

Virtual Memory



Chapter 8

Contents



- ✍ Hardware and control structures
- ✍ Operating system software
- ✍ Unix and Solaris memory management
- ✍ Linux memory management
- ✍ Windows 2000 memory management

Characteristics of simple Paging and Segmentation

- ✍ Memory references are dynamically translated into physical addresses at run time
 - ✍ a process may be swapped in and out of main memory such that it occupies different regions
- ✍ A process may be broken up into pieces that do not need to be located contiguously in main memory
- ✍ Is it necessary that all the pages of a process be in main memory during execution?

Execution of a Program



- ✍ Operating system brings into main memory a few pieces of the program
 - ✍ resident set - 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

Execution of a Program



- ✍ Piece of process 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 complete which causes the operating system to place the affected process in the Ready state

Advantages of This Maneuver



- ✍ More processes may be maintained in main memory
 - ✍ only load in some of the pieces of each process
 - ✍ with so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- ✍ It is possible for a process to be larger than all the main memory
 - ✍ programmer is dealing with memory size of the hard disk

Advantages of This Maneuver



- ✍ It would be wasteful to load in many pieces of the process when only a few pieces will be used
- ✍ Time can be saved because unused pieces are not swapped in and out of memory

Types of Memory



- ✍ Real memory

 - ✍ main memory

- ✍ Virtual memory

 - ✍ memory on disk

 - ✍ allows for effective multiprogramming and relieves the user of tight constraints of main memory

Thrashing




- ✂ Swapping out a piece of a process just before that piece is needed
- ✂ The processor spends most of its time swapping pieces rather than executing user instructions

Principle of Locality



- ✍ 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
- ✍ Possible to make intelligent guesses about which pieces will be needed in the future
- ✍ This suggests that virtual memory may work efficiently

Support Needed for Virtual Memory



- ✍ Hardware must support paging and segmentation
- ✍ OS must be able to manage the movement of pages and/or segments between secondary memory and main memory

VM Paging



- ✍ Each process has its own page table
- ✍ Each page table entry contains the frame number of the corresponding page in main memory
- ✍ A bit is needed to indicate whether the page is in main memory or not

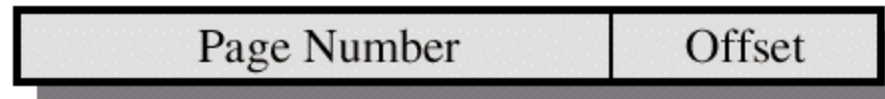
Modify Bit in Page Table



- ✍ A bit is needed to indicate if the page has been altered since it was last loaded into main memory
- ✍ If no change has been made, the page does not have to be written to the disk when it needs to be swapped out

Page Table Entries

Virtual Address



Page Table Entry



(a) Paging only

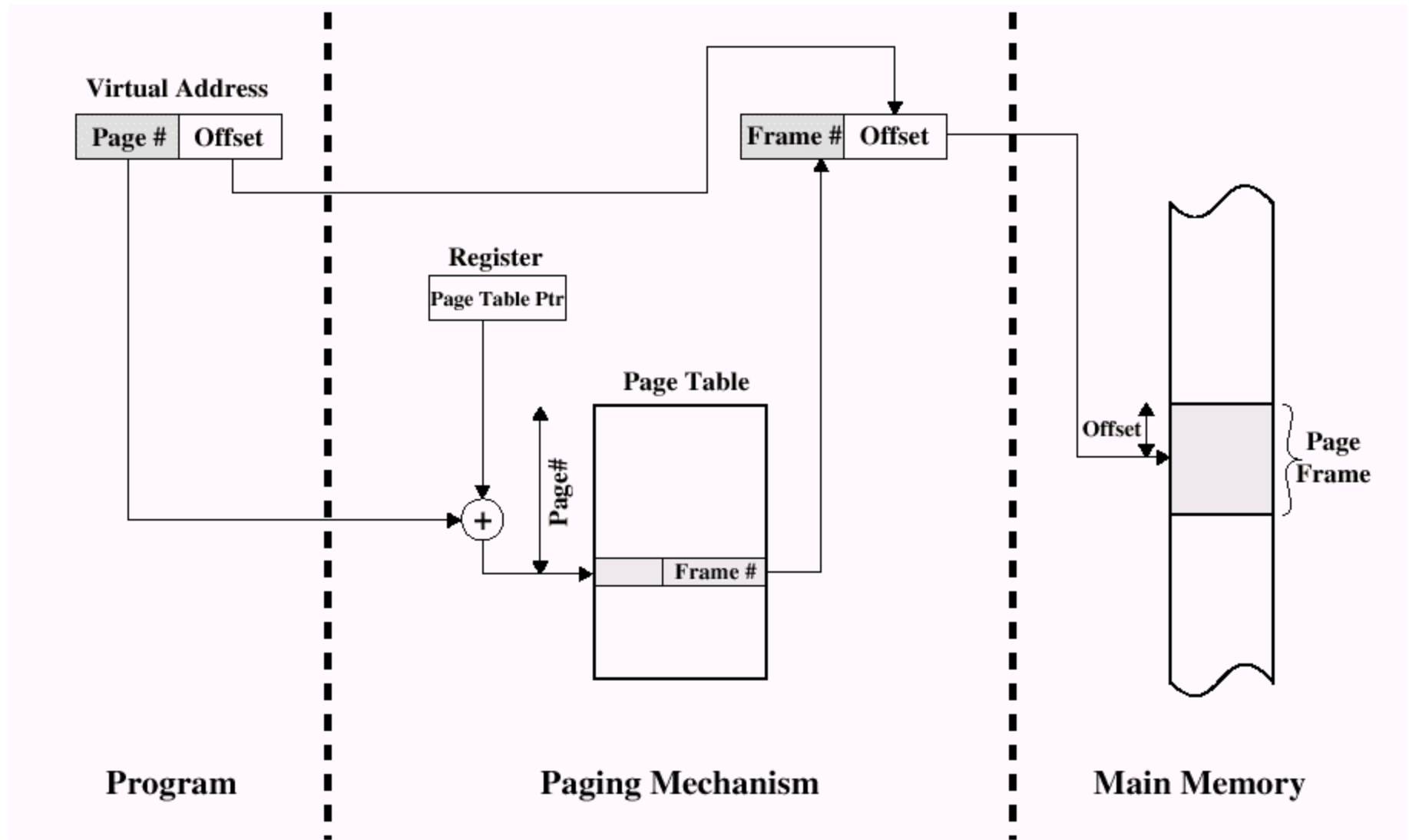


Figure 8.3 Address translation in a paging system

Page Table Structure



- ✍ The entire page table may take up too much main memory
 - ✍ in VAX, each process can have up to 2 GB of memory
 - ✍ if page size is 512 byte, we need 2^{22} page table entries
- ✍ Page tables can also be stored in virtual memory
- ✍ When a process is running, part of its page table is in main memory

Page Table Structure

✍ Two-level scheme

- ✍ assume 32-bit address, 4 KB pages and 4 GB address space

 - ✍ 2^{20} pages, requiring 4 MB

- ✍ root page table with 2^{10} PTEs occupying a page(4 KB)

 - ✍ always remains in main memory

- ✍ user page table is kept in virtual memory

 - ✍ they are mapped by a root page table

Two-Level Scheme for 32-bit Address

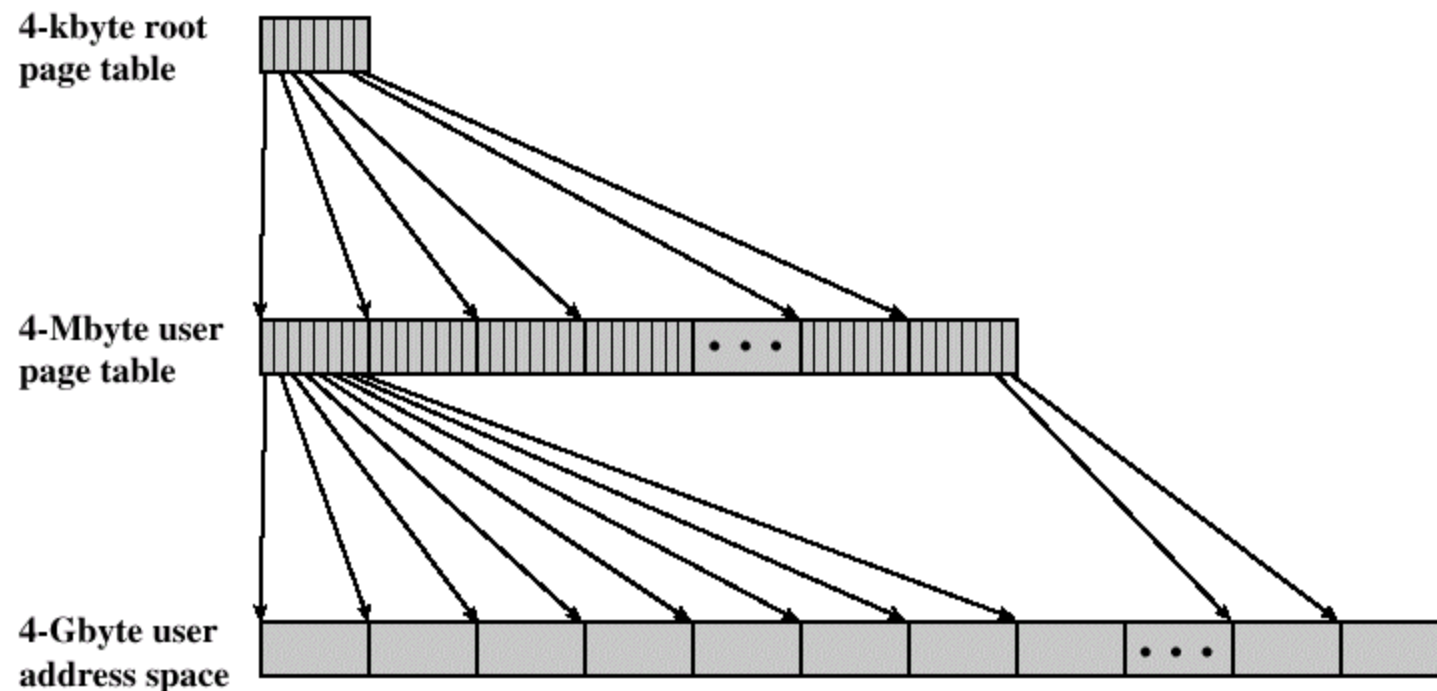


Figure 8.4 A Two-Level Hierarchical Page Table [JACO98a]

