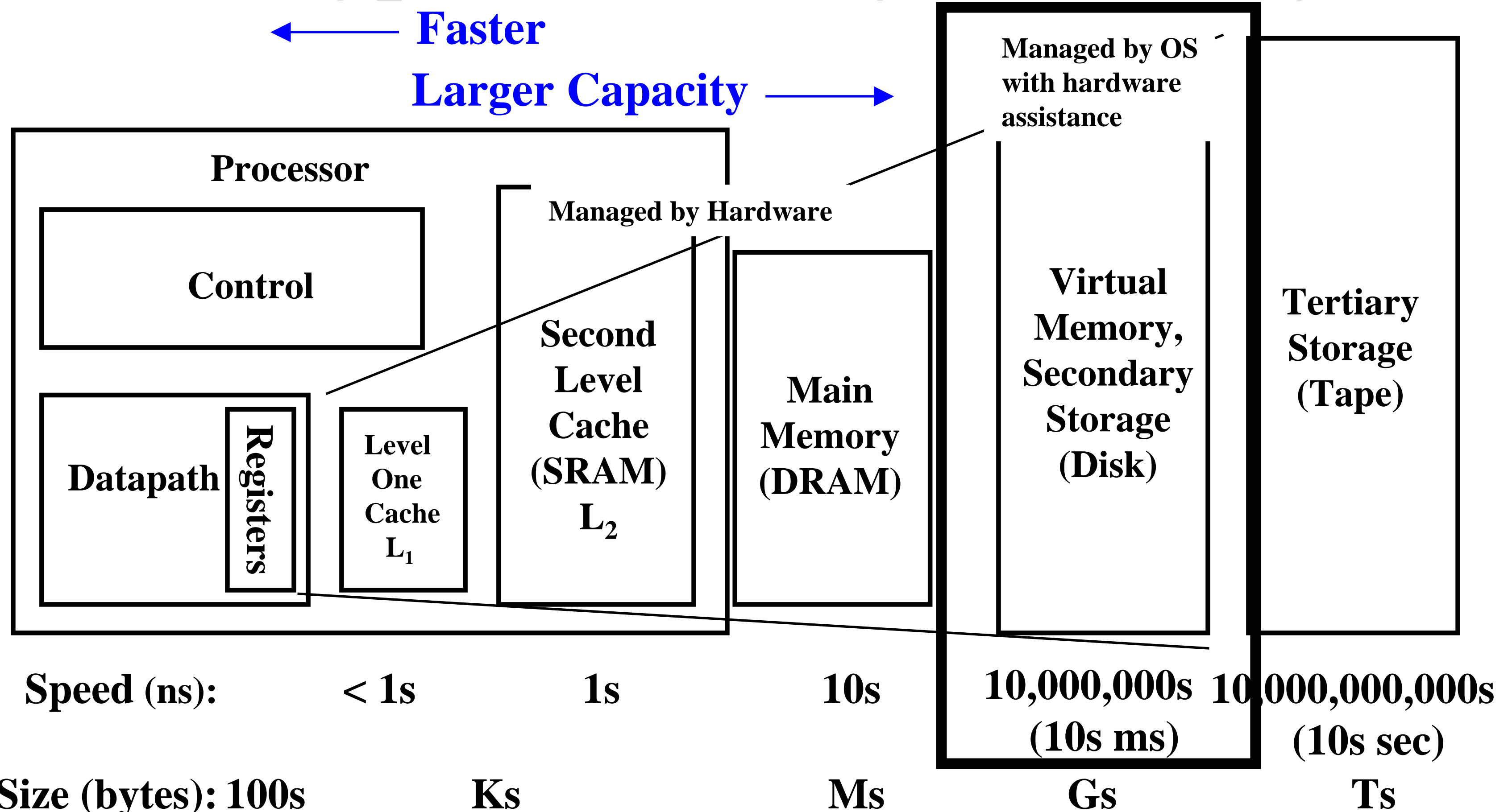
A Typical Memory Hierarchy



Virtual memory Sources:

Textbook Chapter 5.10, 5.11

Virtual memory: Issues of implementation, B. Jacob, and T. Mudge, Computer, vol. 31, no. 6, pp. 33-43. June 1998.
(Virtual Memory paper #1)

Virtual memory in contemporary microprocessors, B. Jacob, and T. Mudge, Micro, vol. 18, no. 4, pp. 60-75. July/Aug. 1998.
(Virtual Memory paper #2)

Virtual Memory

- Overview & Motivation
- Paging Versus Segmentation
- Basic Virtual Memory Management
- Virtual Memory Basic Strategies
- Virtual Address Translation: Direct (Basic) Page Tables
- Speeding-up Translations: *Translation Lookaside Buffer (TLB)*
  - TLB-Refill Mechanisms: *Hardware versus software TLB refill.*
- Global Vs. Per-process Virtual Address Space
- Data/Code Sharing in Virtual Memory Systems
- Address-Space Protection in Virtual Memory Systems
- Page Table Organizations and Page Table Walking
  1. Direct (Basic) Page Tables.
  2. Hierarchical (or Forward-Mapped) Page Tables
  3. Inverted/Hashed Page Tables
- Virtual Memory Architecture Examples
- The Hardware (MMU)/Software (OS) Mismatch In Virtual Memory

Chapter 5.10, 5.11, Two Ref. Papers

Virtual Memory: Overview

- Virtual memory controls two levels of the memory hierarchy:
  - Main memory (DRAM).
  - Mass storage (usually magnetic disks).
- Main memory is divided into blocks allocated to different running processes in the system by the OS:

  Superpages can be much larger

  - Fixed size blocks: Pages (size 4k to 64k bytes). (Most common)
  - Variable size blocks: Segments (largest size 2^{16} up to 2^{32}).
  - Paged segmentation: Large variable/fixed size segments divided into a number of fixed size pages (X86, PowerPC).
- At any given time, for any running process, a portion of its data/code is loaded (allocated) in main memory while the rest is available only in mass storage.
- A program code/data block needed for process execution and not present in main memory result in a page fault (address fault) and the page has to be loaded into main memory by the OS from disk (demand paging).
- A program can be run in any location in main memory or disk by using a relocation/mapping mechanism controlled by the operating system which maps (translates) the address from virtual address space (logical program address) to physical address space (main memory, disk).

Using page tables

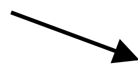
Virtual Memory: Motivation

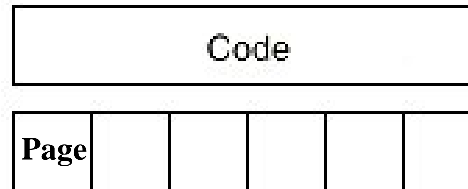
- **Original Motivation:**
  - **Illusion of having more physical main memory (using demand paging)**
  - **Allows program and data address relocation by automating the process of code and data movement between main memory and secondary storage.** Demand paging
- **Additional Current Motivation:**
  - **Fast process start-up.**
  - **Protection from illegal memory access.**
    - *Needed for multi-tasking operating systems.*
  - **Controlled code and data sharing among processes.**
    - *Needed for multi-threaded programs.*
  - **Uniform data access**
    - **Memory-mapped files**
    - **Memory-mapped network communication**

e.g local vs. remote memory access

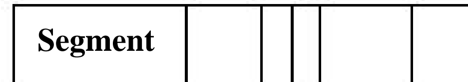
Paging Versus Segmentation

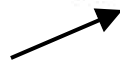
Fixed-size blocks (pages)





Segmentation





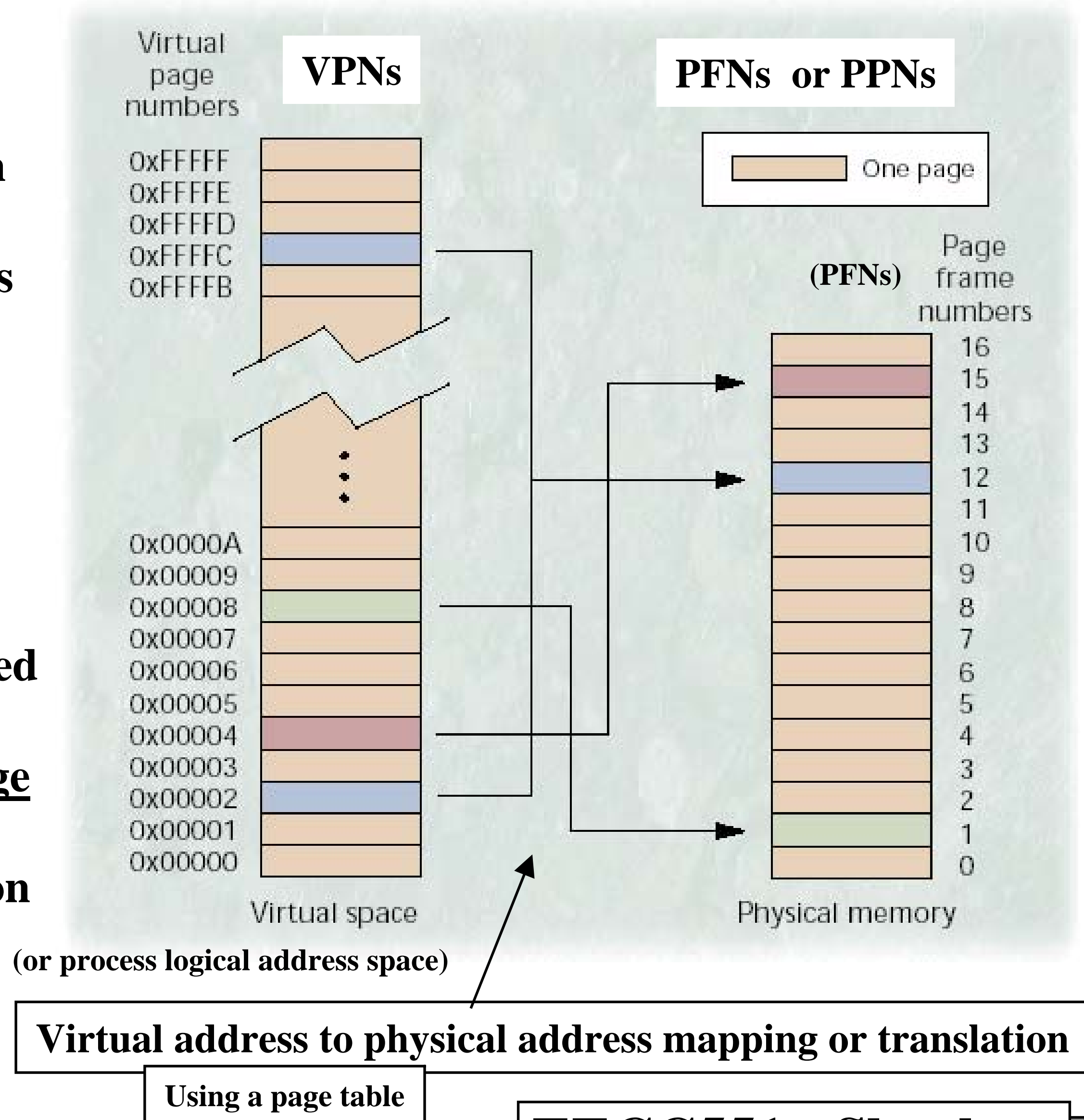
Variable-size blocks (segments)

	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial (all blocks are the same size)	Hard (must find contiguous, variable-size, unused portion of main memory)
Memory use inefficiency	Internal fragmentation (unused portion of page)	External fragmentation (unused pieces of main memory)
Efficient disk traffic	Yes (adjust page size to balance access time and transfer time)	Not always (small segments may transfer just a few bytes)

Virtual Address Space Vs. Physical Address Space

Virtual memory stores only the most often used portions of a process address space in main memory and retrieves other portions from a disk as needed (demand paging).

The virtual-memory space is divided into pages identified by virtual page numbers (VPNs), shown on the far left, which are mapped to page frames or physical page numbers (PPNs) or page frame numbers (PFNs), in physical memory as shown on the right.



(or process logical address space)

Virtual address to physical address mapping or translation

Using a page table

Paging is assumed here

Virtual Address Space = Process Logical Address Space

Basic Virtual Memory Management

- Operating system makes decisions regarding which virtual (logical) pages of a process should be allocated in real physical memory and where (demand paging) assisted with hardware Memory Management Unit (MMU)
- On memory access -- If no valid virtual page to physical page translation (i.e page not allocated in main memory)
  - Page fault to operating system (e.g system call to handle page fault))
  - Operating system requests page from disk
  - Operating system chooses page for replacement
    - writes back to disk if modified
  - Operating system allocates a page in physical memory and updates page table w/ new page table entry (PTE).

