IS210 – Chương 4

Điều khiển đồng thời
Concurrency Control

Trương Thu Thủy

Nội dung

- Các vấn đề trong truy xuất đồng thời
- Các kỹ thuật điều khiển đồng thời
  - Khóa (Locking)
  - Nhãn thời gian (Timestamp)
  - Xác nhận hợp lệ (Validation)
- Deadlock

1

Các vấn đề trong quản lý truy xuất đồng thời

Gồm bốn vấn đề

- Mất dữ liệu cập nhật (Lost updated)
- Dữ liệu đọc không lặp lại (Unrepeatable read)
- Bóng ma (Phantom)
- Đọc dữ liệu rác (Dirty read)

Mất dữ liệu cập nhật – Lost updated

- Xét hai giao tác T_1, T_2 và đơn vị dữ liệu A (A=50)
  - T_1: Read(A); A:=A+10; Write(A)
  - T_2: Read(A); A:=A+20; Write(A)
- Hai giao tác thực hiện đồng thời
- Giá trị của A được thay đổi ở t_3 và được ghi ở t_5 bởi T_1 bị mất
- Giá trị này được ghi đè ở t_3

A=50	T_1	T_2
t_1	Read(A)	
t_2		Read(A)
t_3	A:=A+10	
t_4		A:=A+20
t_5	**Write(A)**	
t_6		**Write(A)**
	A=60	A=70

Dữ liệu đọc không lặp lại - Unrepeatable read

- Xét hai giao tác T_1, T_2 và đơn vị dữ liệu A (A=50)
  - T_1: Read(A); A:=A+10; Write(A)
  - T_2: Read(A); Read(A)
- Hai giao tác thực hiện đồng thời
  - T_2 đọc A hai lần
  - Kết quả hai lần đọc là khác nhau

A=50	T_1	T_2	
t_1	Read(A)		
t_2		**Read(A)**	A=50
t_3	A:=A+10		
t_4		Print(A)	A=50
t_5	Write(A)		
t_6		**Read(A)**	A=60
t_7		Print(A)	A=60

Bóng ma - Phantom

- Xét 2 giao tác T_1 và T_2 được xử lý đồng thời
  - A là một tập các đơn vị dữ liệu a_1, a_2, a_3, a_4,…
  - T_1 xử lý trên toàn bộ tập A
    Khi T_1 đang xử lý, T_2 thêm một hay một số phần tử trong tập A
  - T_1 khi xem dữ liệu sẽ thấy dữ liệu mới thêm khác với ban đầu

  → bóng ma

Đọc dữ liệu rác - Dirty read

- Xét hai giao tác T_1, T_2 và đơn vị dữ liệu A (A=50)
  - T_2 đọc dữ liệu được ghi bởi T_1 nhưng sau đó T_1 hủy bỏ

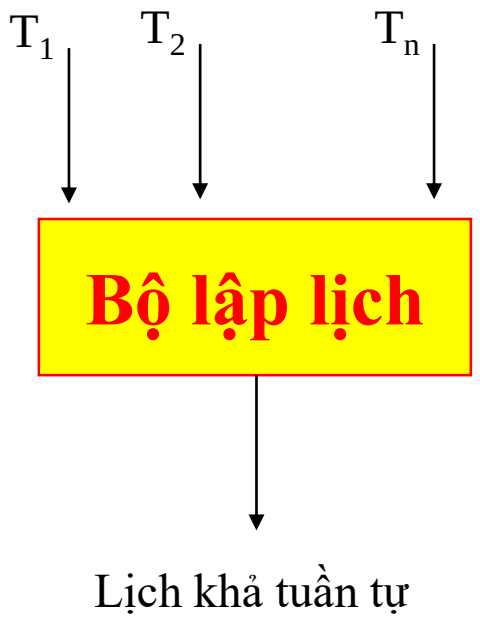
A=50	T_1	T_2	
t_1	Read(A)		
t_2	A:=A+10		
t_3	**Write(A)**		
t_4		**Read(A)**	Dirty read
t_5		Print(A)	
t_6	**Abort**		

Nhận xét

- Một bộ lập lịch phải ngăn chặn được:
  - Lịch không khả tuần tự
  - Những vấn đề trong điều khiển đồng thời
- Làm sao tạo ra lịch S?
  - Các kỹ thuật điều khiển đồng thời

Các kỹ thuật điều khiển đồng thời

- Là những kỹ thuật cho phép bộ lập lịch sử dụng để tạo ra một lịch khả tuần tự từ n giao tác thực hiện đồng thời



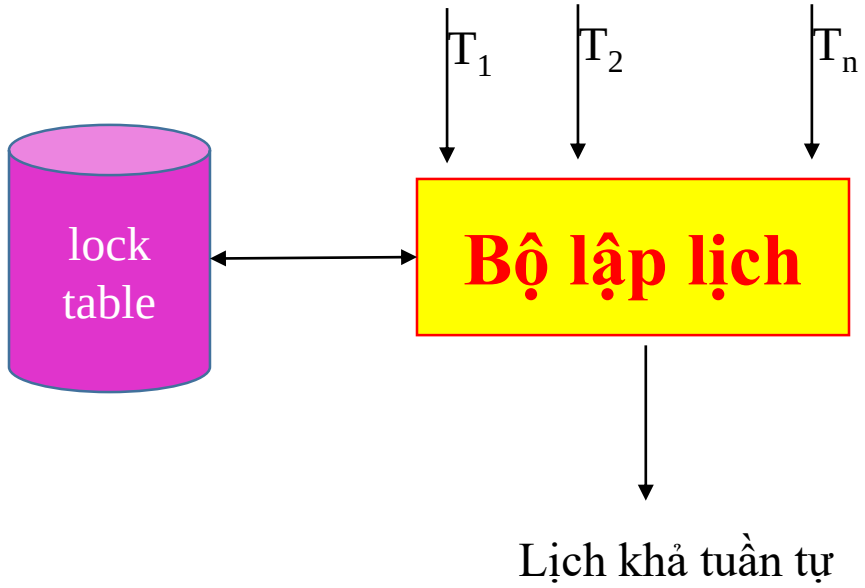
Kỹ thuật khóa

Nhãn thời gian

Xác nhận hợp lệ

Kỹ thuật khóa

- Bộ lập lịch với cơ chế khóa (locking scheduler)
  - Có thêm hai hành động:
    - Lock: cấp phát khóa
    - Unlock: giải phóng khóa
  - Các khóa được thể hiện trong bảng khóa (lock table)



Kỹ thuật khóa

- Một giao tác trước khi muốn đọc/ghi lên một đơn vị dữ liệu phải yêu cầu khóa (lock) trên đơn vị dữ liệu này
  - Ký hiệu: lock(A) hay l(A)
- Sau khi thao tác xong phải giải phóng (unlock) trên đơn vị dữ liệu đó
  - Ký hiệu: unlock(A) hay u(A)

Lock table

Element	Transaction
A	T_1

Kỹ thuật khóa

- Quy tắc: việc sử dụng khóa (lock) phải đảm bảo
  - (1) Tính nhất quán của các **giao tác**
    - T_1: …**l(A)**…r(A)/w(A)…**u(A)**…
  - (2) Tính hợp lệ của **lịch thao tác**
    - S: …l_i(A)…….. u_i(A)

Không có l_j(A)

Kỹ thuật khóa

- Ví dụ
- Giao tác T_1, T_2 có nhất quán không?
- Lịch S có hợp lệ không?

T1, T2 nhat quan
hop le

S

T_1	T_2
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,s)
	s:=s*2
	Write(B,s)
	Unlock(B)
Lock(B)	
Read(B,t)	
t:=t+100	
Write(B,t)	
Unlock(B)	

Kỹ thuật khóa

- Giao tác nào nhất quán?
- Lịch S có hợp lệ không?

ko nhat quan
ko hop le

S

T_1	T_2	T_3
Lock(A)		
Lock(B)		
Read(A)		
Read(B)		
	Lock(B)	
Unlock(A)		
Unlock(B)		
	Read(B)	
	Write(B)	
	Unlock(B)	
		Lock(B)
		Read(B)
		Unlock(B)

Kỹ thuật khóa

- Giao tác nào nhất quán?
- Lịch S có hợp lệ không?

ko nhat quan
ko hop le

S

T_1	T_2	T_3
Lock(A)		
Read(A)		
Write(B)		
Unlock(A)		
	Lock(B)	
	Read(B)	
	Write(B)	
		Lock(B)
		Read(B)
		Unlock(B)

- Nếu S hợp lệ

thì S có khả tuần tự không?

