5
Virtual Memory Architecture

18-548/15-548 Memory SystemArchitecture
Philip Koopman
September 9, 1998

Reading: **Cragon 3.0-3.3.1; 3.4 to top of page 166**

Supplemental Reading: **http://www.cne.gmu.edu/modules/vm/submap.html**
Jacob & Mudge: Virtual Memory: Issues of Implementation
Hennessy & Patterson: 5.7, 5.8

Assignments

- **By next class read:**
  - Cragon 2.2.7, 2.3-2.5.2, 3.5.8
- **Supplemental Reading :**
  - Hennessy & Patterson: 5.3, 5.4
- **Homework 3 due September 16**
- **Lab 3 due Friday September 25th**

Where Are We Now?

- **Where we've been:**
  - Looked at physical memory hierarchy
    - Hierarchy of size/speed tradeoffs to reduce average access time
    - Encompasses registers to disk
  - First look at how cache memories work -- hardware managed fast memory
- **Where we're going today:**
  - Look at virtual memory hierarchy
    - Address translation from program's memory space to physical memory hierarchy
    - Other uses of caching principles to speed up address translation (TLB)
- **Where we're going next:**
  - Data organization & management policies for caches
    - Note that these ideas scale throughout memory hierarchy ... caches are simply a convenient place to study them

Preview -- Virtual Memory

- **Evolution of Virtual Memory**
- **Address mapping with page tables**
- **Why virtual memory is useful**
- **How you make it fast**
  - Translation Lookaside Buffer (TLB)
  - Inverted page table

Simple Definition of Virtual Memory System

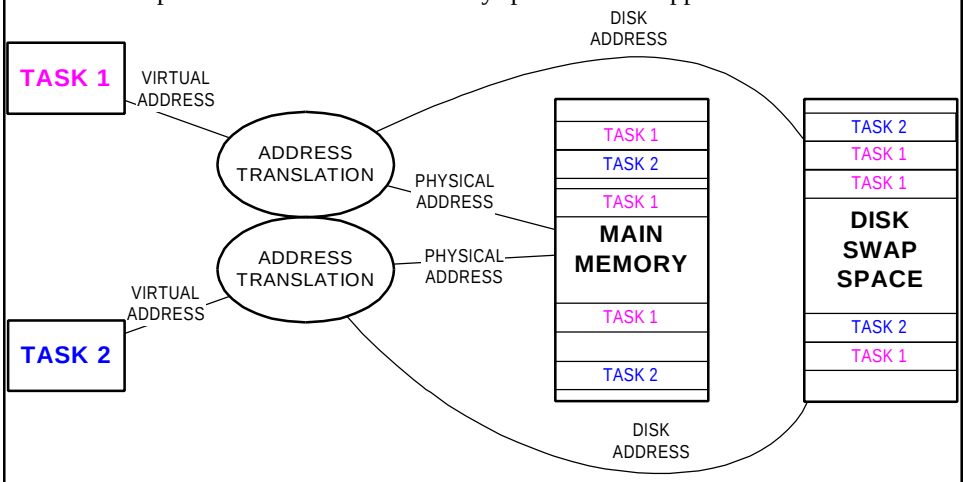
- **Virtual Memory is automatic address translation that provides:**
  - Decoupling of program's "name space" from physical location
  - Provides access to name space potentially greater in size than physical memory
  - Expandability of used name space without reallocation of existing memory
  - Protection from interference with name space by other tasks

- **Components that make virtual memory work include:**
  - Physical memory divided up into pages
  - A swap device (typically a disk) that holds pages not resident in physical memory (that's why it's referred to as *backing store* as well)
  - Address translation
    - Page tables to hold virtual-to-physical address mappings
    - Translation lookaside buffer is cache of translation information
  - Management software in the operating system

