Έλεγχος Ταυτοχρονισμού

(εμπλουτισμένο υλικό)

http://delab.csd.auth.gr/courses/c_dbimpl/

Ευχαριστίες

- Μέρος του υλικού είναι βασισμένο στο βιβλίο “Database Systems: The Complete Book”
  - Κεφάλαια 18.1-18.3
- και στο 15ο κεφάλαιο του βιβλίου «Συστήματα Βάσεων Δεδομένων: θεωρία και πρακτική εφαρμογή» των Ι. Μανωλόπουλου και Α.Ν. Παπαδόπουλου.

Εισαγωγικά

- Ένα σύνολο λειτουργιών το οποίο αποτελεί μία λογική λειτουργική μονάδα καλείται συναλλαγή (transaction).
- Μια συναλλαγή περιέχει:
  - εντολές ανάγνωσης, εισαγωγής, διαγραφής ή ενημέρωσης των δεδομένων της ΒΔ.
- Σε επίπεδο σελίδων Χ, μπορούμε να θεωρήσουμε ότι μία συναλλαγή είτε διαβάζει τη Χ, είτε εγγράφει/τροποποιεί τη Χ.
- Η πολυπλοκότητα των σύγχρονων βάσεων δεδομένων οδηγεί στην ανάπτυξη αξιόπιστων μηχανισμών ελέγχου της ακεραιότητας των δεδομένων:
  - πολλές συναλλαγές μπορεί να βρίσκονται σε εξέλιξη, προσπελαύνοντας κοινά δεδομένα.
- Ο **έλεγχος ταυτοχρονισμού (concurrency control)** διαχειρίζεται τις συναλλαγές έτσι ώστε να αποφεύγονται οι παθολογικές καταστάσεις.
  - Συμπληρωματικά, υπάρχει ανάγκη ύπαρξης μηχανισμών επανάκτησης (recovery) δεδομένων.

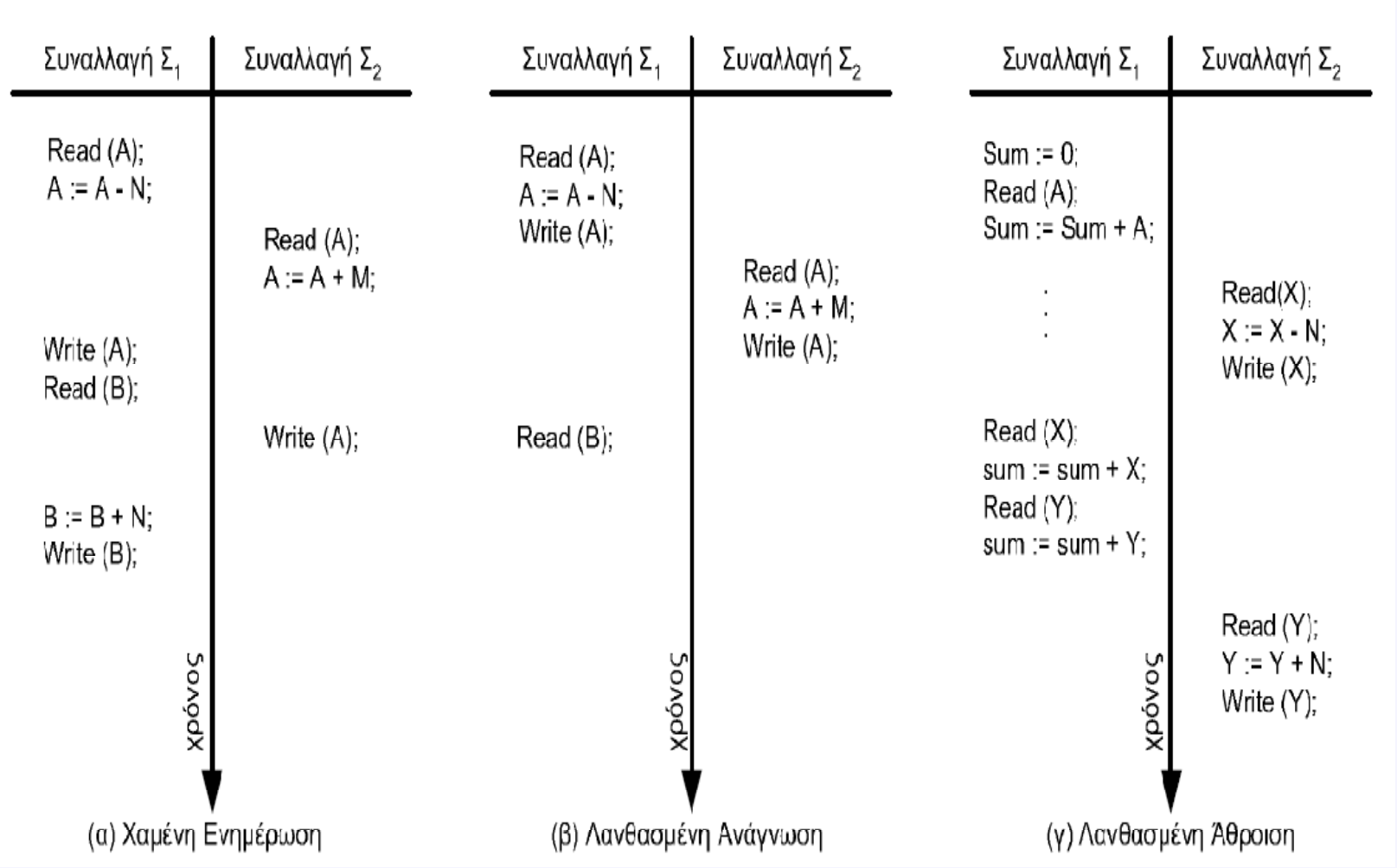
Ιδιότητες συναλλαγών ACID

- Ατομικότητα (**A**tomicity):
  - αν υπάρχει έστω και μία εντολή της συναλλαγής, η οποία αποτυγχάνει να εκτελεσθεί, τότε ολόκληρη η συναλλαγή αποτυγχάνει επίσης.
- Συνέπεια (**C**onsistency):
  - η συναλλαγή πρέπει να μετατρέπει τη βάση δεδομένων από μία συνεπή κατάσταση σε μία άλλη συνεπή κατάσταση (τα δεδομένα πρέπει να είναι ορθά). Οι μηχανισμοί ακεραιότητας δεδομένων δεν επαρκούν για την εγγύηση της συνέπειας.
- Απομόνωση (**I**solation):
  - κάθε συναλλαγή πρέπει να εκτελείται ανεξάρτητα από άλλες συναλλαγές.
- Μονιμότητα (**D**urability):
  - αν μία συναλλαγή ολοκληρωθεί με επιτυχία, τότε οι αλλαγές που έχει επιφέρει καταγράφονται μόνιμα στη ΒΔ και δεν μπορούν να ανακληθούν.

