ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

Βάσεις Δεδομένων ΙΙ

Διαχείριση Συναλλαγών

Μανώλης Μαραγκουδάκης

Τμήμα Μηχανικών Πληροφοριακών και Επικοινωνιακών Συστημάτων

Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Πανεπιστήμιο Αιγαίου**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Πανεπιστήμιο Αιγαίου-
Τμήμα Μηχανικών
Πληροφοριακών και
Επικοινωνιακών Συστημάτων

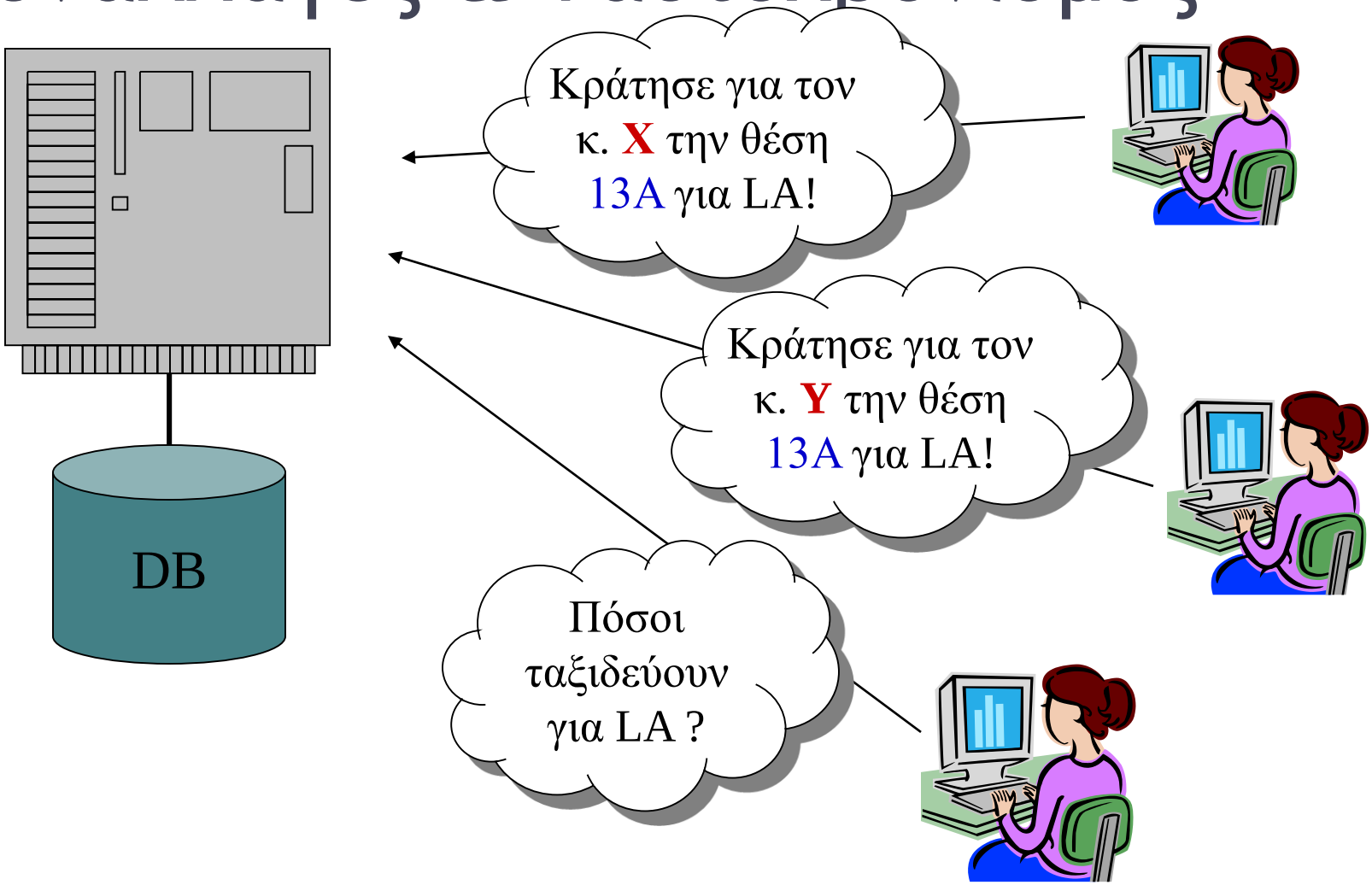
Βάσεις Δεδομένων ΙΙ

Ενότητα 1: Διαχείριση Συναλλαγών

Μανώλης Μαραγκουδάκης

www.icsd.aegean.gr/mmarag

Συναλλαγές & Ταυτοχρονισμός



Περιεχόμενα

- Η έννοια της συναλλαγής
- Ιδιότητες των συναλλαγών
- Καταστάσεις μιας συναλλαγής
- Χρονοπρογράμματα
- Σειριοποιησιμότητα
- Έλεγχος σειριοποιησιμότητας

Συναλλαγή ή Δοσοληψία

- **Συναλλαγή** είναι
  - μια σειρά από ενέργειες, οι οποίες
  - διαβάζουν ή γράφουν
  - αντικείμενα της βάσης
- συχνά: «δοσοληψία»,
- στα αγγλικά “transaction”
- σειρά: διατεταγμένο σύνολο, λίστα

Για προχωρημένους: στην θεωρία [της πληροφορικής γενικά, και των ΒΔ ειδικά], οι λέξεις τύπου «σειρά» έχουν σημασία ... (π.χ., σειρά ≠ σύνολο)

Παράδειγμα συναλλαγής

T_0: μεταφορά 50€ από το λογαριασμό Α στο λογαριασμό Β

```
read(A);
A := A - 50;
write(A);
read(B);
B := B + 50;
write(B).
```

Σε ότι αφορά τη ΒΔ:

R(*A*);**W**(*A*);**R**(*B*);**W**(*B*).

Προβληματισμός

- Δύο είναι τα βασικά προβλήματα με τις συναλλαγές:
  - Τι θα γίνει αν κατά τη διάρκεια της εκτέλεσης, πέσει το σύστημα?
  - Τι θα γίνει αν δύο συναλλαγές επιχειρούν να μεταβάλλουν το ίδιο αντικείμενο ταυτοχρόνως?

Ιδιότητες των συναλλαγών

Ατομικότητα: είτε **όλες** οι πράξεις της συναλλαγής επιτυγχάνουν, είτε **όλες** αποτυγχάνουν.

Συνέπεια: στο τέλος της συναλλαγής, η βάση πρέπει να είναι σε συνεπή μορφή.

Απομόνωση: ακόμα κι αν τρέχουν πολλές συναλλαγές ταυτόχρονα, **κάθε συναλλαγή πρέπει να νομίζει ότι τρέχει μόνη της**.

Μονιμότητα: αν η συναλλαγή επιτύχει, πρέπει το αποτέλεσμά της να επιβιώνει, ακόμα κι αν αποτύχει το σύστημα.

ACID Test

- (**A**)tomicity — Ατομικότητα
- (**C**)onsistency — Συνέπεια
- (**I**)solation — Απομόνωση
- (**D**)urability — Μονιμότητα

*γνωστό ως **ACID** test…*

ACID Test

- (**A**)tomicity — Ατομικότητα
- (**C**)onsistency — Συνέπεια
- (**I**)solation — Απομόνωση
- (**D**)urability — Μονιμότητα

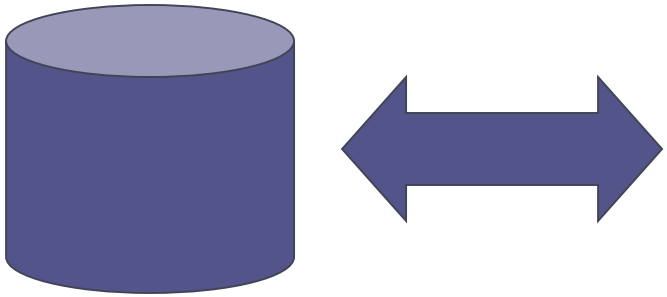
Συνέπεια

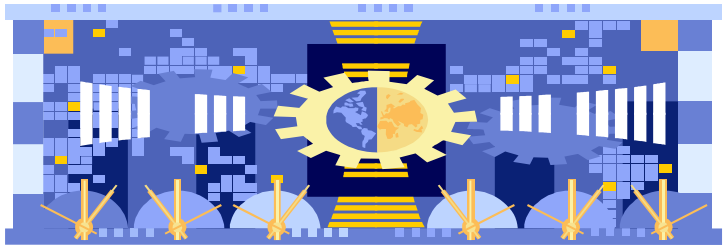
- Μια βάση δεδομένων διέπεται από κανόνες ακεραιότητας (π.χ., πρωτεύοντος κλειδιού ...)
- Επιπλέον, υπάρχουν και λογικοί περιορισμοί, τους οποίους «κρύβουμε» στις εφαρμογές.

- *Πριν και μετά την εκτέλεση της συναλλαγής (αλλά όχι απαραίτητα ενδιαμέσως), όλοι οι περιορισμοί αυτοί, πρέπει να πληρούνται...*

Γιατί?

- Διότι υπάρχει η λανθάνουσα υπόθεση ότι η ΒΔ μοντελοποιεί πλήρως τον πραγματικό κόσμο!





Συνέπεια

EMP(EMP_ID,NAME,AGE,DEPT_ID,SALARY)

- Περιορισμοί ακεραιότητας:
  - EMP_ID πρωτεύον κλειδί
  - AGE <= 65
  - SALARY > 0

Συνέπεια

- Παράδειγμα λογικού περιορισμού:

κατά τη διάρκεια της μεταφοράς χρημάτων από ένα λογαριασμό σε ένα άλλο,

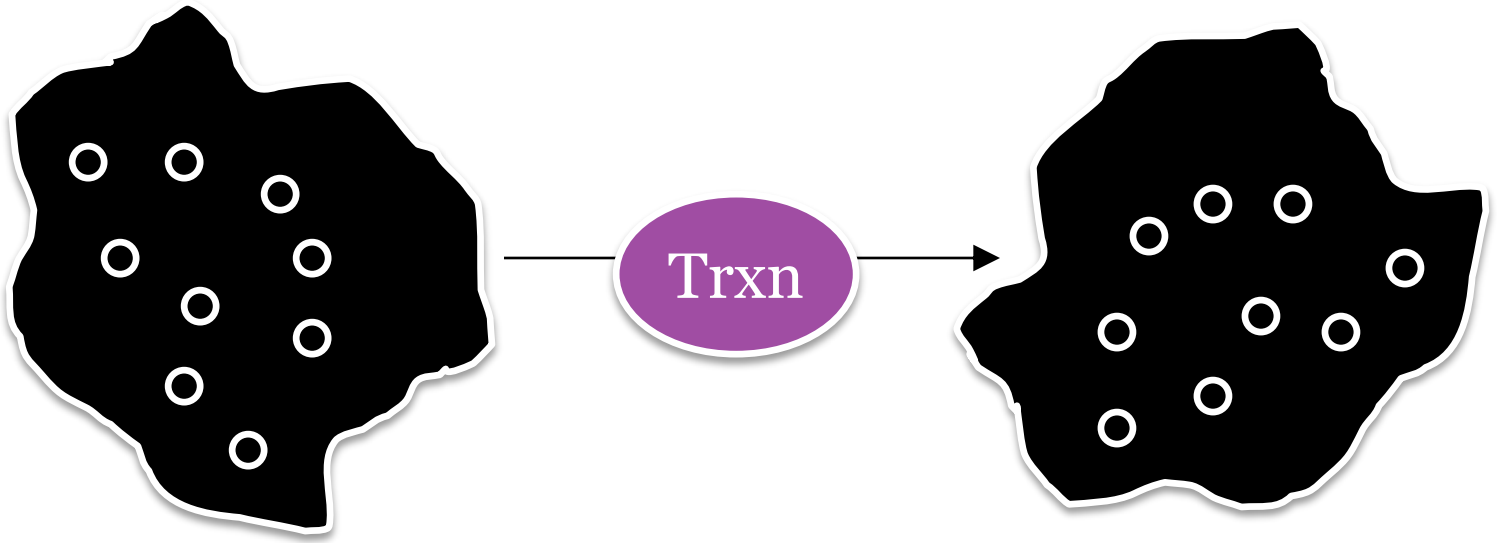
το άθροισμα των δύο λογαριασμών στο τέλος, πρέπει να ισούται με το άθροισμα τους στην αρχή

- **Συνεπής** [κατάσταση της] ΒΔ: όλοι οι περιορισμοί ικανοποιούνται!

Συνέπεια

Ανάμεσα σε άλλες απλοποιητικές υποθέσεις, κάνουμε και την υπόθεση ότι μια συναλλαγή που ξεκινά να τροποποιεί μια συνεπή ΒΔ, θα καταλήξει σε μια συνεπή ΒΔ, επίσης!!!

Συνέπεια

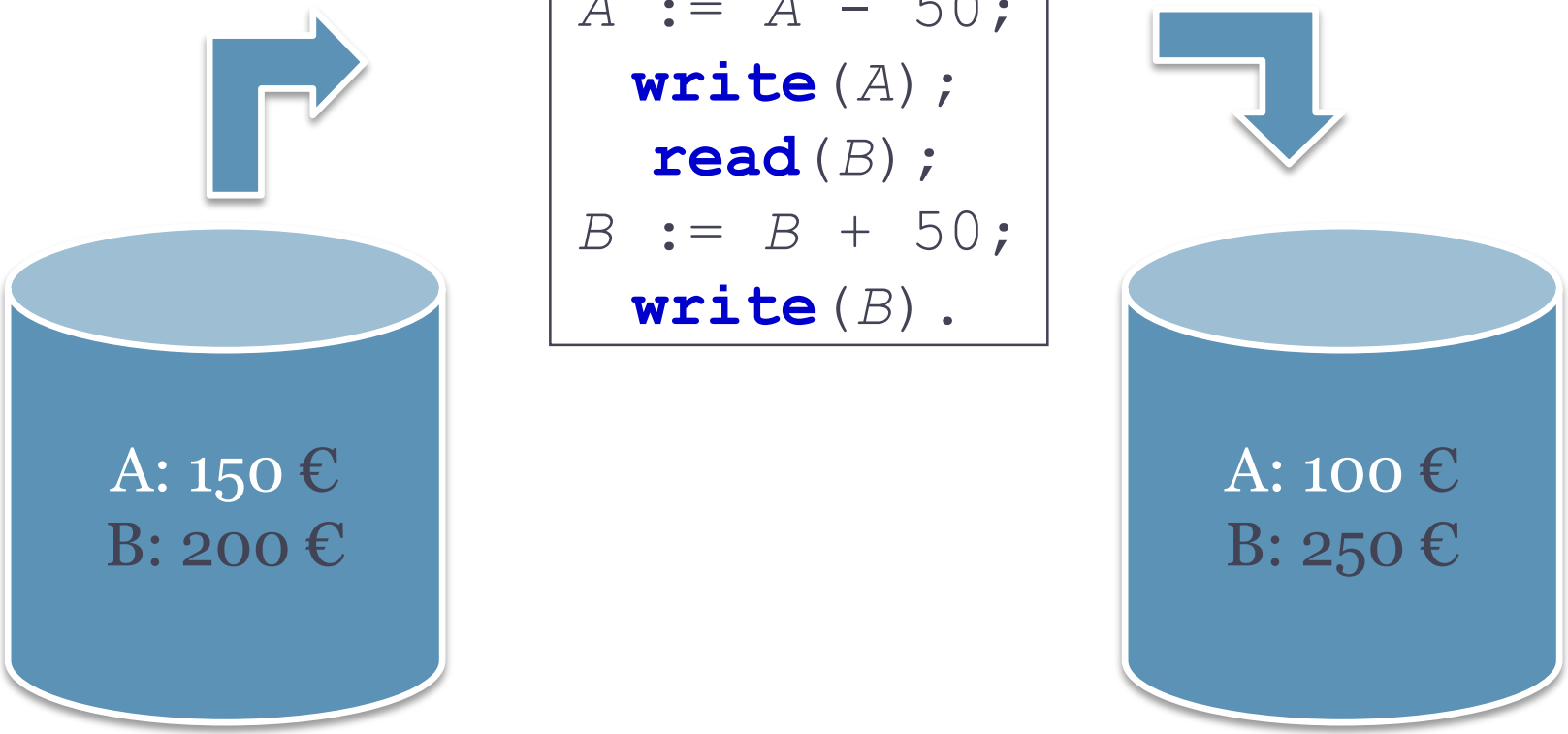


Συνεπής κατάσταση της βάσης πριν την συναλλαγή

Συνεπής κατάσταση της βάσης μετά την συναλλαγή

Συνέπεια

```
read(A);
A := A - 50;
write(A);
read(B);
B := B + 50;
write(B).
```



Υποθέσεις …

- Στο εξής, θα κάνουμε την εύλογη υπόθεση ότι το DBMS δεν έχει bugs, και
- την όχι τόσο εύλογη υπόθεση ότι οι προγραμματιστές γράφουν ορθές [από πλευράς συνέπειας] συναλλαγές…

Υπάρχουν όμως κι άλλα προβλήματα …

ACID Test

- (**A**)tomicity — Ατομικότητα
- (**C**)onsistency — Συνέπεια
- (**I**)solation — Απομόνωση
- (**D**)urability — Μονιμότητα

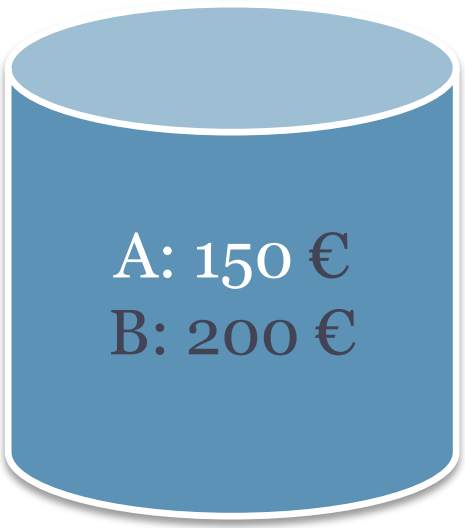
Ατομικότητα

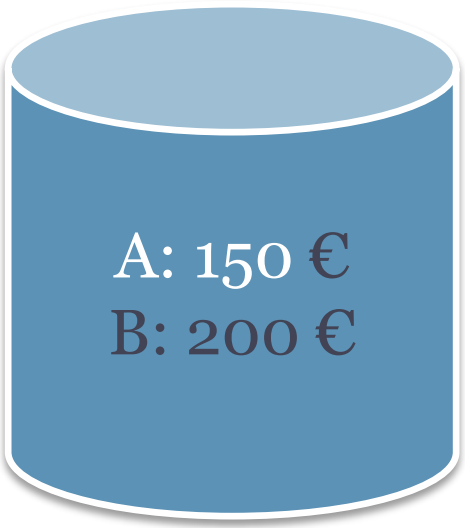
- Η συναλλαγή είναι μια **μονάδα εργασίας**
- Παρότι αποτελείται από πολλές ενέργειες, δεν είναι αποδεκτό να εκτελεστούν μόνο μερικές από αυτές

- Αυτό μπορεί να συμβεί, π.χ., γιατί στη διάρκεια εκτέλεσης, το σύστημα αποτυγχάνει
- Σαν αποτέλεσμα, κάποιοι κανόνες μπορεί να παραβιαστούν (και μαζί και η συνέπεια της βάσης)...

