Transactional Memory

Τα προβλήματα του παράλληλου προγραμματισμού

- Εντοπισμός παραλληλισμού
  - χειροκίνητα (επισκόπηση)
  - αυτόματα (compiler)
- Έκφραση παραλληλισμού
  - low-level (π.χ. Pthreads, MPI)
  - high-level (π.χ. OpenMP, Cilk, Intel TBB, Galois, UPC κ.λπ.)
- Απεικόνιση
  - scheduling, creation, termination, etc.
  - operating system, runtime system
- Συγχρονισμός
  - εύκολος όπως ο coarse-grain
  - αποδοτικός όπως ο fine-grain
  - deadlock-free
  - composable
- Απαιτήσεις
  - κλιμακωσιμότητα
  - ευκολία προγραμματισμού
  - υψηλή παραγωγικότητα
  - ορθότητα
  - architectural awareness

Transactional Memory (TM)

- Memory transaction
  - μία ατομική και απομονωμένη ακολουθία λειτουργιών μνήμης
  - εμπνευσμένη από τις δοσοληψίες στις ΒΔ
- Atomicity (all or nothing)
  - στο commit, όλες οι εγγραφές μνήμης αποκτούν επίδραση «με τη μία»
  - σε περίπτωση abort, καμία από τις εγγραφές δεν έχουν επίδραση
- Isolation
  - κανένα άλλο τμήμα κώδικα δεν μπορεί να δει τις εγγραφές ενός transaction πριν το commit
- Consistency (serializability)
  - τα αποτελέσματα των transactions είναι συνεπή (ίδια με αυτά της σειριακής/σειριοποιημένης εκτέλεσης)
  - τα transactions φαίνονται ότι κάνουν commit σειριακά
  - η συγκεκριμένη σειρά ωστόσο δεν είναι εγγυημένη

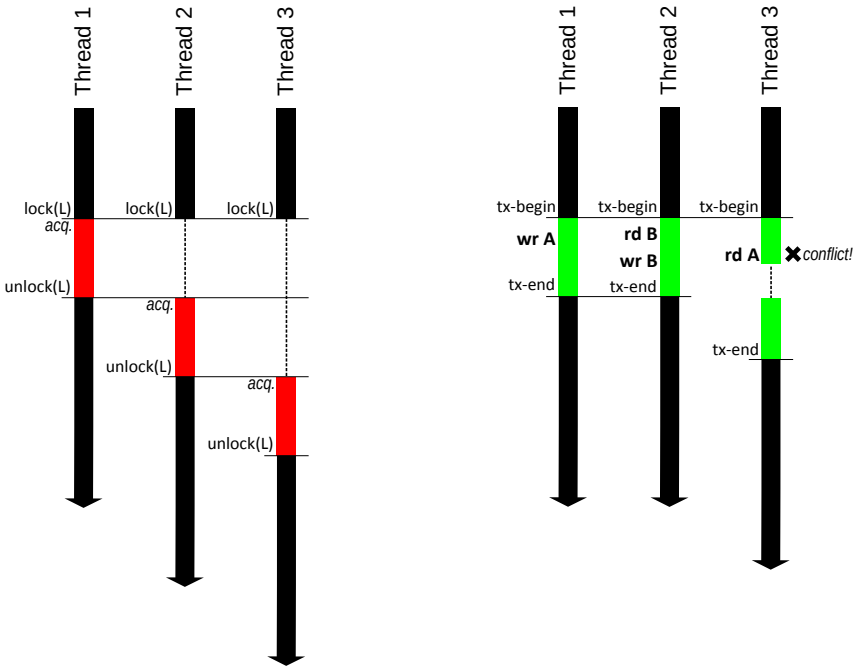
Προγραμματισμός με TM

```
void deposit(account, amount){
   lock(account);
     int t = bank.get(account);
     t = t + amount;
     bank.put(account, t);
   unlock(account);
}
```

```
void deposit(account, amount){
   atomic {
     int t = bank.get(account);
     t = t + amount;
     bank.put(account, t);
   }
}
```

- Περιγραφή συγχρονισμού σε υψηλό επίπεδο
  - ο προγραμματιστής λέει τι, όχι πώς
- Το υποκείμενο σύστημα υλοποιεί το συγχρονισμό
  - εξασφαλίζει atomicity, isolation & consistency

Αμοιβαίος αποκλεισμός vs. TM



- Αναμενόμενο κέρδος σε περίπτωση μη-conflict
- Επιβράδυνση σε conflicts (R-W ή W-W)

Πλεονεκτήματα TM

- Ευκολία προγραμματισμού
  - παραπλήσια με αυτή των coarse-grain locks
  - ο προγραμματιστής δηλώνει, το σύστημα υλοποιεί
- Επίδοση παραπλήσια με αυτή των fine-grain locks
  - εκμεταλλεύεται αυτόματα τον fine-grain ταυτοχρονισμό
  - δεν υπάρχει tradeoff ανάμεσα στην απόδοση & την ορθότητα
- Failure atomicity & recovery
  - δεν «χάνονται» locks όταν ένα thread αποτυγχάνει
  - failure recovery = transaction abort + restart
- Composability
  - σύνθεση επιμέρους ατομικών λειτουργιών / software modules σε μία ενιαία ατομική λειτουργία
  - ασφαλής & κλιμακώσιμη

Παράδειγμα: Java 1.4 HashMap

- Map: key → value

```
public Object get(Object key)  {
   int idx = hash(key);              // Compute hash
   HashEntry e = buckets[idx];       // to find bucket
   while (e != null)  {              // Find element in bucket
      if (equals(key, e.key))
            return e.value;
      e = e.next;
    }
   return null;
 }
```

- not thread-safe

Synchronized HashMap

- Η λύση της Java 1.4: synchronized layer
  - ρητό, coarse-grain locking από τον προγραμματιστή

```
public Object get(Object key) {
    synchronized (mutex) {  // mutex guards all accesses to map m
        return m.get(key);
    }
}
```

- Coarse-grain synchronized HashMap
  - Pros: thread-safe, εύκολο στον προγραμματισμό
  - Cons: περιορίζει τον (όποιον) ταυτοχρονισμό, χαμηλή κλιμακωσιμότητα
    - μόνο ένα thread μπορεί να επεξεργάζεται το HashMap κάθε φορά

Concurrent HashMap (Java 5)

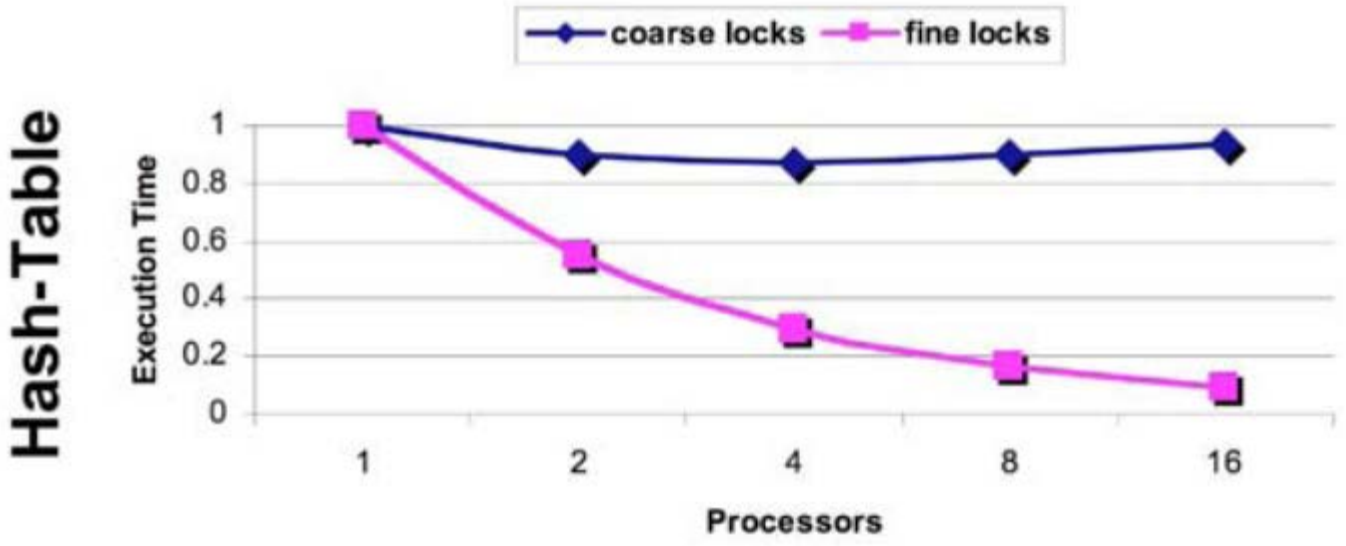
```
public Object get(Object key) {
   int hash = hash(key);
   // Try first without locking...
   Entry[] tab = table;
   int index = hash & (tab.length - 1);
   Entry first = tab[index];
   Entry e;

   for (e = first; e != null; e = e.next) {
     if (e.hash == hash && eq(key, e.key)) {
       Object value = e.value;
       if (value != null)
         return value;
       else
         break;
     }
   }
   ...
```

```
   ...
   // Recheck under synch if key not there or interference
   Segment seg = segments[hash & SEGMENT_MASK];
   synchronized(seg) {
     tab = table;
     index = hash & (tab.length - 1);
     Entry newFirst = tab[index];
     if (e != null || first != newFirst) {
       for (e = newFirst; e != null; e = e.next) {
         if (e.hash == hash && eq(key, e.key))
           return e.value;
       }
     }
     return null;
   }
 }
```

- Fine-grain synchronized concurrent HashMap
  - Pros: fine-grain παραλληλισμός, ταυτόχρονες αναγνώσεις
  - Cons: περίπλοκο & επιρρεπές σε λάθη

Επίδοση: Locks



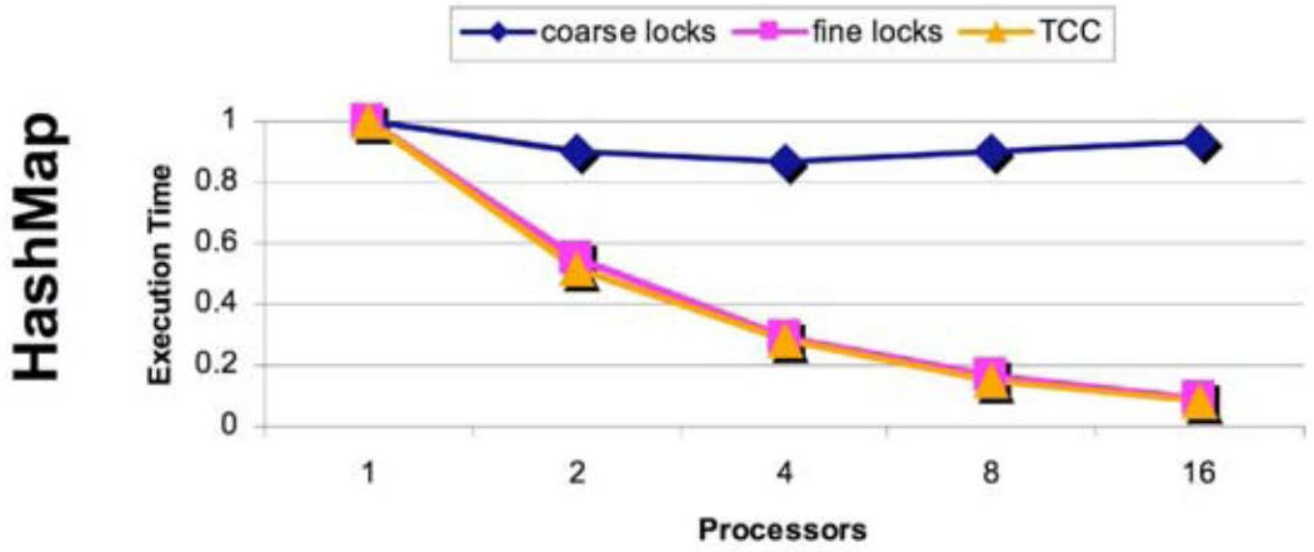
Transactional HashMap

- Απλά εσωκλείουμε τη λειτουργία σε ένα atomic block
  - το σύστημα εξασφαλίζει την ατομικότητα

