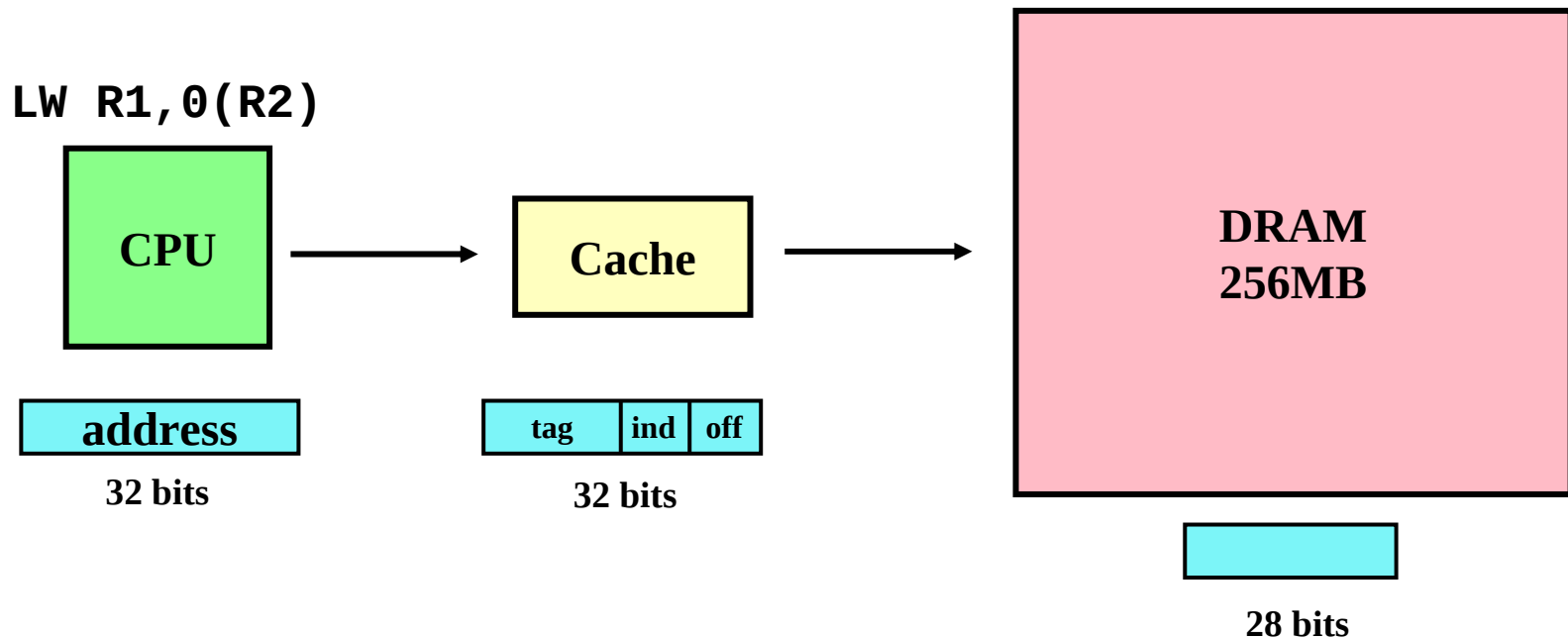


Lecture 15: Virtual Memory

- Last Time:
 - Cache wrap-up
 - Virtual memory motivation
- Today
 - Virtual memory implementations

Physical Memory Addressing



Webster's definition of “virtual”

Pronunciation: 'v&r-ch&-w&l, -ch&l; 'v&rch-w&l

Function: *adjective*

Etymology: Middle English, possessed of certain physical virtues, from Medieval Latin *virtualis*, from Latin *virtus* strength, virtue

1 : being such in essence or effect though not formally recognized or admitted <a *virtual* dictator>

2 : of, relating to, or using virtual memory

3 : of, relating to, or being a hypothetical particle whose existence is inferred from indirect evidence <*virtual* photons>

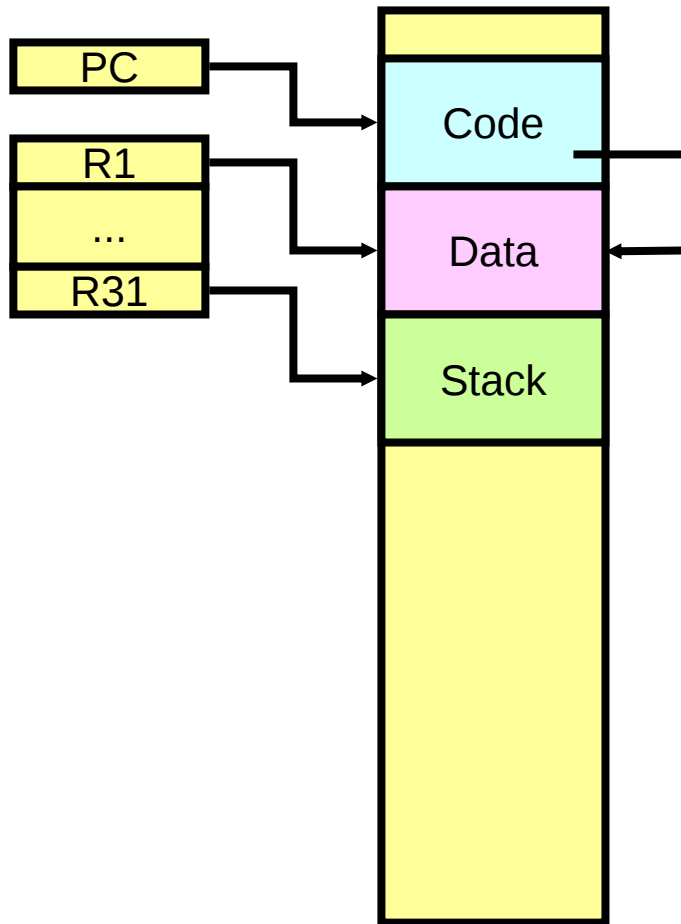
The goal of virtual memory

- Make it appear as if each process has:
 - It's own private memory
 - The memory is nearly infinite in size
- The challenge... Physical memory is:
 - Limited in size
 - Shared by all of the processes running on the machine
- The job of the virtual memory system is to maintain the illusion we want, given the physical limitations.

