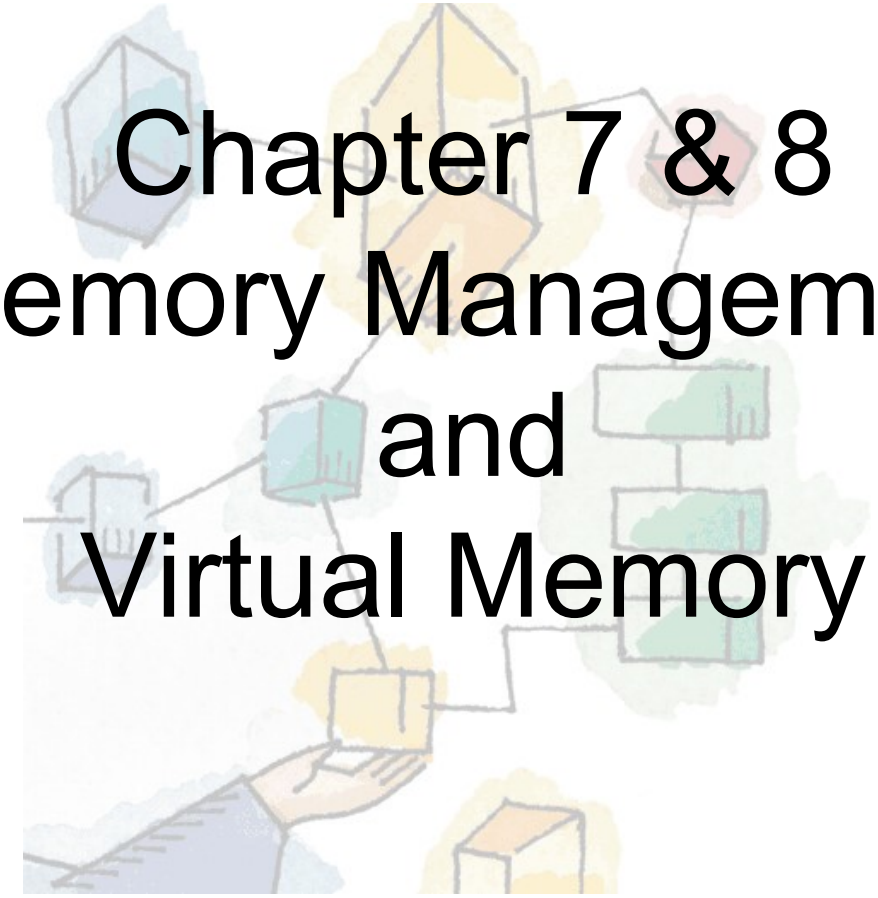
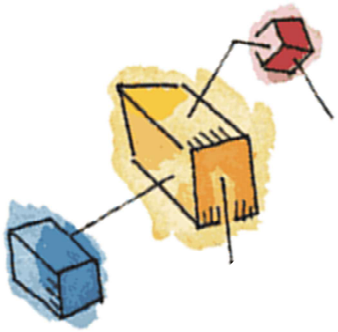
Operating Systems:
Internals and Design Principles
William Stallings



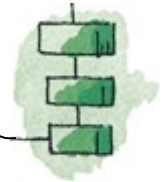
Chapter 7 & 8
Memory Management
and
Virtual Memory

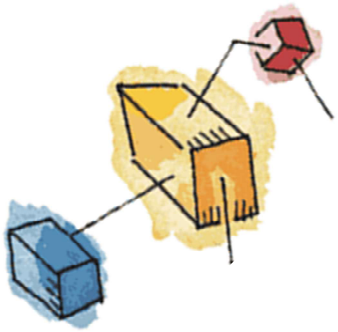


Roadmap

→ Basic requirements of Memory Management

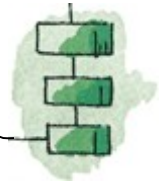
- Basic blocks of memory management
  - Paging
  - Segmentation
- Virtual Memory (VM) Basics
- Hardware and Control Structures of VM
  - Paging
  - Segmentation
  - Combined Paging and Segmentation
- VM Management

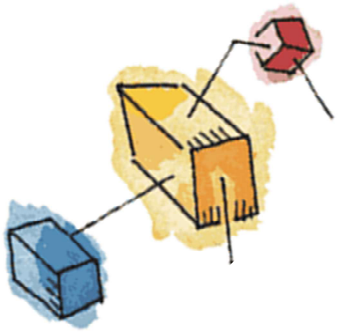




Terminology

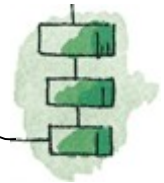
- Real memory
  - Main memory, the actual RAM, where a process executes.
  - ***Real address*** (***physical address***) is the address of a storage location in main memory.
- Virtual memory (memory on disk)
  - A storage allocation scheme in which *secondary memory* can be addressed as though it were part of main memory.
  - ***Virtual address*** (***logical address***) is the address assigned to a location in virtual memory.
- Address space
  - The range of memory addresses available to a process.

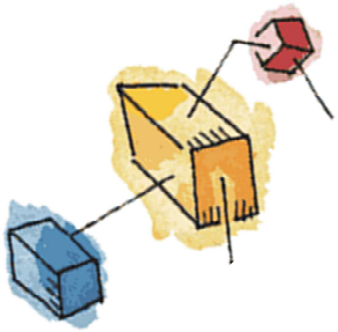




Memory Management

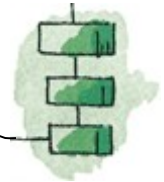
- The principal operation of memory management is to bring processes into main memory for execution by the processor.
- Memory is cheap today, and getting cheaper.
  - But applications are demanding more and more memory, there is never enough!
- In a multiprogramming system, if only a few processes are in memory, then for much of the time, all of the processes will be waiting for I/O and the processor will be idle.
- Thus memory needs to be allocated to ensure a reasonable supply of ready processes to consume available processor time.

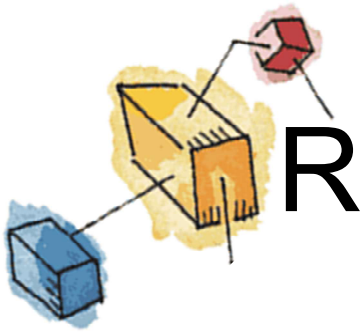




Memory Management Requirements

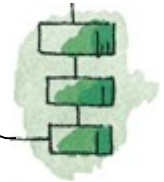
- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

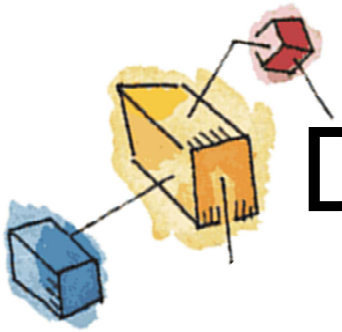




Requirements: Relocation

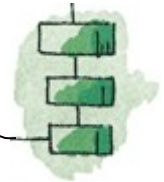
- Programmers do not (need to) know where the program will be placed in memory when it is executed.
- In fact, a process may occupy different actual memory locations during execution.
  - In order to maximize processor utilization, active processes need to be swapped in and out of main memory.
  - When a process is swapped back, may need to ***relocate*** the process to a different area of memory.

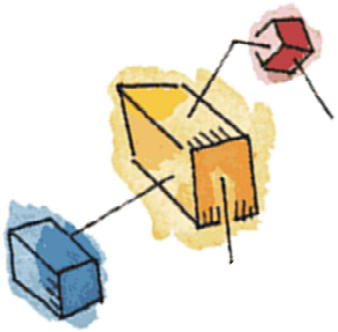




Different Types of Addresses

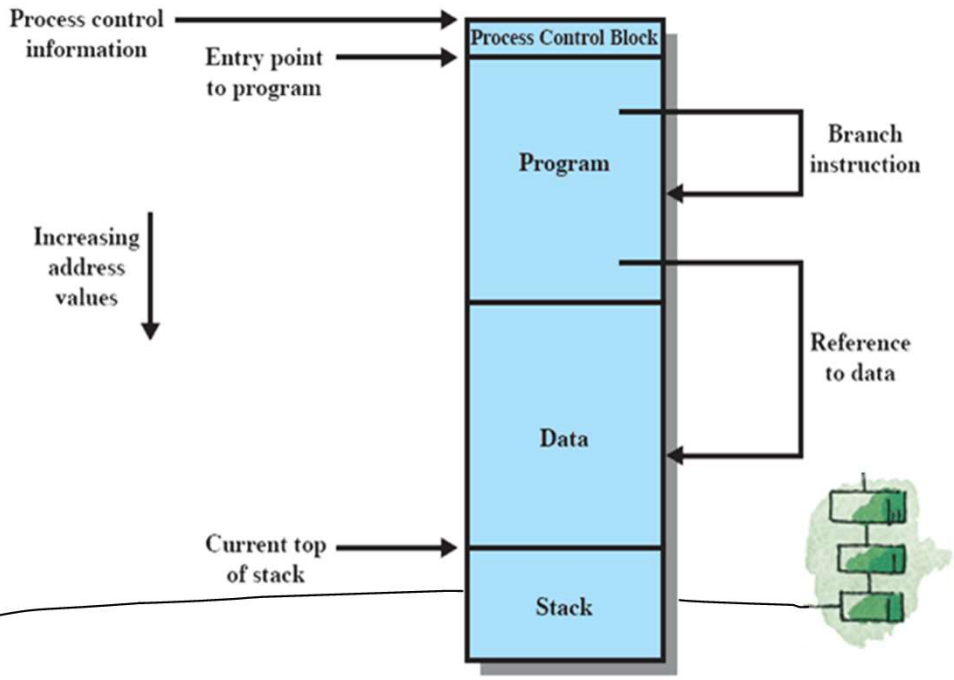
- *Logical address*
  - Reference to a memory location independent of the current assignment of data to memory.
- *Relative address*
  - An example of logical address.
  - Address is expressed as a location relative to some known point such as the origin of the program.
- *Physical* or *Absolute address*
  - Actual location in main memory.

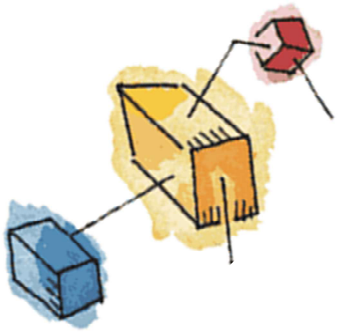




Relocation

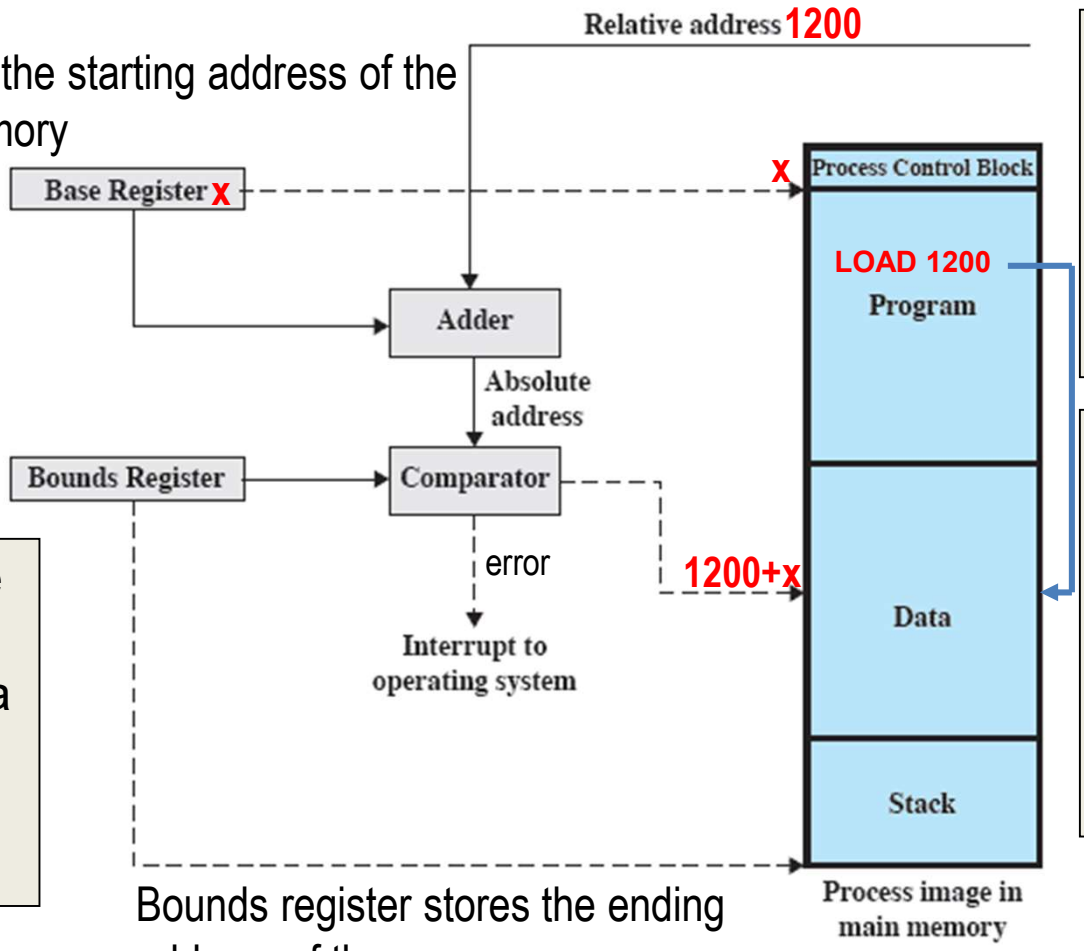
- The processor and OS must be able to translate the memory references (logical addresses) found in the code of the program into physical addresses before memory access can be achieved.
  - Reference to program code
  - Reference to data





Relocation Hardware Support

Base register stores the starting address of the process in main memory

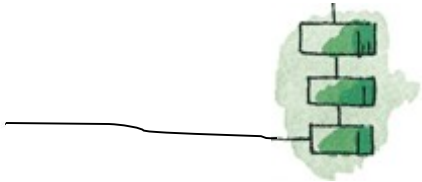


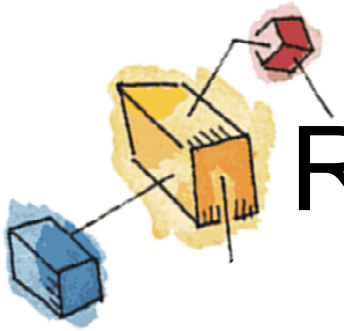
1. Add the value in the base register to the relative address to produce an absolute address.

2. Compare the address to the value in the bounds register. If within bounds, execution may proceed.

0. Initialize base and bounds registers when a process is assigned to the Running state.

Bounds register stores the ending address of the process





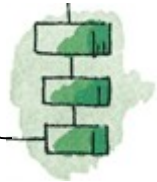
Requirements: Protection

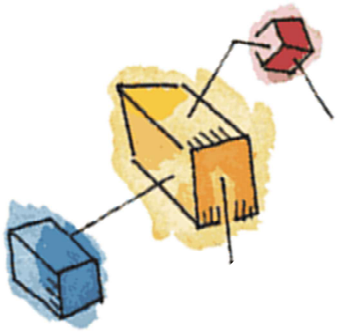
Processes should not be able to reference memory locations in another process without permission.

