CS 61C:
Great Ideas in Computer Architecture
VM²: Virtual Memory and Machines

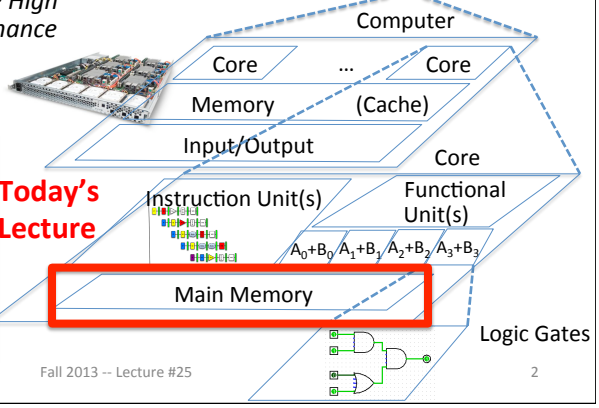
Instructor:
Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa13

Part I: You Are Here!

Software *Hardware*

- 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

Harness Parallelism & Achieve High Performance



Today's Lecture

Agenda

- Virtual Memory
- Administrivia
- Virtual Machines
- And, in Conclusion …

Agenda

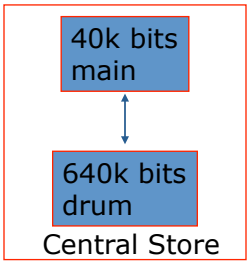
- Virtual Memory
- Administrivia
- Virtual Machines
- And, in Conclusion …

Not Enough Memory

- A problems from the 1960s
- There were many applications whose data could not fit in the main memory, e.g., payroll
  - *Paged memory system reduced fragmentation but still required the whole program to be resident in the main memory*

Manual Overlays

- Assume an instruction can address all the storage on the drum
- *Method 1:* programmer keeps track of addresses in the main memory and initiates an I/O transfer when required
  - *Difficult, error-prone!*
- *Method 2:* automatic initiation of I/O transfers by software address translation
  - *Brooker's interpretive coding, 1960*
  - *Inefficient!*



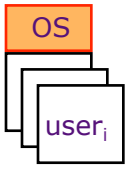
Ferranti Mercury 1956

Not just an ancient black art, e.g., IBM Cell microprocessor used in Playstation-3 has explicitly managed local store!

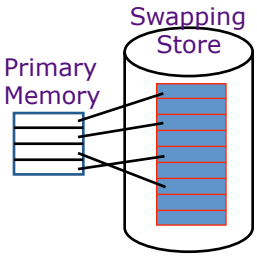
Modern Virtual Memory Systems

Illusion of a large, private, uniform store

Protection & Privacy
- several users, each with their private address space and one or more shared address spaces
  - page table ≡ name space

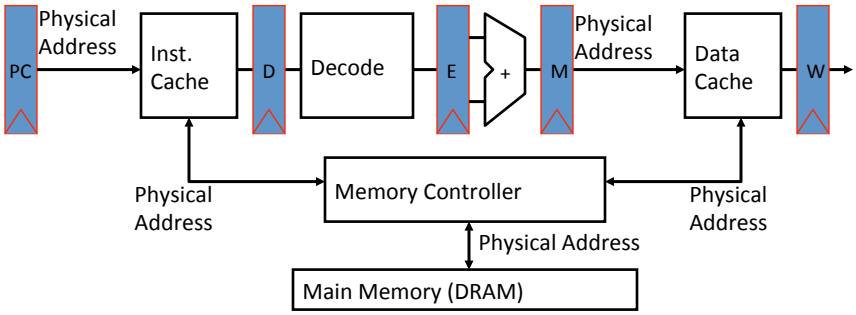


Demand Paging
- Provides the ability to run programs larger than the primary memory
- Hides differences in machine configurations



The price is address translation on each memory reference

"Bare" 5-Stage Pipeline



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

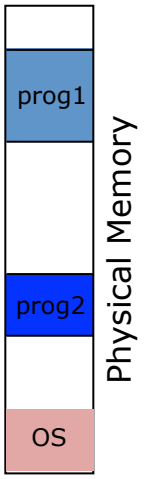
Dynamic Address Translation

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*

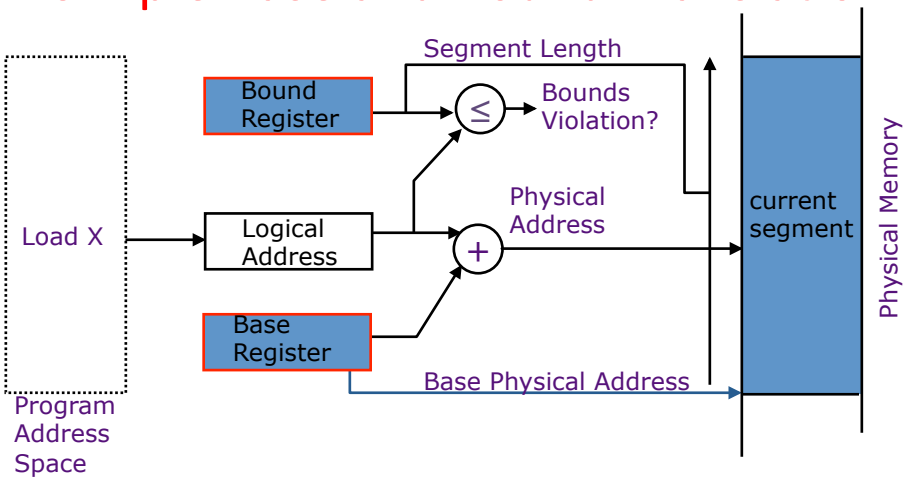
Location-independent programs
- Programming and storage management ease
- ⇒ need for a *base register*

