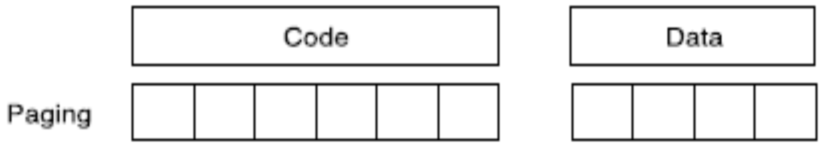
CS161 – Design and Architecture of Computer

Virtual Memory

UNIVERSITY OF CALIFORNIA, RIVERSIDE

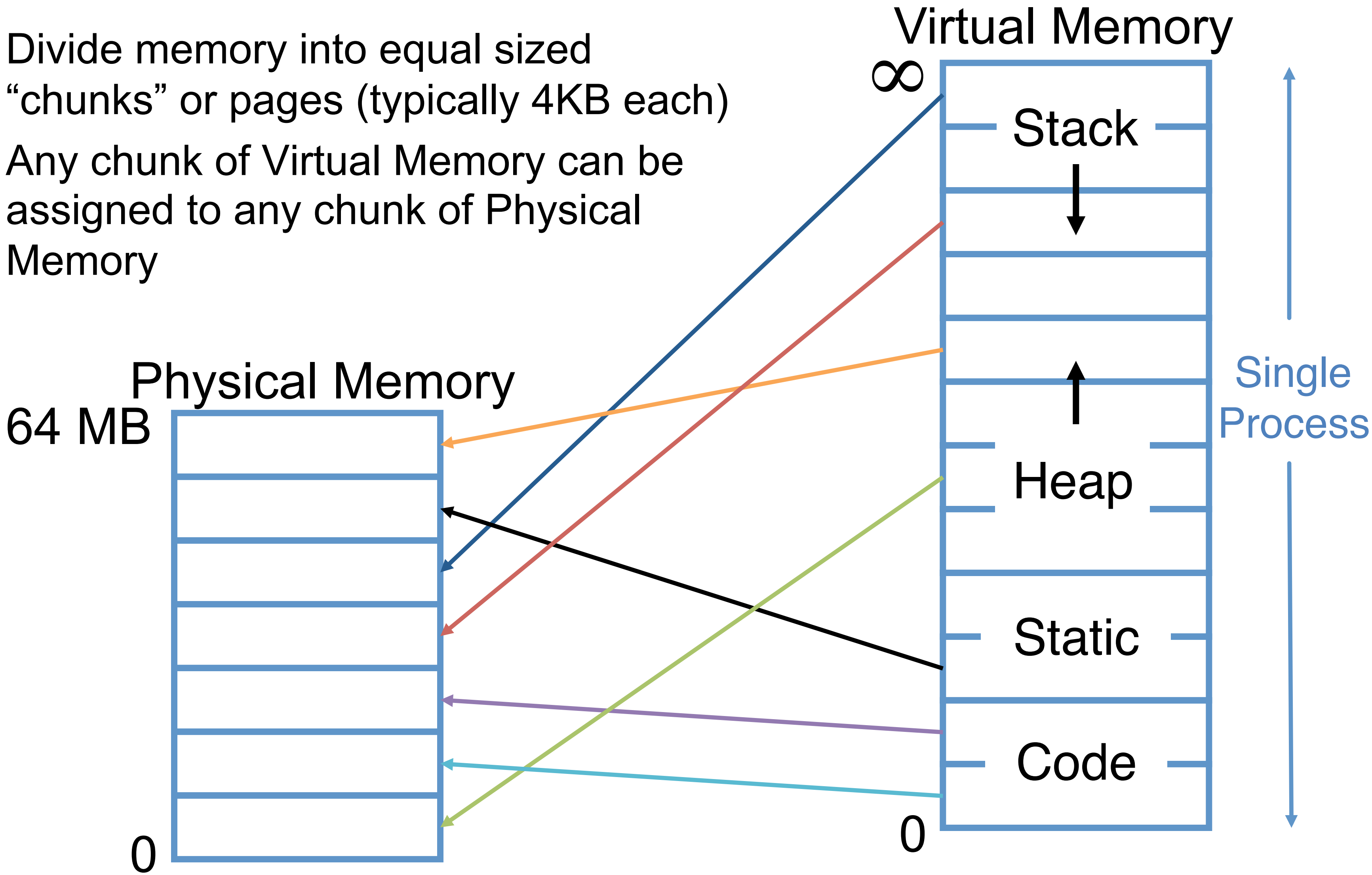
Why Virtual memory?

