ΑΤΟΜΙΚΑ ΑΝΤΙΚΕΙΜΕΝΑ

- Ένα ατομικό αντικείμενο κάποιου τύπου μοιάζει με μια κοινή μεταβλητή αυτού του τύπου.
- Ένα ατομικό αντικείμενο βρίσκεται σε μια κατάσταση και υποστηρίζει ένα σύνολο από λειτουργίες μέσω των οποίων μπορεί να αλλάξει η κατάστασή του.
- Τα ατομικά αντικείμενα έχουν προταθεί σαν δομικά μπλοκ για την κατασκευή πολυ-επεξεργαστικών συστημάτων:
- Εκκίνηση με απλά βασικά ατομικά αντικείμενα που παρέχονται από το υλικό.
- Διαδοχική κατασκευή όλο και πιο πολύπλοκων ατομικών αντικειμένων από πιο απλά αντικείμενα.
- Το σύστημα που προκύπτει είναι απλό, καλά δομημένο, και είναι εύκολο να αποδειχθεί ότι είναι ορθό.

ΒΑΣΙΚΑ ΑΤΟΜΙΚΑ ΑΝΤΙΚΕΙΜΕΝΑ

Καταχωρητής Read/Write

Αποθηκεύει μια τιμή από κάποιο σύνολο και υποστηρίζει δύο λειτουργίες:

- read(R): επιστρέφει την τιμή που είναι αποθηκευμένη στον R χωρίς να τον αλλάξει,
- write(R,v): γράφει την τιμή v στον R και επιστρέφει ack

Καταχωρητής Test-And-Set

Αποθηκεύει μια τιμή από το σύνολο {0,1} και υποστηρίζει δύο λειτουργίες:

- read(R): επιστρέφει την τιμή που είναι αποθηκευμένη στον R χωρίς να τον αλλάξει,
- Test-And-Set(R): αποθηκεύει την τιμή 1 στον R και επιστρέφει την παλιά τιμή του

ΒΑΣΙΚΑ ΑΤΟΜΙΚΑ ΑΝΤΙΚΕΙΜΕΝΑ

Καταχωρητής Load-Link, Store-Conditional (LL/SC)

Αποθηκεύει μια τιμή από κάποιο σύνολο και υποστηρίζει δύο λειτουργίες LL και SC, όπου κάθε LL συσχετίζεται μια κάποια SC.

LL(R): επιστρέφει την τιμή του R,

SC(R,v): αποθηκεύει v στον R αν δεν έχει γίνει καμία αλλαγή στον R από την χρονική στιγμή που εκτελέστηκε η LL η οποία έχει συσχετισθεί με την SC.

Καταχωρητής Compare-And-Swap

Αποθηκεύει μια τιμή από κάποιο σύνολο και υποστηρίζει δύο λειτουργίες:

read(R): επιστρέφει την τιμή του R,

CAS(R,u,v): Αν η κατάσταση του R είναι u αλλάζει σε v και επιστρέφεται TRUE. Διαφορετικά δεν αλλάζει ο R και επιστρέφεται FALSE.

ΒΑΣΙΚΑ ΑΤΟΜΙΚΑ ΑΝΤΙΚΕΙΜΕΝΑ

Καταχωρητής Πολλαπλής-Εγγραφής (MW)

Όλες οι διεργασίες επιτρέπεται να γράφουν στον καταχωρητή

Καταχωρητής Απλής-Εγγραφής (SW)

Μόνο μια από τις διεργασίες μπορεί να γράφει στον καταχωρητή.

Χωρητικότητα καταχωρητή

Ένας καταχωρητής είναι πεπερασμένης χωρητικότητας όταν το σύνολο τιμών που μπορεί να αποθηκεύει είναι πεπερασμένο. Στην αντίθετη περίπτωση είναι μη-πεπερασμένης χωρητικότητας.

- Το υλικό πολλών συστημάτων υποστηρίζει κάποια από τα παραπάνω αντικείμενα. Το υλικό τότε εγγυάται ότι κάθε μια λειτουργία εκτελείται ατομικά.

ΕΠΑΛΗΘΕΥΣΗ ΚΑΘΟΛΙΚΩΝ ΚΑΤΗΓΟΡΗΜΑΤΩΝ (GLOBAL PREDICATE EVALUATION)

Σε πολλά προβλήματα των κατανεμημένων συστημάτων μια ενέργεια πρέπει να συμβεί αν ένα κατηγόρημα (predicate) που αφορά την καθολική κατάσταση του συστήματος είναι αληθές.

Παραδείγματα

- Ανίχνευση αδιεξόδων (deadlock detection).
- Ανίχνευση τερματισμού (termination detection).
- Ανίχνευση απώλειας διακριτικού (token loss detection).
- Διαχείριση μη-προσβάσιμης ή αχρησιμοποίητης μνήμης (garbage collection).
- Εισαγωγή σημείων ελέγχου και επανεκκίνηση (checkpointing & restarting).
- Παρακολούθηση και αποσφαλμάτωση (monitoring & debugging).

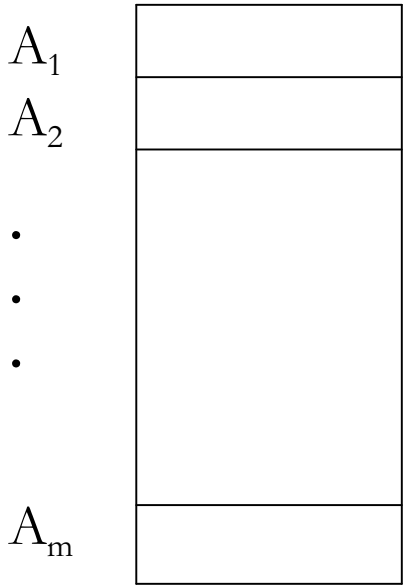
ΟΡΘΟΤΗΤΑ & ΑΠΟΣΦΑΛΜΑΤΩΣΗ

- Η μεγαλύτερη δυσκολία στο να αποδειχθεί η ορθότητα λειτουργίας ενός κατανεμημένου αλγορίθμου είναι η αναγκαιότητα στήριξης της επιχειρηματολογίας σε μη συνεπείς όψεις των διαμοιραζόμενων μεταβλητών.

- Ο υπολογισμός συνεπών όψεων διευκολύνει το έργο επαλήθευσης της ορθότητας των κατανεμημένων αλγορίθμων αλλά δεν είναι εύκολο πρόβλημα.

ΑΤΟΜΙΚΑ ΣΤΙΓΜΙΟΤΥΠΑ ΜΝΗΜΗΣ

- Διαμοιραζόμενο αντικείμενο που αποτελείται από έναν πίνακα με m τμήματα που το καθένα αποθηκεύει μια τιμή.
- Υποστηρίζει δύο είδη ατομικών λειτουργιών:
  - ✓ UPDATE(i,v): εγγράφει στο τμήμα A_i του στιγμιοτύπου την τιμή υ.
  - ✓ SCAN: επιστρέφει ένα διάνυσμα τιμών, μία για κάθε τμήμα και είναι εγγυημένο πως οι τιμές αυτές είναι συνεπείς.



Ατομικό Στιγμιότυπο A

Τα ατομικά στιγμιότυπα αποσκοπούν στην παροχή συνεπών όψεων.

