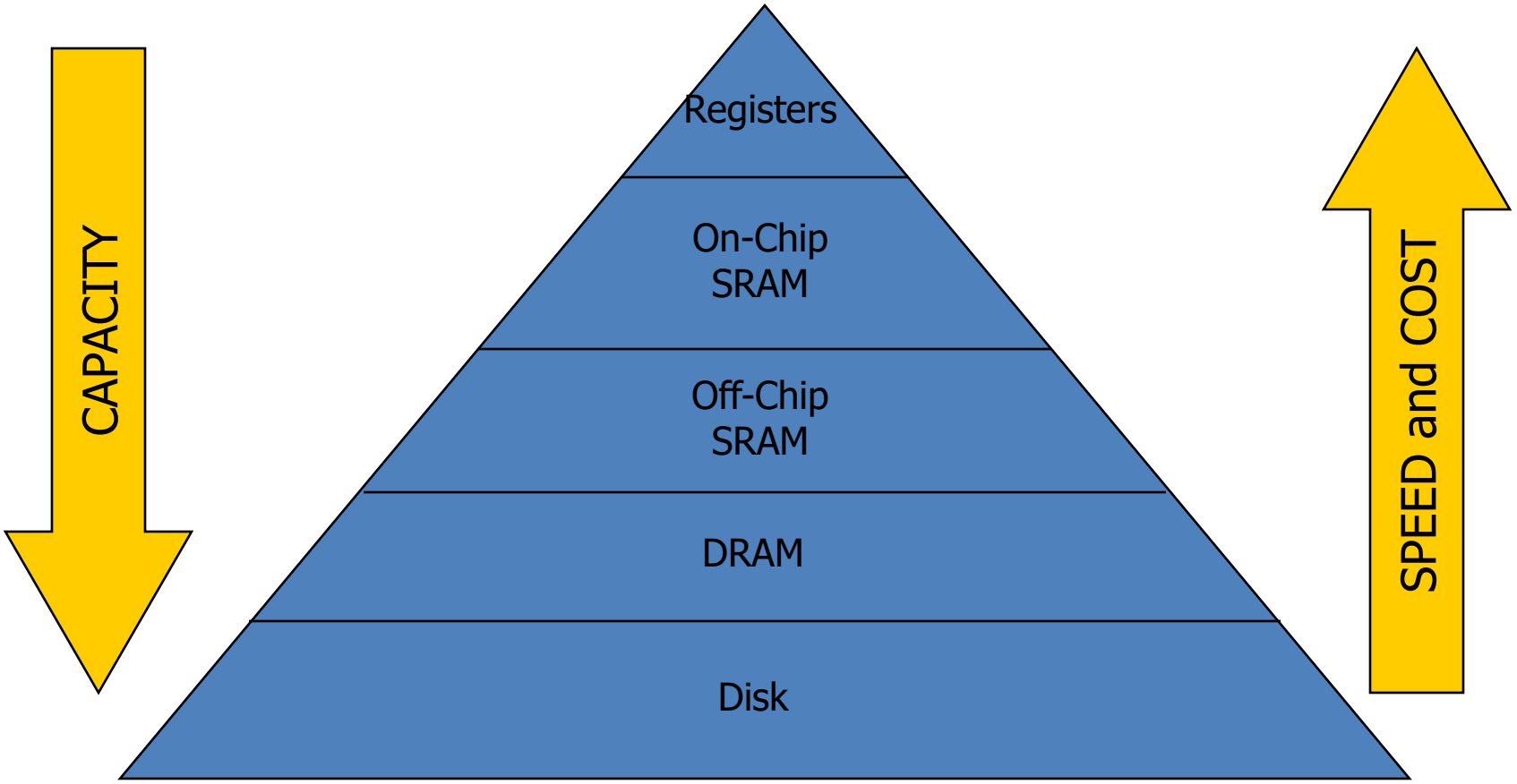
ECE/CS 552: Virtual Memory

© Prof. Mikko Lipasti

Lecture notes based in part on slides created by Mark Hill, David Wood, Guri Sohi, John Shen and Jim Smith

