Προχωρημένα Θέματα Τεχνολογίας και Εφαρμογών Βάσεων Δεδομένων

Βασικές Αρχές Διαχείρισης Ταυτοχρονισμού

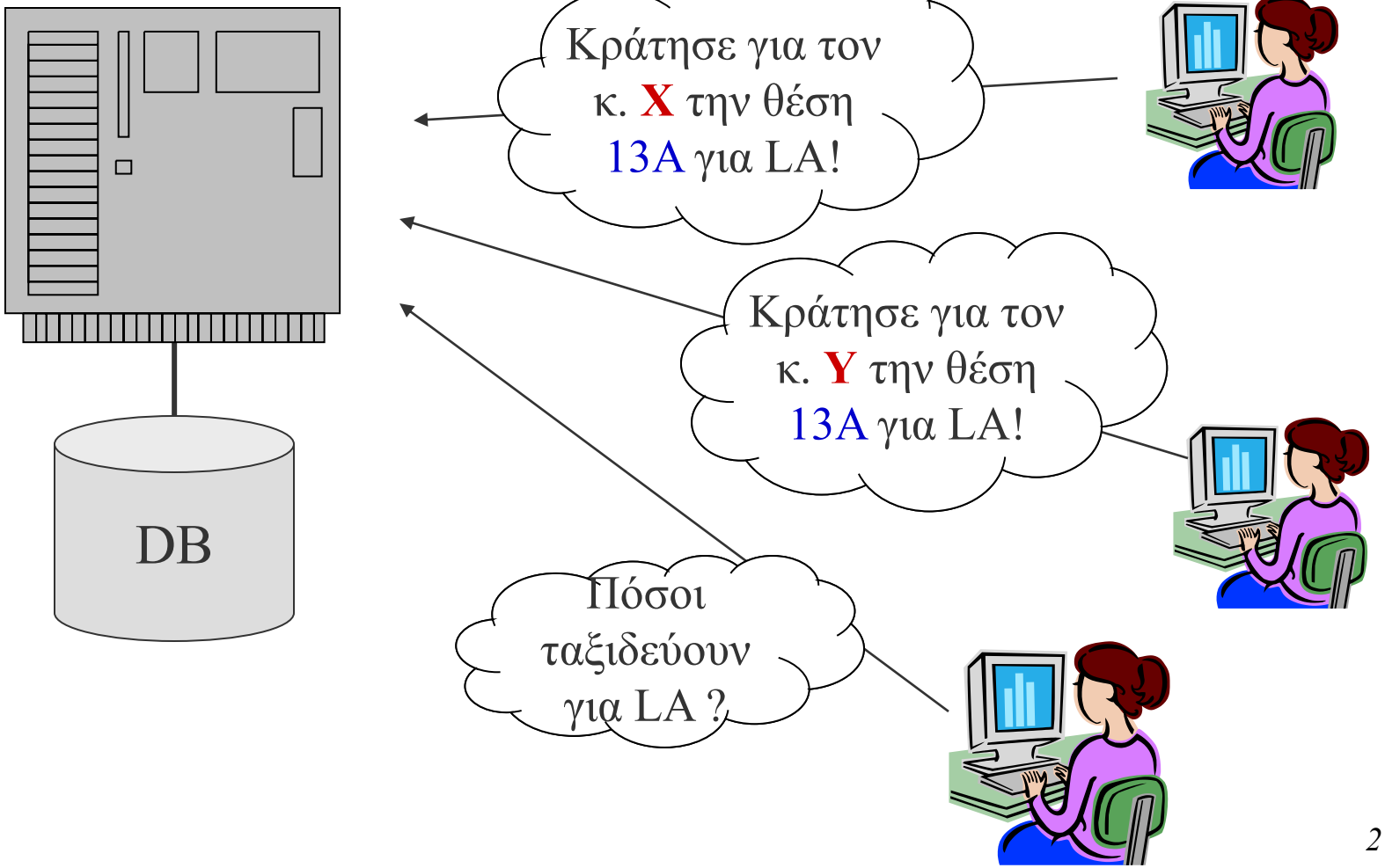
Πάνος Βασιλειάδης

pvassil@cs.uoi.gr

Μάρτιος 2020

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

Συναλλαγές & Ταυτοχρονισμός



Θεματολόγιο

- Κλειδώματα – 2 Phase Locking
- Πώς γίνεται στην πράξη?
- Αδιέξοδα
- SQL & συναλλαγές

Θεματολόγιο

- Κλειδώματα – 2 Phase Locking
- Πώς γίνεται στην πράξη?
- Αδιέξοδα
- SQL & συναλλαγές

Πώς ελέγχουμε σειριοποιησιμότητα στην πράξη?

- Η σειριοποιησιμότητα όψεων είναι πολύ ακριβή για να ελεγχθεί, ούτως ή άλλως...
- Οι αλγόριθμοι για τον έλεγχο σειριοποιησιμότητας συγκρούσεων κοστίζουν n^2 [ενίοτε και καλύτερα] για n συναλλαγές.
- Μόνο που ο έλεγχος αυτός γίνεται αφού φτιαχτεί το χρονοπρόγραμμα, ήτοι γίνεται λίγο αργά ...

- Γι' αυτό και έχουμε αναπτύξει on-line πρωτόκολλα ελέγχου του ταυτοχρονισμού των συναλλαγών.

Κλειδώματα και πρωτόκολλα

- **Πρωτόκολλο κλειδώματος** είναι ένα συγκεκριμένο είδος πρωτοκόλλων, που διασφαλίζουν την σειριοποιησιμότητα μιας σειράς ενεργειών του DBMS, στη βάση κανόνων για το τι μπορεί [ή δεν μπορεί] να προσπελάσει μια συναλλαγή.
- Η βασική έννοια γύρω από την οποία οργανώνεται ένα τέτοιο πρωτόκολλο, είναι η έννοια του «**κλειδώματος**» (ή «κλειδιού», ή «κλειδαριάς» ...)

