this covers the `virtual memory` slide deck

toc:

Background

code needs to be in physical memory to execute but entire program is rarely used

- error code
    - used to handle unusual error conditions
    - almost never executed
- large data structures
    - often allocated more mem than needed
- unusual routines
    - lesser used options and features of a program

we only have to load what we need for execution

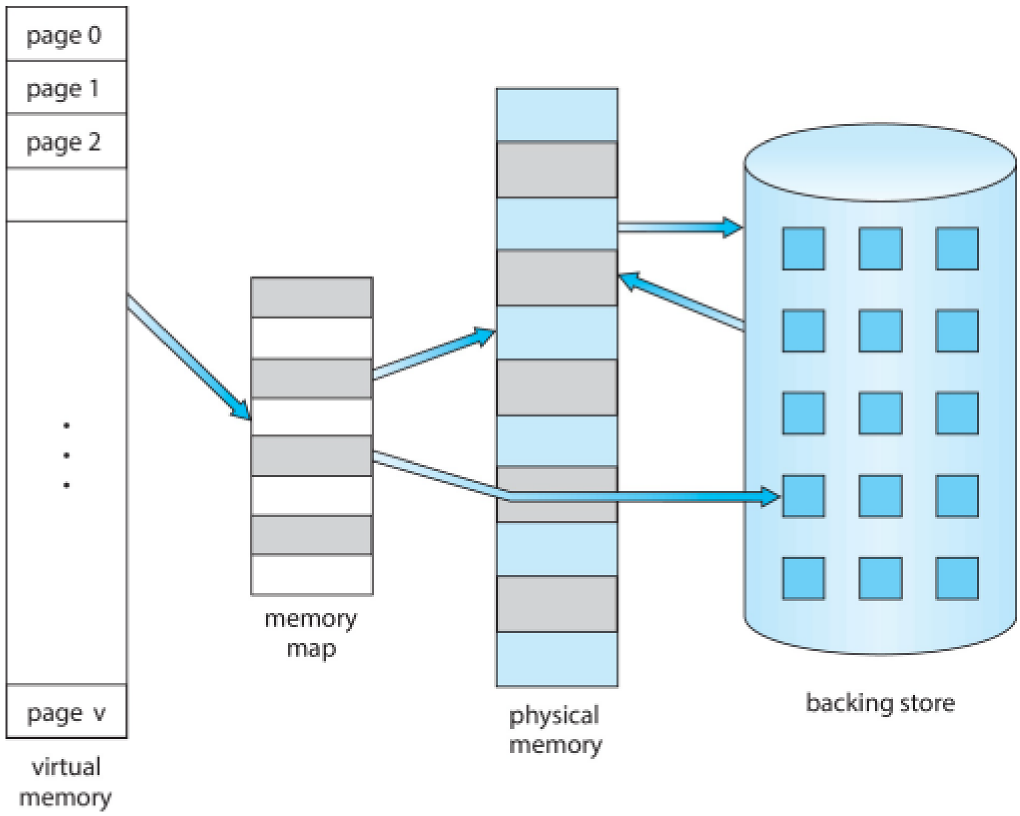
ability to execute partially-loaded program

- no longer limited by physicla mem
- each program takes less memory while running
    - more programs can run at the same time
        - cpu util and throughput increase
        - response time and turnaround time the same
- less i/o needed to load or swap programs into mem
    - each user program runs faster

virtual mem

- separation of user logical mem from physical mem
- only part of the pgram needs to be in mem for execution
- logical address space can be larger than physical address space

- allows address spaces to be shared by several processes
- allows for more efficient process creation
- more programs running concurrently
- less i/o needed to load or swap processes

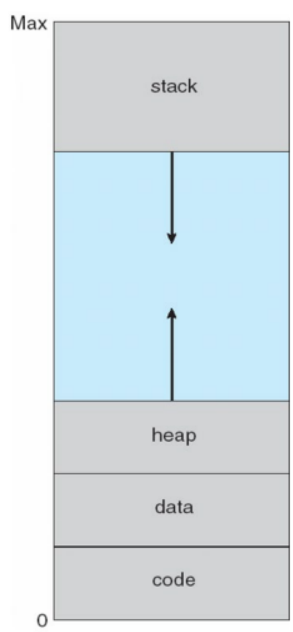


Virtual Memory That is Larger Than Physical Memory

Virtual Address Space

- **Virtual address space** – logical view of how process is stored in memory:
  - Usually start at address 0, contiguous addresses until end of space.
  - Meanwhile, physical memory organized in page frames.
  - MMU must map logical pages to physical page frames in memory.

- Logical memory is also known as virtual address space.



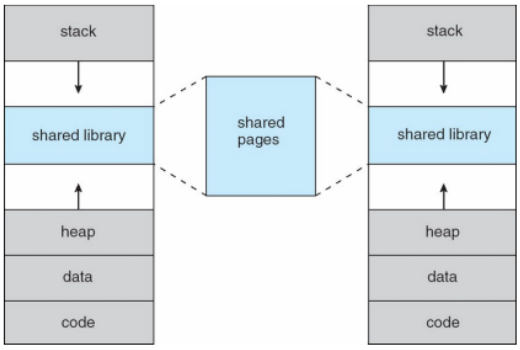
Virtual address space of a process in memory.

