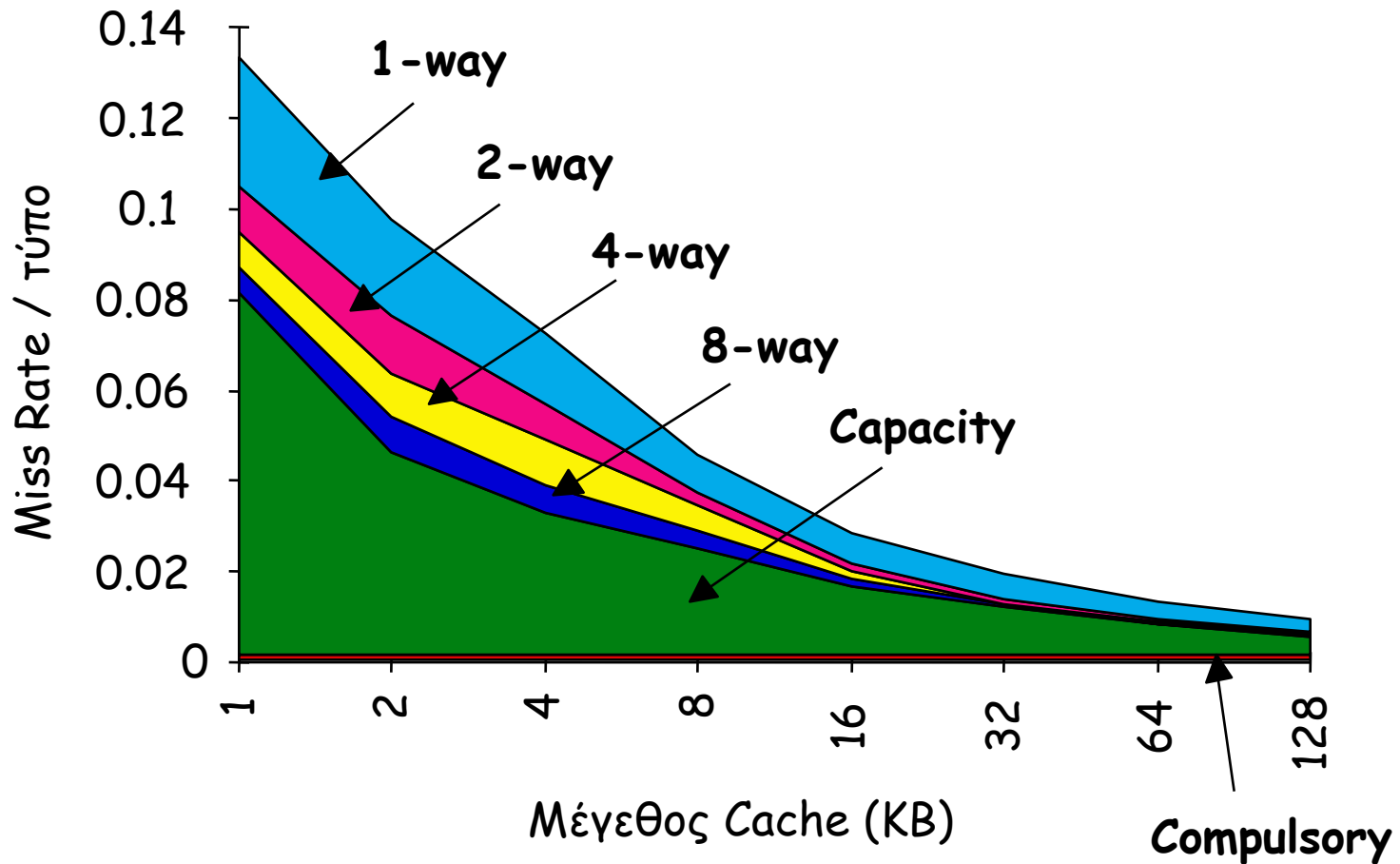


Είδη των Cache Misses: 3C's

- 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).

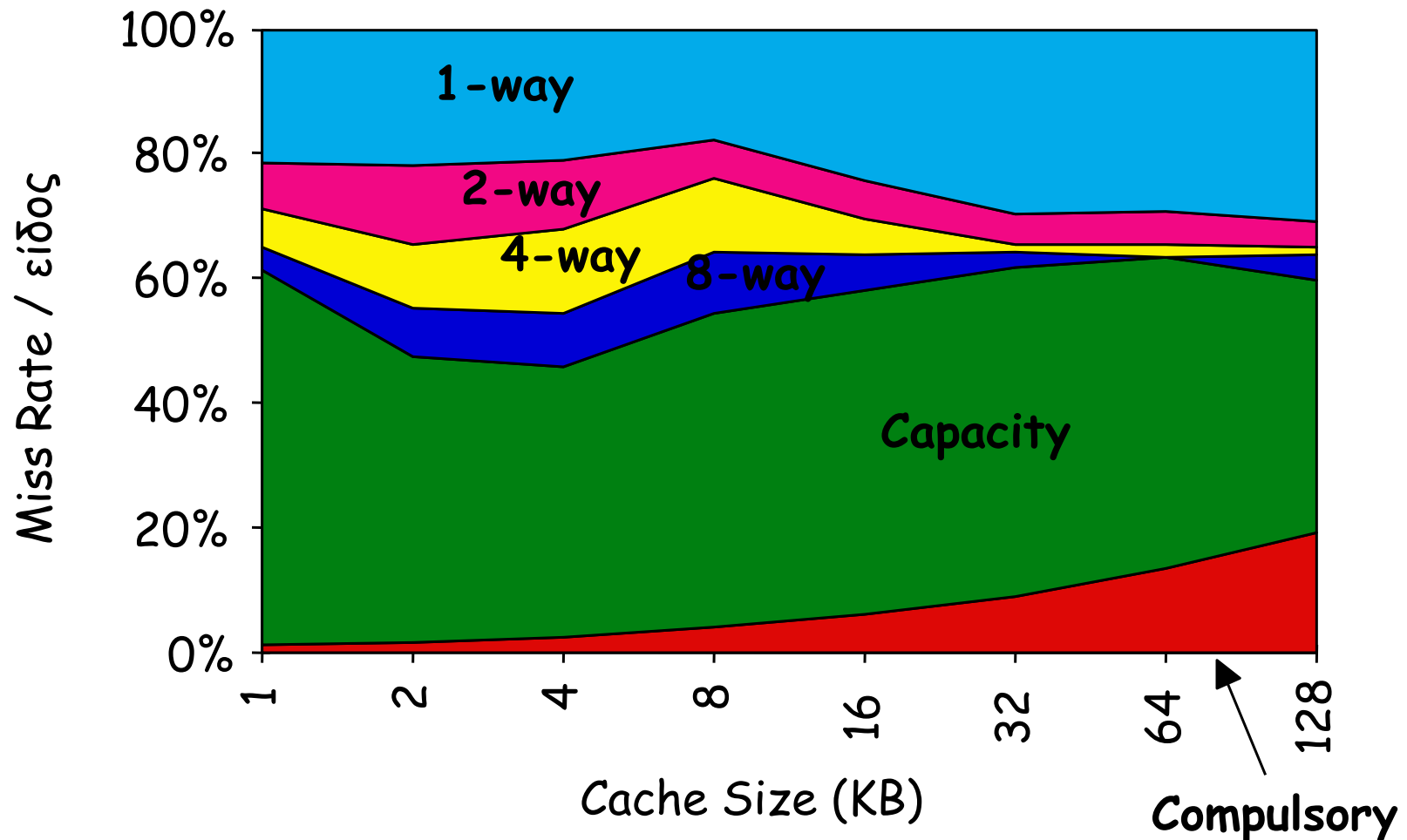
Τα 3 Cs των Cache:

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

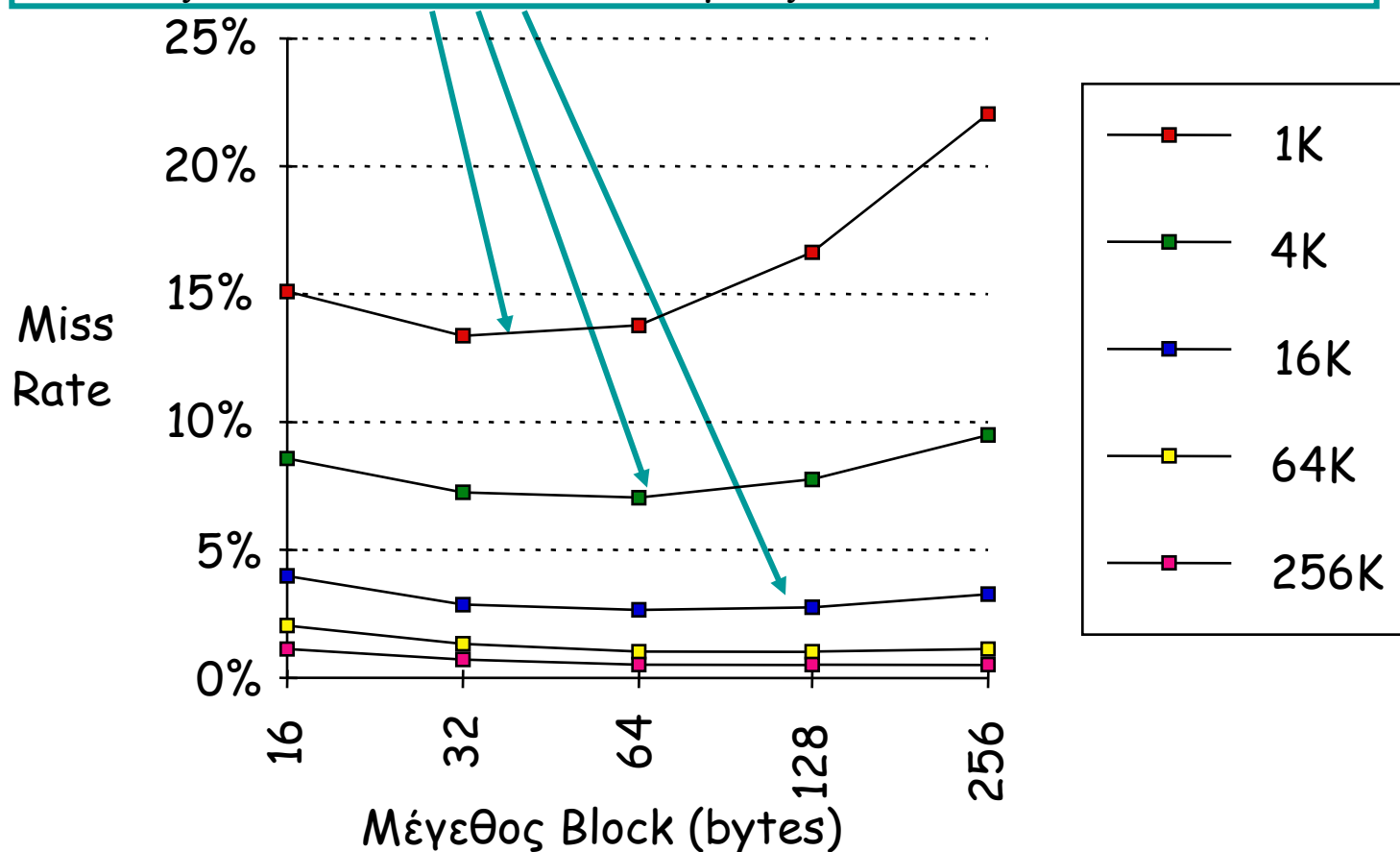
• Τεχνικές μείωσης του Cache Hit Time:

- * 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

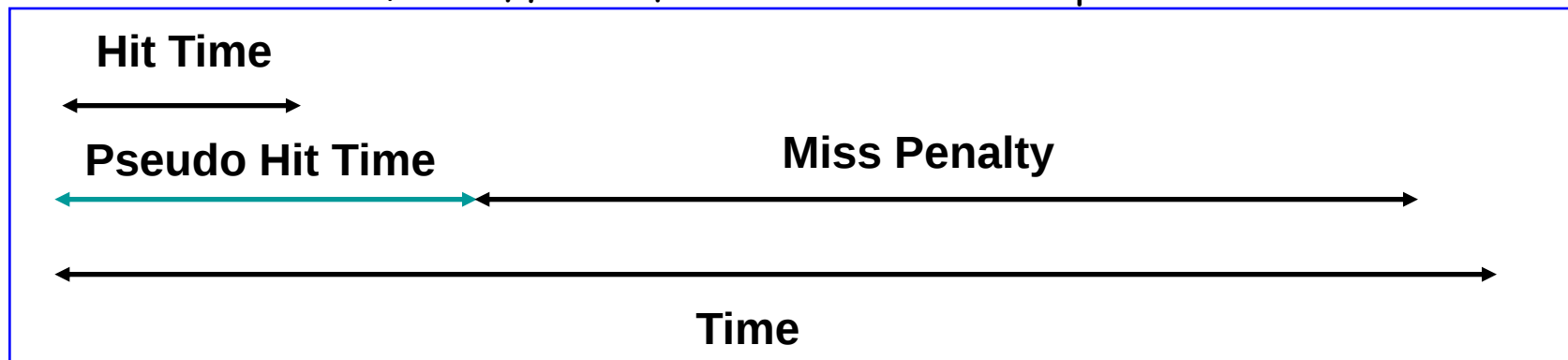
Παράδειγμα: Μέσος χρόνος πρόσβασης στη μνήμη vs. Miss Rate

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

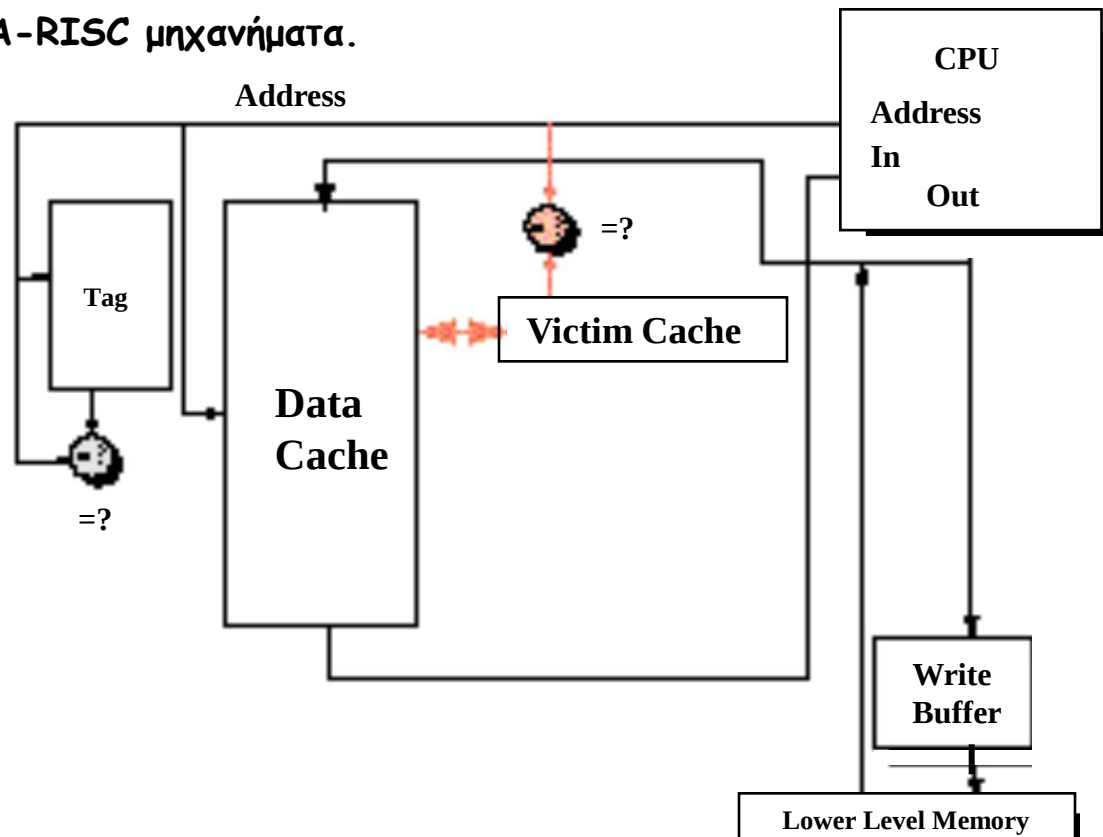
- Συνδυάζει το μικρό χρόνο αναζήτησης (hit time) των Direct Mapped caches και το μικρότερο αριθμό των conflict misses στις 2-way set-associative caches.
- Η cache διαιρείται σε δύο τμήματα: Όταν έχουμε cache miss, ελέγχουμε το άλλο μισό της cache για να δούμε αν τα δεδομένα που αναζητάμε βρίσκονται εκεί. 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 προσπελάζουμε 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 προσπελάζοντας 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 stuctures */
```

```
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;
    }
  };