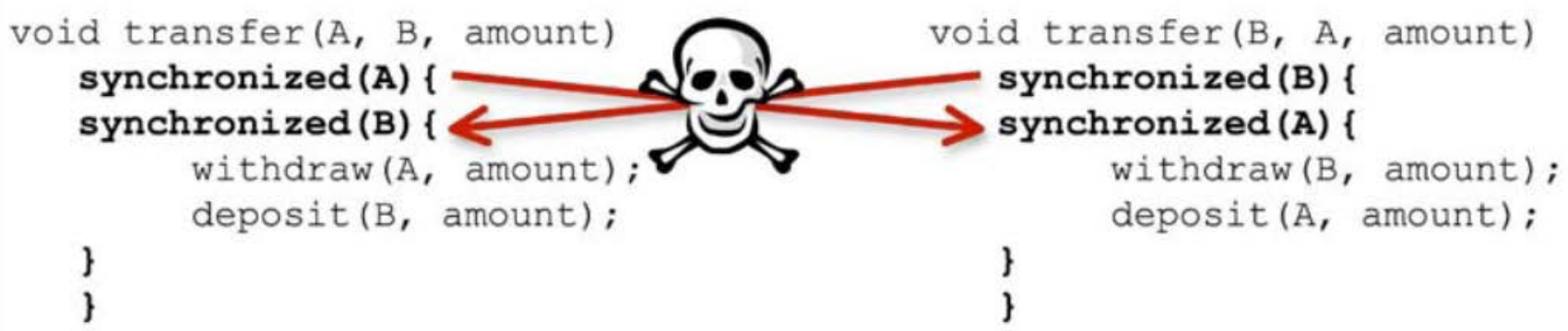
```
public Object get(Object key) {
    atomic {                          // System guarantees atomicity
        return m.get(key);
    }
}
```

- Transactional HashMap
  - Pros: thread-safe, εύκολο στον προγραμματισμό
  - Καλή επίδοση & κλιμακωσιμότητα?
    - Εξαρτάται από την υλοποίηση και το σενάριο εκτέλεσης, αλλά τυπικά ναι

Επίδοση: Locks vs Transactions

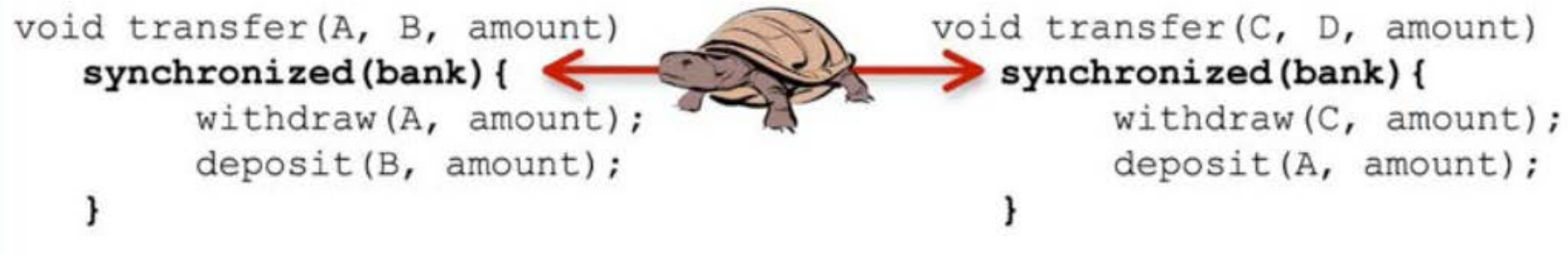


Composability: Locks



- Η σύνθεση lock-based κώδικα είναι δύσκολη
  - Σκοπός: απόκρυψη ενδιάμεσης κατάστασης κατά τη μεταφορά
  - Χρειαζόμαστε καθολική πολιτική για το κλείδωμα
    - δεν μπορεί πάντα να αποφασιστεί a priori
- *Fine-grain locking*: μπορεί να οδηγήσει σε deadlocks

Composability: Locks



- Η σύνθεση lock-based κώδικα είναι δύσκολη
  - Σκοπός: απόκρυψη ενδιάμεσης κατάστασης κατά τη μεταφορά
  - Χρειαζόμαστε καθολική πολιτική για το κλείδωμα
    - δεν μπορεί πάντα να αποφασιστεί a priori
- *Fine-grain locking*: μπορεί να οδηγήσει σε deadlocks
- *Coarse-grain locking*: μηδενικός ταυτοχρονισμός

Composability: Transactions

```
void transfer(A, B, amount)
    atomic{
        withdraw(A, amount);
        deposit(B, amount);
    }
```

```
void transfer(B, A, amount)
    atomic{
        withdraw(B, amount);
        deposit(A, amount);
    }
```

- Οι ατομικές λειτουργίες συντίθενται ομαλά, χωρίς να περιορίζεται ο ταυτοχρονισμός
  - ο προγραμματιστής δηλώνει την καθολική πρόθεση (ατομική μεταφορά)
    - χωρίς να χρειάζεται να ξέρει κάποια καθολική πολιτική κλειδώματος
- Το σύστημα διαχειρίζεται τον ταυτοχρονισμό με τον καλύτερο δυνατό τρόπο
  - σειριοποίηση για transfer(A, B, $100) & transfer(B, A, $200)
  - ταυτοχρονισμός για transfer(A, B, $100) & transfer(C, D, $200)

Υλοποίηση TM

- Τα TM συστήματα πρέπει να παρέχουν ατομικότητα και απομόνωση
  - χωρίς να θυσιάζεται ο ταυτοχρονισμός
- Παράμετροι λειτουργίας
  - Data versioning (ή version management)
  - Conflict detection
  - Conflict resolution
- Επιλογές
  - Hardware transactional memory (HTM)
  - Software transactional memory (STM)
  - Hybrid transactional memory
    - Hardware accelerated STMs
    - Dual-mode systems

Data Versioning

- Διαχείριση uncommitted (νέων) και committed (παλιών) εκδόσεων δεδομένων για ταυτόχρονα transactions
- Eager versioning (undo-log based)
  - απευθείας ενημέρωση μνήμης
  - διατήρηση undo πληροφορίας σε log

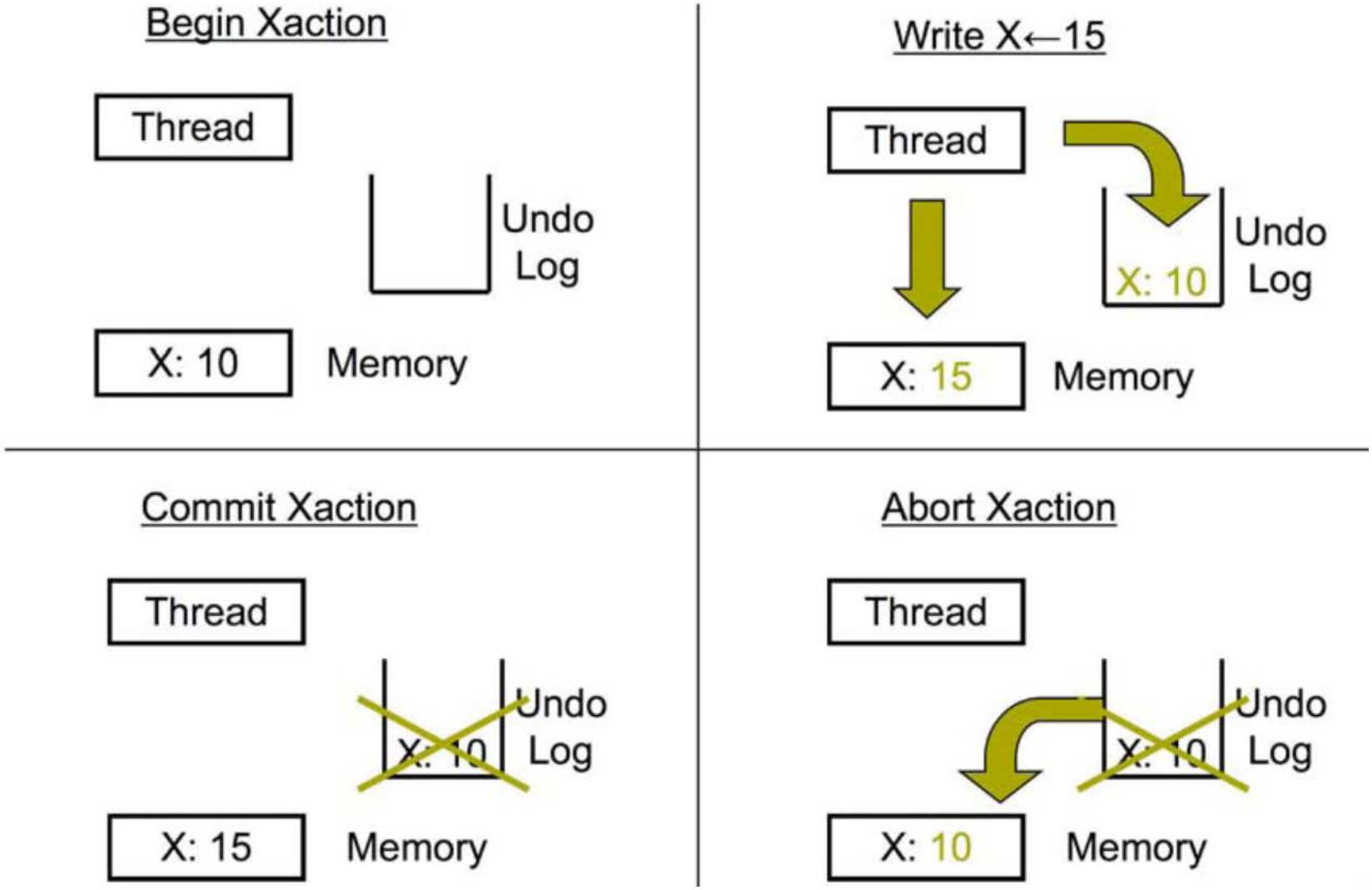
  (+) Γρήγορο commit

  (-) Αργό abort, ζητήματα fault tolerance
- Lazy versioning (write-buffer based)
  - buffering νέων δεδομένων σε write-buffer μέχρι το commit
  - πραγματική ενημέρωση μνήμης κατά το commit

