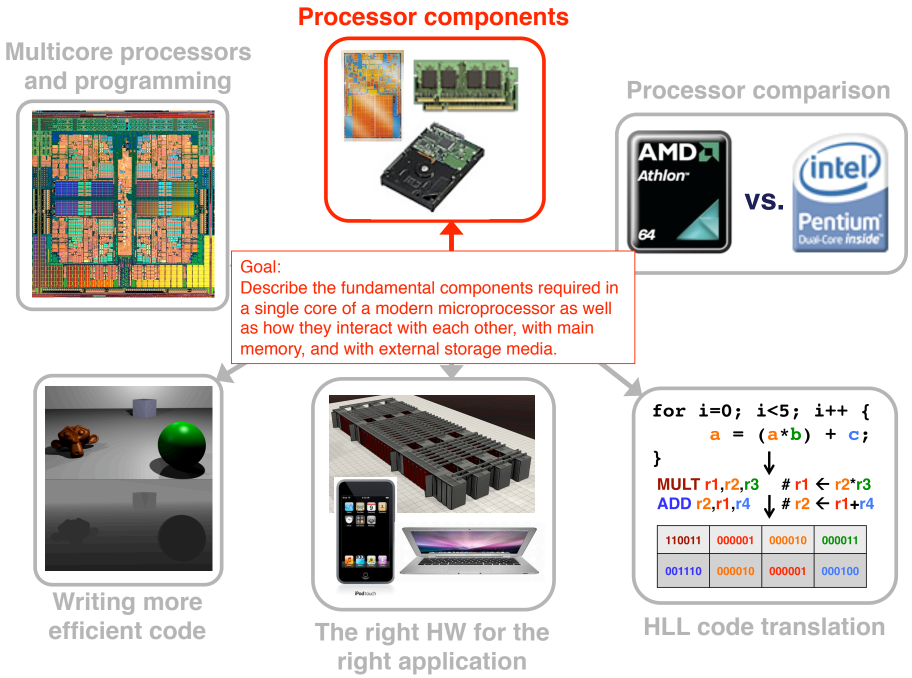
Lectures 21-22
Virtual Memory

Suggested Readings

- Readings
  - H&P: Chapter 5.4 and 5.5

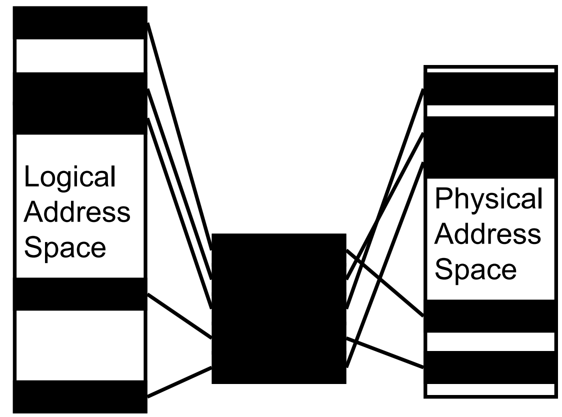


Introduction and Overview

Virtual Memory

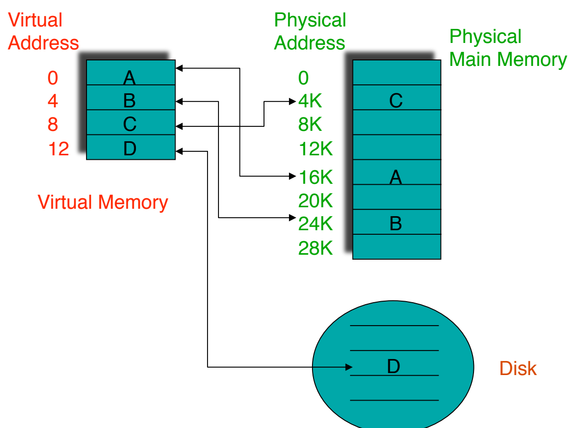
- Some facts of computer life…
  - Computers run lots of processes simultaneously
  - No full address space of memory for each process
    - Physical memory expensive and not dense - thus, too small
  - Must share smaller amounts of physical memory among many processes
- Virtual memory is the answer!
  - Divides physical memory into blocks, assigns them to different processes
    - Compiler assigns data to a “virtual” address.
      - VA translated to a real/physical somewhere in memory
    - Allows program to run anywhere; where is determined by a particular machine, OS
      - + Business: common SW on wide product line (w/o VM, sensitive to actual physical memory size)

