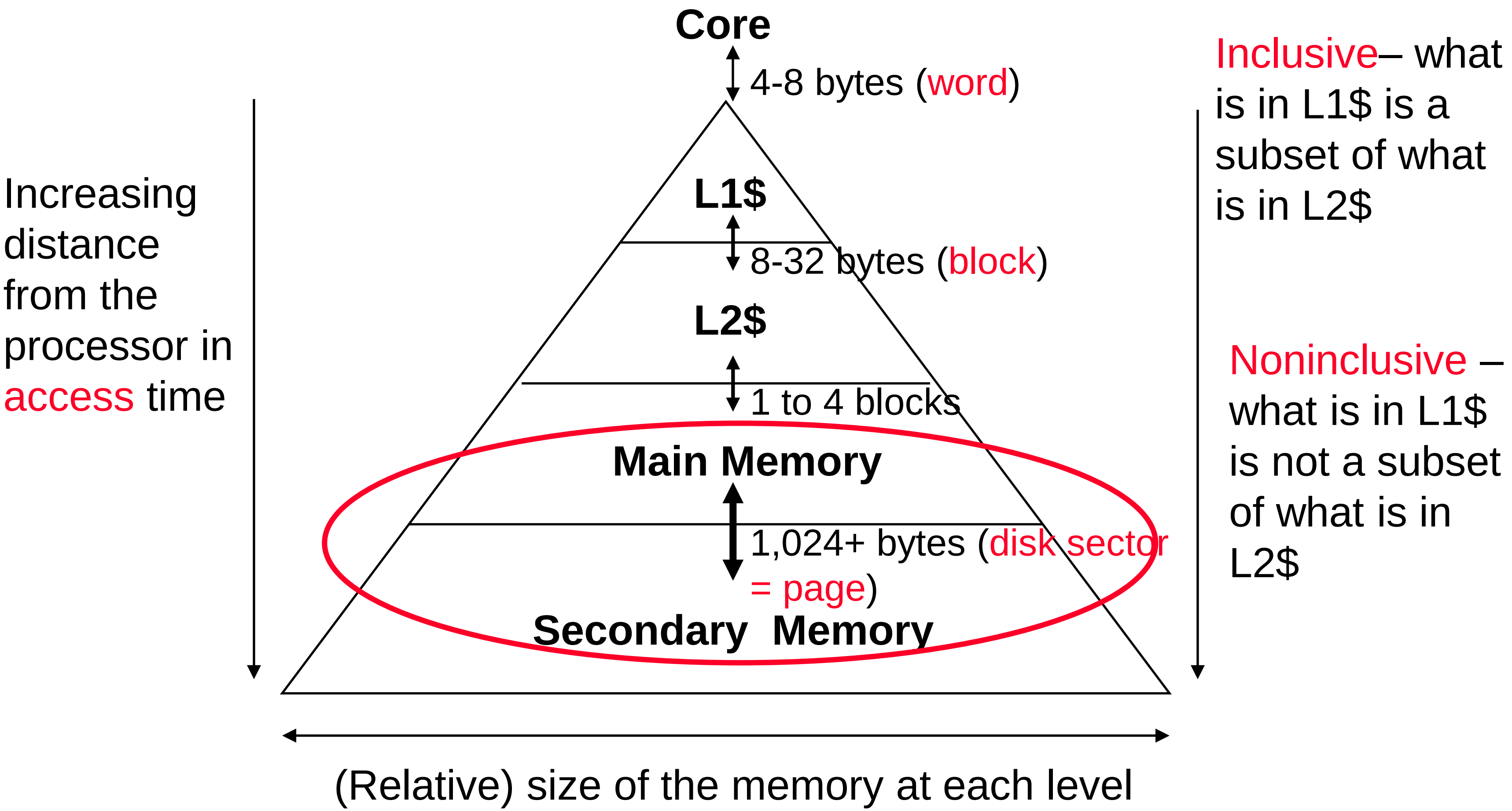
CSE 431
Computer Architecture
Fall 2017
Exploiting the Memory Hierarchy: TLBs

Mahmut Taylan Kandemir (www.cse.psu.edu/~kandemir)

[Adapted from *Computer Organization and Design, 5th Edition*,
Patterson & Hennessy, © 2014, MK]

Review: The Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology



How is the Hierarchy Managed?

