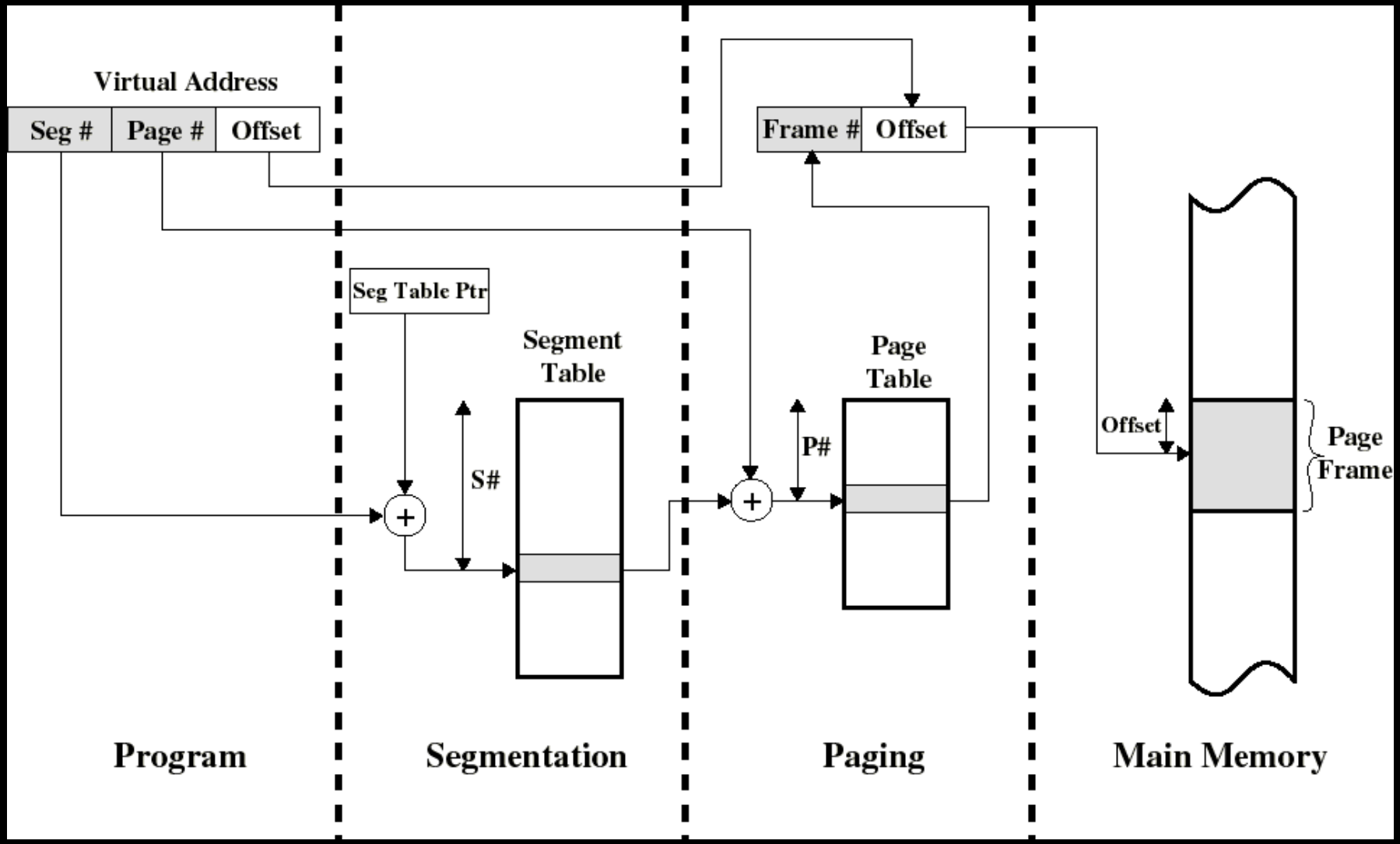
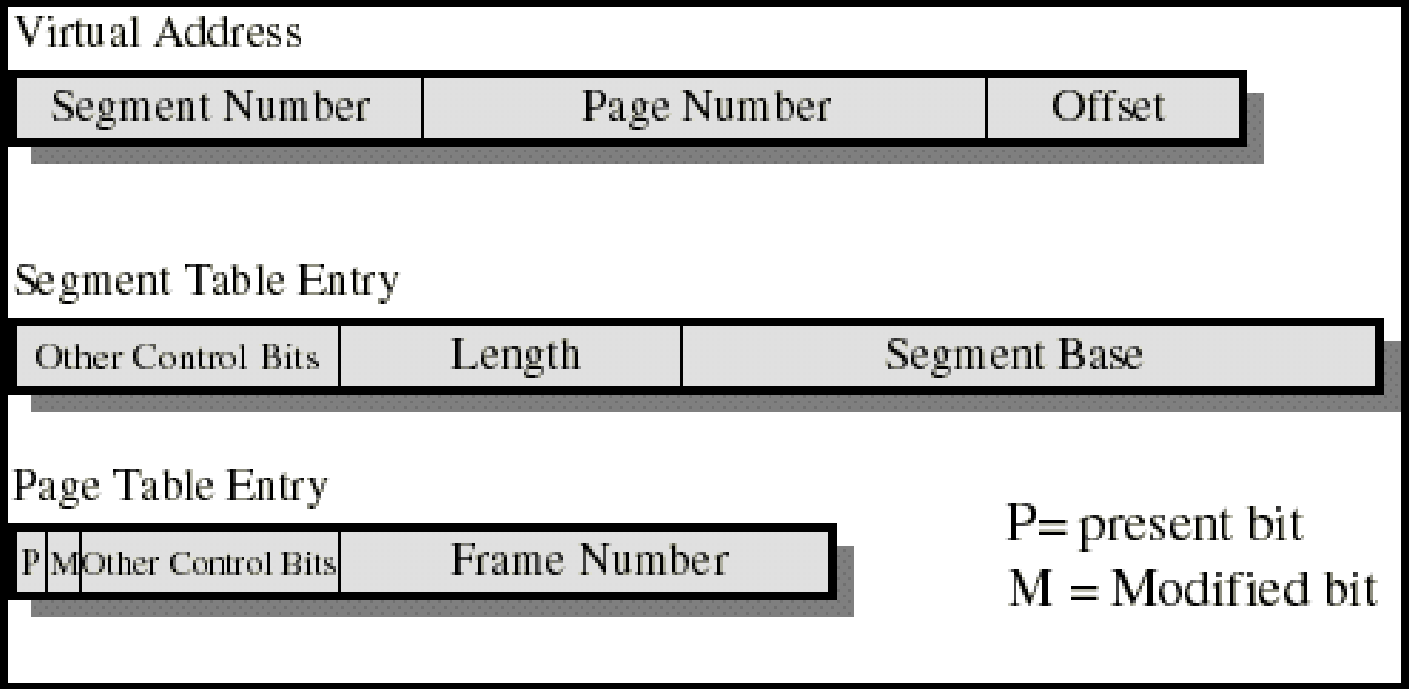
Memory Management

