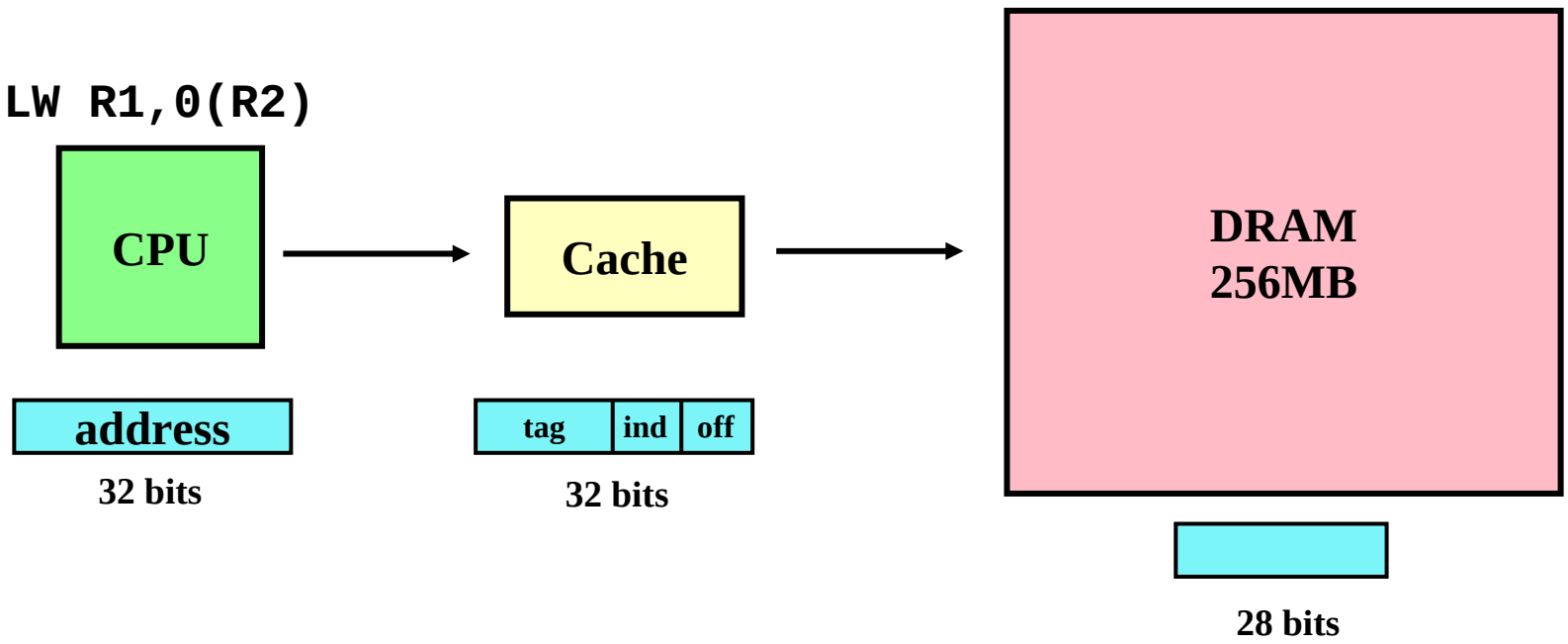
Lecture 16: Virtual Memory

- Last Lecture:
  - Introduction to virtual memory
- Today
  - Review and continue virtual memory discussion
- Administrivia
  - Pick up exam if you didn't
  - Homework averages
  - I'll post next homework online later today – 12 days

