Operating System

Chapter 8. Virtual Memory

Lynn Choi

School of Electrical Engineering

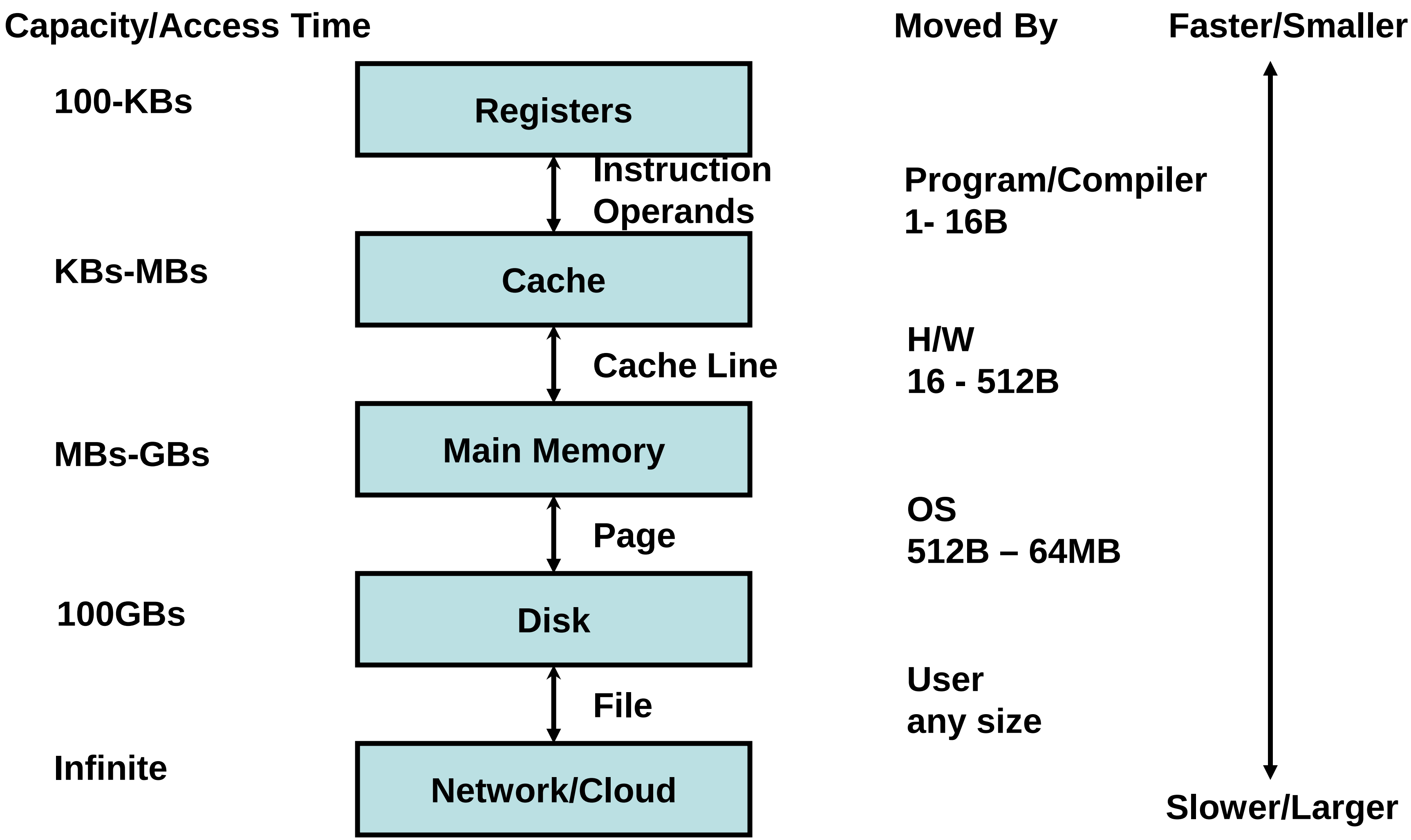
Memory Hierarchy

- **Motivated by**
  - Principles of Locality
  - Speed vs. size vs. cost tradeoff

- **Locality principle**
  - Spatial Locality: nearby references are likely
    - Example: arrays, program codes
    - Access a *block* of contiguous words
  - Temporal Locality: the same reference is likely to occur soon
    - Example: loops, reuse of variables
    - Keep recently accessed data to closer to the processor

- **Speed vs. Size tradeoff**
  - Bigger memory is slower: SRAM - DRAM - Disk - Tape
  - Fast memory is more expensive

Levels of Memory Hierarchy



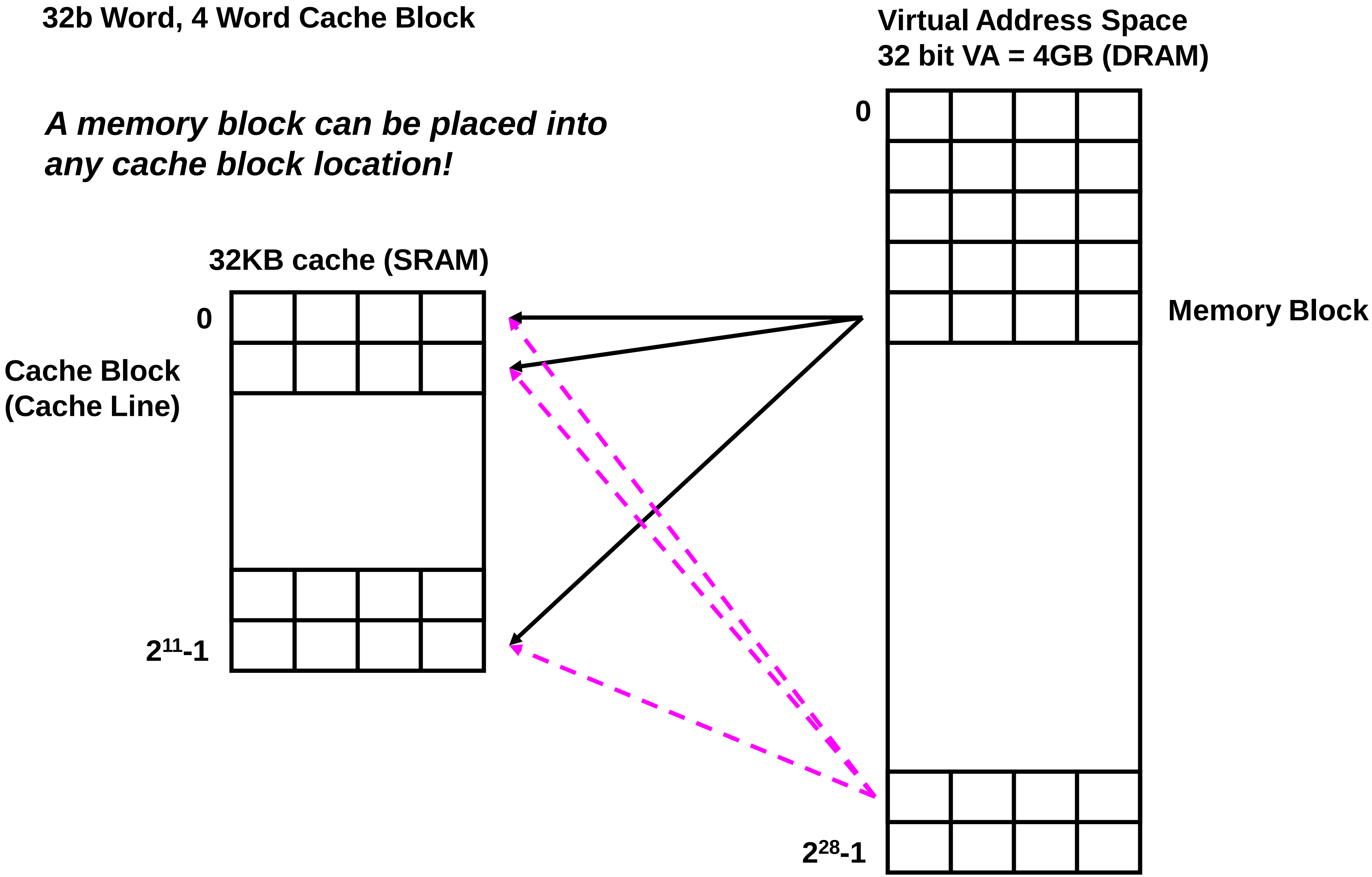
Cache

- **A small but fast memory located between processor and main memory**
- **Benefits**
  - Reduce load latency
  - Reduce store latency
  - Reduce bus traffic (on-chip caches)
- **Cache Block Allocation (When to place)**
  - On a read miss
  - On a write miss
    - Write-allocate vs. no-write-allocate
- **Cache Block Placement (Where to place)**
  - Fully-associative cache
  - Direct-mapped cache
  - Set-associative cache

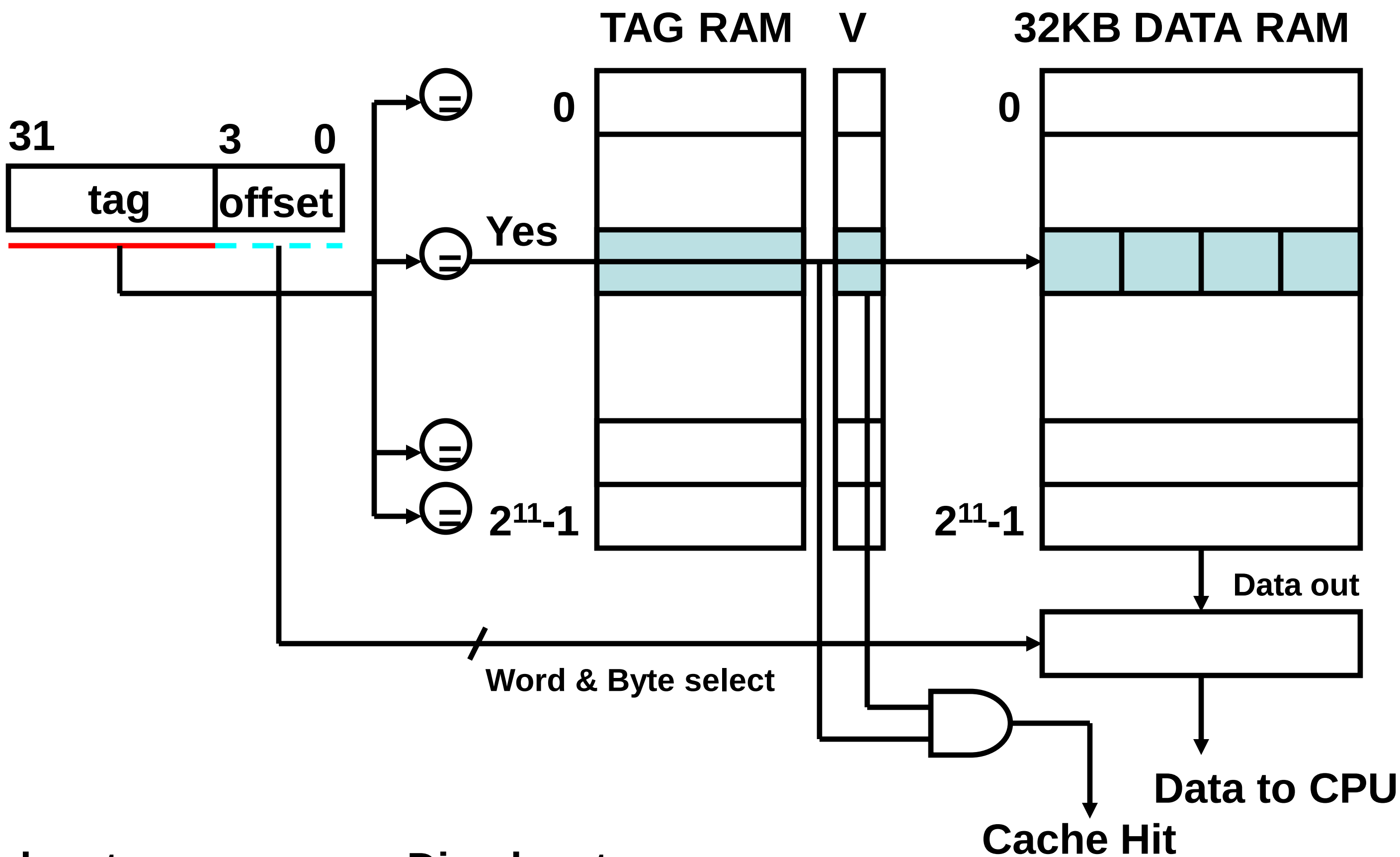
Fully Associative Cache

32b Word, 4 Word Cache Block

A memory block can be placed into any cache block location!



