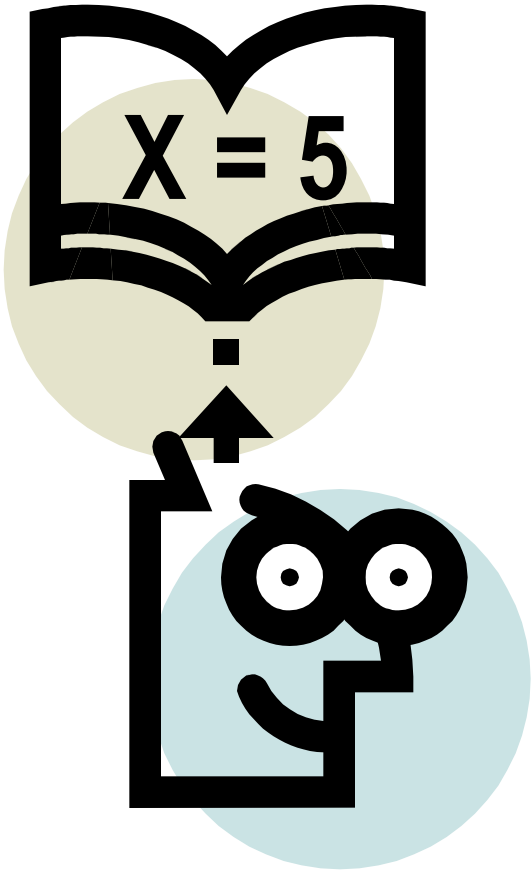
Lecture 15

Advanced Caches + Intro to VM



Adapted from slides originally developed by Profs. Hardavellas, Hill, Falsafi, Marculescu, Patterson, Rutenbar and Vijaykumar of Northwestern, Carnegie Mellon, Purdue, UC-Berkley, UWisconsin

Today's Menu:

- **Wrap-up and summary of caches**
- **Virtual Memory**
  - Virtual vs. Physical Address Spaces
  - Page Table
  - Address Translation

Q1: Block Placement

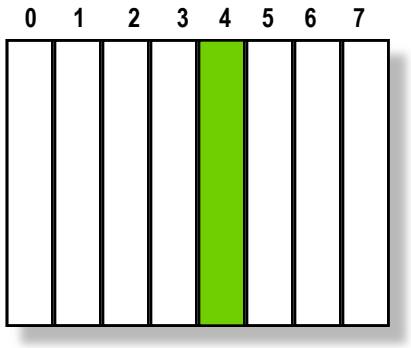
Direct Mapped
block 12 can go only into
block 4 (12 mod 8)

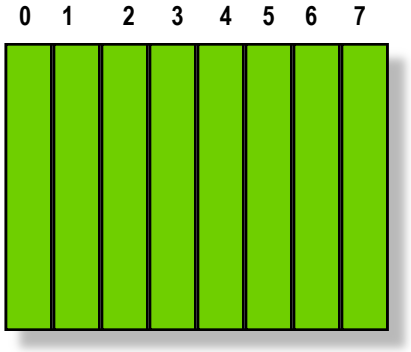
Fully-associative
block 12 can go anywhere

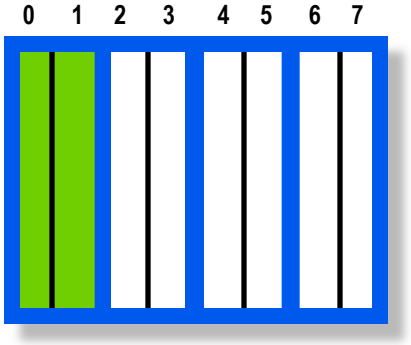
2-way Set-associative
block 12 can go anywhere in set 0
(12 mod 4)

Block Number

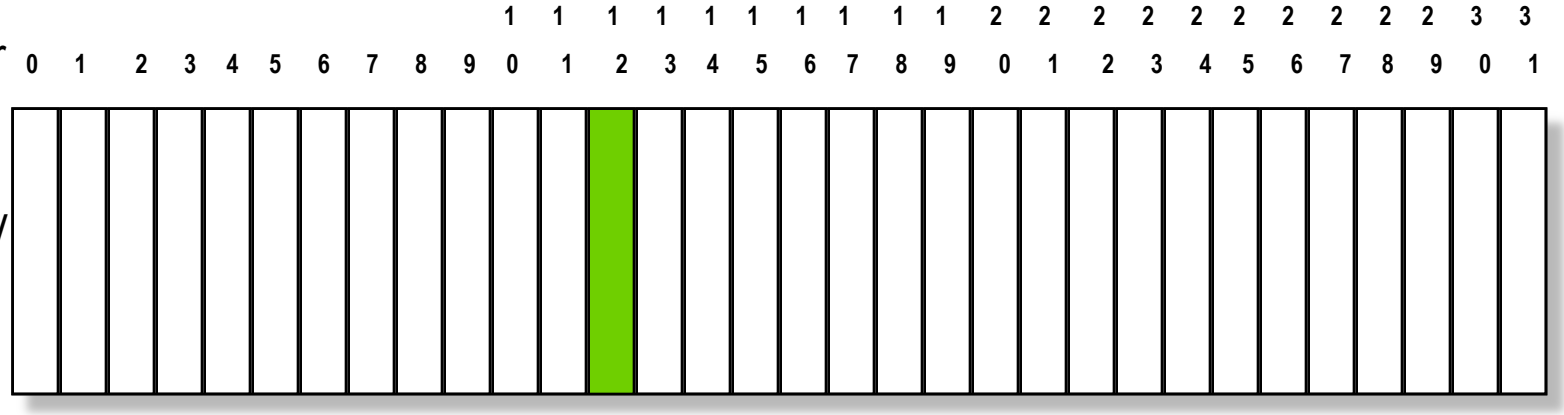
Cache







Block Number

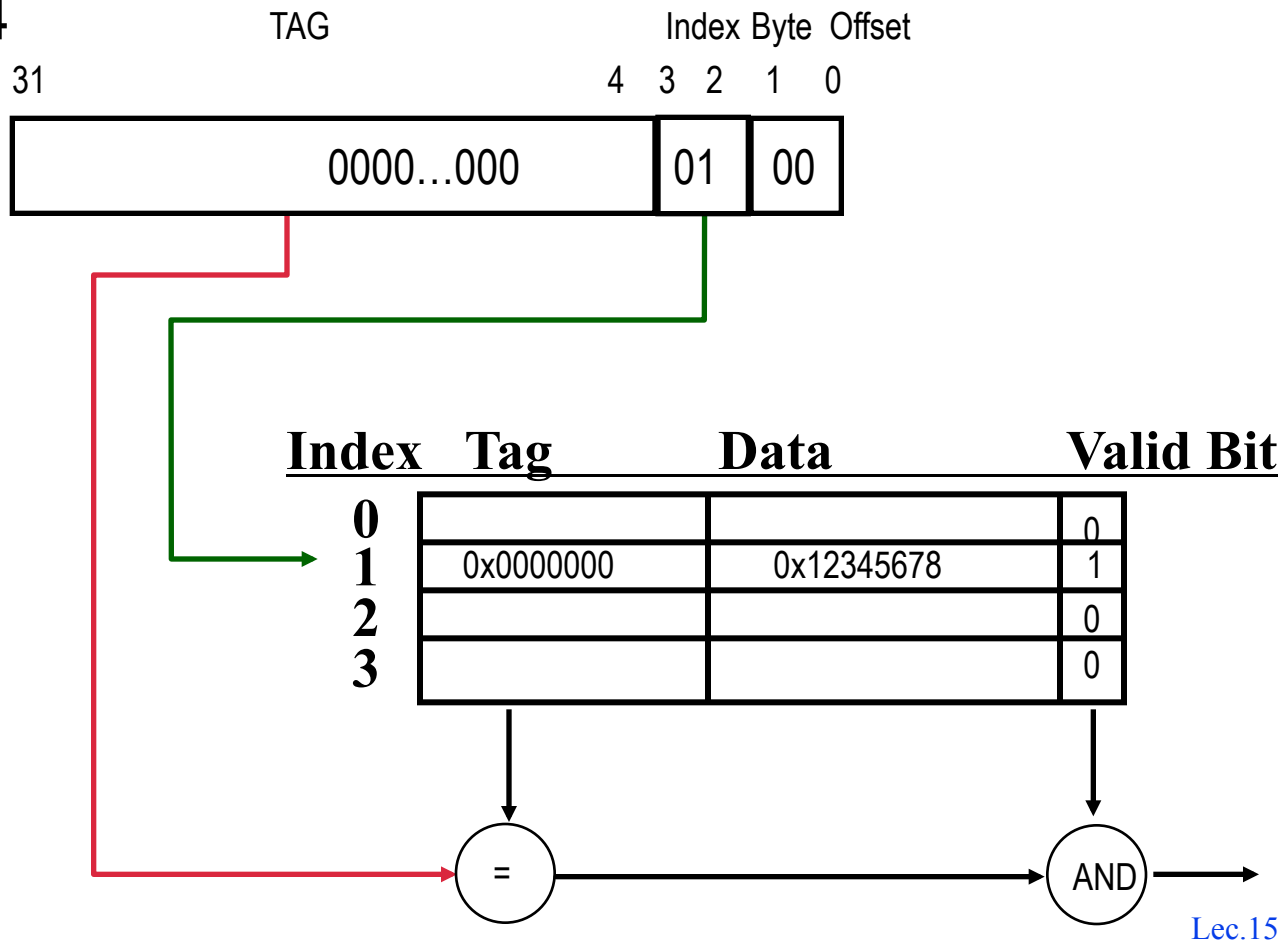


Memory

Q2: Block Lookup

- Assume cache has 4 blocks and each block is 1 word (4 bytes)

