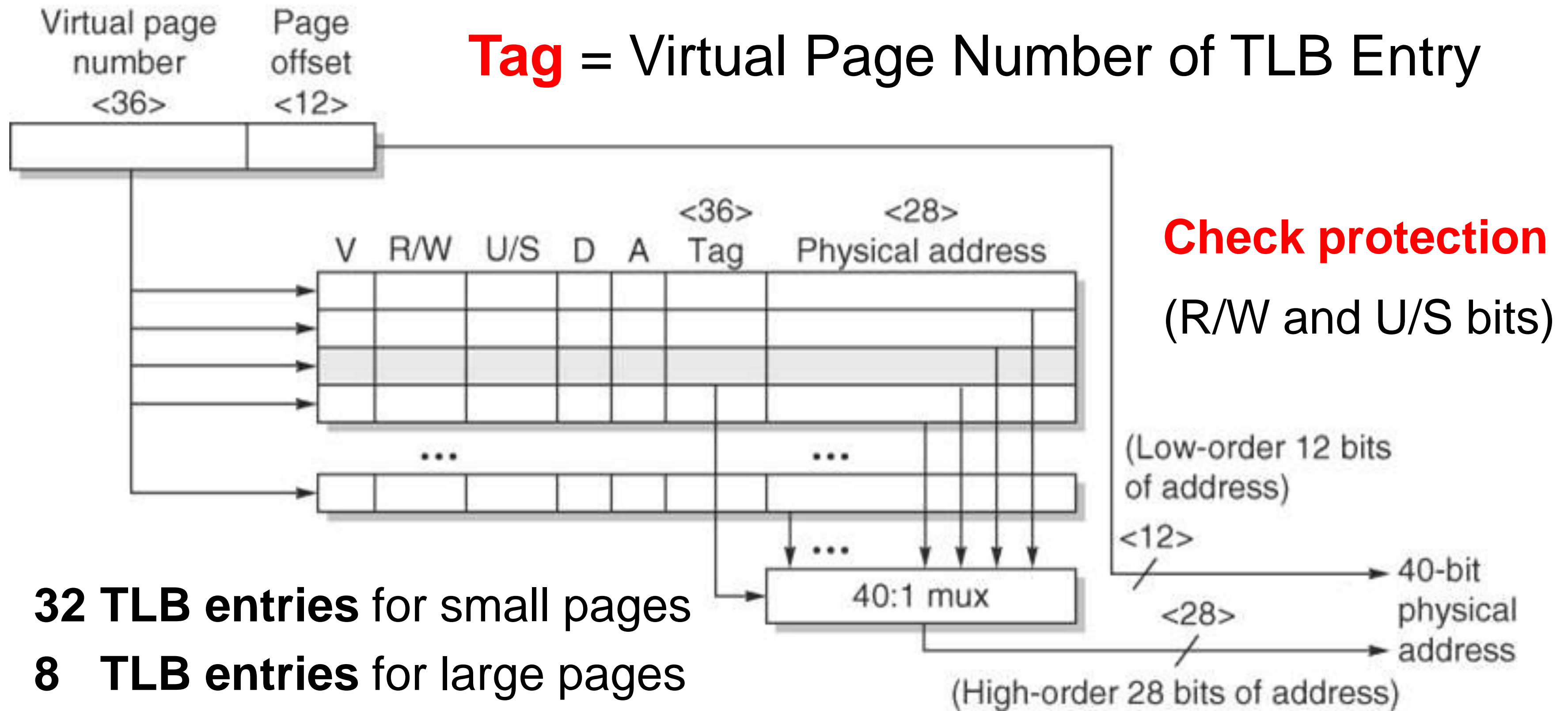
- TLB size can vary between 32 and 512 entries
  - Small TLBs are fully associative
  - Large TLBs are set associative
- Large system can have multi-level TLBs (L1 and L2)
- Hit time is typically one clock cycle for small TLBs
  - TLB Miss Penalty = few cycles to hundreds of clock cycles
  - Miss Rate = 0.01% to 1%
- Random, FIFO, or pseudo-LRU replacement (TLB miss)
- **TLB reach**: maximum virtual space mapped by TLB
  - Example: 64 TLB entries, 4 KB pages, one page per entry
  - TLB reach = 64 entries × 4 KB = 256 KB

Fully Associative TLB (AMD Opteron)

Tag = Virtual Page Number of TLB Entry

Check protection

(R/W and U/S bits)



32 TLB entries for small pages

8 TLB entries for large pages

Copyright © 2012, Elsevier Inc.
All rights Reserved.

