Caches for Parallel Architectures

(Coherence)

- **Figures, examples από**
  1. **Parallel Computer Architecture: A Hardware/Software Approach, D. E. Culler, J. P. Singh, Morgan Kaufmann Publishers, INC. 1999.**
  2. **Transactional Memory, D. Wood, Lecture Notes in ACACES 2009**

Σχεδίαση Επεξεργαστών

- Moore's Law (1964) :
  - # Transistors per IC doubles every 2 years (or 18 months)
    - » Πρακτικά η απόδοση του επεξεργαστή διπλασιάζεται κάθε 2 χρόνια.

- Όλο και περισσότερα προβλήματα…
- Memory wall
  - 1980 memory latency ~ 1 instruction
  - 2006 memory latency ~ 1000 instructions
- Power and cooling walls
- Αύξηση πολυπλοκότητας σχεδιασμού και επαλήθευσης (design and test complexity)
- Περιορισμένα περιθώρια περαιτέρω εκμετάλλευσης ILP

→ Παράλληλες Αρχιτεκτονικές

Παράλληλες Αρχιτεκτονικές (1)

- Οι πολυεπεξεργαστές γνώρισαν ιδιαίτερη ανάπτυξη από τη δεκαετία των 90s :
  - Servers
  - Supercomputers για την επίτευξη μεγαλύτερης επίδοσης σε σύγκριση ένα επεξεργαστή
- Στις μέρες μας (CMPs) :
  - Μείωση κόστους σχεδιασμού μέσω επαναχρησιμοποίησης (replication) σχεδίων
  - Εκμετάλλευση Thread-Level Parallelism (TLP) για την αντιμετώπιση του memory wall
  - Χαμηλότερο per-core power, περισσότερα cores.
- Αποδοτική χρησιμοποίηση πολυεπεξεργαστών (ιδιαίτερα σε servers) όπου υπάρχει thread-level parallelism
  - Αύξηση ενδιαφέροντος για τη σχεδίαση servers και την απόδοση τους

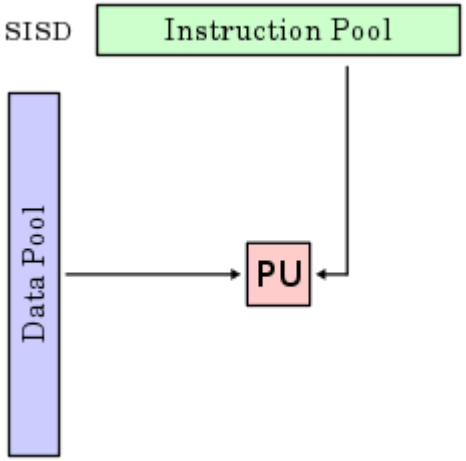
Παράλληλες Αρχιτεκτονικές (2)

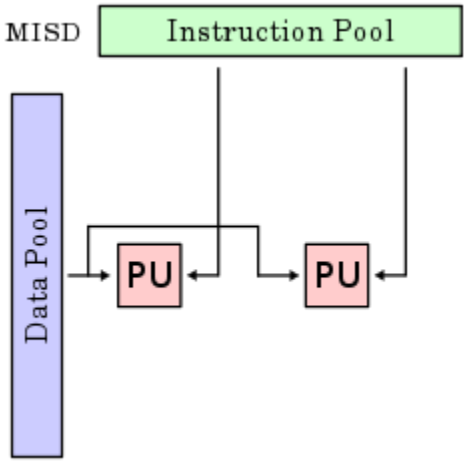
- Όλα αυτά οδηγούν σε μια νέα εποχή όπου τον κύριο ρόλο διαδραματίζουν οι **πολυεπεξεργαστές**
  - Desktop μηχανήματα για κάθε χρήστη με 2, 4, 6, 8, … πυρήνες

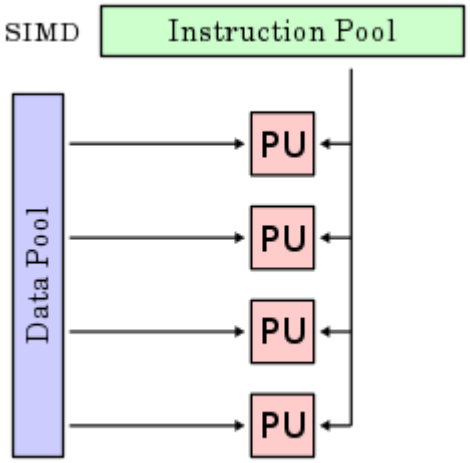
- *"We are dedicating all of our future product development to multicore designs. We believe this is a key inflection point for the industry"*
  - Intel CEO Paul Otellini, 2005

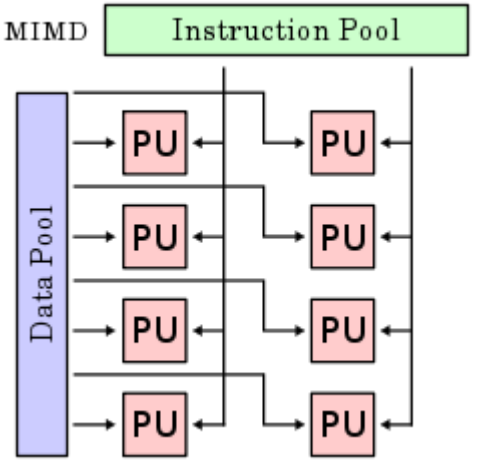
Ταξινόμηση Παράλληλων Αρχιτεκτονικών

- *Single Instruction stream, Single Data stream (SISD)*
- *Single Instruction stream, Multiple Data streams (SIMD)*
  - Πολλαπλοί επεξεργαστές, ίδιες εντολές, διαφορετικά δεδομένα (*data-level parallelism*).
- *Multiple Instruction streams, Single Data stream (MISD)*
  - Μέχρι σήμερα δεν έχει εμφανιστεί στην αγορά κάποιο τέτοιο σύστημα (είναι κυρίως για fault tolerance, π.χ. υπολογιστές που ελέγχουν πτήση αεροσκαφών).
- *Multiple Instruction streams, Multiple Data streams (MIMD)*
  - Ο κάθε επεξεργαστής εκτελεί τις δικές του εντολές και επεξεργάζεται τα δικά του δεδομένα. Πολλαπλά παράλληλα νήματα (thread-level parallelism).
- Θα ασχοληθούμε κυρίως με **MIMD** συστήματα.
  - Thread-level parallelism
  - Ευελιξία: Λειτουργία είτε ως single-user multiprocessors εστιάζοντας στην απόδοση μιας εφαρμογής, είτε ως multiprogrammed multiprocessors εκτελώντας πολλαπλές λειτουργίες ταυτόχρονα.
  - Πλεονεκτήματα κόστους-απόδοσης χρησιμοποιώντας off-the-self επεξεργαστές.


SISD
Instruction Pool
Data Pool
PU


MISD
Instruction Pool
Data Pool
PU
PU


SIMD
Instruction Pool
Data Pool
PU
PU
PU
PU


MIMD
Instruction Pool
Data Pool
PU
PU
PU
PU
PU
PU
PU
PU

MIMD Συστήματα (1)

- Παραδείγματα MIMD συστημάτων
  - Clusters (commodity/custom clusters)
  - Multicore systems

- Κάθε επεξεργαστής εκτελεί διαφορετικό process (διεργασία).
  - **process** : *"A segment of code that can be executed independently"*. Σε ένα πολυπρογραμματιστικό περιβάλλον, οι επεξεργαστές εκτελούν διαφορετικά tasks κι επομένως κάθε process είναι ανεξάρτητη από τις υπόλοιπες.
  - Όταν πολλαπλά processes μοιράζονται κώδικα και χώρο διευθύνσεων (address space) τότε ονομάζονται ***threads (νήματα)***.
  - Σήμερα ο όρος thread χρησιμοποιείται για να περιγράψει γενικά πολλαπλές εκτελέσεις, οι οποίες μπορεί να πραγματοποιηθούν σε διαφορετικούς επεξεργαστές ανεξάρτητα από το αν μοιράζονται ή όχι το address space.
  - Οι multithreaded (πολυνηματικές) αρχιτεκτονικές επιτρέπουν την ταυτόχρονη εκτέλεση πολλαπλών processes με διαφορετικό address space, καθώς και πολλαπλών threads που μοιράζονται το ίδιο address space.

MIMD Συστήματα (2)

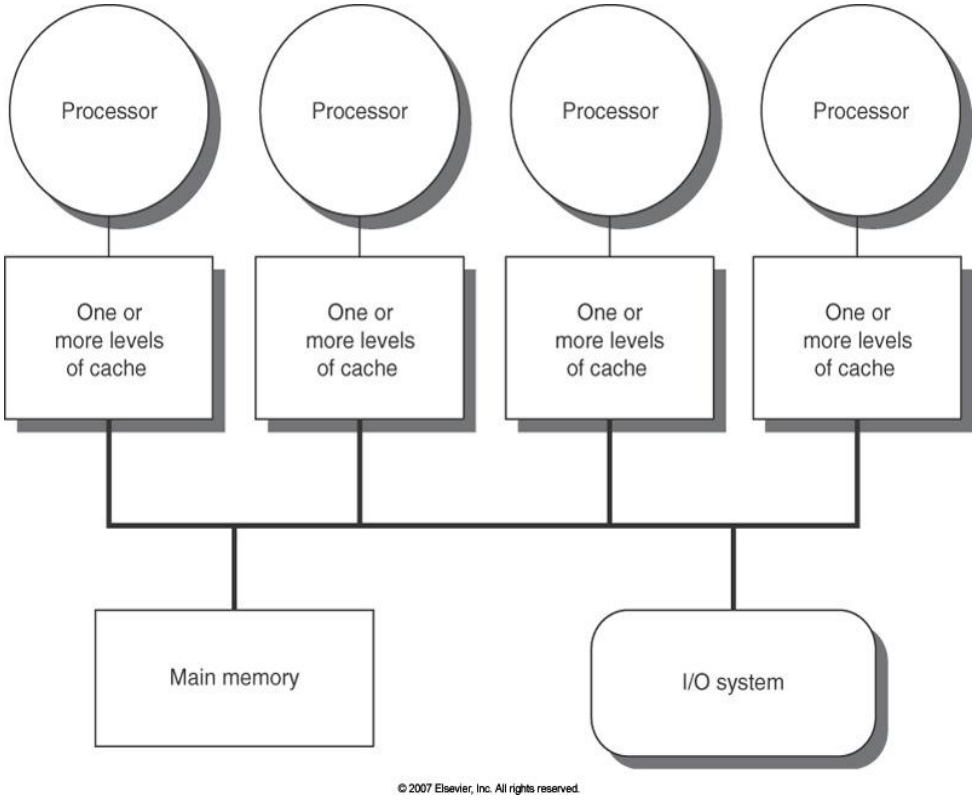
- Για την αποδοτική χρήση ενός MIMD συστήματος με n επεξεργαστές, απαιτούνται *τουλάχιστον* n threads/processes.
  - Δημιουργία από τον προγραμματιστή ή τον compiler

- Grain Size : Το “μέγεθος” (amount of computation) του κάθε thread
  - Fine-grain: Μερικές δεκάδες εντολές (π.χ. κάποιες επαναλήψεις ενός loop, instruction-level parallelism)
  - Coarse-grain: Εκατομμύρια εντολές (thread-level parallelism)

- Τα MIMD συστήματα χωρίζονται σε 2 κατηγορίες με βάση την οργάνωση της ιεραρχίας της μνήμης τους.
  - Centralized shared-memory architectures (Αρχιτεκτονικές συγκεντρωμένης κοινής μνήμης)
  - Distributed memory architectures (Αρχιτεκτονικές φυσικά κατανεμημένης μνήμης)