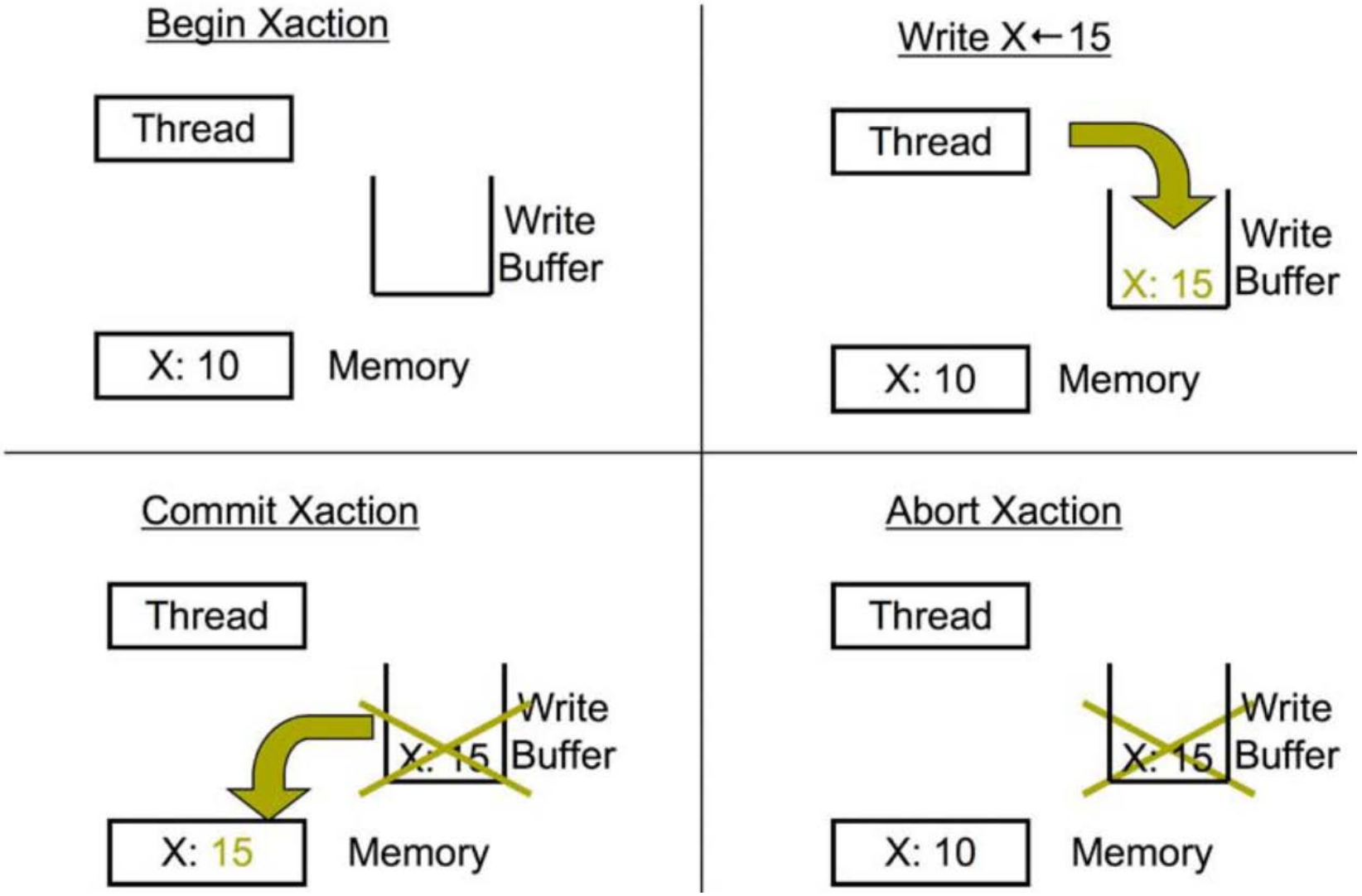
  (+) Γρήγορο abort, fault tolerant

  (-) Αργά commits

Eager versioning



Lazy versioning

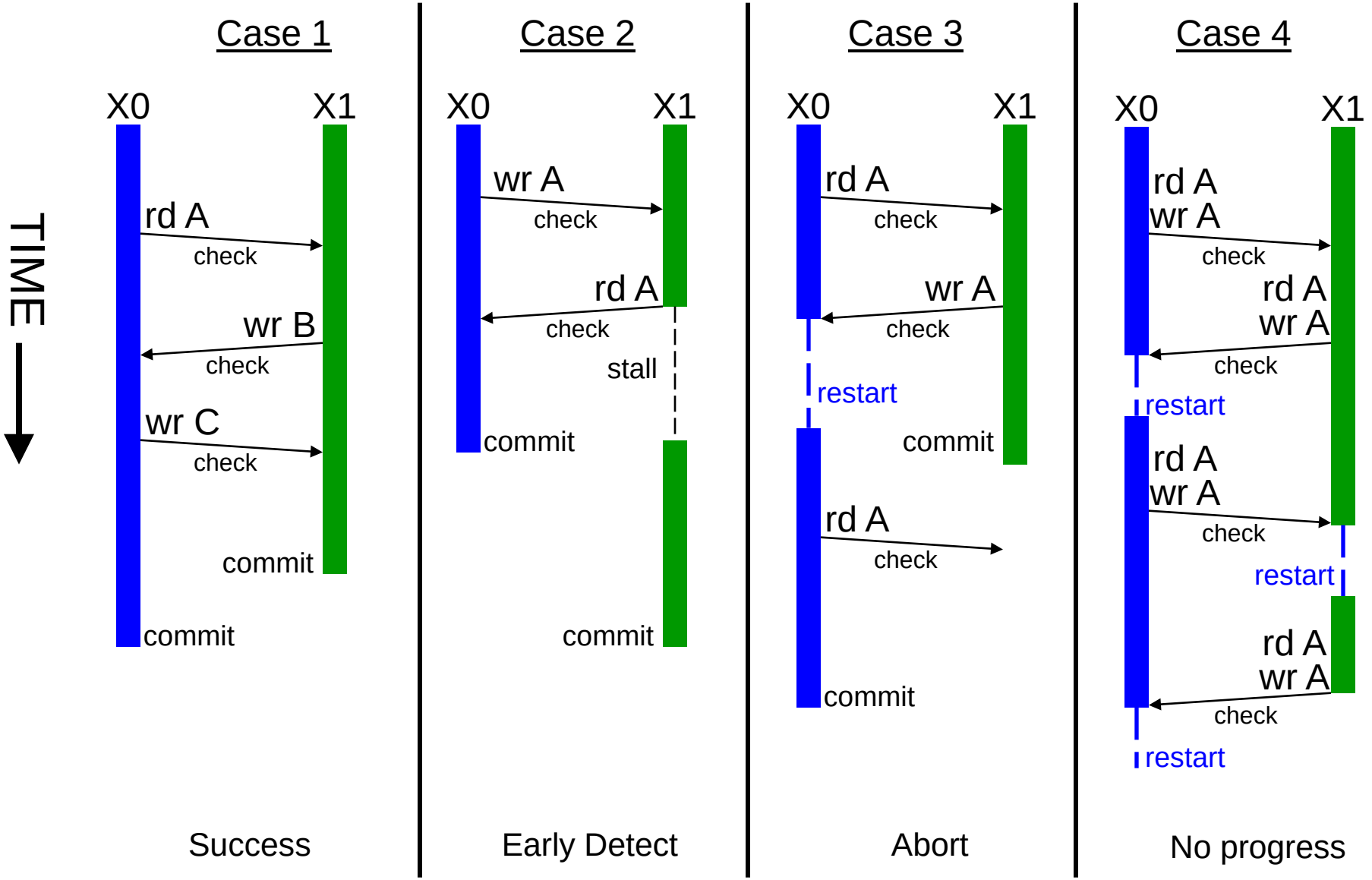


Conflict detection

- Ανίχνευση και διαχείριση conflicts ανάμεσα σε transactions
  - Read-Write και (συχνά) Write-Write conflicts
  - Πρέπει να παρακολουθούνται το read-set & write-set ενός transaction
    - Read-set: οι διευθύνσεις που διαβάζονται εντός του transaction
    - Write-set: οι διευθύνσεις που γράφονται εντός του transaction

- Pessimistic (eager) detection
  - Έλεγχος για conflicts σε κάθε load ή store
    - SW: SW barriers με locks και/ή version numbers
    - HW: έλεγχος μέσω του coherence protocol
  - Χρήση contention manager για να αποφασίσει να κάνει stall ή abort
    - διαφορετικές πολιτικές ανάθεσης προτεραιότητας
    - στόχος: to make the common case fast

Pessimistic detection

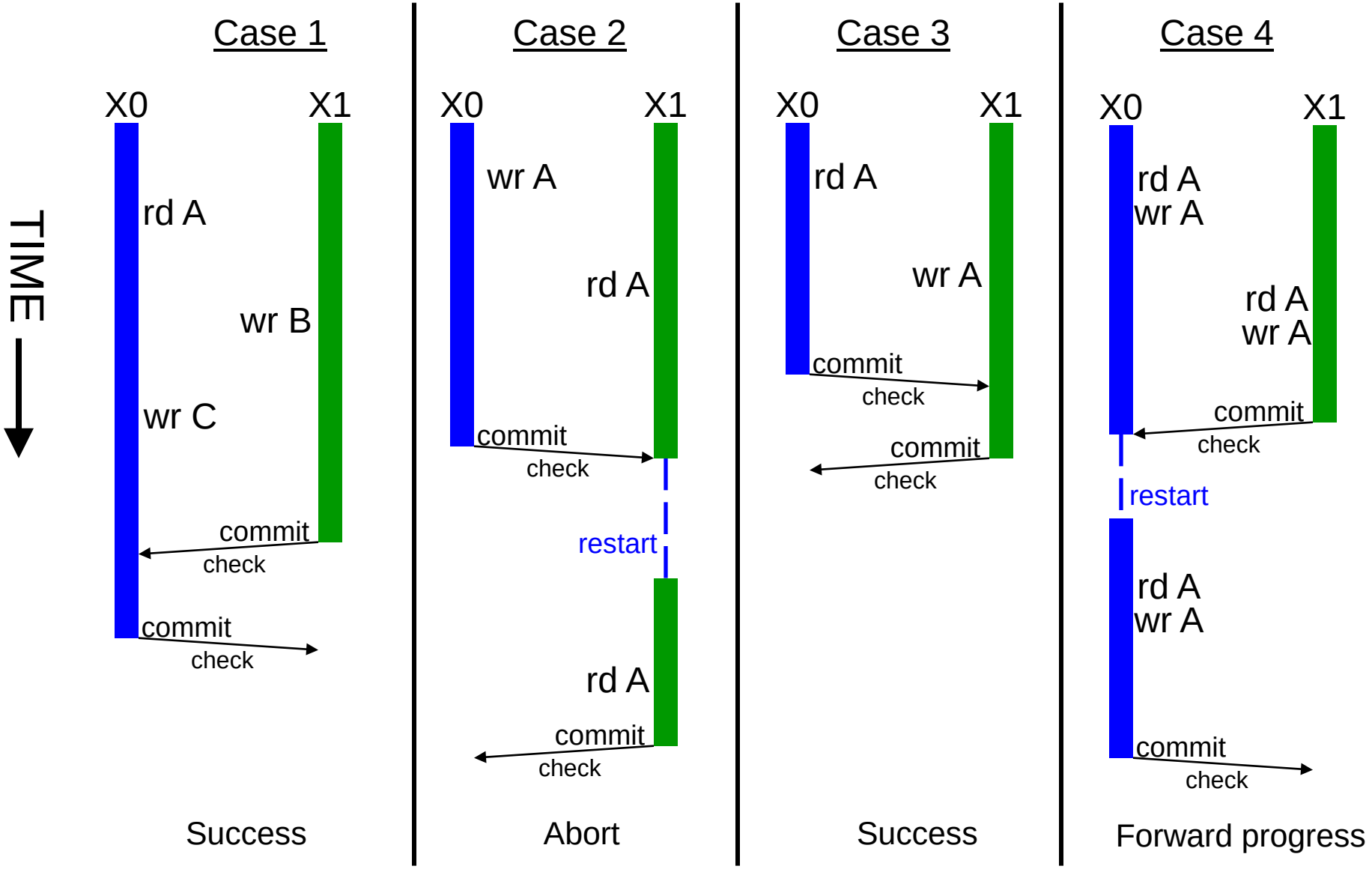


Conflict detection

- Optimistic (lazy) detection
  - Τα conflicts ανιχνεύονται όταν ένα transaction επιχειρήσει να κάνει commit
    - SW: validate τα write/read-set με χρήση locks ή version numbers
    - HW: validate το write-set χρησιμοποιώντας το coherence protocol
  - Σε περίπτωση conflict, προτεραιότητα στο committing transaction
    - τα υπόλοιπα transactions μπορούν να κάνουν abort αργότερα
    - σε περίπτωση conflict ανάμεσα σε committing transactions, την προτεραιότητα την αποφασίζει ο contention manager