- Allows applications to be bigger than main memory size
- Helps with multiple process management
  - Each process gets its own chunk of memory
  - Protection of processes against each other
  - Mapping of multiple processes to memory
  - Relocation
  - Application and CPU run in virtual space
  - Mapping of virtual to physical space is invisible to the application
- Management between main memory and disk
  - Miss in main memory is a page fault or address fault
  - Block is a page

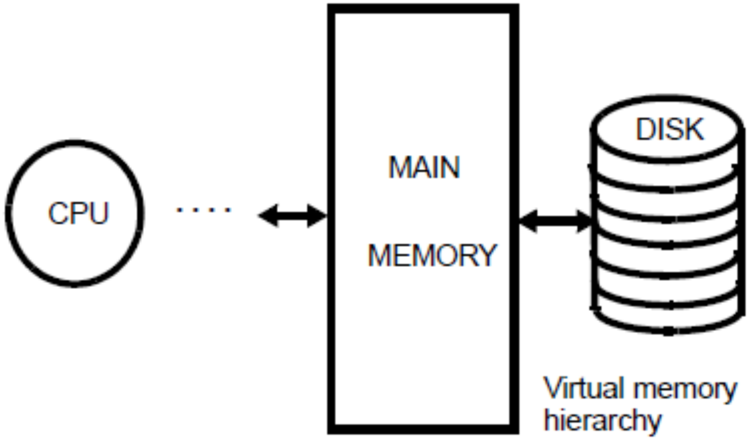


Mapping Virtual to Physical Memory

- Divide memory into equal sized "chunks" or pages (typically 4KB each)
- Any chunk of Virtual Memory can be assigned to any chunk of Physical Memory



Paged Virtual Memory



- Virtual address space divided into pages
- Physical address space divided into pageframes
- Page missing in Main Memory = page fault
  - Pages not in Main Memory are on disk: swap-in/swap-out
  - Or have never been allocated
  - New page may be placed anywhere in MM (fully associative map)
- Dynamic address translation
  - Effective address is virtual
  - Must be translated to physical for every access
  - Virtual to physical translation through page table in Main Memory

Cache vs VM

Cache	*Virtual Memory*
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-16KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	LRU approximation
Write Thru or Back	Write Back
How Managed: Hardware	Hardware + Software (Operating System)

Handling Page Faults

- A page fault is like a cache miss
  - Must find page in lower level of hierarchy
- If valid bit is zero, the Physical Page Number points to a page on disk
- When OS starts new process, it creates space on disk for all the pages of the process, sets all valid bits in page table to zero, and all Physical Page Numbers to point to disk
  - called Demand Paging - pages of the process are loaded from disk only as needed
  - Create “swap” space for all virtual pages on disk

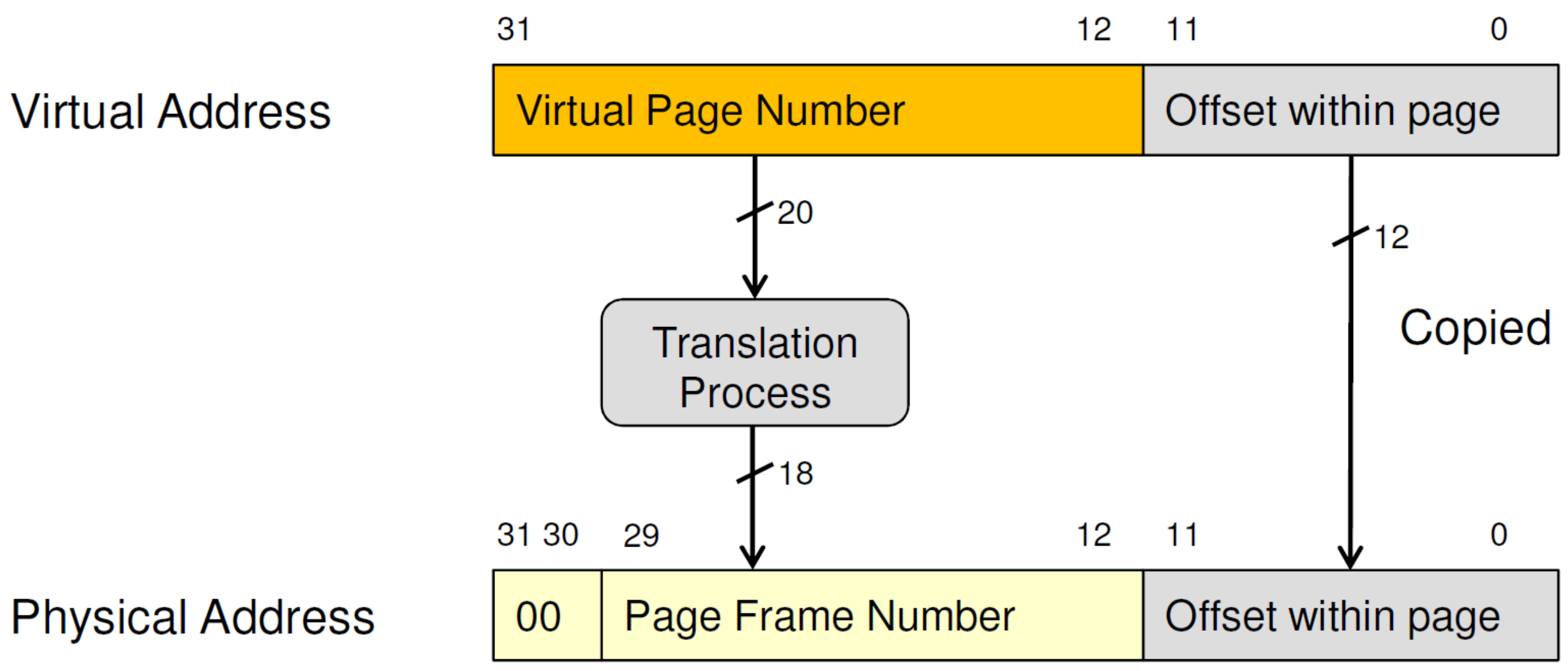
Performing Address Translation

- VM divides memory into equal sized pages
- Address translation relocates entire pages
  - offsets within the pages do not change
  - if page size is a power of two, the virtual address separates into two fields:
    (like cache index, offset fields)

