Chương II. QUẢN LÝ GIAO TÁC

2.1 Giới thiệu

Hai yêu cầu cơ bản của ứng dụng khai thác CSDL trong thực tế:

– Cho phép **nhiều người dùng đồng thời khai thác** CSDL nhưng phải **giải quyết được các tranh chấp.**

– Sự cố kỹ thuật có thể luôn luôn xảy ra nhưng phải **giải quyết được vấn đề về nhất quán dữ liệu.**

Một số tình huống

Hệ thống đặt vé máy bay: – "Khi hành khách mua vé" **– "Khi hai/ nhiều hành khách cùng đặt một ghế trống"** → Lỗi: Có thể có **nhiều hành khách đều đặt được dù chỉ còn 1 ghế** → **Phải giải quyết được tranh chấp để đảm bảo được nhất quán dữ liệu.**	GHẾ (**Mã ghế**, Mã CB, Trạng thái) CHUYẾN BAY(**Mã CB**, Ngày giờ, Số ghế còn) **Thao tác của người dùng:** 1. Tìm thấy một ghế trống 2. Đặt ghế
Hệ thống ngân hàng: – "Khi chuyển tiền từ tài khoản A sang tài khoản B" → Lỗi: Có thể đã rút tiền từ A nhưng chưa cập nhật số dư của B. → Phải đảm bảo được nhất quán dữ liệu khi có sự cố – "Khi rút tiền của một tài khoản"	TÀI KHOẢN(**Mã TK**, Số dư) GIAO DỊCH(**Mã GD**, Loại, Số tiền) 1. update TAIKHOAN set SoDu=SoDu-50 where MATK=A 2. update TAIKHOAN set SoDu=SoDu+50 where MATK=B
– "Nhiều người cùng rút tiền trên một tài khoản" → **Lỗi:** Có thể rút **nhiều hơn số tiền thực** có. → Phải giải quyết được tranh chấp để đảm bảo được nhất quán dữ liệu.	1. Đọc số dư của tài khoản A vào X 2. Cập nhật số dư mới của tài khoản A bằng X – Số tiền
Hệ thống quản lý học sinh: – Thêm một học sinh mới → **Lỗi:** Có thể xảy ra trường hợp **học sinh đã được thêm** nhưng **sĩ số không được cập nhật.** → Phải đảm bảo được nhất quán dữ liệu khi có sự cố.	Lớp học(**Mã lớp**, Tên, Sĩ số) Học sinh (**Mã HS**, Họ tên, Mã lớp) 1. Thêm vào một học sinh của lớp 2. Cập nhật sĩ số lớp tăng lên 1

Nhận xét: Thường xuyên xảy ra vấn đề **nhất quán dữ liệu** nếu **một xử lý gặp sự cố** hoặc khi **các xử lý được gọi *truy xuất đồng thời*.**

Cần 1 khái niệm biểu diễn một đơn vị xử lý với các tính chất: **Nguyên tố – Cô lập – Nhất quán – Bền vững** → **Giao tác:** Là một khái niệm nền tảng của **điều khiển truy xuất đồng thời và khôi phục khi có sự cố.**

2.2 Giao tác

2.2.1 Định nghĩa

- Giao tác (Transaction) là **một đơn vị xử lý nguyên tố** gồm **một chuỗi các hành động đọc / ghi trên các đối tượng CSDL**

– Nguyên tố: Không thể phân chia được nữa → Giao tác là một đơn vị công việc **không thể chia nhỏ được nữa.**

– Các hành động ghi: INSERT, UPDATE, DELETE**; Hành động đọc:** SELECT**.**

Các hành động trong một giao tác hoặc là thực hiện được tất cả hoặc là không thực hiện được bất cứ hành động nào.

- Trong kiến trúc hệ quản trị CSDL: **Bộ phận Điều khiển đồng thời đóng vai trò quản lý giao tác**

2.2.2 Tính chất ACID của giao tác

- **Tính nguyên tố (Atomicity):** Hoặc là toàn bộ hoạt động được phản ánh đúng đắn (thực thi) trong CSDL, hoặc không có hoạt động nào cả (không có hoạt động nào được thực thi)

  Nếu xảy ra lỗi trong quá trình thực thi, tất cả các thao tác trong giao tác sẽ được **hoàn tác về ban đầu (rollback)**.
- **Tính nhất quán (Consistency):** Khi một giao tác kết thúc (thành công hay thất bại), CSDL phải ở trạng thái nhất quán (Đảm bảo mọi RBTV). Một giao tác đưa CSDL từ trạng thái nhất quán này sang trạng thái nhất quán khác.