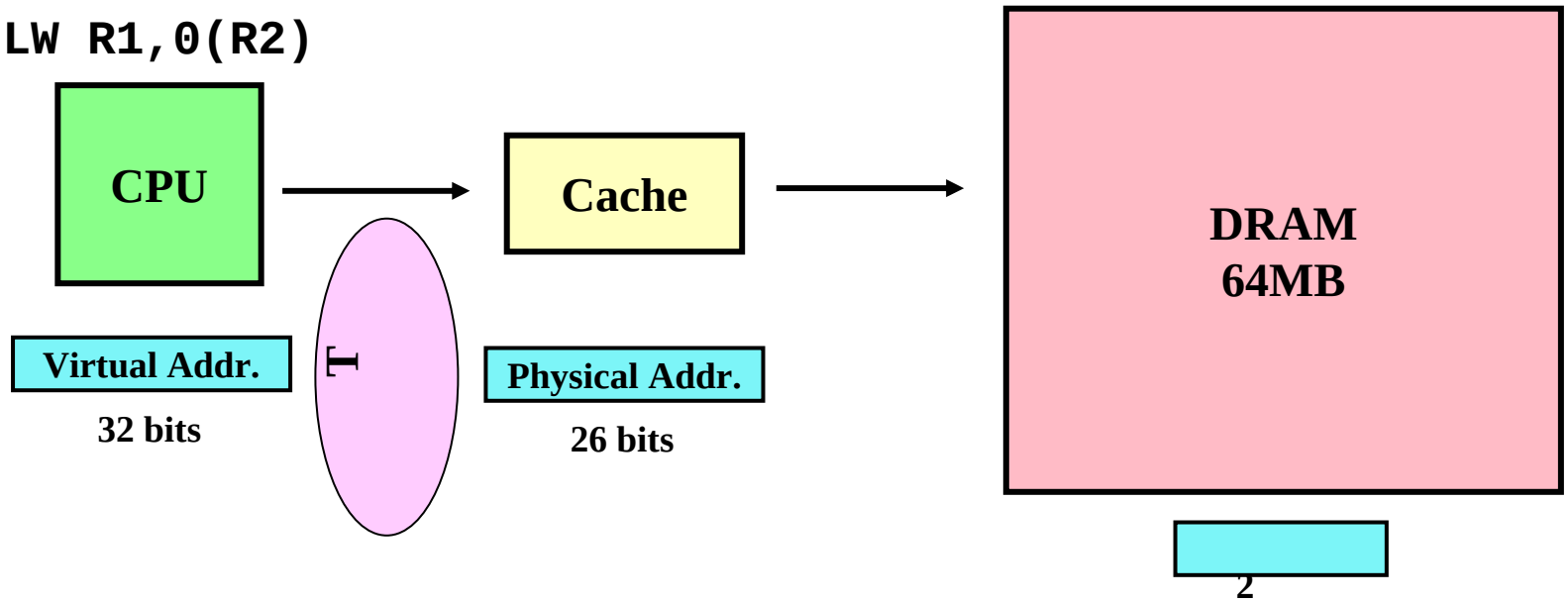
Physical Memory Addressing



The goal of virtual memory

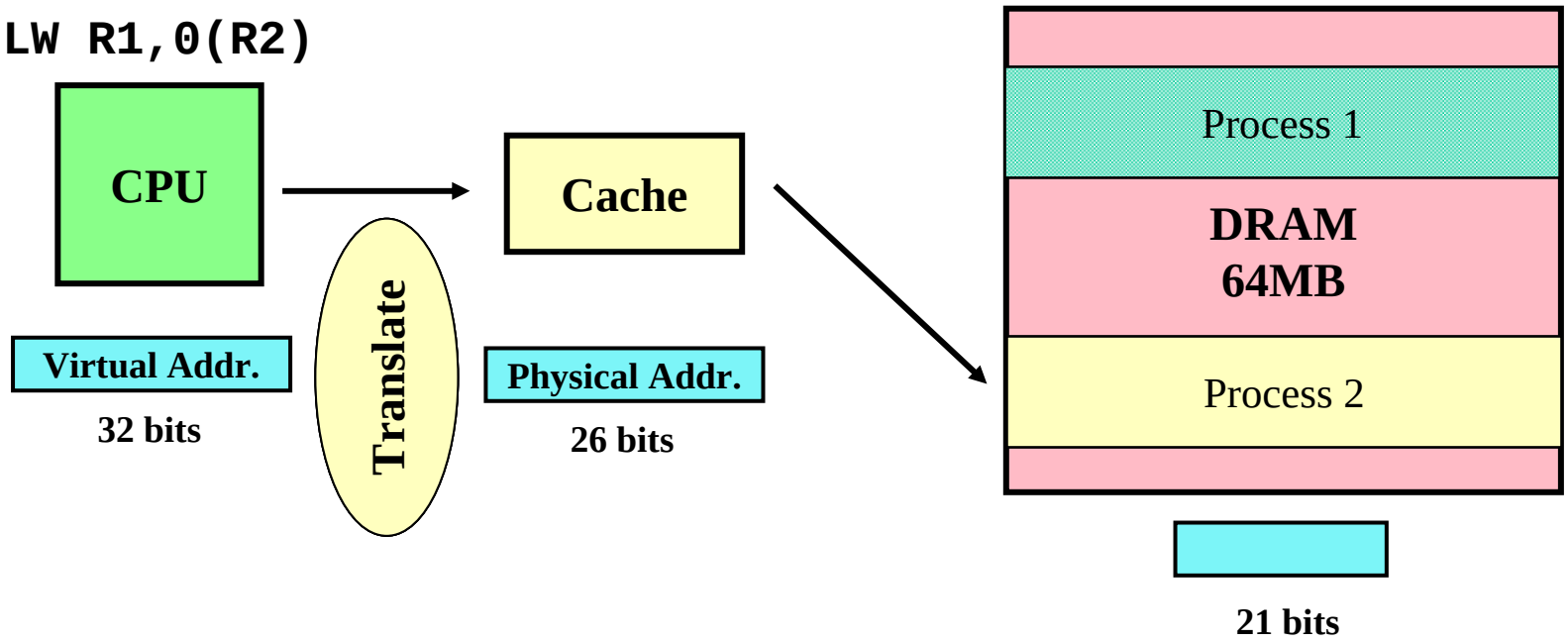
- Make it appear as if each process has:
  - Its own private memory
  - The memory is nearly infinite in size
- The challenge... Physical memory is:
  - Limited in size
  - Shared by all of the processes running on the machine
- The job of the virtual memory system is to maintain the illusion we want,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

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

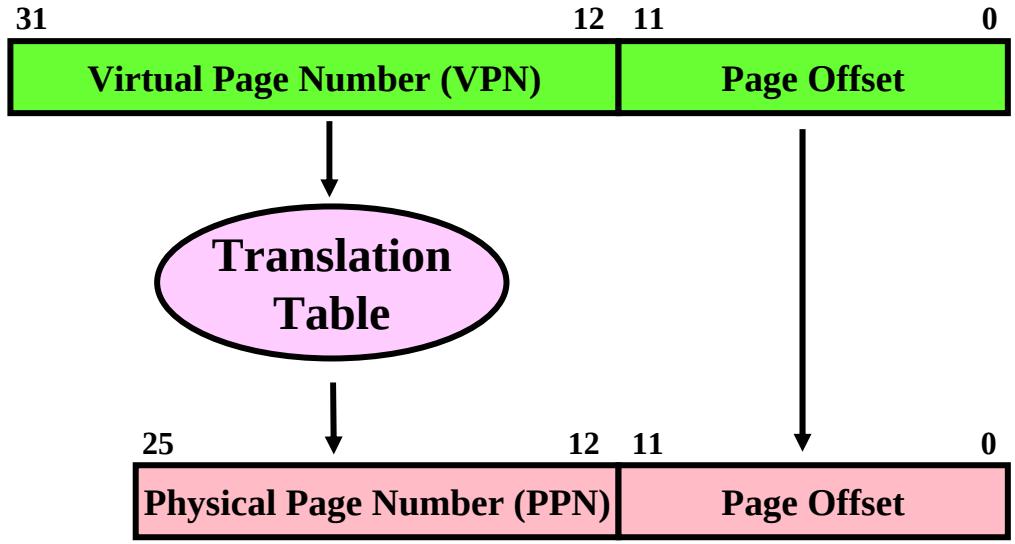


- Both programs can use the same set of addresses!
  - Change translation tables to point same VA to different PA for different programs