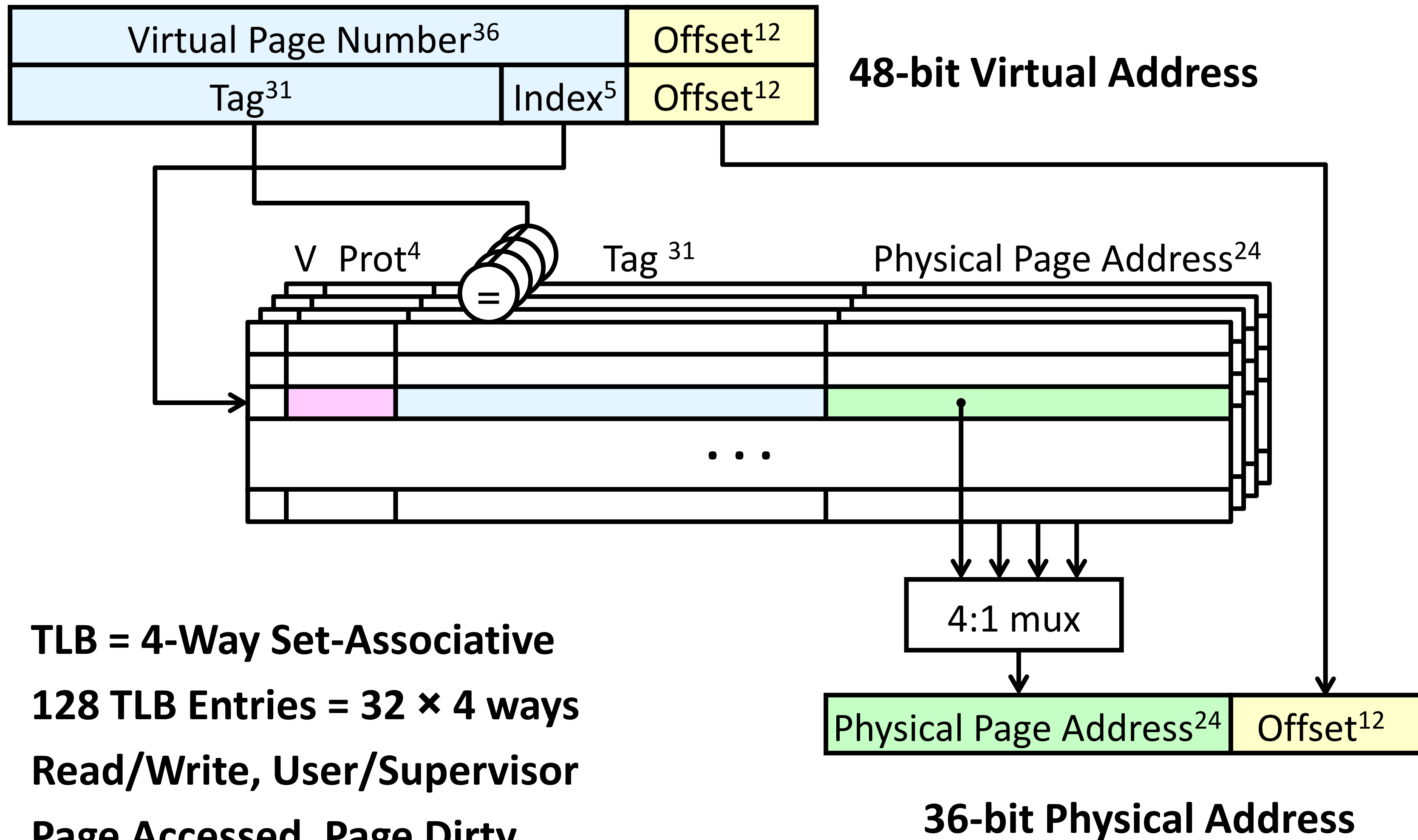
TLB hit: TLB Entry Found

Get physical page address

Fast single cycle translation

TLB miss: Must traverse page table ➔ Slow translation (update TLB)