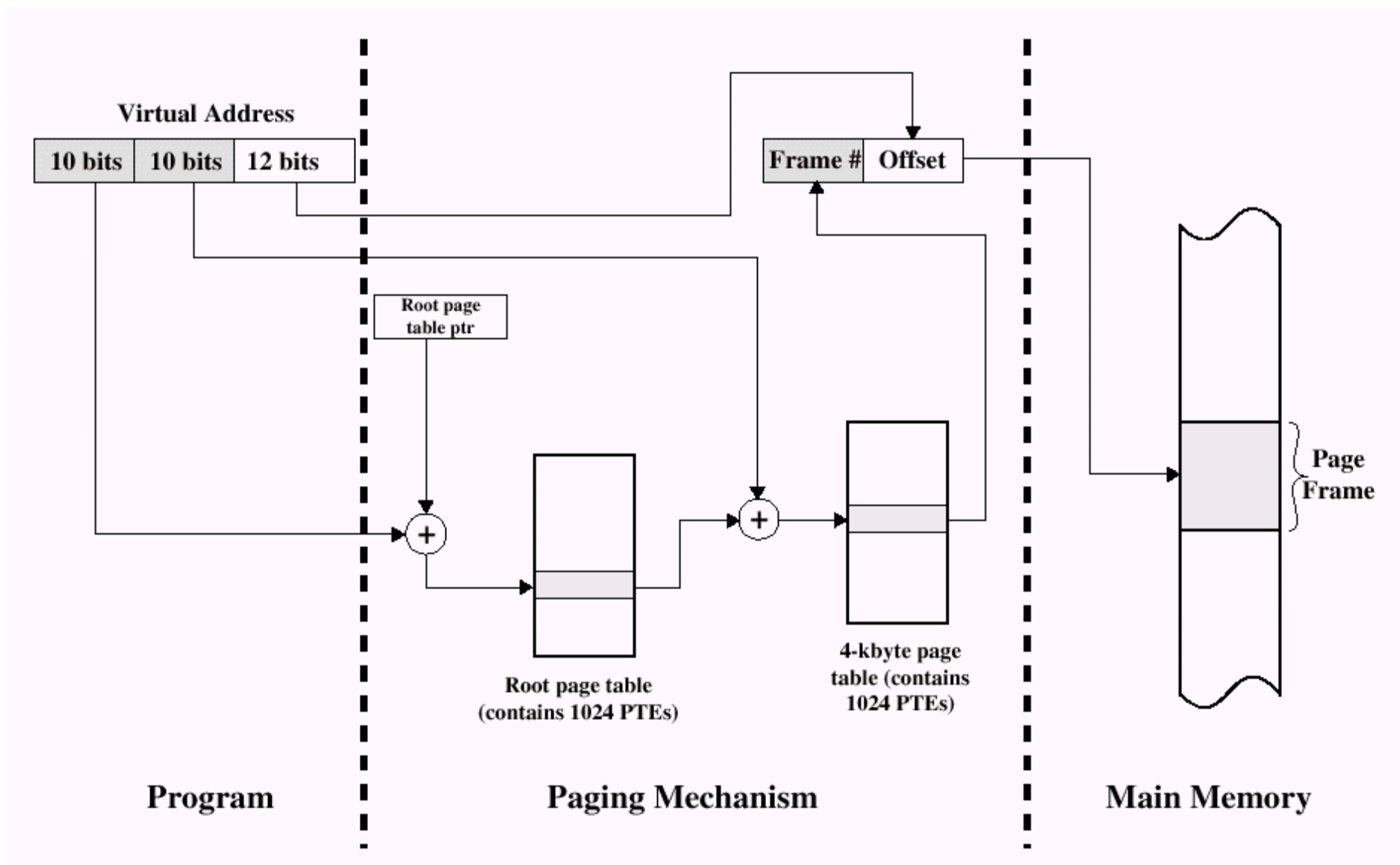








Figure 8.5 Address translation in a two-level paging system

Page Table Structure



Inverted page table structure

-  used on Power PC and on IBM's AS/400
-  page number portion of a virtual address is mapped into a hash table
-  hash table contains a pointer to the inverted page table
 -  there is one entry in the hash table and inverted page table for each real memory page
 -  fixed proportion of real memory is required for the tables
-  faster access to the page is possible

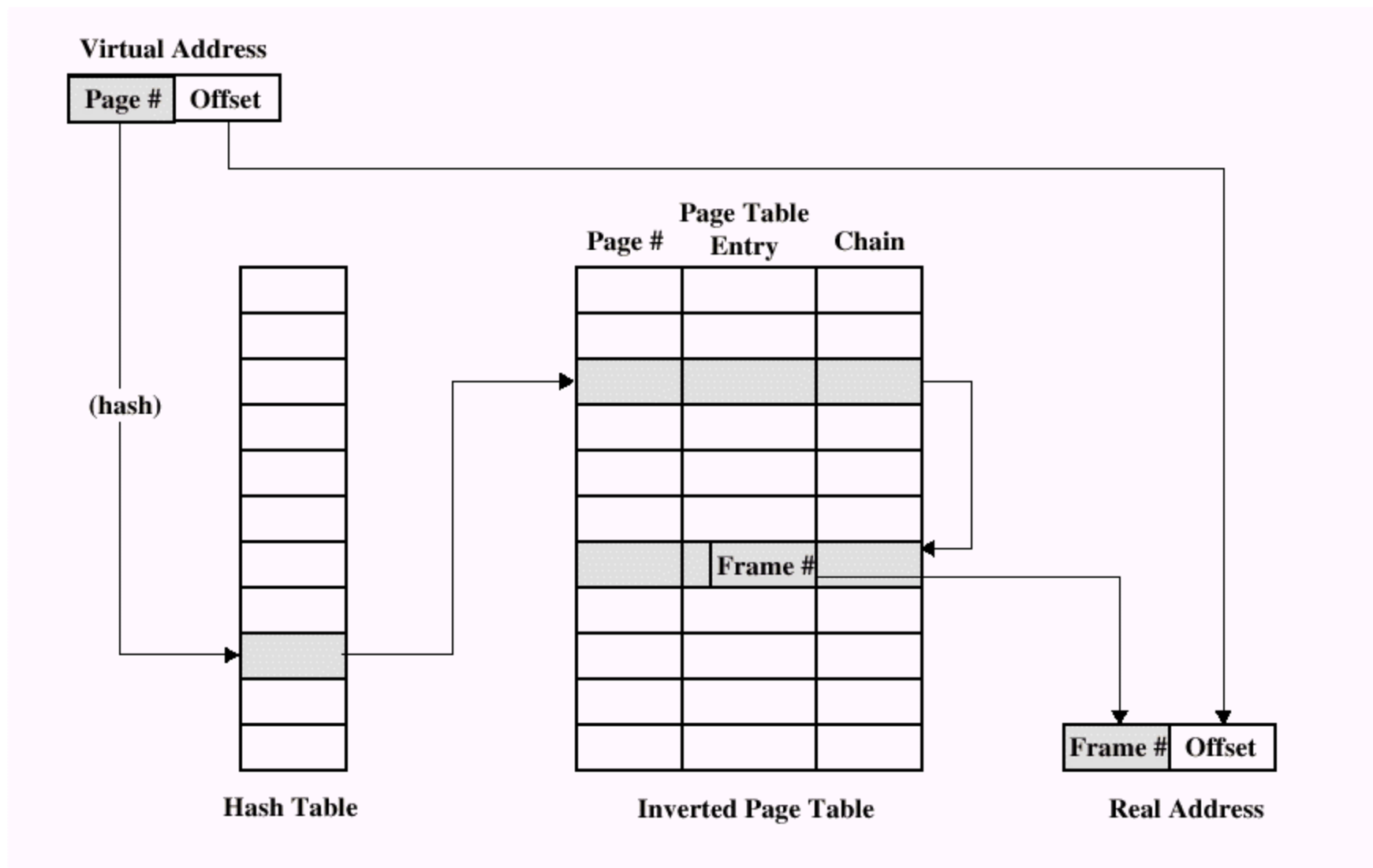



Figure 8.6 Inverted page table structure

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
- ✍ To overcome this problem a special cache is set up for page table entries
 - ✍ called a TLB - Translation Lookaside Buffer

TLB



- ✍ Contains page table entries that have been most recently used
- ✍ Works similar to a memory cache

TLB(Operations)



- ✍ Given a virtual address, processor examines the TLB
- ✍ If page table entry is present (a hit), the frame number is retrieved and the real address is formed
- ✍ If page table entry is not found in the TLB (a miss), the page number is used to index the process page table

TLB(Operations)



- ✍ Check if page is already in main memory
 - ✍ if not in main memory a page fault is issued
- ✍ TLB is updated to include the new page entry