```

		j					
x		0	1	2	3	4	5
	0						
	1						
	2						
	3						
	4						

		k					
y		0	1	2	3	4	5
	0						
	1						
	2						
	3						
	4						

		j					
z		0	1	2	3	4	5
	0						
	1						
	2						
	3						
	4						

- **Οι 2 εσωτερικότεροι βρόχοι:**
 - Προσπελαύνουν όλα τα $N \times N$ στοιχεία του $z[]$
 - Προσπελαύνουν 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

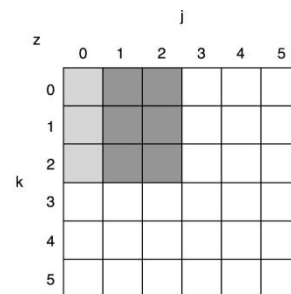
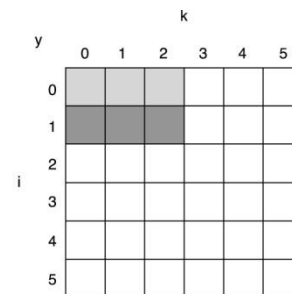
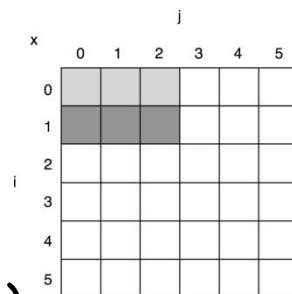
Τεχνικές μείωσης του Miss Rate: Compiler Optimizations

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	Mem[124]
	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

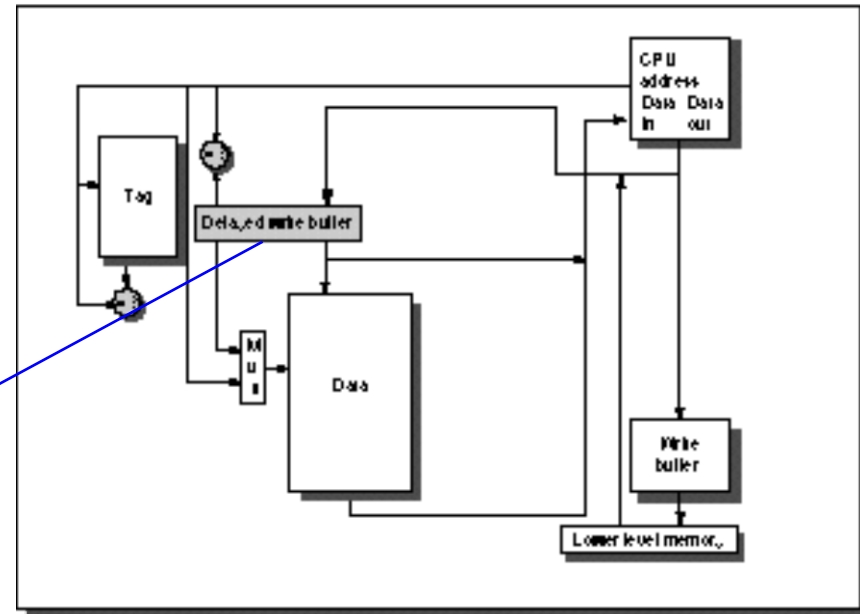
Οι Non-blocking caches ή lockup-free caches επιτρέπουν στις data caches να αποστέλλουν δεδομένα που περιέχουν (cache hits) όσο διεκπεραιώνεται ένα miss:

- 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



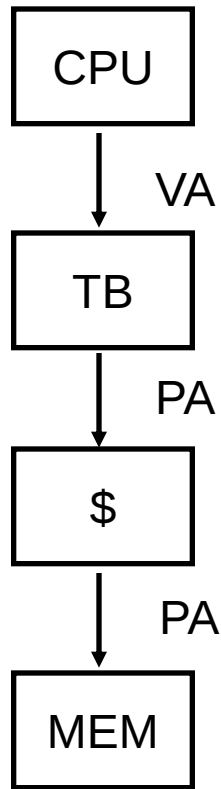
- "Delayed Write Buffer": which must be checked on reads; either complete write or read from buffer

Τεχνικές μείωσης του Hit Time :

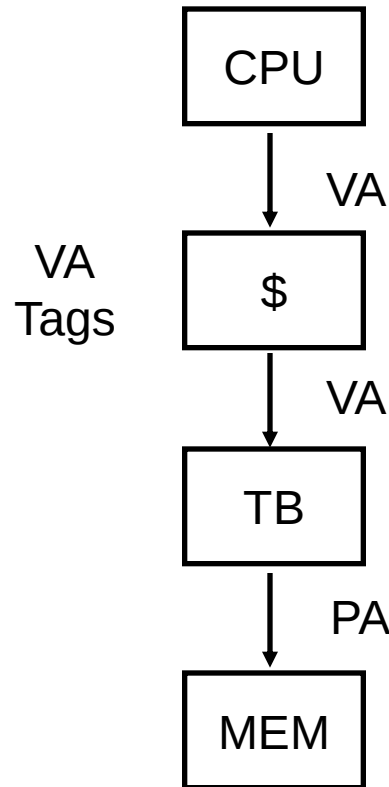
Avoiding Address Translation

