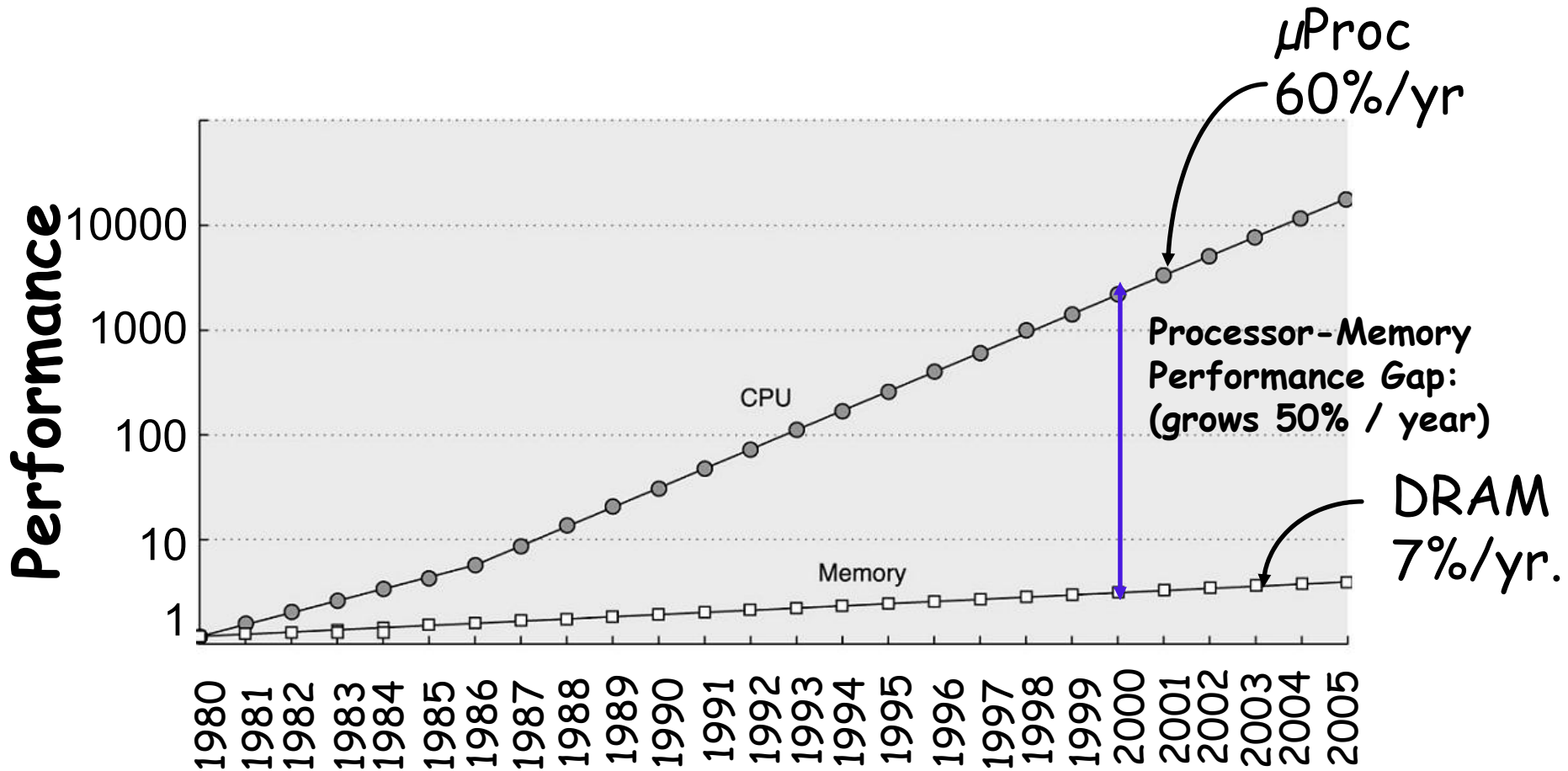
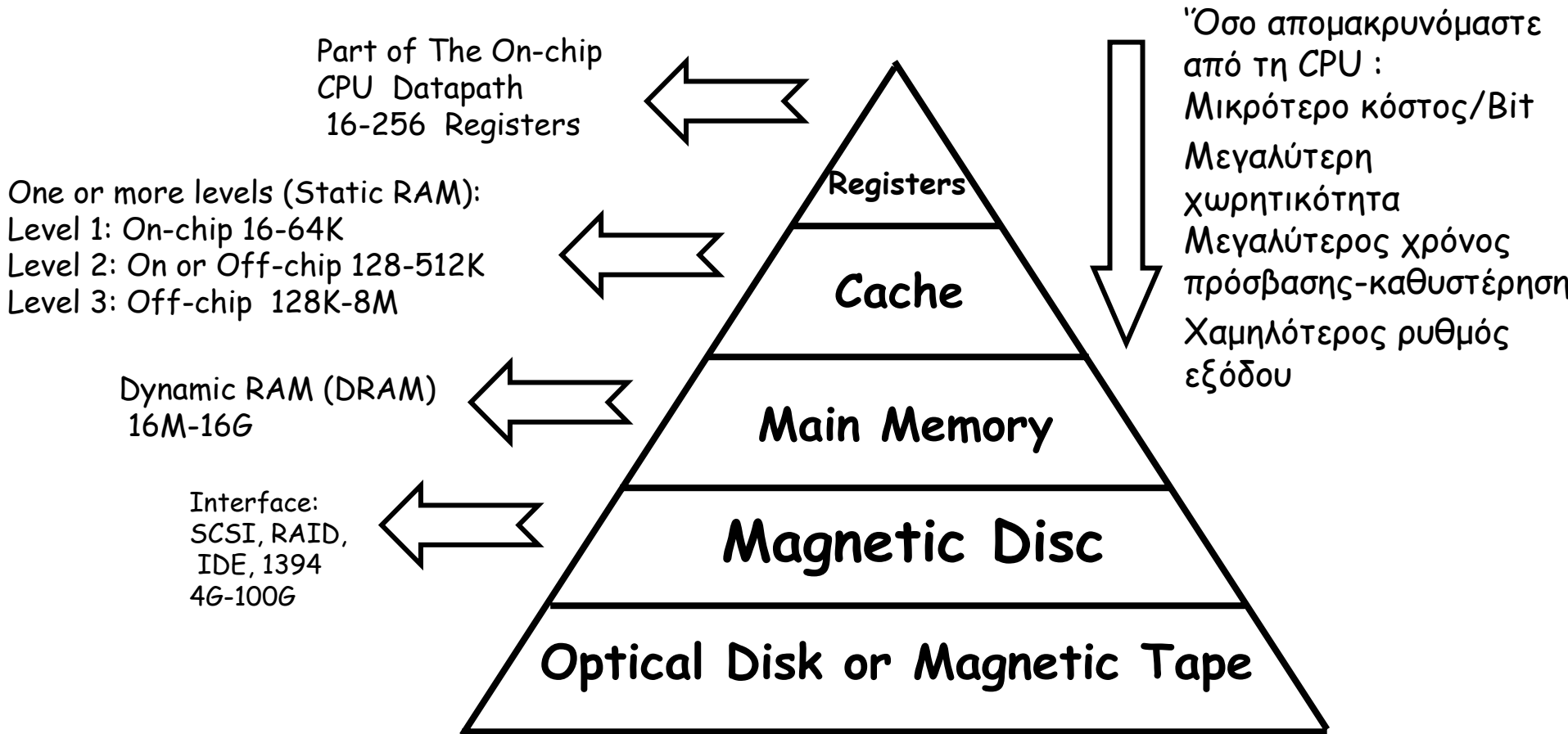


Processor-Memory (DRAM) Διαφορά επίδοσης

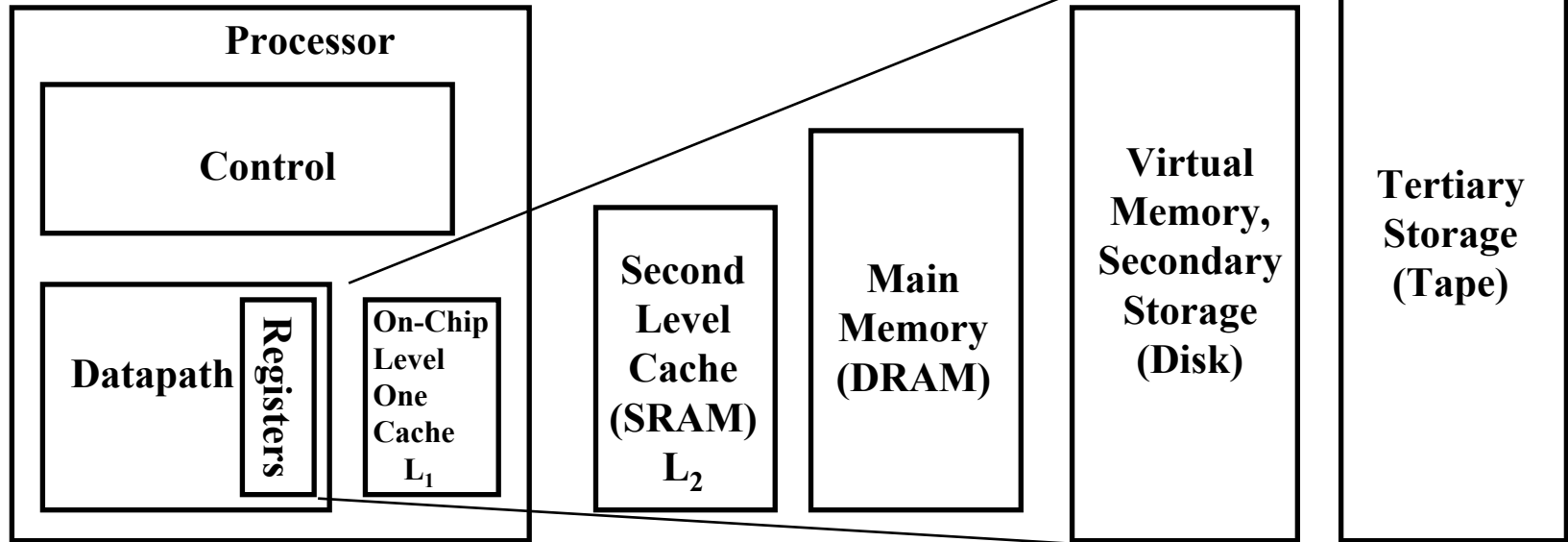


Ιεραρχία μνήμης



Παράδειγμα Ιεραρχίας Μνήμης (με 2 επίπεδα cache)

← Μεγαλύτερη Ταχύτητα
Μεγαλύτερη Χωρητικότητα →

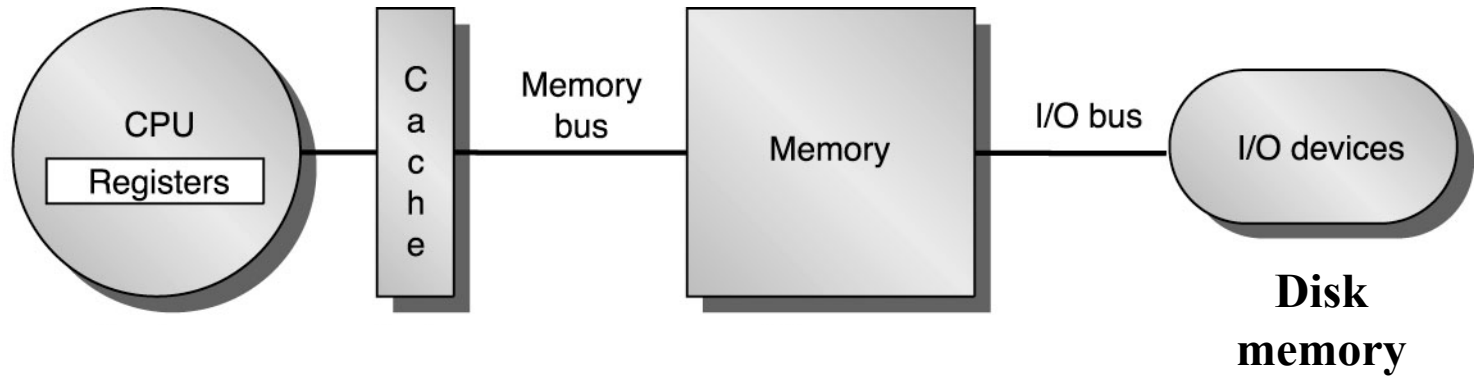


Ταχύτητα : 1-5ns 10ns 30ns 50ns 10ms 10sec

Μέγεθος : <1KB <256KB <8MB <4GB >1GB TB

Bandwidth : 150GB/s 50GB/s 25GB/s 4GB/s 10MB/s

Το μοντέλο της Ιεραρχίας Μνήμης



μέγεθος : 500bytes

ταχύτητα : 0,25ns

64KB

1ns

512MB

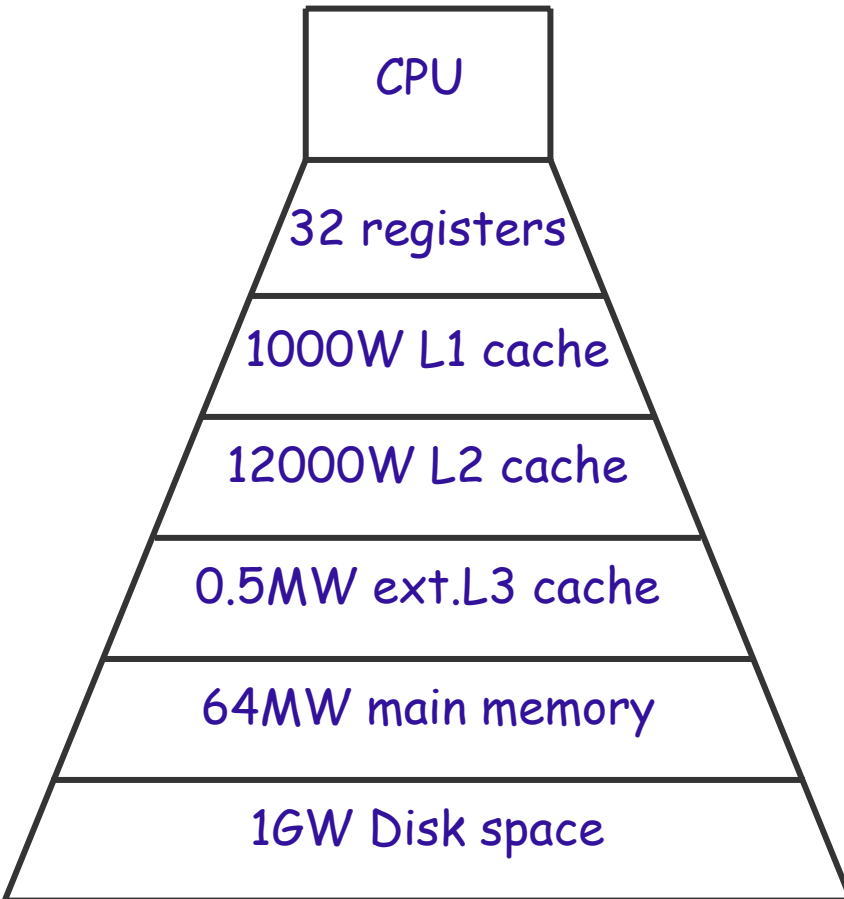
100ns

100GB

5ms

Παράδειγμα Ιεραρχίας μνήμης

Digital PWS 600 au - Alpha 21164 CPU - 600MHz



Level	Capacity	Throughput	Latency
Register	512B	24GB/sec	2ns
L1 cache	8KB	16GB/sec	2ns
L2 cache	96KB	8GB/sec	6ns
L3 cache	4MB	888MB/sec	24ns
Main Mem	512MB	1GB/sec	112ns

ΤΥΠΙΚΕΣ ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ

- **IBM Power 3:**
 - L1 = 64 KB, 128-way set associative
 - L2 = 4 MB, direct mapped, line size = 128, write back
- **Compaq EV6 (Alpha 21264):**
 - L1 = 64 KB, 2-way associative, line size = 32
 - L2 = 4 MB (or larger), direct mapped, line size = 64
- **HP PA:** no L2
 - PA8500, PA8600: L1 = 1.5 MB
 - PA8700: L1 = 2.25 MB
- **AMD Athlon:** L1 = 64 KB, L2 = 256 KB
- **Intel Pentium 4:** L1 = 8 KB, L2 = 256 KB
- **Intel Itanium:**
 - L1 = 16 KB, 4-way associative
 - L2 = 96 KB, 6-way associative
 - L3 = off chip, size varies

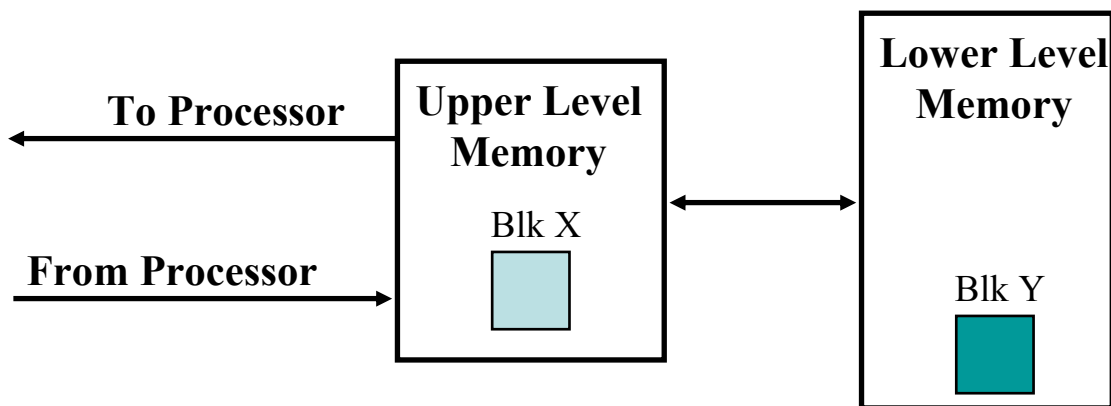
Γιατί είναι ωφέλιμη η Ιεραρχία Μνήμης:

Τοπικότητα δεδομένων (data locality)

- Κατά κανόνα τα προγράμματα προσπελούν ένα μικρό μόνο μέρος του συνόλου των διευθύνσεων (εντολές/δεδομένα) κατά την εκτέλεση ενός συγκεκριμένου τμήματός τους
- Δύο είδη τοπικότητας δεδομένων:
 - **Temporal Locality**: Στοιχεία που έχουν πρόσφατα προσπελαστεί τείνουν να προσπελούνται ξανά στο άμεσο μέλλον
 - **Spatial locality**: Γειτονικά στοιχεία όσων έχουν ήδη προσπελαστεί, έχουν αυξημένη πιθανότητα να προσπελαστούν στο άμεσο μέλλον
- Η ύπαρξη τοπικότητας στις αναφορές ενός προγράμματος, καθιστά εφικτή τη δυνατότητα να ικανοποιούνται η αίτηση για δεδομένα από *επίπεδα μνήμης* που βρίσκονται *ιεραρχικά ανώτερα*

Ορολογία

- **block - line - page** : η μικρότερη μονάδα μεταφοράς δεδομένων μεταξύ των επιπέδων μνήμης



Ορολογία

- **hit** : το block *βρίσκεται* σε κάποια θέση του εξεταζόμενου επιπέδου μνήμης
 - **hit rate** : hits/συνολικές προσπελάσεις μνήμης
 - **hit time** : χρόνος προσπέλασης των δεδομένων
- **miss** : το block *δεν υπάρχει* στο εξεταζόμενο επίπεδο μνήμης
 - **miss rate** : $1 - (\text{hit rate})$
 - **miss penalty** : (χρόνος μεταφοράς των δεδομένων ενός block στο συγκεκριμένο επίπεδο μνήμης) + (χρόνος απόκτησης των δεδομένων από την CPU)
 - **access time** : χρόνος απόκτησης της 1ης λέξης
 - **transfer time** : χρόνος απόκτησης των υπόλοιπων λέξεων

Η Βάση της Ιεραρχίας Μνήμης

- Οι δίσκοι περιέχουν όλα τα δεδομένα
- Όταν ο επεξεργαστής χρειάζεται κάποιο στοιχείο, αυτό ανεβαίνει σε ανώτερα επίπεδα μνήμης
- Η cache περιέχει αντίγραφα των στοιχείων της μνήμης που έχουν χρησιμοποιηθεί
- Η μνήμη περιέχει αντίγραφα των στοιχείων του δίσκου που έχουν χρησιμοποιηθεί

4 Ερωτήσεις για τις caches

- Πού μπορεί να τοποθετηθεί ένα block σε ένα ψηλότερο επίπεδο στην ιεραρχία μνήμης;
 - Τοποθέτηση block :
 - *direct-mapped, fully associative, set-associative*
- Πώς βρίσκουμε ένα block στα διάφορα επίπεδα μνήμης;
 - Αναγνώριση ενός block :
 - *Tag / Block*
- Ποιο από τα ήδη υπάρχοντα block της cache πρέπει να αντικατασταθεί σε περίπτωση ενός miss;
 - Μηχανισμός αντικατάστασης block :
 - *Random, Least Recently Used (LRU), FIFO*
- Τι συμβαίνει όταν μεταβάλλουμε το περιεχόμενο ενός block;
 - μηχανισμοί εγγραφής :
 - *write-through ή write-back*
 - *write-allocate ή no-write-allocate*

Οργάνωση της Cache

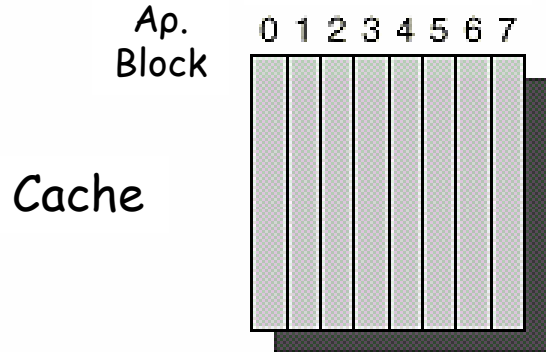
Τοποθέτηση ενός block μνήμης στην cache

- Direct mapped :
(διεύθυνση block) mod (αρ. block στην cache)
- Set associative :
(διεύθυνση block) mod (αρ. sets στην cache)
- Fully associative :
οπουδήποτε!

