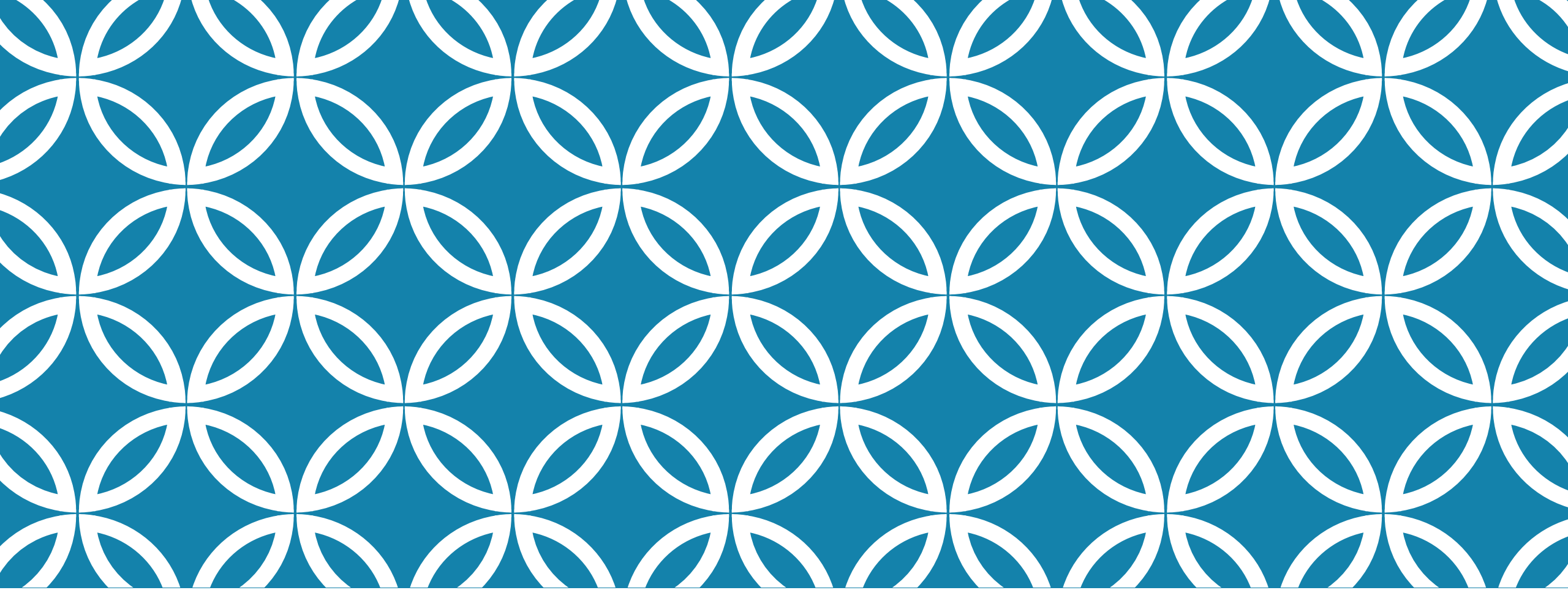


VIRTUAL MEMORY

NHÓM 1

Outline

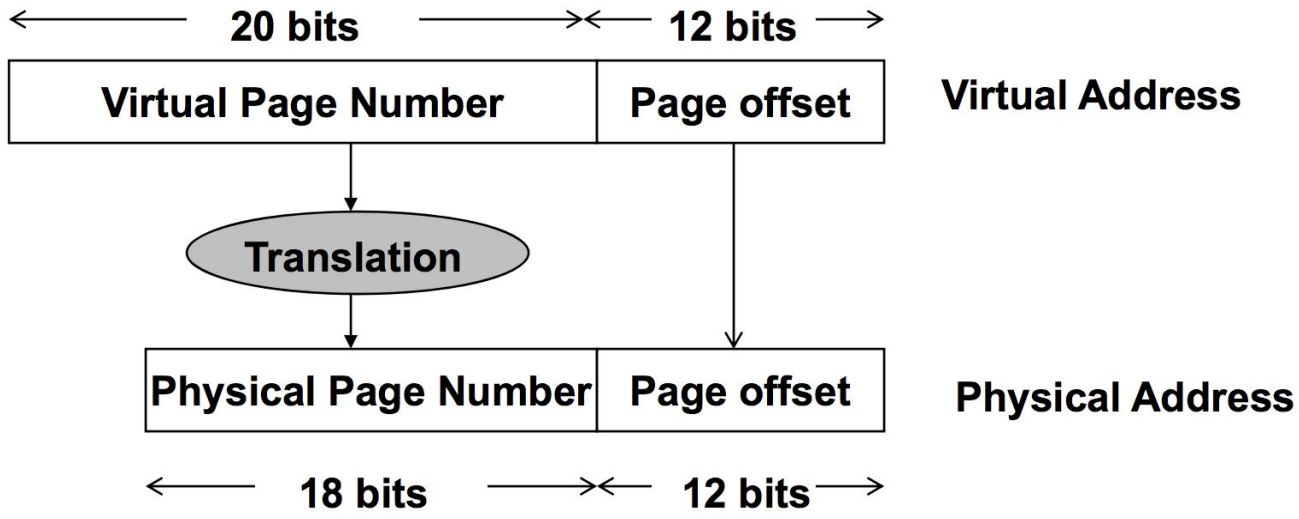
1. Replacement on Virtual Memory Miss
2. Virtual Memory Segmentation
3. Implementing Protection with Virtual Memory

Virtual and Physical Addresses

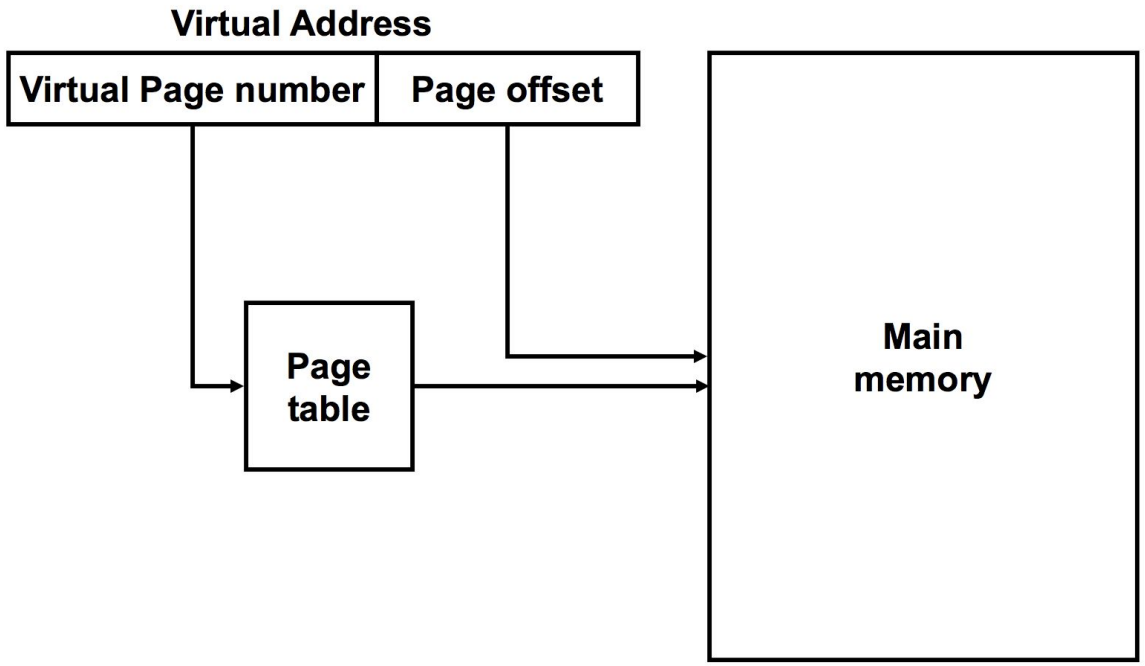
A virtual address consists of a virtual page number and a page offset.