Fully Associative Cache

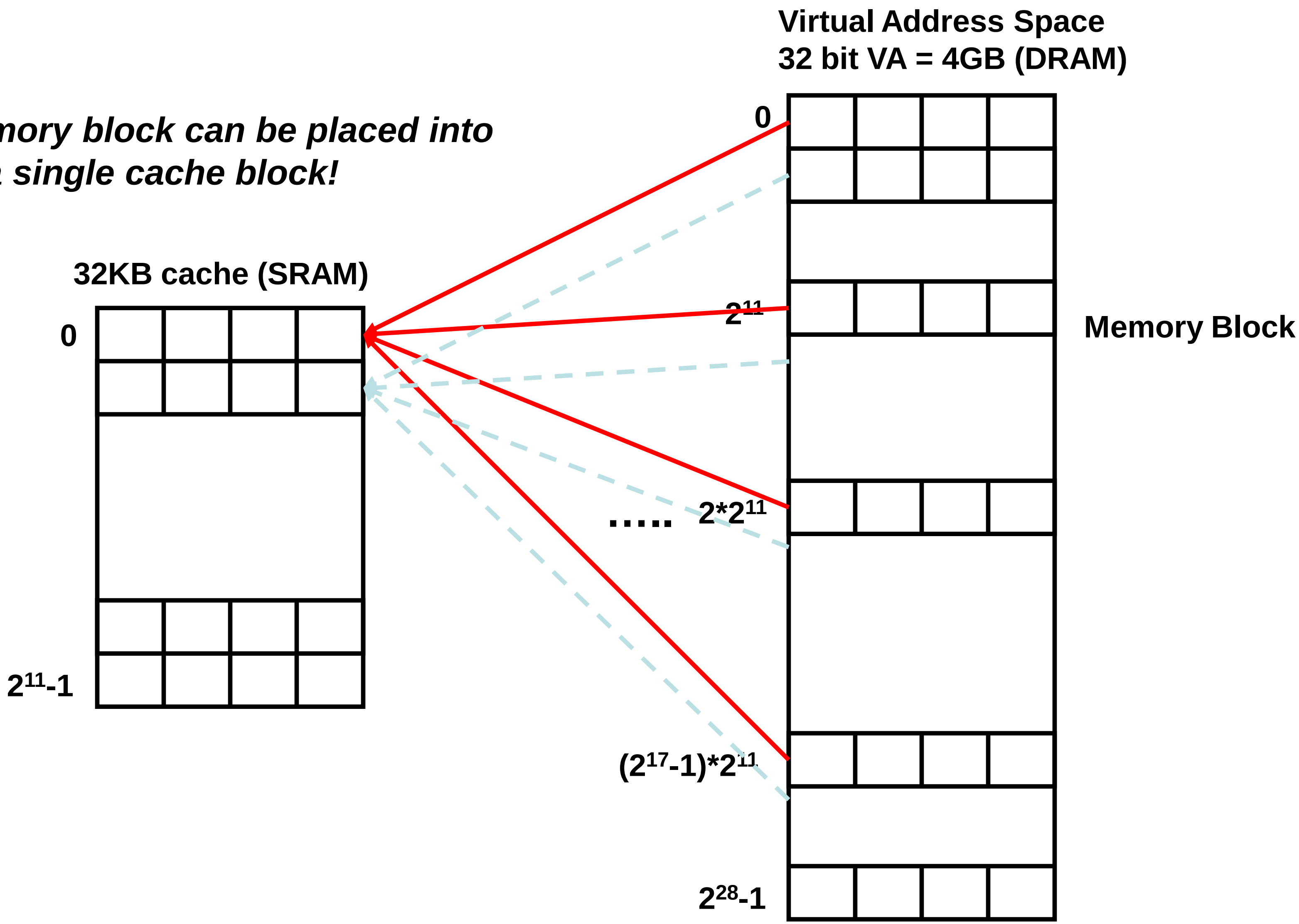


Advantages
1. High hit rate
2. Fast

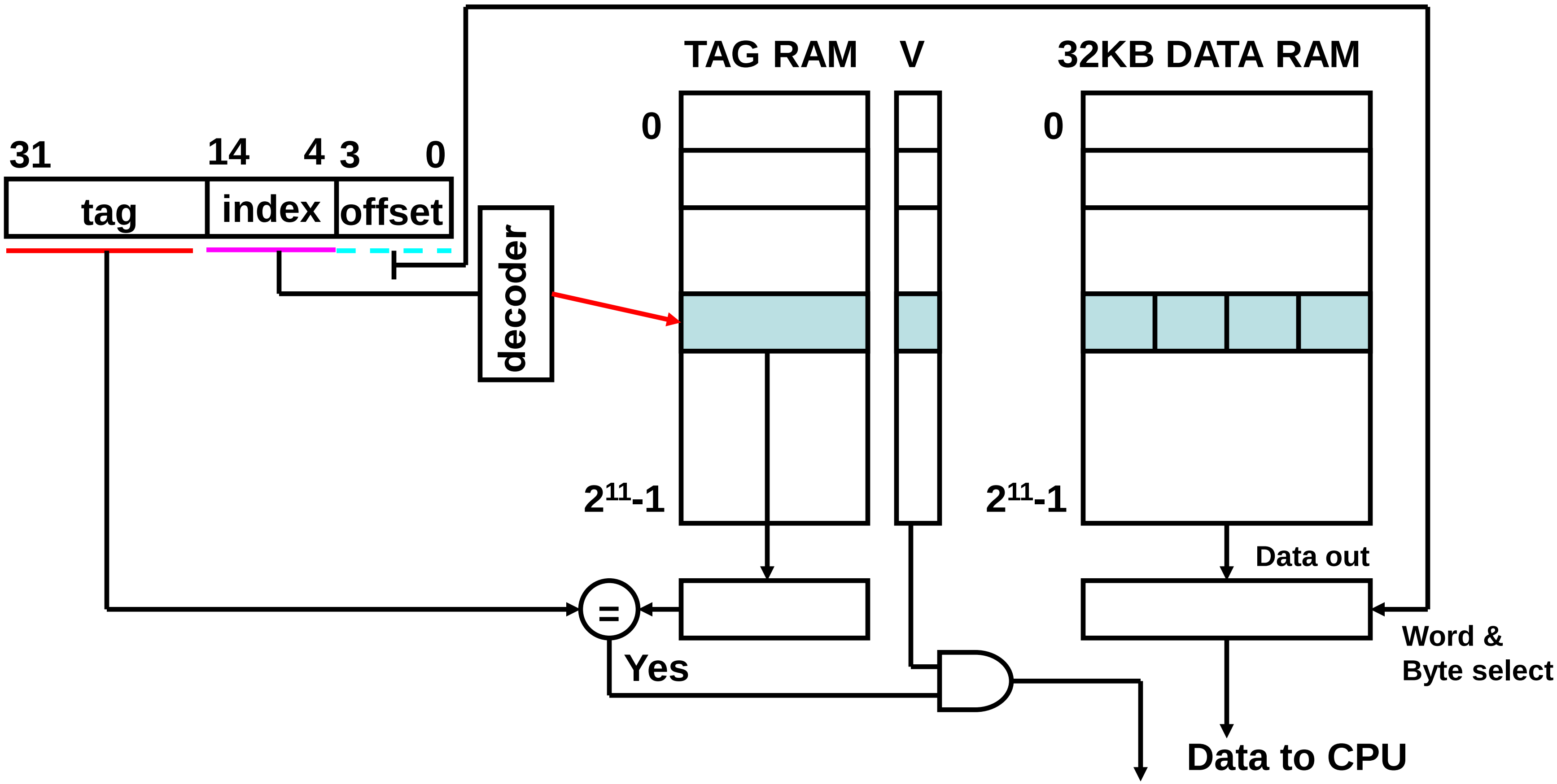
Disadvantages
1. Very expensive

Direct Mapped Cache

A memory block can be placed into only a single cache block!