Οργάνωση της Cache

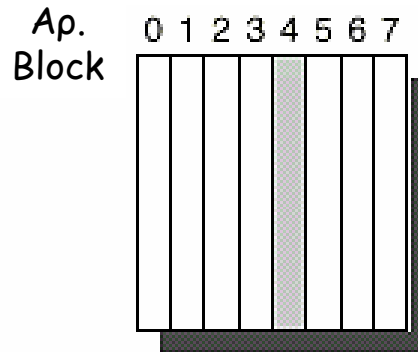
Fully associative:

Το block 12
μπάνει
οπουδήποτε



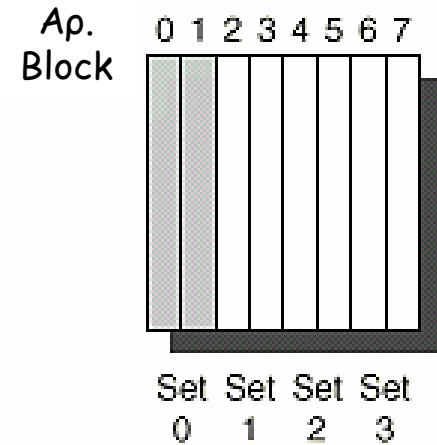
Direct mapped:

Το block 12 μπάνει
μόνο στο block 4
(=12 mod 8)

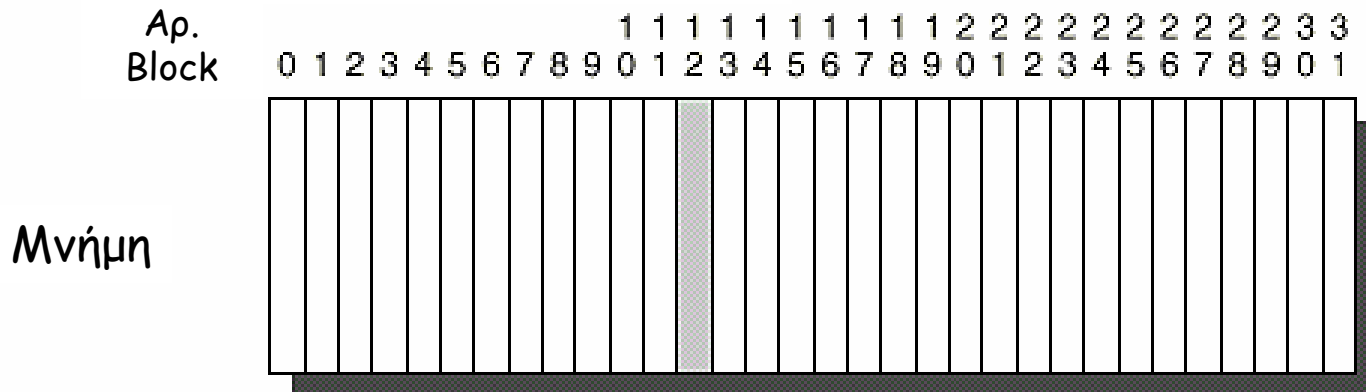


Set associative:

Το block 12 μπάνει
οπουδήποτε μέσα στο
set 0 (=12 mod 4)

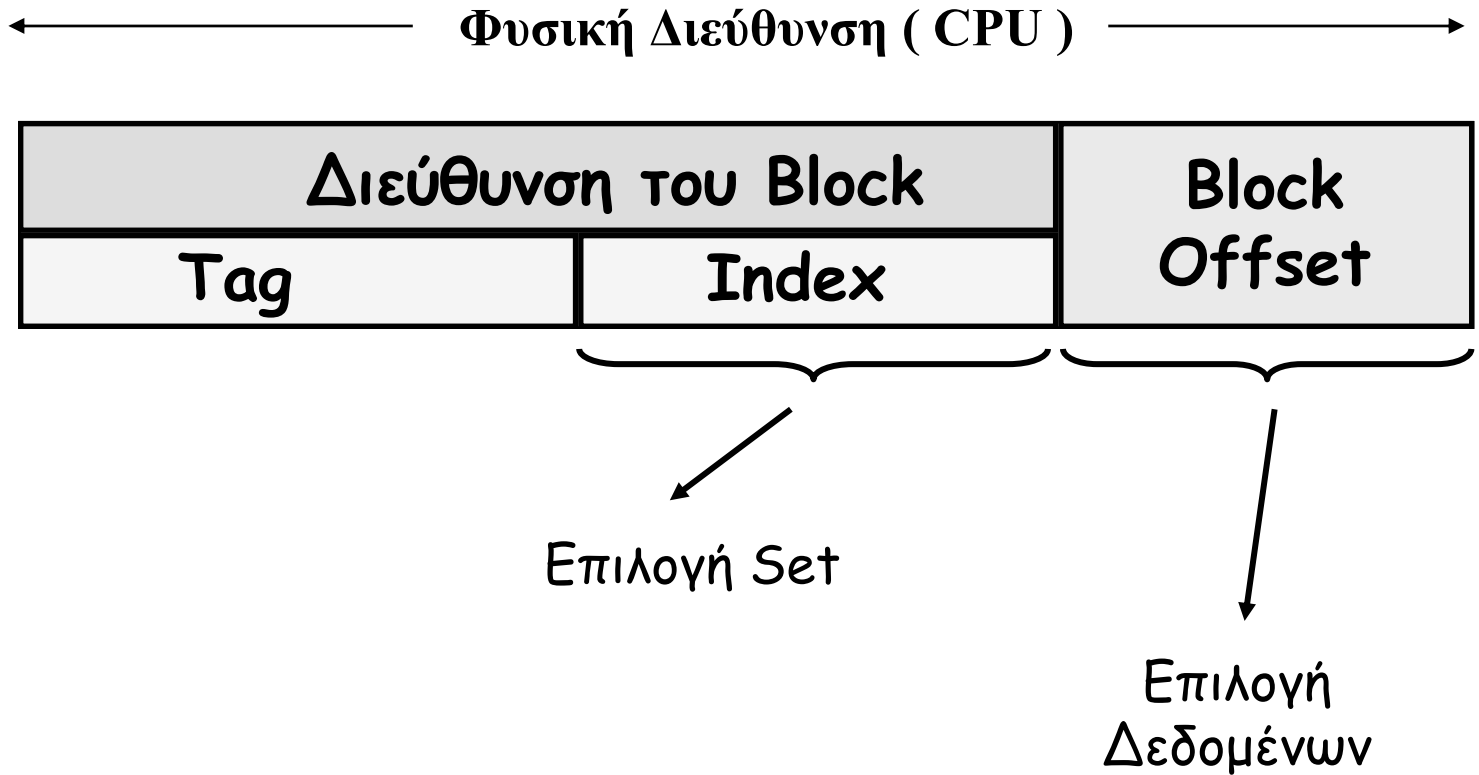


διεύθυνση του block frame



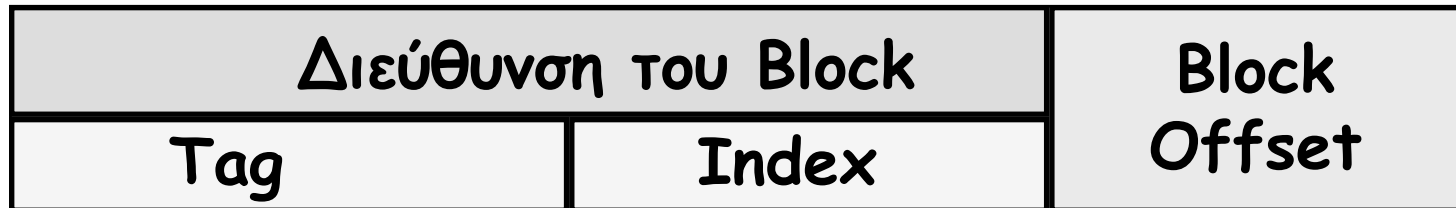
Cache με 8 blocks και μνήμη με 32 blocks

Τα πεδία διεύθυνσης



Τα πεδία διεύθυνσης

← Φυσική Διεύθυνση (CPU) →



Μέγεθος block offset = $\log_2(\text{μέγεθος block})$

Μέγεθος Index = $\log_2(\text{Συνολικός αριθμός blocks/associativity})$

Μέγεθος tag = μέγεθος address - μέγεθος index - μέγεθος offset

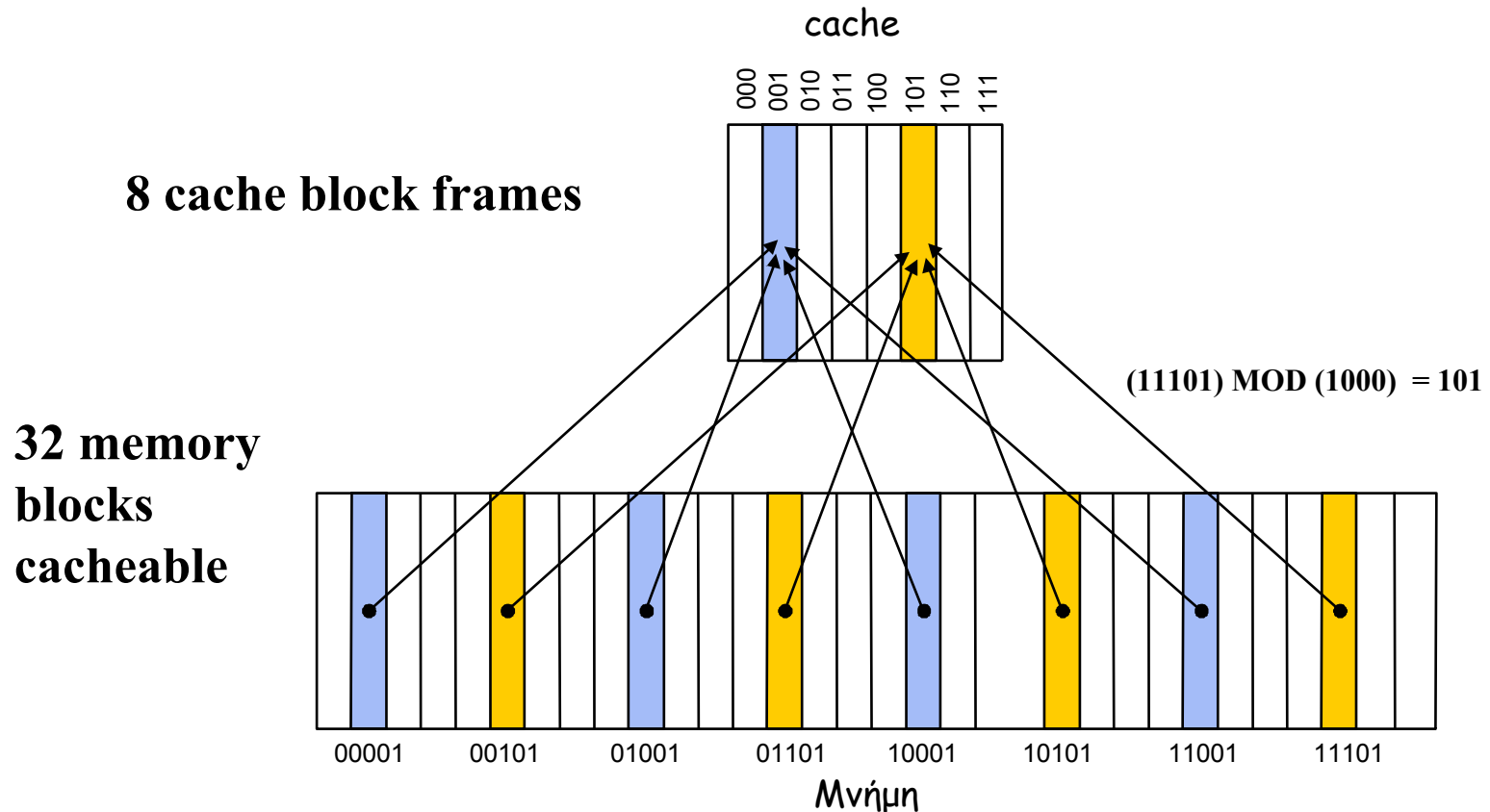
Οργάνωση της Cache

Direct Mapped Cache

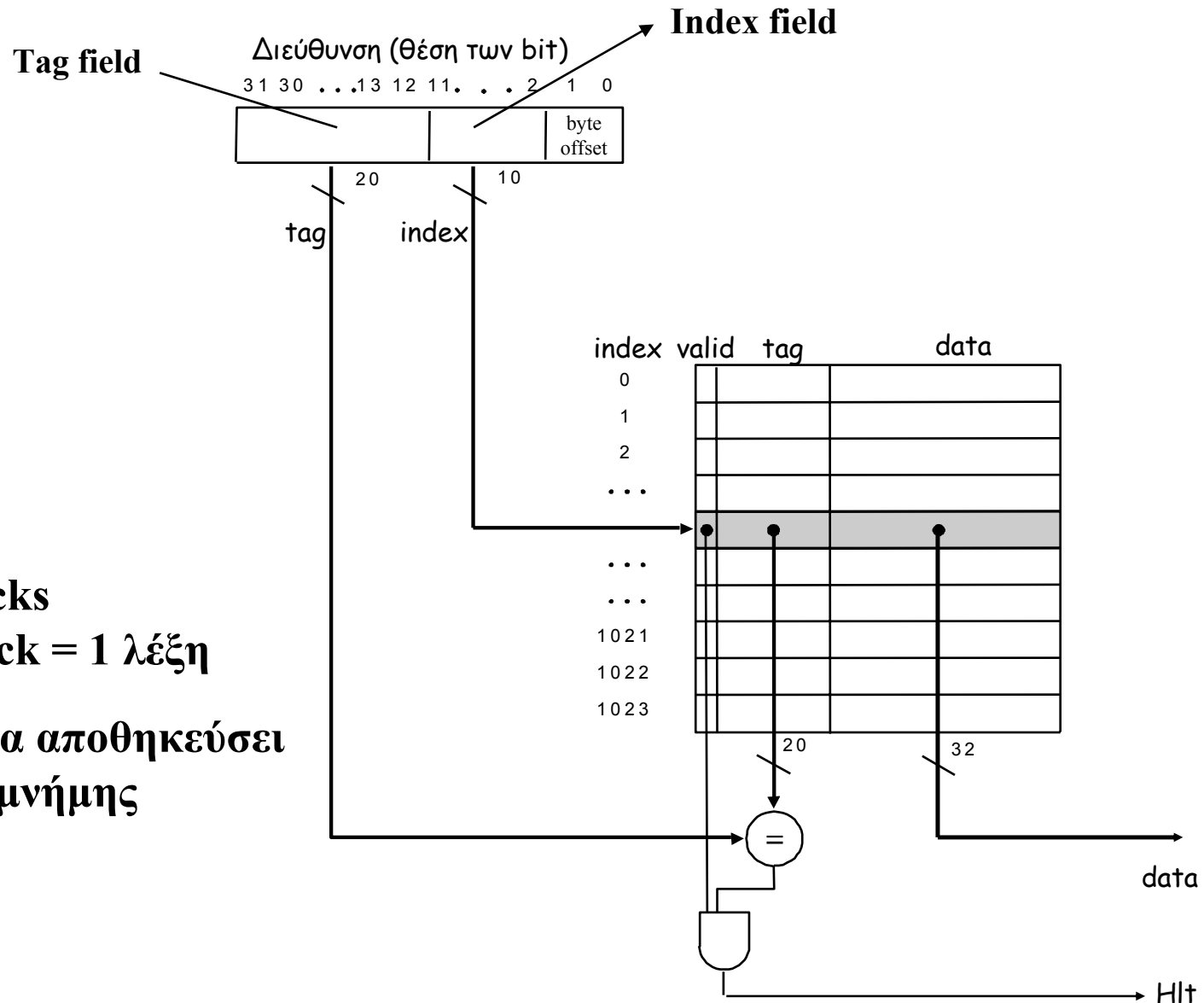
Κάθε block μπορεί να αποθηκευθεί μόνο σε μία θέση :

(διεύθυνση block) MOD (Αρ. blocks στην cache)

στο παράδειγμά μας: (διεύθυνση block address) MOD (8)



Παράδειγμα : Direct Mapped Cache



1024 Blocks

Κάθε block = 1 λέξη

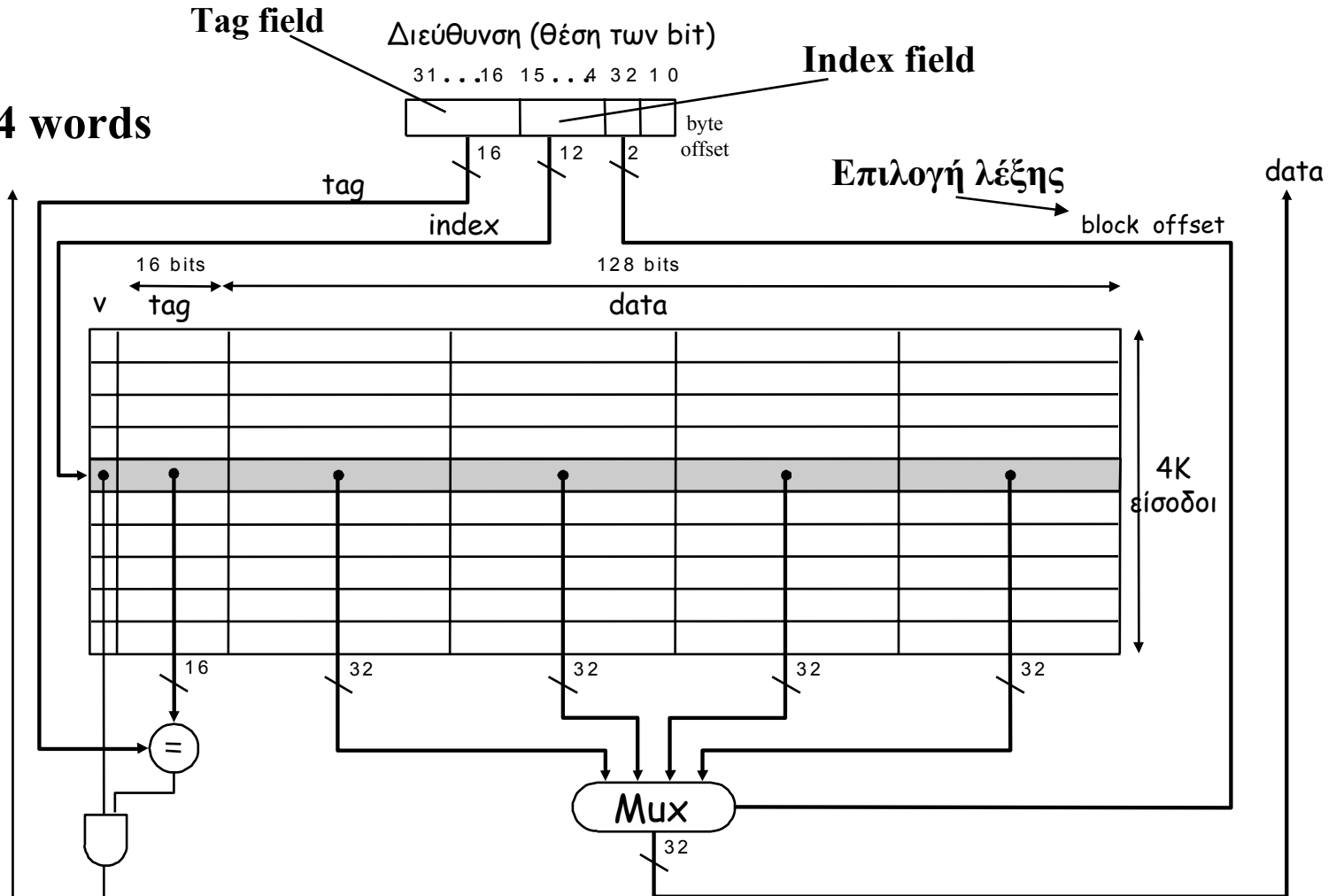
**Μπορεί να αποθηκεύσει
 2^{32} bytes μνήμης**

Παράδειγμα

Direct Mapped Cache

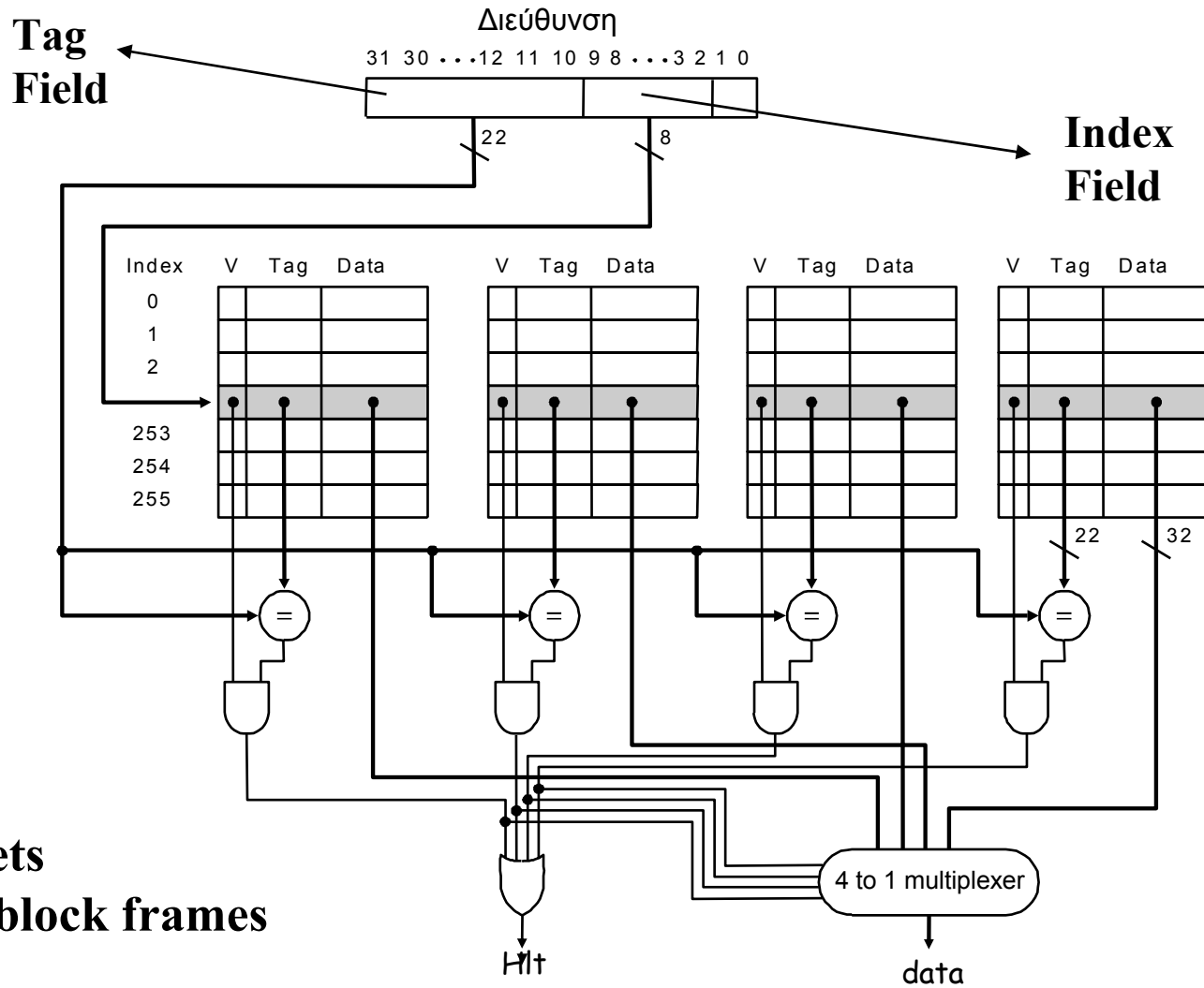
4K blocks

Κάθε block = 4 words



Καλύτερη αξιοποίηση της spatial locality

4-Way Set Associative Cache: (MIPS)