Impossible to check absolute addresses at compile time because the location of a program in main memory is unpredictable.

Memory references generated by a process must be checked at run time.

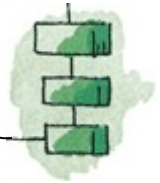
Mechanisms that support relocation also support protection.

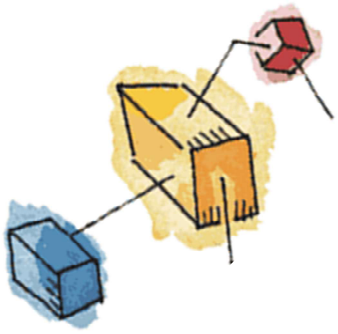




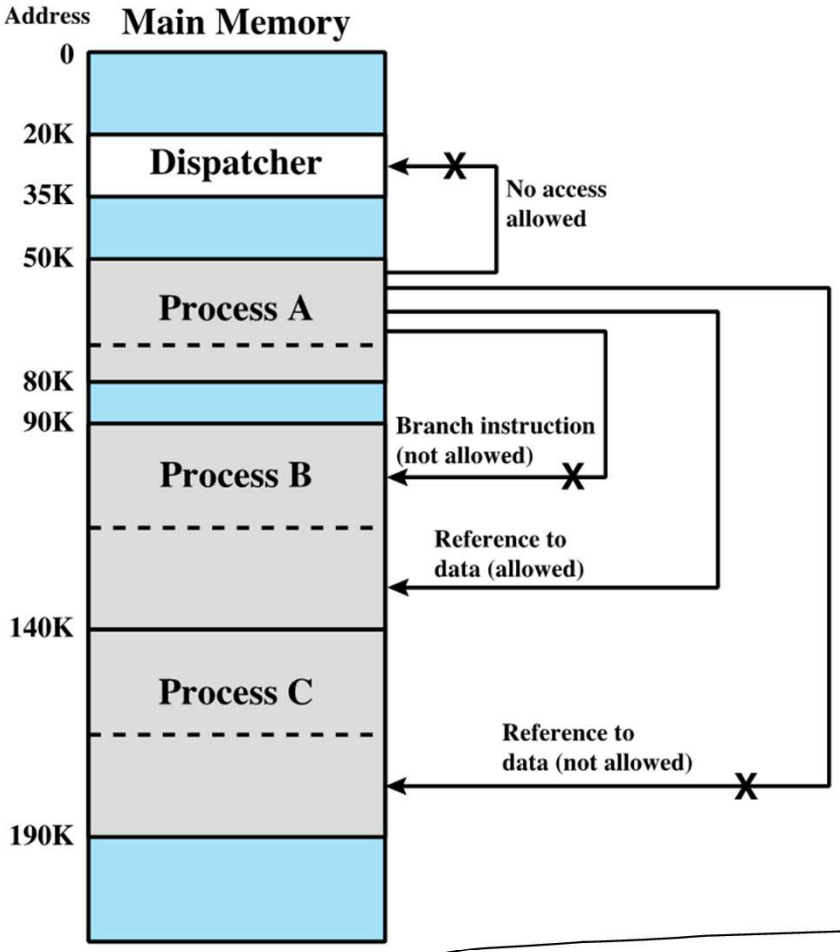
Requirements: Sharing

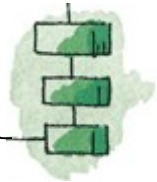
- Memory management must allow controlled access to shared areas of memory without compromising protection.
  - Better to allow each process executing the same program access to the same copy of the program rather than have their own separate copy.
  - Processes that are cooperating on some task may also need to share access to the same data structure.
- Mechanisms used to support relocation support sharing capabilities.

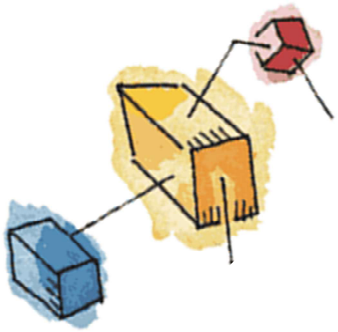




Protection and Sharing Example

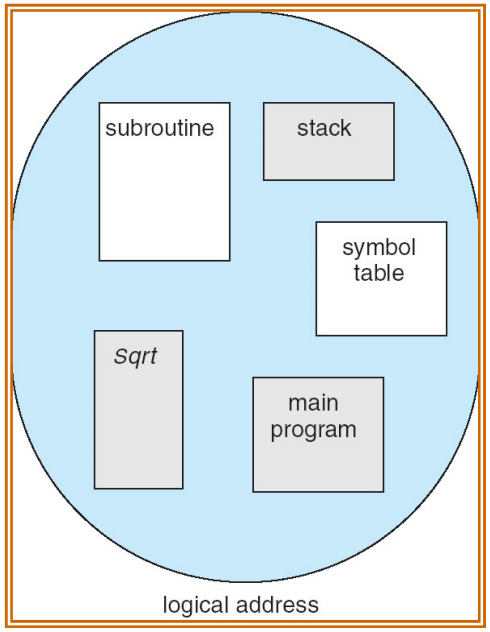


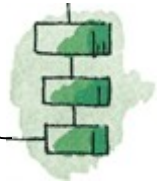


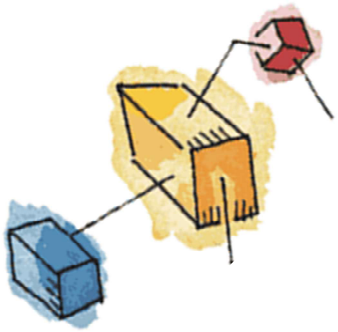


Requirements: Logical Organization

- Memory is organized linearly (usually); in contrast, programs are written in modules
- If OS and hardware can deal with user programs and data in the form of modules
  - modules can be written and compiled independently
  - different degrees of protection can be given to different modules (read-only, execute-only)
  - easy to specify sharing on a module level

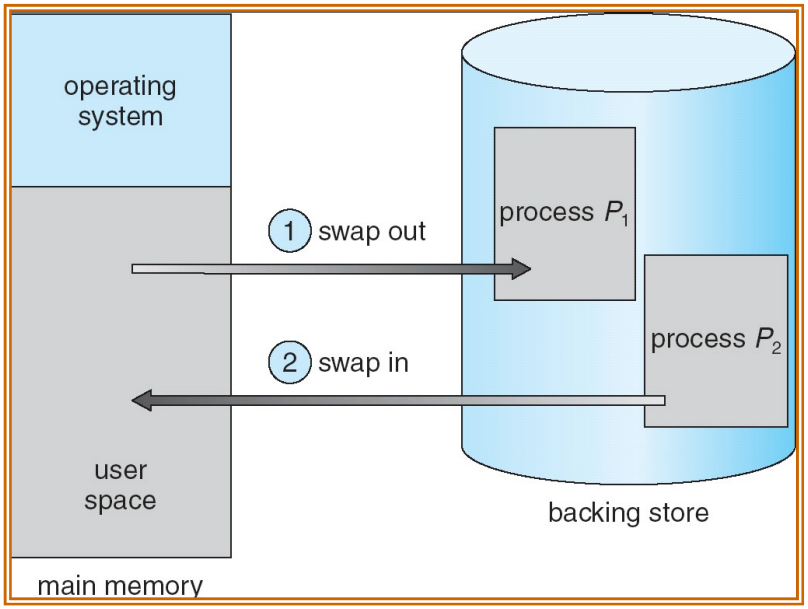


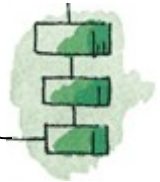


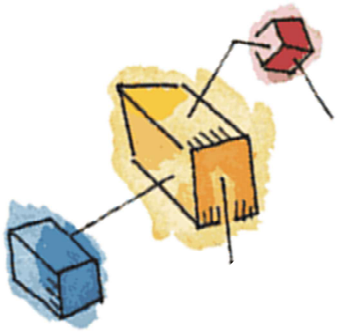


Requirements: Physical Organization

- The task of moving information between different levels of memory should be a system responsibility.
- Cannot leave the programmer with the responsibility to manage memory because
  - memory available for a program plus its data may be insufficient.
  - programmer does not know how much space will be available or where the space will be.

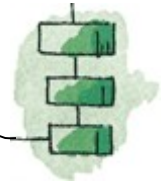


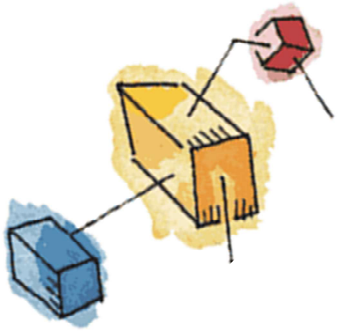




Roadmap

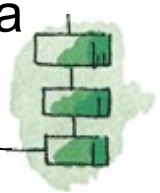
- Basic requirements of Memory Management
- Basic blocks of memory management
  - Paging
  - Segmentation
- Virtual Memory (VM) Basics
- Hardware and Control Structures of VM
  - Paging
  - Segmentation
  - Combined Paging and Segmentation
- VM Management

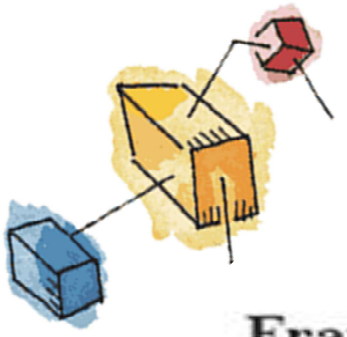




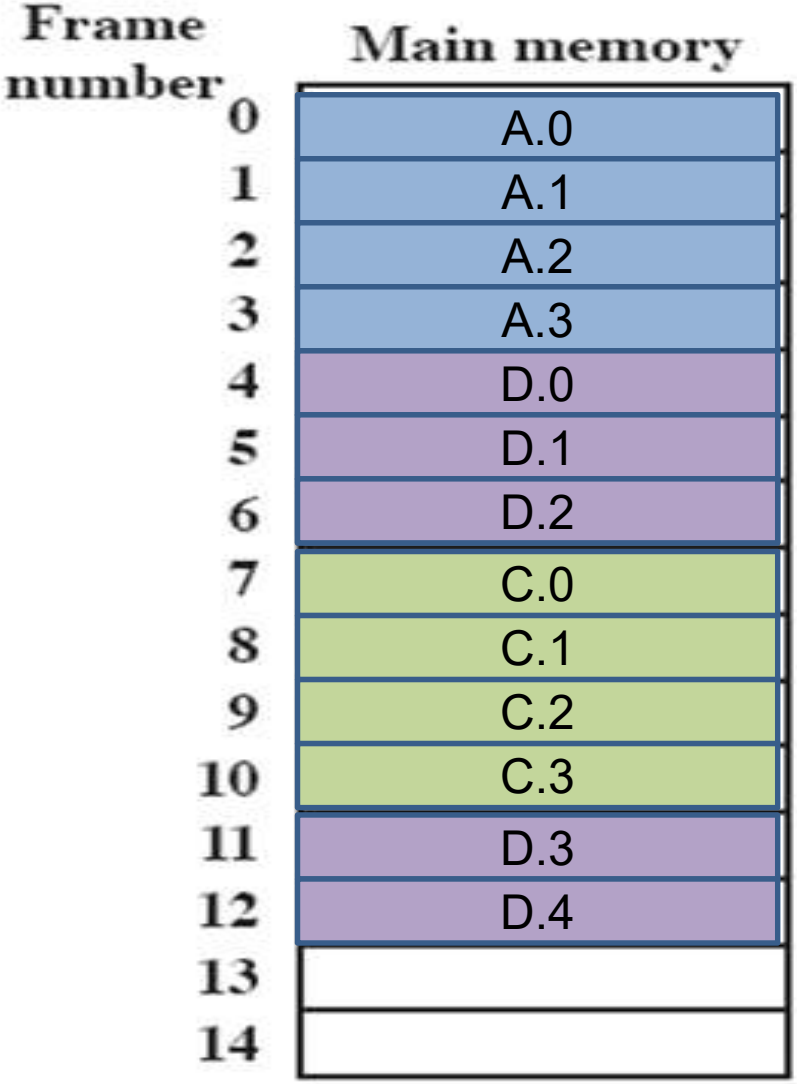
Paging

- Partition memory into small ***equal fixed-size*** chunks (***frames***) and divide each process into the ***same*** size chunks (***pages***).
  - A process is loaded by loading all of its pages into available, but ***not necessarily contiguous*** frames.
- 👍 **No** external fragmentation
  - Every frame can be allocated to a page and so there is no waste space ***outside*** frames.
- Only **little** internal fragmentation
  - There is wasted space ***inside*** the frame which consists of only a fraction of the **last** page of a process.





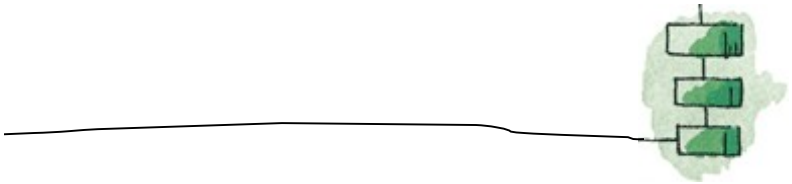
Paging
Processes and Frames

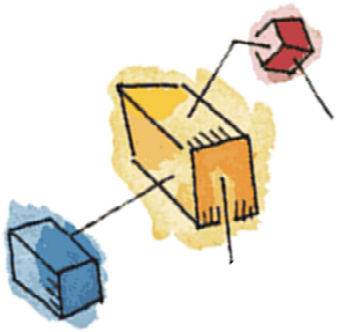


OS finds free frames and loads the pages of Process A, B & C.

Process B is suspended and is swapped out of main memory.

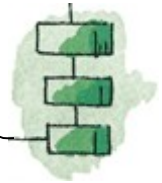
All of the processes in main memory are blocked, and OS brings in Process D.

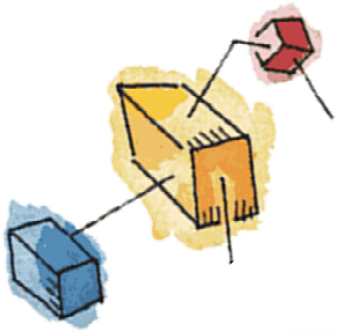




Paging
Address Translation

- OS maintains a **page table** for ***each*** process which contains the frame location for ***each*** page in the process.
- Memory reference in the program uses logical address, which consists of a **page number** and an **offset** within the page.
- During execution, the processor **translates** the logical address into a physical address by using the page table.





