Chapter 10: Virtual Memory

Outline

Background
Demand Paging
Copy-on-Write
Page Replacement
Allocation of Frames
Thrashing
Allocating Kernel Memory
Other Considerations
Example

Objectives

To describe the benefits of a virtual memory system

To explain the concepts of demand paging, page-replacement algorithms, and allocation of page frames

To discuss the principle of the working-set model

To explore how kernel memory is managed

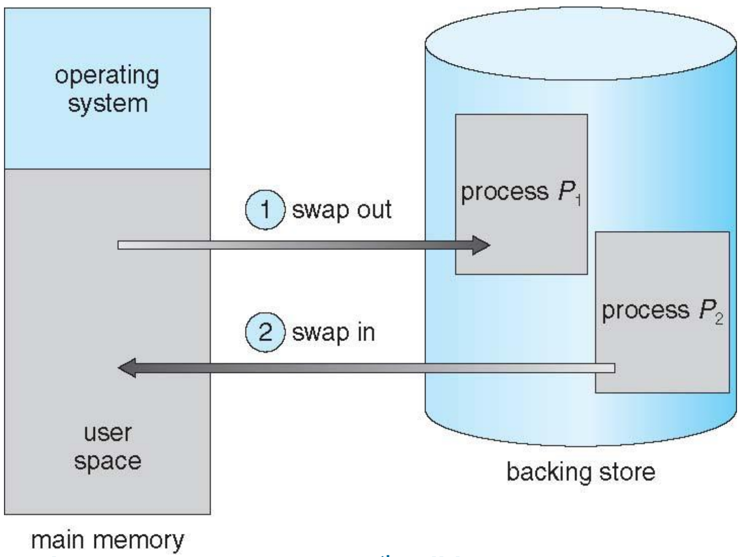
Multiprogramming

Multiprogramming

Multiple processes can reside in the memory at the same time

Problem: if a program needs to be run, but there is no enough free memory

One solution： swap out (换出) a process and swap in (换入) the target process



Backing store（后备存储器）

- is fast hard disk large enough to accommodate copies of all process memory images for all users
- has direct access to memory images

Swapping

A complete process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution

A swapper manipulates entire process swapping

Advantage

- Increase multiprogramming

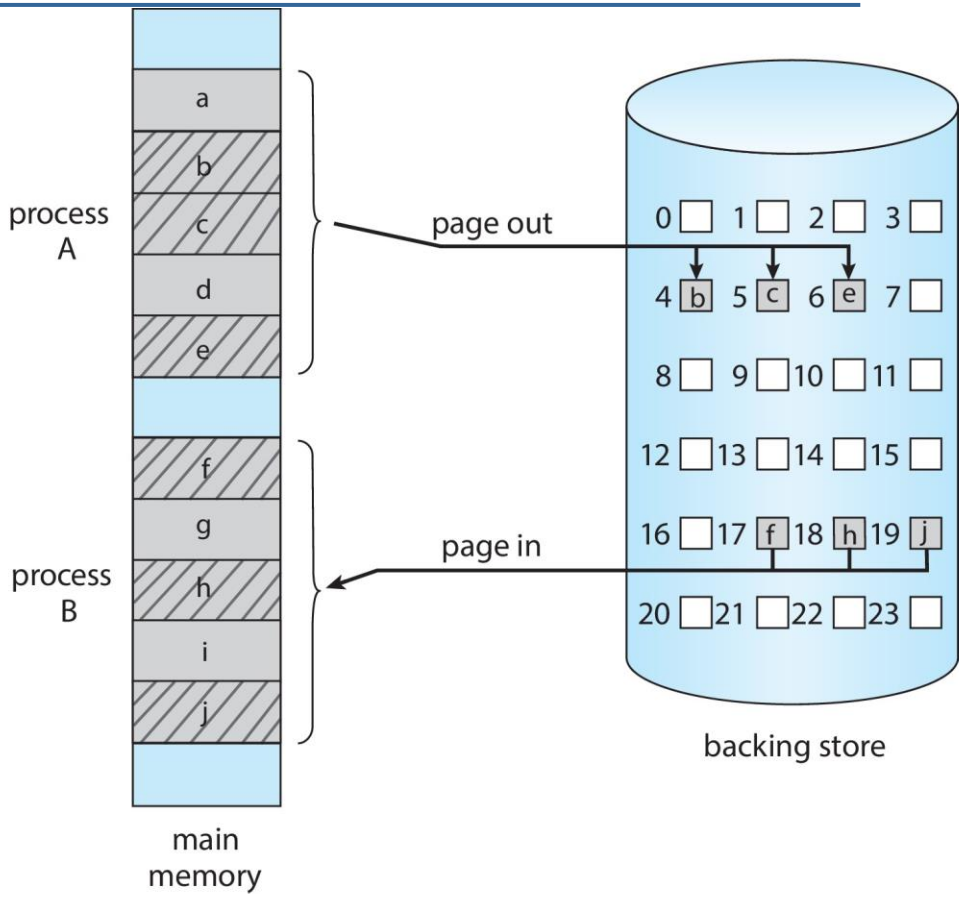
Disadvantage

- Context switch time can then be very high
- Total context switch time includes swapping time for the whole process

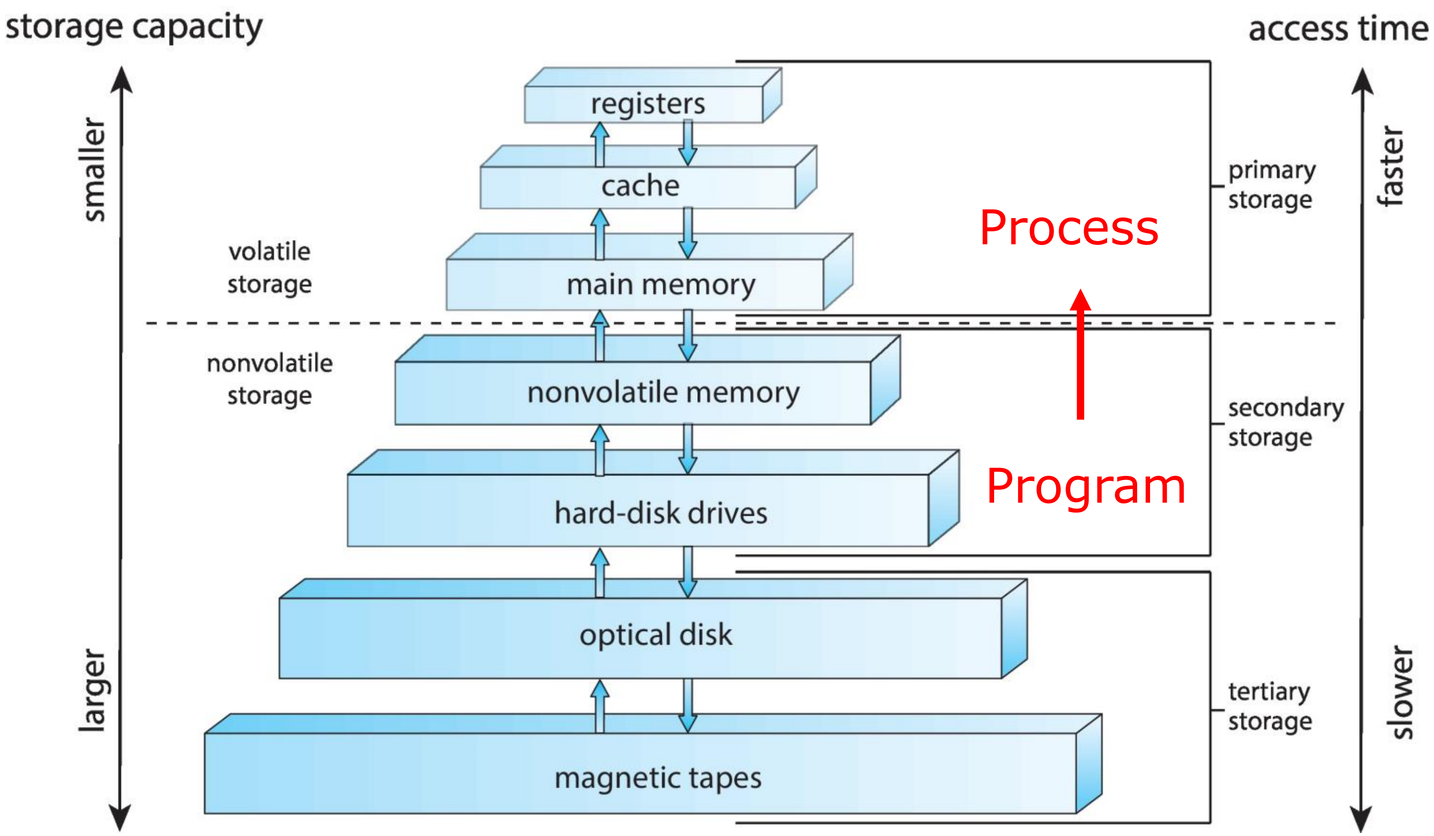
Can the swapping time be reduced by reducing the size of memory swapped?

Swapping with Paging

- When memory is low, unused pages are swapped to disk (instead of whole processes)
  - How to determine which pages are unused?
  - What happens when a process suddenly tries to access a page that was swapped out?



Review: Storage Hierarchy



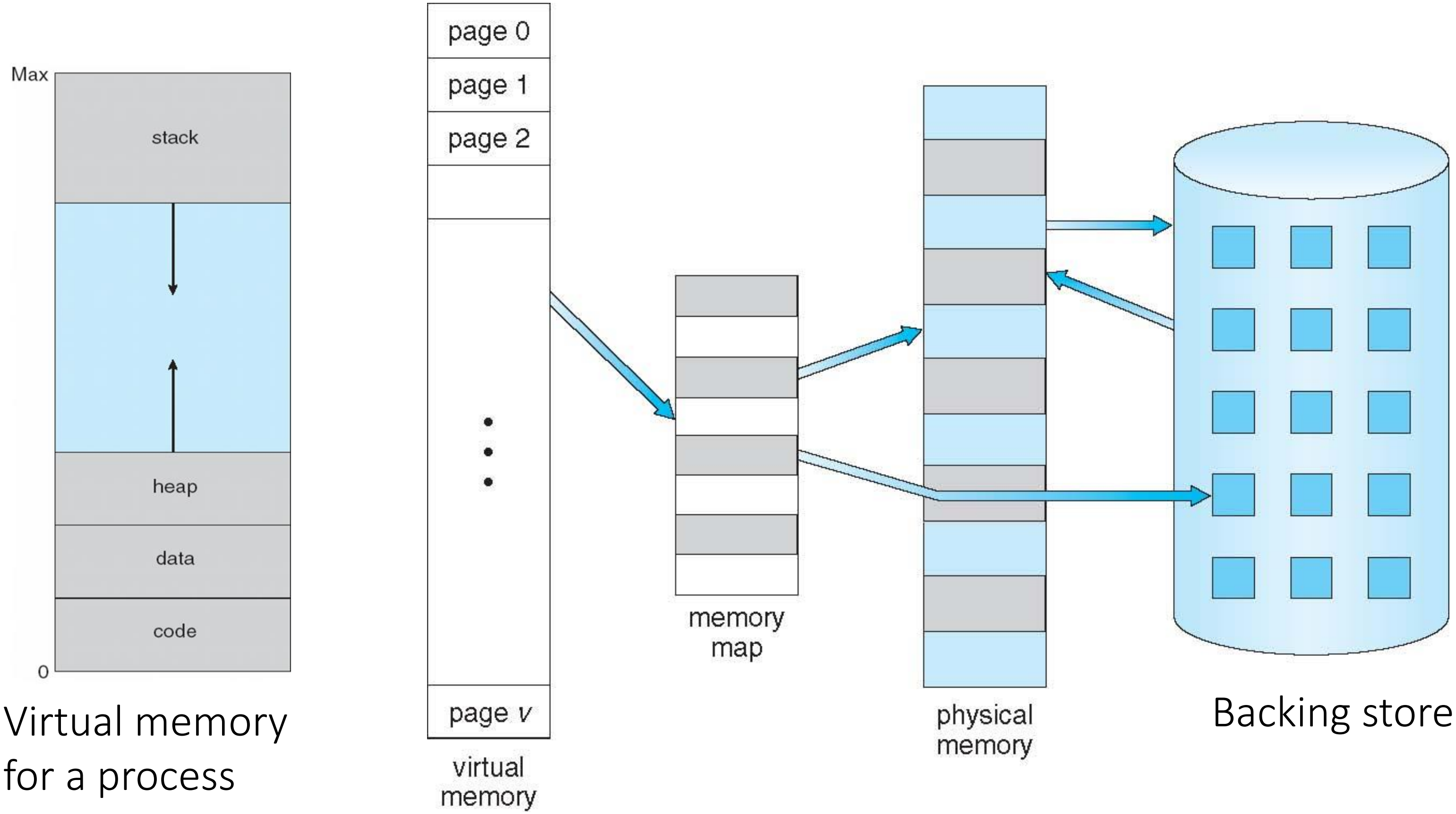
Chapter 9: how CPU accesses the process in the memory?