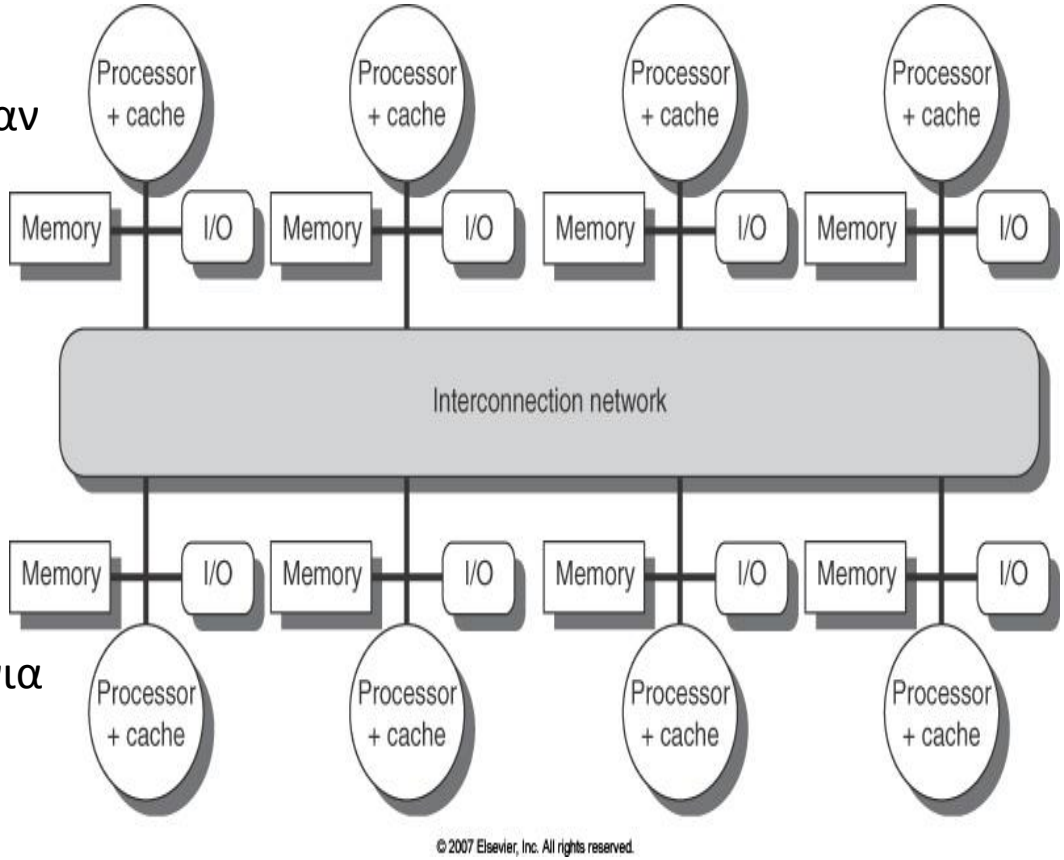
Centralized Shared-Memory Architectures

- Μικρός αριθμός επεξεργαστών (λιγότεροι από 100 το 2006).
- Όλοι οι επεξεργαστές μοιράζονται μια κεντρική μνήμη
  - Πολλαπλά banks
  - point-to-point connections, switches
  - Περιορισμένο scalability
- Symmetric multiprocessors (SMPs)
  - Η μνήμη έχει συμμετρική σχέση με τους επεξεργαστές
  - Ομοιόμορφος χρόνος προσπέλασης (Uniform Memory Access – UMA)



Distributed Memory Architectures (1)

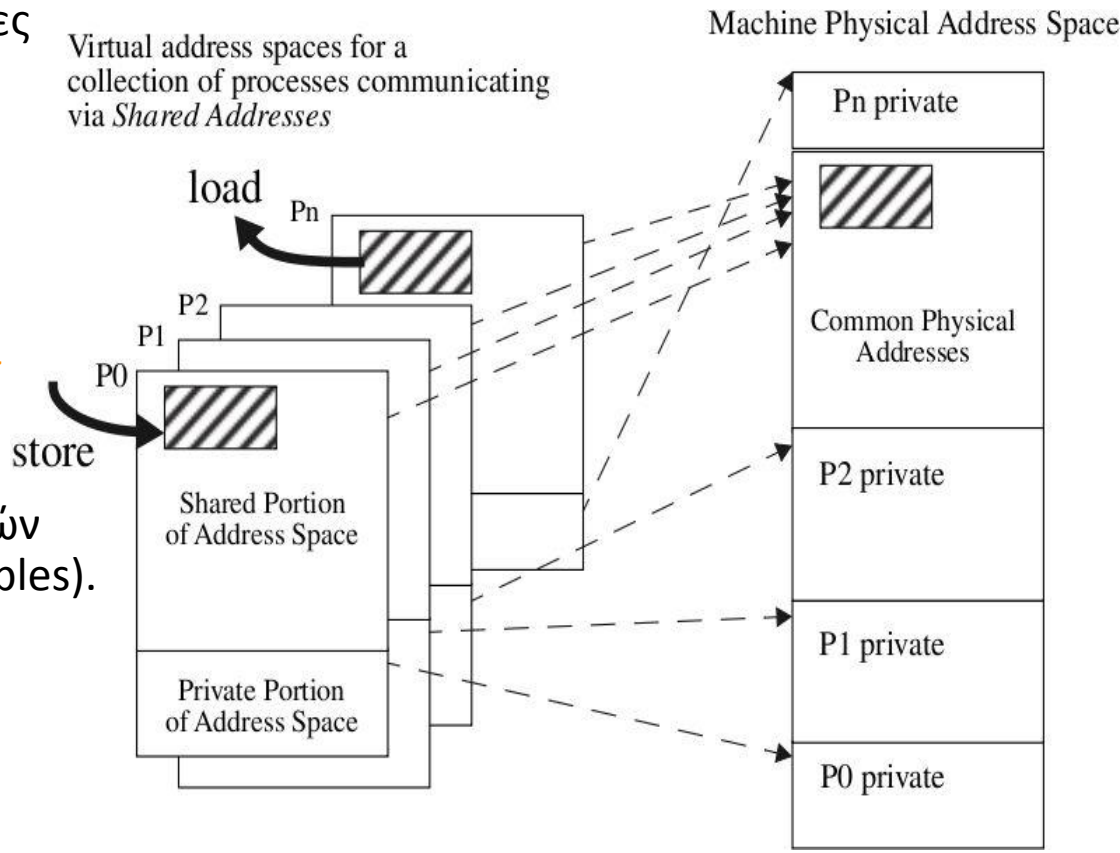
- Η μνήμη μοιράζεται τοπικά σε κάθε επεξεργαστή.
- Πλεονεκτήματα
  - Μεγαλύτερο εύρος ζώνης μνήμης αν η πλειοψηφία των προσπελάσεων γίνονται τοπικά σε κάθε κόμβο.
  - Μείωση χρόνου πρόσβασης σε δεδομένα αποθηκευμένα στην μνήμη του κάθε κόμβου.
- Μειονεκτήματα
  - Πολύπλοκη ανταλλαγή δεδομένων μεταξύ επεξεργαστών.
  - Πιο δύσκολη παραγωγή software για την εκμετάλλευση του αυξημένου εύρους ζώνης της μνήμης.



- Δύο μοντέλα επικοινωνίας για ανταλλαγή δεδομένων
  - Shared Address space
  - Message Passing

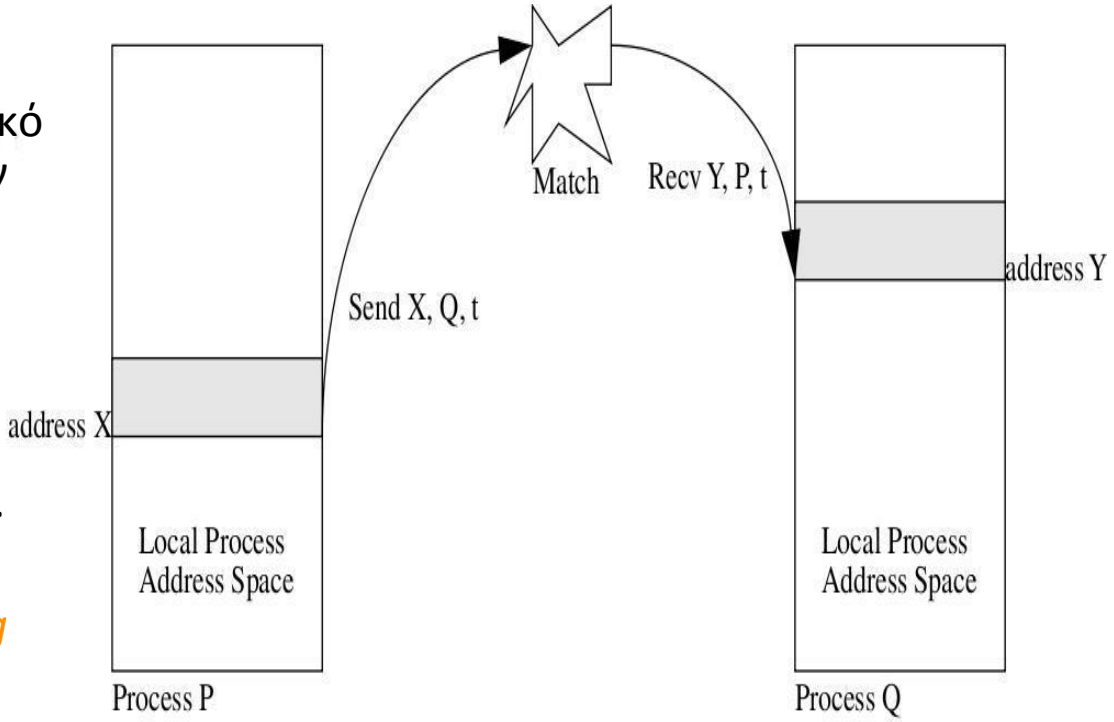
Distributed Memory Architectures (2)

- Shared address space
  - Οι φυσικά κατανεμημένες μνήμες χρησιμοποιούνται σαν ένας μοναδικός, διαμοιραζόμενος χώρος δεδομένων.
  - Η ίδια φυσική διεύθυνση σε 2 επεξεργαστές αναφέρεται στην ίδια τοποθεσία στο ίδιο κομμάτι της φυσικής μνήμης.
  - Επικοινωνία μέσω του κοινού χώρου (implicitly, με χρήση απλών Loads και Stores σε shared variables).
  - Οι πολυεπεξεργαστές αυτοί ονομάζονται *Distributed Shared-Memory (DSM)*.
  - Ο χρόνος πρόσβασης εξαρτάται από την τοποθεσία στην οποία βρίσκονται τα δεδομένα → *NUMA (Non-Uniform Memory Access)*.



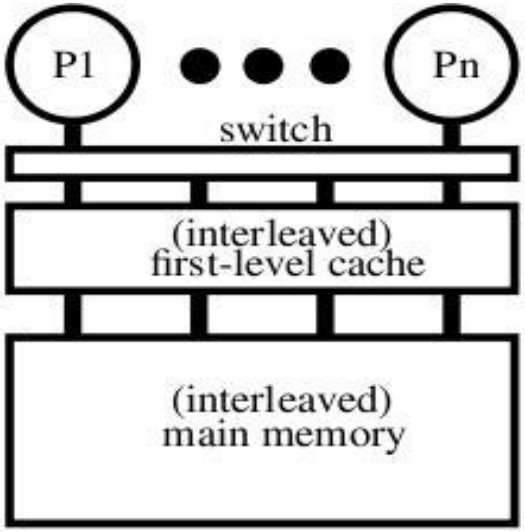
Distributed Memory Architectures (3)

- Private address space
  - Ο κάθε επεξεργαστής έχει το δικό του address space, το οποίο δεν μπορεί να προσπελαστεί από κάποιον άλλο.
  - Η ίδια φυσική διεύθυνση σε 2 επεξεργαστές αναφέρεται σε διαφορετικές τοποθεσίες σε διαφορετικά κομμάτια μνημών.
  - Επικοινωνία (explicitly) μέσω μηνυμάτων → *Message-Passing Multiprocessors*.

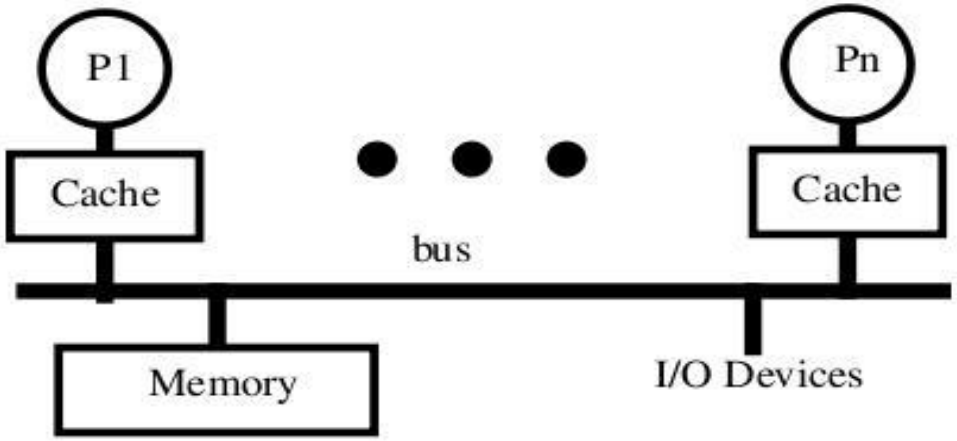


  - Κάθε συνδυασμός *send-receive* πραγματοποιεί ένα *συγχρονισμό ζεύγους* (pairwise synchronization) καθώς και μια *μεταφορά δεδομένων από μνήμη σε μνήμη* (memory-to-memory copy)
  - π.χ. clusters

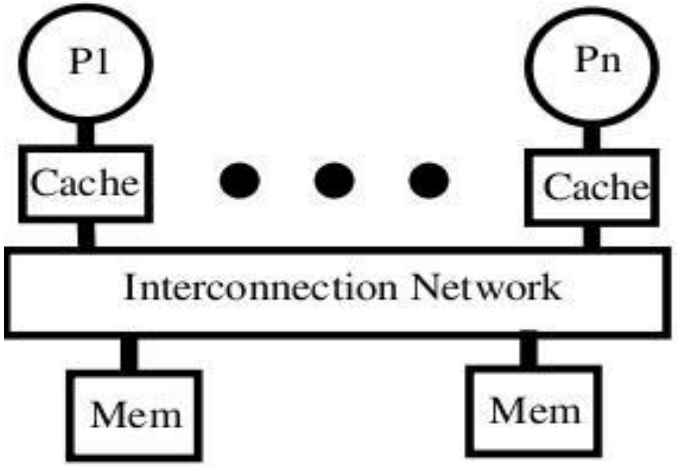
Shared Memory Architectures (1)



