

**CS 61C:**  
**Great Ideas in Computer Architecture**  
*Virtual Memory*

Instructors:  
 Krste Asanovic & Vladimir Stojanovic  
<http://inst.eecs.berkeley.edu/~cs61c/>

1

**Review**

- Programmed I/O
- Polling versus Interrupts
- Asynchronous interrupts versus synchronous traps
- Precise interrupt looks like execution stopped at exactly one instruction, every instruction before finished, no instruction after started.
  - Simplify software view of interrupted state

2

**You Are Here!**

**Software**

- Parallel Requests  
Assigned to computer  
e.g., Search "Katz"
- Parallel Threads  
Assigned to core  
e.g., Lookup, Ads
- Parallel Instructions  
>1 instruction @ one time  
e.g., 5 pipelined instructions
- Parallel Data  
>1 data item @ one time  
e.g., Add of 4 pairs of words
- Hardware descriptions  
All gates @ one time
- Programming Languages

**Hardware**

Warehouse Scale Computer

Smart Phone

**Today's Lecture**

Computer

Core ... Core

Memory (Cache)

Input/Output

Instruction Unit(s)

Functional Unit(s)

Main Memory

Logic Gates

3

**Traps/Interrupts/Exceptions:**  
 altering the normal flow of control

An external or internal event that needs to be processed by another (system) program. The event is usually unexpected or rare from program's point of view.

4

**Terminology**

In CS61C (you'll see other definitions in use elsewhere):

- Interrupt – caused by an event external to current running program (e.g. key press, mouse activity)
  - Asynchronous to current program, can handle interrupt on any convenient instruction
- Exception – caused by some event during execution of one instruction of current running program (e.g., page fault, illegal instruction)
  - Synchronous, must handle exception on instruction that causes exception
- Trap – action of servicing interrupt or exception by hardware jump to "trap handler" code

5

**Precise Traps**

- Trap handler's view of machine state is that every instruction prior to the trapped one has completed, and no instruction after the trap has executed.
- Implies that handler can return from an interrupt by restoring user registers and jumping to EPC
  - Interrupt handler software doesn't need to understand the pipeline of the machine, or what program was doing!
  - More complex to handle trap caused by an exception
- Providing precise traps is tricky in a pipelined superscalar out-of-order processor!
  - But handling imprecise interrupts in software is even worse.

6

### Trap Handling in 5-Stage Pipeline

- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?

### Save Exceptions Until Commit

### Handling Traps in In-Order Pipeline

- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions *for a given instruction*
- Inject external interrupts at commit point (override others)
- If exception/interrupt at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage

### Trap Pipeline Diagram

	time	t0	t1	t2	t3	t4	t5	t6	t7	...
(I <sub>1</sub> ) 096: ADD		IF <sub>1</sub>	ID <sub>1</sub>	EX <sub>1</sub>	MA <sub>1</sub>	overflow!				
(I <sub>2</sub> ) 100: XOR			IF <sub>2</sub>	ID <sub>2</sub>	EX <sub>2</sub>					
(I <sub>3</sub> ) 104: SUB				IF <sub>3</sub>	ID <sub>3</sub>					
(I <sub>4</sub> ) 108: ADD					IF <sub>4</sub>					
(I <sub>5</sub> ) Trap Handler code						IF <sub>5</sub>	ID <sub>5</sub>	EX <sub>5</sub>	MA <sub>5</sub>	WB <sub>5</sub>

## Virtual Memory

### "Bare" 5-Stage Pipeline

- In a bare machine, the only kind of address is a physical address

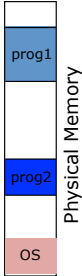
### Dynamic Address Translation

**Motivation**  
 In early machines, I/O operations were slow and each word transferred involved the CPU  
 Higher throughput if CPU and I/O of 2 or more programs were overlapped.  
 How? ⇒ multiprogramming with DMA I/O devices, interrupts