What if?

- A program is loaded into different places in memory each time it runs?
 - Relocation
- A program wants to use more memory than physically exists?
 - Page to disk
- We want to switch between multiple programs that use different data?
 - Protection

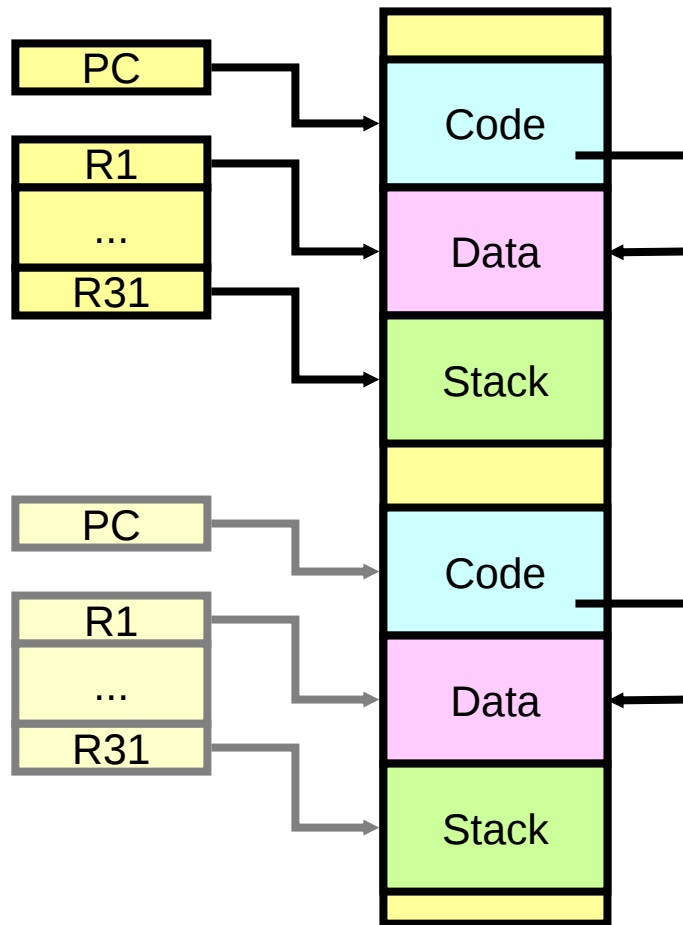
Simple View of Memory



- Single *program* runs at a time
- Code and static data are at fixed locations
 - code starts at fixed location, e.g., 0x100
 - subroutines may be at fixed locations (absolute jumps)
- data locations may be *wired* into code
- Stack accesses relative to stack pointer.

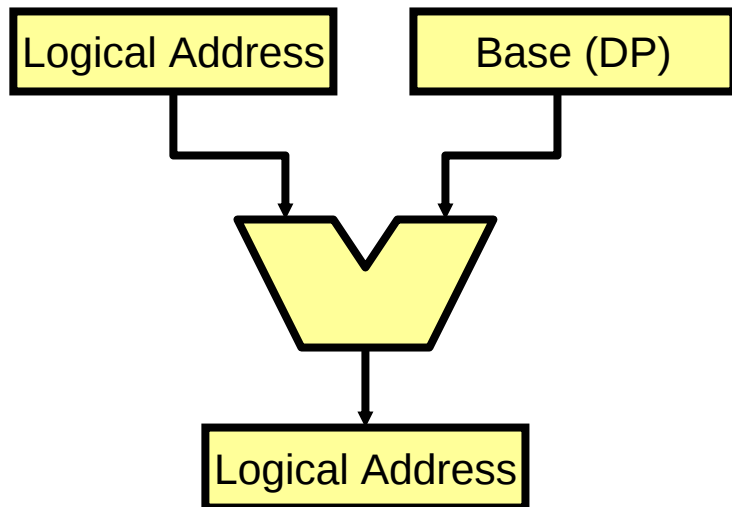
Running Two Programs (Relocation)

No Protection



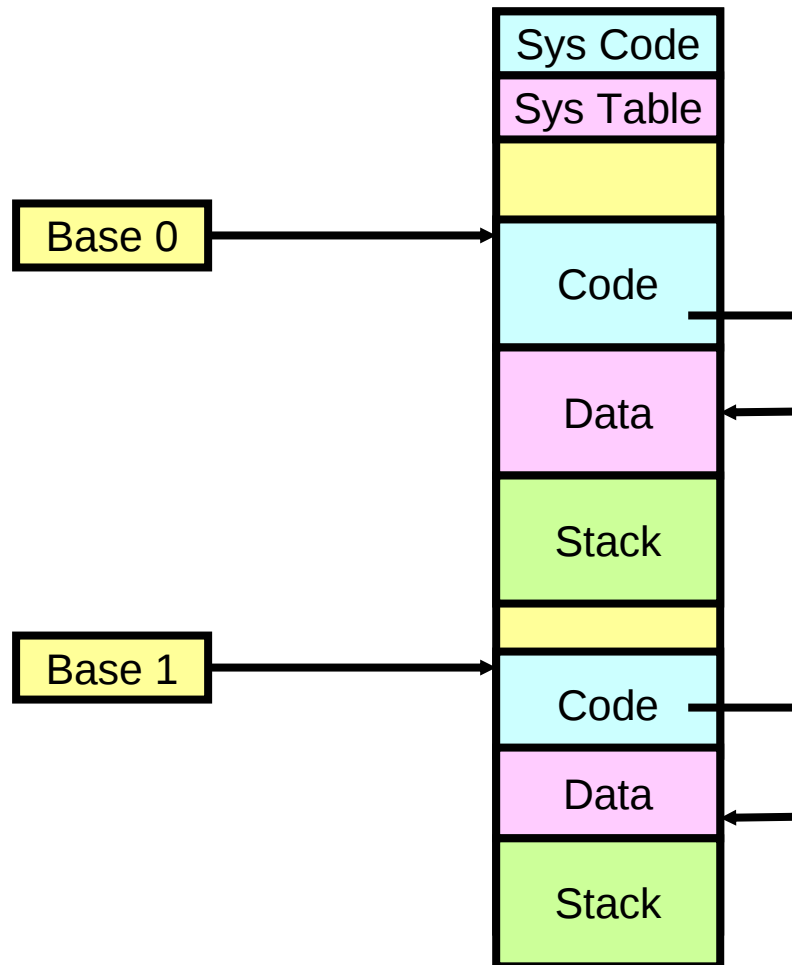
- Need to relocate *logical* addresses to *physical* locations
- Stack is already relocatable
 - all accesses relative to SP
- Code can be made relocatable
 - allow only relative jumps
 - all accesses relative to PC
- Data segment
 - can calculate all addresses relative to a DP
 - expensive
 - faster with hardware support
 - base register

