Virtual Memory

Program

```
# include <stdio.h>
....
main()
{
  /*Lots and lots of initialization code*/

  for (i = 0;  i < N; i++)
  {
      ...
  }
...
}
```

Program

- Typically once you initialize you don't go back to that part of the code.
- Do you need to keep that part of the code in main memory once it has been used?

Virtual Memory: Main Idea

- Processes use a virtual (logical) address space.
- Every process has its own address space
- The virtual address space can be larger than physical memory.
- Only part of the virtual address space is mapped to physical memory at any time.
- Parts of processes' memory content is on disk.
- Hardware & OS collaborate to move memory contents to and from disk.

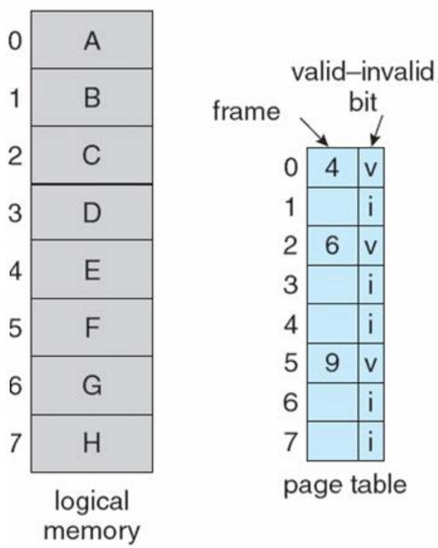
Demand Paging

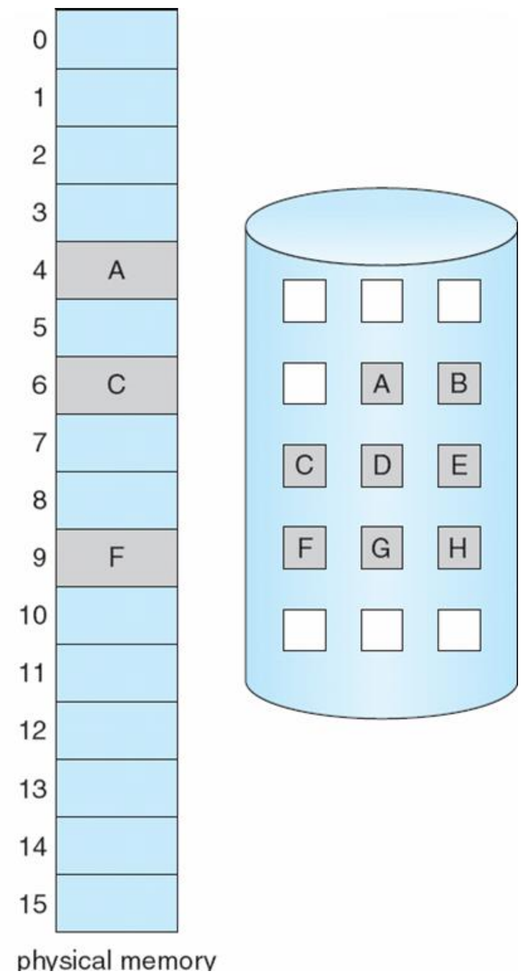
- Bring a page into memory only when it is needed
  - Why? Less I/O needed
    - If a process of 10 pages actually uses only half of them, then demand paging saves the I/O necessary to load the 5 pages not used.
  - Less memory needed
  - Faster response
  - More multiprogramming is possible

Demand Paging

- We need hardware support to distinguish between pages that are in memory and the pages that are on disk
- A valid-invalid bit is part of each page entry
  - When the bit is set to "valid" the associated page is in memory
  - If the bit is set to "invalid" the page is on the disk