Chapter 10: Are all the code and data in the program needed at the same time in the memory?

Background

- Program can be partially loaded into memory
  - Error handling code, unusual routines, large data structures may not be needed for most of time
- Program length is no longer constrained by limits of physical memory
  - Each program takes less memory → more programs run at the same time
  - Less I/O needed to load programs into memory → each user program runs faster
- Virtual memory – separation of user logical memory from physical memory
  - Logical address space can therefore be much larger than physical address space
  - Only needed pages are loaded into memory

Virtual Memory That is Larger Than Physical Memory



On a modern computer,

the logical address space (virtual memory) is 2^{64} bytes = 16 billion GB, but the physical address space (RAM) is only a few GB (a few hundred GB for the biggest computers on earth).

Demand Paging

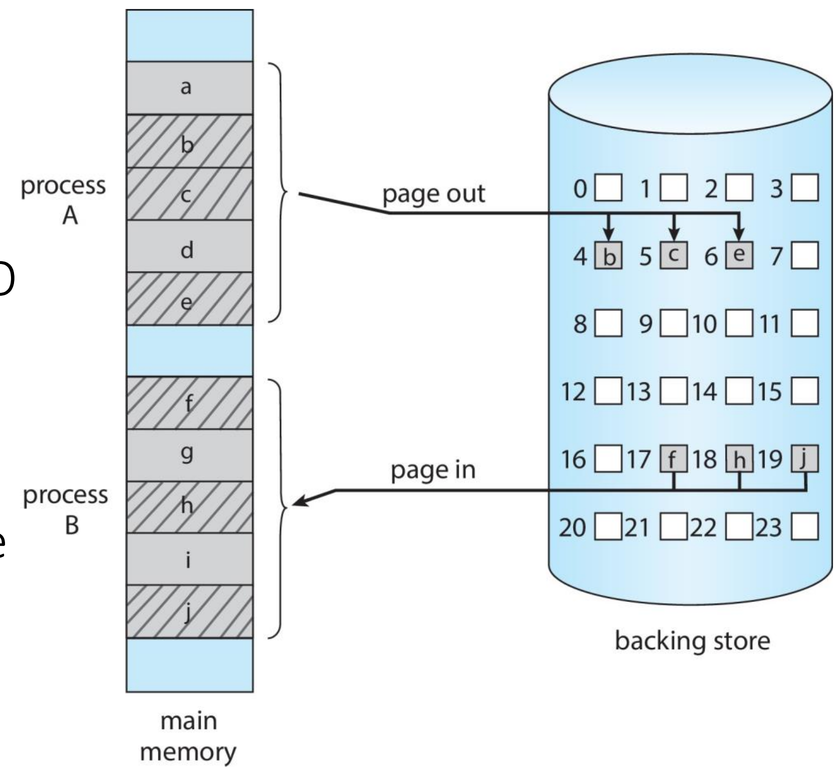
Demand paging

- Could bring a page into memory only when it is needed
- Advantages:
  - Less I/O needed, no unnecessary I/O
  - Less memory needed
  - Faster response
  - More users/processes

Page is needed (called a reference to the page

- invalid reference might be caused by
  - wrong address ⇒ process abort
  - not-in-memory ⇒ bring to memory

A pager can swap in and out one page at a time (not a whole process like swapper does).



Demand Paging: Basic Concepts

The pager brings in only those pages into memory that the process actually wants to use

Question: How to determine that set of pages?

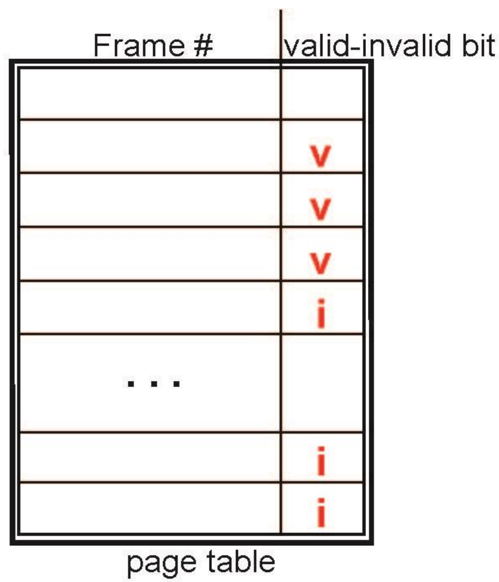
- Hardware support: MMU
  - If pages needed are already memory resident
    - No difference from non-demand paging
  - If pages needed are not in memory
    - Need to detect and load the pages into memory from storage

Valid-Invalid Bit

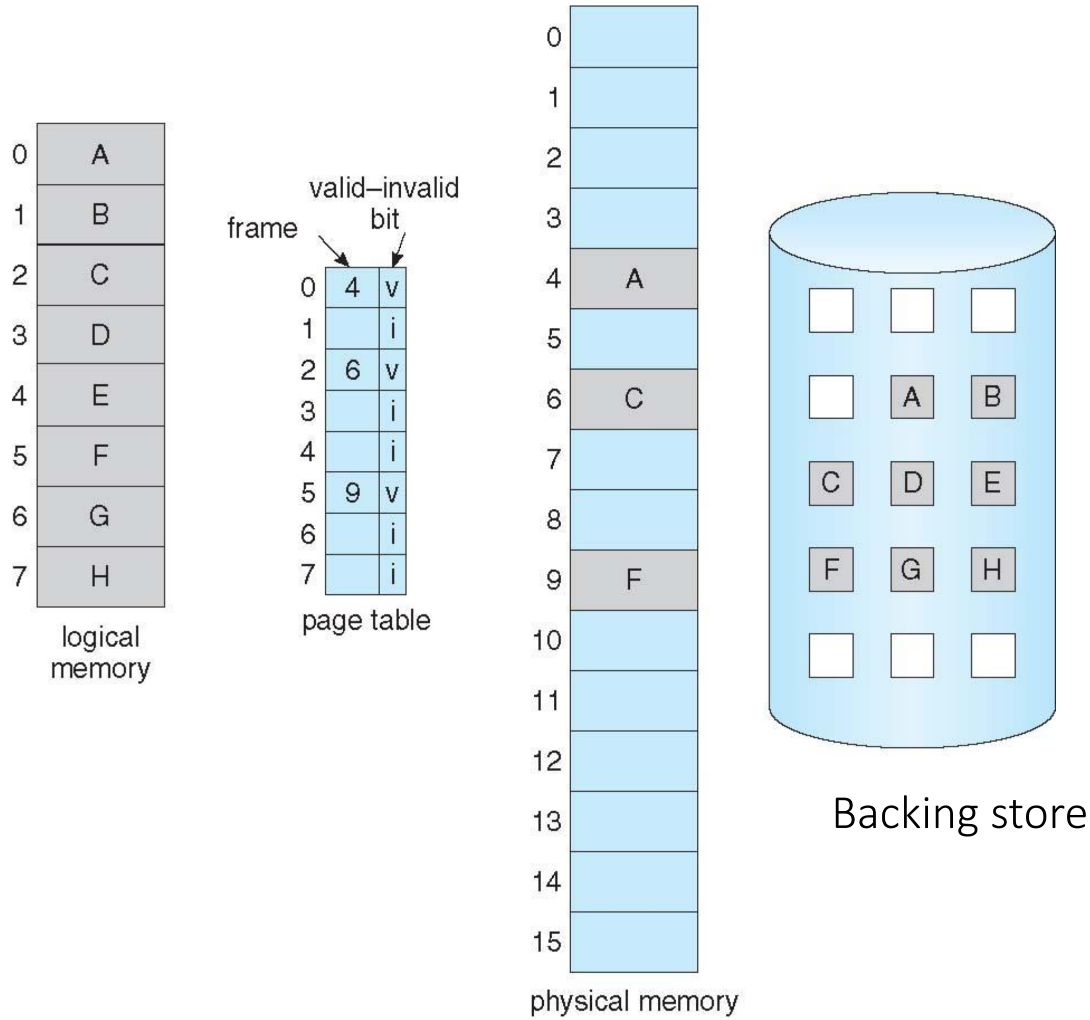
To check if a reference is valid or not, associate each page table entry with a valid–invalid bit

- *v*: the page is in-memory
- *i* : the page is
  - either not-in-memory, or
  - a wrong logical address