Ατομικότητα

```
1.read(A);
2.A := A - 50;
3.write(A);
```



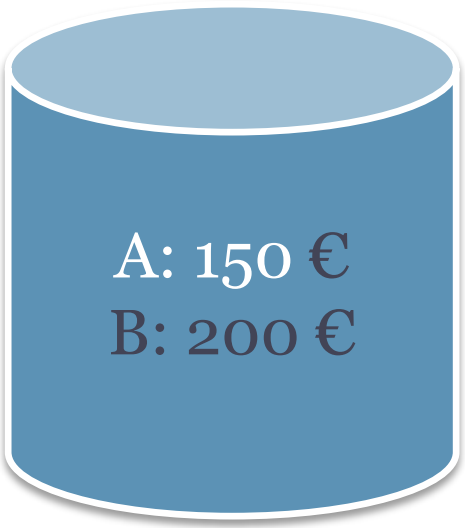


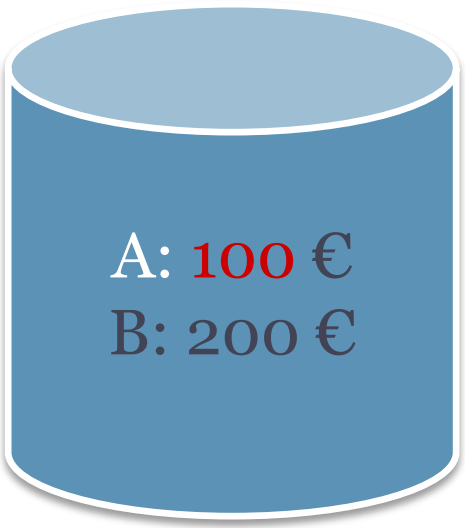
A+B=350

A+B=300

Ατομικότητα

```
1.read(A);
2.A := A - 50;
3.write(A);
```



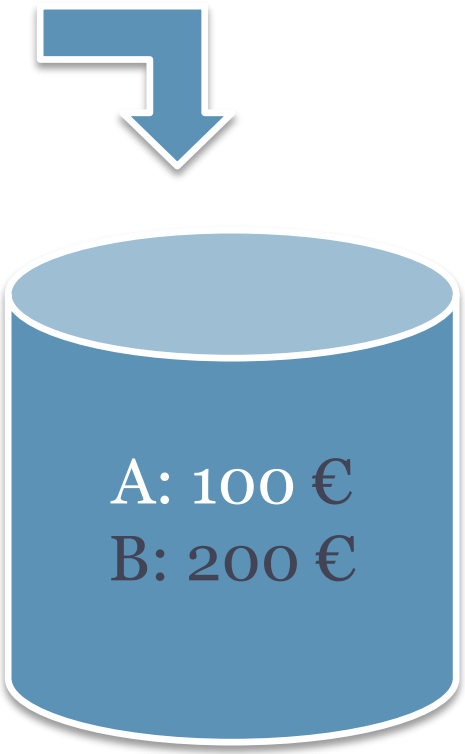


A+B=350

A+B=300

Ατομικότητα

```
   read(A);
A := A - 50;
  write(A);
  --CRASH--
   read(B);
B := B + 50;
  write(B).
```



A+B=350 ⟷ A+B=300

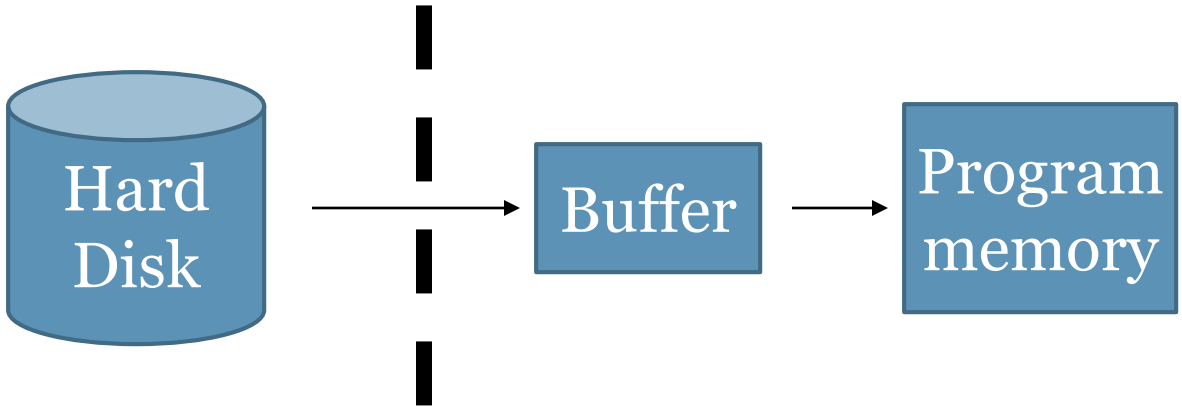
Ατομικότητα

- Το DBMS εξασφαλίζει ότι η ατομικότητα θα διατηρηθεί, **αναιρώντας** όλες τις συναλλαγές που αποτυγχάνουν
- Το πώς γίνεται αυτό, θα το δούμε στο κεφάλαιο της ανάνηψης...

Παράπλευρη παρατήρηση

- Τι πάει να πει `read(A)` ?
- `A` είναι μια μεταβλητή του προγράμματος
- `read(A)` σημαίνει:
  - Διάβασε από το δίσκο την αντίστοιχη με το **`A`** εγγραφή στη βάση,
  - Φέρε την σε κάποιο buffer
  - Αντίγραψέ την στην περιοχή μνήμης του προγράμματος

Παράπλευρη παρατήρηση



- Το αντίστοιχο συμβαίνει και με τη `write`
- Όπως θα δούμε, το `A` μπορεί και να μην είναι εγγραφή, αλλά π.χ., σελίδα στο δίσκο ...

Βασικές Λειτουργίες Συναλλαγών

read(X):

- Εύρεση της διεύθυνσης του μπλοκ που περιέχει το X
- Αντιγραφή αυτού του μπλοκ σε μία ενδιάμεση μνήμη (buffer) – αν δεν είναι ήδη εκεί.
- Αντιγραφή του στοιχείου X από το buffer στη μεταβλητή με όνομα X

Βασικές Λειτουργίες Συναλλαγών

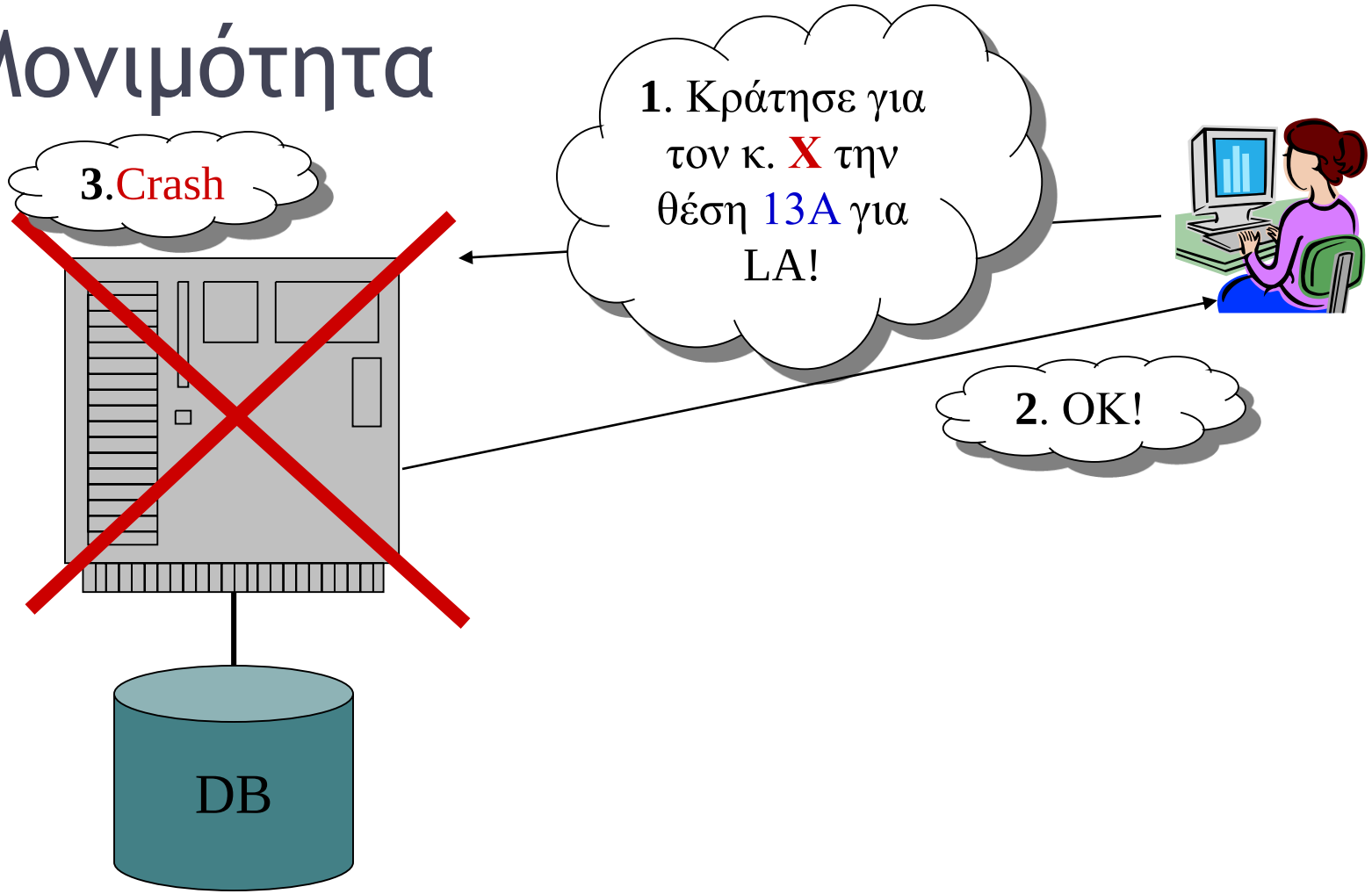
write(X):

- Εύρεση της διεύθυνσης του μπλοκ που περιέχει το X
- Αντιγραφή αυτού του μπλοκ σε μία ενδιάμεση μνήμη (buffer) – αν δεν είναι ήδη εκεί
- Αντιγραφή του στοιχείου X από τη μεταβλητή στην κατάλληλη θέση μέσα στο buffer
- Αποθήκευση του αλλαγμένου μπλοκ από το Buffer στο δίσκο (άμεσα ή αργότερα)

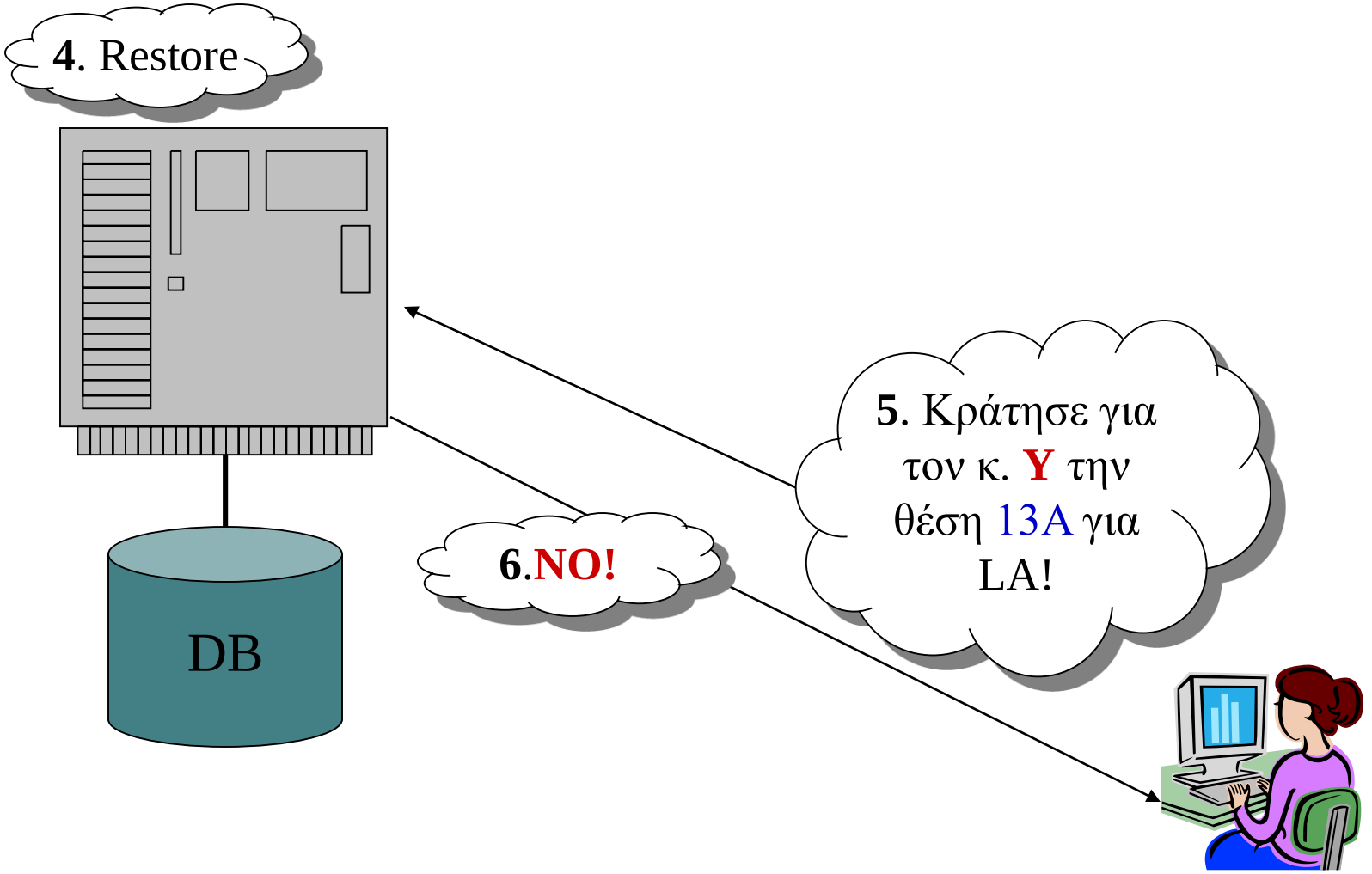
ACID Test

- (**A**)tomicity — Ατομικότητα
- (**C**)onsistency — Συνέπεια
- (**I**)solation — Απομόνωση
- (**D**)urability — Μονιμότητα

Μονιμότητα



Μονιμότητα



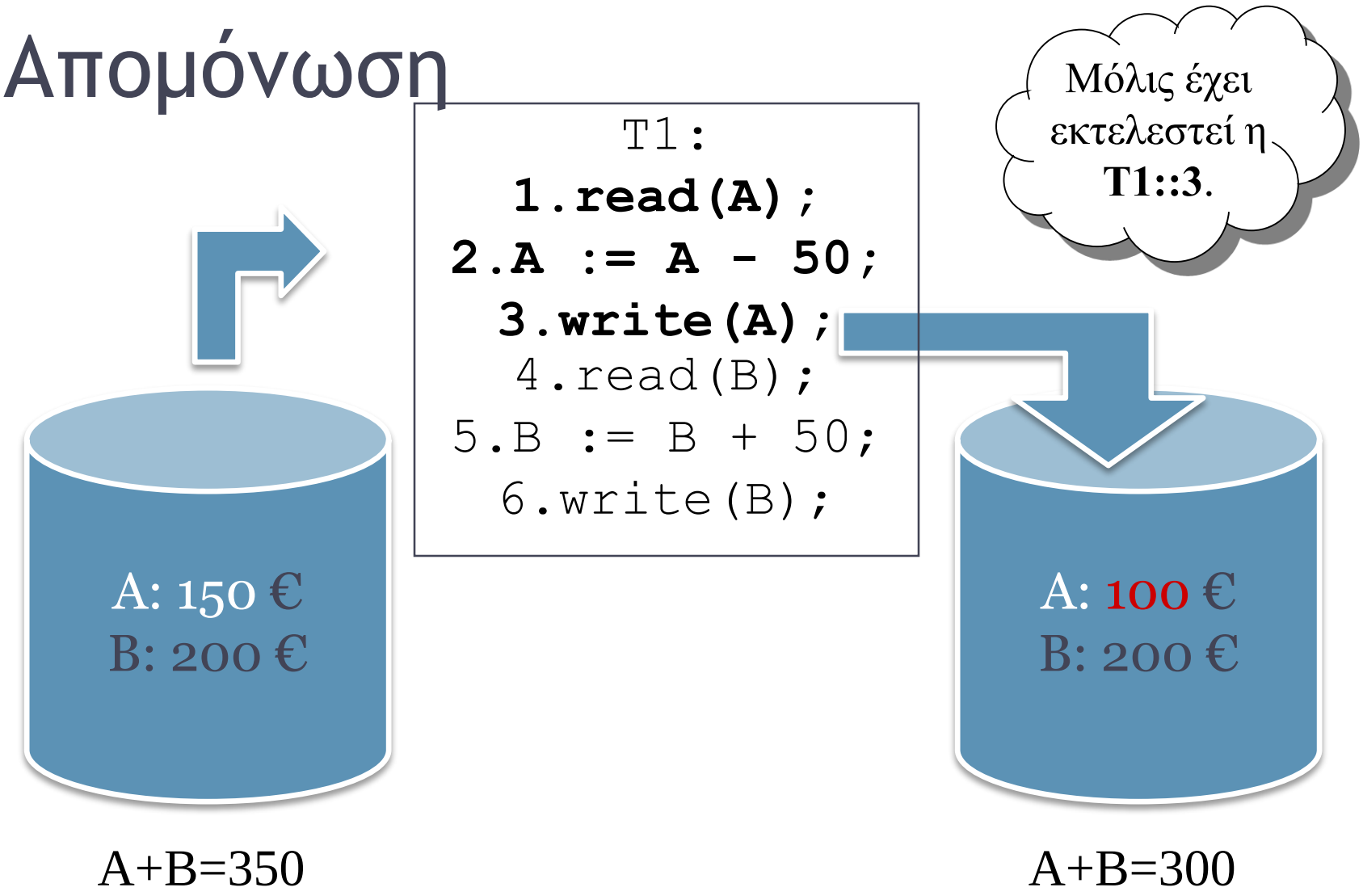
ACID Test

- (**A**)tomicity — Ατομικότητα
- (**C**)onsistency — Συνέπεια
- (**I**)solation — Απομόνωση
- (**D**)urability — Μονιμότητα

Απομόνωση

- Ας υποθέσουμε ότι στο σύστημα τρέχουν περισσότερες από μια συναλλαγές ταυτοχρόνως.
- Αν μια από αυτές μπορέσει να δει τα ενδιάμεσα αποτελέσματα της άλλης, τότε μπορεί να έχουμε ανεπιθύμητα αποτελέσματα (διότι ενδιαμέσως στη συναλλαγή η βάση μπορεί να είναι ασυνεπής).
- Γι' αυτό, θα θέλαμε, ιδεατά, οι συναλλαγές να τρέχουν η μία μετά την άλλη **σειριακά**.
- Για λόγους απόδοσης, όμως, αυτό δεν γίνεται ...