Στην αγγλική lock [πολύ σπάνια, και key].

Κλείδωμα

- **Κλείδωμα** είναι μια μεταβλητή που σχετίζεται με ένα στοιχειώδες δεδομένο και περιγράφει την κατάστασή του, σε σχέση με πιθανές πράξεις που μπορούν να εφαρμοσθούν σε αυτό.

Read@T1

Written@T2

EMP

ID	Name
12	XX
13	YY
14	ZZ

Elmasri & Navathe

2 Phase Locking – 2PL

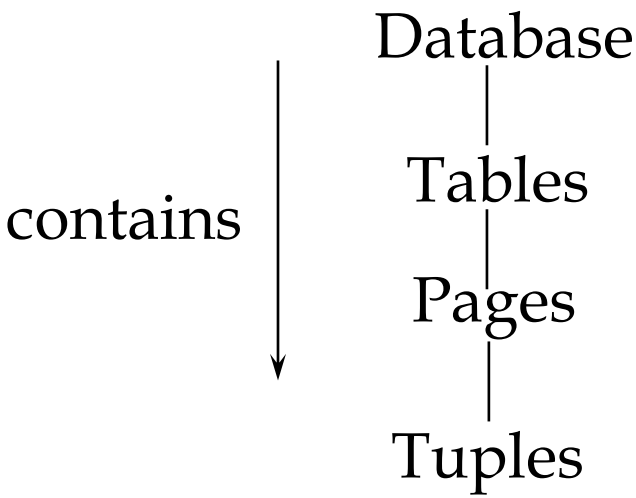
- Ο βασικός αλγόριθμος διαχείρισης του ταυτοχρονισμού συναλλαγών είναι ο «Αλγόριθμος Κλειδώματος Δύο Φάσεων» -- στην αγγλική “2 Phase Locking” ή **2PL**

2PL

- Εξασφαλίζει τη σειριοποιησιμότητα επιτρέποντας στις συναλλαγές να προσπελάσουν αντικείμενα, αν καταφέρουν να αποκτήσουν κάποιο κλείδωμα γι' αυτά.
- Δύο είδη κλειδώματος:
  - **Ανάγνωσης**: λαμβάνεις κλείδωμα για να διαβάσεις ένα αντικείμενο
  - **Εγγραφής**: λαμβάνεις κλείδωμα για να γράψεις ή/και για να διαβάσεις ένα αντικείμενο

Τι είναι «αντικείμενο»?

- Μπορούμε να κλειδώσουμε τη βάση σε πολλά «επίπεδα»
- Θεωρήστε προς το παρόν εγγραφές (στα πιο πολλά DBMS's πλέον)
  - Εσωτερικά, ενίοτε μιλάμε και για σελίδες
  - Πολλές φορές κλειδώνονται και ολόκληροι πίνακες



Ramakrishnan & Gehrke

Πρωτόλειος αλγόριθμος κλειδώματος ≠ 2PL

- Κάθε ενέργεια διαβάσματος/γραψίματος απαιτεί να (περιμένεις μέχρι να) έχεις πάρει το σωστό κλείδωμα
- Για να διαβάσεις, πρέπει να πάρεις ένα read lock – το οποίο και θα αποκαλούμε **Shared lock** (**S-lock**). Είναι δυνατόν να υπάρχουν πολλές συναλλαγές με S-lock για το ίδιο αντικείμενο
- Για να γράψεις, πρέπει να πάρεις ένα write lock – το οποίο και θα αποκαλούμε **eXclusive lock** (X-lock). Αν μια συναλλαγή έχει X-lock για ένα αντικείμενο, απαγορεύεται οποιαδήποτε άλλη να έχει κάποιο κλείδωμα για το αντικείμενο αυτό
- Μπορείς να ξεκλειδώνεις (**Unlock**) αντικείμενα, αν δεν τα χρειάζεσαι.

Συμβολισμός

- $R_T(A)$: η συναλλαγή T διαβάζει το αντικείμενο A
- $W_T(A)$: η συναλλαγή T γράφει το αντικείμενο A
- **$S_T(A)$** : η συναλλαγή T παίρνει S-lock για το αντικείμενο A
- **$X_T(A)$** : η συναλλαγή T παίρνει X-lock για το αντικείμενο A
- **$U_T(A)$** : η συναλλαγή T ξεκλειδώνει το αντικείμενο A
- $COMMIT_T$: η συναλλαγή T τερματίζει επιτυχώς
- $ABORT_T$: η συναλλαγή T αποτυγχάνει

Παράδειγμα

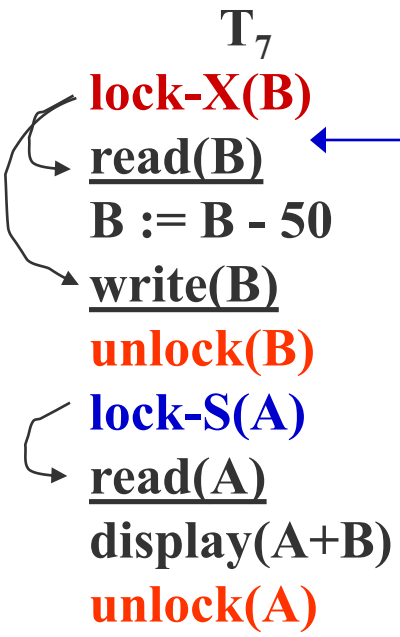
T_7

lock-X(B)
read(B)
B := B - 50
write(B)
unlock(B)
lock-S(A)
read(A)
display(A+B)
unlock(A).