Paging
Page Table

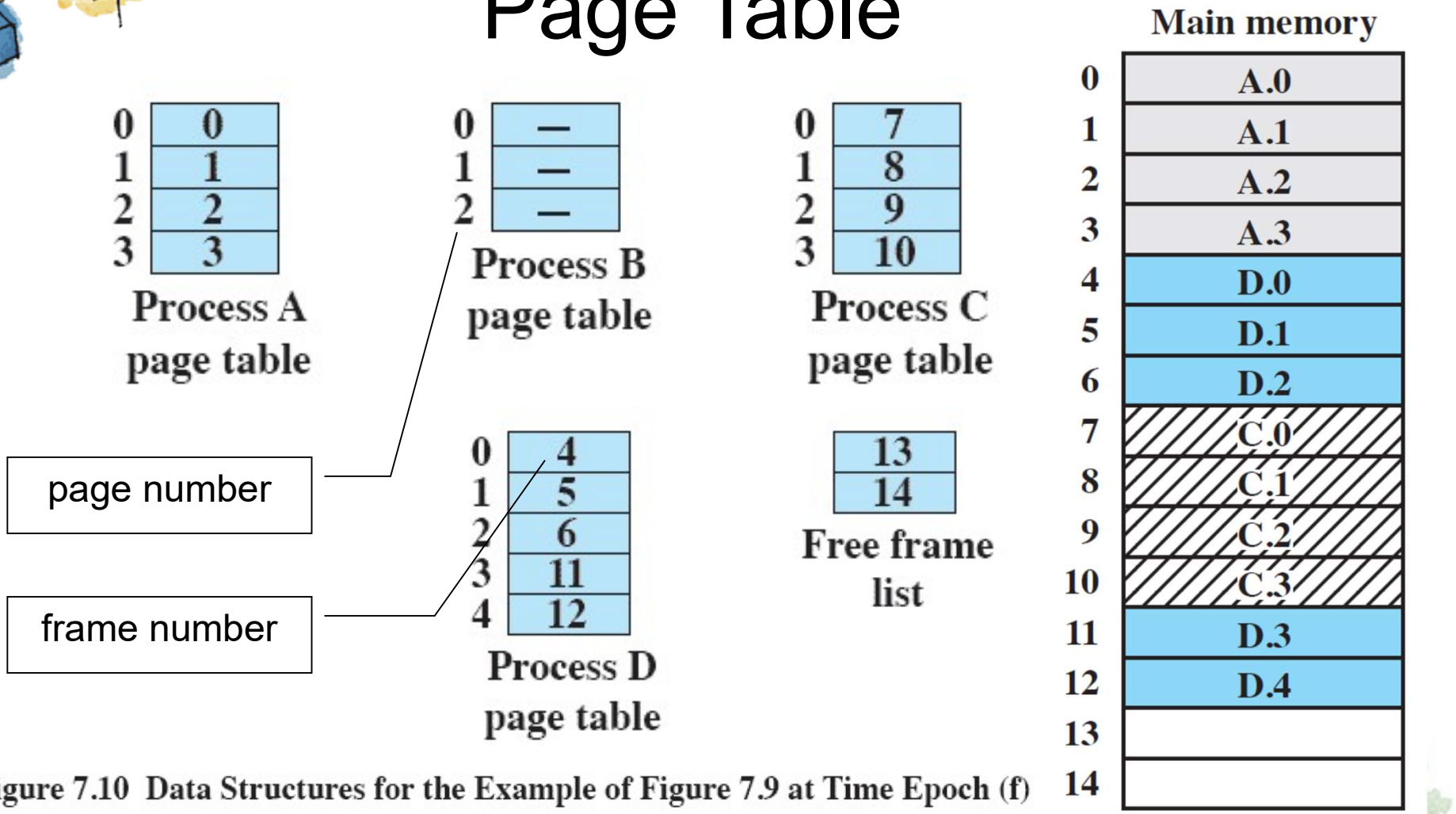
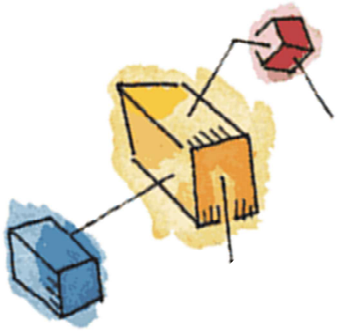
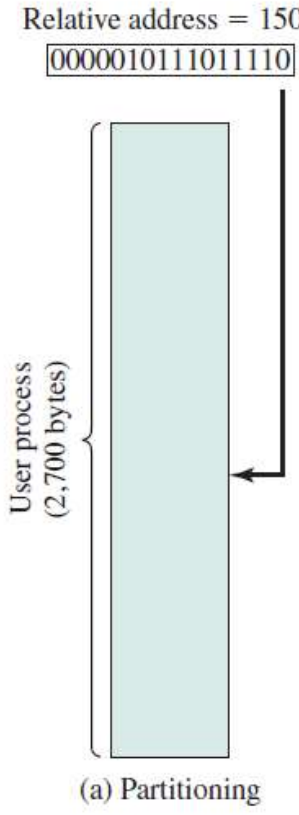


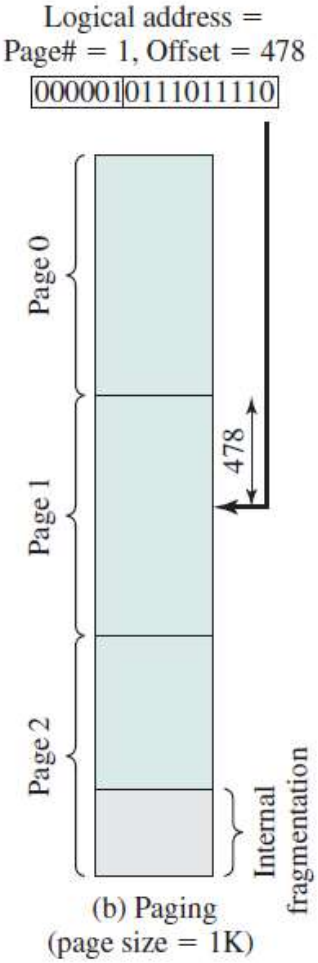
Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

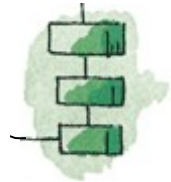


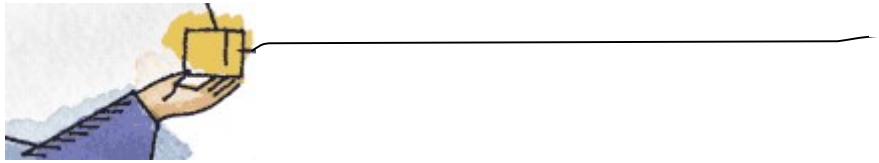
Paging
Logical Addresses

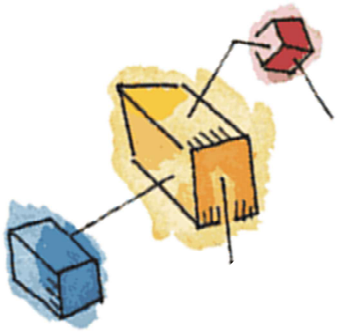
- Using a page size that is a power of 2, a logical address (page no. and offset) is identical to its relative address
- Example
  - 16-bit address
  - 2^{10} =1024-byte page
  - 10-bit offset
  - 6-bit page number
  - A maximum of 2^6 =64 pages





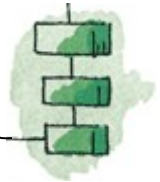


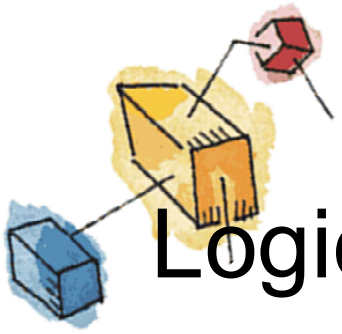




Paging
Logical Addresses

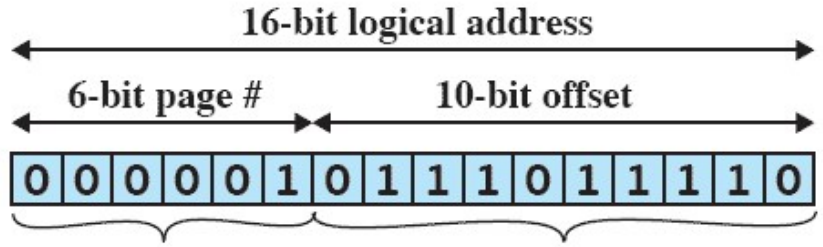
- Using a page size that is a power of 2,
  - 👍 the logical addressing scheme becomes transparent to the programmer, assembler and linker because a logical address is identical to its relative address.
  - 👍 implementing a function in hardware to perform dynamic address translation at run time becomes relatively easy.





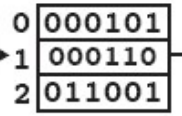
Paging

Logical to Physical Address Translation



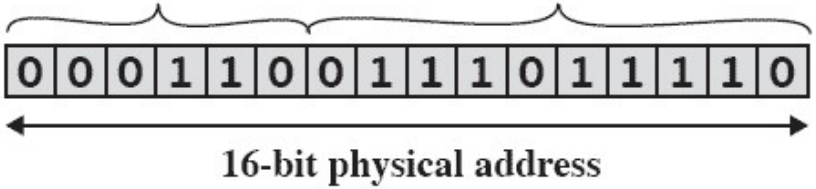
Consider an address of *n*+*m* bits, where the leftmost *n* bits are the page no. and the rightmost *m* bits are the offset.

1. Extract the page no.

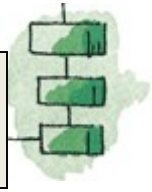


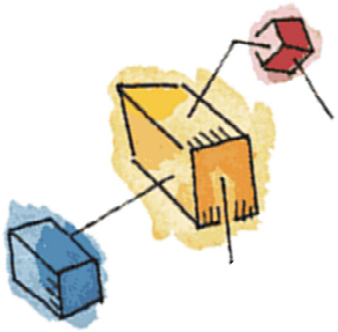
Process page table

2. Use the page no. as an index into the process page table to find the frame number, *k*.



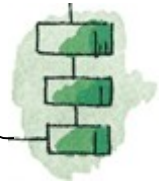
3. The physical address is constructed by appending the offset to *k*.





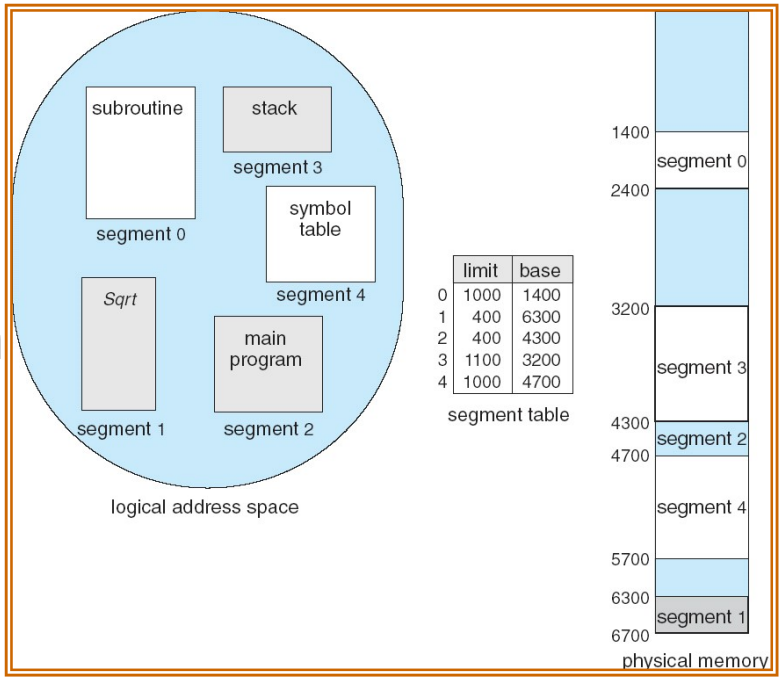
Segmentation

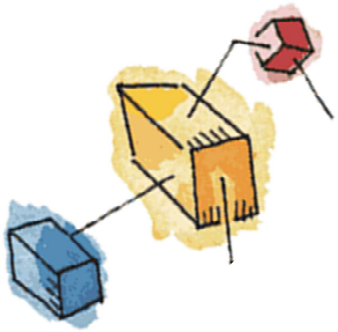
- A program can be subdivided into ***variable-sized segments*** with a maximum length.
  - A process is loaded by loading all of its segments that ***need not be contiguous***.
- Provided as a convenience for programmers to organize programs and data into different segments.
- 👍 No internal fragmentation
  - Segments are allocated exactly as much memory as required.
- 👎 Suffers from external fragmentation
  - Due to relocation, memory external to all segments becomes more and more fragmented, thus too small to fit any segment and become unusable.



Segmentation

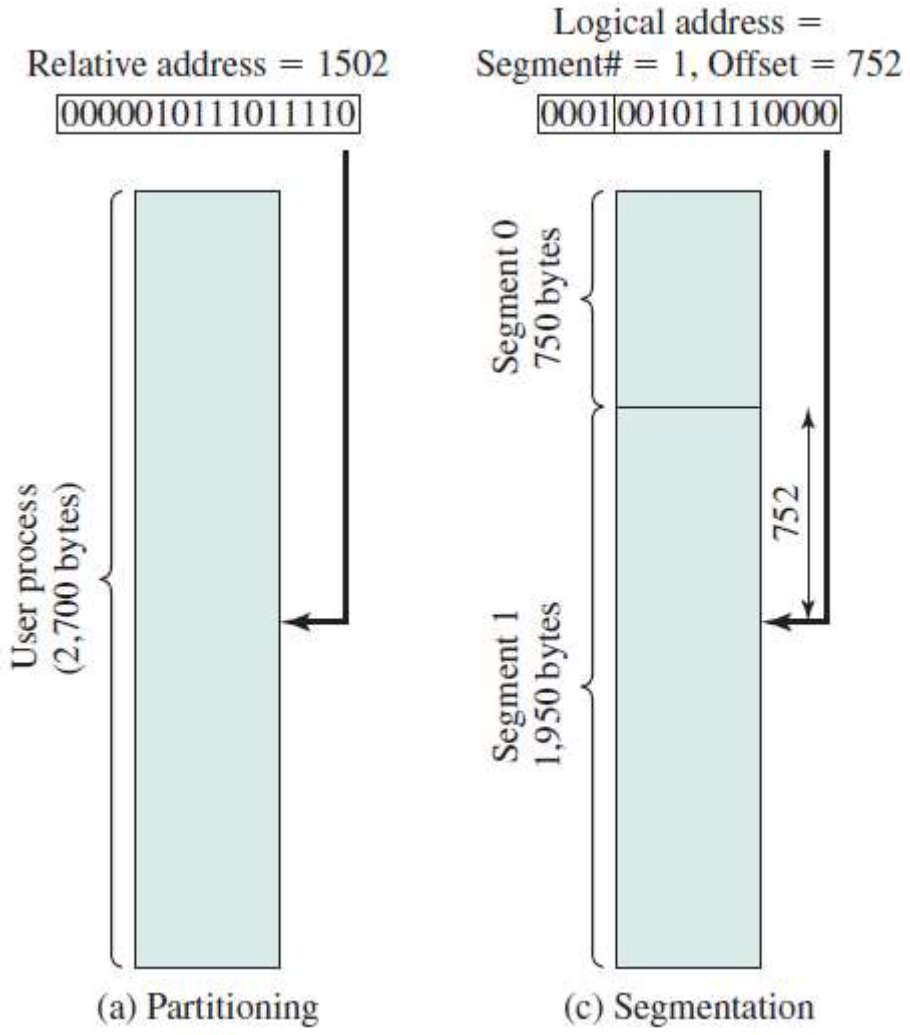
- Memory reference in the program uses logical address, which consists of a **segment number** and an **offset** within the segment.
- There is a **segment table** for **each** process.
- Each segment table entry contains
  - the starting (base) address in main memory of the corresponding segment.
  - the length (limit) of the segment, to assure that invalid addresses are not used.