- Σημείωση: optimistic & pessimistic σχήματα ταυτόχρονα είναι εφικτά
  - Πολλά STMs είναι optimistic στα reads και pessimistic στα writes

Optimistic detection



Conflict Detection Tradeoffs

- Pessimistic CD

  (+) ανιχνεύει τα conflicts νωρίς

  - αναιρεί λιγότερη δουλειά, μετατρέπει μερικά aborts σε stalls

  (-) δεν εγγυάται την πρόοδο προς τα εμπρός, πολλά aborts σε κάποιες περιπτώσεις

  (-) locking issues (SW), fine-grain communication (HW)

- Optimistic CD

  (+) εγγυάται την πρόοδο προς τα εμπρός

  (+) λιγότερα εν δυνάμει conflicts, λιγότερο locking (SW), bulk communication (HW)

  (-) ανιχνεύει τα conflicts αργά, προβλήματα δικαιοσύνης

TM Implementation Space (παραδείγματα)

- HTM συστήματα
  - Lazy + optimistic: Stanford TCC
  - Lazy + pessimistic: MIT LTM, Intel VTM, Sun's Rock
  - Eager + pessimistic: Wisconsin LogTM

- STM συστήματα
  - Lazy + optimistic (rd/wr): Sun TL2
  - Lazy + optimistic (rd) / pessimistic (wr): MS OSTM
  - Eager + optimistic (rd) / pessimistic (wr): Intel STM
  - Eager + pessimistic (rd/wr): Intel STM

Sun Rock's HTM

- «Best-effort» HTM
  - μικρές προσθήκες σε υλικό
  - η εκτέλεση των transactions περιορίζεται από δομές του υλικού
  - μη-εγγυημένη η πρόοδος προς τα εμπρός
- 2 εντολές
  - chkpt failPC
  - <critical section>
  - commit
- Είτε η κρίσιμη περιοχή εκτελείται ατομικά, είτε κάνει abort και μεταβαίνει στο failPC
- Lazy VM
- Eager CD

Παράδειγμα

```
atomic {
  a++;
  c = a + b;
}
```

```
 retry: chkpt retry          // Naïve repeated retry
                r0 = a       // Read a into register
                r0 = r0 + 1  // Arithmetic
                a = r0       // Write new value of a
                r1 = a       // Read new value of a
                r2 = b       // Read b
                r3 = r1 + r2 // Arithmetic
                c = r3       // Write c
          commit             // Commit if appears atomic
```

Παράδειγμα

```
retry: chkpt retry          // Checkpoint registers
               r0 = a       // Add a to read-set
               r0 = r0 + 1  //
               a = r0       // Add a to write-set
                            // Buffer old/new values of a
               r1 = a       // Read new value of a
               r2 = b       // Add b to read-set
               r3 = r1 + r2 //
               c = r3       // Add c to write-set
                            // Buffer old/new values of c
          commit            // Commit if appears atomic
```

Αναπαράσταση read/write sets?
Buffer old/new values?
Conflict detection?

Cache bits & Writebuffer addresses
Register chkpt & Writebuffer values
Cache Coherence Protocol

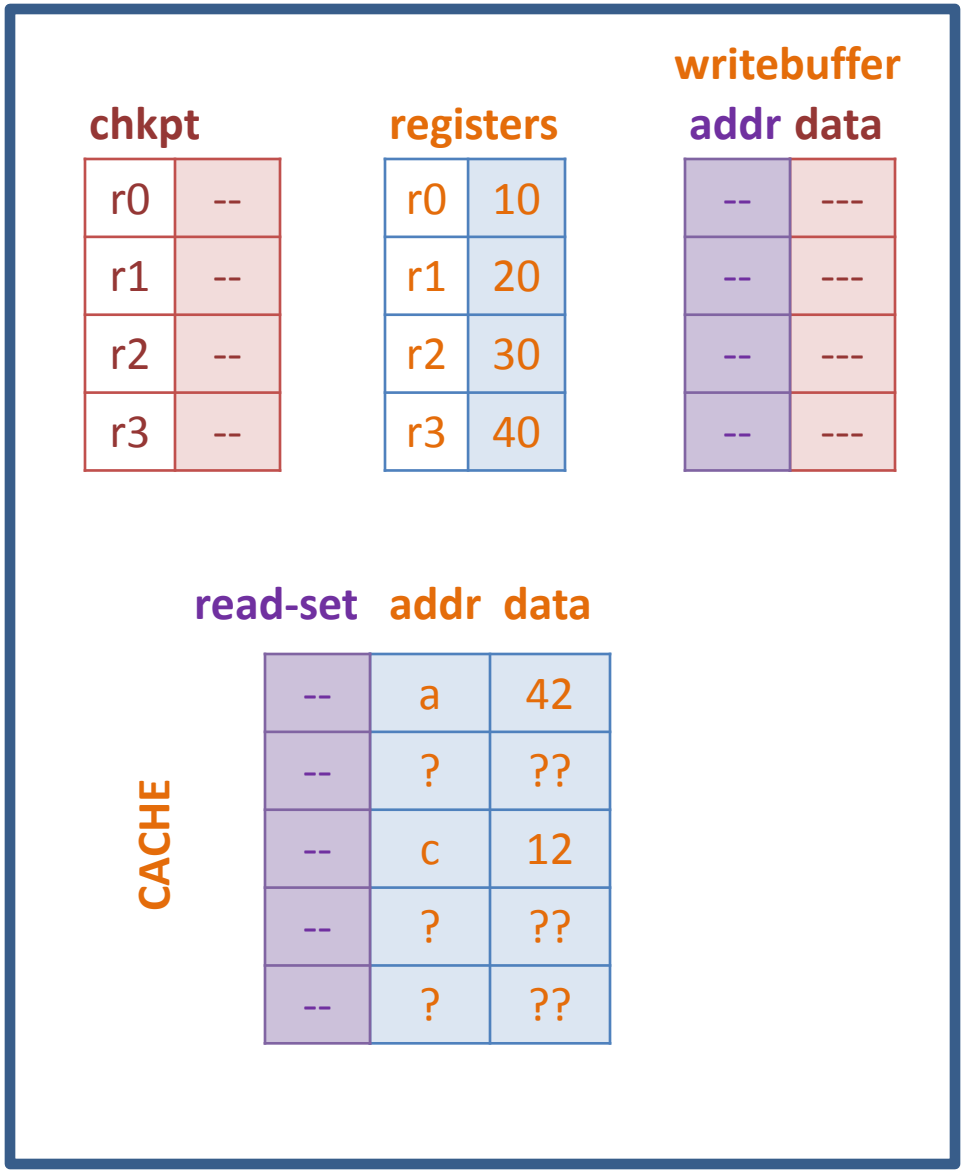
Παράδειγμα

Αναπαράσταση read/write sets
- Read: R-bit in (L1) cache
- Write: writebuffer addresses

Buffer old/new values
- Checkpoint old register values
- Old memory values in L1 cache
- New memory values in writebuffer

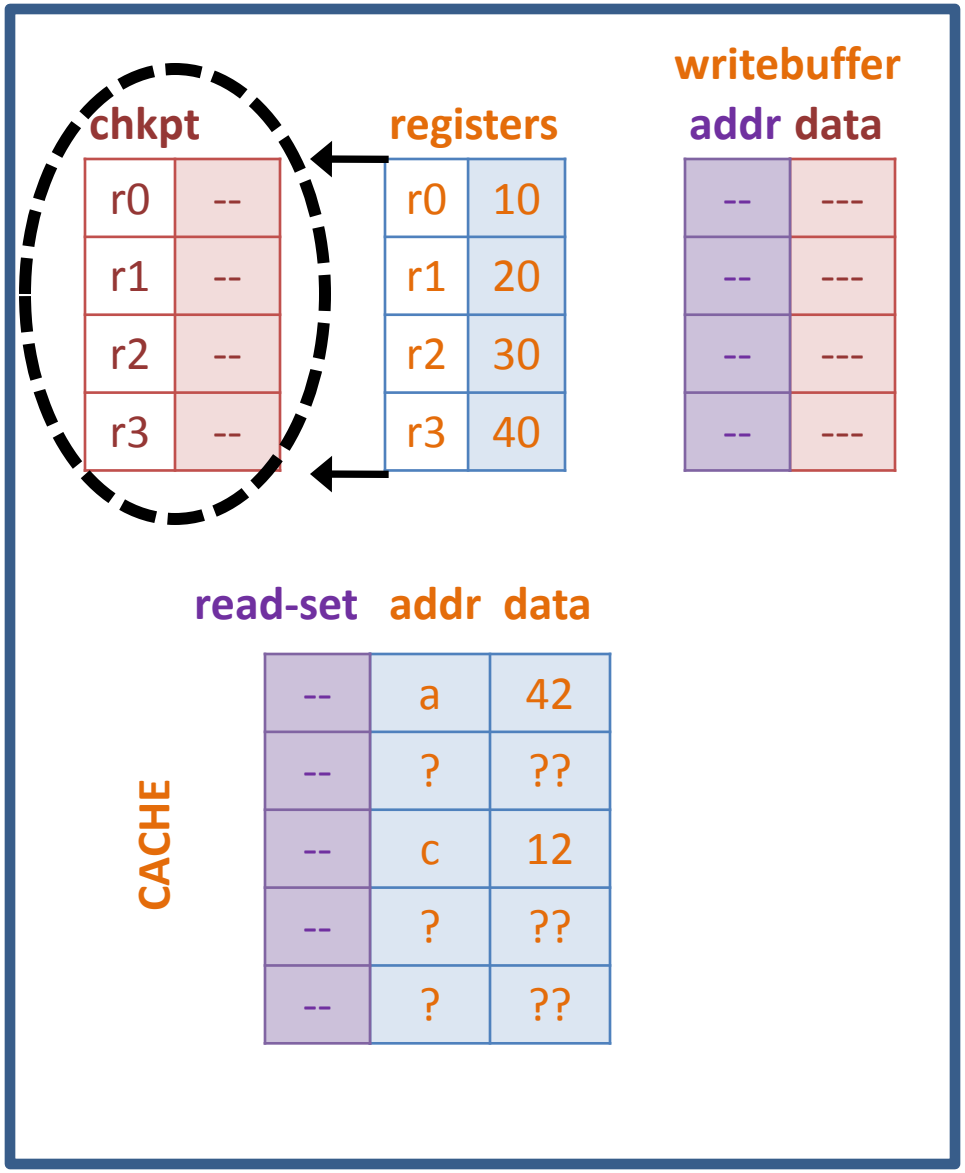
Conflict detection
- Cache Coherence Protocol

Μικρές προσθήκες σε υλικό!