ADDRESS MAPPING -- A General Concept In The Memory Hierarchy

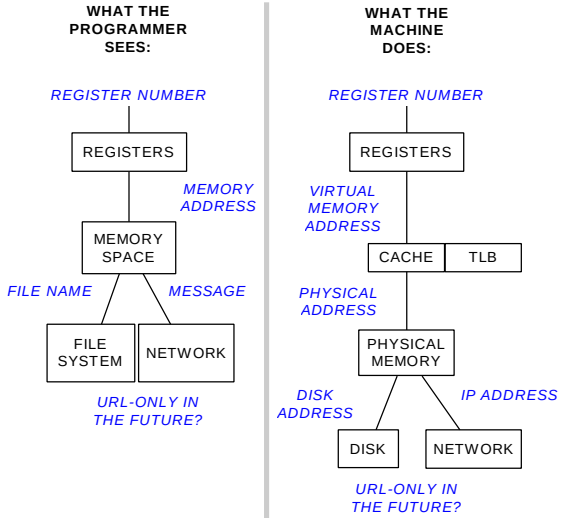
Address Mapping

- **Memory address mapping provides speed, relocation, and protection**
  - Locality can be exploited to keep "working set" in main memory
  - Portions of tasks can be moved around in main memory transparently
  - Attempts to exceed allocated memory space can be trapped



Address Renaming in the Memory Hierarchy

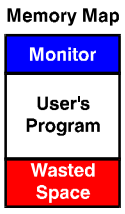
- **Programmer sees address hierarchy:**
  - Registers
  - "Memory"
  - Files
  - Messages
- **Machine hides hardware details:**
  - Virtual to physical memory mapping
  - File / virtual address to disk sector # mapping
  - Messaging destination to IP address/routing information



EVOLUTION OF VIRTUAL MEMORY

Single-Task Batch Systems

- **Originally -- one computer; one program**
  - Early computers -- reserve time for dedicated use by one person
  - Memory space (if any) not used is wasted
  - Time spent waiting for I/O is wasted -- CPU remains idle
    - CPUs were far more valuable that programmer hours -- want to keep it busy all the time



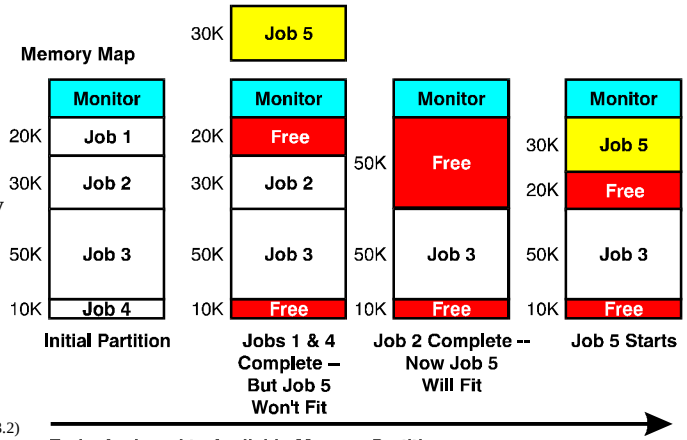
Simple Batch Monitor

(After Cragon Figure 3.2)

Multi-Tasking Batch Systems

- **Batch systems: one computer, several programs**
  - Permitted better efficiency while waiting for slow I/O devices
  - Memory dedicated to job as it is started; program addresses can be modified as the program is loaded
  - Partitioned Allocation -- Memory can become fragmented, resulting in reduced efficiency

Example: Job 5 must wait until 30 KB contiguous memory is available, not 30 KB fragmented while Job 2 is running

