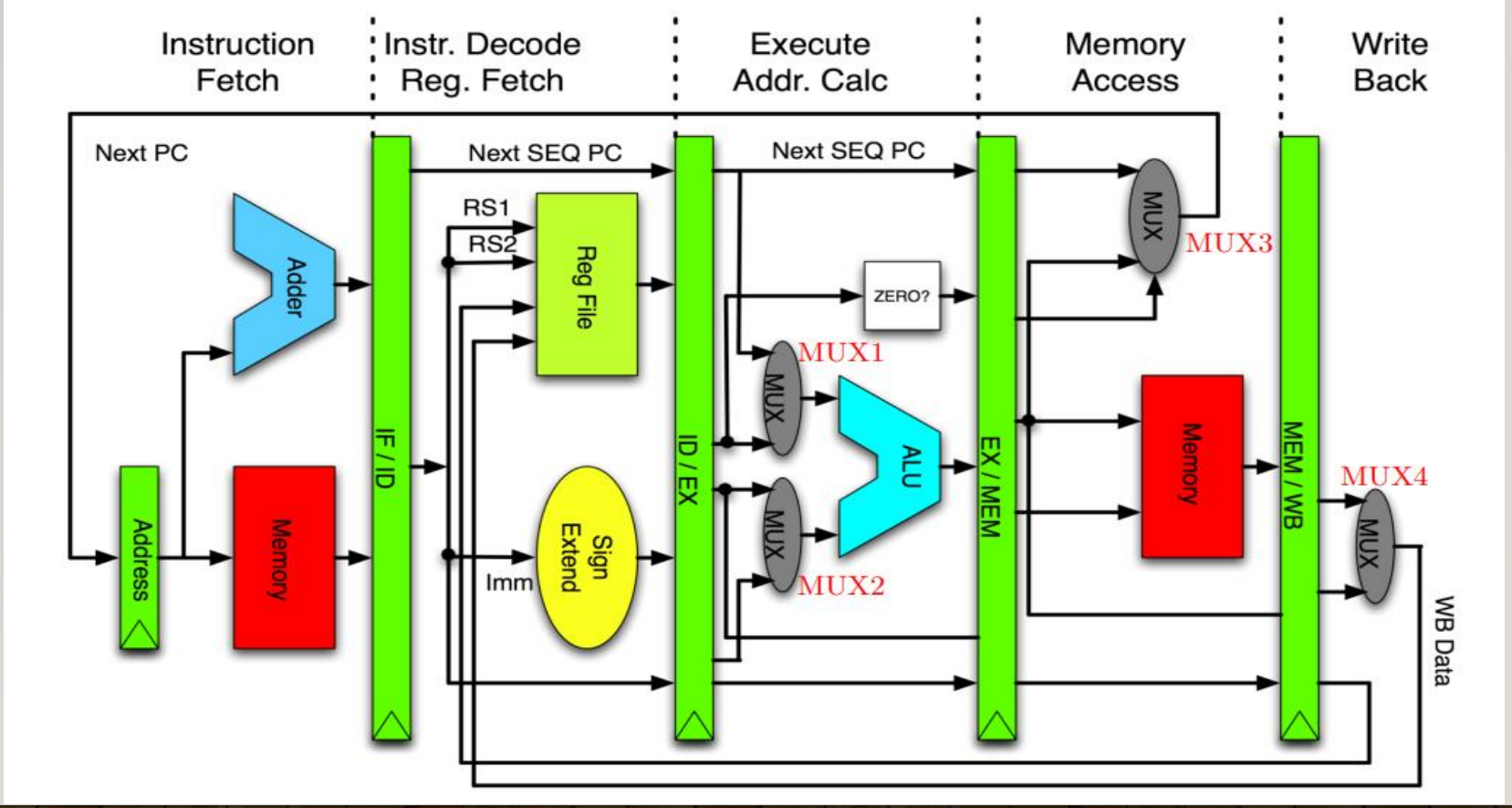
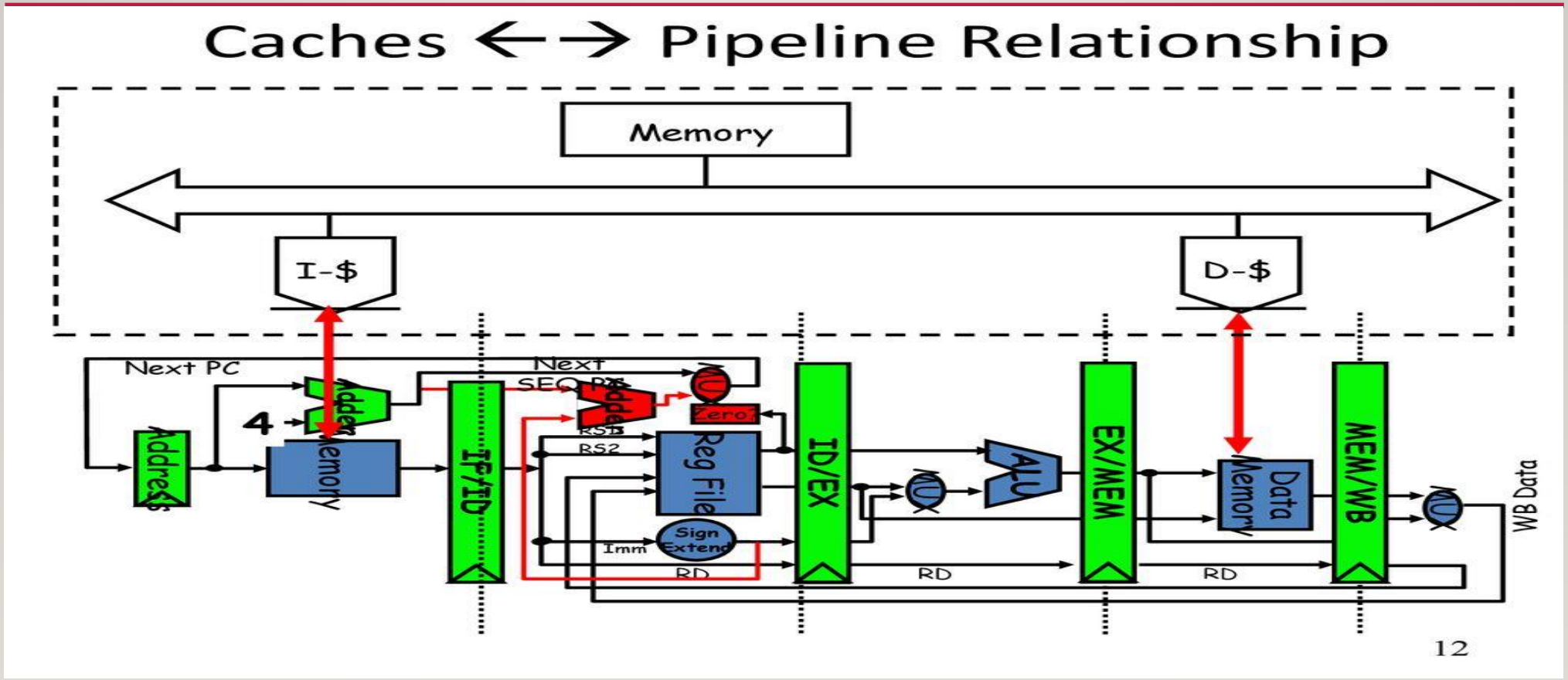
CACHE MEMORY