Set Associative TLB (Intel Core i7)



Second-Level TLB

- Second-Level L2 TLB is used when miss in L1 TLB
- L2 TLB is larger than I-TLB and D-TLB
  - Provides more entries for address translation
  - Miss in L1, which is a hit in L2 TLB ➔ Swap TLB entries
  - Miss in L1 and L2 TLBs ➔ Page table address translation (slow)

Core i7 TLBs	I-TLB	D-TLB	L2 TLB (unified)
Size	128 entries	64 entries	512 entries
Associativity	4-way	4-way	4-way
Replacement	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU
Access Latency	1 cycle	1 cycle	6 cycles
Miss Penalty	7 cycles	7 cycles	Hundreds of cycles

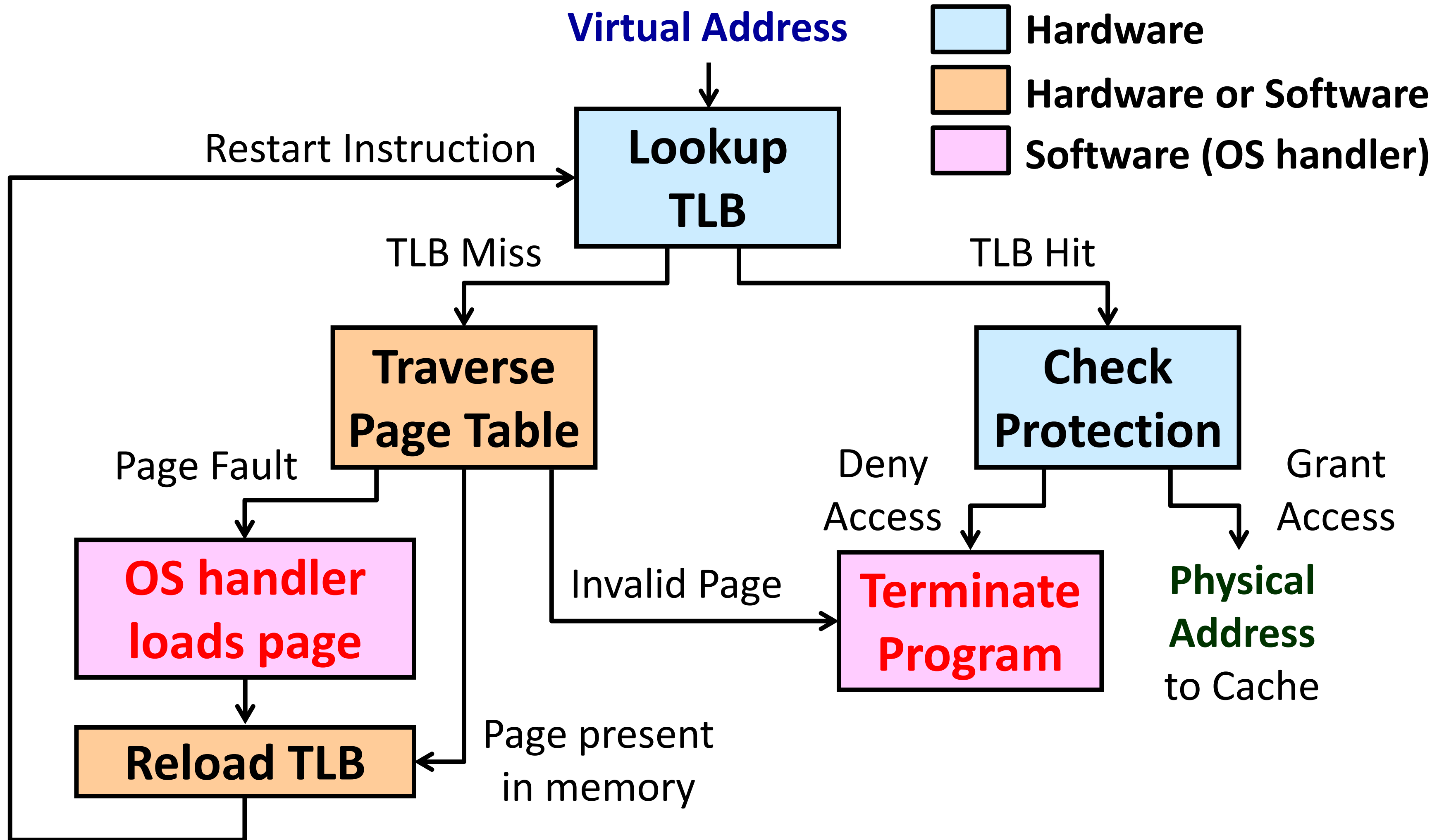
Handling TLB Misses and Page Faults

- **TLB miss**: No TLB entry matches the virtual address
- TLB miss can be handled in software or in hardware
  - Traverse the page table hierarchy (by OS handler or MMU)
  - If page table entry in memory is valid, then reload it into the TLB
  - If page not present in main memory then page fault
- **Page Fault**: Causes a context switch
  - Interrupt the program at the instruction that caused the page fault
