CSCI 360
Introduction to Operating Systems

Memory Management

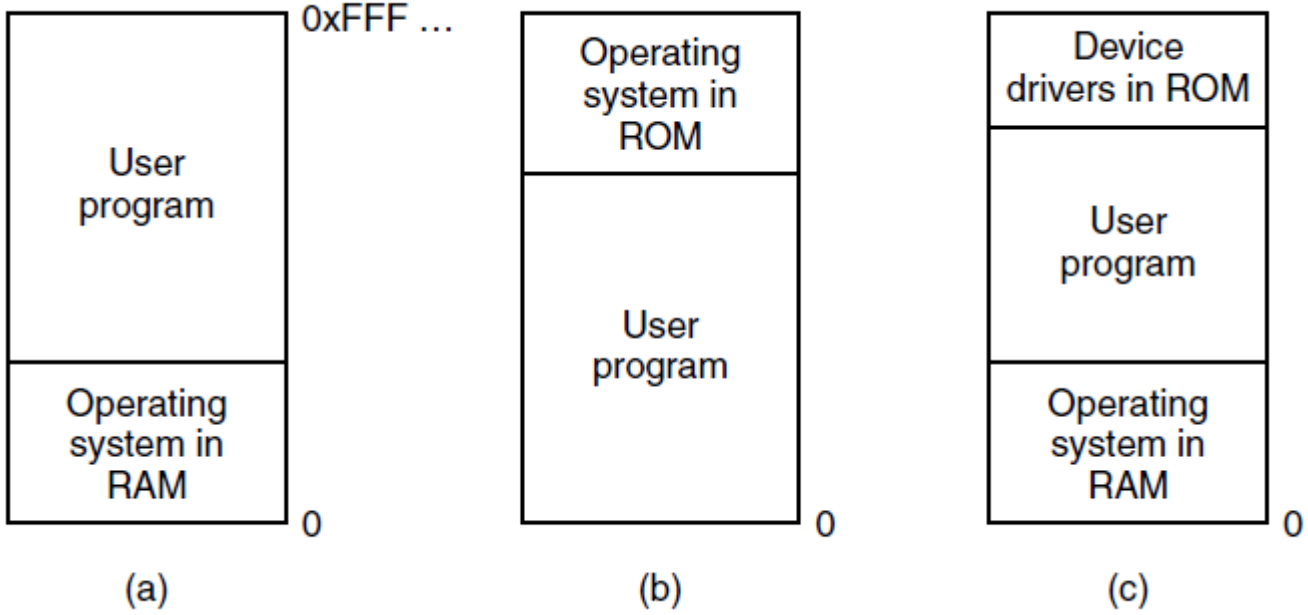
Humayun Kabir
Professor, CS, Vancouver Island University, BC, Canada

Outline

Memory Management

- Address Space
- Swapping
- Free Memory Management
- Memory Allocation Algorithms
- External Fragmentation and Compaction
- Virtual Memory and Paging
- Page Table
- Page Replacement Algorithms
- Page Fault
- Segmentation

No Memory Abstraction



Three simple ways of organizing memory with an operating system and one user process. Other possibilities also exist

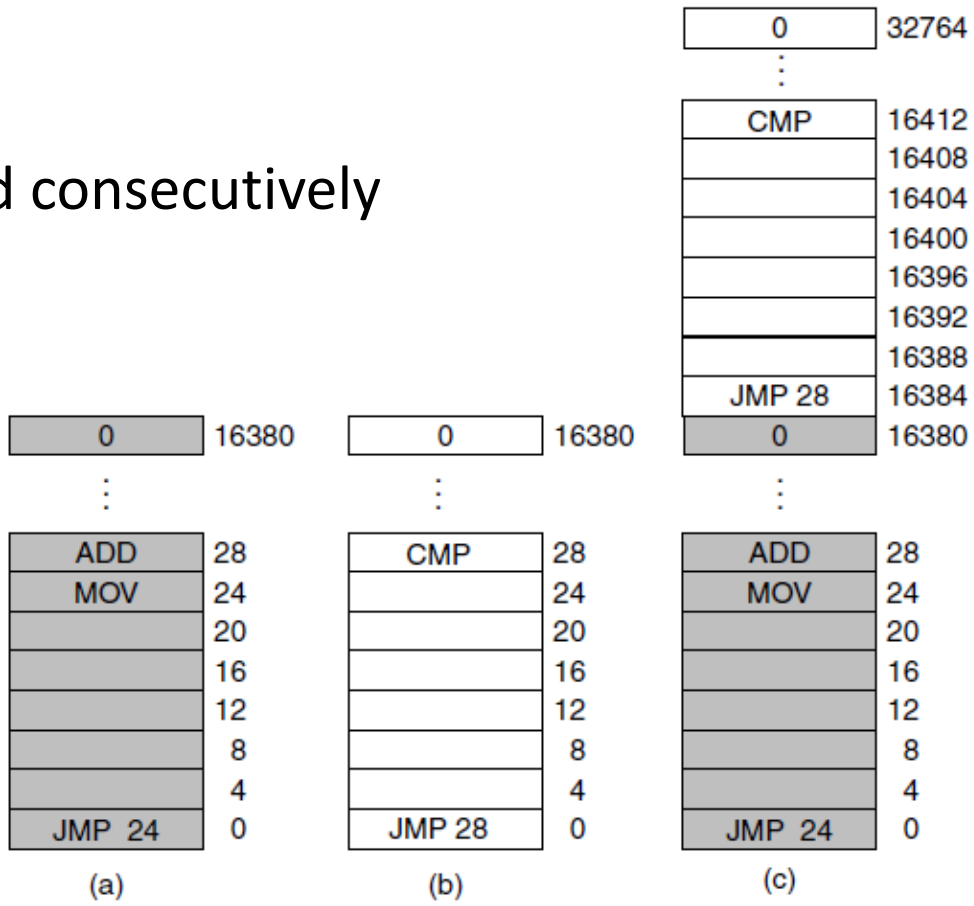
Without a Memory Abstraction

Illustration of the relocation problem

(a) A 16-KB program

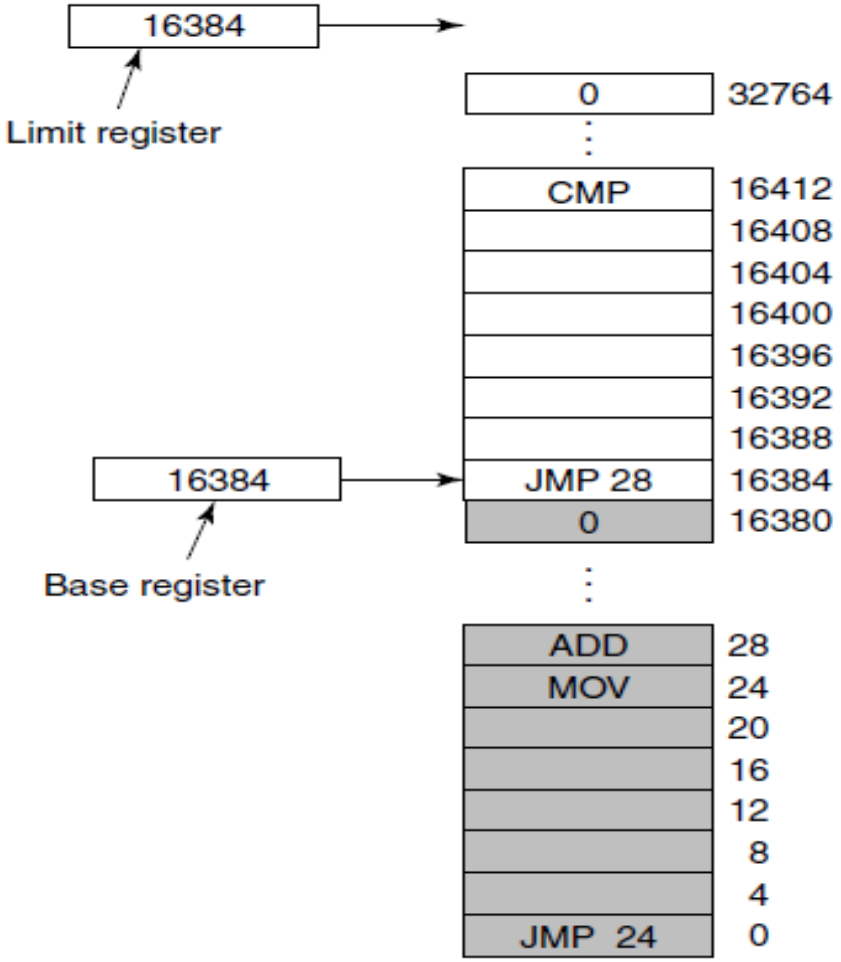
(b) Another 16-KB program

(c) The two programs loaded consecutively into memory.

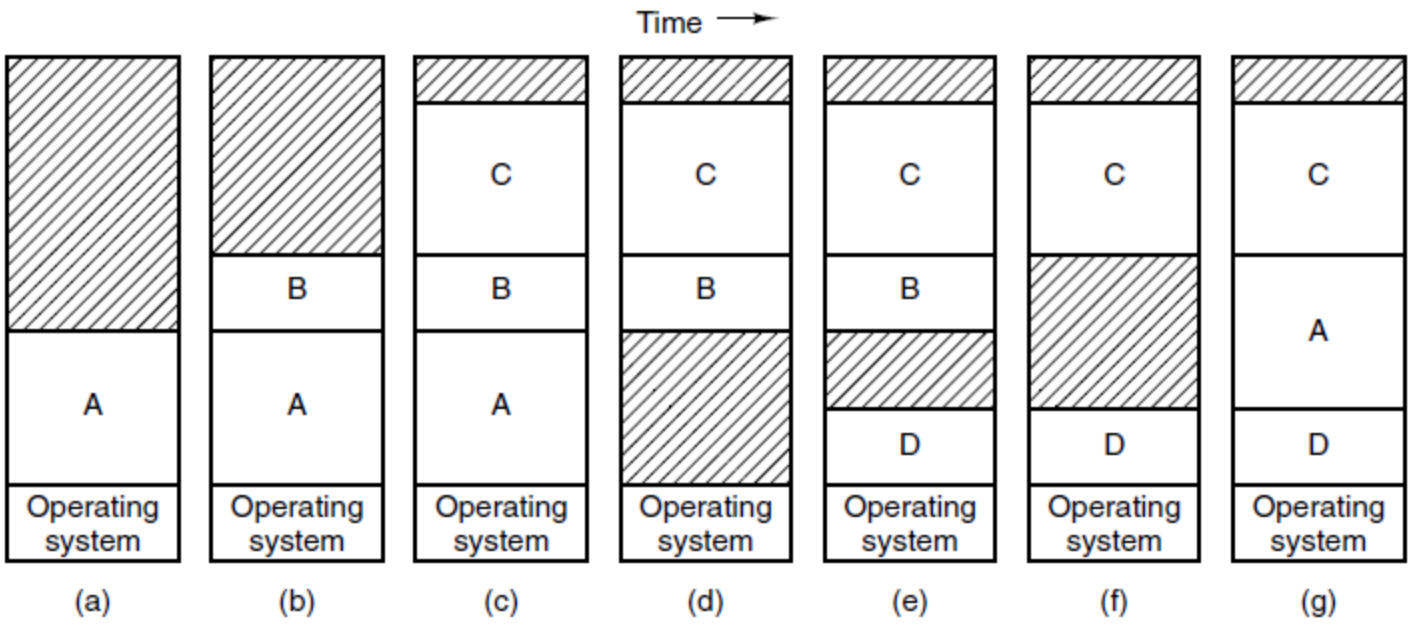


Memory Abstraction: Address Space

Base and Limit registers can be used to give each process a separate address space.

