COEN-4730/EECE-5730 Computer Architecture

# Lecture 3 Review of Caches and Virtual Memory

*Cristinel Ababei*
*Dept. of Electrical and Computer Engineering*
MARQUETTE UNIVERSITY
BE THE DIFFERENCE.

*Credits: Slides adapted from presentations of Pasricha, Kubiatowicz, Patterson, Mutlu, Elsevier*

1

1

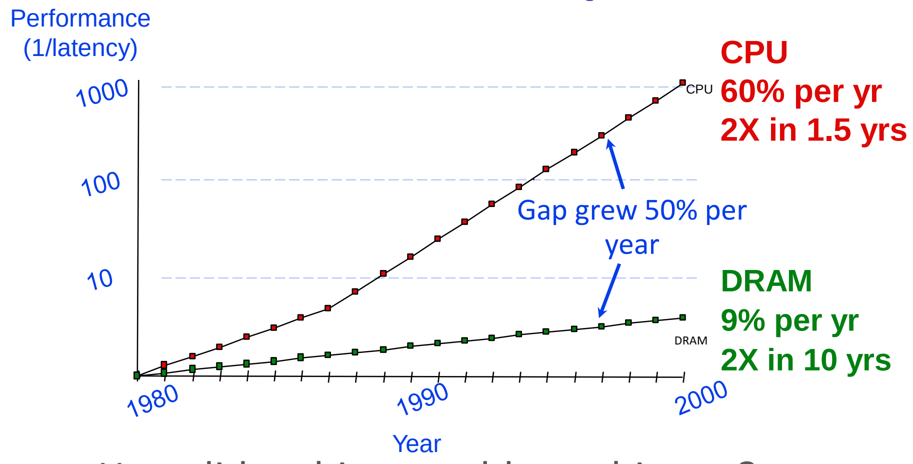
## Outline

- Memory hierarchy, example, terminology
- 4 questions
- 6 basic cache optimizations
- Virtual memory
- Summary

2

2

## Since 1980, CPU has outpaced DRAM ...



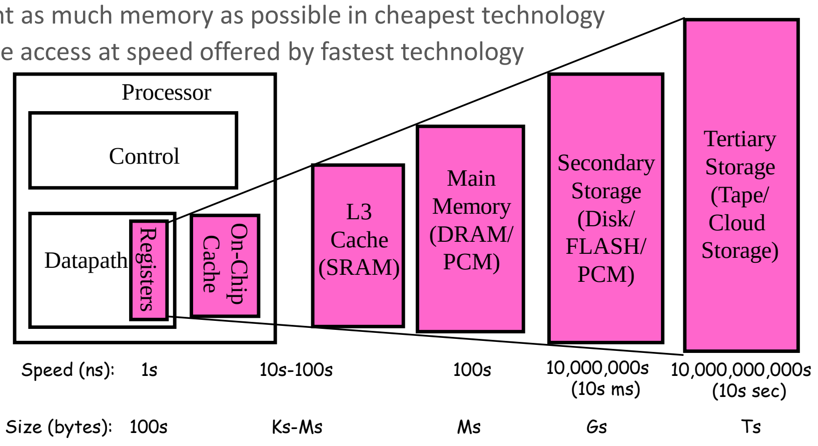
- How did architects address this gap?
  - Put small, fast "cache" memories between CPU and DRAM
  - Create a "memory hierarchy"

3

3

## Memory Hierarchy

- Everything is a cache for something else
- Take advantage of the principle of locality to:
  - Present as much memory as possible in cheapest technology
  - Provide access at speed offered by fastest technology