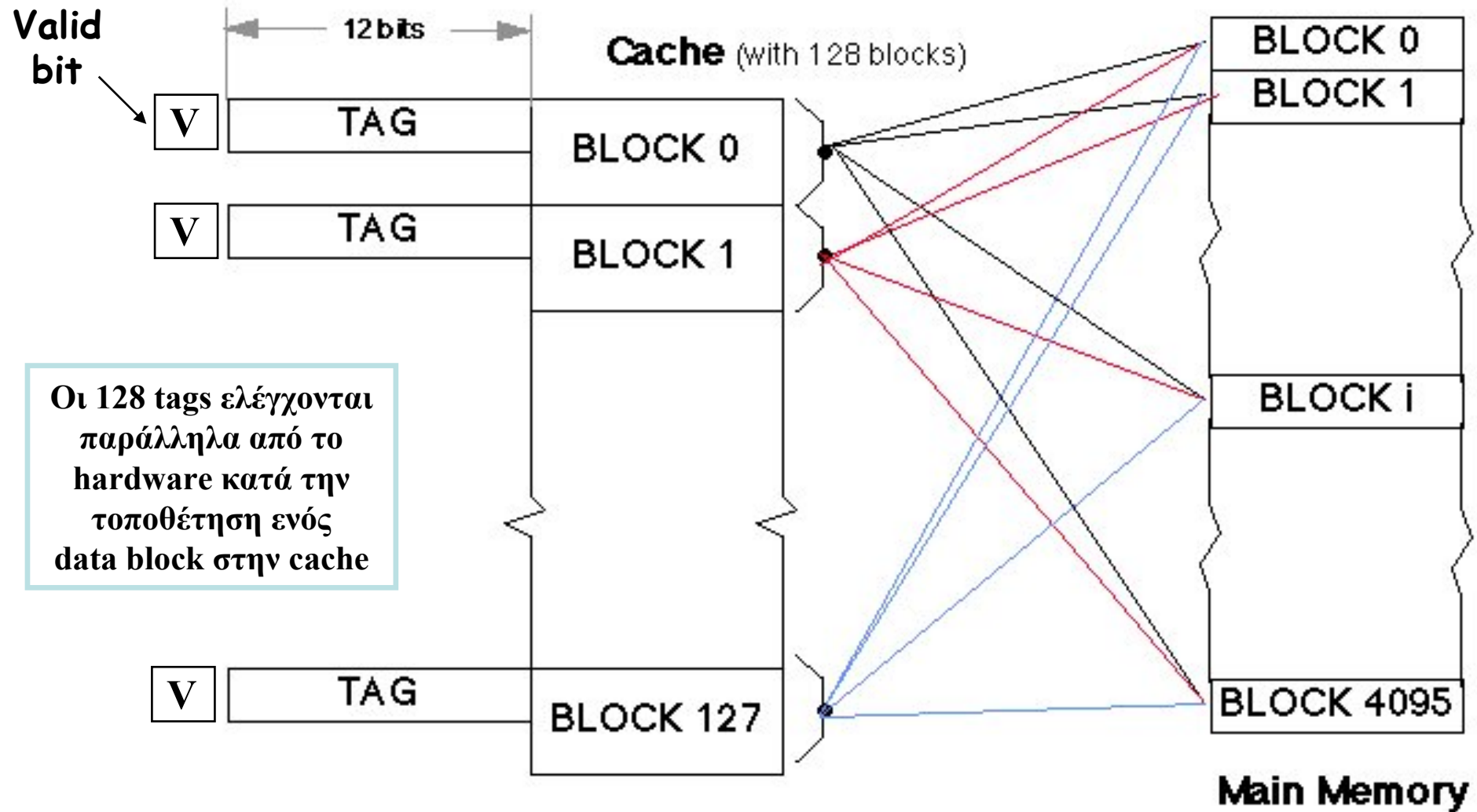
256 sets

1024 block frames

Παράδειγμα οργάνωσης cache- διευθυνσιοδότηση

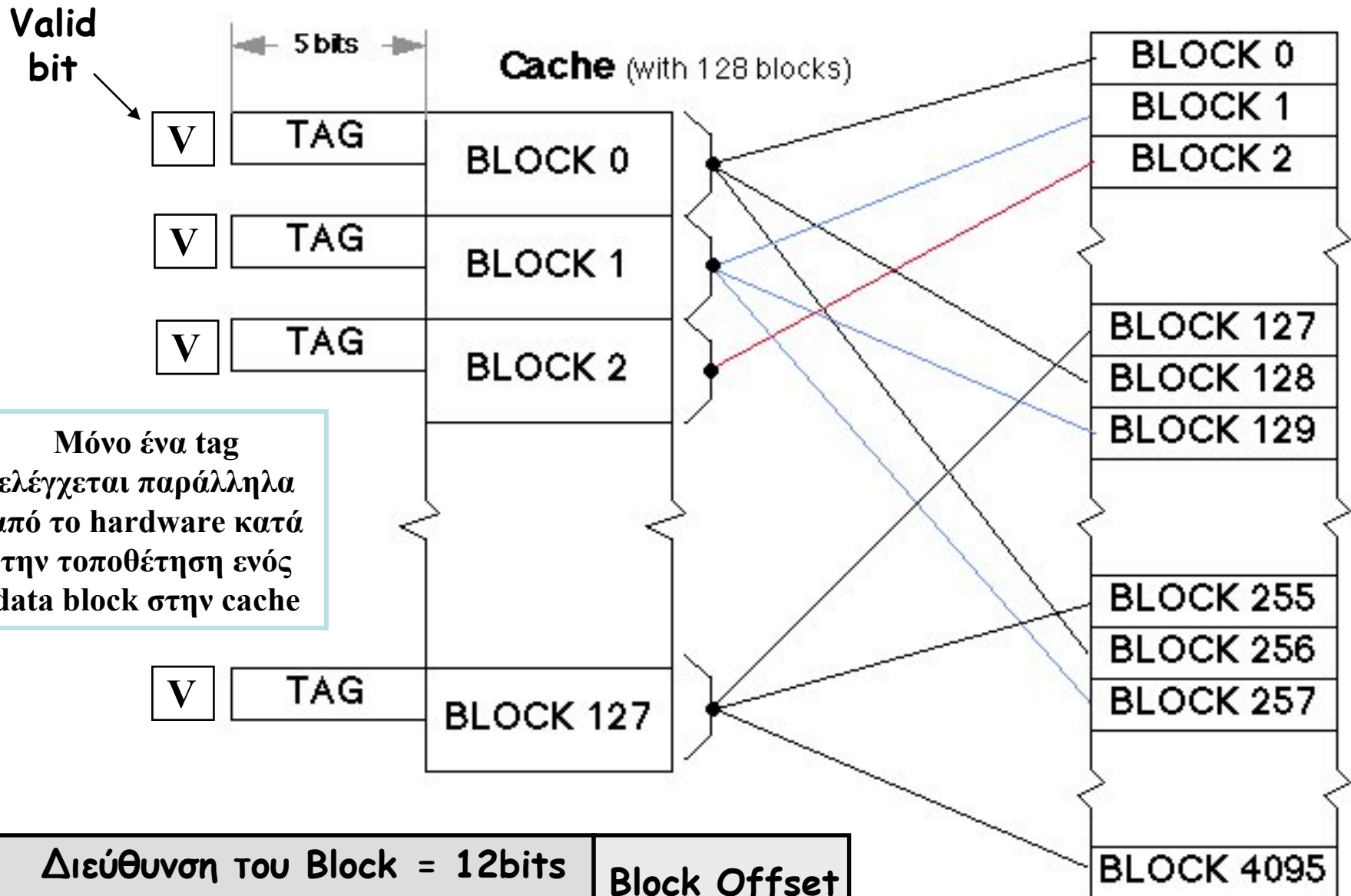
- L_1 cache με 128 cache block frames
- Κάθε block frame περιέχει 4 λέξεις (16 bytes)
- 16-bit διευθύνσεις μνήμης στην cache
(64Kbytes κύρια μνήμη ή 4096 blocks μνήμης)
- Δείξτε την οργάνωση της cache (mapping) και τα πεδία διευθύνσεων της cache για:
 - Fully Associative cache.
 - Direct mapped cache.
 - 2-way set-associative cache.

Fully Associative Case



Διεύθυνση του Block = 12bits	Block Offset = 4bits
Tag = 12bits	

Direct Mapped Cache



Μόνο ένα tag ελέγχεται παράλληλα από το hardware κατά την τοποθέτηση ενός data block στην cache

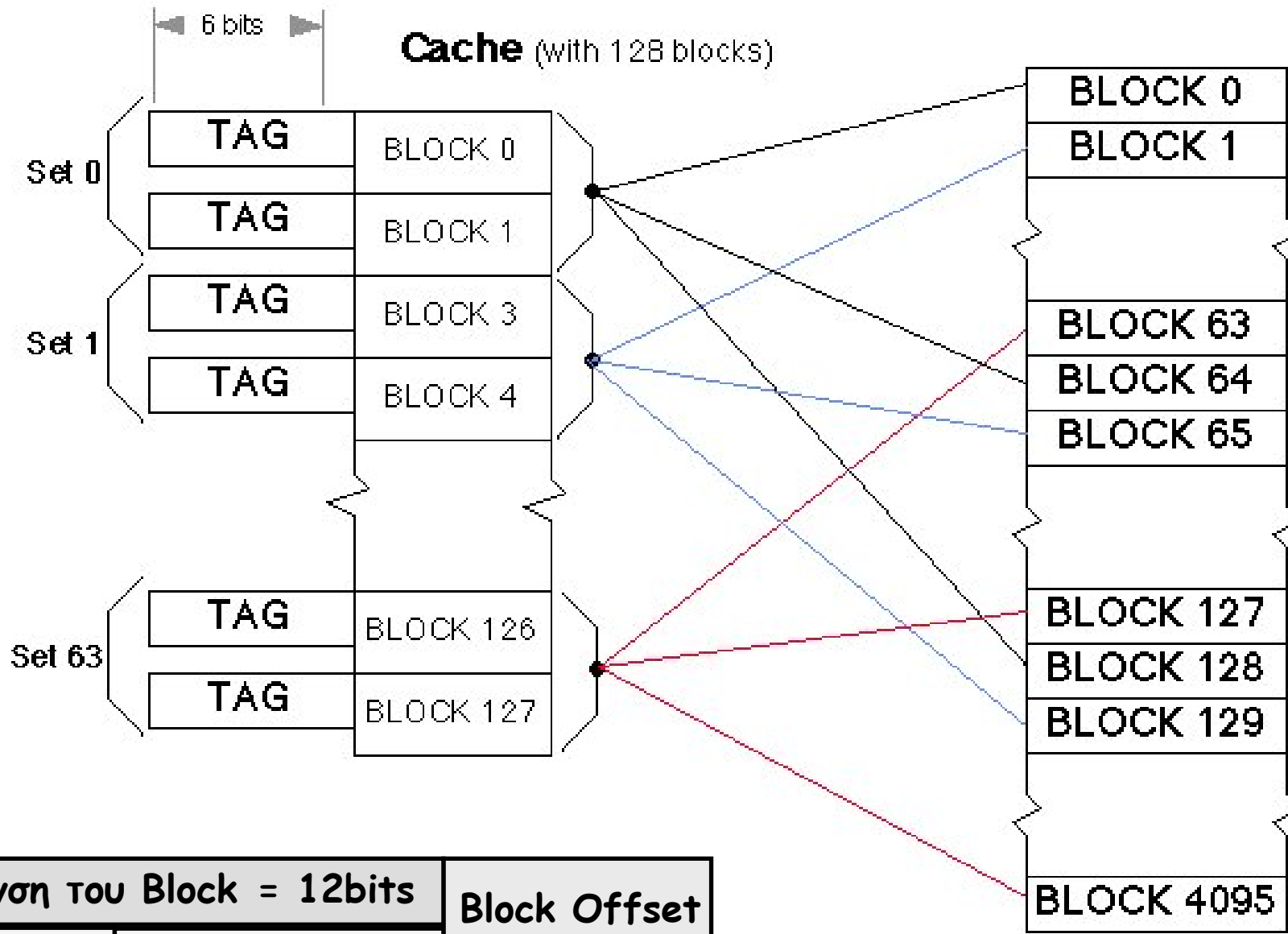
Διεύθυνση του Block = 12bits

Block Offset = 4bits

Tag = 5bits

Index = 7bits

2-Way Set-Associative Cache



Δύο tags σε ένα set ελέγχονται παράλληλα από το hardware κατά την τοποθέτηση ενός data block στην cache

Διεύθυνση του Block = 12bits		Block Offset = 4bits
Tag = 6bits	Index = 6bits	

Προσπέλαση δεδομένων σε Direct Mapped Cache

Η CPU καλεί προς ανάγνωση τις εξής διευθύνσεις:

0x00000014

0x00000048

0x0000001C

0x00004014

Κύρια μνήμη

διεύθυνση

τιμή της λέξης

...

...

00000010

a

00000014

b

00000018

c

0000001C

d

...

...

00000040

e

00000044

f

00000048

g

0000004C

h

...

...

00004010

i

00004014

j

00004018

k

0000401C

l

...

...

8KB Direct-mapped cache 4W blocks

Αρχικά όλες οι θέσεις invalid

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...
510	0					
511	0					

8KB Direct-mapped cache 4W blocks

Read 000000000000000000000000 000000001 0100 (0x00000014)

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...
510	0					
511	0					

Read block 1 : invalid data στο block 1 !

8KB Direct-mapped cache 4W blocks

Read 000000000000000000000000 000000001 0100

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...
510	0					
511	0					

Φόρτωση τα ζητούμενα δεδομένα στην cache !

8KB Direct-mapped cache 4W blocks

Read 000000000000000000000000 000000001 0100

	index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0						
1	1	1	0	a	b	c	d
2	2	0					
3	3	0					
4	4	0					
5	5	0					
6	6	0					
7	7	0					
...
510	510	0					
511	511	0					

Επέστρεψε το b (θέση 0100) στην CPU

8KB Direct-mapped cache 4W blocks

Read 000000000000000000000000 000000100 1000 (0x00000048)

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...
510	0					
511	0					

Read block 4 : invalid data στο block 4 !

8KB Direct-mapped cache 4W blocks

Read 000000000000000000000000 000000100 1000

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	1	0	e	f	g	h
5	0					
6	0					
7	0					
...
510	0					
511	0					

Φόρτωση τα ζητούμενα δεδομένα στην cache και κάνε το block valid !

8KB Direct-mapped cache 4W blocks

Read 000000000000000000000000 000000100 1000

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	1	0	e	f	g	h
5	0					
6	0					
7	0					
...
510	0					
511	0					

Επέστρεψε στην CPU την τιμή g !

8KB Direct-mapped cache 4W blocks

Read 000000000000000000000000 000000001 1100 (0x0000001C)

	index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0	0					
1	1	1	0	a	b	c	d
2	2	0					
3	3	0					
4	4	1	0	e	f	g	h
5	5	0					
6	6	0					
7	7	0					
...
510	510	0					
511	511	0					

Read block 1 !

8KB Direct-mapped cache 4W blocks

Read 000000000000000000000000 0000000001 1100

	index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0						
1	1	1	0	a	b	c	d
2	0						
3	0						
4	1	1	0	e	f	g	h
5	0						
6	0						
7	0						
...
510	0						
511	0						

Το πεδίο tag έχει τη σωστή τιμή ! Άρα επιστρέφεται η τιμή d

8KB Direct-mapped cache 4W blocks

Read 0000000000000000000010 000000001 0100 (0x00004014)

	index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0						
1	1	1	0	a	b	c	d
2	2	0					
3	3	0					
4	4	1	0	e	f	g	h
5	5	0					
6	6	0					
7	7	0					
...
510	510	0					
511	511	0					

Read block 1 !

8KB Direct-mapped cache 4W blocks

Read 0000000000000000000010 000000001 0100

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	1	0	a	b	c	d
2	0					
3	0					
4	1	0	e	f	g	h
5	0					
6	0					
7	0					
...
510	0					
511	0					

Valid data αλλά το πεδίο tag δεν είναι το σωστό $2 \neq 0$

Miss : πρέπει να αντικατασταθεί το block 1 με νέα δεδομένα

8KB Direct-mapped cache 4W blocks

Read 000000000000000000000010 0000000001 0100

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	1	2	i	j	k	l
2	0					
3	0					
4	1	0	e	f	g	h
5	0					
6	0					
7	0					
...
510	0					
511	0					

Φόρτωση το σωστό περιεχόμενο και στείλε το j στην CPU

Υπολογισμός του αριθμού των bits που χρειάζονται

- Πόσα bits συνολικά χρειάζονται σε μία direct-mapped cache με 64 KBytes data και blocks της 1 λέξης, για 32-bit διευθύνσεις;
 - 64 Kbytes = 16 Kwords = 2^{14} words = 2^{14} blocks
 - Block size = 4 bytes \Rightarrow offset size = 2 bits,
 - #sets = #blocks = 2^{14} \Rightarrow index size = 14 bits
 - Tag size = address size - index size - offset size = $32 - 14 - 2 = 16$ bits
 - Bits/block = data bits + tag bits + valid bit = $32 + 16 + 1 = 49$
 - Bits της cache = #blocks \times bits/block = $2^{14} \times 49 = 98$ Kbytes
- Πόσα bits συνολικά χρειάζονται σε μία 4-way set associative cache για την αποθήκευση των ίδιων δεδομένων;
 - Block size και #blocks δεν αλλάζει.
 - #sets = #blocks/4 = $(2^{14})/4 = 2^{12}$ \Rightarrow index size = 12 bits
 - Tag size = address size - index size - offset = $32 - 12 - 2 = 18$ bits
 - Bits/block = data bits + tag bits + valid bit = $32 + 18 + 1 = 51$
 - Bits της cache = #blocks \times bits/block = $2^{14} \times 51 = 102$ Kbytes
- Αύξηση του associativity \Rightarrow Αύξηση των bits της cache

Υπολογισμός του αριθμού των bits της cache που χρειάζονται

- Πόσα bits συνολικά χρειάζονται σε μία direct-mapped cache με 64KBytes data και blocks των 8 λέξεων, για 32-bit διευθύνσεις (2^{32} bytes μπορούν να αποθηκευθούν στη μνήμη);
 - 64 Kbytes = 2^{14} words = $(2^{14})/8 = 2^{11}$ blocks
 - block size = 32 bytes
 - => offset size = block offset + byte offset = 5 bits
 - #sets = #blocks = 2^{11} => index size = 11 bits
 - tag size = address size - index size - offset size = $32 - 11 - 5 = 16$ bits
 - bits/block = data bits + tag bits + valid bit = $8 \times 32 + 16 + 1 = 273$ bits
 - bits in cache = #blocks x bits/block = $2^{11} \times 273 = 68.25$ Kbytes
- Αύξηση του μεγέθους του block => Μείωση των bits της cache.

Μηχανισμοί αντικατάστασης ενός block της cache

- **Random** (τυχαία) - επιλογή ενός τυχαίου block με βάση κάποια ψευδοτυχαία ακολουθία
 - απλή υλοποίηση στο hardware
 - είναι η τεχνική που χρησιμοποιείται συνήθως
- **LRU** (least recently used) - αντικαθιστάται το block που δεν έχει χρησιμοποιηθεί για περισσότερη ώρα
 - ακριβή υλοποίηση στο hardware

Μηχανισμοί εγγραφής σε block (σε περίπτωση write-hit)

- Σε περίπτωση write-hit, γνωστοποιείται η αλλαγή στην κύρια μνήμη ;
 - ναι : **write-through**
 - όχι : **write-back**
- Σε περίπτωση miss, τοποθετείται το block στην cache;
 - ναι : **write-allocate** (συνήθως με write-back)
 - όχι : **write-no-allocate** (συνήθως με write-through)

Write-Back & Write-Through

- **write-back** : ενημέρωση της μνήμης μόνο κατά την απομάκρυνση του block από την cache
 - οι εγγραφές πραγματοποιούνται με την ταχύτητα της cache
 - dirty bit κατά την τροποποίηση - αντικατάσταση των "clean" block χωρίς ενημέρωση της μνήμης
 - Χαμηλό ποσοστό misses
 - Πολλές εγγραφές σε μία ενημέρωση
- **write-through** : ενημέρωση της μνήμης σε κάθε εγγραφή
 - το κατώτερο ιεραρχικά επίπεδο περιέχει τα εγκυρότερα δεδομένα
 - εύκολη υλοποίηση
 - αυξημένη μετακίνηση δεδομένων προς τη μνήμη
 - συχνά χρησιμοποιείται ένας write buffer για αποφυγή καθυστερήσεων όσο ενημερώνεται η μνήμη

Write-Allocate & write-no-Allocate

(στη περίπτωση write-miss)

- **Write-allocate** : το block φορτώνεται από τη μνήμη στη cache και στη συνέχεια μεταβάλλουμε τα δεδομένα του (χωρίς να ενημερώσουμε τα "κατώτερα" (πιο κοντινά στην κεντρική μνήμη) επίπεδα μνήμης)
- **write-no-allocate** : οι μετατροπές των δεδομένων γίνονται απευθείας στα χαμηλότερα επίπεδα της ιεραρχίας μνήμης (χωρίς να εμπλακεί η cache)

"Allocate" a cache line to store the memory block !

Read hit / misses

- **read hit** : ανάγνωση των δεδομένων από την cache
- **read miss** : μεταφορά ολόκληρου του block που περιέχει τα δεδομένα που αναζητάμε στην cache και στη συνέχεια όπως στο read hit

Write hit / misses

- **Write-back & Write-allocate**
 - **write hit** : εγγραφή των δεδομένων στην cache (μόνο). Η κύρια μνήμη ενημερώνεται μόνο όταν απομακρυνθεί το block από την cache και το block είναι dirty
 - **write miss** : το block μεταφέρεται στην cache (στη σωστή θέση) και στη συνέχεια όπως στο write hit

Write hit / misses

- **Write-through & write-no-allocate**
 - **write hit** : εγγραφή των νέων δεδομένων στην cache και ενημέρωση της κύρια μνήμης
 - **write miss** : η εγγραφή γίνεται μόνο στην κύρια μνήμη, ενώ δεν εμπλέκεται καθόλου η cache

Συνέχεια από το προηγούμενο παράδειγμα... 8KB Direct-mapped cache - 4W blocks write through

Write 000000000000000000000000 000000100 0100 (0x00000044), m

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	1	2	i	j	k	l
2	0					
3	0					
4	1	0	e	f	g	h
5	0					
6	0					
7	0					
...
510	0					
511	0					

Read block 4 !

8KB Direct-mapped cache - 4W blocks write through

Write 000000000000000000000000 000000100 0100

index	valid	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0					
1	1	2	i	j	k	l
2	0					
3	0					
4	1	0	e	m	g	h
5	0					
6	0					
7	0					
...
510	0					
511	0					

Valid data - σωστό tag - εγγραφή στο πεδίο 0100 της cache
και ενημέρωση της κύριας μνήμης !

8KB Direct-mapped cache - 4W blocks write back

Write 000000000000000000000000 000000100 0100 (0x00000044), m

Ind. V dirty tag 0x0-3 0x4-7 0x8-B 0xC-F

0	0	0					
1	1	0	2	i	j	k	l
2	0	0					
3	0	0					
4	1	0	0	e	f	g	h
5	0	0					
6	0	0					
7	0	0					
...
510	0	0					
511	0	0					

Read block 4 !

8KB Direct-mapped cache - 4W blocks write back

Write 000000000000000000000000 000000100 0100

	Ind.	V	dirty	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0	0						
1	1	0		2	i	j	k	l
2	0	0						
3	0	0						
4	1	1		0	e	m	g	h
5	0	0						
6	0	0						
7	0	0						
...
510	0	0						
511	0	0						

Valid data - σωστό tag - εγγραφή στο πεδίο 0100 της cache
και ενημέρωση του dirty bit !

8KB Direct-mapped cache - 4W blocks write back

Read 00000000000000000000100 000000100 1100 (0x0000804C)

Ind.	V	dirty	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0	0					
1	1	0	2	i	j	k	l
2	0	0					
3	0	0					
4	1	1	0	e	m	g	h
5	0	0					
6	0	0					
7	0	0					
...
510	0	0					
511	0	0					

Read block 4!

8KB Direct-mapped cache - 4W blocks write back

Read 00000000000000000000100 000000100 1100

Ind.	V	dirty	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0	0					
1	1	0	2	i	j	k	l
2	0	0					
3	0	0					
4	1	1	0	e	m	g	h
5	0	0					
6	0	0					
7	0	0					
...
510	0	0					
511	0	0					

Valid data - το πεδίο tag όμως δεν ταιριάζει : 0!=4

Το dirty bit είναι 1 : Ενημερώνεται η μνήμη (0x00000040-0x0000004F)
και στη συνέχεια φορτώνεται η σωστή διεύθυνση

8KB Direct-mapped cache - 4W blocks write back

Read 00000000000000000000100 0000000100 1100

	Ind.	V	dirty	tag	0x0-3	0x4-7	0x8-B	0xC-F
0	0	0						
1	1	0		2	i	j	k	l
2	0	0						
3	0	0						
4	1	0		4	p	q	r	s
5	0	0						
6	0	0						
7	0	0						
...
510	0	0						
511	0	0						

Φορτώνεται η σωστή διεύθυνση - ενημερώνονται τα πεδία tag - dirty

Επιστρέφεται η τιμή r στη CPU

Επίδοση των επιπέδων μνήμης (performance)

- μέσος χρόνος προσπέλασης των δεδομένων
(access time)

$$t_{avg} = t_{hit} + miss\ rate \cdot t_{miss\ penalty}$$

Cache : ενοποιημένη ή όχι;

- Ενοποιημένη για εντολές και δεδομένα (unified)
 - Μικρότερο κατασκευαστικό κόστος
 - Καλύτερο ισοζύγισμα του χώρου που καταλαμβάνεται από εντολές/δεδομένα
 - Επιπλέον misses λόγω διεκδίκησης κοινών θέσεων στην cache (conflict misses)
- Δύο διαφορετικές caches για εντολές και δεδομένα (data cache & instruction cache)
 - 2-πλάσιο εύρος ζώνης
 - όχι conflict misses

Παράδειγμα

- Σε ποια περίπτωση έχουμε καλύτερη επίδοση;
Σε σύστημα με 16KB instruction cache και 16KB data cache ή σε σύστημα με 32KB unified cache;
Υποθέτουμε ότι το 36% των εντολών είναι εντολές αναφοράς στη μνήμη (load/store).
hit time = 1 clock cycle
miss penalty = 100 clock cycles
στη unified cache είναι: hit time = 2 clock cycles όταν πρόκειται για εντολή load/store

Χρησιμοποιείστε τα δεδομένα του ακόλουθου πίνακα :

	Instr. cache	data cache	unified cache
16KB	3.82	40.9	51.0
32KB	1.36	38.4	43.3

Παράδειγμα (συνέχεια...)

- Λύση

$$\text{miss rate} = \frac{\text{misses}}{\text{mem accesses}}$$

$$\left. \begin{aligned} \text{miss rate}_{16\text{KB instr cache}} &= \frac{3.82}{1000} = 0.0038 \\ \text{miss rate}_{16\text{KB data cache}} &= \frac{40.9}{360} = 0.114 \end{aligned} \right\} 74\% \cdot 0.0038 + 26\% \cdot 0.114 = 0.0324$$
$$\text{miss rate}_{32\text{KB unif cache}} = \frac{43.3}{1000 + 360} = 0.0318$$

miss rate (unified cache) < miss rate (instr + data cache)

Παράδειγμα (συνέχεια...)