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

Multiprogramming drives requirement for resident *supervisor* software to manage context switches between multiple programs



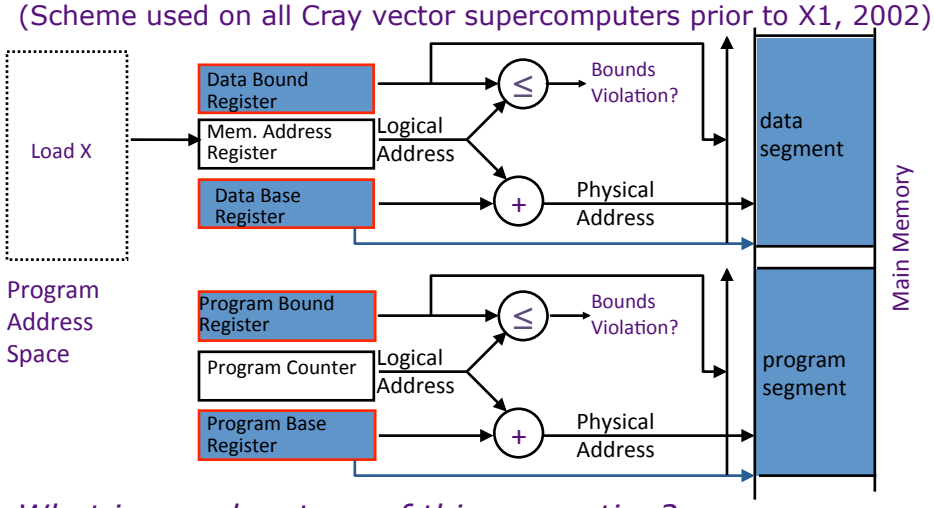
Simple Base and Bound Translation



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

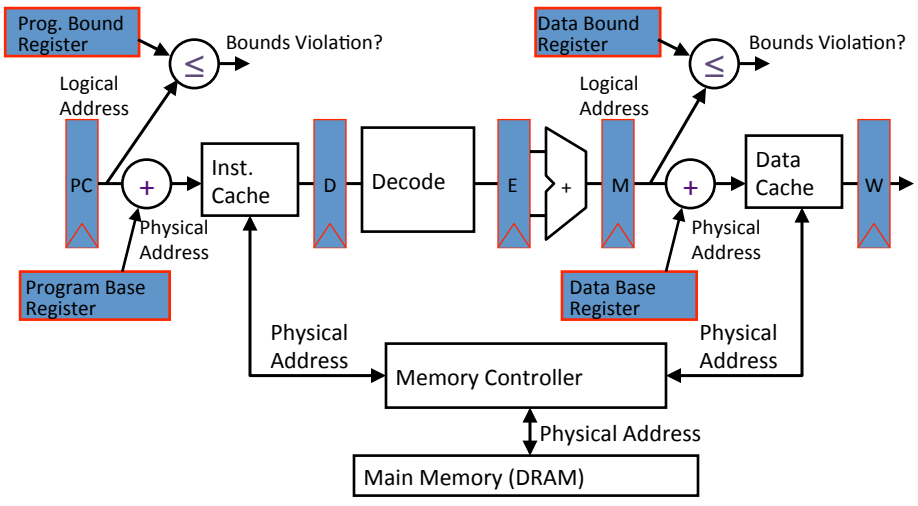
Separate Areas for Program and Data

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



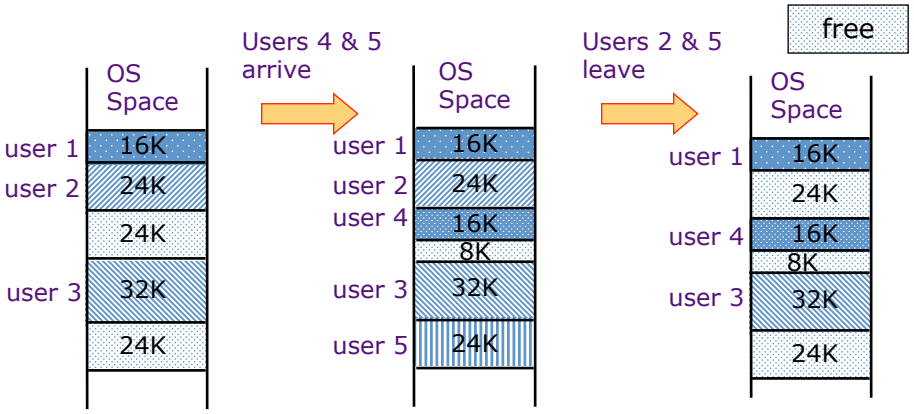
What is an advantage of this separation?

