

## Virtual Memory

Some slides based on those provided by the authors of the text

**Topics**

- Motivations for VM
- Address translation
- Accelerating translation with TLBs

---

---

---

---

---

---

---

---

## Motivations for Virtual Memory

- **Use Physical DRAM as a Cache for the Disk**
  - Address space of a process can exceed physical memory size
  - Sum of address spaces of multiple processes can exceed physical memory
- **Simplify Memory Management**
  - Multiple processes resident in main memory.
    - Each process with its own address space
  - Only "active" code and data is actually in memory
    - Allocate more memory to process as needed.
- **Provide Protection**
  - One process can't interfere with another.
    - because they operate in different address spaces.
  - User process cannot access privileged information
    - different sections of address spaces have different permissions.

CompOrg Fall 2002 - Virtual Memory 2

---

---

---

---

---

---

---

---

## Motivation #1: DRAM a "Cache" for Disk

Full address space is quite large:

- 32-bit addresses: ~4,000,000,000 (4 billion) bytes
- 64-bit addresses: ~16,000,000,000,000,000,000 (16 quintillion) bytes

Disk storage is ~156X cheaper than DRAM storage

- 8 GB of DRAM: ~\$10,000
- 8 GB of disk: ~ \$64

To access large amounts of data in a cost-effective manner, the bulk of the data must be stored on disk

```

graph LR
    SRAM[4 MB: ~$400] <--> DRAM[256 MB: ~$320]
    DRAM <--> Disk[(8 GB: ~$64)]
  
```

CompOrg Fall 2002 - Virtual Memory 3

---

---

---

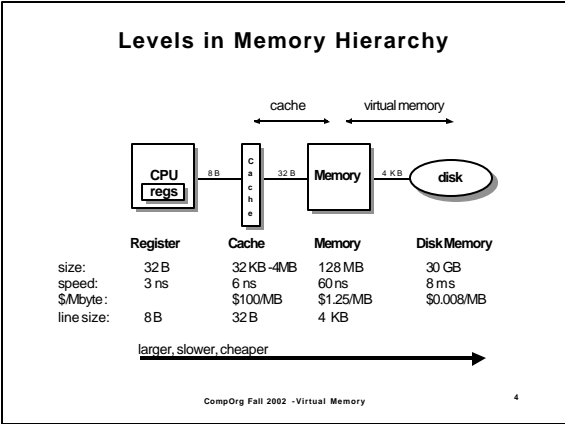
---

---

---

---

---




---

---

---

---

---

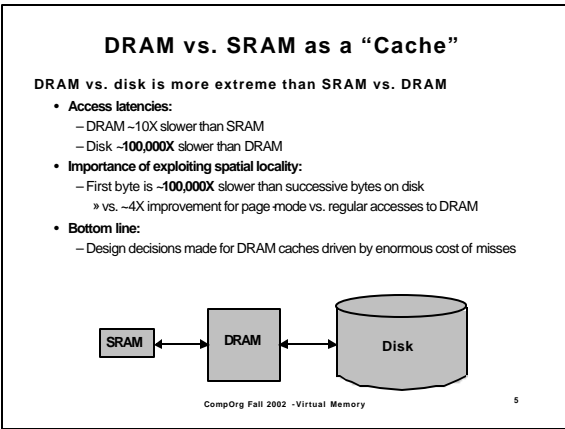
---

---

---

---

---




---

---

---

---

---

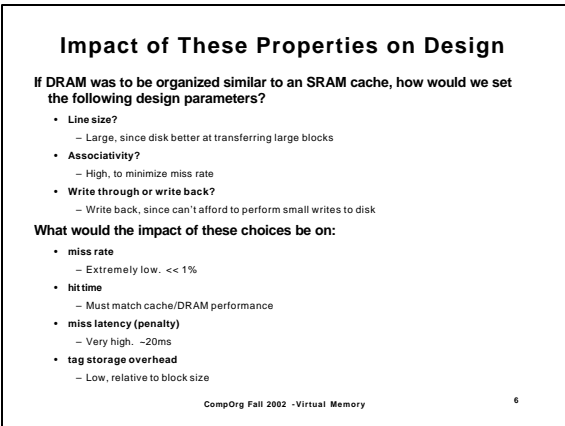
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

### Locating an Object in a "Cache"

**SRAM Cache**

- Tag stored with cache line
- Maps from cache block to memory blocks
  - From cached to uncached form
- No tag for block not in cache
- Hardware retrieves information
  - can quickly match against multiple tags

Object Name

X

= X?

	Tag	Data
0:	D	243
1:	X	17
⋮	⋮	⋮
N-1:	J	105

CompOrg Fall 2002 -Virtual Memory 7

---

---

---

---

---

---

---

---

---

---

### Locating an Object in a "Cache" (cont.)

**DRAM Cache**

- Each allocated page of virtual memory has entry in *page table*
  - Mapping from virtual pages to physical pages
  - From uncached form to cached form
- Page table entry even if page not in memory
  - Specifies disk address
- OS retrieves information

Object Name

X

→

	Location
D:	0
J:	On Disk
⋮	⋮
X:	1

→

	Data
0:	243
1:	17
⋮	⋮
N-1:	105

CompOrg Fall 2002 -Virtual Memory 8

---

---

---

---

---

---

---

---

---

---

### A System with Physical Memory Only

**Examples:**

- most Cray machines, early PCs, nearly all embedded systems, etc.

