Page Replacement Algorithms

- When a page fault occurs, some page that's currently in memory needs to be discarded to make room for a new page
- Picking the right page to discard isn't easy
- Many algorithms have been studied

Idealized Algorithm

- Ideally, we want to discard the page that will be needed last
- Which is that?
- Run the program again, find out which it is; that will tell you which to replace this time
- Oops. . .

Constraints

- Must be efficient — many paging decisions take place
- Must approximate the right answer
- Must be implementable
- Must be implementable on real hardware
- Usually, must work well on multitasking systems

Tools

- The OS has a few tools available to it
- The *referenced* bit — this page has been used recently
- The *modified* bit — discarding this page will be more expensive
- Clock interrupts
- Page fault interrupts
- Advice from the application

Page Fault Interrupts

- (Of course) enter the kernel
- Synchronous to the running process
- Instruction is retryable or restartable (what if the instruction's operand crosses a page boundary?)
- Fault information indicates the offending address as well as the restart point

5 / 36

Sometimes R and M are Missing

- Some hardware platforms omit the M and R bits
- They can be simulated in software
- To imitate M, periodically mark the page read-only; if there's a protection fault, the page should be considered "dirty"
- To imitate R, mark the page invalid; if there's a page fault, it should be considered "referenced"

6 / 36

Not Recently Used (NRU)

- At process start time, reset all R and M bits
- On clock interrupts, clear R bits
- Classify pages by M and R:

	R	M
Class 0:	0	0
Class 1:	0	1
Class 2:	1	0
Class 3:	1	1

- On page fault, discard a random page from the lowest class

7 / 36

Resetting R and M

- Why do we reset R on clock interrupts?
- We want to know if a page has been used *recently*
- Why not reset M?
- M can't be reset until the page has been written out to disk; an old copy won't suffice

8 / 36

Properties of NRU

- Bias towards discarding unmodified pages
- But — better to discard a modified page that hasn't been used recently than one that is in use
- Simple algorithm; may give adequate performance on some systems
- Primarily useful for teaching

9 / 36

Reclaiming Modified Pages

- Must first schedule a disk write operation
- When it's complete, the page frame can be reused
- Do we keep state binding a particular inbound page to that page frame?
- Do we make the other process wait for *two* disk operations before we let it run?
- Reclaiming a page isn't cheap!

10 / 36

What's Interesting about NRU?

- It has the essential properties of any page replacement algorithm
- It looks for a (relatively) idle page
- It handles modified pages, but is biased against using them
- It's reasonably efficient

11 / 36

FIFO

12 / 36

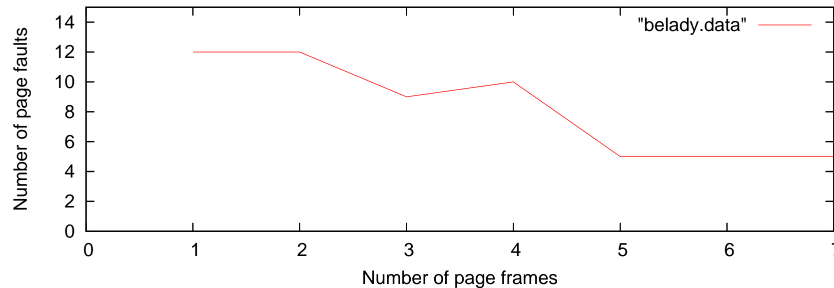
First In, First Out (FIFO)

- Don't bother with R and M
- When a page frame is needed, discard the oldest page
- Of course, the oldest page may still be busy, so it will come right back in
- FIFO is rarely used in this form

