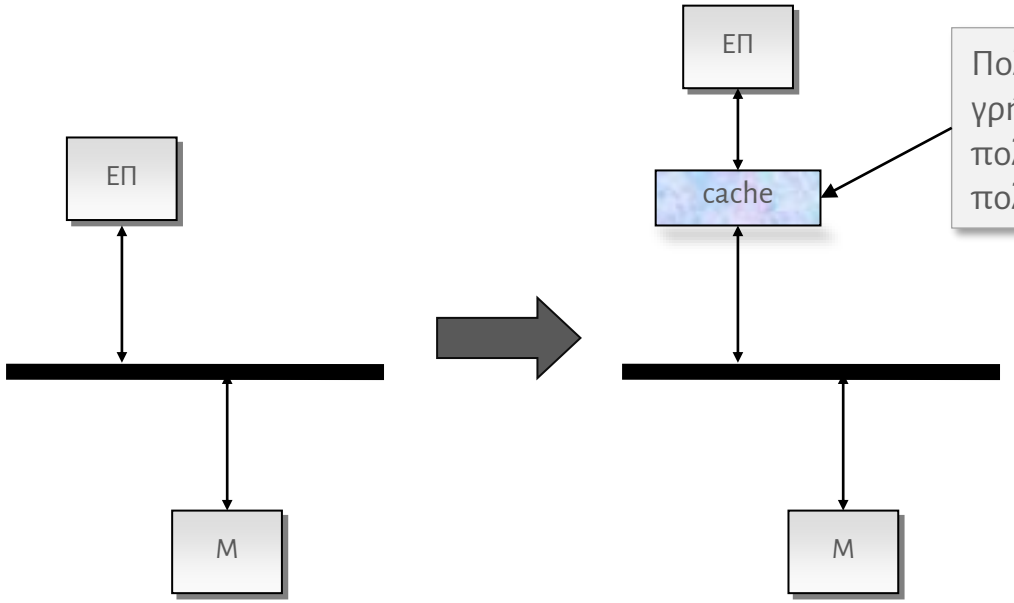
28/3/2023
Οργάνωση κοινόχρηστης μνήμης (II)

Η λειτουργία της μνήμης

Λ8
Συστήματα & Λογισμικό Υψηλών Επιδόσεων

Ιεραρχία μνήμης & cache

- Επεξεργαστής: ταχύτατος
- Μνήμη: αργή (και μάλιστα η διαφορά ταχύτητας αυξάνεται)
- Βασική λύση: *κρυφή μνήμη* (***cache***)



Πολύ γρήγορη, πολύ ακριβή, πολύ μικρή

Στηρίζεται στο γεγονός ότι οι εφαρμογές διαθέτουν *χωρική* και *χρονική τοπικότητα* ***(locality)***

Ποσοστά επιτυχίας (**hit rate**) 70% –95% δεν είναι σπάνια

- Επικοινωνία με επεξεργαστή: ***λέξεις***
- Επικοινωνία με μνήμη: ***blocks λέξεων***
  - » τοποθετούνται στις γραμμές της cache ***(cache lines)***

(Κάποια από τα) Ζητήματα των cache

- Μέγεθος cache vs. μέγεθος cache line
- Τύπος αποθήκευσης:
  - **Split** (ξεχωριστοί χώροι για αποθήκευση εντολών / δεδομένων)
  - **Unified** (ενιαίος χώρος για όλα)
- Οργάνωση / συσχετιστικότητα
  - **Direct-mapped**
  - **Set-associative** (2- or 4-ways common)
  - **Fully associative**
- Πολιτικές αντικατάστασης (replacement)
  - **LRU**, **FIFO**, **random**, … (pseudo-LRU common)

Συμπεριφορά σε αναγνώσεις (read)

- Read **hit**
  - Η ζητούμενη λέξη ξεχωρίζει από την γραμμή της cache και παραδίδεται στον επεξεργαστή.
- Read **miss**
  - Το μπλοκ που περιλαμβάνει τη ζητούμενη λέξη προσκομίζεται από τη μνήμη, τοποθετείται σε μία γραμμή της cache και η λέξη παραδίδεται στον επεξεργαστή
  - Μεγάλη καθυστέρηση

- Hit rate (ή hit ratio) =
  # προσπελάσεων που ήταν hit / συνολικός # προσπελάσεων

Συμπεριφορά σε εγγραφές

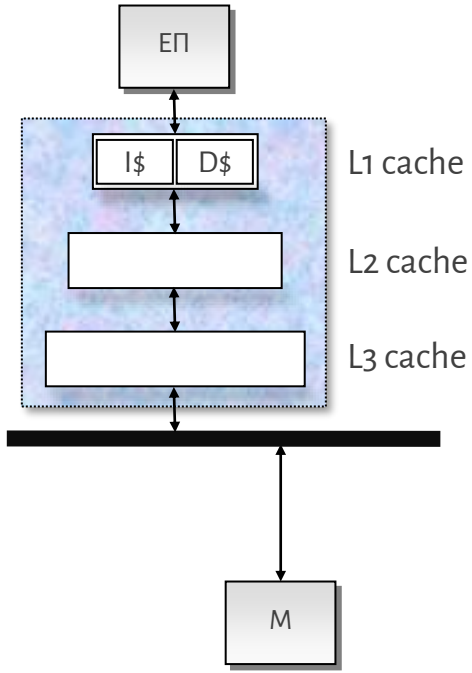
1. Στην περίπτωση που το δεδομένο υπάρχει στην cache (*write hit*)
   - Διεγγραφή (**write-through**): άμεση εγγραφή και στην κύρια μνήμη
   - Υστεροεγγραφή (**write-back**): ΔΕΝ ενημερώνεται η κύρια μνήμη

     Πώς / πότε ενημερώνεται η κύρια μνήμη;
     - Μόνο όταν έρθει η ώρα να αντικατασταθεί το τροποποιημένο (*dirty*) δεδομένο στην cache.
   - Η υστεροεγγραφή προτιμάται: δημιουργεί λιγότερη κίνηση προς τη μνήμη (αλλά επίσης και προβλήματα ενημέρωσης…). Βελτίωση της διεγγραφής γίνεται με *write-buffers*.

2. Στην περίπτωση που το δεδομένο ΔΕΝ υπάρχει στην cache (*write miss*)
   - **No-allocate**: γίνεται εγγραφή κατευθείαν στην κύρια μνήμη
   - **Write-allocate**: το δεδομένο έρχεται πρώτα στην cache (όπως στο read) και στη συνέχεια τροποποιείται
     (πιο χρονοβόρο/χωροβόρο αλλά προτιμάται μιας και ελπίζουμε ότι θα υπάρξει read αργότερα έτσι κι αλλιώς)

Γενίκευση σε πολλαπλά επίπεδα (ιεραρχία caches)

- Για μείωση του κόστους αστοχίας (miss)
- 2-3 επίπεδα από αυξανόμενου μεγέθους και μειούμενου κόστους / ταχύτητας caches:



L1 cache ~ 16 – 64 KB, 1-4 cycles, split, write-through

L2 cache ~ 256 KB – 8 MB, 6-15 cycles

L3 cache ~ 4 – 32 MB, 30-50 cycles

~ Gbytes, 100+ cycles

Inclusion / exclusion

- Πού αποθηκεύεται ένα δεδομένο (δηλ. σε ποιο από όλα τα επίπεδα);
- Τρεις βασικές επιλογές:
  - **Inclusive cache**:

    Η cache μεγαλύτερου επιπέδου φυλάει ΟΛΕΣ τις γραμμές των μικρότερων επιπέδων. Π.χ. L3 cache στους Intel Nehalem (Xeon 55xx)
  - **Exclusive cache**:

