Operating Systems
CS-C3140, Lecture 5

Alexandru Paler

Announcements

- Exam: 7 December 2022, 9:00
  - was 7 December 2022, then became 13 December
  - details about exact form will be announced asap
- Discord is dead, long live Zulip

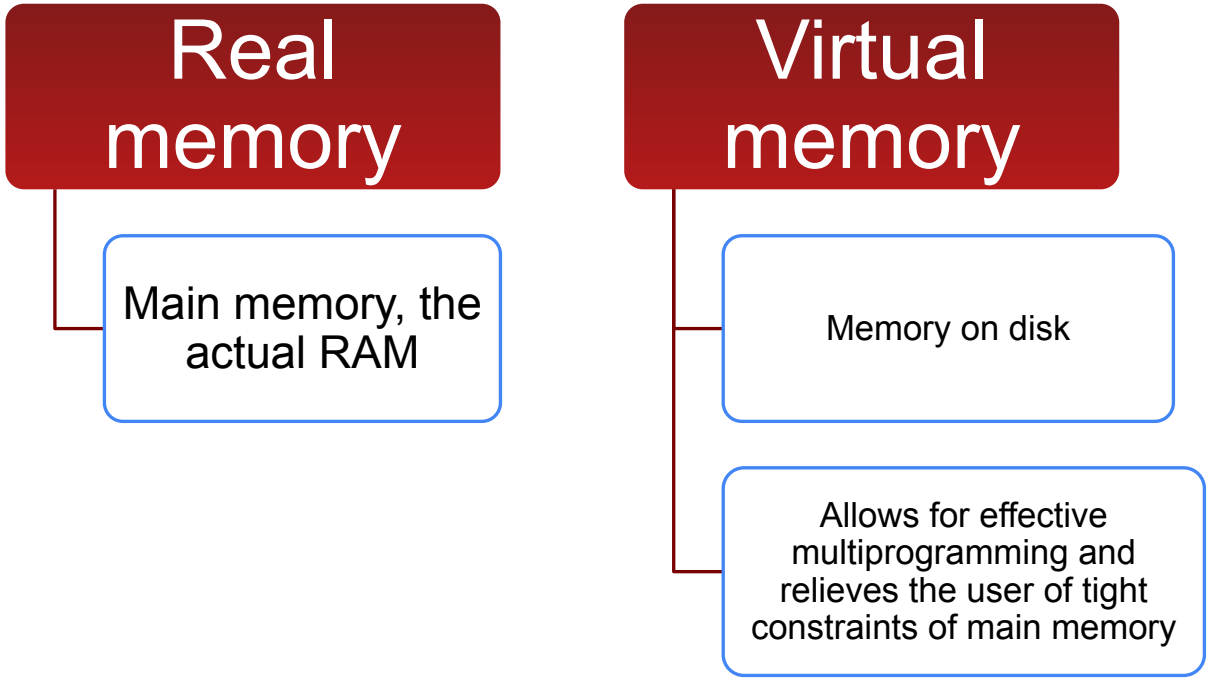
Operating System Software

The design of the memory management portion of an operating system depends on three fundamental areas of choice:

1) Whether or not to use virtual memory techniques
2) The use of paging or segmentation or both
3) Memory Management: the algorithms employed for various aspects

Frame	A fixed-length block of main memory.
Page	A fixed-length block of data that resides in secondary memory (such as disk). A page of data may temporarily be copied into a frame of main memory.
Segment	A variable-length block of data that resides in secondary memory. An entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging).

Real and Virtual Memory



Memory Management Requirements

- Memory management is intended to satisfy the following requirements:
  - Relocation
  - Protection
  - Sharing
  - Logical organization
  - Physical organization
- Two characteristics fundamental to memory management:
  - All memory references are logical addresses that are dynamically translated into physical addresses at run time
  - A process may be broken up into a number of pieces that don’t 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

Implications

- More processes may be maintained in main memory
  - Because only some of the pieces of any particular process are loaded, there is room for more processes
  - This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in a Ready state at any particular time
- A process may be larger than all of main memory
  - If the program being written is too large, the programmer must devise ways to structure the program into pieces that can be loaded separately in some sort of overlay strategy
  - With virtual memory based on paging or segmentation, that job is left to the OS and the hardware
  - The OS automatically loads pieces of a process into main memory as required

Relocation

- Programmers typically do not know in advance which other programs will be resident in main memory at the time of execution of their program
- Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization
- Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting
- May need to relocate the process to a different area of memory

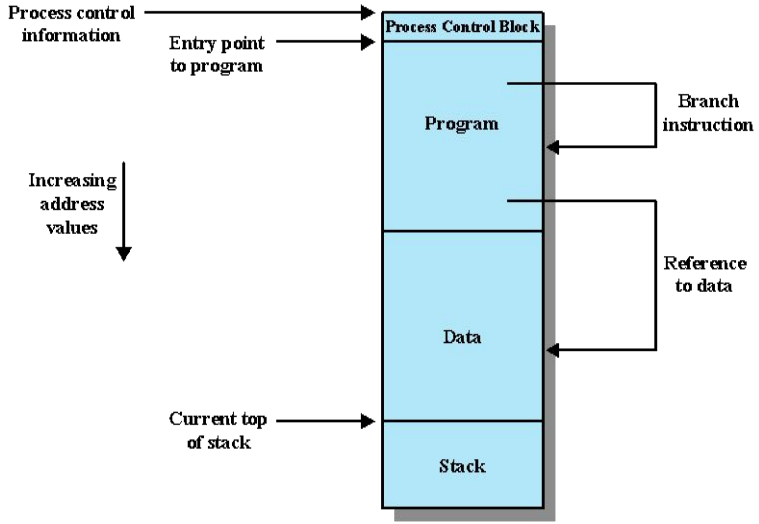


Figure 7.1 Addressing Requirements for a Process

Execution of a Process

- Resident set
  - Operating system brings into main memory a few pieces of the program
  - Portion of process that is in main memory
  - An interrupt is generated when an address is needed that is not in main memory
  - Operating system places the process in a blocking state
- Piece that contains the logical address is brought into main memory
  - operating system issues a disk I/O Read request
  - another process is dispatched to run while the disk I/O takes place
  - an interrupt is issued when disk I/O is complete,
  - place the affected process in the Ready state

