

## Κρυφές Μνήμες (Cache Memories)

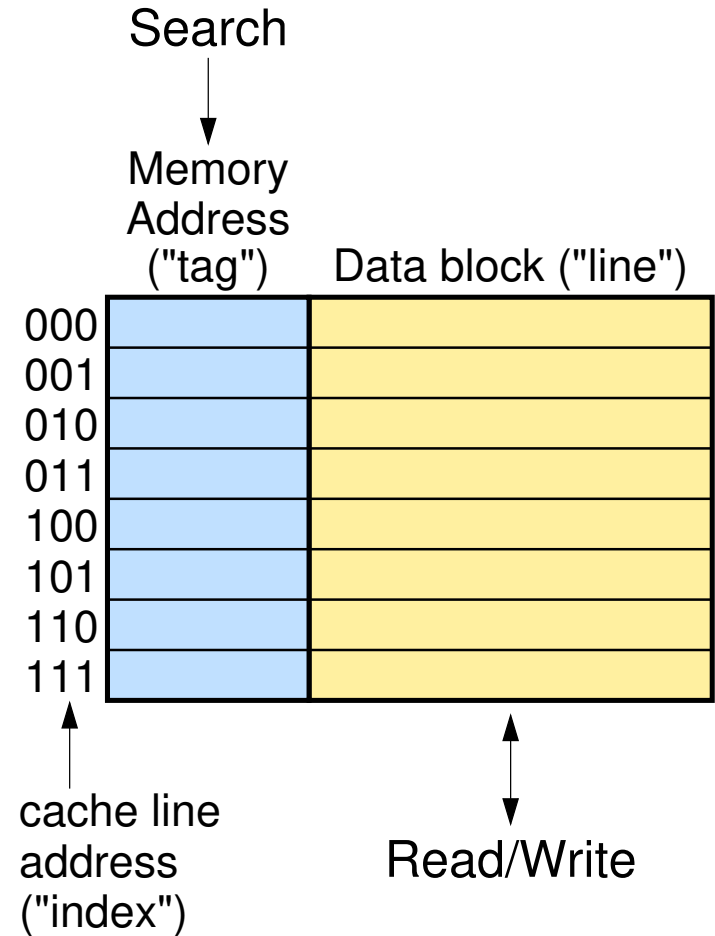
*11b (§11.1-10) – 14-21 Απριλίου 2021 – Μανόλης Κατεβαίνης*

© copyright University of Crete – <https://www.csd.uoc.gr/~hy225/21a/copyright.html>

except those marked as copyrighted by Morgan Kaufmann

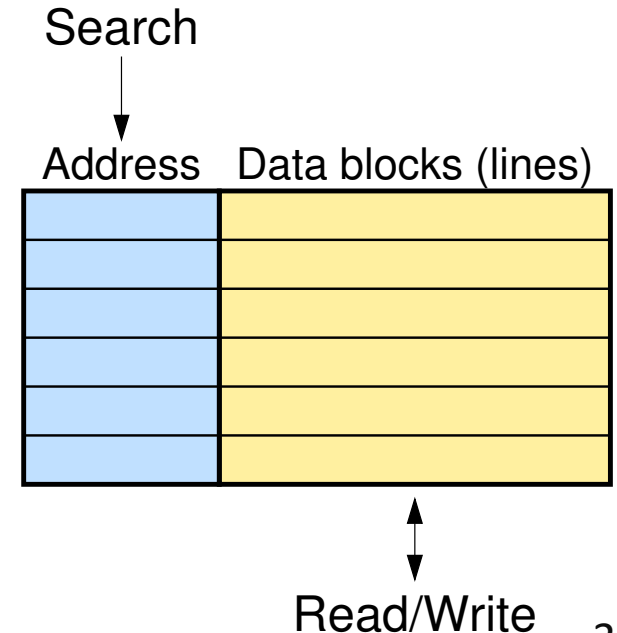
# Η γενική δομή μιάς Κρυφής Μνήμης

- Πλήθος «θέσεων» πολύ μικρότερο από χώρο διευθύνσεων μν.
  - cache “lines” or cache “blocks”
  - cache “index”: η θέση (διευθ.) μιάς γραμμής μέσα στην κρυφή μνήμη
- Κάθε «θέση» περιέχει δεδομένα (αντίγραφο γειτονιάς) από τη μνήμη, καθώς και τη διεύθυνση της μνήμης (“tag”) απ’ όπου την φέραμε αυτή τη γειτονιά
- Αναζήτηση βάσει Διευθ. Μνήμης



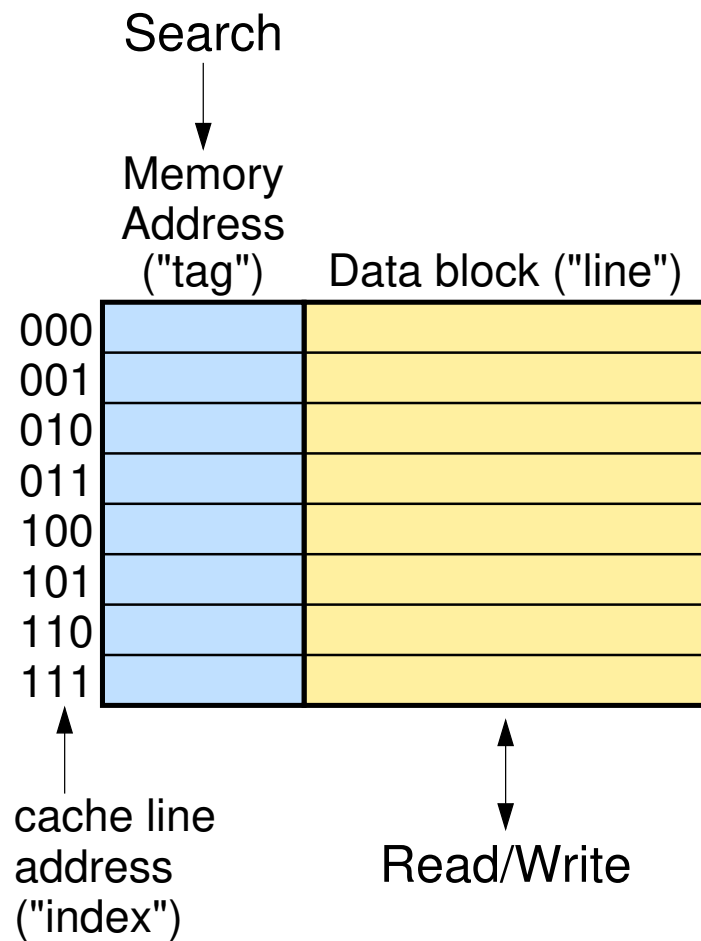
# Πού επιτρέπεται και πού θα ψάξουμε δοθείσα Διευθ.