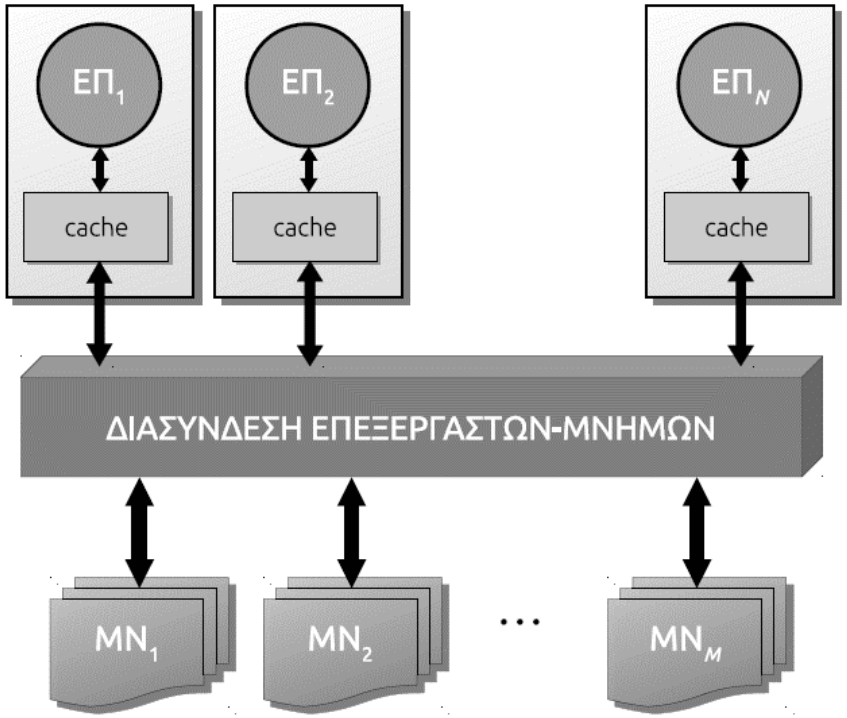
    Η cache μεγαλύτερου επιπέδου δεν φυλάει καμία γραμμή των μικρότερων επιπέδων (άρα κάθε επίπεδο αποθηκεύει εντελώς διαφορετικά πράγματα). Π.χ. L3 cache στους AMD Shangai (Opteron 238x)
  - **Non-Inclusive/Non-exclusive cache**:

    Δεν εξασφαλίζεται τίποτε από τα δύο παραπάνω. Π.χ. η L2 των Intel Sandy Bridge

- Πλεονεκτήματα / μειονεκτήματα;
  - Οι inclusive είναι γενικά πιο εύκολες στον χειρισμό των misses αλλά απαιτούν πολύ μεγαλύτερο χώρο (π.χ. η L2 θα πρέπει ένα μέρος της να το αφιερώσει στο να κρατά αντίγραφο της L1).
  - Στις inclusive ακύρωση (π.χ. αντικατάσταση) σε μεγαλύτερο επίπεδο απαιτεί ακύρωση και στα μικρότερα

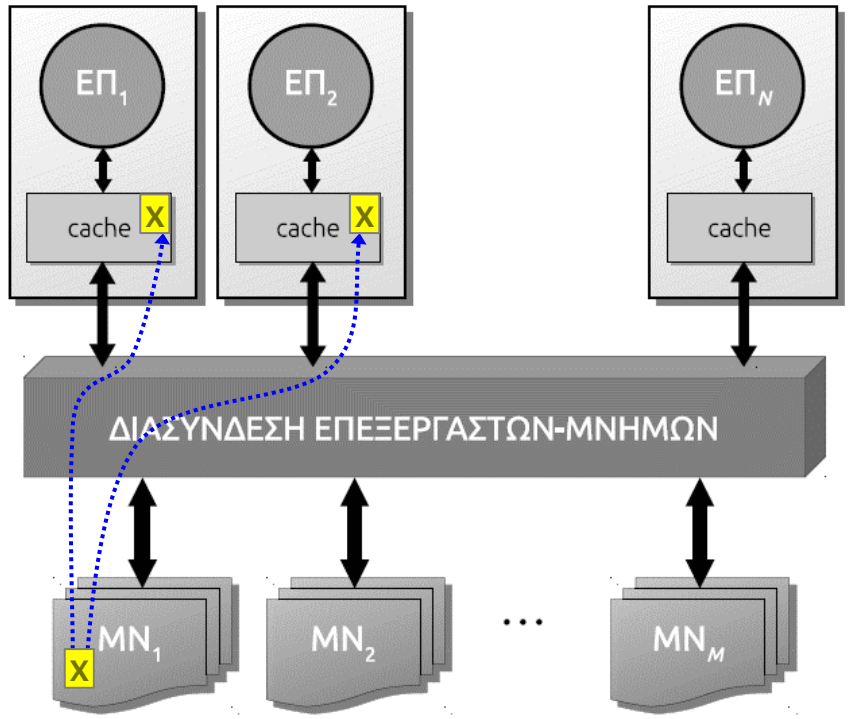
Caches σε παράλληλα συστήματα κοινής μνήμης

- Εκτός του λόγου της «αργής» κύριας μνήμης, είναι απαραίτητες διότι επιπλέον:
  - Υπάρχει συναγωνισμός επεξεργαστών στις κοινές μνήμες
  - Το δίκτυο σύνδεσης επεξεργαστών-μνημών εισάγει επιπλέον καθυστερήσεις



Το πρόβλημα

- Πρόβλημα ***συνοχής*** εφόσον κάποιος μπορεί να τροποποιήσει το δικό του αντίγραφο (**cache coherence**)



Λύσεις

- Αποτροπή του προβλήματος (δεν αφήνουμε το πρόβλημα να εμφανιστεί καν):
  - Μόνο με software
    - με ειδικό compiler που διαχωρίζει τα κοινά δεδομένα και δεν τους επιτρέπει να αντιγραφτούν στις κρυφές μνήμες
  - Με software και hardware:
    - Ειδική διάταξη που λειτουργεί σε συνδυασμό με τον compiler
- Το αφήνουμε και το λύνουμε κατά την εκτέλεση, με hardware
  - πρωτόκολλα συνοχής (coherence protocols)
    - πρωτόκολλα παρακολούθησης (snoopy/snooping protocols)
    - πρωτόκολλα καταλόγων (directory-based protocols)

Πρωτόκολλα παρακολούθησης

Snooping protocols

Πρωτόκολλα παρακολούθησης

- Δύο είδη:
  - εγγραφής – ενημέρωσης (write-update)
  - εγγραφής – ακύρωσης (write-invalidate)
- Συνήθως εγγραφής-ακύρωσης
- Συνήθως caches με πολιτική υστεροεγγραφής (write-back)
- Το πρωτόκολλο υλοποιείται μέσα στον επεξεργαστή
  - Πολλά πρωτόκολλα έχουν προταθεί
  - MSI, MESI, MOSI, MOESI, MESIF, κλπ
  - Κρίσιμα για την κίνηση στον δίαυλο

Πρωτόκολλο MSI

(*Write-back caches*)

Καταστάσεις γραμμής cache:

- Άκυρη (Invalid) ή δεν υπάρχει
  - Είτε η γραμμή δεν υπάρχει στη cache (δηλαδή miss) ή μπορεί να υπάρχει αλλά έχει ακυρωθεί (γιατί κάποιος άλλος ζήτησε να την αλλάξει)
- Μοιραζόμενη (Shared)
  - Η γραμμή δεν έχει αλλαχθεί, επομένως η κύρια μνήμη έχει ενήμερο αντίγραφο
  - Μπορεί να υπάρχουν άλλα αντίγραφα σε άλλες caches
  - Μπορεί και όχι γιατί οι αντικαταστάσεις γίνονται «αθόρυβα»
- Αλλαγμένη (Modified)
  - Έχουμε το μοναδικό ενήμερο αντίγραφο της γραμμής