Memory Management – Paged Segmentation



Memory Management – Paged Segmentation

- The Segment Base is the physical address of the page table of that segment
- Present and modified bits are present only in page table entry
- Protection and sharing info most naturally resides in segment table entry



Question & Answer

Compare the main memory organization schemes between Contiguous Memory Allocation, Pure Segmentation, and Pure Paging.

Contiguous Memory Allocation scheme suffers from external fragmentation as address spaces are allocated contiguously and holes develop as old processes dies and new processes are initiated. It also does not allow processes to share code.

Pure Segmentation also suffers from external fragmentation, however, it enables processes to share code, for instance, two different processes could share a code segment but have distinct data segments.

Pure Paging does not suffer from external fragmentation, but instead, it suffers from internal fragmentation. Processes are allocated in pages and if a page is not completely utilized, it results in internal fragmentation and a corresponding wastage of space occur. Paging also enables processes to share code at the granularity of pages.

Question & Answer

On a system with paging, a process cannot access memory that it does not own; why? How could the operating system allow access to other memory? Why should it or should it not?

An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process. There is no way for a process to refer to a page it does not own because the page will not be in the page table. To allow such access, an operating system simply needs to allow entries for non-process memory to be added to the process's page table. This is useful when two or more processes need to exchange data. They just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient inter-process communication.

Question & Answer

Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual addresses to physical addresses.

Paging requires more memory overhead to maintain the translation structures. Segmentation requires just two registers per segment: one to maintain the base of the segment and the other to maintain the extent of the segment. Paging on the other hand requires one entry per page, and this entry provides the physical address in which the page is located.

Question & Answer

Why are page sizes always power of 2? Consider a logical address space of eight pages of 1024 words each, mapped onto a physical memory of 32 frames.

a) How many bits are there in the logical address?
b) How many bits are there in the physical address?

All addresses are binary and are divided into page or page frame number and offset. By making the page size a power of 2, the page number, the page frame number and the offset can be determined by looking at particular bits in the address. No mathematical calculation is required. The physical address can be generated by concatenating the offset to the frame number.

Question & Answer

A computer has a 32-bit virtual address space and 1024-byte pages. A page table entry takes 4 bytes. A multilevel page table is used because each table must be contained with a page. How many levels are required?

A page table can contain 1024/4 = 256 = 2^8 entries. Given the page size of 2^{10}, 32 – 10 = 22 bits specify the page number. Each level of page table can handle 8 of the 22 bits, so three levels would be required. Tables at two of the levels would have 2^8 entries and the table size at the other level would be only 2^6