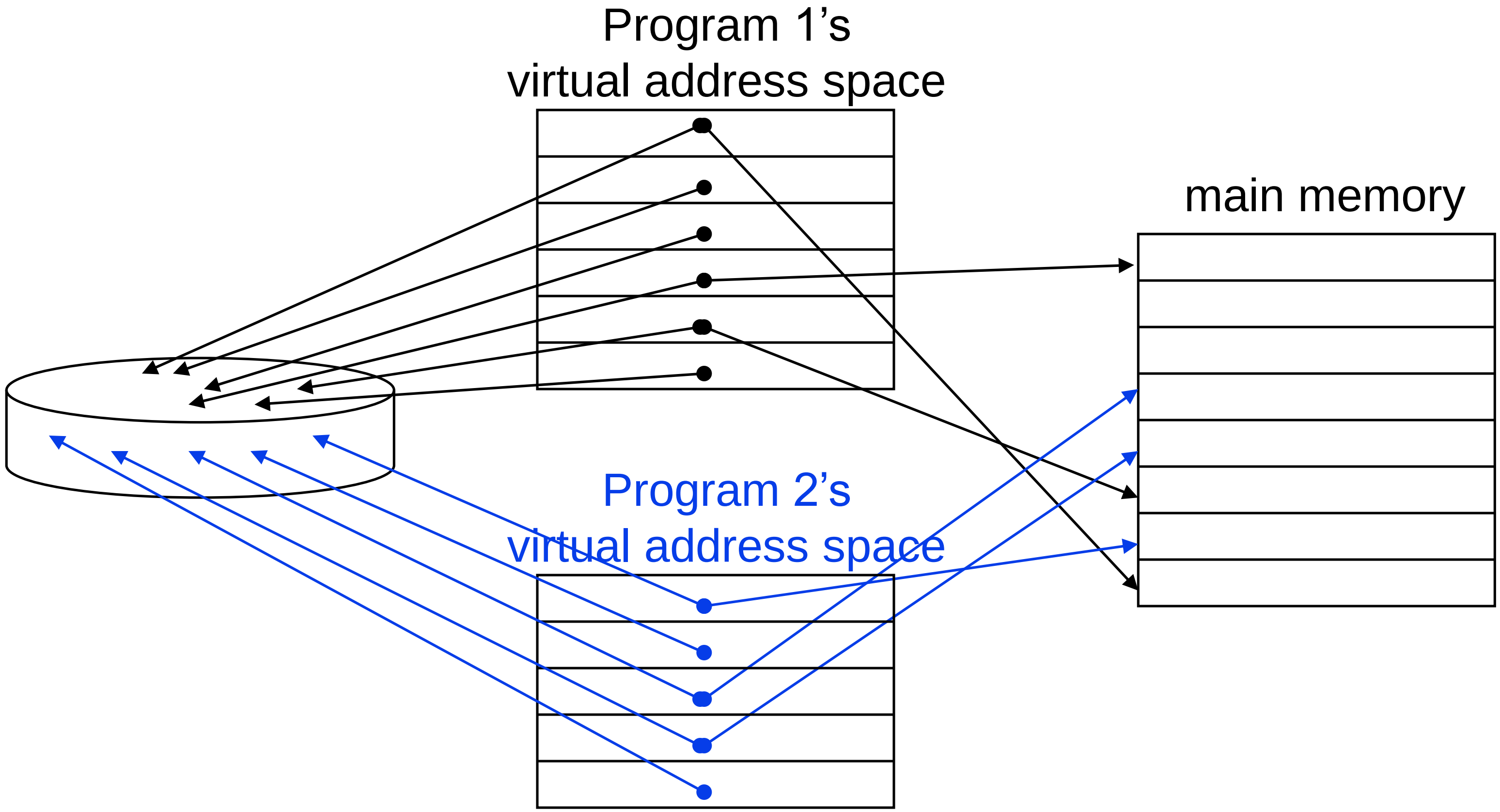
- registers $\leftrightarrow$ memory
  - by compiler (programmer?)
- registers $\leftrightarrow$ cache $\leftrightarrow$ main memory
  - by the cache controller hardware
- main memory $\leftrightarrow$ secondary memory (flash, disk)
  - by the operating system (virtual memory)
    - virtual address to physical address mapping
    - assisted by the hardware (TLB, page tables)
  - by the programmer with OS support (files)

Virtual Memory Concepts

- Use main memory as a “cache” for secondary memory
  - Allows efficient and safe sharing of main memory among multiple processes/threads (running programs)
    - Each program is compiled into its own private virtual address space
  - Provides the ability to run programs and data sets larger than the size of physical memory
  - Simplifies loading a program for execution by providing for code relocation (i.e., the code/data can be loaded in main memory anywhere the OS can find space for it)
- The core and OS work together to translate virtual addresses to physical addresses
  - A virtual memory miss (i.e., when the page is not in physical memory) is called a page fault
- What makes it work efficiently? – the Principle of Locality
  - Programs tend to access a only small portion of their address space over long portions of their execution time

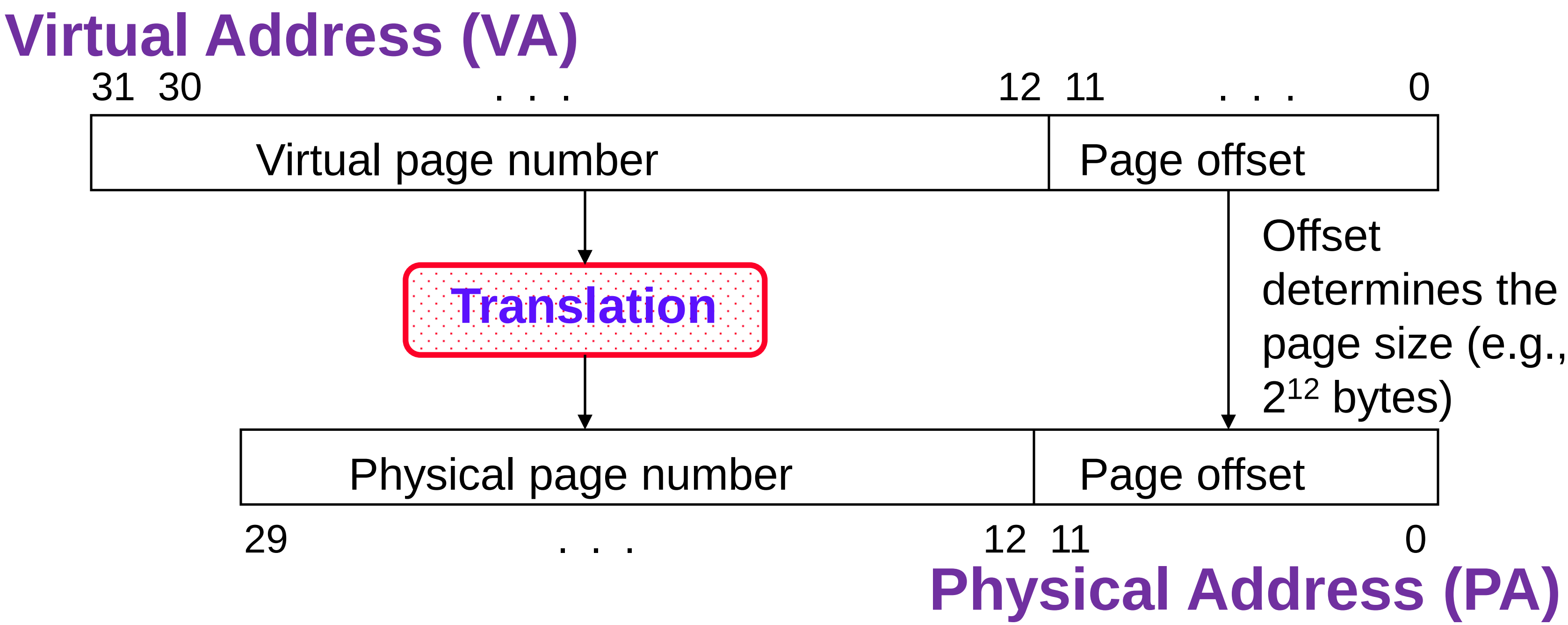
Two Programs Sharing Physical Memory

- A program's address space is divided into pages (all one fixed size) or segments (variable sizes)
  - The starting location of each page (either in main memory or in secondary memory) is contained in the program's page table



