

UNIVERSITY OF MASSACHUSETTS
Dept. of Electrical & Computer Engineering

Computer Architecture
ECE 568/668

Part 11

Memory Hierarchy - I

Israel Koren
Fall 2011

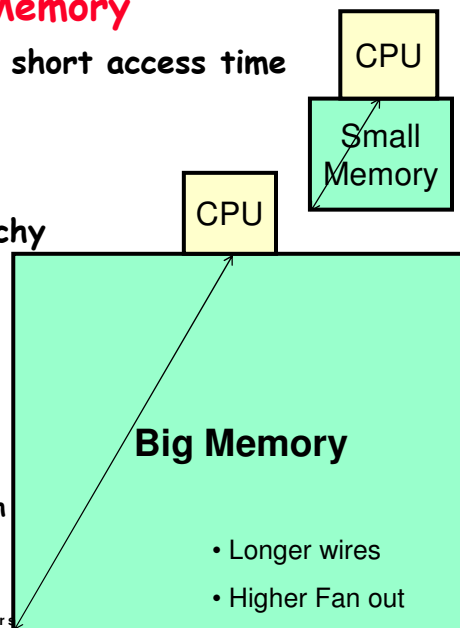
ECE568/Koren Part.11 .1

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Ideal Memory

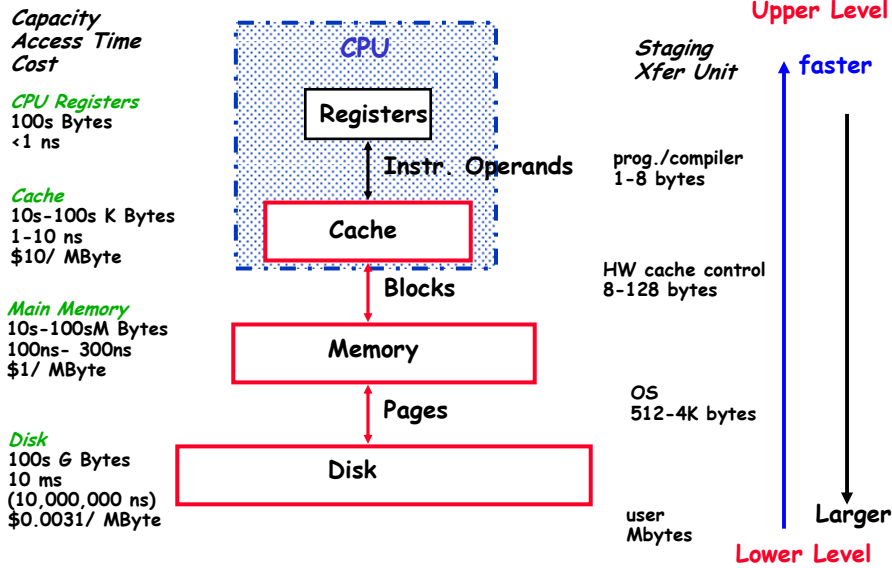
- ◆ Infinitely large and very short access time
- ◆ However
 - Very expensive
 - Technologically infeasible
- ◆ Solution: Memory hierarchy
 - Large and slow units, and
 - Small and fast units
- ◆ Levels in hierarchy characterized by
 - Total capacity
 - Access time
 - Transfer rate - bandwidth
 - Unit of transfer
 - Cost per byte