- Δοθείσας μιάς διεύθυνσης μνήμης, πώς ψάχνουμε εάν και πού την έχουμε; Πού μπορεί να βρίσκεται;
- Επιτρέπεται οπουδήποτε; (“fully associative”)
  - υπερβολικά δαπανηρή αναζήτηση
  - περιττά μεγάλη ευελιξία
- Μόνον σε ορισμένα μέρη επιτρέπεται να την βάλουμε;
  - αρκεί να την ψάξουμε εκεί μόνον
  - όσο λιγότερα τα μέρη, τόσο ευκολότερο το ψάξιμο, αλλά και τόσο μικρότερη ευελιξία ποιόν «παλιόν» θα διώξουμε



# Μονοσήμαντη Απεικόνιση (Direct Mapped Caches)

- Η κάθε δοθείσα διεύθ. μνήμης μόνον σε μία θέση (cache index) επιτρέπεται να τοποθετηθεί
  - πολύ περιοριστικό
  - σημαντική απλοποίηση
  - ξεκινάμε με αυτό, και στη συνέχεια θα δούμε την κάπως πιο ευέλικτη και πιο συνηθισμένη οργάνωση
- Πώς προκύπτει η μία, μόνη θέση;
- $\text{Index} = \text{Hash}_f(\text{Mem. Addr.})$
- Συνάρτ. κατακερματισμού: αρκεί απλώς μερικά bits διευθ. – ποιά;;



# Hash function: ποιά bits?

Block Addr. Memory

|       |        |
|-------|--------|
| 00000 | Blue   |
| 00001 | Yellow |
| 00010 | Pink   |
| 00011 | Grey   |
| 00100 | Orange |
| 00101 | Cyan   |
| 00110 | Red    |
| 00111 | Green  |
| 01000 | Blue   |
| 01001 | Yellow |
| 01010 | Pink   |
| 01011 | Grey   |
| 01100 | Orange |
| 01101 | Cyan   |
| 01110 | Red    |
| 01111 | Green  |
| 10000 | Blue   |
| 10001 | Yellow |
| 10010 | Pink   |
| 10011 | Grey   |
| 10100 | Orange |
| 10101 | Cyan   |
| 10110 | Red    |
| 10111 | Green  |
| 11000 | Blue   |
| 11001 | Yellow |
| 11010 | Pink   |
| 11011 | Grey   |
| 11100 | Orange |
| 11101 | Cyan   |
| 11110 | Red    |
| 11111 | Green  |

Block Addr. Memory

|       |        |
|-------|--------|
| 00000 | Blue   |
| 00001 | Blue   |
| 00010 | Blue   |
| 00011 | Blue   |
| 00100 | Yellow |
| 00101 | Yellow |
| 00110 | Yellow |
| 00111 | Yellow |
| 01000 | Pink   |
| 01001 | Pink   |
| 01010 | Pink   |
| 01011 | Pink   |
| 01100 | Grey   |
| 01101 | Grey   |
| 01110 | Grey   |
| 01111 | Grey   |
| 10000 | Orange |
| 10001 | Orange |
| 10010 | Orange |
| 10011 | Orange |
| 10100 | Cyan   |
| 10101 | Cyan   |
| 10110 | Cyan   |
| 10111 | Cyan   |
| 11000 | Red    |
| 11001 | Red    |
| 11010 | Red    |
| 11011 | Red    |
| 11100 | Green  |
| 11101 | Green  |
| 11110 | Green  |
| 11111 | Green  |

cache line  
address  
("index")

Cache

|     |        |
|-----|--------|
| 000 | Blue   |
| 001 | Yellow |
| 010 | Pink   |
| 011 | Grey   |
| 100 | Orange |
| 101 | Cyan   |
| 110 | Red    |
| 111 | Green  |

Hash on  
LS  
Address bits

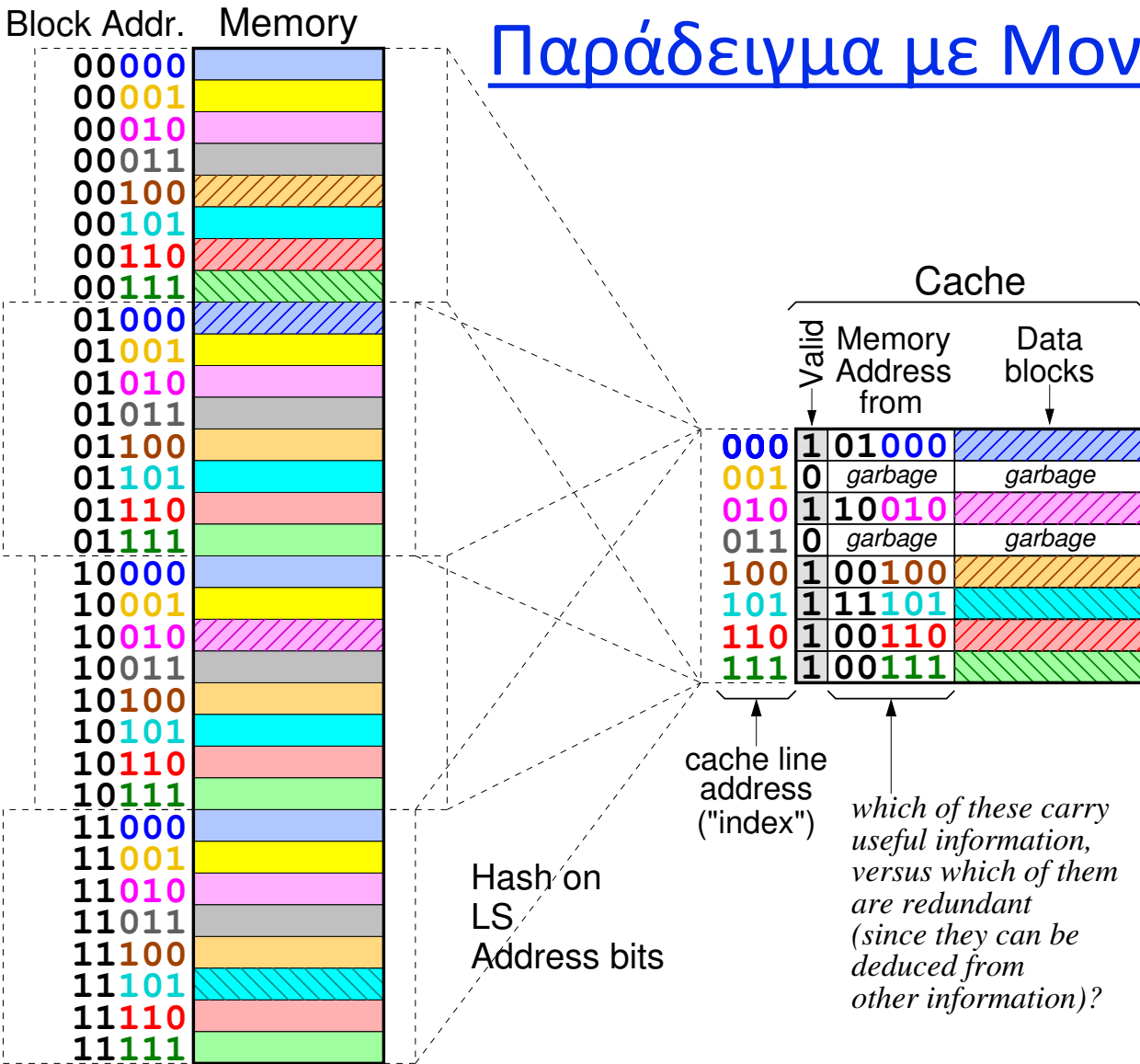
• neighbour blocks  
do not collide

Hash on  
MS  
Address bits

• neighbour blocks  
will usually collide

# Παράδειγμα με Μονοσημ. Απεικόνιση

- Κάθε block μνήμης μόνο σε μία θέση επιτρέπεται να μπει εάν/όταν έλθει στην κρυφή μν. (χρώμα)
- Κάθε block μέσα στην κρυφή μνήμη συνοδεύεται από πληροφ. διεύθυνσης: τίνος block μνήμης αποτελεί αντίγραφο;
- *Validity bit* για όταν κανένα block μνήμης δεν βρισκ. τώρα εδώ



