Demand Paging & Page Replacement

CSE 410, Spring 2009

Computer Systems

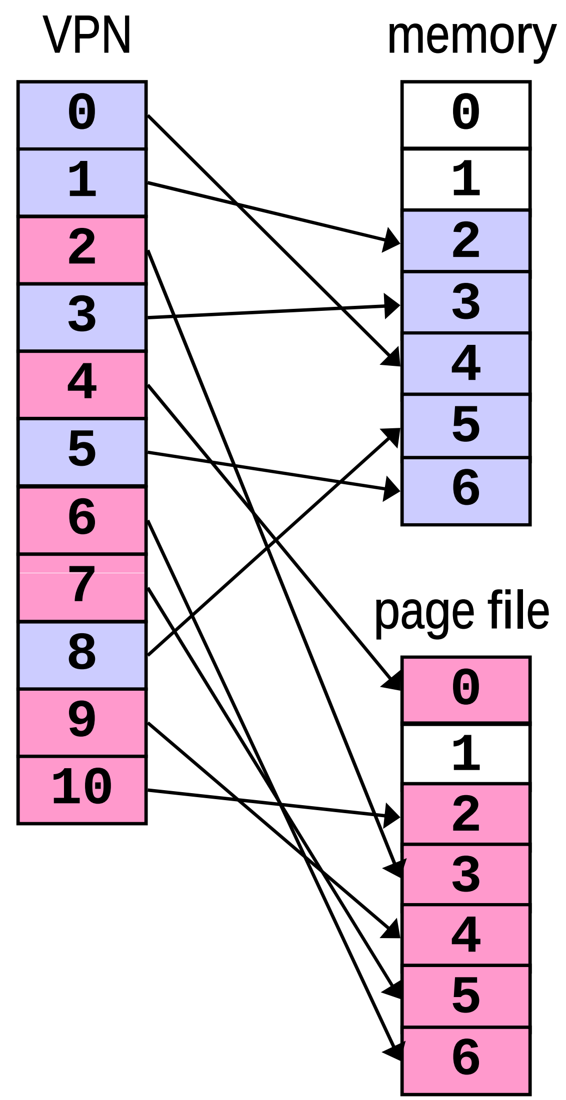
http://www.cs.washington.edu/410

Readings and References

- Reading
  - » Chapter 9 through 9.4.5, *Operating System Concepts*, Silberschatz, Galvin, and Gagne

Virtual Memory

- Page table entry can point to a PPN or a location on disk (offset into **page file**)
- A page on disk is swapped back in when it is referenced but is not actually present in main memory
    - » **page fault**



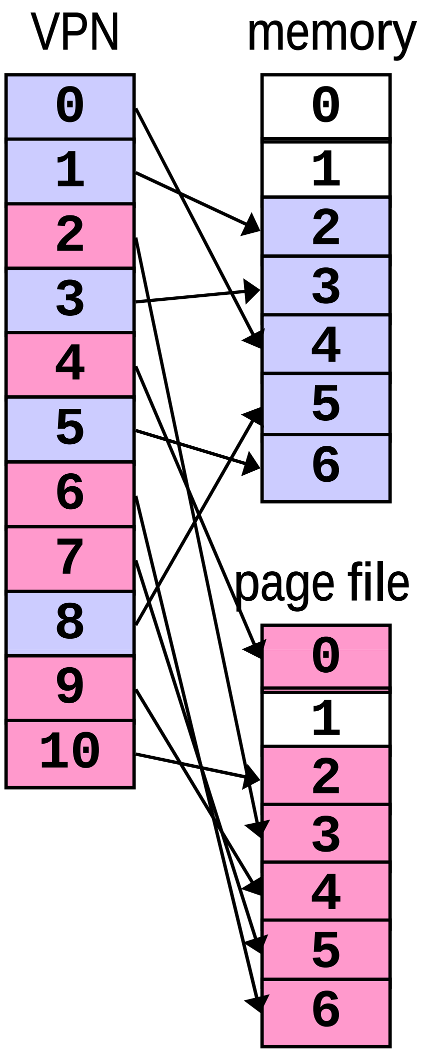
Demand Paging

- As a program runs, the memory pages that it needs may or may not be in memory when it needs them
  - » if in memory, execution proceeds
  - » if not in memory, page is read in from disk and stored in memory
- If desired address is not in memory, the result is a page fault