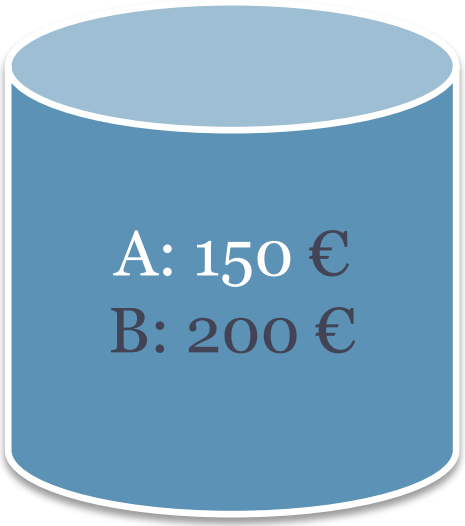
Απομόνωση


Μόλις έχει εκτελεστεί η T1::3.
T1:
1.read(A);
2.A := A - 50;
3.write(A);
4.read(B);
5.B := B + 50;
6.write(B);
A: 150 €
B: 200 €
A: 100 €
B: 200 €
A+B=350
A+B=300

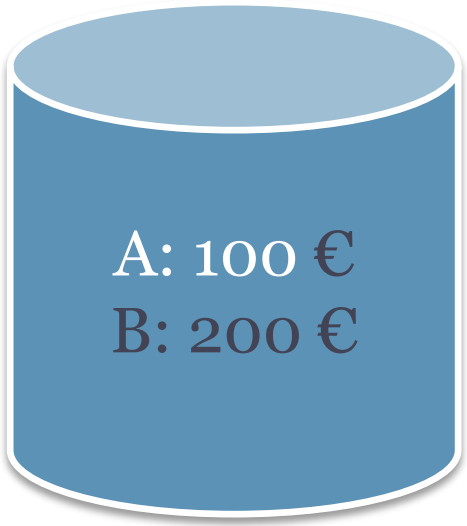
Απομόνωση

```
T1: idle
[hold at T1::3]
```

Μόλις έχει εκτελεστεί η **T1::3**.




```
T2:
1.read(B);
2.If B<220
    B:=B*0.10;
3.write(B);
```



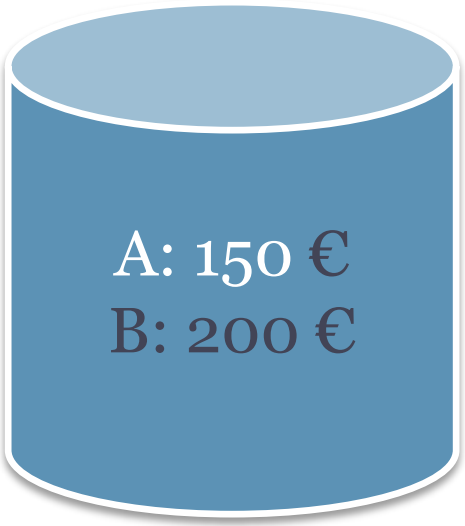
A+B=350

A+B=300

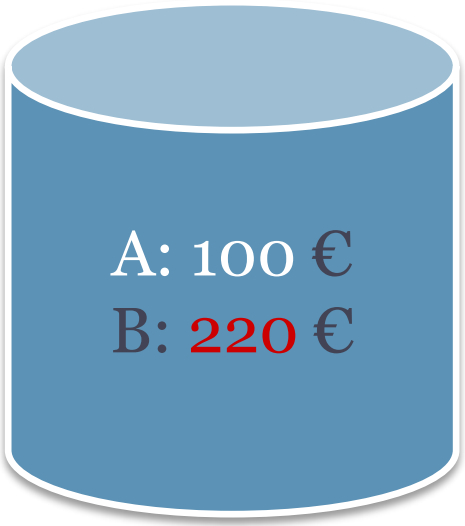
Απομόνωση

```
T1: idle
[hold at T1::3]
```

Μόλις έχει εκτελεστεί η **T2::3**.



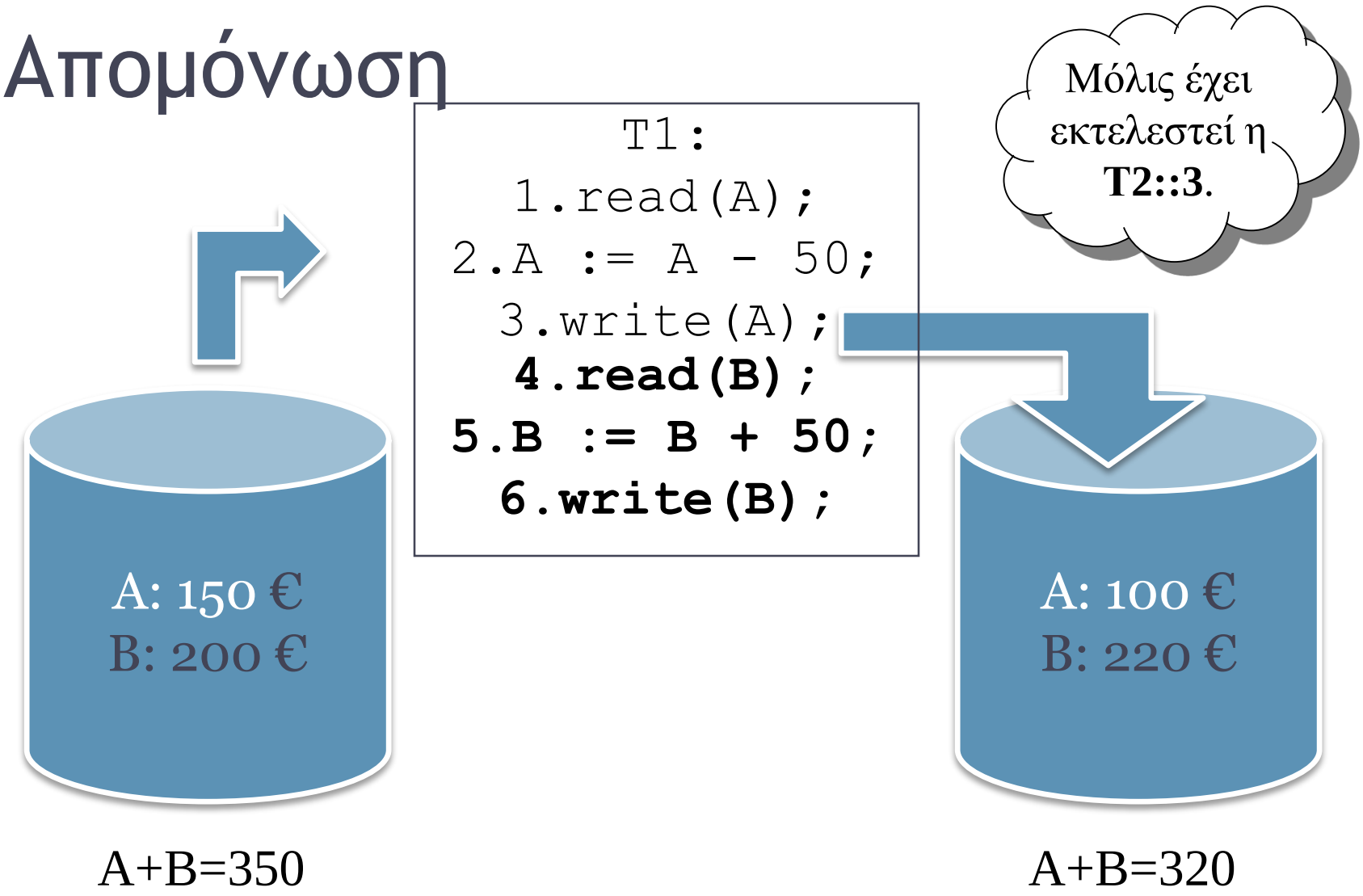

```
T2:
1.read(B);
2.If B<220
    B:=B*0.10;
3.write(B);
```



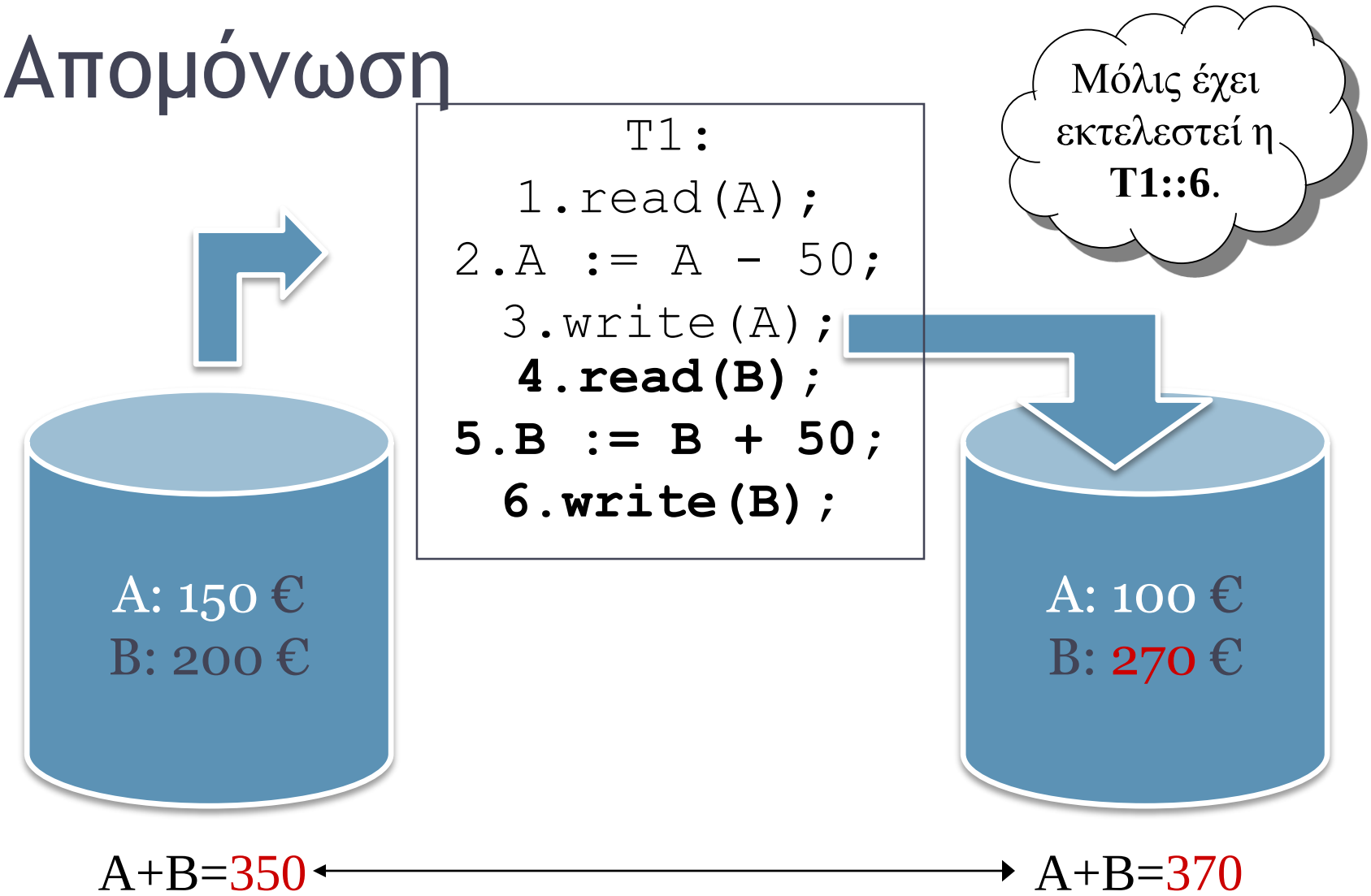
A+B=350

A+B=320

Απομόνωση


Μόλις έχει εκτελεστεί η **T2::3**.
T1:
1.read(A);
2.A := A - 50;
3.write(A);
4.read(B);
5.B := B + 50;
6.write(B);
A: 150 €
B: 200 €
A: 100 €
B: 220 €
A+B=350
A+B=320

Απομόνωση

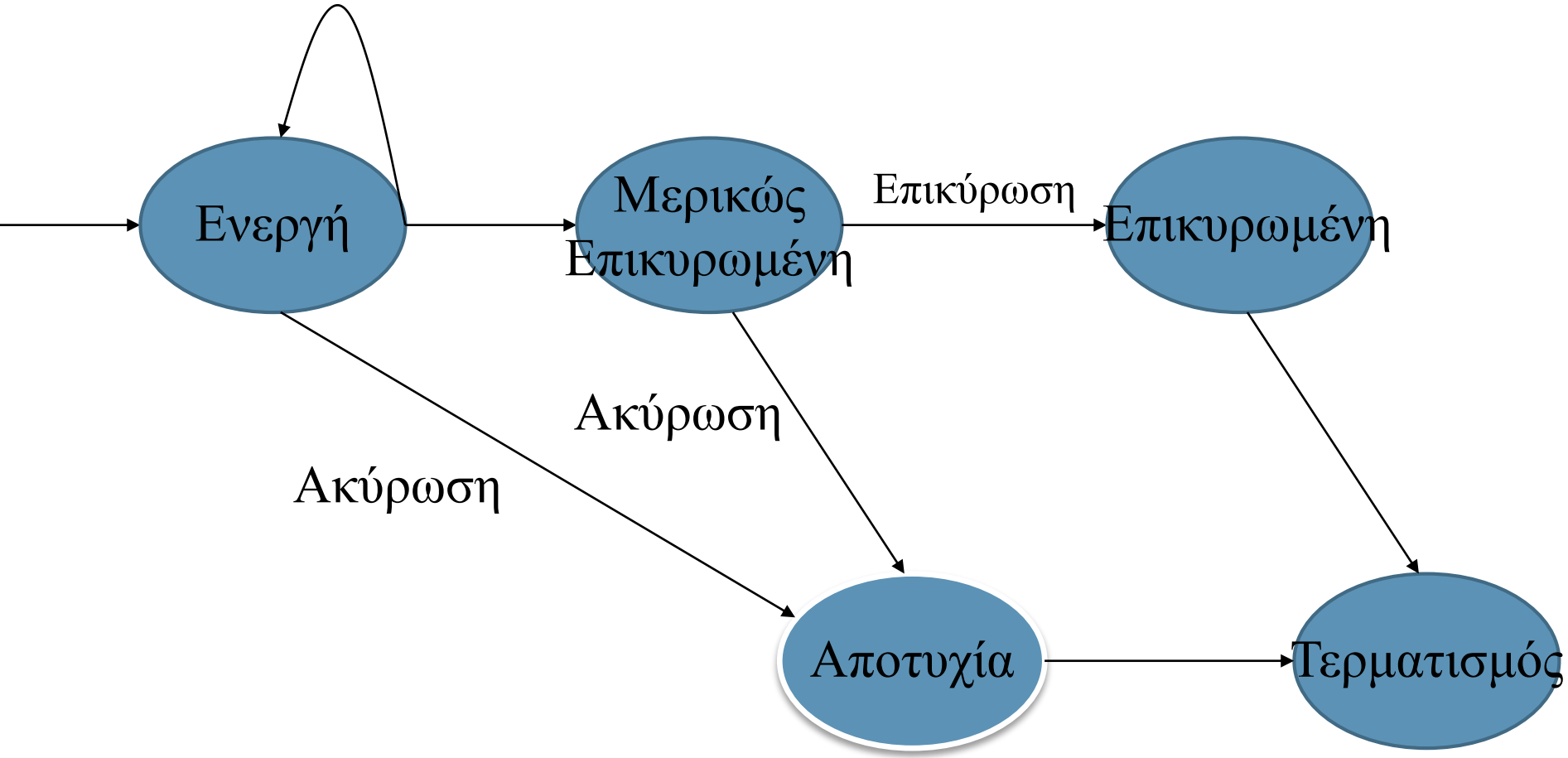

Μόλις έχει εκτελεστεί η **T1::6**.
T1:
1.read(A);
2.A := A - 50;
3.write(A);
4.read(B);
5.B := B + 50;
6.write(B);
A: 150 €
B: 200 €
A: 100 €
B: 270 €
A+B=350
A+B=370

Απομόνωση

Και γιατί να μην τρέχουμε σειριακά τις συναλλαγές, τη μία μετά την άλλη?

