

## **CHƯƠNG 2**

# **GIAO TÁC — ĐIỀU KHIỂN ĐỒNG THỜI**

# Nội dung

1. **Dẫn nhập**
2. **Giao tác**
3. **Giao tác truy xuất đồng thời**
4. **Lịch thao tác**
5. **Điều khiển đồng thời dùng kỹ thuật khóa**
6. **Mức cô lập của giao tác**
7. **Deadlock**
8. **Cách sử dụng các phương thức khóa**
9. **Điều khiển đồng thời dùng kỹ thuật nhãn thời gian**
10. **Điều khiển đồng thời dùng phương pháp kiểm tra hợp lệ**

# 1. DẪN NHẬP

- Vì sao phải thực hiện giao tác?
- Vì sao phải điều khiển đồng thời?

# Dẫn nhập

- Ý tưởng giao tác để mô tả một đơn vị xử lý trong quá trình xử lý dữ liệu trong CSDL.
- Nhiều người cùng thao tác trên CSDL có thể gây ra tình trạng CSDL sai → điều khiển đồng thời.

## 2. GIAO TÁC

- Định nghĩa
- Tính chất
- Viết giao tác

# ĐỊNH NGHĨA GIAO TÁC

Giao tác là **1 tập hợp các thao tác có thứ tự** truy xuất dữ liệu trên CSDL **thành 1 đơn vị công việc logic** (*xem là 1 thao tác nguyên tố*), chuyển CSDL từ trạng thái nhất quán này sang **trạng thái nhất quán khác**.

# GIAO TÁC

## □ Cho 2 quan hệ

- LOP (Malop, Tenlop, SoSV)
- SV (MaSV, TenSV, Malop)

Giao tác thêm 1 SV vào 1 lớp

Giao tác Them\_SV (v\_masv, v\_tensv, v\_malop)

Insert into SV (v\_masv,v\_tensv,v\_malop)

Update LOP Set SoSV= SoSV + 1

Where Malop = v\_malop

Cuối giao tác Them\_SV

# Tính chất của giao tác: ACID

## □ Atomic – Tính nguyên tố

Không thể chia nhỏ

## □ Consistency – Tính nhất quán

Chuyển CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác

## □ Isolation – Tính cô lập

Các giao tác xử lý đồng thời phải độc lập với những thay đổi của giao tác khác

## □ Durability – Tính lâu dài, bền vững

Khi giao tác hoàn tất, tất cả thay đổi phải được ghi nhận chắc chắn lên CSDL



# T-SQL đặc trưng của giao tác

<b>BEGIN TRANSACTION</b>	<b>Bắt đầu giao tác.</b>
<b>COMMIT TRANSACTION</b>	<b>Kết thúc giao tác thành công.</b>
<b>ROLLBACK TRANSACTION</b>	<b>Kết thúc giao tác không thành công, CSDL được trả về tình trạng trước khi thực hiện giao tác.</b>

# MỘT SỐ LƯU Ý

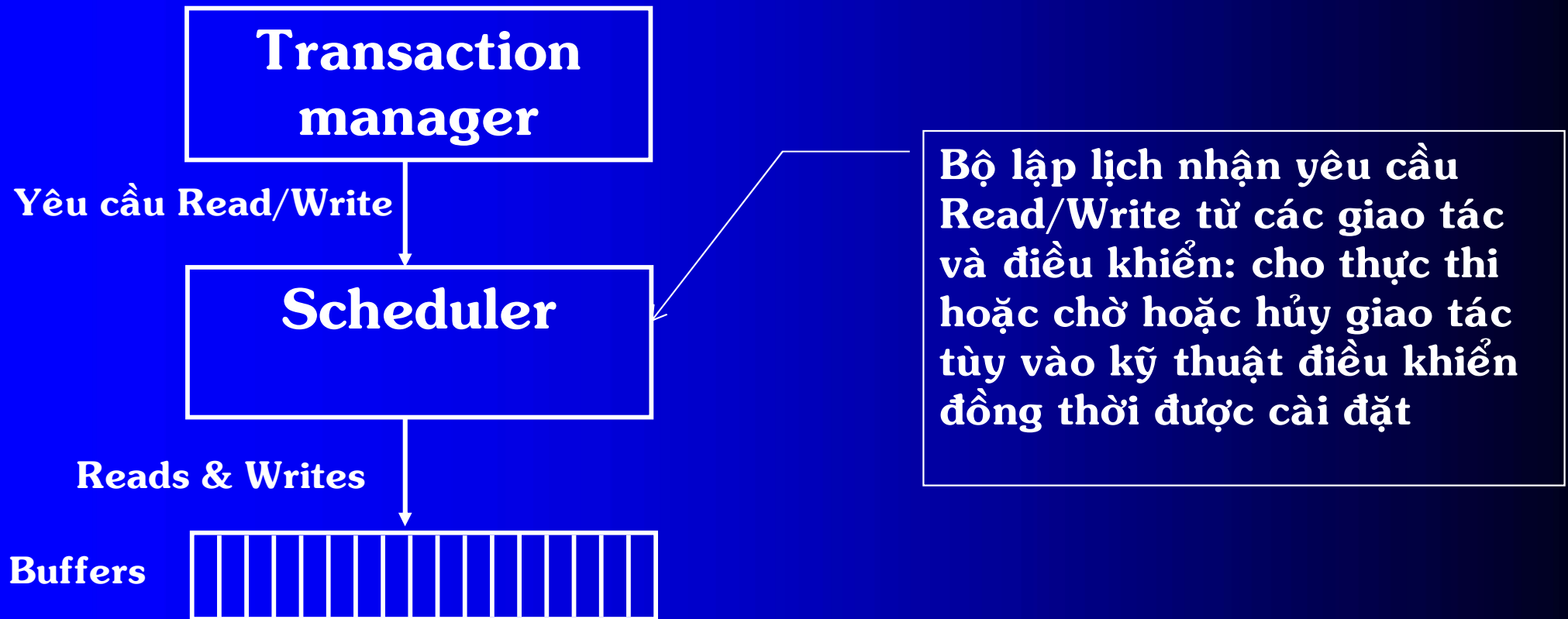
- ❑ Kiểm tra lỗi  
không có quyền, vi phạm ràng buộc, deadlock
- ❑ Biến toàn cục @@ERROR  
=0 : không lỗi, ≠ 0 : có lỗi
- ❑ GT Không tự rollback khi gặp lỗi trong quá trình thực hiện
- ❑ Cần kiểm tra giá trị biến @@ERROR sau mỗi câu lệnh thành phần và xử lý lỗi nếu có
- ❑ Biến @@ROWCOUNT
- ❑ Lồng 32 tầng, lệnh rollback ở tầng bất kỳ làm rollback toàn bộ giao tác.

### 3. GIAO TÁC TRUY XUẤT ĐỒNG THỜI

- Giới thiệu
- Các vấn đề của xử lý đồng thời

# Lý do điều khiển đồng thời

- ❑ Đảm bảo nhiều giao tác thực hiện đồng thời mà vẫn đảm bảo tính đúng đắn trên CSDL



# Các vấn đề của xử lý đồng thời

## 1. Mất dữ liệu cập nhật (Lost update)

T1	T2
Begin Tran	
Read A	
	Begin Tran
	Read A
A:=A+10	
Write A	
	A:=A*100
	Write A
	Commit Tran
Commit Tran	

# Các vấn đề của xử lý đồng thời

## 2. Đọc dữ liệu chưa commit (Uncommit data, Dirty read)

T1	T2
Begin Tran	
Read A	
$A := A + 10$	
Write A	
	Begin Tran
	Read A
	Print A
	Commit Tran
Rollback Tran	

# Các vấn đề của xử lý đồng thời

## 3. Thao tác đọc không thể lặp lại (Unrepeatable data)

T1	T2
Begin Tran	
Read A	
	Begin Tran
	Read A
	$A := A + 10$
	Write A
	Commit Tran
Read A	
Commit Tran	

# Các vấn đề của xử lý đồng thời

## 4. Phantom

<b>T1</b>	<b>T2</b>
<b>Begin Tran</b>	
<b>Select * From SV</b>	
	<b>Begin Tran</b>
	<b>Insert into SV values (...)</b>
	<b>Commit Tran</b>
<b>Select * From SV</b>	
<b>Commit Tran</b>	



## 4. LỊCH THAO TÁC

- Các khái niệm:
  - Lịch biểu
  - Lịch biểu tuần tự
  - Lịch biểu khả tuần tự
  - Lịch biểu có thể phục hồi
  - Lịch biểu không rollback dây chuyền
  - Bộ lập lịch
  - Nghi thức

# Một số khái niệm

1. Hoạt động của các giao dịch đồng thời được coi là đúng đắn nếu và chỉ nếu tác dụng của nó giống như tác dụng có được khi cho thực hiện chúng một cách **tuần tự**

2. **Lịch biểu (schedule)**

Đn1: Lịch biểu của một tập các giao tác là thứ tự trong đó các thao tác trong giao tác được thực hiện. Thứ tự của các thao tác trong lịch biểu phải tuân theo đúng thứ tự của chúng trong giao tác cho trước.

Đn2: Một lịch biểu là một chuỗi sắp theo thời gian các hành động được thực hiện bởi một hoặc nhiều giao tác.



# Một số khái niệm

- ❑ **Lịch biểu được gọi là tuần tự (serial)** nếu thứ tự thực hiện các thao tác trong lịch biểu là tất cả các thao tác của giao tác này rồi đến tất cả các thao tác của giao tác khác và cứ như vậy.
- ❑ Mọi lịch biểu tuần tự đều đảm bảo tính nhất quán cho cơ sở dữ liệu.
- ❑ **Lịch biểu được gọi là khả tuần tự (serializable)** nếu tác dụng của nó giống như tác dụng của một lịch biểu tuần tự nào đó. Tức là chúng sinh ra cùng một giá trị cho mỗi đơn vị dữ liệu.

# Lịch tuần tự

T1	T2	A	B	T1	T2	A	B
		25	25			25	25
Read (A,t)				Read (A,t)			
t:=t+100				t:=t+100			
Write (A,t)		125		Write (A,t)		125	
	Read (A,s)				Read (A,s)		
	s:=s*2				s:=s*1		
	Write (A,s)	250			Write (A,s)	125	
	Read (B,s)				Read (B,s)		
	s:=s*2				s:=s*1		
	Write (B,s)		50		Write (B,s)		25
Read (B,t)				Read (B,t)			
t:=t+100				t:=t+100			
Write (B,t)			150	Write (B,t)			125

Lịch biểu không khả tuần tự

Lịch biểu khả tuần tự nhưng không phải là lịch tuần tự

# Lịch khả tuần tự

- ❑ Lịch biểu có thể là khả tuần tự do hành vi cụ thể của các giao tác trong lịch biểu.
- ❑ Bộ lập lịch không điều khiển đồng thời dựa vào hành vi cụ thể của từng thao tác trong giao tác.
- ❑ Bộ lập lịch dựa trên một nguyên tắc chung để điều khiển đồng thời.

# Ký hiệu

- $r_i(X)$  : giao tác  $T_i$  đọc đơn vị dữ liệu  $X$
- $w_i(X)$  : giao tác  $T_i$  ghi trên đơn vị dữ liệu  $X$
- Giao tác  $T_i$  gồm một chuỗi các thao tác  $o_i$

# Một số khái niệm

- Lịch biểu có thể phục hồi được (Recoverable Schedule)

T1	T2
Begin Tran	
Read A	
Write A	
	Begin Tran
	Read A
	Commit
Read B	
Commit	



T1	T2
Begin Tran	
Read A	
Write A	
	Begin Tran
	Read A
Read B	
Commit	
	Commit

Khi T1 có sự cố, T2 đã commit nên không thể thoát  $\Rightarrow$  tình trạng không thể phục hồi đúng được.



# Một số khái niệm

- Lịch biểu có thể phục hồi được (Recoverable Schedule)

Mọi HQT CSDL yêu cầu lịch biểu phải có thể phục hồi được.

**Định nghĩa:** Một lịch biểu có thể phục hồi được là lịch biểu mà mọi cặp  $T_i, T_j$ , khi  $T_j$  đọc đơn vị dữ liệu vừa được ghi bởi  $T_i$ , thao tác commit của  $T_i$  xuất hiện trước thao tác commit của  $T_j$ .

# Một số khái niệm

- ❑ Lịch biểu không rollback dây chuyền (Uncascading rollback Schedule)

T1	T2	T3
Read A		
Read B		
Write A		
	Read A	
	Write A	
		Read A

Khi T1 fail, phải rollback T2 và kéo theo T3 phải bị rollback.

Hiện tượng một transaction rollback, dẫn đến một loạt các transaction khác phải rollback gọi là rollback dây chuyền (Cascading rollback).

# Một số khái niệm

- Lịch biểu không rollback dây chuyền (Uncascading rollback Schedule)
- Định nghĩa: Lịch biểu không rollback dây chuyền là lịch biểu mà trong đó mọi cặp giao tác  $T_i, T_j$ , nếu  $T_j$  đọc đơn vị dữ liệu được viết trước đó bởi  $T_i$ , thao tác commit của  $T_i$  phải xuất hiện trước thao tác đọc của  $T_j$ .
- Một lịch biểu không rollback dây chuyền là lịch biểu có khả năng phục hồi được.

# Một số khái niệm

□ Tính tương thích của 2 thao tác

**Định nghĩa:** Hai thao tác  $O_i$ ,  $O_j$  là tương thích nếu kết quả của việc thực hiện đồng thời  $O_i$ ,  $O_j$ , giống như kết quả của việc thực hiện tuần tự  $O_i$ ,  $O_j$  hoặc  $O_j$ ,  $O_i$

Hai thao tác không tương thích nhau thì xung đột nhau.

Hai thao tác tương thích thì không xung đột nhau.

VD: Read A và Read A, Read A và Read B, Read A và Write B là các cặp thao tác tương thích

# Một số khái niệm

## Tính khả hoán vị của 2 thao tác

- **Định nghĩa:** Hai thao tác  $O_i$ ,  $O_j$  là khả hoán vị nếu kết quả của việc thực hiện  $O_i$ ,  $O_j$  hay  $O_j$ ,  $O_i$  là như nhau.
- Hai thao tác tương thích thì khả hoán vị.

# Một số khái niệm

□ Hai thao tác truy xuất trên cùng đơn vị dữ liệu, ma trận tương thích

	Read A	Write A
Read A	1	0
Write A	0	0

- Các thao tác truy xuất trên các đơn vị dữ liệu khác nhau thì tương thích và khả hoán vị

# Một số khái niệm

- Với một lịch biểu S1 cho trước, ta có thể lặp lại việc hoán đổi vị trí của hai thao tác không xung đột liên kề, cho đến khi đạt được 1 lịch biểu tuần tự S2, nếu có thể. Khi đó, lịch biểu S1 ban đầu là lịch biểu khả tuần tự.
- Mọi tác động đến CSDL hoàn toàn không thay đổi khi ta thực hiện hoán đổi vị trí các thao tác không xung đột nhau.

# Một số khái niệm

- Hai lịch biểu là tương đương xung đột nếu ta có thể chuyển lịch biểu này thành lịch biểu kia bằng một/một số các thao tác hoán đổi các thao tác không xung đột kề nhau.
- Một lịch biểu là khả tuần tự xung đột nếu nó tương đương xung đột với một lịch biểu tuần tự.