**Location-independent programs**  
 Programming and storage management ease  
 ⇒ need for a *base register*

**Protection**  
 Independent programs should not affect each other inadvertently  
 ⇒ need for a *bound register*

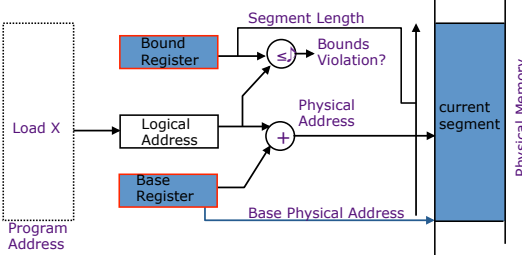
Multiprogramming drives requirement for resident *supervisor (operating system)* software to manage context switches between multiple programs



Physical Memory

13

### Simple Base and Bound Translation

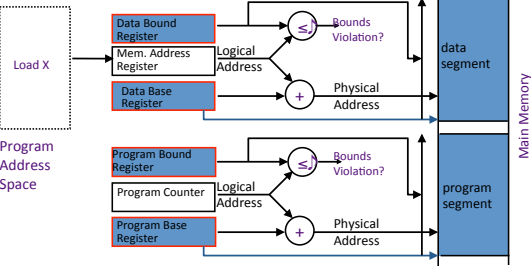


Base and bounds registers are visible/accessible only when processor is running in *supervisor mode*

14

### Separate Areas for Program and Data

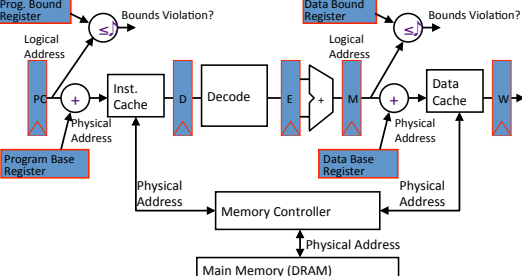
(Scheme used on all Cray vector supercomputers prior to X1, 2002)



What is an advantage of this separation?

15

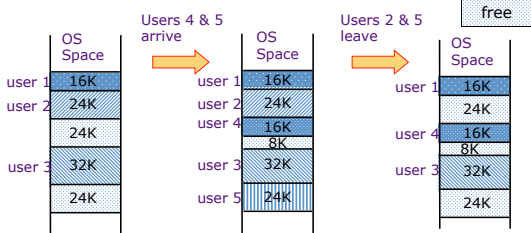
### Base and Bound Machine



[ Can fold addition of base register into (register+immediate) address calculation using a carry-save adder (sums three numbers with only a few gate delays more than adding two numbers) ]

16

### Memory Fragmentation



As users come and go, the storage is "fragmented". Therefore, at some stage programs have to be moved around to compact the storage.

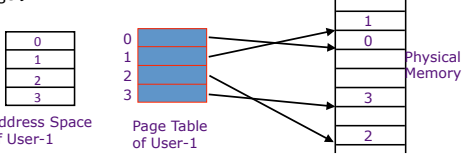
17

### Paged Memory Systems

- Processor-generated address can be split into:
 

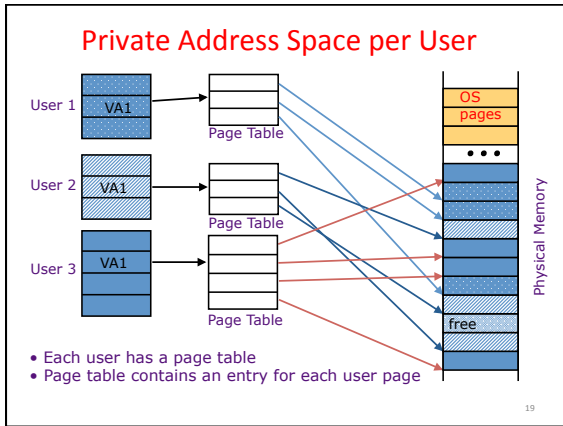
page number	offset
-------------	--------
- A page table contains the physical address of the base of each page:
 

0	1
1	0
2	3
3	2



Page tables make it possible to store the pages of a program non-contiguously.

