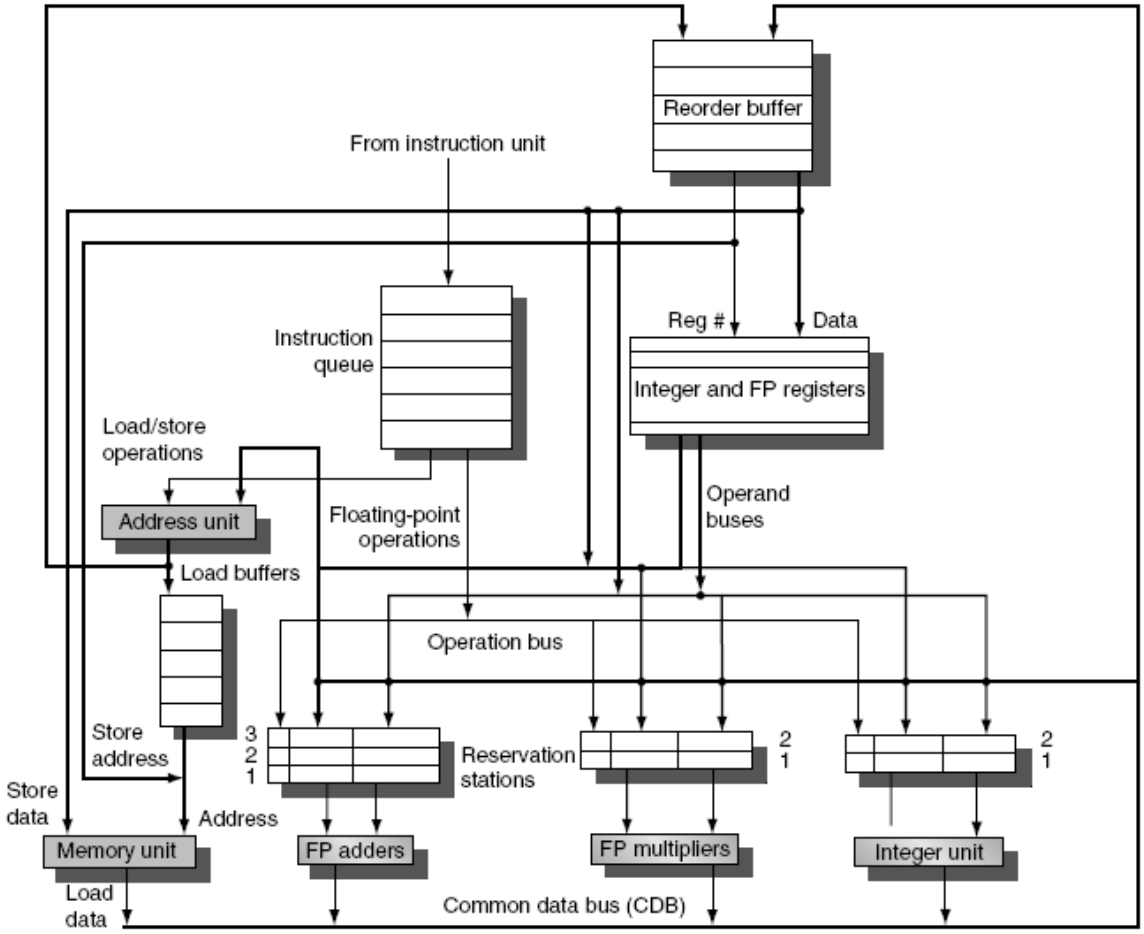
ENGG 5101
Advanced Computer Architecture

Lecture 06 – Memory Hierarchies

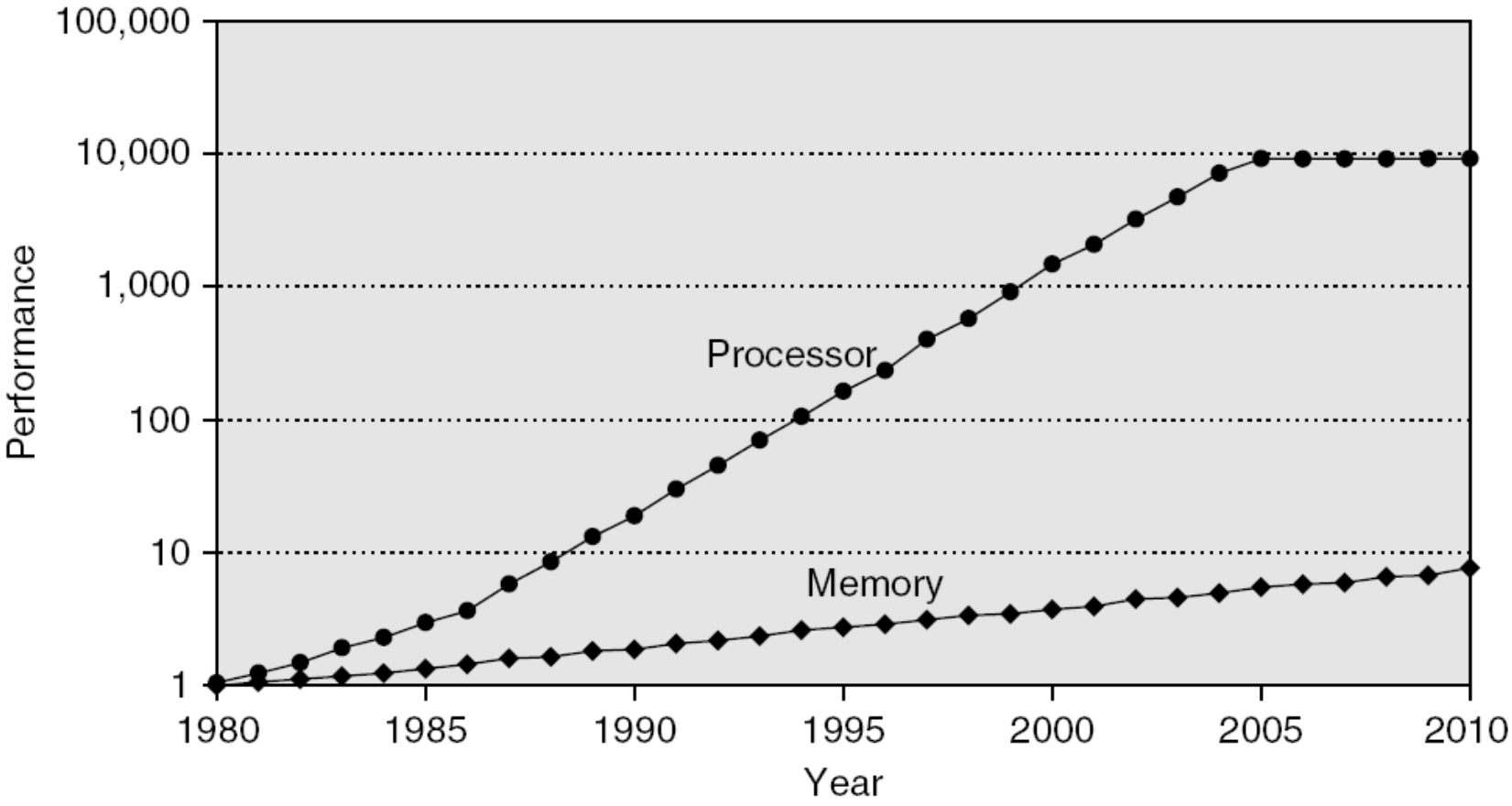
XU, Qiang (Johnny) 徐強

Recap

- Modern microarchitectures:
  - Dynamic scheduling + multiple issue + speculation