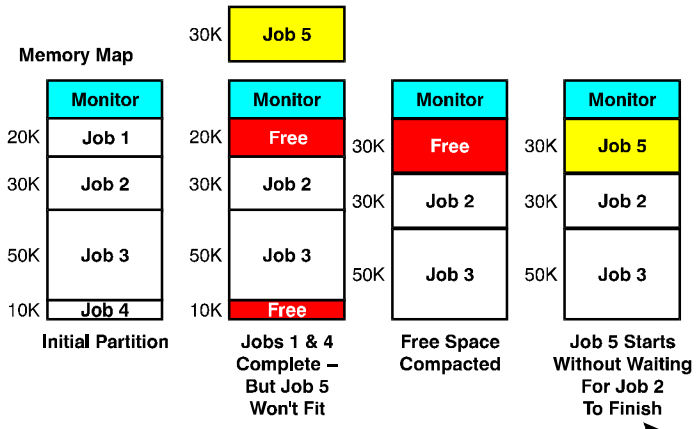


(After Cragon Figure 3.2)

Relocation Registers Provide Partial Solution

- **Base register permits moving program around as a unit**
  - Base register added to all memory references (*e.g.*, IBM S/360; 1964)
- **Bounds register provides rudimentary protection**
  - Addresses below Base or above Base+Bound give protection exceptions

Example: Memory is compacted to make room for Job 5 while Job 2 is still running



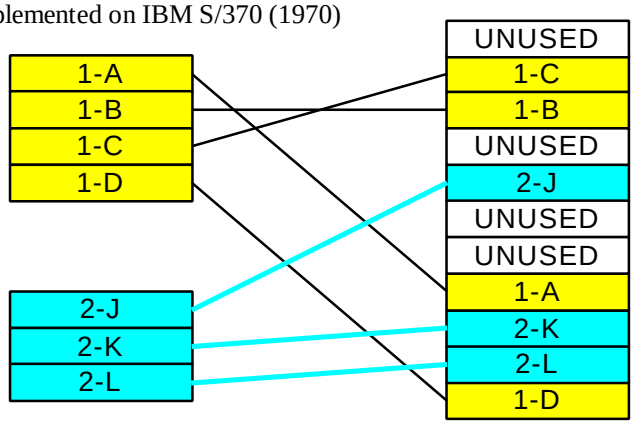
(After Cragon Figure 3.2)

But, Relocation Isn't Enough

- **Time is consumed copying memory blocks**
- **Programs may be bigger than physical memory**
  - Can be solved with overlays
    - Programmer defines many small programs that are loaded into memory as needed
    - Programmer-visible form of "virtual memory"
    - Painful to write -- programmer must do all partitioning manually
  - Or, can be solved with hardware to perform mapping...

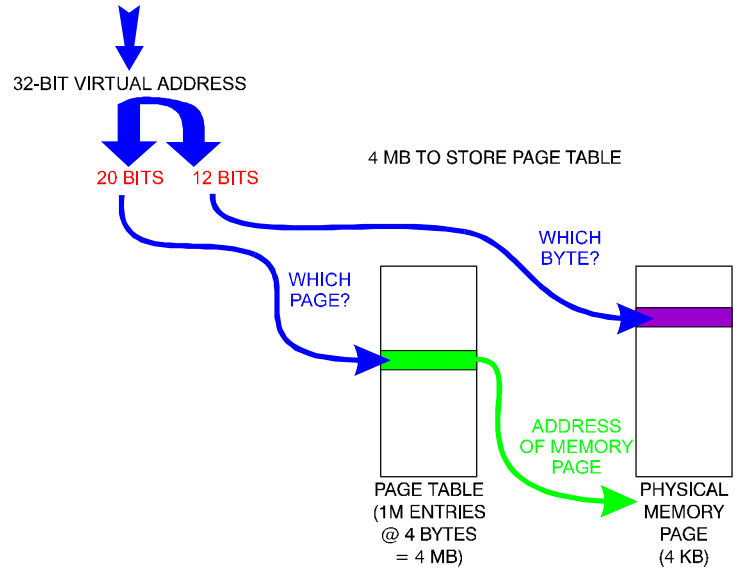
Virtual Memory Uses Mapped Pages

- **Memory space is treated as a set of pages that may be arbitrarily distributed among main memory and swap space**
  - Fixed size pages eliminate memory fragmentation
  - No need for compacting and cost of copying memory blocks
  - Base+Bound registers replaced with mapping tables (page tables)
  - For example, implemented on IBM S/370 (1970)