Then restart faulting process

Paging is assumed

Typical Parameter Range For Cache & Virtual Memory

Parameter	First-level cache	Virtual memory
Block (page) size	16–128 bytes	4096–65,536 bytes
Hit time	1–2 clock cycles	40–100 clock cycles
Miss penalty M	8–100 clock cycles	700,000–6,000,000 clock cycles
(Access time)	(6–60 clock cycles)	(500,000–4,000,000 clock cycles)
(Transfer time)	(2–40 clock cycles)	(200,000–2,000,000 clock cycles)
Miss rate	0.5–10%	0.00001– 0.001%
Data memory size	0.016–1MB	16–8192 MB

i.e page fault

Program assumed in steady state

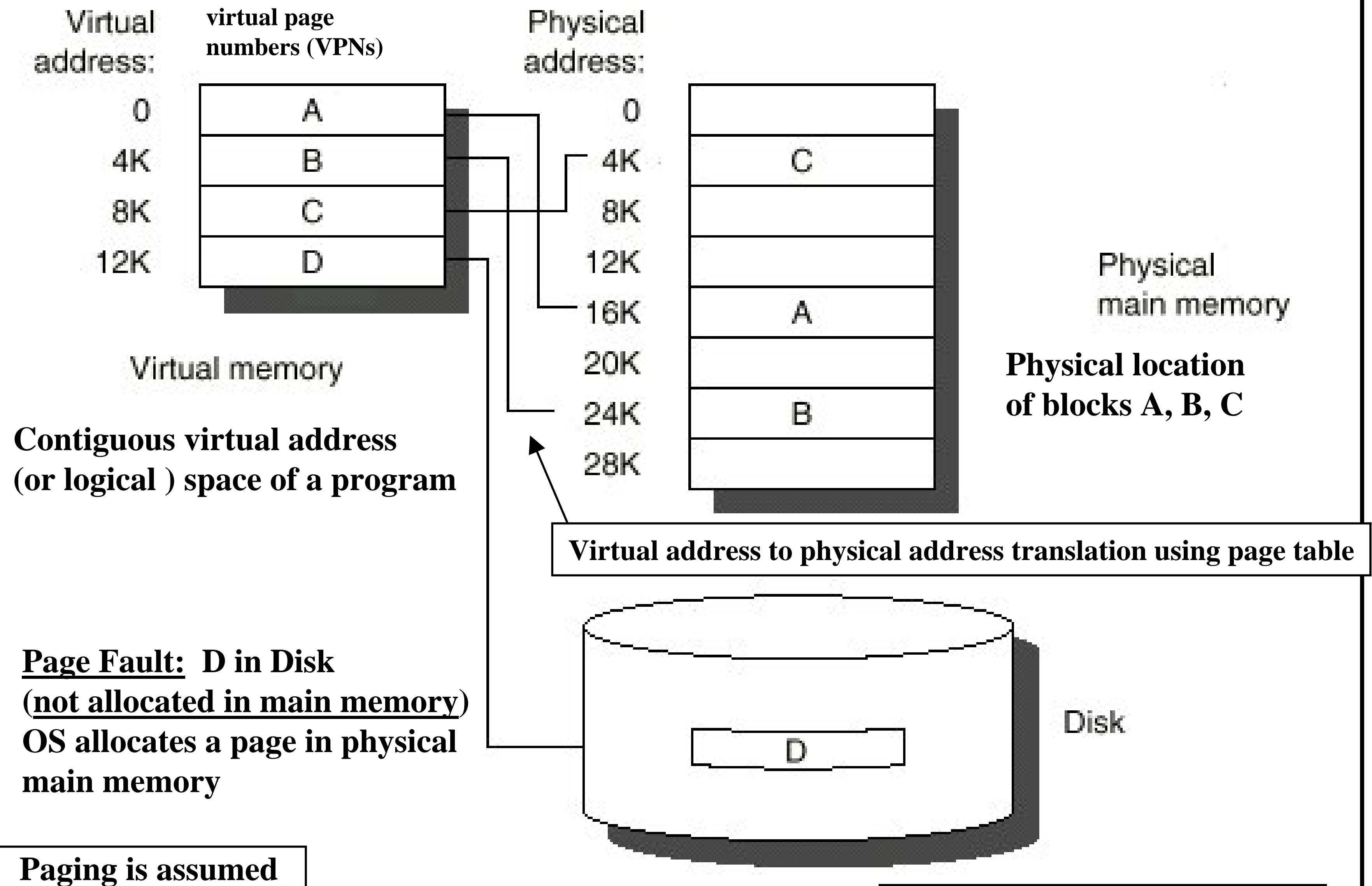
Paging is assumed here

Virtual Memory Basic Strategies

- Main memory page placement(allocation): Fully associative placement or allocation (by OS) is used to lower the miss rate.
- Page replacement: The least recently used (LRU) page is replaced when a new page is brought into main memory from disk.
- Write strategy: Write back is used and only those pages changed in main memory are written to disk (dirty bit scheme is used).
- Page Identification and address translation: To locate pages in main memory a page table is utilized to translate from virtual page numbers (VPNs) to physical page numbers (PPNs) . The page table is indexed by the virtual page number and contains the physical address of the page.
  - In paging: Offset is concatenated to this physical page address.
  - In segmentation: Offset is added to the physical segment address.
- Utilizing address translation locality, a translation look-aside buffer (TLB) is usually used to cache recent address translations (PTEs) and prevent a second memory access to read the page table.

PTE = Page Table Entry

Virtual → Physical Address Translation

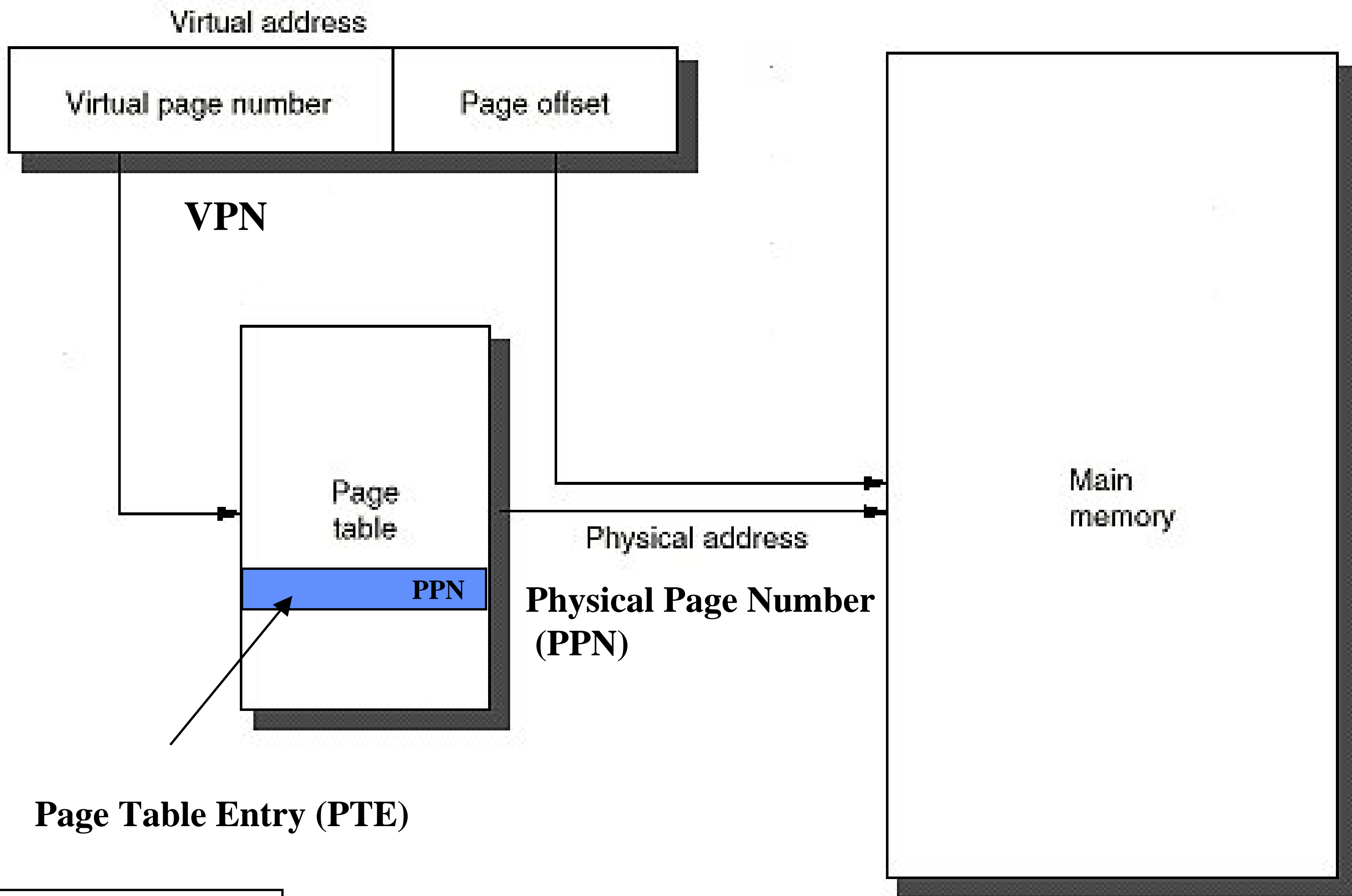


Paging is assumed

Virtual to Physical Address Translation: Page Tables

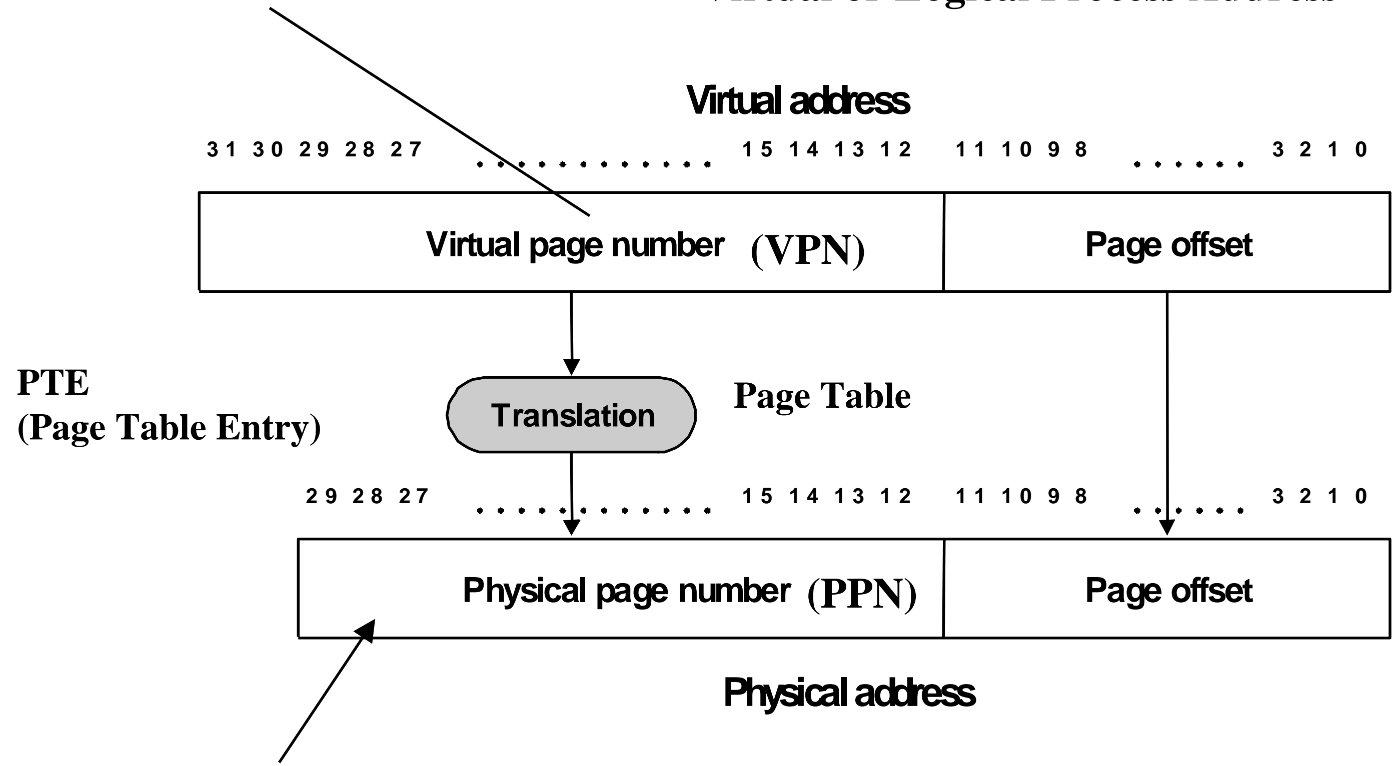
- **Mapping information from virtual page numbers (VPNs) to physical page numbers is organized into a page table which is a collection of page table entries (PTEs).**
- **At the minimum, a PTE indicates whether its virtual page is in memory, on disk, or unallocated and the PPN (or PFN) if the page is allocated.**
- **Over time, virtual memory evolved to handle additional functions including data sharing, address-space protection and page level protection, so a typical PTE now contains additional information including:**
  - **A *valid* bit, which indicates whether the PTE contains a valid translation;**
  - **The page's location in memory (page frame number, PFN) or location on disk (for example, an offset into a swap file);**
  - **The ID of the page's owner (the *address-space identifier (ASID)*, sometimes called Address Space Number (ASN) or *access key*;**
  - **The virtual page number (VPN);**
  - **A *reference* bit, which indicates whether the page was recently accessed;**
  - **A *modify* bit, which indicates whether the page was recently written; and**
  - ***Page-protection* bits, such as read-write, read only, kernel vs. user, and so on.**

Basic Mapping Virtual Addresses to Physical Addresses Using A Direct Page Table



Paging is assumed

Virtual to Physical Address Translation



physical page numbers (PPN) or page frame numbers (PFN)

Paging is assumed

Virtual Memory Terms

- **<u>Page Table Walking:</u>** The process of searching the page table for the translation PTE. Done by: Software (OS), Hardware (Finite State Machine)
- **<u>Allocated or Mapped Virtual Page:</u>** The OS has mapping information on its location (in memory or on disk) using its PTE in the page table.
- **<u>Unmapped Virtual Page:</u>** A page that either not yet been allocated or has been deallocated and its mapping information (PTE) has been discarded.
- **<u>Wired down virtual page:</u>** A virtual page for which space is always allocated in physical memory and <u>not allowed to be paged out to disk</u>.
- **<u>Virtual Address Aliasing:</u>** Mapping of two or more virtual pages to the same physical page to allow processes or threads to <u>share</u> memory
  - Provides threads with different "views" of data with different protections
- **<u>Superpages:</u>** A superpage contains a number of contiguous physical memory pages but require a single address translation. A number of virtual memory architectures currently support superpages.
- **<u>Memory Management Unit (MMU):</u>** Hardware mechanisms and structures to aid the operating system in virtual memory management including address generation/translation, sharing, protection. Special OS privileged ISA instructions provide software/OS access to this support. (e.g. TLBs, special protected registers)

