COS 318: Operating Systems

Virtual Memory: Address Translation

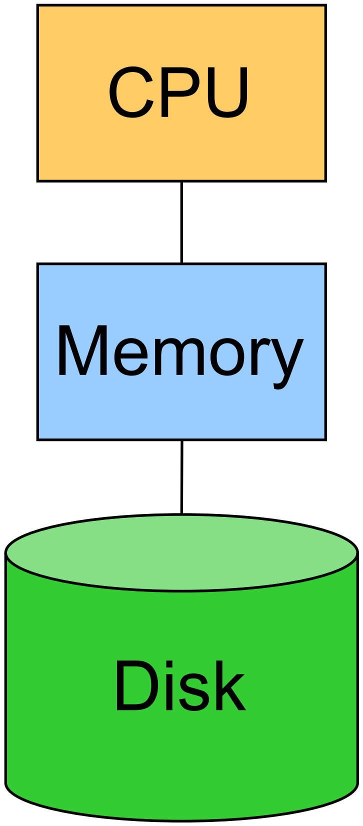
(http://www.cs.princeton.edu/courses/cos318/)

Today's Topics

- Virtual Memory
  - Virtualization
  - Protection
- Address Translation
  - Base and bound
  - Segmentation
  - Paging
  - Translation look-ahead buffer

The Big Picture

- DRAM is fast, but relatively expensive
- Disk is inexpensive, but slow
  - 100X less expensive
  - 100,000X longer latency
  - 1000X less bandwidth
- Our goals
  - Run programs as efficiently as possible
  - Make the system as safe as possible

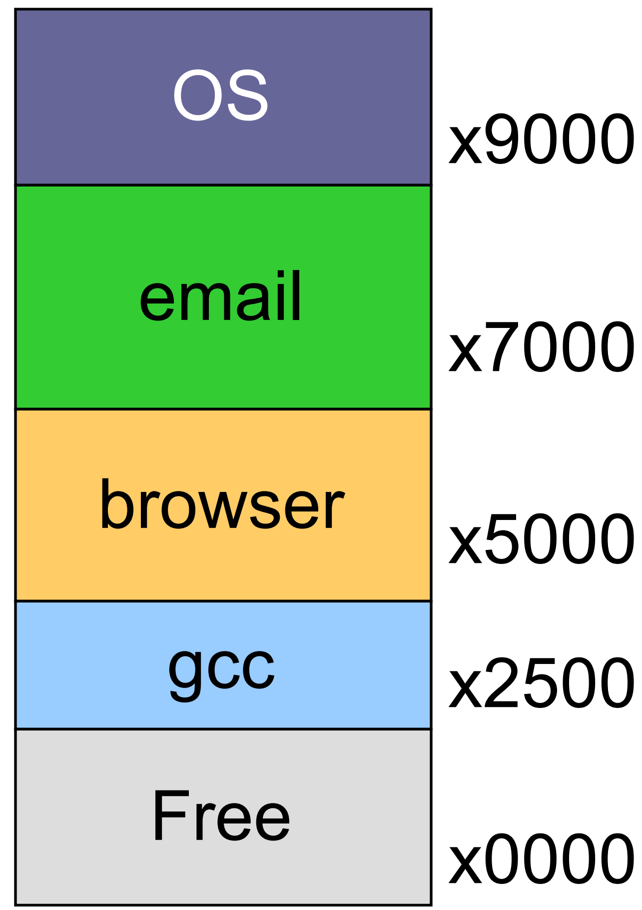


Issues

- Many processes
  - The more processes a system can handle, the better
- Address space size
  - Many small processes whose total size may exceed memory
  - Even one process may exceed the physical memory size
- Protection
  - A user process should not crash the system
  - A user process should not do bad things to other processes

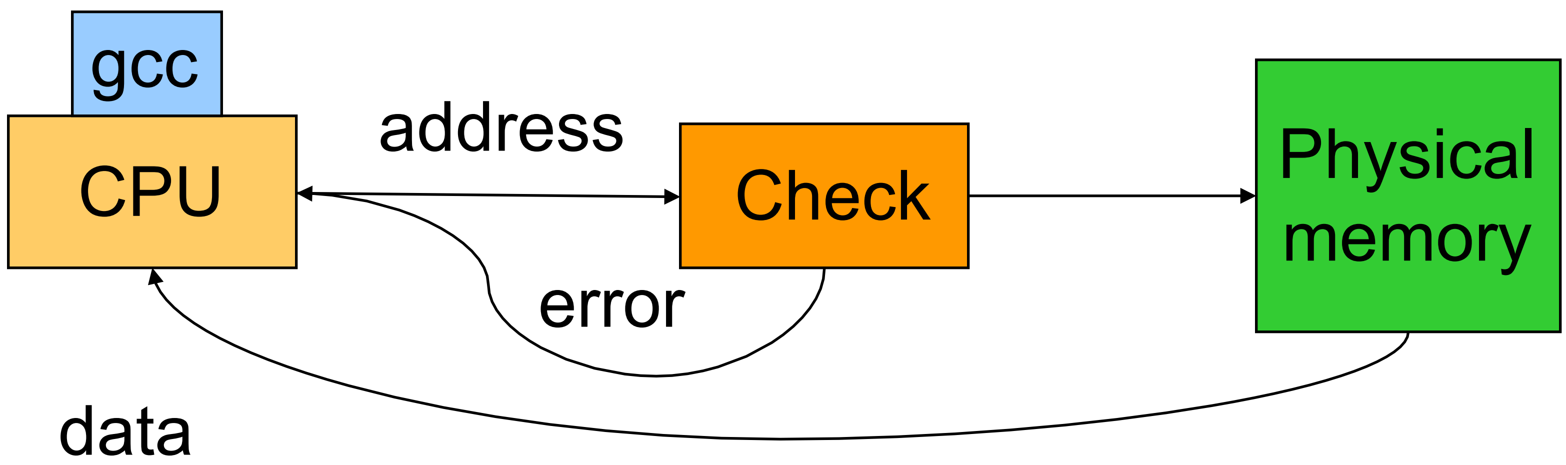
Consider A Simple System

- Only physical memory
  - Applications use physical memory directly
- Run three processes
  - Email, browser, gcc
- What if
  - gcc has an address error?
  - browser writes at x7050?
  - email needs to expand?
  - browser needs more memory than is on the machine?



