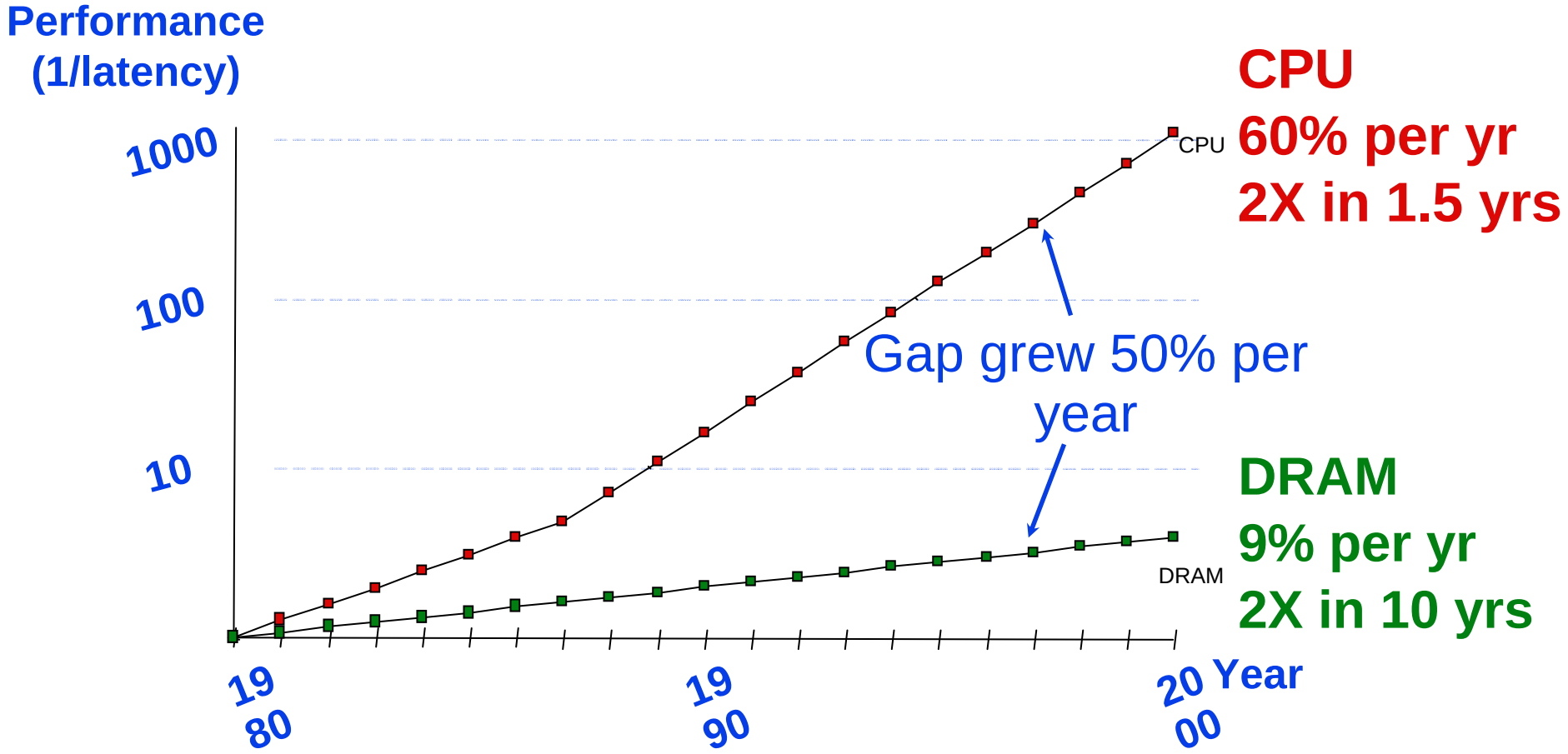
ECE 563
Advanced Computer Architecture

Fall 2007

Lecture 3: Memory Hierarchy Review: Caches

Since 1980, CPU has outpaced DRAM ...

Four-issue 2GHz superscalar accessing 100ns DRAM could execute 800 instructions during time for one memory access!

