Υ-07 – Παράλληλα Συστήματα
Συνοχή κρυφής μνήμης με σύστημα καταλόγων

Αρης Ευθυμίου

Γιατί όχι snooping

- Το snooping στηρίζεται σε εκπομπή σε όλους (broadcast)
  - πρέπει όλοι οι ελεγκτές κρυφής μνήμης να μπορούν να δούν «ταυτόχρονα» κάθε αίτηση προσπέλασης μνήμης των υπόλοιπων επεξεργαστών
- Δουλεύει καλά μόνο σε δίκτυο διασύνδεσης τύπου bus
- Αλλά το bus δεν μπορεί να εξυπηρετήσει πολλούς πυρήνες/επεξεργαστές
  - για παρόμοιους λόγους που το ethernet δεν χρησιμοποιεί πλέον busses αλλά switches/hubs

Εκπομπή σε άλλα δίκτυα

Μπορεί να γίνει

- είτε απευθείας: π.χ. κάθε κόμβος μεταδίδει το μήνυμα σε όλες τις εξόδους του
- είτε με πολλαπλά μυνήματα προς όλους
- Η εκπομπή σε όλους δεν είναι αποδοτική όταν
  - οι πυρήνες είναι πολλοί
  - σχετικά λίγοι από αυτούς μοιράζονται τα ίδια δεδομένα ταυτόχρονα
- Η εκπομπή έχει κόστος
  - ανάλωση ενέργειας
  - ανάλωση bandwidth

Πρωτόκολλο συνοχής

- Ορίζει καταστάσεις, μεταβάσεις καταστάσεων, μηνύματα και πράξεις
- Τι χρειάζεται να κάνει
  - Να αποφασίσει πότε να επικαλεσθεί το πρωτόκολλο
    - Σε περίπτωση αστοχίας της τοπικής cache
  - Να μάθει για την κατάσταση της γραμμής σε άλλες caches (ή να την έχει αποθηκεύσει) για να καθορίσει τί πράξη θα κάνει
    - η πληροφορία συνήθως δεν είναι ακριβής
  - **Να βρεί τα άλλα αντίγραφα**
  - Να επικοινωνίσει με τις caches που έχουν αντίγραφα
    - για να τα ακυρώσει ή να αλλάξει την κατάστασή τους

