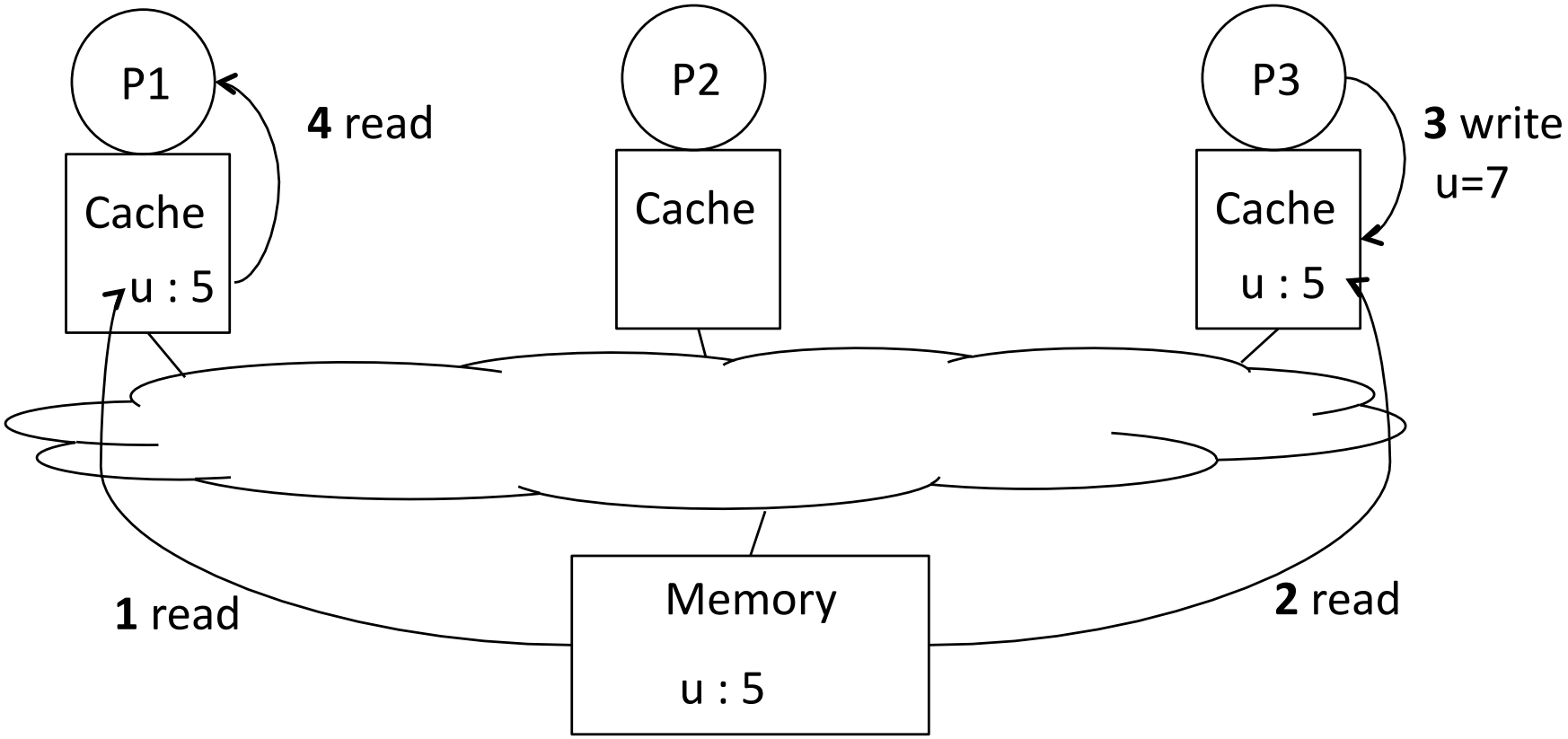
Υ-07 – Παράλληλα Συστήματα
Συνέπεια και συνοχή μνήμης

Αρης Ευθυμίου

Λειτουργία μνήμης

- Η μνήμη είναι ένας πίνακας αποθήκευσης
- Οταν διαβάζουμε μια θέση, περιμένουμε να πάρουμε την **τελευταία** τιμή που έχει γραφτεί εκεί
- Με δεδομένο ότι οι κρυφές μνήμες είναι απαραίτητες για να έχουμε μια αξιοπρεπή ταχυτητα εκτέλεσης, το παραπάνω γίνεται δύσκολο να υλοποιηθεί σε ένα παράλληλο σύστημα
- Ενα σειριακό σύστημα δεν έχει πρόβλημα γιατί όλες οι προσπελάσεις μνήμης γίνονται στην ίδια ιεραρχία μνήμης

Παράδειγμα



Το σύστημα μνήμης δεν έχει συνοχή (coherence)!