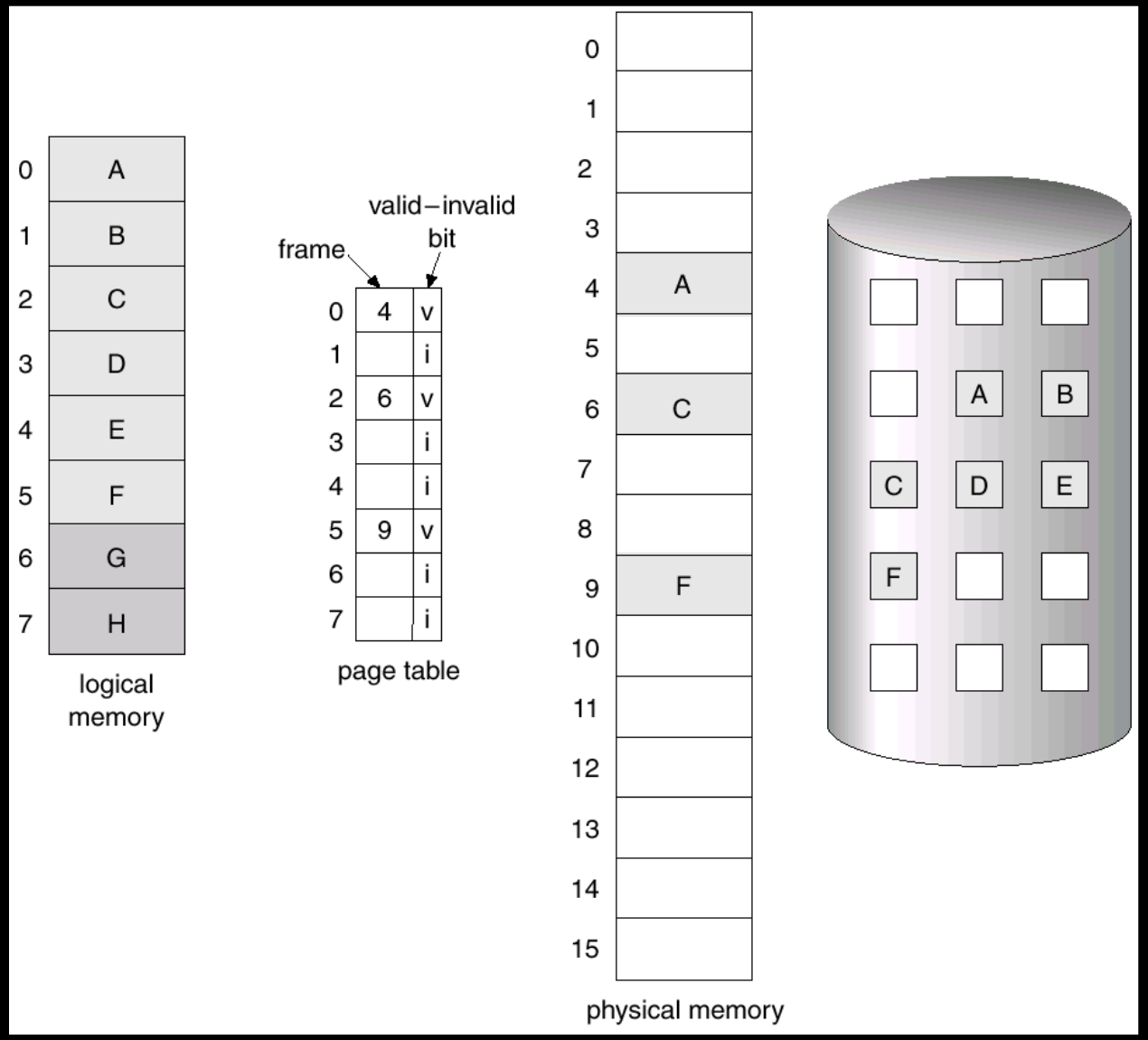
Question & Answer

On a system using paging and segmentation, the virtual address space consist of up to 8 segments where each segment can be up to 2^{29} bytes long. The hardware pages each segment into 256-byte pages. How many bits specify the entire virtual address specify:

(a) Segment number (b) Page Number (c) Offset within page
(d) Entire virtual address

(a) 3, since $8 = 2^3$ so 3 bits are needed to specify the segment number
(b) 21, with $256 = 2^8$ byte pages, a 2^{29}-byte segment can have $2^{29}/2^8 = 2^{21}$ pages. Thus, 21 bits are needed to specify the page number
(c) 8, to specify the offset into the 2^8-byte page, 8 bits are needed
(d) 32, 3+21+8

Memory Management – Virtual Memory



Page Table When Some Pages Are Not in Main Memory

Memory Management – Virtual Memory

If there is ever a reference to a page, first reference will trap to OS $\Rightarrow$ page fault

OS looks at another table to decide:

- Invalid reference $\Rightarrow$ abort.
- Just not in memory.

Get empty frame.

