ΚΕΦΑΛΑΙΟ 2
Πολυεπεξεργαστές κοινόχρηστης μνήμης & multi-cores

Shared-memory multiprocessors & multicores

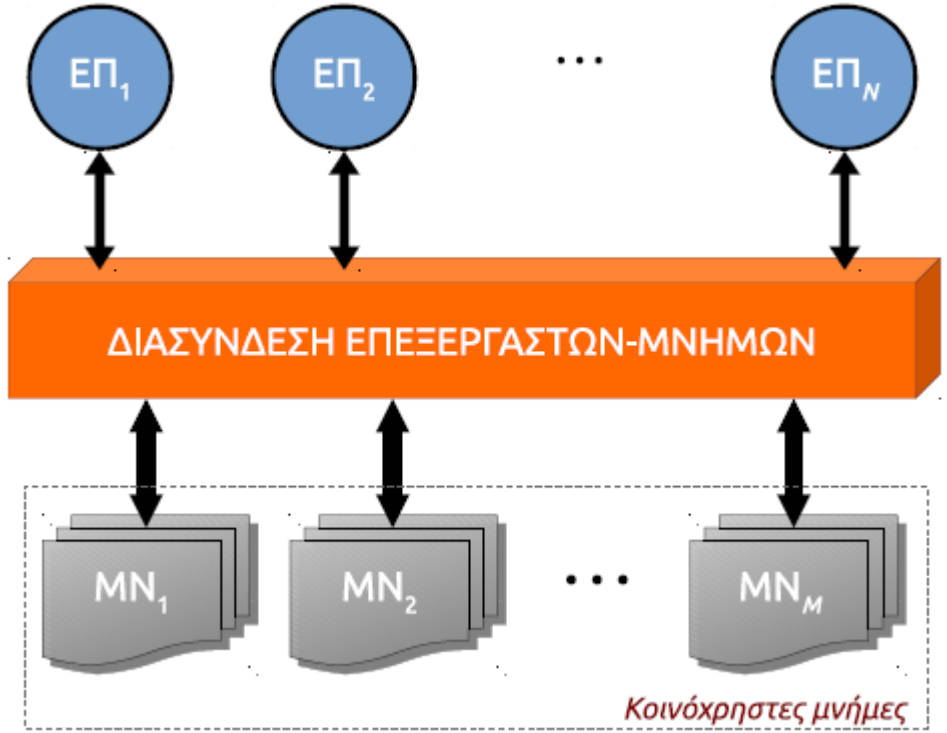
ΜΥΕ023

Παράλληλα Συστήματα & Προγραμματισμός

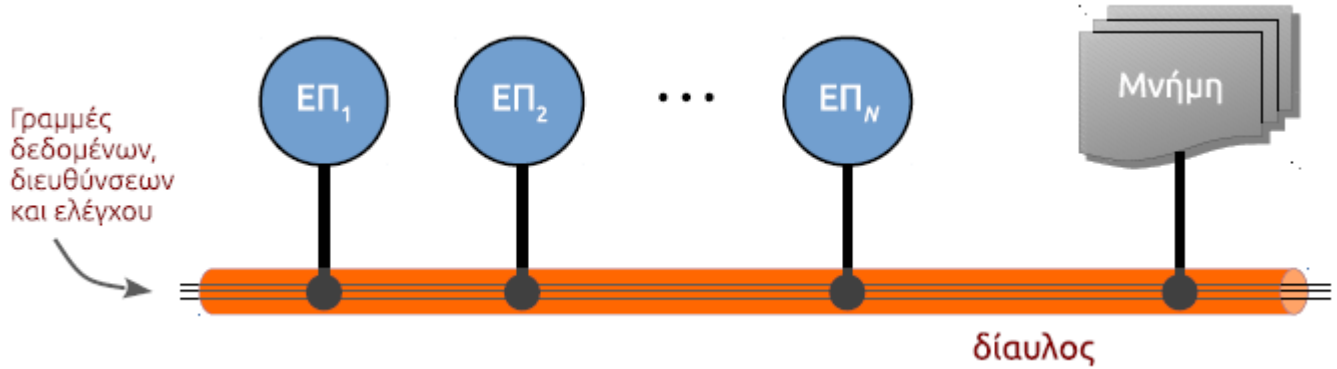
Πολυεπεξεργαστές κοινόχρηστης μνήμης

- Ανεξάρτητοι επεξεργαστές & κοινόχρηστες μνήμες
  - Σα πολυπύρηνα συστήματα είναι πολλοί πυρήνες που μοιράζονται την ίδια κύρια μνήμη
- Διασύνδεση επεξεργαστών – μνημών:
  - δίαυλος
  - δίκτυο διακοπτών
  - Στα πολυπύρηνα υπάρχει ιεραρχία από διαμοιραζόμενες μνήμες (caches)
- Κρυφές μνήμες (caches) και συνοχή
  - Πρωτόκολλα παρακολούθησης
  - Πρωτόκολλα καταλόγων

Βασική οργάνωση



Διασύνδεση με δίκτυο κοινού μέσου, system bus



- System / backplane bus
  - Ο δίαυλος του συστήματος επάνω στις πλακέτες
  - Το backplane bus συνήθως με εξωτερική καλωδίωση
  - Υψηλές ταχύτητες
  - Πυκνή καλωδίωση (50 – 300 γραμμές και παραπάνω)
    - Γραμμές δεδομένων
    - Γραμμές διευθύνσεων
    - Γραμμές ελέγχου

Δίαυλος

- Οικονομική λύση, επιτυχημένη εμπορικά
- SMP (symmetric multiprocessors)
- Δεν κλιμακώνονται εύκολα σε μεγάλο αριθμό επεξεργαστών
  - Λίγοι επεξεργαστές (2-10 συνήθως)
  - Ο δίαυλος γίνεται το bottleneck
- Καλωδίωση:
  - Λίγα καλώδια, μικρή ταχύτητα
  - Πολλά καλώδια, δύσκολη κατασκευή

- Όχι ταυτόχρονη χρήση του μέσου (άρα όχι παράλληλες επικοινωνίες)

- *Χρησιμοποιείται μόνο σε μικρά παράλληλα συστήματα*

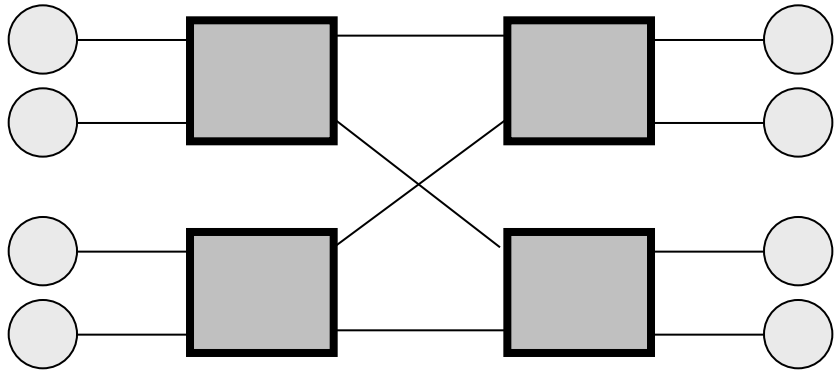
MYE023
MYE023
MYE023

Διακοπτικά δίκτυα

Switching networks

Δίκτυα διακοπτών (switch-based networks)

- Οι επεξεργαστές / μνήμες αποτελούν εισόδους / εξόδους του δικτύου
- Το δίκτυο αποτελείται από διακόπτες συνδεδεμένους μεταξύ τους.
  - Οι διακόπτες ΔΕΝ δημιουργούν κίνηση, απλά την προωθούν
  - Δεν υπάρχουν «γειτονικοί» επεξεργαστές/μνήμες. Για να επικοινωνήσουν ένας επεξεργαστής με μία μνήμη πρέπει να γίνει σύνδεση μέσω διακοπτών.

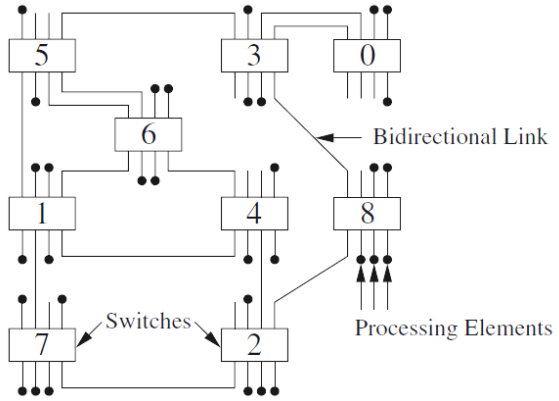


Δίκτυα διακοπτών

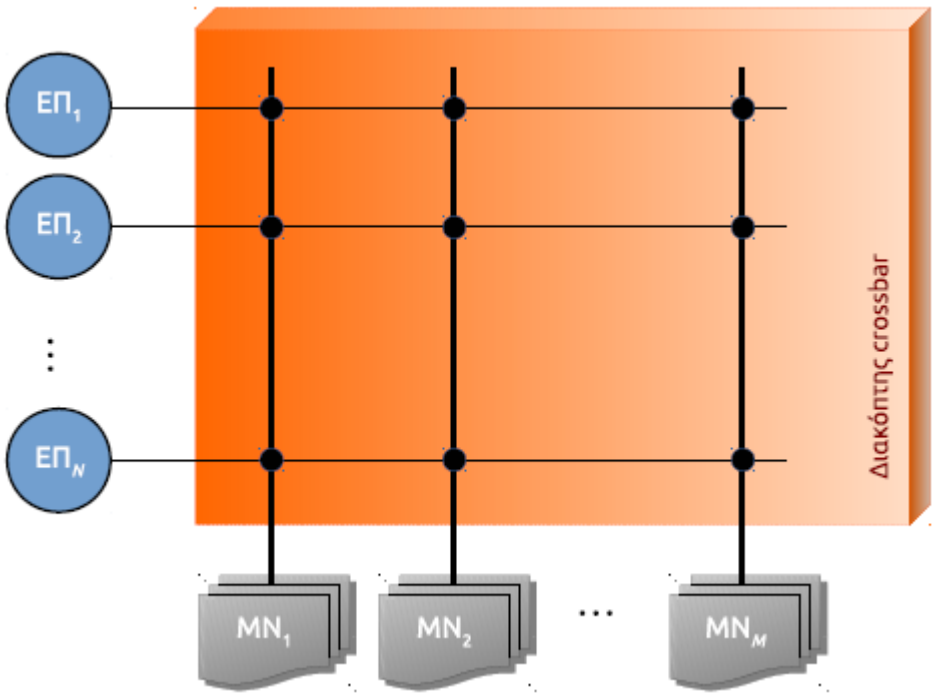
- Πρωτοχρησιμοποιήθηκαν στην τηλεφωνία
- Χρησιμοποιούνται εσωτερικά σε high-performance ethernet / ATM / optical / κλπ. switches (δηλαδή το switch φτιάχνεται από διακοπτικό δίκτυο(!))
- Χρησιμοποιούνται σε παράλληλα συστήματα για διασύνδεση επεξεργαστών μεταξύ τους (κατανεμημένη μνήμη) ή για διασύνδεση επεξεργαστών με μνήμες (κοινή μνήμη)
  - Συνήθως αριστερά οι επεξεργαστές («είσοδοι» του δικτύου) και δεξιά οι μνήμες ή πάλι οι επεξεργαστές («έξοδοι» του δικτύου)

Διάφορα είδη

- 1 μεγάλος διακόπτης (crossbar switch)
  - Μία κατηγορία από μόνος του
- Πολλοί μικρότεροι, διασυνδεδεμένοι μεταξύ τους (multistage networks)
  - Δομημένα (υπάρχει δομημένος τρόπος διασύνδεσης των διακοπτών)
  - Αδόμητα (οι διακόπτες συνδέονται με ακανόνιστο τρόπο παράγοντας ένα εντελώς ασύμμετρο δίκτυο)
    - Π.χ. Myrinet switches μπορούν να συνδεθούν με οποιοδήποτε τρόπο μεταξύ τους.