The virtual page number gets translated to a physical page number.

The page offset is not changed



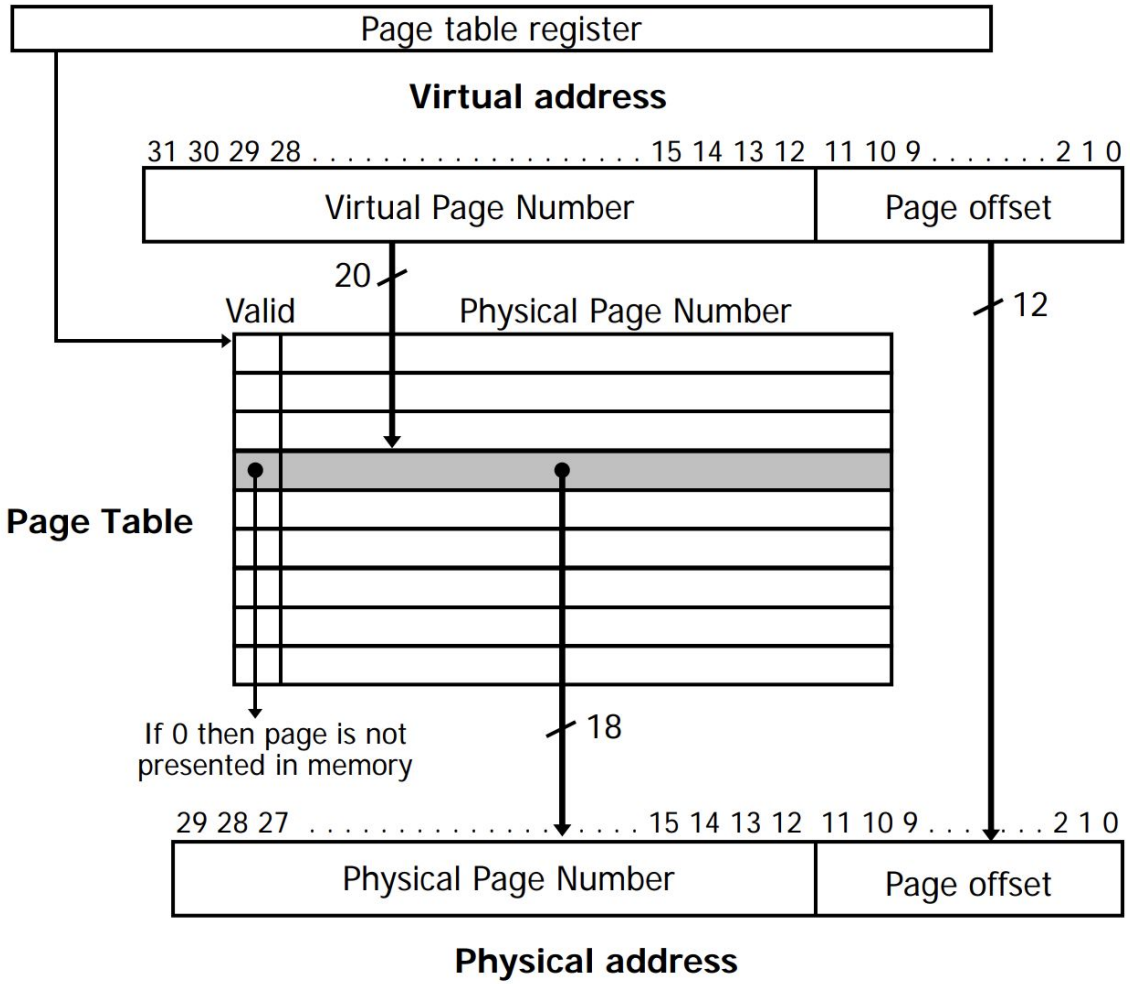
Address Translation



Address Translation with Page Tables

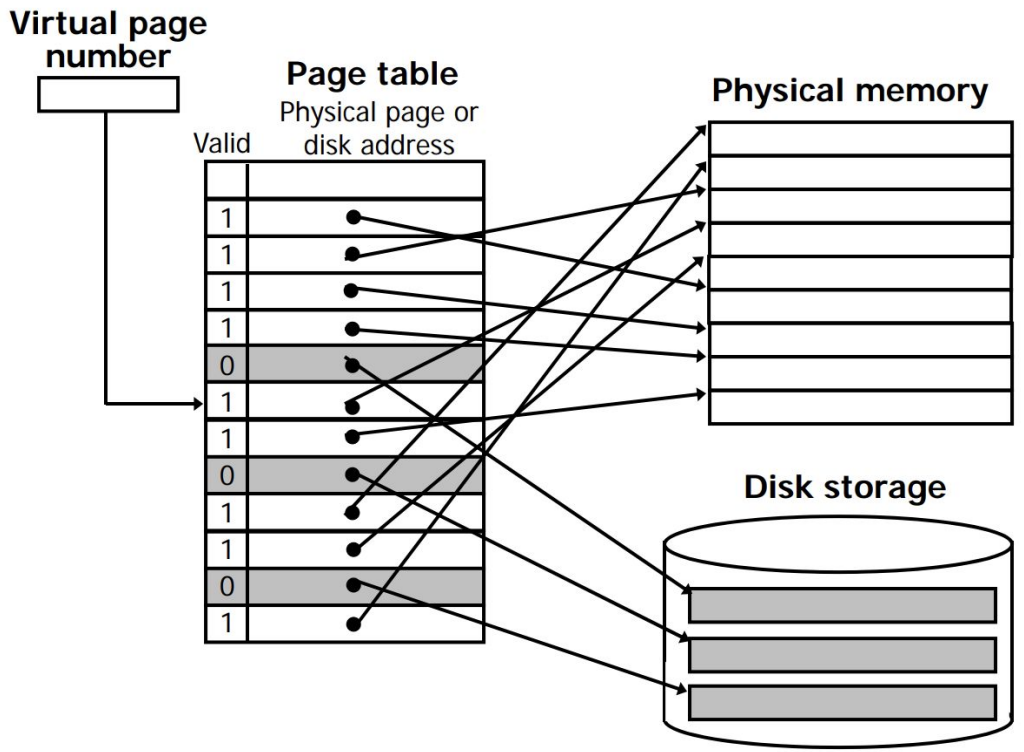
- A page table translates a virtual page number into a physical page number
- A page table register indicates the start of the page table
- The virtual page number is used as an index into the page table that contains
  - The physical page number
  - A valid bit that indicates if the page is present in main memory
  - A dirty bit to indicate if the page has been written
  - Protection information about the page (read only, read/write, etc.)
- Since page tables contain a mapping for every virtual page, no tags are required.