Πρόβλημα συνοχής

- Οι κρυφές μνήμες δημιουργούν το πρόβλημα
  - και store buffers, write buffers
- Μπορούμε να απαγορέψουμε σε μοιραζόμενες μεταβλητές να κρατουνται στις κρυφές μνήμες
  - Πολύ αργό και πολύ δουλειά για το προγραμματιστή
- Είναι πολύ πιο βολικό να αφεθεί στο υλικό η απαίτηση να κρατάει τις κρυφές μνήμες σε συνοχή

- Το πρόβλημα εμφανίζεται και σε σειριακούς υπολογιστές όταν και ο επεξεργαστής και περιφερειακά γράφουν στη μνήμη

Ορισμός προβλήματος

- Η ένοια της τελευταίας τιμής που γράφεται στη μνήμη δεν ειναι αρκετά καθαρή
- Τι γίνεται αν μια τιμή γράφεται ταυτόχρονα από πολλούς πυρήνες ή οι εγγραφές συμβαίνουν τόσο κοντά στο χρόνο που δεν υπάρχει χρόνος να μεταβιβαστεί η πρώτη πριν γίνει η δεύτερη;
- Δεν μπορούμε να βρούμε την ακριβή χρονολογική σειρά
  - μας ενδιαφέρει η σειρά σύμφωνα με το πρόγραμμα
- Το πρόβλημα είναι ότι δεν υπάρχει μία (καθ)ολική σειρά εντολών σε ένα παράλληλο πρόγραμμα
  - και αν την επιβάλαμε θα γινόταν σειριακό!

Πότε γίνεται μια προσπέλαση;

- Πότε καταλαβαίνει ένας πυρήνας ότι έγινε μια πράξη μνήμης;
  - Δεν μας ενδιαφέρει τι πραγματικά γίνεται στη μνήμη, αλλά πώς αντιλαμβάνεται ο πυρήνας τις αλλαγές
- Μια εγγραφή από τον πυρήνα φαίνεται ότι έγινε αν μια επόμενη ανάγνωση από τον ίδιο φέρνει την τιμή που γράφτηκε ή μια επόμενη τιμή
- Μια ανάγνωση από τον πυρήνα φαίνεται ότι έγινε, όταν εγγραφές που γίνονται αργότερα από τον ίδιο δεν μπορούν να αλλάξουν την τιμή που επιστρέφεται
- Παράλληλο σύστημα: αλλάζουμε τα «από τον ίδιο» σε «από οποιονδήποτε»

Συνοχή

1. Εγγραφές σε κάθε θέση μνήμης πρέπει να γίνονται ορατές με την ίδια σειρά απ' όλα τα νήματα
   - ονομάζεται write serialisation
2. Μια πράξη εγγραφής πρέπει να γίνεται ορατή από όλους τους πυρήνες
   - μπορεί να αργήσει όσο θέλουμε αρκεί να μην επηρεάζει την πρώτη ιδιότητα

- Αν ένας πυρήνας διαβάσει μια τιμή που έγραψε μια εντολή w1 πρίν τη τιμή που γράφει μια εντολή w2, τότε και οι άλλοι πυρήνες πρέπει να μήν μπορούν να διαβάσουν την τιμή της w2 πρίν τη w1

Υλοποίηση συνοχής από το υλικό

Δύο τρόποι: όταν γίνεται μια εγγραφή, είτε

- Ακύρωση (invalidation) όλων των αντιγράφων σε κρυφές μνήμες άλλων πυρήνων
- Ενημέρωση (update) τιμών όλων των αντιγράφων

Προβλήματα:

- write serialisation – πως δύο «ταυτόχρονες» εγγραφές γίνονται σε σειρά
- Πολλές λεπτομέρειες υλοποίησης περιπλέκουν την υλοποίηση

Θα επιστρέψουμε σε αυτό το θέμα σύντομα

Συνέπεια μνήμης (consistency)

- Λύνοντας το θέμα της συνοχής δεν είναι αρκετό
- Συχνά ένα παράλληλο πρόγραμμα κάνει υποθέσεις για τη σειρά εκτέλεσης πράξεων μνήμης σε **διαφορετικές** διευθύνσεις

Παράδειγμα (αρχικά a=flag=0):

Processor 1	Processor 2
`a = func();`	`while (flag==0);`
`flag = 1;`	`print a;`

Υποθέτουμε ότι αν ο 2 διαβάσει το flag ως 1, θα διαβάσει επίσης την καινούρια τιμή του a

Η συνοχή δεν το εξασφαλίζει αυτό!

Είναι πράγματι πρόβλημα;

- Μήπως δε γράψαμε καλά τον κώδικα;
  - έπρεπε να βάλουμε κάποιο σαφή συνχρονισμό μεταξύ παραγωγού – καταναλωτή;

Processor 1	Processor 2
a = func();	
Barrier();	Barrier();
	print a;