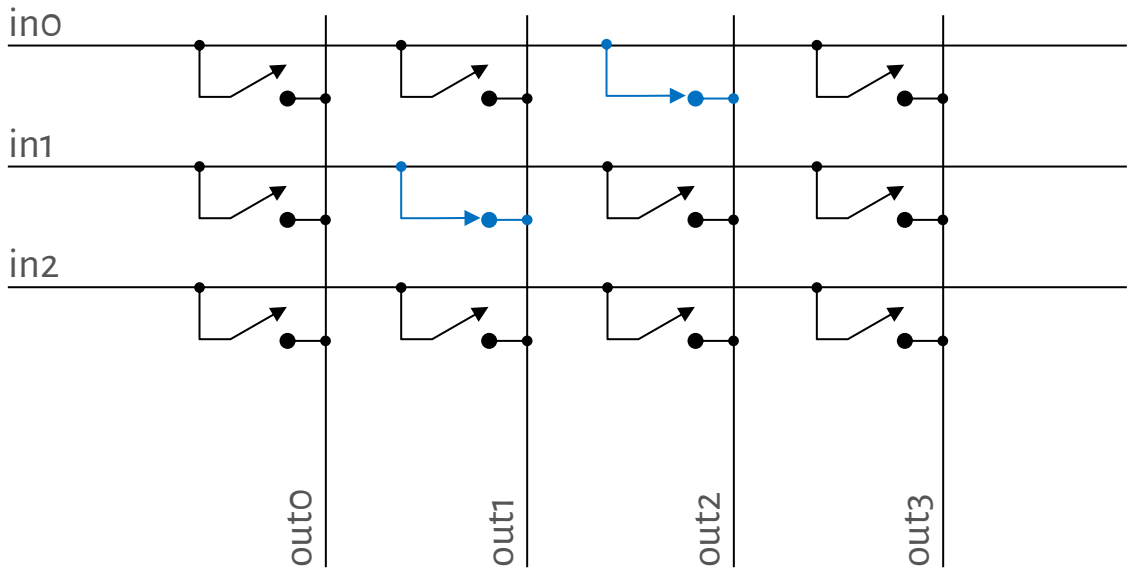


Διασταυρωτικός διακόπτης (crossbar)



- N είσοδοι
- M έξοδοι
- NxM (N^2) σημεία διασταύρωσης

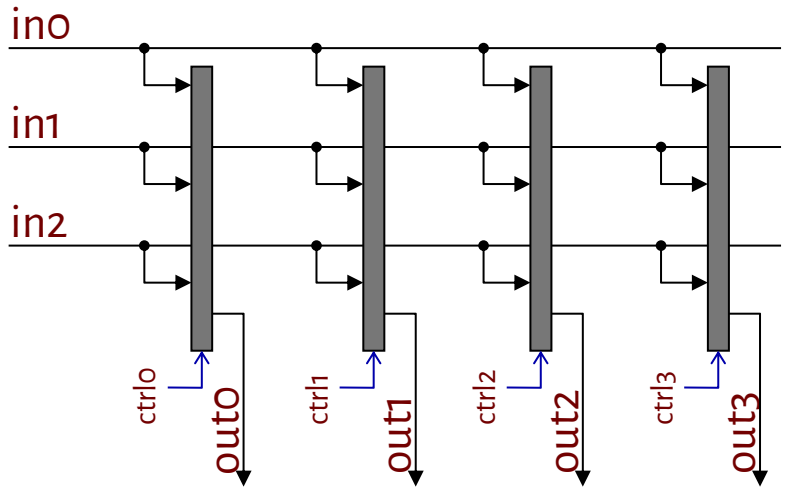
Λειτουργία



- Οποιαδήποτε είσοδος (επεξεργαστής) επικοινωνεί με οποιαδήποτε έξοδο (μνήμη) κλείνοντας το κατάλληλο διακοπτικό στοιχείο.
- Αν N = M, μπορούν ταυτόχρονα όλες οι είσοδοι να είναι συνδεδεμένες με όλες τις εξόδους (κάθε είσοδος με μία διαφορετική έξοδο)
  - Όλες οι δυνατές μεταθέσεις

Θέματα

- Πώς υλοποιείται;



- Για τετράγωνους (N = M), το κόστος είναι τάξης N^2
  - N^2 διακοπτικά σημεία στο πλέγμα
  - N multiplexers, κόστους Θ(N) ο καθένας
- Αν χρησιμοποιείται ως system interconnect (αντί για δίαυλο), τότε ΚΑΘΕ γραμμή εισόδου / εξόδου θα αποτελείται από *εκατοντάδες σήματα / καλώδια* (σήματα διευθύνσεων, δεδομένων, ελέγχου). Αδύνατη η κατασκευή για μεγάλο N.
- Μόνο για μικρό # εισόδων / εξόδων (ή μέτριο με λίγα σήματα ανά γραμμή)

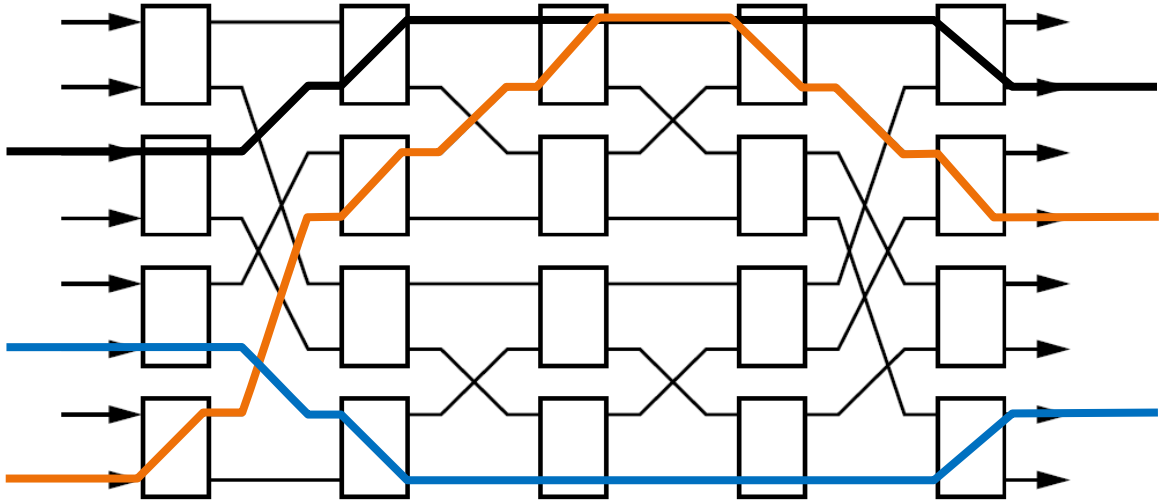
MYE023
MYE023
MYE023

Πολυεπίπεδα διακοπτικά δίκτυα

Multistage switching networks

Πολυεπίπεδα δίκτυα (multistage interconnection networks - MINS)

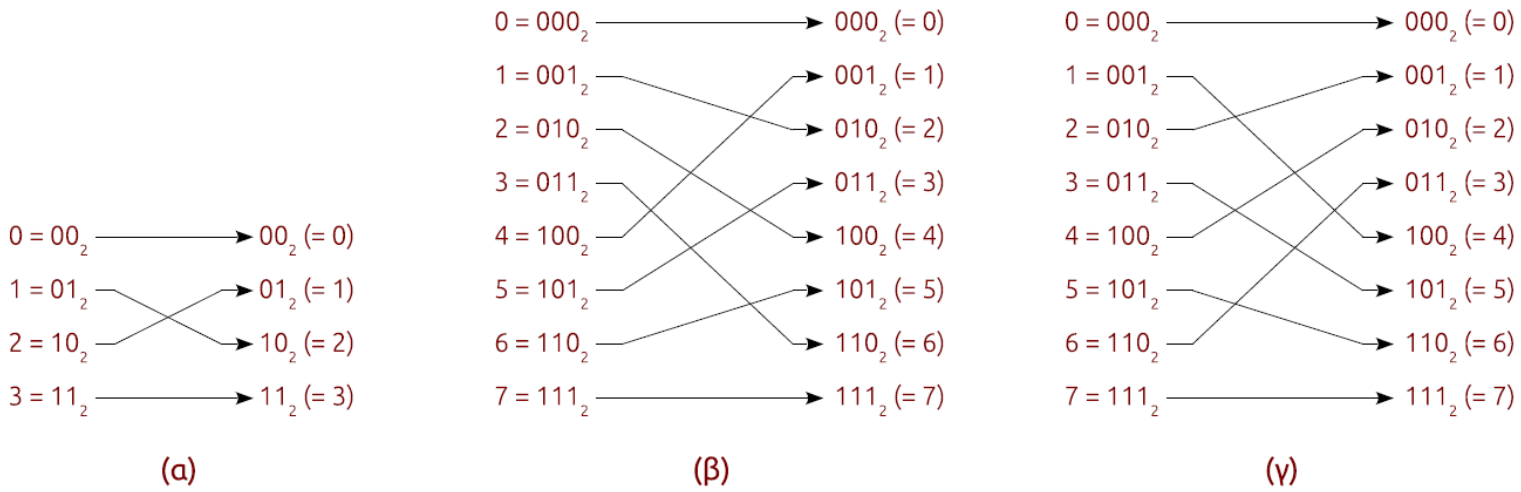
- Πολλαπλά επίπεδα ή *στάδια* από μικρούς διακόπτες crossbar.
- Τα στάδια συνδέονται μεταξύ τους
- Στο πρώτο στάδιο βρίσκονται οι είσοδοι του δικτύου, στο τελευταίο οι έξοδοι



- Μεγαλύτερες καθυστερήσεις για περισσότερα επίπεδα
- Μικρότερο κόστος από τον crossbar
- Διαφέρουν ανάλογα με την διασύνδεση των σταδίων, ***θα δούμε το δίκτυο ΔΕΛΤΑ***

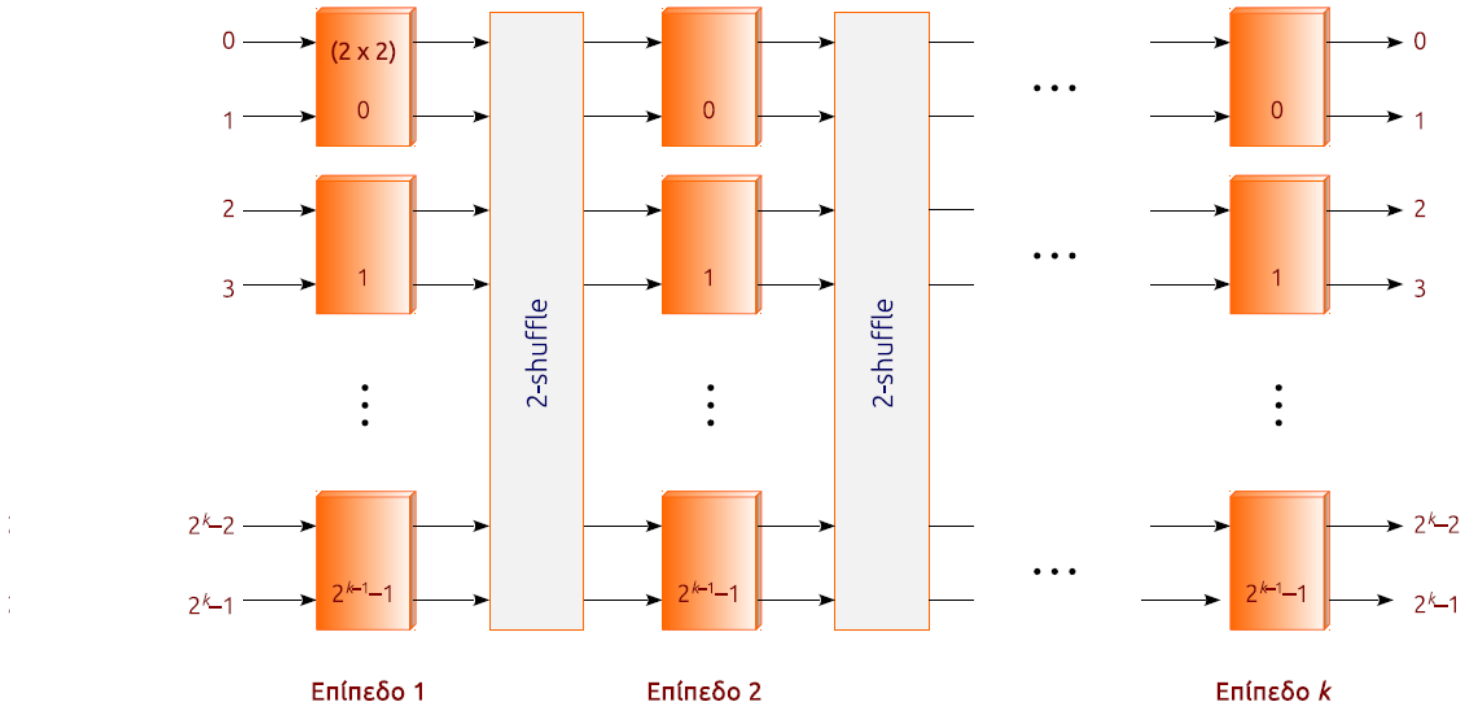
Πράξη shuffle («ανακάτωμα»)

- Πρόκειται για περιστροφή προς τα αριστερά των bits ενός αριθμού
- Έστω x με βάση το 2 (δυαδικός αριθμός):
  - $x = (x_k x_{k-1} \ldots x_1)_2$
- Τότε:
  - Shuffle $S(x) = (x_{k-1} x_{k-2} \ldots x_1, x_k)_2$



