Lecture 16:

The Memory System 2/3

Introduction to Computer Architecture
UC Davis EEC 170, Fall 2019

Locality Properties of Patterns

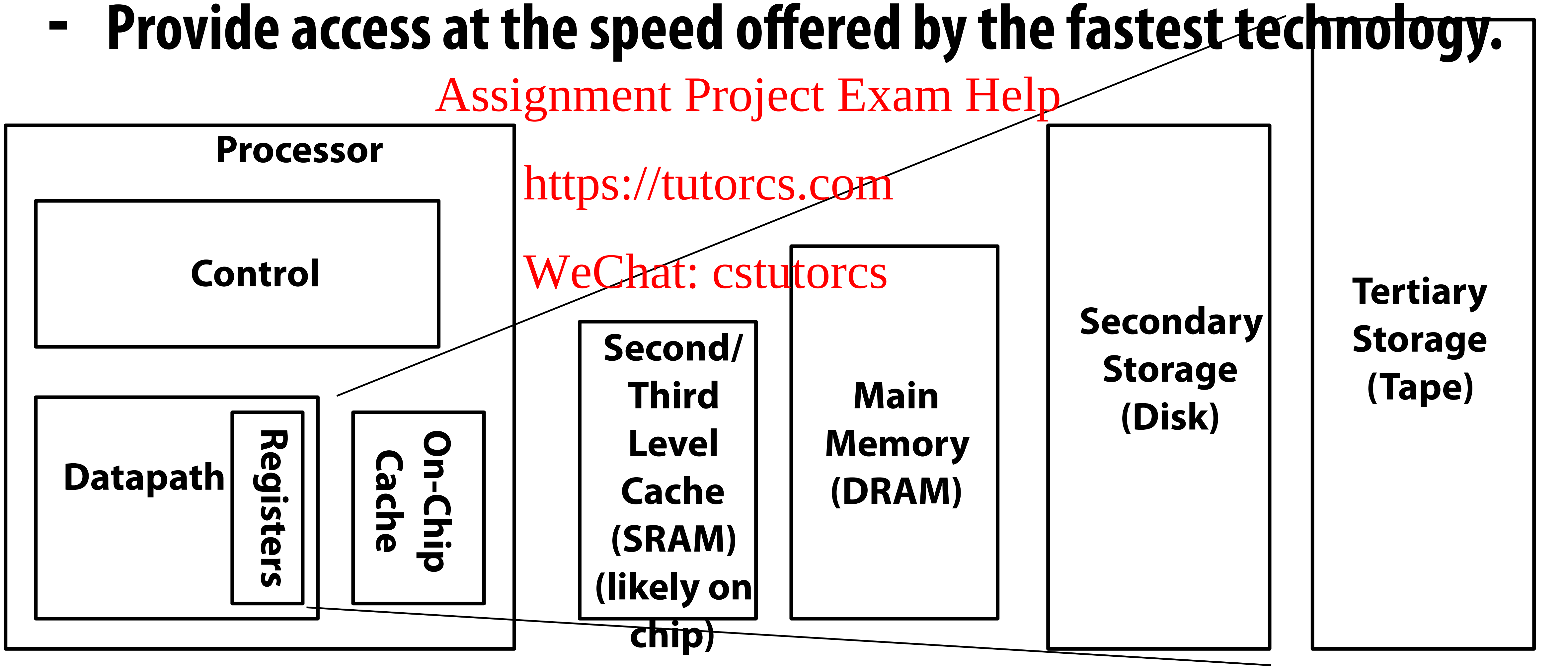
- Two locality properties of memory references:
  - *Temporal* Locality: If a location is referenced it is likely to be referenced again in the near future.
  - *Spatial* Locality: If a location is referenced it is likely that locations near it will be referenced in the near future.

Caches

- **Caches exploit both properties of patterns.**
  - **Exploit *temporal* locality by remembering the contents of recently accessed locations.**
  - **Exploit *spatial* locality by remembering blocks of contents of recently accessed locations.**

Memory Hierarchy of a Modern Computer System

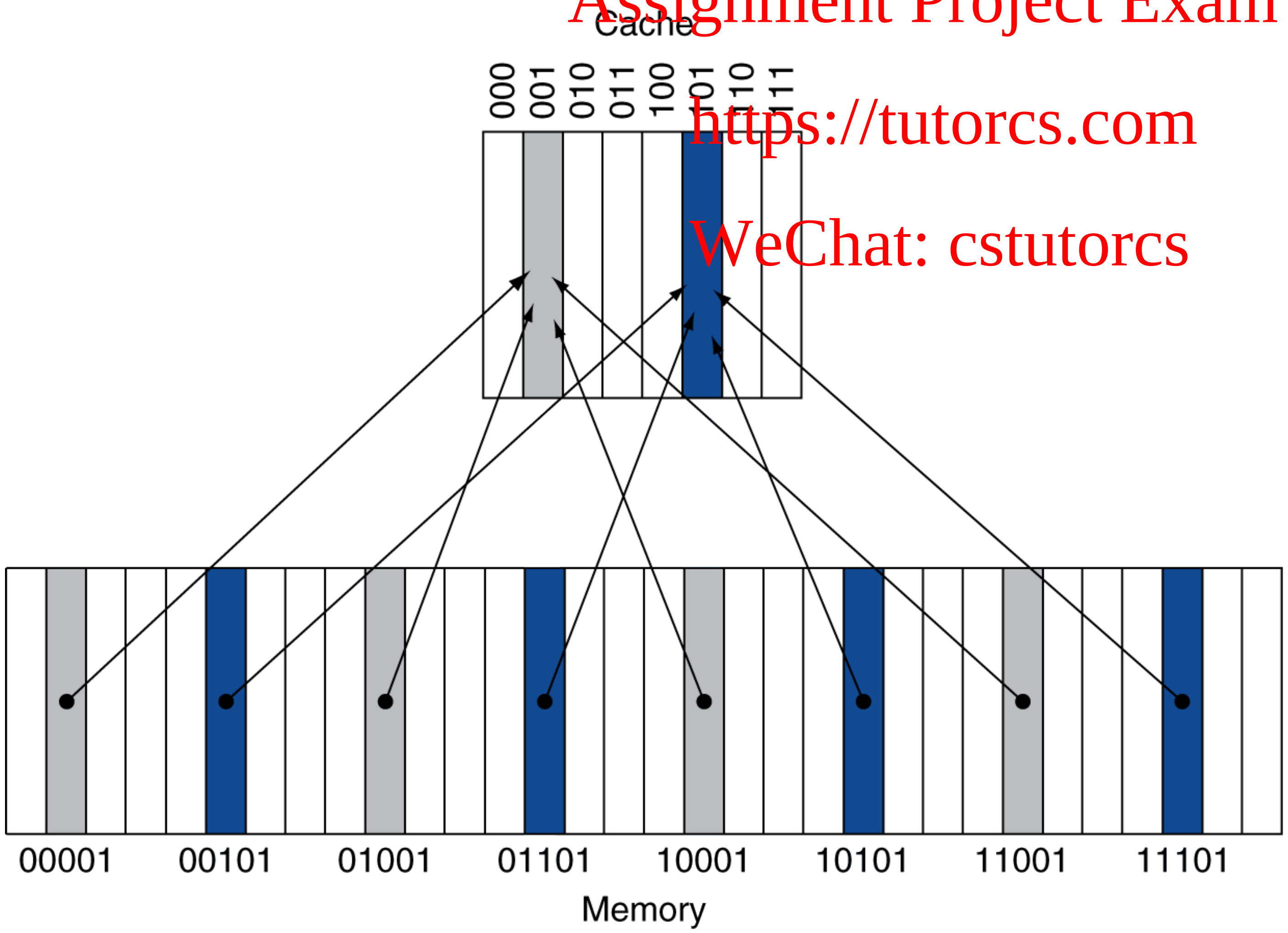
- By taking advantage of the principle of *locality*:
  - Present the user with as much memory as is available in the cheapest technology.
  - Provide access at the speed offered by the fastest technology.