Παράδειγμα

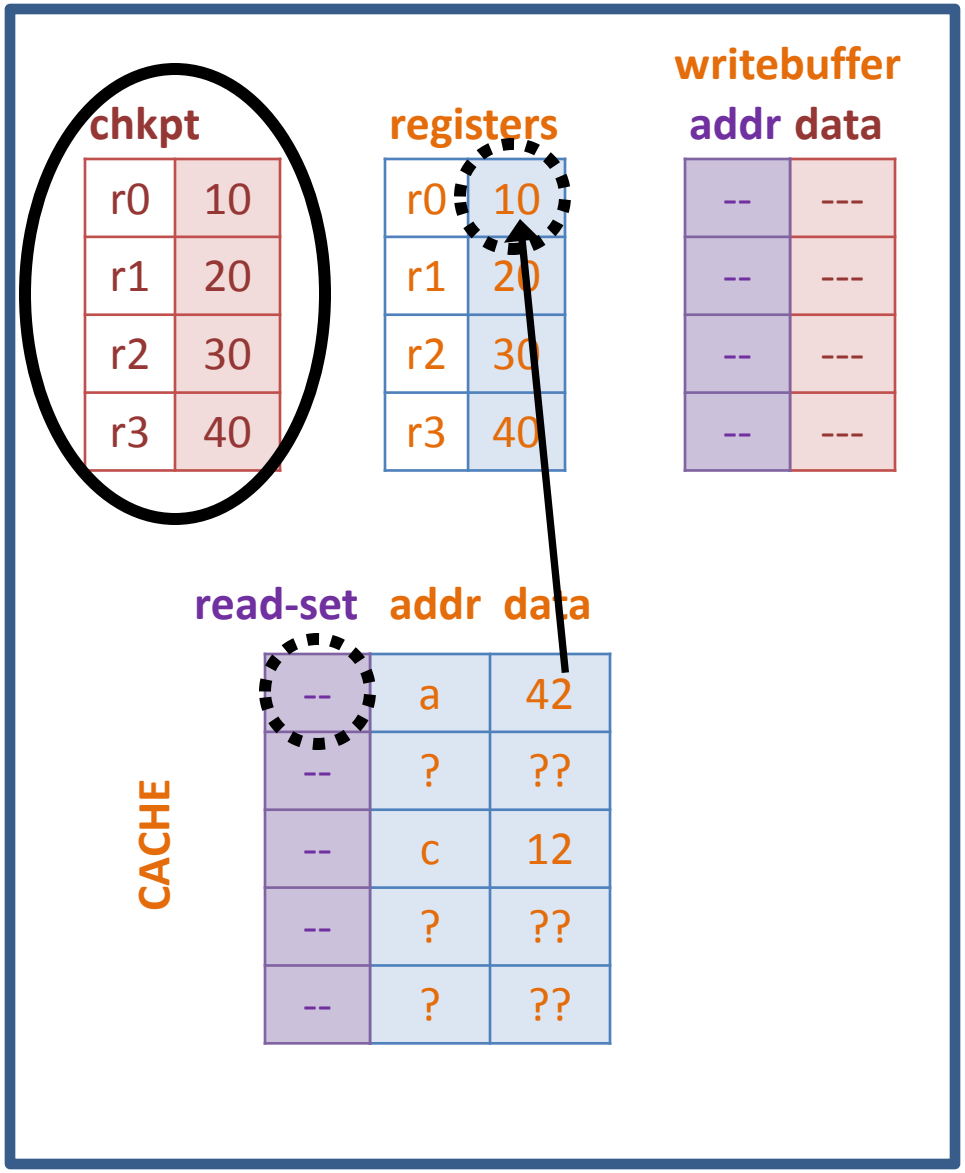
```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```



ΜΩΒ: read/write sets
ΚΟΚΚΙΝΟ: buffer old/new values
ΠΡΑΣΙΝΟ: conflict detection

Παράδειγμα

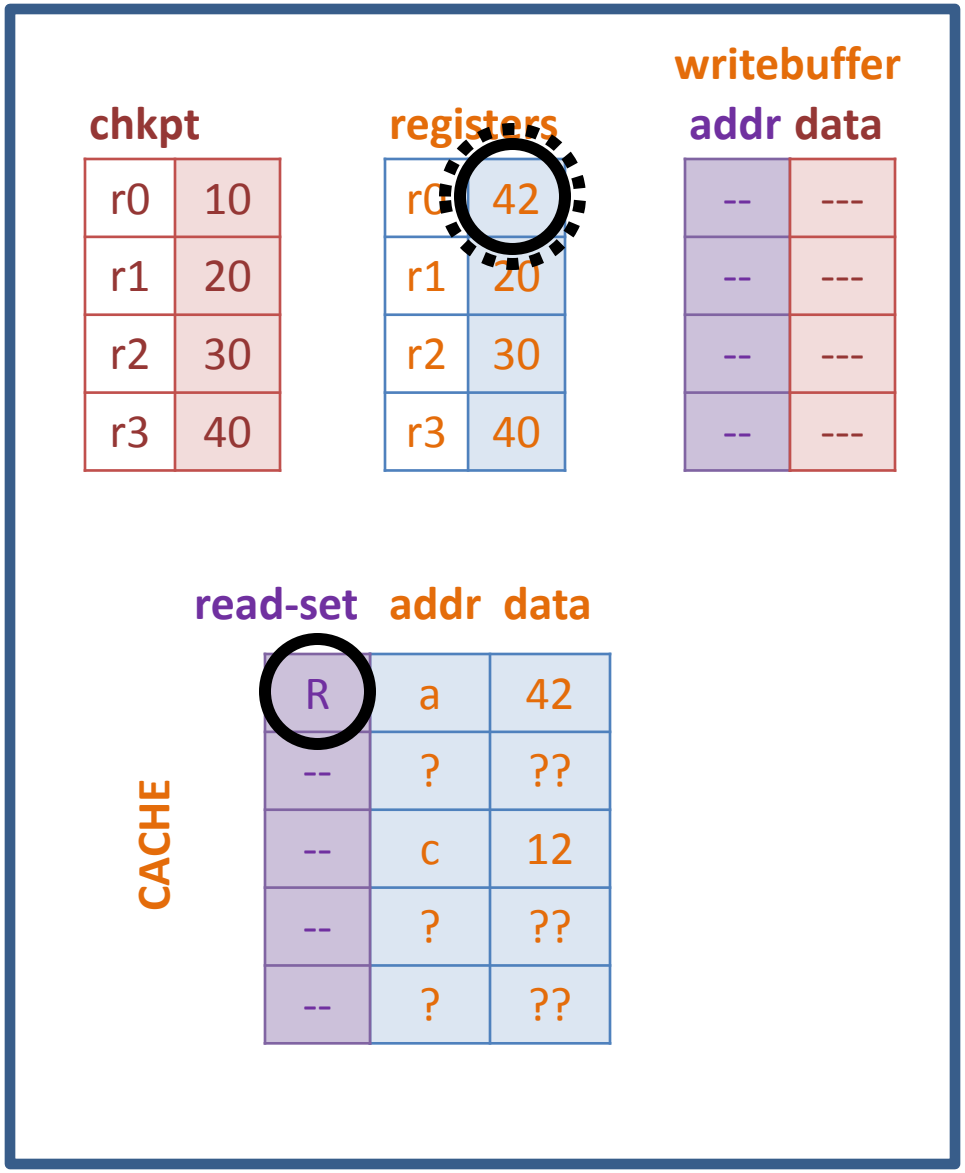
```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```



ΜΩΒ: read/write sets
ΚΟΚΚΙΝΟ: buffer old/new values
ΠΡΑΣΙΝΟ: conflict detection

Παράδειγμα

```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```

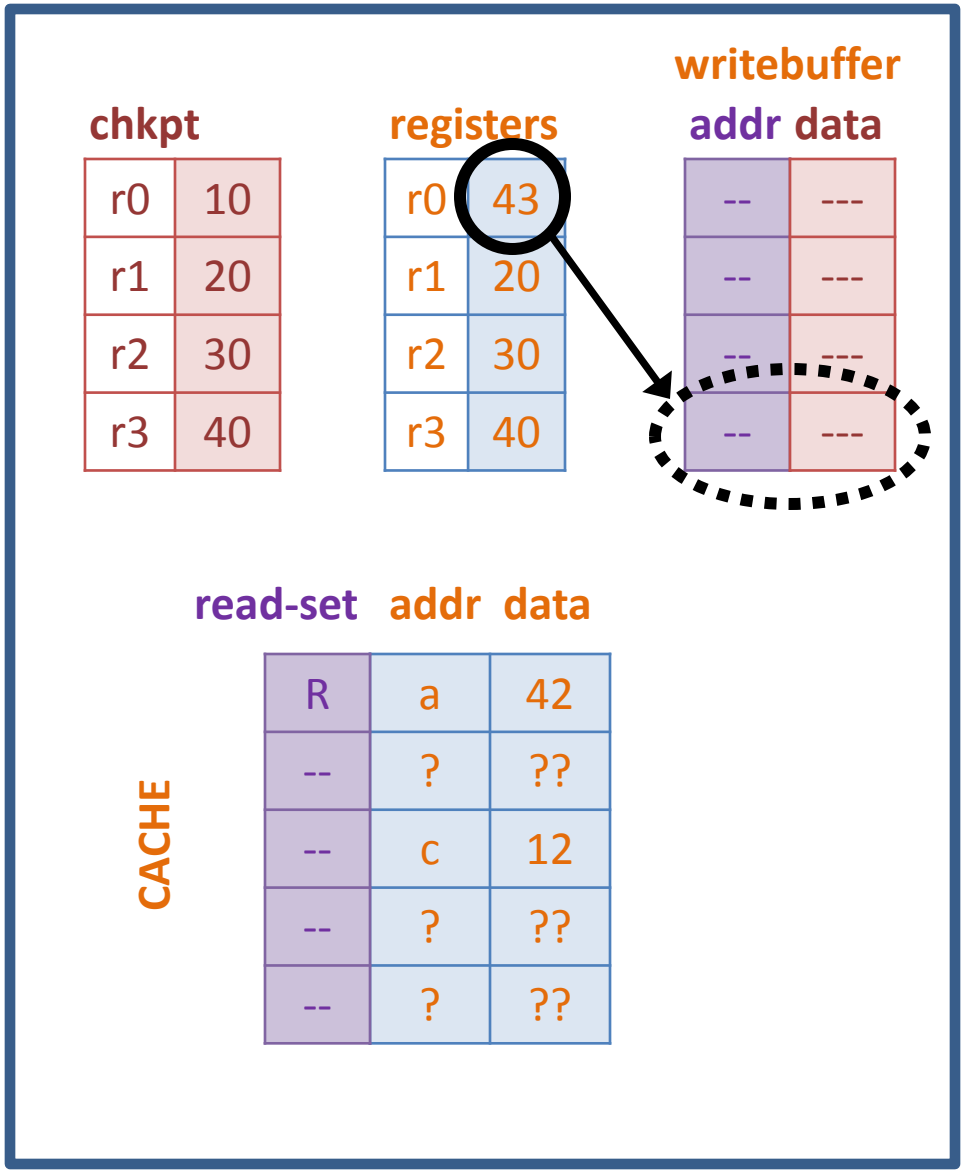


CORE 0

ΜΩΒ: read/write sets
ΚΟΚΚΙΝΟ: buffer old/new values
ΠΡΑΣΙΝΟ: conflict detection

Παράδειγμα