- The large blank space (or hole) between the heap and the stack is part of the virtual address space.
    - Will require actual physical pages only if the heap or stack grows.

- Virtual address spaces that include holes are known as sparse address spaces.
    - Beneficial because the holes can be filled as the stack or heap segments grow or if we wish to dynamically link libraries during program execution.

Shared Library Using Virtual Memory

- Virtual memory allows files and memory to be shared by two or more processes through page sharing.
- Examples:
    - Library mapped as read-only.
    - Shared memory for 2 processes to communicate.



Shared library using virtual memory.

Logical Memory vs. Virtual Memory vs. Real Memory

- Logical memory is the memory space perceived by a process.
- Real memory is the physical RAM installed in the system.
- Virtual memory is a memory management technique used by the operating system to extend available memory beyond physical RAM.

- logical memory
    - memory space perceived by a process
- real/physical memory

    - physical ram sticks in you’re machine
- virtual memory
    - memory management technique
    - used by operating system
    - extends beyond available physical ram

Knowledge check:

- which of the following is a benefit of allowing a program that is only partially in memory to execute?
    - d. all of the above
    - programs can be written to use more memory than is available in physical memory
    - cpu util and throughput is incdreased
    - less io needed to load or swap each user program into mem
- in general virtual memory decreases the degree of multiprogramming in a system
    - false
    - why would these even effect each other???

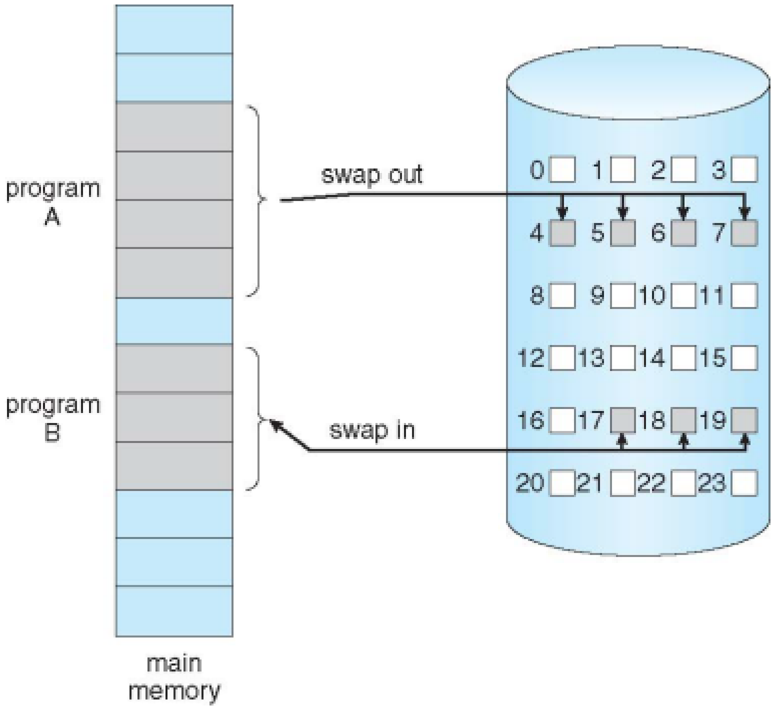
Demand Paging

virtual memory can be implemented via

- demand paging
- demand segmation

demand paging:

- bring a page into memeory ONLY when it is needed
    - less required
        - I/O
        - memory
    - faster response
    - more users
- similar to a paging system with swapping



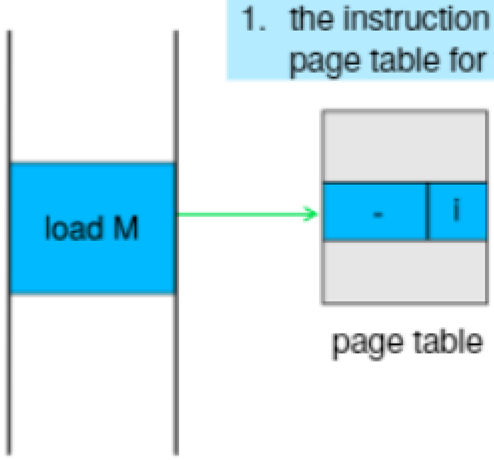
Swapping with paging.

