

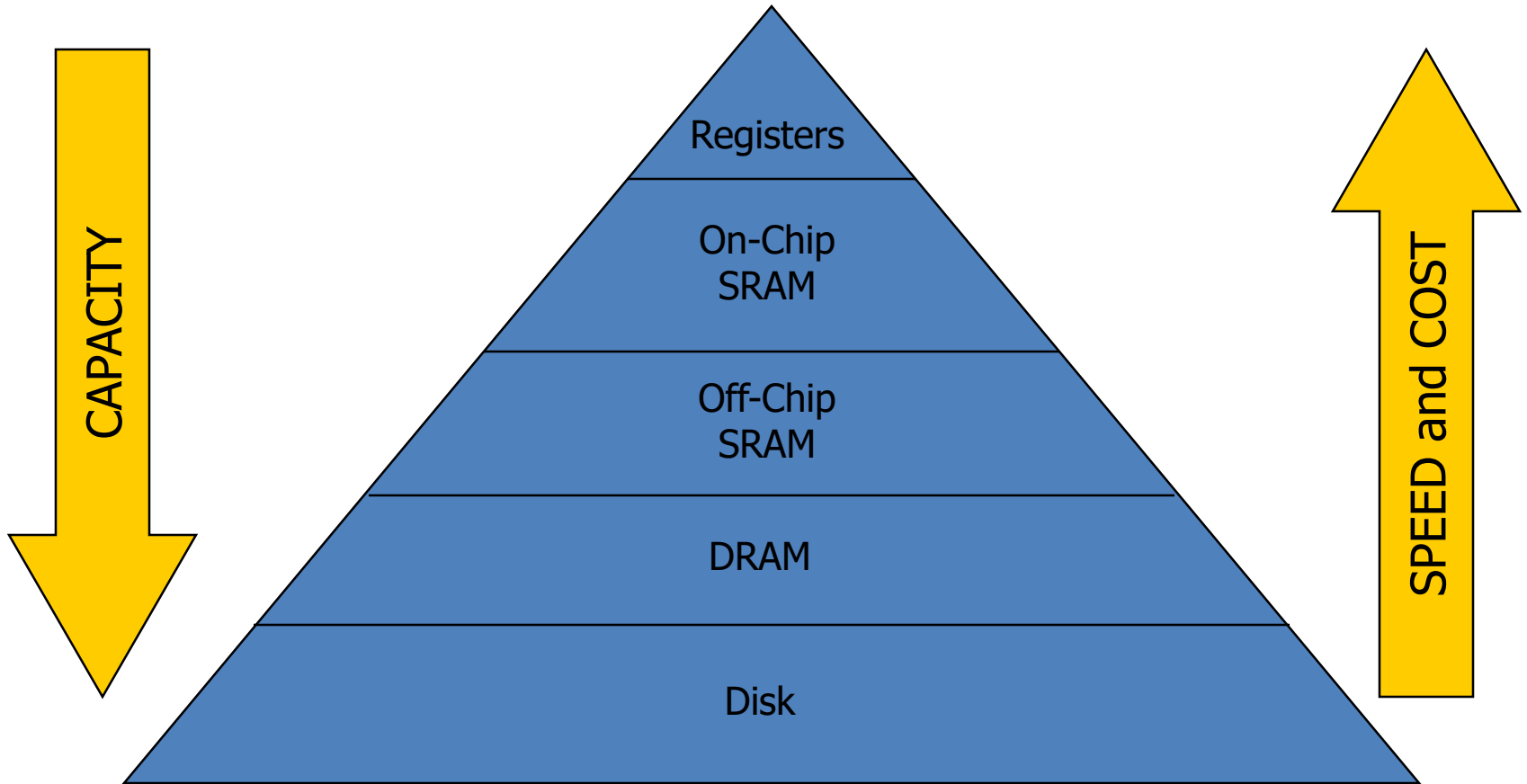
Computer Architecture

13. Virtual Memory

Jianhua Li

College of Computer and Information
Hefei University of Technology

Memory Hierarchy



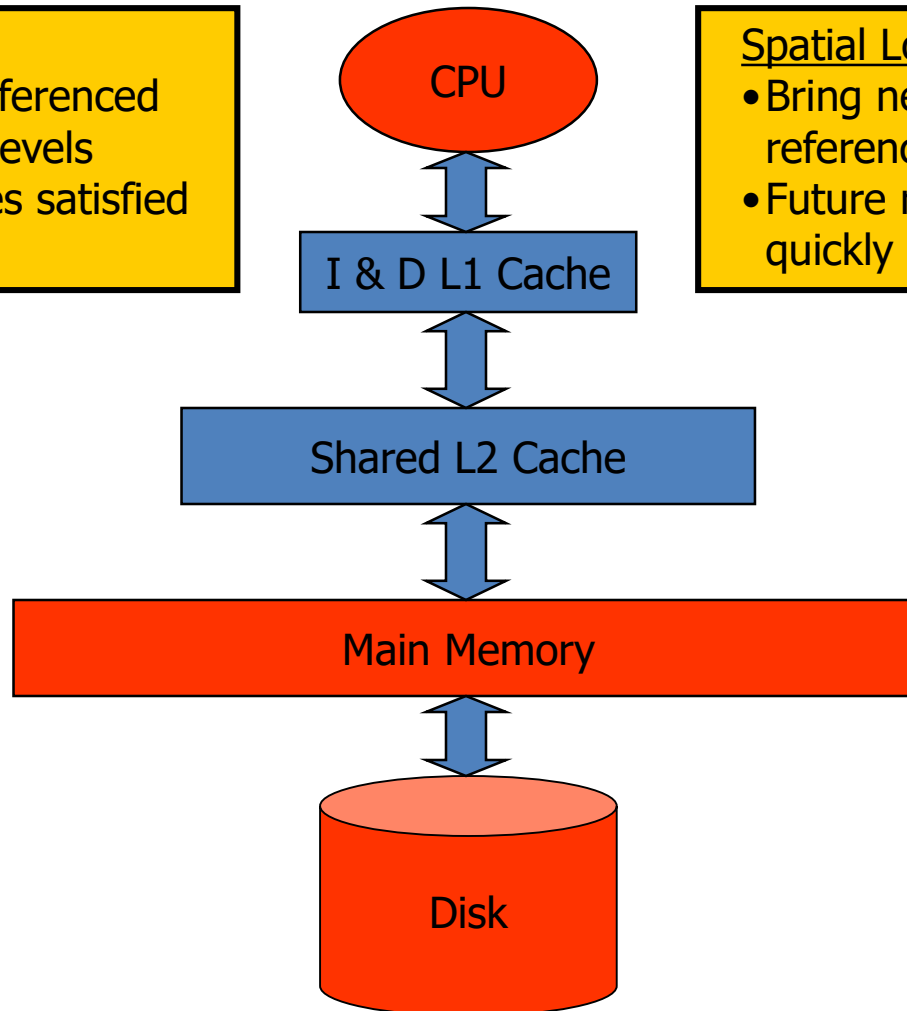
Memory Hierarchy

Temporal Locality

- Keep recently referenced items at higher levels
- Future references satisfied quickly

Spatial Locality

- Bring neighbors of recently referenced to higher levels
- Future references satisfied quickly



Four Kernel Questions

- These are:
 - Placement
 - Where can a block of memory go?
 - Identification
 - How do I find a block of memory?
 - Replacement
 - How do I make space for new blocks?
 - Write Policy
 - How do I propagate changes?
- Consider these for registers and main memory
 - Main memory usually DRAM
 - Novel main memory is in progress
 - Optane, PCM, etc.

Placement

Memory Type	Placement	Comments
Registers	Anywhere; Int, FP, SPR	Compiler/programmer manages
Cache (SRAM)	Fixed in H/W	<i>Direct-mapped, set-associative, fully-associative</i>
DRAM	Anywhere	O/S manages
Disk	Anywhere	O/S manages

Virtual Memory

- Use and benefits of virtual memory
 - Main memory becomes another level in the memory hierarchy
 - Enables programs with address space or working set that exceed physically available memory
 - No need for programmer to manage *overlays*, etc.
 - Sparse use of large address space is OK
 - Allows multiple users or programs to timeshare limited amount of physical memory space and address space
- Bottom line: efficient use of expensive resource, and ease of programming