LOAD R1, 0x04

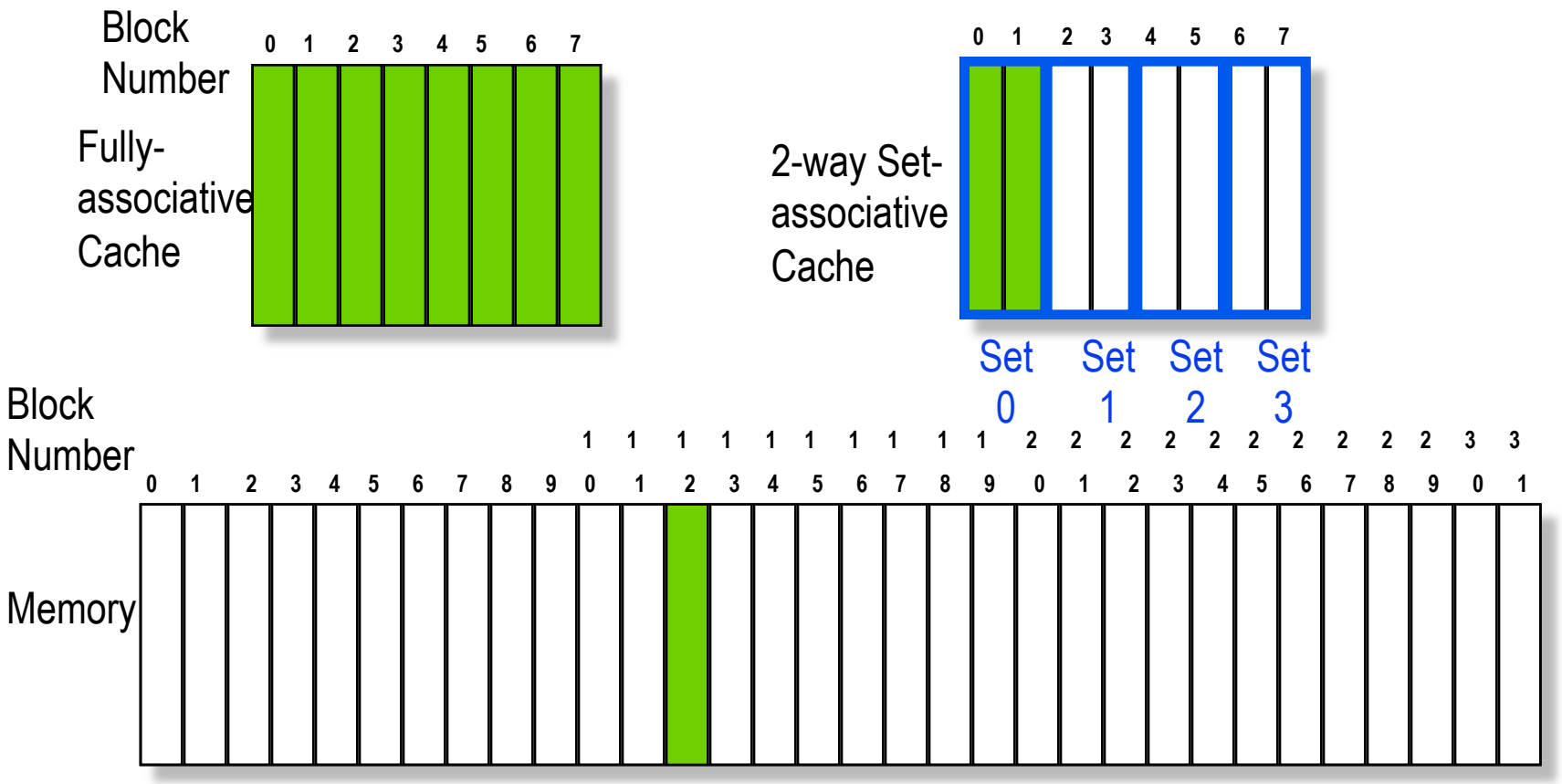


MEMORY

Address	Data
0x00	0x00000000
0x04	0x12345678
0x08	0x87654321
0x0C	0x11111111
0x10	0x22222222
0x14	0x33333333
0x18	0x44444444
0x1C	0x55555555
0x20	0x10101010

Q3: Block Replacement

- In a direct-mapped cache, there is only one block that can be replaced
- In set-associative and fully-associative caches, there are N blocks (where N is the degree of associativity



Q4: Write Strategy

- **When data is written into the cache (on a store), is the data also written to main memory?**
  - If data is written to memory, the cache is called a ***write-through cache***
    - PRO: Easy hardware
    - PRO: good speed on read misses
  - If data is NOT written to memory, the cache is called a ***write-back cache***
    - For a write-back cache, need to remember which blocks have been written ("dirty bit")
    - When you replace a "dirty" block, it must first be written back into the main memory
    - PRO: Good for "write-intensive" apps

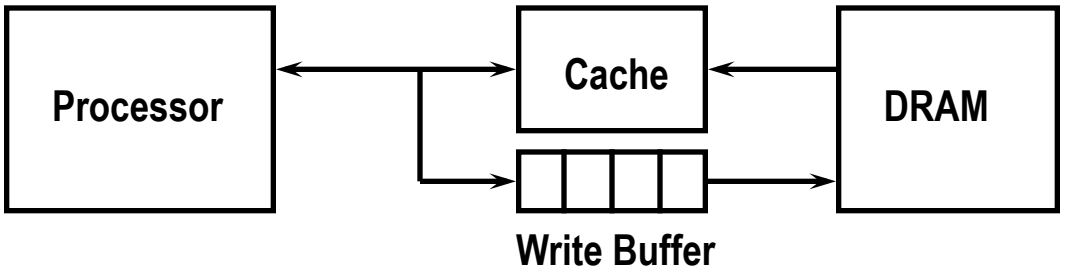
Q4: Write Strategy (Write Allocation)

- **Should you cache a block if it is a write-miss?**
  - Maybe it will not be read again
  - Maybe it will replace a block that could be read again
- **Write allocate:**
  - On ANY miss (write or read) move the missed block into the cache
- **Write no-allocate:**
  - Only move missed block into the cache in case of read-miss

Summary of Cache Questions

- **Q1: Where can a block be placed in the cache?**
  - Direct mapped, associative, fully-associative [Block Placement]
- **Q2: How is a block found if it is in the cache?**
  - indexing (direct mapped), limited search (associative), full search (fully-associative) [Block Lookup]
- **Q3: Which block should be replaced on a cache miss?**
  - random, least-recently used (LRU) [Block Replacement]
- **Q4: What happens on a write?**
  - write-through or write-back [Write Strategy (Write Allocation)]

Write Buffer for Write Through



- **A Write Buffer is needed between the Cache and Memory**
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- **Write buffer is just a FIFO:**
  - Typical number of entries: 8
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle
- **Memory system designer's nightmare:**
  - Store frequency (w.r.t. time) > 1 / DRAM write cycle
  - Write buffer saturation

Metric for Cache Performance - Miss Rate (SPEC 92)

- **32 Byte block size, direct-mapped caches**
  - **Option 1**: 2 separate caches, one for instructions, one data; **a Split cache**
  - **Option 2:** a single cache with both instructions & data; **a Unified cache**
  - Each holds the same # of total bytes

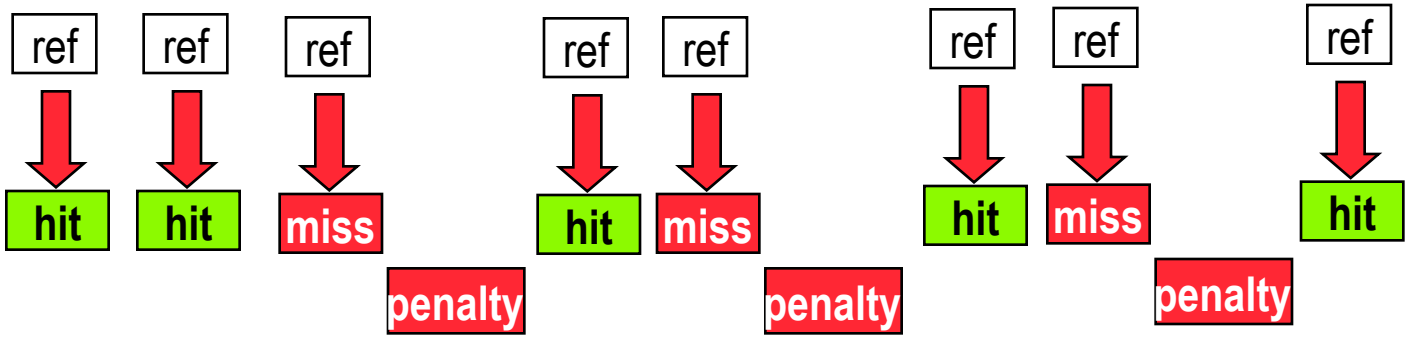
Cache Size	Instruction Cache	Data Cache	Unified Cache
1KB	3.06%	24.61%	13.34%
2KB	2.26%	20.57%	9.78%
4KB	1.78%	15.94%	7.24%
8KB	1.10%	10.19%	4.57%
16KB	0.64%	6.47%	2.87%
32KB	0.39%	4.82%	1.99%
64KB	0.15%	3.77%	1.35%
128KB	0.02%	2.88%	0.95%

- **Miss rate neglects cycle time implications**

Better Metric: Average Memory Access time

- **Average time to access memory (AMAT)**

Average memory access time **=** Hit time **+ (**Miss Rate **X** Miss Penalty**)**
where miss penalty **is the extra time it takes to handle a miss**
(i.e., the time beyond the required 1 cycle hit cost)

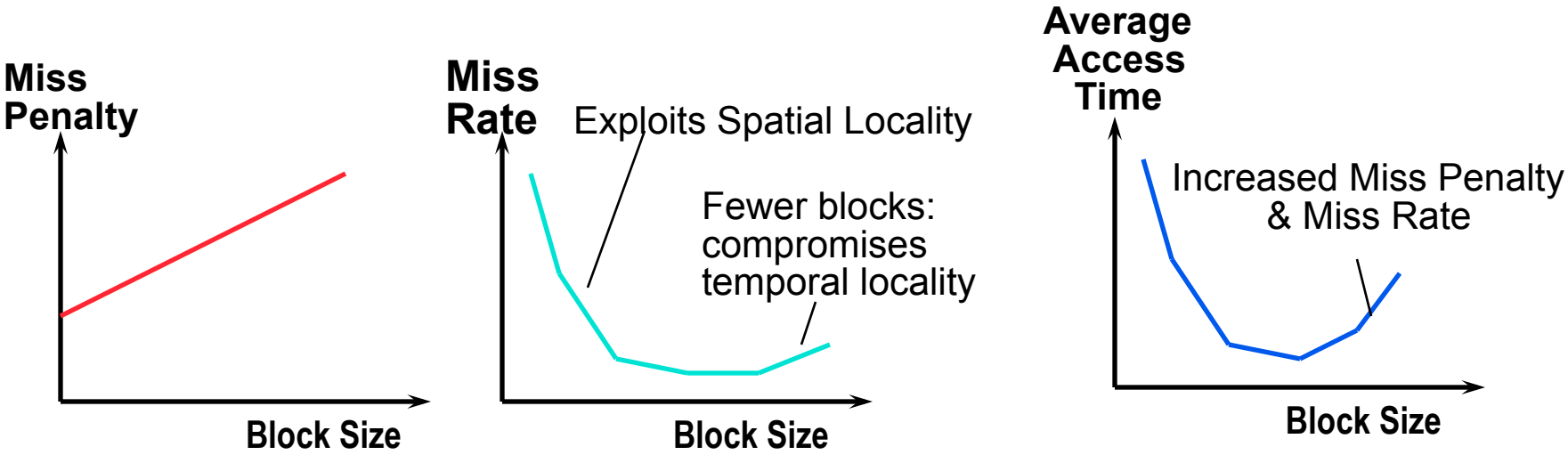


Example: **8** references, **5** hits, **3** misses
Average Mem Access Time =[(**5** X hit time) + (**3** X miss time) + (**3** X miss penalty)] / **8**
=[(**8** x hit time since miss=hit=1 cycle) + (**3** X miss penalty)] / **8**
= (hit time) + (**3 / 8**)(miss penalty)
= (hit time) + (**miss rate**)(miss penalty)

hit = **miss**
penalty >= **hit**

Block Size Tradeoff

- **In general, larger block size takes advantage of spatial locality BUT:**
  - Larger block size means larger miss penalty:
    - Takes longer time to fill up the block
  - If block size is too big relative to cache size, miss rate will go up
- **Average Access Time:**
  - = Hit Time + Miss Penalty x Miss Rate