ΑΤΟΜΙΚΑ ΣΤΙΓΜΙΟΤΥΠΑ

Ατομικό Στιγμιότυπο Πολλαπλής-Εγγραφής (MW Atomic Snapshot)

Κάθε διεργασία μπορεί να εκτελεί UPDATES σε κάθε τμήμα.

Στιγμιότυπο Απλής-Εγγραφής (SW Atomic Snapshot)

Μόνο η διεργασία i μπορεί να εκτελεί UPDATES στο τμήμα A_i.

Τα ατομικά στιγμιότυπα είναι πιο ισχυρά αντικείμενα από τους καταχωρητές και διευκολύνουν τη σχεδίαση κατανεμημένων αλγορίθμων!

☹ Τα ατομικά στιγμιότυπα δεν παρέχονται από το υλικό.

ΕΦΑΡΜΟΓΕΣ

- Αμοιβαίος αποκλεισμός [Katseff - STOC'78, Lamport – JACM'86, Dolev & Gafni & Shavit – STOC'88].
- Επίτευξη ομοφωνίας [Aspnes - J. of Alg.'93, Aspnes & Herlihy – J. of Alg '90] & Approximate Agreement [Attiya, Lynch & Shavit – JACM'94]
- Κατασκευή συστημάτων παράλληλης ανάθεσης χρονοσφραγίδων [Dolev & Shavit - STOC'88].
- Αξιόπιστη και αποδοτική υλοποίηση κατανεμημένων δομών δεδομένων [Aspnes & Herlihy - SPAA'90, Herlihy – PODC'91].
- Υλοποίηση χρήσιμων κατανεμημένων αντικειμένων [Vitanyi & Awerbuch - FOCS'86, Bloom – PODC'87, Peterson & Burns – FOCS'87].

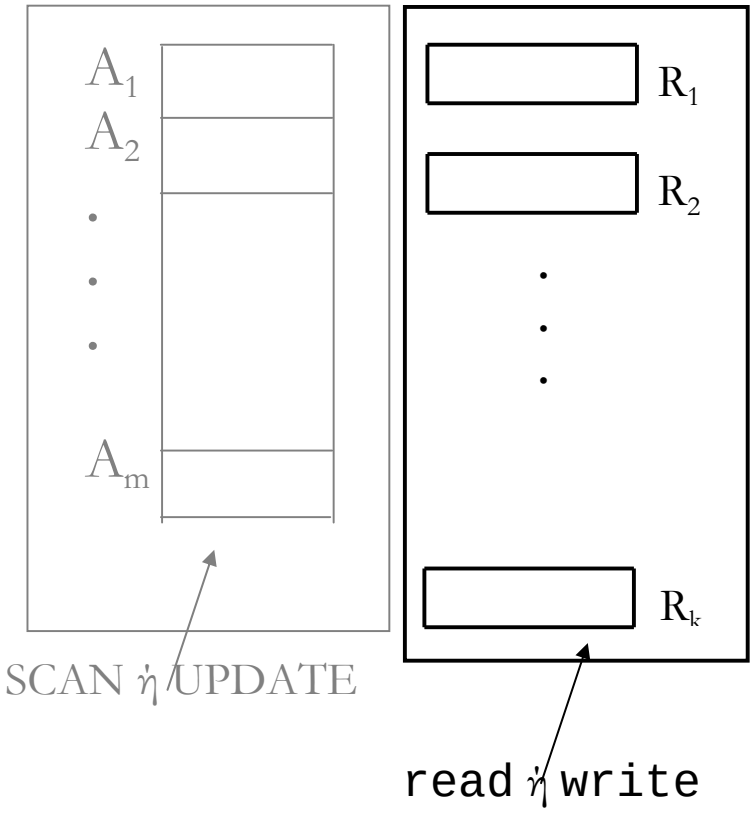
ΥΛΟΠΟΙΩΝΤΑΣ ΑΤΟΜΙΚΑ ΣΤΙΓΜΙΟΤΥΠΑ ΜΕ ΧΡΗΣΗ ΚΑΤΑΧΩΡΗΤΩΝ

- Παροχή αλγορίθμων για τις λειτουργίες SCAN και UPDATE.

- Αποτελεσματικότητα Υλοποίησης

  Χρονική πολυπλοκότητα μιας SCAN ή μιας UPDATE λειτουργίας: ο μέγιστος αριθμός βημάτων που εκτελούνται από οποιαδήποτε διεργασία σε οποιαδήποτε εκτέλεση προκειμένου να υλοποιηθεί η λειτουργία

  Χωρική Πολυπλοκότητα υλοποίησης: # καταχωρητών που χρησιμοποιούνται



Χρήσιμοι Ορισμοί

- Διαφορετικά στιγμιότυπα των λειτουργιών του αντικειμένου μπορούν να εκτελούνται από διαφορετικές διεργασίες ταυτόχρονα.
- Σε κάθε εκτέλεση, οι κλήσεις και οι αποκρίσεις των λειτουργιών του ατομικού αντικειμένου μπορεί να πραγματοποιούνται με οποιαδήποτε σειρά. Πολλές κλήσεις μπορούν π.χ. να συμβούν πριν να συμβεί οποιαδήποτε απόκριση.
- Το *διάστημα εκτέλεσης* μιας λειτουργίας (SCAN ή UPDATE) είναι το διάστημα από την κλήση της μέχρι την απόκρισή της. Τα διαστήματα εκτέλεσης των λειτουργιών ενός ατομικού στιγμιοτύπου μπορεί να επικαλύπτονται (μερικώς ή ολικώς).
- Μια ακολουθία από κλήσεις και αποκρίσεις είναι έγκυρη, αν κάθε κλήση μπορεί να συσχετισθεί με μια απόκριση.

ΤΕΡΜΑΤΙΣΜΟΣ & ΣΕΙΡΙΟΠΟΙΗΣΙΜΟΤΗΤΑ

Ιδιότητα Ελευθερίας Αναμονής (Wait-Freedom)

Κάθε μη-εσφαλμένη διεργασία εκτελεί έναν πεπερασμένο αριθμό βημάτων προκειμένου να διεκπεραιώσει τη λειτουργία της.

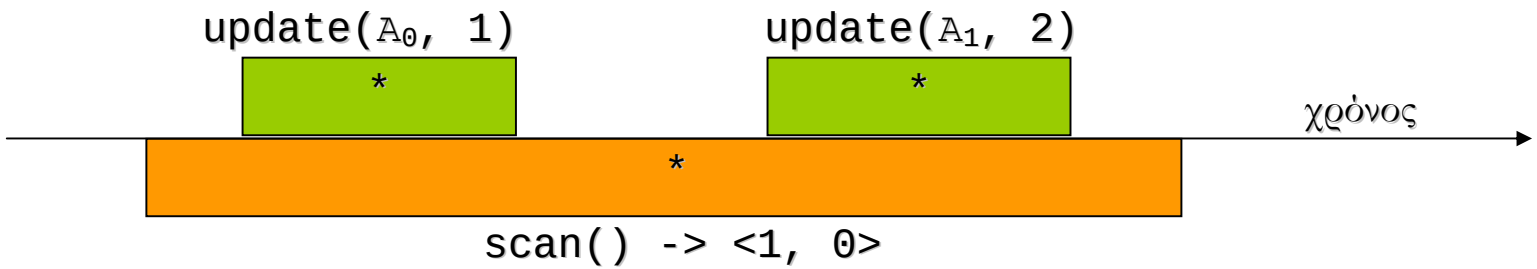
Σειριοποιησιμότητα (Linearizability)

Ακόμα και αν εκτελούνται παράλληλα λειτουργίες σε ένα διαμοιραζόμενο αντικείμενο, υπάρχει μία σειριακή εκτέλεση των λειτουργιών αυτών που έχει τα ίδια αποτελέσματα.