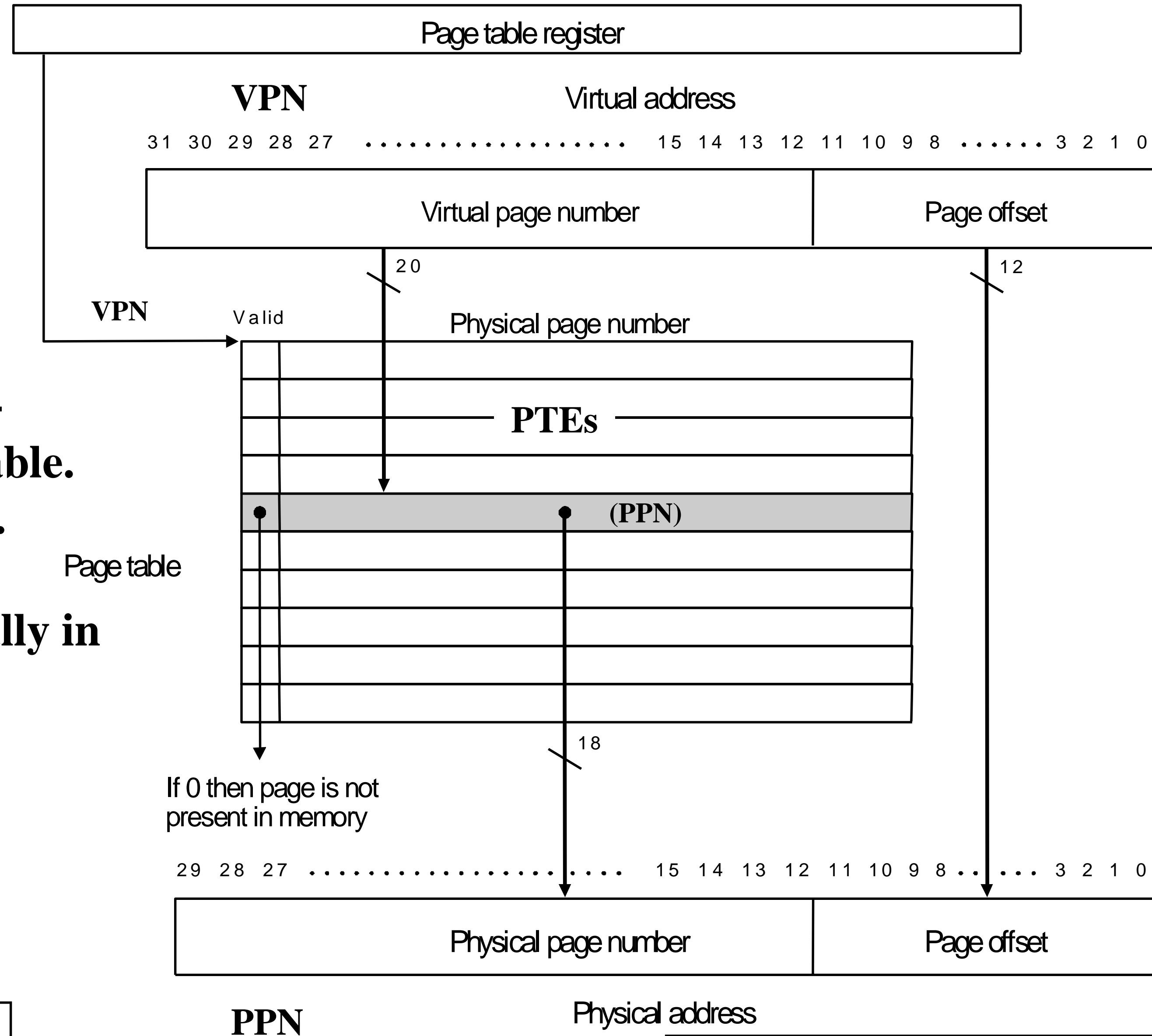
Page Table Organizations

Direct (Basic) Page Table: (Single-level)

- **When address spaces were much smaller, a single-level table—called a *direct table* mapped the entire virtual address space and was small enough to be contained in SRAM and maintained entirely in hardware (hardware page table walking).**
- **As address spaces grew larger, the table size grew to the point that system designers were forced to move it into main memory.**

- **Limitations:**
  - **Translation requires a main memory access:**
    - Solution: Speedup translation by caching recently used PTEs in **a Translation Lookaside Buffer (TLB).**
  - **Large size of direct table:**
    - Example: A 32 bit virtual address with 2^{12} = 4k byte pages and 4 byte PTE entries requires a direct page table with 2^{20} = 1M PTEs and occupies 4M bytes in memory.
    - **Solution: Alternative page table organizations:**
      - **Hierarchical (Forward-Mapped) page tables**
      - **Inverted or hashed page tables**

Direct Page Table Organization



Two memory accesses needed:

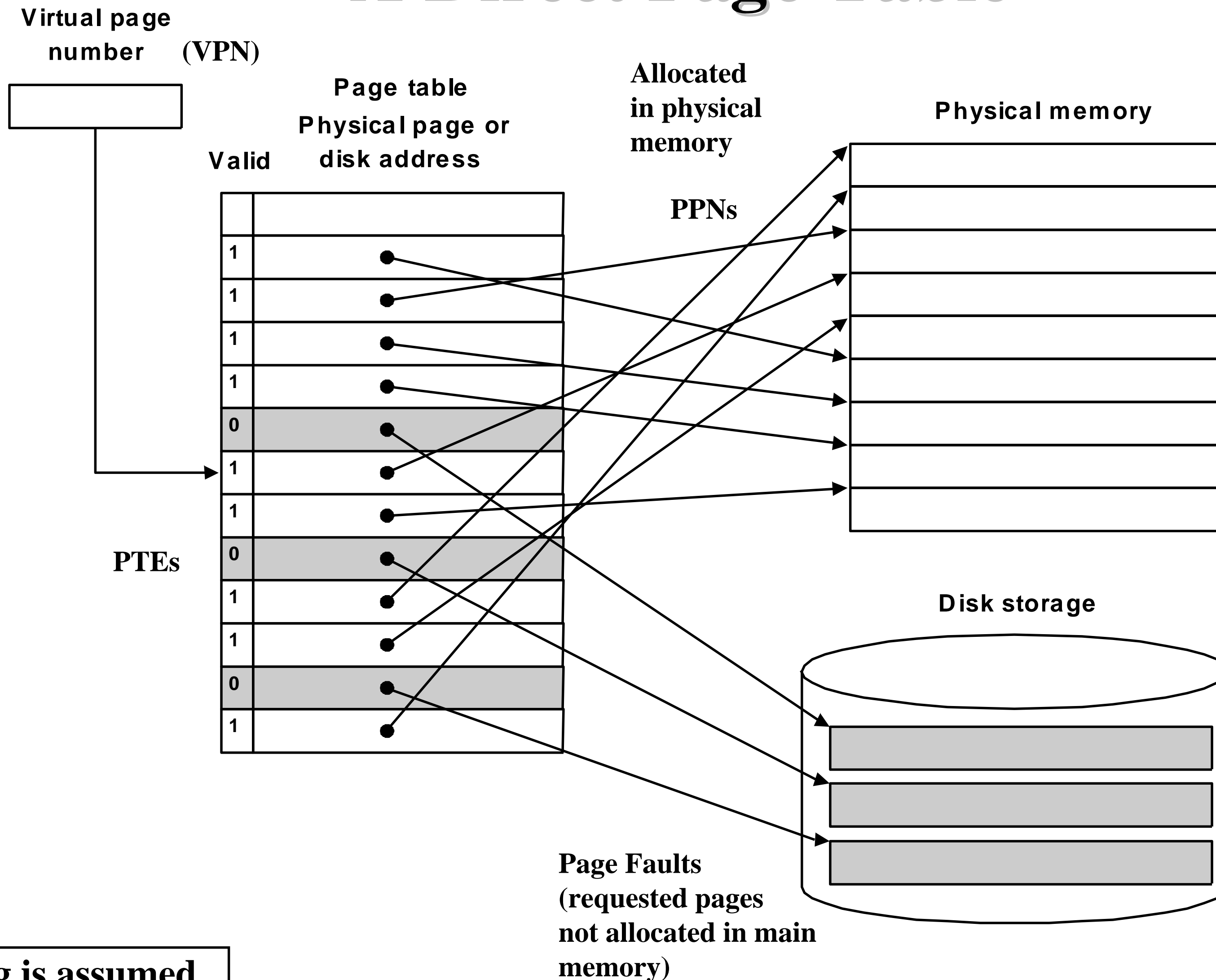
- **First to page table.**
- **Second to item.**

•Page table usually in main memory.

How to speedup virtual to physical address translation?

Paging is assumed

Virtual Address Translation Using A Direct Page Table



Paging is assumed

Speeding Up Address Translation:

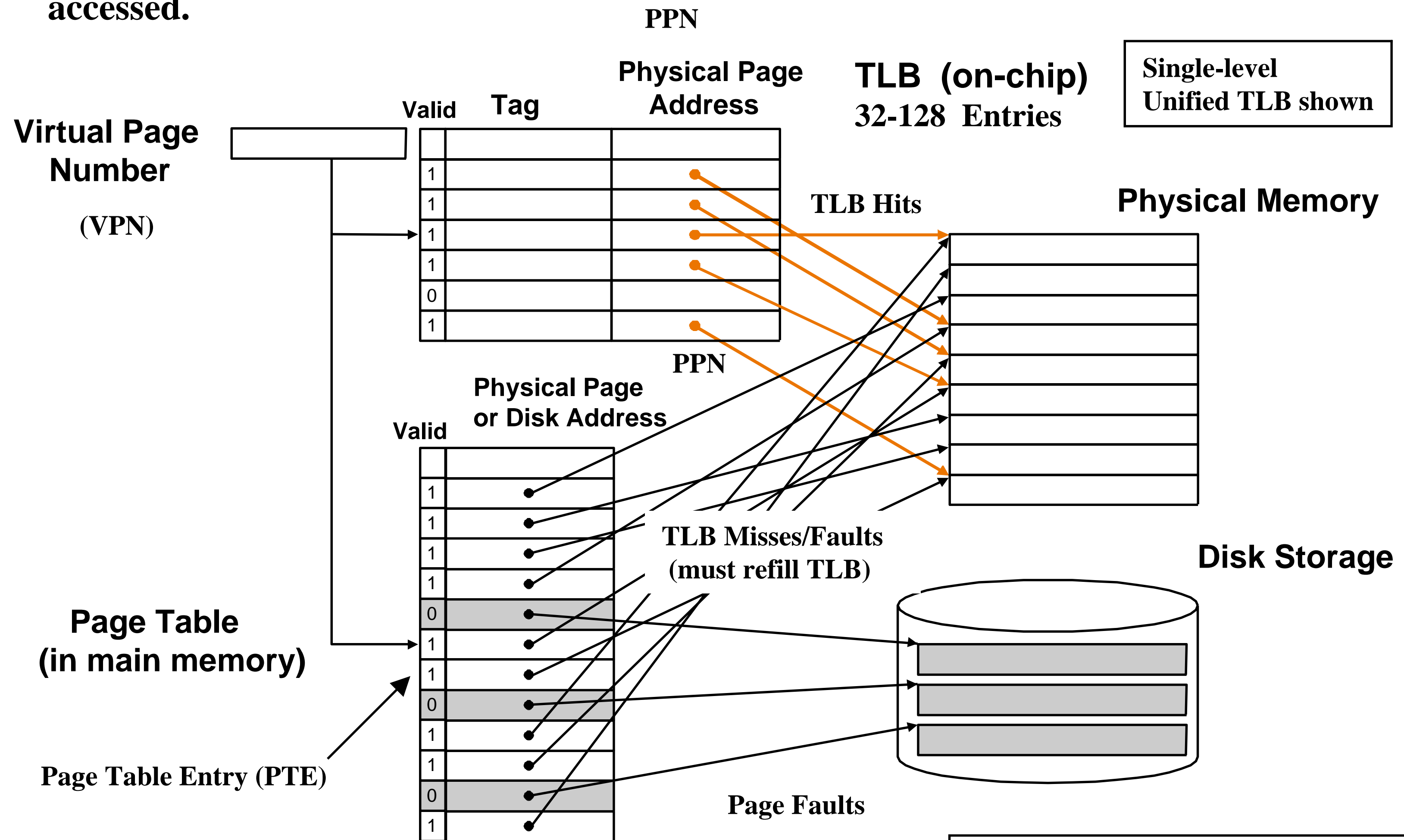
Translation Lookaside Buffer (TLB)

- **Translation Lookaside Buffer (TLB) : Utilizing address reference temporal locality, a small on-chip cache used for address translations (PTEs).** **i.e. recently used PTEs**
  - TLB entries usually 32-128
  - High degree of associativity usually used
  - Separate instruction TLB (I-TLB) and data TLB (D-TLB) are usually used.
  - A unified larger second level TLB is often used to improve TLB performance and reduce the associativity of level 1 TLBs.
- **If a virtual address is found in TLB (a TLB hit), the page table in main memory is not accessed.**
- **TLB-Refill: If a virtual address is not found in TLB, a TLB miss (TLB fault) occurs and the system must search (walk) the page table for the appropriate entry and place it into the TLB this is accomplished by the TLB-refill mechanism .**
- **Types of TLB-refill mechanisms:**
  - **Fast but not flexible** — **Hardware-managed TLB:** A hardware finite state machine is used to refill the TLB on a TLB miss by walking the page table. (PowerPC, IA-32)
  - **Flexible but slower** — **Software-managed TLB:** TLB refill handled by the operating system. (MIPS, Alpha, UltraSPARC, HP PA-RISC, …)

Speeding Up Address Translation:

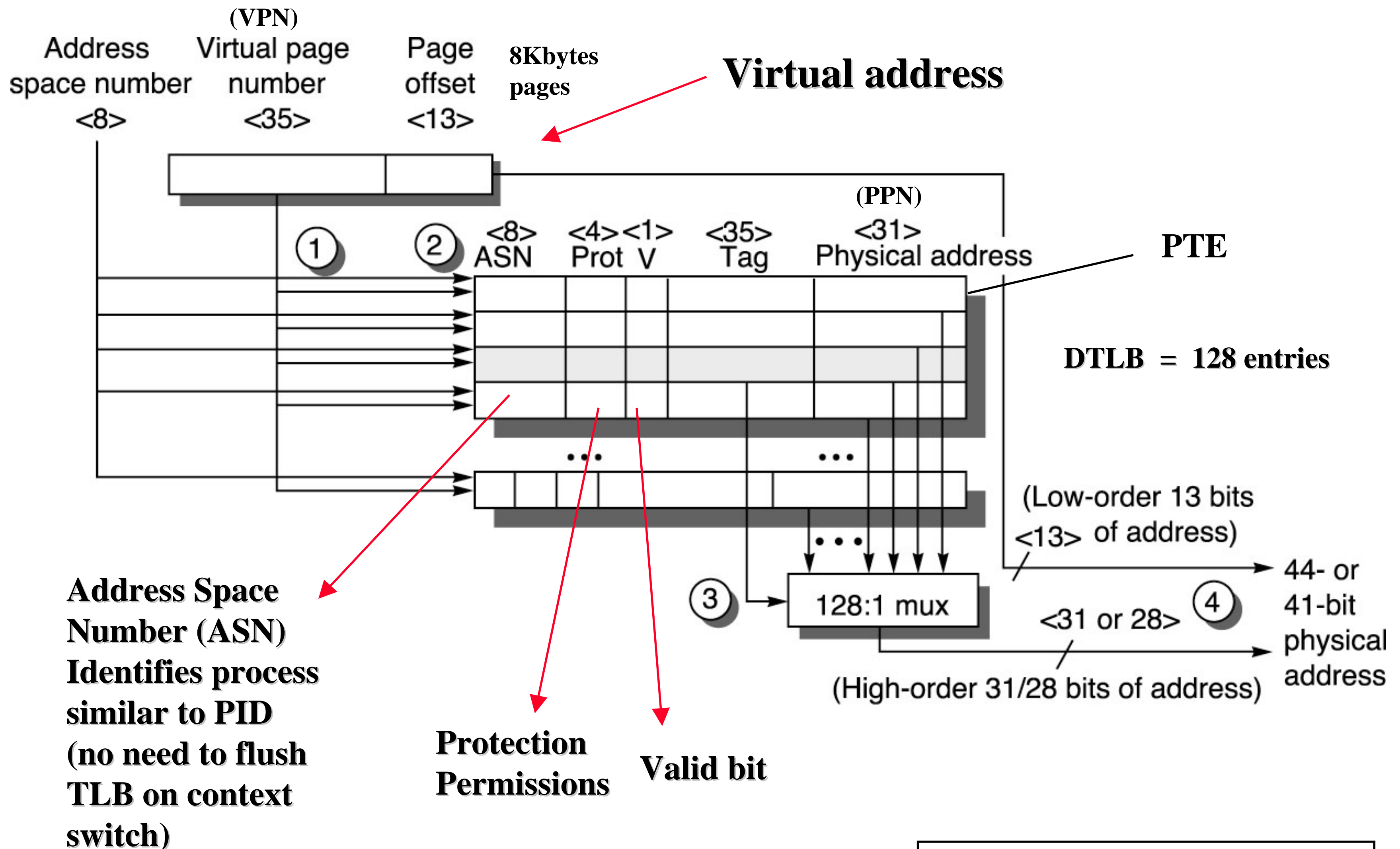
Translation Lookaside Buffer (TLB)

- TLB: A small on-chip cache that contains recent address translations (PTEs).
- If a virtual address is found in TLB (a TLB hit), the page table in main memory is not accessed.