Virtual Memory

- Virtual Memory enables
 - Use more memory than system has
 - Program can think it is the only one running
 - Don't have to manage address space usage across programs
 - E.g. think it always starts at address 0x0
 - Memory protection
 - Each program has private VA space: no-one else can clobber
 - Better performance
 - Start running a large program before all of it has been loaded from disk

Virtual Memory – Placement

- Main memory managed in larger blocks
 - *Page size* typically 4K – 16K
 - Super page is emerging
- Fully flexible placement; fully associative
 - Operating system manages placement
 - Indirection through *page table*
 - Maintain mapping between:
 - Virtual address (seen by programmer)
 - Physical address (seen by main memory)

Virtual Memory – Placement

- Fully associative implies expensive lookup?
 - In caches, yes: check multiple tags in parallel
- In virtual memory, expensive lookup is avoided by using **a level of indirection**
 - Lookup table or hash table
 - Called a *page table*

Virtual Memory – Identification

Virtual Address	Physical Address	Dirty bit
0x20004000	0x2000	Y/N

- Similar to cache tag array
 - Page table entry contains VA, PA, dirty bit
- Virtual address:
 - Matches programmer view
 - Can be the same for multiple programs sharing same system, without conflicts
- Physical address:
 - Invisible to programmer, managed by O/S
 - Created/deleted on demand basis, can change

Virtual Memory – Replacement

- Similar to caches:
 - FIFO
 - LRU; **overhead too high**
 - Approximated with reference bit checks
 - Clock algorithm
 - Random
- O/S decides, manages
 - OS course for reference

Virtual Memory – Write Policy

- Write back
 - Disks are too slow to write through
- Page table maintains dirty bit
 - Hardware must set dirty bit on first write
 - O/S checks dirty bit on eviction
 - Dirty pages written to backing store
 - Disk write, 10+ ms

Virtual Memory Implementation

- Caches have fixed policies, hardware FSM for control, pipeline stall
- VM has very different miss penalties
 - Remember disks are 10+ ms!
 - Even SSDs are (at best) 1.5ms
 - 1.5ms is 3M processor clocks @ 2GHz
- Hence engineering choice are different

Page Faults

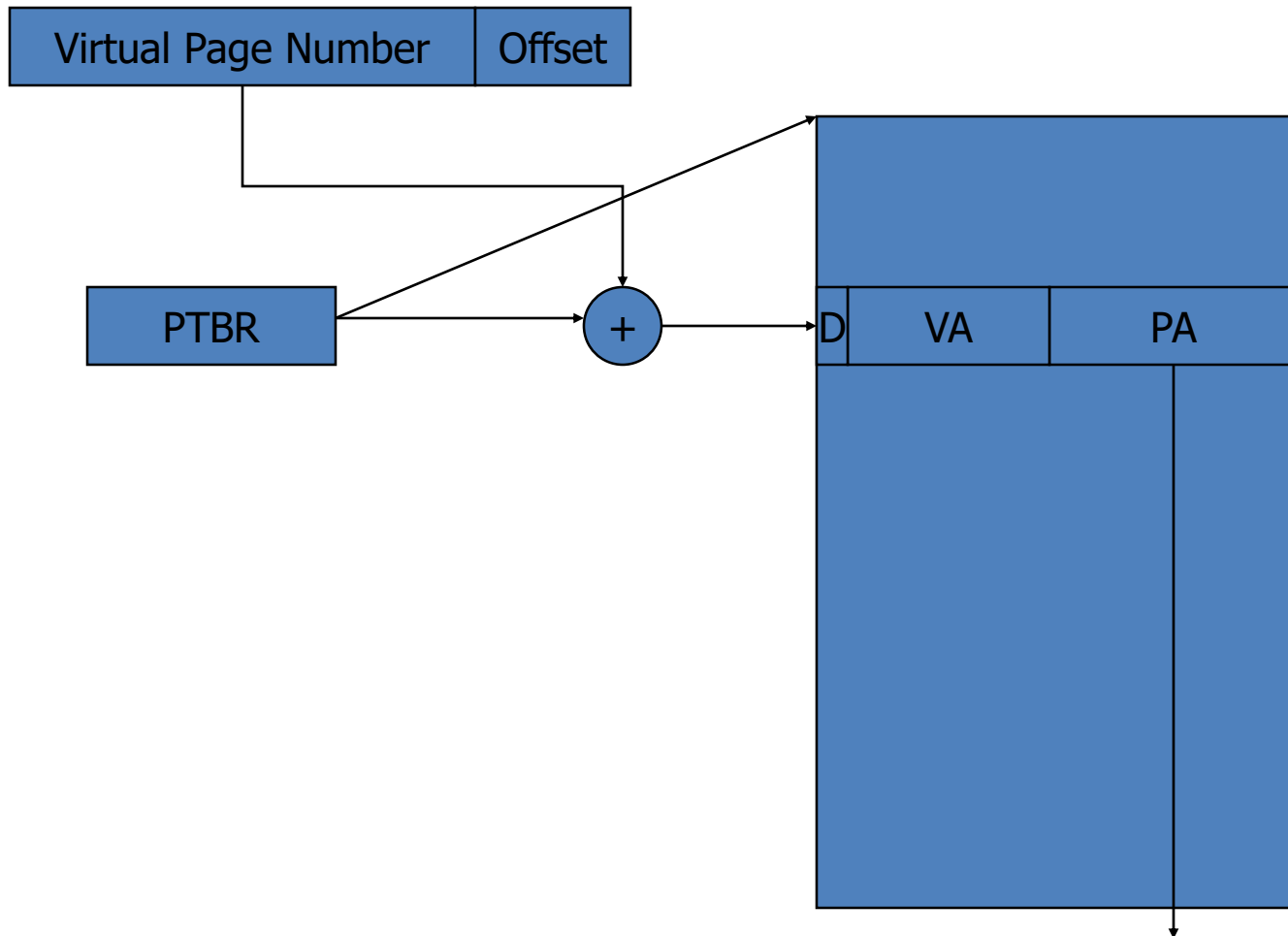
- A virtual memory miss is a page fault
 - Physical memory location does not exist
 - Exception is raised, save PC
 - Invoke OS page fault handler
 - Find a physical page (possibly evict)
 - Initiate fetch from disk
 - Switch to other task that is ready to run
 - Interrupt when disk access complete
 - Restart original instruction
- Why use O/S and not hardware FSM?