Ruminations

- **Recall performance metric:**

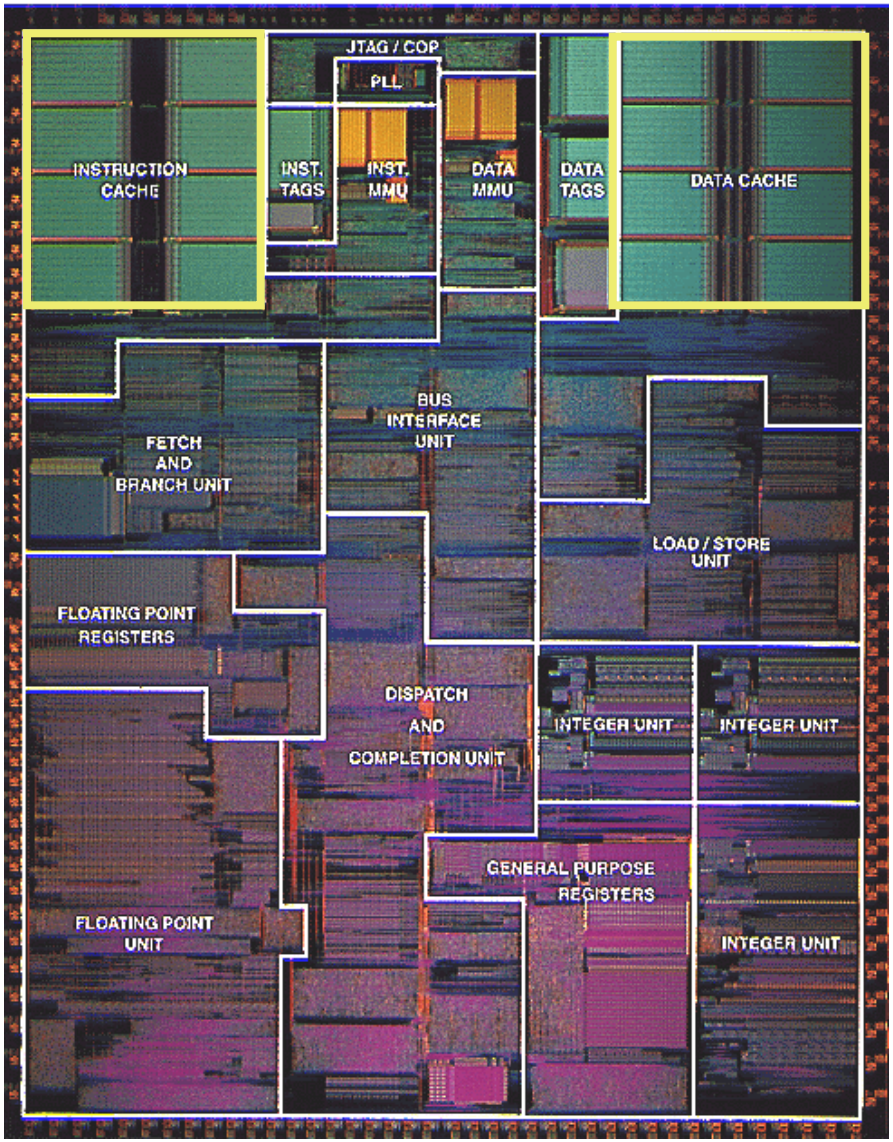
  CPU time **=** IC **X (** $CPI_{execution}$ **+** Memory stall cycles / Instruction**) X** Clock cycle time

  Where: IC **=** Instruction Count

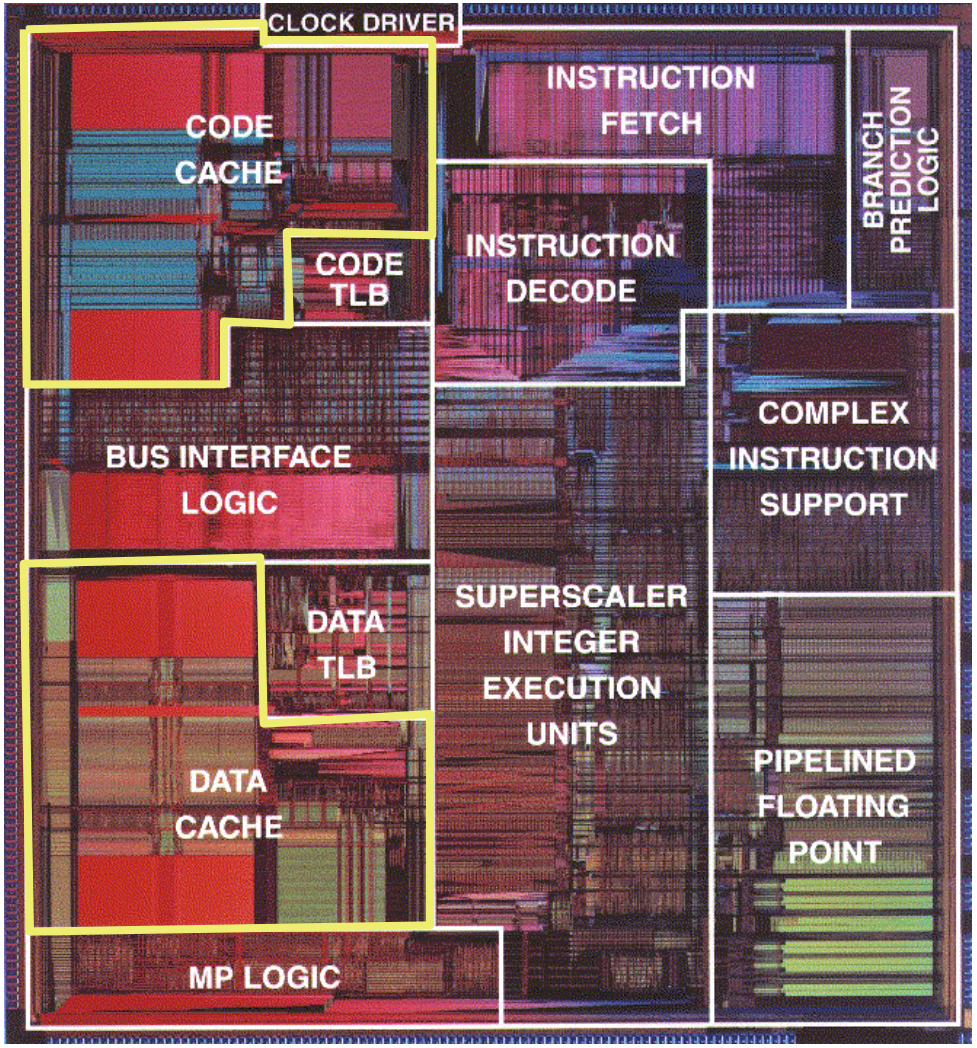
- **Two important results to keep in mind:**
  - The **lower** the $CPI_{execution}$, the **higher** the **relative impact** of a cache miss penalty
  - Comparing two machines with identical memory systems, the machine with the higher clock rate will have the larger number of clock cycles per miss and hence the memory portion of its CPI will be higher.

Real Life Examples: PowerPC

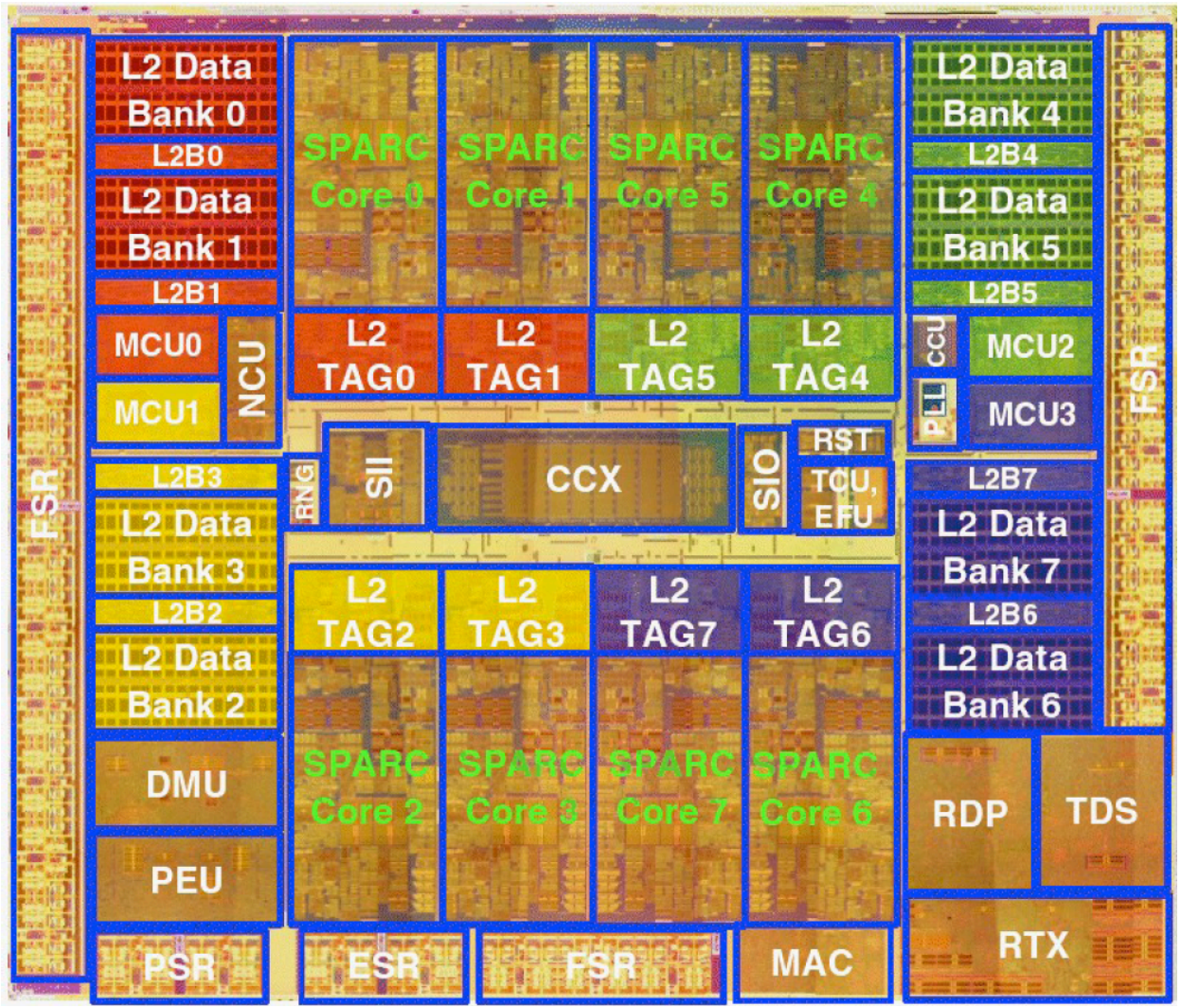
Motorola's PowerPC™ 604 RISC Microprocessor