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

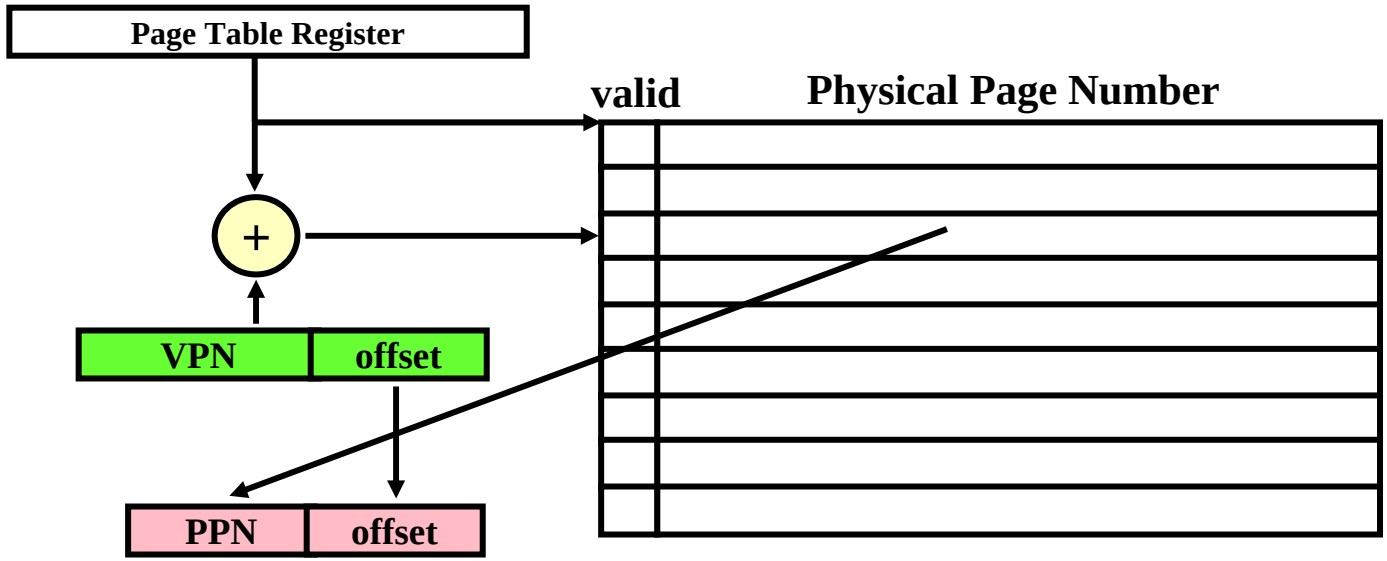
Virtual Address Translation



- Main Memory = 64MB
- Page Size = 4KB
- VPN = 20 bits
- PPN = 14 bits

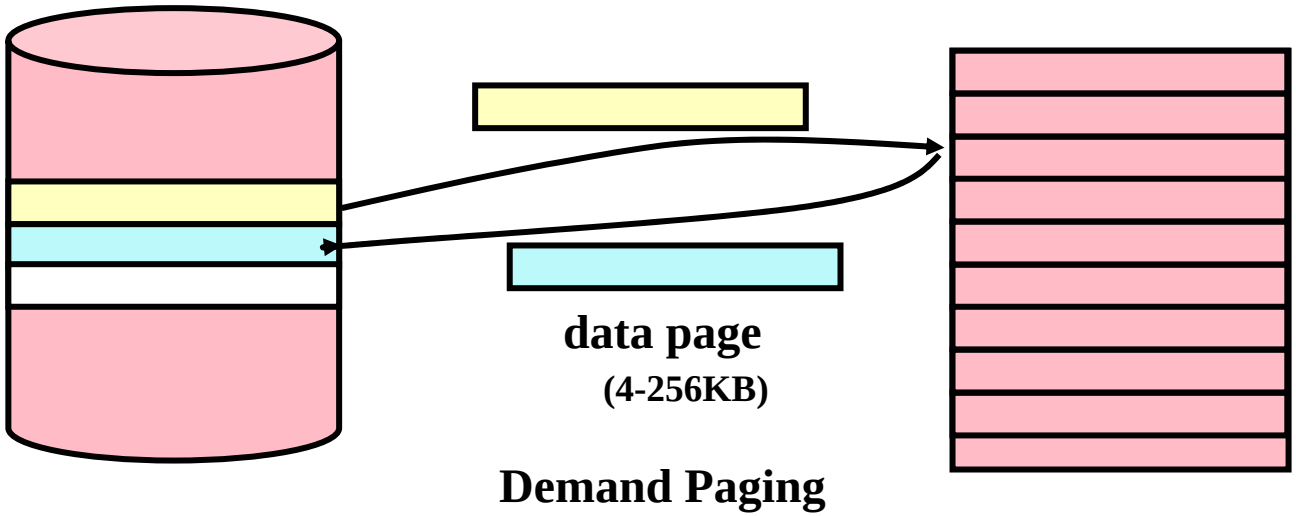
- Translation table
  - aka “Page Table”
  - An “array of pointers”

Page Table Construction



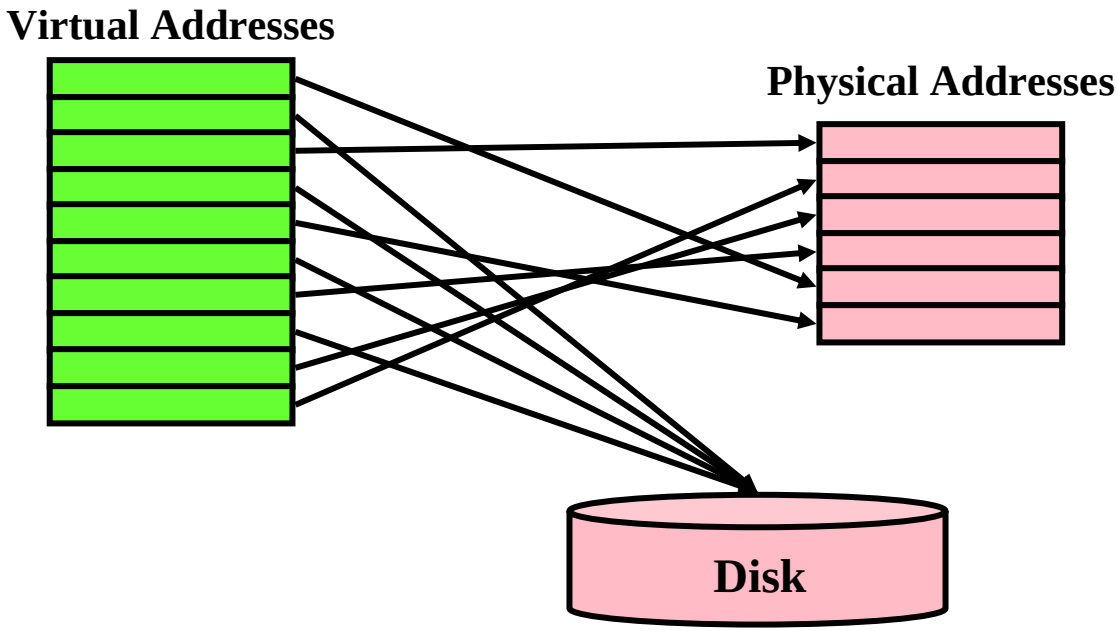
- Page table size
  - (14 + 1) * 2^{20} = 4MB
- Where to put the page table?

