9/5/2023
Συστήματα κατανεμημένης μνήμης (II)

NUMA, DSM, clusters

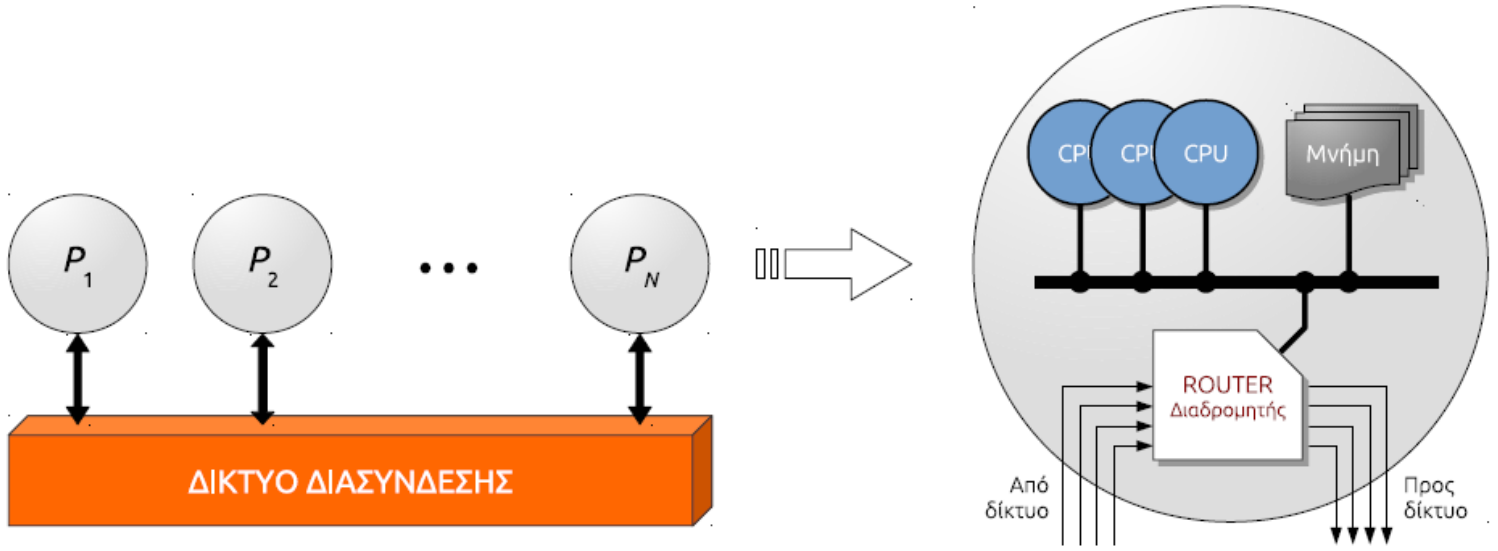
Λ8

Συστήματα & Λογισμικό Υψηλών Επιδόσεων

Κλιμακώσιμα συστήματα, υπολογιστικές συστάδες (clusters)

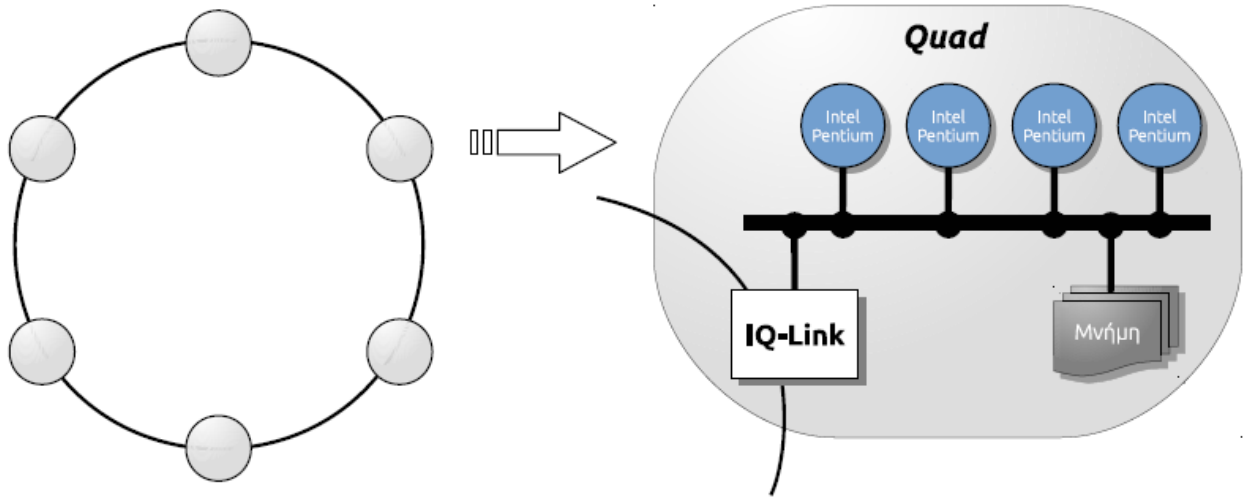
Ομαδοποιημένοι ή κλιμακώσιμοι πολυεπεξεργαστές

- **Scalable** multiprocessors (κλιμακώσιμοι)
- Πολυεπεξεργαστές με οργάνωση κατανεμημένης μνήμης μόνο που:
  - *Κάθε κόμβος αποτελείται από ομάδα επεξεργαστών / πυρήνων που μοιράζονται την τοπική μνήμη.*
  - Άρα, κάθε ομάδα είναι ένας μικρός συμμετρικός πολυεπεξεργαστής
- Το δίκτυο διασύνδεσης συνδέει τις ομάδες



Παραδείγματα (παλιά)

- **SGI Origin 2000** (και μετέπειτα UV-1000, UV-2000):
  - Κόμβος =1 πλακέτα με 2 CPUs MIPS και κοινόχρηστη τοπική μνήμη
  - Σύστημα = υπερκύβος που συνδέει πλακέτες-κόμβους
- **Sequent Numa-Q**:
  - Κόμβος = Pentium Quad (4 επεξεργαστές Pentium) με κοινόχρηστη τοπική μνήμη
  - Σύστημα = δακτύλιος μεταξύ των κόμβων



Υπολογιστικές συστάδες (clusters) και πλέγματα (grids)

- Σύνολο από αυτόνομους υπολογιστές (PCs) συνδεδεμένα με ένα δίκτυο μεταξύ τους:
  - Οι *συστάδες (clusters)* είναι συνήθως ομοιογενείς (παρόμοιοι υπολογιστές, με ίδιο λειτουργικό σύστημα) και είναι τοποθετημένοι στον ίδιο χώρο
  - Τα *πλέγματα (grids)* αναφέρονται συνήθως σε ανομοιογενείς συλλογές από υπολογιστές, με διαφορετικά χαρακτηριστικά και λειτουργικά συστήματα, οι οποίο επίσης μπορεί να είναι εξαπλωμένοι γεωγραφικά σε μεγάλες αποστάσεις.
  - Μιλάμε για clusters αλλά ότι πούμε αντίστοιχα ισχύει και για πλέγματα

- Κάθε κόμβος-PC σε ένα cluster είναι ένας μικρός SMP
  - Π.χ. διαθέτει είτε πολλαπλούς επεξεργαστές είτε – πλέον το συνηθέστερο – έναν πολυπύρηνο επεξεργαστή
  - Επομένως το όλο σύστημα έχει οργάνωση είναι ομαδοποιημένου πολυεπεξεργαστή.

Υπολογιστικές συστάδες (clusters) και πλέγματα (grids)