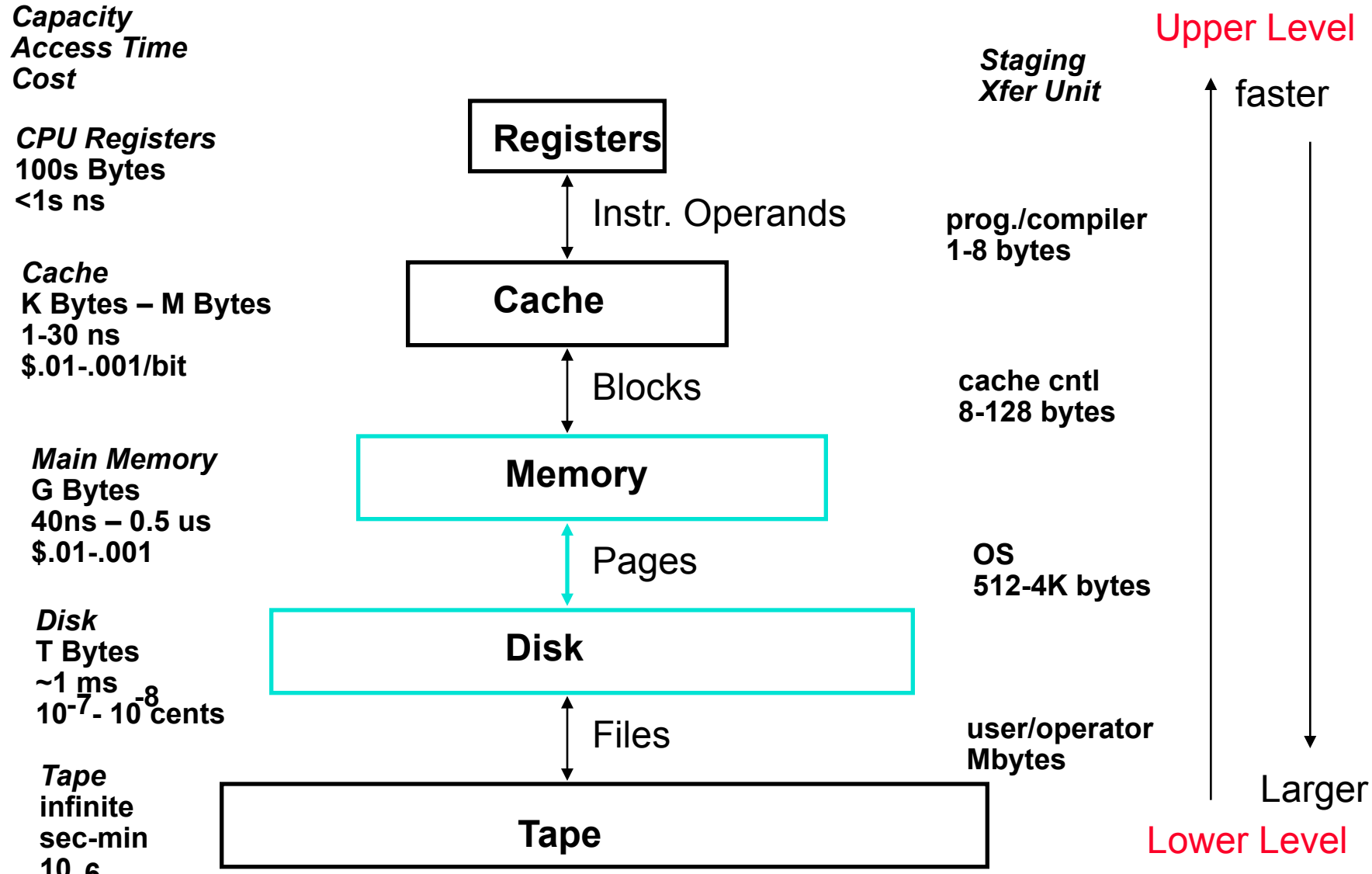
Real Life Examples: Pentium Pro



Real Life Examples: Sun UltraSPARC T2 (Niagara-2)



Summary: Levels of the Memory Hierarchy



Improving Cache Performance

- **Google Scholar found 23,900 entries for "cache performance"**
  - Guess: Probably ~5,000 papers in the last 30 years about cache performance
- **Boiled it all down to three basic types of optimizations**
  - O1: Reducing the cache miss rate
  - O2: Reducing the cache miss penalty
  - O3: Reducing the time to hit in the cache

    (i.e., cache access time)

Cache Performance Design Parameters

- **Organization of a cache can significantly influence performance**
- **Cache design issues**
  - Size — # of bytes in cache
  - Block size — # of bytes per block
  - Associativity — # of lines per index
  - Bandwidth — # of bytes returned per cycle
  - Write through vs. write back
  - Cache partitioning: split vs. unified instruction & data caches
  - Access time
  - Power

O1: Reducing the Cache Miss Rate

- **What are realistic cache miss rates found in systems today?**
  - Very dependent on the application
  - Data for SPEC shows
    - I-cache (instruction cache) miss rates under 5% and often under 1%
    - D-cache (data cache) miss rates as high as 25%
    - From this, we could conclude that the major problem is the D-cache
      - But SPEC is not representative of many applications
  - Data from commercial workloads show
    - Database I-cache miss rates as high as 20% for (16KB, direct mapped)
    - Dick Sites showed that Microsoft's SQL server had an instruction working set of over 500,000 bytes
      - Alpha's 21164 I-cache was 8,192 Bytes
      - Alpha's 21164 L2 I-cache was 96K Bytes

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
- Conflict **(collision):**
  - Multiple memory locations mapped to the same cache location (e.g. in set associative caches)
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- Capacity**:**
  - Cache cannot contain all blocks access by the program
  - Solution: increase cache size
- Invalidation**: other process (e.g., I/O) updates memory**

Compulsory, Conflict and Capacity Miss Rates

Cache Size	Assoc.	Total Miss Rate	Compuls.	Capacity	Conflict
1KB	**1-way**	**0.133**	**0.002**	**0.080**	**0.052**
	2-way	**0.105**	**0.002**	**0.080**	**0.023**
	4-way	**0.095**	**0.002**	**0.080**	**0.013**
	8-way	**0.087**	**0.002**	**0.080**	**0.005**
16KB	**1-way**	**0.029**	**0.002**	**0.015**	**0.012**
	2-way	**0.022**	**0.002**	**0.015**	**0.005**
	4-way	**0.020**	**0.002**	**0.015**	**0.003**
	8-way	**0.018**	**0.002**	**0.015**	**0.001**
128KB	**1-way**	**0.010**	**0.002**	**0.004**	**0.004**
	2-way	**0.007**	**0.002**	**0.004**	**0.001**
	4-way	**0.007**	**0.002**	**0.004**	**0.001**
	8-way	**0.006**	**0.002**	**0.004**	**0.001**

Techniques for Reducing Miss Rates

- **Capacity misses are reduced by increasing the** size **of the cache**
- **Increased associativity** decreases **the number of conflict misses**
- **Compulsory misses** cannot **be reduced**
  - Except by prefetching (which can be accomplished with larger block sizes)

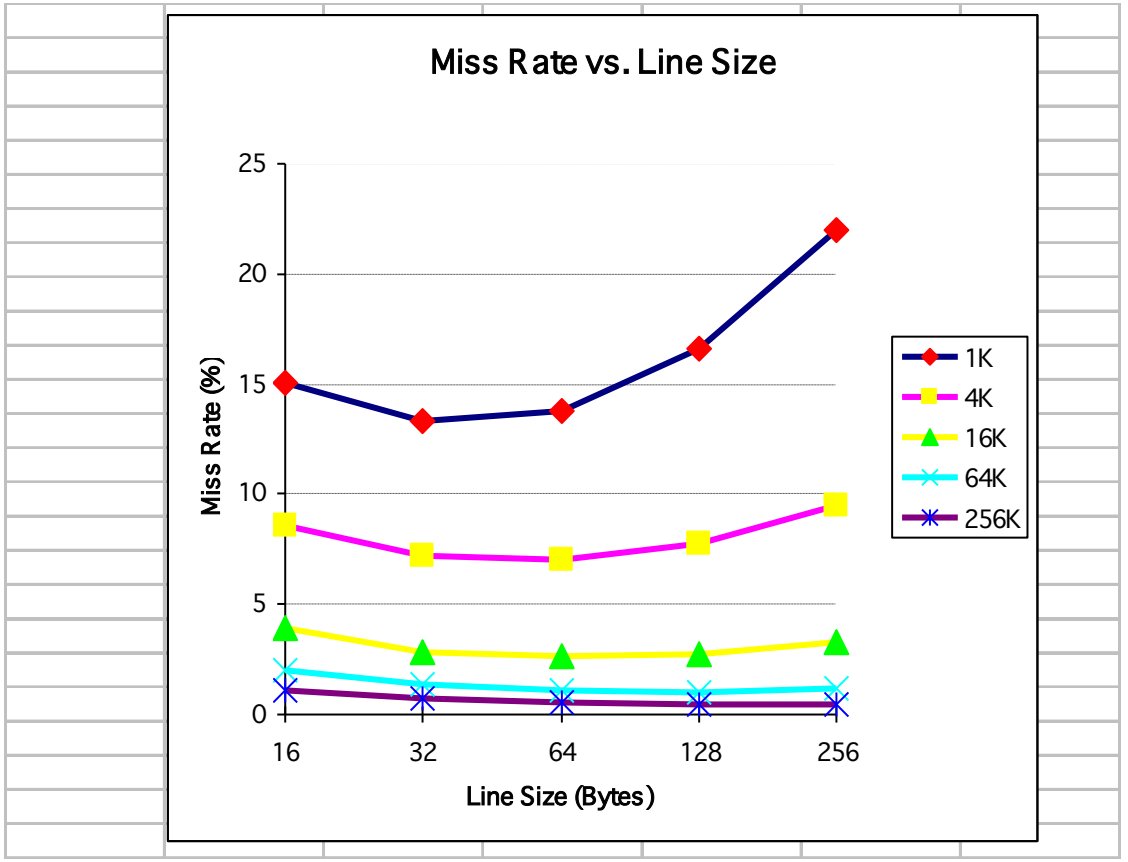
- **We'll look at several techniques**
  - T1. Larger blocks
  - T2. Higher associativity
  - T3. Victim caches
  - T4. Hardware prefetching
  - T5. Compiler prefetching

O1:T1 - Larger Blocks

- **Larger blocks reduce the number of compulsory misses because** more **data is brought into the cache on** each **miss**
  - Programs usually have a high degree of *spatial* locality
- **Unfortunately, larger blocks eventually create *cache pollution***
  - Larger blocks mean that more data is evicted from the cache
- **Unfortunately, larger block sizes** increase **the miss penalty**
  - Larger blocks take **more** cycles to refill
  - At some point, larger blocks **no longer** improve performance

Miss Rate vs. Block Size

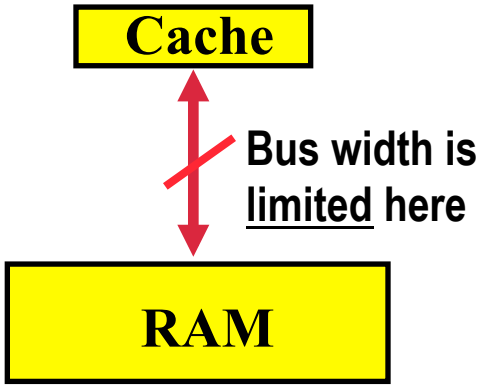
- For smaller size caches, increasing the block size beyond 64 bytes causes **cache-pollution** and increases the cache miss rate



Average Memory Access Time

		Cache Size				
Block Size	Miss Penalty	1K	4K	16K	64K	256K
16	42	7.321	4.599	2.65	1.857	1.458
32	44	*6.870*	*4.186*	*2.263*	1.594	1.308
64	48	7.605	4.360	2.267	*1.509*	*1.245*
128	56	10.318	5.357	2.551	1.571	1.274
256	72	16.847	7.847	3.369	1.828	1.353

- **How can Average Memory Access time increase with longer line size?**
  - Time to refill a line is dependent on the size of the line
  - Example
    - Datapath from memory to cache is 4 bytes (32 bits)
    - Cache line is 4 bytes, Refill takes one "bus cycle"
    - Cache line is 8 bytes, Refill takes two "bus cycles"
    - Cache line is 16 bytes, Refill takes four "bus cycles"