S

T_1	T_2	A	B
		25	25
Lock(A);Read(A,t)			
t:=t+100			
Write(A,t); **Unlock(A)**		125	
	Lock(A);Read(A,s)		
	s:=s*2		
	Write(A,s); **Unlock(A)**	250	
	Lock(B);Read(B,s)		
	s:=s*2		
	Write(B,s); **Unlock(B)**		50
Lock(B);Read(B,t)			
t:=t+100			
Write(B,t); **Unlock(B)**			150

Không khả tuần tự

- Lock(B) được thực hiện trước khi Unlock A?

S

T_1	T_2	A	B
		25	25
Lock(A);Read(A,t)	Không hợp lệ		
t:=t+100			
Write(A,t);		125	
Unlock(A)			
	Lock(A);Read(A,s)		
	s:=s*2 Delay		
	Write(A,s);	250	
	Unlock(A)		
	Lock(B)		
	Read(B,s)		
	s:=s*2		125 250
	Write(B,s); **Unlock(B)**		
Lock(B)			
Read(B,t)			
t:=t+100			
Write(B,t); **Unlock(B)**			250

S

- Lock(B) được thực hiện trước khi Unlock A?

T_1	T_2	A	B
		25	25
Lock(A);Read(A,t)			
t:=t+100			
Write(A,t); **Lock(B)**; **Unlock(A)**		125	
	Lock(A);Read(A,s)		
	s:=s*2		
	Write(A,s);	250	
	Lock(B);		
Read(B,t)			
t:=t+100			
Write(B,t); **Unlock(B)**			125
	Unlock(A)		
	Read(B,s)		
	s:=s*2		
	Write(B,s); **Unlock(B)**		250

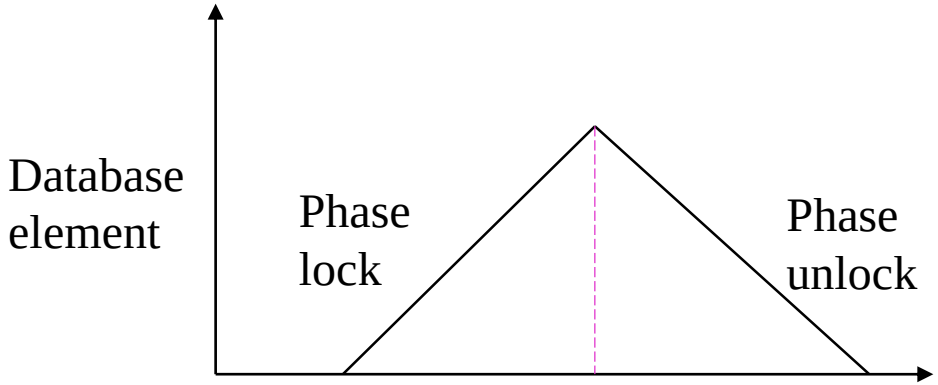
Phải chờ đến lúc này

Khóa hai giai đoạn – Two-phase Locking (2PL)

Thứ tự hoạt động của khóa trong **giao tác**

- (3) 2PL – Two phase locking: tất cả các hành động lock phải thực hiện trước unlock
- **T_i: ……$l_i(A)$……$u_i(A)$……**

no unlock — no lock



Ví dụ

T1	T2
Lock(A)	**Lock(B)**
Read(A)	Read(B)
Lock(B)	**Lock(A)**
Read(B)	Read(A)
B:=B+A	**Unlock(B)**
Write(B)	A:=A+B
Unlock(A)	Write(A)
Unlock(B)	**Unlock(A)**

2PL transactions

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

Không thỏa 2PL transactions

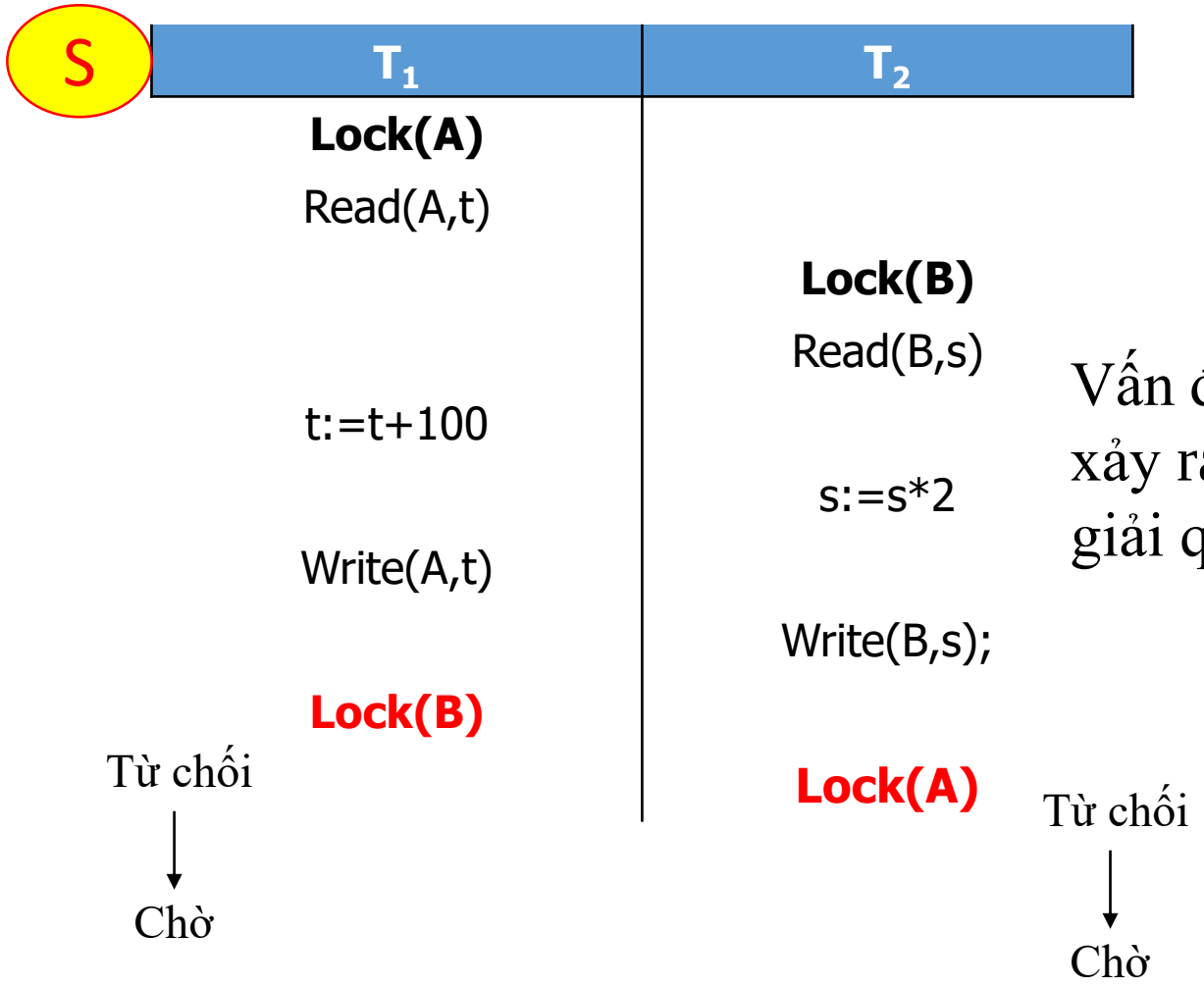
Định lý

- Nếu một lịch S thỏa:
  - (1) Tính nhất quán của các giao tác
  - (2) Tính hợp lệ của lịch
  - (3) Các giao tác phải là 2PL
- Thì S sẽ conflict-serializable

Chứng minh

Khóa hai giai đoạn

- Vấn đề Deadlock



Vấn đề **Deadlock** xảy ra – không được giải quyết bởi 2PL

Kỹ thuật khóa

- Vấn đề
  - Một giao tác T phải lock trên A ngay cả khi T chỉ muốn read A mà không write A
    - Nhưng, không thể tránh việc lock đơn vị dữ liệu, vì không khả tuần tự có thể xảy ra
  - Mặt khác, không có lý do gì khiến một số giao dịch không thể đọc X cùng một lúc, miễn là không có giao dịch nào được phép ghi X .

