Fall 2015 COMP 3511
Operating Systems

Lab #7

Outline

- Review and examples on virtual memory
- Motivation of Virtual Memory
- Demand Paging
- Page Replacement

Q. 1

- What is required to support dynamic memory allocation in the following schemes:
  - *contiguous-memory allocation*
  - *paging*
  - *segmentation*

Q. 1

- contiguous-memory allocation:
  - might require relocation of the entire program
  - since there is not enough space for the program to grow its allocated memory space

- paging:
  - incremental allocation of new pages is possible in this scheme without requiring relocation of the program's address space

Q. 1

- segmentation:
  - might require relocation of the segment that needs to be extended
  - since there is not enough space for the segment to grow its allocated memory space

Q. 2

- Briefly explain the concept of *logical address* and *physical address*
- Logical address
  - Generated by the CPU
  - Also referred to as virtual address, i.e., the address always starts from zero
- Physical address
  - Address seen by the memory management unit (MMU) which maps logical address to physical address
- The user program deals with *logical addresses*; it never sees the real *physical addresses*

Q. 3

- Suppose a computer has an 8-bit address space, i.e., each logical address is 8-bit long. Page size is 32 bytes.

  (a) How many entries does the page table contain?

  (b) Part of the page table is shown here:

Page Number	Frame Number
0	5
1	1
2	3
3	2
4	7

Q. 3

- What are the physical addresses in decimal for the following logical addresses in binary?

  i. 00111111

  ii. 11000000

  iii. 10101010

  iv. 01010101

Q. 3

- Suppose a computer has an 8-bit address space, i.e., each logical address is 8-bit long. Page size is 32 bytes.

  (a) How many entries does the page table contain?

  Answer:

a) (a) 3 bits are left for the page number in logical address, so there are total 8 entries in the page table.

b) (b)

   i. 1 x 32 + 31 = 63

   ii. No translation can be done

   iii. No translation can be done

   iv. 3 x 32 + 21 = 117

Q. 4

- Consider a paging system with the page table stored in memory
- If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?

Q. 4

- 400 nanoseconds:
  - 200 nanoseconds to access the page table
  - 200 nanoseconds to access the word in memory

Q. 4

- Assume that finding a page-table entry in the associative registers takes zero time, if the entry is there
- If we add associative registers, and 75% of all page-table references are found in the associative registers
- what is the effective memory reference time?

Q. 4

- Effective access time

  = 0.75 × (200 nanoseconds) + 0.25 × (400 nanoseconds)

  = 250 nanoseconds.

Q. 5

- Compare the memory organization schemes of contiguous memory allocation, pure segmentation, and pure paging with respect to the following issues

- External fragmentation
- Internal fragmentation
- Ability to share code across processes

Q. 5

- The contiguous memory allocation scheme suffers from external fragmentation
  - Address spaces are allocated contiguously and holes develop as old processes die and new processes are initiated
- It also does not allow processes to share code
  - Process's memory space is not broken into noncontiguous fine-grained segments