```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```



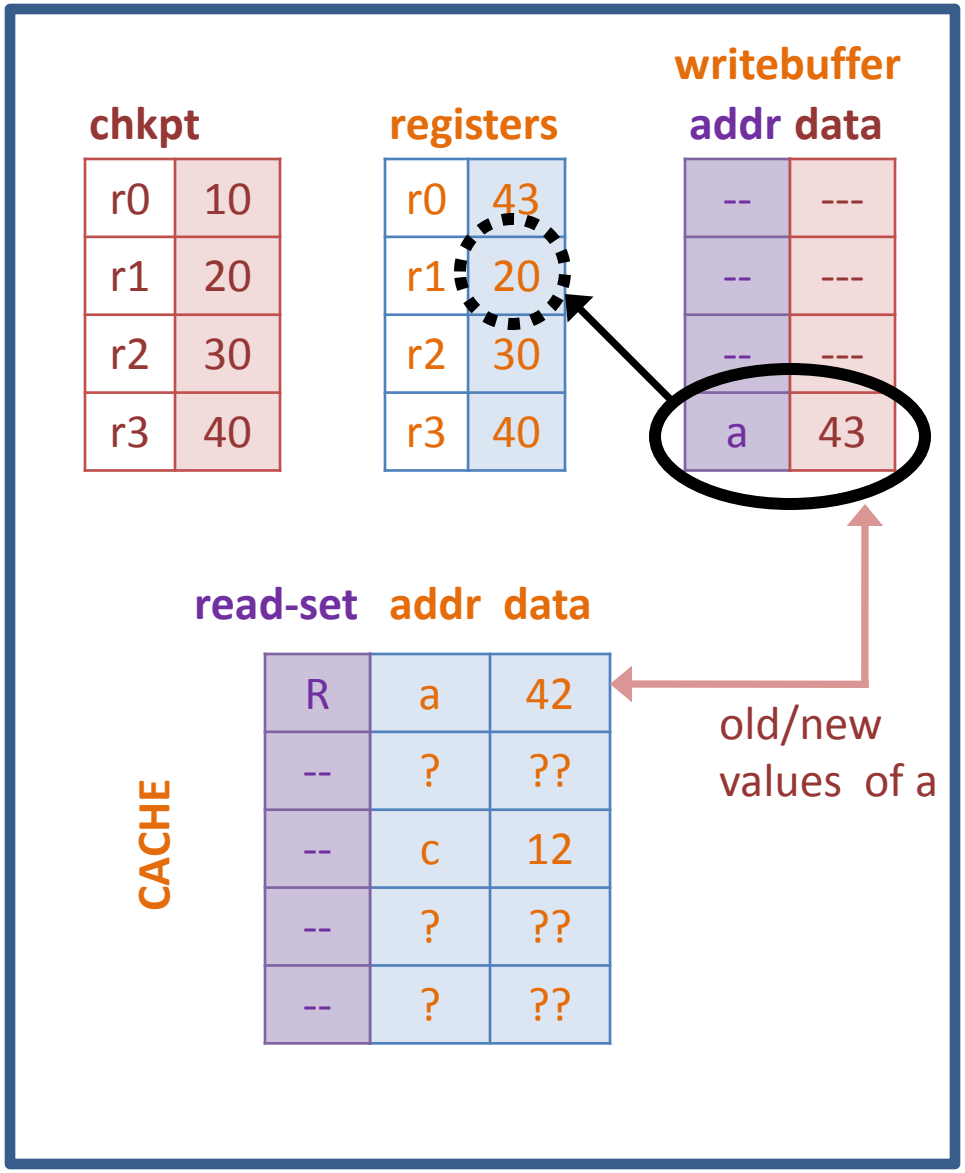
ΜΩΒ: read/write sets

ΚΟΚΚΙΝΟ: buffer old/new values

ΠΡΑΣΙΝΟ: conflict detection

Παράδειγμα

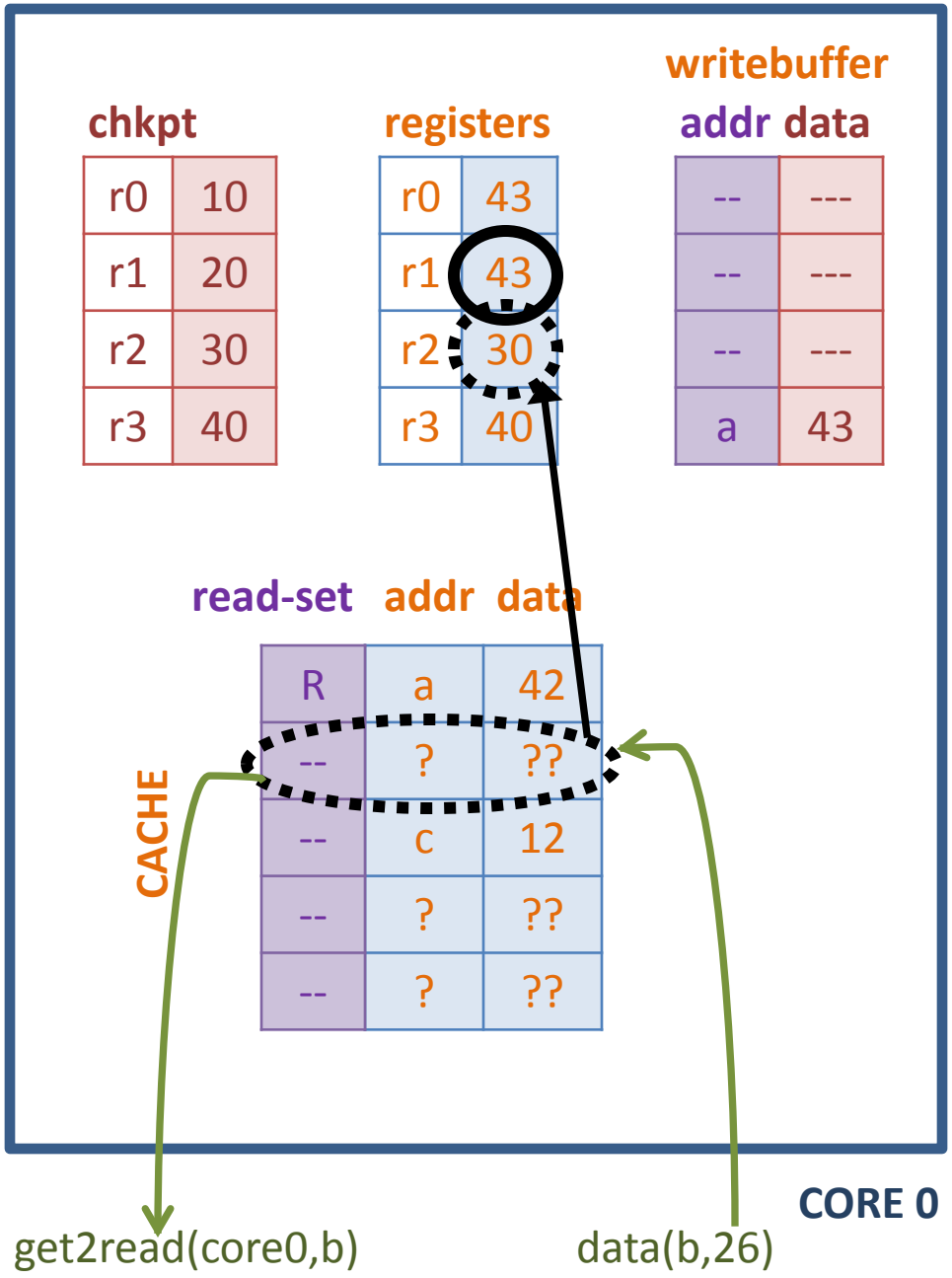
```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```



ΜΩΒ: read/write sets
ΚΟΚΚΙΝΟ: buffer old/new values
ΠΡΑΣΙΝΟ: conflict detection

Παράδειγμα

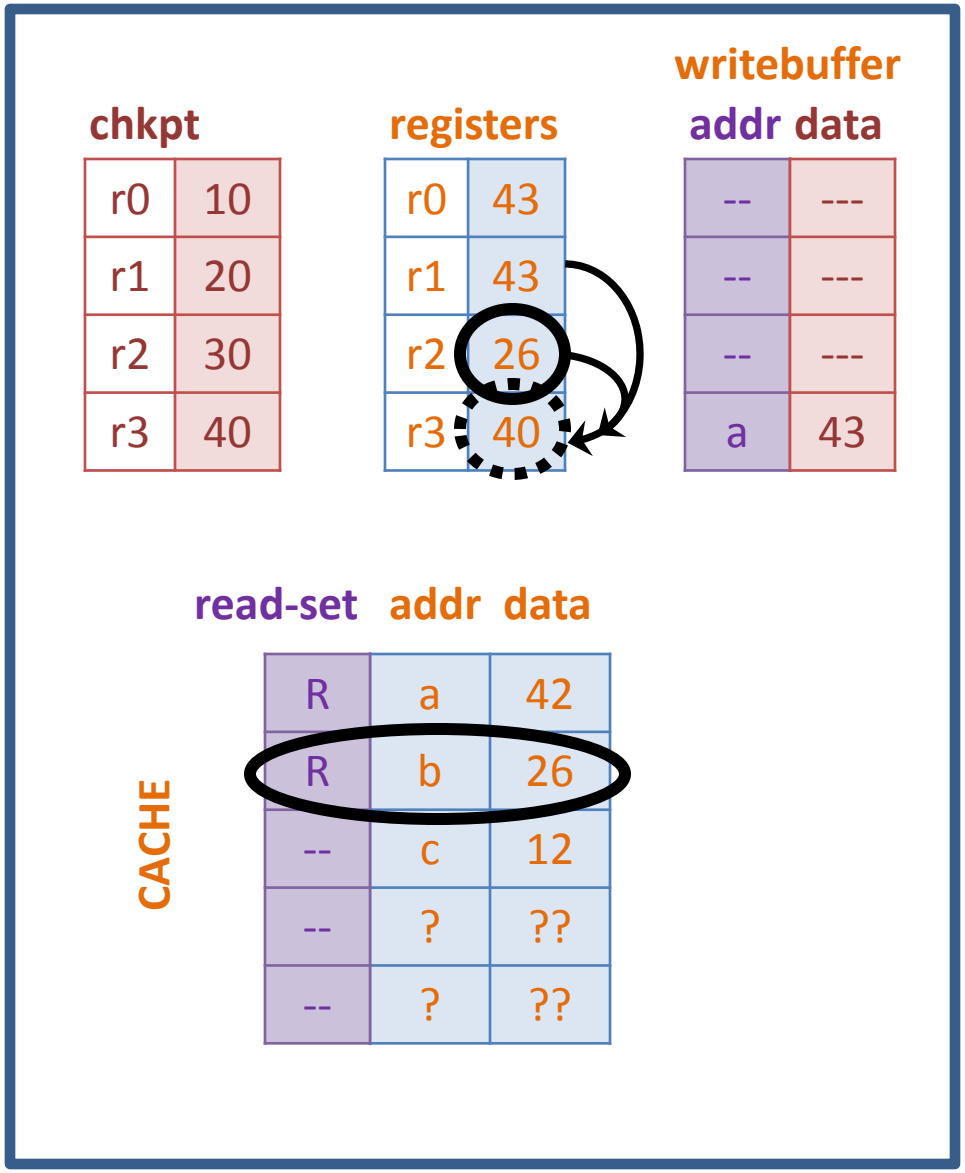
```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```