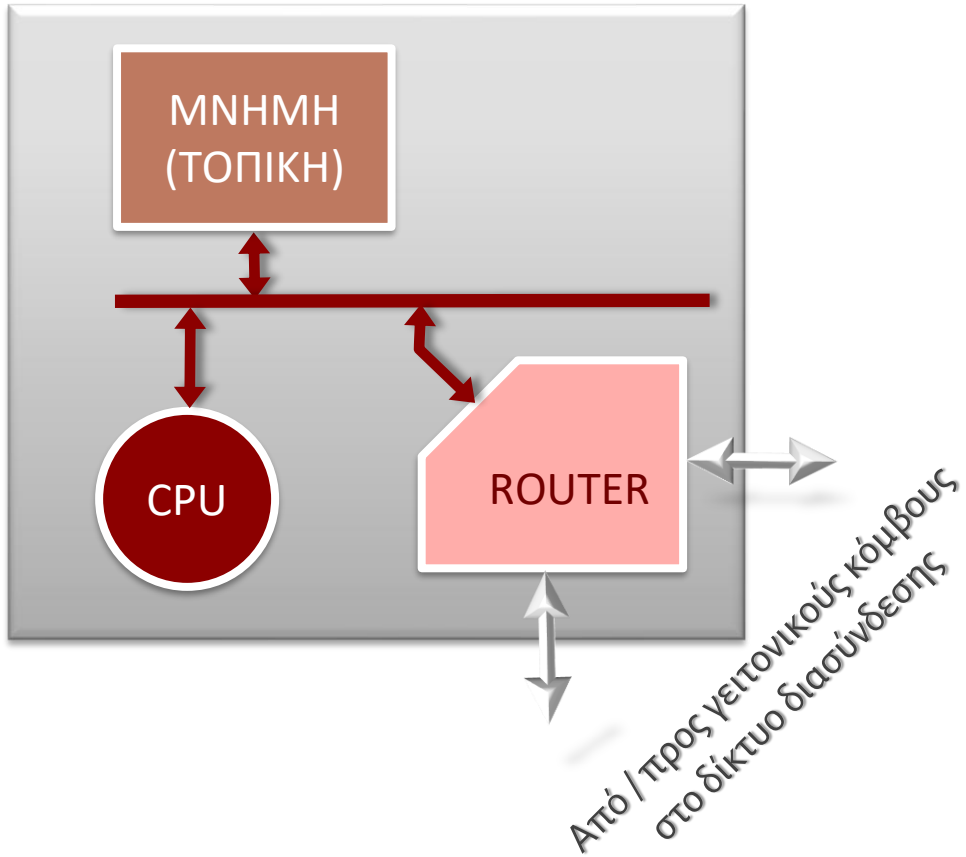
- Δίκτυο διασύνδεσης
  - Κάρτα δικτύου = διαδρομητής
  - Από σχετικά αργό αλλά πολύ οικονομικό (π.χ. Gbit Ethernet)
  - Έως πολύ γρήγορο και ακριβό (π.χ. Myrinet, Infiniband)
- Κάθε κόμβος = αυτόνομος υπολογιστής, με δικό του χώρο μνήμης και δικό του λειτουργικό σύστημα (συνήθως Linux)
  - Υπάρχει όμως ενδιάμεσο λογισμικό ώστε να δίνεται η εντύπωση – ψευδαίσθηση ενιαίου συστήματος (SSI – Single System Image), αν χρειάζεται
- Προγραμματίζονται σαν να είναι 1 μηχανή
  - Κυρίως με μεταβίβαση μηνυμάτων

Κατανεμημένη κοινή μνήμη, NUMA

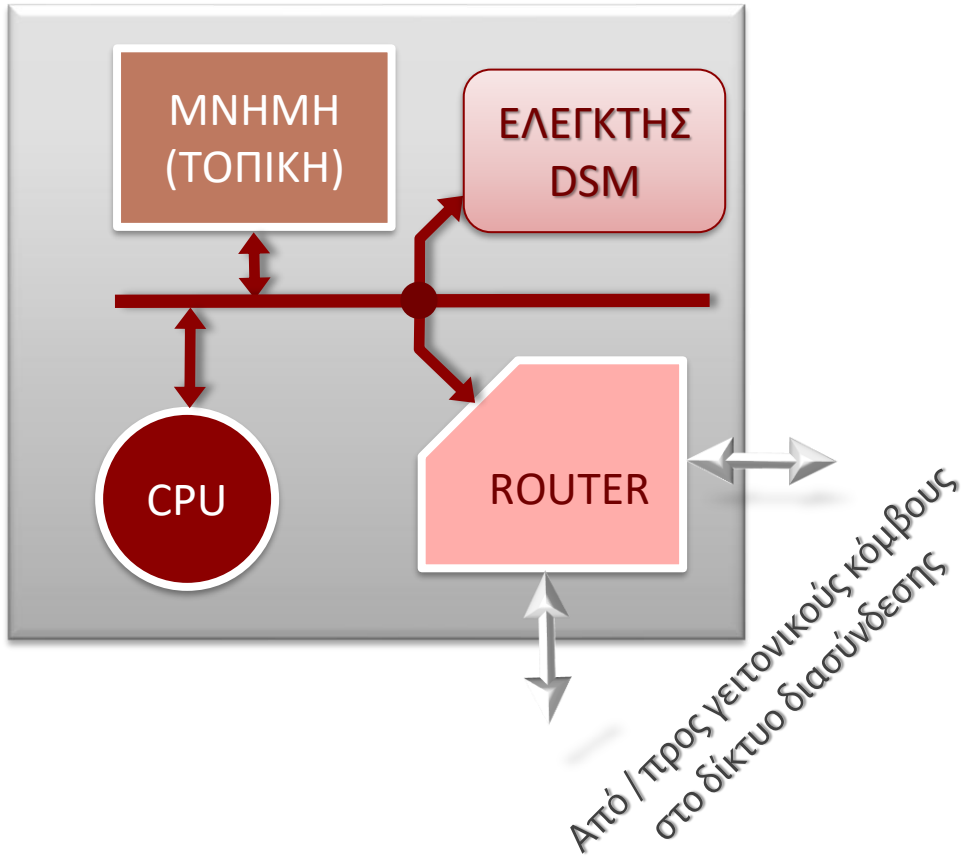
Γιατί;

- Συστήματα κατανεμημένης μνήμης:
  - Αρχιτεκτονική: **κλιμακώσιμη**
  - Προγραμματισμός (MPI): αρκετά **δύσκολος** αλλά και δυνατότητα επιδόσεων
- Συστήματα κοινόχρηστης μνήμης (SMPs):
  - Αρχιτεκτονική: **δύσκολα κλιμακώσιμη**
  - Προγραμματισμός: **οικείος / προσιτός**
- Το ιδεώδες:
  - Κλιμακώσιμες αρχιτεκτονικές που να προγραμματίζονται εύκολα (αλλά μην ξεχνάμε και τις επιδόσεις)
  - Μετά το σειριακό, το πιο εύκολο είναι ο προγραμματισμός κοινής μνήμης (π.χ. OpenMP)
- Λύση: *«emulation» της κοινόχρηστης μνήμης πάνω από το σύνολο των ιδιωτικών μνημών*
  - με hardware
  - με software (π.χ. σε clusters όπου είναι αδύνατον να αλλάξει το hardware)
  - Άρα *λογικά* κοινόχρηστη, *φυσικά* κατανεμημένη μνήμη

Hardware – κόμβος συστήματος

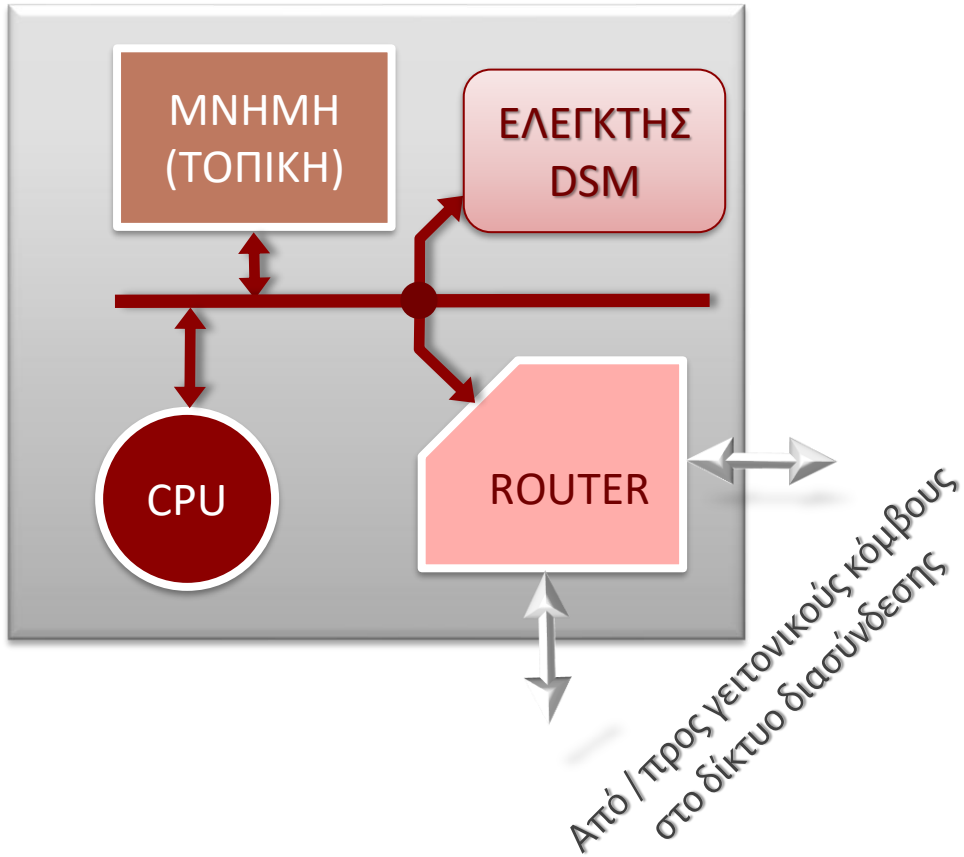


Ανομοιόμορφη προσπέλαση μνήμης (NUMA)



