Lecture 20
Virtual Memory

Suggested reading:
(HP Chapter 5.4-5.5)

Processor components

Multicore processors and programming

Processor comparison

AMD Athlon 64 vs. intel Pentium Dual-Core inside

CSE 30321

```
for i=0; i<5; i++ {
    a = (a*b) + c;
}
MULT r1,r2,r3   # r1 ← r2*r3
ADD r2,r1,r4    # r2 ← r1+r4
```

110011	000001	000010	000011
001110	000010	000001	000100

Writing more efficient code

The right HW for the right application

HLL code translation
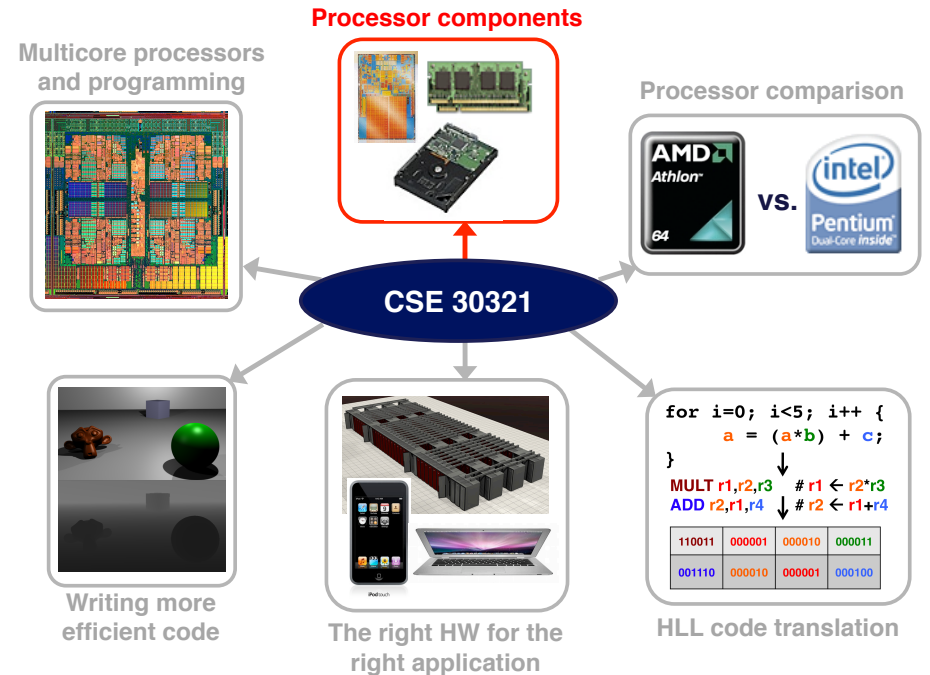
Fundamental lesson(s)

- **Computers run lots of processes with large, virtual address spaces.**

- **The next 2 lectures consider how that address space is managed, and how a virtual address is translated to a physical address**
  - **(so that your processor can actually get the data that it needs to work with).**

Why it's important…

- **This material is central to your OS course.**

- **Essentially, you will learn how the OS (which manages different processes) interfaces with HW such that information/data can actually be accessed and processed.**

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**
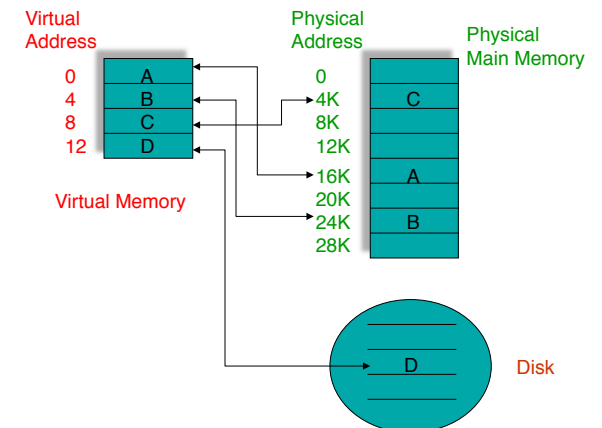   - » **(w/o VM, sensitive to actual physical memory size)**

VA space > Logical address space
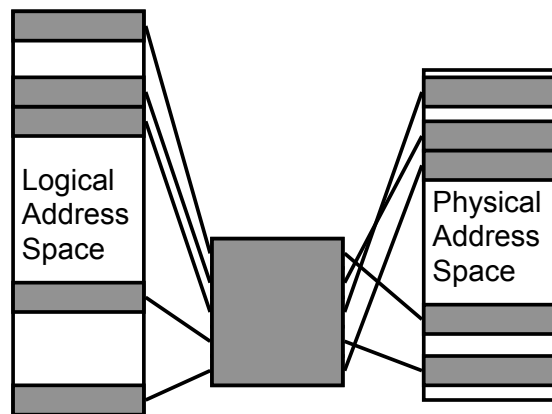
The gist of virtual memory

- · **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)**

Part A

Some definitions (& cache analogies)

- **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…

9

TERMINOLOGY AND PRACTICALITIES

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)

VPN	page offset

↓

PFN	page offset
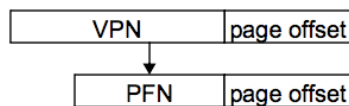
Virtual Memory: The Four Questions

same four questions, different four answers

- **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?