CHAPTER # 05 (APPENDIX B OF BOOK)

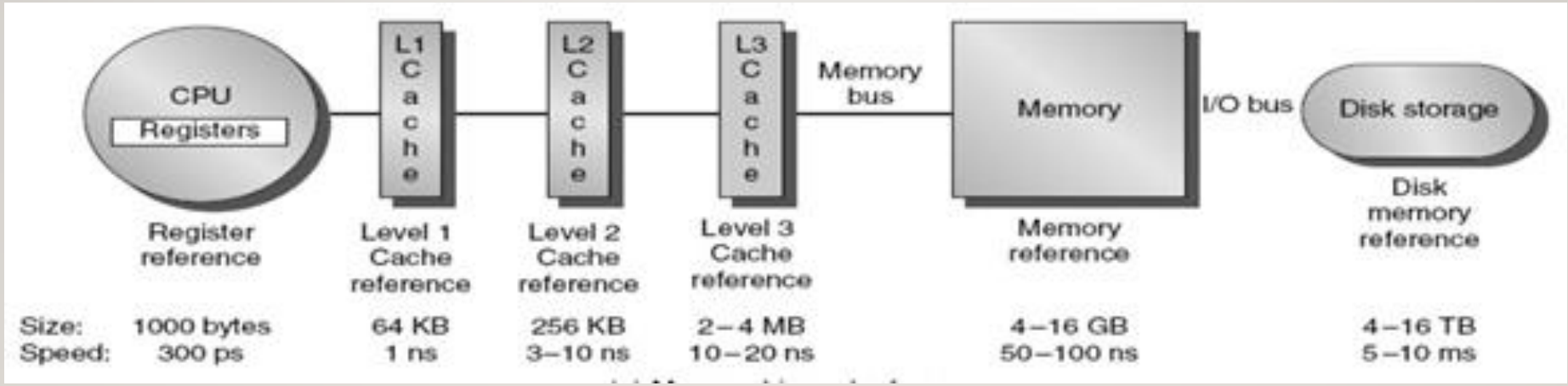
PIPELINED RISC-V DATA PATH



RELATIONSHIP OF CACHES WITH PIPELINE

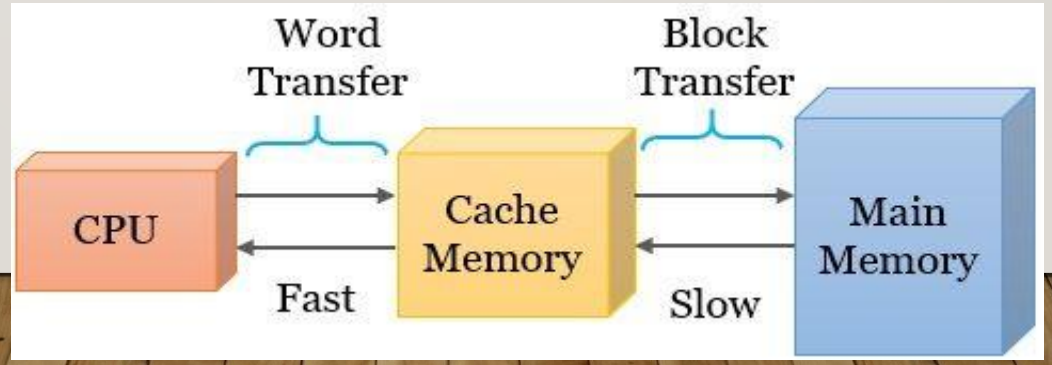


MEMORY HIERARCHY



CACHE MEMORY FUNDAMENTALS

- Cache is a small, fast buffer between processor and memory.
- Old values will be removed from cache to make space for new values.
- **Principle of Locality:** Programs access a relatively small portion of their address space at any instant of time.
- **Temporal Locality:** If an item is referenced , it will tend to be referenced again soon.
- **Spatial Locality:** If an item is referenced , items whose addresses are close by will tend to be referenced soon.