MSI: Διάγραμμα καταστάσεων γραμμής

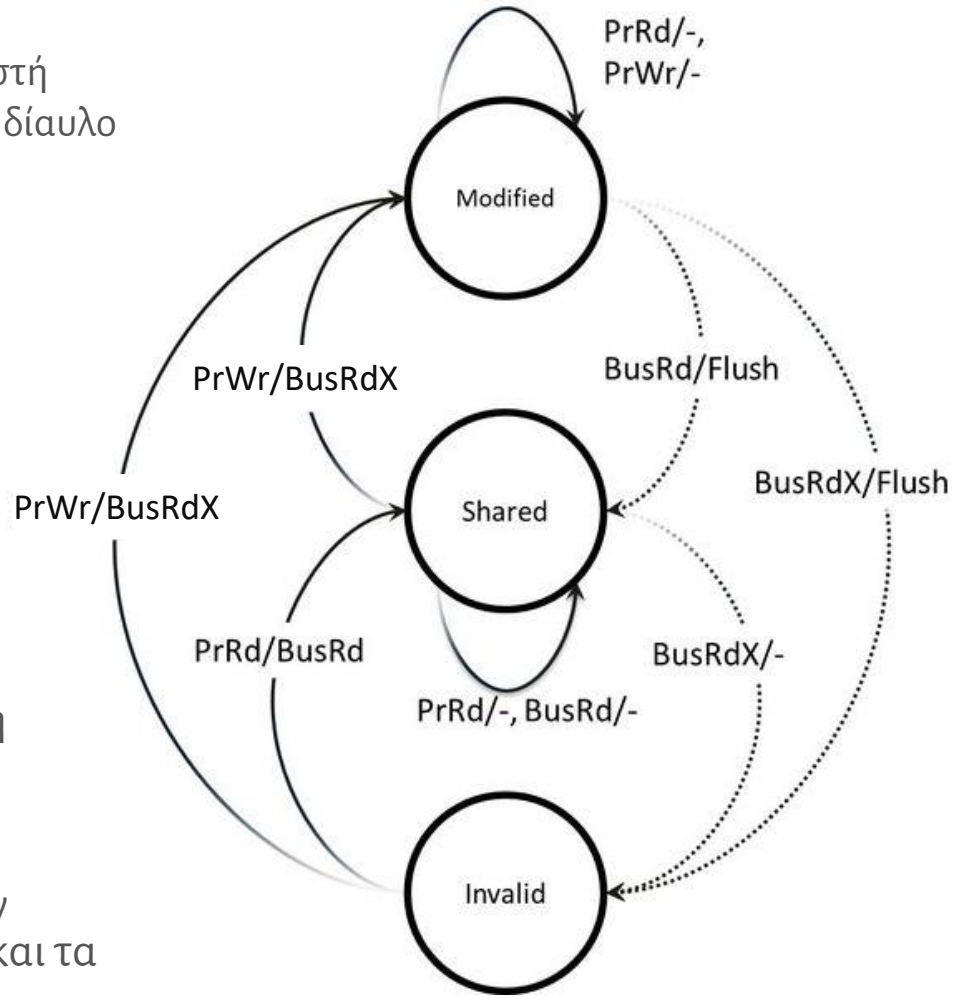
Συμβολισμός: **Γεγονός/Ενέργεια**

PrXX – γεγονός από τοπικό επεξεργαστή

BusXX – γεγονός από ή ενέργεια προς δίαυλο

- **PrRd**, **PrWr** – ανάγνωση, εγγραφή από τον πυρήνα
- **BusRd**, **BusRdX** – διαδικασία ανάγνωσης, αποκλειστικής (exclusive) ανάγνωσης στο δίαυλο
  - το BusRdX ακυρώνει τα όποια αντίγραφα
- **Flush** – δίνω την τιμή της γραμμής στο δίαυλο και ανανεώνεται η κύρια μνήμη (μπορούν να πάρουν την τιμή και οι άλλες cache)

(Στο δίαυλο δεν φαίνονται τα write των επεξεργαστών, παρά μόνο τα flush και τα BusRdX)



Πρωτόκολλο MESI ή Illinois

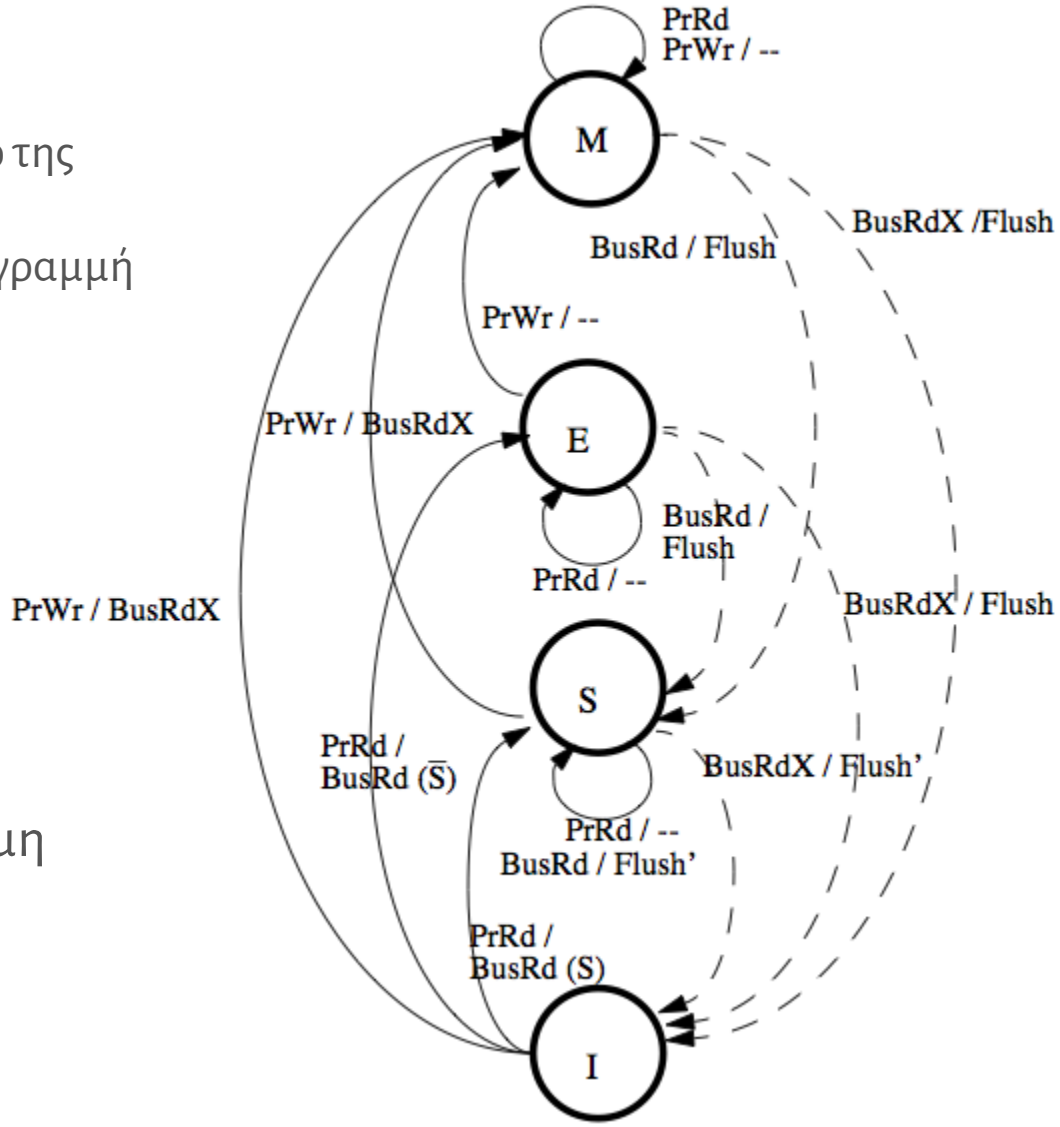
- Αν διαβάσουμε μια καινούρια γραμμή και αμέσως μετά θέλουμε να την αλλάξουμε, χωρίς να (θέλει να) παραμβληθεί άλλος επεξεργαστής, στο MSI:
  - έχουμε 2 misses / κινήσεις στο δίαυλο
    - μία για να φέρουμε τη γραμμή για πρώτη φορά (κατάσταση S)
    - και άλλη μία μόνο για να αλλάξουμε τη κατάσταση σε M
- Αυτό γίνεται πολύ συχνά και σίγουρα γίνεται και σε σειριακά προγράμματα (τα οποία δε θέλουμε να βασανίζουμε!)
- Λύση: κρατάμε πληροφορία αν έχουμε (διαβάσει) το μοναδικό αντίγραφο της γραμμής μέχρι τώρα
  - 4η κατάσταση: αποκλειστική (exclusive)
  - Στην κατάσταση αυτή έχουμε το μοναδικό αντίγραφο και η κύρια μνήμη είναι ενημερωμένη

MESI: Διάγραμμα καταστάσεων

Σήμα S στον δίαυλο:

- Αν μια cache έχει αντίγραφο της γραμμής θέτει το σήμα (S)
- Αν κανείς δεν το θέσει (S) η γραμμή είναι αποκλειστική