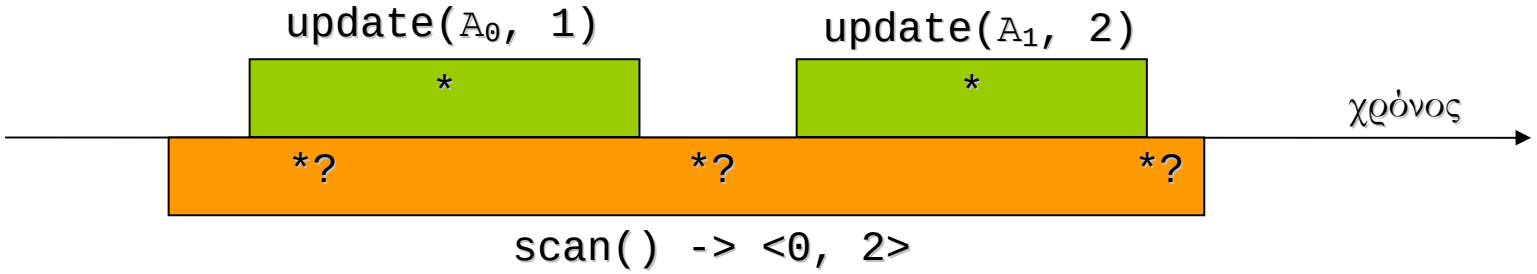
- Σε κάθε εκτέλεση, κάθε SCAN και UPDATE λειτουργία που εμπεριέχεται στην εκτέλεση επιστρέφει το ίδιο αποτέλεσμα σαν να εκτελέστηκε σειριακά σε κάποιο σημείο στο χρονικό διάστημα της εκτέλεσης της. Το σημείο αυτό λέγεται σημείο σειριοποίησης της λειτουργίας (linearization point).

ΣΕΙΡΙΟΠΟΙΗΣΙΜΟΤΗΤΑ

Παράδειγμα σειριοποιήσιμης εκτέλεσης



Παράδειγμα μη-σειριοποιήσιμης εκτέλεσης



Σειριοποιησιμότητα - Φορμαλιστικά

Έστω μια εκτέλεση β. Λέμε ότι η β είναι σειριοποιήσιμη αν τα ακόλουθα είναι δυνατά:

- Για κάθε προσομοιούμενη λειτουργία π που έχει περατωθεί στη β, μπορούμε να επιλέξουμε ένα σημείο σειριοποίησης *π κάπου μεταξύ της κλήσης της π και της απόκρισής της.
- Μπορούμε να επιλέξουμε ένα υποσύνολο Φ των λειτουργιών των οποίων η εκτέλεση δεν έχει περατωθεί στη β τ.ω. για κάθε λειτουργία π στο Φ:
- μπορούμε να εισάγουμε μια απόκριση, και
- μπορούμε να επιλέξουμε ένα σημείο σειριοποίησης *π, το οποίο να βρίσκεται σε κάποιο σημείο της εκτέλεσης που έπεται της κλήσης της π.

Διαισθητικά:

«Κάθε λειτουργία μοιάζει σαν να εκτελέστηκε στιγμιαία κάποια χρονική στιγμή μέσα στο διάστημα εκτέλεσής της.»

«Αν και η εκτέλεση κάποιων λειτουργιών μπορεί να συμβαίνει ταυτόχρονα, η υλοποίηση πρέπει να εγγυάται ότι οι λειτουργίες λαμβάνουν χώρα σειριακά, μια κάθε φορά, και η σειριακή αυτή διάταξη θα πρέπει να σέβεται τις κλήσεις και αποκρίσεις των λειτουργιών».

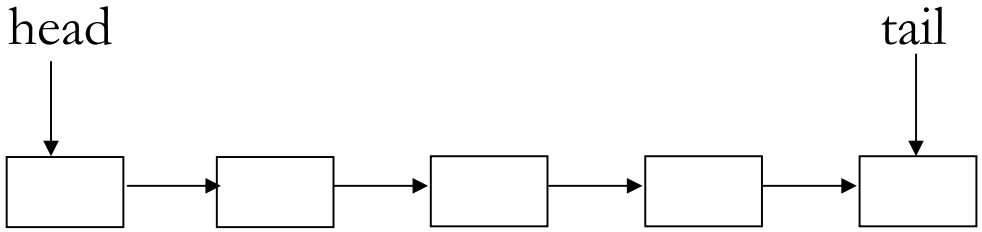
Σειριοποιησιμότητα – Περισσότερα Παραδείγματα

Μια ουρά FIFO Q υποστηρίζει δύο λειτουργίες:

enq(Q,v): προσθέτει το στοιχείο v στο τέλος της ουράς Q

deq(Q): αφαιρεί και επιστρέφει το πρώτο στοιχείο της ουράς Q

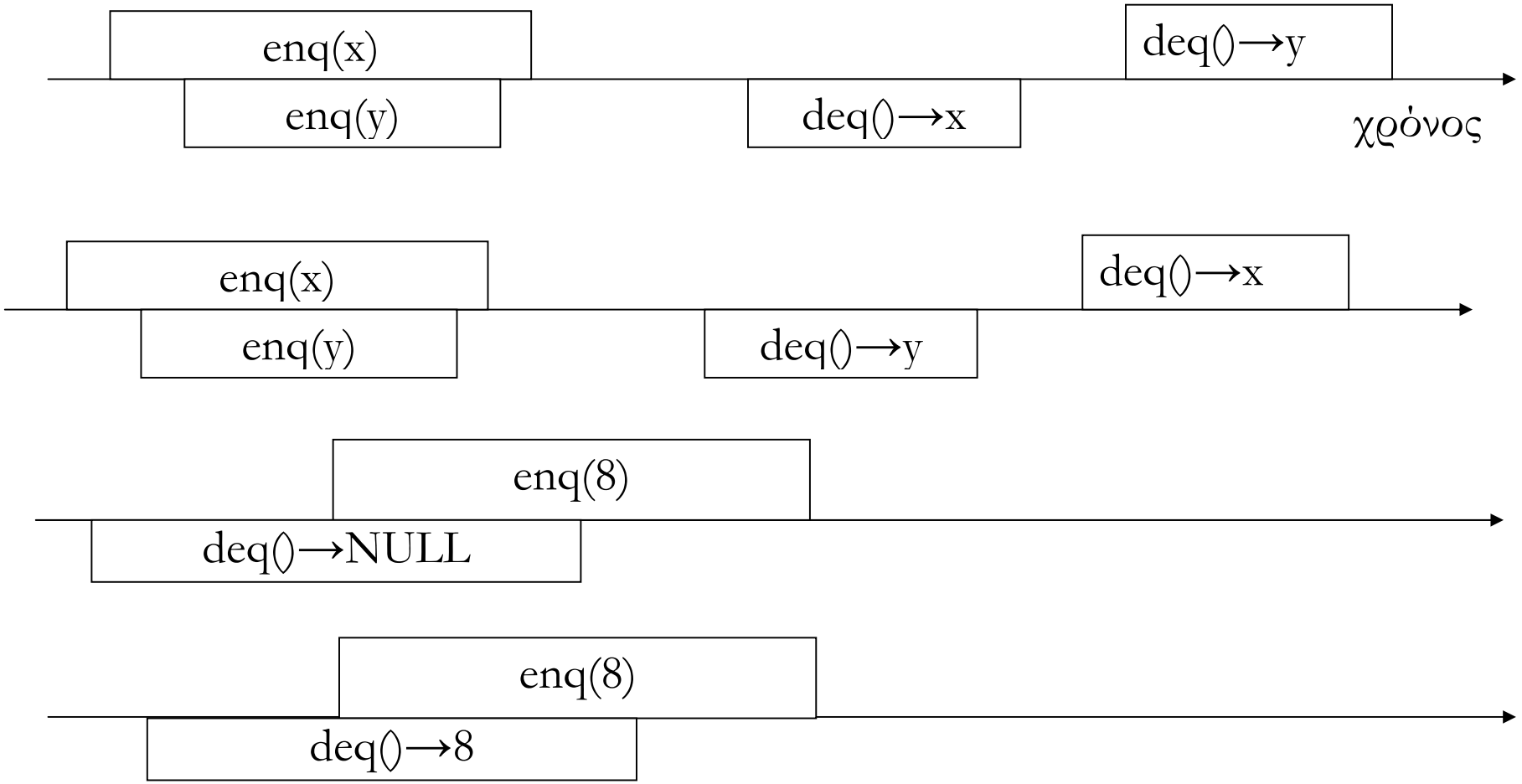
Σε μια παράλληλη ουρά πολλές διεργασίες μπορεί να προσπαθούν να εισάγουν (enq) και να εξάγουν(deq) στοιχεία ταυτόχρονα.