T_1	T_2
Lock(A)	
Read(A)	
Unlock(A)	
	Lock(A)
	Read(A)
	Unlock(A)

Chế độ khóa – Lock mode

- Xem xét một bộ lập lịch khóa (locking scheduler) sử dụng hai loại khóa khác nhau
  - **Khóa chia sẻ** (Shared lock)
    - Read lock
    - rl(X) hay Rlock(X)
  - **Khóa độc quyền** (eXclusive lock)
    - Write lock
    - wl(X) hay Wlock(X)
  - Giải phóng
    - Unlock
    - u(X) hay Unlock(X)

Chế độ khóa – Lock mode

- Đối với mỗi đơn vị dữ liệu X
  - Hoặc là một exclusive lock trên X
  - Hoặc không có exclusive lock nào nhưng có thể nhiều shared lock trên X
- Nếu write(X)
  - Cần phải có Wlock(X)
- Nếu read(X)
  - Cần có Rlock(X) hoặc Wlock(X)

Chế độ khóa – Lock mode

- Ba yêu cầu trong chế độ khóa
  - (1) Tính nhất quán của giao tác

T_i: … rl(X)/wl(X)…r(X)… u(X)

T_i: … wl(X)…w(X)… u(X)

Chế độ khóa – Lock mode

- Ba yêu cầu trong chế độ khóa
  - (2) Tính hợp lệ của lịch thao tác

S: … $rl_i(X)$…… $u_i(X)$…

Không **$wl_j(X)$**

S: … $wl_i(X)$…… $u_i(X)$…

Không **$wl_j(X)$**
Không **$rl_j(X)$**

Chế độ khóa – Lock mode

- Ba yêu cầu trong chế độ khóa
  - (3) Two-phase locking

T: … rl(X)/wl(X)…… u(X)…

Không unlock

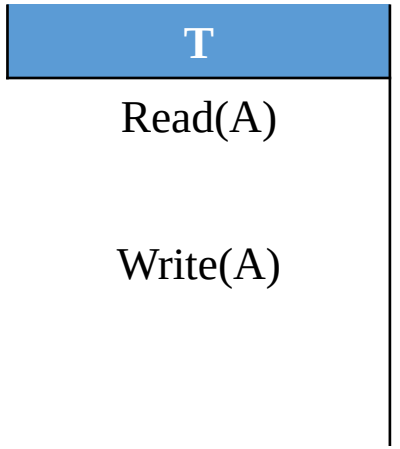
Không lock

Chế độ khóa – Lock mode

- Cho phép giao tác nắm giữ cả shared lock và exclusive lock trên cùng đơn vị dữ liệu
  - Nếu giao tác biết trước những lock mà nó cần
    - Chỉ sử dụng một exclusive lock
    - **T: … wl(A) … r(A) … w(A) … u(A) …**
  - Nếu không
    - Cả shared lock và exclusive lock được sử dụng
    - **T: … rl(A) … r(A) … wl(A) … w(A) … u(A) …**

Nâng cấp khóa

Lúc này T không còn giữ read lock nữa thay vào đó là write lock



Thảo luận

- S là conflict-serializable
- Thứ tự (T_2,T_1)
  - Ngay cả khi T_1 thực hiện trước
  - Nhưng, T_2 unlock trước T_1

S

T_1	T_2
RLock(A); Read(A)	
	RLock(A); Read(A)
	Rlock(B); Read(B)
Wlock(B) **Từ chối**	
Đợi	Unlock(A); Unlock(B)
Wlock(B); Read(B);Write(B)	
Unlock(A);Unlock(B)	

Định lý

- Trong hệ thống với shared lock và exclusive lock
  - Nếu S thỏa
    - (1) Tính nhất quán của các giao tác
    - (2) Tính hợp lệ của lịch
    - (3) Two-phase lock của giao tác
  - Thì S conflict-serializable

Ma trận tương thích

- Ma trận tương thích là một cách thuận tiện để mô tả các cách thức quản lý khóa.
  - Hàng ngang diễn tả một khóa đang được nắm giữ trên đơn vị dữ liệu X
  - Hàng dọc diễn tả chế độ khóa yêu cầu trên X

		Lock requested	
		Share	**eXclusive**
Lock held in mode	**Share**	Yes	No
	eXclusive	No	No

Ví dụ

T_1	T_2
RLock(A); Read(A)	
	RLock(A); Read(A)
	Rlock(B); Read(B)
Wlock(B) Từ chối	
	Unlock(A); Unlock(B)
Đợi	
Wlock(B); Read(B);Write(B)	
Unlock(A) Unlock(B)	

Sử dụng lock thông thường

T_1	T_2
RLock(A); Read(A)	
	RLock(A); Read(A)
	Rlock(B); Read(B)
Rlock(B); Read(B)	
Wlock(B) Từ chối	
	Unlock(A) Unlock(B)
Đợi	
Wlock(B);Write(B)	
Unlock(A) Unlock(B)	

Sử dụng nâng cấp lock
Upgrade lock

Bài tập

- Giao tác nào không thỏa 2PL
- Lịch S có khả tuần tự không?

cả 2 giao tác đều ko thoả 2PL
lịch S ko khả tuần tự

S

T_1	T_2
RLock(A); Read(A)	
Unlock(A)	
	RLock(B); Read(B)
	Unlock(B)
	Wlock(A); Read(A)
	A:=A+B
	Write(A)
	Unlock(A)
Wlock(B); Read(B)	
B:=B+A	
Write(B)	
Unlock(B)	

Bài tập

- Giao tác nào không thỏa 2PL
- Lịch S có khả tuần tự không?

S

T_1	T_2	T_3	T_4
	RL(A)		
		RL(A)	
	WL(B)		
	U(A)		
		WL(A)	
	U(B)		
RL(B)			
		U(A)	
			RL(B)
RL(A)			
			U(B)
WL(C)			
U(A)			
			WL(B)
			U(B)
U(B)			
U(C)			

Nội dung

- Các vấn đề trong truy xuất đồng thời
- Các kỹ thuật điều khiển đồng thời
  - Khóa (Locking)
  - **Nhãn thời gian (Timestamps)**
    - **Nhãn thời gian toàn phần**
    - **Nhãn thời gian riêng phần**
    - **Nhãn thời gian nhiều phiên bản (multiversion)**
  - Xác nhận hợp lệ (Validation)
- Deadlock

2

Nhãn thời gian

Nhãn thời gian - timestamps

- Ý tưởng:
  - Không có hành động nào vi phạm khả tuần tự
  - Khi xảy ra sự cố thì hủy bỏ và thực hiện lại giao tác
- Chọn thứ tự thực hiện các giao tác bằng nhãn thời gian
  - Mỗi giao tác T sẽ được gán một số duy nhất, một nhãn thời gian TS(T)
    - Tại thời điểm giao tác T bắt đầu
  - Thứ tự của nhãn thời gian sẽ tăng dần
    - Những giao tác bắt đầu sau sẽ có giá trị nhãn thời gian lớn hơn giao tác bắt đầu trước đó

Nhãn thời gian - timestamps

- Để gán nhãn
  - Sử dụng đồng hồ của máy tính (system clock)
  - Sử dụng bộ đếm của bộ lập lịch. Khi giao tác bắt đầu, bộ đếm sẽ tăng thêm 1
- Chiến lược cơ bản
  - Nếu $ST(T_i) < ST(T_j)$ thì lịch thao tác phải tương đương với lịch tuần tự $\{T_i, T_j\}$

Nhãn thời gian toàn phần - Total timestamps

- Mỗi giao tác T được gán một nhãn TS(T) ghi lại thời gian T bắt đầu
- Mỗi đơn vị dữ liệu X có một nhãn TS(X), ghi thời gian của TS(T) của T đã thực hiện read/write thành công sau cùng lên X.
- Khi đến lượt giao tác T thao tác lên X, TS(T) và TS(X) sẽ được so sánh

Nhãn thời gian toàn phần - Total timestamps

Read(T, X)

```
If TS(X) <= TS(T)
    Read(X);
    //permit to read X
    TS(X):= TS(T);
Else
    Abort {T};
    //cancel T
    //reset ST
```

Write(T, X)

```
If TS(X) <= TS(T)
    Write(X);
    //permit to write X
    TS(X):= TS(T);
Else
    Abort {T};
    //cancel T
    //reset TS
```

Ví dụ

	T_1 300 $TS(T_1)$=~~100~~	T_2 $TS(T_2)$=200	A TS(A)=0	B TS(B)=0	
1	Read(A)		TS(A)=100		TS(A) <= $TS(T_1)$: T_1 đọc A
2		Read(B)		TS(B)=200	TS(B) <= $TS(T_2)$: T_2 đọc B
	A:=A*2				
3	Write(A)		TS(A)=100		TS(A) <= $TS(T_1)$: T_1 ghi A
		B:=B+20			
4		Write(B)		TS(B)=200	TS(B) <= $TS(T_2)$: T_1 ghi B
5	Read(B)				TS(B) > $TS(T_1)$: T_1 không thể đọc B

Abort

Khởi tạo lại $TS(T_1)$ ➔$TS(T_2) < TS(T_1)$

Lịch khả tuần tự theo thứ tự $\{T_2, T_1\}$

Ví dụ

T_1 TS(T_1)=100	**T_2** TS(T_2)=200	**A** TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=200
	Write(A)	TS(A)=200
Write(A)		

