Virtual Memory in Today's Operating Systems

Part 1 of 2

Greg Hartman

Why Did I Get Interested in Virtual Memory?

- From my MapQuest experience, sub-second response times matter to users
- Users should control computers, but virtual memory is the major rival
  - Frustrating pauses at just the wrong time
  - Becomes more prominent as processor speed increases
- If we had a better handle on virtual memory, we could make most software have predictable, fast response times

Why Use VM?

- 1977 PC's didn't manage memory – Apple II
  - This forces them to be single tasking
  - "Operating system" provides:
    - Device driver for storage
    - Library of reusable routines
  - OS designers documented unused addresses, programmers hard-coded them into programs
  - Routines called by jumping to fixed addresses
  - Therefore, OS can't change
  - Tended to be hobby systems, not tools

1981 – PCs Become Tools

- OS begins to manage memory
  - MSDOS 1.0
- Memory broken into segments
  - OS decides how to pack segments in memory
  - Segments can move
    - Programmer must lock segments before using
  - Maximum segment size generally 32k-64k
- Designers thought this was intuitive, but
  - Segments become a major source of bugs

1984 – Windowing Systems

- Rapid increase in program complexity
- Forces a decade of OS advances:
  - Cut-and-paste was hard
    - Attempt to solve through multitasking, 1987
  - Programs were too big
    - Attempt to solve through virtual memory (paging), 1991
  - Programs crashed too often:
    - Attempt to solve through protected, flat memory, 1995
- Most technology lifted from server OSs
- Suddenly, Windows becomes a server OS

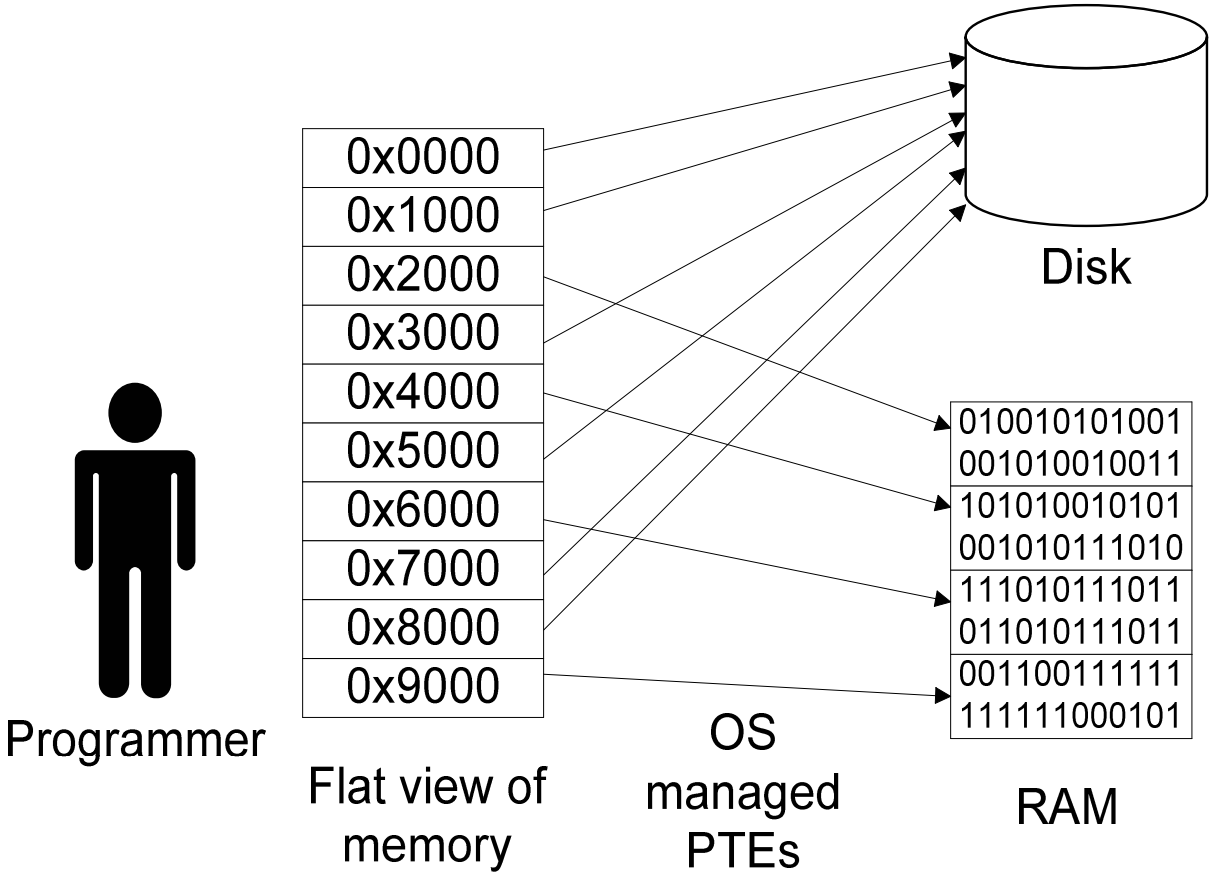
Why Was Paging Adopted?

