

TRƯỜNG ĐẠI HỌC CẦN THƠ
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN

QUẢN TRỊ DỮ LIỆU - CT467

**Chương 4: ĐIỀU KHIỂN
CẠNH TRANH**

Biên soạn:



Ths. Nguyễn Thị Kim Yến



Ntkyен@ctu.edu.vn

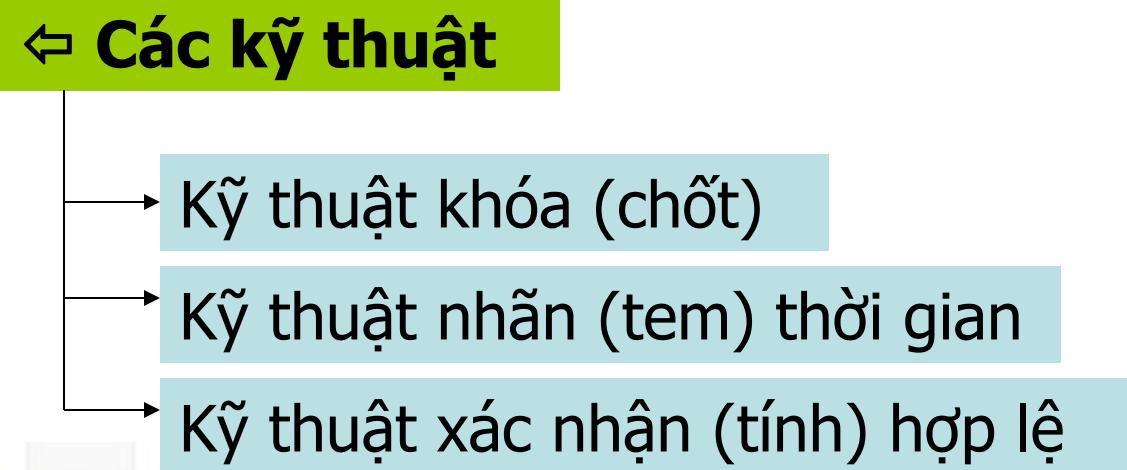
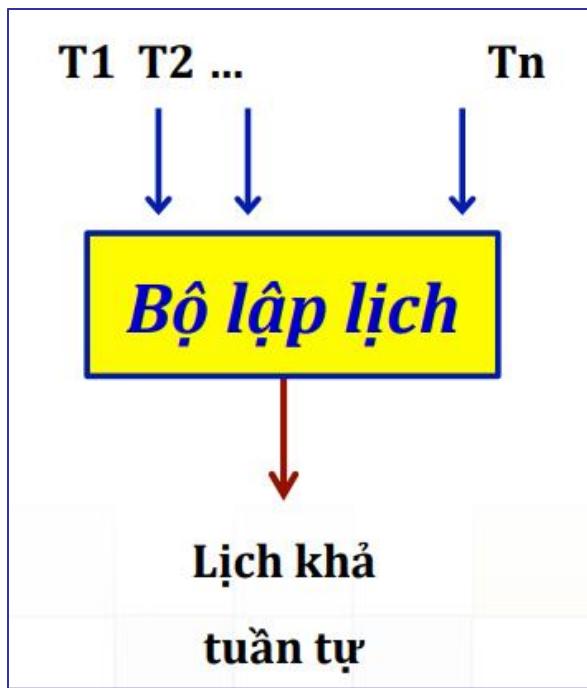
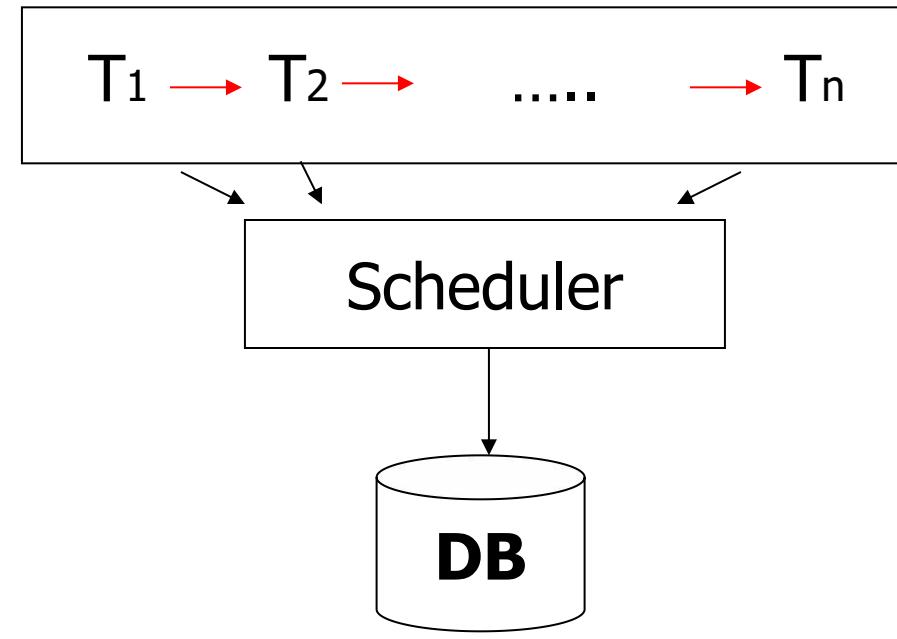


NỘI DUNG

- 1 Kỹ thuật khóa (chốt)
- 2 Kỹ thuật nhãn (tem) thời gian
- 3 Giao thức dựa trên tính hợp lệ
- 4 Quản lý deadlock

GIỚI THIỆU (tt)

- C3_XĐ LT có khả tuần tự ?
- Làm thế nào để sinh ra được các LT **khả tuần tự**?



1. KỸ THUẬT KHÓA (CHỐT)



Các loại:

1.1 Khóa đơn giản

1.3 Khóa 2 giai đoạn
(giao thức chốt 2 kỳ)

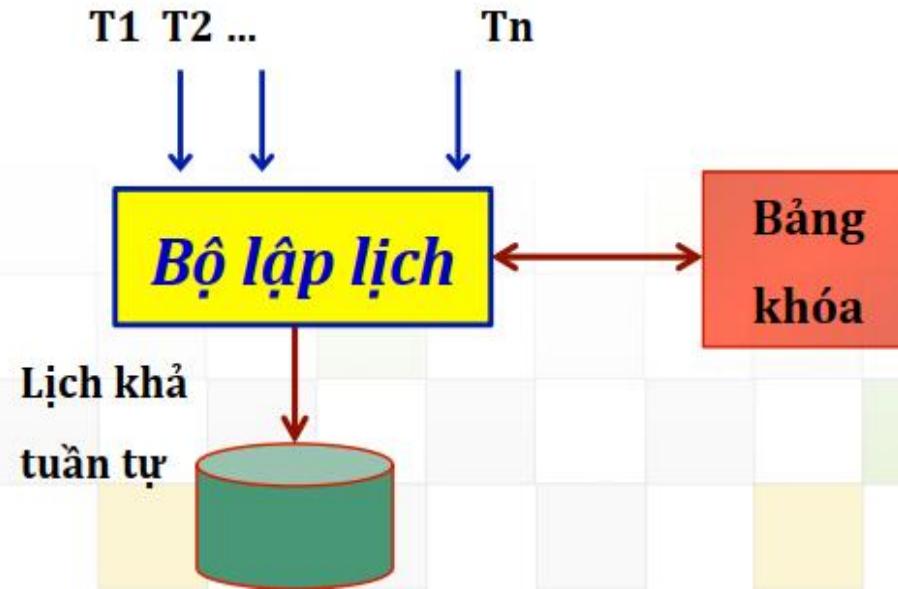
1.2 Khóa đọc ghi (chốt và cấp chốt)

1.4 Khóa đa hạt

1.5 Khóa đồ thị

Kỹ thuật khoá đơn giản

- Kỹ thuật khoá đơn giản còn gọi khoá nhị phân (Binary locks)
- Bộ lập lịch với cơ chế khoá đơn giản (locking scheduler)
 - Là bộ lập lịch với thêm **2** hành động:
 - * **Lock** : Phát khoá
 - * **Unlock** : Giải phóng khoá
 - Các khoá được ghi nhận trong bảng khoá (Lock Table)



Kỹ thuật khóa đơn giản

Quy định:

- Các giao tác trước khi muốn đọc/ghi lên 1 đơn vị dữ liệu phải phát ra 1 yêu cầu xin khóa (lock) đơn vị dữ liệu đó
 - * Ký hiệu: **Lock(A)** hay **l(A)**
- Yêu cầu này được bộ phận quản lý khóa xử lý (Lock Manager)
 - * Nếu yêu cầu được chấp thuận thì giao tác mới được phép đọc/ghi lên đơn vị dữ liệu
 - * Yêu cầu xin khóa được bộ cấp phát chấp thuận nếu đơn vị dữ liệu chưa bị khóa bởi một giao tác nào khác
- Sau khi thao tác xong thì giao tác phải phát ra lệnh giải phóng đơn vị dữ liệu (unlock)
 - * Ký hiệu: **Unlock(A)** hay **u(A)**

Bảng khóa ghi nhận giao
tác T1 đang giữ khóa
trên đơn vị dữ liệu A

BẢNG KHÓA

Element	Transaction
A	T1

Kỹ thuật khóa đơn giản (tt)

■ Quy tắc:

- 1. **Giao tác đúng đắn:** Việc giao tác Ti đọc hay ghi lên đơn vị dữ liệu A phải sau khi Ti phát khoá trên A và trước khi Ti giải phóng khoá trên A. Phát khoá và giải phóng khoá phải đi đôi với nhau (lock trước, unlock sau)

* Ti: ... l(A) ... r(A) / w(A) ... u(A) ...

- 2. **Lịch thao tác hợp lệ:** Khi Ti đang giữ khoá trên một đơn vị dữ liệu A thì không Ti nào khác được phát khoá trên A.