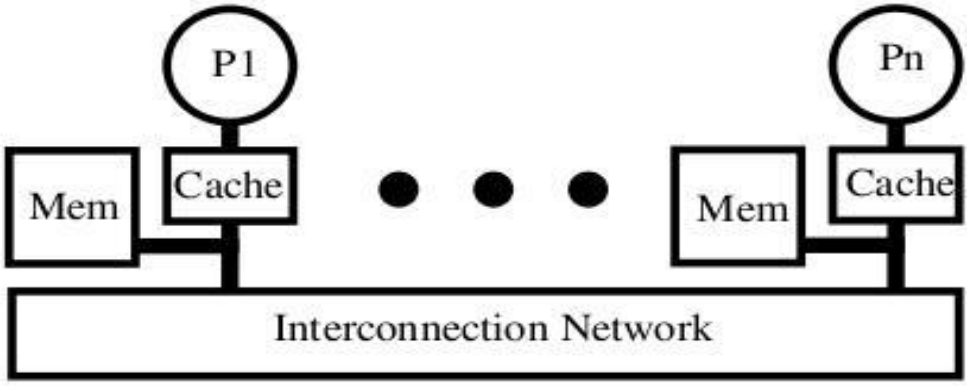
(a) Shared Cache



(b) Bus-based Shared Memory



(c) Dance-hall



(d) Distributed Memory

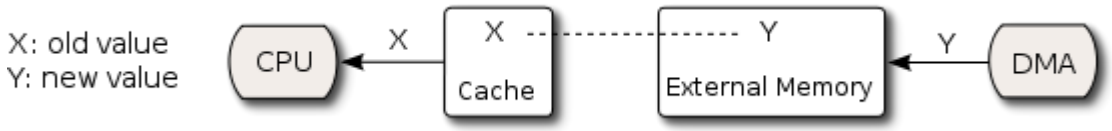
Shared Memory Architectures (2)

- Βασική ιδιότητα των συστημάτων μνήμης
  - *Κάθε ανάγνωση μιας τοποθεσίας, θα πρέπει να επιστρέφει την τελευταία τιμή που γράφτηκε σε αυτή*.
  - Βασική ιδιότητα τόσο για τα σειριακά προγράμματα, όσο και για τα παράλληλα.
- Η ιδιότητα αυτή διατηρείται όταν πολλαπλά threads εκτελούνται σε ένα επεξεργαστή, καθώς “βλέπουν” την ίδια ιεραρχία μνήμης.
- Στα πολυεπεξεργαστικά συστήματα, όμως, κάθε επεξεργαστής έχει τη δική του μονάδα κρυφής μνήμης (cache).
- Πιθανά προβλήματα :
  - Αντίγραφα μίας μεταβλητής είναι πιθανόν να υπάρχουν σε παραπάνω από μία caches.
  - Αν μια εγγραφή δεν είναι ορατή από όλους τους επεξεργαστές, τότε υπάρχει περίπτωση κάποιοι να προσπελάζουν την παλιά τιμή της μεταβλητής που είναι αποθηκευμένη στην cache τους.
  - Πρόβλημα : ***Συνάφειας Κρυφής Μνήμης (Cache Coherence)***

Πρόβλημα coherence στα μονοεπεξεργαστικά συστήματα?

Direct Memory Access

- DMA – CPU στην μνήμη:



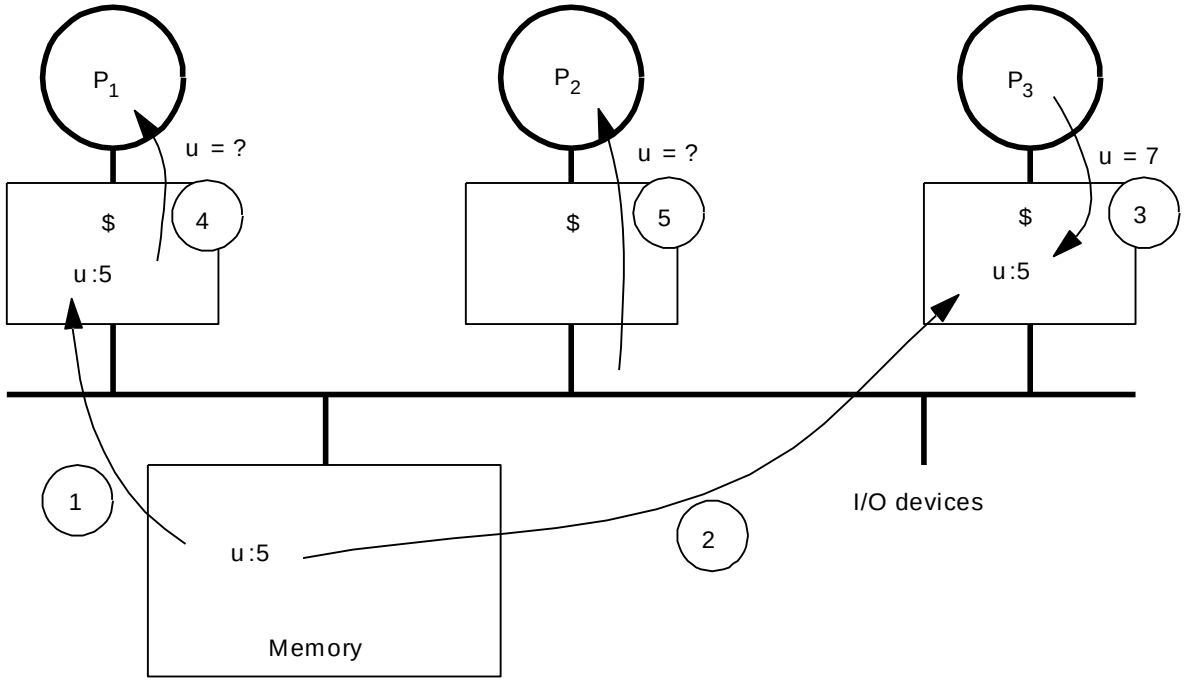
- **<u>Λύσεις:</u>**

  a) HW: cache invalidation for DMA writes or cache flush for DMA reads

  b) SW: OS must ensure that the cache lines are flushed before an outgoing DMA transfer is started and invalidated before a memory range affected by an incoming DMA transfer is accessed

  c) Non cacheable DMAs

Παράδειγμα Προβλήματος Συνάφειας Κρυφής Μνήμης



- Οι επεξεργαστές βλέπουν διαφορετική τιμή για τη μεταβλητή u μετά τη λειτουργία 3
- Με τις write back caches, η τιμή που γράφεται πίσω στη μνήμη εξαρτάται από το ποια cache και πότε διώχνει ή αντιγράφει δεδομένα

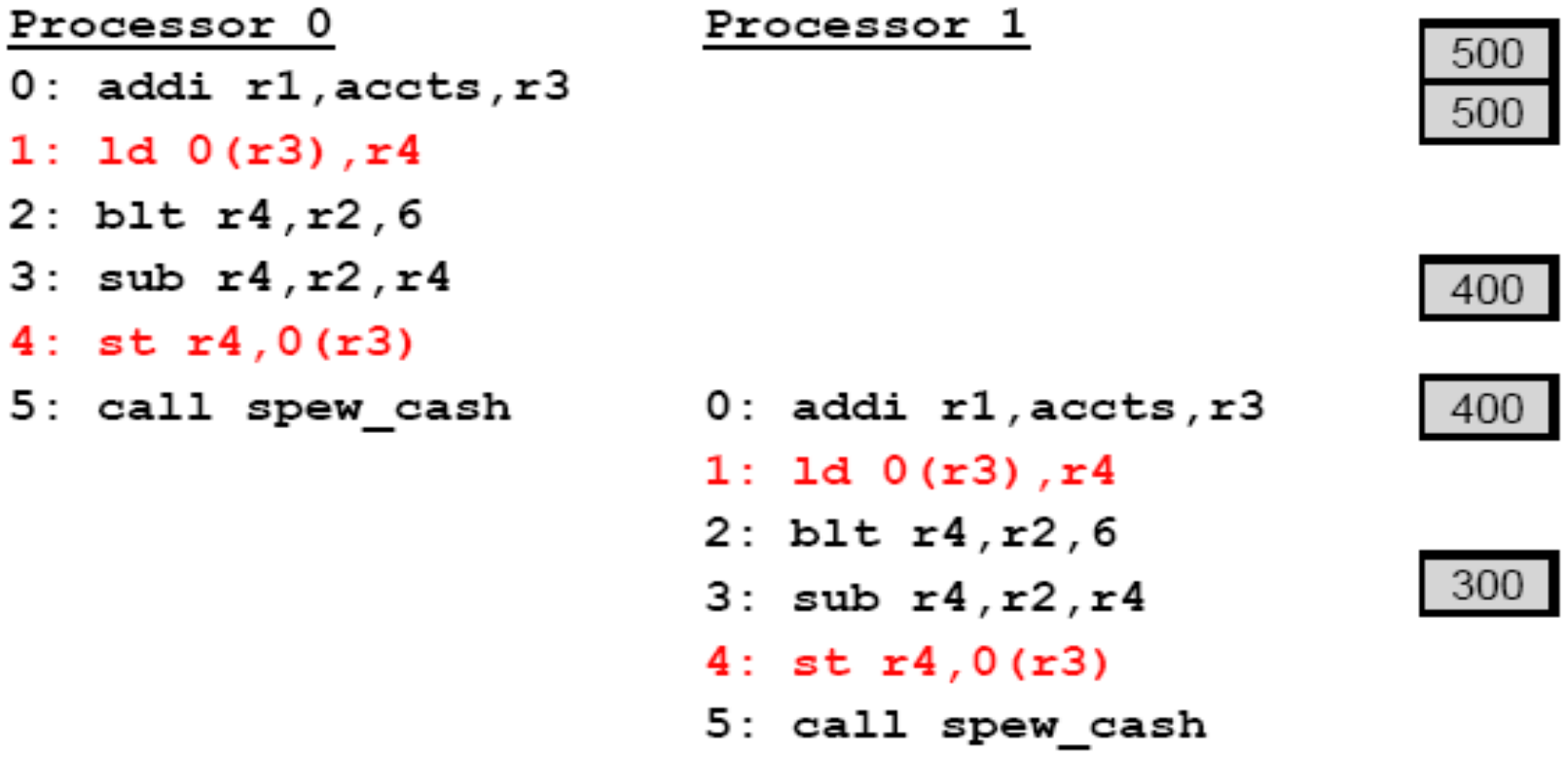
⇨Απαράδεκτο, αλλά συμβαίνει συχνά!

Παράδειγμα

```
Processor 0                 Processor 1
0: addi r1,accts,r3
1: ld 0(r3),r4
2: blt r4,r2,6
3: sub r4,r2,r4
4: st r4,0(r3)
5: call spew_cash           0: addi r1,accts,r3
                            1: ld 0(r3),r4
                            2: blt r4,r2,6
                            3: sub r4,r2,r4
                            4: st r4,0(r3)
                            5: call spew_cash
```

- Δύο ταυτόχρονες αναλήψεις €100 από τον ίδιο λογαριασμό από 2 διαφορετικά ATMs.
  - Κάθε transaction σε διαφορετικό επεξεργαστή.
  - Η διεύθυνση περιέχεται στον καταχωρητή $r3.

Παράδειγμα (Χωρίς caches)



- Χωρίς caches → **κανένα πρόβλημα!**

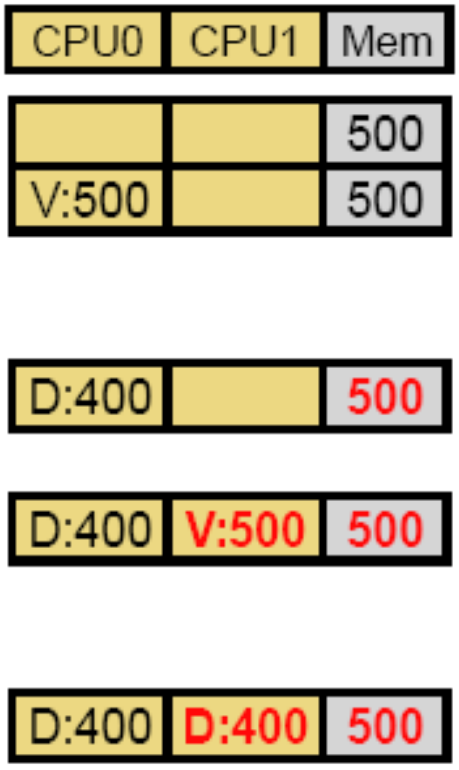
Παράδειγμα (Incoherence)

Processor 0

```
0: addi r1,accts,r3
1: ld 0(r3),r4
2: blt r4,r2,6
3: sub r4,r2,r4
4: st r4,0(r3)
5: call spew_cash
```

Processor 1

```
0: addi r1,accts,r3
1: ld 0(r3),r4
2: blt r4,r2,6
3: sub r4,r2,r4
4: st r4,0(r3)
5: call spew_cash
```



- ***Write-back caches***
  - 3 πιθανά αντίγραφα : memory, p0$, p1$
  - Το σύστημα είναι πιθανό να είναι incoherent.