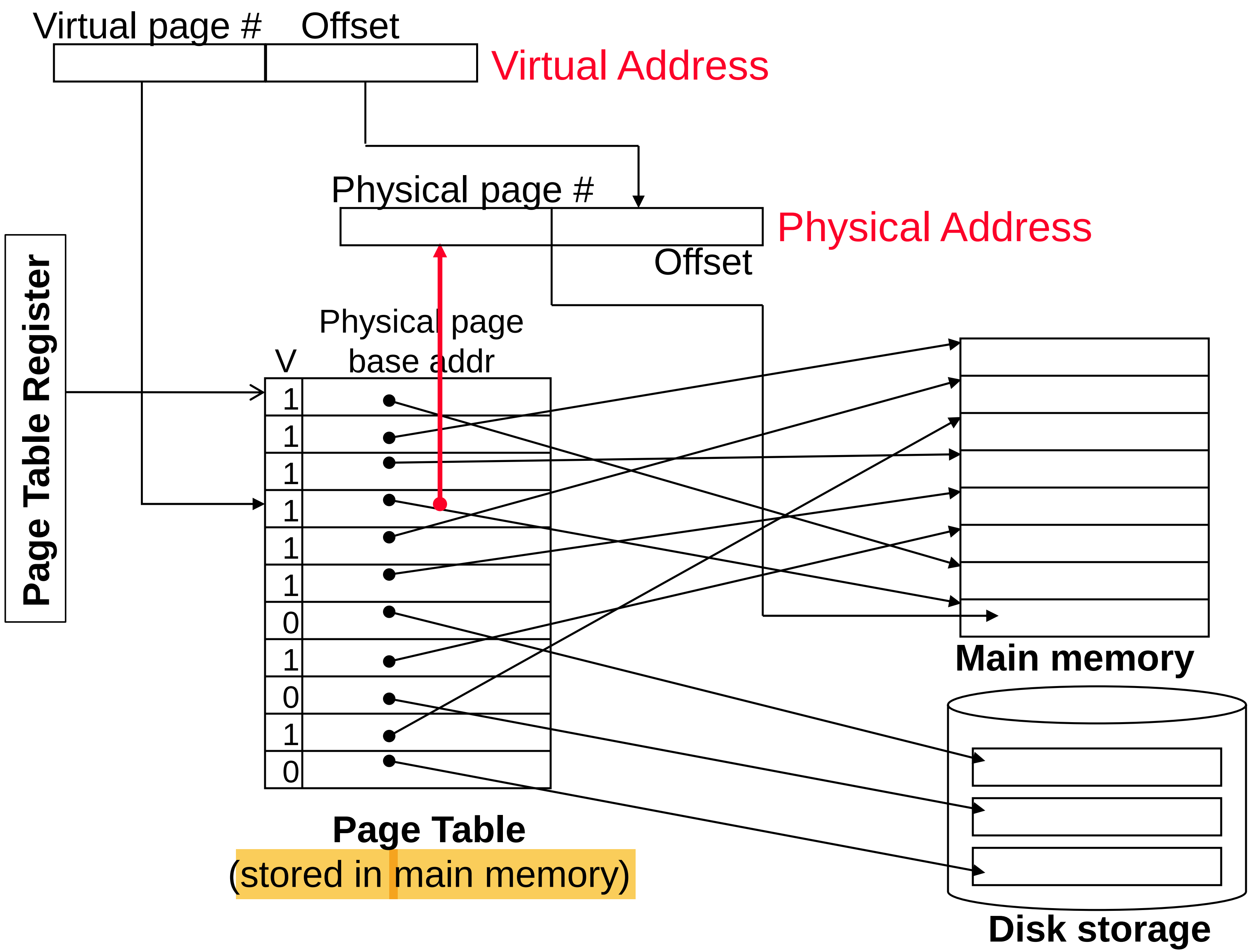
Address Translation

- A virtual address is translated to a physical address by a combination of hardware and software



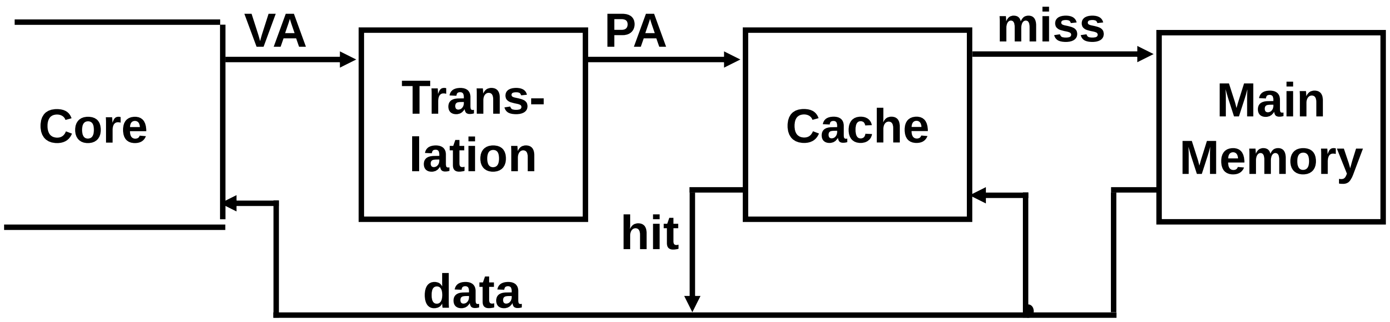
- So, each memory request *first* requires an address translation from the virtual space to the physical space

