# CS510 Computer Architecture

# Lecture 12: Review: Cache Memory

**Soontae Kim**

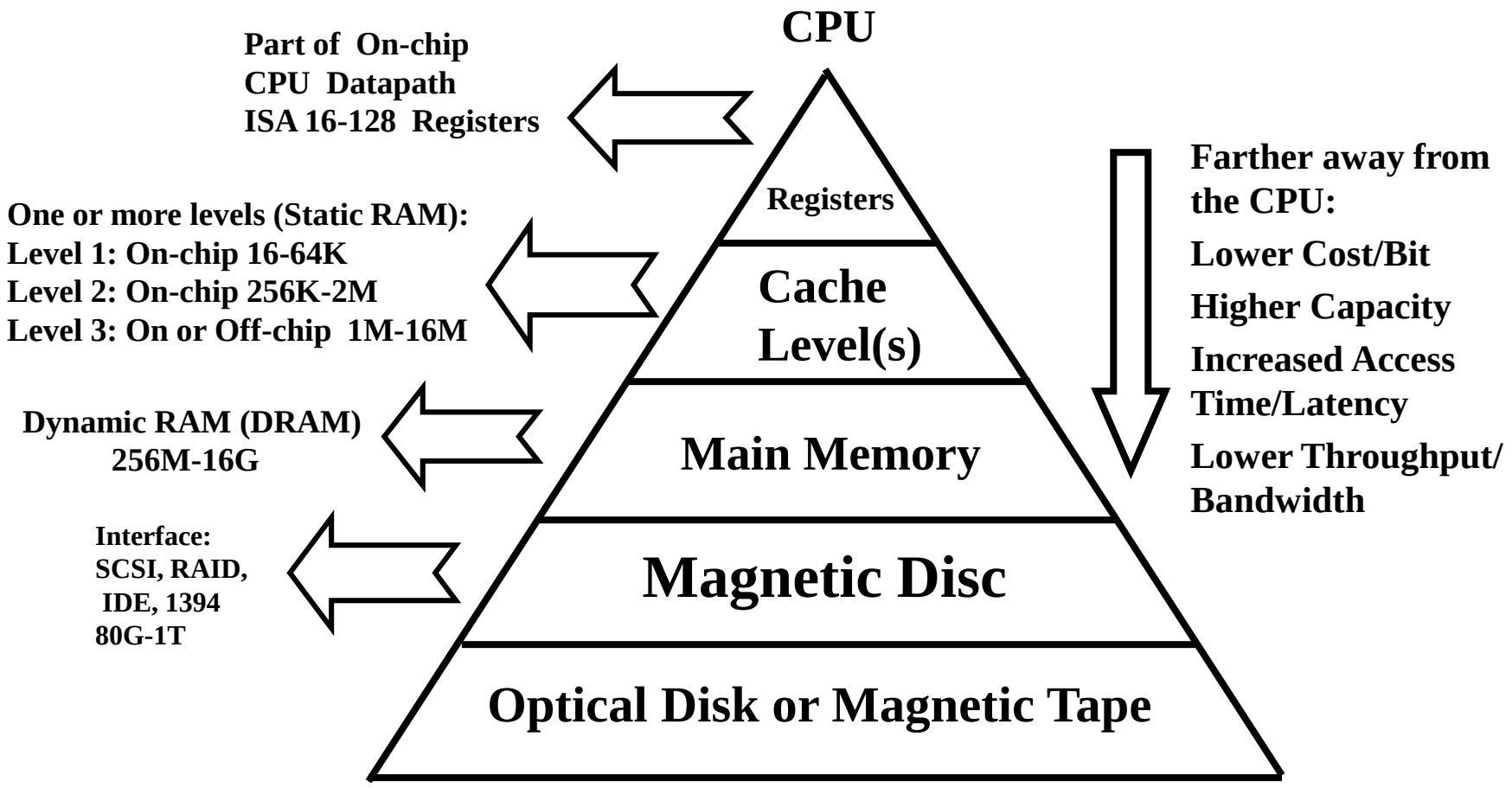
**Spring 2017**

**School of Computing, KAIST**

# Notice

- **Homework assignment#2**
  - Due on April 19 (Wednesday)
- **Midterm exam**
  - Officially scheduled on April 17 but move to 1:00PM on April 19 (Wed.)
- **Term project proposal**
  - April 28 (Friday)
  - Prepare less than 10 slides in 5 minutes.
  - All team members must prepare parts of presentation