ECE568/Koren Part.11 .2

Adapted from UCB and other sources

Levels of the Memory Hierarchy

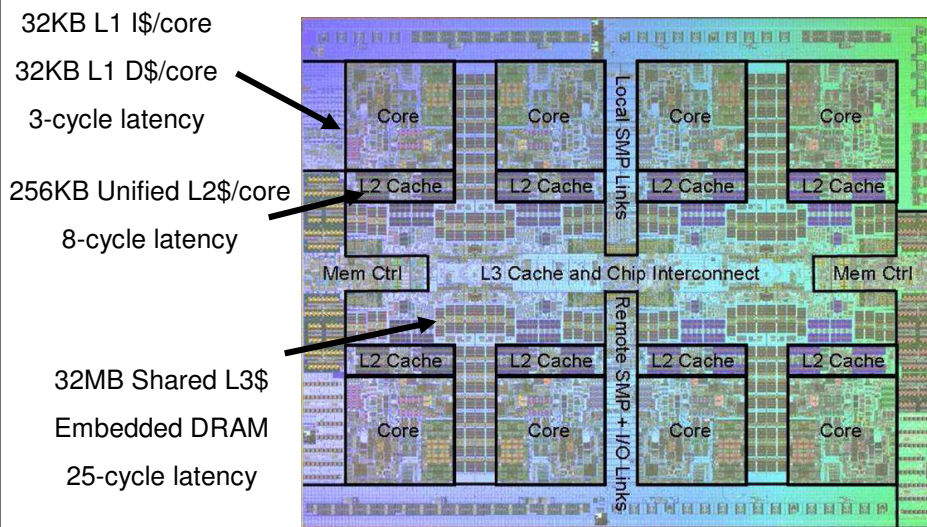


ECE568/Koren Part.11 .3

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Power 7 On-Chip Caches [IBM 2009]



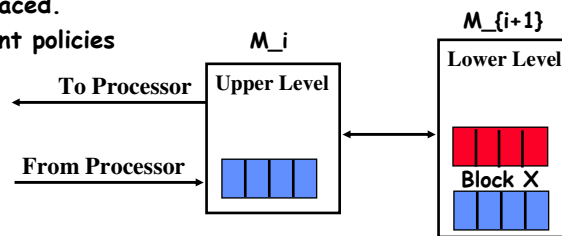
ECE568/Koren Part.11 .4

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Major Properties

- ◆ **Inclusion Property** - $M_i \subseteq M_{i+1}$
 - M_i - upper level, closer to the CPU
- ◆ **Coherence property** - copies in successive memory levels must be consistent
- ◆ **Two update methods**
 - (i) Write-through - immediate update
 - (ii) Write-back - delay update of M_{i+1} until the item in M_i is replaced.
 - » Replacement policies



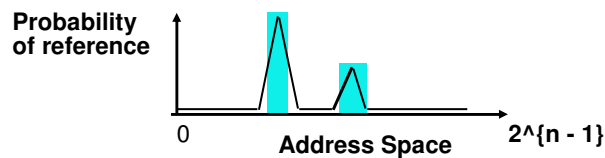
ECE568/Koren Part.11 .5

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

The Principle of Locality

- ◆ **The Principle of Locality:**
 - Program access a relatively small portion of the address space at any instant of time.
- ◆ **Two Different Types of Locality:**
 - **Temporal Locality** (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
 - **Spatial Locality** (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straightline code, array access)



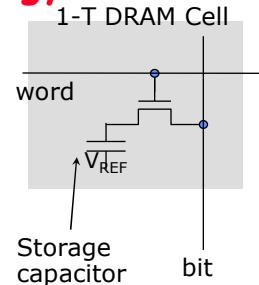
ECE568/Koren Part.11 .6

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Memory Hierarchy Technology

- ◆ **Random Access: same access time for all locations**
 - **DRAM:** Dynamic Random Access Memory
 - » High density, low power, cheap, but slow
 - » Dynamic: need to be "refreshed" regularly
 - **SRAM:** Static Random Access Memory
 - » Low density, high power, expensive, fast
 - » Static: content will last "forever" (until lose power)
 - **The Main Memory: DRAMs + Caches: SRAMs**
- ◆ **"Non-so-random" Access Technology:**
 - Access time varies from location to location and from time to time
 - Examples: Disk, CDROM
- ◆ **Sequential Access Technology: access time linear in location (e.g., Tape)**



ECE568/Koren Part.11 .7

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Memory Hierarchy - General Principles

