## 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

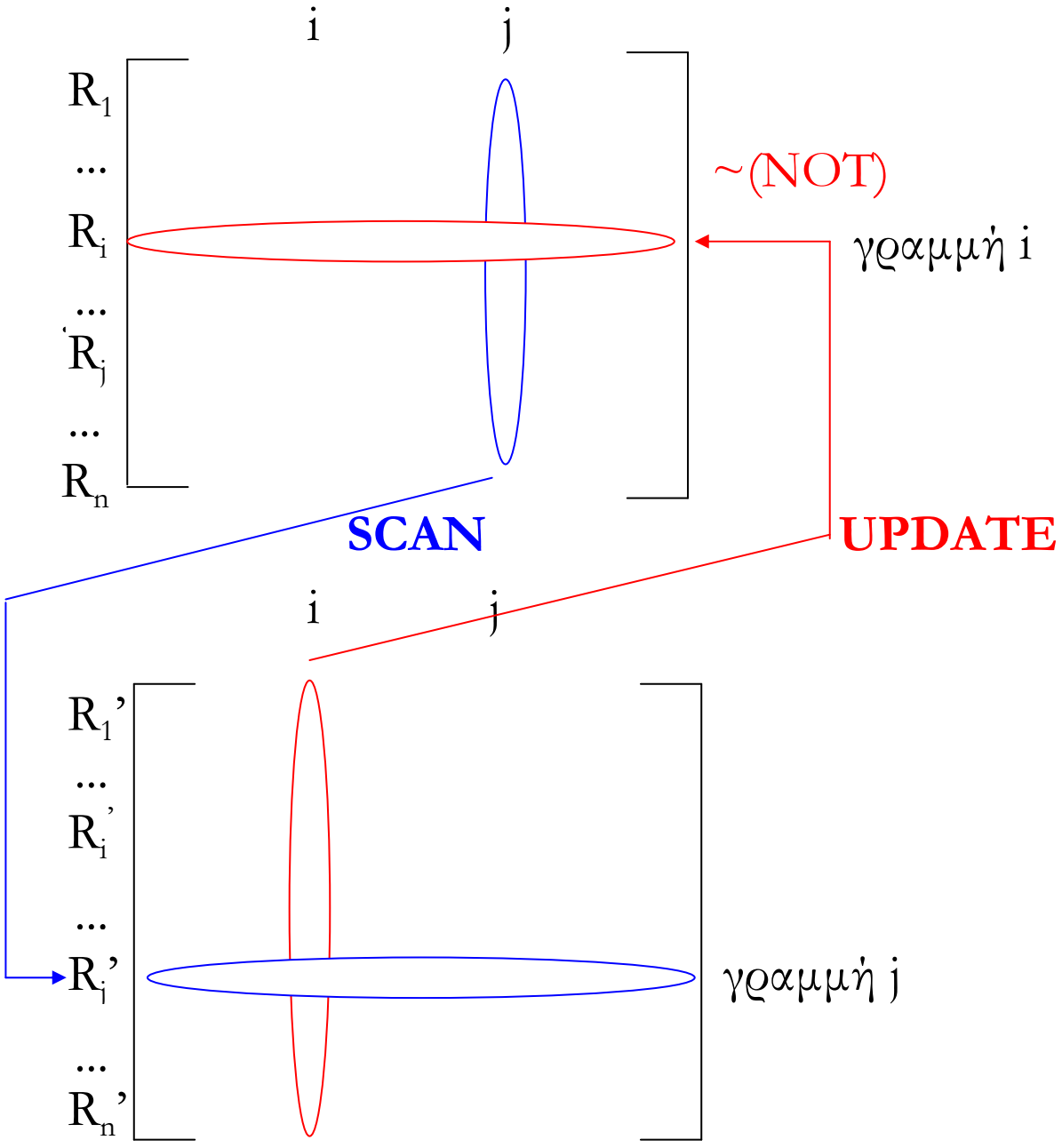
**Κατά την εκτέλεση μιας UPDATE(v) από την $p_i$:**

- 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



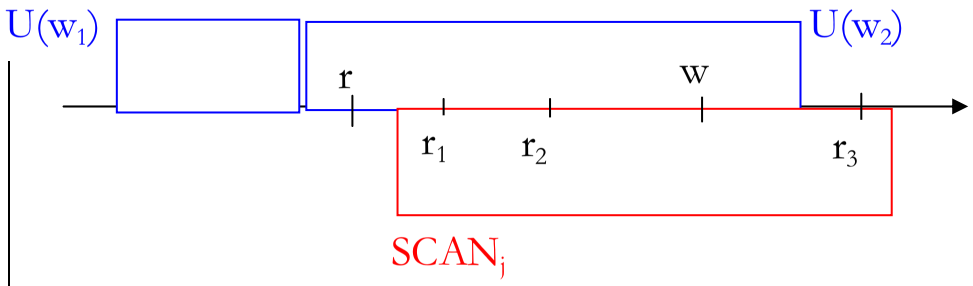

Στον παραπάνω αλγόριθμο, κάθε φορά που μια διεργασία που εκτελεί SCAN βλέπει μια αλλαγή, κάποια UPDATE έχει συμβεί!

# 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.

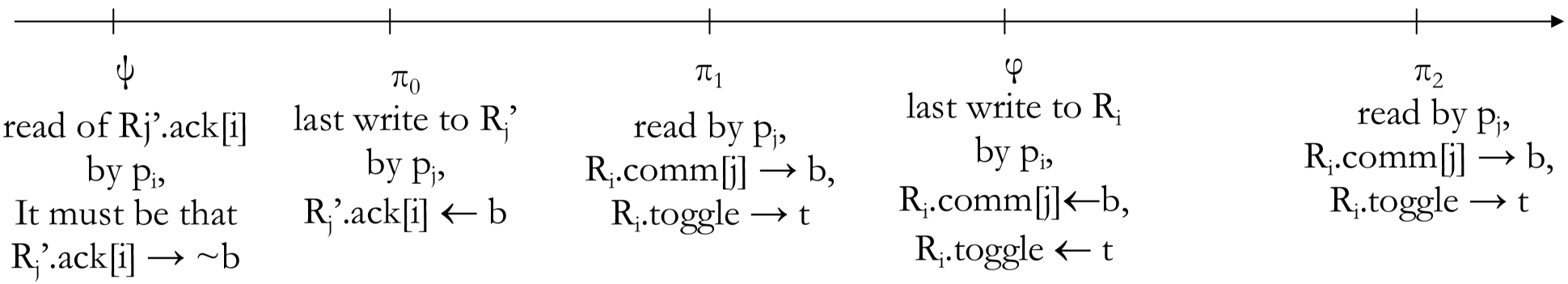
## Αλγόριθμος που χρησιμοποιεί καταχωρητές πεπερασμένου μεγέθους

**Απόδειξη Λήμματος:** Με εις άτοπο απαγωγή.

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

$\pi_1$, $\pi_2$: 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 $\pi_1$ και $\pi_2$. Έστω φ το τελευταίο τέτοιο write από την $p_i$ (που προηγείται του $\pi_2$).




Οι ψ και φ είναι εντολές της ίδιας UPDATE. Άρα, οι $\pi_1$ και $\pi_2$ διαβάζουν ότι έγραψε η $p_i$ σε δύο διαδοχικές write εντολές. Τότε όμως το $R_i$.toggle θα έπρεπε να είναι διαφορετικό στις $\pi_1$, $\pi_2$.