Speed (ns):	1s	10s	100s	10,000,000s (10s ms)	10,000,000,000s (10s sec)
Size (bytes):	100s	Ks	Ms	Gs	Ts

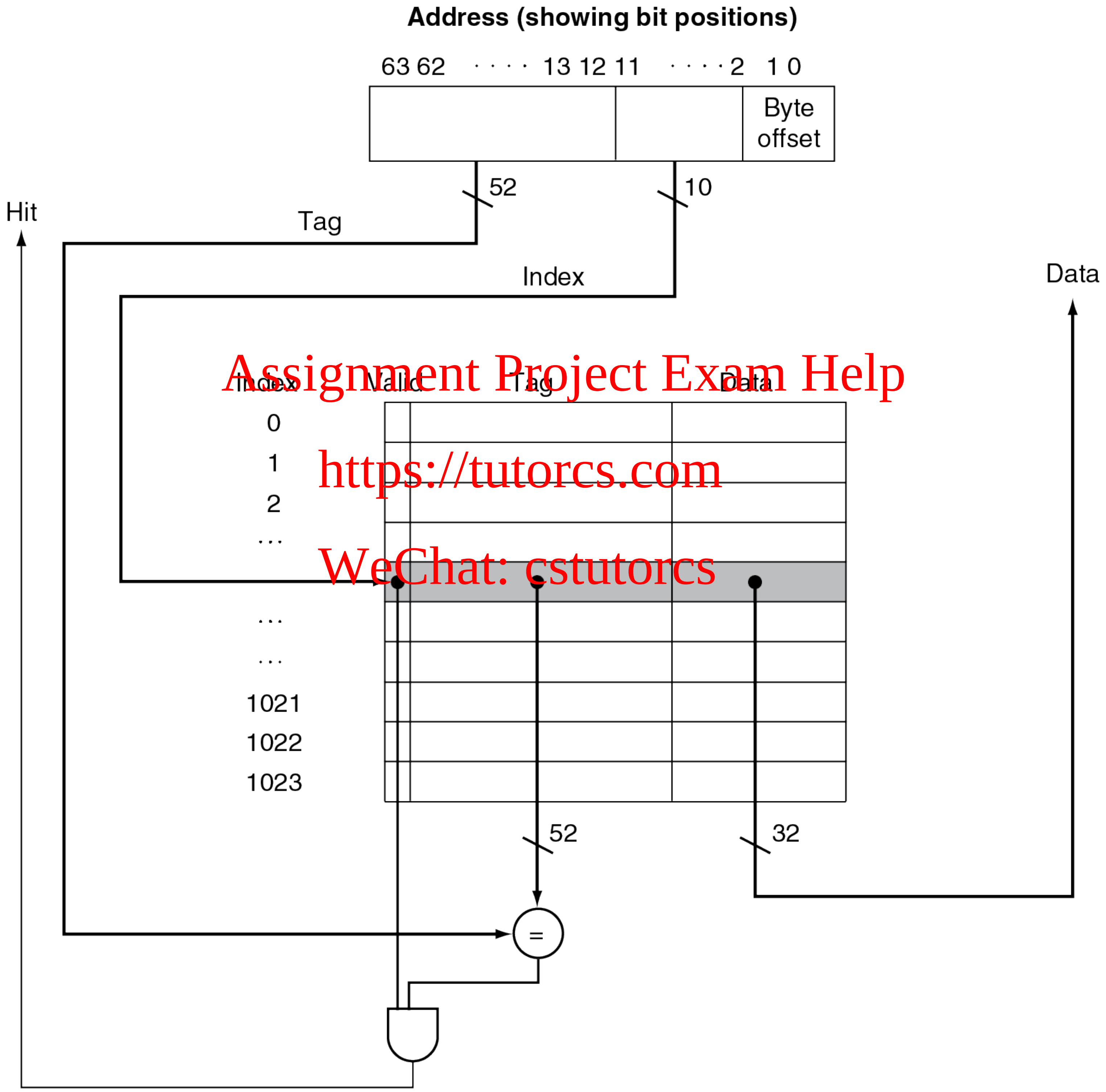
Direct Mapped Cache

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



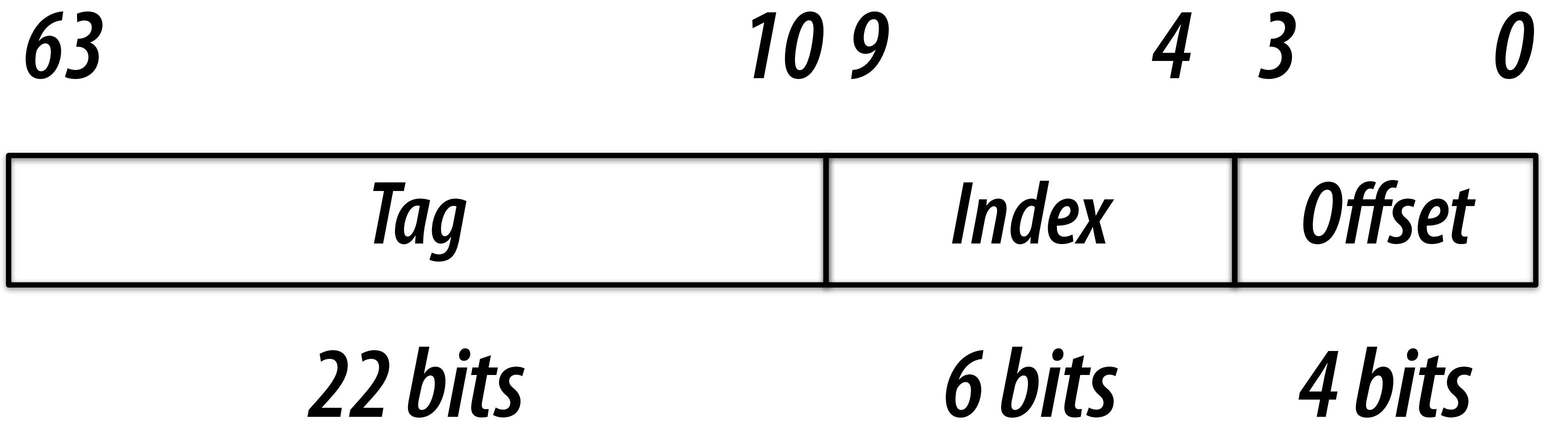
- *#Blocks is a power of 2*
- *Use low-order address bits*

Address Subdivision



Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - 16 bytes/block means 4 bits of offset ($2^4 = 16$)
  - To what block number does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$
  - I am the 75th 16-byte block in the address space
- Block number = 75 modulo 64 = 11



63 10	9 4	3 0
Tag	*Index*	*Offset*
22 bits	*6 bits*	*4 bits*

Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate
  - Larger blocks ⇒ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

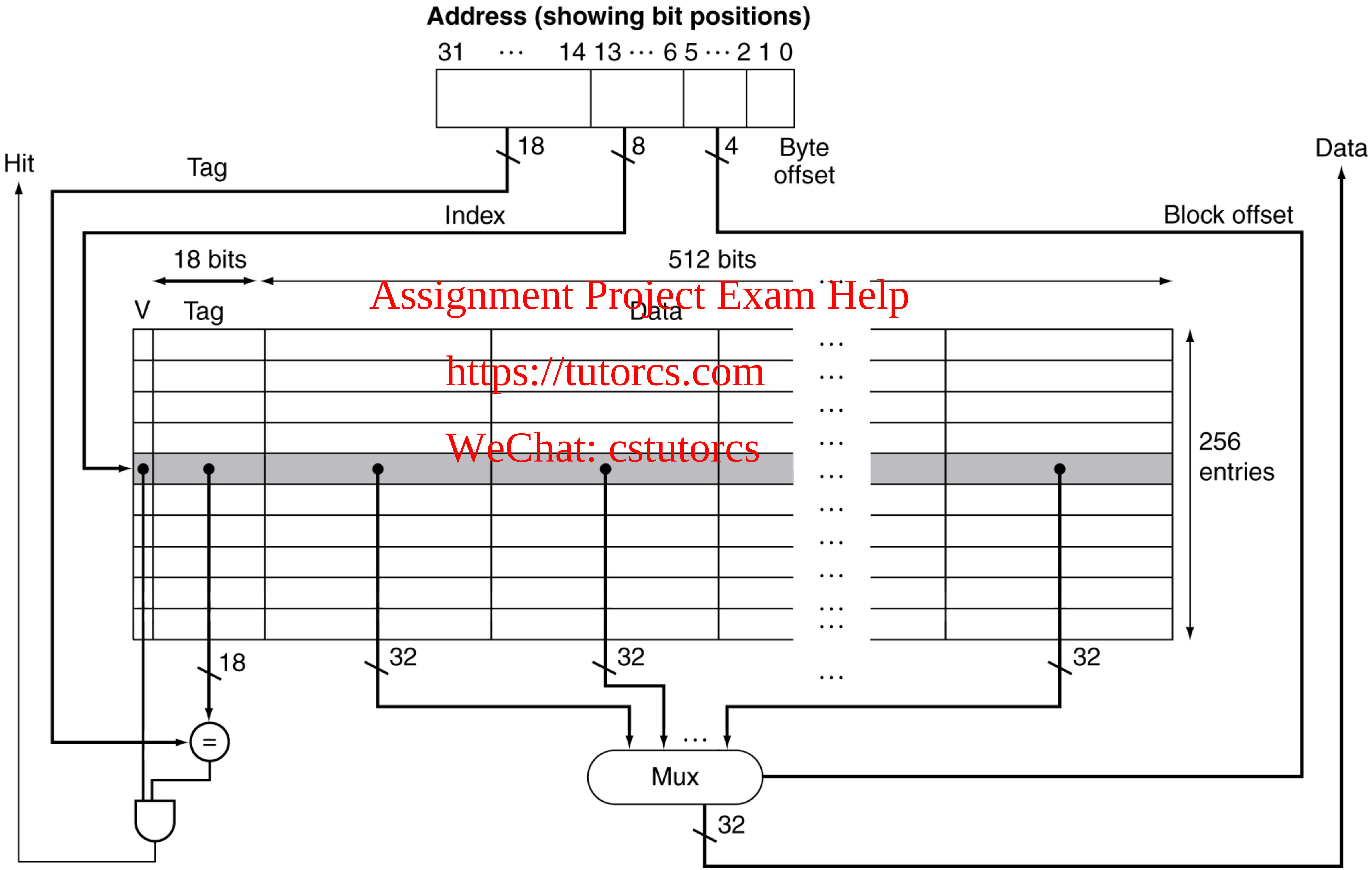
Cache Misses

- On cache hit, CPU proceeds normally
  - This is (hopefully) the common case
  - This is one or a few cycles at most
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Example: Intrinsity FastMATH