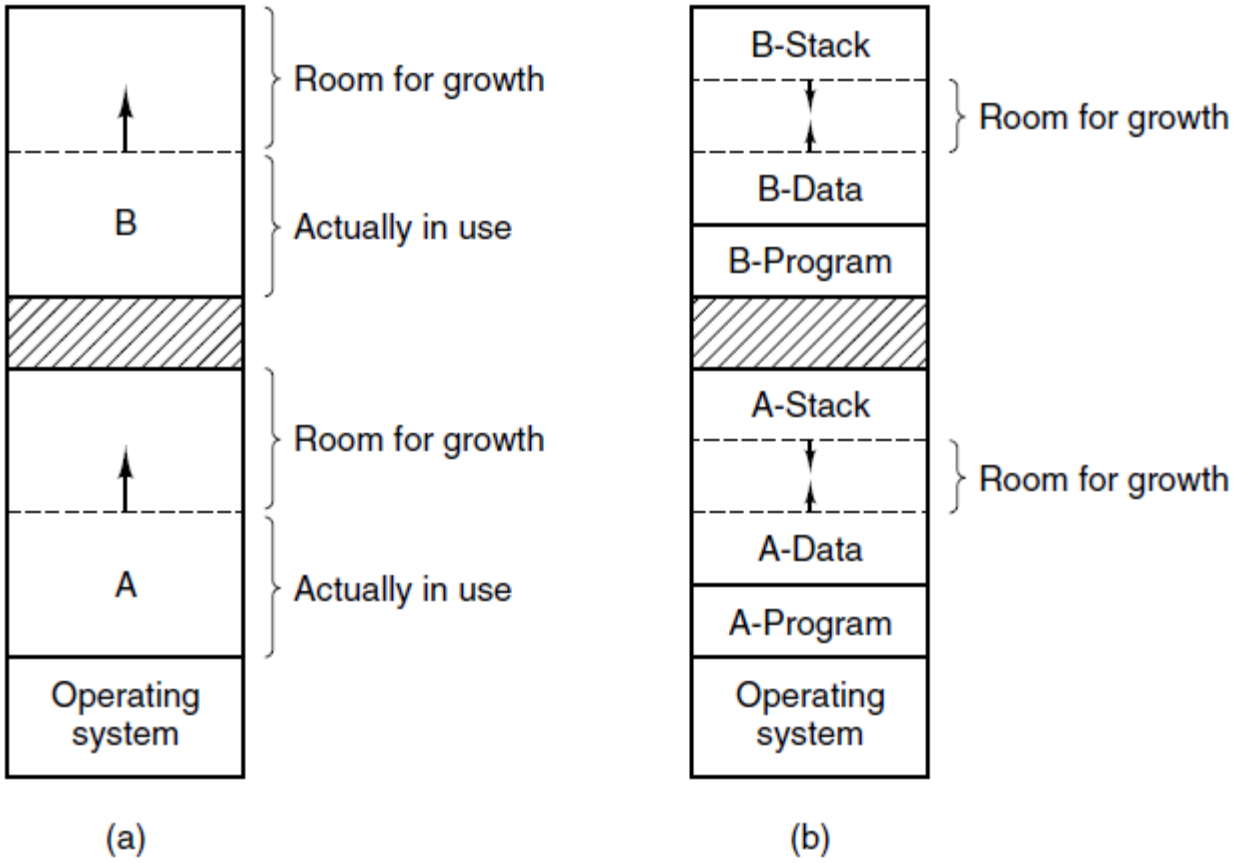


Swapping



Memory allocation changes as processes come into memory and leave it. The shaded regions are unused memory

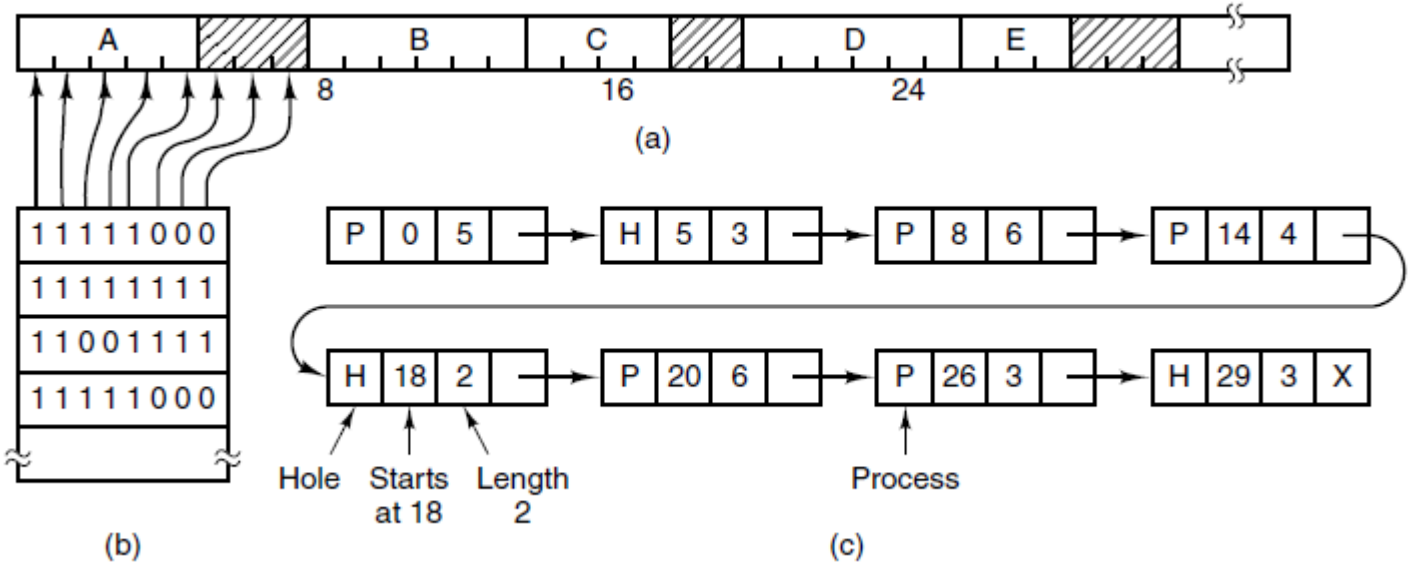
Swapping



(a) Allocating space for a growing data segment.
(b) Allocating space for a growing stack and a growing data segment.

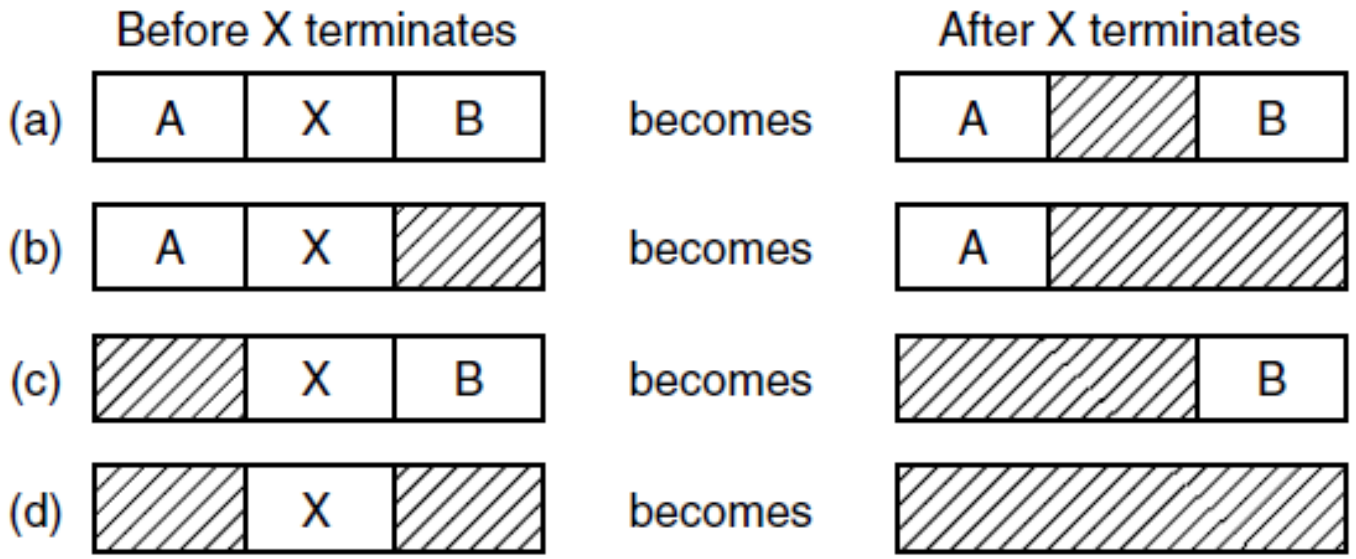
Free Memory Management

(a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions are free.



(b) The corresponding **bitmap**. (c) The same information as a **linked list**.

Free Memory Management



Four neighbor combinations for the terminating process, *X*.

Memory Allocation Algorithms

- First fit
- Next fit
- Best fit
- Worst fit
- Quick fit

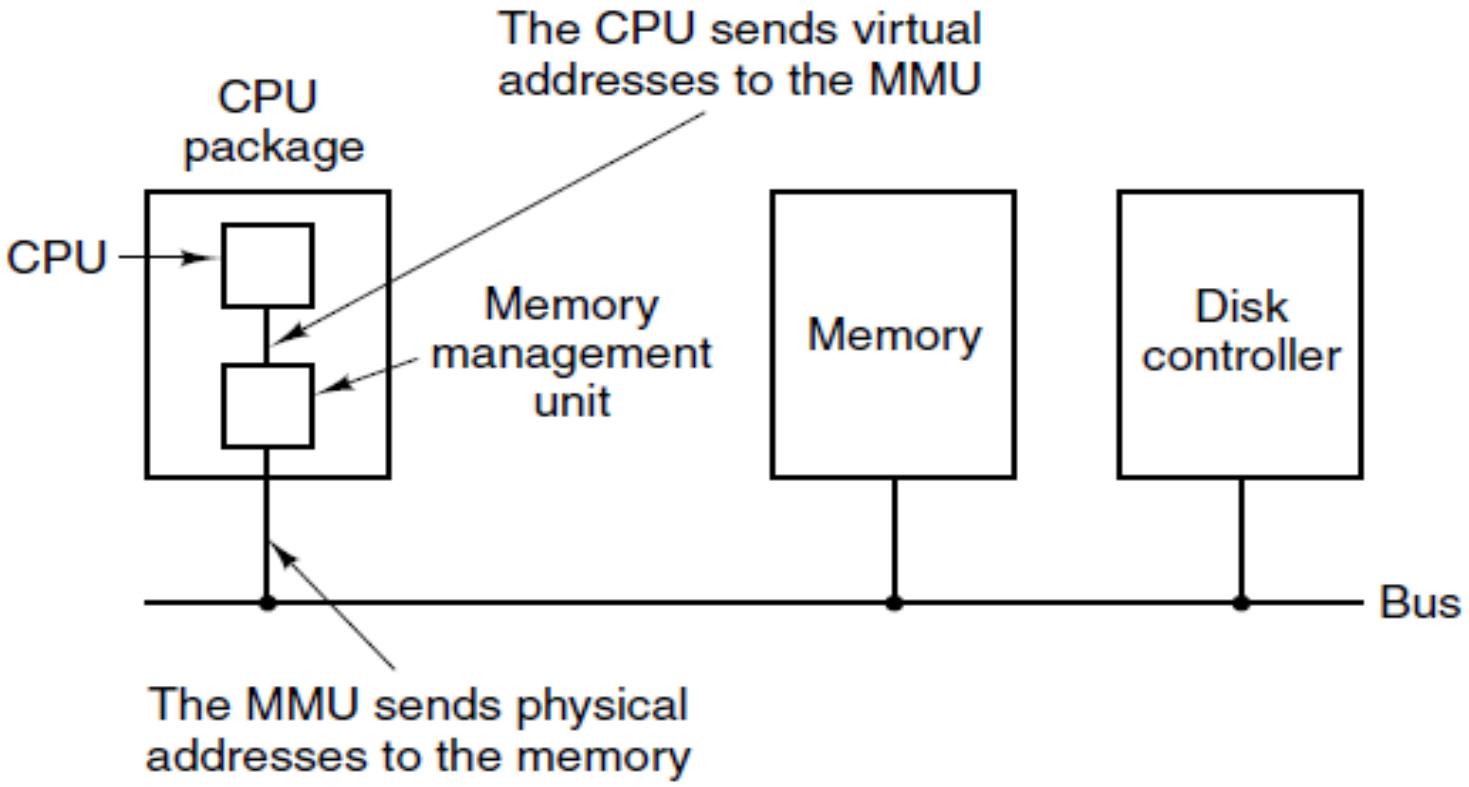
External Fragmentation and Compaction

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

Virtual Memory

- There is a need to run programs that are too large to fit in memory
- Solution adopted in the 1960s, split programs into little pieces, called overlays
  - Kept on the disk, swapped in and out of memory
- Virtual memory : each program has its own address space, broken up into chunks called pages

Paging

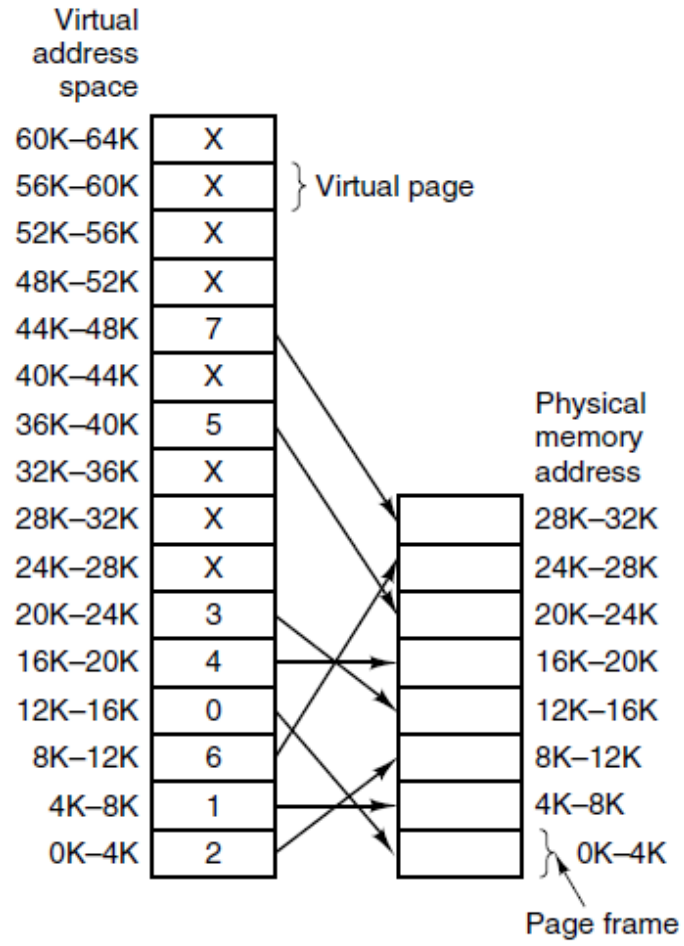


The position and function of the MMU.