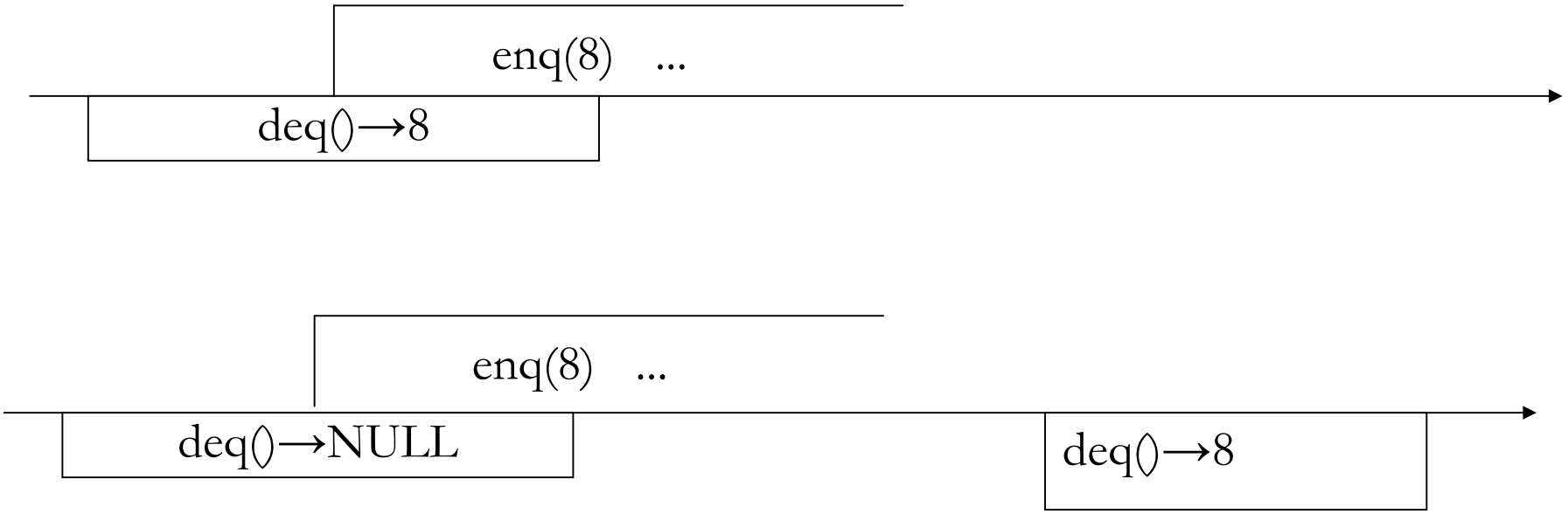
Ουρές μπορούν να υλοποιηθούν από απλούστερα αντικείμενα (π.χ., Test-And-Set και LL/SC registers).

Το ίδιο ισχύει και για άλλα αντικείμενα, π.χ., στοίβες, λίστες, λίστες παράλειψης, γράφους, κλπ.