Page Table Diagram



Accessing Main Memory or Disk

- If the valid bit of the page table is zero, this means that the page is not in main memory.
- In this case, a page fault occurs, and the missing page is read in from disk.



Determining Page Table Size

- Assume
  - 32-bit virtual address
  - 30-bit physical address
  - 4 KB pages => 12 bit page offset
  - Each page table entry is one word (4 bytes)
- How large is the page table?
  - Virtual page number = 32 - 12 = 20 bits
  - Number of entries = number of pages = 2^20
  - Total size = number of entries x bytes/entry = 2^20 x 4 = 4 Mbytes
  - Each process running needs its own page table
- Since page tables are very large, they are almost always stored in main memory, which makes them slow

Block replacement

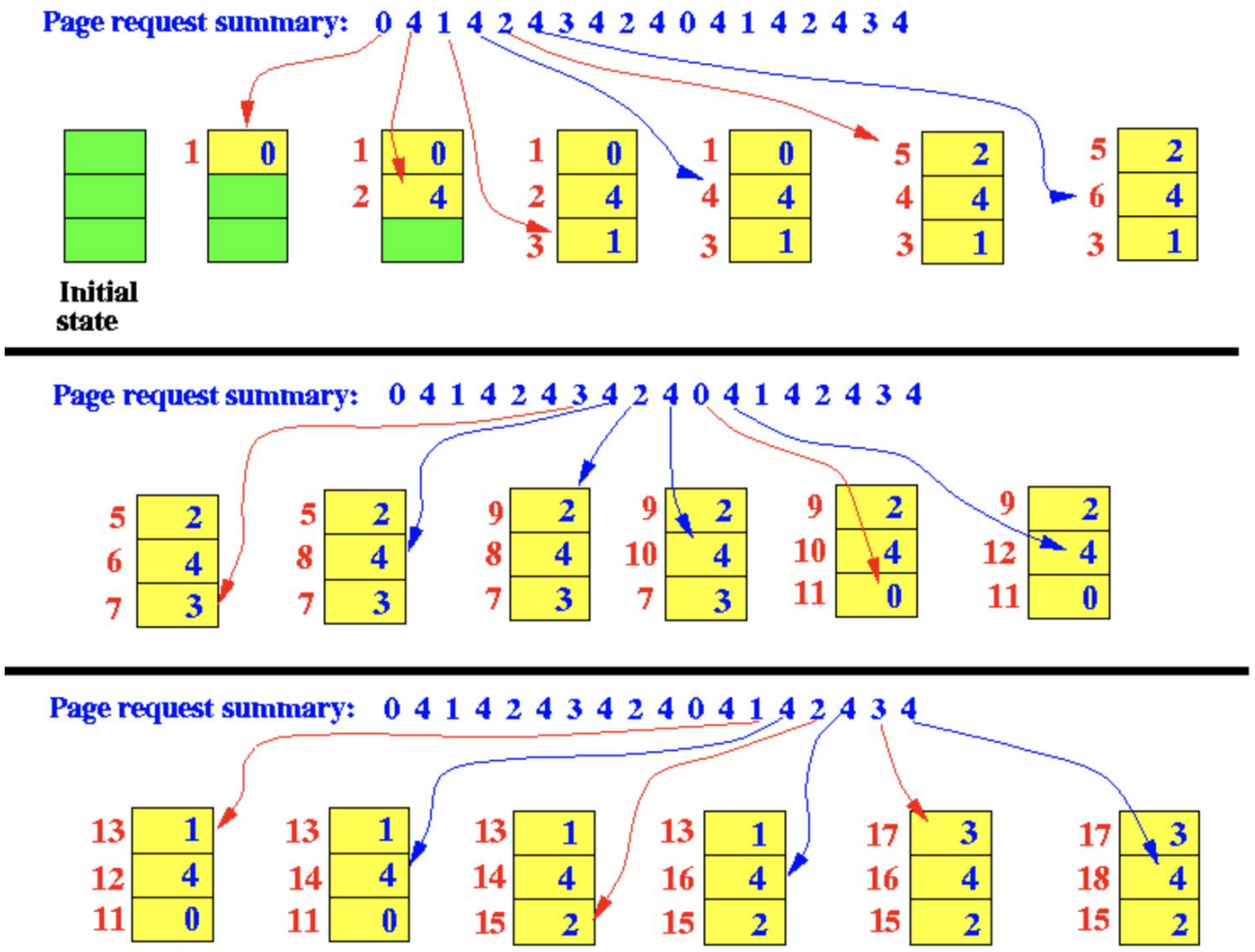
- Which block should be replaced on a virtual memory miss?
  - Want to reduce miss rate & can handle in software
  - Least Recently Used typically used
  - A typical approximation of LRU
    - Hardware set reference bits
    - OS record reference bits and clear them periodically
    - OS selects a page among least-recently referenced for replacement

Page Replacement Algorithms

- Least Recently Used (LRU):
  - selects the least recently used page for replacement
  - requires knowledge about past references, more difficult to implement (thread thru page table entries from most recently referenced to least recently referenced; when a page is referenced it is placed at the head of the list; the end of the list is the page to replace)
  - good performance, recognizes principle of locality

Page Replacement Algorithms

- LRU example:

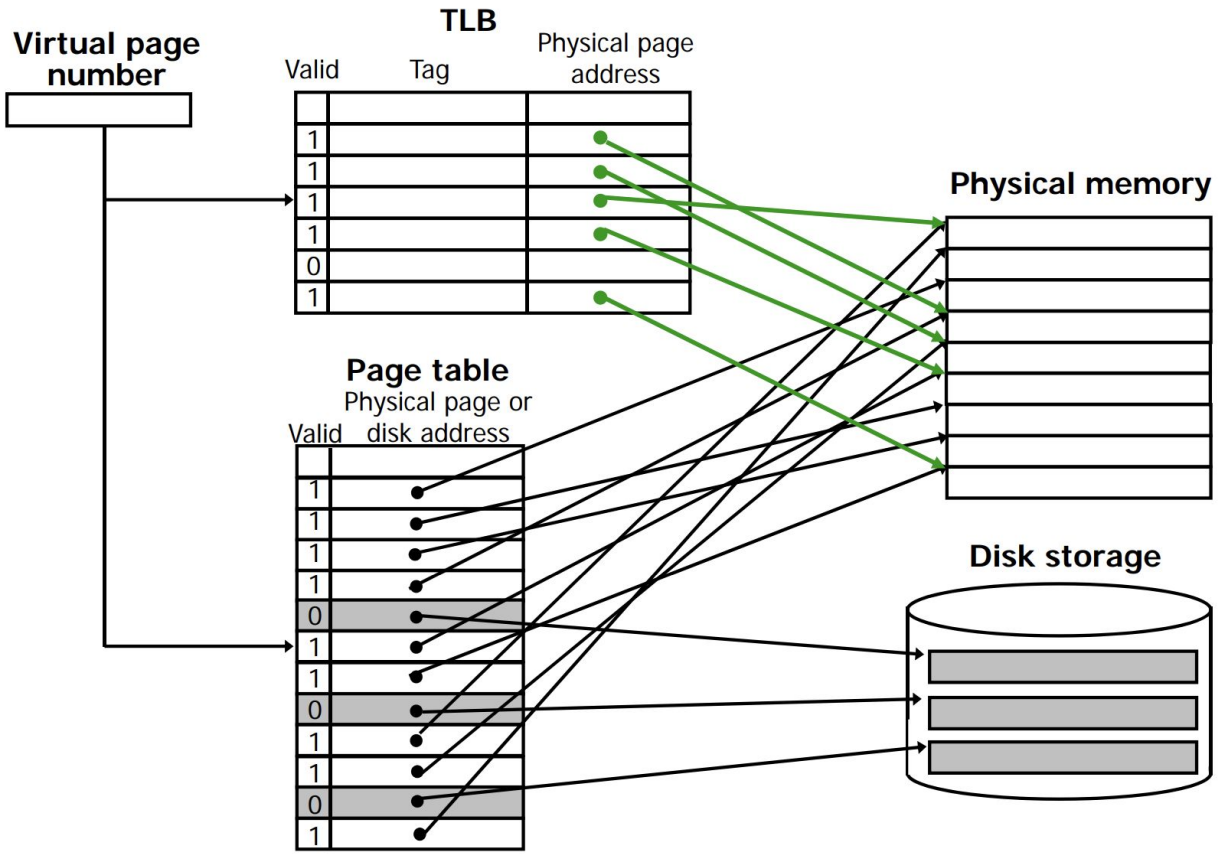


What happens on a write?

- Writing to disk is very expensive
- Write-through
  - Update both upper and lower levels
  - Simplifies replacement, but may require write buffer
- Write-back
  - Update upper level only
  - Update lower level when block is replaced
  - Need to keep more state
- Use a write-back strategy

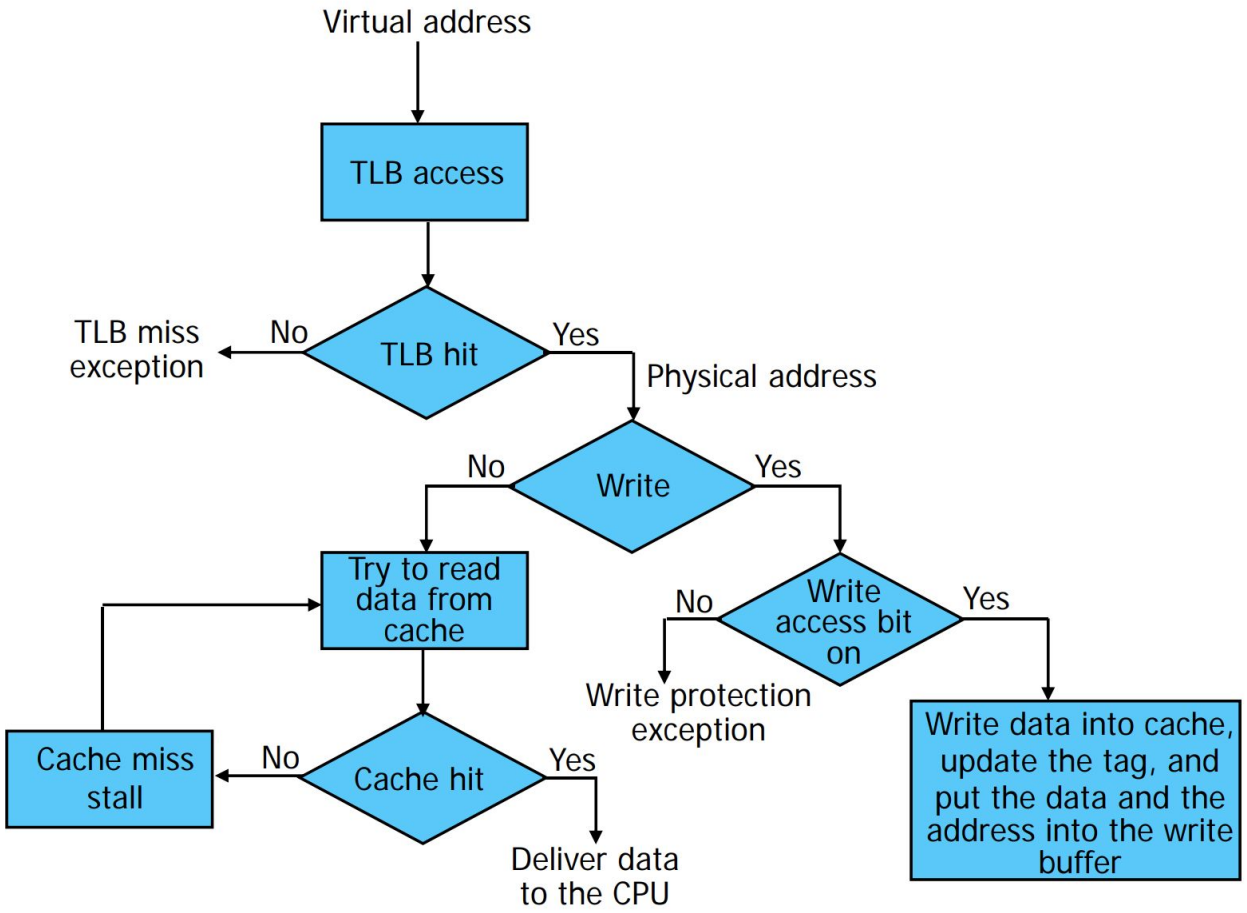
Translation-Lookaside Buffer (TLB)

- A TLB acts as a cache for the page table, by storing physical addresses of pages that have been recently accessed.



TLB and Cache Operation

- On a memory access, the following operations occur.