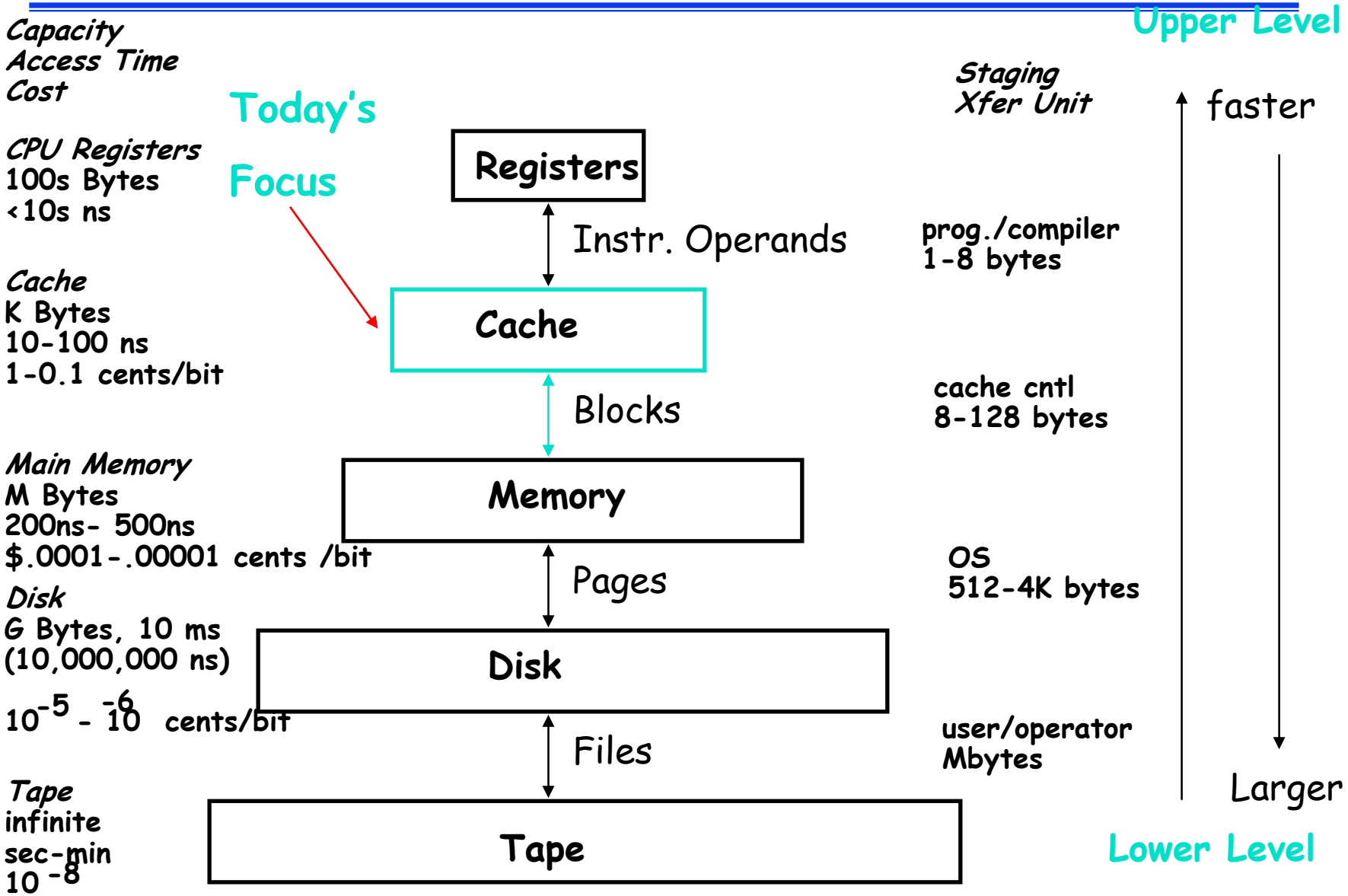


Addressing the Processor-Memory Performance GAP

- **Goal:** Illusion of large, fast, cheap memory. Let programs address a memory space that scales to the disk size, at a speed that is usually as fast as register access

- **Solution:** Put smaller, faster "cache" memories between CPU and DRAM. Create a "memory hierarchy".

