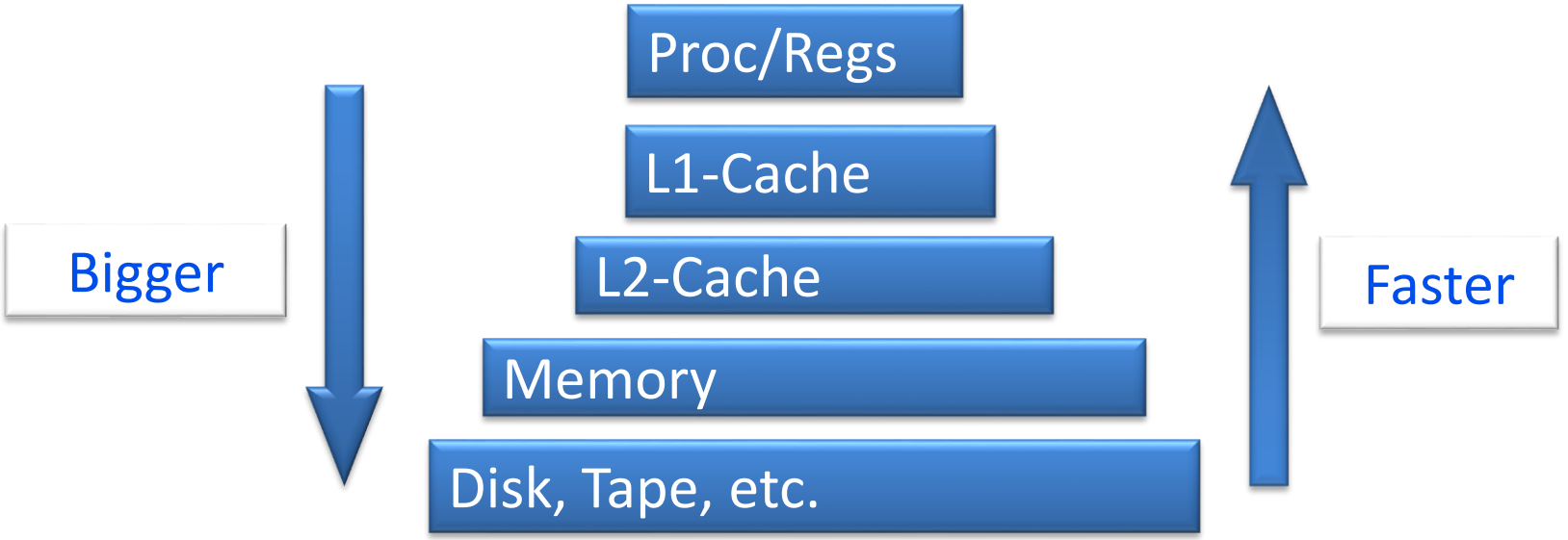
ΙΕΡΑΡΧΙΚΗ ΟΡΓΑΝΩΣΗ ΜΝΗΜΗΣ

Τι είναι κρυφή μνήμη (cache)

- Μικρή και γρήγορη μνήμη που χρησιμοποιείται για να μειώσει τον μέσο χρόνο προσπέλασης σε αργή μνήμη



ΤΟΠΙΚΟΤΗΤΑ

- **Χρονική Τοπικότητα (Temporal Locality)**
  - μια λέξη μνήμης που χρησιμοποιήθηκε πρόσφατα στο παρελθόν, έχει μεγάλη πιθανότητα να ξανά χρησιμοποιηθεί σύντομα στο μέλλον.
- **Τοπική Τοπικότητα (Spatial Locality):**
  - οι λέξεις κοντά σε μία λέξη που χρησιμοποιήθηκε πρόσφατα στο παρελθόν έχουν μεγάλη πιθανότητα να χρησιμοποιηθούν σύντομα στο μέλλον.

Παραδείγματα Τοπικότητας

```
sum = 0;
for (i = 0; i<n; i++)
    sum += a[i];
return sum;
```

- Προσπελάσεις Δεδομένων
  - Προσπέλαση των στοιχείων του πίνακα με τη σειρά (stride-1 reference pattern). — Spatial locality
  - Προσπέλαση της μεταβλητής `sum` σε κάθε επανάληψη. — Temporal locality
- Προσπελάσεις εντολών
  - Προσπέλαση εντολών στη σειρά. — Spatial locality
  - Επανάληψη εντολών σε loop. — Temporal locality