CACHE MEMORY FUNDAMENTALS

- **Block/Line**: Minimum unit of information that can be either present or not present in a cache level.
- **Hit:** An access where the data requested by the processor is present in the cache.
- **Miss** : An access where the data requested by the processor is not present in the cache.
- **Hit Time:** Time to access the cache memory block and return the data to the processor.
- **Hit Rate**/ **Miss Rate**: Fraction of memory access found (not found) in the cache.
- **Miss Penalty** : Time to replace a block in the cache with the corresponding block from the next level.

CACHE PERFORMANCE REVIEW

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$

$$\text{CPU time} = \left(\sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

The number of memory stall cycles depends on both the number of misses and the cost per miss, which is called the miss penalty:

$$\begin{aligned} \text{Memory stall cycles} &= \text{Number of misses} \times \text{Miss penalty} \\ &= \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty} \\ &= \text{IC} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty} \end{aligned}$$

CACHE PERFORMANCE REVIEW

$$\begin{aligned}\text{Memory stall clock cycles} = {} & \text{IC} \times \text{Reads per instruction} \times \text{Read miss rate} \times \text{Read miss penalty} \\ & + \text{IC} \times \text{Writes per instruction} \times \text{Write miss rate} \times \text{Write miss penalty}\end{aligned}$$

NUMERICAL

- Assume we have a computer where the cycles per instruction (CPI) is 1.0 when all memory accesses hit in the cache. The only data accesses are loads and stores, and these total 50% of the instructions. If the miss penalty is 50 clock cycles and the miss rate is 1%, how much faster would the computer be if all instructions were cache hits?

SOLUTION

$$\begin{aligned}\text{CPU execution time} &= (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle} \\ &= (\text{IC} \times \text{CPI} + 0) \times \text{Clock cycle} \\ &= \text{IC} \times 1.0 \times \text{Clock cycle}\end{aligned}$$

Now for the computer with the real cache, first we compute memory stall cycles:

$$\begin{aligned}\text{Memory stall cycles} &= \text{IC} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty} \\ &= \text{IC} \times (1 + 0.5) \times 0.01 \times 50 \\ &= \text{IC} \times 0.75\end{aligned}$$

$$\begin{aligned}\text{CPU execution time}_{\text{cache}} &= (\text{IC} \times 1.0 + \text{IC} \times 0.75) \times \text{Clock cycle} \\ &= 1.75 \times \text{IC} \times \text{Clock cycle}\end{aligned}$$

SOLUTION

The performance ratio is the inverse of the execution times:

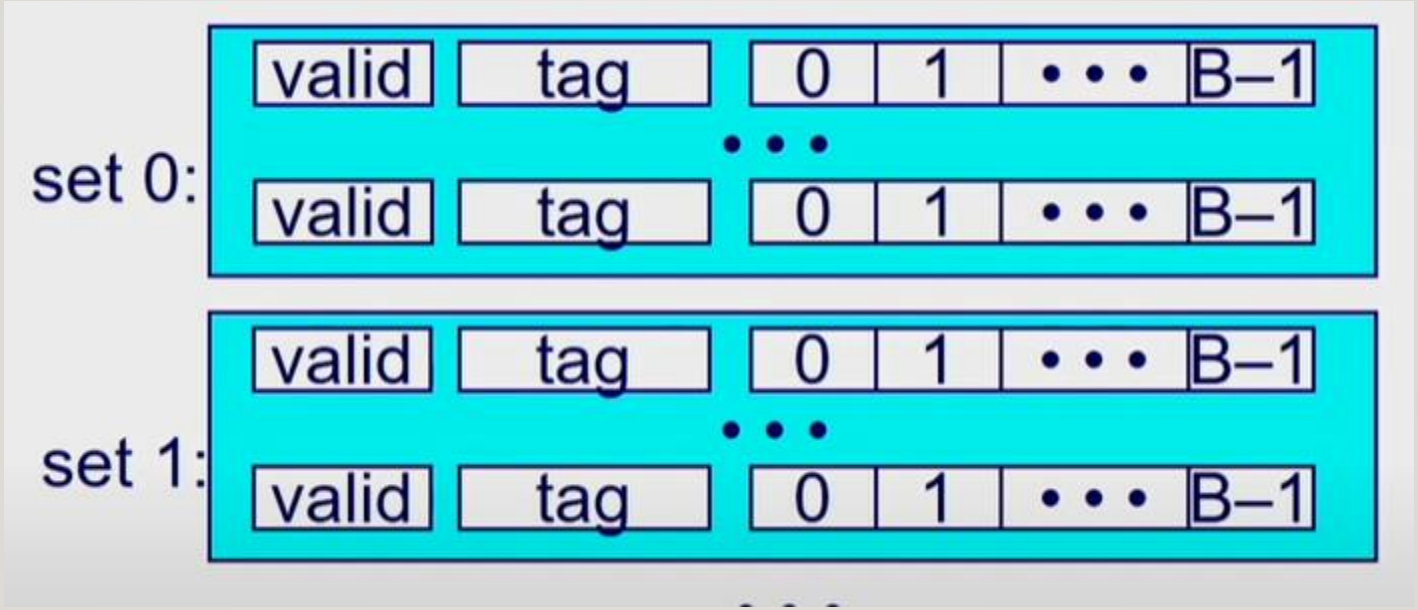
$$\frac{\text{CPU execution time}_{\text{cache}}}{\text{CPU execution time}} = \frac{1.75 \times \text{IC} \times \text{Clock cycle}}{1.0 \times \text{IC} \times \text{Clock cycle}}$$

