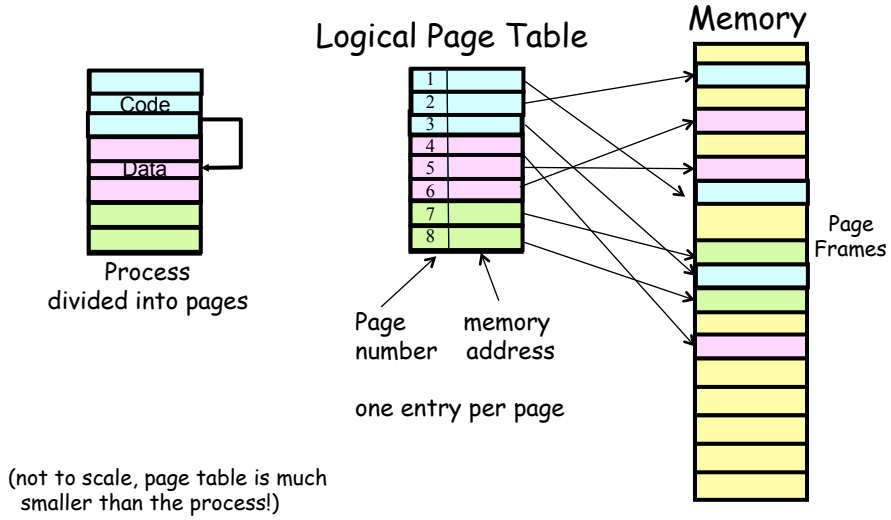
Lecture 18: Virtual Memory

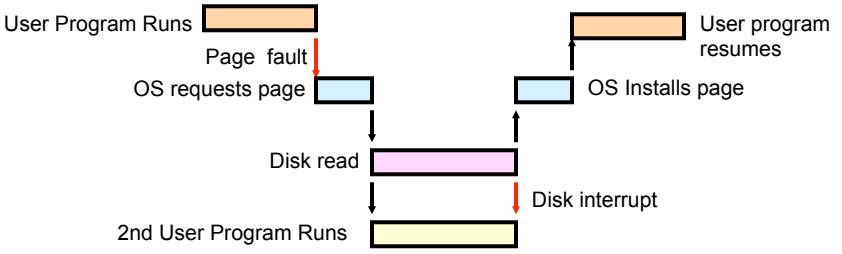
- Administration
  - Take QUIZ 14 over P&H 5.6, 11-13 before 11:59pm today
  - Homework 7 due Thursday April 1, 2010 no joke ☺
  - Exam April 6 (review April 1)
- Last Time
  - Virtual memory abstractions
  - Basics of segments and paging
- Today
  - Integration of virtual memory into cache hierarchy
  - DRAM memory organization, TLBs

Paging: Chop the program up into pages, load each page on demand in any page frame

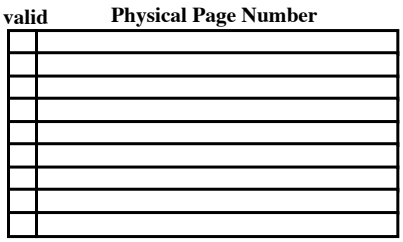


(not to scale, page table is much smaller than the process!)

Page Fault



What else can we put in this table?

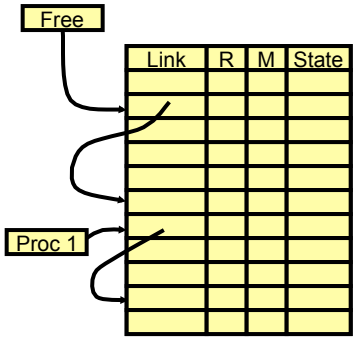


- What if we want a page to be read-only?
- What if we want a page to only hold instructions?
- What if we want to keep track of "dirty" pages?
- What if we want to track frequently used pages?

