Προχωρημένα Θέματα Τεχνολογίας και Εφαρμογών Βάσεων Δεδομένων

Διαχείριση Συναλλαγών

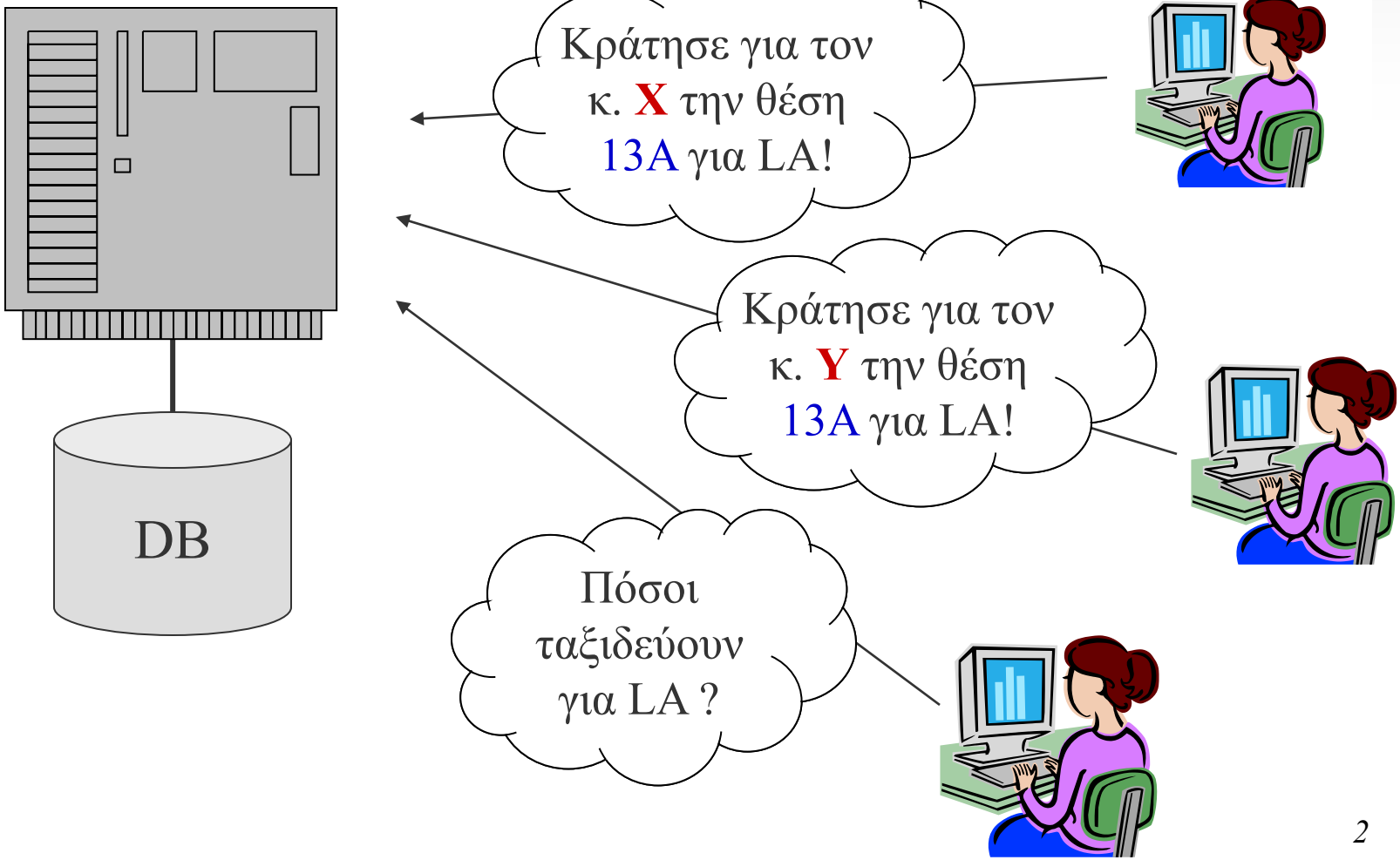
Πάνος Βασιλειάδης

pvassil@cs.uoi.gr

Μάρτιος 2021

www.cs.uoi.gr/~pvassil/courses/db_III/

Συναλλαγές & Ταυτοχρονισμός



Θεματολόγιο

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

Θεματολόγιο

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

Συναλλαγή

- **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 λεξούλες 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?

Ορολογία: Στο εξής το σύστημα δεν «πέφτει», αλλά «αποτυγχάνει» ☺

Θεματολόγιο

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

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

Διεθνώς γνωστό ως ACID test...

ACID Test

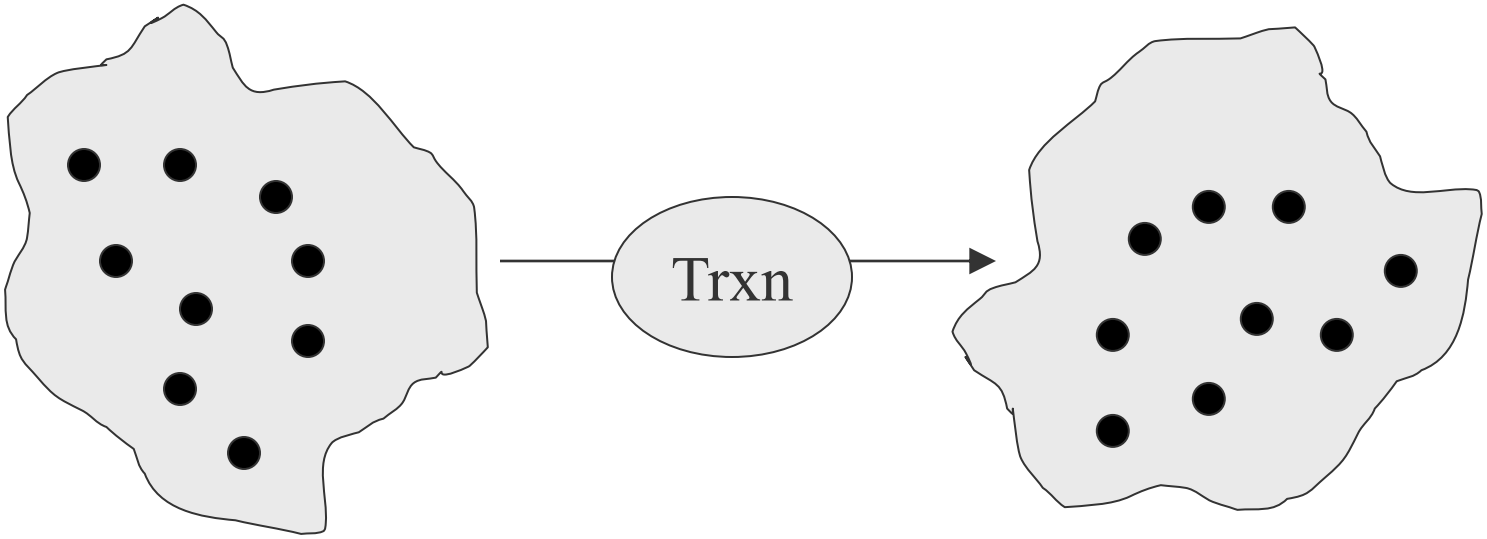
- (**A**)tomicity — Ατομικότητα
- (**C**)onsistency — Συνέπεια
- (**I**)solation — Απομόνωση
- (**D**)urability — Μονιμότητα

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...*

Συνέπεια



Συνεπής κατάσταση της βάσης πριν την συναλλαγή

Συνεπής κατάσταση της βάσης μετά την συναλλαγή

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!

ACID Test

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

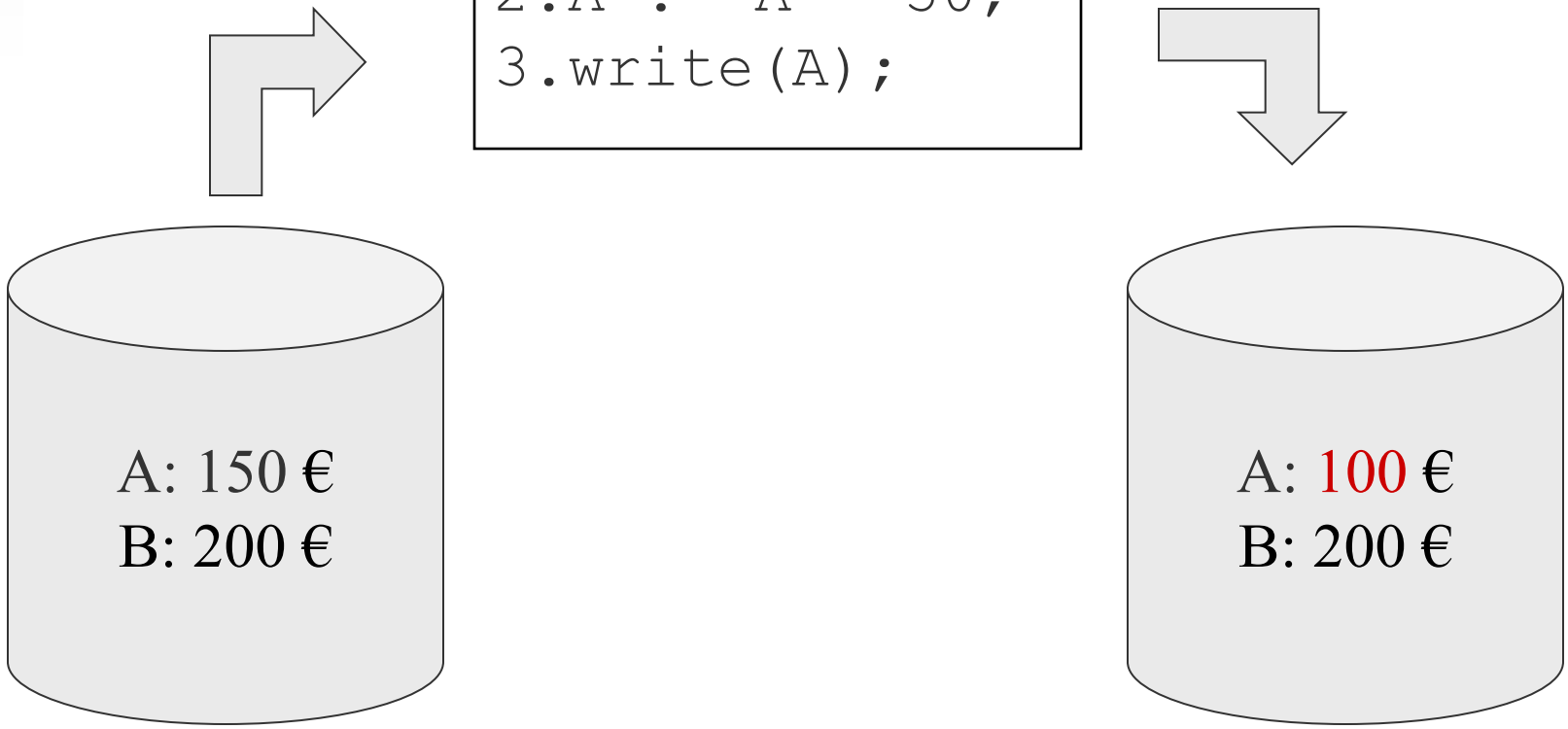
Ατομικότητα

- 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)...