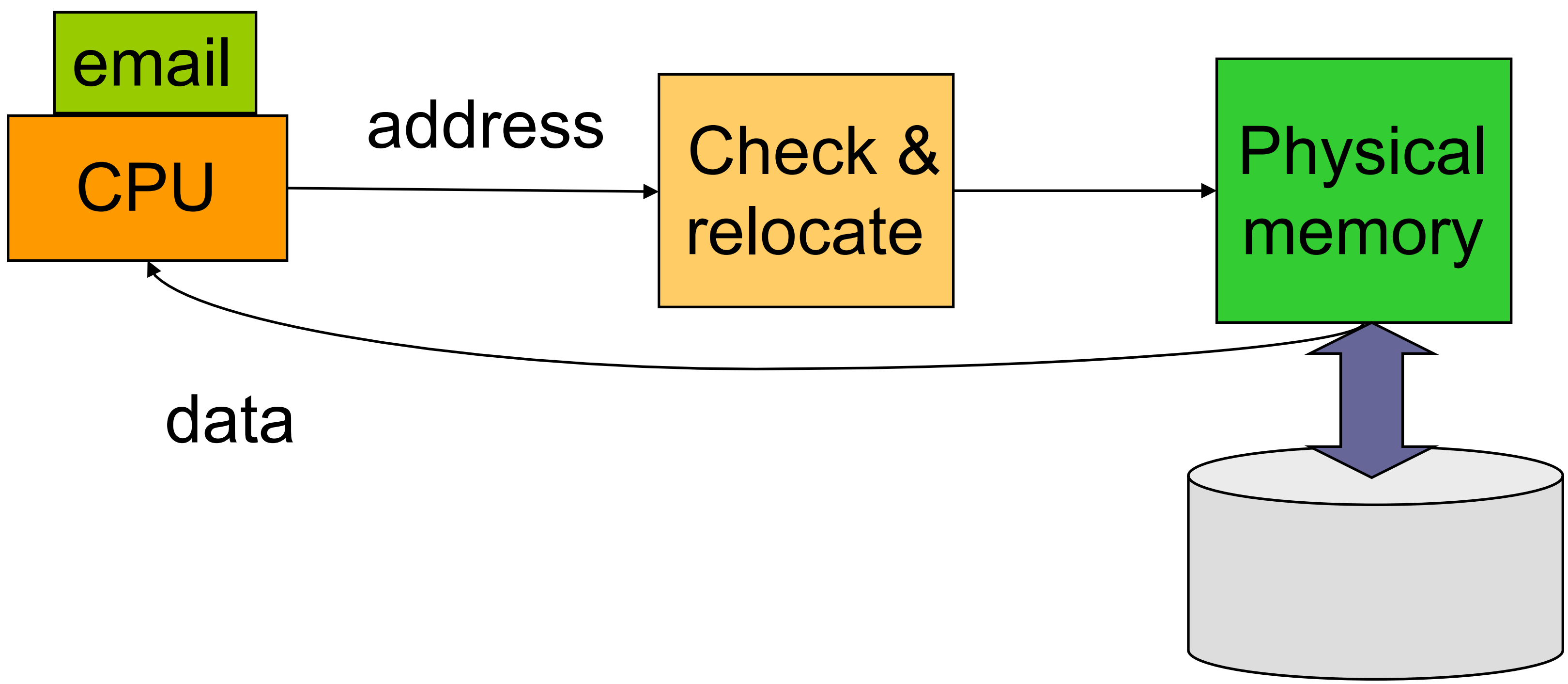
Handling Protection

- Errors in one process should not affect others
- For each process, check each load and store instruction to allow only legal memory references



Handling Finiteness: Relocation

- A process should be able to run regardless of where its data are physically placed or the physical memory size
- Give each process a large, static “fake” address space that is large and contiguous and entirely its own
- As a process runs, relocate or map each load and store to addresses in actual physical memory



Virtual Memory

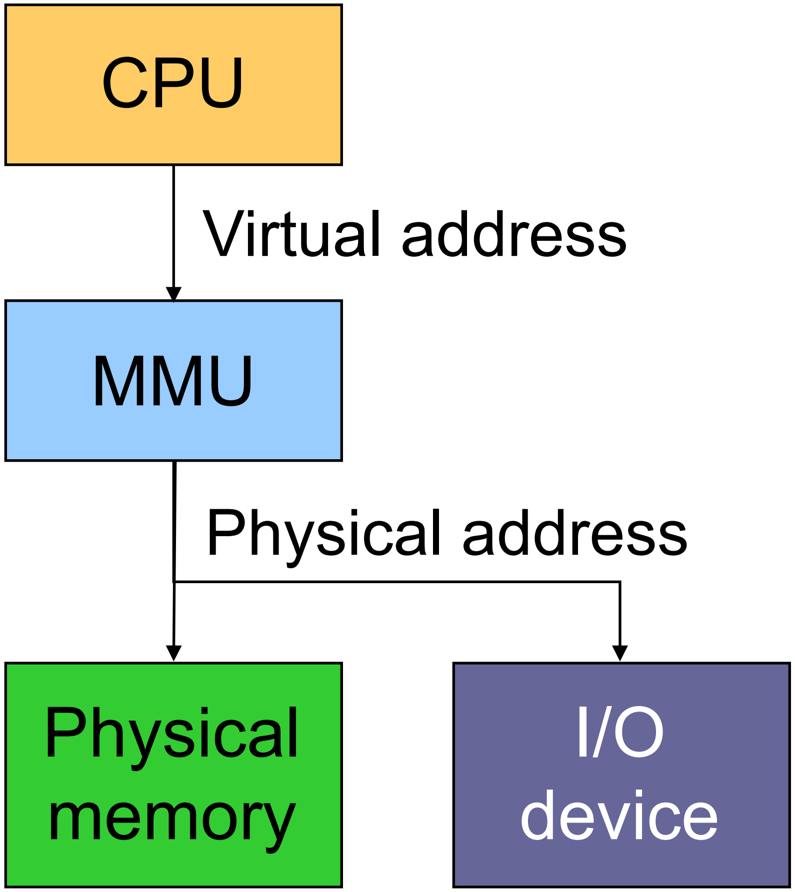
- Flexible
  - Processes (and their data) can move in memory as they execute, and be partially in memory and partially on disk
- Simple
  - Applications generate loads and stores to addresses in the contiguous, large, “fake” address space
- Efficient
  - 20/80 rule: 20% of memory gets 80% of references
  - Keep the 20% in physical memory
- Design issues
  - How is protection enforced?
  - How are processes relocated?
  - How is memory partitioned?