- Αποστολή της virtual address στην cache: Ονομάζεται Virtually 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

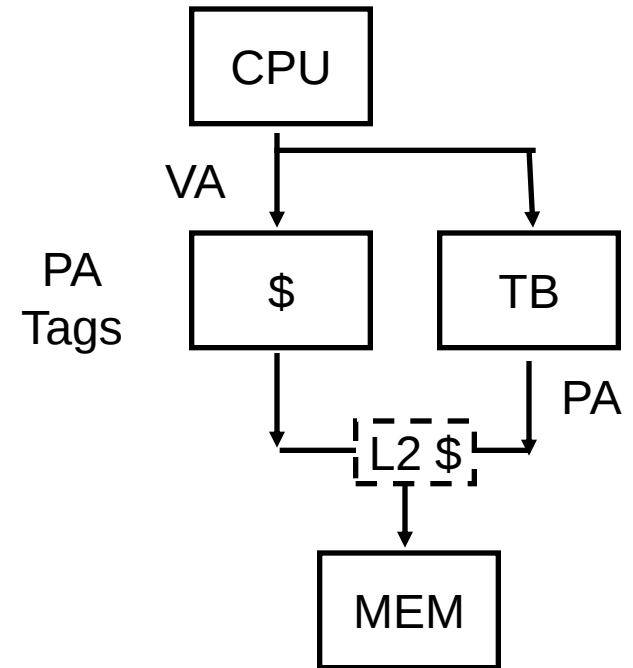
Τεχνικές μείωσης του Hit Time : Virtually Addressed Caches



Συμβατική
Οργάνωση



Virtually Addressed Cache
Μετάφραση μόνο σε miss
Συνοχη προβλήματα

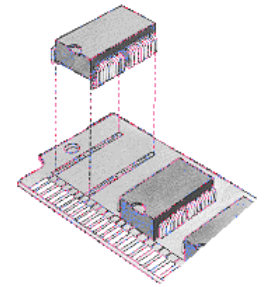


Επικάλυψη της \$
προσπέλασης με VA
μετάφραση: Απαιτείται
δείκτης στην \$ index για
να παραμένει σταθερό
κατά τη μετάφραση

Σύνοψη

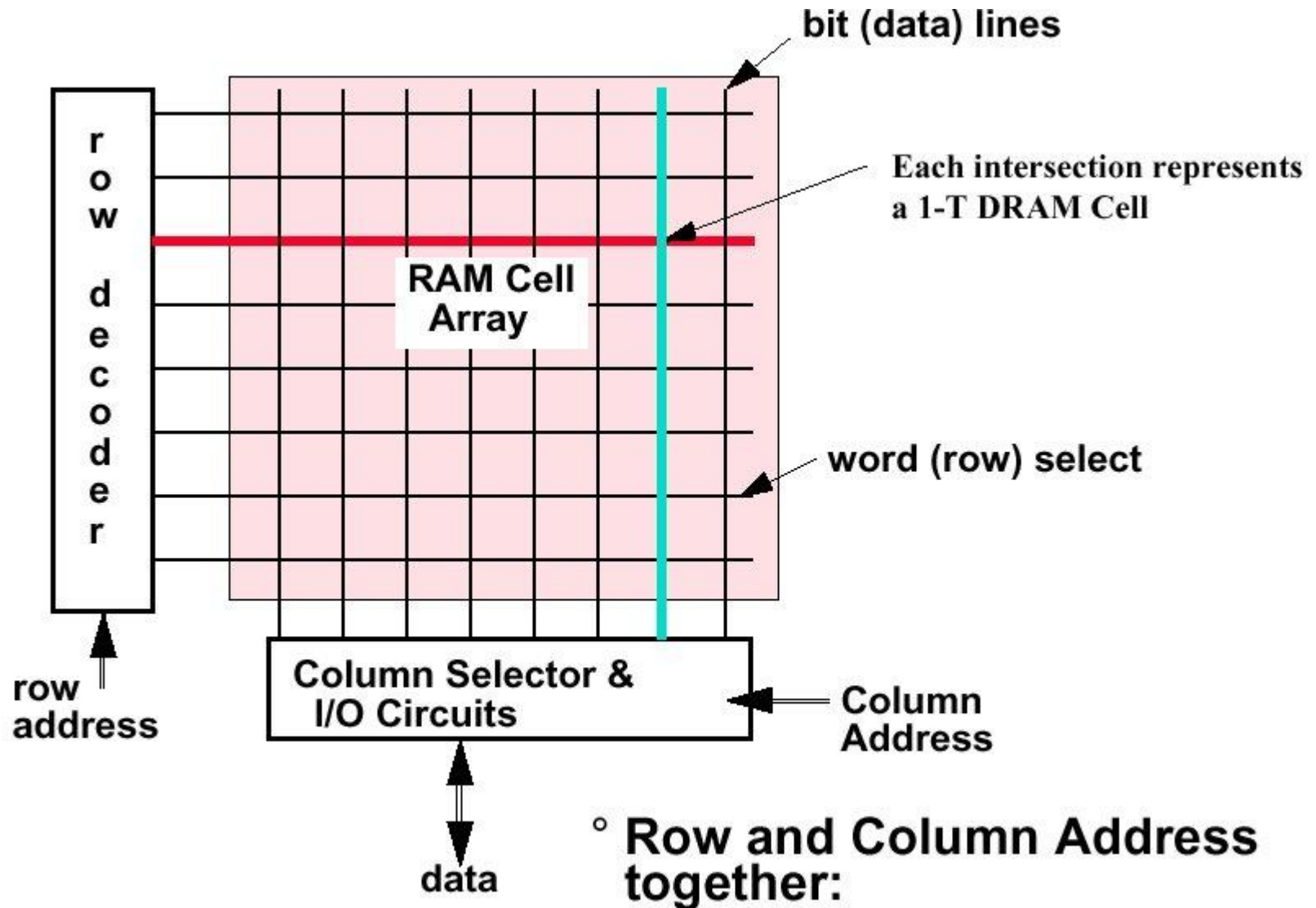
	Τεχνική	MR	MP	HT	Complexity
Miss rate	Μεγαλύτερο μέγεθος Block	+	-		0
	Υψηλότερη Associativity	+		-	1
	Victim Caches	+			2
	Pseudo-Associative Caches	+			2
	HW Prefetching of Instr/Data	+			2
	Compiler Controlled Prefetching	+			3
	Compiler Reduce Misses	+			0
Miss Penalty	Προτεραιότητα στα Read Misses		+		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 επιτρέπει στις γραμμές διευθύνσεων να παραμένουν σταθερές)
 - **Memory bandwidth:** Ο ρυθμός μεταφοράς δεδομένων μεταξύ κύριας μνήμης και cache/CPU.

Οργάνωση της DRAM



- 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.

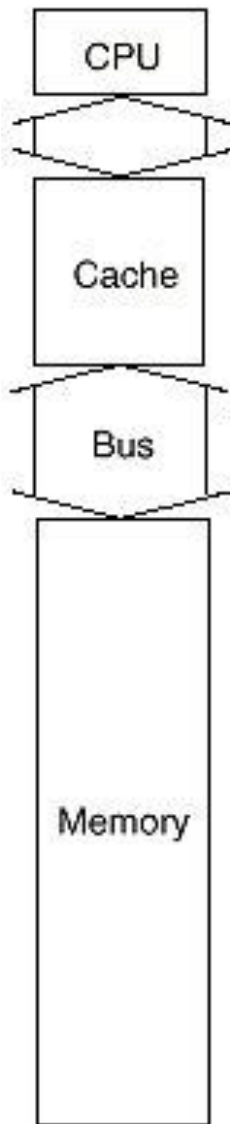