12 / 36

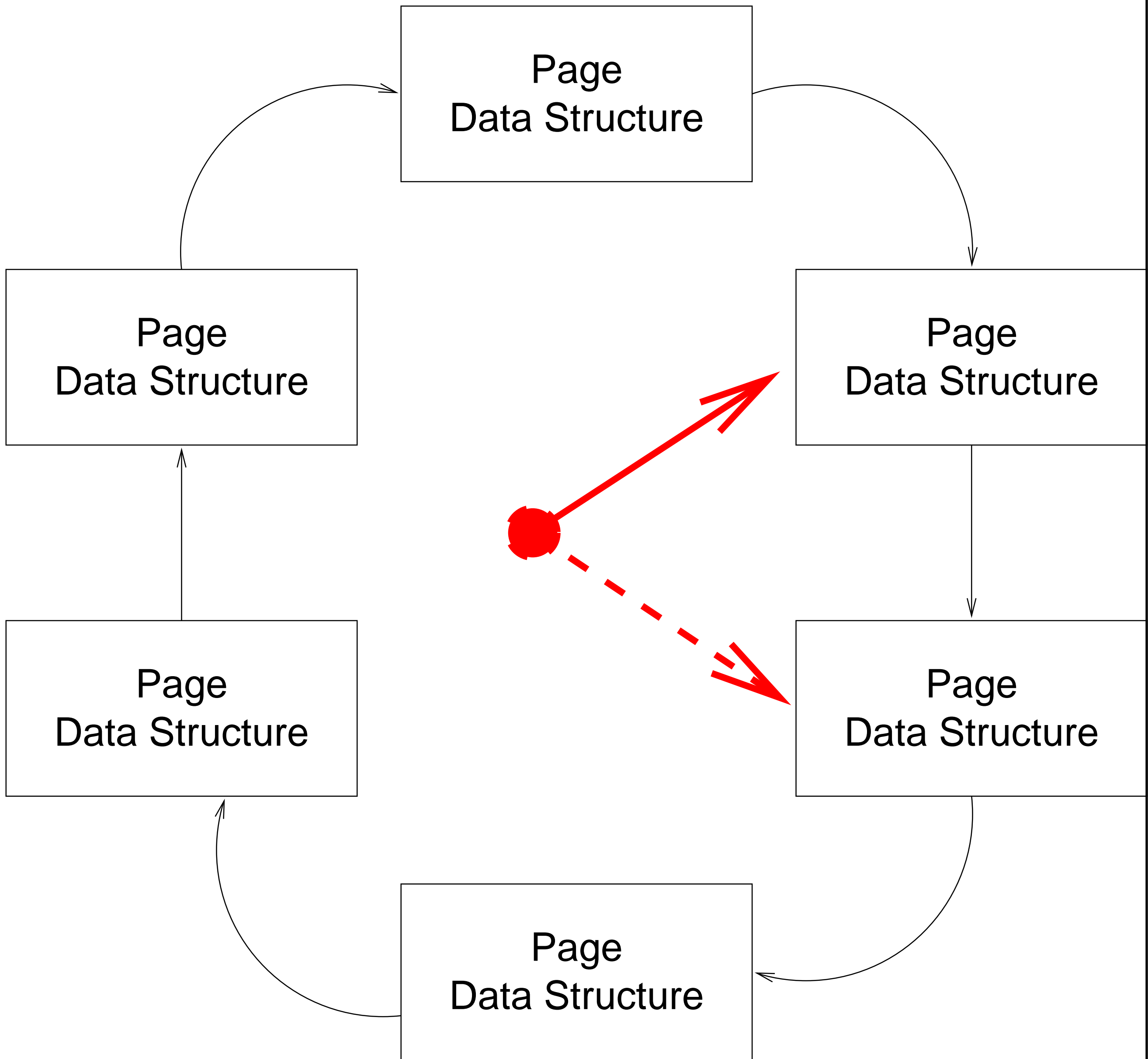
The Belady Anomaly

- Sometimes, having more page frames hurts instead of helps
- Very counter-intuitive
- Example in the text using FIFO page replacement



Second Chance FIFO

- Similar to pure FIFO, but the R bit is checked
- If R is set on an old page, clear the R bit but move it to the just-loaded end of the queue
- Note the problem: it doesn't know if a page has been used *recently* or not
- It only approximates that if page faults are frequent
- If all pages have been referenced, degenerates to FIFO
- FIFO can be implemented with a circular linked list; the list head is changed, rather than moving the page table entry. This is called the *clock* algorithm


The Clock Algorithm
Page
Data Structure
Page
Data Structure
Page
Data Structure
Page
Data Structure
Page
Data Structure
Page
Data Structure
15 / 36

Least Recently Used (LRU)

- Assumption: a page that hasn't been used recently is unlikely to be used soon
- But — how can this be implemented?
- More precisely, what data structure do we use to track this?
- LRU is generally a linked list; do we point to it with a hash table indexed by page address?

