Lecture 17:

The Memory System 3/3

Introduction to Computer Architecture
UC Davis EEC 170, Fall 2019

Four Questions for Caches and 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)

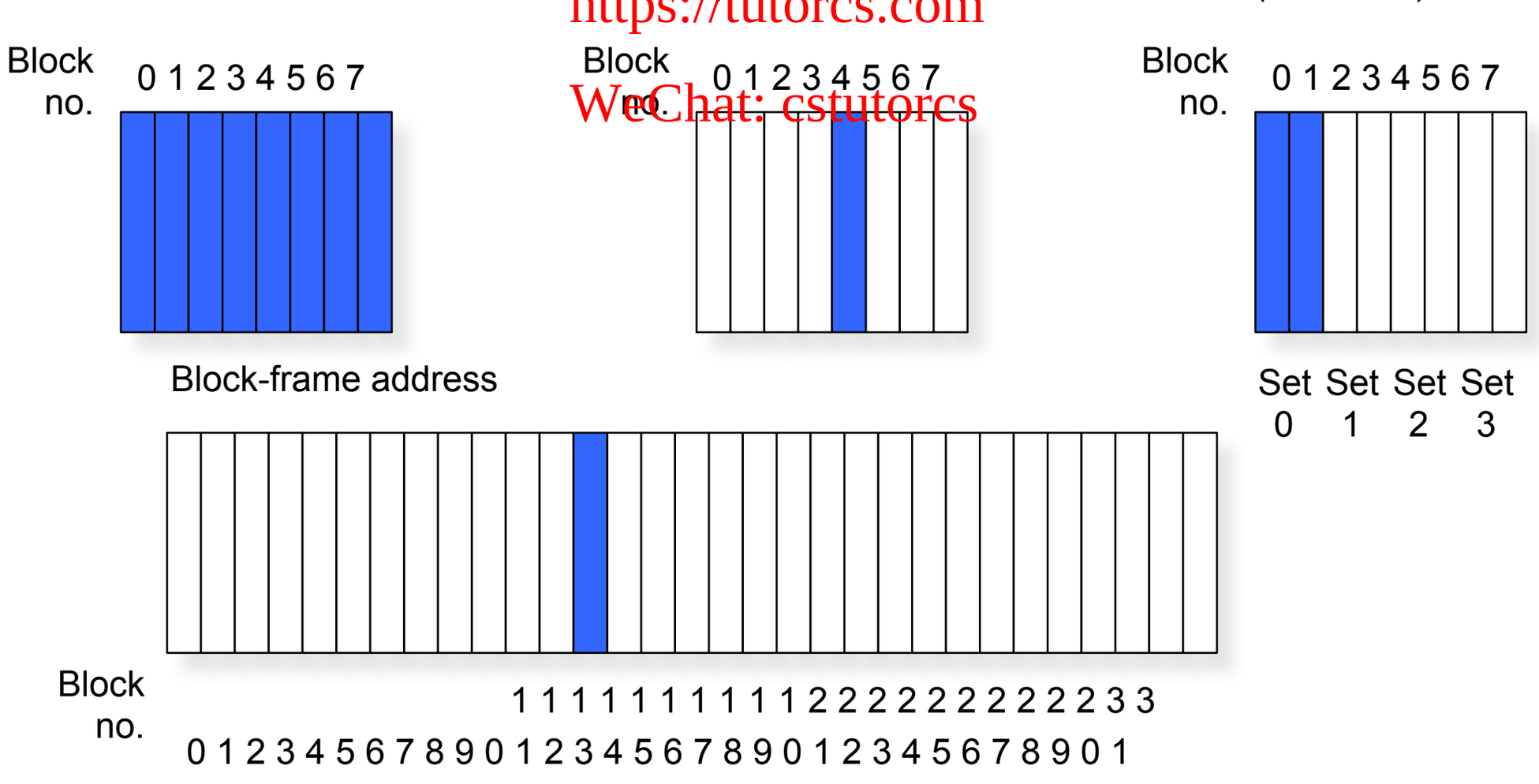
Q1: Where can a block be placed in the upper level?

- Block 12 placed in 8 block cache:
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = Block Number Modulo Number Sets

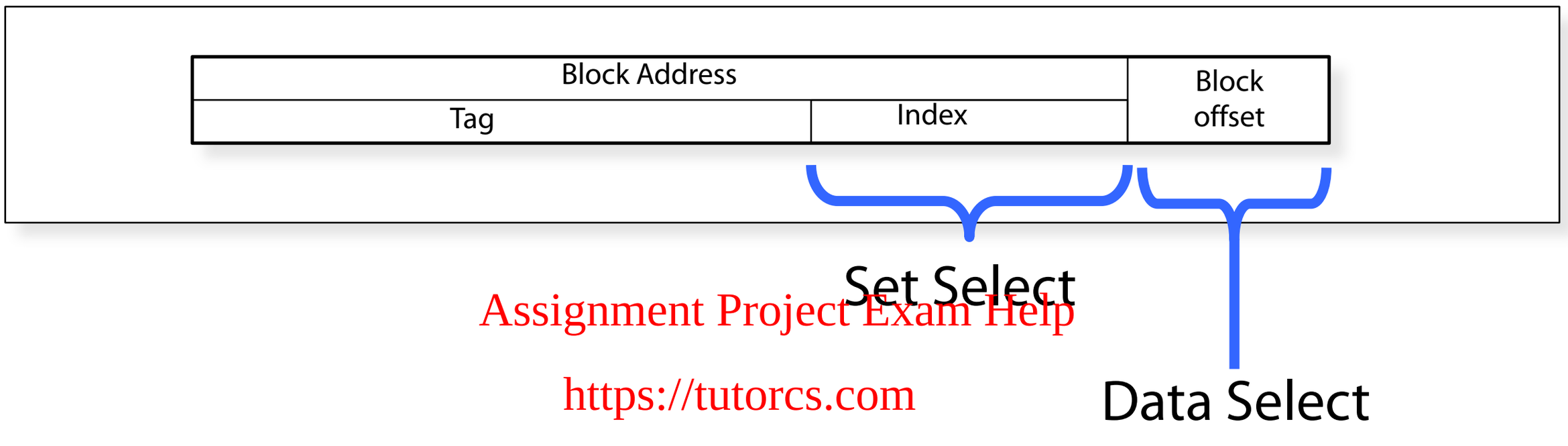
Fully associative: block 12 can go anywhere

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

Set associative: block 12 can go anywhere in set 0 (12 mod 4)



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



- Direct indexing (using index and block offset), tag compares, or combination
- Increasing associativity shrinks index, expands tag

Q3: Which block should be replaced on a miss?

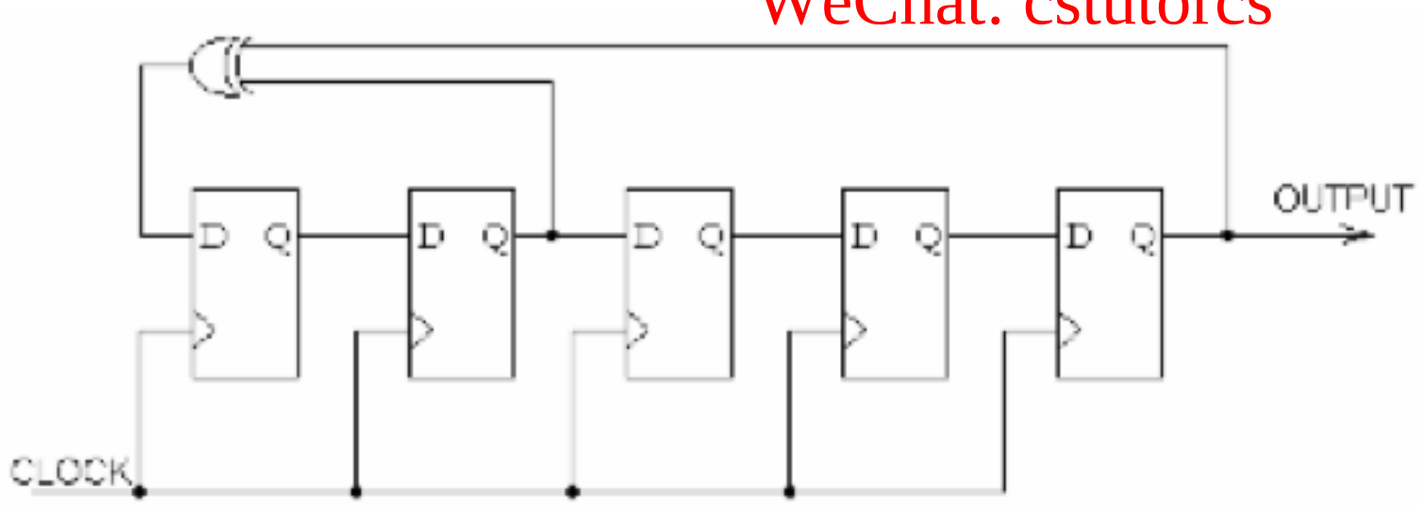
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