* S: ... li(A) ui(A) ...





Kỹ thuật khóa đơn giản (tt)

- Ví dụ:
 - Giao tác T1 và T2 có đúng đắn hay không ?
 - Lịch S có hợp lệ hay không ?

Trả lời:

- Giao tác T1 và T2 đúng đắn
- Lịch trình S có hợp lệ

S	T1	T2
	Lock(A) Read(A, t) $t := t + 100$ Write(A, t) Unlock(A)	Lock(A) Read(A, s) $s := s * 2$ Write(A, s) Unlock(A)
	Lock(B) Read(B, t) $t := t + 100$ Write(B, t) Unlock(B)	Lock(B) Read(B, s) $s := s * 2$ Write(B, s) Unlock(B)



Kỹ thuật khóa đọc ghi

(1.2 Chốt và cấp chốt)

- Dựa trên sự **loại trừ hỗ tương**: khi 1 GD đang truy xuất một hạng mục DL, không một GD nào khác được phép cập nhật hạng mục DL đó.
 - Phương pháp thực hiện:
 - Mỗi hạng mục DL sẽ có một **chốt**.
 - Một GD chỉ được **truy xuất** 1 hạng mục DL nếu nó **đang giữ chốt** trên hạng mục DL đó
- ⇒ Một GD muốn truy xuất một hạng mục DL phải **xin chốt** tương ứng trên hạng mục DL đó.



Kỹ thuật khóa đọc ghi

(1.2 Chốt và cấp chốt)

- **Chốt**

- Share (khóa chia sẻ): chỉ đọc.
- eXclusive (khóa loại trừ): có thể đọc/ghi.

- **Các thao tác trên chốt**

- Xin chốt S trên A: Lock_S(A), L_S(A), LS(A), RLock(A)
- Xin chốt X trên A: Lock_X(A), L_X(A), LX(A), WLock(A)
- Tháo chốt trên A: Unlock(A), U(A)

Kỹ thuật khóa đọc ghi (tt)

(1.2 Chốt và cấp chốt)

- Một GD cần **truy xuất** hạng mục DL => **Xin chốt** Được cấp Chờ
- Yêu cầu chốt Share:** Được cấp nếu không có GD nào giữ chốt **Exclusive**, cho đến khi tháo chốt.
- Yêu cầu chốt Exclusive:** Được cấp nếu không có GD nào giữ **bất kỳ chốt nào**, cho đến khi tháo chốt.
- Ma trận tương thích:** *Yêu cầu khoá*

		RLock(A)	WLock(A)
Trạng thái hiện hành	RLock(A)	✓	✗
	WLock(A)	✗	✗

✓ Tương thích

✗ Không tương thích



Kỹ thuật khóa đọc ghi (tt)

(1.2 Chốt và cấp chốt)

- Sau khi sử dụng xong hạng mục DL thì phải **tháo chốt** (Unlock)
- Xét ví dụ:** Cho 2 lịch trình sau (g/s giá trị A và B trước khi xảy ra các giao dịch lần lượt là 100\$ và 200\$)
 - GD T1 chuyển \$50 từ tài khoản B sang tài khoản A
 - GD T2 hiển thị tổng số tiền trong các tài khoản A và B

T1	R(B) $B = B - 50$ W(B) R(A) $A = A + 50$ W(A)
-----------	--------------------------------------------------------------

T2	R(A) R(B) Display(A+B)
-----------	------------------------------



Kỹ thuật khóa đọc ghi (tt)

- Tháo chốt sớm: tránh deadlock + không nhất quán

T1	T2	Bộ quản lý cạnh tranh
L_X(B); R(B) B = B - 50 W(B); U(B)		Grant - X (B, T1)
	L_S(A); R(A) U(A)	Grant - S (A, T2)
	L_S(B); R(B); U(B) Display(A+B)	Grant - S (B, T2)
L_X(A); R(A) A = A + 50 W(A); U(A)		Grant - X (A, T1)

Schedule 1

Kỹ thuật khóa đọc ghi (tt)

- Tháo chốt trễ: đảm bảo tính nhất quán + deadlock

T1	T2	Bộ quản lý cạnh tranh
$L_X(B); R(B)$ $B = B - 50$ $W(B)$	$L_S(A); R(A)$ $L_S(B); R(B)$ Display(A+B)	Grant - X (B, T1)
$L_X(A); R(A)$ $A = A + 50$ $W(A)$ $U(A); U(B)$	$U(A); U(B)$	Grant - S (A, T2) Wait - S (B, T2) Wait - X (A, T1)

Schedule 2



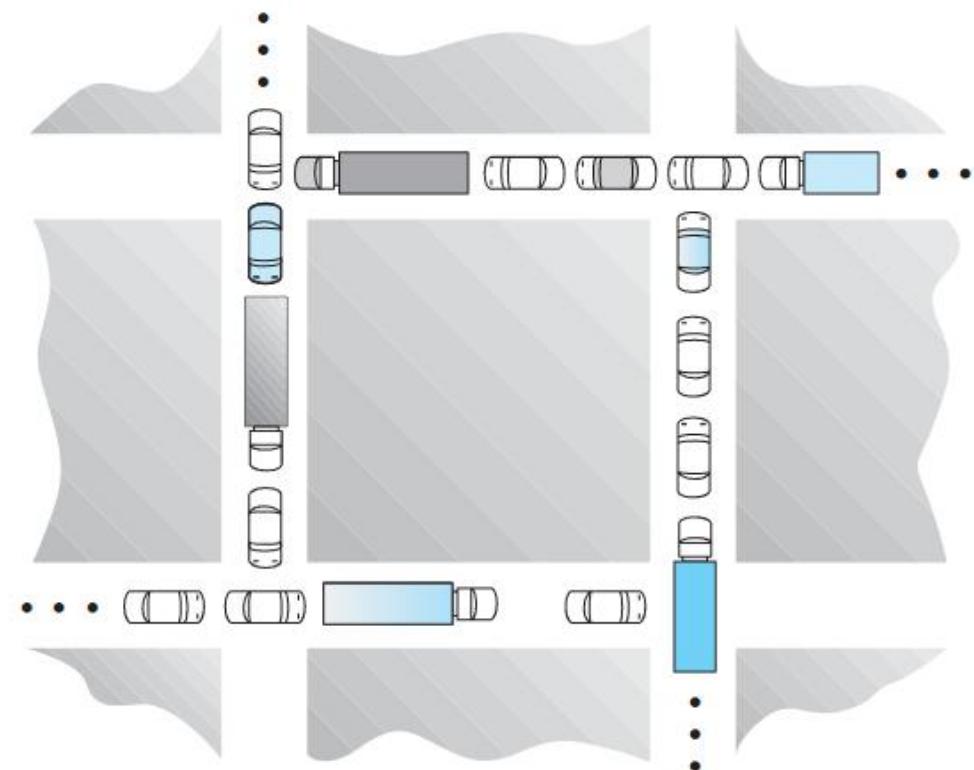
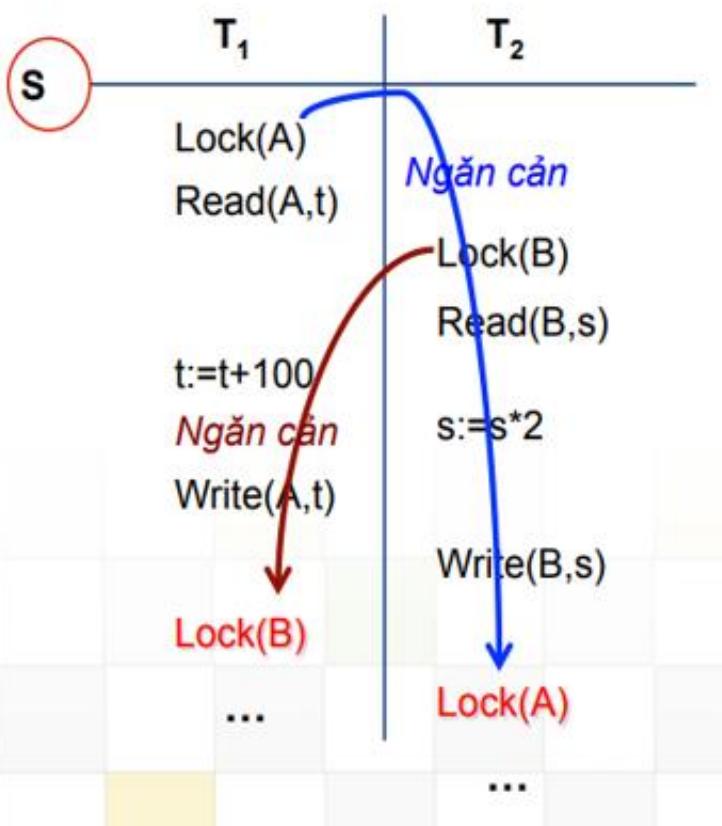
Kỹ thuật khóa đọc ghi (tt)

(1.2 Chốt và cấp chốt)

- Trên thực tế thì **deadlock** được ưa thích hơn trạng thái không nhất quán (!?)
- Để sinh ra được các LT **khả tuần tự** trong hệ thống, ta y/c các GD trong hệ thống phải tuân theo tập các **qui tắc xin và tháo chốt** ⇒ giao thức dựa trên chốt
- Giao thức chốt hạn chế số lịch trình có thể, là **tập con** của tập tất cả các lịch trình khả tuần tự
- Một LT được gọi là **thỏa giao thức chốt** nếu nó tuân thủ các quy tắc của giao thức đó

Kỹ thuật khóa đơn giản (tt)

- Vấn đề khoá chết (Dead lock): Hiện tượng chờ khi cần phát khóa có thể dẫn đến chờ lẫn nhau **vĩnh viễn**





Kỹ thuật khoá đơn giản (tt)