4

4

## Example of Memory Hierarchy: Apple iMac G5

Managed by compiler → Reg

Managed by hardware → L1 Inst, L1 Data, L2, DRAM

Managed by OS, hardware, application → Disk

| | Reg | L1 Inst | L1 Data | L2 | DRAM | Disk |
|---|---|---|---|---|---|---|
| Size | 1K | 64K | 32K | 512K | 256M | 80G |
| Latency Cycles, Time | 1, 0.6 ns | 3, 1.9 ns | 3, 1.9 ns | 11, 6.9 ns | 88, 55 ns | $10^7$, 12 ms |

iMac G5
1.6 GHz

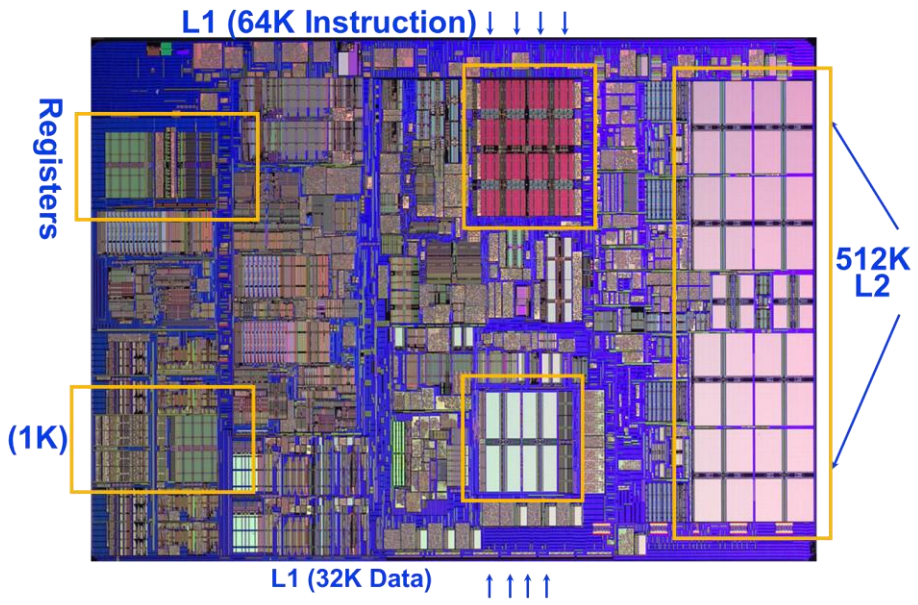
Goal: Illusion of large, fast, cheap memory

Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access.

5

5

## iMac's PowerPC 970: all caches on-chip

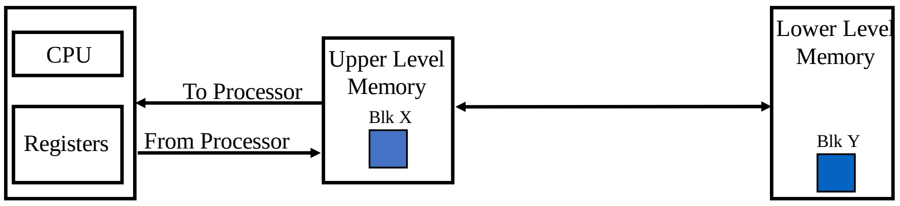


6

6

## Memory Hierarchy: Terminology

- Hit: data appears in some block in the upper level (example: Block X)
  - Hit Rate: Fraction of memory access found in the upper level
  - Hit Time: Time to access the upper level which consists of:
    Time to determine hit/miss + Memory access time
- Miss: data needs to be retrieved from a block in the lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level +
    Time to deliver the block to the processor
- Hit Time << Miss Penalty (500 instructions on 21264!)



## Program Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - E.g., instructions in a loop
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

## Outline

- Memory hierarchy, example, terminology
- 4 questions
- 6 basic cache optimizations
- Virtual memory
- Summary

9

9

## 4 Questions for Memory Hierarchy

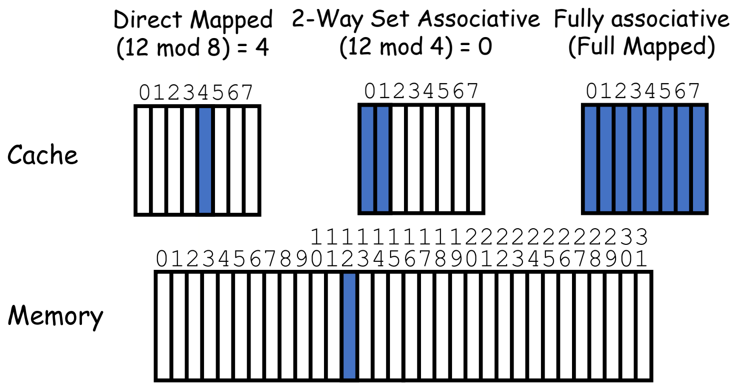
- Q1: Where can a block be placed in the upper level?
  *(Block placement)*
- Q2: How is a block found if it is in the upper level?
  *(Block identification)*
- Q3: Which block should be replaced on a miss?
  *(Block replacement)*
- Q4: What happens on a write?
  *(Write strategy)*