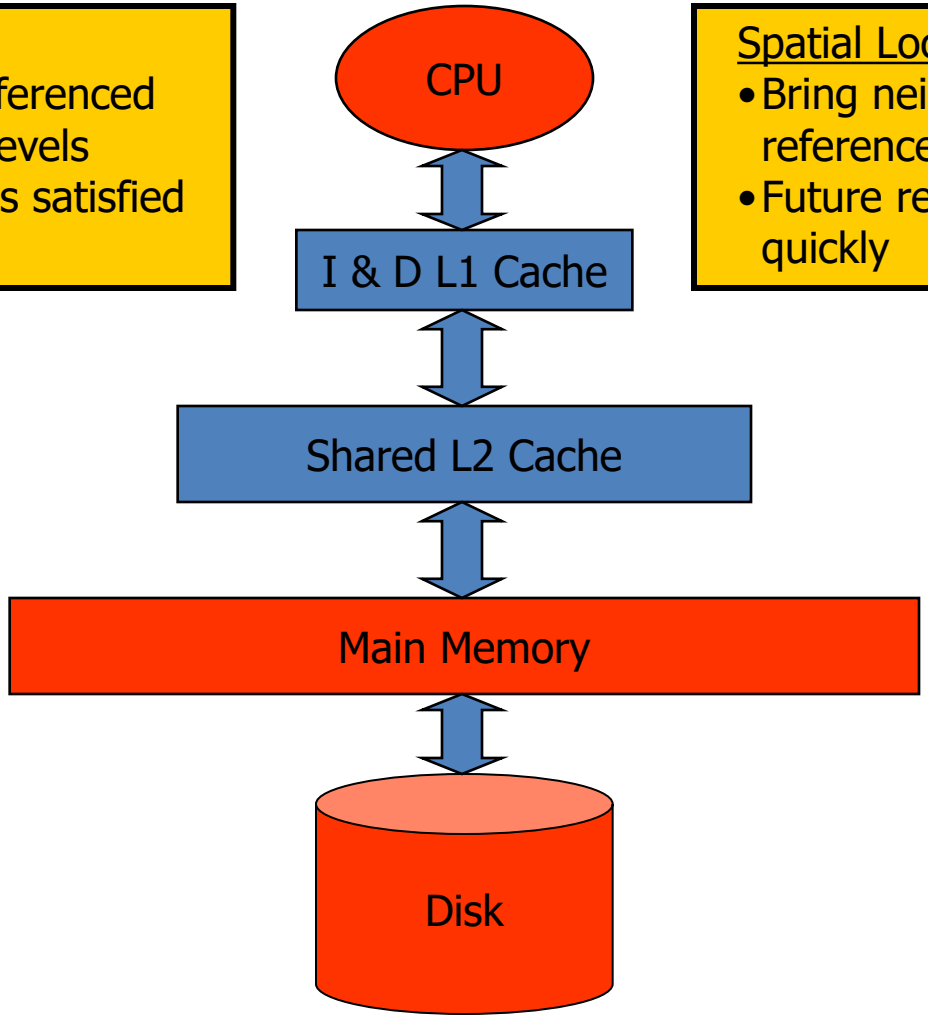
Memory Hierarchy



Memory Hierarchy

THE UNIVERSITY of WISCONSIN MADISON

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 Burning Questions

THE UNIVERSITY of WISCONSIN MADISON

- 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

Placement

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

Register File

- Registers managed by programmer/compiler
  - Assign variables, temporaries to registers
  - Limited name space matches available storage
  - Learn more in CS536, CS701

Placement	Flexible (subject to data type)
Identification	Implicit (name == location)
Replacement	Spill code (store to stack frame)
Write policy	Write-back (store on replacement)

Main Memory and Virtual Memory

- Use 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

THE UNIVERSITY of WISCONSIN MADISON

- 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

THE UNIVERSITY of WISCONSIN MADISON

- Main memory managed in larger blocks
  - *Page size* typically 4K – 16K
- 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; based on register values
  - 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
  - CS537