Hardware Support for Relocation

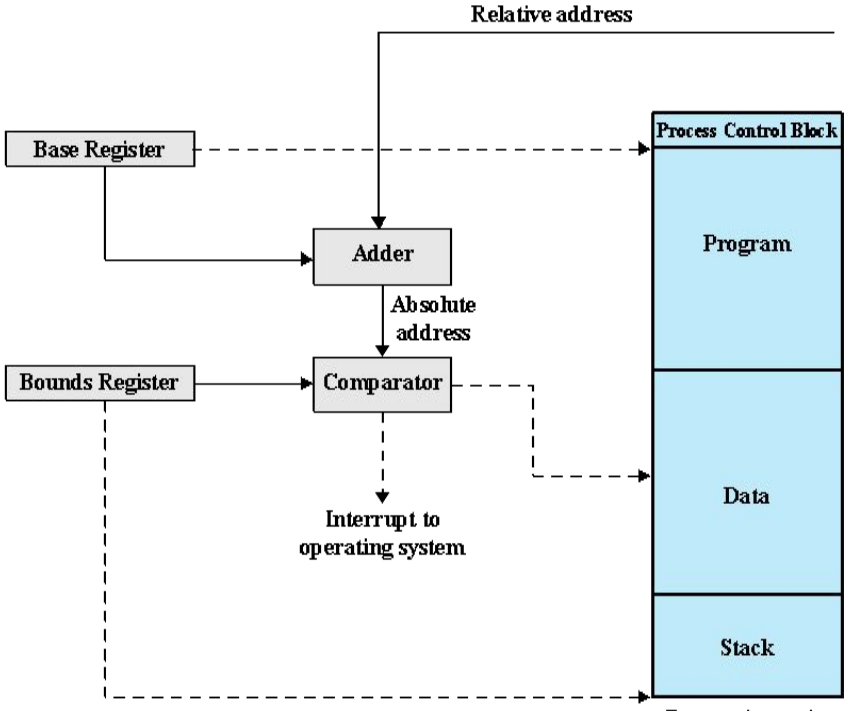
Process in Running state (loaded or swapped into memory)

- Base register indicates the starting address in main memory
- Bounds register indicates the ending location of the program

Memory references in the loaded process:

- are relative to the origin of the program
- hardware mechanism for translating relative addresses to physical main memory addresses at the time of execution of the instruction that contains the reference

Process is isolated by the contents of the base and bounds registers and safe from unwanted accesses by other processes.



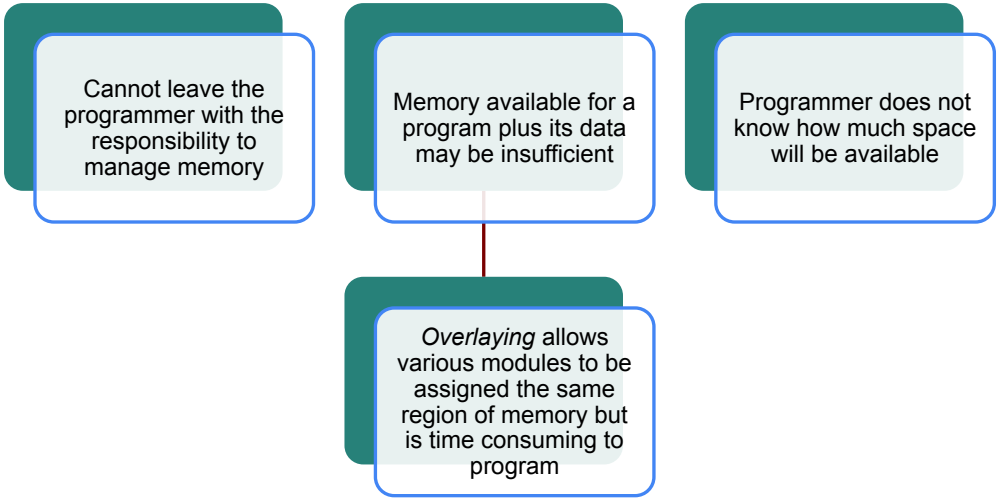
Protection and Sharing

- Processes need to acquire permission to reference memory locations for reading or writing purposes
- 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

- Advantageous to allow each process access to the same copy of the program rather than have their own separate copy
- Memory management must allow controlled access to shared areas of memory without compromising protection
- Mechanisms used to support relocation support sharing capabilities

Logical and Physical Organization

- Memory is organized as linear
  - sequence of bytes or words
  - mirrors the actual machine hardware
  - does not correspond to the way in which programs are typically constructed
- Programs are written in modules
  - written and compiled independently
  - Different degrees of protection given to modules (read-only, execute-only)
- Sharing on a module level corresponds to the user's way of viewing the problem



Memory Management Techniques

Technique	Description	Strengths	Weaknesses
Fixed Partitioning	Main memory is divided into a number of static partitions at system generation time. A process may be loaded into a partition of equal or greater size.	Simple to implement; little operating system overhead.	Inefficient use of memory due to internal fragmentation; maximum number of active processes is fixed.
Dynamic Partitioning	Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.	No internal fragmentation; more efficient use of main memory.	Inefficient use of processor due to the need for compaction to counter external fragmentation.
Simple Paging	Main memory is divided into a number of equal-size frames. Each process is divided into a number of equal-size pages of the same length as frames. A process is loaded by loading all of its pages into available, not necessarily contiguous, frames.	No external fragmentation.	A small amount of internal fragmentation.
Simple Segmentation	Each process is divided into a number of segments. A process is loaded by loading all of its segments into dynamic partitions that need not be contiguous.	No internal fragmentation; improved memory utilization and reduced overhead compared to dynamic partitioning.	External fragmentation
Virtual Memory Paging	As with simple paging, except that it is not necessary to load all of the pages of a process. Nonresident pages that are needed are brought in later automatically.	No external fragmentation; higher degree of multiprogramming; large virtual address space.	Overhead of complex memory management.
Virtual Memory Segmentation	As with simple segmentation, except that it is not necessary to load all of the segments of a process. Nonresident segments that are needed are brought in later automatically.	No internal fragmentation, higher degree of multiprogramming; large virtual address space; protection and sharing support.	Overhead of complex memory management.