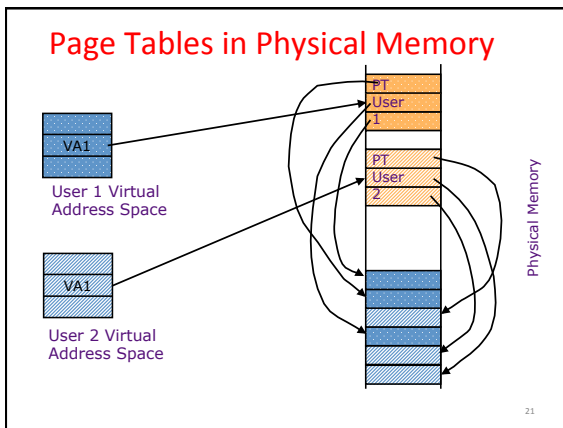
18



### Where Should Page Tables Reside?

- Space required by the page tables (PT) is proportional to the address space, number of users, ...
  - ⇒ Too large to keep in registers inside CPU
- Idea: Keep PTs in the main memory
  - needs one reference to retrieve the page base address and another to access the data word
  - ⇒ doubles the number of memory references!

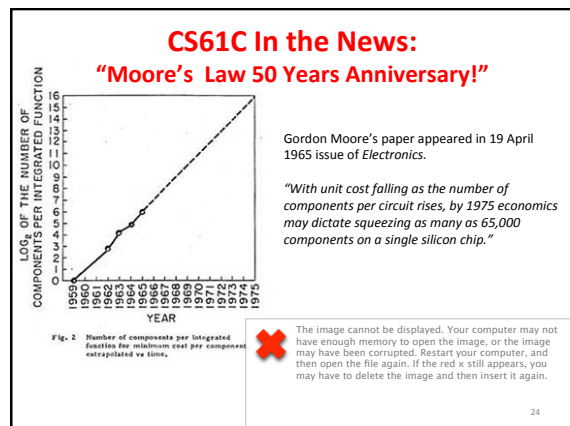
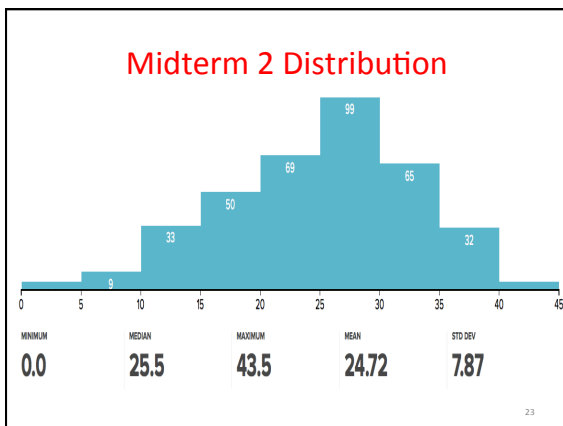
20



### Administrivia

- Midterm 2 scores up:
  - Regrade request deadline is 23:59:59 on Sunday April 26<sup>th</sup>
- Clobber Policy:
  - Final composed of MT1, MT2, post-MT2 sections
  - z-scores on MT1/MT2 sections of Final compared to MT1/MT2 grades, will replace if better
- Proj4-1 due date extended to Wed, April 29