- ❖ Để tránh chết đói, ta có thể dùng giao thức sau:
 - GD T_i chỉ được cấp khóa trên hạng mục dữ liệu Q ở phương thức M, khi cả hai điều kiện sau được thỏa:
 1. Không có GD khác giữ khóa không tương thích với M trên Q
 2. Không có GD khác đang chờ khóa trên Q và đã đưa ra yêu cầu cấp khóa trước T_i
- ❖ Một GD sẽ được cấp khóa nếu như không có GD nào giữ khóa ở phương thức không tương thích



Kỹ thuật khóa đọc ghi (tt)

(1.2 Chốt và cấp chốt)

■ Qui tắc

- (1) Giao tác đúng đắn :

- * Đã có phát khóa thì sau đó phải có giải phóng khóa, giải phóng khóa chỉ có khi trước đó có phát khóa mà chưa giải phóng
- * Thao tác đọc chỉ được thực hiện sau khi phát khóa đọc hoặc ghi và trước khi giải phóng khóa ấy
- * Thao tác ghi chỉ được thực hiện sau khi phát khóa ghi và trước khi giải phóng khóa ghi ấy
- * Các thao tác đọc, ghi, phát khóa và giải phóng khóa đề cập trên đây là xét trong cùng một giao tác và trên cùng 1 đơn vị dữ liệu

Kỹ thuật khóa đọc ghi (tt)

(1.2 Chốt và cấp chốt)

■ Qui tắc

- (2) - *Lịch thao tác hợp lệ*

- * Khi T_i đang giữ khóa đọc trên 1 đơn vị Dữ liệu A thì **không** một T_j nào khác được phép ghi trên A
- * Khi T_i đang giữ khóa ghi trên 1 đơn vị Dữ liệu A thì **không** một T_j nào khác được phép đọc hay ghi trên A

Kỹ thuật khóa đọc ghi (tt)

(1.2 Chốt và cấp chốt)

▪ Qui tắc

- (3) - *Giao tác 2PL* (Giao thức chốt 2 kỳ)

* Ngoại trừ trường hợp nâng cấp khóa, các trường hợp còn lại đều giống với nghi thức khóa hai giai đoạn

* T : ... **rli(A)** ... **wli(A)** **ui(A)** ...

**Không có giải
phóng bất kỳ
khóa nào**

**Không có phát
ra bất kỳ khóa
nào**

* Trường hợp nâng cấp khóa được giải phóng khóa đọc trong pha phát khóa

* T : ... rli(A) **uli(A)** wli(A) ui(A) ...

**Chấp nhận giải phóng
khóa đọc khi nâng cấp
khóa**

▪ Định lý :

- S thoả (1), (2) và (3) → S **conflict-serializable** (Khả tuần tự xung đột)

Ví dụ

	T ₁	T ₂
S1		
	RLock(A)	
	Read(A)	
	U(A)	
		RLock(B)
		Read(B)
		U(B)
	WLock(A)	
	Read(A)	
	A:=A+B	
	Write(A)	
	U(A)	
	WLock(B)	
	Read(B)	
	B:=B+A	
	Write(B)	
	U(B)	

1. Giao tác nào đúng đắn ?
2. Lịch thao tác có hợp lệ hay không ?
3. Giao tác nào không thỏa nghi thức 2PL ?
4. S có khả tuần tự xung đột không?

Trả lời:

1. Giao tác T1 và T2 đúng đắn
2. Lịch thao tác có hợp lệ
3. Giao tác T1 và T2 KHÔNG thỏa nghi thức khóa 2 giao đoạn
4. S KHÔNG khả tuần tự XĐ



Bài tập về nhà (1)

S2	T ₁	T ₂	T ₃	T ₄
	RL(A)	RL(A)		
	WL(B) U(A)		WL(A)	
	U(B)		U(A)	
RL(B)			RL(B)	
RL(A)		U(A)		U(B)
WL(C) U(A)				WL(B) U(B)
U(B) U(C)				

1. Giao tác nào đúng đắn ?
 2. Lịch thao tác có hợp lệ hay không ?
 3. Giao tác nào không thỏa nghi thức 2PL ?
 4. S có khả tuần tự xung đột không?

Kỹ thuật khóa đọc ghi (tt)

- Để tăng tính cạnh tranh cho GD, nâng cấp cho giao thức chốt 2 kỳ dựa trên việc chuyển đổi chốt (lock conversion)
 - Nâng cấp** (upgrade): Khóa chia sẻ (L_S) → Khóa loại trừ (L_X) → Chỉ thực hiện trong kỳ phình to
 - Hạ cấp** (downgrade): Khóa loại trừ (L_X) → Khóa chia sẻ (L_S) → Chỉ thực hiện trong kỳ thu nhỏ

Kỹ thuật khóa đọc ghi (tt)

- Khi 1 GD T_i phát ra 1 chỉ thị **Read(A)**, hệ thống sẽ sinh ra 1 chỉ thị **Lock_S(A)** ngay trước chỉ thị Read(A)
- Khi 1 GD T_i phát ra 1 chỉ thị **Write(A)**, hệ thống sẽ **kiểm tra** xem T_i có giữ khóa chia sẻ trên A hay chưa:
 - Nếu có thì sẽ sinh ra 1 chỉ thị **Upgrade(A)** ngay trước Write(A)
 - Ngược lại, phát ra 1 chỉ thị **Lock_X(A)** ngay trước Write(A)

- Ma trận tương thích

		Yêu cầu khóa		
		RLock	WLock	ULock
Trạng thái hiện hành	RLock	✓	✗	✓
	WLock	✗	✗	✗
	ULock	✗	✗	✗

Kỹ thuật khóa đọc ghi (tt)

(Chốt và cấp chốt)

Ví dụ: Xét 2 giao dịch như sau:

T ₁	T ₂
Read(A ₁)	Read(A ₁)
Read(A ₂)	Read(A ₂)
.....	Display (A ₁ +A ₂)
Read(A _n)	
Write(A ₁)	



	T ₁	T ₂
1	L_S(A ₁)	
2		L_S(A ₁)
3	L_S(A ₂)	
4		L_S(A ₂)
5	L_S(A ₃)	
6	L_S(A ₄)	
7		Unlock(A ₁)
8		Unlock(A ₂)
9	L_S(A _n)	
10	Upgrade(A ₁)	
11	Wite(A ₁)	

Nghi thức khoá 2 giai đoạn

(1.3 Giao thức chốt 2 kỳ)

- Nghi thức khoá 2 giai đoạn chia việc xin khoá và giải phóng khoá của giao tác thành 2 giai đoạn (phase) phân biệt:

(xin khoá)

- **Growing phase** (pha phát khoá):

- * Giao tác chỉ được phép xin khoá chứ không được phép giải phóng khoá trong pha này (số lượng khoá được giữ chỉ được phép ngày càng tăng) (**kỳ phình to**)

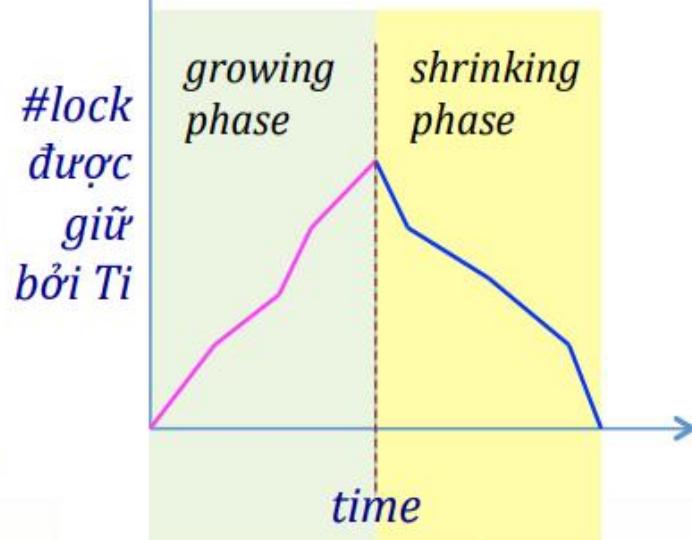
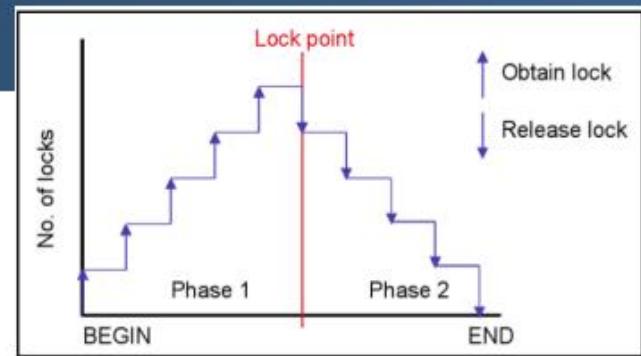
- * Giai đoạn này kết thúc ở lệnh xin khoá cuối cùng.

(trả khoá)

- **Shrinking phase** (pha giải phóng khoá): (**kỳ thu nhỏ**)

- * Giao tác chỉ được phép giải phóng khoá chứ không được phép xin khoá trong pha này

- * Giao tác này bắt đầu từ khi lệnh giải phóng khoá đầu tiên.



Nghi thức khoá 2 giai đoạn (tt)

(1.3 Giao thức chốt 2 kỳ)

- **Mục tiêu:** Vậy làm sao để kỹ thuật khóa cho ta lịch khả tuần tự
- **Giải pháp :** Tuân theo quy tắc sau:
 - (1) và (2): Giống như cũ (giao tác đúng đắn - thao tác hợp lệ)
 - (3) Giao tác phải thỏa **nghi thức khoá 2 giai đoạn (2PL: two phases locking)** : Trong 1 giao tác, tất cả các thao tác phát khóa đều xảy ra trước tất cả các thao tác giải phóng khóa.