- Ταυτόχρονες αναγνώσεις ή εγγραφές πολλών λέξεων επιτυγχάνονται με αποστολή διευθύνσεων μνημών σε πολλά memory banks σε μία φορά.

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

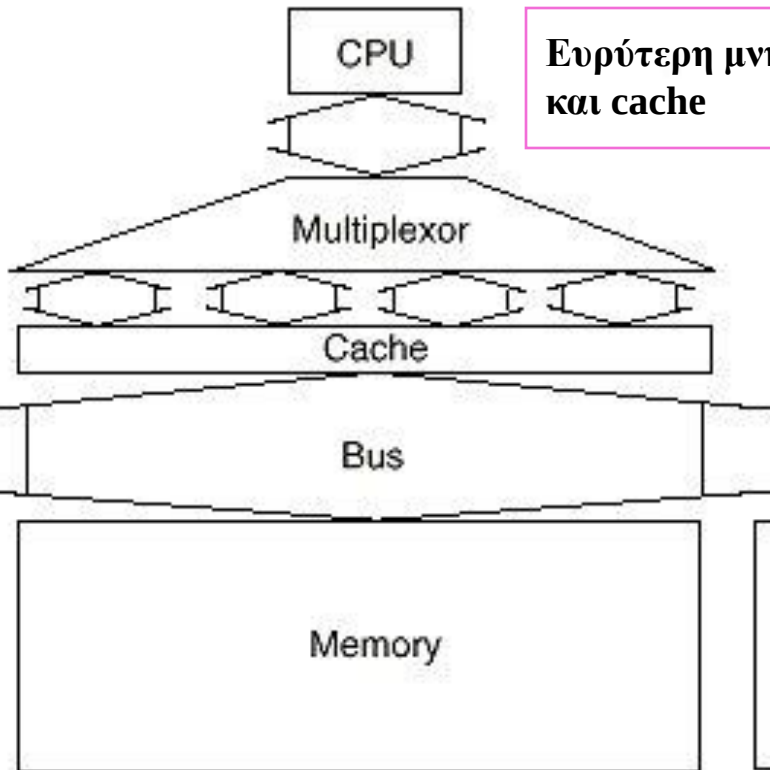
π.χ. χρησιμοποιώντας 4 banks, bank 0 έχει όλες τις λέξεις των οποίων οι διευθύνσεις είναι:

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

(a) One-word-wide memory organization

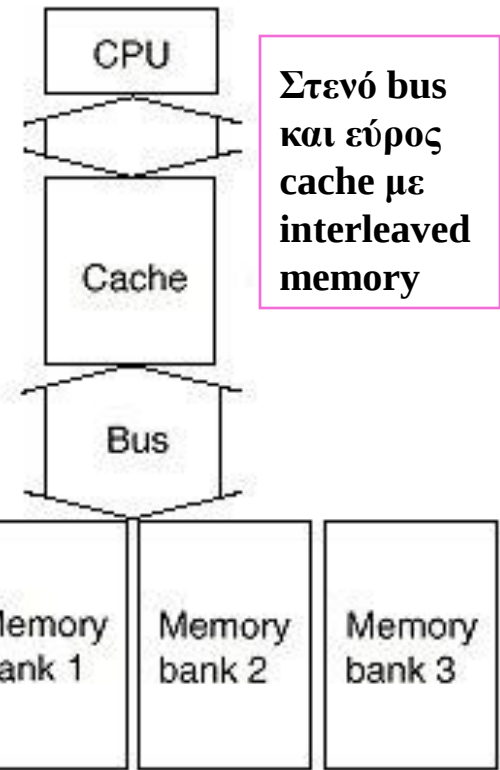


(b) Wide memory organization



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 :

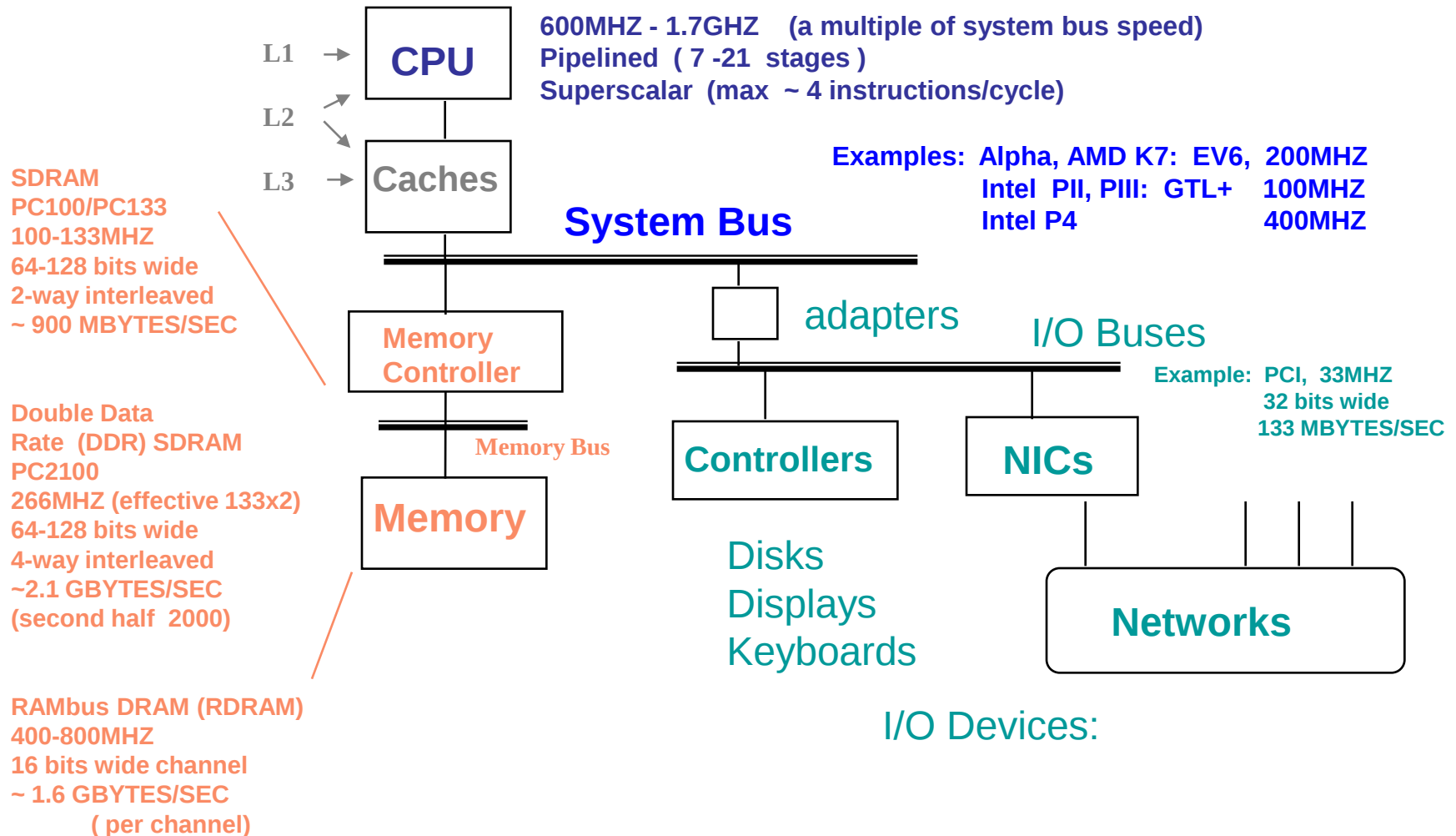
(4 κύκλοι για αποστολή της διεύθυνσης, 24 κύκλοι access time / λέξη, 4 κύκλοι για αποστολή μιας λέξης)

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 = $2 + (1.2 \times 2\% \times 2 \times 32) = 2.77$

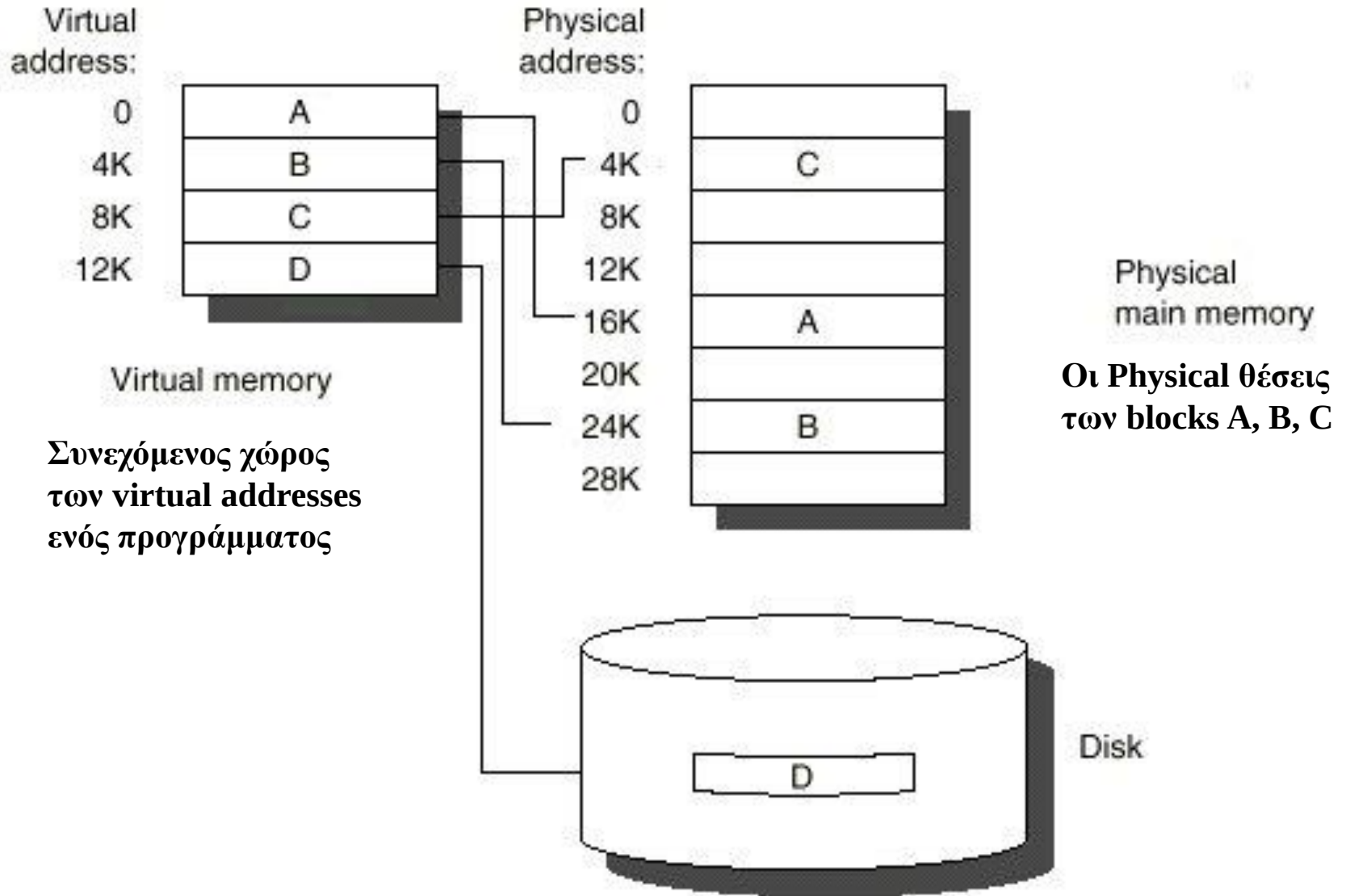
Συστατικά ενός Computer System



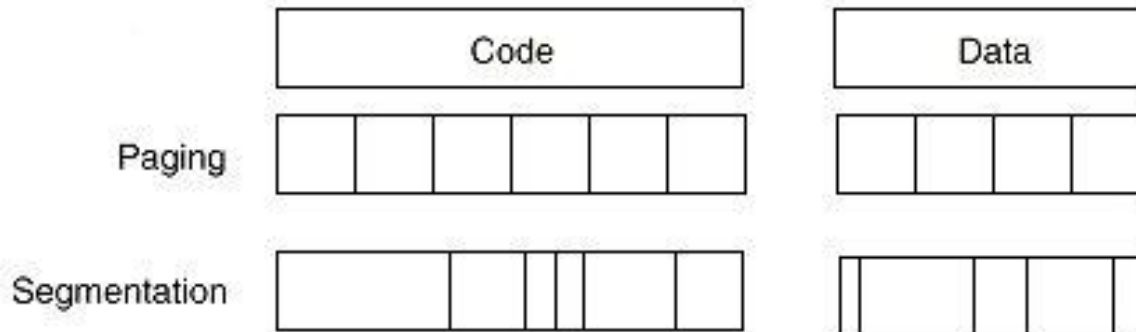
Virtual Memory

- 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

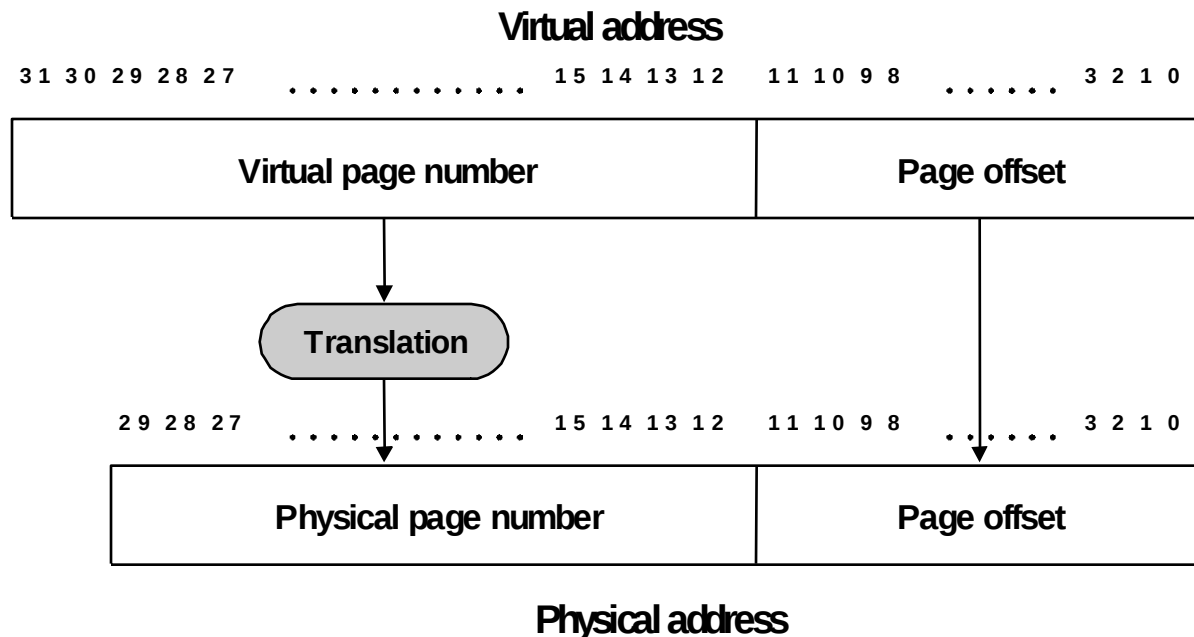


	Page	Segment
Words per address	One	Two (segment and offset)
Programmer visible?	Invisible to application programmer	May be visible to application programmer