  Ví dụ: Nếu một ràng buộc rằng Balance >= 0 được thiết lập cho tài khoản ngân hàng, giao tác phải đảm bảo rằng không có tài khoản nào có số dư âm sau khi giao tác hoàn thành.
- **Cô lập (Isolation):** Một giao tác khi thực hiện sẽ không bị ảnh hưởng bởi các giao tác khác thực hiện **đồng thời** với nó.
- **Bền vững (Durability):** Mọi thay đổi trên CSDL được ghi nhận bền vững vào thiết bị lưu trữ dù có sự cố có thể xảy ra. SQL Server đảm bảo tính bền vững thông qua **transaction logs**.

  Ví dụ: Khi bạn chạy một giao tác và COMMIT, thay đổi của nó sẽ được ghi vào log và lưu trên đĩa. Ngay cả khi hệ thống bị tắt đột ngột, cơ sở dữ liệu vẫn có thể khôi phục từ log.

Ví dụ:

Chuyển khoản tiền từ tài khoản A sang tài khoản B	**Atomicity:** Hoặc cả 2 bước đều thực hiện hoặc không bước nào được thực hiện. Nếu có sự cố thì HQT CSDL có cơ chế khôi phục lại dữ liệu như lúc ban đầu. **Consitency:** Với giao tác chuyển tiền, tổng số dư của A và B luôn luôn không đổi.
Giao tác Chuyển khoản 1. update TAIKHOAN set SoDu=SoDu-50 where MATK=A 2. update TAIKHOAN set SoDu=SoDu+50 where MATK=B **Cuối giao tác**	

Thêm học sinh mới vào một lớp

Giao tác Thêm học sinh mới

1. Thêm một học sinh vào bảng học sinh
2. Cập nhật sĩ số của lớp tăng lên 1

Cuối giao tác

Atomicity: Hoặc cả 2 bước đều thực hiện hoặc không bước nào được thực hiện.

Nếu có sự cố thì HQT CSDL có cơ chế khôi phục lại dữ liệu như lúc ban đầu.

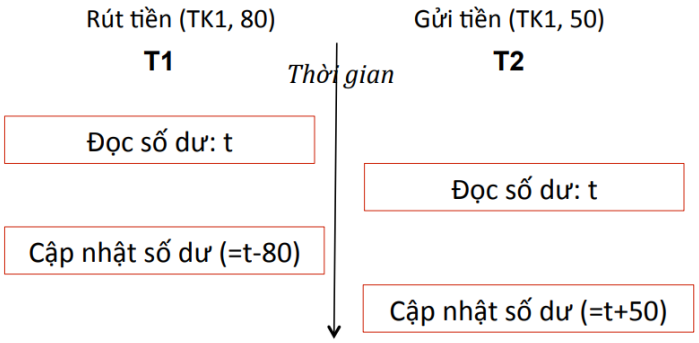
Consistency: Sĩ số của lớp phải luôn bằng số học sinh thực sự và **không quá 3.**

Lớp học(Mã lớp, Tên, Sĩ số)

Mã lớp	Tên	Sĩ số
1	10A	3

Học sinh (*Mã HS*, Họ tên, Mã lớp)

Mã HS	Họ tên	Mã lớp
1	An	1
2	Thảo	1
3	Bình	1



Tài khoản (*Mã TK*, Số dư)

Mã TK	Số dư
1	100
2	500
3	200

2 hành động xảy ra trên cùng 1 thời gian (đồng thời), nhưng thực hiện 2 tác vụ hoàn toàn độc lập.

Isolation: Tính chất cô lập đảm bảo **mặc dù các giao tác có thể đan xen nhau nhưng kết quả của chúng tương tự với một kết quả tuần tự nào đó**

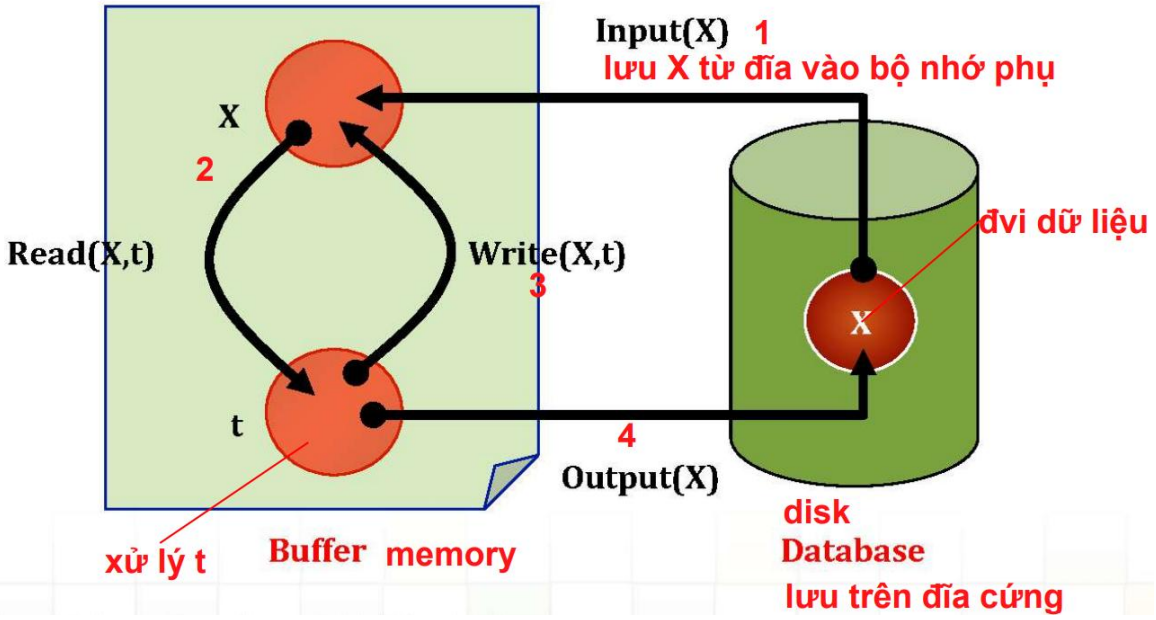
→ Các giao tác không bị ảnh hưởng bởi các giao tác khác khi thực thi.