Addresses

Logical

- Reference to a memory location independent of the current assignment of data to memory

Relative

- A particular example of logical address, in which the address is expressed as a location relative to some known point

Physical or Absolute

- Actual location in main memory

Paging

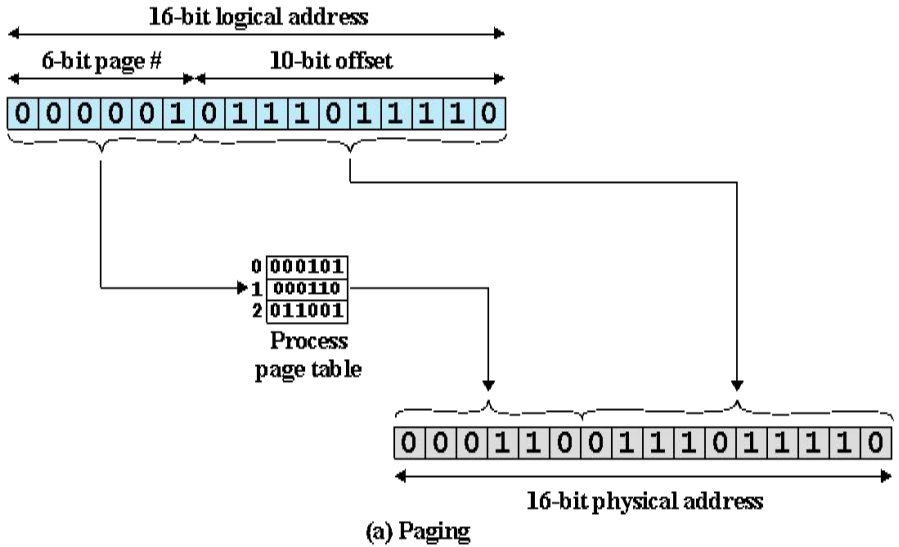
- Partition memory into equal fixed-size chunks that are relatively small
- Process is also divided into small fixed-size chunks of the same size
- Each process has its own page table
  - Each page table entry (PTE) contains the frame number of the corresponding page in main memory
  - A **page table** is also needed for a virtual memory scheme based on paging

Pages	Frames
• Chunks of a process	• Available chunks of memory

Figure 7.9 Assignment of Process Pages to Free Frames

Page Table

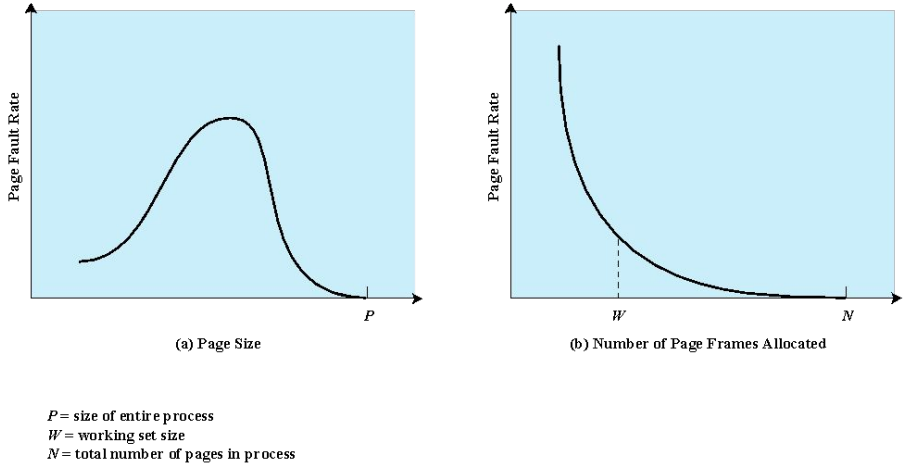
- Maintained by operating system for each process
- Contains the frame location for each page in the process
- Processor must know how to access for the current process
- Used by processor to produce a physical address



(a) Paging

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 some portion of the page tables of active processes must be in virtual memory instead of main memory
  - The physical characteristics of most secondary-memory devices favor a larger page size for more efficient block transfer of data



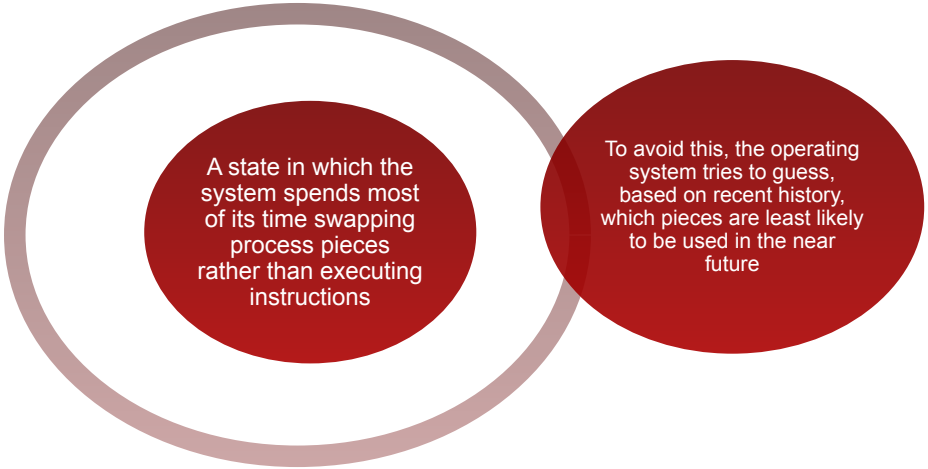
The design issue of page size is related to the size of physical main memory and program size

Main memory is getting larger and address space used by applications is also growing

Most obvious on personal computers where applications are becoming increasingly complex

Principle of Locality and Thrashing

- Program and data references within a process **tend to cluster**
- Only a few pieces of a process will be needed over a short period of time
- Therefore it is possible to make intelligent guesses about which pieces will be needed in the future
- Avoids thrashing



Segmentation: Protection and Sharing

- 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

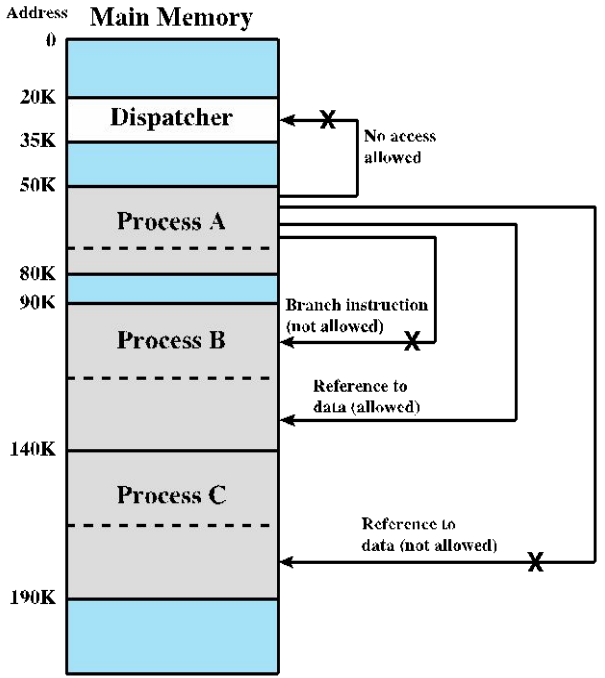


Figure 8.13 Protection Relationships Between Segments

Segment Organization

- Each segment table entry contains the starting address of the corresponding segment in main memory and the length of the segment
- A bit is needed to determine if the segment is already in main memory
- Another bit is needed to determine if the segment has been modified since it was loaded in main memory