Associativity	2 way	2 way	4 way	4 way	8 way	8 way
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Random Replacement (Don McLane, UW)

- Build a single Pseudorandom Number generator for the WHOLE cache. On a miss, roll the dice and throw out a cache line at random.
- Updates only on misses.
- How do you build a random number generator (it's easier than you might think).

Overhead is $O(\log_2 N)$ bits/cache!

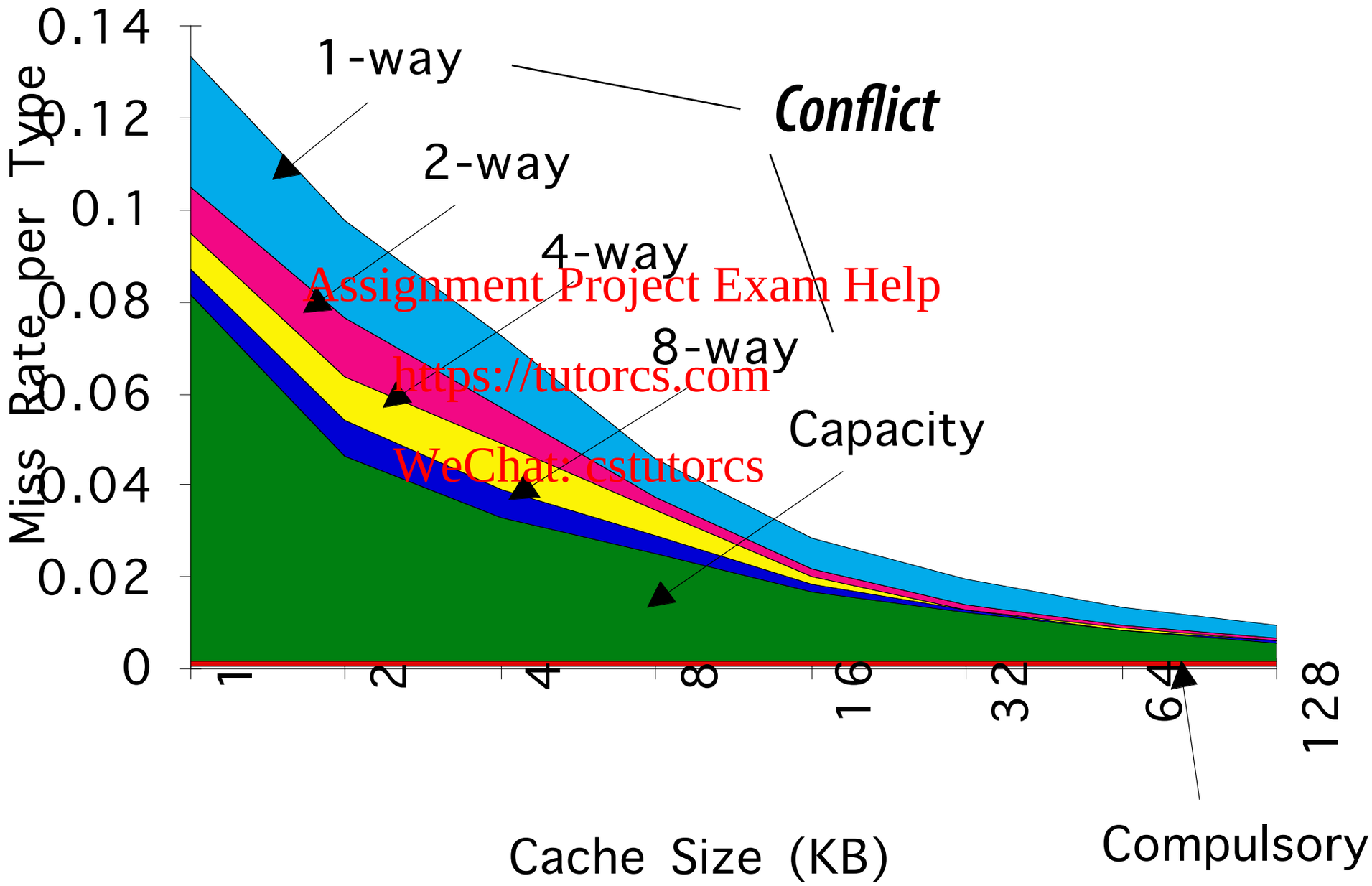


Pseudorandom Linear Feedback Shift Register

Counting Sequence

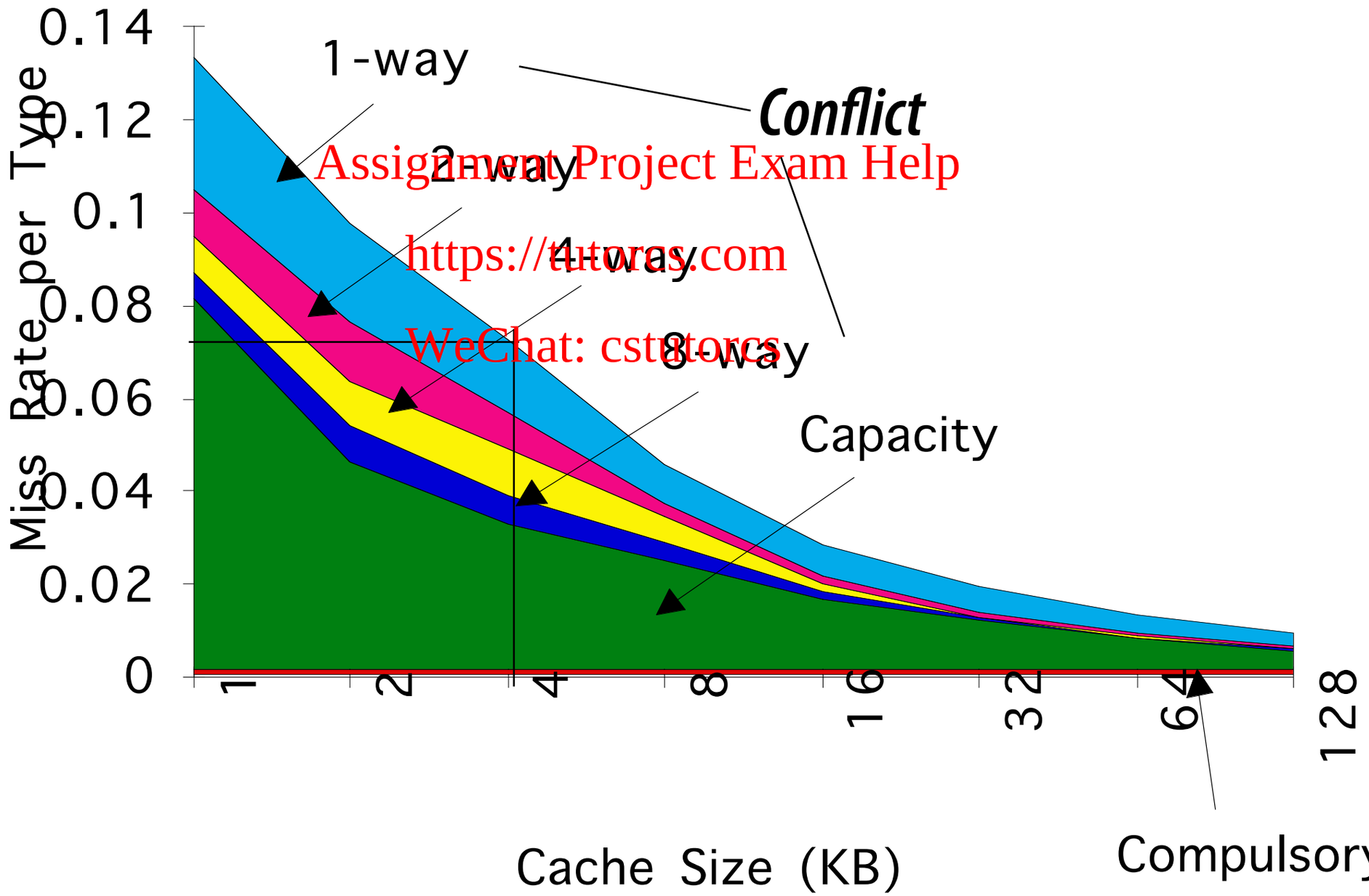
11111	0x1F	01000	0x08
01111	0x0F	10100	0x14
00111	0x07	01010	0x0A
10011	0x13	10101	0x15
11001	0x19	11010	0x1A
01100	0x0C	11101	0x1D
10110	0x16	01110	0x0E
01011	0x0B	10111	0x17
00101	0x05	11011	0x1B
10010	0x12	01101	0x0D
01001	0x09	00110	0x06
00100	0x04	00011	0x03
00010	0x02	10001	0x11
00001	0x01	11000	0x18
10000	0x10	11100	0x1C
		11110	0x1E