Address Mapping and Granularity

- Must have some "mapping" mechanism
  - Map virtual addresses to physical addresses in RAM or disk
- Mapping must have some granularity
  - Finer granularity provides more flexibility
  - Finer granularity requires more mapping information

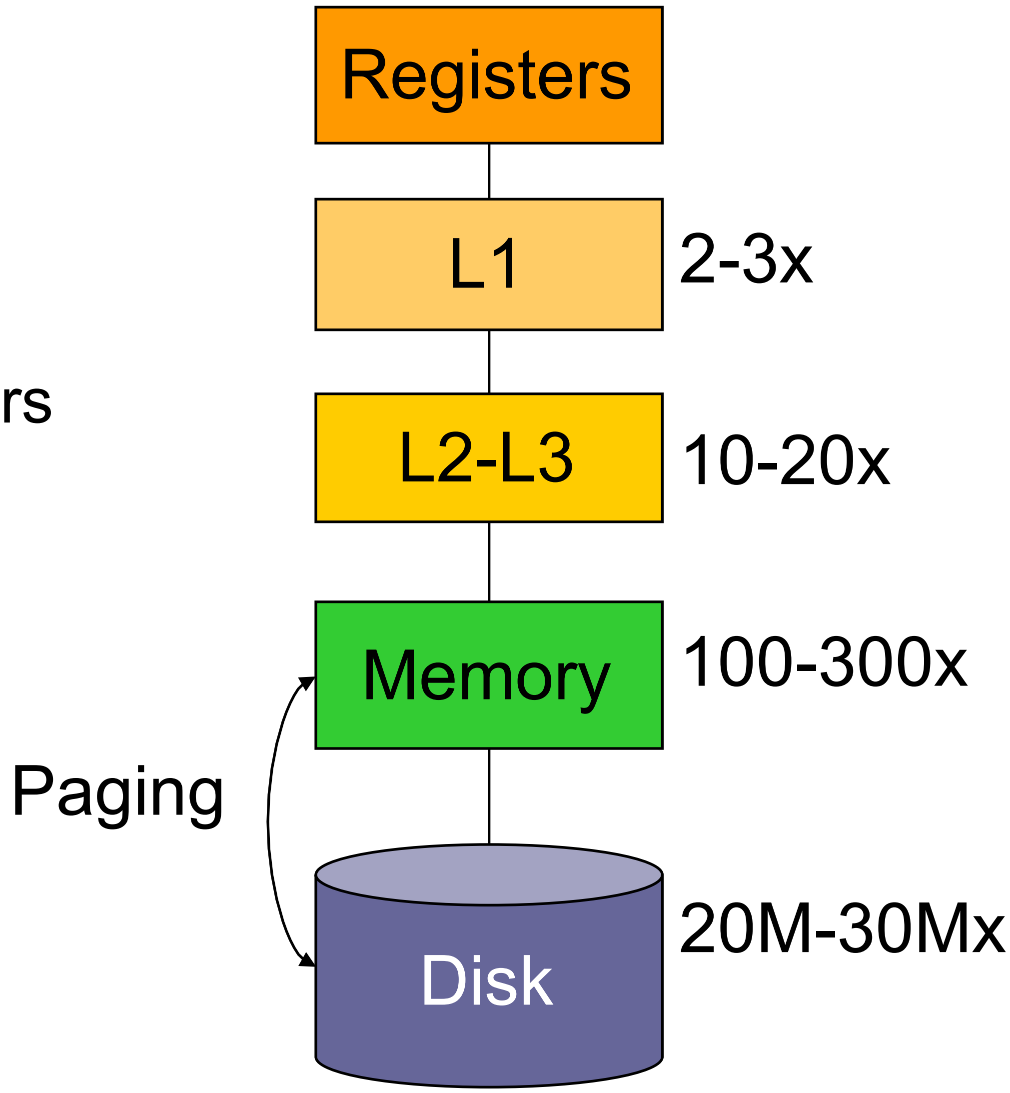
Generic Address Translation

- Memory Management Unite (MMU) translates virtual address into physical address for each load and store
- Combination of hardware and (privileged) software controls the translation
- CPU view
  - Virtual addresses
- Each process has its own memory space [0, high]
  - Address space
- Memory or I/O device view
  - Physical addresses



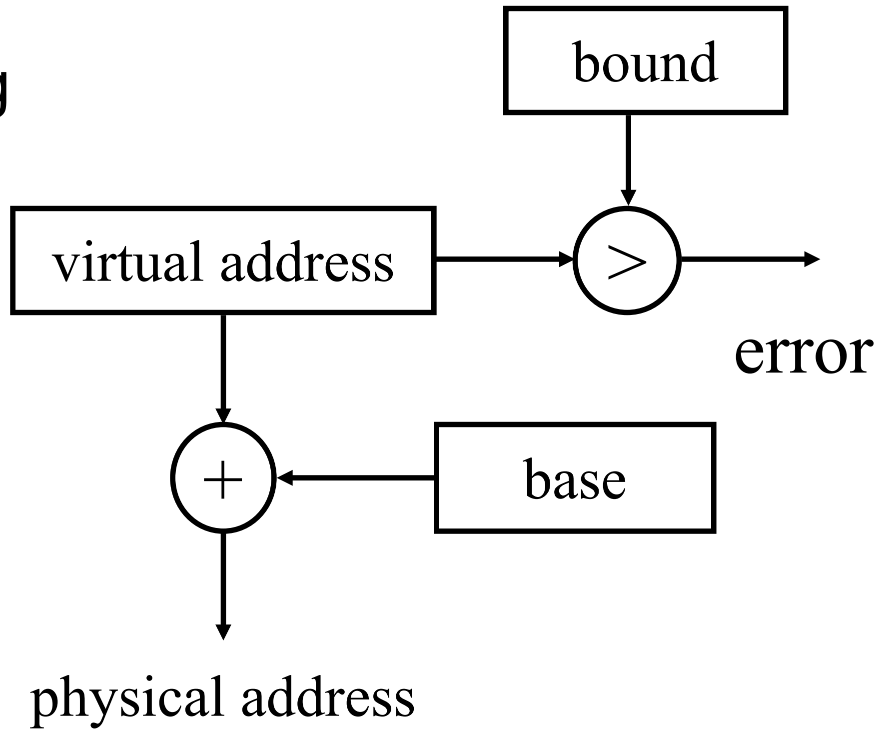
Goals of Translation

- Implicit translation for each memory reference
- A hit should be very fast
- Trigger an exception on a miss
- Protected from user's errors