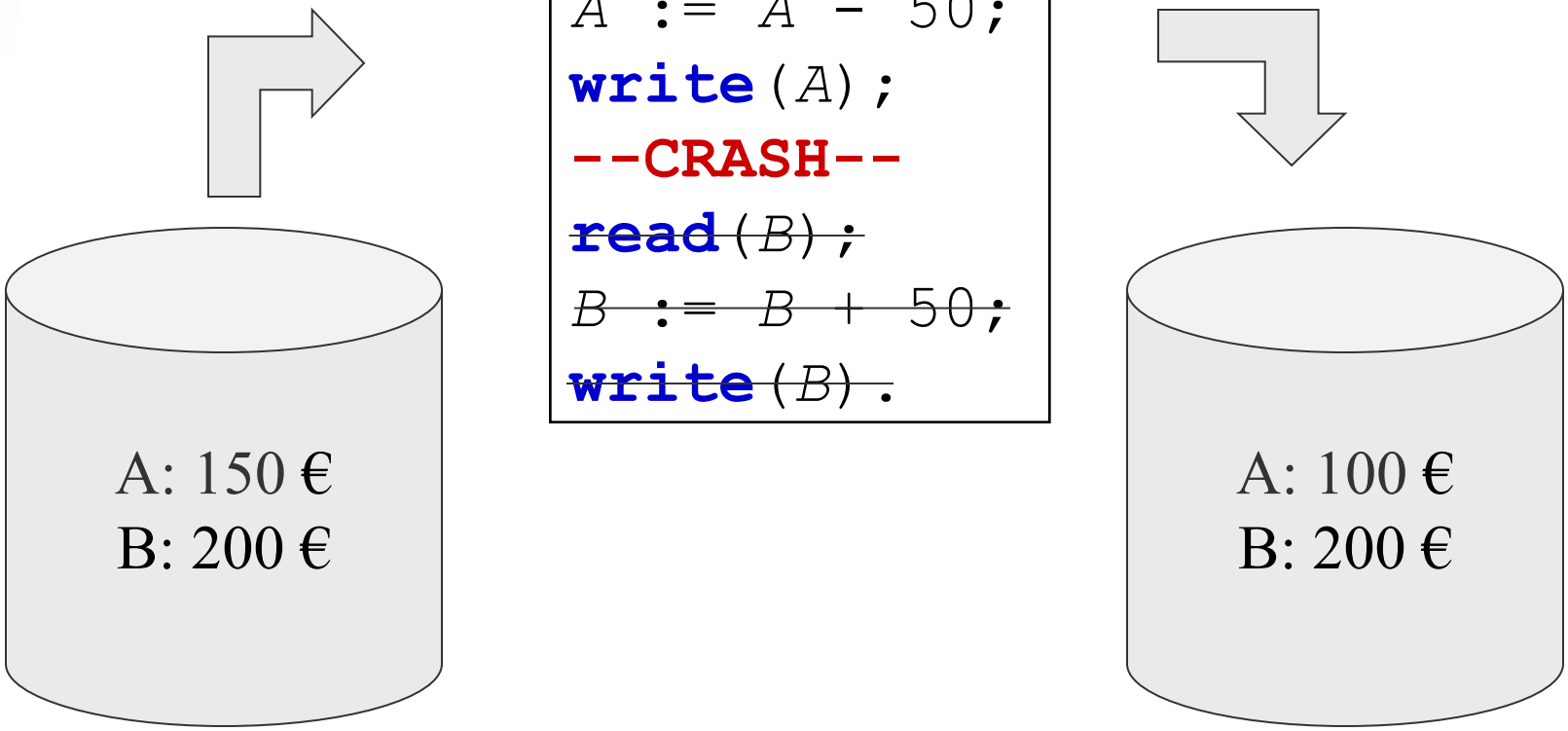
Ατομικότητα

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



Ατομικότητα

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



A+B=350 ⟷ A+B=300

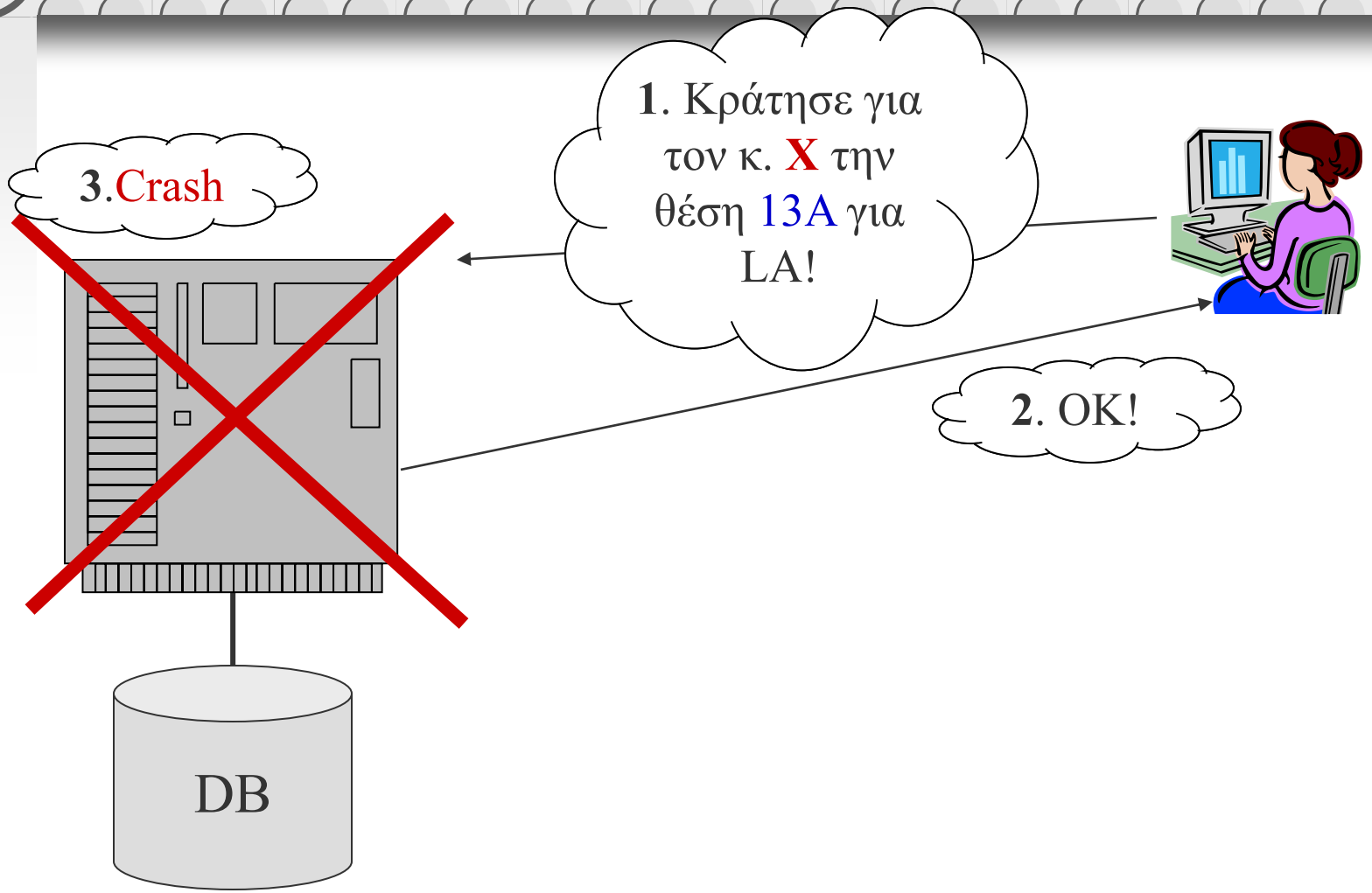
Ατομικότητα

- 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...