# Xét tính khả tuần tự của S

T1	T2	T3	T4
	1.Read A		
		2.Read A	
	3.Write B		
		4.Write A	
5.Read B			
			6.Read B
7.Read A			
8.Write C			
			9.Write A

# Một số khái niệm

- Khả tuần tự xung đột là điều kiện đủ để đảm bảo tính khả tuần tự. Nghĩa là, một lịch biểu khả tuần tự xung đột là một lịch biểu khả tuần tự.
- Không đòi hỏi một lịch biểu phải khả tuần tự xung đột để là lịch biểu khả tuần tự, nhưng khả tuần tự xung đột là điều kiện để những bộ lập lịch trong các hệ thống thương mại bảo đảm tính khả tuần tự.

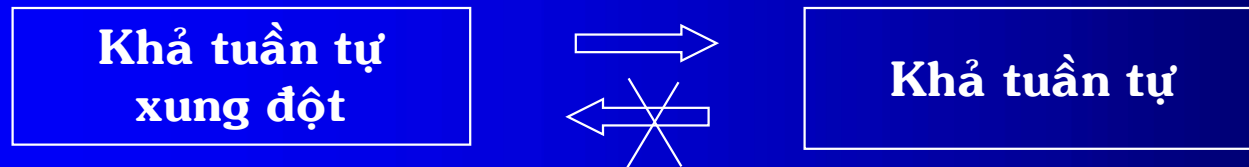
# Một số khái niệm

- Khả tuần tự xung đột không là điều kiện cần cho tính khả tuần tự.

$S1: w_1(Y) ; w_1(X) ; w_2(Y) ; w_2(X) ; w_3(X)$

$S2: w_1(Y) ; w_2(Y) ; w_2(X) ; w_1(X) ; w_3(X)$

- S1: T3 ghi X, T2 ghi Y
- S2: T1, T2 ghi trên X nhưng T3 ghi đè, T2 ghi Y.
- Hai lịch biểu trên cho giá trị trên X và Y giống nhau.
- S1 tuần tự
- ⇒ S2 khả tuần tự
- ⇒ nhưng không khả tuần tự xung đột do không thể chuyển S2 về 1 lịch biểu tuần tự bằng cách hoán đổi.

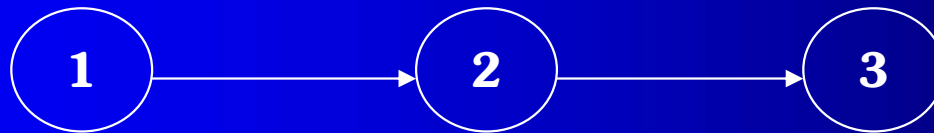


# Một số khái niệm

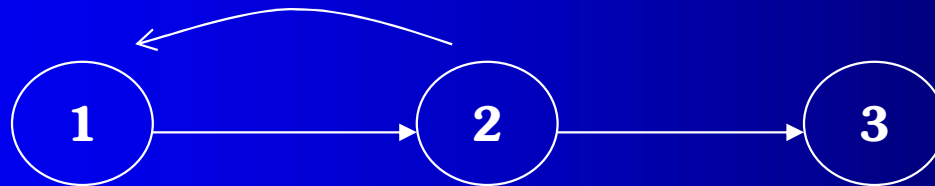
- **Đồ thị ưu tiên (đồ thị đưng độ) để kiểm tra một lịch biểu là khả tuần tự xung đột hay không.**
- Cho  $S$  là lịch biểu gồm các thao tác của 2 giao tác  $T_i$  và  $T_j$ .  $T_i$  ưu tiên hơn  $T_j$  ( $T_i < T_j$ ) nếu tồn tại cặp thao tác  $O_{in} \in T_i$ ,  $O_{jm} \in T_j$  không khả hoán vị và  $O_{in}$  thực hiện trước  $O_{jm}$ .
- Đồ thị ưu tiên có nút là các giao tác, cung có hướng đi từ nút  $i$  đến nút  $j$  nếu  $T_i < T_j$ .
- Lịch biểu là khả tuần tự xung đột nếu đồ thị ưu tiên không có chu trình.
- Nếu đồ thị ưu tiên có chu trình, lịch biểu là không khả tuần tự xung đột.

# Một số khái niệm

□  $S: r_2(A) ; r_1(B) ; w_2(A) ; r_3(A) ; w_1(B) ; w_3(A) ; r_2(B) ; w_2(B)$



□  $S: r_2(A) ; r_1(B) ; w_2(A) ; r_2(B) ; r_3(A) ; w_1(B) ; w_3(A) ; w_2(B)$



# Một số khái niệm

- ❑ Khi thực hiện đồng thời các transaction phát sinh các vấn đề như:
  - Lịch thao tác không khả tuần tự
  - Livelock: tình trạng 1 giao tác chờ hoài để được làm việc trên 1 đvdl.
  - Deadlock: tình trạng hai giao tác cứ chờ nhau mãi để được làm việc trên 1 đơn vị dữ liệu mà không có giao tác nào có thể thực hiện trước.
- ❑ Để giảm bớt những vấn đề này, ta có 2 công cụ:
  - Bộ lập lịch.
  - Nghi thức.

# Một số khái niệm

- ❑ **Bộ lập lịch (schedulers):** sẽ tiến hành lập lịch các thao tác (thao tác sẽ được thực hiện trước, thao tác nào sẽ được thực hiện sau). Bộ lập lịch giải quyết các yêu cầu đung độ, ép các transaction phải chờ trong trường hợp không đáp ứng được yêu cầu lock hoặc hủy bỏ các transaction (khi deadlock).
- ❑ Bộ lập lịch là thành phần của hệ quản trị CSDL, có vai trò làm trọng tài phân xử các yêu cầu có xung đột.

# Một số khái niệm

- ❑ **Nghi thức (Protocol):** cũng giải quyết vấn đề deadlock và đảm bảo tính khả tuần tự.
- ❑ Ví dụ như chiến lược lock 2 pha, hoặc yêu cầu các giao tác lock các đơn vị dữ liệu theo 1 thứ tự cố định nào đó.
- ❑ Nghi thức là một hạn chế trên chuỗi các bước nguyên tử mà một giao dịch có thể thực hiện.



## 5. ĐIỀU KHIỂN ĐỒNG THỜI DÙNG KỸ THUẬT KHÓA

# Nguyên tắc khóa

- Bộ lập lịch dùng kỹ thuật khóa yêu cầu:
  - Phải khóa và nhả khóa ngoài việc Đọc/Ghi dữ liệu.
  - Việc dùng khóa phải hợp lệ: theo 2 điều kiện sau
    - Tính nhất quán của giao tác:
      1. Giao tác chỉ có thể đọc hoặc ghi trên đơn vị dữ liệu nếu trước đó có yêu cầu lock trên đơn vị dữ liệu và chưa nhả lock.
      2. Nếu giao tác đã lock trên đvdl thì sau đó phải unlock.
    - Tính hợp lệ của lịch biểu:
      - Không thể có 2 giao tác đồng thời khóa trên 1 đvdl

# Ký hiệu

## □ Ký hiệu:

$l_i(X) :$	$T_i$ yêu cầu lock trên đvdl X
$u_i(X) :$	$T_i$ unlock trên đvdl X

## □ Ví dụ: T1 và T2 nhất quán

### □ T1:

$l_1(A) ; r_1(A) ; A := A + 100 ; w_1(A) ; u_1(A) ; l_1(B) ; r_1(B) ; B := B + 100 ;$   
 $w_1(B) ; u_1(B) ;$

### □ T2:

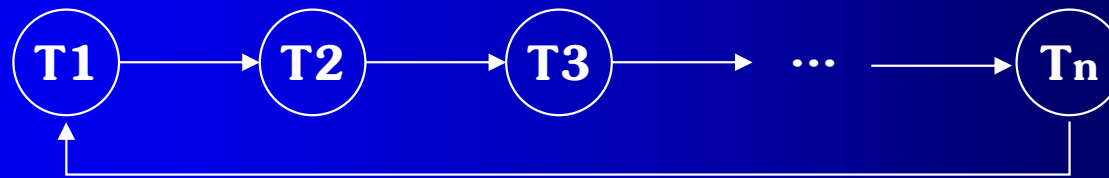
$l_2(A) ; r_2(A) ; A := A * 2 ; w_2(A) ; u_2(A) ; l_2(B) ; r_2(B) ; B := B * 2 ;$   
 $w_2(B) ; u_2(B) ;$

T1	T2	A	B
		25	25
$l_1(A) ; r_1(A) ;$			
$A := A + 100 ;$			
$w_1(A) ; u_1(A) ;$		125	
	$l_2(A) ; r_2(A) ;$		
	$A := A * 2 ;$		
	$w_2(A) ; u_2(A) ;$	250	
	$l_2(B) ; r_2(B) ;$		
	$B := B * 2 ;$		
	$w_2(B) ; u_2(B) ;$		50
$l_1(B) ; r_1(B) ;$			
$B := B + 100 ;$			
$w_1(B) ; u_1(B) ;$			150

**Lịch biểu hợp lệ, giao tác nhất quán nhưng  
lịch biểu không khả tuần tự**

## Nghi thức khóa 2 giai đoạn (Two-phase lock Protocol - 2PL)

- ❑ Phát biểu 2PL: Một giao tác thực hiện cơ chế khóa 2 giai đoạn khi giao tác không thực hiện lock nào nữa sau khi đã unlock.
- ❑ 2PL đảm bảo một lịch biểu hợp lệ gồm các giao tác nhất quán là khả tuần tự xung đột.
- ❑ Chứng minh: Giả sử đồ thị khả tuần tự là có chu trình



- ❑ T1 có dạng Lock...Unlock...Lock => T1 không thỏa nghi thức lock 2 giai đoạn.

## 2PL

- Ta có thể chuyển 1 lịch biểu hợp lệ gồm các giao tác nhất quán và thoả 2PL sang 1 lịch biểu tuần tự tương đương xung đột.
- Chứng minh bằng phương pháp quy nạp. Chỉ quan tâm đến thao tác Read và Write.
- S: T1, T2,..., Tn. Đặt Ti là giao tác unlock đầu tiên, ui(X).
- Ta hoàn toàn có thể chuyển tất cả các thao tác Read và Write của Ti ra đầu lịch biểu mà không gặp phải 1 thao tác xung đột nào. Thật vậy:
  - Xét wi(Y) nào đó của Ti. Giả sử có thao tác wj(Y) của S đi trước wi(Y):  
S: ..., wj(Y) ..., uj(Y), ..., li(Y), ..., wi(Y), ...
  - Vì Ti unlock đầu tiên, ui(X) phải đứng trước uj(Y):  
S: ..., wj(Y), ..., ui(X), ..., uj(Y), ..., li(Y), ..., wi(Y), ... hoặc  
S: ..., ui(X) ..., wj(Y), ..., uj(Y), ..., li(Y), ..., wi(Y), ...
  - Ti không thoả 2PL.
- S được viết lại:  
(các thao tác của Ti) (các thao tác của n-1 giao tác còn lại)  
Phần còn lại là lịch biểu hợp lệ gồm các GT nhất quán và thoả 2PL → bằng quy nạp ta kết luận S là khả tuần tự xung đột.

# 2PL: Deadlock & Rollback dây chuyền

T1	T2	A	B
		25	25
$l_1(A) ; r_1(A) ;$			
	$l_2(B) ; r_2(B) ;$		
$A := A + 100 ;$			
	$B := B * 2 ;$		
$w_1(A) ;$		125	
	$w_2(B) ;$		50
$l_1(B) ; \text{Denied}$	$l_2(A) ; \text{Denied}$		

- Tình trạng rollback dây chuyền có thể xảy ra với lịch biểu thỏa nghi thức khóa 2 giai đoạn.

## **Nghi thức khóa nghiêm ngặt (Strict locking)**

- ❑ **Khóa nghiêm ngặt:** Tất cả các khóa đọc quyền của 1 giao tác bất kỳ phải giữ cho đến khi giao tác commit hoặc abort, và lệnh commit/ abort phải được ghi nhật ký trên đĩa.
- ❑ **Lịch biểu trong đó các giao tác tuân thủ cách khóa nghiêm ngặt là lịch biểu nghiêm ngặt (strict schedule).**
  - Mọi lịch biểu nghiêm ngặt là lịch biểu không rollback dây chuyền.
  - Mọi lịch biểu nghiêm ngặt là khả tuần tự.
    - Vì lịch biểu nghiêm ngặt tương đương với lịch biểu tuần tự trong đó các giao tác thi hành tại thời điểm nó commit.



# Các biến thể của 2PL

- Conservative: GT phải yêu cầu khóa tất cả các mục dữ liệu cần thiết TRƯỚC khi GT bắt đầu thực hiện.
  - Không bị deadlock, nhưng không dễ thực hiện vì chưa thể biết mục dữ liệu nào sẽ cần.
- Rigorous 2PL: nhả tất cả khóa ở cuối GT.
  - Vẫn phải giữ Read lock đến cuối GT, giảm tính đồng thời.
- Strict 2PL: Sau khi GT đã commit/ abort, các khoá đọc mới được nhả.
  - Có thể deadlock, nhưng đảm bảo lịch biểu có thể phục hồi được. Lịch này được dùng phổ biến.

# Các phương thức khóa

## □ Shared lock (S)

- Shared Lock  $\Leftrightarrow$  Read Lock
- Khi đọc 1 đơn vị dữ liệu, SQL Server tự động thiết lập Shared Lock trên đơn vị dữ liệu đó
- Có thể được thiết lập trên 1 bảng, 1 trang, 1 khóa hay trên 1 dòng dữ liệu.
- Nhiều giao tác có thể đồng thời giữ Shared Lock trên cùng 1 đơn vị dữ liệu.
- Không thể thiết lập Exclusive Lock trên đơn vị dữ liệu đang có Shared Lock.
- Shared Lock thường được giải phóng ngay sau khi sử dụng xong dữ liệu được đọc, trừ khi có yêu cầu giữ shared lock cho đến hết giao tác.

# Các phương thức khóa

## ❑ Exclusive Locks (X)

- Exclusive Lock  $\Leftrightarrow$  Write Lock
- Khi thực hiện thao tác ghi (insert, update, delete) trên 1 đơn vị dữ liệu, SQL Server tự động thiết lập Exclusive Lock trên đơn vị dữ liệu đó.
- Exclusive Lock luôn được giữ đến hết giao tác.
- Tại 1 thời điểm, chỉ có tối đa 1 giao tác được quyền giữ Exclusive Lock trên 1 đơn vị dữ liệu.

# Nguyên tắc khóa

## 1. Tính nhất quán của giao tác

- Thao tác đọc  $r_i(X)$  phải đi sau  $sl_i(X)$  hoặc  $xl_i(X)$  mà không có thao tác  $u_i(X)$  xen vào giữa.
- Thao tác ghi  $w_i(X)$  phải đi sau  $xl_i(X)$  mà không có thao tác  $u_i(X)$  xen vào giữa.
- Tất cả các lock phải được unlock trên cùng đvdl.

## 2. Giao tác thỏa 2PL

- Không có thao tác  $u_i(Y)$  nào đi trước  $sl_i(X)$  hoặc  $xl_i(X)$

## 3. Lịch biểu hợp lệ

- Nếu có thao tác  $xl_i(X)$  trong lịch biểu thì không thể có  $xl_j(X)$  hoặc  $sl_j(X)$  theo sau,  $j \neq i$  mà không có lệnh  $u_i(X)$  nào ở giữa.
- Nếu có thao tác  $sl_i(X)$  trong lịch biểu thì không thể có  $xl_j(X)$  theo sau,  $j \neq i$  mà không có lệnh  $u_i(X)$  ở giữa.

# Ma trận tương thích

	S	X
S	yes	no
X	no	no

# Khoá tăng cấp

## □ Khoá tăng cấp (Upgrading lock)

- Shared lock: thân thiện
- T muốn đọc X trước, sau đó ghi X thì trước tiên khoá đọc trên X, sau đó khi muốn ghi thì nâng cấp (upgrade) khoá đọc thành khoá ghi.
- Cách này cho phép tăng tính đồng thời.