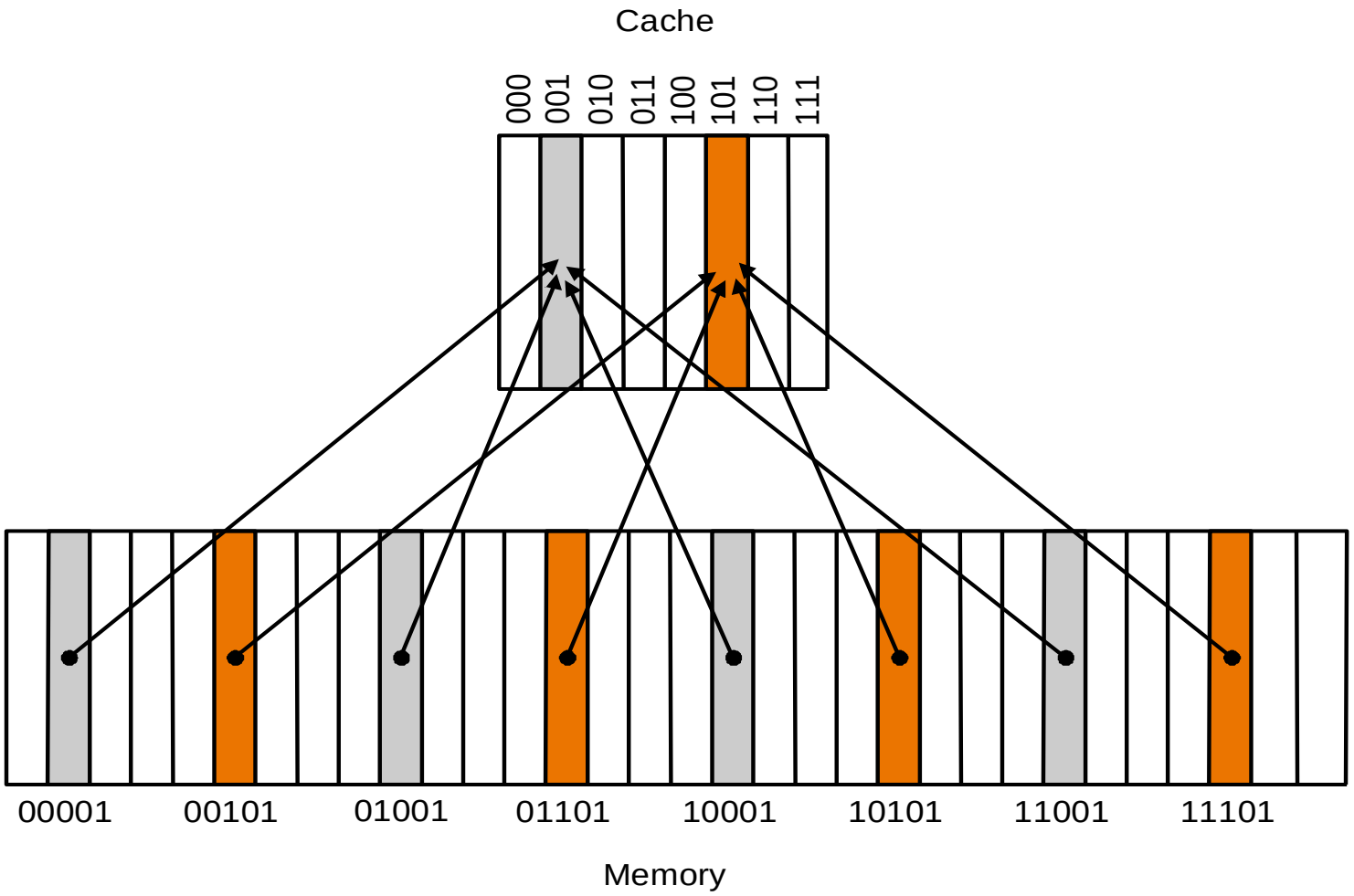
ΜΝΗΜΗ CACHE

- πλήρως προσεταιριστική (fully associative cache)
  - Κάθε λέξη της κύριας μνήμης μπορεί να αποθηκευθεί οπουδήποτε στην cache
    - Μεγαλύτερη ωφέλιμη χωρητικότητα
- μονοσήμαντης απεικόνισης direct mapped cache
  - Κάθε λέξη της κύριας μνήμης μπορεί να αποθηκευθεί μόνο σε μία θέση στην cache
    - Εύκολη αναζήτηση