Segmentation
Logical Addresses

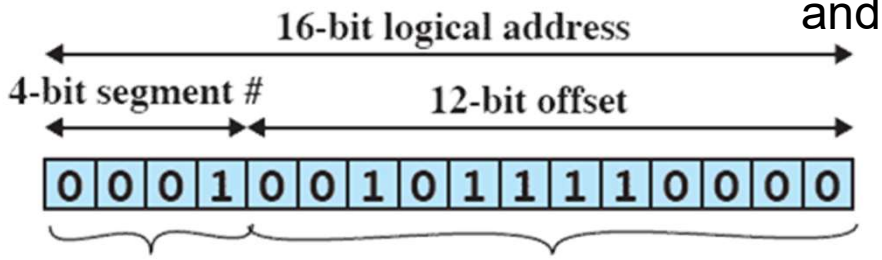
- There is no simple relationship between a logical address (segment no. and offset) and the relative address.
- Example
  - 16-bit address
  - 12-bit offset
  - 4-bit segment number
  - maximum segment size= 2^{12}=4096



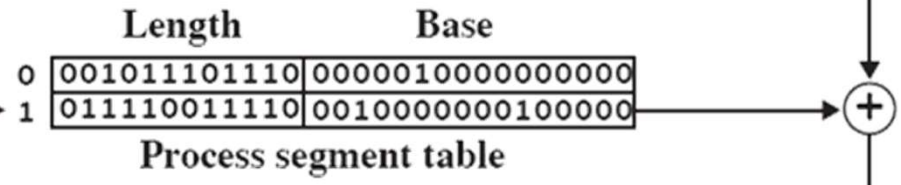
Segmentation

Logical to Physical Address Translation

Consider an address of $n + m$ bits, where the leftmost n bits are the segment no. and the rightmost m bits are the offset.



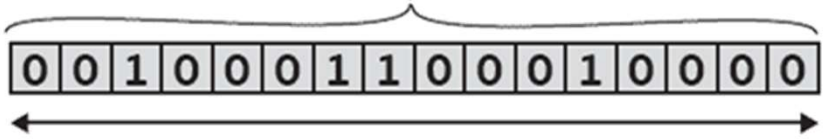
1.Extract the segment no.

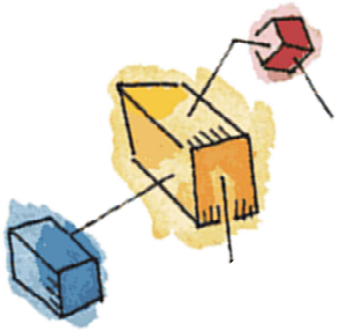


3. If the offset ≥ the length of the segment, the address is invalid.

4.The physical address is the sum of the starting physical address of the segment plus the offset.

2.Use the segment no. as an index into the process segment table to find the starting physical address of the segment.



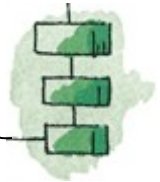


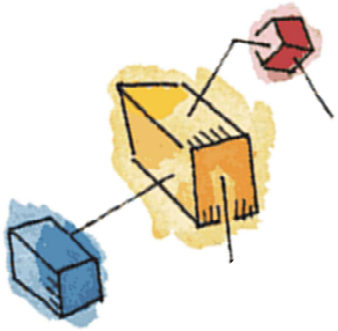
Roadmap

- Basic requirements of Memory Management
- Basic blocks of memory management
  - Paging
  - Segmentation

→ Virtual Memory (VM) Basics

- Hardware and Control Structures of VM
  - Paging
  - Segmentation
  - Combined Paging and Segmentation
- VM Management



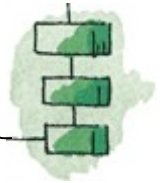


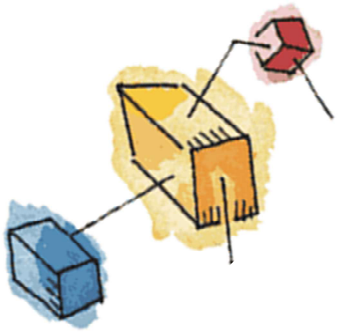
Key Points in Memory Management

1) Memory references are logical addresses that are dynamically translated into physical addresses at run time.

2) A process may be broken up into pieces (***pages*** or ***segments***) that do not need to be contiguously located in main memory during execution.

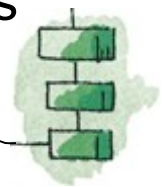
If these two characteristics are present, it is ***not*** necessary that all of the pages or segments of a process be in main memory during execution.

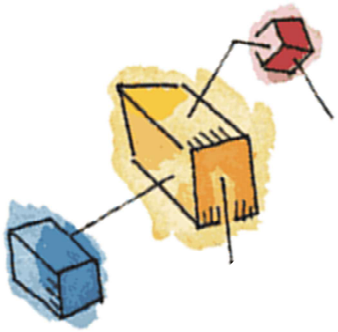




Execution of a Process

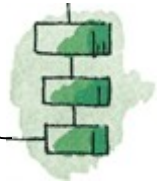
- OS brings into main memory *a few pieces* of the program.
  - ***Resident set***: portion of process that is in main memory
- Execution proceeds smoothly as long as all memory references are to locations that are in the resident set.
- When a needed address is not in main memory, an interrupt (***memory access fault***) is generated.
- OS places the interrupted process in a *Blocked* state.
- OS issues a disk I/O Read request.
- Another process is dispatched to run while the disk I/O takes place.
- Piece of the interrupted process that contains the logical address is brought into main memory.
- Another interrupt is issued when disk I/O is complete, which causes OS to place the interrupted process in the *Ready* state.

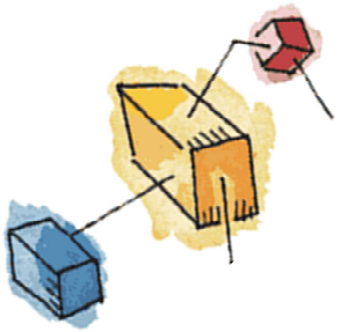




Implications

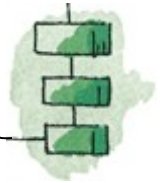
- More processes may be maintained in main memory.
  - Only some of the pieces of any particular process are loaded, there is room for more processes.
  - More efficient utilization of the processor because it is more likely that at least one of the many processes will be in a Ready state at any particular time.
- A process may be larger than all of main memory.
  - This restriction in programming is lifted.
  - OS automatically loads pieces of a process into main memory as required.

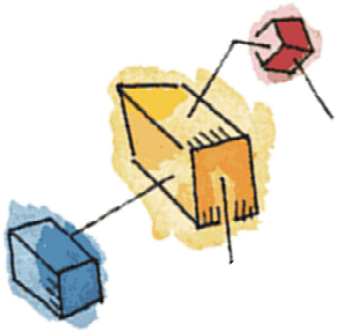




Support Needed for Virtual Memory

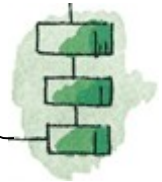
- Hardware must support ***paging*** and ***segmentation*** for address translation and other basic functions.
- OS must manage the movement of pages and/or segments between secondary memory and main memory.

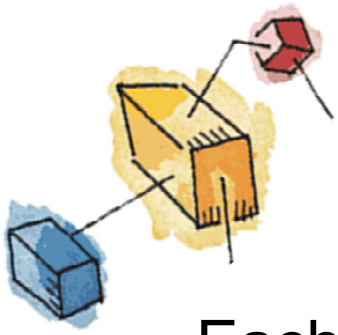




Roadmap

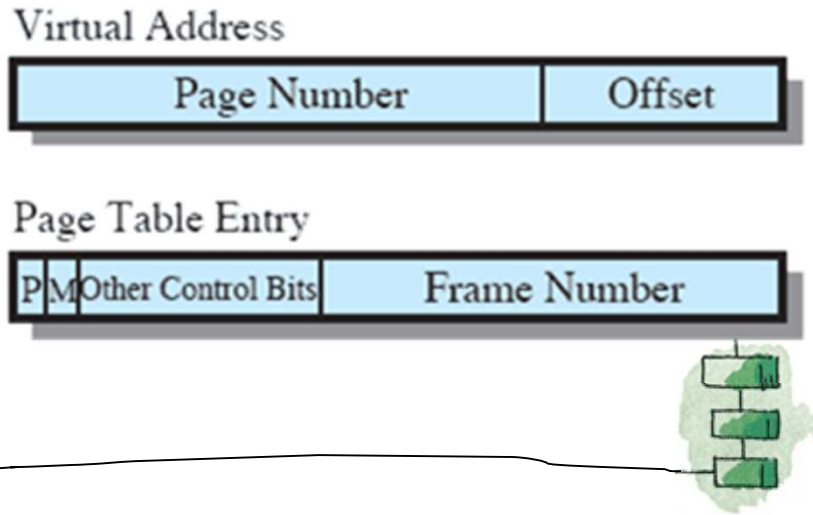
- Basic requirements of Memory Management
- Basic blocks of memory management
  - Paging
  - Segmentation
- Virtual Memory (VM) Basics
- Hardware and Control Structures of VM
  - Paging
  - Segmentation
  - Combined Paging and Segmentation
- VM Management





Paging in VM

- Each process has its own page table.
- Each page table entry (PTE) contains the **frame number** of the corresponding page in main memory.
- Two extra bits are needed to indicate:
  - **P**(resent): whether the page is in main memory or not
    - If a desired page is not in main memory, a memory access fault, called a ***page fault***, occurs
  - **M**(odified): whether the contents of the page has been altered since it was last loaded
    - It is not necessary to write an unmodified page out when it comes to time to replace the page



Address Translation

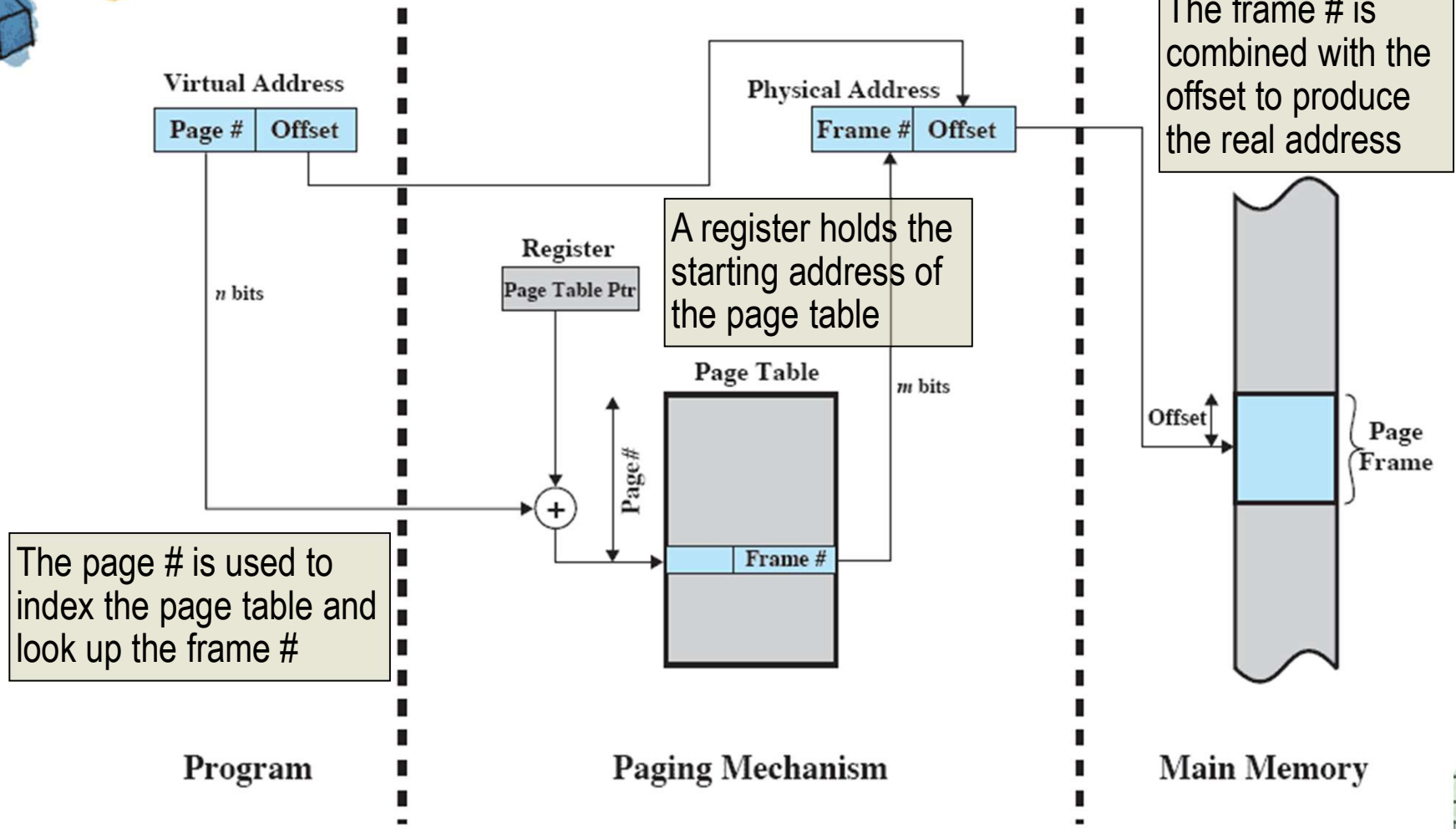
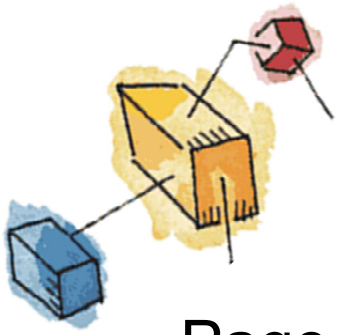


Figure 8.3 Address Translation in a Paging System



Page Tables

- Page tables can be very large
  - Consider a system that supports 4-Gbytes (2^{32}) virtual address space with 4-kbyte (2^{12}) pages. The number of page table entries (PTEs) in a page table can be as many as 2^{20}