Base-Register Addressing



- Add a single base register, BR, to hardware
- Base register loaded with data pointer (DP) for current program
- All data addresses added to base before accessing memory
 - Can relocate code too
- Often implemented with a three-input adder
- Need to bypass base register to access system tables for program switching
 - a place to stand

Base Register Addressing

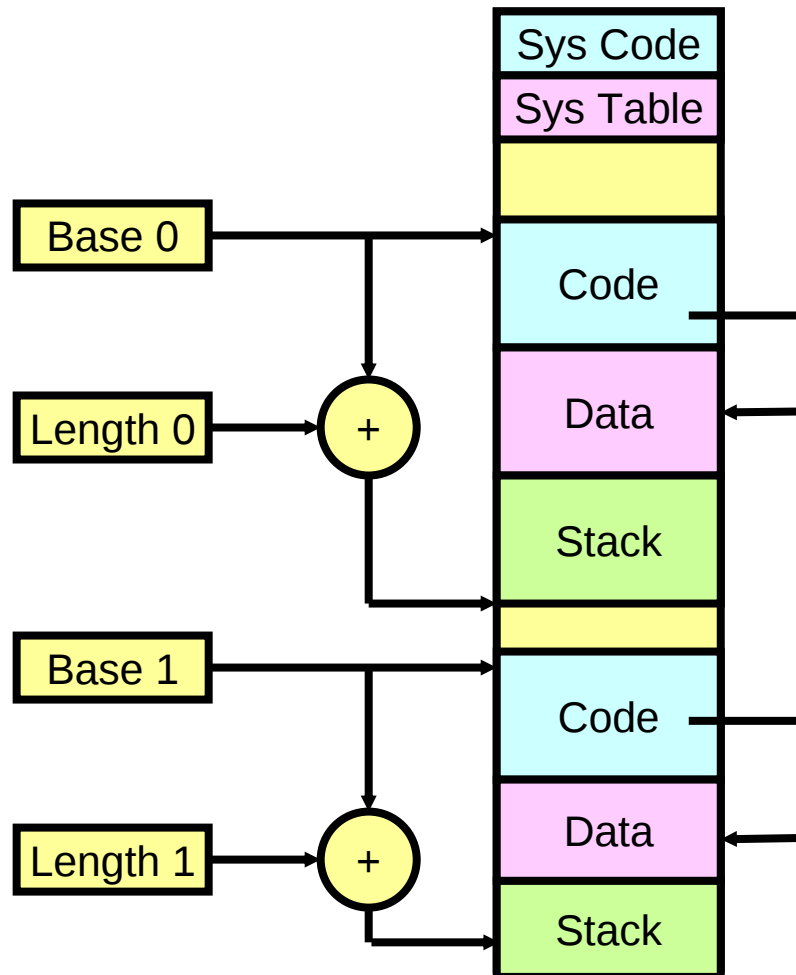


System code handles switching between programs

System table contains:

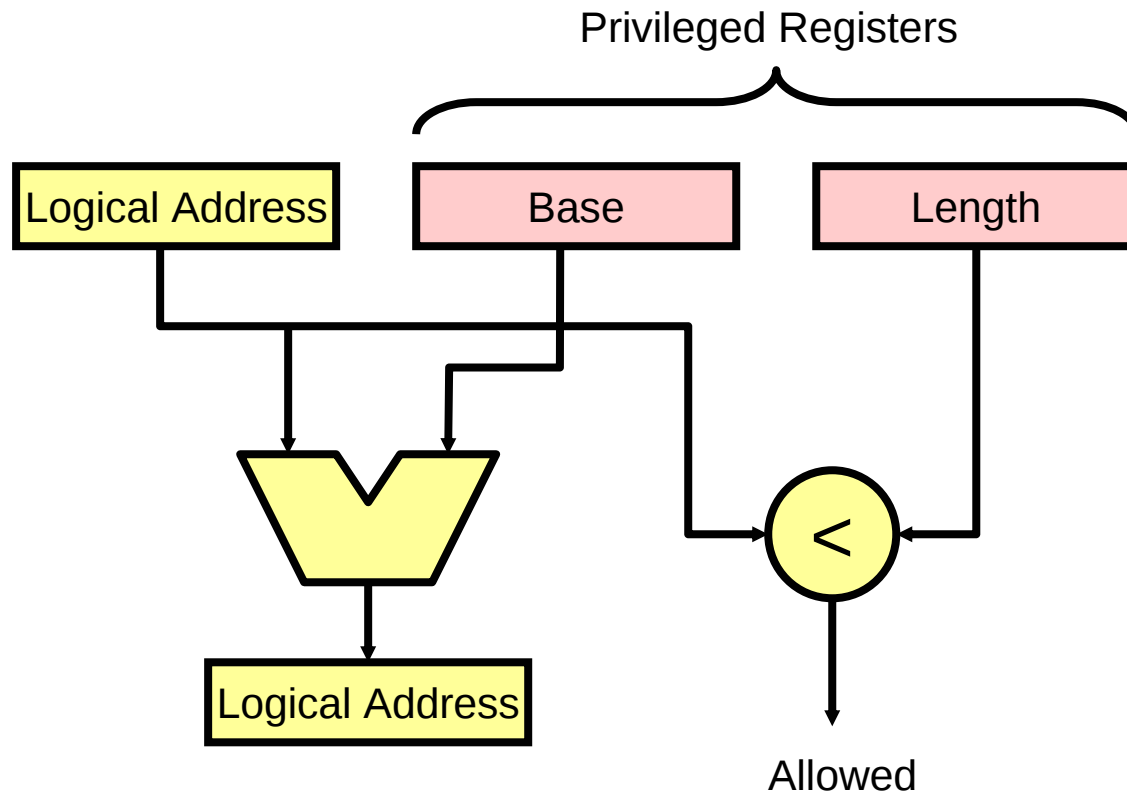
- Base address of each program
- Saved state of non-running programs

Providing Protection Between Programs (Length Registers)



- Add a *Length Register* LR to the hardware
- A program is only allowed to access memory from BR to BR+Length-1
- A program cannot set BR or LR
 - they are *privileged registers*
- But how do we switch programs?