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

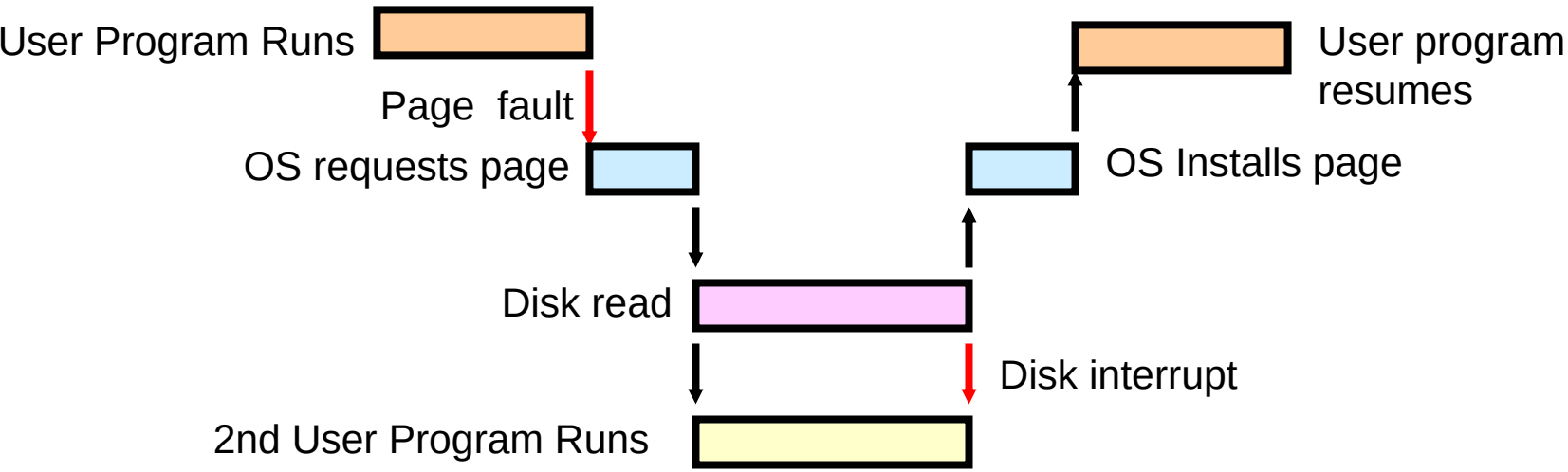
2) Discover that no mapping exists

3) Select page to evict, store back to disk

4) Bring in new page from disk

5) Update page table

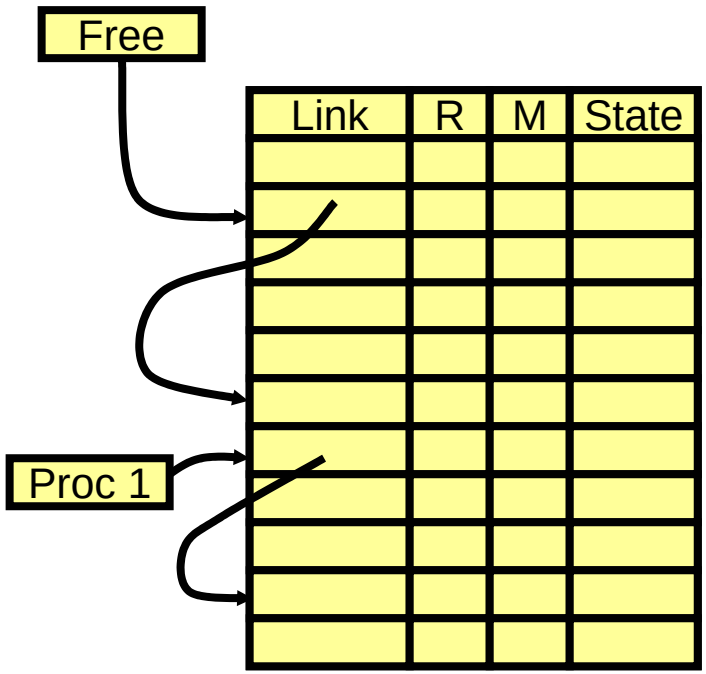
Page Fault


User Program Runs
Page fault
OS requests page
Disk read
2nd User Program Runs
Disk interrupt
OS Installs page
User program resumes