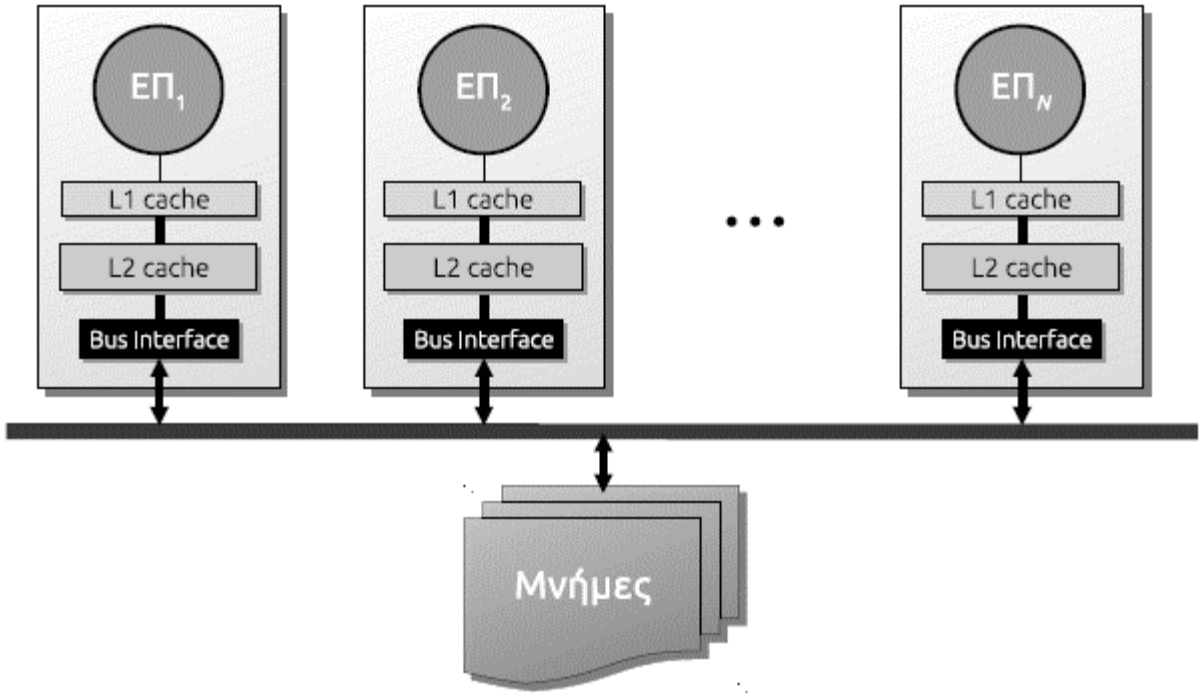
- Από S -> M δεν υπάρχει κίνηση στον δίαυλο.
- Flush – δίνω την τιμή της γραμμής στο δίαυλο και ανανεώνεται η κύρια μνήμη



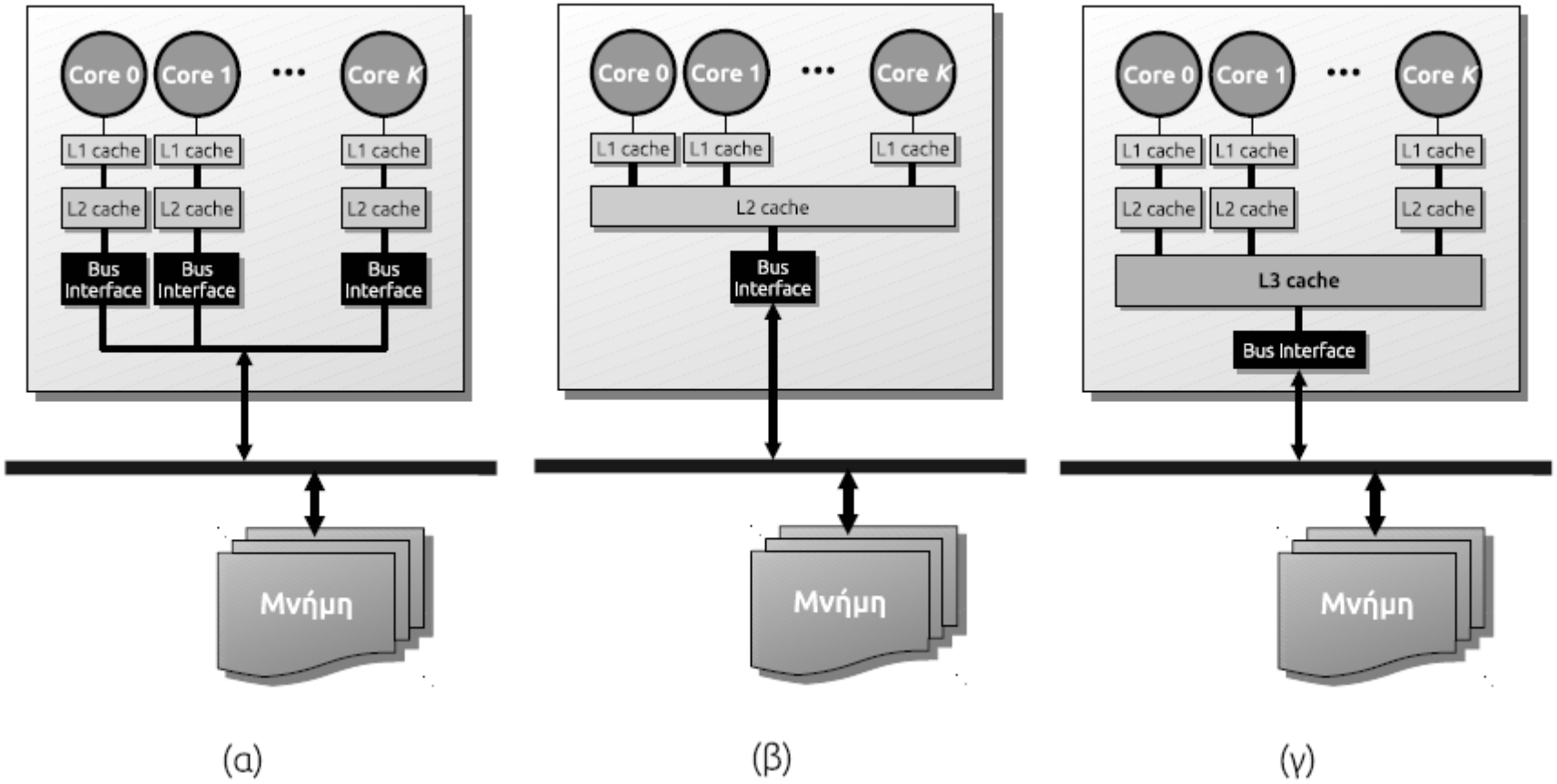
Πρωτόκολλα MOESI και MESIF

- Αν μετά την αλλαγή μιας γραμμής, η γραμμή χρησιμοποιηθεί για ανάγνωση, πρέπει να γραφτεί στη κύρια μνήμη
  - flush έξω από κατάσταση M
  - Η κύρια μνήμη είναι γενικά αργή, οπότε συμφέρει να καθυστερήσουμε την εγγραφή όσο μπορούμε (η cache που έχει τη σωστή τιμή θα τη δίνει σε όσους την χρειάζονται)
  - Χρειαζόμαστε μια επιπλέον κατάσταση όπου η γραμμή είναι αλλαγμένη αλλά υπάρχουν αντίγραφα σε άλλες caches: O – owned
- Αντίστοιχα, όταν μια νέα cache ζητάει το δεδομένο και αυτό δεν έχει τροποποιηθεί, γιατί να το δίνει η μνήμη και όχι μία άλλη cache πιο γρήγορα;
  - Χρειαζόμαστε μία cache που το έχει να το προωθεί (F - forward)
- HOMEWORK !!