    - Save the process context in memory (PC and registers)
  - Transfer control to the operating system to transfer the page
  - Meanwhile, operating system schedules another process to run
  - Later, restart the instruction that caused the page fault

TLB, Page Table, Cache Combinations

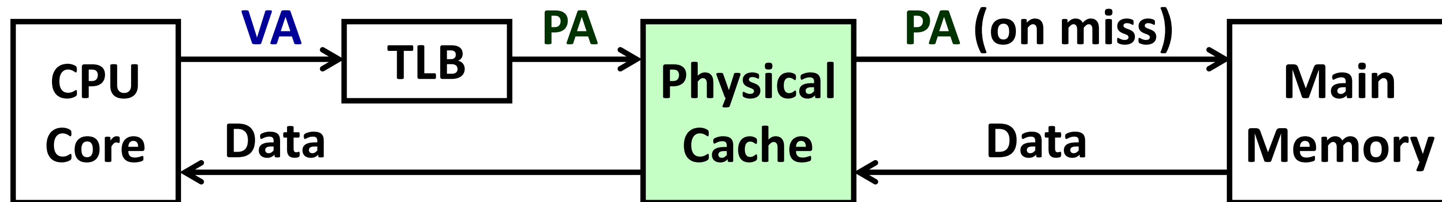
TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes: this is what we want!
Hit	Hit	Miss	Yes: TLB hit ➔ page table is not checked Cache miss ➔ block in memory
Miss	Hit	Hit	Yes: TLB miss ➔ entry in page table
Miss	Hit	Miss	Yes: TLB miss ➔ entry in page table Cache miss ➔ block in memory
Miss	Miss	Miss	Yes: Page fault ➔ page swapped out
Hit	Miss	-	NOT possible: TLB translation is not possible if page is not in memory
Miss	Miss	Hit	NOT possible: data not allowed in cache if page is not in memory

Address Translation Summary

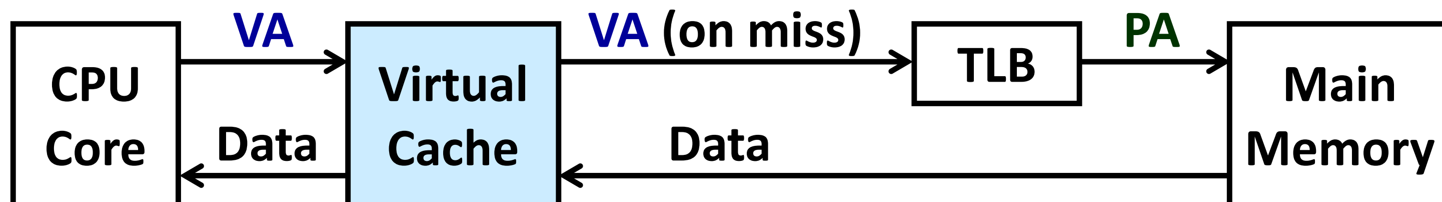


Physical versus Virtual Caches

- **Physical cache** is addressed with **physical addresses**
  - Virtual addresses are generated by the processor
  - Address translation is required, which may increase the hit time