10

10

## Q1: Where can a block be placed in the upper level?

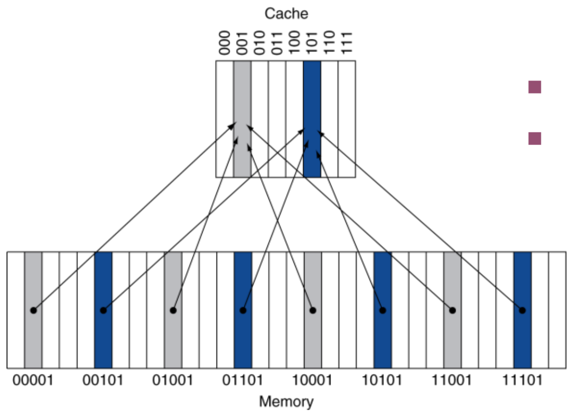
- Block 12 placed in 8 block cache:
  - Direct mapped, 2-way set associative (SA), Fully associative
  - S.A. Mapping = Block Number MODULO Number Sets



11

## Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

12

## Sources of Cache Misses

- Compulsory (cold start or process migration, first reference): first access to a block
  - "Cold" fact of life: not a whole lot you can do about it
  - Note: If you are going to run "billions" of instruction, Compulsory Misses are insignificant
- Capacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- Coherence (invalidation): other process (e.g., I/O) updates memory

13

13

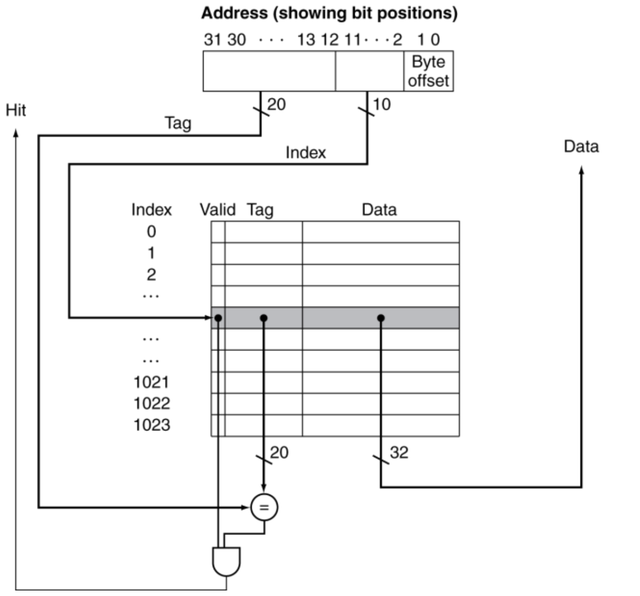
## Q2: How is a block found if it is in the upper level?

- Block is minimum quantum of caching
  - Data select field used to select data within block
- Index Used to Lookup Candidates in Cache
  - Index identifies the set
- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
  - If no candidates match, then declare cache miss
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