# Khóa tăng cấp

T1	T2
$sl_1(A) ; r_1(A) ;$	
	$sl_2(A) ; r_2(A) ;$
	$sl_2(B) ; r_2(B) ;$
$sl_1(B) ; r_1(B) ;$	
$xl_1(B) \text{ denied}$	
	$u_2(A) ; u_2(B) ;$
$xl_1(B) ; w_1(B) ;$	
$u_1(A) ; u_1(B) ;$	

**Upgrading lock giúp tăng tính đồng thời**

# Ví dụ

T1	T2
$sl_1(A) ;$	
	$sl_2(A) ;$
	$sl_2(B) ; r_2(B) ;$
$xl_1(A) ; denied$	
$xl_1(B)$	
	$u_2(A) ; u_2(B) ;$
$xl_1(B) ; w_1(B) ;$	
$u_1(A) ; u_1(B) ;$	



# Các phương thức khóa

## □ Update Lock $ul_i(X)$

- Update Lock sử dụng khi đọc dữ liệu với dự định ghi trở lại trên đơn vị dữ liệu này.
- Update Lock là chế độ khóa trung gian giữa Shared Lock và Exclusive Lock.
- Khi thực hiện thao tác ghi lên 1 đơn vị dữ liệu thì bắt buộc Update Lock phải tự động chuyển đổi thành Exclusive Lock
- Tại 1 thời điểm chỉ cho phép 1 giao tác giữ Update lock trên 1 đvdl

# Ma trận tương thích

	S	X	U
S	Yes	No	Yes
X	No	No	No
U	Yes	No	No

# Ví dụ

T1	T2
$ul_1(A) , r_1(A) ;$	
	$ul_2(A) , Denied$
$xl_1(A) , w_1(A) , u_1(A) ;$	
	$ul_2(A) , r_2(A) ;$
	$xl_2(A) , w_2(A) , u_2(A) ;$

# Khóa tăng/giảm

## □ Increment lock $il_i(X)$ $inc_i(X)$

### ➤ Ý tưởng:

- Hành động tăng giá trị / giảm giá trị nếu dùng khóa đọc/ ghi thì phải chờ nhau.
- Thật ra chúng có thể hoán vị hoặc thực hiện đồng thời vẫn cho kết quả không thay đổi.
- Thay thế

`read (A,t); t:=t+/-c;write(A,t);`

Bởi `INC(A,c)` để làm tăng tính đồng thời mà vẫn đảm bảo đúng đắn.

# Bổ sung điều kiện ứng dụng khóa khi dùng khóa tăng/giảm

## ❑ Giao tác nhất quán

GT chỉ có thể thực hiện hành động tăng giá trị X nếu nó khóa tăng trên đvdl X. Khóa tăng không cho phép đọc hoặc ghi đồng thời.

## ❑ Lịch biểu hợp lệ

- Nhiều giao tác có thể cùng khóa tăng trên X.
- Ma trận tương thích

	S	X	I
S	Yes	No	No
X	No	No	No
I	No	No	Yes

# Ví dụ

T1	T2
$sl_1(A), r_1(A);$	
	$sl_2(A), r_2(A);$
	$il_2(B), inc_2(B);$
$il_1(B), inc_1(B);$	
	$u_2(A), u_2(B);$
$u_1(A), u_1(B);$	

# Khóa trên các đvdl có kích thước khác nhau

- ❑ Ta quan tâm đến cấu trúc phân cấp của đvdl.
- ❑ Hệ thống khác nhau dùng các đơn vị dữ liệu có kích thước khác nhau để khóa. VD: bộ, block, relation.
- ❑ Có ứng dụng, có khi đvdl nhỏ sẽ có lợi, có khi đvdl lớn sẽ có lợi hơn.
- ❑ Ví dụ: T đọc toàn bộ quan hệ, U ghi 1 dòng của quan hệ. Khi T và U thực hiện đồng thời sẽ xảy ra tình trạng sai trên CSDL.
- ❑ Đvdl lớn thì hạn chế việc thực hiện đồng thời. Đvdl nhỏ thì quản lý phức tạp.

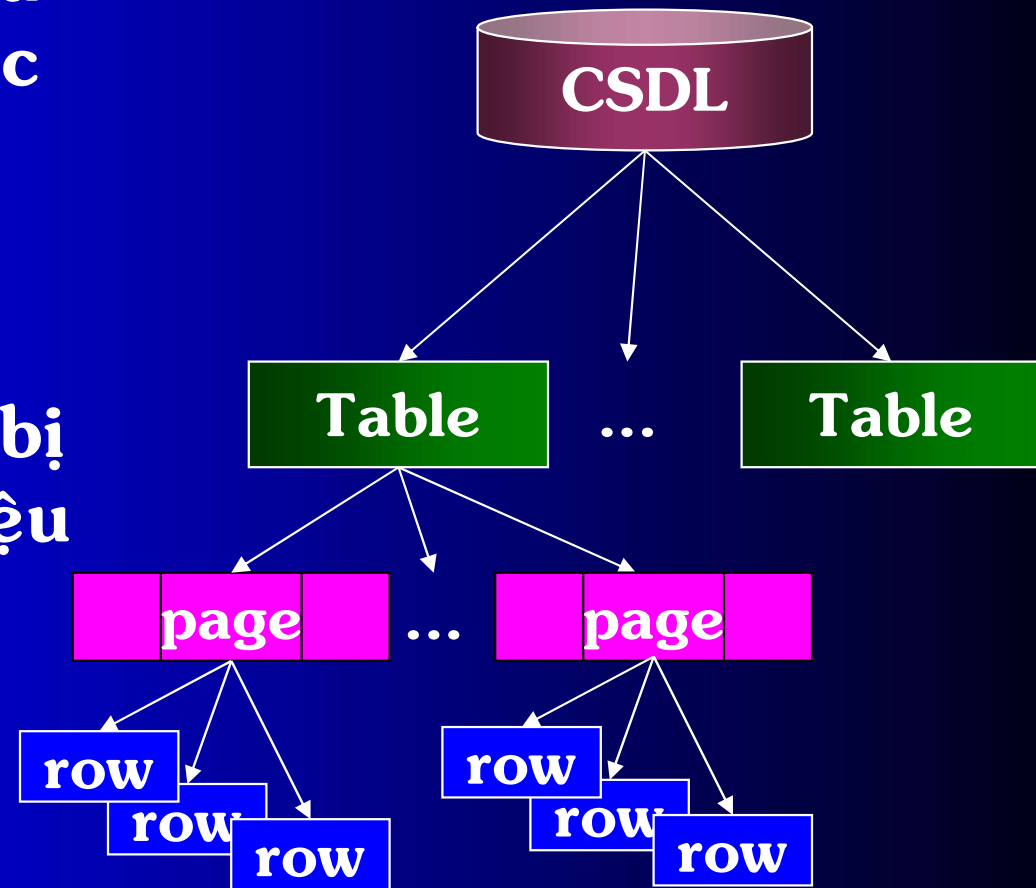
# Các tài nguyên có thể khóa

Tài nguyên	Giải thích
Database	Khóa trên toàn bộ cơ sở dữ liệu. Chỉ nên áp dụng khi tiến hành thay đổi trên lược đồ của CSDL.
Table	Khóa trên 1 bảng trong cơ sở dữ liệu. Toàn bộ các đối tượng trong bảng này, bao gồm tất cả các dòng và tất cả các khóa trong các chỉ mục trong bảng, đều bị khóa.
Extent	Khóa trên 1 extend (= 8 trang).
Page	Khóa trên 1 trang. Tất cả dữ liệu và các khóa chỉ mục trong trang này đều bị khóa.
Row	Được đưa vào SQL Server từ version 7.0. Khóa trên 1 dòng dữ liệu trong 1 bảng.



# Các tài nguyên có thể khóa

- ❑ Khi khóa trên 1 đơn vị dữ liệu ở cấp cao hơn thì các đơn vị dữ liệu con bên trong cũng bị khóa.
- ❑ Khi 1 đơn vị dữ liệu con bị khóa thì các đơn vị dữ liệu ở các cấp cao hơn sẽ bị khóa bằng khóa Intent tương ứng.



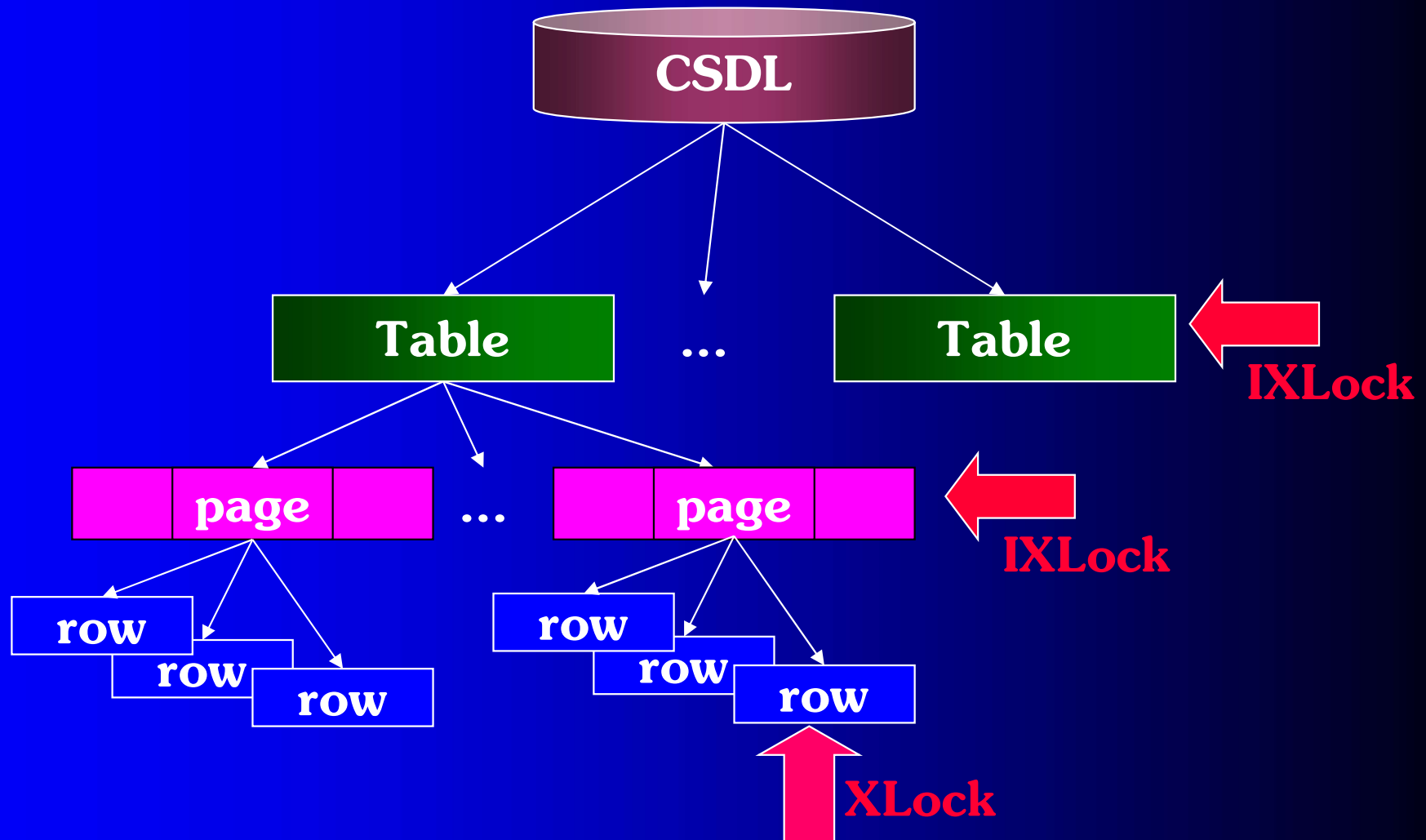
# Các phương thức khóa

## □ *Intent Locks*

- Không phải là 1 chế độ khóa riêng biệt mà được sử dụng kết hợp với các chế độ khóa khác:
  - **Shared Lock => Intent Shared Lock (IS)**
  - **Update Lock => Intent Update Lock (IU)**
  - **Exclusive Lock => Intent Exclusive Lock (IX)**
- Intent Lock chỉ áp dụng trên table và page
- Intent Lock được SQL Server tự động thiết lập, không thể được yêu cầu thiết lập Intent Lock một cách tường minh.
- Kiểm tra 1 đơn vị dữ liệu thành phần của 1 đơn vị dữ liệu có đang bị khóa hay không
- Khi 1 đơn vị dữ liệu thành phần bị khóa, các đơn vị dữ liệu ở cấp cao hơn cũng sẽ bị khóa bằng Intent Lock tương ứng

# Các phương thức khóa

## □ *Intent Locks*



# Nguyên tắc

1. Để đặt khóa S hoặc X trên 1 đvdl, phải bắt đầu tại gốc.
2. Nếu vị trí hiện tại là vị trí muốn khóa, không cần đi tiếp. Yêu cầu khóa S hoặc X tại đvdl hiện tại.
3. Nếu đvdl muốn khóa còn ở cấp dưới, thì:
  1. Đặt khóa cảnh báo tại node hiện tại.
    1. Nếu yêu cầu Slock thì đặt IS tại node hiện tại.
    2. Nếu yêu cầu Xlock thì đặt IX tại node hiện tại.
  2. Đi tiếp theo hướng đến được đvdl cần tìm.
4. Lặp lại bước 2 và 3.

# Ma trận tương thích

	IS	IX	S	X
IS	Yes	Yes	Yes	No
IX	Yes	Yes	No	No
S	Yes	No	Yes	No
X	No	No	No	No

# Giải quyết phantom

- ❑ Chèn hoặc xóa một dòng thì thường xảy ra phantom.
- ❑ Xem thao tác chèn hoặc xóa như là thao tác ghi trên toàn bộ bảng.

# Khóa trên đvdl có kích thước khác nhau

- Có HQT áp dụng kỹ thuật khóa trên kích thước đvdl động.
  - HQT CSDL hỗ trợ nhiều kích thước đvdl khác nhau. VD: record, page, file,...
  - Kích thước đvdl được dùng cho giao tác thay đổi tự động để phù hợp với giao tác.
  - Có thể tự động nâng cấp khóa từ đvdl là record thành page thành file nếu giao tác đang khóa trên một số phần trăm (khá lớn) nào đó các record hoặc page của file.

# Kiểm tra tính khả tuần tự

- Với lịch biểu có Rlock và Wlock
- Input: LB S gồm T1, T2, ..., Tk
- Output: S có khả tuần tự hay không
  1. Nếu Ti Rlock trên đvdl X, tiếp theo Tj yêu cầu Xlock trên X, thì có cung  $T_i \rightarrow T_j$ .
  2. Ti khóa Xlock trên X, Tj yêu cầu Xlock trên X, có cung từ  $T_i \rightarrow T_j$ .
  3. Tm là GT giữ Slock trên X sau khi Ti nhả khóa độc quyền trên X, nhưng trước khi Tj khóa Xlock trên X, (nếu không có Tj thì Tm là giao tác yêu cầu Slock trên X sau khi Ti nhả khóa trên X), thì có cung từ Ti đến Tm.
  4. Nếu đồ thị có chu trình thì S không khả tuần tự. Ngược lại thì S khả tuần tự và có thể tìm được lịch tuần tự tương đương.



# Nghi thức khóa phân cấp

- Dùng cho dữ liệu có cấu trúc cây.
  - VD: B-tree.
- Đvdl phù hợp để khoá là node.
- Dùng 2PL không phù hợp, vì tất cả đều bắt đầu từ root, và chỉ cho phép 1 giao tác không phải là chỉ đọc truy cập cây tại 1 thời điểm.