- **Virtual cache** is addressed with **virtual addresses**
  - Address translation is not required for a hit (but only for a miss)



Virtual Cache Benefits/Drawbacks

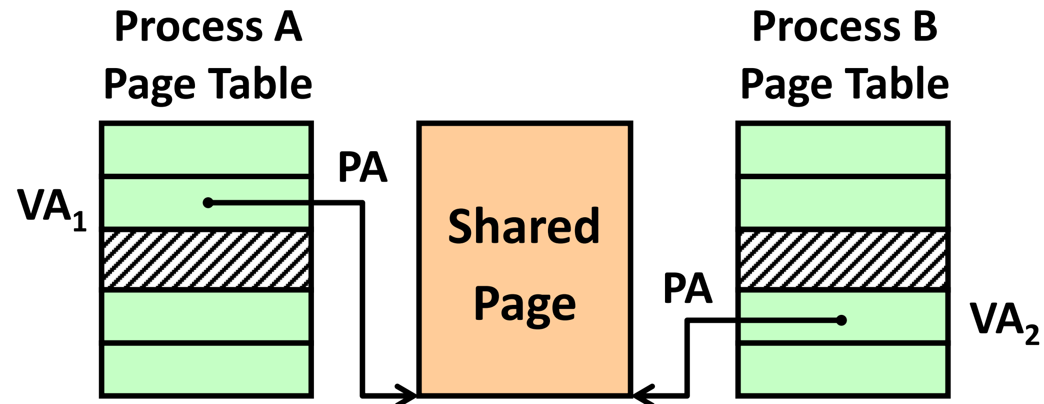
- Benefits of a Virtual Cache
  - Address translation is not required for a cache hit
  - TLB is not along critical path: cache access time is reduced
- Drawbacks of a Virtual Cache
  - Protection bits must be associated with each cache block
  - Flushing the virtual cache on a context switch
    - Unless address space identifiers are included in the tags
    - To avoid mixing virtual addresses of different processes
  - Aliasing problem due to the sharing of physical pages
    - Aliases: different virtual addresses map to same physical page
    - Multiple copies of the same block in a virtual cache
    - Updates make duplicate blocks inconsistent

Aliases in Virtual-Address Caches

Aliases:

Different Virtual Addresses

Map to same physical address



Virtual Cache can have **two copies** of the **same block** in main memory.

Writes to first copy of the block in the cache are not visible to second copy!