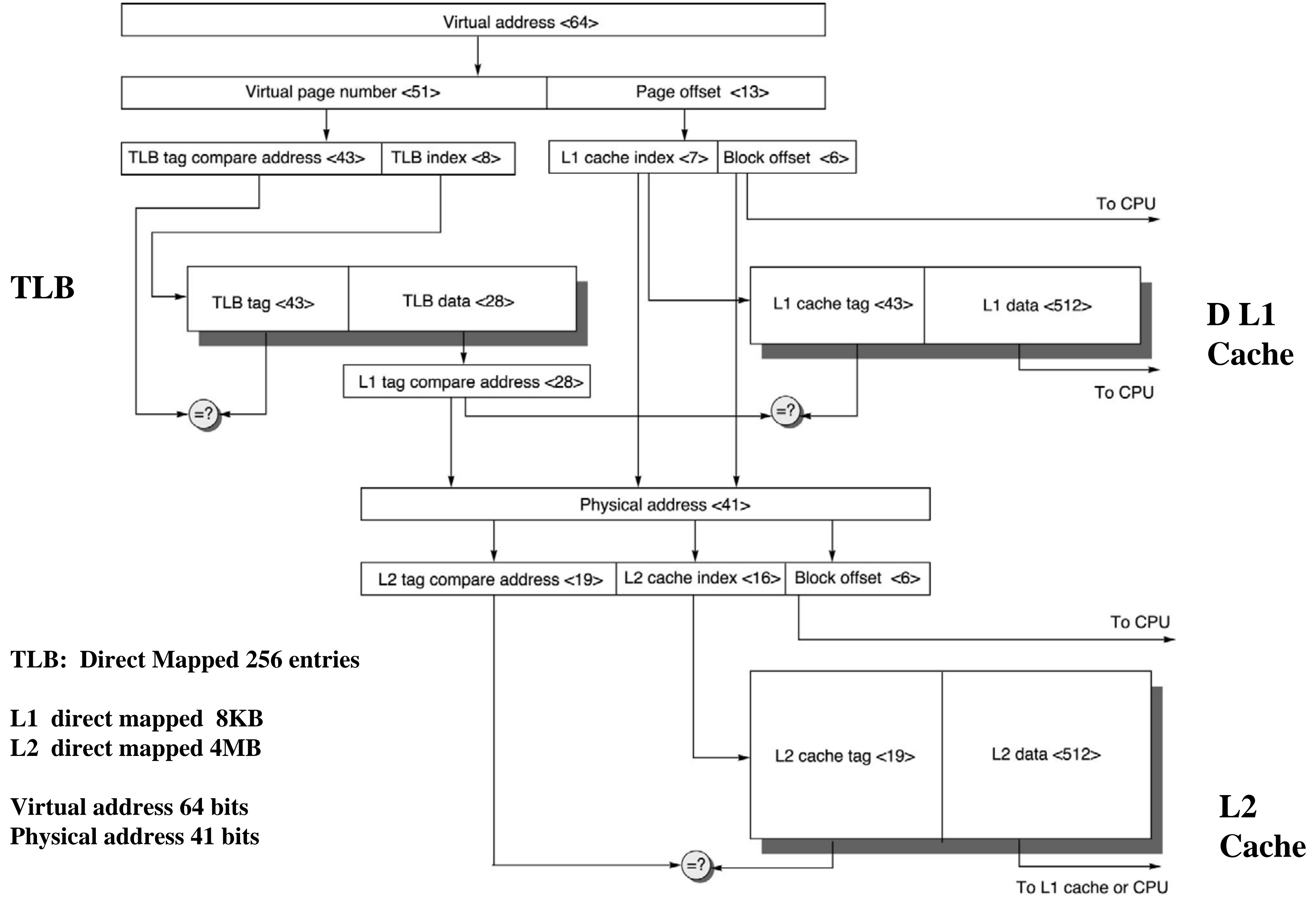
Paging is assumed

Operation of The Alpha 21264 Data TLB (DTLB) During Address Translation



PID = Process ID

A Memory Hierarchy Example: TLB & Two Levels of Cache



TLB: Direct Mapped 256 entries

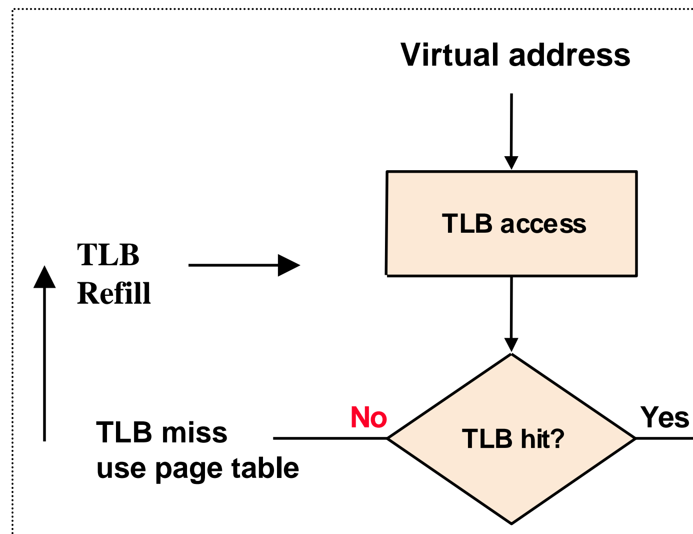
L1 direct mapped 8KB
L2 direct mapped 4MB

Virtual address 64 bits
Physical address 41 bits

<u>Alpha 21164:</u>
L1 Data Cache: Virtually Indexed, Physically Tagged

Basic TLB & Cache Operation

TLB Operation



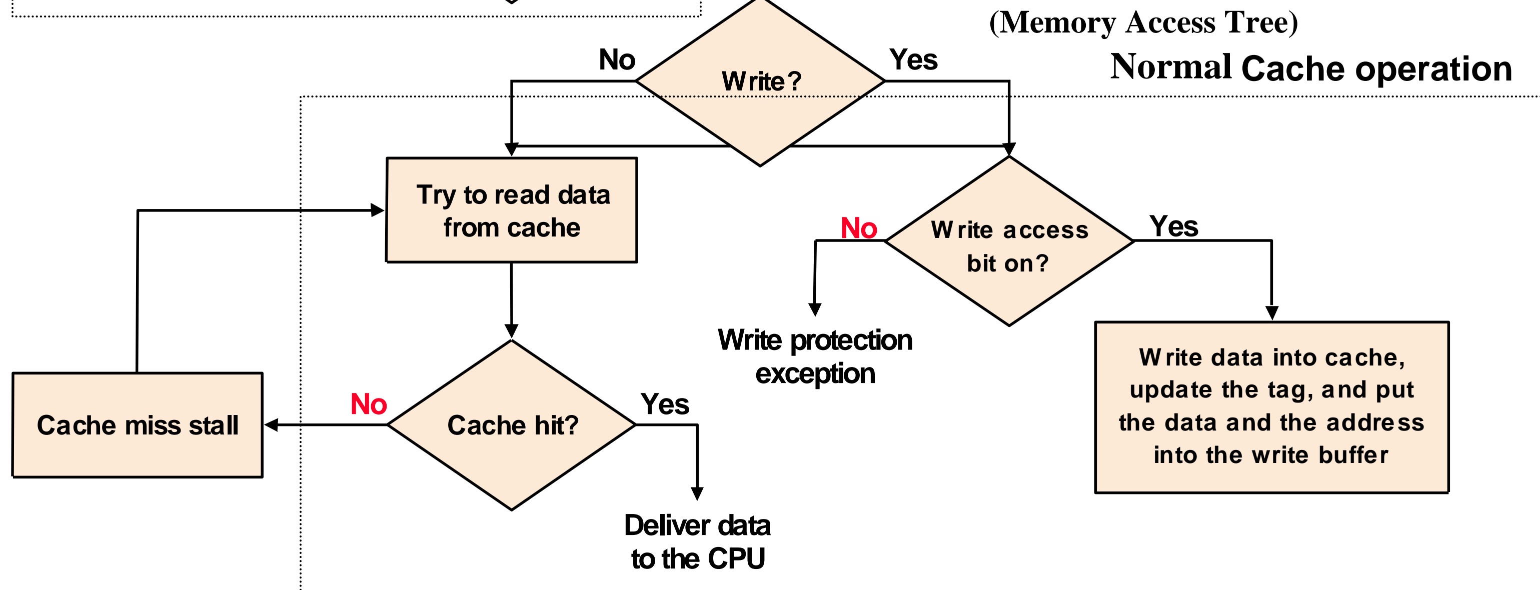
TLB access	TAG check	L1 DATA		
	TLB access	TAG check	L1 DATA	
		TLB access	TAG check	L1 DATA

Cache is usually physically-addressed

Physical address

(Memory Access Tree)

Normal Cache operation



CPU Performance with Real TLBs

When a real TLB is used with a TLB miss rate and a TLB miss penalty (time needed to refill the TLB) is used:

$$CPI = CPI_{execution} + \text{mem stalls per instruction} + \text{TLB stalls per instruction}$$

Where:

Mem Stalls per instruction = Mem accesses per instruction x mem stalls per access

Similarly:

1 + fraction of loads and stores

TLB Stalls per instruction = Mem accesses per instruction x TLB stalls per access

TLB stalls per access = TLB miss rate x TLB miss penalty

(For unified single-level TLB)

Example:

Given: $CPI_{execution}$ **= 1.3 Mem accesses per instruction = 1.4**

Mem stalls per access = .5 TLB miss rate = .3% TLB miss penalty = 30 cycles

What is the resulting CPU CPI?

Mem Stalls per instruction = 1.4 x .5 = .7 cycles/instruction

TLB stalls per instruction = 1.4 x (TLB miss rate x TLB miss penalty)

= 1.4 x .003 x 30 = .126 cycles/instruction

CPI = 1. 3 + .7 + .126 = 2.126

Event Combinations of Cache, TLB, Virtual Memory

Cache	TLB	Virtual Memory	Possible? When?
Hit	Hit	Hit	TLB/Cache Hit
Miss	Hit	Hit	Possible, no need to check page table
Hit	Miss	Hit	TLB miss, found in page table
Miss	Miss	Hit	TLB miss, cache miss
Miss	Miss	Miss	Page fault
Miss	Hit	Miss	Impossible, cannot be in TLB if not in main memory
Hit	Hit	Miss	Impossible, cannot be in TLB or cache if not in main memory
Hit	Miss	Miss	Impossible, cannot be in cache if not in memory

TLB-Refill Mechanisms

(To handle TLB misses or faults)

1- Hardware-managed TLB (ex. PowerPC, Intel IA-32):

- Typical of early memory-management units (MMUs).
- A hardware state machine is used to refill the TLB.
- In the event of a TLB miss, the hardware state machine would walk the page table, locate the mapping, insert it into the TLB, and restart the computation.
- Advantage: Performance
  - Disturbs the processor pipeline only slightly. When the state machine handles a TLB miss, the processor stalls faulting instructions only. Compared to taking an interrupt, the contents of the pipeline are unaffected, and the reorder buffer need not be flushed.
- Disadvantage: Inflexibility of Page table organization design
  - The page table organization is effectively fixed in the hardware design; the operating system has no flexibility in choosing a design.

i.e. Page table walking in hardware

TLB-Refill Mechanisms

2- Software-managed TLB: (ex. MIPS, Alpha, UltraSPARC, HP PA-RISC...)

- Typical of recent memory-management units (MMUs). No hardware TLB-refill finite state machine to handle TLB misses.
- On a TLB miss, the hardware interrupts the operating system and vectors to an OS software routine that walks the page table and refills the TLB.
- Advantage: Flexibility of Page table organization design
  - The page table can be defined entirely by the operating system, since hardware never directly manages the table.
- Disavdantage: Performance cost.
  - The TLB miss handler that walks the page table is an operating system primitive (call) which usually requires 10 to 100 instructions
  - If the handler code is not in the instruction cache (I cache miss) at the time of the TLB miss exception, the time to handle the miss can be much longer than in the hardware walked scheme.
  - In addition, the use of precise exception handling mechanisms adds to the cost by flushing the pipeline, removing a possibly large number of instructions from the reorder buffer. This can add hundreds of cycles to the overhead of walking the page table by software.

i.e. Page table walking in software (OS)

Virtual Memory Architectures:

Global Vs. Per-process Virtual Address Space

- **Per-process virtual address space:**
  - The effective or logical virtual address generated by a process is extended by an address-space identifier (ASID) forming a per-process virtual address and is included in TLB and page table entries (PTEs) to distinguish between processes or contexts.
  - Each process may have a separate page table to handle address translation.
  - e.g MIPS, Alpha, PA-RISC, UltraSPARC.
- **Global system-wide virtual address space:**
  - The effective or logical virtual address generated by a process is extended by a segment number forming a global, flat or extended global virtual address. (paged segmentation)
  - Usually a number of segment registers specify the segments assigned to a process. (OS protected)
  - A global page translation table may be used for all processes running on the system.
  - e.g IA-32 (x86), PowerPC.