Logical Addresses

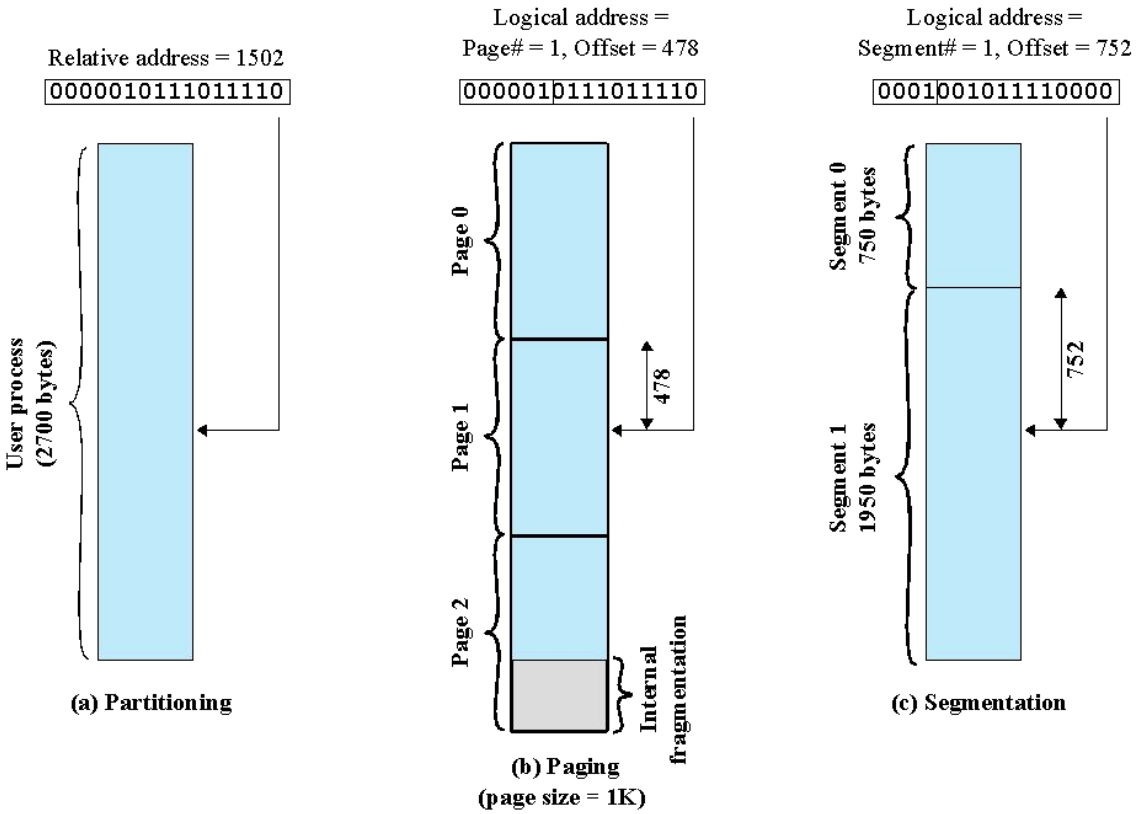


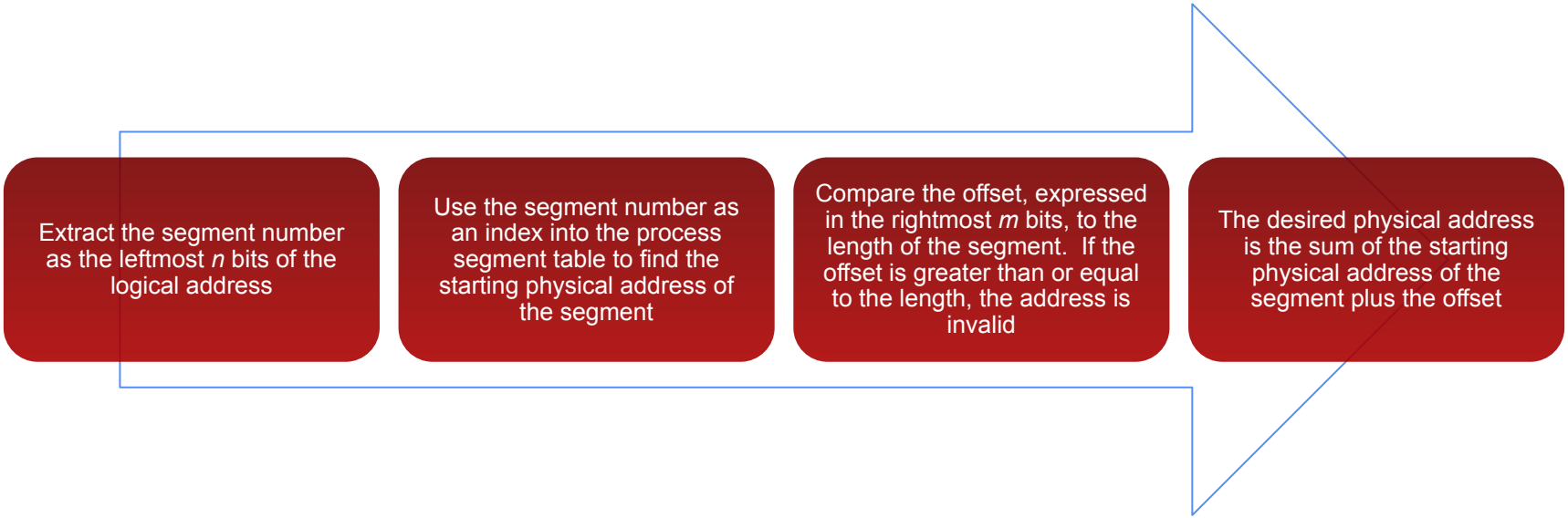
Figure 7.11 Logical Addresses

Segmentation

- A program can be subdivided into segments
  - May vary in length
  - There is a maximum length
- Addressing consists of two parts (similar to paging):
  - Segment number
  - An offset
- Eliminates internal fragmentation
- Typically the programmer will assign programs and data to different segments
  - modularity: the program or data may be further broken down into multiple segments
  - inconvenience: the programmer must be aware of the maximum segment size limitation
- A segment table entry contains
  - the starting address of the corresponding segment in main memory
  - the length of the segment
  - bit to determine if the segment is already in main memory
  - Another bit to determine if the segment has been modified

Address Translation - Steps

- Another consequence of unequal size segments is that there is no simple relationship between logical addresses and physical addresses



Combined Paging and Segmentation

In a combined paging/segmentation system a user's address space is broken up into a number of segments. Each segment is broken up into a number of fixed-sized pages which are equal in length to a main memory frame

- Segmentation is visible to the programmer
- Paging is transparent to the programmer

Virtual Memory

Virtual Memory Terminology

Virtual memory	A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations.
Virtual address	The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.
Virtual address space	The virtual storage assigned to a process.
Address space	The range of memory addresses available to a process.
Real address	The address of a storage location in main memory.