Direct Mapped Cache

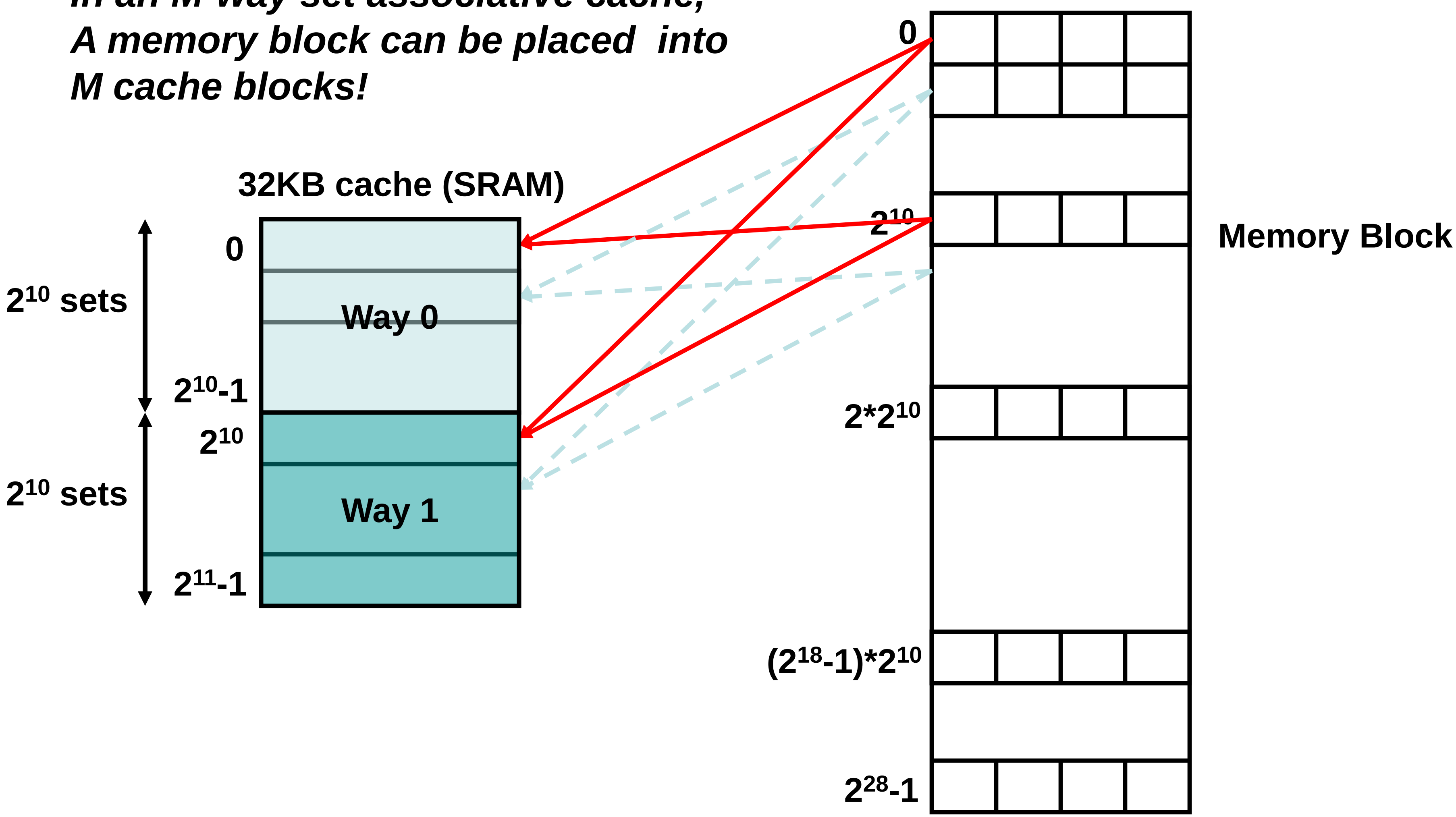


Disadvantages
1. Low hit rate

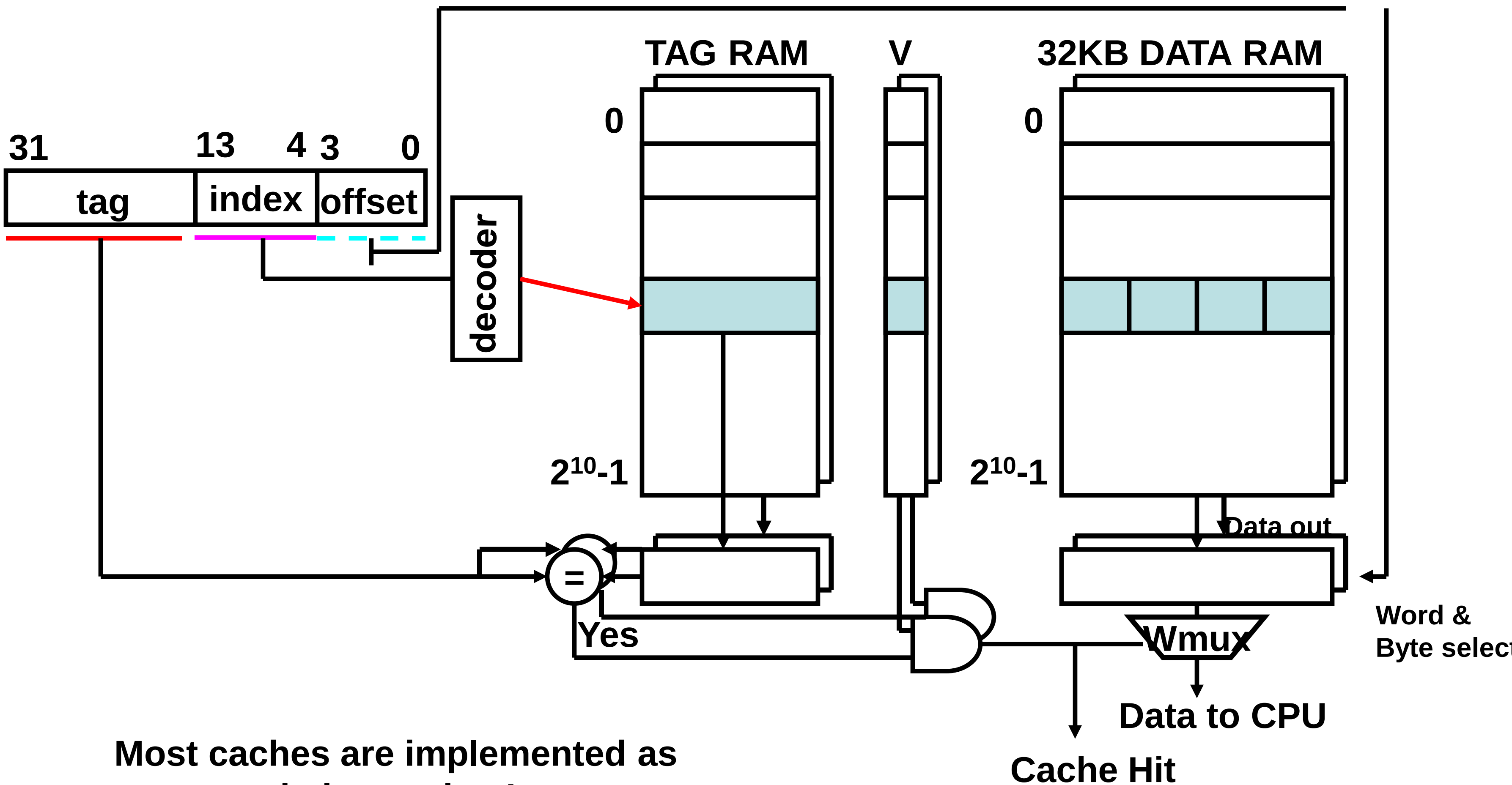
Advantages
1. Simple HW
2. Fast Implementation

Set Associative Cache

In an M-way set associative cache,
A memory block can be placed into
M cache blocks!



Set Associative Cache



Most caches are implemented as set-associative caches!

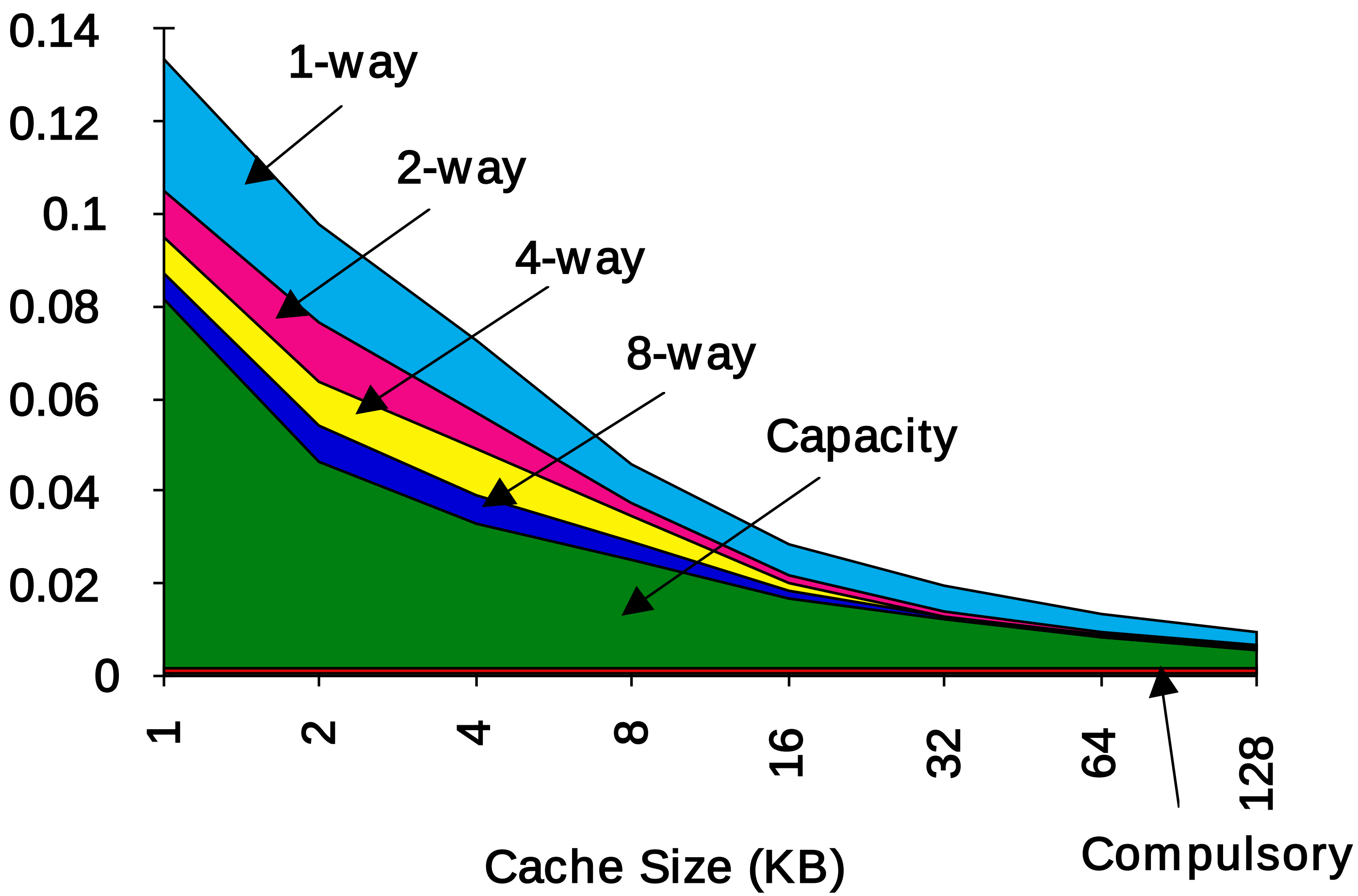
Cache Block Replacement

- **Random**
  - Just pick one and replace it
  - Pseudo-random: use simple hash algorithm using address
- **LRU (least recently used)**
  - need to keep timestamp
  - expensive due to global compare
  - Pseudo-LRU: use LFU using bit tags
- **Replacement policy critical for small caches**

3+1 Types of Cache Misses

- **Cold-start misses (or compulsory misses): the first access to a block is always not in the cache**
  - Misses even in an infinite cache
- **Capacity misses: if the memory blocks needed by a program is bigger than the cache size, then capacity misses will occur due to cache block replacement.**
  - Misses even in fully associative cache
- **Conflict misses (or collision misses): for direct-mapped or set-associative cache, too many blocks can be mapped to the same set.**
- **Invalidation misses (or sharing misses): cache blocks can be invalidated due to coherence traffic**

Miss Rates (SPEC92)



Virtual Memory

- **Virtual memory**
  - Programmer's view of memory
  - A linear array of bytes addressed by the virtual address
- **Physical memory**
  - Machine's physical memory (DRAM)
  - Also, called main memory

- **Virtual address**
  - The address of a program
- **Physical address**
  - The address of a DRAM

Virtual Memory

❑ Functions

- *Large address space*
  - Easy to program
  - Provide the illusion of infinite amount of memory
  - Program code/data can exceed the main memory size
  - Processes partially resident in memory
- *Protection*
  - Privilege level
  - Access rights: read/modify/execute permission
- *Sharing*
- *Portability*
- *Increased CPU utilization*
  - More programs can run at the same time

Virtual Memory

- **Require the following functions**
  - Memory allocation (***Placement***)
  - Memory deallocation (***Replacement***)
  - Memory mapping (***Translation***)

- **Memory management**
  - Automatic movement of data between main memory and secondary storage
    - Done by operating system with the help of processor HW
    - Main memory contains only the most frequently used portions of a process's address space
  - Illusion of infinite memory (size of secondary storage) but access time is equal to main memory
  - Use *demand paging*
    - Bring a page on demand

Paging and Segmentation

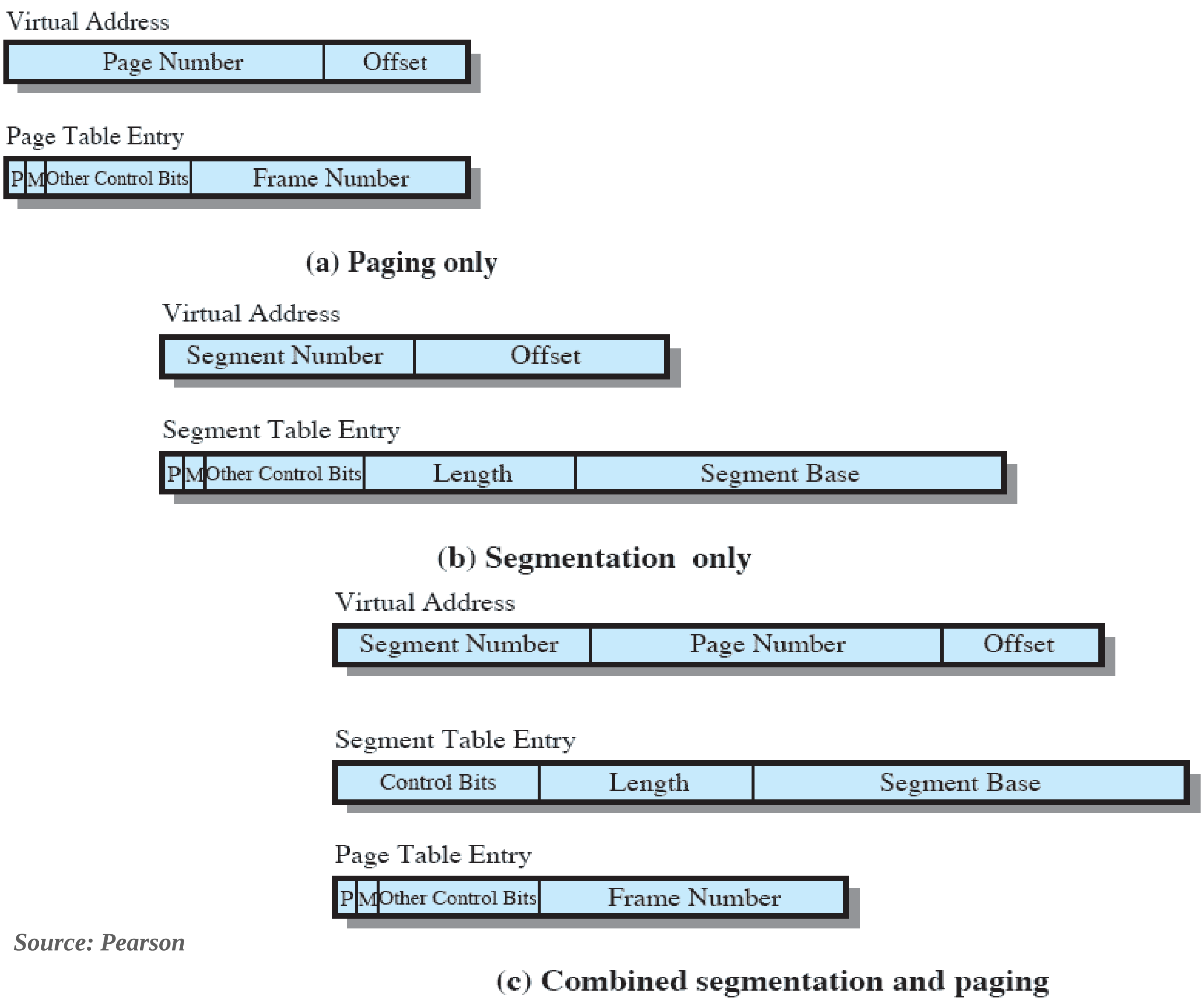
Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames	Main memory partitioned into small fixed-size chunks called frames	Main memory not partitioned	Main memory not partitioned
Program broken into pages by the compiler or memory management system	Program broken into pages by the compiler or memory management system	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)
Internal fragmentation within frames	Internal fragmentation within frames	No internal fragmentation	No internal fragmentation
No external fragmentation	No external fragmentation	External fragmentation	External fragmentation
Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a page table for each process showing which frame each page occupies	Operating system must maintain a segment table for each process showing the load address and length of each segment	Operating system must maintain a segment table for each process showing the load address and length of each segment
Operating system must maintain a free frame list	Operating system must maintain a free frame list	Operating system must maintain a list of free holes in main memory	Operating system must maintain a list of free holes in main memory
Processor uses page number, offset to calculate absolute address	Processor uses page number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address	Processor uses segment number, offset to calculate absolute address
All the pages of a process must be in main memory for process to run, unless overlays are used	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed	All the segments of a process must be in main memory for process to run, unless overlays are used	Not all segments of a process need be in main memory for the process to run. Segments may be read in as needed
	Reading a page into main memory may require writing a page out to disk		Reading a segment into main memory may require writing one or more segments out to disk