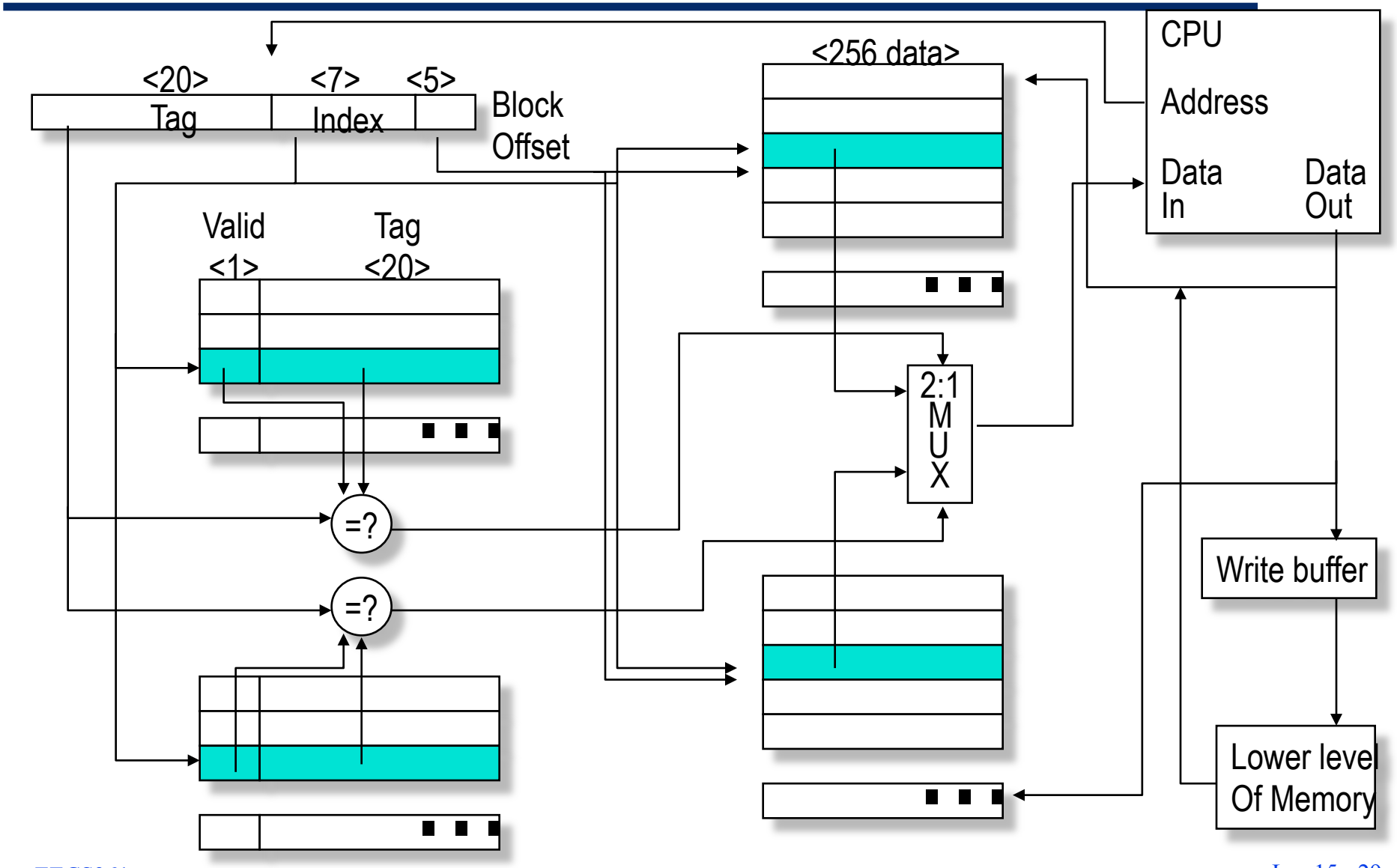
O1:T2 - Higher Associativity

- Higher **degrees of associativity** reduce **conflict misses**
- **Often 8-way set-associative caches are** almost as good **as fully associative caches**
- Increasing **the associativity can** increase **the access time**
  - more logic for compares
  - often, set-associative caches are not used as primary caches where cycle-time is most important
  - instead, they are used in lower-level caches where access time is less of an issue

Average Memory Access Time

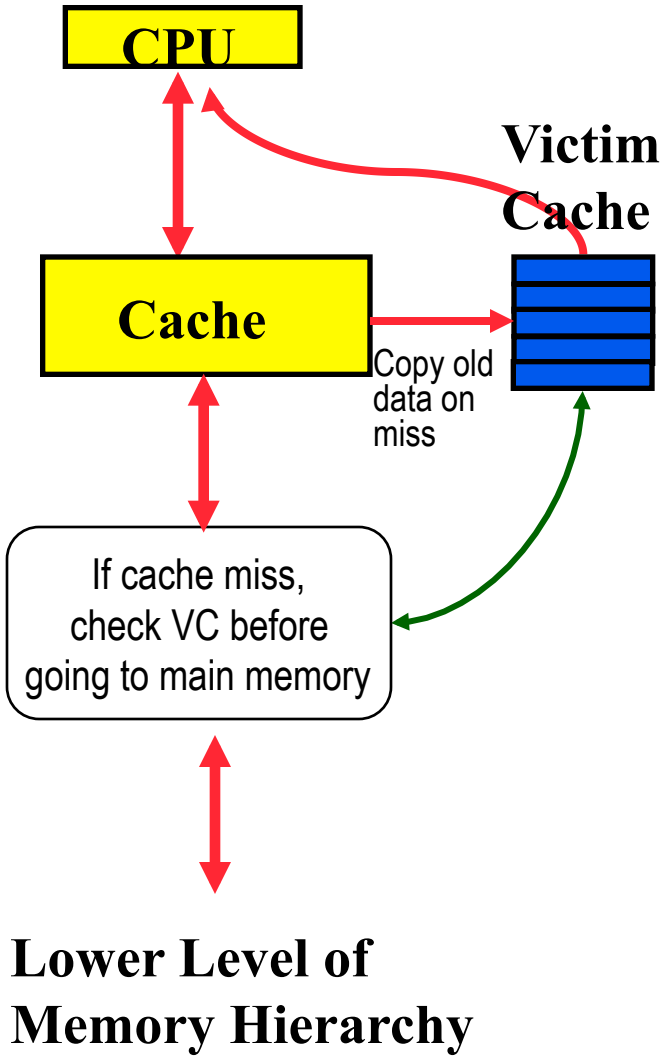
	Associativity			
Cache Size (KB)	1-way	2-way	4-way	8-way
1	7.65	6.60	6.22	5.44
2	5.90	4.90	4.62	4.09
4	4.60	3.95	3.57	3.19
8	3.30	3.00	2.87	2.59
18	2.45	2.20	2.12	2.04
32	2.00	1.80	1.77	1.79
64	1.70	1.60	1.57	1.59
128	1.50	1.45	1.42	1.44

Example 8KByte 2-way Set-associative Cache



O1:T3 - Victim Caches

- **Invented by Norm Jouppi (DEC WRL in Palo Alto)**
- **Small cache that contains the** *most recently discarded* **cache blocks**
  - Often 1-5 blocks
  - On cache miss, victim cache is checked
  - If block is present, victim cache block is placed back into the primary cache
- **How effective?**
  - Very effective for direct-mapped caches
    - A four-entry victim cache can remove 20% to 95% of the conflict misses in a 4-KByte direct-mapped cache



O1:T4 - Hardware Prefetching

- **Prefetching is a tough problem in general**
- **Must predict:**
  - **What** addresses the processor subsequently accesses?
  - **When** the processor accesses these addresses
- **Generalized prefetchers to handle all memory access patterns**
- **Simple stride predictors work well in some cases:**
  - Array element accesses (consecutively marching down array elements)
  - Consecutive instruction fetches

Hardware Stride-of-1 Predictor

- **On a cache miss, fetch** both **missing block & subsequent blocks**
  - Prefetched blocks are not placed directly into the cache, but into a special prefetch buffer
    - This avoids cache pollution
  - Very effective for instruction streams
  - Alpha 21064 fetches two cache blocks on a miss (2nd block is a prefetch)
  - Alpha 21164 has a 6-entry prefetch buffer which streams from the Level 2 (L2) cache

O1:T5 - Compiler Prefetch

- **Allow the** compiler **to insert data** prefetching **instructions into the program**
  - Often the compiler has a **very** good idea of what (and when) to prefetch
  - Prefetching allows the compiler to be wrong without incorrect program behavior
  - Some architectures provide prefetch instructions
  - Example

```
for (i=0; i < 1000; i++){
    prefetch a[i+1];
    prefetch b[i+1];
    a[i] = b[i] * c
}
```

  - Compiler can insert prefetch instructions

O2: Techniques for Reducing Miss Penalties

- **Different memory-system structures impose different cache miss penalties**
  - Example: Write-back vs. write-through caches
- **We'll talk about 4 different techniques**
  - T1. Giving priority to reads over writes
  - T2. Sub-block placement
  - T3. Early restart
  - T4. 2nd-level caches

O2:T1 - Give Reads Priority over Writes

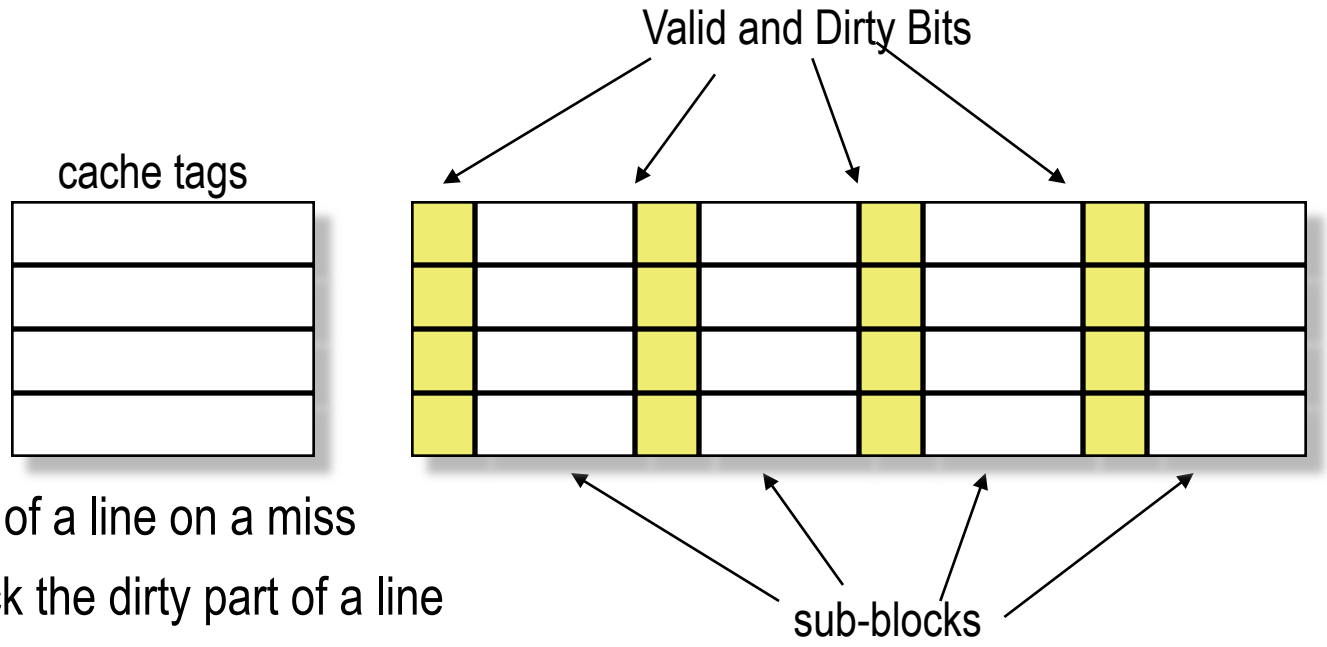
- **Write-buffers allow reads to proceed while writes wait for idle bus cycles**
  - Can have problems
  - Consider this example with a direct-mapped cache

```
SW   512(R0), R3        ; assume cache index is 0
LW   R1,1024(R0)        ; evicts data from previous store
LW   R2,512(R0)         ; misses again
```

  - This is a read-after-write hazard if the write is buffered in the write buffer
  - One solution is to let the write buffer "**drain"** on a read-miss
    - Means you want for all the data in the write buffer to actually write back into the main memory—before you continue with this read-miss
    - MIPS estimates that this would increase average read miss penalty by a factor of 1.5 -- **UGGGGG**!
  - Another solution is to **check for the hazard** (by looking at each entry in the write buffer) and only stall when necessary

O2:T2 - Sub-block Placement

- **Divide a cache line into sub-blocks (sub-lines) and use one-bit per sub-block to denote if the sub-block is valid**



  - Only refill part of a line on a miss
  - Only write-back the dirty part of a line