Demand Paging



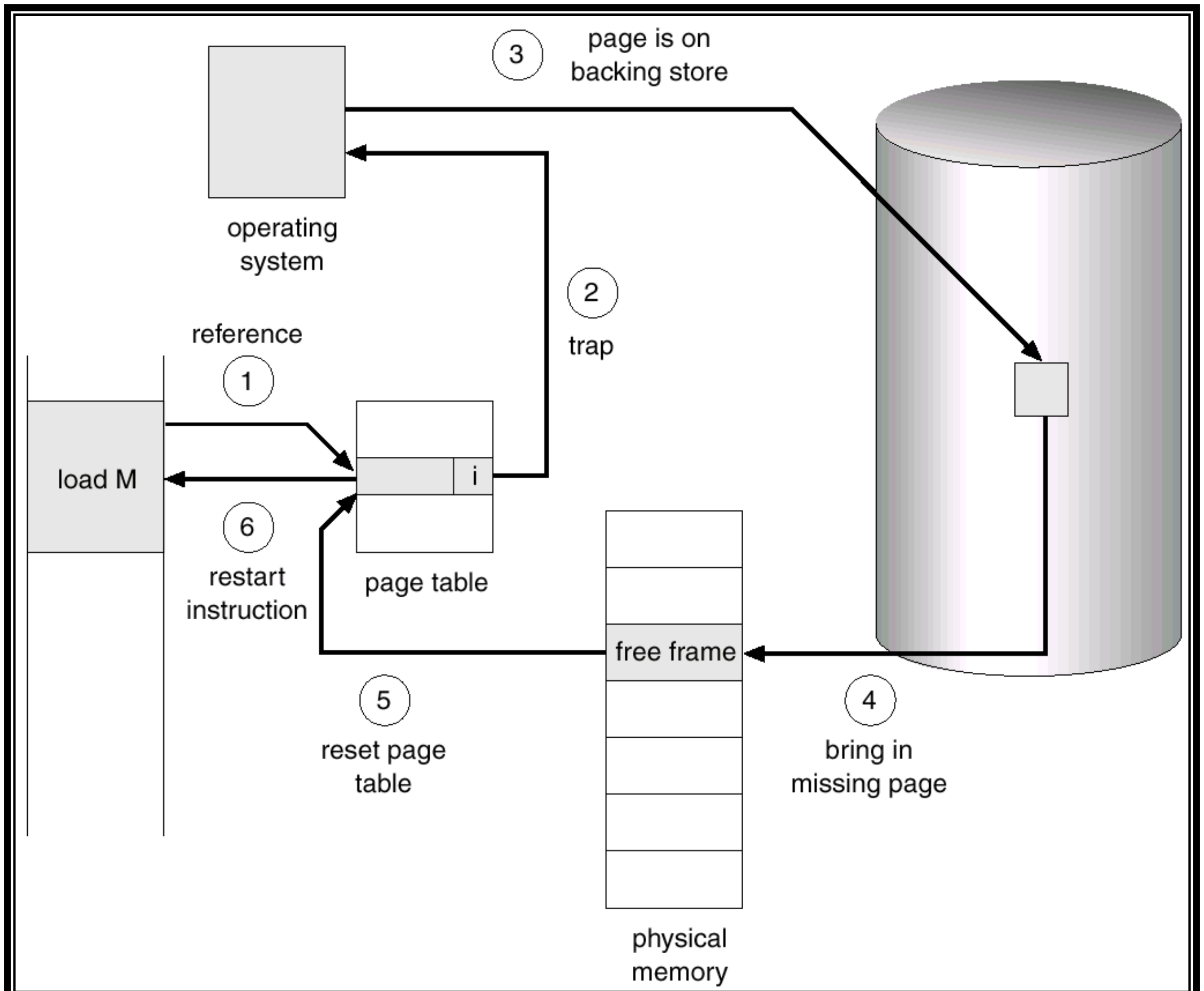


- The valid-invalid bit for 1 is set to “i” since the page is not in the physical memory
- The valid-invalid bit for 0 is “v” since the page is in memory

Page Fault

- What happens if a process tries to access a page that was not brought into memory?
  - Example: User opens a Powerpoint file.
- Access to a page marked invalid causes a page fault
- The paging hardware, in translating the address through the page table, will notice that the invalid bit is set causing a trap to the operating system.

Steps in Handling a Page Fault



Steps in Handling a Page Fault

- Service the page-fault interrupt
  - Trap to the OS
  - Prepare for a context switch
    - Save the user registers and process state
  - Read in the page
    - Issue a read from the disk to a free frame
      - Wait in the queue for this device until the read request is serviced
      - Wait for the device seek and/or latency time
      - Begin the transfer of the page to a free frame

Steps in Handling a Page Fault

- While waiting, allocate the CPU to some other user
- Receive an interrupt from the disk I/O subsystem

Steps in Handling a Page Fault

- Deal with Interrupt from the disk I/O system
  - Save the registers and process state for the other user
  - Correct the page table
  - Process with a page fault is put into the ready queue
- Wait for the CPU to be allocated to the process again
- Restore the user registers, process state and new page table, and then resume the interrupted instruction

Challenge: Performance

- Page Fault Rate $0 \le p \le 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

- Let p be the probability of page fault:
  - Avg. time = (1-p) * memory time + p * page fault time
  - Assuming: memory time = 200ns, page fault time = 8 millisecond, p = 0.1%
    - Avg time = 99.9% * 200 + 0.1% * 8000000 = 8200
    - Access time is directly proportional to the probability of a page fault
  - If one access out of 1000 causes a page fault the effective access time is 8.2 microseconds
- Need to keep the page fault rate small!