22

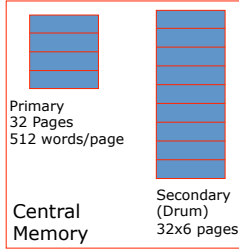


### Demand Paging in Atlas (1962)

"A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor."  
Tom Kilburn

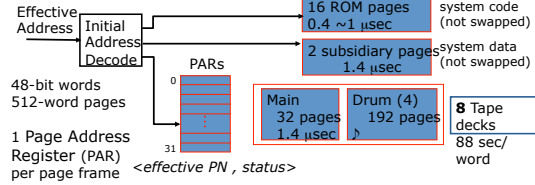
Primary memory as a cache for secondary memory

User sees 32 x 6 x 512 words of storage



25

### Hardware Organization of Atlas



Compare the effective page address against all 32 PARs  
 match => normal access  
 no match => page fault  
 save the state of the partially executed instruction

26

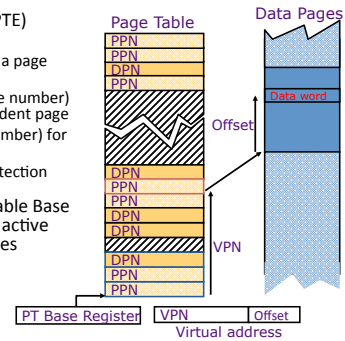
### Atlas Demand Paging Scheme

- On a page fault:
  - Input transfer into a free page is initiated
  - The Page Address Register (PAR) is updated
  - If no free page is left, a page is selected to be replaced (based on usage)
  - The replaced page is written on the drum
    - to minimize drum latency effect, the first empty page on the drum was selected
  - The page table is updated to point to the new location of the page on the drum

27

### Linear Page Table

- Page Table Entry (PTE) contains:
  - A bit to indicate if a page exists
  - PPN (physical page number) for a memory-resident page
  - DPN (disk page number) for a page on the disk
  - Status bits for protection and usage
- OS sets the Page Table Base Register whenever active user process changes



28

### Size of Linear Page Table

With 32-bit addresses, 4-KB pages & 4-byte PTEs:  
 => 2<sup>20</sup> PTEs, i.e., 4 MB page table per user  
 => 4 GB of swap needed to back up full virtual address space

Larger pages?

- Internal fragmentation (Not all memory in page is used)
- Larger page fault penalty (more time to read from disk)

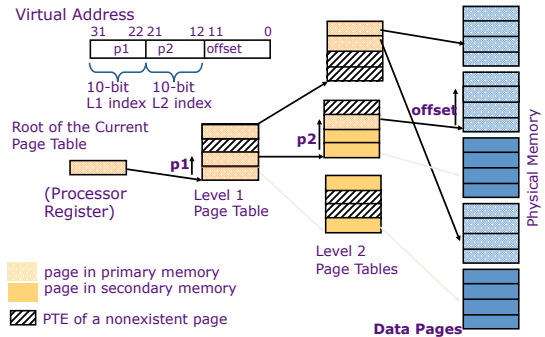
What about 64-bit virtual address space???

- Even 1MB pages would require 2<sup>44</sup> 8-byte PTEs (35 TB!)