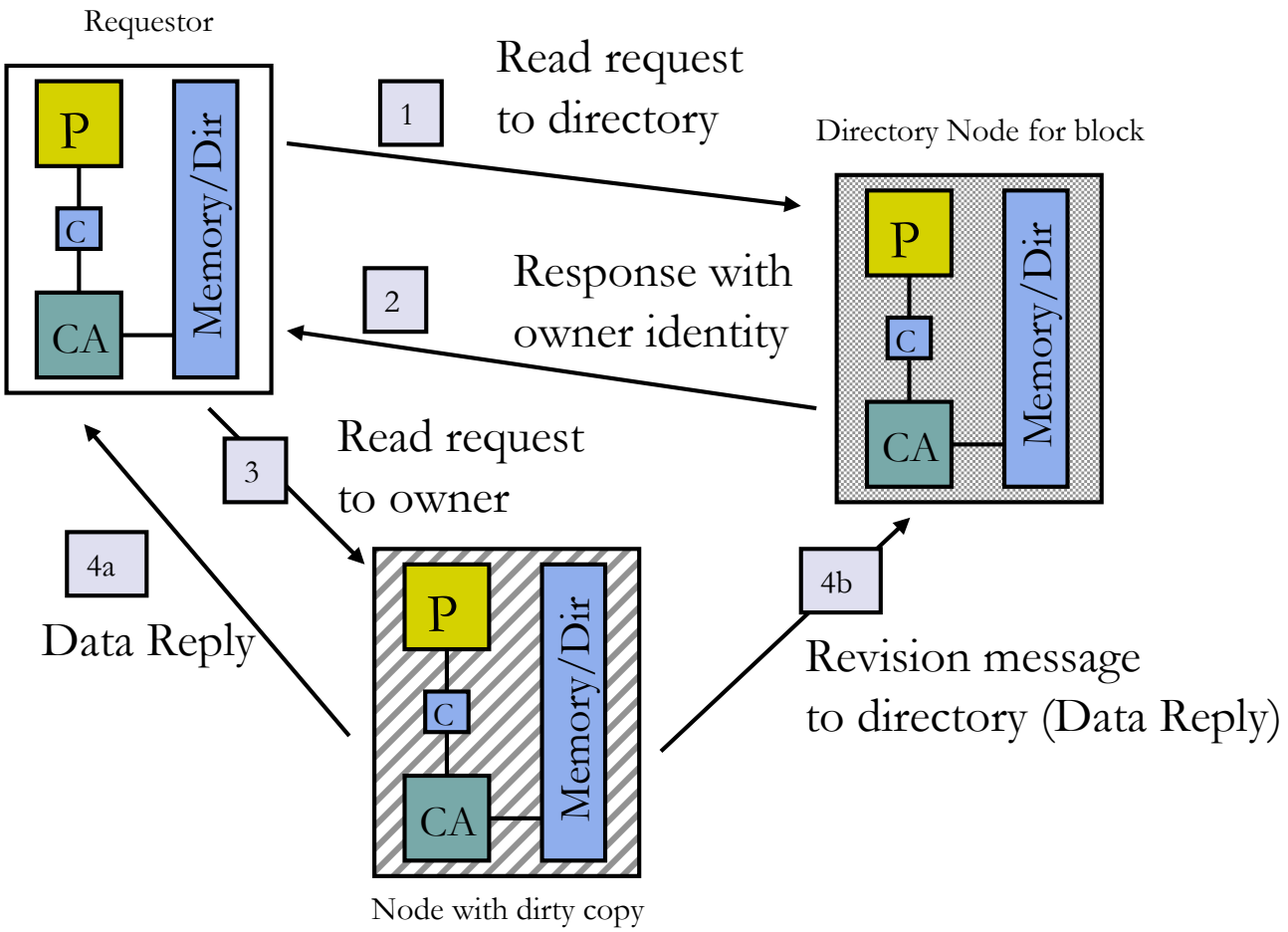
Συνοχή με καταλόγους

- Δύο τρόποι να βρεθεί ποιός έχει αντίγραφο
  - ρωτώντας όλους (broadcast)
  - κρατώντας κατάλογο (directory)
- Για κάθε γραμμή, ο κατάλογος περιέχει πληροφορίες:
  - για την κατάσταση της γραμμής
  - για τις θέσεις των αντιγράφων
- Λίγο αργότερα θα δούμε:
  - πού βρίσκεται ο κατάλογος;
  - πώς είναι οργανωμένος
  - τί ακριβώς πληροφορίες κρατάει και πώς διατηρείται ενημερωμένος