Replacing a block	Trivial (all blocks are the same size)	Hard (must find contiguous, variable-size, unused portion of main memory)
Memory use inefficiency	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)

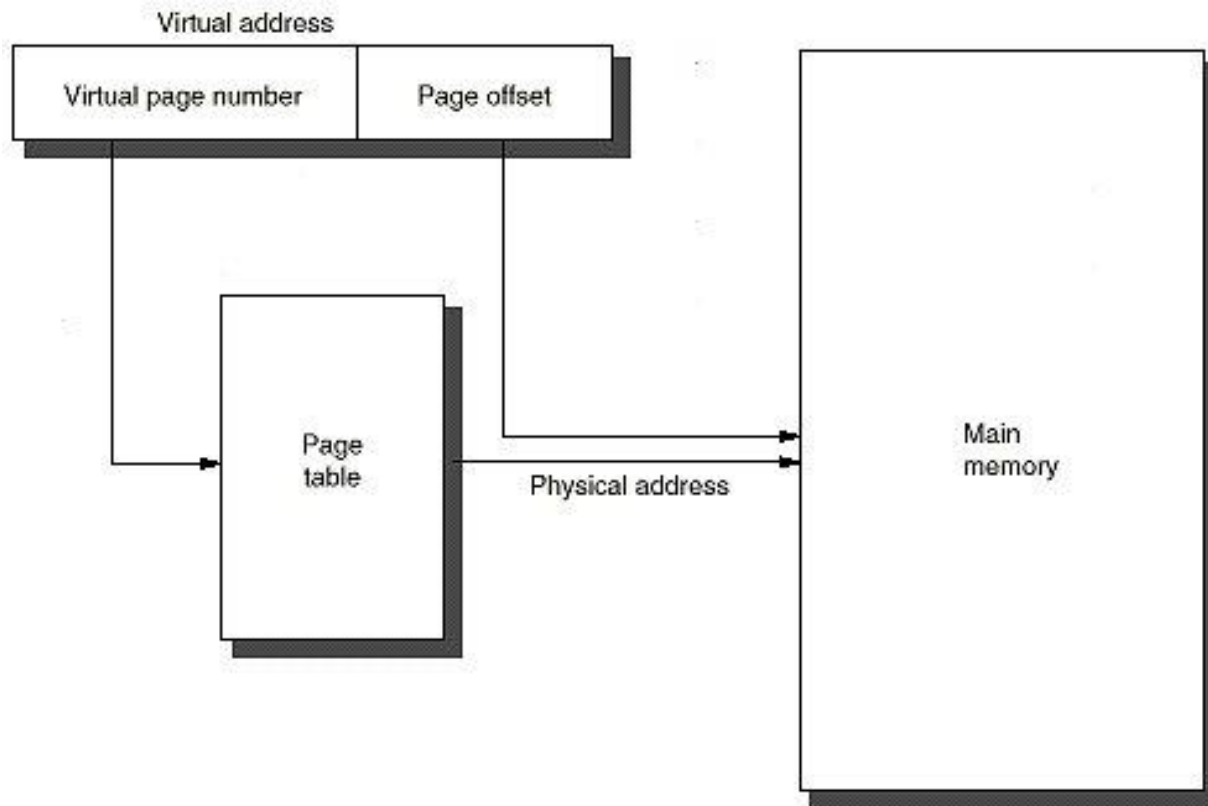
Virtual Memory

Πλεονεκτήματα

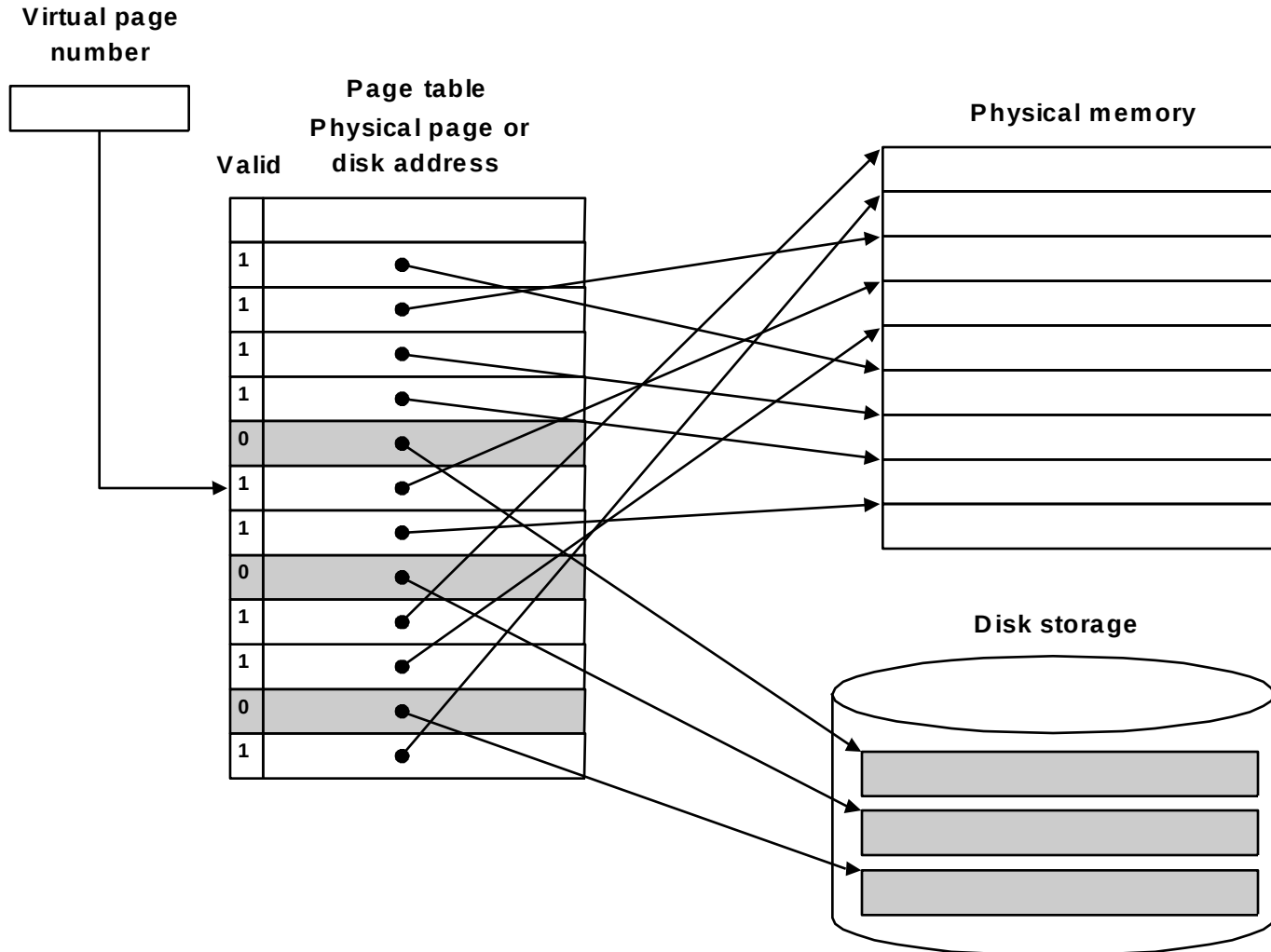
- Έχουμε την ψευδαίσθηση ότι διαθέτουμε περισσότερη φυσική κύρια μνήμη
- Επιτρέπει τον επανατοποθέτηση των προγραμμάτων
- Προστατεύει από παράτυπη πρόσβαση στη μνήμη



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



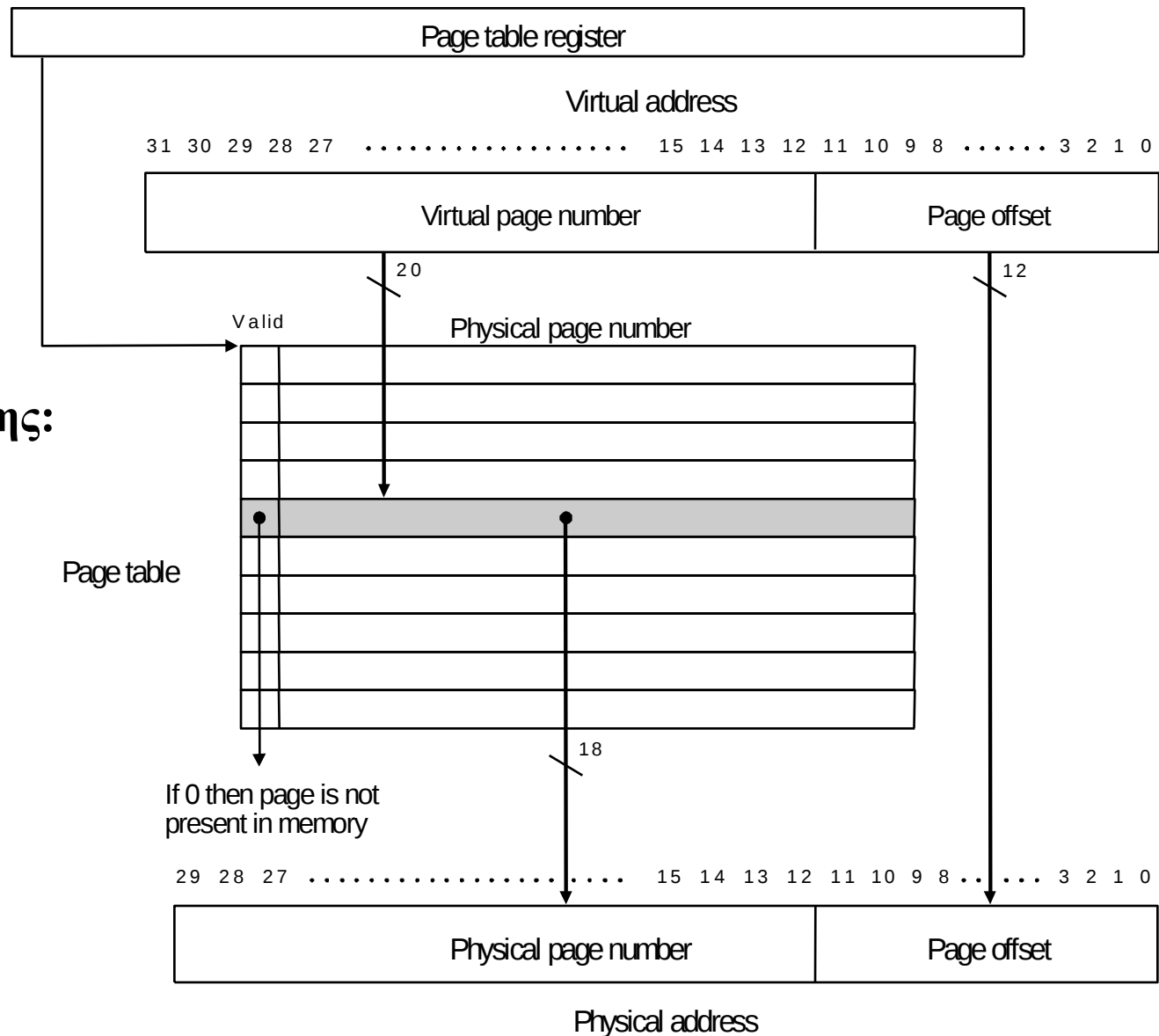
Μετάφραση των Virtual Addresses



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

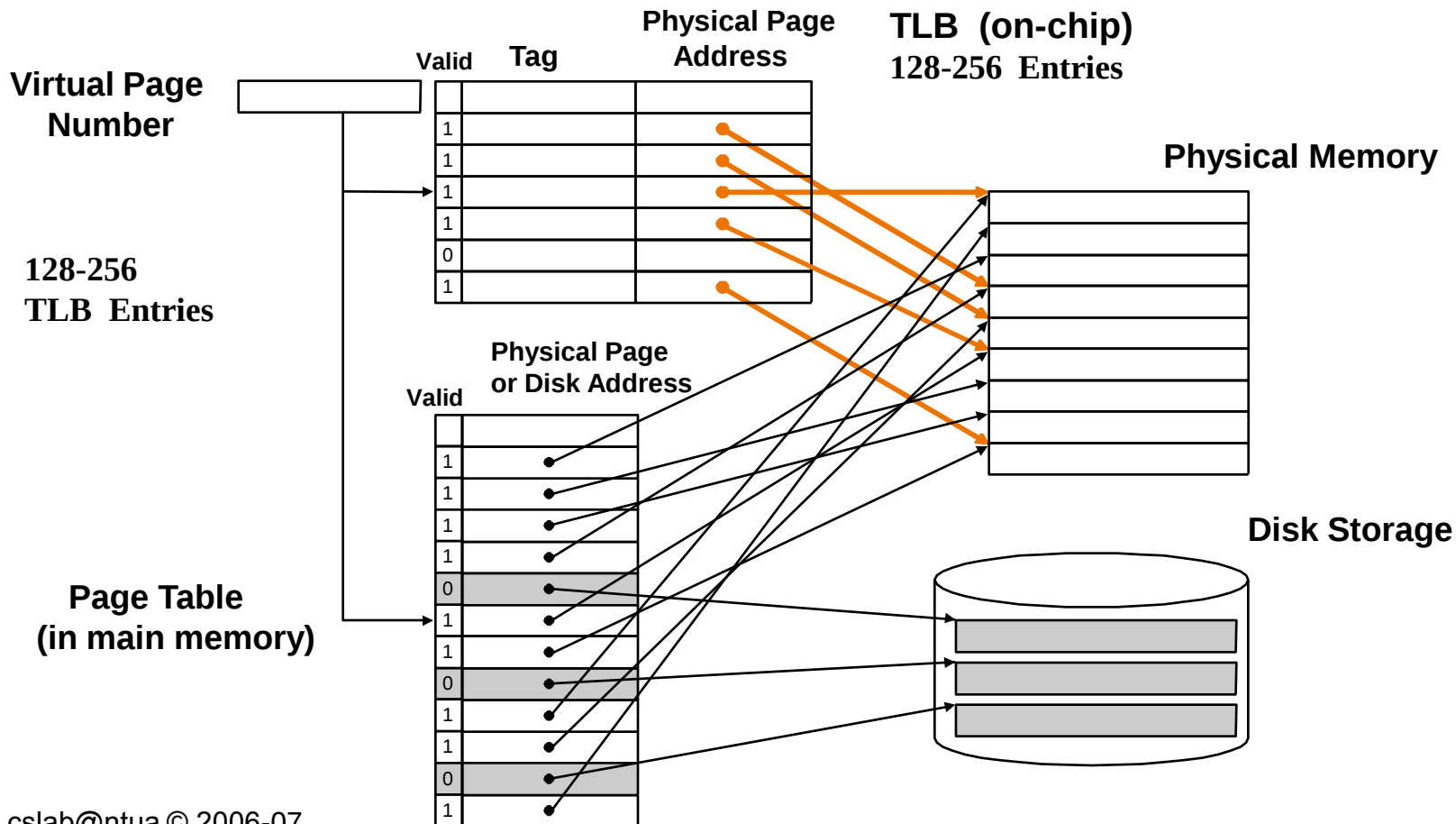
- **Τοποθέτηση του block στην κύρια μνήμη:** Η fully associative τεχνική χρησιμοποιείται για την ελάττωση του miss rate.
- **Αντικατάσταση του block:** The least recently used (LRU) block αντικαθίσταται όταν ένα νέο block έρχεται στη μνήμη από το δίσκο.
- **Στρατηγική εγγραφών:** Χρησιμοποιείται η τεχνική 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 προσπάθεια της μνήμης προκειμένου να διαβαστεί ο πίνακας σελίδων (page table).