$R_7(B)$; $W_7(B)$; $R_7(A)$.

$X_7(B)$; $R_7(B)$; $W_7(B)$; $U_7(B)$; $S_7(A)$; $R_7(A)$ $U_7(A)$

Παράδειγμα



Βασική υπόθεση: ξέρουμε από πριν όλες τις ενέργειες της συναλλαγής ...

Συγκρούσεις

- Έστω δύο συναλλαγές, T1 και T2, εκ των οποίων, η T1 ζητά ένα κλείδωμα και η T2 έχει ήδη ένα κλειδωμα για το ίδιο αντικείμενο.
- Ο πίνακας συγκρούσεων πρέπει να σας θυμίζει κάτι...
- **Άρνηση ⇒ «περίμενε μέχρι να γίνει unlock»**

		Lock held	
	T1\T2	Read	Write
Lock requested	Read	☑	☒
	Write	☒	☒

2 ειδών προβλήματα

- Αν προσπαθήσεις να αυξήσεις τις ταυτόχρονες συναλλαγές, ξεκλειδώνοντας αντικείμενα πολύ νωρίς, μπορεί να έχεις προβλήματα ασυνέπειας
- Αν προσπαθήσεις να καθυστερήσεις το ξεκλείδωμα,
  - μπορεί να προκύψουν **αδιέξοδα** (όπου δύο συναλλαγές περιμένουν –για πάντα– η μία την άλλη για το ξεκλείδωμα διαφορετικών αντικειμένων)
  - καθυστερείς (ενώ έχουμε πιο έξυπνους τρόπους)...

Στη συνέχεια, παραδείγματα από Korth, Silberschatz & Sudarshan…

Ξεκλειδώνοντας νωρίς...

Επειδή η T7 ξεκλειδώνει το B νωρίς, το αποτέλεσμα της T8 είναι λάθος...

T_7	T_8
lock-X(B)	
read(B)	
B := B - 50	
write(B)	
unlock(B)	
	lock-S(A)
	read(A)
	unlock(A)
	lock-S(B)
	read(B)
	unlock(B)
	display(A + B)
lock-X(A)	
read(A)	
A := A + 50	
write(A)	
unlock(A)	

Deadlock

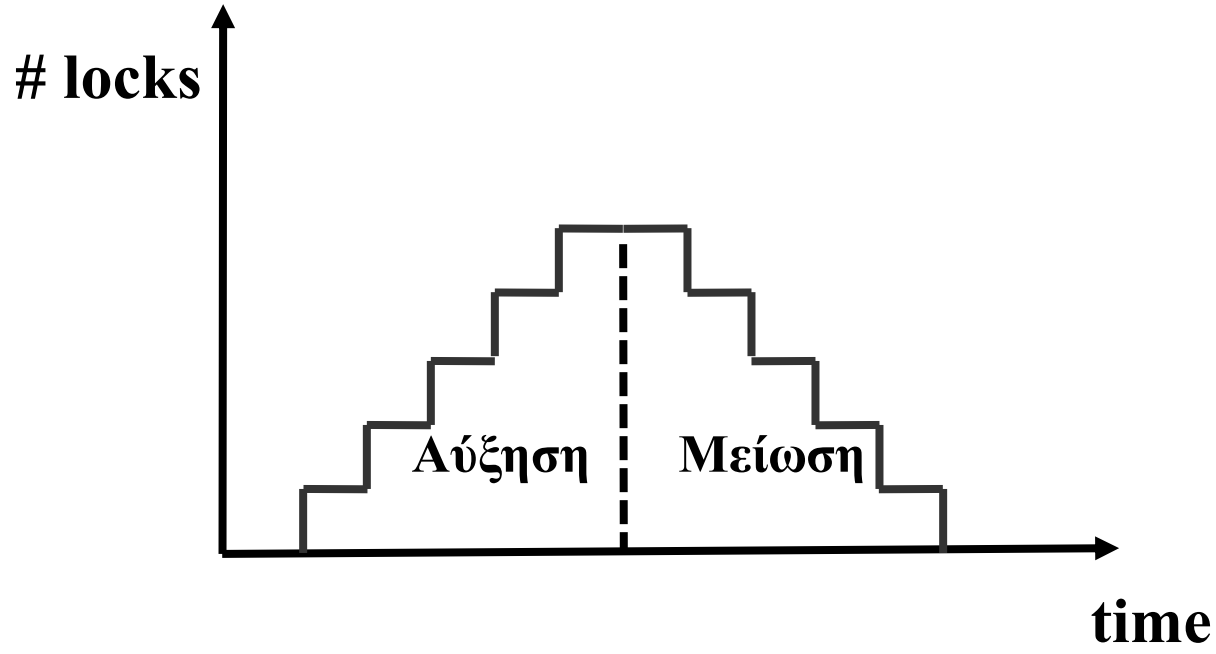
Η μία συναλλαγή έχει κλειδώσει το αντικείμενο που χρειάζεται η άλλη. Σταματούν και οι δύο, αναμένοντας η μία την άλλη ...

T_9	T_{10}
lock-X(B)	
read(B)	
B := B - 50	
write(B)	
	lock-S(A)
	read(A)
	lock-S(B)
lock-X(A)	
read(A)	read(B)
A := A + 50	unlock(A)
write(A)	unlock(B)
unlock(B)	display(A + B)
unlock(A)	

2PL

- **Ζήτα κλείδωμα πριν πράξεις:**
  - Για να διαβάσεις, πρέπει να πάρεις ένα **Shared lock** (**S-lock**). Είναι δυνατόν να υπάρχουν πολλές συναλλαγές με S-lock για το ίδιο αντικείμενο
  - Για να γράψεις, πρέπει να πάρεις ένα **eXclusive lock** (X-lock). Αν μια συναλλαγή έχει X-lock για ένα αντικείμενο, απαγορεύεται οποιαδήποτε άλλη να έχει κάποιο κλείδωμα για το αντικείμενο αυτό
- **Δεν μπορείς να ξεκλειδώσεις αντικείμενο μέχρι να έχεις πάρει και το τελευταίο κλείδωμα που χρειάζεσαι**

2 Phase Locking



Ξανά: έτσι και ξεκλειδώσεις αντικείμενο, δεν μπορείς να πάρεις άλλο κλείδωμα!!!