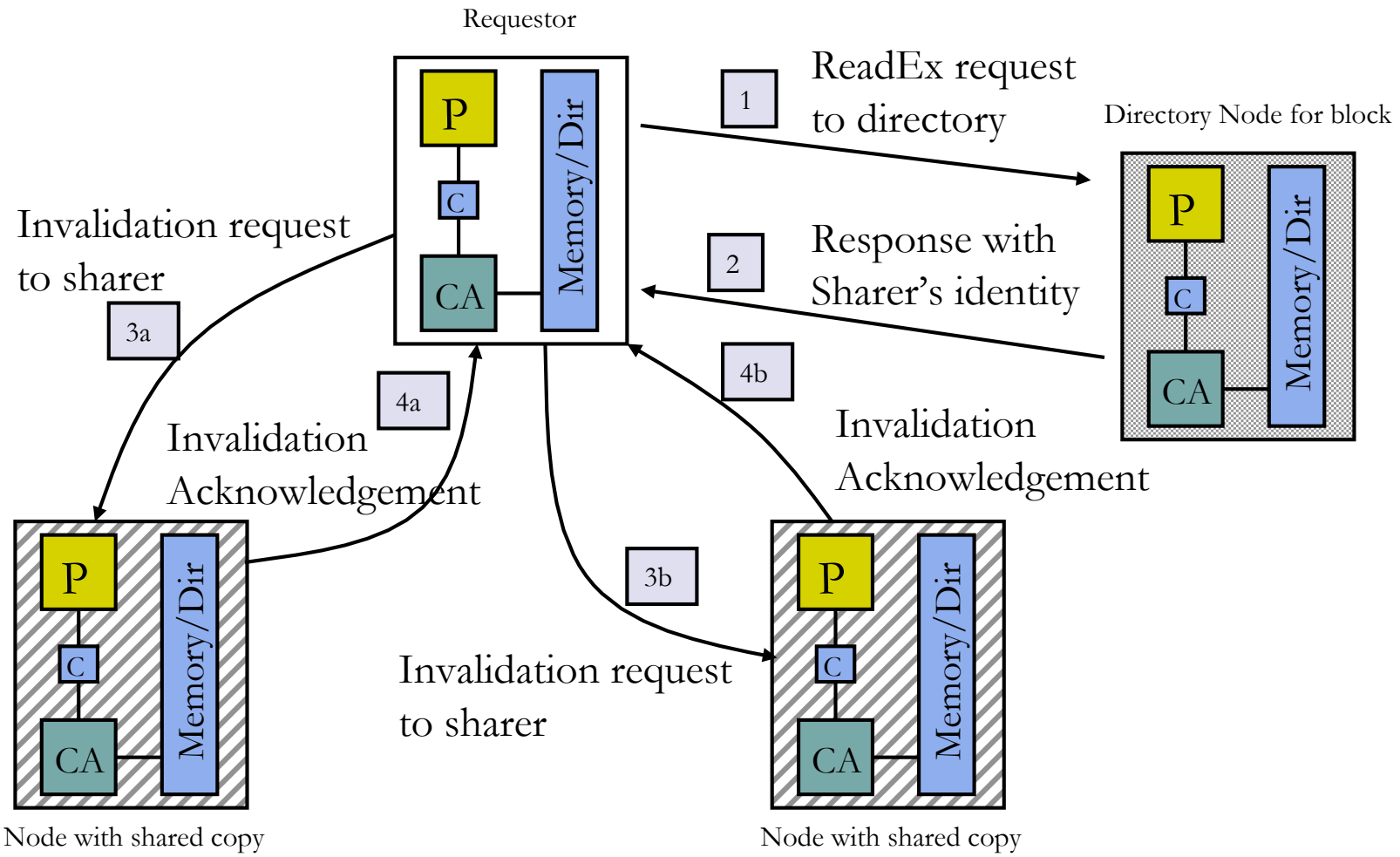
Δίκτυα: σειρά παράδοσης

- Το bus είναι ένα totally ordered δίκτυο
  - όλα τα μηνύματα παραδίδονται σε όλους τους προορισμούς με την ίδια σειρά
- Δίκτυα μεγάλων συστημάτων, γενικά δεν έχουν αυτή την ιδιότητα
  - point-to-point ordering - μόνο μηνύματα από τον ίδιο αποστολέα προς τον ίδιο παραλήπτη είναι σίγουρο ότι παραδίδονται σε σειρά
  - unordered – καμία εξασφάλιση για τη σειρά μετάδοσης
- Σε κάποιες περιπτώσεις του πρωτόκολλου συνοχής, η σειρά έχει σημασία

Read miss


Requestor
P
C
CA
Memory/Dir
1
Read request
to directory
Directory Node for block
P
C
CA
Memory/Dir
2
Response with
owner identity
3
Read request
to owner
4a
Data Reply
4b
Revision message
to directory (Data Reply)
P
C
CA
Memory/Dir
Node with dirty copy