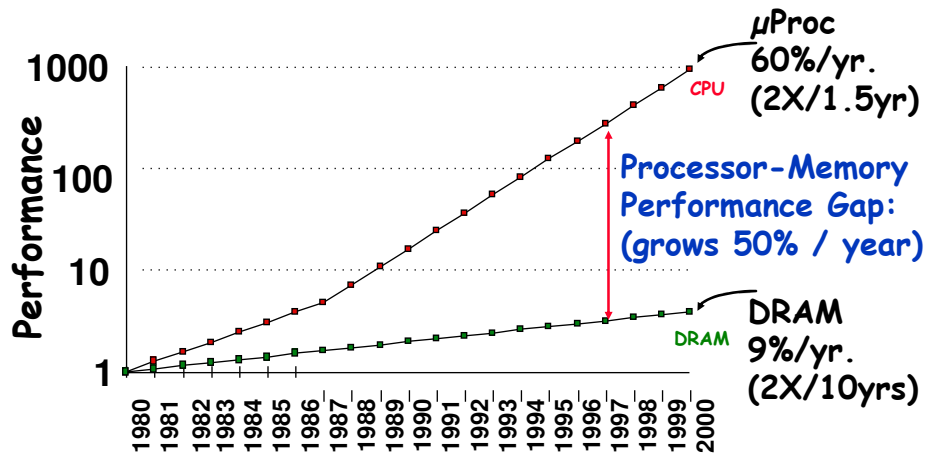
- ◆ **Solve the problems of**
 - Speed gap
 - Memory size
- ◆ **Cache - Main memory: Speed Block**
- ◆ **Main memory - Disk: Capacity Page**
- ◆ **A computer system may have one of these or both**

ECE568/Koren Part.11 .8

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Processor-DRAM Memory Gap (latency)



Four-issue 3GHz superscalar accessing 100ns DRAM could execute 1,200 instructions during time for one memory access!

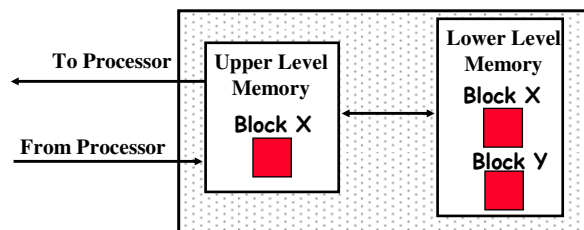
ECE568/Koren Part.11 .9

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Memory Hierarchy: Terminology

- ◆ **Hit**: data appears in some block in the upper level (example: Block X)
 - **Hit_Rate**: the fraction of memory access found in the upper level
- ◆ **Miss**: data needs to be retrieved from a block in the lower level (Block Y)
 - **Miss_Rate** = $1 - (\text{Hit_Rate})$

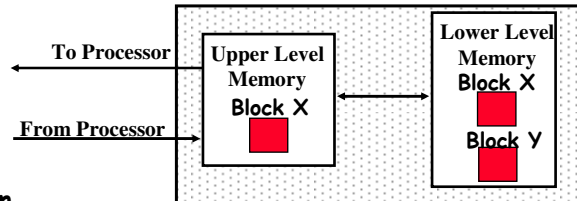


ECE568/Koren Part.11 .10

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Average Memory Access time (AMAT)



- ◆ Simplified expression

$$AMAT = h_{upper} \times T_{upper} + (1 - h_{upper}) \times T_{lower}$$

- ◆ A more precise expression

$$AMAT = Hit_Rate \times Hit_Time + Miss_Rate \times Miss_Penalty$$

$$Hit_time = T_{upper} + Time_to_determine_hit/miss$$

$$Miss_penalty = Time_to_determine_hit/miss + Time_to_replace_a_block_in_the_upper_level + Time_to_deliver_the_data_to_processor$$

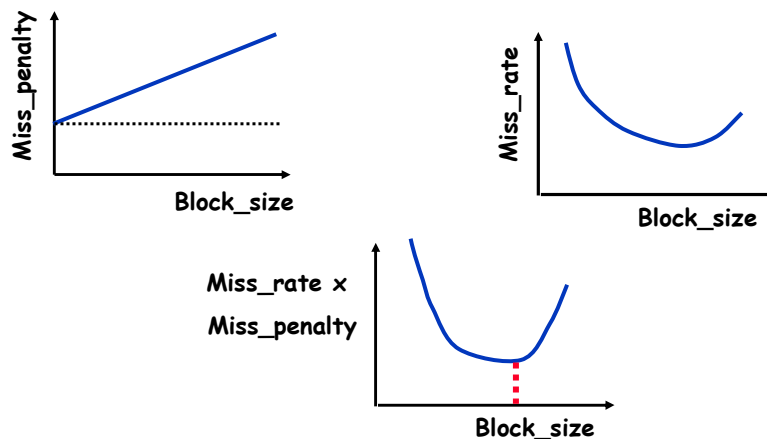
- ◆ Hit_Time \ll Miss_Penalty (50 cycles upon miss in cache; 500 instructions on Alpha 21264 upon miss in main memory)