Ορολογία

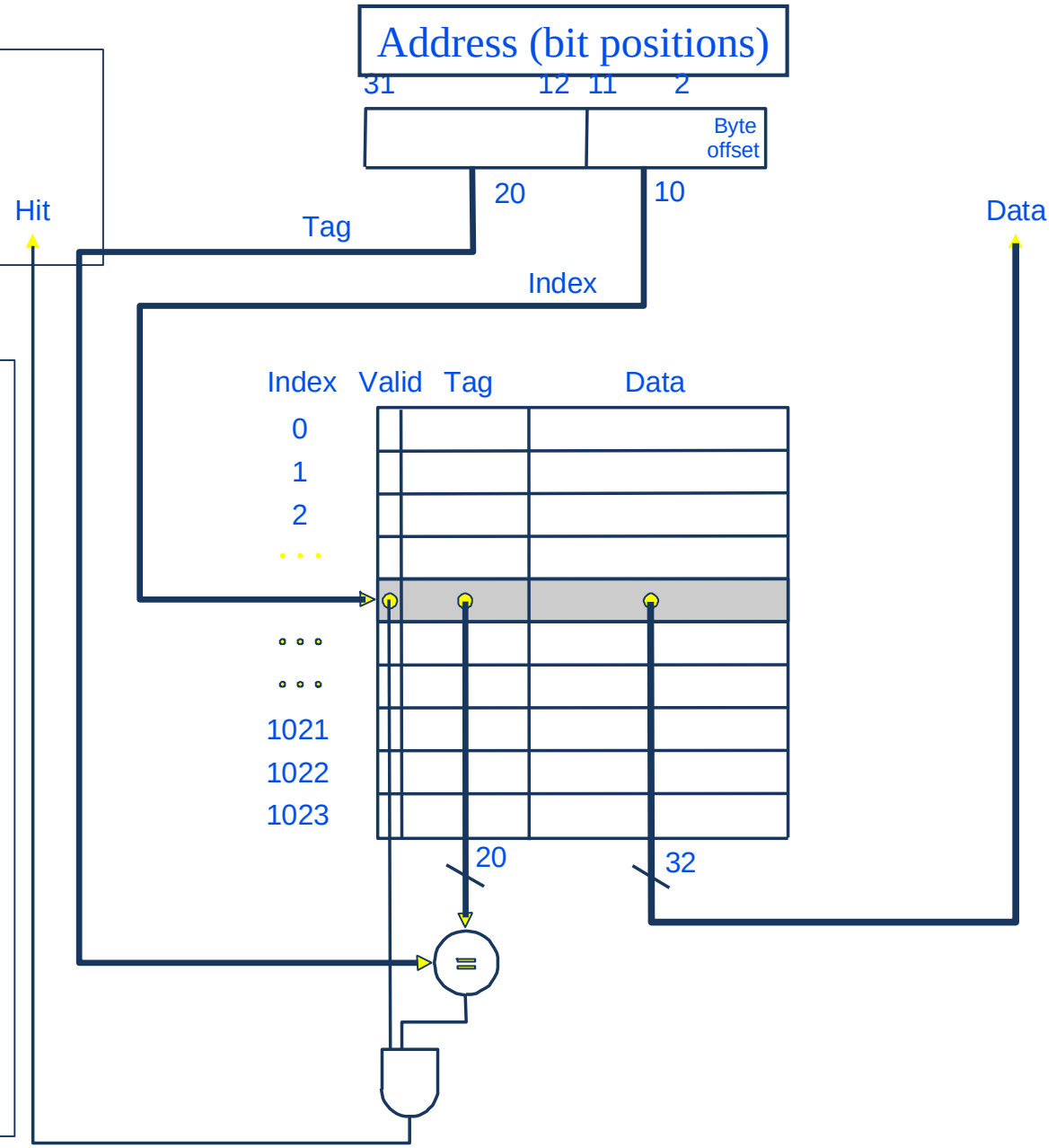
- ***block***: minimum unit of data to move between levels
- ***hit***: data requested is in upper level
- ***miss***: data requested is not in upper level
- ***hit rate***: fraction of memory accesses that are hits (i.e., found at upper level)
- ***miss rate***: fraction of memory accesses that are not hits
  - miss rate = 1 – hit rate
- ***hit time***: time to determine if the access is indeed a hit + time to access and deliver the data from the upper level to the CPU
- ***miss penalty***: time to determine if the access is a miss + time to replace block at upper level with corresponding block at lower level + time to deliver the block to the CPU