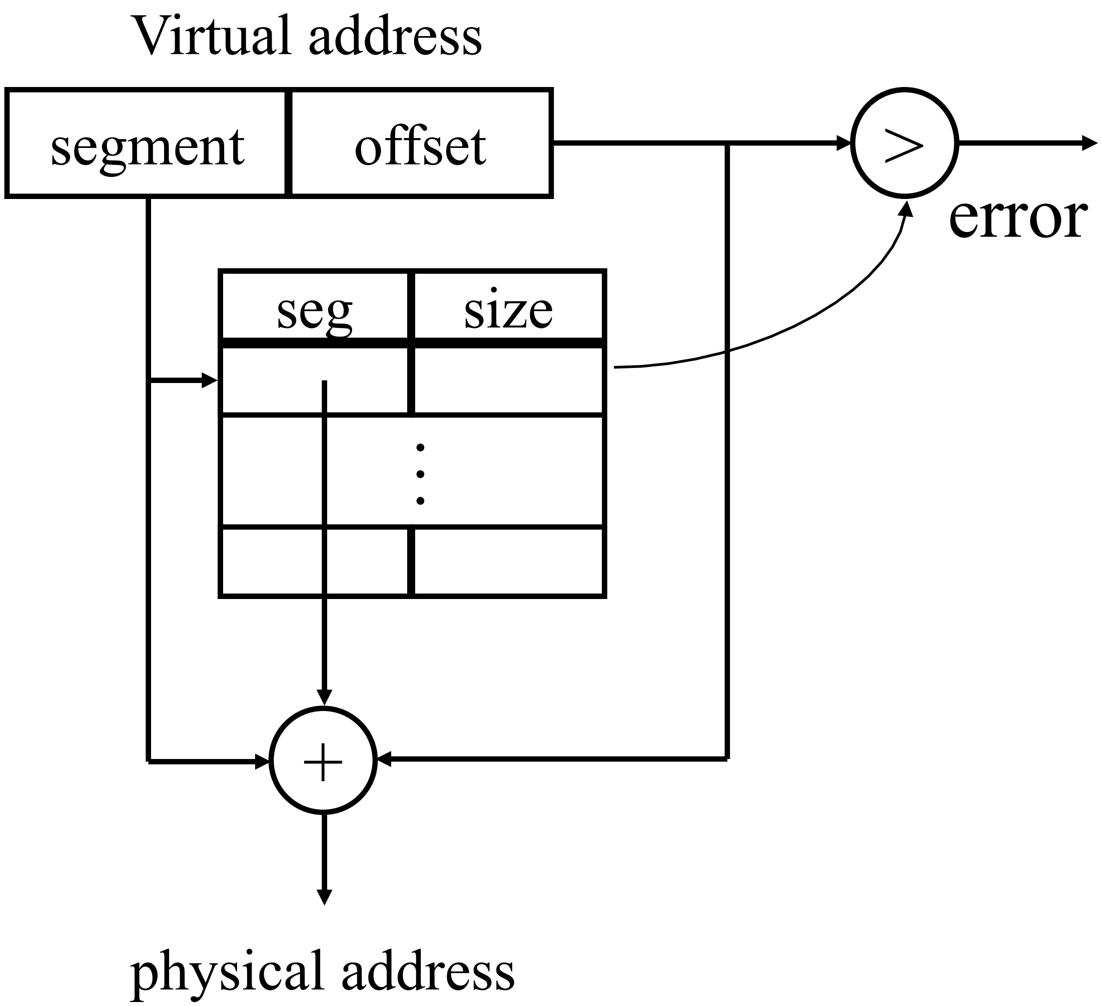
Base and Bound (or Limit)

- Built in Cray-1
- CPU has base and bound reg
- Base holds start address of running process; bound is length of its addressable space
- Protection
  - A process can only access physical memory in [base, base+bound]
- On a context switch
  - Save/restore base, bound regs
- Pros
  - Simple
- Cons
  - Can’t fit all processes, have to swap
  - Fragmentation in memory
  - Relocate processes when they grow
  - Compare and add on every instr.