Virtual Page Number	Page Offset

virtual address

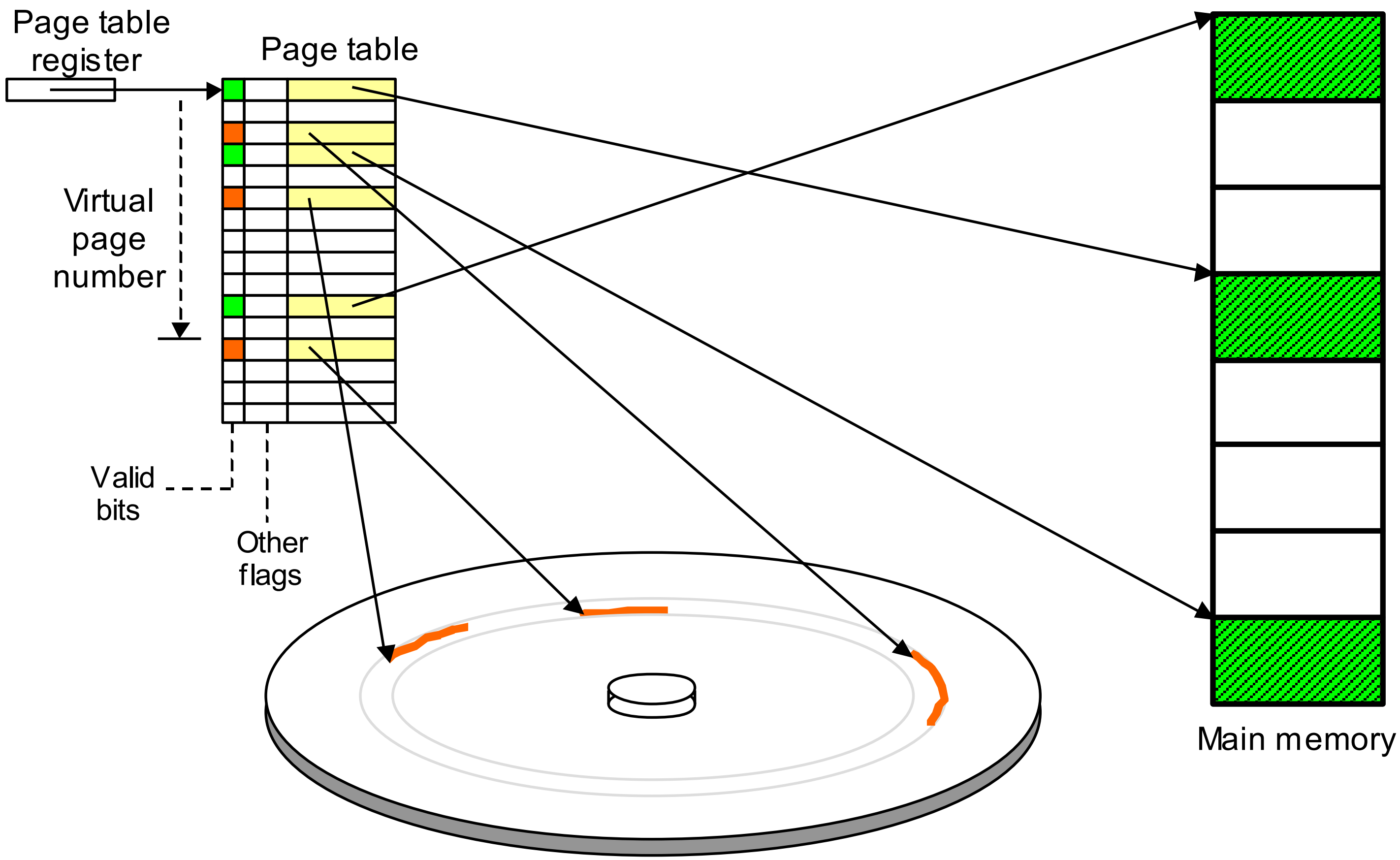
Mapping Virtual to Physical Address



Address Translation

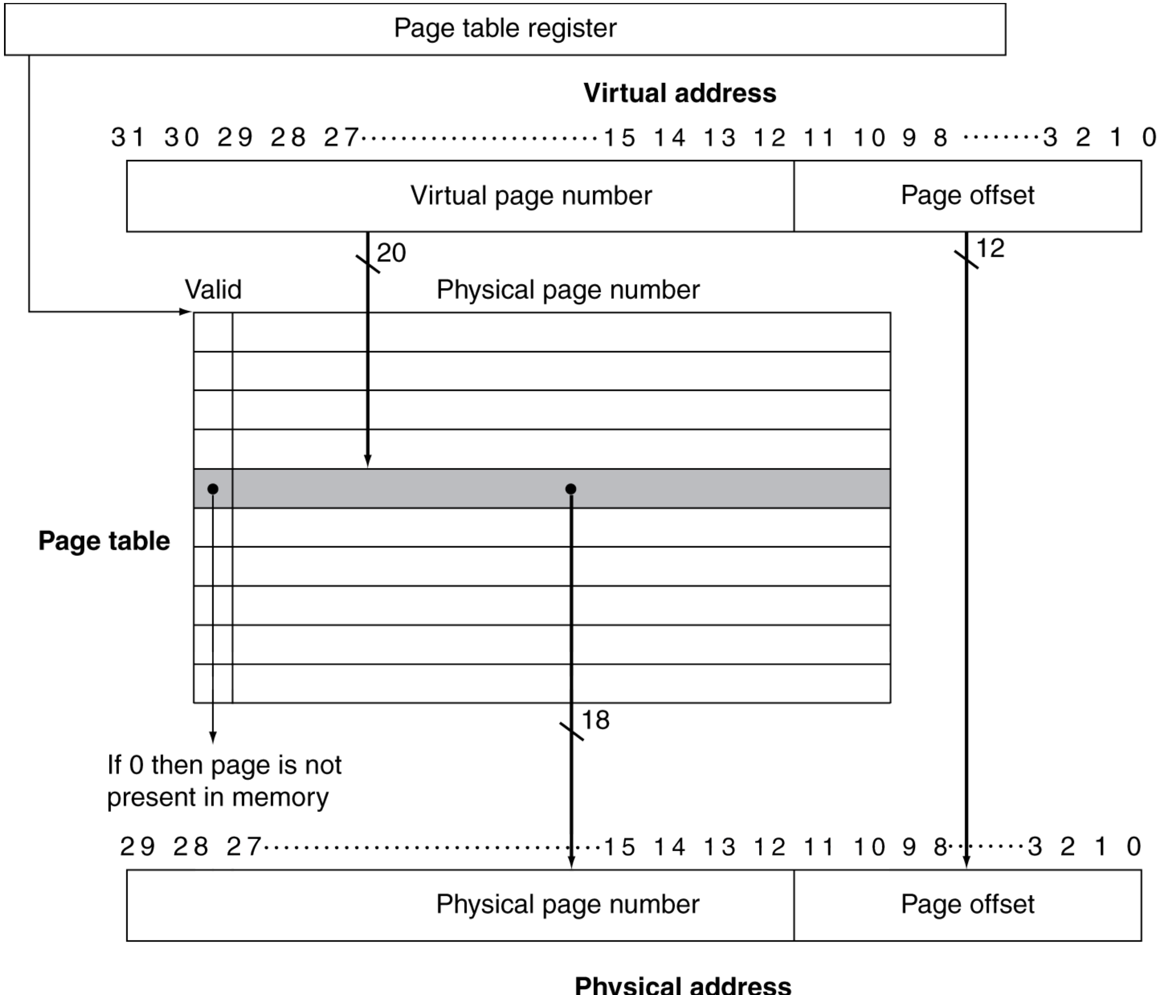
- Want fully associative page placement
- How to locate the physical page?
  - Search impractical (too many pages)
- A page table is a data structure which contains the mapping of virtual pages to physical pages
  - There are several different ways, all up to the operating system, to keep this data around
- Each process running in the system has its own page table