Initially valid–invalid bit is set to *i* on all entries



An Example: Some Pages Are Not in Main Memory



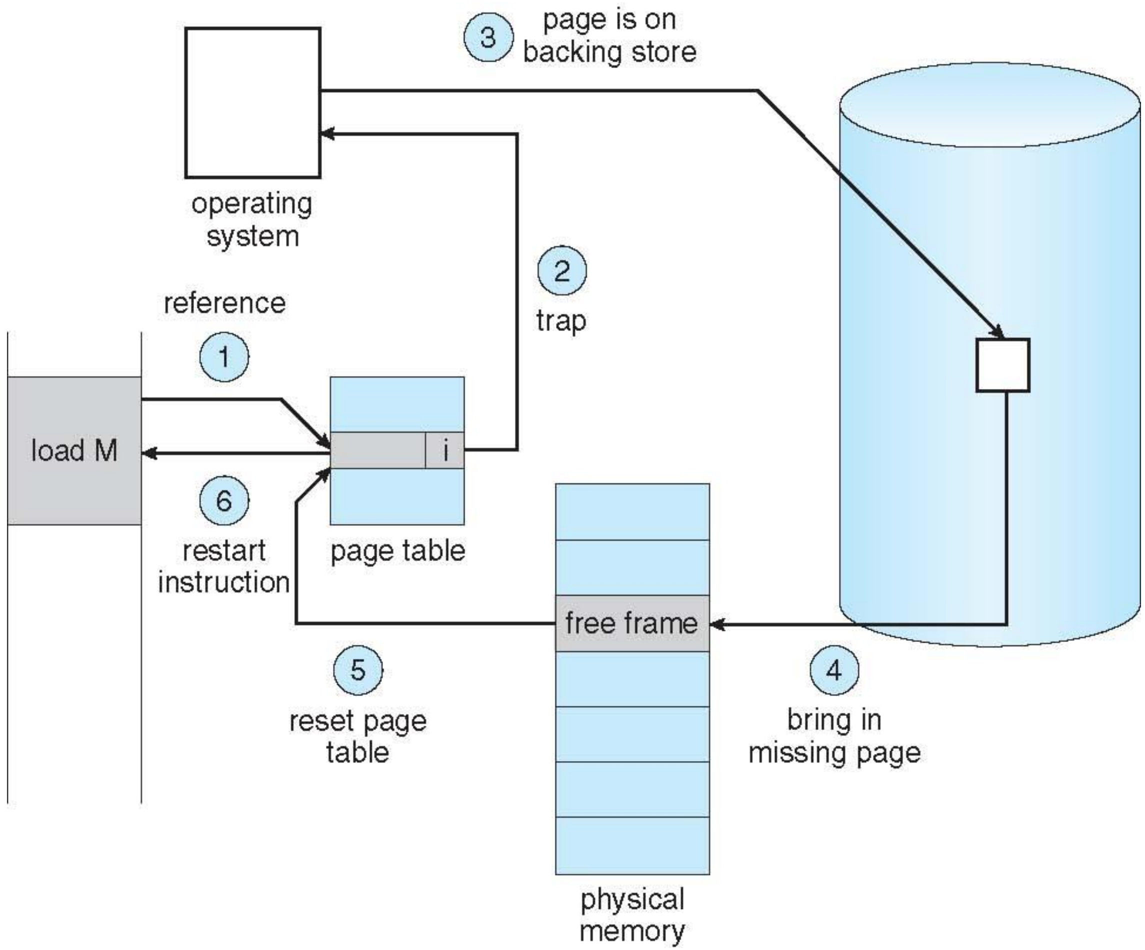
Pages 0, 2, and 5 are in the memory

Pages 1, 3, 4, 6, and 7 are not in the memory

Page Fault

- Page fault
  - Caused when a process makes a reference to a page which is not in RAM,
  - MMU detects that the page is marked as invalid in the process's page table,
- Two possibilities
  - the process is trying to access a page that does not exist at all
    - E.g., the process has pages for 1 to 10, but CPU wants to access page 12
    - Solution: Process aborts
  - The page is not in memory but is somewhere on the hard disk
    - E.g., the process has pages from 1 to 10, CPU wants to access page 2, but this page has not been loaded into RAM
    - Solution: Load the page from hard disk to RAM

Steps in Handling a Page Fault



1. CPU makes a reference to a page which is marked *i* in the page table
2. MMU traps to kernel
   a) if the reference page does not exist at all, stop the process;
   b) otherwise, go to 3
3. Find the page in the hard backing store
4. Find a free frame in the memory and load the page into that frame
5. Reset the page table such that the valid bit is *v*
6. Restart the process by re-executing the instruction which caused the page fault.

Aspects of Demand Paging

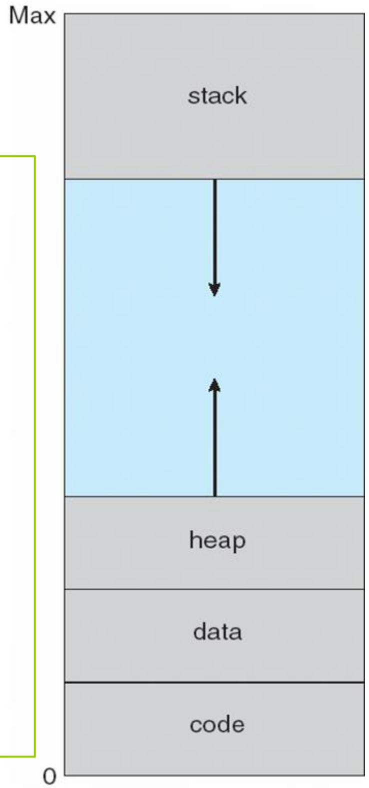
Pure demand paging

- Start process with *no pages* in memory (extreme case)
- Immediate page fault for the first instruction in the process

A given instruction could access multiple pages -> multiple page faults

- E.g., suppose the following code

```
int d; //global variable
int main(){
   int i;   //local variable
   int *p;  // pointer
   p = (int*) malloc(sizeof(int))
   … …
   //Thinking: how many pages are usually
   //accessed for the following instruction
   *p= i + d;
}
```



Performance of Demand Paging (Cont.)

- When there is a page fault, the time to load a page from hard disk is enormous compared with memory access
- Page Fault Rate $0 \leq p \leq 1$
  - if $p = 0$ no page faults
  - if $p = 1$, every memory reference is a page fault
- Effective Access Time (EAT)
  - EAT= $(1 - p)$ * memory_access_time + p * page_fault_time
    - Page_fault_time
      - time for page fault overhead in MMU and kernel , plus
      - time to swap a page out, plus
      - time to swap a page in

An Example: EAT

1s = 10^3 milliseconds
= 10^6 microseconds
= 10^9 nanoseconds

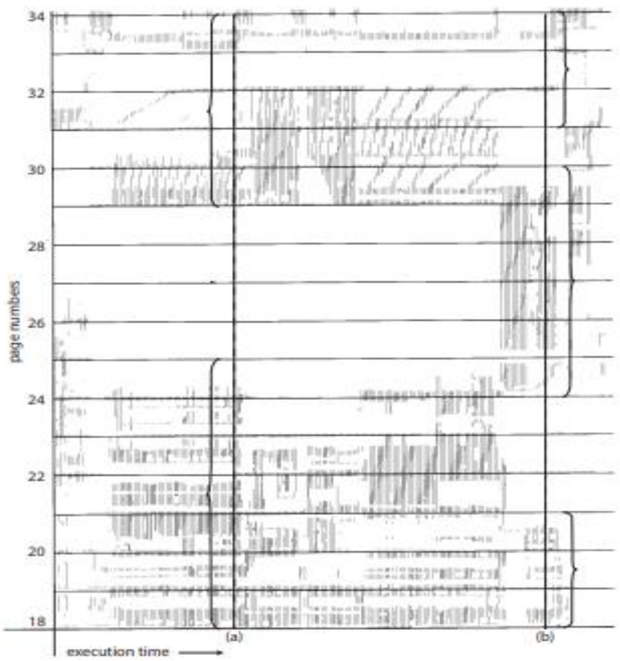
- Suppose
  - Memory access time = 200 nanoseconds
  - Average page-fault service time = 8 milliseconds
  - Then
    - EAT = $(1 – p)$ x 200 + p * 8 milliseconds = $(1 – p)$ x 200 + p x 8,000,000
      = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, i.e., $p = 0.001$,
  - EAT ≈ 200 + 0.001 x 7,999,800 ≈ 8.2 microseconds
  - A slowdown by a factor of 40 compared with one memory access
- To make the degradation < 10%, i.e., EAT < 200 + 200*10%
  - 200 + 7,999,800 x p < 220
    - $p < 0.0000025$, i.e., one page fault in every 400,000 memory accesses at most
- To achieve small p
  - locality of reference

Locality Reference

Locality of reference

The tendency of processes to reference memory in patterns rather than randomly

Results in reasonable performance from demand paging



Locality in a memory-reference pattern

- It will fault for the pages in its locality until all these pages are in memory
- It will not fault again until it changes localities
- Process migrates from one locality to another over time
- Localities may overlap (over time)

Locality Reference

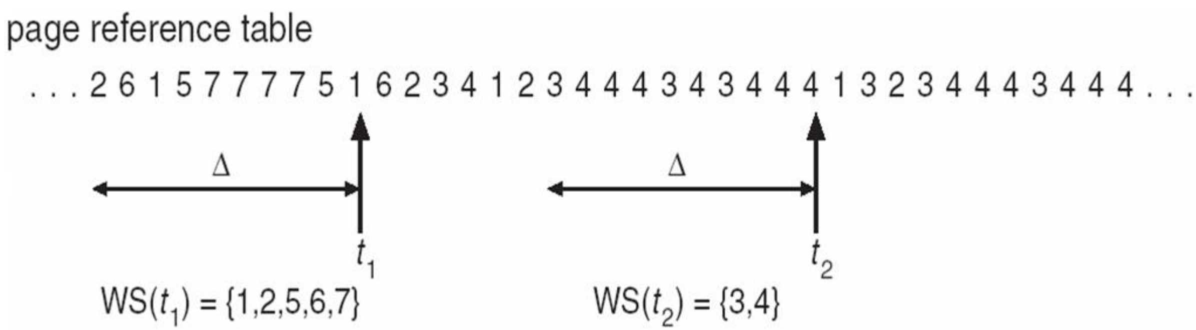
Working set

The set of pages in the most recent page references

Locality model

A model for page replacement based on the working-set strategy

At any point in time, a process usually uses only a small part of all its pages



Working-set model

Working-set window (Δ): a fixed number of page references

Keeping Track of the Working Set

Approximate the working-set model

Use a fixed-interval timer interrupt + a reference bit

Example: Δ = 10,000 references

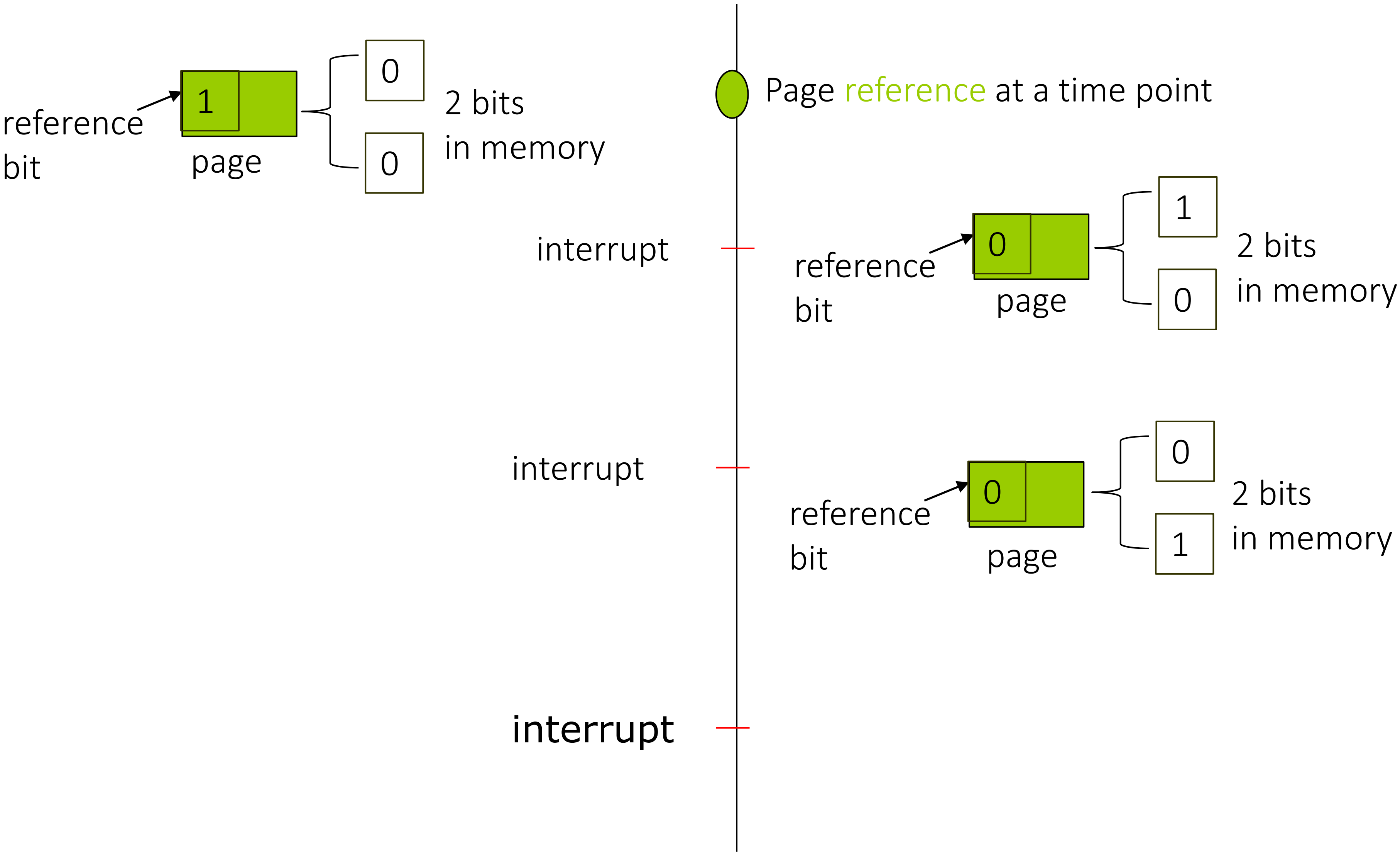
- Timer interrupts after every 5000 references
- Keep in memory 2 bits for each page, plus 1 reference bit
- Whenever a timer interrupts
  - shifting the first bit to become the second bit
  - copy reference bit into the first of the 2 bits
  - reset the value of all reference bits to 0
- If one of the bits in memory is 1 $\Rightarrow$ page in working set