Virtual address space greater than Logical address space



The gist of virtual memory

- Relieves problem of making a program that was too large to fit in physical memory – well…fit!
- Allows program to run in any location in physical memory
  - Really useful as you might want to run same program on lots machines…



Logical program is in contiguous VA space; here, pages: A, B, C, D;
(3 are in main memory and 1 is located on the disk)

Terminology and Practicalities

Some definitions and cache comparisons

- **The bad news:**
  - **In order to understand exactly how virtual memory works, we need to define some terms**
- **The good news:**
  - **Virtual memory is very similar to a cache structure**
- **So, some definitions/"analogies"**
  - **A "*page*" or "*segment*" of memory is analogous to a "block" in a cache**
  - **A "*page fault*" or "*address fault*" is analogous to a cache miss**

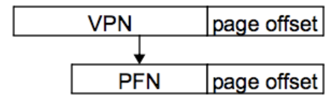
"real"/physical memory

so, if we go to main memory and our data isn't there, we need to get it from disk…

Virtual Memory: The Story

Translating VA to PA sort of like finding right cache entry with division of PA

- blocks called *pages*
- processes use *virtual addresses (VA)*
- physical memory uses *physical addresses (PA)*
- address divided into page offset, page number
  - virtual: virtual page number (VPN)
  - physical: page frame number (PFN)
- *address translation*: system maps VA to PA (VPN to PFN)
- e.g., 4KB pages, 32-bit machine, 64MB physical memory
  - 32-bit VA, 26-bit PA ($\log_2$64MB), 12-bit page offset ($\log_2$4KB)



Virtual Memory: The Four Questions

same four questions, different four answers

Might think about these 2 simultaneously

- page placement: fully (or very highly) associative
  - why?
- page identification: address translation
  - will discuss soon
- page replacement: sophisticated (LRU + "working set")
  - why?
- write strategy: always write-back + write-allocate
  - why?

The Answer Behind the Four Answers

backing store to main memory is *disk*

- memory is 50 to 100 slower than processor
- disk is 20 to 100 *thousand* times slower than memory
  - disk is 1 to 10 *million* times slower than processor

a VA miss (VPN has no PFN) is called a *page fault*

- high cost of page fault determines design
- full associativity + OS replacement ⇒ reduce miss rate
  - have time to let software get involved, make better decisions
- write-back reduces disk traffic
- page size usually large (4KB to 16KB) to amortize reads

Compare Levels of Memory Hierarchy

parameter	L1	L2	Memory
t_{hit}	1,2 cycles	5-15 cycles	10-150 cycles
t_{miss}	6-50 cycles	20-200 cycles	0.5-5M cycles
capacity	4-128KB	128KB-8MB	16MB-8GB
block size	8-64B	32-256B	4KB-16KB
associativity	1,2	2,4,8,16	full
write strategy	write-thru/back	write-back	write-back

t_{hit} and t_{miss} determine everything else

Idea:
Bring large chunks of data from disk to memory
(how big is OS?)

CSE 30321 – Lecture 21-22 – Virtual Memory

Virtual Memory

- Timing's tough with virtual memory:
  - $AMAT = T_{mem} + (1-h) * T_{disk}$
  - $= 100nS + (1-h) * 25,000,000nS$
- h (hit rate) had to be *incredibly* (almost unattainably) close to perfect to work
- so: VM is a "cache" but an odd one.

University of Notre Dame

CSE 30321 – Lecture 21-22 – Virtual Memory

Page Translation

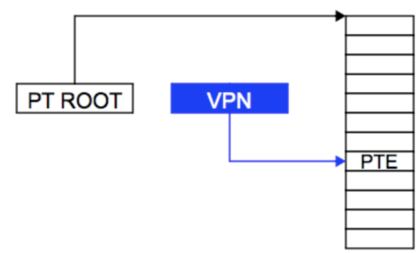
University of Notre Dame

CSE 30321 – Lecture 21-22 – Virtual Memory

Address Translation: Page Tables

OS performs address translation using a *page table*

- each process has its own page table
  - OS knows address of each process' page table
- a page table is an array of *page table entries (PTEs)*
  - one for each VPN of each process, indexed by VPN
- each PTE contains
  - PFN
  - permission
  - dirty bit
  - LRU state
  - e.g., 4-bytes total