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)]

Memory Fragmentation



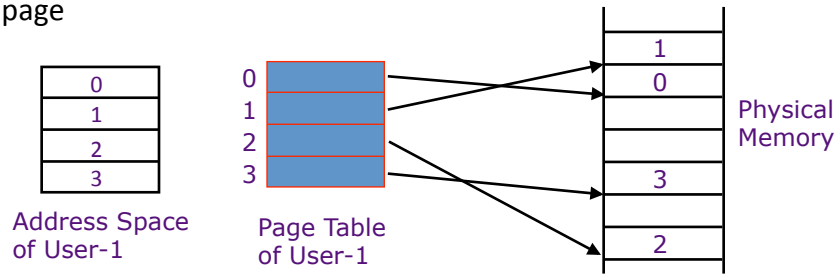
As users come and go, the storage is "fragmented". Therefore, at some stage programs have to be moved around to compact the storage.

Paged Memory Systems

- Processor-generated address can be split into:

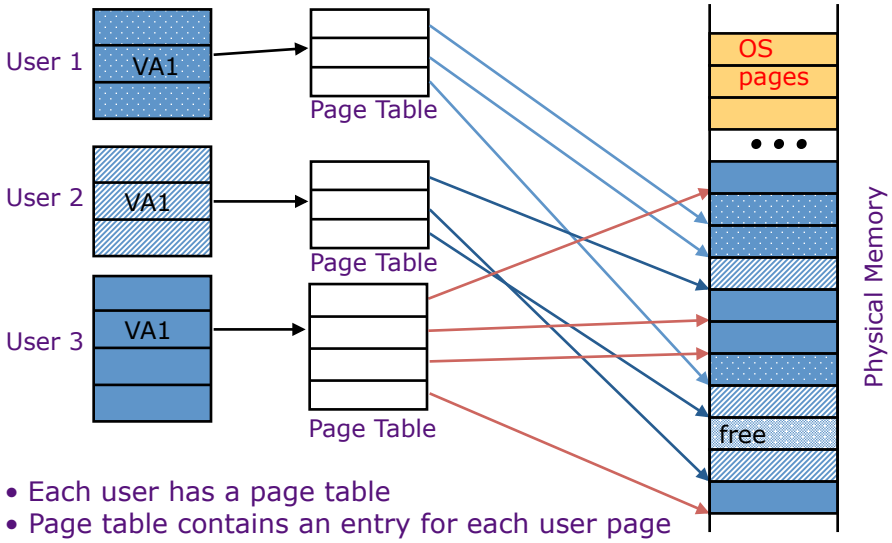
page number	offset

- A page table contains the physical address of the base of each page



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

Private Address Space per User



- Each user has a page table
- Page table contains an entry for each user page

Where Should Page Tables Reside?

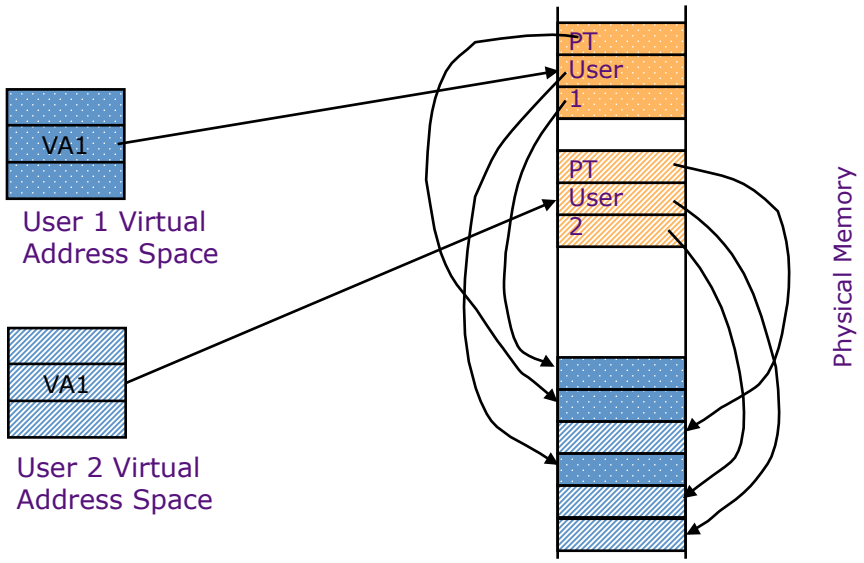
- Space required by the page tables (PT) is proportional to the address space, number of users, ...

  ⇒ *Too large to keep in registers*

- 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!*

Page Tables in Physical Memory

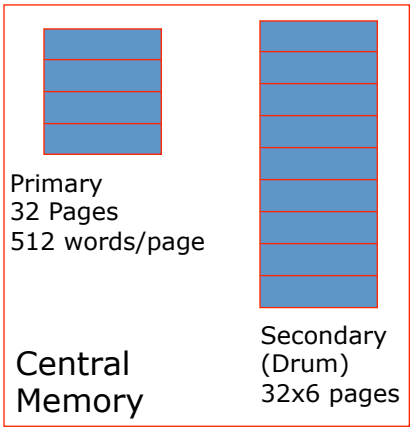


Demand Paging in Atlas (1962)

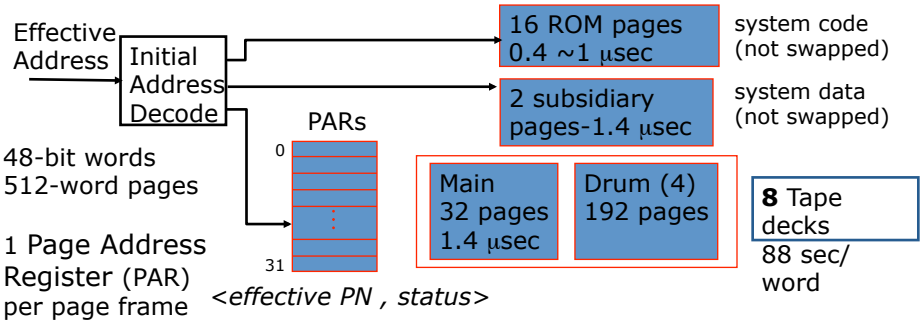
"A page from secondary storage is brought into the primary storage whenever it is (implicitly) demanded by the processor."
Tom Kilburn