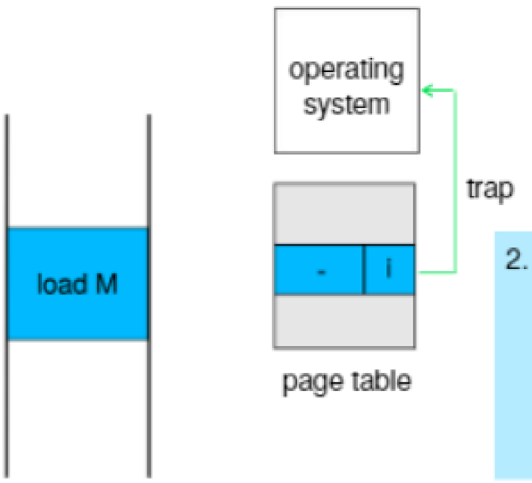
valid-invlid bit

- with each page entry there is a valid-invalid bit
  - v $\longrightarrow$ in-memory - memory resident
  - i $\longrightarrow$ not-in-memory
- initially valid-invalid bit is set to `i` on all entries
  - all invalid
- during MMU address translation
  - if bit is invalid for a page entry then there's a page fault

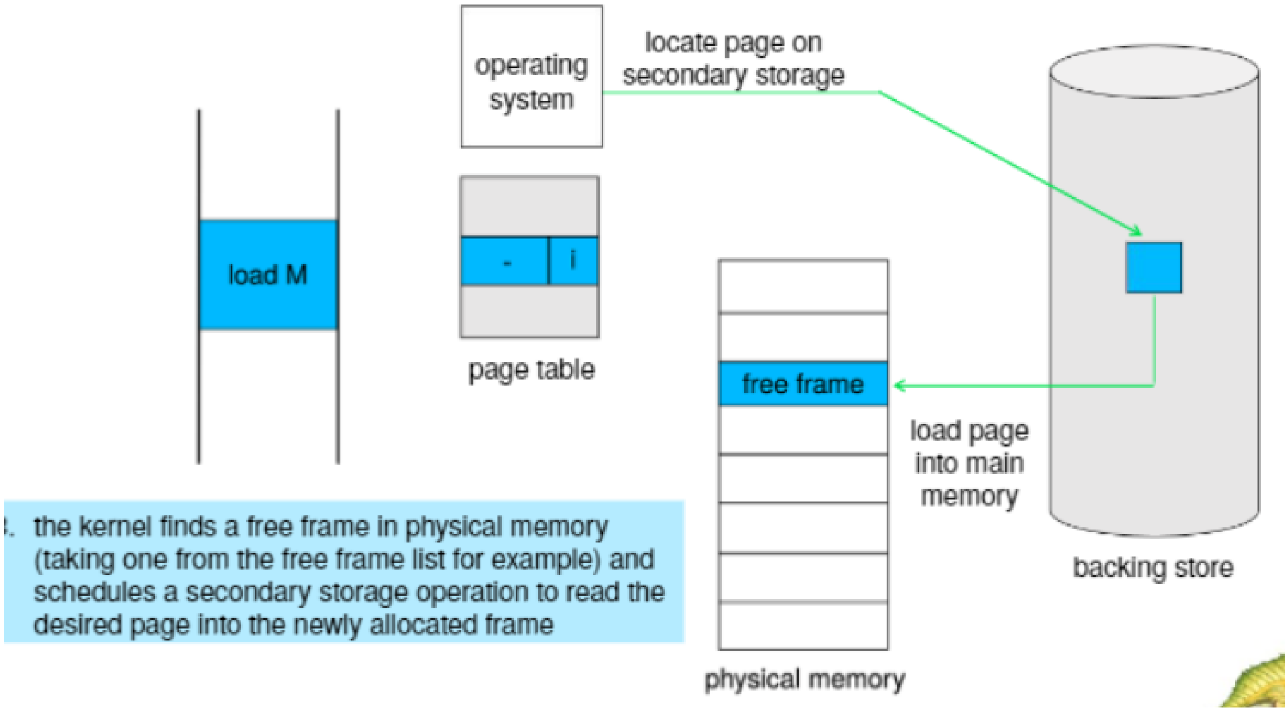
Steps in Handling a Page Fault