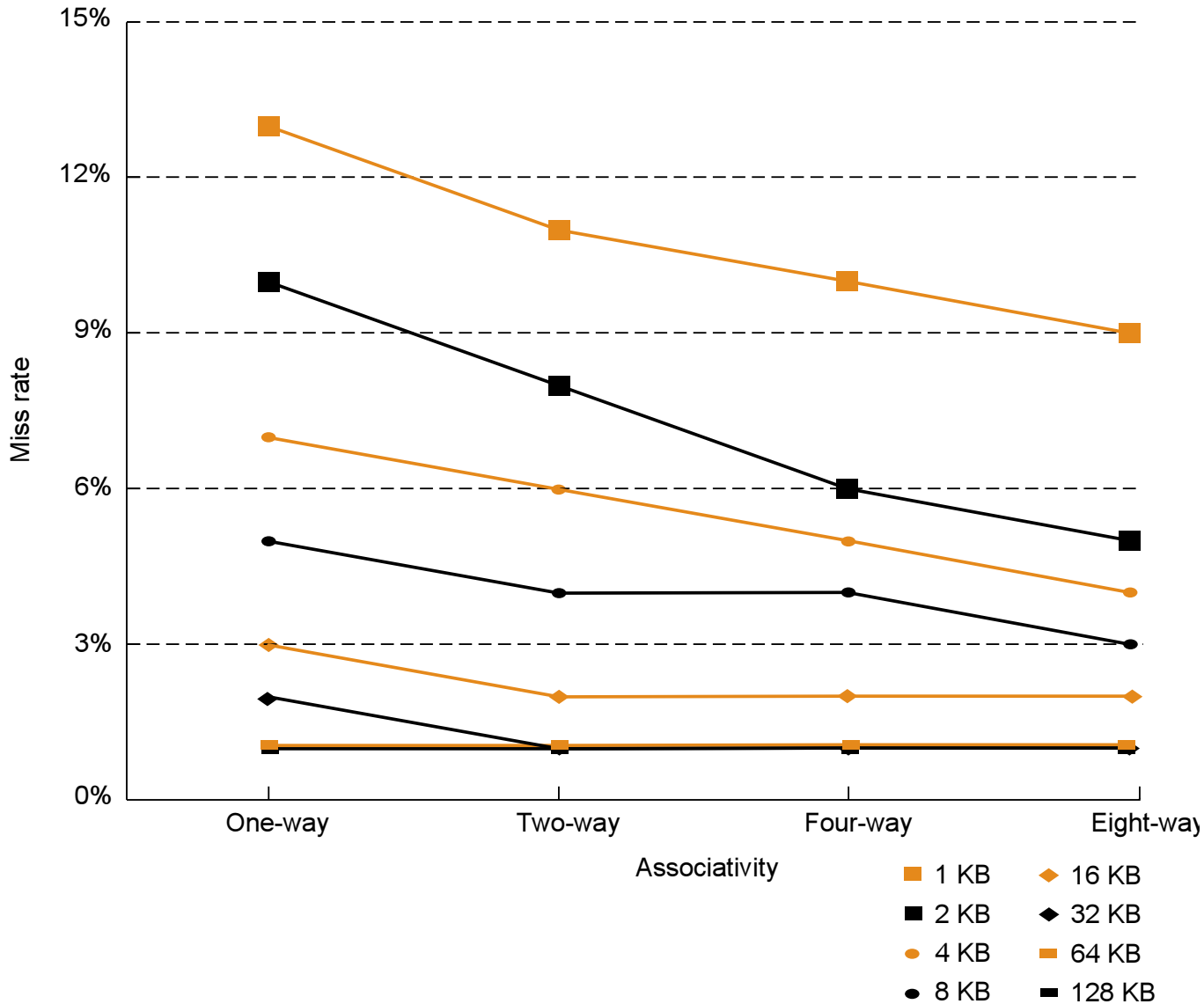
- Λύση

$$t_{avg} = t_{instr} + t_{data} = 74\%(1 + 0.004 \cdot 100) + 26\%(1 + 0.114 \cdot 100) \\ = 4.24$$

$$t_{avg} = 1 + 0.26 + 0.038 \cdot 100 = 4.44$$

μέσος χρόνος/access (instr+data cache) < μέσος χρόνος/access (unified cache)

Cache Associativity



Παρατήρηση :
Μια 4-way cache
έχει σχεδόν το ίδιο
hit rate με μια
direct-
mapped cache
διπλάσιου μεγέθους

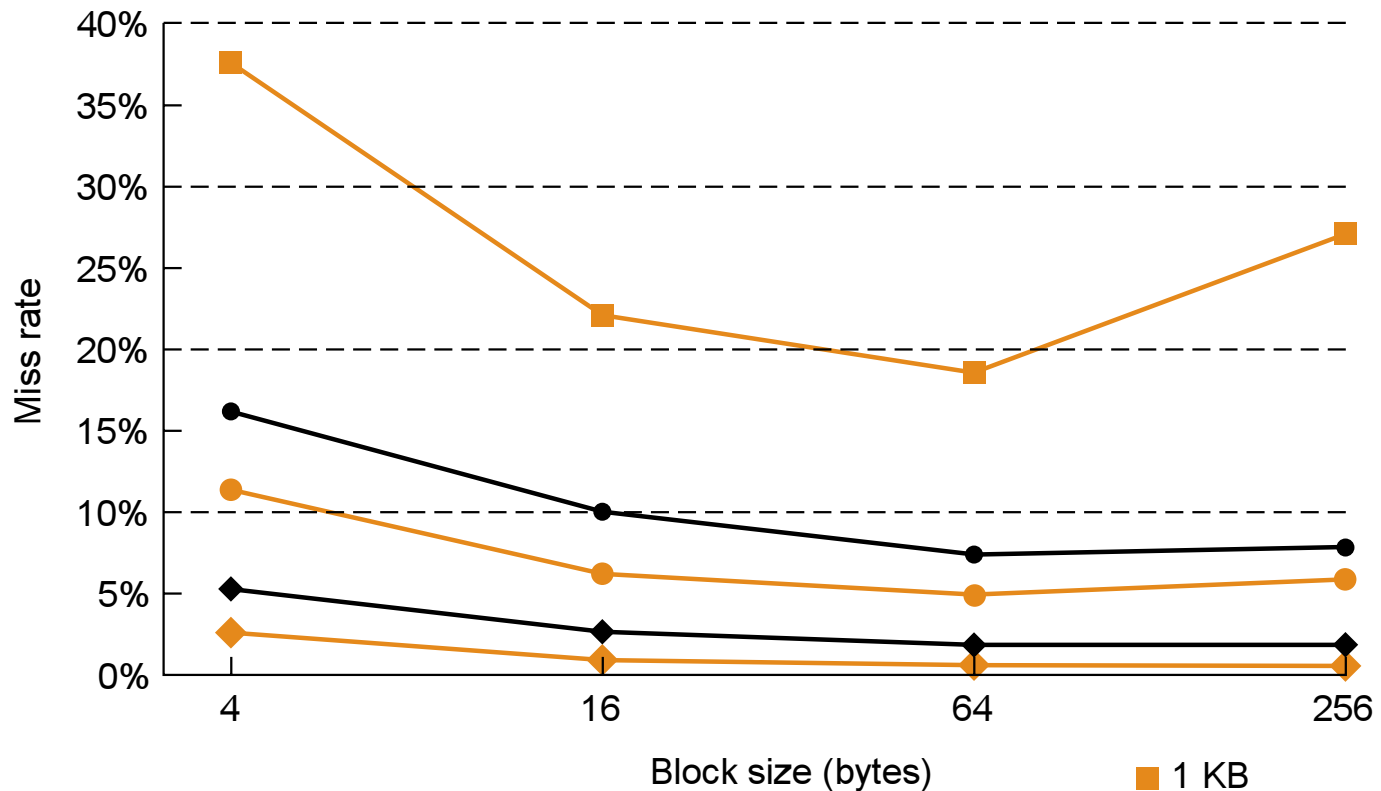
Μεγάλα Cache Blocks

tag	data (χώρος για μεγάλο block)

- Σε μεγάλα cache blocks επωφελούμαστε από την spatial locality.
- Λιγότερος χώρος απαιτείται για tag (με δεδομένη χωρητικότητα της cache)
- Υπερβολικά μεγάλο μέγεθος block σπαταλάει το χώρο της cache
- Τα μεγάλα blocks απαιτούν μεγαλύτερο χρόνο μεταφοράς (transfer time).

Ένας καλός σχεδιασμός απαιτεί συμβιβασμούς!

Μέγεθος Block και Miss Rate



Κανόνας : το μέγεθος του block πρέπει να είναι μικρότερο από την τετραγωνική ρίζα του μεγέθους της cache.

- 1 KB
- 8 KB
- 16 KB
- ◆ 64 KB
- ◆ 256 KB

Miss Rates για Caches διαφορετικού μεγέθους, Associativity & αλγορίθμους αντικατάστασης block

Associativity:	2-way		4-way		8-way	
Μέγεθος	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%

Επίδοση των caches

Για CPU με ένα μόνο επίπεδο (L1) cache και καθόλου καθυστέρηση όταν έχουμε cache hit:

← Με ιδανική μνήμη

Χρόνος CPU = (κύκλοι ρολογιού κατά τη λειτουργία της CPU + κύκλοι ρολογιού λόγω καθυστέρησης από προσπέλαση της μνήμης (Mem stalls))
x χρόνος 1 κύκλου ρολογιού

Mem stalls =

(Αναγνώσεις x miss rate αναγνώσεων x miss penalty αναγνώσεων)
+ (Εγγραφές x miss rate εγγραφών x miss penalty εγγραφών)

Αν τα miss penalties των αναγνώσεων και των εγγραφών είναι ίδια:

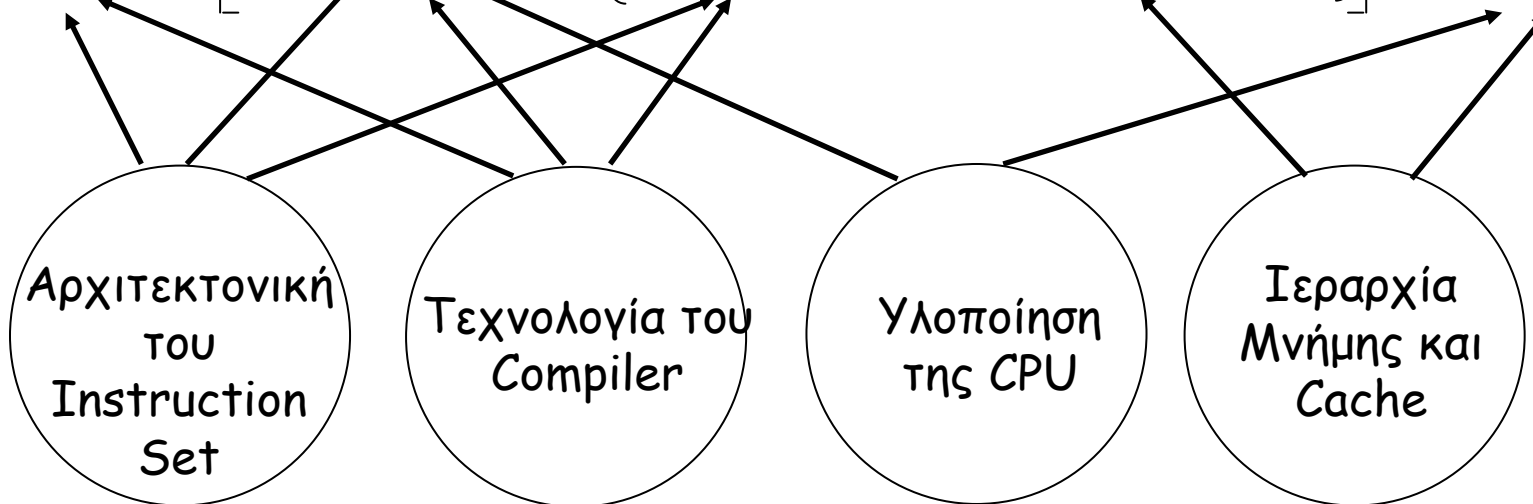
Mem stalls = Προσπελάσεις μνήμης x Miss rate x Miss penalty

Χρόνος εκτέλεσης

$$\text{χρόνος εκτέλεσης} = \text{αριθμός εντολών} \times \frac{\text{κύκλοι}}{\text{εντολή}} \times \text{χρόνος 1 κύκλου}$$

$$= \text{αριθμός εντολών} \times \left[\frac{\text{κύκλοι CPU}}{\text{εντολή}} + \frac{\text{κύκλοι μνήμης}}{\text{εντολή}} \right] \times \text{χρόνος 1 κύκλου}$$

$$= \text{αριθμός εντολών} \times \left[\frac{\text{κύκλοι CPU}}{\text{εντολή}} + \left[\frac{\text{αναφορές}}{\text{εντολή}} \times \frac{\text{κύκλοι μνήμης}}{\text{αναφορά}} \right] \right] \times \text{χρόνος 1 κύκλου}$$



Επίδοση των caches

$CPUtime = \text{Instruction count} \times CPI \times \text{Χρόνος 1 κύκλου ρολογιού}$

$CPI_{\text{execution}} = CPI \text{ με ιδανική μνήμη}$

$CPI = CPI_{\text{execution}} + \text{Mem stalls/εντολή}$

$CPUtime = \text{Instruction Count} \times (CPI_{\text{execution}} + \text{Mem stalls/εντολή}) \times \text{χρόνος 1 κύκλου ρολογιού}$

$\text{Mem stalls/εντολή} =$

$\text{Προσπελάσεις μνήμης/εντολή} \times \text{Miss rate} \times \text{Miss penalty}$

$CPUtime = IC \times (CPI_{\text{execution}} + \text{Προσπελάσεις μνήμης ανά εντολή} \times \text{Miss rate} \times \text{Miss penalty}) \times \text{Χρόνος 1 κύκλου ρολογιού}$

$\text{Misses/εντολή} = \text{Προσπελάσεις μνήμης ανά εντολή} \times \text{Miss rate}$

$CPUtime = IC \times (CPI_{\text{execution}} + \text{Misses/εντολή} \times \text{Miss penalty}) \times \text{Χρόνος 1 κύκλου ρολογιού}(C)$

Παράδειγμα

- Έστω μία CPU λειτουργεί με ρολόι 200 MHz (5 ns/cycle) και cache ενός επιπέδου.
- $CPI_{execution} = 1.1$
- Εντολές: 50% arith/logic, 30% load/store, 20% control
- Υποθέτουμε cache miss rate = 1.5% και miss penalty = 50 cycles.

$$CPI = CPI_{execution} + Mem\ stalls/εντολή$$

$$Mem\ Stalls/εντολή = Mem\ accesses /εντολή \times Miss\ rate \times Miss\ penalty$$

$$Mem\ accesses /εντολή = 1 + 0.3 = 1.3$$

Instruction fetch ↗ ↖ Load/store

$$Mem\ Stalls /εντολή = 1.3 \times 0.015 \times 50 = 0.975$$

$$CPI = 1.1 + 0.975 = 2.075$$

Η ιδανική CPU χωρίς misses είναι $2.075/1.1 = 1.88$ φορές γρηγορότερη

Παράδειγμα

- Στο προηγούμενο παράδειγμα υποθέτουμε ότι διπλασιάζουμε τη συχνότητα του ρολογιού στα 400 MHz. Πόσο γρηγορότερο είναι το μηχάνημα για ίδιο miss rate και αναλογία εντολών;

Δεδομένου ότι η ταχύτητα της μνήμης δεν αλλάζει, το miss penalty καταναλώνει περισσότερους κύκλους CPU:

$$\text{Miss penalty} = 50 \times 2 = 100 \text{ cycles.}$$

$$\text{CPI} = 1.1 + 1.3 \times .015 \times 100 = 1.1 + 1.95 = 3.05$$

$$\begin{aligned} \text{Speedup} &= (\text{CPI}_{\text{old}} \times C_{\text{old}}) / (\text{CPI}_{\text{new}} \times C_{\text{new}}) \\ &= 2.075 \times 2 / 3.05 = 1.36 \end{aligned}$$

Το νέο μηχάνημα είναι μόνο 1.36 φορές ταχύτερο και όχι 2 φορές γρηγορότερο λόγω της επιπλέον επιβάρυνσης των cache misses.

→ CPUs με μεγαλύτερη συχνότητα ρολογιού, έχουν περισσότερους κύκλους/cache miss και μεγαλύτερη επιβάρυνση της μνήμης στο CPI.

2 επίπεδα Cache: L_1 , L_2

CPU

L_1 Cache

Hit Rate = H_1 , Hit time = 1 κύκλος
(καθόλου Stall)

L_2 Cache

Hit Rate = H_2 , Hit time = T_2 κύκλοι

Main Memory

Penalty λόγω προσπέλασης μνήμης, M

Cache 2 επιπέδων

$$\text{CPUtime} = IC \times (\text{CPI}_{\text{execution}} + \text{Mem Stalls/εντολή}) \times C$$

$$\text{Mem Stalls/εντολή} = \text{Mem accesses/εντολή} \times \text{Stalls/access}$$

- Για ένα σύστημα με 2 επίπεδα cache, χωρίς penalty όταν τα δεδομένα βρεθούν στην L_1 cache:

Stalls/memory access =

$$\begin{aligned} & [\text{miss rate } L_1] \times [\text{Hit rate } L_2 \times \text{Hit time } L_2 \\ & + \text{Miss rate } L_2 \times \text{Memory access penalty}] \\ = & (1-H_1) \times H_2 \times T_2 + (1-H_1)(1-H_2) \times M \end{aligned}$$

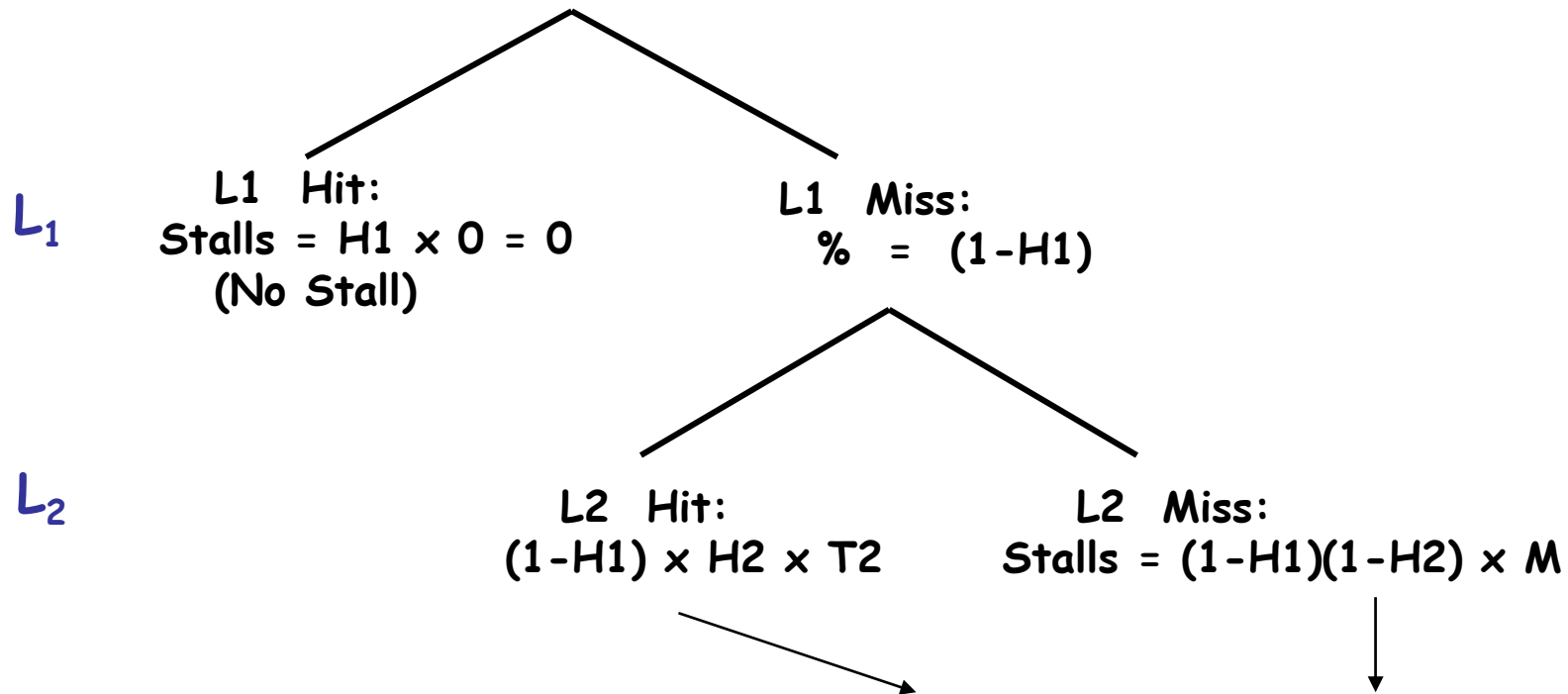
L1 Miss, L2 Hit

L1 Miss, L2 Miss:
Προσπέλαση της Main Memory

Επίδοση της L2 Cache Memory Access Tree

CPU Stalls/Memory Access

CPU Memory Access



Stall cycles per memory access = $(1-H1) \times H2 \times T2 + (1-H1)(1-H2) \times M$

Παράδειγμα L2 Cache

- CPU με $CPI_{execution} = 1.1$ και συχνότητα 500 MHz
- 1.3 memory accesses/εντολή.
- L_1 cache : στα 500 MHz με miss rate 5%
- L_2 cache : στα 250 MHz με miss rate 3%, ($T_2 = 2$ κύκλοι)
- M (Memory access penalty) = 100 κύκλοι. Να βρεθεί το CPI.

$$CPI = CPI_{execution} + Mem\ Stalls/εντολή$$

$$\text{Χωρίς Cache, } CPI = 1.1 + 1.3 \times 100 = 131.1$$

$$\text{Με } L_1 \text{ Cache, } CPI = 1.1 + 1.3 \times 0.05 \times 100 = 7.6$$

$$Mem\ Stalls/εντολή = Mem\ accesses/εντολή \times Stalls/access$$

$$\begin{aligned} Stalls/memory\ access &= (1-H_1) \times H_2 \times T_2 + (1-H_1)(1-H_2) \times M \\ &= 0.05 \times 0.97 \times 2 + 0.05 \times 0.03 \times 100 \\ &= 0.097 + 0.15 = 0.247 \end{aligned}$$

$$Mem\ Stalls/εντολή = Mem\ accesses/εντολή \times Stalls/access = 0.247 \times 1.3 = 0.32$$

$$CPI = 1.1 + 0.32 = 1.42$$

$$Speedup = 7.6/1.42 = 5.35$$

3 επίπεδα Cache

CPU

L1 Cache

Hit Rate = H_1 , Hit time = 1 κύκλος
(καθόλου Stall)

L2 Cache

Hit Rate = H_2 , Hit time = T_2 κύκλοι

L3 Cache

Hit Rate = H_3 , Hit time = T_3

Main Memory

Memory access penalty, M

Επίδοση της L3 Cache

$$CPUtime = IC \times (CPI_{execution} + Mem\ Stalls/εντολή) \times C$$

$$Mem\ Stalls/εντολή = Mem\ accesses / εντολή \times Stalls/access$$

- Για ένα σύστημα με 3 επίπεδα cache, χωρίς penalty όταν τα δεδομένα βρεθούν στην L_1 cache:

$$Stalls/memory\ access =$$

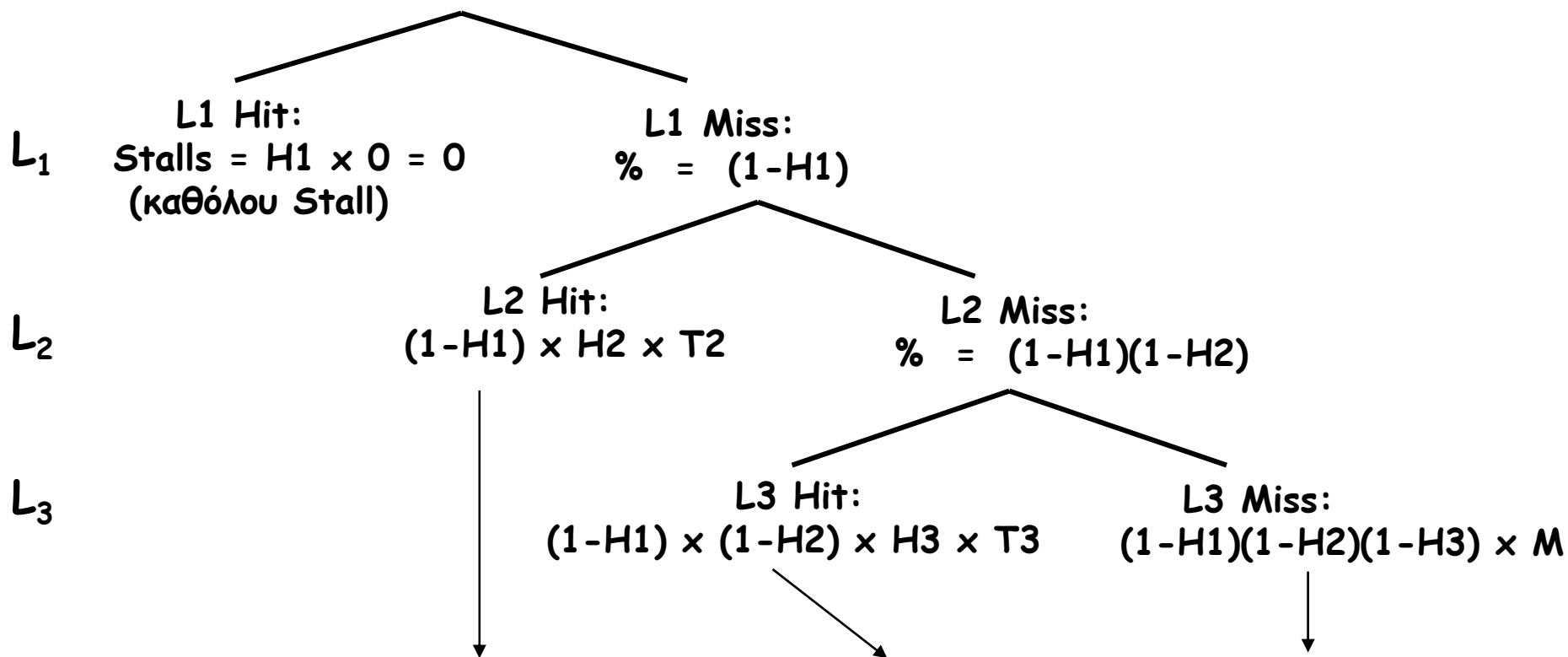
$$\begin{aligned} & [miss\ rate\ L_1] \times [Hit\ rate\ L_2 \times Hit\ time\ L_2 \\ & + Miss\ rate\ L_2 \times (Hit\ rate\ L_3 \times Hit\ time\ L_3 \\ & + Miss\ rate\ L_3 \times Memory\ access\ penalty)] \end{aligned}$$

$$\begin{aligned} = & (1-H1) \times H2 \times T2 \\ & + (1-H1) \times (1-H2) \times H3 \times T3 \\ & + (1-H1)(1-H2) (1-H3) \times M \end{aligned}$$

Επίδοση της L3 Cache Memory Access Tree

CPU Stalls/Memory Access

CPU Memory Access



$$\text{Stalls/memory access} = (1-H1) \times H2 \times T2 + (1-H1) \times (1-H2) \times H3 \times T3 + (1-H1)(1-H2)(1-H3) \times M$$

Παράδειγμα L3 Cache

- CPU με $CPI_{execution} = 1.1$ και συχνότητα 500 MHz
- 1.3 memory accesses/εντολή.
- L_1 cache : στα 500 MHz με miss rate 5%
- L_2 cache : στα 250 MHz με miss rate 3%, ($T_2 = 2$ κύκλοι)
- L_3 cache : στα 100 MHz με miss rate 1.5%, ($T_3 = 5$ κύκλοι)
- Memory access penalty, $M = 100$ cycles. Να βρείτε το CPI.