DIRECT MAPPED CACHE

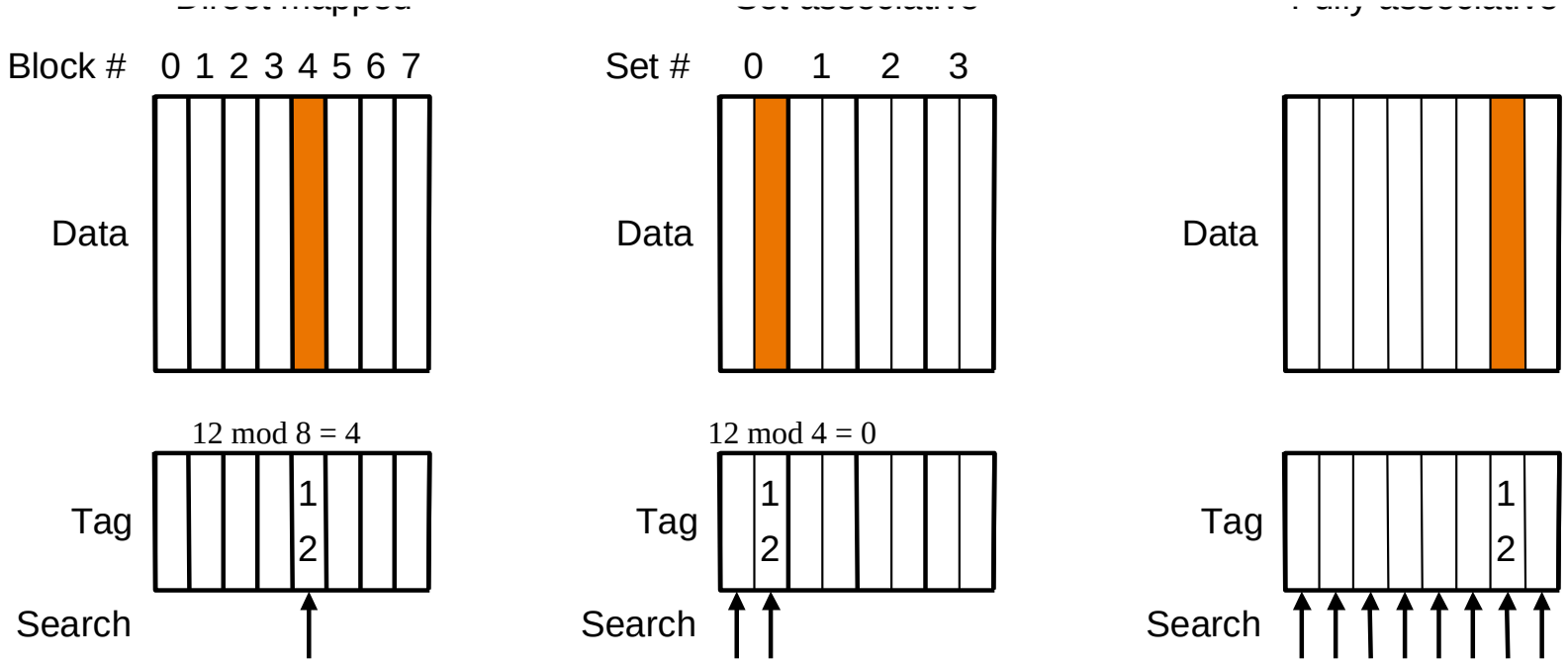


ΑΝΑΓΝΩΣΗ CACHE

- CACHE HIT
  - Ανάγνωση από CACHE
- CACHE MISS
  - Ανάγνωση από RAM
  - Αποθήκευση στην cache
  - Ανανέωση Valid & Tag



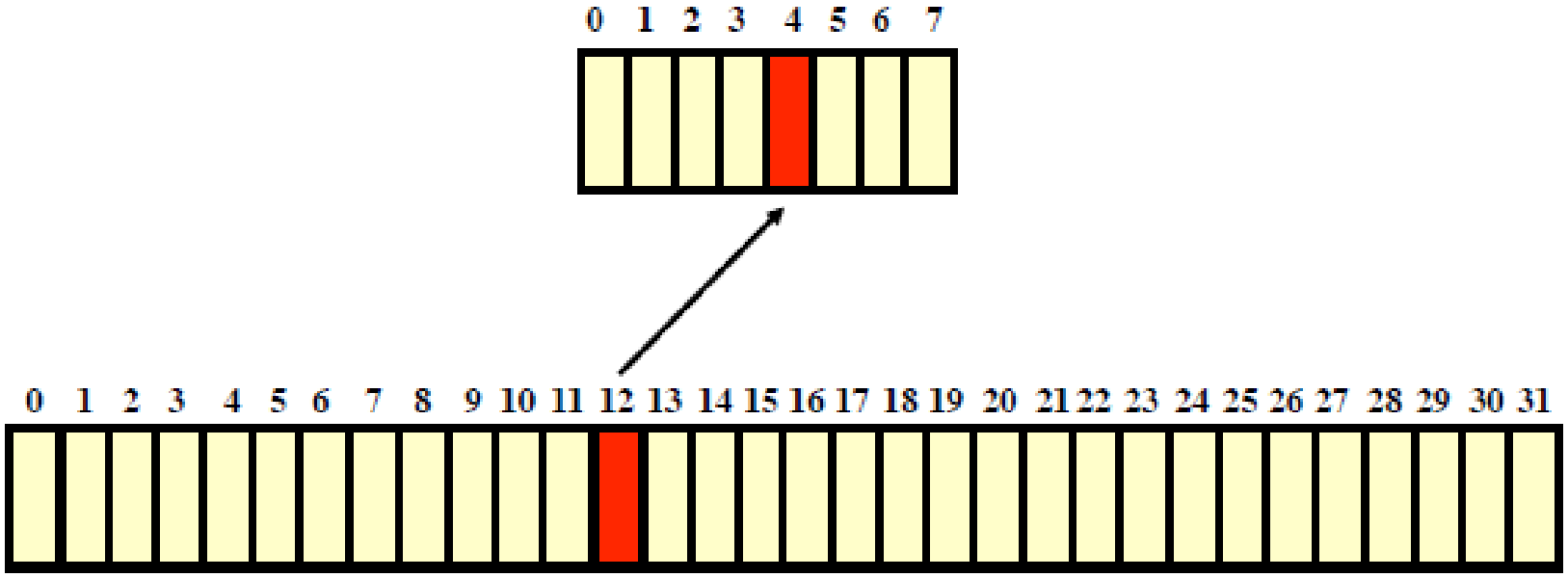
Προσεταιριστική Κρυφή Μνήμη



Θέση ενός block μνήμης με address 12 σε μία cache με 8 blocks με διαφορετικούς βαθμούς προσεταιριστικότητας