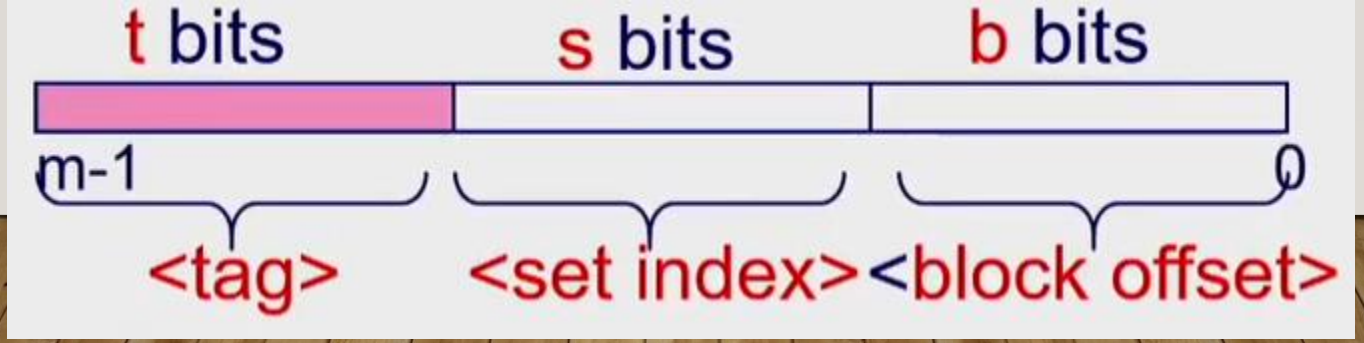
$$= 1.75$$

NUMERICAL

- Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.
- CPIexecution = 1.1
- Instruction mix: 50% arith/logic, 30% load/store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.
- How much faster would the computer be if all instructions were cache hits?