χωρίς Cache, $CPI = 1.1 + 1.3 \times 100 = 131.1$

Με L_1 Cache, $CPI = 1.1 + 1.3 \times 0.05 \times 100 = 7.6$

Με L_2 Cache, $CPI = 1.1 + 1.3 \times (0.05 \times 0.97 \times 2 + 0.05 \times 0.03 \times 100) = 1.42$

$$CPI = CPI_{execution} + Mem\ Stalls/εντολή$$

$$Mem\ Stalls/εντολή = Mem\ accesses/εντολή \times Stall\ cycles/access$$

$$\begin{aligned} Stalls/memory\ access &= (1-H_1) \times H_2 \times T_2 + (1-H_1) \times (1-H_2) \times H_3 \times T_3 + (1-H_1)(1-H_2)(1-H_3) \times M \\ &= 0.05 \times 0.97 \times 2 + 0.05 \times 0.03 \times 0.985 \times 5 + 0.05 \times 0.03 \times 0.015 \times 100 \\ &= 0.097 + 0.0075 + 0.00225 = 0.107 \end{aligned}$$

$$CPI = 1.1 + 1.3 \times 0.107 = 1.24$$

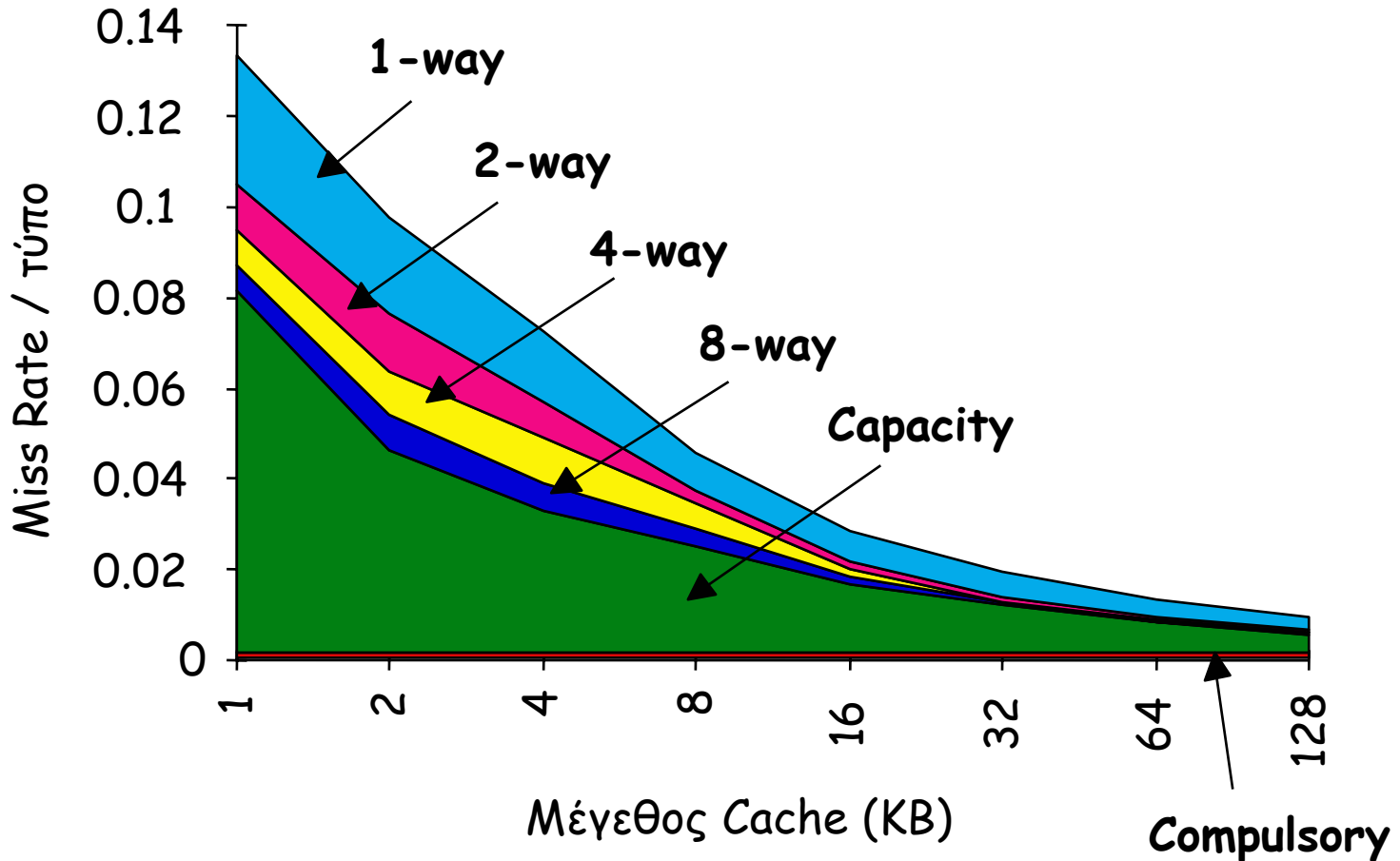
$$Speedup\ σε\ σχέση\ με\ L1\ μόνο = 7.6/1.24 = 6.12$$

$$Speedup\ σε\ σχέση\ με\ L1, L2 = 1.42/1.24 = 1.15$$

Είδη των Cache Misses: 3C's

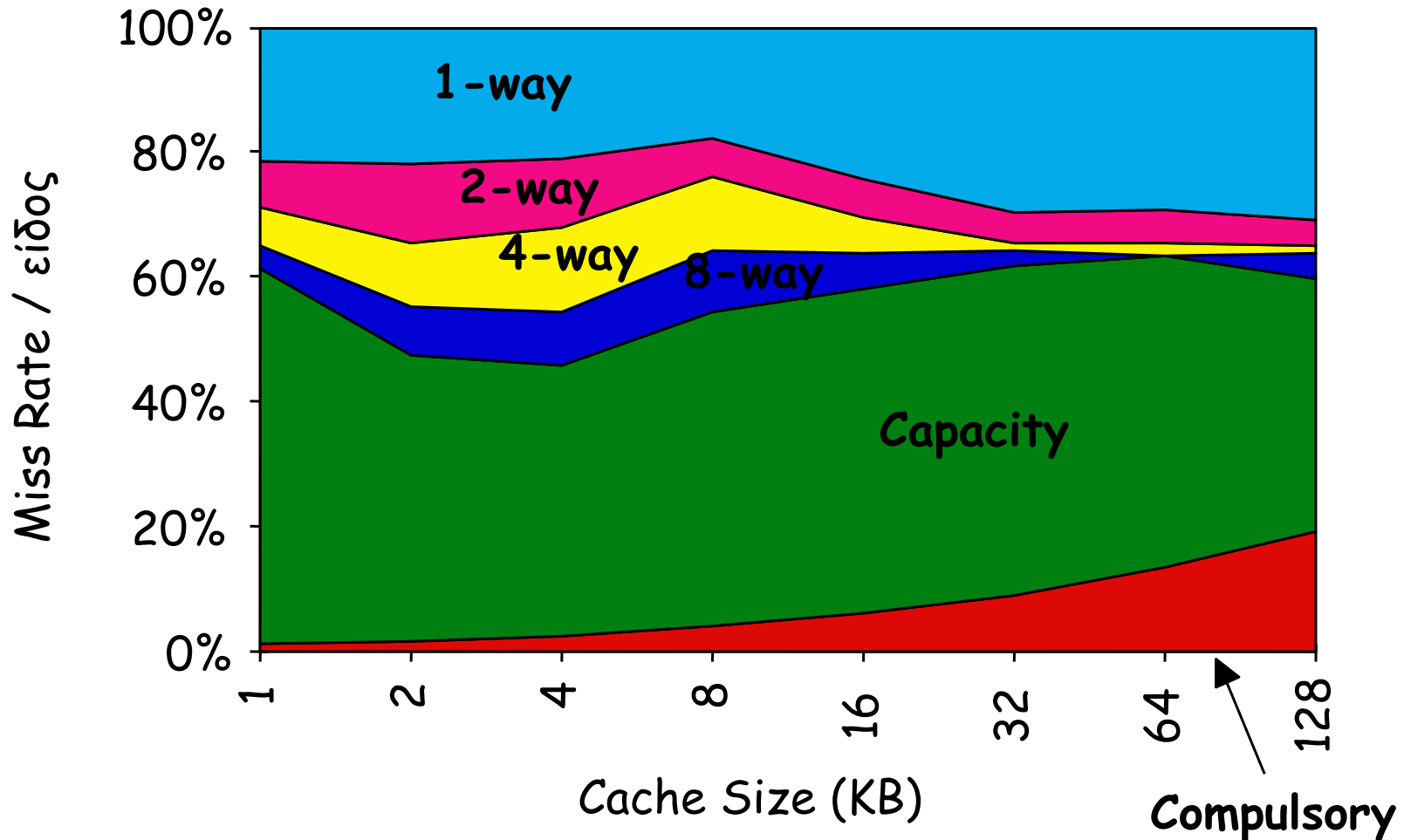
- 1 Compulsory:** Συμβαίνουν κατά την πρώτη πρόσβαση σε ένα block. Το block πρέπει να κληθεί από χαμηλότερα επίπεδα μνήμης και να τοποθετηθεί στην cache (αποκαλούνται και cold start misses ή first reference misses).
- 2 Capacity:** Τα blocks απομακρύνονται από την cache επειδή δεν χωράνε σε αυτήν όλα όσα απαιτούνται κατά την εκτέλεση ενός προγράμματος (το σύνολο των δεδομένων που χειρίζεται ένα πρόγραμμα είναι πολύ μεγαλύτερο από την χωρητικότητα της cache).
- 3 Conflict:** Στην περίπτωση των set associative ή direct mapped caches, conflict misses έχουμε όταν πολλά blocks απεικονίζονται στο ίδιο set (αποκαλούνται και collision misses ή interference misses).

Τα 3 Cs των Cache: Απόλυτα Miss Rates (SPEC92)



Τα 3 Cs των Cache:

Σχετικά Miss Rates (SPEC92)



Βελτιστοποίηση της επίδοσης της Cache

Πώς:

- Περιορισμός του Miss Rate
- Μείωση του Cache Miss Penalty
- Μείωση του χρόνου για Cache Hit

Βελτιστοποίηση της επίδοσης της Cache

• Τεχνικές μείωσης του Miss Rate:

- * Μεγαλύτερο μέγεθος block
- * Μεγαλύτερου βαθμού associativity
- * Victim caches
- * Compiler-controlled prefetching
- * Αύξηση της χωρητικότητας της cache
- * Pseudo-associative Caches
- * Hardware/Software prefetching εντολών-δεδομένων
- * βελτιστοποιήσεις στον Compiler

• Τεχνικές μείωσης του Cache Miss Penalty:

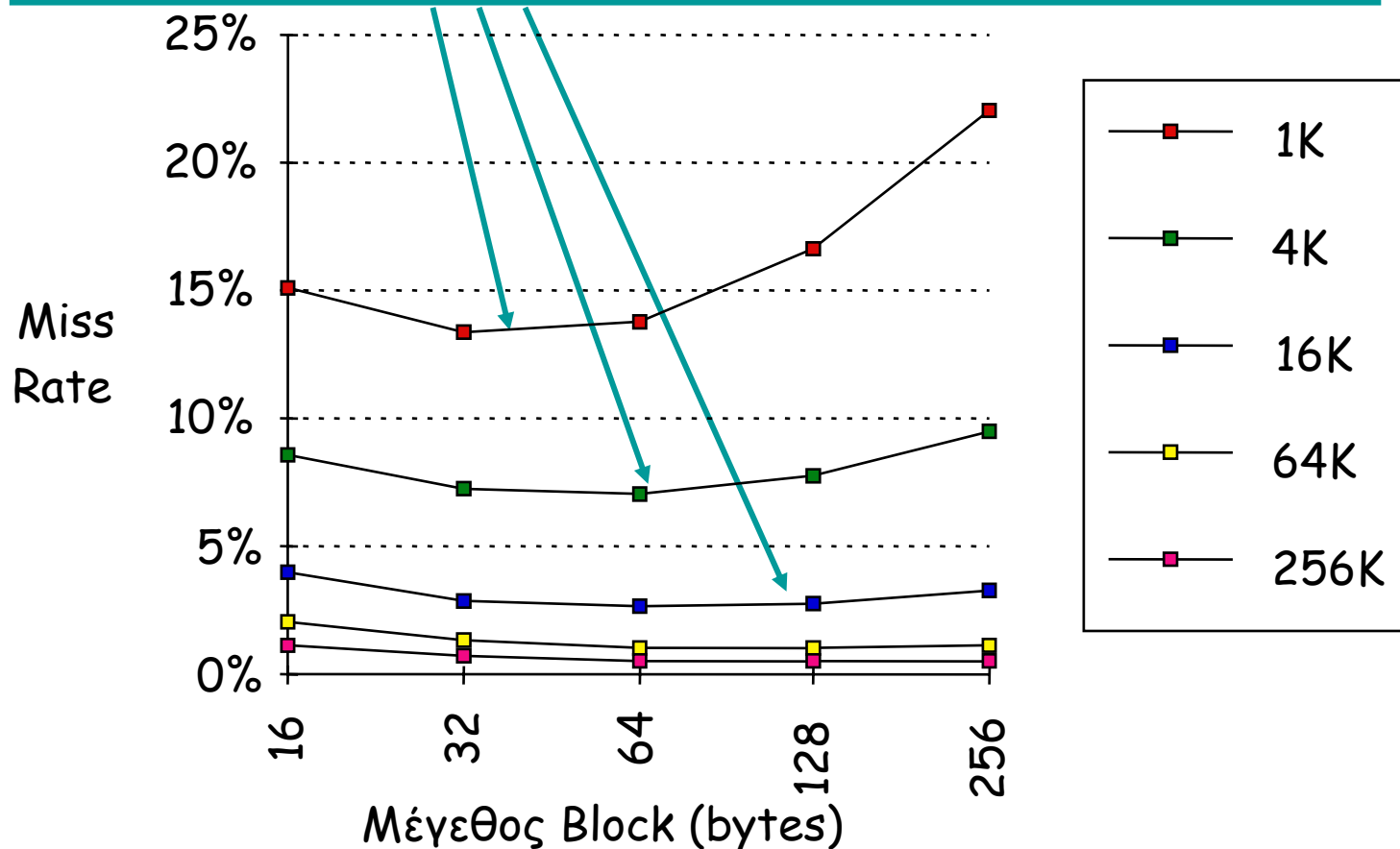
- * Cache 2ου επιπέδου (L_2)
- * Early restart and critical word first
- * Προτεραιότητα στα read misses έναντι των writes
- * merging write buffers
- * Non-blocking caches

• Τεχνικές μείωσης του Cache Hit Time:

- * Μικρές και απλές caches
- * Αποφυγή της μετάφρασης των διευθύνσεων κατά τη διάρκεια του indexing
- * Pipelining writes για γρήγορα write hits

Τεχνικές μείωσης του Miss Rate: Μεγαλύτερο μέγεθος Block

- Το μεγάλο μέγεθος block βελτιώνει την επίδοση της cache επειδή επωφελούμαστε από την spatial locality
- Για δεδομένο μέγεθος cache, μεγαλύτερο μέγεθος block σημαίνει λιγότερα cache block frames
- Η επίδοση βελτιώνεται μέχρι το σημείο όπου λόγω του μικρού αριθμού των cache block frames αυξάνονται τα conflict misses και επομένως και το συνολικό cache miss rate



Τεχνικές μείωσης του Miss Rate:

Μεγαλύτερο μέγεθος cache

- Με την αύξηση του μεγέθους της cache προκαλείται:
 - αύξηση του hit time
 - αύξηση του κατασκευαστικού κόστους
- Αυτή η τεχνική δημοφιλής σε off-chip caches
- Σημείωση : οι L2,L3 caches σήμερα έχουν μέγεθος όσο ήταν η Κύρια Μνήμη πριν 10 χρόνια

Τεχνικές μείωσης του Miss Rate: Μεγαλύτερου βαθμού Associativity

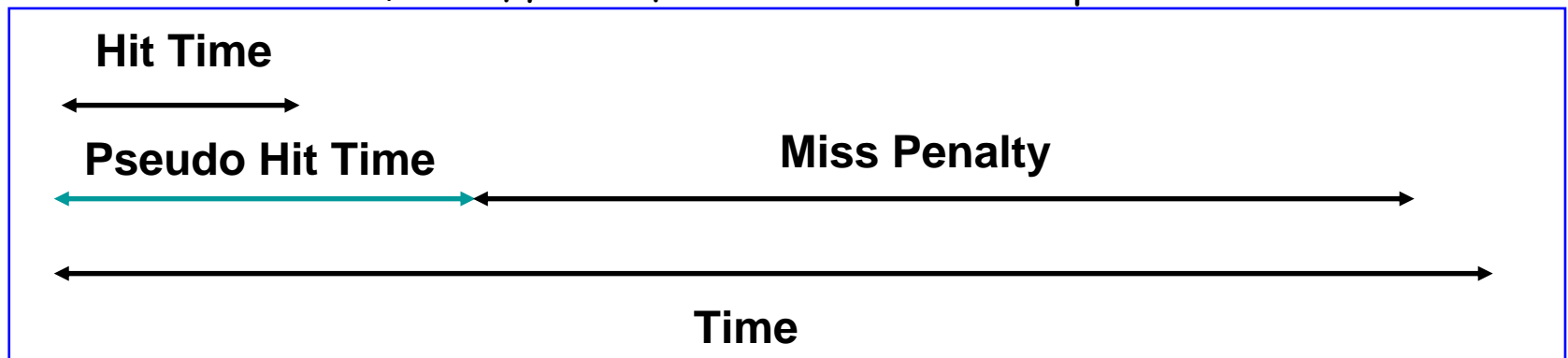
Παράδειγμα: Μέσος χρόνος πρόσβασης στη μνήμη vs. Miss Rate

Cache Size (KB)	Associativity			
	1-way	2-way	4-way	8-way
1	2.33	2.15	2.07	2.01
2	1.98	1.86	1.76	1.68
4	1.72	1.67	1.61	1.53
8	1.46	<u>1.48</u>	<u>1.47</u>	1.43
<u>16</u>	<u>1.29</u>	<u>1.32</u>	<u>1.32</u>	<u>1.32</u>
<u>32</u>	<u>1.20</u>	<u>1.24</u>	<u>1.25</u>	<u>1.27</u>
<u>64</u>	<u>1.14</u>	<u>1.20</u>	<u>1.21</u>	<u>1.23</u>
<u>128</u>	<u>1.10</u>	<u>1.17</u>	<u>1.18</u>	<u>1.20</u>

(Μπλε σημαίνει ότι ο μέσος χρόνος δεν βελτιώνεται με την αύξηση του associativity)

Τεχνικές μείωσης του Miss Rate: Pseudo-Associative Cache

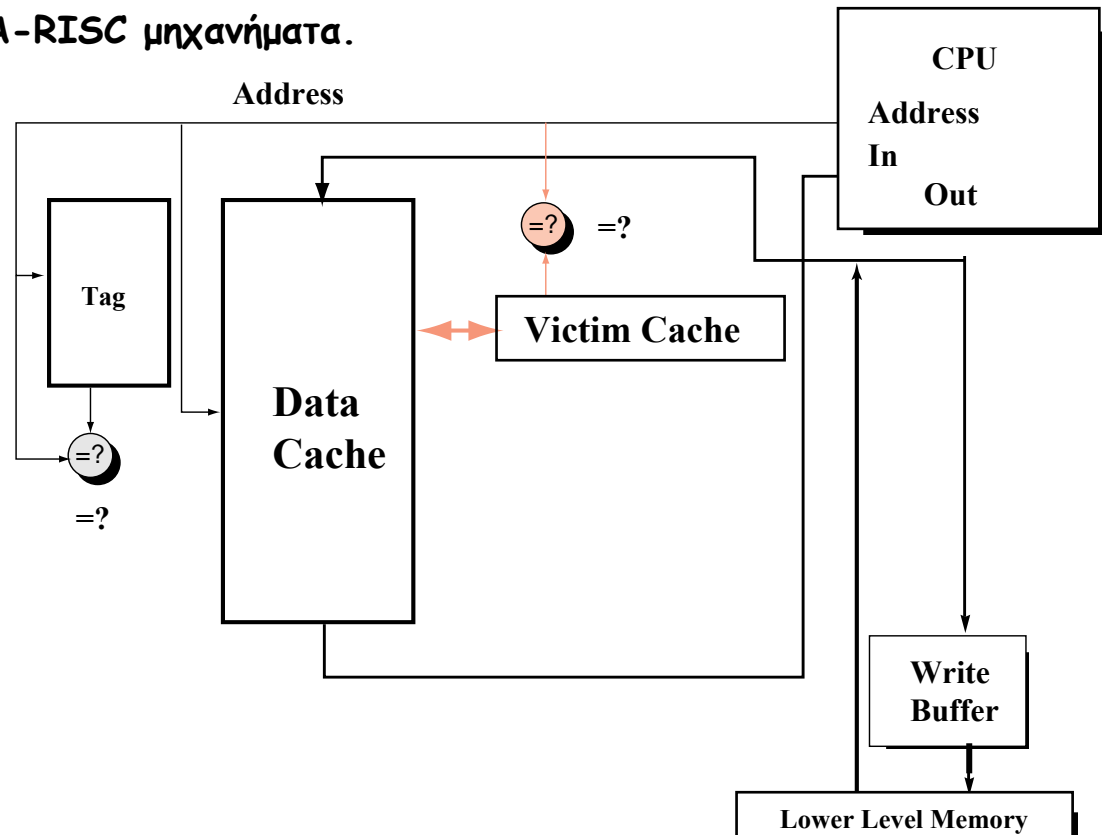
- Συνδυάζει το μικρό χρόνο αναζήτησης (hit time) των Direct Mapped caches και το μικρότερο αριθμό των conflict misses στις 2-way set-associative caches.
- Η cache διαιρείται σε δύο τμήματα: Όταν έχουμε cache miss, ελέγχουμε το άλλο μισό της cache για να δούμε αν τα δεδομένα που αναζητάμε βρίσκονται εκεί. Στην περίπτωση αυτή έχουμε ένα pseudo-hit (slow hit)
- Ο ευκολότερος τρόπος υλοποίησης είναι η αναστροφή του most significant bit στο πεδίο index για να βρίσκουμε το άλλο block στο "pseudo set".



