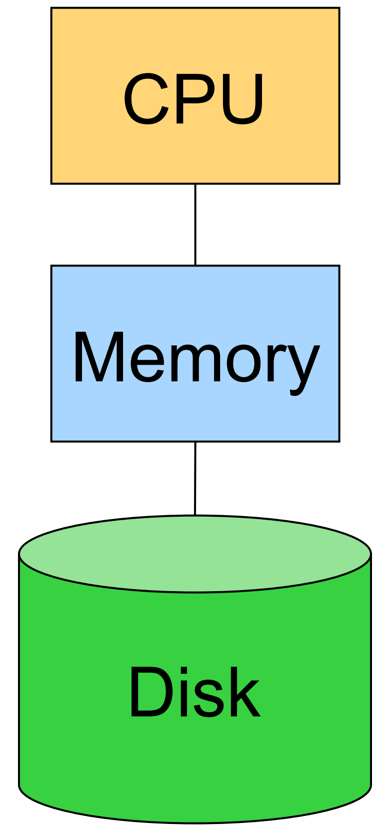
COS 318: Operating Systems

Virtual Memory and Address Translation

The Big Picture

- DRAM is fast, but relatively expensive
  - $25/GB
  - 20-30ns latency
  - 10-80GB's/sec
- Disk is inexpensive, but slow
  - $0.2-1/GB (100 less expensive)
  - 5-10ms latency (200K-400K times slower)
  - 40-80MB/sec per disk (1,000 times less)
- Our goals
  - Run programs as efficiently as possible
  - Make the system as safe as possible