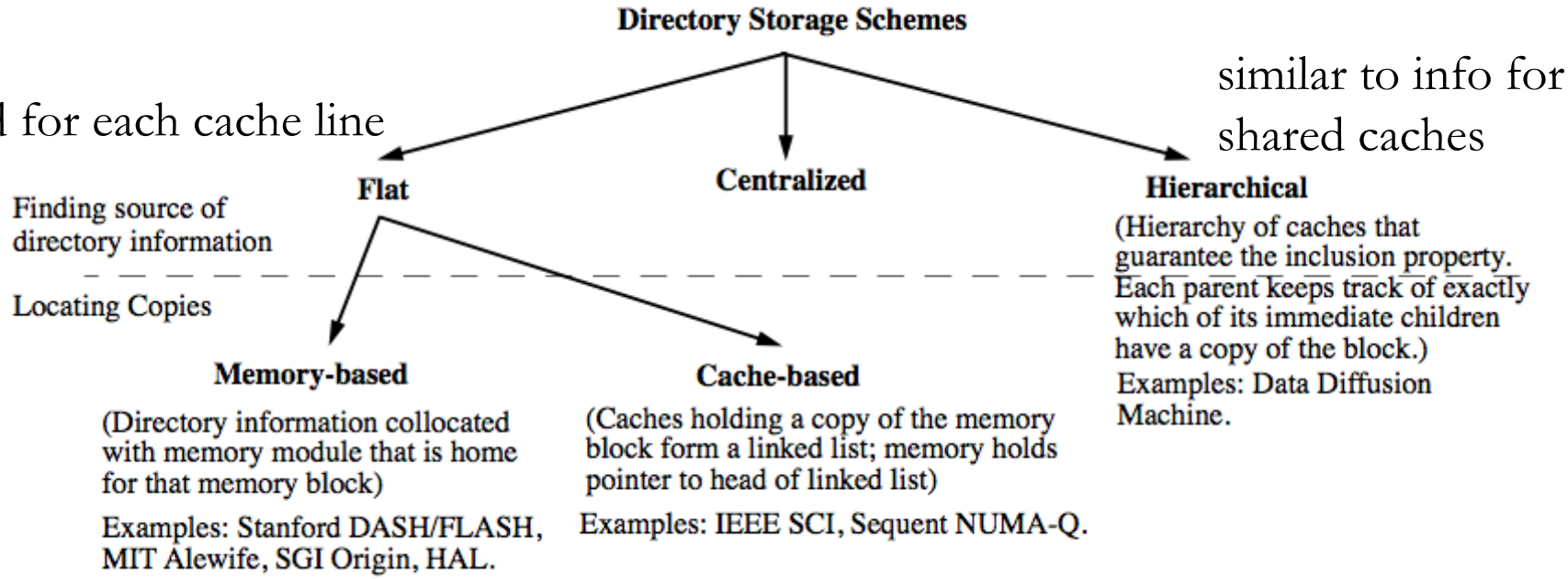
Write miss


Requestor
P
C
CA
Memory/Dir
1
ReadEx request
to directory
Directory Node for block
P
C
CA
Memory/Dir
2
Response with
Sharer's identity
Invalidation request
to sharer
3a
4a
Invalidation
Acknowledgement
4b
Invalidation
Acknowledgement
3b
Invalidation request
to sharer
P
C
CA
Memory/Dir
Node with shared copy
P
C
CA
Memory/Dir
Node with shared copy

Οργάνωση καταλόγων

fixed for each cache line

similar to info for shared caches



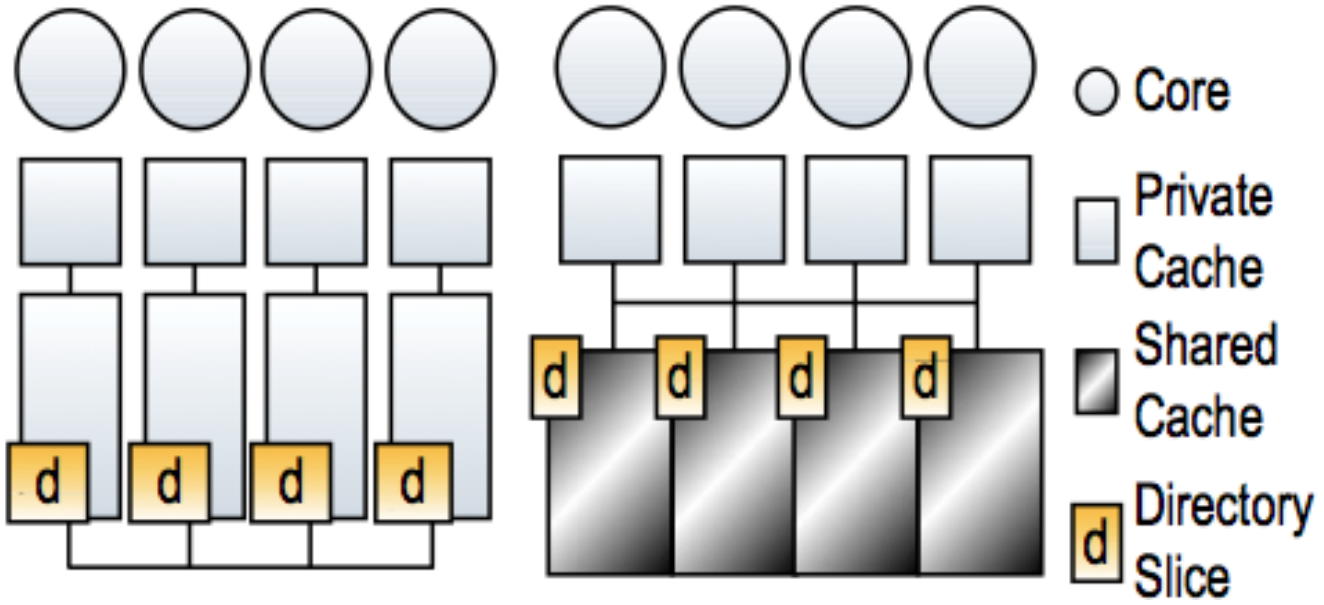
Οργάνωση καταλόγου

- Ο κατάλογος πρέπει να περιέχει καταχωρήσεις για όλες τις γραμμές του συστήματος;
  - μόνο γραμμές που βρίσκονται σε ιδιωτικές cache
  - μόνο κοινόχρηστες γραμμές (αν μπορούμε να τις γνωρίζουμε)
- Ο αριθμός κοινόχρηστων γραμμών δεν είναι γνωστός
  - πρέπει το σύστημα να υποστηρίζει τη χειρότερη περίπτωση: όλες οι γραμμές όλων των cache είναι κοινόχρηστες
- Ο πλήρης κατάλογος είναι πολύ μεγάλος
  - δεν είναι πρακτικό να βρίσκεται συγκεντρωμένος σε ένα σημείο
- Η πιο συνηθισμένη υλοποίηση είναι flat, memory based
  - μέρος της διεύθυνσης καθορίζει σε ποιό κομμάτι (slice) του καταλόγου βρίσκεται η αντίστοιχη καταχώρηση για τη γραμμή