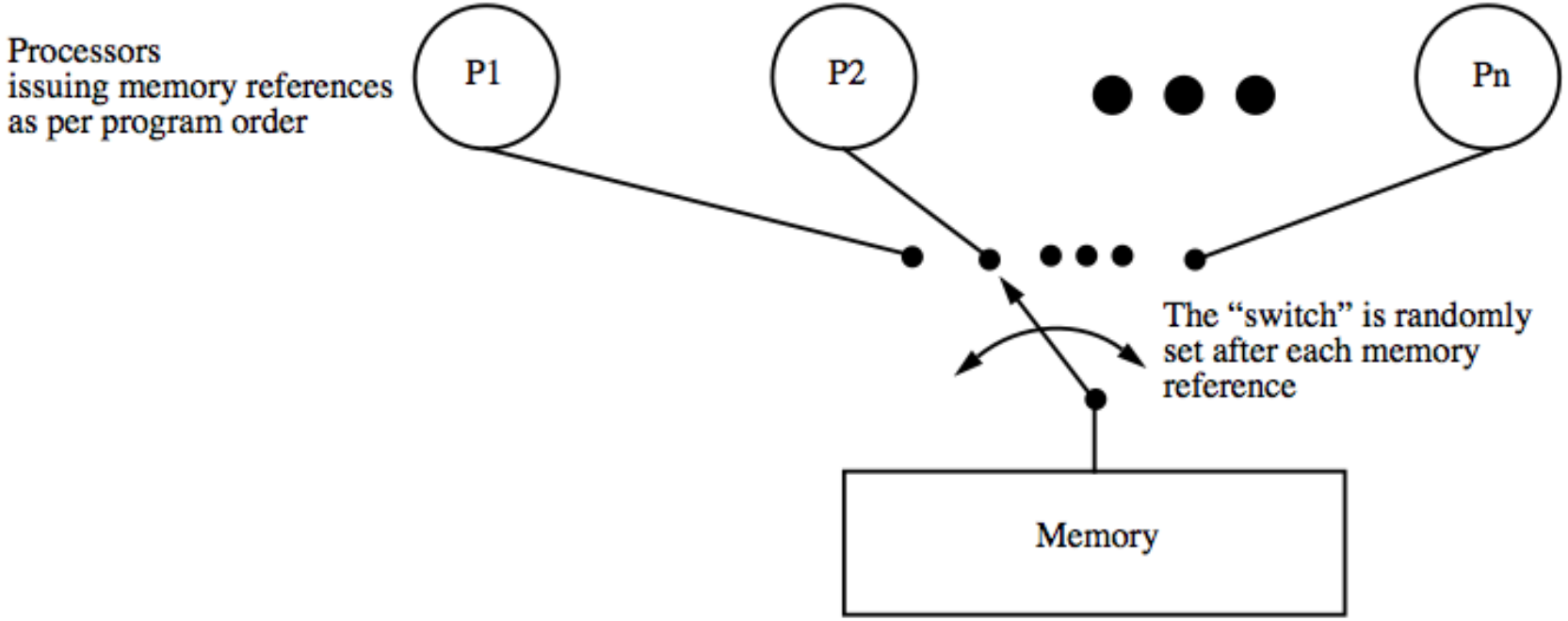
Ακόμα υπάρχουν προβλήματα:

- Η ρουτίνα Barrier δεν εγγυάται ότι εγγραφές πριν το Barrier θα έχουν φτάσει σε όλους μέχρι να βγούν από αυτό
- Η ρουτίνα Barrier χρησιμοποιεί εντολές προσπέλασης μνήμης σε κοινές μεταβλητές

Συνέπεια μνήμης

- Δεν αρκεί ότι η μνήμη επιστρέφει την τελευταία τιμή για **μία** θέση, χρειαζόμαστε κάποιες φορές την ιδιότητα οι εγγραφές και αναγνώσεις σε **διαφορετικές** θέσεις να γίνονται με τη κατάλληλη σειρά.
- Το μοντέλο συνέπειας μνήμης θέτει περιορισμούς ως προς τη σειρά με την οποία φαίνονται (στους πυρήνες) να γίνονται οι προσπελάσεις μνήμης
  - Είναι υπερσύνολο της συνοχής μνήμης

Ακολουθιακή συνέπεια



Κάθε πυρήνας εκτελεί προσπελάσεις σύμφωνα με τη σειρά προγράμματος

Κάθε προσπέλαση γίνεται αδιαίρετα (atomically): ολοκληρώνεται πρίν ξεκινήσει η επόμενη

Παράδειγμα

αρχικά a=b=0

Processor 1	Processor 2
1a: a = 1;	*2a*: print b;
1b: b = 2;	*2b*: print a;

Ποιά είναι τα όλα τα επιτρεπόμενα αποτελέσματα (b,a) υποθέτοντας SC;

- (0,0), (2,1), (0,1)

Ποιές είναι όλες οι δυνατές ακολουθίες εντολών;

1a, 1b, 2a, 2b: (1,2)	2a, 2b, 1a, 1b: (0,0)
1a, 1b, 2b, 2a: (1,2)	2a, 2b, 1b, 1a: (0,0)
1a, 2a, 1b, 2b: (1,0)	2a, 1a, 2b, 1b: (1,0)
1a, 2a, 2b, 1b: (1,0)	2a, 1a, 1b, 2b: (1,0)
1a, 2b, 1b, 2a: (1,2)	2a, 1b, 2b, 1a: (0,0)
1a, 2b, 2a, 1b: (1,0)	2a, 1b, 1a, 2b: (1,0)
1b, 1a, 2a, 2b: (1,2)	2b, 2a, 1a, 1b: (0,0)
1b, 1a, 2b, 2a: (1,2)	2b, 2a, 1b, 1a: (0,0)
1b, 2a, 1a, 2b: (1,2)	2b, 1a, 2a, 1b: (0,0)
1b, 2a, 2b, 1a: **(0,2)**	2b, 1a, 1b, 2a: **(0,2)**
1b, 2b, 1a, 2a: **(0,2)**	2b, 1b, 2a, 1a: **(0,2)**
1b, 2b, 2a, 1a: **(0,2)**	2b, 1b, 1a, 2a: **(0,2)**

Παράδειγμα 2

P1	P2	P3
`a = 1;`	`while (a==0);`	`while (b==0);`
	`b = 1;`	`print a;`

- Αφού ο P2 περιμένει την τιμή του a πριν γράψει το b και ο P3 περιμένει την τιμή του b, είναι λογικό να περιμένουμε ότι ο P3 θα δεί την καινούρια τιμή του a
- Τι μπορεί να πάει στραβά;
- Αν ο P2 αλλάξει το b, πριν η αλλαγή του a γίνει γνωστή στον P3, …
- Πώς μπορεί να γίνει κάτι τέτοιο;