Data/Code Sharing in Virtual Memory Systems

- **Shared memory allows multiple processes to reference the same physical code and data possibly using different virtual addresses**

1 **Global sharing of data can be accomplished by a global access bit in TLBs, PTEs.**
  - **For per-process virtual address space using address-space identifiers (ASIDs), the hardware ASID match check is disabled.**

2 **Sharing at the page level is accomplished by virtual address aliasing, where two or more virtual pages are mapped the same physical page with possibly different protections.**
  - **Disadvantage: Increases overheads of updating multiple PTEs every time the OS changes a page's physical location (page reallocation).**
  - **Used in Unix-based OSs.**

3 **In systems that support a global virtual address (using paged segmentation), sharing at the segment level can be accomplished by assigning two or more processes the same segment number.**

Address-Space Protection in Virtual Memory Systems

- **Per-process virtual address space systems using ASIDs:**
  - The OS places the running process's address-space identifier(ASID) in a protected register, and every virtual address the process generates is concatenated with the address-space identifier.
  - Each process is unable to produce addresses that mimic those of other processes, because to do so it must control the contents of the protected register (OS access only) holding the active ASID.
- **Global virtual address space using using paged segmentation:**
  - A process address space is usually composed of many segments, the OS maintains a set of segment identifiers for each process.
  - The hardware can provide protected registers to hold the process's segment identifiers, and if those registers can be modified only by the OS, the segmentation mechanism also provides address-space protection.

Alternative Page Table Organizations: Hierarchical Page Tables

- **Partition the page table into two or more levels:**
  - Based on the idea that a large data array can be mapped by a smaller array, which can in turn be mapped by an even smaller array.
  - For example, the DEC Alpha supports a four-tiered hierarchical page table composed of Level-0, Level-1, Level-2, and Level-3 tables.
- Highest level(s) typically locked (wired down) in physical memory (i.e physically addressed)
  - Not all lower level tables have to be resident in physical memory or even have to initially exist.
- Hierarchical page table Walking (access or search) Methods:
  - Top-down traversal (e.g IA-32)
    - Hardware managed TLB/page walking usually used
  - Bottom-up traversal (e.g MIPS, Alpha)
    - Software managed TLB/page walking usually used

Sometimes also called: Forward-Mapped Page Tables

Example:
A Two-Level Hierarchical Page Table

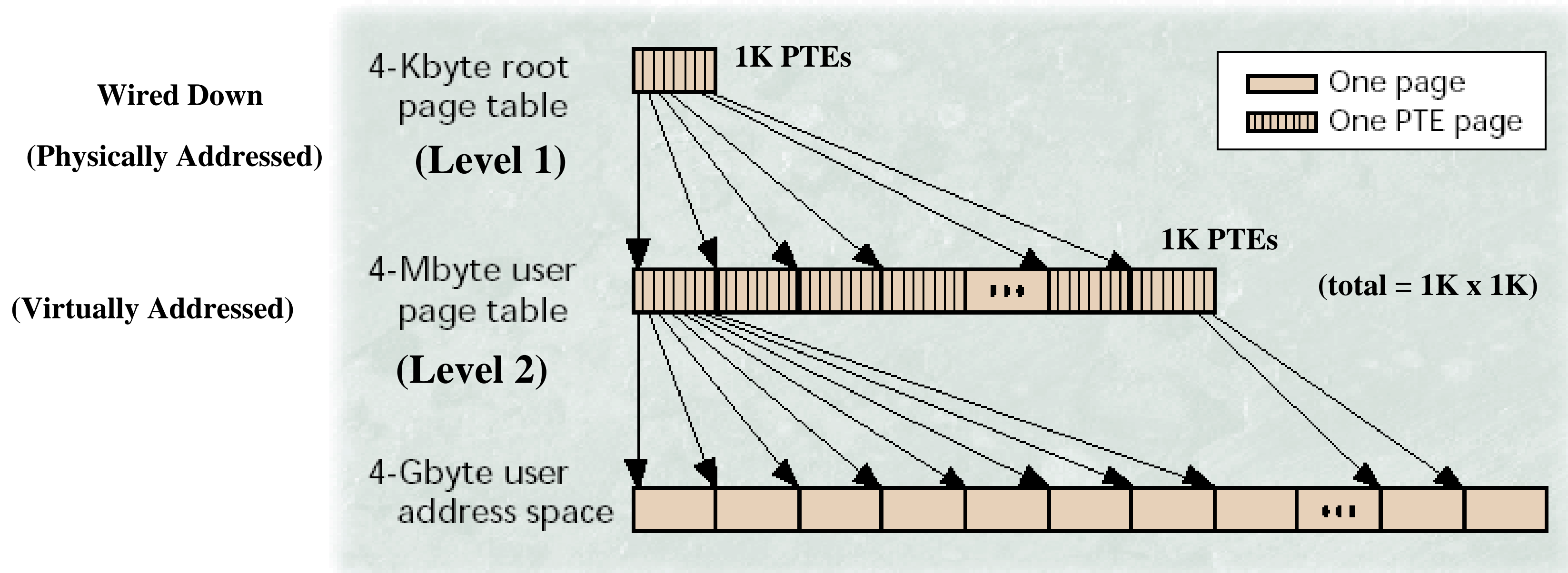
- **Assume 32-bit virtual addresses, byte addressing, and 4-Kbyte pages, the 4-Gbyte address space is composed of 1,048,576 (2^{20}) pages.**
- **If each of these pages is mapped by a 4-byte PTE, we can organize the 2^{20} PTEs into a 4-Mbyte linear structure composed of 1,024 (2^{10}) pages, which can be mapped by a first level or root table with 1,024 PTEs.**

  i.e each first level PTE map to a block of 1024 PTEs in second level
- **Organized into a linear array, the first level table with 1,024 PTEs occupy 4 Kbytes.**
  - **Since 4 Kbytes is a fairly small amount of memory, <u>the OS wires down this root-level table</u> in memory while the process is running (physically addressed).**
  - **Not all the lower page level (level two here also, referred to as the user page table) have to be resident in physical memory or even have to initially exist (virtually addressed).**
  - **As, shown on next page.**

Example:

A Two-Level Hierarchical Page table

32-bit 4-GByte virtual addresses, 4-Kbyte pages, 4-byte PTEs

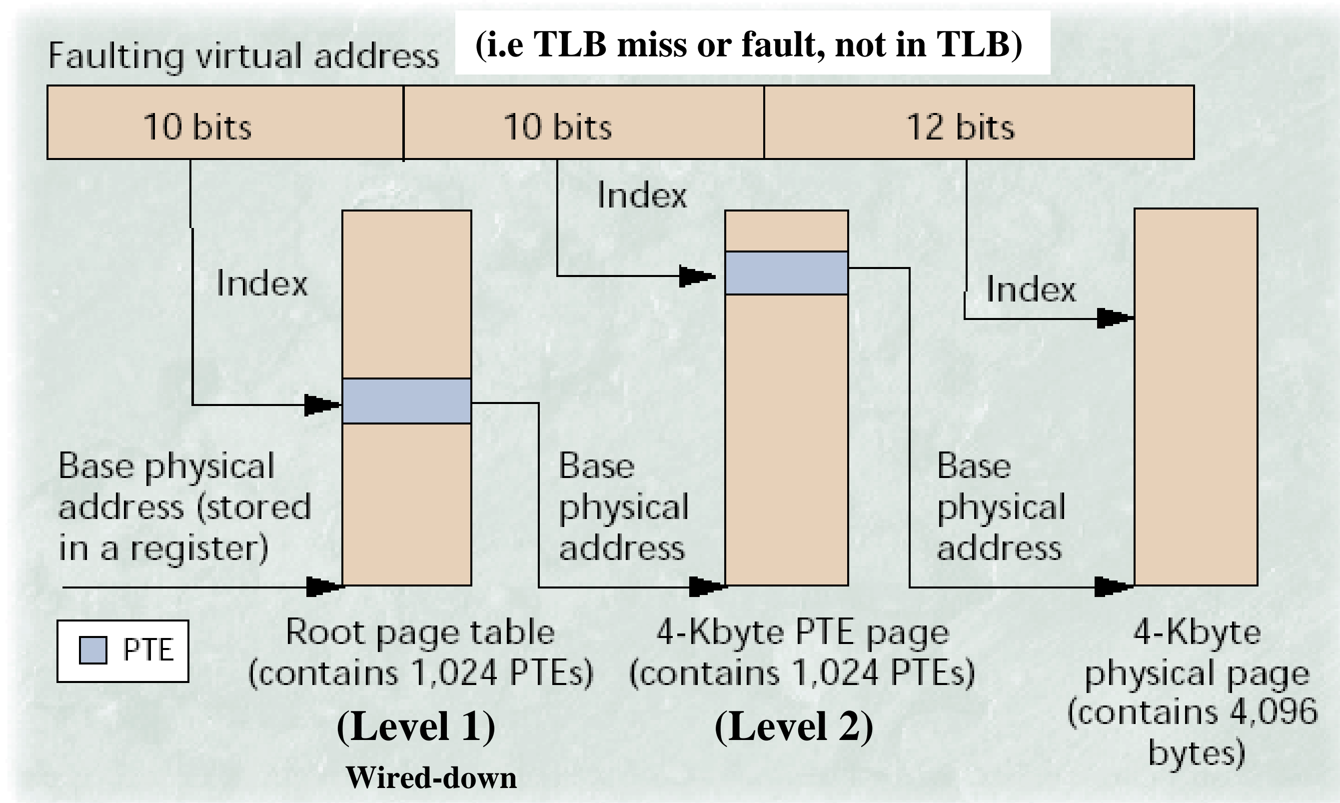


Typically, the root page table is <u>wired down</u> in the physical memory while the process is running.
The user page table (lowest level table, level 2 here) is paged in and out of physical main memory as needed.

Hierarchical Page Table Walking Methods

Top-down traversal or walking: (e.g IA-32)

- **Example for the previous two-level tables:**



Disadvantage: The top-down page table walking method requires as many memory references as there are table levels.

Hardware managed TLB/page walking usually used

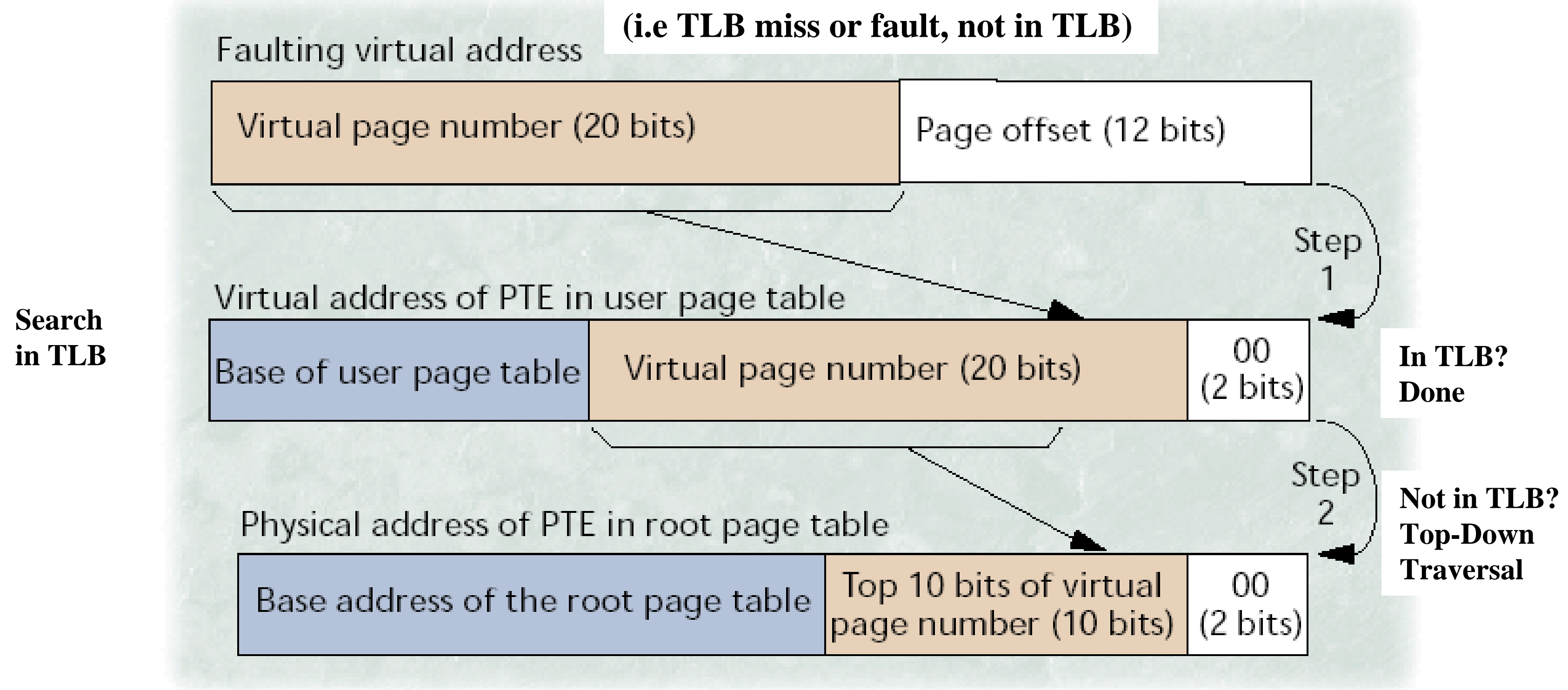
Hierarchical Page Tables Walking Methods

Bottom-up traversal or table walking (e.g MIPS, Alpha)

- **A bottom-up traversal lowers memory access overhead and typically accesses memory only once to translate a virtual address.**
- **For the previous two-level tables example:**
- **Step 1:**
  - The top 20 bits (virtual page number) of a TLB faulting virtual address are concatenated with the virtual offset of the user page table (level two).
  - The virtual page number of the faulting address is equal to the PTE index in the user page table. Therefore this virtual address points to the appropriate user PTE. A TLB lookup is performed using this virtual address.
  - If a load using this address succeeds, the user PTE is placed into the TLB and can translate the faulting virtual address.
  - The user PTE load can, however, cause a TLB miss of its own. Requiring step 2
- **Step 2:**
  - Perform top-down traversal or walking.

Software managed TLB/page walking usually used

Bottom-up Table Walking Example



- **The bottom-up method for accessing the hierarchical page table typically accesses memory only once to translate a virtual address (Step 1).**
- **It resorts to a top-down traversal if the initial attempt fails (Step 2)**

(e.g MIPS, Alpha)

Alternative Page Table Organizations:
Inverted/Hashed Page Tables (PowerPC, PA-RISC)

- Instead of one PTE entry for every virtual page belonging to a process, the inverted page table has one entry for every page frame in main memory.
  - The index of the PTE in the inverted table is usually equal to the page frame number (PFN) of the page it maps.
  - Thus, rather than scaling with the size of the virtual space, it scales with the size of physical memory.
- Since the physical page frame number is not usually available, the inverted table uses a hashed index based on the virtual page number (Typically XOR of upper and lower bits of virtual page number
- Since different virtual page numbers might produce identical hash values, a collision-chain mechanism is used to let these mappings exist in the table simultaneously.
  - In the classical inverted table, the collision chain resides within the table itself.
  - When a collision occurs, the system chooses a different slot in the table and adds the new entry to the end of the chain. It is thus possible to chase a long list of pointers while servicing a single TLB miss.
- Disadvantage: The inverted table only contains entries for virtual pages actively occupying physical memory. An alternate mapping structure is required to maintain information for pages on disk.

