Virtual Memory

Chapter 18

S. Dandamudi

Outline

- Introduction
- Virtual memory concepts
  - Page replacement policies
  - Write policy
  - Page size tradeoff
  - Page mapping
- Page table organization
  - Page table entries
- Translation lookaside buffer
- Page table placement
  - Searching hierarchical page tables
- Inverted page table organization
- Segmentation
- Example implementations
  - Pentium
  - PowerPC
  - MIPS

Introduction

- Virtual memory deals with the main memory size limitations
  - ∗ Provides an illusion of having more memory than the system’s RAM
  - ∗ Virtual memory separates logical memory from physical memory
    - » Logical memory: A process’s view of memory
    - » Physical memory: The processor’s view of memory
  - ∗ Before virtual memory
    - » Overlaying was used
      - – It is a programmer controlled technique

Introduction (cont’d)

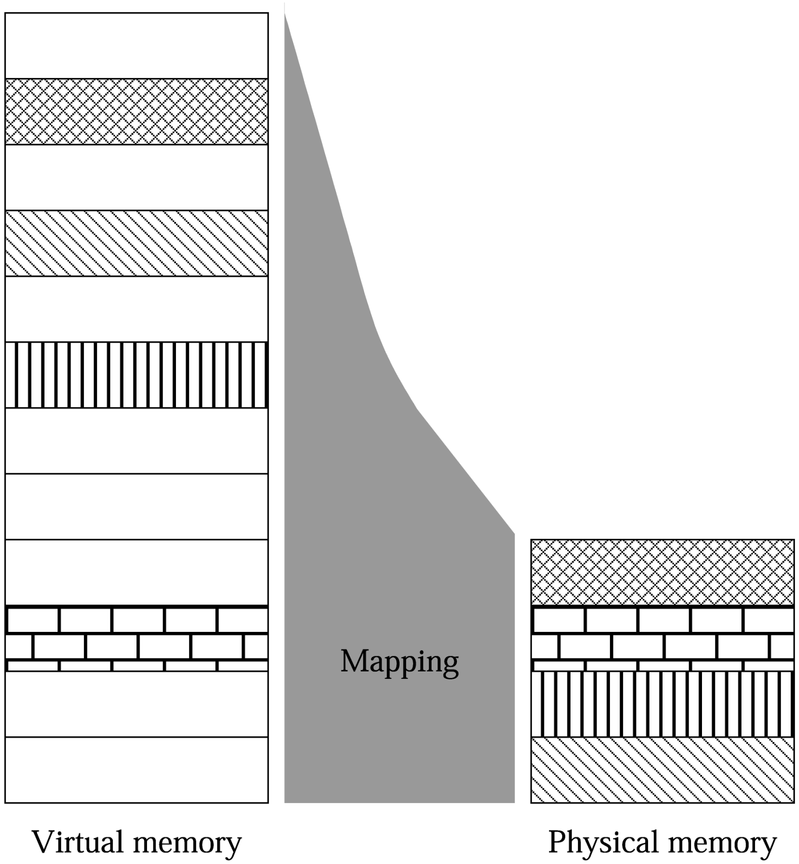
- Virtual memory
  - ∗ Automates the overlay management process
    - » Big relief to programmers
- Virtual memory also provides
  - ∗ Relocation
    - » Each program can have its own virtual address space
    - » Run-time details do not have any impact on code generation
  - ∗ Protection
    - » Programs are isolated from each other
      - – A benefit of working in their own address spaces
    - » Protection can be easily implemented

Introduction (cont'd)

- Principles involved are similar to the cache memory systems
  - ∗ Details are quite different
    - » Due to different objectives
  - ∗ Concept of locality is key
    - » Exploits both types of locality
      - – Temporal
      - – Spatial
  - ∗ Implementation is different
    - » Due to different lower-level memory (disk)
      - – Several orders of magnitude slower

Virtual Memory Concepts

- Implements a mapping function
  - Between virtual address space and physical address space
- Examples
  - PowerPC
    - 48-bit virtual address
    - 32-bit physical address
  - Pentium
    - Both are 32-bit addresses
      - But uses segmentation



To be used with S. Dandamudi, “Fundamentals of Computer Organization and Design,” Springer, 2003.

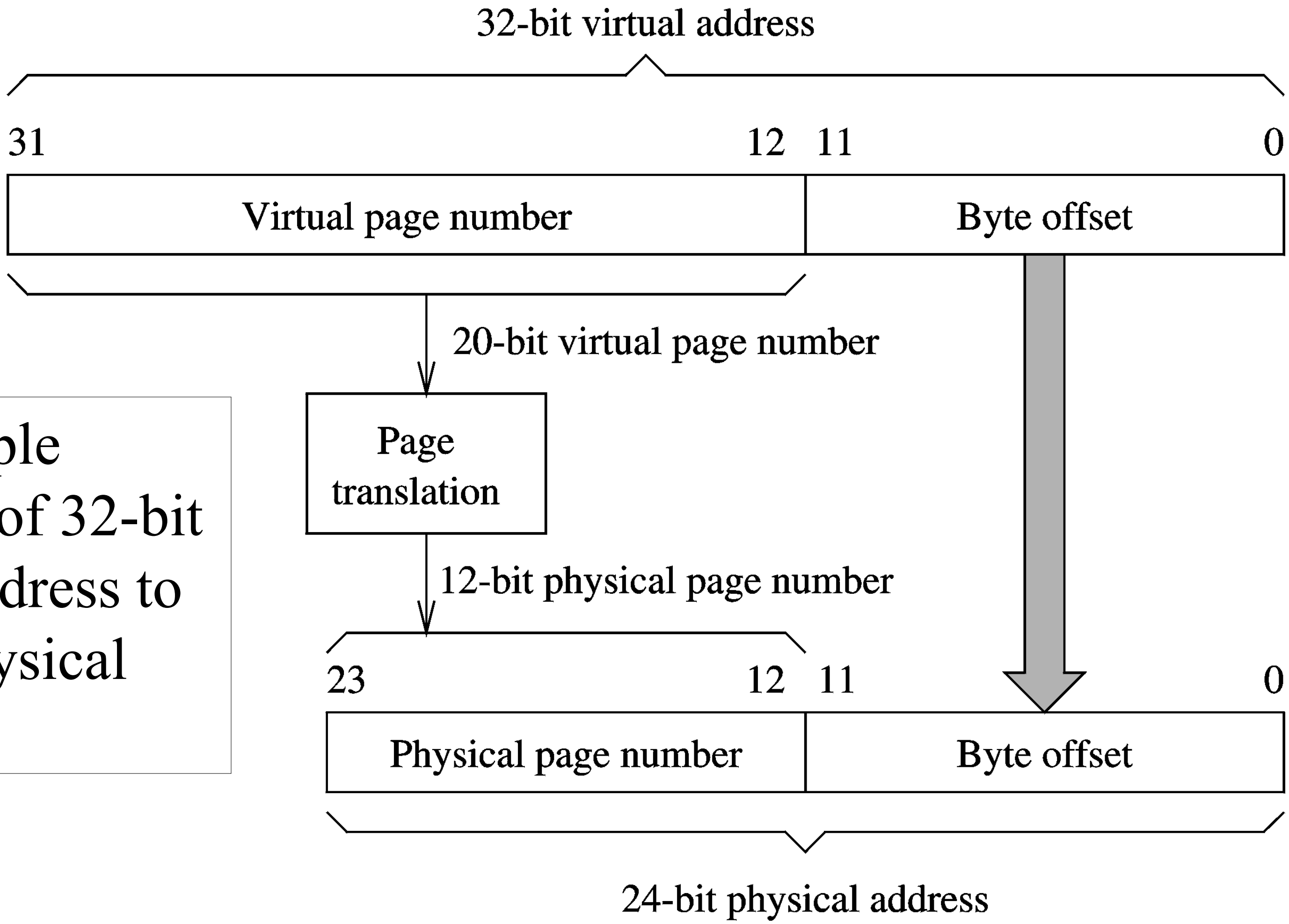
Virtual Memory Concepts (cont'd)

- Virtual address space is divided into fixed-size chunks
  - ∗ These chunks are called ***virtual pages***
  - ∗ Virtual address is divided into
    - » Virtual page number
    - » Byte offset into a virtual page
  - ∗ Physical memory is also divided into similar-size chunks
    - » These chunks are referred to as ***physical pages***
    - » Physical address is divided into
      - – Physical page number
      - – Byte offset within a page

Virtual Memory Concepts (cont'd)

- Page size is similar to cache line size
- Typical page size
  - » 4 KB
- Example
  - ∗ 32-bit virtual address to 24-bit physical address
  - ∗ If page size is 4 KB
    - » Page offset: 12 bits
    - » Virtual page number: 20 bits
    - » Physical page number: 12 bits
  - ∗ Virtual memory maps 2^{20} virtual pages to 2^{12} physical pages