Block Addr. Memory

|       |        |
|-------|--------|
| 00000 | Blue   |
| 00001 | Yellow |
| 00010 | Pink   |
| 00011 | Grey   |
| 00100 | Orange |
| 00101 | Cyan   |
| 00110 | Red    |
| 00111 | Green  |
| 01000 | Blue   |
| 01001 | Yellow |
| 01010 | Pink   |
| 01011 | Grey   |
| 01100 | Orange |
| 01101 | Cyan   |
| 01110 | Red    |
| 01111 | Green  |
| 10000 | Blue   |
| 10001 | Yellow |
| 10010 | Pink   |
| 10011 | Grey   |
| 10100 | Orange |
| 10101 | Cyan   |
| 10110 | Red    |
| 10111 | Green  |
| 11000 | Blue   |
| 11001 | Yellow |
| 11010 | Pink   |
| 11011 | Grey   |
| 11100 | Orange |
| 11101 | Cyan   |
| 11110 | Red    |
| 11111 | Green  |

# Προβλέψιμα bits Διεύθυνσης στην κάθε θέση

Cache

|     |        |
|-----|--------|
| 000 | Blue   |
| 001 | Yellow |
| 010 | Pink   |
| 011 | Grey   |
| 100 | Orange |
| 101 | Cyan   |
| 110 | Red    |
| 111 | Green  |

Index

Hash on  
LS  
Address bits

Which memory  
block addresses  
can ever be placed  
within a given  
cache location  
(cache index)?

00000  
01000  
10000  
11000

00010  
01010  
10010  
11010

00100  
01100  
10100  
11100

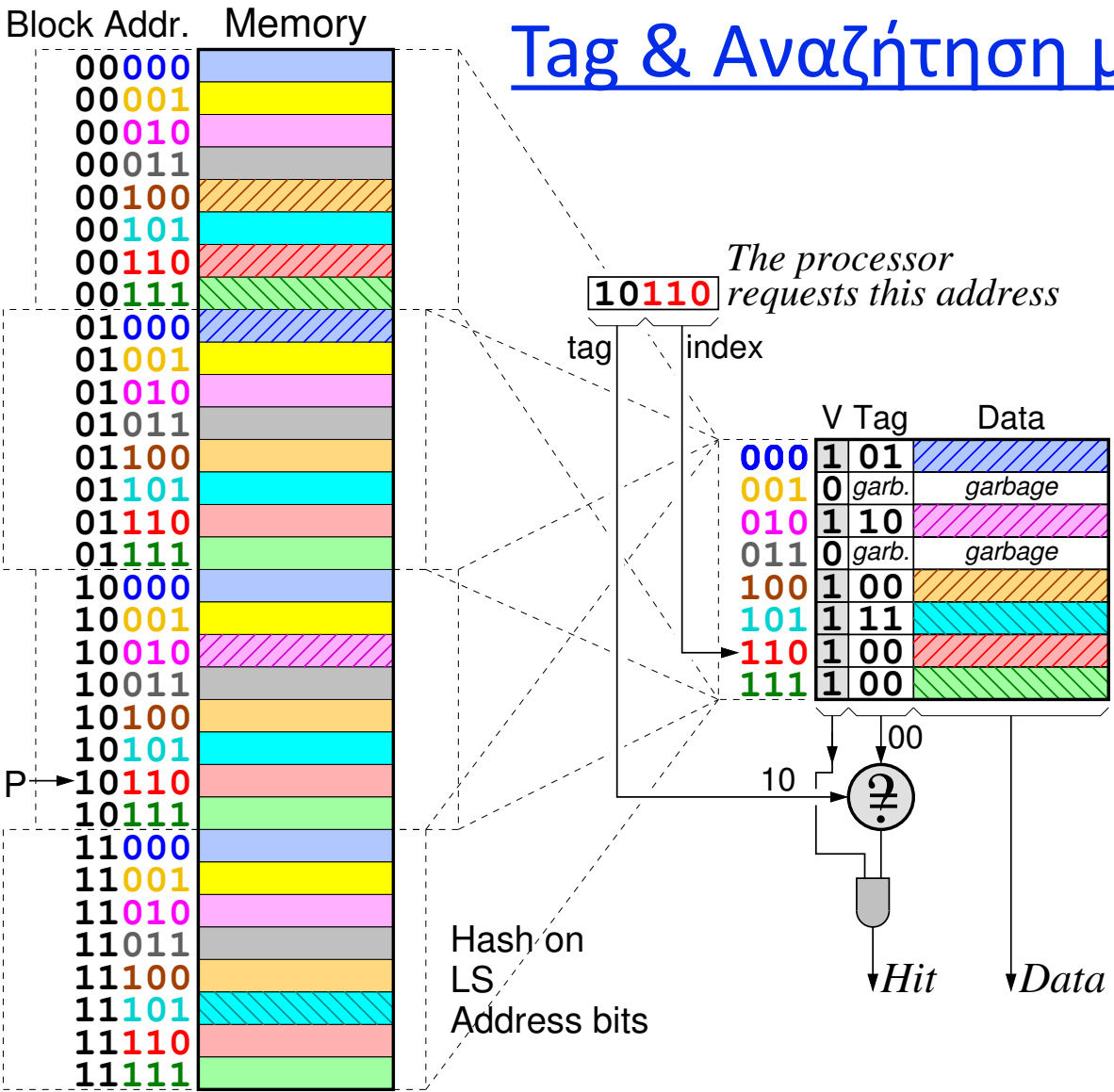
00110  
01110  
10110  
11110

00111  
01111  
10111  
11111

- Τα LS bits διεύθυν. μνήμης (= hash function = cache index), είναι γνωστά στην κάθε θέση της κρ. μνήμ., άρα περιτεύουν στην πληροφορία διεύθυν.

# Tag & Αναζήτηση με Μονοσ. Απεικόν.

- Tag: περιττό να περιέχει τα Index bits
- Δοθείσας Διεύθυνσης που ψάχνουμε:
- Με το Index από αυτήν, διαβάζουμε: Tag, Valid, και ταυτόχρονα τα Data
- Εάν  $V=1$  και  $Tag_{\psi\acute{\alpha}\chi\nu\nu\mu\epsilon} == Tag_{\beta\rho\rho\rho\rho\kappa\omicron\upsilon\mu\epsilon}$  τότε Ευστοχία (Hit)
- Έχουμε και τα Data έτοιμα, εάν ευστοχία





# Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

# Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

| Index | V | Tag | Data |
|-------|---|-----|------|
| 000   | N |     |      |
| 001   | N |     |      |
| 010   | N |     |      |
| 011   | N |     |      |
| 100   | N |     |      |
| 101   | N |     |      |
| 110   | N |     |      |
| 111   | N |     |      |

# Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 22        | 10 110      | Miss     | 110         |

| Index      | V        | Tag       | Data              |
|------------|----------|-----------|-------------------|
| 000        | N        |           |                   |
| 001        | N        |           |                   |
| 010        | N        |           |                   |
| 011        | N        |           |                   |
| 100        | N        |           |                   |
| 101        | N        |           |                   |
| <b>110</b> | <b>Y</b> | <b>10</b> | <b>Mem[10110]</b> |
| 111        | N        |           |                   |

# Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 26        | 11 010      | Miss     | 010         |

| Index | V | Tag | Data       |
|-------|---|-----|------------|
| 000   | N |     |            |
| 001   | N |     |            |
| 010   | Y | 11  | Mem[11010] |
| 011   | N |     |            |
| 100   | N |     |            |
| 101   | N |     |            |
| 110   | Y | 10  | Mem[10110] |
| 111   | N |     |            |

# Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 22        | 10 110      | Hit      | 110         |
| 26        | 11 010      | Hit      | 010         |

| Index | V | Tag | Data       |
|-------|---|-----|------------|
| 000   | N |     |            |
| 001   | N |     |            |
| 010   | Y | 11  | Mem[11010] |
| 011   | N |     |            |
| 100   | N |     |            |
| 101   | N |     |            |
| 110   | Y | 10  | Mem[10110] |
| 111   | N |     |            |

# Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 16        | 10 000      | Miss     | 000         |
| 3         | 00 011      | Miss     | 011         |
| 16        | 10 000      | Hit      | 000         |

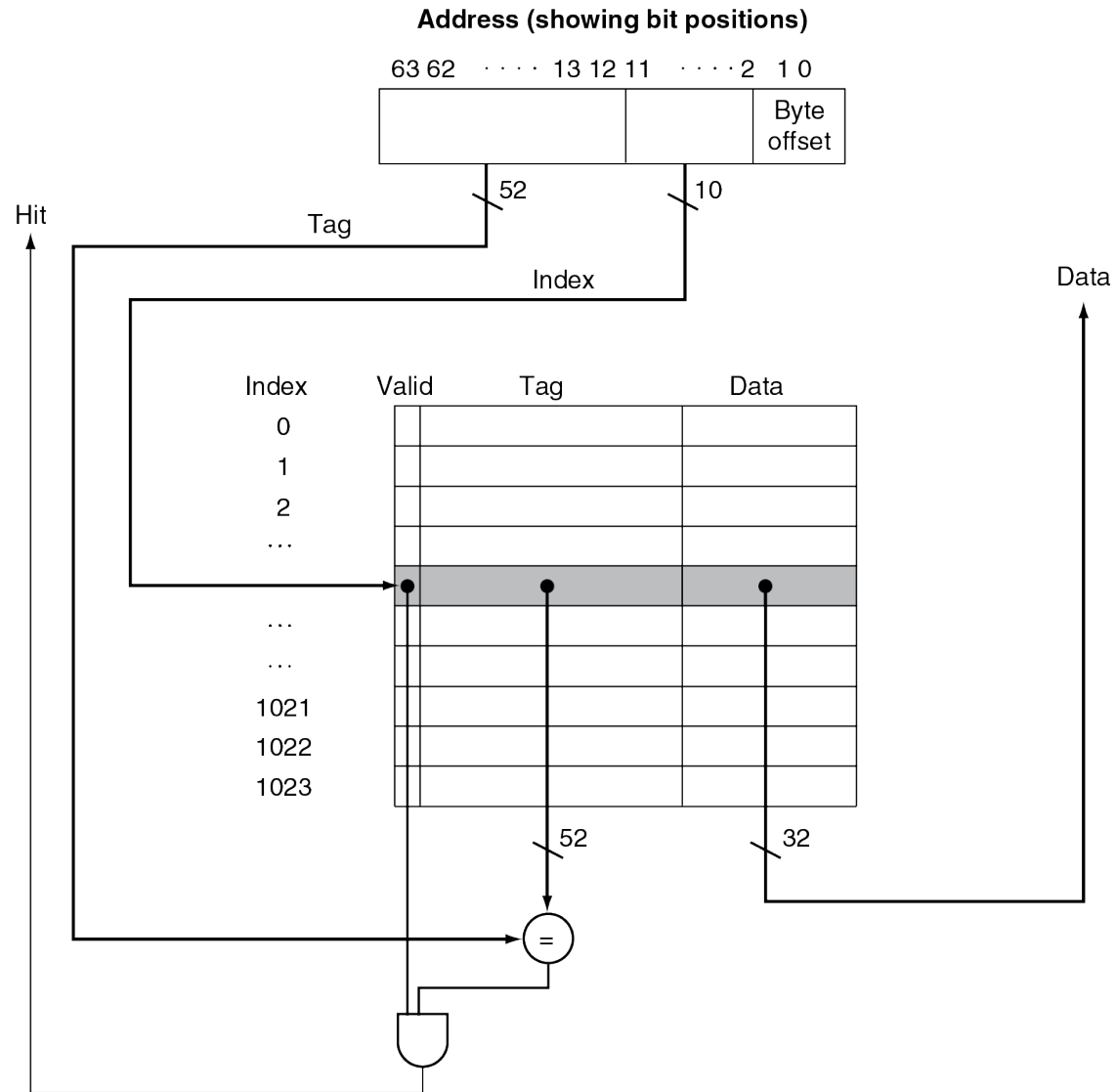
| Index      | V        | Tag       | Data              |
|------------|----------|-----------|-------------------|
| <b>000</b> | <b>Y</b> | <b>10</b> | <b>Mem[10000]</b> |
| 001        | N        |           |                   |
| 010        | Y        | 11        | Mem[11010] ←      |
| <b>011</b> | <b>Y</b> | <b>00</b> | <b>Mem[00011]</b> |
| 100        | N        |           |                   |
| 101        | N        |           |                   |
| 110        | Y        | 10        | Mem[10110]        |
| 111        | N        |           |                   |

# Cache Example

| Word addr | Binary addr | Hit/miss | Cache block |
|-----------|-------------|----------|-------------|
| 18        | 10 010      | Miss     | 010         |

| Index      | V        | Tag       | Data              |
|------------|----------|-----------|-------------------|
| 000        | Y        | 10        | Mem[10000]        |
| 001        | N        |           |                   |
| <b>010</b> | <b>Y</b> | <b>10</b> | <b>Mem[10010]</b> |
| 011        | Y        | 00        | Mem[00011]        |
| 100        | N        |           |                   |
| 101        | N        |           |                   |
| 110        | Y        | 10        | Mem[10110]        |
| 111        | N        |           |                   |

# Address Subdivision





# Average Access Time

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

$$AMAT = (\text{Hit rate}) \times (\text{Hit time}) + (\text{Miss rate}) \times (\text{Miss time})$$

$$\text{Miss time} = \text{Hit time} + \text{Miss penalty}$$

$$\text{Hit rate} + \text{Miss rate} = 1$$

# Measuring Cache Performance

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

$$\begin{aligned}
 & \text{Memory stall cycles} \\
 &= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty} \\
 &= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}
 \end{aligned}$$

# 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

I-Cache  
Accesses  
per Instruction  
= 1

## ■ Miss cycles per instruction

- I-cache:  $0.02 \times 100 = 2$
- D-cache:  $0.36 \times 0.04 \times 100 = 1.44$

