Memory Management

Read from the textbook

- Ch10.2 Demand Paging
- Ch10.5 Allocation of Frames
- Ch10.4 Page Replacement
- Ch10.6 Thrashing
- Operating-System Examples
- Windows Virtual-Memory Manager

Effective Access Time

Page Fault Rate $0 \le p \le 1$

if $p = 0$, no page faults

if $p = 1$, every reference is a fault

Effective Access Time (EAT)

EAT = (1 – p) x memory access
+ p x (page fault overhead
+ [swap page out]
+ swap page in
+ restart overhead)

- Suppose memory access (including TLB and extra page table accesses) = 20 nsecs
- Overhead at both ends say 10000 instructions ≈ 10000 nsecs
- Swap page in = 4 msecs = 4,000,000 nsecs
- 50% of time have to swap page out = 2,000,000 nsecs

EAT = (1 – p) x 20 + p x (10000 + 6,000,000)
≈ 20 + 6 000 000p nsecs

How often do we want page faults?

With

EAT ≈ 20 + 6,000,000p nsecs

If we want the EAT to be only half as slow again as real memory we need:

p = 10 / 6,000,000 ≈ 0.0000016

i.e., one page fault every 600,000 memory accesses

Even though the estimates were very approximate we see that we don't want page faults to happen very often.

With page sizes of 4K – 8KB we need lots of frames or lots of repeated access to make the speed acceptable.

Reducing Page Faults

- Different processes have different memory access patterns and therefore different numbers of pages they need to have in memory at one time.
- There is a minimum number we must have e.g. with the `add @A, @B` instruction in textbook p395 for each process (otherwise this instruction may never complete).
- We can allocate frames equally or proportionally (depending on size or priority).
- We can set minimum and maximum numbers per process.
- We really need the currently required pages in real memory.

Basic Page Replacement

1. Find the location of the desired page on the disk, either in swap space or in the file system.
2. Find a free frame:
   - If there is a free frame, use it.
   - If there is no free frame, use a **page-replacement algorithm** to select an existing frame to be replaced, known as the victim frame.
   - Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.
3. Read in the desired page and store it in the frame. Adjust all related page and frame tables to indicate the change.
4. Restart the process that was waiting for this page.

Need for Pages Replacement

- When there are no free frames to bring in a page the system has to pick one to replace.
- There are two main ways of selecting frames for replacement.
  - **Global** – any frame allocated to any process can be chosen
  - **Local** – chosen frames must come from the processes own allocated frames
- There are different consequences for these:
  - **Global** method: the number of frames for a process varies depending on its behaviour and the behaviour of the other processes. (The same process can run with widely varying speed due to other processes taking some of its frames.)
  - **Local** method: there are less frames to choose from.
- Normally global replacement is chosen, but we might have a maximum number of frames for the process.

We Still Have to Pick

- So of all currently occupied frames which one is chosen.
- We have some preferences:
  - pages that are **read-only** or haven't been modified don't have to be written back to disk (this saves on swapping time)
    - page table entries commonly have a dirty-bit to indicate the frame has been **changed** since the page was loaded
    - pages that aren't going to be accessed again in the **near future** (so we don't end up with another page fault on the page we just moved out) – unfortunately we can't see into the future so we rely on recent behaviour
- if we have a referenced bit we might use this to get an approximation

Replacement Algorithms

- Objective: minimize the **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.
- In all our examples, the reference string is

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- and we have 3 frames .

Random

- it treats every process fairly
- easy to implement
- with enough pages the method won't replace pages just about to be used too frequently

page request	1	2	3	4	1	2	5	1	2	3	4	5
frame 0	1	=	=	=	=	2	5	=	2	=	=	5
frame 1		2	=	4	=	=	=	=	=	3	=	=
frame 2			3	=	=	=	=	1	=	=	4	=