- Μειονέκτημα: είναι δύσκολη η αποδοτική υλοποίηση του CPU pipelining αν το L_1 cache hit παίρνει 1 ή 2 κύκλους.
 - Χρησιμοποιείται καλύτερα σε caches που δεν είναι συνδεδεμένες απευθείας με τη CPU (L_2 cache).
 - Χρησιμοποιείται στην L_2 cache του MIPS R1000. Παρόμοια είναι και η L_2 του UltraSPARC.

Τεχνικές μείωσης του Miss Rate: Victim Caches

- Τα δεδομένα που απομακρύνονται από την cache τοποθετούνται σε έναν μικρό πρόσθετο buffer (victim cache).
- Σε περίπτωση cache miss ελέγχουμε το περιεχόμενο της victim cache πριν τα αναζητήσουμε στην κύρια μνήμη
- Jouppi [1990]: Μία victim cache 4 εισόδων αποτρέπει το 20% έως 95% των conflict misses για μία 4 KB direct mapped cache
- Χρησιμοποιείται σε Alpha, HP PA-RISC μηχανήματα.



Τεχνικές μείωσης του Miss Rate: Hardware/Compiler Prefetching εντολών και δεδομένων

- Φέρνουμε εντολές ή δεδομένα στην cache ή σε έναν εξωτερικό buffer (prefetch) πριν ζητηθούν από τη CPU.
- Παράδειγμα: Ο Alpha APX 21064 φέρνει 2 blocks σε κάθε miss: Το ζητούμενο block τοποθετείται στην cache και το αμέσως επόμενο σε έναν stream buffer εντολών.
- Η ίδια λογική εφαρμόζεται και στις προσπελάσεις δεδομένων με έναν data buffer.
- Μπορεί να επεκταθεί και για πολλαπλούς stream buffers δεδομένων σε διαφορετικές διευθύνσεις (4 streams βελτιώνουν το data hit rate κατά 43%).
- Αποδεικνύεται ότι, σε ορισμένες περιπτώσεις, 8 stream buffers οι οποίοι χειρίζονται δεδομένα ή εντολές, μπορούν να αποτρέψουν το 50-70% των συνολικών misses.

Τεχνικές μείωσης του Miss Rate: Compiler Optimizations

Βελτιστοποίηση του κώδικα επιτυγχάνοντας τοπικότητα κατά την προσπέλαση δεδομένων :

- *Αναδιοργάνωση των procedures* στη μνήμη για τη μείωση των conflict misses.
- *Merging Arrays*: Βελτίωση της spatial locality με έναν πίνακα δεδομένων αντί 2 πίνακες.
- *Loop Interchange*: Αλλαγή της σειράς φωλιάσματος των βρόχων για να προσπελάσουμε τα δεδομένα με την ίδια σειρά όπως αποθηκεύονται στη μνήμη.
- *Loop Fusion*: Συνδυασμός 2 ή περισσότερων ανεξάρτητων βρόχων που περιέχουν τους ίδιους βρόχους και κάποιες κοινές μεταβλητές.
- *Blocking*: Βελτίωση της temporal locality προσπελάνοντας ένα τμήμα μόνο των δεδομένων επαναληπτικά αντί να διατρέχουμε ολόκληρες τις γραμμές ή τις στήλες.

Τεχνικές μείωσης του Miss Rate: Compiler Optimizations

Merging Arrays

```
/* Before: 2 sequential arrays */  
int val[SIZE];  
int key[SIZE];  
  
/* After: 1 array of structures */  
struct merge {  
    int val;  
    int key;  
};  
struct merge merged_array[SIZE];
```

Merging :

- Μειώνονται τα conflicts μεταξύ των στοιχείων των val και key
- Βελτίωση του spatial locality

Τεχνικές μείωσης του Miss Rate: Compiler Optimizations

Loop Interchange

```
/* Before */  
for (k = 0; k < 100; k = k+1)  
    for (j = 0; j < 100; j = j+1)  
        for (i = 0; i < 5000; i = i+1)  
            x[i][j] = 2 * x[i][j];
```

```
/* After */  
for (k = 0; k < 100; k = k+1)  
    for (i = 0; i < 5000; i = i+1)  
        for (j = 0; j < 100; j = j+1)  
            x[i][j] = 2 * x[i][j];
```

Η προσπέλαση δεδομένων που βρίσκονται σε συνεχόμενες θέσεις μνήμης και όχι με απόσταση 100 λέξεων βελτιώνει στο παράδειγμά μας την spatial locality.

Τεχνικές μείωσης του Miss Rate: Compiler Optimizations

Loop Fusion

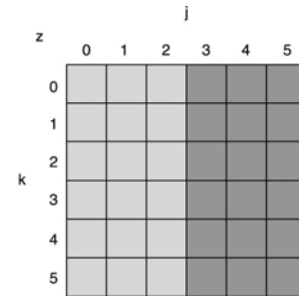
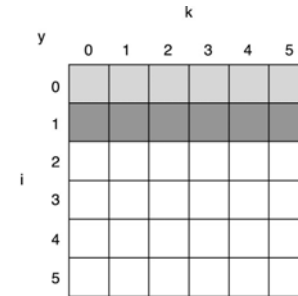
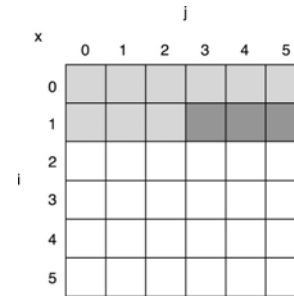
```
/* Before */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    a[i][j] = 1/b[i][j] * c[i][j];
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
    d[i][j] = a[i][j] + c[i][j];
/* After */
for (i = 0; i < N; i = i+1)
  for (j = 0; j < N; j = j+1)
  {
    a[i][j] = 1/b[i][j] * c[i][j];
    d[i][j] = a[i][j] + c[i][j];
  }
```

- Αντί 2 misses/access στα a & c τελικά 1 miss/access
- Βελτίωση της spatial locality

Τεχνικές μείωσης του Miss Rate: Compiler Optimizations

Blocking

```
/* Before */  
for (i = 0; i < N; i = i+1)  
  for (j = 0; j < N; j = j+1)  
    { r = 0;  
      for (k = 0; k < N; k = k+1){  
        r = r + y[i][k]*z[k][j];  
        x[i][j] = r;  
      };  
    };
```



- **Οι 2 εσωτερικότεροι βρόχοι:**
 - Προσπελούν όλα τα $N \times N$ στοιχεία του z []
 - Προσπελούν επαναληπτικά τα N στοιχεία της 1 γραμμής του y []
 - Εγγραφή των N στοιχείων της 1 γραμμής του x []
- **Capacity Misses :** είναι συνάρτηση του N και του μεγέθους της Cache:
 - $3 N \times N \times 4 \leq \text{μεγέθους της Cache} \Rightarrow$ καθόλου capacity misses
- **Βασική ιδέα:** αναζητάμε τον $B \times B$ υποπίνακα που χωράει στην cache

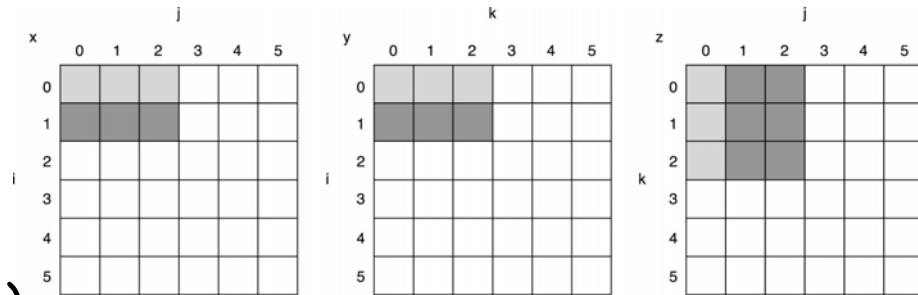
Τεχνικές μείωσης του Miss Rate: Compiler Optimizations

Blocking

```

/* After */
for (jj = 0; jj < N; jj = jj+B)
  for (kk = 0; kk < N; kk = kk+B)
    for (i = 0; i < N; i = i+1)
      for (j = jj; j < min(jj+B-1,N); j = j+1)
        { r = 0;
          for (k = kk; k < min(kk+B-1,N); k = k+1) {
            r = r + y[i][k]*z[k][j];
            x[i][j] = x[i][j] + r;
          };
        };

```



- B : *Blocking Factor*
- Capacity Misses αντί $2N^3 + N^2 \rightarrow 2N^3/B + N^2$
- Πιθανών να επηρεάζονται και τα conflict misses

Τεχνικές μείωσης του Miss Penalty: Early Restart και Critical Word First

- Δεν περιμένουμε να μεταφερθεί το πλήρες block στην cache πριν την επανεκκίνηση της CPU:
 - Early restart: Αμέσως μόλις φορτωθεί η ζητούμενη λέξη του block, αποστέλλεται στη CPU και συνεχίζεται η επεξεργασία των δεδομένων από αυτήν.
 - Critical Word First: Φορτώνεται πρώτη από όλο το block η ζητούμενη λέξη και αποστέλλεται στη CPU αμέσως μόλις φτάσει.
 - Έτσι η CPU συνεχίζει την επεξεργασία ενώ οι υπόλοιπες λέξεις του block μεταφέρονται από την κύρια μνήμη.
- Είναι συνήθως χρήσιμες όταν το μέγεθος των cache block είναι μεγάλο.
- Τα προγράμματα με καλή spatial locality ζητούν δεδομένα που βρίσκονται σε συνεχόμενες θέσεις μνήμης και δεν επωφελούνται από την τεχνική του early restart.

Τεχνικές μείωσης του Miss Penalty:

Προτεραιότητα στα Read Misses έναντι των Writes

- Στις write-through caches με write buffers παρουσιάζεται πρόβλημα με τις συγκρούσεις RAW κατά την ανάγνωση από την κύρια μνήμη σε περίπτωση που έχουμε cache miss:
 - Ο write buffer κρατά τα προσφάτως τροποποιημένα δεδομένα που χρειάζονται για την ανάγνωση.
 - Μία λύση είναι απλά να περιμένουμε μέχρι να αδειάσει ο write buffer, αυξάνοντας έτσι το miss penalty (σε παλιούς MIPS 1000 κατά 50%).
 - Ελέγχουμε τα περιεχόμενα του write buffer πριν την ανάγνωση: αν δεν υπάρχουν εκεί τα ζητούμενα δεδομένα, πρέπει να τα καλέσουμε από την κύρια μνήμη.
- Στις write-back caches, για ένα read miss αντικαθιστάται το block αν είναι dirty:
 - Συνήθως: Πρώτα μεταφέρεται το dirty block στη μνήμη και στη συνέχεια πραγματοποιείται η ανάγνωση.
 - Διαφορετικά: Αντιγράφεται το dirty block σε έναν write buffer, στη συνέχεια πραγματοποιείται η ανάγνωση, και τέλος η εγγραφή.
 - Η CPU καθυστερεί λιγότερο γιατί ξεκινάει την επεξεργασία δεδομένων αμέσως μετά την ανάγνωση.

Τεχνικές μείωσης του Miss Penalty: Merging write buffers

- Ένα cache block frame διαιρείται σε sub-blocks.
- Υπάρχει ένα valid bit ανά sub-block στα cache block frames.
- Δε χρειάζεται να φορτώσουμε ένα ολόκληρο block στην περίπτωση miss αλλά μόνο το ζητούμενο sub-block.

Διεύθυνση
εγγραφής

		V		V		V	
100	1	Mem[100]	0		0		0
108	1	Mem[108]	0		0		0
116	1	Mem[116]	0		0		0
124	1	Mem[124]	0		0		0

Διεύθυνση
εγγραφής

		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1
	0		0		0		0
	0		0		0		0
	0		0		0		0

κάθε buffer χωράει 4
λέξεις των 64-bit.

Μόνο στο 2ο σχήμα
αξιοποιούνται

Τεχνικές μείωσης του Miss Penalty: Non-Blocking Caches

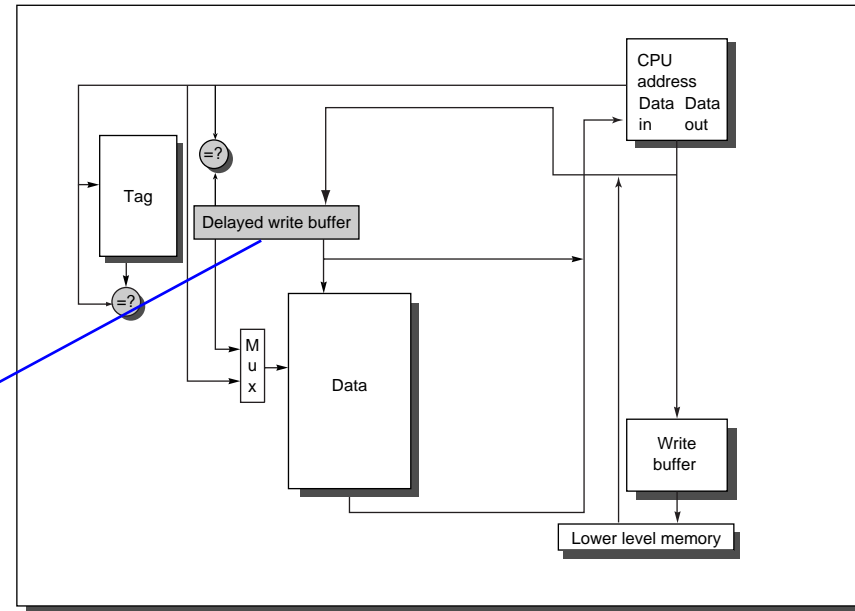
Οι Non-blocking caches ή lockup-free caches επιτρέπουν στις data caches να αποστέλλουν δεδομένα που περιέχουν (cache hits) όσο διεκπεραιώνεται ένα miss:

- Απαιτείται out-of-order εκτέλεση των εντολών από τη CPU.
- "hit under miss" : μειώνει το effective miss penalty γιατί συνεχίζεται η επεξεργασία δεδομένων από τη CPU αντί να αγνοούνται οι αιτήσεις για νέα δεδομένα.
- "hit under multiple miss" ή "miss under miss" : μπορεί να προσφέρει επιπλέον μείωση του effective miss penalty by επικαλύπτοντας τα πολλαπλά misses.
- Αυξάνεται σημαντικά η πολυπλοκότητα του cache controller αφού μπορεί να υπάρχουν πολλές μη διεκπεραιωμένες προσπελάσεις στη μνήμη.
- Απαιτεί πολλαπλά memory banks ώστε να εξυπηρετούνται πολλαπλές προσπελάσεις στη μνήμη.
- Παράδειγμα: Intel Pentium Pro/III επιτρέπει να εκκρεμούν μέχρι και 4 misses.

Τεχνικές μείωσης του Hit Time : Pipelined Writes

- Ο έλεγχος του tag και η ενημέρωση της cache -από την προηγούμενη εντολή- μπορεί να γίνονται ταυτόχρονα (pipeline) αν υλοποιηθούν ως διαφορετικά στάδια
- Μόνο STORES μπορούν να υλοποιηθούν pipeline: πρέπει να αδειάσει ο buffer πριν από ένα miss

Store r2, (r1)	Check r1
Add	--
Sub	--
Store r4, (r3)	M[r1] ← r2 & check r3



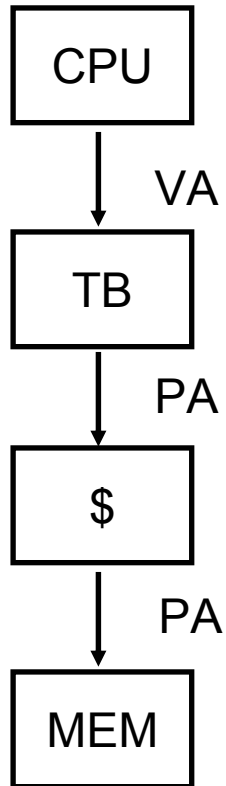
- "Delayed Write Buffer": which must be checked on reads; either complete write or read from buffer

Τεχνικές μείωσης του Hit Time :

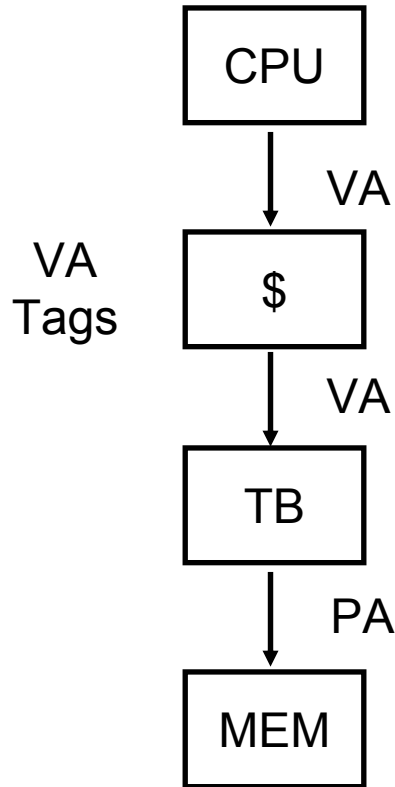
Avoiding Address Translation

- Αποστολή της virtual address στην cache: Ονομάζεται Virtual Addressed Cache ή απλά Virtual Cache vs. Physical Cache
 - Κάθε φορά που αλλάζουμε διεργασία η cache πρέπει να καθαρίζεται (flushed), διαφορετικά θα επιστρέψει λανθασμένα hits
 - Κόστος : χρόνος flush + "compulsory" misses λόγω του αδειάσματος της cache
 - Χειρισμός των aliases (αποκαλούνται και synonyms):
2 διαφορετικές virtual addresses αντιστοιχίζονται στην ίδια physical address
 - I/O πρέπει να επικοινωνεί με την cache, επομένως χρειάζονται οι virtual addresses
- Λύση για τα aliases:
 - Το HW εγγυάται ότι ο συνδυασμός index field & direct mapped είναι μοναδικός : page coloring
- Λύση για το cache flush:
 - Προσθέτουμε μία process identifier tag η οποία αναγνωρίζει τη διεργασία καθώς και τις διευθύνσεις της διεργασίας: δεν επιστρέφεται hit από λάθος διεργασία

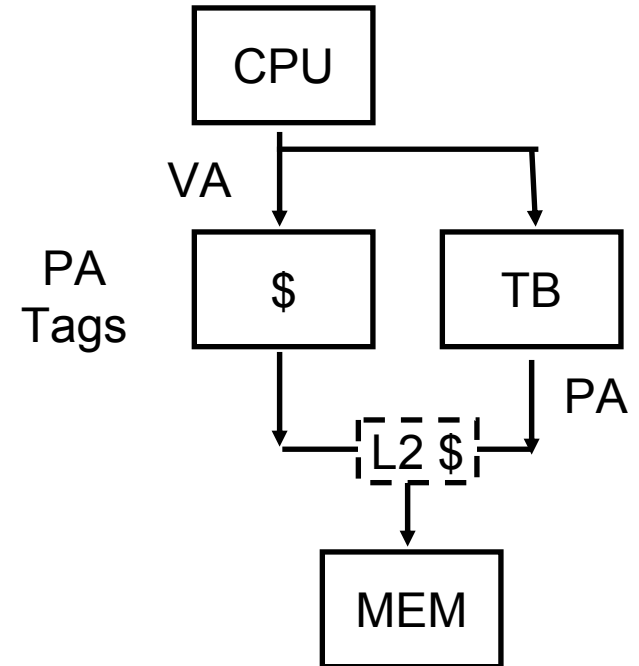
Τεχνικές μείωσης του Hit Time : Virtually Addressed Caches



Συμβατική
Οργάνωση



Virtually Addressed Cache
Μετάφραση μόνο σε miss
Σηνοηγη προβλήματα

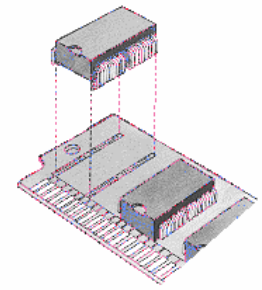


Επικάλυψη της \$
προσπέλασης με VA
μετάφραση: Απαιτείται
δείκτης στην \$ index για
να παραμένει σταθερό
κατά τη μετάφραση

Σύνοψη

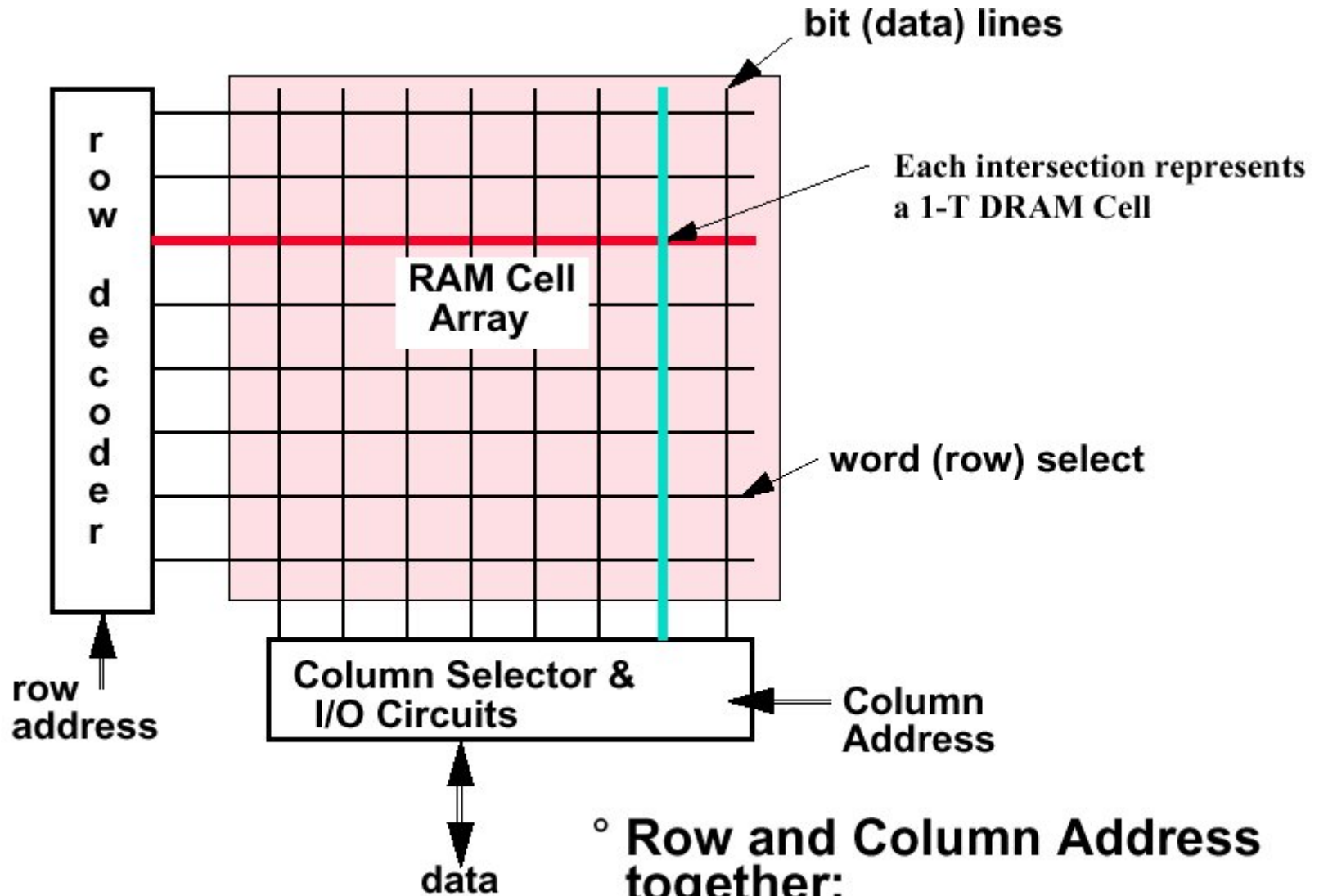
	Τεχνική	MR	MP	HT	Complexity
Miss rate	Μεγαλύτερο μέγεθος Block	+	-		0
	Υψηλότερη Associativity	+		-	1
	Victim Caches	+			2
	Pseudo-Associative Caches	+			2
	HW Prefetching of Instr/Data	+			2
	Compiler Controlled Prefetching	+			3
	Compiler Reduce Misses	+			0
Miss Penalty	Προτεραιότητα στα Read Misses		+		1
	Subblock Placement		+	+	1
	Early Restart & Critical Word 1st		+		2
	Non-Blocking Caches		+		3
	Second Level Caches		+		2
Hit time	Small & Simple Caches	-		+	0
	Avoiding Address Translation			+	2
	Pipelining Writes			+	1

Κύρια Μνήμη (Main Memory)



- Η κύρια μνήμη γενικώς χρησιμοποιεί Dynamic RAM (DRAM), στην οποία χρησιμοποιείται ένα transistor για την αποθήκευση ενός bit, αλλά απαιτεί μία περιοδική ανανέωση των δεδομένων, διαβάζοντας όλες τις σειρές (~κάθε 8 msec).
- Η Static RAM μπορεί να χρησιμοποιηθεί αν το επιπρόσθετο κόστος, η χαμηλή πυκνότητα, και η κατανάλωση ενέργειας είναι ανεκτές (π.χ. Cray Vector Supercomputers).
- Η επίδοση της κύριας μνήμης επηρεάζεται από :
 - **Memory latency:** Επηρεάζει το cache miss penalty και μετριέται από:
 - **Access time:** Ο χρόνος που μεσολαβεί μεταξύ μίας αίτησης προς τη κύρια μνήμη και της στιγμής που η απαιτούμενη πληροφορία είναι διαθέσιμη στην cache/CPU.
 - **Cycle time:** Ο ελάχιστος χρόνος μεταξύ διαδοχικών αιτήσεων προς τη μνήμη (μεγαλύτερος από τον access time στη DRAM για να επιτρέψει στις γραμμές διευθύνσεων να παραμένουν σταθερές)
 - **Memory bandwidth:** Ο ρυθμός μεταφοράς δεδομένων μεταξύ κύριας μνήμης και cache/CPU.