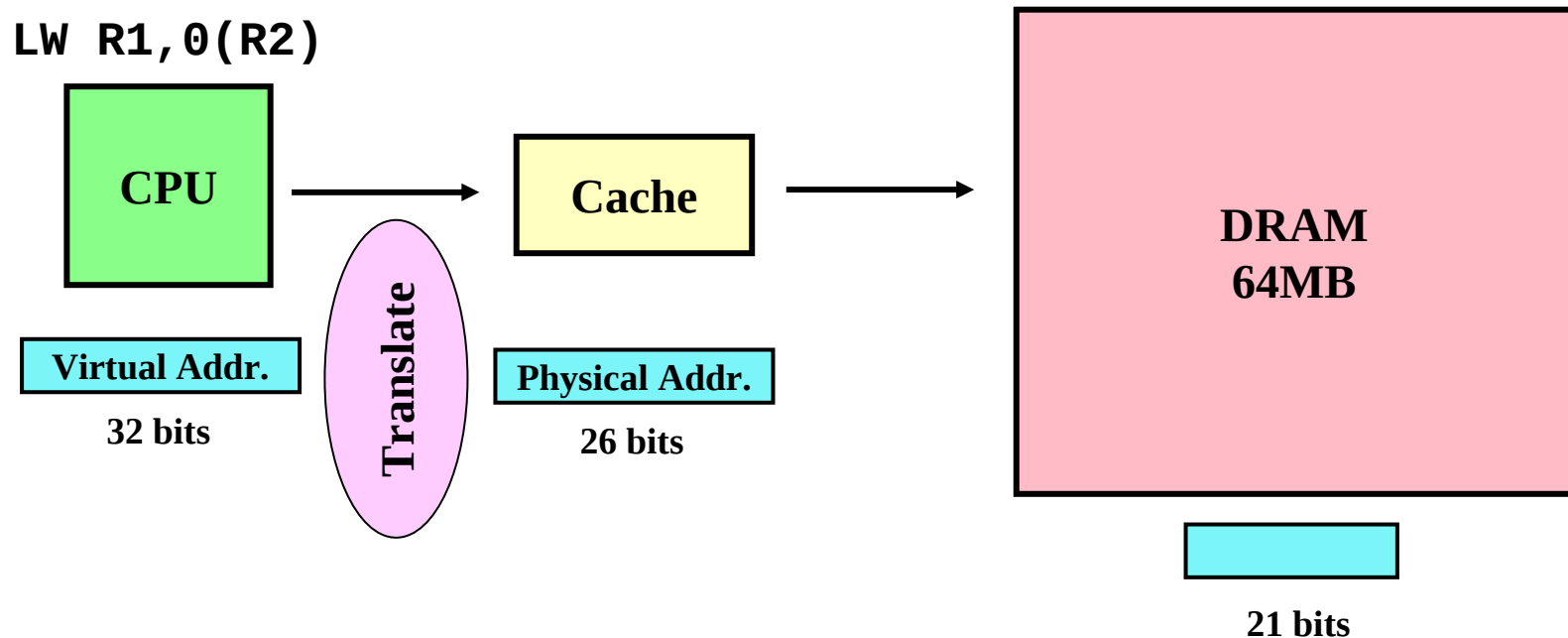
Base + Length Addressing



What a mess!

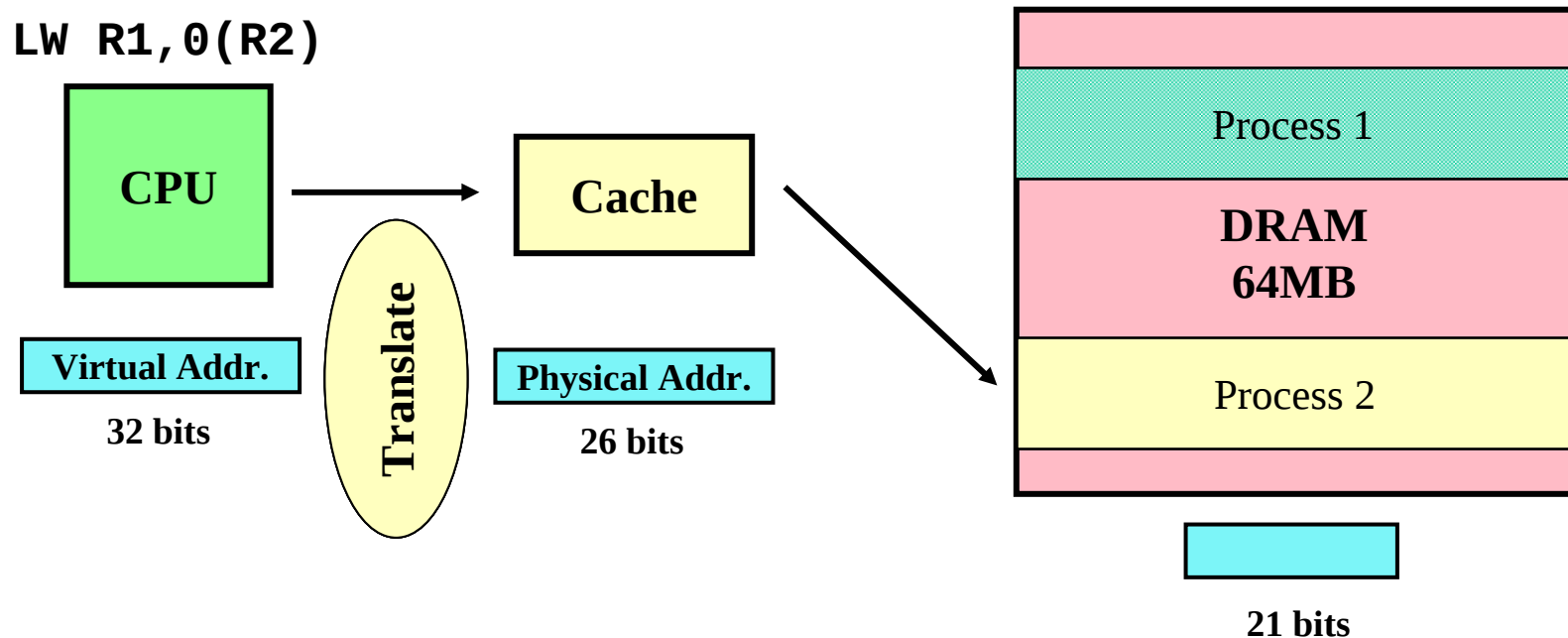
- Is there a better way that:
 - Simplifies protection
 - Enables relocation
 - Extends the physical memory capacity