Caches σε πολυεπεξεργαστές

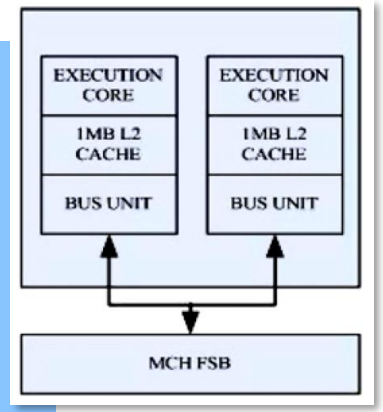


Caches σε multicore

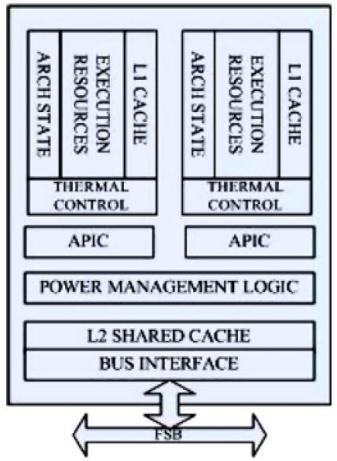


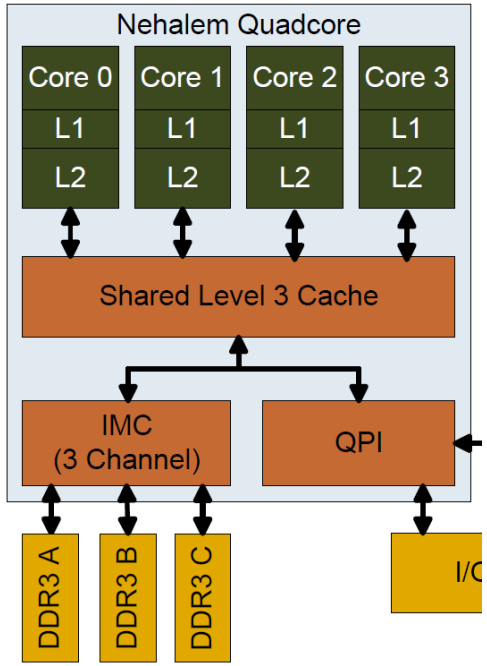
Παραδείγματα

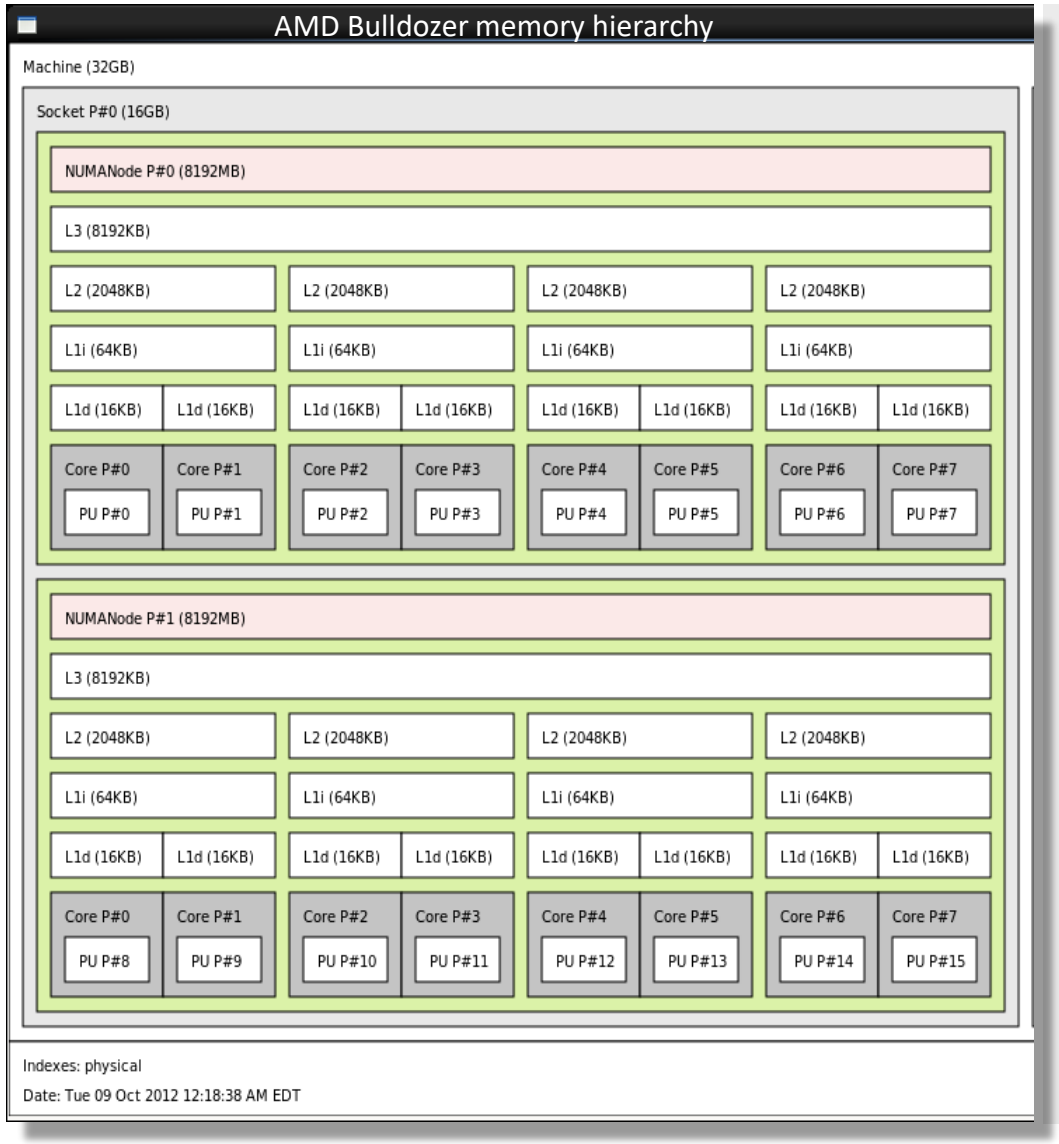
Pentium D



Core 2 Duo







Πλεονεκτήματα κοινόχρηστων cache

- Μείωση χρόνου επικοινωνίας των πυρήνων μέσω της κοινής cache
- Επεξεργαστές που δουλεύουν σε επικαλυπτόμενες περιοχές δεδομένων
  - Ο ένας μπορεί να φέρει δεδομένα που ίσως χρησιμοποιήσει κάποιος άλλος
  - Μειώνεται ο χώρος που απαιτείται
  - Ταχύτητα επικοινωνίας – μείωση κίνησης στο δίαυλο
- Δυναμική διαμοίραση
  - Αν ένας επεξεργαστής χρειάζεται λιγότερο χώρο, κάποιος άλλος μπορεί να πάρει περισσότερο
- Δεν υπάρχει θέμα συνοχής στην κοινόχρηστη cache
- Αποφυγή του *false sharing*

Μειονεκτήματα κοινόχρηστης cache

- Πολλαπλοί πυρήνες
  - Απαίτηση για μεγαλύτερο μέγεθος cache (και άρα και πιο αργή)
  - Απαίτηση για μεγαλύτερο ρυθμό μεταφοράς (μιας και προσπελαύνεται από αρκετές cache μικρότερου επιπέδου ταυτόχρονα)
- Προσπέλαση από πολλαπλές cache μικρότερου επιπέδου => συνήθως crossbar switch => κάποια αύξηση καθυστέρησης προσπέλασης
- Πολυπλοκότερη σχεδίαση
- Ένας πυρήνας μπορεί να είναι «μοναχοφάης» και να την γεμίζει με δικά του δεδομένα, προκαλώντας προβλήματα στους άλλους