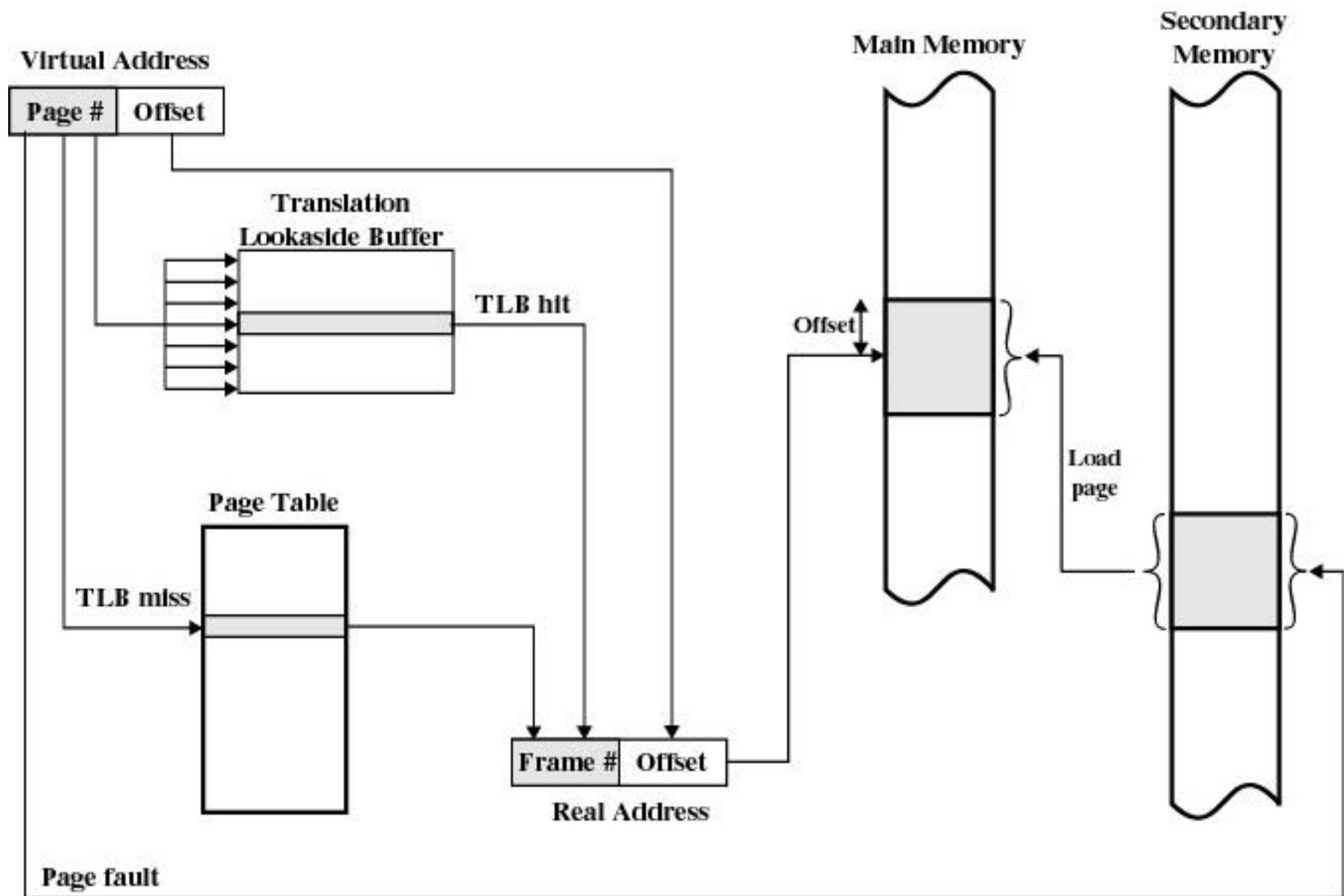


Figure 8.7 Use of a Translation Lookaside Buffer

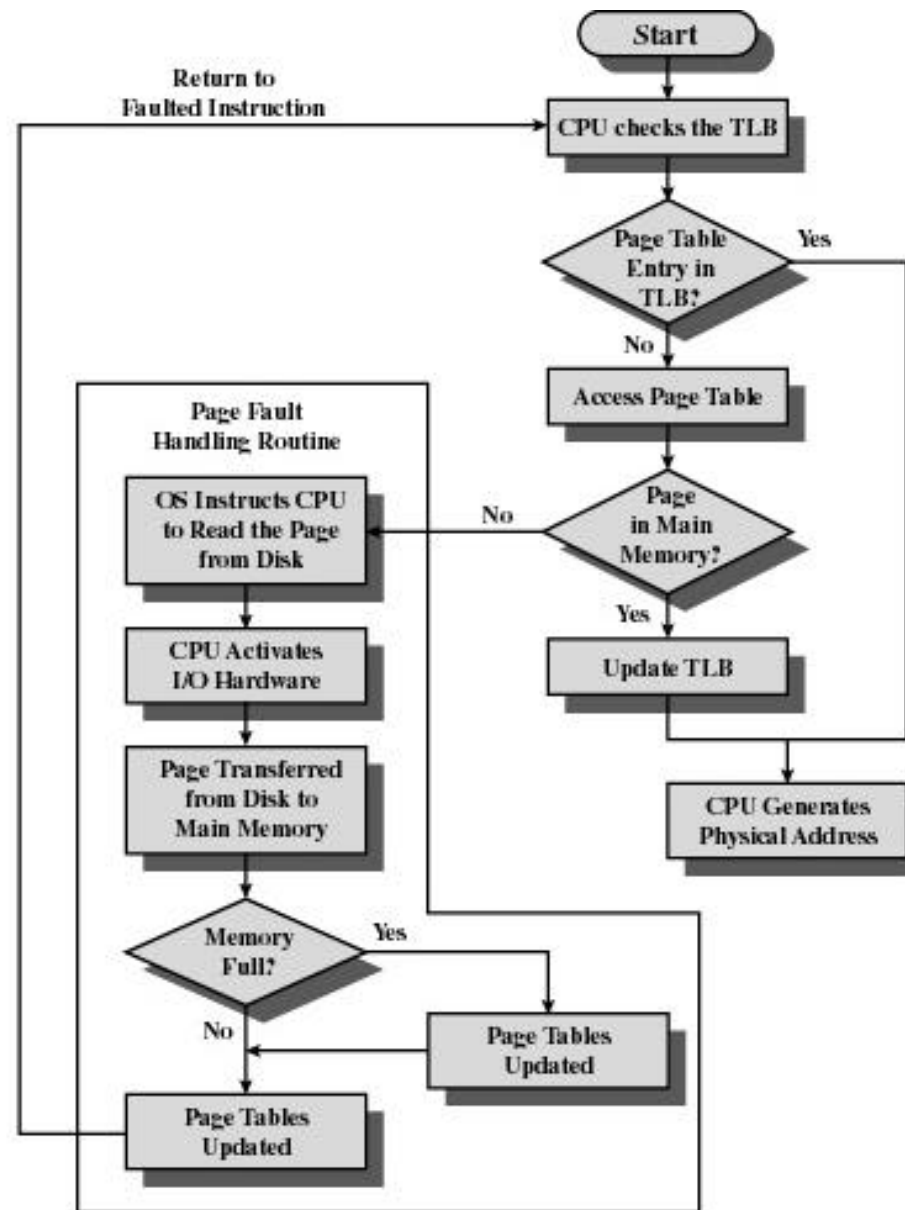


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

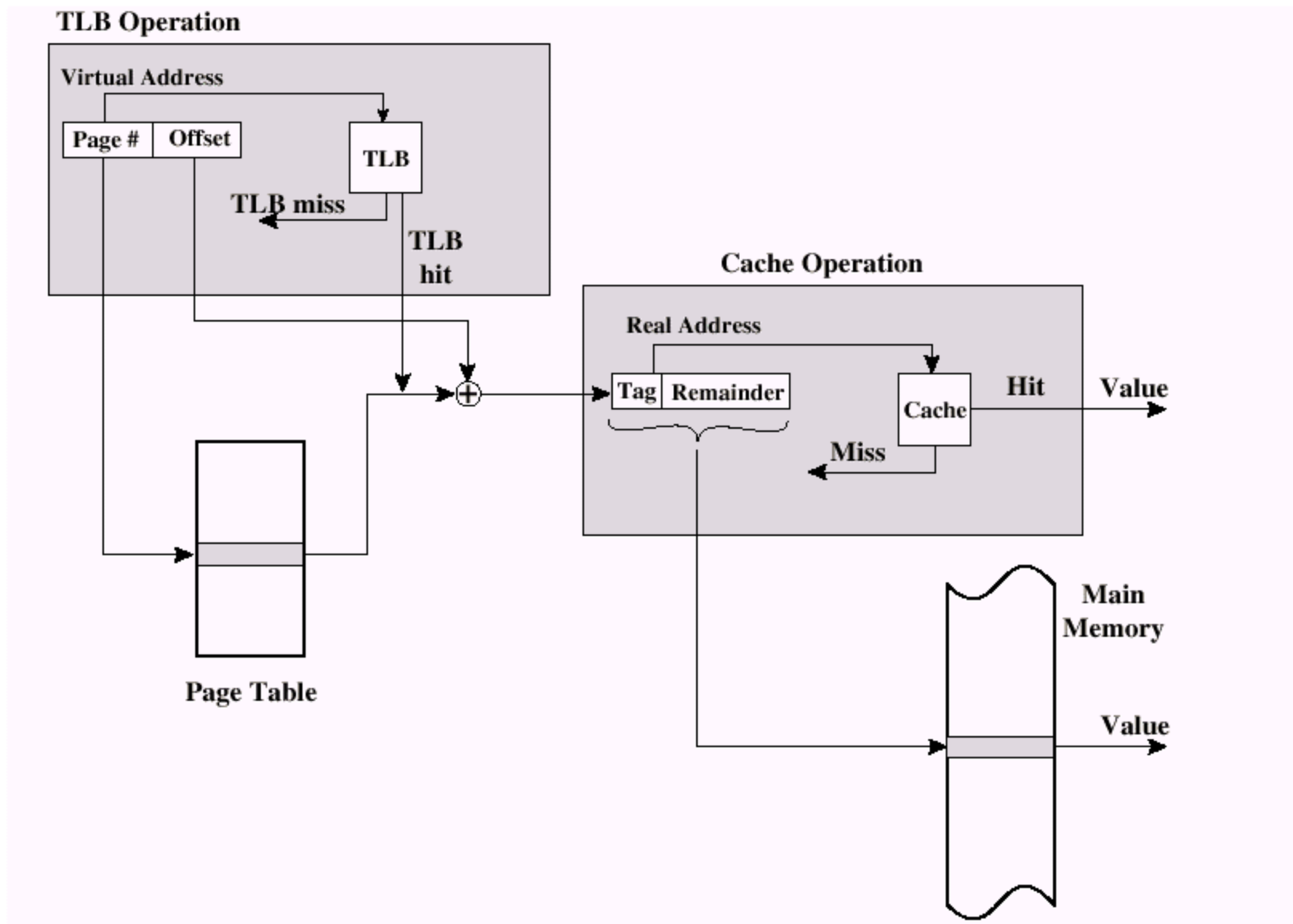


Figure 8.10 TLB and cache operation

Page Size

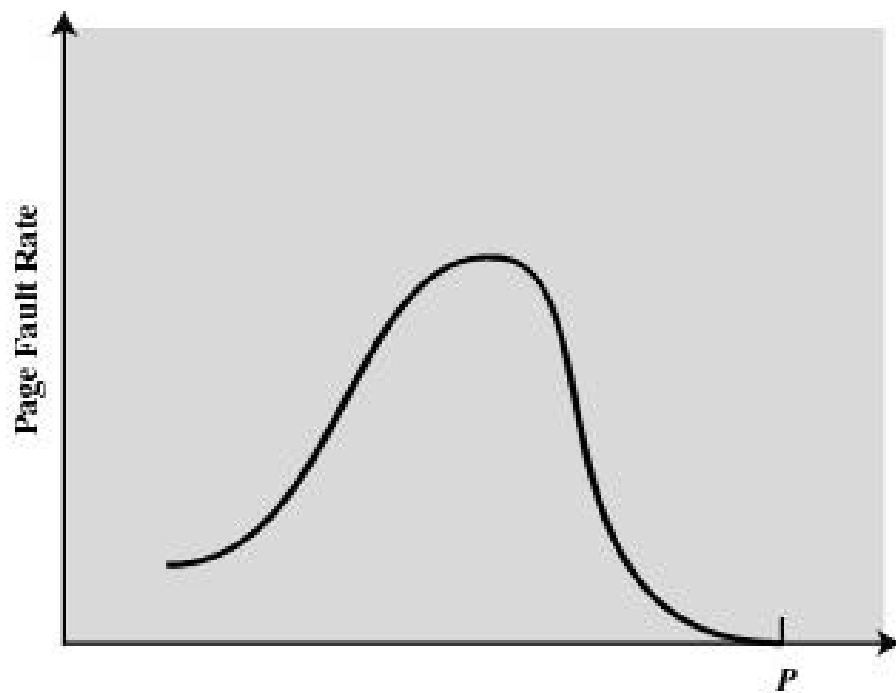


- ✍ Smaller page size, less amount of internal fragmentation
- ✍ Smaller page size, more pages required per process
 - ✍ more pages per process means larger page tables
 - ✍ larger page tables means large portion of page tables in virtual memory
- ✍ Secondary memory devices favor a larger page size for more efficient block transfer of data

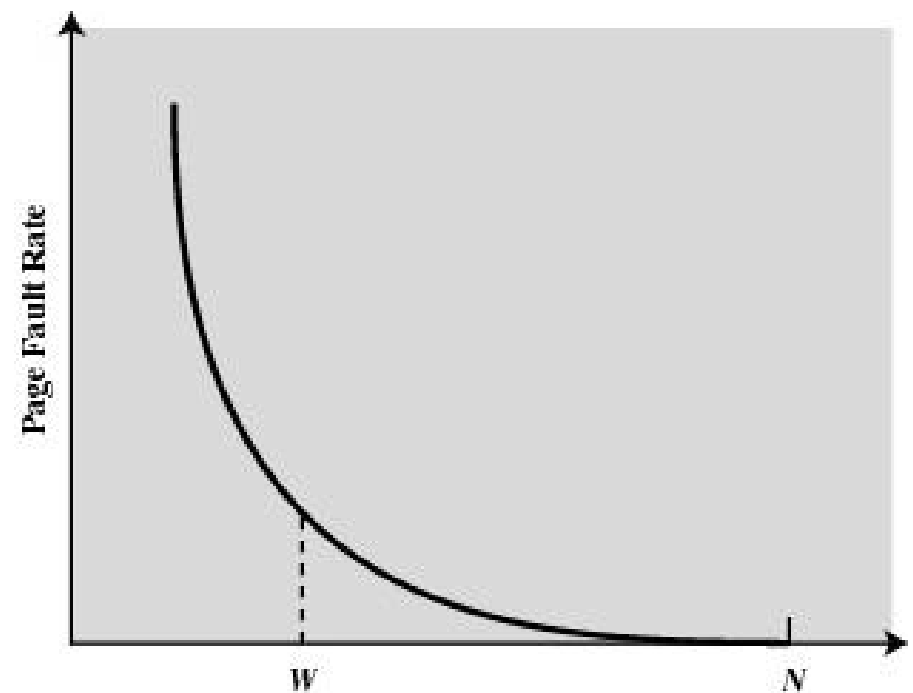
Page Size and Page Fault Rate



- ✍ Small page size, large number of pages will be found in main memory
- ✍ As time goes on during execution, the pages in memory will all contain portions of the process near recent references
 - ✍ page faults low
- ✍ Increased page size causes pages to contain locations further from any recent reference
 - ✍ effect of the principle of locality is weakened and page fault rate rise
 - ✍ page fault rate will begin to fall as the size of a page approaches the size of the entire process



(a) Page Size



(b) Number of Page Frames Allocated

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

Page Size and TLB Performance



- ✍ Multiple page sizes provide the flexibility needed to effectively use a TLB
 - ✍ Large pages can be used for program instructions
 - ✍ Small pages can be used for thread stacks
- ✍ But most operating system support only one page size
 - ✍ page size affects many aspects of OS

Table 8.2 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
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
PowerPc	4 Kbytes

VM Segmentation



- ✍ May be unequal, dynamic size

- ✍ Advantages

 - ✍ Simplifies handling of growing data structures

 - ✍ OS will expand or shrink the segment as needed

 - ✍ Allows programs to be altered and recompiled independently

 - ✍ Used for sharing data among processes

 - ✍ Lends itself to protection

 - ✍ a segment can be constructed to contain a well-defined set of programs or data

Segment Tables



- ✍ Address : (segment number, offset)
- ✍ Each entry contains the starting address of the corresponding segment in main memory
- ✍ Each entry contains the length of the segment
- ✍ A bit is needed to determine if segment is already in main memory
- ✍ A bit is needed to determine if the segment has been modified since it was loaded in main memory

Segment Table Entries

Virtual Address

Segment Number	Offset
----------------	--------

Segment Table Entry

P	M	Other Control Bits	Length	Segment Base
---	---	--------------------	--------	--------------