Τρόπος επεξεργασίας

- Ακολουθιακή ή σειριακή εκτέλεση
  - όπου οι συναλλαγές εκτελούνται η μία μετά την άλλη (άρα δεν υπάρχουν προβλήματα ταυτοχρονισμού) αλλά εμφανίζονται σημαντικές καθυστερήσεις με αποτέλεσμα να μειώνεται η γενική απόδοση του συστήματος.
- Ταυτόχρονη εκτέλεση πολλών συναλλαγών
  - Παράλληλη χρήση δίσκου ή CPU από πολλαπλές συναλλαγές, οπότε αυξάνεται ο αριθμός των συναλλαγών που ολοκληρώνονται στη μονάδα του χρόνου (throughput).
  - Πλεονέκτημα: μειώνονται οι καθυστερήσεις (waiting time) και ο μέσος χρόνος εκτέλεσης των συναλλαγών (mean response time).
  - Μειονέκτημα: ενδέχεται να επιφέρει προβλήματα στην ακεραιότητα και **συνέπεια** των δεδομένων της ΒΔ. Τα προβλήματα αυτά επιλύει ο έλεγχος ταυτοχρονισμού.
    - Δημιουργείται ένα **χρονοδιάγραμμα (schedule)** που ικανοποιεί τις απαιτήσεις συνέπειας των δεδομένων.

Προβλήματα με χρονοδιαγράμματα



Στο (β) θεωρείται ότι η Σ_1 απέτυχε και απορρίφθηκε.

Παράδειγμα

T1:	T2:
Read(A)	Read(A)
A ← A+100	A ← A×2
Write(A)	Write(A)
Read(B)	Read(B)
B ← B+100	B ← B×2
Write(B)	Write(B)

Περιορισμός: A=B

Παράδειγμα - Χρονοδιάγραμμα Α

T1	T2	A	B
		25	25
Read(A); A ← A+100			
Write(A);		125	
Read(B); B ← B+100;			
Write(B);			125
	Read(A);A ← A×2;		
	Write(A);	250	
	Read(B);B ← B×2;		
	Write(B);		250
		250	250

Παράδειγμα - Χρονοδιάγραμμα Β

T1	T2	A	B
		25	25
	Read(A);A ← A×2;		
	Write(A);	50	
	Read(B);B ← B×2;		
	Write(B);		50
Read(A); A ← A+100			
Write(A);		150	
Read(B); B ← B+100;			
Write(B);			150
		150	150

Παράδειγμα – Χρονοδιάγραμμα Γ

T1	T2	A	B
		25	25
Read(A); A ← A+100			
Write(A);		125	
	Read(A);A ← A×2;		
	Write(A);	250	
Read(B); B ← B+100;			
Write(B);			125
	Read(B);B ← B×2;		
	Write(B);		250
		250	250

Παράδειγμα –Χρονοδιάγραμμα Δ

T1	T2
Read(A); A ← A+100	
Write(A);	
	Read(A);A ← A×2;
	Write(A);
	Read(B);B ← B×2;
	Write(B);
Read(B); B ← B+100;	
Write(B);	

A	B
25	25
125	
250	
	50
	150
250	150

Παράδειγμα – Χρονοδιάγραμμα Ε

T1	T2’	A	B
		25	25
Read(A); A ← A+100			
Write(A);		125	
	Read(A);A ← A×1;		
	Write(A);	125	
	Read(B);B ← B×1;		
	Write(B);		25
Read(B); B ← B+100;			
Write(B);			125
		125	125

Αλλαγή της συναλλαγής Τ2 σε σχέση με το Χρονοδιάγραμμα Δ. Η τελική κατάσταση είναι συνεπής από <u>σύμπτωση</u>!