2.2.3 Đơn vị dữ liệu

- *Đối tượng CSDL mà giao tác thực hiện các xử lý đọc/ghi* còn được gọi là **đơn vị dữ liệu.**
- Một đơn vị dữ liệu (element) có thể là các thành phần:
  - Quan hệ (Relations)
  - Khối dữ liệu trên đĩa (Blocks)
  - Bộ (Tuples)

  (Bảng > Blocks > Bộ)
- **Một CSDL bao gồm nhiều đơn vị dữ liệu.**

2.2.4 Các thao tác của giao tác



- **Read (A, t):** Đọc đơn vị dữ liệu A vào t (Đọc đơn vị dữ liệu A, lưu vào biến cục bộ t)
- **Write (A, t):** Ghi t vào đơn vị dữ liệu A (Lấy dữ liệu từ biến t, ghi vào đơn vị dữ liệu A)

Ví dụ:

Giả sử có 2 đơn vị dữ liệu A và B với ràng buộc **A = B** (nếu có một trạng thái nào đó mà A ≠ B thì sẽ mất tính nhất quán)

Giao tác T thực hiện 2 bước: **A = A * 2 ; B = B * 2**

Biểu diễn T: ***Cách 1:*** ➔ ***Cách 2:*** T: Read(A, t); t =t*2; Write(A, t); Read(B, t); t =t*2; Write(B, t)	**T** *Read(A, t);* t =t*2; *Write(A, t)* *Read(B, t);* t =t*2; *Write(B, t)*