Reducing Collision-Chain Length in Inverted Page Tables

1 Increase Size of Inverted Page Table:

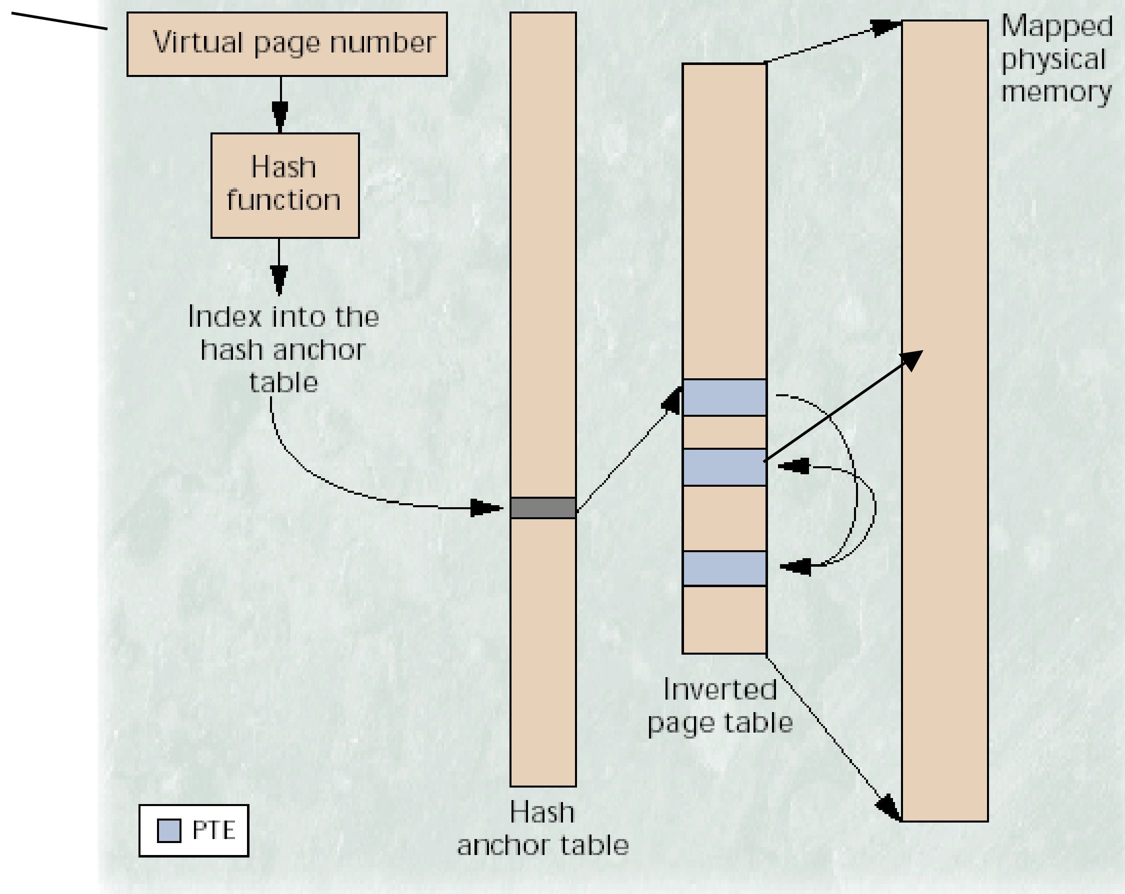
- **To keep the average chain length short, the range of hash values produced can be increased and thus increasing the size of the hash table.**
- **However, if the inverted page table's size were changed, the page frame number (PFN) could no longer be deduced from the PTE's location within the table.**
- **It would then be necessary to explicitly include the page frame number (PFN) in the PTE, thereby increasing the size of every PTE.**

2 Hash Anchor Table (HAT):

- **As a trade-off to keep the table small, the designers of early systems increased the number of memory accesses per lookup:**
  - **They added a level of indirection, the hash anchor table (HAT).**
- **The hash anchor table is indexed by the hash value and points to the chain head in the inverted table corresponding to each value.**
- **Doubling the size of the hash anchor table reduces the average collision-chain length by half, without having to change the size of the inverted page table.**
- **Since the entries in the hash anchor table are smaller than the entries in the inverted table, it is more memory efficient to increase the size of the hash anchor table to reduce the average collision-chain length.**

Inverted/Hashed Page Table

Faulting Virtual Address (i.e TLB miss or fault, not in TLB)



With Hash Anchor Table (HAT)

- **The inverted page table contains one PTE for every page frame in memory, making it densely packed compared to the hierarchical page table.**
- **It is indexed by a hash of the virtual page number.**

(e.g. HP PA-RISC)

Table Walking Algorithm For Inverted Page Table With Hash Anchor Table (HAT)

Step 1:

- The TLB faulting virtual page number is hashed, indexing the hash anchor table.
- The corresponding anchor-table entry is loaded and points to the chain head for that hash value.

Step 2:

- The indicated PTE is loaded, and its virtual page number is compared with the faulting virtual page number. If the two match, the algorithm terminates.

Step 3a:

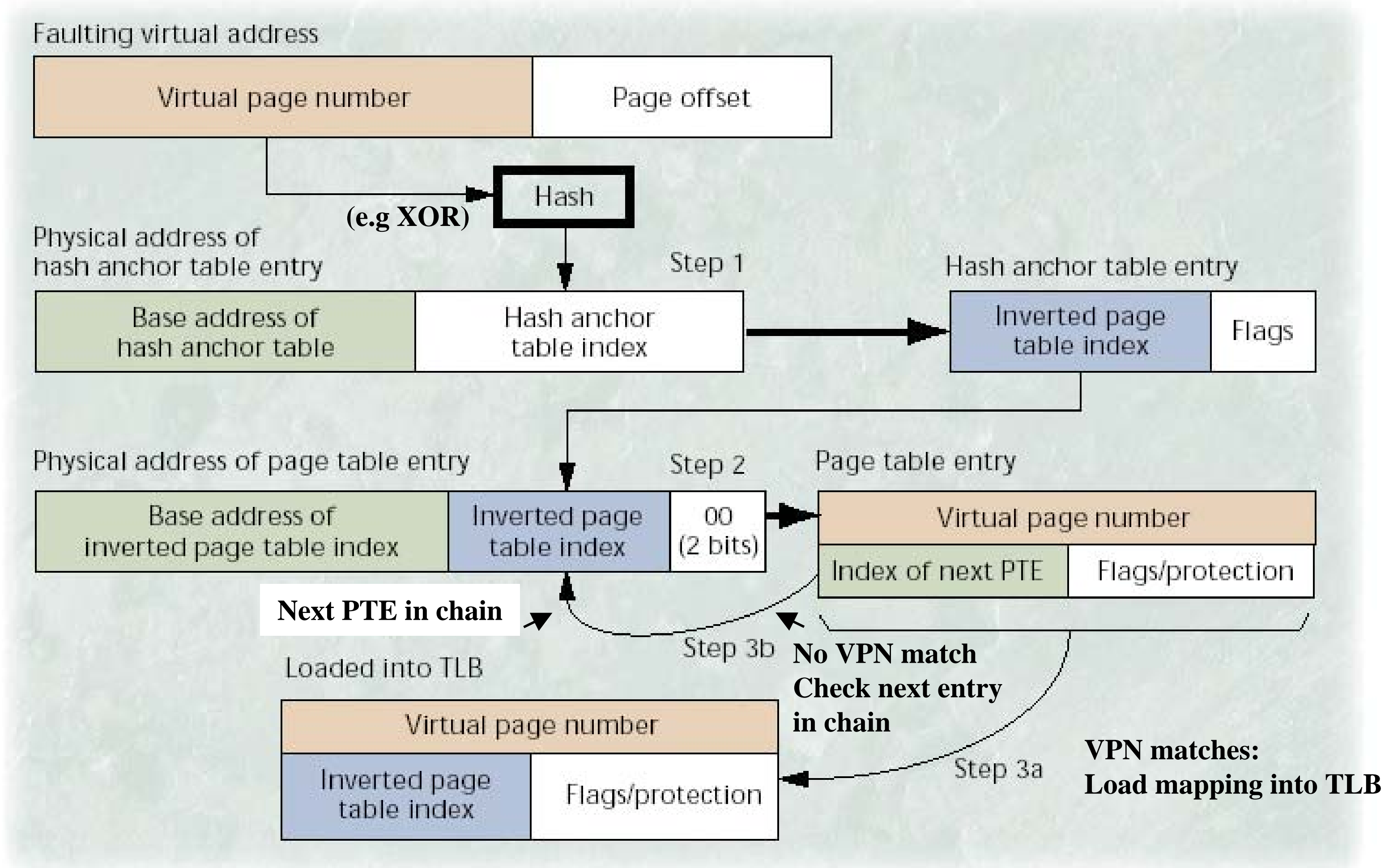
- The mapping, composed of the virtual page number and the page frame number (the PTE's index in the inverted page table), is placed into the TLB.

Step 3b:

- Otherwise, the PTE references the next entry in the chain (step 3b), or indicates that it is the last in the chain. If there is a next entry, it is loaded and compared.
- If the last entry fails to match, the algorithm terminates and causes a page fault.

Table Walking Algorithm For Inverted Page Table

TLB Fault:



Virtual Memory Architecture Examples

Table 1. Comparison of architectural support for virtual memory in six commercial MMUs.

Item	MIPS	Alpha	PowerPC	PA-RISC	UltraSPARC	IA-32
Address space protection	ASIDs	ASIDs	Segmentation	Multiple ASIDs	ASIDs	Segmentation
Shared memory	GLOBAL bit in TLB entry	GLOBAL bit in TLB entry	Segmentation	Multiple ASIDs; segmentation	Indirect specification of ASIDs	Segmentation
Large address spaces	64-bit addressing	64-bit addressing	52-/80-bit segmented addressing	96-bit segmented addressing	64-bit addressing	None
Fine-grained protection	In TLB entry	In TLB entry	In TLB entry	In TLB entry	In TLB entry	In TLB entry; per segment
Page table support	Software-managed TLB	Software-managed TLB	Hardware-managed TLB; inverted page table	Software-managed TLB	Software-managed TLB	Hardware-managed TLB/hierarchical page table
Superpages	Variable page size set in TLB entry: 4 Kbyte to 16 Mbyte, by 4	Groupings of 8, 64, 512 pages (set in TLB entry)	Block address translation: 128 Kbytes to 256 Mbytes, by 2	Variable page size set in TLB entry:4 Kbytes to 64 Mbytes, by 4	Variable page size set in TLB entry: 8, 64, 512 Kbytes, and 4 Mbytes	Segmentation/ variable page size set in TLB entry: 4 Kbytes or 4 Mbytes

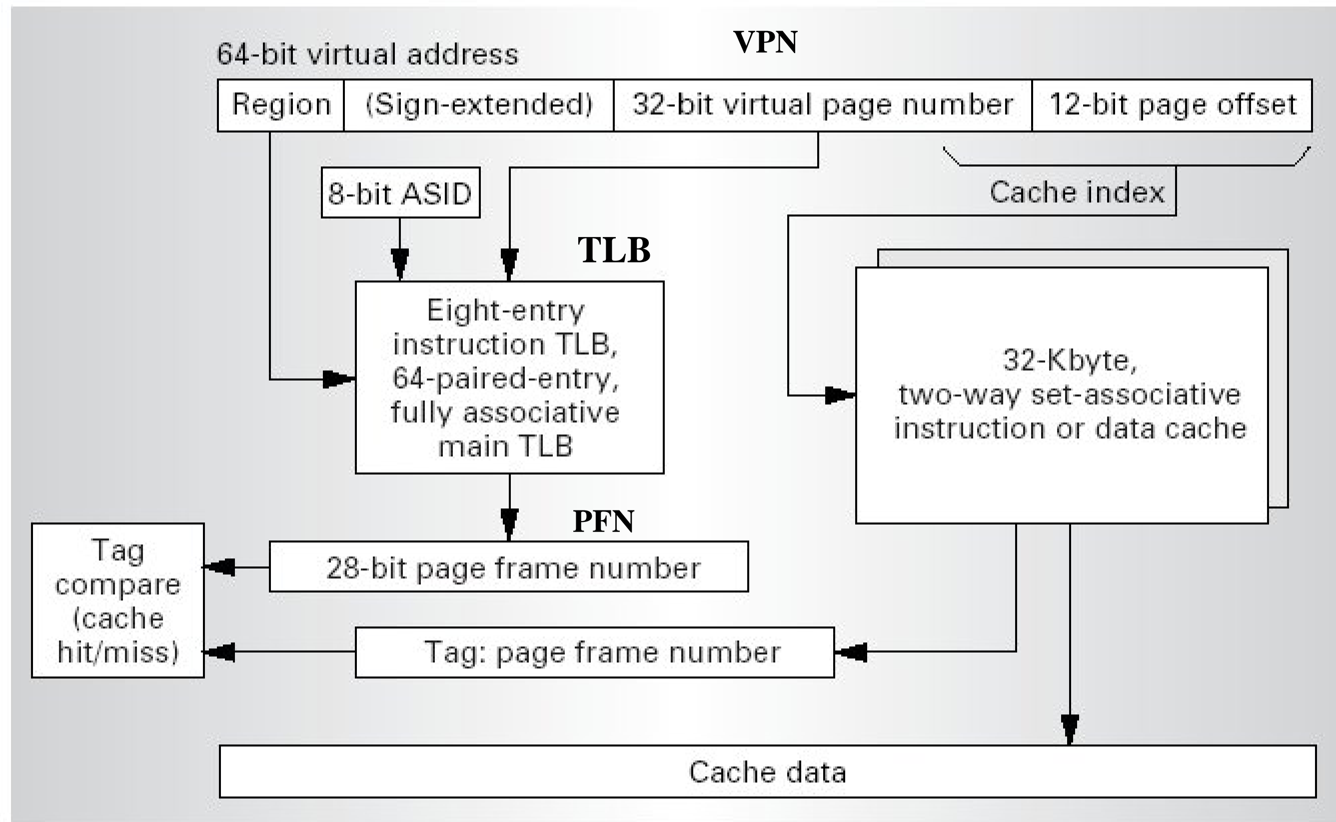
Source:
Virtual memory in contemporary microprocessors, B. Jacob, and T. Mudge, Micro, vol. 18, no. 4, pp. 60-75. July/Aug. 1998.
(Virtual Memory paper #2)