ECE568/Koren Part.11 .11

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Block (Page) Size



ECE568/Koren Part.11 .12

Copyright Koren UMass 2011

Impact on Processor Performance

- ◆ Given a processor with
 - 1 cycle on-chip cache
 - Base CPI = 1.1
 - 50% arith/logic, 30% ld/st, 20% control
- ◆ Suppose that 10% of memory data operations get 50 cycle miss penalty
- ◆ Suppose that 1% of instructions fetch get same miss penalty
- ◆ $CPI = \text{Base CPI} + \text{average stalls per instruction}$
 $= 1.1(\text{cycles/ins}) + [0.30 (\text{DataMops/ins}) \times 0.10 (\text{miss/DataMop}) \times 50 (\text{cycle/miss})] + [1 (\text{InstMop/ins}) \times 0.01 (\text{miss/InstMop}) \times 50 (\text{cycle/miss})]$
 $= (1.1 + 1.5 + .5) \text{ cycle/ins} = 3.1$
- ◆ 64% of the time the proc is stalled waiting for memory!
- ◆ $\text{AvgMemAccessTime} = (1/1.3) \times [1 + 0.01 \times 50] + (0.3/1.3) \times [1 + 0.1 \times 50] = 2.54$

Base CPI	1.1
Data Miss	1.5
Inst Miss	0.5

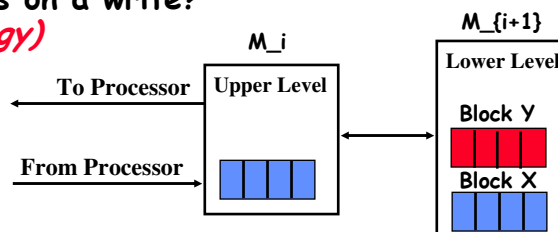
ECE568/Koren Part.11 .13

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Four Questions for Memory Hierarchy