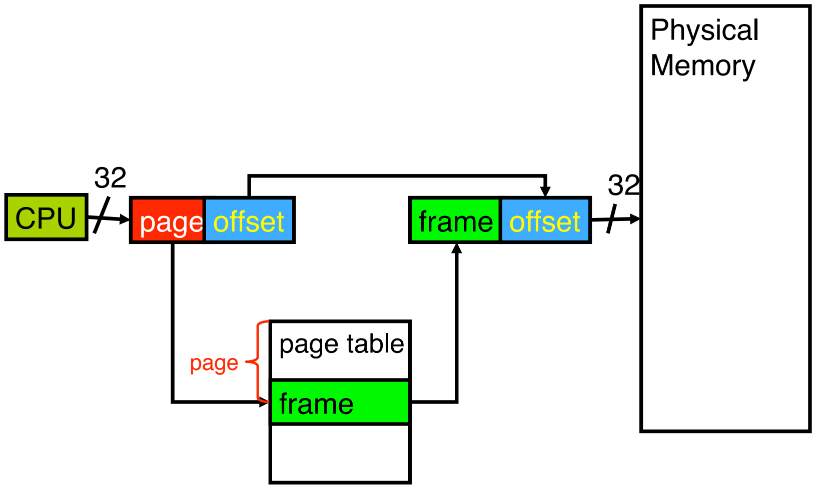


© 2004 by Lebeck, Sorin, Roth, Hill, Wood, Sohi, Smith, Vijaykumar, Lipasti

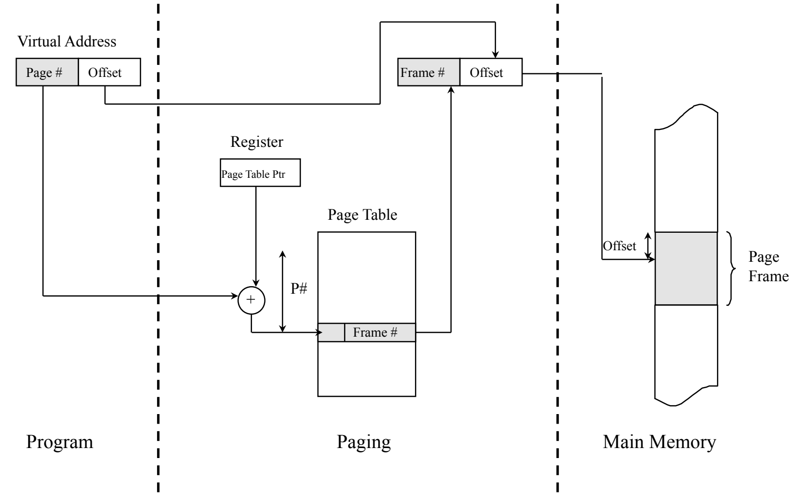
COMPSCI 220 / ECE 252 Lecture Notes
Storage Hierarchy II: Main Memory

University of Notre Dame

Review: Paging Hardware



Review: Address Translation



Test Yourself

A processor asks for the contents of virtual memory address 0x10020. The paging scheme in use breaks this into a VPN of 0x10 and an offset of 0x020.

PTR (a CPU register that holds the address of the page table) has a value of 0x100 indicating that this processes page table starts at location 0x100.

The machine uses word addressing and the page table entries are each one word long.

PTR	0x100

	VPN	OFFSET
Memory Reference	0x010	0x020

Test Yourself

ADDR	CONTENTS
0x00000	0x00000
0x00100	0x00010
0x00110	0x00022
0x00120	0x00045
0x00130	0x00078
0x00145	0x00010
0x10000	0x03333
0x10020	0x04444
0x22000	0x01111
0x22020	0x02222
0x45000	0x05555
0x45020	0x06666

PTR	0x100

	VPN	OFFSET
Memory Reference	0x010	0x020

- What is the physical address calculated?