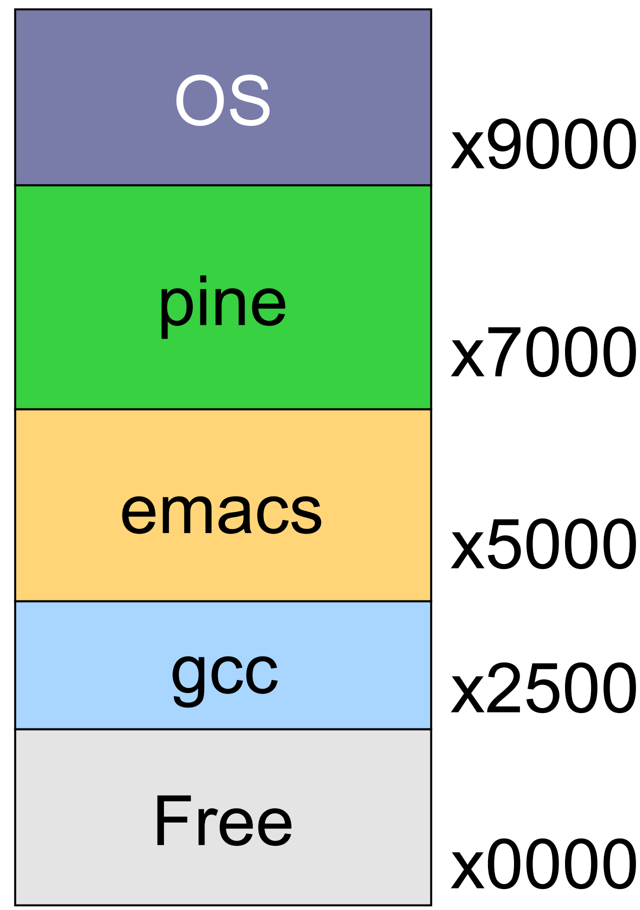


Issues

- 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

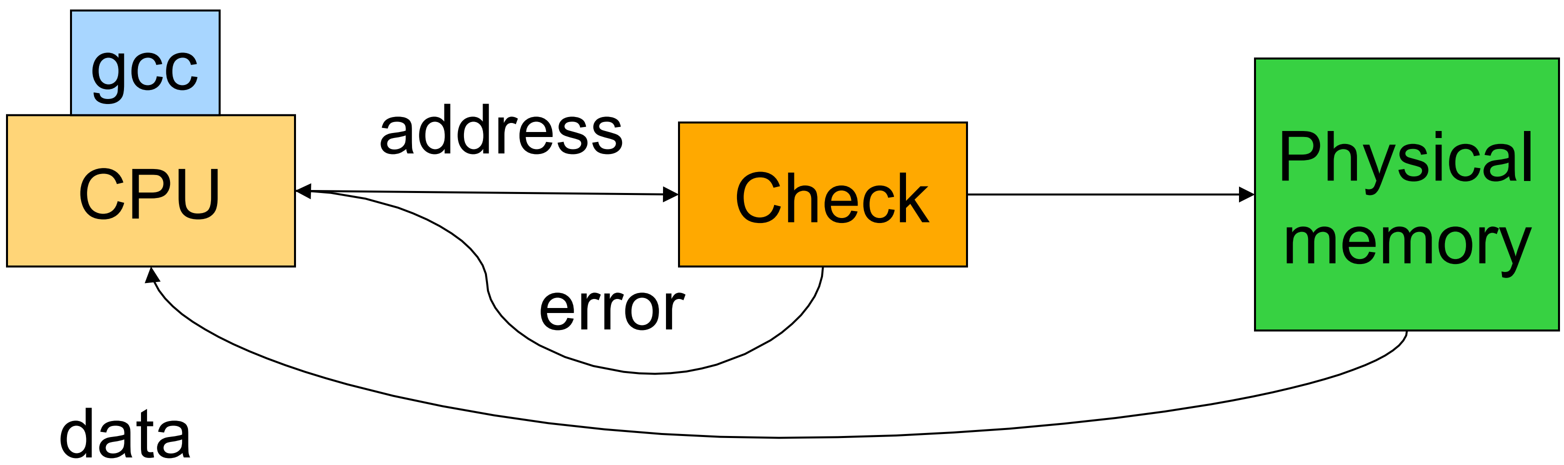
Consider A Simple System

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



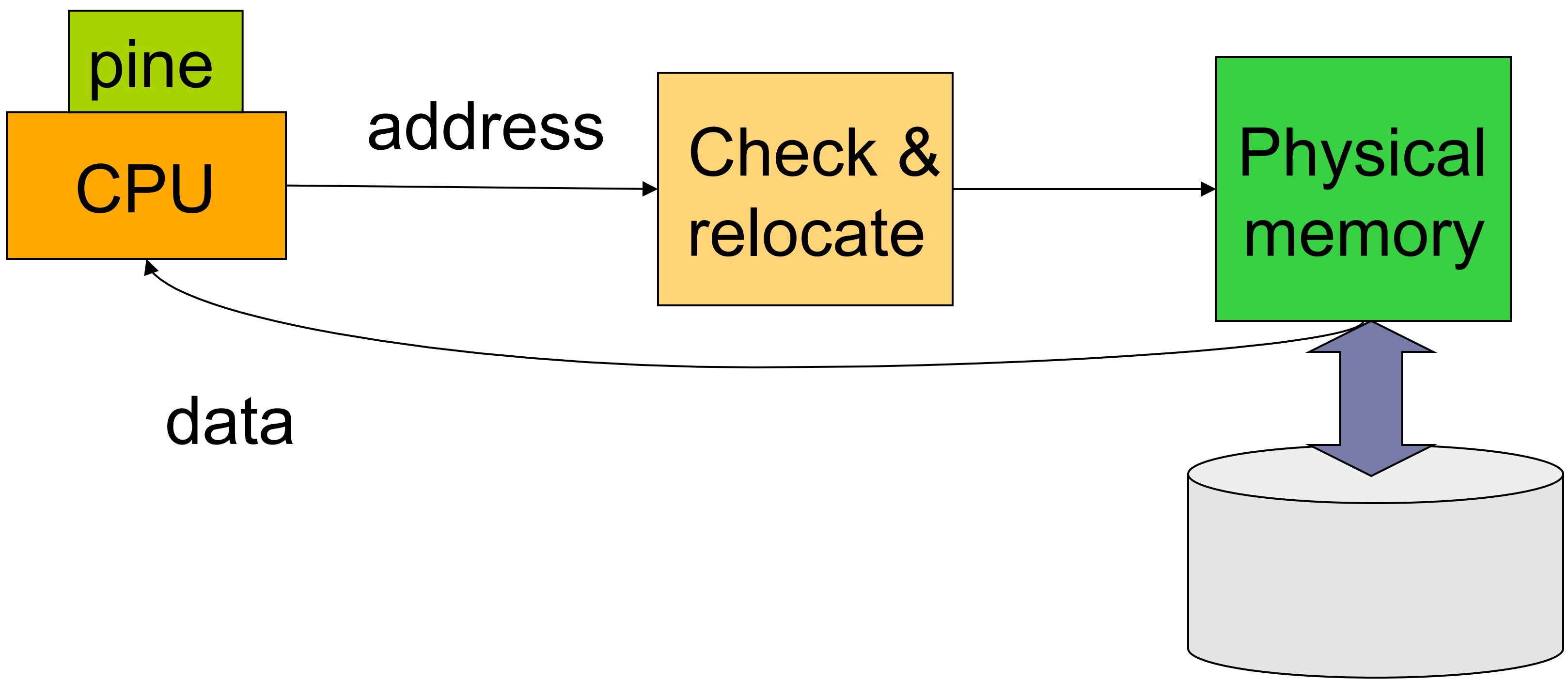
Protection Issue

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



Expansion or Transparency Issue

- A process should be able to run regardless of its physical location or the physical memory size
- Give each process a large, static “fake” address space
- As a process runs, relocate each load and store to its actual memory



Virtual Memory

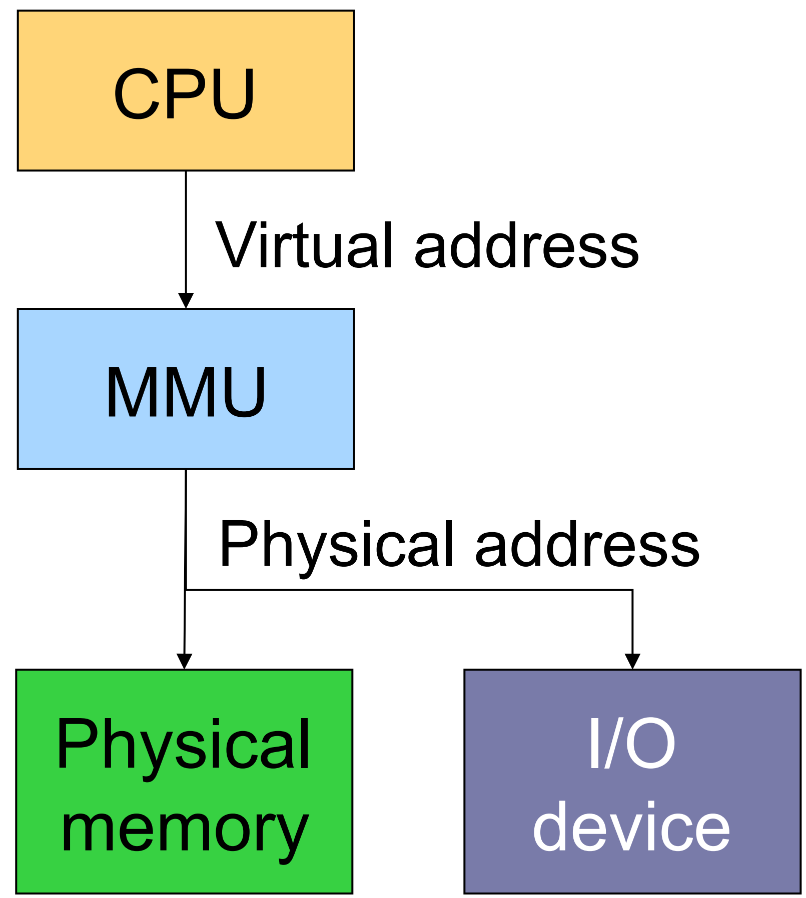
- Flexible
  - Processes can move in memory as they execute, partially in memory and partially on disk
- Simple
  - Make applications very simple in terms of memory accesses
- Efficient
  - 20/80 rule: 20% of memory gets 80% of references
  - Keep the 20% in physical memory
- Design issues
  - How is protection enforced?
  - How are processes relocated?
  - How is memory partitioned?