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*

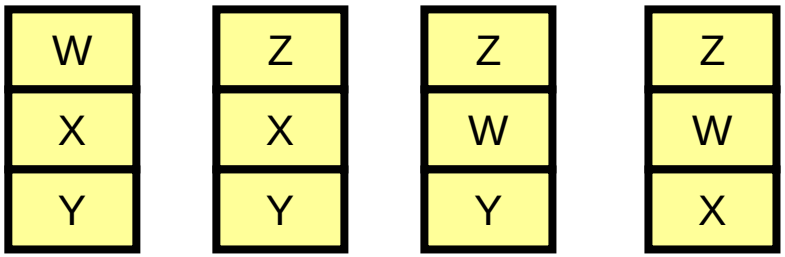
Page Frame Management



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

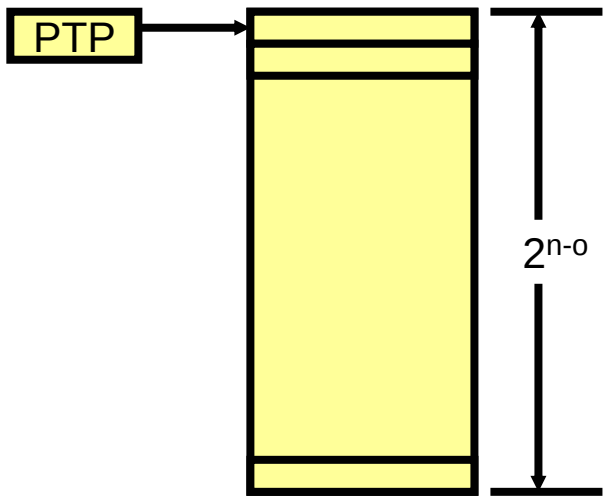
Page Management and Thrashing



Reference four pages in sequence, mapped to three page frames

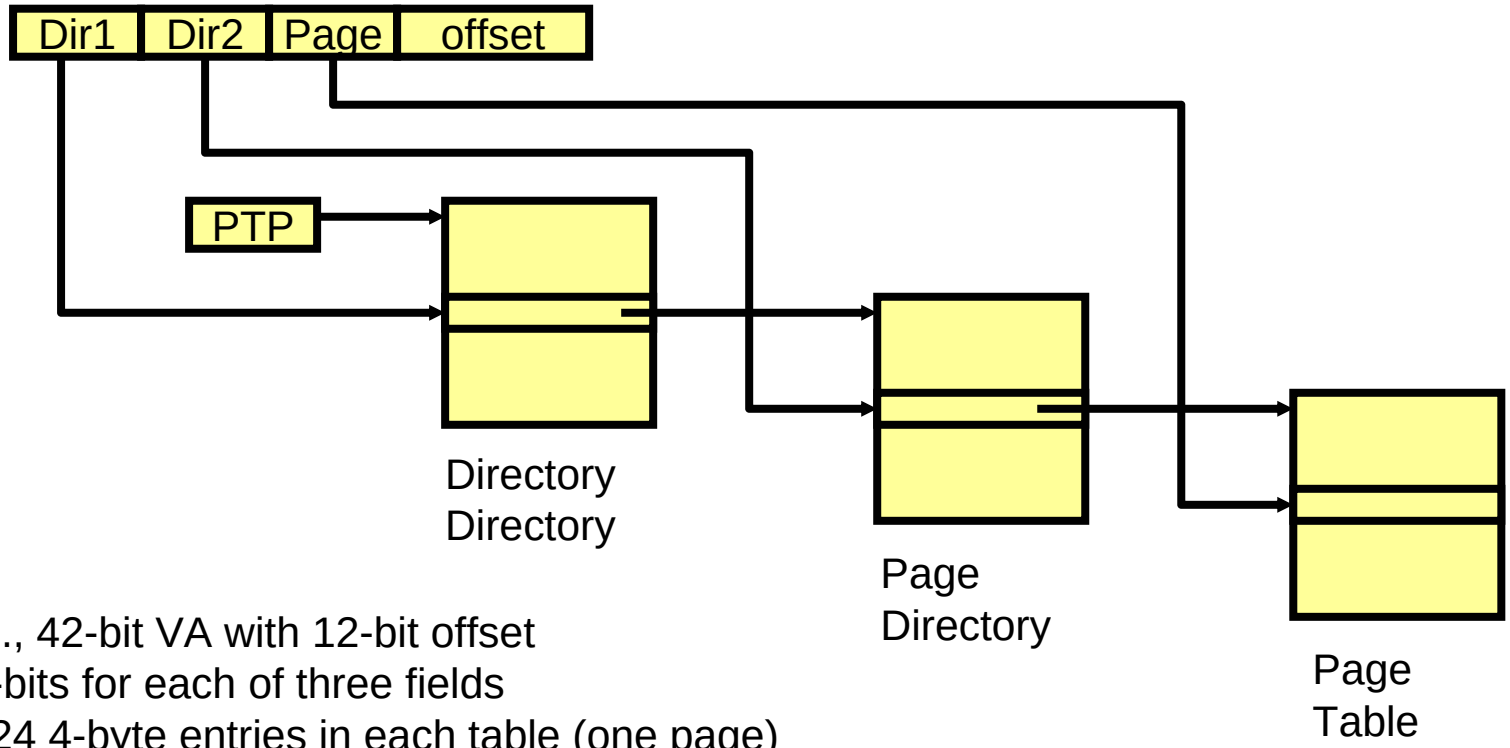
- Need to keep a process' *working set* in memory or *thrashing* will occur
- Find working set size by increasing page frame allocation until PF/s falls below limit
- Swap out whole process if insufficient page frames for working set

Page Table Organization



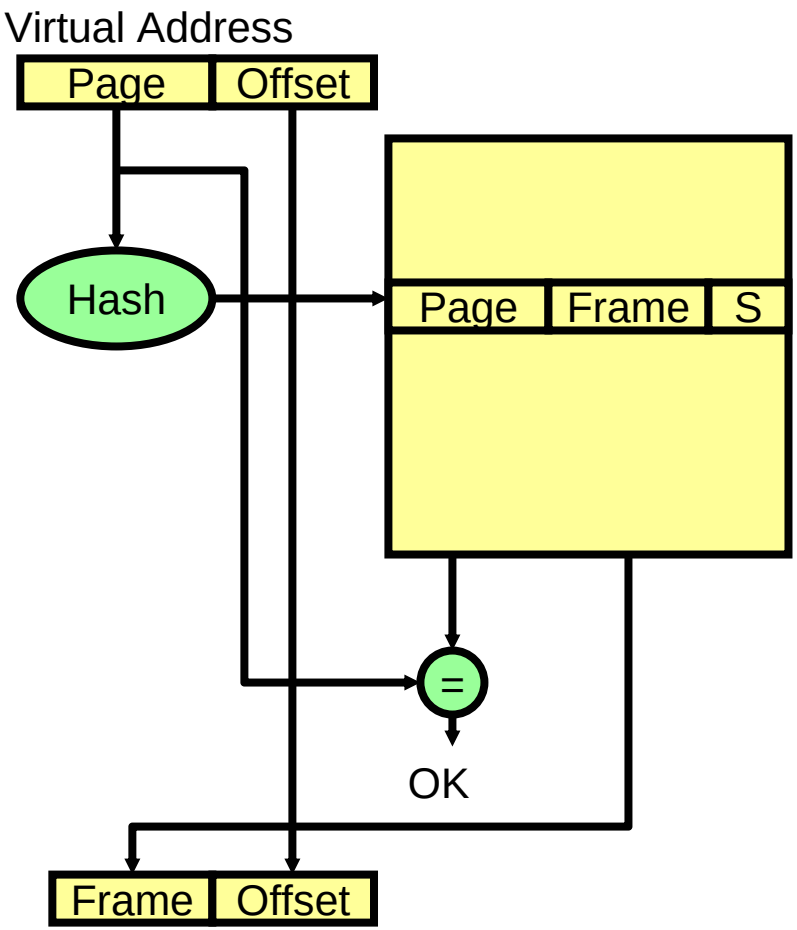
- Flat page table has size proportional to size of *virtual* address space
  - can be very large for a machine with 64-bit addresses and several processes
- Three solutions
  - page the page table (fixed mapping)
    - what really needs to be *locked down?*
  - multi-level page table (lower levels paged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