Virtual Memory – Write Policy

THE UNIVERSITY of WISCONSIN MADISON

- 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 engineered differently

Page Faults

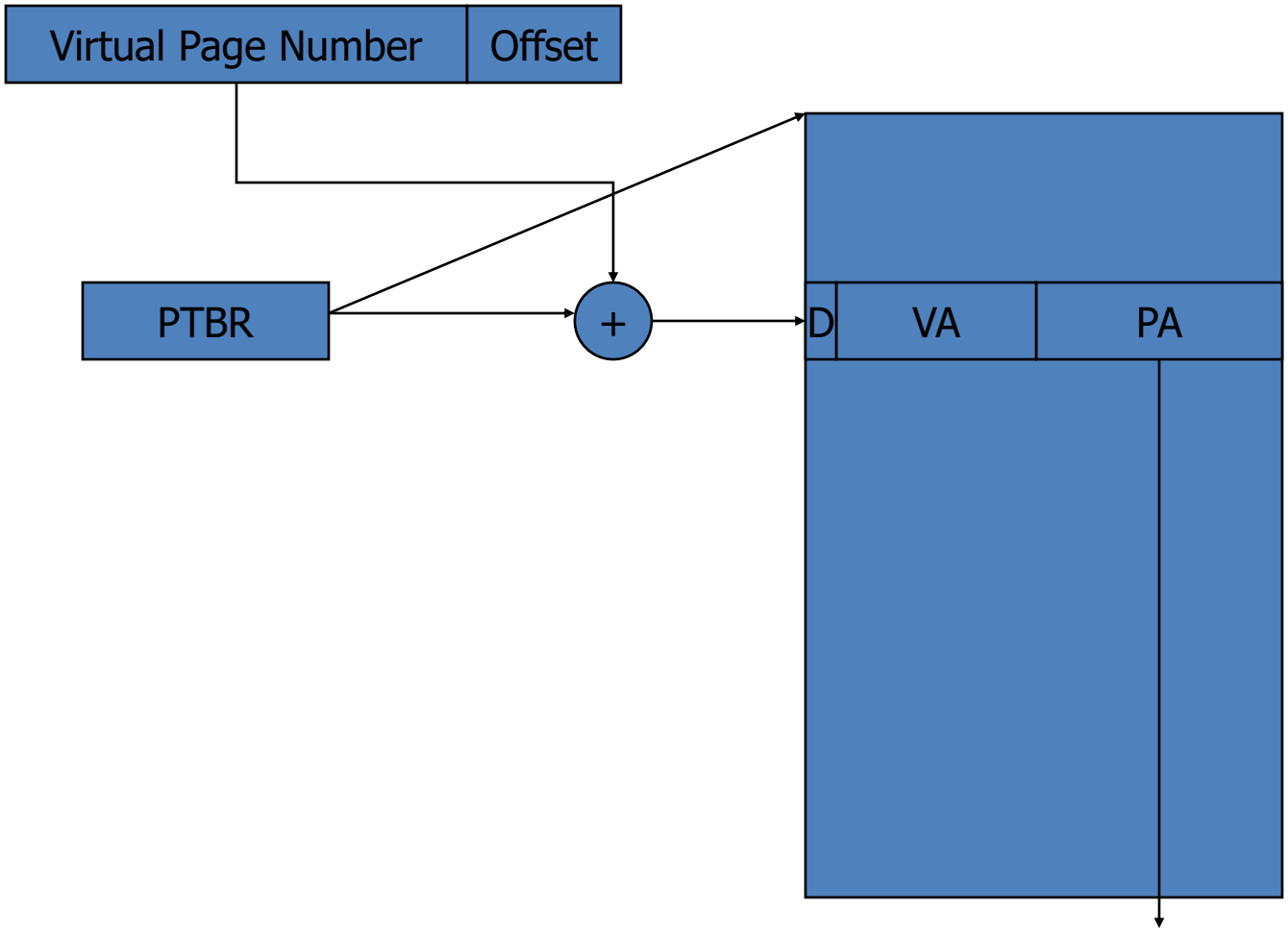
- A virtual memory miss is a page fault
  - Physical memory location does not exist
  - Exception is raised, save PC
  - Invoke OS page fault handler
    - Find a physical page (possibly evict)
    - Initiate fetch from disk
  - Switch to other task that is ready to run
  - Interrupt when disk access 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?
  - &PTE = PTBR + page number * sizeof(PTE)
  - PTBR is private for each program
    - Context switch replaces PTBR contents