MIPS Virtual Memory Architecture

- **OS handles TLB misses entirely in software.**
- **The hardware supports a bottom up hierarchical page table through the TLB context register.**
- **MIPS uses address-space identifiers (ASIDs) to provide address-space protection.**
  - **To access a page, the ASID of the currently active process must match the ASID in the page's TLB entry.**
- **Periodic cache and TLB flushes are unavoidable, as there are only 64 unique context identifiers in the R2000/R3000 and 256 in the R10000 (8-bit ASID).**
  - **This is because systems usually have more active processes than this, requiring sharing of address-space identifiers and periodic ASID remapping.**

Per-Process Virtual Address Space
Software-Managed TLB
(No fixed page table organization)

MIPS R10000 Address Translation Mechanism

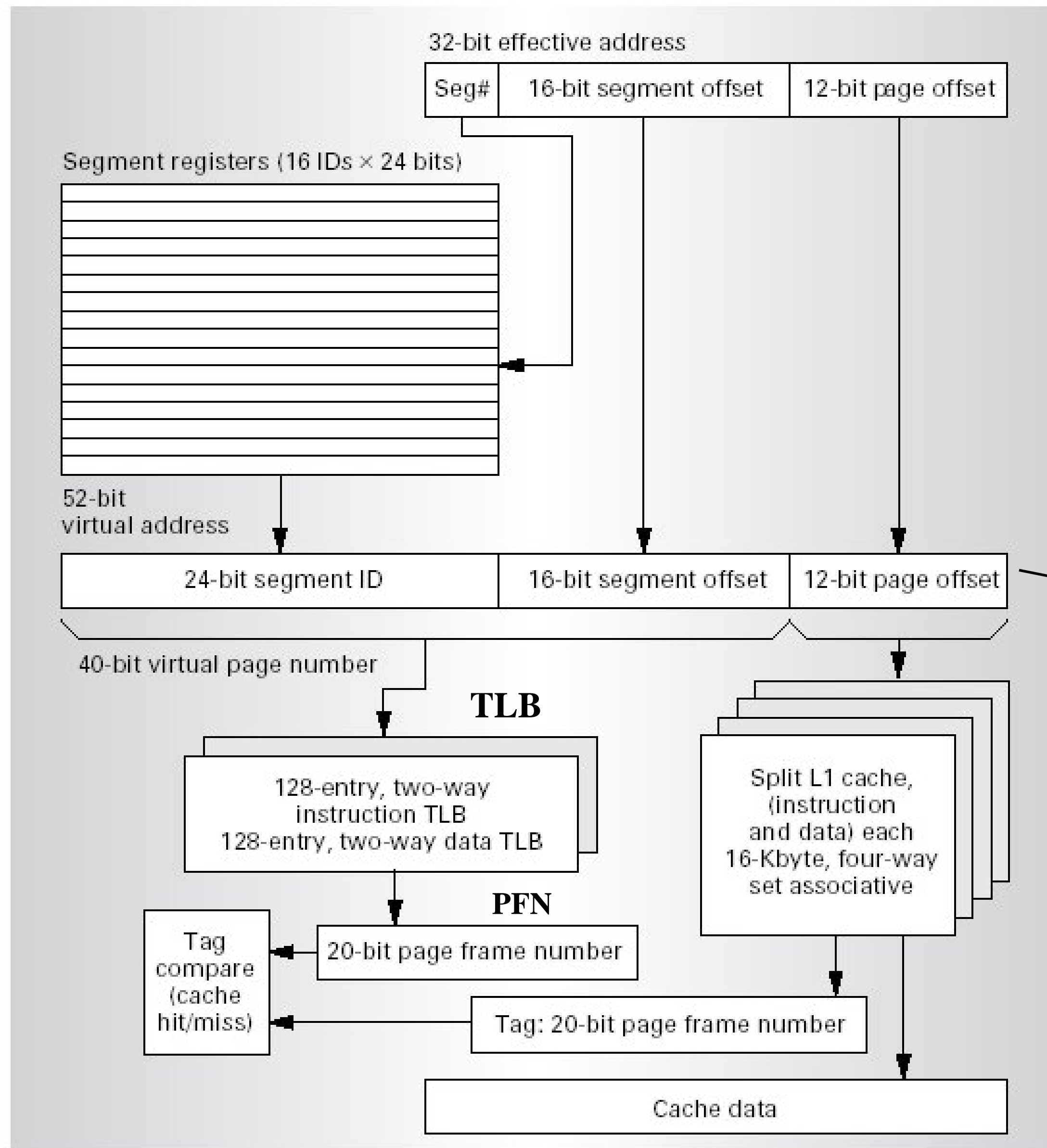


Per-Process Virtual Address Space
Software-Managed TLB
(No fixed page table organization)

PowerPC 604 Virtual Memory Architecture

- **The PowerPC 604, which maps a process's logical/effective addresses onto a global flat virtual address space using paged segmentation.**
- **Segments are 256-Mbyte contiguous regions of virtual space, and 16 segments make up an application's 4-Gbyte address space.**
  - The top 4 bits of the 32-bit effective address select a segment identifier from a set of 16 hardware segment registers.
- **The segment identifier is concatenated with the bottom 28 bits of the effective address to form an extended virtual address that indexes the caches and is mapped by the TLBs and page table.**
- **The PowerPC defines a hashed page table for the OS: a variation on the inverted page table that acts as an eight-way set-associative software cache for PTEs.**
  - Similar to the classic inverted table, it requires a backup page table for maintain information for pages on disk
- **Hardware TLB-Refill: On TLB misses, hardware walks the hashed page table.**
- **Address-space protection is supported through the segment registers, which can only be modified by the OS.**
- **The segment identifiers are 24 bits wide and can uniquely identify over a million processes.**
- **If shared memory is implemented through the segment registers, the OS will rarely need to remap segment identifiers.**

Global Virtual Address Space, Hardware-Managed TLB (Inverted page table organization)

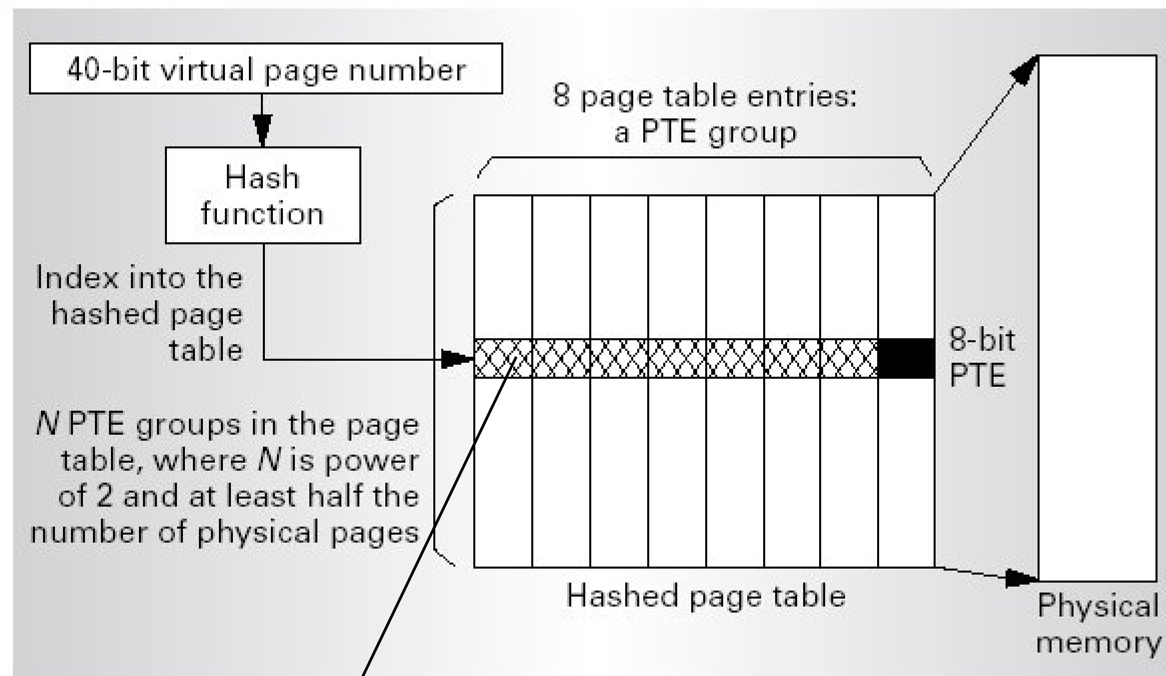


PowerPC 604 Address Translation Mechanism

Global Virtual Address

Global Virtual Address Space, Hardware-Managed TLB (Inverted page table organization)

PowerPC Hashed/Inverted Page Table Structure



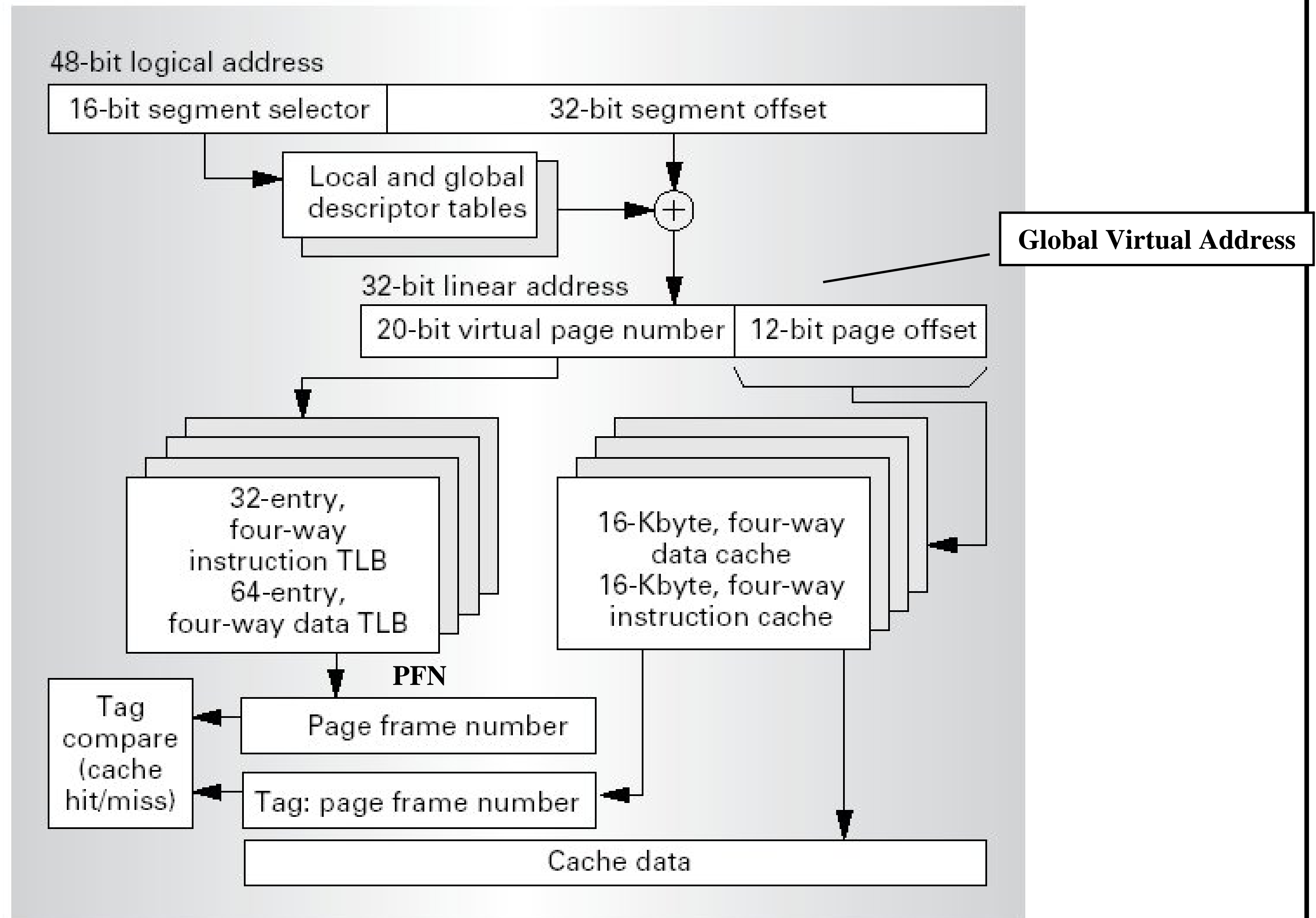
Eight-way set-associative software cache for PTEs

IA-32 (x86) Virtual Memory Architecture

- **X86 another architecture which uses paged segmentation.**
- **The x86's segmentation mechanism often goes unused by today's OSs, which instead flush the TLBs on context switch to guarantee protection.**
- **The per-process hierarchical page tables are hardware defined and hardware-walked.** **i.e hardware TLB refill using a finite state machine**
  - The OS provides to the hardware a physical address for the root page table in one of a set of control registers, CR3.
  - Hardware uses this address to walk the two-tiered table in a **top-down fashion** on every TLB miss.
- **Unlike the PowerPC, the segmentation mechanism supports variable-sized segments from 1 byte to 4 Gbytes in size, and the global virtual space is the same size as an individual user-level address space (4 Gbytes).**
- **User-level applications generate 32-bit addresses that are extended by 16-bit segment selectors.**
- **Hardware uses the 16-bit selector to index one of two software descriptor tables, producing a base address for the segment corresponding to the selector.**
- **This base address is added to the 32-bit virtual address generated by the application to form a global 32-bit linear address.**
- **For performance, the hardware caches six of a process's selectors in a set of on-chip segment registers that are referenced by context.**

Global Virtual Address Space, Hardware-Managed TLB (Two-level hierarchical page table organization)