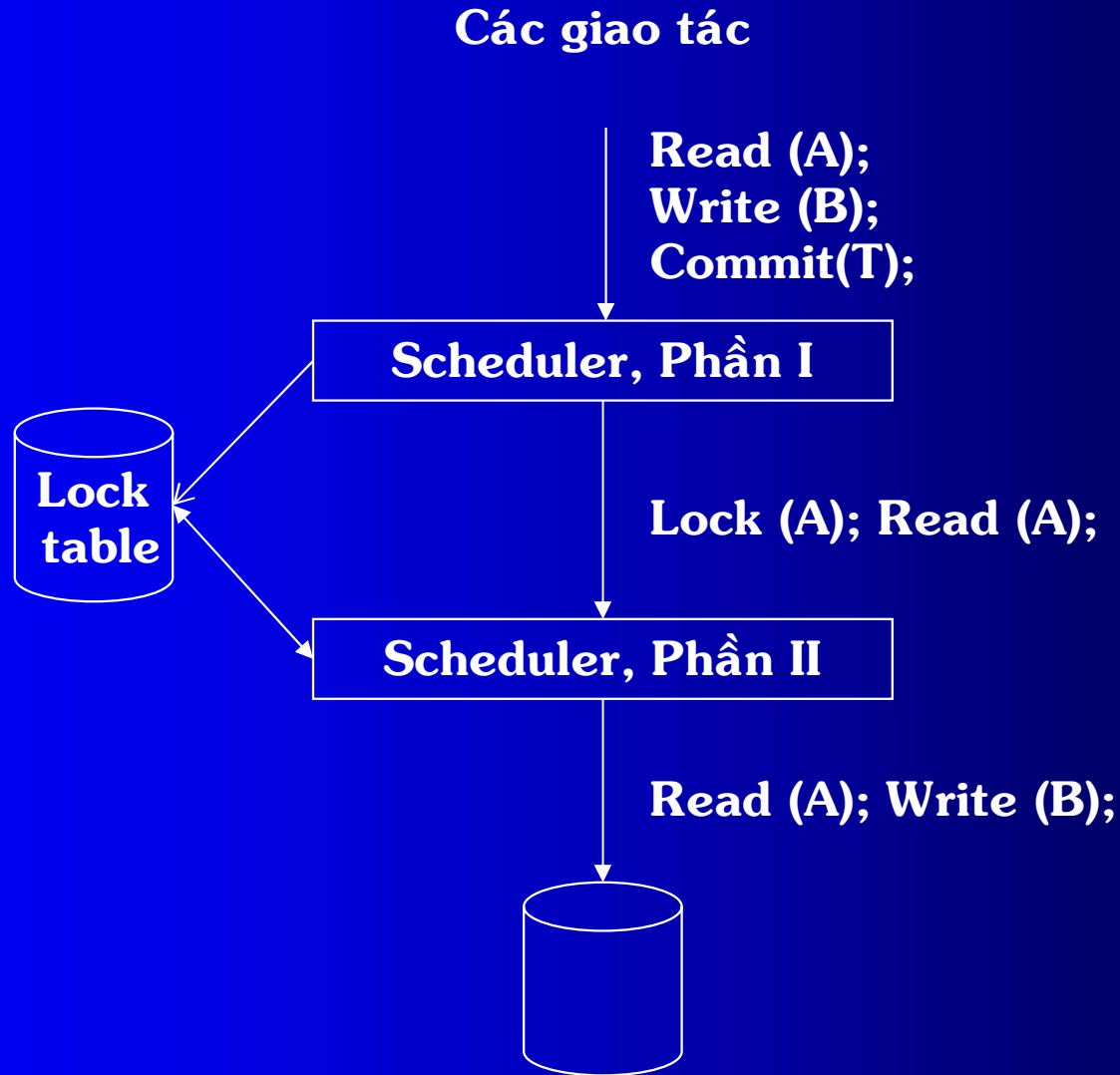
# Nghi thức khoá phân cấp

1. GT có thể bắt đầu khoá tại node bất kỳ của cây.
  2. Mục X chỉ có thể bị khóa bởi Ti nếu Ti đã khoá node cha của X trước đó.
  3. Có thể nhả khóa trên 1 node bất cứ lúc nào.
  4. GT không thể nhả lock trên 1 node rồi lock trở lại node đó, ngay cả khi vẫn còn khóa ở node cha.
- 
- Lịch biểu hợp lệ gồm các GT nhất quán và tuân theo nghi thức khóa phân cấp trên thì khả tuân tự.
  - Lịch tuân tự tương đương:  $T_i < ST_j$  nếu trong S, GT  $T_i$  và  $T_j$  khóa cùng 1 node và  $T_i$  khóa node đó trước.

# **Kiến trúc bộ lập lịch dùng phương thức khóa**

- ❑ **Bản thân giao tác không tự yêu cầu khóa. Việc chèn thao tác khoá vào trước các thao tác Đọc/ Ghi là công việc của Bộ lập lịch.**
  - **Bộ lập lịch nhận yêu cầu read, write, commit hay abort từ giao tác.**
  - **Bộ lập lịch lưu thông tin về việc quản lý khóa trong lock table.**
- ❑ **Giao tác không tự nhả khóa. Bộ lập lịch nhả khóa khi bộ quản lý giao tác (Transaction Manager - TM) báo cho Bộ lập lịch biết giao tác sẽ commit hay abort.**

# Kiến trúc bộ lập lịch dùng phương thức khóa



# Nguyên tắc cấp khóa

1. **Phần I** Chọn phương thức lock phù hợp để chen vào trước các thao tác Read, Write, Increment, update. Có những phương thức khóa nào là tùy thuộc vào HQT CSDL.
2. **Phần II** Nhận output của Part I và thi hành:
  - ❑ T delayed: vì chưa được đáp ứng yêu cầu lock, sẽ chen các thao tác của giao tác đó vào hàng chờ.
  - ❑ T not delayed: yêu cầu lock được đáp ứng:
    - Nếu là thao tác truy cập CSDL thì được HQT thực hiện.
    - Nếu là thao tác lock, Bộ lập lịch kiểm tra lock table xem có thể cấp lock không.
      - Nếu được, cho phép lock và cập nhật lock table với lock vừa cấp.
      - Nếu không, thêm 1 mục vào lock table, T delayed, cho đến khi lock được cấp.
3. Khi T commit hoặc abort, TM báo cho Phần I biết và nhả tất cả các lock giữ bởi T. Nếu có GT nào đang chờ những lock này, Phần I sẽ báo cho Phần II biết.
4. Khi nhận thông báo về các khóa trên X đã nhả từ Phần I, Phần 2 cấp khóa cho T đang chờ trên X, T thi hành tiếp cho đến khi thi hành xong hoặc tiếp tục chờ vì khóa chưa thể được đáp ứng.

# Bảng quản lý khóa (Lock table)

Element Info

<b>A</b>	

Group mode: U
Waiting: Yes
List

Tran	Mode	Wait?	Tnext	Next
T1	S	No		
T2	U	no		
T3	X	yes		

# Bảng quản lý khóa

- ❑ **ĐN:** là một quan hệ lưu thông tin về 1 đơn vị dữ liệu và thông tin các khóa đang yêu cầu trên đvdl đó.
- ❑ **Group mode:** chứa khóa nghiêm ngặt nhất trong các khóa đang yêu cầu trên A. Căn cứ vào khóa này để grant/ deny yêu cầu khóa trên A. Với mô hình Shared-Exclusive-Update thì group mode sẽ là:
  - S nếu chỉ có một/ một số shared lock đang được giữ.
  - U nếu có 1 update lock và có thể có shared lock.
  - X nếu có 1 exclusive lock và không có lock nào khác.
- ❑ **Waiting bit:** cho biết có ít nhất 1 GT đang chờ lock trên A.
- ❑ **List:** các GT đang giữ lock trên A hoặc chờ lock trên A:
  - Tên GT đang giữ lock hoặc đang chờ.
  - Mode của lock.
  - Tình trạng là đang giữ lock hay đang chờ lock.

# Quản lý yêu cầu lock

## □ T yêu cầu lock A:

Nếu chưa có mục A trên lock table

Tạo 1 mục mới và yêu cầu được đáp ứng.

Ngược lại

Dùng thông tin group mode để quyết định cấp lock hay không. Nếu không xung đột thì cấp, nếu xung đột sẽ denied và thêm vào list tên giao tác T, kiểu lock, và Wait? = 'Yes'

## □ Quản lý unlock

### ➤ T unlock A:

- Xóa T trong danh sách liên quan đến A.
- Cập nhật lại Group mode cho phù hợp.
- Cấp một/ một số lock cho các GT chờ
  - First-come-first-served: no starvation.
  - Priority to shared lock: cấp tất cả các shared lock đang chờ, sau đó mới đến update lock (nếu có). Chỉ cấp exclusive lock nếu không có T nào chờ với mode khác. Có starvation, khi cho U hoặc X lock.
  - Priority to updating: ưu tiên các GT giữ Ulock và đang chờ Xlock.



## 6. MỨC CÔ LẬP CỦA GIAO TÁC

# **Các mức độ cô lập (Isolation levels)**

- ❑ **Mức độ cô lập của 1 giao tác quy định mức độ nhạy cảm của 1 giao tác đối với những sự thay đổi trên CSDL do các giao tác khác tạo ra.**
- ❑ **Mức độ cô lập của giao tác quy định cách thức lock và thời gian giữ lock trên đơn vị dữ liệu mà giao tác có truy cập.**
- ❑ **Các mức độ cô lập được SQL Server hỗ trợ:**
  - **Read Uncommitted.**
  - **Read Committed (default).**
  - **Repeatable Read.**
  - **Serializable.**

# Các mức cô lập

- ❑ Mỗi transaction đều có 1 trong 4 mức độ cô lập nêu trên.
- ❑ Lệnh T-SQL thiết lập mức độ cô lập:
- ❑ *SET TRANSACTION ISOLATION LEVEL <READ COMMITTED/READ UNCOMMITTED/REPEATABLE READ/SERIALIZABLE>*

# Read Uncommitted

Không thiết lập Shared Lock trên những đơn vị dữ liệu cần đọc, có thiết lập Exclusive lock khi ghi.

Không bị ảnh hưởng bởi những lock của các giao tác khác trên những đơn vị dữ liệu cần đọc.

Không phải chờ khi đọc dữ liệu (kể cả khi dữ liệu đang bị lock bởi giao tác khác).

## Ưu điểm:

Tốc độ xử lý rất nhanh, khắc phục được Lost Update.

Không cản trở những giao tác khác thực hiện việc cập nhật dữ liệu.

## Khuyết điểm:

Các vấn đề gặp phải khi xử lý đồng thời: Dirty Reads, Unrepeatable Read, Phantoms.

## Nhận xét:

Chỉ nên dùng để đọc dữ liệu trong trường hợp cần dữ liệu tổng quan về CSDL, ví dụ như tạo những báo cáo về tình hình chung.

Không dùng khi cần đọc những số liệu chính xác.

# Read Committed

Là MCL mặc định của SQL Server

Tạo Shared Lock trên đvdl được đọc, Shared Lock được giải phóng ngay sau khi đọc xong dữ liệu => Giải quyết vấn đề Dirty Reads

Tạo Exclusive Lock trên đvdl được ghi, và giữ cho đến hết giao tác

## Ưu điểm:

Giải quyết vấn đề Lost Update, Dirty Reads

Shared Lock được giải phóng ngay, không cần phải giữ cho đến hết giao tác nên không ngăn cản thao tác cập nhật của các giao tác khác.

## Khuyết điểm:

Chưa giải quyết được vấn đề Unrepeatable Reads, Phantoms

Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đơn vị dữ liệu đang bị giữ lock bởi giao tác khác.

# Repeatable Read

**Repeatable Read = Read Committed+ Giải quyết Unrepeatable Reads**

**Tạo Shared Lock trên đvdl được đọc, Shared Lock được giữ cho đến hết giao tác => Không cho phép các giao tác khác cập nhật trên đvdl này.**

**Tạo Exclusive Lock trên đvdl được ghi, Exclusive Lock được giữ cho đến hết giao tác.**

## **Ưu điểm:**

**Giải quyết vấn đề Lost Update, Dirty Read và Unrepeatable Read**

## **Khuyết điểm:**

**Chưa giải quyết được vấn đề Phantom**

**Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đơn vị dữ liệu đang bị giữ lock bởi giao tác khác.**

**Các giao tác khác không được phép cập nhật trên những đơn vị dữ liệu đang bị giữ Shared Lock.**

**Vẫn cho phép Insert những dòng dữ liệu thỏa mãn điều kiện thiết lập những Shared Lock => Phantoms**

# Serializable

**Serializable = Repeatable Read + Giải quyết Phantom**

**Giải quyết cả 4 vấn đề của truy xuất đồng thời**

**Tạo Shared Lock trên đvdl được đọc, Shared Lock được giữ cho đến hết giao tác => Không cho phép các giao tác khác cập nhật trên đvdl này.**

**Không cho phép Insert những dòng dữ liệu thỏa mãn điều kiện thiết lập những Shared Lock**

**Tạo Exclusive Lock trên đvdl được ghi, Exclusive Lock được giữ cho đến hết giao tác.**

**Ưu điểm:**

**Giải quyết được cả 4 vấn đề của TXĐT**

**Khuyết điểm:**

**Phải chờ khi chưa thể được đáp ứng yêu cầu lock trên đvdl đang bị giữ lock bởi giao tác khác.**

# Thử nghiệm các mức cô lập

- ❑ Vào SQL Query Analyzer, tạo 2 connection trên 2 cửa sổ riêng biệt trong Query Analyzer, mỗi connection ứng với 1 giao tác.
- ❑ Trong mỗi giao tác, sử dụng lệnh WAITFOR DELAY để yêu cầu 1 giao tác tạm dừng xử lý. Cú pháp:

WAITFOR DELAY 'hh:mm:ss'

- ❑ Giả sử có bảng SINHVIEN (MASV, TEN)

MASV	TEN
1	Nam
2	Toan
3	Tam



## TH1: a. Read UnCommitted & Read Committed

T1	T2
<pre>UPDATE SINHVIEN SET  TEN = 'Minh' WAITFOR DELAY '00:00:20'  ROLLBACK TRAN</pre>	<pre>BEGIN TRAN SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED SELECT * FROM SINHVIEN WHERE TEN = 'Minh' COMMIT TRAN</pre>

T2 cho ra tất cả các dòng của bảng SINHVIEN

## TH1: b. Read UnCommitted & Read Committed

T1	T2
<pre>UPDATE SINHVIEN SET  TEN = 'Minh' WAITFOR DELAY '00:00:20'  ROLLBACK TRAN</pre>	<pre>BEGIN TRAN SET TRANSACTION ISOLATION LEVEL <i>READ COMMITTED</i> SELECT * FROM SINHVIEN WHERE TEN = 'Minh' COMMIT TRAN</pre>

T2 không cho dòng nào của bảng SINHVIEN

## TH2: a. Read Committed & Repeatable Read

T1	T2
<pre>BEGIN TRAN  SET TRANSACTION ISOLATION LEVEL READ COMMITTED SELECT TEN FROM SINHVIEN WAITFOR DELAY '00:00:20'  SELECT TEN FROM SINHVIEN COMMIT TRAN</pre>	<pre>BEGIN TRAN  UPDATE SINHVIEN SET TEN= 'Minh' COMMIT TRAN</pre>

Kết quả 2 câu SELECT của T1 là khác nhau.