Memory Performance Gap



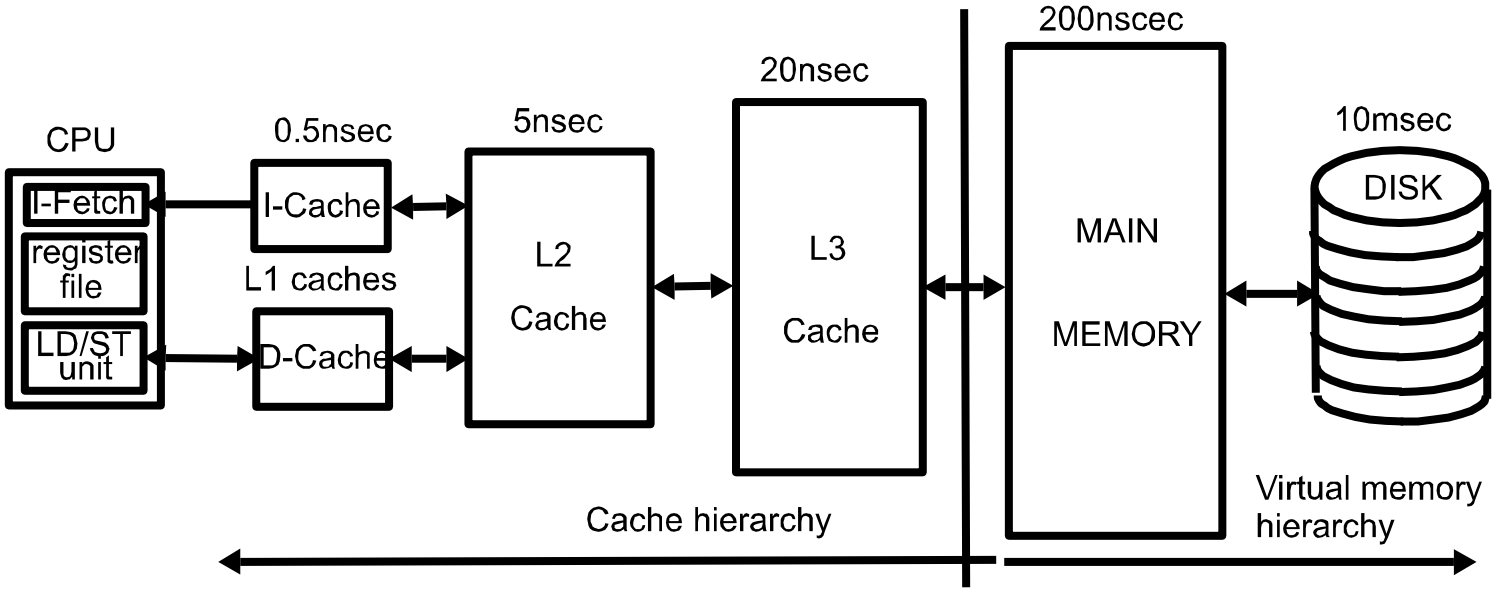
Memory Bandwidth Gap

- **Memory bandwidth becomes more crucial with multi-core processors:**
  - **Intel Core i7 can generate two references per core per clock**
  - **Four cores and 3.2 GHz clock, aggregated peak rate is,**
    - » 25.6 billion 64-bit data references/second +
    - » 12.8 billion 128-bit instruction references
    - » = 409.6 GB/s!
  - **DRAM bandwidth is only 6% of this (25 GB/s)**

Memory Hierarchy

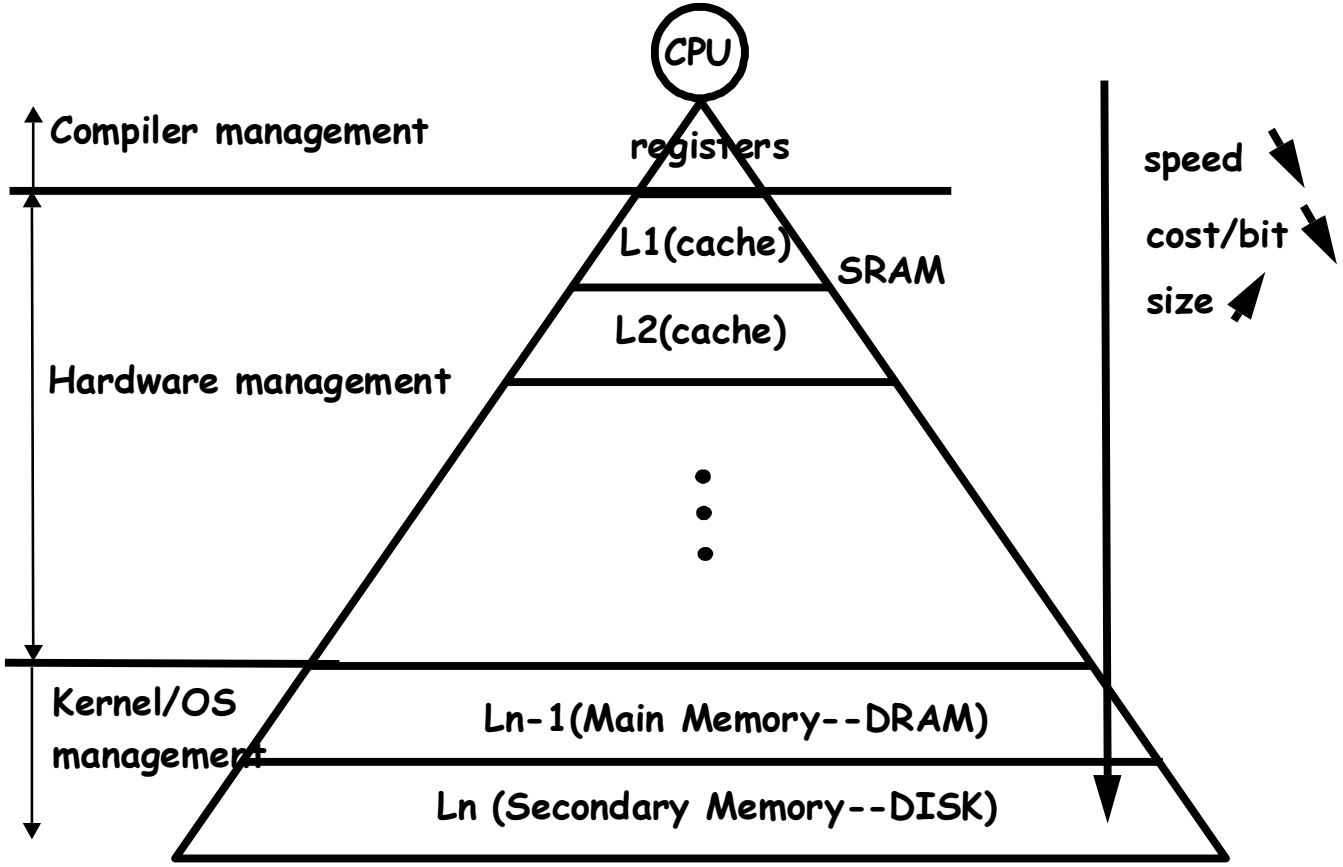
- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: exploit the principle of locality and organize memory system into a hierarchy
  - A program accesses a relatively small portion of the address space at a time
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor

Typical Memory Hierarchy



- Two types of locality:
  - Temporal locality: If an item is referenced, it will tend to be referenced again soon
  - Spatial locality: If an item is referenced, items whose address are close tend to be referenced soon
  - Spatial locality turns to temporal locality in blocks/pages

Pyramid of Memory Levels

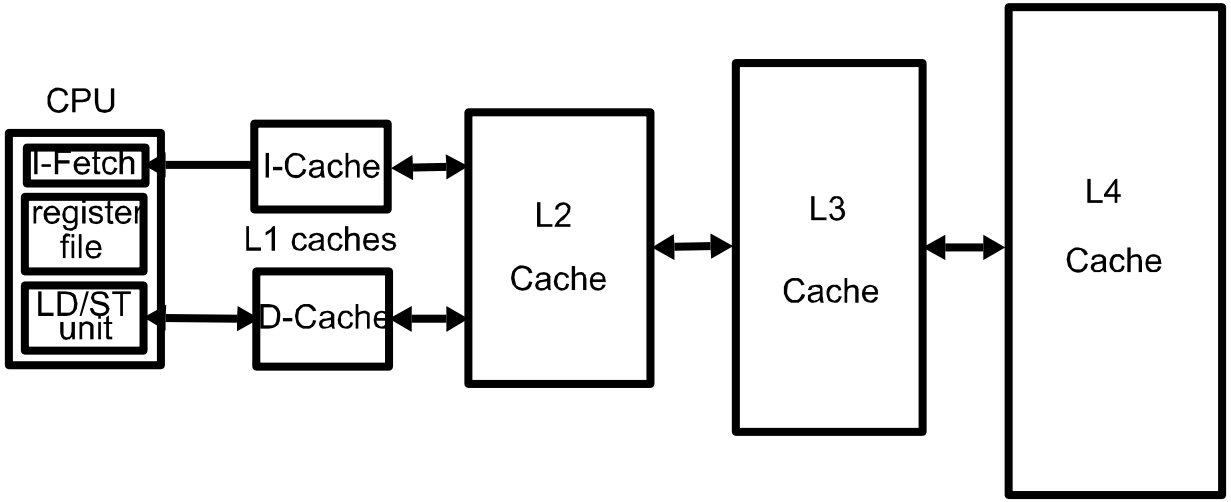


GOALS: HIGH SPEED, LOW COST, HIGH CAPACITY

INCLUSION

COHERENCE

Memory Inclusion



- 1st level and 2nd level are on-chip, 3rd and 4th levels are mostly off-chip
- Usually memory inclusion is maintained
  - When a block misses in L1, then it must be brought into all levels
  - When a block is replaced in Level i, then it must be removed from all level j with j<i