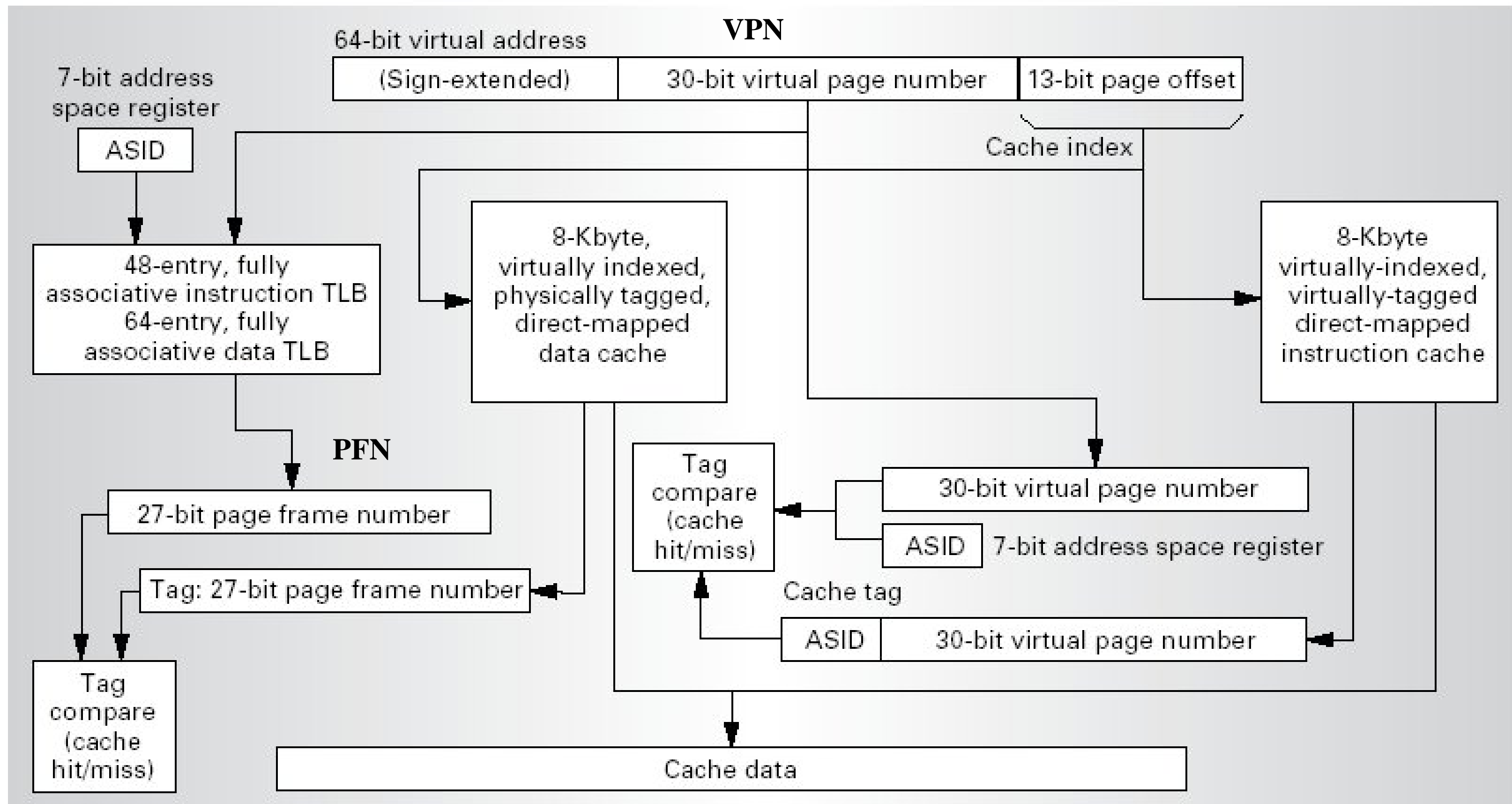
Intel Pentium II/III Address Translation Mechanism



Global Virtual Address

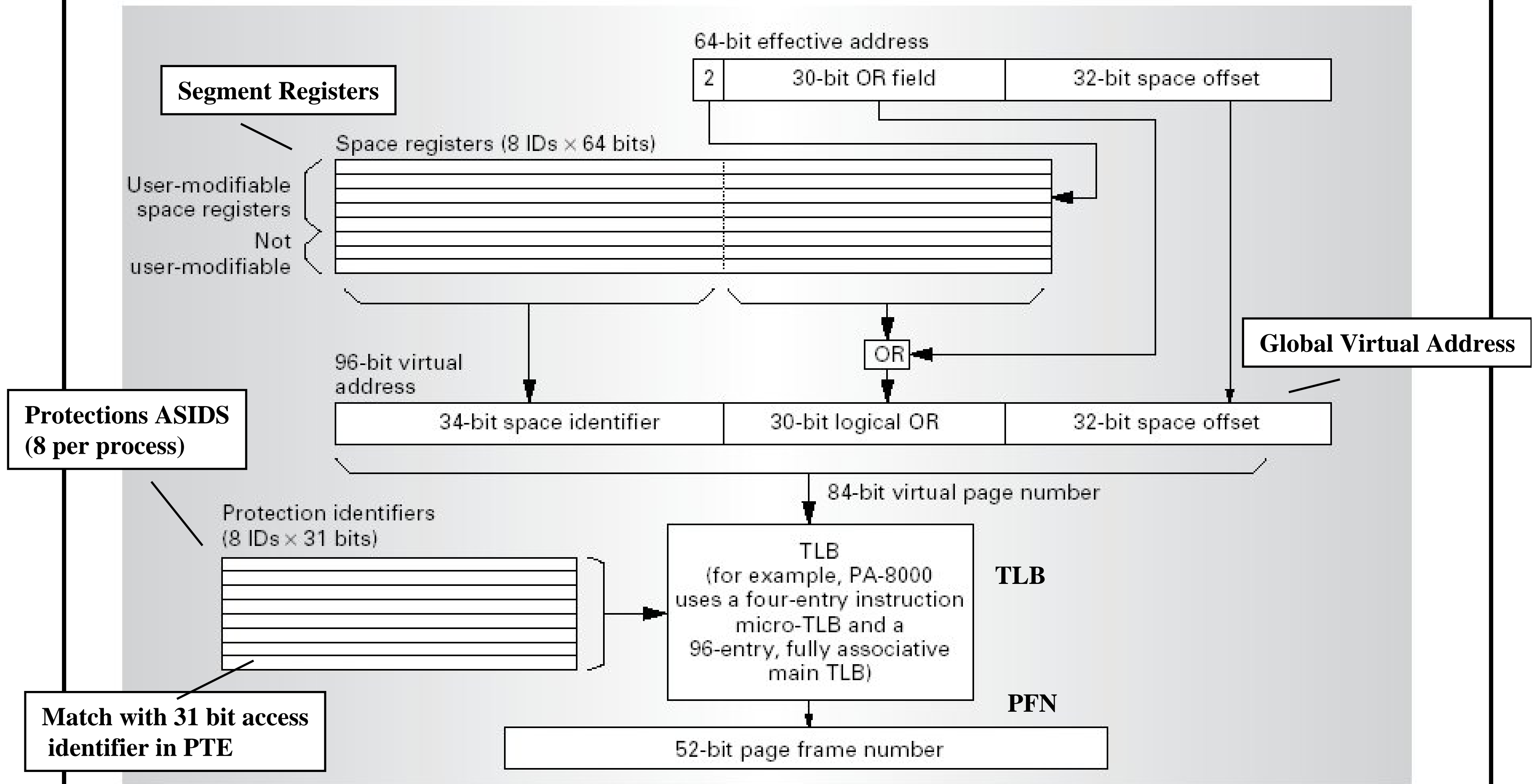
Global Virtual Address Space, Hardware-Managed TLB
(Two-level hierarchical page table organization)

Alpha 21164 Address Translation Mechanism



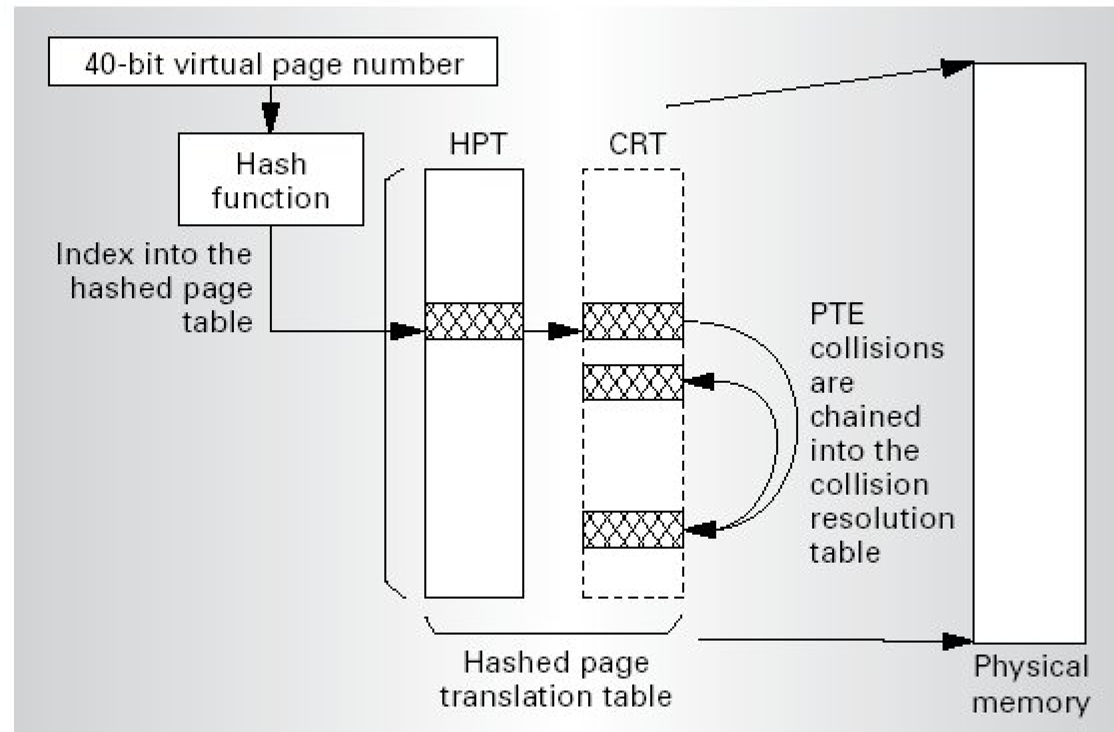
Per-Process Virtual Address Space
Software-Managed TLB. No fixed page table organization):
But 3-level hierarchical page with Bottom-up Table Walking is common

PA-RISC 2.0 Address Translation Mechanism



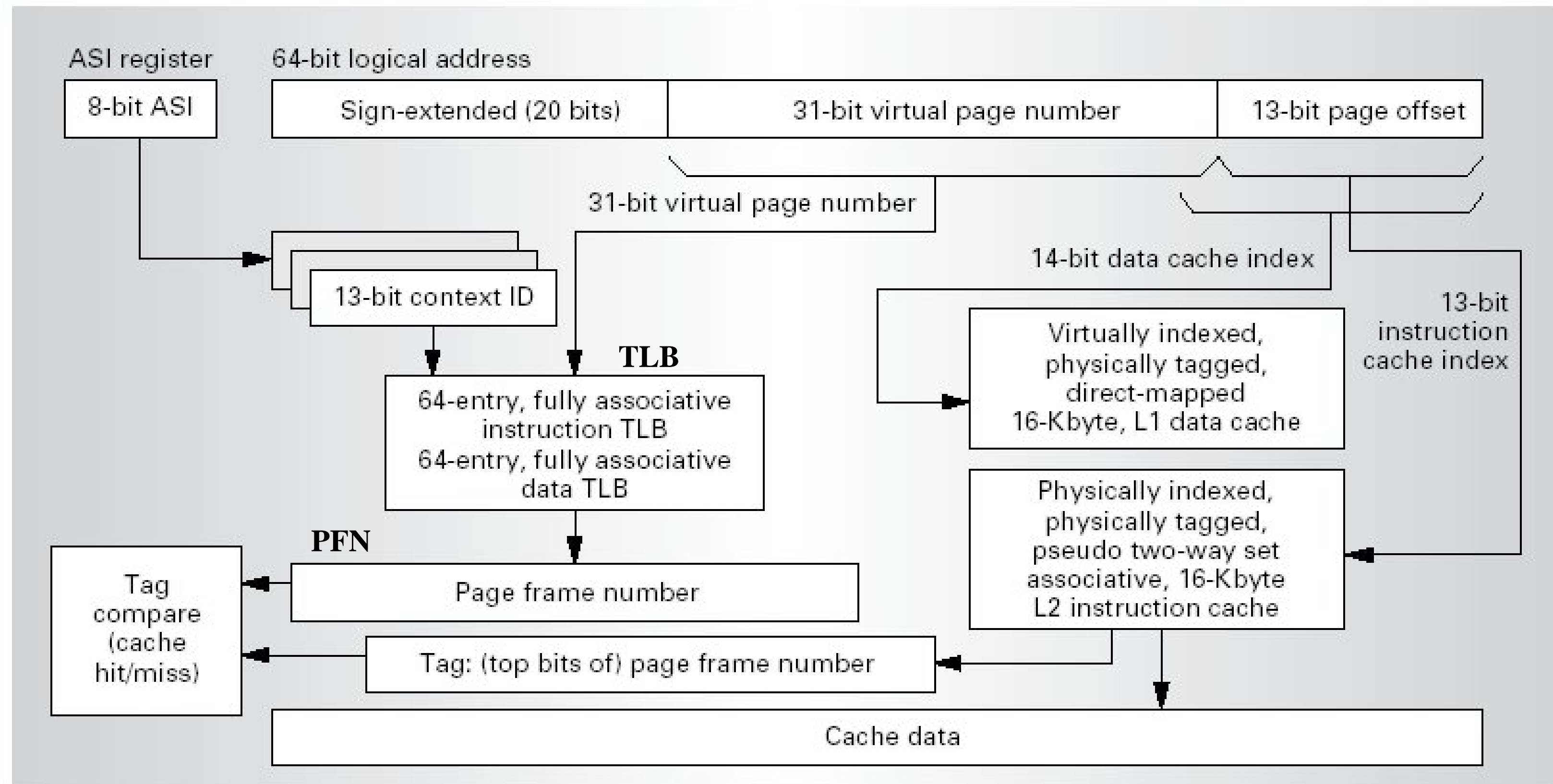
Global virtual address space using segmentation + Multiple ASIDS

PA-RISC Hashed/Inverted Page Translation Table (HPT)



Classic inverted page table.
Walked by software (any page organization can be supported by OS)

UltraSPARC I Address Translation Mechanism



Per-Process Virtual Address Space
Walked by software (any page organization can be supported by OS)

The Hardware (MMU)/Software (OS) Mismatch In Virtual Memory

- **Most modern processors have hardware to support virtual memory in terms Memory Management Units (MMUs) including TLBs, special registers.**
- **Unfortunately, there has not been much agreement on the form that this support should take.**
- **No serious attempts have been made to create a common memory-management support or a standard interface to the OS.**
- **The result of this lack of agreement is that hardware mechanisms are often completely incompatible in terms of:**
  - **Hardware support for Global or Per-process Virtual Address Space**
  - **Protection and data/code sharing mechanisms.**
  - **Page table organization supported by hardware.**
  - **TLB-refill & page walking mechanisms**
  - **Hardware support for Global or Per-process Virtual Address Space.**
  - **Hardware support for superpages ….**
- **Thus, designers and porters of system-level software have three somewhat unattractive choices:**
  - **Write software to fit many different architectures, which can compromise performance and reliability; or**
  - **Insert layers of software to emulate a particular hardware interface, possibly compromising performance and compatibility**
  - **Operating system developers often use only a small subset of the complete functionality of memory-management units (MMUs) to make OS porting more manageable.**

Solution ? Common industry-standard virtual memory interface ?
or …. No MMU/hardware TLB ?