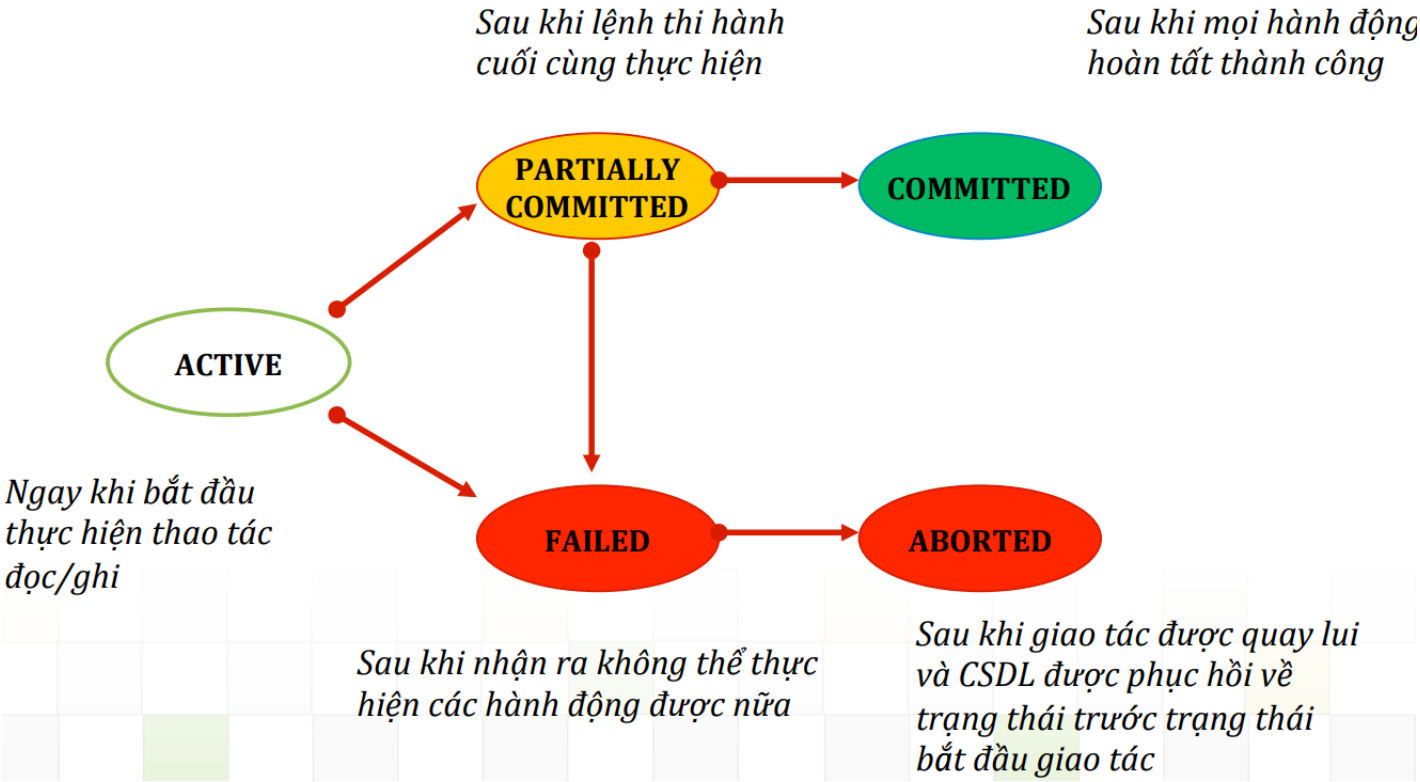
Ví dụ a/

Hành động	t	Mem A	Mem B	Disk A	Disk B
Input (A)		8		8	8
Read (A, t)	8	8		8	8
t := t * 2	16	8		8	8
Write (A, t)	16	16		8	8
Input (B)	16	16	8	8	8
Read (B, t)	8	16	8	8	8
t := t * 2	16	16	8	8	8
Write (B, t)	16	16	16	8	8
Output (A)	16	16	16	16	8
Output (B)	16	16	16	16	16

Ví dụ b/

	T	Mem A	Mem B	Disk A	Disk B
Input (A)		6		6	6
Read (A, t)	6	6		6	6
t = t*3	18	6		6	6
Write (A, t)	18	18		6	6
Input(B)		18	6	6	6
Read (B, t)	6	18	6	6	6
t = t*3	18	18	6	6	6
Write (B, t)	18	18	18	6	6
Output (A)	18	18	18	18	6
Output(B)	18	18	18	18	18

2.2.5 Các trạng thái của giao tác



2.2.6 Khai báo giao tác trong T-SQL

- **BEGIN TRANSACTION** Bắt đầu giao tác
- **COMMIT TRANSACTION** Kết thúc giao tác thành công
- **ROLLBACK TRANSACTION** Kết thúc giao tác không thành công. CSDL được đưa về tình trang **trước khi thực hiện giao tác.**

Ví dụ: giao tác Chuyển khoản

```
create proc sp_ChuyenKhoan
@matk_chuyen char(10), @matk_nhan char(10), @sotien float
as
        declare @sodu float
        begin transaction
        select @sodu=SoDu from TAIKHOAN
        where MATK=@matk_chuyen
        if (@sodu < @sotien)
        begin
                rollback tran
                return
        end
        update TAIKHOAN set SoDu=SoDu-500000 where MATK=@matk_chuyen
        if (@@error <> 0) --có lỗi
        begin
                rollback tran
                return
        end
        update TAIKHOAN set SoDu=SoDu+500000 where MATK=@matk_nhan
        if (@@error <> 0) --nếu có lỗi từ hệ thống
        begin
                rollback tran
                return
        end
commit transaction
```

2.3 Lịch giao tác

2.3.1 Các cách thực hiện của các giao tác

- **Thực hiện tuần tự:** Các thao tác khi thực hiện mà **không giao nhau về mặt thời gian**.
  - **Ưu:** Nếu thao tác **đúng đắn** thì **luôn luôn đảm bảo nhất quán dữ liệu.**
  - **Khuyết: Không tối ưu** về việc sử dụng tài nguyên và tốc độ.
- **Thực hiện đồng thời:** ***Các lệnh của các giao tác khác nhau*** **xen kẽ nhau trên trục thời gian.**
  - **Khuyết:** Gây ra nhiều phức tạp về nhất quán dữ liệu
  - **Ưu:**
    - **Tận dụng tài nguyên và thông lượng (throughput).** Ví dụ: Trong khi một giao tác đang thực hiện việc đọc / ghi trên đĩa, một giao tác khác đang xử lý tính toán trên CPU.
    - **Giảm thời gian chờ.** Ví dụ: Chia sẽ chu kỳ CPU và truy cập đĩa để làm giảm sự trì hoãn trong các giao tác thực thi.