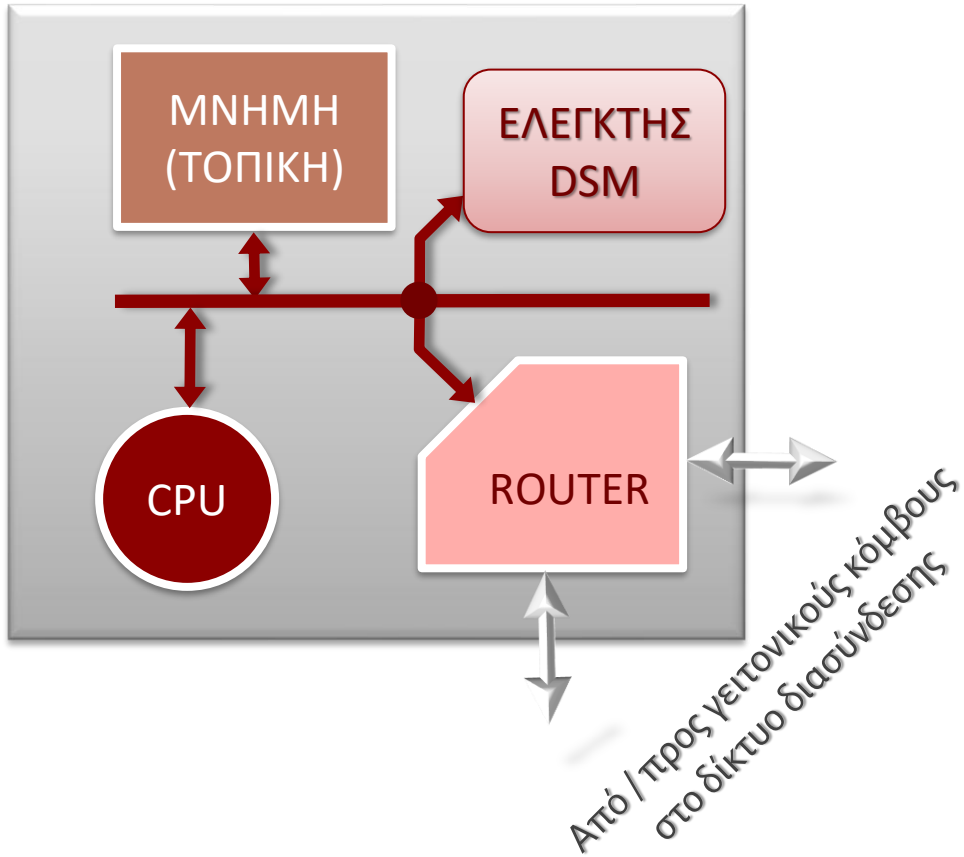
- Η CPU προσπελαύνει όλο τον χώρο διευθύνσεων ενιαία
- Η τοπική μνήμη έχει μόνο ένα μικρό κομμάτι του χώρου
- Ο ελεγκτής DSM ελέγχει κάθε διεύθυνση που προσπελαύνει η CPU
  1. Αν είναι για την τοπική μνήμη δεν κάνει τίποτε
  2. Αν όχι, αναλαμβάνει την επικοινωνία με τον κόμβο που την χειρίζεται (**home node**), στέλνοντας κατάλληλο μήνυμα. Μόλις έρθει η απάντηση, δίνει δεδομένα στη CPU σαν να ήταν αποθηκευμένο τοπικά

Ανομοιόμορφη προσπέλαση μνήμης (NUMA)



- Το μόνο που καταλαβαίνει η CPU είναι η διαφορά στην ταχύτητα προσπέλασης κάποιων δεδομένων (τα απομακρυσμένα κάνουν πολύ παραπάνω χρόνο να έρθουν)
  - NUMA (non-uniform memory access)
  - Π.χ. Cray T3D: 2 κύκλοι για τα τοπικά και περίπου 150 κύκλοι για τα απομακρυσμένα δεδομένα
  - Συστήματα με 4 επεξεργαστές AMD Opteron 8347HE: περίπου 26% (32%) επιπλέον χρόνος για απομακρυσμένη ανάγνωση (εγγραφή).

Ανομοιόμορφη προσπέλαση μνήμης (NUMA)



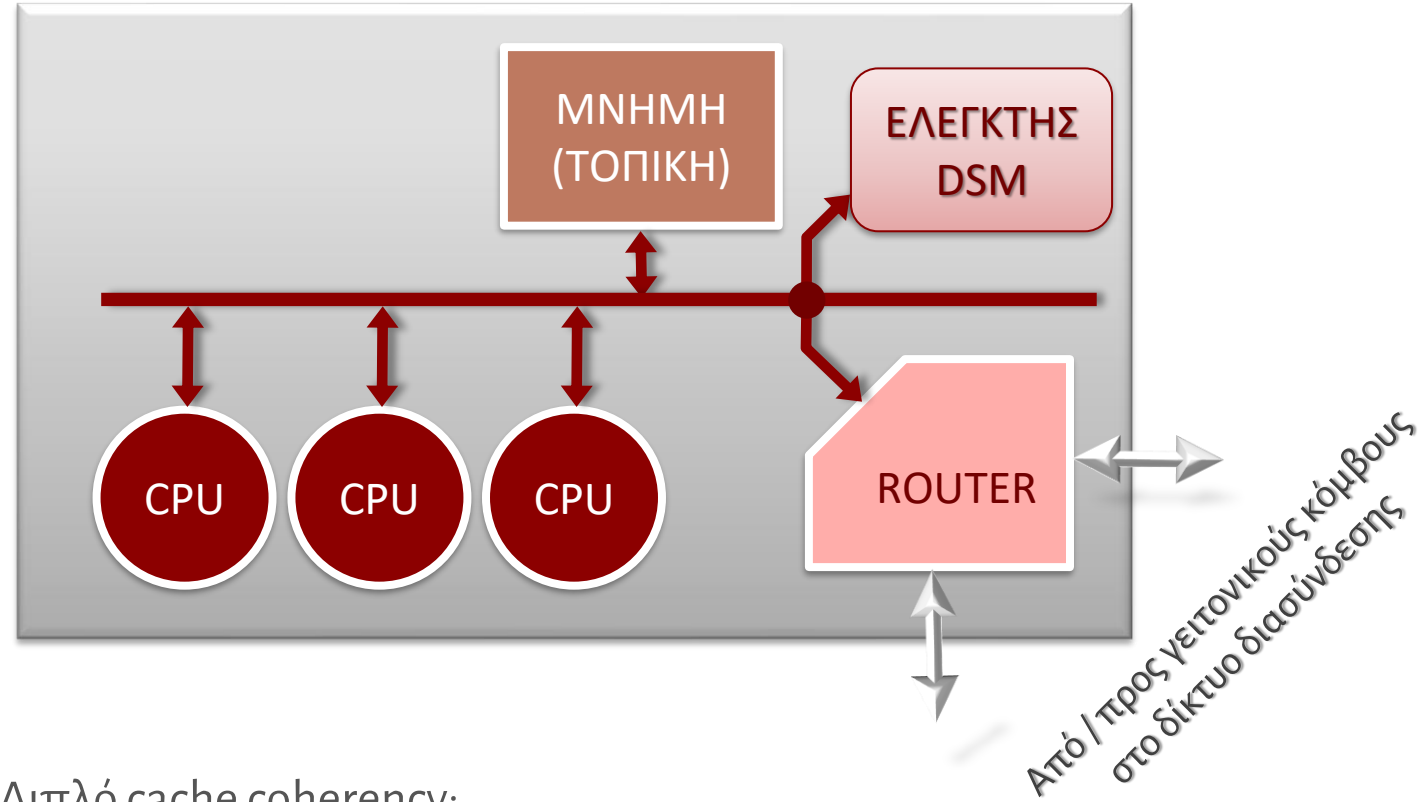
Ερώτηση: Μείωση χρόνου προσπέλασης???

- Απάντηση: caches για τα απομακρυσμένα δεδομένα
- Όμως, πάλι: πρόβλημα συνοχής για τα κοινά δεδομένα!

Λύσεις:

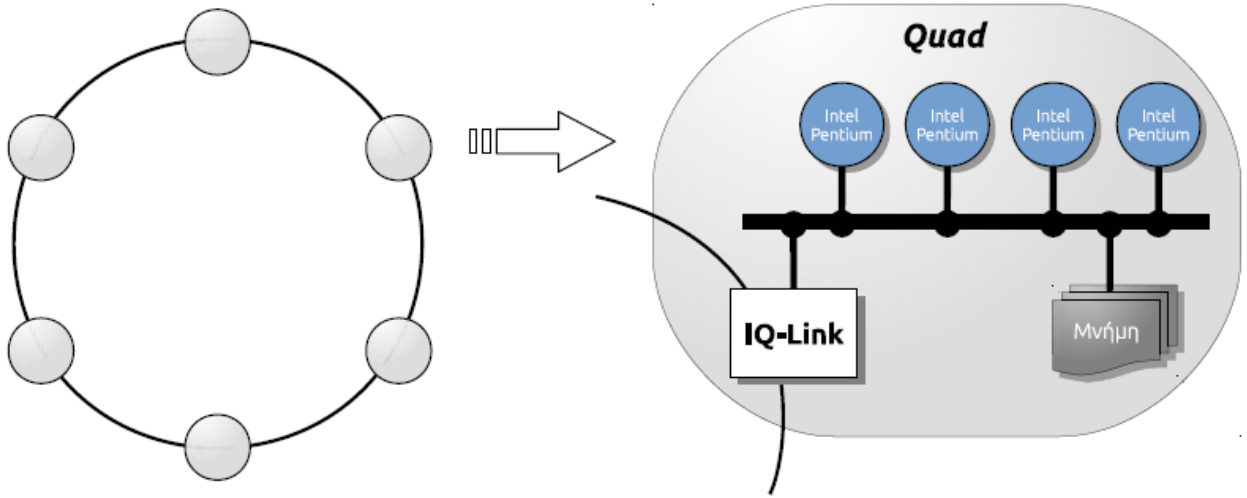
- Πρωτόκολλα συνοχής => cache-coherent NUMA (ccNUMA)
- Αποφυγή προβλήματος. Π.χ. στον Cray T3D δεν υπήρχε πρωτόκολλο συνοχής. Απλά, στις caches δεν επιτρέπονταν κοινά δεδομένα