- Embedded MIPS processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back (we will talk about this later)
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

Example: Intrinsity FastMATH



Associative Caches

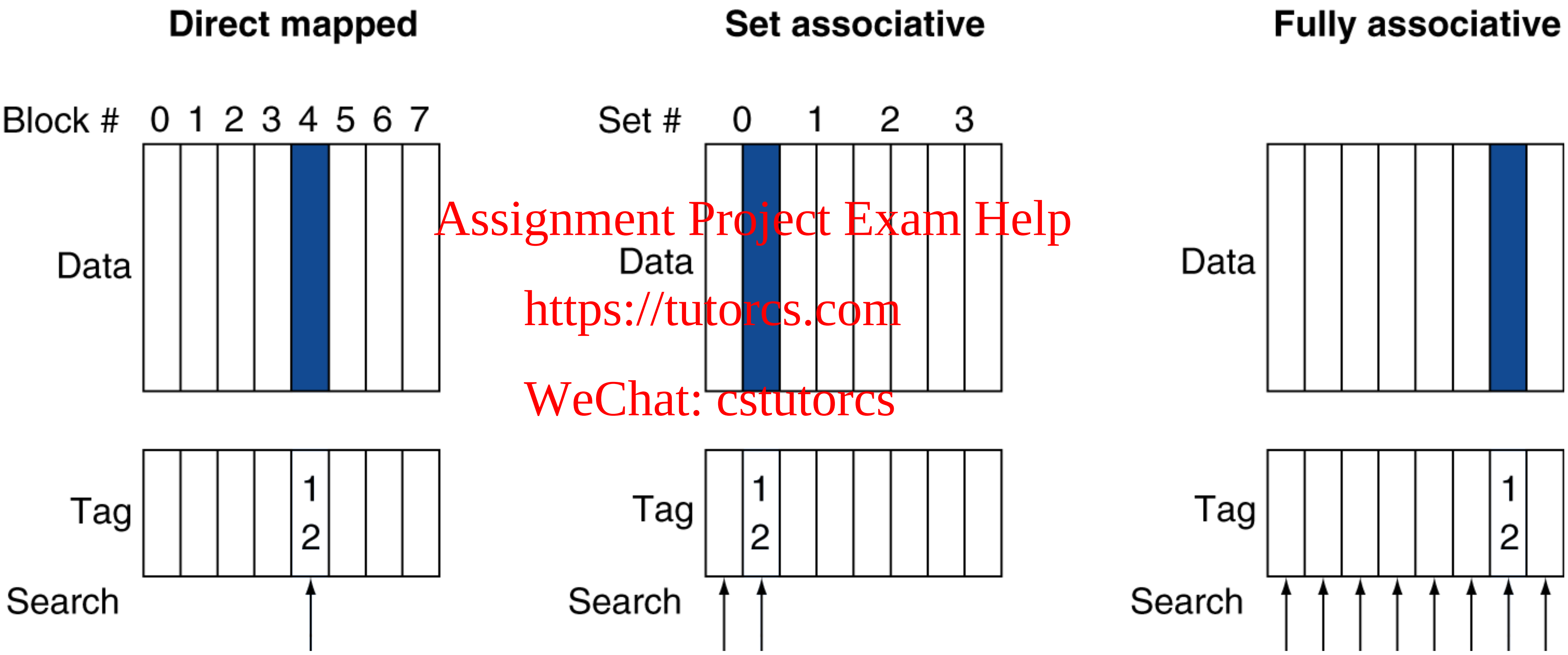
- Fully associative
    - Allow a given block to go in any cache entry
    - Requires all entries to be searched at once
    - Comparator per entry (expensive)

- *n*-way set associative

    - Each set contains *n* entries

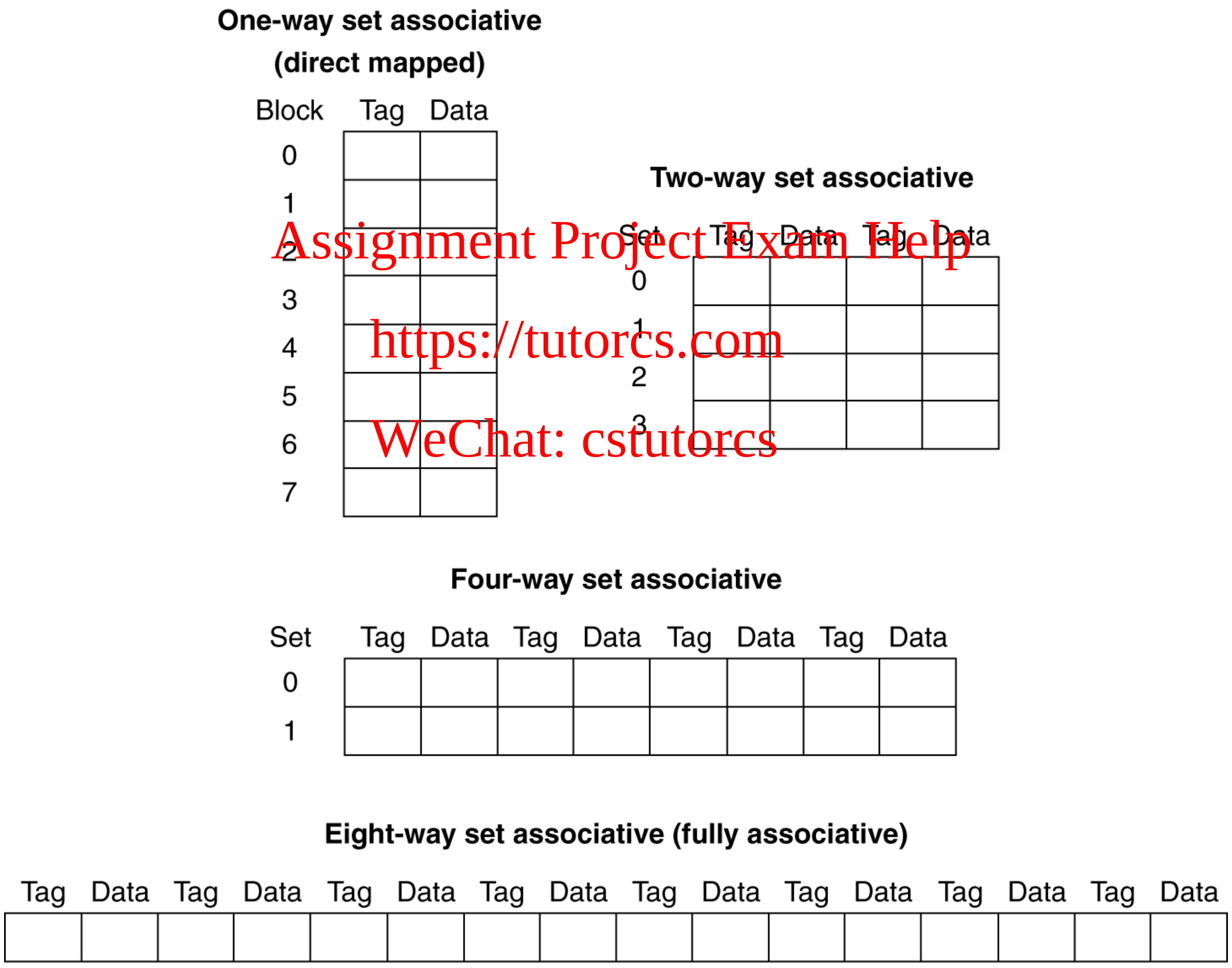
        - 1-way set associative == direct mapped
    - Block number determines which set
        - (Block number) modulo (#Sets in cache)
    - Search all entries in a given set at once
    - *n* comparators (less expensive)

Associative Cache Example



Spectrum of Associativity

- For a cache with 8 entries



One-way set associative (direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8

- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example

- Block access sequence: 0, 8, 0, 6, 8
- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	Mem[0]			
8	0	miss	Mem[0]	Mem[8]		
0	0	hit	Mem[0]	Mem[8]		
6	0	miss	Mem[0]	Mem[6]		
8	0	miss	Mem[8]	Mem[6]		