T_1 bị hủy và bắt đầu lại với một nhãn thời gian mới

T_1 TS(T_1)=100	**T_2** TS(T_2)=200	**A** TS(A)=0
Read(A)		TS(A)=100
	Read(A)	TS(A)=200
	Read(A)	TS(A)=200
Read(A)		

T_1 bị hủy và bắt đầu lại với một nhãn thời gian mới

Không có sự phân biệt giữa hoạt động read và write
→T_1 vẫn bị hủy bỏ và bắt đầu lại

Nhãn thời gian riêng phần – Partial timestamps

- Nhãn thời gian của đơn vị dữ liệu X được chia làm 2 loại
  - RT(X) – read timestamp
    - Ghi nhận giá trị TS(T) gần nhất đọc X thành công
  - WT(X) – write timestamp
    - Ghi nhận giá trị TS(T) gần nhất ghi X thành công
- Ngoài ra bộ lập lịch còn quản lý một bit thể hiện trạng thái của giao tác
  - C(X) – commit bit
    - True nếu dữ liệu X đã được commit. Ngược lại false

Nhãn thời gian riêng phần – Partial timestamps

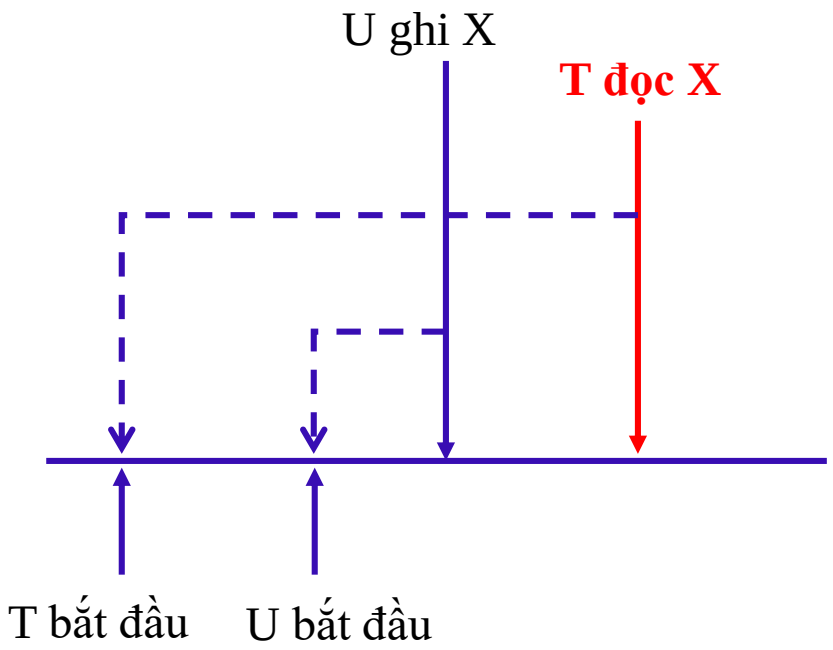
- RT(X) – read timestamp
  - Ghi nhận giá trị TS(T) gần nhất đọc X thành công
- WT(X) – write timestamp
  - Ghi nhận giá trị TS(T) gần nhất ghi X thành công

T_1 $TS(T_1)=100$	T_2 $TS(T_2)=200$	T_3 $TS(T_3)=300$	A RT(A)=0 WT(A)=0
Read(A)			RT(A)=100 WT(A)=0
	Read(A)		RT(A)=200 WT(A)=0
Read(A)			RT(A)=200 WT(A)=0
	Write(A)		RT(A)=200 WT(A)=200
		Write(A)	RT(A)=200 WT(A)=300

Nhãn thời gian riêng phần

- Công việc của bộ lập lịch
  - Gán, cập nhật nhãn thời gian RT(X), WT(X), C(X)
  - Kiểm tra khi nào hoạt động read/write xảy ra
  - Xử lý tính huống
    - Đọc quá trễ (Read too late)
    - Ghi quá trễ (Write too late)
    - Đọc dữ liệu rác (Dirty read)
    - Quy tắc ghi Thomas (Thomas write rule)

Đọc quá trễ – Read too late

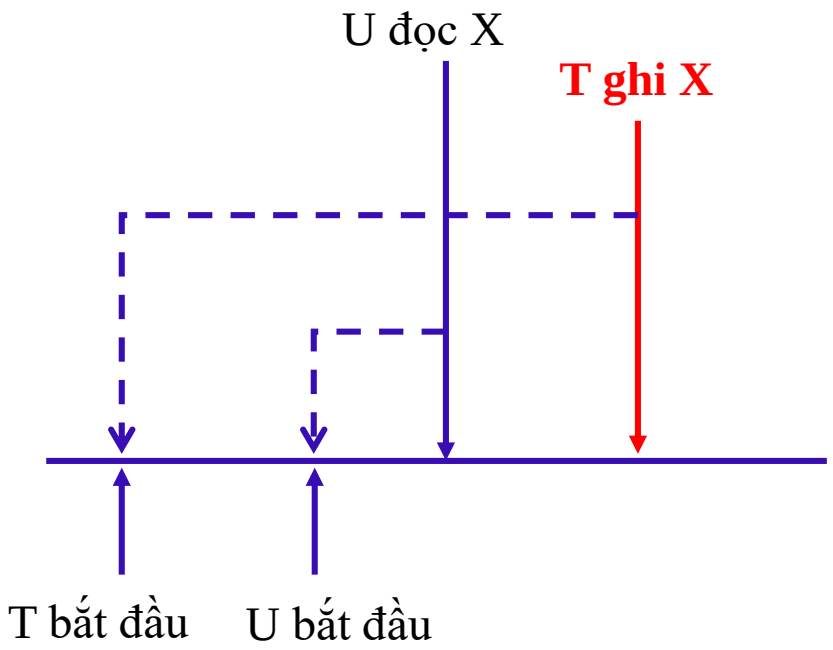


- TS(T) < TS(U)
- U ghi X trước, T đọc X sau
  - TS(T)<WT(X)
- T không thể đọc giá trị được ghi bởi U

→ Abort T

T TS(T)=100	**U** TS(U)=200
Write(A)	
	Write(A)
Read(A)	

Ghi quá trễ – Write too late

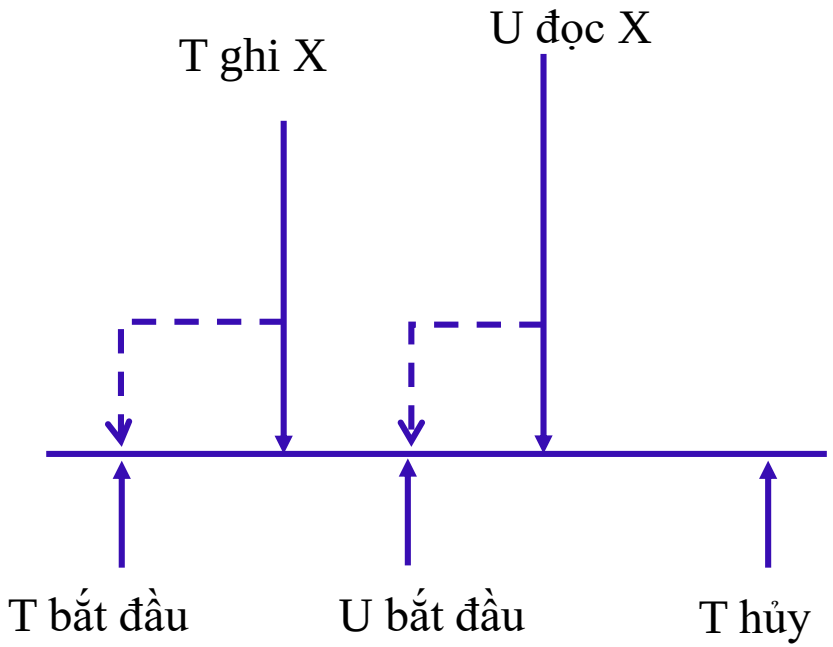


- TS(T) < TS(U)
- U đã đọc X trước, T đang ghi X sau
  - WT(X) <TS(T)<RT(X)
- U phải đọc giá trị được ghi bởi T

→Abort T

V TS(V)=50	**T** TS(T)=100	**U** TS(U)=200
Write(A)		
		Read(A)
	Write(A)	