Σωστά χρονοδιαγράμματα

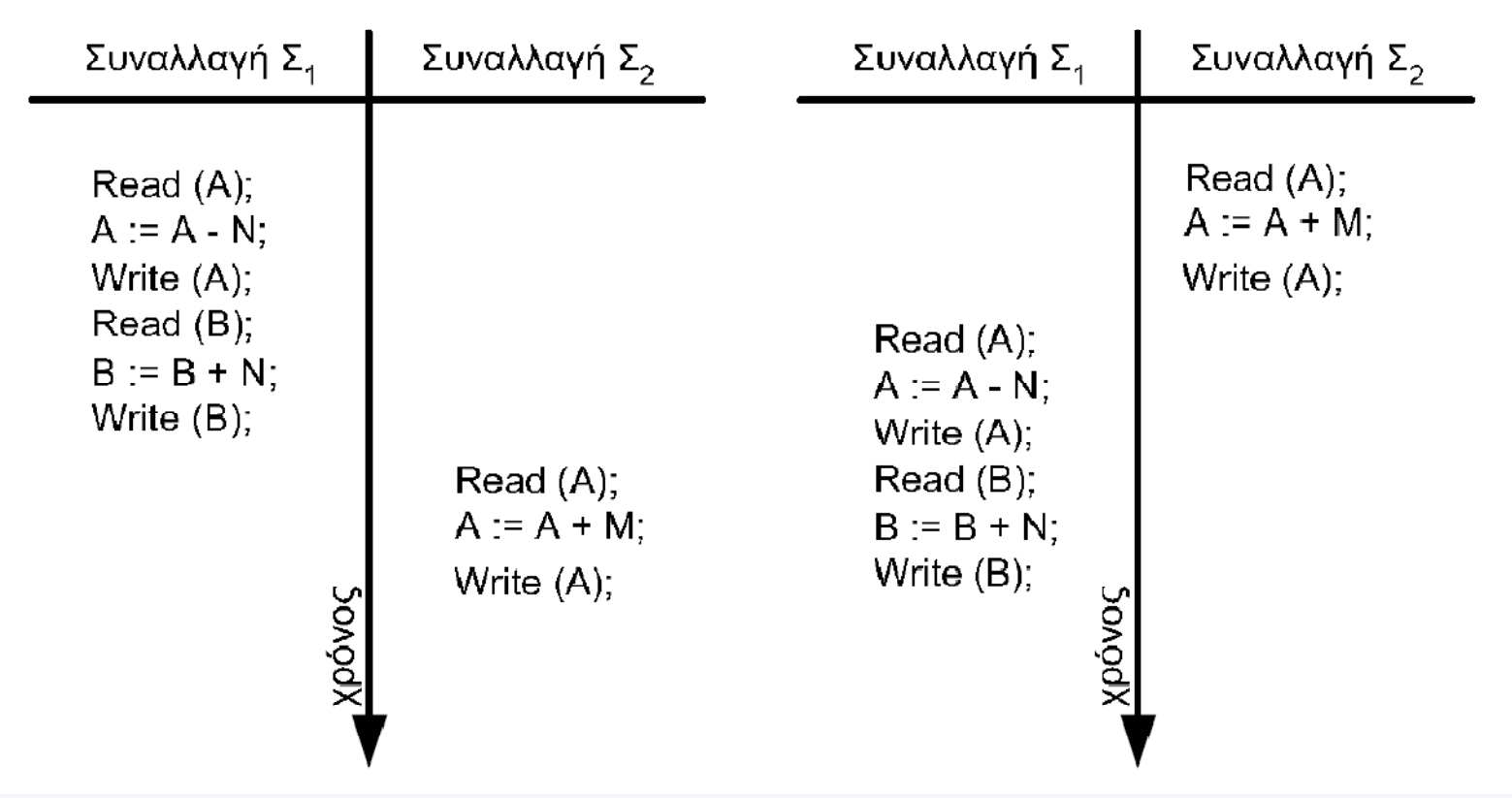
- Πρέπει να ορίζονται ανεξάρτητα από:
  - Αρχική κατάσταση.
  - Τρόπο αλλαγής των δεδομένων.
- Η ορθότητα πρέπει να εξαρτάται απόκλειστικά από την σειρά των αναγνώσεων (reads) και εγγραφών (writes)

Το χρονοδιάγραμμα Γ ξαναγράφεται:

$S_Γ = r_1(A)w_1(A)r_2(A)w_2(A)r_1(B)w_1(B)r_2(B)w_2(B)$

Σειριακό χρονοδιάγραμμα

- Σε ένα **σειριακό** (serial) χρονοδιάγραμμα οι εντολές μίας συναλλαγής δεν περιπλέκονται χρονικά με εντολές των άλλων συναλλαγών → επιτυγχάνεται η συνέπεια.
- Υπάρχουν n! σειριακά χρονοδιαγράμματα για n συναλλαγές.



Σειριοποίηση

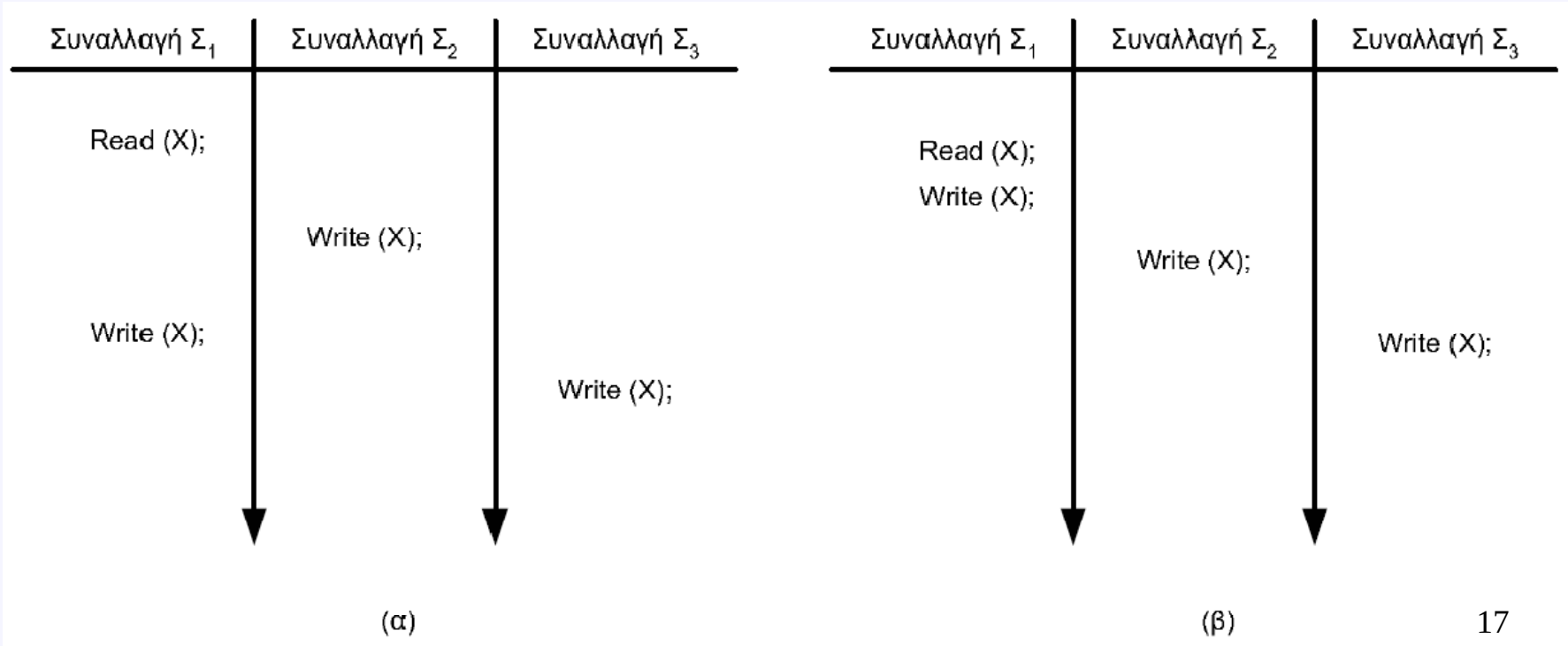
- Για να διατηρεί τη συνέπεια των δεδομένων της ΒΔ θα πρέπει το χρονοδιάγραμμα εκτέλεσης συναλλαγών να είναι ισοδύναμο με κάποιο σειριακό χρονοδιάγραμμα.
- Δύο χρονοδιαγράμματα καλούνται **ισοδύναμα** αν μετά την εκτέλεσή τους επιφέρουν τις ίδιες αλλαγές στη ΒΔ.
- Η παραγωγή ενός χρονοδιαγράμματος ισοδύναμου με ένα σειριακό χρονοδιάγραμμα καλείται **σειριοποίηση** (serializability), που μπορεί να έχει δύο μορφές:
  - **σειριοποίηση σύγκρουσης** (conflict), και
  - *σειριοποίηση όψης (view).*
- ***Σειριοποίηση → επίτευξη (και) απομόνωσης!***