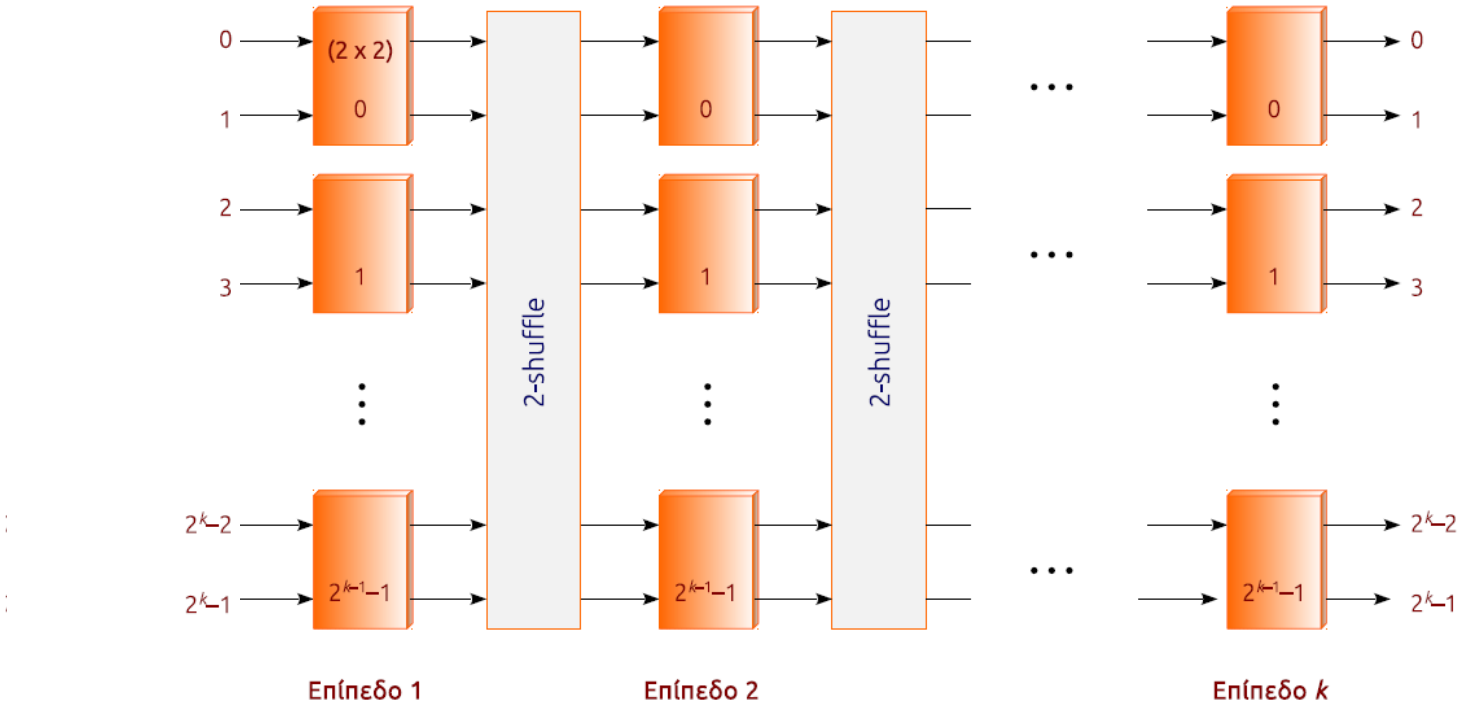
Δυαδικό, συμμετρικό δίκτυο Δέλτα

- Συμμετρικό γιατί #εισόδων = #εξόδων, N = M = 2^k
  - Διακόπτες 2x2
  - k στάδια
  - Shuffle διασύνδεση μεταξύ των σταδίων

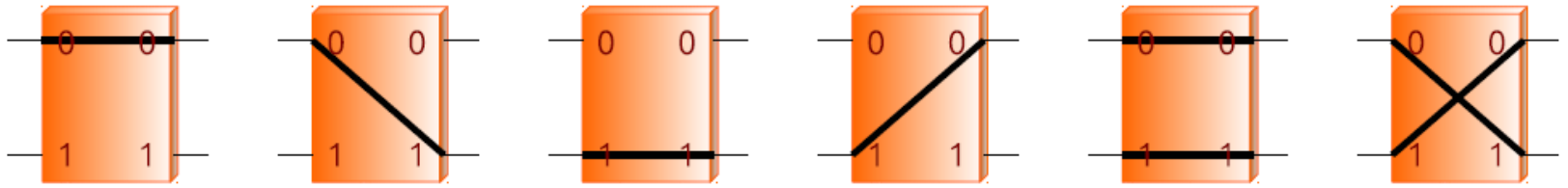


Δυαδικό, συμμετρικό δίκτυο Δέλτα … συνέχεια

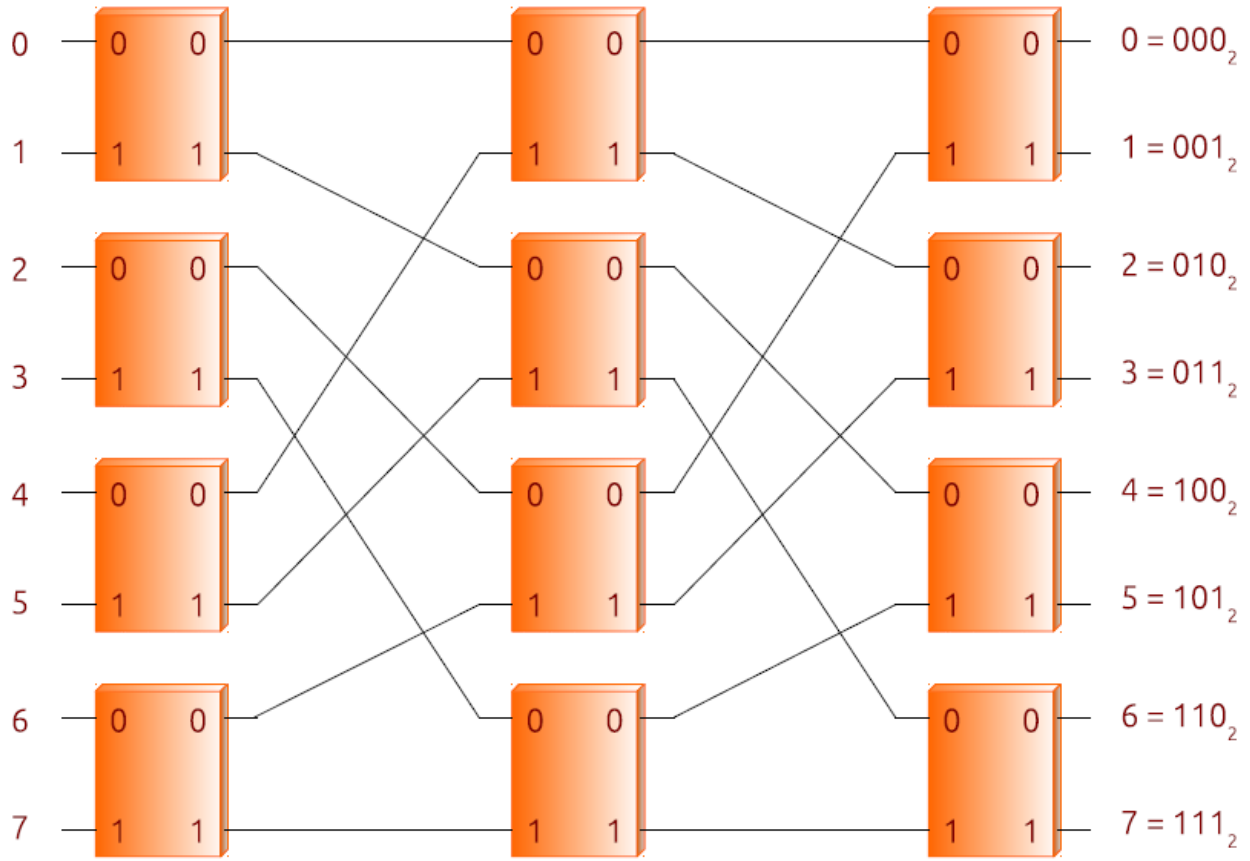
- Συμμετρικό γιατί #εισόδων = #εξόδων, $N = M = 2^k$
- Σε κάθε επίπεδο, N γραμμές, διακόπτες 2x2, άρα:
  - $N/2 = 2^{k-1}$ διακόπτες ανά στάδιο
- k στάδια ($= \log_2 N$), άρα:
  - $(N/2) \log_2 N$ διακόπτες, δηλαδή $\Theta(N \log_2 N)$ κόστος



Διακόπτες crossbar 2x2



Δίκτυο Δέλτα 2^3 x 2^3



Διαδρόμηση στο δίκτυο Δέλτα

- Κατανεμημένος αλγόριθμος

Επεξεργαστής $\mathbf{x = (x_k x_{k-1} \ldots x_1)_2}$ προς μνήμη $\mathbf{y = (y_k, y_{k-1} \ldots y_1)_2}$.

$$out_1 = (x_k x_{k-1} \ldots y_k)_2$$

$$in_2 = (x_{k-1} \ldots x_2 y_k x_k)_2$$

$$out_2 = (x_{k-1} \ldots x_2 y_k y_{k-1})_2$$

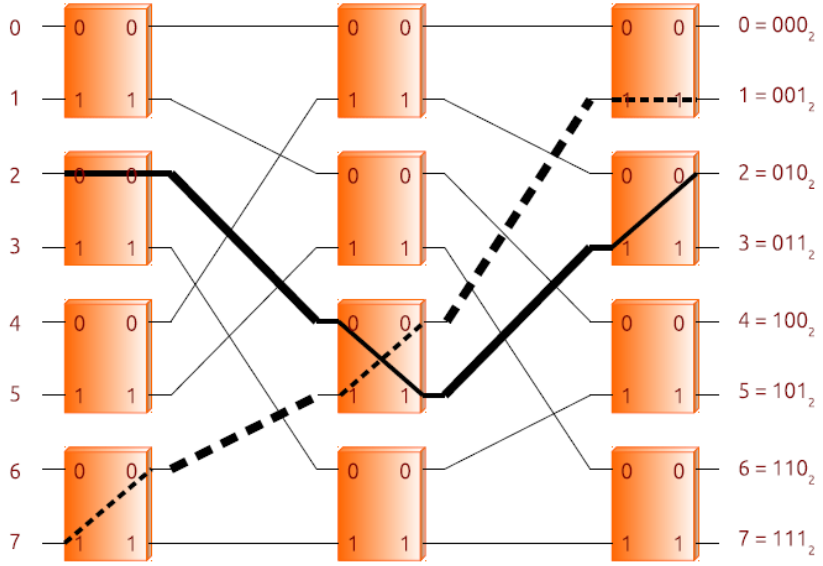
...

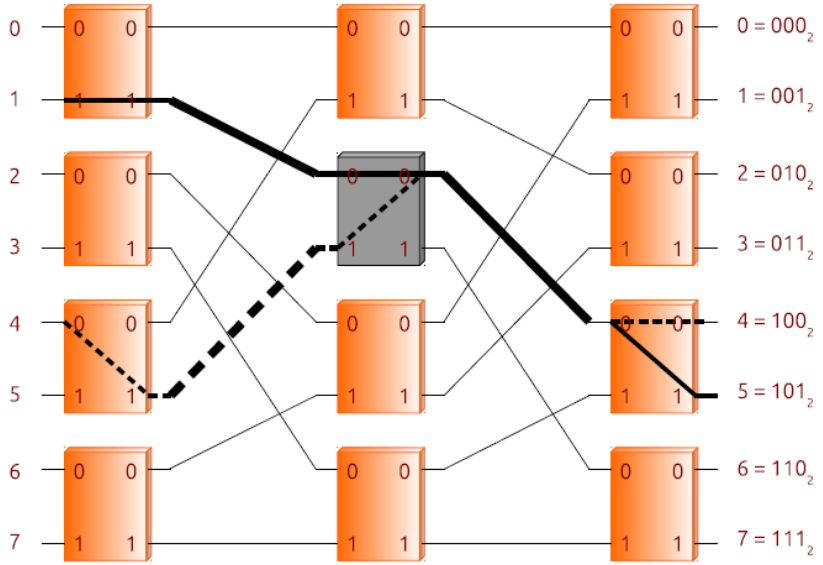
$$in_k = (y_k y_{k-1} \ldots y_2 x_2)_2$$

$$out_k = (y_k y_{k-1} \ldots y_2 y_1)_2$$

Όταν ένας διακόπτης στο επίπεδο i λαμβάνει είσοδο για σύνδεση προς την μνήμη y, την συνδέει με την y_{k-i+1} έξοδό του.