- Παράλληλη χρήση της CPU και του I/O
- Οι σύντομες συναλλαγές, δεν έχουν λόγο να αναμένουν την ολοκλήρωση των πιο χρονοβόρων
- Έχουμε έξυπνους αλγόριθμους διαπλοκής των συναλλαγών... [σε επόμενο κεφάλαιο]

Καταστάσεις μιας συναλλαγής



Καταστάσεις μιας συναλλαγής

- **Active**: στο ξεκίνημα και κατά τη διάρκειά της
- **Failed**: όταν το DBMS αντιληφθεί ότι η συναλλαγή δεν μπορεί να συνεχίσει
- **Aborted**: όταν η αποτυχημένη συναλλαγή έχει αναιρεθεί από το σύστημα και η ΒΔ είναι σε συνεπή μορφή
- **Committed**: όταν η συναλλαγή επιτύχει και η ΒΔ είναι σε συνεπή μορφή.

Καταστάσεις μιας συναλλαγής

- **Partially committed**: όταν έχει εκτελεστεί η τελευταία εντολή της συναλλαγής

Λεπτή διάκριση με την committed, θα επανέλθουμε στο κεφάλαιο της ανάκαμψης…

Συναλλαγή – Ορθή επαναδιατύπωση

- **Συναλλαγή** είναι
  - μια σειρά από ενέργειες, οι οποίες
  - διαβάζουν ή γράφουν
  - αντικείμενα της βάσης
  - και η οποία τελειώνει είτε με COMMIT, είτε με ABORT

Συμβολισμός

- $R_X(A)$: η συναλλαγή X διαβάζει το αντικείμενο A
- $W_X(A)$: η συναλλαγή X γράφει το αντικείμενο A
- $COMMIT_X$: η συναλλαγή X τερματίζει επιτυχώς
- $ABORT_x$: η συναλλαγή X αποτυγχάνει

Π.χ.,

$R_4(r3)$: η συναλλαγή T4 διαβάζει το αντικείμενο r3

Χρονοπρογράμματα

- **Χρονοπρόγραμμα** είναι
  - μια σειρά από ενέργειες (read, write, commit, abort)
  - μιας ομάδας συναλλαγών
  - όπου εμφανίζονται όλες οι ενέργειες αυτών των συναλλαγών
  - διατηρώντας τη σειρά με την οποία εμφανίζονται σε κάθε συναλλαγή

Στην αγγλική: "schedule"

Παράδειγμα

- Συναλλαγή T_1: R(A);R(B);W(A);COMMIT
- Συναλλαγή T_2: R(A);R(B);W(B);COMMIT

- Schedule S_1:
  $R_1(A);R_1(B);W_1(A);C1;R_2(A);R_2(B);W_2(B);C2.$
- Schedule S_2:
  $R_2(A);R_2(B);W_2(B);C2;R_1(A);R_1(B);W_1(A);C1.$
- Schedule S_3:
  $R_1(A);R_1(B);R_2(A);W_1(A);R_2(B);C1;W_2(B);C2.$

Αντιπαράδειγμα

- Συναλλαγή T_1: R(A);R(B);W(A);COMMIT
- Συναλλαγή T_2: R(A);R(B);W(B);COMMIT

- Schedule S_4:
  $R_1(B);W_1(A);C1;R_2(A);R_2(B);W_2(B);C2.$
- Schedule S_5:
  $W_2(B);R_1(A);R_1(B);R_2(A);W_1(A);R_2(B);C1;C2.$

Χρονοπρόγραμμα

- Schedule S_3:
  $R_1(A);R_1(B);R_2(A);W_1(A);R_2(B);C1;W_2(B);C2.$

- Ένα χρονοπρόγραμμα περιγράφει *τι βλέπει το DBMS* και όχι τι προγραμματίζουμε εμείς!

Παράδειγμα

T1: μεταφέρει €50 από Α σε Β

T2: μεταφέρει 10% του Α στο Β

Συνέπεια: Α+Β σταθερό

T_1	T_2
	read(*A*);
	temp := *A* * 0.1;
	A := *A* - *temp*;
	write(*A*);
	read(*B*);
	B := *B* + *temp*;
	write(*B*);
read(*A*);	
A := *A* - 50;	
write(*A*);	
read(*B*);	
B := *B* + 50;	
write(*B*).	

Παραδείγματα από Siberschatz, Korth & Sudarsan

Σειριακό Χρονοπρόγραμμα

Serial Schedule: Όταν οι ενέργειες που ανήκουν σε μια συναλλαγή εμφανίζονται ***σειριακά*** η μία με την άλλη

Εναλλακτικά: όταν οι συναλλαγές εκτελούνται εξ ολοκλήρου η μία μετά την άλλη

T_1	T_2
	read(A);
	$temp := A * 0.1$;
	$A := A - temp$;
	write(A);
	read(B);
	$B := B + temp$;
	write(B);
read(A);	
$A := A - 50$;	
write(A);	
read(B);	
$B := B + 50$;	
write(B).	

Σειριακό Χρονοπρόγραμμα

για n συναλλαγές

n! δυνατά σειριακά χρονοπρογράμματα

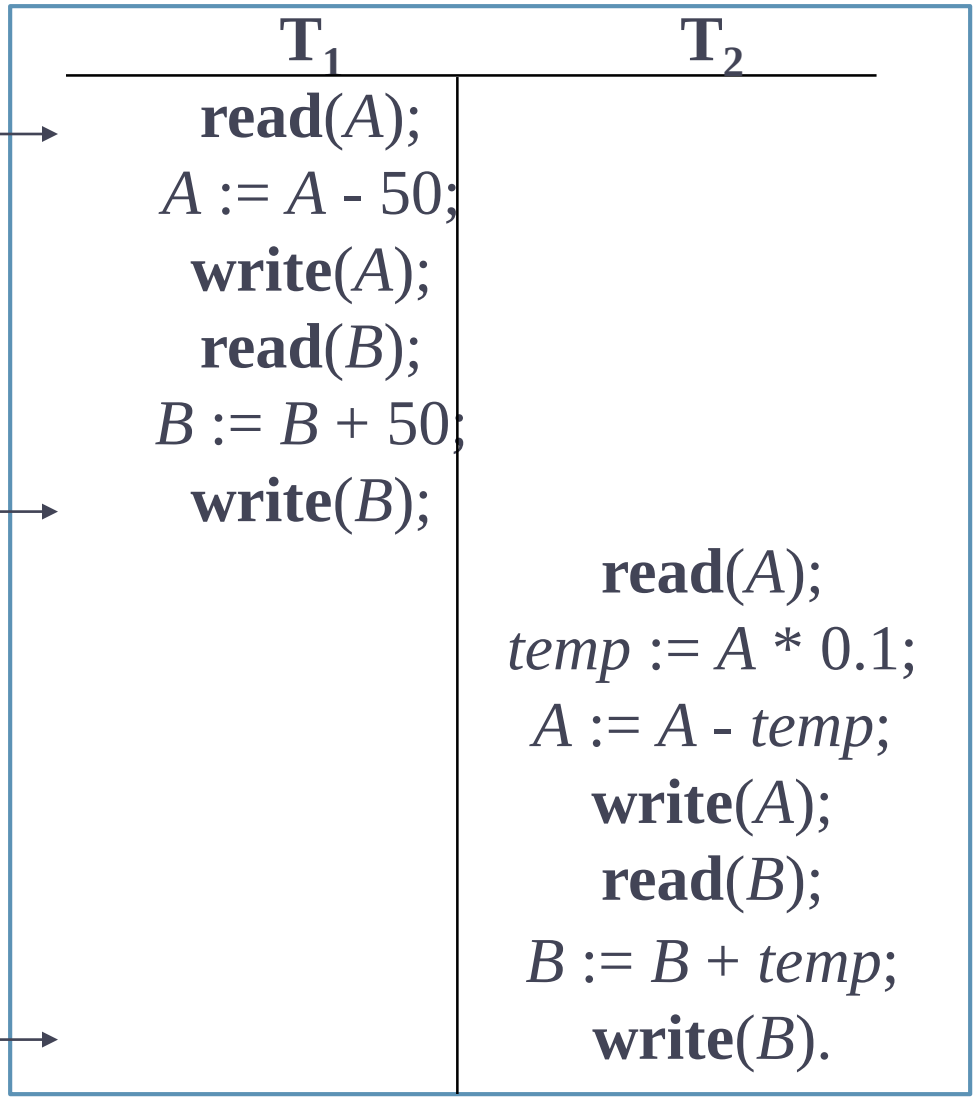
T_1	T_2
read(A);	
$A := A - 50$;	
write(A);	
read(B);	
$B := B + 50$;	
write(B);	
	read(A);
	$temp := A * 0.1$;
	$A := A - temp$;
	write(A);
	read(B);
	$B := B + temp$;
	write(B).

Σειριακό Χρονοπρόγραμμα

A:150, B:200,
A+B: 350

A:100, B:250,
A+B: 350

A:90, B:260,
A+B: 350

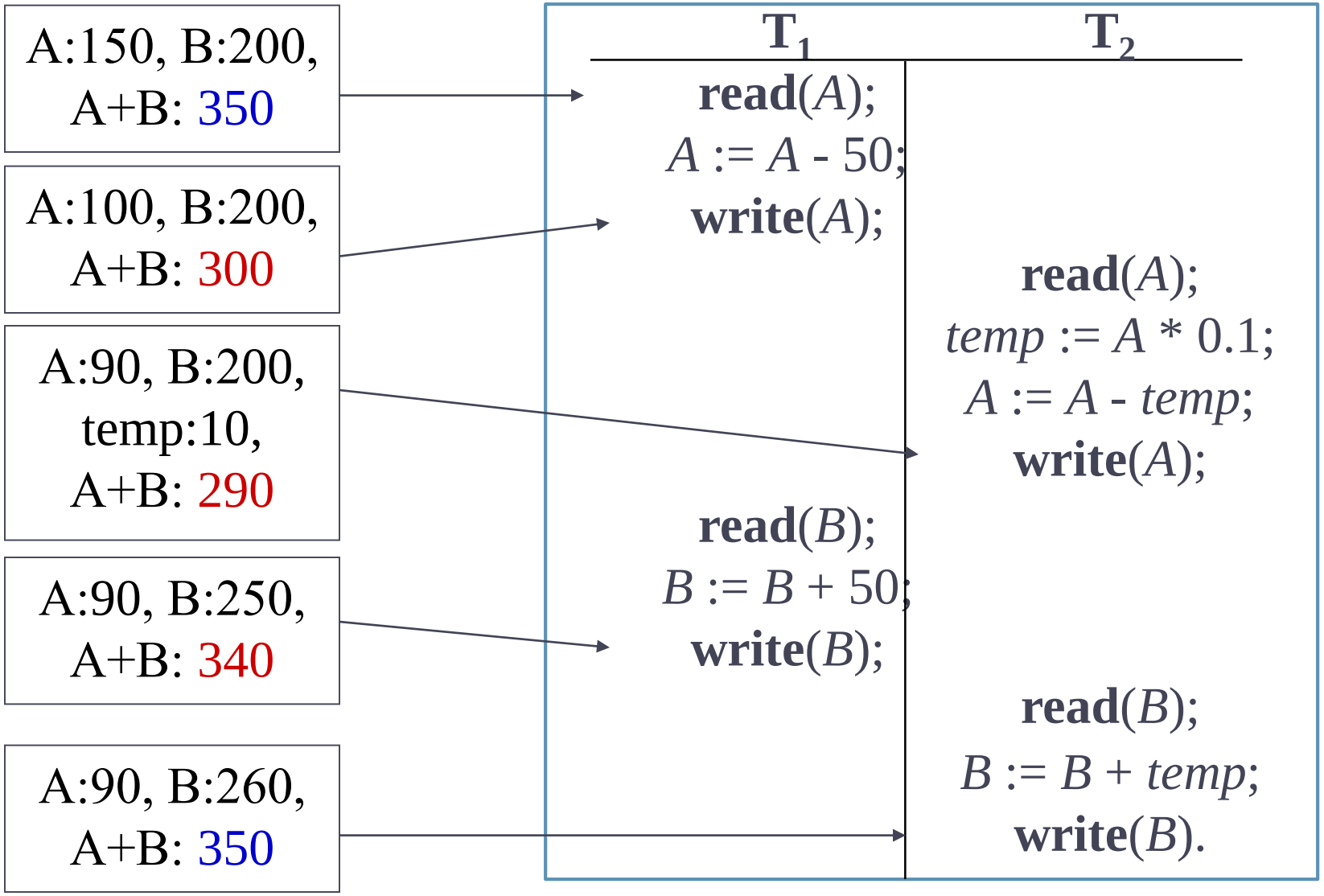


Προσοχή!!

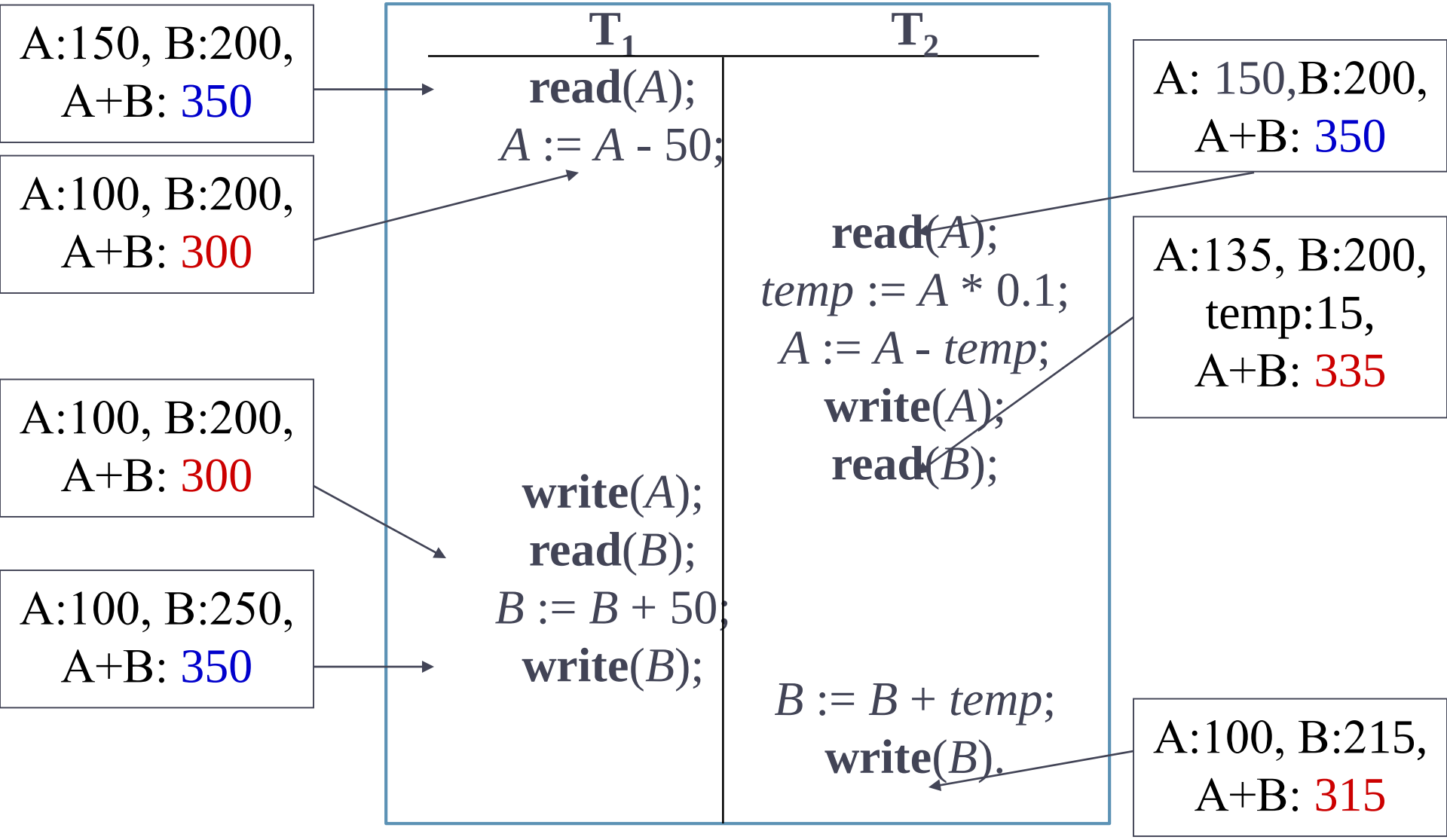
- Κάθε συναλλαγή στο χρονοπρόγραμμα, είναι σαν ***συνάρτηση***: έχει δικό της χώρο μνήμης

[γι' αυτό και η τιμή των A, B εξαρτάται ΜΟΝΟ από τα read, write και τις τοπικές μεταβολές – και όχι από τις αλλαγές σε άλλες συναλλαγές]

Μη Σειριακό Χρονοπρόγραμμα



Προβληματικό Χρονοπρόγραμμα



3 ειδών προβλήματα με τα χρονοπρογράμματα:

- Ασυνεπείς αναγνώσεις (dirty reads)
- Απώλειες ενημερώσεων (lost updates)
- Μη επαναλήψιμες αναγνώσεις (non-repeatable reads)