2.3.2 Lịch thao tác là gì?

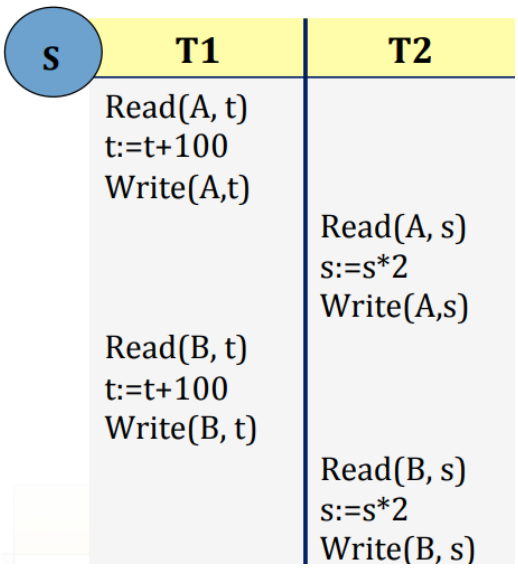
- **Đặt vấn đề:** *Để xử lý một vấn đề gồm có nhiều hành động cần thực hiện, các hành động cần được thực hiện đồng thời, vì thế DBMS cần một cơ chế để quản lý – lập lịch (schedule).*
- **Định nghĩa:** **Một lịch** thao tác S được *lập từ n giao tác* $T_1, T_2, \ldots, T_n$ được **xử lý đồng thời** là *một* ***thứ tự*** *thực hiện* ***xen kẽ các hành động của n giao tác*** này.
- *Thứ tự xuất hiện của các thao tác trong lịch phải giống với thứ tự xuất hiện của chúng trong giao tác.*

- **Bộ lập lịch (Scheduler):** Là một thành phần của DBMS có nhiệm vụ *lập một lịch để thực hiện n giao tác xử lý đồng thời.*
- **Các loại lịch thao tác:**
  - **Lịch tuần tự (Serial)**
  - **Lịch khả tuần tự (Serializable):** Conflict – Serializability ; View – Serializability

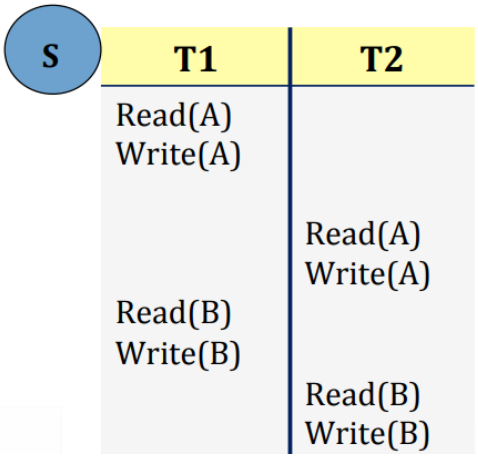
Feature	Serial Schedule	Serializable Schedule
Execution	Transactions execute one after the other.	Transactions can execute **concurrently**.
Concurrency	No concurrency *(đồng thời).*	**Allows** concurrency.
Performance	Lower performance due to no concurrency.	**Higher** performance due to concurrency.
Complexity	Simpler to implement.	**More complex** to implement due to concurrency control.
Final Outcome	Depends on the order of execution.	**Equivalent** to some **serial execution.**
Data Consistency	Guarantees *(bảo đảm)* data consistency.	Guarantees data consistency.

2.3.2.1 Biểu diễn lịch thao tác

Cách 1:



Cách 2: (chỉ quan tâm đọc/ ghi)

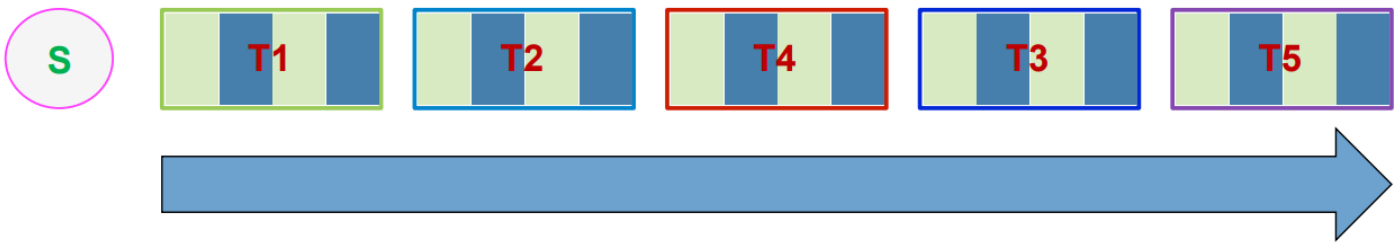


Cách 3: S: R1(A); W1(A); R2(A); W2(A); R1(B); W1(B); R2(B); W2 (B)