Γενίκευση: πολυπύρηνος ή πολυεπεξεργαστικός κόμβος



- Διπλό cache coherency:
  - «εξωτερικό» πρωτόκολλο από ελεγκτή DSM για απομακρυσμένα δεδομένα
    - **Υποχρεωτικά πρωτόκολλο καταλόγων**
  - «εσωτερικό» πρωτόκολλο από caches των CPUs του κόμβου για τα τοπικά
    - Πρωτόκολλο snooping

Παράδειγμα: Sequent Numa-Q



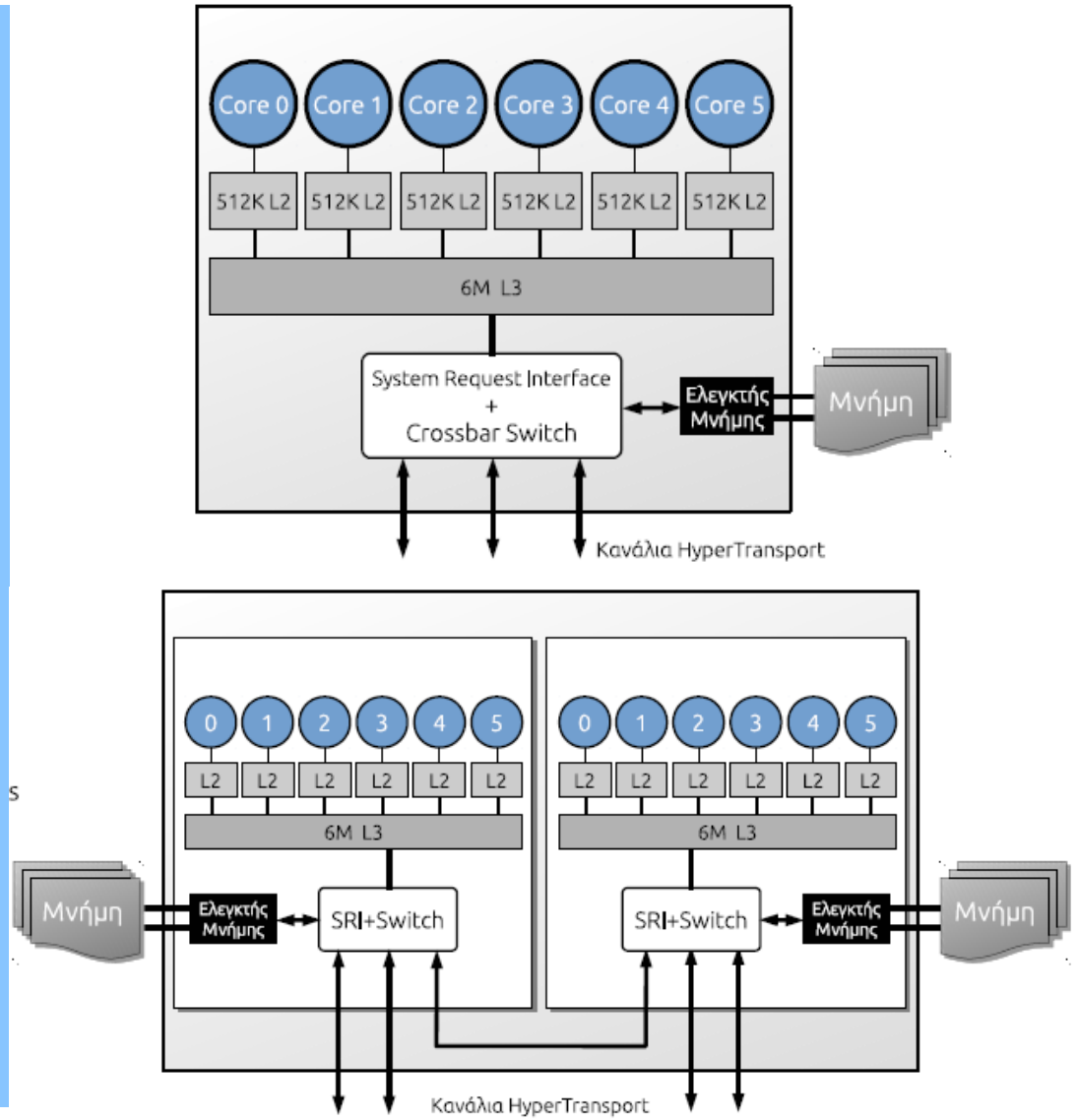
- Το IQ-Link ενσωμάτωνε το ρόλο του διαδρομητή και του ελεγκτή DSM.
- Υλοποιούσε το εξωτερικό πρωτόκολλο
  - SCI (αλυσιδωτοί κατάλογοι)
- Εσωτερικά στην ομάδα, οι Pentium είχαν διατηρούσαν τη συνέπεια με πρωτόκολλο MESI

Κατανεμημένη μνήμη και multicores

Πολυπύρηνοι επεξεργαστές και κατανεμημένη μνήμη

- Κυρίως δύο περιπτώσεις:
  - Χρήση multicore CPUs ως κόμβων ενός μεγαλύτερου συστήματος
    - Το σύστημα δομείται ως ένα δίκτυο από τέτοιους κόμβους
  - Χρήση multicore CPU ώστε να αποτελεί ολόκληρο το σύστημα
    - Λογικά, μιλάμε για CPU με αρκετούς πυρήνες (ίσως many-core CPU)
- Στην πρώτη περίπτωση ο κάθε επεξεργαστής έχει δικά του κανάλια για να συνδεθεί με τοπική μνήμη + επιπλέον κανάλια για να συνδεθεί με άλλους επεξεργαστές
  - AMD Epyc (Opterons παλαιότερα)
  - Intel Xeon (ξεκινώντας από τη σειρά E5)
- Στη δεύτερη περίπτωση, μέσα στον ίδιο τον επεξεργαστή περιλαμβάνονται οι πυρήνες, οι ιδιωτικές μνήμες και το δίκτυο διασύνδεσης
  - NoC (Network-on-Chip)
  - MPSoC / MCSoC (Multiprocessor/Multicore System-on-Chip)

Παράδειγμα: AMD Opterons



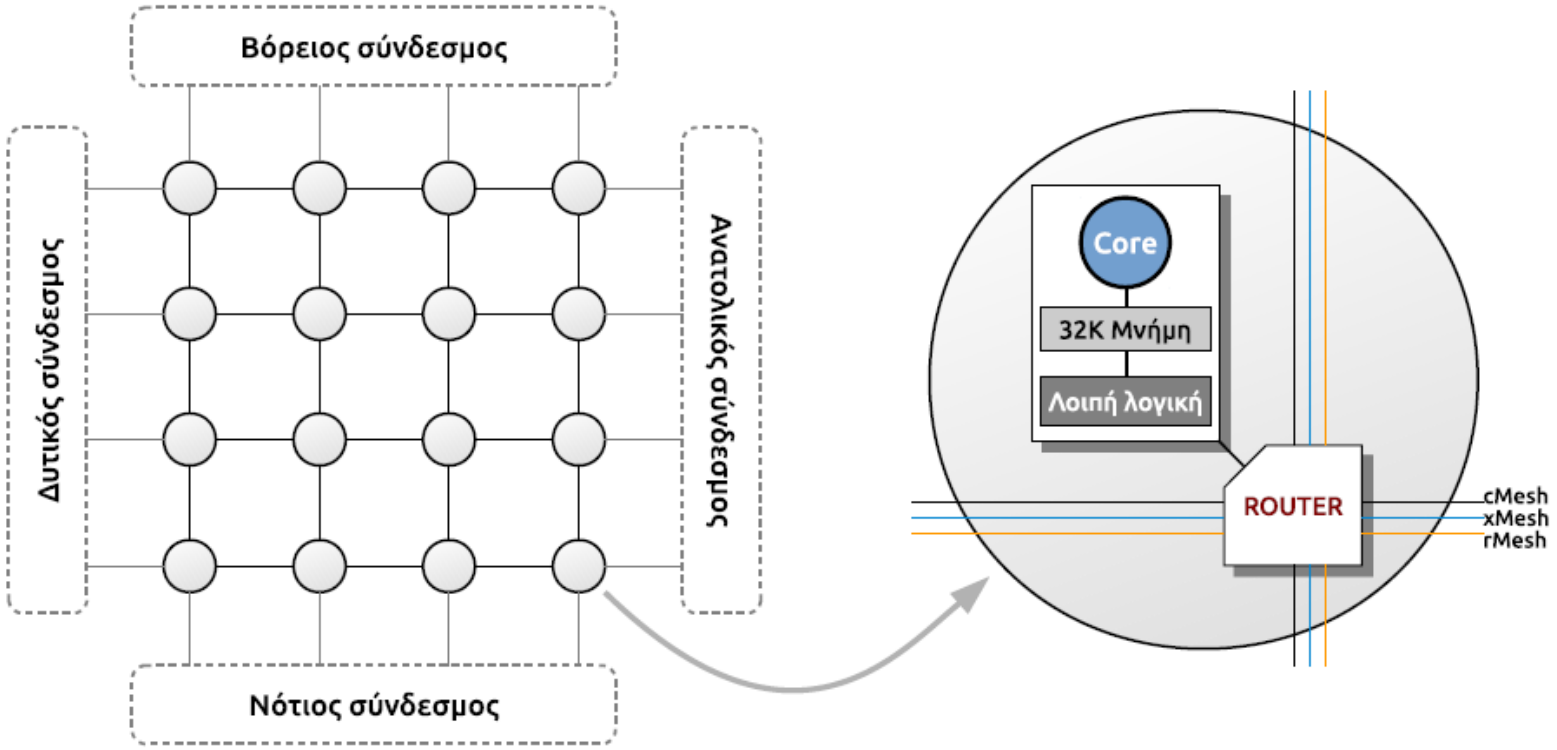
ISTANBUL (6-πύρηνος)