Replacement Algorithms

FIFO - first in first out

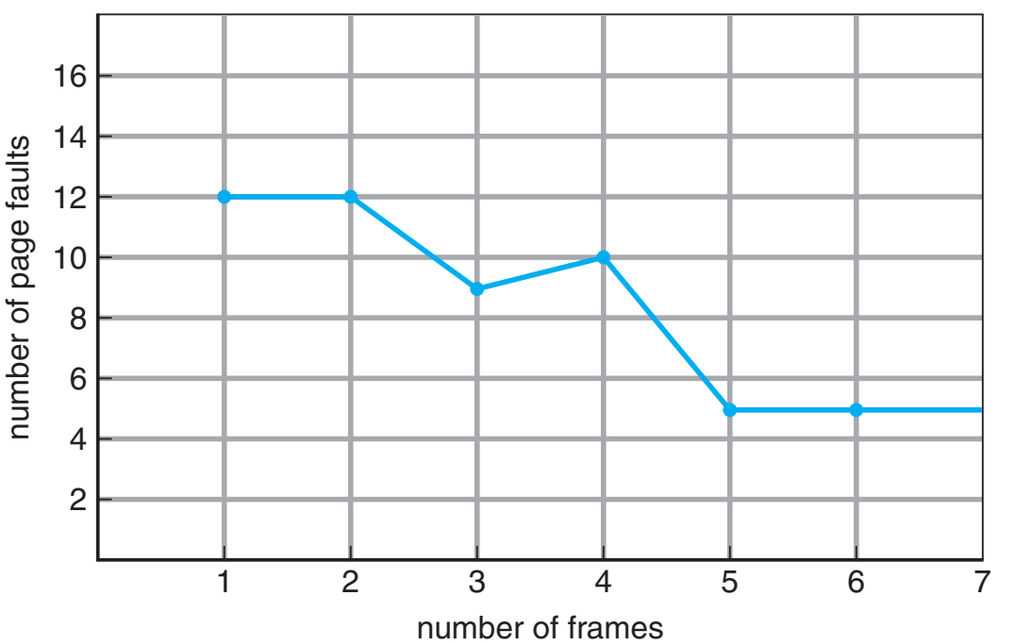
- Keep a list of pages in a queue.
  Remove the one at the head put new ones at the tail.
- Simple

Disadvantages:

- Very important pages (such as part of the operating system) which are referenced frequently will be paged out just as frequently as pages which are hardly ever referred to
- Belady's anomaly – increasing the number of frames occasionally increases the number of page faults

page request	1	2	3	4	1	2	5	1	2	3	4	5
frame 0	1	=	=	4	=	=	5	=	=	=	=	=
frame 1		2	=	=	1	=	=	=	=	3	=	=
frame 2			3	=	=	2	=	=	=	=	4	=

page request	1	2	3	4	1	2	5	1	2	3	4	5
frame 0	1	=	=	=	=	=	5	=	=	=	4	=
frame 1		2	=	=	=	=	=	1	=	=	=	5
frame 2			3	=	=	=	=	=	2	=	=	=
frame 3				4	=	=	=	=	=	3	=	=



Replacement Algorithms

Optimal Algorithm

- If we can see the future, the optimal algorithm replaces a page which is going to be used furthest away in the future.
- If pages are not going to be used again then FIFO on those is suitable.

page request	1	2	3	4	1	2	5	1	2	3	4	5
frame 0	1	=	=	=	=	=	=	=	=	3	=	=
frame 1		2	=	=	=	=	=	=	=	=	4	=
frame 2			3	4	=	=	5	=	=	=	=	=

Replacement Algorithms

Least Recently Used – LRU

- Based on the assumption that a page not used recently will not be used in the near future.
- In this example not as good as FIFO – generally better.
- Why can't LRU suffer from Belady's anomaly?

page request	1	2	3	4	1	2	5	1	2	3	4	5
frame 0	1	=	=	4	=	=	5	=	=	3	=	=
frame 1		2	=	=	1	=	=	=	=	=	4	=
frame 2			3	=	=	2	=	=	=	=	=	5

Disadvantages:

- Very expensive – need to have hardware that keeps track of last access time for each page.
- Or maintain a list of pages and move a page to the top of the list when accessed.

Approximations to LRU

Use the referenced bit – originally clear, set when the page is used.

Keep regular track (additional reference bits)

Every 100 msecs (say) move the referenced bit into the high bit of a value (say 8 bits), shifting all bits to the right and clear the referenced bit for every page.

e.g.

R: 1 referenced byte: 0 0 0 1 1 1 1 1

becomes

R: 0 referenced byte: 1 0 0 0 1 1 1 1

The pages with the lowest numbers have either been used the longest time ago (or not used as regularly).