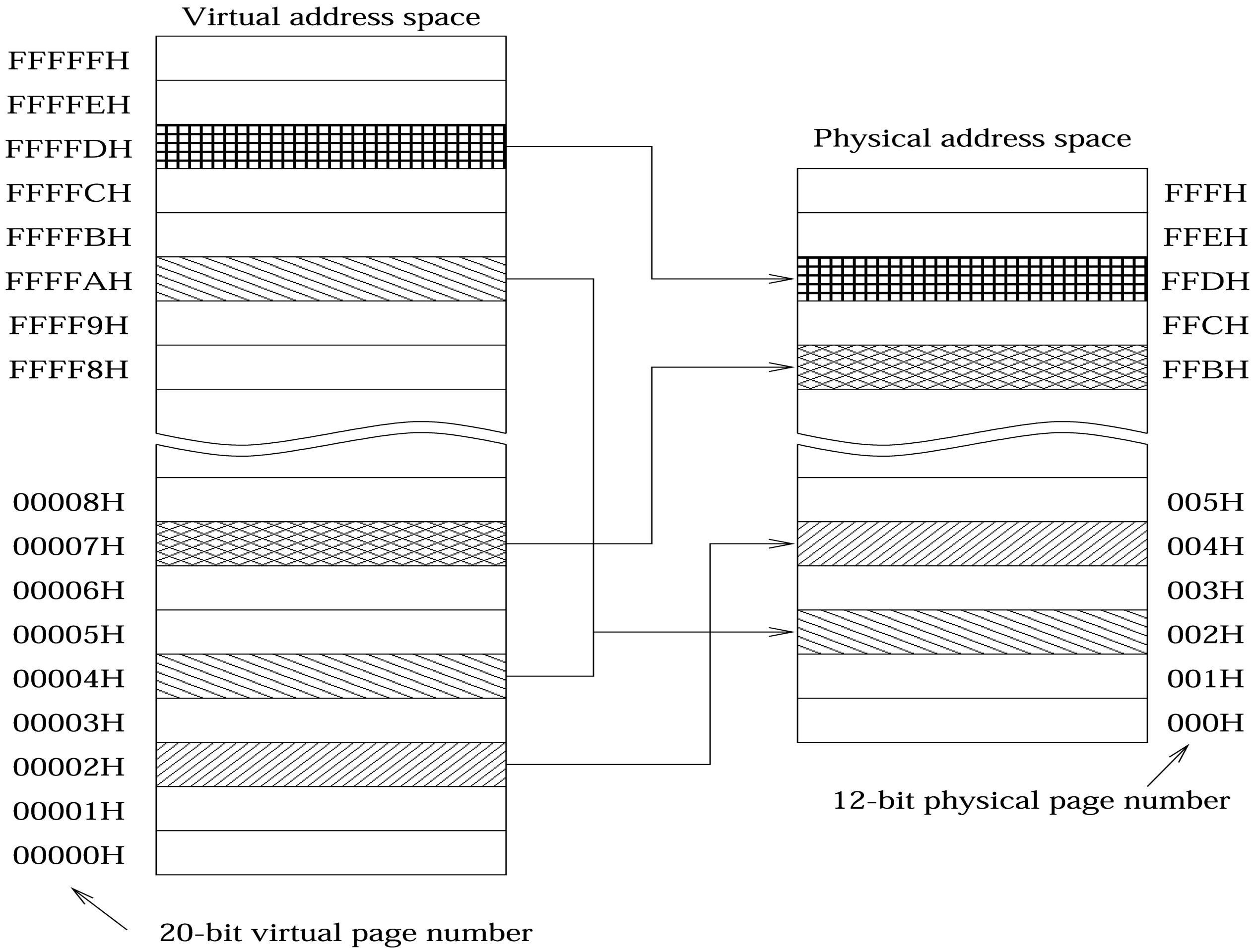
Virtual Memory Concepts (cont'd)



An example mapping of 32-bit virtual address to 24-bit physical address

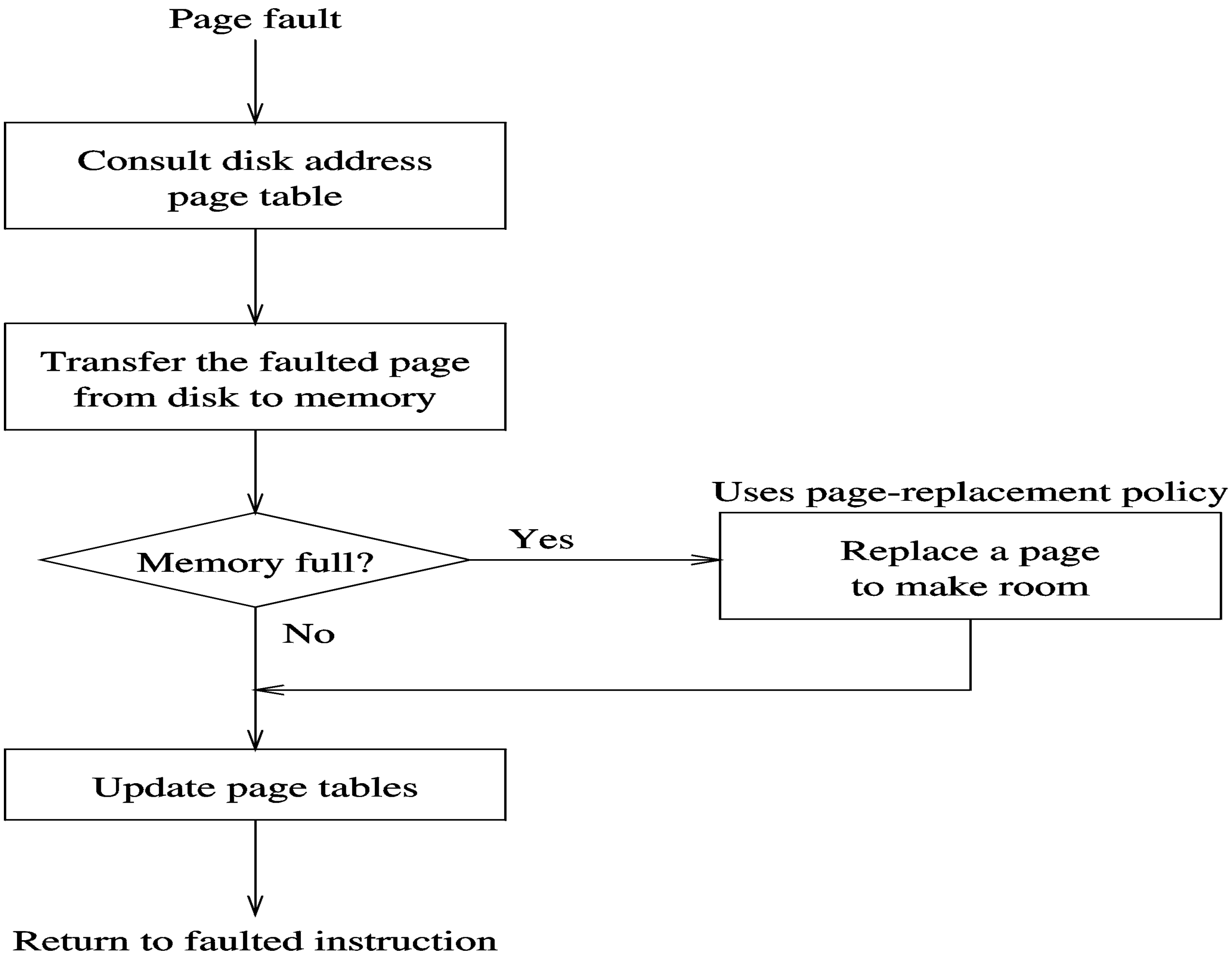
Virtual Memory Concepts (cont'd)

Virtual to physical address mapping



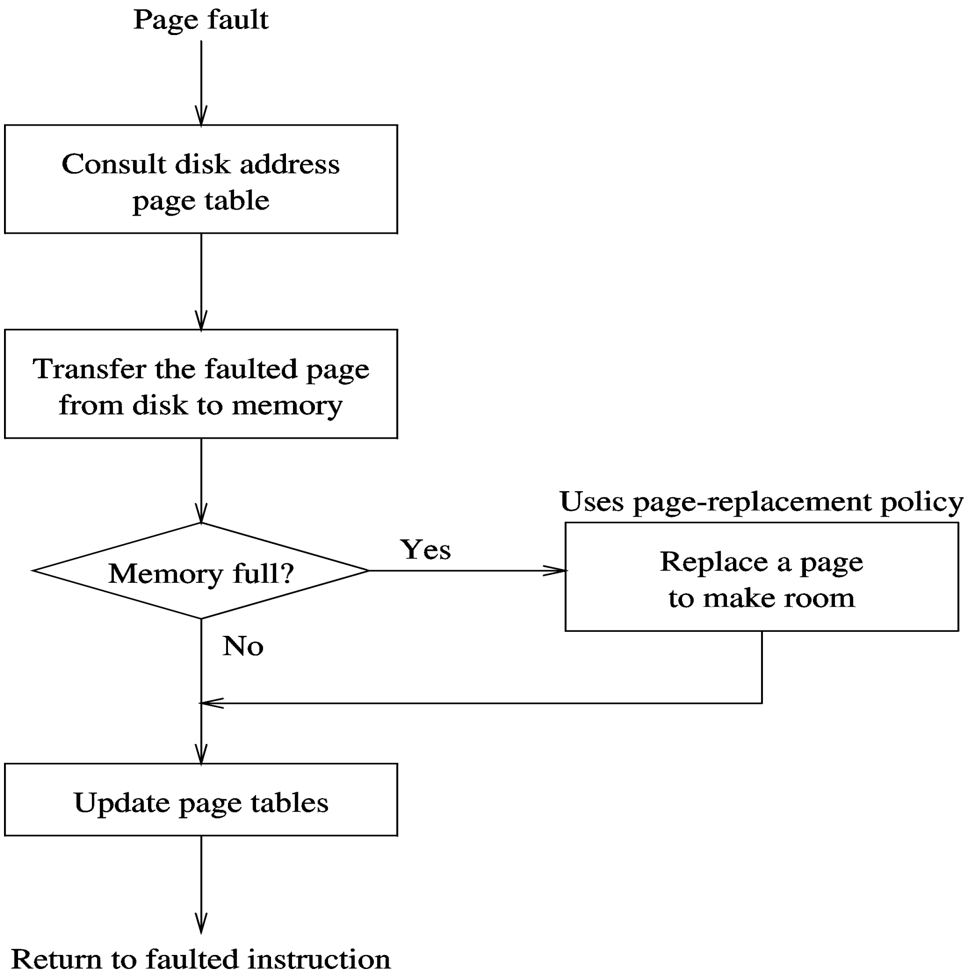
Virtual Memory Concepts (cont'd)

Page fault handling routine


Page fault
Consult disk address page table
Transfer the faulted page from disk to memory
Uses page-replacement policy
Memory full?
Yes
Replace a page to make room
No
Update page tables
Return to faulted instruction

Virtual Memory Concepts (cont’d)