CPU

CPU

→

Physical Addresses

→

	Memory
0:	
1:	
⋮	
N-1:	

Addresses generated by the CPU point directly to bytes in physical memory

CompOrg Fall 2002 -Virtual Memory 9

---

---

---

---

---

---

---

---

---

---





### Macintosh Memory Management

**Allocation / Deallocation**

- Similar to free-list management of malloc/free

**Compaction**

- Can move any object and just update the (unique) pointer in pointer table

CompOrg Fall 2002 - Virtual Memory 16

---

---

---

---

---

---

---

---

### Mac vs. VM-Based Memory Mgmt

**Allocating, deallocating, and moving memory:**

- can be accomplished by both techniques

**Block sizes:**

- **Mac: variable-sized**  
– may be very small or very large
- **VM: fixed-size**  
– size is equal to one page (4KB on x86 Linux systems)

**Allocating contiguous chunks of memory:**

- **Mac: contiguous allocation is required**
- **VM: can map contiguous range of virtual addresses to disjoint ranges of physical addresses**

**Protection**

- Mac: "wild write" by one process can corrupt another's data

CompOrg Fall 2002 - Virtual Memory 17

---

---

---

---

---

---

---

---

### MAC OS X

**"Modern" Operating System**

- Virtual memory with protection
- Preemptive multitasking  
– Other versions of MAC OS require processes to voluntarily relinquish control

**Based on MACH OS**

- Developed at CMU in late 1980's

CompOrg Fall 2002 - Virtual Memory 18

---

---

---

---

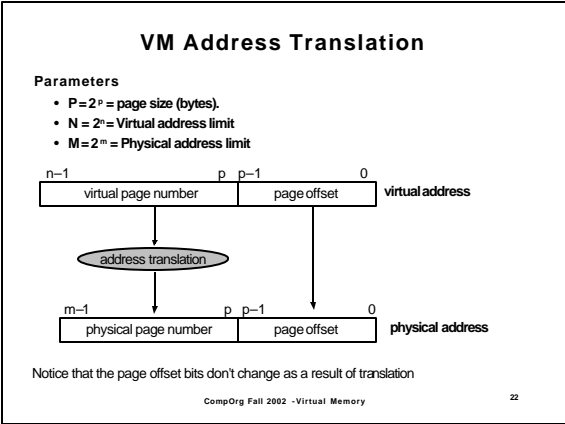
---

---

---

---






---

---

---

---

---

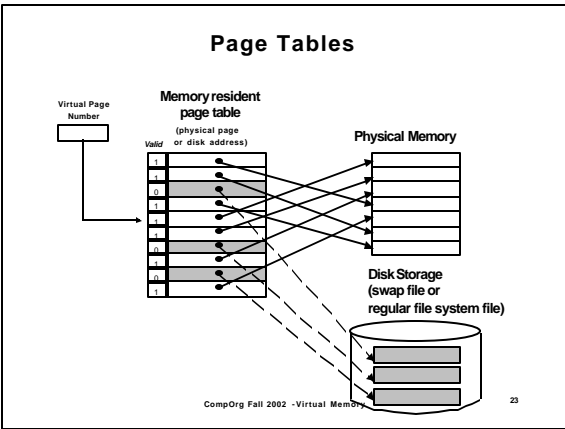
---

---

---

---

---




---

---

---

---

---

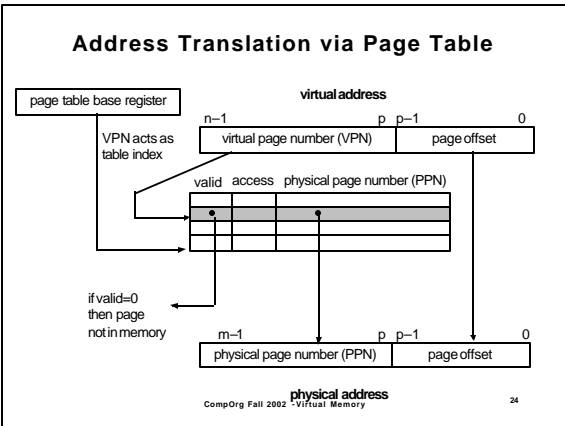
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

## Page Table Operation

### Translation

- Separate (set of) page table(s) per process
- VPN forms index into page table (points to a page table entry)

### Computing Physical Address

- Page Table Entry (PTE) provides information about page
  - if (valid bit = 1) then the page is in memory.
    - » Use physical page number (PPN) to construct address
  - if (valid bit = 0) then the page is on disk
    - » Page fault
    - » Must load page from disk into main memory before continuing