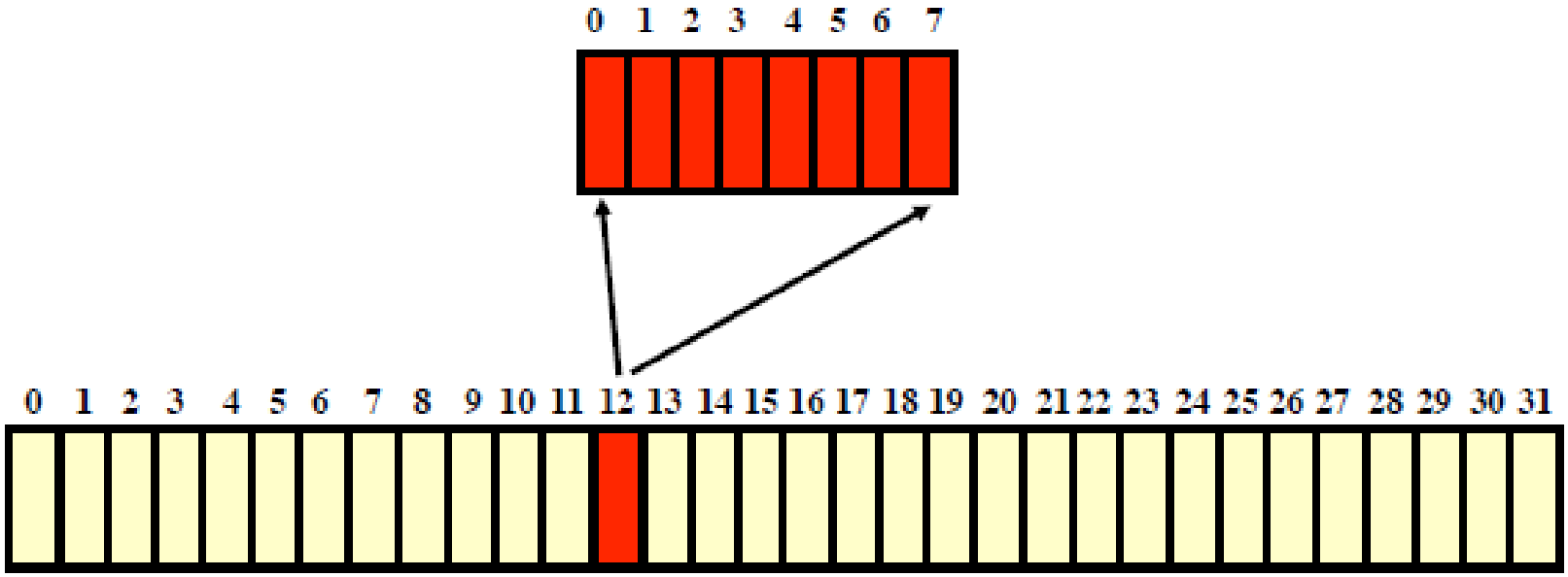
Άμεσης Απεικόνισης (Direct Mapped)

- Κάθε block απεικονίζεται ακριβώς σε 1 θέση της cache
- **Cache location = (block address) MOD (# blocks in cache)**