Παραδείγματα διαδρόμησης



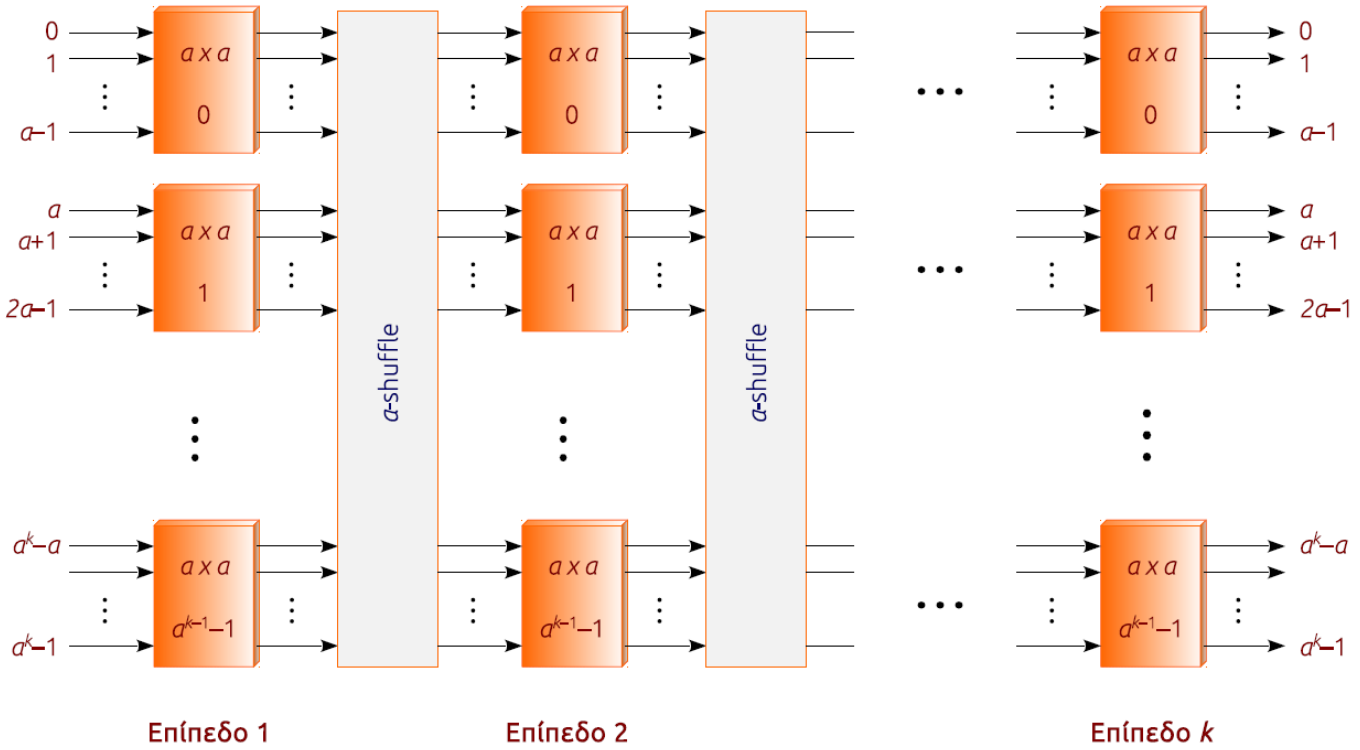


Μερικά άλλα δίκτυα

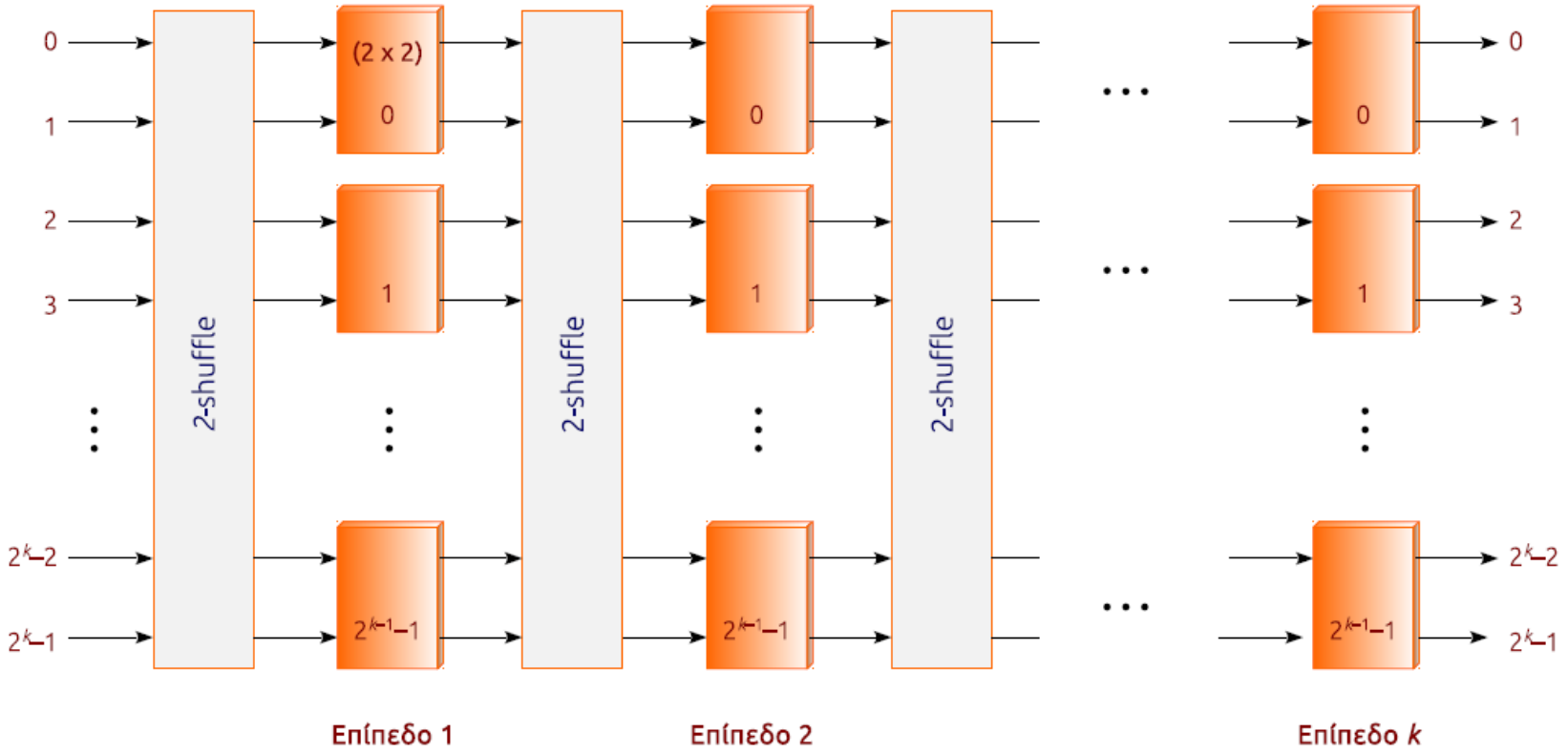
- Γενικό δίκτυο Δέλτα
- Δίκτυο Omega
- Δίκτυο baseline
- Δίκτυο butterfly
- Δίκτυο Benes
- Δίκτυο fat tree
- …

Γενικό, συμμετρικό δίκτυο Δέλτα

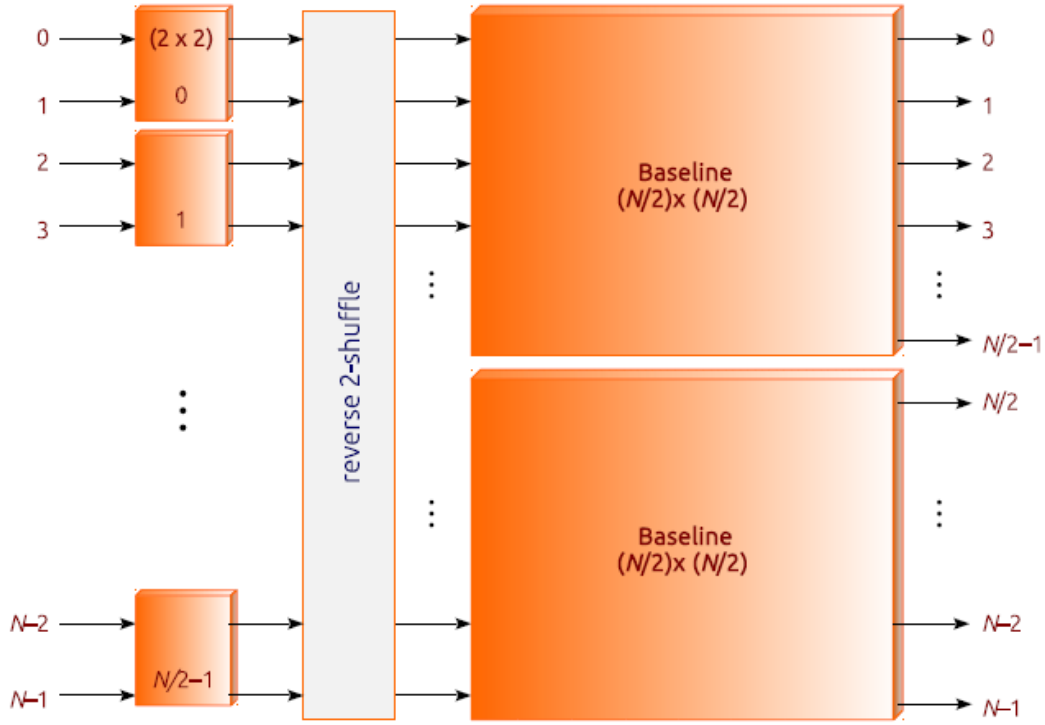
- Συμμετρικό γιατί #εισόδων = #εξόδων, N = M = a^k
  - Διακόπτες axa
  - k στάδια
  - a-shuffle διασύνδεση μεταξύ των σταδίων

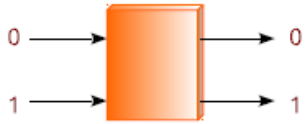


Δίκτυο Ωμέγα

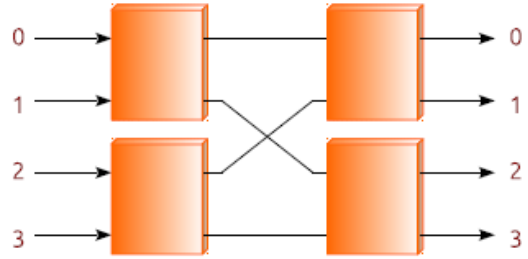


Δίκτυο baseline



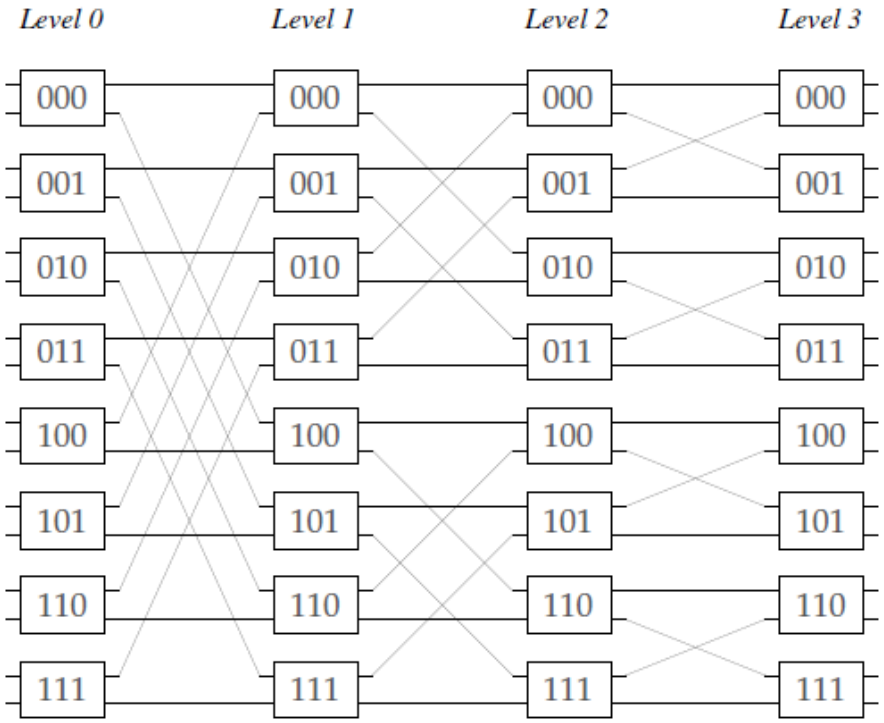


Baseline 2x2



Baseline 4x4

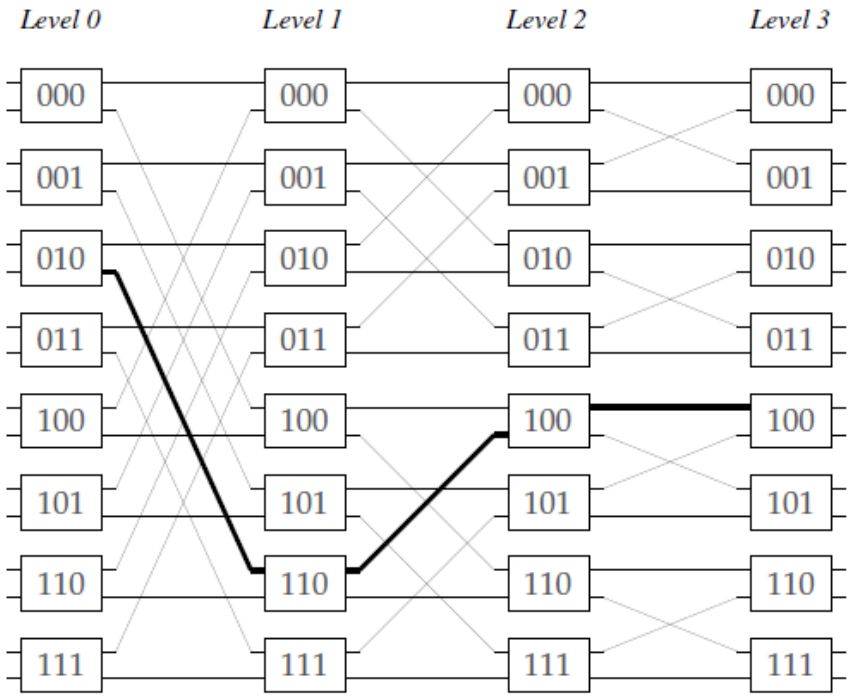
Δίκτυο πεταλούδας



r–dimensional Butterfly: 2^r switches at each level, (r+1) levels

Switch (w, i) at level i is connected to switch (w, i+1) and (w', i+1) where w' differs from w at the i–th most significant bit