Paging

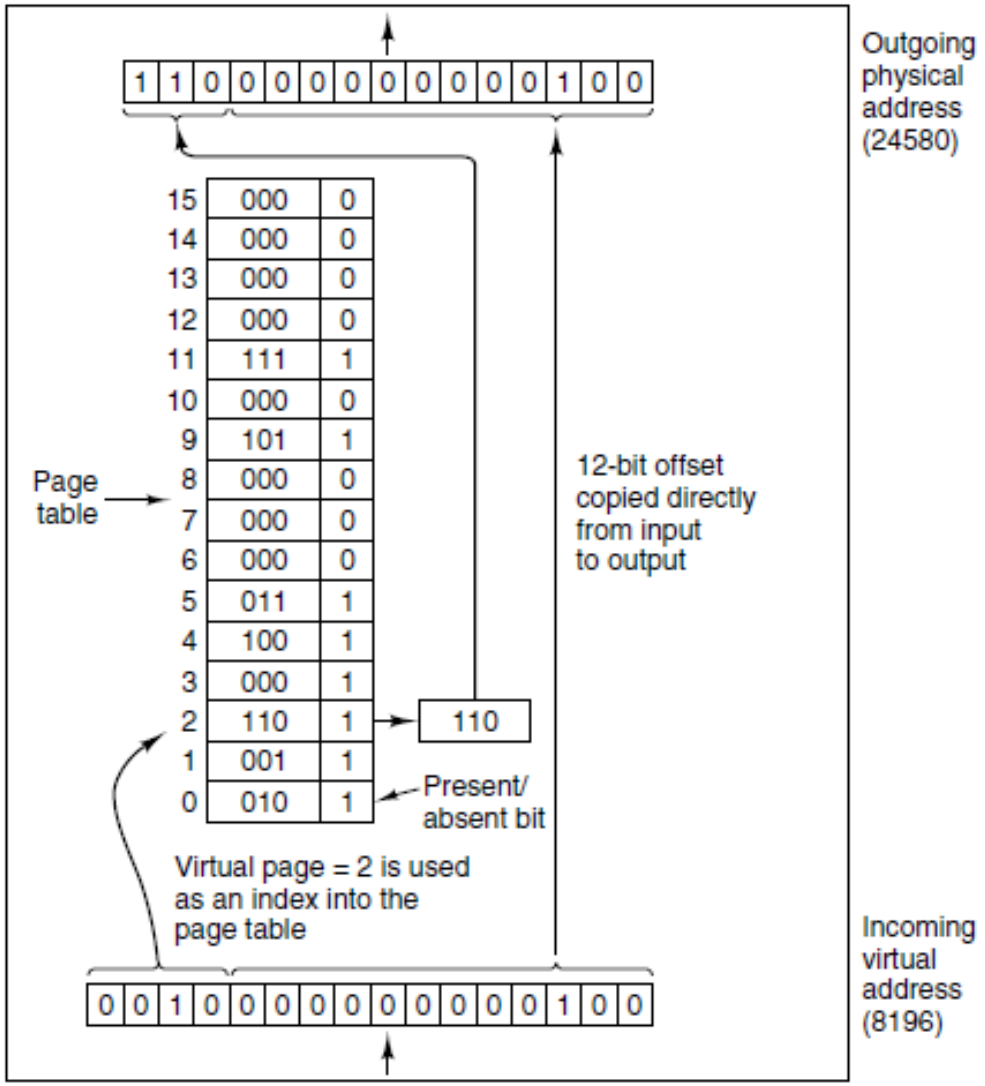
The relation between virtual addresses and physical memory addresses is given by the **page table**.

Every page begins on a multiple of 4096 and ends 4095 addresses higher, so 4K–8K really means 4096–8191 and 8K to 12K means 8192–12287



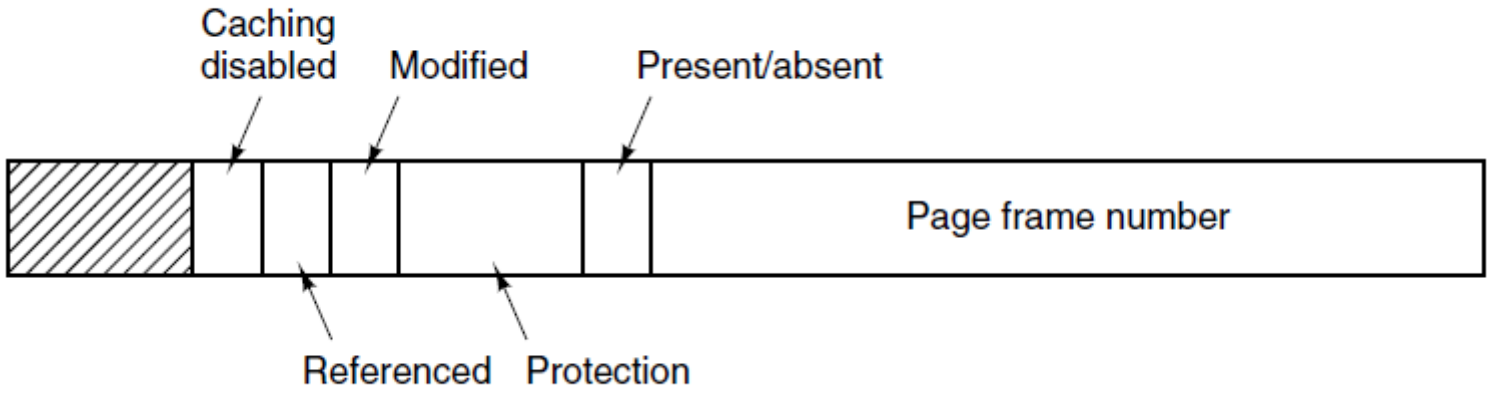
Paging

Virtual to **Physical address translation** operation MMU with 16 4-KB pages.



Page Table

A typical page table entry.



Speeding Up Paging

Major issues faced:

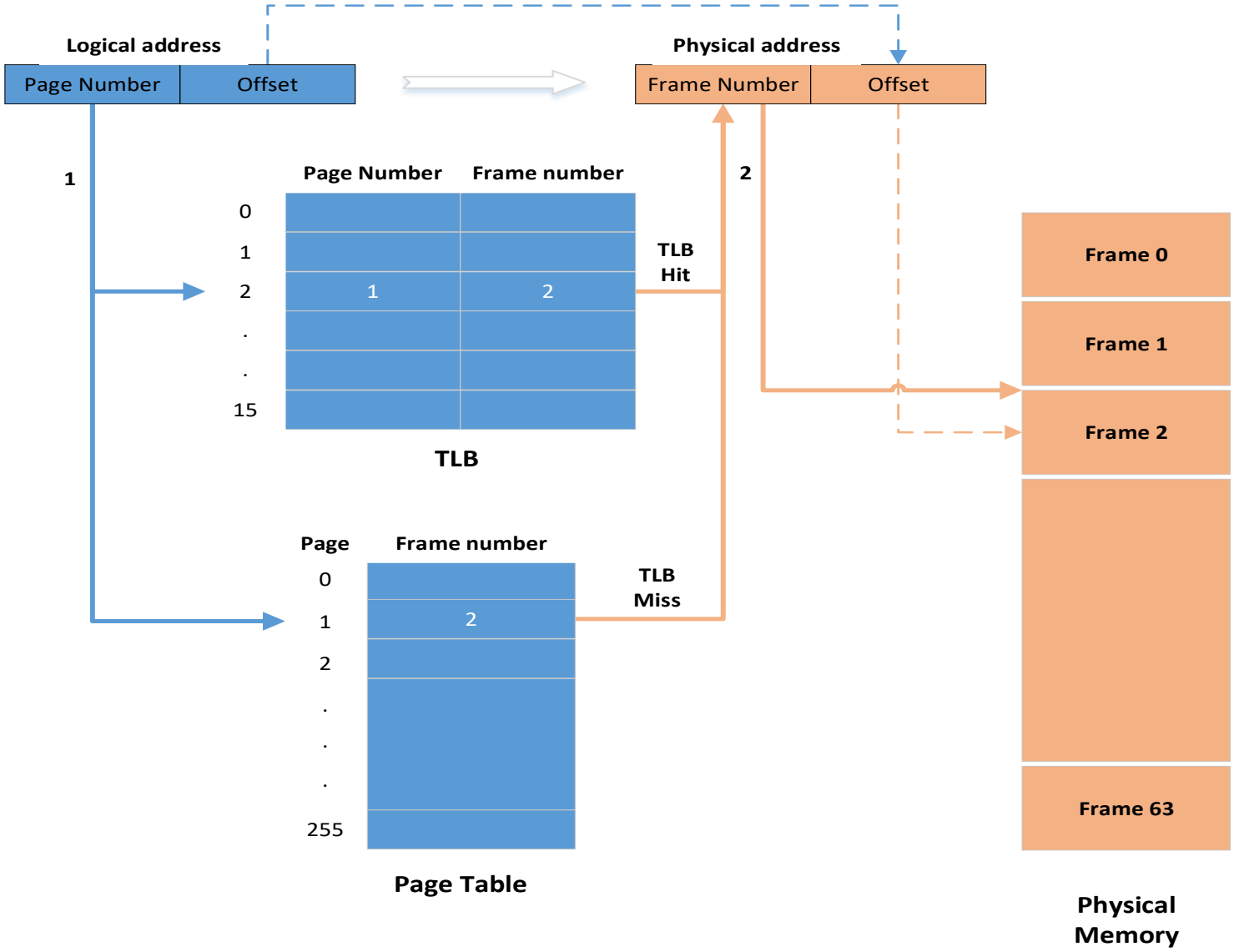
1. The mapping from virtual address to physical address must be fast.
2. If the virtual address space is large, the page table will be large.

Page Table

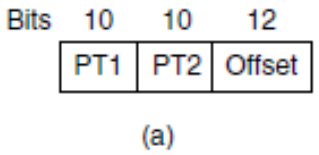
Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

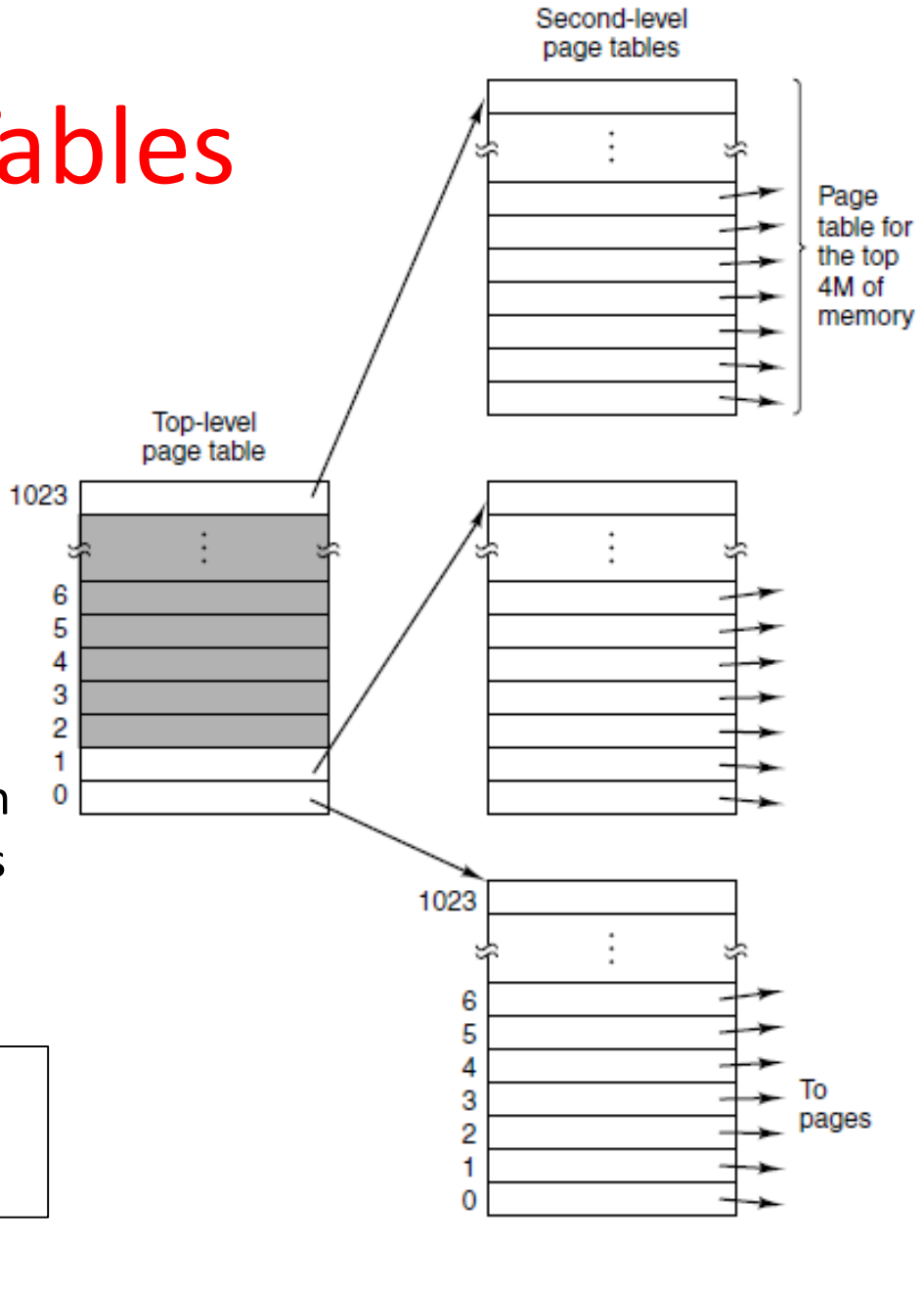
Paging


Logical address
Page Number
Offset
Physical address
Frame Number
Offset
1
2
Page Number
Frame number
0
1
2
.
.
15
1
2
TLB
TLB Hit
TLB Miss
Page
Frame number
0
1
2
.
.
.
255
2
Page Table
Frame 0
Frame 1
Frame 2
Frame 63
Physical Memory