3Cs Absolute Miss Rate (SPEC92)

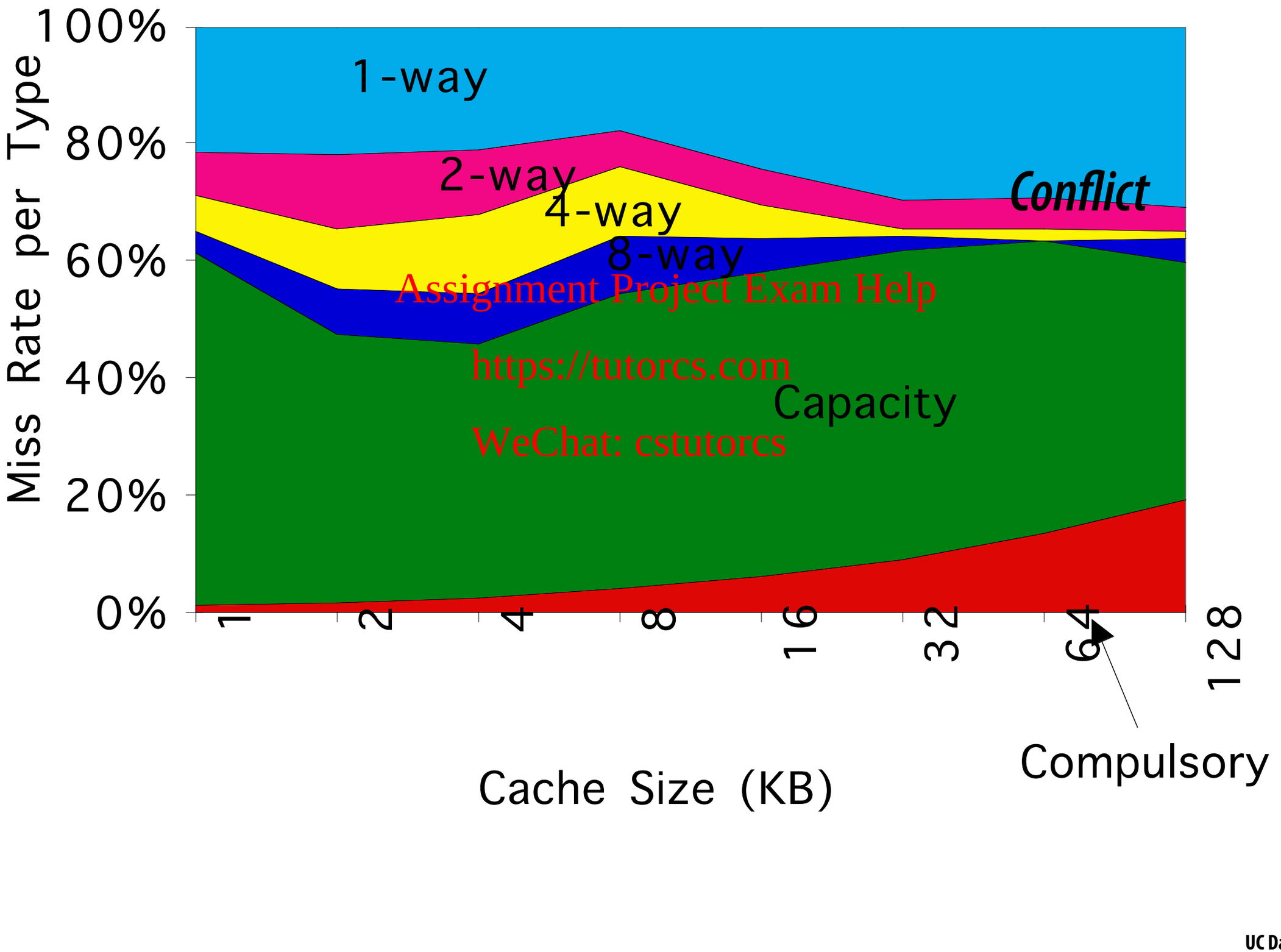


2:1 Cache Rule

- *miss rate 1-way associative cache size X*
  = miss rate 2-way associative cache size X/2



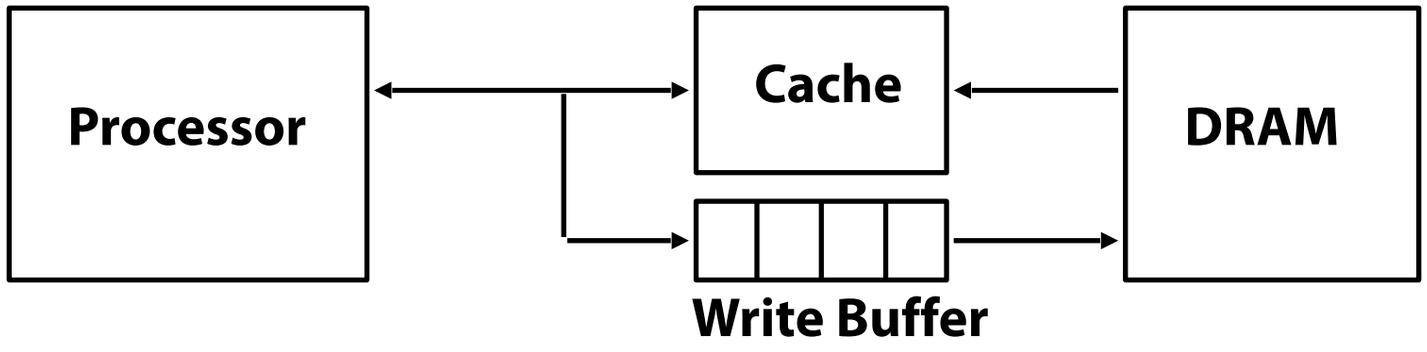
3Cs Relative Miss Rate



Q4: What happens on a write?