Αδιαίρετες εγγραφές

- write atomicity
- Η σειρά με την οποία μια εγγραφή γίνεται (σε σχέση με την υποθετική σειρά όλων των προσπελάσεων όλων των νημάτων) πρέπει να φαίνεται η ίδια σε όλους τους πυρήνες
- Αν κάποιος πυρήνας πάρει μια τιμή που γράφτηκε, τίποτα απ'ότι κάνει μετά δεν πρέπει να εμφανιστεί πριν όλοι οι άλλοι πυρήνες δουν την εγγραφή

Ικανές συνθήκες

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

Συνέπειες για το υλικό

Ενα παράλληλο σύστημα που εγγυάται την ακολουθιακή συνέπεια δεν μπορεί να χρησιμοποιεί μια σειρά από αρχιτεκτονικές τεχνικές αύξησης απόδοσης:

- Εκτέλεση εκτός σειράς
- store buffers
- non-blocking caches
- και βελτιστοποιήσεις του μεταφραστή

Οι πυρήνες των παράλληλων μηχανών θα είναι πολύ αργοί!

Μεγάλο μέρος του κέρδους από τον παραλληλισμό, χάνεται

Χαλαρή συνέπεια μνήμης

- Γι'αυτό τα περισσότερα παράλληλα συστήματα δεν υποστηρίζουν ακολουθιακή συνέπεια αλλά πιο χαλαρά μοντέλα συνέπειας μνήμης (relaxed mem consistency)
  - επιτρέπουν αλλαγές σειράς ή επικαλυπτόμενη εκτέλεση εντολών
  - π.χ. total Store order (TSO) - μια read μπορεί να εκτελεστεί πριν από μια προηγούμενη, σύμφωνα με το πρόγραμμα, write
- Για να μπορεί ο προγραμματιστής να εξασφαλίσει τη σειρά εκτέλεσης υπάρχουν εντολές «φράκτες μνήμης» (fence): οι εντολές πριν το φράκτη πρέπει να ολοκληρωθούν πριν ξεκινήσουν οι εντολές μετά

Συνέπειες για προγραμματισμό

- Πολύ λίγοι προγραμματιστές στηρίζονται σε SC
  - οι περισσότεροι χρησιμοποιούν υψηλού επιπέδου συγχρονισμό
- Ενα καλά συγχρονισμένο πρόγραμμα προστατεύει τις μοιραζόμενες μεταβλητές του ώστε να μην υπάρχει ανταγωνισμός όταν ανανεώνονται οι τιμές τους
  - υπάρχει όμως ανταγωνισμός για τις μεταβλητές συγχρονισμού
- Με αυτές τις συνθήκες, η αλλαγή της σειράς εκτέλεσης εντολών δεν επηρεάζει το παράλληλο πρόγραμμα, αρκεί να εξασφαλίσουμε ότι κατά την είσοδο ή έξοδο από ένα κλείδωμα, οι τιμές έχουν ανανεωθεί παντού στη μνήμη

Παράδειγμα

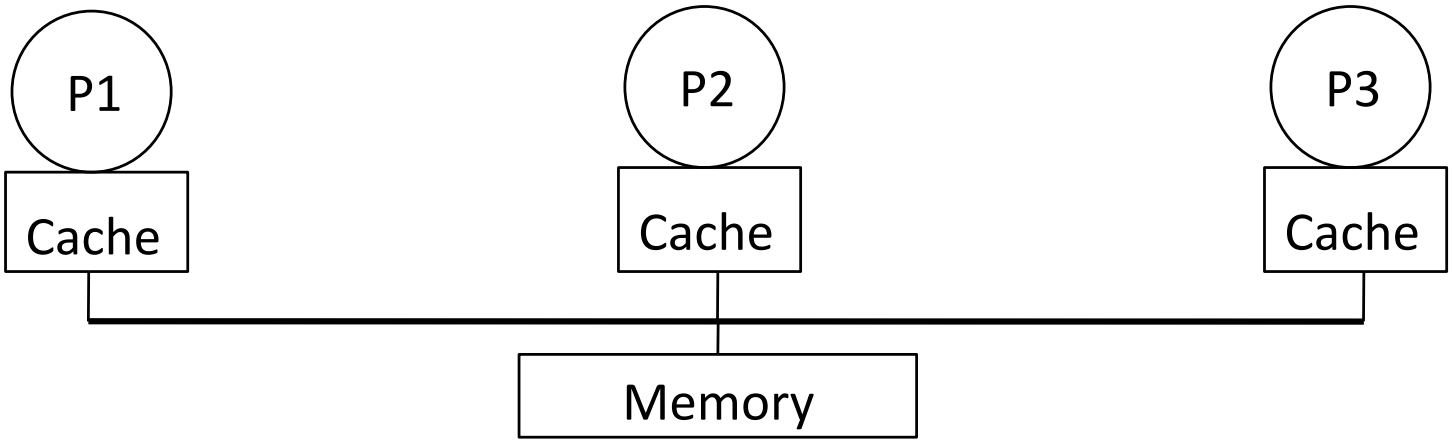
όλες οι επόμενες αναγνώσεις διαβάζουν τις τελευταίες τιμές

```
....
lock();
read and modify shared data
unlock();
....
```