Source: Pearson

Paging

- **Divide address space into fixed size pages**
  - VA consists of (VPN, offset)
  - PA consists of (PPN, offset)
- **Map a virtual page to a physical page frame at runtime**
- **Each process has its own** *page table*
  - *The page table contains mapping between VPN and PPN*
  - VPN is used as an index into the page table
- **Page table entry (PTE) contains**
  - PPN
  - *Presence* bit – 1 if this page is in main memory
  - *Dirty* bit – 1 if this page has been modified
  - *Reference* bits – reference statistics info used for page replacement
  - *Access control* – read/write/execute permissions
  - *Privilege level* – user-level page versus system-level page
  - Disk address
- **Internal fragmentation**

Virtual Address and PTE



P= present bit
M = Modified bit

Source: Pearson

Virtual to Physical Address Translation

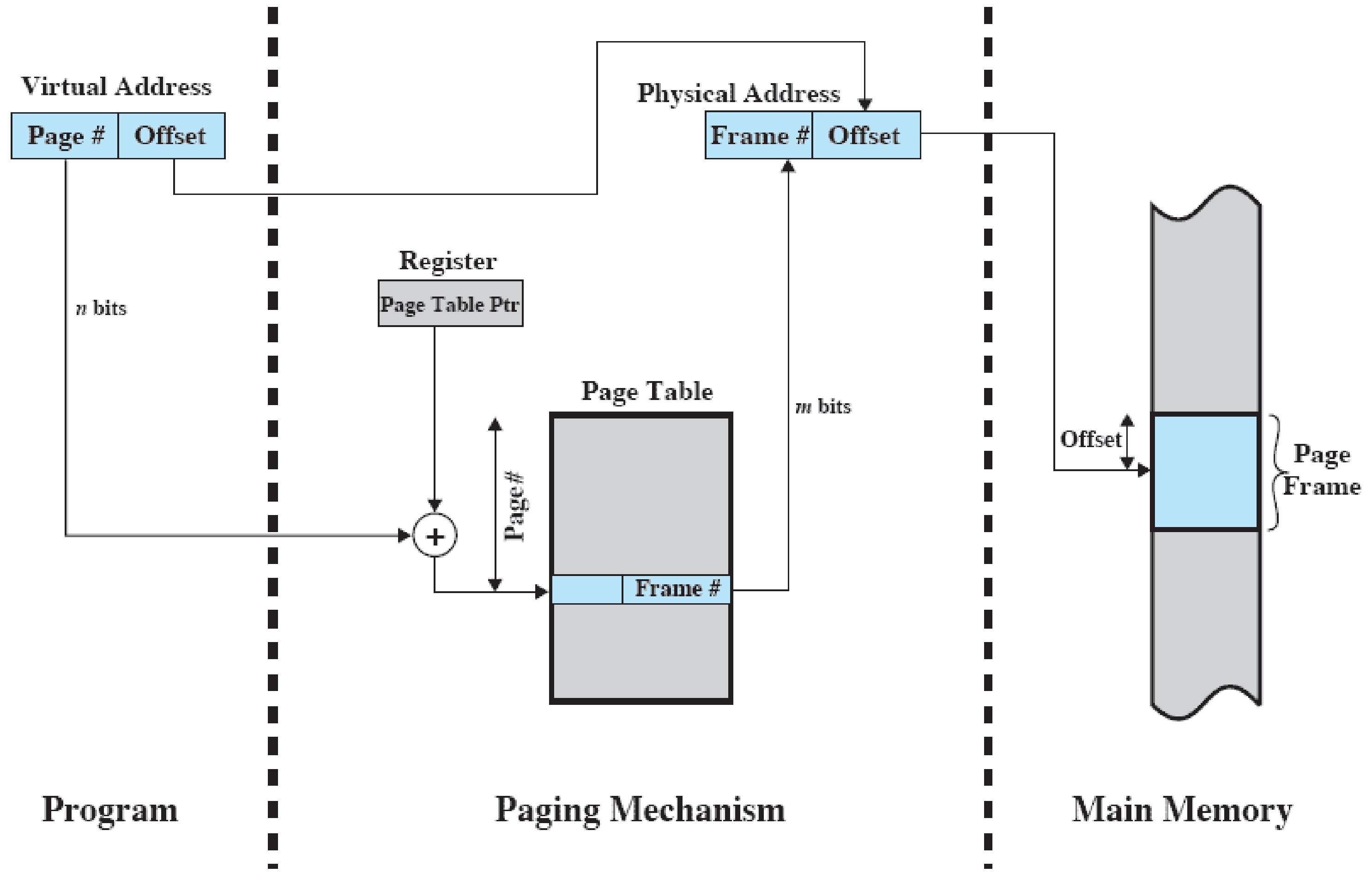


Figure 8.3 Address Translation in a Paging System

Source: Pearson

Paging

❑ Page table organization

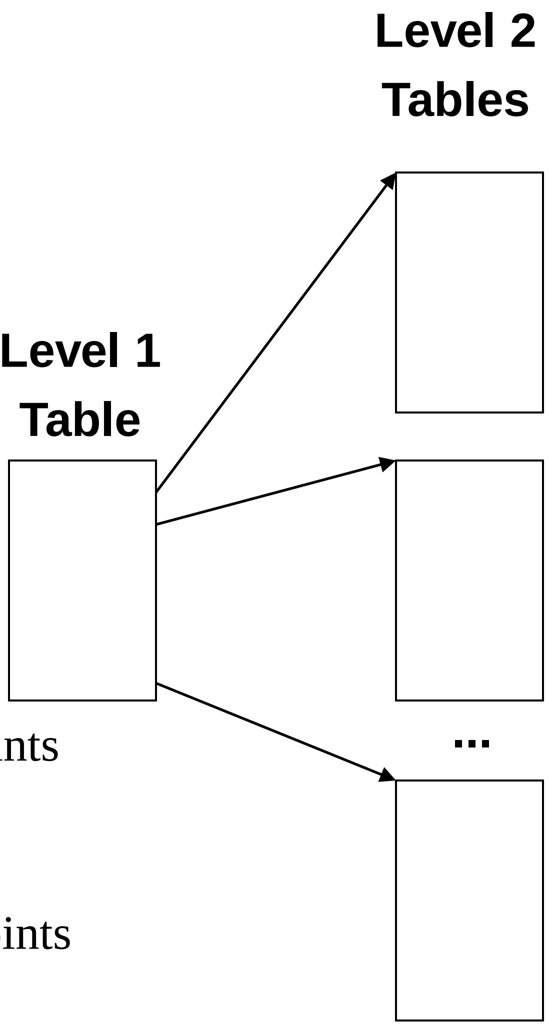
- Linear: one PTE per virtual page
- Hierarchical: tree structured page table
  - Page table itself can be paged due to its size
    - For example, 32b VA, 4KB page, 16B PTE requires 16MB page table
  - Page directory tables
    - PTE contains descriptor (i.e. index) for page table pages
  - Page tables - only leaf nodes
    - PTE contains descriptor for page
- Inverted: PTEs for only pages in main memory
- Page table entries are dynamically allocated as needed

Paging

- **Different virtual memory faults**
  - TLB miss - PTE not in TLB
  - PTE miss - PTE not in main memory
  - Page miss - page not in main memory
  - Access violation
  - Privilege violation

Multi-Level Page Tables

- **Given:**
  - 4KB (2^{12}) page size
  - 32-bit address space
  - 4-byte PTE
- **Problem:**
  - Would need a 4 MB page table!
    - 2^{20} *4 bytes
- **Common solution**
  - multi-level page tables
  - e.g., 2-level table (P6)
    - Level 1 table: 1024 entries, each of which points to a Level 2 page table.
      - This is called *page directory*
    - Level 2 table: 1024 entries, each of which points to a page



Two-Level Hierarchical Page Table

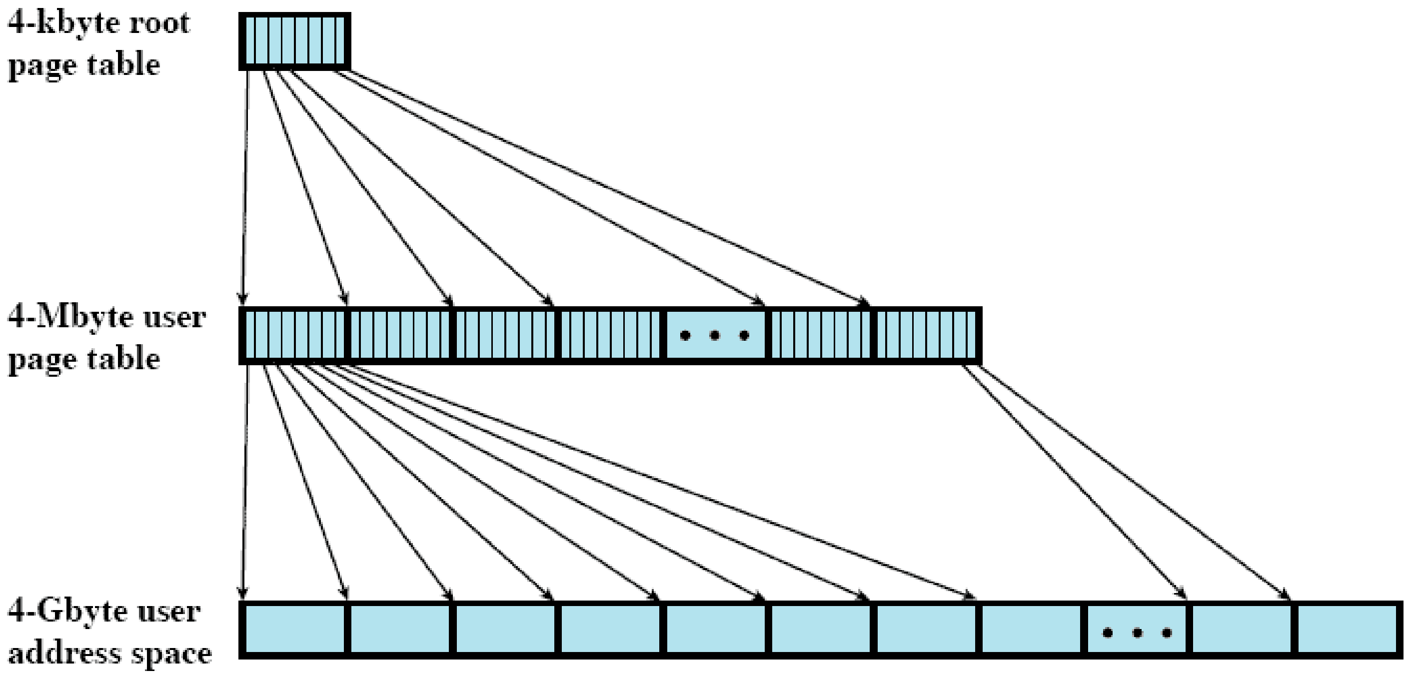


Figure 8.4 A Two-Level Hierarchical Page Table

Source: Pearson

Inverted Page Table

- **PTEs for only pages in main memory**
- **VPN is mapped into a hash value, which points to an inverted page table entry**
  - Fixed proportion of physical memory is required for the page tables regardless of the number of processes