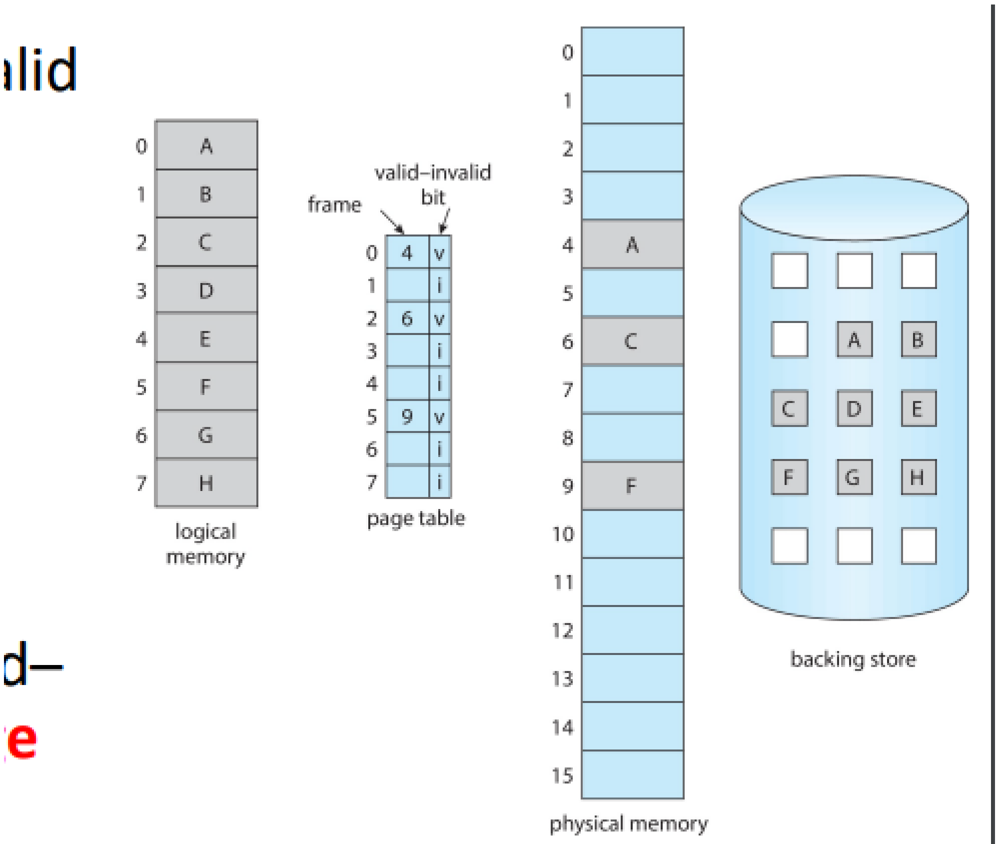
1. the instruction “load M” executes, accessing the page table for the address containing “M”



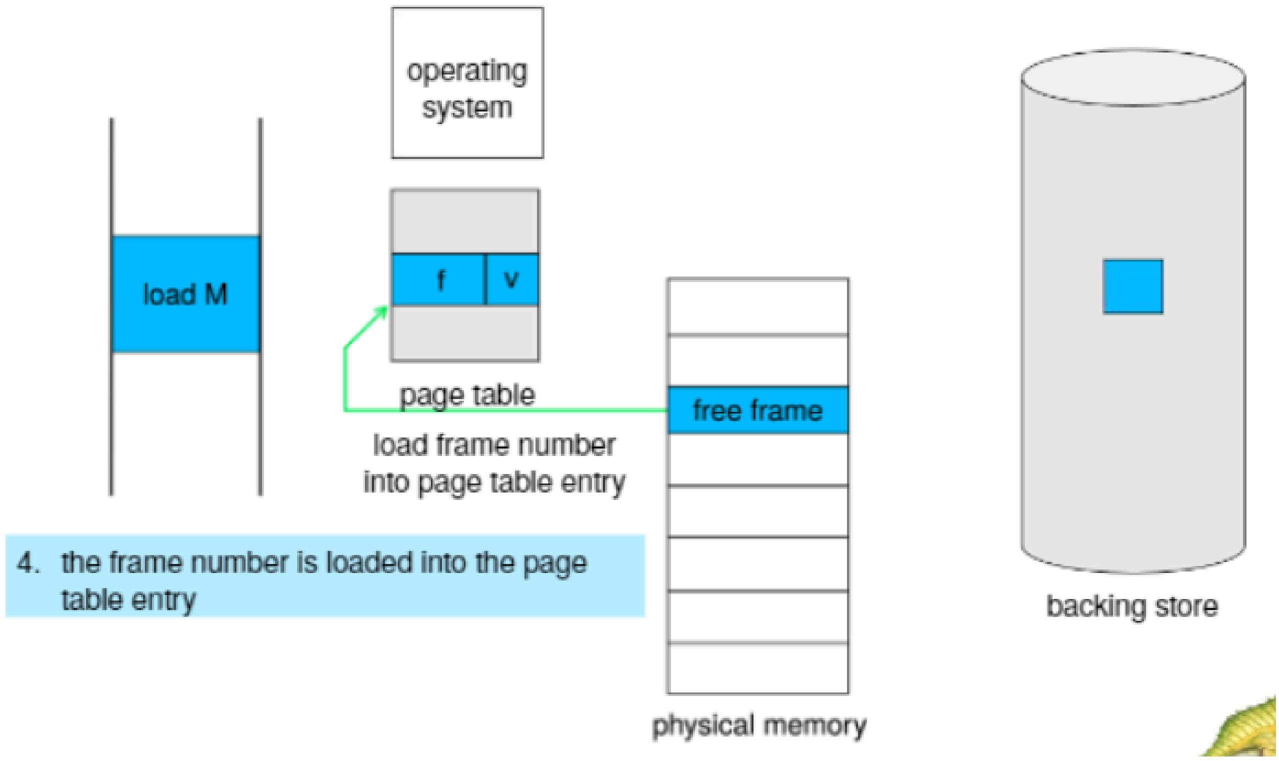


2. the page is marked as invalid, causing a page fault trap to the kernel
   – the operating system checks if the reference is legal or not
   – if it is illegal, the process is terminated
   – in this case it is valid but the page is not currently in a frame in physical memory