- What happens on a write miss?
  - Don't have to read data or check tags—just write!
- Writes are more complex than reads:
  - Desirable: Cache is always a subset of memory ("consistent")
  - So when we write to the cache, we also should write to the memory as well ("write through")
    - If we don't do this: "write back"
- Write-through could lead to performance problems—writing to memory could mean performance is proportional to memory time instead of cache time
- Solution: "write buffer"

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: 4
  - Must handle bursts of writes
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle

Q4: What happens on a write?

- Write through—The information is written to both the block in the cache and to the block in the lower-level memory.
  - On a write miss, do we allocate space in the cache? Typically no, since all writes have to go to main memory anyway. This is called *write no-allocate*.
- Write back—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - On a write miss, do we allocate space in the cache? Typically yes, since we hope future writes might also be captured by the cache. This is called *write allocate*.
- You'll have to make a decision on this in your project design!

Q4: What happens on a write?

- Write through—The information is written to both the block in the cache and to the block in the lower-level memory.
- Write back—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
  - Now have to keep track if block is clean or dirty ("dirty bit")
- Pros and Cons of each?
  - WT: read misses cannot result in writes, also simpler
  - WB: no writes of repeated writes (so less bandwidth), more complex (dirty bit)
- WT always combined with write buffers so that don't wait for more distant memory

Improving Cache Performance: 3 general options

- Time = IC x CT x (ideal CPI + memory stalls)

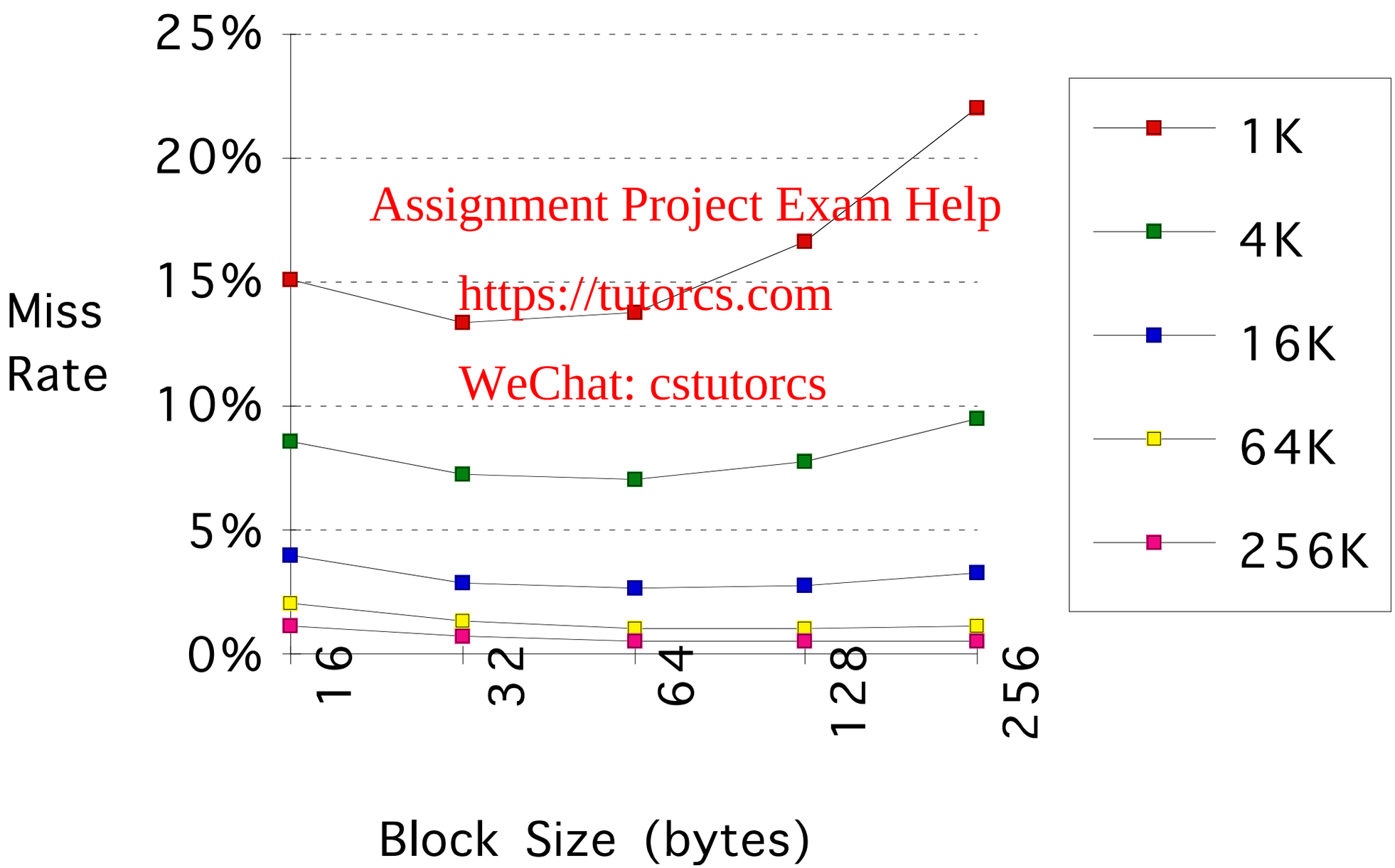
- Average Memory Access time =
  - Hit Time + (Miss Rate x Miss Penalty) =
  - (Hit Rate x Hit Time) + (Miss Rate x Miss Time)
  - [note this means Miss Penalty + Hit Time = Miss Time]

- Options to reduce AMAT:
  1. Reduce the miss rate,
  2. Reduce the miss penalty, or
  3. Reduce the time to hit in the cache.

Improving Cache Performance

1. *Reduce the miss rate,*
2. **Reduce the miss penalty, or**
3. **Reduce the time to hit in the cache.**

1. Reduce Misses via Larger Block Size



2. Reduce Misses via Higher Associativity

- 2:1 Cache Rule:
  - Miss Rate DM cache size N ~= Miss Rate 2-way cache size N/2
- Beware: Execution time is only final measure!
  - Will Clock Cycle time increase?
  - Hill [1988] suggested hit time for 2-way vs. 1-way external cache +10%, internal + 2%