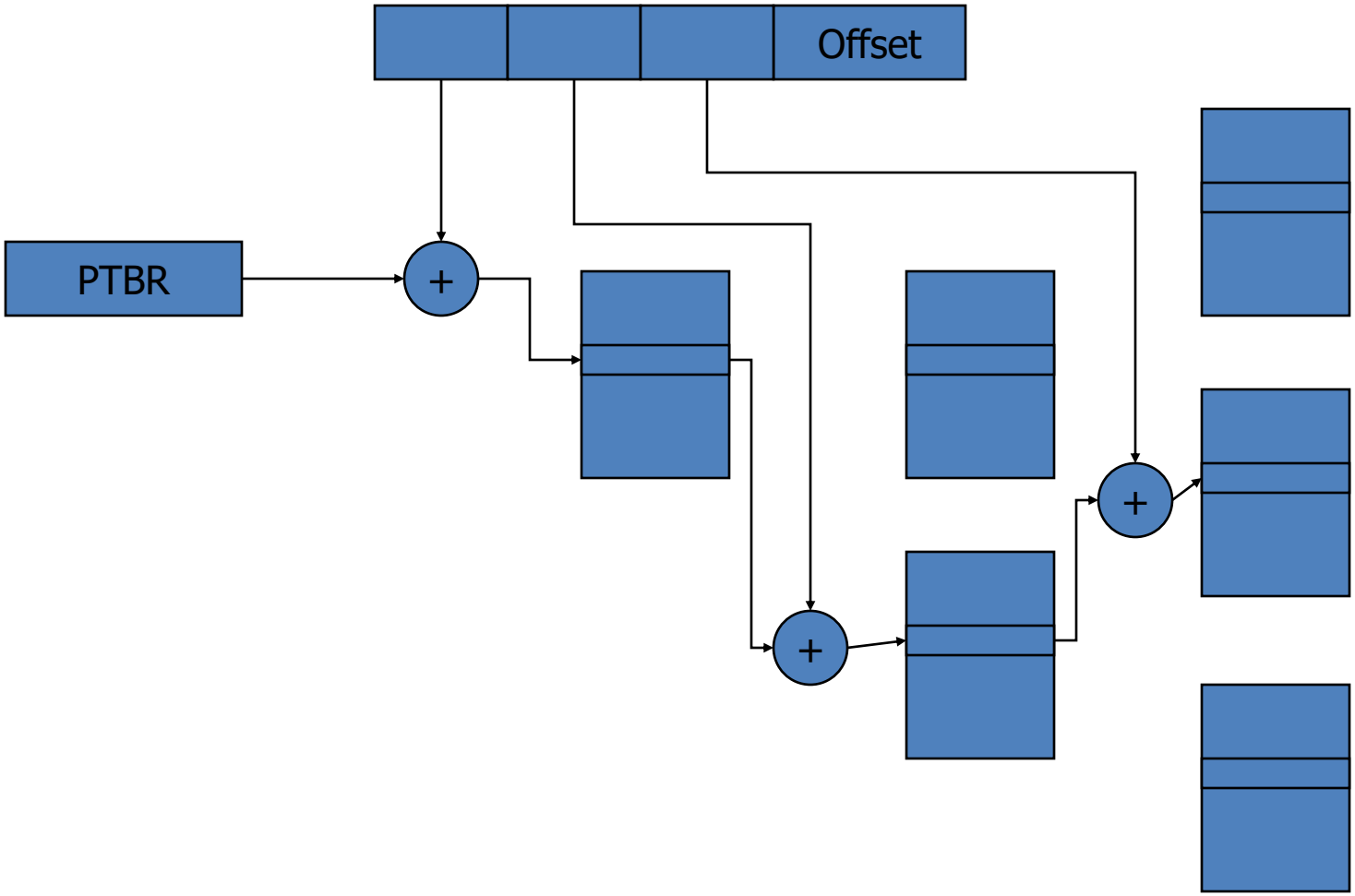
Address Translation



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

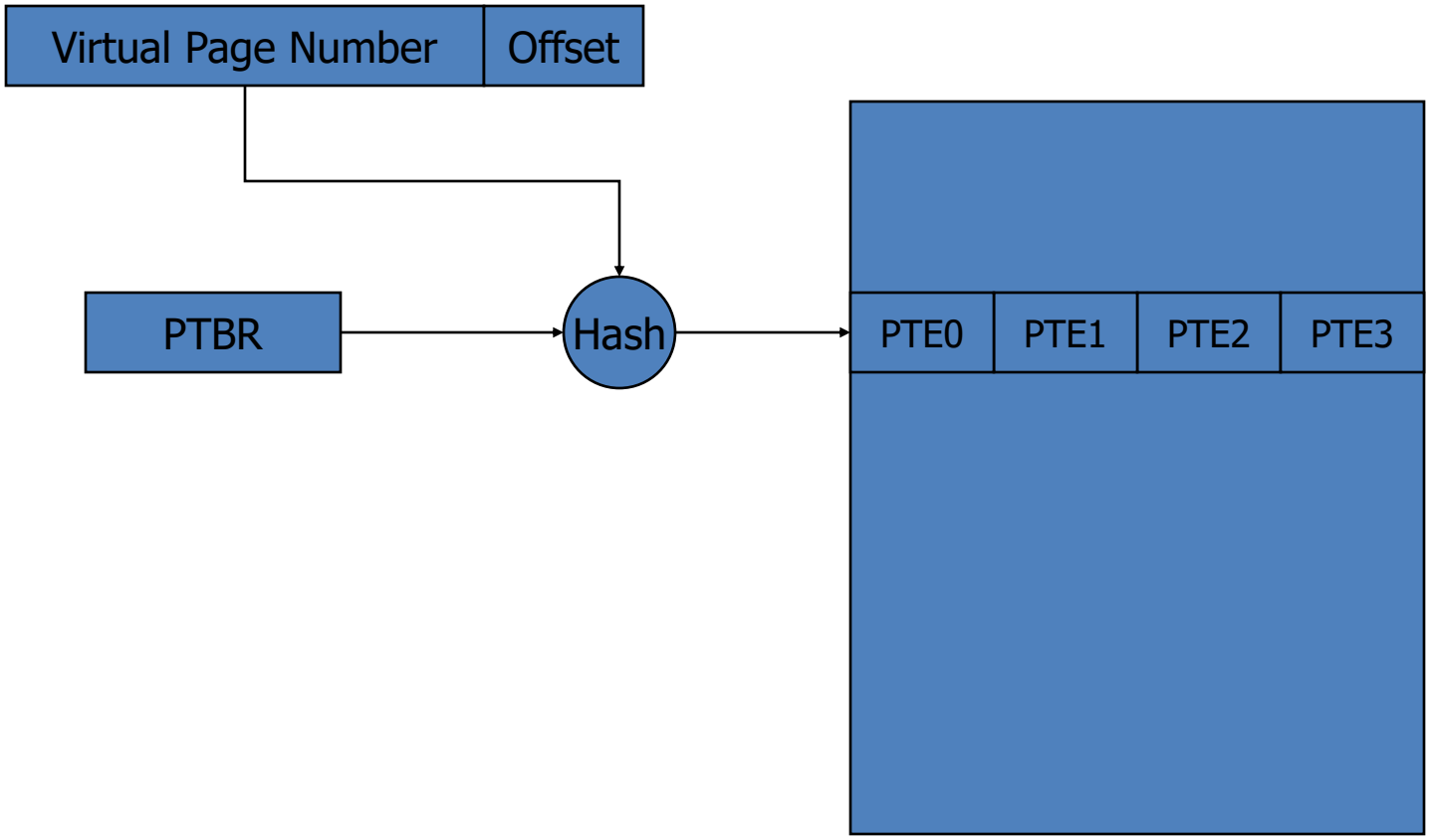
Multilevel Page Table



Hashed Page Table

- Use a hash table or inverted page table
  - PT contains an entry for each real address
    - Instead of entry for every virtual address
  - Entry is found by hashing VA
  - Oversize PT to reduce collisions: #PTE = 4 x (#phys. pages)