Primary memory as a *cache* for secondary memory

User sees 32 x 6 x 512 words of storage



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

Atlas Demand Paging Scheme

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

Agenda

- Virtual Memory
- Administrivia
- Virtual Machines
- And, in Conclusion …

Administrivia

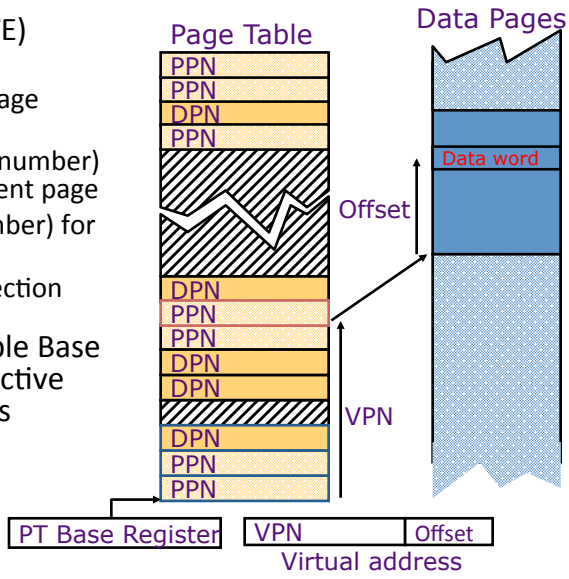
- Verify all HW, Lab, Midterm, and Project Grades by Friday, 12/6!
- Final Exam
  - Friday, December 20, 8:00-11:00 RSF Fieldhouse!
  - Short answer, fill in the blank, multiple choice, mix and match: 100 points/minutes
  - Comprehensive, but concentrated on material since midterm examination
  - Closed book/note, open crib sheet as before, MIPS Green Card provided
  - Review Session, Monday, 12/9, 3-6 PM, Room TBD
  - *Special consideration students, please contact*

Administrivia

- Topics for Final Exam
  - Cache Aware Programming/Cache Blocking
  - Data Level Parallelism: Intel SIMD SSE instructions and programming
  - Thread Parallelism: Cache Coherency + Synchronization concepts
  - OpenMP Programming
  - Hardware: Transistors to Gates
  - Hardware: Truth Tables to Boolean Algebra
  - Hardware: Synchronous System Timing and Timing Diagrams
  - Finite State Machines: State Diagrams and Implementation
  - CPU Design: Data Path Design (ALUs, Shifters, Register Files, Muxes)
  - CPU Design: Controller Design (FSMs for processor implementation)
  - Instruction Level Parallelism/Instruction Pipelining
  - Set Associative Caches
  - Dependability: ECC + RAID
  - Virtual Memory
  - X-semester issues: Great Ideas in Computer Architecture

Linear Page Table

- Page Table Entry (PTE) contains:
  - 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



Size of Linear Page Table

With 32-bit addresses, 4-KB pages & 4-byte PTEs:

⇒ 2^{20} PTEs, i.e, 4 MB page table per user

⇒ 4 GB of swap needed to back up full virtual address space

Larger pages?

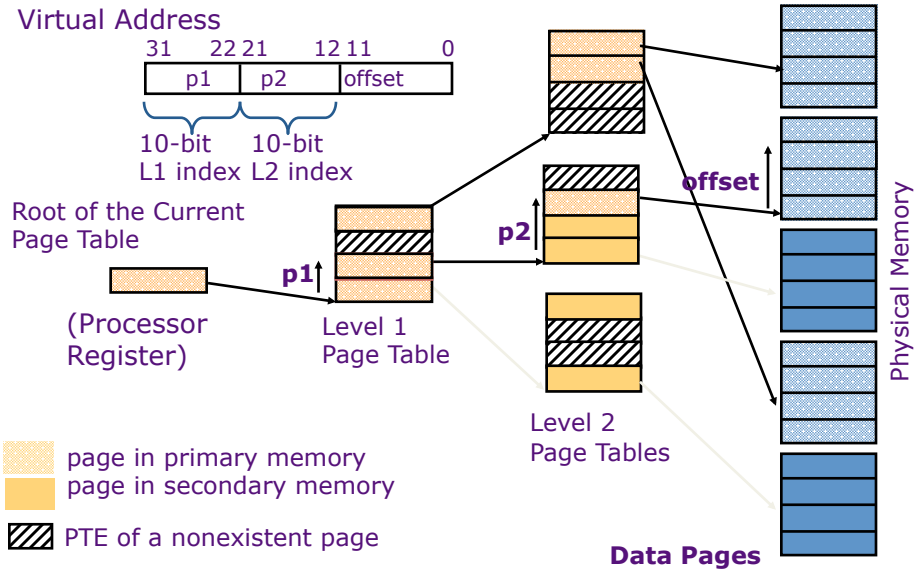
- Internal fragmentation (Not all memory in page is used)
- Larger page fault penalty (more time to read from disk)

What about 64-bit virtual address space???