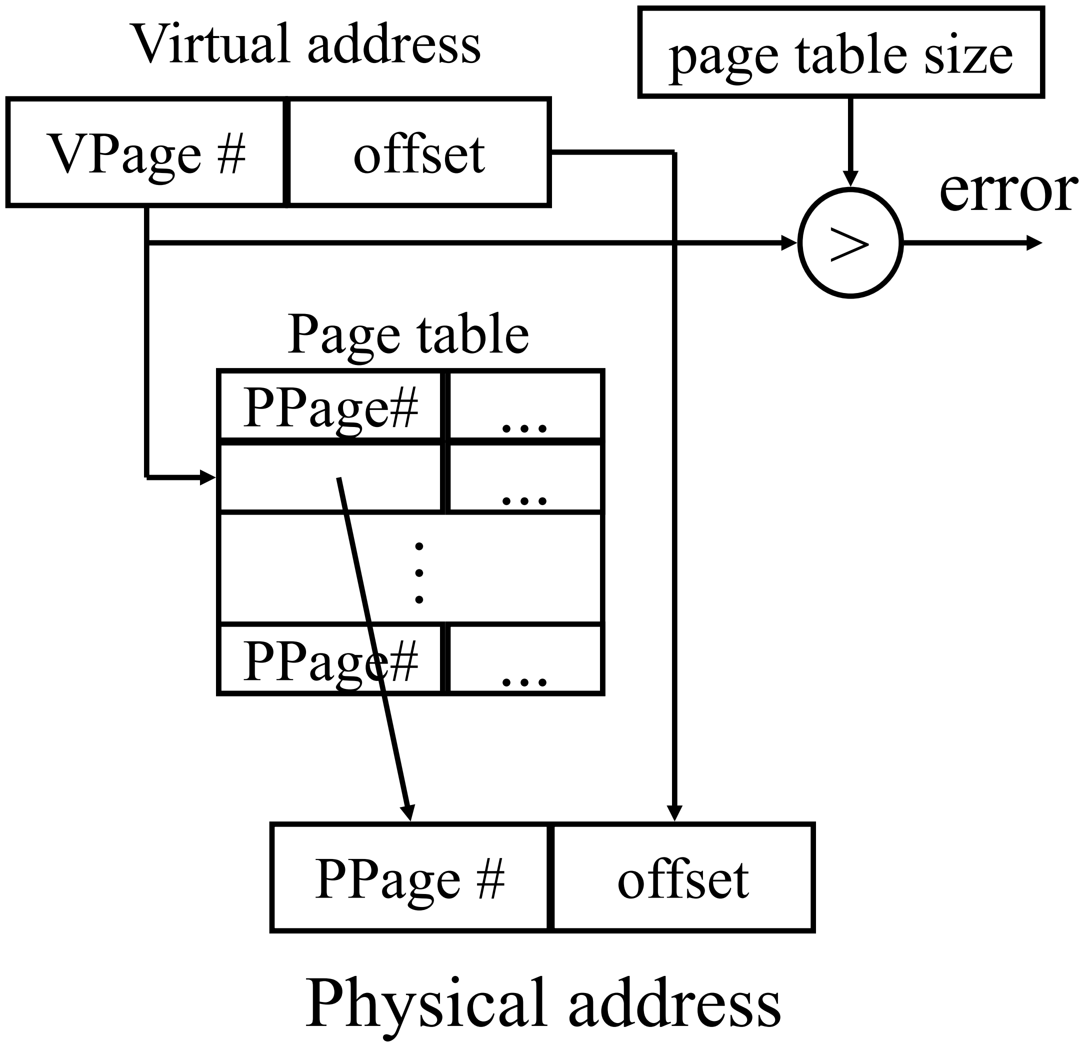
Segmentation

- Each process has a table of (seg, size)
- Treats (seg, size) as a fine-grained (base, bound)
- Protection
  - Each entry has (nil, read, write, exec)
- On a context switch
  - Save/restore table in kernel memory
- Pros
  - Efficient: programmer knows program and so segments
  - Provides logical protection
  - Easy to share data
- Cons
  - Complex management
  - Fragmentation



Paging

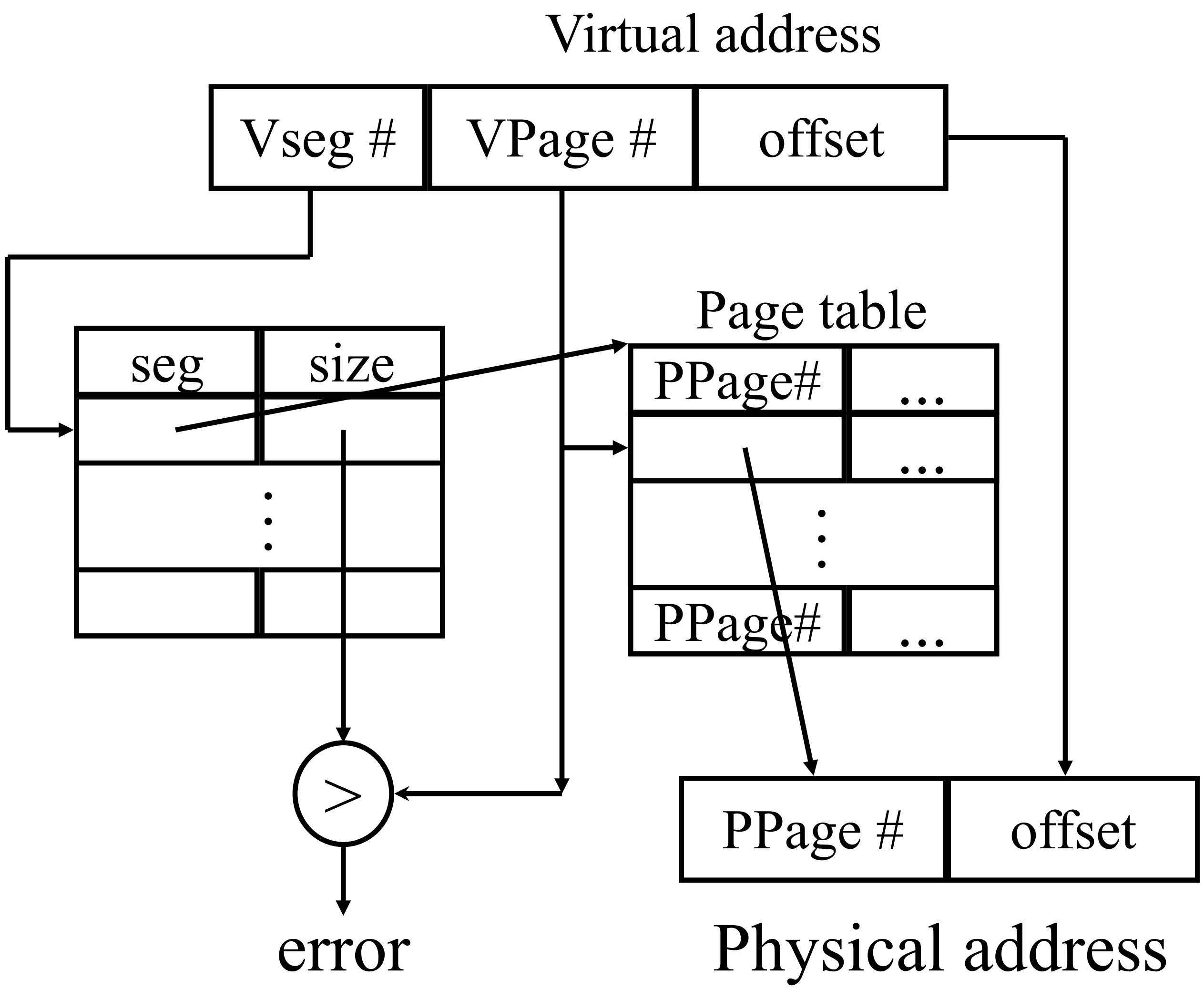
- Use a fixed size unit called page instead of segment
- Use a page table to translate
- Various bits in each entry
- Context switch
  - Similar to segmentation
- What should be the page size?
- Pros
  - Simple allocation
  - Easy to share
- Cons
  - Big table
  - How to deal with holes?