Inverted Page Table

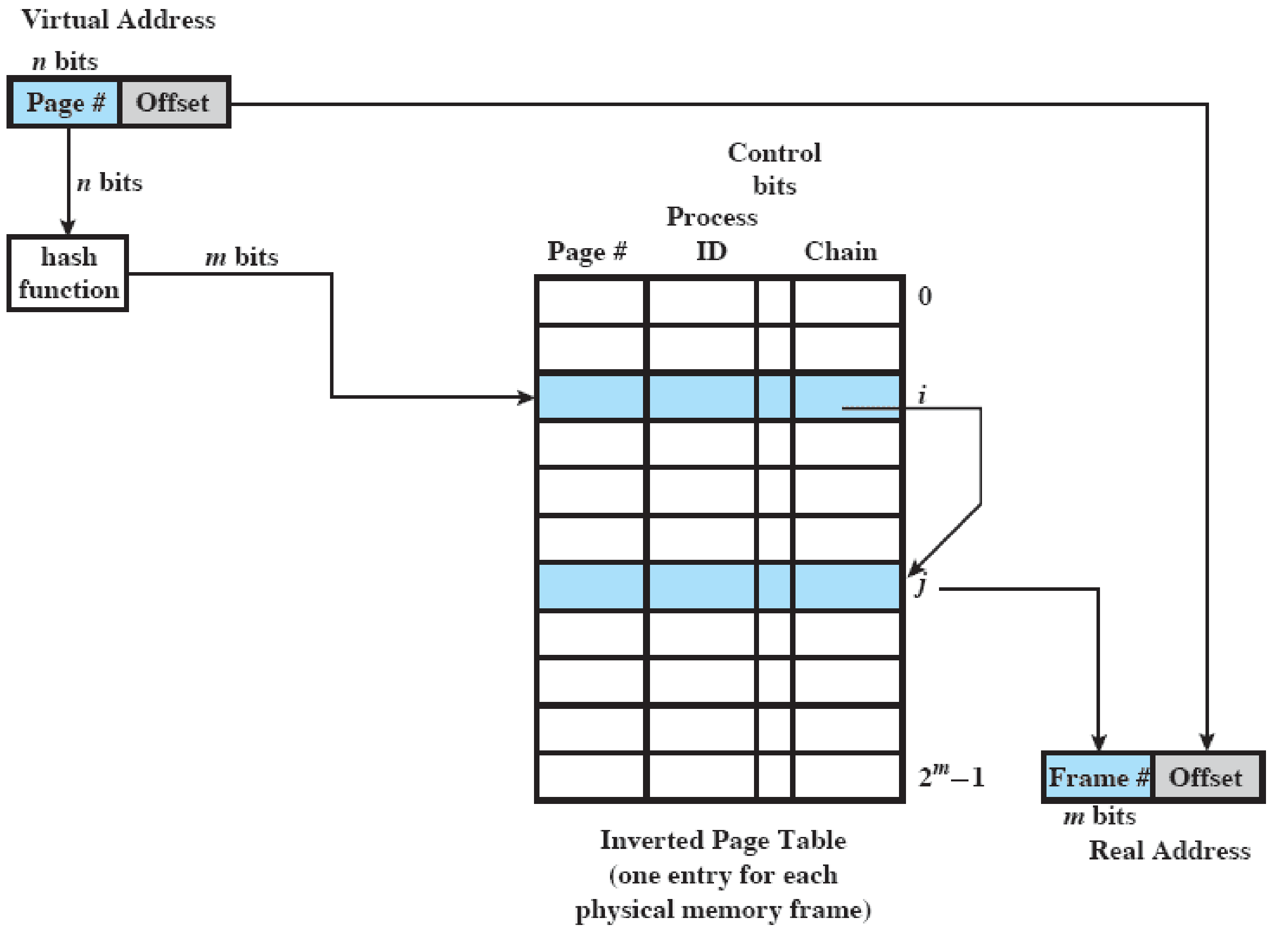


Figure 8.6 Inverted Page Table Structure

Source: Pearson

TLB

TLB (Translation Lookaside Buffer)

- Cache of page table entries (PTEs)
- On TLB hit, can do virtual to physical translation without accessing the page table
- On TLB miss, must search the page table for the missing entry

TLB configuration

- ~100 entries, usually fully associative cache
- sometimes mutil-level TLBs, TLB shootdown issue
- usually separate I-TLB and D-TLB, accessed every cycle
- Miss handling
  - On a TLB miss, exception handler (with the help of operating system) search page table for the missed TLB entry and insert it into TLB
    - Software managed TLBs - TLB insert/delete instructions
    - Flexible but slow: TLB miss handler ~ 100 instructions
  - Sometimes, by HW - HW page walker

Address Translation with TLB

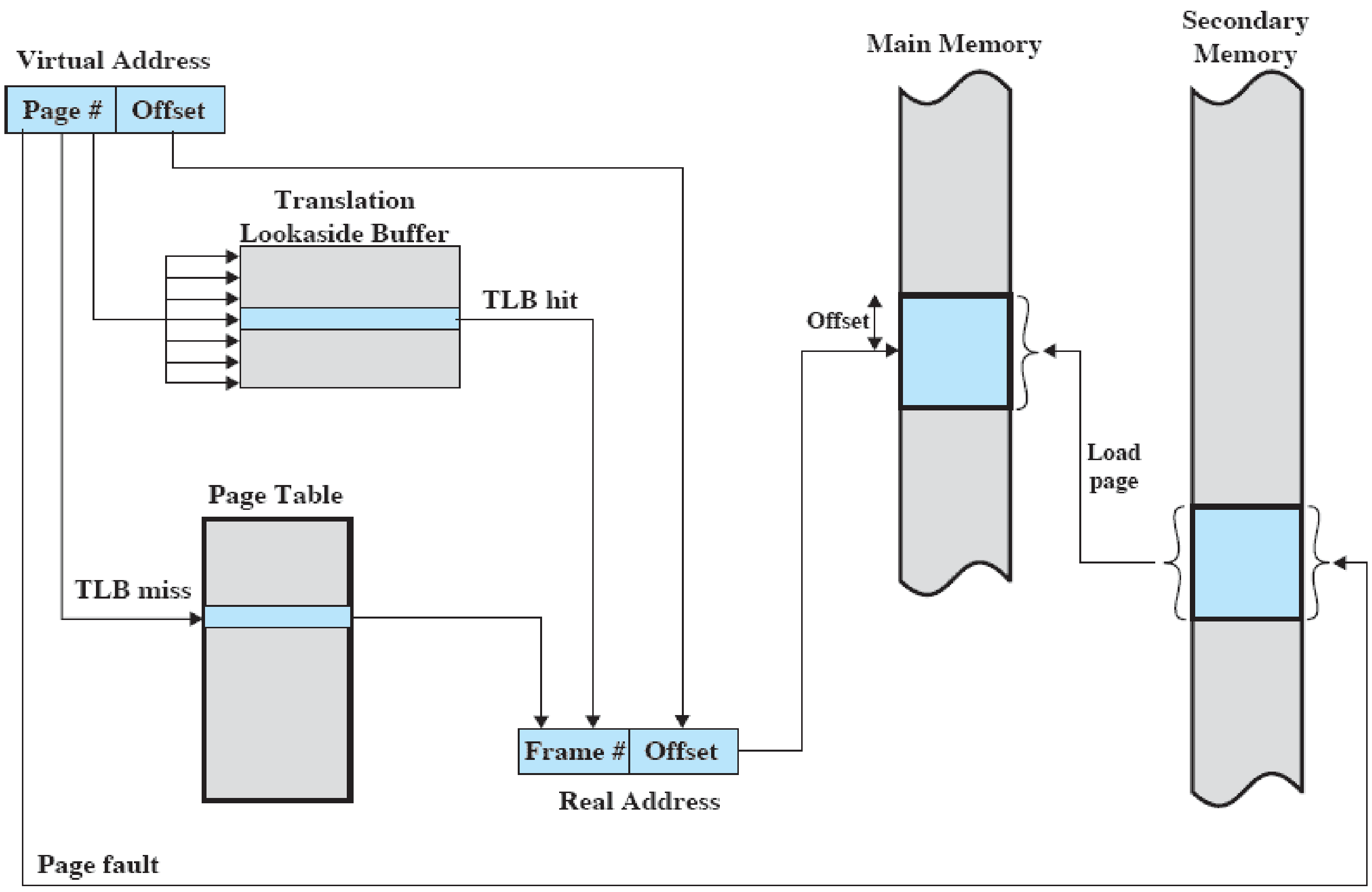
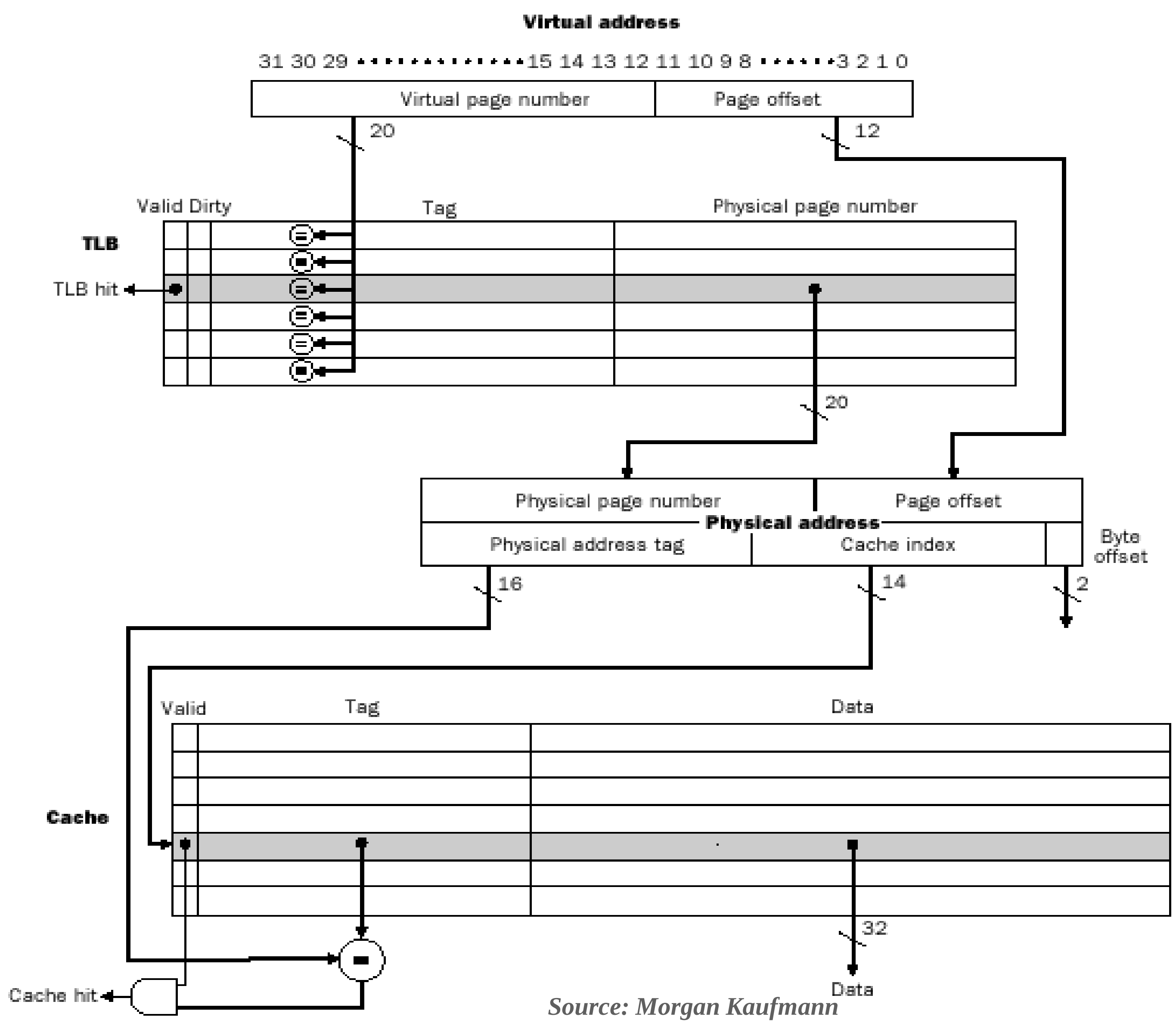