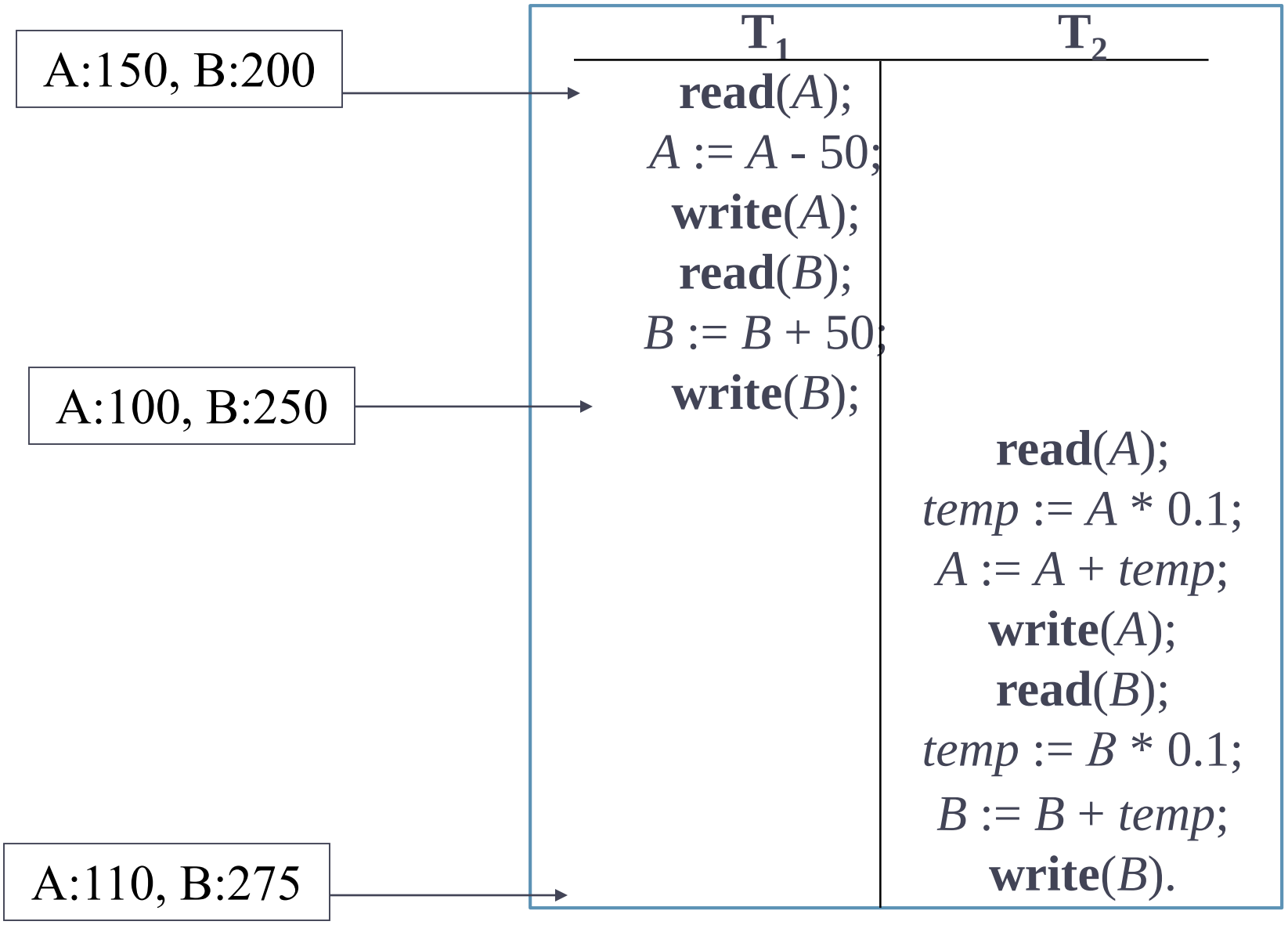
Παράδειγμα

- Τ1: μεταφέρει 50 € από τον Α στον Β
- Τ2: κάνει αύξηση στον Α και στον Β κατά 10%

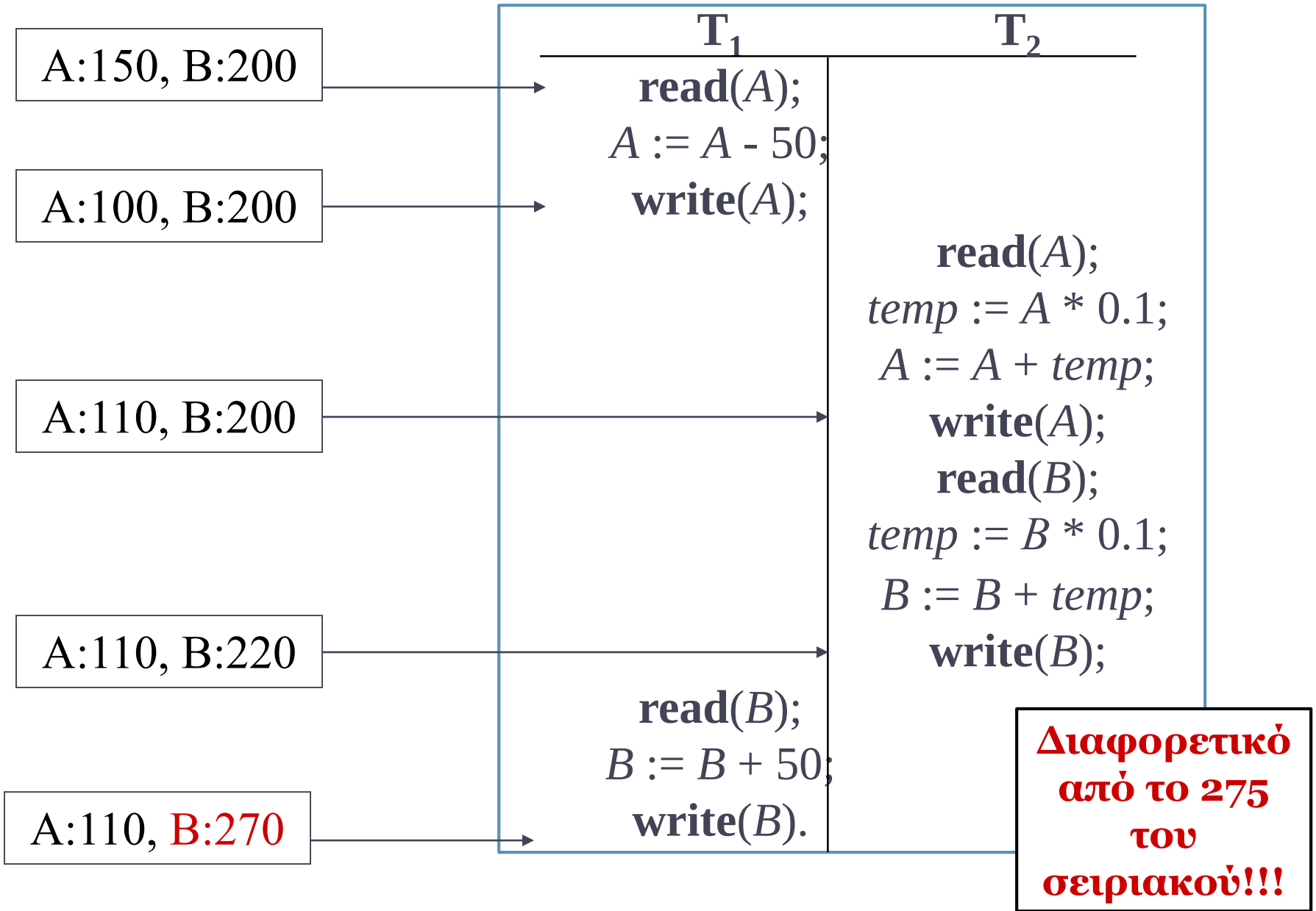
Προσοχή: Δεν υφίσταται περιορισμός για το Α+Β, πλέον!

Σκοπός είναι να δείξουμε προβλήματα που μπορούν να προκύψουν…

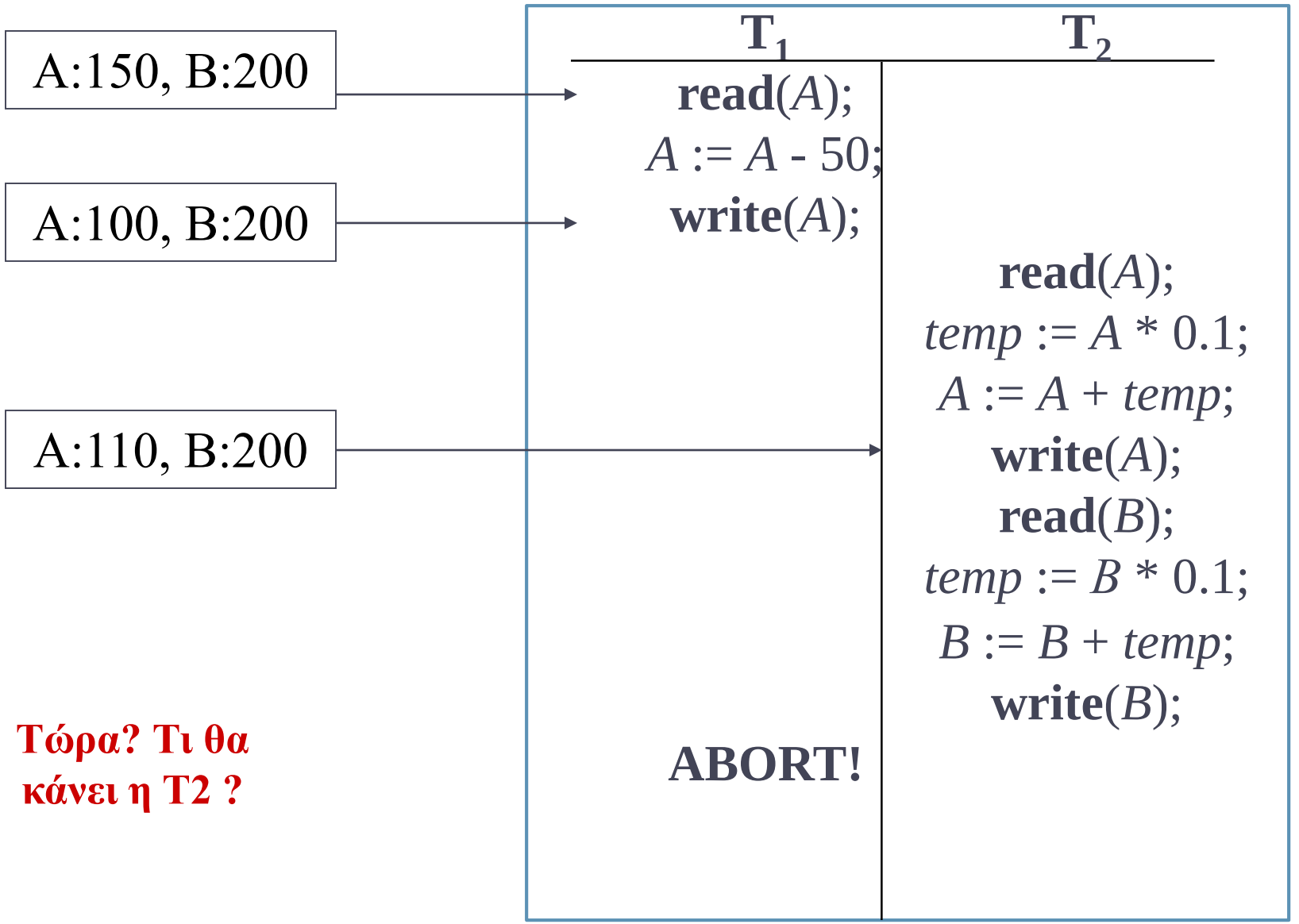
Σειριακό Χρονοπρόγραμμα



Ασυνεπής Ανάγνωση (Dirty Read)



Ασυνεπής Ανάγνωση (Dirty Read)

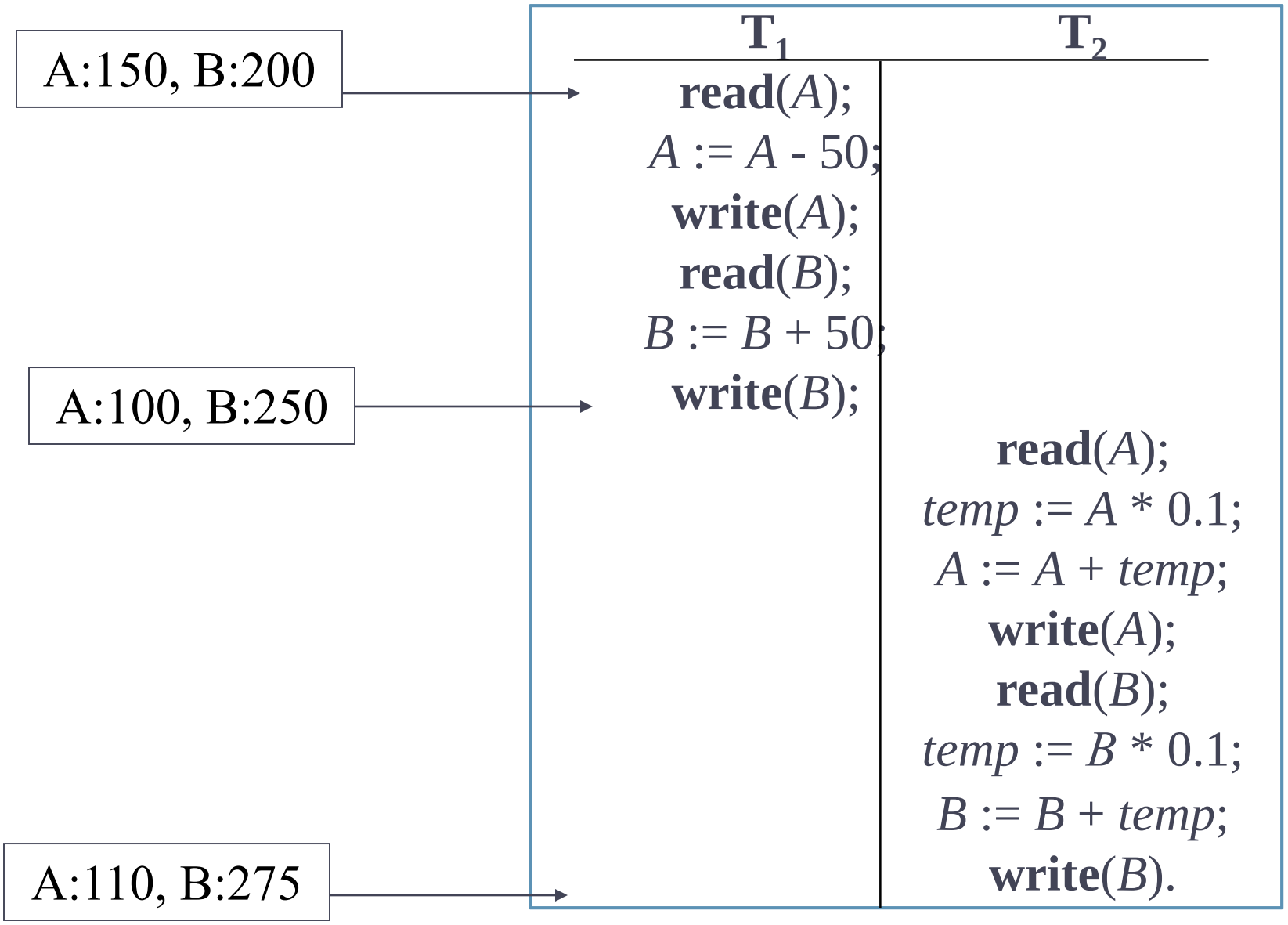


Τώρα? Τι θα κάνει η T2 ?

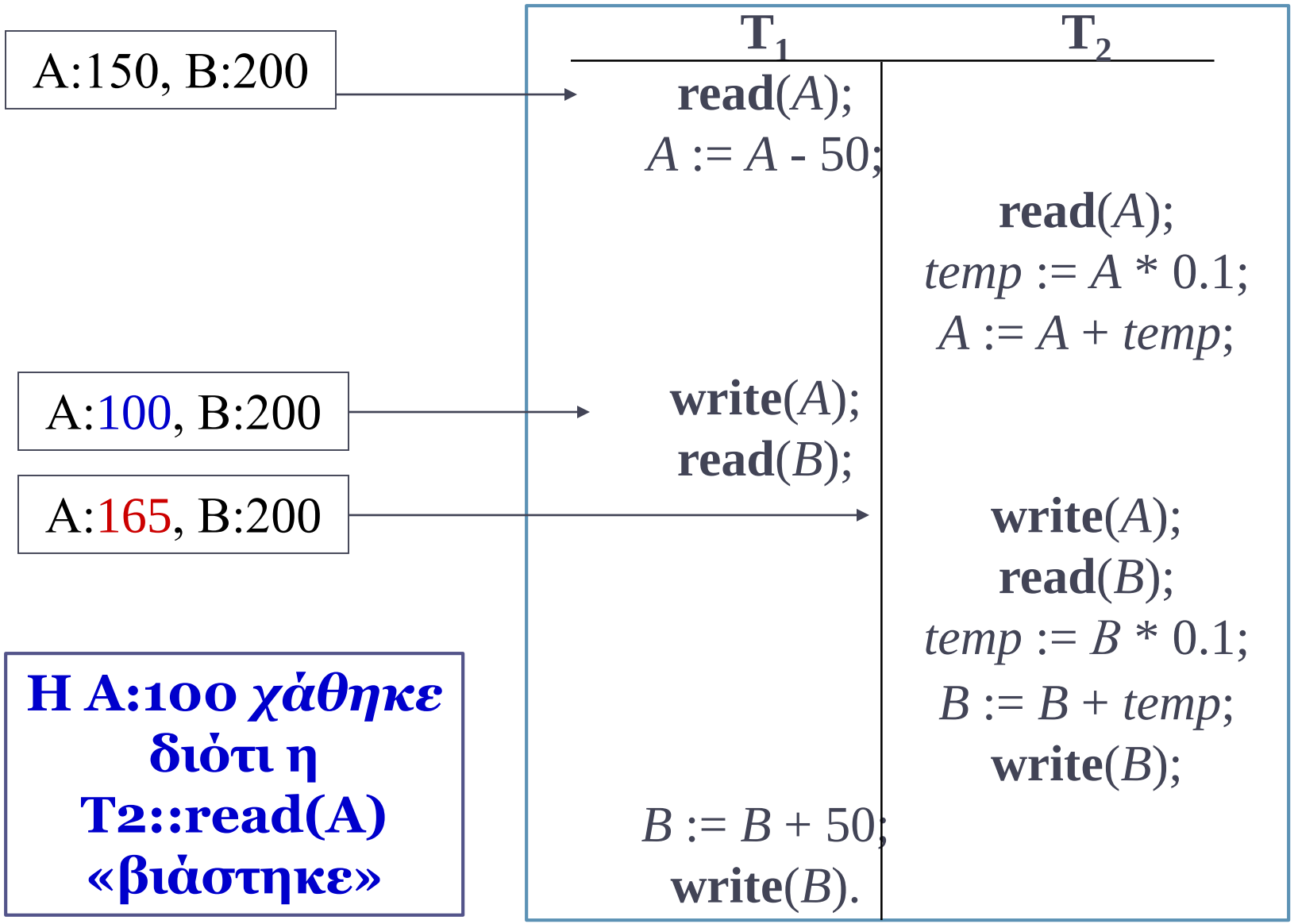
Dirty read

- Ο όρος προκύπτει από το γεγονός ότι η Τ2 διαβάζει μια τιμή για το Α, ενώ η Τ1, η οποία είχε ξεκινήσει να το τροποποιεί, δεν έχει ολοκληρώσει ακόμα.
- Κατά συνέπεια, αν η Τ1 κάνει abort, πρέπει να κάνει και η Τ2 [ασχέτως που έχει ήδη ολοκληρώσει]…