Address Translation

VA	PA	Dirty	Ref	Protection
0x20004000	0x2000	Y/N	Y/N	Read/Write/ Execute

- O/S and hardware communicate via PTE
- How do we find a PTE?
 - $\&\text{PTE} = \text{PTBR} + \text{page number} * \text{sizeof}(\text{PTE})$
 - PTBR is private for each program
 - Context switch replaces PTBR contents

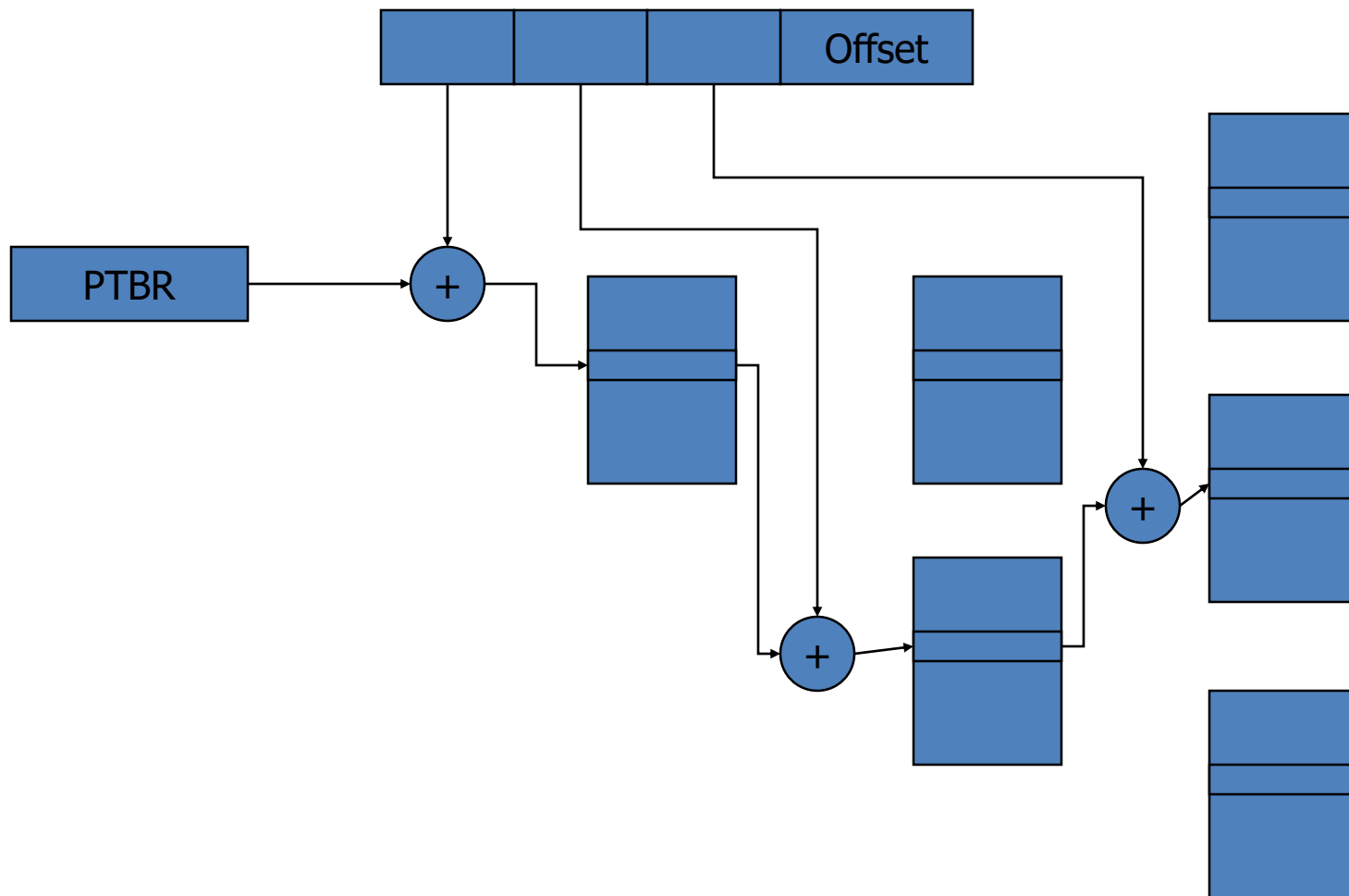
Address Translation



Page Table Size

- How big is page table?
 - $2^{32} / 4K * 4B = 4M$ per program (!)
 - Much worse for 64-bit machines
- To make it smaller
 - Use a multi-level page table
 - Use an inverted (hashed) page table
 - Not mentioned in this course