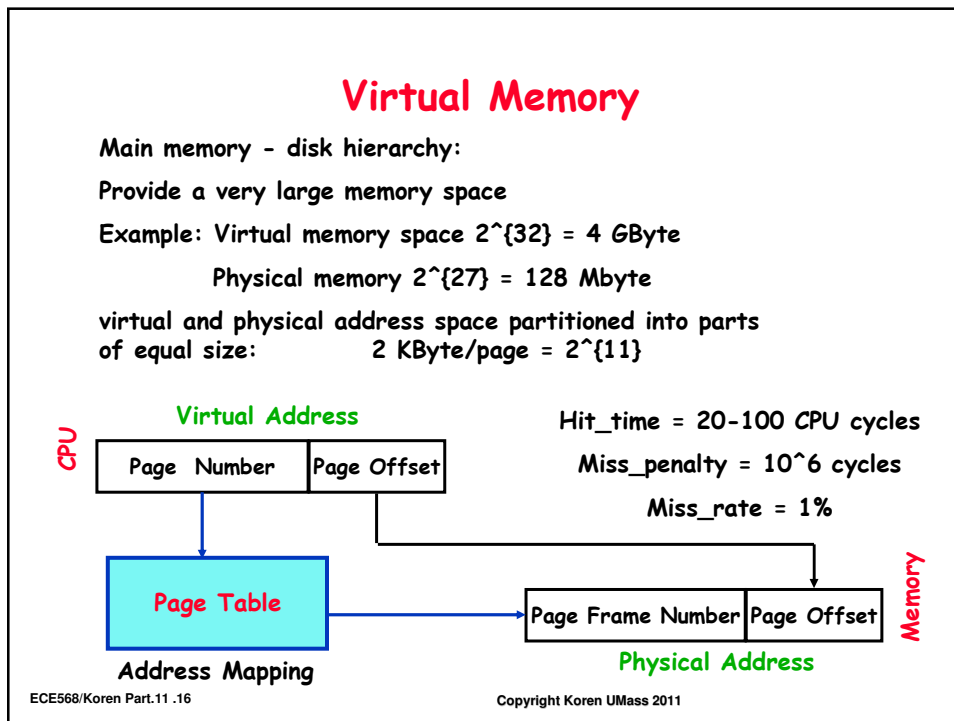
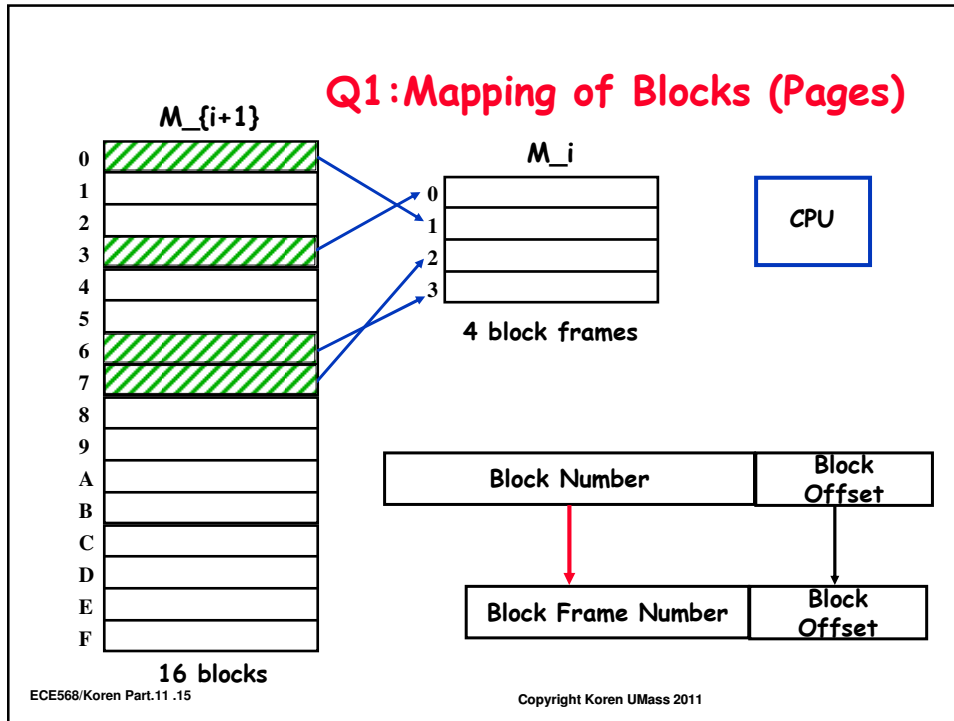
- ◆ Q1: Where can a block be placed in the upper level?
(Block placement)
- ◆ Q2: How is a block found if it is in the upper level?
(Block identification)
- ◆ Q3: Which block should be replaced on a miss?
(Block replacement)
- ◆ Q4: What happens on a write?
(Write strategy)



ECE568/Koren Part.11 .14

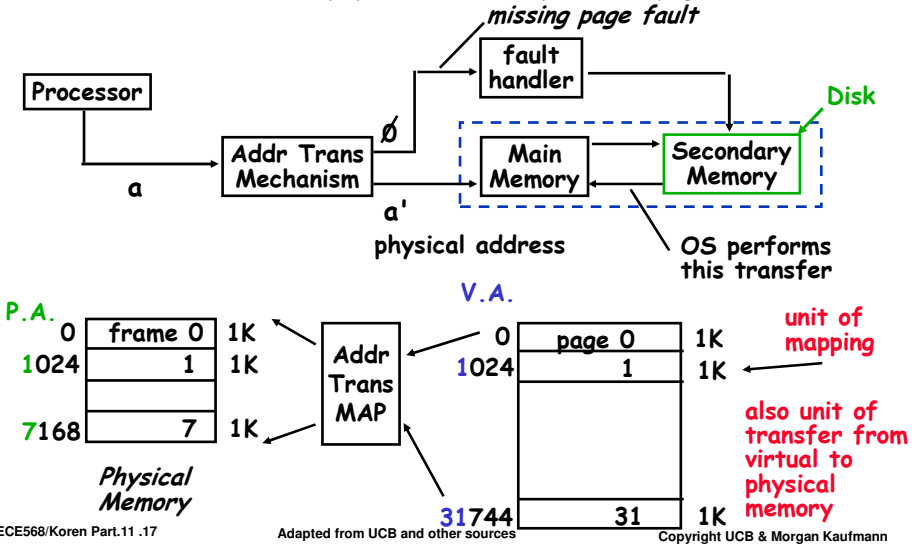
Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