O2:T3 - Early Restart & Critical Word First

- **Early restart** - as soon as the requested word of the line arrives, send it to the CPU and let the CPU continue execution
- **Critical word first** - have the memory system return the requested word first

O2:T4 - 2nd Level Caches

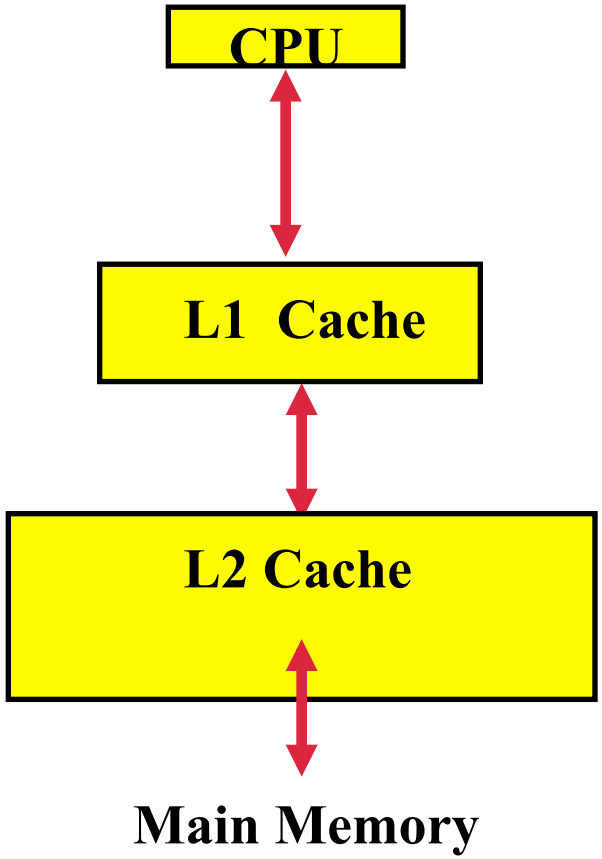
- Time from CPU to main memory continues to grow
- Often, multiple levels of cache are placed in a system
- First level has single-cycle access (in reality, up to 4 cycles)
- Second level usually takes < 10 cycles
- Third level is main memory or another level of cache which can take ~10 - 100 cycles

2nd Level Cache

- **Average memory access time is**

  Hit Time_{L1}
  $+ \text{MissRate}_{L1} * (\text{HitTime}_{L2} +$
  $\text{MissRate}_{L2} * \text{MissPenalty}_{L2})$

- **L2 cache can have longer access time**
  - ▷ can be bigger (more bytes)
  - ▷ can be more associative
  - ▷ can have longer lines



O3: Reducing Hit Time

- **For primary caches (Level 1), the time to access the cache is very important - can determine the overall cycle time of CPU**

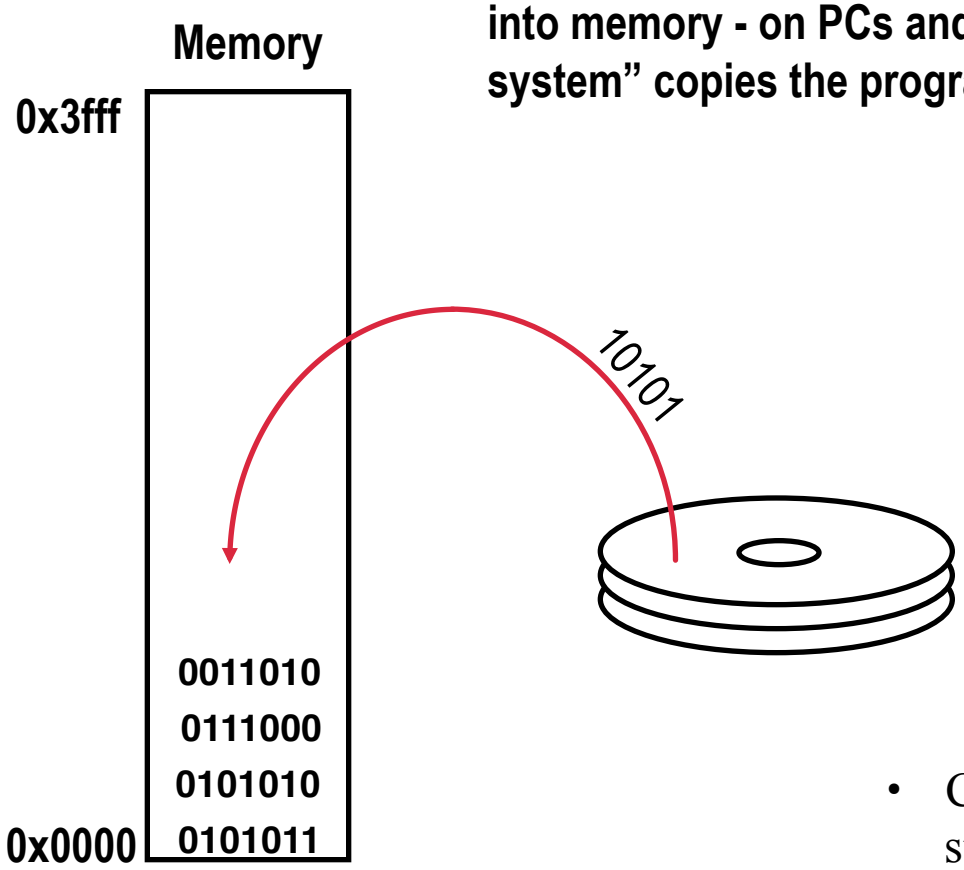
- **Two basic techniques for reducing hit time**
  - Small caches
    - small enough to fit on-chip
    - wires are shorter, less signal propagation time
  - Avoid address translation
    - There are two types of addresses in a machine
      - physical
      - virtual
    - Virtual addresses require a translation to index a physical cache
    - Some designers use virtual caches to avoid the translation time
    - We'll come back to this later

Virtual Memory

- What is it?
- Why do we need it?
- How does it work?
- How do we make it go fast? (later)

How Does a Program Start Running?

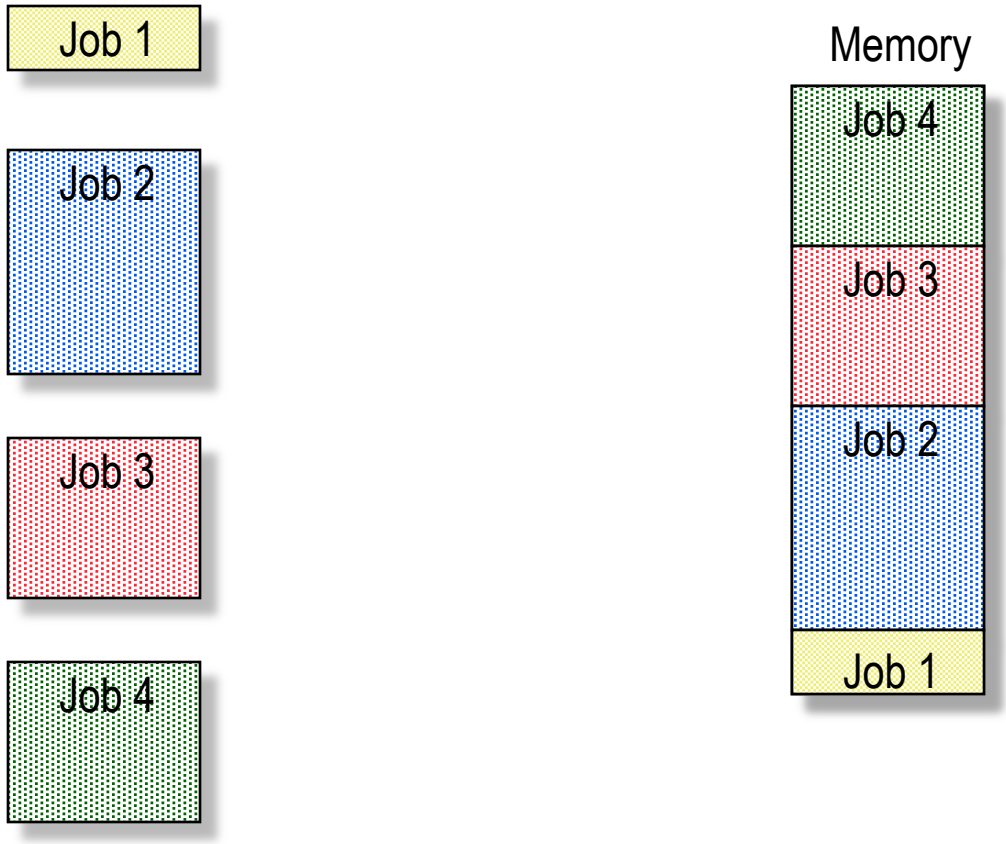
- **The program is copied from permanent storage into memory - on PCs and workstations, the "operating system" copies the program (bits) from disk**



- CPU's Program Counter is then set to starting address of program and program begins execution

Memory Allocation for Multiprogramming

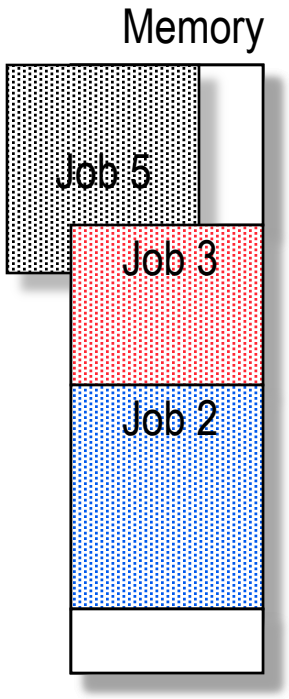
- **Multiple jobs (multiprogramming) each require memory**



4 jobs and their memory requirements

Memory Allocation for Multiprogramming (cont.)

- **Job 5 doesn't fit**
  - There is enough empty memory to hold the program
  - But job 5 won't contiguously fit in either hole
  - We call this external fragmentation



2 MB Unused Memory

Virtual Memory

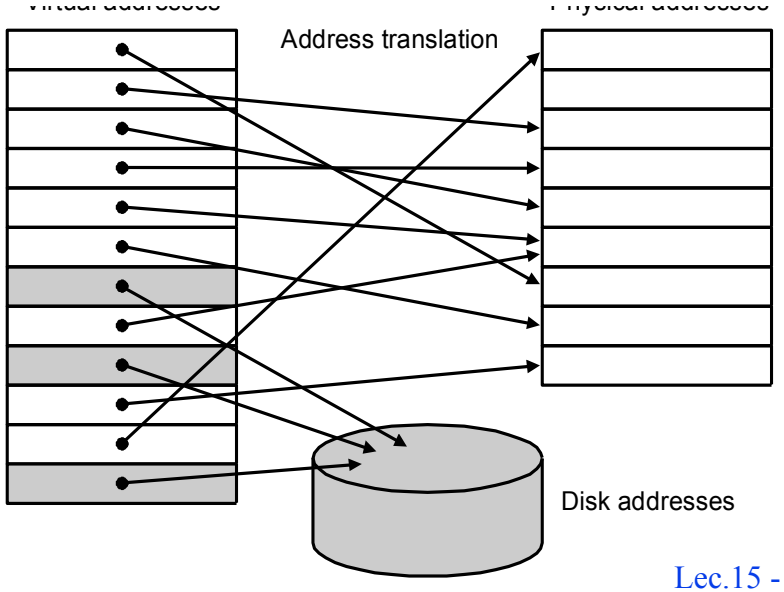
- **What is virtual memory?**
  - Technique that allows execution of a program that
    - can reside in non-contiguous memory locations
    - does not have to completely reside in memory
  - Allows the computer to "fake" a program into believing that its
    - memory is contiguous
    - memory space is larger than physical memory