Σειριοποίηση όψης

- Έστω δύο χρονοδιαγράμματα S και S′ με τα ίδια σύνολα συναλλαγών. Τα S και S′ ονομάζονται **ισοδύναμα** αν για κάθε στοιχείο δεδομένων D:
  - αν η συναλλαγή Σ_i διαβάζει την αρχική τιμή του D στο χρονο-διάγραμμα S, τότε πρέπει η συναλλαγή Σ_i επίσης να διαβάζει την αρχική τιμή του D στο χρονοδιάγραμμα S′,
  - αν η συναλλαγή Σ_i εκτελεί την εντολή Read(D) στο χρονοδιά-γραμμα S και αυτή η τιμή έχει παραχθεί από τη συναλλαγή Σ_j, τότε η Σ_i πρέπει επίσης να διαβάσει την τιμή που παράγει η Σ_j στο χρονοδιάγραμμα S′,
  - η συναλλαγή που εκτελεί την τελευταία εντολή Write(D) στο χρονοδιάγραμμα S, πρέπει και στο χρονοδιάγραμμα S' να εκτελεί την τελευταία εντολή Write(D).

Σειριοποίηση όψης

- Ένα χρονοδιάγραμμα S καλείται **σειριοποιήσιμο ως προς την όψη** (view serializable) αν είναι ισοδύναμο ως προς την όψη με κάποιο σειριακό χρονοδιάγραμμα.



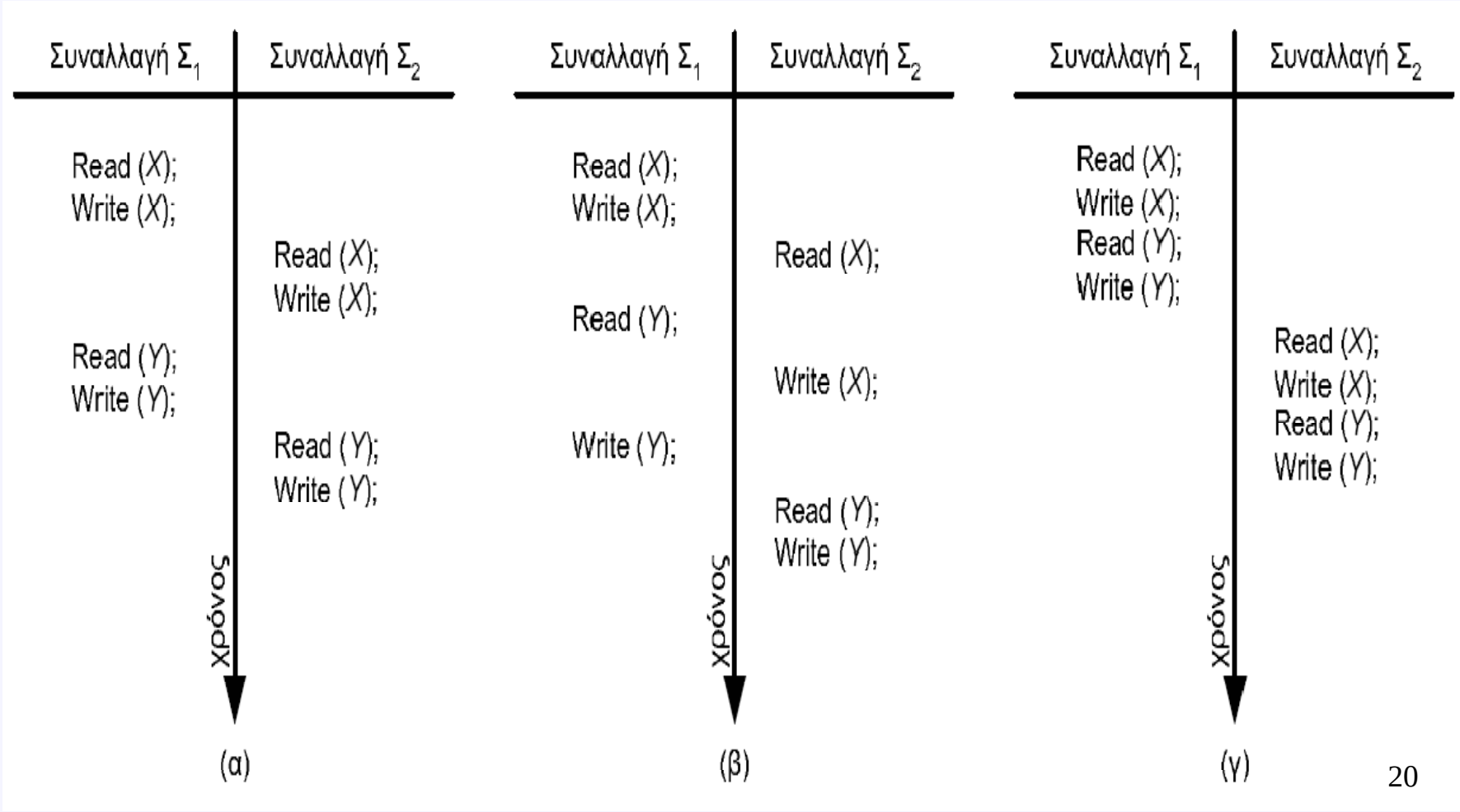
Σειριοποίηση σύγκρουσης (1)