Can select randomly from lowest valued or use a FIFO strategy to choose.

Second chance (clock algorithm)

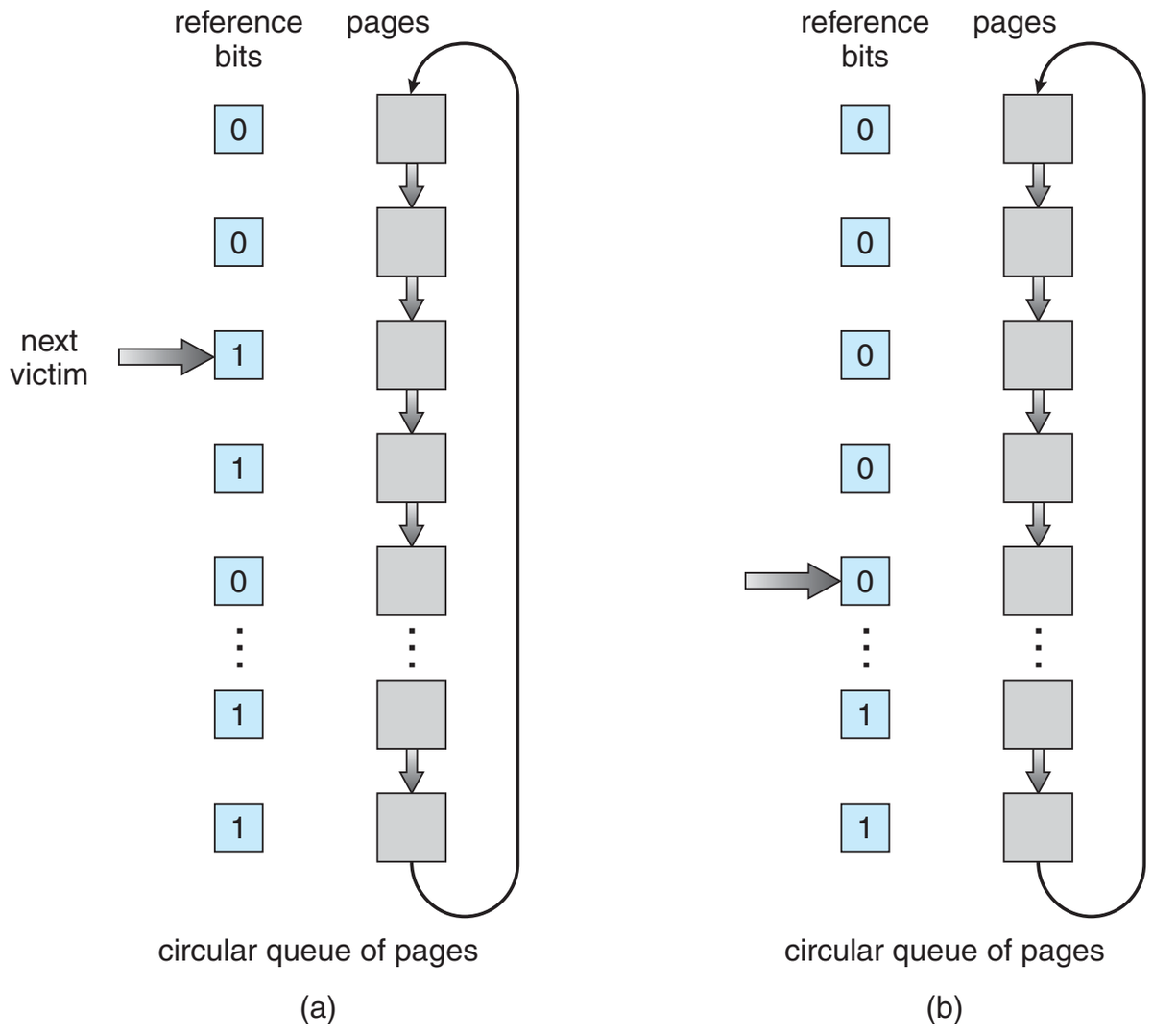
FIFO + hardware-provided reference bit: if a page has a 1 in its referenced bit when it is chosen we don't replace it, but clear its referenced bit instead (and change its arrival time to be now).