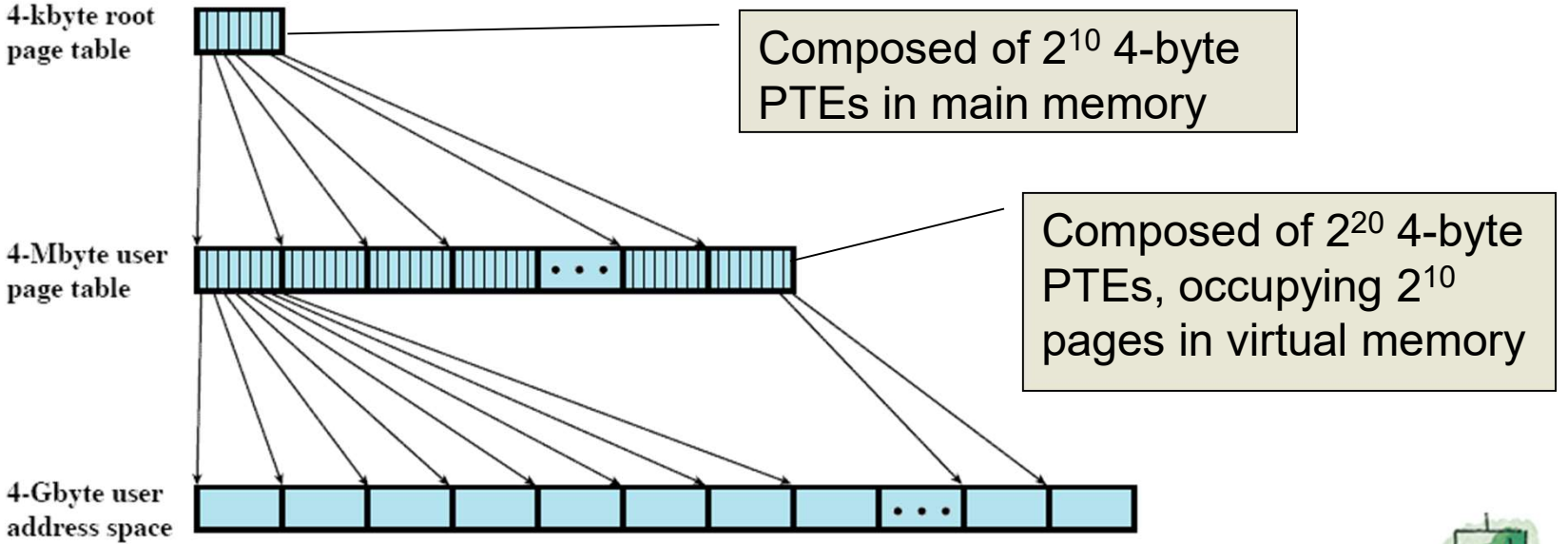


Figure 8.4 A Two-Level Hierarchical Page Table

Address Translation for Hierarchical Page Table

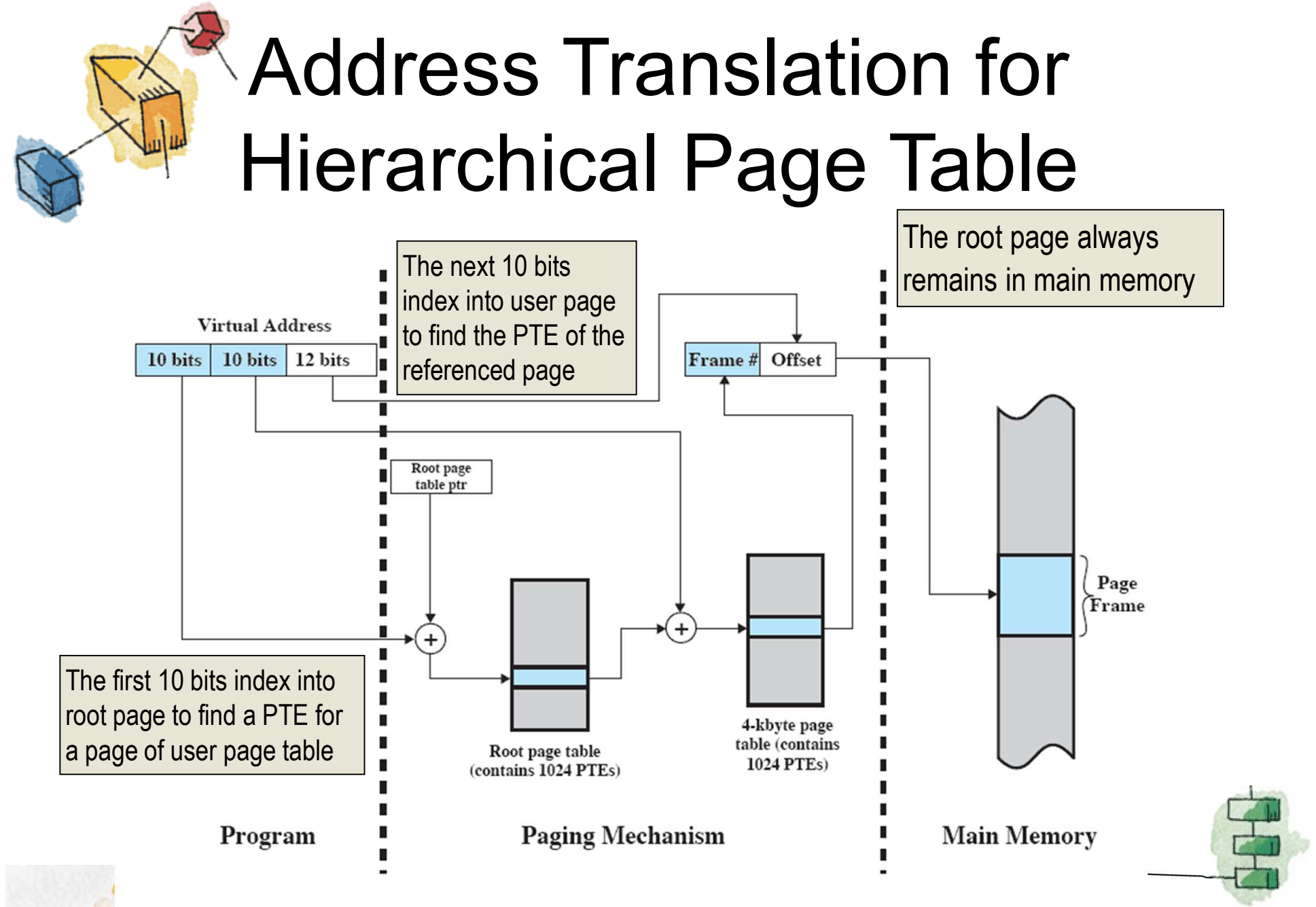
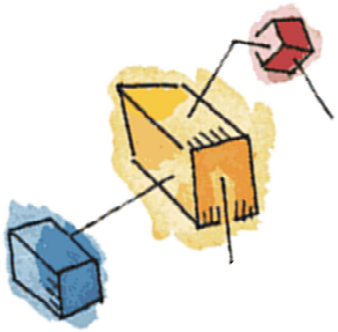
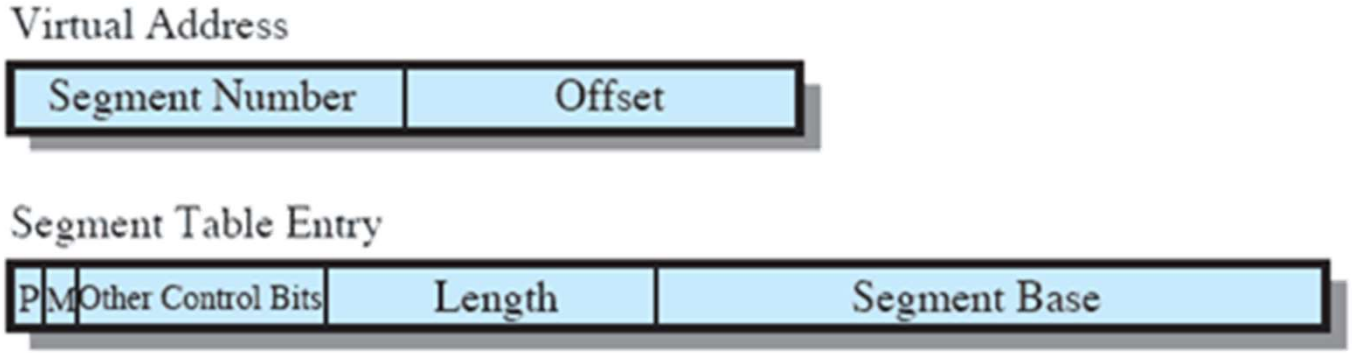


Figure 8.5 Address Translation in a Two-Level Paging System

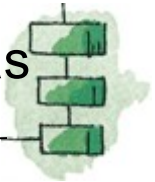


Segmentation in VM

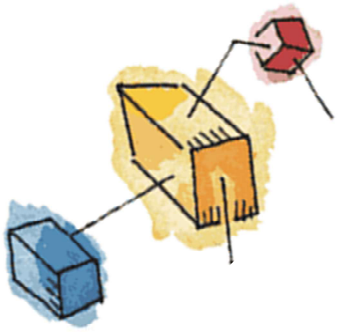
- Each process has its own segment table.



- Each segment table entry contains
  - **Segment base**: the starting address of the corresponding segment in main memory
  - The **length** of the segment
  - **P-bit** : determines if the segment is already in main memory
  - **M-bit:** determines if the segment has been modified since it was loaded in main memory



Address Translation in Segmentation



The segment base is added to the offset to produce the real address

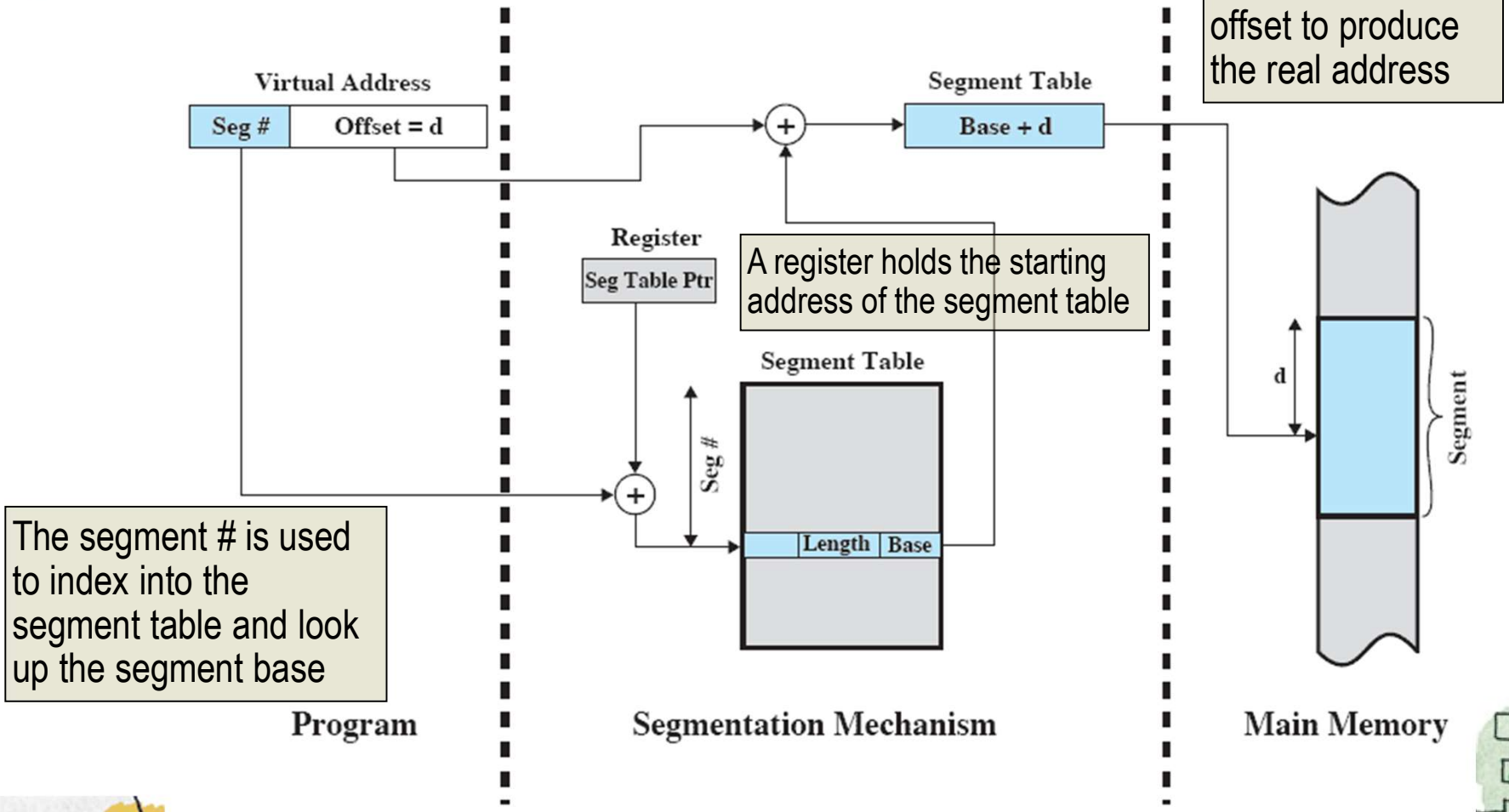
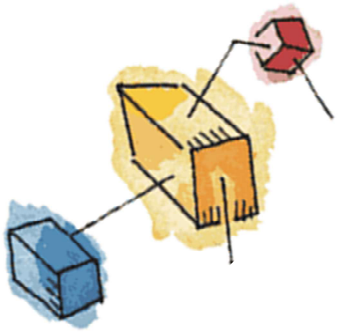
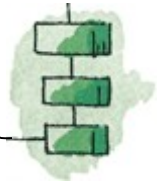