Example: Avg. Memory Access Time vs. Miss Rate

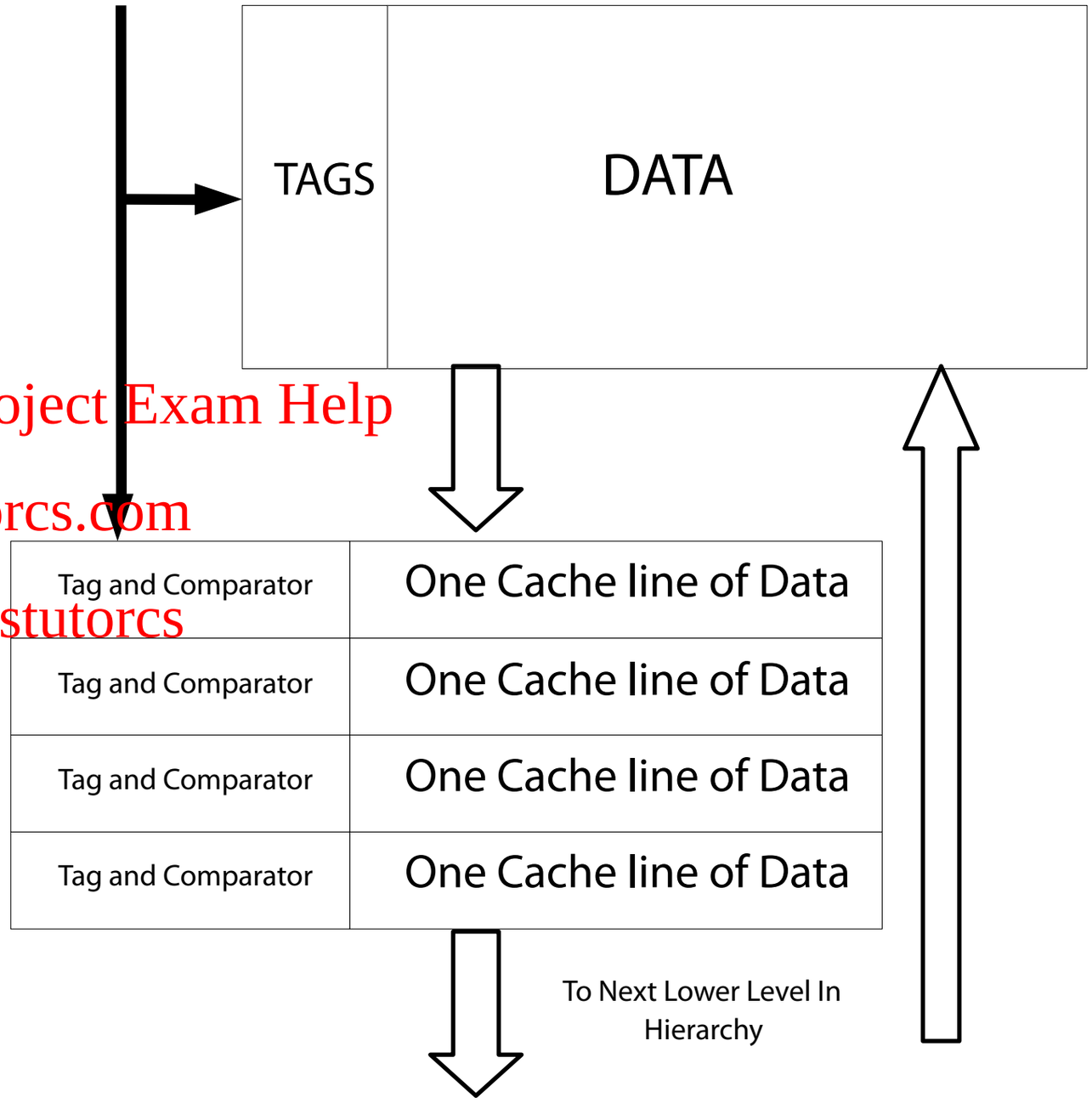
- Assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT direct mapped

	1-way	2-way	4-way	8-way
1 KB	*2.33*	*2.15*	*2.07*	*2.01*
2 KB	*1.98*	*1.86*	*1.76*	*1.68*
4 KB	*1.72*	*1.67*	*1.61*	*1.53*
8 KB	1.46	*1.48*	*1.47*	*1.43*
16 KB	1.29	1.32	1.32	1.32
32 KB	1.20	1.24	1.25	*1.23*
64 KB	1.14	1.20	1.21	1.23
128 KB	1.10	1.17	1.18	1.20

- (Italics means A.M.A.T. improved by more associativity)

3. Reducing Misses via a "Victim Cache"

- How to combine fast hit time of direct mapped yet still avoid conflict misses?
- Add buffer to place data discarded from cache



- Jouppi [1990]: 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Used in Alpha, HP machines
  - And AMD Ryzen

4. Reducing Misses by Hardware Prefetching

- E.g., Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in "stream buffer"
  - On miss check stream buffer
- Works with data blocks too:
  - Jouppi [1990] 1 data stream buffer removed 25% misses from 4 KB cache; 4 streams got 43%
  - Palacharla & Kessler [1994] for scientific programs for 8 streams removed 50% to 70% of misses from 2 64KB, 4-way set associative caches
- Prefetching relies on having extra memory bandwidth that can be used without penalty
  - Could reduce performance if done indiscriminately!

Final Exam Question F03

- You are looking through some MIPS assembly code that your colleague has written. In it you see a strange instruction (lw $r0, 0($r5)) and then a little later in the program, a load from the same memory location into register $r6 (lw $r6, 0($r5)). You check your MIPS notes and confirm that yes, the first instruction writes to register 0, which is hardwired to zero, and thus will not change architecturally visible state. Since it writes to $r0, it doesn't stall. You also confirm this instruction was not used as a nop. However, you know this was written by an ace programmer.

Final Exam Question F03

- (3 points) Why might you see this instruction? (Hint: What would be the effect in hardware of issuing this instruction?

- (2 points) Briefly describe a circumstance in which this technique would reduce performance.

5. Reducing Misses by Software Prefetching Data

- Data Prefetch
  - Locality hints in load/store instructions
  - Cache Prefetch: load into cache
    (MIPS IV, PowerPC, SPARC v9)
    - RISC-V does not appear to have prefetching nor is it discussed in the design document
  - Special prefetching instructions cannot cause faults; a form of speculative execution
  - http://gcc.gnu.org/projects/prefetch.html
- Issuing Prefetch Instructions takes time
  - Is cost of prefetch issues < savings in reduced misses?

MIPS Prefetch

- The PREF (Prefetch) instruction, supported by MIPS32 and MIPS64, takes a hint with one of the following values:

load_modified	data is expected to be read, not modified
store	data is expected to be stored or modified
load_streamed	data is expected to be read but not reused
store_streamed	data is expected to be stored but not reused
load_retained	data is expected to be read and reused extensively
store_retained	data is expected to be stored and reused extensively
writeback_invalidate	data is no longer expected to be used
PrepareForStore	prepare the cache for writing an entire line

http://gcc.gnu.org/projects/prefetch.html

6. Reducing Misses by Compiler Optimizations

- McFarling [1989] reduced cache misses by 75% on 8 KB direct mapped cache, 4 byte blocks in software
- Instructions
  - Reorder procedures in memory so as to reduce conflict misses
  - Profiling to look at conflicts (using tools they developed)
- Data
  - *Merging Arrays*: improve spatial locality by single array of compound elements vs. 2 arrays
  - *Loop Interchange*: change nesting of loops to access data in order stored in memory
  - *Loop Fusion*: Combine 2 independent loops that have same looping and some variables overlap
  - *Blocking*: Improve temporal locality by accessing "blocks" of data repeatedly vs. going down whole columns or rows

Improving Cache Performance (Continued)

1. **Reduce the miss rate,**
2. ***Reduce the miss penalty*, or**
3. **Reduce the time to hit in the cache.**

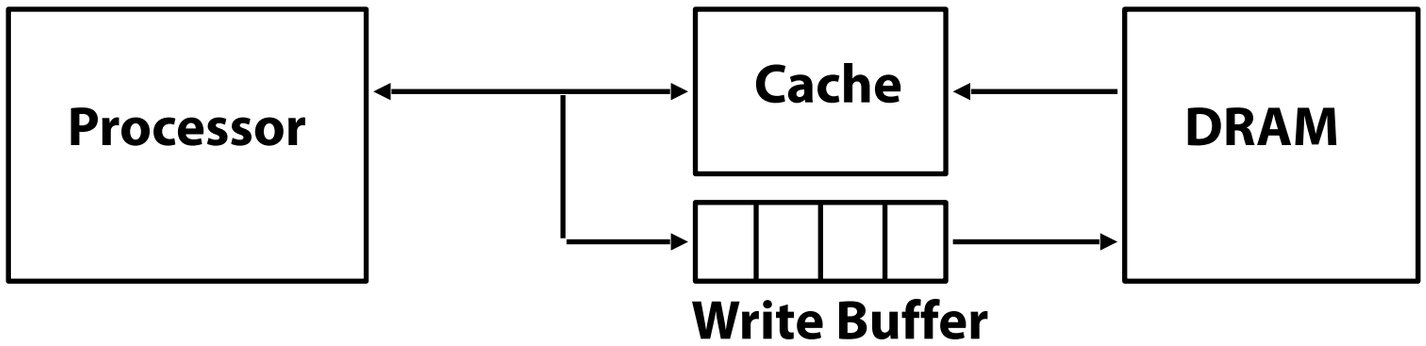
MIPS R2000 cache

- Separate I and D caches
- 4–64 KB
- Write-through
- Direct-mapped
- Physically indexed and tagged
- 4-byte (1-word) cache line