Data Cache  
Accesses  
per Instruction

## ■ Actual CPI = $2 + 2 + 1.44 = 5.44$

- Ideal CPU is  $5.44/2 = 2.72$  times faster

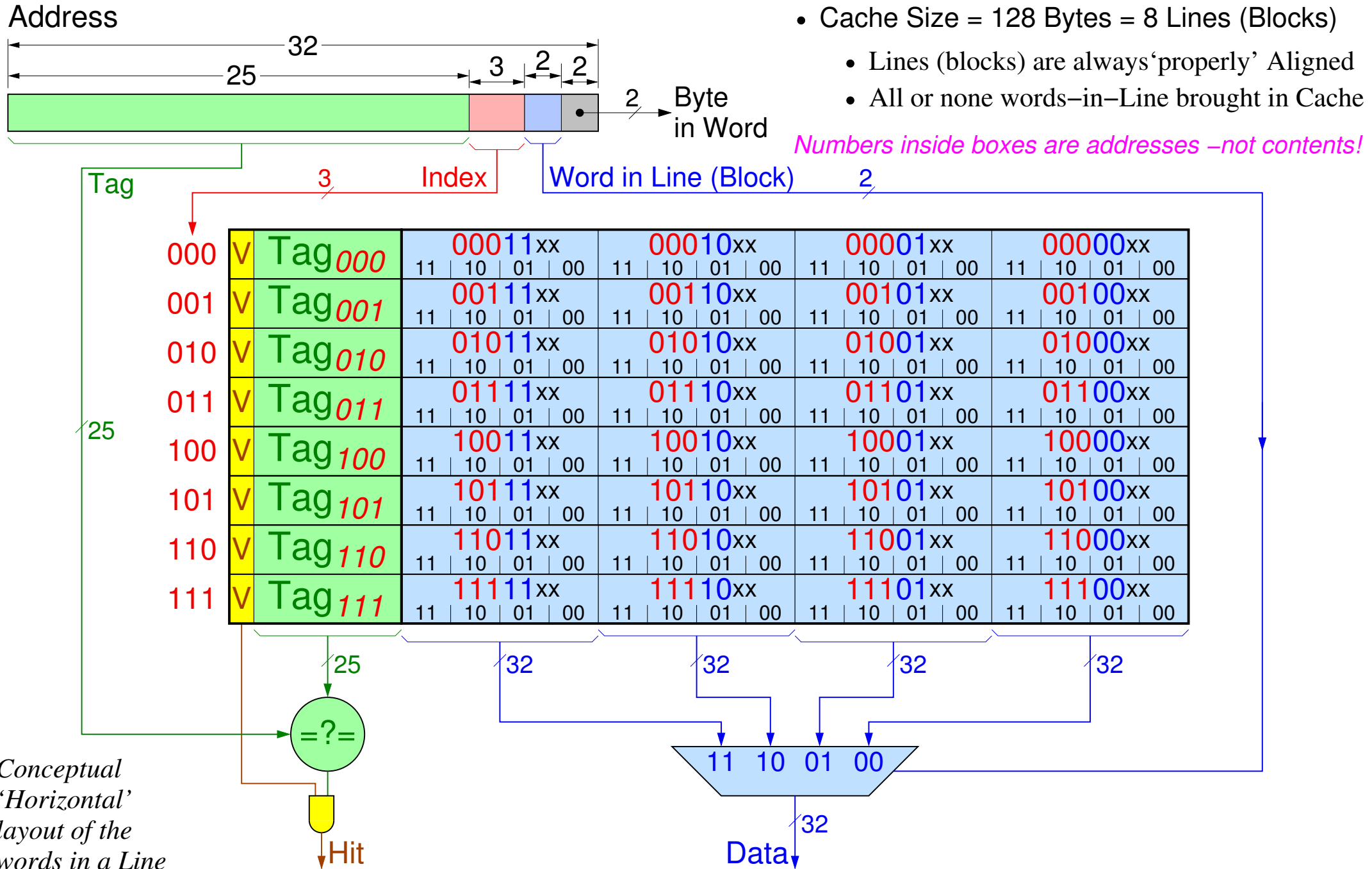
# Performance Summary

- 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

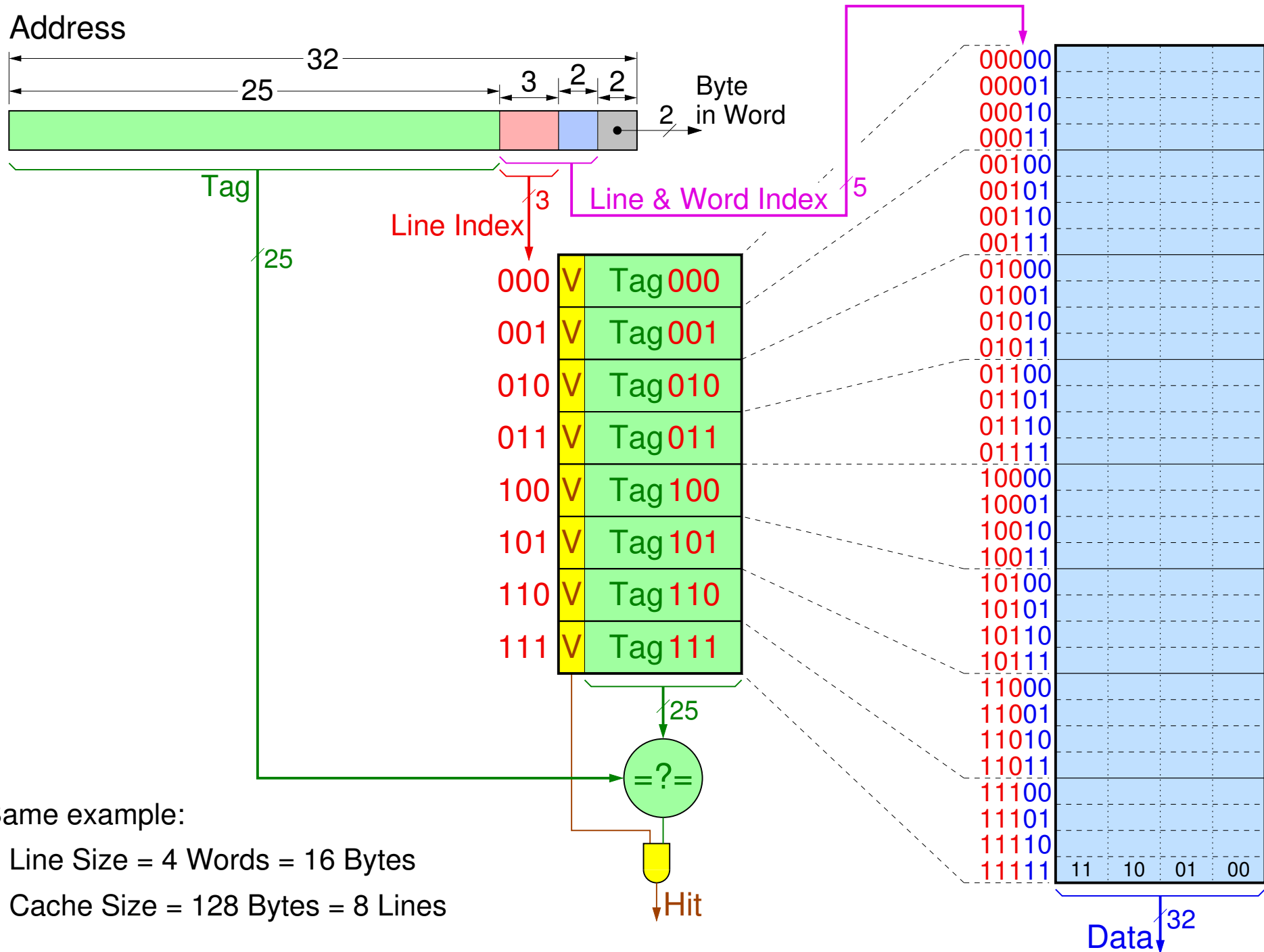
# Increased Line (Block) Size, to exploit Spatial Locality