- 3 κανάλια HT ανά επεξεργαστή
- Μεταγωγή Virtual Cut-Through
- Μπορεί να συνδεθούν 8 κόμβοι με μέγιστη απόσταση 3 hops και να περισσέψουν και κανάλια για I/O
- HT Assist (1MiB στην L3) για υλοποίηση απλού πρωτοκόλλου καταλόγων

MAGNY-COURS (12-πύρηνος)

- 2 Istanbul ενωμένοι
- Μέχρι 4 επεξεργαστές στο σύστημα
- Επομένως μέχρι 48 πυρήνες (χωρίς προσθήκη επιπλέον υλικού)
- *NUMA factor:*
  - ≈1.2 για 1 hop (20% πιο αργή)
  - ≈1.5 για 2 hop (50% πιο αργή)

Παράδειγμα: Epiphany-16



- Για εγγραφές, 1.5 κύκλος per hop
  - για εγγραφή σε μνήμη που είναι σε απόσταση D => 1.5xD κύκλοι
- Epiphany-64 (δεν κυκλοφοράει στο εμπόριο προς το παρόν):
  - πλέγμα 8x8

Παράδειγμα: AMD Opteron & HyperTransport

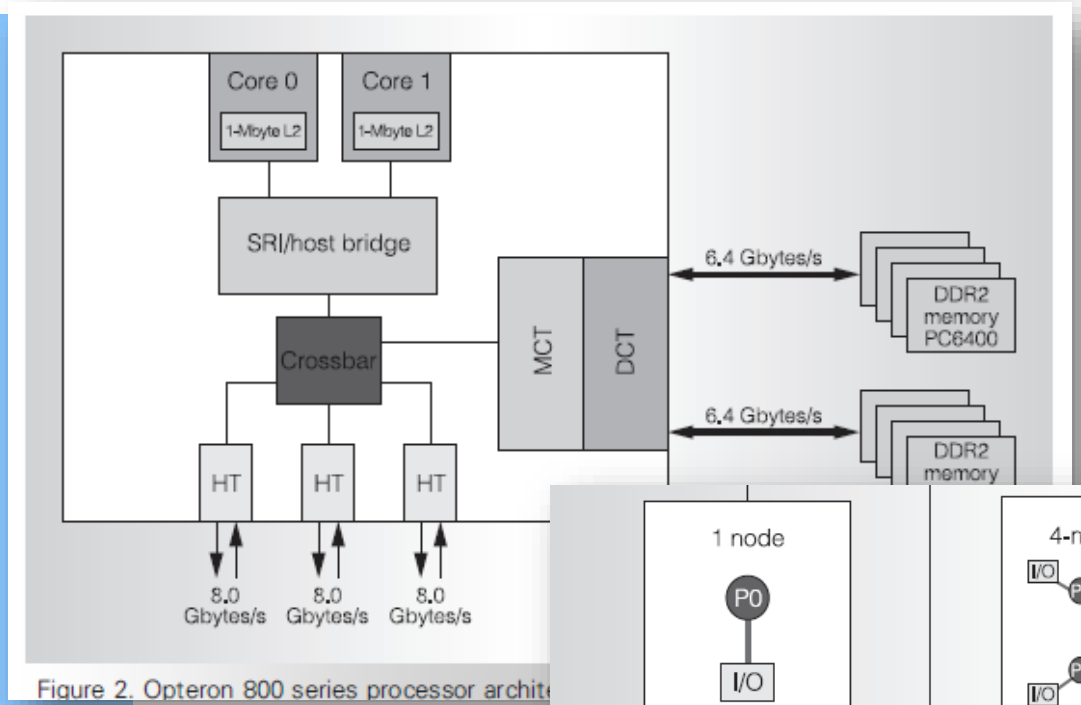
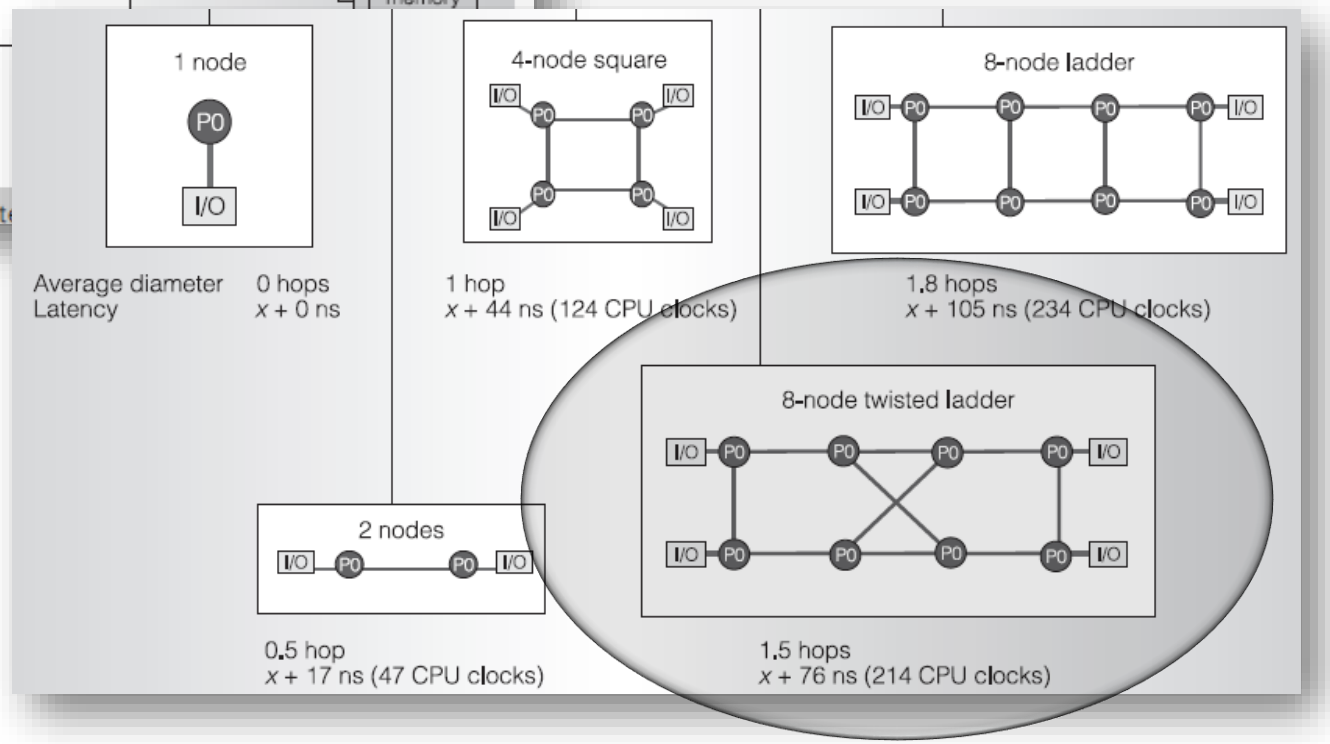


Figure 2. Opteron 800 series processor archite

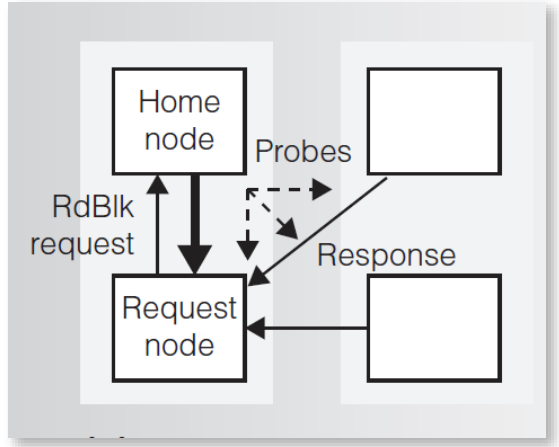
- 3 HT channels per processor (node)
- Cut-through switching
- Can connect 8 nodes with up to 3 hops distance plus leave channels for I/O

  (average distance is < 2)