(b) Segmentation only

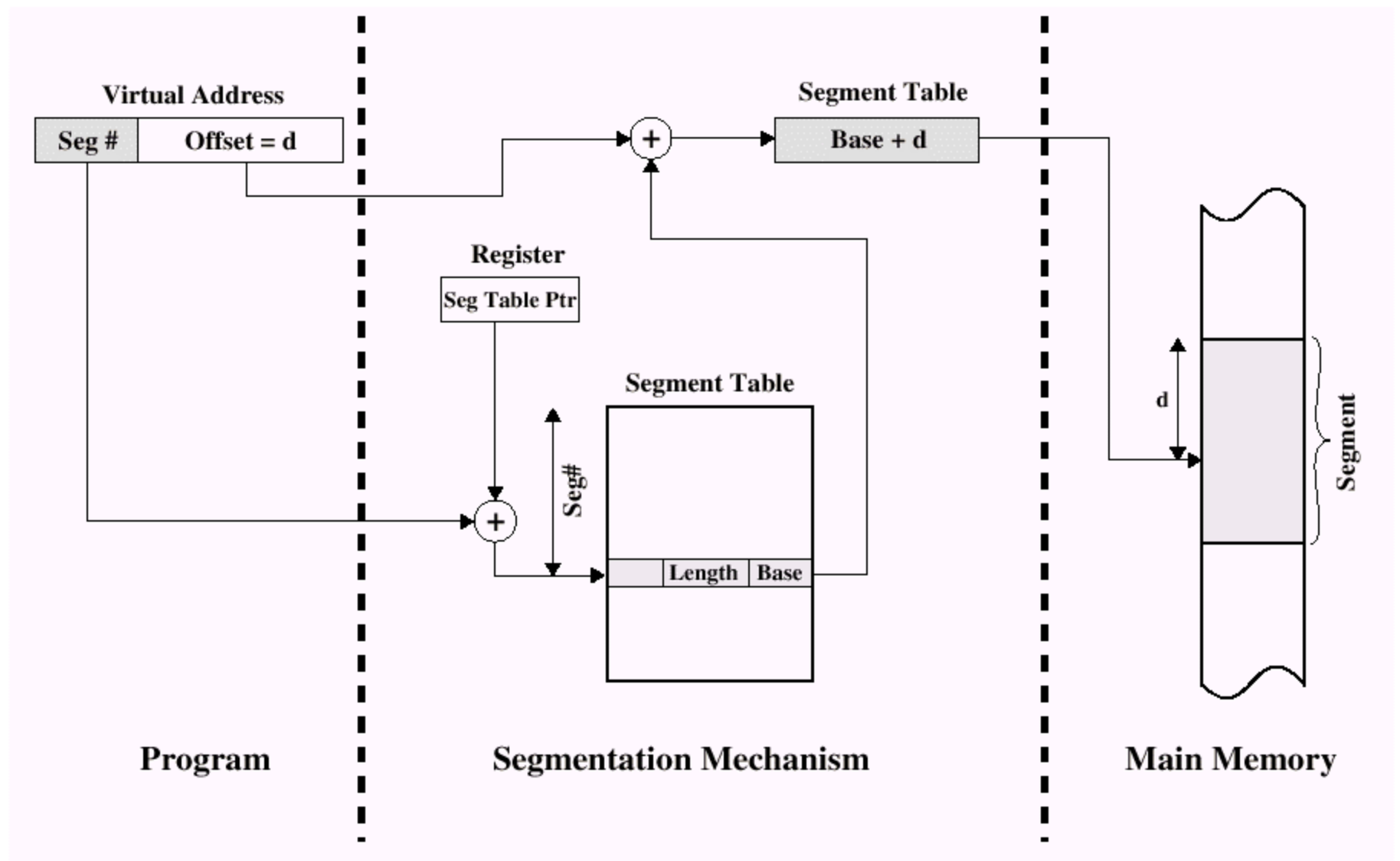


Figure 8.12 Address translation in a segmentation system

Combined Paging and Segmentation



- ✍ Paging is transparent to the programmer
- ✍ Paging eliminates external fragmentation
- ✍ Segmentation is visible to the programmer
- ✍ Segmentation allows for growing data structures, modularity, and support for sharing and protection
- ✍ Each segment is broken into fixed-size pages

Combined Segmentation and Paging

Virtual Address

Segment Number	Page Number	Offset
----------------	-------------	--------

Segment Table Entry

Other Control Bits	Length	Segment Base
--------------------	--------	--------------

Page Table Entry

P	M	Other Control Bits	Frame Number
---	---	--------------------	--------------

P= present bit
M = Modified bit

(c) Combined segmentation and paging



OS Software



- ✍ Design of memory-management depends on the following areas of choice
 - ✍ whether or not to use virtual memory
 - ✍ use of paging or segmentation or both
 - ✍ algorithms employed for various aspects of memory management
 - ✍ see next slide

Table 8.3 Operating System Policies for Virtual Memory

<p>Fetch Policy</p> <ul style="list-style-type: none"> Demand Prepaging <p>Placement Policy</p> <p>Replacement Policy</p> <ul style="list-style-type: none"> Basic Algorithms <ul style="list-style-type: none"> Optimal Least recently used (LRU) First-in-first-out (FIFO) Clock Page buffering 	<p>Resident Set Management</p> <ul style="list-style-type: none"> Resident set size <ul style="list-style-type: none"> Fixed Variable Replacement Scope <ul style="list-style-type: none"> Global Local <p>Cleaning Policy</p> <ul style="list-style-type: none"> Demand Precleaning <p>Load Control</p> <ul style="list-style-type: none"> Degree of multiprogramming
---	--

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
 - ✍ Prepaging brings in more pages than needed
 - ✍ more efficient to bring in pages that reside contiguously on the disk

Placement Policy



- ✍ Determines where in real memory a process piece is to reside
- ✍ Case of the segmentation system
 - ✍ best-fit, first-fit, next-fit...
- ✍ Case of the paging system
 - ✍ nothing to consider

Replacement Policy



- ✍ Deals with the selection of a page in memory to be replaced when a new page is brought in
- ✍ Frame locking used for frames that may not be replaced
 - ✍ kernel and key control structures of OS
 - ✍ I/O buffers

Algorithm in 5.9 should be changed

FROM

```
boolean choosing[n];
int number[n];
while(true)
{
    choosing[i] = true;
    for (int j = 0; j <> n: j++)
    {
        while(choosing[j])
        { };
        while((number[j] != 0) &&
            (number[j],j) <> (number[i], i))
        { };
        /* critical section */
        number[i] = 0;
        /* remainder */
    }
}
```

TO

```
boolean choosing[n];
int number[n];
while(true)
{
    choosing[i] = true;
    for (int j = 0; j <> n: j++)
    {
        while(choosing[j])
        { };
        while((number[j] != 0) &&
            (number[j],j) < (number[i], i))
        { };
    }
    /* critical section */
    number[i] = 0;
    /* remainder */
}
```

Replacement Policy



Basic algorithms

-  Optimal

-  Least recently used(LRU)





-  First-in-first-out(FIFO)

-  Clock

Replacement Policy







Optimal algorithm

-  selects for replacement that page for which the time to the next reference is the longest
-  results in the fewest number of page faults
-  impossible to have perfect knowledge of future events: impossible to implement
-  used to judge other algorithms

Replacement Policy







Least Recently Used (LRU)

-  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
-  Each page need to be tagged with the time of last reference
 -  require a great deal of overhead.

Replacement Policy



First-In, First-Out (FIFO)

-  Simplest replacement policy to implement
-  Treats page frames allocated to a process as a circular buffer
-  Page that has been in memory the longest is replaced
 -  these pages may be needed again very soon

Replacement Policy

✍ Second Chance algorithm

- ✍ FIFO with *use-bit* (or *reference bit*, R)

- ✍ avoid the problem of throwing out heavily used pages

- ✍ if the R bit is 0, the page is both old and unused, so it is replaced immediately






- ✍ if the R bit is 1, the bit is cleared, the page is put onto the end of the list of pages, and its load time is updated as though it had just arrived in memory

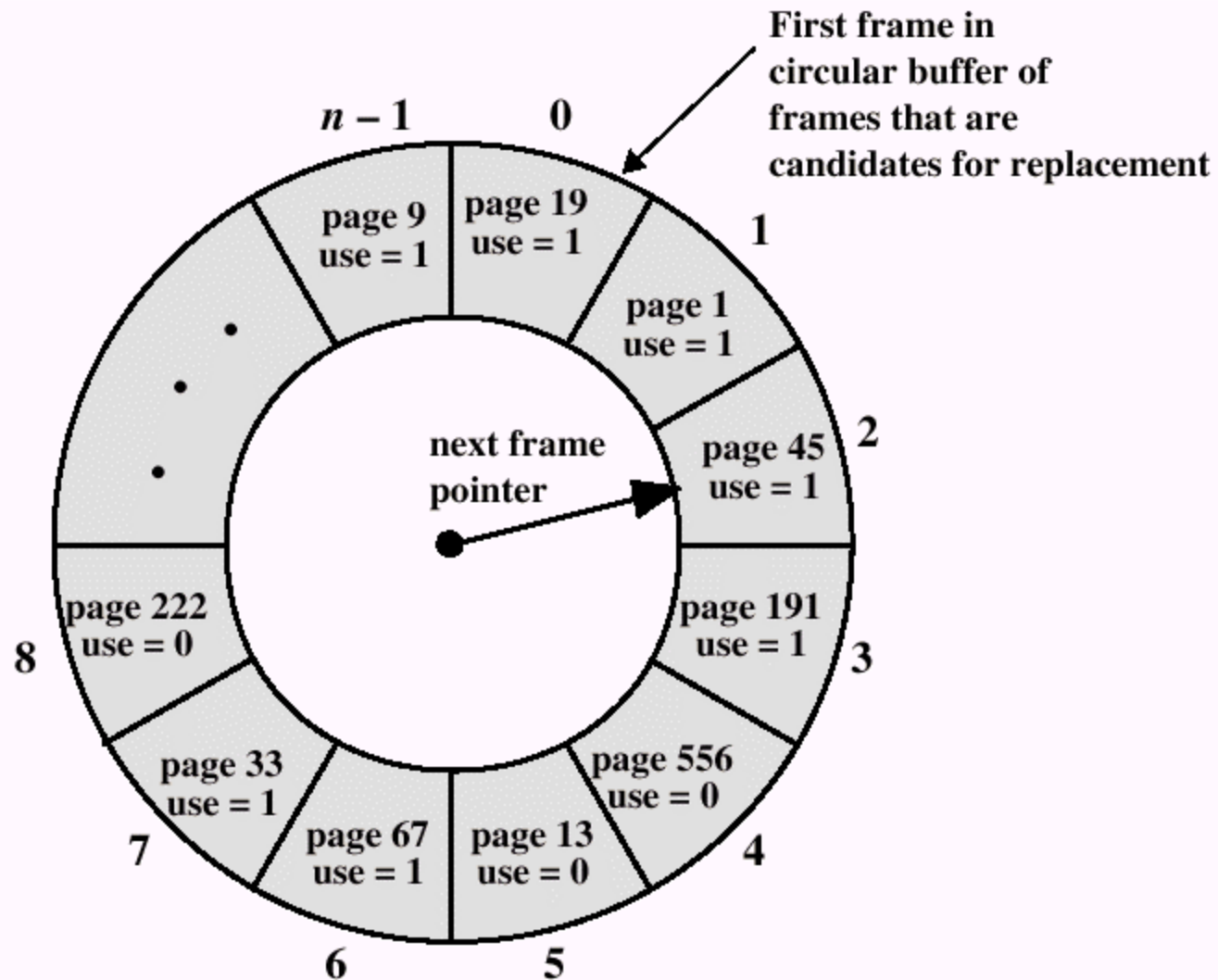
- ✍ inefficient because it is constantly moving pages around on its list