16 / 36

Hardware-Assisted LRU

- Have a 64-bit instruction counter
- On each memory reference, store the counter in a per-page frame field
- On a page fault, scan the page table for the lowest value
- Note: this is O(n) in the number of page frames

17 / 36

A Cheaper Hardware Assist

- For n pages, create a $n \times n$ bit matrix initialized to 0
- When referencing page k, set row k to 1 and set column k to 0
- The lowest binary value is the least-recently used page
- Again, O(n) in the number of page frames

18 / 36

Software-Simulated LRU

- Have an array of counters, one per page
- At each clock tick, add the value of R to the counter
- Implements NFU — *Not Frequently Used*
- Problem: never forgets

NFU With Aging

- Shift each counter right (i.e., divide by 2) before each addition
- Add R to the high-order bit
- Recent references have more weight
- Pages referenced this clock tick and previous clock ticks are more important than those referenced only this time
- Note: *still* O(n), every clock tick

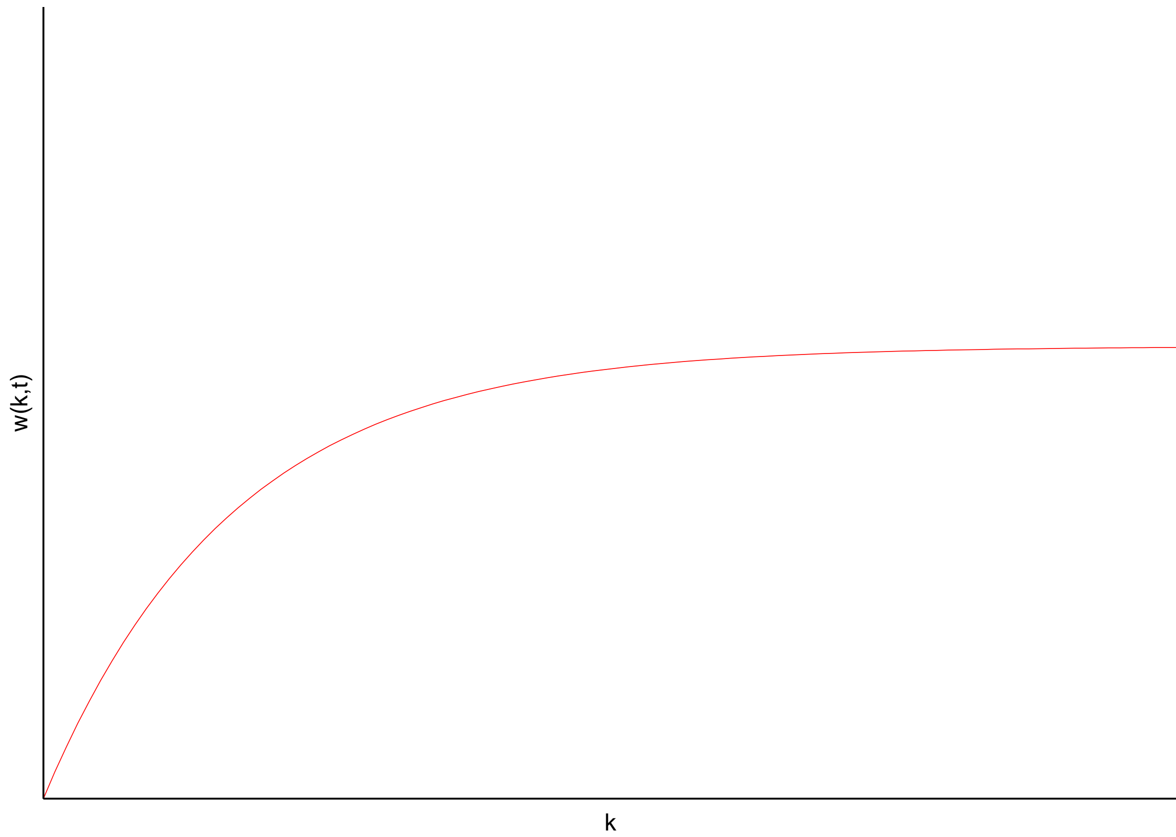
Working Set

The Working Set

- At any time, a program is only using a small fraction of its pages (locality of reference)
- The set of pages in use at the moment is the *working set*
- At time t, the working set $w(k,t)$ is the k most recently-reference pages
- Note that $w(k,t)$ is monotonically increasing as a function of k and it asymptotically approaches the total program size
- A program needs to have its working set in memory
- If there isn't enough memory to hold the entire working set, the program will *thrash*