2PL?

T_9	T_{10}	T_7	T_8
lock-X(B)	lock-S(A)	lock-X(B)	lock-S(A)
read(B)	read(A)	read(B)	read(A)
B := B - 50	lock-S(B)	B := B - 50	unlock(A)
write(B)	read(B)	write(B)	lock-S(B)
lock-X(A)	unlock(A)	unlock(B)	read(B)
read(A)	unlock(B)	lock-X(A)	unlock(B)
A := A + 50	display(A + B)	read(A)	display(A + B)
write(A)		A := A + 50	
unlock(B)		write(A)	NOT LEGAL
unlock(A)		unlock(A)	
LEGAL	LEGAL	NOT LEGAL	

KEY

GROWING

SHRINKING

VIOLATION

Απλοϊκή περίπτωση που κάθε schedule έχει μόνο μία συναλλαγή

Διαχειριστής κλειδαριών

- Το τμήμα του DBMS που διαχειρίζεται τα κλειδώματα
- Αν το DBMS υποστηρίζει 2PL → **2PL scheduler**
- **Πίνακας κλειδαριών**:
  - ObjectID
    - TransactionID, LockType
    - Queue με συναλλαγές εν αναμονή

	x	y	z
T1		S	
T2		S	
T3		W	

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

Μέθοδος:

- *Ας δοκιμάσω να βάλω locks & unlocks…*
- *Έχω μη επιτρεπτές συγκρούσεις?*
- *Έχουν όλες οι συναλλαγές τις 2 φάσεις?*

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

	x	y	z
T1			
T2			
T3			

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

	x	y	z
T1		X	
T2			
T3			

X1[y];**R1[y]**;

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

	x	y	z
T1		X	
T2	S		
T3			

X1[y];**R1[y]**;S2[x];**R2[x]**;

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

Έχω ήδη το lock…

	x	y	z
T1		X	
T2	S		
T3			

X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

Unlock κάνω?
Το T3 θέλει το y

ΟΧΙ!!

	x	y	z
T1		X	
T2	S		
T3			

X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

Και τι κάνω?

Ελέγχω το ενδεχόμενο, proactively, να έχει πάρει το lock W1[z] (ξέρουμε το μέλλον) και έχοντας πλέον ΟΛΑ τα locks που θα χρειαστεί, η T1 να προβαίνει σε unlock…

Αυτό ΕΙΝΑΙ συμβατό με το πρωτόκολλο του 2PL

X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;X1[z];U1[y];

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

	x	y	z
T1			X
T2	S		
T3		X	

X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;X1[z];U1[y];X3[y];**W3[y]**;

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

	x	y	z
T1			X
T2	S		
T3		X	

Έχω ήδη το lock…

X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;X1[z];U1[y];X3[y];**W3[y]**;**W1[z]**;

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

	x	y	z
T1			
T2	S		
T3		X	

X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;X1[z];U1[y];X3[y];**W3[y]**;**W1[z]**;U1[z];

← Unlock ASAP

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

	x	y	z
T1			
T2	S		S
T3		X	

X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;X1[z];U1[y];X3[y];**W3[y]**;**W1[z]**;U1[z];S2[z];**R2[z]**;

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

ΔΕΝ υπάρχει σύγκρουση για S-locks!!

	x	y	z
T1			
T2	S		S
T3		X	S

X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;X1[z];U1[y];X3[y];**W3[y]**;**W1[z]**;U1[z];S2[z];**R2[z]**;S3[z];**R3[z]**;

… ναι …

S2:

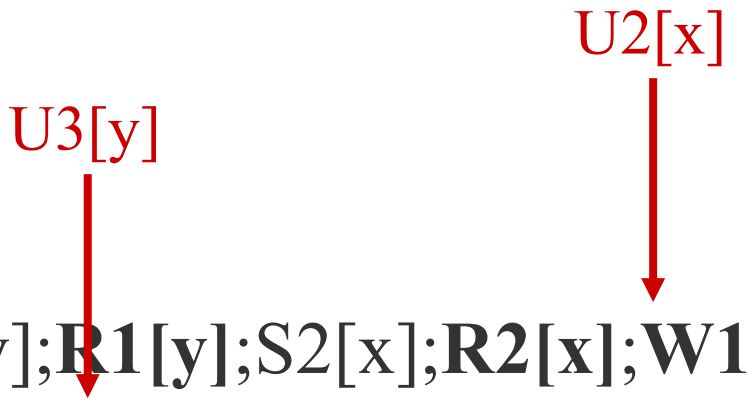
R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;X1[z];U1[y];X3[y];**W3[y]**;**W1[z]**;U1[z];S2[z];**R2[z]**;S3[z];**R3[z]**;U_{123}[all] …

Μπορούσα να είχα κάνει ?

S2:

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].



X1[y];**R1[y]**;S2[x];**R2[x]**;**W1[y]**;X1[z];U1[y];X3[y];**W3[y]**;**W1[z]**;U1[z];S2[z];**R2[z]**;S3[z];**R3[z]**;U_{123}[all] …

Θεώρημα

- Όλα τα χρονοπρογράμματα που προκύπτουν από ένα διαχειριστή κλειδαριών 2PL, έχουν ακυκλικό γράφο σειριοποίησης
- Το αντίστροφο ΔΕΝ ισχύει.