14

14

## Address Subdivision

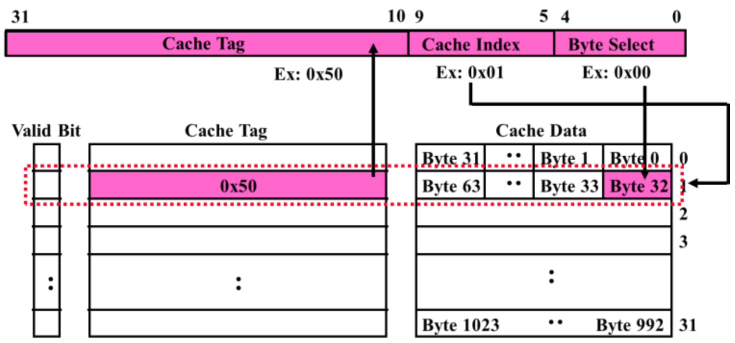


15

15

## Review: Direct Mapped Cache

- Direct Mapped $2^N$ byte cache:
  - The uppermost (32 - N) bits are always the Cache Tag
  - The lowest M bits are the Byte Select (Block Size = $2^M$)
- Example: 1 KB Direct Mapped Cache with 32 B Blocks
  - Index chooses potential block
  - Tag checked to verify block
  - Byte select chooses byte within block

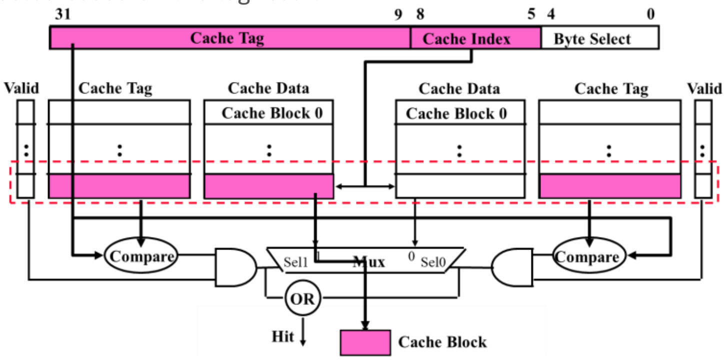


16

16

## Review: Set Associative (SA) Cache

- N-way set associative: N entries (blocks) per Cache Index
  - N direct mapped caches operate in parallel
- Example: Two-way Set Associative cache
  - Cache Index selects a "set" from the cache
  - Two tags in the set are compared to input in parallel
  - Data is selected based on the tag result

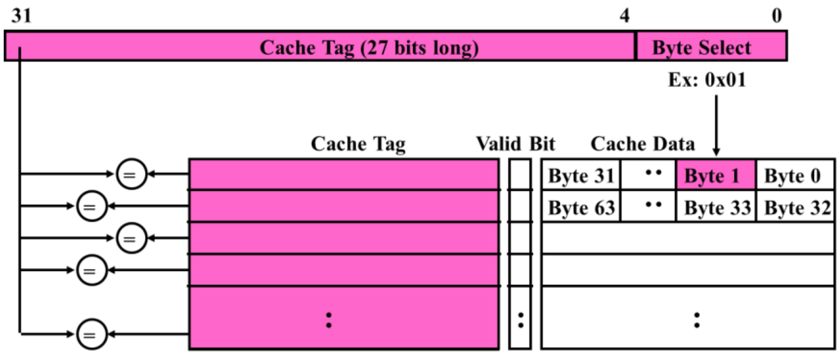


17

17

## Review: Fully Associative Cache

- Fully Associative: Every cache entry can hold/store any block/line
  - Address does not include a cache index
  - Compare Cache Tags of all Cache Entries in Parallel
- Example: Block Size = 32 B blocks
  - We need N 27-bit comparators
  - Still have byte select to choose from within block



18

18

## Q3: Which block should be replaced on a miss?

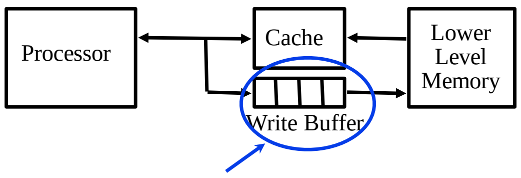
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - LRU (Least Recently Used): Appealing, but hard to implement for high associativity
  - Random: Easy, but – how well does it work?
    - Miss rates:

| Assoc: | 2-way | | 4-way | | 8-way | |
|---|---|---|---|---|---|---|
| **Size** | **LRU** | **Ran** | **LRU** | **Ran** | **LRU** | **Ran** |
| 16K | 5.2% | 5.7% | 4.7% | 5.3% | 4.4% | 5.0% |
| 64K | 1.9% | 2.0% | 1.5% | 1.7% | 1.4% | 1.5% |
| 256K | 1.15% | 1.17% | 1.13% | 1.13% | 1.12% | 1.12% |