Figure 8.12 Address Translation in a Segmentation System

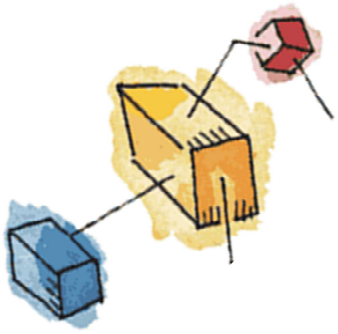


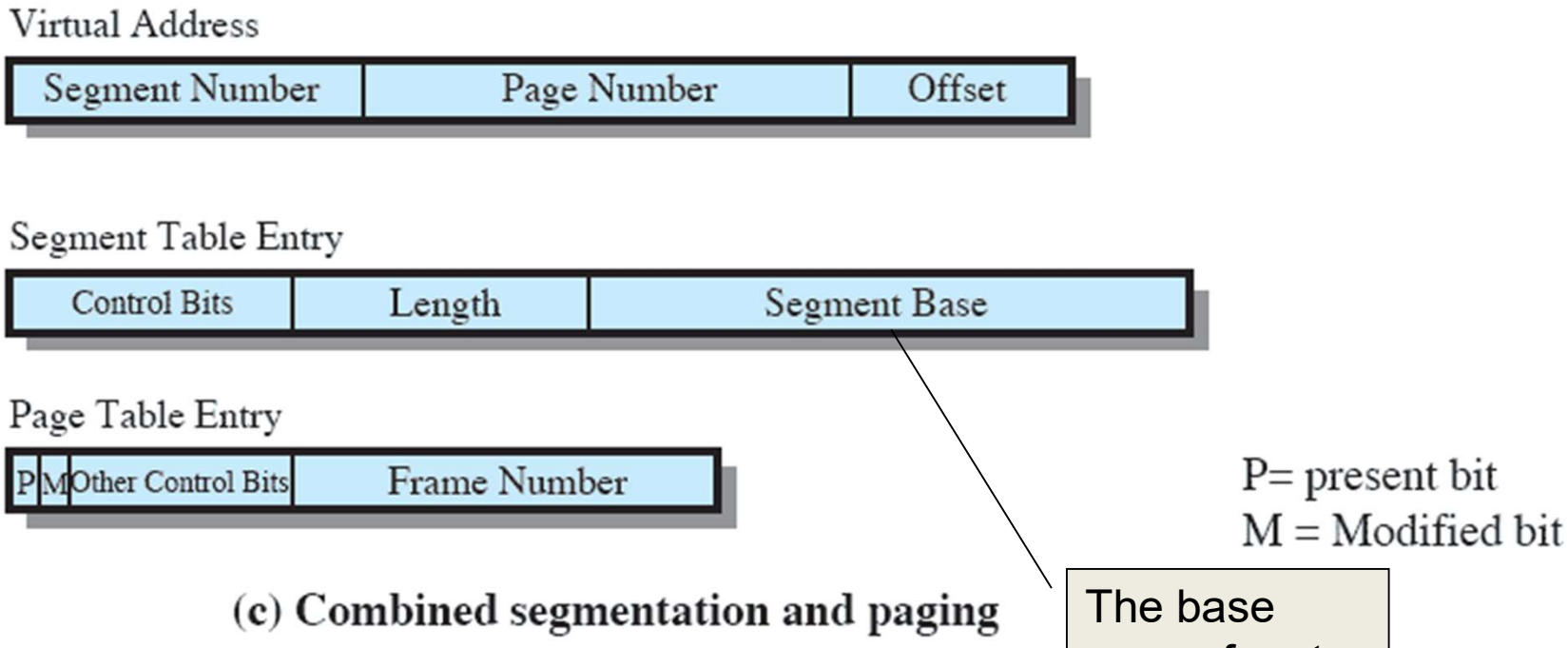
Combined Paging and Segmentation

- A user's address space is broken up into a number of segments and each segment is broken up into a number of fixed-size pages.
- From the programmer's point of view, a logical address still consists of a segment number and a segment offset.
  - Segmentation is visible to the programmer.
- From the system's point of view, the segment offset is viewed as a page number and a page offset.
  - Paging is transparent to the programmer.

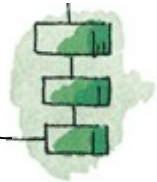


Combined Paging and Segmentation





(c) Combined segmentation and paging



Address Translation

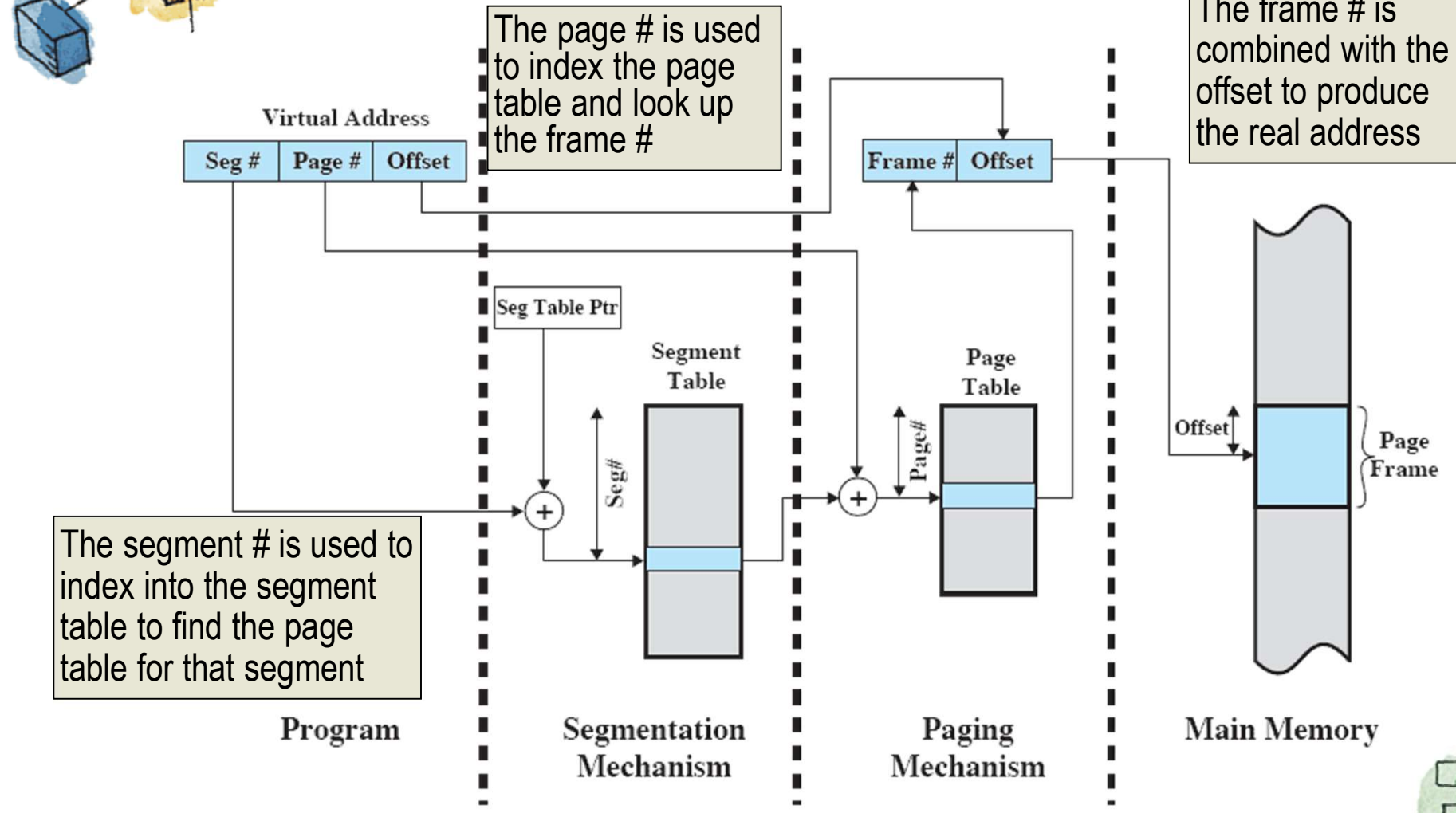
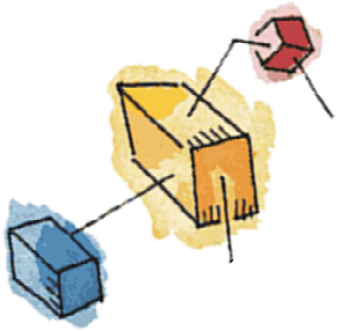


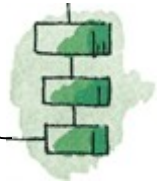
Figure 8.13 Address Translation in a Segmentation/Paging System

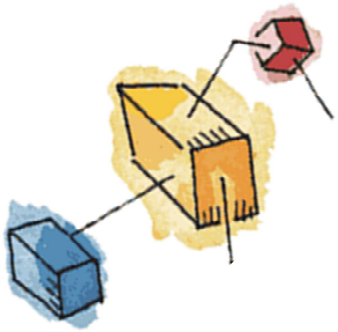


Roadmap

- Basic requirements of Memory Management
- Basic blocks of memory management
  - Paging
  - Segmentation
- Virtual Memory (VM) Basics
- Hardware and Control Structures of VM
  - Paging
  - Segmentation
  - Combined Paging and Segmentation

→ VM Management

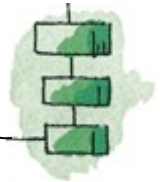


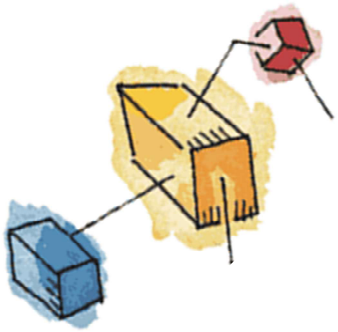


OS Policies for VM

- Fetch policy
- Placement policy
- Replacement policy
- Resident set management
- Cleaning policy
- Load control

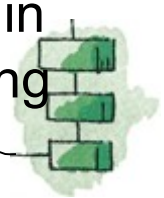
Key aim: to minimize page faults

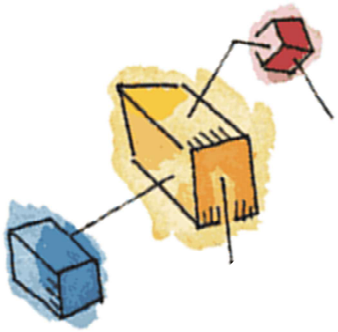




Fetch Policy

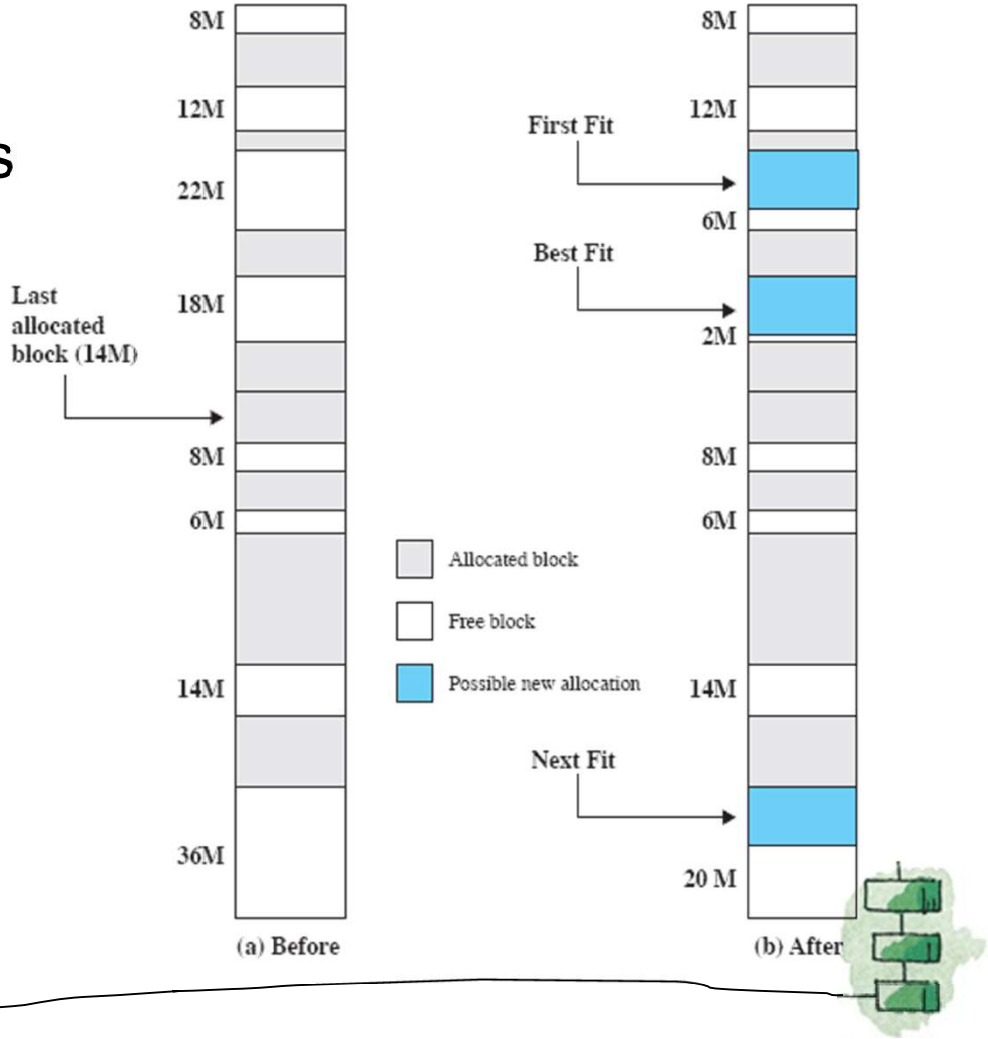
- Determines ***when*** a page should be brought into memory.
- Demand Paging
  - only brings pages into main memory when a reference is made to a location on the page
  - many page faults when process first started
  - but, as more and more pages are brought in, due to the ***principle of locality***, most future references will be to pages that have recently been brought in, and page faults should drop to a very low level
    - Principle of locality: program and data references within a process tend to cluster, so, only a few pieces of a process will be needed over a short period of time
- Prepaging
  - pages other than the one demanded by a page fault are brought in
  - if pages are stored contiguously on disk, it is more efficient to bring in a number of pages at one time

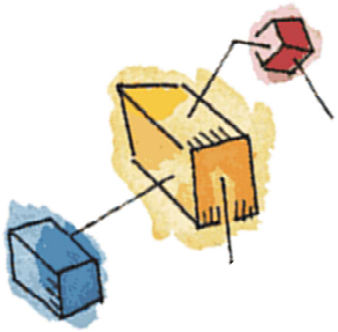




Placement Policy

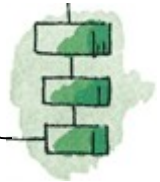
- Determines ***where*** in real memory a process piece is to reside.
- Irrelevant for pure paging or combined paging with segmentation scheme.
- Important in a segmentation system
  - best-fit, first-fit, next fit

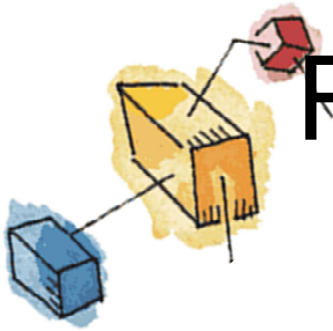




Replacement Policy

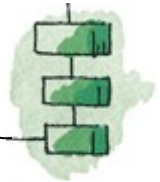
- When all of the frames in main memory are occupied, the replacement policy determines ***which page*** in memory to be replaced when a new page must be brought in.
- Objective: the page that is removed should be the page least likely to be referenced in the near future.
- Tradeoff: the more sophisticated the replacement policy, the greater the overhead to implement it.

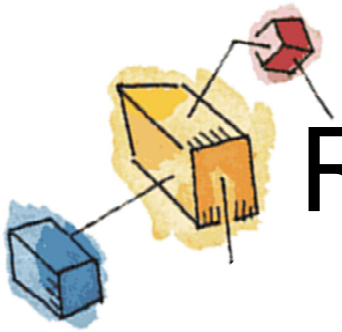




Replacement Restriction: Frame Locking

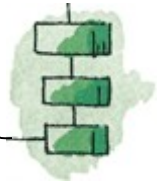
- Some of the frames in main memory may be locked. When a frame is locked, the page currently stored in that frame may not be replaced.
  - Kernel of the OS
  - Key control structures
  - I/O buffers
- Locking is achieved by associating a lock bit with each frame.