0. Reducing Penalty: Faster DRAM / Interface

- New DRAM Technologies
  - RAMBUS—same initial latency, but much higher bandwidth
  - Synchronous DRAM
  - Merged DRAM/Logic - IRAM project at Berkeley
  - TMJ-RAM (Tunneling magnetic-junction RAM) from IBM
- Better bus interfaces
- CRAY Technique: only use SRAM

1. Reducing Penalty: Read Priority over Write on Miss



- Write buffers complicate memory access
  - Consider write-through: RAW hazard in main memory on cache misses
    - SW R3, 512(R0) # (cache index 0)
    - LW R1, 1024(R0) # (cache index 0)
    - LW R2, 512(R0) # (cache index 0)
- Wait for write buffer to empty?
  - Might increase read miss penalty (old MIPS 1000 by 50%)
- Check write buffer contents before read
  - If no conflicts, let the memory access continue

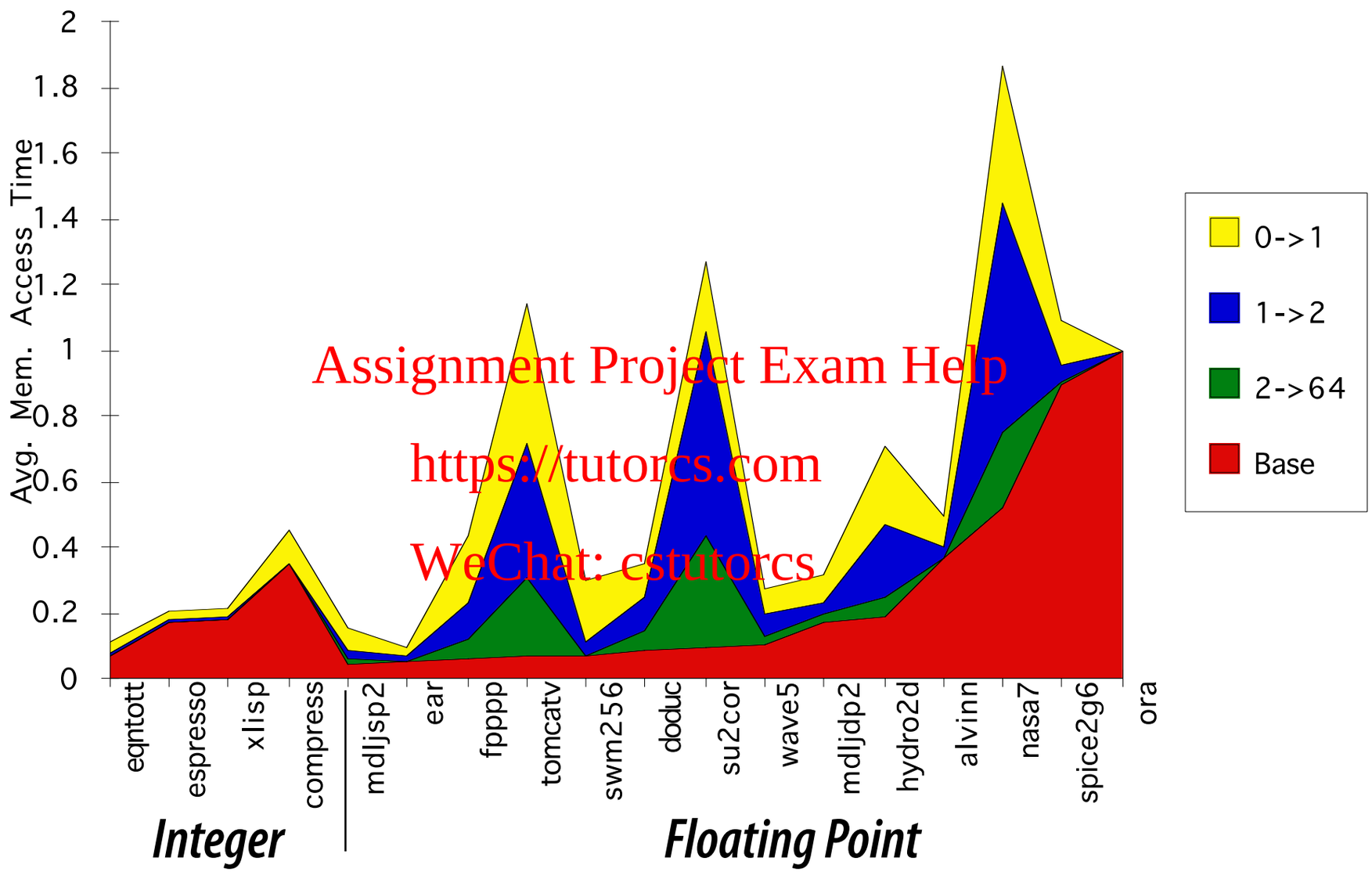
2. Reduce Penalty: Early Restart and Critical Word First

- Don't wait for full block to be loaded before restarting CPU
  - Early restart—As soon as the requested word of the block arrives, send it to the CPU and let the CPU continue execution
  - Critical Word First—Request the missed word first from memory and send it to the CPU as soon as it arrives; let the CPU continue execution while filling the rest of the words in the block. Also called wrapped fetch and requested word first

3. Reduce Penalty: Non-blocking Caches

- Non-blocking cache or lockup-free cache allows data cache to continue to supply cache hits during a miss
  - requires extra Full/Empty bits on registers or out-of-order execution
  - requires multi-bank memories
- "hit under miss" reduces the effective miss penalty by working during miss vs. ignoring CPU requests
- "hit under multiple miss" or "miss under miss" may further lower the effective miss penalty by overlapping multiple misses
  - Significantly increases the complexity of the cache controller as there can be multiple outstanding memory accesses
  - Requires multiple memory banks (otherwise cannot support)
  - Pentium Pro allows 4 outstanding memory misses

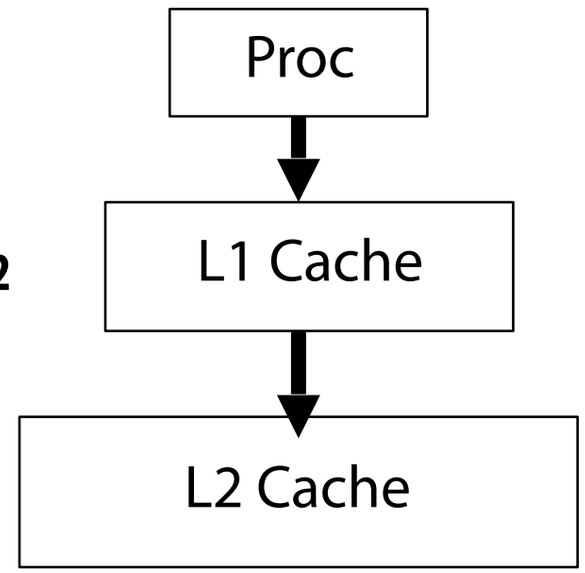
Value of Hit Under Miss for SPEC



- FP programs on average: AMAT= 0.68 -> 0.52 -> 0.34 -> 0.26
- Int programs on average: AMAT= 0.24 -> 0.20 -> 0.19 -> 0.19
- 8 KB Data Cache, Direct Mapped, 32B block, 16 cycle miss