- Example:
- Line (Block) Size = 4 Words = 16 Bytes
  - Cache Size = 128 Bytes = 8 Lines (Blocks)

- Lines (blocks) are always 'properly' Aligned
- All or none words-in-Line brought in Cache



# 'Vertical' Layout of the Words in a Line(Block)



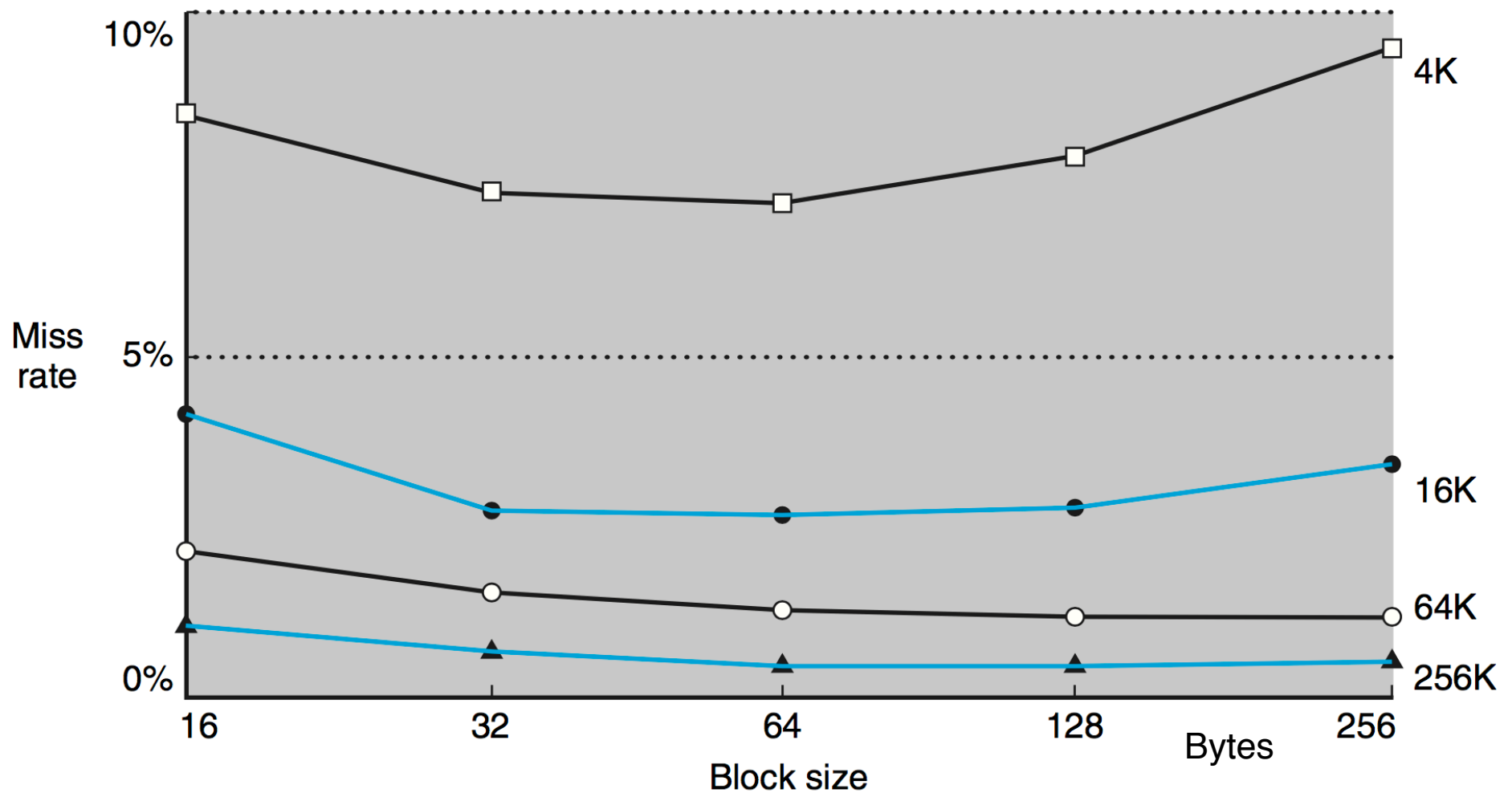
Same example:

- Line Size = 4 Words = 16 Bytes
- Cache Size = 128 Bytes = 8 Lines

# Block Size Considerations

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

They also reduce the number of Tags, hence speed up Tag look-up



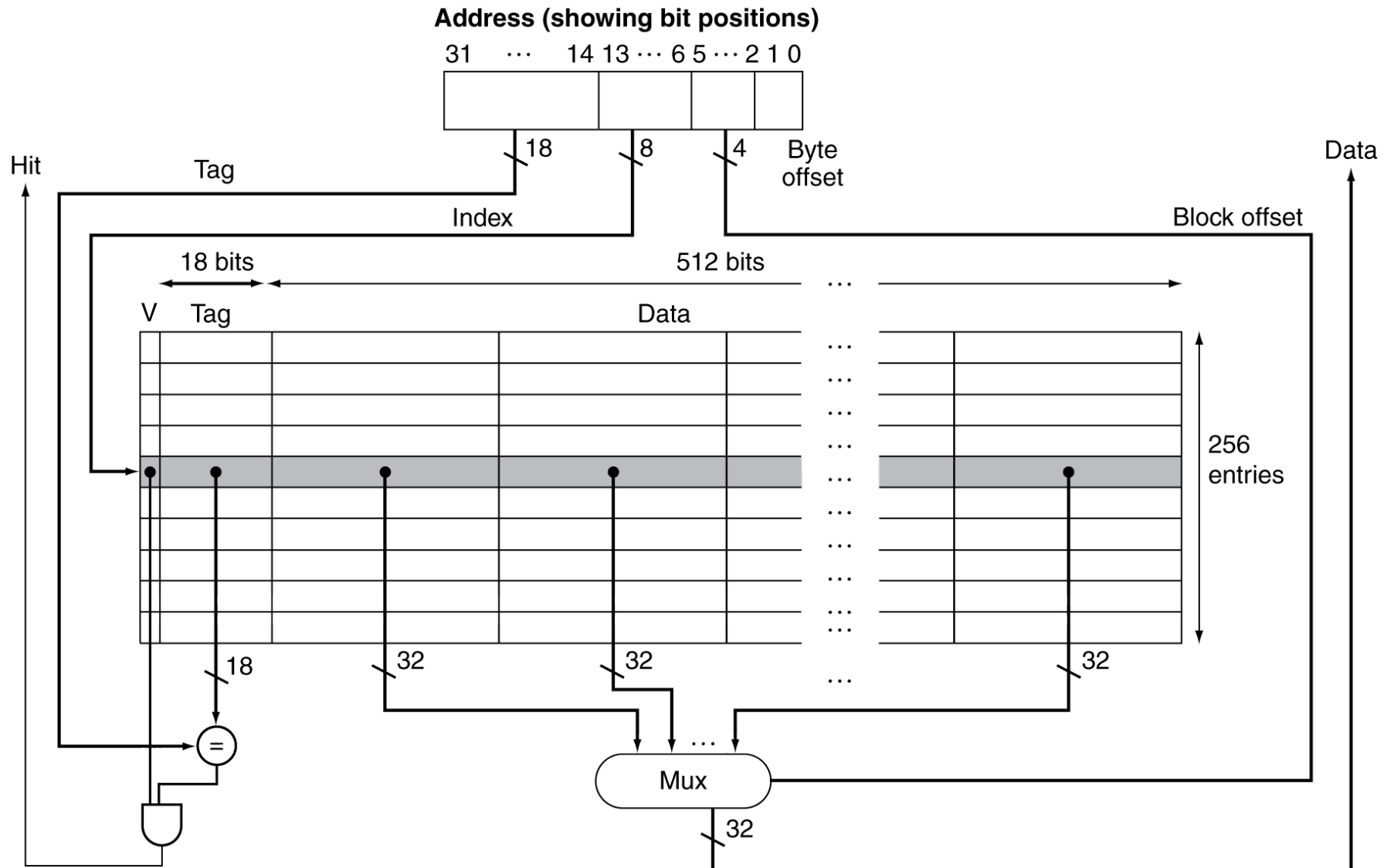
**FIGURE 5.11 Miss rate versus block size.** Note that the miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. (This figure is independent of associativity, discussed soon.) Unfortunately, SPEC CPU2000 traces would take too long if block size were included, so these data are based on SPEC92.