21 / 36

Working Set



22 / 36

Using the Working Set

- Before running a process, make sure that the working set is in memory
- Because k is asymptotic, the exact choice of k isn't critical, as long as it's large enough
- The trick is to determine the working set

Approximating the Working Set

- Ideally, we would track the last k memory references
- Instead, we track pages referenced during the last τ seconds
- For each page, keep a clock field
- At each tick, update the clock field if R is set
- Pages not referenced during the last τ seconds may be discarded
- Algorithm is still O(n). But we can do better.

WSClock

- Use the working set concept; use the clock algorithm's data structure
- Have a circular linked list of page frames
- At each page fault, check the page pointed to by the clock hand; if R is 1, the page is current and can't be discarded; advance the clock hand
- If $R = 0$, check the age. If $R > \tau$, the page can be reused

Dealing with Modified Pages

- If $M = 0$, the page frame exists on disk and can be reused immediately. (We initialize M to 1 if it's never been written.)
- If $M = 1$, schedule it to be written to disk *and keep scanning* — you might find a clean page
- Limit the number of writes for any pass to some value n

What Happens if the Clock Hand Circles?

- If some writes have been scheduled, keep looking; a write will complete eventually
- If no writes have been scheduled, all pages are in the working set; pick a random clean page to reuse
- But — the current process no longer has its working set in memory. Do we schedule it?
- In general, we have to be *very* cautious here; we don't want the total system to thrash

27 / 36

System Issues

28 / 36

The Scheduler and Paging

- As noted, the WSClock algorithm can force the scheduler to leave the CPU idle
- This will generally improve total throughput
- Also note that the pager generally needs a process context — it may want to wait for disk I/O to complete, for example

28 / 36

Anticipatory Paging

- The system can only reclaim a page immediately if it already exists on disk; otherwise, it has to wait for a disk operation to finish
- When the disk subsystem is relatively idle, find and write out dirty, unused pages
- The M bit can then be reset, making them immediately available for reuse if necessary

Prepaging

- When a process blocks for an extended period, remember its working set
- Its pages will tend to be reclaimed, since they're not being used
- When the process wakes up, restore its working set to memory before resuming execution
- Avoid the overhead of too many page faults

The Paging Disk

- You want the paging disk to be as fast as possible
- Sometimes, dedicate a separate disk to paging; avoid contention from file I/O
- Old systems often used *drums* or *head-per-track* disks, to minimize or avoid rotational delay or seek time

Sharing Page Tables

- If two processes are running the same executable and code pages are read-only, the two can share the same page table
- Do they share the same working set data structure? What if a page is in one process' working set but not the other's?
- Extra data structures are needed to track such shared pages; reference counters are mandatory

Page-Faulting in Executables

- Suppose that files with executables are formatted like page areas?
- That is, each in-memory page of the program begins on a disk block boundary
- Metadata and data areas are in different blocks
- No need to read in a program before executing it; just set up the paging data structures to point to the actual file
- This is one reason an executing file can't be opened for output

Implementing Copy on Write

- When discussing processes, we talked about "copy on write". We can now see how to implement it
- The code pages are shared
- The two processes share data *pages* but *not* the page table for the data area
- The shared pages are marked read-only in both processes
- When a protection fault occurs, the page is copied to an empty page frame (where does it come from?), and both page table entries are updated

Locking Pages

- Real-time systems can't afford page faults, at least on parts of the program
- Solution: let the application *lock* some pages into memory
- Also used for some I/O
- If too many pages are locked, it can affect overall system performance
- Locking is thus restricted, either to root or to a certain quota (run `ulimit -a` on CLIC)

Summary

- NRU is never actually used
- Plain FIFO works poorly; second chance FIFO works reasonably well
- NFU with aging is a good choice
- WSClock is efficient and gives good results