«Υψος» καταλόγου

- Μικρότερος κατάλογος από τη χειρότερη περίπτωση;
- Μπορούμε να κάνουμε caching στον κατάλογο
  - παρόμοια με το πίνακα σελίδων του Λ.Σ. και το TLB
  - υπάρχει τοπικότητα στις αναφορές
- Χρειάζεται μόνο 1 επίπεδο ιεραρχίας
  - Αντικατάσταση καταχώρησης καταλόγου: αν υπάρχουν αντίγραφα της γραμμής στις caches των πυρήνων, πρέπει να ακυρωθούν (και αν είναι modified να γραφτούν στη μνήμη)
- Αλλα χαρακτηριστικά
  - δεν μοιράζονται οι καταχωρήσεις καταλόγου (η θέση εξαρτάται από τη διεύθυνση) – δεν υπάρχει θέμα coherence
  - η cache καταλόγου δέχεται αιτήματα από όλους τους πυρήνες

Οργάνωση Chip MultiProcessors



Source: Ferdman et al, Cuckoo directory

- Η μοιραζόνενη cache μπορεί να είναι
  - κατανεμημένη ανά κόμβο/ψηφίδα (tile)
  - σε χωριστά tiles ειδικές για cache
- Τα κομμάτια καταλόγου (slice) μπορεί να μη βρίσκονται σε κάθε tile

Συνδιασμός καταλόγου με LLC tags

- Για inclusive, κοινόχρηστες LLCs μπορεί να συνδιαστεί η πληροφορία καταλόγου με τα tags
  - λέγεται in-cache directory
- Φαίνεται η πιο λογική, ίσως, λύση
  - αφού η LLC πρέπει να προσπελαστεί, γιατί να μην περιέχει και την πληροφορία του καταλόγου
- Προβλήματα:
  - δεν είναι γενική λύση – δουλεύει μόνο για την παραπάνω κατηγορία
  - μπορεί να έχει μεγάλο κόστος: απαιτούνται επιπλέον bits σε κάθε γραμμή της LLC, και το σύνολο των θέσεων γραμμών που περιέχει συνήθως είναι πολύ περισσότερο από το άθροισμα των γραμμών όλων των LLC-1

Αντίγραφα LLC-1 tags

- Κάθε slice του καταλόγου περιέχει τα αντίγραφα των tags όλων των ιδιωτικών caches του προηγούμενου επιπέδου (LLC-1) που αντιστοιχούν στο slice
  - υποθέτουμε inclusive ιδιωτικές ιεραρχίες cache
- Η αντιστοίχιση γίνεται χρησιμοποιώντας κάποια από τα bit διεύθυνσης
  - π.χ. τα 2 τελευταία του index για 4 slices
  - η αντιστοίχιση σε κομμάτια της LLC cache, αν είναι κοινόχρηστη, μπορεί να είναι διαφορετική
- Η πληροφορία των θέσεων αντιγράφων εξορύσεται από τη σύγκριση με τα αντίγραφα όλων των tags

Αξιολόγηση duplicate tags dir

- Ο χώρος (αριθμός bits) που απαιτείται για τον κατάλογο είναι σχετικά μικρός
- Κάθε slice πρέπει να έχει μεγάλη associativity
  - number of processors x LLC-1 associativity
  - γιατί κάθε LLC-1 cache μπορεί να έχει διαφορετική γραμμή σε κάθε θέση του set του
- Η associativity εξαρτάται από τον αριθμό των πυρήνων
  - το ίδιο tile δεν μπορεί να χρησιμοποιηθεί σε διαφορετικά Ο.Κ.
- Μεγάλη associativity
  - υλοποιείται δύσκολα, με μεγάλο κόστος, μπορεί να είναι αργή η προσπέλαση
  - έχει μεγάλη κατανάλωση ενέργειας

Παράδειγμα κόστους καταλόγου

- Σύνθεση συστήματος:
  - 16 πυρήνες, 48b διευθύνσεις, 64 byte γραμμές cache και για τα δύο επίπεδα $
  - Κατανεμημένος κατάλογος σε 16 tiles
  - L1 $: 2-way, 64KB ανά πυρήνα
  - L2 $: κοινόχρηστη, κατανεμημένη, 16-way, 1MB ανά πυρήνα/tile
- Πόσο χώρο σε bits χρειάζεται ένα slice καταλόγου για τις μεθόδους in-cache, duplicate-tag;