Address Mapping and Granularity

- Must have some "mapping" mechanism
  - Virtual addresses map to DRAM physical addresses or disk addresses
- Mapping must have some granularity
  - Granularity determines flexibility
  - Finer granularity requires more mapping information
- Extremes
  - Any byte to any byte: mapping equals program size
  - Map whole segments: larger segments problematic

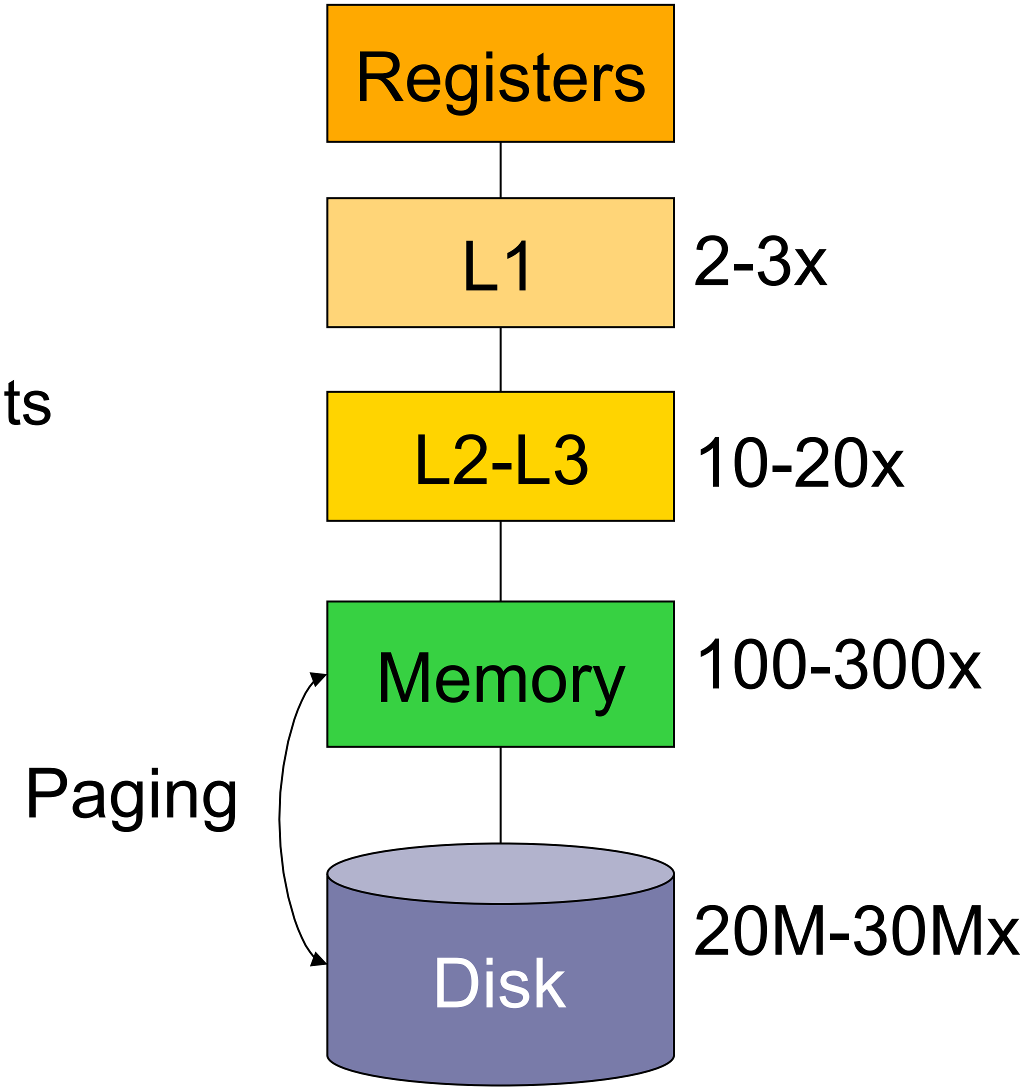
Generic Address Translation

- Memory Management Unit (MMU) translates virtual address into physical address for each load and store
- Software (privileged) 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