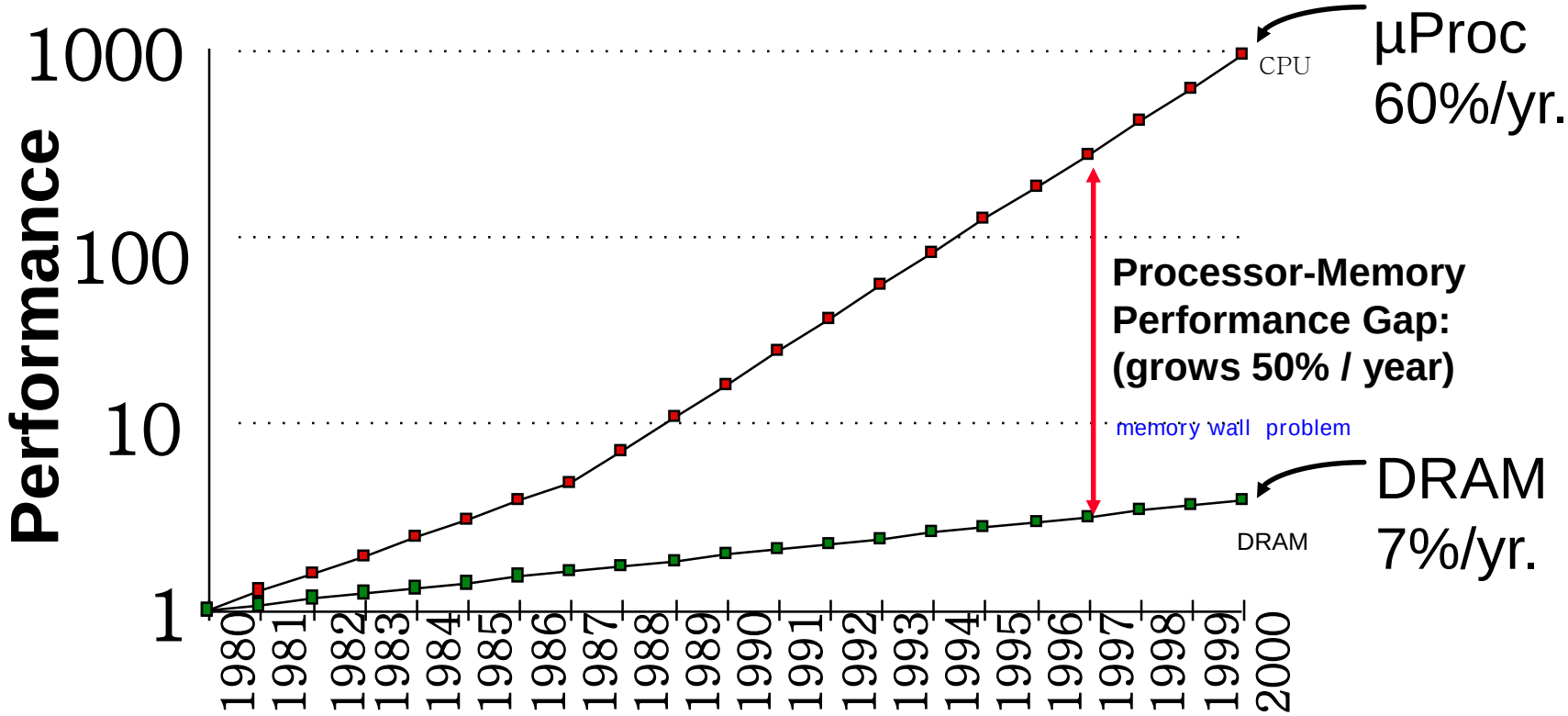
# Levels of The Memory Hierarchy



# Memory Hierarchy: Motivation

- **The gap between CPU performance and main memory has been widening with higher performance CPUs creating performance bottlenecks for memory access instructions.**
- **The memory hierarchy is organized into several levels of memory with the smaller, faster memory levels closer to the CPU: registers, then primary Cache Level ($L_1$), then additional secondary cache levels ($L_2$, $L_3$…), then main memory, then mass storage (virtual memory).**
- **Each level of the hierarchy is usually a subset of the level below: data found in a level is also found in the level below (farther from CPU) but at lower speed (longer access time).**
- **Each level maps addresses from a larger physical memory to a smaller level of physical memory closer to the CPU.**
- **This concept is greatly aided by the principal of locality both temporal and spatial which indicates that programs tend to reuse data and instructions that they have used recently or to use those stored nearby.**

# Memory Hierarchy: Motivation
# Processor-Memory (DRAM) Performance Gap



# Processor-DRAM Performance Gap Impact

- To illustrate the performance impact, assume a single-issue pipelined CPU with CPI = 1 using non-ideal memory.
- Ignoring other factors, the minimum cost of a full memory access in terms of number of wasted CPU cycles:

| Year | CPU speed MHZ | CPU cycle ns | Memory Access ns | Minimum CPU memory stall cycles |
|---|---|---|---|---|
| | | | | 1 cycle = ideal cycle |
| 1986: | 8 | 125 | 190 | 190/125 - 1 = 0.5 |
| 1989: | 33 | 30 | 165 | 165/30 -1 = 4.5 |
| 1992: | 60 | 16.6 | 120 | 120/16.6 -1 = 6.2 |
| 1996: | 200 | 5 | 110 | 110/5 -1 = 21 |
| 1998: | 300 | 3.33 | 100 | 100/3.33 -1 = 29 |
| 2000: | 1000 | 1 | 90 | 90/1 - 1 = 89 |
| 2002: | 2000 | .5 | 80 | 80/.5 - 1 = 159 |
| 2004: | 3000 | .333 | 60 | 60/.333 - 1 = 179 |