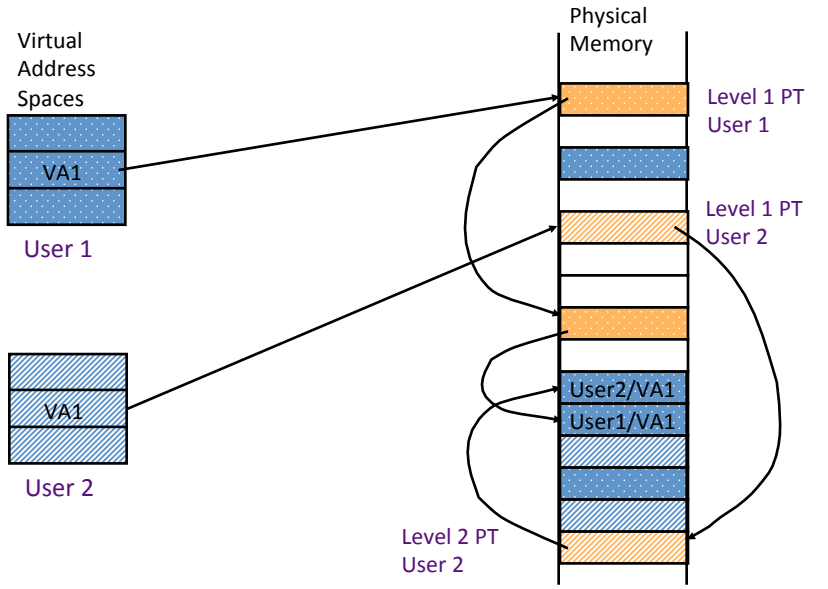
- Even 1MB pages would require 2^{44} 8-byte PTEs (35 TB!)

What is the "saving grace" ?

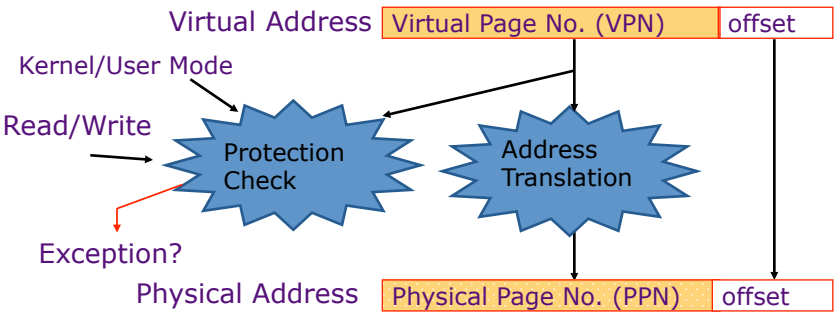
Hierarchical Page Table



Two-Level Page Tables in Physical Memory



Address Translation & Protection



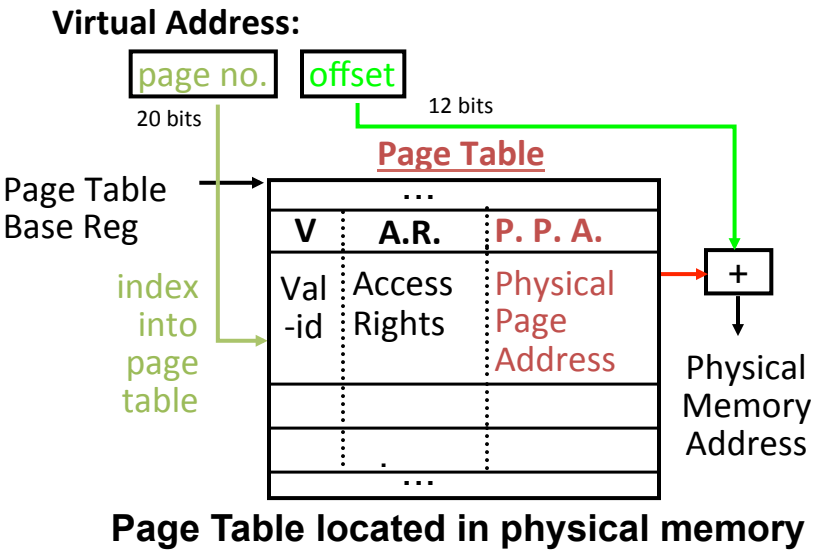
- Every instruction and data access needs address translation and protection checks

A good VM design needs to be fast (~ one cycle) and space efficient

Protection via Page Table

- Access Rights checked on every access to see if allowed
  - Read: can read, but not write page
  - Read/Write: read or write data on page
  - Execute: Can fetch instructions from page
- Valid = Valid page table entry
  - Invalid means it's on the disk, not in physical memory

More Depth on Page Tables

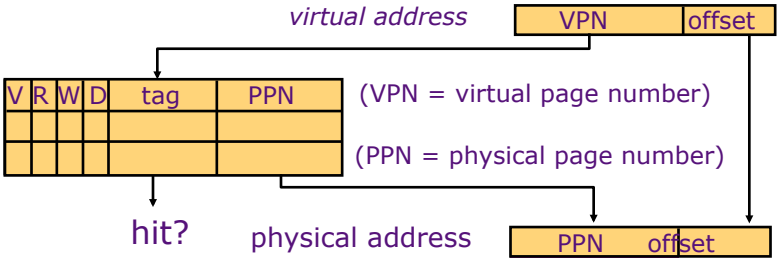


Translation Lookaside Buffers (TLB)