- A virtual page can be
  - ∗ In main memory
  - ∗ On disk
- Page fault occurs if the page is not in memory
  - ∗ Like a cache miss
- OS takes control and transfers the page
  - ∗ Demand paging
    - » Pages are transferred on demand


Page fault
Consult disk address page table
Transfer the faulted page from disk to memory
Uses page-replacement policy
Memory full?
Yes
Replace a page to make room
No
Update page tables
Return to faulted instruction

Virtual Memory Concepts (cont'd)

- Page Replacement Policies
  - ∗ Similar to cache replacement policies
  - ∗ Implemented in software
    - » As opposed to cache's hardware implementation
  - ∗ Can use elaborate policies
    - » Due to slow lower-level memory (disk)
  - ∗ Several policies
    - » FIFO
    - » Second chance
    - » NFU
    - » LRU (popular)
      - – Pseudo-LRU implementation approximates LRU

Virtual Memory Concepts (cont'd)

- Write policies
  - ∗ For cache systems, we used
    - » Write-through
      - – Not good for VM due to disk writes
    - » Write-back
- Page size tradeoffs
  - ∗ Factors favoring small page sizes
    - » Internal fragmentation
    - » Better match with working set
  - ∗ Factors favoring large page sizes
    - » Smaller page sizes
    - » Disk access time

Pentium, PowerPC:
4 KB
MIPS:
7 page sizes
between
4 KB to 16 MB

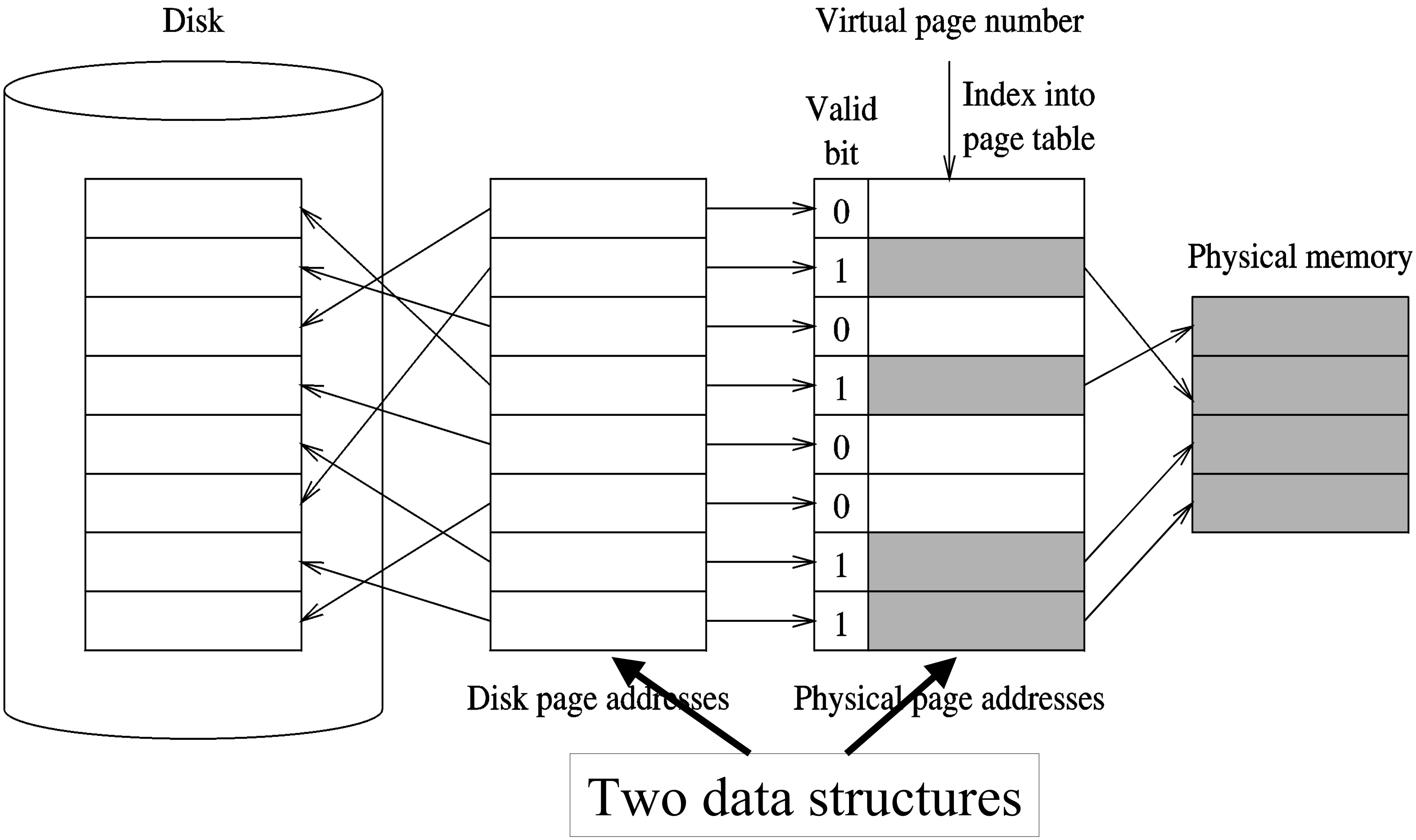
Virtual Memory Concepts (cont'd)

- Page mapping
  - ∗ Miss penalty is high
    - » Should minimize miss rate
  - ∗ Can use fully associative mapping
    - » Could not use this for cache systems due to hardware complexity
  - ∗ Uses a translation table
    - » Called ***page table***
  - ∗ Several page table organizations are possible

Page Table Organization (cont'd)

- Simple page table organization
  - ∗ Each entry in a page table consists of
    - » A virtual page number (VPN)
    - » Corresponding physical page number (PPN)
  - ∗ Unacceptable overhead
- Improvement
  - ∗ Use virtual page number as index into the page table
- Typical page table is implemented using two data structures

Page Table Organization (cont’d)



Two data structures

Page Table Organization (cont'd)

- Page table entry
  - ∗ Physical page number
    - » Gives location of the page in memory if it is in memory
  - ∗ Disk page address
    - » Specifies location of the page on the disk
  - ∗ Valid bit
    - » Indicates whether the page is in memory
      - – As in cache memory
  - ∗ Dirty bit
    - » Indicates whether the page has been modified
      - – As in cache memory

Page Table Organization (cont'd)

- ∗ Reference bit
  - » Used to implement pseudo-LRU
    - – OS periodically clears this bit
    - – Accessing the page turns it on
- ∗ Owner information
  - » Needed to implement proper access control
- ∗ Protection bits
  - » Indicates type of privilege
    - – Read-only, execute, read/write
  - » Example: PowerPC uses three protection bits
    - – Controls various types of access to user- and supervisor-mode access requests

Translation Lookaside Buffer

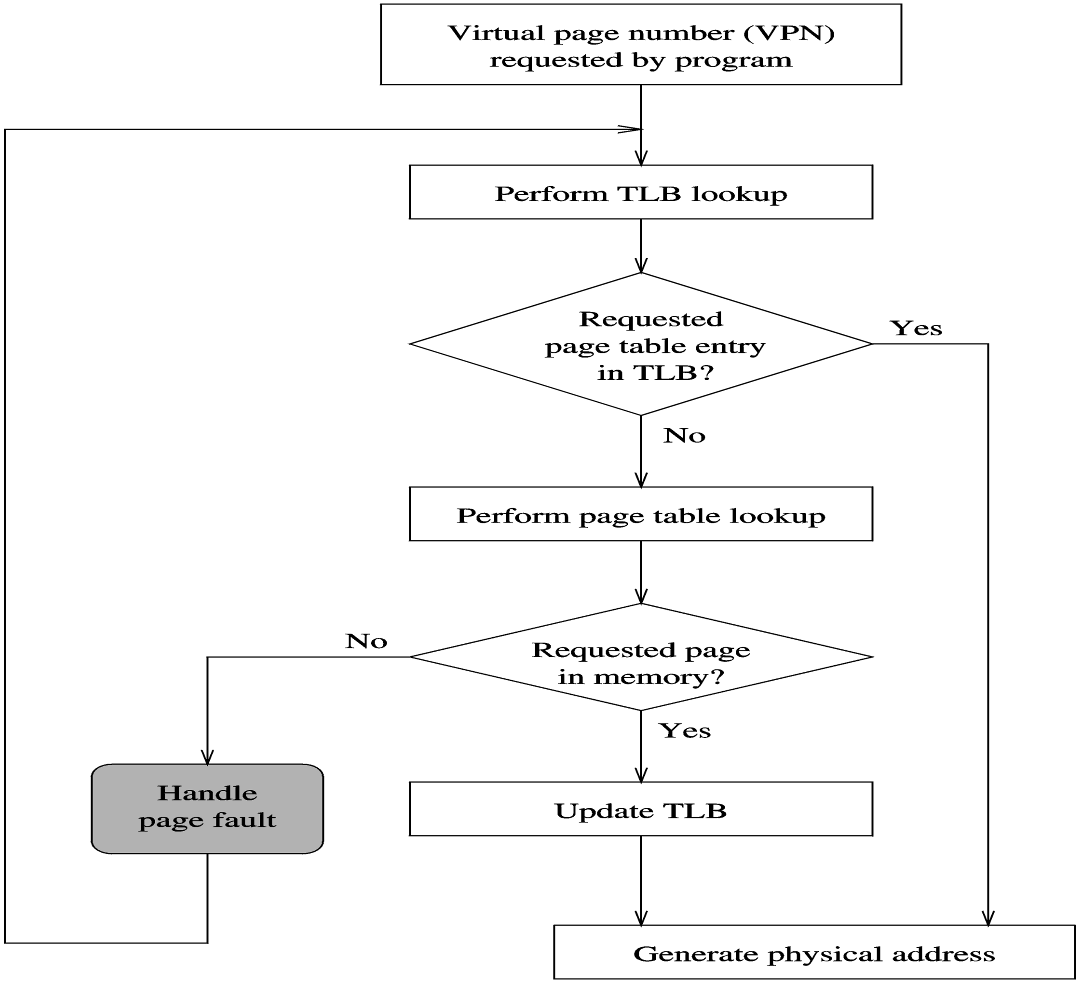
- For large virtual address spaces
  - ∗ Translation table must be in stored in virtual address space
    - » Every address translation requires two memory accesses:
      - – To get physical page number for the virtual page number
      - – To get the program’s memory location
- To reduce this overhead, most recently used PTEs are kept in a cache
  - ∗ This is called the ***translation lookaside buffer*** (TLB)
    - » Small in size
    - » 32 – 256 entries (typical)