Διαδρόμηση στο δίκτυο πεταλούδας



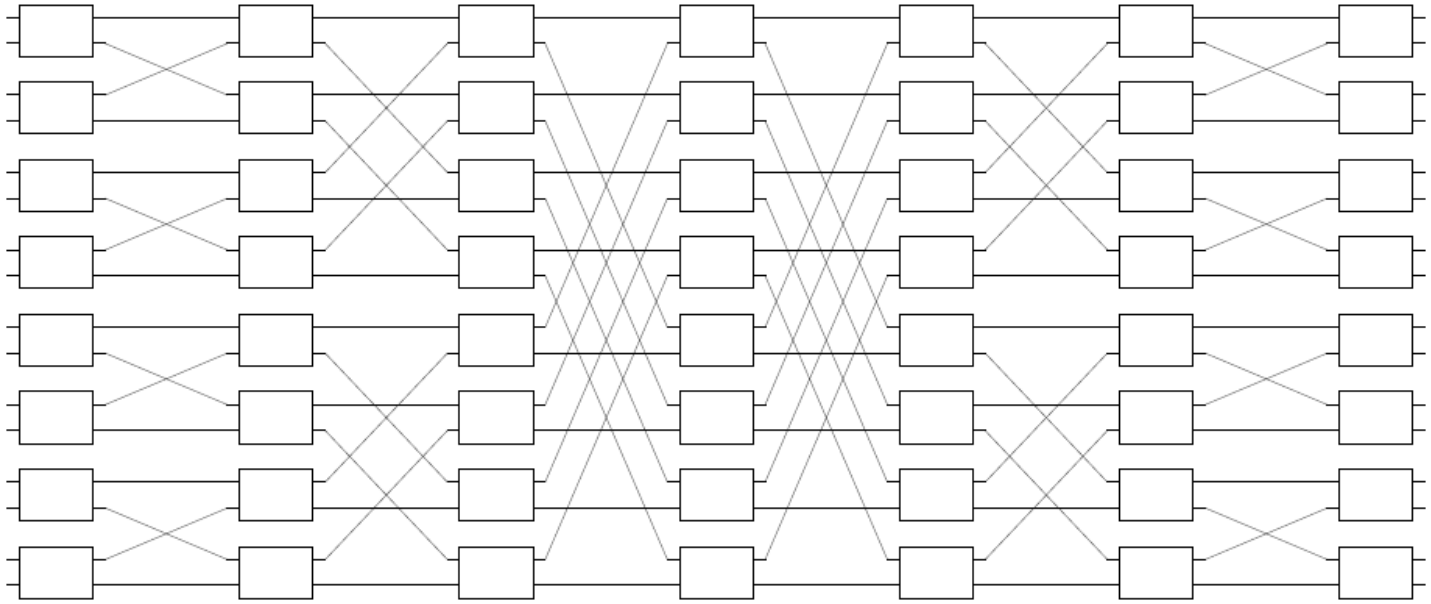
Butterfly Routing: go from switch <s,0> to destination <d,r>

A switch (w, i) at level i chooses:

- *the straight edge if w and d have the same i–th most significant bit*
- *the cross edge otherwise*

Δίκτυο Benes

- Δύο πεταλούδες πλάτη με πλάτη
- Πολλαπλά μονοπάτια



Μεταθέσεις και εμποδιστικότητα

- Υποθέστε συμμετρικό δίκτυο (M = N) και ότι όλοι οι επεξεργαστές θέλουν να συνδεθούν με διαφορετικές μνήμες ο καθένας.
- Θέλουμε καθένας από τους επεξεργαστές { 0, 1, …, N–1 } να συνδεθεί με μία διαφορετική από τις { 0, 1, …, N–1 } μνήμες
- Άρα αντιστοίχιση 1-προς-1 από το σύνολο { 0, 1, …, N–1 } στο σύνολο { 0, 1, …, N–1 } (*μετάθεση*).
- Είδαμε όμως ότι ΔΕΝ γίνεται πάντα να συνυπάρχουν ταυτόχρονες συνδέσεις, επομένως το ερώτημα είναι:

Ποιες μεταθέσεις επιτρέπονται;

Εμποδιστικότητα και είδη δικτύων

- Πόσες μεταθέσεις επιτρέπει το δίκτυο:
  - Μερικές => **ΕΜΠΟΔΙΣΤΙΚΟ (blocking)** δίκτυο
    - Δέλτα, Ωμέγα, baseline, πεταλούδα
  - Όλες => **ΜΗ-ΕΜΠΟΔΙΣΤΙΚΟ (non-blocking ή universal)** δίκτυο
    - Μόνο το Closs (και προφανώς και ο crossbar)
    - Το Benes είναι επαναπρογραμματιζόμενο (rearrangeable)

- Αν και δεν φαίνεται εύκολα, έχει αποδειχθεί ότι τα περισσότερα από τα δίκτυα που είδαμε (Δέλτα, Ωμέγα, baseline, πεταλούδας) είναι

  ισοδύναμα μεταξύ τους

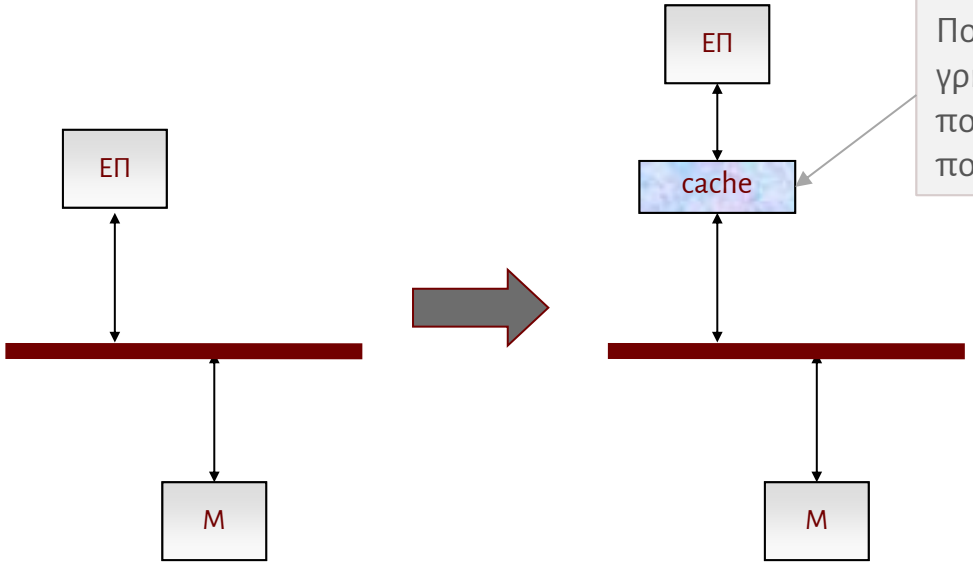
MYE023
MYE023
MYE023

Κρυφές μνήμες και συνοχή

Cache coherency

Σύνοψη ιεραρχίας μνήμης & cache σε 4 διαφάνειες

- 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

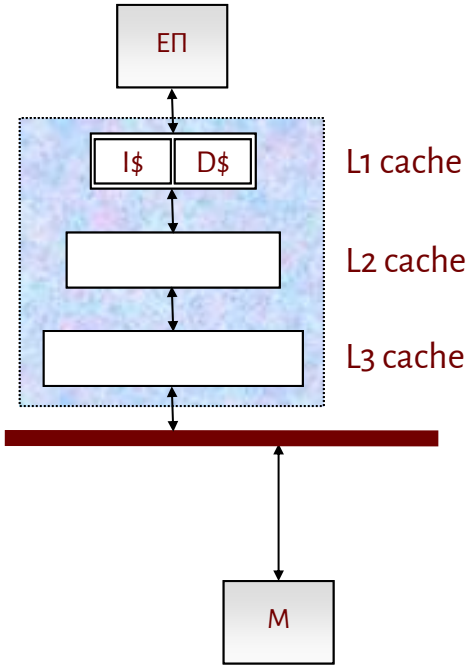
     Πώς / πότε ενημερώνεται η κύρια μνήμη;
     - Μόνο όταν έρθει η ώρα να αντικατασταθεί το τροποποιημένο (*dirty*) δεδομένο στην cache.

   - 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

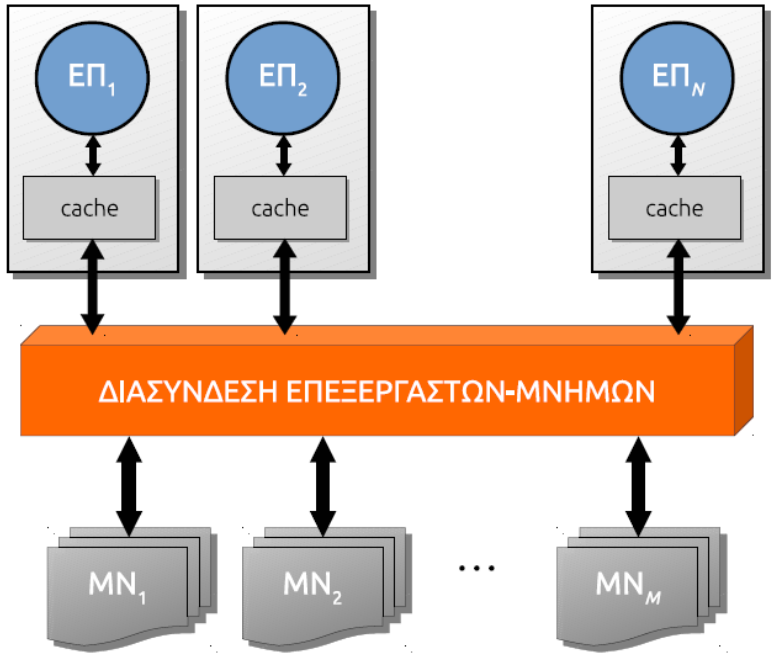
L2 cache ~ 256 KB – 8 MB, 6-15 cycles

L3 cache ~ 4 – 32 MB, 30-50 cycles

~ Gbytes, 100+ cycles

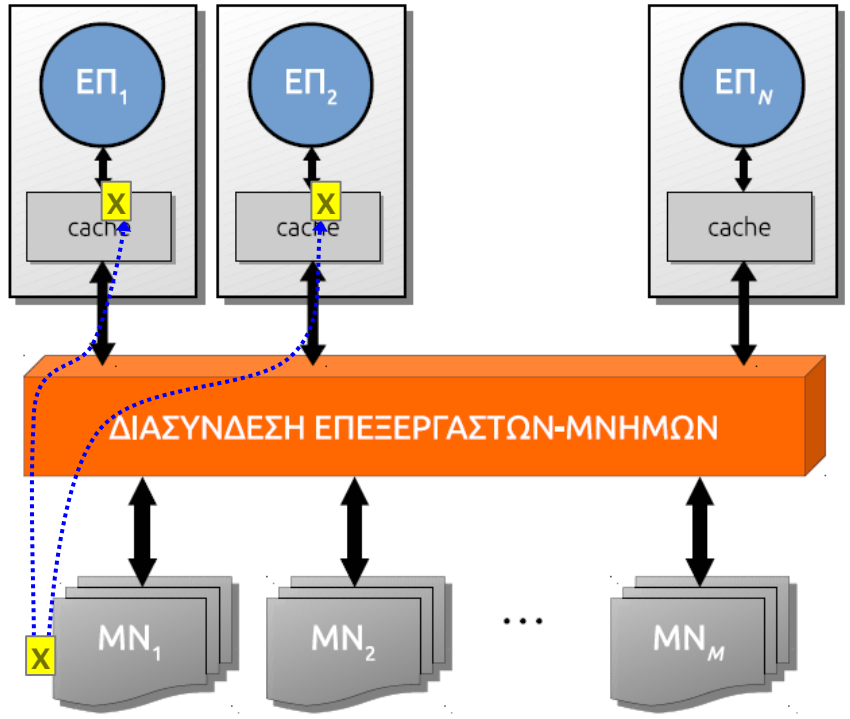
Caches σε παράλληλα συστήματα κοινής μνήμης