Οργάνωση της DRAM



◦ Row and Column Address together:

- Select 1 bit a time

Τεχνικές Βελτιστοποίησης του Memory Bandwidth

- Ευρύτερη Κύρια Μνήμη:

Το εύρος της μνήμης αυξάνεται κατά έναν αριθμό λέξεων (συνήθως κατά το μέγεθος ενός cache block επιπέδου 2).

⇒ Το Memory bandwidth είναι ανάλογο του εύρους της μνήμης.

π.χ. Διπλασιάζοντας το εύρος της cache, διπλασιάζεται και το memory bandwidth

- Απλή Interleaved Memory:

Η μνήμη οργανώνεται σε έναν αριθμό από banks καθένα με εύρος 1 λέξης.

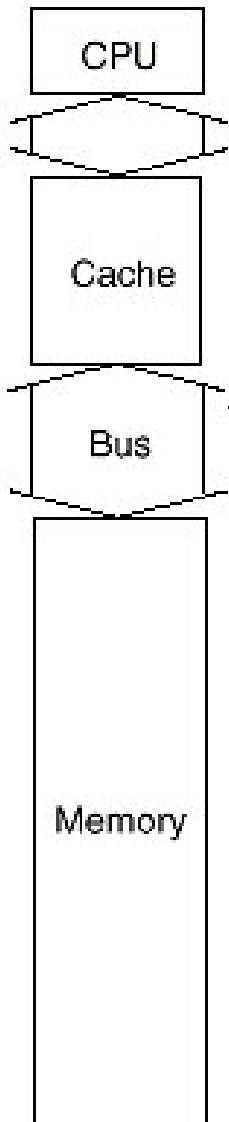
- Ταυτόχρονες αναγνώσεις ή εγγραφές πολλών λέξεων επιτυγχάνονται με αποστολή διευθύνσεων μνημών σε πολλά memory banks σε μία φορά.

- Interleaving factor: Αναφέρεται στην αντιστοίχιση των διευθύνσεων μνήμης στα memory banks.

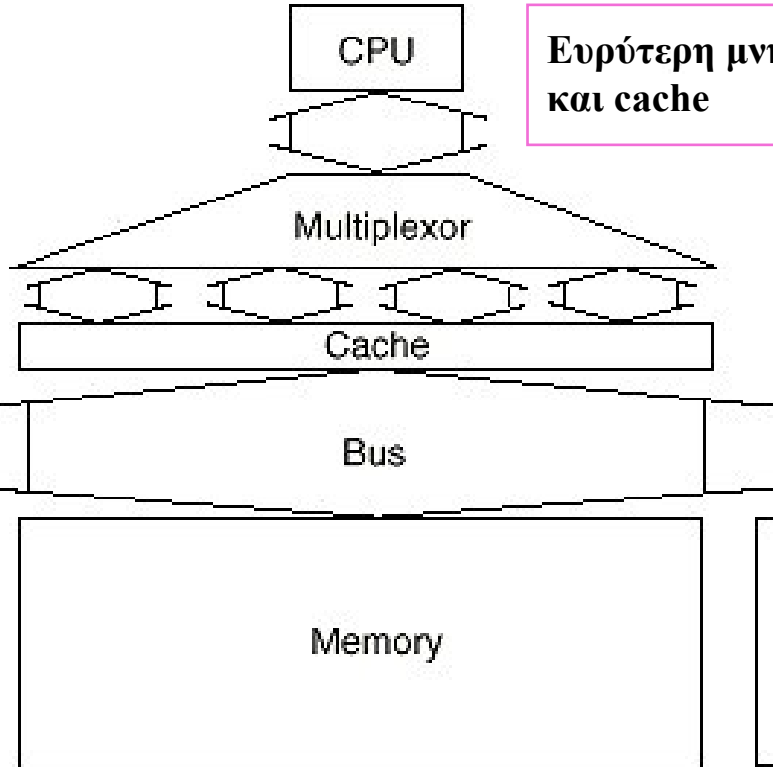
π.χ. χρησιμοποιώντας 4 banks, bank 0 έχει όλες τις λέξεις των οποίων οι διευθύνσεις είναι:

$$(\text{διεύθυνση λέξης}) \pmod{4} = 0$$

(a) One-word-wide memory organization

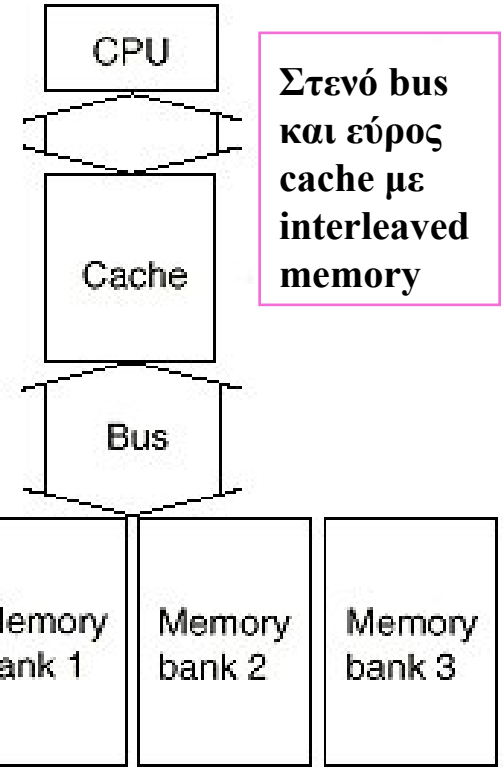


(b) Wide memory organization



Ευρύτερη μνήμη, bus και cache

(c) Interleaved memory organization



Στενό bus και εύρος cache με interleaved memory

3 παραδείγματα εύρους bus, memory, και memory interleaving για να επιτύχουμε μεγαλύτερο memory bandwidth

Ο απλούστερος σχεδιασμός: Όλα έχουν το μέγεθος μίας λέξης

Memory Width, Interleaving: Παράδειγμα

Δίνεται ένα σύστημα με τις ακόλουθες παραμέτρους:

Μέγεθος Cache Block = 1 word, Memory bus width = 1 word, Miss rate = 3%

Miss penalty = 32 κύκλους :

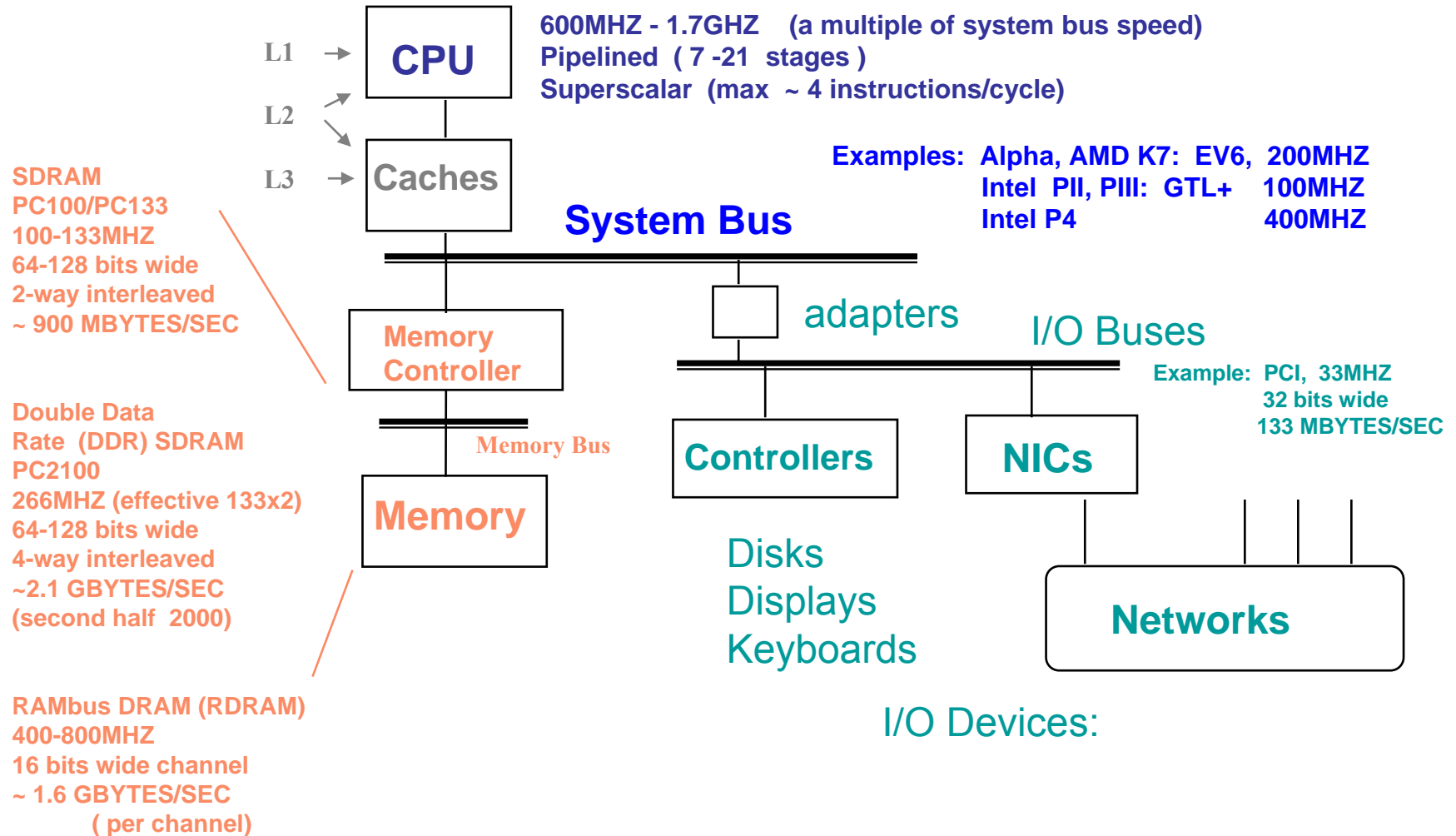
(4 κύκλοι για αποστολή της διεύθυνσης, 24 κύκλοι access time / λέξη, 4 κύκλοι για αποστολή μιας λέξης)

Memory access / εντολή = 1.2 Ιδανικό execution CPI (αγνοώντας τα cache misses) = 2

Miss rate (μέγεθος block=2 word) = 2% Miss rate (μέγεθος block=4 words) = 1%

- Το CPI του μηχανήματος με blocks της 1 λέξης = $2 + (1.2 \times .03 \times 32) = 3.15$
- Μεγαλώνοντας το μέγεθος του block σε 2 λέξεις δίνει το ακόλουθο CPI:
 - 32-bit bus και memory, καθόλου interleaving = $2 + (1.2 \times .02 \times 2 \times 32) = 3.54$
 - 32-bit bus και memory, interleaved = $2 + (1.2 \times .02 \times (4 + 24 + 8)) = 2.86$
 - 64-bit bus και memory, καθόλου interleaving = $2 + (1.2 \times .02 \times 1 \times 32) = 2.77$
- Μεγαλώνοντας το μέγεθος του block σε 4 λέξεις, δίνει CPI:
 - 32-bit bus και memory, καθόλου interleaving = $2 + (1.2 \times 1\% \times 4 \times 32) = 3.54$
 - 32-bit bus και memory, interleaved = $2 + (1.2 \times 1\% \times (4 + 24 + 16)) = 2.53$
 - 64-bit bus και memory, καθόλου interleaving = $2 + (1.2 \times 2\% \times 2 \times 32) = 2.77$

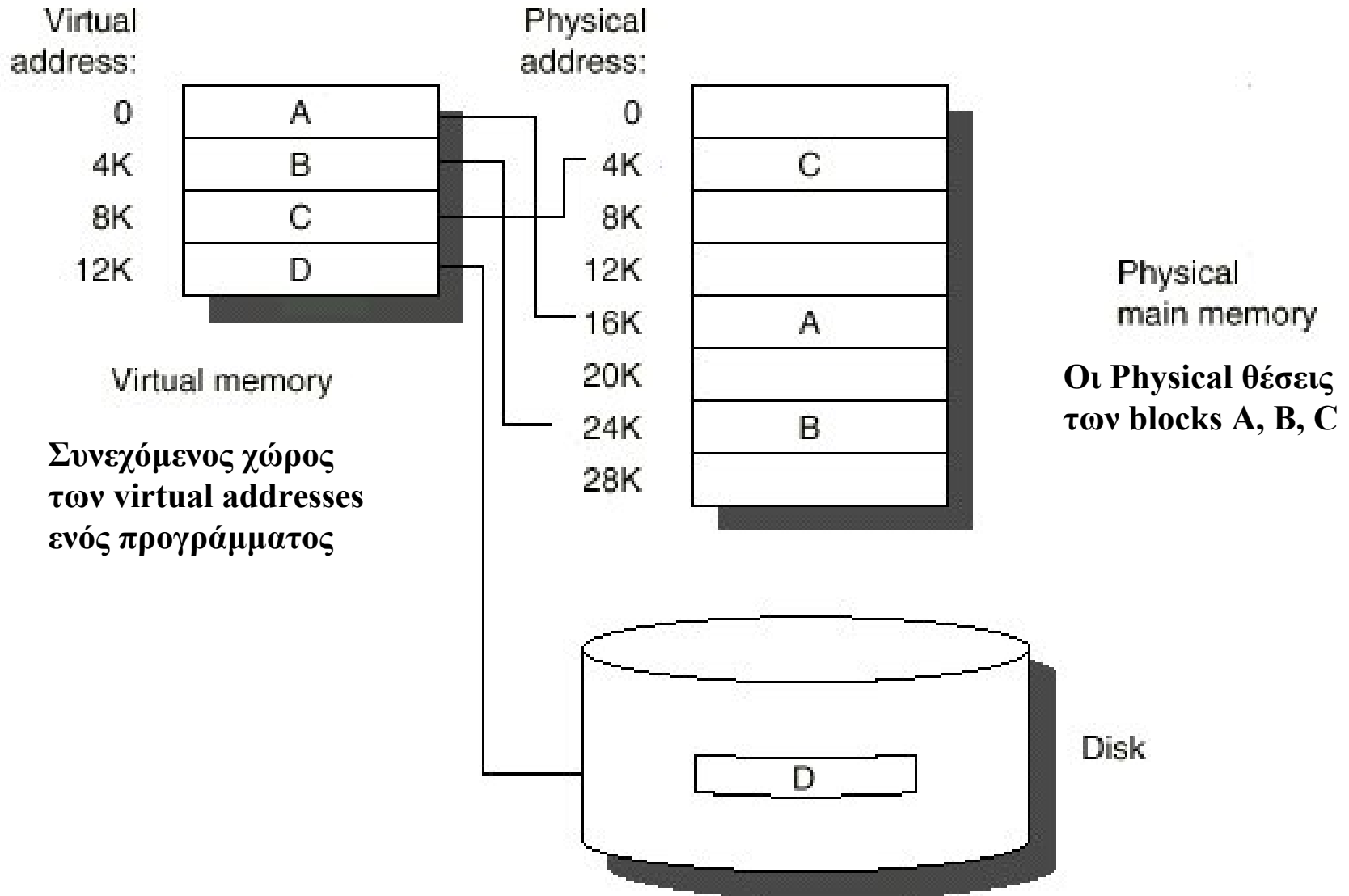
Συστατικά ενός Computer System



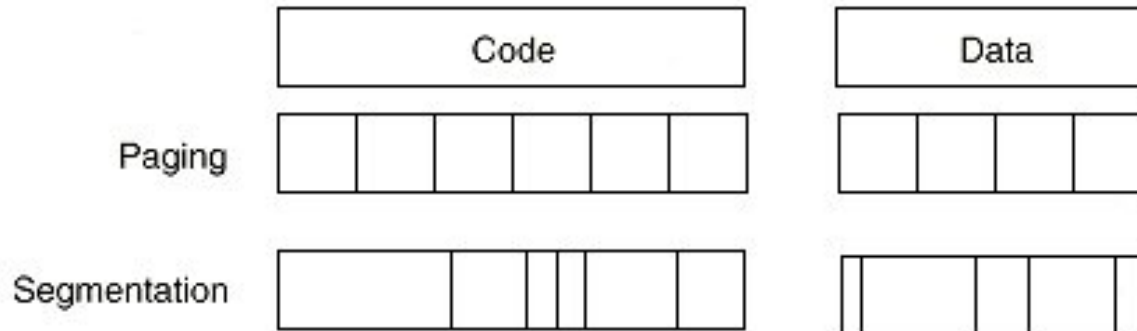
Virtual Memory

- Η Virtual memory ελέγχει 2 επίπεδα της ιεραρχίας μνήμης:
 - Κύρια μνήμη (DRAM)
 - Μαζική αποθήκευση (συνήθως μαγνητικοί δίσκοι)
- Η κύρια μνήμη διαιρείται σε blocks κατανεμημένες σε διαφορετικές τρέχουσες διεργασίες του συστήματος:
 - Blocks καθορισμένου μεγέθους: Pages (μέγεθος 4k έως 64k bytes).
 - Blocks μεταβλητού μεγέθους : Segments (μέγεθος το πολύ 216 μέχρι 232)
- Σε δεδομένο χρόνο, για κάθε τρέχουσα διεργασία, ένα κομμάτι των δεδομένων ή του κώδικα φορτώνεται στην κύρια μνήμη ενώ το υπόλοιπο είναι διαθέσιμο μόνο στους μαγνητικούς δίσκους.
- Ένα block κώδικα ή δεδομένων που χρειάζεται για την εκτέλεση μιας διεργασίας αλλά δεν υπάρχει στη κύρια μνήμη έχει ως αποτέλεσμα ένα page fault (address fault) και το block πρέπει να φορτωθεί στην κύρια μνήμη από το δίσκο ή τον χειριστή του λειτουργικού συστήματος (OS handler).
- Ένα πρόγραμμα μπορεί να εκτελεστεί σε οποιαδήποτε θέση της κύριας μνήμης ή του δίσκου χρησιμοποιώντας έναν μηχανισμό επανατοποθέτησης ο οποίος να ελέγχεται από το λειτουργικό σύστημα που να αντιστοιχεί τις διευθύνσεις από τον χώρο των virtual addresses (logical program address) στο χώρο των physical addresses (κύρια μνήμη, δίσκος).

Μετάφραση Virtual -> Physical Addresses



Paging & Segmentation

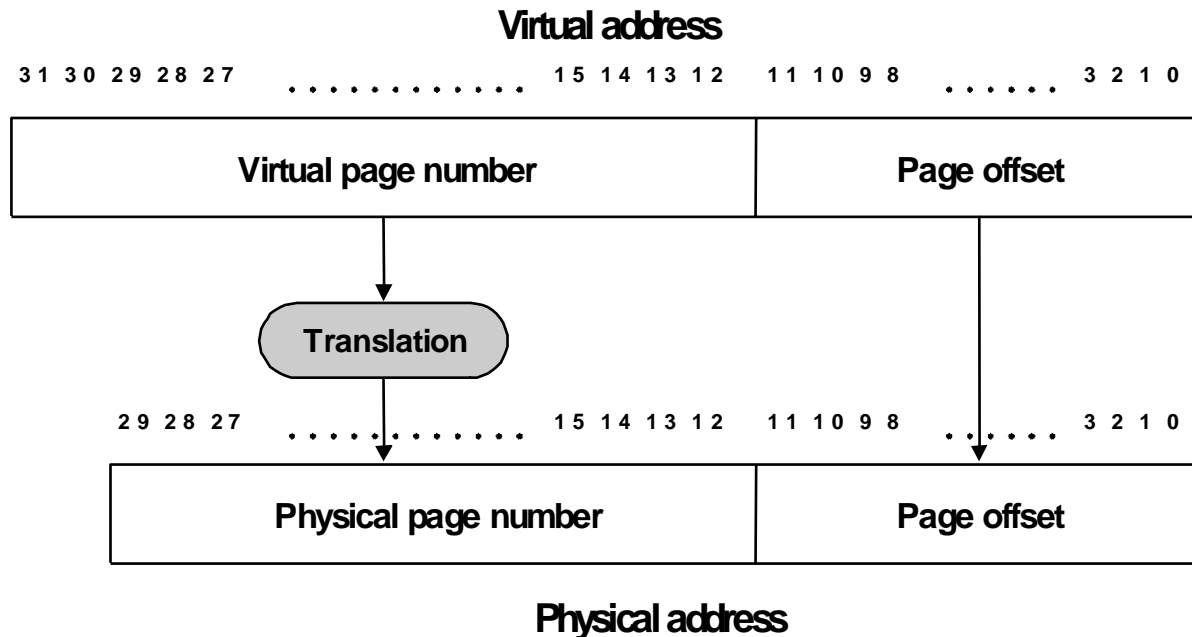


	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial (all blocks are the same size)	Hard (must find contiguous, variable-size, unused portion of main memory)
Memory use inefficiency	Internal fragmentation (unused portion of page)	External fragmentation (unused pieces of main memory)
Efficient disk traffic	Yes (adjust page size to balance access time and transfer time)	Not always (small segments may transfer just a few bytes)