Πρωτόκολλα καταλόγων

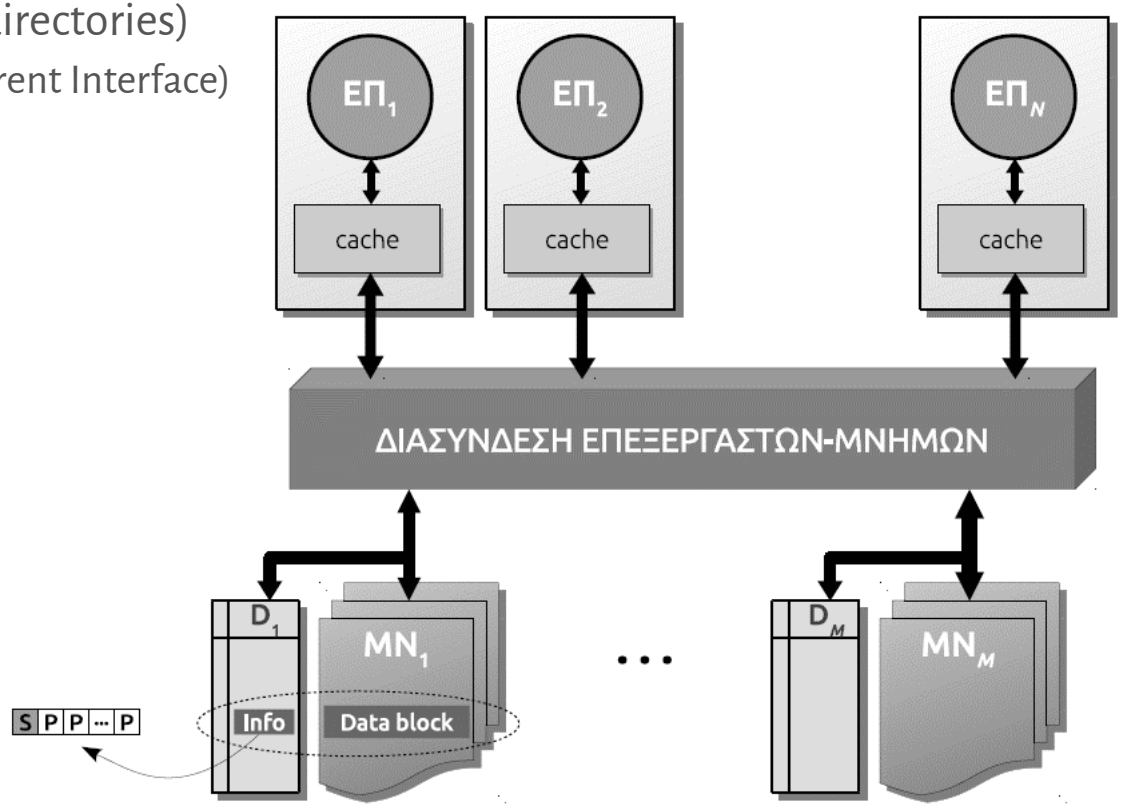
Directory protocols

Πρωτόκολλα με καταλόγους

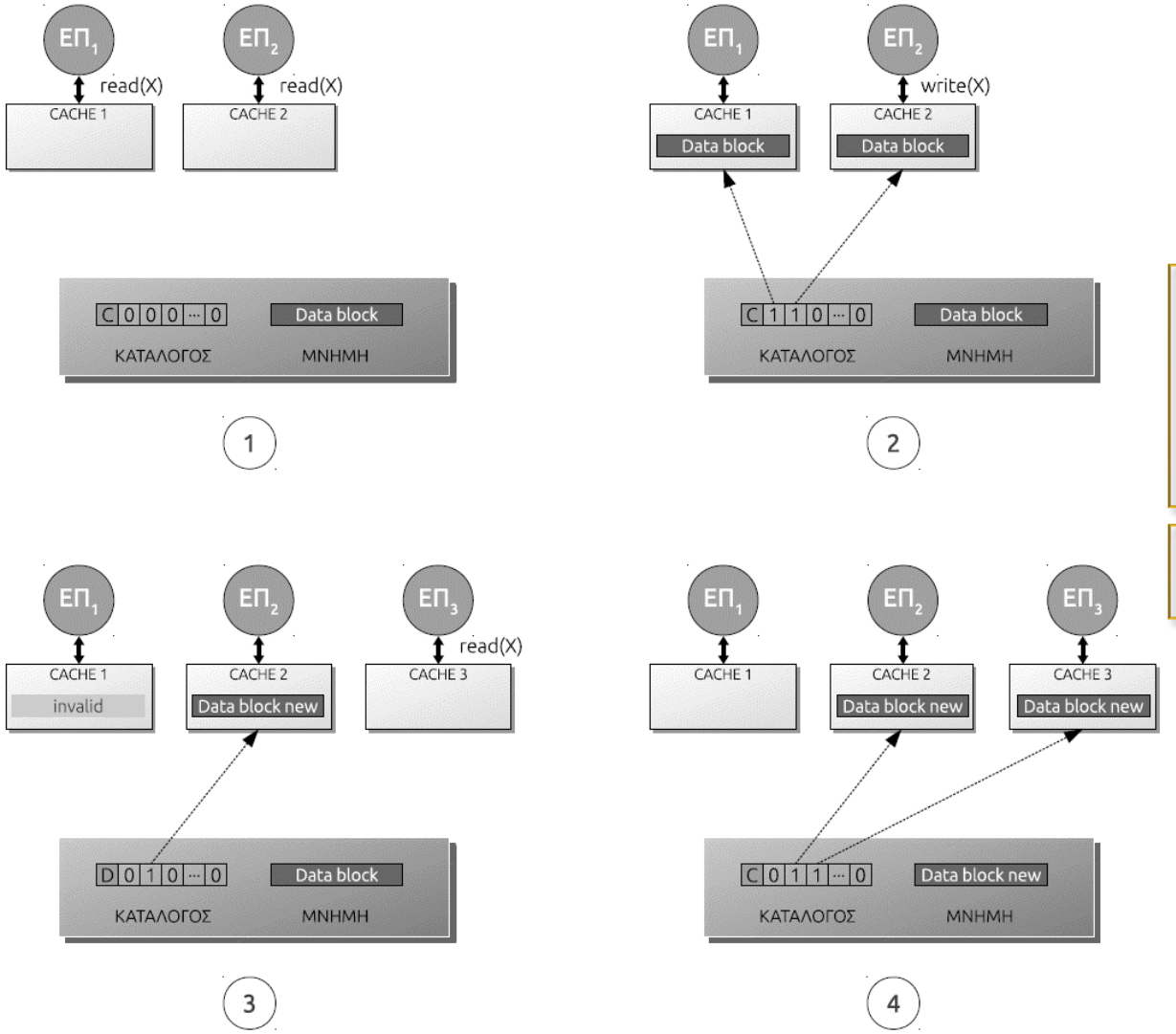
- Δεν υπάρχει κοινό μέσο (π.χ. διασύνδεση με διακοπτικά δίκτυα ή συστήματα NUMA που θα δούμε αργότερα)
  - Άρα αδύνατα / ασύμφορα τα πρωτόκολλα παρακολούθησης
- Η μνήμη είναι υπεύθυνη για όλα
  - Κατάλογος όπου καταγράφεται ποιες cache έχουν αντίγραφο των δεδομένων
  - Επικοινωνία για συνοχή *μόνο* με αυτές τις cache
- Κεντρικός κατάλογος (κακό) ή
- Κατανεμημένοι κατάλογοι

Κατανεμημένοι κατάλογοι

- Πεδία «παρουσίας» στους καταλόγους
- Κατανεμημένοι κατάλογοι:
  - πλήρεις (full-map directories)
  - περιορισμένοι (limited directories)
  - αλυσιδωτοί (chained directories)
    - » IEEE SCI (Scalable Coherent Interface)



Πλήρεις κατάλογοι

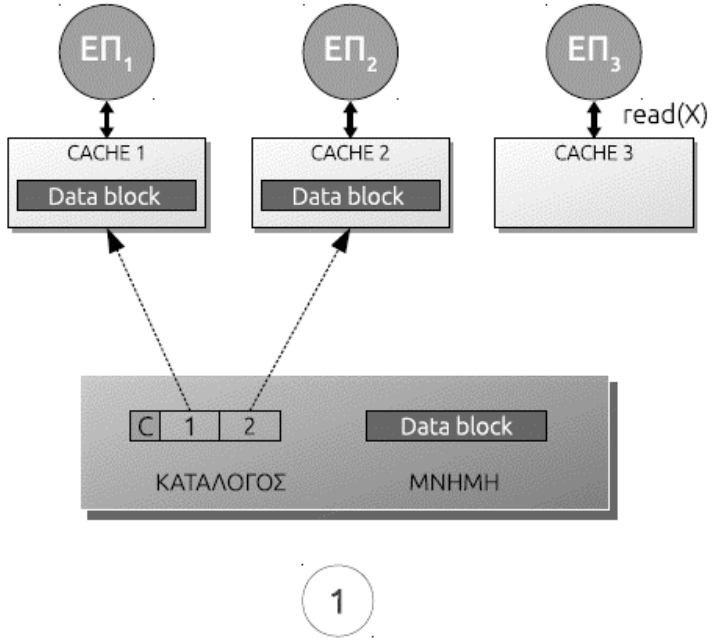


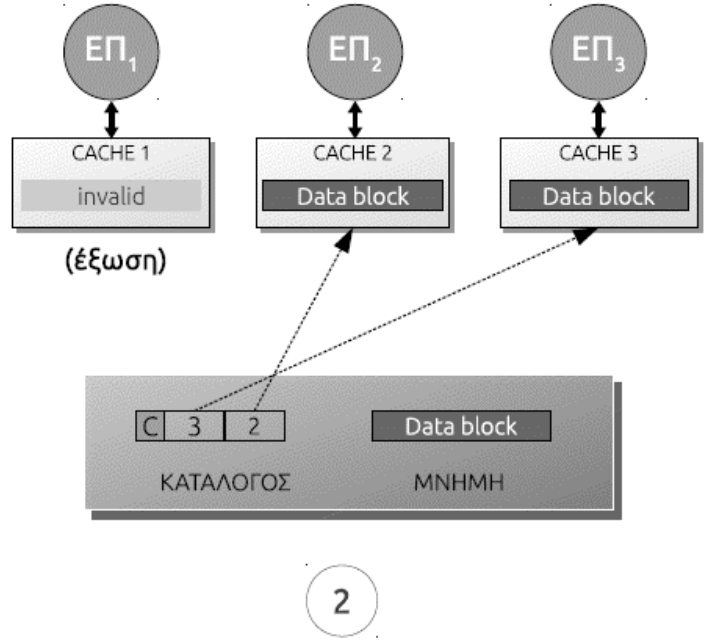
Για συστήματα μεσαίου μεγέθους

Π.χ.
- 128 cpus
- block size / cache line = 64 bytes
- ποιο το μέγεθος του directory?

Για
- 1024 cpus?