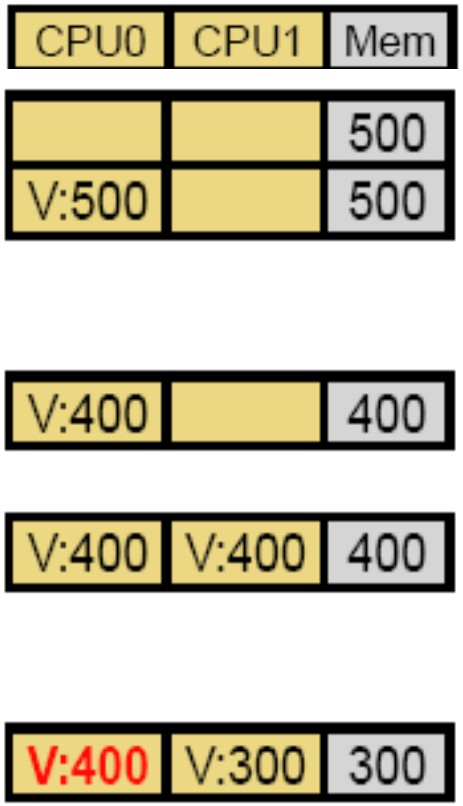
Παράδειγμα (Incoherence)

Processor 0

```
0: addi r1,accts,r3
1: ld 0(r3),r4
2: blt r4,r2,6
3: sub r4,r2,r4
4: st r4,0(r3)
5: call spew_cash
```

Processor 1

```
0: addi r1,accts,r3
1: ld 0(r3),r4
2: blt r4,r2,6
3: sub r4,r2,r4
4: st r4,0(r3)
5: call spew_cash
```



- ***Write-through caches***
  - Τώρα 2 διαφορετικά αντίγραφα!
  - Και πάλι πρόβλημα! (π.χ. έστω ότι ο p0 εκτελεί και άλλη ανάληψη)
- Οι write-through caches δεν λύνουν το πρόβλημα!

Cache Coherence (1)

- Διατήρηση της βασικής ιδιότητας
  - *Κάθε ανάγνωση μιας τοποθεσίας, θα πρέπει να επιστρέφει την **ΤΕΛΕΥΤΑΙΑ** τιμή που γράφτηκε σε αυτή.*
- Πώς ορίζεται το *"ΤΕΛΕΥΤΑΙΑ"* ;
- Σειριακά προγράμματα
  - Ορίζεται σύμφωνα με τη σειρά που επιβάλλεται από τον κώδικα.
- Παράλληλα προγράμματα
  - Δύο threads μπορεί να γράψουν στην ίδια διεύθυνση την ίδια χρονική στιγμή.
  - Ένα thread μπορεί να διαβάσει μια μεταβλητή ακριβώς μετά την εγγραφή της από κάποιο άλλο, αλλά λόγω της ταχύτητας μετάδοσης η εγγραφή αυτή δεν έχει γίνει ακόμα ορατή.
  - Η σειρά που επιβάλλει ο κώδικας ορίζεται εντός του thread.
  - Απαιτείται όμως και ο ορισμός μιας σειράς που να αφορά όλα τα threads (global ordering).

Cache Coherence (2)

- Έστω ότι υπάρχει μια κεντρική μνήμη και καμία cache.
  - Κάθε λειτουργία σε μια θέση μνήμης προσπελάζει την ίδια φυσική θέση.
  - Η μνήμη επιβάλλει μια *καθολική σειρά* στις λειτουργίες όλων των threads σε αυτή τη θέση.
  - Οι λειτουργίες κάθε thread διατηρούν τη σειρά του προγράμματος του.
  - Κάθε διάταξη που διατηρεί τη σειρά των λειτουργιών των επιμέρους προγραμμάτων είναι αποδεκτή / έγκυρη.
- Ως "*τελευταία*" ορίζεται η πιο πρόσφατη λειτουργία σε μια υποθετική ακολουθία που διατηρεί τις παραπάνω ιδιότητες.
- Σε ένα πραγματικό σύστημα δεν μπορεί να κατασκευαστεί αυτή η καθολική σειρά.
  - Χρήση caches.
  - Αποφυγή serialization.
- Το σύστημα ***πρέπει*** να είναι κατασκευασμένο ώστε τα προγράμματα να συμπεριφέρονται ***σαν να υπήρχε*** αυτή η καθολική σειρά.

Cache Coherence - Ορισμός

- Ένα σύστημα είναι *coherent (συναφές)* αν για κάθε εκτέλεση τα αποτελέσματα (*οι τιμές που επιστρέφονται από τις λειτουργίες ανάγνωσης*) είναι τέτοια, ώστε σε κάθε θέση να μπορούμε να κατασκευάσουμε μια υποθετική ακολουθιακή σειρά όλων των λειτουργιών **στη θέση αυτή**, που να είναι συνεπής με τα αποτελέσματα της εκτέλεσης και στην οποία :
  - Οι λειτουργίες κάθε thread πραγματοποιούνται με την σειρά κατά την οποία κλήθηκαν από αυτό το thread.
  - Η τιμή που επιστρέφεται από μια λειτουργία ανάγνωσης είναι η τιμή της τελευταίας εγγραφής στη συγκεκριμένη θέση σύμφωνα με την υποθετική ακολουθιακή σειρά.

- 3 συνθήκες για να είναι ένα σύστημα coherent.

Cache Coherence - Συνθήκες

1. *A read by processor P to a location X that follows a write by P to X, with no writes of X by another processor occurring between the write and the read by P, always returns the value written by P.*
   - Διατήρηση της σειράς του προγράμματος.
   - Ισχύει και για uniprocessors.
2. *A read by a processor to location X that follows a write by another processor to X returns the written value if the read and write are sufficiently separated in time and no other writes to X occur between the two accesses.*
   - ***write propagation***
   - Μια λειτουργία ανάγνωσης δεν μπορεί να επιστρέφει παλιότερες τιμές.
3. *Writes to the same location are serialized; that is, two writes to the same location by any two processors are seen in the same order by all processors. (e.g. if values 1 and then 2 are written to a location, processors can never the value of the location as 2 and then later read it as 1)*
   - ***write serialization***. Χρειαζόμαστε read serialization;

Bus Snooping Cache Coherence (1)

- Χρήση διαδρόμου
  - Προσφέρει μια απλή και κομψή υλοποίηση για cache coherence.
  - Προβλήματα scalability.

- Όλες οι συσκευές που είναι συνδεδεμένες πάνω στο διάδρομο μπορούν να παρακολουθούν όλα τα bus transactions.

- Τρεις φάσεις σε κάθε transaction
  - *Διαιτησία*: Ο bus arbiter αποφασίζει ποια συσκευή έχει το δικαίωμα να χρησιμοποιήσει το bus
  - *Αποστολή εντολής/διεύθυνσης*: Η επιλεγμένη συσκευή μεταδίδει το είδος της εντολής (read / write) καθώς και τη διεύθυνση της αντίστοιχης θέσης. Όλοι παρακολουθούν και αποφασίζουν αν τους "ενδιαφέρει" ή όχι.
  - *Μεταφορά δεδομένων*

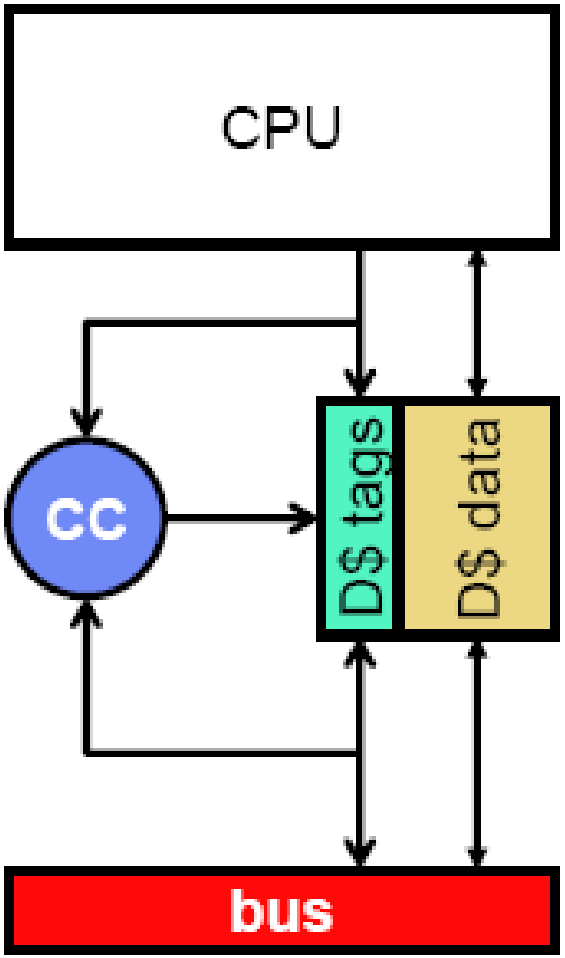
Bus Snooping Cache Coherence (2)

- Εκμετάλλευση του cache block state
  - Κάθε cache μαζί με τα tag και data αποθηκεύει και την κατάσταση στην οποία βρίσκεται το block (π.χ. invalid, valid, dirty).

- Ουσιαστικά για κάθε block 'λειτουργεί' μια μηχανή πεπερασμένων καταστάσεων (FSM)
  - Κάθε πρόσβαση σε ένα block ή σε κάποια διεύθυνση που αντιστοιχεί στο ίδιο cache line με αυτό το block, προκαλεί μια μεταβολή του state ή αλλιώς μια αλλαγή κατάστασης στο FSM.

- Σε multiprocessor συστήματα το state ενός block είναι ένας πίνακας μήκους *p*, όπου *p* ο αριθμός των caches.
  - Το ίδιο FSM καθορίζει τις αλλαγές καταστάσεων για όλα τα blocks σε όλες τις caches.
  - Το state ενός block μπορεί να διαφέρει από cache σε cache.

Hardware για Cache Coherence



- Coherence Controller (CC)
- Παρακολουθεί την κίνηση στο διάδρομο (διευθύνσεις και δεδομένα)
- Εκτελεί το πρωτόκολλο συνάφειας (coherence protocol).
  - Αποφασίζει τι θα κάνει με το τοπικό αντίγραφο με βάση αυτά που βλέπει να μεταδίδονται στο διάδρομο.

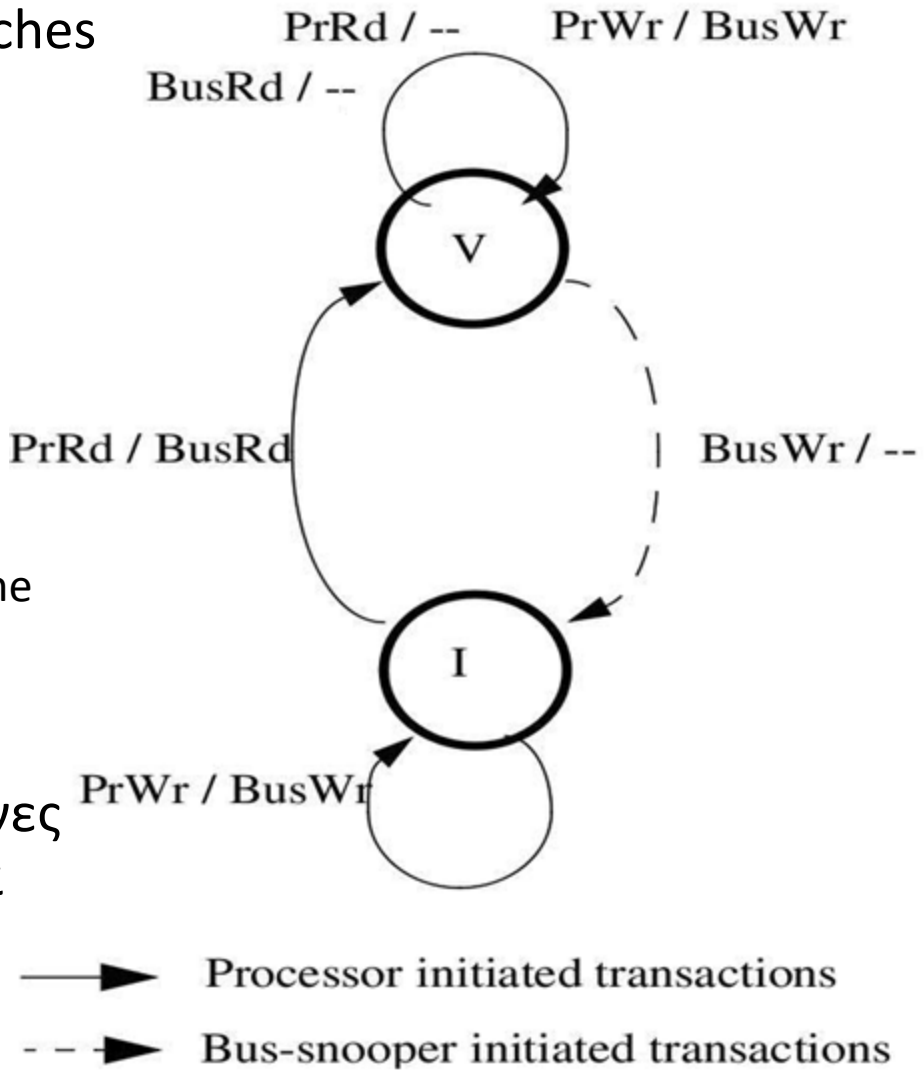
Bus Snooping Cache Coherence (3)

Υλοποίηση Πρωτοκόλλου

- Ο ελεγκτής της cache δέχεται είσοδο από 2 μεριές
  - Αιτήσεις πρόσβασης στη μνήμη από τον επεξεργαστή.
  - Ο "κατάσκοπος" (bus snooper) ενημερώνει για bus transactions που πραγματοποιούν οι υπόλοιπες caches.
- Σε κάθε περίπτωση ανταποκρίνεται
  - Ενημερώνει την κατάσταση του block με βάση το FSM.
  - Αποστολή δεδομένων.
  - Παραγωγή νέων bus transactions.
- Κάθε πρωτόκολλο αποτελείται από τα παρακάτω δομικά στοιχεία
  - Το σύνολο των επιτρεπτών states για κάθε block στις caches.
  - Το state transition diagram που με είσοδο το state του block και τη αίτηση του επεξεργαστή ή το παρατηρούμενο bus transaction υποδεικνύει ως έξοδο το επόμενο επιτρεπτό state για το block αυτό.
  - Τις ενέργειες που επιβάλλεται να πραγματοποιηθούν κατά την αλλαγή κατάστασης του block.