GENERAL ORGANIZATION OF CACHE

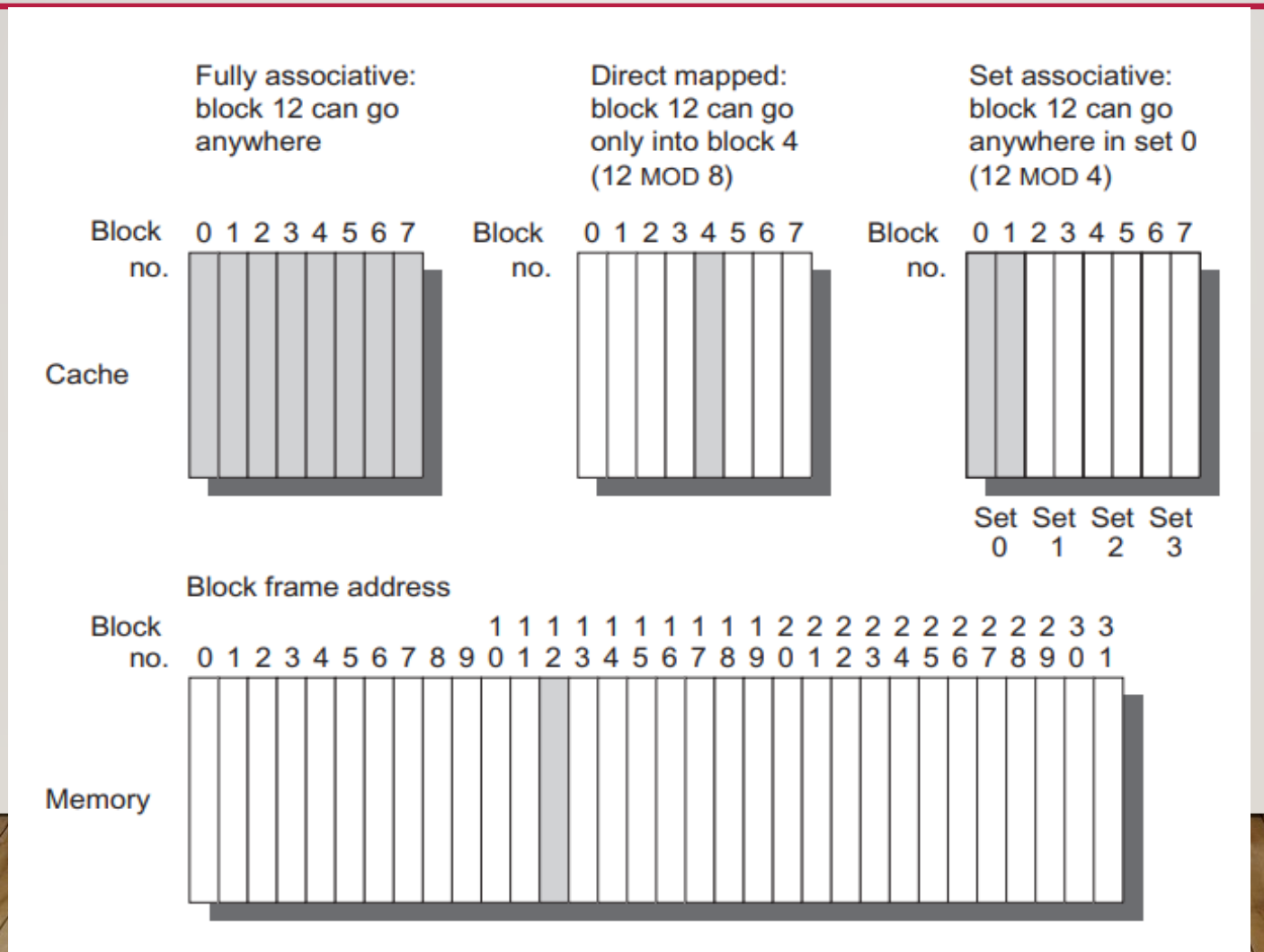




FOUR MEMORY HIERARCHY QUESTIONS

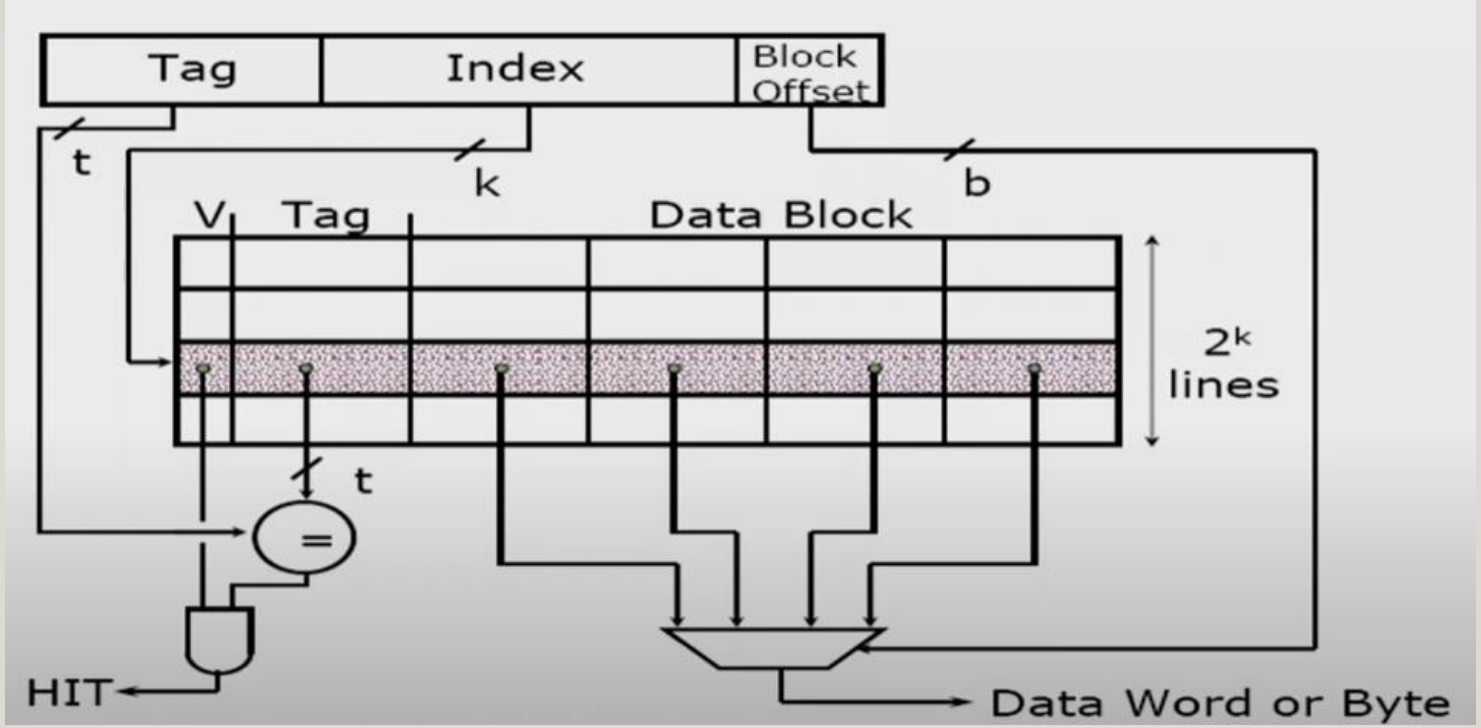
- Q1: Where can a block be placed in the upper level? (block placement)
- Q2: How is a block found if it is in the upper level? (block identification)
- Q3: Which block should be replaced on a miss? (block replacement)
- Q4: What happens on a write? (write strategy)

BLOCK PLACEMENT (DIRECT, ASSOCIATIVE)