- Έστω χρονοδιάγραμμα S και δύο χρονικά συνεχόμενες εντολές E_i, E_j (που αναφέρονται στα ίδια δεδομένα D) των συναλλαγών Σ_1, Σ_2, αντιστοίχως.
- Διακρίνουμε τις εξής περιπτώσεις:
  - E_i = read(D) και E_j = read(D)
  - E_i = read(D) και E_j = write(D)
  - E_i = write(D) και E_j = read(D)
  - E_i = write(D) και E_j = write(D)
- Αν τουλάχιστον μία εντολή είναι εντολή αποθήκευσης, τότε οι εντολές E_i και E_j βρίσκονται σε σύγκρουση.
- Όλες οι υπόλοιπες εντολές, δηλ. αυτές που δεν βρίσκονται σε σύγκρουση, μπορούν να αλλάξουν σειρά χωρίς να δημιουργείται πρόβλημα.
  - Π.χ., $read_1(X)$ $write_2(Y)$ ισοδύναμο με $write_2(Y)$ $read_1(X)$

Σειριοποίηση σύγκρουσης (2)

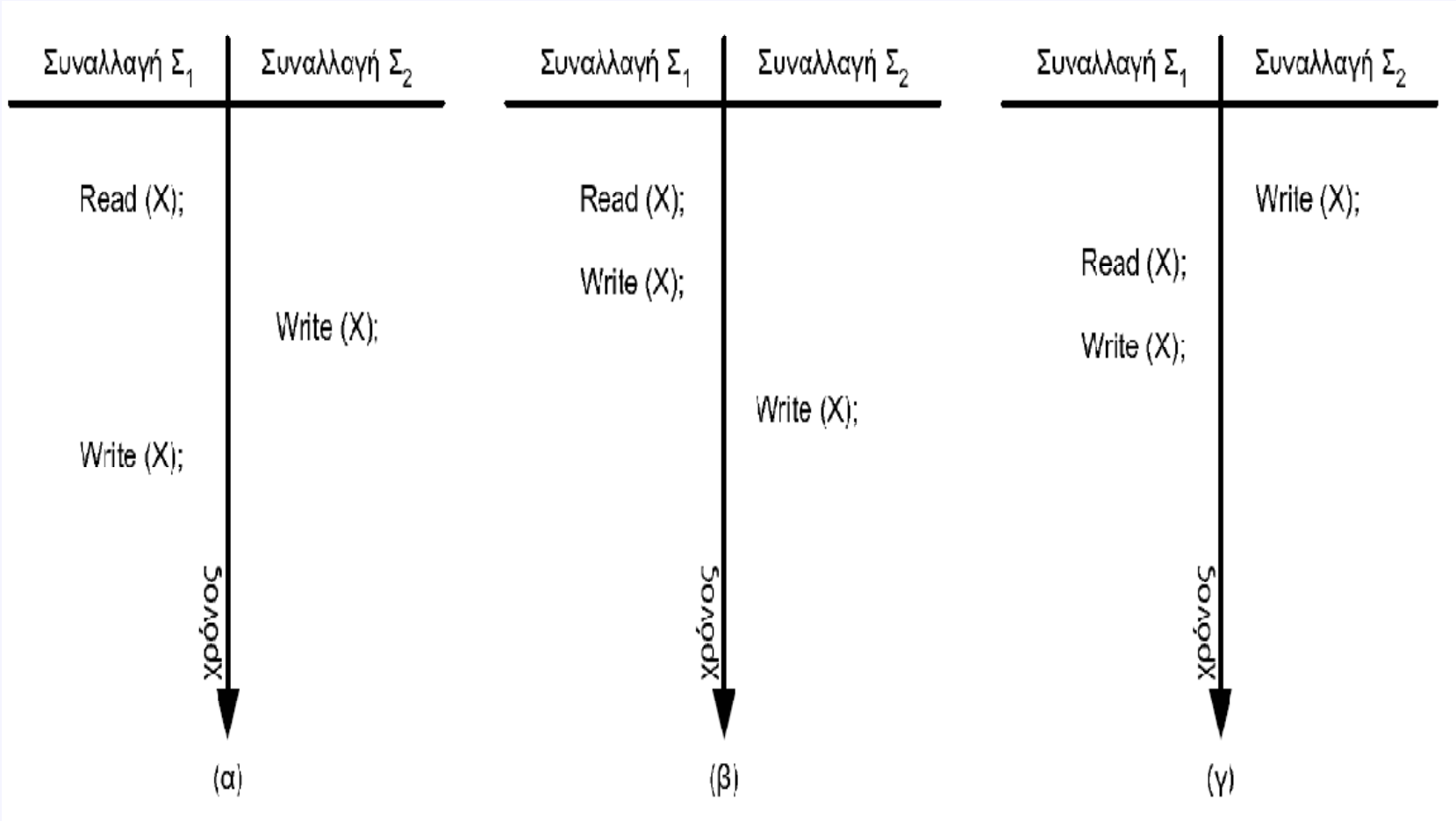
- Αν ένα χρονοδιάγραμμα εκτέλεσης S μπορεί να μετασχηματισθεί σε ένα άλλο χρονοδιάγραμμα S' αλλάζοντας τη σειρά εκτέλεσης διαδοχικών εντολών που δεν συγκρούονται, τότε τα S και S' καλούνται **ισοδύναμα ως προς τις συγκρούσεις** (conflict equivalent)**.**
- Το χρονοδιάγραμμα S καλείται **σειριοποιήσιμο ως προς τις συγκρούσεις** (conflict serializable) αν είναι ισοδύναμο ως προς τις συγκρούσεις με ένα σειριακό χρονοδιάγραμμα.
- Τα εμπορικά συστήματα συνήθως παράγουν σειριοποιήσιμα ως προς τις συγκρούσεις χρονοδιαγράμματα,
  - παρόλο που υπάρχουν και σειριοποιήσιμα χρονοδιαγράμματα άλλου τύπου (δεν είναι όλα τα σειριοποιήσιμα χρονοδιαγράμματα conflict serializable).