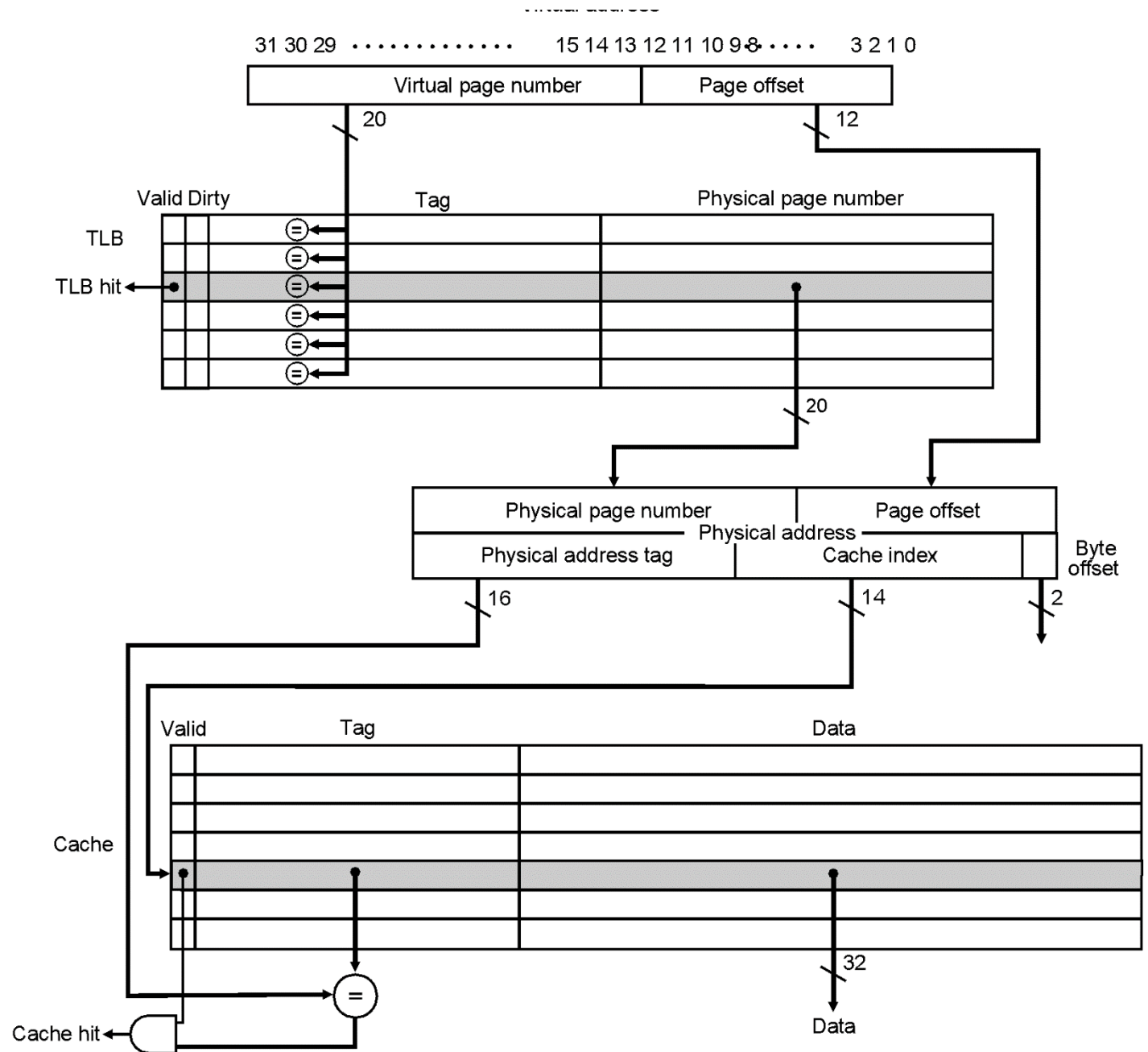
Multilevel Page Table



High-Performance VM

- VA translation
 - Additional memory reference to PTE
 - Each instruction fetch/load/store now 2 memory references
 - Or more, with multilevel table or hash collisions
 - Even if PTE are cached, still slow
- Hence, use special-purpose buffer for PTEs
 - Called TLB (translation lookaside buffer)
 - Caches PTE entries
 - Exploits temporal and spatial locality (just a cache)

TLB



Virtual Memory Protection

- Each process/program has private virtual address space
 - Automatically protected from rogue (恶意) programs
- Sharing is possible, necessary, desirable
 - Avoid copying, staleness issues, etc.
- Sharing in a controlled manner
 - Grant specific permissions
 - Read
 - Write
 - Execute
 - Any combination
 - Store permissions in PTE and TLB

VM Sharing

- Share memory locations by:
 - Map shared physical location into both address spaces:
 - E.g. PA 0xC00DA becomes:
 - VA 0x2D000DA for process 0
 - VA 0x4D000DA for process 1
 - Either process can read/write shared location
- However, causes homonym and synonym problem

VA Homonyms

- Same VA to different PA
- How to solve?
 - Flush cache on context switch
 - Add process ID to each tag
 - Use physically addressed caches

VA Synonyms

- Different VA to same PA
 - Can end up with two copies of same physical line
- Solutions:
 - hardware synonym detection
 - flush cache on context switch
 - doesn't help for aliasing within address space
 - detect synonyms and ensure
 - all read-only, OR
 - only one synonym mapped
 - restrict VM mapping so synonyms map to same cache set

Summary (1)

- Memory hierarchy: Register file
 - Under compiler/programmer control
 - Complex register allocation algorithms to optimize utilization
- Memory hierarchy: Virtual Memory
 - Placement: fully flexible
 - Identification: through page table
 - Replacement: approximate LRU using PT reference bits
 - Write policy: write-back

Summary (2)

- Page tables
 - Forward page table
 - Multilevel page table
 - Inverted or hashed page table
 - Also used for protection, sharing at page level
- Translation Lookaside Buffer (TLB)
 - Special-purpose cache for PTEs