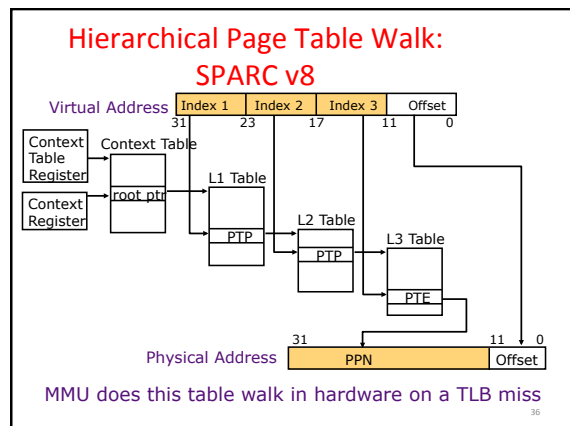
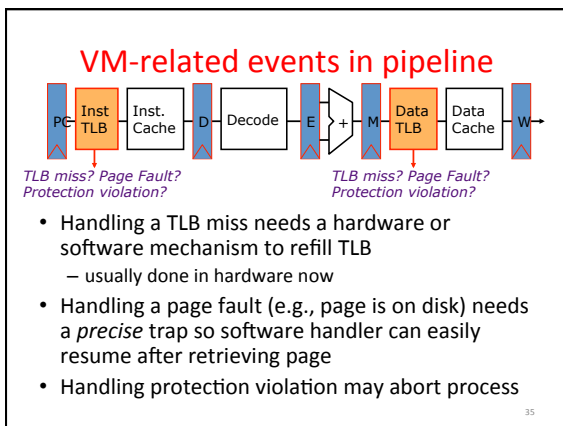
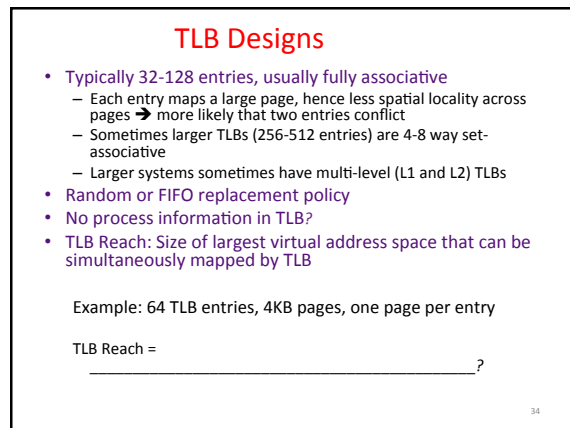
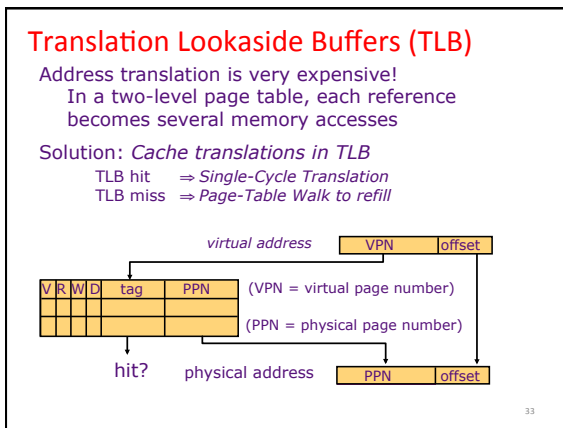
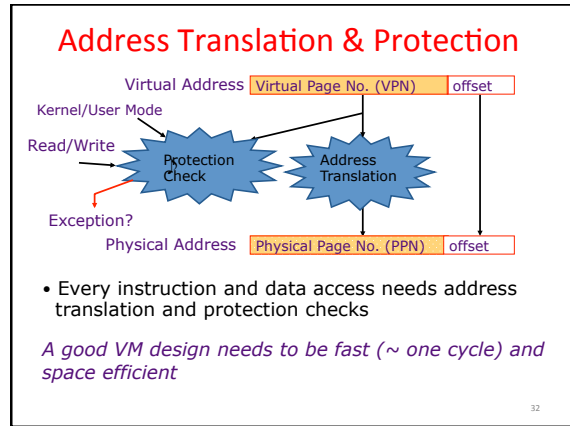
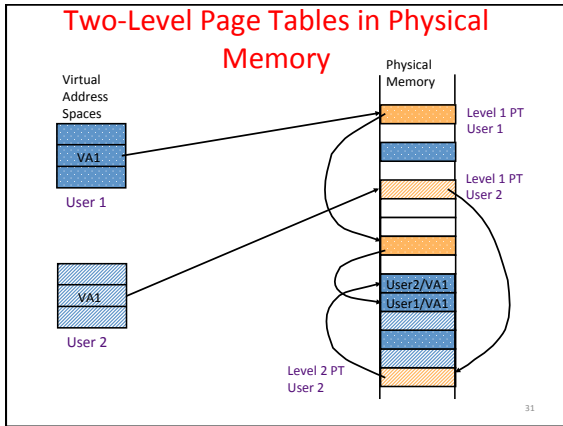
What is the "saving grace" ?