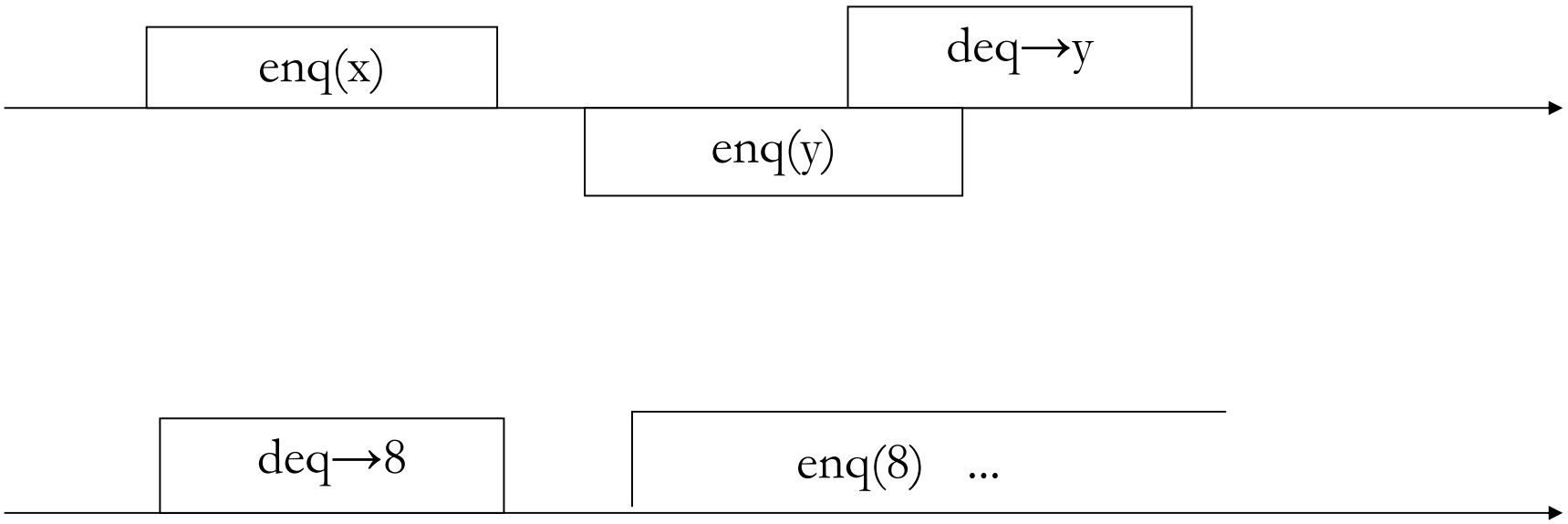
Σειριοποιήσιμες Εκτελέσεις



Σειριοποιήσιμες Εκτελέσεις



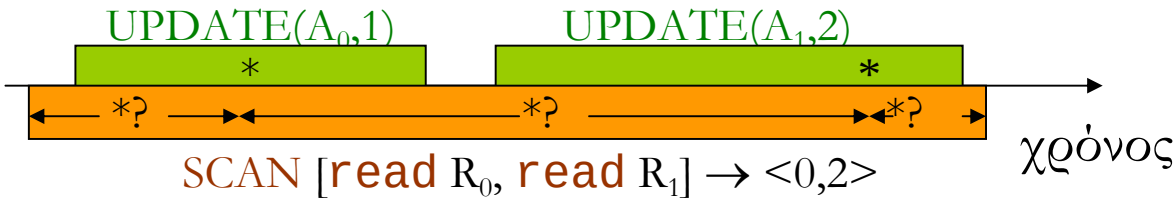
Μη-Σειριοποιήσιμες Εκτελέσεις



Ατομικά Στιγμιότυπα – Προφανής Λύση

Είναι δύσκολο για μια διεργασία να καταφέρει να υπολογίσει μια συνεπή όψη όλων των καταχωρητών.

Λειτουργίες Διεργασιών		Τιμές Καταχ/τών	
P	**Q**	**R_0**	**R_1**
		0	0
read $R_0 \rightarrow 0$			
	write(R_0,1)		
		1	0
	write(R_1,2)		
		1	2
read $R_1 \rightarrow 2$			



☹ Σε καμία χρονική στιγμή δεν συνυπάρχουν οι τιμές 0 και 2 στους καταχωρητές.

☞ Τα ατομικά στιγμιότυπα έχουν σειριοποιήσιμες, ελεύθερες-αναμονής υλοποιήσεις χρησιμοποιώντας καταχωρητές. *[Afek et. al–JACM'93, Anderson–DistComp'93, Aspnes&Herlihy–SPAA'90]*

ΑΠΛΗ ΥΛ/ΣΗ SW Snapshot ΜΕ ΧΡΗΣΗ n SW ΚΑΤΑΧΩΡΗΤΩΝ

Υπάρχουν n τμήματα, A_1, ..., A_n, ένα για κάθε διεργασία. Στο τμήμα A_i μπορεί να εκτελεί UPDATES μόνο η p_i.

Κάθε καταχωρητής R_i μπορεί να εγγραφεί μόνο από την p_i αλλά να διαβαστεί από όλες τις διεργασίες.

Το τμήμα A_i έχει συσχετισθεί με τον καταχωρητή R_i. UPDATES στο A_i γράφουν μόνο στον R_i.

Κάθε καταχωρητής R_i αποτελείται από τα πεδία:

- val_i: τιμή του τμήματος με το οποίο έχει συσχετισθεί
- tag_i: χρονοσφραγίδα την οποία χρησιμοποιεί η p_i για να ξεχωρίζει τις UPDATES της
- $view_i$: διάνυσμα n τιμών, μια για κάθε τμήμα.

Ο αλγόριθμος UnboundedSnapshot

UPDATE στο τμήμα A_i με τιμή v:

view := SCAN;

increment p_i's tag;

`write`(R_i,<v, tag,view>);

SCAN από τη διεργασία p:

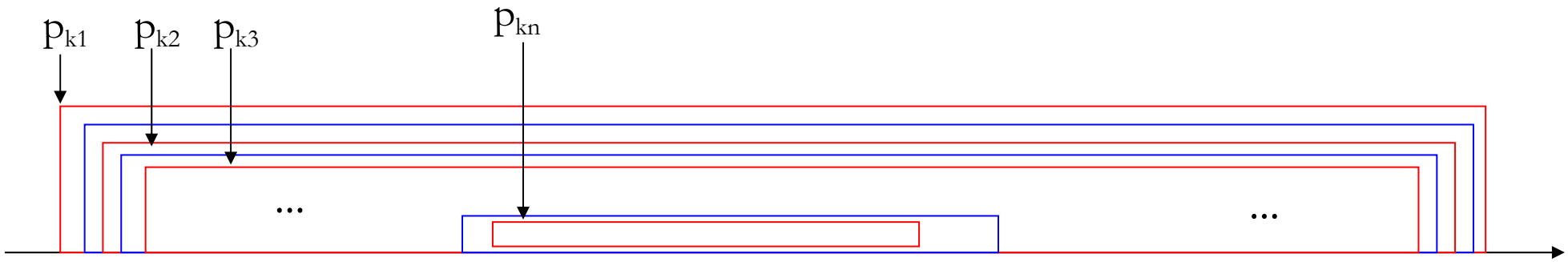
repeatedly `read` $R_1, \ldots, R_n$ until see

1. either the same vector twice (then, return the vector), or
2. two new UPDATES by some process (return the view written 2nd time);

☞ Χρονική Πολυπλοκότητα = $O(n^2)$ για τη SCAN και την UPDATE.

☞ Απαιτούνται n μη-πεπερασμένης χωρητικότητας MW καταχωρητές.

Ο αλγόριθμος UnboundedSnapshot



SCANS ή embedded SCANS

UPDATES

Όλες οι p_{k1}, p_{k2}, ..., p_{kn} είναι ενεργές ταυτόχρονα. Έχουμε n διεργασίες στο σύστημα. Άρα, η embedded SCAN που εκτελείται από την p_{kn} πρέπει να τερματίζει βλέποντας το ίδιο διάνυσμα τιμών σε δύο συνεχόμενες ομάδες αναγνώσεων.

Ο αλγόριθμος UnboundedSnapshot

Θεώρημα: Ο παραπάνω αλγόριθμος είναι μια σωστή υλοποίηση ενός ατομικού στιγμιοτύπου μνήμης και επιτυγχάνει τερματισμό ελεύθερο-αναμονής.

Ορισμός: Μια SCAN επιστρέφει ένα **συνεπές διάνυσμα** τιμών αν, για κάθε τμήμα του στιγμιοτύπου, η τιμή που επιστρέφει η SCAN για το τμήμα είναι η τιμή που χρησιμοποίησε (ως παράμετρο), η τελευταία UPDATE στο τμήμα που σειριοποιείται πριν τη SCAN. Αν δεν υπάρχει τέτοια UPDATE η τιμή που επιστρέφει η SCAN για το τμήμα είναι η αρχική τιμή του τμήματος.

Για να αποδείξουμε το θεώρημα θα πρέπει:

Να αποδώσουμε σημεία σειριοποίησης.

Να δείξουμε ότι για κάθε λειτουργία (SCAN ή UPDATE), το σημείο σειριοποίησης της βρίσκεται εντός του διαστήματος εκτέλεσής της.

Να δείξουμε ότι οι SCANS επιστρέφουν συνεπή διανύσματα.

Ο αλγόριθμος UnboundedSnapshot

Απόδοση Σημείων Σειριοποίησης

Κάθε UPDATE σειριοποιείται στο σημείο που εκτελεί τη write εντολή της.

Για ευκολία αποδίδουμε σημεία σειριοποίησης όχι μόνο στις SCANS αλλά και στις embedded SCANS (δηλαδή σε αυτές που καλούνται από τις UPDATES). Στη συνέχεια θα τις ονομάζουμε όλες SCAN.