Simple Invalidation-based protocol (1)

- write-through, write-no-allocate caches
- 2 states για κάθε block
  - Valid
  - Invalid
- Σε περίπτωση εγγραφής ενός block
  - Ενημερώνεται η κύρια μνήμη μέσω ενός bus transaction.
  - Κάθε bus snooper ενημερώνει τον cache controller του, ο οποίος ακυρώνει το τοπικό αντίγραφο αν υπάρχει.
- Επιτρέπονται πολλαπλές ταυτόχρονες αναγνώσεις (multiple readers). Μια εγγραφή όμως τους ακυρώνει.
- Είναι coherent ;



Simple Invalidation-based protocol (2)

- Μπορούμε να κατασκευάσουμε μια καθολική σειρά που να ικανοποιεί τη σειρά του προγράμματος και τη σειριοποίηση των εγγραφών;

- Υποθέτουμε ***atomic*** bus transactions και memory operations.
  - Ένα transaction κάθε φορά στο bus.
  - Κάθε επεξεργαστής περιμένει να ολοκληρωθεί μια πρόσβαση του στη μνήμη πριν αιτηθεί καινούρια.
  - Οι εγγραφές (και οι ακυρώσεις) ολοκληρώνονται κατά τη διάρκεια του bus transactions.
- Όλες οι εγγραφές εμφανίζονται στο bus (write-through protocol).
  - Οι εγγραφές σε μια θέση σειριοποιούνται σύμφωνα με τη σειρά με την οποία εμφανίζονται στο bus. (*bus order*)
  - Οι ακυρώσεις πραγματοποιούνται επίσης σύμφωνα με το bus order.
- Πως παρεμβάλλουμε τις αναγνώσεις στη σειρά αυτή;
  - Οι αναγνώσεις δεν είναι υποχρεωτικό να προκαλέσουν bus transaction και μπορούν να εκτελούνται ανεξάρτητα και ταυτόχρονα στις caches.

Simple Invalidation-based protocol (3)

- Σειριοποίηση αναγνώσεων
  - Read hit ή read miss?

- Read Miss
  - Ικανοποιείται μέσω bus transaction. Επομένως σειριοποιείται μαζί με τις εγγραφές.
  - Θα δει την τιμή της τελευταίας εγγραφής σύμφωνα με το bus order.

- Read Hit
  - Ικανοποιείται από την τιμή που βρίσκεται μέσα στην cache.
  - Βλέπει την τιμή της πιο πρόσφατης εγγραφής από τον *ίδιο* επεξεργαστή ή της πιο πρόσφατης ανάγνωσης (read miss).
  - Και τα 2 (write και read miss) ικανοποιούνται μέσω bus transactions.
  - Επομένως και τα read hits βλέπουν τις τιμές σύμφωνα με το bus order.

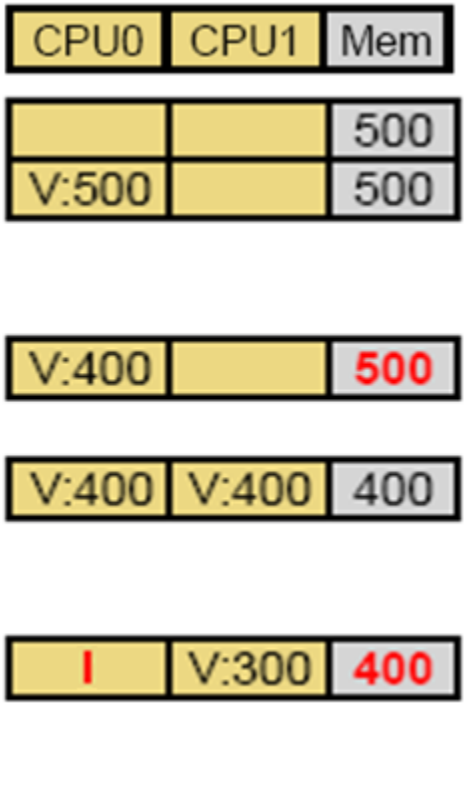
VI protocol - Παράδειγμα (write-back caches)

Processor 0

```
0: addi r1,accts,r3
1: ld 0(r3),r4
2: blt r4,r2,6
3: sub r4,r2,r4
4: st r4,0(r3)
5: call spew_cash
```

Processor 1

```
0: addi r1,accts,r3
1: ld 0(r3),r4
2: blt r4,r2,6
3: sub r4,r2,r4
4: st r4,0(r3)
5: call spew_cash
```



- Το **ld** του p1 δημιουργεί ένα BusRd
  - Ο p0 απαντά γράφοντας πίσω το modified block (WB) και ακυρώνοντας το στην cache του (μετάβαση στην κατάσταση I)

MSI Write-Back Invalidation Protocol (1)

- Το VI πρωτόκολλο δεν είναι αποδοτικό
- **VI → MSI**
  - "Σπάσιμο" του V σε 2 καταστάσεις
- 3 καταστάσεις (states)
  1. Τροποποιημένη [Modified(M)]
  2. Μοιραζόμενη [Shared(S)]
  3. Άκυρη [Invalid(I)]
- 2 τύποι αιτήσεων από τον επεξεργαστή
  - PrRd (ανάγνωση) και PrWr (εγγραφή)
- 3 bus transactions
  - BusRd : Ζητά αντίγραφο χωρίς σκοπό να το τροποποιήσει
  - BusRdX : Ζητά αντίγραφο για να το τροποποιήσει
  - BusWB : Ενημερώνει τη μνήμη

MSI Write-Back Invalidation Protocol (2)

- Διάγραμμα Μετάβασης Καταστάσεων

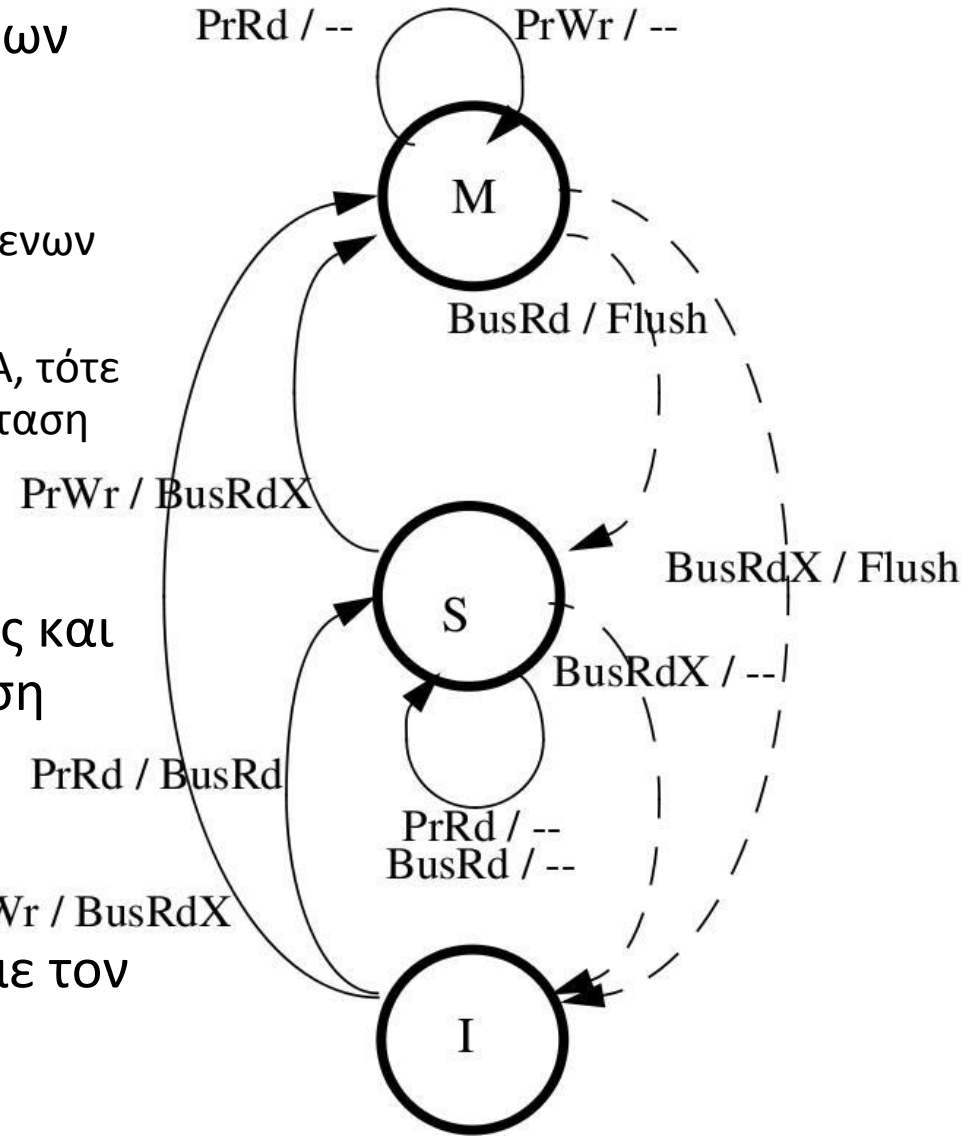
  ⟶ Μεταβάσεις εξαιτίας λειτουργιών του "τοπικού" επεξεργαστή.

  - - - -> Μεταβάσεις εξαιτίας των παρατηρούμενων bus transactions.

  A/B Αν ο cache controller παρατηρήσει το A, τότε εκτός από τη μετάβαση στη νέα κατάσταση προκαλεί και το B.

  -- Καμία ενέργεια.

- Δεν περιλαμβάνονται οι μεταβάσεις και οι ενέργειες κατά την αντικατάσταση ενός block στην cache.
- Όσο πιο "ψηλά" στο διάγραμμα βρίσκεται ένα block, τόσο πιο στενά συνδεδεμένο (bound) είναι με τον επεξεργαστή.



MSI protocol - Παράδειγμα (write-back caches)

Processor 0

```
0: addi r1,accts,r3
1: ld 0(r3),r4
2: blt r4,r2,6
3: sub r4,r2,r4
4: st r4,0(r3)
5: call spew_cash
```

Processor 1

```
0: addi r1,accts,r3
1: ld 0(r3),r4
2: blt r4,r2,6
3: sub r4,r2,r4
4: st r4,0(r3)
5: call spew_cash
```

CPU0	CPU1	Mem
		500
S:500		500

M:400		**500**
S:400	S:400	400

I:	M:300	**400**

- Το **ld** του p1 δημιουργεί ένα BusRd
  - Ο p0 απαντά γράφοντας πίσω το modified block (WB) και αλλάζοντας το αντίγραφο του σε S
- Το **st** του p1 δημιουργεί ένα BusRdX
  - Ο p0 απαντά ακυρώνοντας το αντίγραφο του (μετάβαση σε I)

MSI – Coherence

- Η διάδοση των εγγραφών είναι προφανής.
- Σειροποίηση εγγραφών :
  - Όλες οι εγγραφές που εμφανίζονται στο διάδρομο (BusRdX) διατάσσονται από αυτόν.
  - Οι αναγνώσεις που εμφανίζονται στο διάδρομο διατάσσονται ως προς τις εγγραφές.
  - Για τις εγγραφές που **δεν** εμφανίζονται στο διάδρομο:
    - Μια ακολουθία τέτοιων εγγραφών μεταξύ 2 bus transactions για το ίδιο block **πρέπει** να προέρχονται από τον ίδιο επεξεργαστή P.
    - Στη σειριοποίηση η ακολουθία εμφανίζεται μεταξύ αυτών των 2 transactions.
    - Οι αναγνώσεις από τον P θα βλέπουν τις εγγραφές με αυτή τη σειρά ως προς τις υπόλοιπες εγγραφές.
    - Οι αναγνώσεις από άλλους επεξεργαστές διαχωρίζονται από την ακολουθία με ένα bus transaction, η οποία τις τοποθετεί έτσι σε σειρά ως προς τις εγγραφές.
    - Οι αναγνώσεις από όλους τους επεξεργαστές βλέπουν τις εγγραφές με την ίδια σειρά.

MESI Write-Back Invalidation Protocol (1)

- Πρόβλημα MSI
  - 2 transactions για ανάγνωση και τροποποίηση ενός block, ακόμα και αν δεν τα μοιράζεται κανείς.
- 4 καταστάσεις (states)
  1. Τροποποιημένη [Modified(M)]
  2. Αποκλειστική [Exclusive(E)] → Μόνο αυτή η cache έχει αντίγραφο (μη τροποποιημένο).
  3. Μοιραζόμενη [Shared(S)] → Δύο ή περισσότερες caches έχουν αντίγραφο.
  4. Άκυρη [Invalid(I)]
- Αν κανείς δεν έχει αντίγραφο του block, τότε ένα PrRd έχει σαν αποτέλεσμα την μετάβαση I → E.
  - Στο διάδρομο χρειάζεται ένα σήμα shared ως απάντηση σε ένα BusRd.