ΜΩΒ: read/write sets
ΚΟΚΚΙΝΟ: buffer old/new values
ΠΡΑΣΙΝΟ: conflict detection

Παράδειγμα

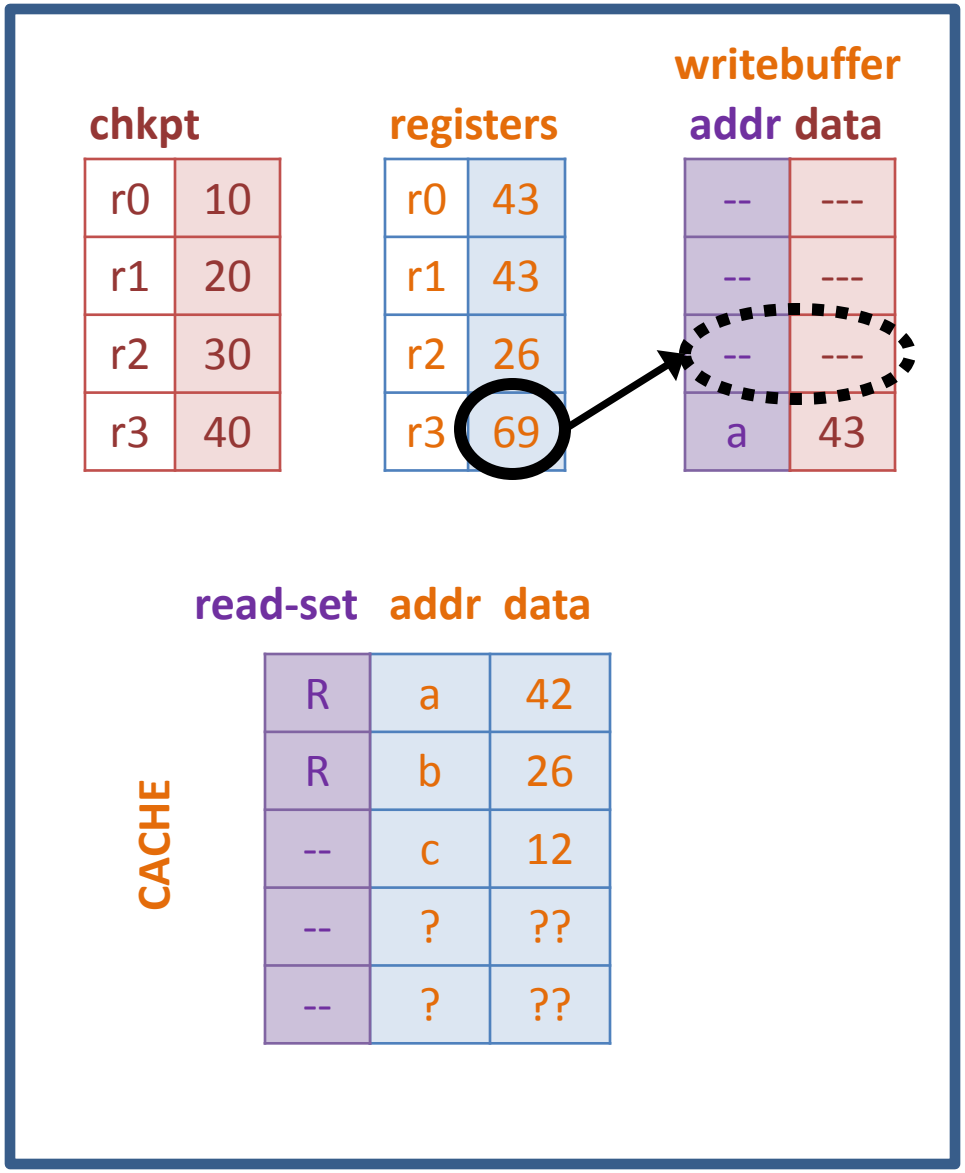
```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```



ΜΩΒ: read/write sets
ΚΟΚΚΙΝΟ: buffer old/new values
ΠΡΑΣΙΝΟ: conflict detection

Παράδειγμα

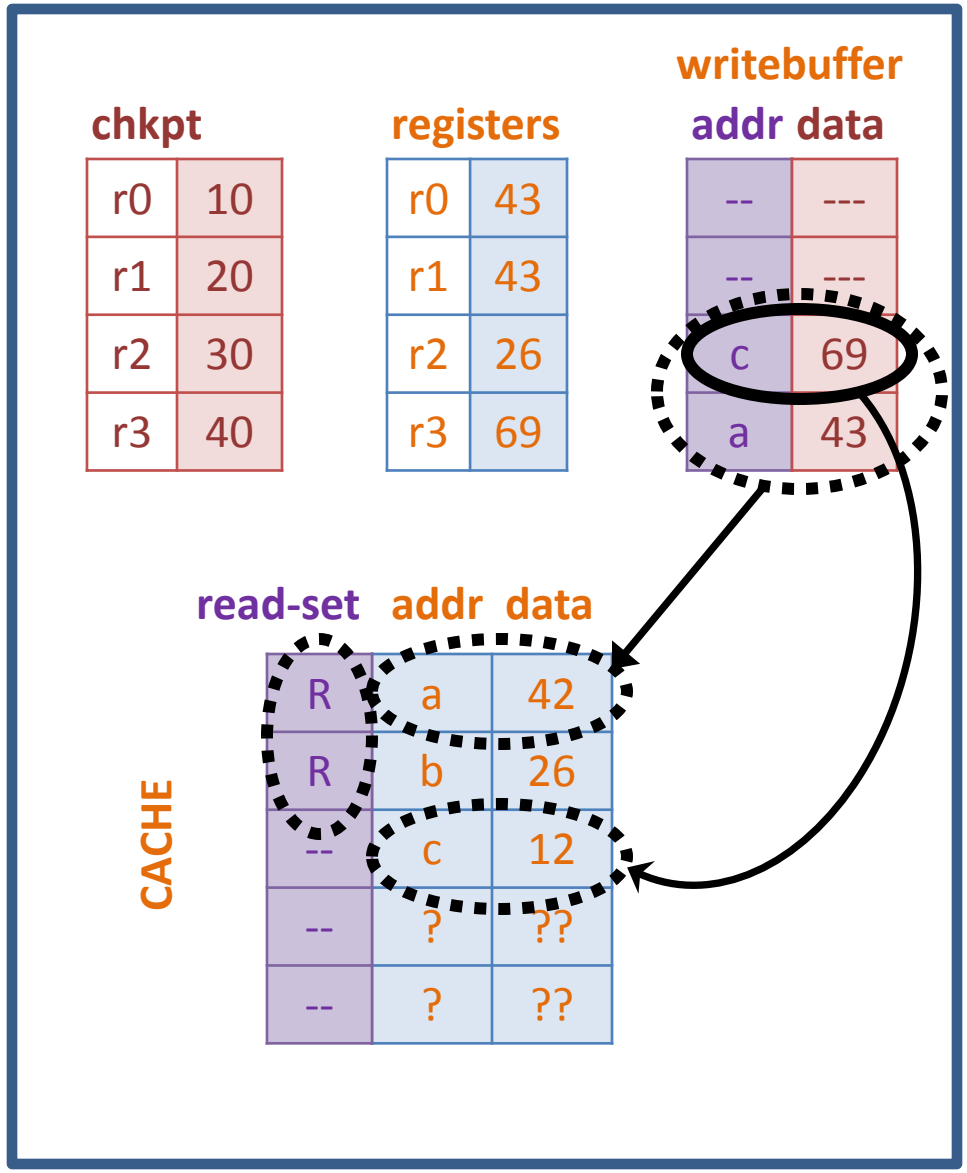
```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```



ΜΩΒ: read/write sets
ΚΟΚΚΙΝΟ: buffer old/new values
ΠΡΑΣΙΝΟ: conflict detection

Παράδειγμα

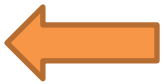
```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```

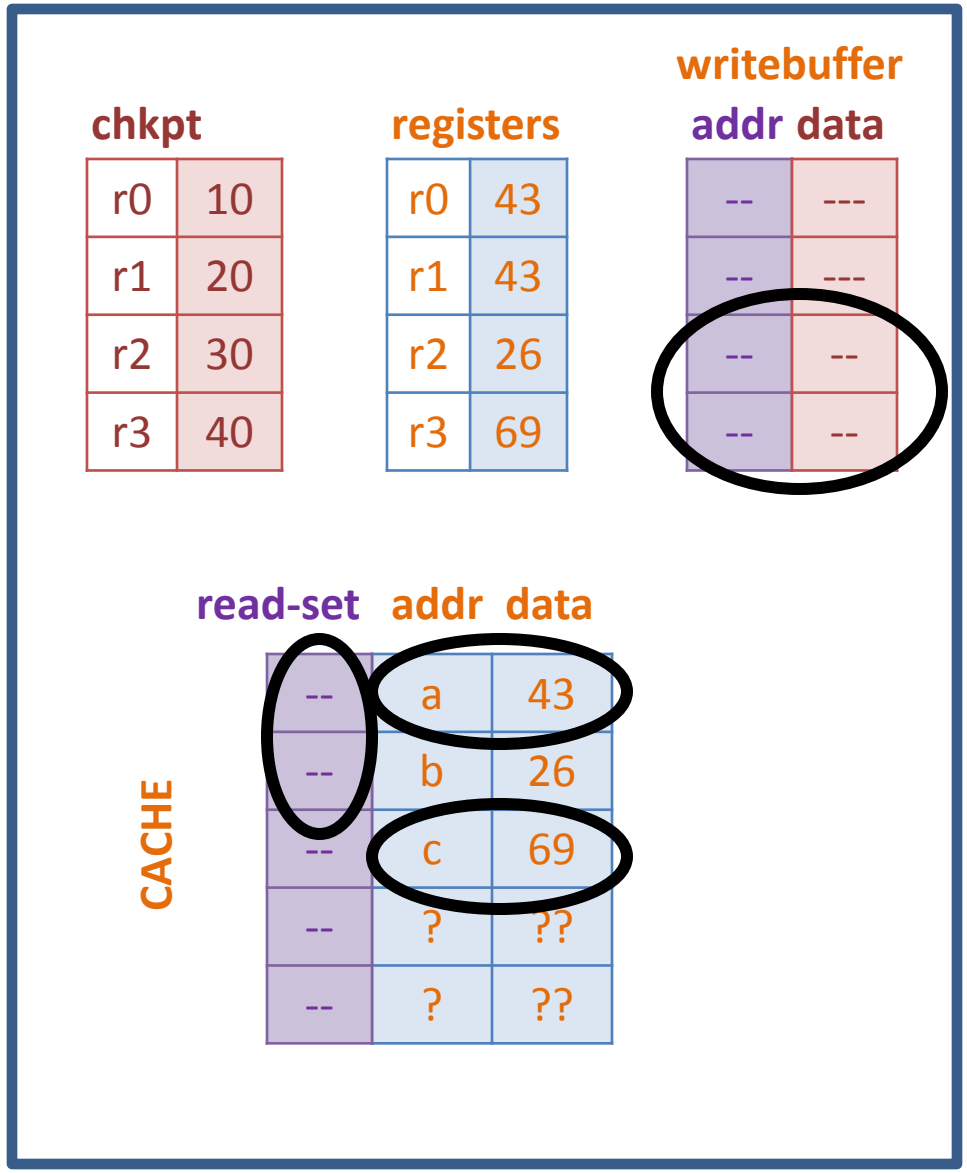


ΜΩΒ: read/write sets
ΚΟΚΚΙΝΟ: buffer old/new values
ΠΡΑΣΙΝΟ: conflict detection

Παράδειγμα

```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2
         c = r3
       commit
```





CORE 0

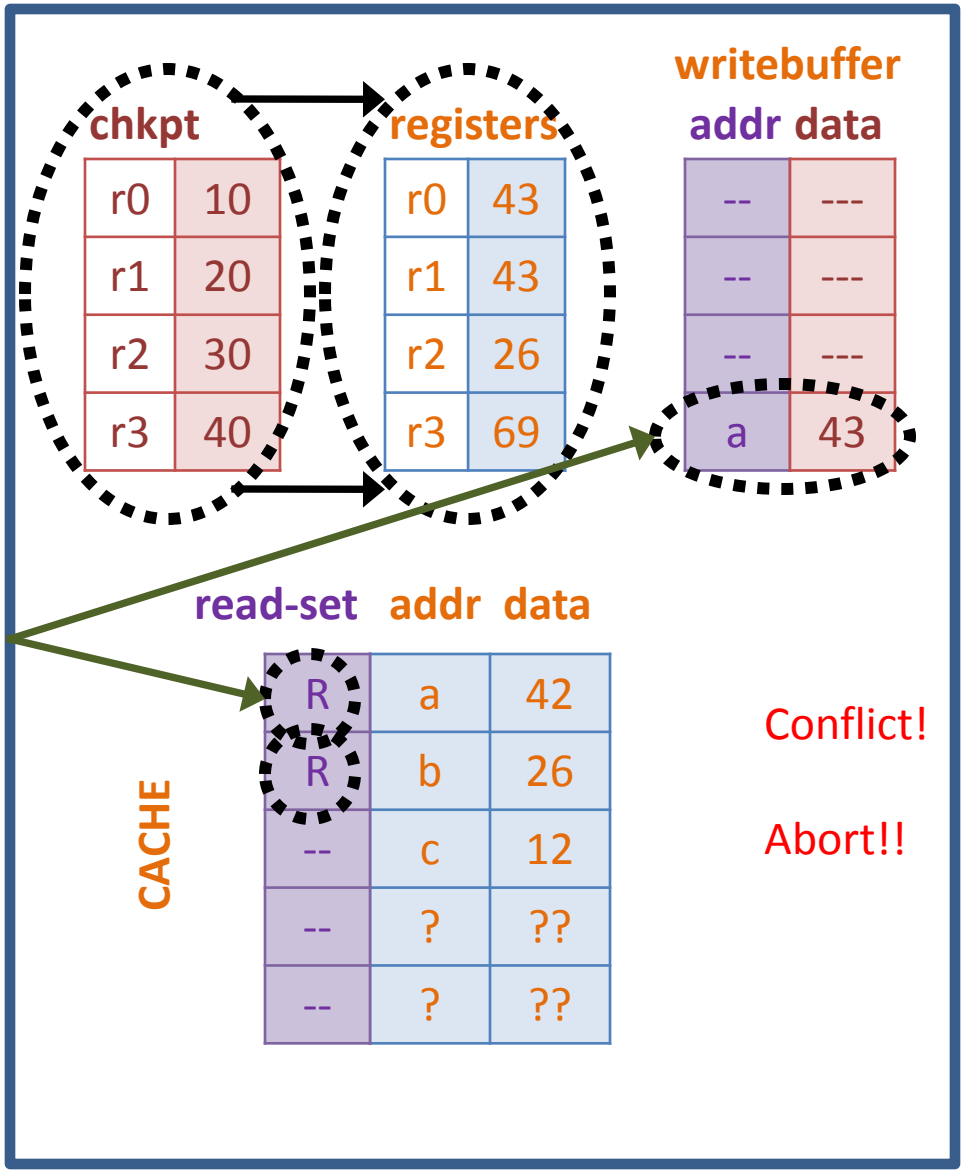
ΜΩΒ: read/write sets
ΚΟΚΚΙΝΟ: buffer old/new values
ΠΡΑΣΙΝΟ: conflict detection

Ανίχνευση διένεξης κατά την αίτηση συνάφειας του πρωτοκόλλου