Page Replacement

- So why allow demand paging?
- If a process of 10 pages actually uses only half of them, the demand paging saves the I/O necessary to load the 5 pages that are never used
- This allows us to increase the level of multiprogramming

Page Replacement

- Let's assume that our physical memory consists of 40 frames
- We have 8 processes with 10 pages. That is 80 pages.
  - Obviously 80 pages is more than 40 frames
  - But if a process is only using half of its pages is this really a problem?
- But there is a reason why there are 10 pages
  - The process may need them
- The frames have been over-allocated i.e., overbooked

Page Replacement

- What do we do when a process needs a frame and there isn't one free?
- Essentially we choose a frame and free it of the page that is currently residing on it

Page Replacement

- A page replacement algorithm describes which frame becomes a victim.
- Designing an appropriate algorithm is important since disk I/O is expensive
- Slight improvements in algorithms yield large gains in system performance

Page Replacement

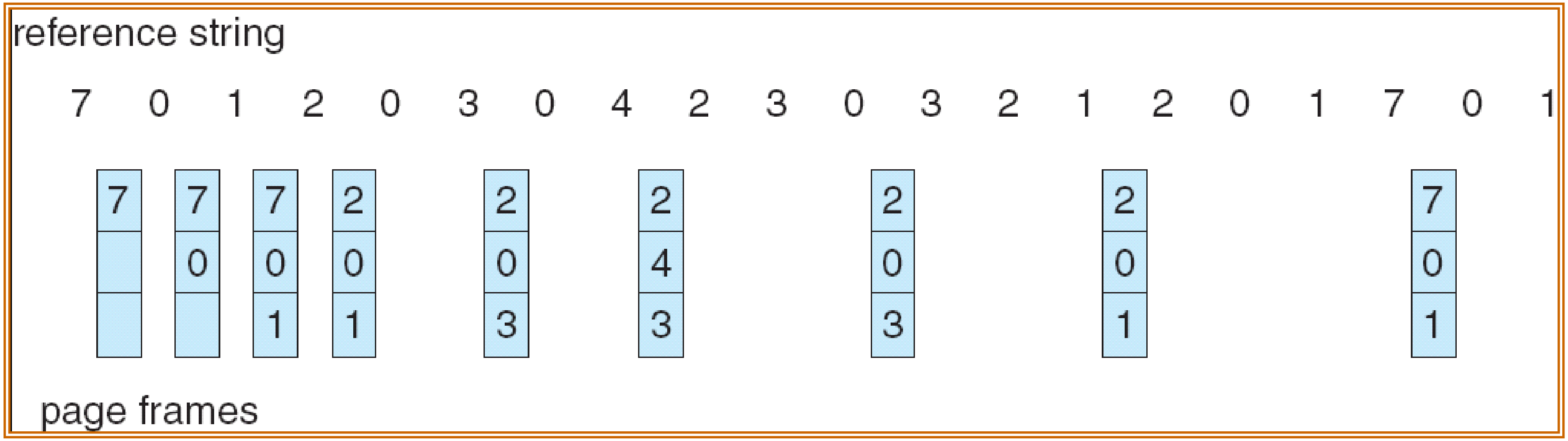
- We will discuss several algorithms
- The examples assume:
  - 3 frames
  - Reference string:
    7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
  - Each of the numbers refers to a page number

Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future i.e. replace the page that will not be used for the longest period of time