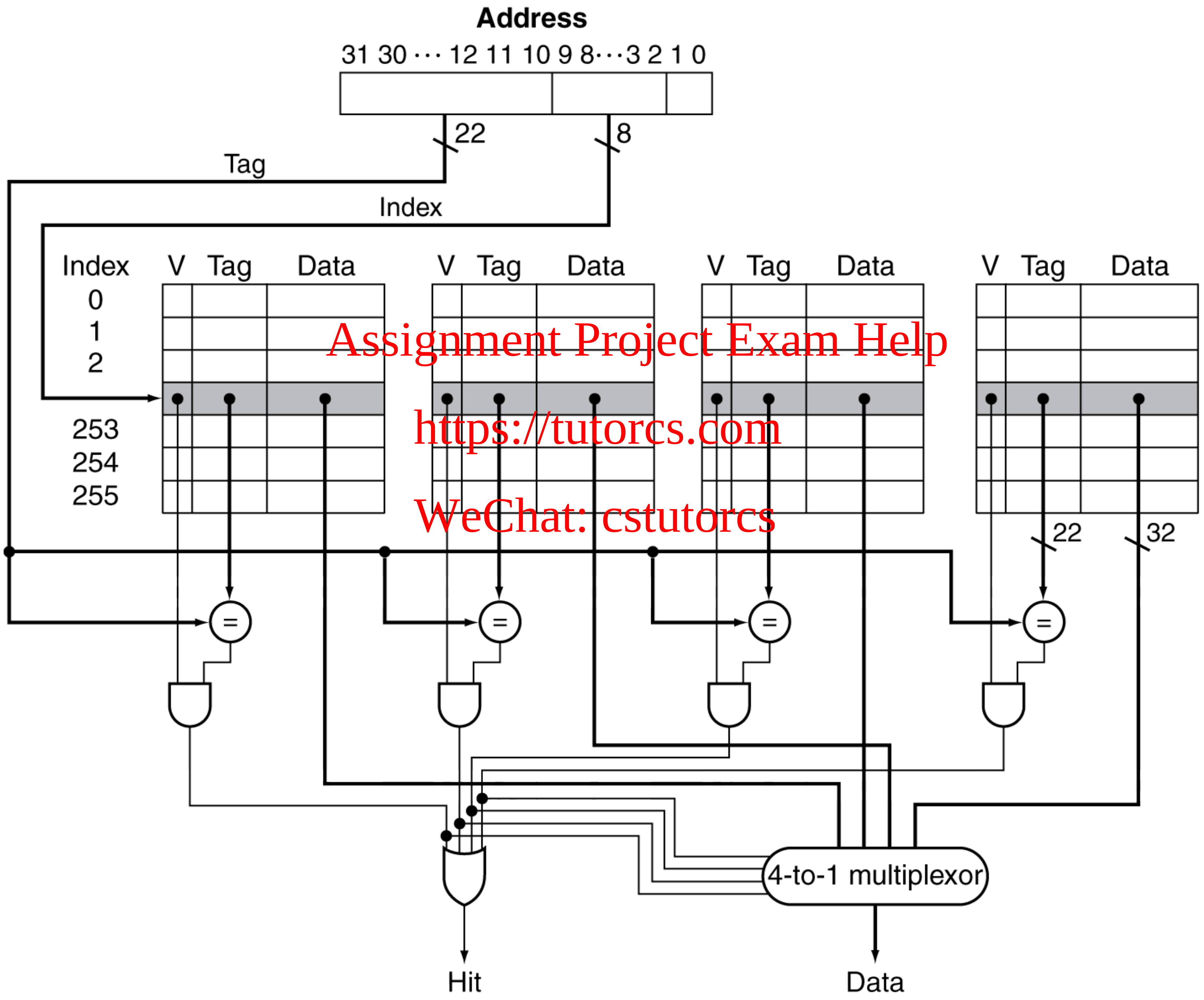
- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	Mem[0]			
8		miss	Mem[0]	Mem[8]		
0		hit	Mem[0]	Mem[8]		
6		miss	Mem[0]	Mem[8]	Mem[6]	
8		hit	Mem[0]	Mem[8]	Mem[6]	

How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64 KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Set Associative Cache Organization



Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Measuring Cache Performance

§5.4 Measuring and Improving Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses

- With simplifying assumptions:

$$\text{Memory stall cycles}$$

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI $= 2 + 2 + 1.44 = 5.44$
  - Ideal CPU is 5.44/2 =2.72 times faster

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1 ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance
  - Consequence: Increased attention to cache; more levels of cache

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L2 cache misses
- Some high-end systems include L3 cache

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100 ns
- With just primary cache
  - Miss penalty = 100 ns/0.25 ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L2 cache
  - Access time = 5 ns
  - Global miss rate to main memory = 0.5%
    - *Be careful on miss rates: miss rate on L2 alone, or global?*
- Primary miss with L2 hit
  - Penalty = 5 ns/0.25 ns = 20 cycles
- Primary miss with L2 miss
  - Extra penalty = 400 cycles
- CPI = $1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance ratio = $9/3.4 = 2.6$

Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L1 cache usually small enough to allow single-cycle access
  - L1 block size smaller than L2 block size

Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock

- Example cache block read

  - 1 bus cycle for address transfer

  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

Summary

- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon.
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon.

- By taking advantage of the principle of locality:

  - Present the user with as much memory as is available in the cheapest technology.

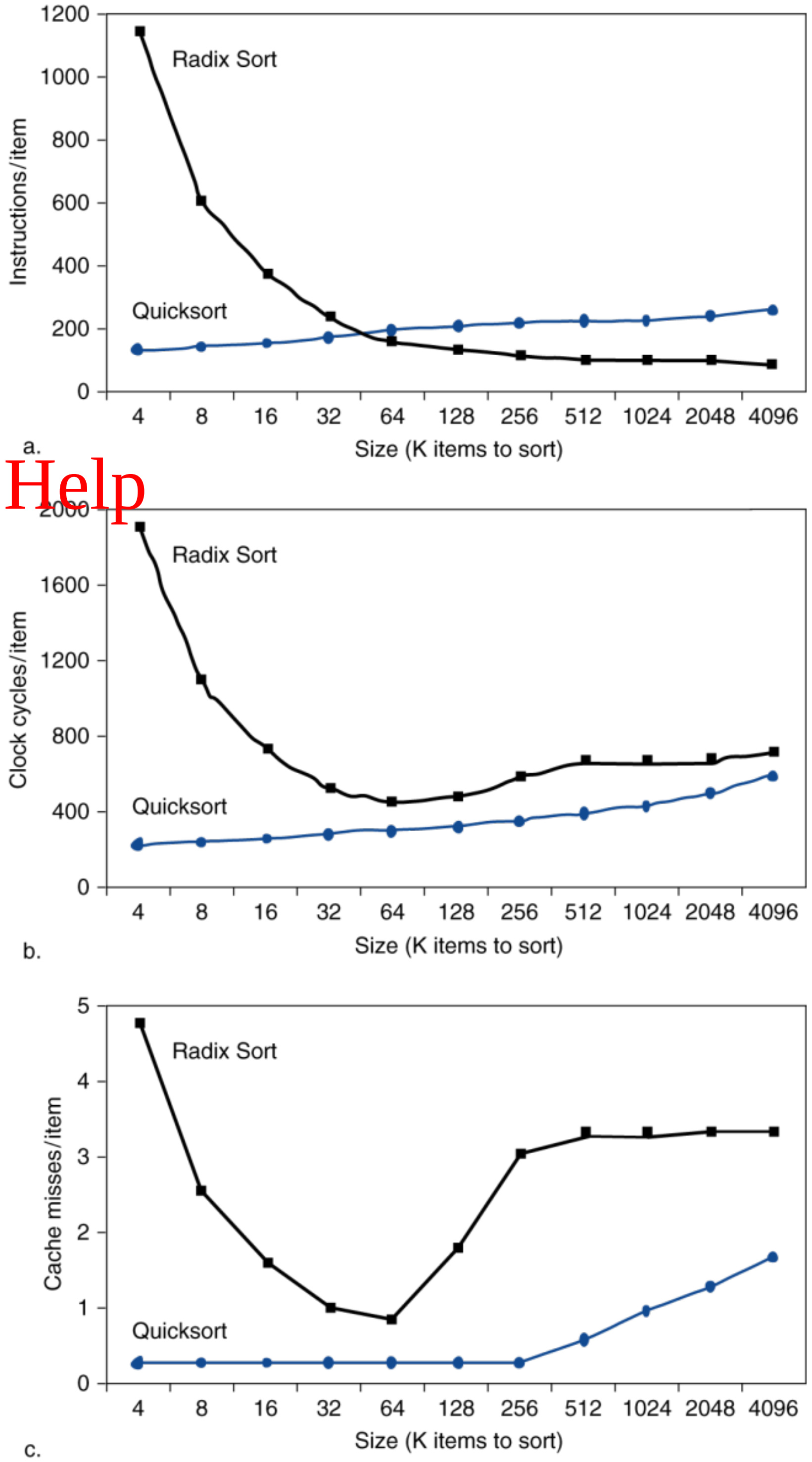
  - Provide access at the speed offered by the fastest technology.
- DRAM is slow but cheap and dense:
  - Good choice for presenting the user with a BIG memory system
- SRAM is fast but expensive and not very dense:
  - Good choice for providing the user FAST access time.

Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyze
  - Use system simulation

Interactions with Software

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access

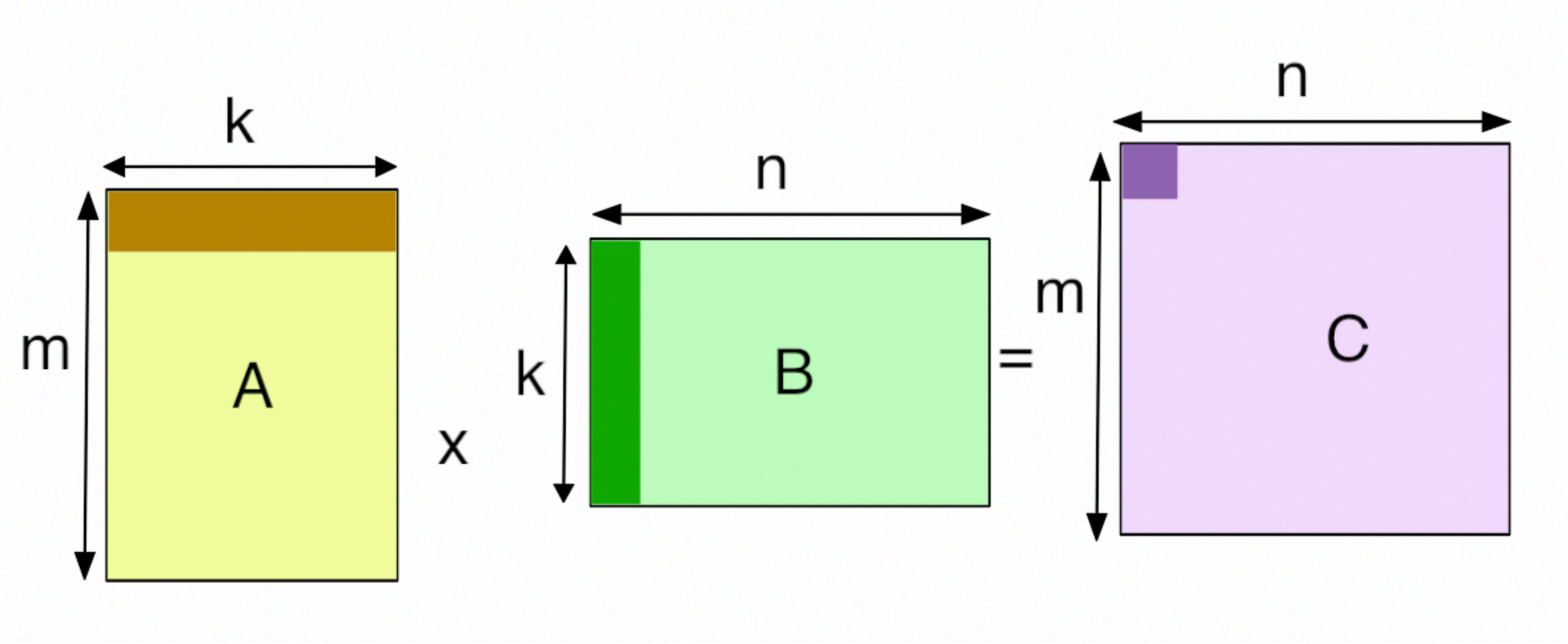


Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- Consider inner loops of DGEMM:

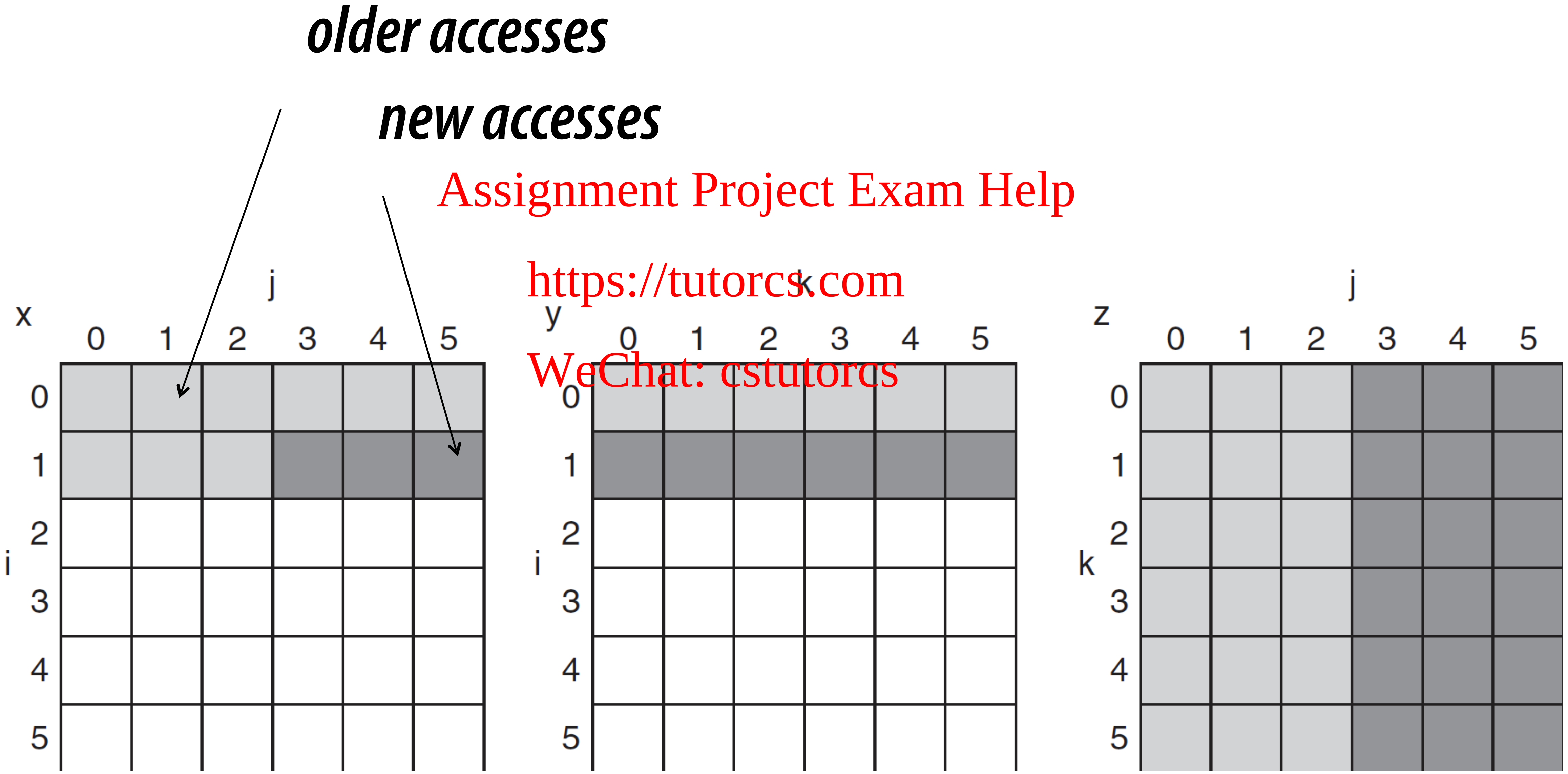
```
for (int i = 0; i < m; i++) {
  for (int j = 0; j < n; ++j) {
    double cij = C[i+j*n];
    for (int kk = 0; kk < k; kk++)
      cij += A[i*k+kk] * B[j+kk*n];
    C[j+i*n] = cij;
}
```