MESI Write-Back Invalidation Protocol (2)

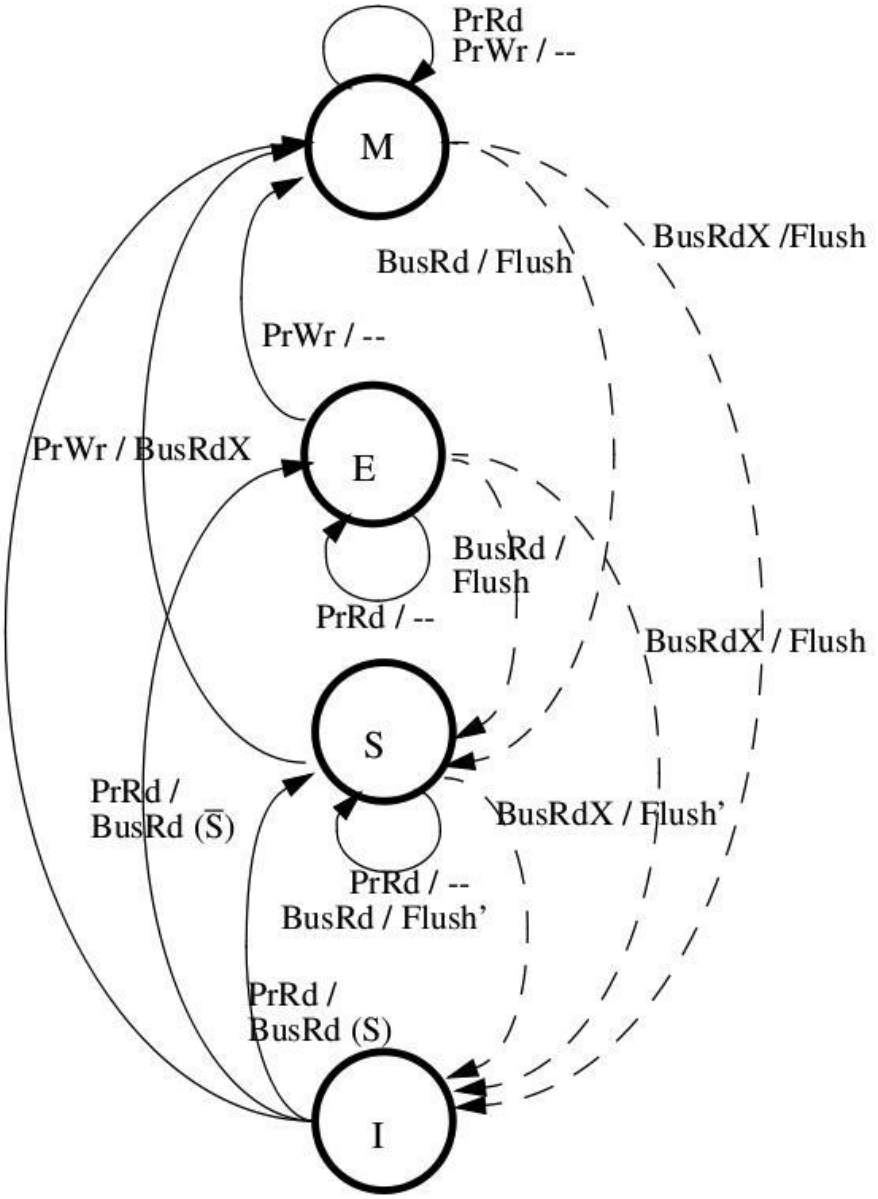
- Διάγραμμα Μετάβασης Καταστάσεων
  - ——→ Μεταβάσεις εξαιτίας λειτουργιών του "τοπικού" επεξεργαστή.
  - - - - → Μεταβάσεις εξαιτίας των παρατηρούμενων bus transactions.

A/B Αν ο cache controller παρατηρήσει το A, τότε εκτός από τη μετάβαση στη νέα κατάσταση προκαλεί και το B.

-- Καμία ενέργεια.

PrWr / BusRdX

- Ένα block μπορεί να βρίσκεται σε κατάσταση S ενώ δεν υπάρχουν άλλα αντίγραφα.
  - Πώς;



Ανακεφαλαίωση - Coherence Snooping Protocols

- Διατηρούμε τον επεξεργαστή, τη κύρια μνήμη και τις caches.
  - Επέκταση του cache controller - εκμετάλλευση του bus.
- Write-back caches
  - Αποδοτική αξιοποίηση του περιορισμένου bus bandwidth.
  - Δεν προκαλούν bus transactions όλες οι λειτουργίες μνήμης.
  - Πιο δύσκολη υλοποίηση της συνάφειας.
- Χρήση του modified state (τροποποιημένη κατάσταση)
  - Αποκλειστική ιδιοκτησία → δεν υπάρχει άλλο έγκυρο αντίγραφο.
  - Η κύρια μνήμη μπορεί να έχει ή να μην έχει αντίγραφο.
  - Η cache είναι υπεύθυνη να παρέχει το block σε όποιον το ζητήσει.
- Exclusivity (αποκλειστικότητα)
  - Η cache μπορεί να τροποποιήσει το block χωρίς να ειδοποιήσει κανένα → χωρίς bus transaction
  - Πριν την εγγραφή **πρέπει** να αποκτήσει αποκλειστικότητα.
  - Ακόμα και αν το block είναι valid → write miss

Invalidation Protocols

- Write-miss
  - Προκαλεί ένα ειδικό transaction : *read-exclusive* (RdX)
  - Ειδοποιεί τους υπόλοιπους ότι ακολουθεί εγγραφή και αποκτά αποκλειστική ιδιοκτησία.
  - Όλοι όσοι διαθέτουν αντίγραφο του block το διαγράφουν.
  - Μόνο μια RdX επιτυγχάνει κάθε φορά. Πολλαπλές αιτήσεις σειριποιούνται από το διάδρομο.
  - Τελικά τα νέα δεδομένα γράφονται στην κύρια μνήμη όταν το block εκδιωχθεί από την cache.
  - Αν ένα block δεν έχει τροποποιηθεί (modified state), τότε δεν χρειάζεται να γραφτεί στην κύρια μνήμη όταν εκδιωχθεί από την cache.

Update Protocols

- Μια λειτουργία εγγραφής ενημερώνει και τυχόν αντίγραφα του block στις υπόλοιπες caches.

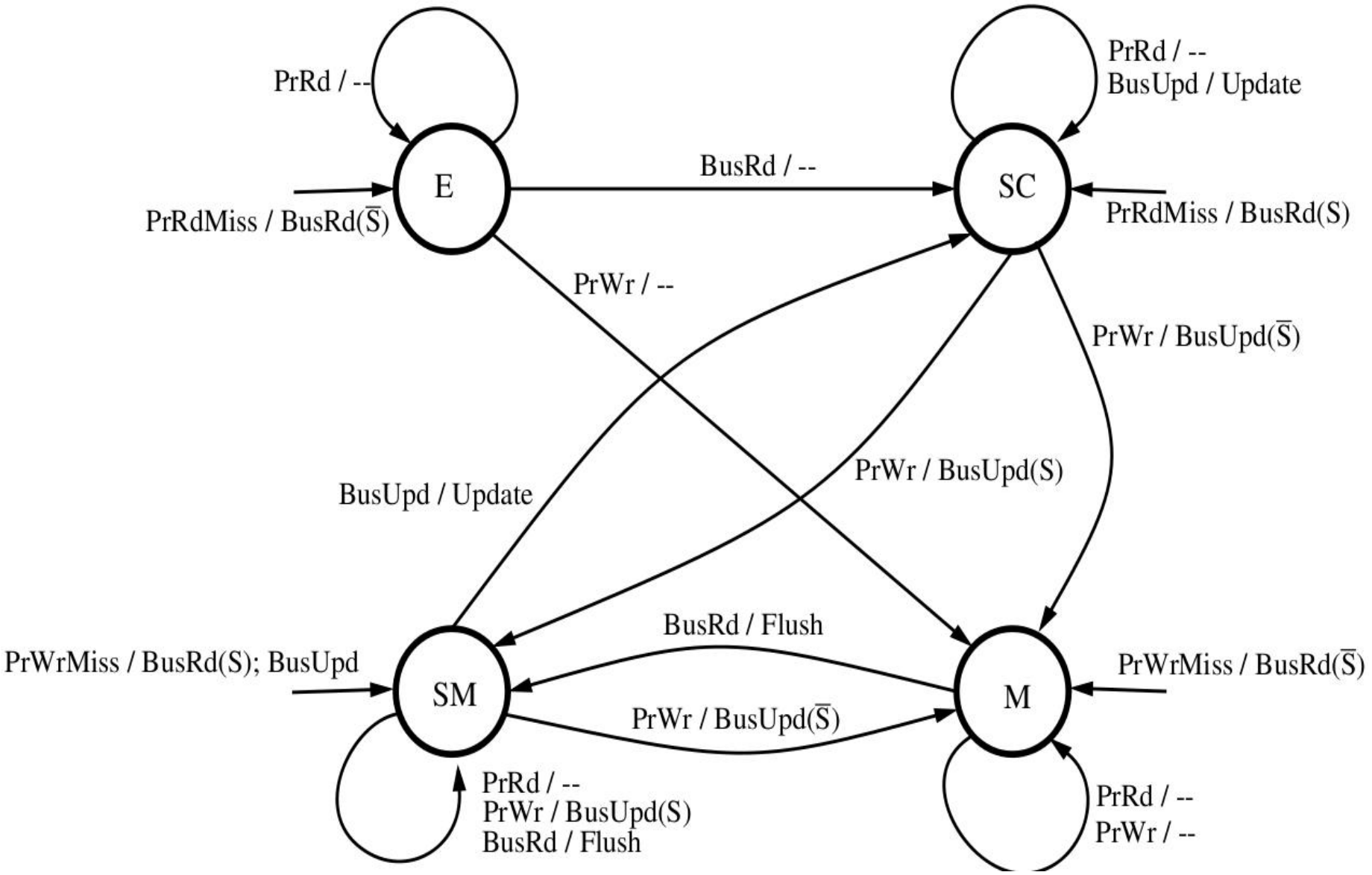
- Πλεονεκτήματα
  - Μικρότερη καθυστέρηση πρόσβασης στο block από τις άλλες caches.
  - Όλοι ενημερώνονται με ένα μόνο transaction.

- Μειονεκτήματα
  - Πολλαπλές εγγραφές στο block από τον ίδιο επεξεργαστή προκαλούν πολλαπλά transactions για τις ενημερώσεις.

Dragon Write-Back Update Protocol (1)

- 4 καταστάσεις (states)
  1. Αποκλειστική [Exclusive (E)] → Μόνο αυτή η cache έχει αντίγραφο (μη τροποποιημένο). Η κύρια μνήμη **είναι ενημερωμένη** (up-to-date).
  2. Μοιραζόμενη-καθαρή [Shared-clean (Sc)] → Δύο ή περισσότερες caches έχουν αντίγραφο. Η κύρια μνήμη **δεν είναι υποχρεωτικά up-to-date**.
  3. Μοιραζόμενη-τροποποιημένη [Shared-modified (Sm)] → Δύο ή περισσότερες caches έχουν αντίγραφο, η κύρια μνήμη **δεν είναι up-to-date** και η cache αυτή έχει την ευθύνη να ενημερώσει την κύρια μνήμη όταν εκδιώξει το block.
  4. Τροποποιημένη [Modified (M)] → Μόνο η cache αυτή διαθέτει το τροποποιημένο block ενώ η κύρια μνήμη **δεν είναι up-to-date**.
- Δεν υπάρχει Invalid state.
  - Το πρωτόκολλο διατηρεί πάντα τα blocks που βρίσκονται στις caches up-to-date.
- Δύο νέες αιτήσεις από τον επεξεργαστή : PrRdMiss, PrWrMiss
- Ένα νέο bus transaction : BusUpd

Dragon Write-Back Update Protocol (2)



Dragon – Παράδειγμα

Ενέργεια στον επεξεργαστή	*Κατάσταση P1*	*Κατάσταση P2*	*Κατάσταση P3*	*Ενέργεια στο διάδρομο*	*Τα δεδομένα παρέχονται από*
P1 διαβάζει u	E	---	---	BusRd	Mem
P3 διαβάζει u	Sc	---	Sc	BusRd	Mem
P3 γράφει u	Sc	---	Sm	BusUpd	P3 Cache
P1 διαβάζει u	Sc	---	Sm	---	---
P2 διαβάζει u	Sc	Sc	Sm	BusRd	P3 Cache

Invalidation vs. Update Protocols

- Σε κάποια cache γίνεται εγγραφή σε ένα block. Πριν την επόμενη εγγραφή στο ίδιο block, θέλει κάποιος άλλος να το διαβάσει;
- Ναι :
  - Invalidation
    - Read-miss → πιθανώς πολλαπλά transactions ☹
  - Update
    - Read-hit αν είχαν από πριν αντίγραφα→ ενημέρωση με ένα μόνο transaction ☺
- Όχι :
  - Invalidation
    - Πολλαπλές εγγραφές χωρίς επιπλέον κίνηση στο bus ☺
    - Εκκαθάριση αντιγράφων που δε χρησιμοποιούνται ☺
  - Update
    - Πολλαπλές αχρείαστες ενημερώσεις (και σε πιθανώς “νεκρά” αντίγραφα) ☹

Protocol Design Tradeoffs (1)

- Η σχεδίαση πολυεπεξεργαστικών συστημάτων είναι πολύπλοκη
  - Αριθμός επεξεργαστών
  - Ιεραρχία μνήμης (levels, size, associativity, bs, …)
  - Διάδρομος
  - Memory System (interleaved banks, width of banks, …)
  - I/O subsystem
  - **Cache Coherence Protocol** (Protocol class, states, actions, …)

- Το πρωτόκολλο επηρεάζει χαρακτηριστικά του συστήματος, όπως latency και bandwidth.
- Η επιλογή του πρωτοκόλλου επηρεάζεται από τη ζητούμενη απόδοση και συμπεριφορά του συστήματος καθώς και από την οργάνωση της ιεραρχίας μνήμης και της επικοινωνίας.