Commonly implemented as a circular queue

Approximations to LRU

Clock replacement

- If page to be replaced has
  - 4Reference bit = 0 -> replace it
  - 4reference bit = 1 then:
    - set reference bit 0, leave page in memory
    - replace next page, subject to same rules



Replacement Algorithms

Least Frequently Used – LFU

- maintain a count of memory accesses for each page
- keep heavily used pages
- Pages can stay around after they are needed – can decrease the count over time.
- New pages get picked on.

ref. cnt: 1 2 2

page request	1	2	3	4	1	2	5	1	2	3	4	5
frame 0	1	=	=	4	=	=	5	=	=	3	4	5
frame 1	=	2	=	=	1	=	=	=	=	=	=	=
frame 2	=	=	3	=	=	2	=	=	=	=	=	=

Disadvantages: expensive adding to the count with every access

Replacement Algorithms

Most Frequently Used – MFU

- Pages with very few accesses may have just been brought in to memory.

Neither is commonly used.

Death Row

- Put frames into a replacement pool according to FIFO selection.
- Keep track of which page is in each frame.
- If a page is accessed while its frame is in the replacement pool then retrieve it.
- There is no penalty for paging from disk in this situation.

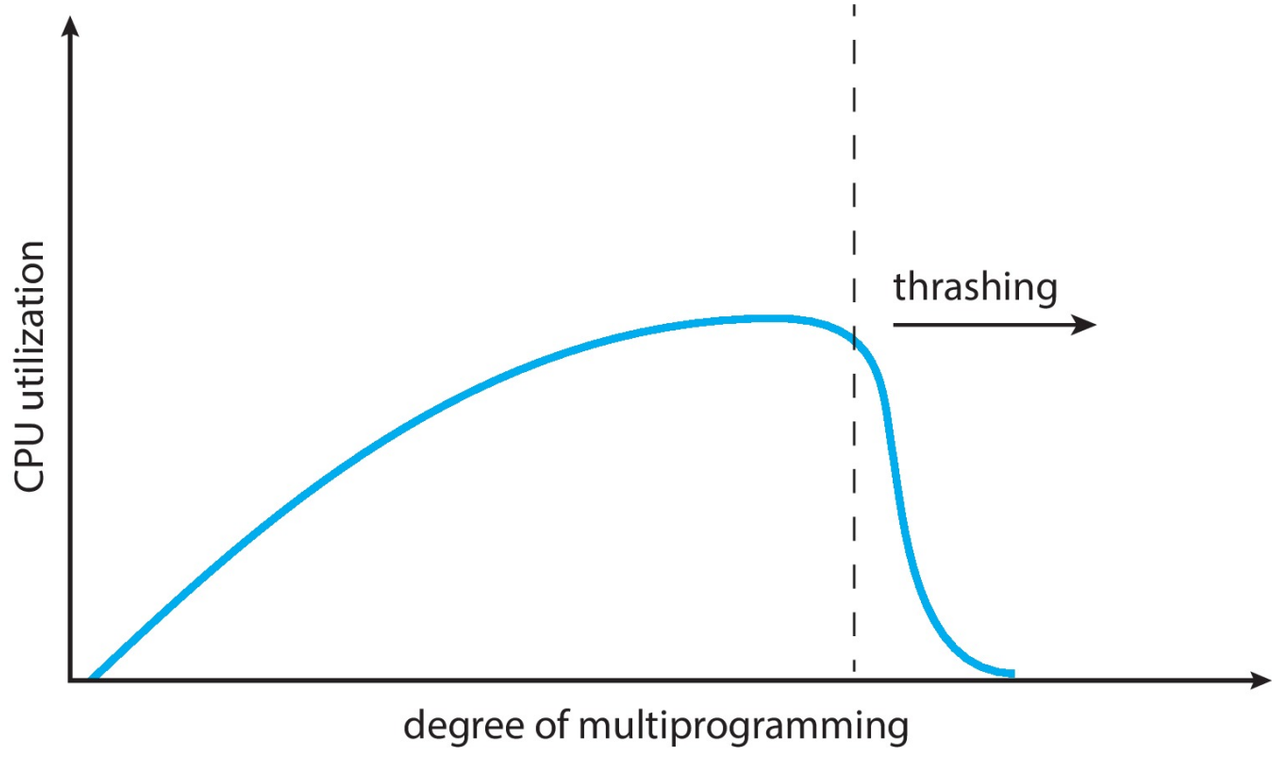
Thrashing

Thrashing: A process is busy swapping pages in and out

- If the sum of the number of pages of the working sets of all processes in the system exceeds the number of frames we are in deep trouble —> thrashing
- Pages are continuously tossed out while they are still in use
  - Every page fault causes a page from the working set of a process to be removed.
  - By definition the removed page is going to be accessed soon causing another page fault.
  - And so on.
- It severely affects the amount of work that can be done.
- Any process that falls below the number of pages in its working set should be suspended and swapped out (it is not going to get any work done anyway).

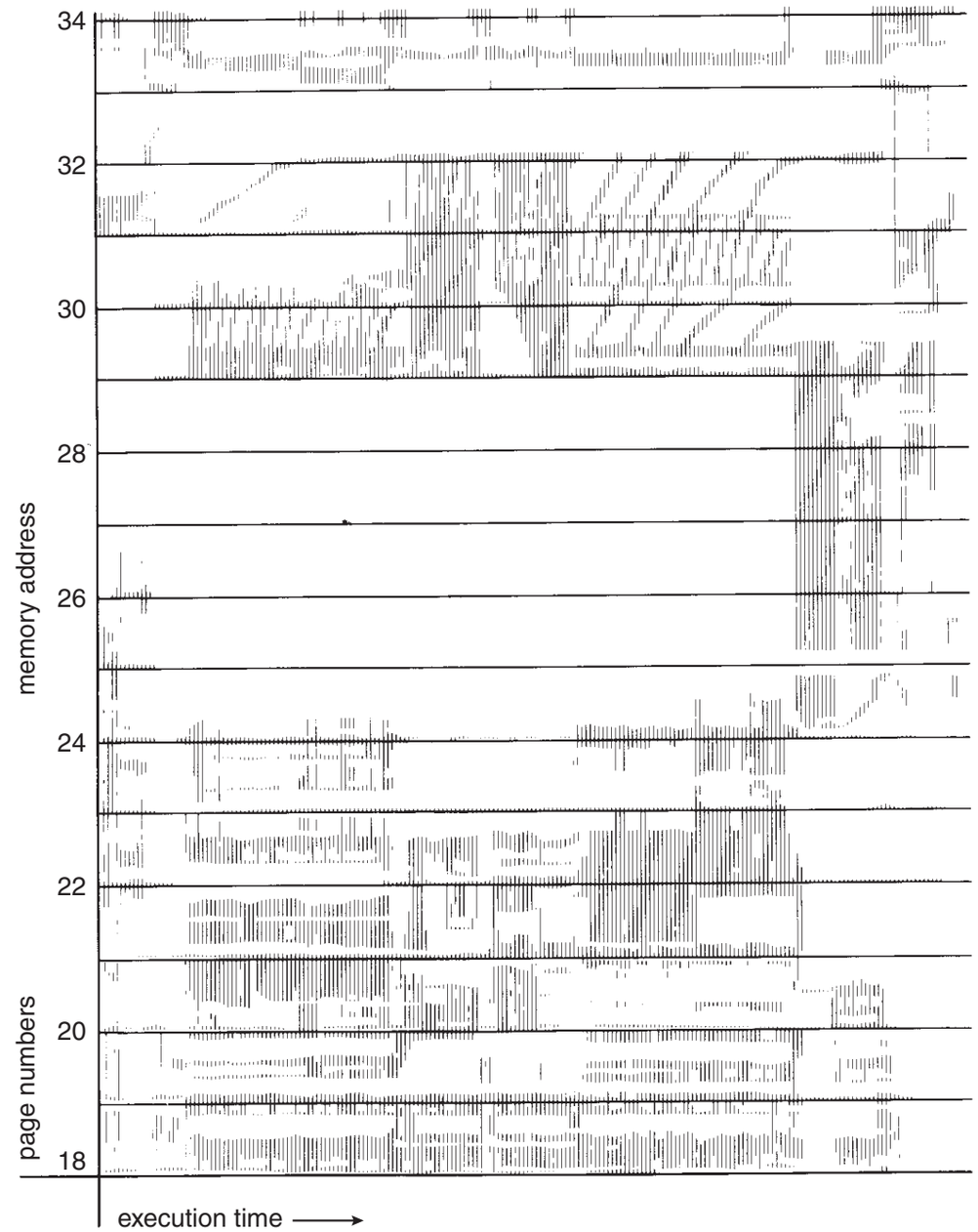
Batch System Thrashing

- If a batch system is set up to increase the number of programs running in the system at a time if the CPU utilisation gets too small we can get thrashing very easily.