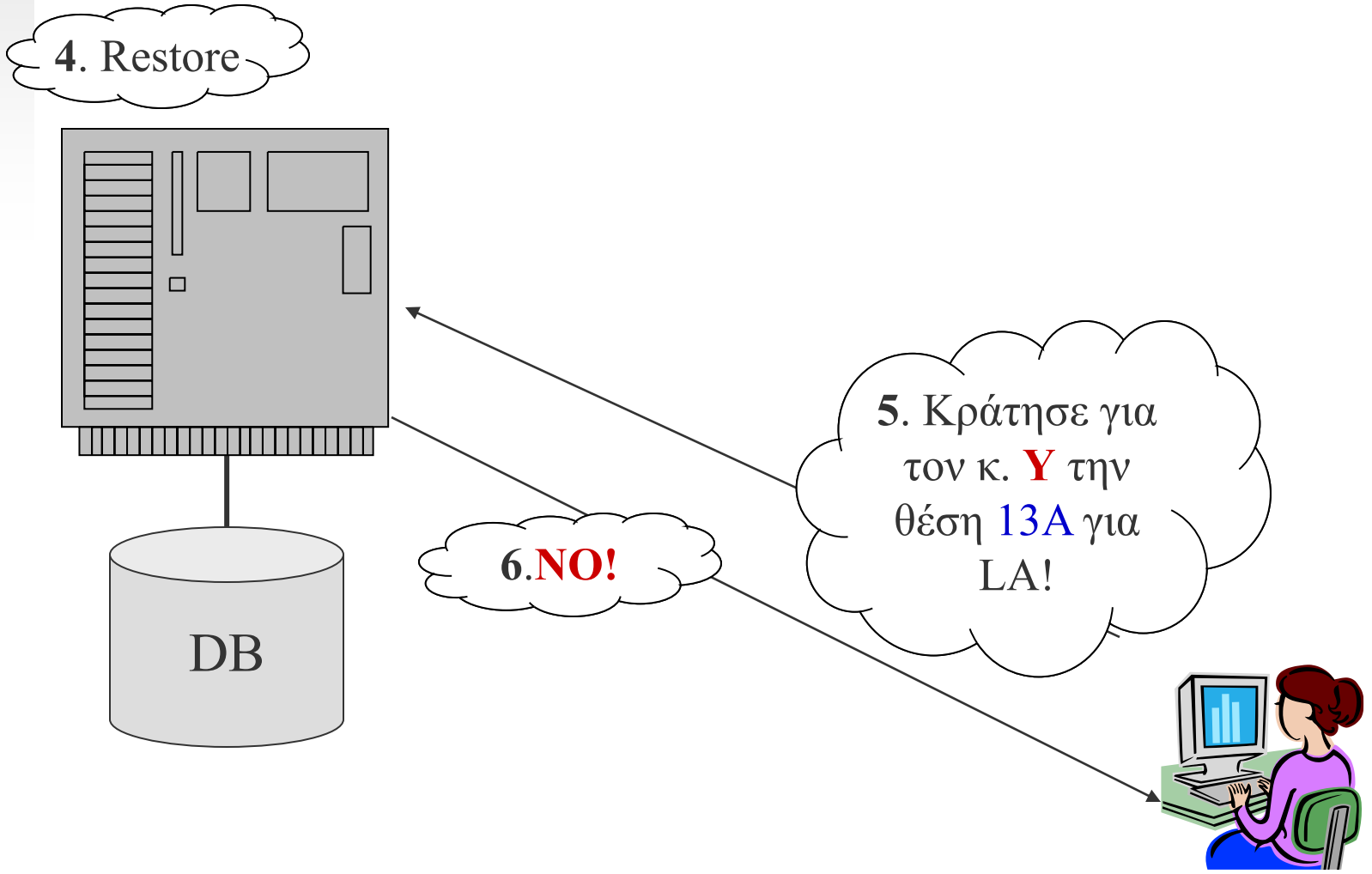
ACID Test

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

Μονιμότητα



Μονιμότητα



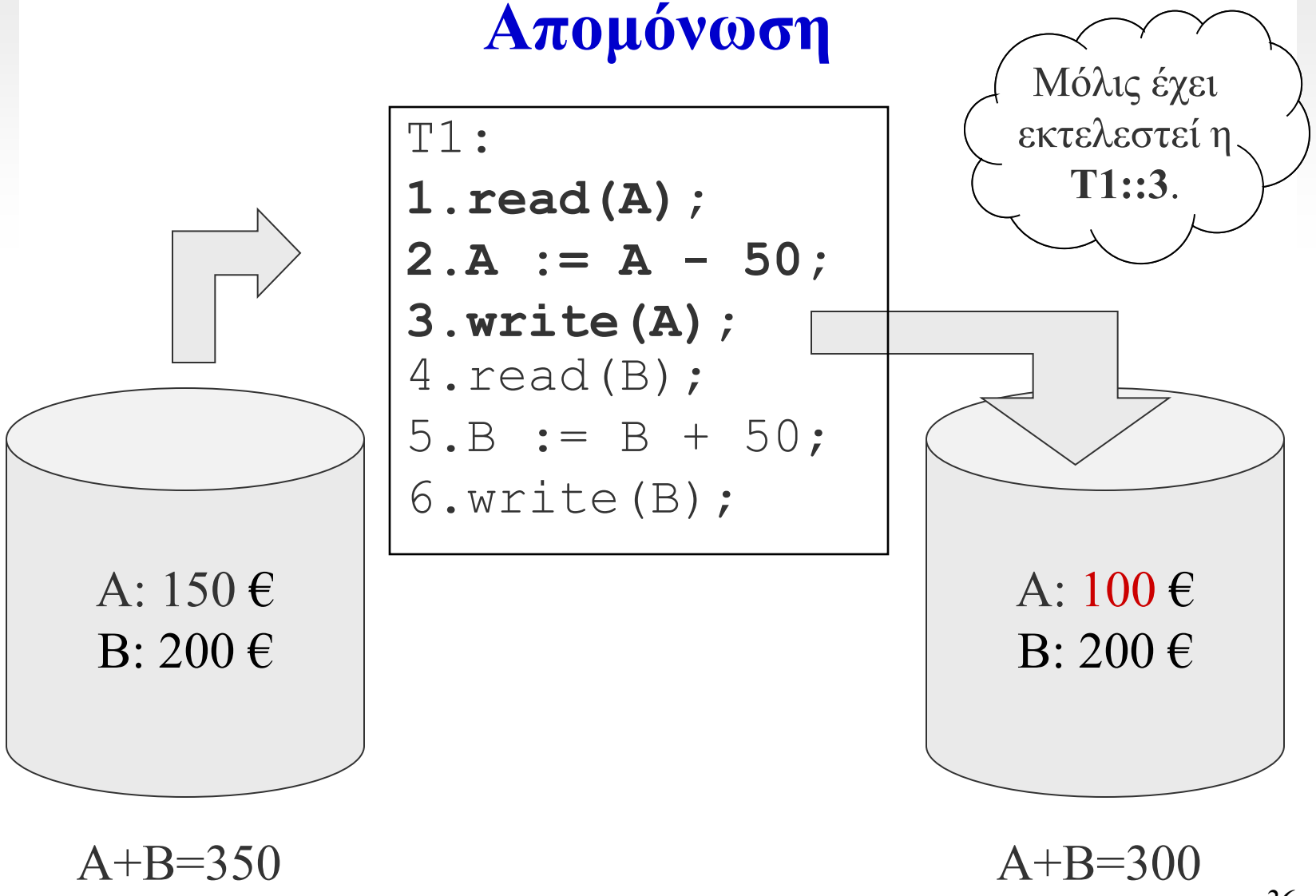
ACID Test

- (**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 ...

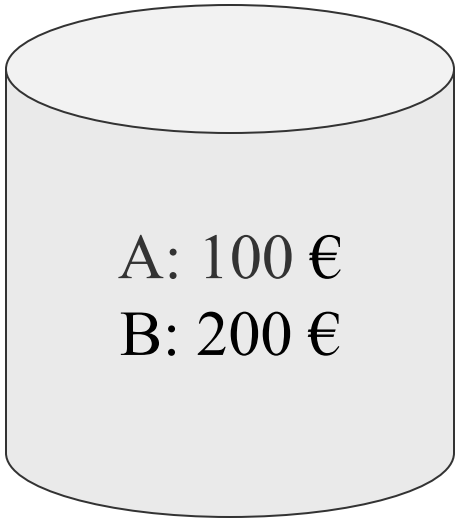
Απομόνωση


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**.



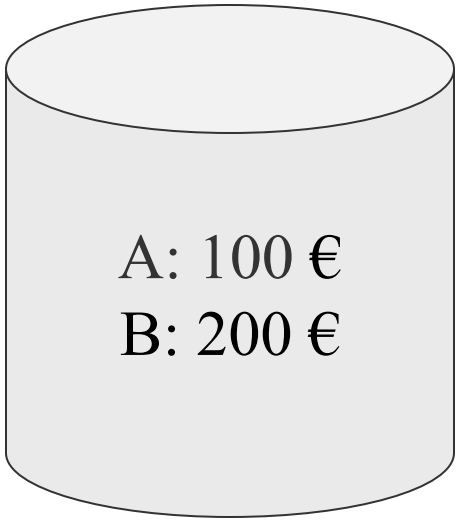

```
T2:
1.read(B);
2.If B<220
      B:=B*1.10;
3.write(B);
```

A+B=300

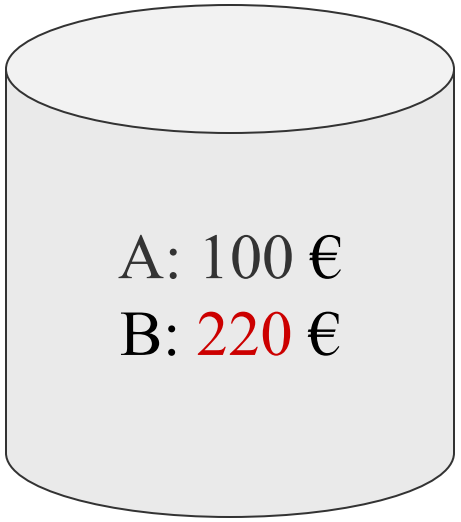
Απομόνωση

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

Μόλις έχει εκτελεστεί η **T2::3**.



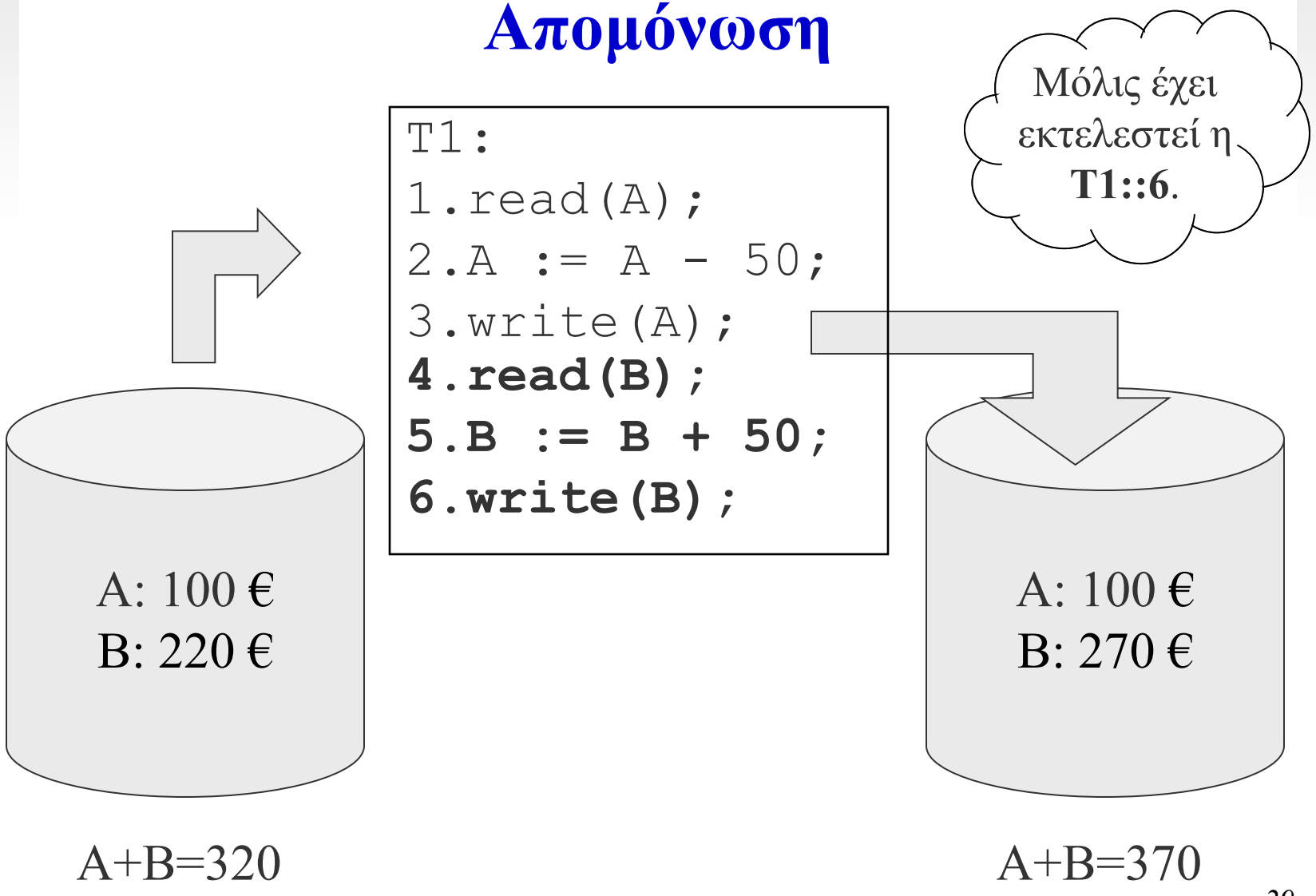

```
T2:
1.read(B);
2.If B<220
        B:=B*1.10;
3.write(B);
```