Παράδειγμα: AMD Opteron & HyperTransport

- **Coherent** HyperTransport
  - Η συνοχή επιτυγχάνεται με flooding (broadcasting)

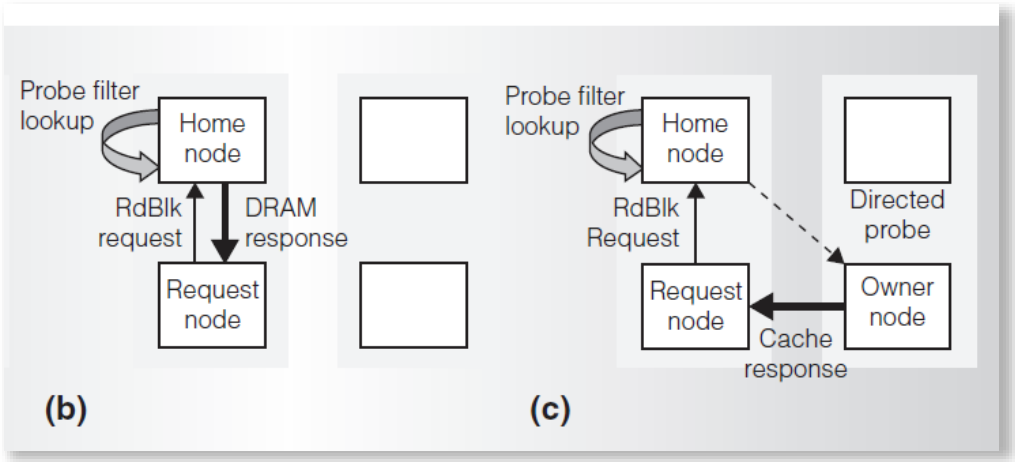


- Typical transaction:
  1. Requestor sends message to home node
     (ζητώντας κάποιο δεδομένο)
  2. Home node forwards / broadcasts to all nodes
     (διότι δεν γνωρίζει ποιος έχει αντίγραφο ενημερωμένο)
  3. Every node replies (with acknowledgement or the data) directly to the requestor
  4. Requestor selects the correct data *and notifies the home node*.

- Αν οι κόμβοι είναι πολλοί, τα βήματα 2 και 3 προκαλούν υπερβολική κίνηση
- Δεν υπάρχει (χρειάζεται) κατάλογος.

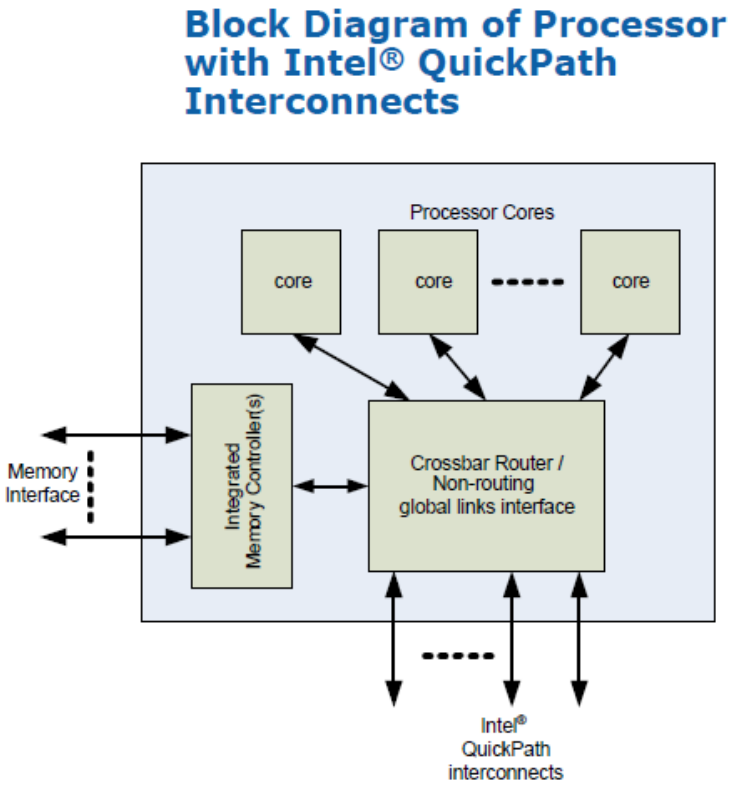
Κλιμακώνοντας το Coherent HT

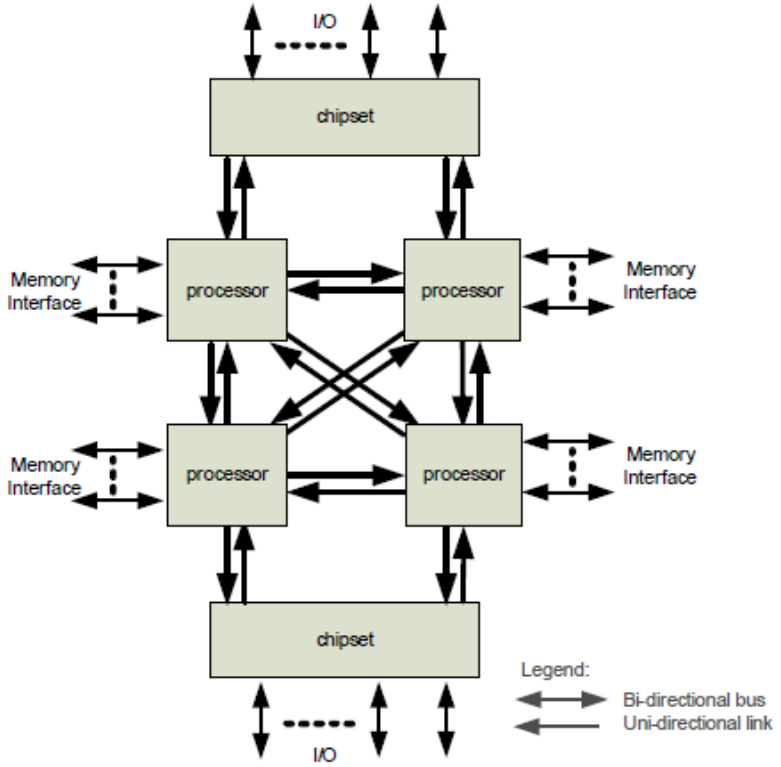
- **"HyperTransport Assist"**
  - Από τη σειρά Magny Cours (έως 12 πυρήνες)
  - Συστήματα με 4 sockets => 48 πυρήνες, το Coherent HT εισάγει υπερβολική κίνηση
- Υπάρχει πλέον κατάλογος (directory) ώστε να αποφεύγονται τα broadcasts
- Ο κατάλογος υλοποιείται στην L3 cache
  - Καταναλώνει ≈ 1 MB από τα 6 MB.
- Δημιουργείται εγγραφή μόνο αν
  - Ο κόμβος είναι home node για το δεδομένο ΚΑΙ το δεδομένο το έχει πάρει και κάποιος άλλος επεξεργαστής
  - Άρα το cache miss σημαίνει ότι το δεδομένο δεν είναι cached πουθενά
  - Ο κατάλογος είναι «μερικός»:
    - 1 sharer ή more than one sharer



Intel QuickPath (QPI)

- Ξεκινώντας με τους επεξεργαστές Core i7-9xx

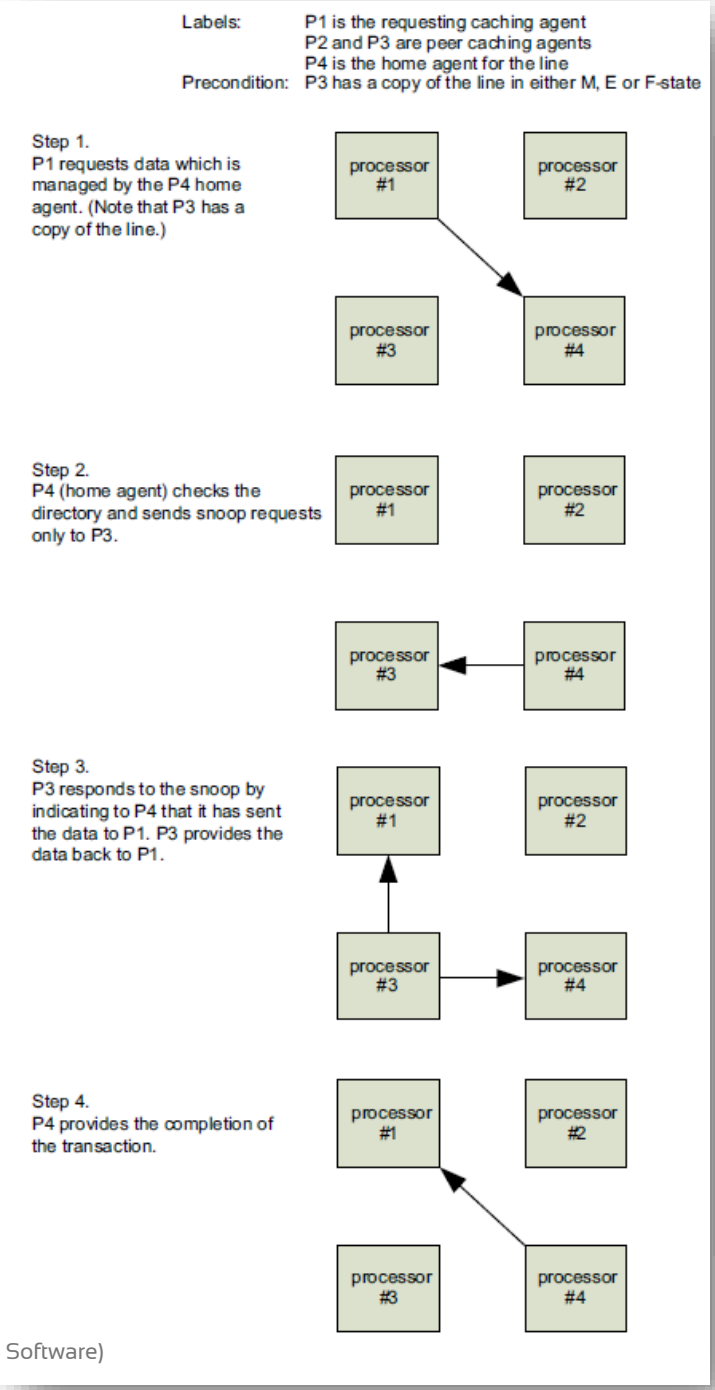




QPI Home "Snoop"

- Το "snoop" είναι ατυχές...