Levels of the Memory Hierarchy



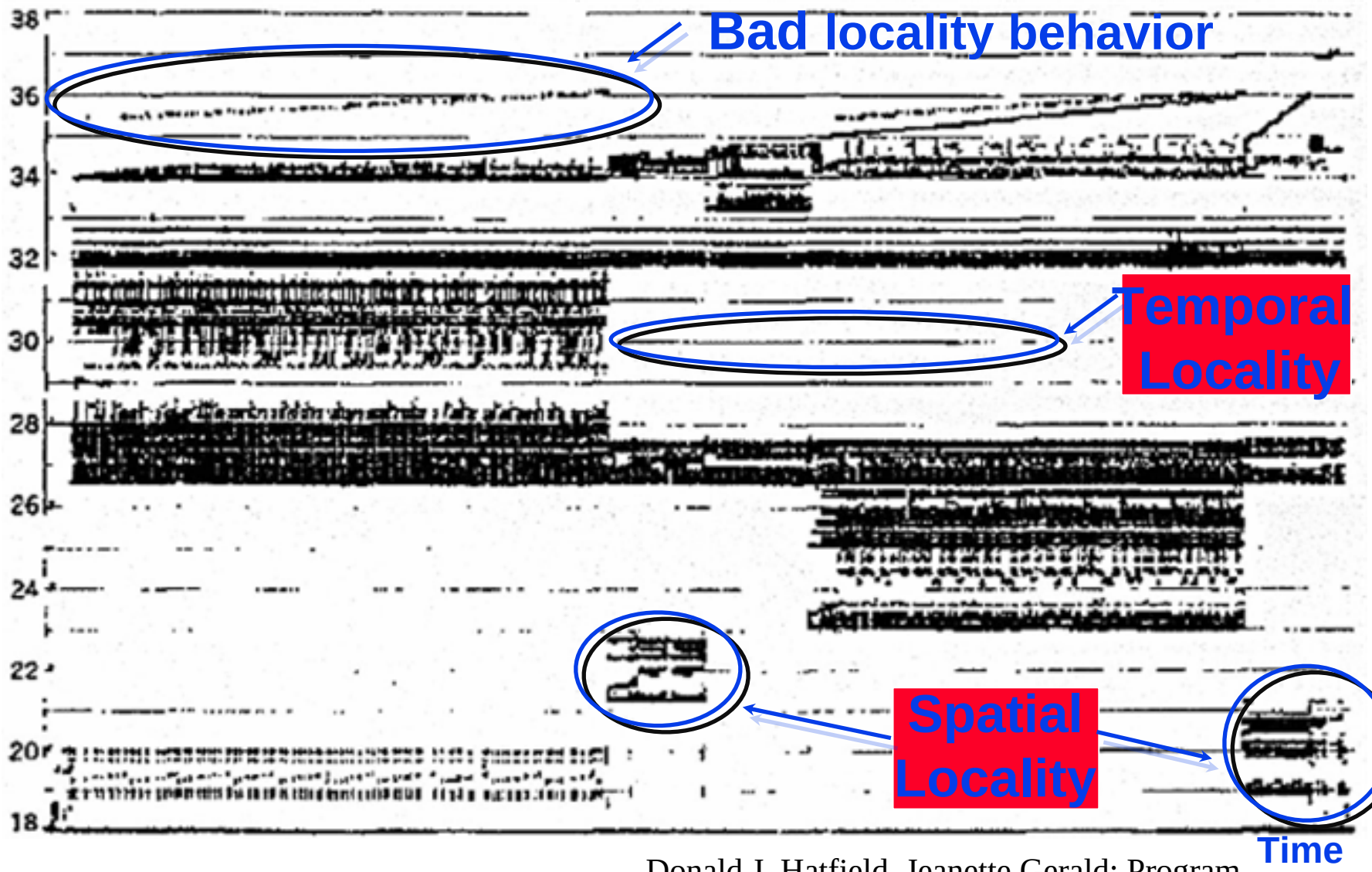
Common Predictable Patterns

Two predictable properties of memory references:

- **Temporal Locality: If a location is referenced, it is likely to be referenced again in the near future (e.g., loops, reuse).**

- **Spatial Locality: If a location is referenced it is likely that locations near it will be referenced in the near future (e.g., straightline code, array access).**

Memory Reference Patterns



Donald J. Hatfield, Jeanette Gerald: Program Restructuring for Virtual Memory. IBM Systems Journal 10(3): 168-192 (1971)

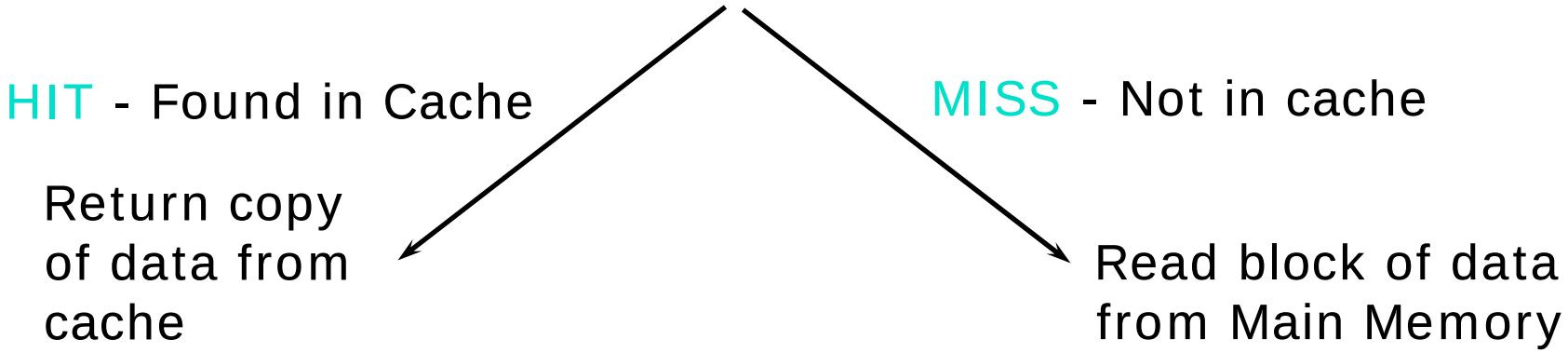
Caches

Caches exploit both types of predictability:

- **Exploit temporal locality by remembering the contents of recently accessed locations.**
- **Exploit spatial locality by fetching blocks of data around recently accessed locations.**

Cache Algorithm (Read)