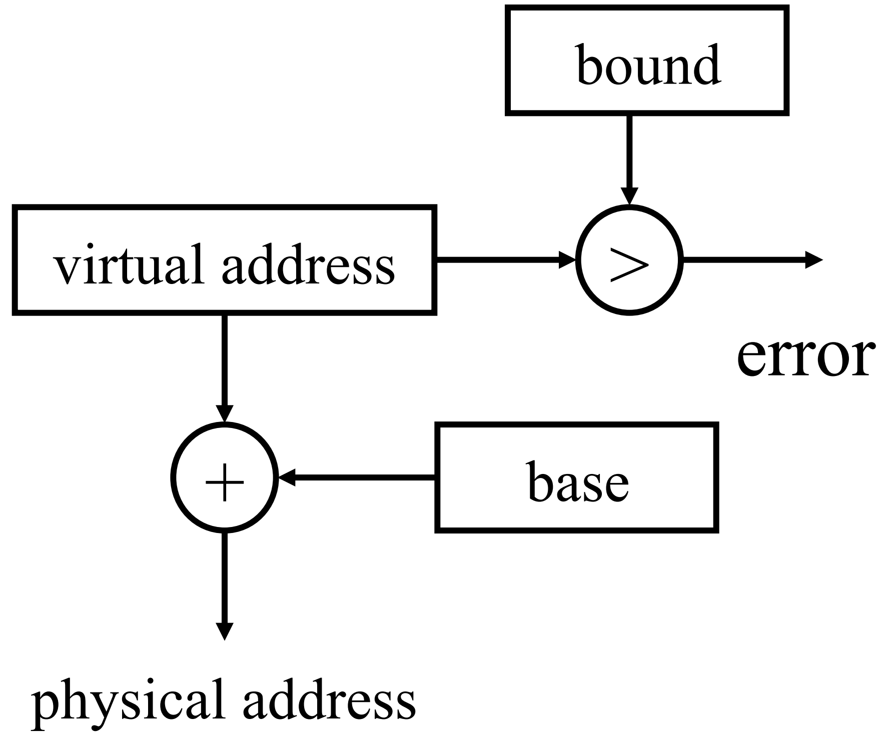
Goals of Translation

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



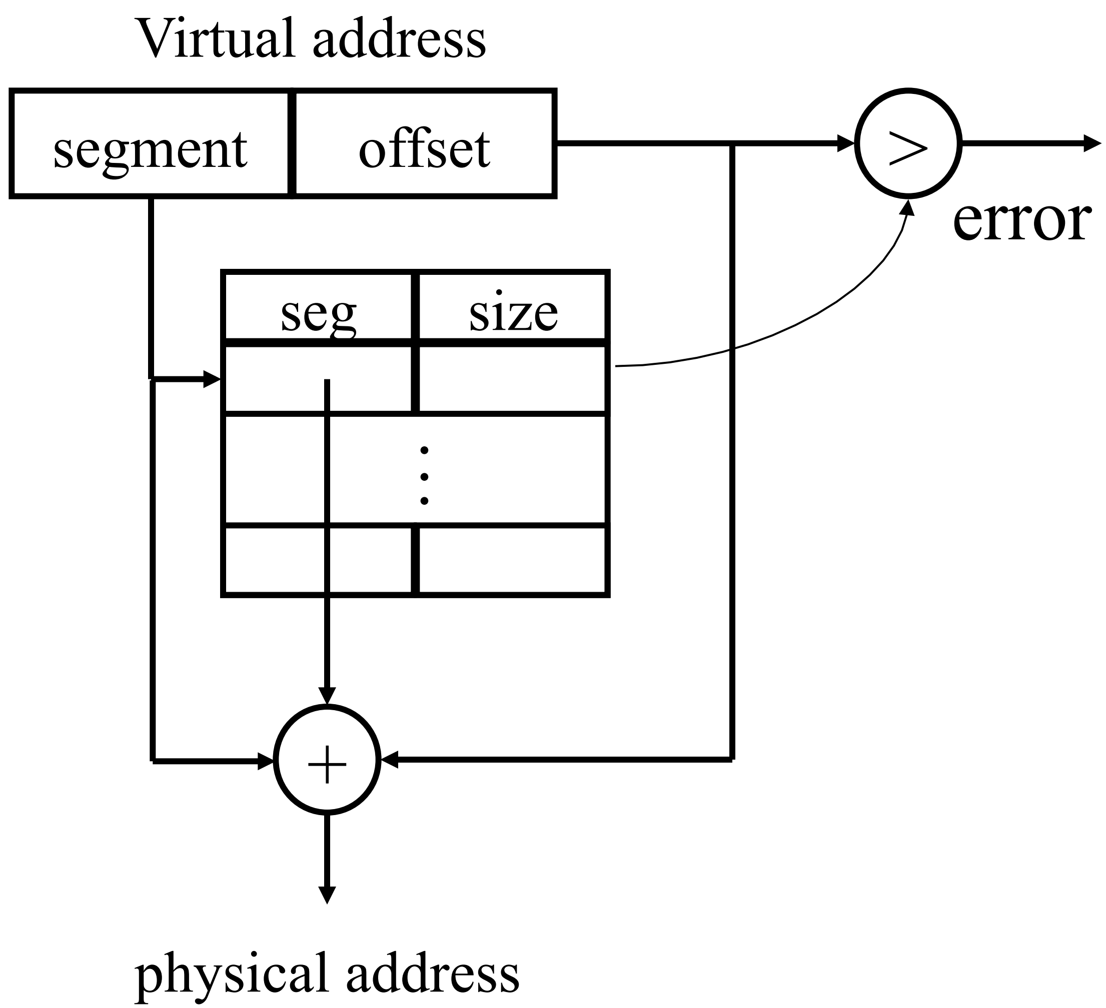
Base and Bound

- Built in Cray-1
- Each process has a pair (base, bound)
- Protection
  - A process can only access physical memory in [base, base+bound]
- On a context switch
  - Save/restore base, bound registers
- Pros
  - Simple
  - Flat and no paging
- Cons
  - Fragmentation
  - Hard to share
  - Difficult to use disks