Translation Lookaside Buffer (cont’d)

- Each TLB entry consists of
  - ∗ A virtual page number
  - ∗ Corresponding physical page number
  - ∗ Control bits
    - » Valid bit
    - » Reference bit
    - » . . .
- Most systems keep separate TLBs for data and instructions
  - ∗ Examples: Pentium and PowerPC

Translation Lookaside Buffer (cont’d)

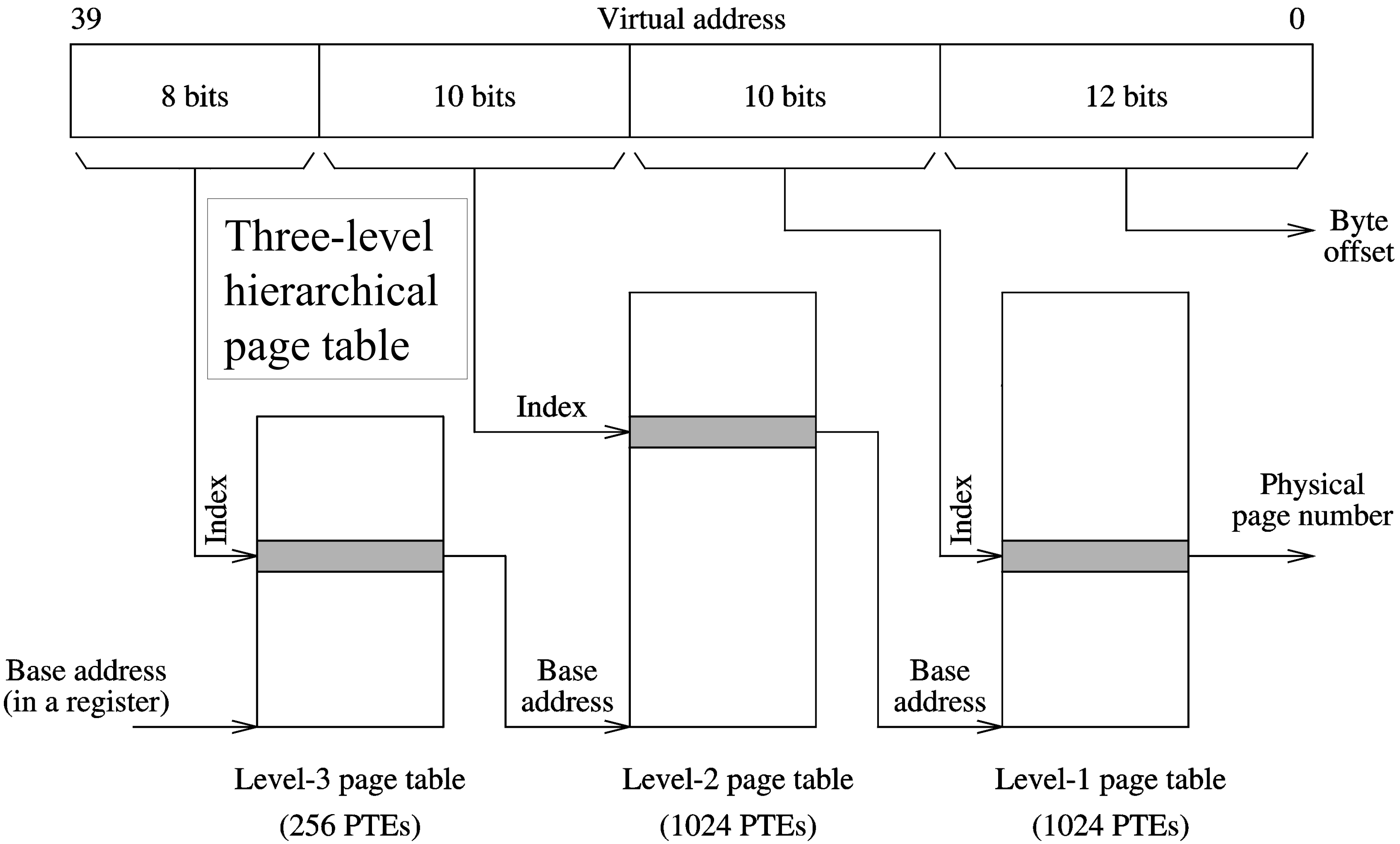
Translation using a TLB


Virtual page number (VPN) requested by program
Perform TLB lookup
Requested page table entry in TLB?
Yes
No
Perform page table lookup
Requested page in memory?
No
Yes
Handle page fault
Update TLB
Generate physical address

Page Table Placement

- Large virtual address spaces need large translation tables
  - Placed in virtual address space
  - Since the entire page table is not needed, we can use a hierarchical design
    - Partition the page table into pages (like the user space)
    - Bring in the pages that are needed
    - We can use a second level page table to point to these first-level tables pages
    - We can recursively apply this procedure until we get a small page table

Page Table Placement (cont'd)



Page Table Placement (cont'd)

- Hierarchical page tables are also called ***forward-mapped page tables***
  - ∗ Translation proceeds from virtual page number to physical page number
    - » In contrast to inverted page table
- Examples
  - ∗ Pentium
    - » 2-level hierarchy
    - » Details later
  - ∗ Alpha
    - » 4-level hierarchy

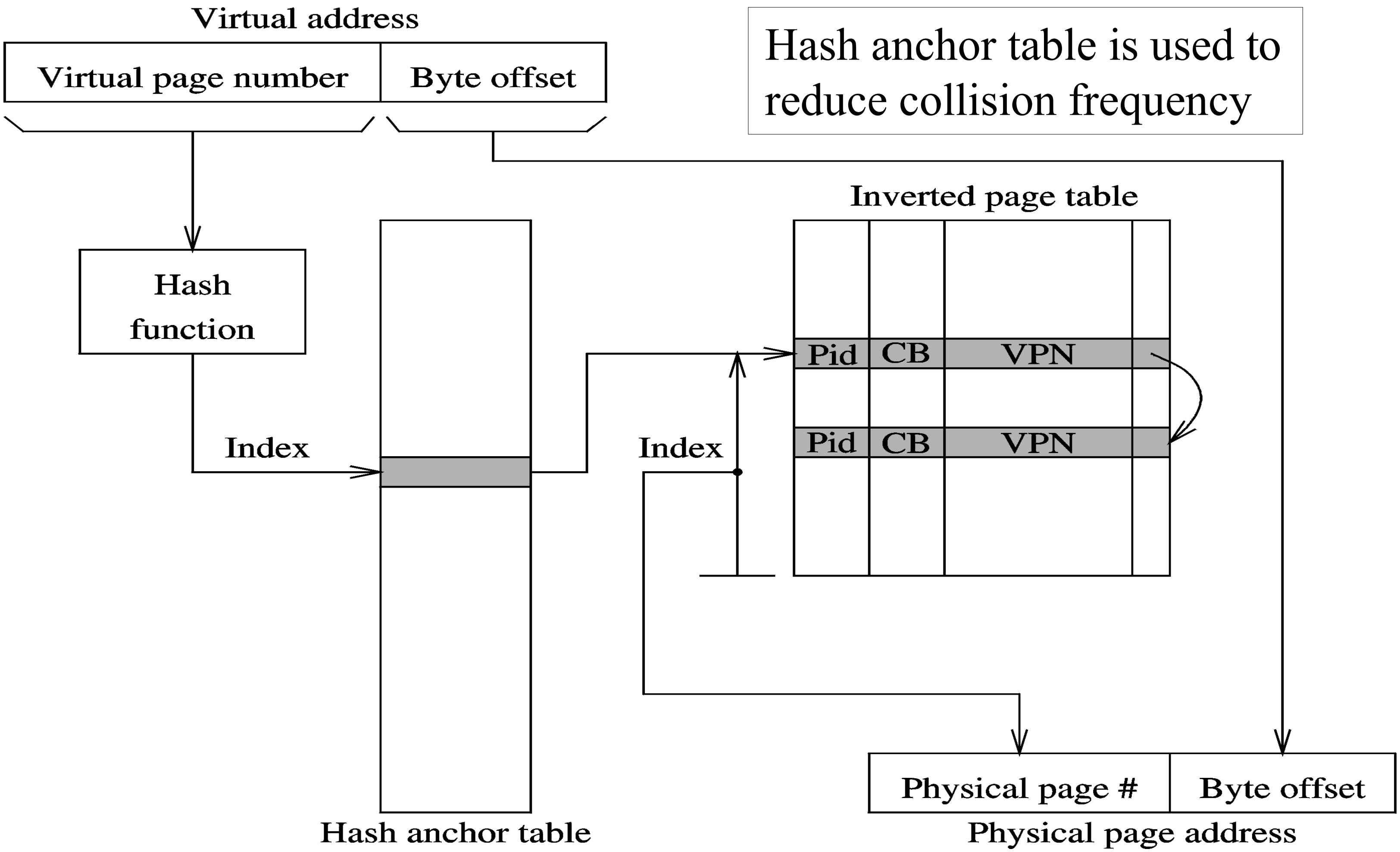
Page Table Placement (cont’d)

- Searching hierarchical page tables
  - ∗ Two strategies
    - » Top-down
      - – Starts at the root and follows all the levels
      - – Previous example requires four memory accesses
        - ✈3 for the three page tables
        - ✈1 to read user data
    - » Bottom-up
      - – Reduces the unacceptable overhead with top-down search
      - – Starts at the bottom level
        - ✈If the page is in memory, gets the physical page number
        - ✈Else, resorts to top-down search