Handling TLB Misses and Page Faults

- When a TLB miss occurs either
  - Page is present in memory and update the TLB
    - occurs if valid bit of page table is set
  - Page is not present in memory and O.S. gets control to handle a page fault
- If a page fault occur, the operating system
  - Access the page table to determine the physical location of the page on disk
  - Chooses a physical page to replace - if the replaced page is dirty it is written to disk
  - Reads a page from disk into the chosen physical page in main memory.
- Since the disk access takes so long, another process is typically allowed to run during a page fault.

PAGE TABLES

VM “block” is called a page

Stores placement information

- Array of page table entries, indexed by virtual page number
- Page table register in CPU points to page table in physical memory

If page is present in memory

- PTE stores the physical page number
- Plus other status bits (referenced, dirty, …)

If page is not present

- PTE can refer to location in swap space on disk

REPLACEMENT AND WRITES

To reduce page fault rate, prefer least-recently used (LRU) replacement

- Reference bit (aka use bit) in PTE set to 1 on access to page
- Periodically cleared to 0 by OS
- A page with reference bit = 0 has not been used recently

REPLACEMENT AND WRITES

Each level in the hierarchy can use either write-through or write-back.

- **Write-through**: The information is written to both the block in the cache and the block in the lower level of the memory hierarchy (main memory for a cache).

- **Write-back**: The information is written only to the block in the cache. The modified block is written to the lower level of the hierarchy only when it is replaced.

REPLACEMENT AND WRITES

Disk writes take millions of cycles

- Block at once, not individual locations
- Write through is impractical
- Use write-back
- Dirty bit in PTE set when page is written

FAST TRANSLATION USING A TLB

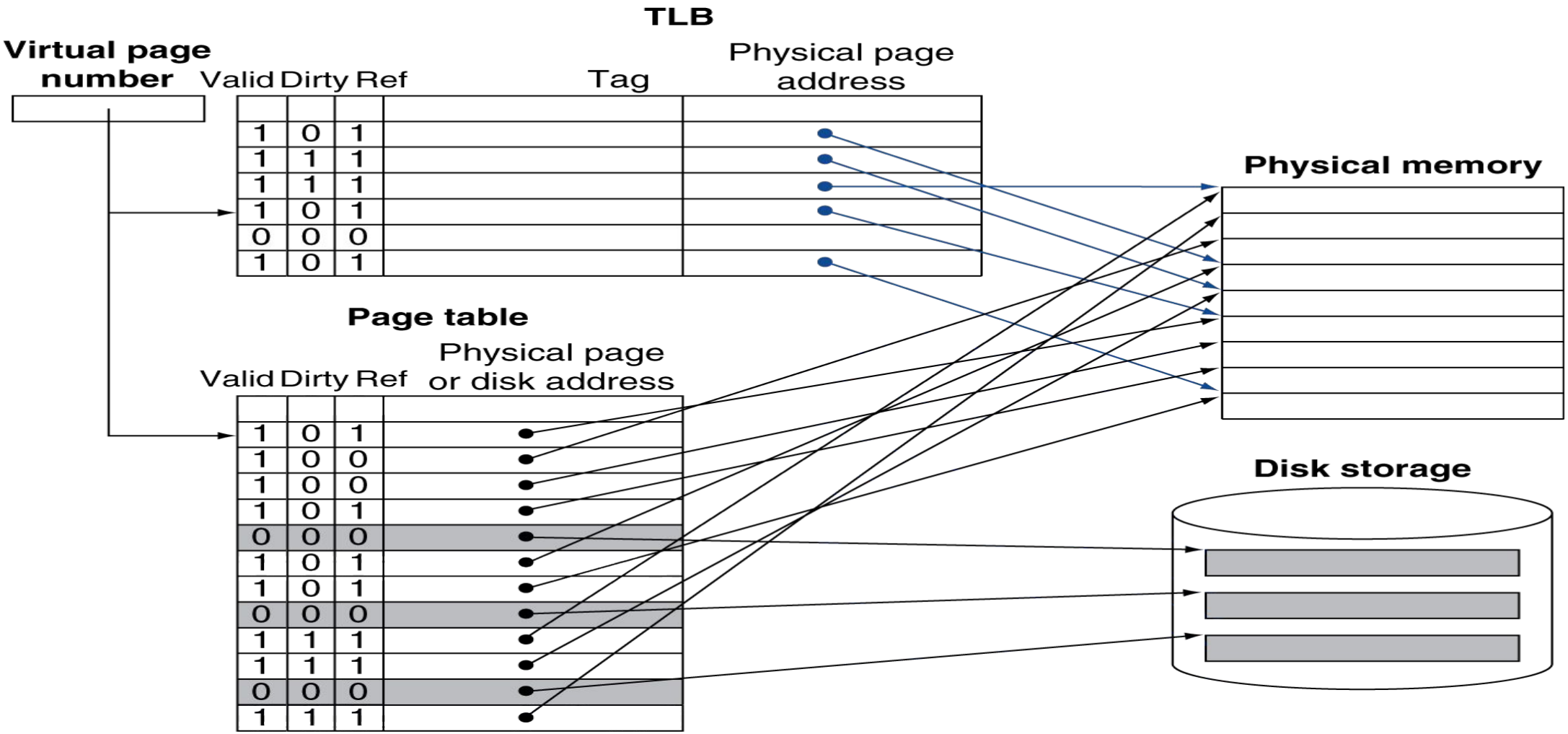
Address translation would appear to require extra memory references

- One to access the PTE
- Then the actual memory access

But access to page tables has good locality

- So use a fast cache of PTEs within the CPU
- Called a Translation Look-aside Buffer (TLB)
- Typical: 16–512 PTEs, 0.5–1 cycle for hit, 10–100 cycles for miss, 0.01%–1% miss rate
- Misses could be handled by hardware or software

FAST TRANSLATION USING A TLB



TLB MISSES

If page is in memory

- Load the PTE from memory and retry
- Could be handled in hardware
  - Can get complex for more complicated page table structures
- Or in software
  - Raise a special exception, with optimized handler

If page is not in memory (page fault)

- OS handles fetching the page and updating the page table
- Then restart the faulting instruction