- Invisible to programmers
- Intrinsic memory protection
- Code reuse through shared libraries
  - Use memory more efficiently
  - Bugs can be fixes without recompiling apps
- Reduces program load time
- Allows for zero-copy OS operations
  - Largely unexploited in POSIX
    - Involves API changes

Why Paging Doesn't Work

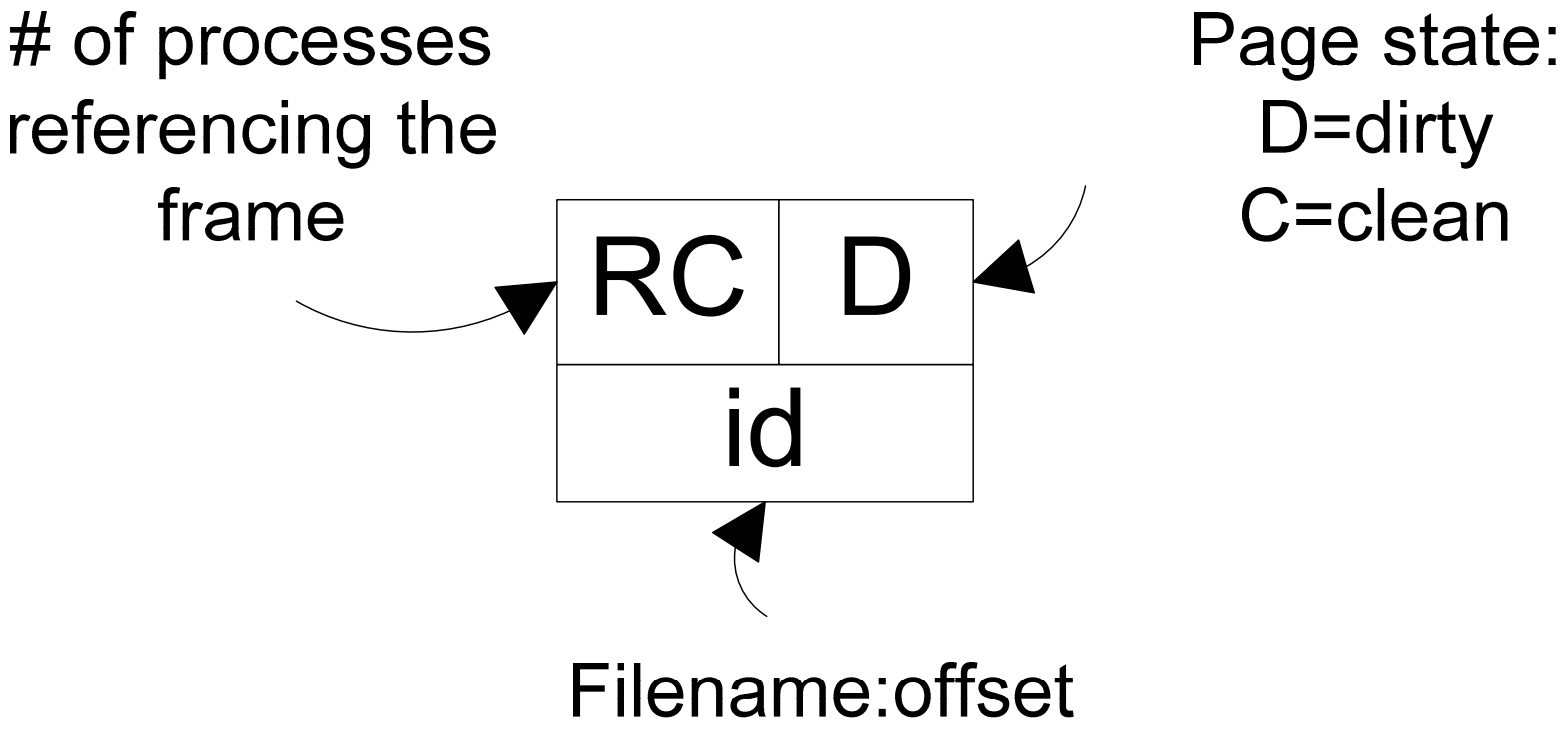
- User's expectations
  - User controls number of tasks
  - Expects quick feedback from all of them
  - May want to launch especially difficult tasks in background
- Operating system's reality
  - OS has no way of predicting what users will do next
  - OS runs self-maintenance tasks, e.g., virus scanners
    - Ideally, should be invisible; in reality, not the case