Inverted page Table Organization

- Number of entries in the forward page table
  - ∗ Proportional to number of virtual pages
    - » Quite large for modern 64-bit processors
    - » Why?
      - – We use virtual page number as index
- To reduce the number of entries
  - ∗ Use physical page number as index
    - » Table grows with the size of memory
- Only one system-wide page table
  - ∗ VPN is hashed to generate index into the table
    - » Needs to handle collisions

Inverted page Table Organization (cont'd)



Hash anchor table is used to reduce collision frequency

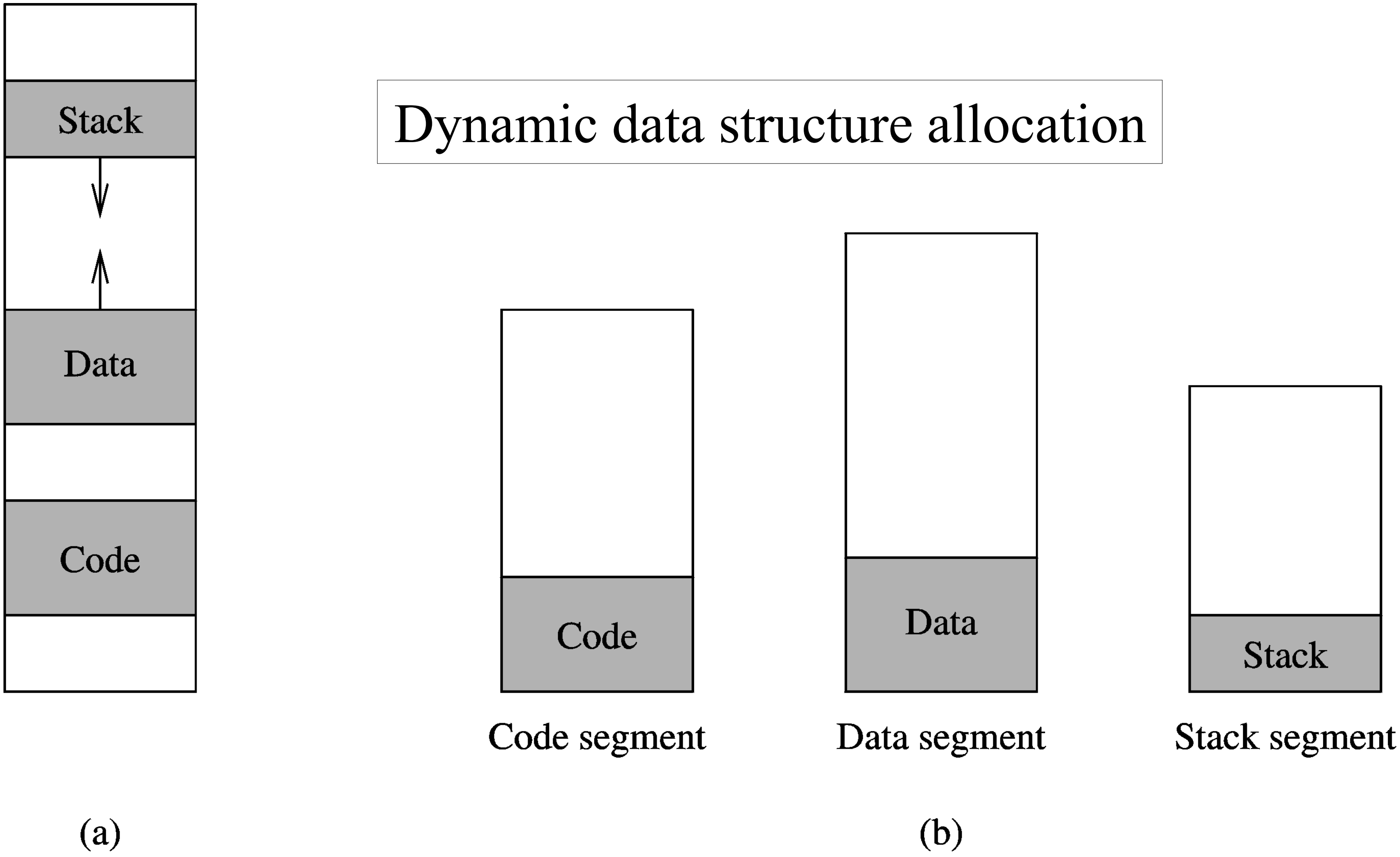
Segmentation

- Virtual address space is linear and 1-dimensional
  - ∗ Segmentation adds a second dimension
- Each process can have several virtual address spaces
  - ∗ These are called ***segments***
  - ∗ Example: Pentium
    - » Segments can be as large as 4 GB
- Address consists of two parts
  - ∗ Segment number
  - ∗ Offset within the segment

Segmentation (cont'd)

- Pentium and PowerPC support segmented-memory architecture
  - ∗ Paging is transparent to programmer
    - » Segmentation is not
      - – Pentium assembly programming makes it obvious
        - ✈Uses three segments: data, code, and stack
- Segmentation offers some key advantages
  - ∗ Protection
    - » Can be provided on segment-by-segment basis
  - ∗ Multiple address spaces
  - ∗ Sharing
    - » Segments can be shared among processes

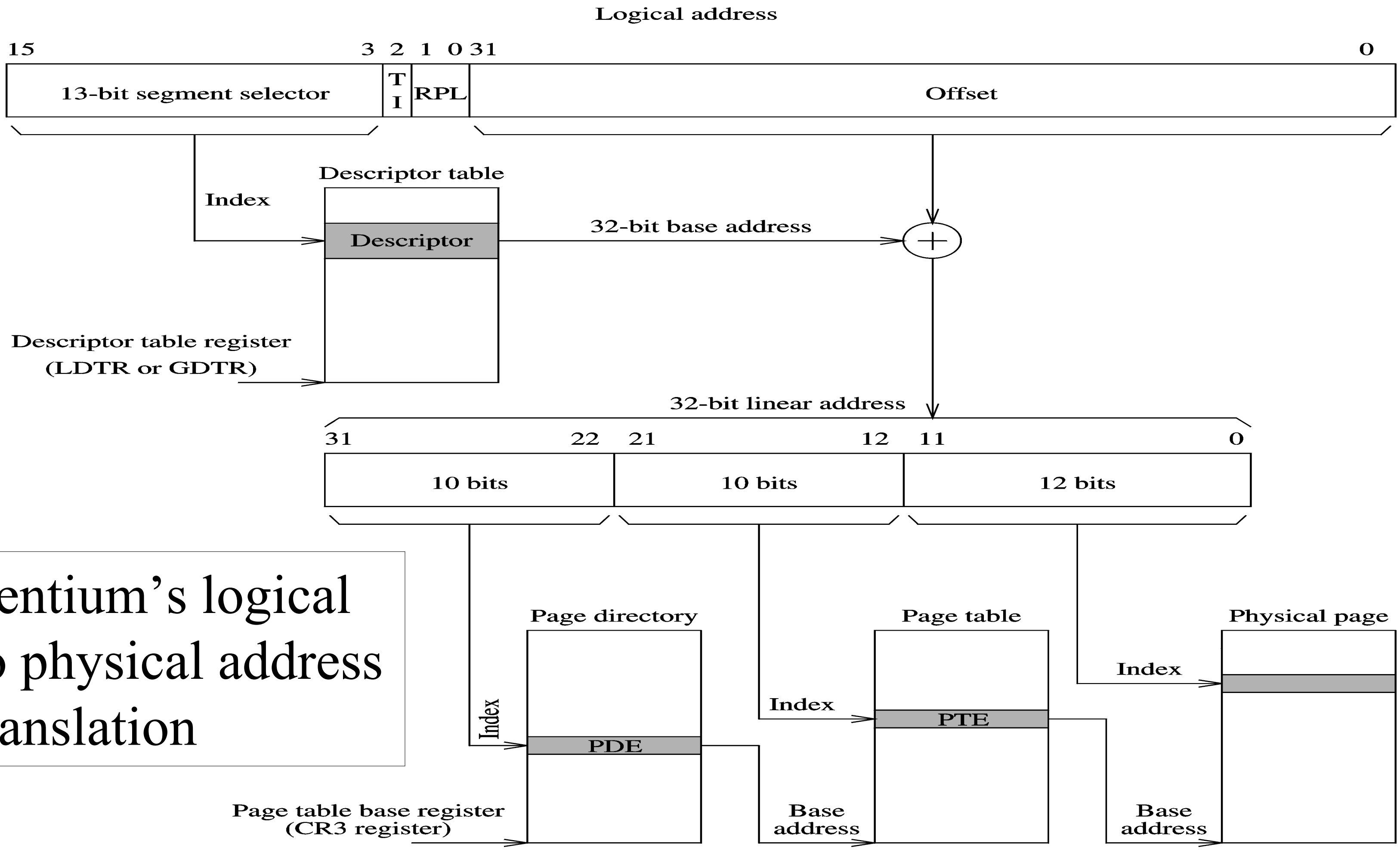
Segmentation (cont'd)



Example Implementations

- Pentium
  - ∗ Supports both paging and segmentation
    - » Paging can turned off
    - » Segmentation can be turned off
  - ∗ Segmentation translates a 48-bit logical address to 32-bit linear address
    - » If paging is used
      - – It translates the 32-bit linear address to 32-bit physical address
    - » If paging is off
      - – Linear address is treated as the physical address