Replacement Policy



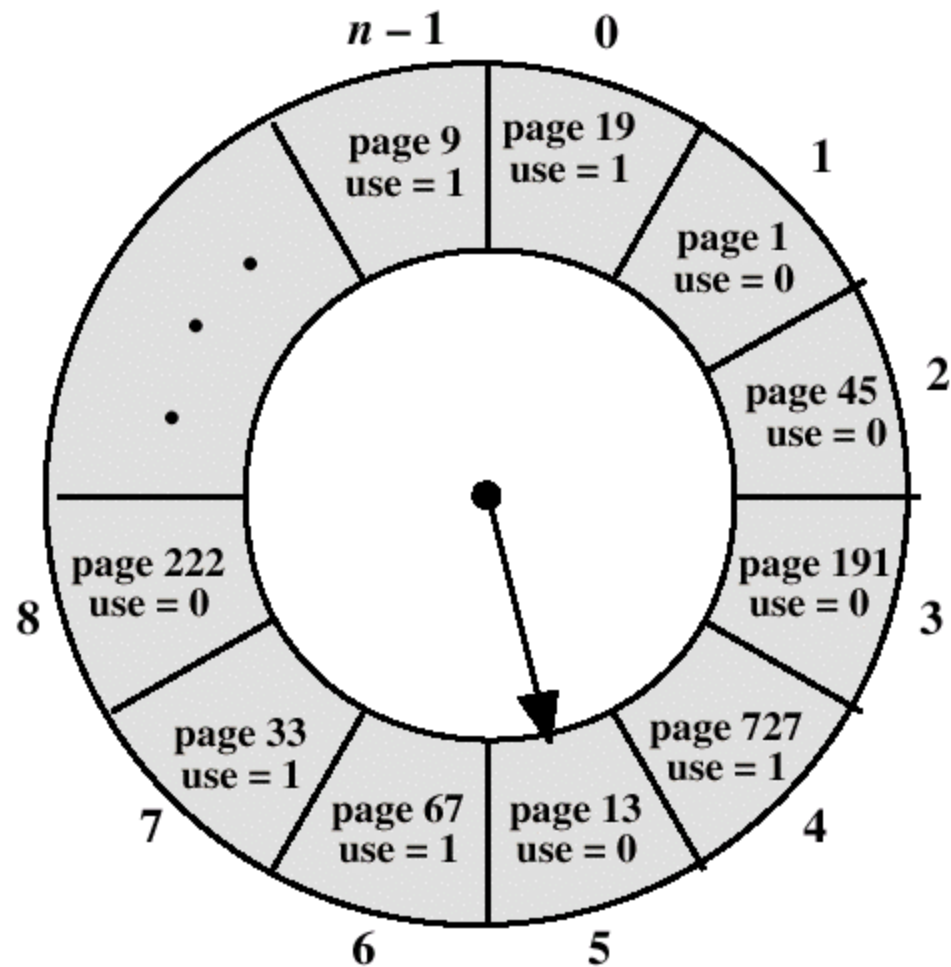
Clock algorithm

-  Additional bit called a use bit
-  When a page is first loaded in memory, use bit is set to 1
-  When the page is referenced, use bit is set to 1
-  When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced.
-  During the search for replacement, each use bit with 1 is changed to 0



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

Example 8.16 Example of clock policy operation



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

Example 8.16 Example of clock policy operation

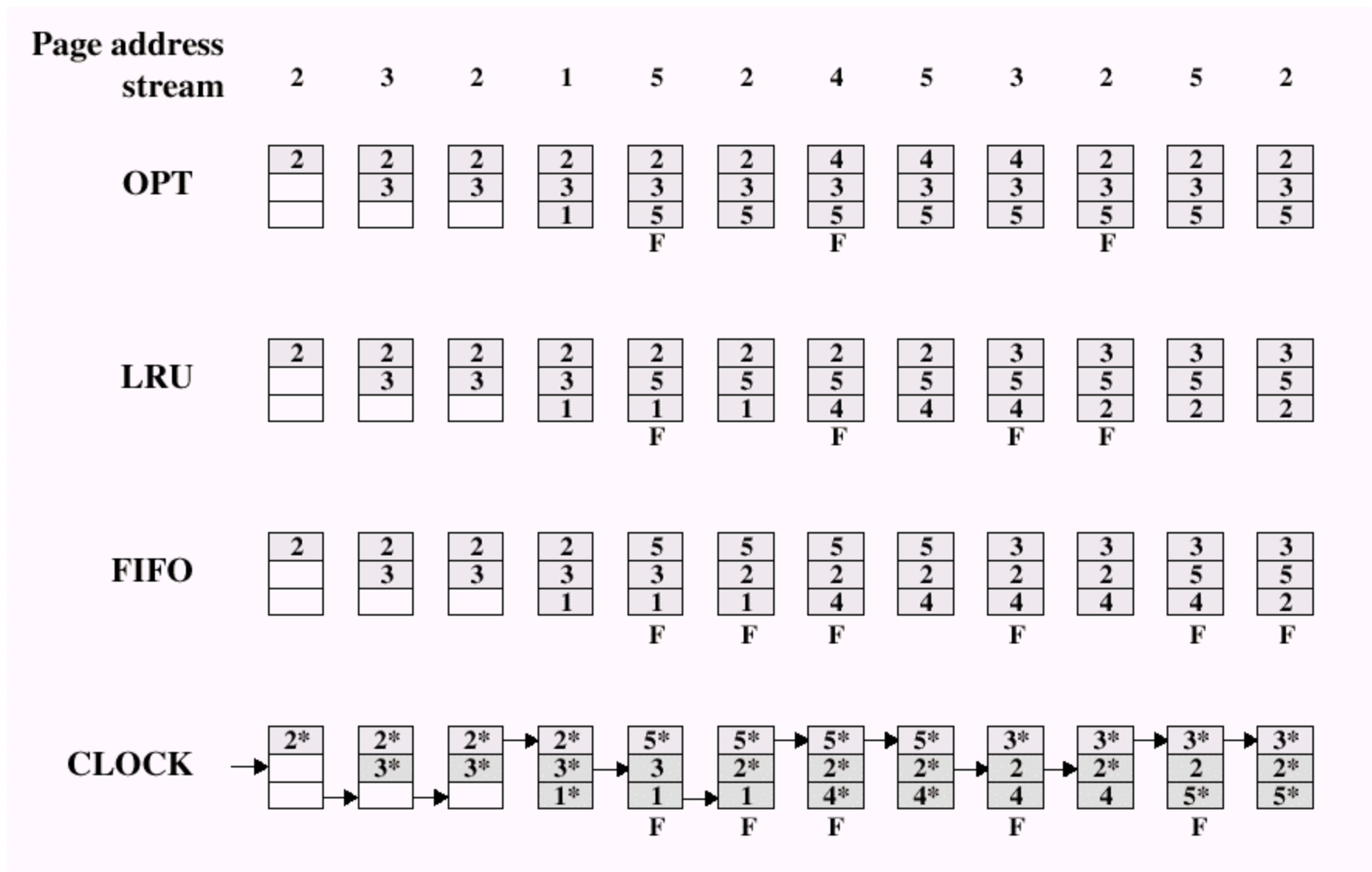


Figure 8.15 Behavior of four page replacement algorithms

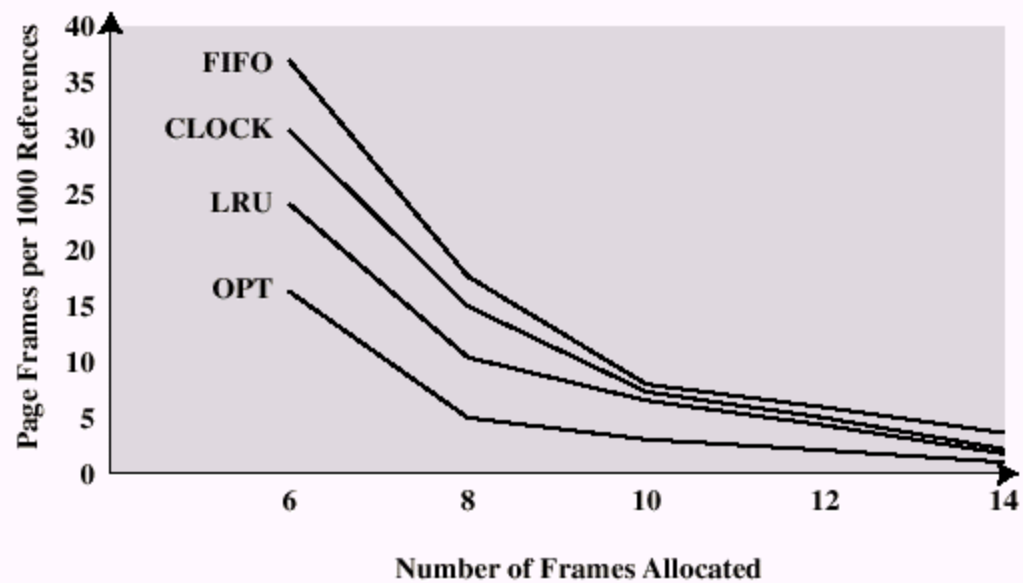


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Replacement Policy



NRU (Not Recently Used)

-  reference bit (R) and modified bit (M)