Virtual Address Translation Mechanisms



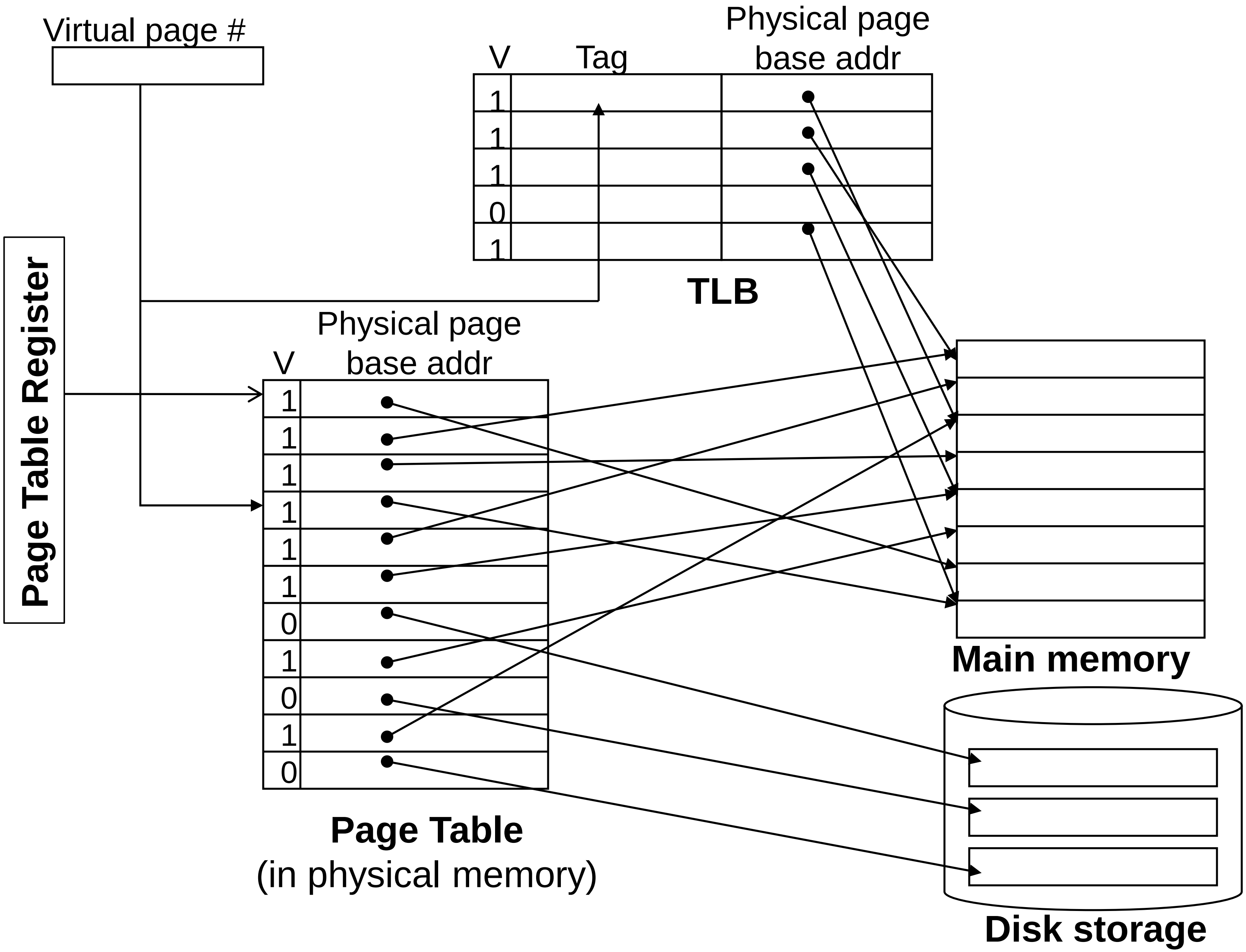
Virtual Addressing with a Cache

- Thus, it takes an *extra* memory access to translate a VA to a PA



- This makes memory (cache) accesses very expensive (if every access is really *two* accesses)
- The hardware fix is to use a Translation Lookaside Buffer (TLB) – a fast, small **cache** that keeps track of recently used **address mappings** to avoid having to do a page table lookup in memory (i.e., cache or main memory)

Making Address Translation Fast



Page Table
(in physical memory)

Translation Lookaside Buffers (TLBs)