Look at Processor Address, search cache tags to find match. Then either



Hit Rate = fraction of accesses found in cache

Miss Rate = 1 – Hit rate

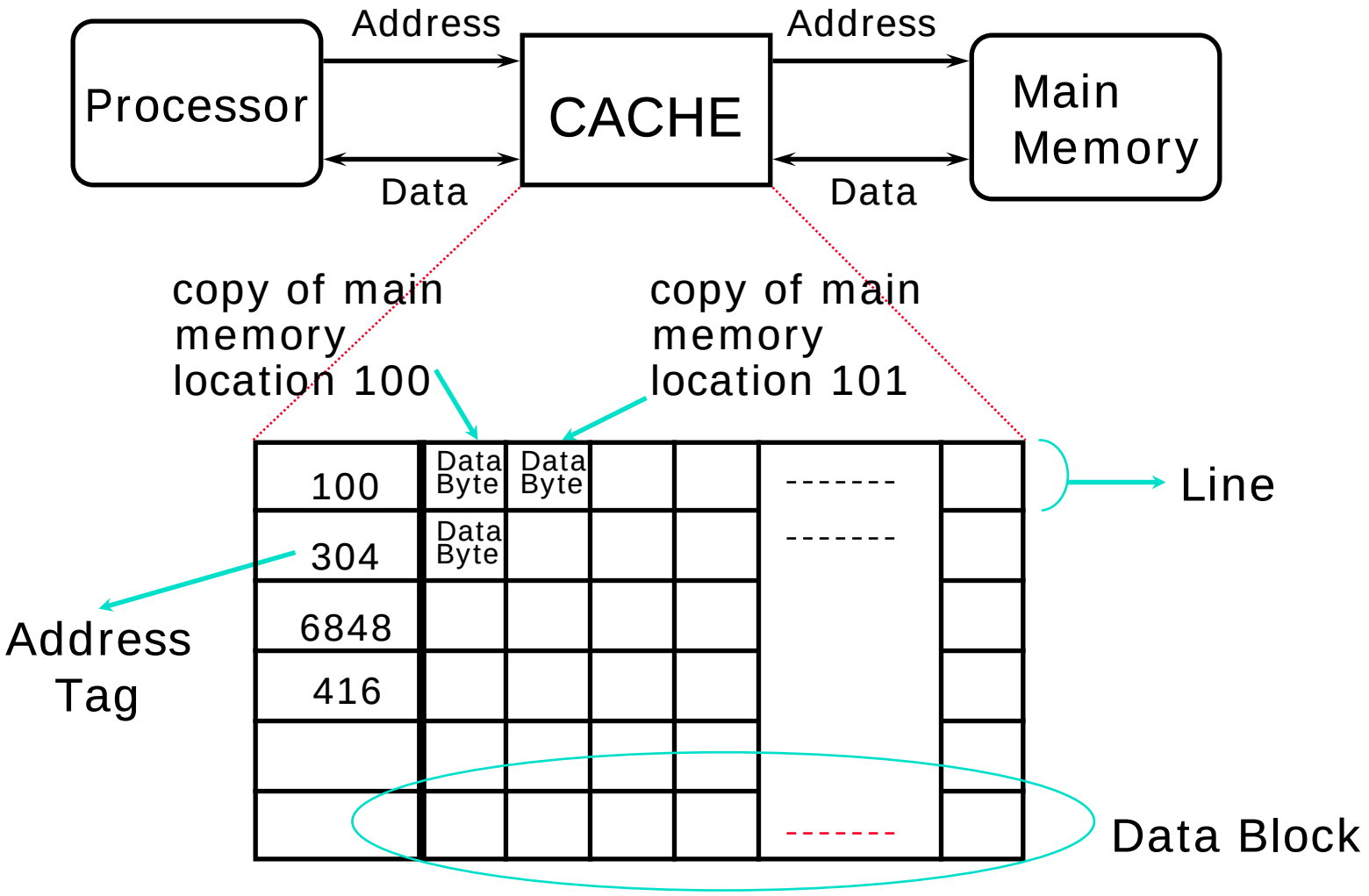
Hit Time = RAM access time +

time to determine HIT/MISS

Miss Time = time to replace block in cache +

time to deliver block to processor

Inside a Cache



4 Questions for Memory Hierarchy

- **Q1: Where can a block be placed in the cache?**