T : **I(A) I(B)** **u(A) u(B)**

**Không có giải
phóng bất kỳ
khóa nào**

**Không có phát
ra bất kỳ
khóa nào**

■ Định lý:

- Nếu lịch S thoả nghi thức 2PL thì S conflict-serializable

Nghi thức khóa 2 giai đoạn (tt)

T1	T2	T3	T4
Lock(A)	Lock(B)	Lock(B)	Lock(A)
Read(A)	Read(B)	Read(B)	Read(A)
Lock (B)	Lock(A)	B=B-50	Unlock(A)
Read(B)	Read(A)	Write(B)	Lock(B)
B:= B + A	Unlock(B)	Unlock(B)	Read(B)
Write(B)	A:= A+B	Lock(A)	Unlock(B)
Unlock(A)	Write(A)	Read(A)	Print (A+B)
Unlock(B)	Unlock(A)	A:=A+50	
		Write(A)	
		Unlock(A)	

*Thỏa nghi thức khóa 2 giai
đoạn*

*Không thỏa nghi thức khóa
2 giai đoạn*

Nghi thức khoá 2 giai đoạn (tt)

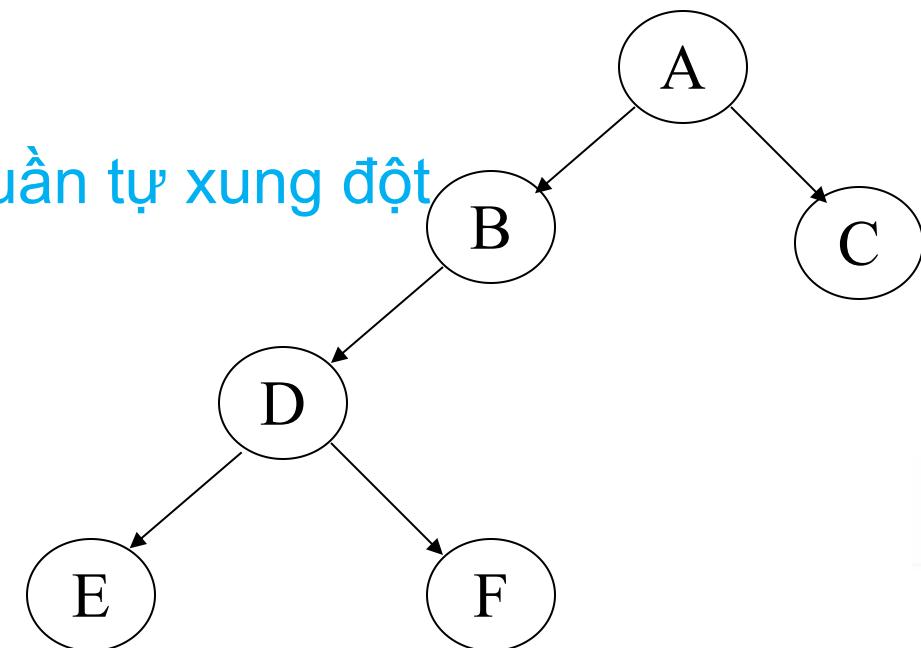
(1.3 Giao thức chốt 2 kỳ)

- ❖ Các lịch trình tuân theo giao thức chốt 2 kỳ: lịch trình **khả tuân tự xung đột** → tuy nhiên, không tránh được **deadlock** và **bị cuộn lại hàng loạt**.
- ❖ **Giải pháp:** **tránh cuộn lại hàng loạt** là thêm ràng buộc
 - Giao thức chốt 2 kỳ **nghiêm ngặt**: tất cả các chốt X phải được trả ở cuối giao dịch
 - Giao thức chốt 2 kỳ **nghiêm khắc**: tất cả các chốt phải được trả ở cuối giao dịch.

(1.4 Giao thức dựa trên đồ thị)

- ❖ Có thể thực hiện nếu biết trước thứ tự mà các hạng mục DL được truy xuất. Xem như đồ thị có hướng phi chu trình
⇒ Đồ thị CSDL
- ❖ Tính chất:
 - LT tuân theo GT cây **khả tuần tự xung đột**
 - **Tránh deadlock**
 - Chỉ sử dụng chốt **Lock_X**.

A → B có nghĩa là nếu GD nào truy xuất cả A và B phải truy xuất A trước B



Kỹ thuật khóa - Đồ thị (tt)

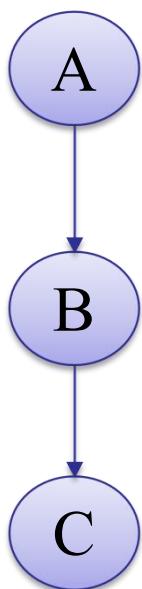
(1.4 Giao thức dựa trên đồ thị)

◆ Quy tắc:

- (1) Mỗi hạng mục có thể chốt nhiều nhất 1 lần
 - (2) Chốt đầu tiên bởi T có thể trên bất kỳ hạng mục DL nào
 - (3) Sau đó, Q có thể được chốt bởi T nếu T đã chốt cha Q
 - (4) Các hạng mục DL có thể tháo chốt bất kỳ lúc nào
- ◆ Giao thức này có thuận lợi hơn GT chốt 2 kỳ là ta **có thể tháo chốt sớm** nhưng có hạn chế là phải chốt các hạng mục DL không truy xuất (3) → **lãng phí khóa, tăng thời gian chờ và giảm tính cạnh tranh**
- ◆ GT cây cũng không đảm bảo tính chất khả phục hồi và tránh cuộn lại hàng loạt => **tất cả các khóa phải giữ đến cuối GD**

Kỹ thuật khóa - Đồ thị (tt)

◆ Ví dụ: GD thỏa GT cây nhưng không thỏa GT chốt 2 kỳ và ngược lại



T1	T2
L_X(A)	
L_X(B)	
U(A)	L_X(A)
L_X(C)	
U(B)	L_X(B)
	U(A)
	U(B)
U(C)	
<i>Tree, !2PL</i>	

T1	T2
L_X(A)	
	L_S(B)
L_X(C)	
	U_S(B)
U(A)	
U(C)	
<i>2PL, !Tree</i>	

Bài tập về nhà (2)

(Giao thức dựa trên đồ thị)

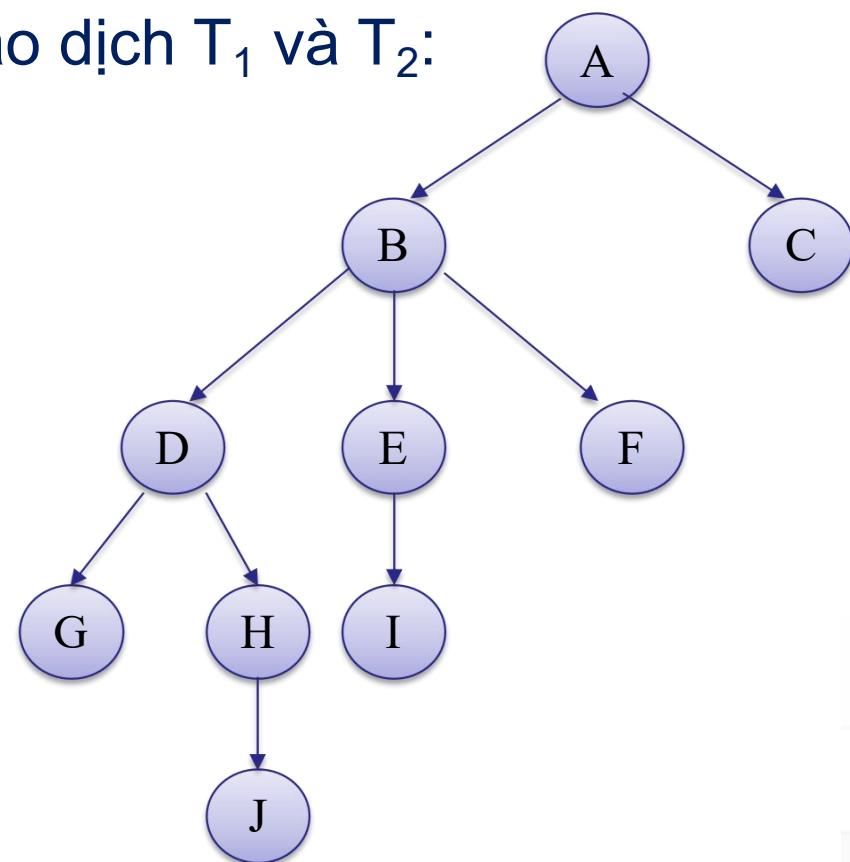
◆ Bài tập 2:

Cho đồ thị CSDL như sau và hai giao dịch T_1 và T_2 :

- T_1 : Truy xuất B, E, I.
- T_2 : Truy xuất D, H.

1) Viết các chỉ thị xin/tháo khóa cho các giao dịch trên theo đúng giao thức cây.

2) Tạo một lịch trình cho 2 giao dịch trên thỏa giao thức cây và giao thức chốt hai kỳ.





Kỹ thuật khóa đa hạt (Multiple Granularity)

(1.5 Đa hạt)

- Xét ví dụ hệ thống ngân hàng
 - Quan hệ TaiKhoan (maTK, soDu)
 - Giao tác gửi tiền và rút tiền

❖ Khóa Relation:

- Các giao tác thay đổi giá trị của số dư của 1 tài khoản X
sẽ yêu cầu khóa độc quyền
- Vì khóa ở mức độ quan hệ nên toàn bộ bảng bị khóa
- Các giao tác khác muốn truy cập tài khoản Y (Y khác X)
cũng phải chờ → vô lý, tốc độ xử lý đồng thời chậm



Kỹ thuật khóa đa hạt

(1.5 Đa hạt)

❖ Nên sử dụng chốt nhỏ hay lớn?