Address translation is very expensive!
In a two-level page table, each reference becomes several memory accesses

Solution: *Cache translations in TLB*

TLB hit ⇒ *Single-Cycle Translation*
TLB miss ⇒ *Page-Table Walk to refill*



TLB Designs

- Typically 32-128 entries, usually fully associative
  - Each entry maps a large page, hence less spatial locality across pages ➔ more likely that two entries conflict
  - Sometimes larger TLBs (256-512 entries) are 4-8 way set-associative
  - Larger systems sometimes have multi-level (L1 and L2) TLBs
- Random or FIFO replacement policy
- No process information in TLB?
- TLB Reach: Size of largest virtual address space that can be simultaneously mapped by TLB

Example: 64 TLB entries, 4KB pages, one page per entry

TLB Reach = _____?

Handling a TLB Miss

Software (MIPS, Alpha)

TLB miss causes an exception and the operating system walks the page tables and reloads TLB. *A privileged "untranslated" addressing mode used for walk*

Hardware (SPARC v8, x86, PowerPC, RISC-V)

A memory management unit (MMU) walks the page tables and reloads the TLB

If a missing (data or PT) page is encountered during the TLB reloading, MMU gives up and signals a Page-Fault exception for the original instruction

Flashcard Quiz: Which statement is false?