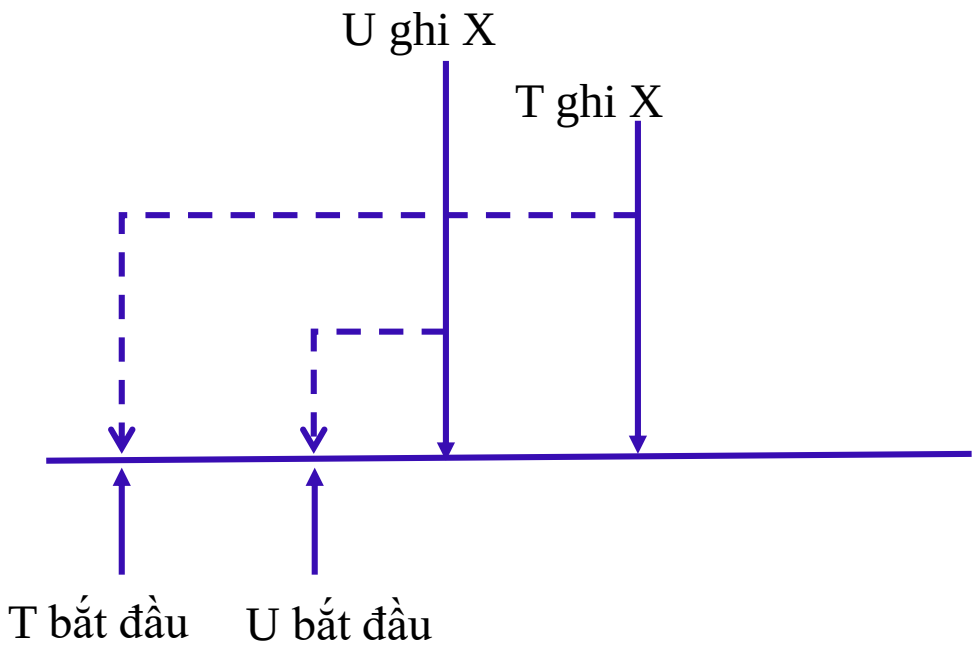
Đọc dữ liệu rác – Dirty read



- TS(T) < TS(U)
- T ghi X trước, U đọc X sau
- Nhưng T bị hủy bỏ → giá trị X mà U đọc là giá trị rác.
- Giải pháp: trì hoãn U cho đến khi T commit hoặc abort

T TS(T)=100	**U** TS(U)=200
Write(A)	
	Read(A)
Abort	

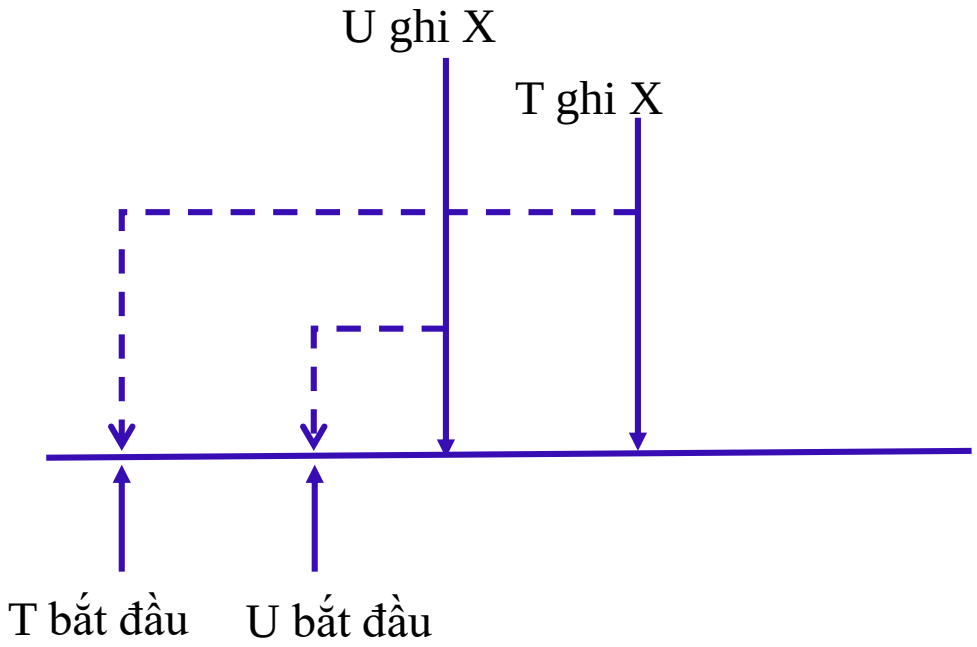
Quy tắc ghi Thomas – Thomas write rule



- TS(T) < TS(U)
- U ghi X trước, T đang ghi X sau
  - TS(T) < WT(X)
- Sau khi T ghi X cũng không làm được gì vì:
  - Không có giao tác nào đọc giá trị của T (nếu có thì sẽ đọc quá trễ)
  - Các giao tác khác sau T và U đều muốn đọc giá trị X được ghi bởi U

→Bỏ qua thao tác ghi của T

Quy tắc ghi Thomas – Thomas write rule



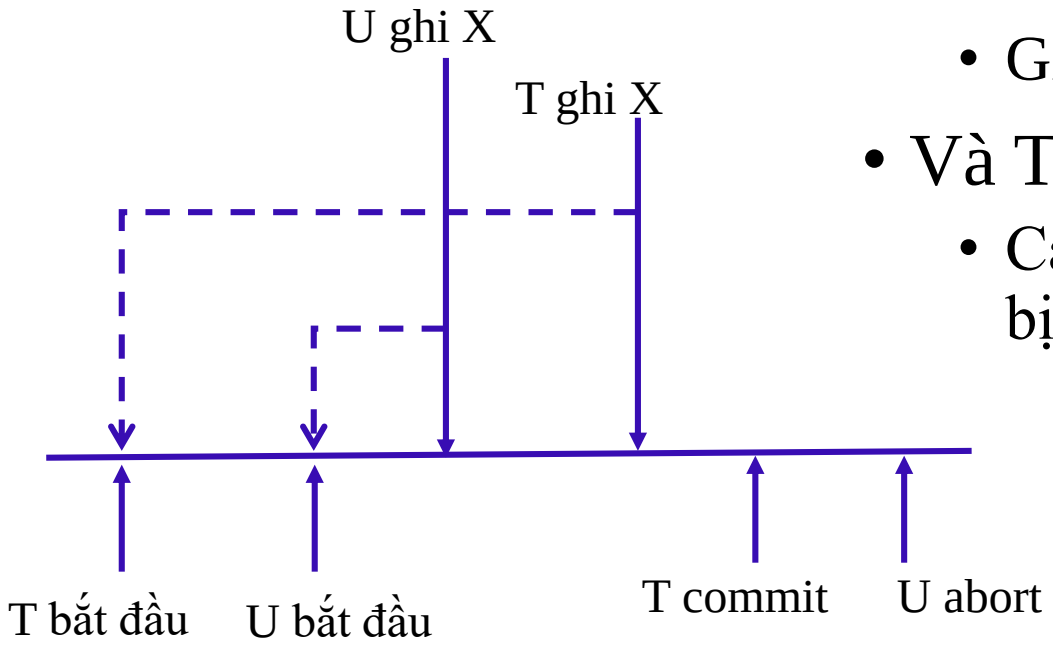
- Sau khi T ghi X cũng không làm được gì vì:
  - Không có giao tác nào đọc giá trị của T (nếu có thì sẽ đọc quá trễ)
  - Các giao tác khác sau T và U đều muốn đọc giá trị X được ghi bởi U

→Bỏ qua thao tác ghi của T

T TS(T)=100	**U** TS(U)=200
……..	……
	Write(A)
Write(A)	

Bỏ qua →

Quy tắc ghi Thomas – Thomas write rule



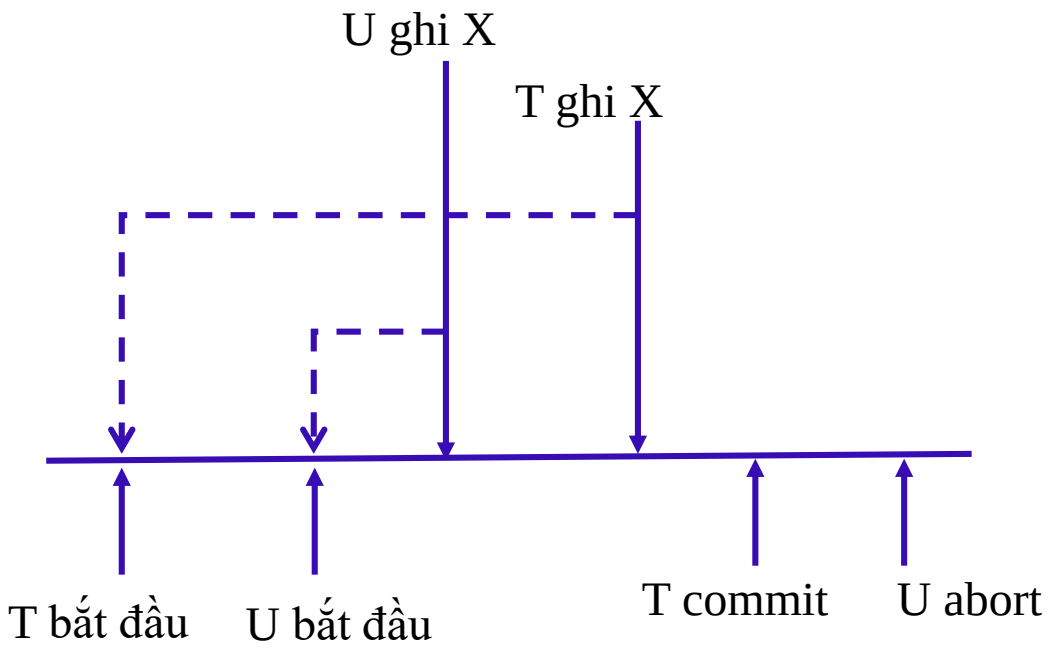
- Trường hợp U ghi X nhưng sau đó bị hủy bỏ
  - Giá trị của X phải được loại bỏ
  - Giá trị trước đó của X nên được phục hồi
- Và T commit
  - Cần khôi phục lại giá trị X từ T mà lệnh ghi X đã bị bỏ qua