Multilevel Page Tables



A 32-bit address with two page table fields

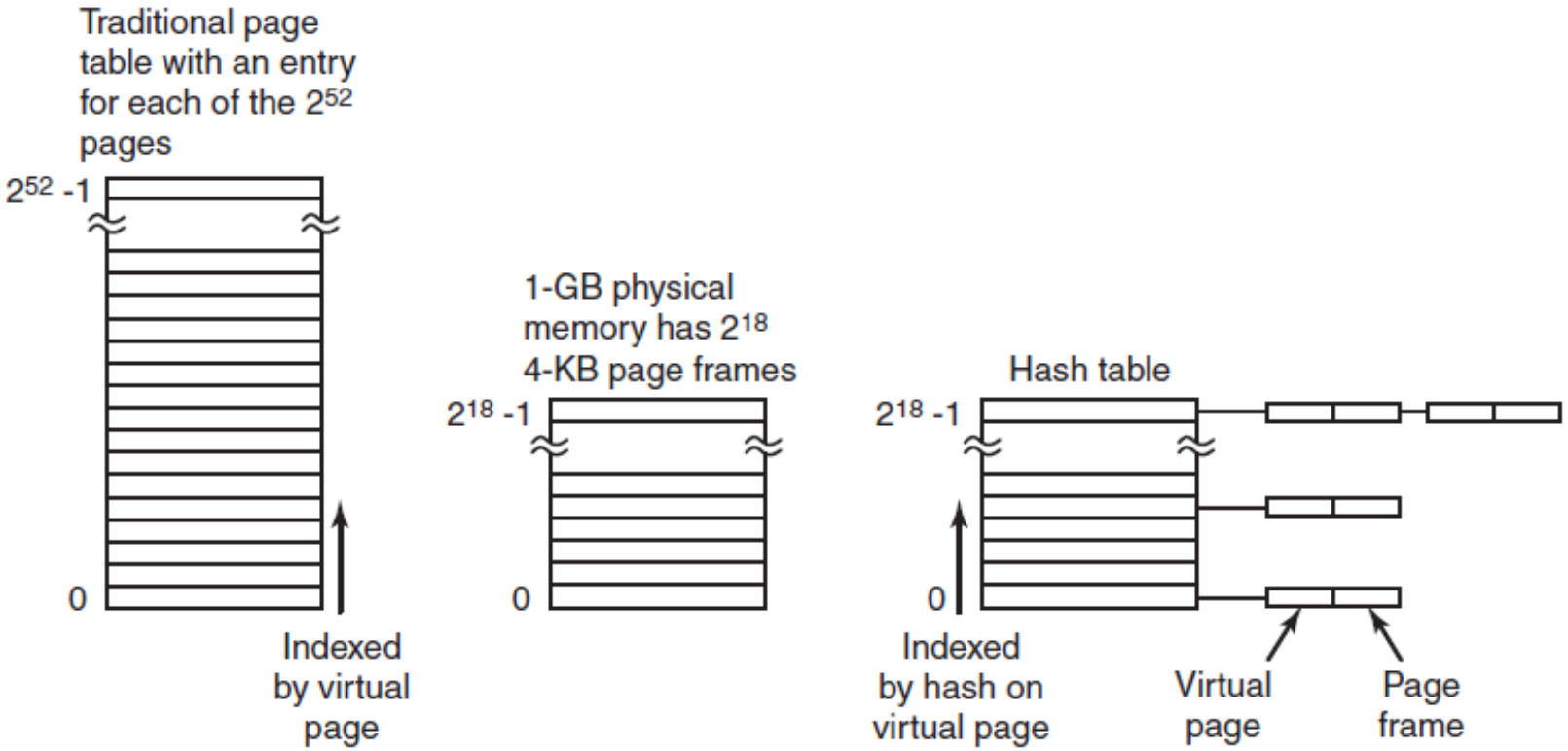


Two-level page tables

Virtual Addresses

0x00403004, 0x00803004

Inverted Page Tables



Comparison of a traditional page table with an inverted page table for 64-bit virtual address space and 1-GB physical memory

Page Replacement Algorithms

- Optimal algorithm
- Not recently used (NRU) algorithm
- First-in, first-out (FIFO) algorithm
- Second-chance algorithm
- Clock algorithm
- Least recently used (LRU) algorithm
- Working set algorithm
- WSClock algorithm

Optimal Algorithm

- Each page can be labeled with the number of instructions that will be executed before the page is first referenced.
- At page fault, the page with the highest label should be removed.
- It is unrealizable.

Not Recently Used (NRU) Algorithm

- R bit of a page is set when the page reference occurs and is cleared at every clock interrupt.
- M bit of a page is set when the page reference occurs to write and is not cleared at clock interrupts.

Not Recently Used (NRU) Algorithm

- At page fault, system categories pages based on the current values of their *R* and *M* bits:

  Class 0: not referenced, not modified.

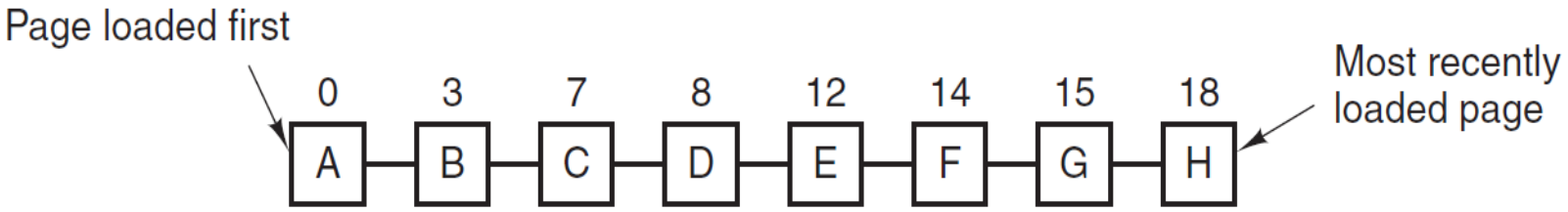
  Class 1: not referenced, modified.

  Class 2: referenced, not modified.

  Class 3: referenced, modified.

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

First In First Out (FIFO) Algorithm

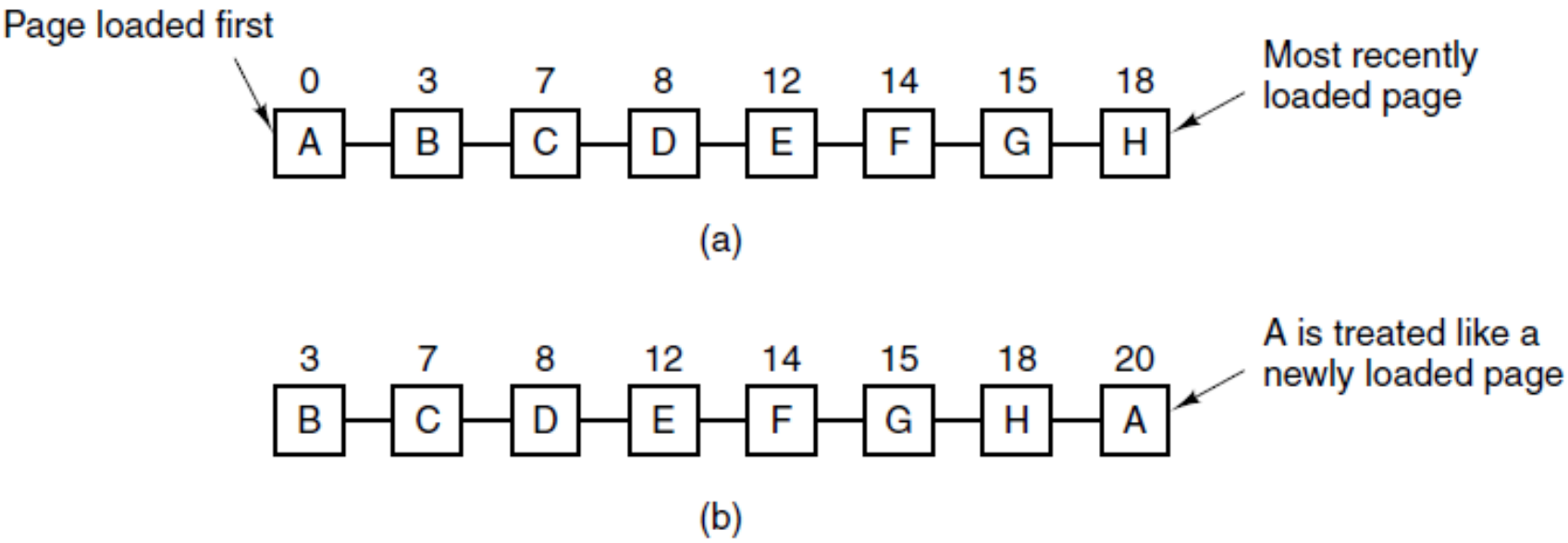


- System maintains a list of all pages currently in memory, with the most recent arrival at the tail and the least recent arrival at the head.
- At page fault, the page at the head is removed and the new page added to the tail of the list.
- Not used in real systems.

Second-Chance Algorithm

- System maintains a list of all pages currently in memory, with the most recent arrival at the tail and the least recent arrival at the head like FIFO algorithm.
- At page fault, the page at the head's R bit is checked.
  - If R bit is zero, the page is removed and the new page added to the tail of the list.
  - If R bit is one, the page is removed from the head and added back at the tail with R bit cleared and load time modified and the search continues.

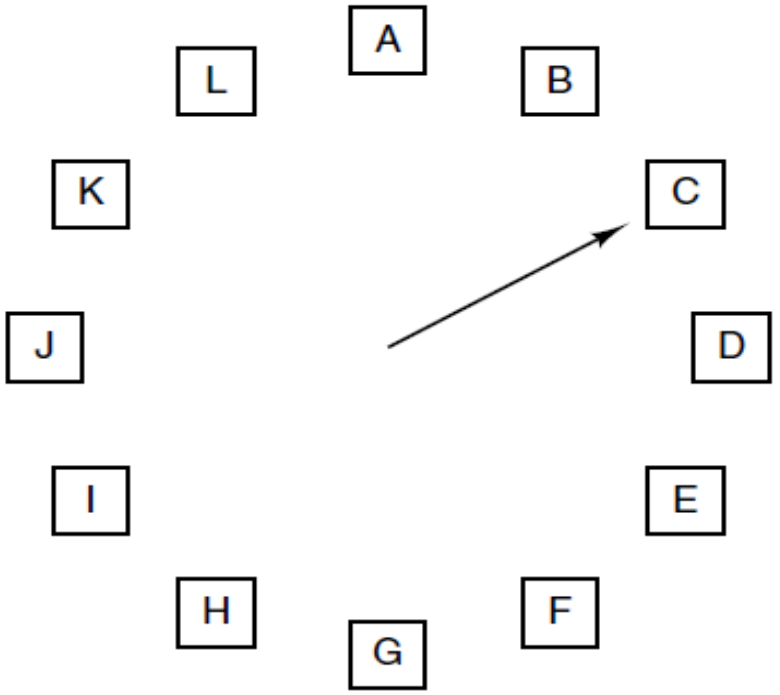
Second-Chance Algorithm



Operation of second chance. (a) Pages sorted in FIFO order. (b) Page list if a page fault occurs at time 20 and A has its R bit set.

Clock Page Replacement Algorithm

- Keeps all the pages in memory on a circular list in the form of a clock.



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

R = 0: Evict the page

R = 1: Clear R and advance hand

Least Recently Used (LRU) Algorithm

- System is equipped with a 64-bit counter in hardware.
- At every instruction execution the counter value is incremented.
- At every page reference current counter value is recorded in the corresponding page table entry.
- At page fault, the page with the lowest counter value is removed.