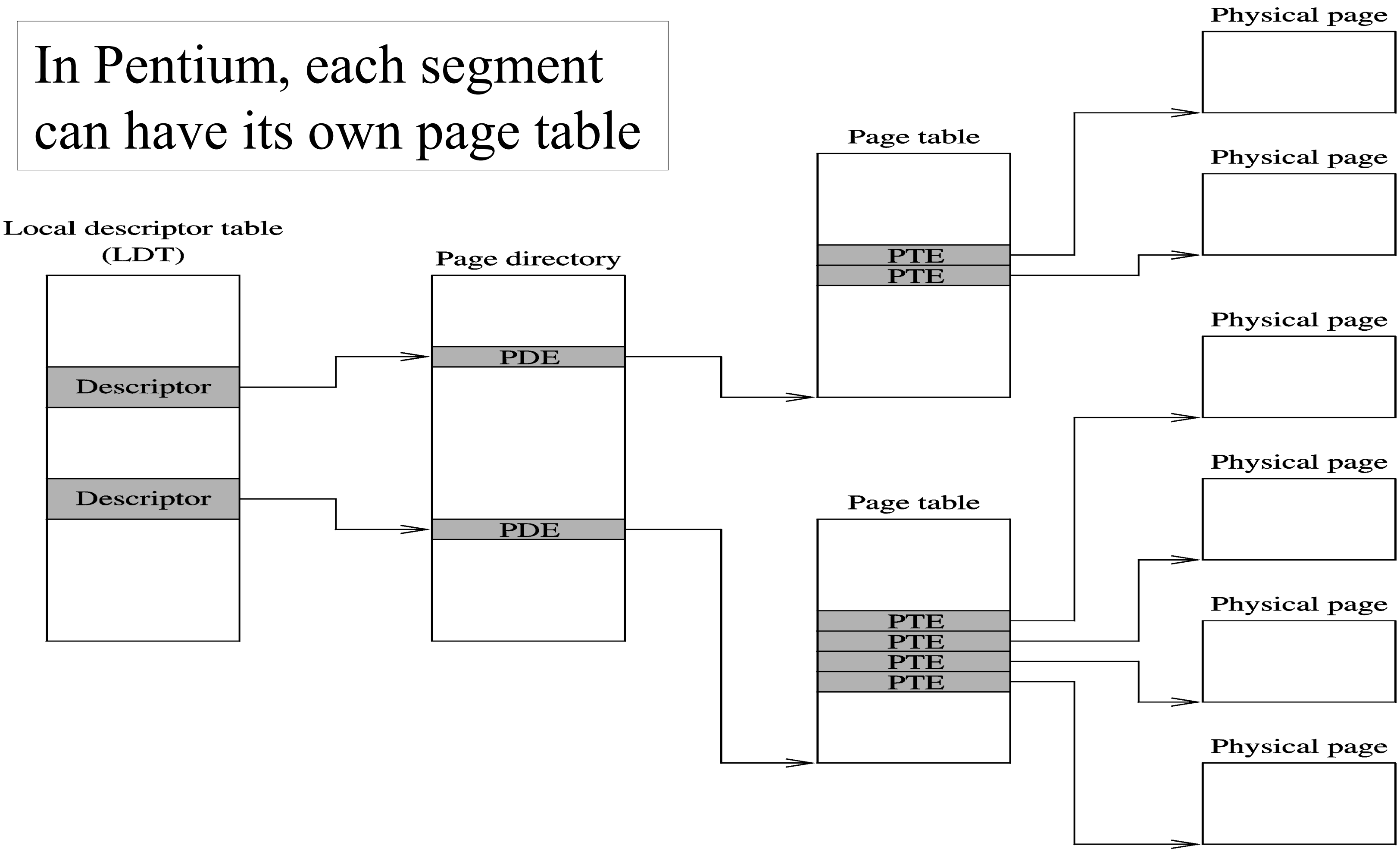
Example Implementations (cont'd)



Pentium's logical to physical address translation

Example Implementations (cont'd)

In Pentium, each segment can have its own page table



Example Implementations (cont'd)

31 12	11 9	8	7	6	5	4	3	2	1	0
20-bit physical page number	Avail	0	0	0	A	PCD	PWT	U	W	P

0 for 4-KB page size

(a) Page directory entry

Pentium's page directory & page table entry format

31 12	11 9	8	7	6	5	4	3	2	1	0
20-bit physical page number	Avail	0	0	D	A	PCD	PWT	U	W	P

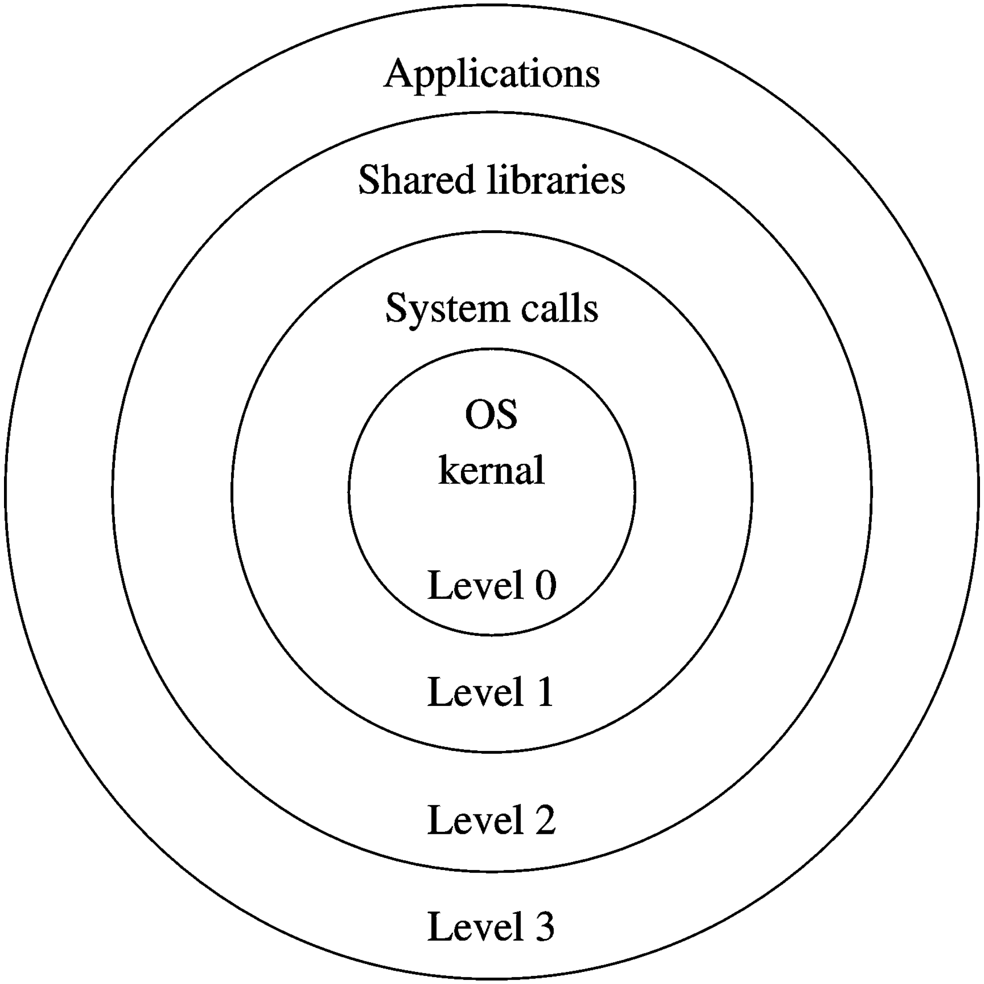
(b) Page table entry

Bit 0	Page present bit (P)
Bit 1	Writes permitted (W)
Bit 2	User/supervisor (U)
Bit 3	Page-level write-through (PWT)
Bit 4	Page-level cache-disable (PCD)
Bit 5	Page accessed (A)
Bit 6	0 in PDT dirty bit (D) in PTE
Bit 7	Page size in PDE (0 for 4 KB pages)

Avail:	Available for system programmer use
Shaded bits:	Reserved by Intel (must be zero)

Example Implementations (cont'd)