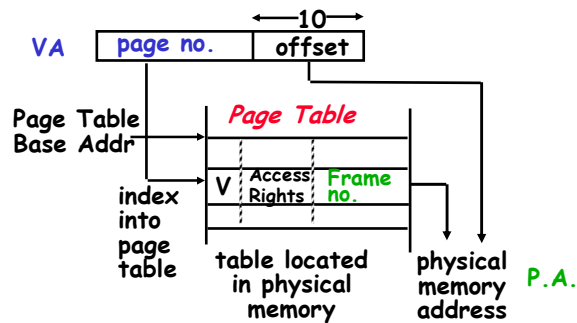
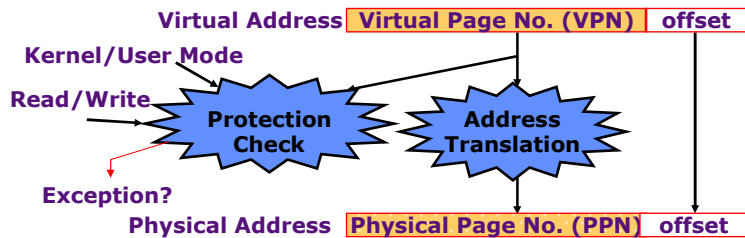


Address Mapping

$V = \{0, 1, \dots, n - 1\}$ virtual address space - n pages
 $M = \{0, 1, \dots, m - 1\}$ physical address space - m page frames $n \gg m$



Paging Organization (Protection)

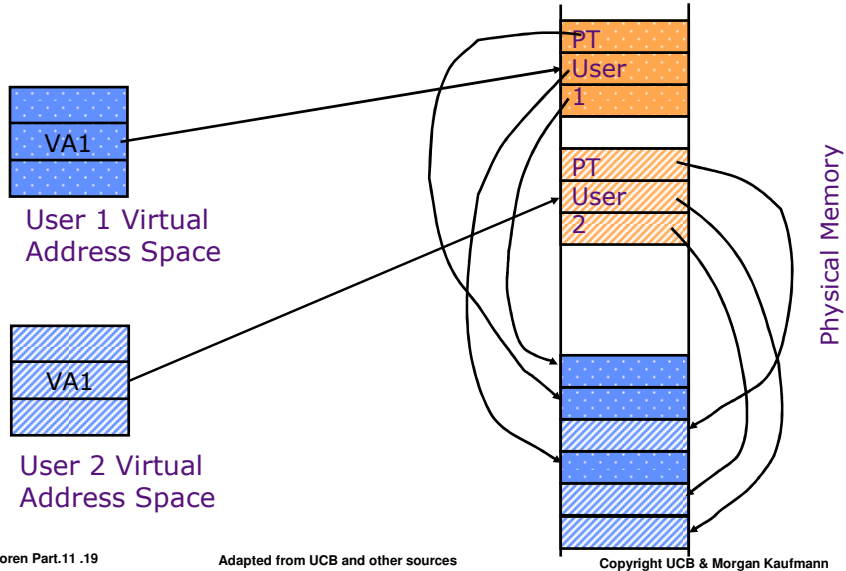


ECE568/Koren Part.11 .18

Adapted from UCB and other sources

Copyright UCB & Morgan Kaufmann

Page Tables in Physical Memory



Page Table

◆ Page Table Entry (PTE) contains:

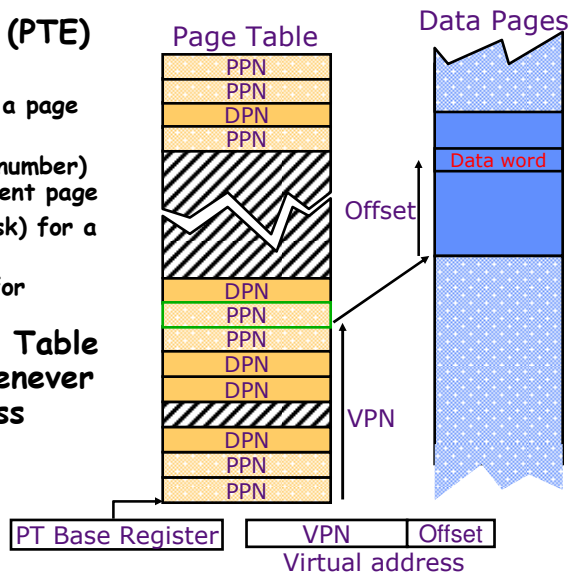
- A bit to indicate if a page exists (V)