A Load to Virtual Memory



- Translate from virtual space to physical space
 - $VA \Rightarrow PA$
 - May need to go to disk

A Load to Virtual Memory

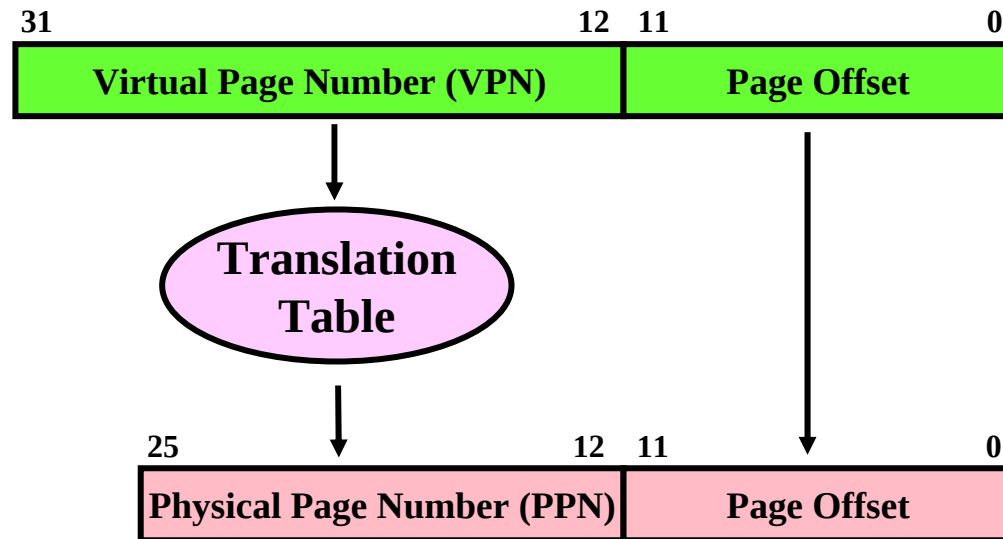


- Both programs can use the same set of addresses!
 - Change translation tables to point same VA to different PA for different programs

Paging and Protection

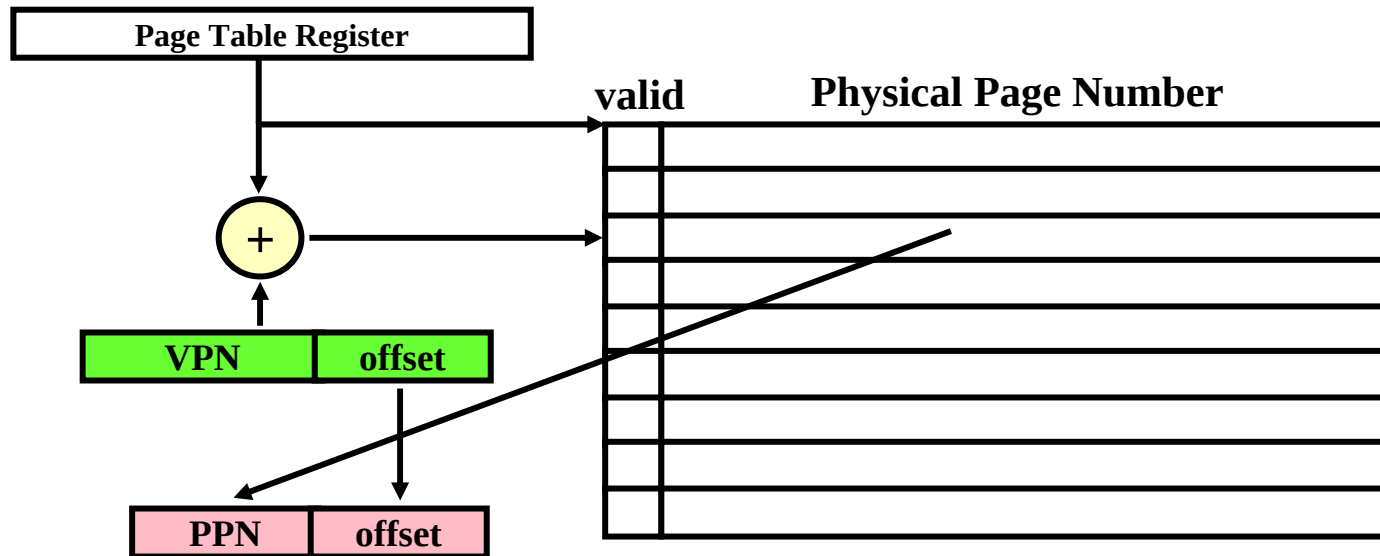
- How to ensure that processes can't access each other's data
 - Put them in separate virtual address spaces
 - Control the mappings of VA to PA for each process
 - Separate page tables
- How can you share data between processes
 - Give them each a VA mapping to the same PA
 - Similar entry in each process' page table