- Λήμμα: αν δεν έχει ακυκλικό γράφο σειριοποήσης ➔ δεν είναι 2PL.

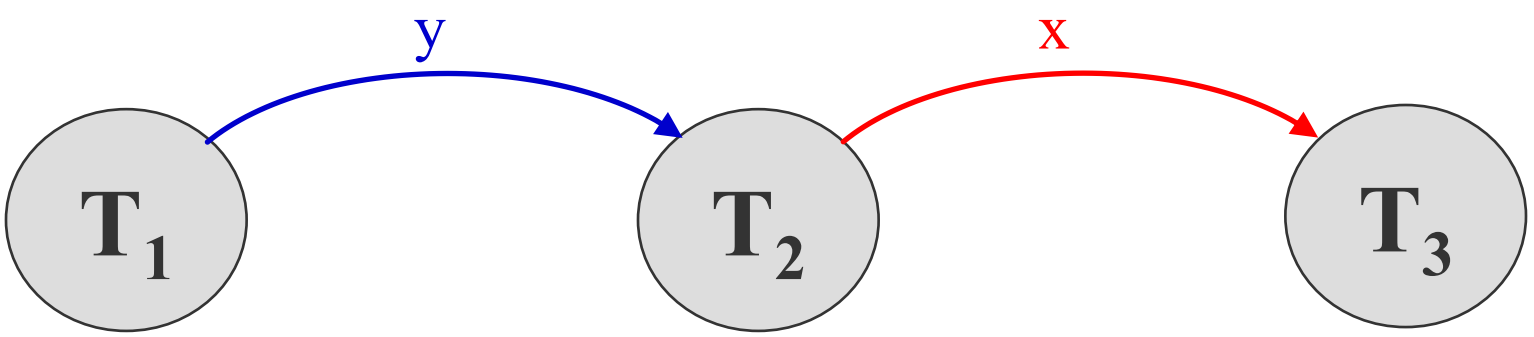
Μπορώ να σειριοποιήσω το παρόν χρονοπρόγραμμα?

- **S3**: W2[x];W3[x];W1[y];W2[y].

T1: W1[y]
T2: W2[x]W2[y]
T3: W3[x]

Μπορώ να σειριοποιήσω το παρόν χρονοπρόγραμμα?

- **S3**: W2[x];W3[x];W1[y];W2[y].



Άκυκλο γράφημα ⇔ Σειριοποιήσιμο χρονοπρόγραμμα

Προκύπτει το παρακάτω χρονοπρόγραμμα από αλγόριθμο 2PL ?

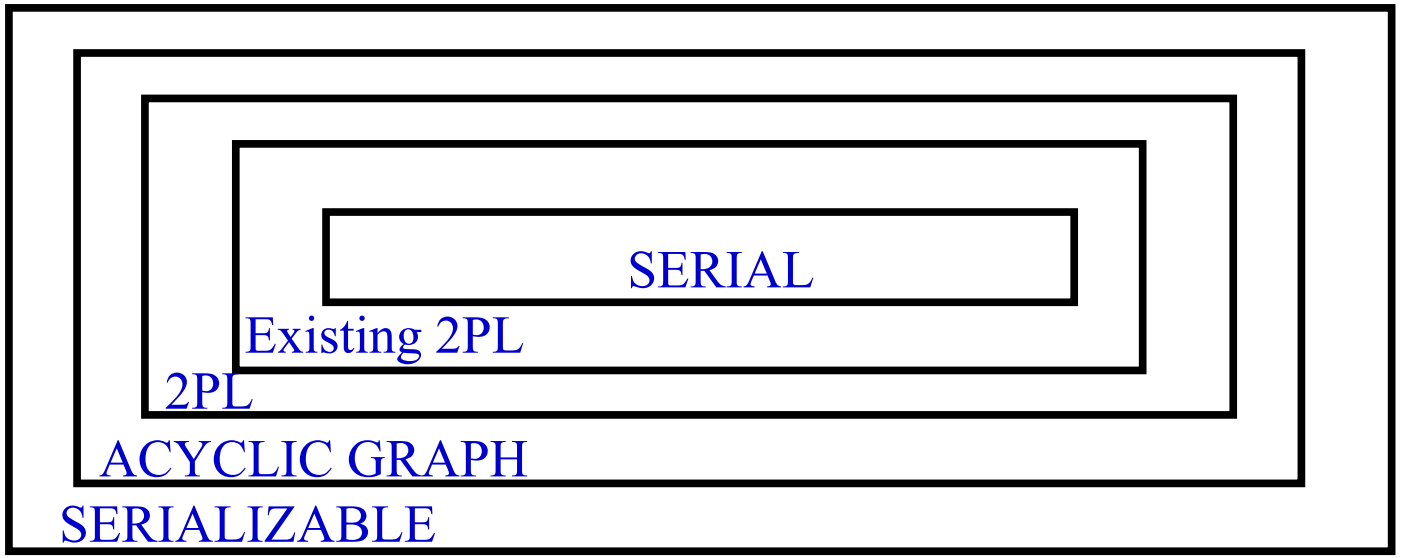
S3: W2[x];W3[x];W1[y];W2[y].