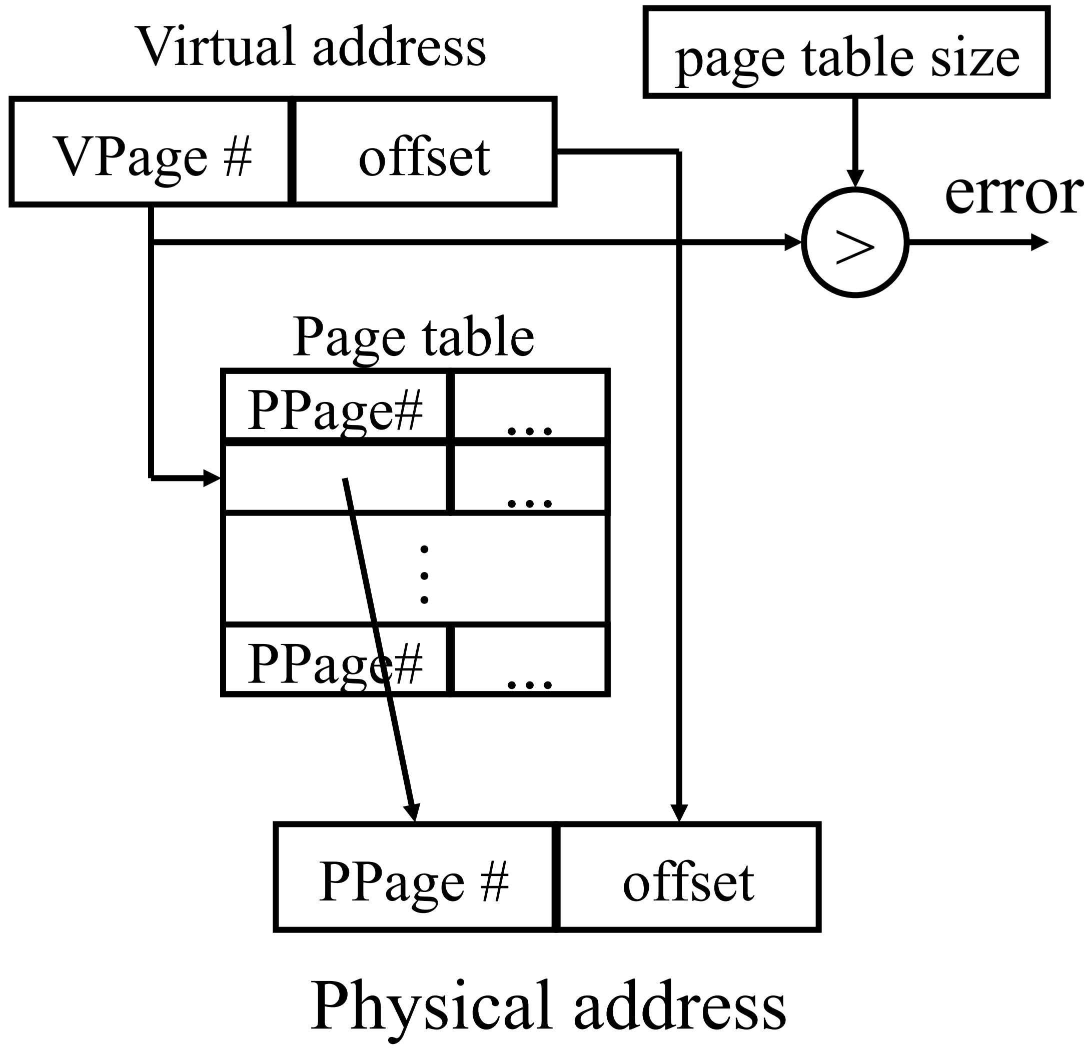
Segmentation

- Each process has a table of (seg, size)
- Treats (seg, size) has a fine-grained (base, bound)
- Protection
  - Each entry has (nil, read, write, exec)
- On a context switch
  - Save/restore the table and a pointer to the table in kernel memory
- Pros
  - Efficient
  - Easy to share
- Cons
  - Complex management
  - Fragmentation within a segment