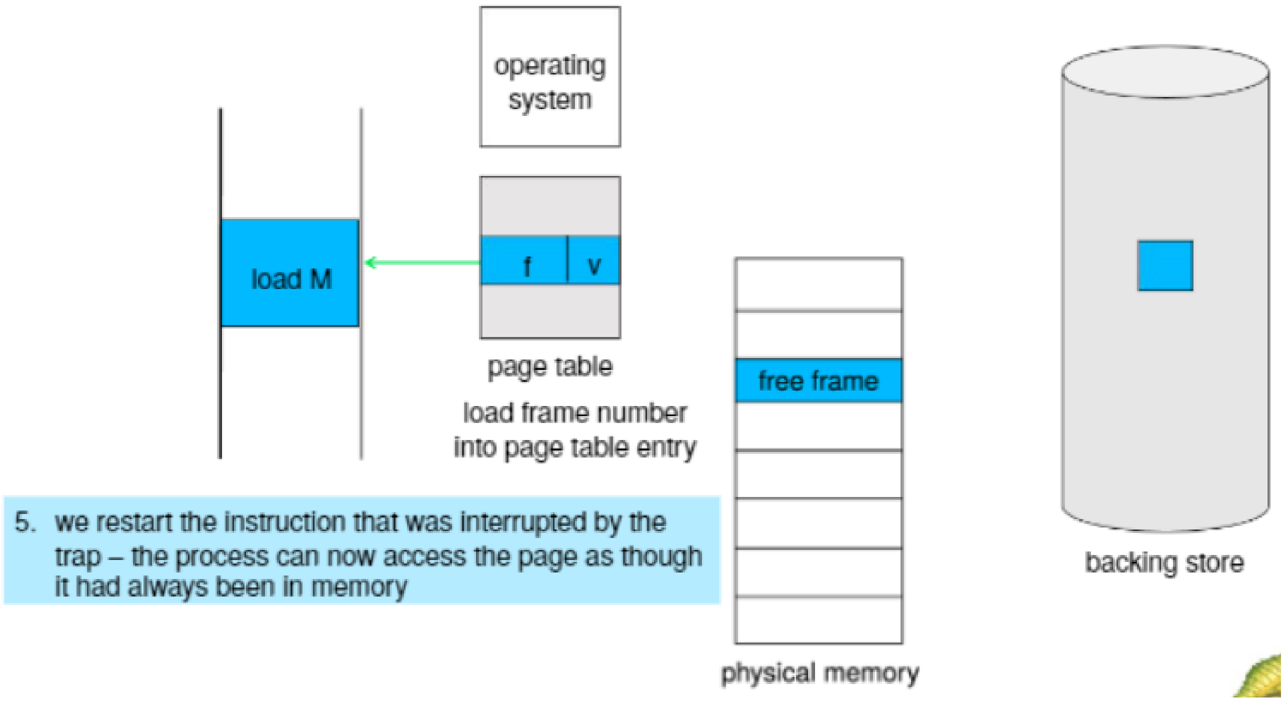




Page Table When Some Pages Are Not in Main Memory



Steps in Handling a Page Fault



`there are some more slides that are the same thing`

free-frame list

- a pool of free frames for satisfying such requests
- maintained by most operating systems

- when a page fault occurs, the operating system must bring the desired page from secondary storage into main memory
- OS typically allocate free frames using zero-fill-on-demand
    - we flush out the frame
    - the content of the frames zeroed-out before being allocated
- when a system starts up, all available memory is placed on the free-frame list
    - this lets us start allocating frames as they are demanded

List of free frames.

performance of demand paging

- 3 major activities of servicing a page-fault
    - serivce the page-fault interrupt
    - read in the page
        - this is where most of the time is spent
    - restart the process
- page fault rate $0 \leq p \leq 1$
    - $p = 0 \rightarrow$ no page faults
    - $p = 1 \rightarrow$ every reference is a fault
- Effective Access Time (EAT) =
    - $(1-p) \times \text{memory access} + p(\text{page fault overhead} + \text{swap page out} + \text{swap page in})$
    - we're basically averaging it out

Example of Demand Paging

memory access time = 200 ns

avg page-fault service time = 8 ms

$$\begin{aligned} \text{EAT} &= (1-p) \times 200 + p(8\text{ms}) \\ &= ((1-p) \times 200) + (p \times 8,000,000) \\ &= 200 + (p \times 7,999,800) \end{aligned}$$

if 1 in 1000 accesses causes a page fault ($p = 0.001$) then EAT = 8.2 microseconds. a slowdown by a factor of 40

if we want a perfoamnce degredation (slowdown) less than 10 percent

- 10% of 200 (effective access time we want) = 20 ns
- 220>200 + 7,999,800 * p
- 20 > 7,999,800 * p
- p < 0.0000025
- p < one page fault in every 400,000 memory accesses

This needs hardware support, it can't all be done by the OS.