- 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

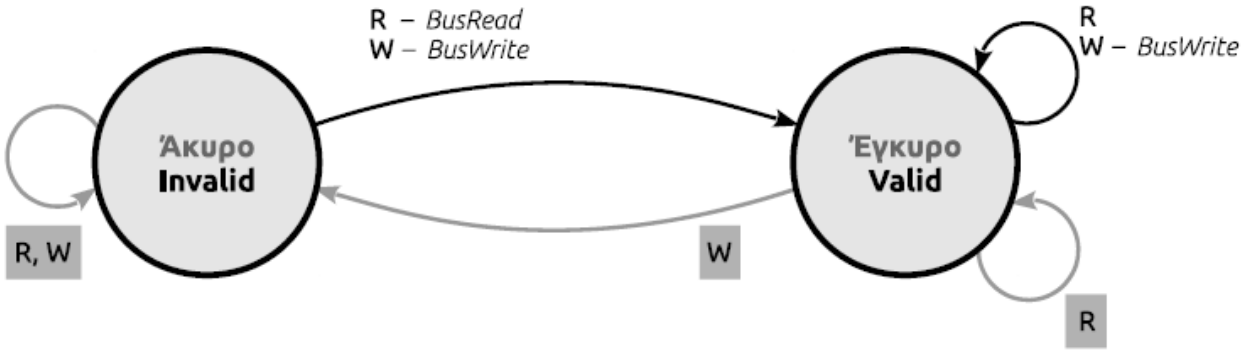
- Το αφήνουμε και το λύνουμε κατά την εκτέλεση, με hardware
  - πρωτόκολλα συνοχής (coherence protocols)
    - πρωτόκολλα παρακολούθησης (snoopy/snooping protocols)
    - πρωτόκολλα καταλόγων (directory-based protocols)

Πρωτόκολλα παρακολούθησης

- Δύο είδη:
  - εγγραφής – ενημέρωσης (write-update)
  - εγγραφής – ακύρωσης (write-invalidate)
- Συνήθως κρυφές μνήμες με πολιτική υστεροεγγραφής (write-back)
- Μάλλον (?) εγγραφής-ακύρωσης
- Μοντέρνες CPU (Pentium, UltraSparc, κλπ):
  - όλες διαθέτουν hardware για πρωτόκολλα παρακολούθησης (multiprocessor-ready)

Πρωτόκολλο εγγραφής-ακύρωσης με διεγγραφή

- Κάθε δεδομένο, σε κάθε cache είναι σε μία από δύο καταστάσεις

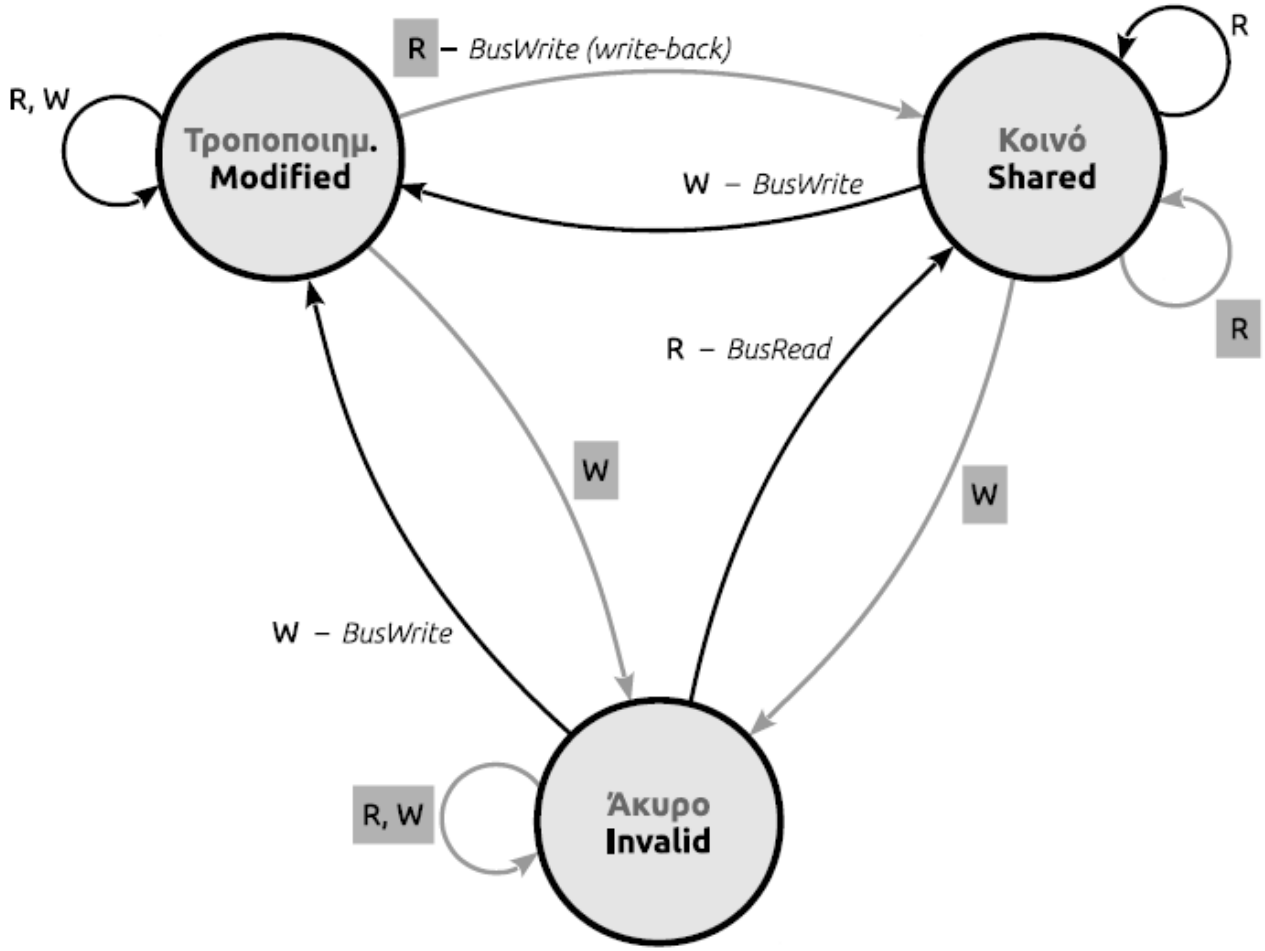


Πρωτόκολλο MSI για υστεροεγγραφή

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)
- Κοινή (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»
- Τροποποιημένη (Modified)
  - Έχουμε το μοναδικό ενήμερο αντίγραφο της γραμμής

Πρωτόκολλο εγγραφής-ακύρωσης με υστεροεγγραφή (MSI)



Άλλα πρωτόκολλα παρακολούθησης

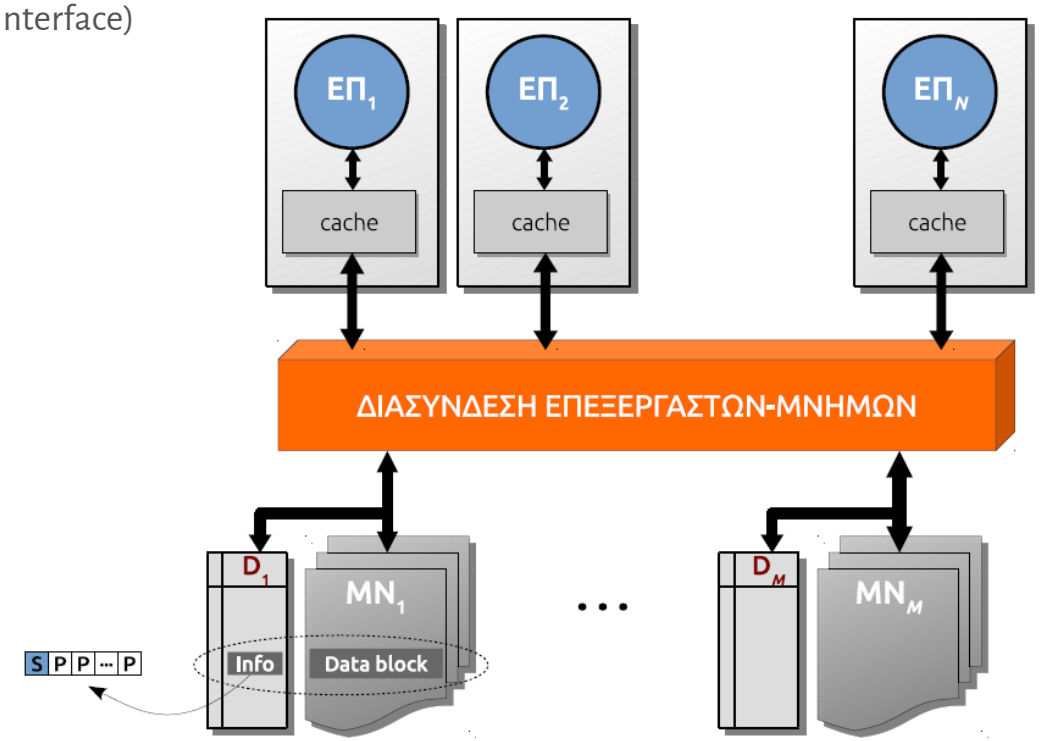
- MESI (Modified/Exclusive/Shared/Invalid)
  - Γνωστό ως πρωτόκολλο Illinois
  - Π.χ. στους Intel Pentium
  - Στην exclusive, η cache έχει το μοναδικό αντίγραφο μη τροποποιημένο (η κύρια μνήμη είναι ενημερωμένη)
    - **Γιατί**;
    - **Απάντηση**: για τη συνηθισμένη περίπτωση που διαβάζω μία *νέα* γραμμή και αμέσως την τροποποιώ (γλιτώνω 1 miss, κερδίζω και στα σειριακά προγράμματα)
- MOESI (Modified/shared-mOdified/Exclusive/Shared/Invalid)
  - Π.χ. στους AMD Athlon, SUN UltraSparc
  - Η O είναι σαν την S μόνο που η κύρια μνήμη δεν έχει ενημερωθεί.