όλες οι εγγραφές έχουν ολοκληρωθεί

- Ενα σειριακό σύστημα δεν χρειάζεται να γνωρίζει πότε ολοκληρώθηκε μια εγγραφή
- Ενα παράλληλο σύστημα, ακόμα και με ένα χαλαρό μοντέλο συνοχής, πρέπει να παίρνει αυτή τη πληροφορία
  - επιβεβαίωση (acknowledgement) εγγραφής
- όταν εκτελείται μια εγγραφή δεν γνωρίζουμε αν ακολουθεί μια fence οπότε η επιβεβαίωση είναι απαραίτητη πάντα

Κατασκοπευτικά(snoopy) πρωτόκολλα συνοχής



- Τα πρώτα παράλληλα συστήματα χρησιμοποιούσαν για δίκτυο ένα κοινό δίαυλο (bus)
  - Τα μηνύματα είναι ορατά σε όλους τους επεξεργαστές που συνδέονται στο δίαυλο
- Αυτή η ιδιότητα μπορεί να εκμεταλευτεί για πρωτόκολλα συνοχής
  - κάθε απαραίτητη ενέργεια πρέπει να φαίνεται στο δίαυλο
  - κάθε cache κατασκοπεύει το δίαυλο και κάνει τις απαραίτητες ενέργειες για cache blocks που έχει

Snoopy πρωτόκολλα

- Κάθε cache πλεον εκτός από τις εντολές που δέχεται από τον πυρήνα (ανάγνωση, εγγραφή),
- δέχεται και «εντολές» από το δίαυλο:
  - κάποιος διαβάζει μια γραμμή
  - κάποιος (θέλει να) γράφει μια γραμμή
- Ο ελεγκτής χρειάζεται να ψάχνει στην cache και για τους δύο αιτούμενους
  - χρειάζεται διπλή πόρτα πρόσβασης στα tags
    - όχι για τα δεδομένα, λόγω μεγάλου κόστους
  - μερικές φορές κάποιος πρέπει να περιμένει...

Πρωτόκολλο MSI - μηνύματα

- Θεωρούμε 1 επίπεδο κρυφής μνήμης (ανά πυρήνα)
  - write back, write-allocate (write-through είναι ευκολότερο)
- Ανάγνωση, cache hit:
  - η τιμή επιστρέφεται στον πυρήνα
- Εγγραφή, cache hit:
  - αν η γραμμή είναι dirty, το αντίγραφό μας είναι αποκλειστικό, μπορούμε να το αλλάξουμε χωρίς να ειδοποιήσουμε κανέναν
  - Αλλιώς, παρόλο που έχουμε τη γραμμή, πρέπει να ειδοποιήσουμε τις υπόλοιπες caches ότι σκοπεύουμε να το αλλάξουμε – write miss

Πρωτόκολλο MSI – μηνύματα

- Ανάγνωση, cache miss:
  - Διαδικασία ανάγνωσης στο δίαυλο
  - Αν κάποια άλλη cache έχει τη γραμμή (αλλαγμένη – dirty), πρέπει να απαντήσει δίνοντας τη γραμμή. Πρέπει επίσης να σημειώσει ότι η γραμμή δεν είναι πια αποκλειστικά δική του...
  - Αν δεν απαντήσει καποια cache, η κύρια μνήμη απαντάει με τα δεδομένα
- Εγγραφή, cache miss:
  - Διαδικασία readExclusive στο δίαυλο
  - Ολες οι cache που έχουν αντίγραφο, πρέπει να το ακυρώσουν (Invalidate)
  - αν μια cache έχει τη γραμμή αλλαγμένη, απαντάει με τα δεδομένα (και ακυρώνει το αντίγραφό της), αλλιώς απαντάει η κύρια μνήμη

Αντικατάσταση γραμμών

- Οταν η επιλεγμένη γραμμή είναι «καθαρή», δεν χρειάζεται καμιά ενέργεια
  - silent drop
- Αν έχει τροποποιημένα δεδομένα (dirty), διαδικασία επανεγραφής (write back)
  - αφορά μόνο τη κύρια μνήμη
  - αφού είναι dirty, καμιά άλλη cache δεν έχει αντίγραφο