T TS(T)=100	**U** TS(U)=200
……..	……
	Write(A)
Write(A)	
	Abort
Commit	

Bỏ qua → Write(A)

Phục hồi A → Abort

Quy tắc ghi Thomas – Thomas write rule



- Giải pháp cho việc T ghi X
  - Việc ghi này là sơ bộ và sẽ gỡ bỏ nếu T bị hủy bỏ.
  - Giá trị C(X) được gán là false
  - Bộ lập lịch sẽ tạo một bản copy giá trị của X cũ và giá trị WT(X) trước đó

Tóm tắt

- 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 một nhãn thời gian (timestamp) mới
  - Trì hoãn T, sau đó sẽ quyết định hủy bỏ T hay là đáp ứng yêu cầu

Nhãn thời gian riêng phần – Partial timestamps

- Quy tắc

Read(T,X)

```
If  WT(X) <= TS(T)
        Read(X);//allow to read X
        RT(X):= max(RT(X),TS(T));
Else
        Rollback{T};
        //cancel T & reset TS t
```

Write(T,X)

```
If RT(X) <= TS(T)
        If WT(X) <= TS(T)
                Write(X);//allow to write X
                WT(X):= TS(T);
        //Else do nothing
Else
        Rollback{T};
        //cancel T & reset TS
```

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	
Read(A)		RT(A)=100 WT(A)=0			$WT(A) < TS(T_1)$ T_1 đọc A
	Read(B)		RT(B)=200 WT(B)=0		$WT(B) < TS(T_2)$ T_2 đọc B
Write(A)		RT(A)=100 WT(A)=100			$RT(A) < TS(T_1)$; $WT(A) = TS(T_1)$ T_1 ghi A
	Write(B)		RT(B)=200 WT(B)=200		$RT(B) < TS(T_2)$; $WT(B) = TS(T_2)$ T_2 ghi B
	Read(C)			RT(C)=200 WT(C)=0	$WT(B) < TS(T_2)$ T_2 đọc C
Read(C)				RT(C)=200 WT(C)=0	$WT(B) < TS(T_1)$ T_1 đọc C
Write(C)					$RT(B) < TS(T_1)$ T_1 không ghi C được

↓ rollback

Ví dụ 2

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

Rollback

Giá trị của A đã được ghi bởi T_1
➔ T_3 không cần ghi A

Bài tập

- Cho các lịch sau:

a. st_1; st_2; $r_1(A)$; $r_2(B)$; $w_2(A)$; $w_1(B)$;

b. st_1; $r_1(A)$; st_2; $w_2(B)$; $r_2(A)$; $w_1(B)$;

c. st_1; st_2; st_3; $r_1(A)$; $r_2(B)$; $w_1(C)$; $r_3(B)$; $r_3(C)$; $w_2(B)$; $w_3(A)$;

d. st_1; st_3; st_2; $r_1(A)$; $r_2(B)$; $w_1(C)$; $r_3(B)$; $r_3(C)$; $w_2(B)$; $w_3(A)$;

- Hãy điều khiển việc truy xuất đồng thời của các giao tác dùng kỹ thuật timestamp riêng phần (partial)

Nhãn thời gian riêng phần – Partial timestamps

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

Rollback

Nhãn thời gian riêng phần – Partial timestamps

- Nhận xét
  - Hành động $Read_3(A)$ làm T_3 bị hủy do T_3 đọc quá trễ
  - Giá trị A ghi bởi T_1 (sau đó bị ghi đè bởi T_2) phù hợp với T_3 để đọc
  - Nếu có giá trị cũ của T_1 thì T_3 sẽ đọc được và không bị hủy

T_1 TS(T_1)=150	T_2 TS(T_2)=200	T_3 TS(T_3)=175	T_4 TS(T_4)=255
Read(A)			
Write(A)			
	Read(A)		
	Write(A)		
		Read(A)	
			Read(A)

Rollback

Nhãn thời gian nhiều phiên bản- Multiversion Timestamps

- Ý tưởng
  - Bên cạnh lưu trữ giá trị hiện hành của A, vẫn giữ lại các giá trị được sao lưu trước kia của A (nhiều phiên bản của A)
  - Giao tác T sẽ đọc được giá trị của A ở một phiên bản thích hợp nào đó

Nhãn thời gian nhiều phiên bản- Multiversion Timestamps

- Mỗi phiên bản của một đơn vị dữ liệu sẽ có
  - RT(X): ghi nhận giao tác T gần nhất đọc X thành công
  - WT(X): ghi nhận giao tác T gần nhất ghi X thành công
- Khi giao tác T yêu cầu thao tác trên X
  - Tìm một phiên bản phù hợp của X
  - Đảm bảo khả tuần tự
- Một phiên bản của X sẽ được tạo khi hành động ghi X thành công

Nhãn thời gian nhiều phiên bản- Multiversion Timestamps

- Read(T, X)

```
i="the latest version number of X"
While WT(Xi) > TS(T)
        i:=i-1;//get back
Read(Xi);
RT(Xi):= max(RT(Xi), TS(T));
```

Write(T, X)

```
i="the latest version number of X"
While WT(Xi) > TS(T)
        i:=i-1;//get back
If RT(Xi) > TS(T)
        Rollback T;//abort & create new TS
Else
        Create version Xi+1;
        Write(Xi+1);
        RT(Xi+1) = 0;//no transaction reads yet
        WT(Xi+1) = TS(T);
```

Ví dụ 1

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 2

T_1 TS=100	T_2 TS=200	A_0 RT=0 WT=0	A_1	A_2	B_0 RT=0 WT=0	B_1
Read(A)		RT=100 WT=0				
	Write(A)			RT=0 WT=200		
	Write(B)					RT=0 WT=200
Read(B)					RT=100 WT=0	
Write(A)			RT=0 WT=100			

Bài tập

- Cho các lịch sau:

a. st_1; st_2; st_3; st_4; $w_1(A)$; $w_2(A)$; $w_3(A)$; $r_2(A)$; $r_4(A)$;

b. st_1; st_2; st_3; st_4; $w_1(A)$; $w_3(A)$; $r_4(A)$; $r_2(A)$;

c. st_1; st_2; st_3; st_4; $w_1(A)$; $w_4(A)$; $r_3(A)$; $w_2(A)$;

- Hãy điều khiển việc truy xuất đồng thời của các giao tác dùng kỹ thuật timestamp nhiều phiên bản (multiversion)

Nhãn thời gian nhiều phiên bản- Multiversion Timestamps

- Nhận xét
  - Hành động đọc
    - Giao tác T chỉ đọc giá trị của phiên bản được cập nhật bởi T hoặc những giao tác trước đó
    - Giao tác T không đọc giá trị của phiên bản được cập nhật sau T
    - → Không gây ra rollback
  - Hành động ghi
    - Thực hiện thêm một phiên bản mới
    - Nếu không thêm được thì rollback
  - Tốn nhiều chi phí tìm kiếm, tốn bộ nhớ
  - Nên giải phóng các phiên bản quá cũ, không còn được sử dụng

Timestamps so với locking

- Nhãn thời gian tốt hơn tong các tình huống mà hầu hết các giao tác là chỉ đọc (read-only) hoặc hiếm có các giao tác xử lý đồng thời đọc và ghi trên cùng một đơn vị dữ liệu
- Trong tình huống có sự xung đột cao thì locking xử lý tốt hơn
- Trong các hệ thống thương mại, bộ lập lịch chia giao tác thành read-only transaction và read/write transaction
- Read/write transaction sử dụng two-phase locking
- Read-only transaction sử dụng multiversion timestamping