PPN (physical page number) for a memory-resident page

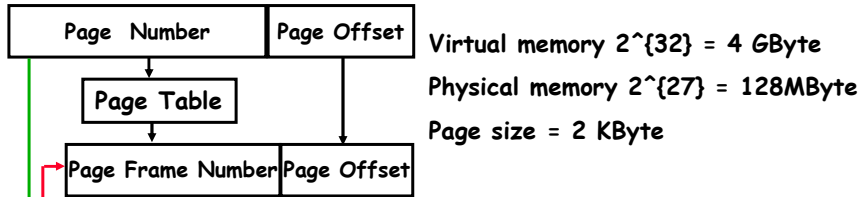
DPN (address on disk) for a page on the disk

- Access rights bits for protection

◆ OS sets the Page Table Base Register whenever active user process changes



Address Translation Overhead



- ◆ Page table is a large data structure in memory: entries!
- ◆ **Two memory accesses** for every load, store, or instr. fetch!!!
- ◆ "Cache" the address translations - use content-addressable memory (CAM, a.k.a Associative memory)

No. of entries =

If tag & Page # match

tag	data	
Virtual page #	Page Frame #	V

ECE568/Koren Part.11 .21

Copyright Koren UMass 2011

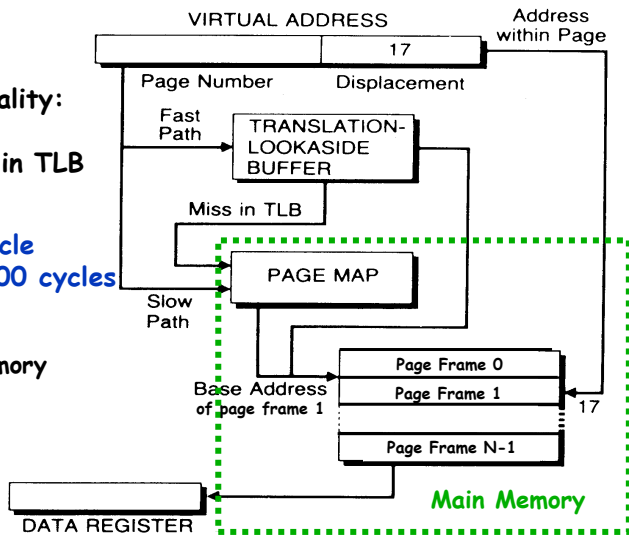
Translation Lookaside Buffer (TLB)

Rely on temporal locality:
keep only the most recently used pages in TLB

32 to 512 entries
 TLB_Hit_time = 1 cycle
 Miss_penalty = 20-100 cycles
 Hit_rate = 99%

X cycles to access memory

$$\begin{aligned} \text{Memory_Hit_time} &= 0.99(X+1) + 0.01(2X) \\ &= 1.01X + 1 \\ &= 41.4 \text{ for } X=40 \end{aligned}$$