- **Why is VM important?**
  - Cheap - no longer have to buy lots of RAM
  - Removes burden of memory resource management from the programmer
  - Enables multiprogramming, time-sharing, protection

Virtual Memory

- **Main memory (physical memory) can act as a *cache* for the secondary storage (disk)**

- **Advantages:**
  - illusion of having more and contiguous physical memory
  - program relocation by “pages”
  - protection in multiprogramming



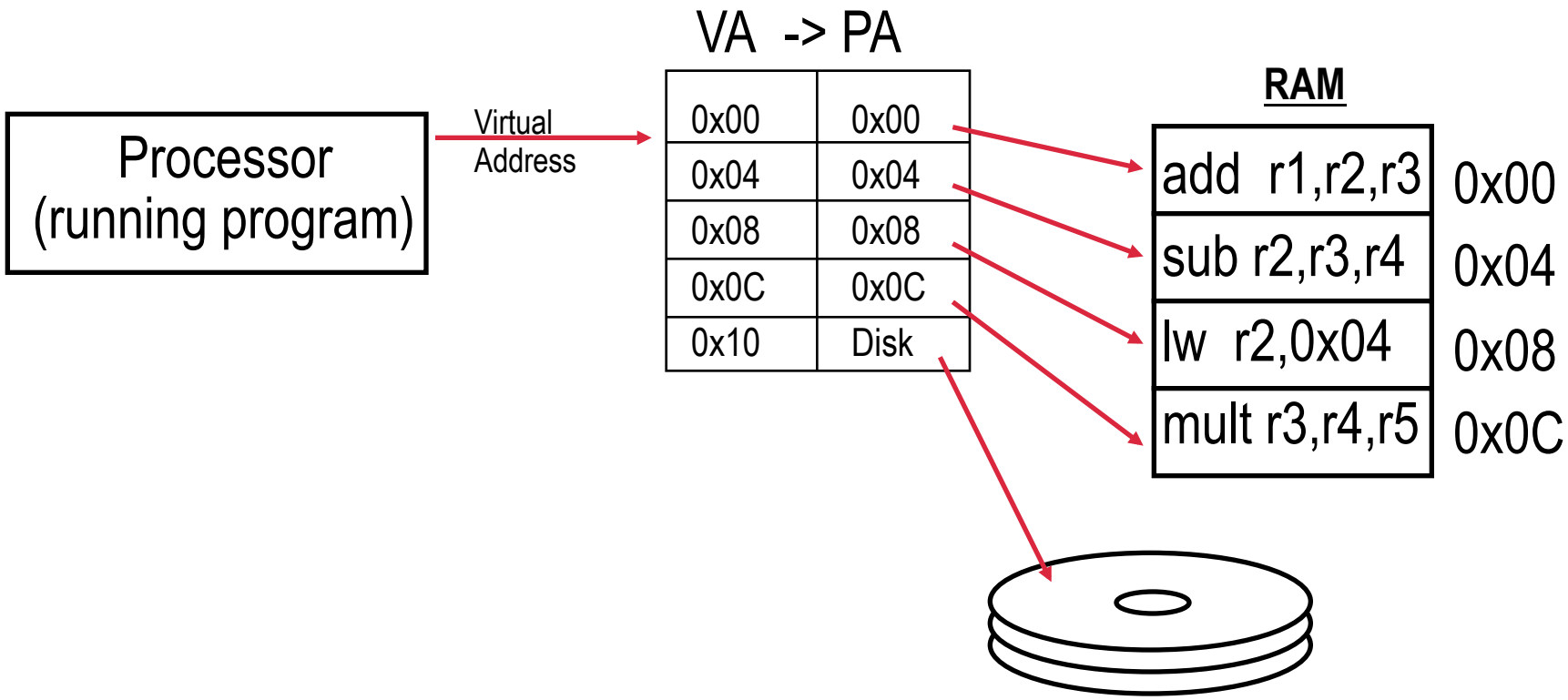
How Does VM Work

- **Two memory "spaces"**
  - **Virtual memory space** - what the program "sees"
  - **Physical memory space** - what the program runs in (size of RAM)

- **On program startup**
  - OS copies program into RAM
  - If there is not enough RAM, OS stops copying program & starts running the program with some portion of the program loaded in RAM
  - When the program touches a part of the program not in physical memory (RAM), OS copies that part of the program from disk into RAM
  - In order to copy some of the program from disk to RAM, OS must evict parts of the program already in RAM
    - OS copies the evicted parts of the program back to disk if the pages are dirty (i.e., if they have been written into, and changed)

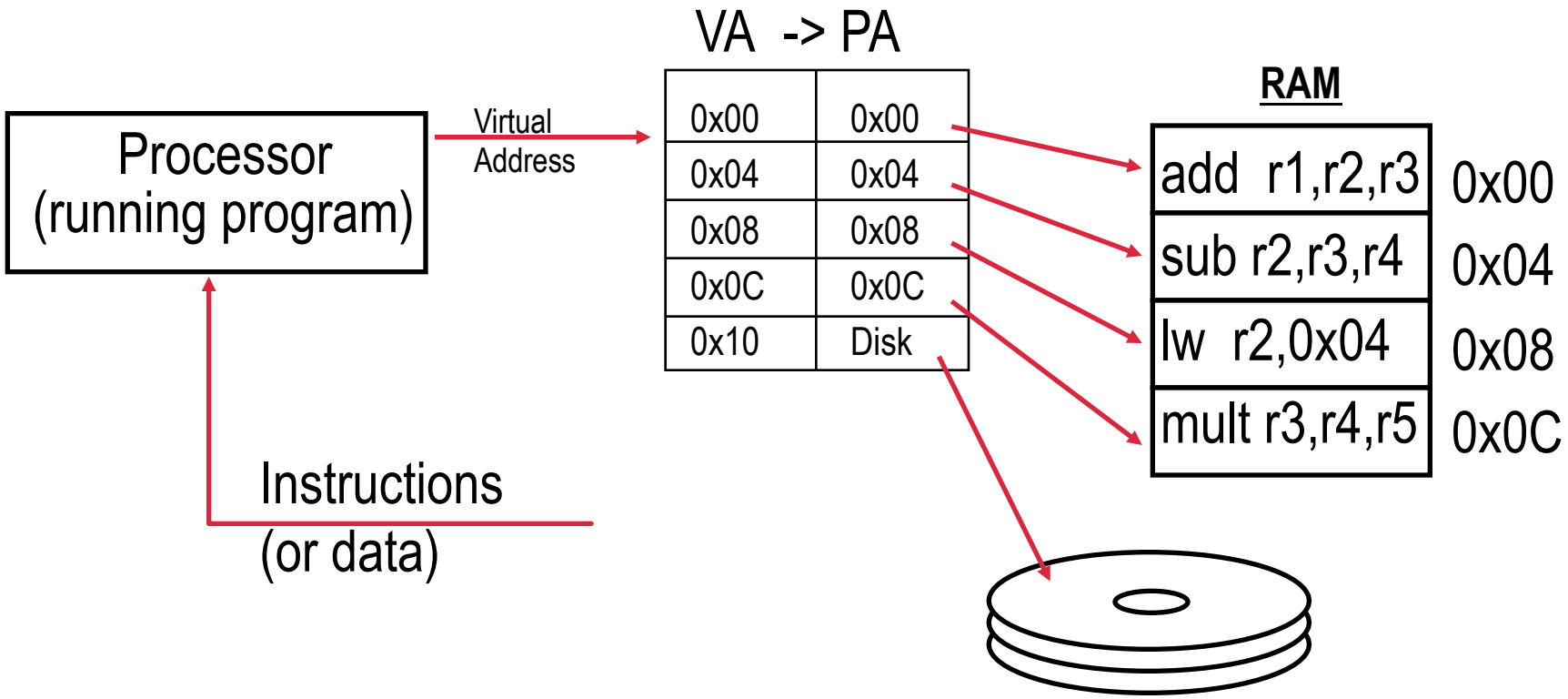
Basic VM Algorithm

- Program presents virtual address (load, store, instruction fetch)
- Computer translates virtual addr. (*VA*) to physical addr. (*PA*)
- Computer reads RAM using PA, returning the data to program



Page Tables

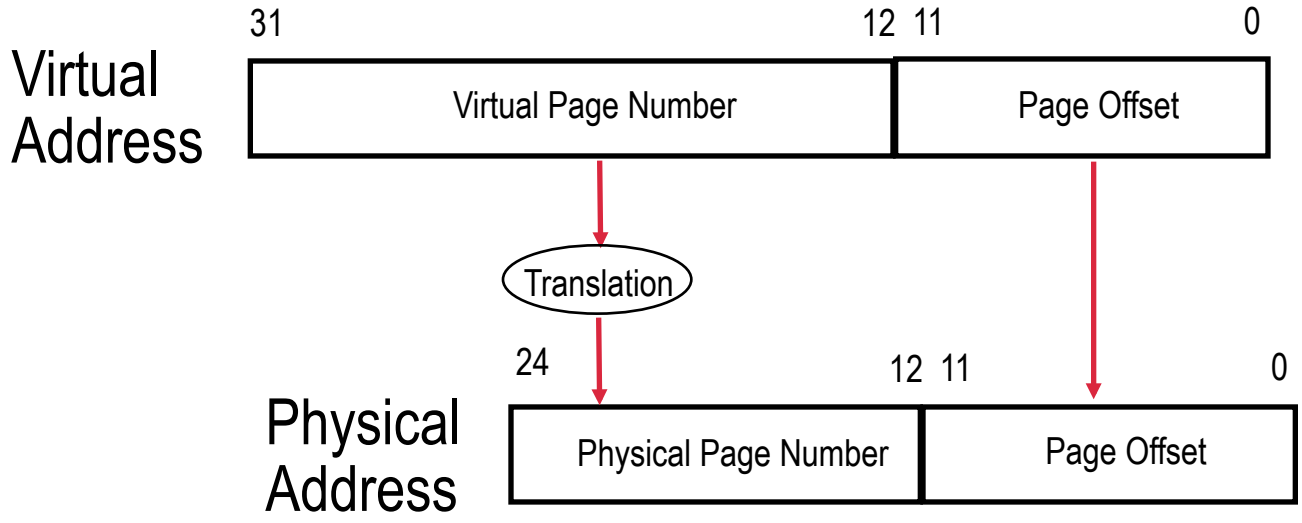
- **Table which holds VA -> PA translations is called the page table**
- **In our current scheme, each word is translated from a virtual address to a physical address**
  - How big is the page table?



Page Tables (cont.)