Μερικοί κατάλογοι

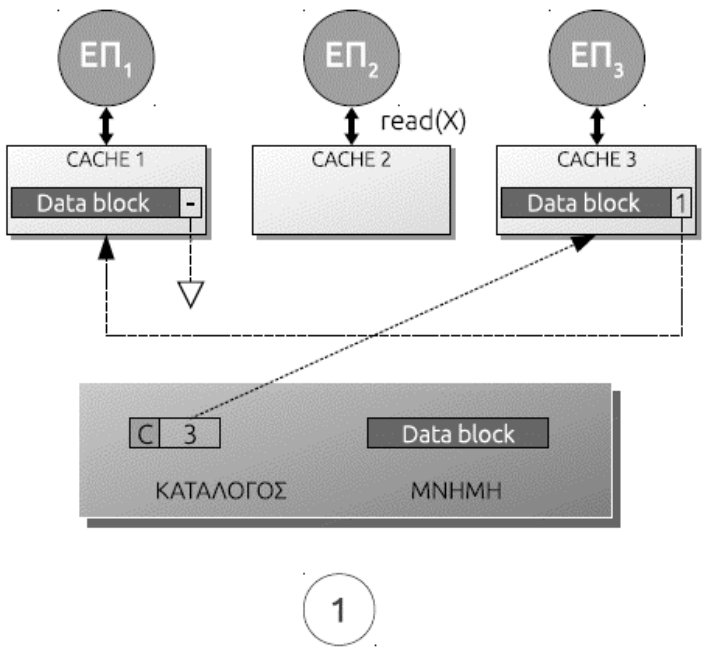


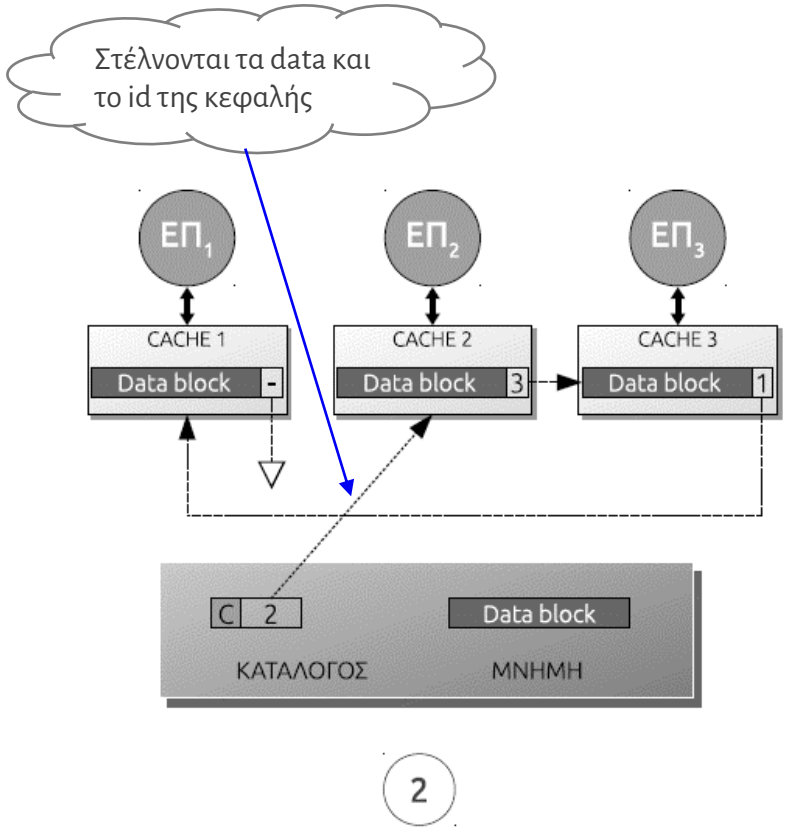


Για
- 1024 cpus
- block size / cache line = 64 bytes
- K = χώρος για 10 αντίγραφα
- ποιο το μέγεθος του directory?

K = 5 είναι αρκετός χώρος για να περιορίσει πολύ τις συχνές εξώσεις.

Αλυσιδωτοί κατάλογοι





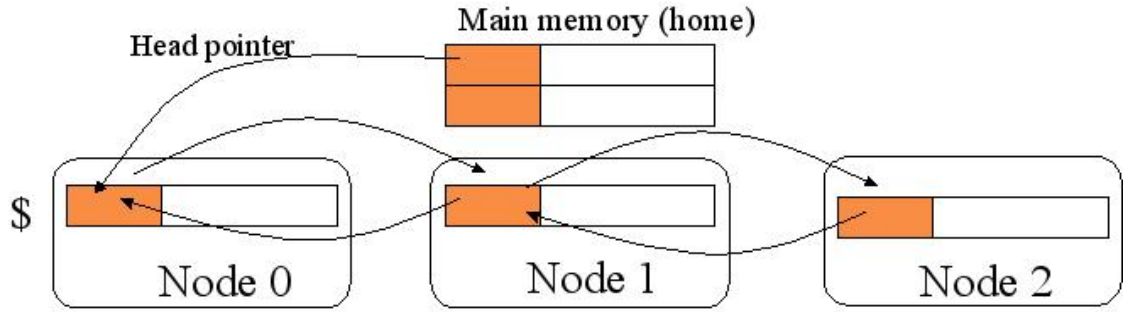
Write hit στην cache C:

- Η cache C στέλνει Write Request στη μνήμη
- Η μνήμη στέλνει πακέτο (Invalidate, C) στην κεφαλή της λίστας
- Το πακέτο ακυρώνει το αντίγραφο και προωθείται στον επόμενο κόμβο
- Ο τελευταίος κόμβος στέλνει Invalidate Acknowledge στον C.

Replacement σε μία cache:

- Η cache στέλνει Invalidate στη μνήμη
- Το Invalidate προωθείται και ακυρώνεται όλη η αλυσίδα.

Αλυσιδωτοί κατάλογοι – διπλή σύνδεση



Παράδειγμα υλοποίησης:

- Replacement:
  - Roll out της cache με επικοινωνία με προηγούμενο + επόμενο κόμβο
- Write hit: *η cache C πρέπει να πάρει exclusive access*
  - Διαβάζει πάλι το δεδομένο ώστε να μπει στην κορυφή της λίστας
  - Στέλνει purge (invalidation) στον επόμενο κόμβο ο οποίος ακυρώνει το αντίγραφό του και επιστρέφει acknowledgement + το id του επόμενου
  - Συνεχίζεται μέχρι να στείλει acknowledgement ο τελευταίος στη λίστα
- Τυπικό παράδειγμα: **IEEE Scalable Coherent Interface (SCI)**
- Θα τα ξαναδούμε όταν μιλήσουμε για μηχανές ccNUMA

Συνέπεια μνήμης

Memory consistency

Ένα απλό πρόγραμμα

Initially `A = B = 0`

Process P1	Process P2
…	…
`A = 1;  /* write(A) */`	`printf("%d", B);  /* read(B) */`
`B = 1;  /* write(B) */`	`printf("%d", A);  /* read(A) */`

- Εκτύπωση: **10** (!? Call service?)
  - Αποδεκτές εκτυπώσεις: 00, 11, 01

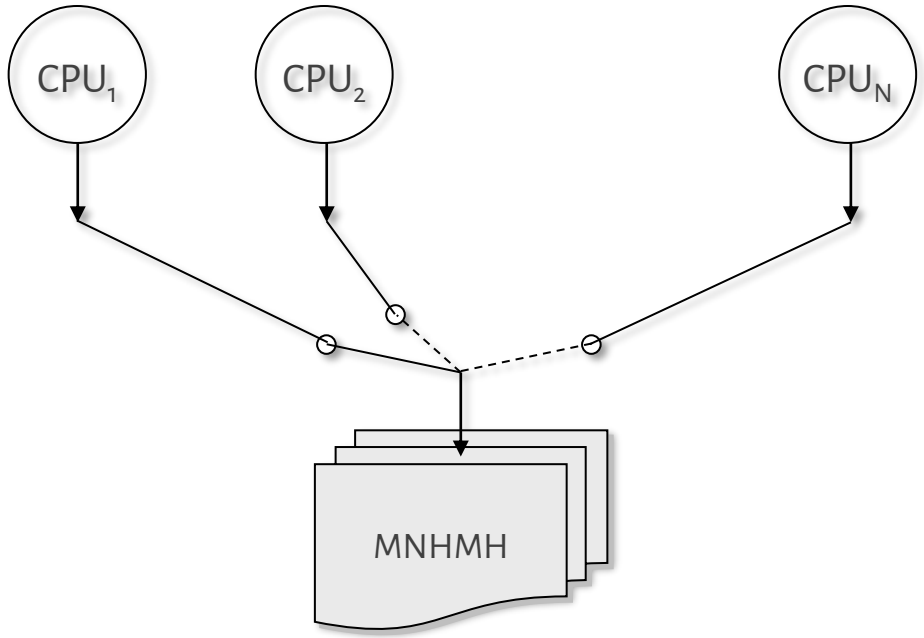
- Το πρωτόκολλο συνέπειας (cache coherency) εξασφαλίζει ότι τα αντίγραφα **μίας** μεταβλητής θα είναι πάντα ενημερωμένα. Αν τροποποιούνται **δύο** άσχετες μεταξύ τους μεταβλητές;
  - Δεν εξασφαλίζει πώς / πότε / με ποια σειρά γίνονται οι ενημερώσεις τους (κάθε μία θα είναι τελικά OK αλλά μέχρι να ενημερωθούν τα αντίγραφα της μίας, μπορεί να ενημερώνονται ανεξάρτητα και ταυτόχρονα της άλλης).