Q. 5

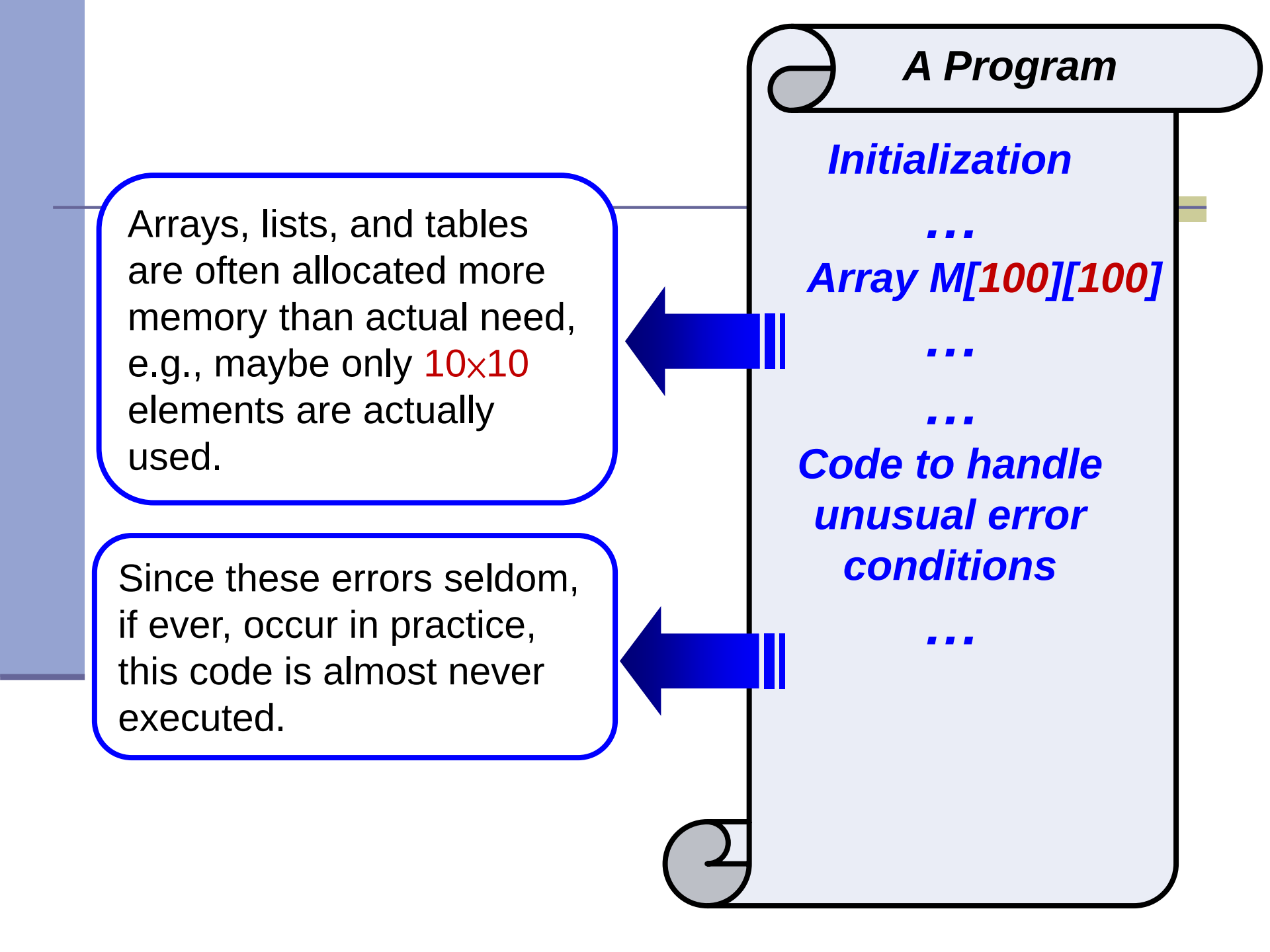
- Pure segmentation also suffers from external fragmentation
  - A segment of a process is laid out contiguously in physical memory and fragmentation would occur as segments of dead processes are replaced by segments of new processes
- It enables processes to share code
  - For instance, two different processes could share a code segment but have distinct data segments

Q. 5

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

Motivation of virtual memory

- Should an entire process be in memory before it can execute?
  - In fact, real programs show us that, in many cases, the entire program is not needed
    - *e.g., figure in the next slide*
  - Even in those cases where the entire program is needed, it may not all be needed at the same time


A Program
Initialization
…
Array M[100][100]
…
…
Code to handle unusual error conditions
…
Arrays, lists, and tables are often allocated more memory than actual need, e.g., maybe only 10×10 elements are actually used.
Since these errors seldom, if ever, occur in practice, this code is almost never executed.

Motivation of virtual memory

- Virtual memory benefits both the system and the user
  - *Logical address space can be much larger than physical address space*
  - *A program would no longer be constrained by the amount of available physical memory*

Motivation of virtual memory

- Cont.
  - *More programs could run concurrently, increasing CPU utilization and throughput*
  - *Less I/O would be needed to load or swap each user program into memory, so each user program would run faster*
  - *Allow processes to share files easily and to implement shared memory*