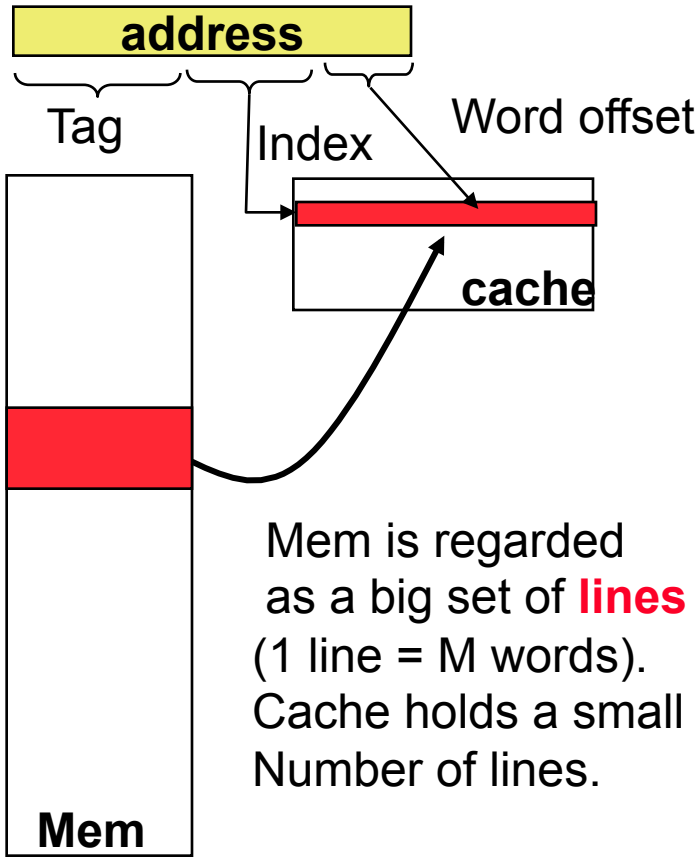
- **Instead of a "fine-grain" VM where any word in VM can map to any RAM word location, we partition memory into bigger chunks called pages**
  - Typical page size today is 4 or 8 KBytes

- **Reduces the number of VA → PA translation entries**
  - Only one translation per page
  - For 4 KByte page, that's one VA → PA translation for every 1,024 words

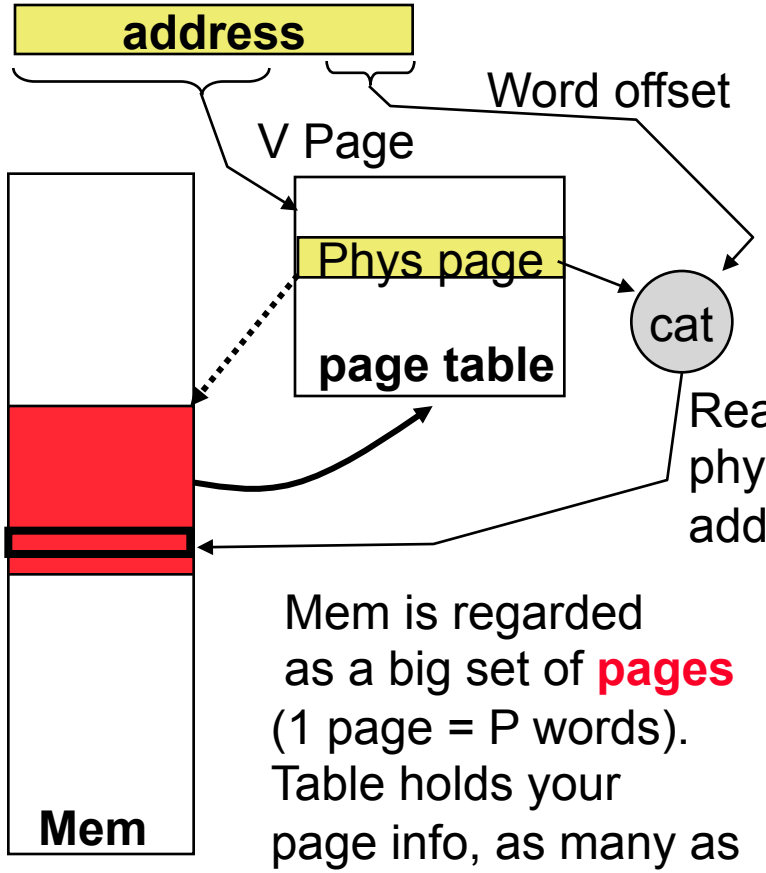


Aside: Caches and VM

- **Idea: *just* like lines in caches**



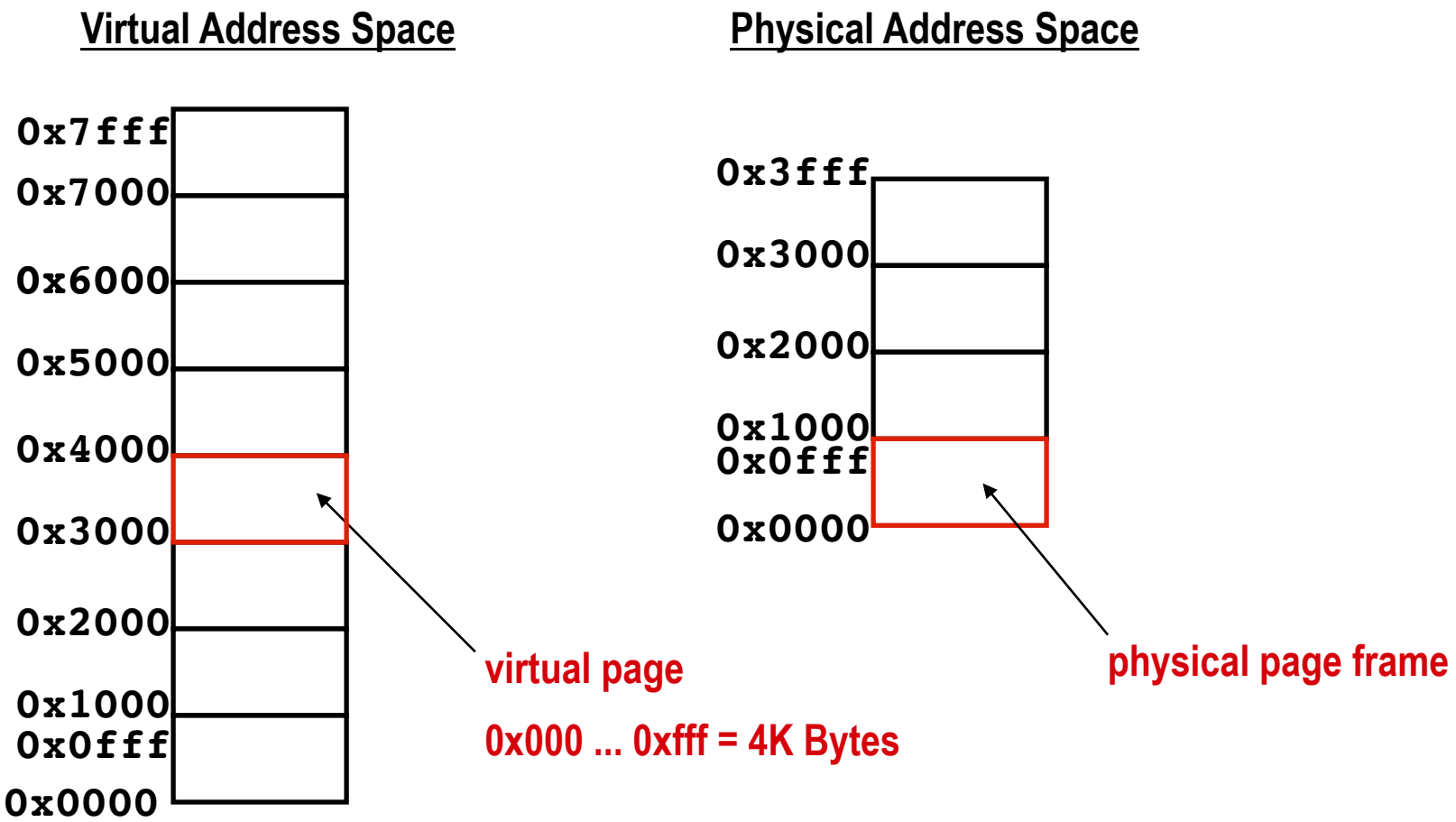
Mem is regarded as a big set of **lines** (1 line = M words). Cache holds a small Number of lines.



Mem is regarded as a big set of **pages** (1 page = P words). Table holds your page info, as many as fit in your physical mem.

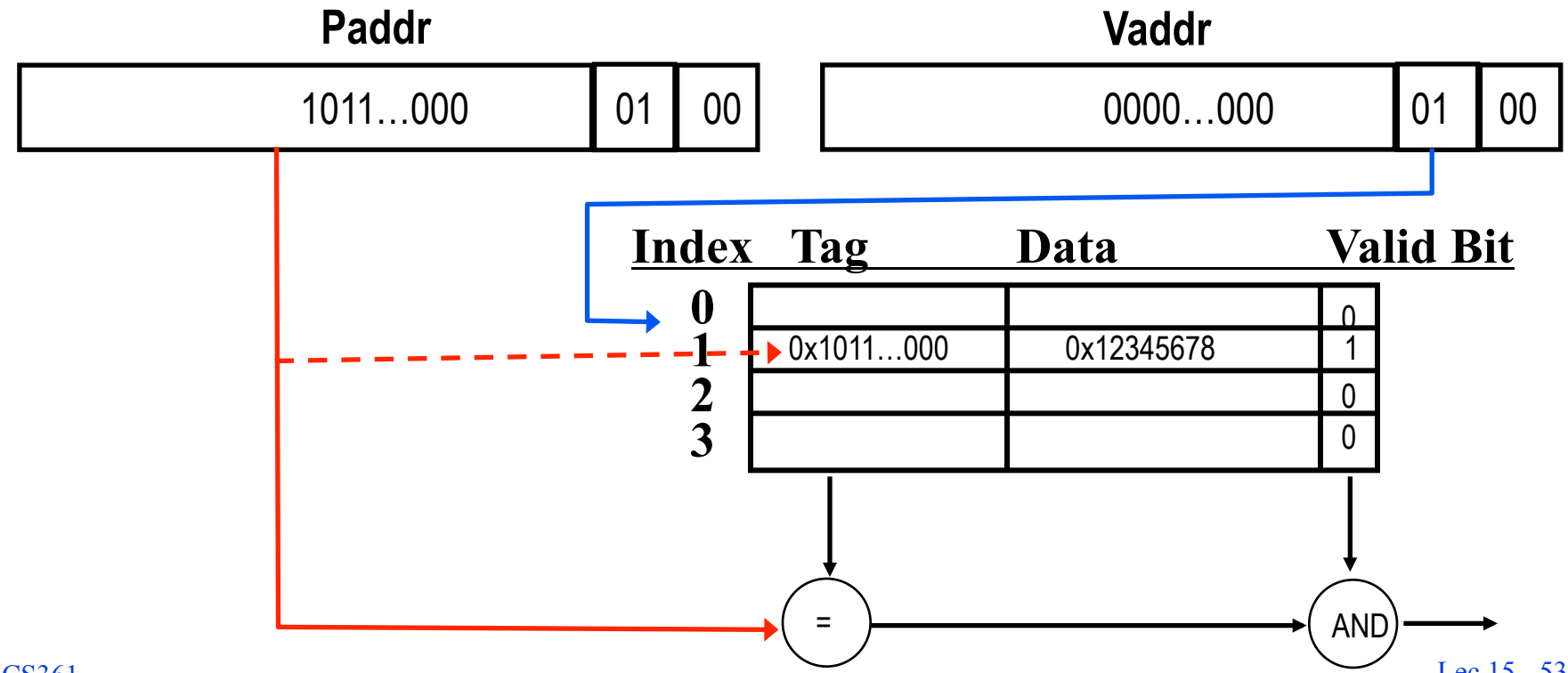
Virtual Pages & Physical Page Frames

- **Every address within a virtual page maps to the same location within a physical page frame**
  - In other words, bottom $\log_2$(page size in bytes) addr bits not translated



Virtual Addresses and Caches

- **Translation Lookaside Buffer: like a cache for page table's virtual-to-physical mappings**
  - May use Vaddr for both cache index and tag (virtual cache)
  - May use Vaddr for cache index, Paddr for tag (virtually-indexed physically-tagged cache)
  - May use Paddr for both cache index and tag (physical cache)



Summary

- **Key approaches to cache performance improvement**
  - Makes a huge difference in apps with adverse memory access patterns
  - E.g., online transaction processing
- **Virtual memory**
  - Gives illusion of a LARGE physical RAM, even if you have LESS real RAM
  - RAM divided into chunks called pages. “Live” pages are in the physical RAM. Pages that don’t fit are on the disk.
  - Hardware translates the virtual address (big address space) into the physical address (real RAM address). Allows a page to be placed anywhere