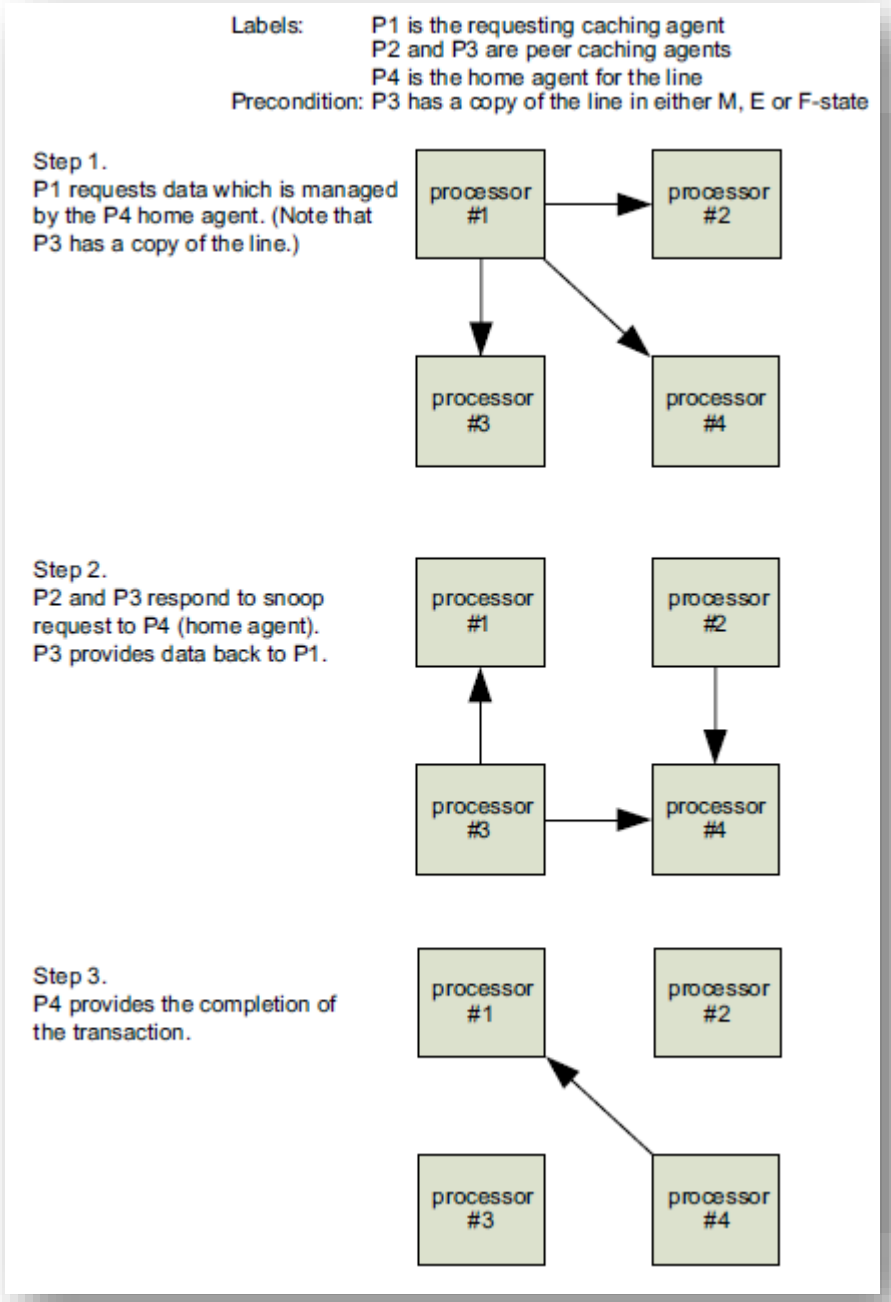
1. The caching agent issues a request to the home agent that manages the memory in question.
2. The home agent uses its directory structure to target a snoop to the caching agent that may have a copy of the memory in question.
3. The caching agent responds back to the home agent with the status of the address. In this example, processor #3 has a copy of the line in the proper state, so the data is delivered directly to the requesting cache agent.
4. The home agent resolves any conflicts, and if necessary, returns the data to the original requesting cache agent (after first checking to see if data was delivered by another caching agent, which in this case it was), and completes the transaction.



QPI Source "Snoop"

- Το "snoop" είναι ατυχές...

1. The caching agent issues a request to the home agent that manages the memory in question and issues snoops to all the other caching agents to see if they have copies of the memory in question.
2. The caching agents respond to the home agent with the status of the address. In this example, processor #3 has a copy of the line in the proper state, so the data is delivered directly to the requesting cache agent.
3. The home agent resolves any conflicts and completes the transaction.



QPI

- Το δεύτερο (source snoop) «γλιτώνει» ένα βήμα, άρα χαμηλότερο latency
  - Καλύτερο για μικρά συστήματα
- Όμως δημιουργεί μεγαλύτερη κίνηση
  - Το home snoop είναι κλιμακώσιμο και κάνει για μεγαλύτερα συστήματα.
- Το source snoop είναι παρόμοιο με το Coherent HT
  - Όμως ο requestor (και όχι ο home) είναι αυτός που εκπέμπει σε όλους
  - Αυτό δυσκολεύει τη διευθέτηση των races (όταν γίνουν ταυτόχρονες αιτήσεις για το ίδιο δεδομένο, οι οποίες μεταδίδονται στους κόμβους με διαφορετική ίσως σειρά) – κάτι που το έλυνε ο home στο coherent HT.

Κατανεμημένη κοινή μνήμη με λογισμικό

sDSM (Software DSM) ή SVM (shared virtual memory)

Software DSM

- Τι γίνεται αν θέλουμε να υποστηρίξουμε κοινόχρηστη μνήμη ανάμεσα στους κόμβους ενός cluster?
  - Δεν υπάρχει δυνατότητα επέμβασης στο hardware
  - Άρα μόνο λύσεις software
- Κλασική υλοποίηση: *page-based sDSM*
  - Χωρίς παρεμβάσεις στο Λ.Σ. (είναι μία απλή εφαρμογή σε επίπεδο χρήστη)
  - Όλοι οι συμμετέχοντες κόμβοι «δεσμεύουν» έναν χώρο μνήμης (πολλές *σελίδες*) που θα τον έχουν ως κοινόχρηστο
    - Τα κοινόχρηστα δεδομένα θα τοποθετούνται σε αυτό τον χώρο
  - Καθένας «μαρκάρει» τις σελίδες ως «απαγορευμένης προσπέλασης» και ορίζει έναν χειριστή σημάτων για το σήμα SIGSEGV
  - Η εφαρμογή ξεκινά και οι προσπελάσεις στον κοινόχρηστο χώρο οδηγούν σε σφάλματα μνήμης που εκκινούν τον χειριστή για το SIGSEGV.
  - Ο χειριστής:
    - επικοινωνεί με τον κόμβο που διαθέτει το δεδομένο, φέρνοντας ολόκληρη τη σελίδα που το περιέχει
    - μετά τη λήψη και την τοποθέτηση στη σωστή θέση, αλλάζει τα δικαιώματα της σελίδας (π.χ. επιτρέπεται πλέον η ανάγνωση)
    - τελειώνει, επανεκτελώντας την εντολή που προκάλεσε τη διακοπή