## Q4: What happens on a Write?

| | Write-Through | Write-Back |
|---|---|---|
| **Policy** | Data written to cache block also written to lower-level memory | • Write data only to the cache block • Update lower level when a block falls out of the cache |
| **Debug** | Easy | Hard |
| **Do read misses produce writes?** | No | Yes |
| **Do repeated writes make it to lower level?** | Yes | No |

## Write Buffers for Write-Through Caches



Holds data awaiting write-through to lower-level memory

| | |
|---|---|
| Q. Why a write buffer? | A. So CPU doesn’t stall |
| Q. Why a buffer, why not just one register? | A. Bursts of writes are common |
| Q. Are Read After Write (RAW) hazards an issue for write buffer? | A. Yes! Drain buffer before next read, or check write buffers for match on reads |

21

21

## Outline

- Memory hierarchy, example, terminology
- 4 questions
- 6 basic cache optimizations
- Virtual memory
- Summary

22

22

## 6 Basic Cache Optimizations

- Reducing Miss Rate
  1. Larger Block size (compulsory misses)
  2. Larger Cache size (capacity misses)
  3. Higher Associativity (conflict misses)
- Reducing Miss Penalty
  4. Multilevel Caches to reduce miss penalty
- Reducing Hit Time
  5. Giving Reads priority over Writes
  6. Avoid address translation during indexing of Cache

## Outline

- Memory hierarchy, example, terminology
- 4 questions
- 6 basic cache optimizations
- Virtual memory
- Summary

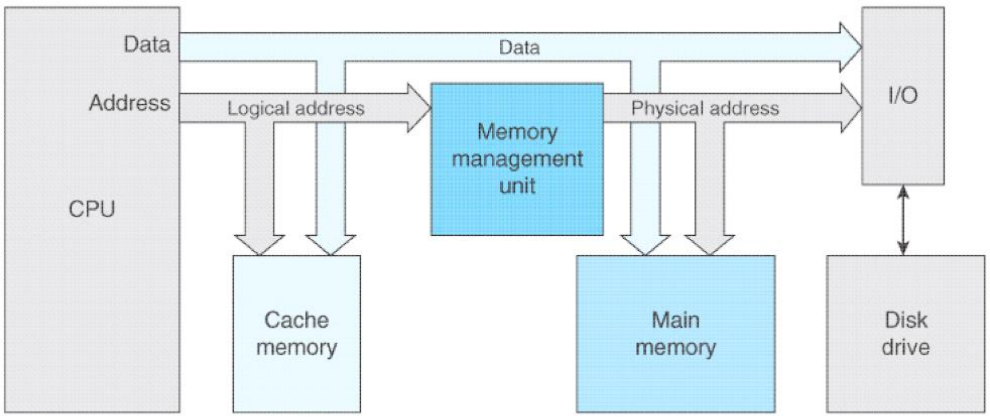
## Virtual Memory (VM)

- A memory management approach
  - Programmer views memory as large address space (larger than physical memory) without concerns about the amount of physical memory or memory management
- Main idea: Use main **memory as a "cache" for secondary** (disk) storage
  - Managed jointly by CPU and the operating system (OS)
  - VM "block" is called a page; Typical size of a page: 1-8K
  - VM translation "miss" is called a page fault
- Programs share main memory
  - Each gets a private virtual address space holding its frequently used code and data
  - Protected from other programs
- CPU and OS translate virtual addresses to physical addresses