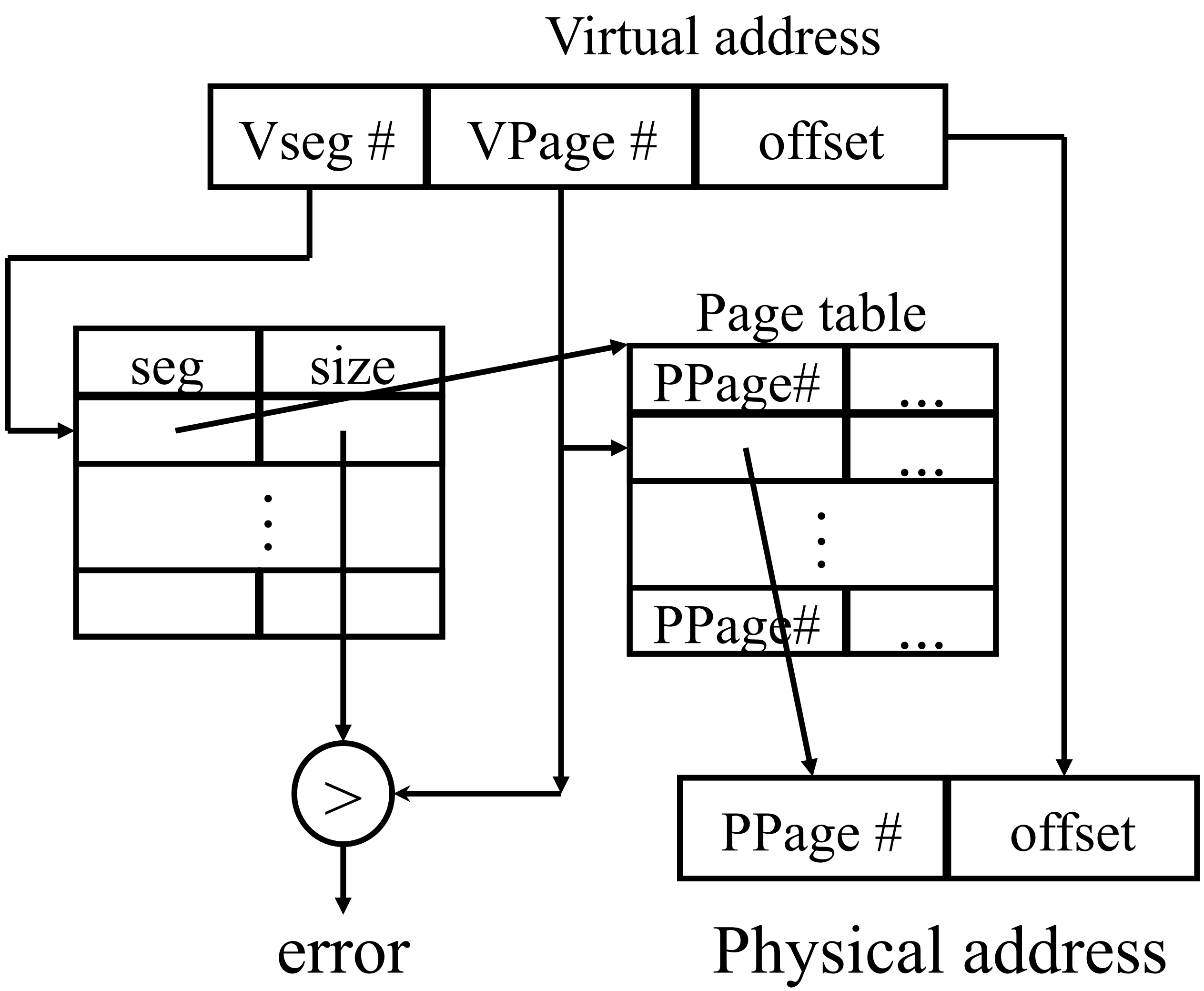


Paging

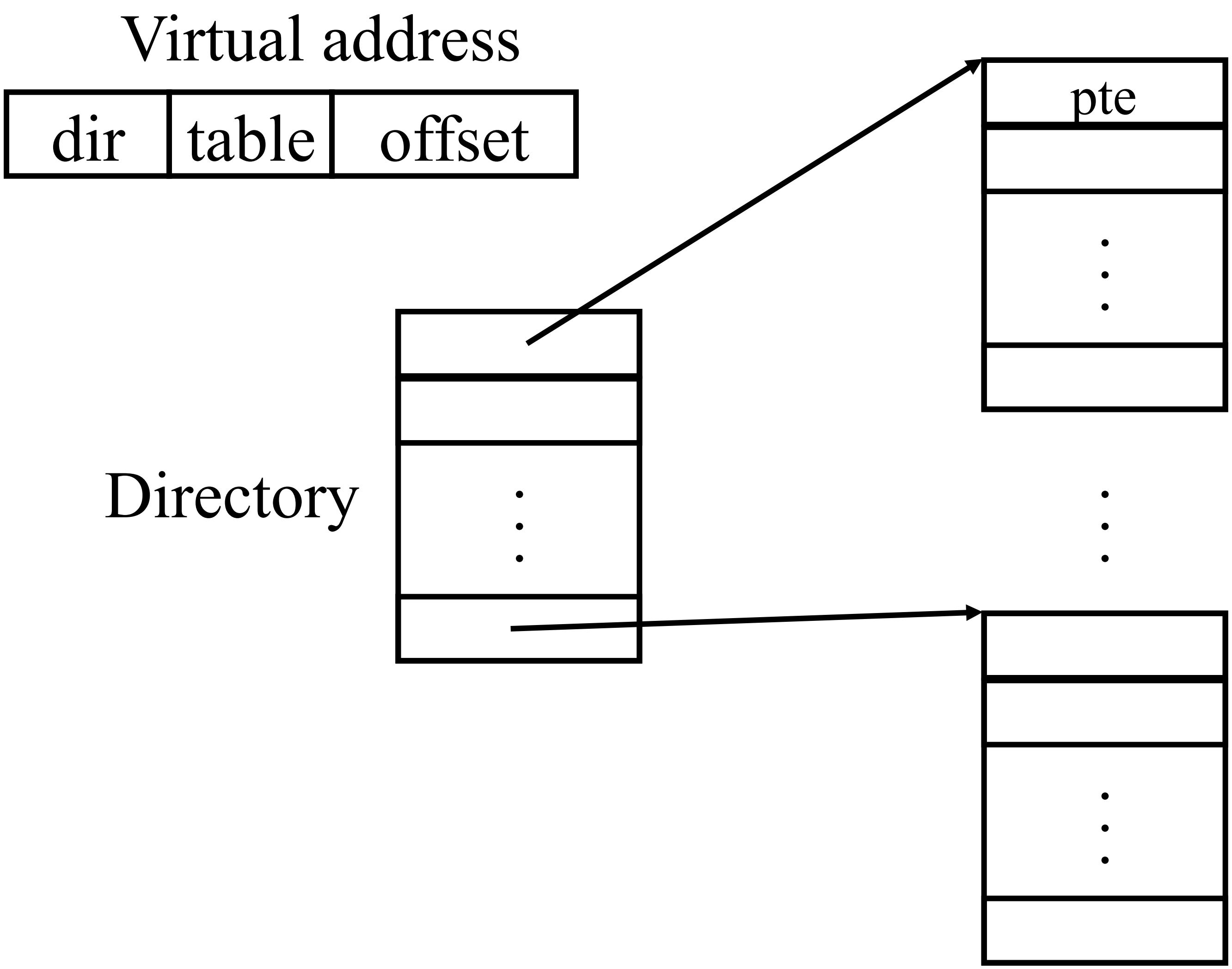
- Use a fixed size unit called page instead of segment
- Use a page table to translate
- Various bits in each entry
- Context switch
  - Similar to segmentation
- What should page size be?
- Pros
  - Simple allocation
  - Easy to share
- Cons
  - Big table
  - How to deal with holes?