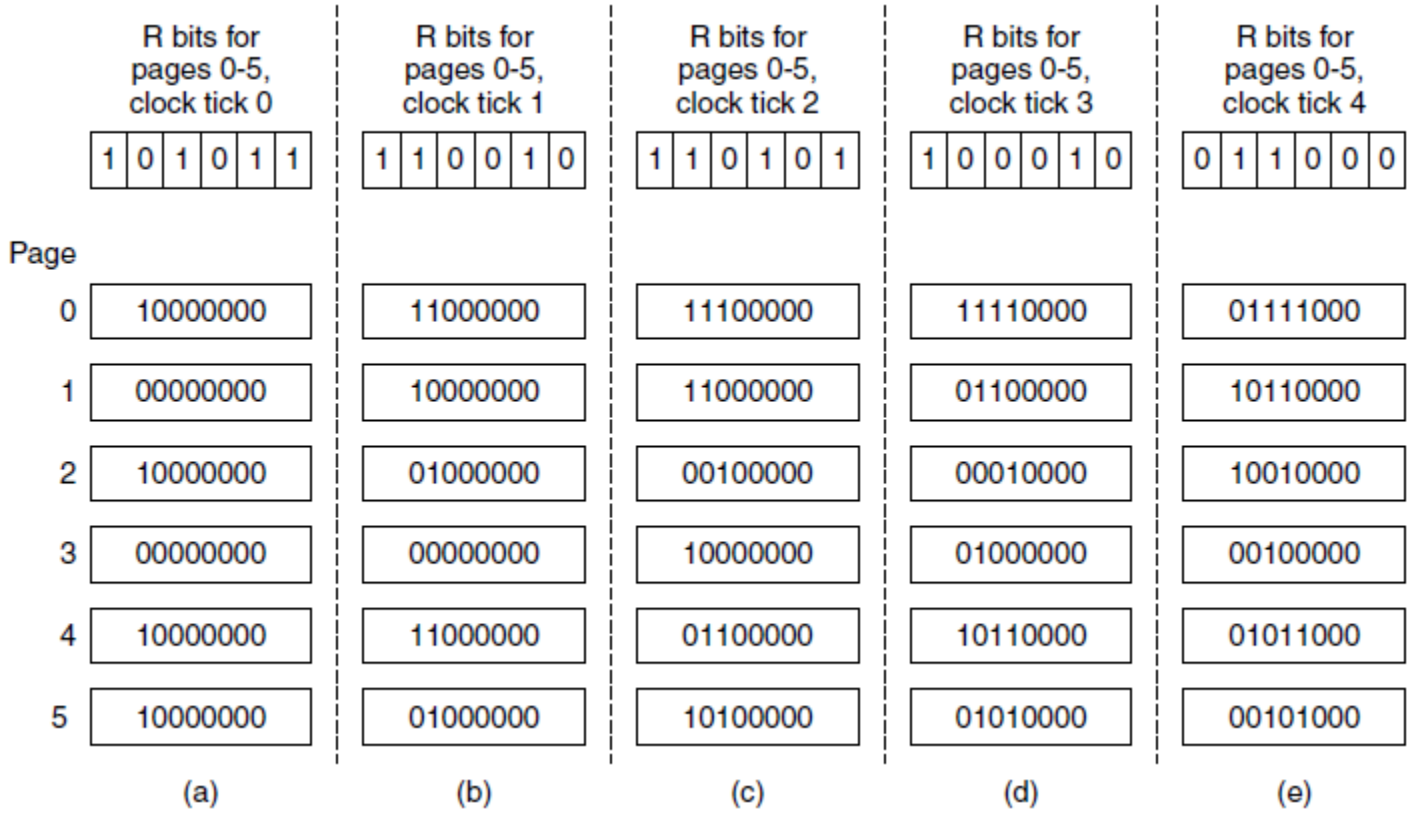
Not Frequently Used (NFU) Algorithm

- System maintains a software counter for each page and initializes it with zero.
- At every clock interrupt, all the pages in the memory are checked and its R bit (0 or 1), is added to its counter.
- At page fault, the page with the lowest counter value is removed.

Simulating LRU in Software

- System keeps a software counter for each page in the memory, initializes it to zero.
- At every clock interrupt, each page counter is shifted 1 bit right and a 1 is added at the left (most significant bit) if the page has been referenced in this clock tick. This is called **aging**.
- At page fault, the page with the lowest counter value is removed.

Simulating LRU in Software

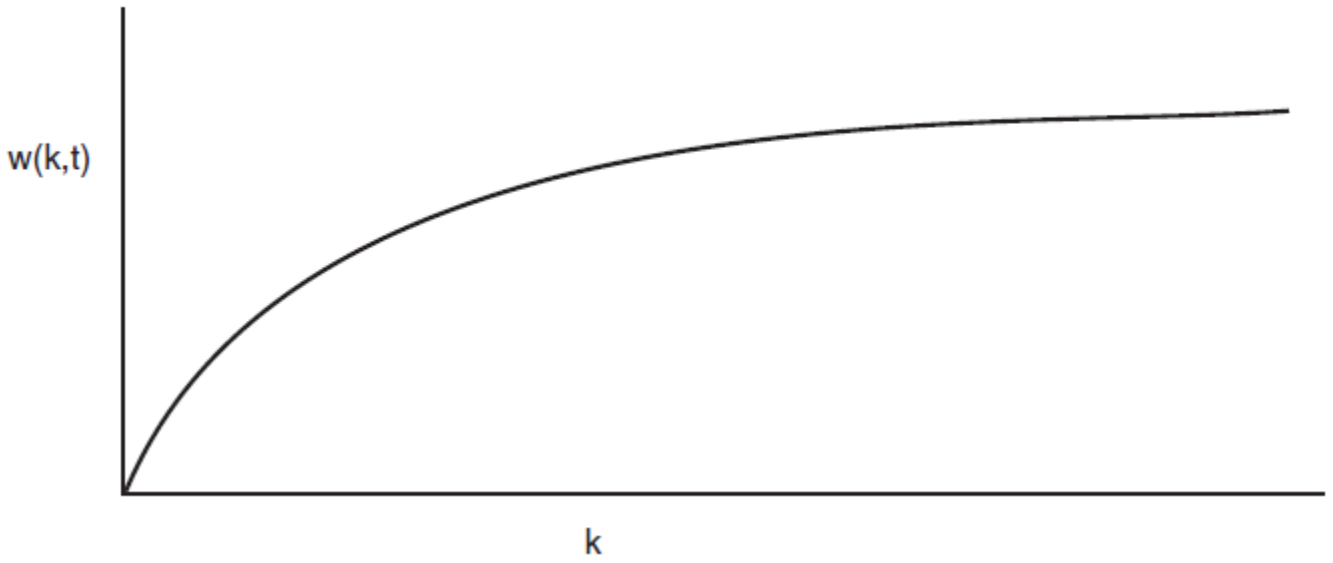


The aging algorithm simulates LRU in software. Shown are six pages for five clock ticks. The five clock ticks are represented by (a) to (e).

Working Set Algorithm

- The set of pages that a process is currently using is its working set.
- If the entire working set is in memory, the process will run without causing many page faults.
- A program causing page faults every few instructions is said to be **thrashing**.
- Working set algorithm aims to minimize thrashing.
- **Prepaging** the working set into the memory may cause less thrashing compared to **demand paging**.
- Prepaging is less practical than demand paging.

Working Set Algorithm



The working set is the set of pages used by the *k* most recent memory references. The function *w(k, t)* is the size of the working set at time *t*.

Working Set Algorithm

- System can maintain a shift register of length k, with every memory reference shifting the register left one position and inserting the most recently referenced page number on the right.
- The set of all k page numbers in the shift register would be the working set.
- At page fault, the page not in the shift register can be removed.
- Maintaining the shift register and processing it at a page fault would both be expensive, i.e., never used.

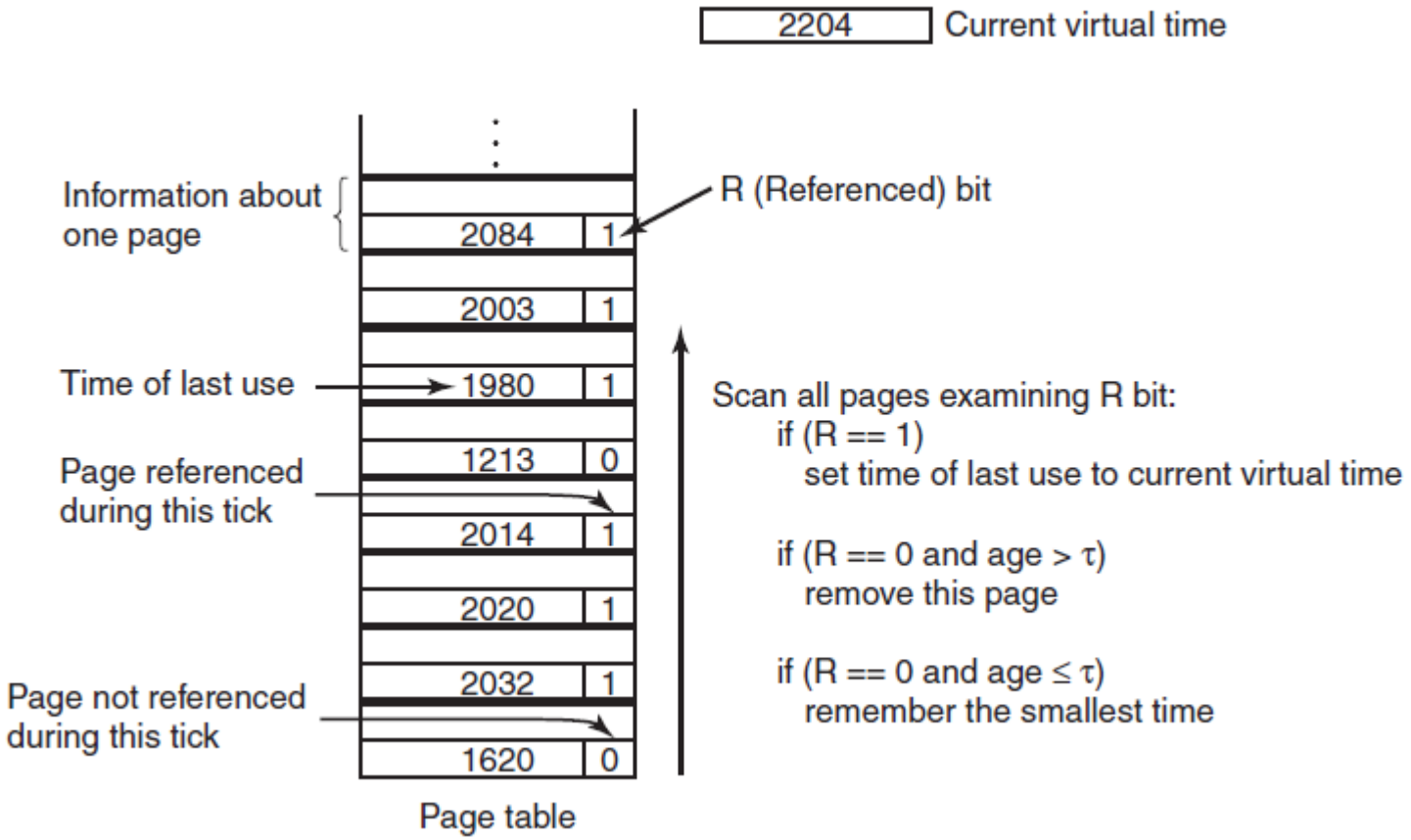
Working Set Algorithm with Time of Last Use Approximation

- **Current virtual time** of each process is tracked by tracking its actual cpu usage from the start.
- Current virtual time is used to approximate the **time of last use** of each page of the process and recorded in the page table entry.
- At page fault, page table entries are scanned to evict one.
- If R bit is 1, the current virtual time of the process is written as the time of last use of the page and proceeds.

Working Set Algorithm with Time of Last Use Approximation

- If R bit is 0, the **age** of the last use of the page is computed by subtracting its time of last use from current virtual time.
  - If the age is greater than τ, the page is evicted and the scan continues to update other page entries.
  - If the age is less than or equal to τ, the page is kept and the page with the greatest age is kept tracked. If no page with the age greater than τ is found in the whole scan, the page with the greatest age is evicted.
- If no page with R bit 0 is found in the whole scan, a page is picked randomly to evict.