VM Requires Hardware

- Restartable (or resumable) instructions
  - must be able to resume program after recovering from a page fault
- Ability to mark a page *not present*
  - and raise a page fault when referencing such a page
- (Optional) Maintain status bits per page
  - R - referenced - for use by replacement algorithm
  - M - modified - to determine when page is *dirty*
  - *OS can store these elsewhere if need be*

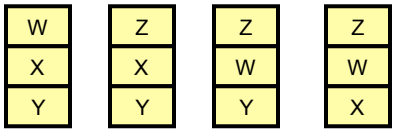
Page Frame Management



Page Frame Table

- OS maintains
  - **page table** for each user process
  - **page frame table**
  - free page list
    - pages evicted when number of free pages falls below a *low water mark.*
  - pages evicted using a *replacement policy*
    - random, FIFO, LRU, *clock*
  - if M-bit is clear, need not copy the page back to disk

Page Management and Thrashing

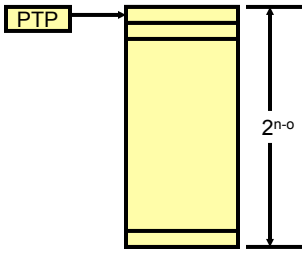


Example: Reference four pages in sequence, mapped to three page frames

- Need to keep a process' *working set* in memory or *thrashing* will occur
- Find working set size by increasing page frame allocation until Page faults/ms falls below limit
- Swap out whole process if insufficient page frames for working set

Page table organization

Page Table Organization



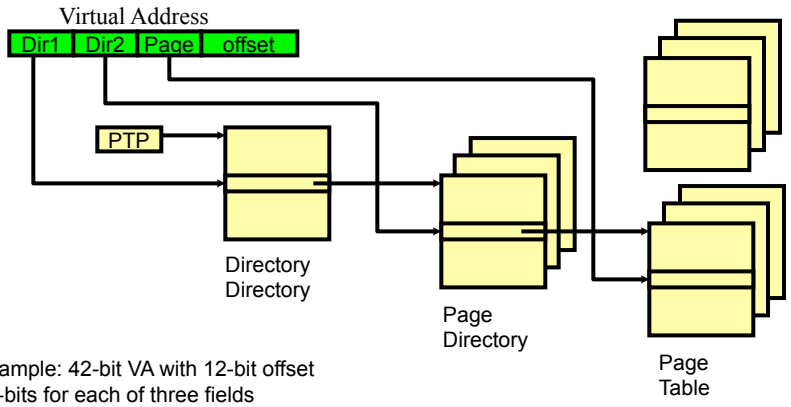
Flat page table has size proportional to size of *virtual* address space

- can be very large for a machine with 64-bit addresses and several processes

Three solutions

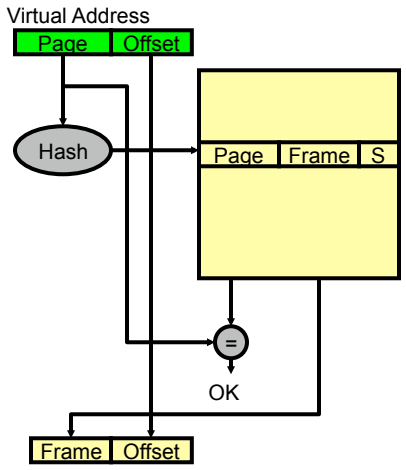
- page the page table (fixed mapping)
  - what really needs to be *locked down?*
- multi-level page table (lower levels paged - Tree)
- inverted page table (hash table)

Multi-Level Page Table



Example: 42-bit VA with 12-bit offset
10-bits for each of three fields
1024 4-byte entries in each table (one page)

Inverted Page Tables

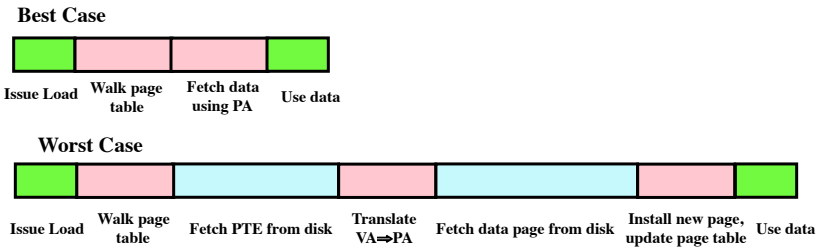


- Only store table entries for pages in physical memory
- Miss in page table implies page is on disk
- Usual hash-table rules apply:
  - Hash table should not be full
  - Rule of thumb: at least twice as many hash table entries as there are pages in memory.