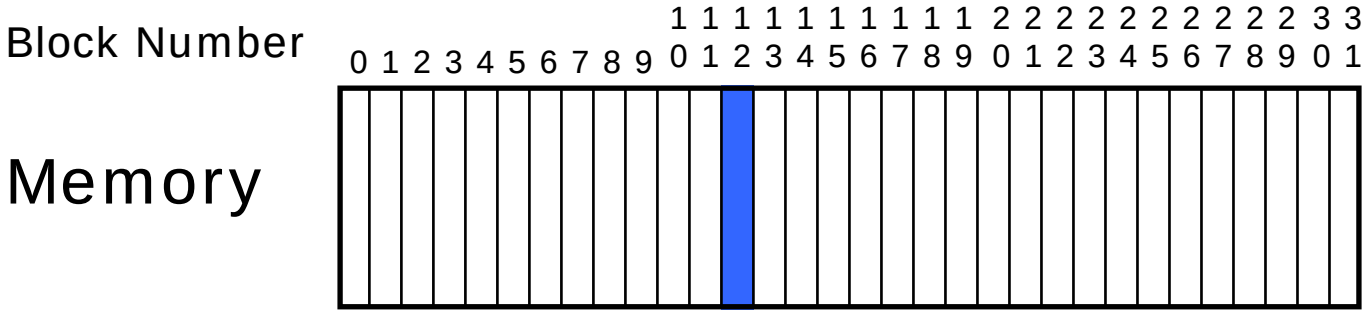
  (Block placement)

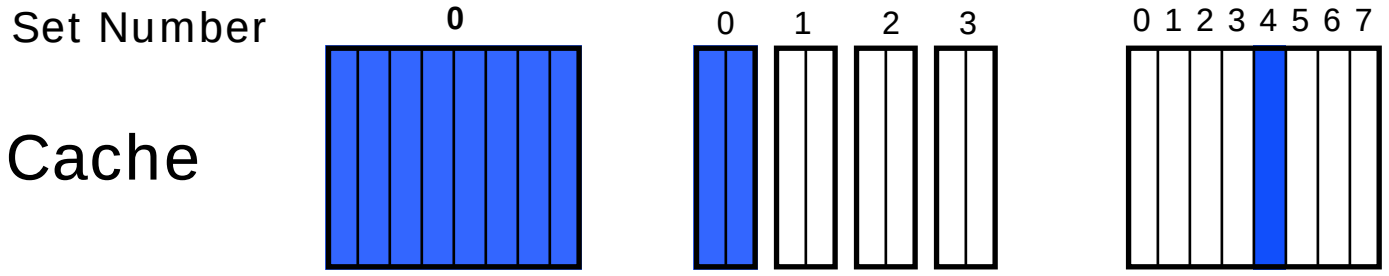
- **Q2: How is a block found if it is in the cache?**
  (Block identification)

- **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?





Fully Associative
anywhere

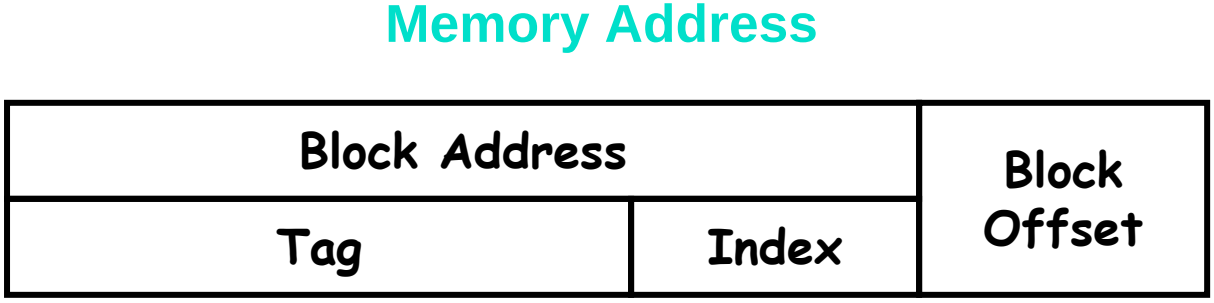
(2-way) Set Associative
anywhere in set 0
(12 mod 4)

Direct Mapped
only into block 4
(12 mod 8)