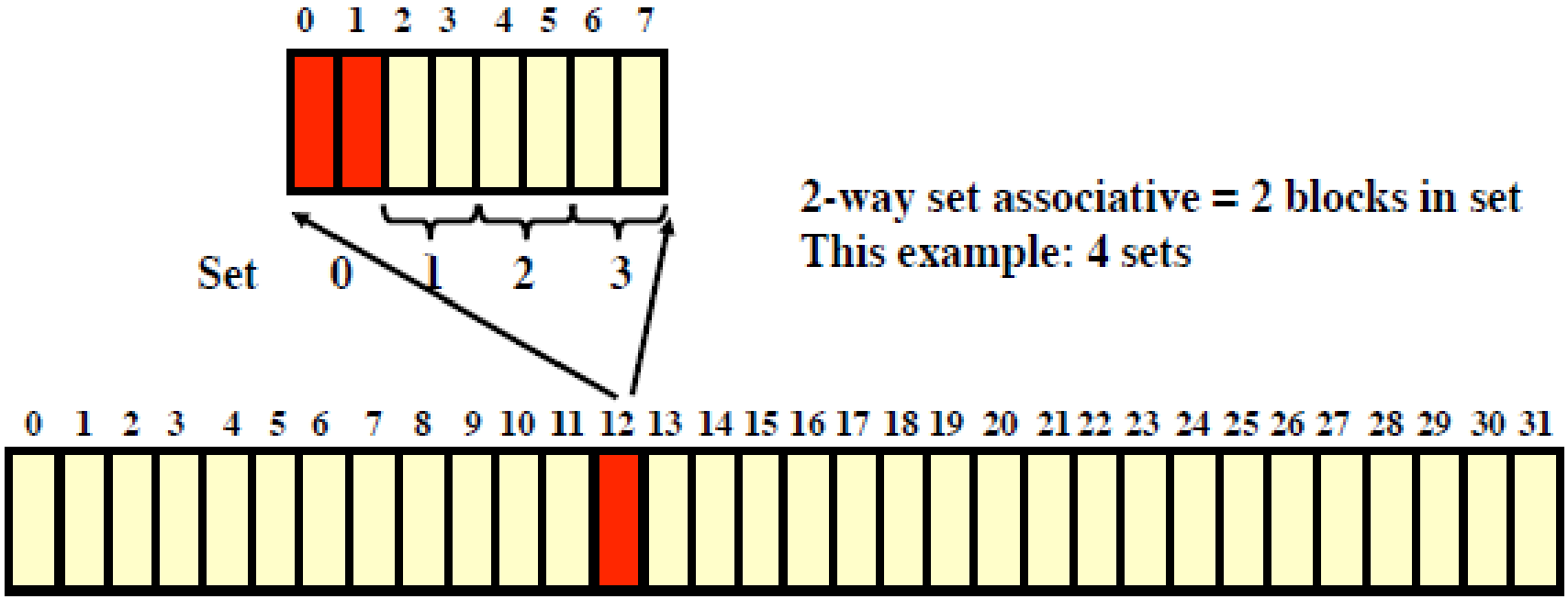
Πλήρως Προσεταιριστική

- Κάθε block απεικονίζεται σε οποιαδήποτε
- **Cache location = any**

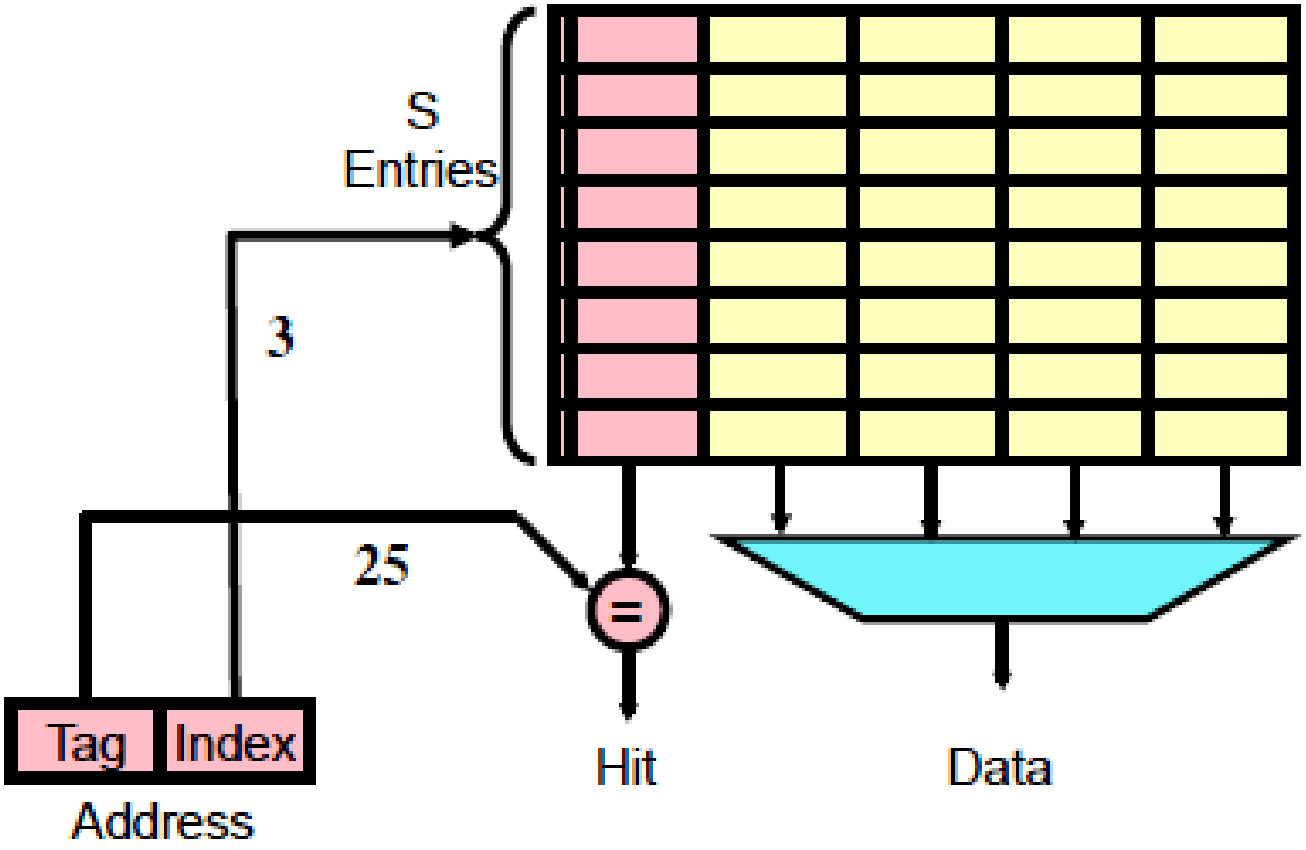


Προσεταιριστική με Σετ

- Κάθε block απεικονίζεται σε ένα υπο-σετ της cache
- **Set selection = (block address) MOD (# sets in cache)**