T1: W1[y]
T2: W2[x]W2[y]
T3: W3[x]

Ας βάλουμε κλειδαριές:
X2[x]W2[x] X2[y] U2[x] X3[x]W3[x]
?? U2[x][X3[x]W3[x] ??
??? U2[y]X1[y]W1[y] ??
ΟΧΙ! Διότι έπεται W2[y]

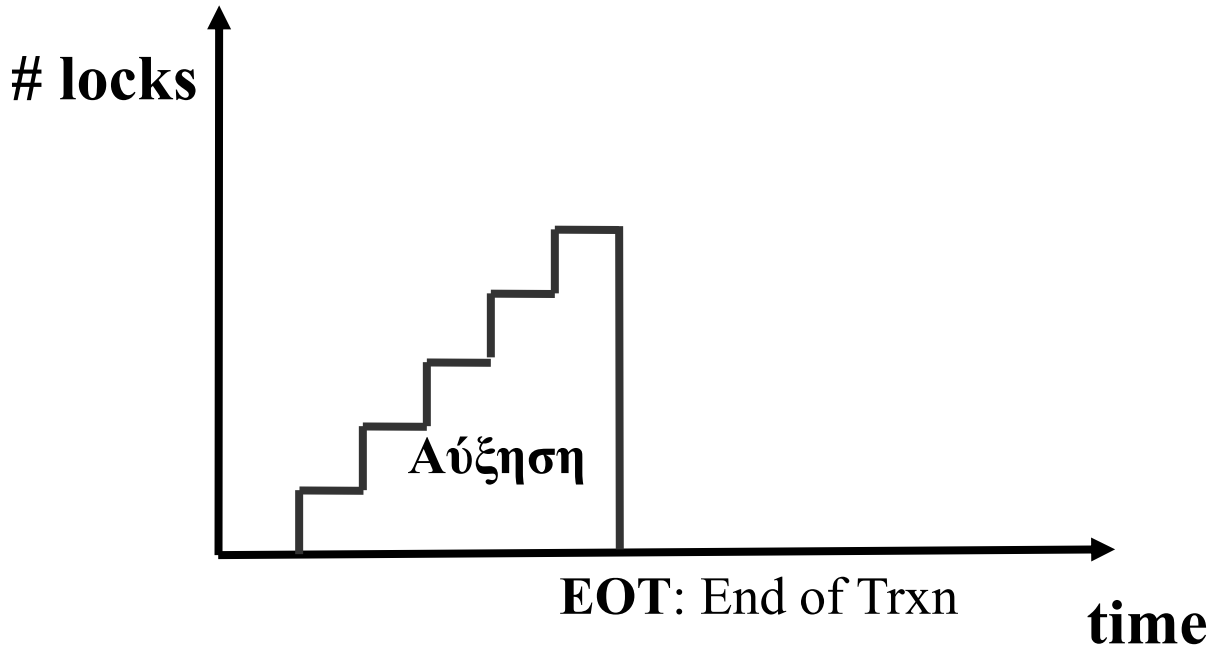
	x	y
T1		
T2		X
T3	X	

Venn διάγραμμα



Υπάρχων 2PL?

- **Αυστηρός 2PL**: όλα τα κλειδώματα αφήνονται στο τέλος της συναλλαγής, μαζί.



Αυστηρός 2PL

- Στην πράξη όλα τα συστήματα χρησιμοποιούν αυστηρό 2PL...

[Για το σπίτι] Προκύπτει από Strict 2PL?

R1[y];R2[x];W1[y];W3[y];W1[z];R2[z];R3[z].

Θεματολόγιο

- Κλειδώματα – 2 Phase Locking
- Πώς γίνεται στην πράξη?
- Αδιέξοδα
- SQL & συναλλαγές

Πώς γίνεται στην πράξη?

- Τα κλειδώματα και την πρόσβαση στον πίνακα κλειδαριών, τα διαχειρίζεται ο **διαχειριστής κλειδωμάτων** (**lock manager**)
- Κάθε συναλλαγή ζητά από το διαχειριστή κλειδωμάτων το αντίστοιχο κλειδί
- Αυτός ελέγχει τον πίνακα κλειδωμάτων και αν μπορεί, παραχωρεί το κλειδί και ενημερώνει τον πίνακα
- Αν δεν μπορεί, χρησιμοποιεί μια ουρά αναμονής ανά αντικείμενο, όπου και καταχωρείται η εν λόγω συναλλαγή. Η συναλλαγή περιμένει.

Πώς γίνεται στην πράξη?

- Όταν μια συναλλαγή τελειώνει [επιτυχώς, ή μη], όλα τα κλειδώματα αποδεσμεύονται.
- Όσες συναλλαγές περιμένουν στην ουρά αναμονής για κάθε κλείδωμα, ενεργοποιούνται και λαμβάνουν τα κλειδώματα με τη σειρά

- Π.χ., αν η ουρά είναι `S1[x]S2[x]X3[x]` για το x, όταν θα ενεργοποιηθεί η ουρά, η T1 και T2 θα πάρουν το S-lock και η ουρά θα γίνει `X3[x]`

Λιμοκτονία -- starvation

- Έστω ότι η T1 έχει S-lock στο x
- Έστω ότι η T2 ζητά X-lock στο x ➔ θα μπει στην ουρά αναμονής `Q:X2[x]`
- Έστω ότι, ακολούθως, έρχεται η T3 και ζητά S-lock στο x. Θα το λάβει?