TLB MISS HANDLE

TLB miss indicates

- Page present, but PTE not in TLB
- Page not preset

Must recognize TLB miss before destination register overwritten

- Raise exception

Handler copies PTE from memory to TLB

- Then restarts instruction
- If page not present, page fault will occur

TLB FAULT HANDLE

Use faulting virtual address to find PTE

Locate page on disk

Choose page to replace

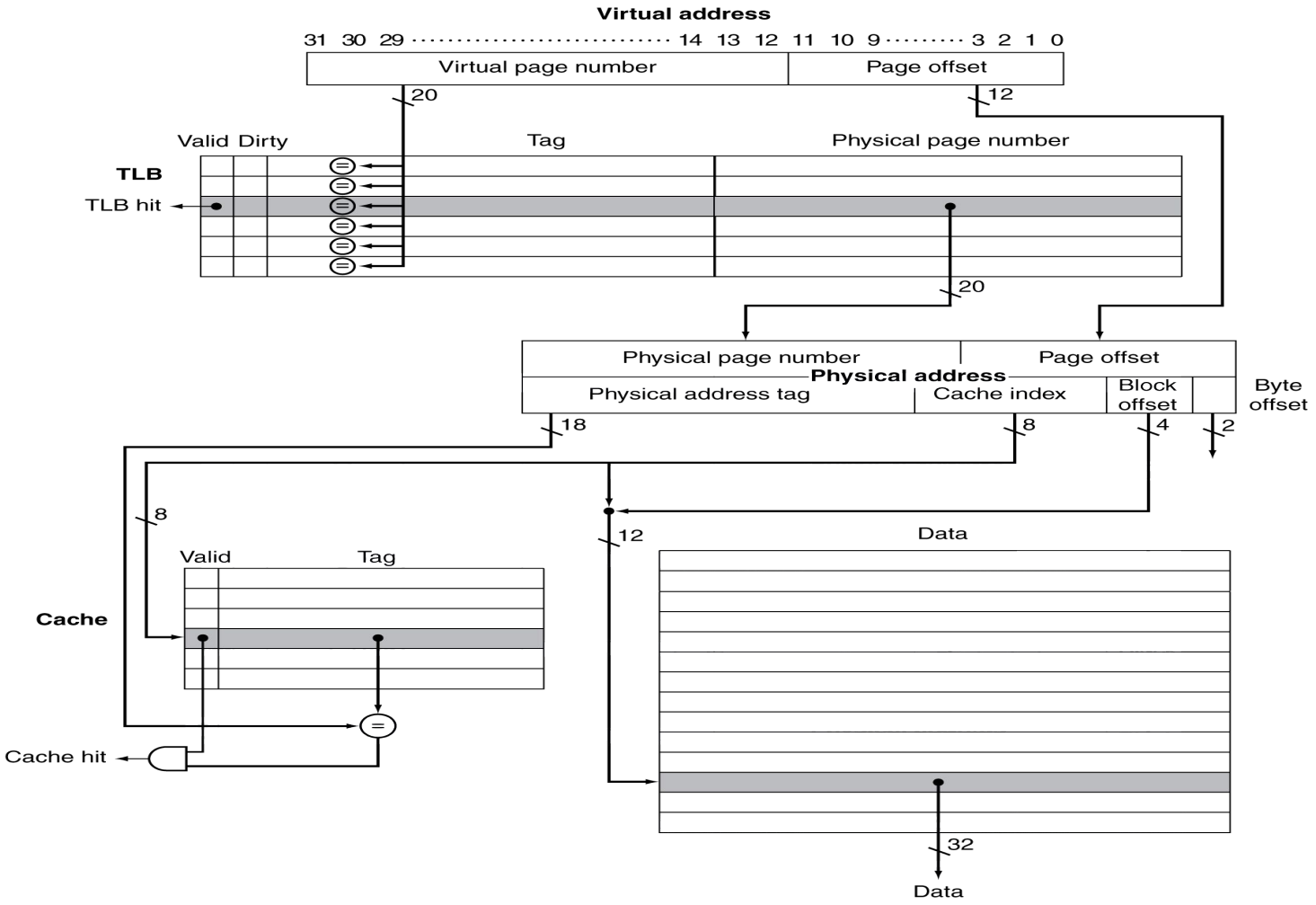
- If dirty, write to disk first

Read page into memory and update page table

Make process runnable again

- Restart from faulting instruction

TLB AND CACHE INTERACTION



If cache tag uses physical address

- Need to translate before cache lookup

Alternative: use virtual address tag

- Complications due to aliasing
  - Different virtual addresses for shared physical address

Segmentation

Segmentation is another non-contiguous memory allocation scheme like paging.

In segmentation, process isn't divided indiscriminately into mounted(fixed) size pages. It is variable size partitioning theme.

Secondary and main memory are not divided into partitions of equal size.

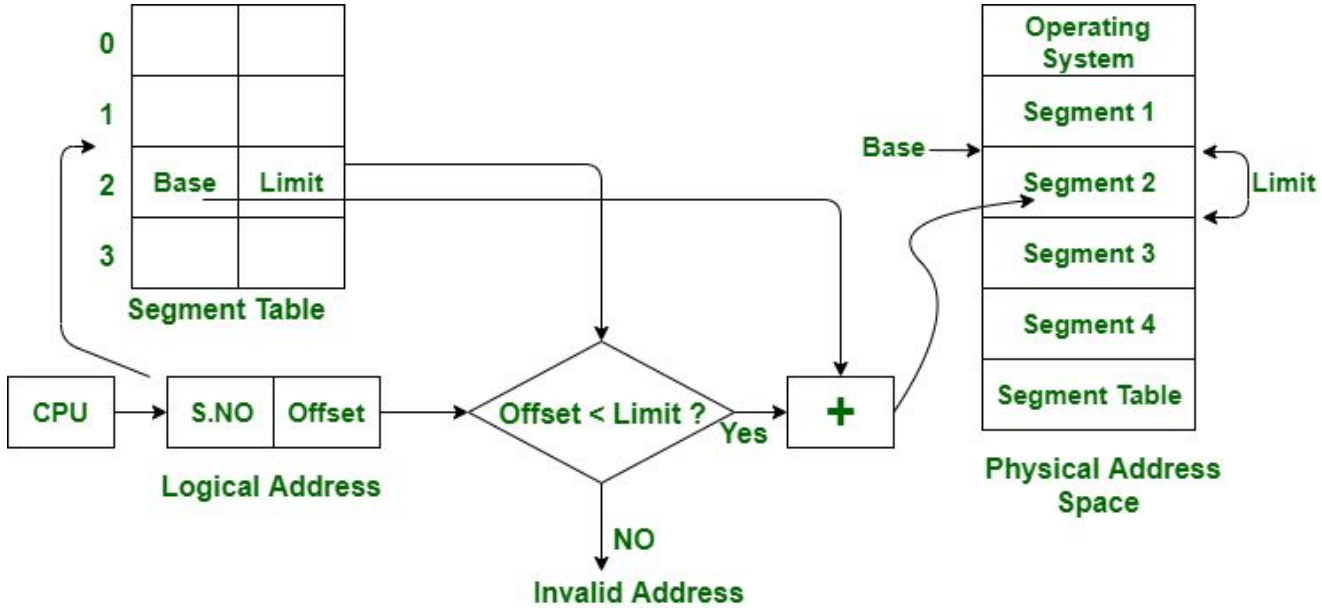
The partitions of secondary memory area unit known as segments. The details concerning every segment are hold in a table known as segmentation table.

Segment table contains two main data concerning segment, one is Base, which is the bottom address of the segment and another is Limit, which is the length of the segment.

Segmentation

CPU generates logical address that contains Segment number and Segment offset.