Might think about these 2 simultaneously

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

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

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

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**

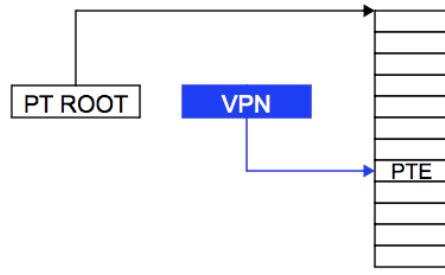- **so: VM is a "cache" but an odd one.**

PAGE TRANSLATION
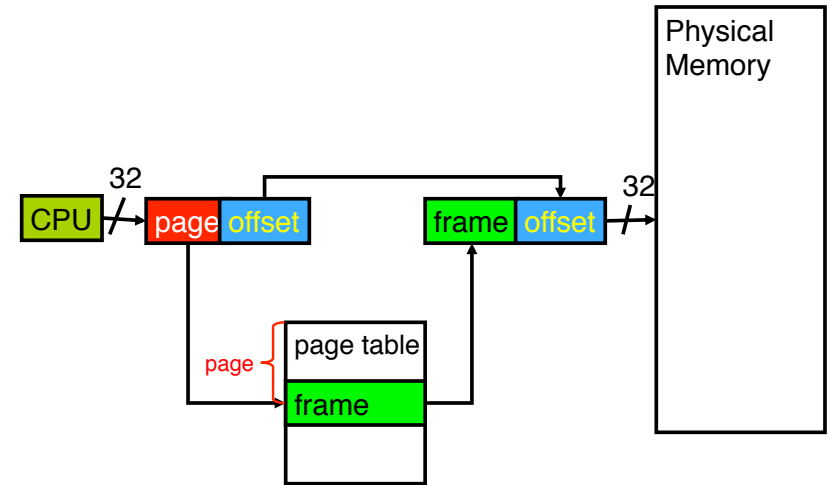
Part B

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
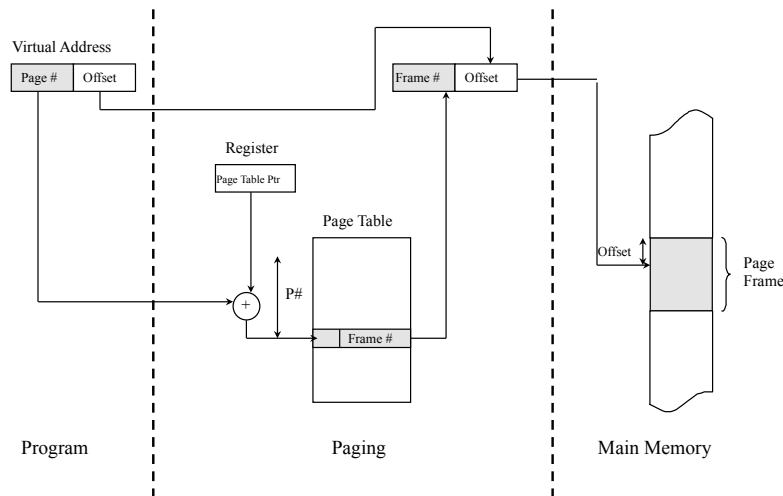 - LRU state
  - e.g., 4-bytes total

PT ROOT   VPN   PTE

Review: Paging Hardware

CPU — 32 — page | offset → frame | offset — 32 → Physical Memory

page → page table → frame → frame

Review: Address Translation

Virtual Address: Page # | Offset

Register: Page Table Ptr

Page Table: Frame #

Frame # | Offset

P#

Offset → Page Frame

Program | Paging | Main Memory

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.
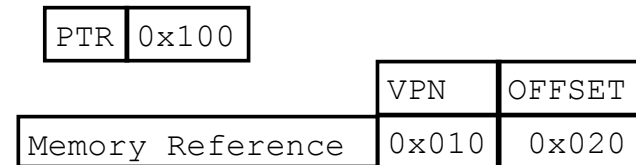
PTR	0x100

	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

Logical memory

0	a
1	b
2	c
3	d
4	e
5	f
6	g
7	h
8	i
9	j
10	k
11	l
12	m
13	n
14	o
15	p

Given PT, how many bits of VPN?

Page Table

00	0	5	101
01	1	6	110
10	2	1	001
11	3	2	010

01	01

110	01
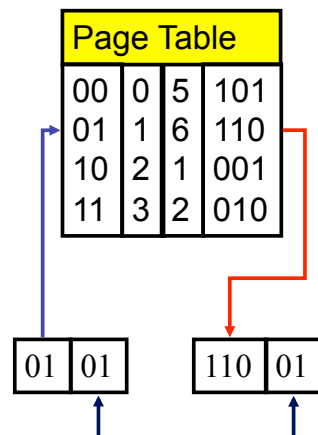
Physical memory

0	
1	
2	
3	
4	i
5	j
6	k
7	l
8	m
9	n
10	o
11	p
12	
13	
14	
15	
16	
17	
18	
19	
20	a
21	b
22	c
23	d
24	e
25	f
26	g
27	h
28	
29	
30	
31	