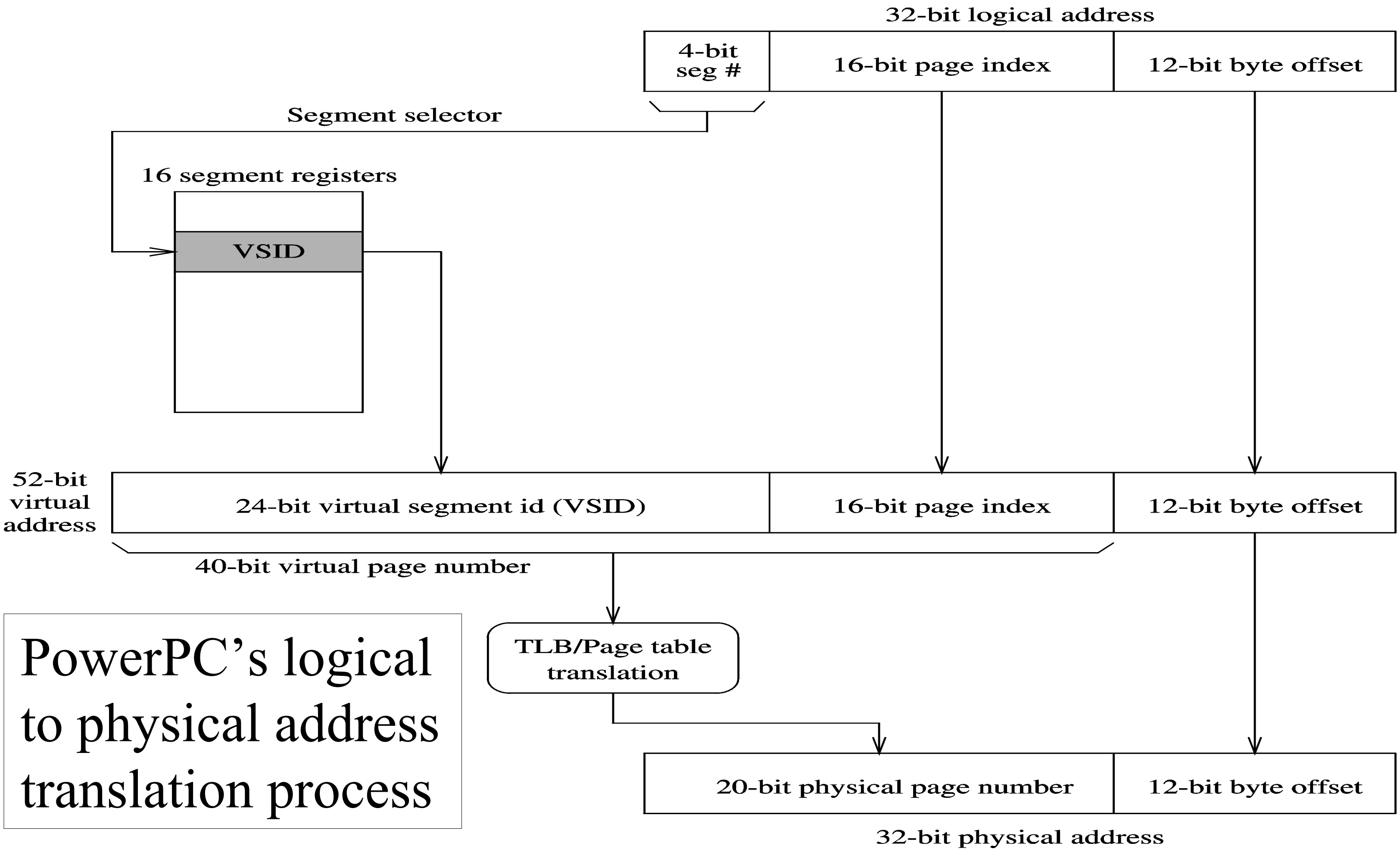
Pentium's protection rings



Example Implementations (cont'd)

- PowerPC
  - Supports both segmentation and paging
  - Logical and physical addresses are 32-bit long
  - 32-bit logical address consists of
    - 12-bit byte offset
    - 16-bit page index
    - 4-bit segment number
      - Selects one of 16 segment registers
      - Segment descriptor is a 24-bit virtual segment id (VSID)
  - 52-bit virtual address consists of
    - 40-bit VPN
    - 12-bit offset

Example Implementations (cont'd)

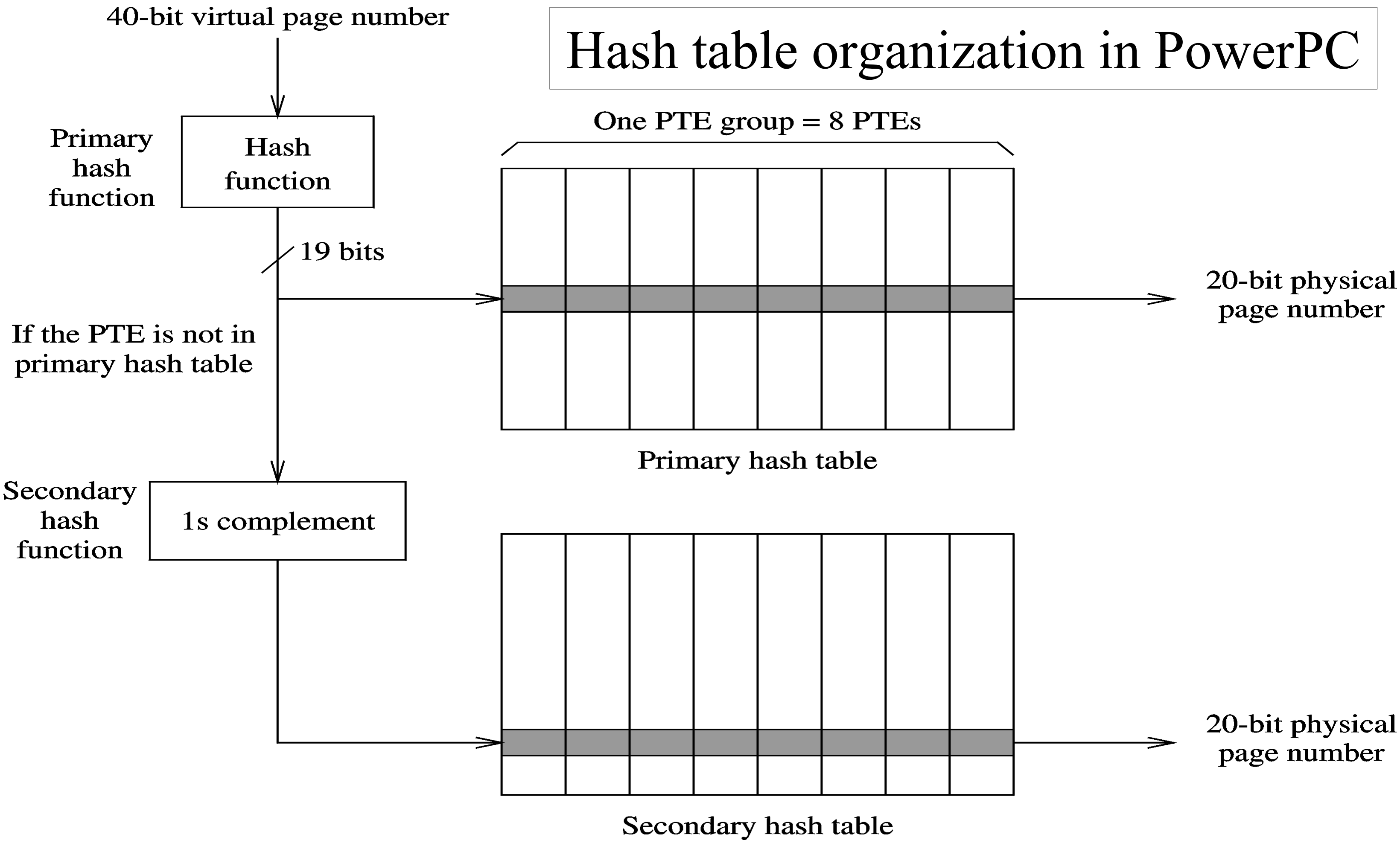


PowerPC's logical to physical address translation process

Example Implementations (cont'd)

- PowerPC uses inverted page table
  - ∗ Uses two hash tables
    - » Primary
      - – Uses 8-way associative page table entry groups
    - » Secondary
      - – 1s complement of the primary hash function
  - ∗ PTEs are 8-bytes wide
    - » Stores valid bit, reference bit, changed bit (i.e., dirty bit)
    - » W bit (write-through)
      - – W = 1: write-through policy
      - – W = 0: write-back policy
    - » I bit (cache inhibit)
      - – I = 1: cache inhibited (accesses main memory directly)

Example Implementations (cont'd)

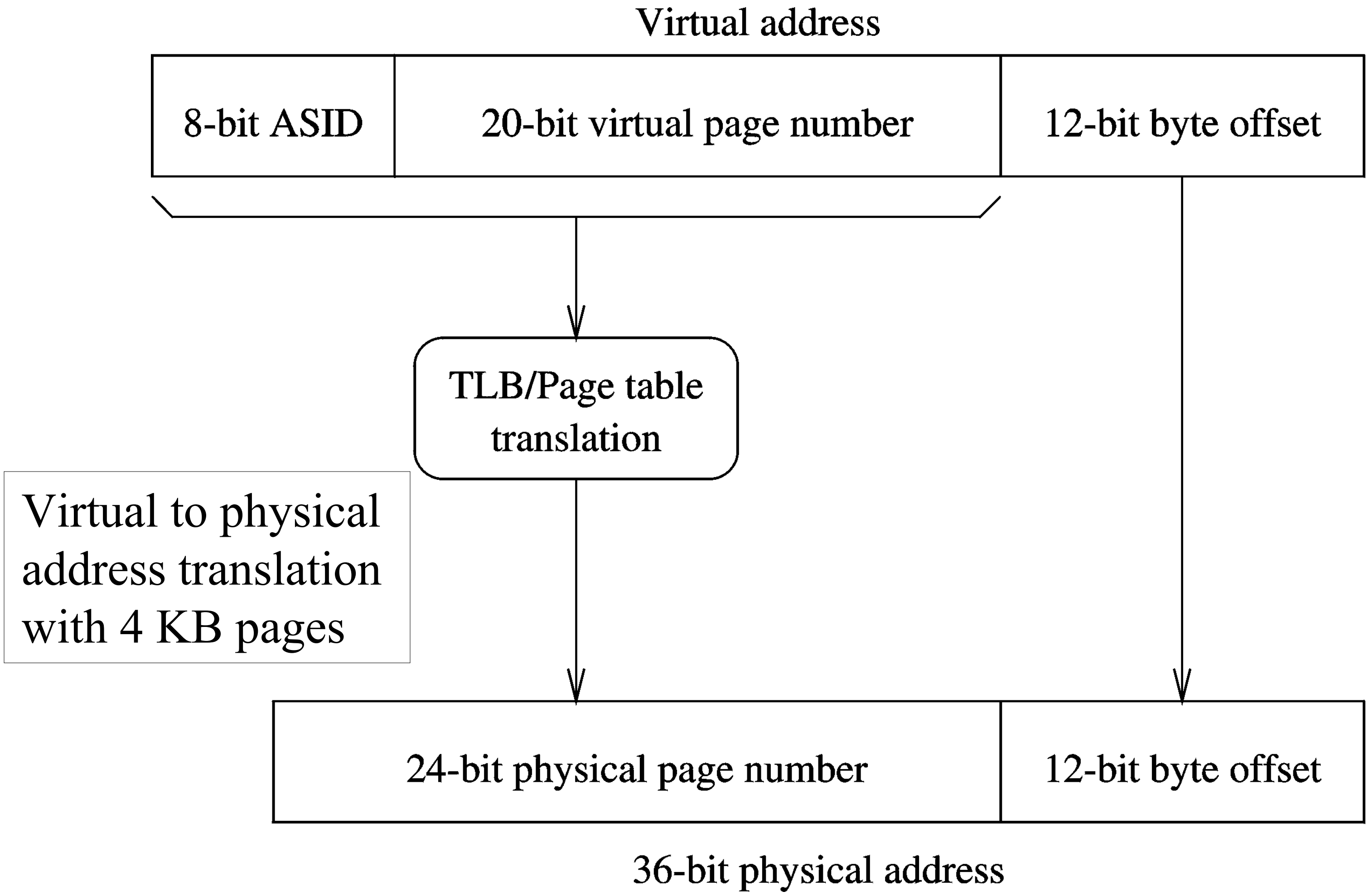


40-bit virtual page number
Hash table organization in PowerPC
Primary hash function
Hash function
One PTE group = 8 PTEs
19 bits
20-bit physical page number
If the PTE is not in primary hash table
Primary hash table
Secondary hash function
1s complement
20-bit physical page number
Secondary hash table