# Example: Intrinsicity 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
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

# Example: Intrinsic FastMATH



# Cache Misses

- On cache hit, CPU proceeds normally
  - On cache miss
    - Stall the CPU pipeline (on I-cache miss, can also let the rest of the pipeline proceed to completion)
    - Fetch block from next level of hierarchy
    - Instruction cache miss
      - Restart instruction fetch
    - Data cache miss
      - Complete data access
- Out-of-Order Pipelines do not stall the pipeline, but look for subsequent instructions that do not depend on the miss data; their D-cache must support one or more outstanding misses

# Write-Through

## Ταυτόχρονη Εγγραφή

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI =  $1 + 0.1 \times 100 = 11$
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Write-Combining:  
sequential accesses  
to DRAM take shorter  
for subsequent words  
beyond the first one

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

Main Memory is inconsistent with Cache

We will revisit this when talking about I/O, then about multicores...

# Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don't fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Several modern processors allow software to control this policy on a per-page granularity (via page-table flag)

# Associative Caches

Μερικώς Προσεταιριστικές  
Κρυφές Μνήμες

- Fully associative

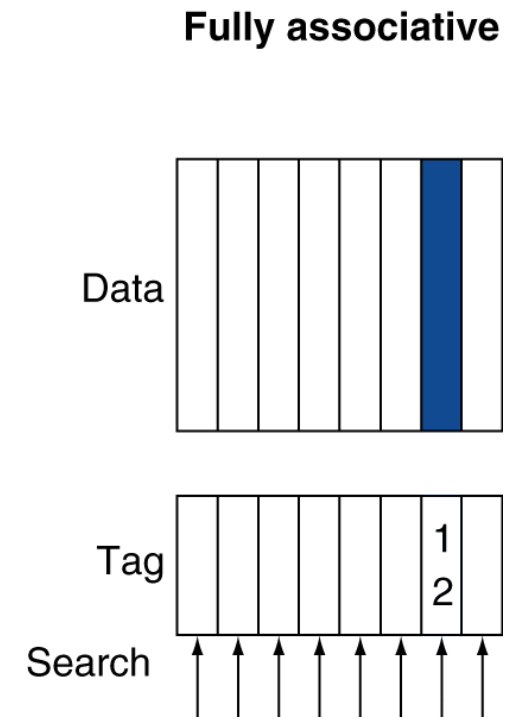
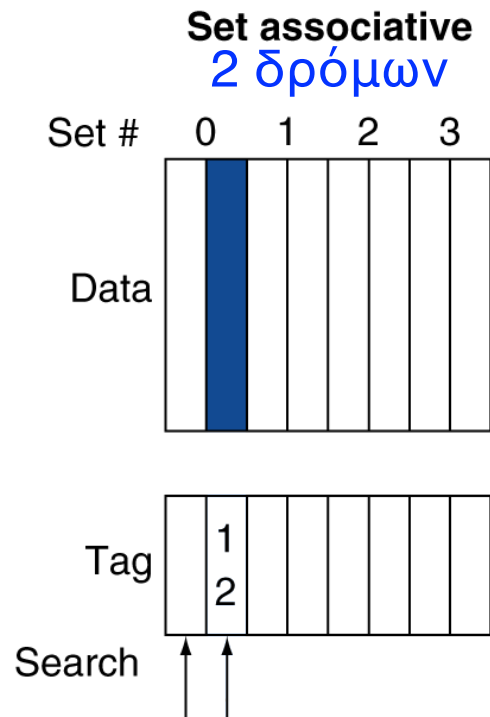
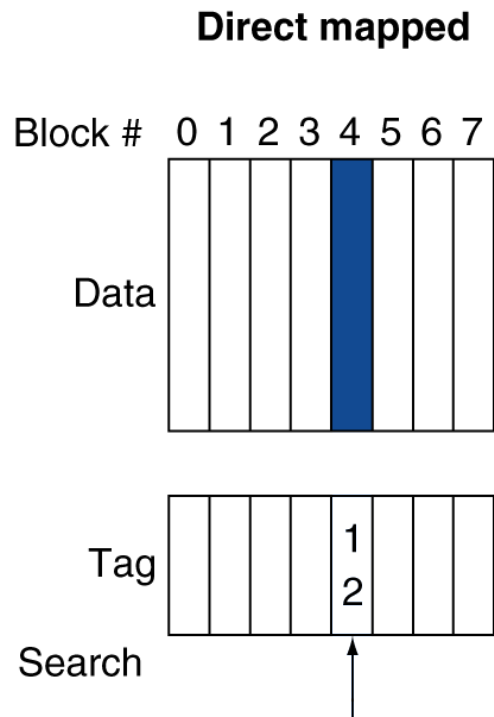
- Allow a given block to go in any cache entry
- Requires all entries to be searched at once
- Comparator per entry (expensive)

- *n*-way set associative

- Each set contains *n* entries
- Block number determines which set
  - (Block number) modulo (#Sets in cache)
- Search all entries in a given set at once
- *n* comparators (less expensive)

"Block number" = MS part  
of memory address, after  
discarding the LS addr. bits  
corresponding to  
byte-within-block  
(line) offset

# 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 (Line) number

| 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] |   |

Χρόνος

# Associativity Example

Block access seq.: 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] |       |  |