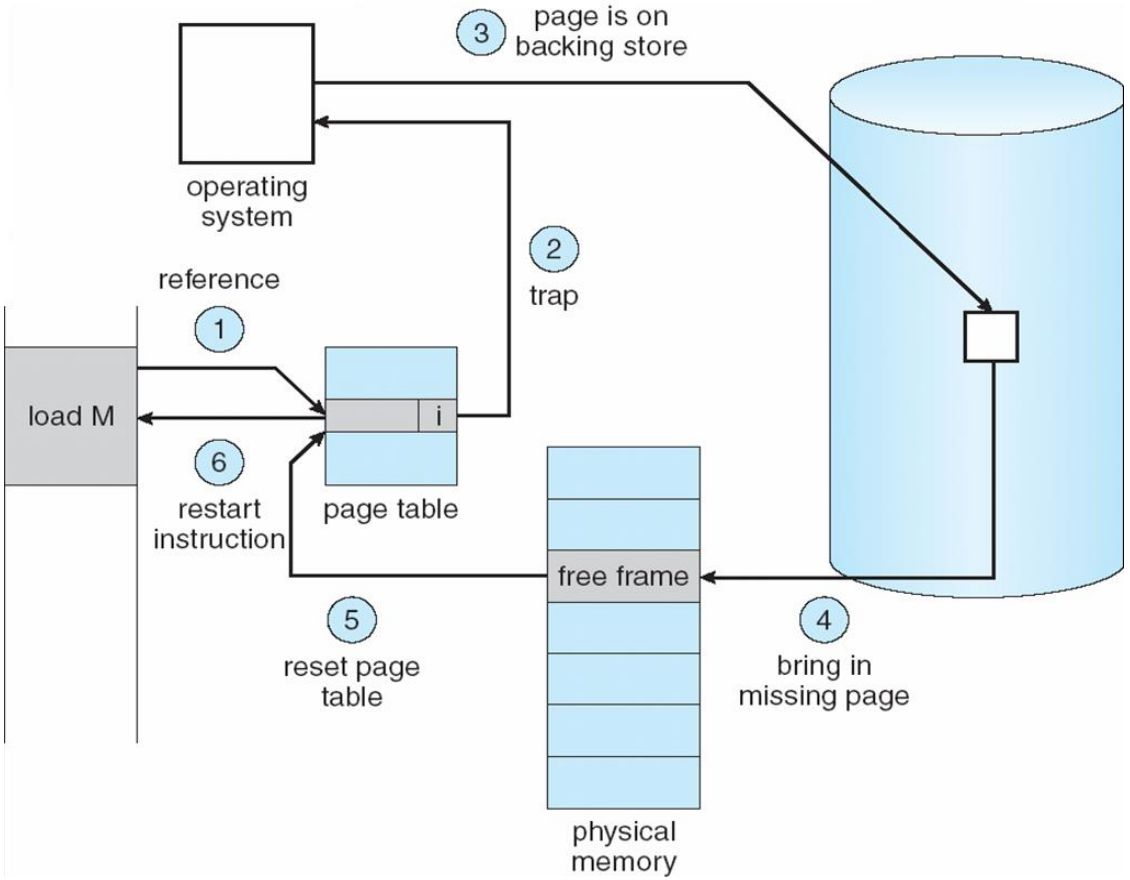
Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users

- Page is needed $\Rightarrow$ reference to it
  - Invalid reference $\Rightarrow$ abort
  - Not-in-memory $\Rightarrow$ bring to memory

Steps in Handling a Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system: page fault



An example of demand-paged memory

- Assume we have a demand-paged memory
  - The page table is held in registers
  - It takes 8 milliseconds to service a page fault if an empty page is available or the replaced page is not modified
  - It takes 20 milliseconds if the replaced page is modified
  - Memory access time is 100 nanoseconds

An example of demand-paged memory

- Assume that the page to be replaced is modified 70 percent of the time.

- *What is the maximum acceptable page-fault rate for an effective access time of no more than 200 nanoseconds ?*

An example of demand-paged memory

$$0.2\mu\text{sec} = (1 - P) \times 0.1\mu\text{sec} + (0.3P) \times 8 \text{ millisec} + (0.7P) \times 20 \text{ millisec}$$

$$0.1 = -0.1P + 2400\,P + 14000\,P$$

$$0.1 \approx 16{,}400\,P$$

$$P \approx 0.000006$$

Hardware support for demand paging

- For every memory access operation, the page table needs to be consulted:
  - check whether the corresponding page is resident or not
  - check whether the program has read or write privileges for accessing the page.

Hardware support for demand paging

- These checks would have to be performed in hardware.
- For example, a **TLB** could serve as a cache and improve the performance of the lookup operation.