Σειριοποίηση σύγκρουσης – παράδειγμα



Σειριοποίηση σύγκρουσης – αντιπαράδειγμα

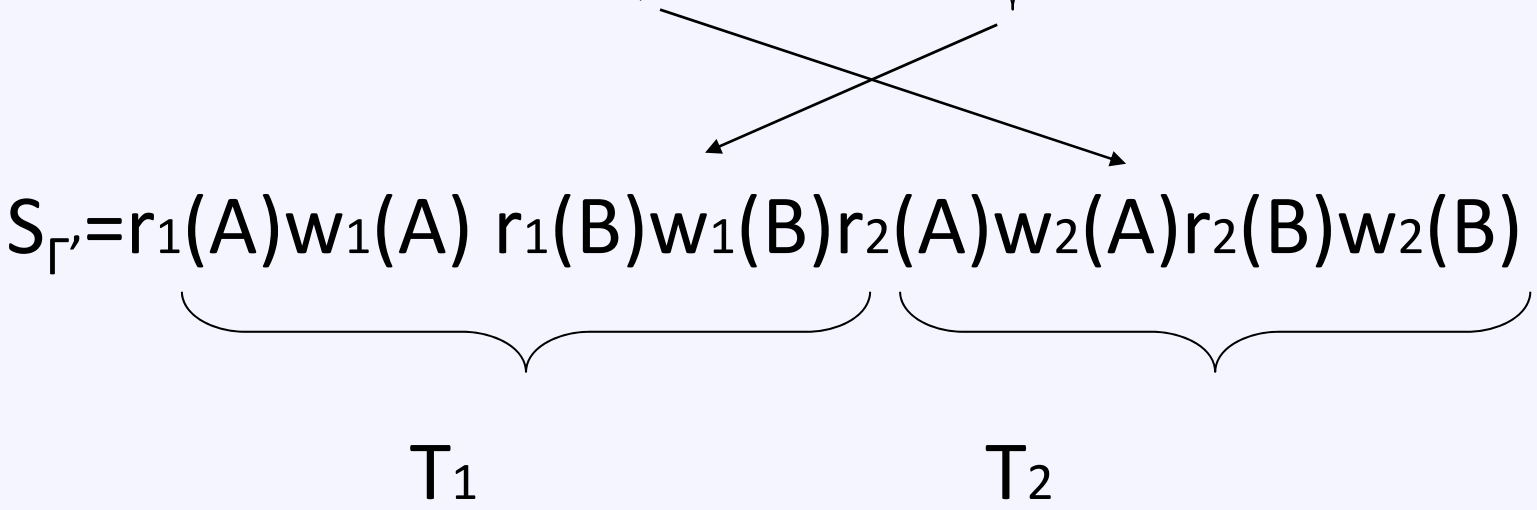
- Τα (β) και (γ) δεν είναι ισοδύναμα ως προς τις συγκρούσεις με το (α).



Επιπλέον παράδειγμα (1)

Χρονοδιάγραμμα Γ –διαφάνεια 10:

$S_Γ$=r1(A)w1(A)r2(A)w2(A)r1(B)w1(B)r2(B)w2(B)



$S_{Γ'}$=r_1(A)w_1(A) r_1(B)w_1(B)r_2(A)w_2(A)r_2(B)w_2(B)

T_1 T_2

Επιπλέον παράδειγμα (2)

Για το χρονοδιάγραμμα Δ:

$S_{\Delta}=r_1(A)w_1(A)r_2(A)w_2(A)\ r_2(B)w_2(B)r_1(B)w_1(B)$

Κανένα από τα δύο ζεύγη δεν μπορεί να μετακινηθεί!

Λόγω του Α: η Τ1 πρέπει να είναι πριν την Τ2: Τ1→ Τ2
Λόγω του Β: η Τ2 πρέπει να είναι πριν την Τ1: Τ2→ Τ1

Κύκλος!

Αντίθετα στο προηγούμενο χρονοδιάγραμμα S_{Γ} ίσχυε μόνο Τ1→Τ2

Γράφοι προήγησης

- Γράφος προήγησης (Precedence graph) P(S), όπου S είναι ένα χρονοδιάγραμμα.

Κόμβοι: συναλλαγές στο S

Ακμές: Ti $\rightarrow$ Tj όταν

- $p_i(A)$, $q_j(A)$ είναι εντολές στο S
- $p_i(A) <_S q_j(A)$ //η $p_i(A)$ προηγείται της $q_j(A)$ στο S
- τουλάχιστον μία από τις p_i, q_j είναι write

Παράδειγμα:

- $w_3(A)r_1(A)w_1(B)r_2(B)\ w_2(C)r_3(C)$
- P: T3→ T1 → T2 → (T3)

Γράφοι προήγησης

Λήμμα: S_1, S_2 conflict equivalent $\Rightarrow$ $P(S_1)=P(S_2)$

<u>Απόδειξη:</u>

Έστω ότι $P(S1) \neq P(S2)$

$\Rightarrow \exists$ Ti: Ti $\rightarrow$ Tj στο S1 αλλά όχι στο S2

$\Rightarrow$ S1 = …pi(A)... qj(A)…

S2 = …qj(A)…pi(A)...