25

25

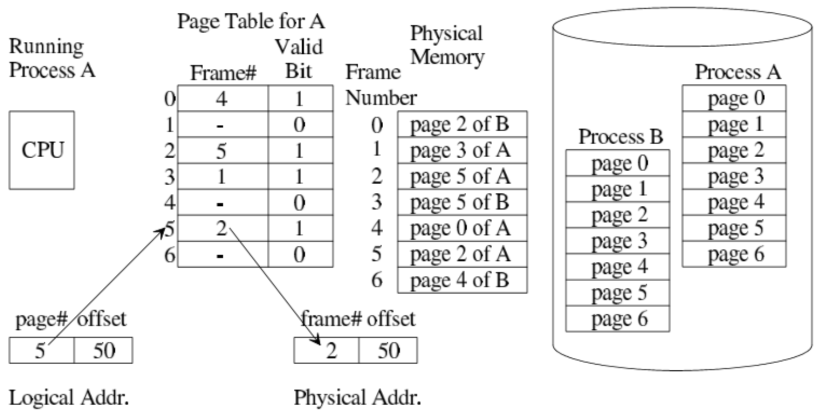
## Cache and Virtual Memory



26

26

## Memory Management Unit (MMU) for Paging

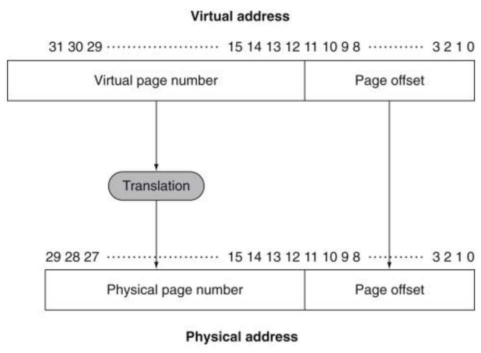


**Notes:**
- Virtual (Logical) memory is organized into Pages (or virtual page).
- Physical memory is organized into Page Frames (or physical page).
- The size of Page matches a Page Frame.

27

## Virtual Address to Physical Address

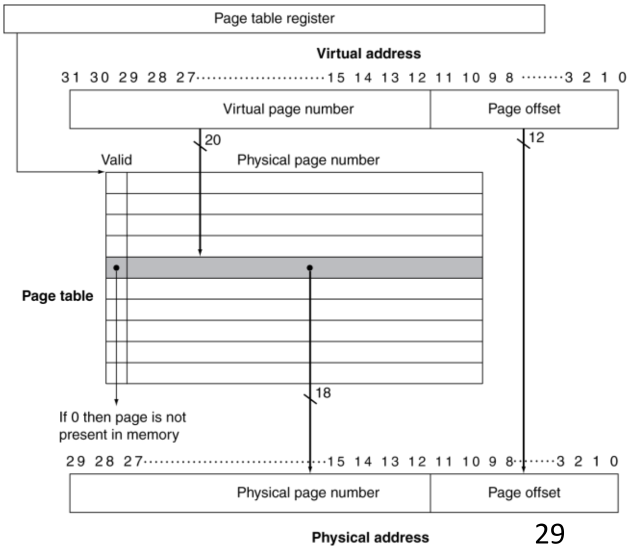
- In virtual memory, blocks of memory (called pages) are mapped from one set of addresses (called virtual addresses) to another set (called physical addresses).
- The processor generates virtual addresses while the memory is accessed using physical addresses. Both the virtual memory and the physical memory are broken into pages, so that a virtual page is mapped to a physical page.
- It is possible for a virtual page to be absent from main memory and not be mapped to a physical address; in that case, the page resides on disk.
- Physical pages can be shared by having two virtual addresses point to the same physical address. This capability is used to allow two different programs to share data or code.



28

## Translation Using a Page Table (PT)

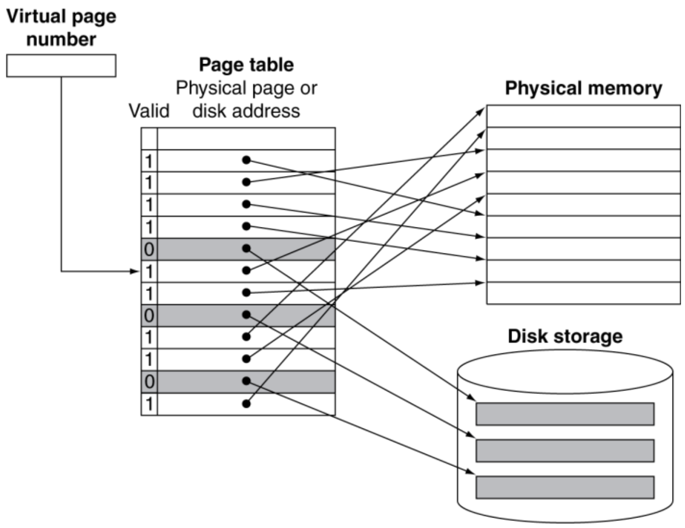
- The page table is indexed with the virtual page number to obtain the corresponding portion of the physical address