1. 10020
2. 22020
3. 45000
4. 45020
5. none of the above

Test Yourself

ADDR	CONTENTS
0x00000	0x00000
0x00100	0x00010
0x00110	0x00022
0x00120	0x00045
0x00130	0x00078
0x00145	0x00010
0x10000	0x03333
0x10020	0x04444
0x22000	0x01111
0x22020	0x02222
0x45000	0x05555
0x45020	0x06666

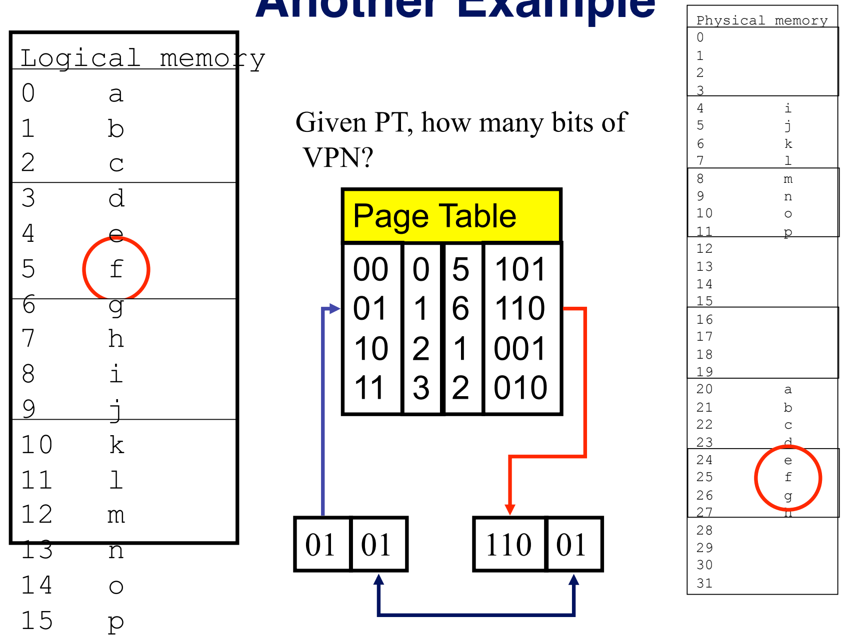
PTR	0x100

	VPN	OFFSET
Memory Reference	0x010	0x020

- What is the physical address calculated?
- What is the contents of this address returned to the processor?
- How many memory accesses in total were required to obtain the contents of the desired address?

Another Example

Given PT, how many bits of VPN?



Relative Sizes

Page Table Size

page table size

- example #1: 32-bit VA, 4KB pages, 4-byte PTE
  - 1M pages (32 bits = 4 GB address space / 4 KB page = 1M pages)
  - 1M pages*4bytes = 4MB page table (bad, but could be worse)
- example #2: 64-bit VA, 4KB pages, 4-byte PTE
  - 4P pages, 16PB page table (not a viable option)
- upshot: can't have page tables of this size in memory

techniques for reducing page table size

- multi-level page tables
- inverted page tables

COMPSCI 220 / ECE 252 Lecture Notes
Storage Hierarchy II: Main Memory

Block replacement

- Which block should be replaced on a virtual memory miss?
  - Again, we'll stick with the strategy that it's a good thing to eliminate page faults
  - Therefore, we want to replace the LRU block
    - Many machines use a "use" or "reference" bit
    - Periodically reset
    - Gives the OS an estimation of which pages are referenced

Writing a block

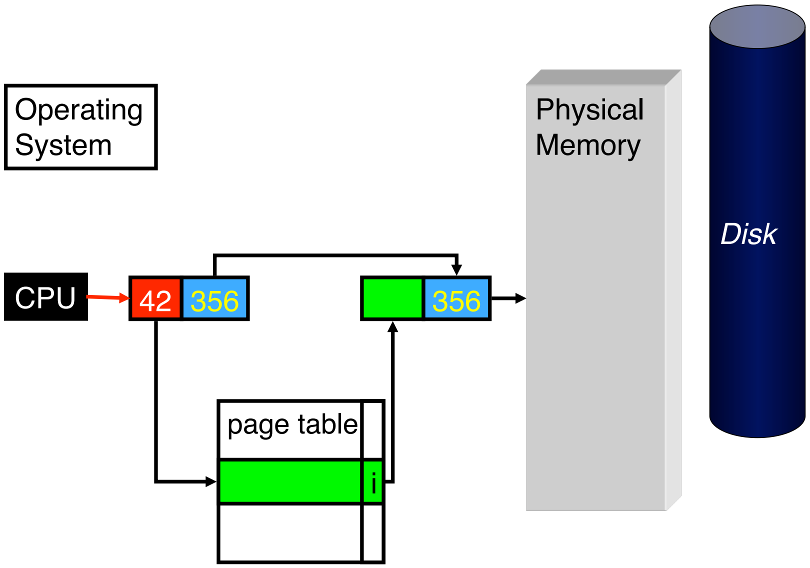
- What happens on a write?
  - We don't even want to think about a write through policy!
    - Time with accesses, VM, hard disk, etc. is so great that this is not practical
  - Instead, a write back policy is used with a dirty bit to tell if a block has been written