- Chốt nhỏ (mẫu tin), phù hợp với các GD truy xuất ít hạng mục DL => cho phí chốt cao nhưng tăng tính cạnh tranh
- Chốt lớn (bảng, cả CSDL), phù hợp với các GD truy xuất nhiều hạng mục DL => cần ít khóa nhưng ít cạnh tranh

❖ Giải pháp:

- Đa hạt (multiple granulatiry)
- Hạt ở đây có nghĩa là đơn vị cấp chốt: hệ thống hỗ trợ nhiều đơn vị cấp chốt hay nhiều mức hạt khác nhau.



Kỹ thuật khóa đa hạt (tt)

- Hạt ở đây có nghĩa là **đơn vị cấp chốt**
- Phải quản lý khóa ở nhiều mức độ
 - Tính chất hạt (granularity) : Tính hạt càng tăng khi đơn vị dữ liệu bị khóa càng lớn
 - Tính đồng thời (concurrency) : Tính đồng thời càng tăng khi đơn vị dữ liệu bị khóa càng nhỏ
 - Tính hạt tăng thì tính đồng thời giảm và ngược lại → phải thỏa hiệp

Tính hạt tăng

Quan hệ (Relation)

Khối (Block)

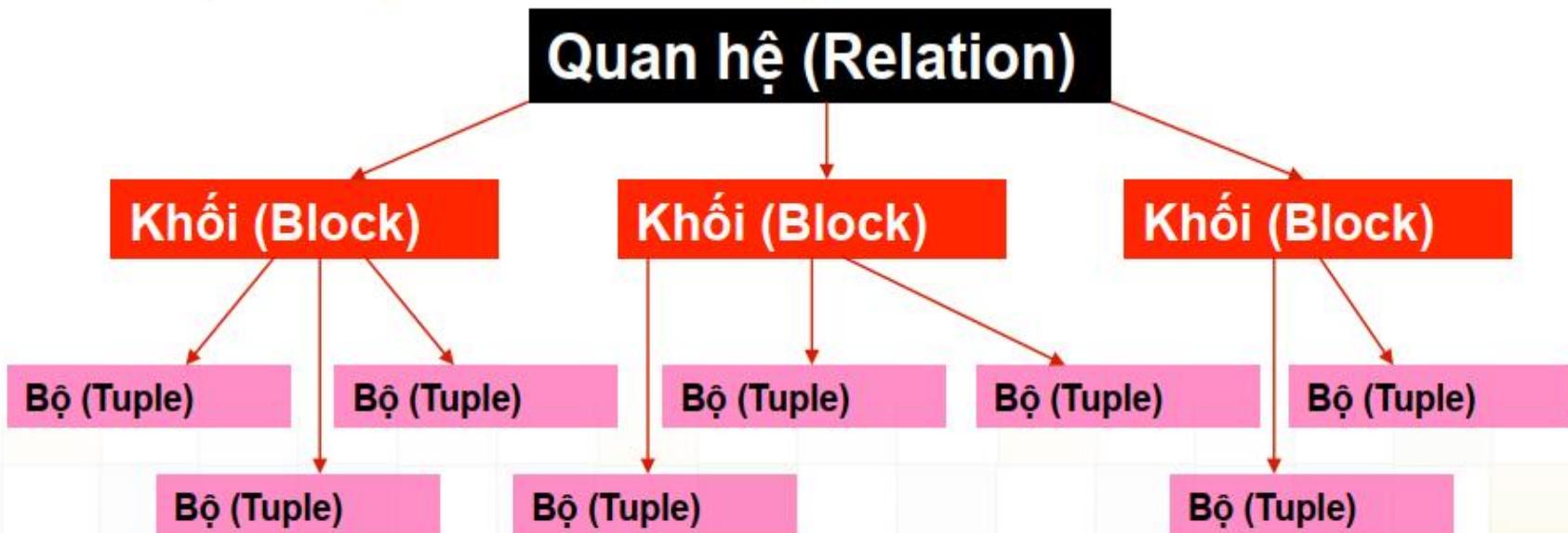
Bộ (Tuple)

Tính đồng thời tăng

Kỹ thuật khóa đa hạt (tt)

■ Phân cấp Dữ liệu

- Relations là đơn vị dữ liệu khóa lớn nhất (mức cao nhất: toàn bộ CSDL)
- Một relation gồm 1 hoặc nhiều blocks (pages) (vùng, không gian bảng)
- Một block gồm 1 hoặc nhiều tuples (mức thấp nhất: các mẫu tin)





Kỹ thuật khóa đa hạt (tt)

❖ Gồm các khóa:

- **Khóa thông thường (khóa tường minh):**
 - Shared lock: S
 - Exclusive lock: X
- **Khóa cảnh báo (khóa tăng cường):**
 - Warning (intensive) shared lock: IS cho khóa chia sẻ S. Khi 1 nút được khóa ở phương thức chia sẻ S, thì khóa IS nó cũng sẽ hiện diện trên tất cả các nút tổ tiên.
 - Warning (intensive) exclusive lock: IX cho khóa loại trừ X
 - Shared intensive exclusive: SIX kết hợp của khóa S và IX

Kỹ thuật khóa đa hạt (tt)

- **Ma trận tương thích trên cùng một node**
 - Cho biết các khóa có thể cùng tồn tại trên 1 nút dữ liệu

		Requester				
		IS	IX	S	SIX	X
Holder	IS	T	T	T	T	F
	IX	T	T	F	F	F
	S	T	F	T	F	F
	SIX	T	F	F	F	F
	X	F	F	F	F	

Ma trận cạnh tranh khóa



Kỹ thuật khóa đa hạt (tt)

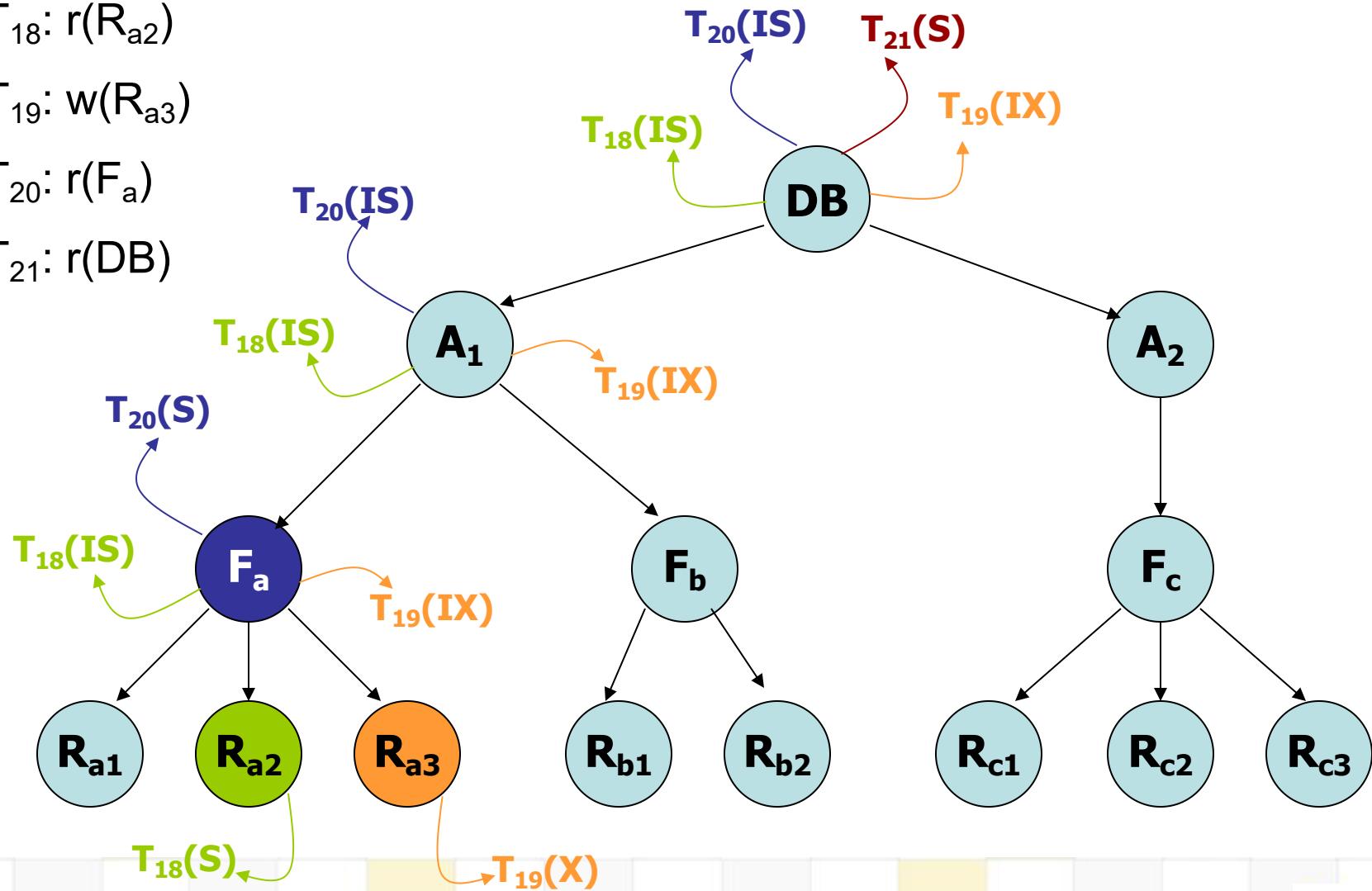
◆ Qui tắc cấp chốt:

1. Kiểm chứng **hàm tương thích**
2. **Gốc của cây** phải được chốt đầu tiên
3. Q có thể chốt bởi T ở **{S, IS}** nếu cha Q đang bị chốt bởi T ở **{IS, IX}**
4. Q có thể chốt bởi T ở **{X, SIX, IX}** nếu cha Q đang bị chốt bởi T ở **{IX, SIX}**
5. T tuân theo **giao thức chốt 2 kỳ**
6. T có thể **tháo chốt** Q nếu không có con của Q đang bị chốt bởi T