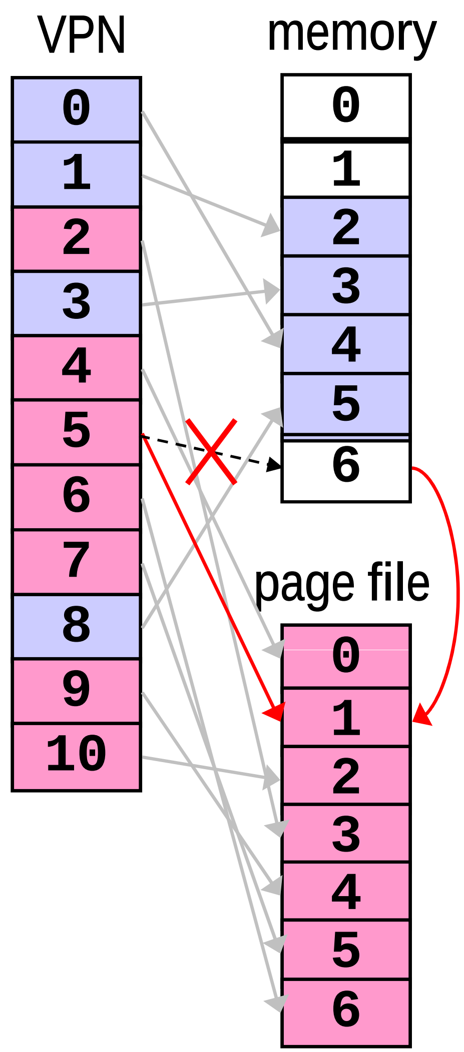
A reference to memory location X

- MMU: Is X's VPN in the Translation Lookaside Buffer?
  - » Yes => get data from cache or memory. **Done.**
  - » No => Trap to OS to load X's VPN/PPN into the TLB
- OS/hardware: Is X's page actually in physical memory?
  - » Yes => replace a TLB entry with X's VPN/PPN. Return control to original thread and restart instruction. (MIPS: software, x86: hardware) **Done.**
  - » No => must load the page from disk
- OS: replace a current page in memory with X's page from disk
  - » pick a page to replace, write it back to disk if dirty
  - » load X's page from disk into physical memory
  - » Replace the TLB entry with X's VPN/PPN.
  - » Return control to original thread and restart instruction. **Done!**

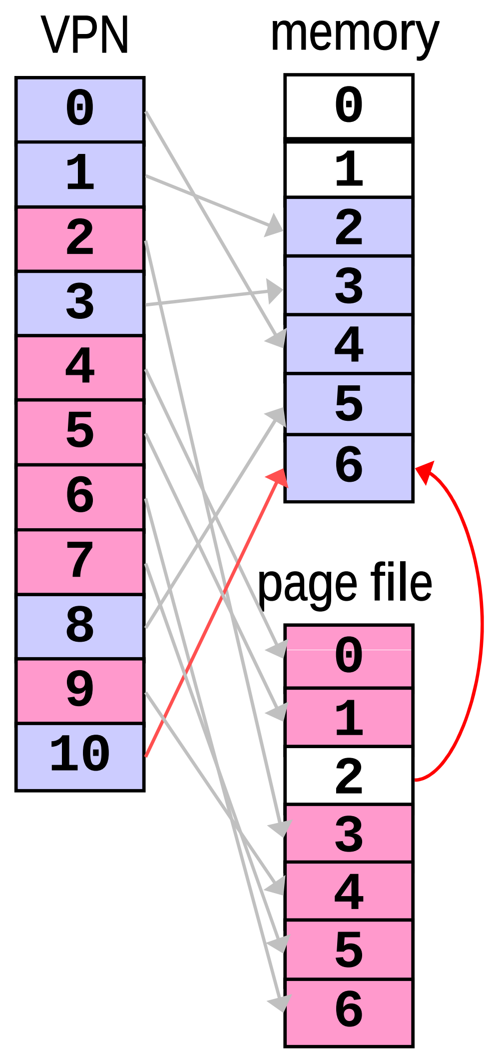
Page Fault Example



Reference to VPN 10 causes a **page fault** because it is not in memory.