This method is not completely accurate

Unable to tell where, within an interval of 5,000, a reference occurred.

An Example

Δ = 10,000 references, interrupts after every 5000 references



Demand Paging Optimizations

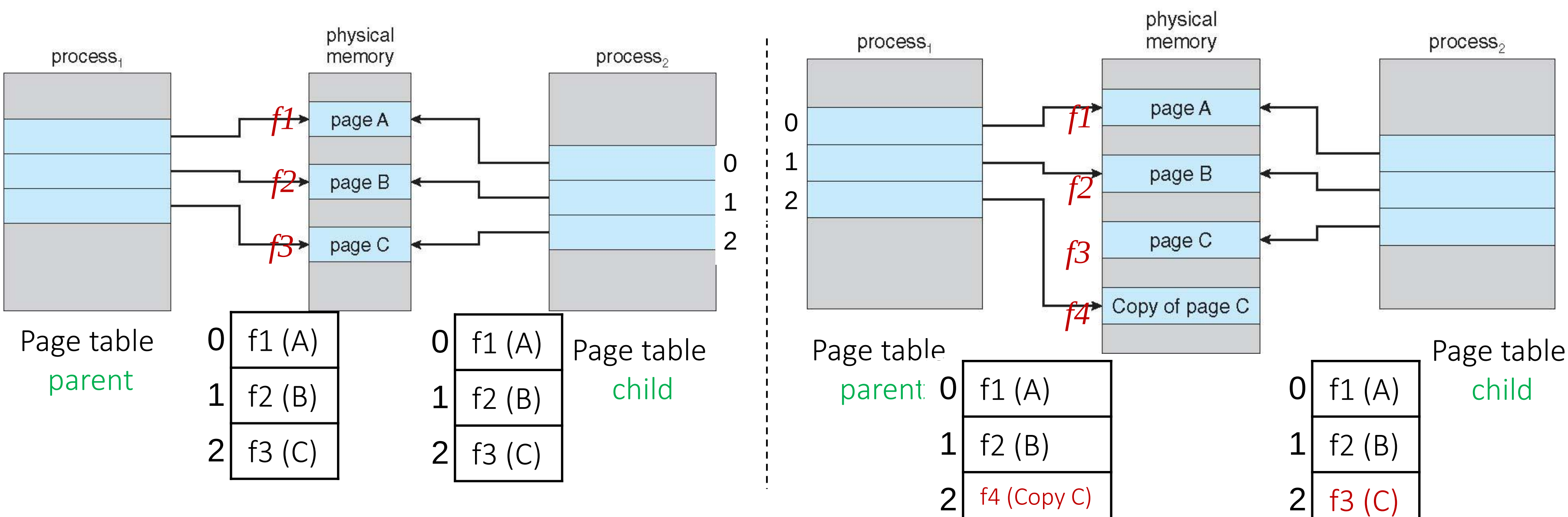
- Optimizations
  - Set a swap space on hard disk (not supported in mobile systems)
    - I/O faster than file system I/O
    - Less management is needed than file system
    - Copy entire process image to swap space at process load time, then page in and out of swap space (e.g., in older BSD Unix)
  - Discard the pages if they are not modified in the memory
    - When a page is needed to page out to free a frame, if its content is not changed, no need to transfer out

Copy-on-Write

Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory (vfork instead of fork)

When a child process is initially created,

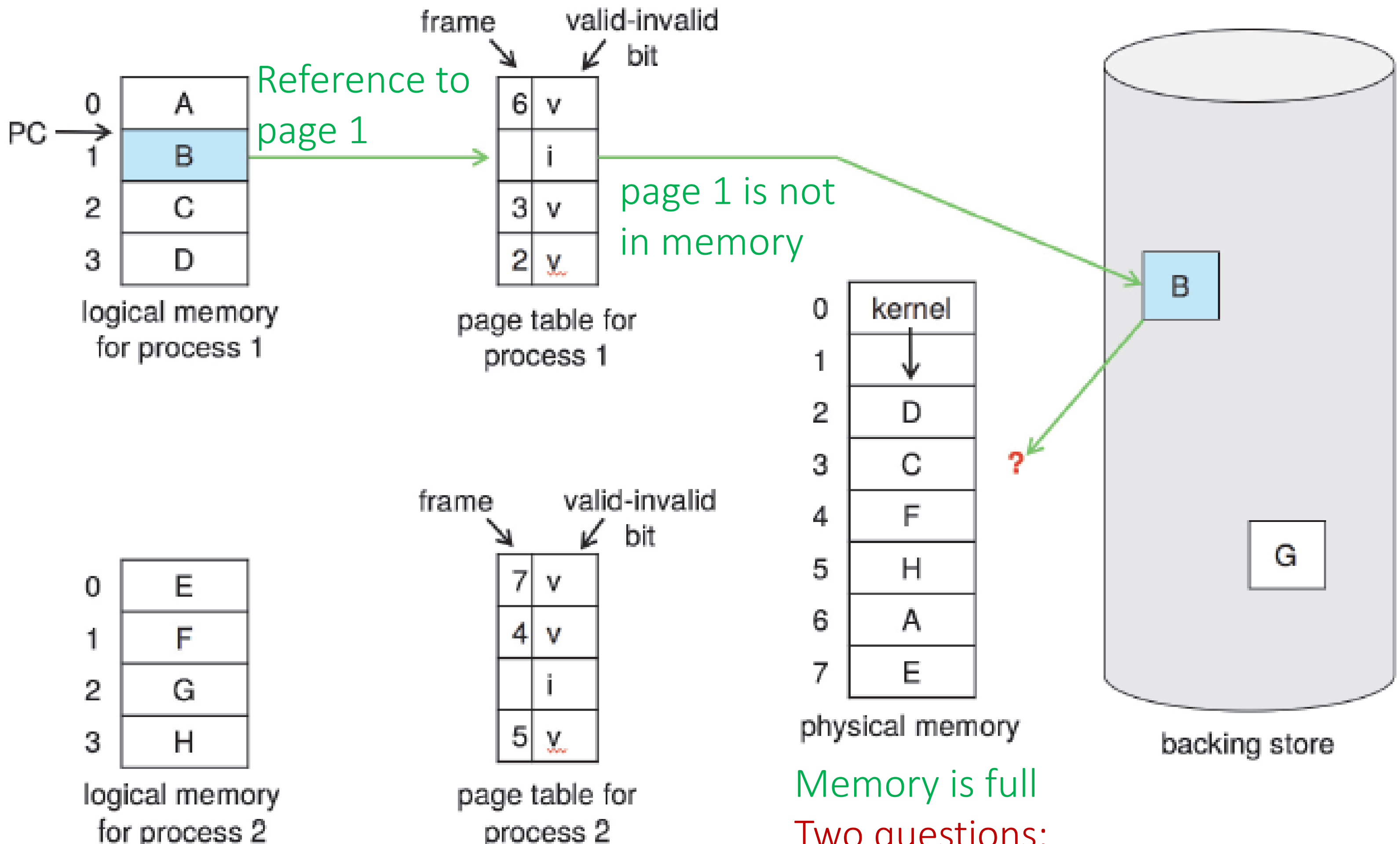
- only copy the page table of its parent
- share parents' pages
- the shared pages are write-protected (using protection bits of page tables).
- If either process modifies a shared page, then copy that page



What Happens if There is no Free Frame?

- When there is a page fault, a new page should be loaded in
  - Find a free frame for this new page to load in
  - Question: Memory has used up, i.e., no frame is available for a new page
  - Solution: swap pages out from memory
- Page replacement
  - Find some page in memory, but not really in use, page it out
  - Algorithm consideration
    - How to know which frames are free?
    - If no frame is free, which frames should be freed (i.e., which pages should be replaced, a victim)
      - expectation: minimum number of page faults
- Same pages may be brought into memory several times

Need For Page Replacement



Memory is full

Two questions:

1. Replace page from p1 or p2?
2. Which page to replace?

Global vs. Local Allocation

Question 1: from which process a page is replaced?

Global allocation

- Process selects a replacement frame from the set of all frames
- One process can take a frame from another process.
- The # of frames allocated to each process will change.

Local allocation

- Each process selects from only its own set of allocated frames.
- The # of frames allocated to each process does not change.

Global vs. Local Allocation

Which is better: Global or Local

- Global replacement
  - Process execution time can vary greatly
  - But greater throughput, so more commonly used
- Local replacement
  - More consistent per-process performance
  - But possibly under-utilized memory

Page Replacement

Question 2: Which pages to replace (be removed from RAM)?