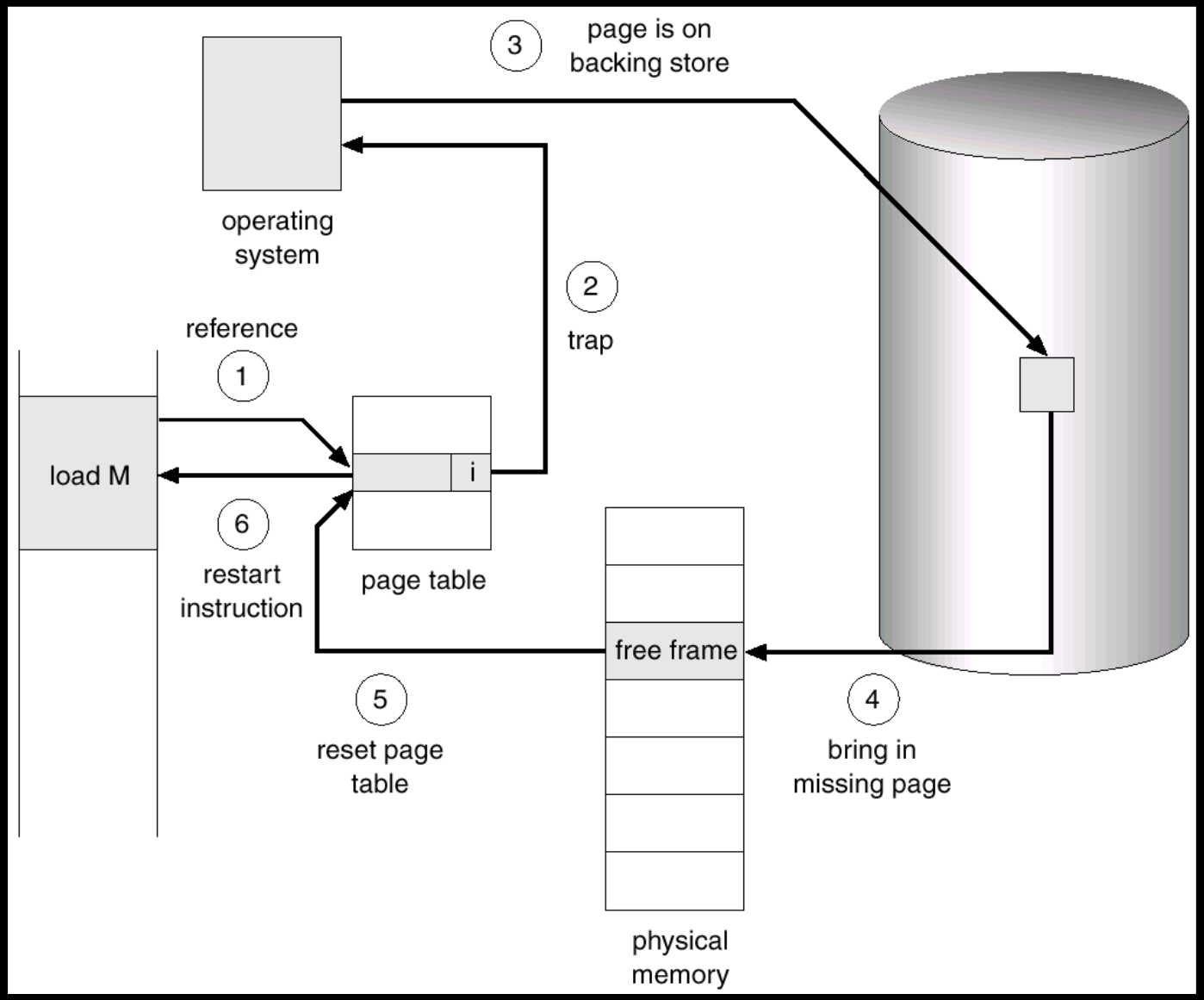
Swap page into frame.

Reset tables, validation bit = 1.

Restart instruction: Least Recently Used

- block move
- auto increment/decrement location

Memory Management – Virtual Memory



Steps in Handling a Page Fault

Memory Management – Virtual Memory

Want lowest page-fault rate.

Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.

Suppose we record following address sequences:

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

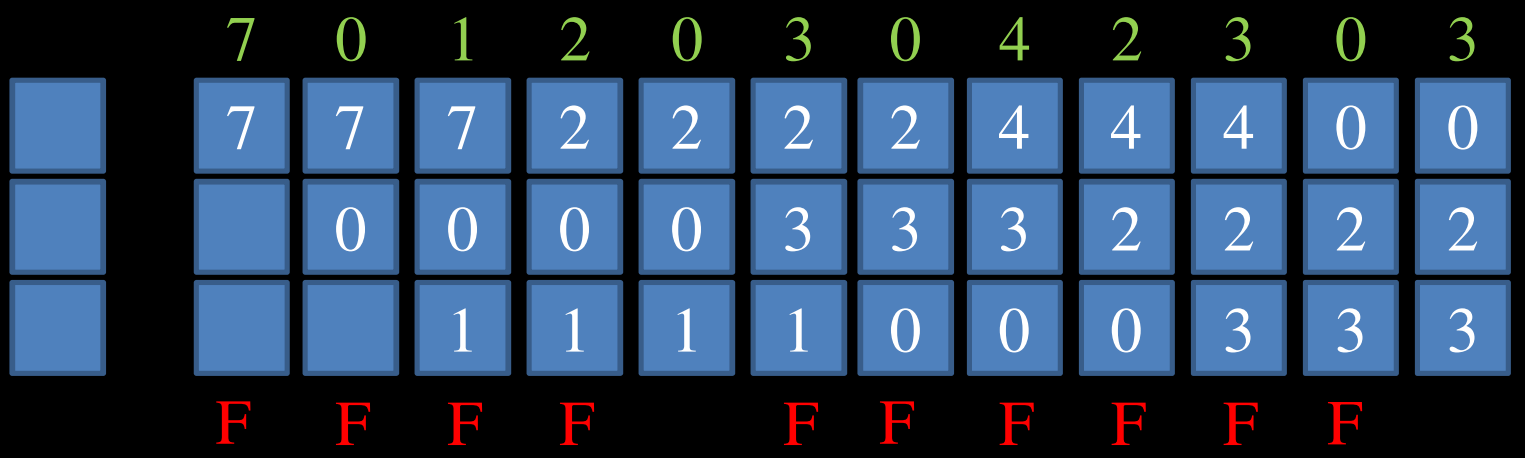
With 100 byte per page, the reference string will be:

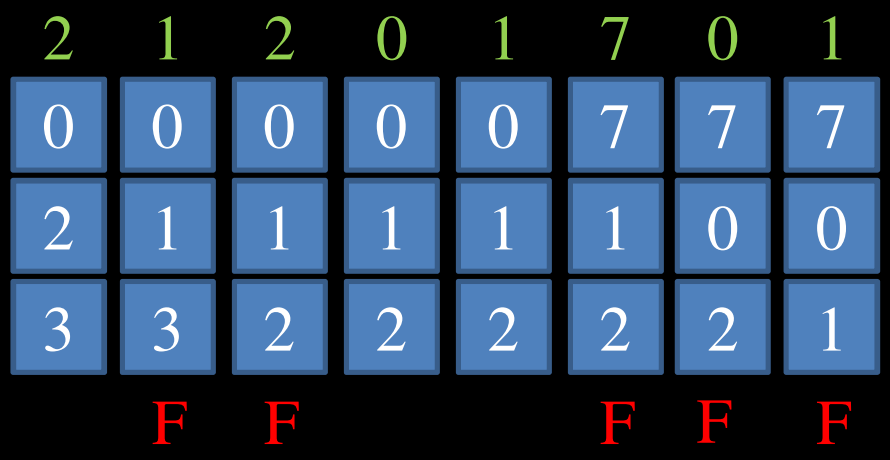
1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1

Memory Management – First In First Out (FIFO)

Reference String:

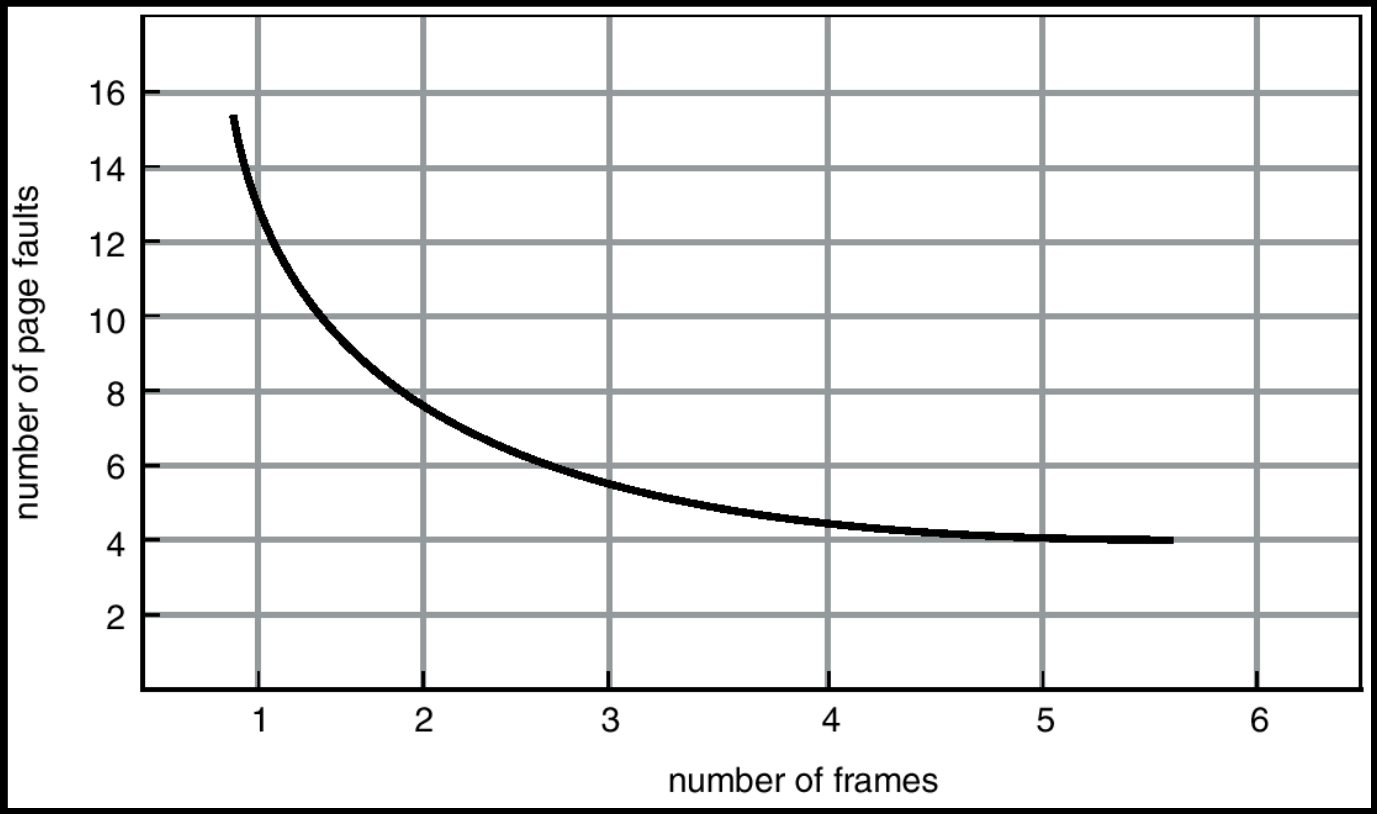
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 Frame Size: 03