average memory stall cycle more than 300

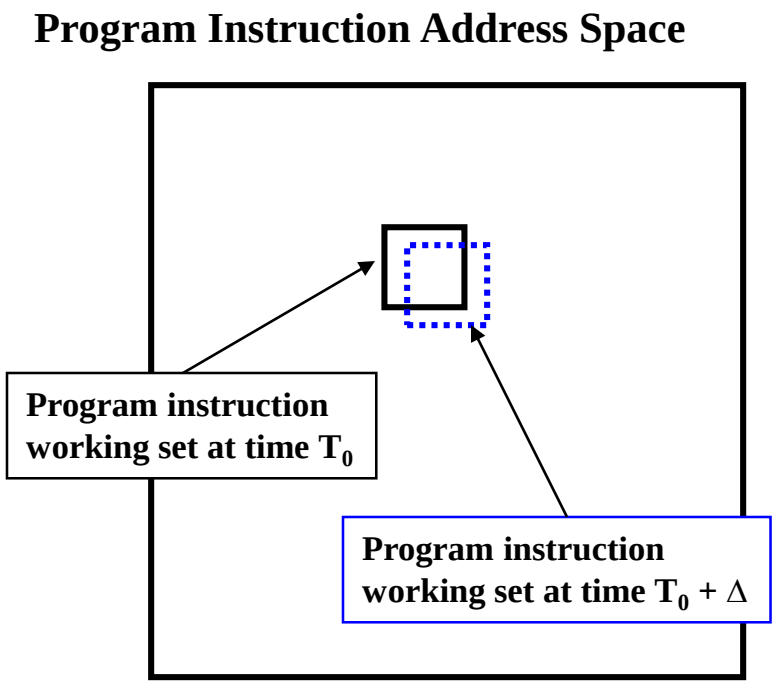
# Memory Hierarchy: Motivation

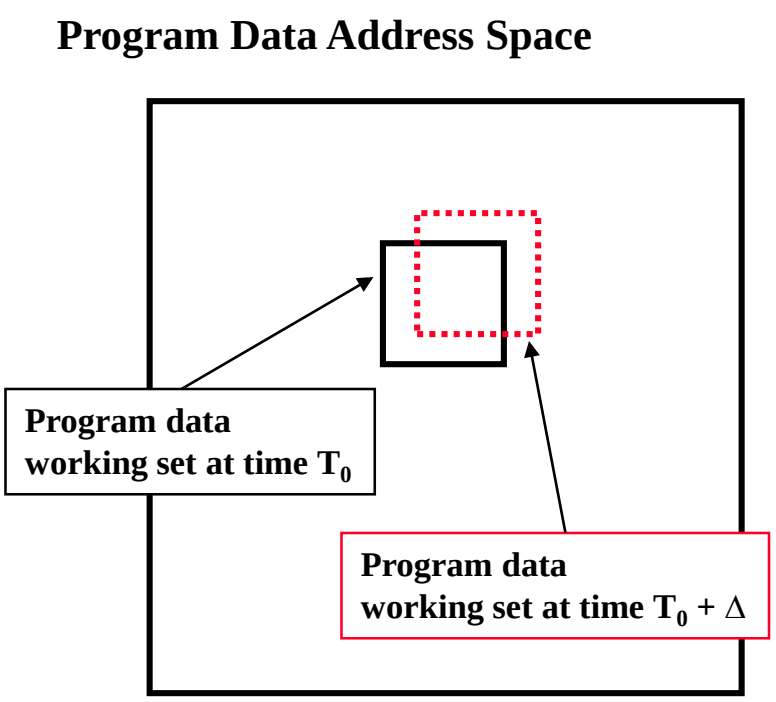
## The Principle of Locality

- **Programs usually access a relatively small portion of their address space (instructions/data) at any instant of time (program working set).**
- **Two Types of access locality:**
  - ***Temporal Locality:*** **If an item (instruction or data) is referenced, it will tend to be referenced again soon.**
    - **e.g. instructions in the body of inner loops**
  - ***Spatial locality:*** **If an item is referenced, items whose addresses are close will tend to be referenced soon.**
    - **e.g. sequential instruction execution, sequential access to elements of array**

# Access Locality & Program Working Set

- **The presence of locality in program behavior or memory access patterns, makes it possible to satisfy a large percentage of memory accesses using faster memory levels with *much smaller capacity* than program address space.**



# Memory Hierarchy Operation

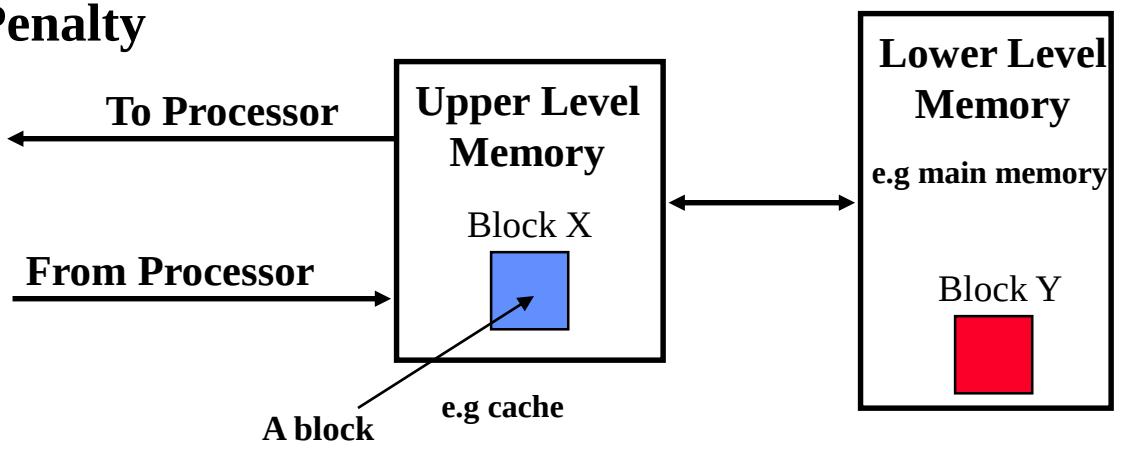
- **If an instruction or operand is required by the CPU, the levels of the memory hierarchy are searched for the item starting with the level closest to the CPU (Level 1 cache):**
  - **If the item is found, it's delivered to the CPU resulting in a cache hit without searching lower levels.**
  - **If the item is missing in an upper level, resulting in a cache miss, the level just below is searched.**
  - **For systems with several levels of cache, the search continues with cache level 2, 3 etc.**
  - **If all levels of cache report a miss then main memory is accessed for the item.**
    - **CPU ↔ cache ↔ memory: Managed by hardware.**
  - **If the item is not found in main memory resulting in a page fault, then disk (virtual memory) is accessed for the item.**
    - **Memory ↔ disk: Managed by the operating system with hardware support**

# Memory Hierarchy: Terminology

- **A Block:** The smallest unit of information transferred between two levels.
- **Hit:** Item is found in some block in the upper level (example: Block X)
  - **Hit Rate:** The fraction of memory accesses found in the upper level.
  - **Hit Time:** Time to access the upper level which consists of

    RAM access time + Time to determine hit/miss
- **Miss:** Item needs to be retrieved from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level +

    M Time to deliver the missed block to the processor
- **Hit Time << Miss Penalty**



# Basic Cache Concepts

- **Cache is the first level of the memory hierarchy searched first for the requested data.**
- **If the data requested by the CPU is present in the cache, it is retrieved from the cache and the data access is a cache hit. Otherwise, a cache miss and data must be read from main memory.**
- **On a cache miss, a block of data must be brought in from main memory to cache to possibly *replace* an existing cache block.**
- **The allowed cache block addresses where data blocks can be mapped (placed) into from main memory is determined by *cache placement strategy*.**
- **Locating a block of data in cache is handled by cache block identification mechanism: Tag matching.**
- **On a cache miss choosing the cache block being removed (replaced) is handled by the *block replacement strategy*.**
- **When a write to cache is requested, a number of main memory update strategies exist as part of *the cache write policy*.**