Characteristics of 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 not partitioned	
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)	
Internal fragmentation within frames		No internal fragmentation	
No 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 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 list of free holes in main memory	
Processor uses page 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

Hardware Support and Issues

- Hardware must support paging and segmentation
- Operating system must include software for managing the movement of pages and/or segments between secondary memory and main memory

Issues to Address

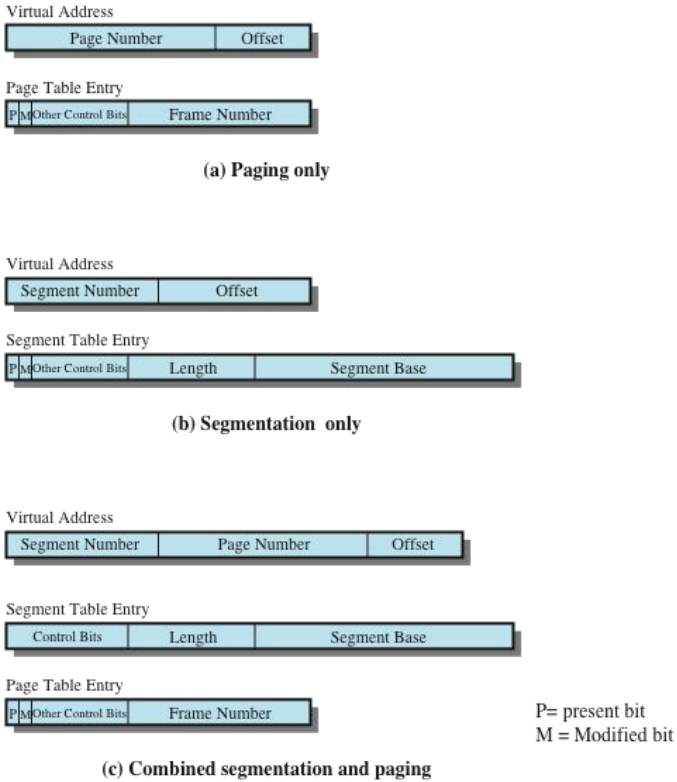
• Size of page-tables would be very large!

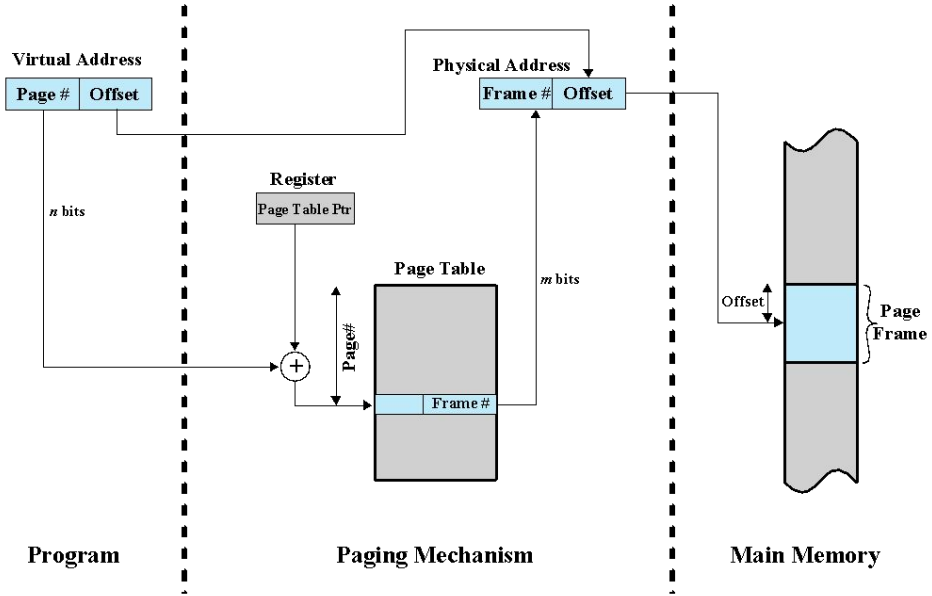
• For example, 32-bit virtual address spaces (4 GB) and a 4 KB page size would have ~1 M pages/entries in page-tables.

• A process does not access all of its address space at once!

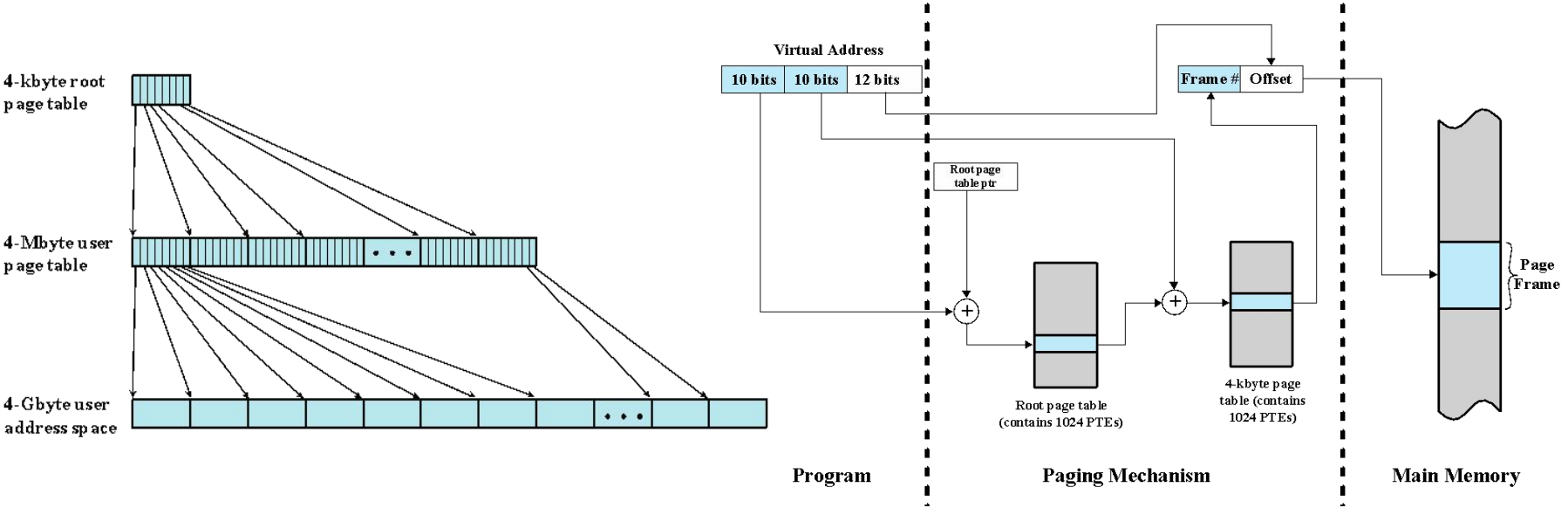
Exploit this locality factor.