Virtual Address Translation



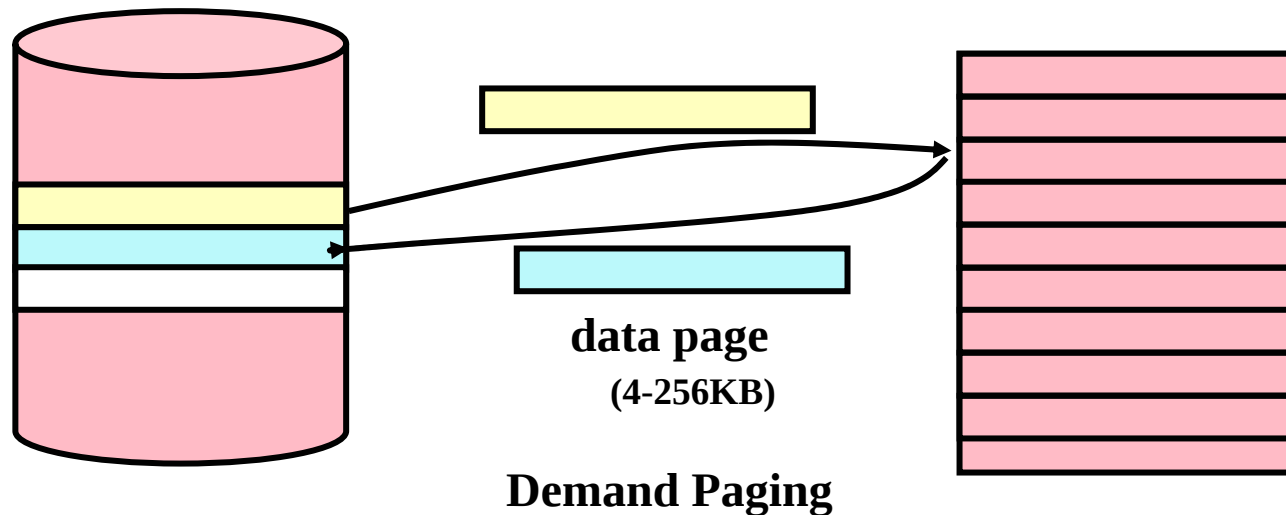
- Main Memory = 64MB
- Page Size = 4KB
- VPN = 20 bits
- PPN = 14 bits
- Translation table
 - aka “Page Table”

Page Table Construction



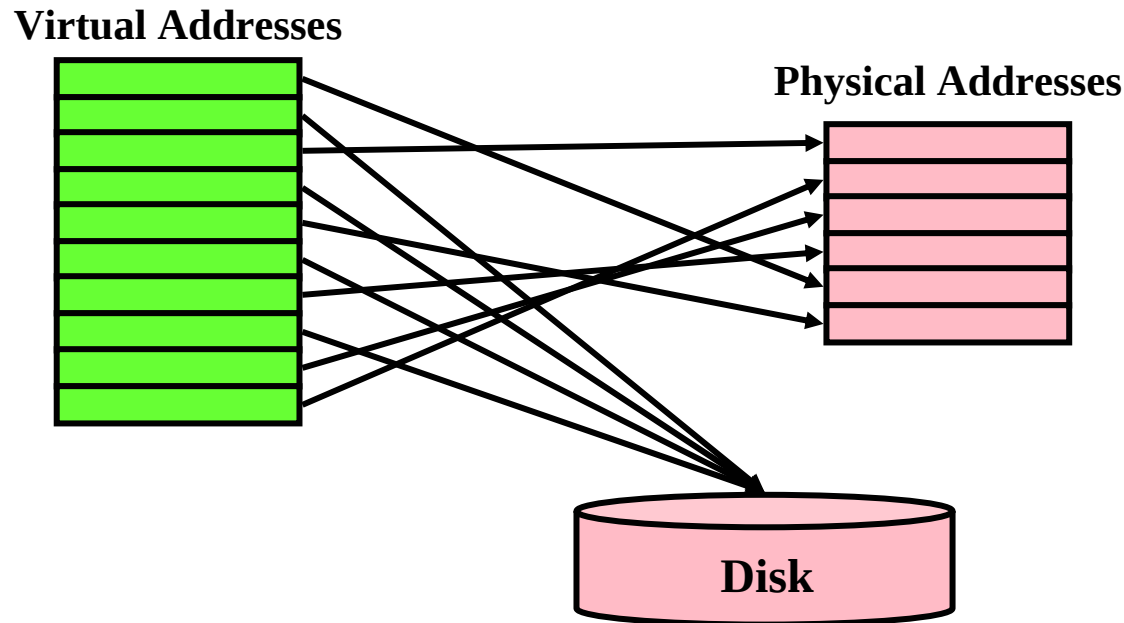
- Page table size
 - $(14 + 1) * 2^{20} = 4\text{MB}$
- Where to put the page table?

Paging: Main Memory as a Cache for Disk



- 32 bit addresses = 4GB, Main Memory = 64MB
- Dynamically adjust what data stays in main memory
 - Page similar to cache block
- Note: file system >> 4GB, managed by O/S