3 Xác nhận hợp lệ

Xác nhận hợp lệ - Validation

- Ý tưởng
  - Cho phép các giao tác truy xuất dữ liệu tự do
  - Kiểm tra tính khả tuần tự của các giao tác
    - Trước khi ghi, một tập đơn vị dữ liệu của giao tác sẽ được so sánh với tập đơn vị dữ liệu của các giao tác khác
    - Nếu hợp lệ, giao tác sẽ phải rollback

Xác nhận hợp lệ - Validation

- Các giao tác có ba giai đoạn
- (1) Read: read set – RS(T)
  - Đọc tất cả những đơn vị dữ liệu
  - Ghi vào bộ nhớ tạm
  - Không sử dụng cơ chế khóa
- (2) Validate
  - Kiểm tra tính khả tuần tự
- (3) Write: write set – WS(T)
  - Nếu (2) hợp lệ thì ghi xuống CSDL

Xác nhận hợp lệ - Validation

- Chiến lược
  - Nếu T_1, T_2, T_3, … là thứ tự hợp lệ thì lịch kết quả sẽ conflict-equivalent với một lịch tuần tự $S_s = \{T_1, T_2, T_3, \ldots\}$

Xác nhận hợp lệ - Validation

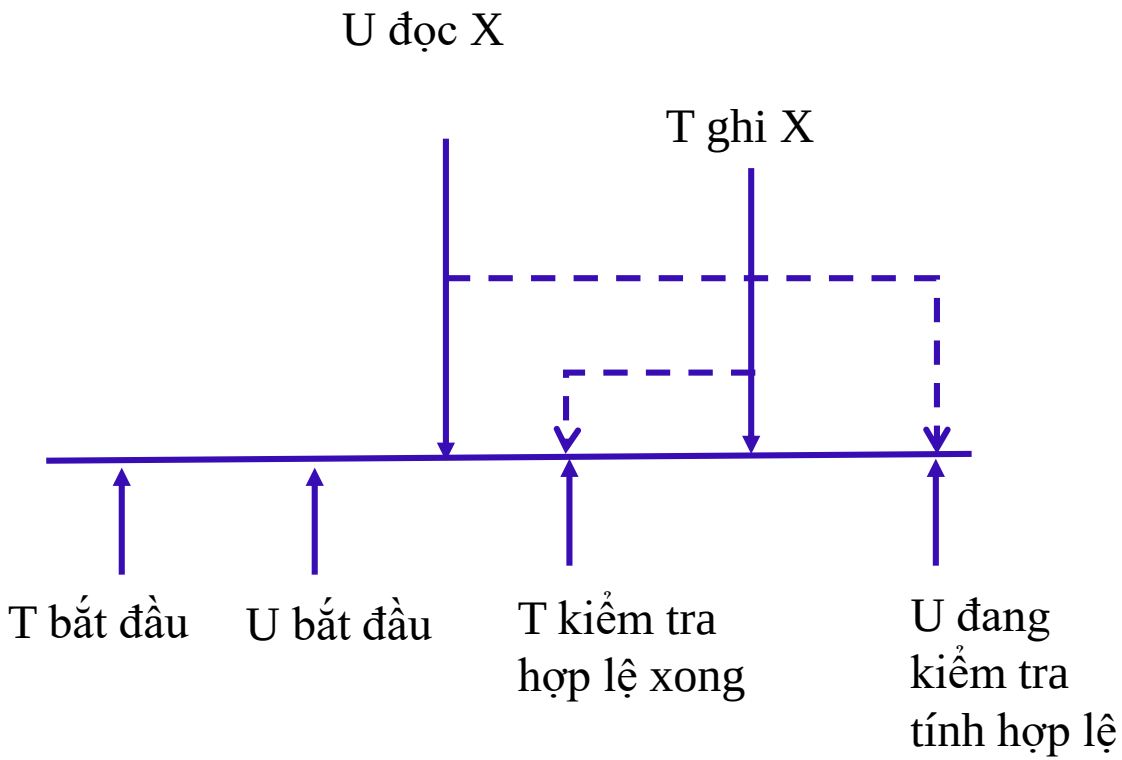
- Bộ lập lịch xét 3 tập hợp
  - START
    - Tập các giao tác đã bắt đầu nhưng chưa kiểm tra hợp lệ xong
    - START(T) ghi nhận thời gian bắt đầu của T
  - VAL
    - Tập các giao tác đã kiểm tra tính hợp lệ (các giao tác đã hoàn tất giai đoạn (2))
    - VAL(T) ghi nhận thời điểm T kiểm tra xong
  - FIN
    - Tập các giao tác đã hoàn tất việc ghi (các giao tác đã hoàn tất giai đoạn (3))
    - FIN(T) ghi nhận thời điểm T hoàn tất

Xác nhận hợp lệ - Validation

T_1	T_2
Read(B)	
	Read(B)
	B:=B-50
	Read(A)
	A:=A+50
Read(A)	
Xác nhận hợp lệ	
Display(A+B)	
	Xác nhận hợp lệ
	Write(B)
	Write(A)

Xác nhận hợp lệ - Validation

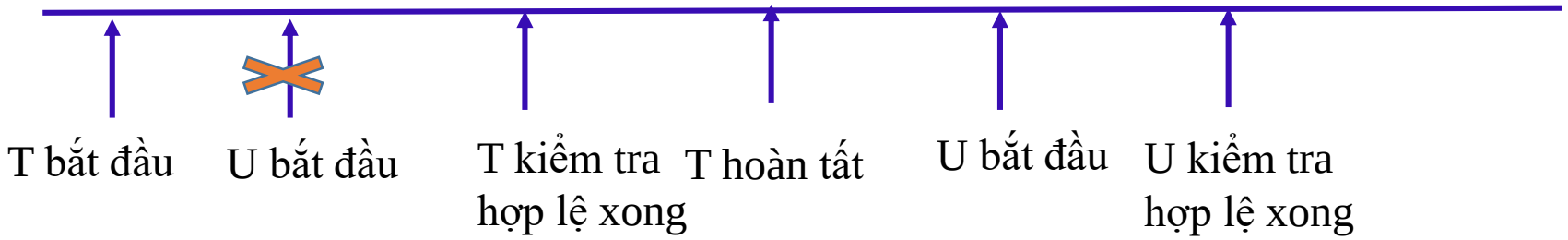
Vấn đề 1



- T đã kiểm tra tính hợp lệ xong
- T chưa hoàn tất ghi thì U bắt đầu đọc
- $RS(U) \cap WS(T) = \{X\}$
- U không thể đọc được giá trị X ghi bởi T
- → Rollback U