### Checking Protection

- Access rights field indicate allowable access
  - e.g., read-only, read-write, execute-only
  - typically support multiple protection modes (e.g., kernel vs. user)
- Protection violation fault if user doesn't have necessary permission

CompOrg Fall 2002 - Virtual Memory

25

---

---

---

---

---

---

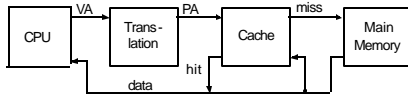
---

---

---

---

## Integrating VM and Cache



### Most Caches "Physically Addressed"

- Accessed by physical addresses
- Allows multiple processes to have blocks in cache at same time
- Allows multiple processes to share pages
- Cache doesn't need to be concerned with protection issues
  - Access rights checked as part of address translation

### Perform Address Translation Before Cache Lookup

- But this could involve a memory access itself (of the PTE)
- Of course, page table entries can also become cached

CompOrg Fall 2002 - Virtual Memory

26

---

---

---

---

---

---

---

---

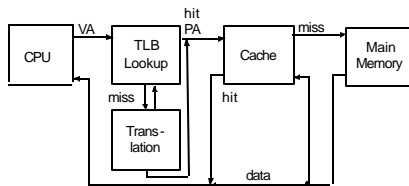
---

---

## Speeding up Translation with a TLB

### "Translation Lookaside Buffer" (TLB)

- Small hardware cache in MMU
- Maps virtual page numbers to physical page numbers
- Contains complete page table entries for small number of pages



CompOrg Fall 2002 - Virtual Memory

27

---

---

---

---

---

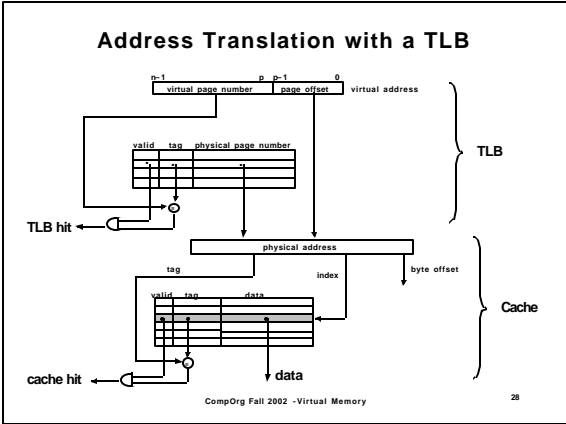
---

---

---

---

---




---

---

---

---

---

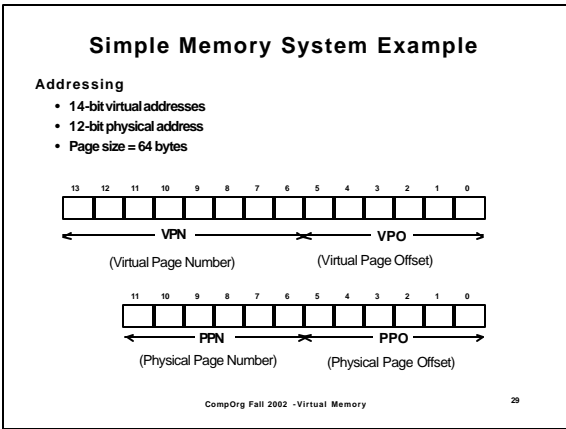
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

### Simple Memory System Page Table

- Only show first 16 entries

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	-	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	-	0
04	-	0	0C	-	0
05	16	1	0D	2D	1
06	-	0	0E	11	1
07	-	0	0F	0D	1

CompOrg Fall 2002 - Virtual Memory 30

---

---

---

---

---

---

---

---

---

---



## Multi-Level Page Tables

### Given:

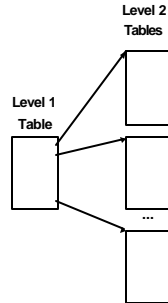
- 4KB ( $2^{12}$ ) page size
- 32-bit address space
- 4-byte PTE

### Problem:

- Would need a 4 MB page table!  
–  $2^{20} \cdot 4$  bytes

### Common solution

- multi-level page tables
- e.g., 2-level table (P6)
  - Level 1 table: 1024 entries, each of which points to a Level 2 page table.
  - Level 2 table: 1024 entries, each of which points to a page



CompOrg Fall 2002 - Virtual Memory

34

---

---

---

---

---

---

---

---

## Main Themes

### Programmer's View

- Large "flat" address space
  - Can allocate large blocks of contiguous addresses
- Processor "owns" machine
  - Has private address space
  - Unaffected by behavior of other processes

### System View

- User virtual address space created by mapping to set of pages
  - Need not be contiguous
  - Allocated dynamically
  - Enforce protection during address translation
- OS manages many processes simultaneously
  - Continually switching among processes
  - Especially when one must wait for resource
    - » E.g., disk I/O to handle page fault

CompOrg Fall 2002 - Virtual Memory

35

---

---

---

---

---

---

---

---