Example Implementations (cont'd)

- MIPS R4000
  - ∗ Segmentation is not used
    - » Uses address space identifiers (ASIDs) for
      - – Protection
      - – Virtual address space extension
  - ∗ 32-bit virtual address consists of
    - » 8-bit ASID
    - » 20-bit VPN
      - – Depends on the page size
    - » 12-bit offset
      - – Depends on the page size
      - – Supports pages from 4 KB to 16 MB

Example Implementations (cont'd)



Virtual address

8-bit ASID

20-bit virtual page number

12-bit byte offset

TLB/Page table translation

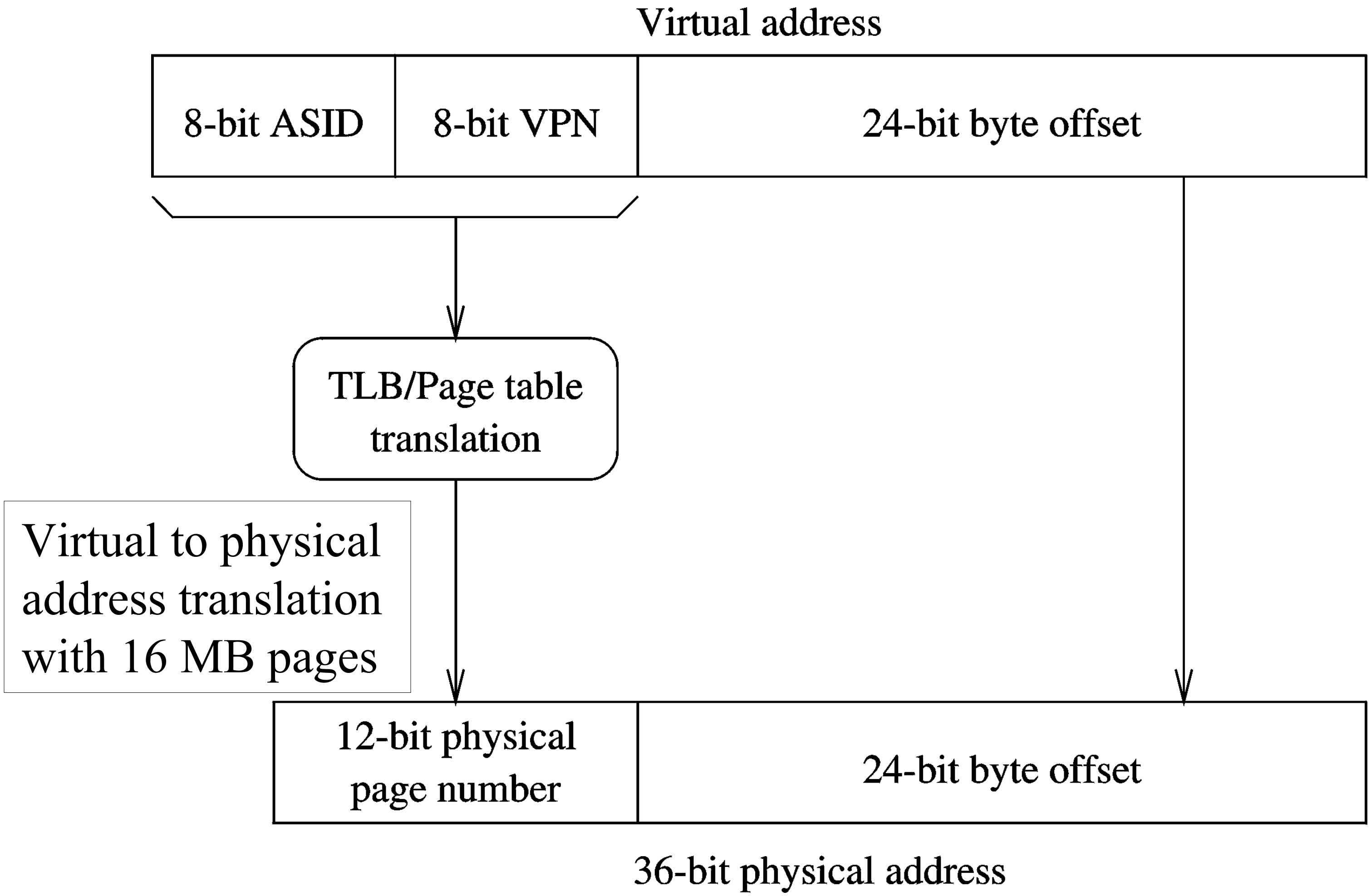
24-bit physical page number

12-bit byte offset

36-bit physical address

Virtual to physical address translation with 4 KB pages

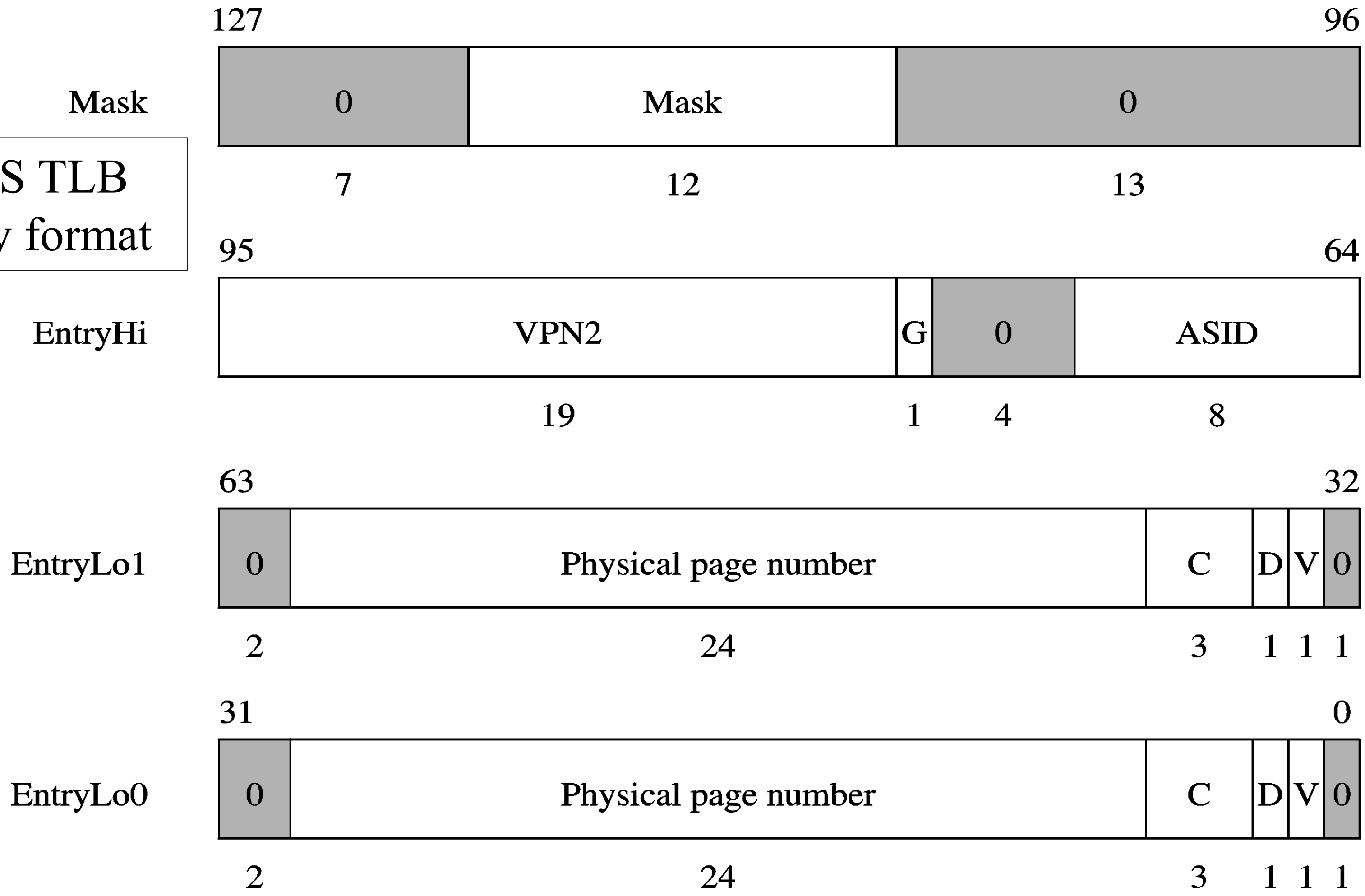
Example Implementations (cont’d)



Virtual to physical address translation with 16 MB pages

Example Implementations (cont’d)

MIPS TLB entry format



Mask	127		96
Field	0	Mask	0
Bits	7	12	13

EntryHi	95			64
Field	VPN2	G	0	ASID
Bits	19	1	4	8

EntryLo1	63					32
Field	0	Physical page number	C	D	V	0
Bits	2	24	3	1	1	1

EntryLo0	31					0
Field	0	Physical page number	C	D	V	0
Bits	2	24	3	1	1	1

Example Implementations (cont’d)

- MIPS R 4000 supports two TLB replacement policies
  - Random
    - Randomly selects an entry
  - Indexed
    - Selects the entry specified
- Two registers support these two policies
  - A Random register for the random policy
  - An Index register for the index policy
    - Specifies the entry to be replaced

Last slide