Xác nhận hợp lệ - Validation

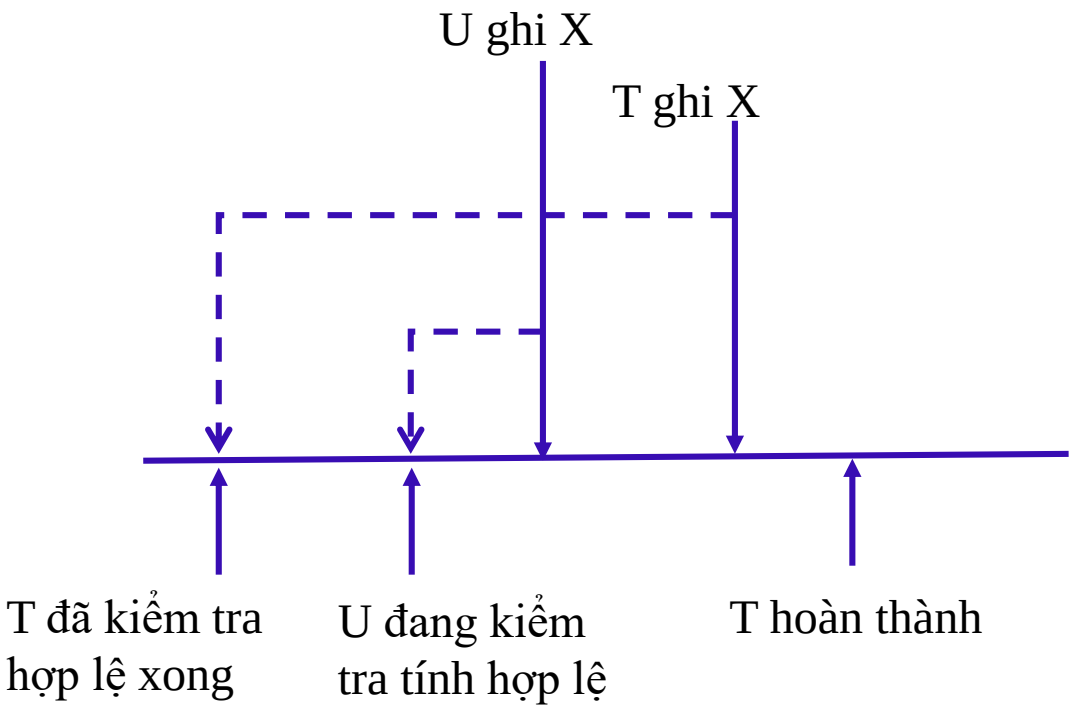
Vấn đề 1



Sau khi T hoàn tất thì U mới bắt đầu

Xác nhận hợp lệ - Validation

Vấn đề 2

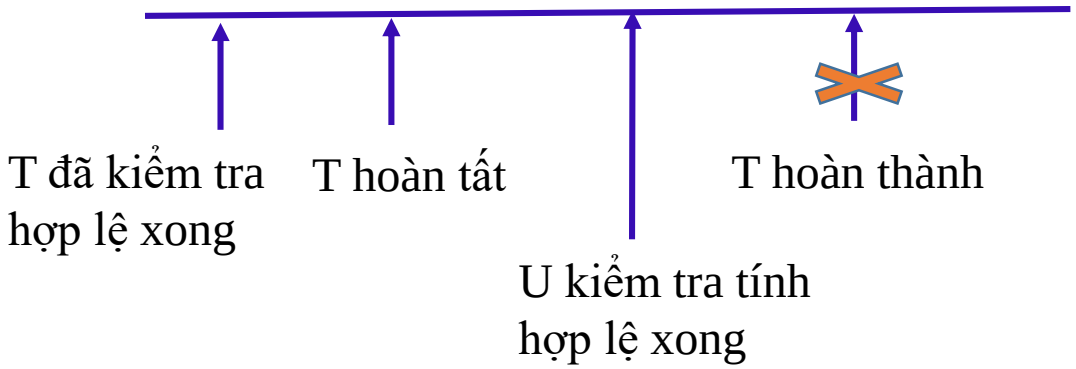


- T đã kiểm tra tính hợp lệ xong
- T chưa hoàn tất ghi thì U kiểm tra hợp lệ
- $WS(U) \cap WS(T) = \{X\}$
- U có thể ghi X trước T

→ Rollback U

Xác nhận hợp lệ - Validation

Vấn đề 2



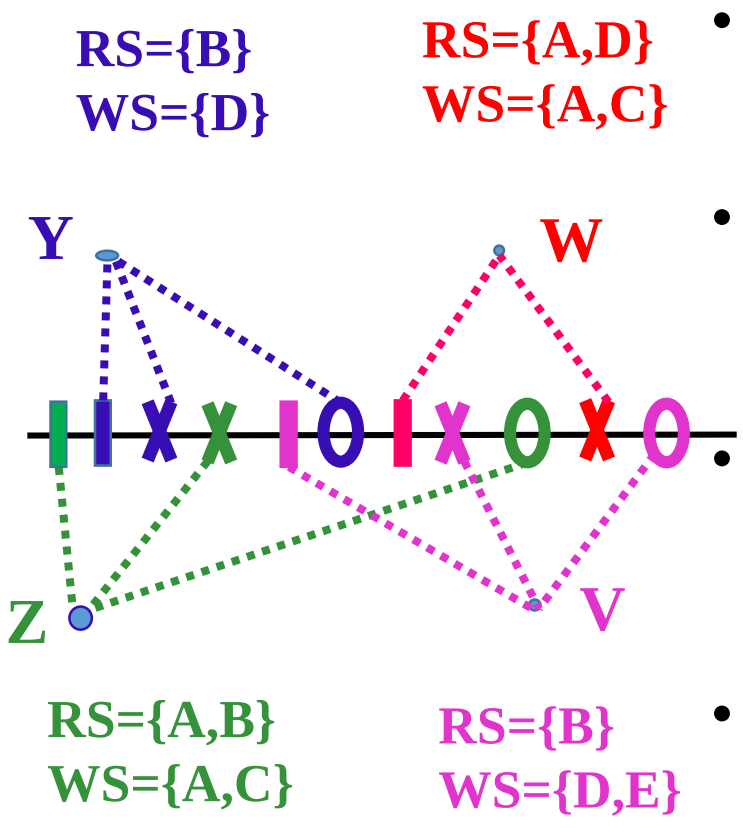
T phải hoàn tất trước khi U kiểm tra tính hợp lệ xong

Xác nhận hợp lệ - Validation

Quy tắc

- Khi đang kiểm tra tính hợp lệ của giao tác U
  - (1) Nếu T đã hợp lệ nhưng chưa hoàn tất mà U bắt đầu
    - Kiểm tra $RS(U) \cap WS(T) = \emptyset$
  - (2) Nếu T đã hợp lệ nhưng chưa hoàn tất trước khi U hợp lệ
    - Kiểm tra $WS(U) \cap WS(T) = \emptyset$

Khi đang kiểm tra tính hợp lệ của giao tác U
(1) Nếu T đã hợp lệ nhưng chưa hoàn tất mà U bắt đầu: Kiểm tra RS(U) ∩ WS(T) = ∅
(2) Nếu T đã hợp lệ nhưng chưa hoàn tất trước khi U hợp lệ: Kiểm tra WS(U) ∩ WS(T) = ∅



- Khi Y kiểm tra tính hợp lệ:
  - Không có giao tác nào kiểm tra tính hợp lệ xong trước đó

  → Y kiểm tra hợp lệ thành công và ghi D
- Khi Z kiểm tra tính hợp lệ:
  - Y đã kiểm tra hợp lệ xong nhưng chưa hoàn tất giao tác nên kiểm tra WS(Y) và [RS(Z), WS(Z)]
  - →Z kiểm tra hợp lệ thành công và ghi A,C
- Khi V kiểm tra tính hợp lệ
  - Vì V bắt đầu trước khi U hoàn tất nên kiểm tra RS(V) ∩ WS(Y)
  - V thì đã hợp lệ nhưng chưa hoàn tất, kiểm tra WS(Z) ∩ [RS(V), WS(V)]
  - →V hợp lệ và ghi D,E
- Khi W kiểm tra tính hợp lệ
  - Y hoàn thành trước khi W bắt đầu → không kiểm tra
  - Vì W bắt đầu trước khi Z hoành thành, kiểm tra RS(W) ∩ WS(Z)→A
  - V hợp lệ nhưng chưa hoàn tất, kiểm tra WS(V) ∩ [RS(W), WS(W)] →D
  - →W chưa hợp lệ và phải rollback

Nhận xét

- So sánh ba phương pháp
  - Khóa (locking)
  - Nhãn thời gian (timestamps)
  - Xác nhận hợp lệ (validation)
- Dựa vào
  - Lưu trữ
    - Số lượng đơn vị dữ liệu
  - Khả năng thực hiện
    - Các giao tác ảnh hưởng với nhau như thế nào, nhiều hay ít

Bài tập

a. $R_1(A,B); R_2(B,C); V_1; R_3(C,D); V_3; W_1(A); V_2; W_2(A); W_3(B);$

b. $R_1(A,B); R_2(B,C); V_1; R_3(C,D); V_3; W_1(A); V_2; W_2(A); W_3(D);$

c. $R_1(A,B); R_2(B,C); V_1; R_3(C,D); V_3; W_1(C); V_2; W_2(A); W_3(D);$

d. $R_1(A,B); R_2(B,C); R_3(C); V_1; V_2; V_3; W_1(A); W_2(B); W_3(C);$

e. $R_1(A,B); R_2(B,C); R_3(C); V_1; V_2; V_3; W_1(C); W_2(B); W_3(A);$

f. $R_1(A,B); R_2(B,C); R_3(C); V_1; V_2; V_3; W_1(A); W_2(C); W_3(B);$

- Trong đó:
  - $R_i(X)$: giao tác T_i bắt đầu, tập dữ liệu đọc là X.
  - V_i: T_i kiểm tra hợp lệ.
  - $W_i(X)$: giao tác T_i kết thúc, ghi đơn vị dữ liệu X.
- Yêu cầu: áp dụng kỹ thuật xác nhận hợp lệ cho tập các giao tác trên.

Nhận xét

	Khóa	Nhãn thời gian	Xác nhận hợp lệ
Delay Rollback	Trì hoãn các giao tác, ít rollback	Không trì hoãn các giao tác, nhưng gây rollback	
		Xử lý rollback nhanh chóng	Xử lý rollback chậm
Storage	Phụ thuộc số lượng đơn vị dữ liệu bị khóa	Phụ thuộc vào nhãn thời gian đọc và ghi của mỗi đơn vị dữ liệu	Phụ thuộc vào tập WS và RS của các giao tác
		Sử dụng nhiều bộ nhớ	