Page Replacement

- If there is no free frame
- Page replacement – find some page in memory, but not really in use, swap it out
  - Replacement algorithm
  - performance – want an algorithm which will result in minimum number of page faults
  - Same page may be brought into and out of memory several times

Page Replacement

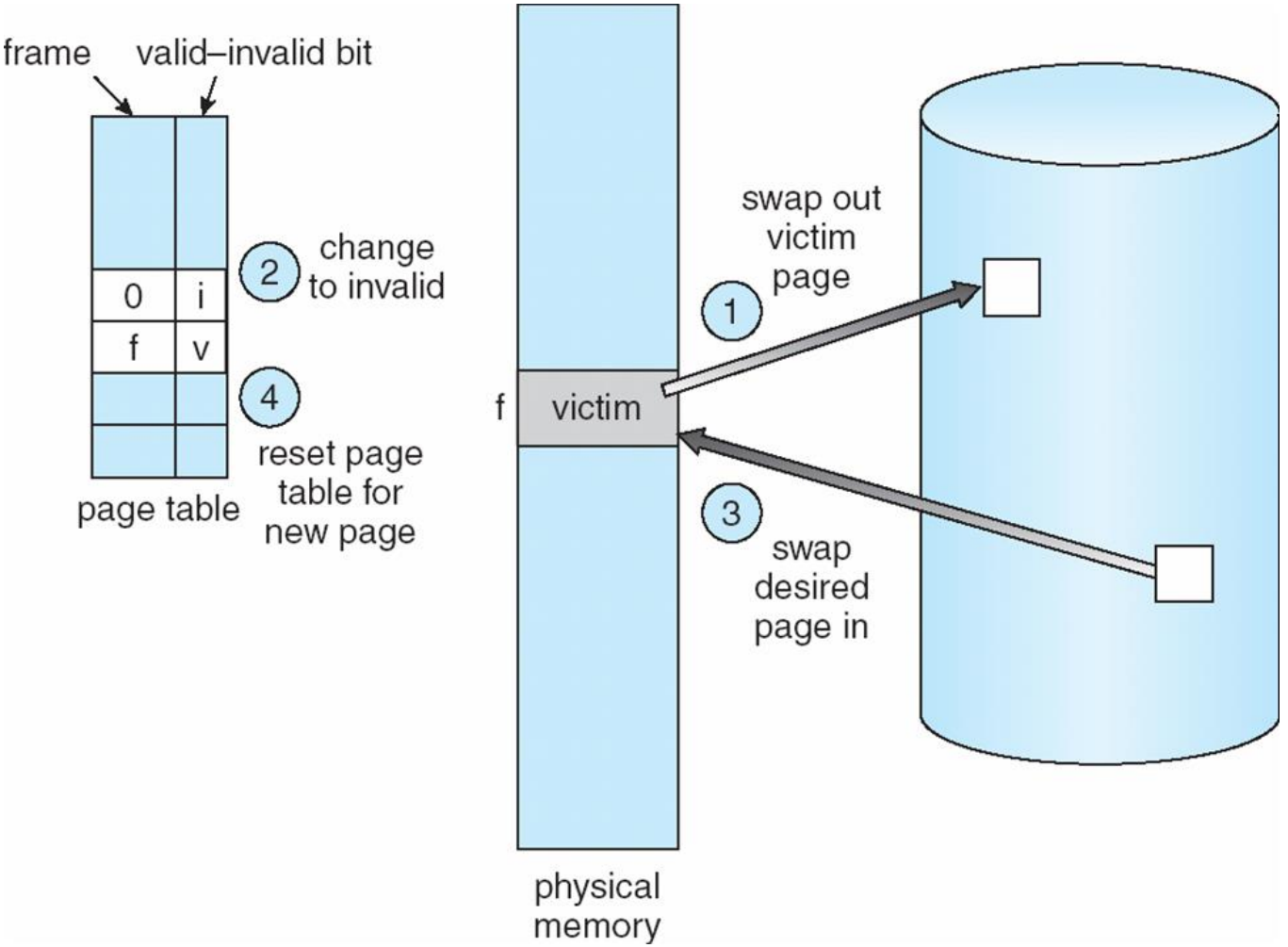
1. Find the location of the desired page on disk

2. Find a free frame inside the memory:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim** frame

3. Read the desired page into the (newly) free frame. Update the page and frame tables.

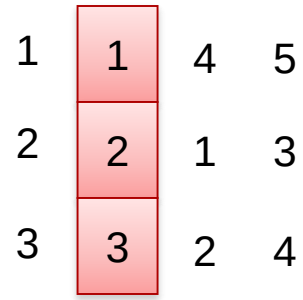
4. Restart the process

Page Replacement



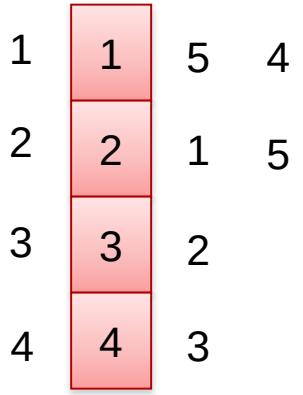
First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)