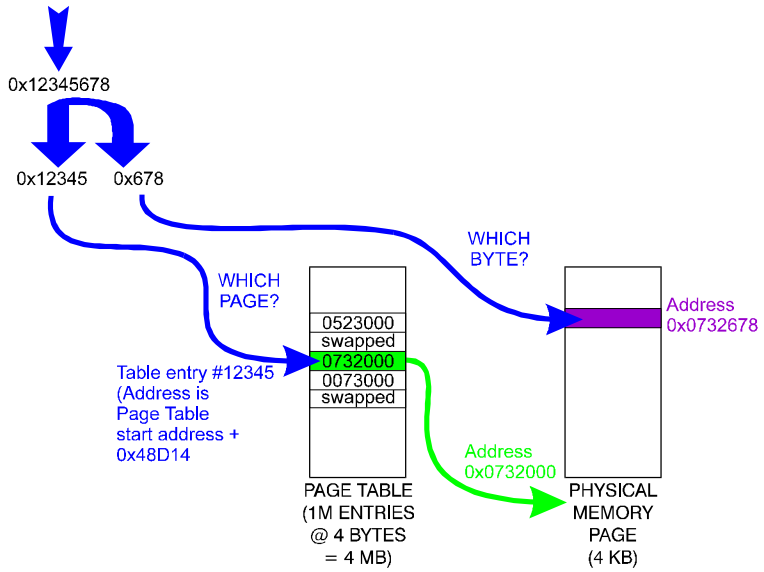


ADDRESS MAPPING WITH PAGE TABLES

1-Level Direct Page Table Translation



1-Level Page Table Example



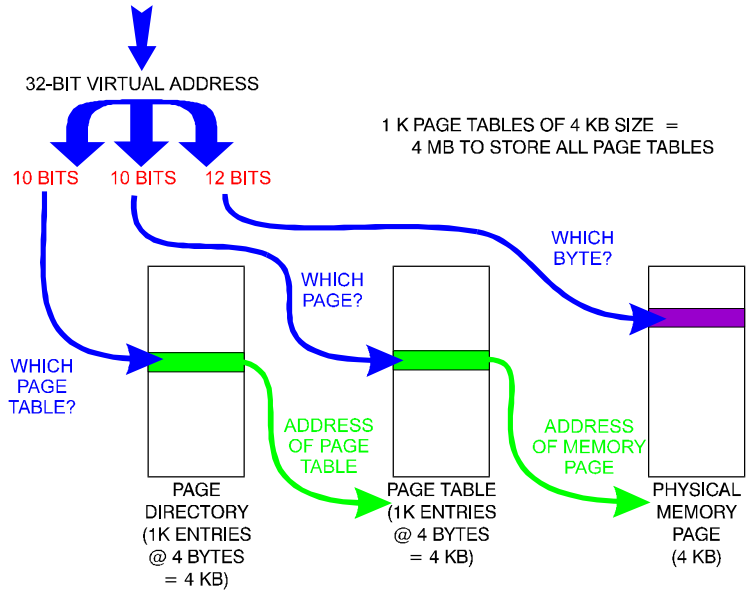
Format of Page Table Entry

- **Accomplishes mapping via use of look-up table**
- **Address**
  - Pointer to location of page in memory
  - Or, if page is swapped, can be disk address instead
- **Control bits**
  - Valid/Present bit
    - If set, page being pointed to is resident in memory
  - Modified/dirty bit
    - Set if at least one word in page has been modified
  - Referenced bit
    - Set if page has been referenced (with either read or write)
    - Used to support software replacement policies
  - Protection bits
    - Used to restrict access
    - For example, read-only access, or system-only access