# Basic Cache Design & Operation Issues

- **Q1: Where can a block be placed in cache?**
  ***(Block placement strategy & Cache organization)***
  - **Fully Associative, Set Associative, Direct Mapped.**
- **Q2: How is a block found if it is in cache?**
  ***(Block identification)***
  - **Tag/Block.**
- **Q3: Which block should be replaced on a miss?**
  ***(Block replacement)***
  - **Random, LRU, FIFO.**
- **Q4: What occurs on a write?**
  ***(Cache write policy)***
  - **Write through, write back.**

# Cache Organization & Placement Strategies

**Placement strategies or mapping of a main memory data block onto cache block frames divide cache designs into three organizations:**

1. **Direct mapped cache:** **A block can be placed in only one location (cache block frame), given by the mapping function:**

   Mapping Function

   **index = (Block address) MOD (Number of blocks in cache)**

2. **Fully associative cache:** **A block can be placed anywhere in cache. (no mapping function).**

3. **Set associative cache:** **A block can be placed in a restricted set of places, or cache block frames. A set is a group of block frames in the cache. A block is first mapped onto the set and then it can be placed anywhere within the set. The set in this case is chosen by:**

   Mapping Function

   **index = (Block address) MOD (Number of sets in cache)**

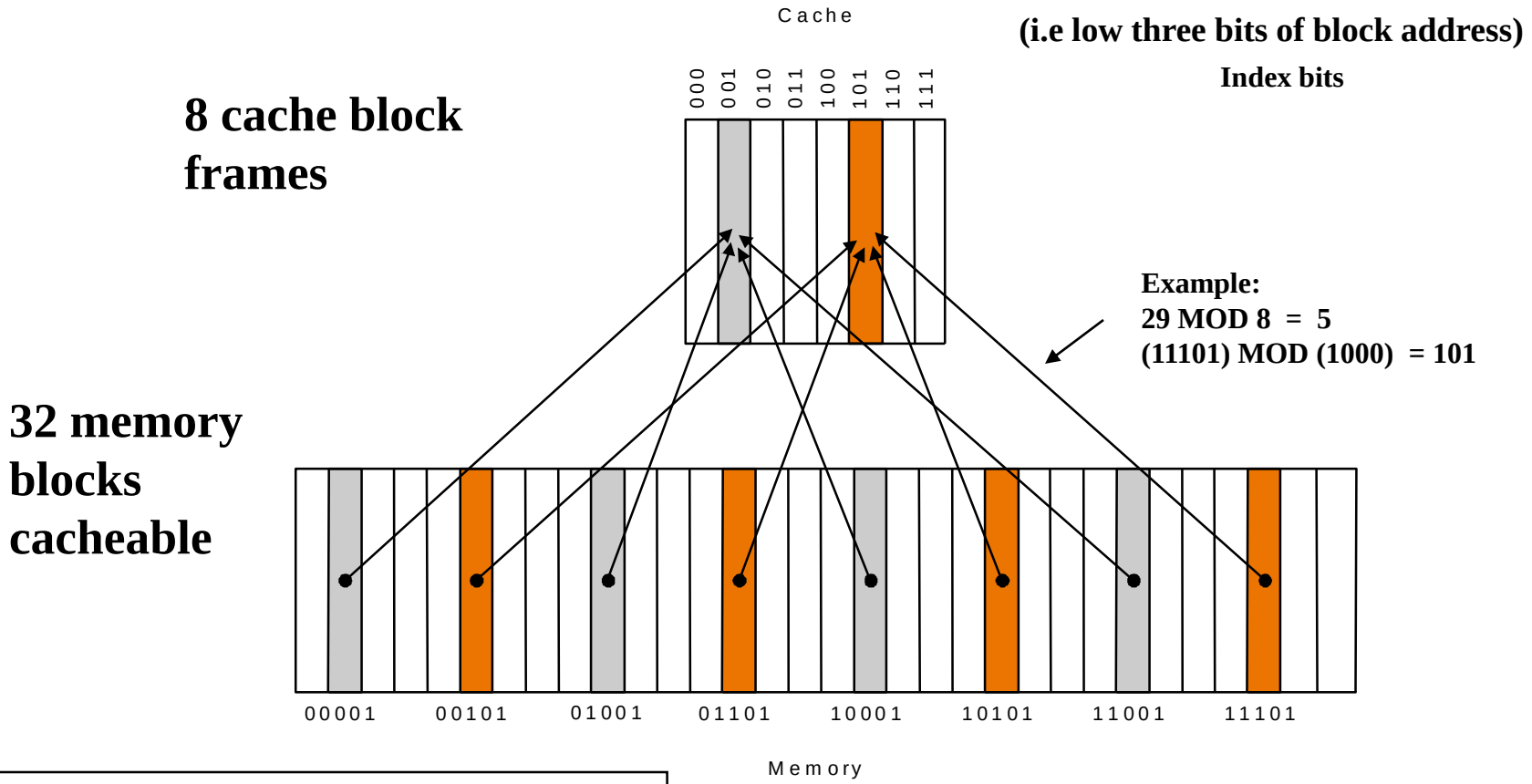
   **If there are *n* blocks in a set the cache placement is called *n*-way set-associative.**

# Cache Organization:
# Direct Mapped Cache

**A block can be placed in one location only, given by**:

(Block address) MOD (Number of blocks in cache)

**In this case, mapping function:** (Block address) MOD (8)




**Limitation of Direct Mapped Cache:** Conflicts between memory blocks that map to the same cache block frame

# 4KB Direct Mapped Cache Example

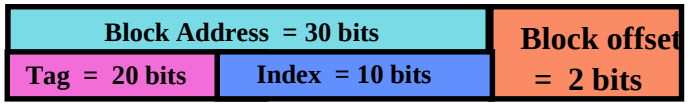
**1K = 1024 Blocks**
**Each block = one word**