Multi-Level Page Table



e.g., 42-bit VA with 12-bit offset
10-bits for each of three fields
1024 4-byte entries in each table (one page)

Inverted Page Tables

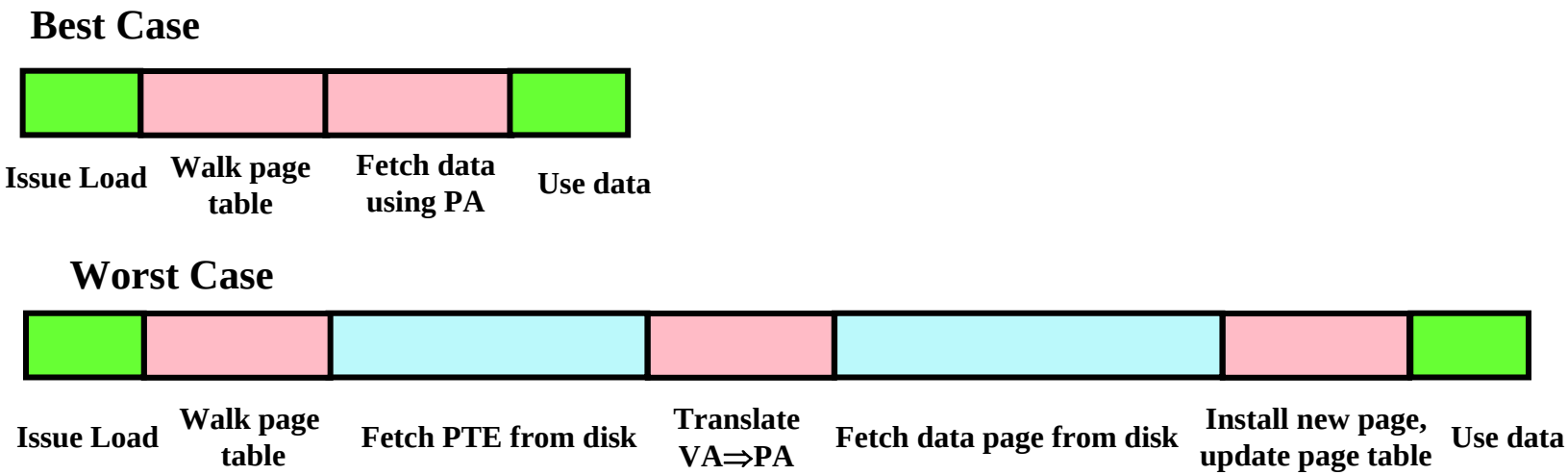


- Store only PTEs for pages *in* physical memory
- Miss in page table implies page is on disk
- Need KP entries for P page frames (usually K > 2)

Summary

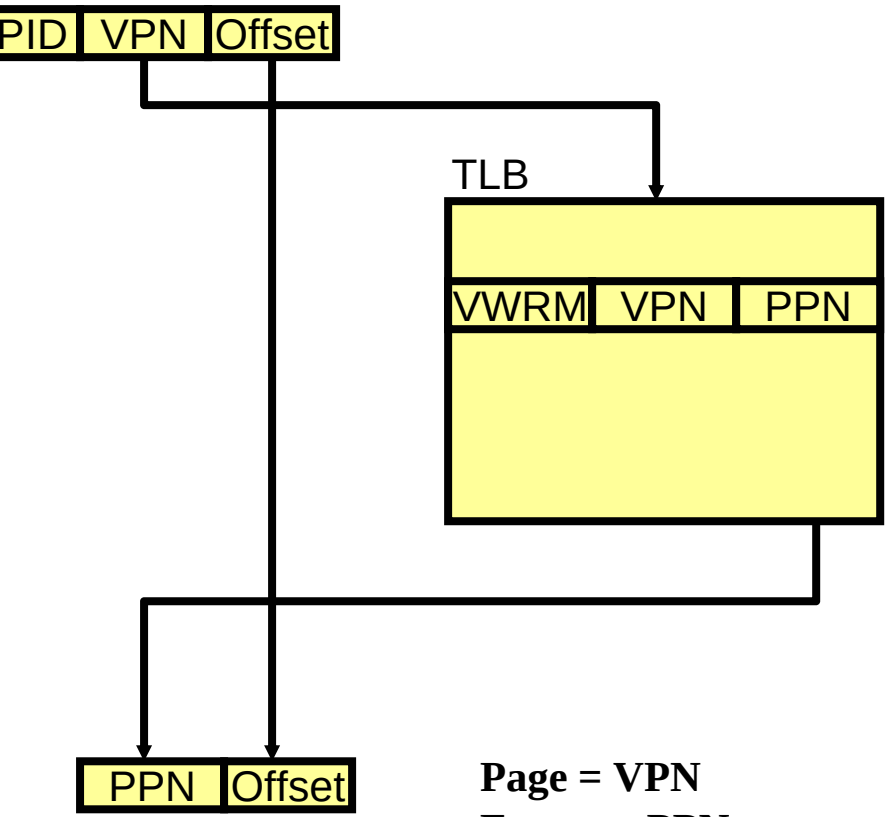
- Virtual memory provides
  - Illusion of private memory system for each process
  - Protection
  - Relocation in memory system
  - Demand paging
- But – page tables can be large
  - Motivates: paging page tables, multi-level tables, inverted page tables
- Next time
  - Integration of virtual memory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

How Long does it Take to Access VM?



- Problems
  - Multiple memory and potentially disk accesses
  - Where does the cache fit in?
- Let's accelerate this process!

Translation Lookaside Buffers



- Store most frequently used translations in small, fast memory (cache for page table entries)
- Valid, Writeable, Referenced, Modified
  - Access protection
  - Replacement strategies
- Size – often 128 entries
- Highly associative (sometimes fully assoc.)