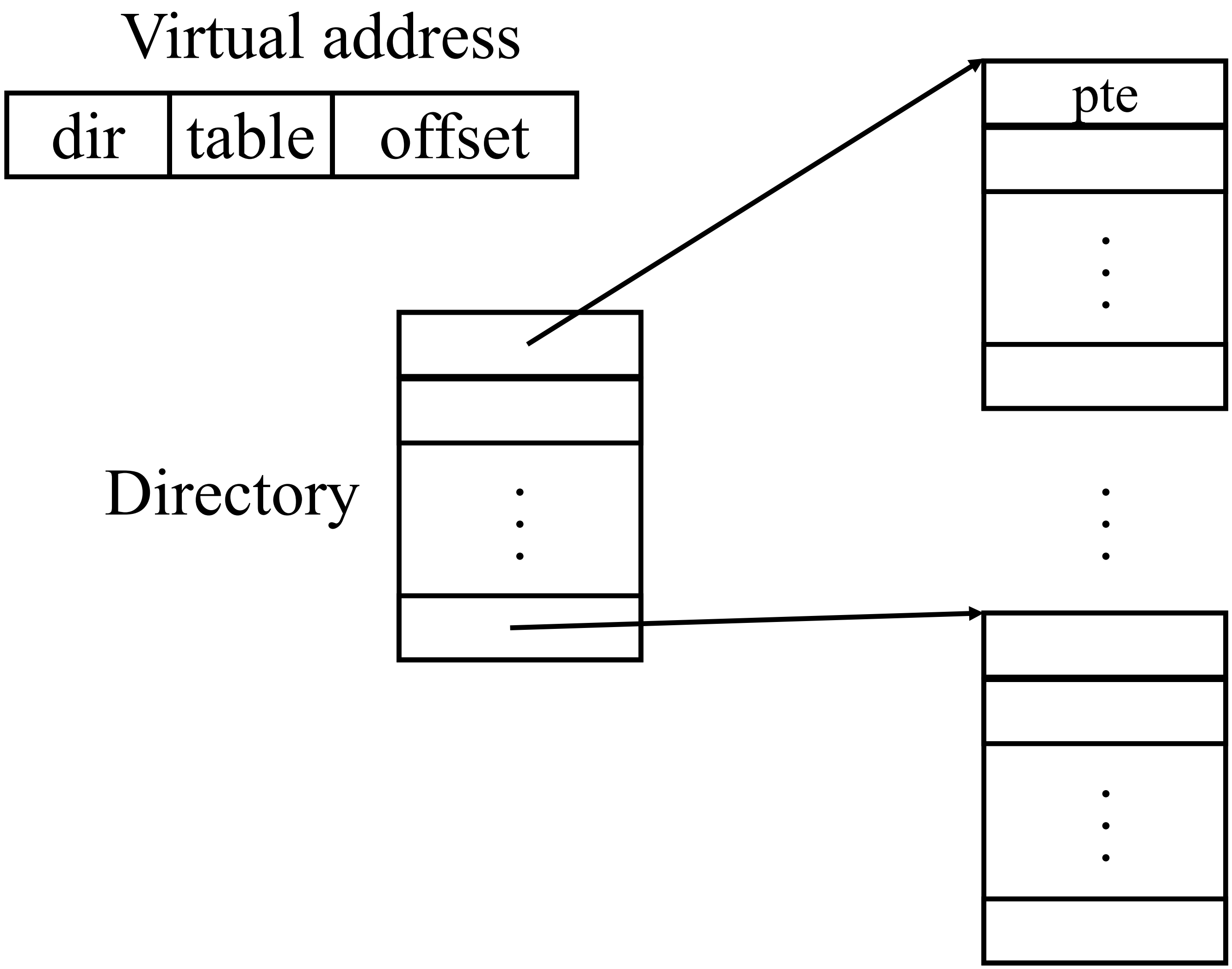
How Many PTEs Do We Need?

- Assume 4KB page
  - Needs “low order” 12 bits
- Worst case for 32-bit address machine
  - # of processes × 2^{20}
  - 2^{20} PTEs per page table (~4Mbytes), but there might be 10K processes. They won’t fit in memory together
- What about 64-bit address machine?
  - # of processes × 2^{52}
  - A page table cannot fit in a disk (2^{52} PTEs = 16PBytes)!