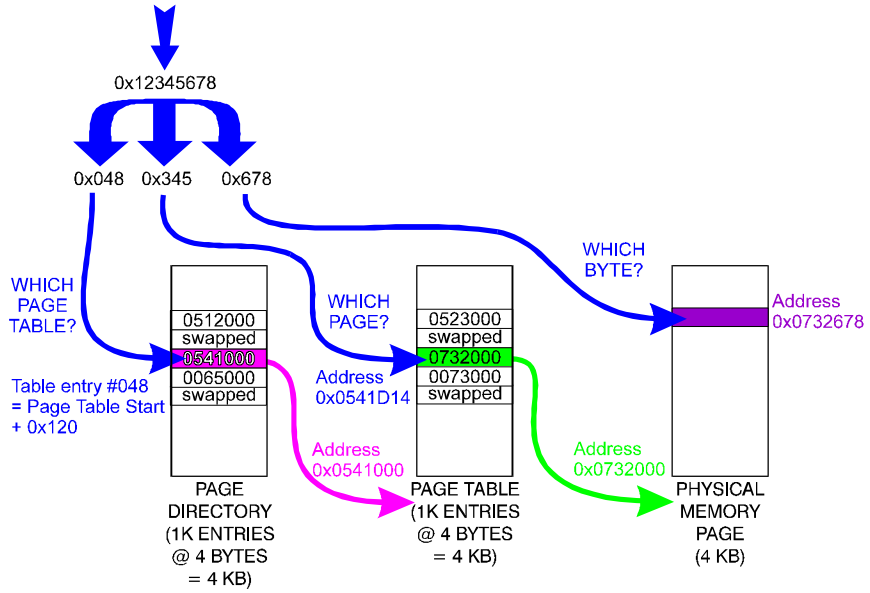
1-Level Table Simple, But Limited

- **For large virtual address spaces can grow very large**
  - Alpha 21164 has 43-bit virtual address Þ 1G page table entries » 8 GB
  - Impractical today -- that's a substantial part of a disk drive just for the page table
- **But, can keep only portions of the page table in physical memory**
  - VAX 11/780 can page portions of page table
  - Need not allocate entire page table (can allocate only first portion if desired)
  - Gets complicated... is there a better way?

2-Level Page Table



2-Level Page Table



2-Level Page Table Advantages

- **Only page directory & active page tables need to be in main memory**
  - Page tables can be created on demand
  - Page tables can be one page in size -- uniform paging mechanism for both virtual memory management and actual memory contents!
- **More sophisticated protection**
  - Can have a different page directory per process
    - Don't have to check owner of page table against process ID
    - Facilitates sharing of pages
- **Can scale up with 3-level scheme and keep page directory size small**
  - Example -- Alpha: say an 8 KB page holds 1K directory entries
    - 1 level: 1K page table = 8 MB
    - 2 levels: 1K page directory * 1K page tables = 8 GB
    - 3 levels: 1K page directory * 1K intermediate directories * 1K page tables = 8 TB = 2^{43}
      - » Alpha 21164 is specified to have a 43-bit virtual address space with 8KB pages...

WHY VIRTUAL MEMORY MATTERS

Simplified Programming Model

- **Decoupling of program's "name space" from physical location**
  - Programmer does not have to worry about managing physical address space
  - Hardware-assisted relocation without copying memory blocks
  - Today, *programmers are more valuable than machines*
- **Original motivation: programs can use single address model**
  - Memory seen by program can be greater than physical memory size
  - Far easier to program with than doing overlays
  - Automatic movement of data between disk & physical memory to maximize average speed (extending cache concept across entire memory hierarchy)

Flexibility In Memory Management

- **Programs have illusion of owning entire potential address space**
  - All programs can start at address 0 without being remapped by loader or using base register.
  - Machine need not have large contiguous address space for program (especially important with use of "malloc")
  - Can leave huge "holes" in address space with no penalty
    - For example, don't have to worry about multiple stacks growing together & colliding
- **Memory allocation is simple and painless**
  - Simply ask for any address in virtual memory space, and you can get it
  - No need to move other items in memory around to make space
  - Memory address space is relatively cheap -- you don't have to fill up holes in address usage with DRAM chips (or even with disk space)