9 page faults

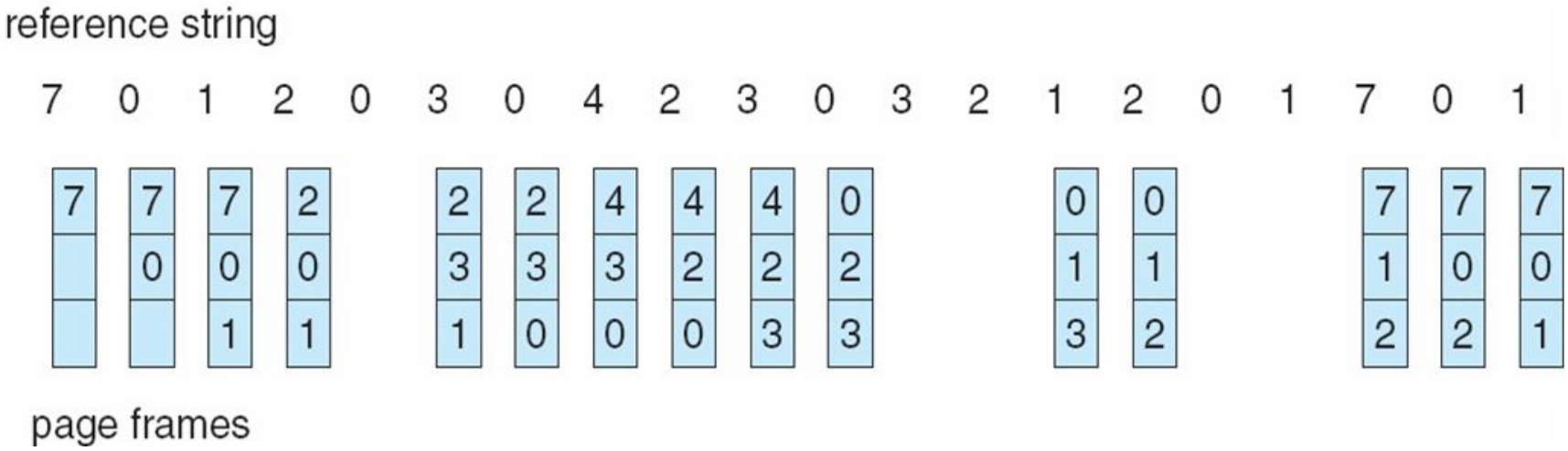
- 4 frames



10 page faults

more frames ⇒
more page faults

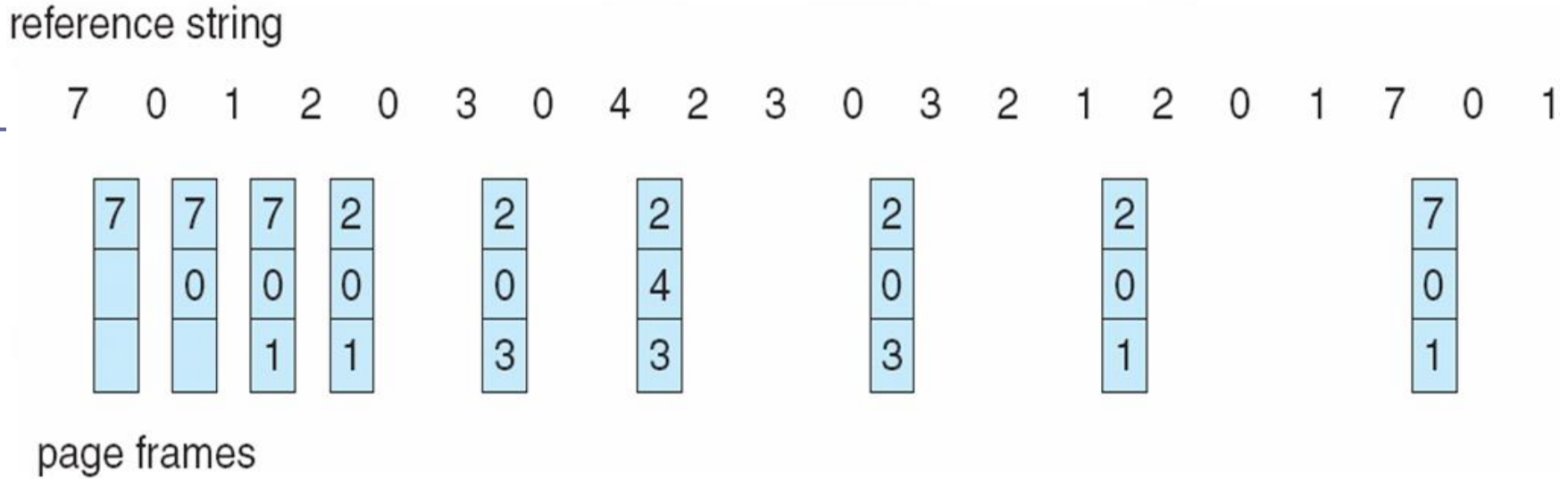
FIFO Page Replacement



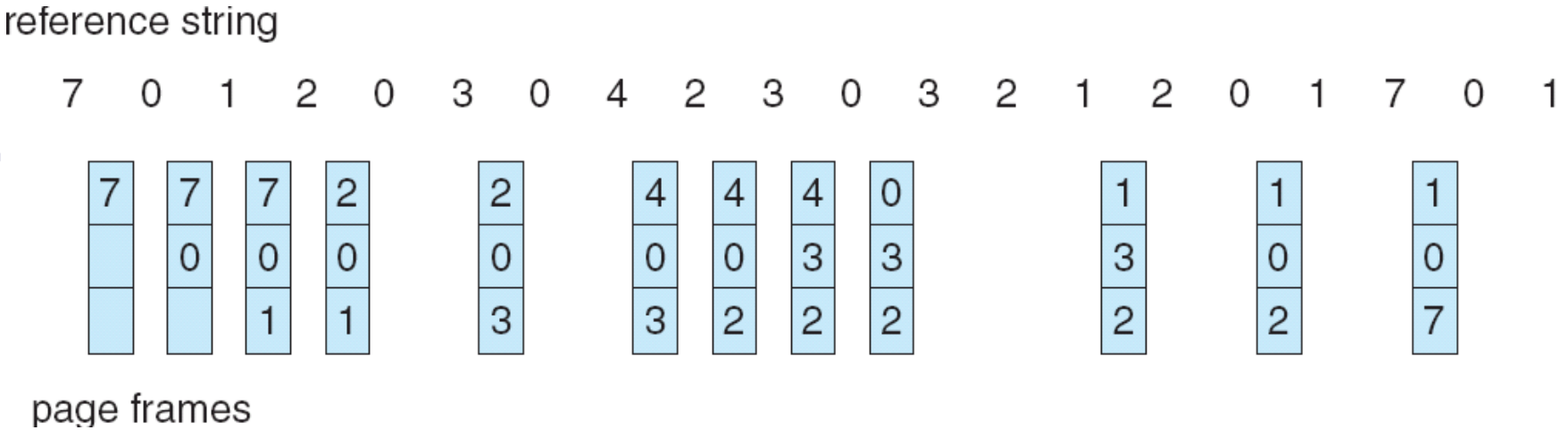
Algorithms for approximating optimal page replacement

- **LRU** (Least Recently Used) algorithm
  - Use the recent past as an approximation of the near future
    - Replace the page that has not been used for the longest period of time
  - Considered to be good, but how to implement
    - Few computer systems provide sufficient hardware support for true LRU
    - LRU-approximation: Reference bits, Second chance

- Optimal page replacement (9 page faults)



- LRU page replacement (12 page faults)



LRU Approximation Algorithms

- Reference bit
  - Each page is associate a reference bit, initially = 0.
  - When page is referenced, the reference bit set to 1.
  - We can determine which pages have been used or not used by examining the reference bits and replace the page whose reference bit is 0 (if one exists).
  - However, we do not know the *order* of use.
- Second chance
  - When a page is selected, check the reference bit.
  - If the value is 0, this page is replaced.
  - If the value is 1, set the bit to 0, then move on to select another page (FIFO). This page gets a second chance.

Algorithms for approximating optimal page replacement

- Keep a counter of the number of references that have been made to each page
  - **LFU** (Least Frequently Used) algorithm
    - an actively used page should have a large reference count
    - replaces page with smallest count
  - **MFU** (Most Frequently Used) algorithm
    - the page with the smallest count was probably just brought in and has yet to be used
  - The implementation of these algorithms is expensive, and they do not approximate OPT replacement well

An example of page-replacement algorithm

***MFU**: most frequently used page-replacement algorithm*

VS.