Demand Paging and Thrashing

- Why does demand paging work?
  - Ans: **Locality model**
  - Process migrates from one locality to another
  - Localities may overlap
- Why does thrashing occur?
  - Ans: Σ **size of locality > total memory** size
  - Limit effects by using local or priority page replacement
- E.g., Locality In A Memory-Reference Pattern



Working Set Model

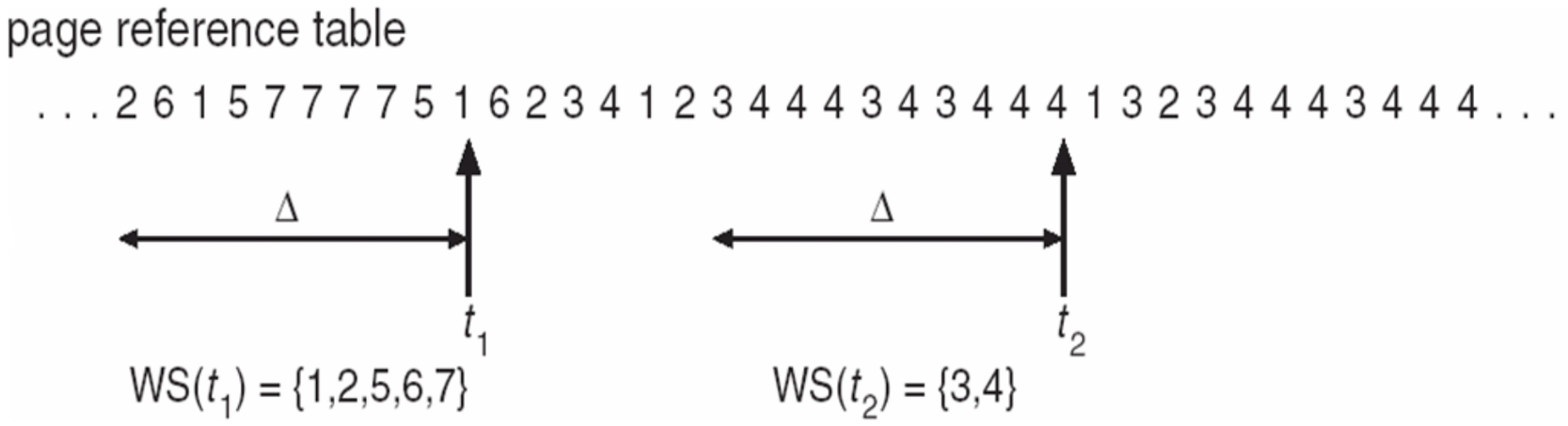
Working set: the set of pages that a process referenced in past T seconds

We talked earlier of the notion of *locality of reference*.

- The working set of a process is the **collection of pages** needed in real memory in order to keep the process running.
- If we observe a process running over a short period of time (a window) we can record the page accesses the process makes. This is a picture of the page's working set.
- The trick is getting the window the right size:
  - if it is too small —> not enough pages are included in the working set
  - if it is too big —> too many pages are included
- Approximate with interval timer + a reference bit
- A **reference bit** is available in the page table entry in some architectures to indicate the **page has been accessed** (read or written) since it was cleared.
- Example: window = 1,000 msec
- Timer interrupts after every 500 msec.
- Keep in memory 2 bits for each page.
- Whenever the timer interrupts read and then set the values of all reference bits to 0.
- If one of the bits in memory = 1 ⇒ page in working set.