Protection

- **Page fault mechanism provides hardware-assisted address range checks almost for free**
  - Can set permissions on per-page basis for selective sharing
  - Can detect some programming errors (such as dereferencing a null pointer)
  - But, granularity can be too large to be completely useful (4KB or 8KB chunks)
- **Isolates tasks to prevent memory corruption**
  - Detects attempts (innocent or otherwise) to access memory not owned by process

Other Virtual Memory Tricks

- **Lazy initialization of data**
  - Can fill page with zeros first time it is touched, not at start of program
- **Isolate data structures**
  - Sandwich data structure between unallocated pages
  - Catch "walking off the end" of data structure bugs (but only to granularity supported by paging system)
- **Garbage collected heap management**
  - Use page fault to perform allocation limit check instead of doing a comparison for every item allocated
  - Detect modification of "long-lived" data that is presumed not to be modified

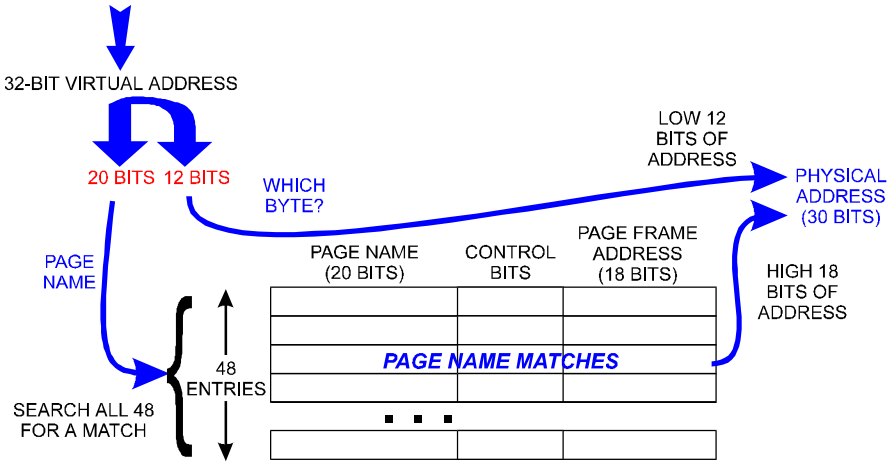
BUT HOW DO YOU MAKE IT FAST?

--

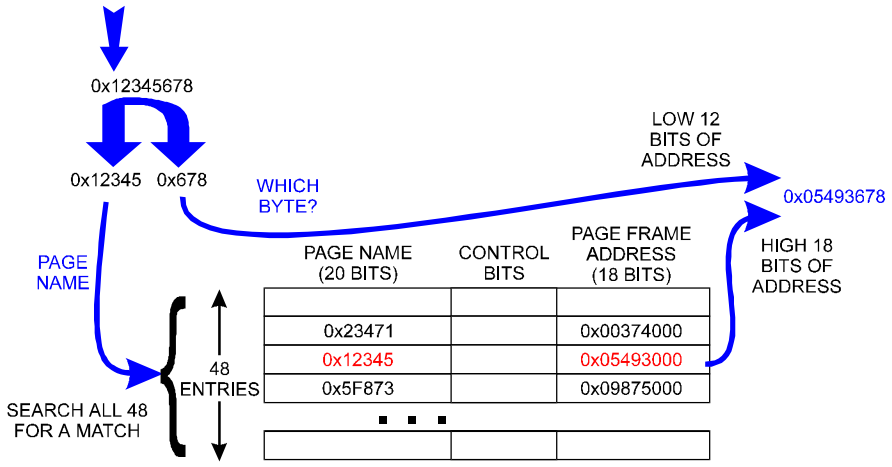
Translation Lookaside Buffer

How Do You Make Address Translation Fast?