Two kinds of pages are good candidates

- pages which are not used by a process right now
  - If a page has not been used for a while, maybe it will not be used in near future
- Pages that are not modified (called dirty) recently
  - These kinds of pages do not need to be paged out

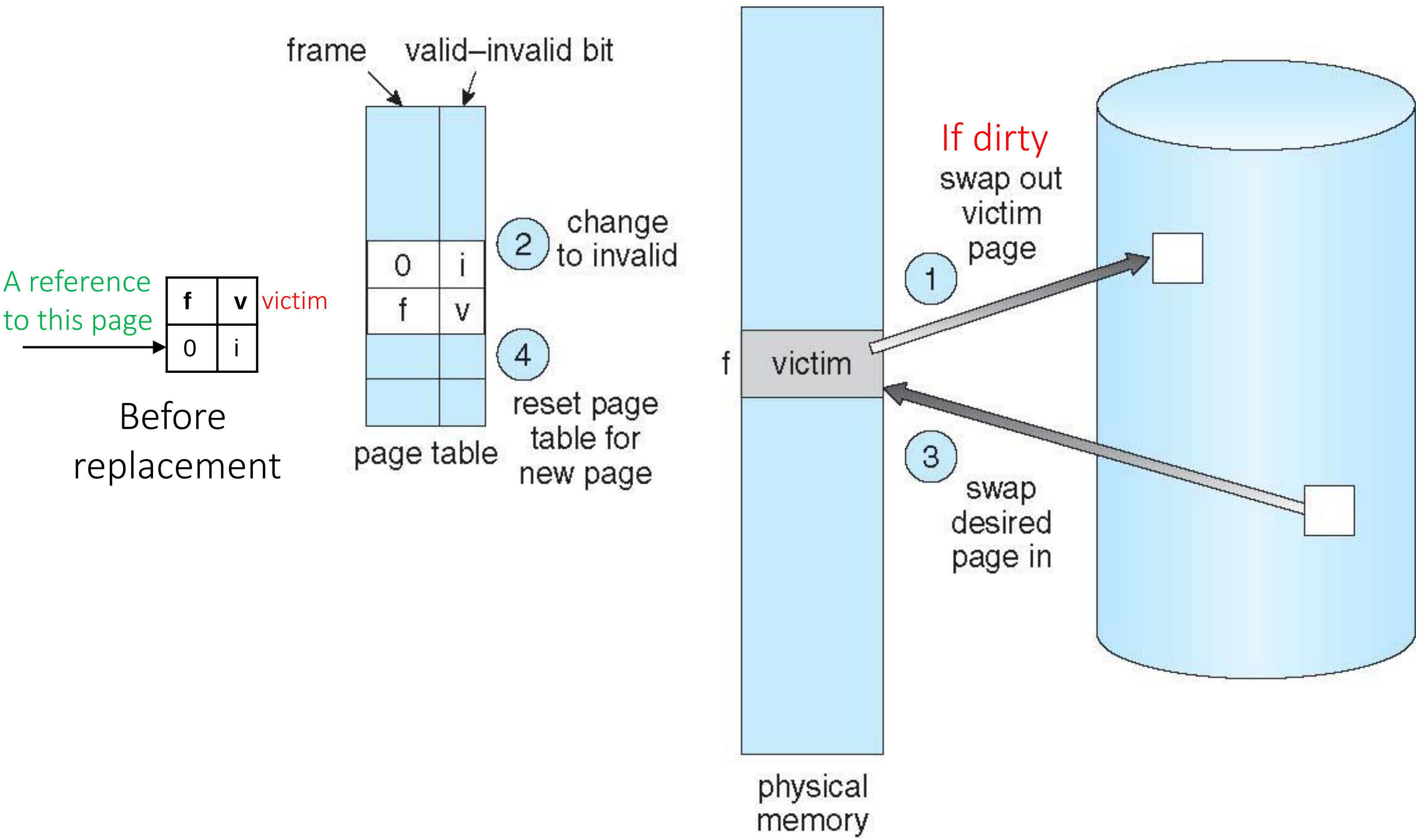
An algorithm is needed to find the right frame to free

- which pages have not been used recently?
- which pages have not been modified recently?
- “recent”: how to decide the period for “recent”?

Victim page

- The page which will be replaced

Page Replacement



Page and Frame Replacement Algorithms

- Frame-allocation algorithm determines
  - How many frames to give each process
- Page-replacement algorithm
  - Which frames to replace
  - Want lowest page-fault rate on both first access and re-access
- Algorithm evaluation
  - Run it on a particular string of memory references (reference string) and compute the number of page faults on that string
  - String is just page numbers, not full addresses
  - Results depend on number of frames available

Page Replacement Algorithms

- Algorithms
  - FIFO
  - Optimal
  - Least Recently Used (LRU)
  - Second chance
  - Enhanced second chance
- Reference string used in the examples
  - 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
  - Each number is the page number to reference

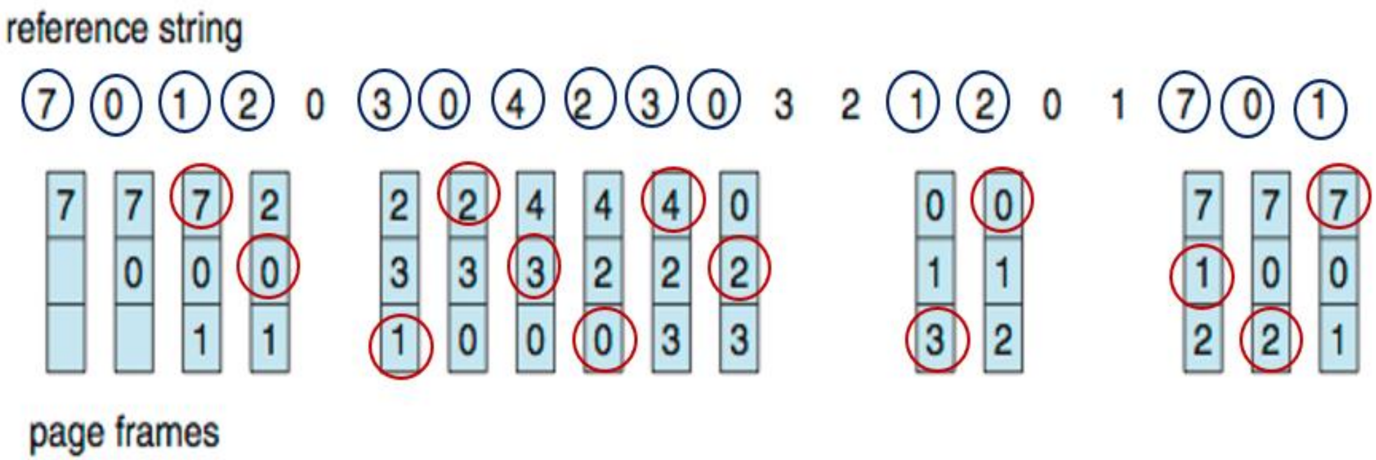
FIFO

- Rule
  - Replace the oldest one
- Implementation
  - OS maintains a circular queue of all pages
    - Page at head of the list: Oldest one
    - Page at the tail: Recent arrival
  - When there is a page fault
    - Page at the head is removed
    - New page added to the tail

First-In-First-Out (FIFO) Algorithm

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

3 frames (i.e., 3 pages can be in memory at a time)

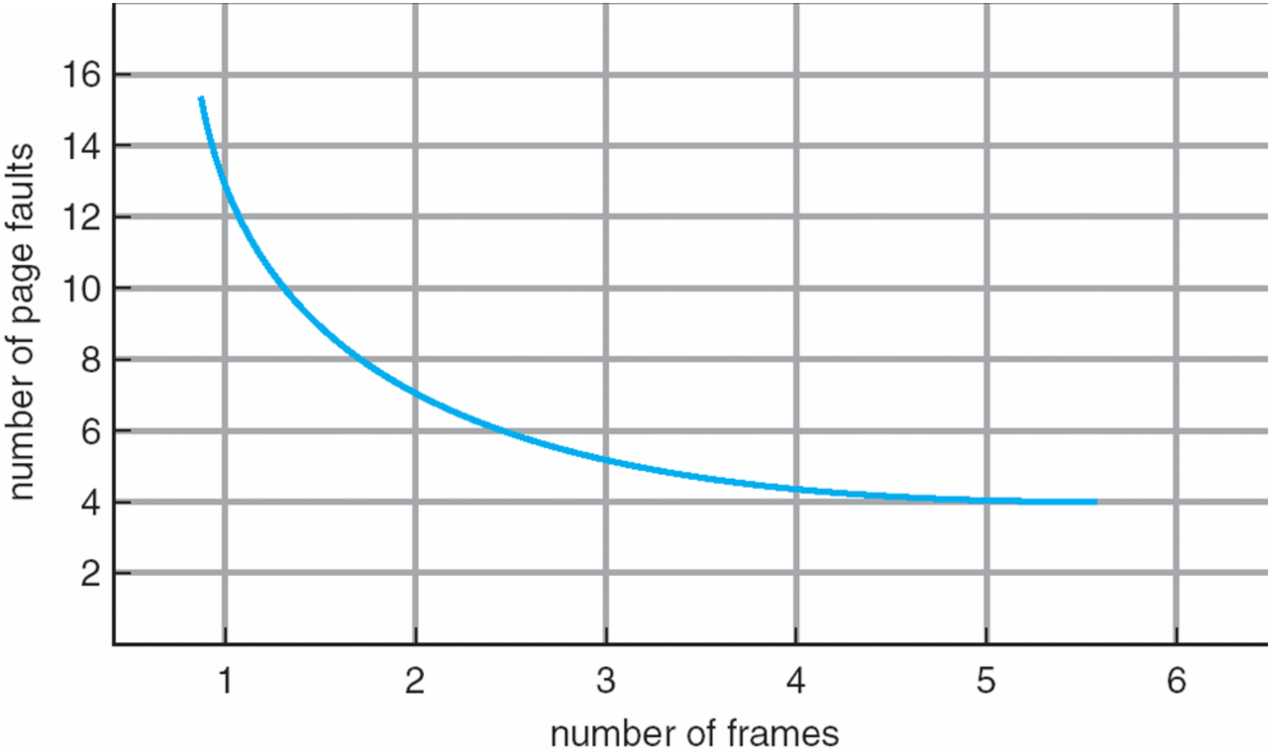


Total page faults: 15

Thinking:

Can adding more frames reduce the number of page faults?

Graph of Page Faults Versus The Number of Frames



Intuition but not Reality

FIFO Illustrating Belady's Anomaly

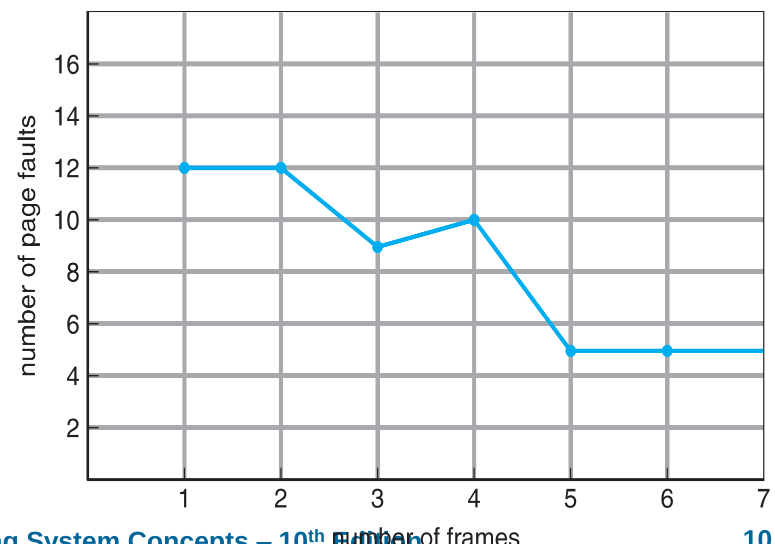
Consider 1,2,3,4,1,2,5,1,2,3,4,5

Adding more frames might cause more page faults!