Memory Hierarchy Basics

- When a word is not found in the cache, a miss occurs:
  - Fetch word from lower level (another cache or main memory) in hierarchy, requiring a higher latency reference
  - Also fetch the other words contained within the block
    - » Takes advantage of spatial locality
  - Place block into cache in any location within its set, determined by address
    - » block address MOD number of sets
    - » A cache line can host a memory block at any given time
- Memory coherence in single-core systems
  - A load must return the value of the previous store (in process order) to the same address

Cache Performance

- Average Memory Access Time (AMAT)

  AMAT= hit time + miss rate x miss penalty

- Miss rate: Fraction of cache access that result in a miss
  - Hit Rate = 1 – miss rate

- Miss penalty: Average delay per miss caused by the processor
  - If processor blocks on misses, it is simply the number of cycles to bring a block in
  - For an OoO processor, the penalty of a miss cannot be measured directly

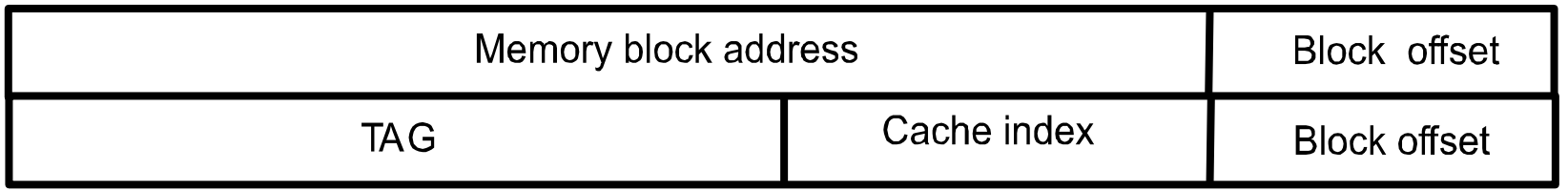
- Miss rate and penalty can be defined at every cache level

Cache Mapping

- There are 3 types of cache mapping between a memory block and a cache line
  - Direct-mapped: each memory block can be mapped to only one cache line
  - Set-associative: each memory block can be mapped to a set of cache lines
    - » Access to each set is direct-mapped, but a block mapped to a set can reside in any line in the set
  - Fully associative: each memory block can be in any cache line
- Cache is made of directory memory + data memory, one entry per cache line
  - Directory: status (state) bits: valid, dirty, reference, cache coherence

Direct-Mapped Cache

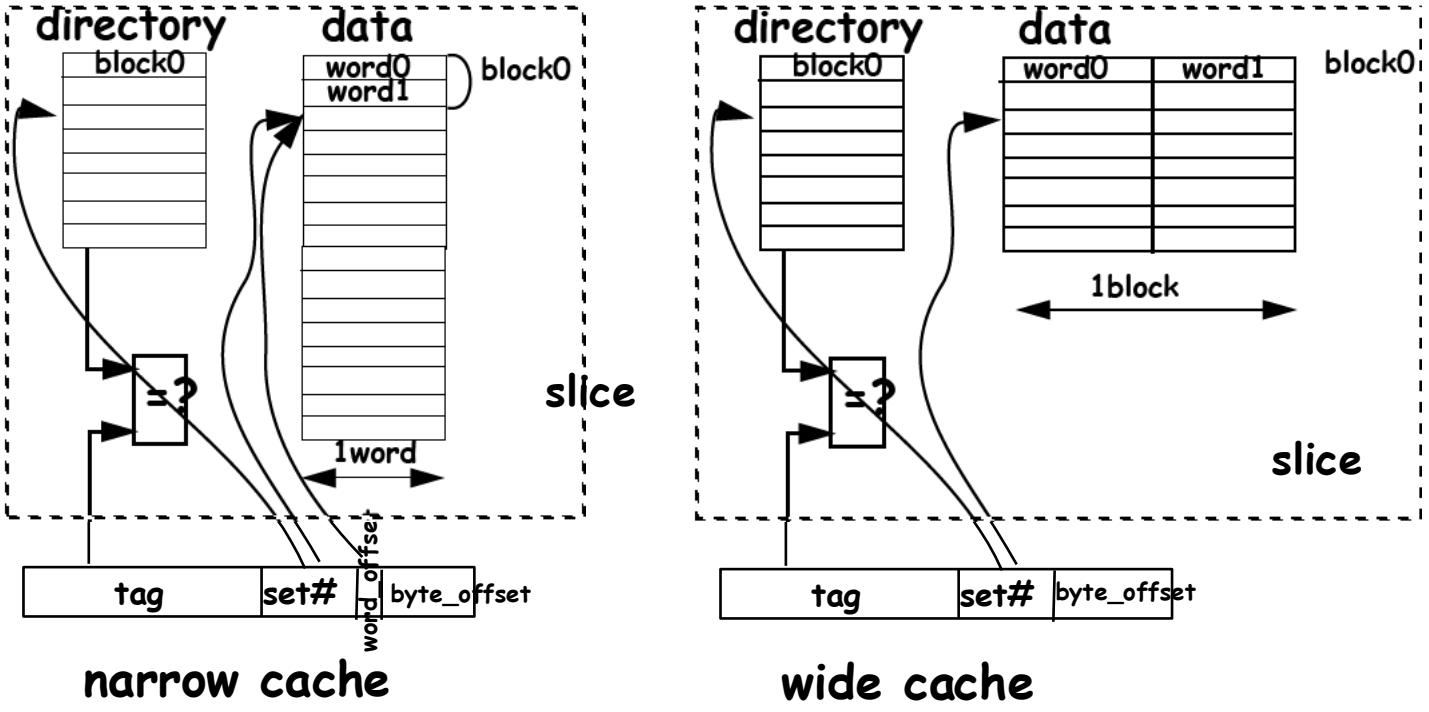
Physical Address



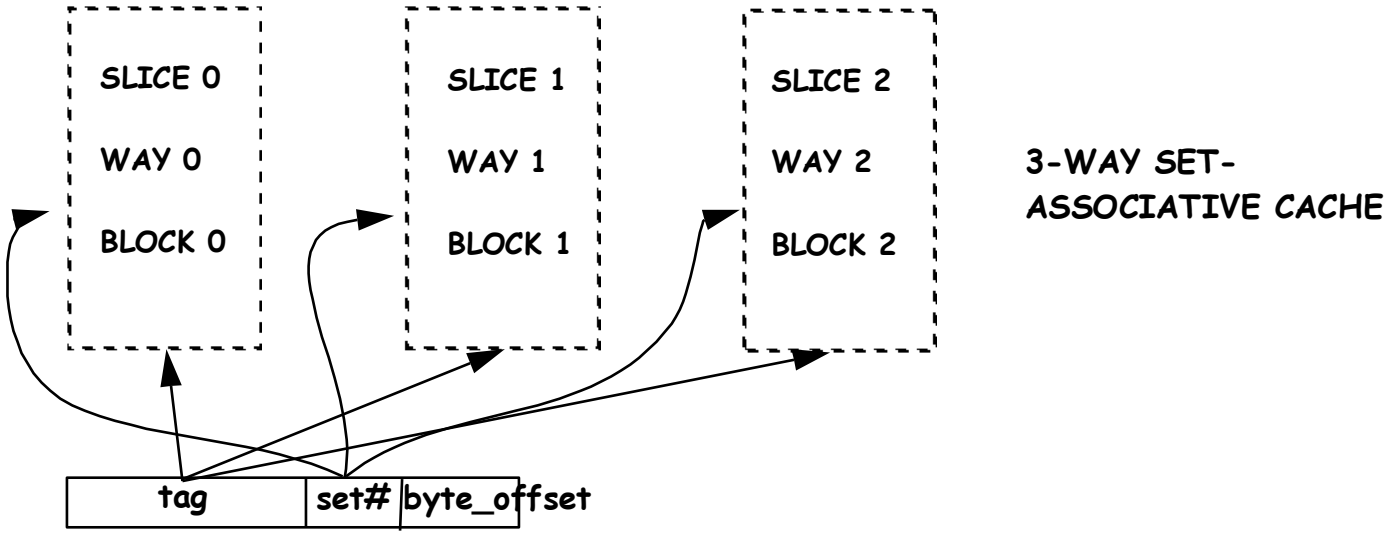
- The least significant bits of the block address usually serve as cache index
- Cache access has two phases:
  - Cache indexing: use index bits to fetch the tags and data
  - Tag checking: detect hit/miss