Χωρίζουμε τις SCAN σε δύο κατηγορίες:

Εύκολες: Αυτές που τερματίζουν λόγω της συνθήκης 1 στον ψευδο-κώδικα.

Δύσκολες: Αυτές που τερματίζουν λόγω της συνθήκης 2.

Σειριοποιούμε κάθε εύκολη SCAN ανάμεσα στις δύο ομάδες αναγνώσεων (collect) που επιστρέφουν το ίδιο διάνυσμα.

Σειριοποιούμε τις δύσκολες SCAN με επαγωγή ως προς τα σημεία απόκρισης τους.

Ο αλγόριθμος UnboundedSnapshot

Βάση Επαγωγής: Απόδοση σημείο σειριοποίησης στη δύσκολη SCAN (έστω S) της οποίας το σημείο απόκρισης είναι 1ο.

Η S επιστρέφει ένα διάνυσμα τιμών το οποίο έχει εγγραφεί από μια UPDATE U (δηλαδή το διάνυσμα αυτό υπολογίστηκε από την embedded SCAN S' της UPDATE):

Η S' είναι εύκολη SCAN και έτσι της έχει ήδη αποδοθεί σημείο σειριοποίησης.

Η U (και άρα και η S') εμπεριέχονται στην S.

Το σημείο σειριοποίησης της S τοποθετείται στο ίδιο σημείο με εκείνο της S'.

Επαγωγική Υπόθεση: Έστω S η δύσκολη SCAN της οποίας το σημείο απόκρισης είναι το k-οστό. Ας υποθέσουμε ότι έχουμε αποδώσει σημεία σειριοποίησης σε όλες τις δύσκολες SCAN των οποίων το σημείο απόκρισης προηγείται εκείνου της S.

Ο αλγόριθμος UnboundedSnapshot

Επαγωγικό Βήμα: Αποδίδουμε σημείο σειριοποίησης στην S.

Η S επιστρέφει ένα διάνυσμα τιμών το οποίο έχει εγγραφεί από μια UPDATE U (δηλαδή το διάνυσμα αυτό υπολογίστηκε από την embedded SCAN S' της UPDATE):

Η U (και άρα και η S') εμπεριέχονται στην S.

Αν η S' είναι εύκολη SCAN, της έχει ήδη αποδοθεί σημείο σειριοποίησης.

Αν η S' είναι δύσκολη SCAN, έχει τερματίσει πριν την S $\Rightarrow$ της έχει αποδοθεί σήμειο σειριοποίησης (από επαγωγική υπόθεση).

Το σημείο σειριοποίησης της S τοποθετείται στο ίδιο σημείο με εκείνο της S'.

Λήμμα: Τα σημεία σειριοποίησης κάθε SCAN ή UPDATE βρίσκεται μέσα στο διάστημα εκτέλεσης της.

Απόδειξη: Για τις UPDATES και τις εύκολες SCAN αυτό είναι προφανές.

Για τις δύσκολες SCANS αποδεικνύεται επαγωγικά ως προς τα σημεία απόκρισής τους.

Ο αλγόριθμος UnboundedSnapshot

Στη βάση της επαγωγής (θυμηθείτε ότι) η SCAN S της οποίας το σημείο απόκρισης είναι $1^{ο}$ σειριοποιείται στο ίδιο σημείο με την εύκολη embedded SCAN S' της οποίας το διάνυσμα επιστρέφει. Επίσης, η S' εμπεριέχεται στην S.

Το σημείο σειριοποίησης της S' βρίσκεται στο διάστημα εκτέλεσής της $\Rightarrow$ το σημείο σειριοποίησης της S εμπεριέχεται στο διάστημα εκτέλεσής της.

Επαγωγικό Βήμα: Αποδεικνύουμε τον ισχυρισμό για τη δύσκολη SCAN S της οποίας το σημείο σειριοποίησης είναι το k-οστό.

Η S επιστρέφει ένα διάνυσμα τιμών το οποίο έχει εγγραφεί από μια UPDATE U (δηλαδή το διάνυσμα αυτό υπολογίστηκε από την embedded SCAN S' της UPDATE):

Η U (και άρα και η S') εμπεριέχονται στην S.

Αν η S' είναι εύκολη το σημείο σειριοποίησης είναι μέσα στο διάστημα εκτέλεσής της.

Το ίδιο ισχύει αν η S' είναι δύσκολη (επαγωγική υπόθεση: η S' τερματίζει πριν την S).

Το σημείο σειριοποίησης της S τοποθετείται στο ίδιο σημείο με εκείνο της S'.

$\Rightarrow$ το σημείο σειριοποίησης της S βρίσκεται μέσα στο διάστημα εκτέλεσής της.

Ο αλγόριθμος UnboundedSnapshot

Λήμμα: Σε κάθε εκτέλεση του αλγορίθμου, οι SCAN λειτουργίες επιστρέφουν συνεπή διανύσματα τιμών.

Απόδειξη: Ο ισχυρισμός ισχύει προφανώς για τις εύκολες SCAN.

Για δύσκολες SCAN θα αποδειχθεί με επαγωγή ως προς τα σημεία απόκρισης.

Στη βάση της επαγωγής (θυμηθείτε ότι) η SCAN S, της οποίας το σημείο απόκρισης είναι 1^o, σειριοποιείται στο ίδιο σημείο με την εύκολη embedded SCAN S' της οποίας το διάνυσμα επιστρέφει. Αφού η S' επιστρέφει συνεπές διάνυσμα, το ίδιο και η S.

Επαγωγικό Βήμα: Η Αποδεικνύουμε τον ισχυρισμό για τη δύσκολη SCAN S της οποίας το σημείο σειριοποίησης είναι το k-οστό.

Η S επιστρέφει το ίδιο διάνυσμα τιμών που επιστρέφει μια embedded SCAN S' και σειριοποιείται στο ίδιο σημείο.

Αν η S' είναι εύκολη, επιστρέφει προφανώς συνεπές διάνυσμα. Αν η S' είναι δύσκολη επιστρέφει συνεπές διάνυσμα από επαγωγική υπόθεση (η S' τερματίζει πριν την S).

Άρα και η S επιστρέφει συνεπές διάνυσμα.

Αλγόριθμος που χρησιμοποιεί καταχωρητές πεπερασμένου μεγέθους

- Οι καταχωρητές που χρησιμοποιεί ο UnboundedSnapshot είναι μη-πεπερασμένου μεγέθους λόγω του πεδίου tag που αποθηκεύει τιμές από μη-πεπερασμένο σύνολο (σύνολο ακεραίων).

- Τα tags χρησιμοποιούνται από τις διεργασίες που εκτελούν SCAN (και embedded SCAN) για να καθορίσουν αν συνέβησαν νέες UPDATES κατά τη διάρκεια τους.

- Αυτή η πληροφορία μπορεί να παρασχεθεί, και αν αντικατασταθούν τα tags από έναν λιγότερο ισχυρό μηχανισμό που αποτελείται από ένα συνδυασμό bits χειραψίας (handshaking bits) και ένα bit εναλλαγής (toggle bit).