Memory Management Formats



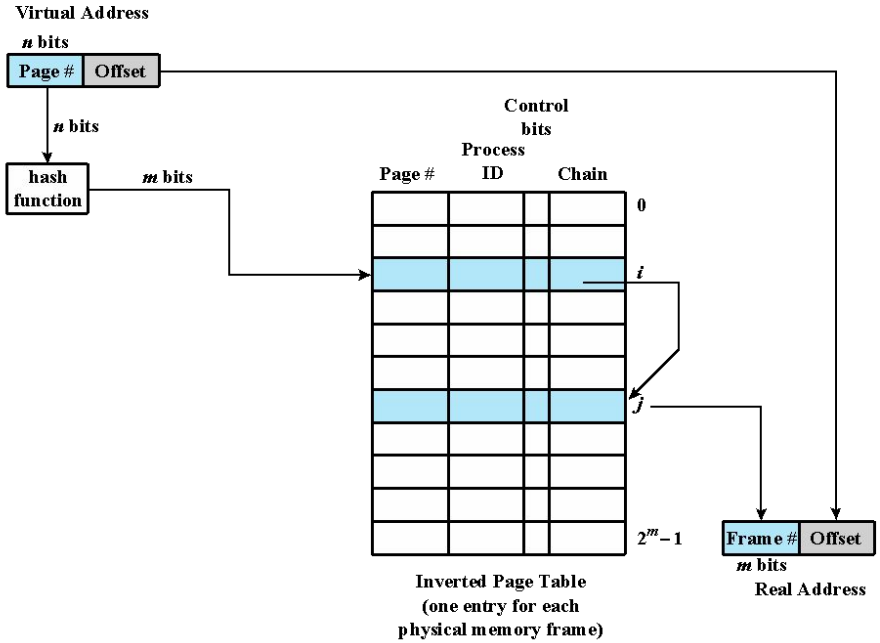


Two-Level Hierarchical Page Table



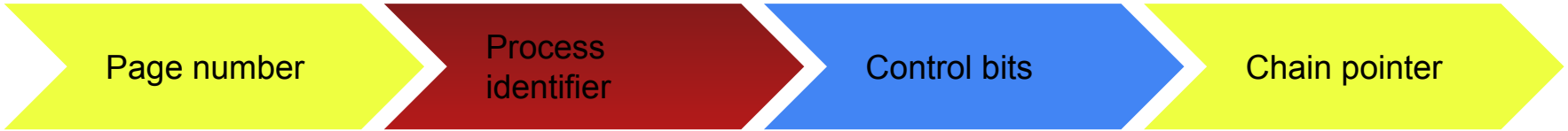
Inverted Page Table

- Used on the PowerPC, UltraSPARC, and the IA-64 architecture
- Page number portion of a virtual address is mapped into a hash value
- Hash value points to inverted page table
- Fixed proportion of real memory is required for the tables regardless of the number of processes or virtual pages supported
- Structure is called inverted because it indexes page table entries by frame number rather than by virtual page number



Inverted Page Table

Each entry in the page table includes:



This is the page number portion of the virtual address

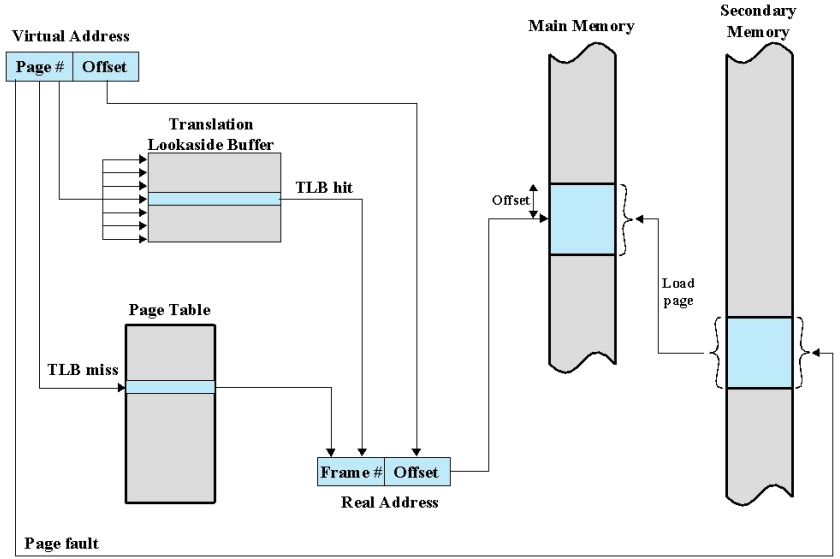
The process that owns this page

Includes flags and protection and locking information

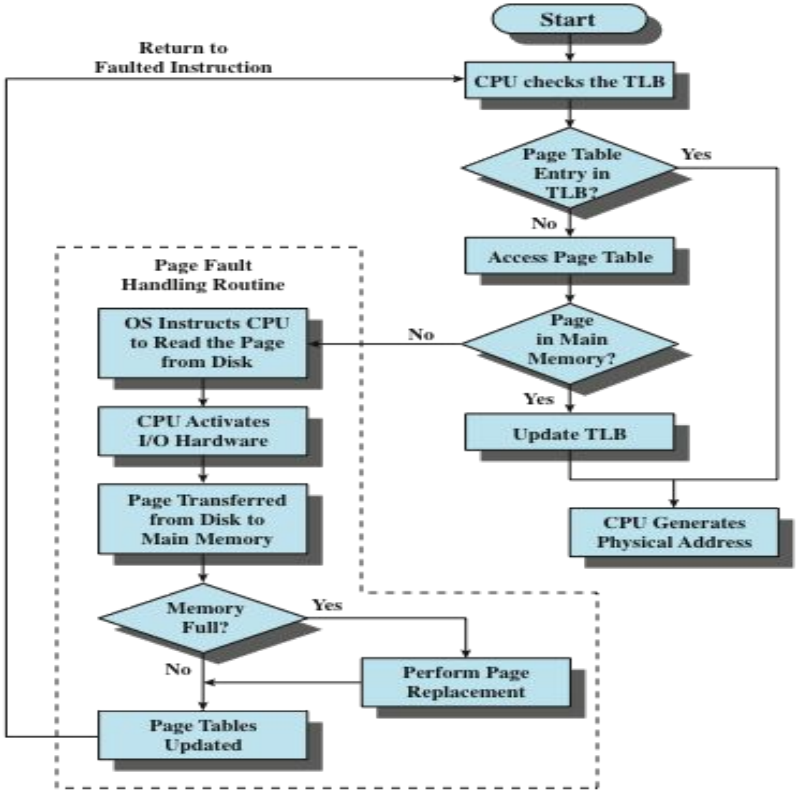
The index value of the next entry in the chain