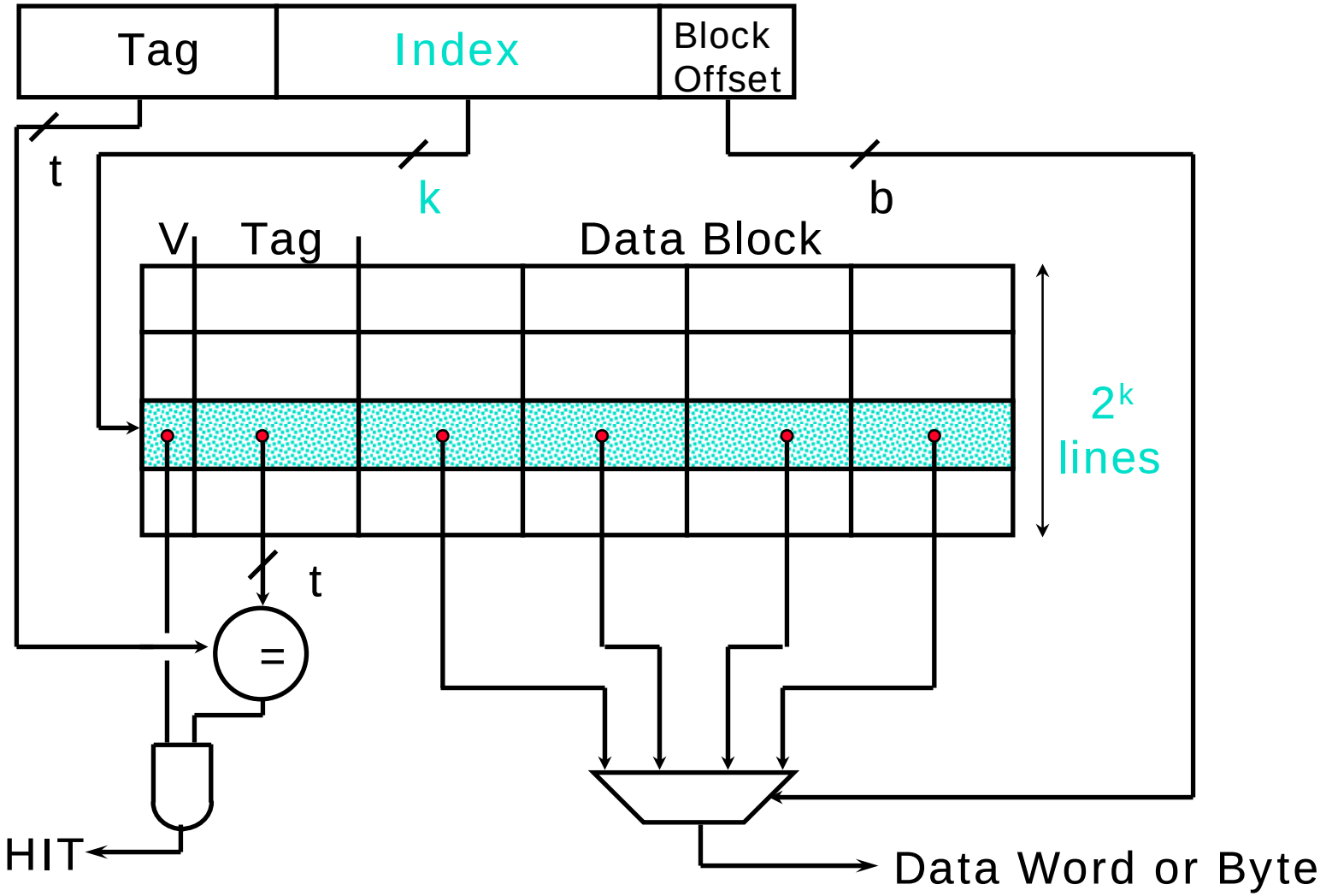
Block 12 can be placed

Q2: How is a block found?

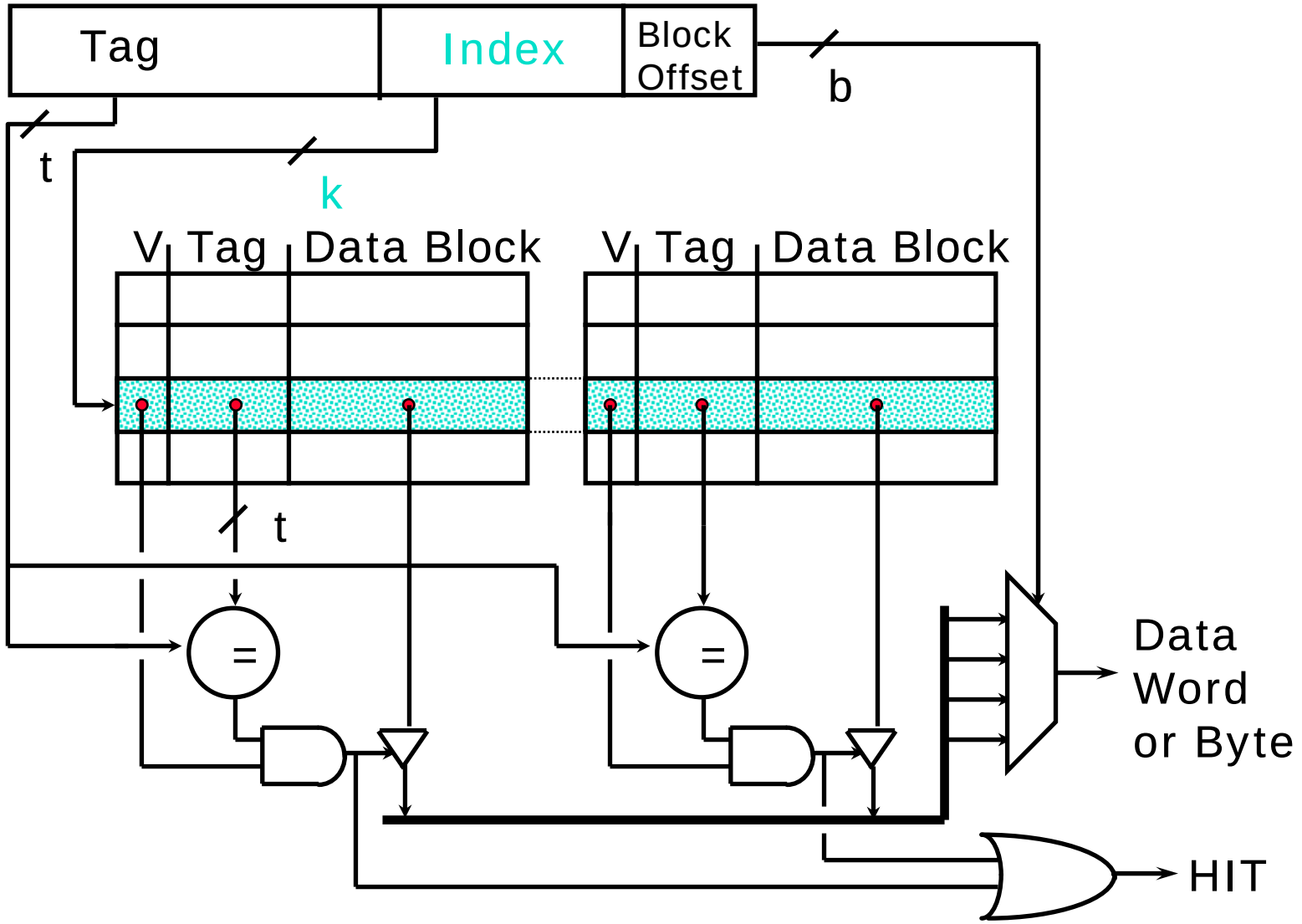
- Index selects which set to look in
- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag. Fully Associative caches have no index field.