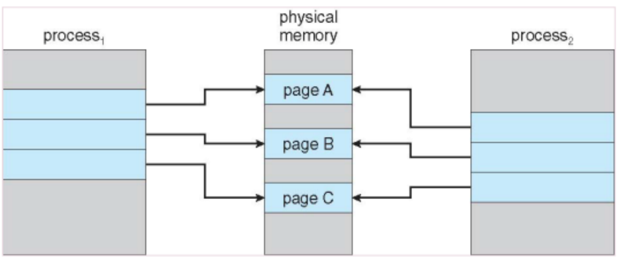
Knowledge Check

- on a system with demaend-paging, a process will experience a high page fault rate when the process begins execution
  - true
  - similar principle to caching where we need to slowly fill things up in the memory that the process asks for
  - since the page table is invalid to start with we end up faulting a ton as we're setting things up for the process
  - as it starts going on it'll start hitting the pages we already loaded for it
- a page fault occurs when
  - c. a process tries to access a page that is not loaded in memory
- if memory access time is 250 ns and average page fault service time 10 ms the probability of page faults must be less ___ to keep the performance degradation less than 20%
  - options
    - a) 0.0000025
    - b) 0.000005
    - c) 0.0000075
    - d) 0.00001
  - takehome exercise

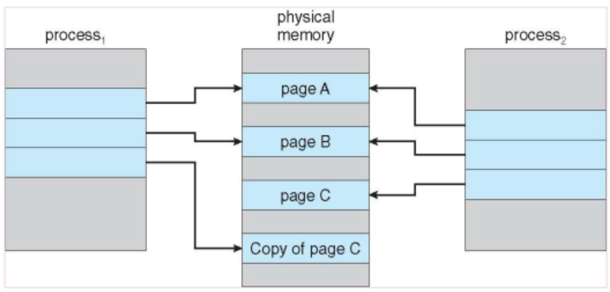
Copy on Write

copy-on-write (COW):

- allows both parent and child processes to initially share the same pages in memory
- the page is copied once either process modifies a shared page
  - read: write
- allows more efficient process creation as only modified pages are copied



Before process 1 modifies page C.



After process 1 modifies page C.

Knowledge Check

- ___ allows the parent and child processes to initially share the same pages, but when either process modifies a page, a copy of the shared page is created.

- a. copy-on-write

Page Replacement

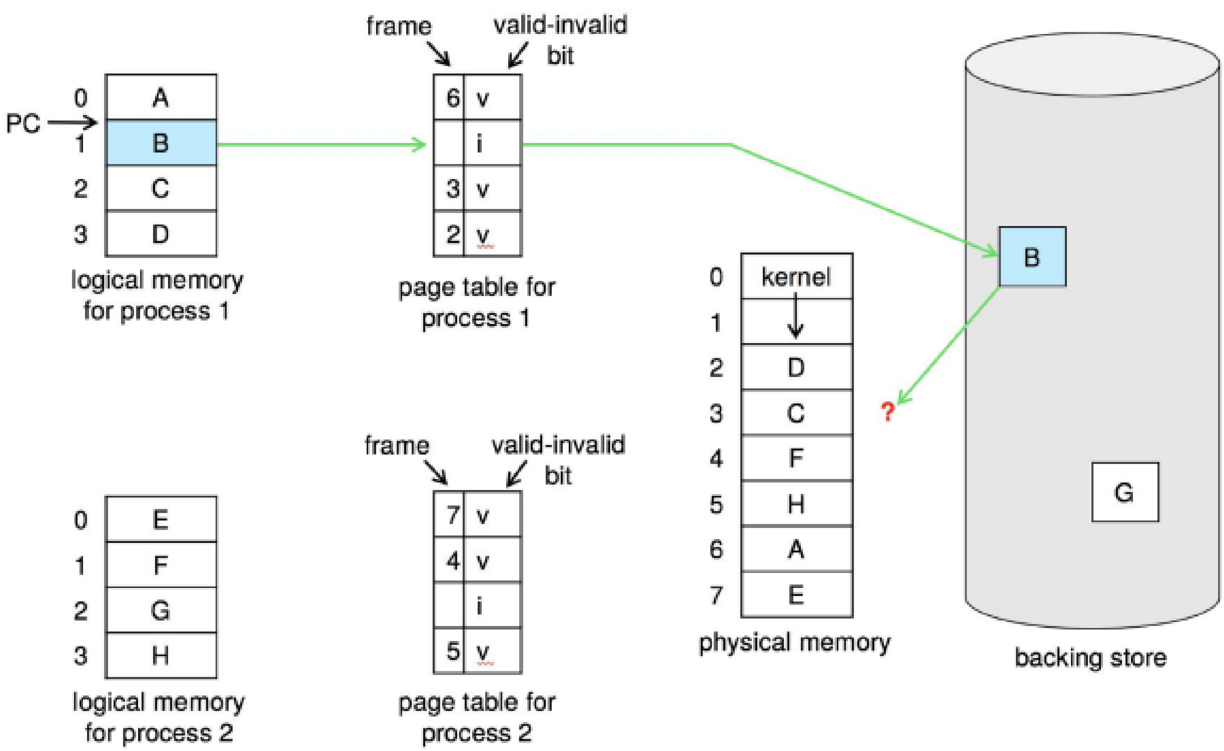
sometimes there is no free frame

how:

- memory is used up by process pages but also in demand from the kernel, I/O buffers, etc
- the page is a hot commodity that everyone needs access to

scenario

- process executing
- page fault occurs
- OS determines where the desired page is residing on secondary storage
- but there are no free frames on the free-frame list
    - all mem is in use



Need for page replacement.

need for page replacement

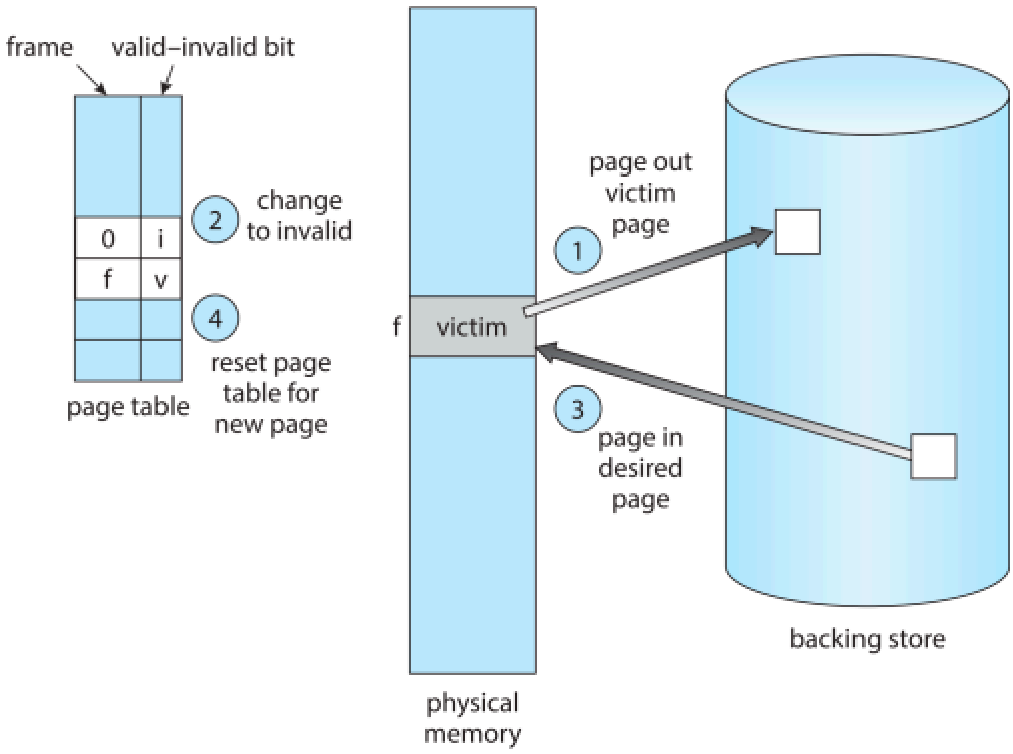
- algorithms
    - do we terminate it
    - do we swap out the whole process till there are more free frames
    - do we replace the page

    - we can find some page in memory but notreally in use, page it out
    - most operating systems now combine swapping pages with page replacement

we want an algor that has the minimum number of page faults

basic page replacement:

- prevent over-allocation of mem
    - modify page-fault service routine to include page replacement
- use modify (dirty) bit to reduce the overhead of page transfers
    - only modified pages are written to disk
    - unmodified pages have no need ot be written back to backing store since it wouldn't make a differnce
- page replacement completes separation b/w logicla mem and phys mem
    - large virt mem can be provided on a smaller phys mem



Page Replacement

1. Find the location of the desired page on disk.
2. Find a free frame:
   a) If there is a free frame, use it
   b) If there is no free frame, use a page replacement algorithm to select a **victim frame**
   c) Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables.
4. Continue the process from where the page occurred.

frame allocation algo determines how many frames given to each process

- some small processes are smaller
- are frames reserved
- etc

page-replacement algo

- which pages to replace
- want lowest page-fault rate on both first access and re-access

Page Replacement Algorithms

Page Replacement Algo Evalutation

evalutate algo by running it on a particular string of memory reference (reference string) and computing the number of page faults on that string