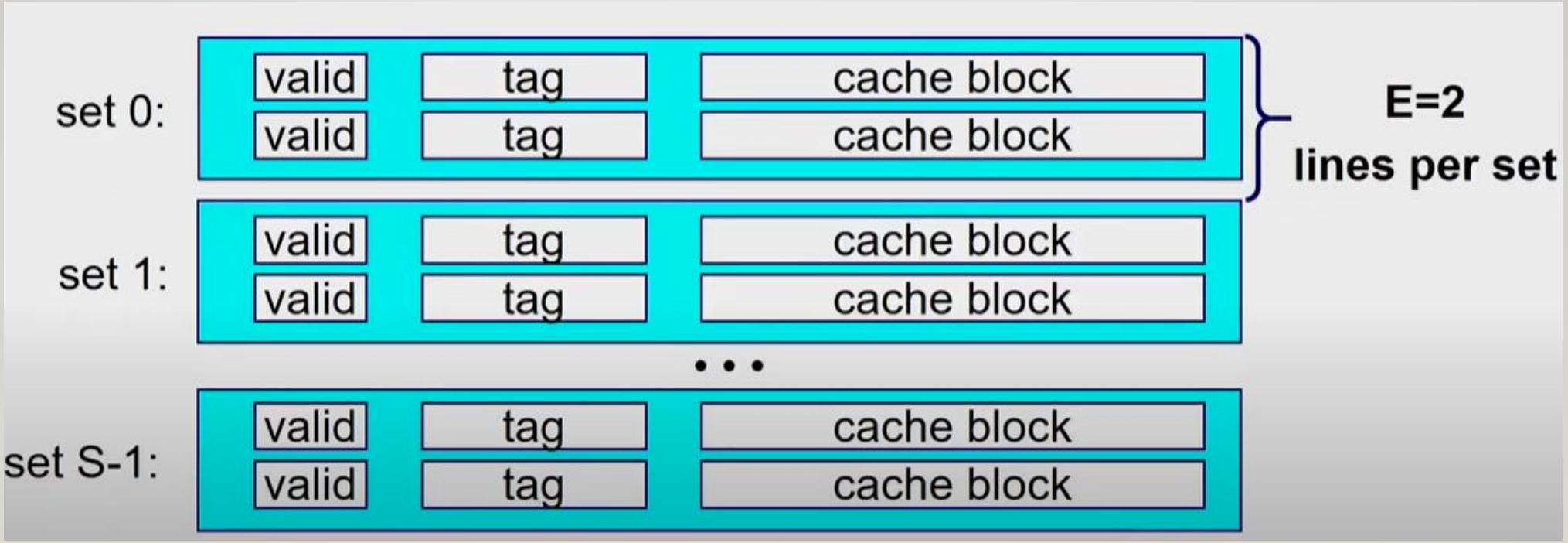


BLOCK IDENTIFICATION (DIRECT MAPPED)

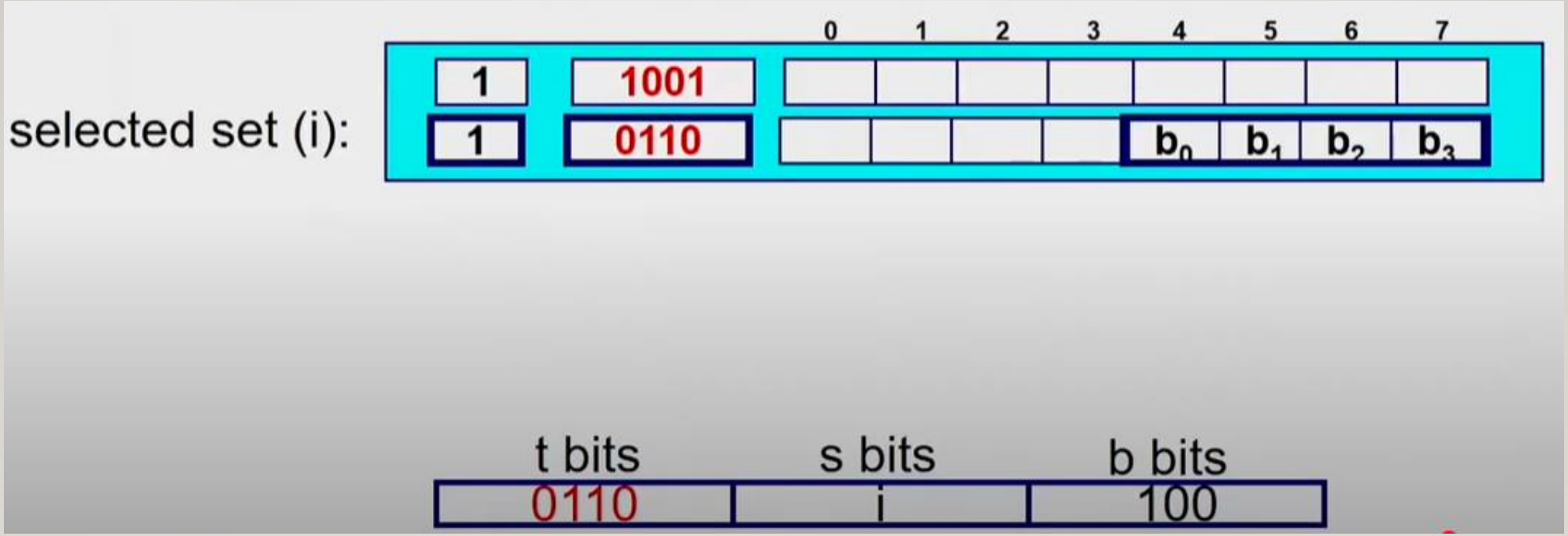
BLOCK IDENTIFICATION (DIRECT MAPPED)