Direct-Mapped Cache

- Example of a direct-mapped cache with two words per line

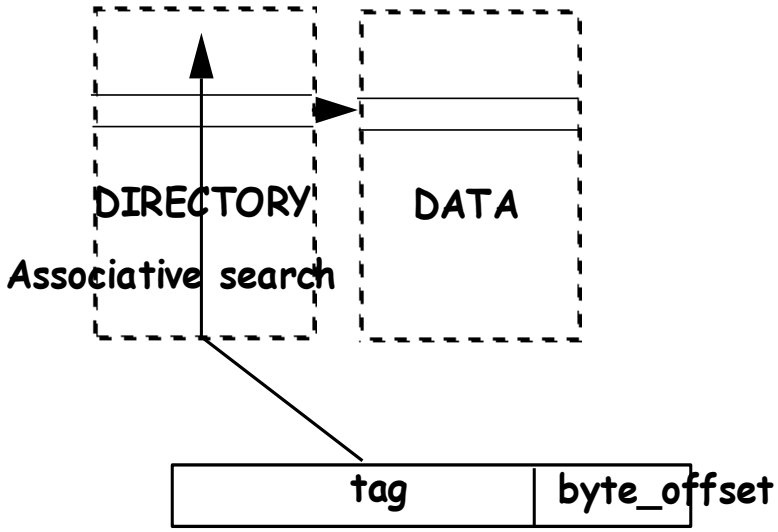


Set-Associative Cache



- In this example, each set contains three lines
  - In a read access, all three directory and data memory entries are fetched in parallel and are compared to the tag bits
  - Write access takes two cycles: one to check tags and one to write into data memory
- In a N-way (usually between 2 and 8) set-associative cache, increasing N reduces the # of hot sets

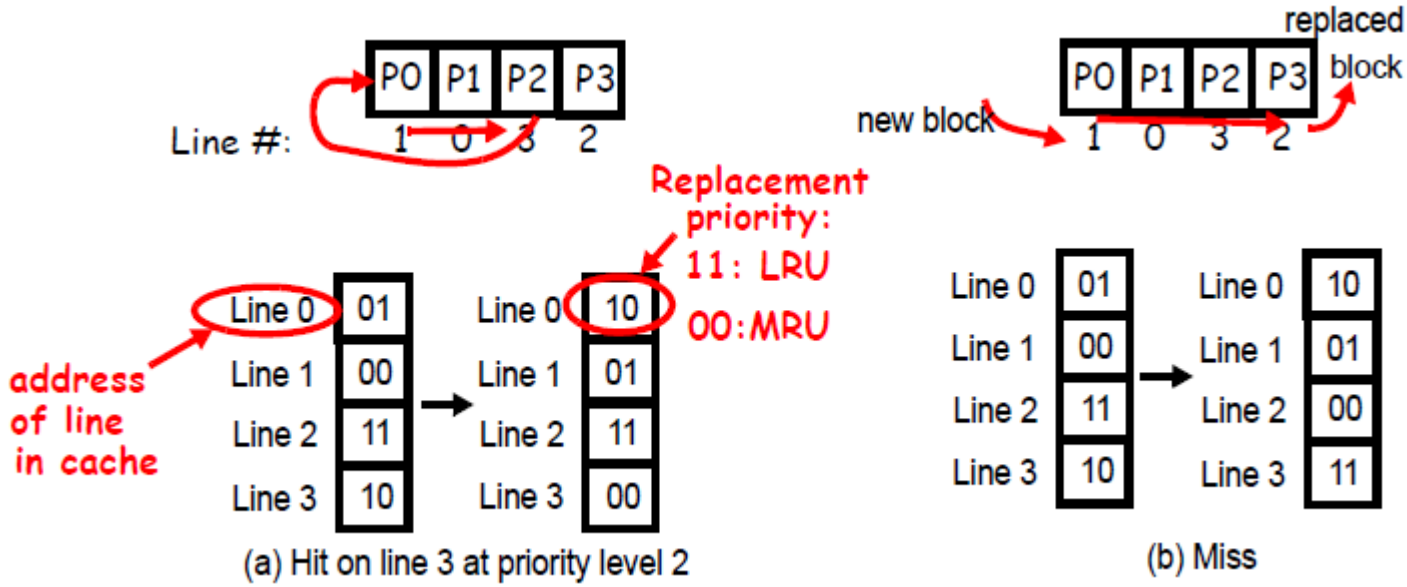
Fully-Associative Cache



- The tag is the entire block address and all entries must be checked at once in parallel, how?
  - The directory is made of content-addressable memory
    - » Comparator associated with every directory entry
- Preferable for small caches where the potential for conflicts in hot sets is damaging (e.g., TLB)

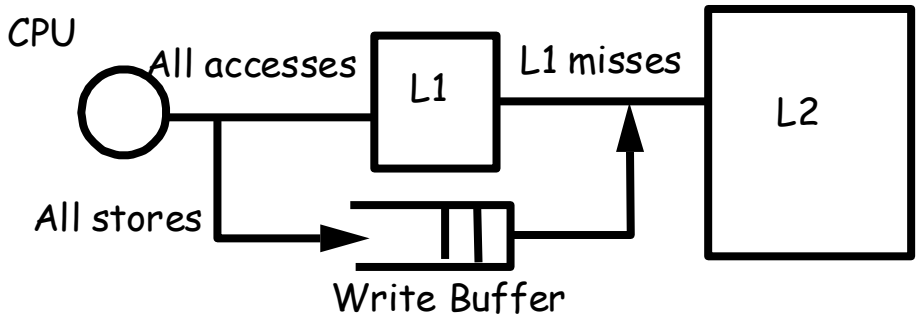
Replacement Policies

- When associativity is larger than 1, there are multiple candidate victim blocks for replacement
  - Random; Least recently used (LRU); FIFO; Pseudo-LSU



Write Policies

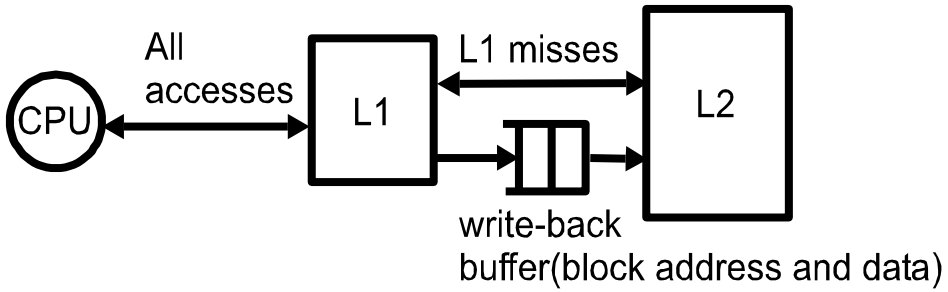
- Write-through cache: all stores update lower-level caches
  - Simple, no inconsistency among cache levels
  - Store buffer used to mitigate processor stalls



- The store traffic does not improve with cache size

Write Policies

- Write-back cache: write to next level on replacement
  - With the help of a dirty bit and a write-back buffer
  - Writes happen on a miss only



- What is the purpose of the write-back buffer?