Segmentation with Paging



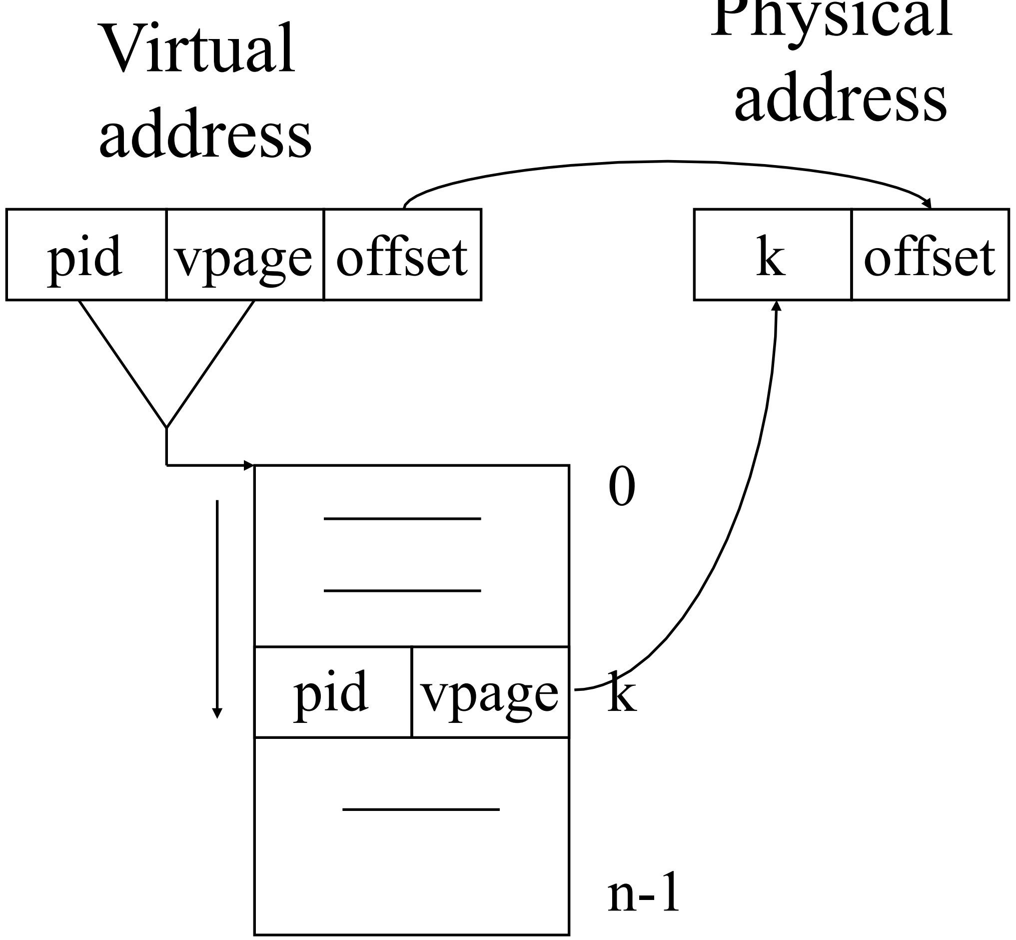
Multiple-Level Page Tables



What does this buy us?

Inverted Page Tables

- Main idea
  - One PTE for each physical page frame
  - Hash (Vpage, pid) to Ppage#
- Pros
  - Small page table for large address space
- Cons
  - Lookup is difficult
  - Overhead of managing hash chains, etc

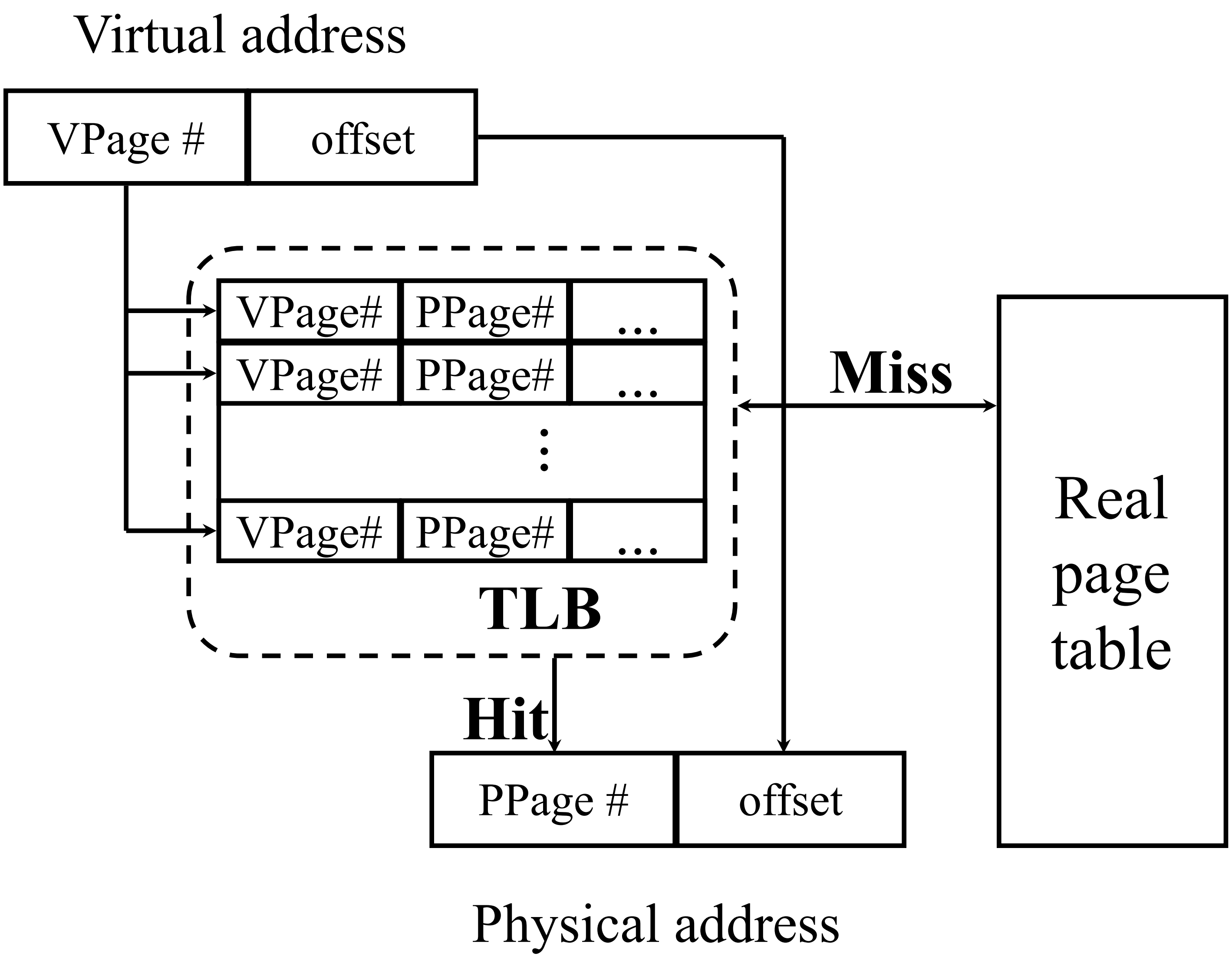


Inverted page table

Virtual-To-Physical Lookups

- Programs only know virtual addresses
  - Each program or process starts from 0 to high address
- Each virtual address must be translated
  - May involve walking through the hierarchical page table
  - Since the page table stored in memory, a program memory access may requires several actual memory accesses
- Solution
  - Cache "active" part of page table in a very fast memory

Translation Look-aside Buffer (TLB)



Bits in a TLB Entry

- Common (necessary) bits
  - Virtual page number
  - Physical page number: translated address
  - Valid
  - Access bits: kernel and user (nil, read, write)
- Optional (useful) bits
  - Process tag
  - Reference
  - Modify
  - Cacheable