29

29

## Mapping Pages to Storage



30

30

## Storage of Page Tables Issues

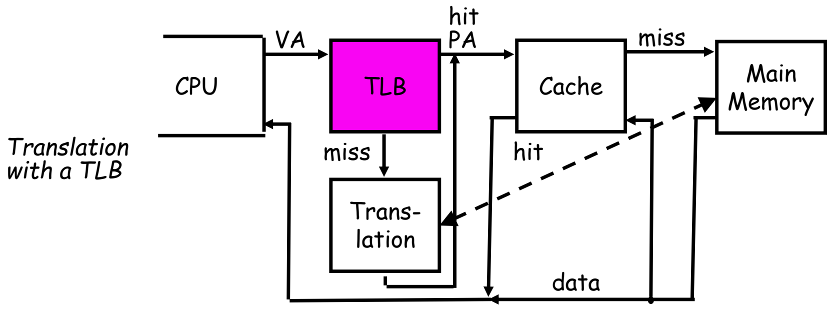
- If in physical memory, each memory reference in the program results in 2 memory accesses:
  - One for page table entry
  - Another to perform desired memory access
- Solution: TLB (Translation Lookaside Buffer) – small cache to hold PT entries

31

31

## Translation Lookaside Buffers (TLB)

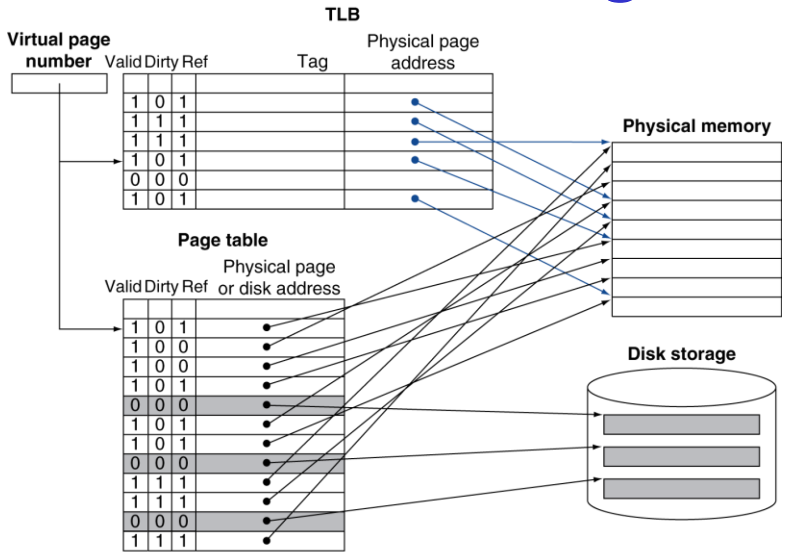
- Cache applied to address translations
- Fully Associative, Set Associative, or Direct Mapped