Introduction to TLBs

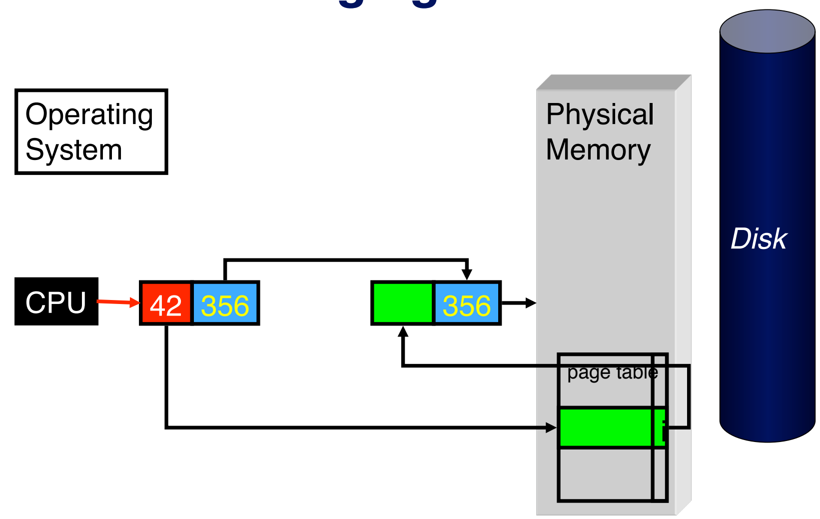
Page tables and lookups…

- 1. it's slow! We've turned every access to memory into two accesses to memory
  - solution: add a specialized "cache" called a "translation lookaside buffer (TLB)" inside the processor
- 2. it's still huge!
  - even worse: we're ultimately going to have a page table for every *process*. Suppose 1024 processes, that's 4GB of page tables!