Πρωτόκολλα με καταλόγους

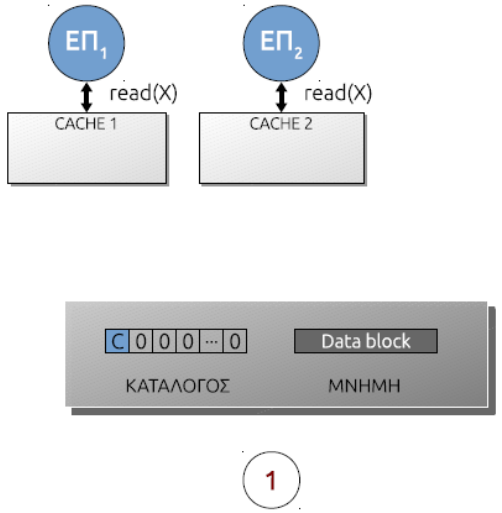
- Δεν υπάρχει κοινό μέσο (π.χ. διακοπτικά δίκτυα)
  - Άρα αδύνατα / ασύμφορα τα πρωτόκολλα παρακολούθησης
- Η μνήμη είναι υπεύθυνη για όλα
  - Κατάλογος όπου καταγράφεται ποιες cache έχουν αντίγραφο των δεδομένων
  - Επικοινωνία για συνοχή *μόνο* με αυτές τις cache
- Κεντρικός κατάλογος (κακό) ή
- Κατανεμημένοι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

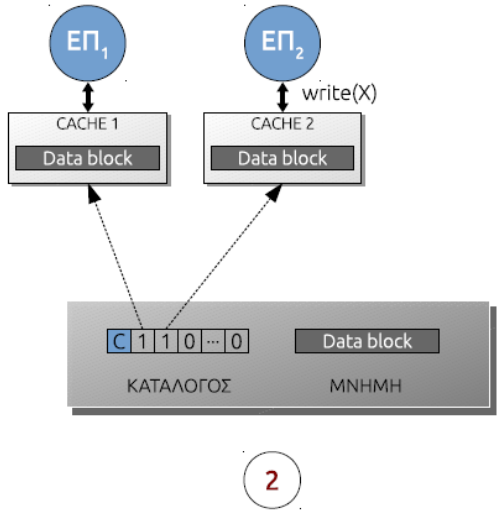
Κατανεμημένοι κατάλογοι

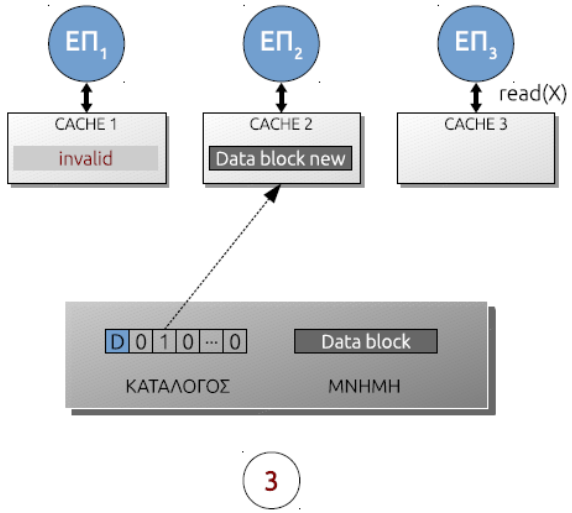
- Πεδία «παρουσίας» στους καταλόγους
- Κατανεμημένοι κατάλογοι:
  - πλήρεις (full-map directories)
  - περιορισμένοι (limmited directories)
  - αλυσιδωτοί (chained directories)
    - IEEE SCI (Scalable Coherent Interface)

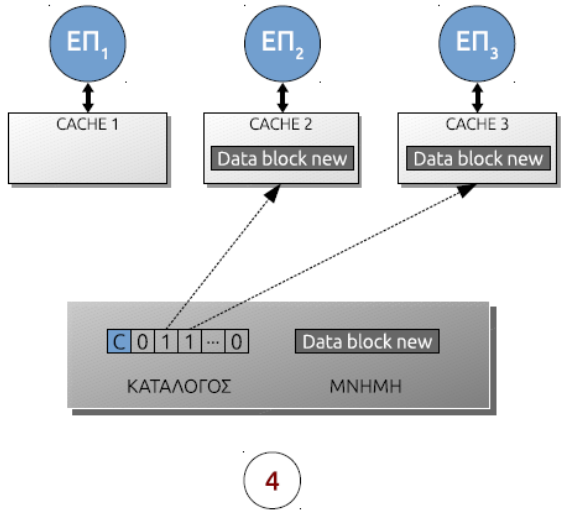


Πλήρεις κατάλογοι







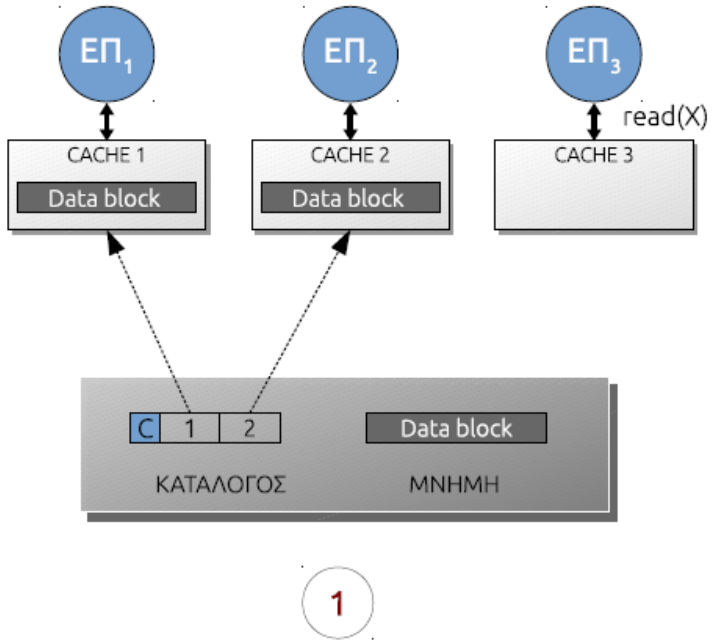


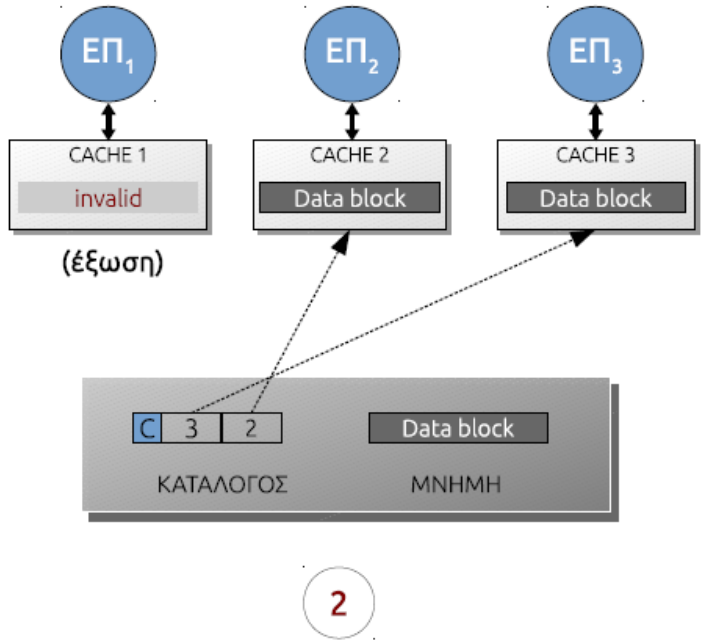
Για συστήματα μεσαίου μεγέθους

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

Για
- 1024 cpus?

Μερικοί κατάλογοι

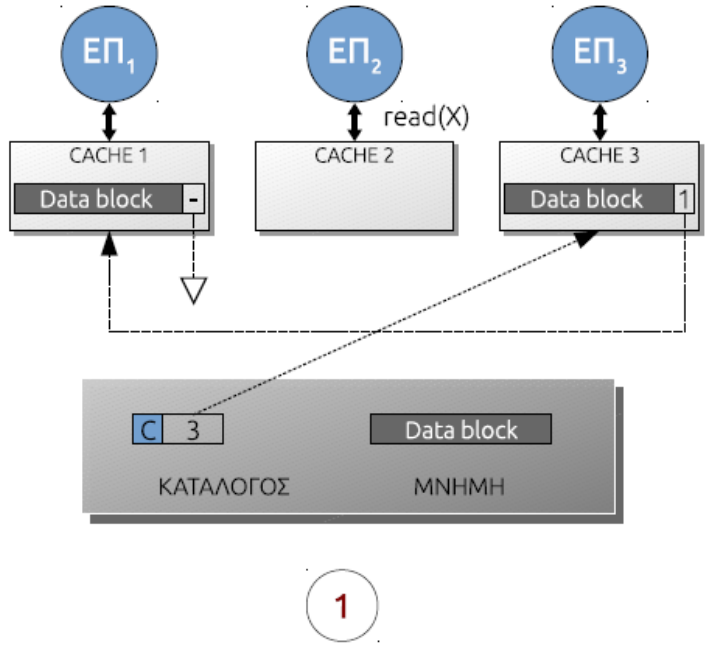


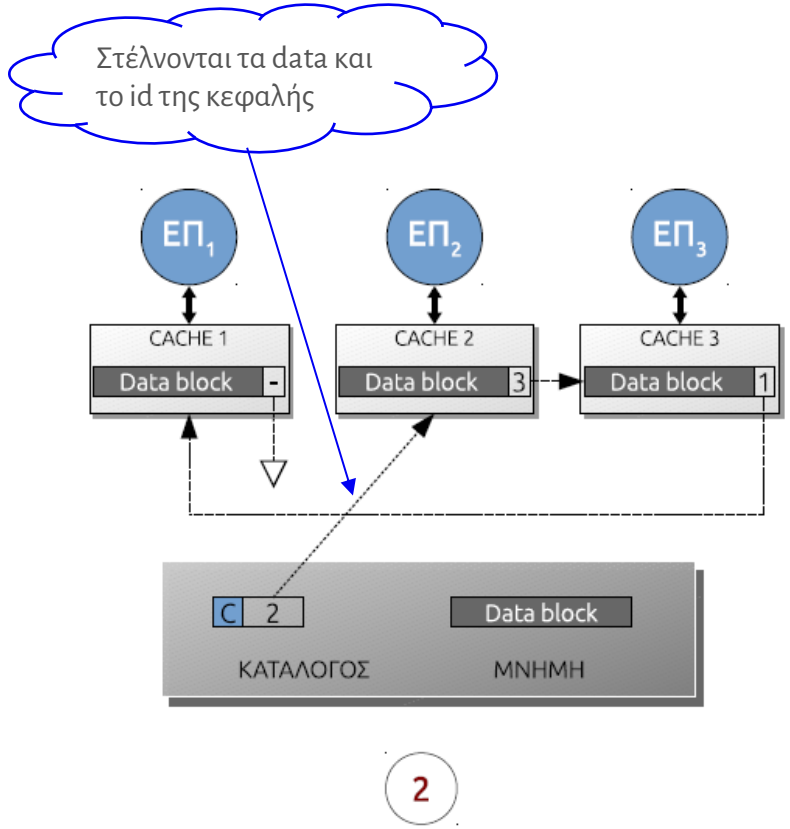


Για
- 1024 cpus
- block size / cache line = 64 bytes
- K = χώρος για 10 αντίγραφα
- ποιο το μέγεθος του directory?

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

Αλυσιδωτοί κατάλογοι (κυρίως για NUMA-- αργότερα)





Write hit στην cache A:

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

Replacement σε μία cache:

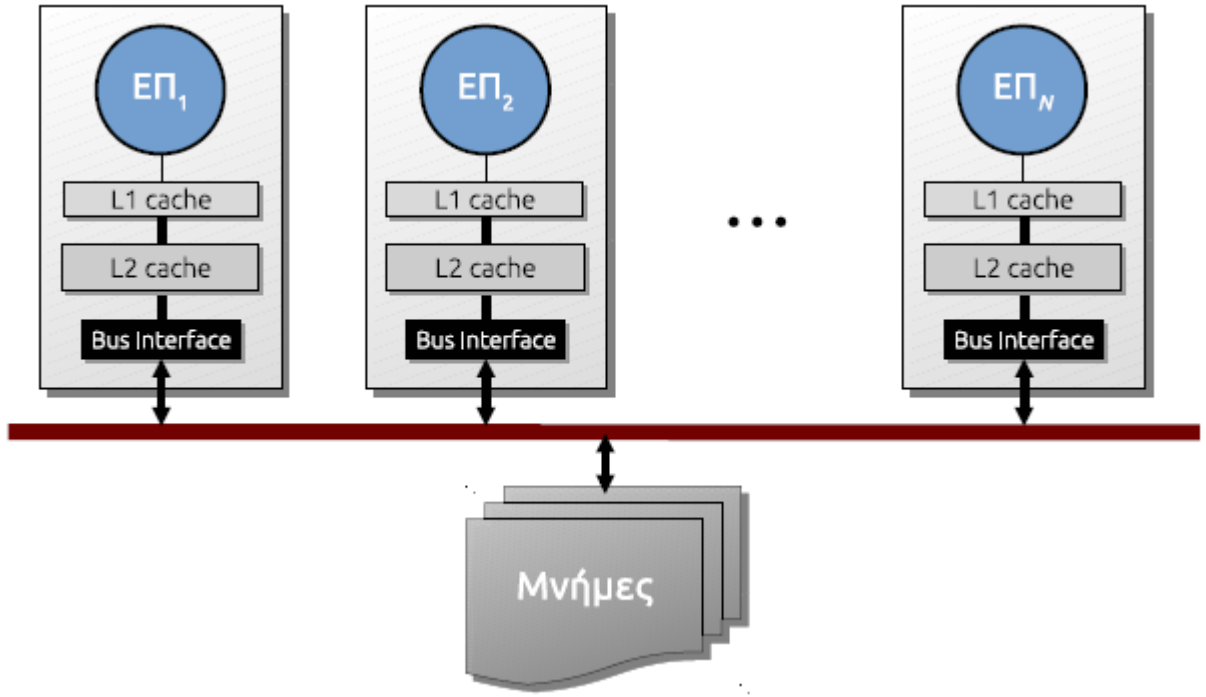
- Η cache στέλνει Invalidate στη μνήμη
- Το Invalidate προωθείται και ακυρώνεται όλη η αλυσίδα.