Σειριακό Χρονοπρόγραμμα



Απώλεια Ενημερώσεων (lost updates)



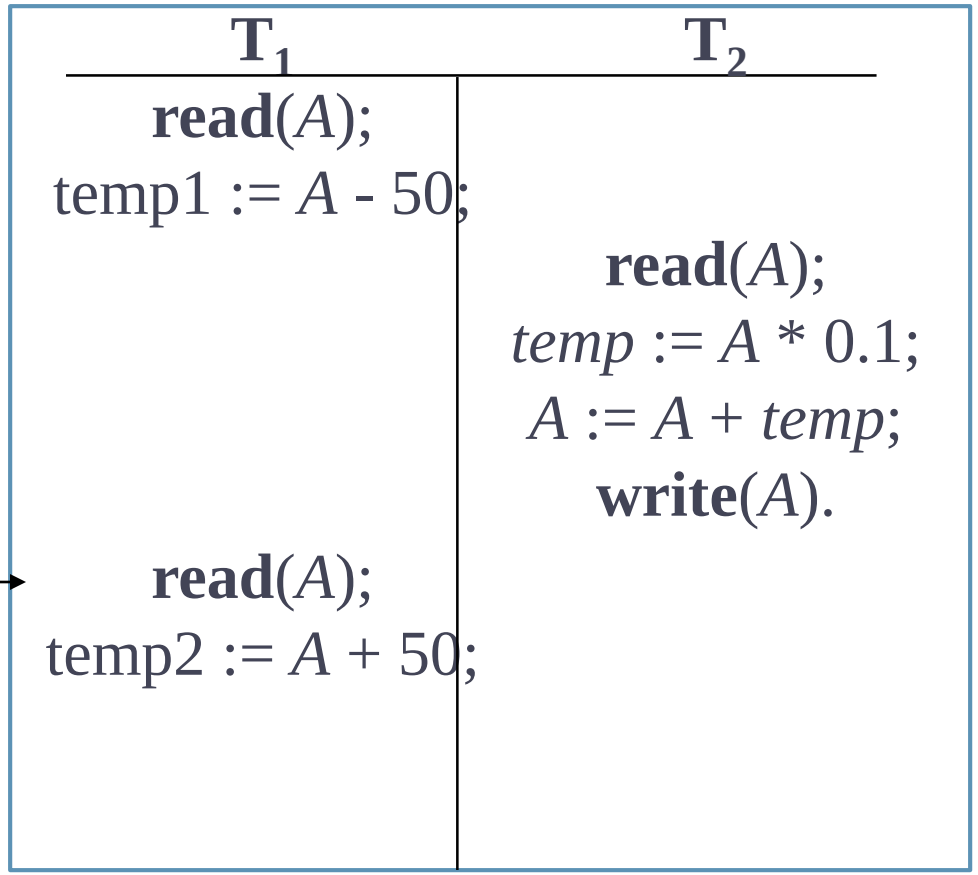
Μη επαναλήψιμες αναγνώσεις (Non-repeatable reads)

Ακόμα κι αν δεν έχει νόημα να διαβάσει 2 φορές το Α, το Τ1 είναι ένα έγκυρο transaction

T_1	T_2
read(*A*);	
temp1 := *A* - 50;	
read(*A*);	
temp2 := *A* + 50;	
	read(*A*);
	temp := *A* * 0.1;
	A := *A* + *temp*;
	write(*A*).

Μη επαναλήψιμες αναγνώσεις (Non-repeatable reads)

Στην ίδια συναλλαγή, διαβάσαμε 2 φορές το Α και πήραμε διαφορετική τιμή!!



Πλήρες χρονοπρόγραμμα

- **Πλήρες χρονοπρόγραμμα**: ένα χρονοπρόγραμμα που συμπεριλαμβάνει **abort** ή **commit** στο τέλος της κάθε συναλλαγής.

Επί της ουσίας …

- Ωραία, και πώς εγγυόμαστε ότι ένα χρονοπρόγραμμα δεν παραβιάζει τη συνέπεια της βάσης?
  - Σειριοποιησιμότητα …

Σειριοποιησιμότητα

- Βασικό ζητούμενο είναι η συνέπεια της βάσης
- Η σειριακή εκτέλεση των συναλλαγών (σειριακό χρονοπρόγραμμα) εγγυάται τη συνέπεια
- **Σειριοποιήσιμο χρονοπρόγραμμα**: ένα χρονοπρόγραμμα που εγγυημένα έχει το ίδιο αποτέλεσμα με ένα πλήρες σειριακό χρονοπρόγραμμα.

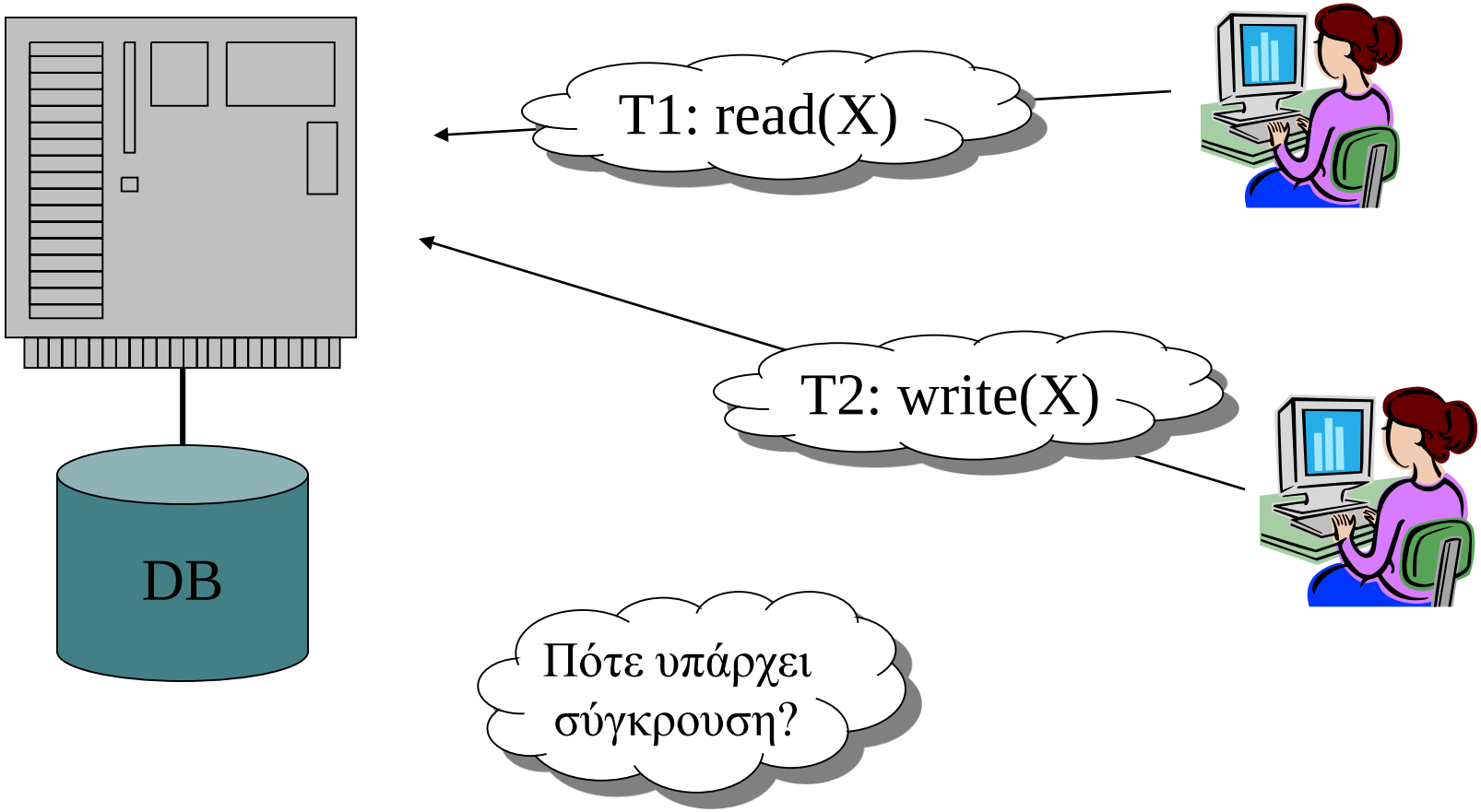
Απομόνωση: θυμάστε τι πρεσβεύει η έννοια ?

Σειριοποιησιμότητα

- Κατά συνέπεια, αν μας δοθεί ένα χρονοπρόγραμμα, πρέπει να μπορέσουμε να αποφανθούμε αν είναι σειριοποιήσιμο ή όχι!
- Δύο τεχνικές:
  - Σειριοποιησιμότητα συγκρούσεων
  - Σειριοποιησιμότητα όψεως

- **Στο εξής, θα αγνοούμε το processing στη μνήμη και θα μας απασχολούν μόνο οι read και write αλληλεπιδράσεις των συναλλαγών με τη βάση δεδομένων!!**

Σειριοποιησιμότητα συγκρούσεων



Συγκρούσεις

- Έστω δύο συναλλαγές, T1 και T2, οι οποίες θέλουν να ενεργήσουν μέσα στο ίδιο χρονοπρόγραμμα πάνω στο ίδιο αντικείμενο Α. Πότε θα το επιτρέψουμε? [Εναλλακτικά:, πότε συγκρούονται οι ενέργειές τους?]

T1\T2	Read	Write
Read	☑	☒
Write	☒	☒

Συγκρούσεις -Αντίθετες Πράξεις

- Οι πράξεις l_i και l_j των δοσοληψιών T_i και T_j αντίστοιχα είναι **αντίθετες** αν και μόνο αν υπάρχει ένα στοιχείο Q που προσπελαύνεται από τις l_i και l_j, και τουλάχιστον μία από τις δύο έκανε εγγραφή του Q.
  1. l_i = **read**(Q), l_j = **read**(Q). δεν είναι αντίθετες
  2. l_i = **read**(Q), l_j = **write**(Q). είναι αντίθετες
  3. l_i = **write**(Q), l_j = **read**(Q). είναι αντίθετες
  4. l_i = **write**(Q), l_j = **write**(Q). είναι αντίθετες
- Διαισθητικά, οι πράξεις l_i και l_j είναι αντίθετες όταν υπάρχει μία λογική χρονική σειρά στην εκτέλεσή τους.
  - Αν οι l_i και l_j ήταν συνεχόμενες σε ένα χρονικό και δεν ήταν αντίθετες, τότε το αποτέλεσμα θα ήταν το ίδιο ακόμα και αν είχαμε αλλάξει τη σειρά τους στο χρονικό.

Σειριοποιησιμότητα συγκρούσεων

- Δύο χρονοπρογράμματα καλούνται **ισοδύναμα συγκρούσεων** αν για κάθε σύγκρουση, οι συγκρουόμενες πράξεις έχουν την ίδια σειρά στα δύο προγράμματα.

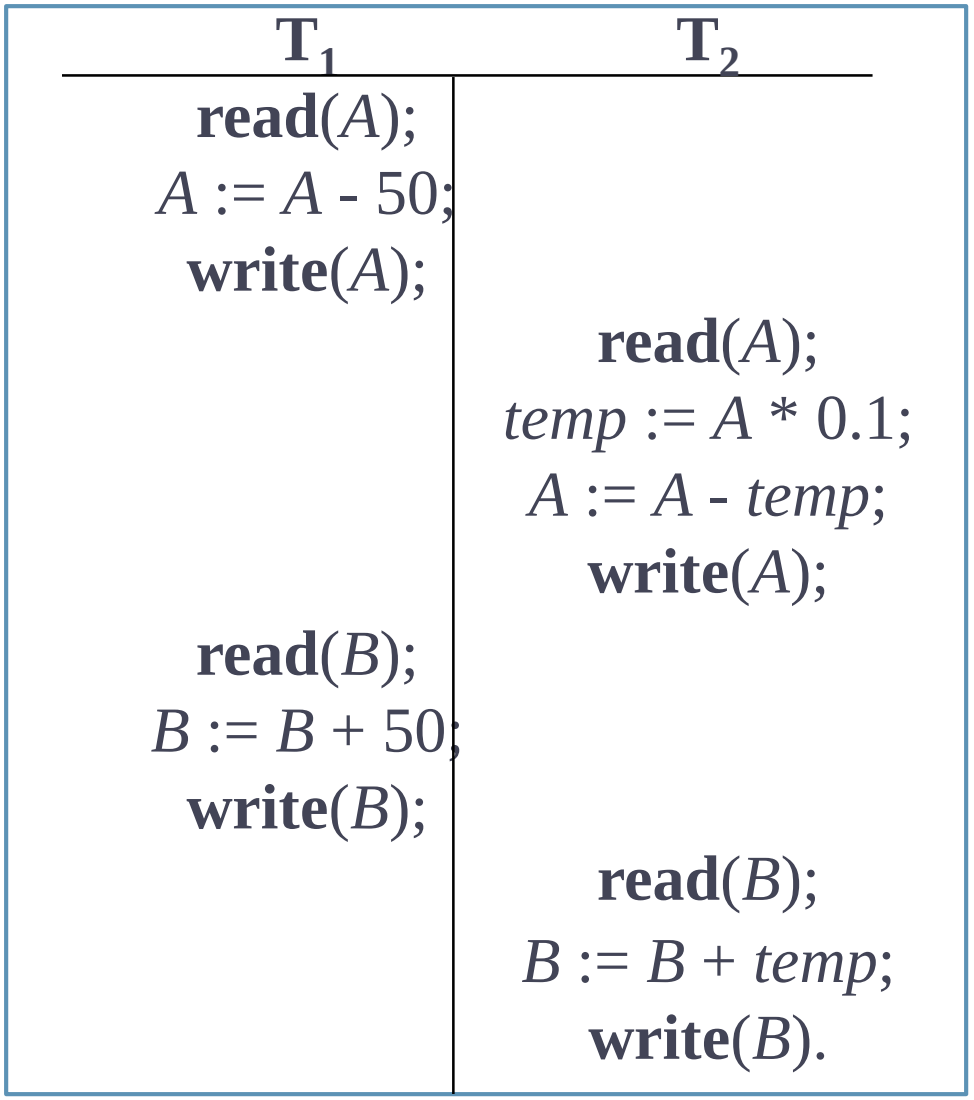
Σειριακό Χρονοπρόγραμμα S1

R1(A);
W1(A);
R1(B);
W1(B);
C1;
R2(A);
W2(A);
R2(B);
W2(B);
C2

T_1	T_2
read(A);	
$A := A - 50$;	
write(A);	
read(B);	
$B := B + 50$;	
write(B);	
	read(A);
	$temp := A * 0.1$;
	$A := A - temp$;
	write(A);
	read(B);
	$B := B + temp$;
	write(B).

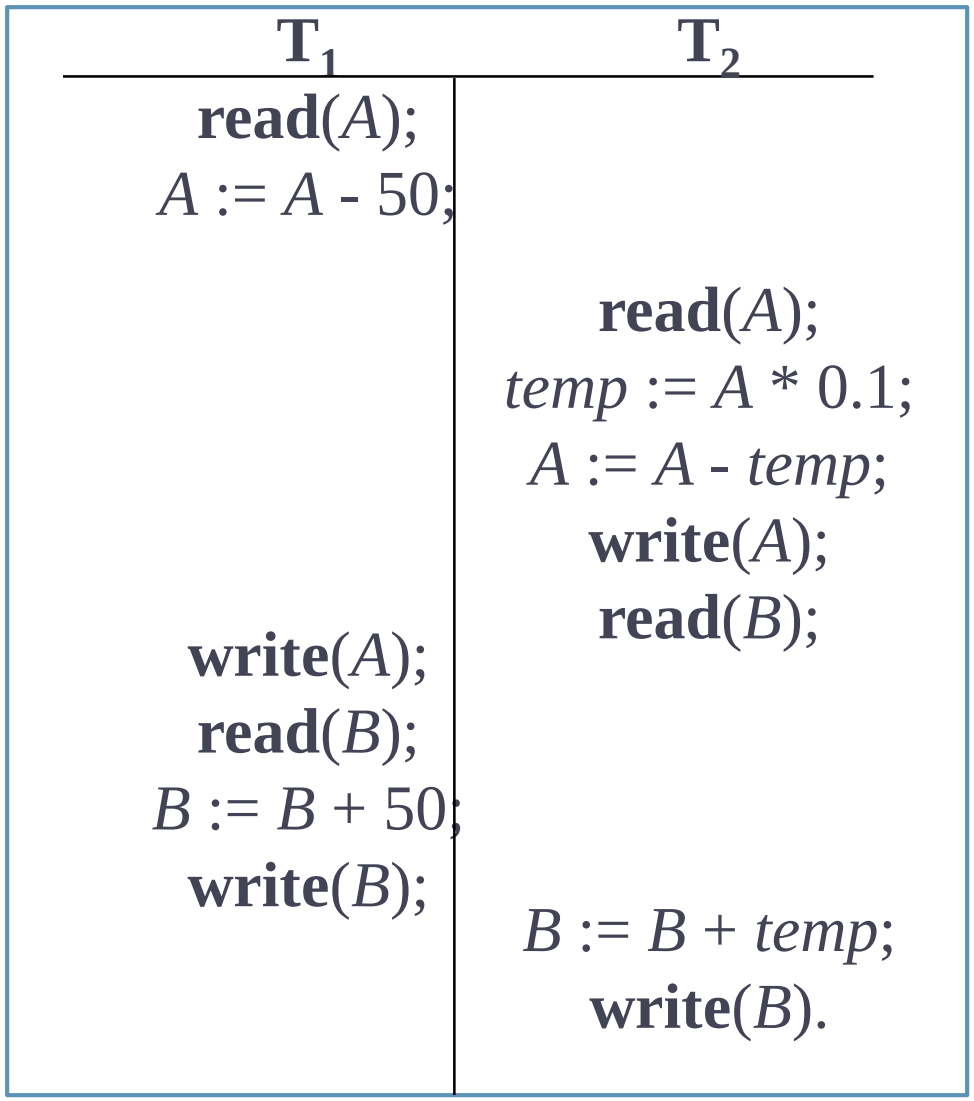
Μη Σειριακό Χρονοπρόγραμμα S2

R1(A);
W1(A);
R2(A);
W2(A);
R1(B);
W1(B);
C1;
R2(B);
W2(B);
C2



Προβληματικό Χρονοπρόγραμμα S3

R1(A);
R2(A);
W2(A);
R2(B);
W1(A);
R1(B);
W1(B);
C1;
W2(B);
C2

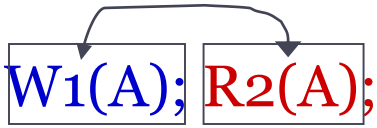


Ισοδυναμία ?