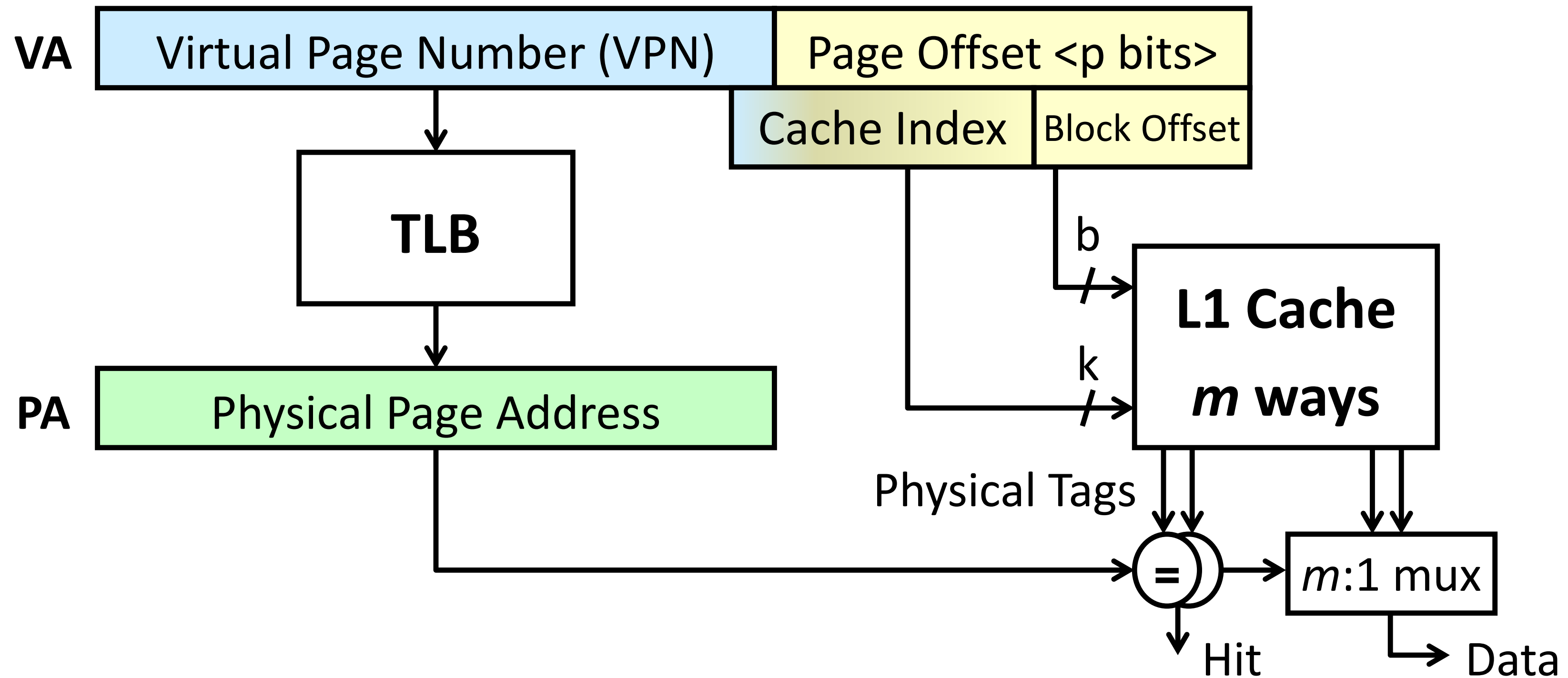
Tag	Data
VTag1	Copy 1 of Data Block at Physical Address PA
	. . .
VTag2	Copy 2 of Data Block at Physical Address PA

- General Solution: **Disallow aliases to coexist in cache**

TLB Translation during Indexing

- To lookup a cache, we should …
  - Index the cache: Physical or Virtual address can be used
  - Compare tags: Physical or Virtual address can be used
- Virtual cache eliminates address translation for a hit
  - However, causes problems (protection, flushing, and aliasing)
- Best combination for L1 cache
  - Address translation starts concurrently with indexing
    - Same page offset used in both virtual and physical address
    - Use part of page offset for indexing ➔ limits cache size
  - Compare tags using physical address (TLB output)
  - Ensure that each cache block has a unique physical address