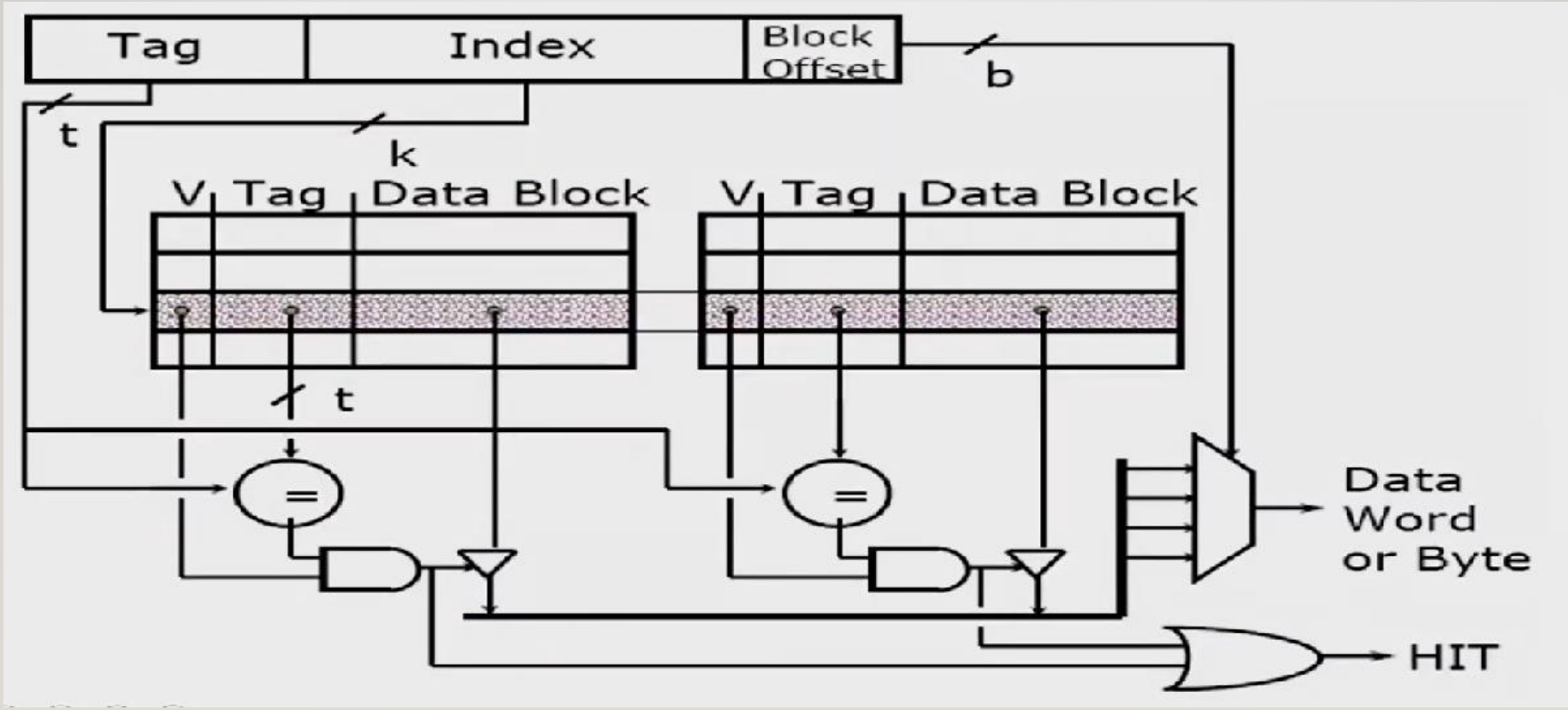
BLOCK IDENTIFICATION (SET ASSOCIATIVE CACHE)



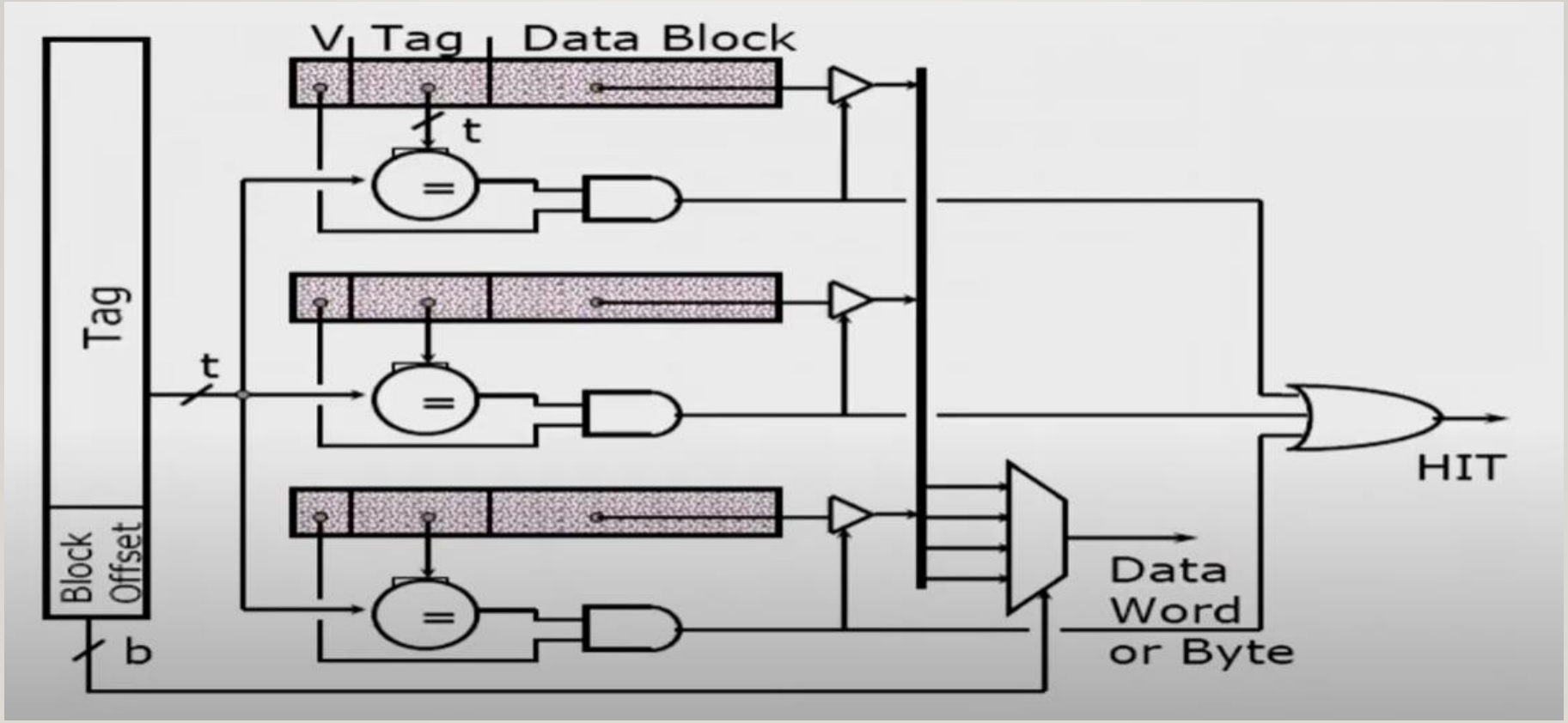
BLOCK IDENTIFICATION (SET ASSOCIATIVE CACHE)