Σποραδικοί (sparse) κατάλογοι

- Χρησιμοποιούν λίγα από τα bits που αντιστοιχούν στο τμήμα διεύθυνσης του tag της γραμμής για index
  - μικραίνει η απαιτούμενη associativity
  - Χάνεται η 1-προς-1 αντιστοιχία μεταξύ γραμμών cache και καταχωρήσεων καταλόγου
  - Χρειάζεται ρητή αποθήκευση της πληροφορίας θέσεων αντιγράφων
- Η ανομοιόμορφη κατανομή των καταχωρήσεων καταλόγου προκαλεί συγκρούσεις
  - αντικατάσταση χρήσιμων καταχωρήσεων προκαλεί ακύρωση γραμμών cache χωρίς να απαιτείται από το πρόγραμμα
- Περισσότερες θέσεις (overprovisioning) στον κατάλογο για να περιοριστούν οι συγκρούσεις

Πληροφορίες καταλόγου

Κατάσταση γραμμής: π.χ. exclusive/modified, shared, invalid, …

Θέσεις αντιγράφων. Επιλογές:

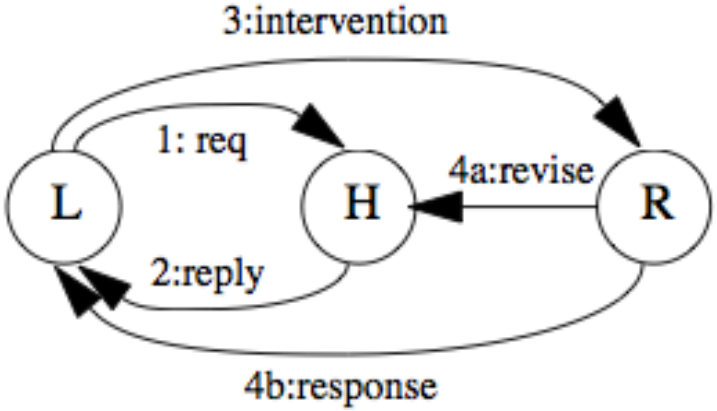
- αντίγραφα tags
- διάνυσμα: ένα bit ανά πυρήνα
- περιοριμένοι δείκτες: ένας αριθμός από $\log_2(P)$ bit δείκτες στους αριθμούς/κωδικούς πυρήνων που έχουν αντίγραφα
  - χρειάζεται τρόπο χειρισμού περισότερων αντιγράφων

Κλιμάκωση (scaling) καταλόγου

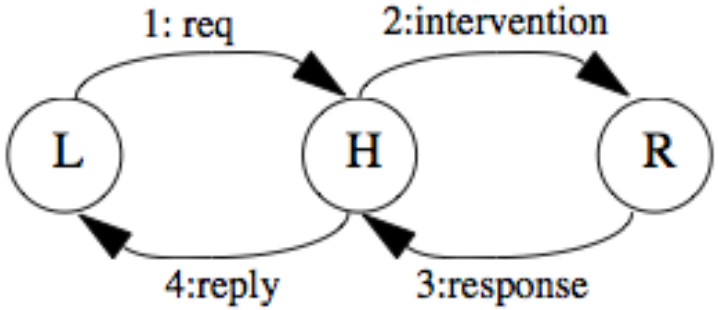
- Ο κατάλογος πρέπει να κλιμακώνεται σύμφωνα με τις διαστάσεις του συστήματος
  - αριθμός πυρήνων
- Από άποψη υλικού, πόσο χώρο σε bits καταλαμβάνει ο κατάλογος
  - σχετικά λίγοι πυρήνες μοιράζονται σε κάθε στιγμή δεδομένα
  - μεγαλύτερες γραμμές cache απαιτούν λιγότερες καταχωρήσεις καταλόγου
  - ή πιο χονδρική πληροφορία ανά καταχώρηση (συνδιασμός πληροφορίας για πολλές γραμμές)
- Από άποψη ταχύτητας
  - πόσα μηνύματα χρειάζονται για κάθε miss;
  - πόσα από αυτά είναι στο κρίσιμο μονοπάτι;

Βελτιώσεις πρωτόκολλου

- Αρχική σχεδίαση (strict request-reply)
  - 5 μηνύματα, 4 στο critical path

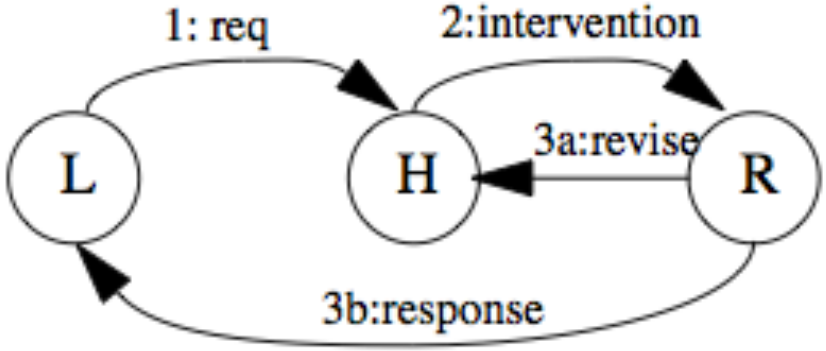


- Προώθηση παρέμβασης (intervention forwarding)
  - 4 μηνύματα, όλα στο critical path