- ΟΧΙ! Αν ακολουθήσουμε αυτή την πολιτική, είναι πιθανό η T2 να περιμένει για πάντα...
- Η T3 θα μπει στην ουρά `Q:X2[x]S3[x]`

Ατομικότητα του κλειδώματος

- Για να διασφαλίζεται η ατομικότητα του κλειδώματος και ξεκλειδώματος:
  - χρησιμοποιείται σηματοφορέας στην μνήμη για τον πίνακα κλειδωμάτων
  - χρησιμοποιούνται μικρής διάρκειας **μάνταλα** (latches) για τις σελίδες που διαβάζονται ή ενημερώνονται, αν το κλείδωμα γίνεται ανά εγγραφή (ουσιαστικά κλειδώνοντας την αντίστοιχη I/O ενέργεια read/write, για να μην υπάρχει σύγκρουση σε επίπεδο σελίδας)

Αναβάθμιση κλειδώματος

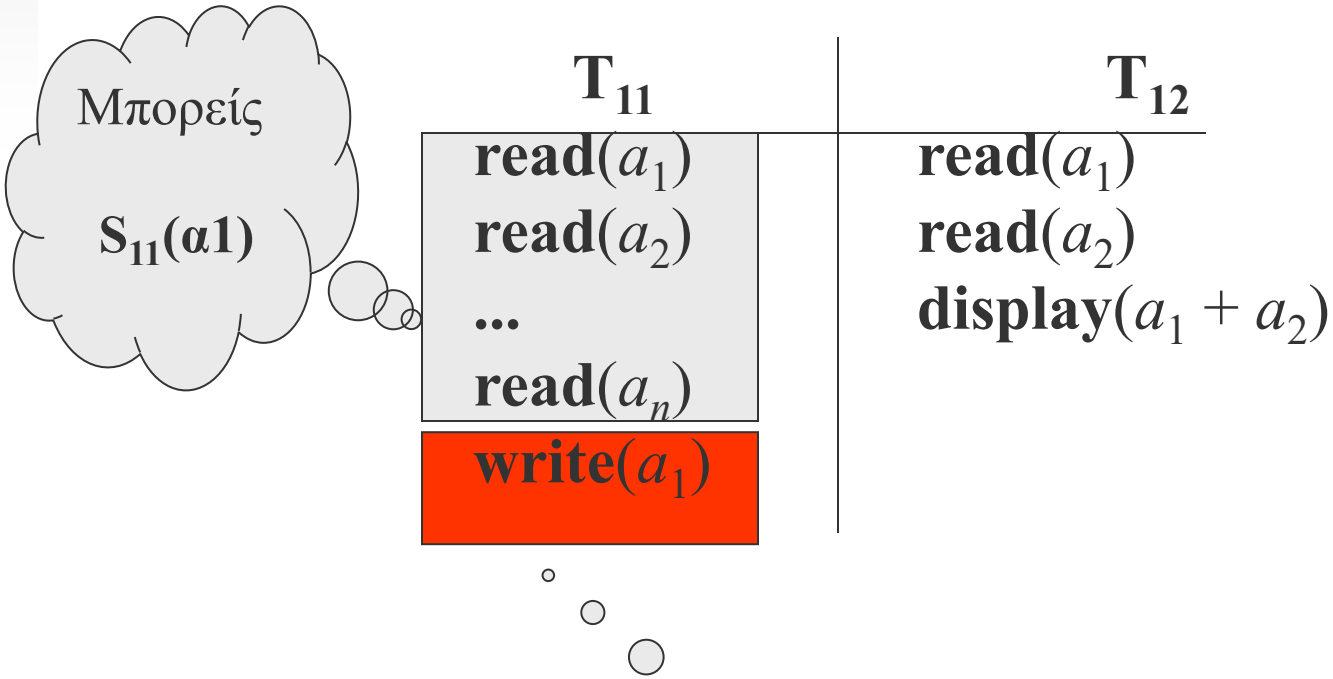
- Μέχρι τώρα υποθέταμε ότι κάθε συναλλαγή γνωρίζει εκ προοιμίου όλα τα κλειδώματα που θα ζητήσει.
- Π.χ. T: **R(A) R(B) W(A)** θα κλειδώσει με **X** το A στην αρχή, λόγω του **W(A)** ➔

  T: $\mathbf{X}_T$(A) **R(A)** S_T(B) **R(B) W(A)** U(A) U(B) .
- Τι γίνεται όμως αν δεν τα γνωρίζει?
- Η λύση έγκειται στην **αναβάθμιση κλειδώματος (αντίστοιχα, υποβάθμιση)**

Κανόνες αναβάθμισης

- Αναβαθμίσεις επιτρέπονται μόνο στη φάση αύξησης
- Υποβαθμίσεις επιτρέπονται μόνο στη φάση μείωσης
- Αν υπάρχει και άλλη συναλλαγή που έχει S-lock ένα αντικείμενο, και θες να αναβαθμίσεις από S- σε X-lock, πρέπει να περιμένεις...

Αναβάθμιση



Upgrade από $S_{11}(\alpha 1)$ σε $X_{11}(\alpha 1)$ προϋποθέτει να κάνει UNLOCK η T_{12} το α1

Θεματολόγιο

- Κλειδώματα – 2 Phase Locking
- Πώς γίνεται στην πράξη?
- Αδιέξοδα
- SQL & συναλλαγές

Αδιέξοδο

- Η κατάσταση κατά την οποία δύο συναλλαγές Τ και Τ’ αναμένουν η μία την άλλη για την απελευθέρωση κάποιων κλειδωμάτων
- Εναλλακτικοί όροι: «Λειτουργική παύση» ή “deadlock”
- Προφανώς, ο παραπάνω ορισμός γενικεύεται για περισσότερες από δύο συναλλαγές

Αδιέξοδο

- Θεωρήστε το παρακάτω χρονοπρόγραμμα

S1[x],R1[x],X2[y],W2[y],X2[x],W2[x],X1[y],W1[y]

- Η T1 έχει S-lock στο x
- Η T2 έχει X-lock στο y. Όταν ζητά X-lock στο x, υποχρεωτικά περιμένει την T1
- Η T1 πάει να ζητήσει X-lock στο y, οπότε υποχρεωτικά περιμένει την T2 ...