Σειρά προγράμματος

- Θέλουμε το “read” από μία θέση μνήμης να επιστρέφει πάντα την πιο πρόσφατη τιμή που έγινε “write” εκεί, όχι μία παλιότερη τιμή.
  - Δεν αρκεί μόνο αυτό όμως! Πρέπει οι προσπελάσεις σε **διαφορετικές** θέσεις να ακολουθούν μία γενικότερη σειρά.
- Το ***μοντέλο συνέπεια μνήμης (memory consistency model)*** καθορίζει τη σειρά με την οποία οι προσπελάσεις φαίνονται στους επεξεργαστές
- Η «λογική» λειτουργία εξασφαλίζεται από την «ακολουθιακή συνέπεια» (sequential consistency)

Ακολουθιακή συνέπεια

- Κάθε CPU ολοκληρώνει τις αναφορές της στη μνήμη με τη *σειρά προγράμματός* της
- Όλες οι αναφορές στη μνήμη γίνονται *αδιαίρετα*
  - ατομικότητα
  - σειριοποίηση



SC

- Ικανές (όχι αναγκαίες) συνθήκες για SC:
  1. κάθε νήμα ξεκινάει προσπελάσεις μνήμης με τη σειρά του προγράμματος
  2. όταν ένα νήμα ξεκινάει μια εγγραφή, το νήμα πρέπει να περιμένει να ολοκληρωθεί η εγγραφή πριν ξεκινήσει την επόμενη προσπέλαση
  3. όταν ένα νήμα ξεκινήσει μια ανάγνωση, το νήμα πρέπει να περιμένει να ολοκληρωθεί η ανάγνωση και η εγγραφή την τιμή της οποίας η ανάγνωση φέρνει, πριν ξεκινήσει την επόμενη προσπέλαση
- Το πιο λογικό μοντέλο, δουλεύουν όλοι οι αλγόριθμοι συγχρονισμού χαμηλού επιπέδου, π.χ. αλγόριθμος Dekker:

Initially `A = B = 0`

Process P1	Process P2

`A = 1;`
`if (B == 0)`
`  <critical section>`
	`B = 1;`
	`if (A == 0)`
	`  <critical section>`

SC

- Για να εγγυηθεί την ακολουθιακή συνέπεια, ένα παράλληλο σύστημα δεν μπορεί να χρησιμοποιεί μια σειρά από αρχιτεκτονικές τεχνικές αύξησης απόδοσης.
- Υποχρεωτικά περιορισμοί στη σχεδίαση CPU και compilers:
  - Most ILP techniques (e.g. pipelining with overlapping memory operations), out-of-order execution, speculative execution etc.
  - Register allocation, subexpression elimination, loop transformations

  - BE CAREFUL: avoid register variables for sensitive data
  - USE: **volatile** for sensitive variables

- Λόγω των παραπάνω, υλοποιείται πολύ σπάνια.
  - Παράδειγμα: SGI Origin 2000 (helios.cc.uoi.gr) based on MIPS R10000 processors
    - CPU overlaps memory operations (i.e. NON-ATOMIC) but COMPLETES them in program order
    - Memory subsystem guarantees atomicity. Thus machine has SC.

Τι χάνουμε από την SC

- Υποθέστε:
  - Πρωτόκολλο καταλόγων με 5 CPUs να έχουν αντίγραφο μίας μεταβλητής, και η CPU 1 θέλει να την τροποποιήσει (write)
- Για ατομικότητα της εγγραφής:
  - Η CPU 1 ζητά exclusiveness (20 cycles)
  - Η μνήμη στέλνει στις υπόλοιπες invalidations (10 cycle κάθε μία)
  - Cache invalidation + acknowledgement στη μνήμη (50 cycles) για κάθε μία από τις 4 caches
  - Η μνήμη παραχωρεί exclusiveness στη CPU 1 (20 cycles)

Συνολικός απαιτούμενος χρόνος:

$20 + 4 \times 10 + 50 + 20 = \underline{130}$ *cycles*.

- ΕΡΩΤΗΣΗ: Γιατί όχι μόνο <u>20</u> ?
- ΑΠΑΝΤΗΣΗ: Κανένα πρόβλημα! Ξεχνάμε όμως της ατομικότητα και την ακολουθιακή συνέπεια.

Χαλαρά μοντέλα συνέπειας

- Ξεχνάμε τη σειρά προγράμματος
  - Επιτρέπονται οι προσπελάσεις εκτός σειράς (εφόσον είναι σε διαφορετικές θέσεις μνήμης)
  - Άρα επιτρέπονται τεχνικές ILP στους επεξεργαστές και optimization στους compilers
- Μερικές φορές θυσιάζεται και η ΑΤΟΜΙΚΟΤΗΤΑ των εγγραφών (π.χ. PC – processor consistency, Intel)
- Τα διάφορα χαλαρά μοντέλα χαρακτηρίζονται από το ποιες προσπελάσεις επιτρέπεται να αναδιαταχτούν. Οι τέσσερις πιθανοί συνδυασμοί είναι:
  - Write(x); Read(y) ;
  - Write(x); Write(y);
  - Read(x); Read(y);
  - Read(x); Write(y);

  ή απλά:
  - W->R, W->W, R->R, R->W

Π.χ. χαλαρώνοντας τη σειρά W->R (σχετικά «ανώδυνο»)

- Επιτρέπεται ένα επόμενο read να ξεκινήσει πριν από προηγούμενο write
  - Sun SPARC (TSO – *total store order*)
  - Intel Pentium Pro, MMX (PC – *processor consistency* – τα write είναι μη-ατομικά)

Initially `A = flag = 0`

Process P1	Process P2

`A = 1;`
`flag = 1;`
	`while (flag == 0)`
	`  ;`
	`print("%d", A);`

Initially `A = B = 0`

Process P1	Process P2	Process P3

`A = 1;`	`while (A == 0)`
	`  ;`
	`B = 1;`
		`while (B == 0)`
		`  ;`
		`printf("%d", A);`

Χαλαρώνοντας όλες τις σειρές

- Επιτρέπεται να αναδιαταχτούν οποιεσδήποτε προσπελάσεις σε διαφορετικές θέσεις μνήμης.
  - Π.χ. **weak order** (sync and non-sync memory accesses) και **release consistency** (acquire, release and normal accesses)

- Πάντα παρέχονται εντολές "*memory barriers*" ή "*fences*" οι οποίες όταν εισάγονται σε ένα σημείο του προγράμματος εξασφαλίζουν ότι:
  - Όλες οι προηγούμενες προσπελάσεις έχουν ολοκληρωθεί πριν ξεκινήσουν οι προσπελάσεις που έπονται.
  - Κανονικά σε γλώσσα μηχανής, ο gcc όμως παρέχει «συνάρτηση» που μπορεί να κληθεί από C: `__sync_synchronize();`

Συμπερασματικά

- Οι προγραμματιστές υποθέτουν συνήθως το «λογικό» μοντέλο της ακολουθιακής συνέπειας
- Η ακολουθιακή συνέπεια απαγορεύει αρκετές βελτιστοποιήσεις σε επεξεργαστές και μεταφραστές
- Τα χαλαρότερα μοντέλα έχουν δυνατότητα αυξημένων επιδόσεων (όχι εξωφρενικά καλύτερες όμως…) και είναι σχεδόν πάντα αυτά που χρησιμοποιούνται
- Όμως, *σχεδόν καμία εφαρμογή δε χρειάζεται να απασχολείται με το μοντέλο συνέπειας που υποστηρίζει το hardware*
  - Εκτός αν η εφαρμογή χρησιμοποιεί χαμηλού επιπέδου συγχρονισμό μεταξύ νημάτων
  - Π.χ. προγραμματισμός συστήματος (λειτουργικά συστήματα, βιβλιοθήκες, μεταφραστές κλπ)

Βιβλιογραφία (Πλήρες και ελεύθερο βιβλίο σε PDF)

- Vijay Nagarajan, Daniel J. Sorin, Mark D. Hill, and David A. Wood, *A Primer on Memory Consistency and Cache Coherence, 2nd Edition*, Morgan & Claypool, February 2020
- https://www.morganclaypool.com/doi/10.2200/S00962ED2V01Y201910CAC049