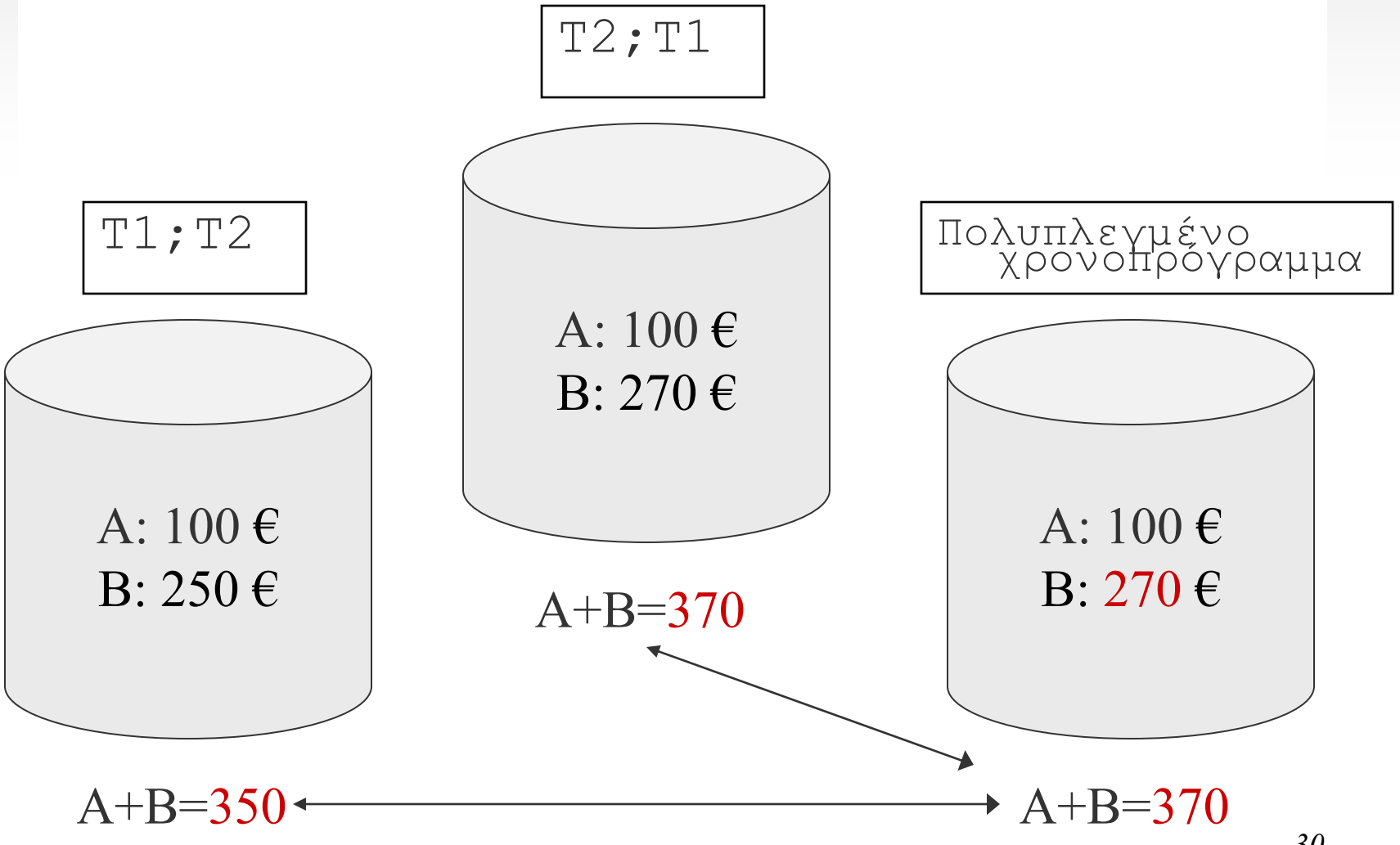
A+B=300

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: 100 €
B: 220 €
A: 100 €
B: 270 €
A+B=320
A+B=370

Απομόνωση


T2;T1
T1;T2
Πολυπλεγμένο χρονοπρόγραμμα
A: 100 €
B: 270 €
A: 100 €
B: 250 €
A: 100 €
B: 270 €
A+B=370
A+B=350
A+B=370

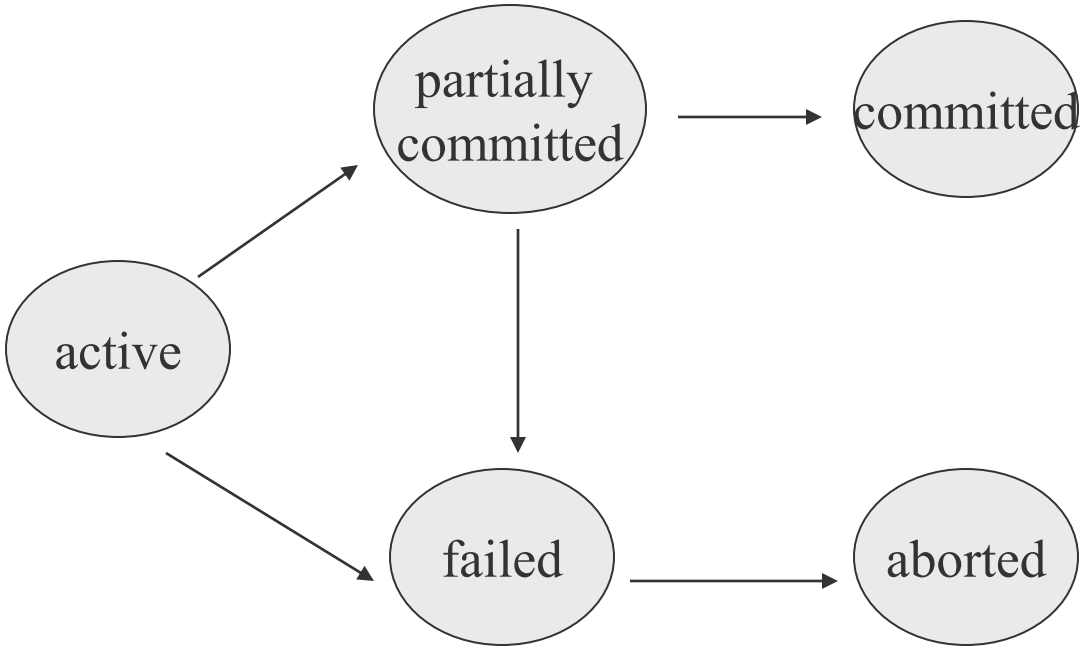
Απομόνωση

- 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... [σύντομα κοντά σας ☺]

Θεματολόγιο

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

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

Προχωρημένα Θέματα Τεχνολογίας και Εφαρμογών Βάσεων Δεδομένων

Διαχείριση Συναλλαγών

Πάνος Βασιλειάδης

pvassil@cs.uoi.gr

Μάρτιος 2014

www.cs.uoi.gr/~pvassil/courses/db_III/

Θεματολόγιο

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

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 κολλητά η μία με την άλλη

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

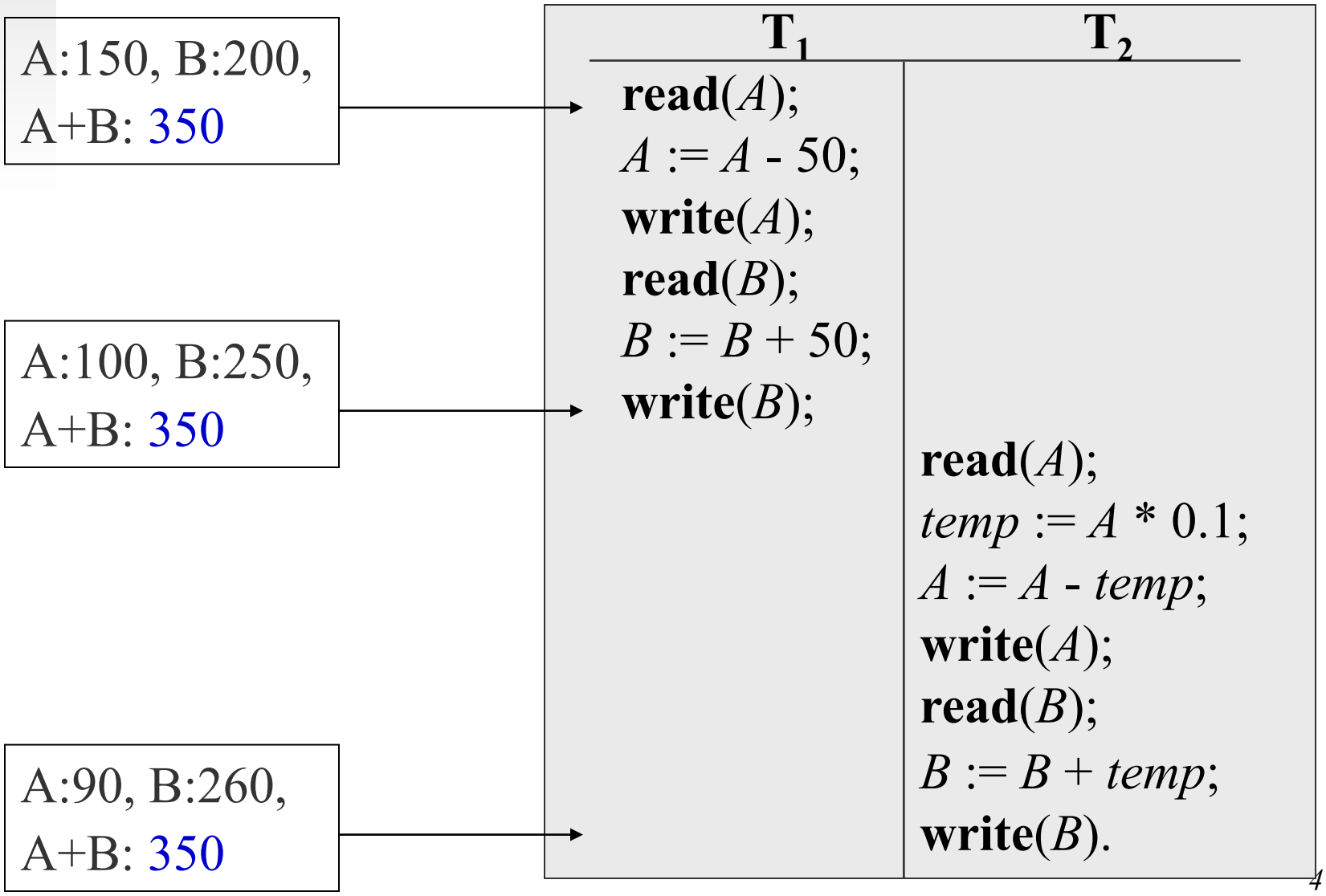
T_1	T_2
	read(A);
	$temp := A * 0.1$;
	$A := A - temp$;
	write(A);
	read(B);
	$B := B + temp$;
	write(B);
read(A);	
$A := A - 50$;	
write(A);	
read(B);	
$B := B + 50$;	
write(B).	

Σειριακό Χρονοπρόγραμμα

n! δυνατά σειριακά χρονοπρογράμματα για n συναλλαγές

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).