Introduction to Entities



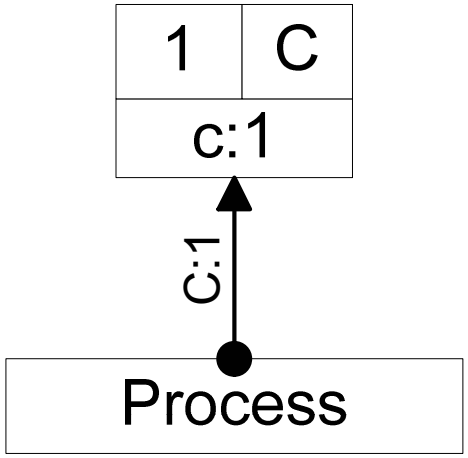
Pages

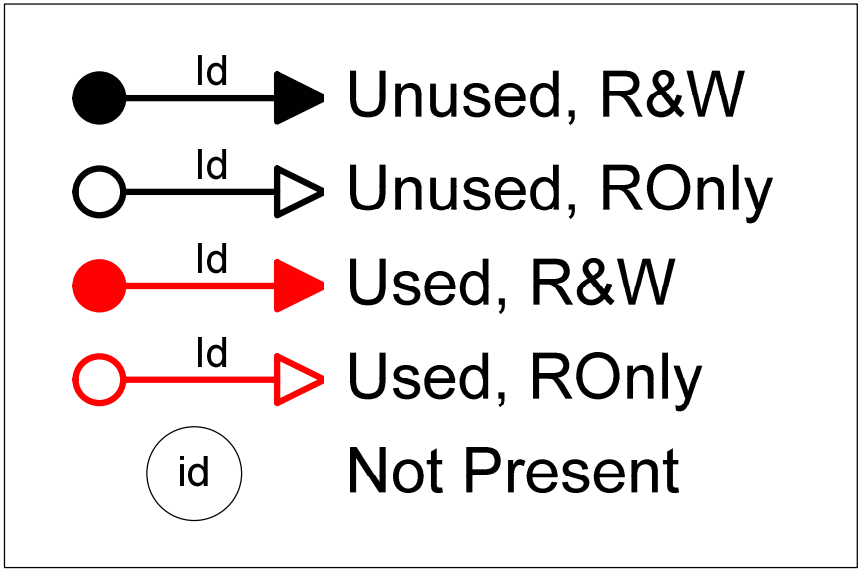
- Each page holds 4k of data, and has the following attributes:



PTE Entries

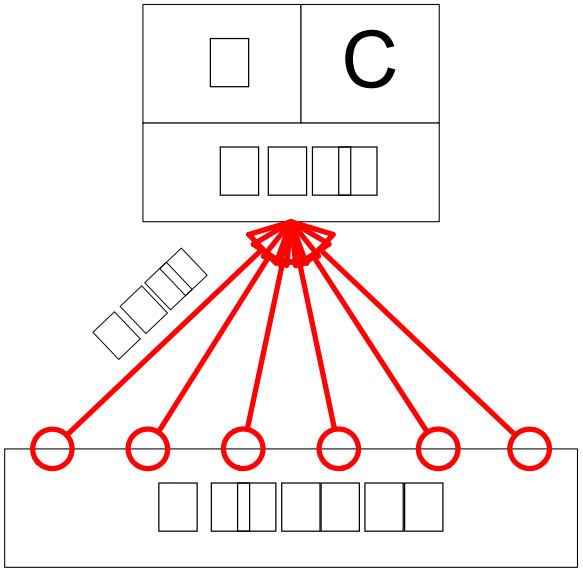
- Connect process address space to pages
  - May be a one-way mapping
  - Read-only or read-write
  - Processor marks PTEs as they are used

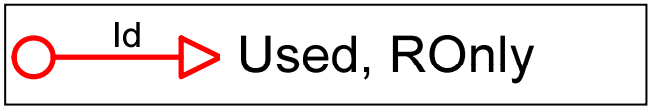




Memory Allocation

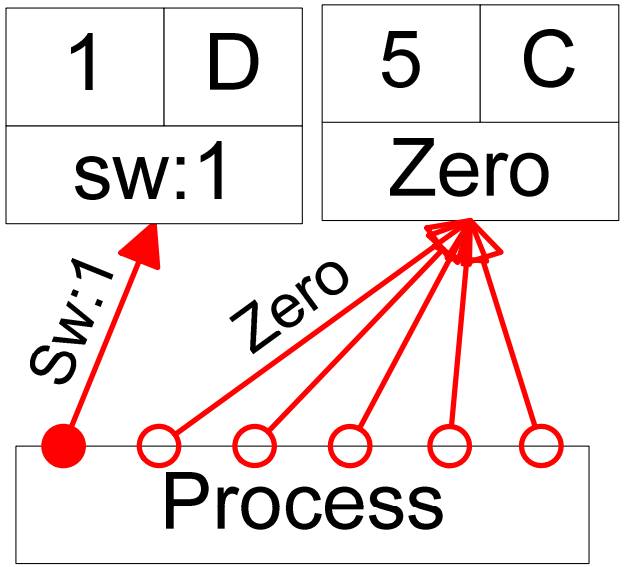
– OS points process to one zero'd page
– Process doesn't have write permission
– Saves time clearing memory

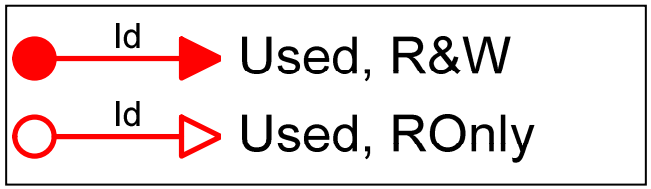




Copy on Write

- On first attempt to write, process faults
- OS zeros a free page and updates the PTEs
- Space for page is allocated in swap (sw:1)