4. Reduce Penalty: Second-Level Cache

- L2 Equations
  - $\text{AMAT} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} \times \text{Miss Penalty}_{L1}$
  - $\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2}$
- $\text{AMAT} = \text{Hit Time}_{L1} +$
  $\text{Miss Rate}_{L1} \times (\text{Hit Time}_{L2} + \text{Miss Rate}_{L2} \times \text{Miss Penalty}_{L2})$
- Definitions:
  - Local miss rate—misses in this cache divided by the total number of memory accesses to this cache (Miss rate_{L2})
    - This miss rate often looks really high because it's the hard cases
  - Global miss rate—misses in this cache divided by the total number of memory accesses generated by the CPU
    ($\text{Miss Rate}_{L1} \times \text{Miss Rate}_{L2}$)
    - Global Miss Rate is what matters



L2: Global vs. Local Miss Rate

- Consider a L1 cache that hits 95% of the time, miss penalty = 50 cycles. L2 cache hits 80% of the time, miss penalty = 500 cycles.
- What is AMAT?
- AMAT = 1 + (0.05*50) + (0.05*0.2*500)
  = 1 + 2.50 + 5 = 8.5

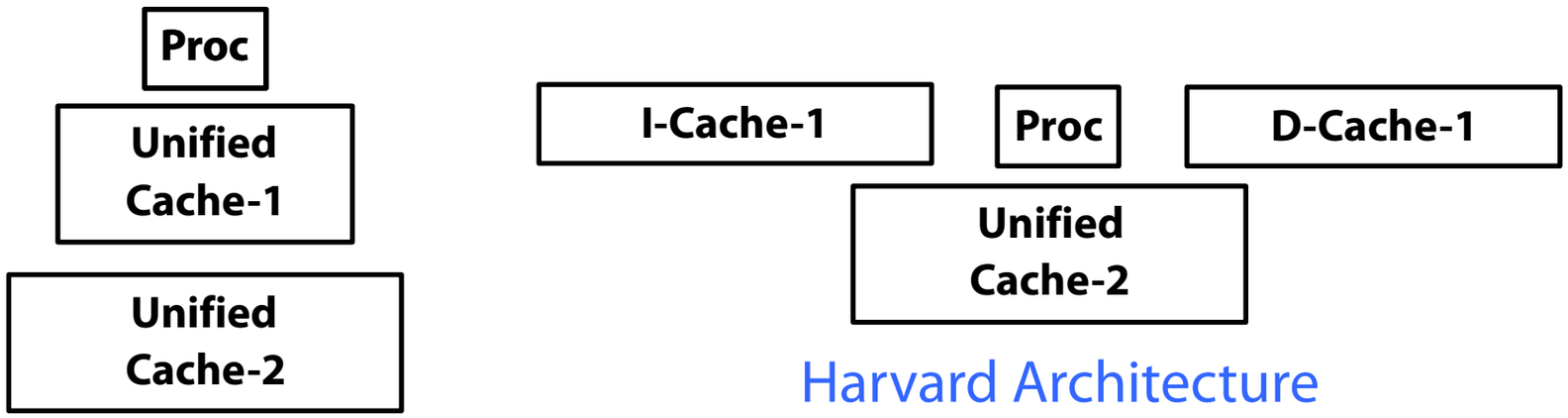
4. Reduce Penalty: Second-Level Cache

- Local miss rate—misses in this cache divided by the total number of memory accesses to this cache (Miss $rate_{L2}$)
- Global miss rate—misses in this cache divided by the total number of memory accesses generated by the CPU (Miss $Rate_{L1}$ x Miss $Rate_{L2}$)
- Global Miss Rate is what matters

Improving Cache Performance (Continued)

- 1. Reduce the miss rate,
- 2. Reduce the miss penalty, or
- 3. *Reduce the time to hit in the cache*:
  - Lower Associativity (+victim caching or 2nd-level cache)?
  - To avoid hit penalty, . . schedule pipeline to tolerate multiple-cycle cache access (common in high-end processors)
  - Careful Virtual Memory Design (next lecture)
  - Harvard Architecture

Example: Harvard Architecture



- **Sample Statistics:**
  - 16KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32KB unified: Aggregate miss rate=1.99%
- **Which is better (ignore L2 cache)?**
  - Assume 33% loads/store, hit time=1, miss time=50
  - Note: data hit has 1 stall for unified cache (only one port)

Example: Harvard Architecture

- Sample Statistics:
  - 16 KB I&D: Inst miss rate=0.64%, Data miss rate=6.47%
  - 32 KB unified: Aggregate miss rate=1.99%
- Which is better (ignore L2 cache)?
  - Assume 33% loads/store, hit time=1, miss time=50
  - Note: data hit has 1 stall for unified cache (only one port)

- $AMAT_{Harvard}=(1/1.33)x(1+0.64\%x50)+(0.33/1.33)x(1+6.47\%x50) = 2.05$
- $AMAT_{Unified}=(1/1.33)x(1+1.99\%x50)+(0.33/1.33)x(1+1+1.99\%x50)= 2.24$

Intel Pentium 4 vs. AMD Opteron

	Intel P4	AMD Opteron
L1 cache org	Separate I/D	Separate I/D
L1 size	8 KB data, 96 KB trace	64 KB each I/D
L1 assoc	4-way	2-way
L1 replacement	Approx LRU	LRU
L1 blocks	64 B	64 B
L1 write	Write-through	Write-back
L2 cache org	Unified	Unified
L2 size	512 KB	1024 KB
L2 assoc	8-way	16-way
L2 replacement	Approx LRU	Approx LRU
L2 blocks	128 B	64 B
L2 write	Write-back	Write-back

AMD Ryzen Memory Hierarchy

- L0 µOP cache:
  - 2,048 µOPs, 8-way set associative
    - 32 sets, 8-µOP line size
  - Parity protected
- L1I Cache:
  - 64 KiB 4-way set associative
    - 256 sets, 64 B line size
    - Shared by the two threads per core
  - Parity protected
- L1D Cache:
  - 32 KiB 8-way set associative
    - 64-sets, 64 B line size
    - Write-back policy
  - 4–5 cycles latency for Int, 7–8 cycles latency for FP
  - SEC-DED ECC
- L2 Cache:
  - 512 KiB 8-way set associative
  - 1,024-sets, 64 B line size
  - Write-back policy
  - Inclusive of L1
  - Latency: 17 cycles latency (ONLY Summit Ridge), 12 cycles latency (All others)
  - DEC-TED ECC
- L3 Cache:
  - Victim cache
  - Summit Ridge, Naples: 8 MiB/CCX, shared across all cores.
  - Raven Ridge: 4 MiB/CCX, shared across all cores.
  - 16-way set associative (8,192 sets, 64 B line size)
  - 40 cycles latency
  - DEC-TED ECC
- System DRAM:
  - 2 channels per die
  - Summit Ridge: up to PC4-21300U (DDR4-2666 UDIMM)
  - Raven Ridge: up to PC4-23466U (DDR4-2933 UDIMM)
  - Naples: up to PC4-21300L (DDR4-2666 RDIMM/ LRDIMM)
  - ECC support: x4 DRAM device failure correction (Chipkill), x8 SEC-DED ECC, Patrol and Demand scrubbing, Data poisoning

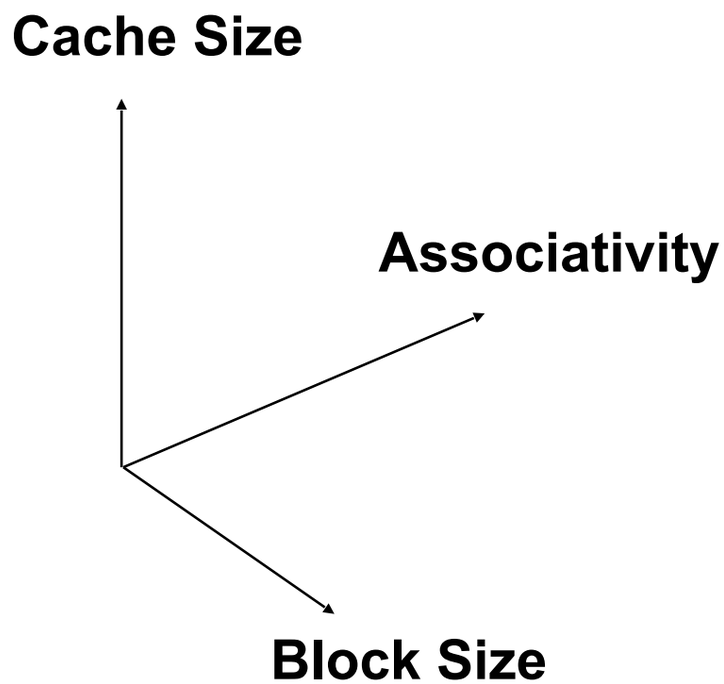
https://en.wikichip.org/wiki/amd/microarchitectures/zen#Memory_Hierarchy