***LRU**: least recently used page replacement algorithm*

- (a) Under which situations, the **MFU** generates fewer page faults than the **LRU** ? Please give an example.
- (b) Under what circumstance does the opposite holds ?

An example of page-replacement algorithm

- (a) Consider the sequence in a system that holds four pages in memory: ***1 2 3 4 4 4 5 1***
  - the **MFU** evicts page 4 while fetching page 5
  - the **LRU** evicts page 1 while fetching page 5, then another page fault for fetching page 1 again
- (b) For the sequence "***1 2 3 4 4 4 5 4***," the **LRU** algorithm makes the right decision

Example 1

- A certain computer with
  - virtual-memory space of 2^{32} bytes
  - 2^{18} bytes of physical memory
  - The virtual memory is implemented by paging, and the page size is 1024 bytes
- A user process generates the virtual address 11123987.

Example 1

- How the system establishes the corresponding physical location ?

- The virtual address in binary form is

0001 0001 0001 0010 0011 1001 1000 0111

displacement in the page table, since the page table size is 2^{22}

displacement into the page, since the page size is 2^{10}

Example 2

- Consider the following sequence of memory accesses in a system that can hold four pages in memory: *1 1 2 3 4 5 1*

- Which of the following page replacement algorithms generates fewer page faults?

  - a) LRU: least recently used page replacement
  - b) LFU: least frequently used page replacement

Example 2

- memory accesses: *1 1 2 3 4 5 1*
- four pages in memory

- The LRU evicts page 1 while fetching page 5

- The LFU evicts a page other than 1 while fetching page 5
  - No page fault when page 1 is accessed again
  - fewer page faults

Example 3

- Discuss a situation under which the LRU generates fewer page faults than the LFU.

Example 3

- memory accesses: *1 1 2 3 4 3 4 5 2*
- four pages in memory

- The LFU evicts page 2 while fetching page 5

- The LRU evicts page 1 while fetching page 5
  - No page fault when page 2 is accessed again
  - fewer page faults

Q. 1

- Consider a system that allocates pages of different sizes rather than fixed_sized ones to its processes.
  - What are the advantages of such a paging scheme?
  - What modifications did to the virtual memory system can provide this functionality?

Q. 1

What are the advantages of such a paging scheme?

- The large program could have a large code segment or use large sized arrays as data.
- These portions could be allocated to larger pages, thereby decreasing the memory overheads associated with a fixed_sized page table.

Q. 1

What modifications are required in the virtual memory system that can provide this functionality?

- The virtual memory system would then
  - have to maintain multiple free lists of pages for the different sizes.
  - should also need to have more complex code for address translation to take into account different page sizes.

Q. 2

- Give the definition to each of the following terms:
  a) Optimal replacement
  b) LRU replacement
  c) Clock (second-chance) replacement
  d) FIFO replacement

Q.2

- **Optimal**: Replace the page that will not be used for the longest period of time by making use of future knowledge.
- **Least recently used (LRU) algorithm** : use the recent past as an approximation of the near future, then we can replace the page that has not been referred for the longest period of time.
- **Clock(Second chance) algorithm**: A page's reference bit is inspected when selecting the page. The page is replaced if the value is 0; but if the reference bit is 1, it is set to 0. Because a circular list is used so that the page gets a second chance to be replaced and we move on to select the next FIFO page. When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time.
- **FIFO**: Each page is associated with a time when arrived, the algorithm replaces the oldest page.

Q. 3a Page Replacement Algorithm

- Consider the following page reference string:

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.
- How many page faults would occur for the following replacement algorithms, assuming one, two, three, four, five, six, or seven frames? Remember all frames are initially empty, so your first unique pages will all cost one fault each.
  - LRU replacement
  - FIFO replacement
  - Optimal replacement

Q. 3a (LRU)

- LRU (1-frame): 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Total 20 page faults occurred.
- LRU(2-frame), total 18 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

  1 1 3 3 2 2 5 5 2 2 2 7 7 3 3 1 3 3

  2 2 4 4 1 1 6 6 1 3 3 6 6 2 2 2 6
- LRU (3-frame), total 15 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

  1 1 1 4 4 5 5 5 1 1 7 7 2 2 2

  2 2 2 2 2 6 6 6 3 3 3 3 3 3

  3 3 1 1 1 2 2 2 2 6 6 1 6

Q. 3a LRU

- LRU(4-frame): total 10 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1		1	1				1	1	6			6				
	2	2	2		2	2				2	2	2			2				
		3	3		5	5				3	3	3			3				
			4		4	6				6	7	7			1				

- LRU(5-frame): total 8 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1		1	1		1			1								
	2	2	2		2	2		2			2								
		3	3		3	6		6			6								
			4		4	4		3			3								
					5	5		5			7								

Q. 3a LRU

- LRU(6-frame): total 7 page faults occurred.