Page Table and Address Translation

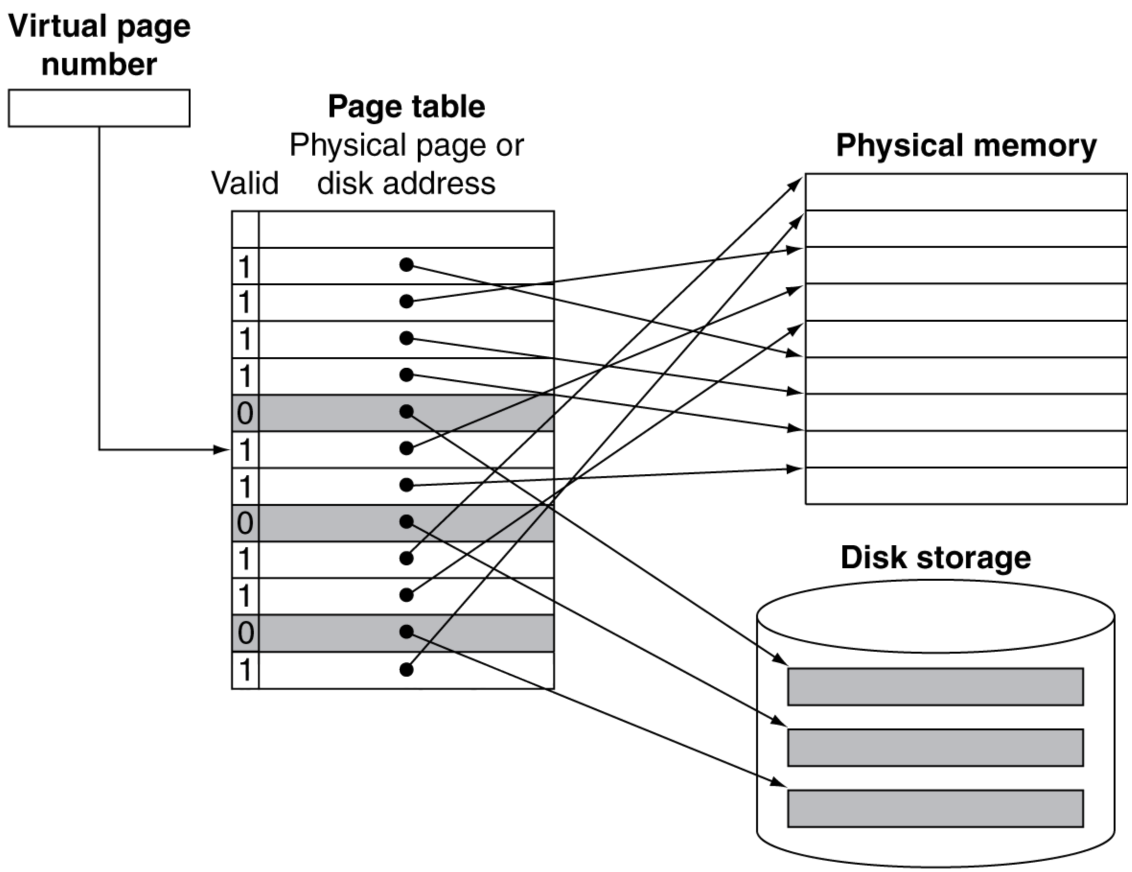


Page Table

- Page table translates address



Mapping Pages to Storage



Replacement and Writes

- To reduce page fault rate, prefer least-recently used (LRU) replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently
- Disk writes take millions of cycles
  - Block at once, not individual locations
  - Write through is impractical
  - Use write-back
  - Dirty bit in PTE set when page is written

Optimizing VM

- Page Table too big!
  - 4GB Virtual address space / 4 KB page
    - 2^{20} page table entries. Assume 4B per entry.