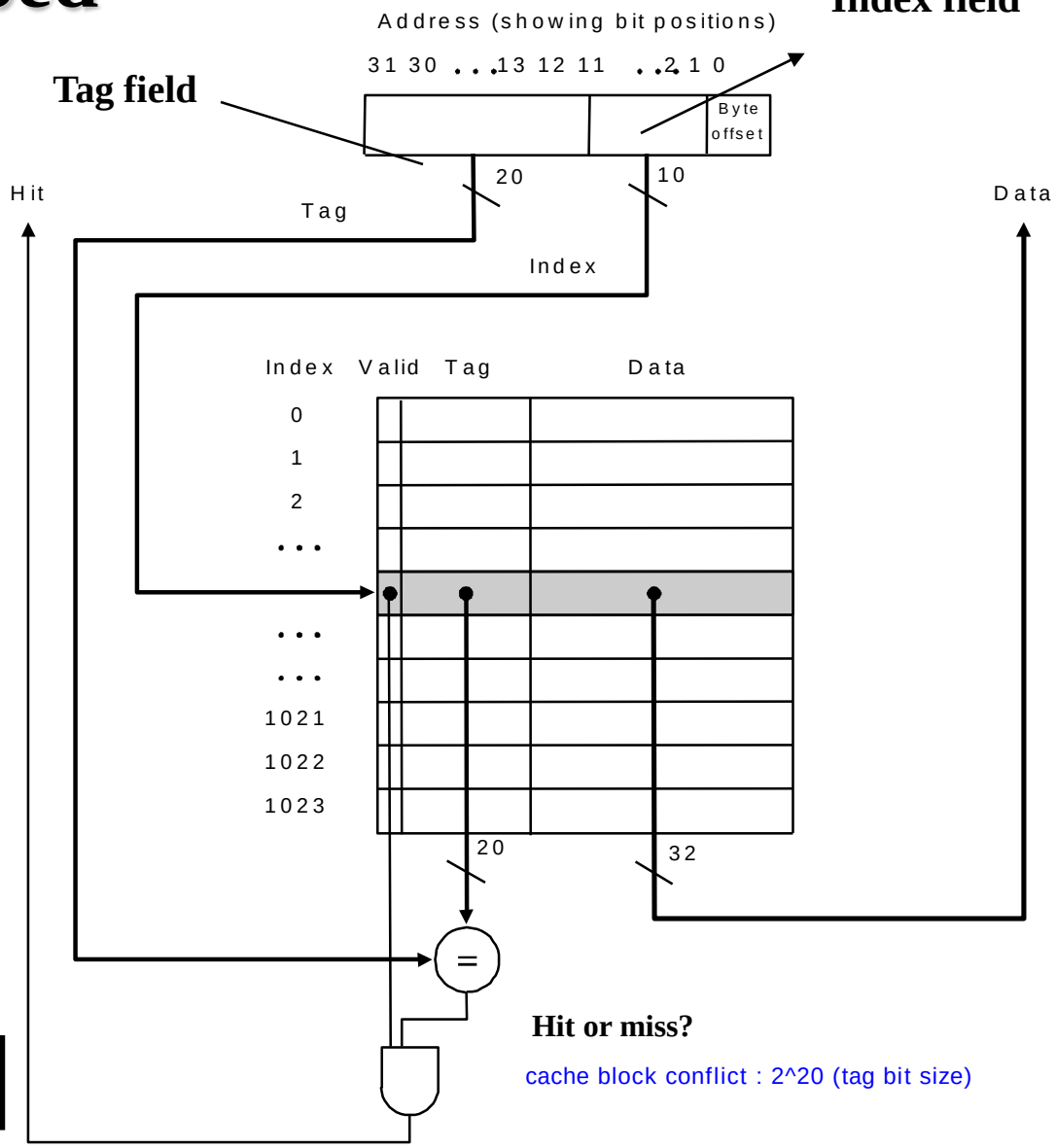
**Can cache up to**
**$2^{32}$ bytes = 4 GB**
**of memory**