Ιδέες για υλοποιήσεις page-based sDSM

- Μία σελίδα *p* θα υπάρχει μόνο σε έναν κόμβο
  - Κεντρικός server που γνωρίζει ποιος κόμβος έχει ποια σελίδα
    - Απορρίπτεται – μεγάλη κίνηση στον server
  - Κατανεμημένα & προκαθορισμένα – π.χ. ο κόμβος *p* % N είναι υπεύθυνος για γνωρίζει που βρίσκεται η σελίδα p
    - Καλύτερο, μιας και κάθε κόμβος μπορεί να ρωτήσει κατευθείαν αυτόν που πρέπει
  - Αν υπάρχουν πολλοί readers, σε κάθε προσπέλαση «σελίδες πάνε κι έρχονται»
- Λύση: πολλαπλά αντίγραφα σελίδων (page replication)
  - Απλούστερη υλοποίηση: multi-readers **Ή** single-writer
    - Αν εμφανιστεί writer, τότε όλα τα read-only copies ακυρώνονται
    - Στο write υπάρχει *μεγάλο* overhead
    - Πολύ καλό αν υπάρχουν πολλά reads και ελάχιστα writes
  - Καλύτερη (σε γενικές γραμμές) υλοποίηση: multi-readers **ΚΑΙ** multi-writers
    - Πιο δύσκολος ο χειρισμός των πολλαπλών εγγραφέων

Πρωτόκολλα home-based (multi-writer)

- Home-based
  - Ένας κόμβος (συνήθως ο πρώτος που θα γράψει στη σελίδα) επιλέγεται να είναι το «σπίτι» της σελίδας αυτής
  - *Κατά την εγγραφή*, οι άλλοι κόμβοι στέλνουν τις αλλαγές στον home node.
    - Ο home node στέλνει **ακυρώσεις** (invalidations) σε όλα τα άλλα αντίγραφα

    Τι στέλνουν οι writers στον home node ?
    - Τη νέα σελίδα: απλό, αλλά μεγάλο false sharing
    - Diffs (μόνο τις αλλαγές): ταχύτερο, αλλά ο κάθε κόμβος πρέπει να φυλάει και δεύτερο αντίγραφο (“twin”) από τις σελίδες που τροποποιεί για να βρίσκει τις αλλαγές που έκανε
  - *Κατά την ανάγνωση*, οι άλλοι κόμβοι παίρνουν την σελίδα (ή τις αλλαγές) απευθείας από τον home node.

- Πιθανή βελτιστοποίηση:
  - adaptive (floating/migrating home)

Συνέπεια μνήμης (memory consistency)

- Κανονικά σε κάθε εγγραφή θα πρέπει να γίνεται ενημέρωση του home και ακύρωση των αντιγράφων που υπάρχουν στο δίκτυο
- Μπορεί να είναι υπερβολική η κίνηση που δημιουργείται, πράγμα που μειώνει πολύ τις επιδόσεις
- Η λύση είναι να «χαλαρώσουν» λίγο οι απαιτήσεις, *αποφεύγοντας να ενημερώνουμε τους υπόλοιπους κόμβους όταν κάνουμε μία εγγραφή*
  - **Πλέον οι τροποποιήσεις ΔΕΝ ανακοινώνονται στην ώρα τους αλλά κάποτε αργότερα**
  - «Χαλαρώνει» το μοντέλο της μνήμης μιας και πλέον οι προσπελάσεις στις κοινές μεταβλητές είναι κάπως ανεξέλεγκτες, π.χ. στο παρακάτω

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) */

  - … θα μπορούσαμε άνετα να δούμε εκτύπωση **10** όπως είχαμε δει.

Sequential Consistency

- Η *ακολουθιακή συνέπεια (sequential consistency)* είναι λογικός τρόπος λειτουργίας
  - Όλες οι διεργασίες *εκτελούν και ολοκληρώνουν* τις εντολές (r/w) με τη **τοπική/ιδιωτική σειρά προγράμματος** και τα writes είναι **ατομικά**.
    - Συνολικά η σειρά γεγονότων είναι «τυχαία» αλλά πάντως προκύπτει από κάποιο interleaving των εντολών των διεργασιών (είναι όμως **μία,** δηλαδή όλες οι διεργασίες αντιλαμβάνονται την **ίδια** σειρά γεγονότων)
  - Δεν απαιτούνται συγχρονισμένα ρολόγια
  - Όμως, τα writes μεταδίδονται σε όλους **άμεσα**
  - Μειώνει πολύ τις επιδόσεις όπως είπαμε λόγω του υψηλού επικοινωνιακού φόρτου

Relaxed Consistency

- **ΛΥΣΗ:** Χαλάρωμα της συνέπειας για επιδόσεις, αλλά όπως είχαμε δει, αλλάζει πολύ τα δεδομένα για τον προγραμματιστή, ο οποίος πρέπει να είναι πολύ προσεκτικός:
  - Δεν πρέπει κοινές μεταβλητές να προσπελαύνονται εκτός κρίσιμων περιοχών
  - Τα «χαλαρά» μοντέλα μνήμης και τα αντίστοιχα πρωτόκολλα που χρησιμοποιούνται παρέχουν «ορθή» λειτουργία μόνο όταν οι προσπελάσεις γίνονται μέσα σε κρίσιμες περιοχές (lock/unlock ή acquire/release).
  - Επομένως, οι εγγραφές σε κοινές σελίδες είναι μόνο γνωστές στον κόμβο που τις έκανε – **οι υπόλοιποι ενημερώνονται μόνο σε κατάλληλα σημεία συγχρονισμού / αμοιβαίου αποκλεισμού**
  - Λειτουργεί, μιας και εκτός κρίσιμων περιοχών ο προγραμματιστής εφαρμογών δεν (πρέπει να) προσπελαύνει αυτές τις σελίδες. Αν κάποιος χρειαστεί να προσπελάσει κοινές μεταβλητές εκτός κρίσιμων περιοχών θα πρέπει «με το χέρι» να φροντίσει να μεταδοθούν οι όποιες αλλαγές έγιναν.

Release Consistency (RC)

- (Eager) RC: Οι αλλαγές μεταδίδονται μόνο *κατά την έξοδο* από την κρίσιμη περιοχή (κατά το unlock/release)
  - Στα home-based συστήματα τα diffs των τροποποιημένων σελίδων πάνε στο αντίστοιχο home ΚΑΙ ενημερώνονται/ακυρώνονται τα άλλα αντίγραφα στους υπόλοιπους κόμβους
  - Στα homeless, ενημερώνονται κατευθείαν όλοι
- LRC (Lazy Release Consistency): Οι αλλαγές μεταδίδονται οριστικά *κατά την είσοδο* στην κρίσιμη περιοχή (κατά το lock/acquire)
  - Στο release στέλνονται οι αλλαγές μόνο στο home
  - Όποιος κάνει στη συνέχεια acquire, επικοινωνεί με το home και τότε παίρνει τις αλλαγές
  - Μειώνει την κίνηση στο release
  - π.χ. Treadmarks (δεν ήταν όμως home-based)
- Στα σημεία καθολικού συγχρονισμού (barriers) ενημερώνονται τα πάντα

Παράδειγμα: HLRC (Home-based Lazy Release Consistency)

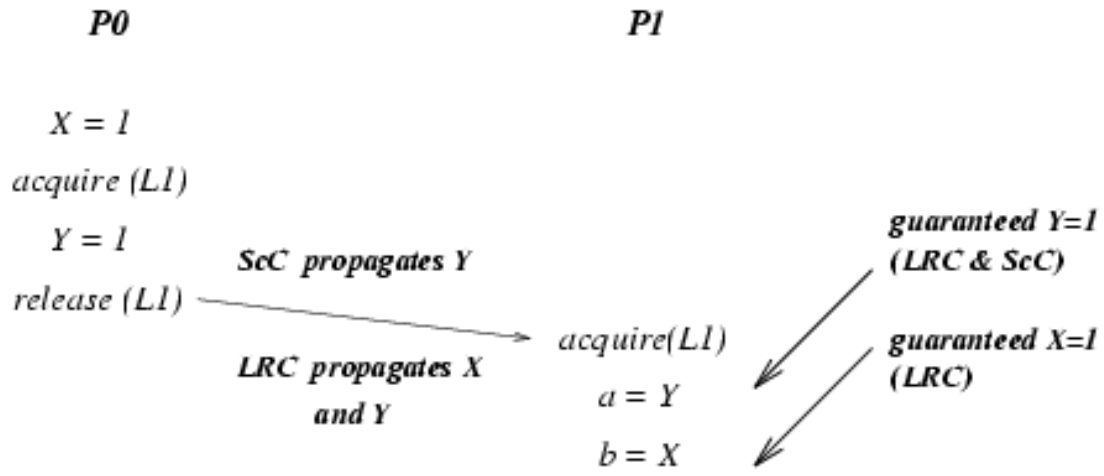
- Στο *release*
  - Υπολογισμός diffs (από τα twin που έχει κάθε κόμβος)
  - Αποστολή diffs στον home node
  - Ο home node:

    Εφαρμόζει τα diffs όπως έρχονται, ενημερώνοντας τη σελίδα

- Στο *acquire*
  - Ο κόμβος ζητά από τον home node τη νέα σελίδα

ScC (Scope Consistency)

- Παρόμοιο με το LRC μόνο που για κάθε scope (κρίσιμη περιοχή – ορίζεται από την κλειδαριά της) οι ενημερώσεις αφορούν *μόνο τις σελίδες που τροποποιήθηκαν στο scope αυτό*.
  - Ενώ στο LRC, στο release μεταδίδονται όλες οι αλλαγές σε όλες τις σελίδες, ακόμα και αυτές που έγιναν πριν την κρίσιμη περιοχή



Scope Consistency versus Lazy Release Consistency.

  - Πλήρης ενημέρωση μόνο στα barriers