Classification of Cache Misses

- The 3 C's
  - Compulsory (code) misses: on the 1st reference to a block
  - Capacity misses: space is not sufficient to host data/code
  - Conflict misses: happen when two memory blocks map on the same cache block

- How to find out?
  - Cold misses: simulate infinite cache size
  - Capacity misses: simulate fully-associative cache then deduct cold misses
  - Conflict misses: simulate cache then deduct the above two misses

The Impact of Cache Parameters

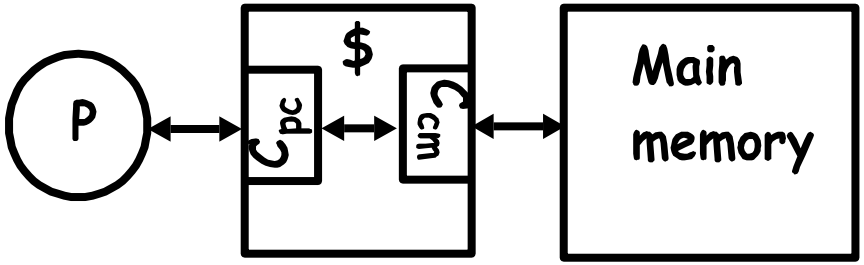
- Larger cache size
  - Slower, more complex and less capacity misses
- Larger block size
  - Exploit spatial locality
  - Too big increases miss penalty and capacity misses
- Higher associativity
  - Address conflict misses: a 2-way cache of size N has a similar miss rate as a direct-mapped cache with size 2N
  - 8-way is nearly as good as fully-associative
  - Higher hit time

Non-Blocking (Lockup-Free) Caches

- Cache is a 2-port device interfacing with memory and processor

C_{cm}: cache to memory interface

C_{pc}: processor to cache interface



- If a cache accept only one processor request at a time, it is called a blocking cache
- Lockup-free cache are capable of handling multiple hits/misses at the same time
  - Required for OoO execution and prefetching
  - Cache has to bookkeep all pending misses
    - MSHRs (Miss Status Handling Registers) contain the address of pending miss, the destination block in cache, and the destination register
    - Number of MSHRs limits the number of pending misses
  - Data dependencies eventually block the processor

Lockup-Free Caches: Primary/Secondary Miss

- Primary miss: The first miss to a block
- Secondary miss: Following accesses to blocks pending due to primary miss
  - Many more misses than blocking cache
  - Needs MSHRs for both primary and secondary misses
  - Misses are overlapped with computation and other misses

Comparison between a Blocking and Lockup-Free Cache

- Consider the following micro-benchmark:

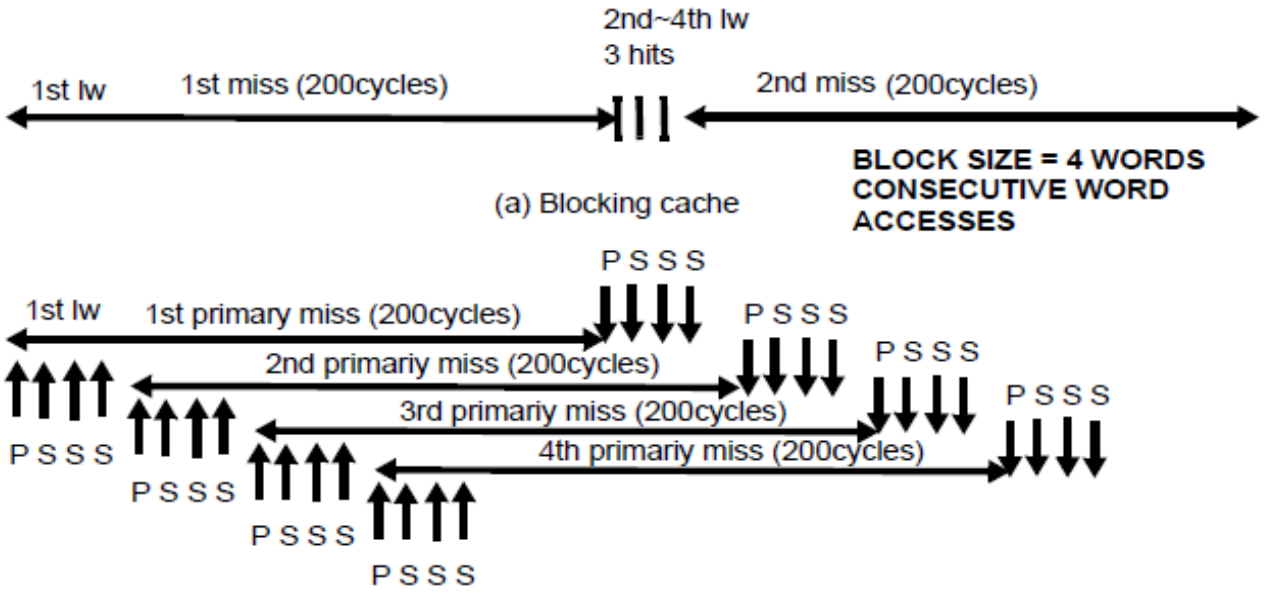
```
Toy:    LW R1, 0(R2)
        ADDI R2, R2, #4
        BNE R2, R4, Toy
```

The cache has infinite size, a block size of 16 bytes (4 words), and is initially empty. Assuming there is always a load instruction ready to issue to cache in every cycle, a hit takes 1 cycle and a miss takes 200 cycles. The lockup-free cache has 16 MSHRs. Calculate how many clock cycles are needed for the execution of each iteration for blocking cache and for lockup-free cache, respectively.

Comparison between a Blocking and Lockup-Free Cache

- Consider the following micro-benchmark:

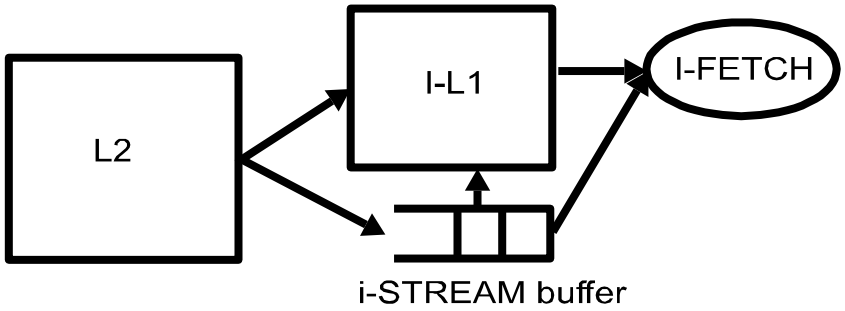
```
Toy:    LW R1, 0(R2)
        ADDI R2, R2, #4
        BNE R2, R4, Toy
```



(b) Non-blocking cache with 16 MSHRs

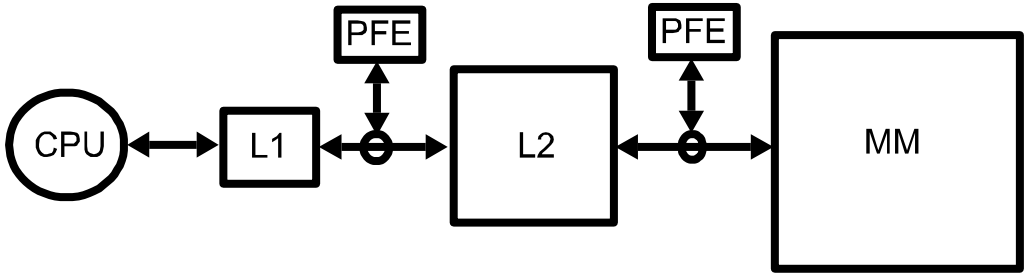
Hardware Prefetching

- Sequential prefetching of instructions



  - On an I-Fetch miss, fetch two blocks instead of one
  - Second block is stored in a I-Stream buffer (most often just one block). Why not put it in the cache?
  - If I-stream buffer hit, move it to L1 cache.
  - Less effective for data cache

- Hardware prefetch engines



Software Prefetching

- With high-level understanding of the code, compiler may insert explicit prefetch instructions in the code, e.g.,

```
LOOP    L.D F2,0(R1)
        PREF    -24(R1)
        ADD.D F4,F2,F0
        S.D F4,0(R1)
        SUBI R1,R1,#8

        BNEZ R1, LOOP
```

  - Require a lockup-free cache
  - Not free – instruction overhead
  - Compiler needs to predict the number of cycles per iteration to determine the displacement of the prefecth
    - » Underestimated -> still have miss penalty
    - » Overestimated -> cache pollution