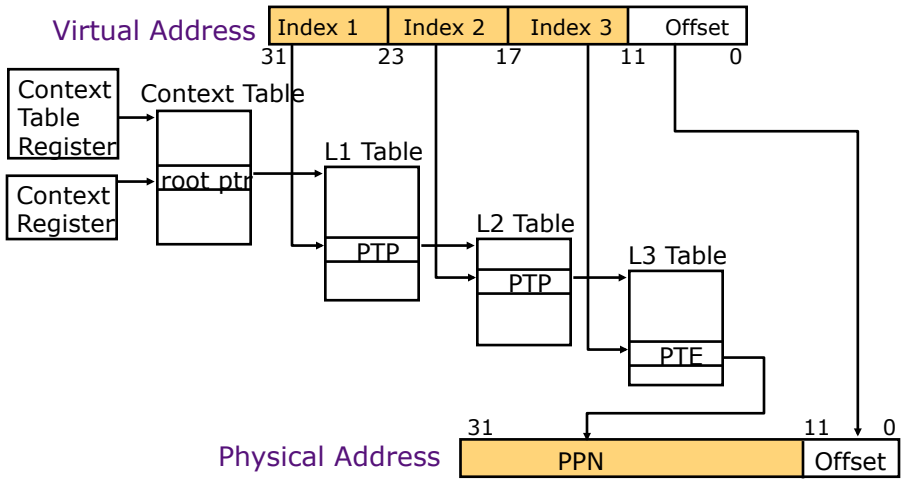
TLB miss is much faster than page fault

TLB exploits spatial locality

TLB hardware grows with larger page size

TLB exploits temporal locality

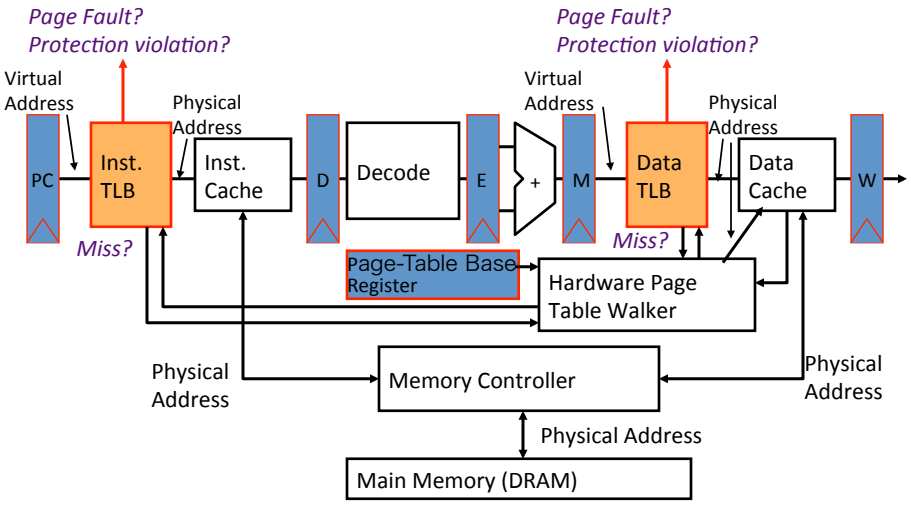
Hierarchical Page Table Walk: SPARC v8



MMU does this table walk in hardware on a TLB miss

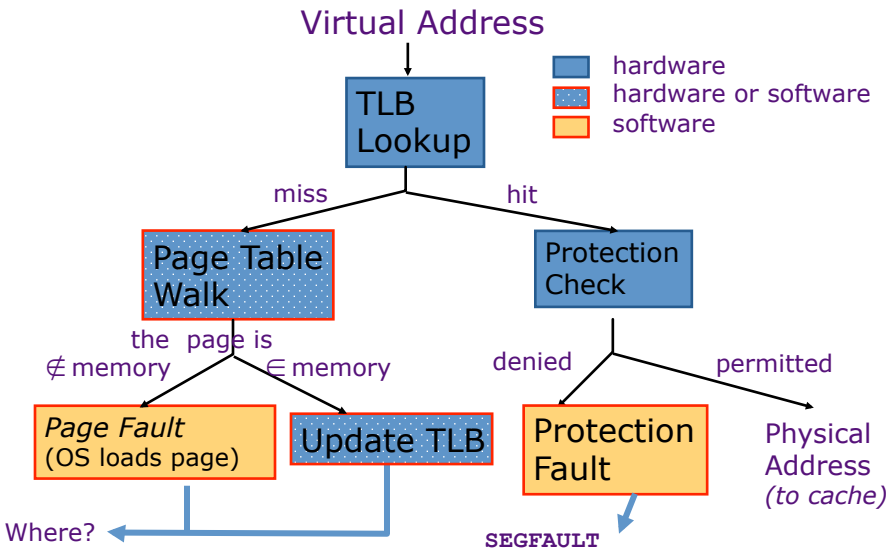
Page-Based Virtual-Memory Machine

(Hardware Page-Table Walk)

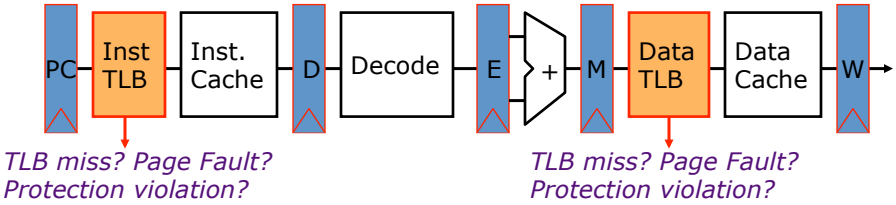


- Assumes page tables held in untranslated physical memory

Address Translation: Putting it All Together

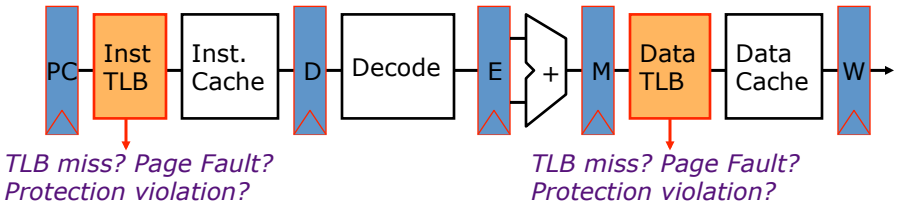


Handling VM-Related Traps



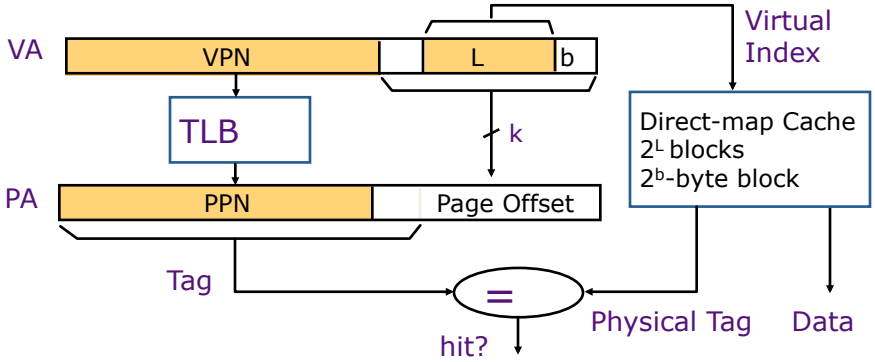
- Handling a TLB miss needs a hardware or software mechanism to refill TLB
- Handling a page fault (e.g., page is on disk) needs a *restartable* trap so software handler can resume after retrieving page
  - Precise exceptions are easy to restart
  - Can be imprecise but restartable, but this complicates OS software
- Handling protection violation may abort process

Address Translation in CPU Pipeline



- Need to cope with additional latency of TLB:
  - slow down the clock?
  - pipeline the TLB and cache access?
  - virtual address caches (see CS152)
  - parallel TLB/cache access

Concurrent Access to TLB & Cache (Virtual Index/Physical Tag)



Index L is available without consulting the TLB

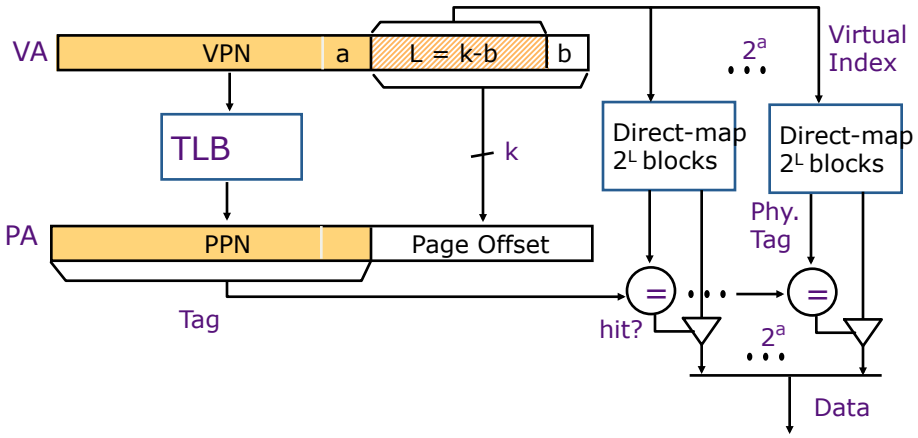
⇒ *cache and TLB accesses can begin simultaneously!*

Tag comparison is made after both accesses are completed

Cases: $L + b = k$, $L + b < k$, $L + b > k$

Virtual-Index Physical-Tag Caches:

Associative Organization



After the PPN is known, 2^a physical tags are compared

How does this scheme scale to larger caches?