“Rare” Behavior in VM system

- TLB Miss
  - Translation is not in TLB – but everything could be in memory
  - Two approaches
    - Hardware state machine *walks* the page table
      - fast but inflexible
    - Exception raised and software walks the page table
- Page Fault
  - Entry not in TLB and target page not in main memory
- Worst case
  - Page fault and page table and target page

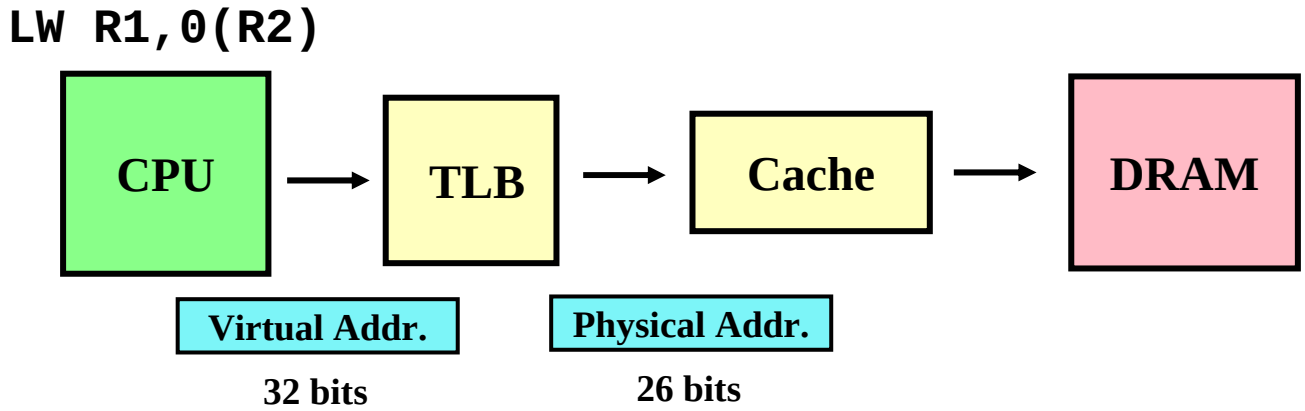
Reducing TLB misses

- Same type of optimizations as for cache
  - Associativity (many TLBs are fully associative)
  - Capacity – TLBs tend to be 32-128 entries
- Adjust page size
  - Small pages
    - Reduces internal fragmentation
    - Speeds page movement to/from disk
  - Large pages
    - Can cover more physical memory with same number of TLB entries
  - Solution – typically have a variable page size
    - Select by OS, 4KB-256KB (superpages)

Virtual Memory + Caching

- Conflicting demands:
  - Convenience of flexible memory management (translation)
  - Performance of memory hierarchy (caching)
- Requires cooperation of O/S
  - Data in cache implies that data is in main memory
- Combine VM and Caching
  - Where do we put the Cache and the TLB????

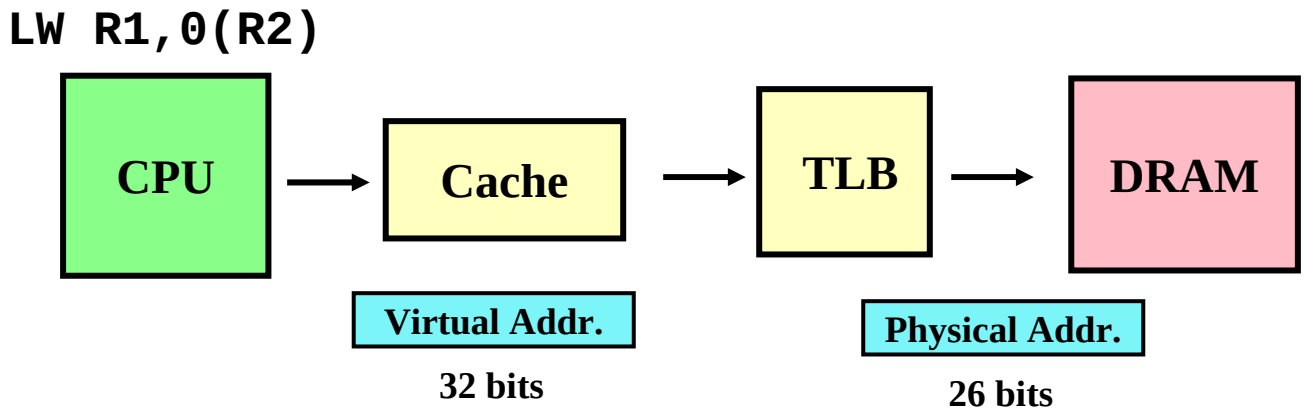
Physically Addressed Cache



- Translate first from VA $\Rightarrow$ PA
- Access cache with PA

- Problems?

Virtually Addressed Cache



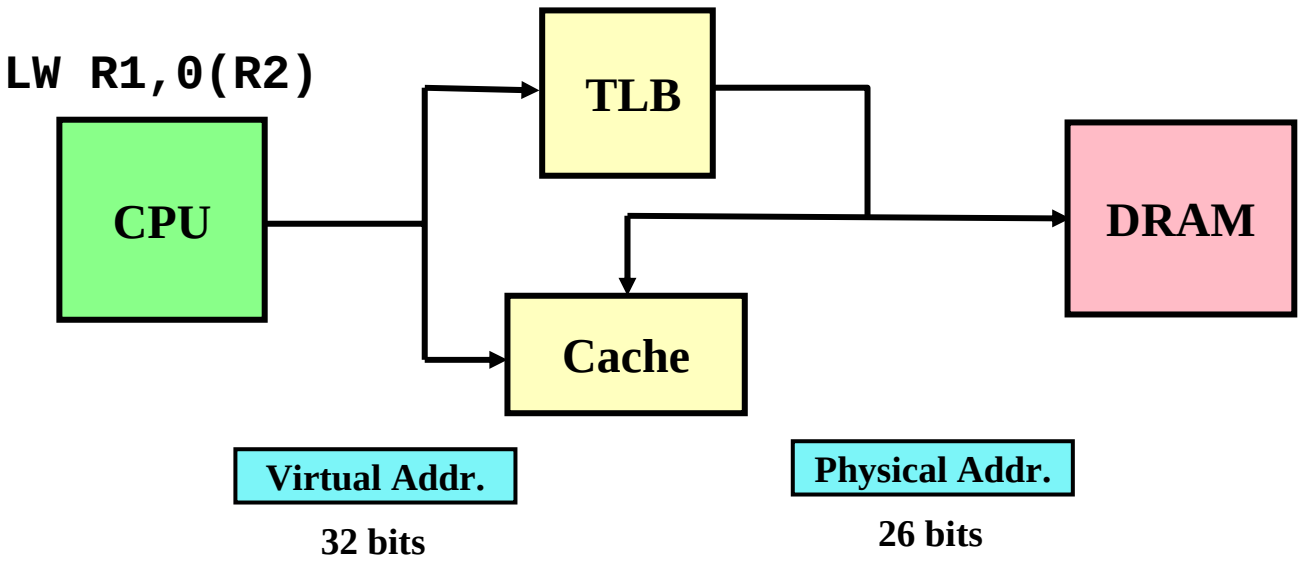
- Access cache first
- Only translate if going to main memory