```
retry: chkpt retry
       r0 = a
       r0 = r0 + 1
       a = r0
       r1 = a
       r2 = b
       r3 = r1 + r2
```

get2write(other-core,a)

```
       c = r3
commit
```



Η εξωτερική αίτηση για εγγραφή ελέγχει τον writebuffer & read-set bits

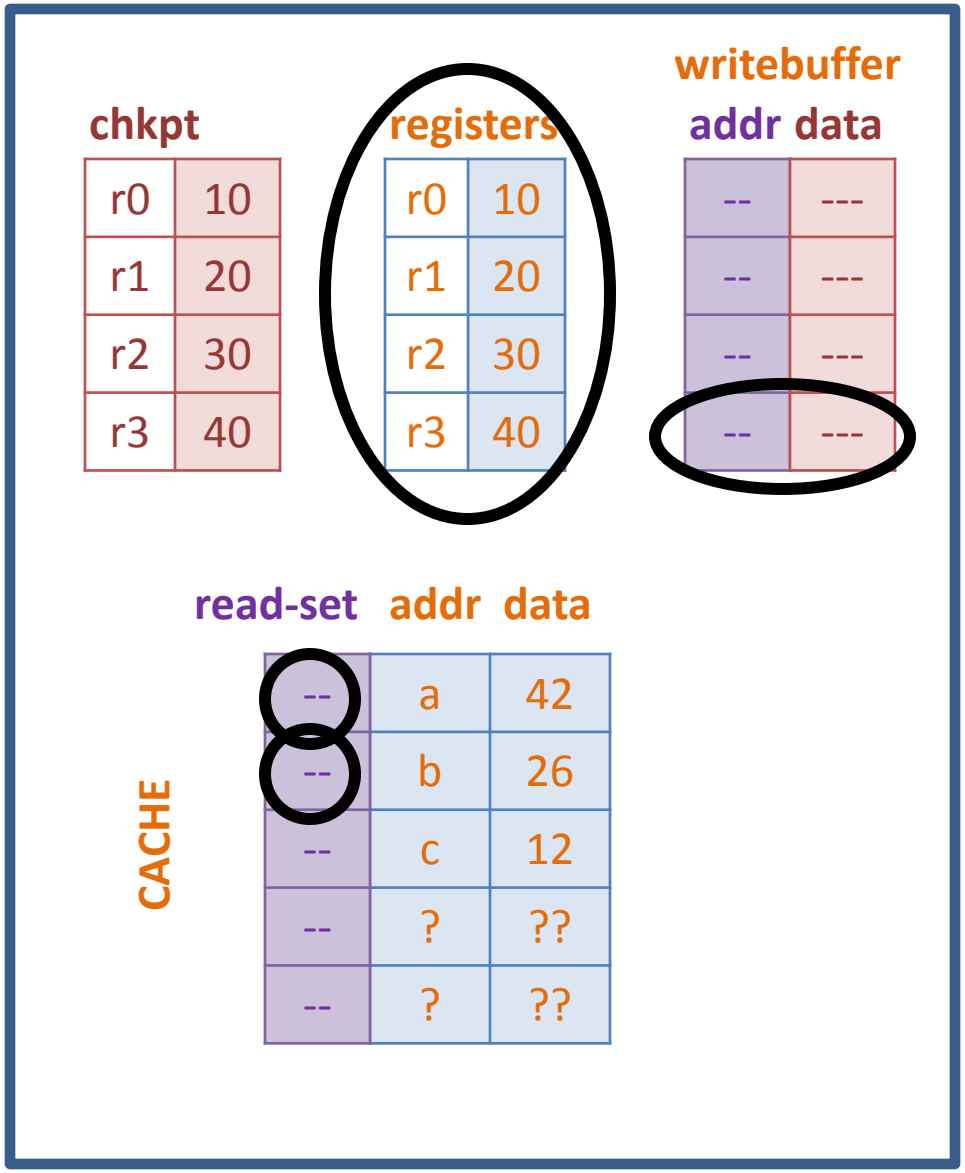
Η εξωτερική αίτηση για ανάγνωση ελέγχει τον writebuffer

Ανίχνευση διένεξης κατά την αίτηση συνάφειας του πρωτοκόλλου

```
retry: chkpt retry
         r0 = a
         r0 = r0 + 1
         a = r0
         r1 = a
         r2 = b
         r3 = r1 + r2

         c = r3
       commit
```

Abort done
Resume at retry



TM support στον Intel Haswell

- Νέες εντολές για «restricted transactional memory» (RTM)
  - `xbegin`: takes pointer to "fallback address" in case of abort
  - `xend`
  - `xabort`
  - πληροφορίες για το abort στον EAX (data conflicts, cache line evictions, interrupts, faults, etc.)

credits for slides:

- C. Kozyrakis (Stanford)
  - ACACES summer school 2008
- D. Wood (Wisconsin)
  - ACACES summer school 2009

Βιβλιογραφία

HTM Systems

- [Hammond04] L. Hammond, B. Carlstrom, V. Wong, M. Chen, C. Kozyrakis and K. Olukotun. "Transactional Coherence and Consistency: Simplifying Parallel Hardware and Software". IEEE Micro, Special Issue on Top Picks from Architecture Conferences, vol. 24, no. 6, November/December 2004.
- **[Herlihy93] M. Herlihy and E. Moss. "Transactional memory: Architectural support for lock-free data structures". International Symposium on Computer Architecture (ISCA '93).**
- [Moir08] M. Moir, K. Moore and D. Nussbaum. "The Adaptive Transactional Memory Test Platform: A Tool for Experimenting with Transactional Code for Rock". 3rd ACM SIGPLAN Workshop on Transactional Computing (TRANSACT '08).
- [Yen07] L. Yen, J. Bobba, M. Marty, K. Moore, H. Volos, M. Hill, M. Swift and D. Wood. "LogTM-SE: Decoupling Hardware Transactional Memory from Caches". 13th International Symposium on High Performance Computer Architecture (HPCA'07).
- [ASF] AMD Corporation. "Advanced Synchronization Facility – Proposed Architectural Specification". Mar 2009. Revision 2.1.
- [Dice08] D. Dice, M. Herlihy, D. Lea, Y. Lev, V. Luchangco, W. Mesard, M. Moir, K. Moore and D. Nussbaum. "Applications of the Adaptive Transactional Memory Test Platform". 3rd ACM SIGPLAN Workshop on Transactional Computing (TRANSACT '08).
- [Dice09] D. Dice , Y. Lev, M. Moir and D. Nussbaum. "Early experience with a commercial hardware transactional memory implementation". 14th international conference on Architectural support for programming languages and operating systems (ASPLOS '09).

STM systems

- [Dice06] D. Dice, O. Shalev and N. Shavit. "Transactional locking II". 20th International Symposium on Distributed Computing (DISC '06). Stockholm, Sweeden, Sept. 2006.
- [Herlihy06] M. Herlihy, V. Luchangco and M. Moir. "A flexible framework for implementing software transactional memory". 21st Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '06). pp. 253–262. ACM Press, 2006.
- [Marathe06] V. Marathe, M. Spear, C. Heriot, A. Acharya, D. Eisenstat, W. Scherer III and M. Scott. "Lowering the overhead of software transactional memory". First ACM SIGPLAN Workshop on Transactional Computing (TRANSACT '06). June 2006.
- [Ni08] Y. Ni, A. Welc, A. Adl-Tabatabai, M. Bach, S. Berkowits, J. Cownie, R. Geva, S. Kozhukow, R. Narayanaswamy, J. Olivier, S. Preis, B. Saha, A. Tal and X. Tian. "Design and Implementation of Transactional Constructs for C/C++". 23rd Annual Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA '08). pp. 195–212, ACM Press, 2008.
- [Riegel06] T. Riegel, P. Felber and C. Fetzer. "A lazy snapshot algorithm with eager validation". LNCS, vol. 4167, pp. 284–298. Springer, Heidelberg (2006).
- [Saha06a] B. Saha, A. Adl-Tabatabai, R. Hudson, C. Minh and B. Hertzberg. "McRT-STM: a high performance software transactional memory system for a multicore runtime". 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '06). pp. 187–197. ACM Press, New York, 2006.
- [Baek07] W. Baek, C. Cao Minh, M. Trautmann, C. Kozyrakis and K. Olukotun. "The OpenTM Transactional Application Programming Interface". 16th International Conference on Parallel Architecture and Compilation Techniques (PACT '07). pages 376–387, Washington, DC, USA, 2007. IEEE Computer Society.

Hybrid Systems

- [Damron06] P. Damron, A. Fedorova, Y. Lev, V. Luchangco, M. Moir and D. Nussbaum. “Hybrid transactional memory”. 12th International conference on Architectural Support for Programming Languages and Operating System (ASPLOS ’06).
- [Diestelhorst08] S. Diestelhorst, M. Hohmuth. “Hardware acceleration for lock-free data structures and software transactional memory”. In the proceedings of the Workshop on Exploiting Parallelism with Transactional Memory and other Hardware Assisted Methods (EPHAM ‘08), April 2008. Boston, MA
- [Minh07] C. Minh, M. Trautmann, J. Chung, A. McDonald, N. Bronson, J. Casper, C. Kozyrakis and K. Olukotun. “An Effective Hybrid Transactional Memory System with Strong Isolation Guarantees”. 34th Annual International Symposium on Computer Architecture (ISCA ’07). San Diego, California, 9-13 June 2007.
- [Saha06b] B. Saha, A.-R. Adl-Tabatabai and Q. Jacobson. “Architectural Support for Software Transactional Memory”. In MICRO 2006.

Applications

- [Kang09] S. Kang and David Bader. "An Efficient Transactional Memory Algorithm for Computing Minimum Spanning Forest of Sparse Graphs". 14th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '09).
- [Nikas09] K. Nikas, N. Anastopoulos, G. Goumas and N. Koziris. "Employing Transactional Memory and Helper Threads to Speedup Dijkstra's Algorithm." International Conference on Parallel Processing (ICPP '09). Vienna, Austria.
- [Scott07] M. Scott, M. Spear, L. Dalessandro and V. Marathe. "Delaunay Triangulation with Transactions and Barriers". 10th International Symposium on Workload Characterization (IISWC '07).
- [Watson07] I. Watson, C. Kirkham and Mikel Lujan. "A Study of a Transactional Parallel Routing Algorithm". 16th International Conference on Parallel Architecture and Compilation Techniques (PACT '07).