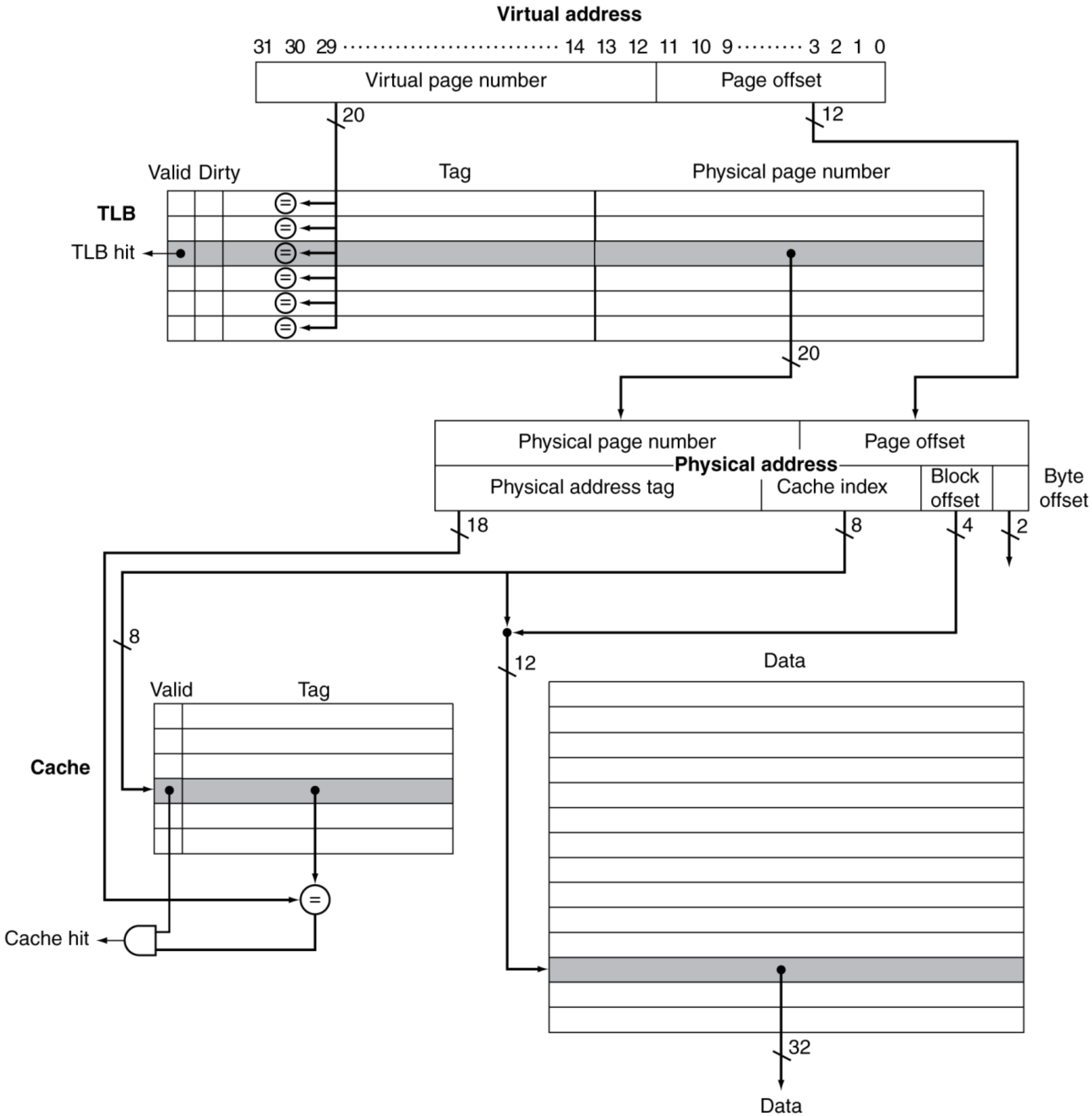
- Just like any other cache, the TLB can be organized as *fully associative*, *set associative*, or *direct mapped*
  - `simplescalar` defaults are `itbl:16:4096:4:l` (16 sets per way, 4-way set associative so 64 entries, 4096B pages) and `dtlb:32:4096:4:l` and `tlb:lat 30` (cycles to service a TLB miss)

V	Virtual Page #	Physical Page #	Dirty	Ref	Access

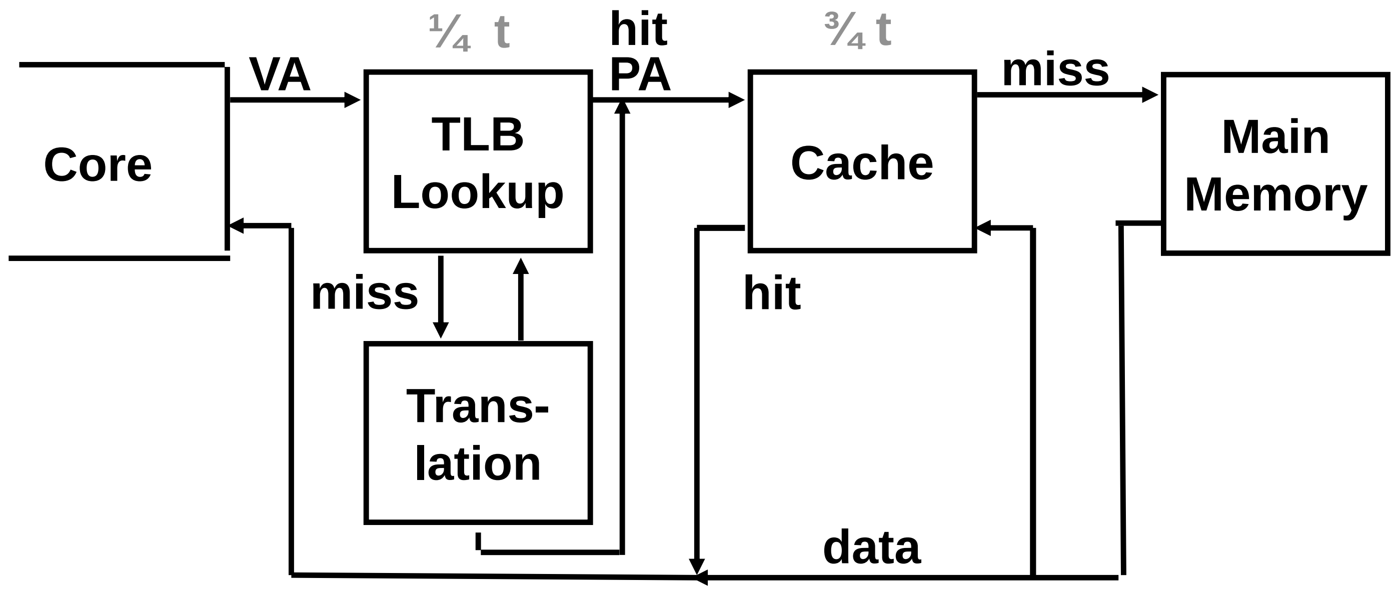
V = Valid?, Dirty = is the page dirty (so will have to be written back on replacement)?, Ref = Referenced recently?, Access = Write access allowed?