Προώθηση απάντησης

- Reply – forwarding
  - 4 μηνύματα, 3 στο critical path
  - 3-message miss



Ορθότητα πρωτοκόλλου

- Σωστές αλλαγές καταστάσεων, ακύρωση, αντικατάσταση γραμμών
  - αυτονόητο, αλλά δύσκολο λόγω πολυπλοκότητας
- Διατήρηση ορθής σειράς πράξεων μνήμης σύμφωνα με τα πρωτόκολλα συνοχής και συνέπειας
- Εξασφάλιση από deadlock, livelock, starvation

Δύο κύρια προβλήματα:

- δεν υπάρχει μία οντότητα που «βλέπει» όλες τις πράξεις για να τις βάλει σε σειρά
- πολλά μηνύματα μπορεί να κατευθύνονται προς ένα κόμβο

Προβλήματα σειράς εγγραφής

- Απαίτηση του coherence: Ολοι οι πυρήνες πρέπει να «βλέπουν» εγγραφές σε μία γραμμή με την ίδια σειρά
  - το home directory μπορεί να εξασφαλίσει τη σειρά;

Παράδειγμα: γραμμή modified, δύο διαφορετικοί πυρήνες ζητούν read-exclusive

- το home απαντάει με τον κόμβο που έχει τη γραμμή
- η σειρά των αιτήσεων στον κόμβο δεν είναι σίγουρο ότι θα είναι η ίδια με τη σειρά αιτήσεων στον κάτοχο
  - ακόμα και αν το δίκτυο διατηρεί σειρά point-to-point

Σειρά με busy states

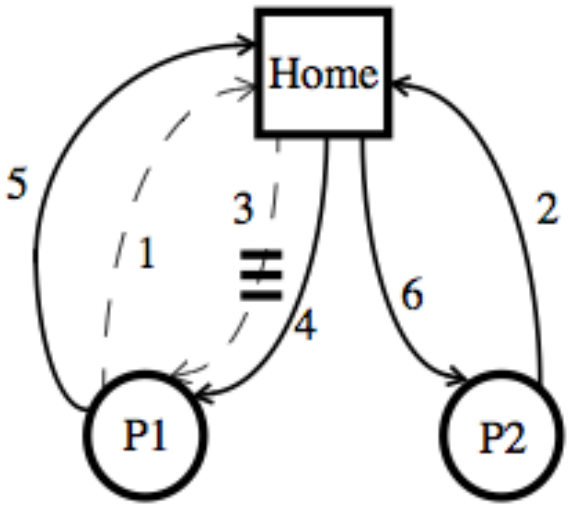
Μέχρι να ολοκληρωθεί μία πράξη η γραμμή είναι σε κατάσταση busy/pending

Νέα αιτήματα που φτάνουν στο home

- απορίπτονται (Nack) και πρέπει να ξαναγίνουν
- περιμένουν, buffer at home
  - Το home πρέπει να ενημερώνεται για την ολοκλήρωση της πράξης
  - χρειάζεται μεγάλο χώρο αποθήκευσης
- προωθούνται στον κάτοχο
  - η σειρά καθορίζεται είται από το home είτε από τον κάτοχο
  - αν ο κάτοχος αλλάξει, ο προηγούμενος απορίπτει αιτήματα

Το πρόβλημα παραμένει!

- Δεν αρκεί η σειρά που καθορίζει το home ή ο κάτοχος
  - ξέρουν πότε το δικό τους μέρος της δουλειάς τελείωσε, αλλά όχι πότε όλοι οι πυρήνες έχουν ενημερωθεί



1. P1 read,
2. P2 read-exclusive. περιμένει το (1)
3. Reply to P1 (καθυστερεί στο δίκτυο)
4. ακύρωση λόγω (2). ξεπερνάει το 3
5. Επιβεβαίωση ακύρωσης
6. απάντηση στο (2)

το (3) φτάνει στο P1 και γράφει πάνω από την ακυρωμένη γραμμή

Απαιτήσεις σειράς εγγραφής

- Local serialisation: Αν υπάρχει (τοπικό) αίτημα για μια γραμμή σε εξέλιξη, δεν πρέπει να επιτραπεί καμία πρόσβαση στη γραμμή μέχρι να ολοκληρωθεί το αρχικό αίτημα
- Προσπελάσεις σε άλλες γραμμές πρέπει να επιτρέπονται
  - αλλιώς υπάρχει περίπτωση deadlock