Page Faults

- Page faults: the data is not in memory, retrieve it from disk
 - huge miss penalty, thus pages should be fairly large (e.g., 4KB)
 - reducing page faults is important (LRU is worth the price)
 - can handle the faults in software instead of hardware
 - using write-through is too expensive so we use writeback

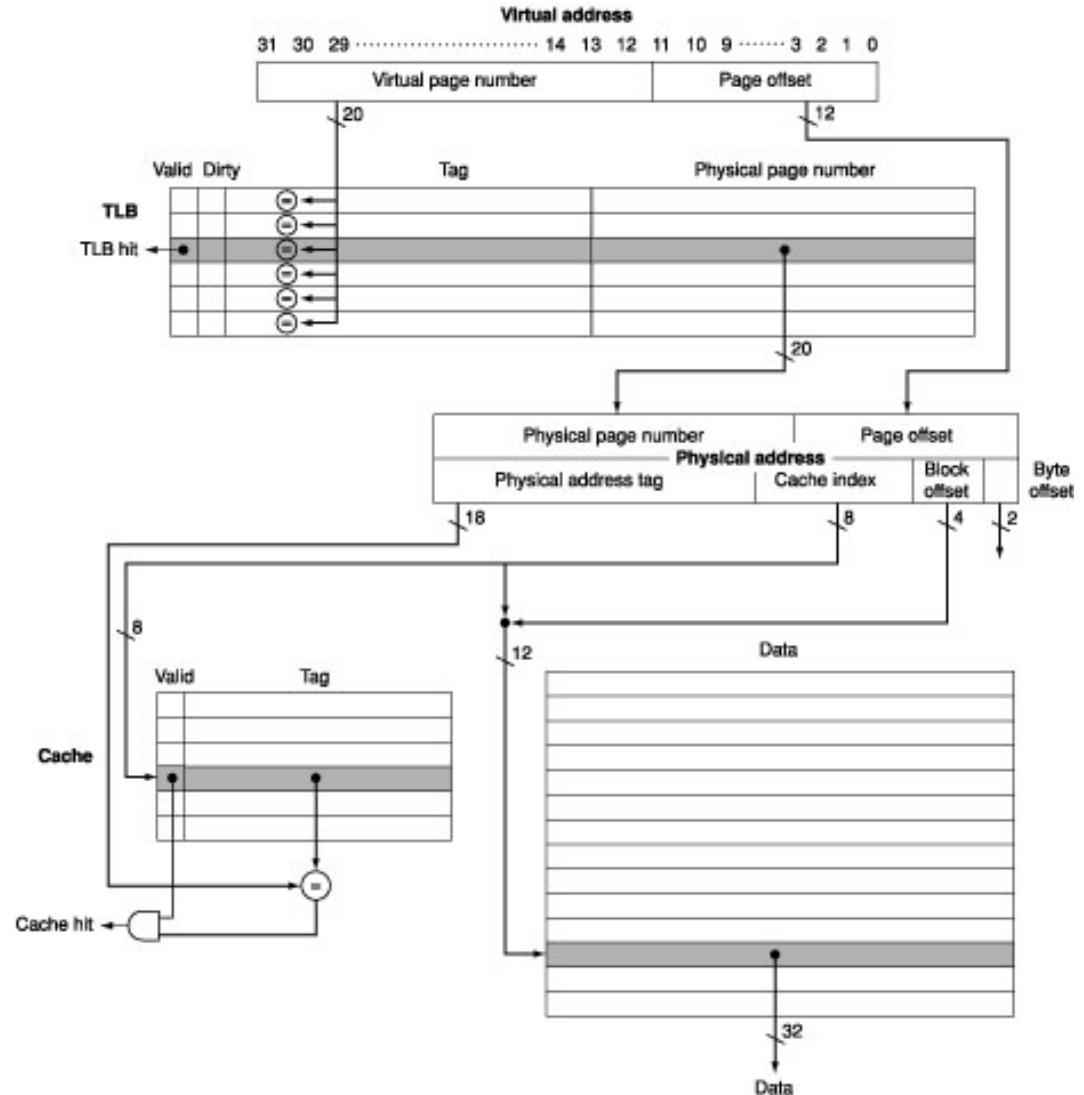
Επιτάχυνση της μετάφρασης διευθύνσεων: Translation Lookaside Buffer (TLB)

- 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 προσπελάζεται ο πίνακας σελίδων της κύριας μνήμης.



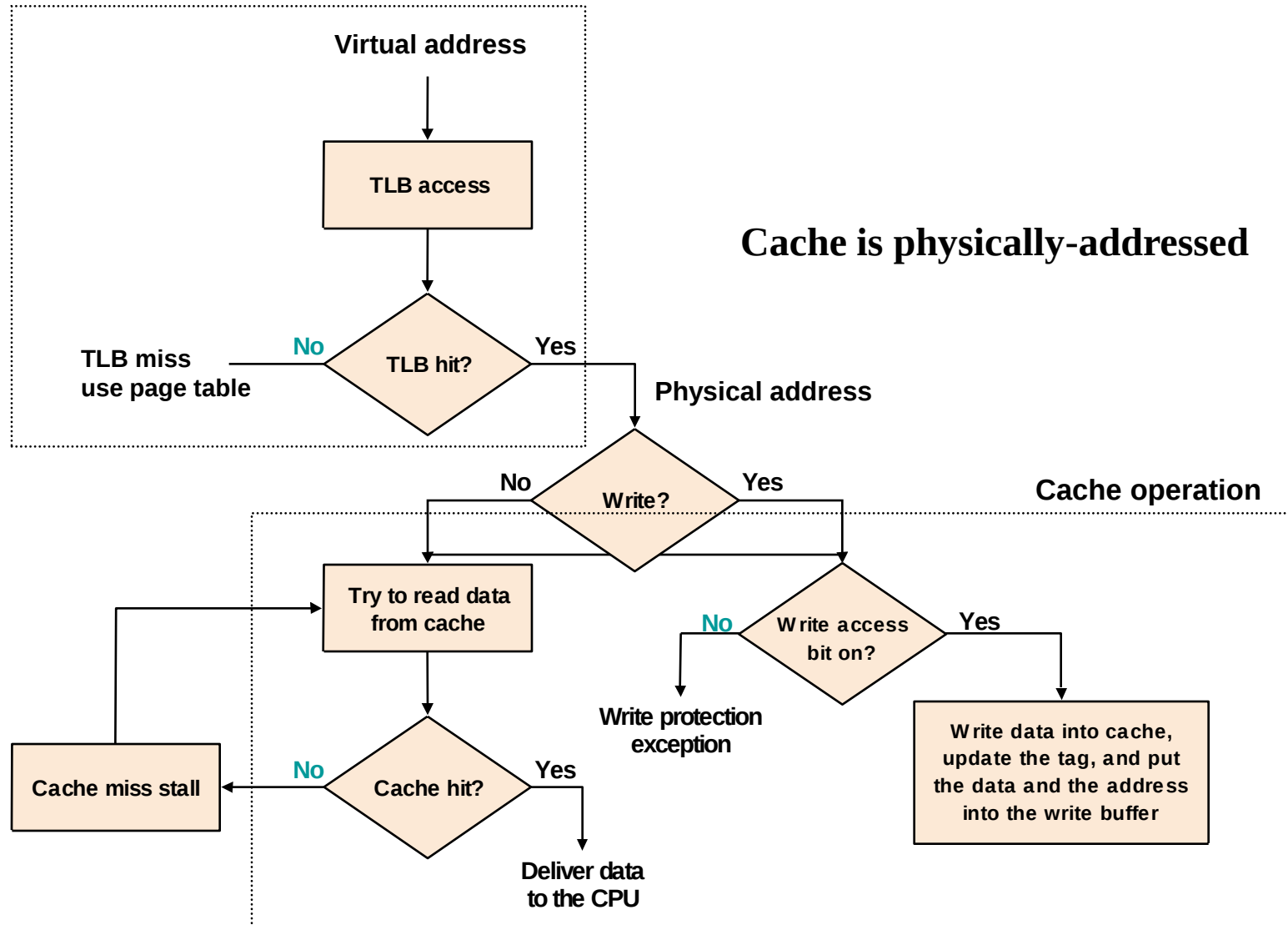
Intrinsity fastMATH processor

- 32 bit address space/byte addressing
- 4KB/page
- TLB fully associative: 16 entries



TLB & Cache Operation (Intrinsity FastMATH)

TLB Operation



TLB, virtual memory, cache συνδυασμοί

TLB	Page Table	Cache	Possible? If so under what circumstance?
hit	hit	miss	Possible, although the page table is never really checked if TLB hits
miss	hit	hit	TLB misses, but entry found in page table; after retry, data is found in cache
miss	hit	miss	TLB misses, but entry found in page table; after retry data misses in cache
miss	miss	miss	TLB misses and is followed by a page fault; after retry, data must miss in cache
hit	miss	miss	Impossible: cannot have a translation in TLB if page is not present in memory
hit	miss	hit	Impossible: cannot have a translation in TLB if page is not present in memory
miss	miss	hit	Impossible: data cannot be allowed in cache if the page is not in memory

Pentium P4 / AMD Opteron

Characteristic	Intel Pentium P4	AMD Opteron
Virtual address	32 bits	48 bits
Physical address	36 bits	40 bits
Page size	4 KB, 2/4 MB	4 KB, 2/4 MB
TLB organization	1 TLB for instructions and 1 TLB for data Both are four-way set associative Both use pseudo-LRU replacement Both have 128 entries TLB misses handled in hardware	2 TLBs for instructions and 2 TLBs for data Both L1 TLBs fully associative, LRU replacement Both L2 TLBs are four-way set associativity, round-robin LRU Both L1 TLBs have 40 entries Both L2 TLBs have 512 entries TLB misses handled in hardware

FIGURE 7.34 Address translation and TLB hardware for the Intel Pentium P4 and AMD Opteron. The word size sets the maximum size of the virtual address, but a processor need not use all bits. The physical address size is independent of word size. The P4 has one TLB for instructions and a separate identical TLB for data, while the Opteron has both an L1 TLB and an L2 TLB for instructions and identical L1 and L2 TLBs for data. Both processors provide support for large pages, which are used for things like the operating system or mapping a frame buffer. The large-page scheme avoids using a large number of entries to map a single object that is always present.

Pentium P4 / AMD Opteron

Characteristic	Intel Pentium P4	AMD Opteron
L1 cache organization	Split Instruction and data caches	Split Instruction and data caches
L1 cache size	8 KB for data, 96 KB trace cache for RISC Instructions (12K RISC operations)	64 KB each for Instructions/data
L1 cache associativity	4-way set associative	2-way set associative
L1 replacement	Approximated LRU replacement	LRU replacement
L1 block size	64 bytes	64 bytes
L1 write policy	Write-through	Write-back
L2 cache organization	Unified (Instruction and data)	Unified (Instruction and data)
L2 cache size	512 KB	1024 KB (1 MB)
L2 cache associativity	8-way set associative	16-way set associative
L2 replacement	Approximated LRU replacement	Approximated LRU replacement
L2 block size	128 bytes	64 bytes
L2 write policy	Write-back	Write-back

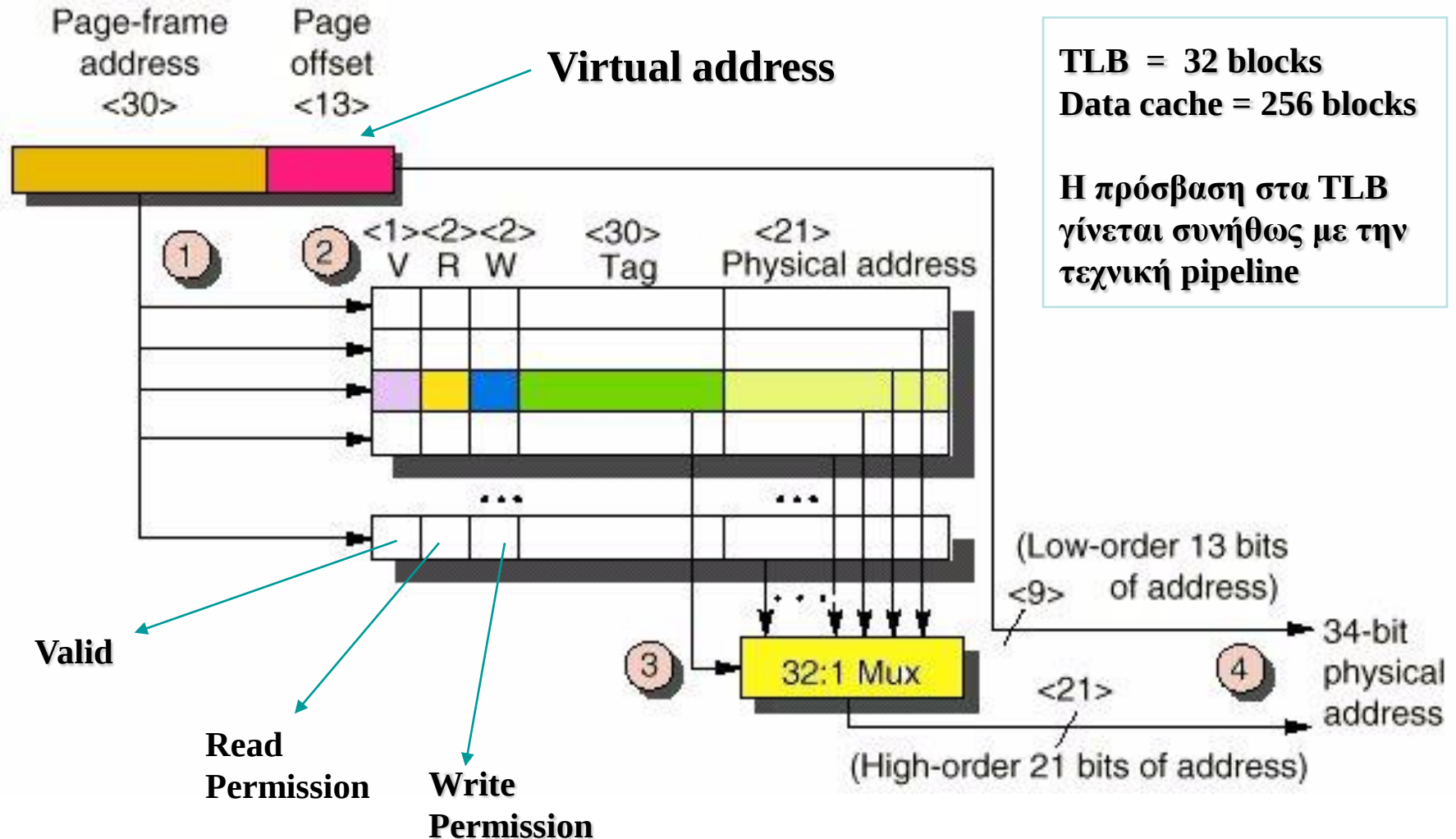
FIGURE 7.35 First-level and second-level caches in the Intel Pentium P4 and AMD Opteron. The primary caches in the P4 are physically indexed and tagged; for a discussion of the alternatives, see the Elaboration on page 527.