-  timer interrupt : periodically clear R bit

 -  Class 0 : not recently referenced, not modified

 -  Class 1 : not recently referenced, modified

 -  Class 2 : referenced, not modified

 -  Class 3 : referenced, modified

-  removes a page at random from the lowest numbered nonempty class

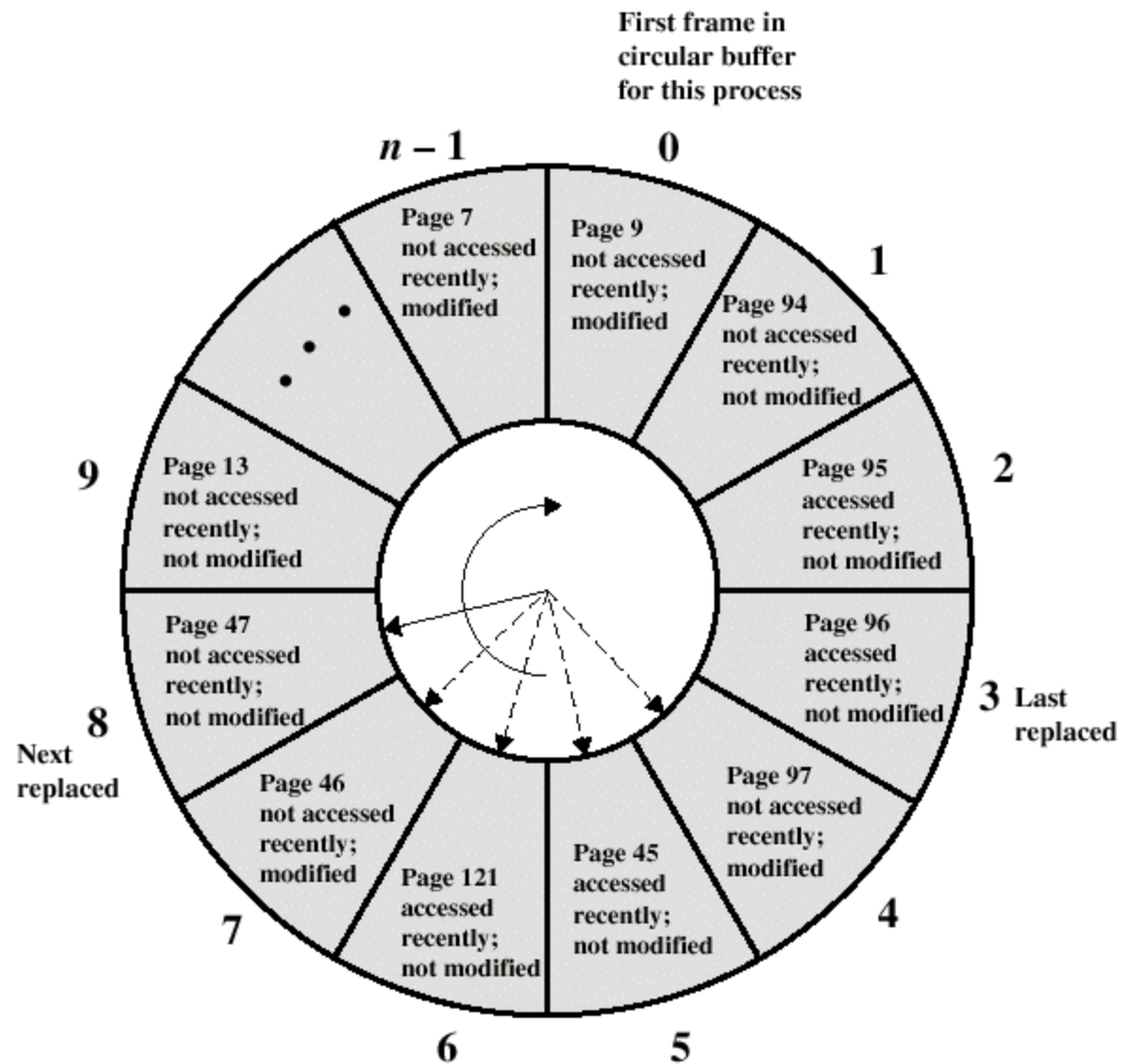


Figure 8.18 The clock page replacement algorithm

Replacement Policy




NFU (Not Frequently Used)

-  reference bit (R) and software counter

-  timer interrupt

 -  periodically add R bit (0 or 1) to the counter

-  the counters are an attempt to keep track of how often each page had been referenced


-  when a page fault occurs, the page with the lowest counter is chosen for replacement

Replacement Policy




Aging (NFU with aging)

timer interrupt

 periodically add R bit (0 or 1) to the leftmost bit of the counter, rather than the rightmost bit



 counter is shifted right 1 bit before the R bit is added in

 when a page fault occurs, the page with the lowest counter is chosen for replacement

Resident Set Management



Fixed-allocation

-  gives a process a fixed number of pages within which to execute
-  when a page fault occurs, one of the pages of that process must be replaced

Variable-allocation

-  number of pages allocated to a process varies over the lifetime of the process





Resident Set Management

	Local Replacement	Global Replacement
Fixed Allocation	Number of frames allocated to 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

Resident Set Management



Variable-allocation, local scope

-  when a new process is loaded into main memory, allocate to it a certain number of page frames
-  when a page fault occurs, select the page to replace from among the resident set of the faulting process
-  from time to time, reevaluate the allocation provided to the process, and increase or decrease
-  key elements of this strategy are the resident set size and the timing of changes

Working Set Strategy

✍ Working Set : $W(t, ?)$

✍ the set of pages in the most recent ?page references

✍ ? : working set window

✍ the working set size

✍ nondecreasing function of the window size

✍ $W(t, ? + 1) \supseteq W(t, ?)$

✍ $1 \leq |W(t, ?)| \leq \min(?, N)$




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

Figure 8.19 Working set of process as defined by window size

Working Set Strategy



Working Set Policy

-  monitor the working set of each process
-  periodically remove from the resident set of a process those pages that are not in its working set
-  a process may execute only if its working set is in main memory

Problems with Working Set










- ✍ The past does not always predict the future
- ✍ A true measurement of working set is impractical (too much overhead)
- ✍ Optimal value of α is unknown and in any case would vary

PFF algorithm



Page Fault Frequency algorithm

-  an attempt to approximate the working set strategy
-  use bit is set to 1 when a page is accessed
-  a threshold F is defined
-  when a page fault occurs, OS notes the time since the last page fault
 -  if the amount of time since the last page fault is less than F , a page is added to the resident set
 -  otherwise, discard all pages with a use bit of zero
 -  reset the use bit on the remaining pages to zero

Cleaning Policy



Demand cleaning

 a page is written out only when it has been selected for replacement

Precleaning

 pages are written out in batches

Cleaning Policy



- ✍ Better approach uses page buffering
 - ✍ Replaced pages are placed in two lists
 - ✍ modified and unmodified
 - ✍ Pages in the modified list are periodically written out in batches
 - ✍ Pages in the unmodified list are either reclaimed if referenced again or lost when its frame is assigned to another page

Load Control



- ✍ Determines the number of processes that will be resident in main memory
- ✍ Too few processes, many occasions when all processes will be blocked and processor will be idle
- ✍ Too many processes will lead to thrashing

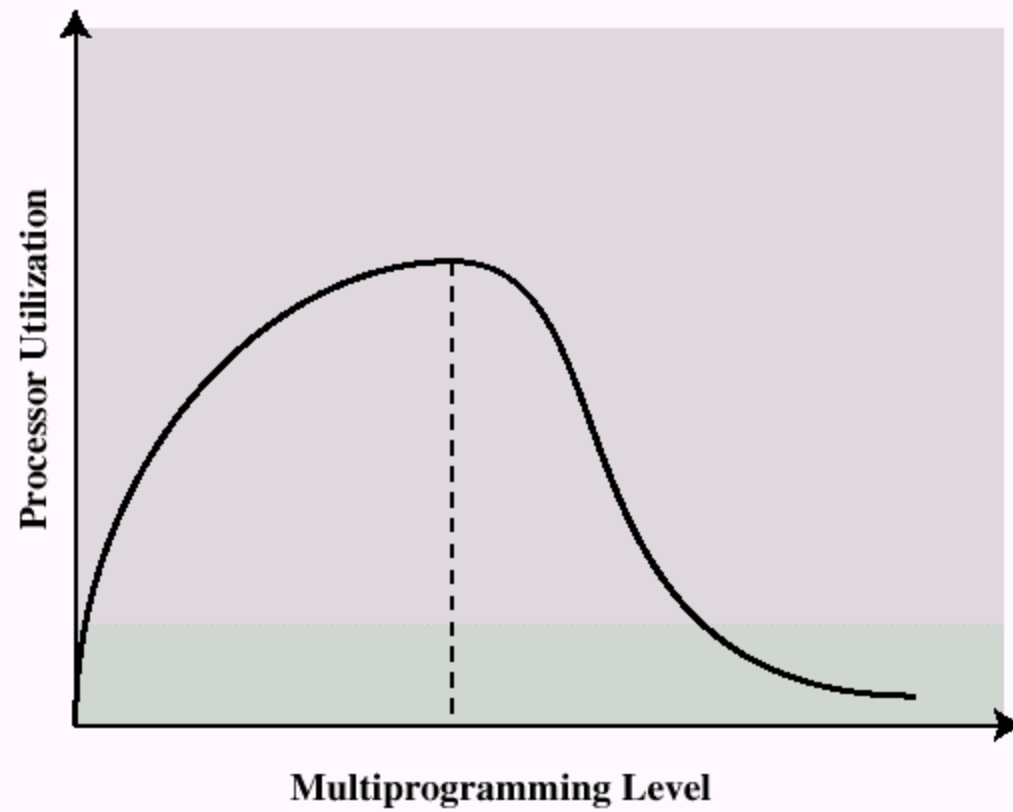


Figure 8.21 Multiprogramming Effects


Load Control



Process suspension

-  Lowest priority process

-  Faulting process

 -  this process does not have its working set in main memory so it will be blocked anyway

-  Last process activated

 -  this process is least likely to have its working set resident

Load Control



- ✍ Process with smallest resident set
 - ✍ this process requires the least future effort to reload
- ✍ Largest process
 - ✍ obtains the most free frames
- ✍ Process with the largest remaining execution window

UNIX and Solaris Memory Management







- ✍ Paging system for user processes
 - ✍ Page table
 - ✍ Disk block descriptor
 - ✍ Page frame data table
 - ✍ Swap-use table
- ✍ Kernel memory allocator for memory allocation for the kernel

UNIX and Solaris Memory Management



Data Structures

-  Page table - one per process
-  Disk block descriptor - describes the disk copy of the virtual page
-  Page frame data table - describes each frame of real memory
-  Swap-use table - one for each swap device

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

(a) Page table entry

Swap device number	Device block number	Type of storage
---------------------------	----------------------------	------------------------

(b) Disk block descriptor

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 zero 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.