Hardware-Controlled TLB

- On a TLB miss
  - Hardware loads the PTE into the TLB
    - Write back and replace an entry if there is no free entry
  - Generate a fault if the page containing the PTE is invalid
  - VM software performs fault handling
  - Restart the CPU
- On a TLB hit, hardware checks the valid bit
  - If valid, pointer to page frame in memory
  - If invalid, the hardware generates a page fault
    - Perform page fault handling
    - Restart the faulting instruction

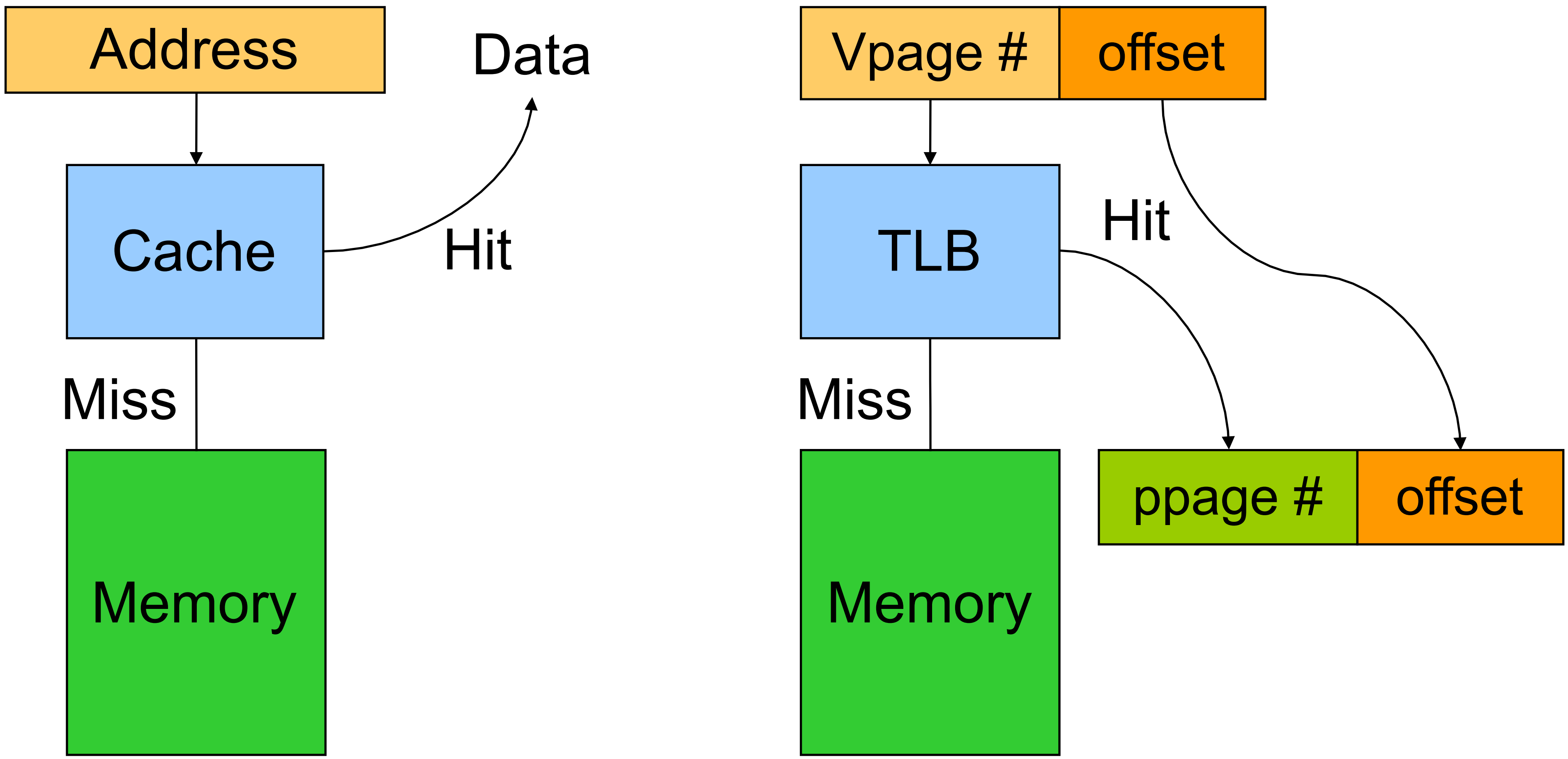
Software-Controlled TLB

- On a miss in TLB
  - Write back if there is no free entry
  - Check if the page containing the PTE is in memory
  - If not, perform page fault handling
  - Load the PTE into the TLB
  - Restart the faulting instruction
- On a hit in TLB, the hardware checks valid bit
  - If valid, pointer to page frame in memory
  - If invalid, the hardware generates a page fault
    - Perform page fault handling
    - Restart the faulting instruction

Hardware vs. Software Controlled

- Hardware approach
  - Efficient
  - Inflexible
  - Need more space for page table
- Software approach
  - Flexible
  - Software can do mappings by hashing
    - PP# $\rightarrow$ (Pid, VP#)
    - (Pid, VP#) $\rightarrow$ PP#
  - Can deal with large virtual address space

Cache vs. TLB



- Similarities
  - Cache a portion of memory
  - Write back on a miss
- Differences
  - Associativity
  - Consistency

TLB Related Issues

- What TLB entry to be replaced?
  - Random
  - Pseudo LRU
- What happens on a context switch?
  - Process tag: change TLB registers and process register
  - No process tag: Invalidate the entire TLB contents
- What happens when changing a page table entry?
  - Change the entry in memory
  - Invalidate the TLB entry

Consistency Issues

- "Snoopy" cache protocols (hardware)
  - Maintain consistency with DRAM, even when DMA happens
- Consistency between DRAM and TLBs (software)
  - You need to flush related TLBs whenever changing a page table entry in memory
- TLB "shoot-down"
  - On multiprocessors, when you modify a page table entry, you need to flush all related TLB entries on all processors, why?

Summary

- Virtual Memory
  - Virtualization makes software development easier and enables memory resource utilization better
  - Separate address spaces provide protection and isolate faults
- Address translation
  - Base and bound: very simple but limited
  - Segmentation: useful but complex
- Paging
  - TLB: fast translation for paging
  - VM needs to take care of TLB consistency issues