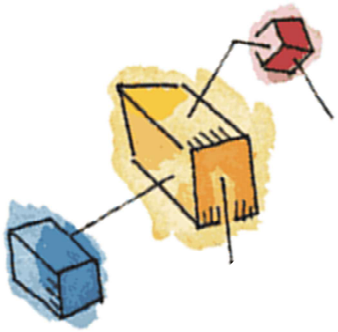




Replacement Algorithms

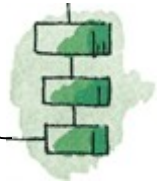
- Algorithms used for the selection of a page to replace:
  - Optimal
  - First-in-first-out (FIFO)
  - Least recently used (LRU)
  - Clock
- Consider the following page address stream formed by executing a program with three frames allocated.
  - 2 3 2 1 5 2 4 5 3 2 5 2
  - Which means that the first page referenced is 2, the second page referenced is 3, and so on.

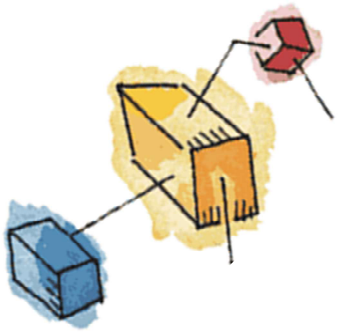




Replacement Policy
Optimal

- Selects the page for which the time to the ***next*** reference (i.e., the future reference) is the ***longest***.
- Results in the fewest number of page faults but it is impossible to have perfect knowledge of future events.
- Serves as a standard to judge real-world practical algorithms.





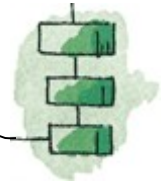
Optimal Replacement Example

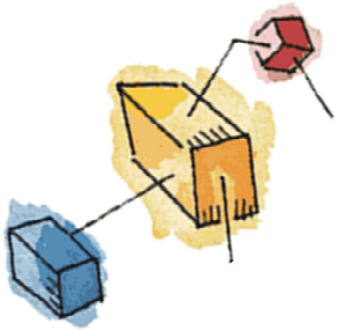
Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					F		F			F		

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

Figure 8.14 Behavior of Four Page-Replacement Algorithms

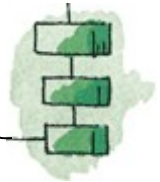
- The optimal policy produces 3 page faults after the frame allocation has been filled

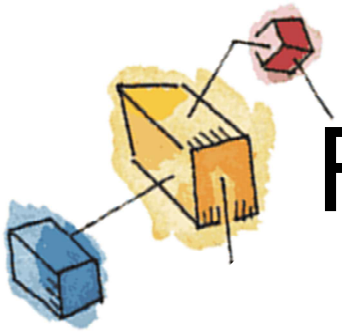




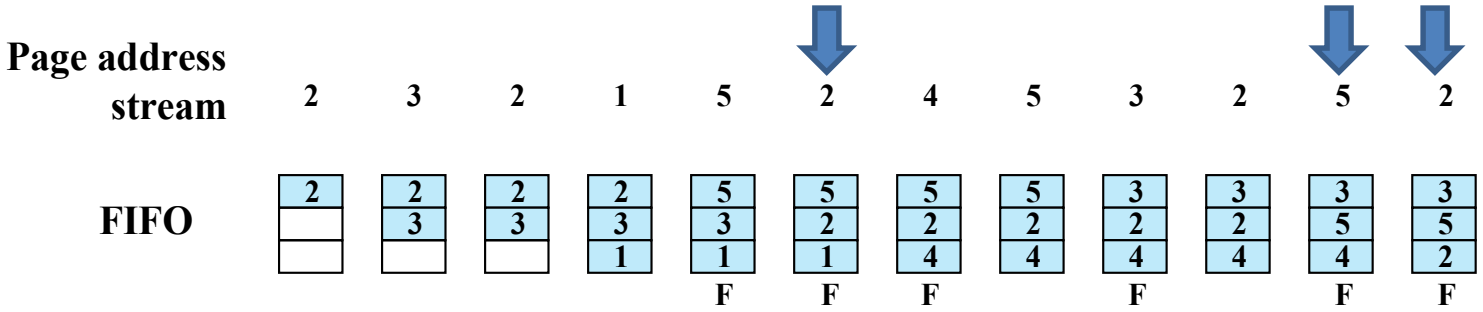
Replacement Policy
FIFO

- FIFO (First In First Out) policy treats page frames allocated to a process as a ***circular buffer***.
- Pages are removed in round-robin style.
  - Simplest replacement policy to implement (only requires a pointer circle through the page frames)
- Page that has been in memory the ***longest*** is replaced.
  - But, these pages may be needed again very soon if it hasn't truly fallen out of use.





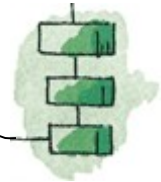
FIFO Replacement Example

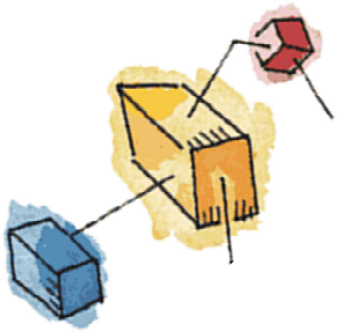


F = page fault occurring after the frame allocation is initially filled

Figure 8.14 Behavior of Four Page-Replacement Algorithms

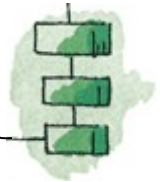
- The FIFO policy results in 6 page faults.
  - Note that FIFO does not recognize that pages 2 and 5 are referenced more frequently than other pages.



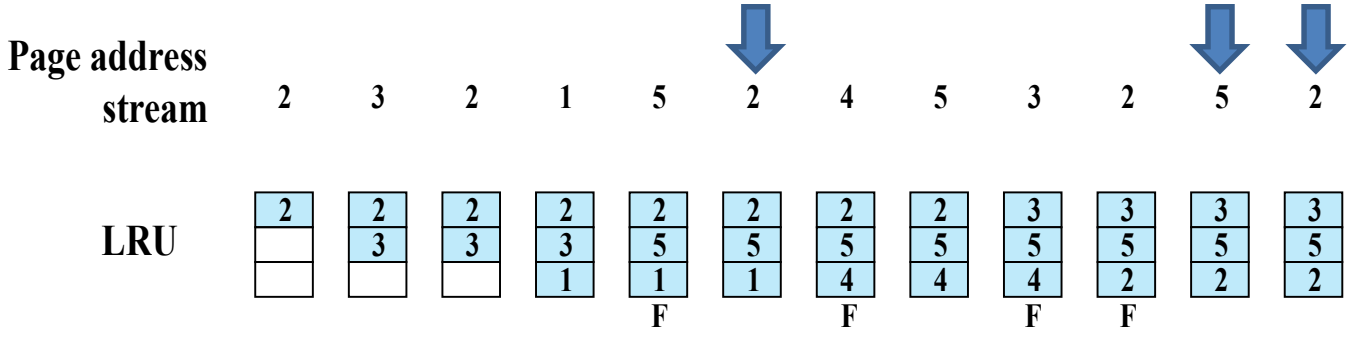


Replacement Policy
Least Recently Used

- LRU (Least Recently Used) policy replaces the page that has not been referenced for the ***longest*** time.
- By the principle of locality, this should be the page ***least*** likely to be referenced in the near future.
- Difficult to implement
  - One approach is to tag each page with the time of last reference.
  - This requires a great deal of overhead.



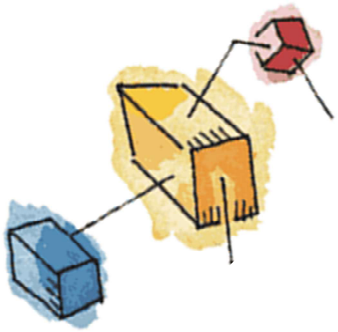
LRU Replacment Example



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

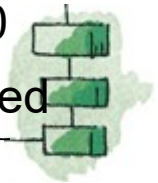
Figure 8.14 Behavior of Four Page-Replacement Algorithms

- The LRU policy does nearly as well as the optimal policy.
  - In this example, there are 4 page faults.

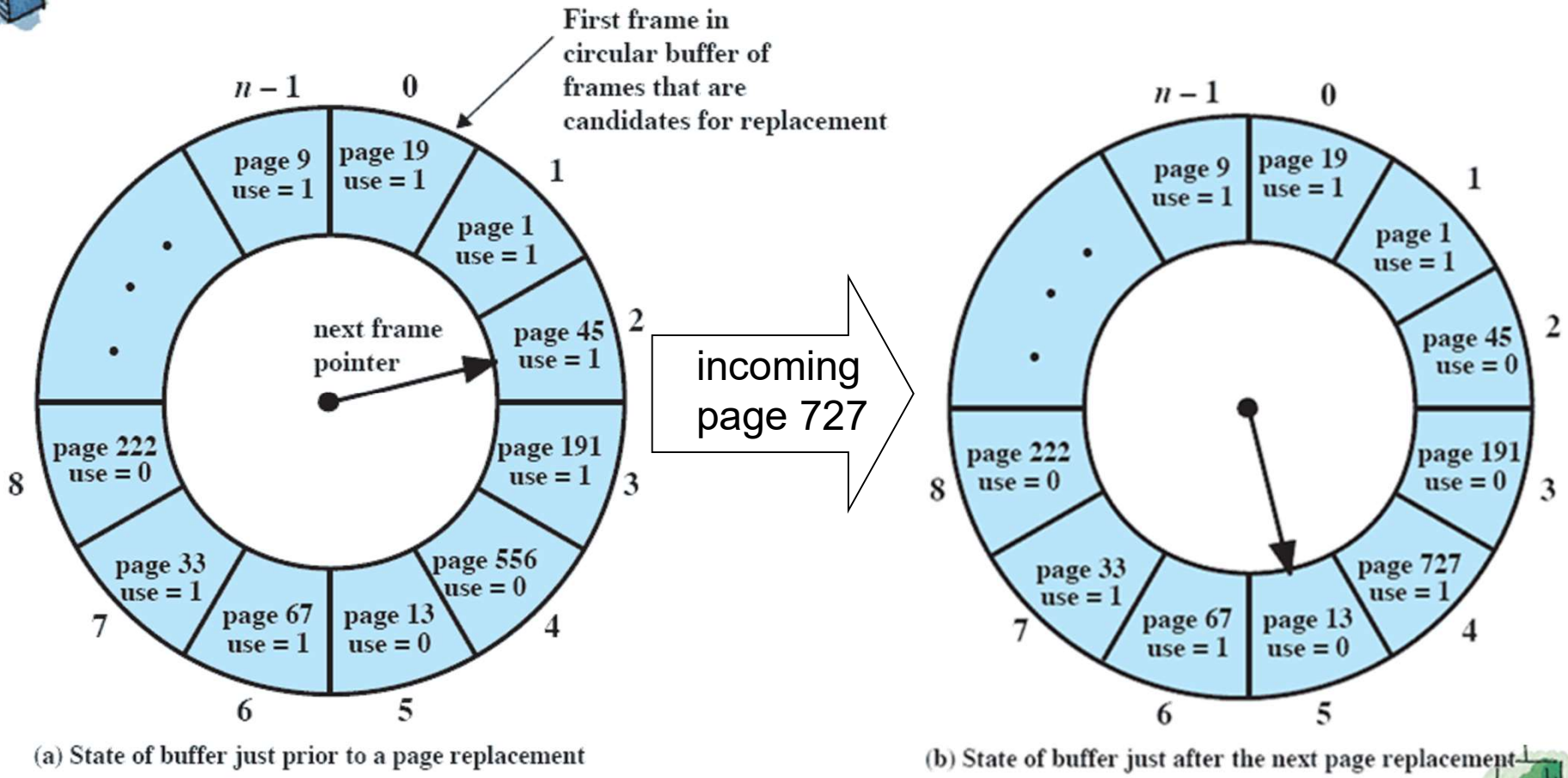


Replacement Policy
Clock

- Clock replacement policy associates an additional bit called a ***use bit*** with each memory frame.
- Similar to FIFO, except any frame with a use bit of 1 is passed over.
- The set of frames is considered to be a circular buffer with a pointer set to the ***next*** frame after the page just replaced.
  - When a page is first loaded in memory or referenced, the use bit is set to 1.
  - When it is time to replace a page, OS scans the set of frames,
    - any frame with a use bit of 1 is passed over and resetting the bit to 0
    - the first frame encountered with the use bit already being 0 is replaced



Clock Replacment Example

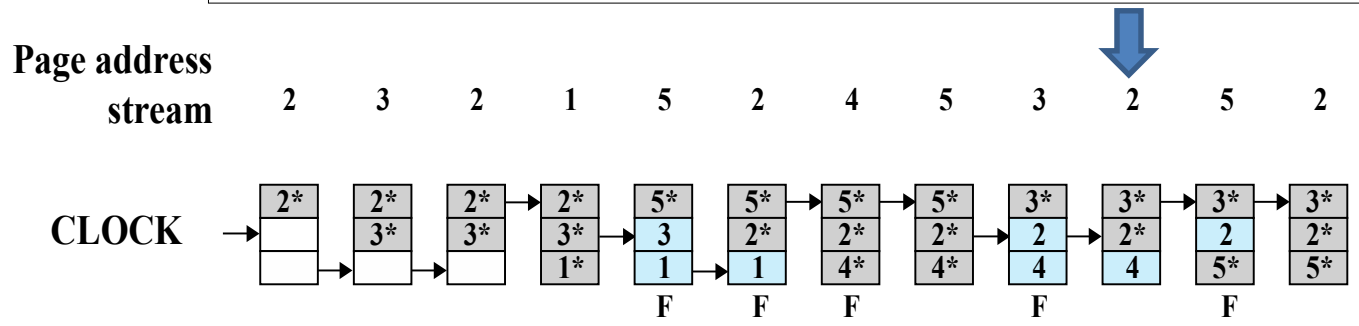


(a) State of buffer just prior to a page replacement

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

Clock Replacement Example

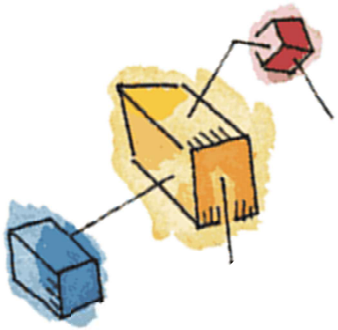
An asterisk indicates that the corresponding use bit is equal to 1.
The arrow indicates the current position of the pointer.