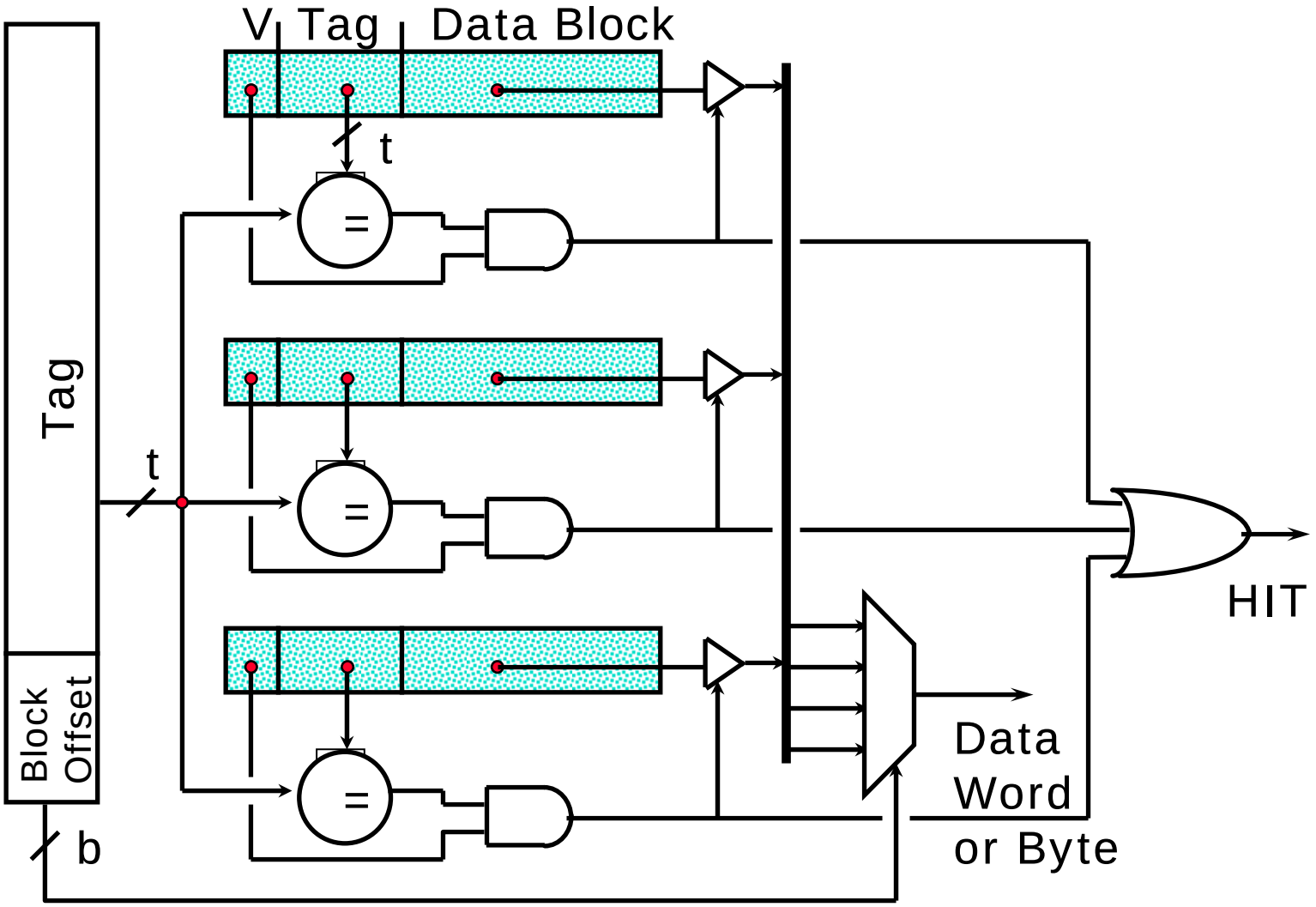
Direct-Mapped Cache



2-Way Set-Associative Cache



Fully Associative Cache

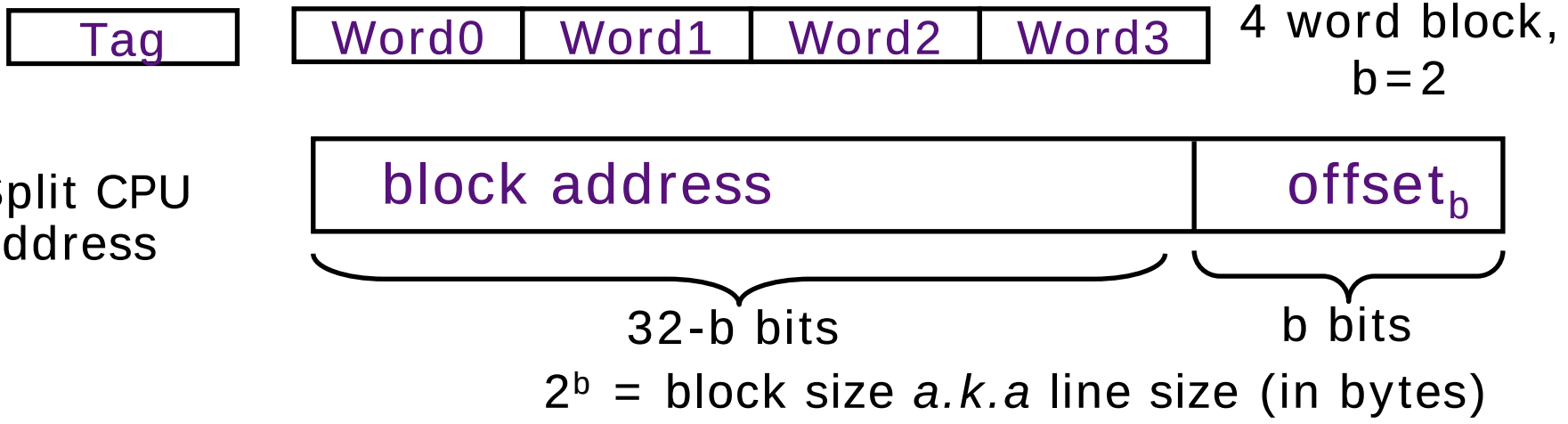


What causes a MISS?

- Three Major Categories of Cache Misses:
  - Compulsory Misses: first access to a block
  - Capacity Misses: cache cannot contain all blocks needed to execute the program
  - Conflict Misses: block replaced by another block and then later retrieved - (affects set assoc. or direct mapped caches)
    - Nightmare Scenario: ping pong effect!

Block Size and Spatial Locality

Block is unit of transfer between the cache and memory



Larger block size has distinct hardware advantages

- less tag overhead
- exploit fast burst transfers from DRAM
- exploit fast burst transfers over wide busses

What are the disadvantages of increasing block size?

Fewer blocks => more conflicts. Can waste bandwidth.

Q3: Which block should be replaced on a miss?

- **Easy for Direct Mapped**
- **Set Associative or Fully Associative:**
  - **Random**
  - **Least Recently Used (LRU)**
    - LRU cache state must be updated on every access
    - true implementation only feasible for small sets (2-way)
    - pseudo-LRU binary tree often used for 4-8 way
  - **First In, First Out (FIFO) a.k.a. Round-Robin**
    - used in highly associative caches
- **Replacement policy has a second order effect since replacement only happens on misses**

Q4: What happens on a write?

- **Cache hit:**
  - ***write through:*** **write both cache & memory**
    - **generally higher traffic but simplifies cache coherence**
  - ***write back:*** **write cache only (memory is written only when the entry is evicted)**
    - **a dirty bit per block can further reduce the traffic**
- **Cache miss:**
  - ***no write allocate:*** **only write to main memory**
  - ***write allocate (aka fetch on write):*** **fetch into cache**
- **Common combinations:**
  - **write through and no write allocate**
  - **write back with write allocate**

5 Basic Cache Optimizations

- **Reducing Miss Rate**
  1. Larger Block size (compulsory misses)
  2. Larger Cache size (capacity misses)
  3. Higher Associativity (conflict misses)

- **Reducing Miss Penalty**
  1. Multilevel Caches

- **Reducing hit time**
  5. Giving Reads Priority over Writes
     - E.g., Read complete before earlier writes in write buffer