} pi, qj συγκρούονται

$\Rightarrow$ S1, S2 δεν είναι ισοδύναμα ως προς τις συγκρούσεις.

Προσοχή: αν $P(S_1)=P(S_2)$ δεν μπορούμε να πούμε σίγουρα αν τα S1, S2 είναι ισοδύναμα ως προς τις συγκρούσεις:

Π.χ., S1=w1(A) r2(A) w2(B) r1(B)

S2=r2(A) w1(A) r1(B) w2(B)

Θεώρημα

Αν $P(S_1)$ ακυκλικός $\Leftarrow\Rightarrow$

S_1 σειριοποιήσιμο ως προς τις συγκρούσεις.

($\Rightarrow$) Έστω P(S1) ακυκλικός

$\Rightarrow$ S1 ισοδύναμο ως προς τις συγκρούσεις με ένα S που προέρχεται από την τοπολογική ταξινόμηση του P(S1)

$\Rightarrow$ που βάσει ορισμού είναι σειριακό χρονοδιάγραμμα.

($\Leftarrow$) Έστω S1 σειριοποιήσιμο ως προς τις συγκρούσεις

$\Rightarrow \exists$ Sserial: Sserial, S1 ισοδύναμα ως προς τις συγκρούσεις

$\Rightarrow$ P(Sserial) = P(S1)

$\Rightarrow$ P(S1) ακυκλικό γιατί και το P(Sserial) είναι ακυκλικό

Θέματα ταυτοχρονισμού που θα εξεταστούν

- *Σειριοποίηση (προηγούμενες διαφάνειες)*
- **Επιβολή σειριοποίησης με μηχανισμούς κλειδώματος**
- Επιβολή σειριοποίησης με χρονικές σφραγίδες.
- Αδιέξοδα

Μηχανισμοί κλειδώματος

- Χρησιμοποιούνται για την **επιβολή** ενός σειριοποιήσιμου χρονοδιαγράμματος .
- Η **κλειδαριά** (lock) αντιστοιχεί σε ένα τμήμα δεδομένων και είναι μεταβλητή που περιγράφει την κατάσταση του τμήματος των δεδομένων σε σχέση με τις λειτουργίες που ενεργούν στα συγκεκριμένα δεδομένα.

Πρωτόκολλο κλειδώματος

- Ένας μηχανισμός κλειδώματος πρέπει να ικανοποιεί κανόνες εγγύησης της συνέπειας των δεδομένων.
- Οι κανόνες αυτοί ορίζουν ένα **πρωτόκολλο κλειδώματος** (locking protocol) και εγγυώνται ότι το χρονοδιάγραμμα που προκύπτει είναι σειριοποιήσιμο.

Γενικοί κανόνες πρωτοκόλλων κλειδώματος

Κανόνας 1 (επίπεδο συναλλαγής):

- Κάθε συναλλαγή T_i, πριν εκτελέσει οποιαδήποτε πράξη ανάγνωσης ή εγγραφής στα δεδομένα A:
  - της έχει χορηγηθεί η κλειδαριά για τα δεδομένα αυτά, που δηλώνεται με την πράξη $lock_i(A)$ ή $l_i(A)$,
  - δεν έχει απελευθερώσει την κλειδαριά, που δηλώνεται με την πράξη $unlock_i(A)$ ή $u_i(A)$
  - Δηλαδή ισχύει: T_i: … $l_i(A)$ … $p_i(A)$ … $u_i(A)$ ...
  - Επίσης κάθε συναλλαγή που κλειδώνει ένα στοιχείο, πρέπει μετέπειτα να το ξεκλειδώνει.

Κανόνας 2 (επίπεδο χρονοδιαγράμματος):

- Καμία άλλη συναλλαγή T_j, δεν μπορεί επιτυχώς να ζητήσει την κλειδαριά $l_j(A)$ ανάμεσα από τις πράξεις $l_i(A)$ και $u_i(A)$ μιας άλλης συναλλαγής T_i

Παραδείγματα

Έστω τα ακόλουθα τρία χρονοδιαγράμματα. Πληρούνται οι κανόνες;

- l1(A)l1(B)r1(A)w1(B)l2(B)u1(A)u1(B) r2(B)w2(B)u2(B)l3(B)r3(B)u3(B)
  Η πράξη l2(B) παραβιάζει τον κανόνα 2.

- l1(A)r1(A)w1(B)u1(A)u1(B) l2(B) r2(B)w2(B)l3(B)r3(B)u3(B)

  Η πράξη w1(B) παραβιάζει τον κανόνα 1. Η πράξη l3(B) είναι πριν μία πράξη u2(B), που κακώς απουσιάζει (παραβίαση και των δύο κανόνων).

- l1(A)r1(A)u1(A)l1(B)w1(B)u1(B)l2(B) r2(B)w2(B)u2(B)l3(B)r3(B)u3(B)
  Όλα ΟΚ!

Παράδειγμα με κλειδαριές

T1	T2
l1(A); Read(A);	
A ← A+100;Write(A); u1(A)	
	l2(A);Read(A);A ← A×2;
	Write(A); u2(A);
	l2(B);Read(B);B ← B×2;
	Write(B); u2(B);
l1(B);Read(B); B ← B+100;	
Write(B); u1(B);	

A	B
25	25
125	
250	
	50
	150
250	150

Πρωτόκολλο κλειδώματος δύο φάσεων

- **Πρωτόκολλο Κλειδώματος Δύο Φάσεων** (two-phase locking – 2PL):
- Επιπλέον των 2 γενικών κανόνων υπάρχει και 3ος κανόνας, σύμφωνα με τον οποίο κάθε συναλλαγή αποτελείται από 2 φάσεις:
  - **φάση ανάπτυξης** (growing phase), όπου η συναλλαγή ζητά τη χορήγηση κλειδαριών και δεν επιτρέπεται η απελευθέρωση κλειδαριών,
  - **φάση συρρίκνωσης** (shrinking phase), όπου η συναλλαγή απελευθερώνει κλειδαριές και δεν επιτρέπεται να ζητήσει τη χορήγηση νέων κλειδαριών.