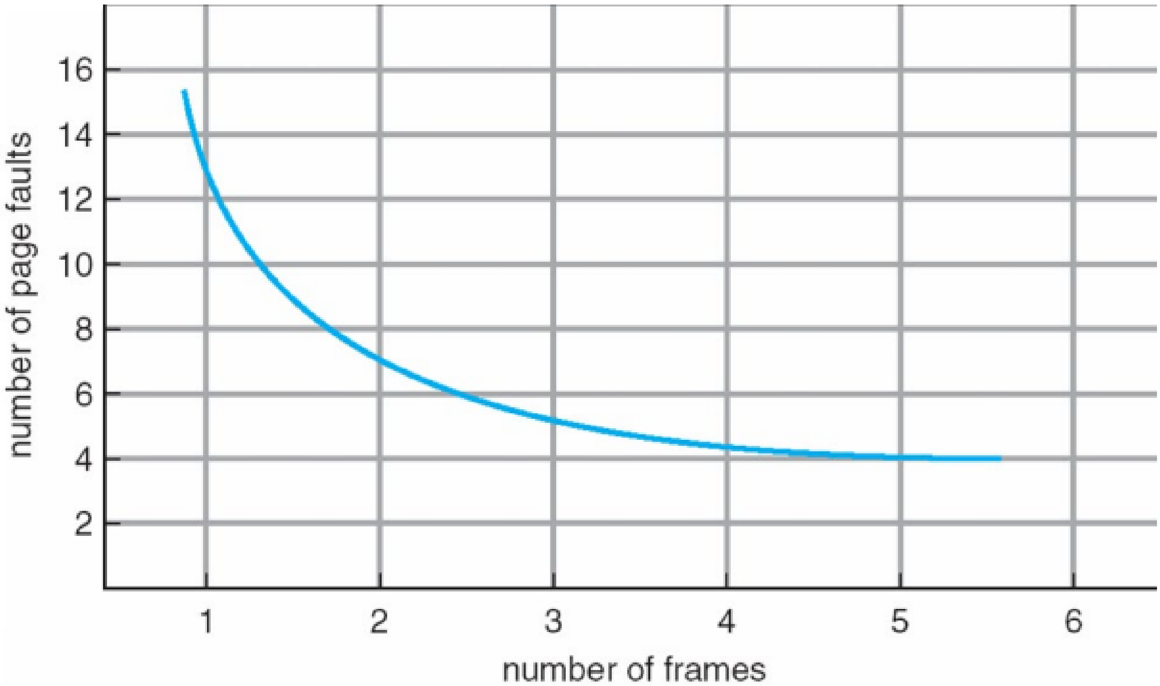
- string is just page numbers, not full addresses
- if we have a referece to a page `p`, then any references to page `p` that immediately follow will never cuase a page fault
- results depend on the number of frames available

- Example: if we trace a particular process, we may record the address sequence:

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

- This sequence is reduced to the following reference string:

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



Graph of Page Faults Versus The Number of Frames (in general)

Different Types

algos:

- FIFO
- Optimal
- Least Recently Used (LRU)
- LRU-Approximation
  - Second-Chance
  - Enhanced Second-Chance
- Counting-Based

In all our examples, the reference string of referenced page numbers is: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

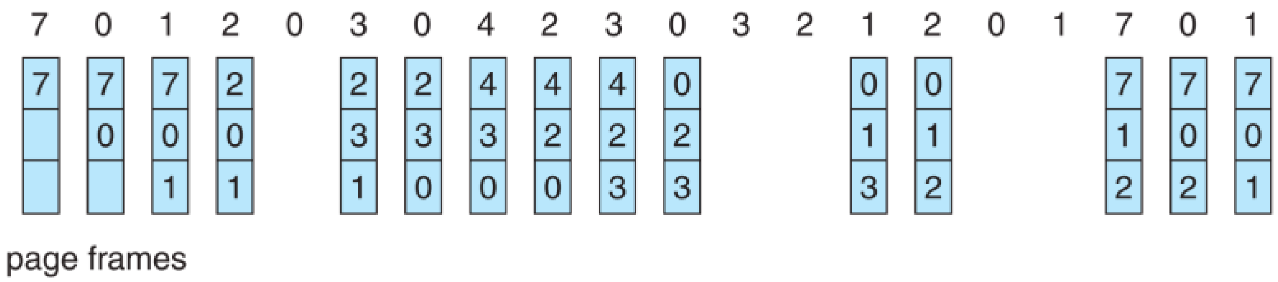
FIFO

first-in-first-out algo

Reference string: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

3 frames (3 pages can be in memory at a time per process)

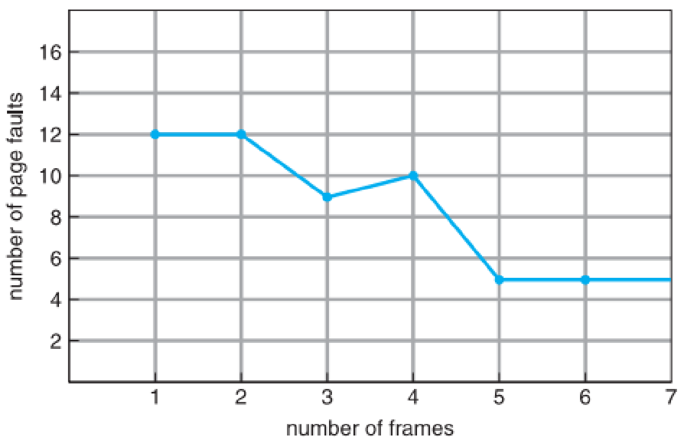
How to track ages of pages? Just use a FIFO queue.



FIFO: 15 page-faults

FIFO Illustrating Belady's Anomaly

- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults! ➔ **Belady's Anomaly**



Page-fault curve for FIFO replacement on a reference string.

Allocation of Frames

Thrashing

Allocating Kernel Memory