## TH3: a. Repeatable Read & Serializable

T1	T2
<pre>BEGIN TRAN  SET TRANSACTION ISOLATION LEVEL REPEATABLE READ  SELECT TEN FROM SINHVIEN  WAITFOR DELAY '00:00:20'        SELECT TEN FROM SINHVIEN COMMIT TRAN</pre>	<pre>BEGIN TRAN        INSERT INTO SINHVIEN VALUES ('4', 'Tuyet') COMMIT TRAN</pre>

**Kết quả 2 câu SELECT của T1 là khác nhau.**

## TH3: a. Repeatable Read & Serializable

T1	T2
<pre>BEGIN TRAN  SET TRANSACTION ISOLATION LEVEL SERIALIZABLE  SELECT TEN FROM SINHVIENT  WAITFOR DELAY '00:00:20'         SELECT TEN FROM SINHVIENT COMMIT TRAN</pre>	<pre>BEGIN TRAN         INSERT INTO SINHVIENT VALUES ('4', 'Tuyet') COMMIT TRAN</pre>

Kết quả 2 câu SELECT của T1 là như nhau.

## 7. DEADLOCK

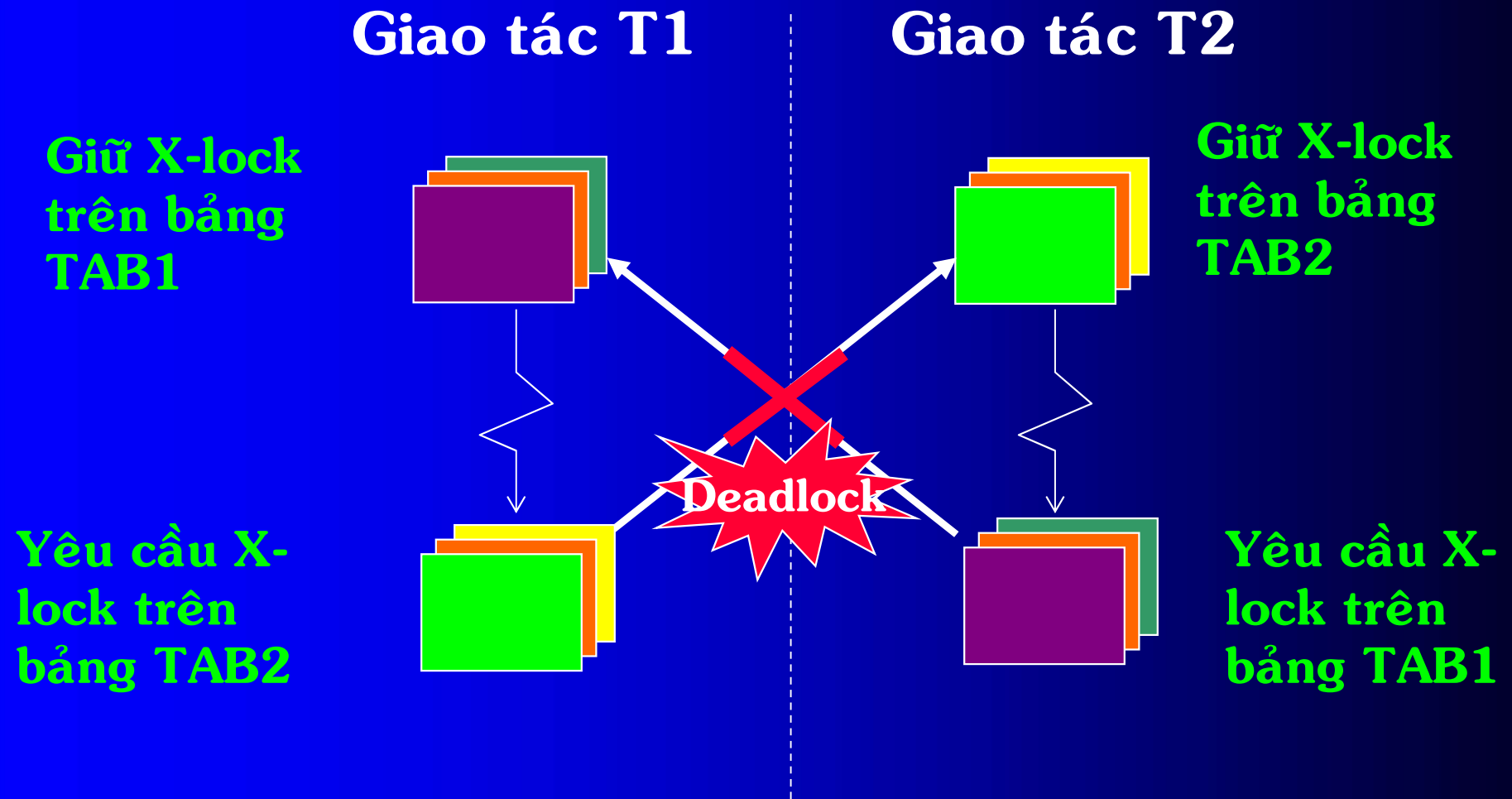
- **Khái niệm**
- **Phát hiện deadlock**
- **Ngăn ngừa deadlock**

# Deadlock

- ❑ **Deadlock là tình trạng trong đó những giao tác có liên quan không thể thực hiện tiếp các thao tác của nó mà phải chờ nhau mãi.**
- ❑ **Đây là tình trạng không mong muốn ở các hệ quản trị CSDL điều khiển đồng thời dùng cơ chế khóa.**
- ❑ **Các tình huống xảy ra Deadlock:**
  - **Cycle deadlock**
  - **Conversion deadlock**

# Cycle deadlock

- T1 giữ X-lock trên TAB1, T2 giữ X-lock trên TAB2
- T1 yêu cầu X-Lock trên TAB2 => T1 chờ T2
- T2 yêu cầu X-Lock trên TAB1 => T2 chờ T1



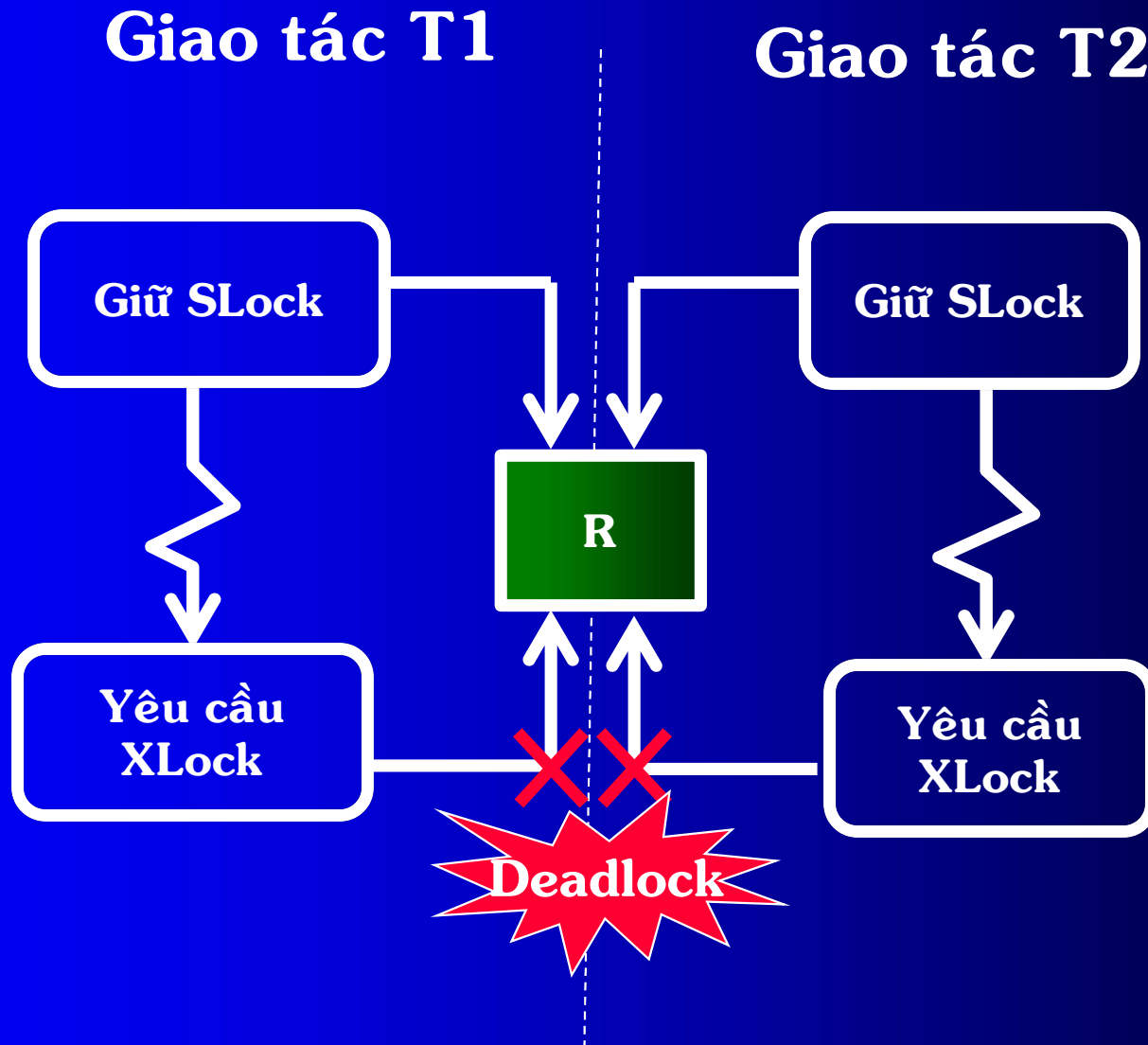


# Conversion Deadlock

Giao tác T1 và T2 cùng giữ S-Lock trên 1 tài nguyên R

Giao tác T1 yêu cầu X-Lock trên R => T1 chờ T2

Giao tác T2 yêu cầu X-Lock trên R => T2 chờ T1



# Giải pháp cho Deadlock

## ❑ Giải quyết Deadlock

- Hủy tất cả=> không phải là cách giải quyết tốt.
- Hủy giao tác gây ra deadlock. Giao tác nào gây ra?
  - Dùng thời gian quá hạn (timeout), giới hạn thời gian giao tác ở trạng thái kích hoạt, thời gian timeout nhỏ quá hoặc lớn quá đều không có lợi.
  - Dùng đồ thị chờ.

## ❑ Ngăn ngừa Deadlock

- Sắp xếp các đơn vị dữ liệu theo 1 thứ tự cố định và các giao tác yêu cầu lock trên chúng theo thứ tự này.
  - Các transaction chờ lẫn nhau => deadlock. Các transaction chờ theo 1 chiều nhất định => ngăn ngừa deadlock
- Dò tìm deadlock dùng nhãn thời gian:
  - Thuật toán WAIT - DIE
  - Thuật toán WOUND - WAIT

# Đồ thị chờ

- ❑ Khi có tình trạng Deadlock xảy ra, hệ thống hủy tình trạng Deadlock, thực hiện lúc runtime.
- ❑ Dùng đồ thị chờ để phát hiện deadlock
  - Cho S là lịch thao tác của các giao tác  $T_1, T_2, \dots, T_n$ .
  - Đồ thị có đỉnh là các giao tác
  - Cung có hướng  $T_i \rightarrow T_j$  nếu  $T_j$  phải chờ  $T_i$
  - Đồ thị có chu trình  $\Leftrightarrow$  Deadlock
- ❑ Để giải quyết: Hủy đỉnh (ứng với giao tác) có nhiều cung vào ra nhất.

# Ví dụ

T1	T2	T3
Rlock(A)		
	Rlock(C)	
		Wlock(E)
Wlock (B)		
	Rlock(B)	
		Rlock(B)
Wlock(C)		
	Wlock(E)	
	Rlock(D)	
		Wlock(C)
...	...	...

**Có xảy ra Deadlock không?**

# **Dò tìm Deadlock dùng nhãn thời gian**

- ❑ Nhãn thời gian này chỉ dùng cho việc dò tìm deadlock, không giống như nhãn thời gian dùng cho việc điều khiển đồng thời, mặc dù hệ thống có thể đang dùng pp điều khiển đồng thời dựa trên nhãn thời gian.
- ❑ Nhãn thời gian dùng cho việc dò tìm deadlock không thay đổi khi giao tác rollback.
- ❑ Nhãn thời gian cho biết thời điểm T đang chờ lock trên đvdl giữ bởi giao tác khác.

# Thuật toán WAIT - DIE

$T_i, T_j$  có timestamp  $t_{T_i}, t_{T_j}$ .  $T_i$  yêu cầu lock trên 1 đvdl đang bị giữ lock bởi  $T_j$

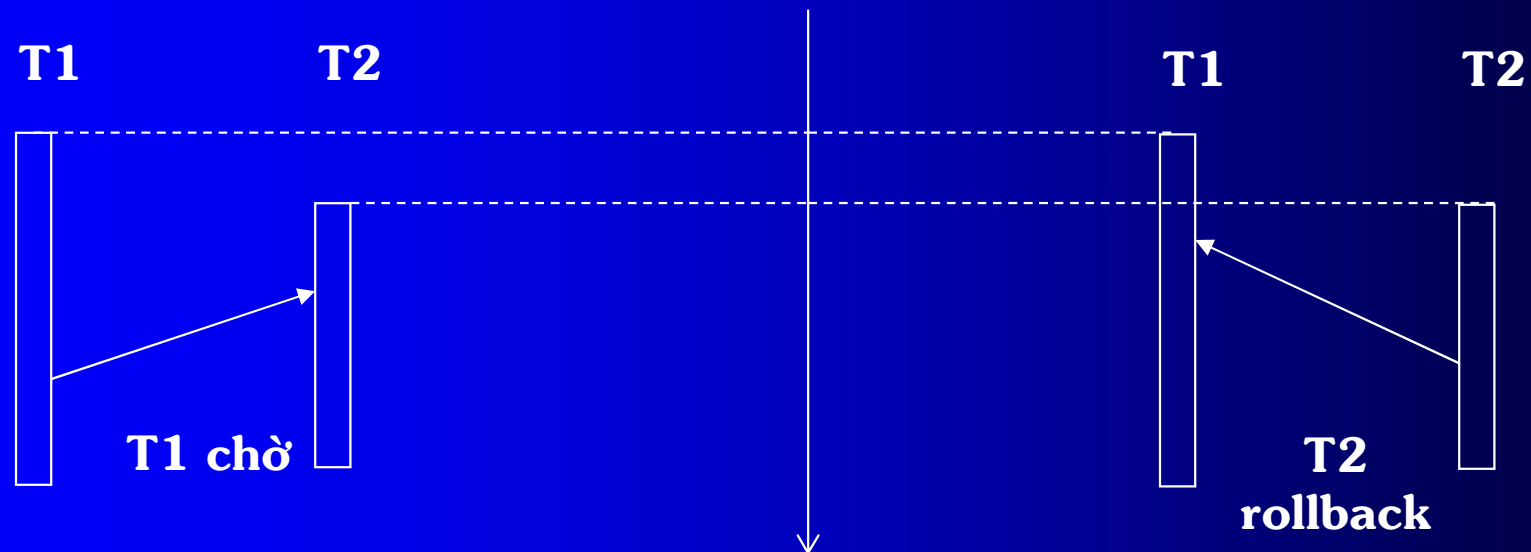
*If  $t_{T_i} < t_{T_j}$  then*

*$T_i$  chờ*

*Else*

*Rollback  $T_i$ ,  $T_i$  biết nữa àu laii*

*EndIf*



# Nhận xét TT WAIT- DIE

- Ưu tiên cho các transaction già (bắt đầu trước, timestamp nhỏ)
- Transaction thực hiện trước chỉ phải chờ khi tranh chấp lock với transaction được thực hiện sau.
- Transaction thực hiện sau phải bị rollback và thực hiện lại khi tranh chấp lock với transaction được thực hiện trước.