ECE568/Koren Part.11 .22

H. Stone, "High Performance Computer Architecture," AW 1993

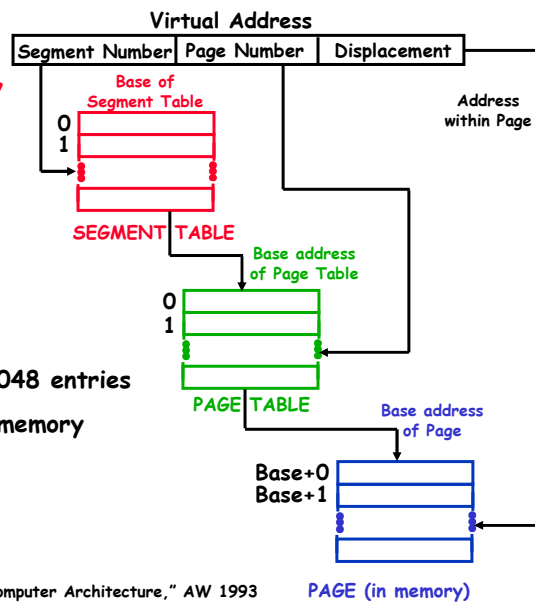
Page Table Size Problem

- ◆ **Increase page size**
 - Also: transfer of pages to/from disk more efficient
- ◆ **However:**
 - Lower Hit_rate
 - Internal fragmentation
- ◆ **Solution 1: Hashing (reduce from 2^{21} to 2^{16})**
 - Hashing function takes 21 bits and outputs 16 bits. It performs some arithmetic operation (ignoring overflows) and then a mod k operation ($k=2^{16}$). The result is an integer in $[0, k-1]$
 - The resulting table (called **inverted page table**) of 64K entries is sometimes divided into several smaller tables
 - The table entry must include the virtual page number

ECE568/Koren Part.11 .23

Copyright Koren UMass 2011

Solution 2: Segmented Memory



Paged segments

- * No table has more than 2048 entries
- * Only active PTs reside in memory
- * Penalty?

H. Stone, "High Performance Computer Architecture," AW 1993

ECE568/Koren Part.11 .24

Q3: Page Replacement Policies

- ◆ Goal: minimize number of page faults
- ◆ Effectiveness depends on:
 - (1) Page size (2) No. of available frames (3) Program behavior
- ◆ Policies - examples:
 - Optimal algorithm (upper bound)
 - Random replacement (lower bound)
 - First-In First-Out (FIFO) - longest time, e.g., 12131415
 - Least Recently Used (LRU) - was not used recently for the longest time
 - Least Frequently Used (LFU) - least referenced
 - Example: Page x 1111 0000
Page y 0000 1000
Page z 0000 0111

ECE568/Koren Part.11 .25

Copyright Koren UMass 2011

LRU vs. FIFO

page size: 4 words; 3 page frames: a,b,c; 10 pages: 0,1,2, ..., 9

Word Trace: 0,1,2,3, 4,5,6,7, 8, 16,17, 9,10,11, 12, 28,29,30, 8,9,10, 4,5, 12, 4,5

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
 Page Trace: 0 1 2 4 2 3 7 2 1 3 1

	PF	0	1	2	4	2	3	7	2	1	3	1	Hit Rate
LRU	a	0	0	0	4	4	4	7	7	7	3	3	
	b		1	1	1	1	3	3	3	1	1	1	
	c			2	2	2	2	2	2	2	2	2	
	Faults	*	*	*	*		*	*		*	*		
OPT	a	0	0	0	4	4	3	7	7	7	3	3	
	b		1	1	1	1	1	1	1	1	1	1	
	c			2	2	2	2	2	2	2	2	2	
	Faults	*	*	*	*		*	*			*		
FIFO	a	0	0	0	4	4	4	4	2	2	2	2	
	b		1	1	1	1	3	3	3	1	1	1	
	c			2	2	2	2	7	7	7	3	3	
	Faults	*	*	*	*		*	*	*	*	*		

ECE568/Koren Part.11 .26

Implementing FIFO & LRU

- ◆ **FIFO - linked list**
 - Add new page at tail, remove from head
 - Improvement: Put pages on the **free-page list**, scan it upon page fault and re-establish (**soft page fault** - no disk access)
- ◆ **LRU - most commonly used policy**
 - **Use (or Reference) bit** - set when page accessed; OS periodically sorts and moves the referenced pages to the top & resets all Use bits
 - A more accurate implementation - (software) Age Counter, +1 if not referenced, reset if referenced
 - Must provide timer interrupt to update LRU bits

The four questions (virtual storage)

- ◆ 1) Where can a page be placed? Fully associative
- ◆ 2) How is a page found? TLB + PTs
 - Page tables map virtual address to physical address
 - TLBs make virtual memory practical - temporal and spatial locality
- ◆ 3) What block is replaced on miss? Replacement policy
- ◆ 4) How are writes handled? Always Write Back
 - Minimize penalty by using a Dirty Bit

TLB Summary

Translation Lookaside Buffer or TLB

Virtual Address	Physical Address	Dirty	Ref	Valid	Access

Access bits: No_access, Read_only, Execute_only, RW&Execute