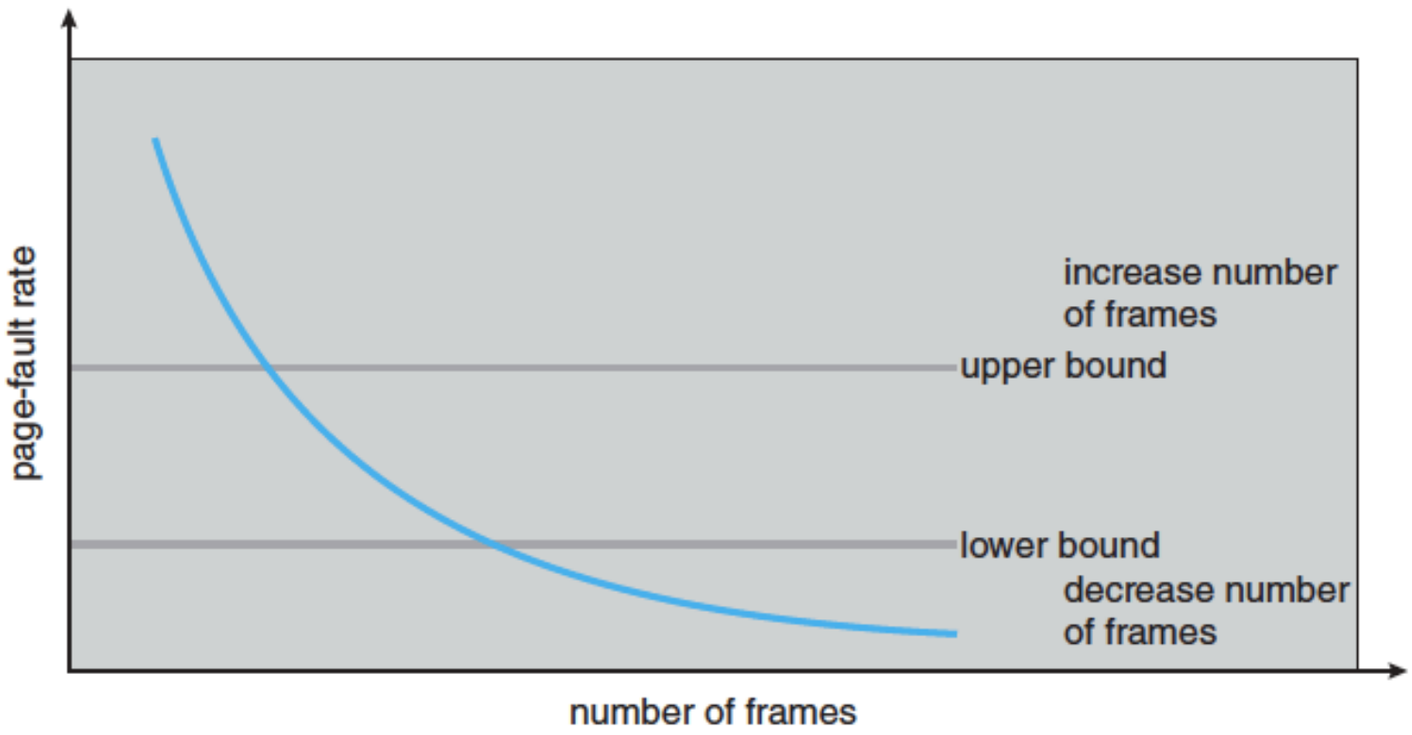
Working Set Model

- Δ := working-set window := a fixed number of page references
    - Example: 10,000 instructions
- WSSi (working set of Process Pi) = total number of pages referenced in the most recent Δ (varies in time)
    - if Δ too small will not encompass entire locality
    - if Δ too large will encompass several localities
    - if $\Delta = \infty$ => will encompass entire program
- $\Delta = \Sigma$ WSSi := total demand frames
- Approximation of locality



Page Fault Frequency

- We can also use the **Page Fault Frequency** (**PFF**) to control the number of frames allocated to a process.
- As the number of frames increases the number of page faults drops rapidly at first, then reaches a point where adding more frames hardly alters the rate at which paging occurs. We set upper and lower bounds and add or remove frames to stay within them.



Before Next Time

Read from the textbook

- Ch17.1 Goals of Protection
- Ch17.2 Principles of Protection
- Ch17.4 Domain of Protection
- Ch17.5 Access Matrix