Faster Hit Times for L1 Cache

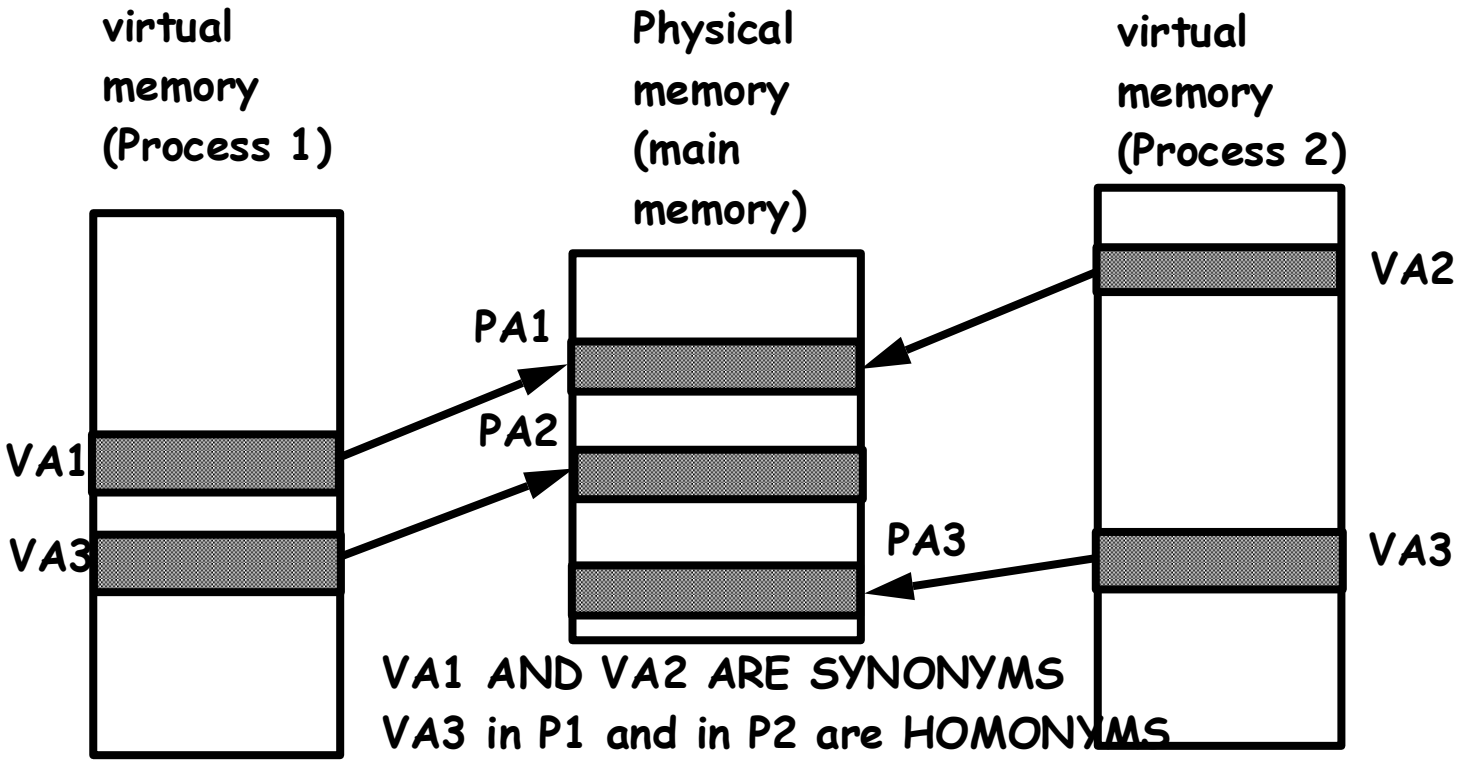
- Keep the cache simple and fast
  - This seems to favor direct-mapped cache
  - Interestingly, the size of L1 cache tends to decrease and the associativity goes up

Processor	L1 data cache
Alpha 21164	8KB, direct mapped
Alpha 21364	64KB, 2-way
MPC750	32KB, 8-way, PLRU
PA-8500	1MB, 4-way, PLRU
Classic Pentium	16K, 4-way, LRU
Pentium-II	16KB, 4 -way, PLRU
Pentium-III	16K, 4-way, PLRU
Pentium-IV	8KB, 4-way, PLRU
Mips R10K/12K	32KB, 2-way, LRU
UltraSPARC-II*i*	16KB, direct mapped
UltraSPARC-III	64KB, 4-way, Random

Motivations for Virtual Memory

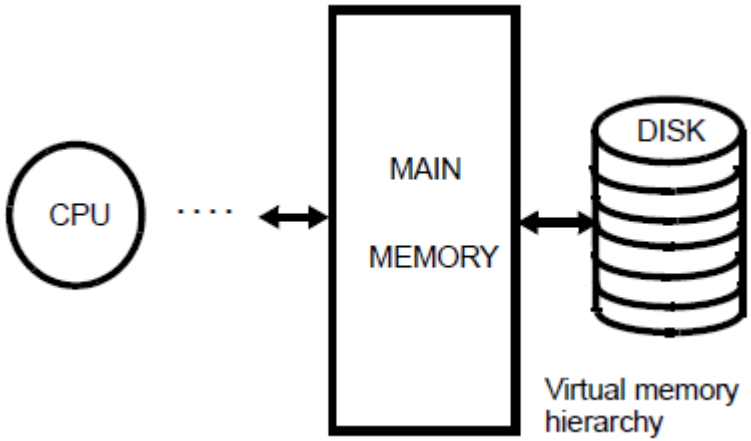
- Allows applications to be bigger than main memory size
- Facilitate process management
  - Each process gets its own chunk of memory
  - Mapping of multiple processes to memory
  - Protection of processes against each other
  - Relocation and sharing

Virtual Space Mapping



Main memory acts as a fully-associative cache for the virtual memory space!

Paged Virtual Memory



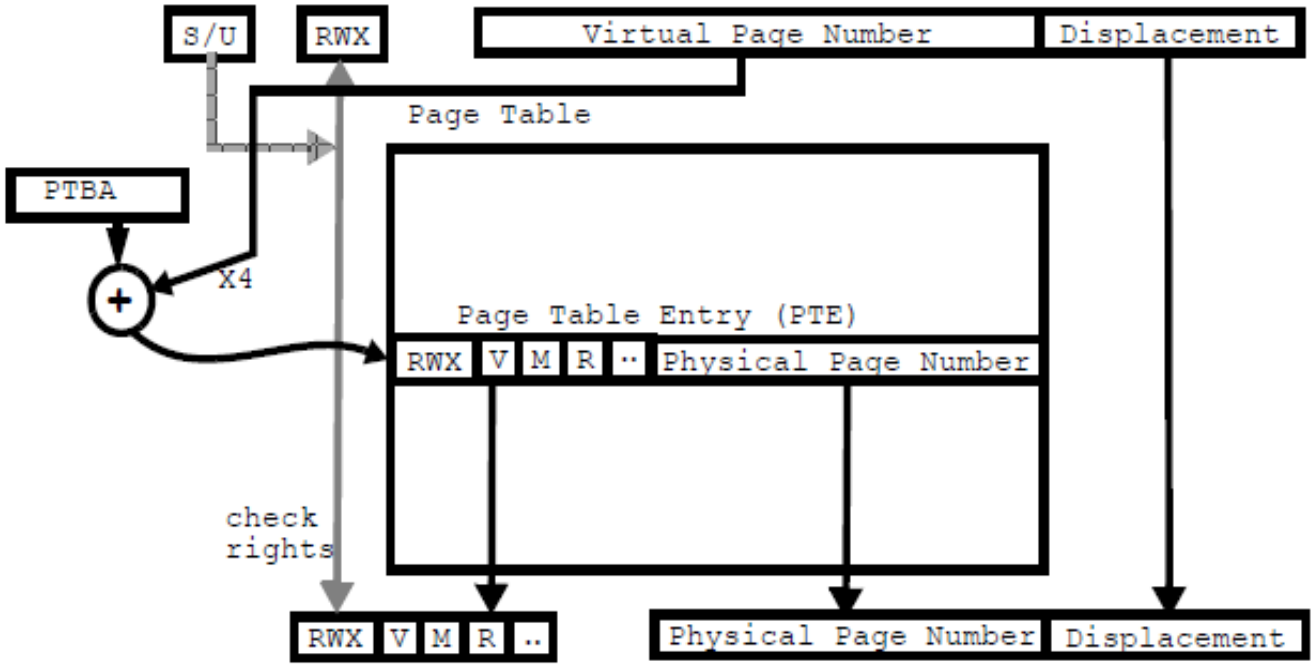
- Virtual address space divided into pages, while physical address space divided into page frames of the same size as pages
  - Analogous to the memory blocks and cache lines in the context of caches
  - Miss in MM triggers a page fault
  - Pages not in MM are on disk: swap-in/swap-out with software handler
- Dynamic address translation
  - Effective address is virtual
  - Must be translated to physical for every access
  - Virtual to physical translation through page table in MM

Page Replacement Policy

- LRU?
  - Impossible with the sheer size of physical memory
- A common algorithm is working set (approximate LRU)
  - Keep tracks of page frames that have been referenced in a recent time window
    - » Reference bit (R) per page periodically reset by OS
  - Pages not in this window marked as not valid and inserted in a page cache
  - Hard vs. Soft page faults
- Write strategy is write-back using modify (M) bit