Segmentation with Paging



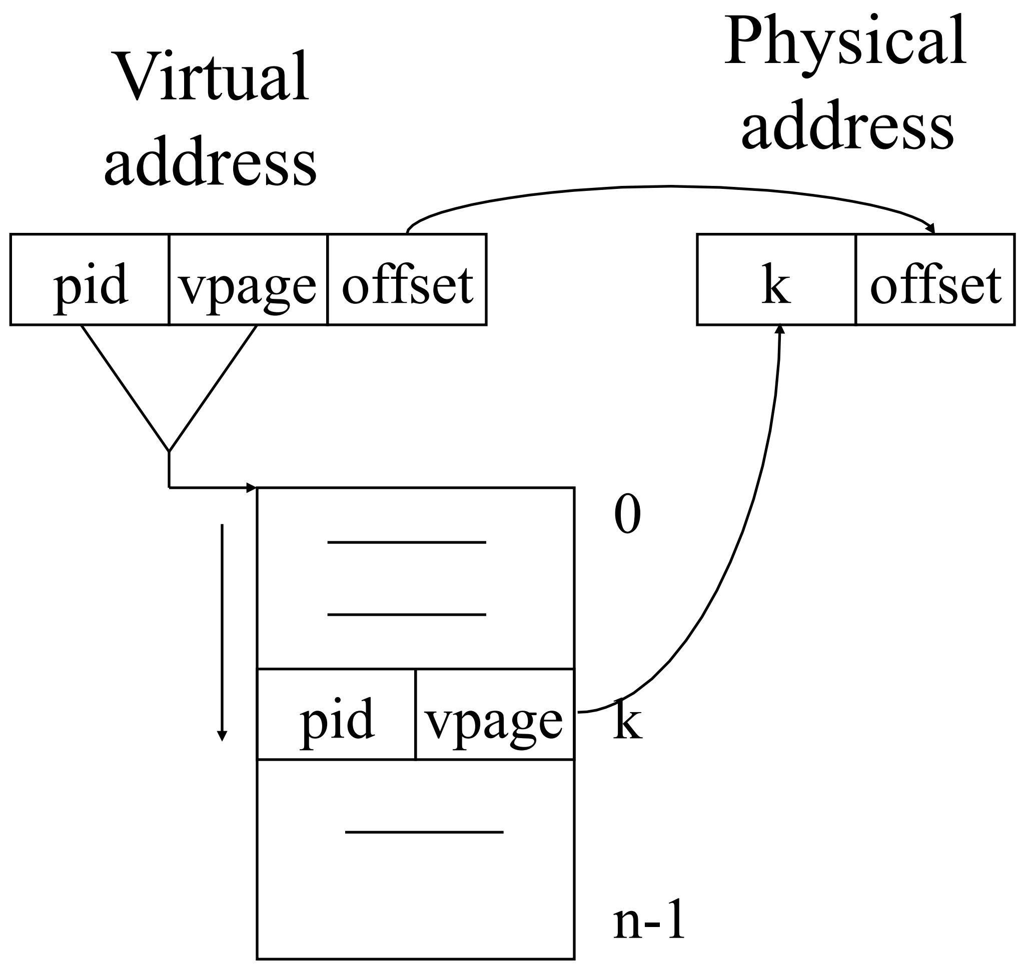
Multiple-Level Page Tables



What does this buy us?