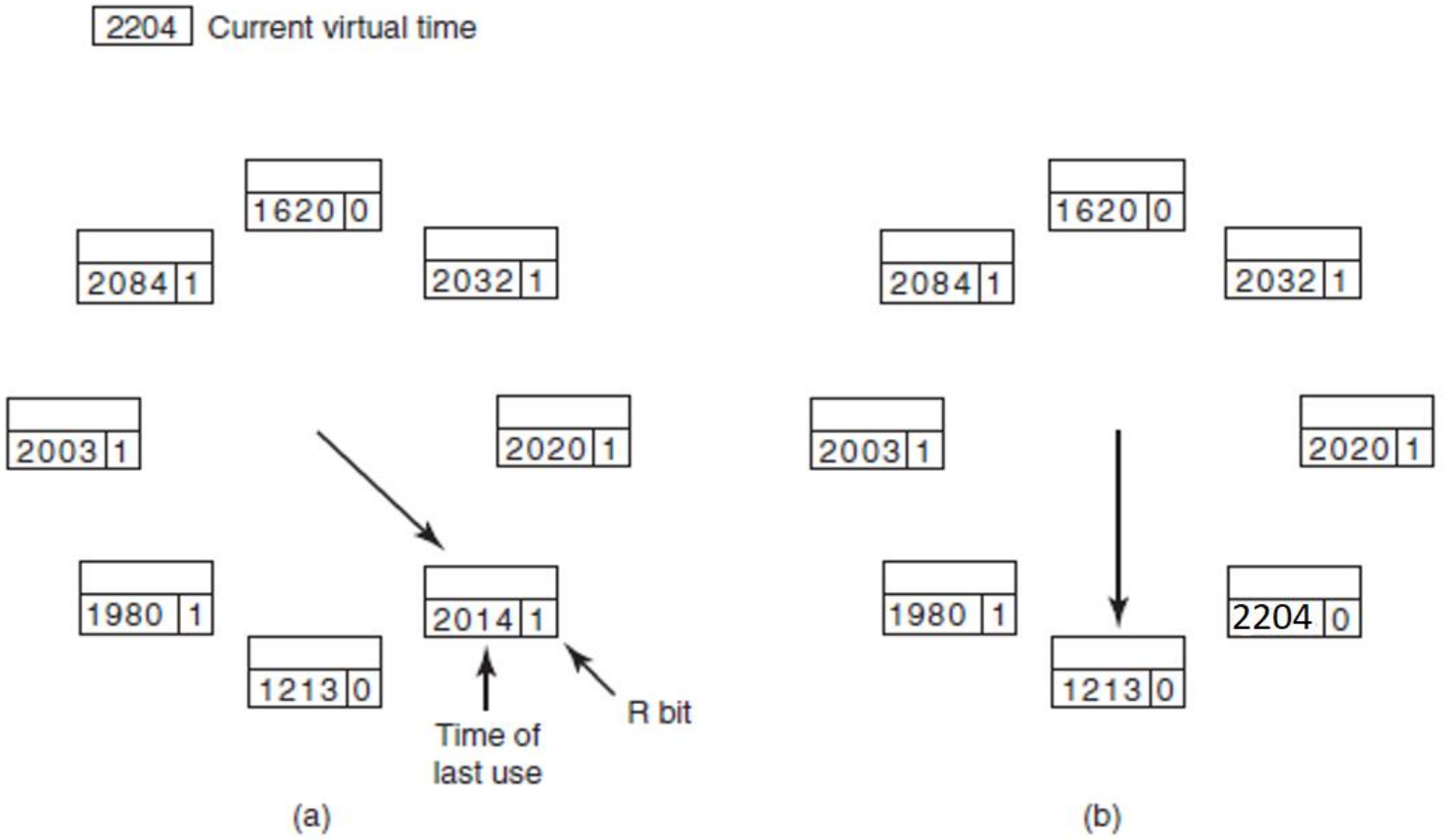
Working Set Algorithm with Time of Last Use Approximation



WSClock Algorithm

- Combined **working set** and **clock** algorithms.
- At page fault, the page table entry at clock hand is checked.
- If its R bit is 1, it is set to 0 and the current virtual time of the process is written as the time of last use of the page and proceeds by advancing the clock hand.

WSClock Algorithm

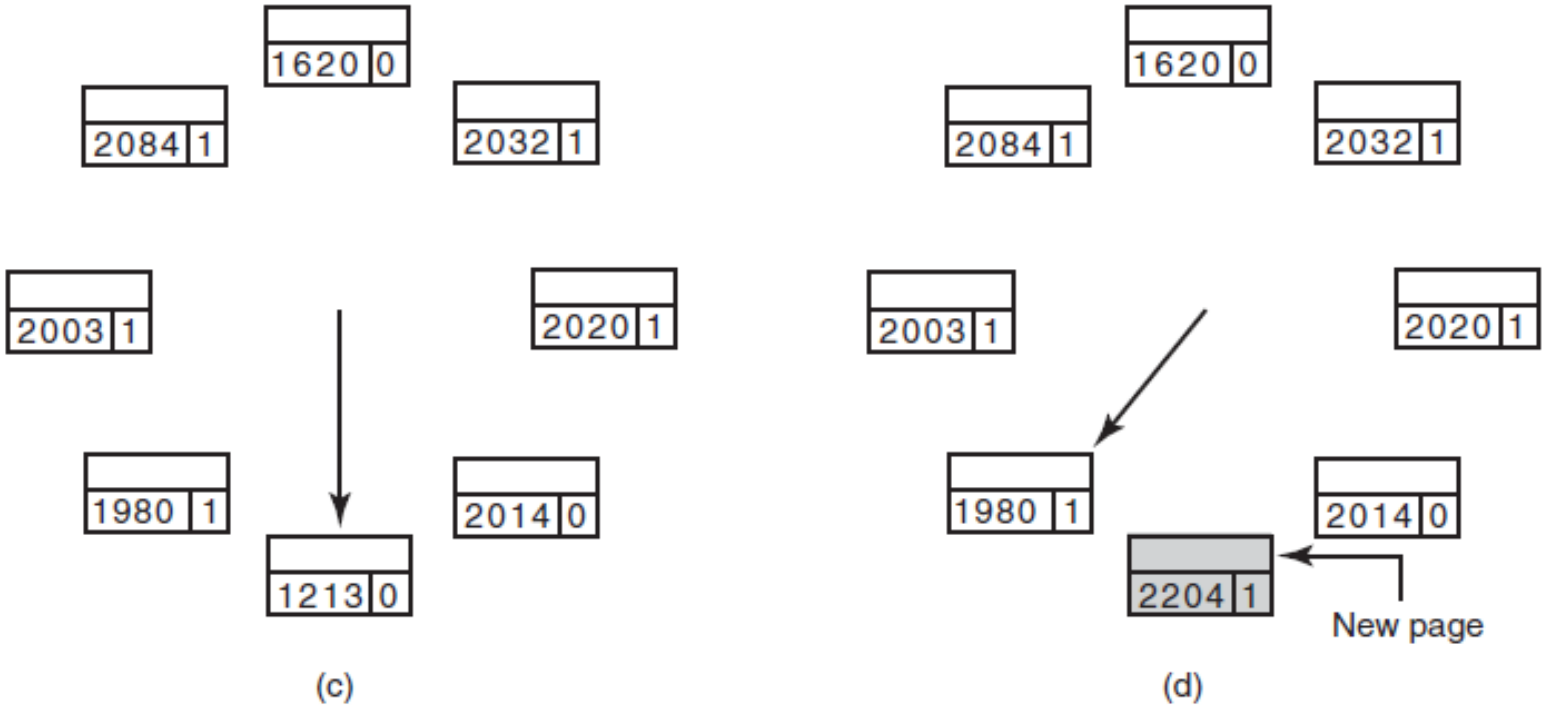


Operation of the WSClock algorithm. (a) and (b) give an example of what happens when R = 1.

WSClock Algorithm

- If R bit is 0, and the age is greater than τ, it checks the M bit
  - If M bit is 0, the page is evicted and the new page is loaded and its last time use is set to current virtual time.
  - If the M bit is 1, the page is not evicted but a disk write for the page is scheduled.
  - Clock hand is moved to the next page in both cases.
- If no page has been evicted but some pages have scheduled for disk write, clock hand keeps moving to find a clean page to evict.
- If no page has been scheduled for disk write, clock hand keeps moving to find any clean page to evict.
- If no clean page has been found the page at clock hand is evicted.

WSClock Algorithm



Operation of the WSClock algorithm.
(c) and (d) give an example of R = 0.

WSClock Algorithm

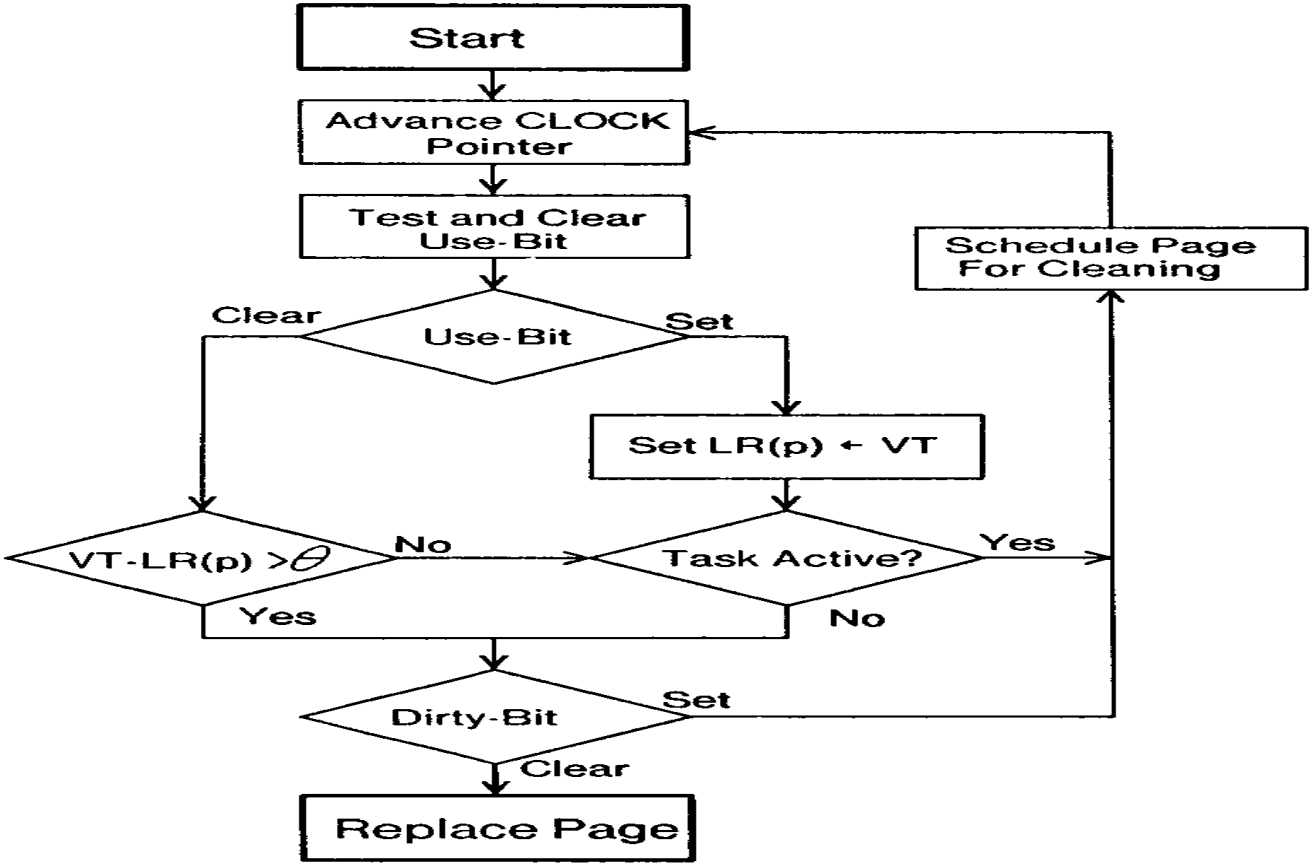


Figure 2. WSCLOCK Replacement Algorithm

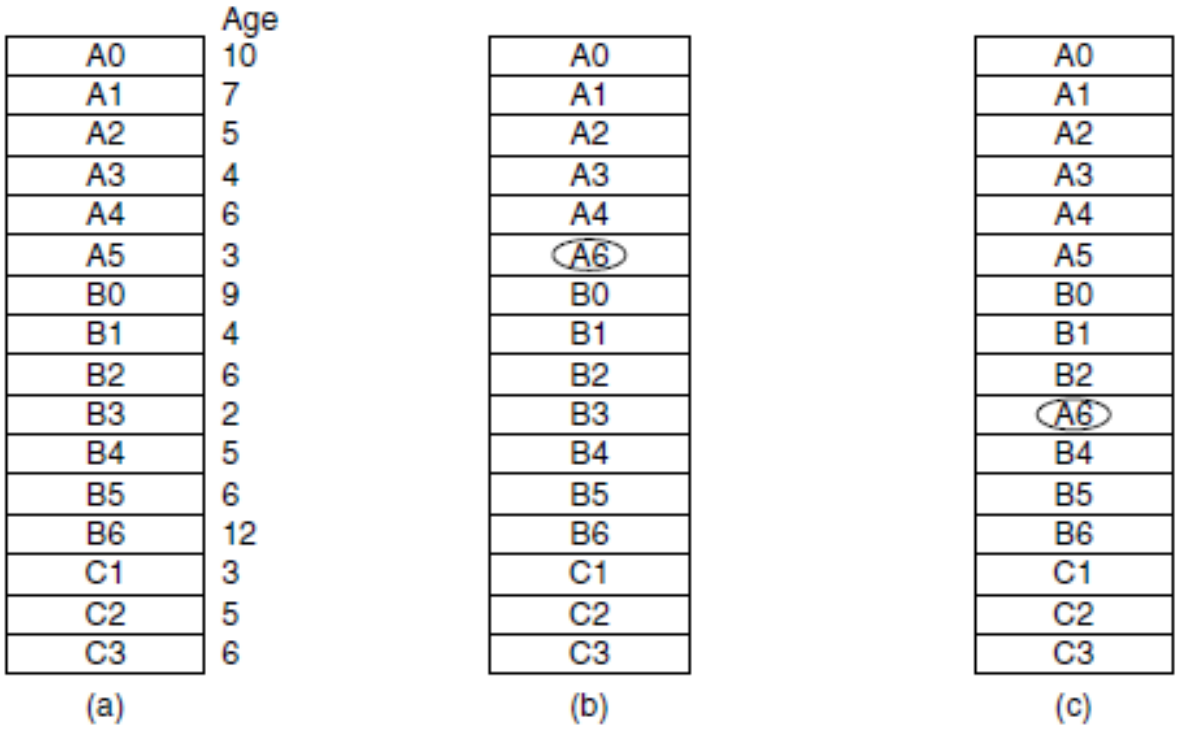
WSClock (Carr and Hennessey, 1981)

Summary of Page Replacement Algorithms

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude approximation of LRU
FIFO (First-In, First-Out)	Might throw out important pages
Second, chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Page replacement algorithms discussed in the text.