1	2	3	4	1	2	5	1	2	3	4	5	page fault
1	1	1	4	4	4	5	5	5	5	5	5	
	2	2	2	1	1	1	1	1	3	3	3	9
		3	3	3	2	2	2	2	2	4	4	
1	1	1	1	1	1	5	5	5	5	4	4	
	2	2	2	2	2	2	1	1	1	1	5	10
		3	3	3	3	3	3	2	2	2	2	
			4	4	4	4	4	4	3	3	3	

3 frames

4 frames

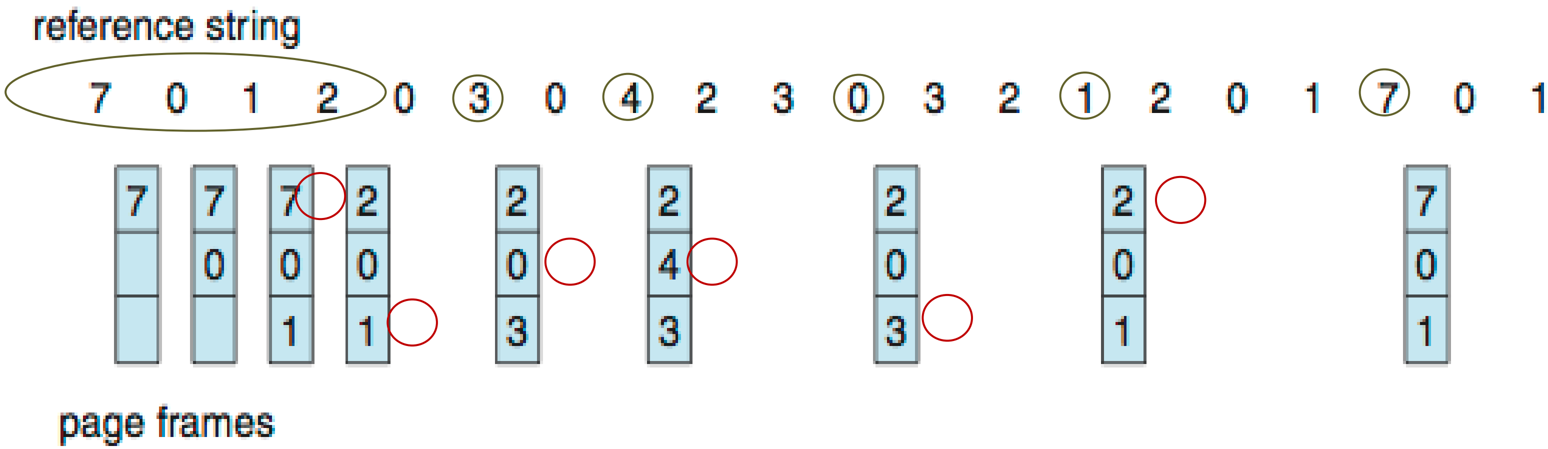


Belady's **Anomaly**

For some page-replacement algorithms, the page-fault rate may *increase* as the number of allocated frames increases.

Optimal Algorithm

Replace the page that will not be used for longest period of time



Total page faults: 9

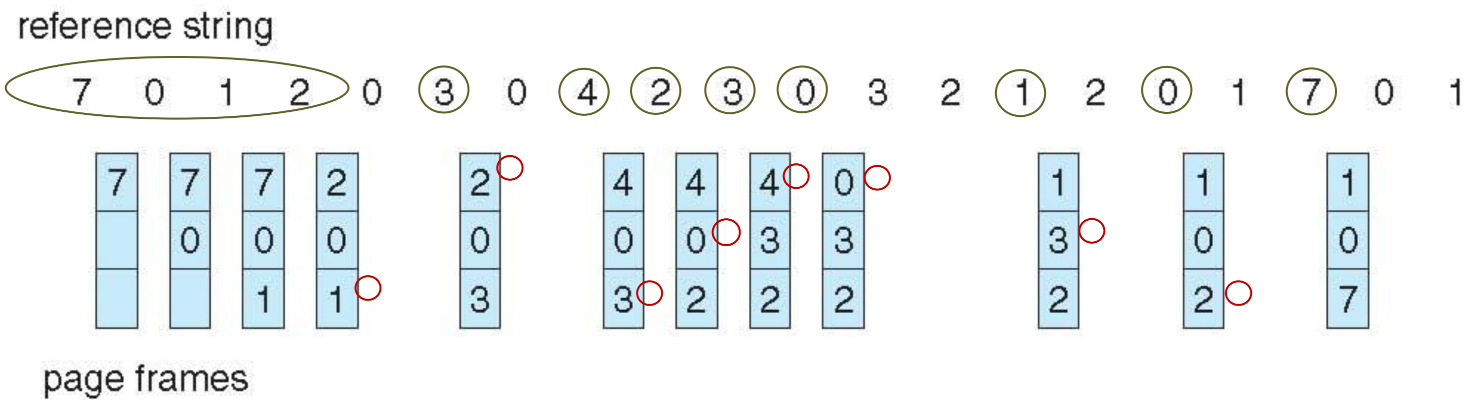
Problem

- Can’t read the future

This method can only be used to measure how other algorithms are close to the optimal

Least Recently Used (LRU) Algorithm

Replace page that has not been used in the most amount of time



Total page faults: 12

Better than FIFO(15) but worse than OPT(9)

LRU is a good algorithm and frequently used

LRU and OPT don’t have Belady’s Anomaly

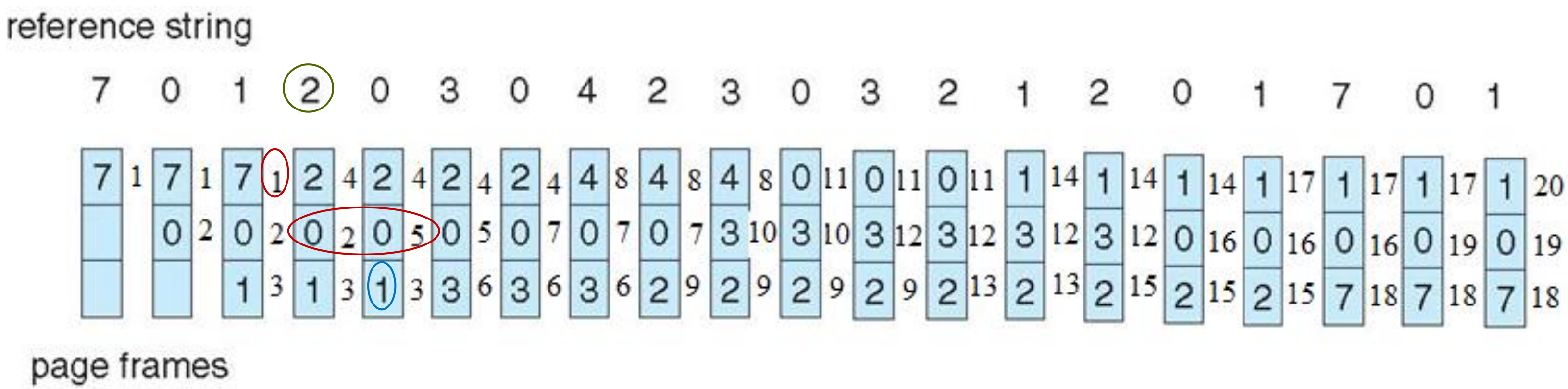
LRU Algorithm Implementation 1: Counter

Counter implementation

There is a global counter that increases by 1 whenever a page is referenced. Every page in the memory has its own counter

When a page is referenced, its counter is synchronized with the global counter

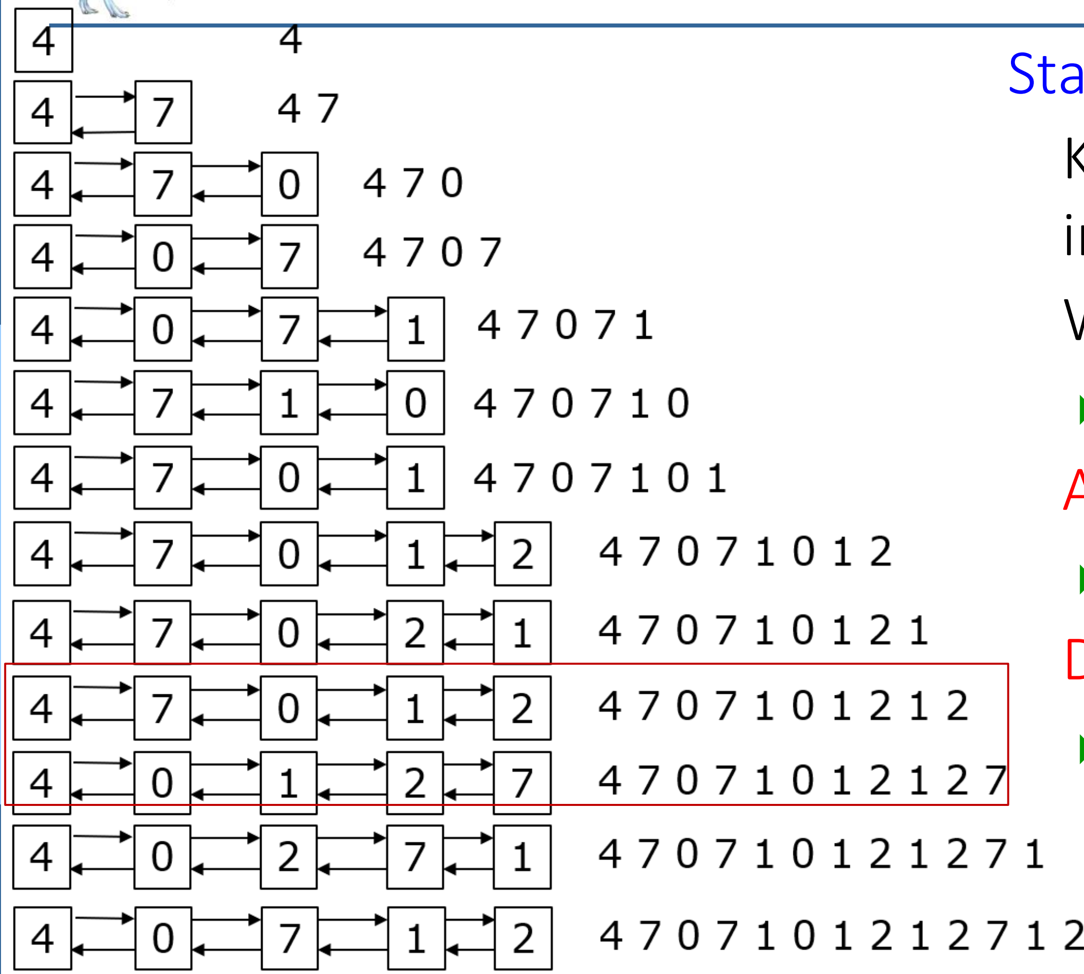
When a page needs to be replaced, find the page in the memory with the smallest value



Thinking:

What is the disadvantage of this algorithm (time issue)?

LRU Algorithm Implementation 2: Stack



Stack implementation

Keep a stack of page numbers in a double link form:

When a page referenced:

- move it to the top

Advantage

- No search for replacement

Disadvantage

- Each update of the stack might requires 6 pointers to be changed

Reference string:

4 7 0 7 1 0 1 2 1 2 7 1 2

Thinking:

1. In the stack, which one is least recently used? which one is the most recently used
2. Why needs 6 pointer changes for each update?