Γράφος αναμονής (blocking graph)

- Για κάθε συναλλαγή και ένας κόμβος
- Μία κατευθυνόμενη ακμή από την συναλλαγή T1 στην συναλλαγή T2, αν η T1 **περιμένει** την T2 να απελευθερώσει κάποιο κλείδωμα

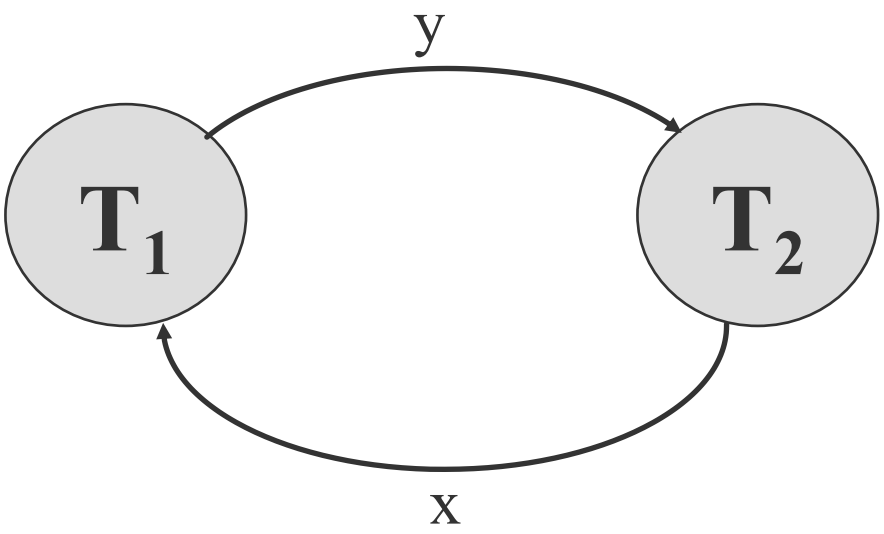
- Επίσης γνωστός και ως γράφος **waits-for**

ΠΡΟΣΟΧΗ: Δεν είναι ίδιος με το γράφο σειριοποιησιμότητας!!!

Αδιέξοδο

- Θεωρήστε το παρακάτω χρονοπρόγραμμα

S1[x],R1[x],X2[y],W2[y],X2[x],W2[x],X1[y],W1[y]



Θεώρημα

- **Θεώρημα**: Υπάρχει αδιέξοδο αν και μόνο αν ο γράφος αναμονής έχει κύκλο

- Στην πράξη το σύστημα ελέγχει τον γράφο αναμονής περιοδικά
- Εναλλακτικά, αντί για γράφο, μπορεί να μετρά πόση ώρα μια συναλλαγή αναμένει για ένα κλείδωμα και να την τερματίζει αν περάσει κάποιο όριο...

Ανίχνευση ή αποτροπή?

- Αντί να περιμένουμε να συμβεί το αδιέξοδο, μπορούμε να το αποτρέψουμε προληπτικά

- **Συντηρητικός 2PL**: μια συναλλαγή αποκτά όλα τα κλειδώματα που χρειάζεται στο ξεκίνημα της. Αν δεν μπορεί να τα πάρει ΟΛΑ, δεν ξεκινά, αλλά αναμένει!

Εν γένει, προτιμώνται τα αδιέξοδα...

Ποια συναλλαγή θα «τερματιστεί»?

- Αυτή με τα λιγότερα κλειδώματα ...
- Αυτή με τα περισσότερα κλειδώματα ...
- Αυτή που είναι η πιο πρόσφατη ...
- Αυτή που εκτιμάται ότι χρειάζεται την πιο πολλή δουλειά...

Για το σπίτι: τι προβλήματα μπορεί να έχει η χρήση timeouts?

Θεματολόγιο

- Κλειδώματα – 2 Phase Locking
- Πώς γίνεται στην πράξη?
- Αδιέξοδα
- SQL & συναλλαγές

Συναλλαγές και SQL

- Στην SQL-92 κάθε συναλλαγή έχει:
  - Μέθοδο πρόσβασης: **`READ ONLY`** vs. **`READ WRITE`**
  - Ένα από 4 επιτρεπτά επίπεδα απομόνωσης:
    - **`READ UNCOMMITTED`**
    - **`READ COMMITTED`**
    - **`REPEATABLE READ`**
    - **`SERIALIZABLE`**
- Παράδειγμα χρήσης:

```
SET ISOLATION LEVEL SERIALIZABLE READ WRITE
```

Συναλλαγές και SQL

- **READ UNCOMMITTED**:
  - Δεν υποστηρίζει κλειδώματα.
  - Επιτρέπεται μόνο σε READ ONLY συναλλαγές
- **READ COMMITTED**:
  - Κρατά X-locks για writes, και τα ελευθερώνει στο EOT
  - Κρατά S-locks για reads, και τα ελευθερώνει αμέσως
  - Δεν έχει πρόβλημα με dirty reads (W-R συγκρούσεις), αλλά έχει πρόβλημα σε unrepeatable reads (R-W συγκρούσεις)

Συναλλαγές και SQL

- **`REPEATABLE READ`**:
  - Διαβάζει μόνο ότι προκύπτει από committed συναλλαγές
  - Ελευθερώνει locks στο τέλος (S-locks & X-locks)
  - Ευάλωτη σε φαντάσματα
- **`SERIALIZABLE`**:
  - Αυστηρό 2PL
  - Ανεπηρέαστη από φαντάσματα

SQL-92

Επίπεδο	Dirty Read	Μη επαναλαμ-βανόμενη ανάγνωση	Πρόβλημα φαντάσματος
Read Uncommitted	Ίσως	Ίσως	Ίσως
Read Committed	Όχι	Ίσως	Ίσως
Repeatable Reads	Όχι	Όχι	Ίσως
Serializable	Όχι	Όχι	Όχι