**Mapping function:**

**Cache Block frame number =**
**(Block address) MOD (1024)**

**i.e. index field or**
**10 low bit of block address**

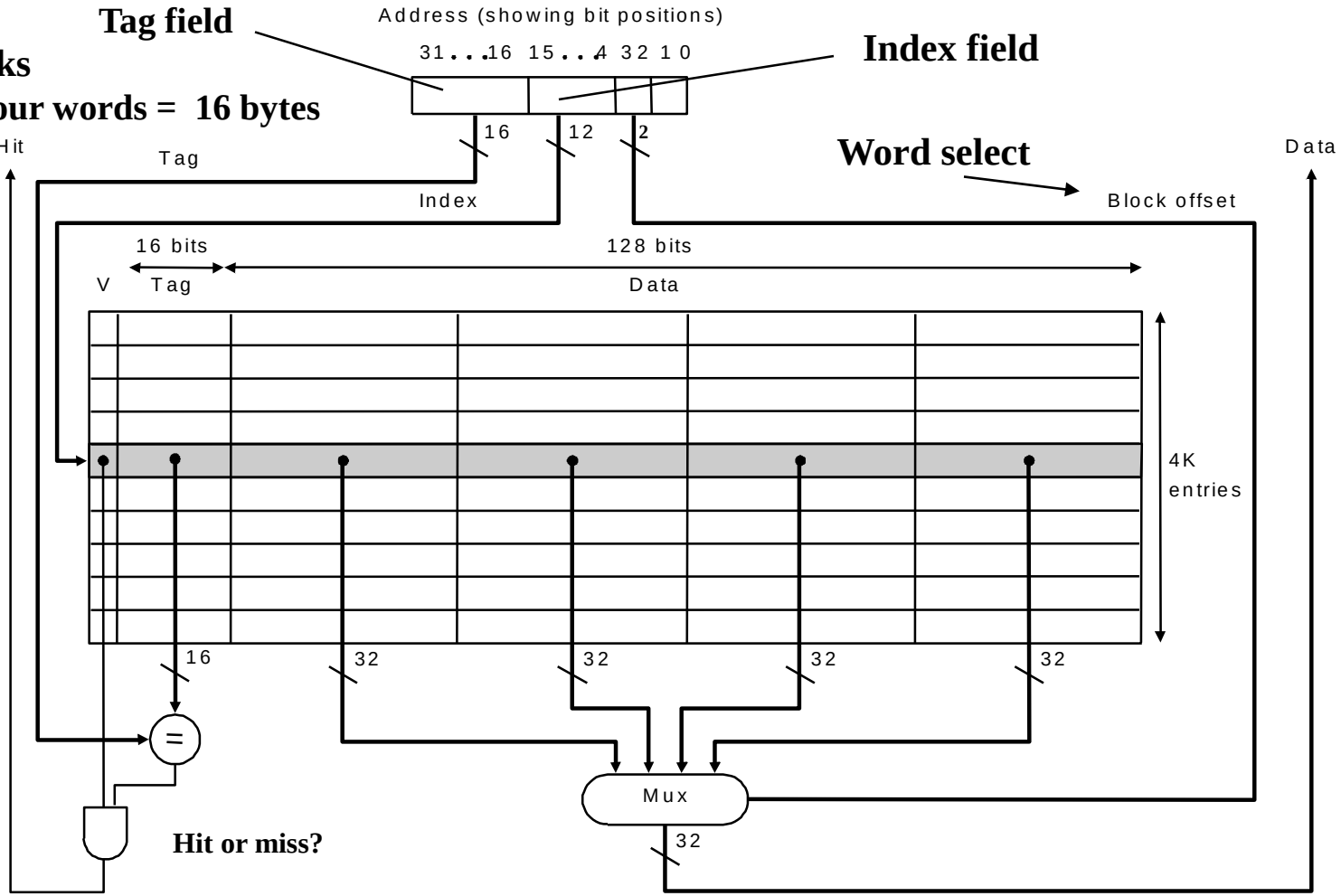


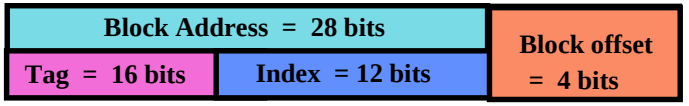
# 64KB Direct Mapped Cache Example

**4K= 4096 blocks**
**Each block = four words = 16 bytes**

**Can cache up to $2^{32}$ bytes = 4 GB of memory**




**Larger cache blocks take better advantage of spatial locality and thus may result in a lower miss rate**

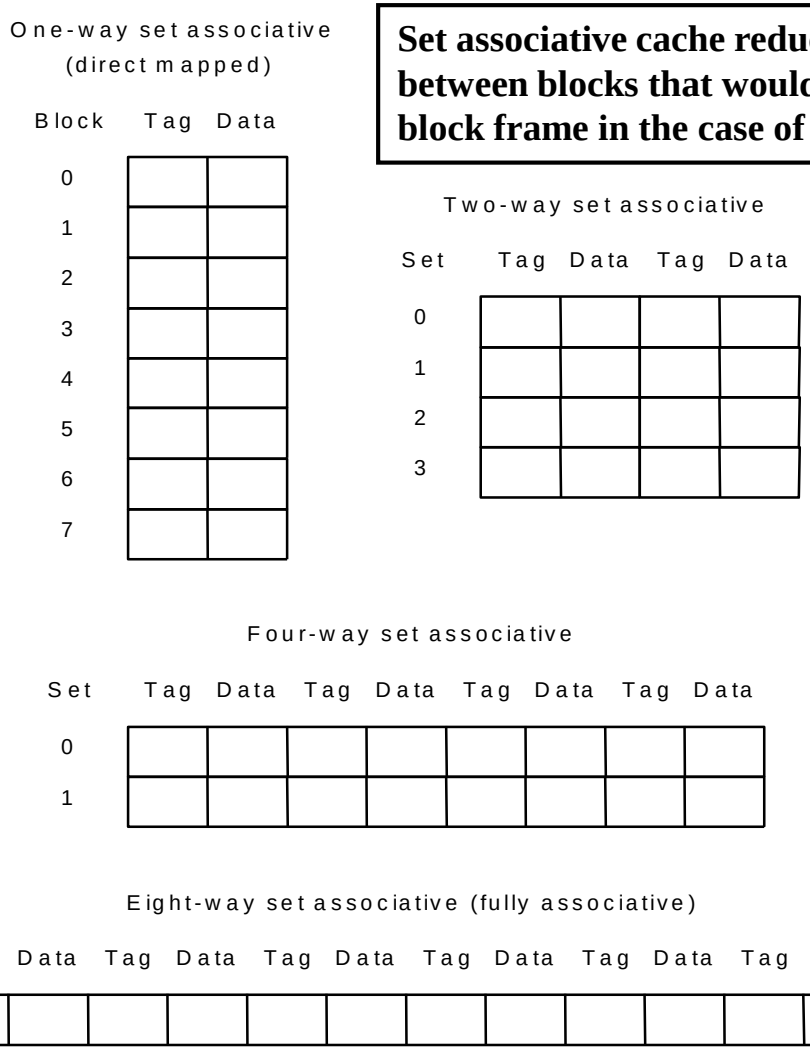