Paging/VM

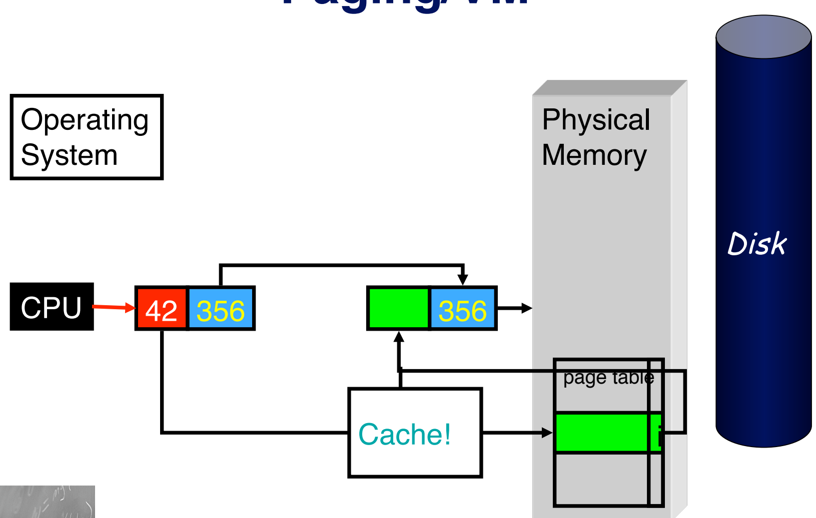


Paging/VM



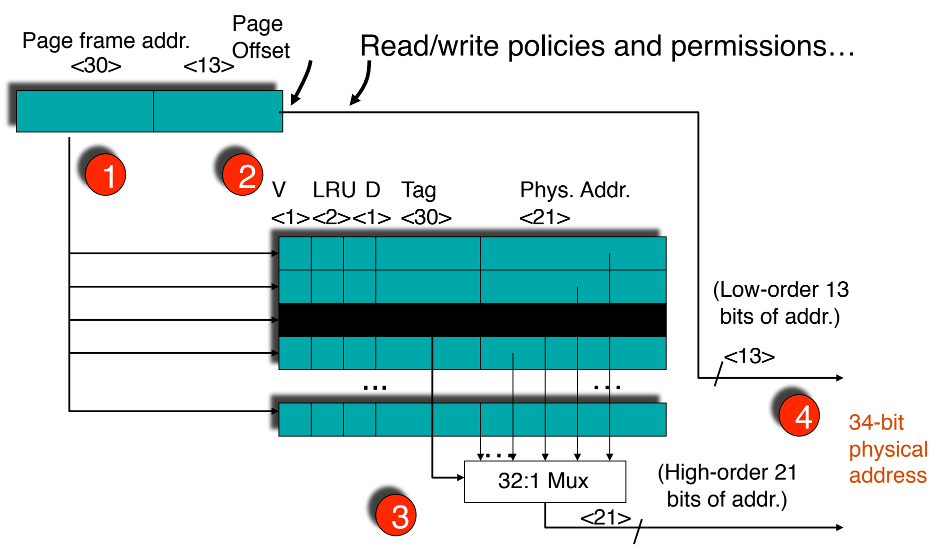
Place page table in physical memory
However: this doubles the time per memory access!!

Paging/VM



Special-purpose cache for translations
Historically called the TLB: Translation Lookaside Buffer

An example of a TLB



Review: Translation Cache

A way to speed up translation is to use a special cache of recently used page table entries -- this has many names, but the most frequently used is *Translation Lookaside Buffer* or *TLB*

Virtual Page #	Physical Frame #	Dirty	Ref	Valid	Access

tag

Really just a cache (a special-purpose cache) on the page table mappings

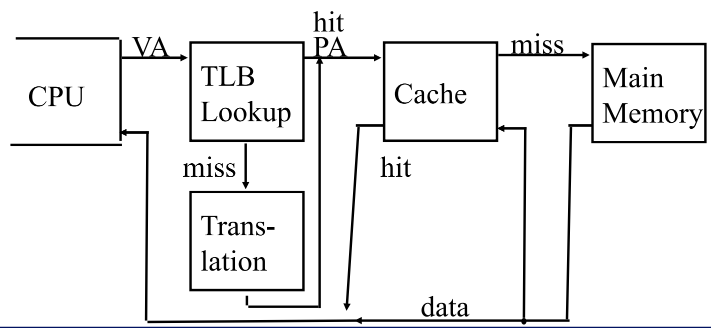
TLB access time comparable to cache access time (much less than main memory access time)

Review: Translation Cache

Just like any other cache, the TLB can be organized as fully associative, set associative, or direct mapped

TLBs are usually small, typically not more than 128 - 256 entries even on high end machines. This permits fully associative lookup on these machines. Most mid-range machines use small n-way set associative organizations.

Translation with a TLB

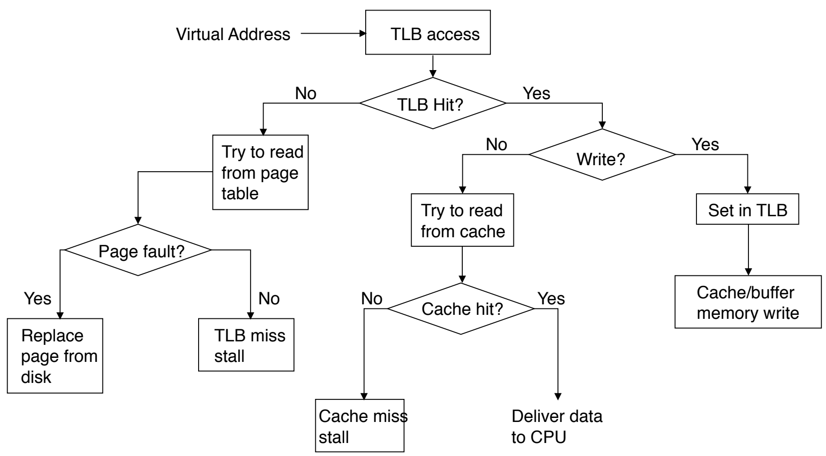


A Full Address Translation

The "big picture" and TLBs

- **Address translation is usually on the critical path…**
  - **…which determines the clock cycle time of the μP**
- **Even in the simplest cache, TLB values must be read and compared**

The “big picture” and TLBs



CSE 30321 – Lecture 21-22 – Virtual Memory

Examples

University of Notre Dame