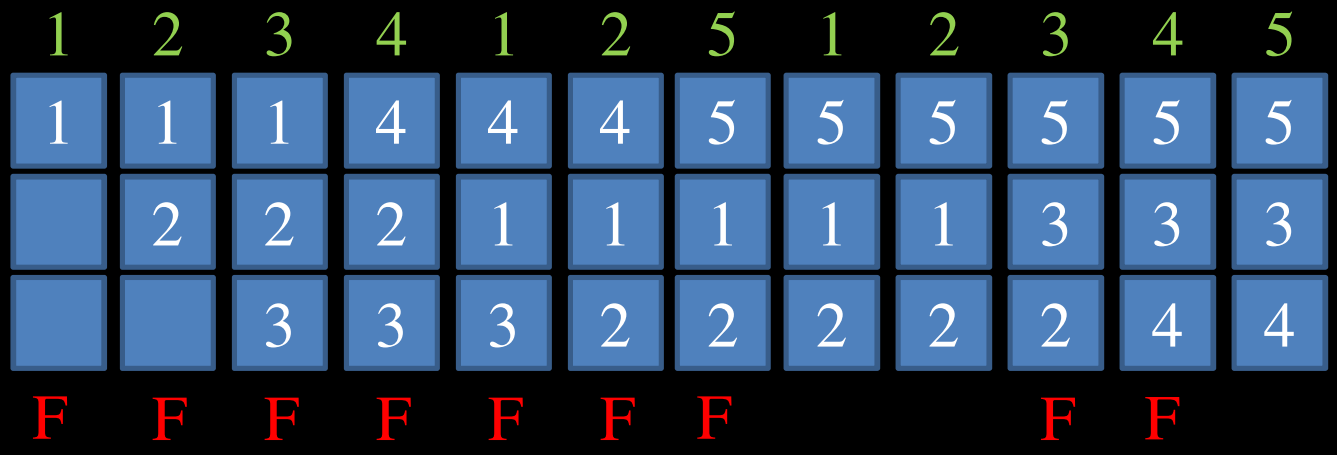
Memory Management – Virtual Memory

Graph of Page Faults Versus The Number of Frames

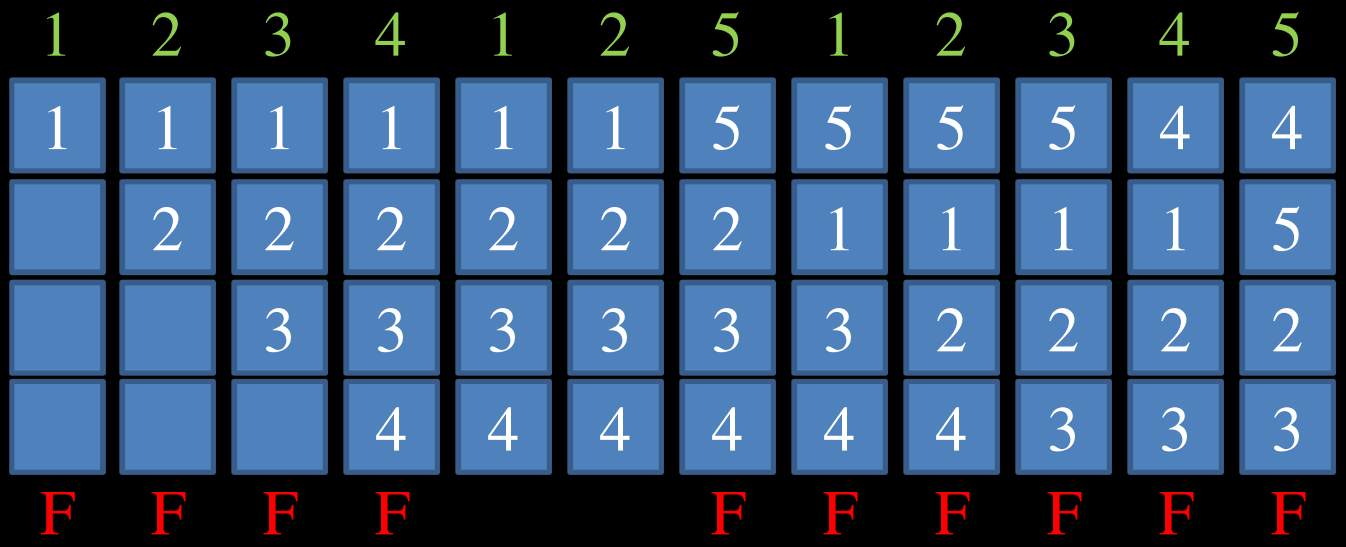


Memory Management – First In First Out (FIFO)

Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



Fault = 09



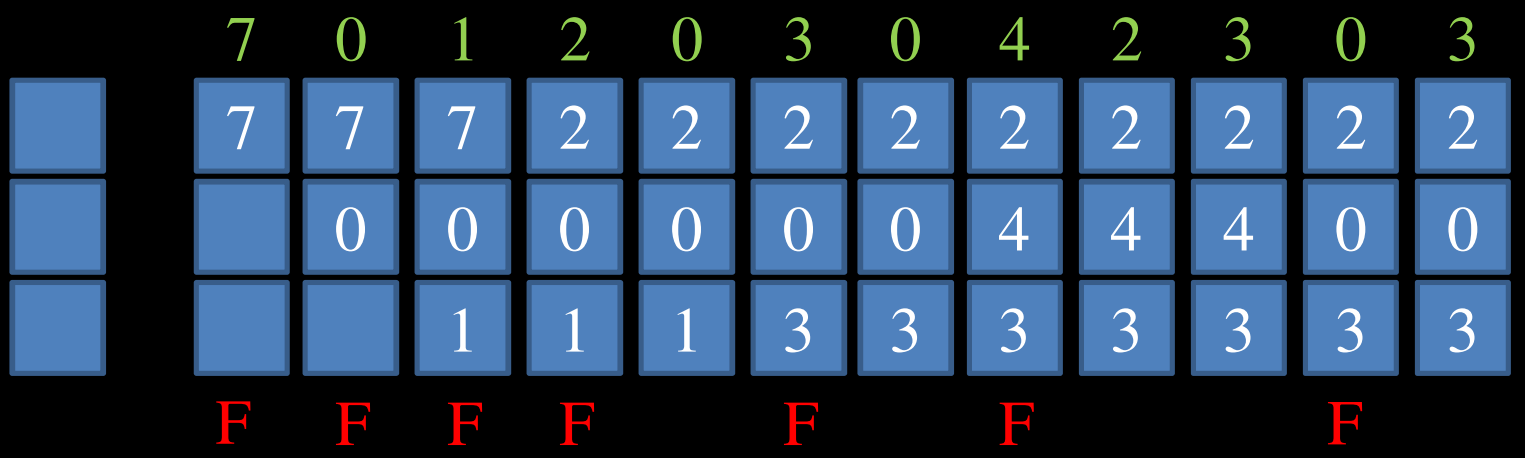
Fault = 10

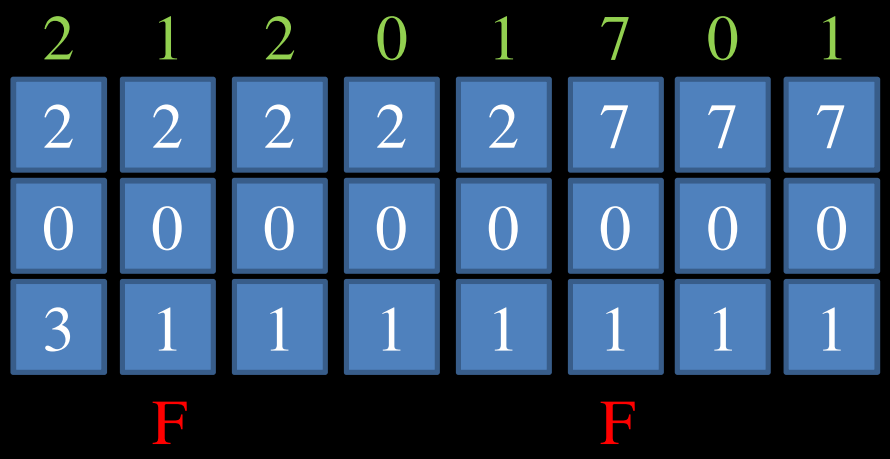
Belady's Anomaly

Memory Management – Optimum Algorithm

Reference String:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 Frame Size: 03