    - 4MB just for Page Table of single process
  - With 100 process, 400MB of memory is required!

- Virtual Memory too slow!
  - Requires two memory accesses.
    - One to access page table to get the memory address
    - Another to get the real data

Fast Address Translation

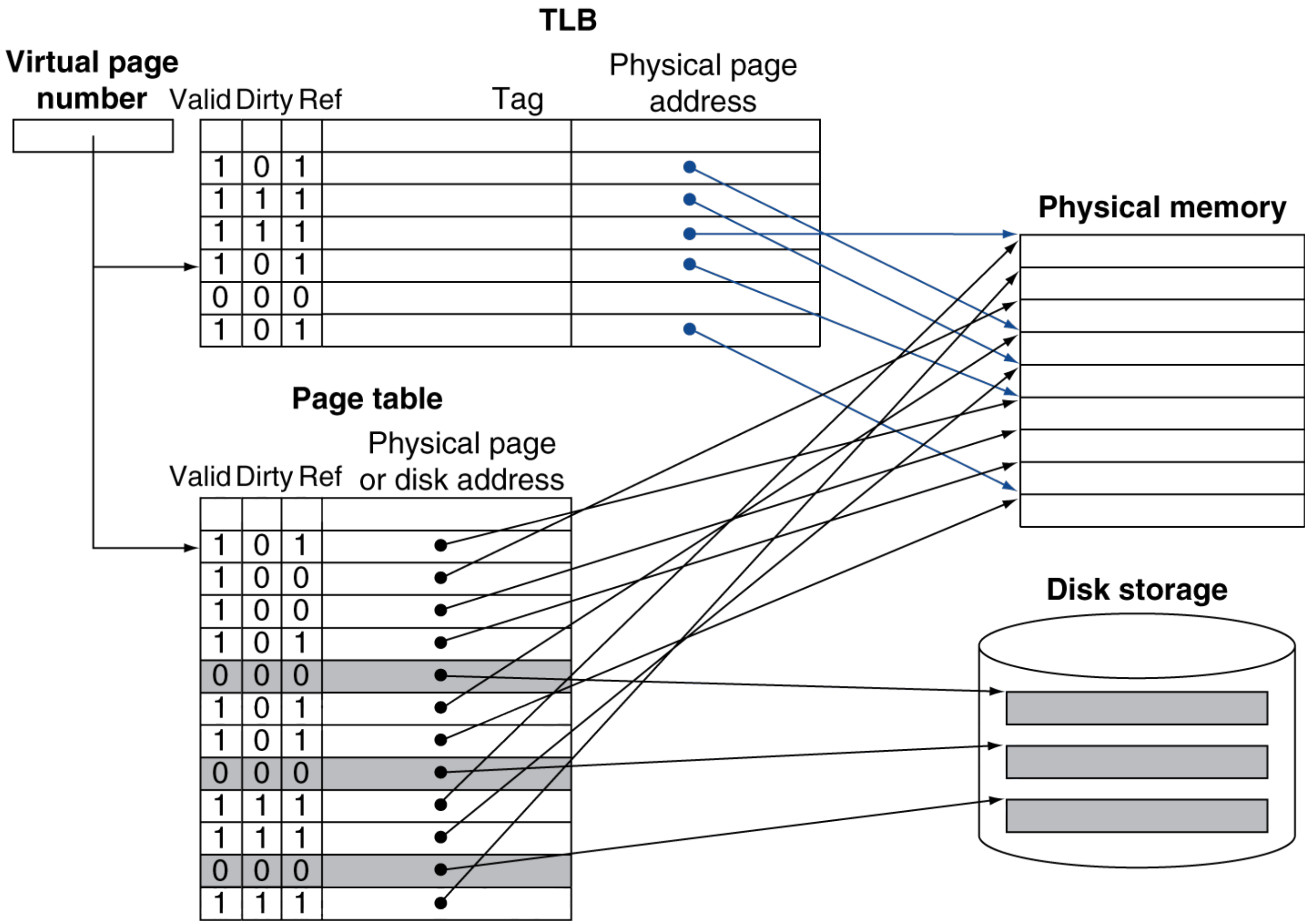
- Problem: Virtual Memory requires <u>two</u> memory accesses!
  - one to translate Virtual Address into Physical Address (page table lookup)
  - one to transfer the actual data (hit)
  - But Page Table is in physical memory! => 2 main memory accesses!