Virtual Addresses Span Memory+Disk

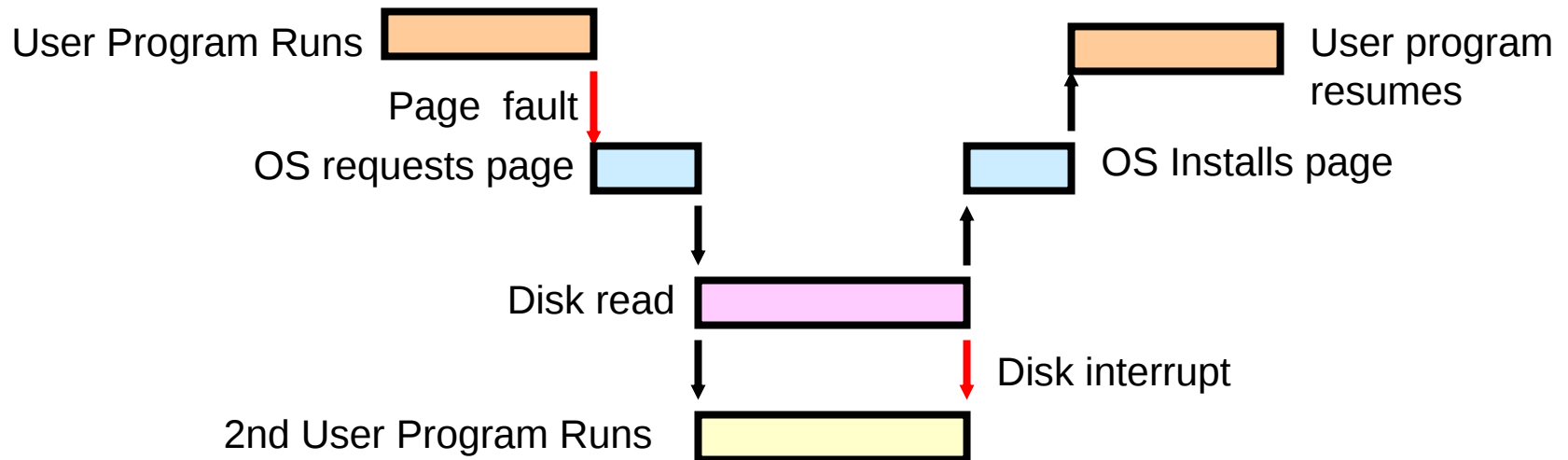


- Mappings changed dynamically by O/S
 - In response to users data accesses
 - OS triggered by hardware

What if Data is Not in DRAM?

- 1) Examine page table
- 2) Discover that no mapping exists
- 3) Select page to evict, store back to disk
- 4) Bring in new page from disk
- 5) Update page table

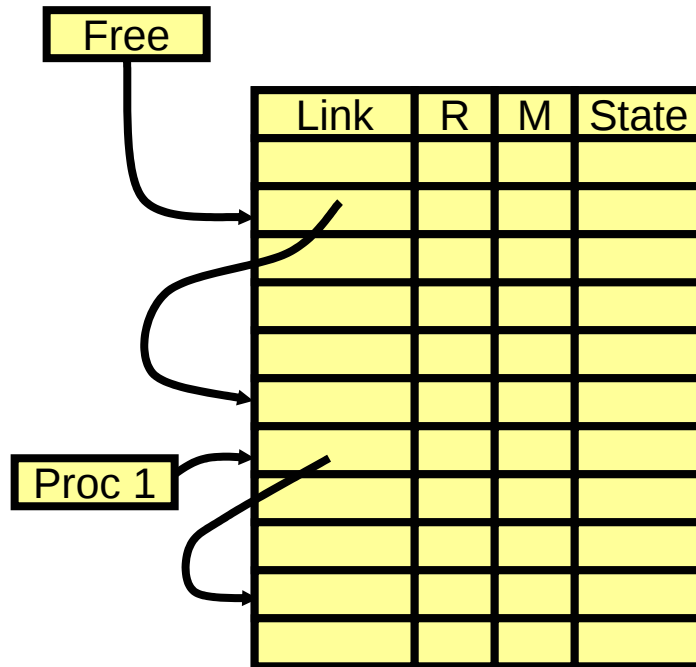
Page Fault



VM Requires

- Restartable (or resumable) instructions
 - must be able to resume program after recovering from a page fault
- Ability to mark a page *not present*
 - and raise a page fault when referencing such a page
- (Optional) Maintain status bits per page
 - R - referenced - for use by replacement algorithm
 - M - modified - to determine when page is *dirty*

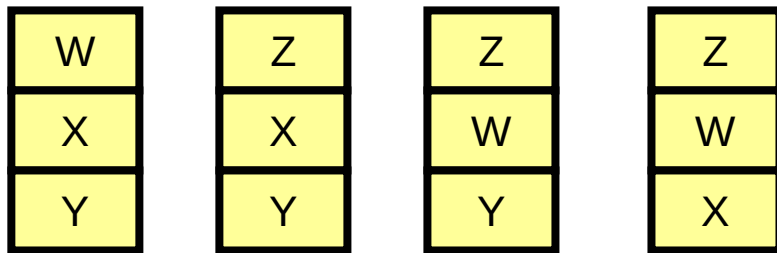
Page Frame Management



Page Frame Table

- OS maintains
 - **page table** for each user process
 - **page frame table**
 - free page list
 - pages evicted when number of free pages falls below a *low water mark*.
 - pages evicted using a *replacement policy*
 - random, FIFO, LRU, *clock*
 - if M-bit is clear, need not copy the page back to disk