Desktop/embedded/server microprocessors (2004)

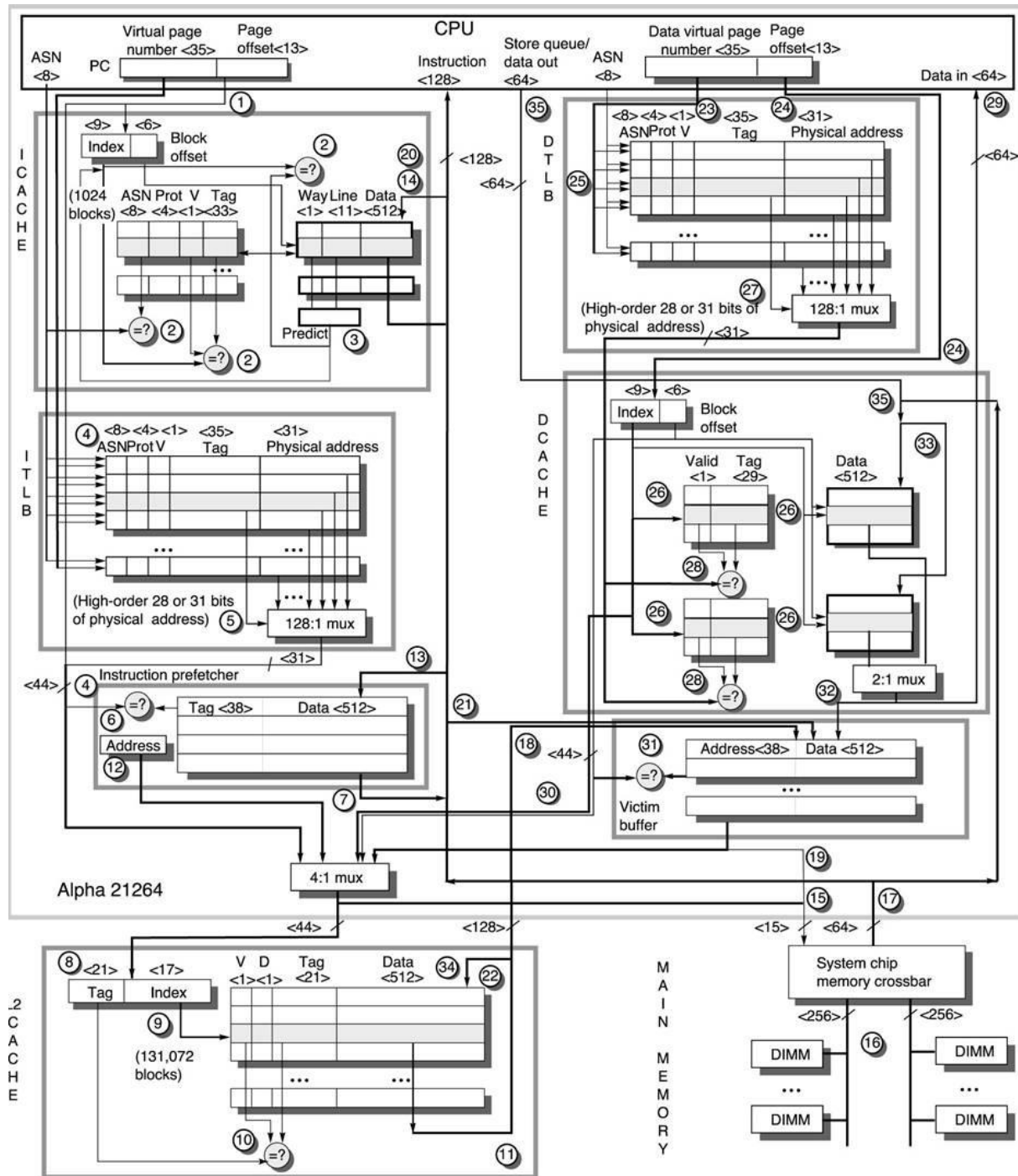
MPU	AMD Opteron	Intrinsity FastMATH	Intel Pentium 4	Intel PXA250	Sun UltraSPARC IV
Instruction set architecture	IA-32, AMD64	MIPS32	IA-32	ARM	SPARC v9
Intended application	server	embedded	desktop	low-power embedded	server
Die size (mm ²) (2004)	193	122	217		356
Instructions Issued/clock	3	2	3 RISC ops	1	4 × 2
Clock rate (2004)	2.0 GHz	2.0 GHz	3.2 GHz	0.4 GHz	1.2 GHz
Instruction cache	64 KB, 2-way set associative	16 KB, direct mapped	12000 RISC op trace cache (~96 KB)	32 KB, 32-way set associative	32 KB, 4-way set associative
Latency (clocks)	3?	4	4	1	2
Data cache	64 KB, 2-way set associative	16 KB, 1-way set associative	8 KB, 4-way set associative	32 KB, 32-way set associative	64 KB, 4-way set associative
Latency (clocks)	3	3	2	1	2
TLB entries (I/D/L2 TLB)	40/40/512/512	16	128/128	32/32	128/512
Minimum page size	4 KB	4 KB	4 KB	1 KB	8 KB
On-chip L2 cache	1024 KB, 16-way set associative	1024 KB, 4-way set associative	512 KB, 8-way set associative	—	—
Off-chip L2 cache	—	—	—	—	16 MB, 2-way set associative
Block size (L1/L2, bytes)	64	64	64/128	32	32

FIGURE 7.36 Desktop, embedded, and server microprocessors in 2004. From a memory hierarchy perspective, the primary differences between categories is the L2 cache. There is no L2 cache for the low-power embedded, a large on-chip L2 for the embedded and desktop, and 16 MB off chip for the server. The processor clock rates also vary: 0.4 GHz for low-power embedded, 1 GHz or higher for the rest. Note that UltraSPARC IV has two processors on the chip.