Figure 8.7 Use of a Translation Lookaside Buffer

Source: Pearson

DECStation 3100 Example



Source: Morgan Kaufmann

TLB Organization

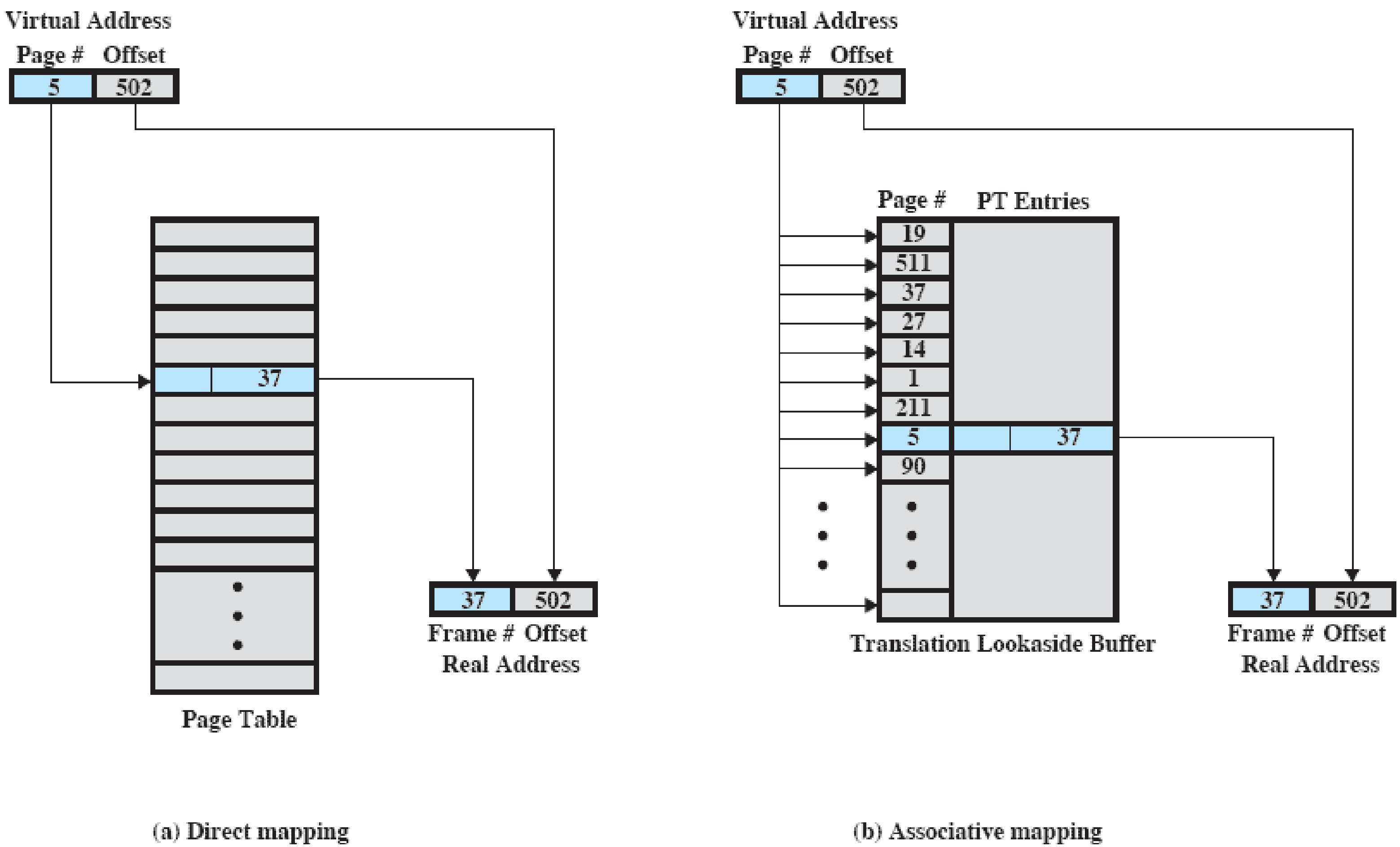


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

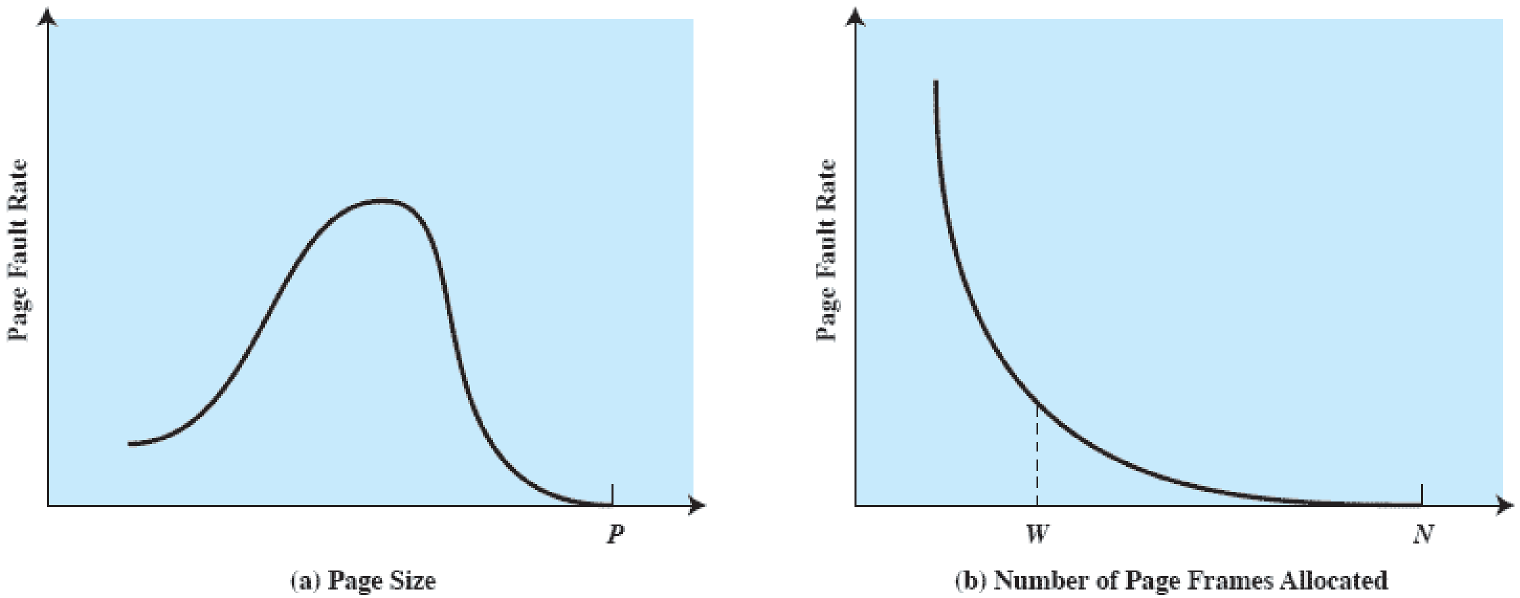
Source: Pearson

Page Size

- **The smaller the page size, the lesser the amount of internal fragmentation**
  - However, more pages are required per process
  - More pages per process means larger page tables
  - For large programs in a heavily multiprogrammed environment portion of the page tables of active processes must be in secondary storage instead of main memory
  - The physical characteristics of most secondary-memory devices favor a larger page size for more efficient block transfer

Paging Behavior of a Program

- As page size increases, each page will contain locations further away from recent references, increasing the page fault rate, but the fault rate begin to fall as the page size approaches the size of the entire process



P = size of entire process
W = working set size
N = total number of pages in process

Figure 8.11 Typical Paging Behavior of a Program

Source: Pearson

Example: Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit words
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
IBM POWER	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

Source: Pearson

Segment Organization

- **Segmentation allows a programmer to view a virtual memory as a collection of segments**
- **Advantages**
  - Simplify the handling of growing data structures
  - Allow program modules to be altered and recompiled independently
  - Facilitate sharing among processes
- **Segment table entry contains the starting address of the corresponding segment in main memory and the length of the segment**
  - A *presence* bit is needed to determine if the segment is already in main memory
  - A *dirty* bit is needed to determine if the segment has been modified since it was loaded in main memory