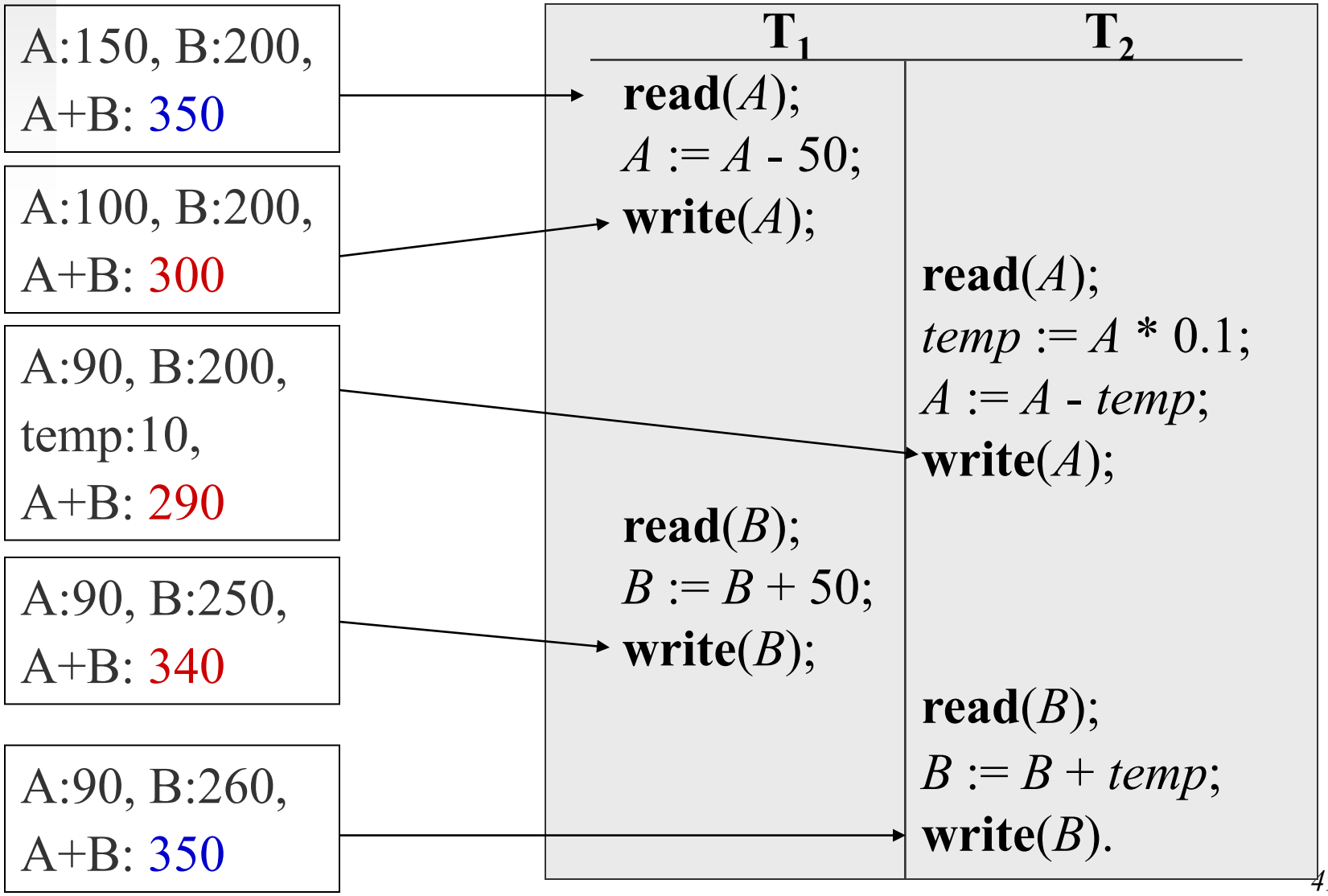
Σειριακό Χρονοπρόγραμμα



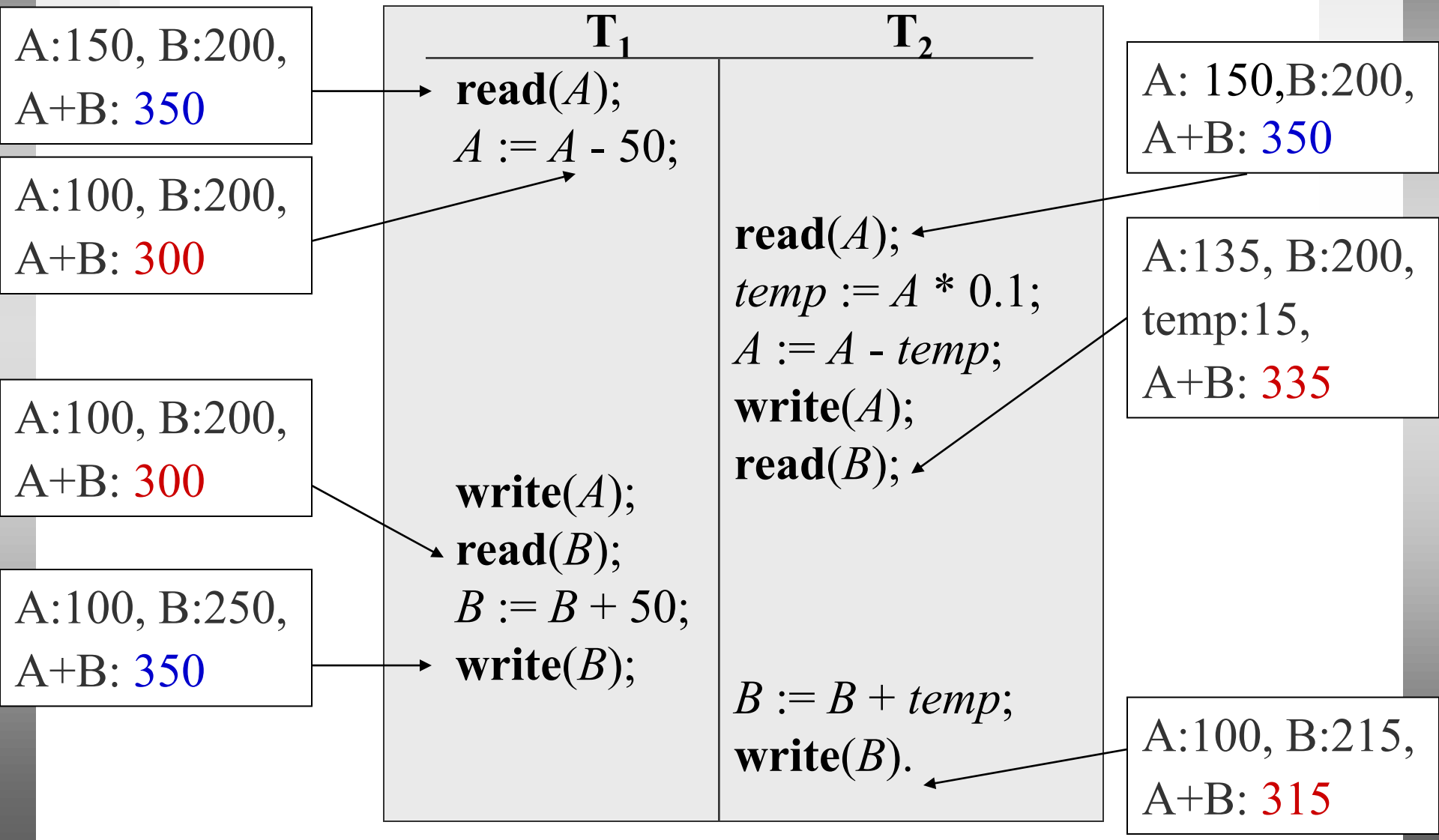
Προσοχή!!

- Κάθε συναλλαγή στο χρονοπρόγραμμα, είναι σαν συνάρτηση: έχει δικό της χώρο μνήμης [γι' αυτό και η τιμή των A, B εξαρτάται MONO 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)

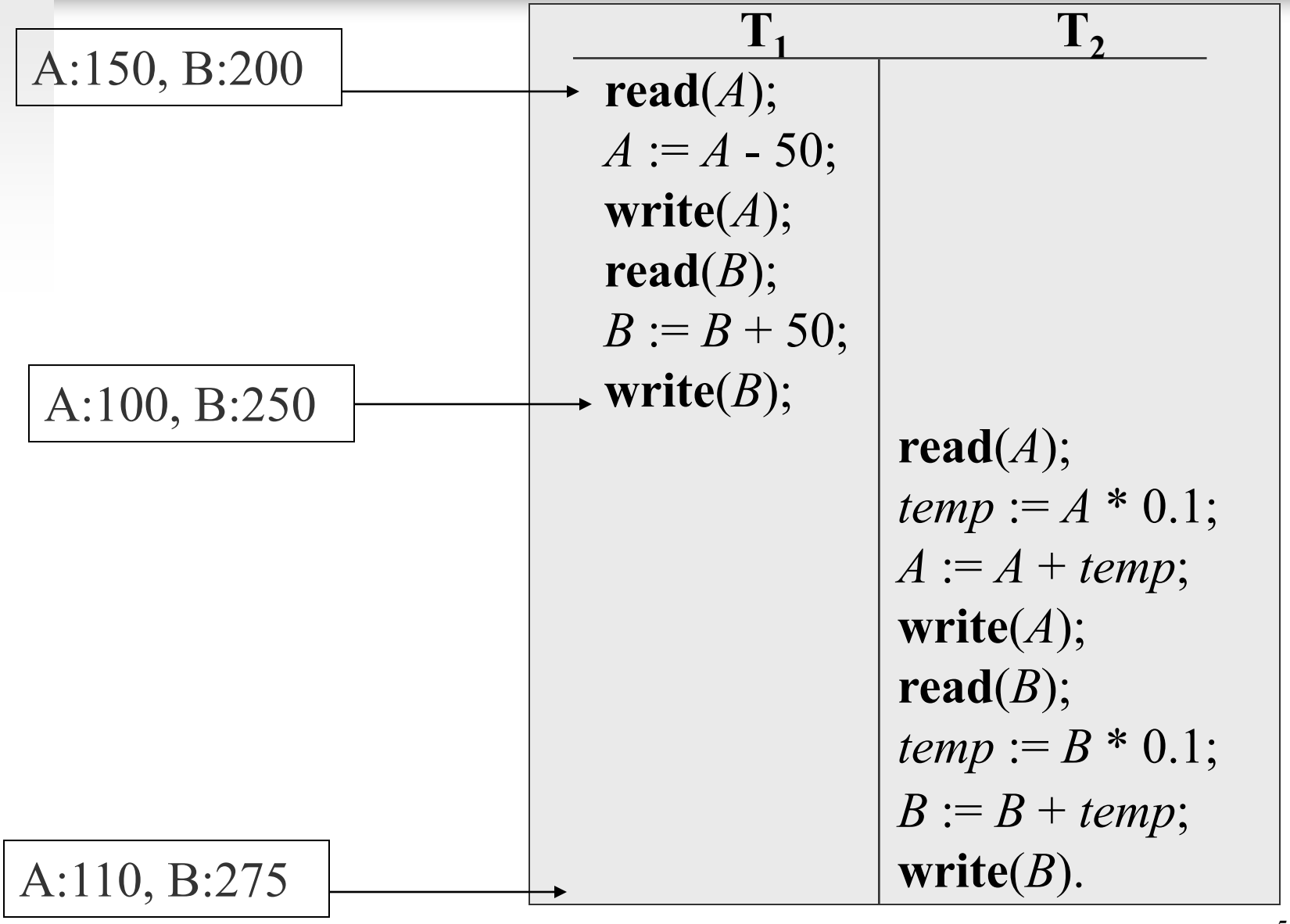
Παράδειγμα

- T1: μεταφέρει 50 € από τον Α στον Β
- T2: κάνει αύξηση στον Α και το Β κατά 10%

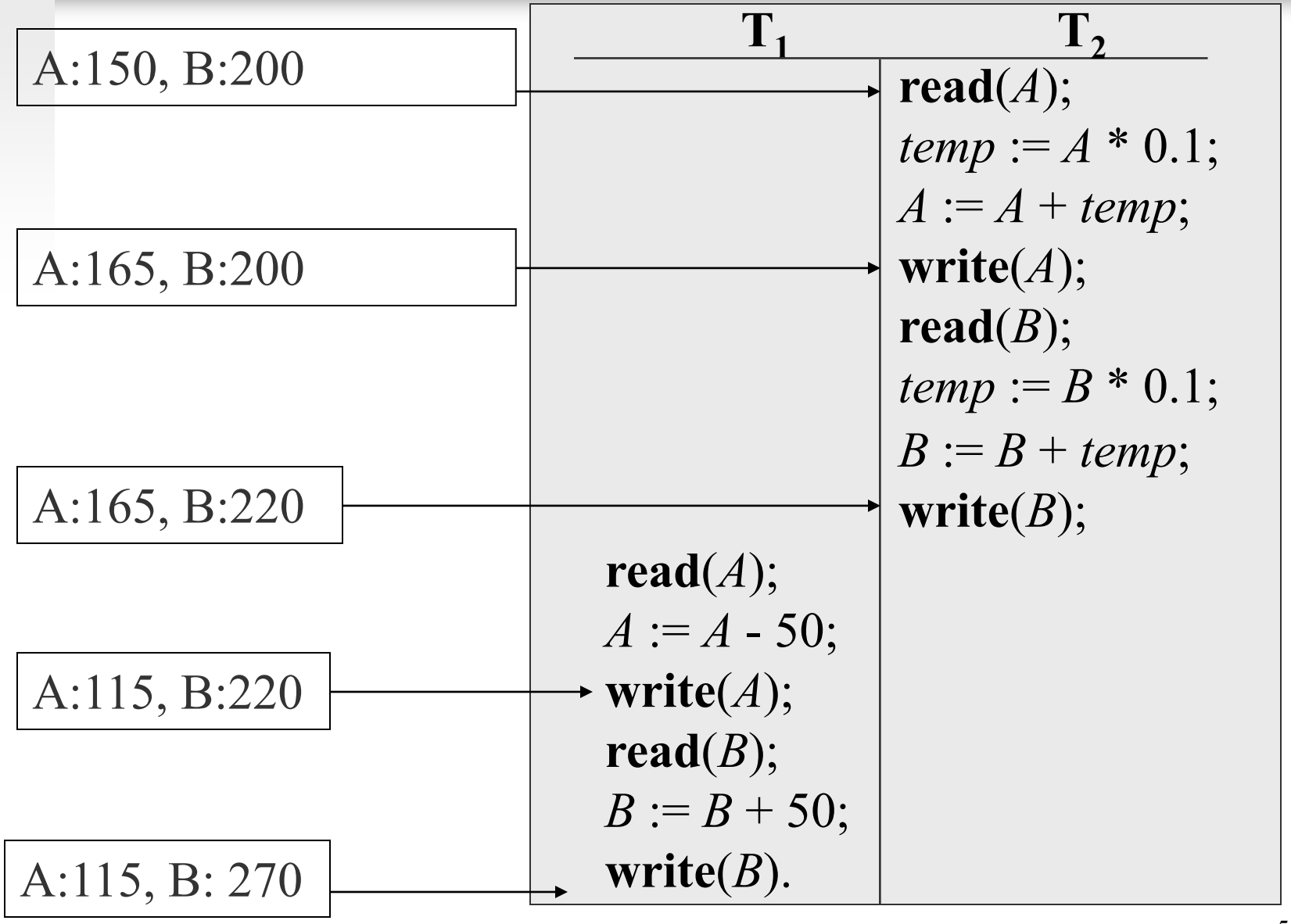
Προσοχή: Δεν υφίσταται περιορισμός για το Α+Β, πλέον!