I did this indexing by hand and it's probably wrong



DGEMM Access Pattern

- C, A, and B arrays

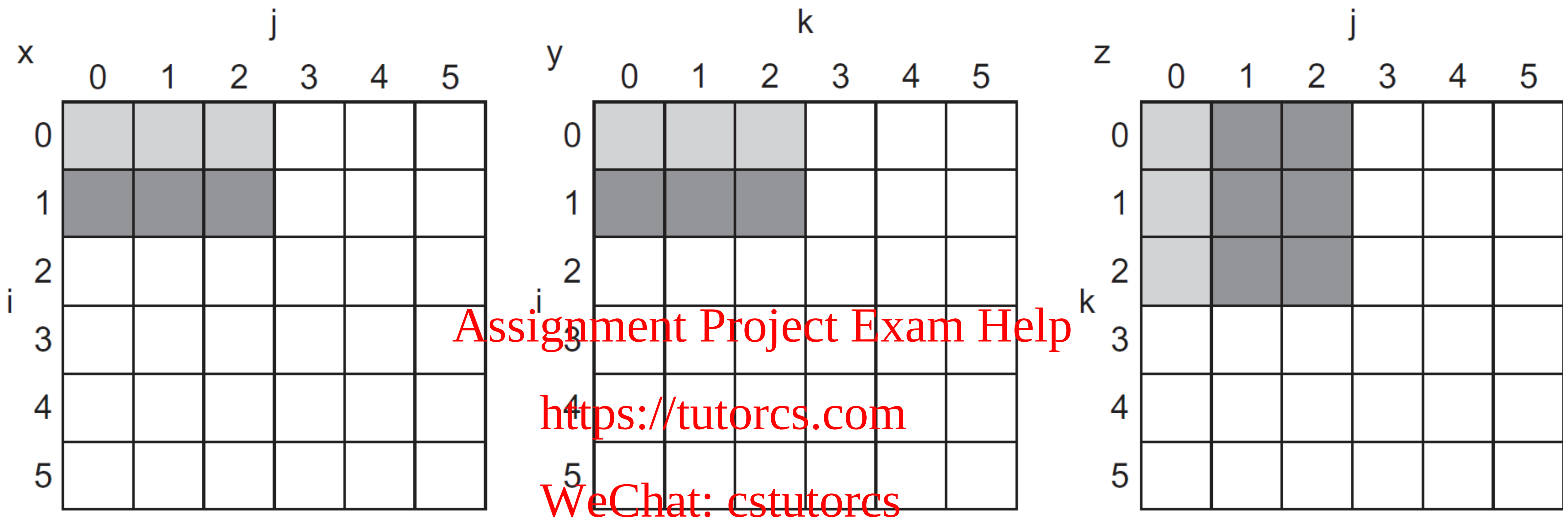


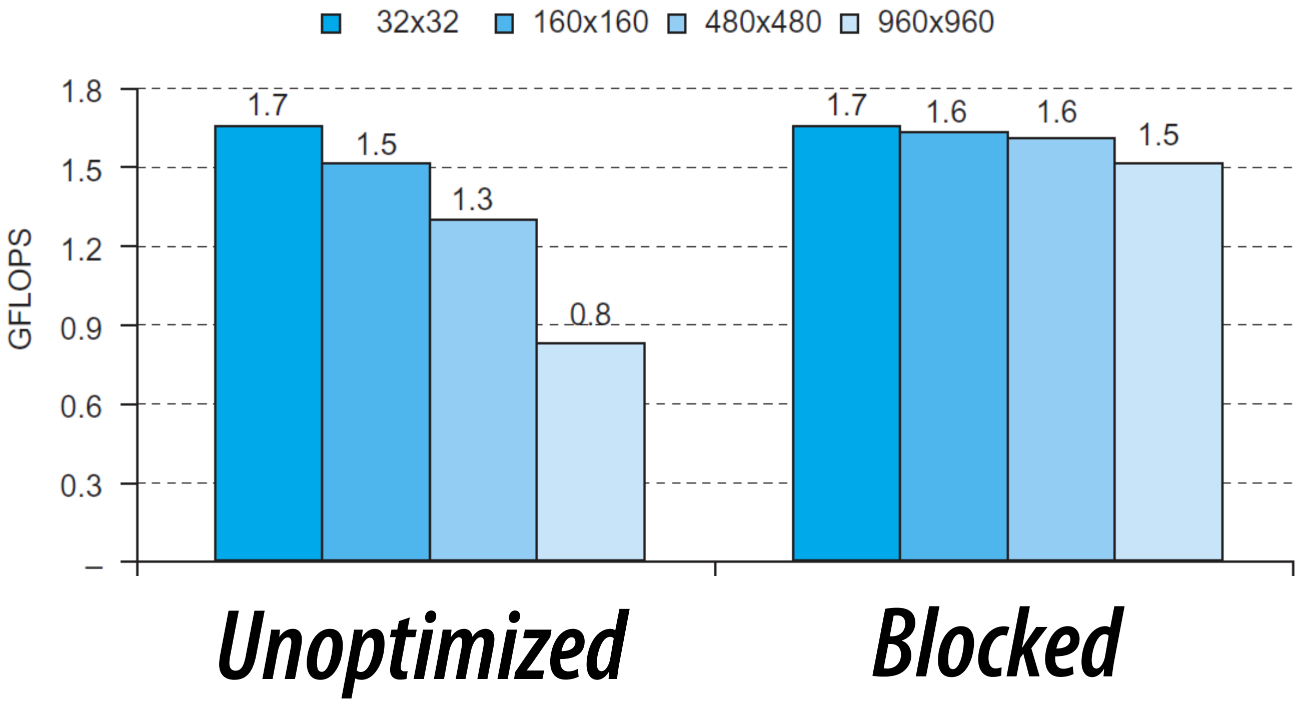
note you can transpose! might be helpful

Cache Blocked DGEMM

```
#define BLOCKSIZE 32
void do_block (int n, int si, int sj, int sk, double *A, double
*B, double *C)
{
 for (int i = si; i < si+BLOCKSIZE; ++i)
  for (int j = sj; j < sj+BLOCKSIZE; ++j)
  {
   double cij = C[i+j*n];/* cij = C[i][j] */
   for( int k = sk; k < sk+BLOCKSIZE; k++ )
    cij += A[i+k*n] * B[k+j*n];/* cij+=A[i][k]*B[k][j] */
   C[i+j*n] = cij;/* C[i][j] = cij */
  }
 }
 void dgemm (int n, double* A, double* B, double* C)
 {
  for ( int sj = 0; sj < n; sj += BLOCKSIZE )
   for ( int si = 0; si < n; si += BLOCKSIZE )
    for ( int sk = 0; sk < n; sk += BLOCKSIZE )
     do_block(n, si, sj, sk, A, B, C);
 }
```

Blocked DGEMM Access Pattern





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)

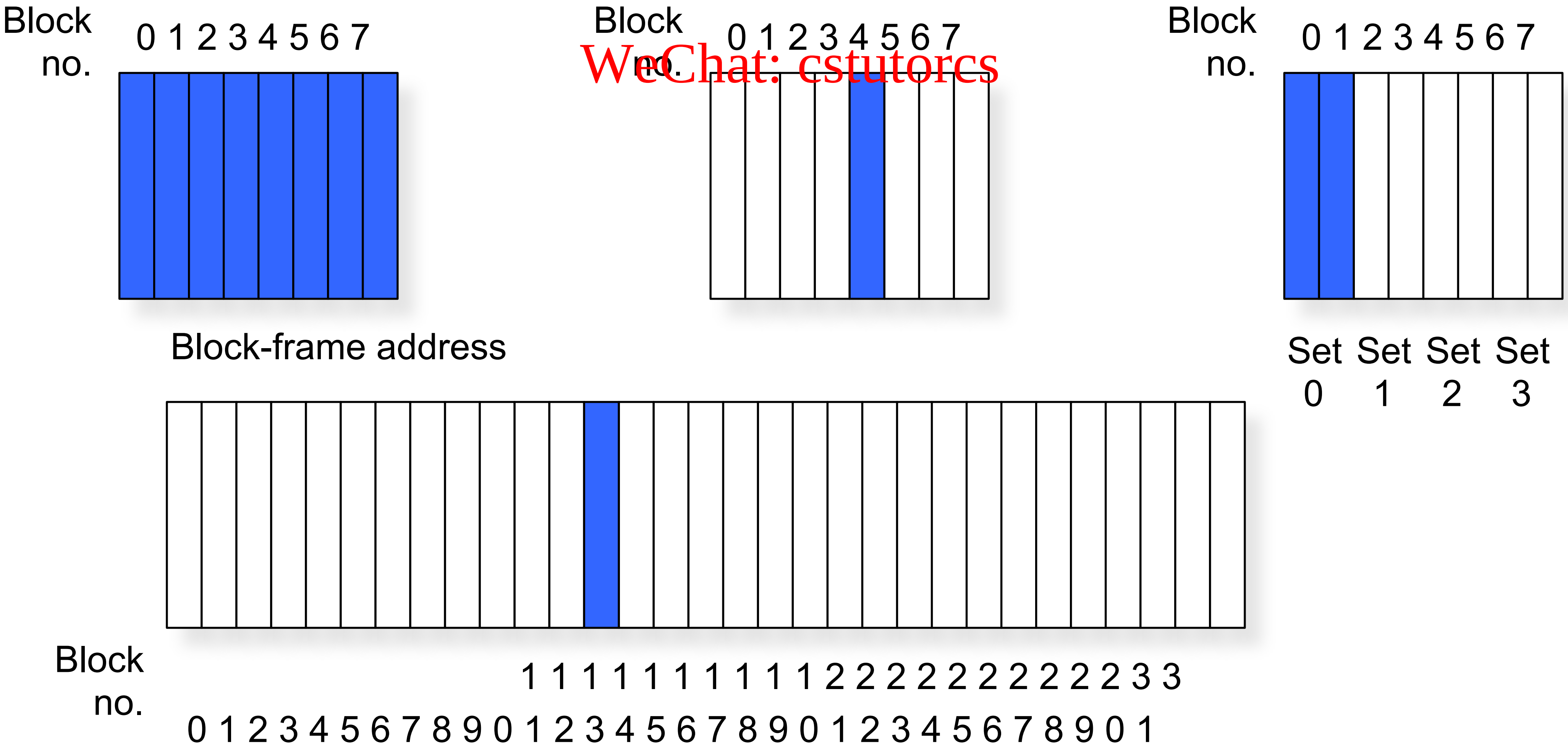
Q1: Where can a block be placed in the upper level?