⇒ TẬU CHỐT: Top-Down, THÁO CHỐT: Bottom-Up

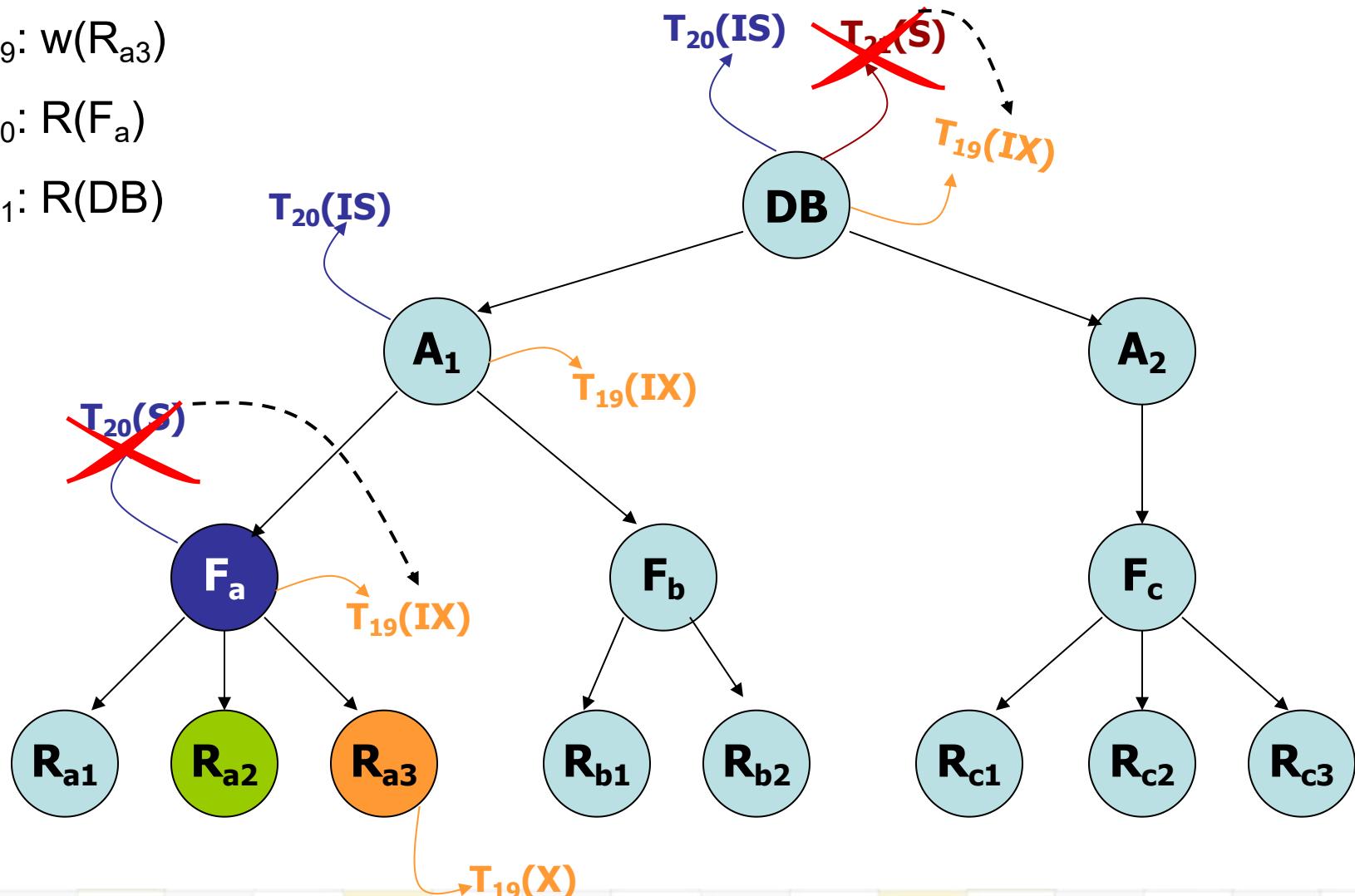
Kỹ thuật khóa đa hạt (tt)

- $T_{18}: r(R_{a2})$
- $T_{19}: w(R_{a3})$
- $T_{20}: r(F_a)$
- $T_{21}: r(DB)$



Kỹ thuật khóa đa hạt (tt)

- T_{19} : $w(R_{a3})$
- T_{20} : $R(F_a)$
- T_{21} : $R(DB)$





2. KỸ THUẬT NHÃN (TEM) THỜI GIAN

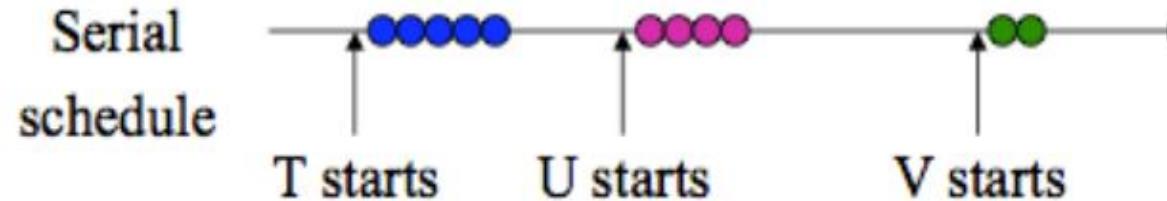
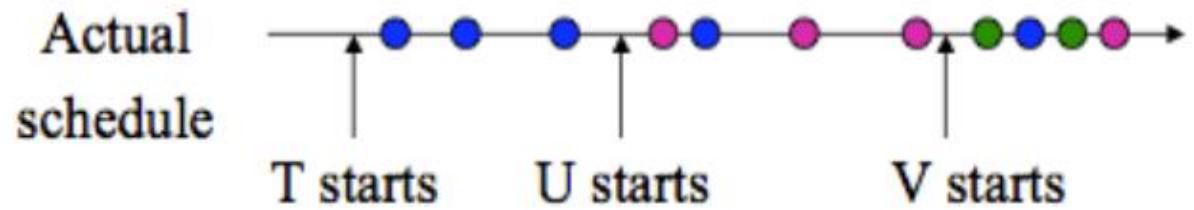


Giới thiệu

- **Ý tưởng:**
 - Đảm bảo các **chỉ thị xung đột** sẽ được thực hiện theo đúng thứ tự thực thi của các GD.
 - Giao thức này cũng sinh ra được các lịch trình **khả tuần tự xung đột** và **tránh được deadlock**
- Chọn một thứ tự thực hiện nào đó cho các giao tác bằng cách gán nhãn thời gian (timestamping)
 - ✓ Mục đích của gán nhãn thời gian: phân biệt được GD nào xảy ra trước, GD nào xảy ra sau trong hệ thống.
 - ✓ Mỗi GD T_i sẽ được gán một nhãn TG khi nó bắt đầu, ký hiệu: **TS(T_i)**
 - ✓ Thứ tự của các nhãn:
 - GD T_i bắt đầu **trước** GD T_j thì $TS(T_i) < TS(T_j)$
 - Hoặc, GD T_i bắt đầu **trễ hơn** GD T_j thì $TS(T_i) > TS(T_j)$

Giới thiệu

- Chiến lược cơ bản:
 - Nếu $TS(T_i) < TS(T_j)$ thì lịch thao tác được phát sinh phải tương đương với lịch biểu tuần tự $\{T_i, T_j\}$





Giới thiệu

❖ Có 2 phương pháp để gán nhãn TG cho các GD:

- **Đồng hồ máy tính:** Khi GD T_i bắt đầu sẽ lấy giá trị của **đồng hồ hệ thống** để gán cho $TS(T_i)$.
- **Bộ lặp lịch tự đếm:** Khi GD T_i bắt đầu sẽ được khởi tạo 1 giá trị của **biến đếm bất kỳ** để gán cho $TS(T_i)$, đồng thời **tăng giá trị** của biến đếm đó lên.

Nhãn thời gian toàn phần

- Mỗi giao tác T khi phát sinh sẽ được gán 1 nhãn TS(T) ghi nhận lại thời điểm phát sinh của T
- Mỗi đơn vị dữ liệu X cũng có 1 nhãn thời TS(X), nhãn này ghi lại TS(T) của giao tác T đã thao tác read/write thành công sau cùng lên X
- Khi đến lượt giao tác T thao tác trên dữ liệu X, so sánh TS(T) và TS(X)
 - **Nếu T muốn đọc X:**
 - * Nếu $TS(T) \geq TS(X)$ thì cho T đọc X và gán $TS(X) = TS(T)$
 - * Ngược lại T bị hủy (abort)
 - **Nếu T muốn ghi X:**
 - * Nếu $TS(T) \geq TS(X)$ thì cho T ghi X và gán $TS(X) = TS(T)$
 - * Ngược lại T bị hủy (abort)

Nhận thời gian toàn phần

Read(T, X)

```
If TS(X) <= TS(T)
    Read(X); //cho phép đọc X
    TS(X):= TS(T);
Else
    Abort {T}; //hủy bỏ T
```

Write(T, X)

```
If TS(X) <= TS(T)
    Write(X); //cho phép ghi
    TS(X):= TS(T);
Else
    Abort {T}; //hủy bỏ T
```

Ví dụ

T_1 $TS(T_1)=100$	T_2 $TS(T_2)=200$	A $TS(A)=0$	B $TS(B)=0$
Read(A)		$TS(A)=100$	
	Read(B)		$TS(B)=200$
$A=A^*2$			
Write(A)		$TS(A)=100$	
	$B=B+20$		
	Write(B)		$TS(B)=200$
Read(B)			