- **Use a cache! -- Translation Lookaside Buffer (TLB)**
  - Caches recently used virtual to physical address translations
  - Used to translate addresses for accesses to virtual memory



TLB Operation Example

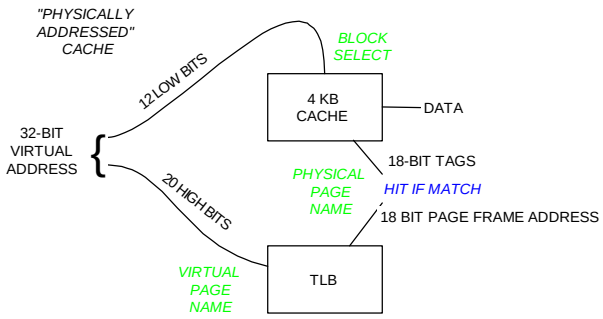


TLB Operation

- **Page Name stores virtual address of page for that translation entry**
  - May also contain process ID to avoid having to flush TLB on process switch
- **Page Frame Address stores physical memory location of that page**
- **Example control bits (MIPS R2000):**
  - Read-only
  - Non-cacheable
  - Valid
  - Global (globally accessible; unprotected access)
- **Classical TLB is fully associative**
  - All TLB entries are searched on each page translation
  - Works fine for a small number of entries -- use content-addressable memory cells

TLB Access Timing

- **Don't want TLB in critical path to cache**
  - But, TLBs are early select -- have to match page name before accessing memory address
  - Naive, slow approach is:
    TLB page name Þ TLB Page Frame Address Þ Cache Access
- **Solution: Access TLB & cache concurrently**



Note On Virtually Addressed Cache

- **Virtually addressed cache is accessed with virtual, not physical, addresses**
  - Tags stored in cache are virtual addresses
    - Cache hit determined by comparing tag with untranslated address
    - Address translation done only on cache miss
    - Must use process ID or flush caches on context switch
  - Complicates things because of aliasing (two virtual addresses can point to same physical address)
    - Easier to implement for I-cache than D-cache because no need to update multiple copies of data on a write operation
    - We'll look at this again in the multiprocessing/coherence lecture
- **For now, we'll only be talking about physically addressed caches**

A Hierarchy of Misses

- **TLB miss: translation for a page wasn't in the TLB**
  - Example Penalty: 50-100 clocks
- **Cache miss: page table entry wasn't found in the cache when reloading TLB**
  - Example Penalty: 6 clocks for L1 miss; 48 clocks for L2 miss
- **Page fault: page wasn't in physical memory ("physical memory miss")**
  - Example Penalty: millions of clocks
- **Note: multiple instances of any of these misses can occur for any single instruction execution**
  - TLB miss may require fetching page table which isn't in memory, and then cause a page fault when fetching the target page which isn't in memory
  - Some pages are "locked" into main memory to prevent page faults (analogous to software management of cache contents)