- This should have the lowest page fault rate

Optimal Page Replacement



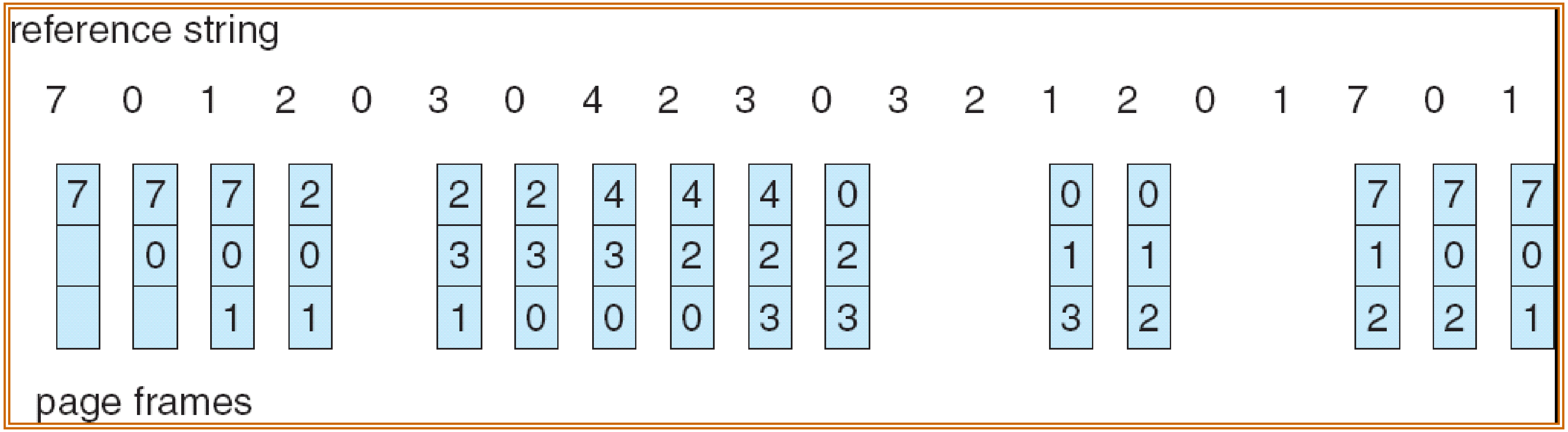
Optimal Page Replacement

- Optimal is easy to describe but impossible to implement
- At the time of the page fault, the OS has no way of knowing when each of the pages will be referenced next

FIFO Page Replacement Algorithm

- Maintain a linked list of all pages
  - Each page is associated with the time when that page was brought into memory
- Page chosen to be replaced is the oldest page
- Implementation: FIFO queue
  - A variable head points to the oldest page
  - A variable tail points to the newest page brought in

FIFO Page Replacement



FIFO Page Replacement

- Advantages
  - Easy to understand and program
- Disadvantage
  - Performance is not always good
  - The page replaced may be an initialization module that was used a long time ago and is no longer needed, but on the other hand …
    - The page may contain a heavily used variable that was initialized early and is in constant use