# Thuật toán WOUND - WAIT

$T_i, T_j$  có timestamp  $t_{T_i}, t_{T_j}$ .  $T_i$  yêu cầu lock trên 1 đvdl đang bị giữ lock bởi  $T_j$

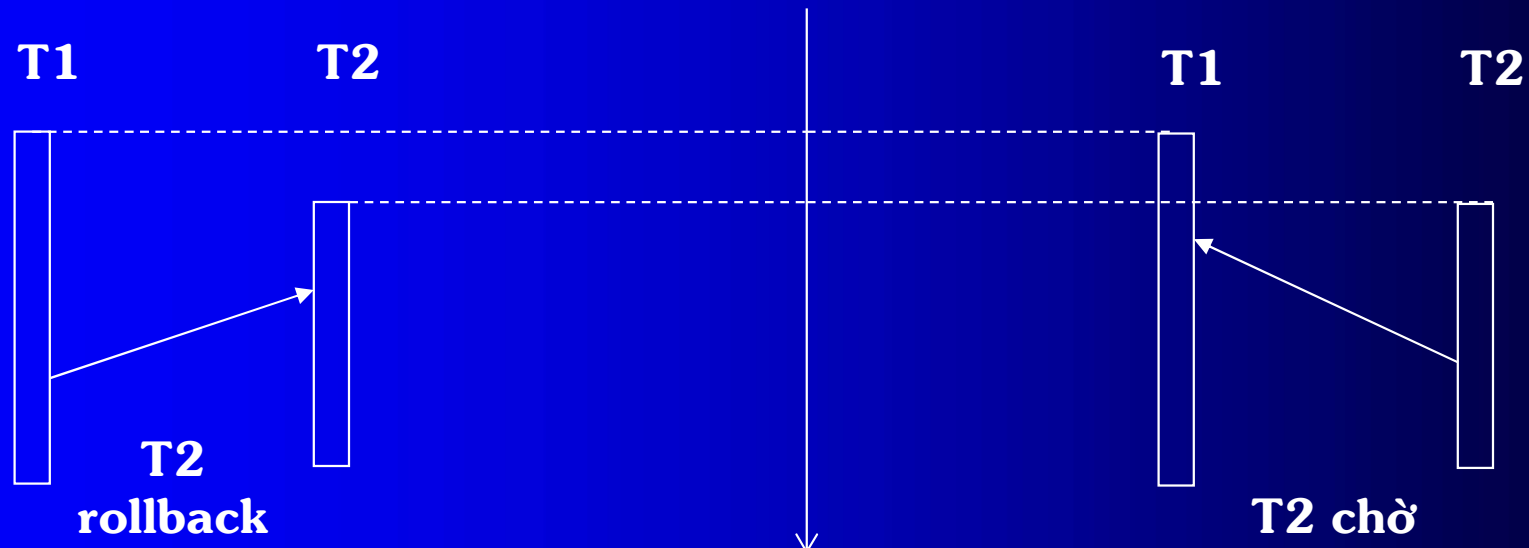
If  $t_{T_i} < t_{T_j}$  then

Rollback  $T_j$

Else

$T_i$  phaûi chờ

EndIf





# Nhận xét TT WOUND - WAIT

- Ưu tiên cho các transaction già (bắt đầu trước, timestamp nhỏ)
- Transaction thực hiện trước không bao giờ bị rollback hay chờ khi tranh chấp lock với 1 transaction thực hiện sau nhưng đang giữ lock.
- Transaction thực hiện sau phải chờ khi tranh chấp lock với transaction thực hiện trước và phải bị rollback khi bị transaction thực hiện trước tranh chấp lock.

# Các pp quản lý deadlock

PP dùng nhãn thời gian		ĐT chờ
Wait - Die	Wound-Wait	
GT rollback ở thời điểm xảy yêu cầu lock, là giai đoạn sớm, nên có thể có nhiều GT bị rollback hơn, và GT rollback thường thực hiện ít công việc hơn.	Nếu GT yêu cầu lock gần thời điểm GT bắt đầu, ít khi xảy ra tình trạng GT già không yêu cầu được lock giữ bởi GT trẻ hơn, vì vậy rollback ít xảy ra. GT bị rollback đã thực hiện nhiều việc trước khi bị rollback.	Đồ thị có thể rất lớn, phân tích và tìm chu trình sẽ tốn nhiều thời gian.
Có khi không có deadlock xảy ra vẫn yêu cầu GT rollback.		Chỉ yêu cầu 1 GT rollback khi thực sự GT gây ra deadlock
Ưu tiên GT “già”, GT “già” hơn sẽ kill giao tác “trẻ” hơn. Đảm bảo mọi GT đều hoàn tất, không có tình trạng starvation.		
Dễ cài đặt hơn đồ thị chờ		Cài đặt khó, đặc biệt đối với hệ thống phân tán.

## 8. CÁCH SỬ DỤNG CÁC PHƯƠNG THỨC KHÓA

# Chiến lược sử dụng lock

- ❑ Các phương thức lock:
  - FASTFIRSTROW
  - HOLDLOCK
  - NOLOCK
  - PAGLOCK
  - READCOMMITTED
  - READPAST
  - READUNCOMMITTED
  - REPEATABLEREAD
  - ROWLOCK
  - SERIALIZABLE
  - TABLOCK
  - TABLOCKX
  - UPDLOCK
  - XLOCK

# Ví dụ

❑ `SELECT COUNT(*) FROM SINHVIEN WITH (TABLOCK, HOLDLOCK)`

❑ Các kiểu dùng chung:

`(TABLOCK, XLOCK)` : khóa độc quyền trên bảng.

`(ROWLOCK, XLOCK)` : khóa độc quyền trên dòng.

# Ví dụ

GP (STT, NGÀY CAP, SOXE, LYDOCAP)

Giả sử STT  
lớn nhất hiện tại = 10

Số thứ tự bắt đầu từ 1, tăng dần, không nhảy số






Thao tác	Thủ tục CGP
S1	Begin Tran
S2	M= Select STT from GP where STT = (Select max (STT) from GP)
S3	Insert into GP values (M+1,...)
S4	Commit tran

U1 gọi CGP	U2 gọi CGP
S1	
	S1
S2	
10	
	S2
	10
S3	
11	
	S3
	Lỗi! 11
S4	
	S4

# Giải pháp ĐKĐT

GP (STT, NGÀY CAP, SOXE, LYDOCAP)

Thao tác	Thủ tục CGP
S1	Begin Tran
S2	M= Select STT from GP with (rowlock, xlock) where STT = (Select max (STT) from GP)
S3	Insert into GP values (M+1,...)
S4	Commit tran

U1 gọi CGP	U2 gọi CGP
S1	
	S1
S2 	
	S2 
S3 	
S4	
	S2 
	S3 ✓ 
	S4

# Bài tập

□ Cho CSDL sau:

SV (MASV, TENS<sub>V</sub>, MANGANH)

NGANH (MANGANH, TEN<sub>NGANH</sub>)

CD (MACĐ, TEN<sub>CD</sub>, SOS<sub>VTD</sub>)

NG\_CD (MANGANH, MACĐ)

CD\_MO (MACĐ, NAM, HOCKY)

ĐK (MASV, MACĐ, NAM, HOCKY)

Mỗi khi chuyên đề có mã là MACĐ được mở, số sv đăng ký không vượt quá SOS<sub>VTD</sub>

Cho biết sinh viên theo học ngành có mã là MANGANH phải học những chuyên đề gì

1 bộ của CD\_MO cho biết chuyên đề có mã là MACĐ được mở trong năm NAM, học kỳ HOCKY

- Sinh viên chỉ được đăng ký các chuyên đề có mở
- SV không được đk quá 3 chuyên đề trong 1 học kỳ.



# Giao tác đăng ký học chuyên đề

1. **Thủ tục Đăng\_ky\_CD (@msv, @mcd, @nam, @hk)**
2. @ng = ngành mà sv mã là @msv theo học (SV)
3. @mcd thuộc ds các cd mà ngành @ng phải học? (NG\_CD)  
Nếu không thỏa => Báo lỗi và Rollback tran
4. **KT CD @mcd có mở trong hk @hk, năm @nam**  
Nếu không mở => Báo lỗi và Rollback tran
5. @max = số sv tối đa có thể cho đk học cd mã @mcd (CD)
6. @sosv = đếm số sv đã đk cd @mcd, vào năm @nam, học kỳ @hk
7. Nếu @sosv < @max  
Insert into ĐK (@msv, @mcd, @nam, @hk)  
Ngược lại => Báo hết chỗ và Rollback tran
8. **Kết thúc thủ tục**  
Đưa ra giải pháp sao cho hệ thống làm việc đồng thời hiệu quả.  
(Gt thống kê, cập nhật, thêm mới, đăng ký đồng thời)

# Thảo luận

- ❑ Đv các gt đơn giản: thêm, cập nhật, xóa
- ❑ Đv các giao tác thống kê.
- ❑ Đv giao tác đăng ký.

Khóa độc quyền trên các bảng lq?

Áp dụng mức cô lập Serializable?

- ❑ Hãy cho biết giải pháp?

## 9. ĐIỀU KHIỂN ĐỒNG THỜI DÙNG KỸ THUẬT NHẢY THỜI GIAN

# Ý tưởng

- ❑ Gán nhãn thời gian cho giao tác, ghi lại nhãn thời gian của giao tác cuối cùng đọc/ghi trên đvdl.
- ❑ Quản lý đồng thời theo nguyên tắc: đảm bảo việc thực hiện lịch biểu đã cho tương đương với lịch tuần tự theo thứ tự nhãn thời gian của giao tác.

# Nhãn thời gian (timestamp) của giao tác

- Nhãn thời gian của giao tác  $T$ ,  $TS(T)$ , được phát sinh bởi bộ lập lịch.
  - Dùng một giá trị số gắn vào các giao tác để chỉ thời điểm bắt đầu của giao tác.
    - Sử dụng đồng hồ hệ thống hoặc bộ đếm để tạo nhãn thời gian.
  - Nhãn thời gian có tính chất duy nhất và tăng dần.  
( $T_i < T_j$  thì  $TS(T_i) < TS(T_j)$ )
- Nếu  $TS(T_i) < TS(T_j)$  thì bộ lập lịch phải đảm bảo rằng lịch biểu đã cho phải tương đương với lịch biểu tuần tự  $T_i < T_j$ .

# Nhãn thời gian của đơn vị dữ liệu

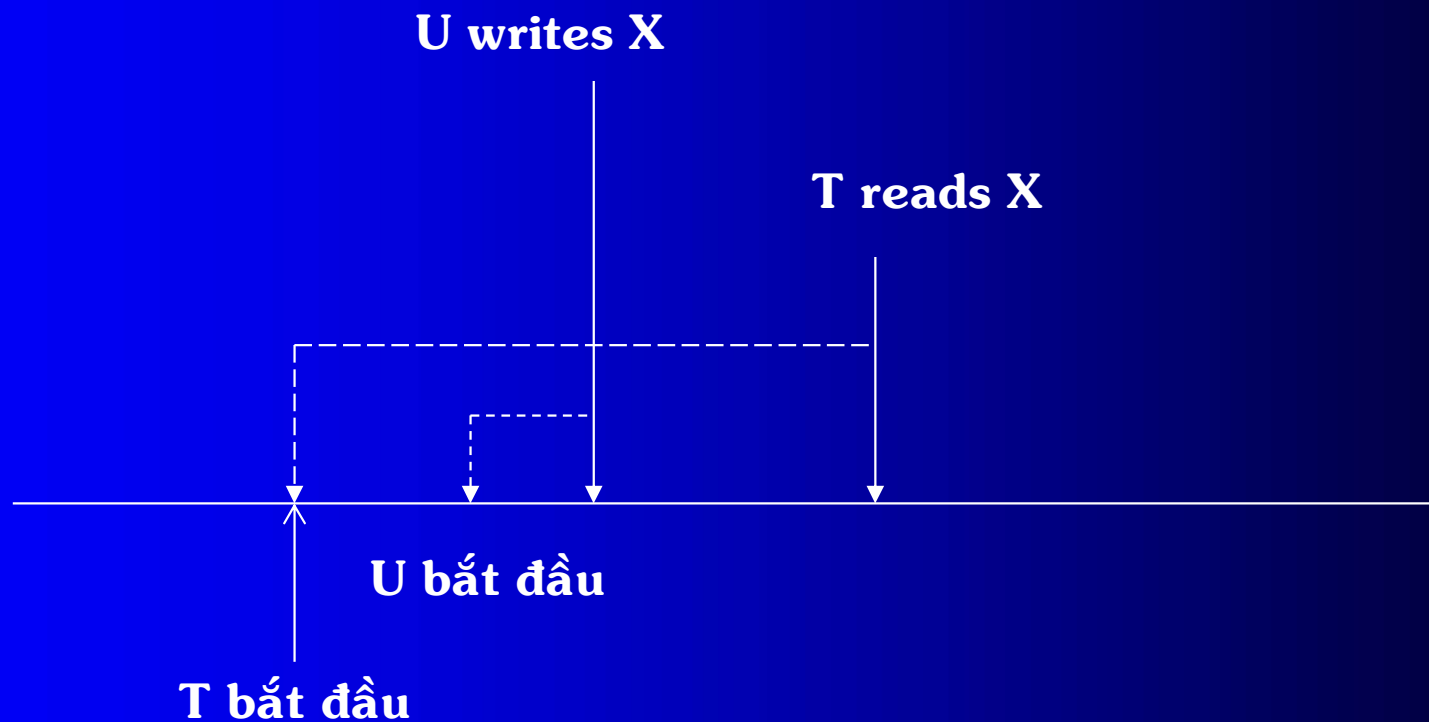
- Các đơn vị dữ liệu được gán cho 1 giá trị số gọi là nhãn thời gian của đơn vị dữ liệu.
  - $RT(X)$ , thời điểm đọc  $X$ , là nhãn thời gian lớn nhất của giao tác đã đọc  $X$ .
  - $WT(X)$ , thời điểm ghi  $X$ , là nhãn thời gian lớn nhất của giao tác đã ghi  $X$ .
  - $C(X)$ , commit bit của  $X$ , = true nếu giao tác vừa mới thực hiện  $write(X)$  đã commit.
    - $C(X)$  nhằm tránh tình trạng một giao tác  $T$  đọc dữ liệu ghi bởi giao tác  $U$  nhưng sau đó  $U$  abort.

# Nguyên tắc

- Truy xuất theo thứ tự nhãn thời gian
  - Khi T đọc X thì  $WT(X)$  phải  $< TS(T)$ .
  - Khi T ghi X thì  $WT(X)$  và  $RT(X)$  phải  $< TS(T)$ .
  - Abort T nếu không tuân theo thứ tự, T khởi tạo lại với nhãn thời gian mới.