Parallel Access to TLB and L1 Cache



Page Offset = p bits ➔ Page Size = 2^p bytes

Block Offset = b bits ➔ Block Size = 2^b bytes

Cache Index = k bits ➔ Cache Size = m ways × 2^k × 2^b bytes

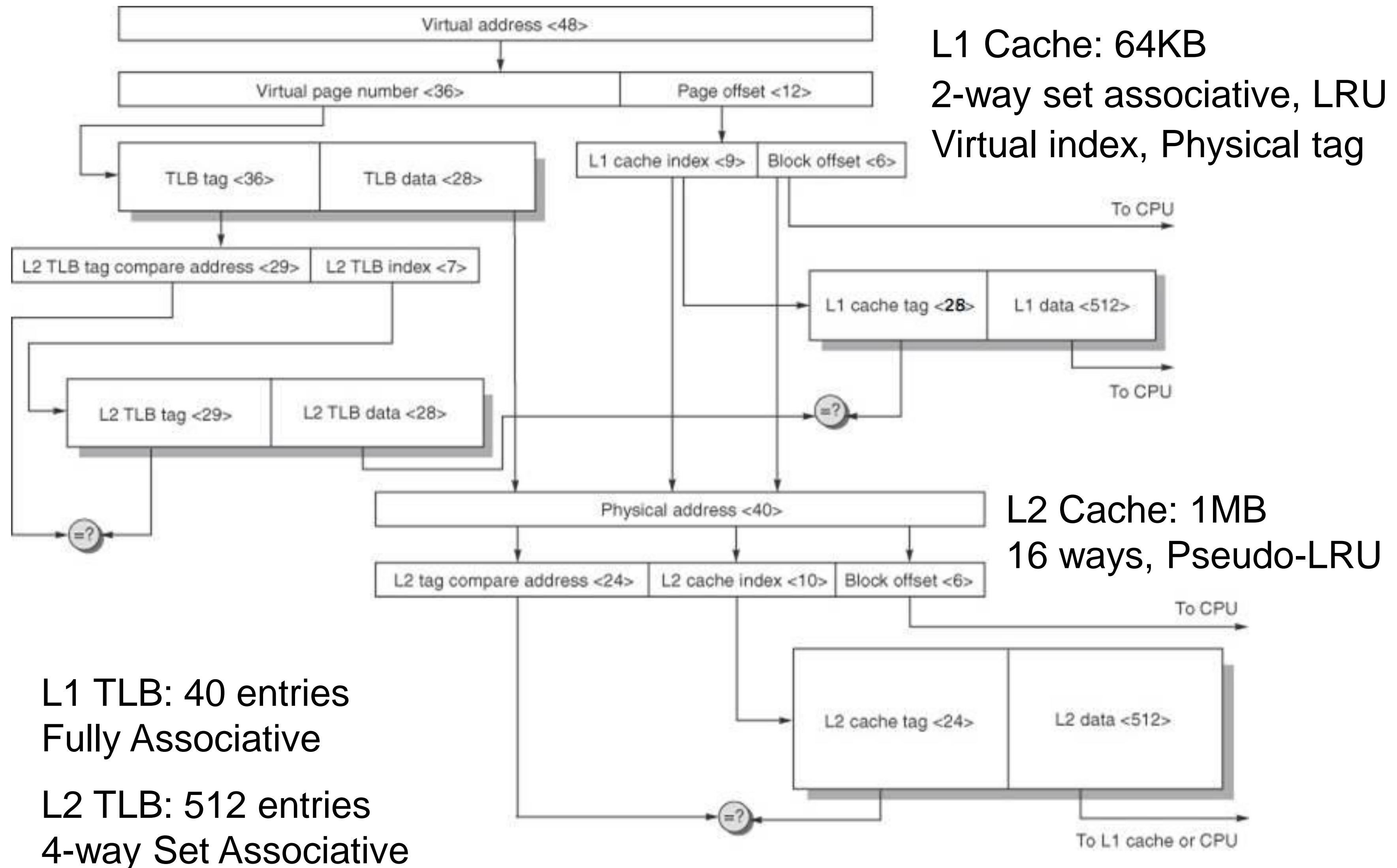
Page Offset is identical in virtual and physical addresses

If $(k + b) > p$ then Alias Problem ➔ Lower VPN bits used in cache index

Anti-Aliasing Techniques

- Increase the Page Size
  - Larger page ➔ larger cache physical index ➔ larger cache
  - Shared pages should be large and aligned to avoid alias problem
  - Requires OS and architectural support for large page size
- Increase Cache Associativity
  - Increase cache capacity, without changing index (or page size)
  - Maximum capacity of directly-mapped physical cache = page size
  - m-way set associative cache increases capacity by a factor of m
- Disallow aliases to coexist while processing a cache miss
  - if $(k + b) > p$ ➔ Cache is virtually indexed ➔ Alias problem
  - Examine L1 cache tags while processing a cache miss, if an alias is found (with same physical tag) then invalidate block

AMD Opteron TLB and Cache



AMD Opteron Memory Hierarchy

- Exclusion policy between L1 and L2
  - Block can exist in L1 or L2 but not in both
  - Both D-cache and L2 use write-back with write-allocate
- L1 cache is pipelined, latency is 2 clock cycles
- L1 TLB = 40 entries, L2 TLB = 512 entries
- L1 cache is virtually indexed and physically tagged
  - Lower 3-bit of Virtual Page Number (VPN) are used in L1 index
- On a L1 cache miss, controller checks for alias in L1
  - $8 = 2^3$ L1 cache tags per way are examined in parallel for an alias during an L2 cache lookup.
  - If an alias is found ➔ the offending block is invalidated