Σκοπός είναι να δείξουμε προβλήματα που μπορούν να προκύψουν…

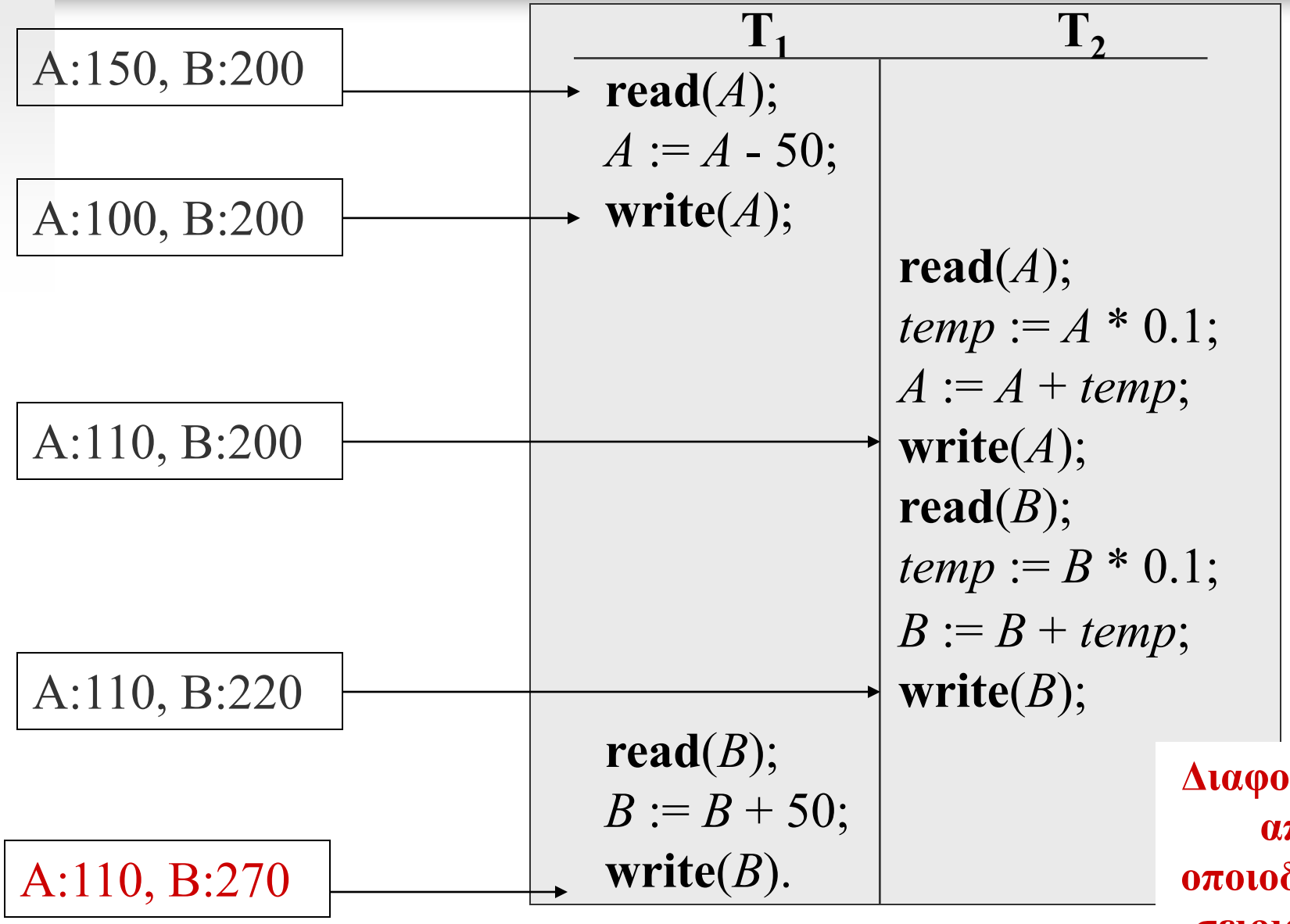
Σειριακό Χρονοπρόγραμμα



Σειριακό Χρονοπρόγραμμα

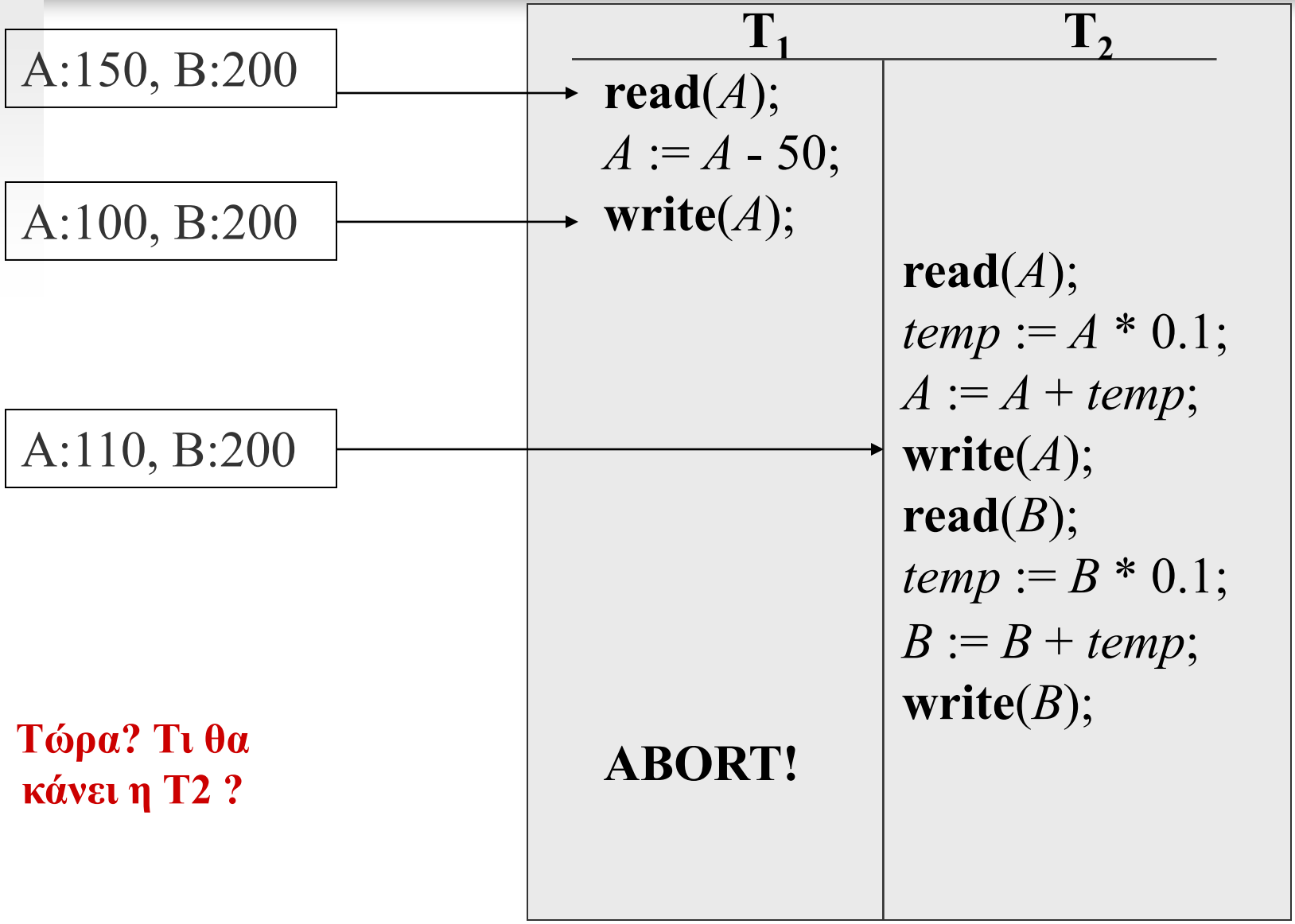


Ασυνεπής Ανάγνωση (Dirty Read)



Διαφορετικό από οποιοδήποτε σειριακό!!!

Ασυνεπής Ανάγνωση (Dirty Read)