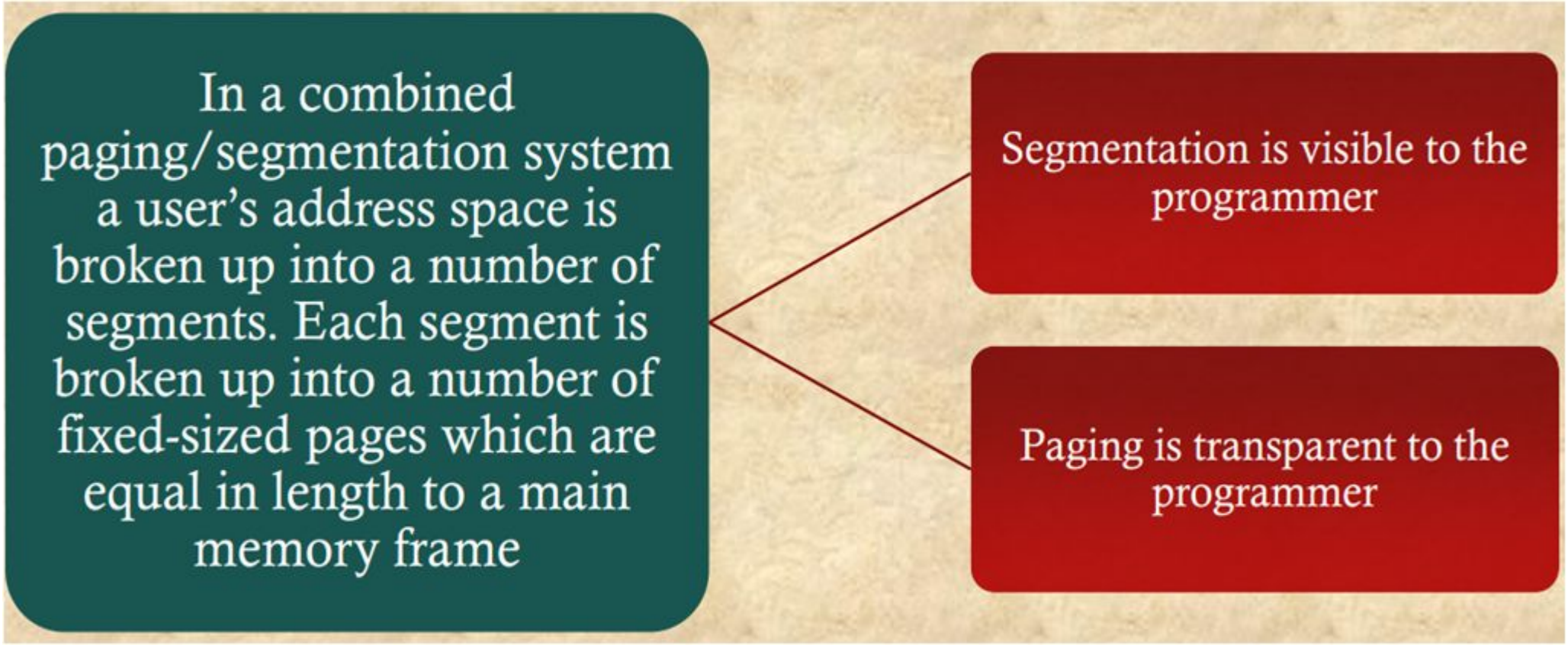
If the segment offset is a smaller amount than the limit then the address called valid address otherwise it throws miscalculation because the address is invalid.



Segmentation vs Paging

No	Paging	Segmentation
1	program is divided into fixed size pages	program is divided into variable size sections
2	operating system is accountable.	For segmentation compiler is accountable.
3	Page size is determined by hardware.	Here, the section size is given by the user.
4	It is faster in the comparison of segmentation.	Segmentation is slow
5	Paging could result in internal fragmentation	Segmentation could result in external fragmentation
6	In paging, logical address is split into page number and page offset	Here, logical address is split into section number and section offset
7	Paging comprises a page table which encloses the base address of every page.	While segmentation also comprises the segment table which encloses segment number and segment offset
8	Page table is employed to keep up the page data	Section Table maintains the section data

How about is combination of segmentation and paging?