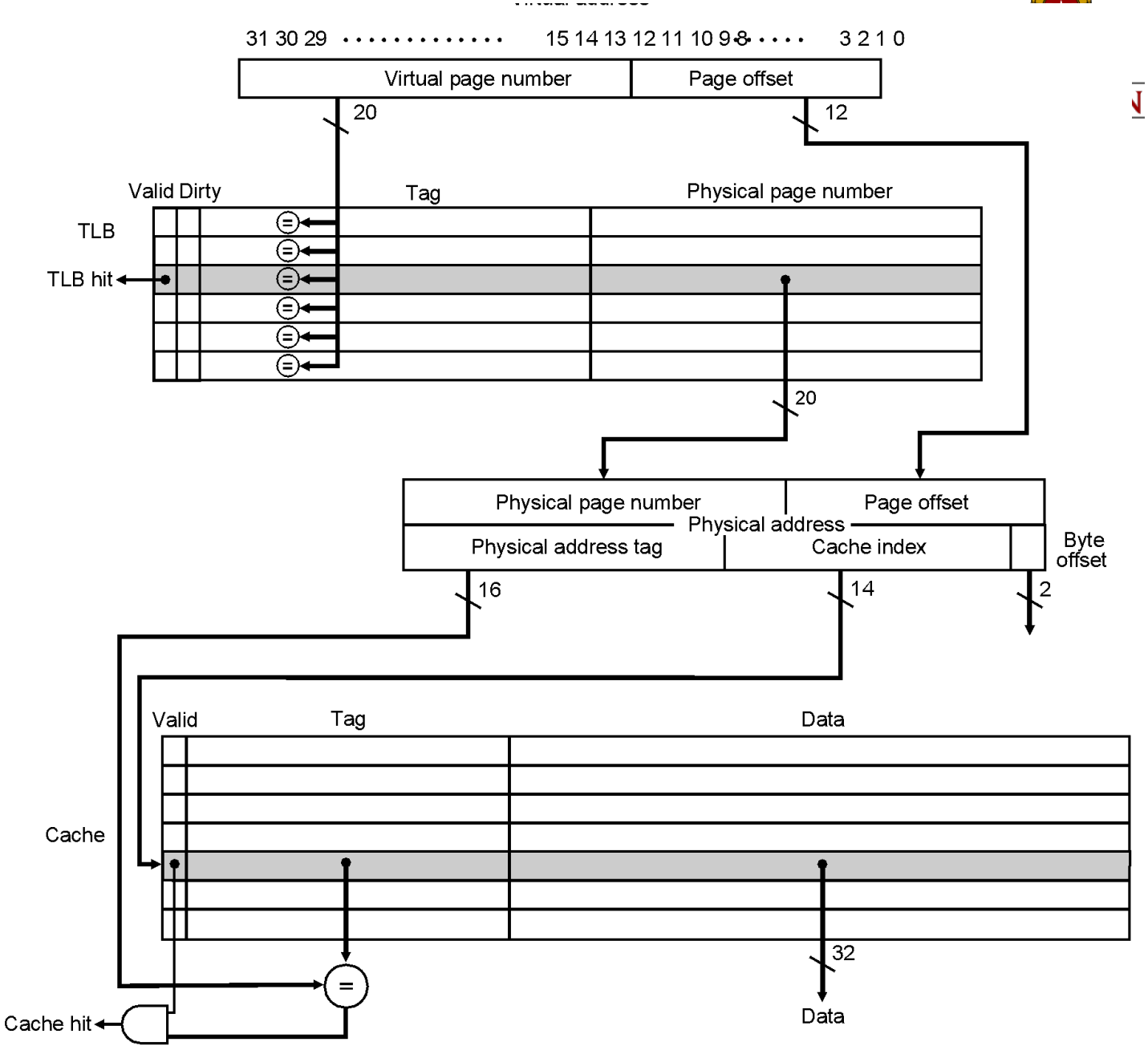
Hashed Page Table



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 cache for PTEs
  - Called TLB (translation lookaside buffer)
  - Caches PTE entries
  - Exploits temporal and spatial locality (just a cache)

TLB


31 30 29 . . . . . 15 14 13 12 11 10 9 8 . . . . 3 2 1 0
Virtual page number
Page offset
20
12
Valid Dirty
Tag
Physical page number
TLB
TLB hit
=
20
Physical page number
Page offset
Physical address
Physical address tag
Cache index
Byte offset
16
14
2
Valid
Tag
Data
Cache
32
=
Data
Cache hit

Virtual Memory Protection

- Each process/program has private virtual address space
  - Automatically protected from rogue 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

THE UNIVERSITY of WISCONSIN MADISON

- 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 synonym problem

VA Synonyms

- Virtually-addressed caches are desirable
  - No need to translate VA to PA before cache lookup
  - Faster hit time, translate only on misses
- However, VA synonyms cause problems
  - Can end up with two copies of same physical line
- Solutions:
  - Flush caches/TLBs on context switch
  - Extend cache tags to include PID & prevent duplicates
    - Effectively a shared VA space (PID becomes part of address)

Summary

- 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

- 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