Page state	Reference count	Logical device	Block number	Pfdata pointer
-------------------	----------------------------	---------------------------	-------------------------	---------------------------

(c) Page frame data table entry

Reference count	Page/storage unit number
----------------------------	-------------------------------------

(d) Swap-use table entry

Figure 8.22 UNIX SVR4 Memory Management Formats

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.

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.

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.

UNIX and Solaris Memory Management



Page Replacement

-  refinement of the clock policy known as the two-handed clock algorithm
-  uses the reference bit
 -  set to 0 when the page is first brought in
 -  set to 1 when the page is referenced
-  fronthand sweeps through the pages and sets the reference bit to 0
-  sometime later, backhand sweeps through the pages and collects the pages with reference bit 0

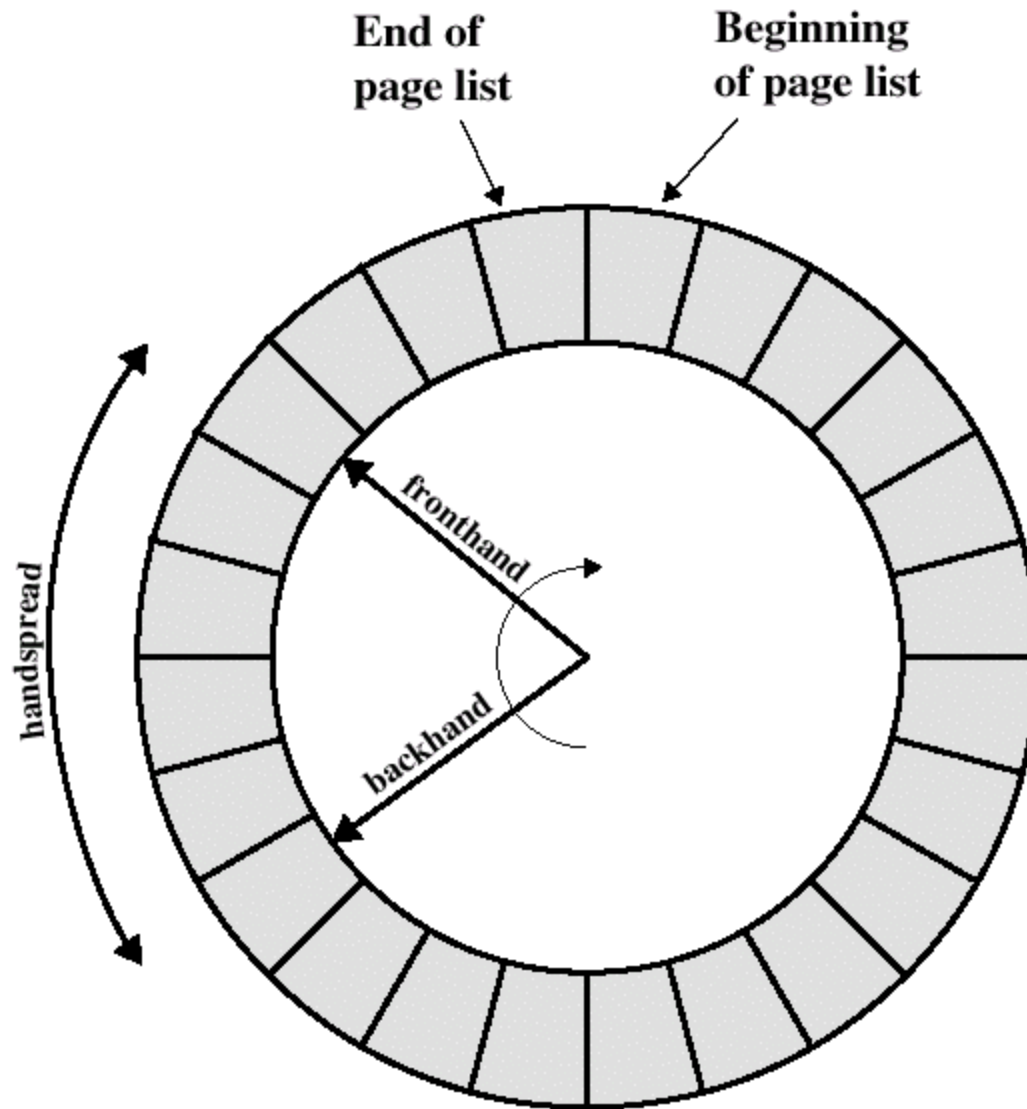


Figure 8.23 Two-handed clock page replacement algorithm


UNIX and Solaris Memory Management



Kernel Memory Allocator

-  kernel generates and destroys small tables and buffers frequently during execution

 -  they require dynamic memory allocation

-  most of these blocks are significantly smaller than the typical machine page size

 -  paging mechanism is inefficient here

-  in SVR4, modification of the buddy system is used

Linux Memory Management



Linux virtual memory

Virtual memory addressing(3 level scheme)

 Page directory

 Page middle directory

 Page table


 Virtual address consists of four fields

Linux Memory Management



Page allocation

-  buddy system is used

-  to enhance the efficiency of reading and writing, Linux defines a mechanism for dealing with contiguous blocks of pages mapped into contiguous blocks of page frames

Page replacement algorithm

-  modified clock algorithm is used


-  aging is considered

Windows 2000

Memory Management



W2K virtual address map

-  each process has a 32-bit address space

 -  2 GB for user process

 -  2 GB for system space which is shared by all processes

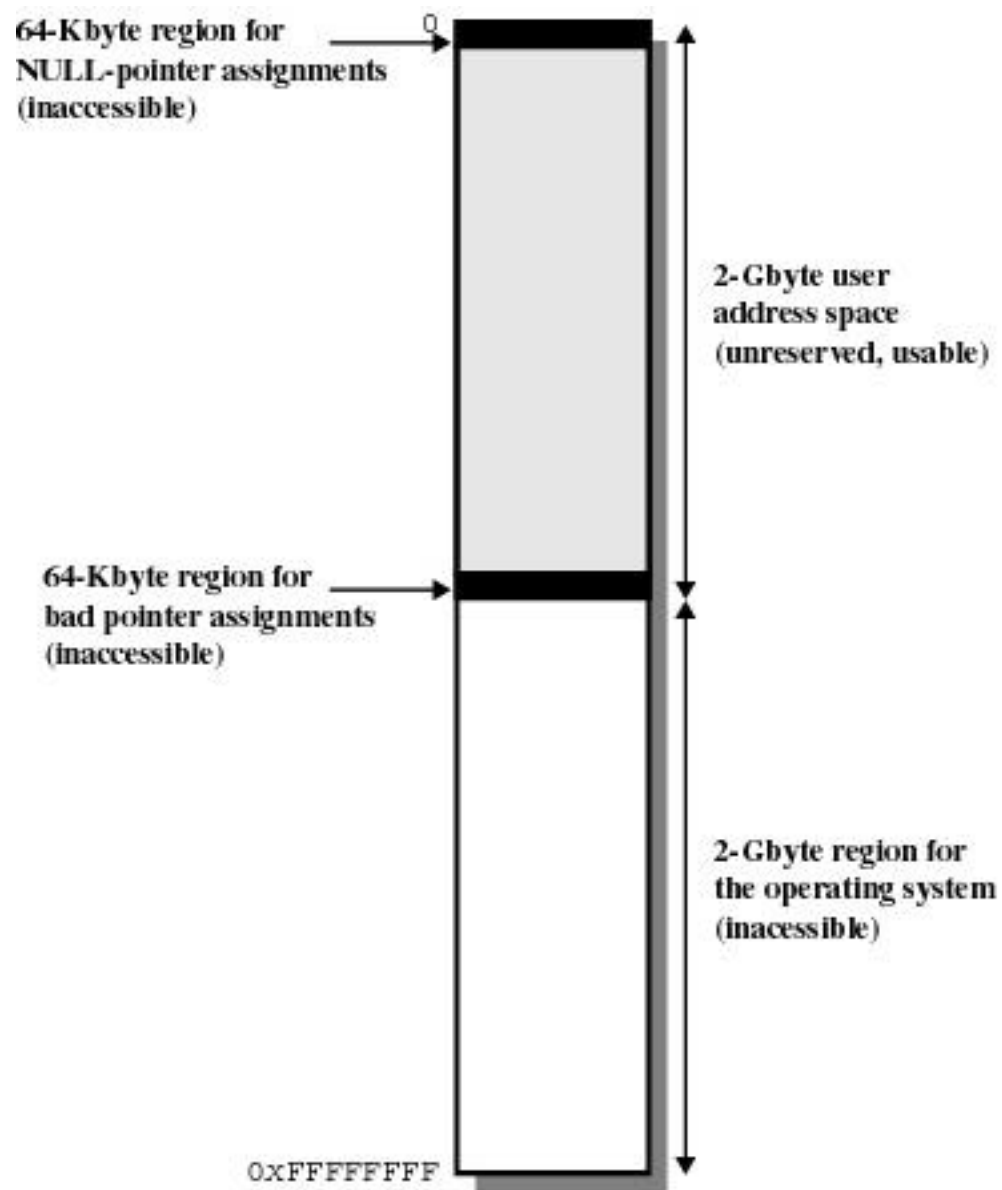


Figure 8.25 Windows 2000 Default Virtual Address Space

Windows 2000 Memory Management



W2K Paging

-  a page can be in one of 3 states


-  available

-  pages not currently used by this process

-  reserved

-  reserved but not counted against the process's memory quota

-  committed

-  pages for which the virtual memory manager has set aside space in its paging file