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

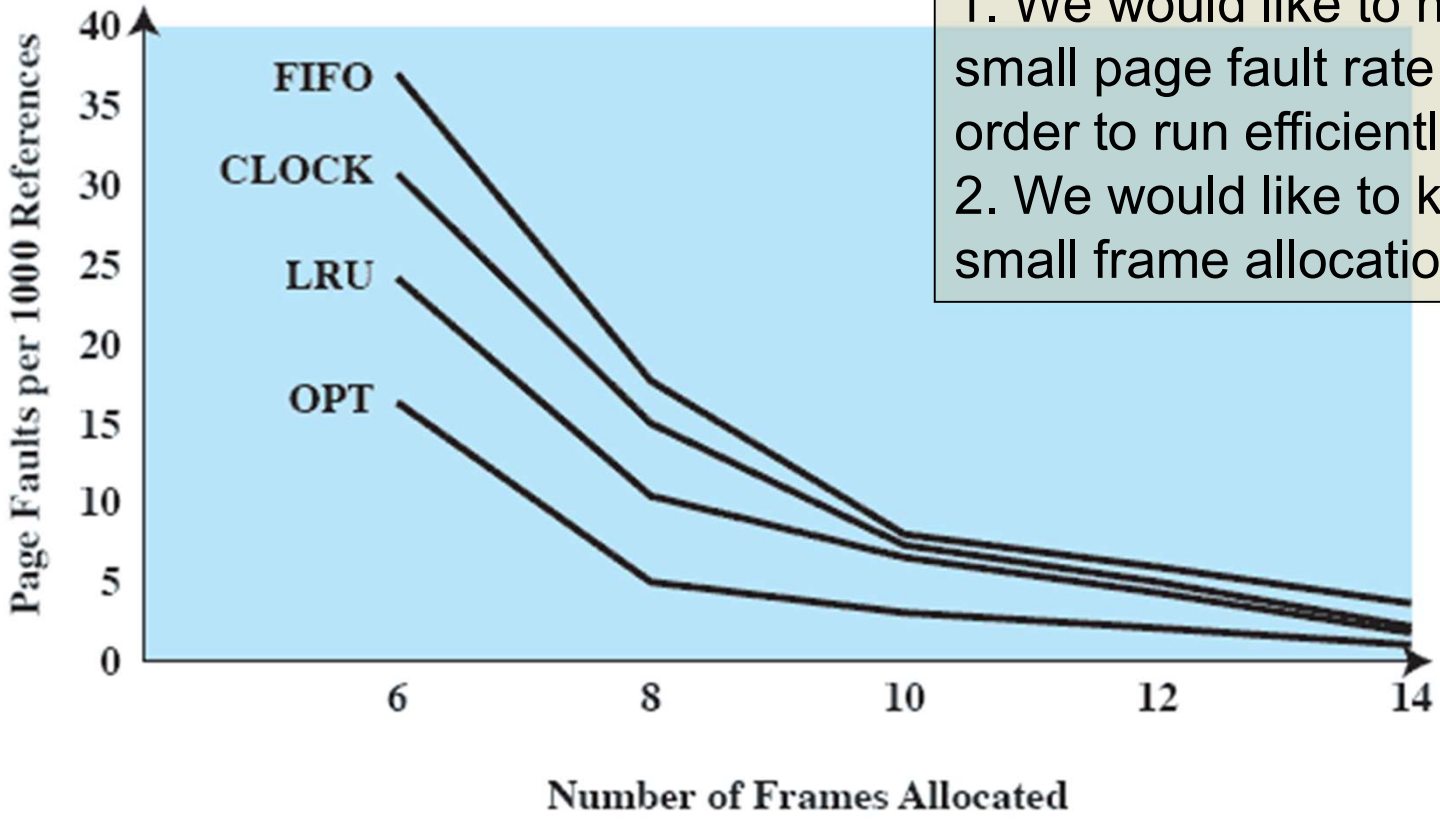
Figure 8.14 Behavior of Four Page-Replacement Algorithms

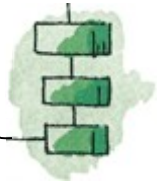
- Note that the clock policy is good at protecting frame 2 from replacement.

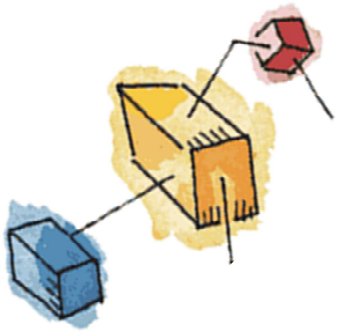


Comparison

Two conflicting constraints:
1. We would like to have a small page fault rate in order to run efficiently
2. We would like to keep a small frame allocation

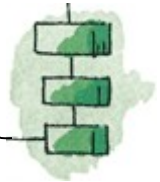


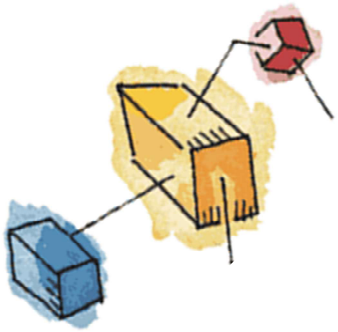




Cleaning Policy

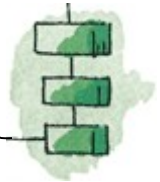
- A cleaning policy determines ***when*** a modified page should be written out to secondary memory.
- Two approaches
  - Demand cleaning
  - Precleaning

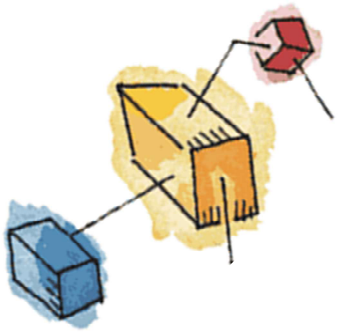




Cleaning Policy

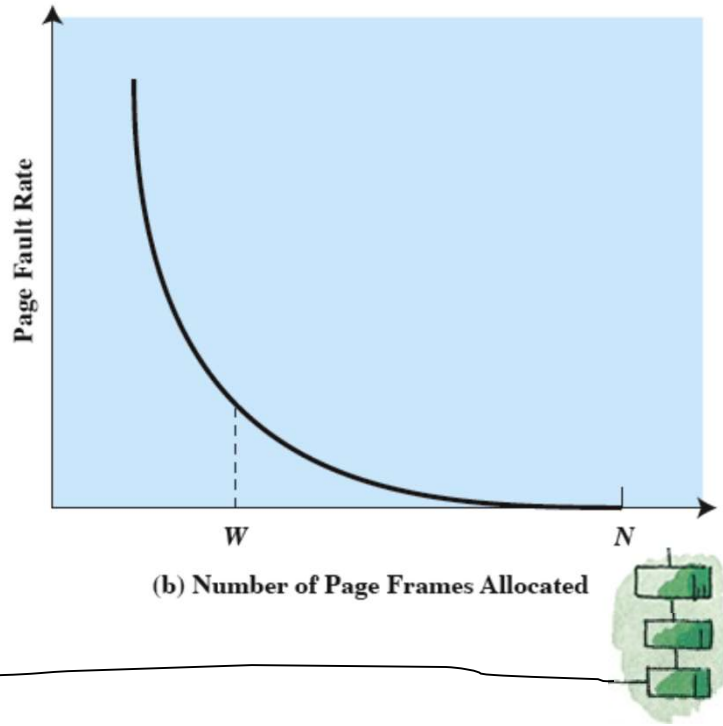
- Demand cleaning
  - A page is written out to secondary memory only **when** it has been selected for replacement.
  - 👍 Minimizes page writes.
  - 👎 A process that suffers a page fault may have to wait for *two* page transfers before it can be unblocked.
- Precleaning
  - Modified pages are written out **before** their frames are needed.
  - 👍 Pages can be written out in batches.
  - 👎 Pages written out may have been modified again before they are replaced → waste of I/O operations with unnecessary cleaning operations.

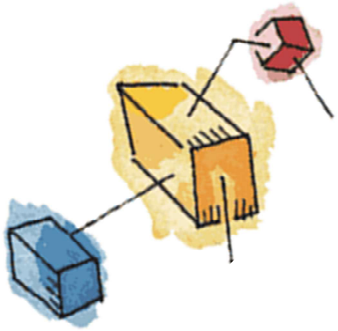




Resident Set Management

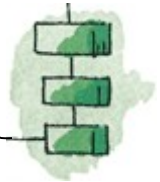
- The OS must decide ***how many pages*** to bring into main memory (how much main memory to allocate to a particular process).
  - The smaller the amount of memory allocated to each process, the more processes that can reside in memory.
  - Small number of pages loaded increases page faults.
  - Beyond a certain number, further allocations of pages will have no noticeable effect on the page fault rate for that process because of the principle of locality.

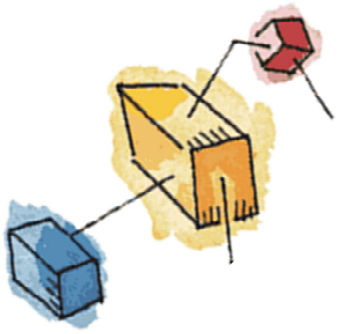




Resident Set Management

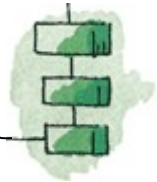
- Fixed-allocation
  - Gives a process a fixed number of frames in main memory.
  - When a page fault occurs, one of the pages of that process must be replaced.
- Variable-allocation
  - Number of frames allocated to a process varies over the lifetime of the process.
  - Give additional frames to process that is suffering persistently high levels of page faults.
  - Reduce allocation to a process with an exceptionally low page fault rate.

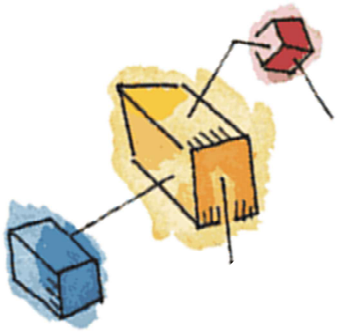




Resident Set Management

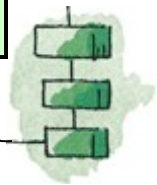
- Replacement scope
  - The scope of a replacement strategy can be categorized as global or local.
  - Both types are activated by a page fault when there are no free page frames.
- Local replacement policy
  - Chooses only among the resident pages of the process that generated the page fault.
- Global replacement policy
  - Considers all unlocked pages in main memory.

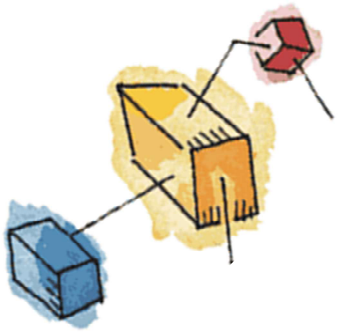




Resident Set Management

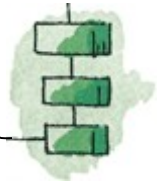
	Local Replacement	Global Replacement
Fixed Allocation	•Number of frames allocated to a process is fixed. •Page to be replaced is chosen from among the frames allocated to that process.	•Not possible.
Variable Allocation	•The number of frames allocated to a process may be changed from time to time to 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.

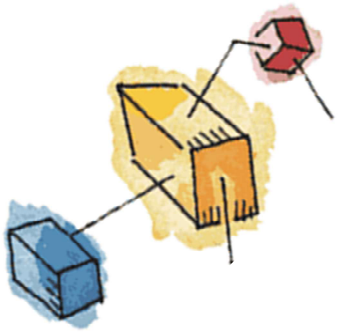




Load Control

- Determines ***the number of processes*** that will be resident in main memory.
  - *Multiprogramming* level
- Too few processes: many occasions when all processes will be blocked.
- Too many processes: inadequate resident set leads to frequent faulting and results in ***thrashing***.
  - thrashing: a state in which the system spends most of its time swapping process pieces rather than executing instructions

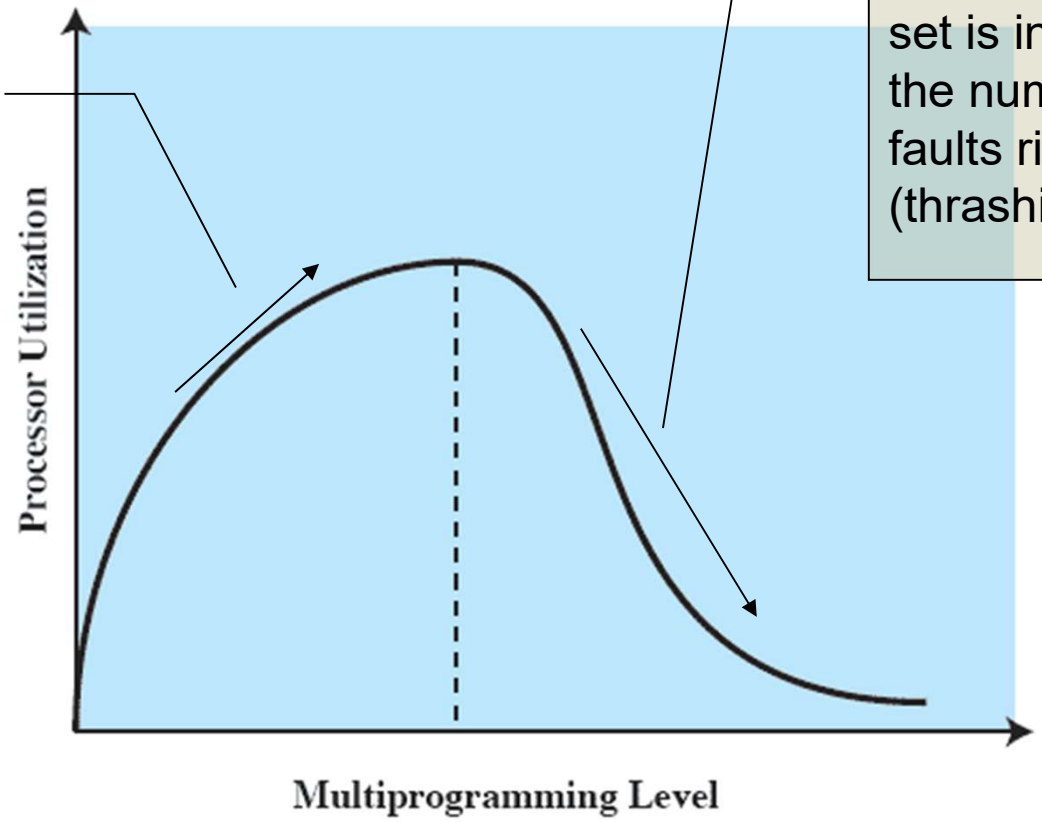


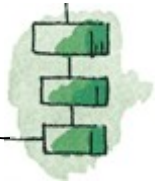


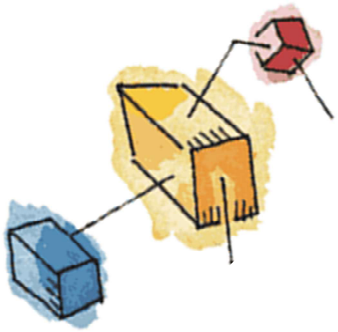
Load Control

There is less chance that all resident processes are blocked

The average resident set is inadequate and the number of page faults rises dramatically (thrashing)







Load Control

- One way to approach the problem: monitor the rate at which the pointer scans the circular buffer of frames in the clock page replacement algorithm, using a global scope.
- If the rate is below a given lower threshold, increase the multiprogramming level.
  - few page faults are occurring
  - there are many resident pages not being referenced and are readily replaceable
- If the rate exceeds an upper threshold, the multiprogramming level is too high.