**Mapping Function: Cache Block frame number = (Block address) MOD (4096)**

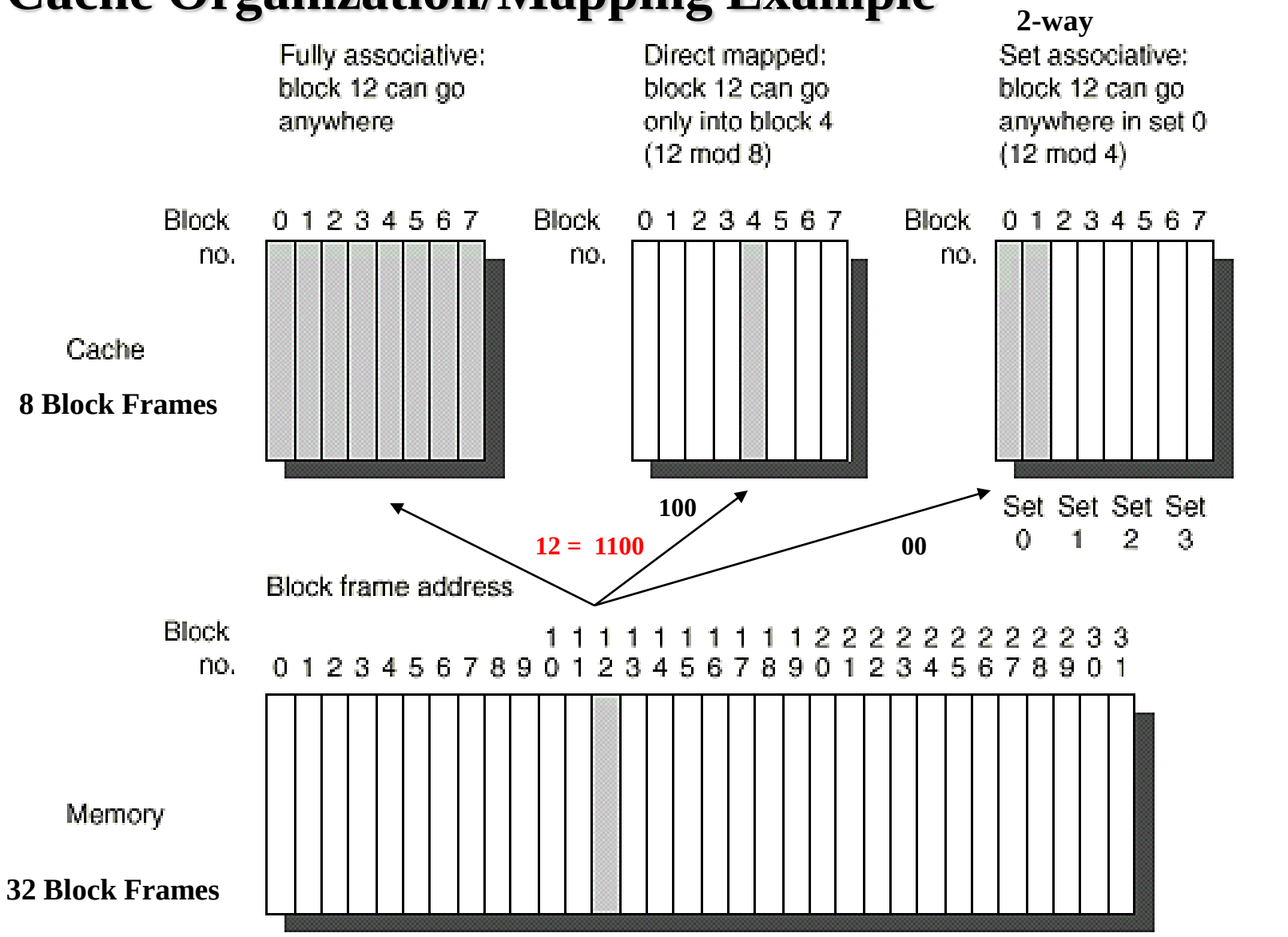
**i.e. index field or low 12 bits of block address**

# Cache Organization:
# Set Associative Cache

**Set associative cache reduces cache misses by reducing conflicts between blocks that would have been mapped to the same cache block frame in the case of direct mapped cache**