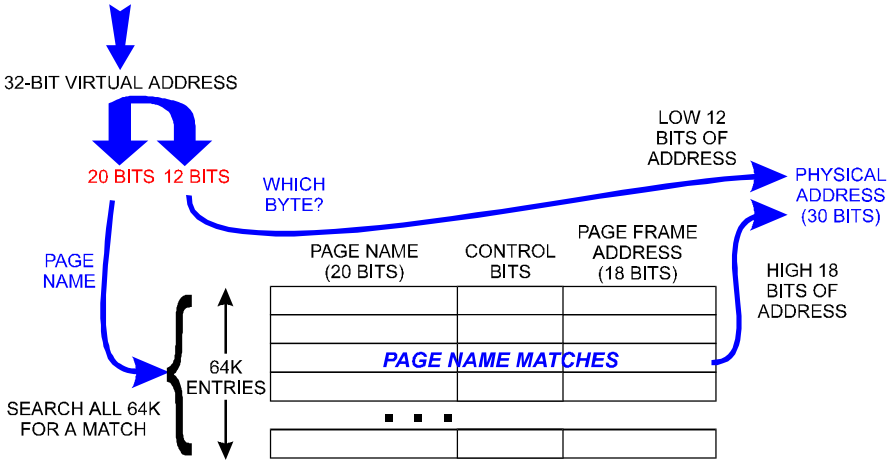
INVERTED PAGE TABLE

Problems With TLB + Direct Page Tables

- **Hierarchical page tables don't change fact that lots of page table entries are needed**
  - Can "lazily" create page tables
- **Large gaps in address space can complicate mapping**
  - Part of reason to use virtual memory is to remove constraints on addresses that programs use
  - If swap space is smaller than virtual memory space, must map swapped elements to disk
- **Assumes locality within the scope of a page table**
  - 1 page of memory used for every active page mapping
    - Locality assumed within 1K entries/page table = 4MB - 8 MB granularity
  - What if only 1 entry of a page table is relevant for memory contents?
    - Worst case is that if this locality is missing, factor of 2 space & swap speed penalty
  - Lack of locality can be fixed with an inverted page table...

Inverted Page Table » Big TLB

- **The significant difference is that there are thousands of entries in an Inverted Page Table, as opposed to tens of entries in a TLB**



Inverted Page Table Operation

- **Example: 512 MB DRAM on an Alpha with 8 KB pages**
  - 512 MB / 8 KB = 64K pages
  - Want a TLB big enough to hold mappings for all 64K pages
  - Use memory-resident translation table (64K * ~16 bytes = 1MB Þ 2% space penalty)
- **Same function as a TLB, except bigger**
  - Looks the same, but memory resident & handled by software
  - Must have enough entries to map all of physical memory
    - Number of entries proportional to physical memory size, not virtual address space
  - Can still use TLB to cache recently used inverted page table entries
- **Used to map virtual addresses to physical memory addresses**
  - Still need some other data structure to track swapped pages

REVIEW

Simple Definition of Virtual Memory System

- **Virtual Memory is automatic address translation that provides:**
  - Decoupling of program's "name space" from physical location
  - Provides access to name space potentially greater in size than physical memory
  - Expandability of used name space without reallocation of existing memory
  - Protection from interference with name space by other tasks
- **Components that make virtual memory work include:**
  - Physical memory divided up into pages
  - A swap device (typically a disk) that holds pages not resident in physical memory (that's why it's referred to as *backing store* as well)
  - Address translation
    - Page tables to hold virtual-to-physical address mappings
    - Translation lookaside buffer is cache of translation information
  - Management software in the operating system

Review

- **Evolution of Virtual Memory**
  - Complete software remapping
  - Base & bound registers
  - Paged allocation
- **Address mapping with page tables**
  - Single & multi-level page tables
- **Why virtual memory is useful**
  - Simple programming, flexibility, protection
- **How you make virtual memory fast**
  - Translation Lookaside Buffer (TLB)
  - Inverted page table

Virtual Memory Is Similar To Cache

	CACHE	VIRTUAL MEMORY
Implemented with	Cache SRAM	Main Memory DRAM
Baseline access via	Demand Fetching	Demand Paging
Misses to go	Main Memory	Swap Device (disk)
Size of transfered data set	Block (4-64 bytes)	Page (4KB - 8KB)
Mappings stored in	Tags	Page Tables

- **Both deal with similar implementation issues**
  - Selecting which block/page to replace
  - Dealing with modified data
- **But, the answers differ...**
  - Cache is mostly hardware; virtual memory has more software

Key Concepts

- **Latency**
  - Virtual memory is used to cache potential disk contents in main memory
  - Main memory is used to cache page table entries
  - TLB is used to cache page table entries
  - Inverted page tables are used to cache page table entries too
  - Disks are really slow, so it's worth some effort to get page fault rate low
- **Bandwidth**
  - Page sizes of 4KB or 8KB exploit bandwidth capabilities of system with block data transfers
- **Concurrency**
  - Accessing Cache & TLB in parallel takes address translation off critical path
- **Balance**
  - Need to balance amount of page table information in memory with amount of actual data
    - Inverted page tables avoid problems with degenerate cases