Αλγόριθμος που χρησιμοποιεί καταχωρητές πεπερασμένου μεγέθους

- Υπάρχουν 2n κοινές μεταβλητές, R_1, ..., R_n, R_1', ..., R_n'.
- Κάθε καταχωρητής R_i εγγράφεται από την p_i κάθε φορά που εκτελεί μια UPDATE στο A_i. Κάθε καταχωρητής R_i' εγγράφεται από την p_i κάθε φορά που εκτελεί μια SCAN.
- Κάθε καταχωρητής R_i περιέχει μια τιμή val, ένα διάνυσμα τιμών view, και αντί για tag, ένα διάνυσμα comm των n bits, μέσω των οποίων η p_i μπορεί να ενημερώνει τις υπόλοιπες διεργασίες για τις UPDATES της.
- Κάθε καταχωρητής R_j' αποτελείται από ένα διάνυσμα ack των n bits, όπου το R_j'.ack[i] χρησιμοποιείται από την p_j για να επιβεβαιώσει ότι έχει δει νέες UPDATES από την p_i.
- Για κάθε ζεύγος διεργασιών (p_i, p_j), υπάρχει ένα ζεύγος bits χειραψίας, το (R_i.comm[j], R_j'.ack[i]).

$1^{η}$ Προσπάθεια Σχεδίασης του Αλγορίθμου

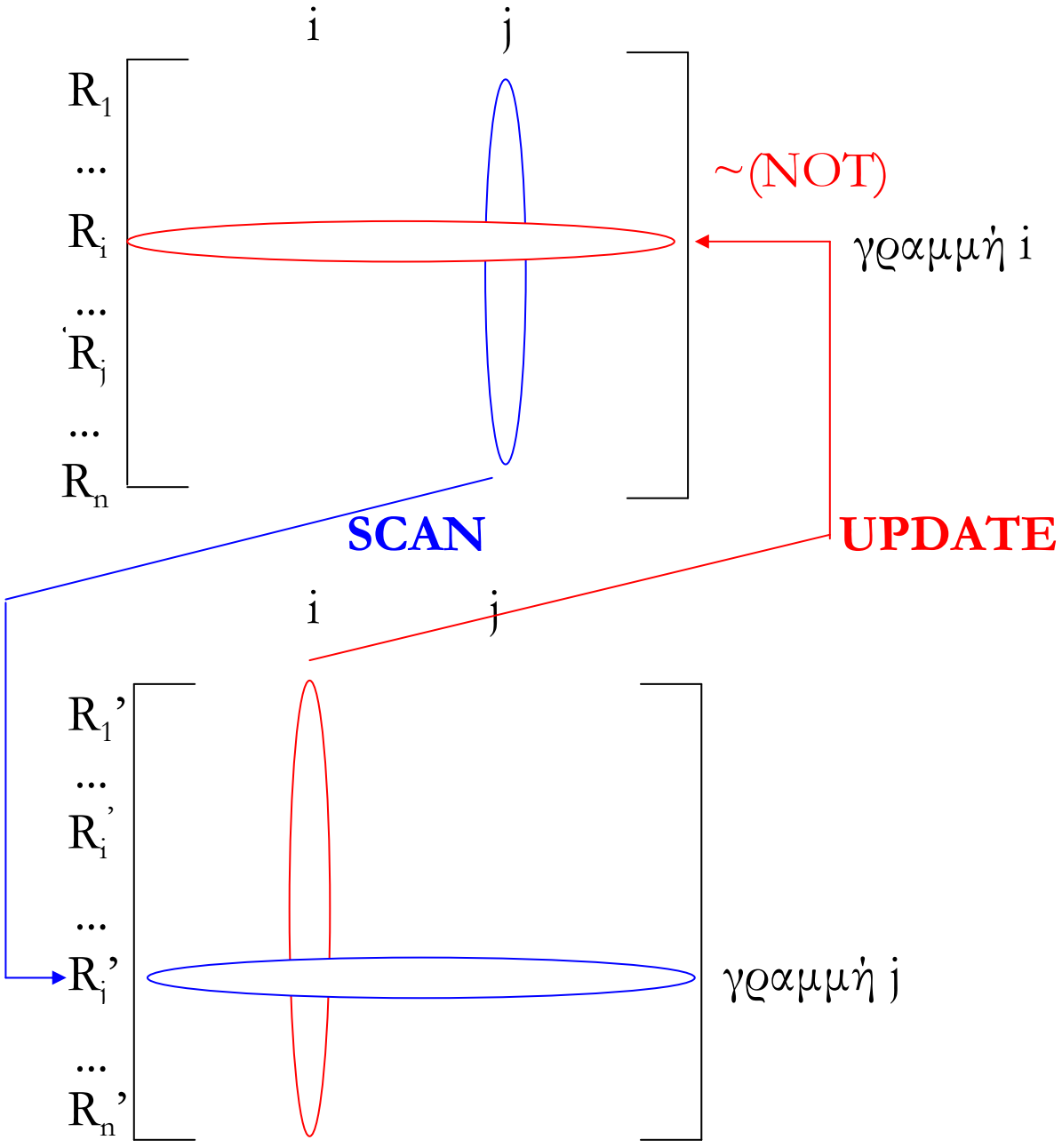
Κατά την εκτέλεση μιας UPDATE(v) από την p_i:

- διάβασε τα n bits R_k'.ack[i], για κάθε k
- υπολόγισε ένα διάνυσμα n bits, όπου για κάθε k, η τιμή του k-οστού στοιχείου του διανύσματος είναι NOT(R_k'.ack[i])
- γράψε το διάνυσμα που υπολογίστηκε στο R_i.comm

Κατά την εκτέλεση μιας SCAN από την p_j:

- επαναληπτικά εκτέλεσε 2 ομάδες αναγνώσεων και εξέτασε αν έχει αλλάξει κάτι από την $1^{η}$ στη $2^{η}$ ομάδα αναγνώσεων:
  - ◊ πριν εκτελέσει τις ομάδες αναγνώσεων, η p_j διαβάζει όλα τα bits χειραψίας R_k.comm[j], για κάθε k, υπολογίζοντας έτσι ένα διάνυσμα n bits το οποίο αντιγράφει ως έχει στον R_j'.
  - ◊ η p_j ελέγχει για αλλαγές στα R_k.comm[j], ∀ k, μεταξύ των δύο ομάδων αναγνώσεων της.
- αν σε 2n+1 προσπάθειες δει τέτοιες αλλαγές, τότε έχει δει 3 UPDATES από την ίδια διεργασία οπότε δανείζεται το διάνυσμα της $3^{ης}$ UPDATE.

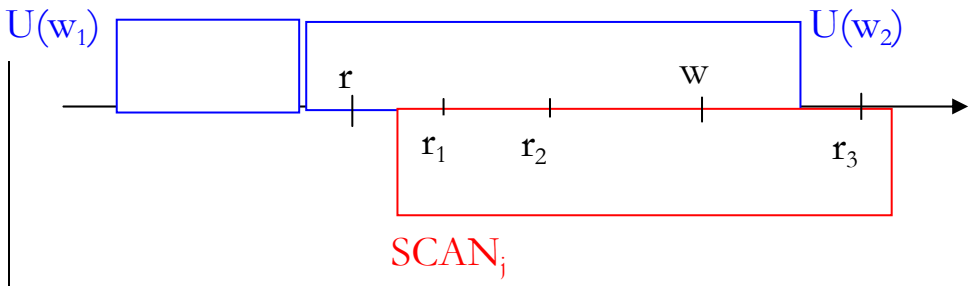
1η Προσπάθεια Σχεδίασης του Αλγορίθμου



Στον παραπάνω αλγόριθμο, κάθε φορά που μια διεργασία που εκτελεί SCAN βλέπει μια αλλαγή, κάποια UPDATE έχει συμβεί!