# Vấn đề 1: Đọc quá trễ

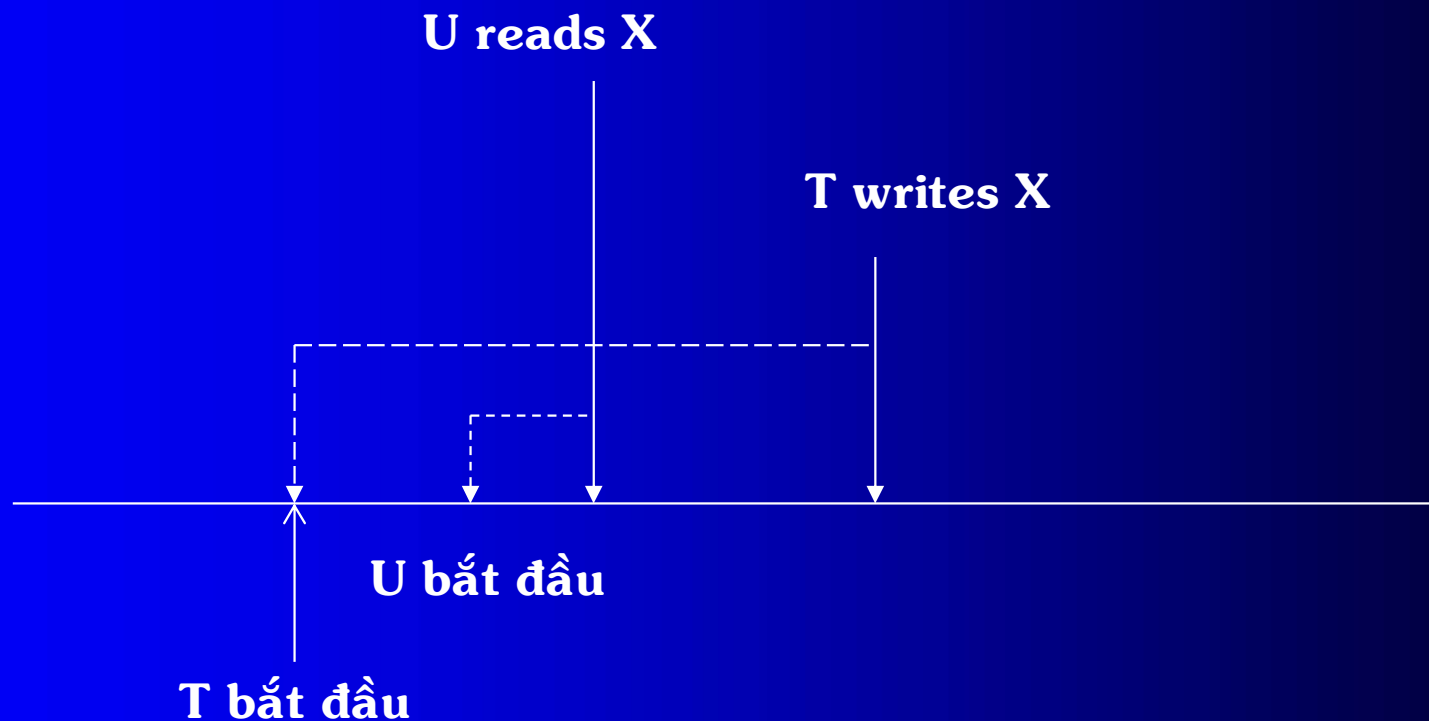
- T muốn đọc X
- Nhưng  $TS(T) < WT(X) = TS(U)$  : X đã được ghi sau khi T đã được khởi tạo.





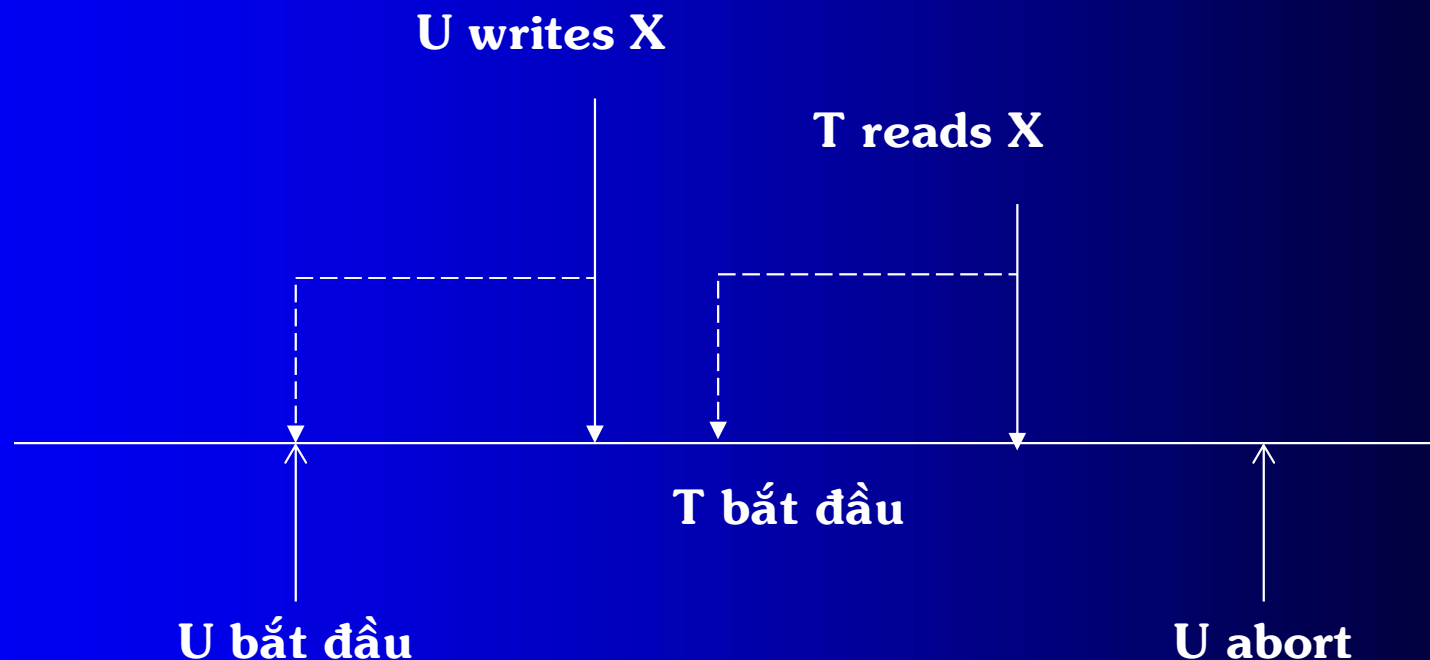
## Vấn đề 2: Ghi quá trễ

- T muốn ghi X
- Nhưng  $WT(X) < TS(T) < RT(X)$ : có một giao tác khác lẽ ra nên đọc giá trị ghi bởi T nhưng đã đọc giá trị khác



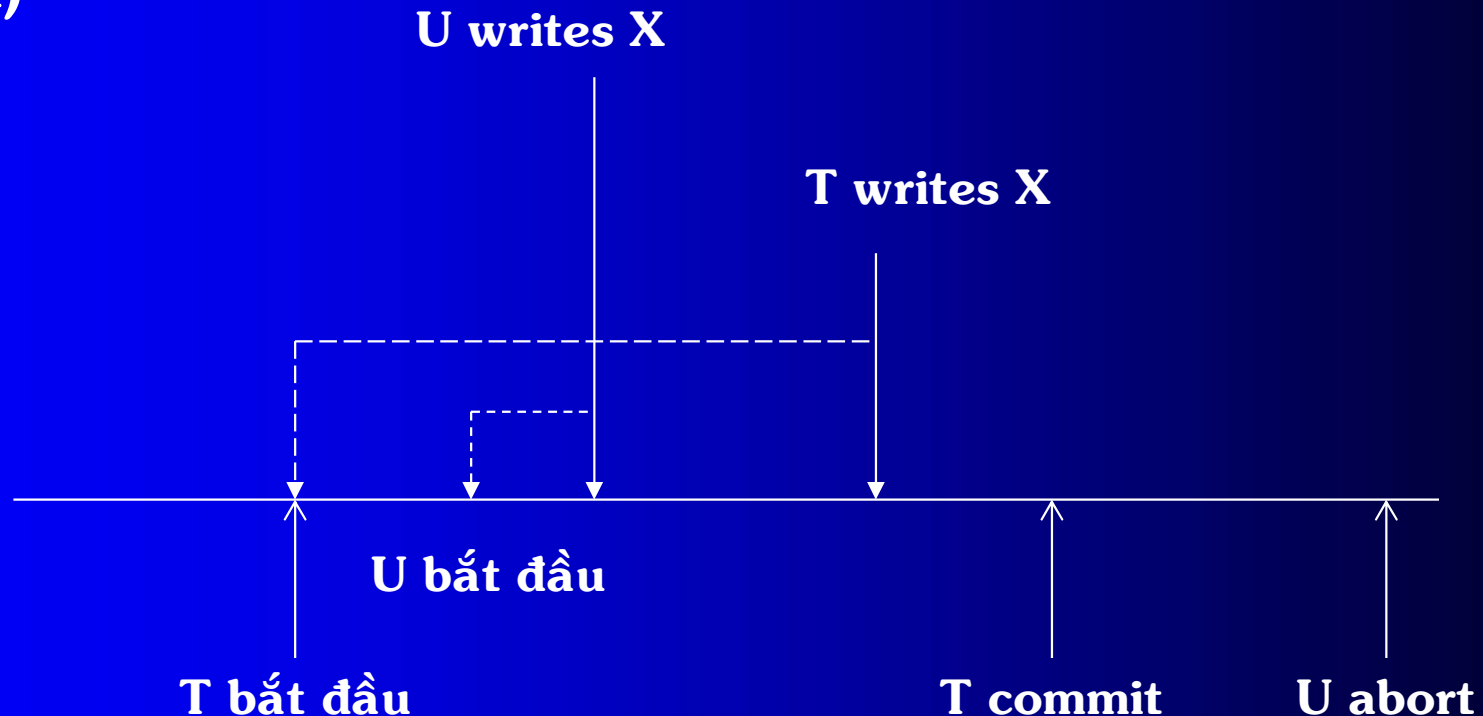
# Vấn đề 3: Dirty read

- T đọc X, X được ghi lần cuối bởi U.
- Nhưng T vừa đọc X xong, U abort
  - Trì hoãn việc đọc của T cho tới khi U commit/abort.
  - Dựa vào commit bit để nhận biết.



# Vấn đề 4

- ❑ U bắt đầu sau nhưng lại ghi trước
- ❑ Khi T muốn ghi thì không cần làm gì cả (Qui tắc ghi Thomas)
- ❑ Nếu U abort và T commit
  - Giá trị cuối của X lẽ ra là giá trị do T ghi nhưng đã bị bỏ qua.
  - Khi U abort, đặt  $C(X) = \text{false}$ , chép lại giá trị cũ của X và  $WT(X)$



# Nguyên tắc

- Khi T yêu cầu Read/Write, bộ lập lịch phản hồi:
  - Đáp ứng yêu cầu.
  - Abort T và khởi tạo T với nhãn thời gian mới (Rollback).
  - Trì hoãn T và sau đó quyết định Abort T hay đáp ứng yêu cầu

# Thuật toán điều khiển

## 1. T yêu cầu Read X

### ➤ Nếu $TS(T) \geq WT(X)$

- Nếu  $C(X) = \text{TRUE}$ , đáp ứng yêu cầu

- Nếu  $TS(T) > RT(X)$

- $RT(X) := TS(T)$

- Ngược lại không thay đổi  $RT(X)$

- Nếu  $C(X) = \text{FALSE}$ , trì hoãn T cho đến khi  $C(X)$  trở thành TRUE hay giao dịch đã ghi X abort.

### ➤ Nếu $TS(T) < WT(X)$

- Rollback T (Đọc quá trễ)

# Thuật toán điều khiển

## 2. Giao dịch T yêu cầu Write X

- Nếu  $TS(T) \geq RT(X)$  and  $TS(T) \geq WT(T)$ 
  - Ghi giá trị mới trên X
  - Đặt  $WT(T) := TS(T)$
  - Đặt  $C(X) := FALSE$ .
- Nếu  $TS(T) \geq RT(X)$  and  $TS(T) < WT(T)$ 
  - Nếu  $C(X) = TRUE$ , bỏ qua việc ghi của T.
  - Nếu  $C(X) = FALSE$ , trì hoãn T
- Nếu  $TS(T) < RT(X)$ 
  - Rollback T (Ghi quá trễ)

3. Nếu T commit, bộ lập lịch đặt tất cả các đvdl X mà T ghi giá trị  $C(X) := TRUE$ . Các GT đang chờ trên X có thể tiếp tục.

4. Trường hợp T rollback, các GT khác đang chờ trên X yêu cầu đọc/ ghi lại.

# Ví dụ

T1	T2	T3	T4	A	B	C
420	400	425	415	tr=tw=0	tr=tw=0	tr=tw=0
			ReadA	tr=415		
ReadA				tr =420		
			WriteB		tw=415	
WriteA				tw=420		
	ReadB				T2 rollback	
		ReadB			tr=425	
	ReadA					
	WriteC					
		WriteA		tw=425		

# Ví dụ

T1	T2	T3	T4	A	B	C
<b>510</b>	<b>550</b>	<b>575</b>	<b>500</b>	tr=tw=0	tr=tw=0	tr=tw=0
			ReadA	tr=500		
ReadA				tr =510		
			WriteB		tw=500	
WriteA				tw=510		
	ReadB				tr=550	
		ReadB			tr=575	
	ReadA			tr=550		
	WriteC					Tw=550
		WriteA		tw=575		



# PP nhãn thời gian đa phiên bản

- Là một biến thể của pp nhãn thời gian.
- Ngoài phiên bản mới nhất của đvdl được lưu lại, còn có các phiên bản trước đó.
- Mục đích: giúp cho thao tác đọc không bao giờ làm giao tác rollback.
  - Hỏi: Khi nào thì thao tác đọc làm giao tác rollback?
  - PP nhãn thời gian đa phiên bản sẽ làm giao tác T thay vì abort vì lý do trên sẽ tiếp tục đọc phiên bản (của đvdl cần đọc) phù hợp với nhãn thời gian của T.

# PP nhãn thời gian đa phiên bản

- Mỗi đvdl  $X$  có nhiều phiên bản  $X_1, X_2, \dots, X_k$ .
- Đối với mỗi phiên bản, lưu lại:
  - Giá trị của phiên bản  $X_i$ ,
  - Nhãn thời gian đọc của  $X_i$  :  $r_T(X_i)$ , là nhãn thời gian lớn nhất trong các GT đã đọc  $X_i$ .
  - Nhãn thời gian ghi của  $X_i$   $w_T(X_i)$ , là nhãn thời gian của giao tác đã tạo ra phiên bản  $X_i$ .
- Khi  $T$  được phép ghi trên  $X$ , một phiên bản mới  $X_{k+1}$  được tạo, và:
  - $r_T(X_{k+1}) = w_T(X_{k+1}) = TS(T)$ .
- Khi  $T$  được phép đọc trên  $X_i$ , thì:
  - $r_T(X_i) = \max(r_T(X_i), TS(T))$

# PP nhãn thời gian đa phiên bản

- Để bảo đảm khả tuần tự, có 2 nguyên tắc sau phải được đảm bảo:

## 1. T yêu cầu ghi trên X:

- Trong các phiên bản của X có nhãn thời gian ghi  $\leq TS(T)$ , chọn phiên bản  $X_i$  có  $w_T(X_i)$  lớn nhất:
  - Nếu  $r_T(X_i) > TS(T)$  thì cho T rollback.
  - Nếu  $r_T(X_i) \leq TS(T)$ , thì:
    - Nếu  $TS(T) = w_T(X_i)$  thì giá trị của  $X_i$  bị ghi đè.
    - Nếu  $TS(T) > w_T(X_i)$  thì tạo phiên bản mới  $X_j$ ,  $r_T(X_j) = w_T(X_j) = TS(T)$

## 2. T yêu cầu đọc trên X:

- Trong các phiên bản của X có nhãn thời gian ghi  $\leq TS(T)$ , chọn phiên bản  $X_i$  có  $w_T(X_i)$  lớn nhất:
  - Trả về giá trị  $X_i$  cho giao tác T.
  - $r_T(X_i) = \max(r_T(X_i), TS(T))$

# PP nhận thời gian đa phiên bản

- Nếu T rollback thì tình trạng rollback dây chuyền có thể xảy ra.
- Để lịch biểu là phục hồi được, khi các GT ghi trên các đvdl mà T đã đọc commit thì T mới được phép commit.