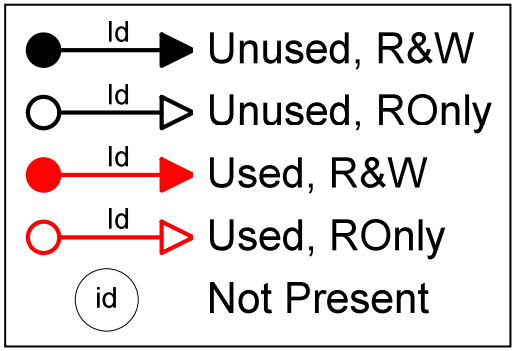




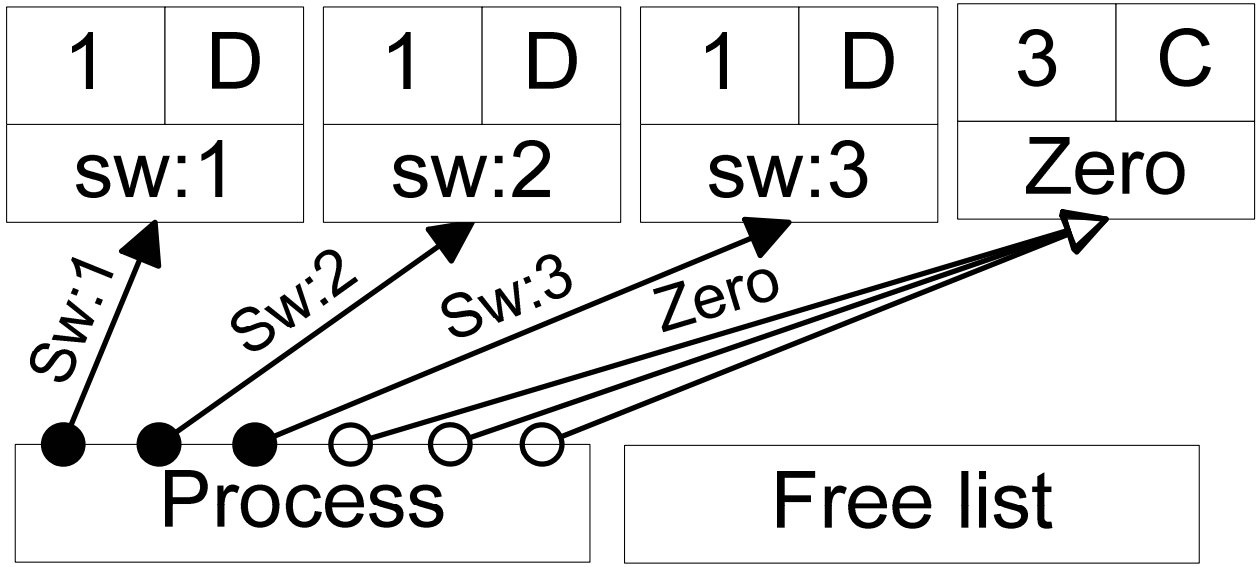
Why Reclaim?

- OS needs a pool of free pages
  - For example, copy-on-write relied on this
- OS needs to refill when it runs low
- OS needs to make this transparent
- This involves several steps…