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1			1	1					1							
	2	2	2			2	2					2							
		3	3			3	3					3							
			4			4	4					7							
						5	5					5							
							6					6							

- LRU(7-frame): total 7 page faults occurred.

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1			1	1					1							
	2	2	2			2	2					2							
		3	3			3	3					3							
			4			4	4					4							
						5	5					5							
							6					6							
												7							

Q. 3a (FIFO)

- FIFO(1-frame): 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Total 20 page faults occurred.
- FIFO(2-frame): total 18 page faults occurred.

1,	2,	3,	4,	2,	1,	5,	6,	2,	1,	2,	3,	7,	6,	3,	2,	1,	2,	3,	6.
1	1	3	3	2	2	5	5	2	2		3	3	6	6	2	2		3	3
	2	2	4	4	1	1	6	6	1		1	7	7	3	3	1		1	6

- FIFO(3-frame): total 16 page faults occurred.

1,	2,	3,	4,	2,	1,	5,	6,	2,	1,	2,	3,	7,	6,	3,	2,	1,	2,	3,	6.
1	1	1	4		4	4	6	6	6		3	3	3		2	2		2	6
	2	2	2		1	1	1	2	2		2	2	6		6	6		3	3
		3	3		3	5	5	5	1		1	7	7		7	1		1	1

Q. 3a (FIFO)

- FIFO(4-frame): total 14 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1		5	5	5	5		3	3	3		3	1		1		
	2	2	2		2	6	6	6		6	7	7		7	7		3		
		3	3		3	3	2	2		2	2	6		6	6		6		
			4		4	4	4	1		1	1	1		2	2		2		

- FIFO(5-frame): total10 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1		1	6		6	6	6	6								
	2	2	2		2	2		1	1	1	1								
		3	3		3	3		3	2	2	2								
			4		4	4		4	4	3	3								
					5	5		5	5	5	7								

Q. 3a FIFO

- FIFO(6-frame): total10 page faults occurred.

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1			1	1			7			7	7	7				
	2	2	2			2	2			2			1	1	1				
		3	3			3	3			3			3	2	2				
			4			4	4			4			4	4	3				
						5	5			5			5	5	5				
							6			6			6	6	6				

- FIFO(7-frame): total 7 page faults occurred.

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1			1	1			1									
	2	2	2			2	2			2									
		3	3			3	3			3									
			4			4	4			4									
						5	5			5									
							6			6									
										7									

Q. 3a (Optimal)

- Optimal(1-frame): 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. Total 20 page faults occurred.
- Optimal(2-frame): total 15 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

  1 1 3 4 1 5 6 1 3 3 3 3 1 1 6

  2 2 2 2 2 2 2 2 7 6 2 2 3 3
- Optimal(3-frame): total 11 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

  1 1 1 1 1 1 3 3 3 3 6

  2 2 2 2 2 2 7 2 2 2

  3 4 5 6 6 6 6 1 1

Q. 3a (Optimal)

- Optimal(4-frame): total 8 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1			1	1				7				1				
	2	2	2			2	2				2				2				
		3	3			3	3				3				3				
			4			5	6				6				6				

- Optimal(5-frame): total 7 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1			1	1				1								
	2	2	2			2	2				2								
		3	3			3	3				3								
			4			4	6				6								
						5	5				7								

Q. 3a (Optimal)

- Optimal(6-frame): total 7 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1			1	1							1					
	2	2	2			2	2							2					
		3	3			3	3							3					
			4			4	4							7					
						5	5							5					
							6							6					

- Optimal(7-frame): total 7 page faults occurred.

  1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

1	1	1	1			1	1							1					
	2	2	2			2	2							2					
		3	3			3	3							3					
			4			4	4							4					
						5	5							5					
							6							6					
														7					

Q. 3a Page Fault Variation for LRU, FIFO, Optimal based on number of Frames

Number of Frames	LRU	FIFO	Optimal
1	20	20	20
2	18	18	15
3	15	16	11
4	10	14	8
5	8	10	7
6	7	10	7
7	7	7	7