Summary so far

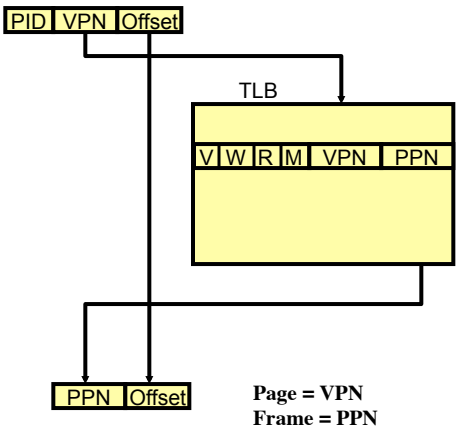
- Virtual memory provides
  - Illusion of private memory system for each process
  - Protection
  - Relocation in memory system
  - Demand paging
- But – page tables can be large
  - Motivates: paging page tables, multi-level tables, inverted page tables
- Next:
  - How can we improve performance of page tables?

How Long does it Take to Access VM?



- Problems
  - Multiple memory and potentially disk accesses
  - Can we use cache for the page-table access? How?

Use a cache: Translation Lookaside Buffers



- Store most frequently used translations in small, fast memory (cache for page table entries)
- Valid, Writeable, Referenced, Modified
  - Access protection
  - Replacement strategies
- Size – often 128 entries
- Highly associative (sometimes fully assoc.)

"Rare" Behavior in VM system

- TLB Miss
  - Translation is not in TLB – but everything could be in memory
  - Two approaches
    - Hardware state machine *walks* the page table
      - fast but inflexible
    - Exception raised and software walks the page table
- Page Fault
  - Entry not in TLB and target page not in main memory
- Worst case
  - Page fault and page table and target page

Reducing TLB misses

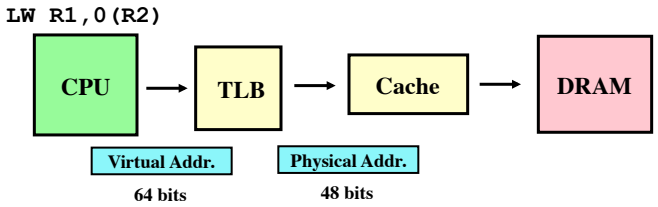
- Same type of optimizations as for cache
  - Associativity (many TLBs are fully associative)
  - Capacity – TLBs tend to be 32-128 entries
- Adjust page size
  - Small pages
    - Reduces internal fragmentation
    - Speeds page movement to/from disk
  - Large pages
    - Can cover more physical memory with same number of TLB entries
  - Solution – typically have a variable page size
    - Select by OS, 4KB-256KB (superpages)
    - AMD "Barcelona" supports 1 GB pages!

Combining virtual memory with conventional caching

Virtual Memory + Caching

- Conflicting demands:
  - Convenience of flexible memory management (translation)
  - Performance of memory hierarchy (caching)
- Requires cooperation of O/S
  - Data in cache implies that data is in main memory
- Combine VM and Caching
  - Where do we put the Cache and the TLB????

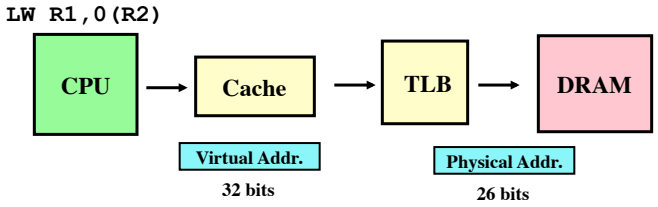
Physically Addressed Cache



- Translate first from VA ⇒ PA
- Access cache with PA

- Problems?