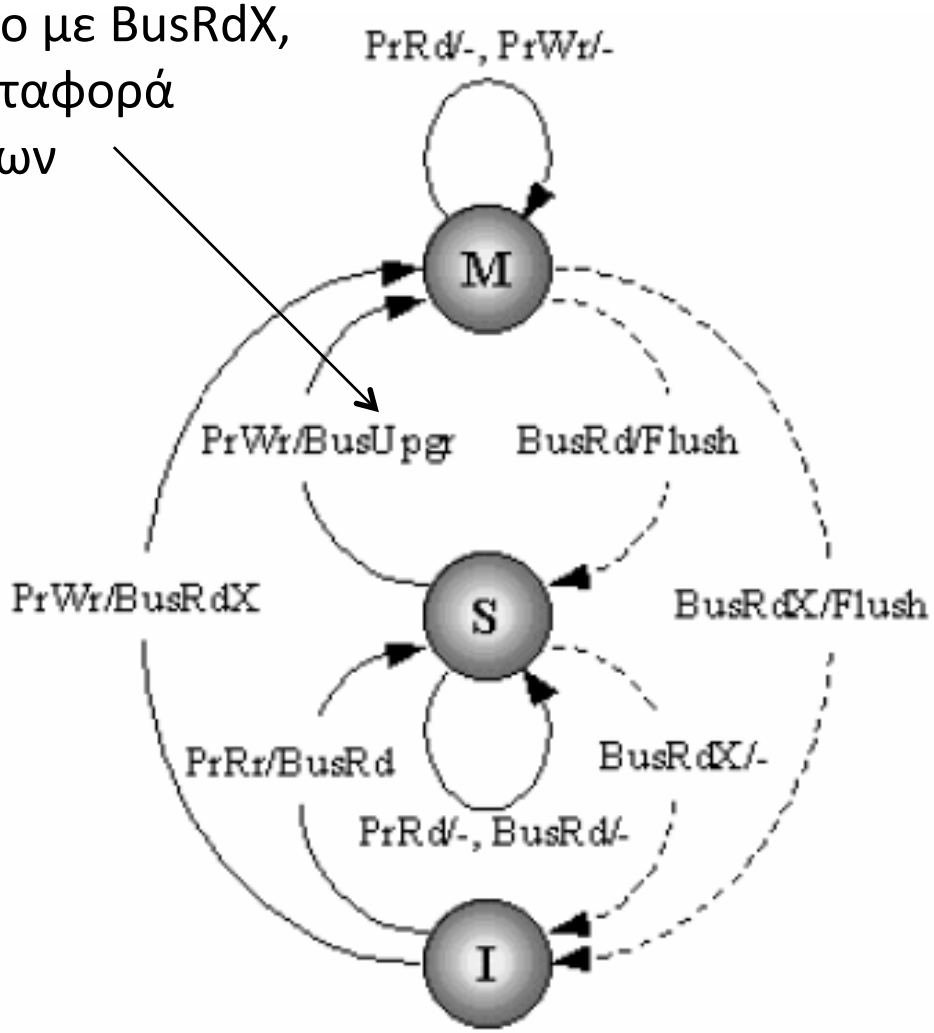
Καταστάσεις γραμμής

- Ακυρη (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, επομένος η κύρια μνήμη έχει ενήμερο αντίγραφο
  - Μπορεί να υπάρχουν άλλα αντίγραφα σε άλλες caches
  - Μπορεί και όχι γιατί οι αντικαταστάσεις γίνονται «αθόρυβα»
- Αλλαγμένη (Modified)
  - Εχουμε το μοναδικό ενήμερο αντίγραφο της γραμμής

Διάγραμμα καταστάσεων

Εντολή/πράξη

- PrRd, PrWr – ανάγνωση, εγγραφή από τον πυρήνα
- BusRd, BusRdX – διαδικασία ανάγνωσης, αποκλειστικής ανάγνωσης στο δίαυλο
- Flush – δίνω την τιμή της γραμμής στο δίαυλο και ανανεώνεται η κύρια μνήμη



Πρωτόκολλο MESI

- Αν διαβάσουμε μια καινούρια γραμμή και αμέσως μετά θέλουμε να την αλλάξουμε, χωρίς να (θέλει να) παραμβληθεί άλλος επεξεργαστής
- Εχουμε 2 misses
  - 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
- Λύση: κρατάμε πληροφορία αν έχουμε (διαβάσει) το μοναδικό αντίγραφο της γραμμής μέχρι τώρα
  - αποκλειστική (exclusive) κατάσταση