Local versus Global Allocation Policies



Local versus global page replacement.
(a) Original configuration. (b) Local page replacement.
(c) Global page replacement.

Page Fault

1. The hardware traps to kernel, saving program counter on stack.
2. Assembly code routine started to save general registers and other volatile info
3. system discovers page fault has occurred, tries to discover which virtual page needed
4. Once virtual address caused fault is known, system checks to see if address valid and the protection consistent with access

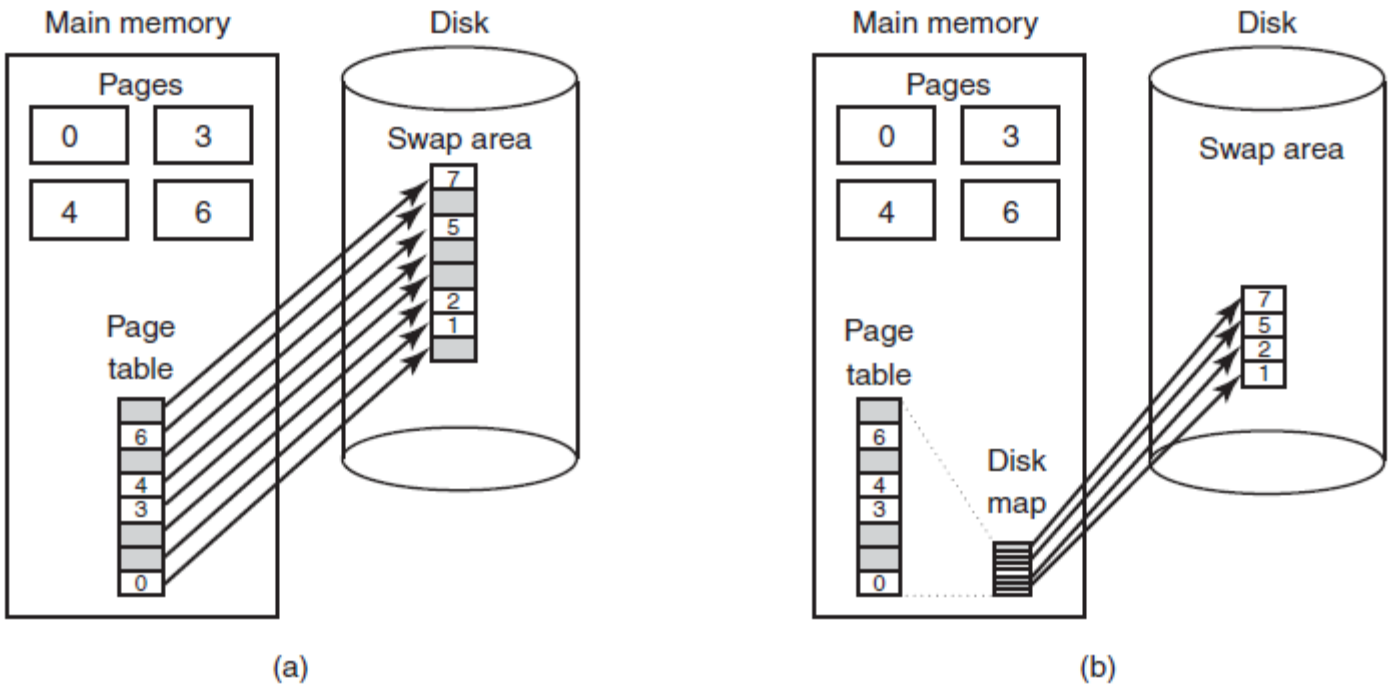
Page Fault

5. If frame selected dirty, page is scheduled for transfer to disk, context switch takes place, suspending faulting process
6. As soon as frame clean, operating system looks up disk address where needed page is, schedules disk operation to bring it in.
7. When disk interrupt indicates page has arrived, tables updated to reflect position, and frame marked as being in normal state.

Page Fault

8. Faulting instruction backed up to state it had when it began and program counter is reset
9. Faulting process is scheduled, operating system returns to routine that called it.
10. Routine reloads registers and other state information, returns to user space to continue execution

Backing Store



(a) Paging to a static swap area.
(b) Backing up pages dynamically.

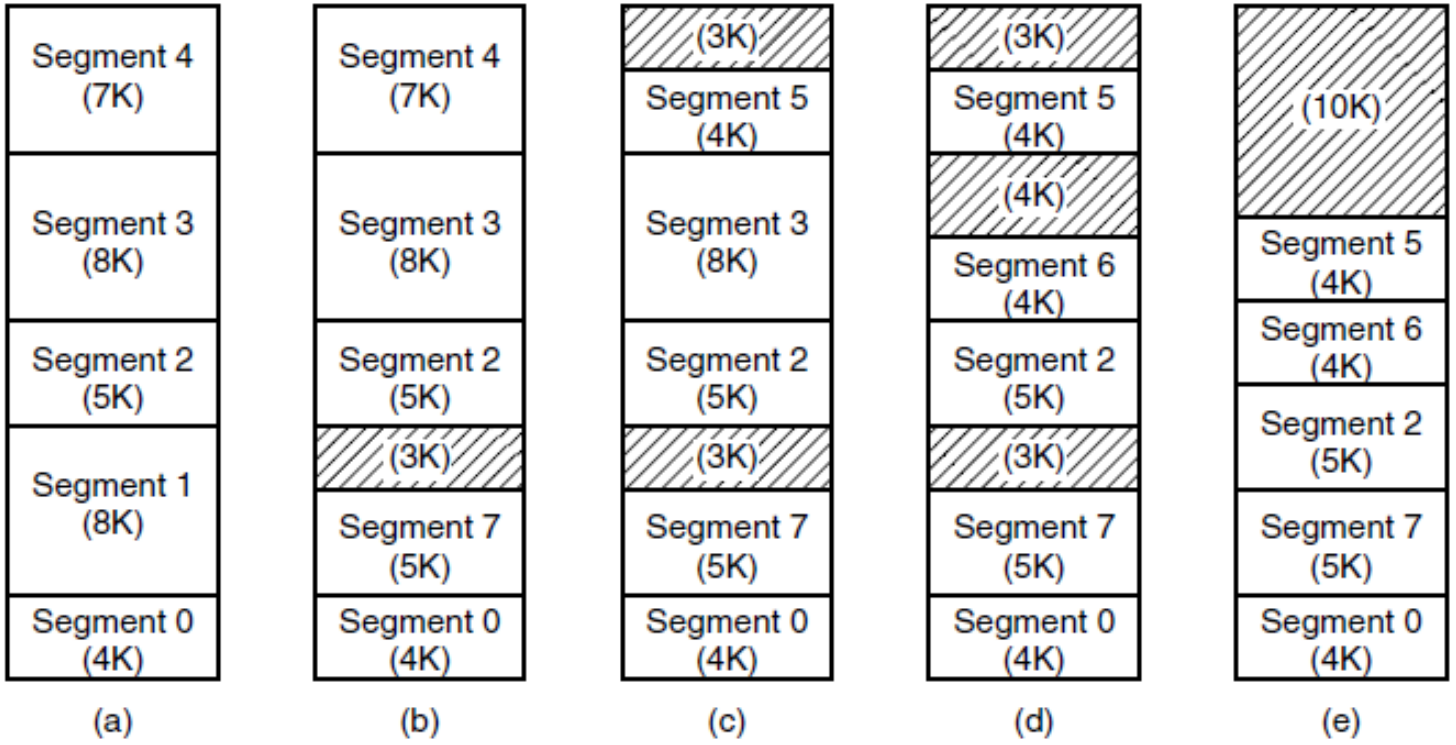
Segmentation

- Paging suffers from **internal fragmentation**.
- The address space of a process has logical divisions as follows:
    1. Code
    2. Data
    3. Constants
    4. Stack
    5. Heap
- Instead of using a single and flat address space, a process can use **multiple segmented address spaces** (called **segments**); one segment for each logical division.

Segmentation

- It is not necessary to place all the segments of process together in the main memory, rather, they can be placed independently anywhere in the memory.
- Different segments of a process can be of different sizes.
- Each segment can grow and shrink independently without bumping into each other.
- No internal fragmentation but **external fragmentation** is unavoidable.

Segmentation

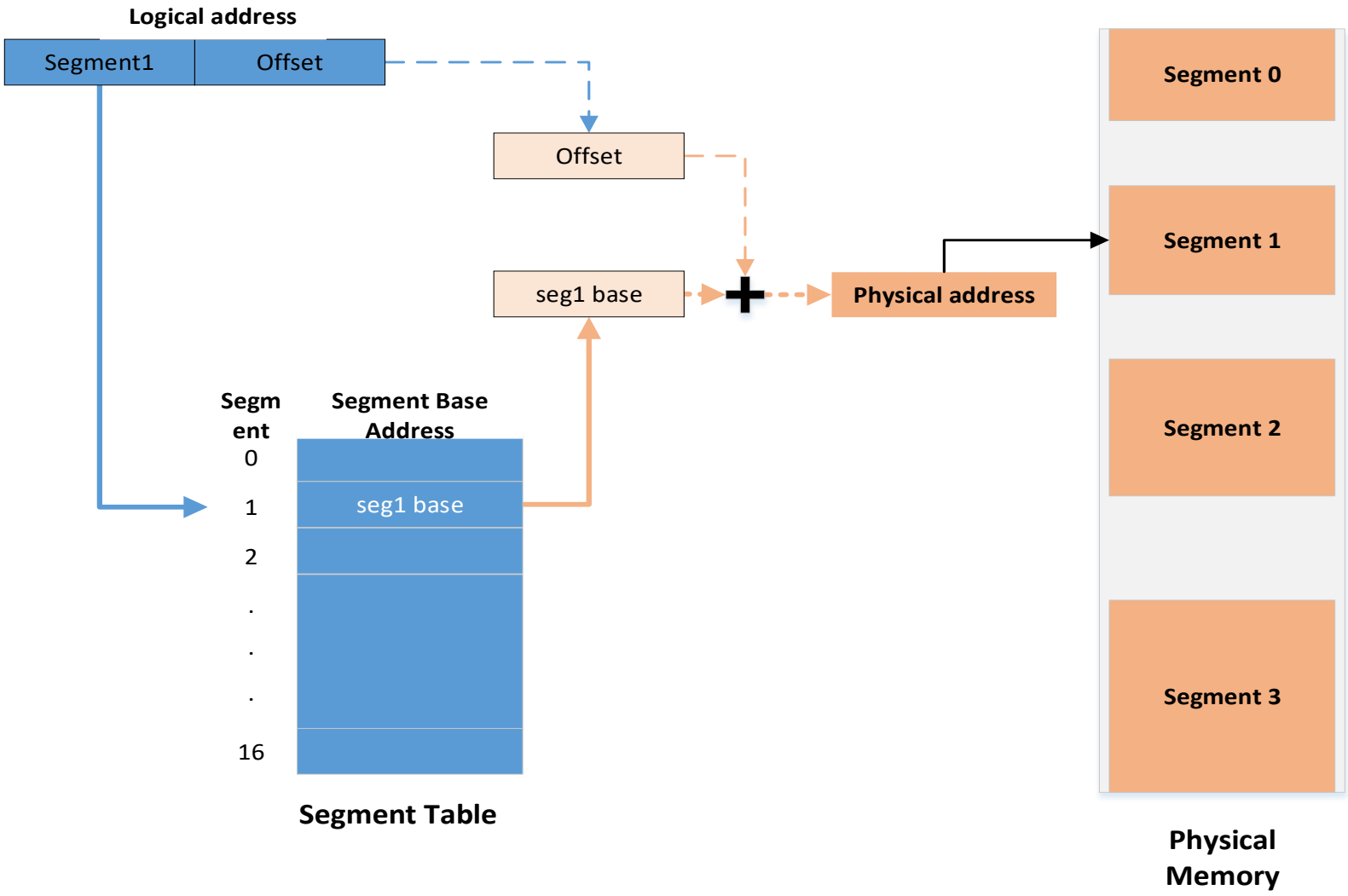


(a)-(d) Development of checkerboarding.
(e) Removal of the checkerboarding by compaction.