PPN 6 has not been used recently. **Write** it to the page file.



Read VPN 10 from the page file into physical memory at PPN 6.

Evicting the best page

- Page replacement: need to evict some page to free a page frame
- Goal: minimize fault rate by selecting best page to evict
  - » Best is one that will never be touched again!
- Belady's algorithm (min): evict the page that won't be used for the longest period of time
  - » provably optimal, minimizes page fault rate
  - » Can't implement (requires clairvoyance)
- So need to find some feasible approximation

Replacement Algorithms

- FIFO - First In, First Out
  - » throw out the oldest page
  - » rationale: it’s been around a long time, less likely to be currently used
  - » then again, it might be quite active; we have no information either way
  - » Belady’s Anomaly: fault rate might increase when FIFO is given more physical memory!
    - a very bad property

LRU & LRU Clock

- LRU - Least Recently Used
  - » exploits temporal locality
    - if we have used a page recently, we probably will use it again in the near future
  - » LRU is hard to implement exactly since there is significant record keeping overhead
- CLOCK - approximation of LRU
  - » and LRU is an approximation of MIN

Perfect LRU

- Least Recently Used
  - » timestamp <u>each</u> page on <u>every</u> reference
  - » on page fault, find oldest page
  - » can keep a queue ordered by time of reference
    - but that requires updating the queue on <u>every</u> reference
  - » too much overhead per memory reference

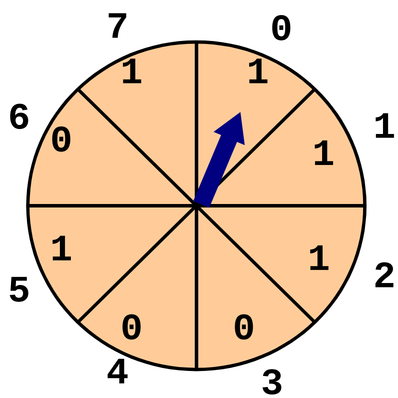
LRU Approximation: Clock

- Clock algorithm
  - » replace an old page, not necessarily the oldest page
- Keep a reference bit for every physical page
  - » memory hardware sets the bit on every reference
  - » bit isn't set => page not used since bit last cleared
- Maintain a “next victim” pointer
  - » can think of it as a clock hand, iterating over the collection of physical pages

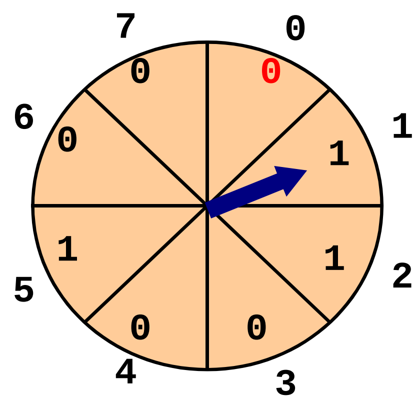
Tick, tick, ...

- On page fault (we need to replace somebody)
  - » advance the victim pointer to the next page
  - » check state of the reference bit
  - » If set, clear the bit and go to next page
    - this page has been used since the last time we looked. Clear the usage indicator and move on.
  - » If not set, select this page as the victim
    - this page has not been used since we last looked
    - replace it with a new page from disk