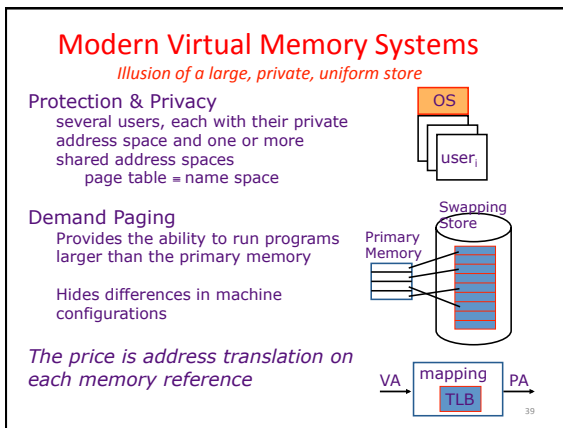
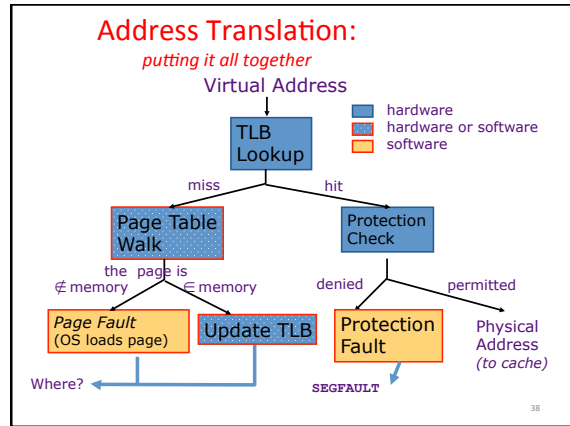
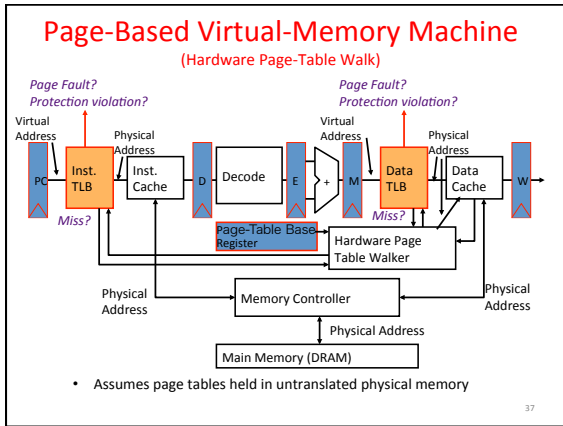
29

### Hierarchical Page Table



30





- ### VM features track historical uses:
- **Bare machine, only physical addresses**
    - One program owned entire machine
  - **Batch-style multiprogramming**
    - Several programs sharing CPU while waiting for I/O
    - Base & bound: translation and protection between programs (not virtual memory)
    - Problem with external fragmentation (holes in memory), needed occasional memory defragmentation as new jobs arrived
  - **Time sharing**
    - More interactive programs, waiting for user. Also, more jobs/second.
    - Motivated move to fixed-size page translation and protection, no external fragmentation (but now internal fragmentation, wasted bytes in page)
    - Motivated adoption of virtual memory to allow more jobs to share limited physical memory resources while holding working set in memory
  - **Virtual Machine Monitors**
    - Run multiple operating systems on one machine
    - Idea from 1970s IBM mainframes, now common on laptops
      - e.g., run Windows on top of Mac OS X
    - Hardware support for two levels of translation/protection
      - Guest OS virtual -> Guest OS physical -> Host machine physical
    - Also basis of Cloud Computing
      - Virtual machine instances for Project 4
- 40