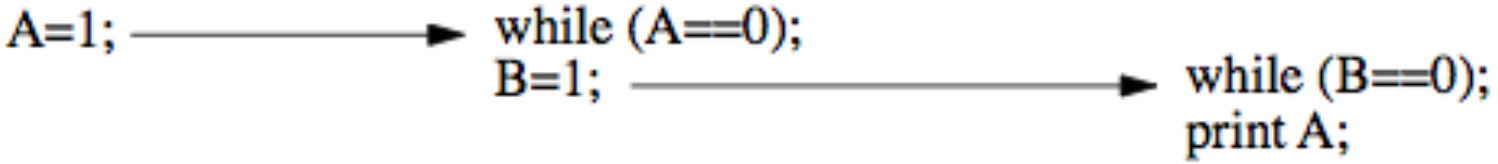
Επιπτώσεις μοντέλου consistency

- Είδαμε ότι το μοντέλο sequential consistency είναι πολύ περιοριστικό
  - αλλά κάποιες φορές θέλουμε να το επιβάλουμε προσωρινά χρησιμοποιώντας memory fences
- Τι χρειάζεται από ένα σύστημα καταλόγου ένα memory-fence
  - πότε έχει ολοκληρωθεί μια εγγραφή;
- Είδαμε ότι οι εντολές εγγραφής παίρνουν επιβεβαίωση από το σύστημα μνήμης
- Ενα memfence περιμένει την επιβεβαίωση πριν επιτρέψει άλλες προσπελάσεις μνήμης

write completion

- Μια εγγραφή δεν ολοκληρώθηκε όταν φτάνει στο home
  - Εστω δύο εγγραφές από τον ίδιο πυρήνα σε X, Y (με αυτή τη σειρά)
  - Αν το home (του X) επιβεβαιώσει την εγγραφή X όταν την παραλάβει δεν είναι σίγουρο ότι όλα τα αντίγραφα έχουν ακυρωθεί
  - Η επόμενη εγγραφή στο Y μπορεί να εμφανιστεί σε άλλους πυρήνες πρίν από αυτή στο X
- Η εγγραφή ολοκληρώνεται όταν οι επιβεβαιώσεις από όλα τα αντίγραφα παραδοθούν

write atomicity



- Πρόβλημα: ο P2 μπορεί να γράψει το B και η νέα τιμή να φτάσει τον P3, πριν η νέα τιμή του A φτάσει στον P3
- Ο κάτοχος του A (P1 cache) πρέπει να μην επιτρέπει αναγνώσεις της γραμμής μέχρι να έχουν έρθει όλες οι επιβεβαιώσεις ακύρωσης από τα αντίγραφα

Αλλα προβλήματα

- Deadlock
  - συνήθως πρόβλημα με χώρο σε buffers
- Livelock
  - σπάνιο φαινόμενο
- Οι λεπτομέρειες είναι σημαντικές
  - π.χ. έξώση (αντικατάσταση) γραμμής ταυτόχρονα με πρόσβαση σε αυτή από άλλο πυρήνα
  - δείτε τον κώδικα του sniper, που είναι αρκετά απλουστευμένος ήδη!

Αξιολόγηση επιδόσεων

- Οι κατάλογοι δουλεύουν καλά αν ο αριθμός των ακυρώσεων είναι μικρός για κάθε εγγραφή
  - και συμβαίνουν αρκετά συχνά ώστε η εκπομπή να είναι ασύμφορη

Ορισμοί:

- invalidating writes – εγγραφές που απαιτούν ακυρώσεις άλλων αντιγράφων
  - το τοπικό αντίγραφο δεν ακυρώνεται
- Συχνότητα ακυρώσεων (invalidation frequency)
- Κατανομή αριθμού ακυρώσεων (invalidation size distribution)

Μοτίβα μοιράσματος δεδομένων

- Μόνο ανάγνωση (read-only)
  - δεν υπάρχουν invalidating writes
- Παραγωγός-καταναλωτής (producer-consumer)
  - Ενας πυρήνας γράφει, μετά ένας ή περισσότεροι διαβάζουν
  - invalidation size είναι συχνά 1, όλοι ή αρκετοί
- Αποδημητικά (migratory)
  - ένας πυρήνας γράφει και μετά διαβάζει τα δεδομένα
  - invalidation size είναι 1, ακύρωση του προηγούμενου
- Ακανόνιστα (irregular)
  - π.χ. task queues
  - invalidation size συνήθως είναι μικρό

Αποτελέσματα