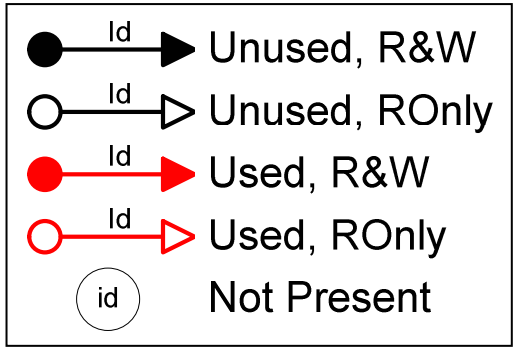
Reclaiming Pages



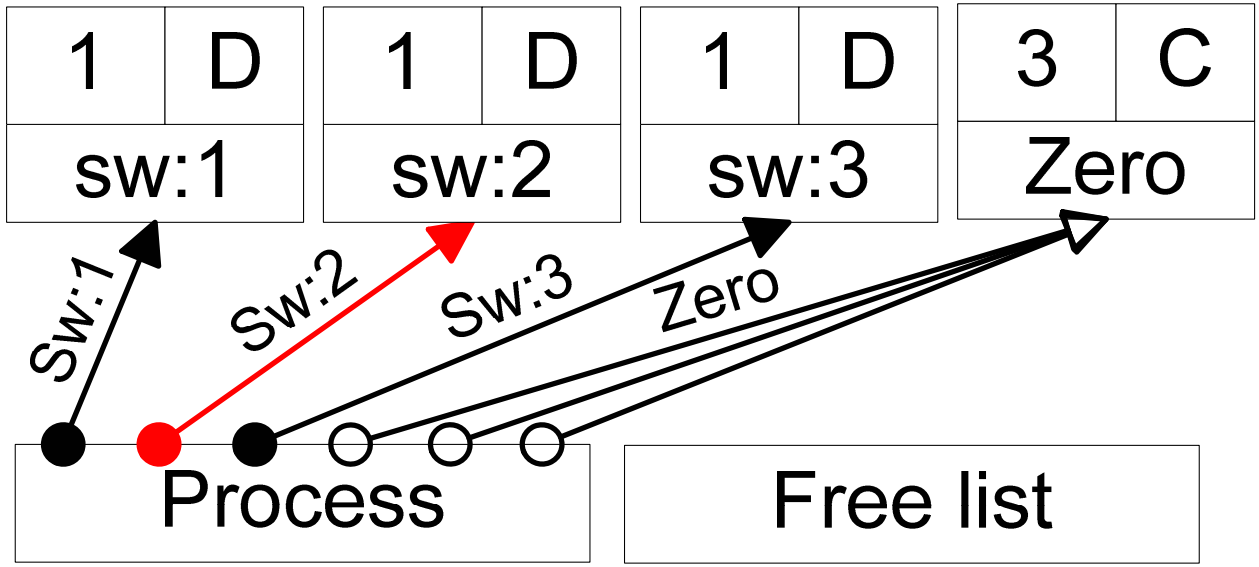
OS periodically sets pages to unused



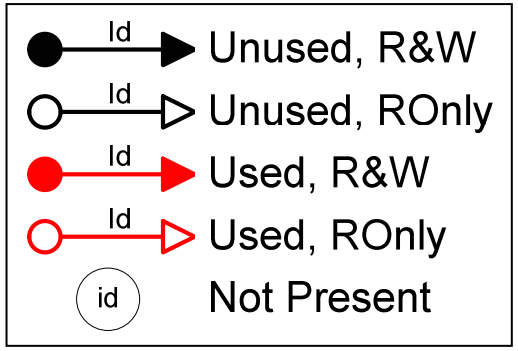
Reclaiming – Part 2



Processor marks pages as they are used



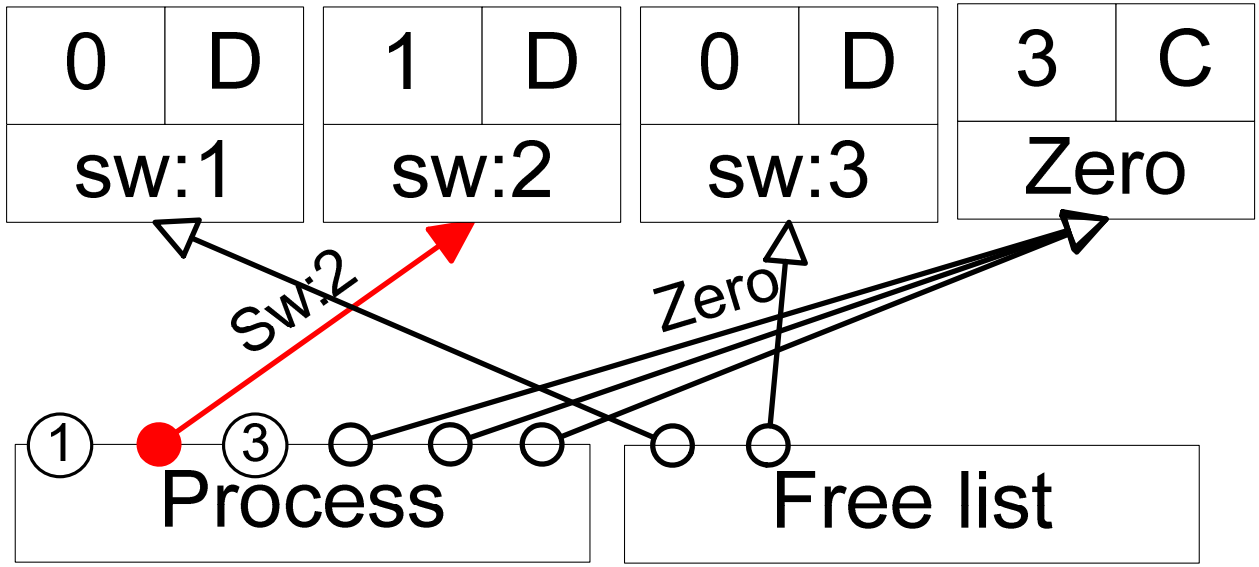
Reclaiming – Part 3



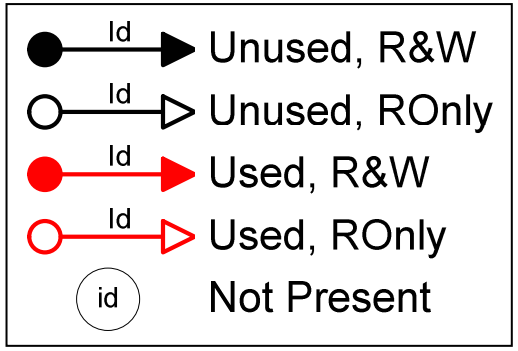
OS breaks unused PTE entries

Writes dirty pages to disk

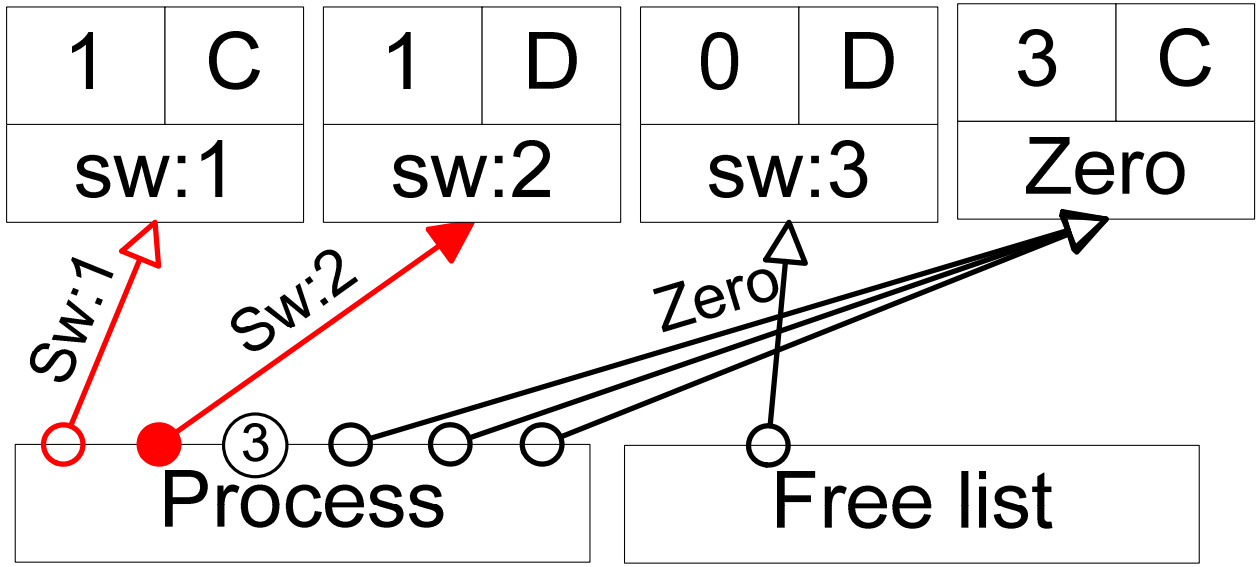
Makes “free” list of unreferenced pages



Page In



- If a process references a page:
  - Minor fault: page on the free list
  - Major fault: page loaded from disk



What are the Technical Issues?

- VM subsystem must mask huge performance differences
  - Process can access ~100,000 pages/sec
  - Disk can only move ~10,000 pages/sec

    (Measured Red Hat 8, Pentium 4 1.7GHz 512MB RAMBUS memory)
- Need to write old page, so get ~ 5,000 pages/sec
- Page replacement algorithms ignore or limit multi-tasking
- Performance hard to predict
  - Program behavior not well understood
  - VM implementations take shortcuts

Open Questions

- How does VM interact with scheduling?
- What scenarios cause VM to break down?
  - Quantify performance impact, duration, recovery time
- Can these scenarios be predicted?
  - Prediction requires understanding the program
  - Need to develop an efficient memory profiling system.
- How do we present an interface to the programmer/user?
  - There are no “one-size-fits-all” algorithms
- Approaching limits of 32-bit address space
  - Limits memory to 4GB per process
- Memory becoming a significant power drain on handheld devices

Related Research

- Memory protection in real-time systems
- Distributed VM
- Predictive pre-fetching
- Track extents
- Performance isolation
- Adaptive algorithms
- Compiler instrumentation

Preview of Part 2

- Description of my VM instrumentation
  - Data logging system
  - Automated tester
- Show examples of VM failures/instability
- Explain these results
  - Will use concepts covered in this talk
- Discuss future research directions