- TLB access time is typically much smaller than cache access time (because TLBs are much smaller than caches)
  - TLBs are typically not more than 512 entries even on high end machines

TLB with Cache Example



A TLB in the Memory Hierarchy



- **A TLB miss – is it a page fault or merely a TLB miss?**
  - If the page is loaded into main memory, then the TLB miss can be handled (in hardware or software) by loading the translation information from the page table into the TLB
    - Takes 10's of cycles to find and load the translation info into the TLB
  - If the page is not in main memory, then it's a *true* page fault
    - Takes 1,000,000's of cycles to service a page fault
- TLB misses are much more frequent than true page faults

TLB Event Combinations

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – this is what we want!
Hit	Hit	Miss	Yes – although the page table is not checked after the TLB hits
Miss	Hit	Hit	Yes – TLB missed, but PA is in page table and data is in cache; update TLB
Miss	Hit	Miss	Yes – TLB missed, but PA is in page table, data not in cache; update TLB
Miss	Miss	Miss	Yes – page fault; OS takes control
Hit	Miss	Miss/ Hit	No – TLB translation is not possible if the page is not present in main memory
Miss	Miss	Hit	No – data is not allowed in the cache if the page is not in memory

Handling a TLB Miss

- Consider a TLB miss for a page that is present in memory (i.e., the Valid bit in the page table is set)
  - A TLB miss (or a page fault exception) must be asserted by the end of the same clock cycle that the memory access occurs so that the next clock cycle will begin exception processing

Register	CP0 Reg #	Description
`EPC`	14	Where to restart after exception
`Cause`	13	Cause of exception
`BadVAddr`	8	Address that caused exception
`Index`	0	Location in TLB to be read/written
`Random`	1	Pseudorandom location in TLB
`EntryLo`	2	Physical page address and flags
`EntryHi`	10	Virtual page address
`Context`	4	Page table address & page number

A MIPS Software TLB Miss Handler

- When a TLB miss occurs, the hardware saves the address that caused the miss in `BadVAddr` and transfers control to 8000 0000_{hex}, the location of the TLB miss handler

```
TLBmiss:
 mfc0   $k1, Context    #copy addr of PTE into $k1
 lw     $k1, 0($k1)     #put PTE into $k1
 mtc0   $k1, EntryLo    #put PTE into EntryLo
 tlbwr                  #put EntryLo into TLB
                        #     at Random
 eret                   #return from exception
```

- `tlbwr` copies from `EntryLo` into the TLB entry selected by the control register `Random`
- A TLB miss takes about a dozen clock cycles to handle

Some Virtual Memory Design Parameters

	Paged VM	TLBs
Total size (blocks)	16,000 to 250,000	40 to 1,024
Total size (KB)	1,000,000 to 1,000,000,000	0.25 to 16
Block size (B)	4000 to 64,000	4 to 32
Hit time		0.25 to 1 clock cycle
Miss penalty (clocks)	10,000,000 to 100,000,000	10 to 1,000
Miss rates	0.00001% to 0.0001%	0.01% to 1%

Current TLB Stats

Characteristic	ARM Cortex-A8	Intel Core i7
Virtual address	32 bits	48 bits
Physical address	32 bits	44 bits
Page size	Variable: 4, 16, 64 KiB, 1, 16 MiB	Variable: 4 KiB, 2/4 MiB
TLB organization	1 TLB for instructions and 1 TLB for data Both TLBs are fully associative, with 32 entries, round robin replacement TLB misses handled in hardware	1 TLB for instructions and 1 TLB for data per core Both L1 TLBs are four-way set associative, LRU replacement L1 I-TLB has 128 entries for small pages, 7 per thread for large pages L1 D-TLB has 64 entries for small pages, 32 for large pages The L2 TLB is four-way set associative, LRU replacement The L2 TLB has 512 entries TLB misses handled in hardware

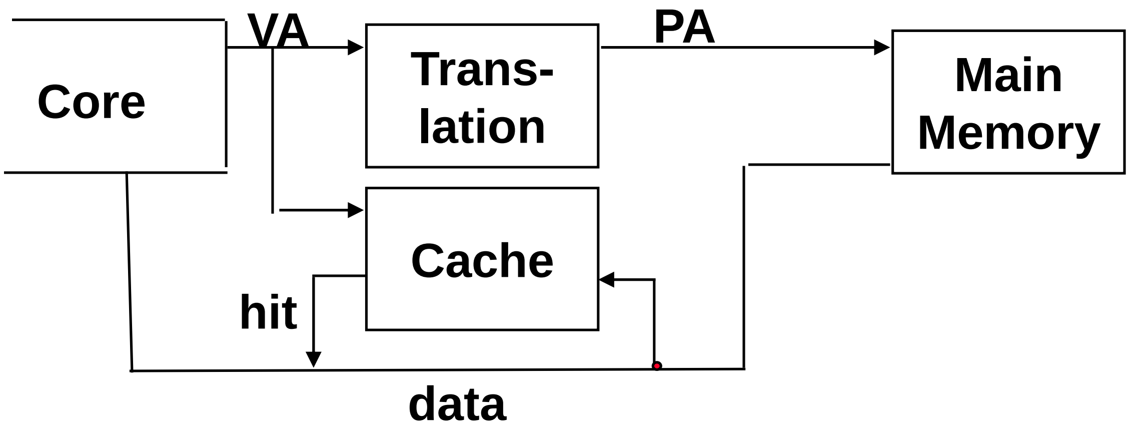
TLB Management

- **Hardware-managed TLB**
  - No need for expensive interrupts
  - Pipeline remains largely unaffected
  - OS cannot employ alternate design

- **Software-managed TLB**
  - Data structure design is flexible since the OS controls the page table walk
  - Miss handler is also instructions
    - It may itself miss in the inst. cache.
  - Data cache may be polluted by the page table walk

Why Not a Virtually Addressed Cache?