Εύρεση θέσης σε Direct Mapped



With cache capacity = 8 blocks

Εύρεση θέσης σε 2-way Set Associative

S - sets

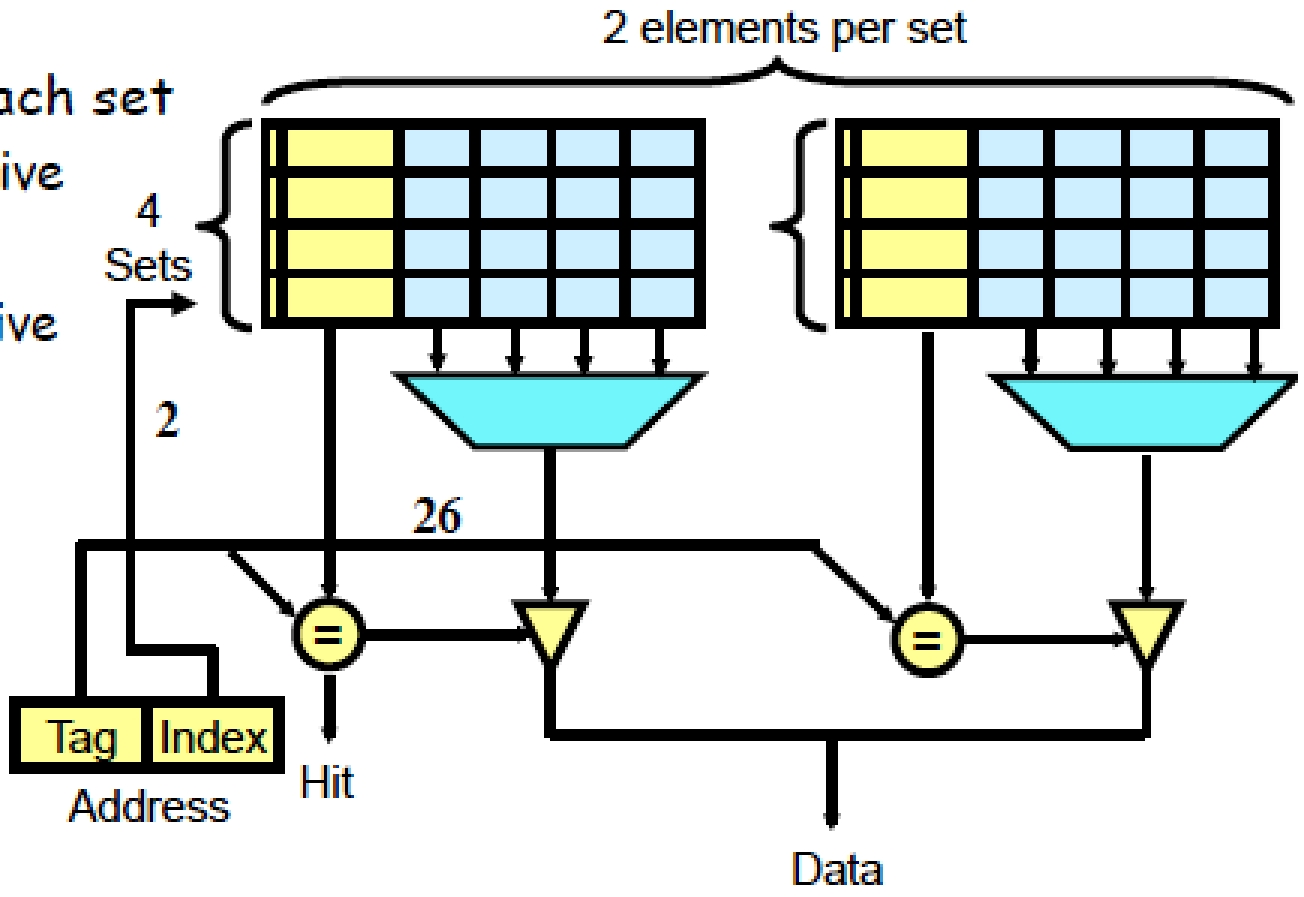
A - elements in each set

A-way associative

S=4, A=2

2-way associative

8-entry cache



Τι συμβαίνει όταν εκτελείται ένα SW?

- Need to keep cache consistent with main memory

– Reads are easy - no modifications

– Writes are harder - when do we update main memory?