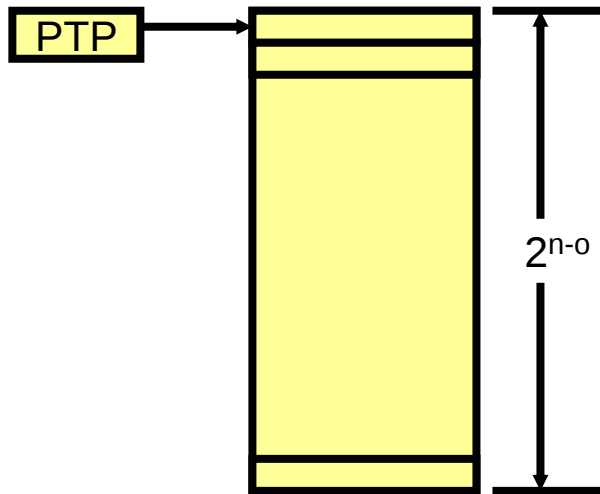
Page Management and Thrashing



Reference four pages in sequence, mapped to three page frames

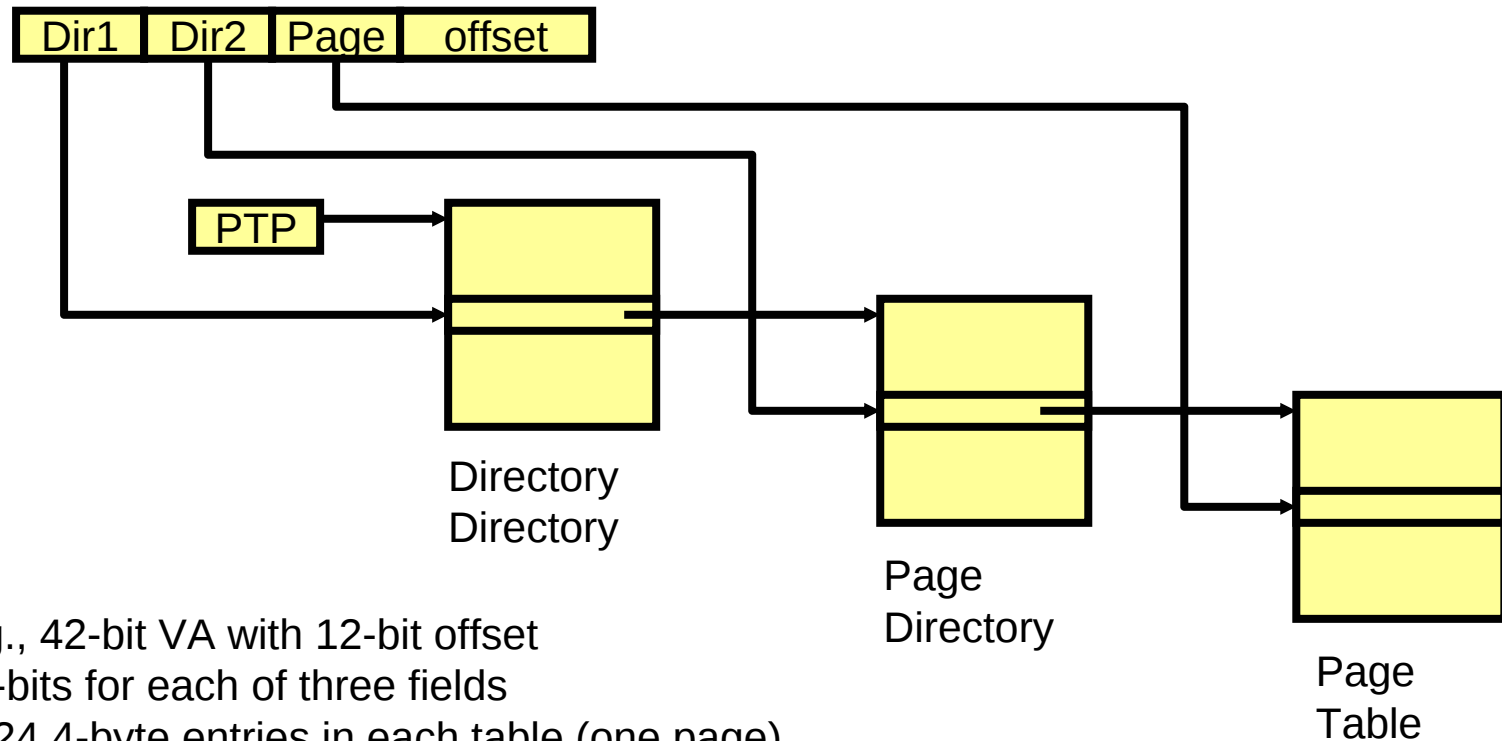
- Need to keep a process' *working set* in memory or *thrashing* will occur
- Find working set size by increasing page frame allocation until PF/s falls below limit
- Swap out whole process if insufficient page frames for working set

Page Table Organization

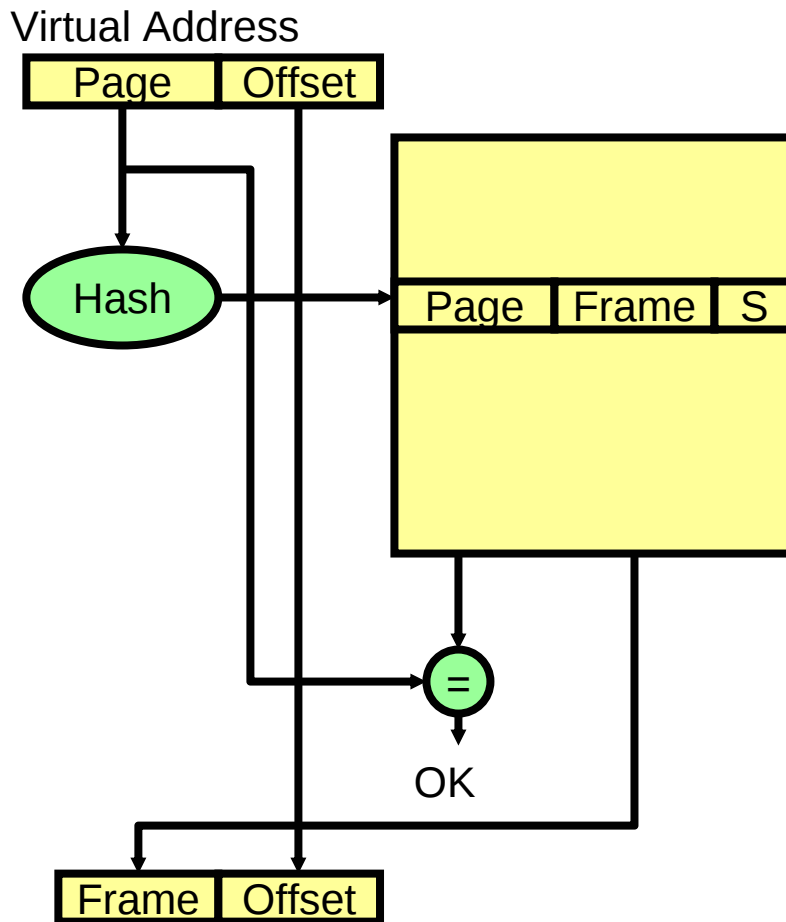


- Flat page table has size proportional to size of *virtual* address space
 - can be very large for a machine with 64-bit addresses and several processes
- Three solutions
 - page the page table (fixed mapping)
 - what really needs to be *locked down*?
 - multi-level page table (lower levels paged - Tree)
 - inverted page table (hash table)

Multi-Level Page Table



Inverted Page Tables



- Store only PTEs for pages *in* physical memory
- Miss in page table implies page is on disk
- Need KP entries for P page frames (usually $K > 2$)

Summary

- Virtual memory provides
 - Illusion of private memory system for each process
 - Protection
 - Relocation in memory system
 - Demand paging
- But – page tables can be large
 - Motivates: paging page tables, multi-level tables, inverted page tables
- Next time
 - Integration of virtual memory into cache hierarchy
 - DRAM memory organization