- A virtually addressed cache would only require address translation on cache misses

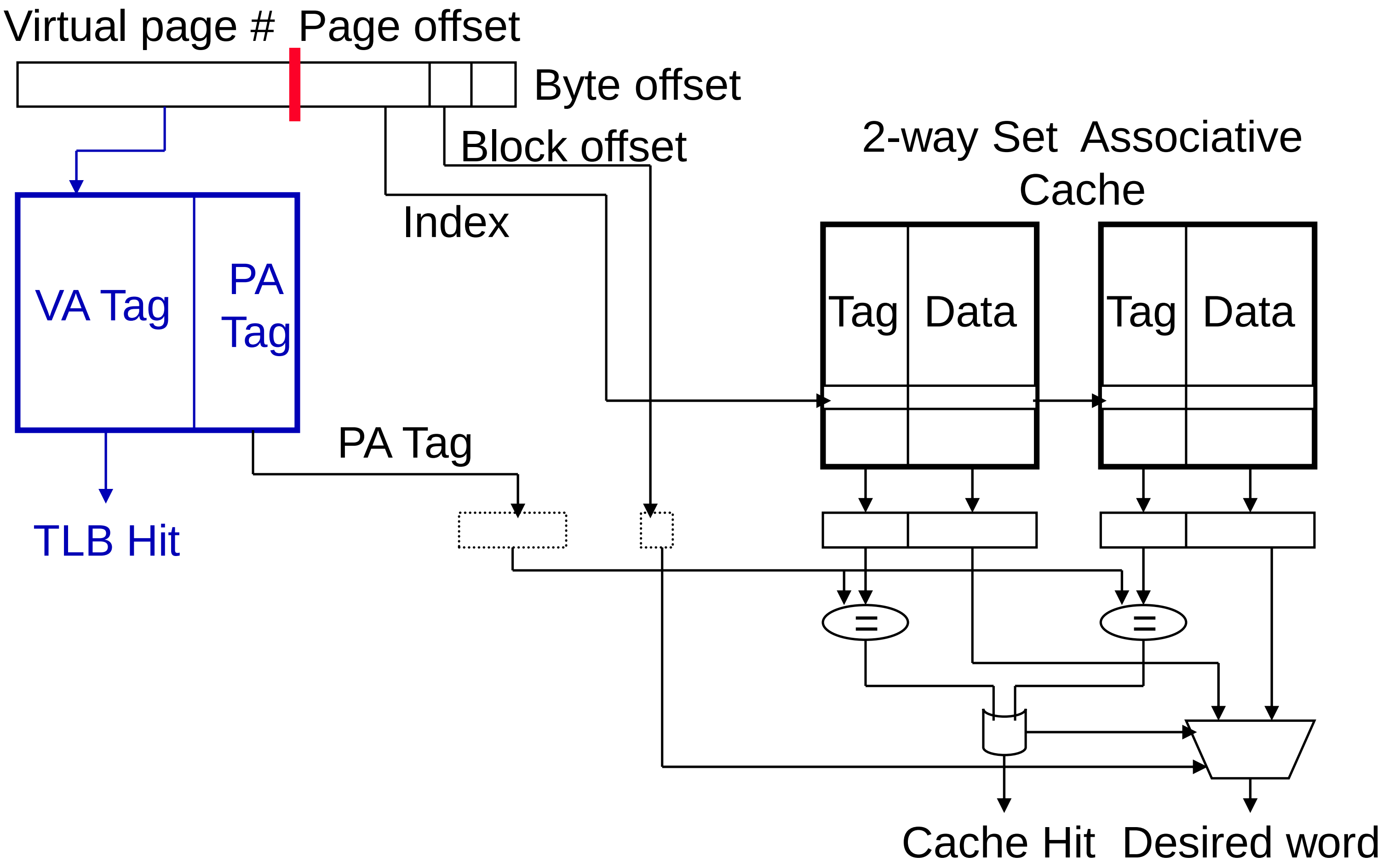


But,

- Two programs which are sharing data will have two different virtual addresses for the same physical address – aliasing – so will have two copies of the shared data in the cache and two entries in the TBL which would lead to coherence issues
  - Must update all cache entries with the same physical address or the memory becomes inconsistent

Further Reducing Translation Time

- Can overlap the cache access with the TLB access
  - Works when the high order bits of the VA are used to access the TLB while the low order bits are used as index into cache



The Hardware/Software Interface

- What parts of the virtual to physical address translation is done by or assisted by the hardware?
  - Translation Lookaside Buffer (TLB) that caches the recent translations
    - TLB access time is part of the cache hit time
    - May need to allot an extra stage in the pipeline for TLB access
  - Page table storage, fault detection and updating
    - Page faults result in interrupts (precise) that are then handled by the OS
    - Hardware must support (i.e., update appropriately) Dirty and Reference bits (e.g., ~LRU) in the Page Tables
  - Disk placement
    - Bootstrap (e.g., out of disk sector 0) so the system can service a limited number of page faults before the OS is even loaded

Memory and OS Protection

- Different processes can share parts of their virtual address spaces
  - For example, run a program under control of a debugger, or sharing data and computing effort between processes
  - Need to protect against improper access
  - Requires OS assistance via the page tables and locking mechanisms
- Hardware support for OS protection
  - Privileged supervisor mode (aka kernel mode)
  - Privileged instructions, registers and addresses
  - Page tables and other state information only accessible in supervisor mode
  - System call exception (e.g., syscall in MIPS)
  - etc., etc.

Common Memory Hierarchy Framework

Common Memory Framework

- The Principle of Locality:
  - Program likely to access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Caches, TLBs, Virtual Memory all understood by examining how they deal with the four questions
  1. Where can an entry be placed in the upper level?
  2. How is an entry found if it is in the upper level?
  3. What entry is replaced on miss?
  4. How are writes handled?
- Page tables map virtual address to physical address
  - TLBs are important for fast translation

Direct vs Associate Caching Memory Organizations

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Q1&Q2: Where can a entry be placed/found?

	# of sets	Entries per set
Direct mapped	# of entries	1
Set associative	(# of entries)/ associativity	Associativity (typically 2 to 16)
Fully associative	1	# of entries

	Location method	# of comparisons
Direct mapped	Index	1
Set associative	Index the set; compare set's tags	Degree of associativity
Fully associative	Compare all entries' tags	# of entries
	Separate lookup (page) table	0

Q3: Which entry should be replaced on a miss?

- Easy for direct mapped – only one choice
- Set associative or fully associative
  - Random
  - LRU (Least Recently Used)
- For a 2-way set associative, random replacement has a miss rate about 10% higher than LRU
- LRU is too costly to implement for high levels of associativity (> 16-way) since tracking the usage information is costly

Q4: What happens on a write?