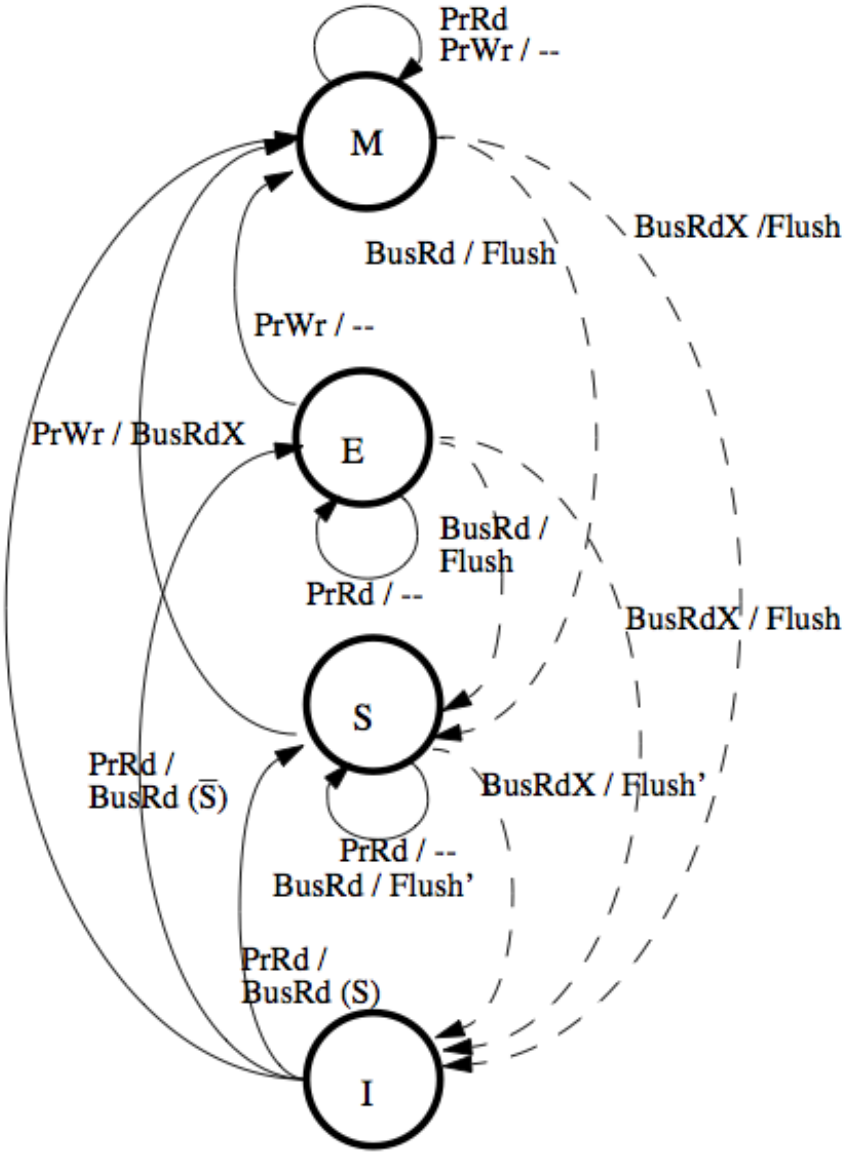
Διάγραμμα καταστάσεων

Σήμα S στον δίαυλο:

Αν μια cache έχει αντίγραφο της γραμμής θέτει το σήμα

Αν κανείς δεν το θέσει η γραμμή είναι αποκλειστική

- Flush – δίνω την τιμή της γραμμής στο δίαυλο και ανανεώνεται η κύρια μνήμη

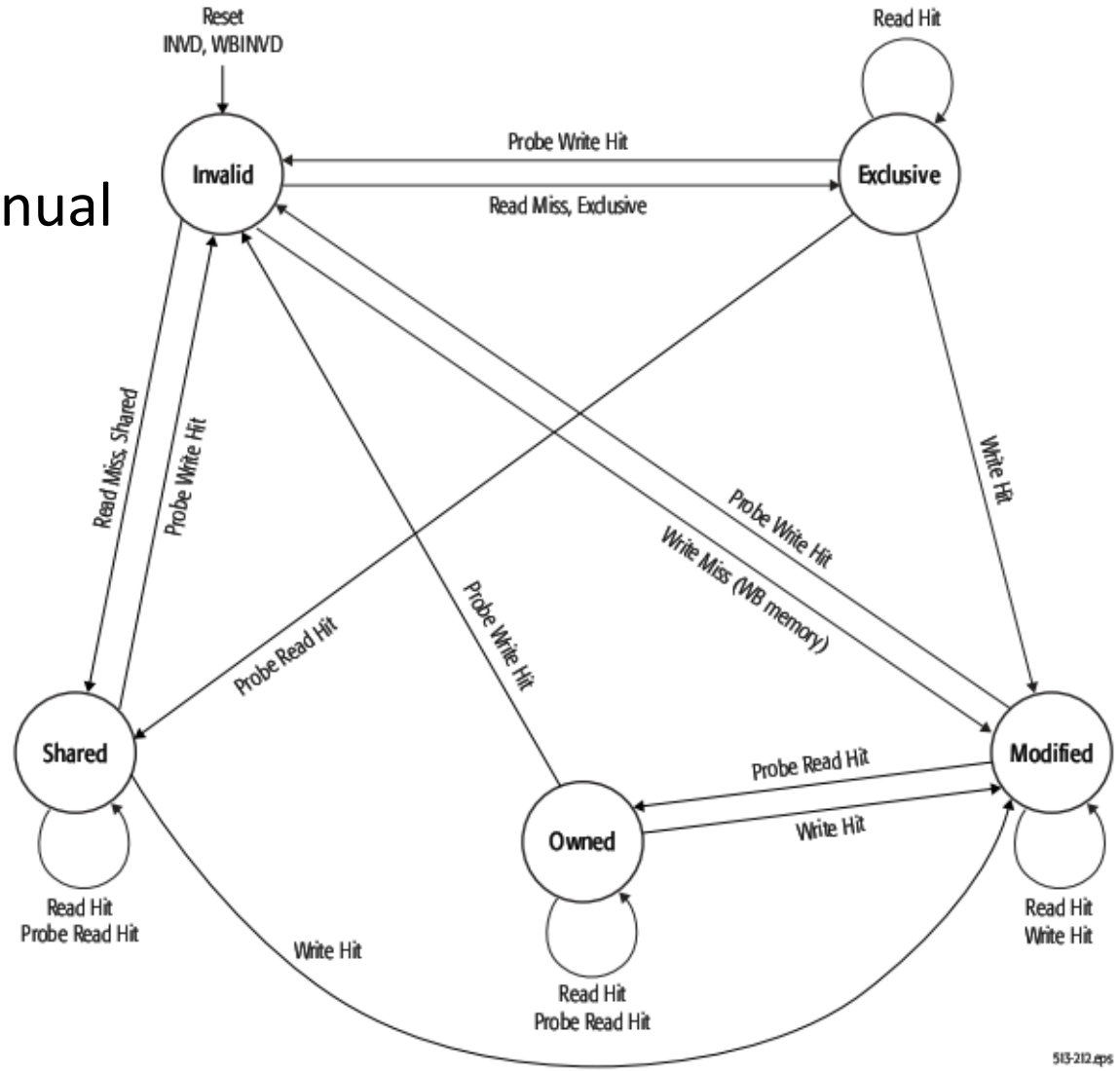


Πρωτόκολλο MOESI

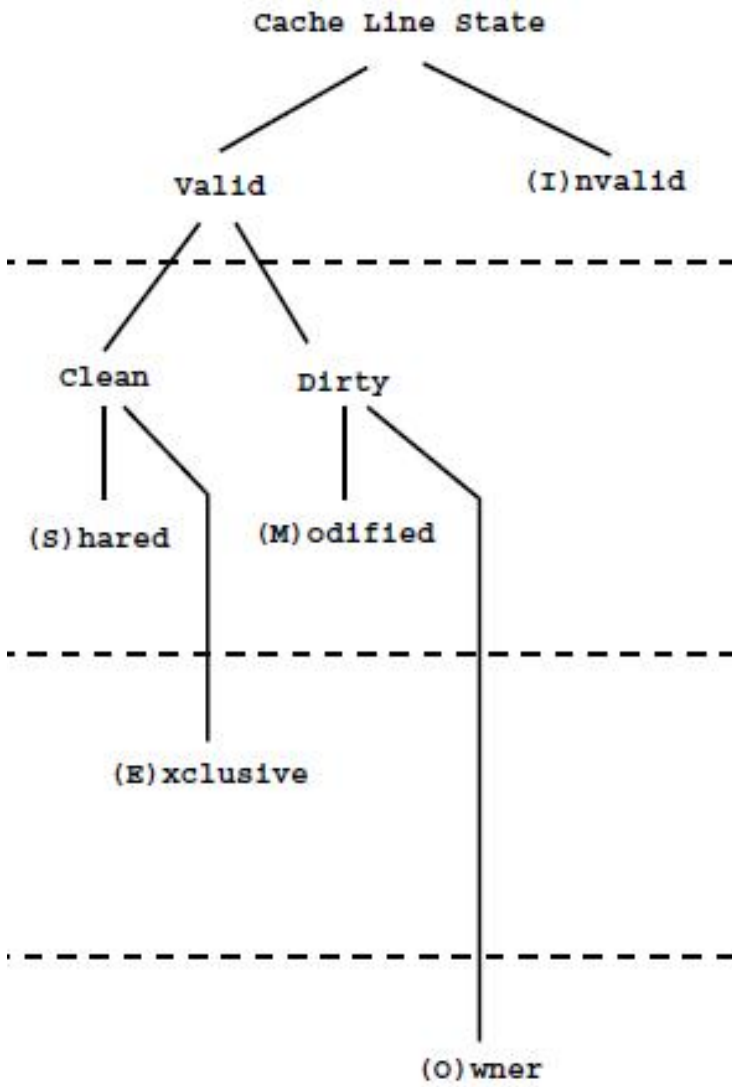
- Αν μετά την αλλαγή μιας γραμμής, η γραμμή χρησιμοποιηθεί για ανάγνωση, πρέπει να γραφτεί στη κύρια μνήμη
  - flush έξω από κατάσταση M
- Η κύρια μνήμη είναι γενικά αργή, οπότε συμφέρει να καθυστερήσουμε την εγγραφή όσο μπορούμε
- Χρειαζόμαστε μια επιπλέον κατάσταση όπου η γραμμή είναι αλλαγμένη αλλά υπάρχουν αντίγραφα σε άλλες caches: O - owned

Διάγραμμα καταστάσεων

AMD64 Architecture programmer's manual



Κατάταξη



MSI

MESI

MOESI

Πρωτόκολλα ενημέρωσης

- Οταν γίνεται μια αλλαγή, τα πρωτόκολλα που εξετάσαμε ακυρώνουν τα υπόλοιπα αντίγραφα (της γραμής cache)
- Αντί για ακύρωση, θα μπορούσαμε να ενημερώσουμε τις τιμές (update protocols)
- Ανάλογα με τον τρόπο που μοιράζονται τα δεδομένα τα νήματα μιας εφαρμογής, συμφέρει το ένα ή το άλλο είδος πρωτοκόλλου
  - update καλύτερα σε producer-consumer παραλληλισμό
  - αλλά προκαλούν πολύ περισσότερα μηνύματα στο δίκτυο

Ενδιάμεσες καταστάσεις

- Η μέχρι τώρα παρουσίαση ήταν ωραιοποιημένη (!)
- Οι αλλαγές κατάστασης δεν είναι αδιαίρετες (atomic)
- Παράδειγμα (MESI):
  - Δύο πυρήνες (A, B) έχουν τοπικό αντίγραφο γραμμής X και ζητούν (ταυτόχρονα) να αλλάξουν τη γραμμή: busUpgrade
  - Τα αιτήματα φτάνουν στους ελεγκτές cache και έστω ότι ο B καταφέρνει και «παίρνει» το δίαυλο πρώτος
  - Η cache του A, θα δει το παραπάνω και θα ακυρώσει το τοπικό αντίγραφο της γραμμής X.
  - Αλλά ο ελεγκτής έχει ήδη ζητήσει το δίαυλο με σκοπό να κάνει BusUpgrade. Τώρα πρέπει να ζητήσει αποκλειστική ανάγνωση...
- Πρέπει να ελέγχεται κάθε αίτημα που είναι σε εκρεμότητα για τυχόν αλλαγές