Page Table

- Page table translates address, enforces protection

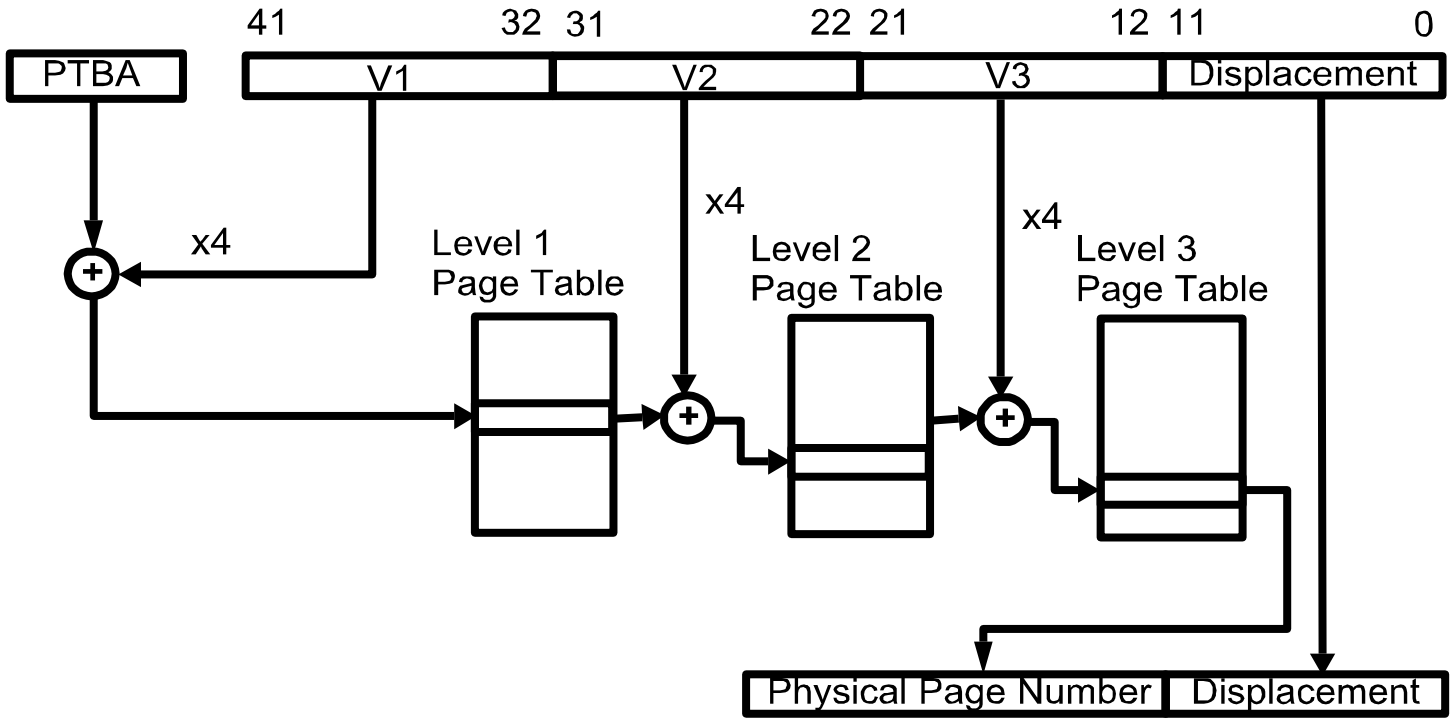


- The RWX bits are used for checking access rights
- Some state bits for virtual memory management

Page-Table Fragmentation

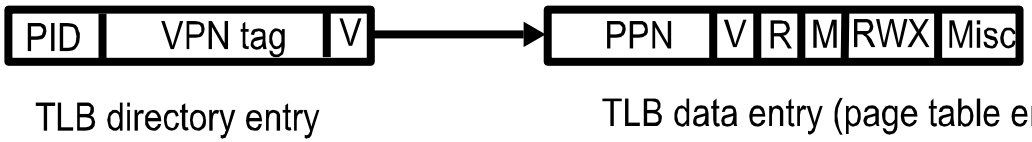
- Page-table fragmentation problem
  - All page-table entries must be allocated in main memory even if a large of them is invalid
  - Consider a 4Gbyte virtual address space with each page size of 4Kbytes, if the size of each entry is 4 bytes, how large is the page table for each process?
    - » 2^{20} pages, require 4Mbytes of memory!
- How to resolve this problem?
  - Page tables can be accessed hierarchically

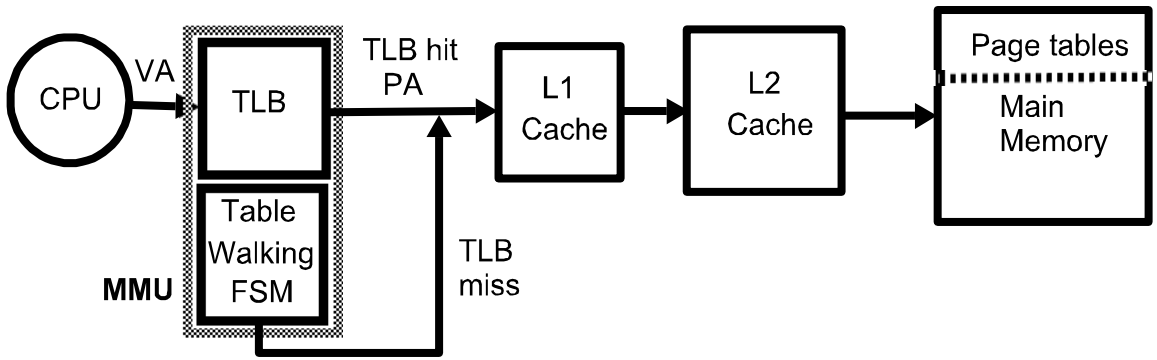
Hierarchical Page Table



- What is the range of total amount of memory required for the above page table?
  - Only a few tables are required at levels 2 and 3, so it is in the tens of Kbytes.
- Hierarchical page table supports superpages, how?