$TS(A) \leq TS(T_1)$: T_1 đọc được A

$TS(B) \leq TS(T_2)$: T_2 đọc được B

$TS(A) \leq TS(T_1)$: T_1 ghi lên A được

$TS(B) \leq TS(T_2)$: T_2 ghi lên B được

$TS(B) > TS(T_1)$: T_1 không đọc được B

T1 bị hủy, sau đó khởi động lại T1 với một nhãn thời gian mới

Nhãn thời gian toàn phần

T1 TS(T1)=100	T2 TS(T2)=120	A TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=120
	Write(A)	TS(A)=120
Write(A)		



T1 bị huỷ và bắt đầu thực hiện với timestamp mới

Mặc dù trong quản lý giao tác 2 hành động đọc/ghi có tác dụng rất khác nhau

Không phân biệt tính chất của thao tác là đọc hay ghi → T1 vẫn bị hủy và làm lại từ đầu với 1 timestamp mới

T1 TS(T1)=100	T2 TS(T2)=120	A TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=120
	Read(A)	TS(A)=120
Read(A)		



T1 bị huỷ và bắt đầu thực hiện với timestamp mới



Sử dụng nhãn thời gian riêng phần

Nhãn thời gian riêng phần

- Nhãn của đơn vị dữ liệu X được tách ra thành 2 loại:
 - **RT(X)** – read time of X hay **R-TS(X)**
 - * Ghi nhận TS(T) gần nhất (giao tác có nhãn thời gian lớn nhất) đọc X thành công
 - **WT(X)** – write time of X hay **W-TS(X)**
 - * Ghi nhận TS(T) gần nhất (giao tác có nhãn thời gian lớn nhất) ghi X thành công
- Ngoài ra bộ lập lịch cũng quản lý trạng thái commit hay chưa của đơn vị dữ liệu X
 - $C(X)=1$ nếu dữ liệu X đã được commit. Ngược lại $C(X)=0$
 - Bộ lập lịch dựa vào hoạt động của các giao tác để gán nhãn $C(X)$ lên các đơn vị dữ liệu.
 - Kỹ thuật này được sử dụng để tránh việc lỗi đọc phải dữ liệu rác.

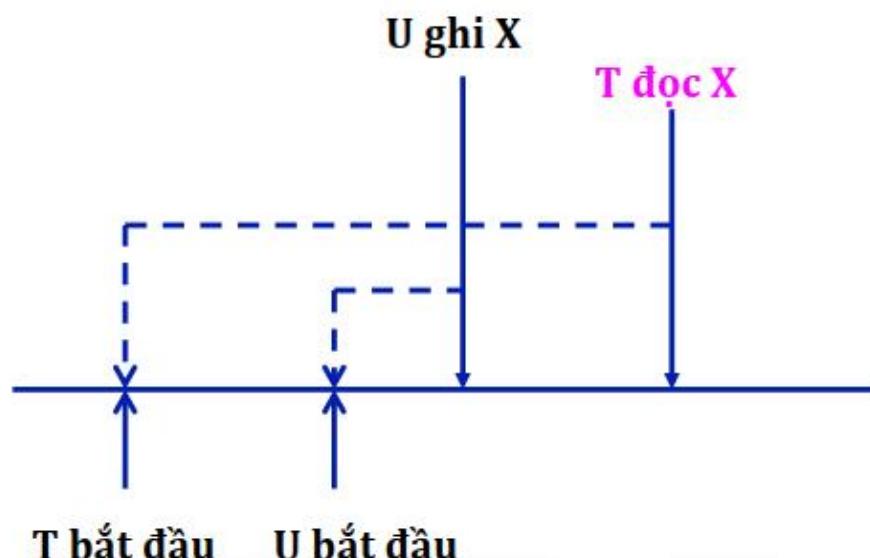


Nhãn thời gian riêng phần

- Công việc của bộ lập lịch:
 - Gán gắn các nhãn thời gian RT(X) và WT(X) và C(X)
 - Kiểm tra thao tác đọc/ghi xuất hiện khi nào để quyết định:
 - * Chấp nhận yêu cầu
 - * Trì hoãn giao tác
 - * Huỷ bỏ giao tác
 - * Bỏ qua hoạt động đó
 - Xử lý các tình huống:
 - * Đọc quá trễ
 - * Ghi quá trễ
 - * Đọc dữ liệu rác
 - * Quy tắc ghi Thomas

Nhận thời gian riêng phần

■ Đọc quá trễ

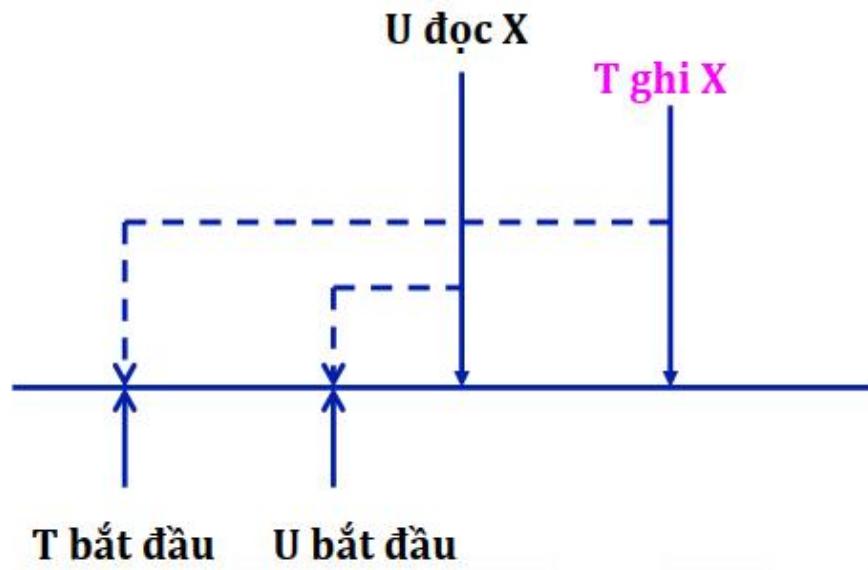


- T vào trước ($TS(T) < TS(U)$) muốn đọc X nhưng khi giá trị X hiện tại đã bị ghi bởi một giao tác khác vào sau T ($TS(T) < WT(X)$)
- T không nên đọc X do U ghi bởi vì T vào trước

→ Giải pháp: Hủy T

Nhận thời gian riêng phần

■ Ghi quá trễ

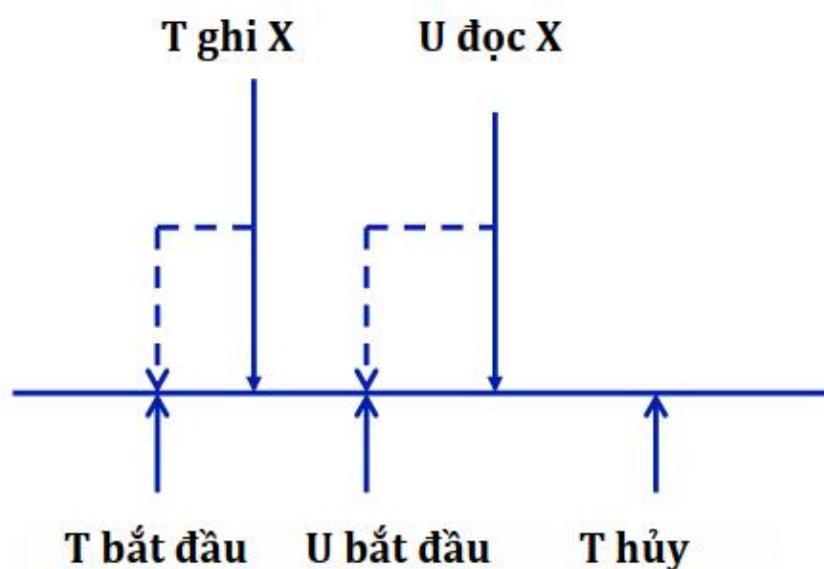


- T vào trước ($TS(T) < TS(U)$) và muốn ghi lên X, tuy nhiên X đã bị đọc bởi một giao tác vào sau T ($WT(X) < TS(T) < RT(X)$)
- T không nên ghi X do giao tác U đã đọc X. (U phải đọc giá trị do T ghi)

→ Giải pháp: Hủy T

Nhận thời gian riêng phần

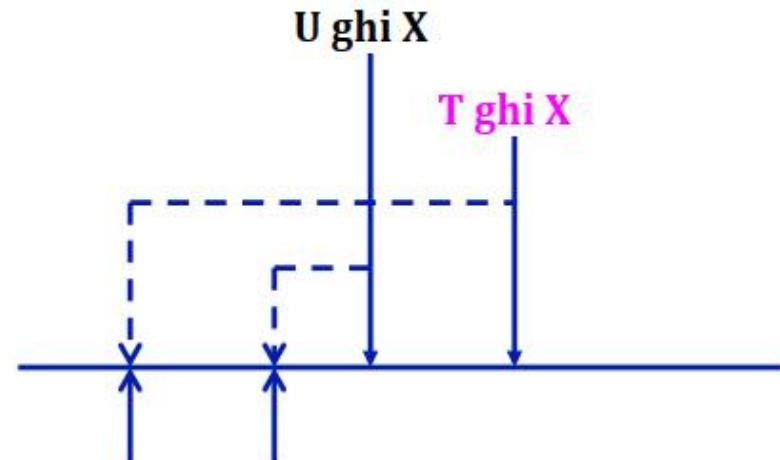
■ Đọc dữ liệu rác



- T vào trước U và thực hiện việc ghi X trước. U vào sau và thực hiện việc đọc X.
- Nhưng T hủy → giá trị X mà U đọc là giá trị rác
 - Giải pháp: U đọc X sau khi T commit hoặc sau khi T abort.