- Problems?

Aliasing

- Can occur when switching among multiple address spaces
- Synonym aliasing
  - Different VAs point to the same PA
  - Occurs when data shared among multiple address spaces
  - One solution – always translate before going to the cache
- Homonym aliasing
  - Same VA point to different PAs
  - Occurs on context switching
  - Two solutions:
    - Flush TLB on process switch/system call
    - TLB includes process ID

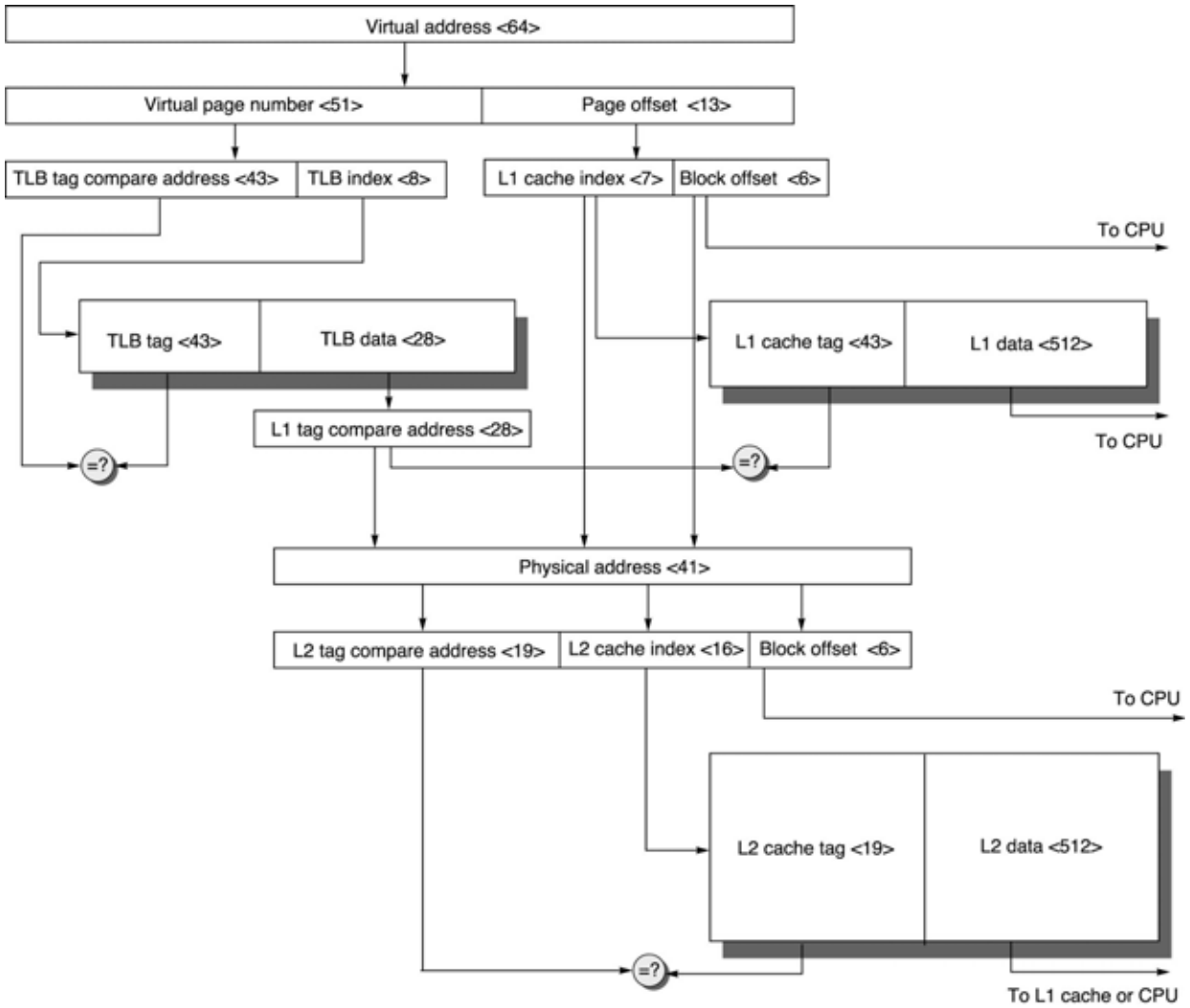
Best of Both Worlds:



Virtually addressed, Physically Tagged

- **Parallel Access**
- **Eliminate synonym problem**

Virtual Index, Physical Tag



Other Aliasing Solutions

- Note virtually indexed/physically tagged put constraints on cache capacity, page size, etc.
- Other solutions:
  - 21264: 8KB pages, 64KB i-cache, 2-way set associative
    - Aliases could reside in 8 different places in cache
    - On cache miss, invalidate any possible aliases in cache
  - Intel Pentium 4
    - Virtually indexed/virtually tagged cache
    - Check for TLB misses off line (roll back if necessary)

Virtual Memory Summary

- Relocation, Protection
- Access memory >> DRAM capacity

- Translation
  - From large VA space to smaller PA space
  - Page tables hold translations

- Provides
  - Separation of memory management from user programs
  - Ability to use DRAM as cache for disk
  - Fast translation using Translation Lookaside Buffer (TLB)
    - Cache for page table
  - Speed - translate in parallel with cache lookup