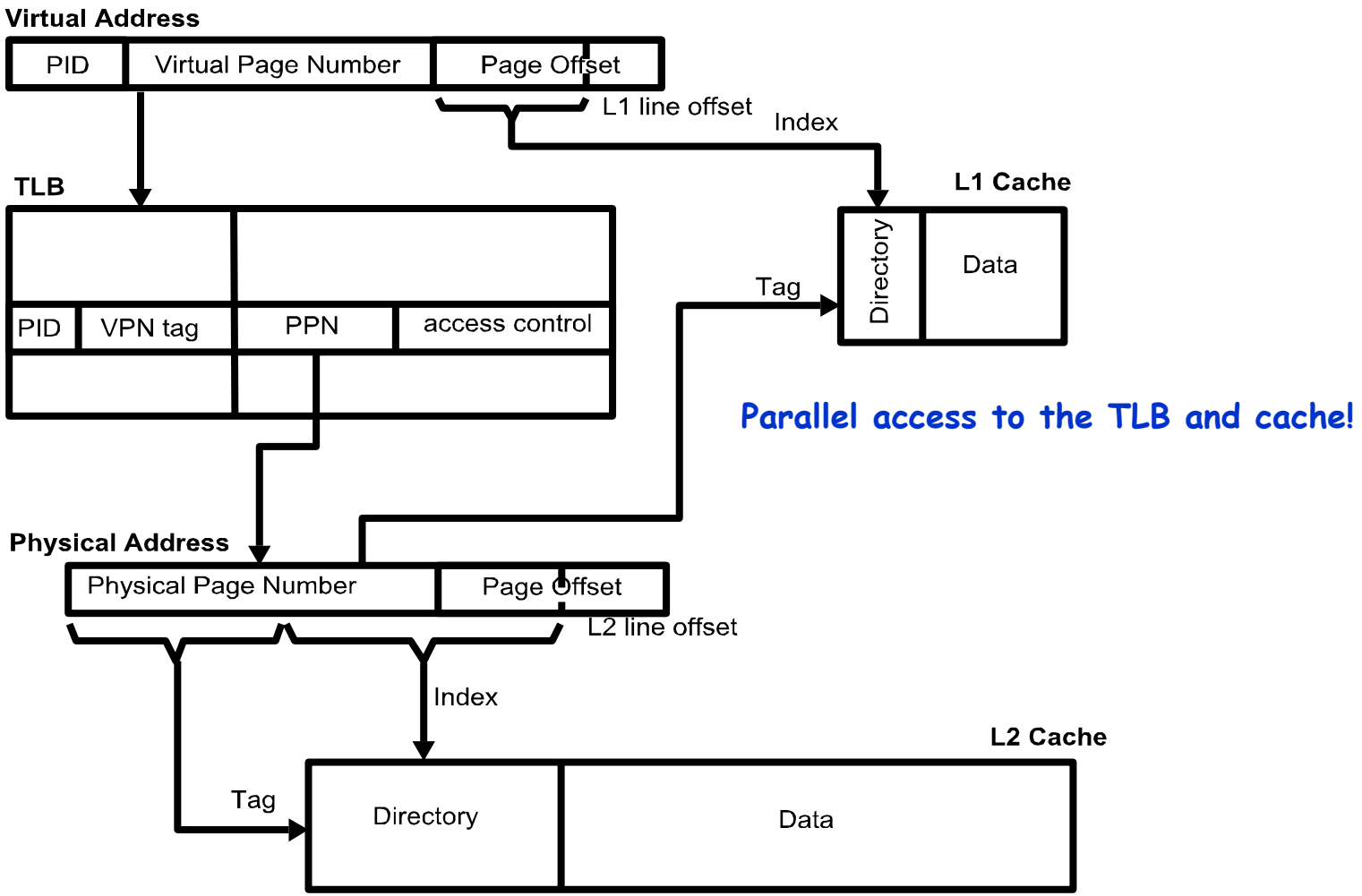
Translation Lookaside Buffer





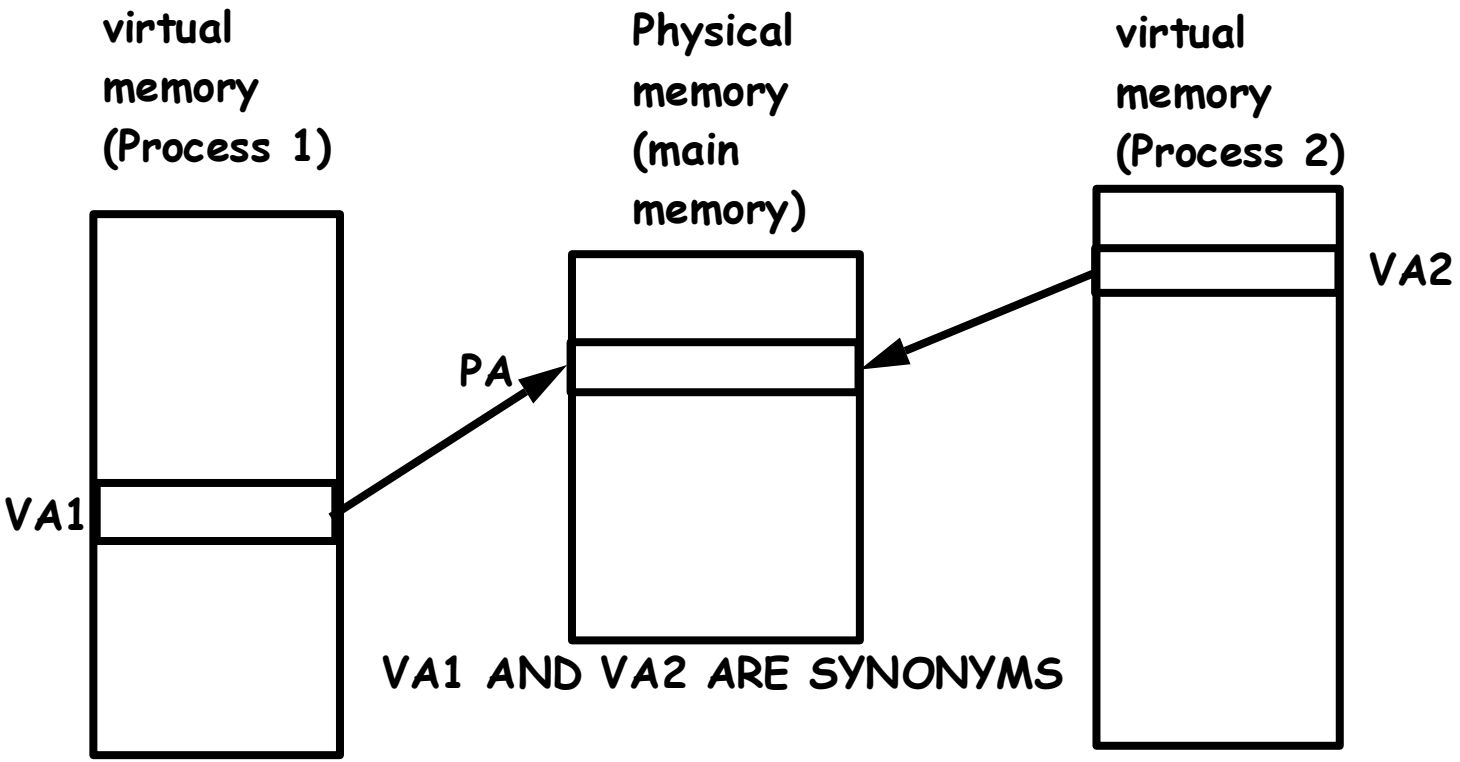
- With hierarchical page tables, each memory reference takes a number of memory/cache accesses to complete
- Translation lookaside buffer (TLB) serves as a cache for page-table entries accessed with virtual page number
  - PID added to deal with homonyms
  - TLB miss can be handled in a hardware MMU or by a software trap handler
  - TLBs are usually much smaller than regular caches, why?

Typical Memory Hierarchy

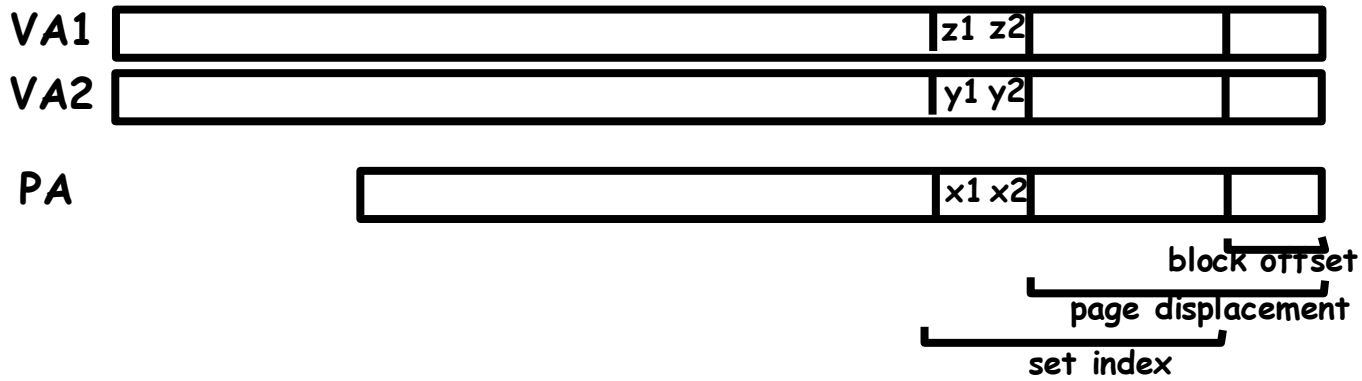


Indexing Cache with Virtual Address Bits

- When the L1 cache size is larger than 1 page per way of associativity, we run into synonym (aliasing) problem and lead to coherence issue



Aliasing Problem



- If the cache is indexed with the 2 extra bits of VA, then, a block may end up in different sets if accessed with two synonyms, causing consistency problems. How to avoid this?
  - On a miss, search all the possible sets (in this case 4 sets), move block if needed (short miss), and settle on a (LONG) miss only if all 4 sets miss, OR
  - Since L2 cache is physically addressed and inclusion is enforced, make sure that L2 cache entry contains a pointer to the latest block copy in cache, OR
  - Page coloring: make sure that all aliases have the same two extra bits(z1=y1=x1 and z2=y2=x2)

Virtual Machines

- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable

- Allows different ISAs and operating systems to be presented to user programs
  - "System Virtual Machines"
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs"

Impact of VMs on Virtual Memory

- Each guest OS maintains its own set of page tables
  - VMM adds a level of memory between physical and virtual memory called "real memory"
  - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
    - » Requires VMM to detect guest's changes to its own page table
    - » Occurs naturally if accessing the page table pointer is a privileged operation

Reminders

- Next lecture is about multithreading and reliability
- Homework 3 will be online this week