Χρόνος

Set number, Block (Line) within it

## ■ 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] |  |

Χρόνος

# How Much Associativity

- Increased associativity decreases miss rate

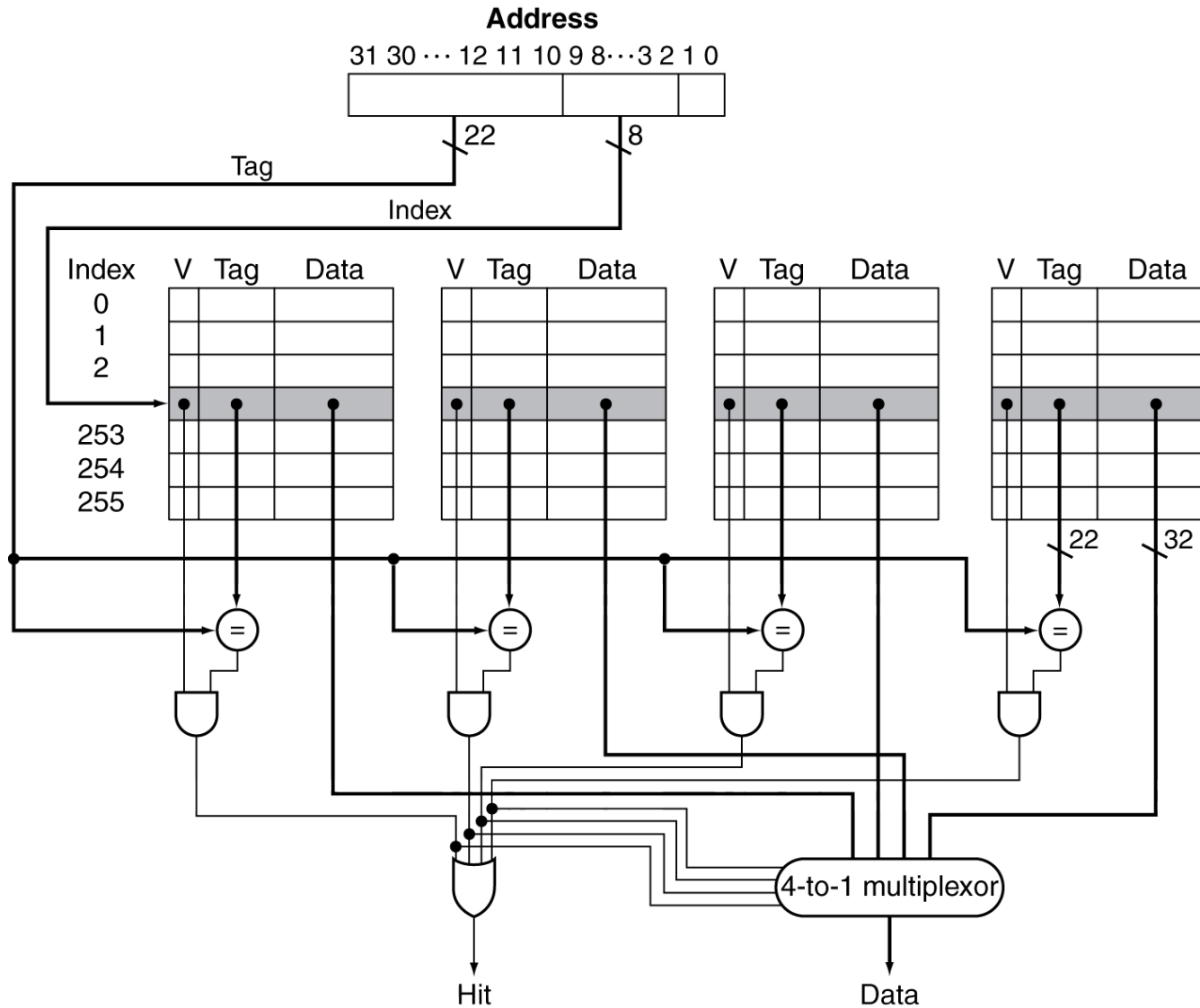
- But with diminishing returns

- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000

- 1-way: 10.3%
- 2-way: 8.6%
- 4-way: 8.3%
- 8-way: 8.1%

In modern processors, the main cases of associativity wider than ~4 are:  
(1) in L2/L3 caches;  
(2) to overcome the problem of interaction between page size and cache size in physical-address caches where we need the TLB to operate in parallel with Tag lookup —see next chapter, on virt. mem.

# Set Associative Cache Organization



# Replacement Policy

Πώς να προβλέψουμε το μέλλον;;  
Συχνά, το πρόσφατο παρελθόν αποτελεί καλή ένδειξη για το προσεχές μέλλον!...

- 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

# 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 L-2 cache misses
- Some high-end systems include L-3 cache

# Multilevel Cache Example

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



# Example (cont.)

- Now add L-2 cache

- Access time = 5ns

- Global miss rate to main memory = 0.5%

per Instruction

- Primary miss with L-2 hit

- Penalty =  $5\text{ns}/0.25\text{ns} = 20$  cycles

- Primary miss with L-2 miss

- Extra penalty = ~~500~~<sup>400</sup> cycles

- $\text{CPI} = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$

- Performance ratio =  $9/3.4 = 2.6$

Relative to ALL accesses to the entire cache hierarchy — not just the accesses to the L2 cache

# Multilevel Cache Considerations

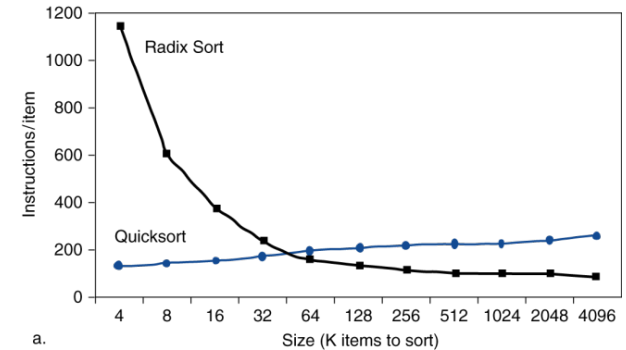
- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - & often fewer ways (smaller associativity)
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

# Interactions with Advanced CPUs

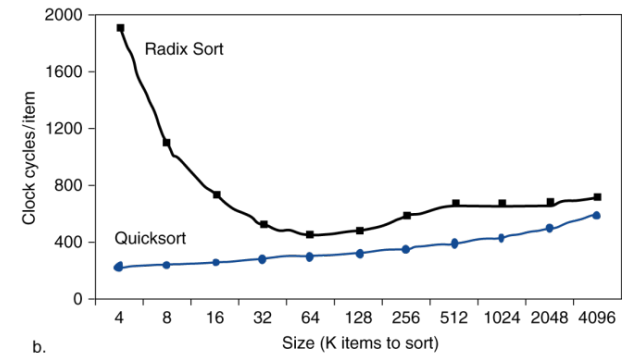
- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

# Interactions with Software

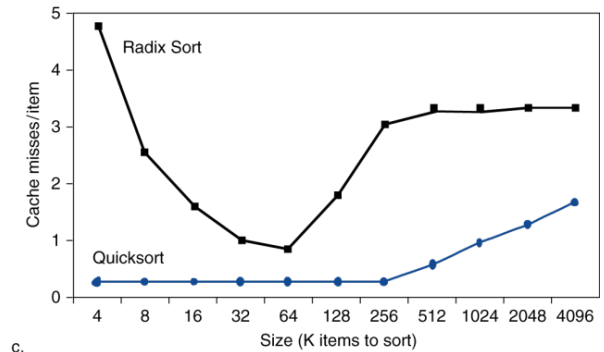
- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access



a.



b.



c.