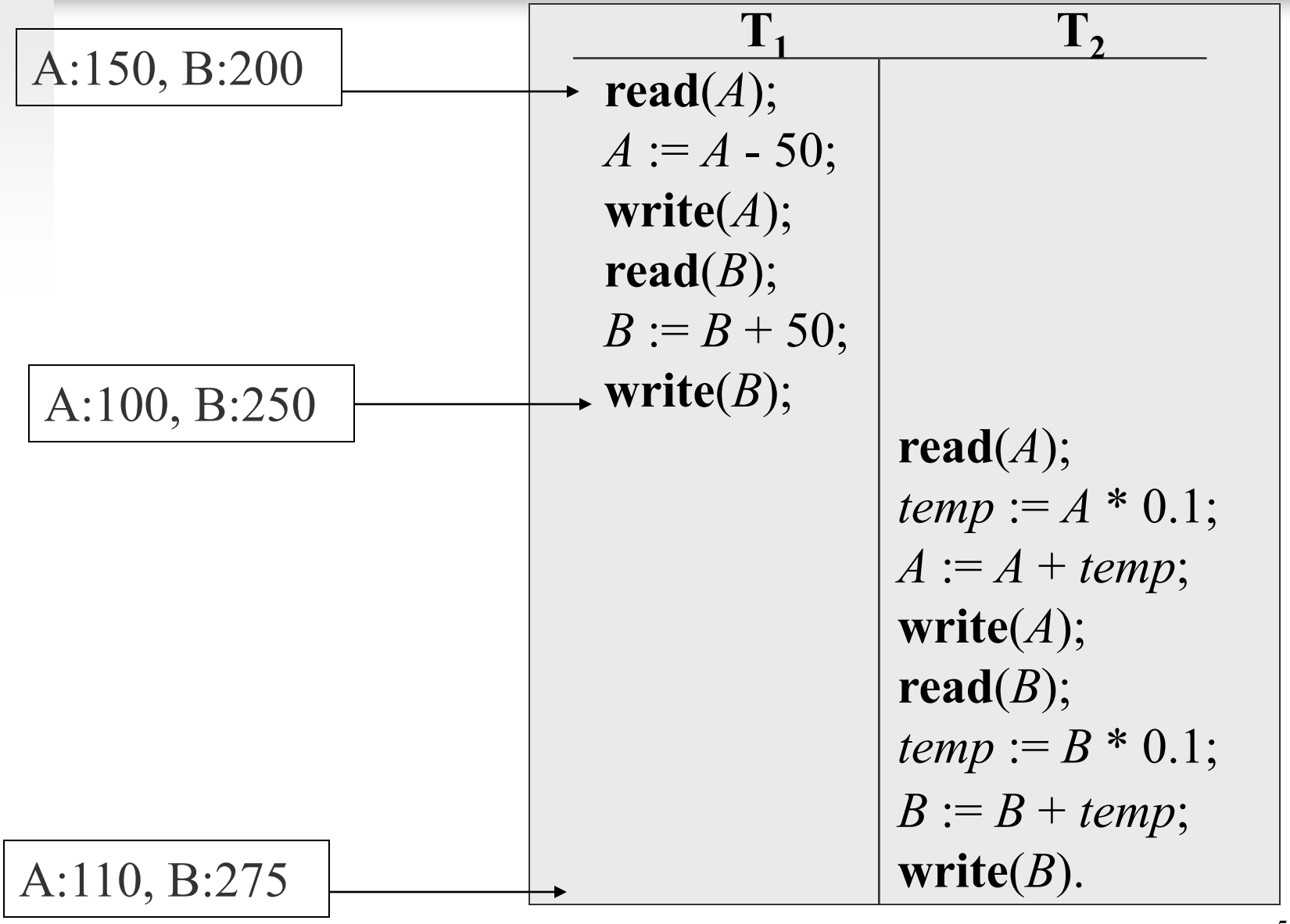


Τώρα? Τι θα κάνει η T2 ?

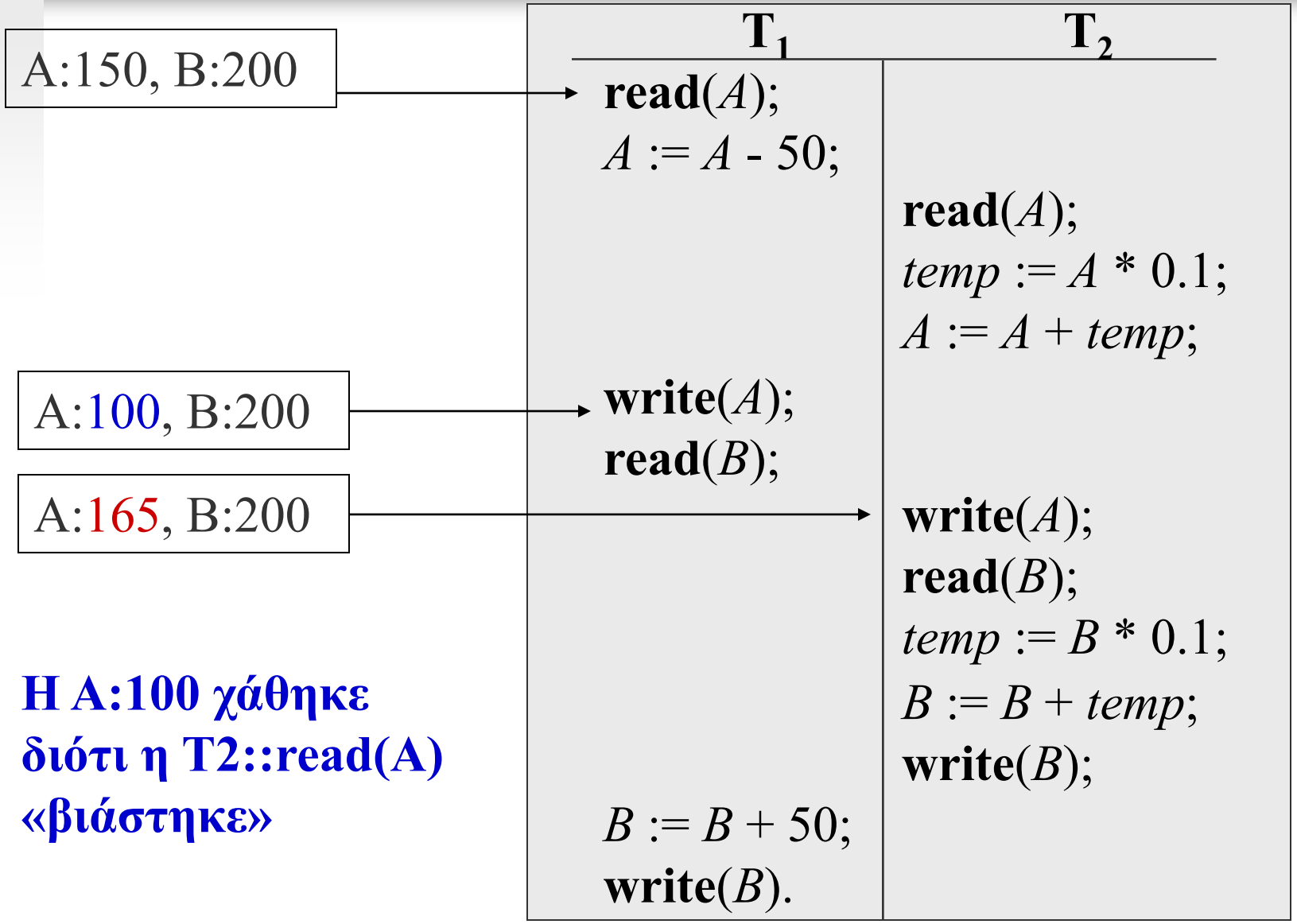
Dirty read

- Ο όρος προκύπτει από το γεγονός ότι η T2 διαβάζει μια τιμή για το A, ενώ η T1, η οποία είχε ξεκινήσει να το τροποποιεί, δεν έχει ολοκληρώσει ακόμα.
- Κατά συνέπεια, αν η T1 κάνει abort, πρέπει να κάνει και η T2 [ασχέτως που έχει ήδη ολοκληρώσει]...

Σειριακό Χρονοπρόγραμμα



Απώλεια Ενημερώσεων (lost updates)



Η A:100 χάθηκε διότι η T2::read(A) «βιάστηκε»

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).

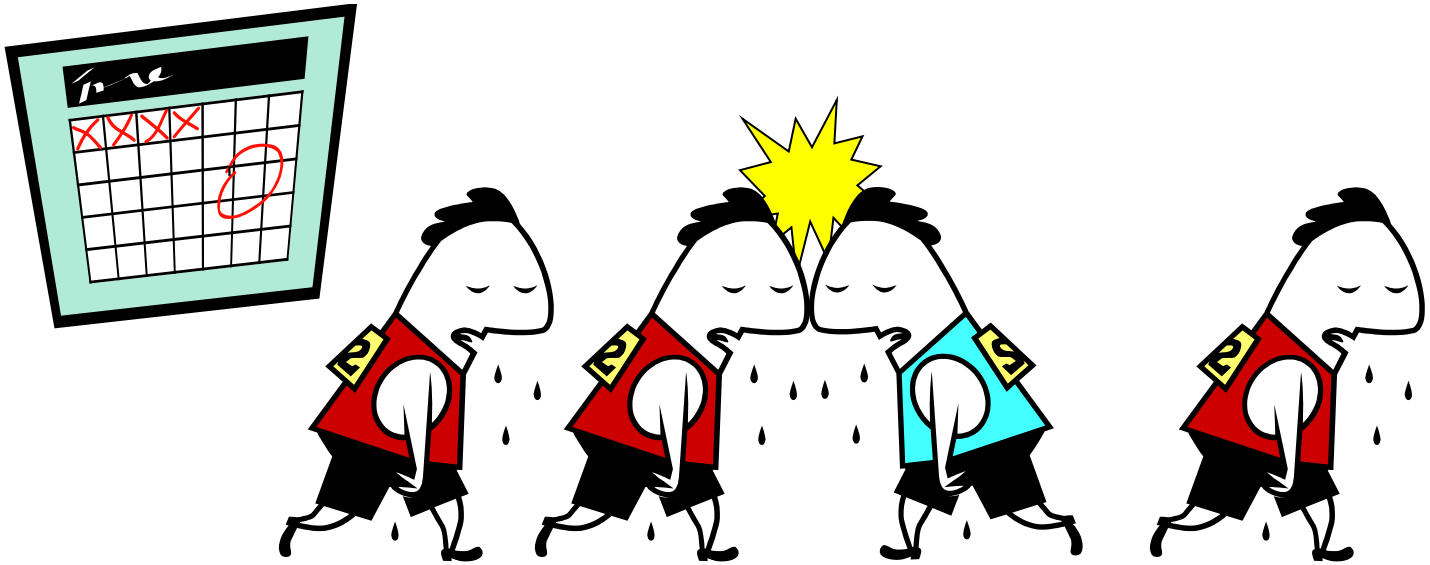
Μη επαναλήψιμες αναγνώσεις (Non-repeatable reads)

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

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

Επί της ουσίας ...

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



Θεματολόγιο

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

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.

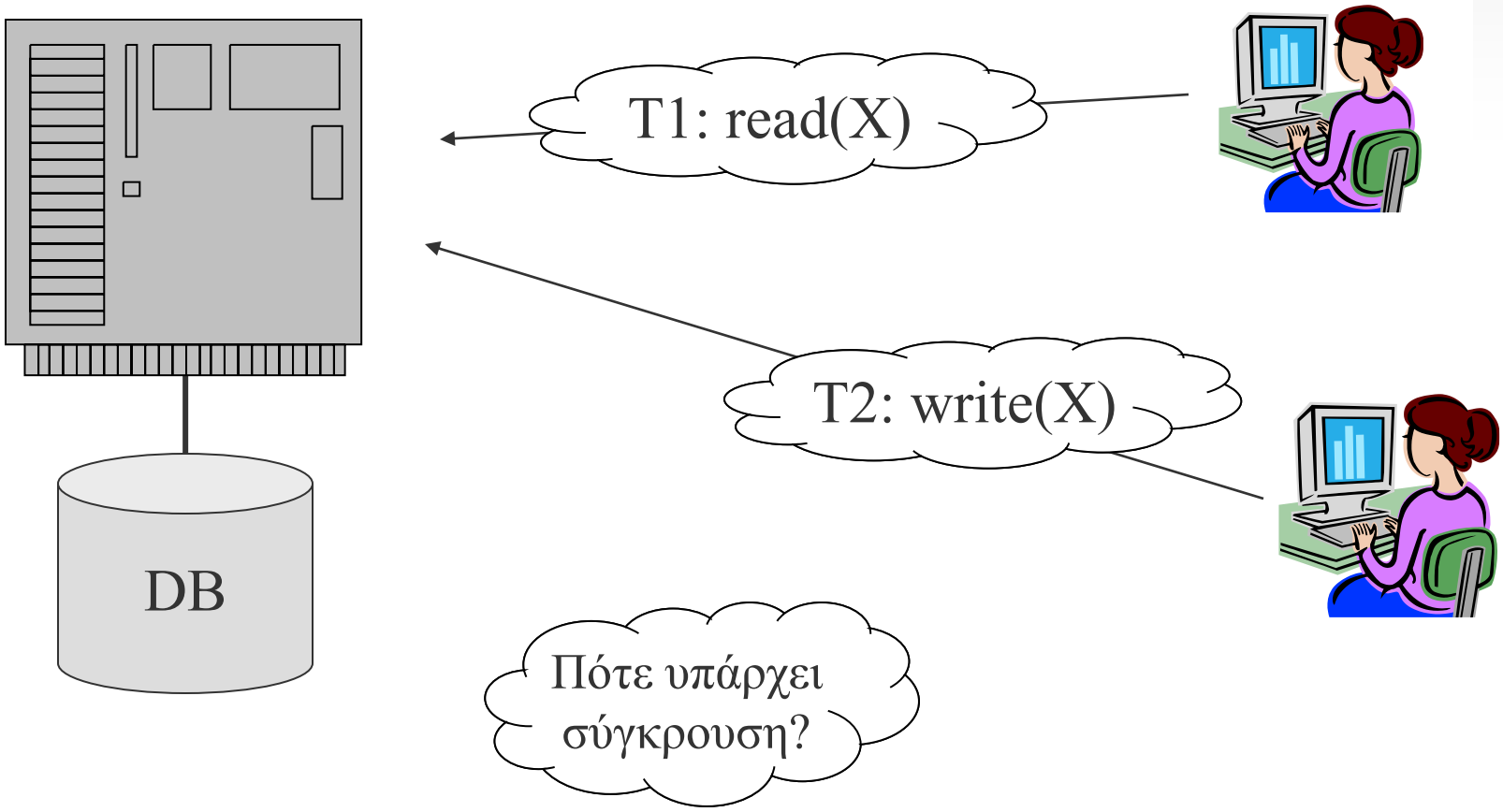
Συνέπεια: θυμάστε τι πρεσβεύει η έννοια ?