- **Write-Through**

– On cache write - always update main memory as well

– Use a write buffer to stockpile writes to main memory for speed

- **Write-Back**

– On cache write - remember that block is modified (dirty bit)

– Update main memory when dirty block is replaced

– Sometimes need to flush cache (I/O, multiprocessing)

- **Write-Allocate**

– Bring written block into cache

– Update word in block

– Anticipate further use of block

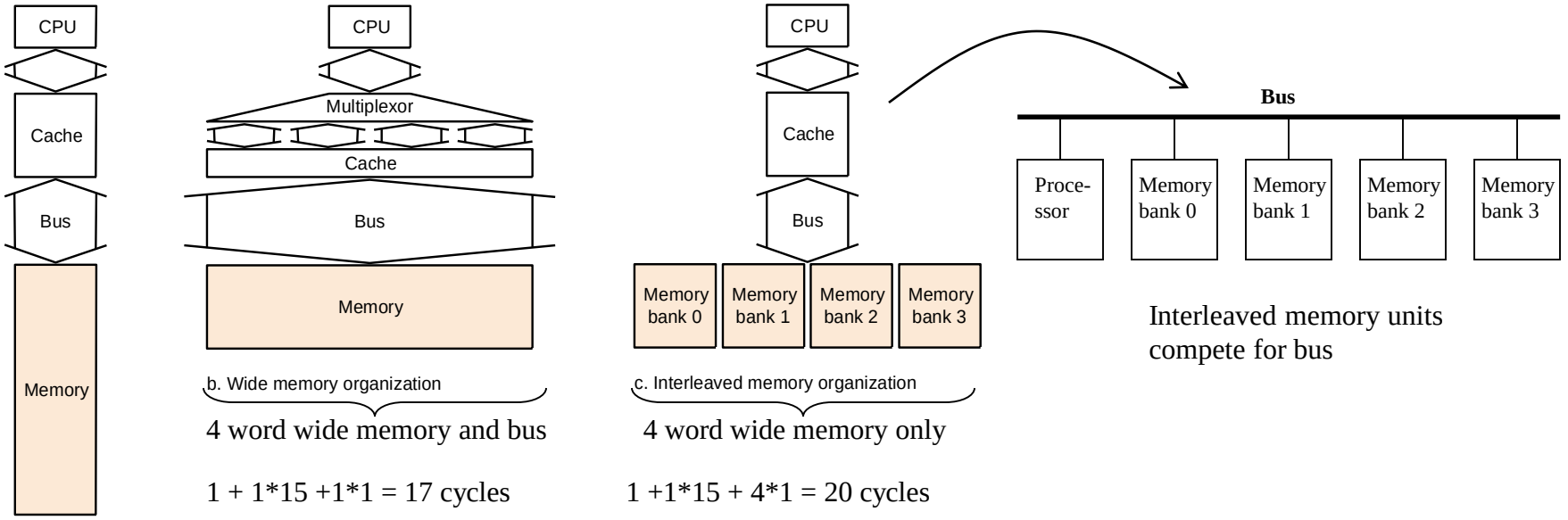
- **No-write Allocate**

– Main memory is updated

– Cache contents unmodified

Βελτιστοποίηση Απόδοσης της Cache

- Assume:
  - cache block of 4 words
  - 1 clock cycle to send address to memory address buffer (1 bus trip)
  - 15 clock cycles for each memory data access
  - 1 clock cycle to send data to memory data buffer (1 bus trip)



1 + 4*15 + 4*1 = 65 cycles

Miss penalties

Χρήσιμοι σύνδεσμοι

- http://www.ecs.umass.edu/ece/koren/architecture/Cache/tutorial.html

Απόδοση Cache

MissRate=MemMisses/MemAccess

- Miss-oriented Approach to Memory Access:

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

  - $CPI_{Execution}$ includes ALU and Memory instructions
- Separating out Memory component entirely
  - AMAT = Average Memory Access Time
  - CPI_{ALUOps} does not include memory instructions

$$CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT \right) \times CycleTime$$

$$AMAT = HitTime + MissRate \times MissPenalty$$
$$= (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}) +$$
$$(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})$$

Παράδειγμα

- clock per instruction (CPI) is 1.0 when all memory accesses hit in the cache.
- The only data accesses are loads and stores: 50% of the instructions.
- miss penalty is 25 clock cycles and the miss rate is 2% (Unified instruction cache and data cache),
- how much faster would the computer be if all instructions and data were cache hit?

$$CPUtime = (CPUClockCycles + MemeoryStalls) \times ClockCycleTime$$
$$= (IC \times CPI + MemoryStalls) \times ClockCycleTime$$

When all instructions are hit:

$$CPUtime\_Ideal = (IC \times CPI + MemoryStalls) \times ClockCycleTime$$
$$= (IC \times 1.0 + 0) \times ClockCycleTime$$
$$= IC \times ClockCycleTime$$

In reality:

$$MemoryStallCycles = IC \times \frac{MemAccess}{Inst} \times MissRate \times MissPenalty$$
$$= IC \times (1 + 0.5) \times 0.02 \times 25 = IC \times 0.75$$

$$CPUtime\_Cache = (IC \times CPI + MemoryStalls) \times ClockCycleTime$$
$$= (IC \times 1.0 + IC \times 0.75) \times ClockCycleTime$$
$$= 1.75 \times IC \times ClockCycleTime$$