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

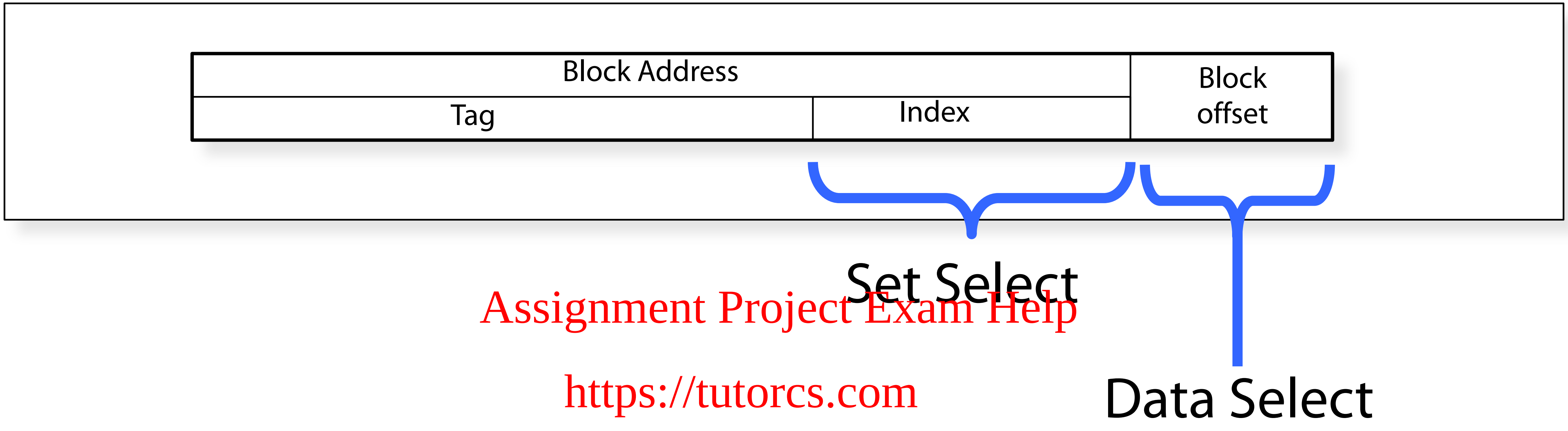
Fully associative: block 12 can go anywhere

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?

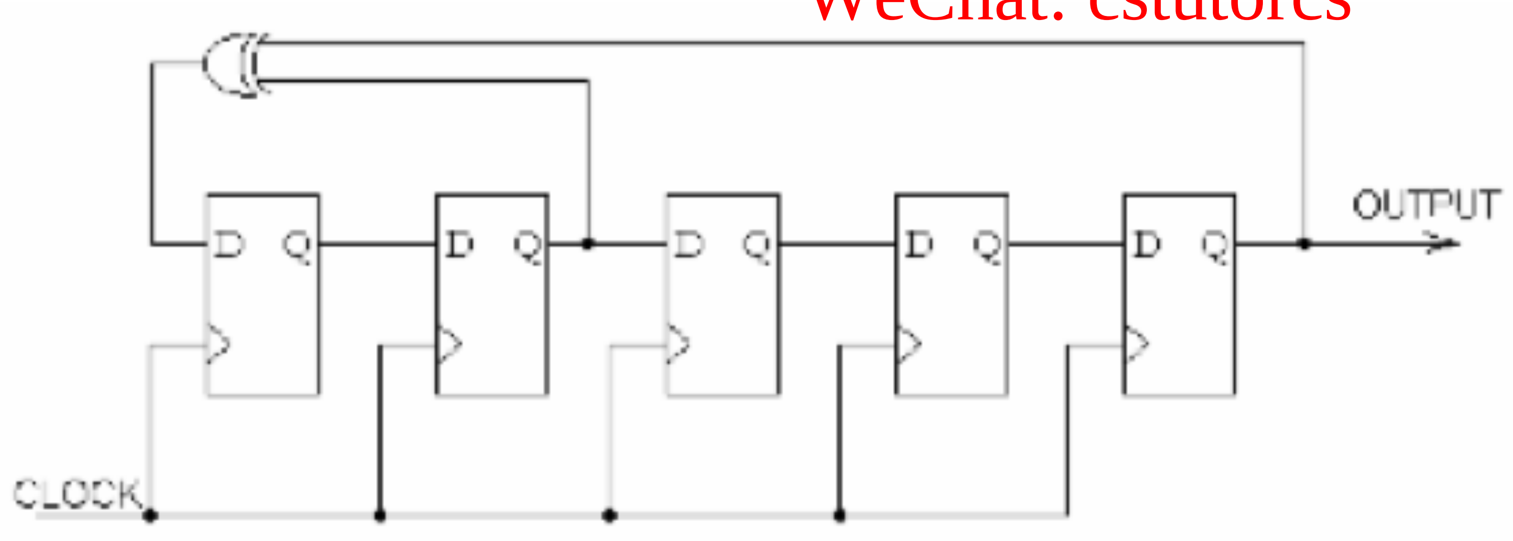
- Easy for Direct Mapped
- Set Associative or Fully Associative:
  - Random
  - LRU (Least Recently Used)

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!