Address Translation

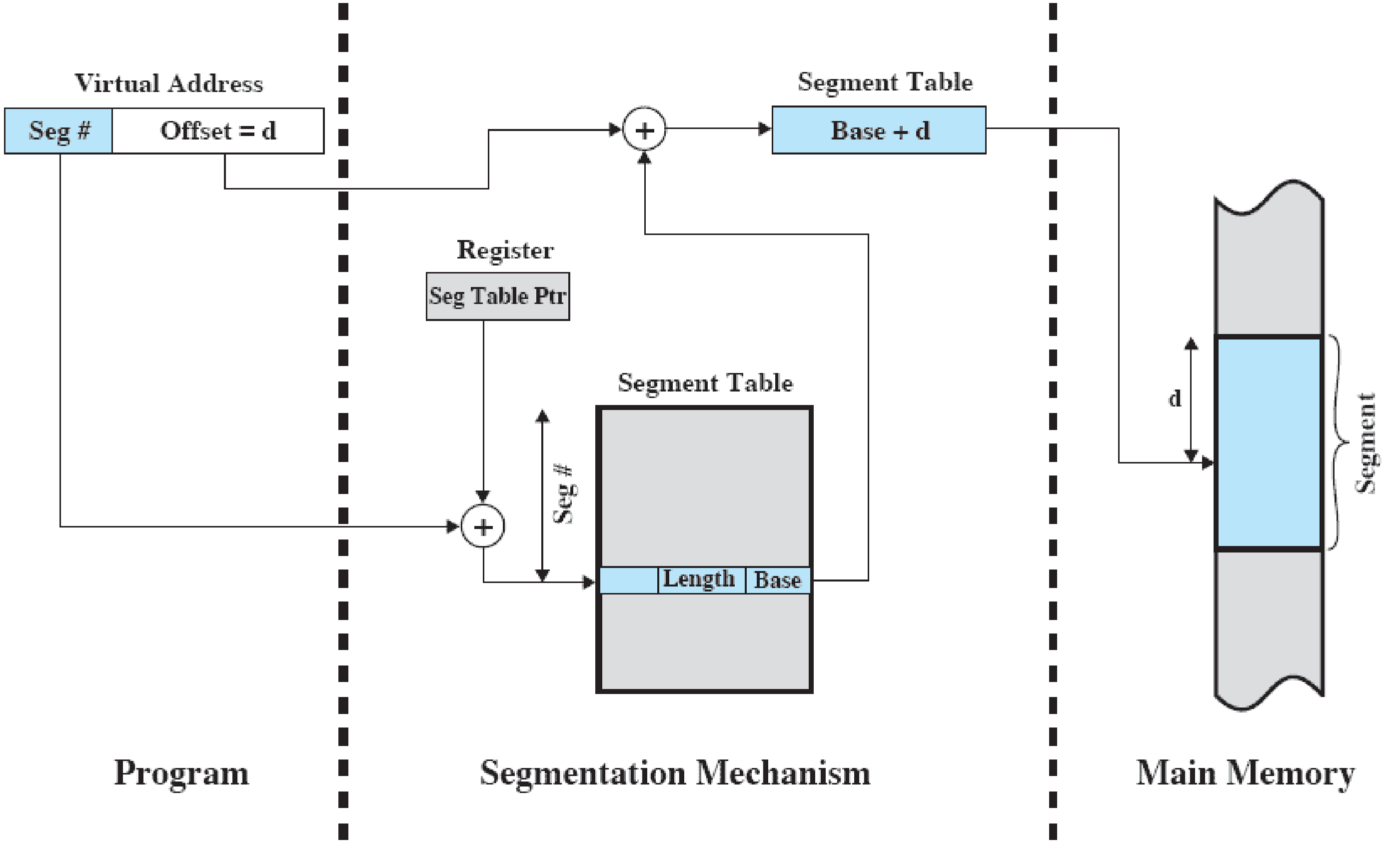
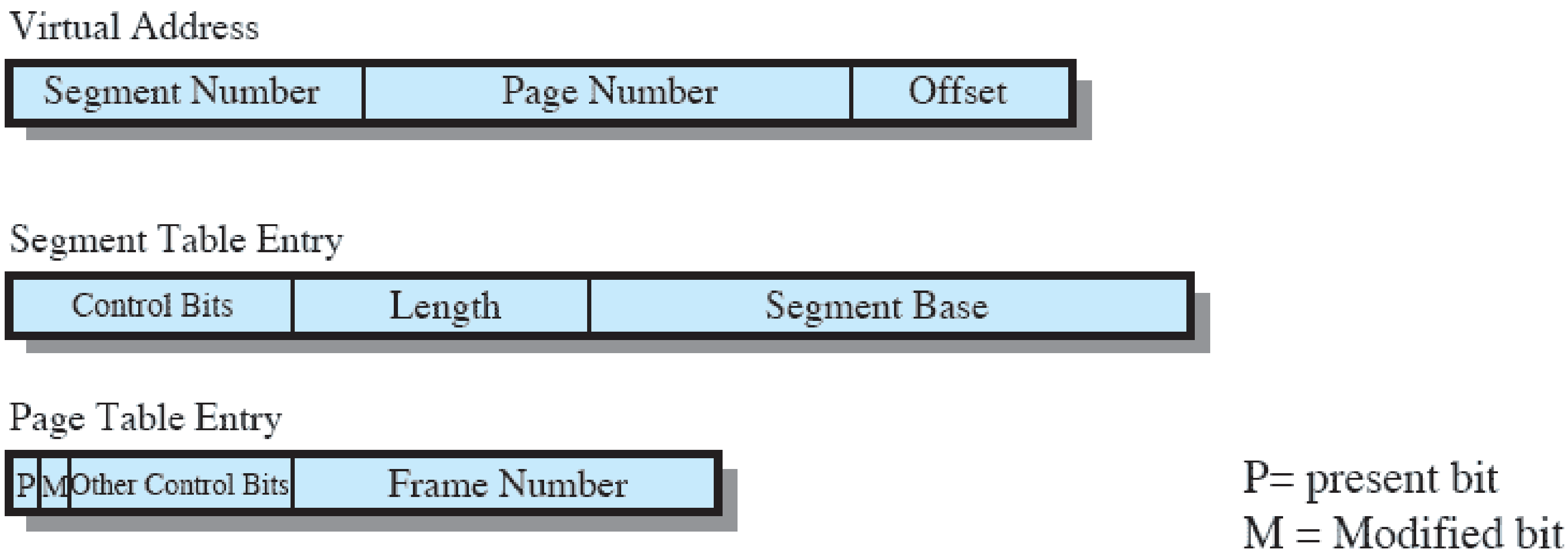


Figure 8.12 Address Translation in a Segmentation System

Source: Pearson

Paged Segmentation

- Virtual address space is broken up into a number of segments. Each segment is broken up into a number of fixed-sized pages.



(c) Combined segmentation and paging

Source: Pearson

Address Translation

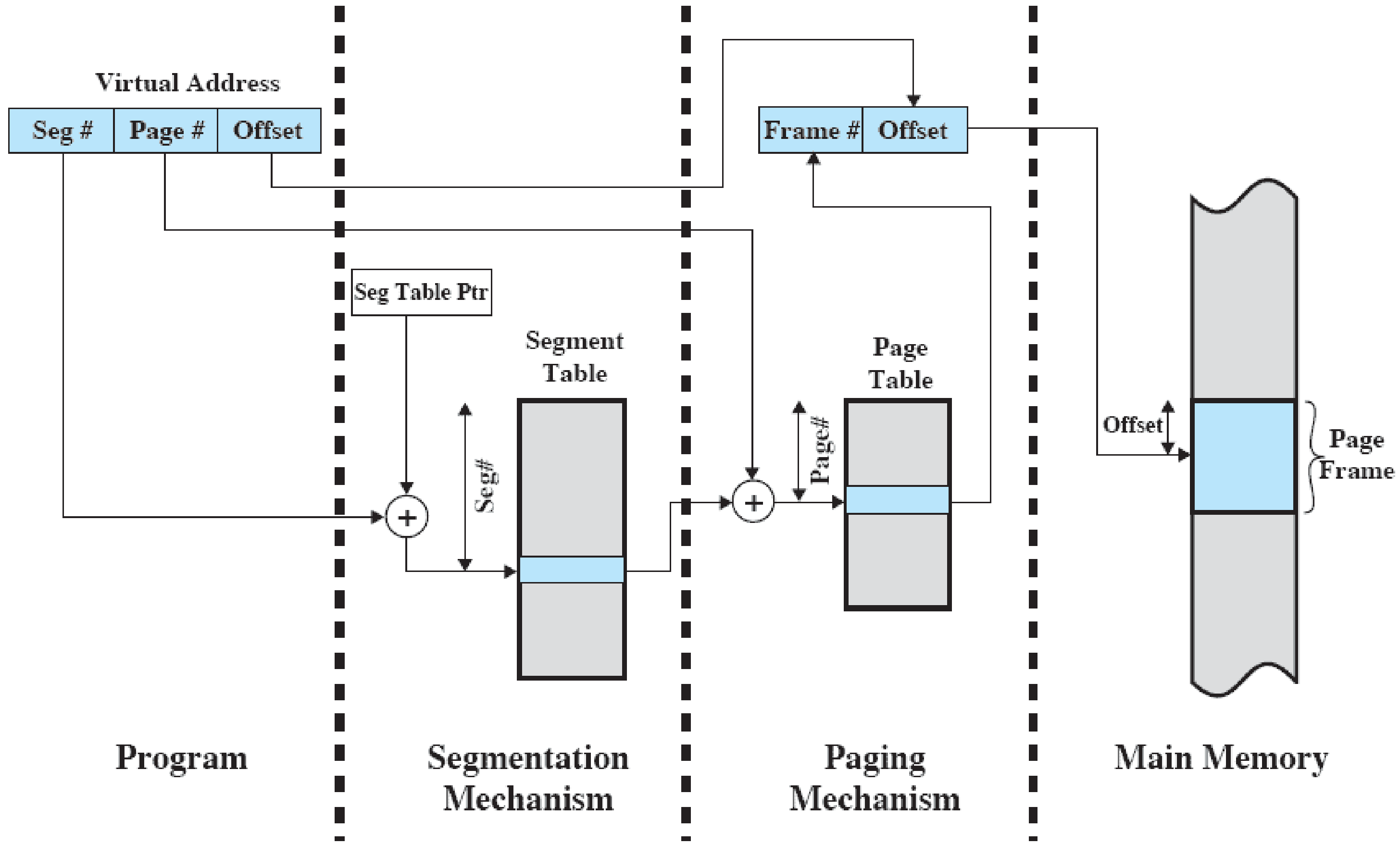


Figure 8.13 Address Translation in a Segmentation/Paging System

Source: Pearson

Virtual Memory Policies

- **Key issues: Performance**
  - Minimize page faults

Fetch Policy Demand paging Prepaging Placement Policy Replacement Policy Basic Algorithms Optimal Least recently used (LRU) First-in-first-out (FIFO) Clock Page Buffering	Resident Set Management Resident set size Fixed Variable Replacement Scope Global Local Cleaning Policy Demand Precleaning Load Control Degree of multiprogramming

Source: Pearson

Fetch Policy

- **Demand Paging**
  - Bring a page into main memory only on a page miss
  - Generate many page faults when process is first started
  - Principle of locality suggests that as more and more pages are brought in, most future references will be to pages that have recently been brought in, and page faults should drop to a very low level
- **Prepaging**
  - Pages other than the one demanded by a page fault are brought in
  - If pages of a process are stored contiguously in secondary memory it is more efficient to bring in a number of pages at one time
  - Ineffective if extra pages are not referenced

Frame Locking

- **When a frame is locked, the page currently stored in that frame should not be replaced**
  - OS kernel and key control structures are locked
  - I/O buffers and time-critical areas may be locked
  - Locking is achieved by associating a lock bit with each frame

Replacement Algorithms

- **Optimal**
  - Select the page for which the time to the next reference is the longest
- **LRU**
  - Select the page that has not been referenced for the longest time
- **FIFO**
  - Page that has been in memory the longest is replaced
- **Clock**
  - Associate a *use* bit with each frame
  - When a page is first loaded or referenced, the use bit is set to 1
  - Any frame with a use bit of 1 is passed over by the algorithm
  - Page frames visualized as laid out in a circle

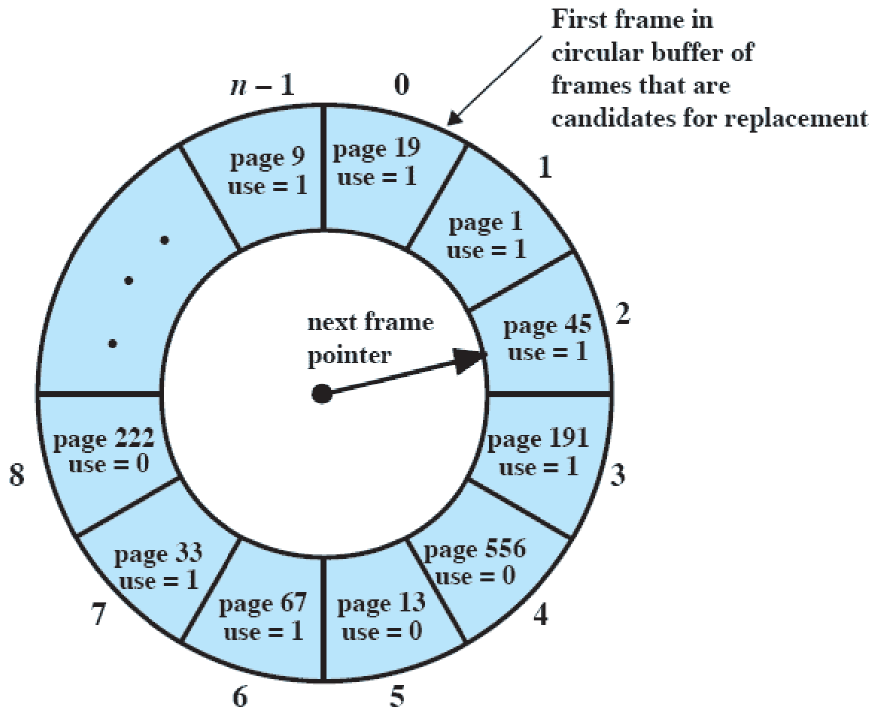
Combined Examples

F= page fault occurring after the frame allocation is initially filled

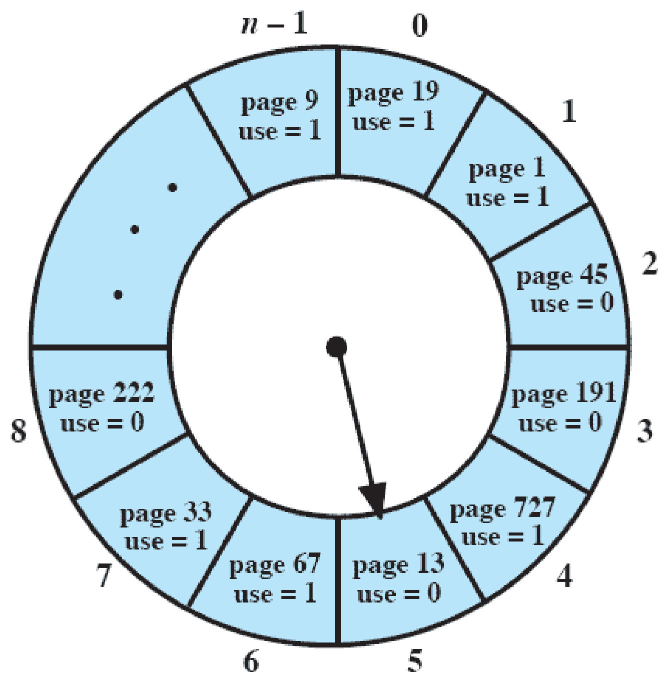
Figure 8.15 **Behavior of Four Page Replacement Algorithms**