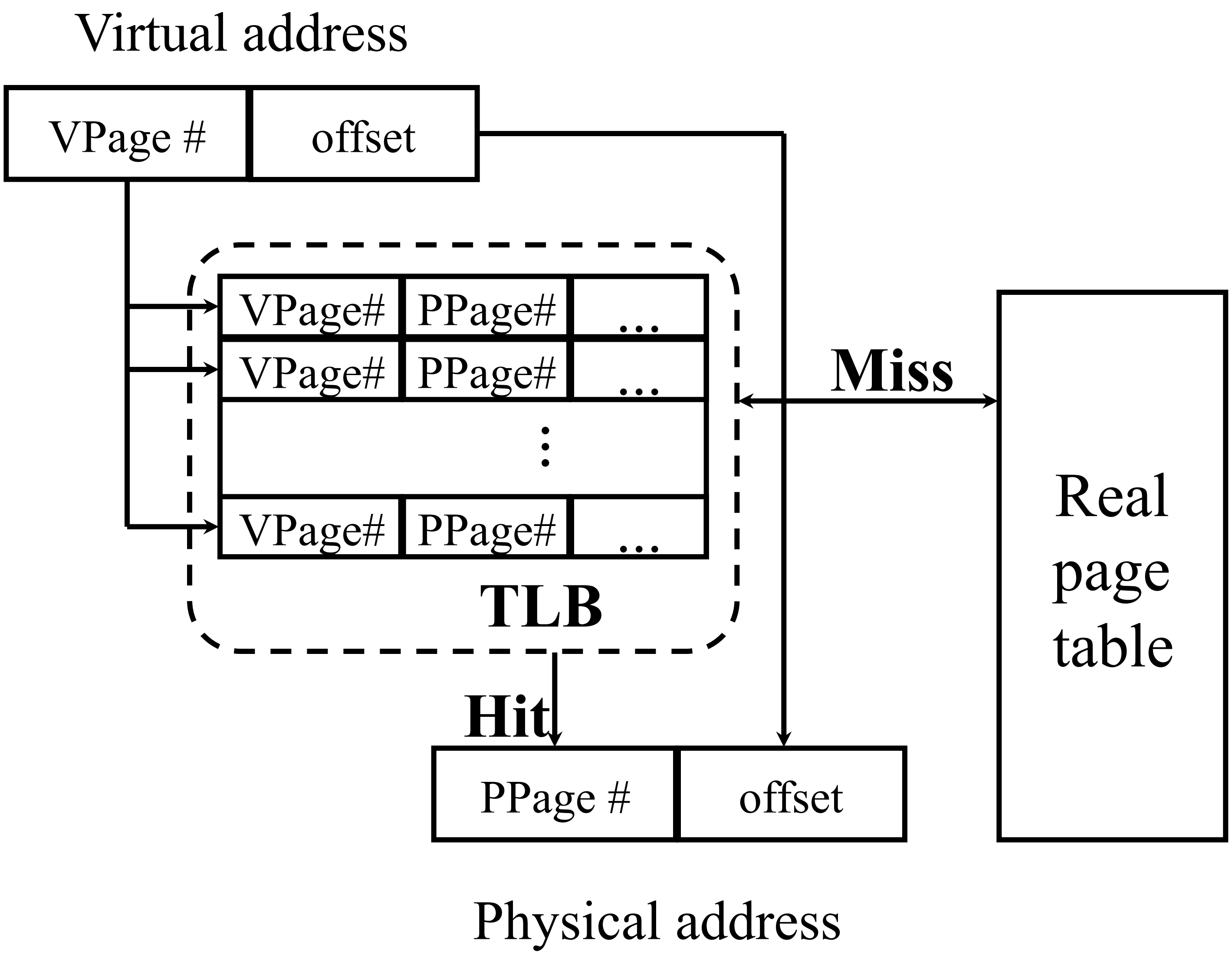
Inverted Page Tables

- 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

Translation Look-aside Buffer (TLB)



Bits in a TLB Entry

- Common (necessary) bits
  - Virtual page number: match with the virtual address
  - 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

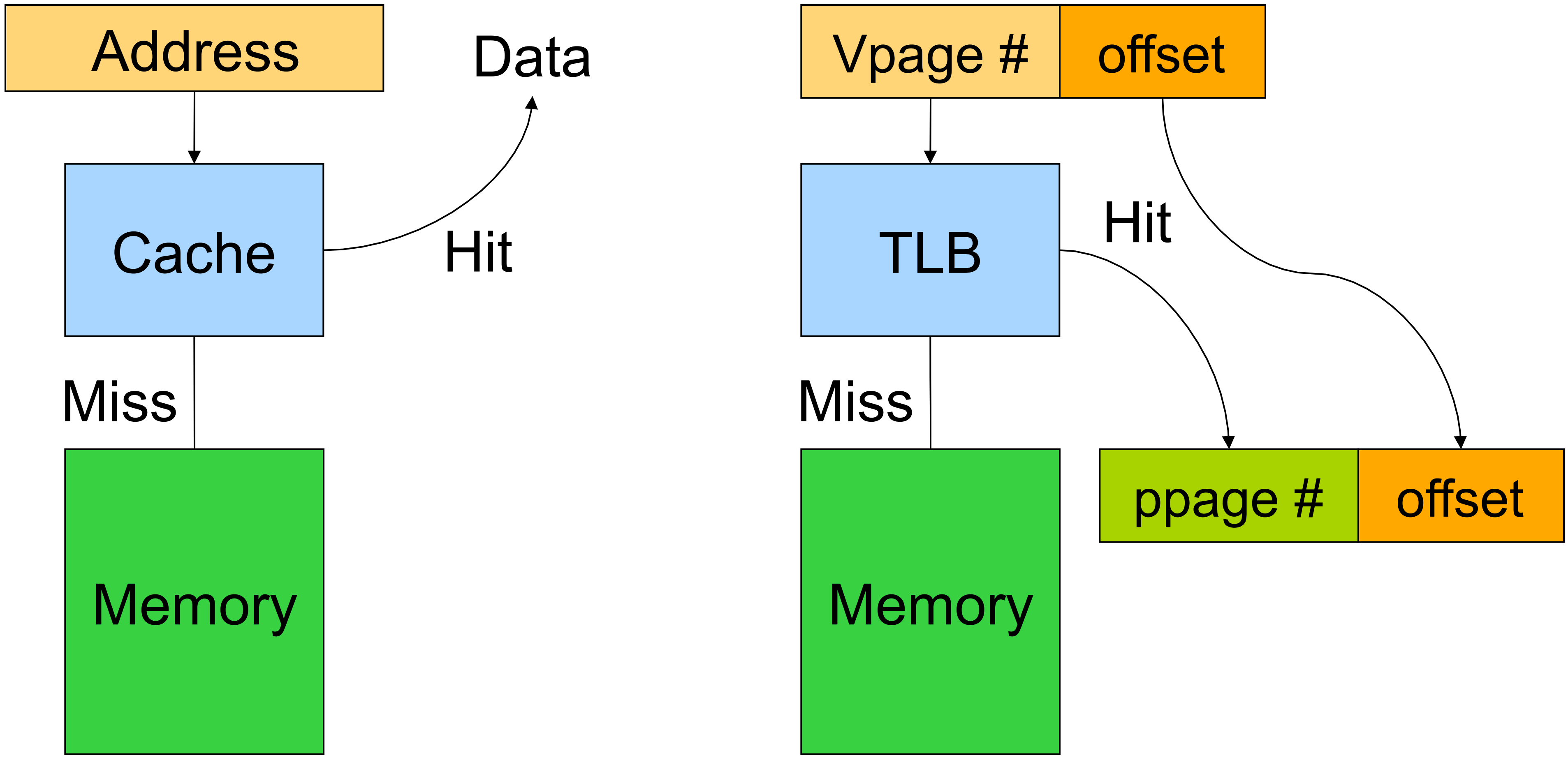
Software-Controlled TLB

- 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# → (Pid, VP#)
    - (Pid, VP#) → 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

- Regroup NOW