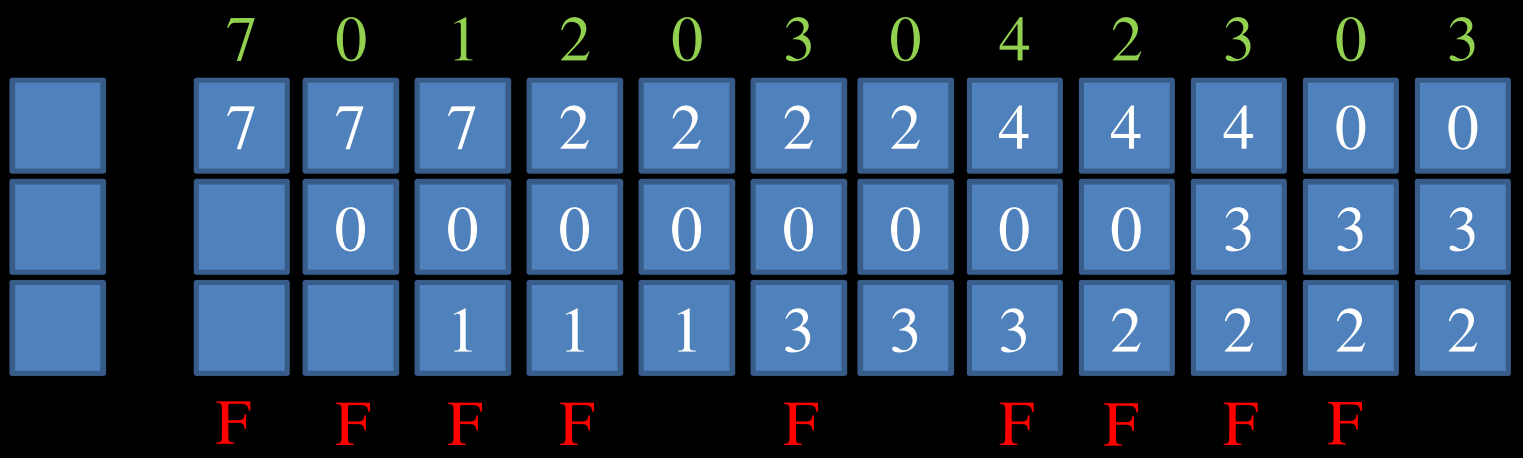
Memory Management

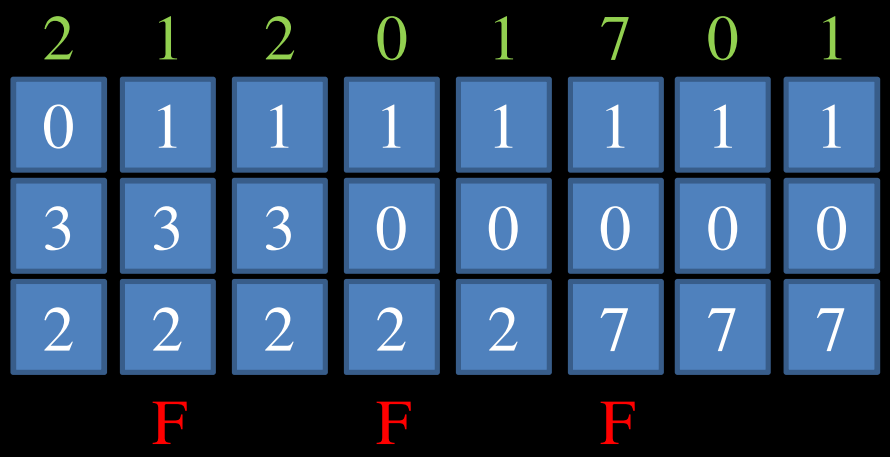
The optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string like SJF CPU-scheduling algorithm. It is generally used mainly for comparative analysis and studies.

Memory Management – Least Recently Used (LRU)

Reference String:

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 Frame Size: 03





Question & Answer

If an instruction takes time *m* if there is no page fault, and time *n* if there is a page fault, what is the effective instruction time if page fault occur once every *i* instructions?

For every *i* instruction, *i-1* instructions will take time *m*, and one instruction will take time *n*. The average instruction time is:

$$\frac{(i-1)\times m+n}{i} = m + \frac{(n-m)}{i}$$

Question & Answer

Consider the following page table with "In/Out" field. "In/Out" field implies whether the corresponding frame is in or out and page size is 2000 bytes long.

Page No	Frame No	In/Out
0	20	In
1	22	Out
2	200	In
3	150	In
4	30	Out
5	50	Out
6	120	In
7	101	In

Question & Answer

Which of the following virtual addresses would generate a page fault? For those that do not generate a page fault, to what physical address would they translate?

(a) 10451 (b) 5421 (c) 14123 (d) 9156

(a) Fault, Virtual address 10451 is offset 451 in page 5. Page 5 is "out" resulting in a fault

(b) 401421, virtual address 5421 is offset 1421 in page 2. Page 2 is "in" at frame 200 which has address 400000. Adding the offset to the frame's starting address yields physical address 401421

(c) 202123

(d) Fault

Question & Answer

On a system using demand-paged memory, it takes 200 ηs to satisfy a memory request if the page is in memory. If the page is not in memory, the request takes 7 ms if a free frame is available or the page to be swapped out has not been modified. It takes 15 ms if the page to be swapped out has been modified. What is the effective access time if the page fault rate is 5%, and 60% of the time the page to be replaced has been modified? Assume the system is only running a single process and the CPU is idle during page swaps.

The percentage of accesses satisfied in 200 ηs is 95%. Of the 5% of accesses that result in a page fault, 40% requires 7ms. Thus 5% × 40% = 2% of all accesses takes 7ms. Similarly 5% × 60% = 3% of accesses takes 15ms. Converting all times to µs yields:
0.95 × 0.2 + 0.02 × 7000 + 0.03 × 15000 = 590.19 µs

Question & Answer

Assume the amount of memory on a system is inversely proportional to the page fault rate. Each time memory doubles, the page fault rate is cut to half. Currently the system has 32Mb of memory. When a page fault occurs, the average access time is 1ms, otherwise 1µs. Overall, the effective access time is 300µs. How much additional memory would be needed to cut the effective access time to 100µs.

The current page fault rate (p) can be computed as:
$(1-p) \times 1 + p \times 1000 = 300$ i.e. $p \approx 30\%$.
The page fault rate necessary to maintain access time as 100µs is
$(1-p) \times 1 + p \times 1000 = 100$ i.e. $p \approx 10\%$.
So, the page fault rate of 30% must be halved twice to 7.5% to get it under 10%. Hence memory must be increased to 128 Mb.

Question & Answer

Given a page size of *s* and an average process size of *p* in a simple paged system, what is the average amount of memory lost per process due to internal fragmentation

The only space a process wastes is the portion of the last page that is unused. On average, half of the last page will be unused. The average wasted space per process is *s/2*. The average process size does not affect the answer.

Given the average size of a process is 30 KB, the page size is 2 KB and the size of a page table entry is 100 B, What will be the wasted space lost due to internal and table fragmentation ?

Average no of pages required per process = 30 / 2 =15. So, the amount of space required by the page table = 15 × 100 B = 1.5 KB. The amount of space lost due to internal fragmentation on average is 1 KB. So, Total wasted memory is 1.5 KB + 1 KB = 2.5 KB.