LRU Approximation Algorithms

LRU approximation algorithm 1

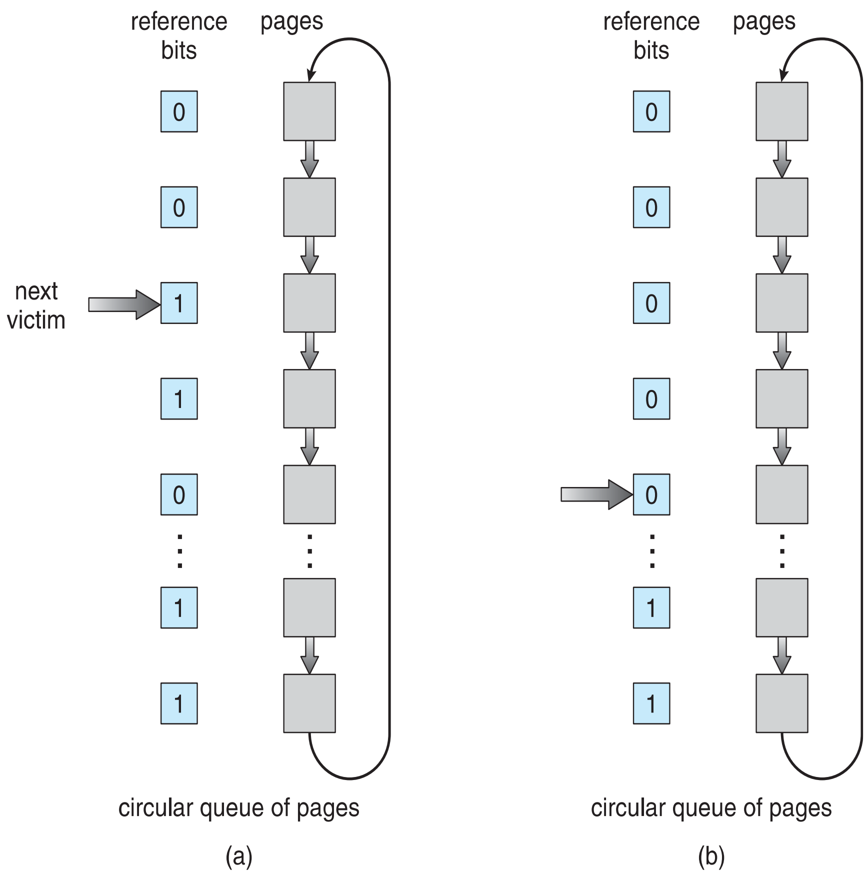
Usage of a reference bit

- Associate each page with a reference bit, initial value 0
- When a page is referenced, set the bit 1
- When a page is needed to be replaced, find the ones with reference bit 0 if exist one

Thinking:

What is the problem with this algorithm?

LRU Approximation Algorithms



LRU approximation algorithm 2

Second-chance algorithm

- Use a circular FIFO and reference bit for each page in memory
- If page to be replaced has
  - reference bit = 0
    - replace it
  - reference bit = 1
    - set reference bit to 0
    - search for the next page

LRU Approximation Algorithms

- Enhanced second-chance algorithm
  - Improve algorithm by using reference bit and modify bit for each page in memory, i.e., an ordered pair (reference, modify)
  - All pages in memory fall into four classes
    1. (0, 0) neither recently used not modified – best page to replace
    2. (0, 1) not recently used but modified – not quite as good, must write out before replacement
    3. (1, 0) recently used but clean – probably will be used again soon
    4. (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement – worst page to replace
- When page replacement called for, replace page in lowest non-empty class in clock scheme
  - Disadvantage
    - Might need to search circular queue several times

Other Algorithms

- Counting algorithms
  - Keep a counter of the number of references that have been made to each page
    - Least Frequently Used (LFU) Algorithm:
      - replaces page with smallest count
    - Most Frequently Used (MFU) Algorithm:
      - based on the argument that the page with the smallest count was probably just brought in and has yet to be used
- None of these counting algorithms are common

Page-Buffering

- More strategies (algorithms) in improving the efficiency of page replacement
- Page-buffering consideration
  - Keep a pool of free frames always
    - Whenever it is possible, select a victim to evict **(逐出)** and add it to free pool
    - When convenient, evict the victim
    - When frame needed, read page into free frame
    - Advantage: Reduce time to find a free frame at a page fault
  - Expansion
    - Keep a list of modified pages
      - Whenever the paging device is idle, write the modified page to the disk. Its modify bit is then reset to 0
    - Keep free frame contents intact **（完整的，内容未被清除的）**
      - Reduce penalty if wrong victim frame was selected
      - If the page is referenced again, no need to load from the disk

Allocation of Frames

For performance reason, each process needs minimum number of frames

- This number is defined by the computer architecture (CPU)
  - Example: IBM 370 – 6 pages to handle SS MOVE instruction
    - instruction might span 2 pages
    - 2 pages to handle *from*
    - 2 pages to handle *to*

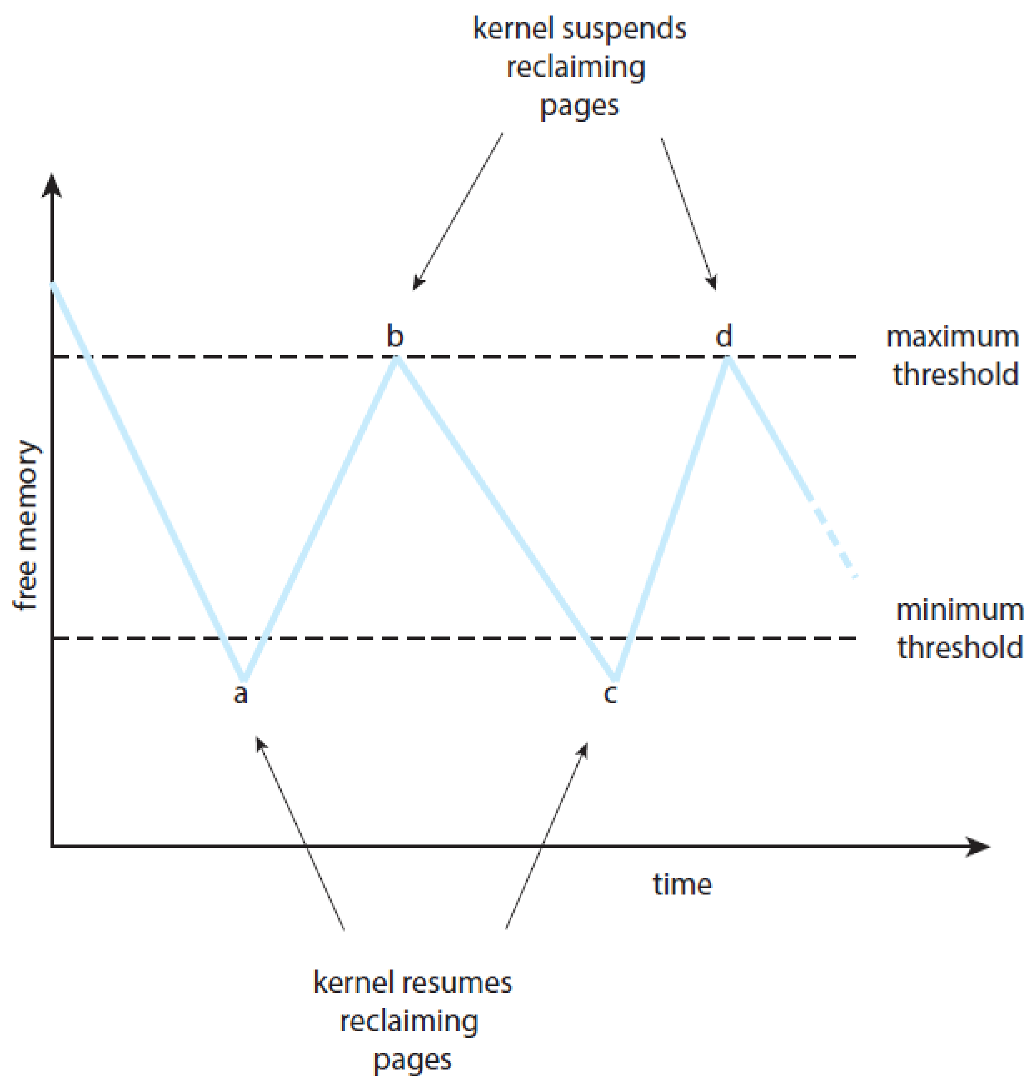
The maximum number of frames is defined by the amount of total frames in the system (i.e., physical memory, RAM)

Allocation of Frames

Two major allocation schemes

1. Fixed allocation
   - Equal allocation
     - Each process is allocated same number of frames
     - Disadvantage
       - Space waste for small process
   - Propositional allocation
     - Allocate frames according to the size of a process
     - Disadvantage
       - Process size might be changed during the execution
2. Priority allocation
   - the ratio of frames depends on the combination of size and priority of a process
   - Replace the pages of process with lower priority

Allocation of Frames

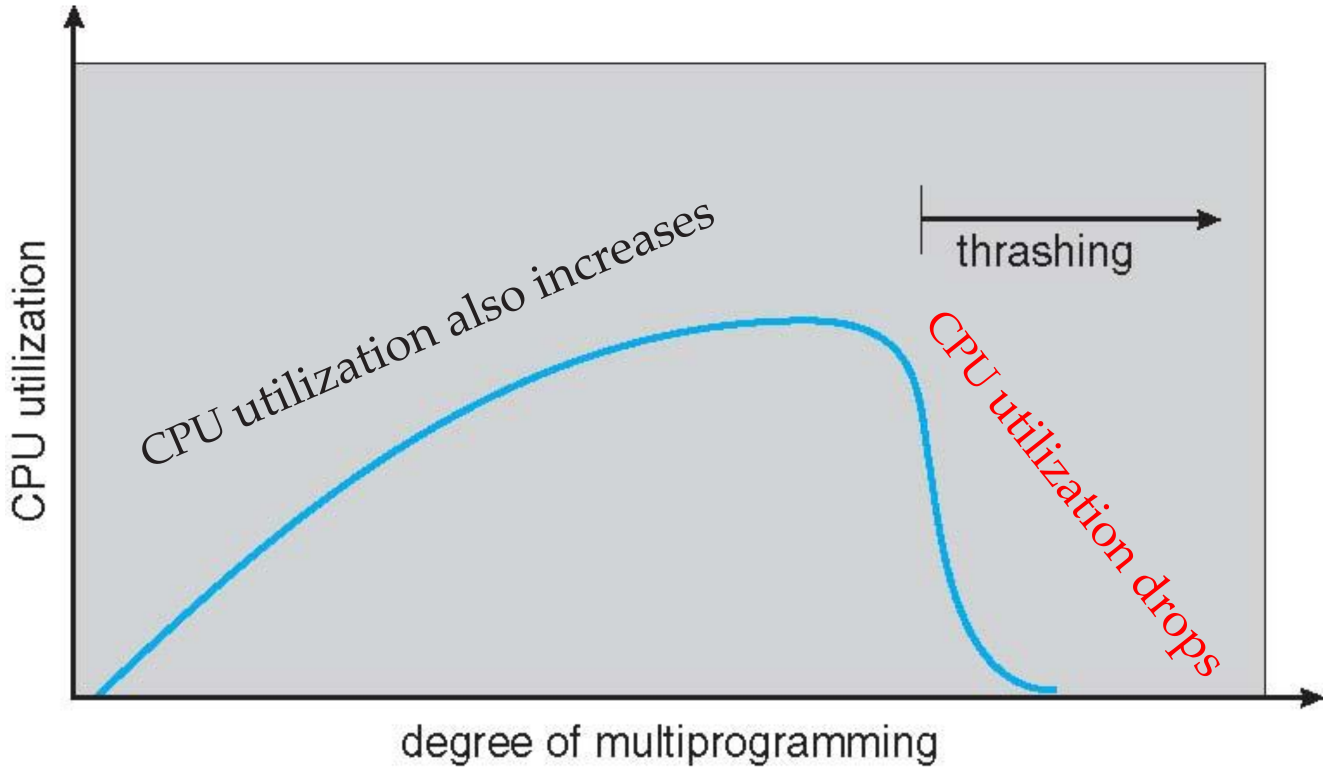


Page replacement happens before the number of frames falls below a certain threshold

Thrashing(系统颠簸)

- If a process does not have “enough” frames in memory, the page-fault rate is very high
  - Replace page frequently
  - The replaced page might be used again
  - This leads to:
    - Low CPU utilization
    - Operating system keeps adding new process, trying to increase CPU utilization
    - Things get worse -> entering a bad cycle
    - Thrashing
      - A process is busy swapping pages in and out, instead of doing useful work.