Protocol Design Tradeoffs (2)

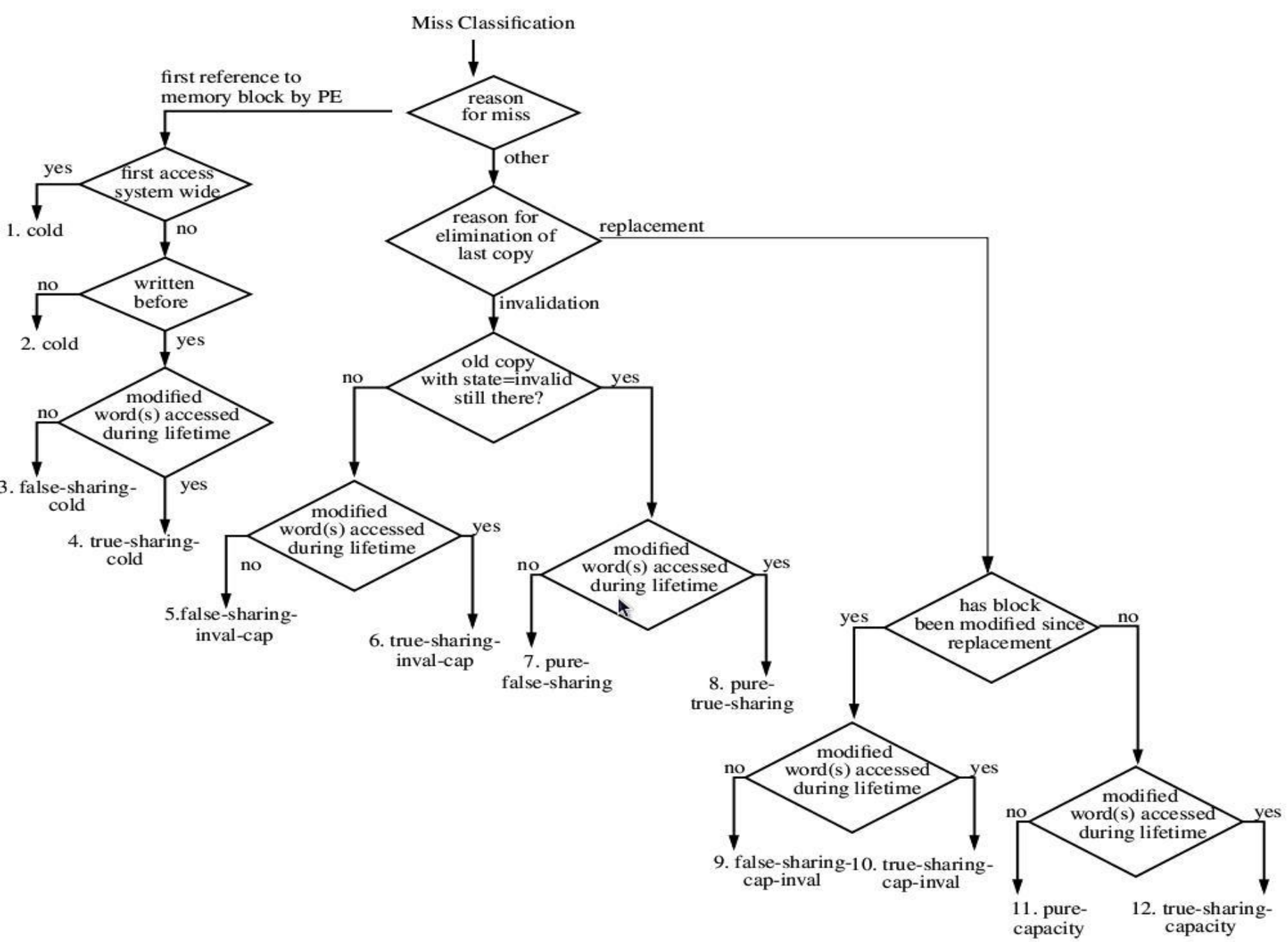
- Write-Update vs. Write-Invalidate

- Write-run: Μια σειρά εγγραφών από **ένα** επεξεργαστή σε ένα block μνήμης, η αρχή και το τέλος της οποίας ορίζονται από λειτουργίες σε αυτό το block από **άλλους** επεξεργαστές.
  - W2, R1, W1, W1, R1, W1, R3
  - Write-run length = 3

- Write-Invalidate: Ένα write-run οποιουδήποτε μήκους θα δημιουργήσει **ένα μοναδικό** coherence miss.
- Write-Update: Ένα write-run **μήκους L** θα προκαλέσει **L updates**.

4C Cache Misses Model

- Compulsory misses (cold)
  - Πρώτη πρόσβαση σε ένα block.
  - Αύξηση του block size.
- Capacity misses
  - Το block δε χωρά στην cache (ακόμα και σε full associative cache).
  - Αύξηση cache size.
- Conflict misses
  - Το block δε χωρά στο set που γίνεται mapped.
  - Αύξηση associativity.
- **Coherence misses** (communication)
  - **True sharing** : Όταν ένα data word χρησιμοποιείται από 2 ή παραπάνω επεξεργαστές.
  - **False sharing** : Όταν ανεξάρτητα data words που χρησιμοποιούνται από διαφορετικούς επεξεργαστές ανήκουν στο ίδιο cache block.


Miss Classification
reason for miss
first reference to memory block by PE
other
yes
first access system wide
1. cold
no
no
written before
2. cold
yes
no
modified word(s) accessed during lifetime
3. false-sharing-cold
yes
4. true-sharing-cold
reason for elimination of last copy
replacement
invalidation
no
old copy with state=invalid still there?
yes
no
modified word(s) accessed during lifetime
5.false-sharing-inval-cap
yes
6. true-sharing-inval-cap
no
modified word(s) accessed during lifetime
7. pure-false-sharing
yes
8. pure-true-sharing
yes
has block been modified since replacement
no
no
modified word(s) accessed during lifetime
9. false-sharing-cap-inval
yes
10. true-sharing-cap-inval
no
modified word(s) accessed during lifetime
11. pure-capacity
yes
12. true-sharing-capacity

Protocol Design Tradeoffs (3)

- Cache Block Size
- Αύξηση του block size μπορεί να οδηγήσει :
  - √ Μείωση του miss rate (good spatial locality).
  - X Αύξηση του miss penalty και ίσως του hit cost.
  - X Αύξηση του miss rate εξαιτίας false sharing (poor spatial locality).
  - X Αύξηση του traffic στο bus, λόγω μεταφοράς «αχρείαστων» δεδομένων (mismatch fetch/access size, false sharing).
- Υπάρχει η τάση για χρησιμοποίηση μεγαλύτερων cache blocks.
  - Απόσβεση κόστους του bus transaction και της πρόσβασης στη μνήμη μεταφέροντας περισσότερα δεδομένα.
  - Hardware και software μηχανισμοί για αντιμετώπιση του false sharing.

False sharing reduction

1. Βελτιωμένο data layout προκειμένου να αποφευχθεί η τοποθέτηση ανεξάρτητων δεδομένων στο ίδιο block.

- Data Padding
  - eg. Dummy variables μεταξύ lock variables που είναι τοποθετημένες κοντά η μια στην άλλη.
  - Tradeoff : *locality vs. false sharing*
- Χρήση array of arrays ώστε να βεβαιωθούμε ότι κάθε submatrix είναι τοποθετημένο συνεχόμενα στη μνήμη.
  - Tradeoff : *false sharing vs. instruction overhead*

2. Partial-Block Invalidation
   - Το block “σπάει” σε sub-blocks, για κάθε ένα από τα οποία διατηρείται το state.
   - Σε κάθε miss φέρνουμε όλα τα invalid sub-blocks.
   - Κάνουμε invalidate μόνο το sub-block που περιέχει τα δεδομένα που θα τροποποιηθούν.
   - Tradeoff : *less false sharing miss vs. more invalidation messages*

Scalable Multiprocessor Systems

- Τα συστήματα που στηρίζονται στη χρήση διαδρόμου δεν είναι scalable.
  - Όλα τα modules (cores, memories, etc) συνδέονται με ένα set καλωδίων.
  - Περιορισμένο bandwidth → Δεν αυξάνεται με την πρόσθεση παραπάνω επεξεργαστών → **Saturation** (κορεσμός).
  - Μεγαλύτερο bus → Μεγαλύτερο latency.
- Ένα scalable σύστημα πρέπει να αντιμετωπίζει αυτά τα προβλήματα.
  - Το συνολικό bandwidth θα πρέπει να αυξάνει με τον αριθμό των επεξεργαστών.
  - Ο χρόνος που απαιτείται για κάποια ενέργεια δε θα πρέπει να αυξάνει πολύ (πχ. Εκθετικά) με το μέγεθος του συστήματος.
  - Πρέπει να είναι cost-effective.
- Χάνουμε βασικές ιδιότητες του διαδρόμου.
  - Άγνωστος αριθμός ταυτόχρονων transactions.
  - Δεν υπάρχει global arbitration.
  - Τα αποτελέσματα (πχ. αλλαγές στο state) γίνονται απευθείας ορατά μόνο από τους κόμβους που συμμετέχουν στο transaction.

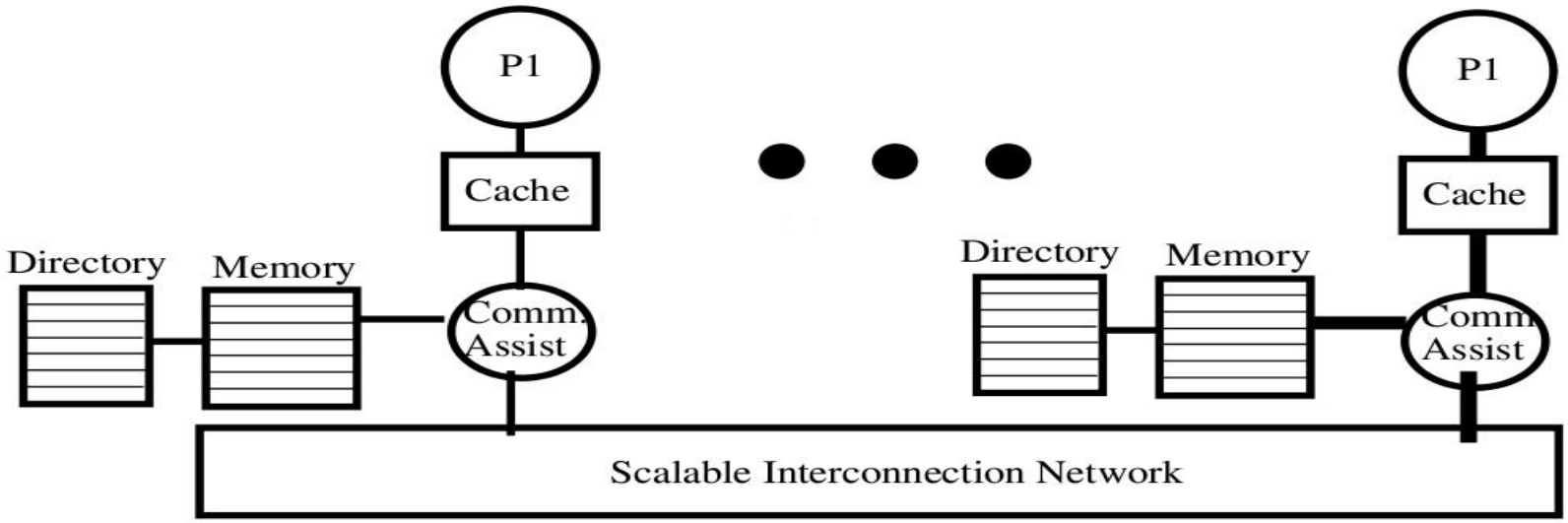
Scalable Cache Coherence

- Interconnect
  - Αντικατάσταση του διαδρόμου με scalable interconnects (point-to-point networks, eg. mesh)
- Processor snooping bandwidth
  - Μέχρι τώρα τα πρωτόκολλα έκαναν broadcast (*spam everyone!*)
  - Μεγάλο ποσοστό snoops δεν προκαλούν κάποια μετάβαση
  - Για loosely shared data, κατά πάσα πιθανότητα μόνο ένας επεξεργαστής έχει αντίγραφο