Virtually Addressed Cache



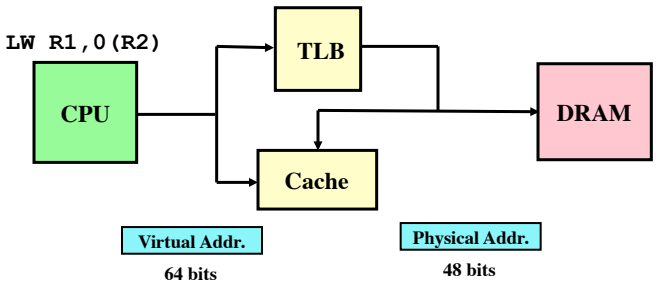
- Access cache first
- Only translate if going to main memory

- Problems?

Virtually addressed caches give aliasing problems

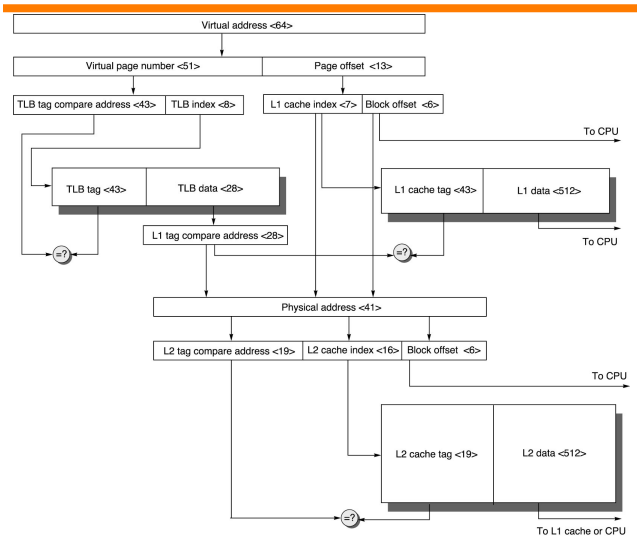
- Can occur when switching among multiple address spaces
- Synonym aliasing
  - Different VAs point to the same PA
  - Occurs when data shared among multiple address spaces
  - One solution – always translate before going to the cache
- Homonym aliasing
  - Same VA point to different PAs
  - Occurs on context switching
  - Two solutions:
    - Flush TLB on process switch/system call
    - TLB includes process ID

Best of Both Worlds:



Virtually addressed, Physically Tagged
- **Parallel Access**
- **Eliminate synonym problem**

Virtual Index, Physical Tag



Other Aliasing Solutions

- Note virtually indexed/physically tagged put constraints on cache capacity, page size, etc.
- Other solutions:
  - 21264: 8KB pages, 64KB i-cache, 2-way set associative
    - Aliases could reside in 8 different places in cache
    - On cache miss, invalidate any possible aliases in cache
  - Intel Pentium 4
    - Virtually indexed/virtually tagged cache
    - Check for TLB misses off line (roll back if necessary)

Virtual Memory Summary

- Relocation, Protection
- Apparent memory size >> DRAM capacity

- Translation
  - From large VA space to smaller PA space
  - Page tables hold translations

- Provides
  - Separation of memory management from user programs
  - Ability to use DRAM as cache for disk
  - Fast translation using Translation Lookaside Buffer (TLB)
    - Cache for page table
  - Speed - translate in parallel with cache lookup

Recap: caches and virtual memory

- Two basic ideas:
  - Indirection (data = memory[table[address]])
  - Caching

- Caches
  - Uses caching idea

- Virtual memory
  - Uses indirection to convert virtual->physical address
  - Uses caching to store active pages in DRAM, inactive pages on disk
  - Uses caching to store active page translations in TLB

Virtual memory "cache" policies

- Write back
- Fully associative
- Clever replacement policies, implemented in SW

Summary

- Virtual memory: the illusion of a memory as large as the disk!
- Next Time
  - Exam Review!
  - Bring your questions
- Exam April 6, open notes