Thrashing (Cont.)



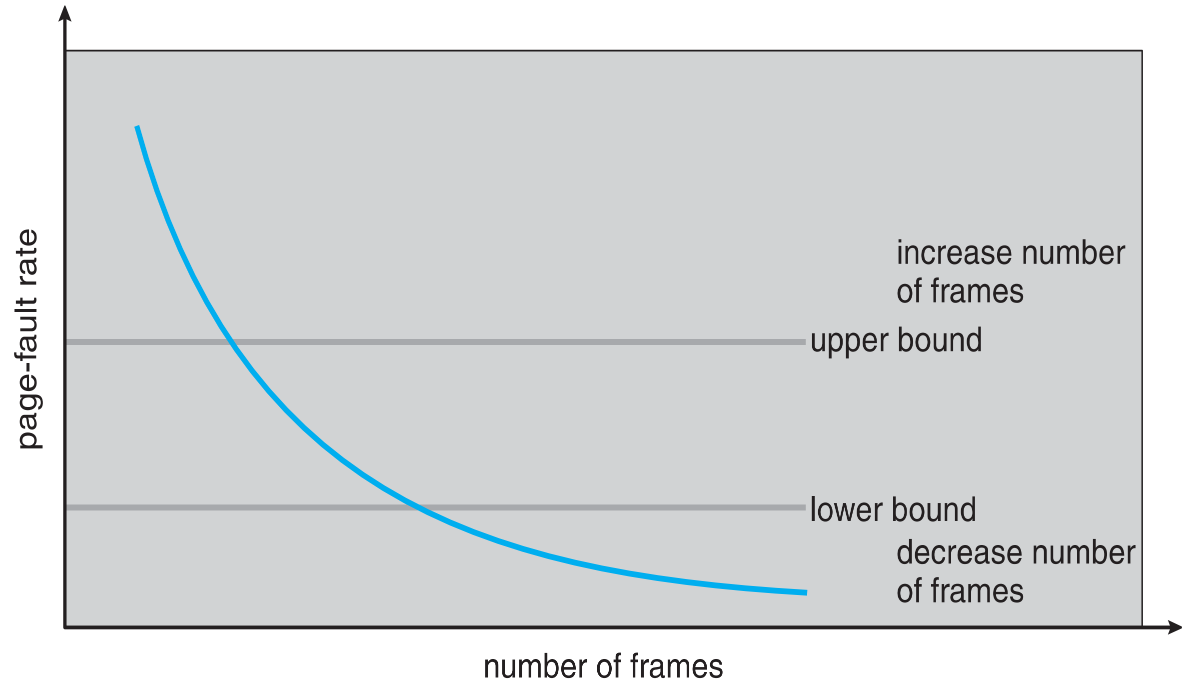
Solutions to thrashing

1. Decrease the degree of multiprogramming
2. Establish "acceptable" page-fault frequency (PFF) rate and use local replacement policy
3. Install enough physical memory (hardware)
4. Install faster hard disk

Page-Fault Frequency

If actual rate too low, process loses frame (we think too many frames assigned, remove some frames)

If actual rate too high, process gains frame



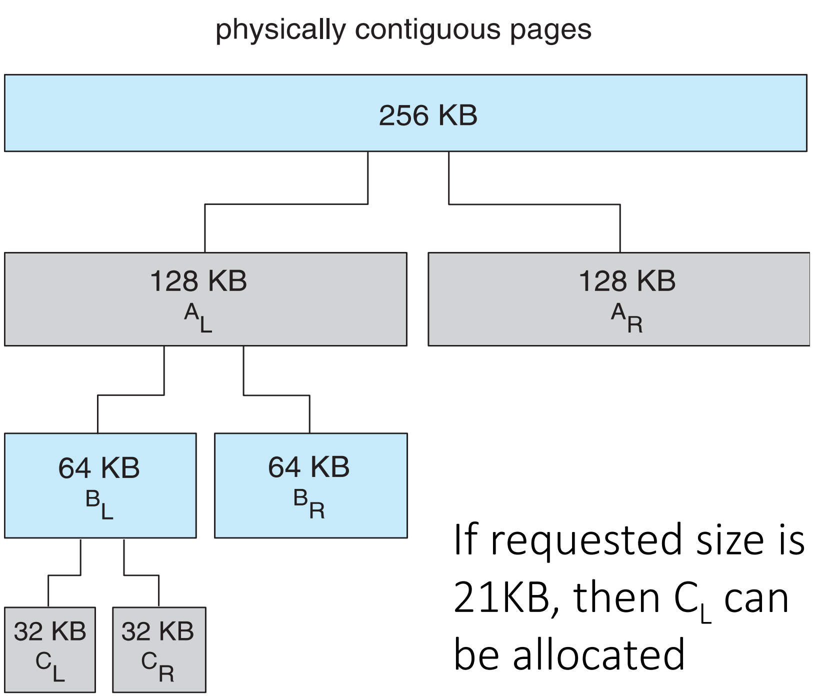
Allocating Kernel Memory

- Allocating memory for Kernel is different from the allocation of memory for user applications
- Special features of kernel
  - Kernel requests memory for structures of varying sizes,
    - E.g.,
      - structure for PCB
      - structure for file descriptor
  - There are multiples instances for each structure,
    - e.g., there is a PCB for each process
  - Memory needs to be contiguous

Allocating Kernel Memory: Two Methods

Two allocation methods

1. Buddy system
   - Allocates memory from fixed-size segment consisting of physically-contiguous pages
   - Power-of-2 allocator : memory allocated in units is sized as power of 2, one smallest but bigger than requested size



If requested size is 21KB, then C_L can be allocated

Advantage:

Unused chunks can be merged to a bigger one

Thinking:

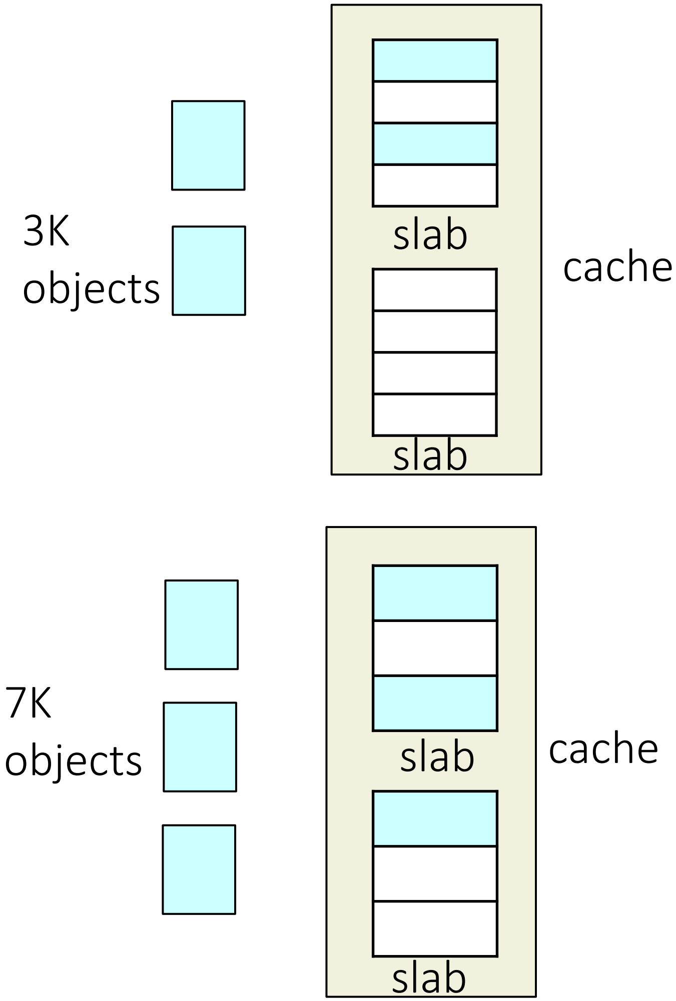
Disadvantage?

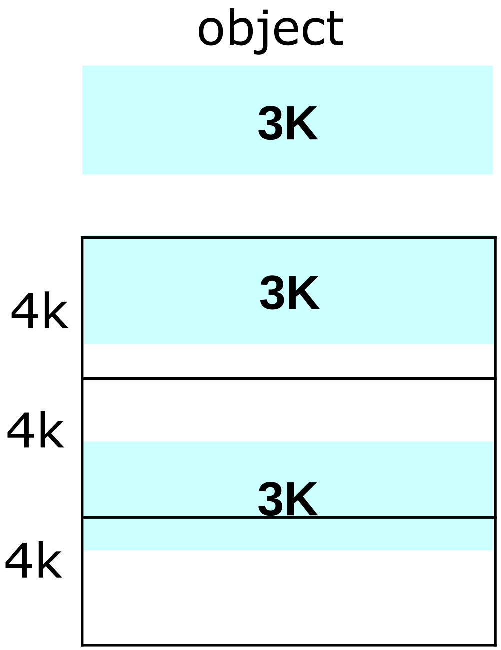
Allocating Kernel Memory: Two Methods

Two allocation methods

2. Slab allocator
   - A cache
     - is used for unique kernel data structure
     - consists of one or more slabs
     - A slab is one or more physically contiguous pages
       - filled with same kind of objects – instantiations of the data structure,
       - Each object has two status: free and used
     - If slab is full of used objects, next object is allocated from empty slab
       - If no empty slabs, new slab allocated

Slab Allocation: An Example





A slab
(3 continuous pages, each page: 4K)

Thinking:

How many objects in total can be put in this slab

Other Considerations

- To reduce page faults
  - Pre-paging
    - Pre-page all or some of the pages a process will need, before they are referenced
    - If pre-paged pages are unused, I/O and memory was wasted
    - Need to balance
  - Page size selection
    - Fragmentation
    - Page table size
    - Need to balance
  - TLB reach
    - Depend on page size and TLB size
    - Need to balance
  - Program structure
    - What can programmers do?

Other Issues – Program Structure

Two programs (assume data are stored row by row)

Program 1: accessing data column by column

```
for (j = 0; j <128; j++)
   for (i = 0; i < 128; i++)
         data[i][j] = 0; //each i is in different page
```

128 x 128 = 16,384 page faults!

Program 2: accessing data row by row

```
for (i = 0; i < 128; i++)
   for (j = 0; j < 128; j++)
         data[i][j] = 0; //each j is in different page
```

128 page faults only!

data[0][0]
data[0][1] .....
data[1][0]
data[1][1] .....
data[2][0]
data[2][1] .....
data[3][0]
data[2][1] .....

Example: Microsoft Windows

- Uses demand paging with clustering.
  - Clustering brings in pages surrounding the faulting page
- Processes are assigned working set minimum and working set maximum
  - Working set minimum is the minimum number of pages the process is guaranteed to have in memory
  - A process may be assigned as many pages up to its working set maximum
  - When the amount of free memory in the system falls below a threshold, automatic working set trimming is performed to restore the amount of free memory
    - Working set trimming removes pages from processes that have pages in excess of their working set minimum

End of Chapter 10