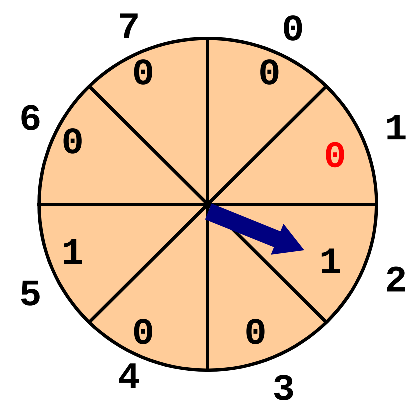
Find a victim



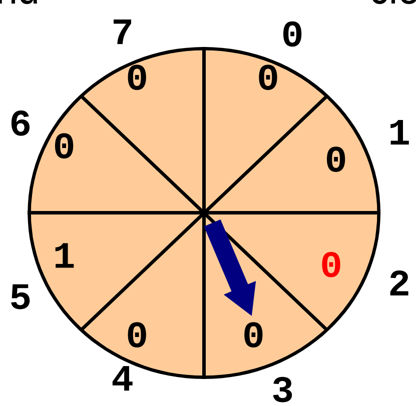
advance; PPN 0 has been **used**; clear and advance



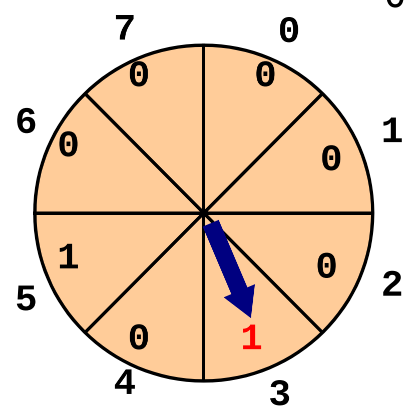
PPN 1 has been **used**; clear and advance



PPN 2 has been **used**; clear and advance



PPN 3 has been **not** been used; replace



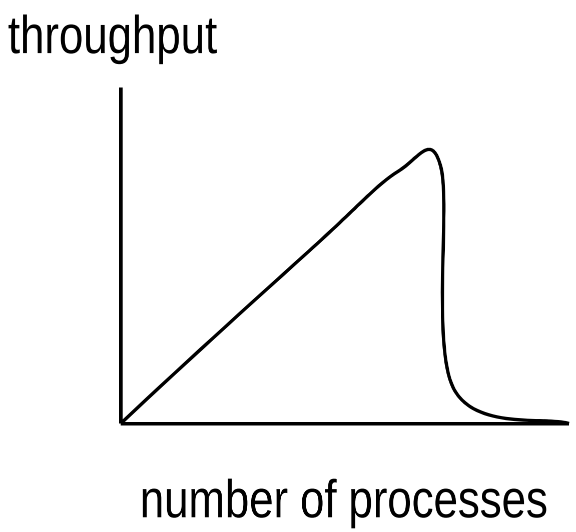
PPN 3 use bit set on next memory reference

Clock Questions

- Will Clock always find a page to replace?
  - » at worst it will clear all the reference bits, finally coming around to the oldest page
- If the hand is moving slowly?
  - » not many page faults
- If the hand is moving quickly?
  - » many page faults
  - » lots of reference bits set

Thrashing

- **Thrashing** occurs when pages are tossed out, but are needed again right away
  - » listen to the hard drive grind
- Example: a program touches 50 pages often but there are only 40 physical pages in system
- What happens to performance?
  - » enough memory 200 **ps**/ref (most refs hit in cache)
  - » not enough memory 10 **ms**/ref (page faults every few instructions)



Thrashing Solutions

- If one job causes thrashing
  - » rewrite program to have better locality of reference
- If multiple jobs cause thrashing
  - » only run as many processes as can fit in memory
  - » swap out hogs if they can't run without thrashing and run when fewer processes active
- Buy more memory

Working Set

- The working set of a process is the set of pages that it is actually using
  » set of pages a job has used in the last T seconds
  » usually much smaller than the amount it might use
- If working set fits in memory process won't thrash
- Why do we adjust the working set size?
  » too big => inefficient because programs keep pages in memory that they are not using very often
  » too small => thrashing results because programs are losing pages that they are about to use

Page Fault Frequency (PFF) Algorithm

- We've glossed over issue of how to divide available page frames among contending processes
- One solution: PFF – allocate page frames based on process working sets
  - » Goal: minimize paging, avoid thrashing
  - » Issue: how do we do this fairly among contending processes?
    - Answer: ?