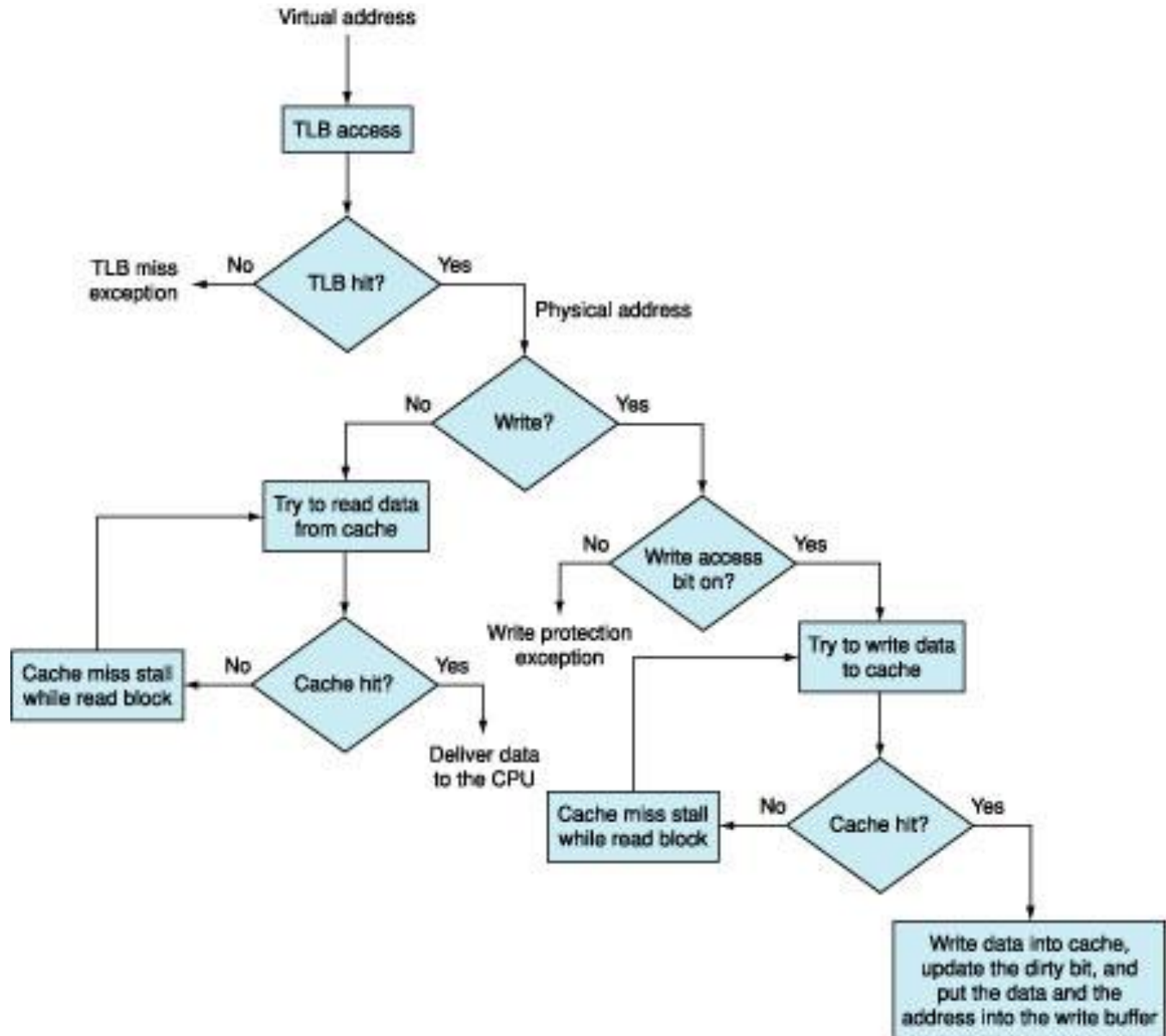
Η λειτουργία του Alpha AXP 21064 Data TLB κατά τη μετάφραση των διευθύνσεων



Σύνοψη



Read/Write (Intrinsity FastMATH cpu)



Intel Hub Architecture (850 Chipset)

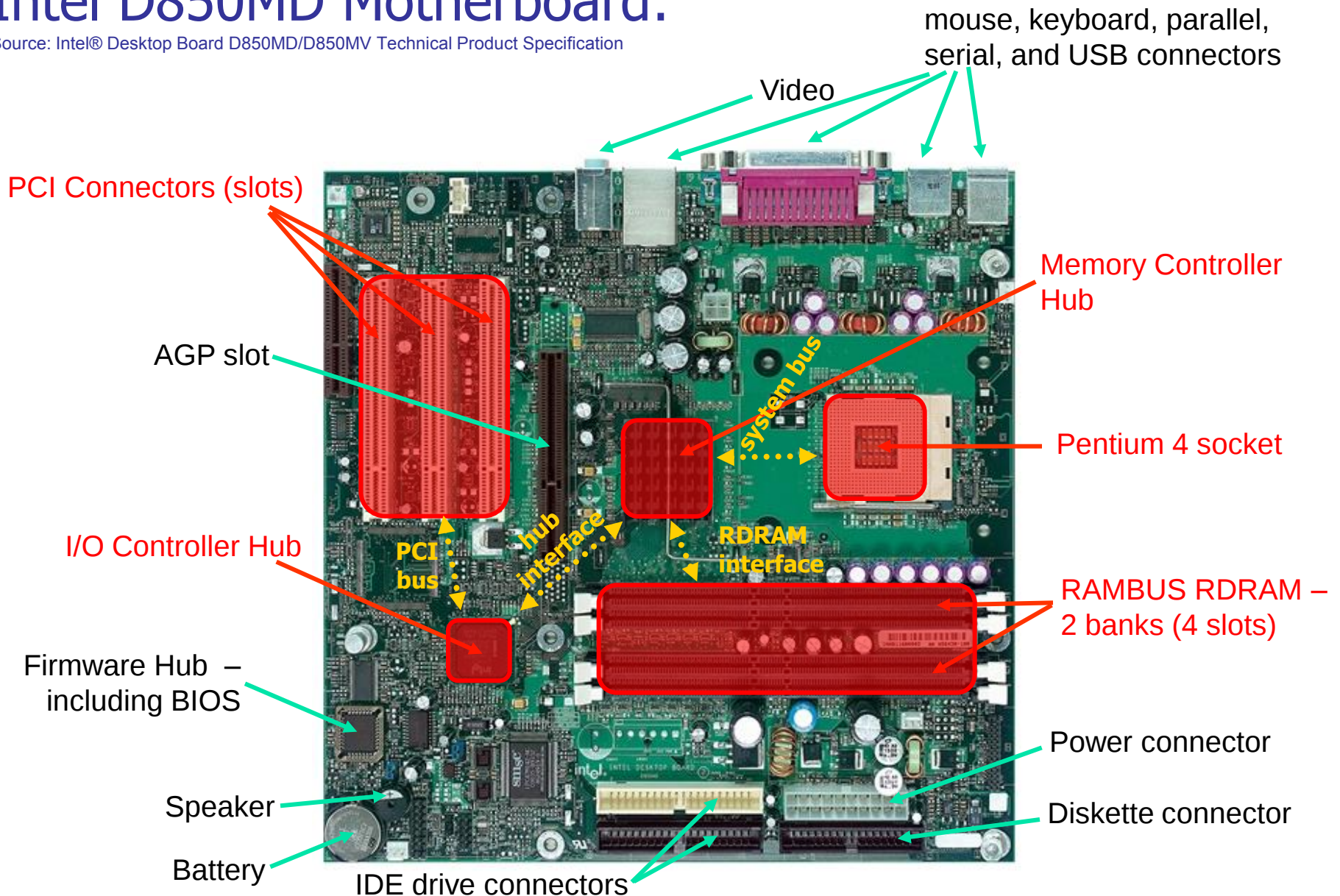


Intel D850MD Motherboard:

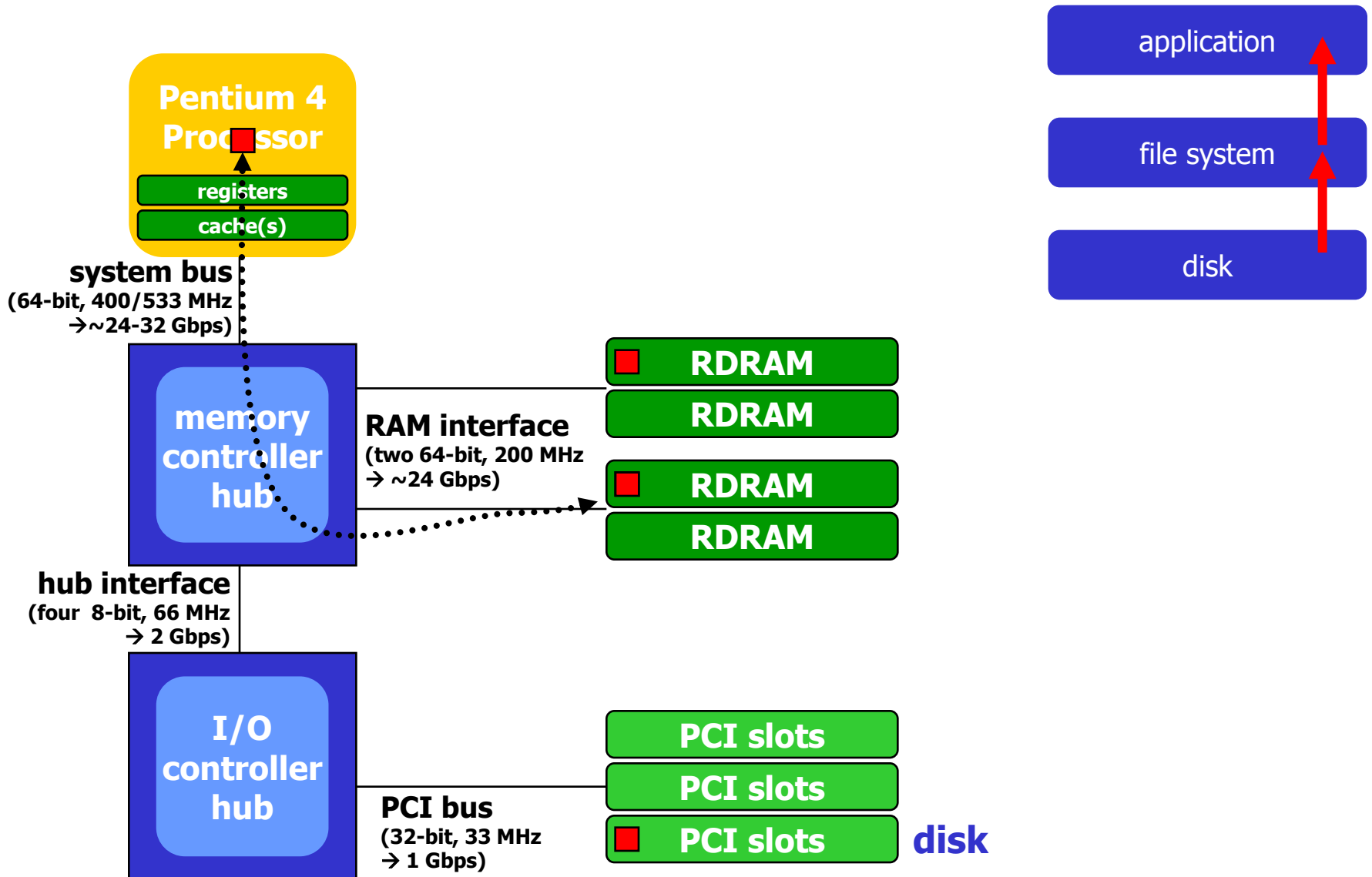
Source: Intel® Desktop Board D850MD/D850MV Technical Product Specification

Intel D850MD Motherboard:

Source: Intel® Desktop Board D850MD/D850MV Technical Product Specification



Example: Intel Hub Architecture (850 Chipset)



Intel 32-bit Architecture (IA32): Basic Execution Environment

- ✓ **Address space:** $1 - 2^{36}$ (64 GB),
each program may have a linear address space of 4 GB (2^{32})

- ✓ **Basic program execution registers:**

- 8 general purpose registers (EAX, EBX, ECX, EDX, ESI, EDI, EBP and ESP)
- 6 segment registers (CS, DS, SS, ES, FS and GS)
- 1 flag register (EFLAGS)
- 1 instruction pointer register (EIP)

- ✓ **Stack** – a continuous array of memory locations

- Current stack is referenced by the SS register
- ESP register – stack pointer
- EBP register – stack frame base pointer (fixed reference)
- PUSH – stack grow, add item (ESP decrement)
- POP – remove item, stack shrinks (ESP increment)

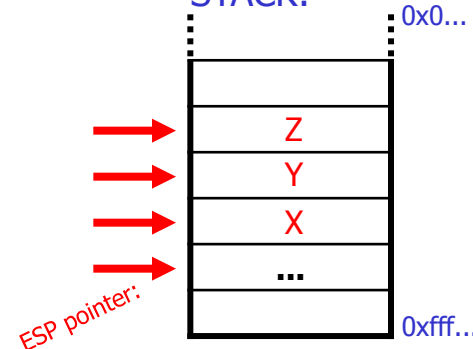
- ✓ Several other registers like Control, MMX, XMM, FPU, MTRR, MSR and performance monitoring

PUSH %eax
PUSH %ebx
PUSH %ecx
<do something>
POP %ecx
POP %ebx
POP %eax

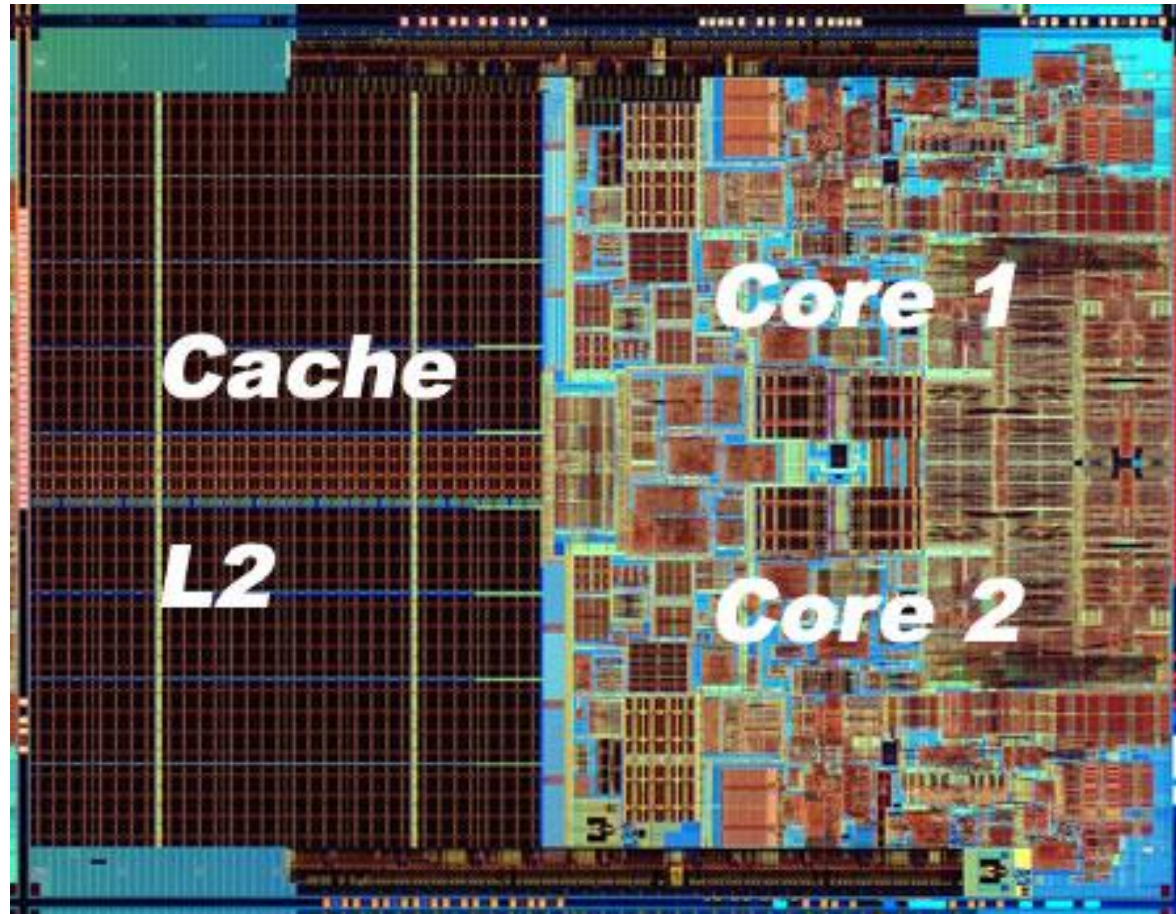
GPRs:

EAX:	X
EBX:	Y
ECX:	Z
EDX:	
ESI:	
EDI:	
EBP:	
ESP:	

STACK:



Core 2 CPU die (Conroe μ arch)



Intel i975X chipset