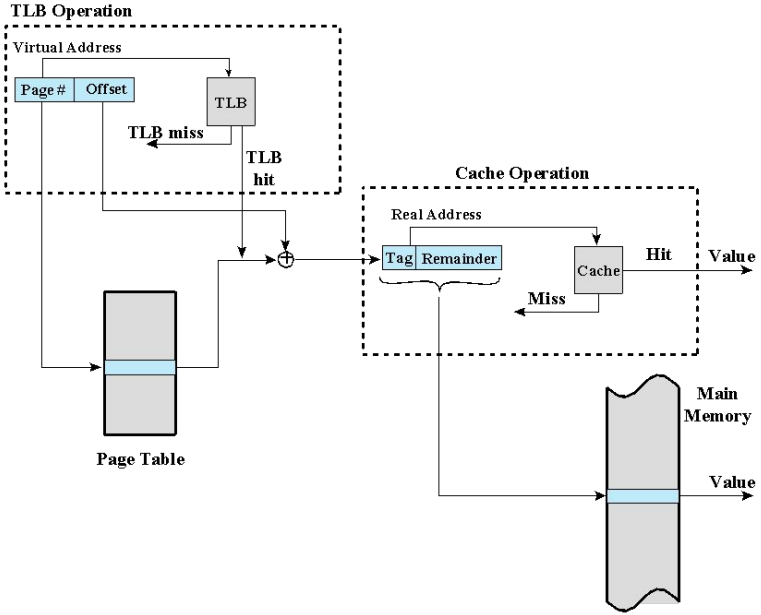
Translation Lookaside Buffer (TLB)

- To overcome the effect of doubling the memory access time, most virtual memory schemes make use of a special high-speed cache called a translation lookaside buffer (TLB)
- Each virtual memory reference can cause two physical memory accesses:
  - One to fetch the page table entry
  - One to fetch the data
- This cache functions in the same way as a memory cache and contains those page table entities that have been most recently used



Operation of Paging, TLB and Cache





Associative Mapping

- The TLB only contains some of the page table entries so we cannot simply index into the TLB based on page number
  - Each TLB entry must include the page number as well as the complete page table entry
- The processor is equipped with hardware that allows it to interrogate simultaneously a number of TLB entries to determine if there is a match on page number

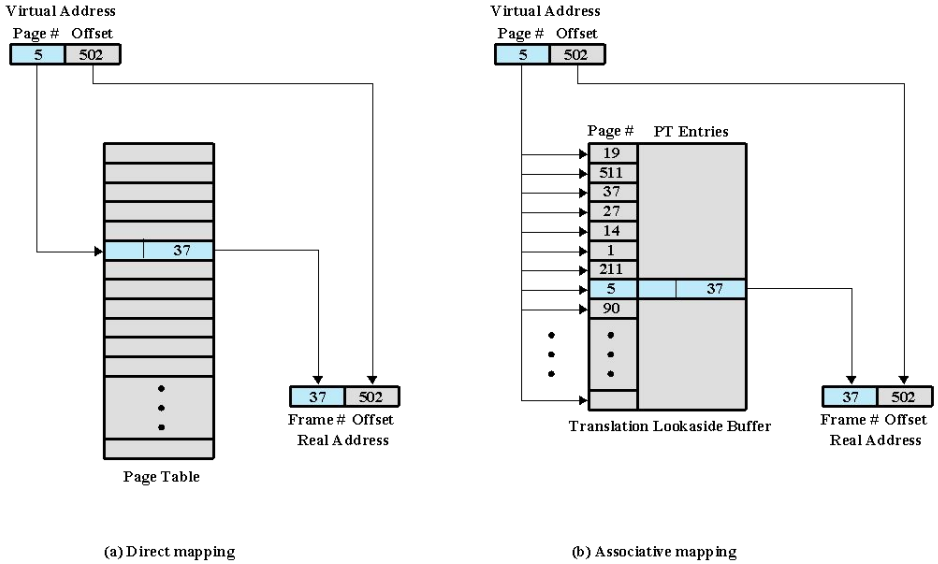


Figure 8.8 Direct Versus Associative Lookup for Page Table Entries

Warm-ups and cold starts

- A CPU runs fast when its caches and TLB are “warm”
  - Being “warm” means that the CPU has the cache and TLB contents that fit the memory reference patterns of its program
  - Warming up can take several milliseconds even for fast systems
- “Cold” when cache and TLB contents are unsuitable
  - Typically after a context switch (e.g., to a different thread)
- Note that there are several levels of coldness
  - A cold cache, a cold TLB, a cold memory
    - A cold memory is really costly, it is typical for process starts or after a process swap (everything must be fetched from secondary mem)
- The misses and faults are named accordingly
  - A cold cache miss, a cold TLB miss, a cold page fault
  - Note that the choice of replacement algorithms has no effect

Management of Pages

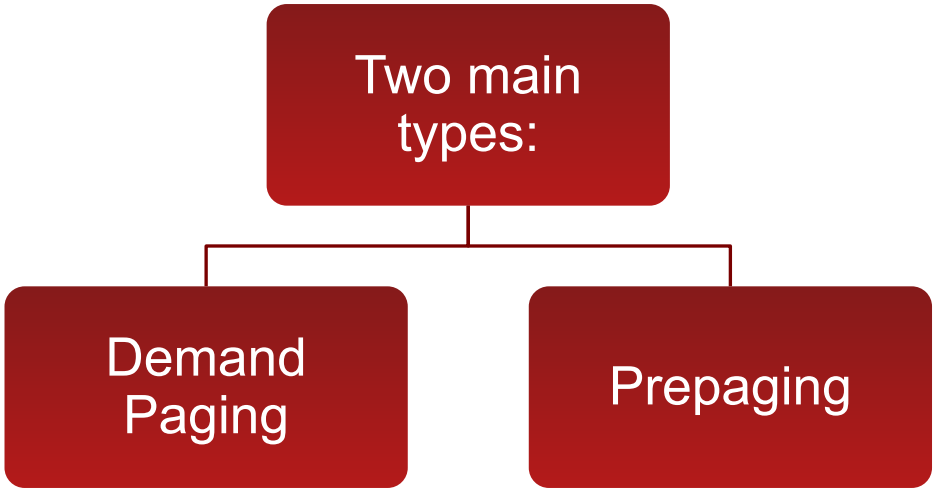
Process working sets and swapping

- Maintaining working sets in main memory
  - A program usually access data in the same area
    - Loops (etc.) in code iterating through data structures
  - Only currently used parts of a process need to be in memory
    - The OS tries to figure out the size of the working sets of processes and balance between them
- Processes can be swapped out
  - All the pages of a process can be in the secondary memory
  - Without swapping processes out, there can be too much competition over memory => performance degradation
  - Note: there are different meaning for the word “swapping”, e.g., process swapping and page swapping

- **Resident set management**
  - How many pages of a process to keep in memory?
  - Small resident set size
  - Larger number of processes in memory
- Scheduler can find ready processes to run
  - There is not enough of the process in the memory
  - Lots of page faults
- Large resident set size
  - Due to the **principle of locality**, there is a limit after which increasing resident set size does not help
- There are several policies for this also
  - Fixed allocation policy, variable allocation policy, etc.