- S1:

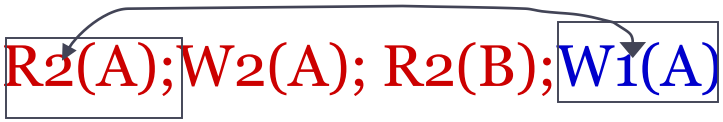
R1(A);W1(A);R1(B);W1(B);C1;R2(A);W2(A);R2(B);W2(B);C2

- S2: Διατήρηση της σειράς



R1(A);W1(A); R2(A);W2(A);R1(B);W1(B);C1;R2(B);W2(B);C2

- S3: η σειρά άλλαξε!



R1(A); R2(A);W2(A); R2(B);W1(A); R1(B);W1(B);C1;W2(B);C2

Σειριοποιησιμότητα [τυπικά]

- Ένα χρονοπρόγραμμα είναι **σειριοποιήσιμο συγκρούσεων** αν είναι ισοδύναμο συγκρούσεων με ένα σειριακό.

- ... αν, δηλαδή, όλες οι συγκρουόμενες πράξεις έχουν την ίδια σειρά που θα είχαν σε ένα σειριακό...

Διαισθητικά…

- Στο σειριακό χρονοπρόγραμμα, κάθε συναλλαγή «ξεμπερδεύει» ξεχωριστά (σε απομόνωση) με κάθε αντικείμενο και μετά το «αναλαμβάνει» μια άλλη…

- Σε ένα σειριοποιήσιμο, το να ΜΗΝ υπάρχει σύγκρουση σημαίνει ότι το χρονοπρόγραμμα «ξεμπερδεύει» με τα αντικείμενα με την ίδια σειρά ανά συναλλαγή, με την οποία θα το έκανε και το σειριακό…

Ερώτηση

T_1	T_2
read(A);	
write(A);	
	read(B);
	write(B);
read(B);	
write(B).	
	read(A);
	write(A).

Είναι σειριοποιήσιμο ή όχι?

Γράφος Σειριοποιησιμότητας

- Μοντελοποιούμε τις συγκρούσεις ενός χρονοπρογράμματος με ένα γράφο, ο οποίος έχει:
  - Για κάθε συναλλαγή και ένα κόμβο
  - Μια κατευθυνόμενη ακμή από την συναλλαγή T_i στην συναλλαγή T_j, αν μια ενέργεια της T_i συγκρούεται με μια επακόλουθή της, της T_j.

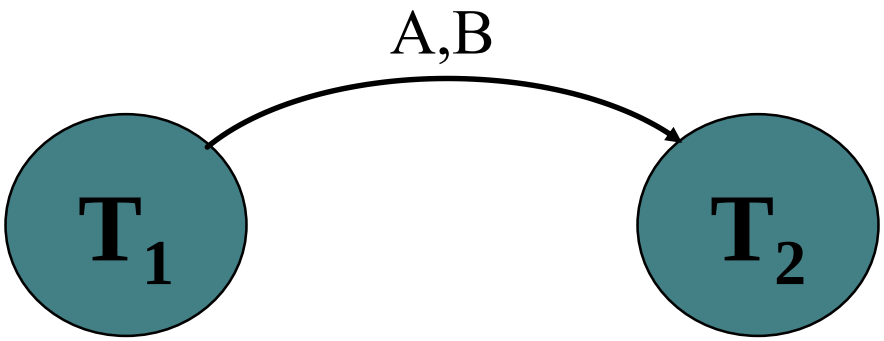
Ήτοι, για κάθε σύγκρουση $Πράξη_i(X);Πράξη_j(X)$ μια ακμή από τον προηγούμενο κόμβο i στον επόμενο κόμβο j

Γράφος σειριοποιησιμότητας

- Για ευκολία, μπορούμε να σημειώνουμε και το αντικείμενο για το οποίο οι δύο συναλλαγές συγκρούονται ...

Γράφος σειριοποιησιμότητας

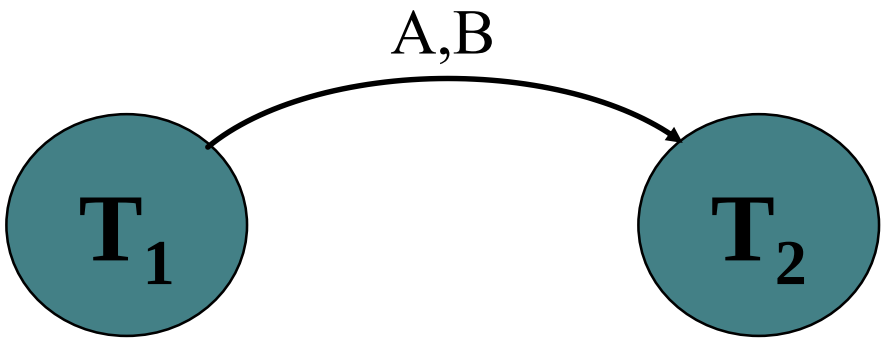
- S1:
  R1(A);W1(A);R1(B);W1(B);C1;R2(A);W2(A);R2(B);W2(B);C2



Θυμίζω: ακμή από τον προηγούμενο κόμβο i στον επόμενο κόμβο j

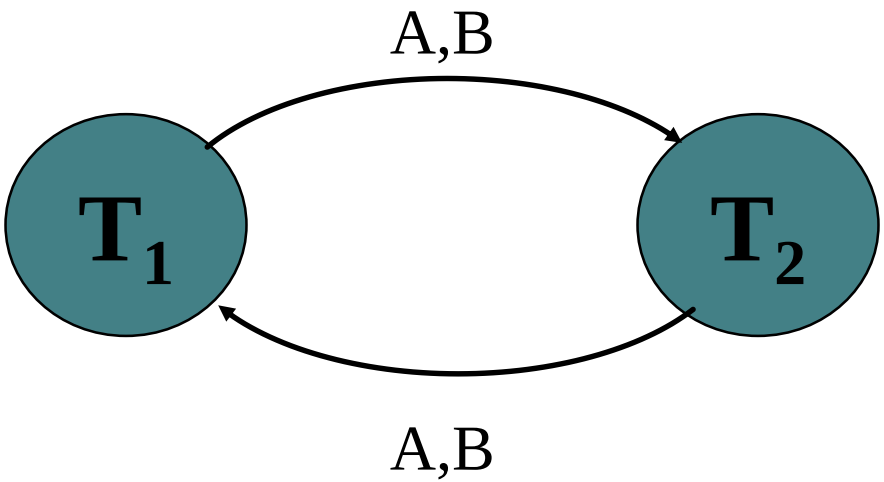
Γράφος σειριοποιησιμότητας

- S2:
  R1(A);W1(A); R2(A);W2(A);R1(B);W1(B);C1;R2(B);W2(B);C2



Γράφος σειριοποιησιμότητας

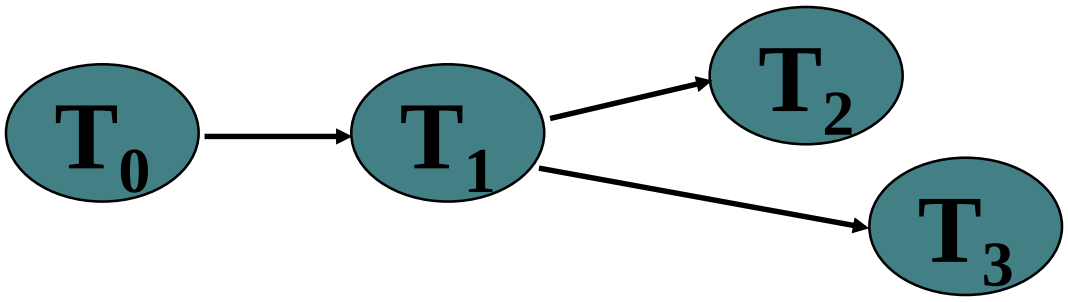
- S3:
  R1(A); R2(A);W2(A); R2(B);W1(A); R1(B);W1(B);C1;W2(B);C2

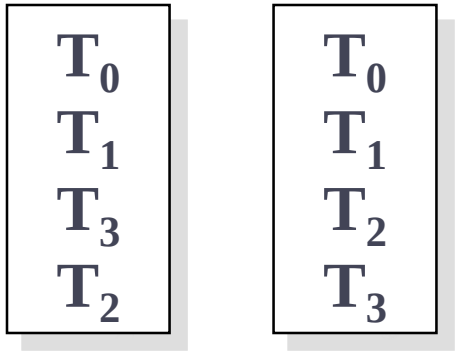


Θεώρημα

- Γράφος με κύκλο είναι μη σειριοποιήσιμος σε σχέση με τις συγκρούσεις
- Γράφος χωρίς κύκλο είναι σειριοποιήσιμος σε σχέση με τις συγκρούσεις

Το ισοδύναμο σειριακό πρόγραμμα προκύπτει από την τοπολογική ταξινόμηση του γράφου.





Ερώτηση

T_1	T_2
read(A);	
write(A);	
	read(B);
	write(B);
read(B);	
write(B).	
	read(A);
	write(A).

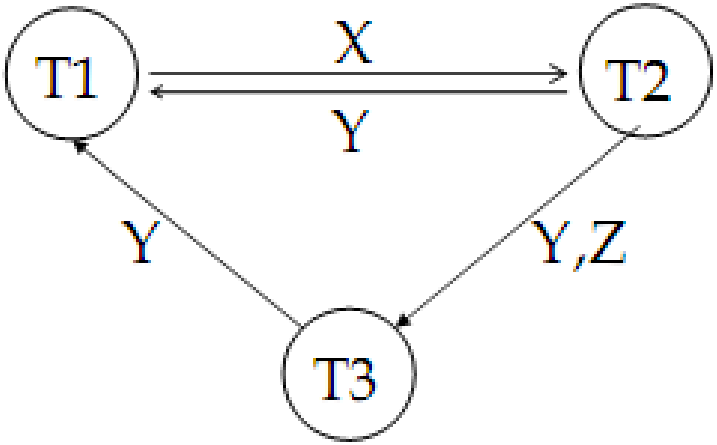
Είναι σειριοποιήσιμο ή όχι?

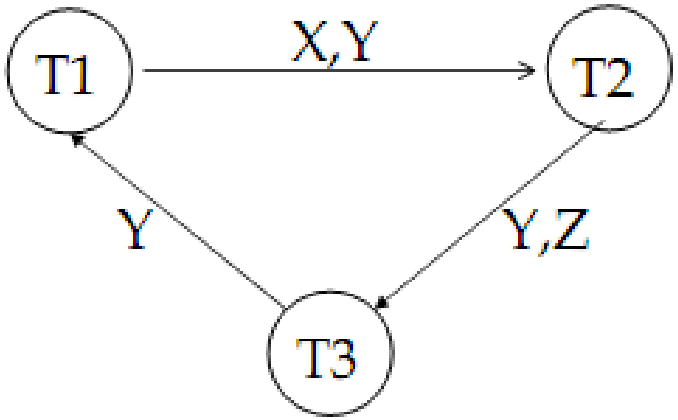
Ερώτηση

Είναι σειριοποιήσιμα ή όχι?

S1: R2(Z)R2(Y)W2(Y)R3(Y)R3(Z)R1(X)W1(X)W3(Y)W3(Z)C3R2(X)R1(Y)W1(Y)C1W2(X)C2

S2: R3(Y)R3(Z)R1(X)W1(X)W3(Y)W3(Z)C3R2(Z)R1(Y)W1(Y)C1R2(Y)W2(Y)R2(X)W2(X)C2



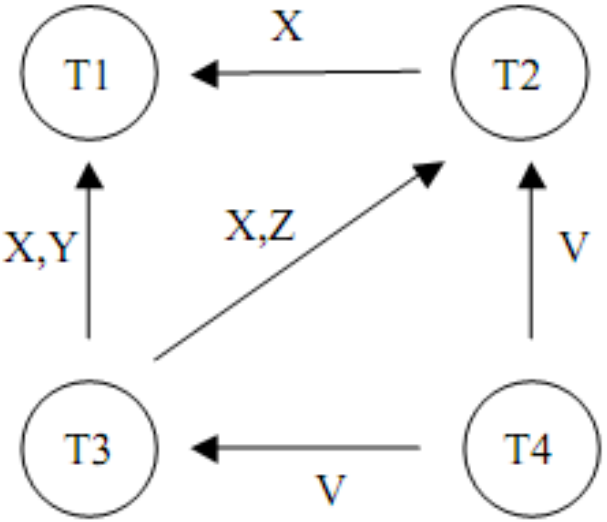


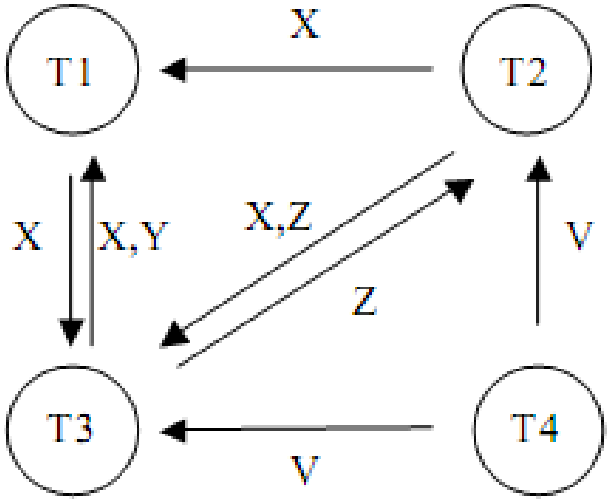
Ερώτηση

Είναι σειριοποιήσιμα ή όχι?

S1 R4(V) W3(X) R2(X) R1(X) W4(V) W3(Y) R1(Y) W1(X) R3(V) W3(Z) R2(Z) W1(Y) R2(V) W2(Z)

S2 R1(X) R2(X) R2(Z) W3(X) W3(Y) R4(V) R1(Y) W4(V) W1(X) R2(V) R3(V) W3(Z) W2(Z) W1(Y)





Θεώρημα

Σειριοποιησιμότητα συγκρούσεων $\Leftrightarrow$
έλλειψη κύκλου στο γράφο

- Υπάρχει όμως και άλλη εκδοχή σειριοποιησιμότητας, η σειριοποιησιμότητα όψεως…

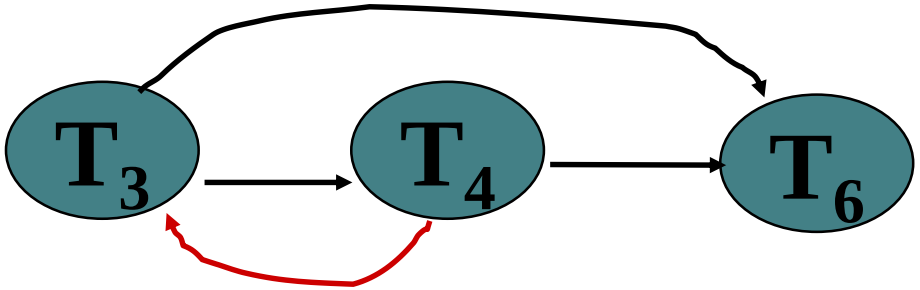
Σειριοποιησιμότητα όψεως

- Ποιος ο γράφος σειριοποίησης του παρακάτω χρονοπρογράμματος?

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Παραδείγματα από Siberschatz, Korth & Sudarsan

Σειριοποιησιμότητα όψεως



T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Σειριοποιησιμότητα όψεως

- Και λοιπόν? Αφού ούτως ή άλλως, σημασία έχει τι γράφει η T6...