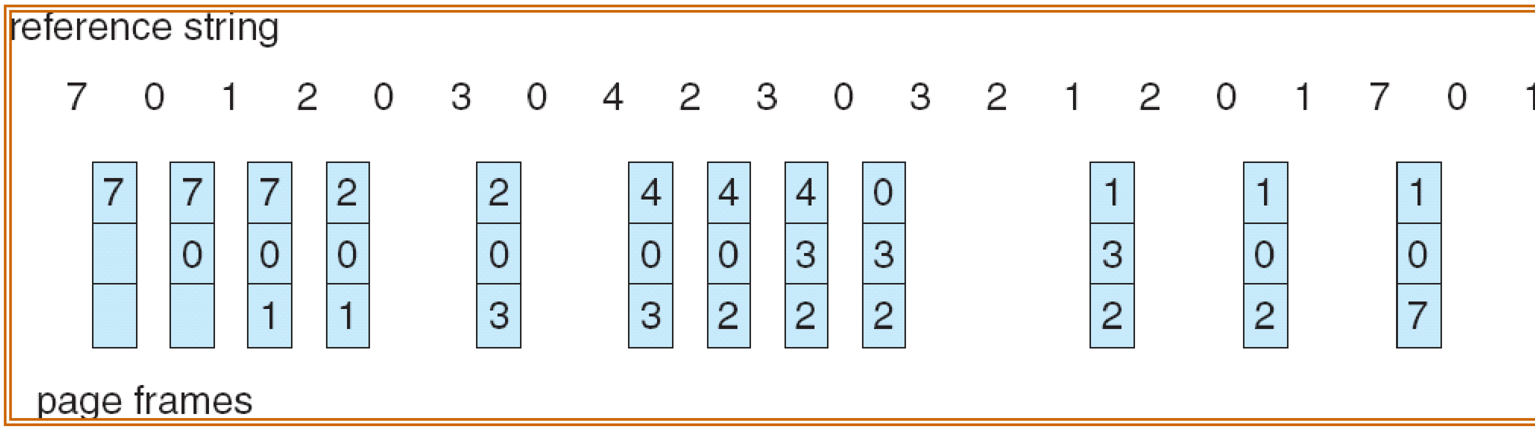
LRU Page Replacement

- FIFO replacement algorithm uses time when a page was brought into memory
- The optimal replacement algorithm uses the time when a page is to be used.
- Can we use the recent past as an approximation of the near future?
  - This means replace the page *that has not been used* for the longest period of time
  - This approach is the Least-Recently-Used (LRU) algorithm.

LRU Replacement Algorithm

- LRU replacement associates with each page the time that of that page's last use
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.

LRU Page Replacement



LRU Page Replacement

- LRU is often used and is considered to be good
- Challenge: Implementing LRU
- Implementation using Counters:
  - Associate with each page-table entry a time-of use field
  - Add to the CPU a logical clock or counter
    - Clock is incremented for every memory reference
  - Each time a page is referenced the time-of-use field is updated with the logical clock
  - A search of the page table is needed to find the least recently used page

LRU Page Replacement

- Implementation using Stack:
  - Keep a stack of page numbers
  - When a page is referenced, it is removed from the stack and put on the top
  - The most recently used page is always at the top of the stack and the least recently used is at the bottom
  - Should use a doubly-linked list since entries can be removed from the middle of the stack

LRU Approximation Algorithms

- LRU needs special hardware and still slow
- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced, bit set to 1
  - Replace any with reference bit = 0 (if one exists)
    - We do not know the order, however

LRU Page Replacement

- Implementation using Reference Bits:
  - Each page entry has a set of reference bits e.g., 8 bits
  - The bits represent the history of page use for the last 8 time periods
    - 00000000: implies that the page has not be used in the last 8 intervals
    - If a page history has bits as 11000100 then it has been used more than a page with bits as 01110111
  - At regular intervals (e.g., 100 milliseconds) a timer interrupt transfers control to the OS
  - OS shifts the reference bit for each page into the high-order bit of its 8-bit byte and shifts the other bits to the right by 1 bit
  - Low-order bit falls off and it is discarded
  - Page with lowest number is LRU

LRU Page Replacement

- The last implementation is an approximation of LRU
- Commonly found

2^{nd} Chance Page Replacement Algorithm

- A modification to FIFO avoids the problem of throwing out a heavily used page
- Each page entry has a reference bit, R.
- The modification is based on an inspection of the R bit of the oldest page
- If R=0 the page is both old and unused
  - Replace immediately
- If R=1
  - R is cleared and put at the end of the list
  - The load time is reset to the current time

Other Algorithms

- Least frequently used (LFU)
- Most frequently used (MFU)

- Most OS's use LRU

Least Recently Used (LRU) Algorithm

- Why does LRU work?
- Consider the following code segment:

```
sum = 0;
for (i=0; i< n; i++)
{
    sum  = sum + a[i];
}
```

- What do we see here?
  - We see that a[i+1] is accessed soon after a[i]
  - We see that the sum is periodically referenced

Least Recently Used (LRU) Algorithm

- What else do we see?
  - We are cycling through the for-loop repeatedly
- The program exhibits
  - Temporal locality: recently-referenced items are likely to be referenced in the near future
    - Referencing sum
    - For loop instructions
  - Spatial locality: Items with nearby addresses tend to be referenced close together in time
    - Variable a[i+1] accessed after variable a[i]

The Working Set Model

- Processes tend to exhibit a locality of reference e.g.,
  - This means that during any phase of execution, the process references only a relatively small fraction of its pages
- The set of pages that a process is currently using is called its working set
- If the entire working set is in memory, the process will run without causing many faults until it moves into another execution phase
  - Example: Move to another loop

LRU Replacement Algorithm

- Locality suggests that memory references are on the same set of pages
- Studies suggest that programs exhibit high spatial and temporal locality

Summary

- We have studied the need for page replacement algorithms
- Several algorithms have been discussed including:
  - Optimal
  - FIFO
  - LRU