## **10. Điều khiển đồng thời dùng phương pháp kiểm tra hợp lệ (Validation Technique)**

# Phương pháp kiểm tra hợp lệ

- KT khóa và KT nhãn thời gian là các PP điều khiển đồng thời bị quan: giả sử các GT sẽ dụng độ và tránh sự dụng độ này.
- KT khoá và KT nhãn thời gian điều khiển thực hiện đồng thời theo cú pháp, vì phân biệt đọc/ghi.
- PP kiểm tra hợp lệ là PP điều khiển đồng thời lạc quan.

# Phương pháp kiểm tra hợp lệ

## □ Cú cho GT thực hiện.

- Cập nhật của GT chỉ trên biến cục bộ, không cập nhật trực tiếp trên CSDL cho đến khi GT kết thúc.
- Khi GT thi hành xong, kỳ xác nhận kiểm tra các cập nhật của GT có vi phạm tính khả tuần tự không.
  - Nếu không, GT commit và CSDL được cập nhật thật sự từ biến cục bộ.
  - Nếu vi phạm, GT abort và khởi động lại.

# Phương pháp kiểm tra hợp lệ

## □ GT trải qua các kỳ:

### ➤ 1. Kỳ đọc (Read phase):

- Ti đọc các mục dl cần thiết.
- Các cập nhật chỉ thực hiện trên biến cục bộ, không phải trên CSDL.

### ➤ 2. Kỳ Kiểm tra hợp lệ (Validation phase):

- Kiểm tra có vi phạm tính khả tuần tự hay không để cập nhật CSDL.
  - Nếu vi phạm thì rollback.
  - Nếu không thì làm tiếp kỳ 3.

### ➤ 3. Kỳ ghi (Write phase):

- Ghi từ biến cục bộ xuống CSDL.



# Phương pháp kiểm tra hợp lệ

## □ Ghi nhận các nhãn thời gian:

- **START( $T_i$ )** – thời điểm  $T_i$  start
- **VAL( $T_i$ )** – thời điểm  $T_i$  thực hiện xong phase validate
- **Finish( $T_i$ )** – thời điểm  $T$  thực hiện xong phase finish.

# Phương pháp kiểm tra hợp lệ

## □ Bộ lập lịch theo dõi 3 tập hợp:

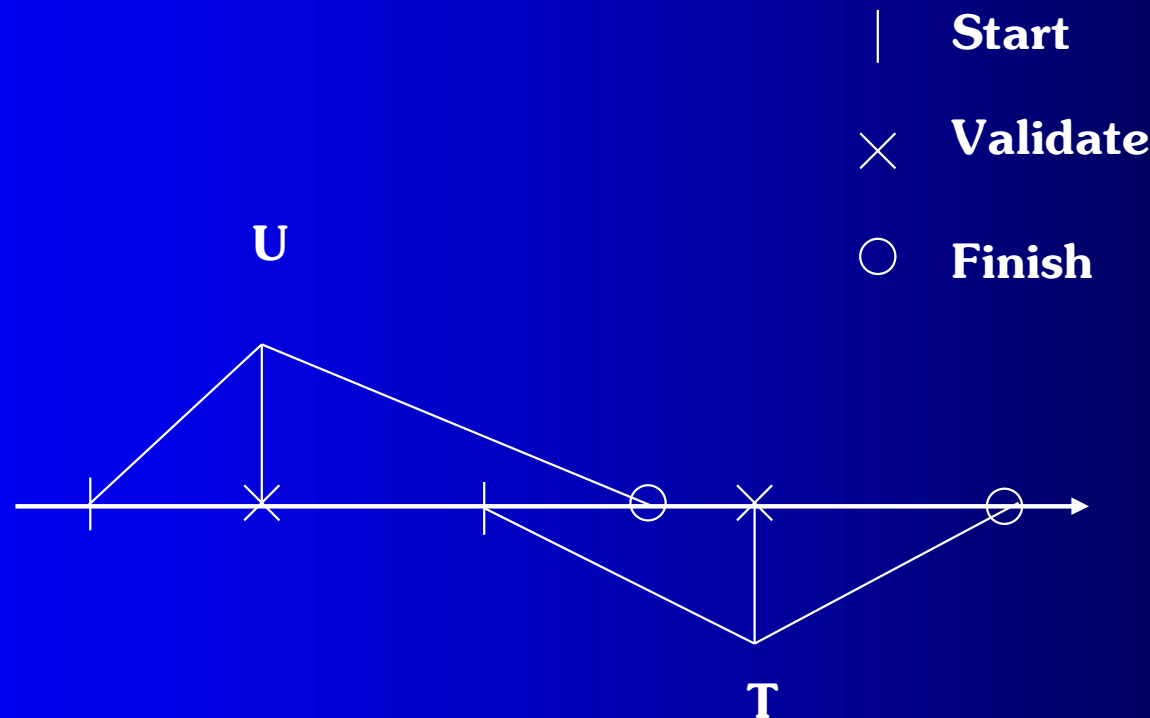
- $START = \{T, T \text{ đã start, nhưng chưa validate xong}\} + START(T)$
- $VAL = \{T, T \text{ validate xong nhưng chưa xong phase finish}\} + START(T) + VAL(T)$ 
  - $VAL(T)$  là thời điểm để căn cứ vào đó điều khiển tuần tự.
- $FIN = \{T, T \text{ thực hiện xong phase finish}\} + START(T), VAL(T), FIN(T)$ 
  - Bộ lập lịch không quan tâm đến GT T sao cho  $FIN(T) < START(U)$ , U là giao tác đang ở trạng thái kích hoạt.



# Kiểm tra hợp lệ

□ Giả sử có GT U:

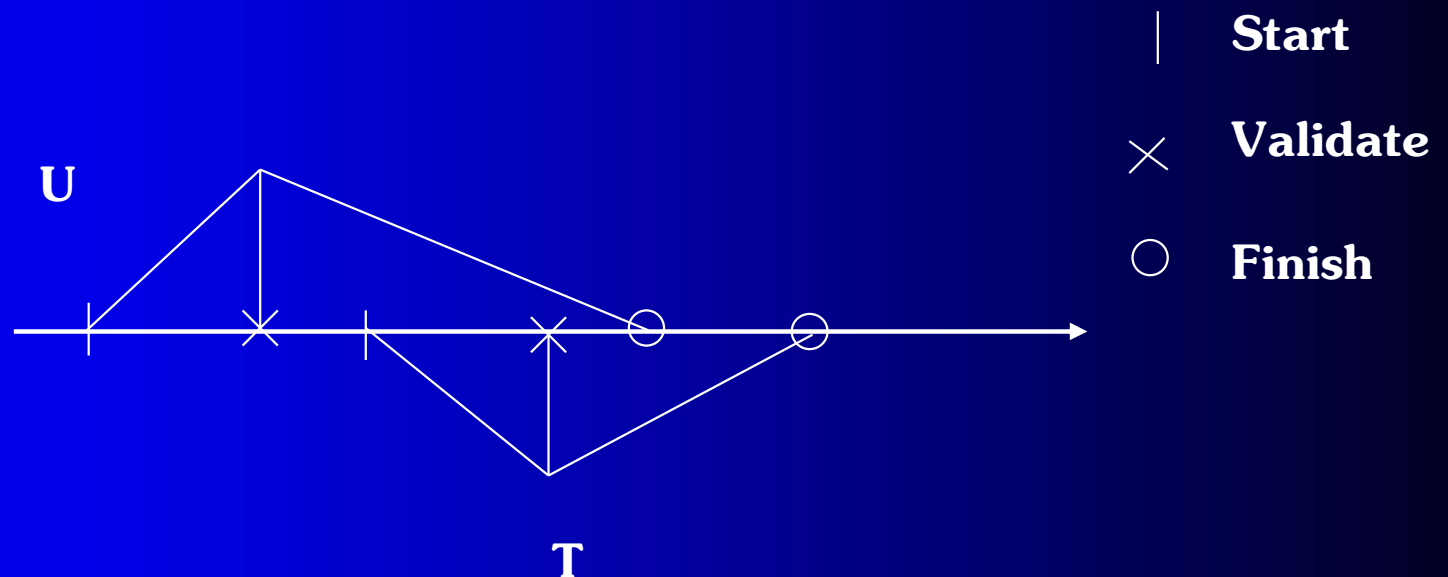
- $U \in \text{VAL}$  hoặc  $U \in \text{FIN}$
- $\text{FIN}(U) > \text{START}(T)$ , nghĩa là U không finish trước khi T start.
- $\text{RS}(T) \cap \text{WS}(U) = \{X\} \neq \emptyset$ : có thể U write X sau khi T read X. Ta không chắc T đã đọc giá trị U ghi trước đó hay không, tốt hơn hết cho T rollback để tránh rủi ro T và U không tuân theo thứ tự tuần tự.



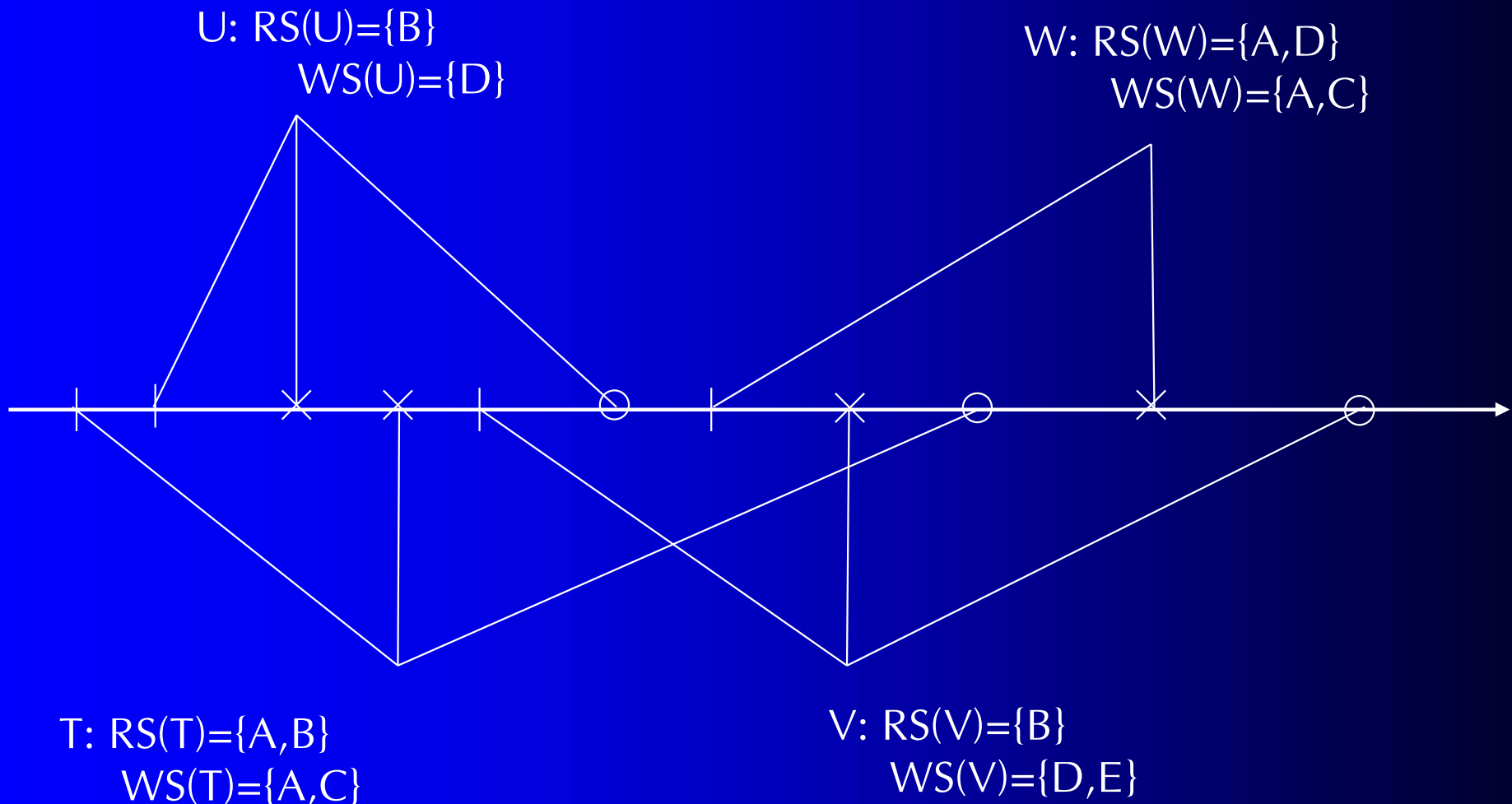
# Kiểm tra hợp lệ

## □ Giả sử có GT U:

- $U \in VAL$ , nghĩa là đã validate xong
- $FIN(U) > VAL(T)$ , nghĩa là U không finish trước khi T bắt đầu giai đoạn validate.
- $WS(T) \cap WS(U) = \{X\} \neq \emptyset$ : có thể T sẽ ghi X trước khi U ghi X. Ta không chắc điều này có xảy ra hay không, tốt hơn hết là rollback T để không vi phạm tính tuần tự.



# Ví dụ



**Hãy điều khiển đồng thời dùng phương pháp kiểm tra hợp lệ cho 4 GT trên.**

# Phương pháp kiểm tra hợp lệ

- Nói chung,  $TS(T_i) = \text{Validation}(T_i)$ . Kiểm tra hợp lệ cho  $T_j$ :
  - Với mọi  $T_i$  thỏa  $TS(T_i) < TS(T_j)$  thì một trong hai điều kiện sau phải thỏa:
    - $\text{Finish}(T_i) < \text{Start}(T_j)$ : nếu  $T_i$  xong trước khi  $T_j$  bắt đầu.
    - $\text{Start}(T_j) < \text{Finish}(T_i) < \text{Validation}(T_j)$ : Việc ghi của  $T_i$  và  $T_j$  không ghi đè. Thao tác ghi của  $T_j$  không ảnh hưởng thao tác đọc của  $T_j$ .
- Lịch biểu dùng kỹ thuật này là không rollback dây chuyền vì thao tác ghi được thực hiện sau khi giao tác commit.

# Nhận xét chung

## ❑ Lưu trữ:

- PP lock: lock table, không gian lưu trữ tỉ lệ với số lượng đvdl bị lock.
- PP nhãn thời gian: cần không gian lưu trữ cho nhãn thời gian đọc và ghi của từng đvdl, bất kể có đang được truy cập hay không.
- Validation: cần lưu lại nhãn thời gian và read set, write set cho từng giao tác đang được kích hoạt.

➤ Nói chung, không gian lưu trữ gần như tỉ lệ với số lượng đvdl được truy cập bởi các giao tác đang ở trạng thái kích hoạt.

## ❑ Về khả năng giao tác hoàn tất mà không bị trì hoãn:

- Hiệu quả của các phương pháp còn tùy thuộc vào sự ảnh hưởng lẫn nhau của các giao tác (truy cập cùng đvdl) là nhiều hay ít.
  - PP lock: trì hoãn GT nhưng tránh được tình trạng rollback, ngay cả khi sự ảnh hưởng lẫn nhau giữa các GT đồng thời là nhiều. PP nhãn thời gian và pp Kiểm tra hợp lệ không trì hoãn GT nhưng có thể gây ra tình trạng GT rollback → lãng phí tài nguyên.
  - Nếu sự ảnh hưởng lẫn nhau giữa các GT đồng thời là nhỏ thì cả PP nhãn thời gian lẫn Kiểm tra hợp lệ đều không gây ra rollback. Khi đó cả hai làm việc tốt hơn và tốn ít chi phí hơn PP lock.
  - Khi rollback xảy ra, pp Nhãn thời gian bắt vấn đề sớm hơn pp Kiểm tra hợp lệ.

**Hết chương 2.**