Segmentation

- Each **logical address** of a process will have a **segment number** and an **offset**.
- Each process will have a **segment table** with **segment descriptors** that will keep track where each segment starts.
- Address translation in a segmentation mechanism is very simple.
  - Segmentation number is used as the index of the segmentation table to find the start of the segment.
  - Once the start of the segment is found that is added with the offset to compute the physical address.

Segmentation

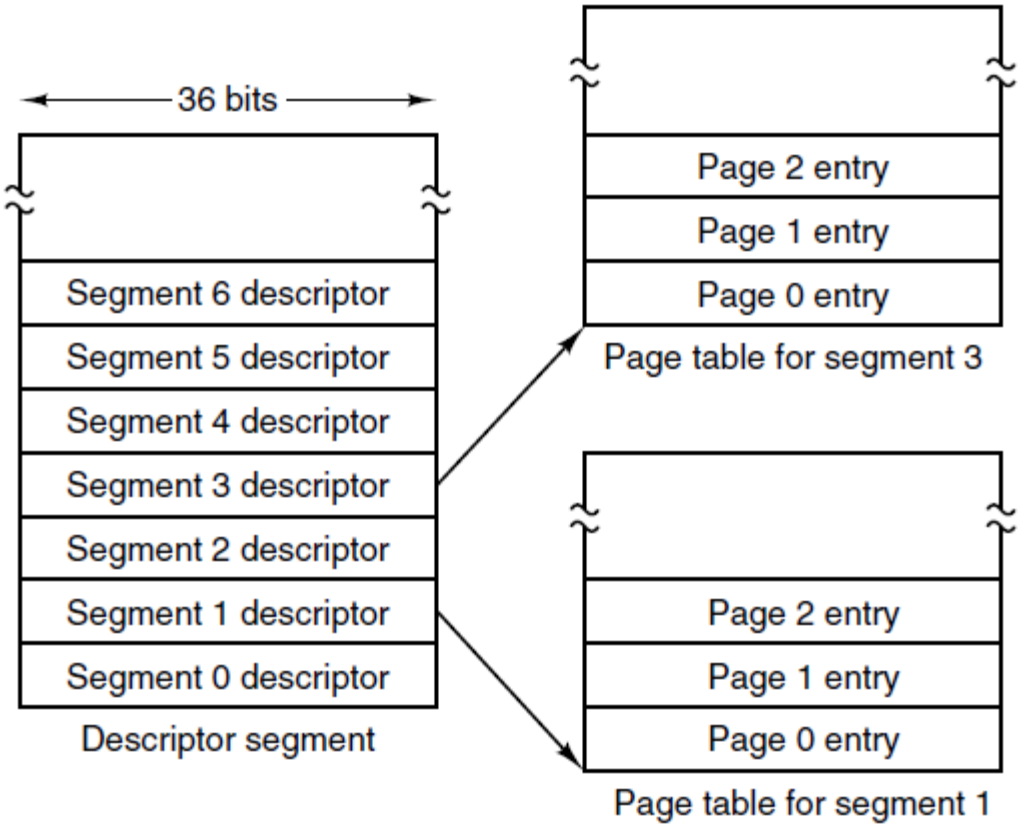


Segmentation

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

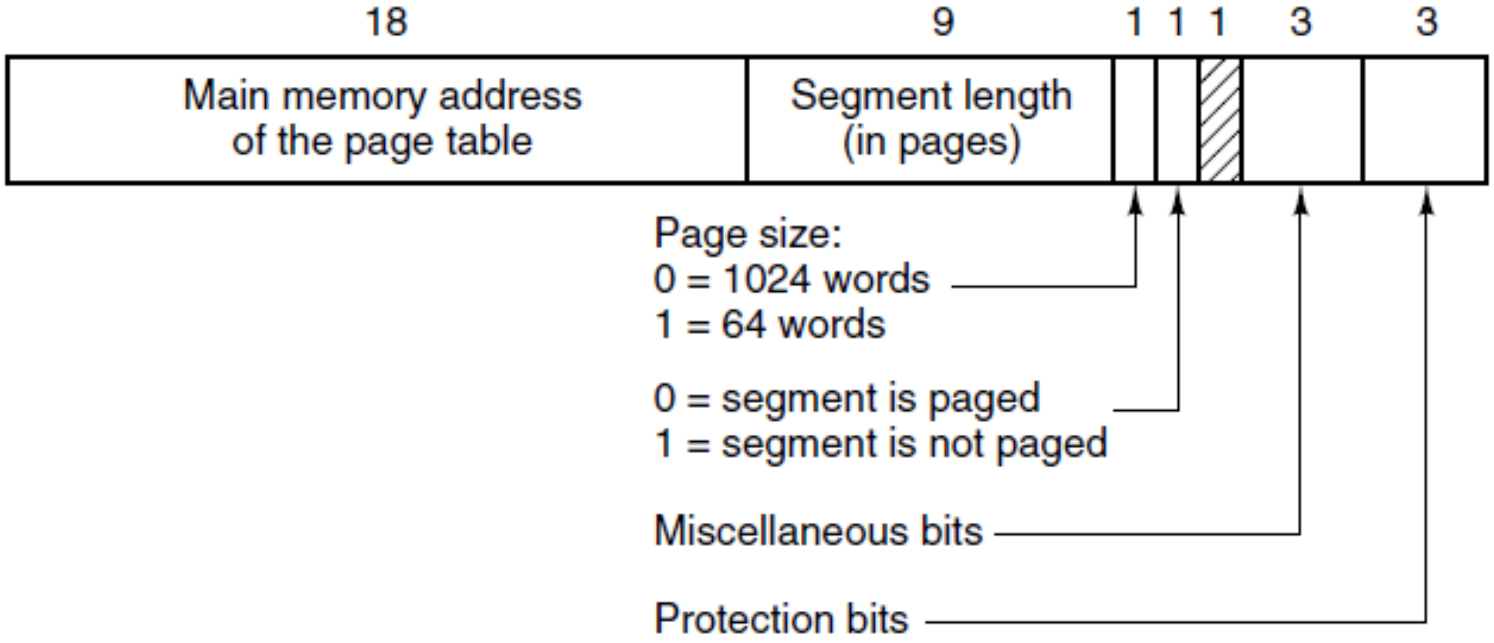
Comparison of paging and segmentation

Segmentation with Paging: MULTICS



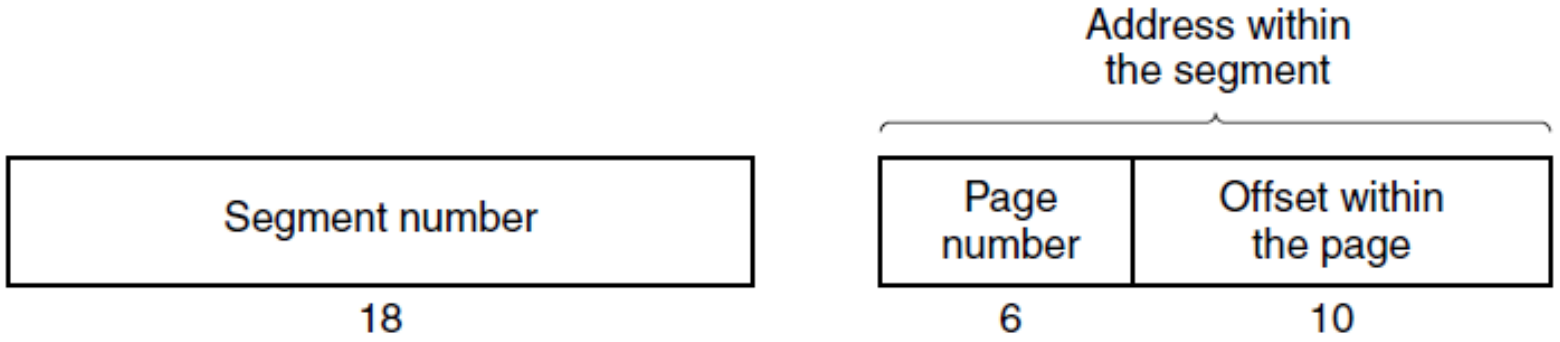
The MULTICS virtual memory. (a) The descriptor segment pointed to the page tables.

Segmentation with Paging: MULTICS



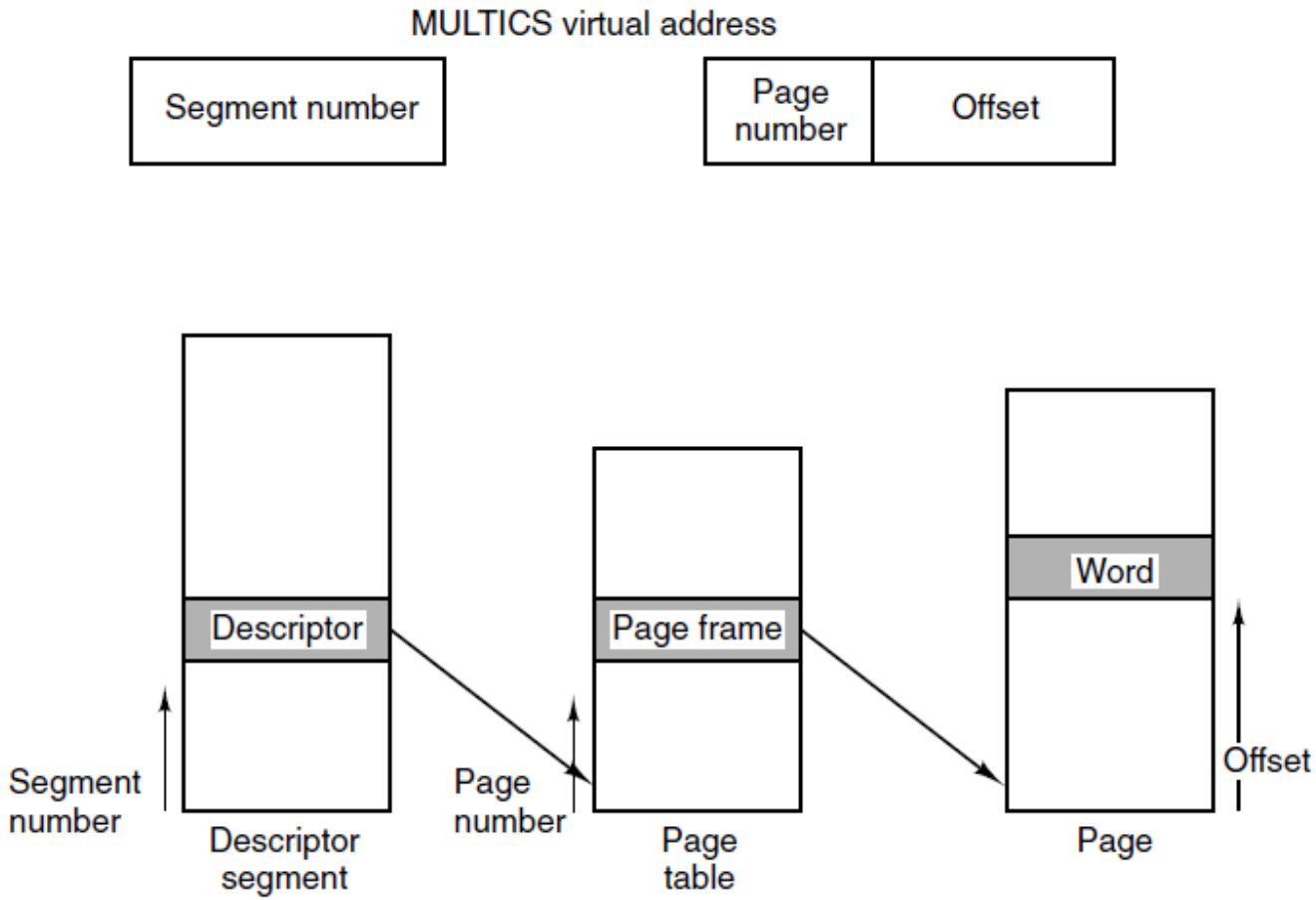
The MULTICS virtual memory. (b) A segment descriptor. The numbers are the field lengths.

Segmentation with Paging: MULTICS



A 34-bit MULTICS virtual address.

Segmentation with Paging: MULTICS



Conversion of a two-part MULTICS address into a main memory address.

Segmentation with Paging: MULTICS

Comparison field					Is this entry used?
Segment number	Virtual page	Page frame	Protection	Age	
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

A simplified version of the MULTICS TLB. The existence of two page sizes made the actual TLB more complicated.

Summary

- Address Space
- Swapping
- Free Memory Management
- Memory Allocation Algorithms
- External Fragmentation and Compaction
- Virtual Memory and Paging
- Page Table
- Page Replacement Algorithms
- Page Fault
- Segmentation

Next

File System

- File Abstraction
- File Concepts
- Directory Concepts
- Disk Partitions and File System Layout
- Disk Block Allocation
- Free Disk Block Management
- File System Performance