- Observation: since there is locality in pages of data, must be locality in virtual addresses of those pages!

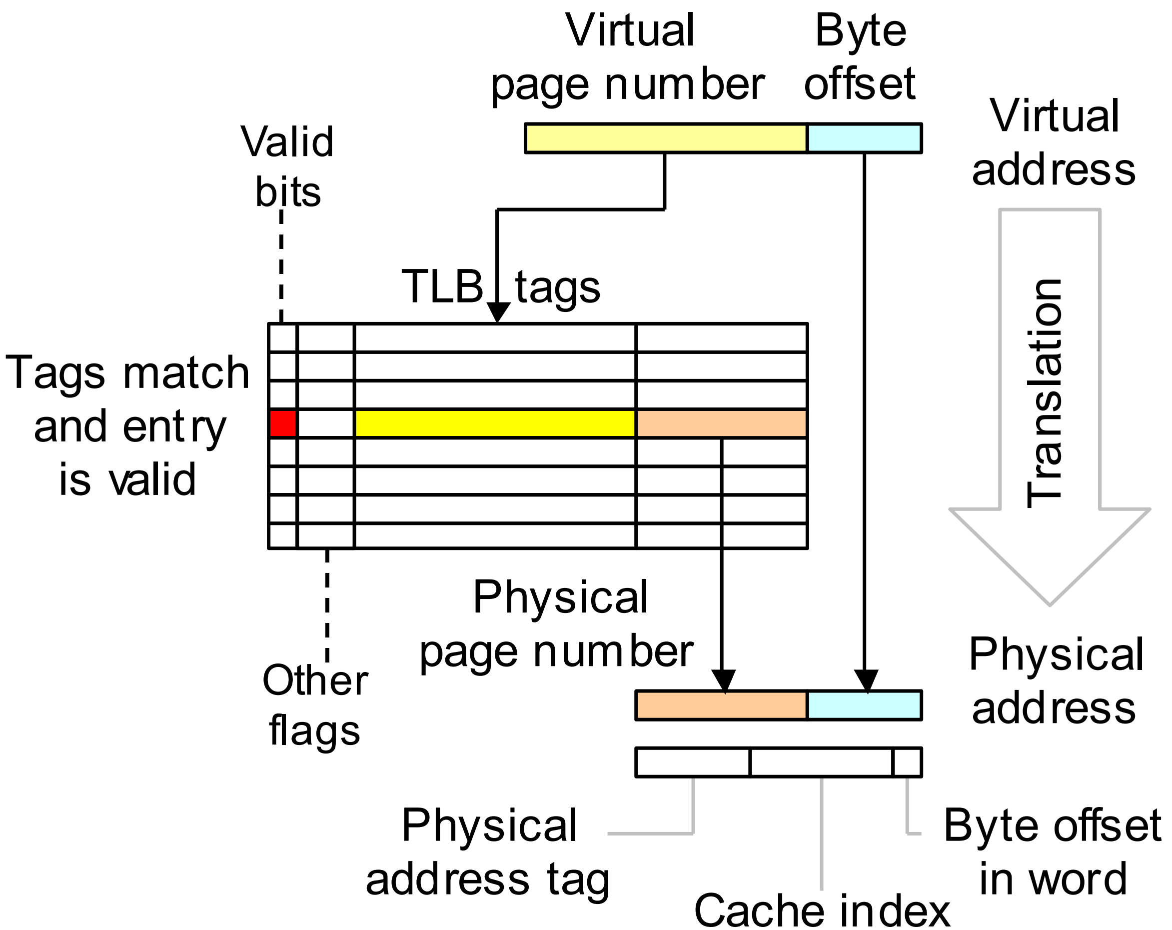
- Why not create a cache of virtual to physical address translations to make translation fast? (smaller is faster)

- For historical reasons, such a “page table cache” is called a <u>Translation Lookaside Buffer</u>, or <u>TLB</u>

Fast Translation Using a TLB



TLB Translation



Virtual-to-physical address translation by a TLB and how the resulting physical address is used to access the cache memory.

TLB Misses

- If page is in memory
  - Load the PTE from memory and retry
  - Could be handled in hardware
    - Can get complex for more complicated page table structures
  - Or in software
    - Raise a special exception, with optimized handler
- If page is not in memory (page fault)
  - OS handles fetching the page and updating the page table
  - Then restart the faulting instruction