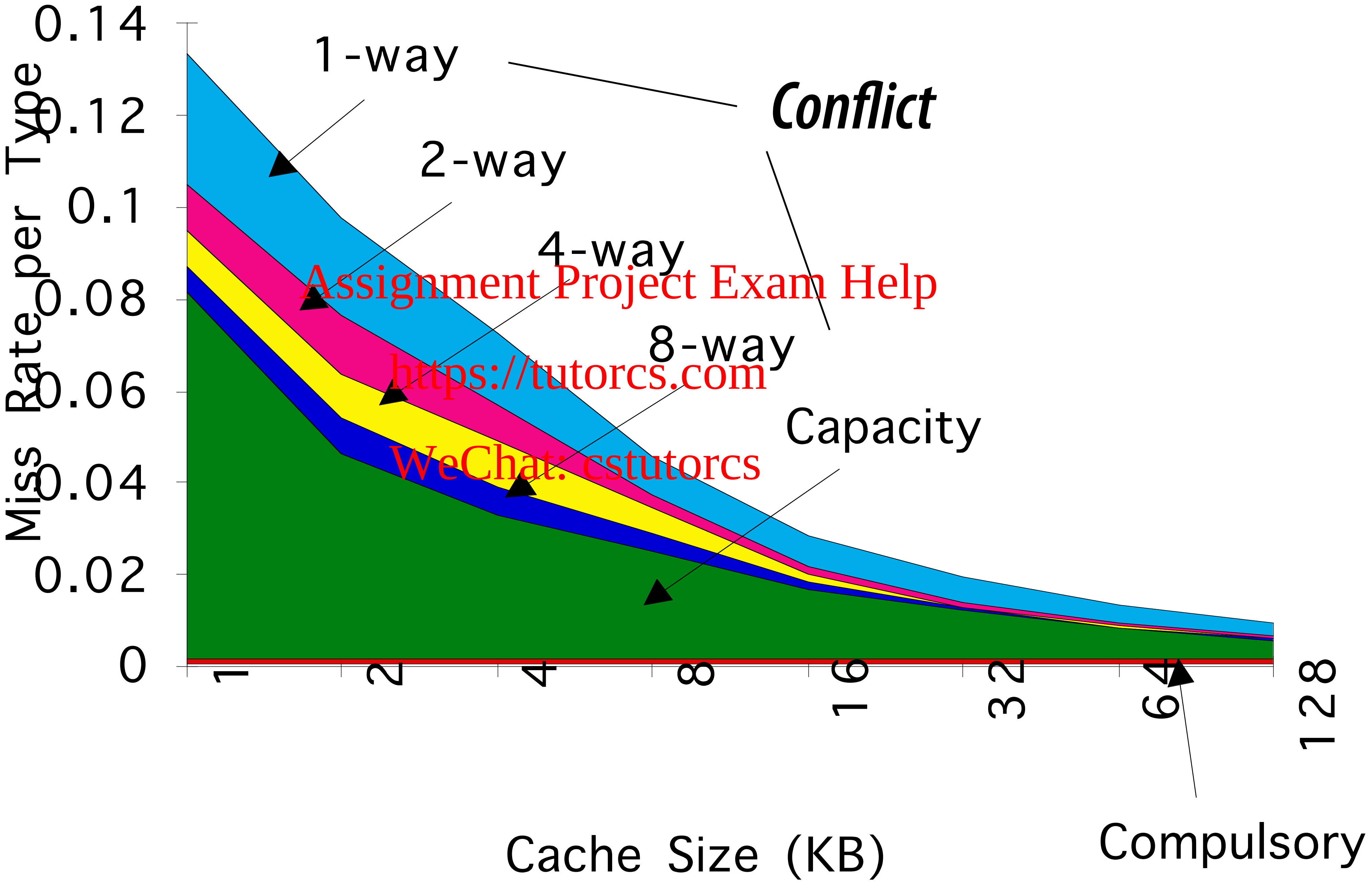


Pseudorandom Linear Feedback Shift Register

Counting Sequence

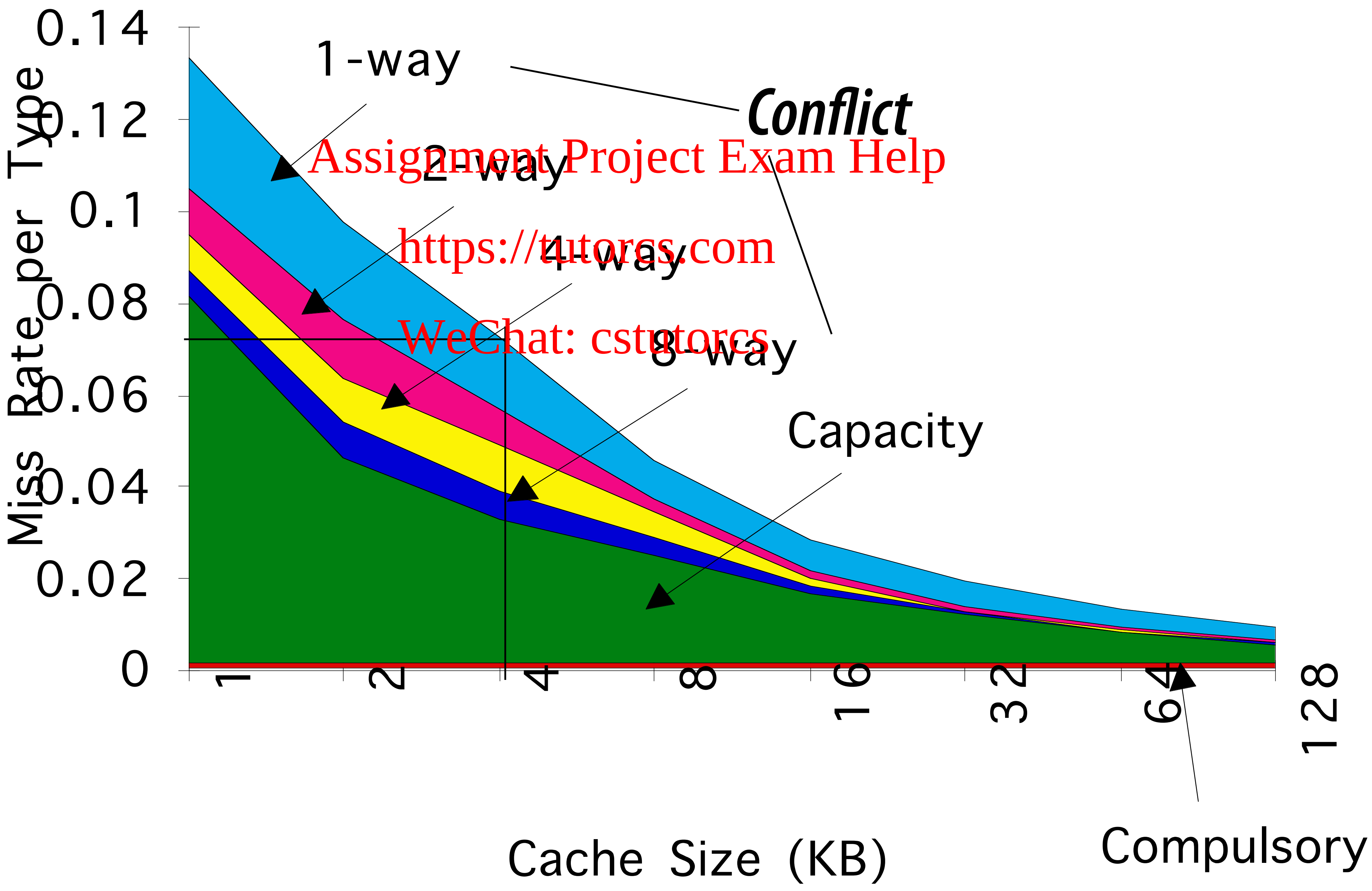
11111	0x1F
01111	0x0F
00111	0x07
10011	0x13
11001	0x19
01100	0x0C
10110	0x16
01011	0x0B
00101	0x05
10010	0x12
01001	0x09
00100	0x04
00010	0x02
00001	0x01
10000	0x10
01000	0x08
10100	0x14
01010	0x0A
10101	0x15
11010	0x1A
11101	0x1D
01110	0x0E
10111	0x17
11011	0x1B
01101	0x0D
00110	0x06
00011	0x03
10001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

3Cs Absolute Miss Rate (SPEC92)

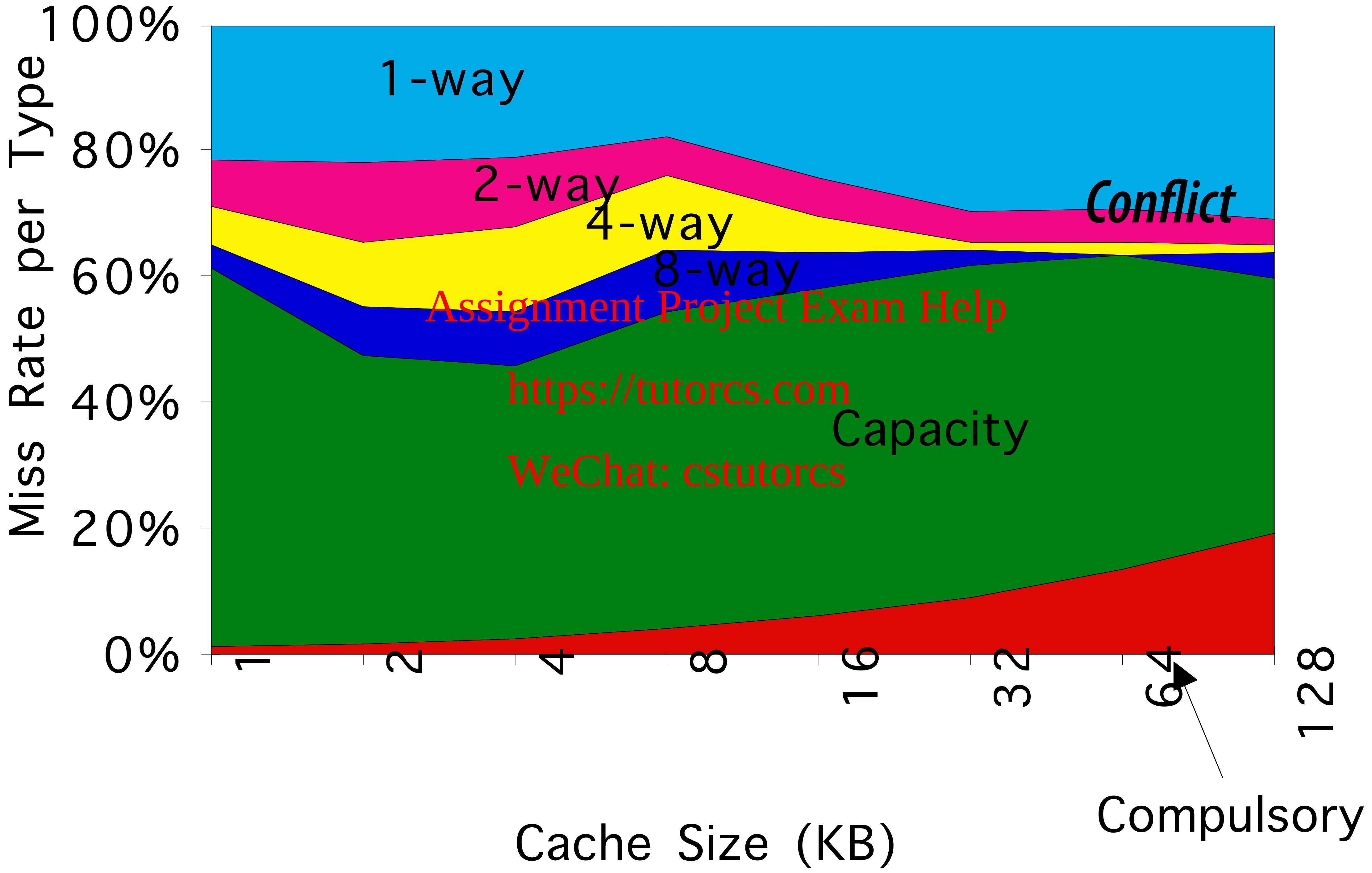


2:1 Cache Rule

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



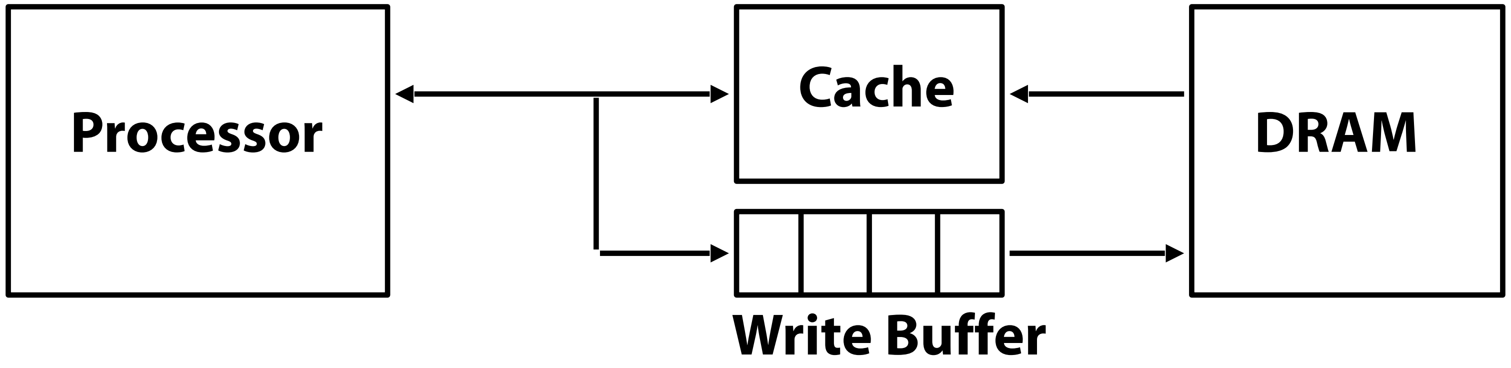
3Cs Relative Miss Rate



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*.

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.
  - is block clean or dirty?

- 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 lower level memory