Address translation in seg/pag sys

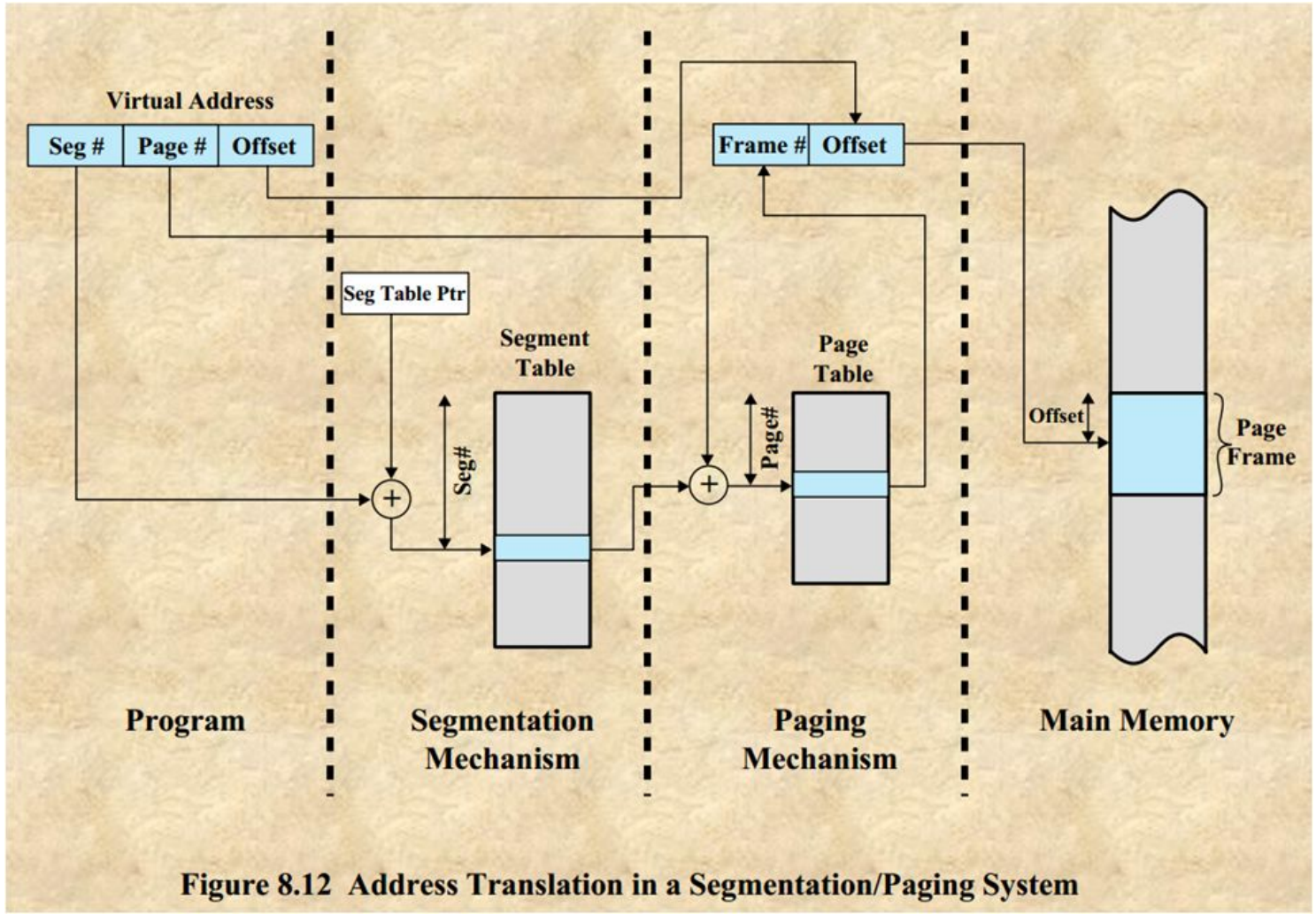
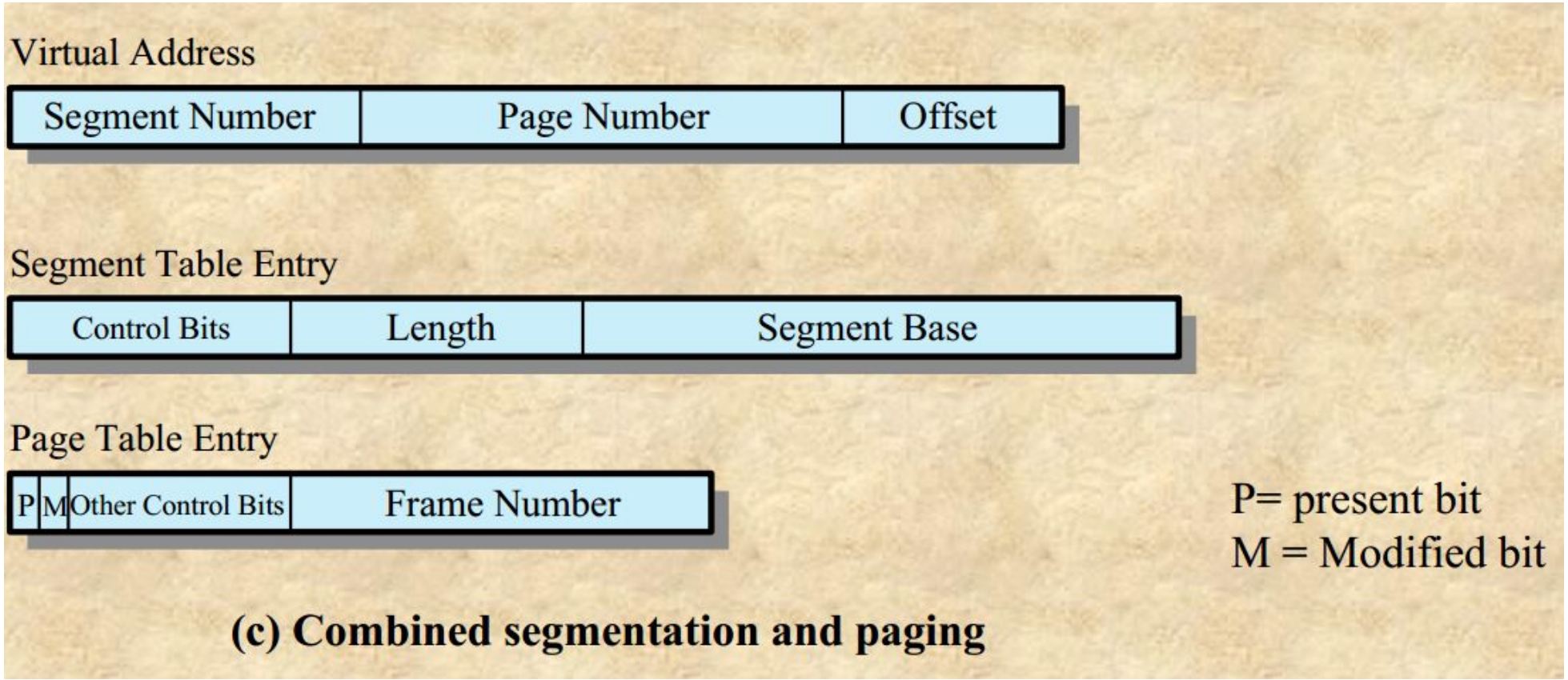


Figure 8.12 Address Translation in a Segmentation/Paging System

Combined segmentation and paging



How is memory protected in virtual memory system

- **Memory protection** is a way to control memory access rights on a computer, and is a part of most modern instruction set architectures and operating systems.
- The main purpose of memory protection is to prevent a process from accessing memory that has not been allocated to it.
- **Virtual memory system** is responsible for managing the disk and memory and providing the protection of the system.

How is memory protected in virtual memory system

Virtual memory system Provides many facilities:

- The property of placing Many programs in physical memory at the same time (Multiprogramming).
- The ability of executing the programs whose sizes are larger than the size of the physical memory.
- Relocating the programs at the execution time.
- Sharing the programs and data.
- **Protection.**

How is memory protected in virtual memory system

Protection:

- Protecting some regions in memory against operations such as writing, reading, or executing.
- Protection against unexpected errors during program execution such as:
  - Access to a page or segment not present in memory.

How is memory protected in virtual memory system

- Virtual memory mechanism responsible to detect any activity against the protection criteria.
- When detects an incorrect activity is in progress which may fail the operation of the system, it suspends that activity and transfers the control of the cpu to the operating system.

How is memory protected in virtual memory system

Paged virtual memory:

- It is impossible for an application to access a page that has not been explicitly allocated to it, because every memory address either points to a page allocated to that application, or generates an interrupt called a **page fault**.
- Unallocated pages, and pages allocated to any other application, do not have any addresses from the application point of view.

Protection with Paging

- The paging process is protected by the concept of insertion of an additional bit called VALID/INVALID BIT
- Consider a 14 bit address space= 2^14=16383 bytes
- Let us set an address limit of 10468
- If five process are defined within this address space (P0-P4), it is considered as a Valid bit
- Process P5 has started before 10468, so that alone is considered
- The remaining processes are considered as Invalid In this way the pages are internally fragmented
- This is how Paging is protected

Protection with Segments

- Segmentation lends itself to the implementation of protection and sharing policies
- Each entry has a base address and length so inadvertent memory access can be controlled
- Sharing can be achieved by segments referencing multiple processes
- Two processes that need to share access to a single segment would have the same segment name and address in their segment tables.

Thank you

Q&A