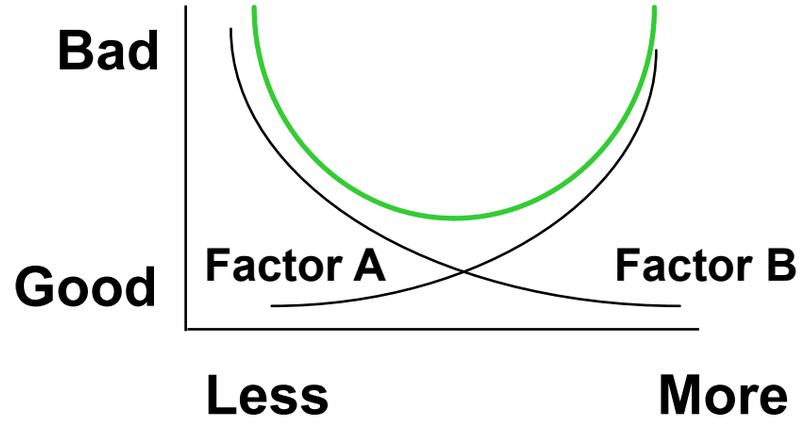
Summary 1/2

- The Principle of Locality:
  - Program likely to 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 (+1) Major Categories of Cache Misses:
  - Compulsory Misses: sad facts of life. Example: cold start misses.
  - Conflict Misses: increase cache size and/or associativity.
    Nightmare Scenario: ping pong effect!
  - Capacity Misses: increase cache size
  - Coherence Misses: Caused by external processors or I/O devices
- Cache Design Space
  - total size, block size, associativity
  - replacement policy
  - write-hit policy (write-through, write-back)
  - write-miss policy

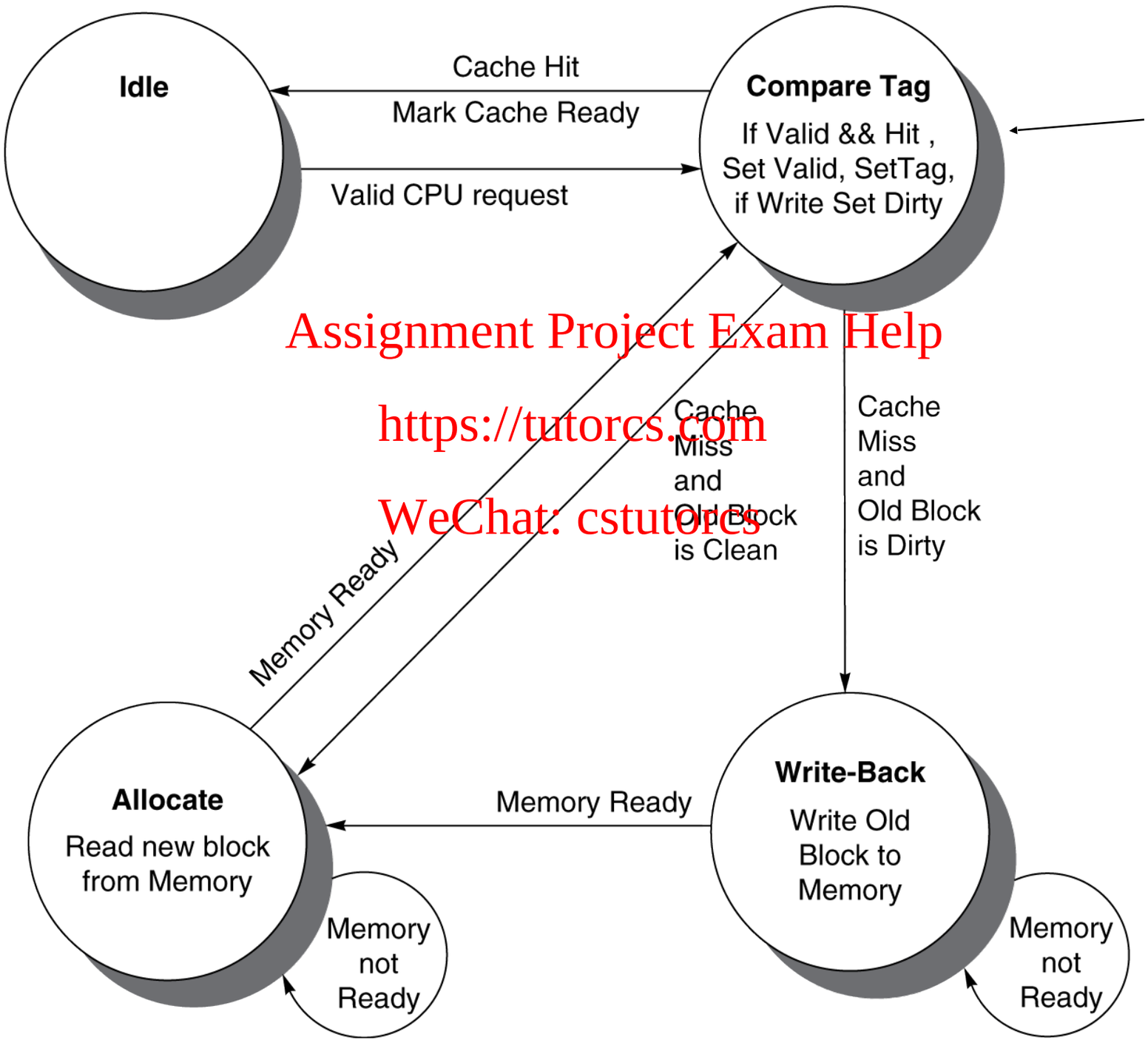
Summary #2 / 2: The Cache Design Space

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





Cache Controller FSM



Cache Coherence Problem

- Suppose two CPU cores share a physical address space
  - For simplicity: write-through caches

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

Coherence Defined

- Informally: Reads return most recently written value
- Formally:
  - P writes X; P reads X (no intervening writes)
    ⇒ read returns written value
  - P_1 writes X; P_2 reads X (sufficiently later)
    ⇒ read returns written value
    - c.f. CPU B reading X after step 3 in example
  - P_1 writes X, P_2 writes X
    ⇒ all processors see writes in the same order
    - End up with the same final value for X

Memory Consistency

- When are writes seen by other processors
  - "Seen" means a read returns the written value
  - Can't be instantaneously
- Assumptions
  - A write completes only when all processors have seen it
  - A processor does not reorder writes with other accesses
- Consequence
  - P writes X then writes Y

    ⇒ all processors that see new Y also see new X
  - Processors can reorder reads, but not writes

Cache Coherence Protocols

- Operations performed by caches in multiprocessors to ensure coherence
  - Migration of data to local caches
    - Reduces bandwidth for shared memory
  - Replication of read-shared data
    - Reduces contention for access
- Snooping protocols
  - Each cache monitors bus reads/writes
- Directory-based protocols
  - Caches and memory record sharing status of blocks in a directory

Invalidating Snooping Protocols

- Cache gets exclusive access to a block when it is to be written
  - Broadcasts an invalidate message on the bus
  - Subsequent read in another cache misses
    - Owning cache supplies updated value

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidate for X	1		0
CPU B read X	Cache miss for X	1	1	1