Nhận thời gian riêng phần

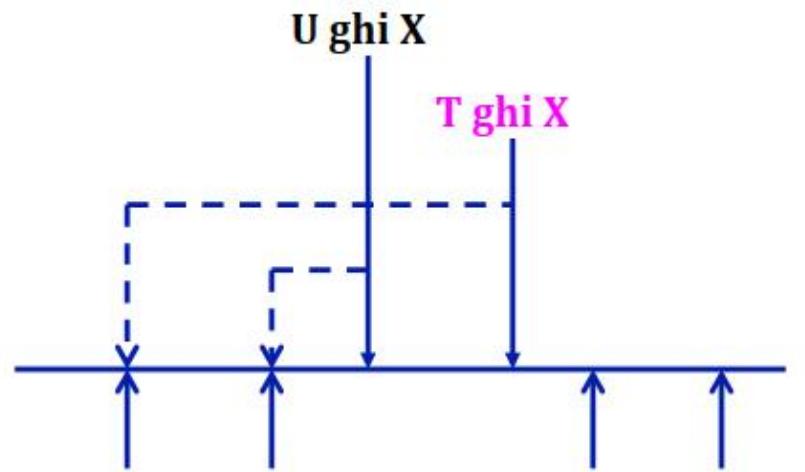
■ Quy tắc ghi Thomas



- T vào trước U vào sau nhưng khi T muốn ghi lên X thì U đã ghi lên X trước ($TS(T) < WT(X)$)
- T nếu có ghi xong X cũng không làm được gì vì:
 - T không thể đọc X vì nếu đọc X thì sẽ dẫn đến đọc trễ
 - Các giao tác khác sau T và U thì muốn đọc giá trị X được ghi bởi U
- Giải pháp: Bỏ qua thao thác ghi X của T [Quy tắc ghi Thomas]

Nhân thời gian riêng phần

■ Vấn đề với quy tắc ghi Thomas



T bắt đầu U bắt đầu T kết thúc U huỷ

- Trường hợp U ghi và sau đó bị huỷ → giá trị được ghi bởi U đã bị mất
- Do T đã kết thúc → cần khôi phục lại giá trị X từ T mà lệnh ghi X đã bị bỏ qua

→ Giải pháp: Khi T ghi X, nếu giao tác U đã commit thì bỏ qua T, hoặc đợi đến thời điểm U commit hoặc abort.



Nhãn thời gian riêng phần

- Tóm lại khi có yêu cầu đọc và ghi từ giao tác T. Bộ lập lịch sẽ:
 - Đáp ứng yêu cầu
 - Hủy T và khởi tạo lại T với 1 timestamp mới
 - * T rollback
 - Trì hoãn T, sau đó mới quyết định phải hủy T hoặc đáp ứng yêu cầu

Nhân thời gian riêng phần

■ Quy tắc :

- Nếu T cần đọc X

- * Nếu $WT(X) \leq TS(T)$ thì chờ cho X trở thành Dữ liệu đã Commit rồi cho T đọc X và gán $RT(X) = MAX(RT(X), TS(T))$
 - * Ngược lại hủy T và khởi động lại T với $TS(T)$ mới (đọc quá trễ)

- Nếu T cần ghi X

- * Nếu $RT(X) \leq TS(T)$

- Nếu $WT(X) \leq TS(T)$ thì cho T ghi X và gán $WT(X) = TS(T)$
 - Ngược lại bỏ qua thao tác ghi này của T (không hủy T)

- * Ngược lại hủy T và khởi động lại T với $TS(T)$ mới (ghi quá trễ)

Nhân thời gian riêng phần (tt)

Quy tắc:

Read(T, X)

```
If WT(X) <= TS(T)
    Read(X); //cho phép đọc X
    RT(X):= max(RT(X), TS(T));
Else
    Rollback{T}; //hủy T và khởi tạo lại với TS mới
```

Write(T, X)

```
If RT(X) <= TS(T)
    If WT(X) <= TS(T)
        Write(X); //cho phép ghi X
        WT(X):= TS(T);
    Else
        Nếu X đã COMMIT → bỏ qua, ngược lại chờ cho
        đến khi giao tác thực hiện ghi trên X commit
        hoặc abort
    Else
        Rollback{T}; //hủy T và khởi tạo lại với TS mới
```

Ví dụ

	T_1 $TS(T_1)=100$	T_2 $TS(T_2)=200$	A $RT(A)=0$ $WT(A)=0$	B $RT(B)=0$ $WT(B)=0$	C $RT(C)=0$ $WT(C)=0$	
1	Read(A)		$RT(A)=100$ $WT(A)=0$			$WT(A) < TS(T_1)$ T_1 đọc được A
2		Read(B)		$RT(B)=200$ $WT(B)=0$		$WT(B) < TS(T_2)$ T_2 đọc được B
3	Write(A)		$RT(A)=100$ $WT(A)=100$			$RT(A) < TS(T_1)$ $WT(A) = TS(T_1)$
4		Write(B)		$RT(B)=200$ $WT(B)=200$		$RT(B) < TS(T_2)$ $WT(B) = TS(T_2)$
5		Read(C)			$RT(C)=200$ $WT(C)=0$	$WT(B) < TS(T_2)$ T_2 đọc được C
6	Read(C)				$RT(C)=200$ $WT(C)=0$	$WT(C) < TS(T_1)$ T_1 đọc được C
7	Write(C)					$RT(C) > TS(T_1)$ T_1 không ghi lên C được
						Abort

Ví dụ (tt)

T_1 TS=200	T_2 TS=150	T_3 TS=175	A RT=0 WT=0	B RT=0 WT=0	C RT=0 WT=0
Read(B)				RT=200 WT=0	
	Read(A)		RT=150 WT=0		
		Read(C)			RT=175 WT=0
Write(B)				RT=200 WT=200	
Write(A)			RT=150 WT=200		
	Write(C)				
		Write(A)			

↓ ↓

Rollback **Giá trị của A đã sao lưu bởi $T_1 \rightarrow T_3$ không bị rollback và không cần ghi A**

Ví dụ (tt)

T_1 TS=150	T_2 TS=200	T_3 TS=175	T_4 TS=255	A RT=0 WT=0
Read(A)				RT=150 WT=0
Write(A)				RT=150 WT=150
	Read(A)			RT=200 WT=0
	Write(A)			RT=200 WT=200
		Read(A)		
			Read(A)	RT=255 WT=200
				↓
				Rollback

T3 bị hủy vì nó định đọc giá trị A ghi bởi T2 (mà T2 lại có nhãn thời gian lớn hơn nó). Giả sử T3 đọc giá trị A ghi bởi T1 thì T3 sẽ không bị hủy

Ý tưởng lưu giữ nhiều phiên bản của A



Nhận thời gian nhiều phiên bản

- Ý tưởng
 - Cho phép thao tác `read(A)` của T3 thực hiện
- Bên cạnh việc lưu trữ giá trị hiện hành của A, ta giữ lại các giá trị được sao lưu trước kia của A (phiên bản của A)
- Giao tác T sẽ đọc được giá trị của A ở 1 phiên bản thích hợp nào đó



Nhận thời gian nhiều phiên bản (tt)

- Mỗi phiên bản của 1 đơn vị dữ liệu X có
 - RT(X)
 - * Ghi nhận lại giao tác sau cùng đọc X thành công
 - WT(X)
 - * Ghi nhận lại giao tác sau cùng ghi X thành công
- Khi giao tác T phát ra yêu cầu thao tác lên X
 - Tìm 1 phiên bản thích hợp của X
 - Đảm bảo tính khả thi tự
- Một phiên bản mới của X sẽ được tạo khi hành động ghi X thành công

Nhân thời gian nhiều phiên bản (tt)

Quy tắc:

Read(T, X)

i=“số thứ tự phiên bản sau cùng nhất của X”

While $WT(X_i) > TS(T)$

i:=i-1;//lùi lại

Read(X_i);

$RT(X_i) := \max(RT(X_i), TS(T))$;

Write(T, X)

i=“số thứ tự phiên bản sau cùng nhất của X”

While $WT(X_i) > TS(T)$

i:=i-1;//lùi lại

If $RT(X_i) > TS(T)$

Rollback T//Hủy và khởi tạo TS mới

Else

Tạo phiên bản X_{i+1} ;

Write(X_{i+1});

$RT(X_{i+1}) = 0$;//chưa có ai đọc

$WT(X_{i+1}) = TS(T)$;

Ví dụ

T_1 $TS=150$	T_2 $TS=200$	T_3 $TS=175$	T_4 $TS=255$	A_0 $RT=0$ $WT=0$	A_1	A_2
Read(A)				$RT=150$ $WT=0$		
Write(A)					$RT=0$ $WT=150$	
	Read(A)				$RT=200$ $WT=150$	
	Write(A)					$RT=0$ $WT=200$
		Read(A)			$RT=200$ $WT=150$	
			Read(A)			$RT=255$ $WT=200$

Ví dụ (tt)

T_1 TS=100	T_2 TS=200	A_0 RT=0 WT=0	B_0 RT=0 WT=0	A_2	B_1
Read(A)		RT=100 WT=0			
	Write(A)		RT=0 WT=200	RT=0 WT=200	
	Write(B)				RT=0 WT=200
Read(B)				RT=100 WT=0	
Write(A)			RT=0 WT=100		



Nhận thời gian nhiều phiên bản (tt)

■ Nhận xét

- Thao tác đọc
 - * Giao tác T chỉ đọc giá trị của phiên bản do T hay những giao tác trước T cập nhật
 - * T không đọc giá trị của các phiên bản do các giao tác sau T cập nhật
→ Thao tác đọc không bị rollback
- Thao tác ghi
 - * Thực hiện được thì chèn thêm phiên bản mới
 - * Không thực hiện được thì rollback
- Tốn nhiều chi phí tìm kiếm, tốn bộ nhớ
- Nên giải phóng các phiên bản quá cũ không còn được các giao tác sử dụng