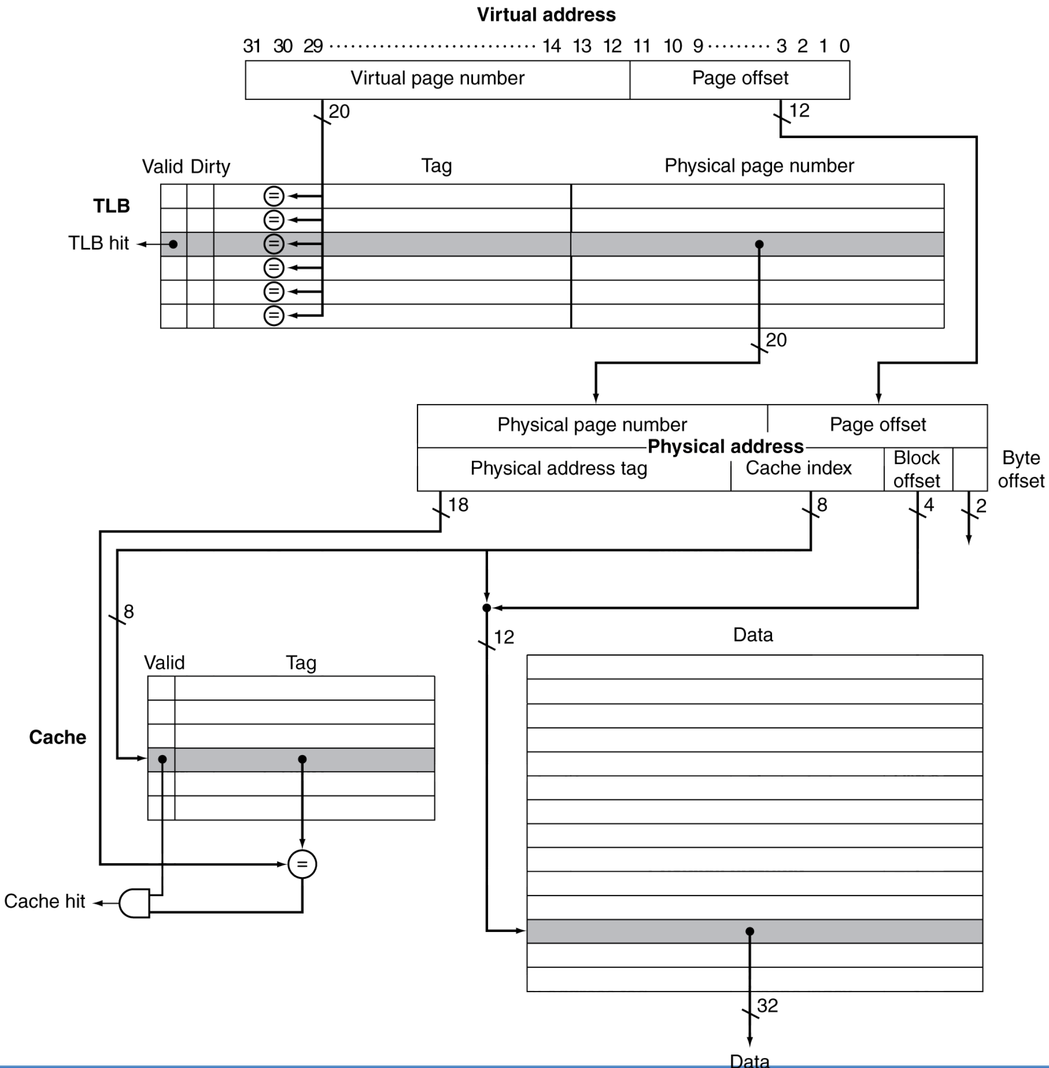
TLB Miss Handler

- TLB miss indicates
  - Page present, but PTE not in TLB
  - Page not preset
- Must recognize TLB miss before destination register overwritten
  - Raise exception
- Handler copies PTE from memory to TLB
  - Then restarts instruction
  - If page not present, page fault will occur

Page Fault Handler

- Use faulting virtual address to find PTE
- Locate page on disk
- Choose page to replace
  - If dirty, write to disk first
- Read page into memory and update page table
- Make process runnable again
  - Restart from faulting instruction

TLB and Cache Interaction



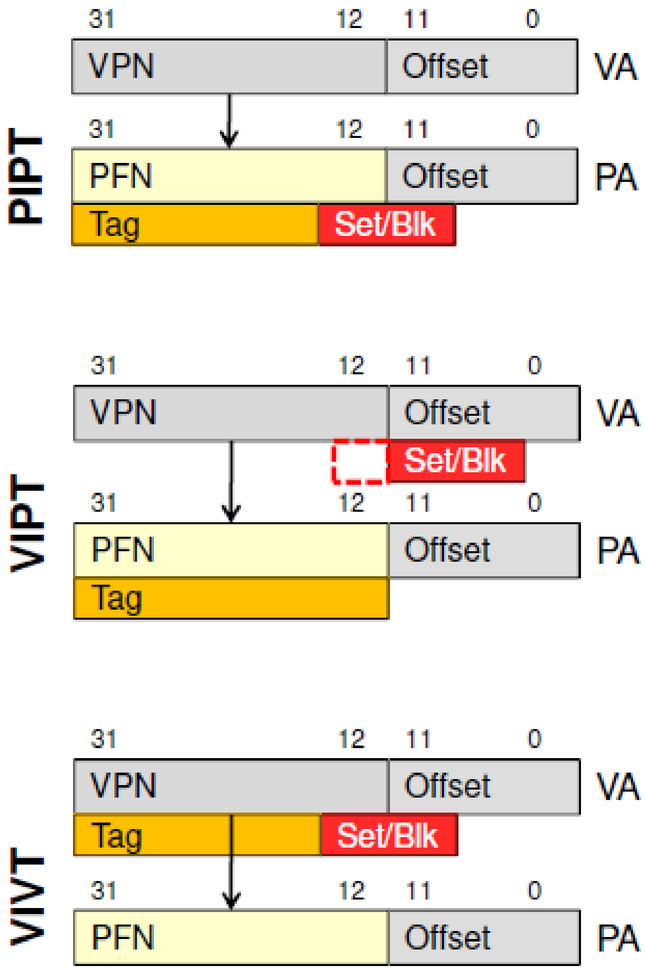
- If cache tag uses physical address
  - Need to translate before cache lookup
- Physically Indexed, Physically Tagged