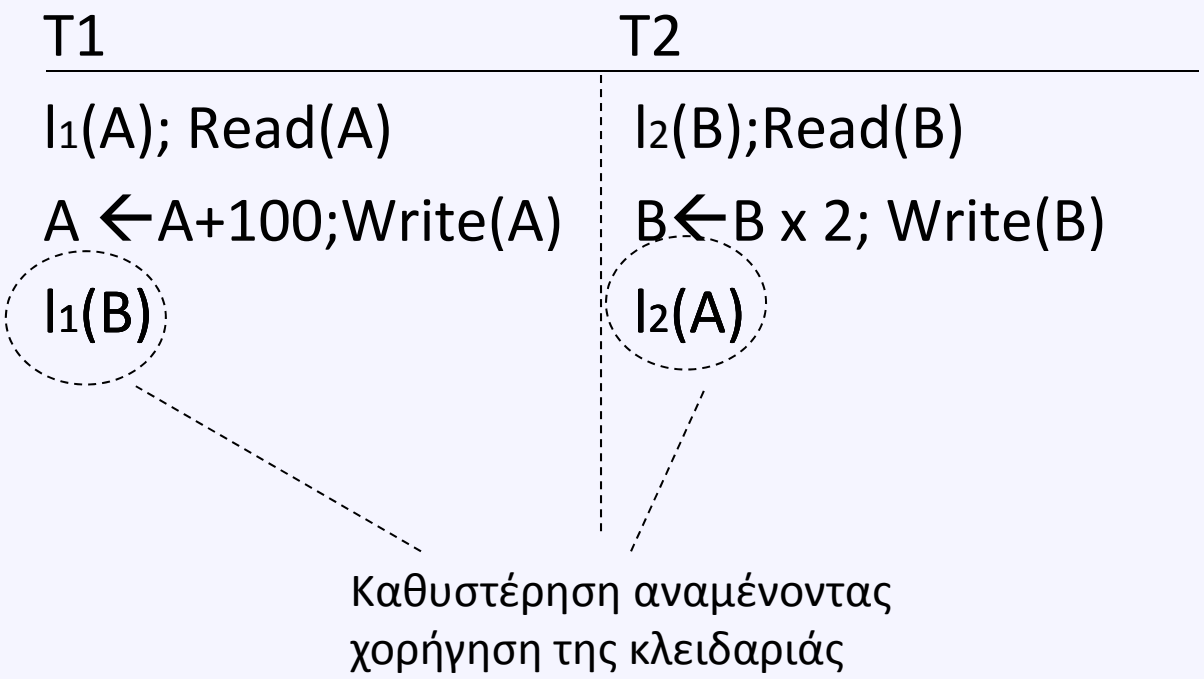
Παράδειγμα με κλειδαριές και 2PL

T1	T2
l_1(A);Read(A)	
A← A+100;Write(A)	
l_1(B); u_1(A)	
	l_2(A);Read(A)
	A ←Ax2;Write(A);l_2(B)
Read(B);B ←B+100	
Write(B); u_1(B)	
	l_2(B); u_2(A);Read(B)
	B ←Bx2;Write(B);u_2(B);

Καθυστέρηση αναμένοντας χορήγηση της κλειδαριάς

Αδιέξοδα

- Έστω οι δύο ίδιες συναλλαγές με το προηγούμενο παράδειγμα, αλλα με αντίστροφη σειρά πράξεων στην Τ2. Ένα πιθανό χρονοδιάγραμμα που οδηγεί σε αδιέξοδο είναι το ακόλουθο:



Ιδιότητες 2PL χρονοδιαγραμμάτων (1)

- Έστω SH(Ti) ο χρόνος που η συναλλαγή Ti αποδεσμεύει για $1^η$ φορά μία κλειδαριά.

Αν Ti $\rightarrow$ Tj στο χρονοδιάγραμμα S $\Rightarrow$ SH(Ti) $<_S$ SH(Tj)

Απόδειξη:

Ti $\rightarrow$ Tj σημαίνει ότι το χρονοδιάγραμμα S έχει την μορφή:

S = … pi(A) … qj(A) …; p,q συγκρούονται

Σύμφωνα με τους κανόνες 1,2:

S = … pi(A) … ui(A) … lj(A) ... qj(A) …

Σύμφωνα με τον Κανόνα 3: SH(Ti) δεν είναι μετά τον χρόνο της ui(A). Η ui(A) είναι πριν την lj(A), που είναι πριν το χρόνο SH(Tj).

Άρα, SH(Ti) $<_S$ SH(Tj)

Ιδιότητες 2PL χρονοδιαγραμμάτων (2)

Θεώρημα: Τα 2PL χρονοδιαγράμματα είναι σειριοποιήσιμα ως προς τις συγκρούσεις.

Απόδειξη:

(1) Έστω ότι ο P(S) έχει κύκλο

$$T_1 \rightarrow T_2 \rightarrow \ldots T_n \rightarrow T_1$$

(2) Τότε: $SH(T_1) < SH(T_2) < \ldots < SH(T_1)$

(3) Αυτό είναι αδύνατο, άρα ο P(S) είναι ακυκλικός.

(4) $\Rightarrow$ S είναι σειριοποιήσιμο ως προς τις συγκρούσεις.

Περιορισμοί

S1: w1(x) w3(x) w2(y) w1(y)

- Το S1 είναι σειριοποιήσιμο (T2, T1, T3).
- Αλλά δεν μπορεί να υλοποιηθεί με το 2PL πρωτόκολλο.

- Τι συμβαίνει με τα χρονοδιαγράμματα:

w1(A) w2(A) w1(B) w2(B)

w1(A) w2(A) w2(B) w1(B)