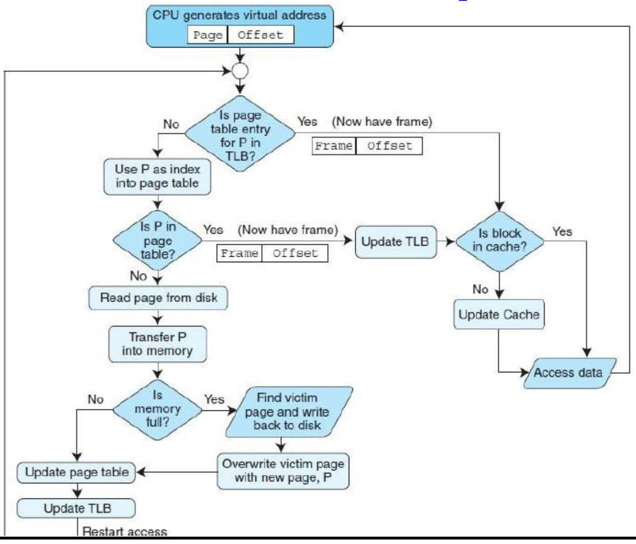
32

32

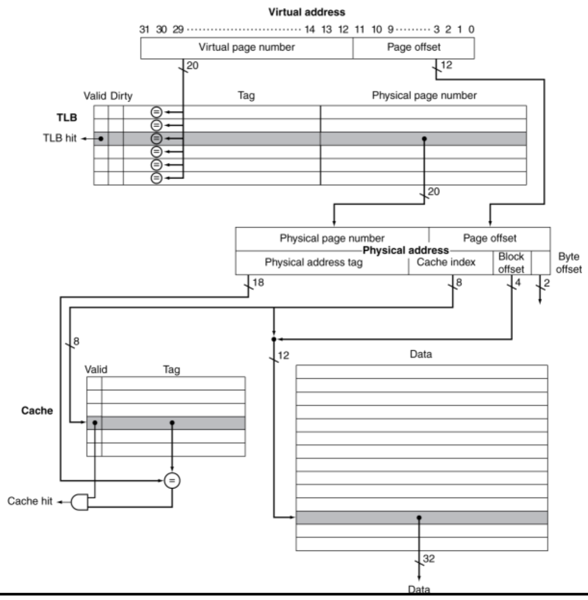
## Fast Translation Using a TLB



## Putting it All Together: TLB, Page Table, Cache, Main Memory



## TLB and Cache Interaction



35

35

## Three Advantages of Virtual Memory

1. Translation:
   - Program can be given consistent view of memory, even though physical memory is scrambled
   - Makes multithreading reasonable
   - Only the most important part of program ("Working Set") must be in physical memory.
   - Contiguous structures (like stacks) use only as much physical memory as necessary yet still grow later.
2. Protection:
   - Different threads (or processes) protected from each other.
   - Different pages can be given special behavior
     - (Read Only, Invisible to user programs, etc.)
   - Kernel data protected from User programs
   - Very important for protection from malicious programs
3. Sharing:
   - Can map same physical page to multiple users (i.e., processes or programs) ("Shared memory")

36

36

## Outline

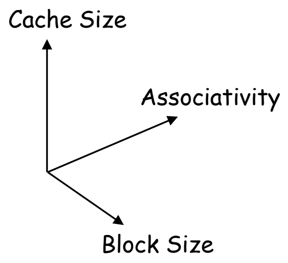
- Memory hierarchy, example, terminology
- 4 questions
- 6 basic cache optimizations
- Virtual memory
- Summary

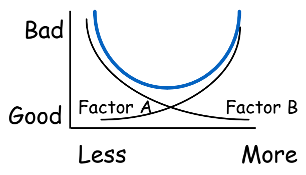
37

37

## Summary #1/3: The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs. write-back
- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology/cost
- Simplicity often wins





38

38

## Summary #2/3: Caches

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
    - Temporal Locality: Locality in Time
    - Spatial Locality: Locality in Space
- Three Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Capacity Misses: increase cache size
  - Conflict Misses: increase cache size and/or associativity.
- Write Policy: Write Through vs. Write Back
- Today CPU time is a function of (ops, cache misses) vs. just of (ops): affects Compilers, Data structures, and Algorithms

39

39

## Summary #3/3: Virtual Memory (VM)

- Page tables map virtual address to physical address
- TLBs are important for fast translation
- TLB misses are significant in processor performance
- Caches, TLBs, Virtual Memory all understood by examining how they deal with 4 questions:
  1) Where can block be placed?
  2) How is block found?
  3) What block is replaced on miss?
  4) How are writes handled?
- Today VM allows many processes to share single memory without having to swap all processes to disk; today VM protection is paramount!

40

40