BLOCK IDENTIFICATION (SET ASSOCIATIVE CACHE)



BLOCK IDENTIFICATION (FULLY ASSOCIATIVE CACHE)



BLOCK REPLACEMENT

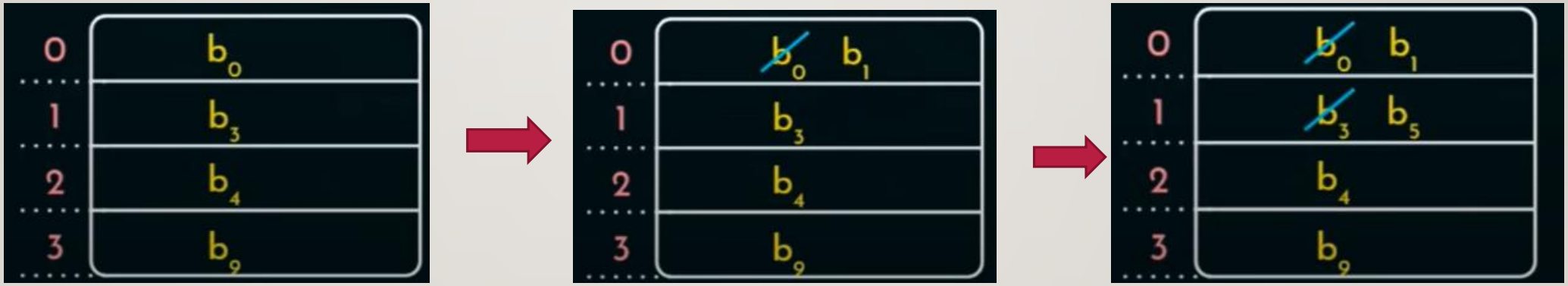
- When a miss occurs, the cache controller must select a block to be replaced with the desired data. A benefit of direct-mapped placement is that hardware decisions are simplified—in fact, so simple that there is no choice: only one block frame is checked for a hit, and only that block can be replaced. With fully associative or set associative placement, there are many blocks to choose from on a miss. There are three primary strategies employed for selecting which block to replace:
- **Random**
- **Least recently used (LRU)**
- **First in, first out (FIFO)**

BLOCK REPLACEMENT (RANDOM)

- To spread allocation uniformly, candidate blocks are randomly selected. Some systems generate pseudorandom block numbers to get reproducible behavior, which is particularly useful when debugging hardware.

- Not implemented now, was used in ARM architecture .

BLOCK REPLACEMENT (FIFO)

- Blocks are evicted in their order of Arrival.
- Cache behaves as FIFO queue .
- 

NUMERICAL

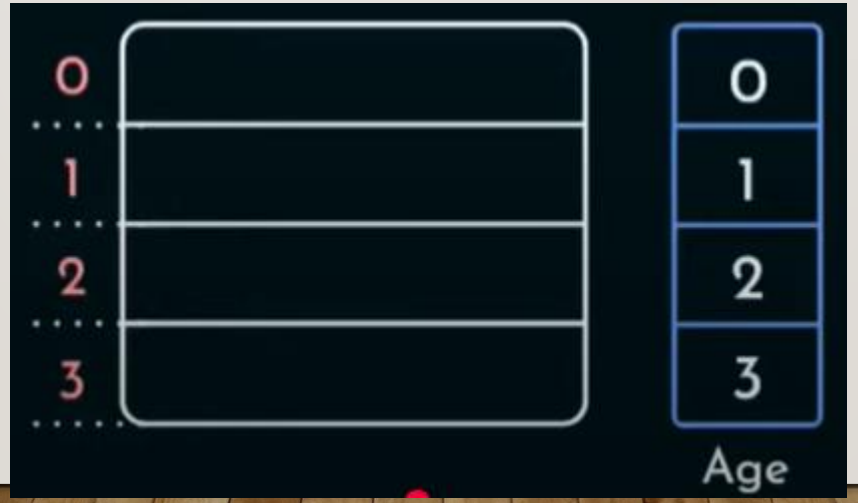
- Consider a Fully Associative Cache memory with 4 lines that implements FIFO cache replacement policy . For the following block request :

2, 3, 4, 7, 6, 3, 4, 7, 5, 4, 7, 8

- What is Hit and Miss Ratio respectively?

BLOCK REPLACEMENT (LEAST RECENTLY USED)

- Exploits Temporal locality principle
- Evicts least recently used block.



WRITE STRATEGY (CACHE HIT)

- **Write through**—The information is written to both the block in the cache and to the block in the lower-level memory.
- Cons: (Slower Cache Writes, increases traffic on memory bus)
- **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. (use of dirty bit)
- Pros: (Faster Cache Writes, Reduced traffic on memory bus)
- Cons: (Complex Hardware, Data Loss)

WRITE STRATEGY (CACHE MISS)

- **Write allocate**—The block is allocated on a write miss, followed by the preceding write hit actions. In this natural option, write misses act like read misses.
- Cons: (if data is not used frequently, creates cache pollution)
- **No-write allocate**—This apparently unusual alternative is write misses do not affect the cache. Instead, the block is modified only in the lower-level memory.