Fetch policy

- When page should be brought to memory
  - Note that accessing main memory is typically much faster than accessing secondary memory (e.g. HDD/SDD)
- Demand paging
  - Bring the page once it is referenced
  - When a process is started, there is a bunch of page faults
  - After a while, page fault rate should drop to a low level
- Prepaging
  - Bring pages even if they are not referred to
  - Once we have to go to the disk, makes sense to read more than one block if data is suitably located on the disk



Replacement policy

- Which page should be kicked out when a new page has been loaded
- Principle of locality
  - Does not make sense to throw out recently referenced pages
- Efficiency
  - The more complex the replacement policy the more overhead is caused
- Frame locking
  - Some frames of memory need to be locked
  - E.g., some I/O devices and parts of kernel code and data need to be placed in fixed physical addresses

Replacement algorithms

- Optimal
  - Replace the page for which the time to next reference is the longest
  - impossible to implement but can be calculated by recording the references and analyzing them afterwards
- First In First Out (FIFO)
  - Replaces the page that has been longest in memory
  - SW implementation is easy: maintain a queue buffer
  - may throw out busy pages ⇒ often not even near the optimal
- Least Recently Used (LRU)
  - Replace the page that has not been referenced the longest time
  - Tag the references or maintain a list
  - Hardware acceleration would be needed, but is complicated
    - accesses are frequent, replacements are (typically) rare

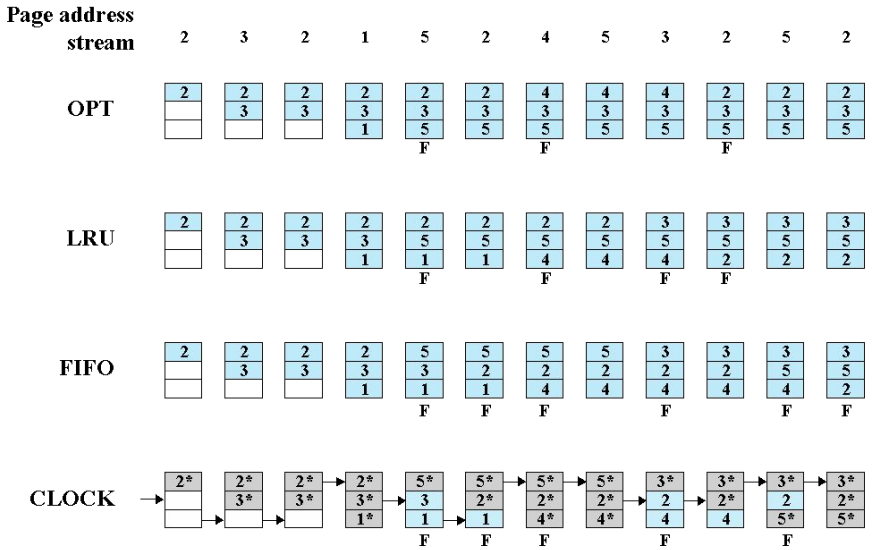


Figure 8.14 Behavior of Four Page-Replacement Algorithms

Clock algorithm

- Referenced bit
  - Use a bit per each frame to indicate if it is referenced
  - this can be done by MMU hardware (e.g., in TLB)
- Replacement
  - Frames are considered to be a circular buffer
  - If any of the frames have use bit zero ⇒ choose it for replacement
  - Each time use bit 1 is encountered ⇒ set it to zero
  - If all bits are 1 then a complete cycle has been done, and the first frame will be replaced
- Can be visualized as a clock diagram
  - Thus, we have the name, but there is no real clock!
- Can be improved by checking the dirty bit
  - This way algorithm prefers replacing pages that are clean

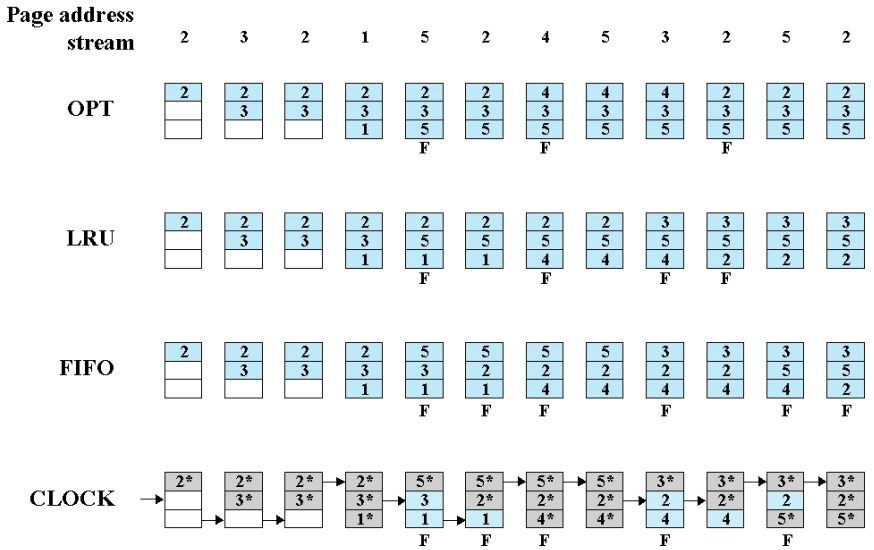


Figure 8.14 Behavior of Four Page-Replacement Algorithms

Cleaning policy

- Writing dirty pages to the secondary memory
  - Which pages? When?
- Demand cleaning
  - Write the page only if it is chosen for replacement
  - Page is only written if a new page is brought to memory
- Precleaning
  - Write modifies pages before the frames are needed
  - Allows bulk writing of page (but deciding which is hard…)
- Page buffering
  - A pool of free frames is maintained
  - When a page fault occurs, the desired page is read into a free frame from the pool. A victim frame is later swapped out if necessary and put into the free frames pool.