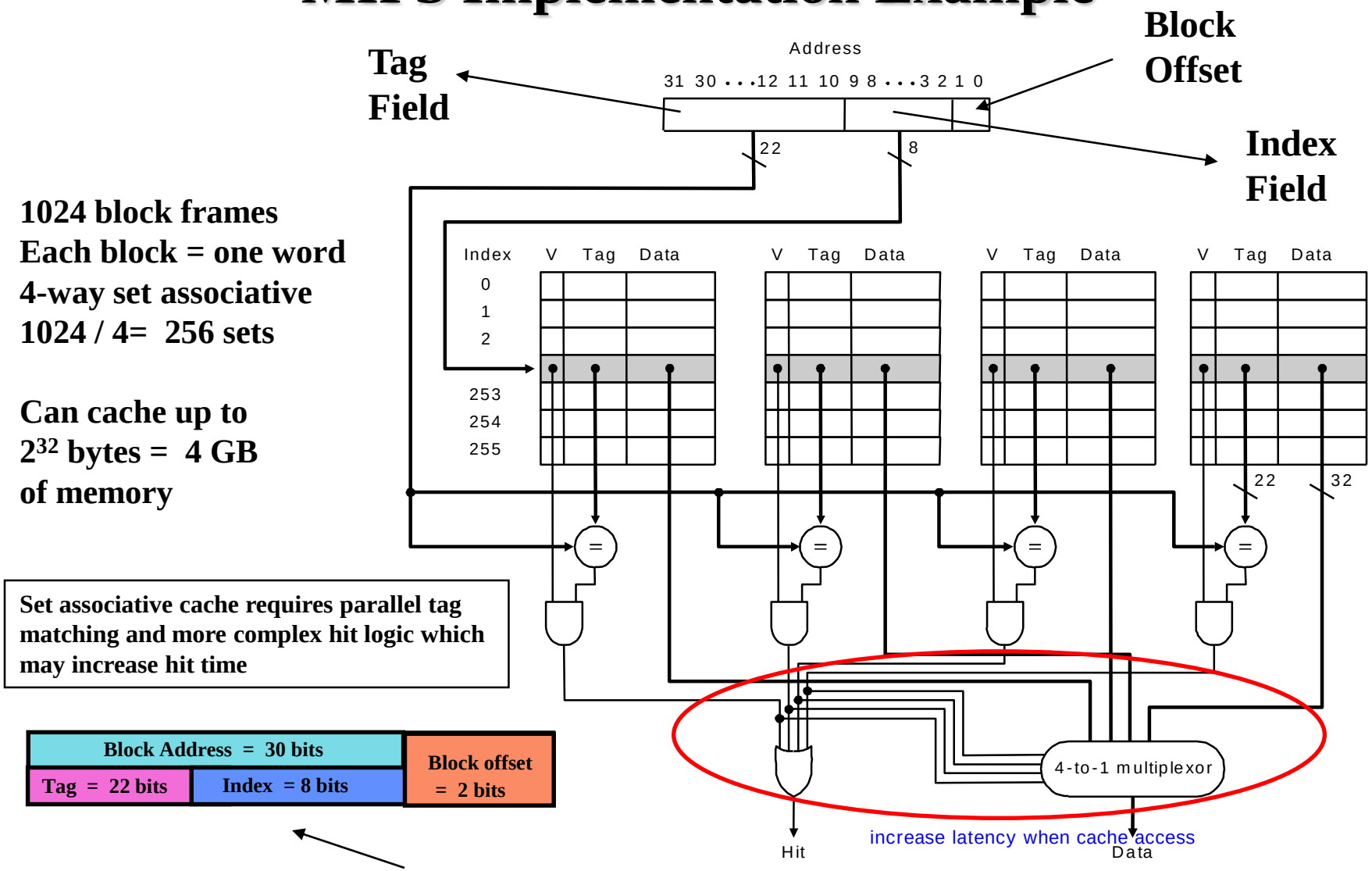


# Cache Organization/Mapping Example




2 This example cache has eight block frames and memory has 32 blocks.

# 4K Four-Way Set Associative Cache: MIPS Implementation Example




**1024 block frames**
**Each block = one word**
**4-way set associative**
**1024 / 4= 256 sets**

**Can cache up to**
**$2^{32}$ bytes = 4 GB**
**of memory**

**Set associative cache requires parallel tag matching and more complex hit logic which may increase hit time**

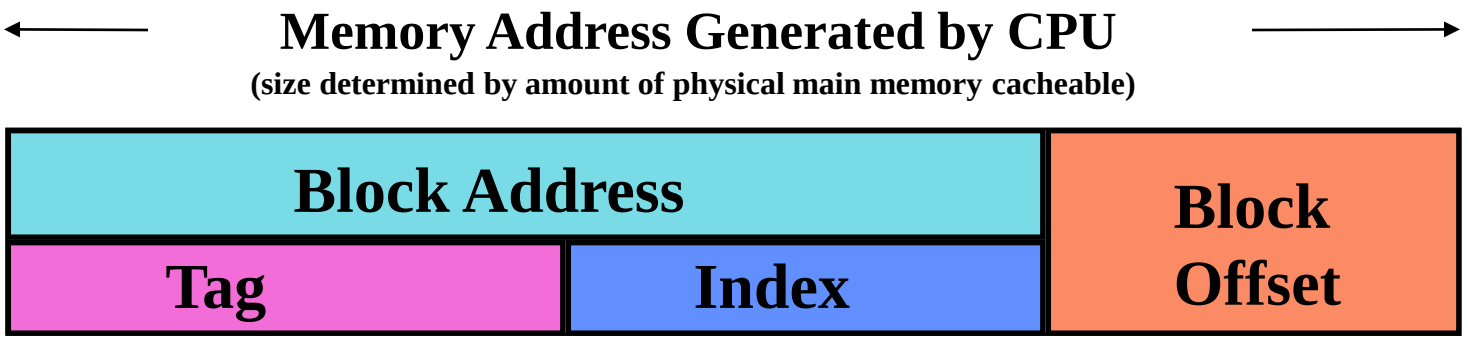
| Block Address = 30 bits | | Block offset = 2 bits |
|---|---|---|
| Tag = 22 bits | Index = 8 bits | |

**Mapping Function:** **Cache Set Number = index= (Block address) MOD (256)**

# Locating A Data Block in Cache

- **Each block frame in cache has an address tag.**
- **The tags of all cache blocks that might contain the requested data are checked in parallel.**
- **A valid bit is added to the tag to indicate whether this entry contains a valid address.**
- **The address from the CPU to cache is divided into:**
  - **A block address, further divided into:**
    - **An index field to choose a block frame/set in cache.**
      **(no index field for fully associative $).**
    - **A tag field to search and match addresses in the selected set.**
  - **A block offset to select bytes from the block.**

# Address Field Sizes/Mapping




Block offset size = $\log_2$(block size)

Index size = $\log_2$(Total number of blocks/associativity)

Tag size = address size - index size - offset size

Number of Sets in cache

**Mapping function:**

**Cache set or block frame number = Index =**

**= (Block Address) MOD (Number of Sets)**

**No index/mapping function for fully associative cache**

fully associativity = index 0 tag large
direct mapped cache = index large tag small

# Cache Replacement Policy

- **When a cache miss occurs the cache controller may have to select a block of cache data to be removed from a cache block frame and replaced with the requested data, such a block is selected by one of three methods:**

  ***(No cache replacement policy in direct mapped cache)***

  - **Random:**
    - **Any block is randomly selected for replacement providing uniform allocation.**
    - **Simple to build in hardware. Widely used cache replacement strategy.**
  - **Least-recently used (LRU):**
    - **Accesses to blocks are recorded and the block replaced is the one that was not used for the longest period of time.**
    - **Full LRU is *expensive* to implement, as the number of blocks to be tracked increases, and is usually approximated by block usage bits that are cleared at regular time intervals.**
  - **First In, First Out (FIFO):**
    - **Because LRU is complicated to implement, this approximates LRU by determining the oldest block rather than LRU**

# Miss Rates for Caches with Different Size, Associativity & Replacement Algorithm

## Sample Data

| Associativity: | 2-way | | 4-way | | 8-way | |
|---|---|---|---|---|---|---|
| Size | LRU | Random | LRU | Random | LRU | Random |
| 16 KB | 5.18% | 5.69% | 4.67% | 5.29% | 4.39% | 4.96% |
| 64 KB | 1.88% | 2.01% | 1.54% | 1.66% | 1.39% | 1.53% |
| 256 KB | 1.15% | 1.17% | 1.13% | 1.13% | 1.12% | 1.12% |

**Program steady state cache miss rates are given**
**Initially cache is empty and miss rates ~ 100%**

**FIFO replacement miss rates (not shown here) is better than random but worse than LRU**

**For SPEC92**