Source: Pearson

Clock Policy



(a) State of buffer just prior to a page replacement



(b) State of buffer just after the next page replacement

Source: Pearson

Figure 8.16 Example of Clock Policy Operation

Comparison of Algorithms

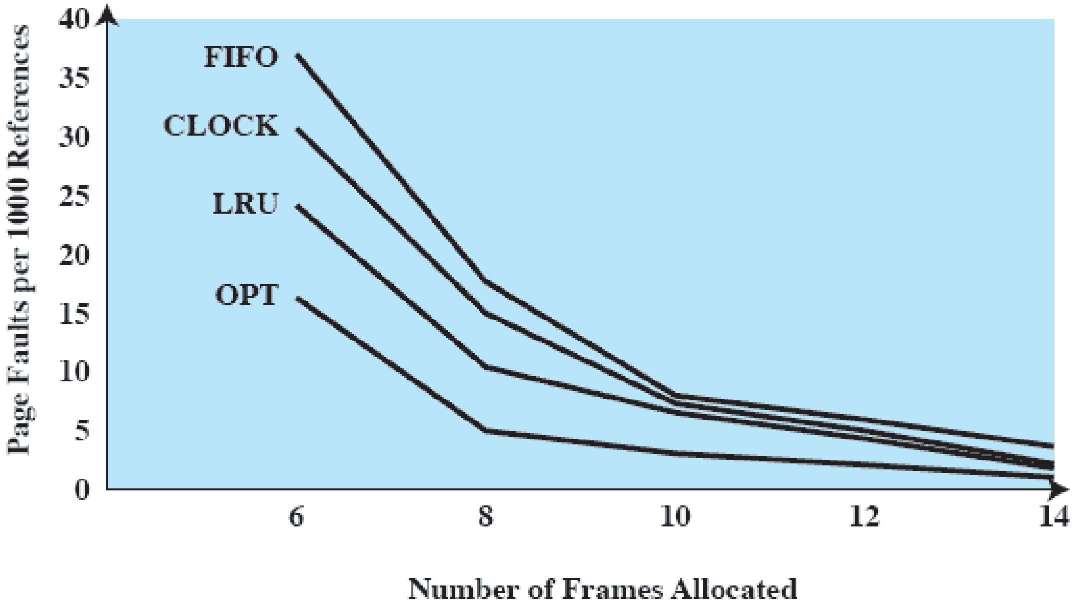


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Source: Pearson

Page Buffering

- **A replaced page is not lost, but rather assigned to one of two lists**
  - *Free page list* is a list of page frames available for reading in pages
    - When a page is to be read in, the page frame at the head of the list is used, destroying the page that was there
    - When a unmodified page is to be replaced, it remains in memory and its page frame is added to the tail of the free page list
  - *Modified page list* is a list of page frames that have been modified
    - When a modified page is to be written out and replaced, the page frame is added to the tail of the modified page list
  - Note that when a page is replaced, the page is not physically moved. Instead, the PTE for this page is removed and placed in either the free or modified page list
- **Used in VAX VMS**

Working Set Management

- **The OS must decide how many pages to bring in**
  - The smaller the amount of memory allocated to each process, the more processes can reside in memory
  - Small number of pages loaded increases page faults
  - Beyond a certain size, further allocations of pages will not effect the page fault rate
- **Fixed allocation**
  - Allocate a fixed number of frames to a process
  - On a page fault, one of the pages of that process must be replaced
- **Variable allocation**
  - Allow the number of page frames allocated to a process to be varied over the lifetime of the process

Replacement Scope

- **The scope of a replacement strategy can be *global* or *local***
- **Local scope**
  - Choose only among the resident pages of the process generating the fault
- **Global scope**
  - Consider all unlocked pages in main memory

	Local Replacement	Global Replacement
Fixed Allocation	•Number of frames allocated to a process is fixed. •Page to be replaced is chosen from among the frames allocated to that process.	•Not possible.
Variable Allocation	•The number of frames allocated to a process may be changed from time to time to maintain the working set of the process. •Page to be replaced is chosen from among the frames allocated to that process.	•Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Source: Pearson

Fixed Allocation, Local Scope

- Necessary to decide ahead of time the amount of allocation to a process
- If allocation is too small, there will be a high page fault rate
- If allocation is too large, there will be too few processes in main memory
  - Increase processor idle time
  - Increase time spent in swapping

Variable Allocation, Global Scope

- Easiest to implement
  - Adopted in many operating systems
- OS maintains a list of free frames
- Free frame is added to working set of a process when a page fault occurs
- If no frames are available, the OS must choose a page currently in memory, except the locked frames

Variable Allocation, Local Scope

- **When a new process is loaded into main memory, allocate to it a certain number of page frames as its working set**
- **When a page fault occurs, select the page to replace from among the resident set of the process that suffers the fault**
- **Reevaluate the allocation provided to the process and increase or decrease it to improve overall performance**
  - Decision to increase or decrease a working set size is based on the assessment of future demands

Working Set of a Process

Sequence of Page References	Window Size, Δ 2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

Source: Pearson

Page Fault Frequency (PFF)

- **Requires a *use* bit to be associated with each page in memory**
  - Bit is set to 1 when that page is accessed
  - When a page fault occurs, the OS notes the virtual time since the last page fault for that process
  - If the amount of time since the last page fault is less than a threshold, then a page is added to the working set of the process
  - The strategy can be refined by using 2 thresholds: An upper threshold is used to trigger a growth in the working set size while a lower threshold is used to trigger a shrink in the working set size.
- **Does not perform well during the transient periods when there is a shift to a new locality**

Cleaning Policy

- **Concerned with determining when a modified page should be written out to secondary memory**
- **Demand cleaning**
  - A page is written out to secondary memory only when it has been selected for replacement
- **Precleaning**
  - Write modified pages before they are replaced
  - The pages may be modified again before they are replaced

Multiprogramming

- Determines the number of processes that will be resident in main memory
  - *Multiprogramming* level
- Too few processes lead to swapping
- Too many processes, lead to insufficient working set size, resulting in thrashing (frequent faults)

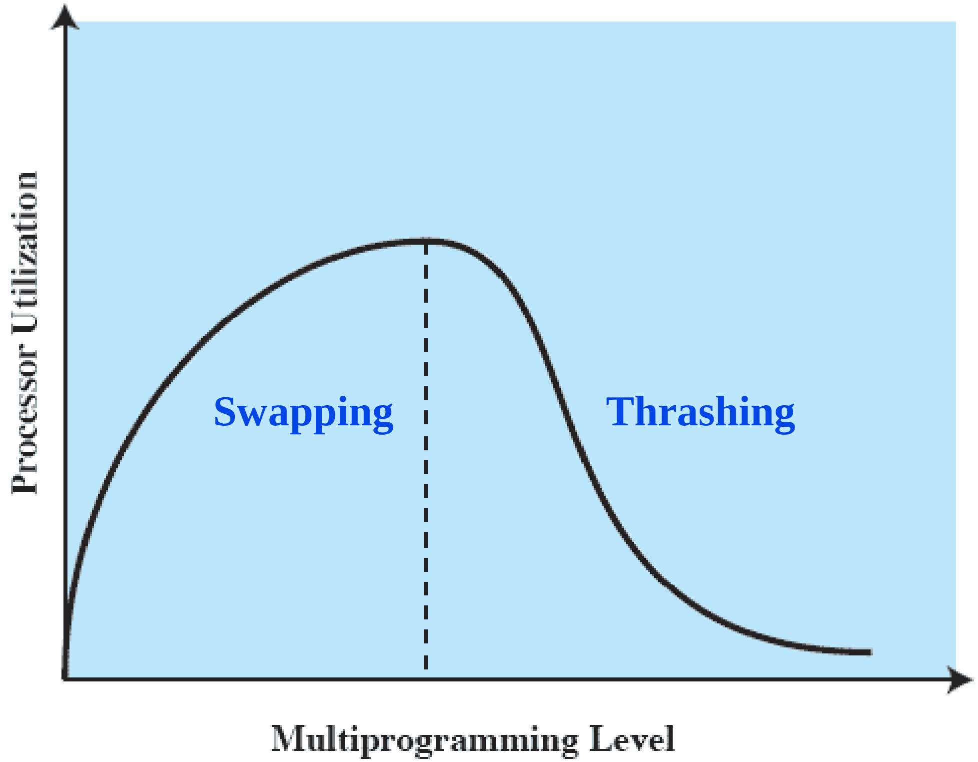


Figure 8.21 Multiprogramming Effects

Source: Pearson

Process Suspension

- **If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be swapped out**
- **Six possibilities**
  - Lowest-priority process
  - Faulting process
  - Last process activated
  - Process with the smallest working set
  - Largest process
  - Process with the largest remaining execution window

Unix

- **Intended to be machine independent**
  - Early Unix: variable partitioning with no virtual memory scheme
  - Current implementations of UNIX and Solaris make use of <u>two separate memory management schemes</u>
    - *Paging system* for user processes and disk I/O
    - *Kernel memory allocator* to manage memory allocation for the kernel

UNIX SVR4 Memory Management Format

Page frame number	Age	Copy on write	Mod- ify	Refe- rence	Valid	Pro- tect

(a) Page table entry

One entry for each page

Swap device number	Device block number	Type of storage

(b) Disk block descriptor

One entry for each page

Page state	Reference count	Logical device	Block number	Pfdata pointer

(c) Page frame data table entry

Indexed by frame number and used by the replacement algorithm

Reference count	Page/storage unit number

One entry for each page (one table for each swap devic

(d) Swap-use table entry

Figure 8.22 UNIX SVR4 Memory Management Formats

Source: Pearson

UNIX SVR4 Memory Management Parameters

Page Table Entry

Page frame number
Refers to frame in real memory.

Age
Indicates how long the page has been in memory without being referenced. The length and contents of this field are processor dependent.

Copy on write
Set when more than one process shares a page. If one of the processes writes into the page, a separate copy of the page must first be made for all other processes that share the page. This feature allows the copy operation to be deferred until necessary and avoided in cases where it turns out not to be necessary.

Modify
Indicates page has been modified.

Reference
Indicates page has been referenced. This bit is set to 0 when the page is first loaded and may be periodically reset by the page replacement algorithm.

Valid
Indicates page is in main memory.

Protect
Indicates whether write operation is allowed.

Disk Block Descriptor

Swap device number
Logical device number of the secondary device that holds the corresponding page. This allows more than one device to be used for swapping.

Device block number
Block location of page on swap device.

Type of storage
Storage may be swap unit or executable file. In the latter case, there is an indication as to whether the virtual memory to be allocated should be cleared first.

Source: Pearson

UNIX SVR4 Memory Management Parameters

Page Frame Data Table Entry

Page state
Indicates whether this frame is available or has an associated page. In the latter case, the status of the page is specified: on swap device, in executable file, or DMA in progress.

Reference count
Number of processes that reference the page.

Logical device
Logical device that contains a copy of the page.

Block number
Block location of the page copy on the logical device.

Pfdata pointer
Pointer to other pfdata table entries on a list of free pages and on a hash queue of pages.

Swap-Use Table Entry

Reference count
Number of page table entries that point to a page on the swap device.

Page/storage unit number
Page identifier on storage unit.

Source: Pearson

Homework 7

- Exercise 8.1
- Exercise 8.2
- Exercise 8.6
- Exercise 8.9
- Exercise 8.15
- Exercise 8.16