MYEO23
MYEO23
MYEO23

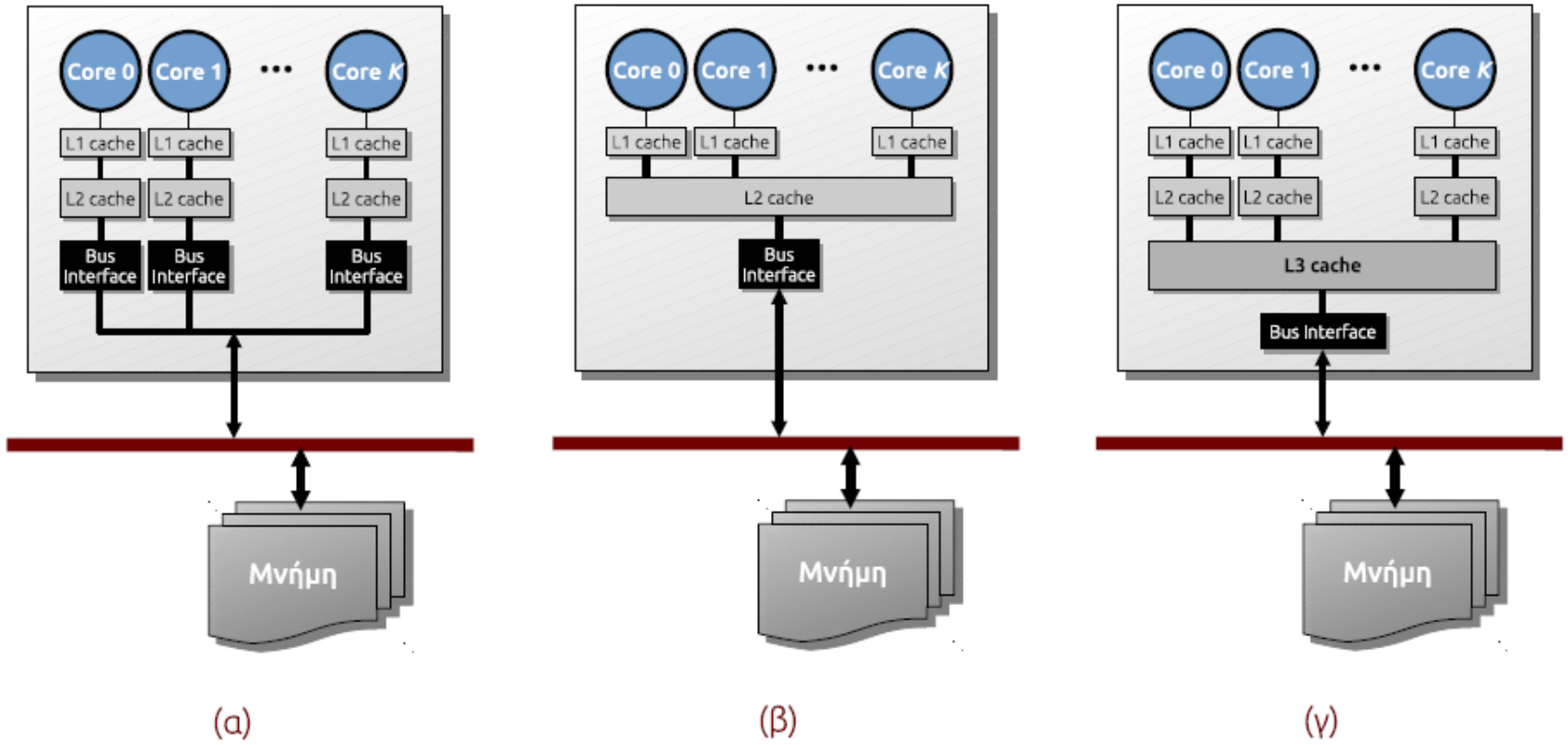
Πολυπύρηνοι επεξεργαστές

Multicores

Πολυεπεξεργαστές

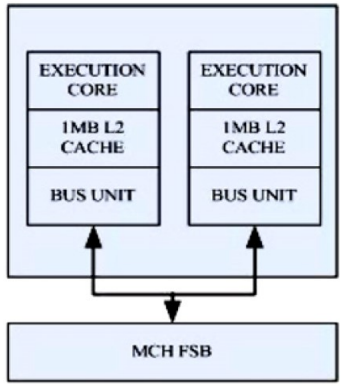


Οργανώσεις multicore

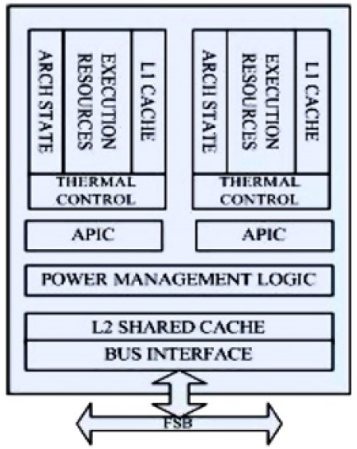


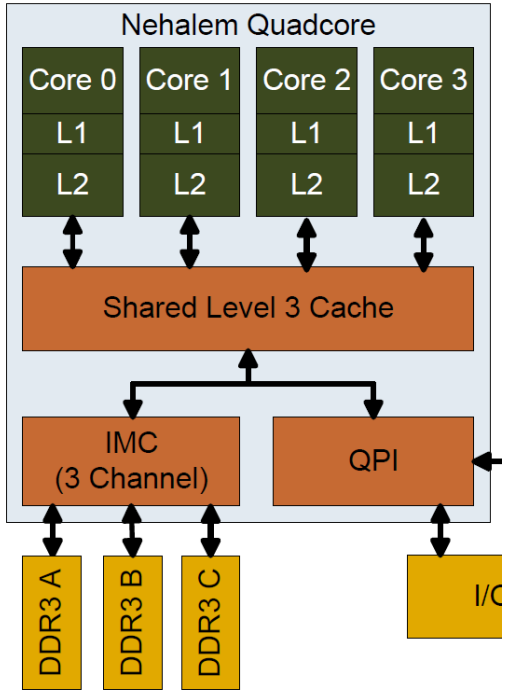
Παραδείγματα

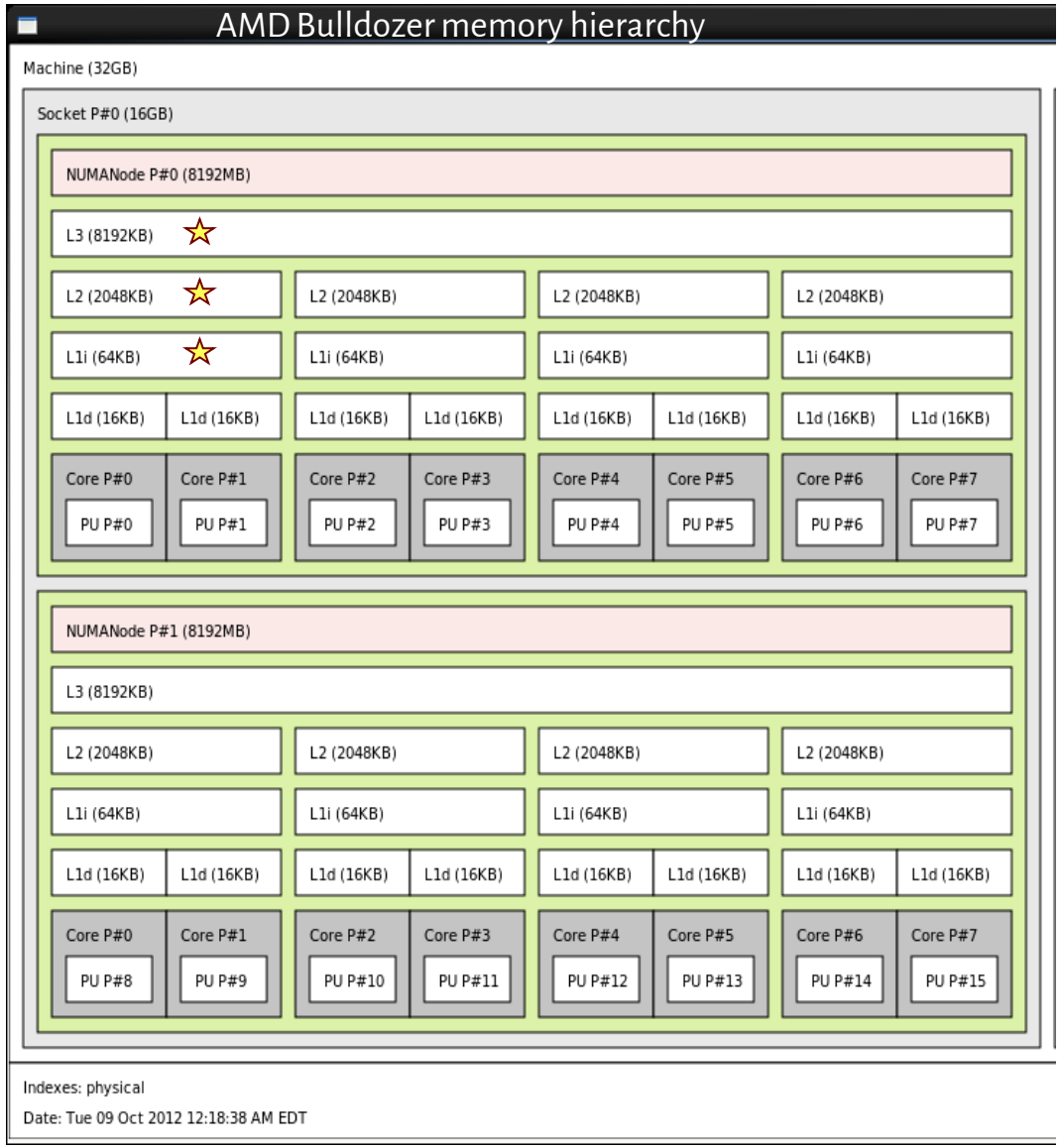
Pentium D



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

MYEO23
MYEO23
MYEO23

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) */`

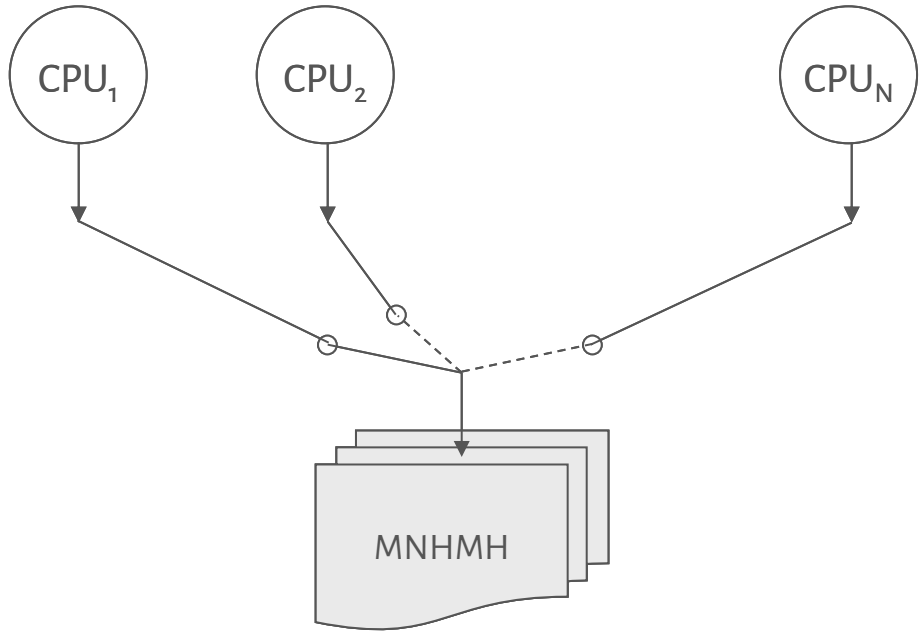
- Εκτύπωση: **10** (!? Call service?)
  - Αποδεκτές εκτυπώσεις: 00, 11, 01
- Το πρωτόκολλο συνέπειας (cache coherency) εξασφαλίζει ότι τα αντίγραφα **μίας** μεταβλητής θα είναι πάντα ενημερωμένα. Αν τροποποιούνται **δύο** άσχετες μεταξύ τους μεταβλητές;
  - Δεν εξασφαλίζει πώς / πότε / με ποια σειρά 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 συνέπειας 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

1. Κάθε CPU ολοκληρώνει τις αναφορές της στη μνήμη με τη *σειρά προγράμματός* της (δεν υπάρχει καθολική σειρά)
2. 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* 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)
  - Εντολές read-modify-write (π.χ. SWAP) επιβάλλουν program order

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)
- Εφόσον χρησιμοποιούνται μόνο μηχανισμοί που παρέχονται από το σύστημα (π.χ. κλειδαριές, κλήσεις συγχρονισμού) και όχι δικά μας «τρυκ», το πρόβλημα της συνέπειας δεν μας απασχολεί καθόλου.