TLB and Cache Addressing

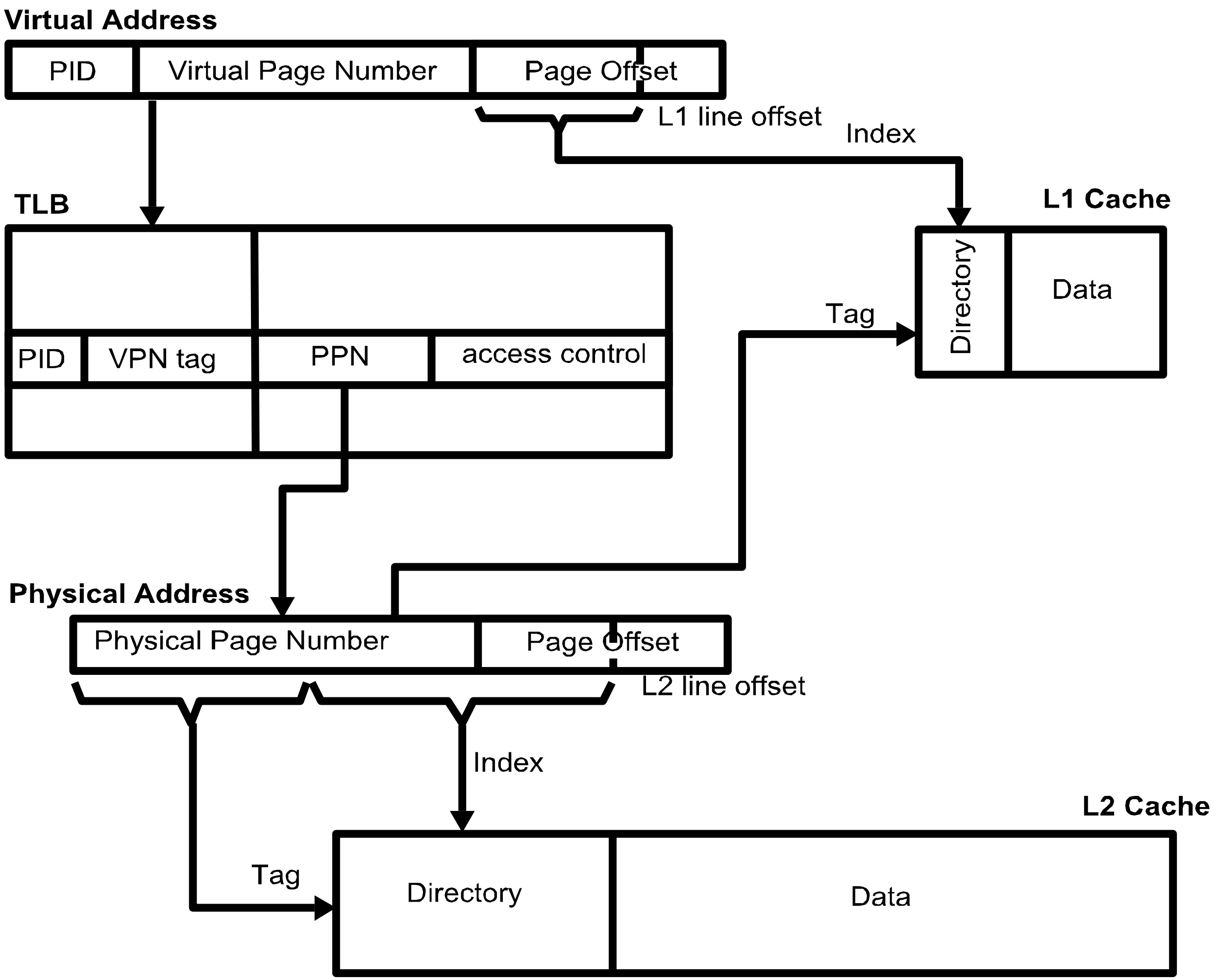
- Cache review
  - Set or block field indexes are used to get tags
  - 2 steps to determine hit:
    - Index (lookup) to find tags (using block or set bits)
    - Compare tags to determine hit
    - Sequential connection between indexing and tag comparison
- Rather than waiting for address translation and then performing this two step hit process, can we overlap the translation and portions of the hit sequence?
  - Yes!

Cache Index/Tag Options

- Physically indexed, physically tagged (PIPT)
  - Wait for full address translation
  - Then use physical address for both indexing and tag comparison
- Virtually indexed, physically tagged (VIPT)
  - Use portion of the virtual address for indexing then wait for address translation and use physical address for tag comparisons
- Virtually indexed, virtually tagged (VIVT)
  - Use virtual address for both indexing and tagging…No TLB access unless cache miss
  - Requires invalidation of cache lines on context switch or use of process ID as part of tags



Virtually Index Physically Tagged



Cache & Virtual memory

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

Summary

- Virtual Memory overcomes main memory size limitations

- VM supported through Page Tables

- TLB enables fast address translation