Σειριοποιησιμότητα

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

- **Στο εξής, θα αγνοούμε το processing στη μνήμη και θα μας απασχολούν μόνο οι read και write αλληλεπιδράσεις με 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 A. Πότε θα το επιτρέψουμε? [Εναλλακτικά:, πότε συγκρούονται οι ενέργειές τους?]

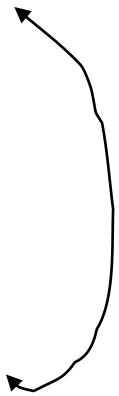
T1\T2	Read	Write
Read	☑	☒
Write	☒	☒

Σειριοποιησιμότητα συγκρούσεων

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

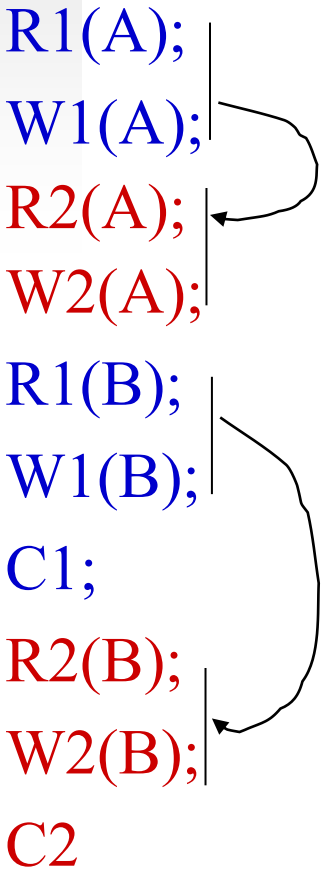
Σειριακό Χρονοπρόγραμμα S1

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).

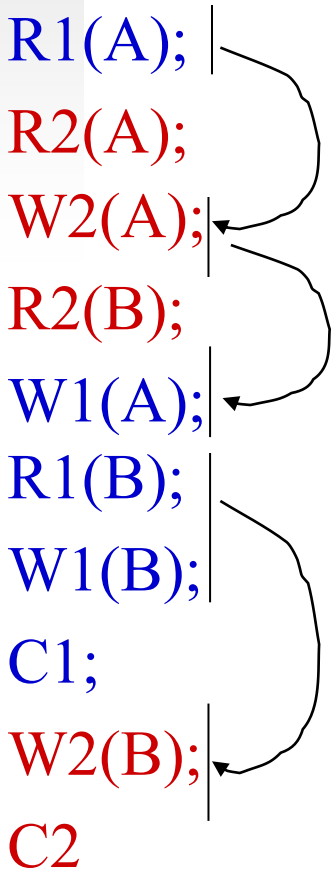
Μη Σειριακό Χρονοπρόγραμμα S2



Multiple conflicts hidden in the arrows; observe the order however: in every conflict, "blue" T1 is **before** the "read" T2, **always!**

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

Προβληματικό Χρονοπρόγραμμα S3



observe the order of conflicts: in in the first conflict, "blue" T1 is before the "read" T2, but then, T2 is before the next conflict on A with T1. **The order in the conflicts changes!**

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

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 [και όχι τυπικά]

- 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...

Επιβεβαιώστε με τα προηγούμενα...

Θεματολόγιο

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

Γράφος Σειριοποιησιμότητας

- Μοντελοποιούμε τις συγκρούσεις ενός χρονοπρογράμματος με ένα γράφημα, που 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_i.

Ήτοι, για κάθε σύγκρουση $\text{Πράξη}_i(X);\text{Πράξη}_j(X)$ μια ακμή από τον προηγούμενο κόμβο i στον επόμενο κόμβο j

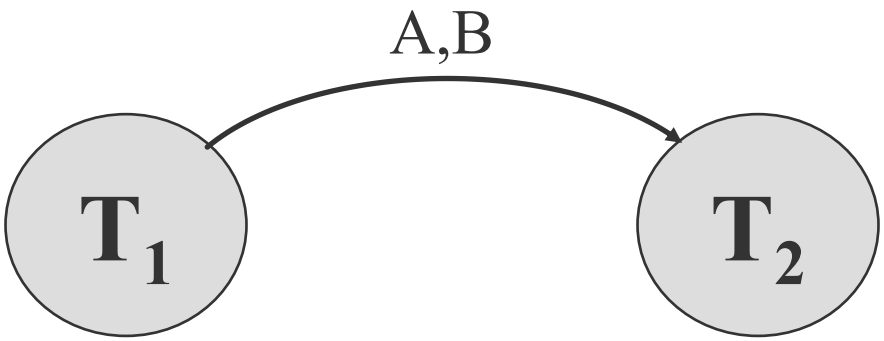
Γράφος σειριοποιησιμότητας

- 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

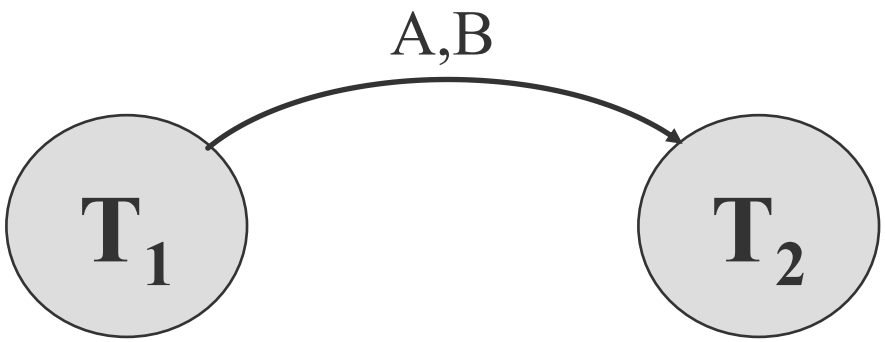


Θυμίζω: ακμή από τον προηγούμενο κόμβο i στον επόμενο κόμβο j

Γράφος σειριοποιησιμότητας

- S2:

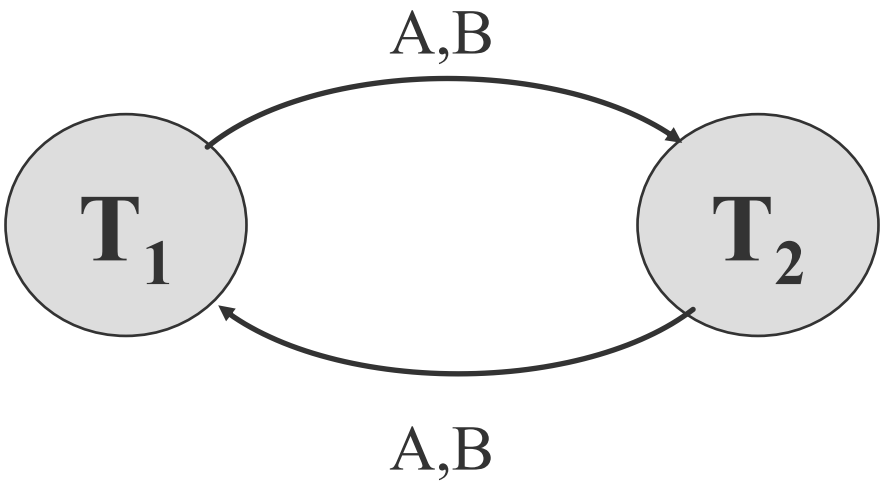
R1(A);W1(A); R2(A);W2(A);R1(B);W1(B);C1;R2(B);W2(B);C2



Γράφος σειριοποιησιμότητας

- S3:

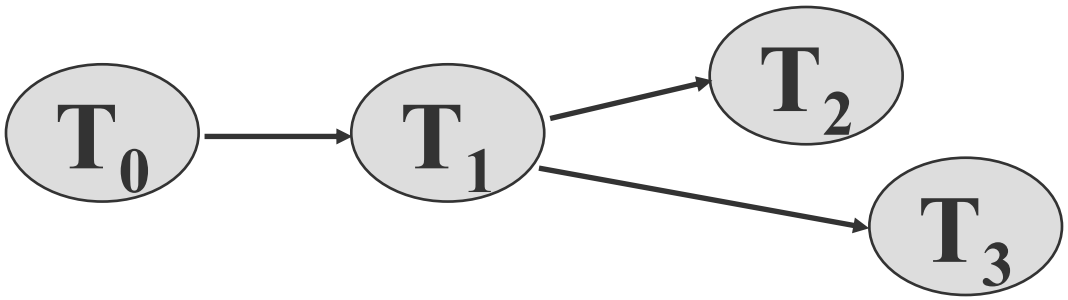
R1(A); R2(A);W2(A); R2(B);W1(A); R1(B);W1(B);C1;W2(B);C2



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.



T_0
T_1
T_3
T_2

T_0
T_1
T_2
T_3

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?

Θεώρημα

Σειριοποιησιμότητα συγκρούσεων <=>
έλλειψη κύκλου στο γράφημα

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

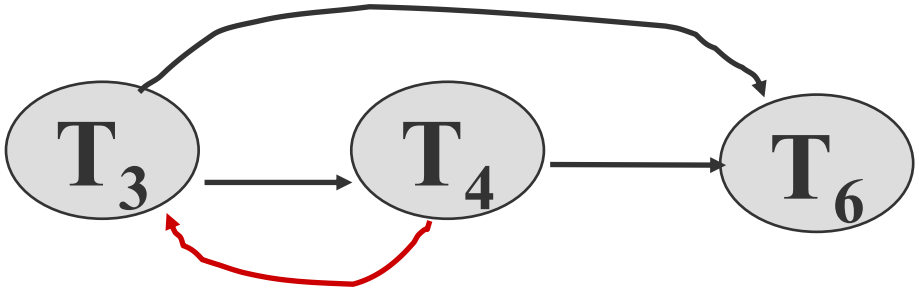
Σειριοποιησιμότητα όψεως

- 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)

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

Εν γένει, σε σχέση με τη σειριοποιησιμότητα όψεως, στο πλαίσιο του μαθήματος, το πολύ να σας ζητηθεί να υποψιαστείτε αν ένα χρονοπρόγραμμα είναι σειριοποιήσιμο...

Αλγόριθμος για view serializability

- Από τον γράφο προτεραιότητας με ετικέτες στις ακμές,
- προσπάθησε να βρεις ένα συνεκτικό ακυκλικό γράφο,
- διαλέγοντας από κάθε ζεύγος ακμών με ίδια ετικέτα, μία εκ των δύο...

Συναλλαγές & Ταυτοχρονισμός