1η Προσπάθεια Σχεδίασης του Αλγορίθμου

Είναι όμως το αντίθετο σωστό;

UPDATE$(v_1)_i$ η i διαβάζει το R_j'.ack[i] = 1 η i γράφει v_1 και θέτει R_i.comm[j] = 0 acknowledgement$_i$ UPDATE$(v_2)_i$ η i διαβάζει το R_j'.ack[i] = 1 η i γράφει v_2 και θέτει το R_i.comm[j] = 0 (w) acknowledgement$_i$	 SCAN$_j$ η j διαβάζει R_i.comm[j] = 0 (r_1) η j θέτει το R_j'.ack[i] = 0 η j διαβάζει το R_i.comm[j] = 0 (r_2) η j διαβάζει R_i.comm[j]=0 (r_3) και αποφασίζει ότι δεν έχει συμβεί καμία UPDATE από την προηγούμενη ανάγνωση του R_i

Αλγόριθμος που χρησιμοποιεί καταχωρητές πεπερασμένου μεγέθους

Για να επιλύσουμε το πρόβλημα, αποθηκεύουμε επιπλέον ένα bit εναλλαγής σε κάθε καταχωρητή R_i, το οποίο αλλάζει τιμή κάθε φορά που εγγράφεται ο R_i.

Ο Αλγόριθμος BoundedSnapshot

Κάθε καταχωρητής R_i περιέχει τα εξής:

- val: η τιμή του τμήματος A_i
- comm: ένα διάνυσμα n bits
- toggle: ένα bit εναλλαγής
- view: ένα διάνυσμα τιμών, μια για κάθε τμήμα

Κάθε καταχωρητής R_j' περιέχει ένα διάνυσμα ack των n bits.

Αλγόριθμος που χρησιμοποιεί καταχωρητές πεπερασμένου μεγέθους

SCAN από τη διεργασία j:

- η j διαβάζει όλα τα bits χειραψίας R_i.comm[j], για κάθε i, υπολογίζοντας έτσι ένα διάνυσμα n bits το οποίο αντιγράφει ως έχει στον R_j'.
- η j εκτελεί δύο ομάδες αναγνώσεων των R_1, .., R_n
- αν για κάθε i, τα R_i.comm[j] και R_i. toggle είναι ίδια στις δύο ομάδες αναγνώσεων, και η κοινή τιμή R_i.comm[j] είναι ίδια με την αρχική τιμή του R_i.comm[j] που η j διάβασε αρχικά, τότε επιστρέφεται το διάνυσμα των R_i.val που διαβάστηκαν στη 2^{η} ομάδα αναγνώσεων
- διαφορετικά η j καταγράφει τι αλλαγές έχουν γίνει
- αν καταγραφεί ότι κάποιος καταχωρητής R_i έχει αλλάξει δύο φορές (από αυτό που διαβάστηκε αρχικά), επιστρέφεται το R_i.view της τελευταίας ομάδας αναγνώσεων.

Αλγόριθμος που χρησιμοποιεί καταχωρητές πεπερασμένου μεγέθους

Update(v) από την i:

- για κάθε j, διάβασε το R_j'.ack[i] και υπολόγισε ένα διάνυσμα **b** των n bits όπου για κάθε j, **b**[j] = NOT(R_j'.ack[i])
- εκτέλεσε μια embedded SCAN
- εκτέλεσε ένα write για να γράψεις στον R_i τα εξής:
- R_i.val = v
- R_i.comm = **b**
- R_i.toggle = NOT(R_i.toggle)
- R_i.view = το διάνυσμα που υπολογίστηκε από την embedded SCAN
- Επέστρεψε acknowledgement

Αλγόριθμος που χρησιμοποιεί καταχωρητές πεπερασμένου μεγέθους

Θεώρημα: Ο αλγόριθμος BoundedSnapshot αποτελεί σωστή υλοποίηση ενός ατομικού στιγμιοτύπου μνήμης και επιτυγχάνει τερματισμό ελεύθερο αναμονής.

Απόδειξη

Τα σημεία σειριοποίησης εισάγονται με τον ίδιο τρόπο όπως και στον UnboundedSnapshot.

Όπως στον UnboundedSnapshot, αποδεικνύεται ότι για κάθε λειτουργία, το σημείο σειριοποίησης της είναι μέσα στο διάστημα εκτέλεσής της.

Τώρα είναι ενδιαφέρον να αποδειχθεί ότι εύκολες SCAN και embedded SCAN είναι συνεπείς. Για τις δύσκολες SCAN (και embedded SCAN) τα επιχειρήματα είναι ίδια με εκείνα του UnboundedSnapshot.

Λήμμα: Κάθε καλή SCAN (ή embedded SCAN) επιστρέφει ένα συνεπές διάνυσμα τιμών.

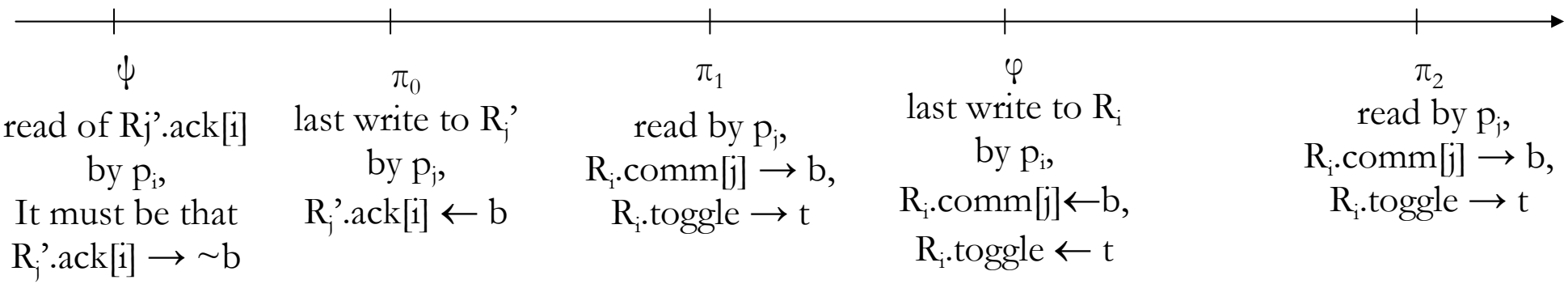
Αλγόριθμος που χρησιμοποιεί καταχωρητές πεπερασμένου μεγέθους

Απόδειξη Λήμματος: Με εις άτοπο απαγωγή.

Έστω ότι μια SCAN από την p_j επιστρέφει επειδή δεν είδε αλλαγή σε δύο διαδοχικές ομάδες αναγνώσεων.

π_1, π_2: τα 2 reads από την p_j στον R_i (στην $1^η$ και στη $2^η$ ομάδα αναγνώσεων, αντίστοιχα).

Ας υποθέσουμε ότι τουλάχιστον μια write από την p_i συμβαίνει μεταξύ των π_1 και π_2. Έστω φ το τελευταίο τέτοιο write από την p_i (που προηγείται του π_2).



Οι ψ και φ είναι εντολές της ίδιας UPDATE. Άρα, οι π_1 και π_2 διαβάζουν ότι έγραψε η p_i σε δύο διαδοχικές write εντολές. Τότε όμως το R_i.toggle θα έπρεπε να είναι διαφορετικό στις π_1, π_2.