→ Scalable Directory protocol

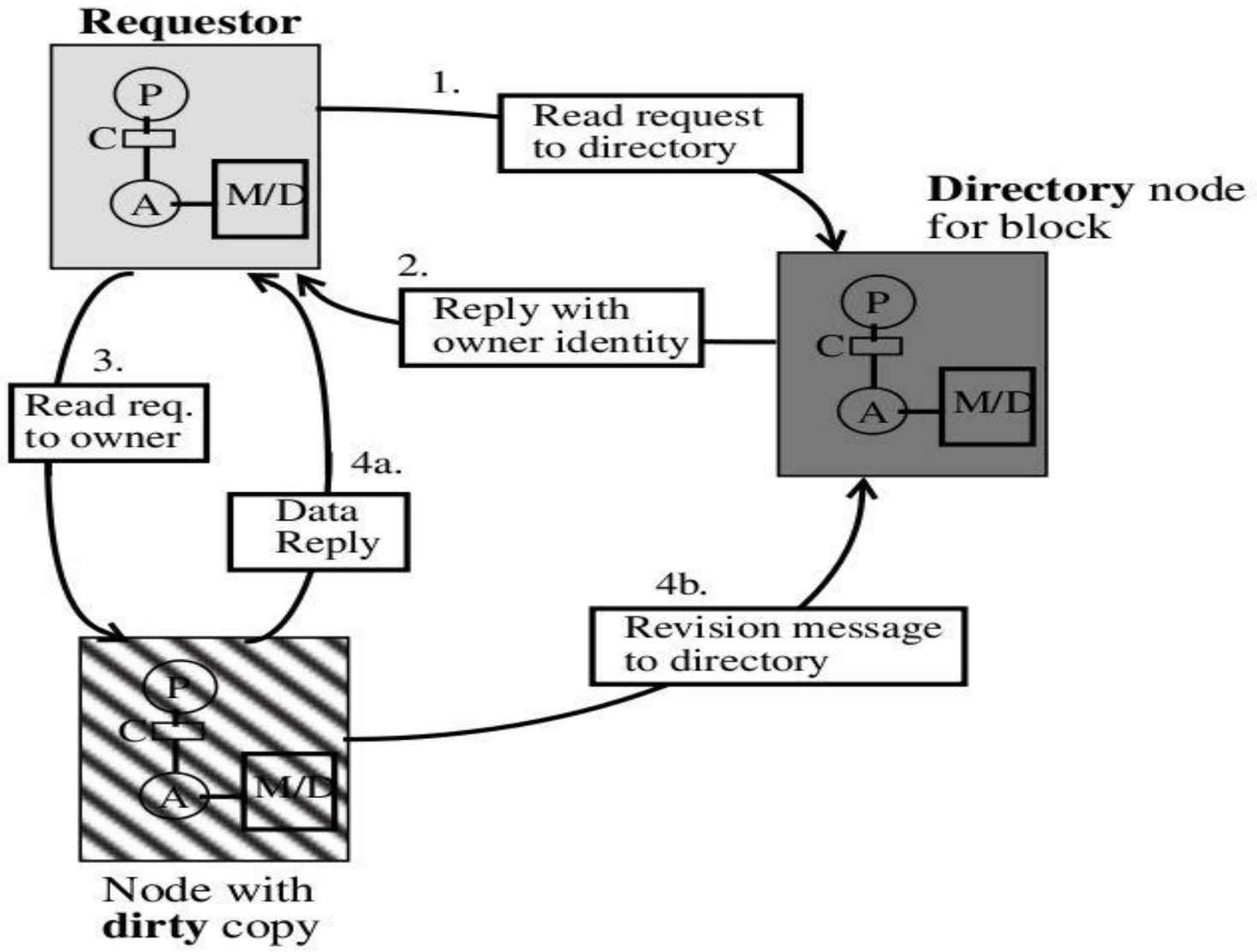
- Ειδοποίηση μόνο των επεξεργαστών που τους ενδιαφέρει ένα συγκεκριμένο block (*spam only those that care!*)

Directory-Based Cache Coherence (1)



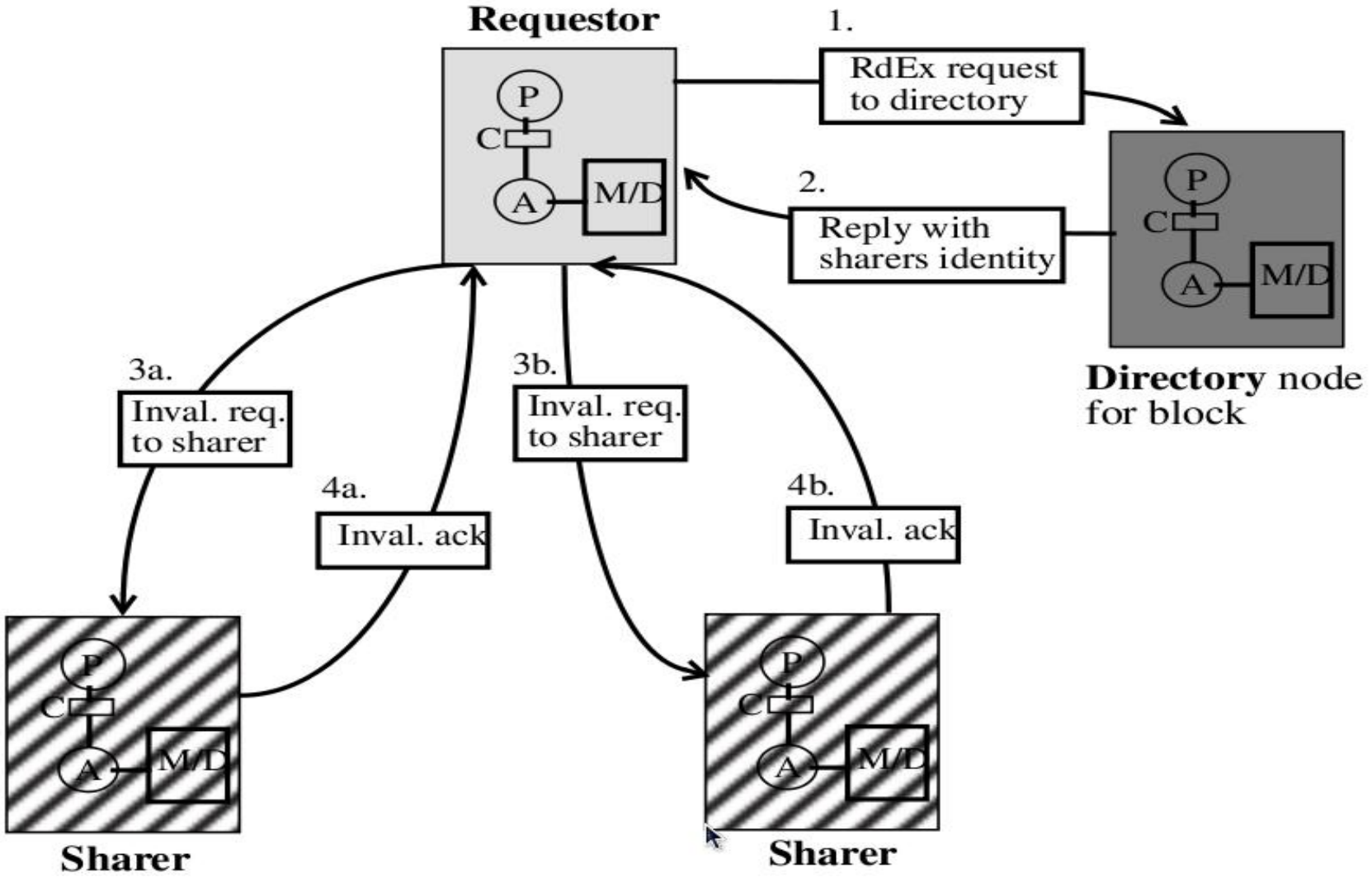
- Το cache block state δεν μπορεί να καθοριστεί πλέον παρακολουθώντας τα requests στο shared bus. (implicit determination)
- Καθορίζεται και διατηρείται σε ένα μέρος (directory) όπου τα requests μπορούν να απευθυνθούν και να το ανακαλύψουν. (explicit determination)
- Κάθε memory block έχει ένα directory entry
  - Book-keeping (ποιοι nodes έχουν αντίγραφα, το state του memory copy, …)
  - Όλα τα requests για το block πηγαίνουν στο directory.

Directory-Based Cache Coherence (2)



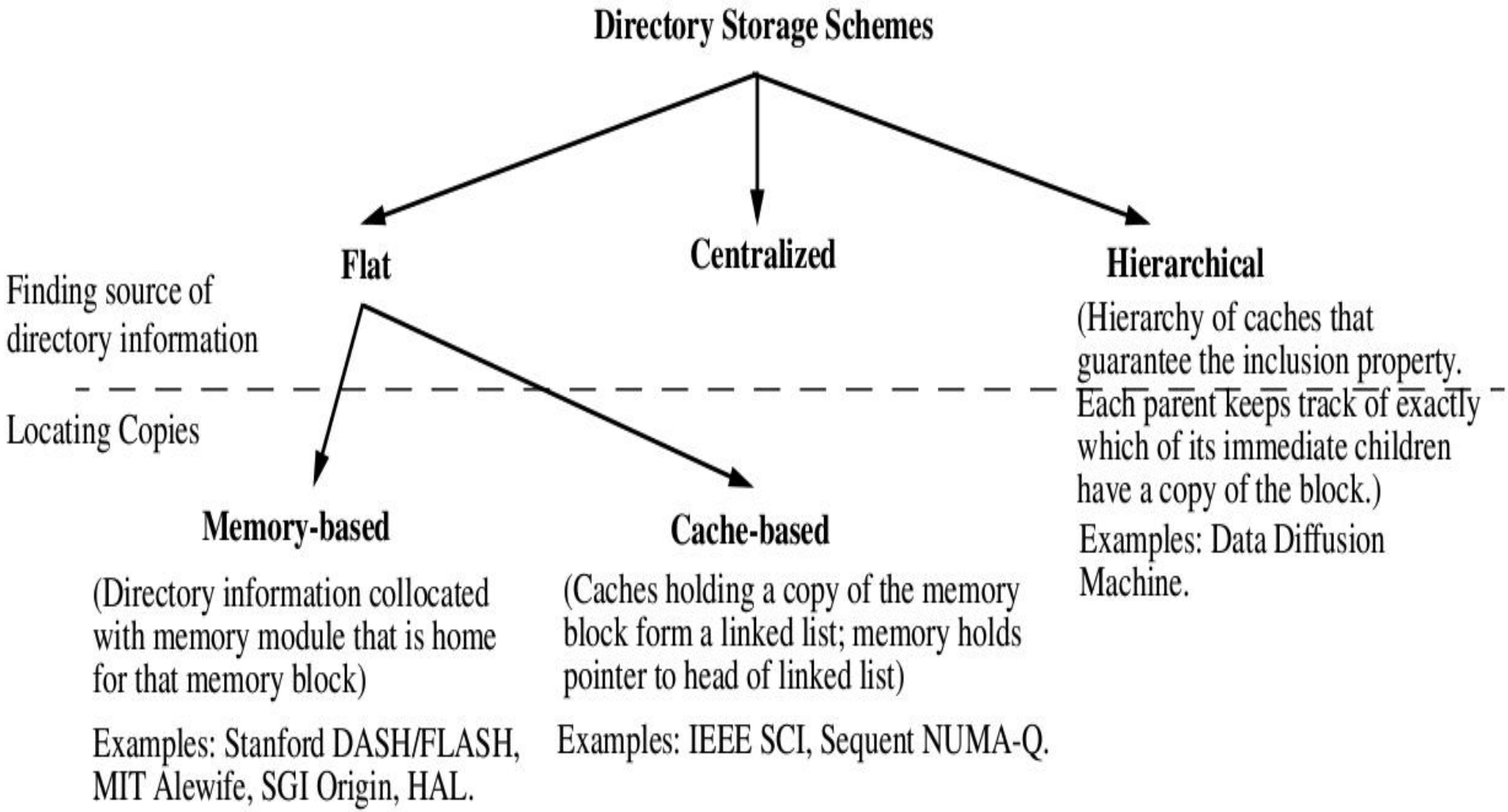
(a) Read miss to a block in dirty state

Directory-Based Cache Coherence (3)



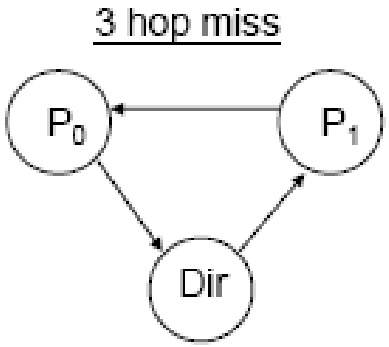
(b) Write miss to a block with two sharers

Directory Protocol Taxonomy



Directory-Based Cache Coherence (4)

- Directory Protocols
  - \+ Χαμηλότερη κατανάλωση bandwidth
  - \- Μεγαλύτερες καθυστερήσεις (latency)
- Δυο περιπτώσεις read miss :
  - **Unshared block** → get data from memory
    - » Bus : 2 hops (P0 → memory → P0)
    - » Directory : 2 hops (P0 → memory → P0)
  - **S/E block** → get data from processor (P1)
    - » Bus : 2 hops (P0 → P1 → P0) (υποθέτοντας ότι επιτρέπεται η cache-to-cache μεταφορά δεδομένων)
    - » Directory : **3 hops** (P0 → memory → P1→ P0)



- Η δεύτερη περίπτωση παρατηρείται αρκετά συχνά σε πολυεπεξεργαστικά συστήματα
  - Υψηλή πιθανότητα να έχει το block ένας επεξεργαστής