T_3	T_4	T_6
read(Q)		
	write(Q)	
write(Q)		
		write(Q)

Ισοδυναμία Όψεων

- Δυο χρονοπρογράμματα S1 και S2 είναι ισοδύναμα όψεων αν:
  - Τα S1 και S2 περιέχουν το ίδιο σύνολο δοσοληψιών και τις ίδιες πράξεις των συγκεκριμένων δοσοληψιών
  - Για κάθε στοιχείο X, αν η Ti διαβάζει την αρχική του τιμή στο S1, πρέπει να διαβάζει επίσης την αρχική τιμή στο S2
  - Για κάθε πράξη Ri(X) στο S1 μετά από μια πράξη Wj(X) πρέπει να ισχύει η ίδια συνθήκη και στο S2.
    - Οι πράξεις ανάγνωσης βλέπουν την ίδια όψη
  - Αν η πράξη Wk(Y) στο S1 είναι η τελευταία πράξη που τροποποιεί το Y τότε πρέπει να ισχύει η ίδια συνθήκη και στο S2.
    - Η τελική κατάσταση της βάσης θα είναι η ίδια

Ισοδυναμία Όψεων-Διαισθητικά…

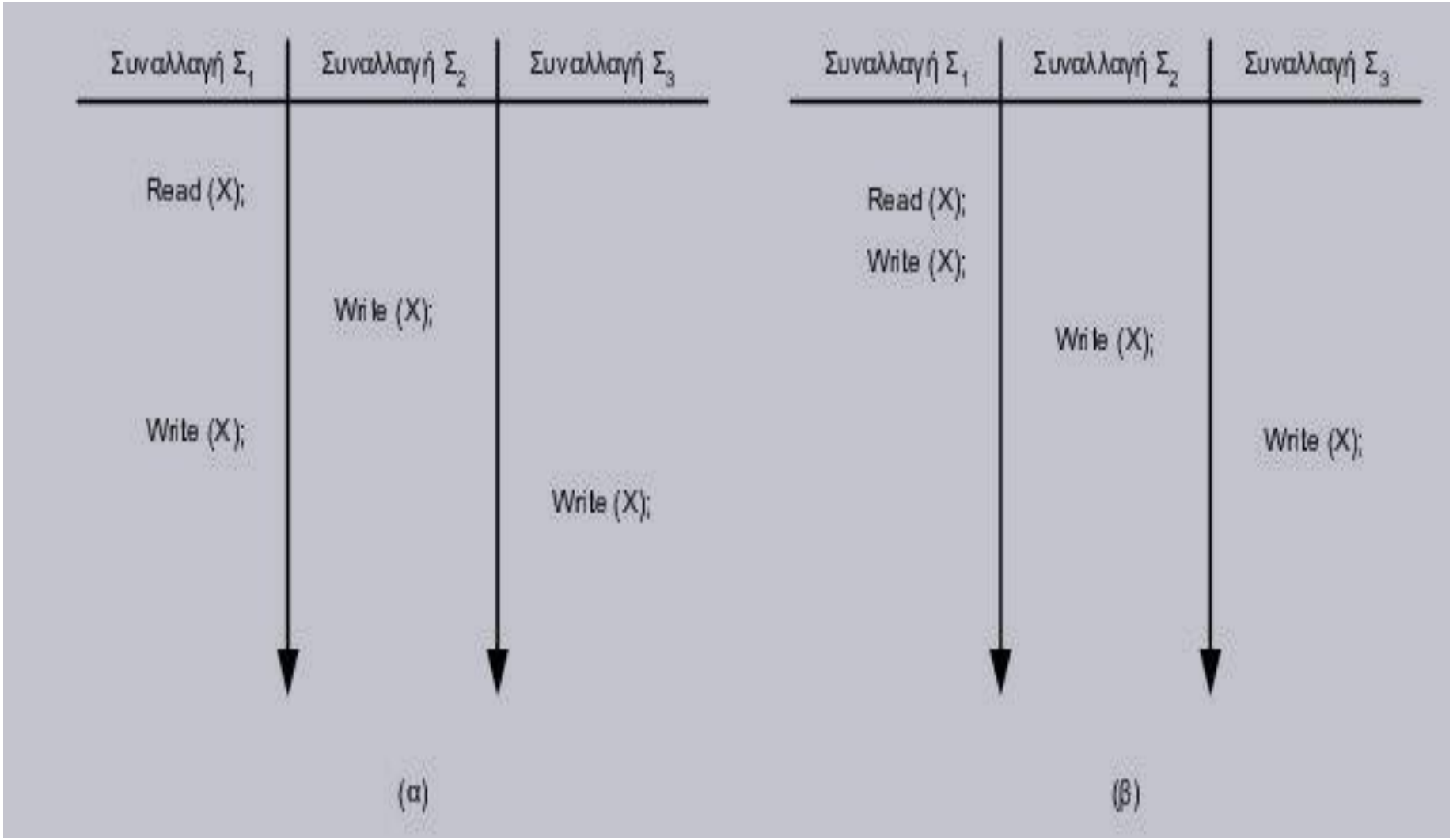
- Κάθε πράξη ανάγνωσης διαβάζει το αποτέλεσμα της ίδιας πράξης εγγραφής και στα δύο χρονοπρογράμματα.
- Οι πράξεις ανάγνωσης βλέπουν την ίδια ***όψη*** και στα δύο χρονοπρογράμματα

Ισοδυναμία Όψεων-Παράδειγμα1

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

T_1	T_2
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

Ισοδυναμία Όψεων-Παράδειγμα2



Ισοδυναμία Όψεων

- Υπόθεση εγγραφής υπό περιορισμό
  - Πριν κάθε πράξη εγγραφής εκτελείται μια πράξη ανάγνωσης της ίδιας δοσοληψίας και
  - Η τιμή που καταχωρείται εξαρτάται μόνο από την πράξη ανάγνωσης που προηγήθηκε
    - Οι ορισμοί σειριοποιησιμότητας όψεων και αντιθέσεων είναι παρόμοιοι

Σειριοποιησιμότητα συγκρούσεων και όψεων

- Κάθε χρονοπρόγραμμα που είναι σειριοποιήσιμο σε σχέση με συγκρούσεις, είναι σειριοποιήσιμο όψεως.
- Το αντίστροφο ΔΕΝ ισχύει.
- *Κάθε χρονοπρόγραμμα που είναι σειριοποιήσιμο όψεως, και ΔΕΝ είναι σειριοποιήσιμο σε σχέση με συγκρούσεις, περιέχει* ***τυφλές εγγραφές*** *(writes που δεν έχει προηγηθεί read γι' αυτές στην συναλλαγή τους)*

Ελέγχοντας την σειριοποιησιμότητα όψεως

- Ο αλγόριθμος είναι NP-complete και κατά συνέπεια, όχι πρακτικός

Παρόλα αυτά…

Πολύ-γράφος για view serializability

1. Για κάθε συναλλαγή ένας κόμβος και επίσης δυο υποθετικές συναλλαγές T_o,T_f
2. Για κάθε Ri(X) με προέλευση Tj, μια ακμή από Tj→Ti.
3. Αν η Tj είναι η προέλευση της Ri(X) και Tk μια άλλη συναλλαγή που γράφει το X, τότε
   - Δεν μπορεί η Tk να είναι μεταξύ Tj→Ti και πρέπει να γίνει:
     - Tk→Tj ή Ti→Tk (τα βάζουμε με διακεκομμένη και τελικά επιλέγουμε όποιο δεν προκαλεί κύκλο)
   - Ειδικές περιπτώσεις:
     - Αν η Tj=To, δεν μπορεί να προηγείται η Tk, άρα επιλέγουμε μόνο το Ti→Tk
     - Αν η Ti=Tf, δεν μπορεί να ακολουθεί η Tk, άρα επιλέγουμε μόνο το Tk→Tj.

- Υποψήφιες Tk
  - Αυτές που γράφουν το X που εμπλέκεται σε μια ακμή Tj→Ti
  - Όχι οι To ή Tf
  - Όχι οι Ti,Tj.

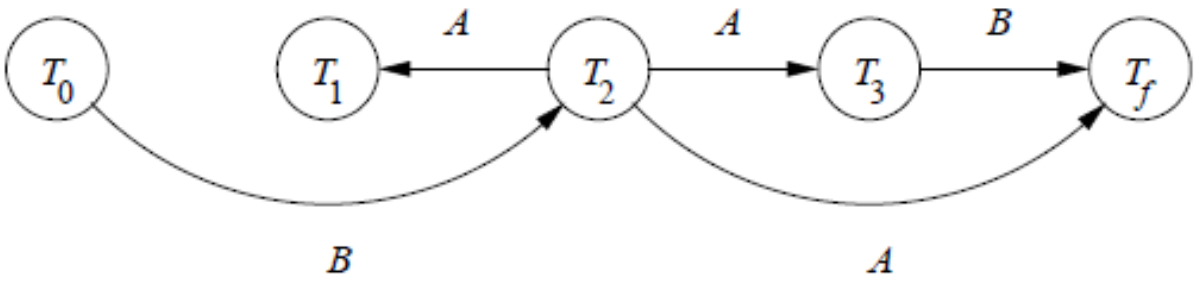
Παράδειγμα:

T_1: $r_1(A)$ $w_1(B)$

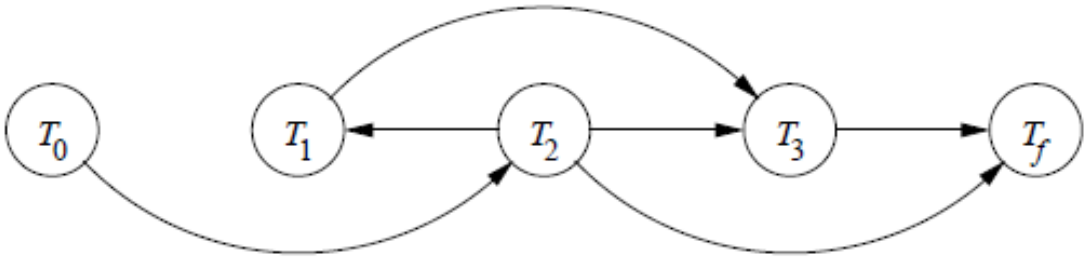
T_2: $r_2(B)$ $w_2(A)$ $w_2(B)$

T_3: $r_3(A)$ $w_3(B)$

Αρχικό
(βήμα 1&2)



Τελικό
(βήμα 3)

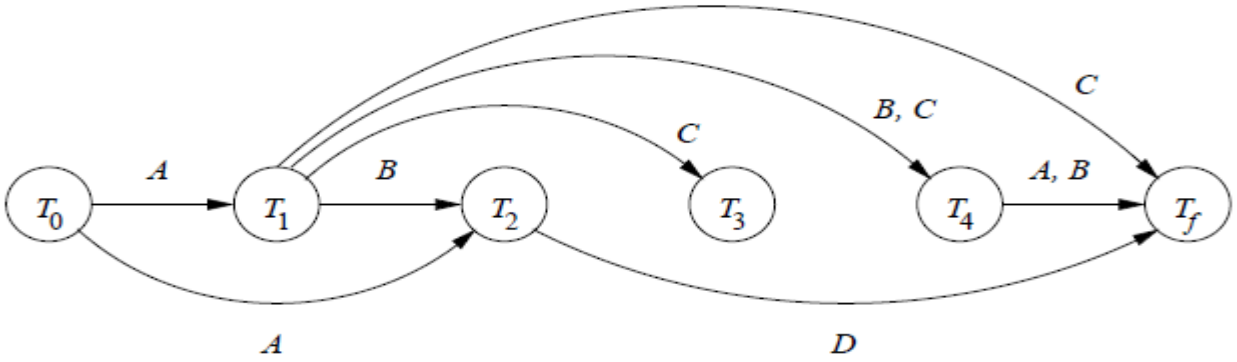


Ισοδύναμο με T2→T1→T3

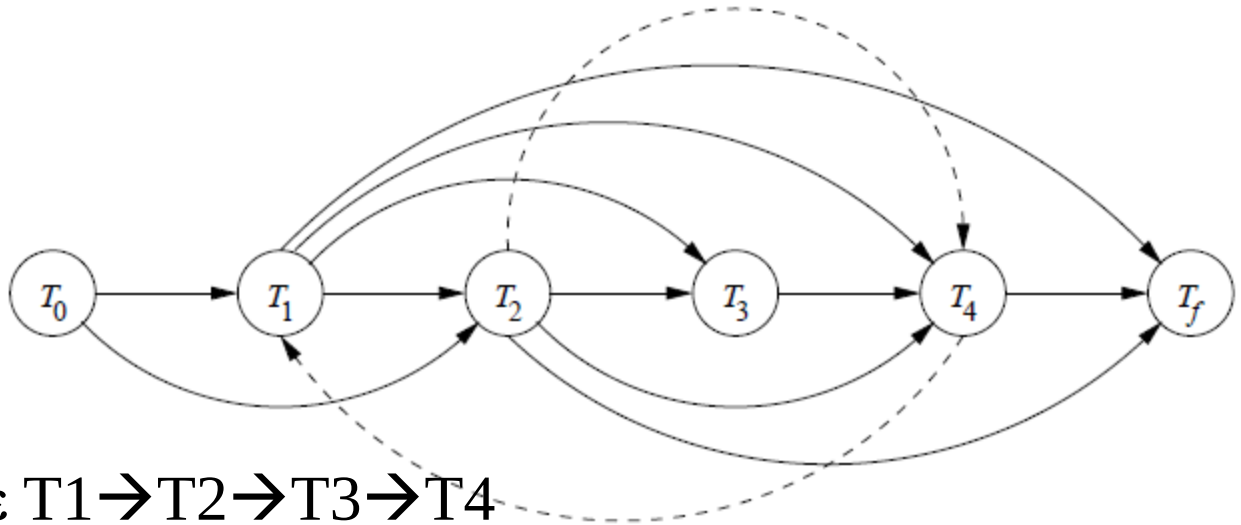
Παράδειγμα:

T_1	T_2	T_3	T_4
	$r_2(A);$		
$r_1(A);\ w_1(C);$			
		$r_3(C);$	
$w_1(B);$			
			$r_4(B);$
		$w_3(A);$	
			$r_4(C);$
	$w_2(D);\ r_2(B);$		
			$w_4(A);\ w_4(B);$

Αρχικό
(βήμα 1&2)



Τελικό
(βήμα 3)



Ισοδύναμο με T1→T2→T3→T4

Άλλες Σειριοποιήσεις (δοσοληψίες χρέωσης - πίστωσης)

- Το διπλανό χρονικό παράγει το ίδιο αποτέλεσμα με το σειριακό < T_1 , T_5 >. Παρόλα αυτά δεν είναι **ούτε ισοδύναμο αντιθέσεων** ούτε **ισοδύναμο όψης**.
- Ο καθορισμός αυτής της ισοδυναμίας προϋποθέτει ανάλυση και άλλων πράξεων εκτός της read και write.
- Στην προκειμένη περίπτωση οι πράξεις + και – μπορούν να εκτελεστούν με οποιαδήποτε σειρά.

T_1	T_5
read(A)	
$A := A - 50$	
write(A)	
	read(B)
	$B := B - 10$
	write(B)
read(B)	
$B := B + 50$	
write(B)	
	read(A)
	$A := A + 10$
	write(A)

Χρονοπρογράμματα με δυνατότητα ανάκαμψης

- Ένα χρονοπρόγραμμα S έχει δυνατότητα ανάκαμψης
  - Αν καμία δοσοληψία T στο S δεν επικυρώνεται έως ότου επικυρωθούν όλες οι δοσοληψίες T' οι οποίες τροποποίησαν ένα δεδομένο που διαβάζει η T

S1: $R_1(X)$ $W_1(X)$ $R_2(X)$ $R_1(Y)$ $W_2(X)$ C_2 A_1

Διαδιδόμενες Ανακλήσεις

- **Διαδιδόμενες Ανακλήσεις** – η αποτυχία μίας δοσοληψίας προκαλεί αλυσιδωτές ανακλήσεις δοσοληψιών. Υποθέστε ότι καμία από τις δοσοληψίες δεν έχει επικυρωθεί (επομένως είναι ανακάμψιμες)

T_{10}	T_{11}	T_{12}
read(A)		
read(B)		
write(A)		
	read(A)	
	write(A)	
		read(A)

Αν η T_{10} αποτύχει, τότε οι T_{11} και T_{12} θα πρέπει να ανακληθούν

- Οδηγεί σε αύξηση του κόστους

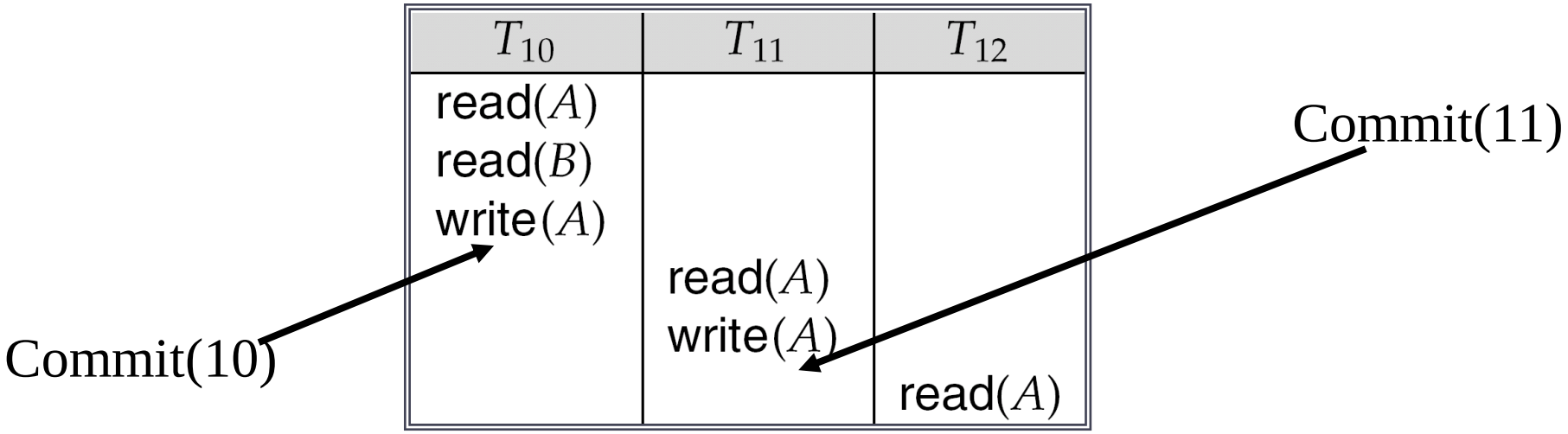
Αποφυγή διάδοσης ανακλήσεων

- Μια μη-επικυρωμένη δοσοληψία πρέπει να ανακληθεί γιατί διάβασε κάποιο στοιχείο από μια δοσοληψία που απέτυχε.

- Μια δοσοληψία αποφεύγει τη διάδοση ανακλήσεων
  - αν διαβάζει στοιχεία μόνο από επικυρωμένες δοσοληψίες

Χρονοπρογράμματα με αποφυγή διάδοσης ανάκλησης

- **Αποφυγή διάδοσης ανάκλησης** — η διάδοση ανάκλησης δεν μπορεί να συμβεί: για κάθε ζευγάρι δοσοληψιών T_i and T_j έτσι ώστε η T_j να διαβάζει ένα στοιχείο που ενημερώθηκε πριν από την T_i, η πράξη επικύρωσης της T_i εμφανίζεται πριν από την πράξη ανάγνωσης της T_j.
- Κάθε τέτοιο χρονοπρόγραμμα είναι ανακάμψιμο



Αυστηρά Χρονοπρογράμματα

Οι δοσοληψίες δεν μπορούν ούτε να διαβάσουν ούτε να γράψουν ένα στοιχείο Χ έως ότου επικυρωθεί η δοσοληψία που έγραψε το Χ