Agenda

- Virtual Memory
- Administrivia
- Virtual Machines
- And, in Conclusion …

Safely Sharing a Machine

- Amazon Web Services allows independent tasks run on same computer
  - Can sell each "instance"
- Can a "small" operating system (~10,000 LOC) simulate the hardware of some machine, so that
  - Another operating system can run in that simulated hardware?
  - More than one instance of that operating system run on the same hardware at the same time?
  - More than one *different* operating system can share the same hardware at the same time?
  - And none can access each others' data?
- Answer: *Yes*

Solution – *Virtual Machine*

- A virtual machine provides interface *identical* to underlying bare hardware
  - I.e., all devices, interrupts, memory, etc.
- Examples
  - IBM VM/370 (1970s technology!)
  - VMWare (founded by Mendel & Diane Rosenblum)
  - Xen (used by AWS)
  - Microsoft Virtual PC
- Called "System Virtual Machines" vs. language interpreters (e.g., Java Virtual Machine)
- Virtualization has some performance impact
  - Feasible with modern high-performance computers

Virtual Machines

- *Host Operating System:*
  - OS actually running on the hardware
  - Together with *virtualization layer*, it simulates environment for …
- *Guest Operating System:*
  - OS running in the simulated environment
  - Runs identical as if on native hardware (except performance)
  - Cannot change access of real system resources
  - Guest OS code runs in native machine ISA
- The resources of the physical computer are shared to create the virtual machines
- Processor scheduling by OS can create the appearance that each user has own processor

Virtual Machine Monitor (a.k.a. Hypervisor)

- Maps virtual resources to physical resources
  - Memory, I/O devices, processors
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest OS
- Host OS must intercept attempts by Guest OS to access real I/O devices, allocate resources

Why Virtual Machines Popular (Again)?

- Increased importance of isolation and security
- Failures in security and reliability of modern OS's
- Sharing of single computer between many unrelated users
  - E.g., Cloud computing
- Dramatic increase in performance of processors makes VM overhead acceptable

5 Reasons Amazon Web Services uses Virtual Machines

- (Uses x86 ISA, Linux Host OS, and Xen VMM)

1. Allow AWS protect users from each other
2. Simplified SW distribution within WSC
   - Customers install image, AWS distributes to all instances
3. Can reliably "kill" a VM => control resource usage
4. VMs hide identity of HW => can keep selling old HW AND can introduce new more efficient HW
   - VM Performance not need be integer multiple of real HW
5. VM limiting rate of processing, network, and disk space => AWS offers many price points

Peer Instruction: True or False

Which statements is *True* about Virtual Machines?

I. Multiple Virtual Machines can run on one computer

II. Multiple Virtual Machine Monitors can run on one computer

III. The Guest OS must be the same as the Host OS

A) I only

B) II only

C) III only

D) I and II

Virtual Machine/Memory Instruction Set Support

- 2 modes in hardware: User and System modes
  - Some instruction only run in System mode
- Page Table Base Register (PTBR): Page Table addr
- Privileged instructions only available in system mode
  - Trap to system (and VMM) if executed in user mode
- All physical resources only accessible using privileged instructions
  - Including interrupt controls, I/O registers
- Renaissance of virtualization support in ISAs
  - Current ISAs (e.g., x86) adapting, following IBM's path

Example: Timer Virtualization

- In native machine (no VMM), on timer interrupt
  - OS suspends current process, handles interrupt, selects and resumes next process
- With Virtual Machine Monitor
  - VMM suspends current VM, handles interrupt, selects and resumes next VM
- If a VM requires timer interrupts
  - VMM emulates a virtual timer
  - Emulates interrupt for VM when physical timer interrupt occurs

Virtual Machine Monitor (a.k.a. Hypervisor)

- Maps virtual resources to physical resources
  - Memory, I/O devices, processors
- Guest OS code runs on native machine ISA in user mode
  - Traps to VMM on privileged instructions and access to protected resources
- Guest OS may be different from host OS
- VMM handles real I/O devices
  - Emulates generic virtual I/O devices for guest

Performance Impact of Virtual Machines?

- No impact on computation bound program, since they spend ≈0 time in OS
  - E.g., matrix multiply
- Big impact on I/O-intensive programs, since spend lots of time in OS and execute many systems calls and many privileged instructions
  - Although if I/O-bound => spend most time waiting for I/O device, then can hide VM overhead

Virtual Machines and Cores

- Host OS can also limit amount of *time* a virtual machine uses a processor (core)
- Hence, at cost of swapping registers, it can run multiple virtual machines on a *single* core
- AWS cheapest VM was originally 2 VMs per core
- Now, with faster processors, can install more VMs per core and deliver same performance or have multiple speeds of cores

Agenda

- Virtual Memory
- Administrivia
- Virtual Machines
- And, in Conclusion …

And in Conclusion, …

- Virtual Memory, Paging for Isolation and Protection, help Virtual Machines share memory
  - Can think of as another level of memory hierarchy, but not really used like caches are today
  - Not really routinely paging to disk today
- Virtual Machines as even greater level of protection to allow greater level of sharing
  - Enables fine control, allocation, software distribution, multiple price points for Cloud Computing