- *Write-through* – The information is written to the entry in the current memory level *and* to the entry in the next level of the memory hierarchy
  - Always combined with a write buffer so write waits to next level memory can be eliminated (as long as the write buffer doesn't fill)
- *Write-back* – The information is written only to the entry in the current memory level. The modified entry is written to next level of memory only when it is replaced.
  - Need a dirty bit to keep track of whether the entry is clean or dirty
  - Virtual memory systems always use write-back of dirty pages to disk
- Pros and cons of each?
  - Write-through: read misses don't result in writes (so are simpler and cheaper), easier to implement
  - Write-back: writes run at the speed of the cache; repeated writes require only one write to lower level

Handling writes (stores)

- Do we allocate a cache block on a <u>write miss</u>?
  - Allocate on write miss: Yes
  - No-allocate on write miss: No

- Allocate on write miss
  - + Can consolidate writes instead of writing each of them individually to next level
  - + Simpler because write misses can be treated the same way as read misses
  - -- Requires (?) transfer of the whole cache block

- No-allocate
  - + Conserves cache space if locality of writes is low (potentially better cache hit rate)

Memory hierarchy design challenges

Design change	Effect on miss rate	Possible negative performance effect
Increases cache size	Decreases capacity misses	May increase access time
Increases associativity	Decreases miss rate due to conflict misses	May increase access time
Increases block size	Decreases miss rate for a wide range of block sizes due to spatial locality	Increases miss penalty. Very large block could increase miss rate

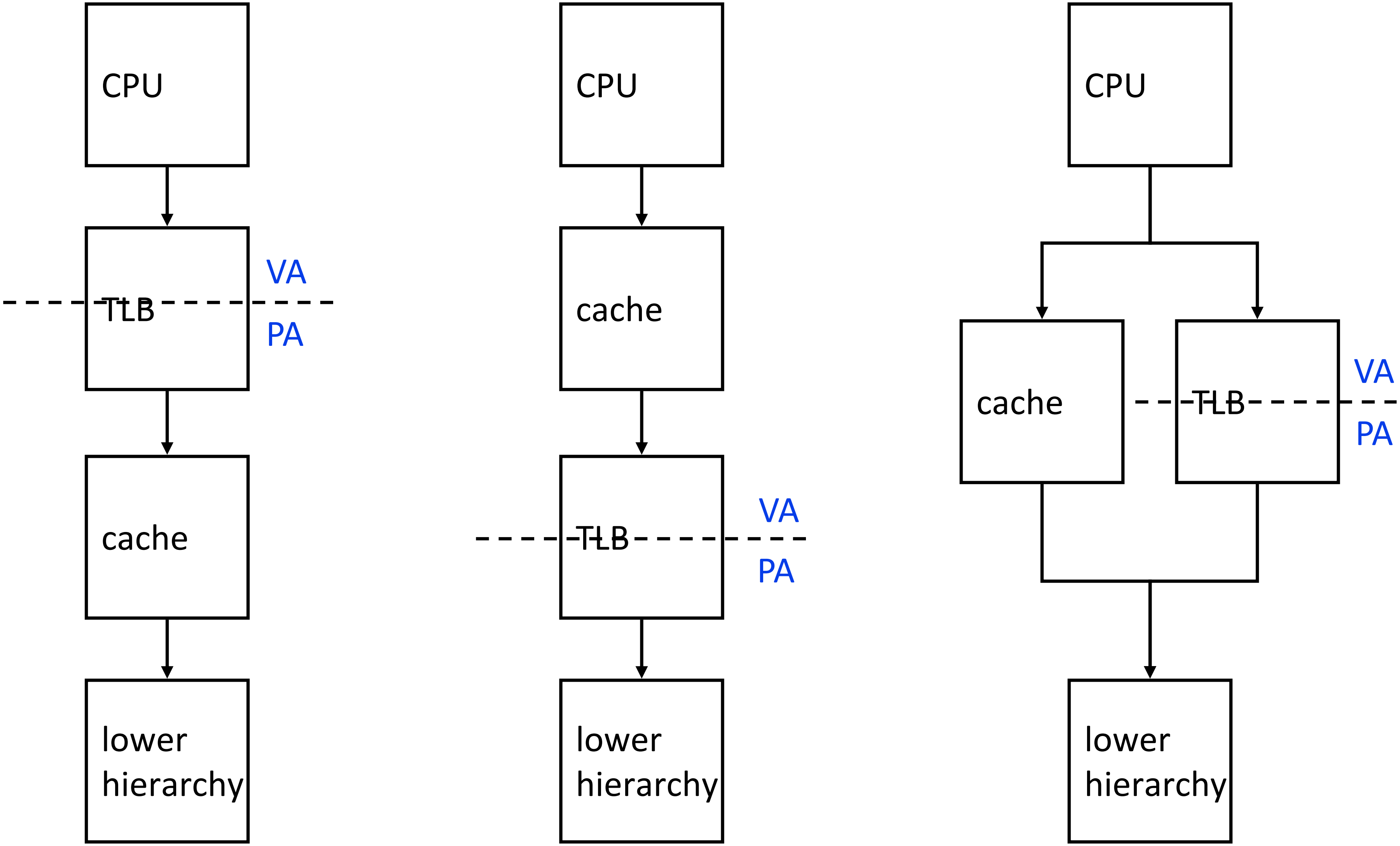
Cache versus page replacement

- Physical memory (DRAM) is a cache for disk
  - Usually managed by system software via the virtual memory subsystem

- Page replacement is similar to cache replacement
- Page table is the “tag store” for physical memory data store

- What is the difference?
  - Hardware versus software
  - Number of blocks in a cache versus physical memory
  - “Tolerable” amount of time to find a replacement candidate

Cache-virtual memory interaction



physical cache

virtual (L1) cache

virtual-physical cache

Virtually-Indexed Physically-Tagged

- If C≤(page_size × associativity), the cache index bits come only from page offset (same in VA and PA)
- If both cache and TLB are on chip
  - index both arrays concurrently using VA bits
  - check cache tag (physical) against TLB output at the end