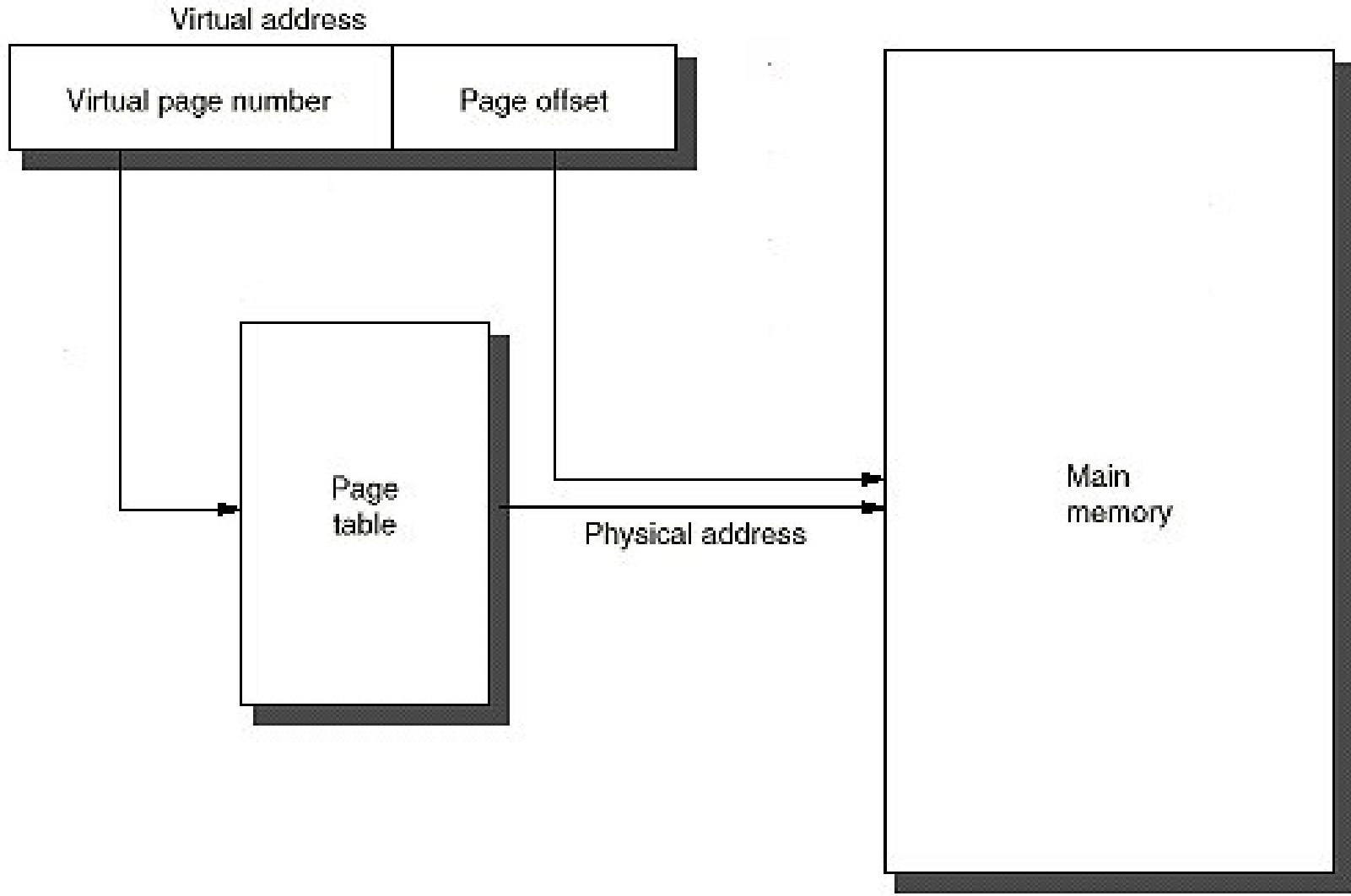
Virtual Memory

Πλεονεκτήματα

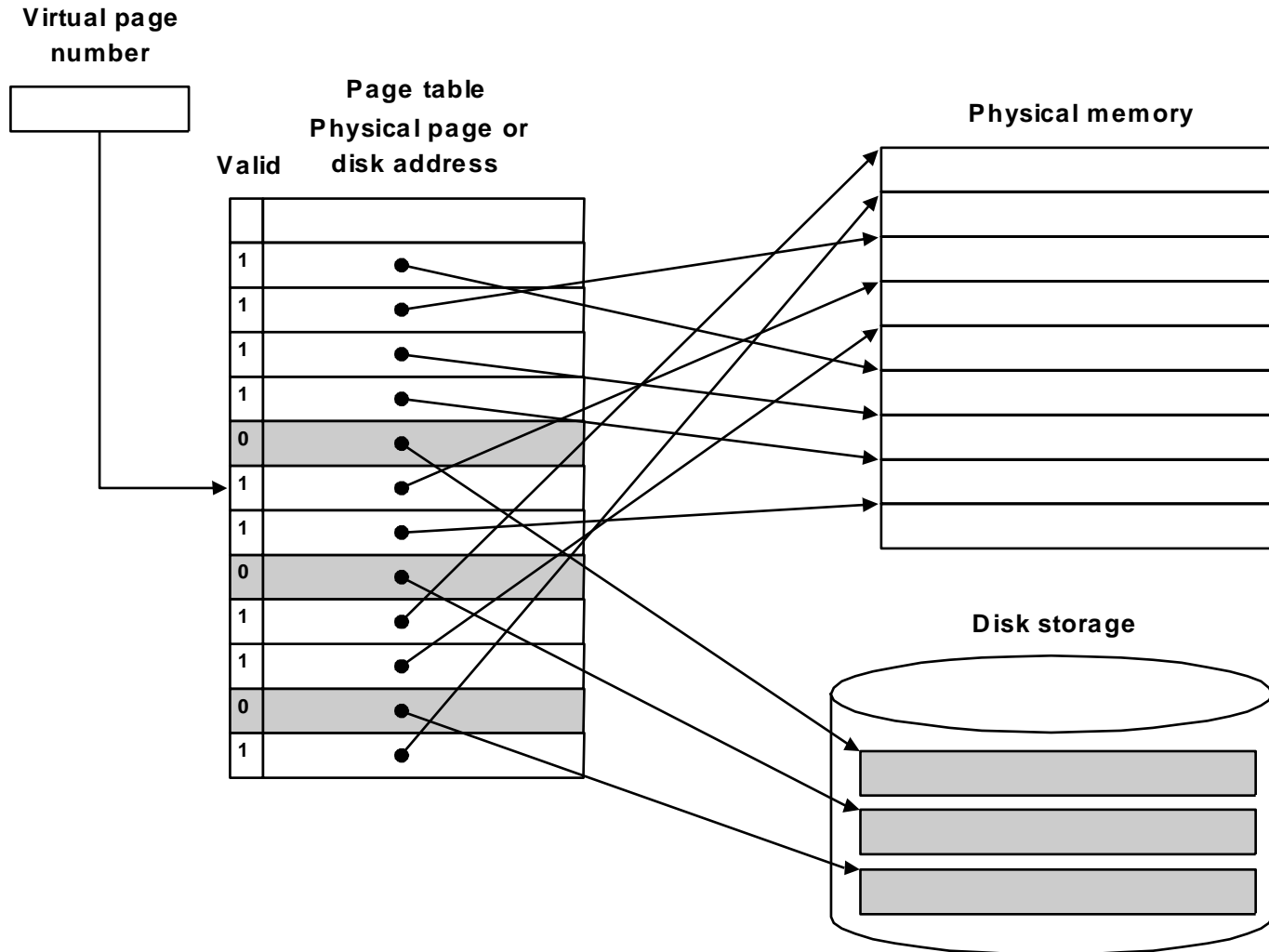
- Έχουμε την ψευδαίσθηση ότι διαθέτουμε περισσότερη φυσική κύρια μνήμη
- Επιτρέπει τον επανατοποθέτηση των προγραμμάτων
- Προστατεύει από παράτυπη πρόσβαση στη μνήμη



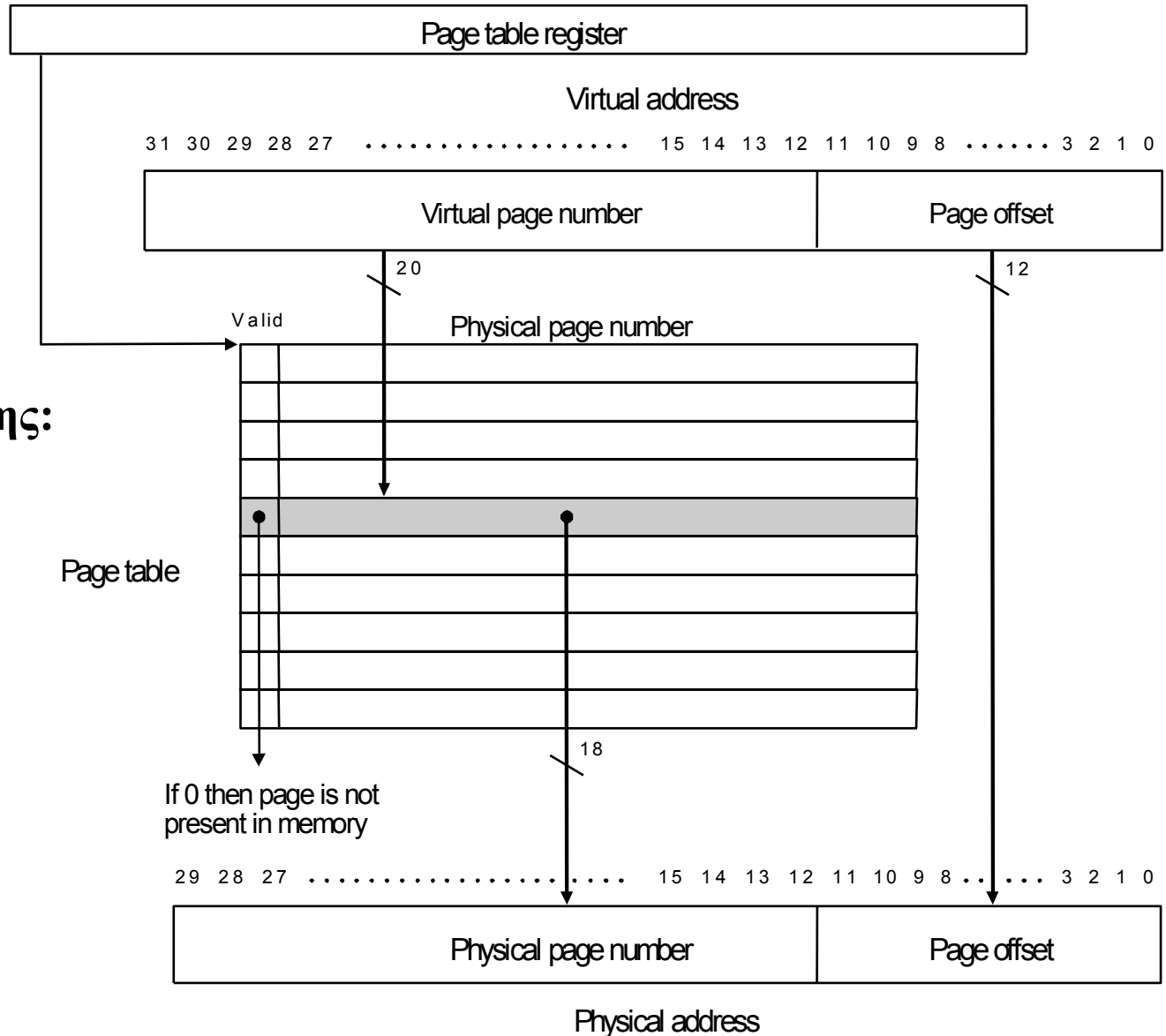
Αντιστοίχιση των Virtual Addresses σε Physical Addresses μέσω ενός πίνακα σελίδων (Page Table)



Μετάφραση των Virtual Addresses



Page Table



Χρειάζονται 2 προσπελάσεις μνήμης:

- στο page table
- στο αντικείμενο

Τυπικές παράμετροι της Cache και της Virtual Memory

Parameter	First-level cache	Virtual memory
Block (page) size	16–128 bytes	4096–65,536 bytes
Hit time	1–2 clock cycles	40–100 clock cycles
Miss penalty	8–100 clock cycles	700,000–6,000,000 clock cycles
(Access time)	(6–60 clock cycles)	(500,000–4,000,000 clock cycles)
(Transfer time)	(2–40 clock cycles)	(200,000–2,000,000 clock cycles)
Miss rate	0.5–10%	0.00001– 0.001%
Data memory size	0.016–1MB	16–8192 MB

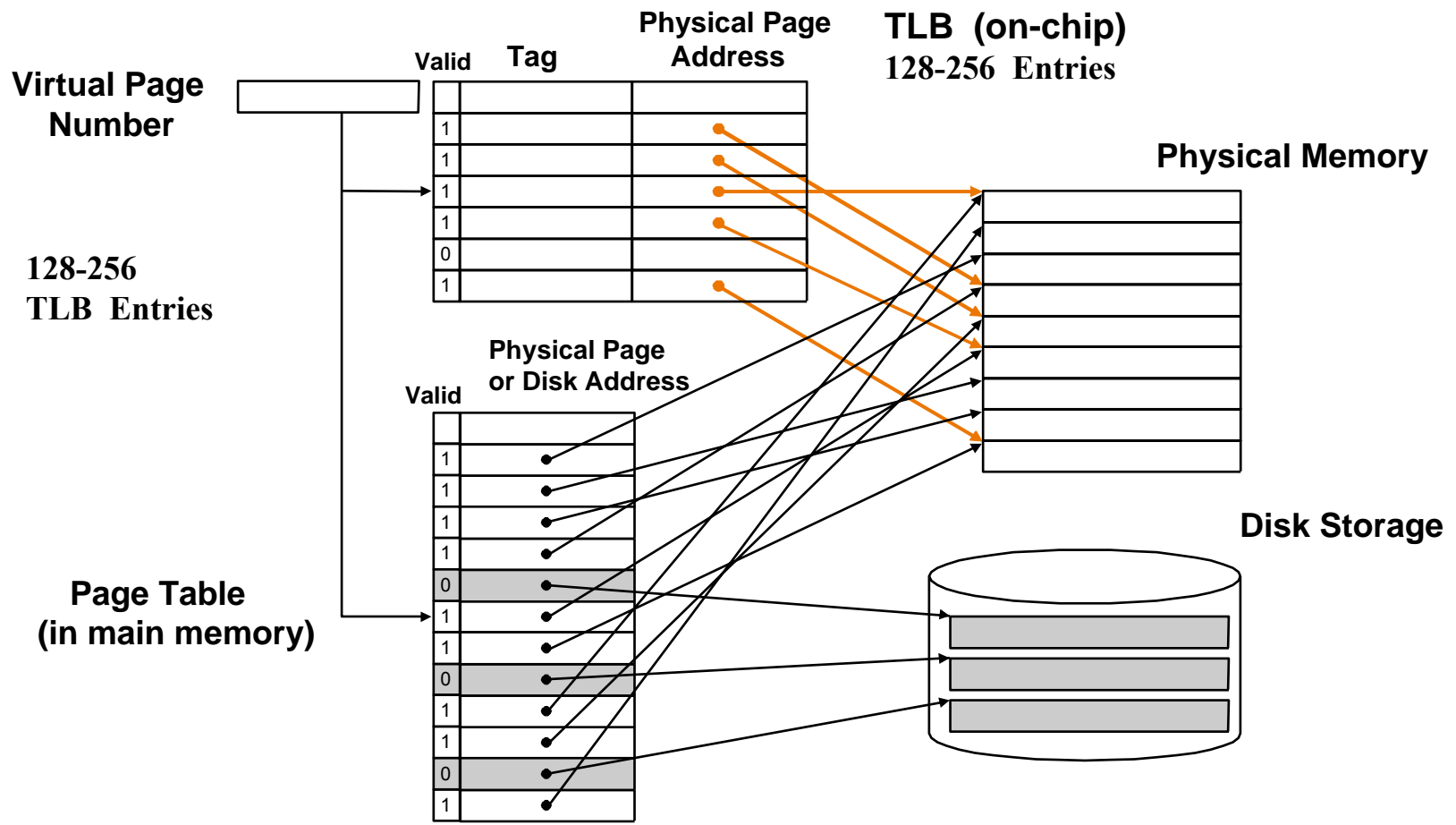
Virtual Memory

Στρατηγικές

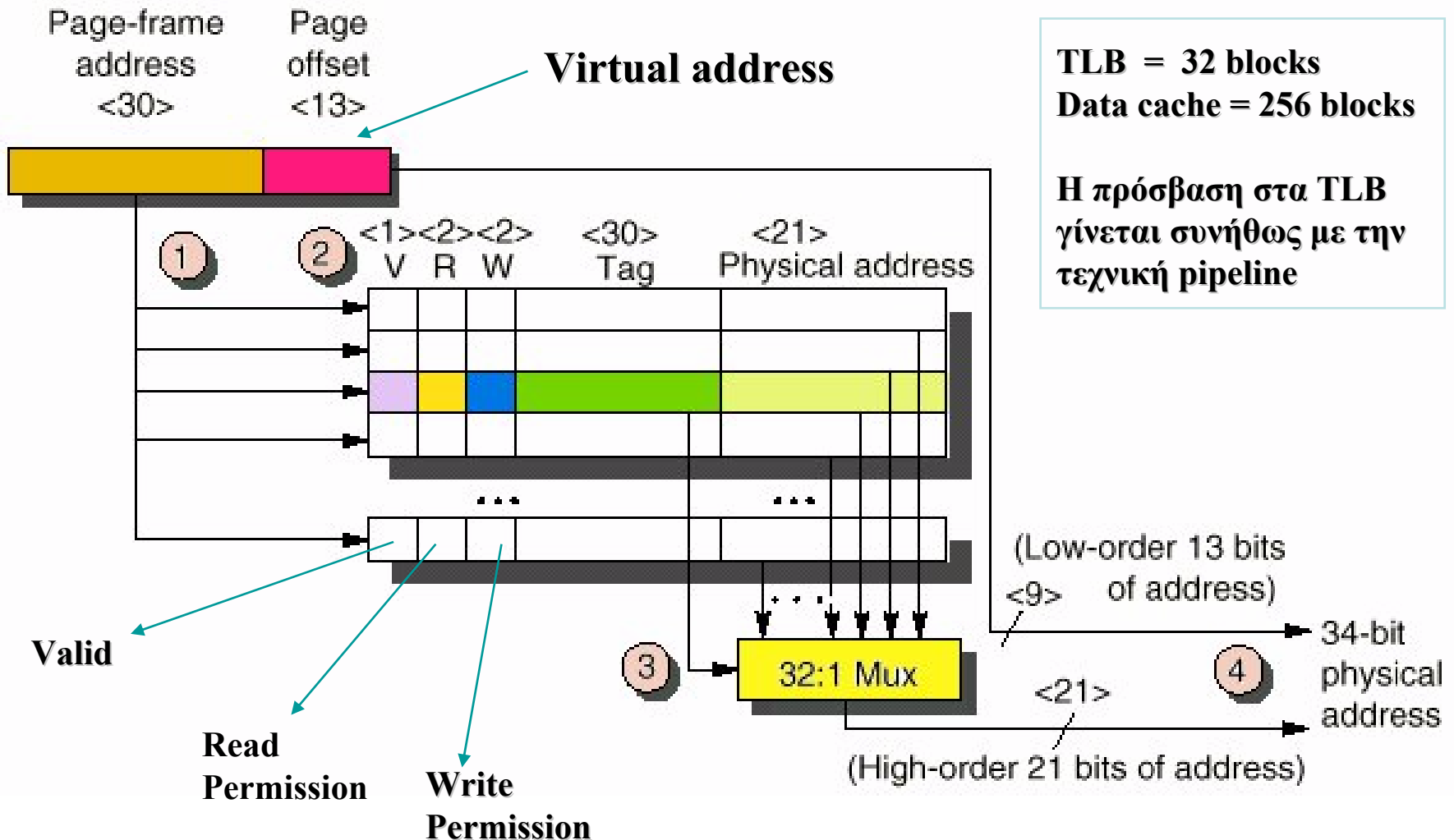
- **Τοποθέτηση του block στην κύρια μνήμη:** Η fully associative τεχνική χρησιμοποιείται για την ελάττωση του miss rate.
- **Αντικατάσταση του block:** The least recently used (LRU) block αντικαθίσταται όταν ένα νέο block έρχεται στη μνήμη από το δίσκο.
- **Στρατηγική εγγραφών:** Χρησιμοποιείται η τεχνική write back και μόνο οι dirty σελίδες μεταφέρονται από την κύρια μνήμη στο δίσκο.
- Για την τοποθέτηση των blocks στην κύρια μνήμη χρησιμοποιείται ένας **page table**. Ο page table δεικτοδοτείται από τον εικονικό αριθμό σελίδας (virtual page number) και περιέχει τη φυσική διεύθυνση (physical address) του block.
 - **Paging:** Το Offset συγχωνεύεται με τη διεύθυνση της φυσικής σελίδας.
 - **Segmentation:** Το Offset προστίθεται στη διεύθυνση του physical segment.
- Για την αξιοποίηση της address locality, χρησιμοποιείται συνήθως ο **translation look-aside buffer (TLB)** για την αποθήκευση των προσφάτως μεταφρασμένων διευθύνσεων ώστε να αποφεύγεται προσπέλαση της μνήμης προκειμένου να διαβαστεί ο πίνακας σελίδων (page table).

Επιτάχυνση της μετάφρασης διευθύνσεων: Translation Lookaside Buffer (TLB)

- TLB: Μία μικρή on-chip fully-associative cache που χρησιμοποιείται για τη μετάφραση διευθύνσεων.
- Αν μία virtual address υπάρχει μέσα στο TLB (TLB hit), δεν προσπελαύνεται ο πίνακας σελίδων της κύριας μνήμης.

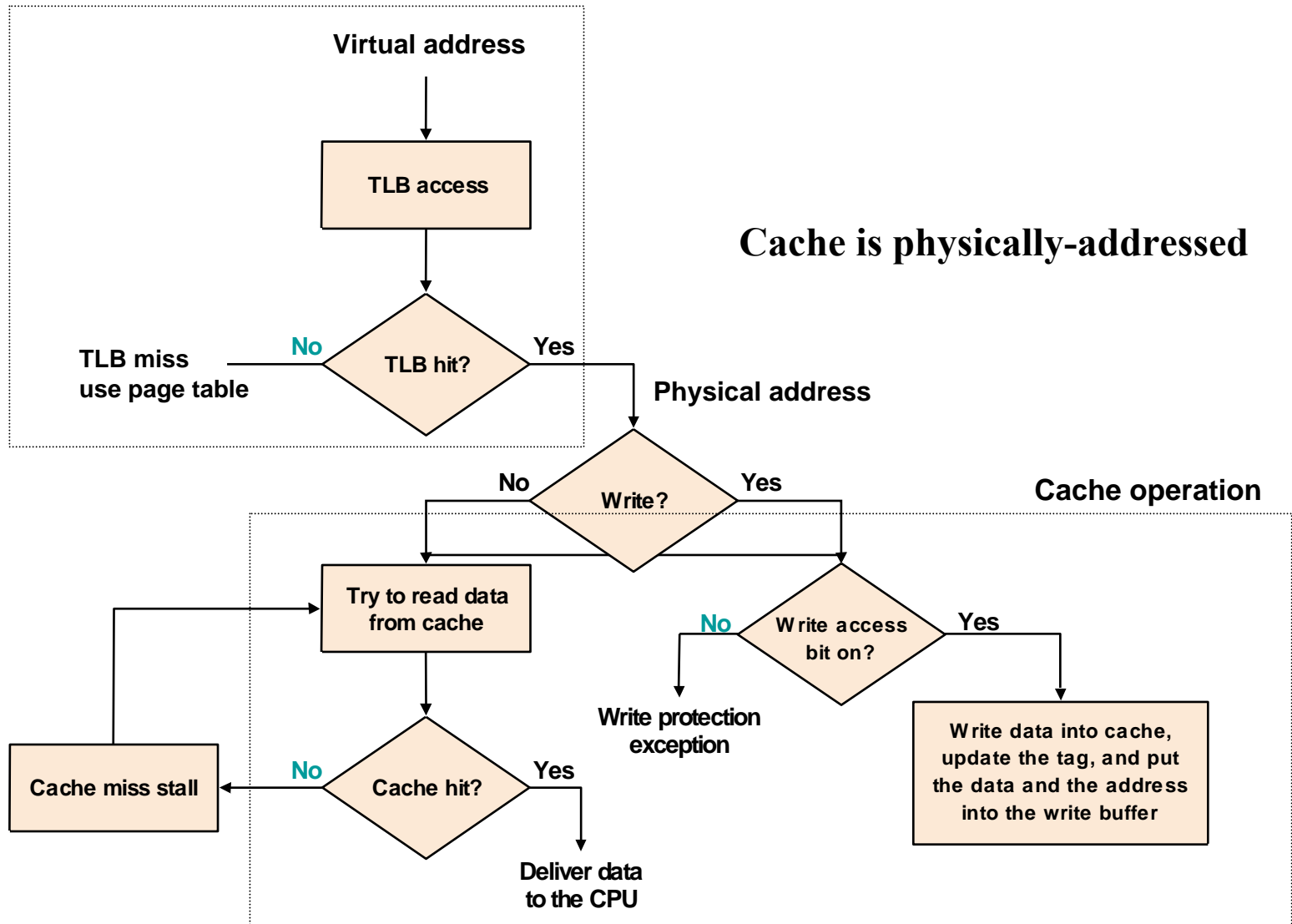


Η λειτουργία του Alpha AXP 21064 Data TLB κατά τη μετάφραση των διευθύνσεων



TLB & Cache Operation

TLB Operation



Συνδυασμός των Cache, TLB, Virtual Memory

Cache	TLB	Virtual Memory	Είναι δυνατό? Πότε?
Miss	Hit	Hit	Ναι, δεν ελέγχεται το page table
Hit	Miss	Hit	TLB miss, βρίσκεται στο page table
Miss	Miss	Hit	TLB miss, cache miss
Miss	Miss	Miss	Page fault
Miss	Hit	Miss	Αδύνατο, αν όχι στη μνήμη ούτε στο TLB
Hit	Hit	Miss	Αδύνατο, αν όχι στη μνήμη ούτε στο TLB ή στην cache
Hit	Miss	Miss	Αδύνατο, αν όχι στη μνήμη ούτε στην cache

Σύνοψη