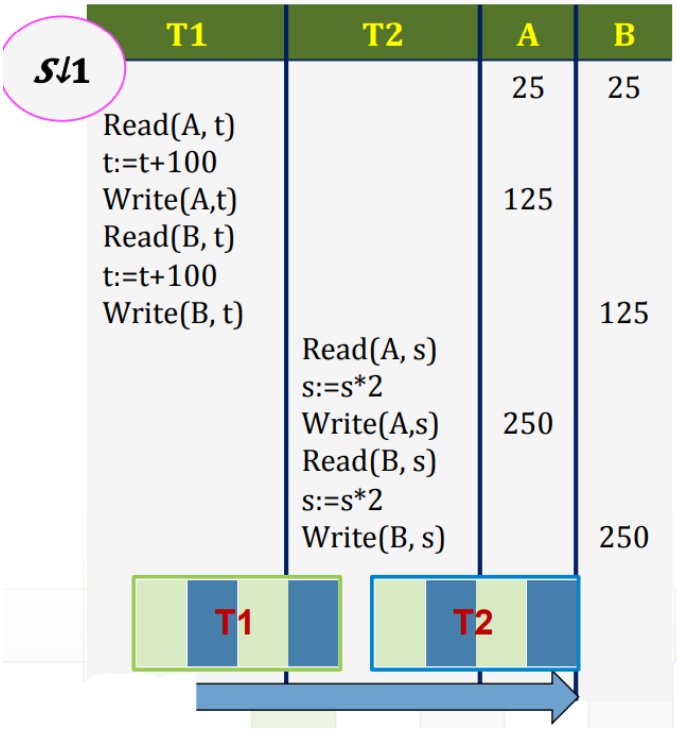
→ Thông thường người ta biểu diễn cách 3, ta cần chuyển về cách 2 cho dễ quan sát.

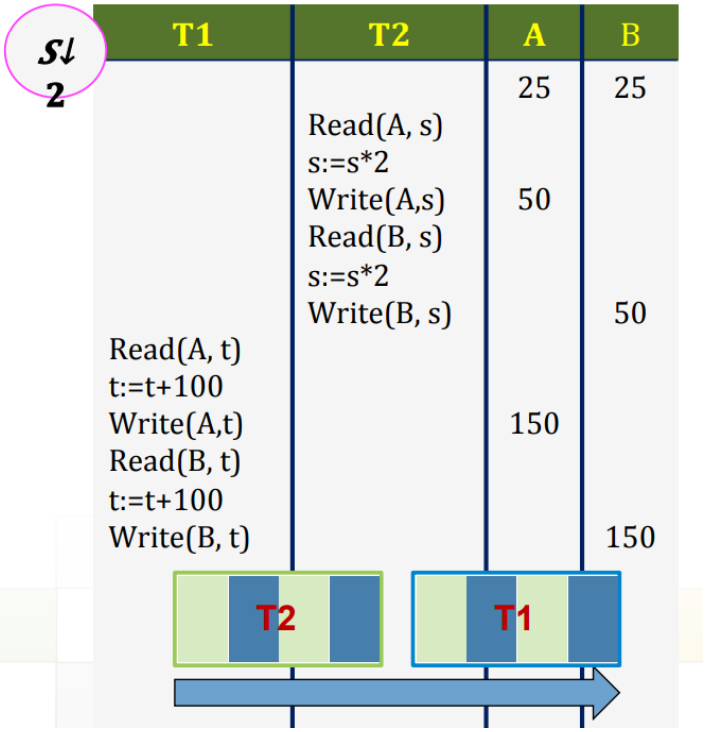
2.3.2.2 Lịch tuần tự (serial schedule)

- Một lịch S được gọi là **tuần tự** nếu **các hành động của các giao tác T_i được thực hiện liên tiếp nhau**, không có sự giao nhau về mặt thời gian.
- Lịch tuần tự **luôn luôn đảm bảo được tính nhất quán** của CSDL



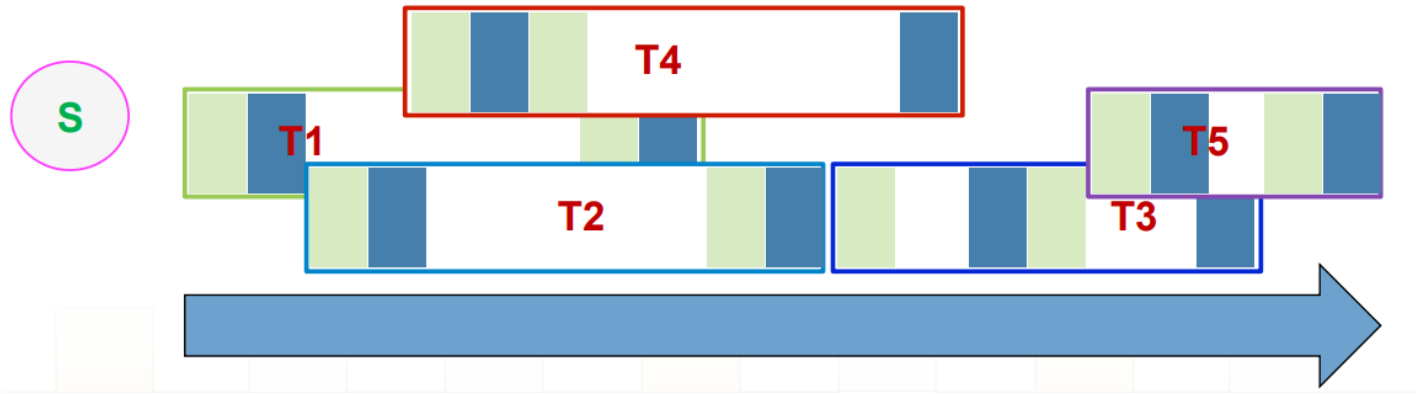
Ví dụ:





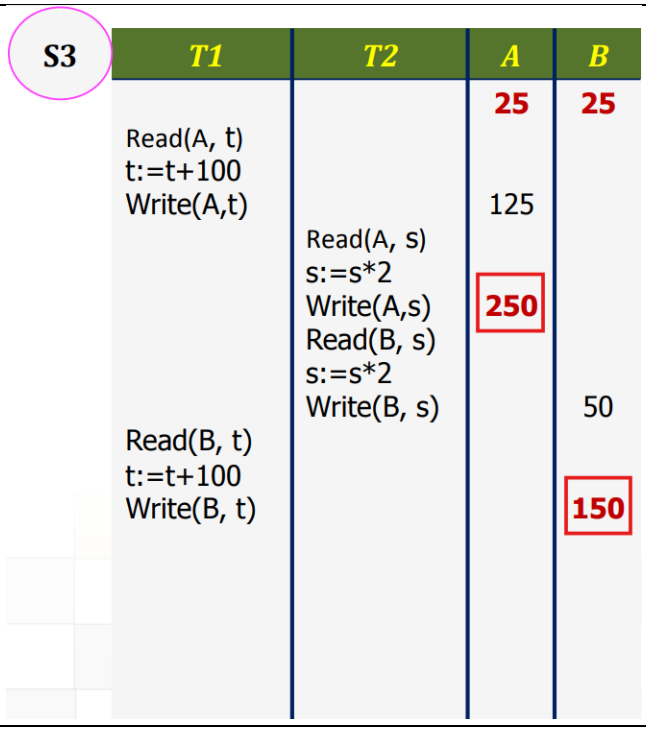
2.3.2.3 Lịch xử lý đồng thời (Non-serial Schedule) (Simultaneous Schedule)

- Lịch xử lý đồng thời là lịch mà **các giao tác** trong đó **giao nhau về mặt thời gian.**
- Luôn luôn tiềm ẩn khả năng làm cho CSDL **mất tính nhất quán**



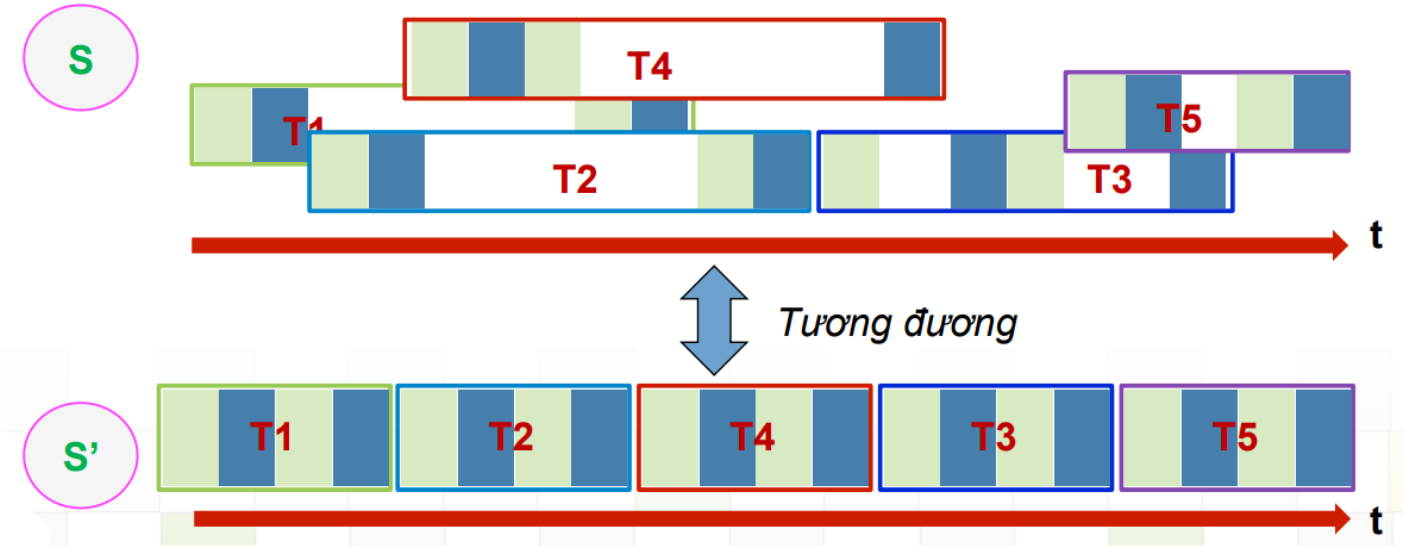
Ví dụ:

- S_3 là một lịch xử lý đồng thời vì **các giao tác giao thoa với nhau**
- Lịch xử lý đồng thời S_3 gây ra sự **mất nhất quán dữ liệu**
  - Trước S khi thực hiện: **A=B**
  - Sau khi S kết thúc: **A ≠ B**



2.3.2.4 Lịch khả tuần tự (Serializable Schedule)

- Một lịch S được lập ra từ n giao tác T_1, T_2, …, T_n ***xử lý đồng thời*** được gọi là lịch khả tuần tự **nếu** nó **cho cùng kết quả với một lịch tuần tự nào đó** được ***lập ra từ n giao tác này.***
- Lịch khả tuần tự cũng không gây nên tình trạng mất nhất quán dữ liệu



Ví dụ:

S_4

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

- Trước S_4 khi thực hiện: **A = B = c, với c là hằng số** (trong ví dụ này c = 25).
- Sau khi S_4 kết thúc: **A=2*(c+100); B=2*(c+100)**
- Trạng thái CSDL nhất quán **(A = B).**
- **Kết quả của lịch S_4 <T1, T2> tương tự với kết quả của lịch tuần tự S_1<T1, T2>**
- Vậy S_4 là khả tuần tự.

S_5

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

- Trước S_5 khi thực hiện: A = B = c, với c là hằng số
- Sau khi S_5 kết thúc: **A = 2*(c+100); B = 2*c + 100**
- **Trạng thái CSDL không nhất quán (A ≠ B)**
- S_5 **không khả tuần tự**

S_{5b}

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

- Trước S_{5b} khi thực hiện: A = B = c, với c là hằng số
- Sau khi S_{5b} kết thúc: A = 1*(c+100); B = 1*c + 100
- **Trạng thái CSDL vẫn nhất quán**

- Để xem xét tính **mất nhất quán** một cách kỹ lưỡng, nếu **phải hiểu rõ ngữ nghĩa** của từng giao tác → **không khả thi**
- **Thực tế chỉ xem xét các lệnh của giao tác là đọc hay ghi**

Có 2 loại lịch khả tuần tự:

- **Conflict Serializable (Khả tuần tự xung đột):** Dựa trên ý tưởng **hoán vị các hành động không xung đột** để **chuyển một lịch đồng thời S về một lịch tuần tự S'**. Nếu có **một cách biến đổi như vậy** thì **S là một lịch conflict serializable.**
- **View Serializable (Khả tuần tự view):** Dựa trên ý tưởng **lịch đồng thời S và lịch tuần tự S' đọc và ghi** những **giá trị dữ liệu giống nhau.** Nếu có **một lịch S' như vậy** thì **S là một lịch view serializable.** ***(S' có sẵn, và so sánh với S).***

→ **Ta có: Conflict Serializable ⊂ View Serializable ⊂ Serializable**

2.3.2.4.1 Conflict Serializability

Ý tưởng: Xét **2 hành động liên tiếp nhau** của **2 giao tác khác nhau** trong một lịch thao tác, ***khi 2 hành động đó được đảo thứ tự*** có thể dẫn đến 1 trong 2 hệ quả:

– **Hoạt động của cả hai giao tác** chứa 2 hành động ấy **không bị ảnh hưởng** gì → **2 hành động đó không xung đột với nhau.**

– **Hoạt động của ít nhất một trong 2 giao tác** chứa 2 hành động ấy **bị ảnh hưởng** → **2 hành động xung đột.**

T	T'
Hành động 1	
Hành động 2	
	Hành động 1'
	Hành động 2'
Hành động 3	
Hành động 4	
	Hành động 3'
	Hành động 4'

Cho lịch S có **2 giao tác T_i và T_j,** xét các trường hợp:

– $r_i(X)$; r_j (Y)

(Đọc trên đơn vị dữ liệu X của giao tác thứ i, đọc trên đơn vị dữ liệu Y của giao tác thứ j)

- **Không bao giờ có xung đột, ngay cả khi X = Y**
- Cả 2 thao tác *không làm thay đổi giá trị của X và Y*

– $r_i(X)$; w_j (Y)

- **Không xung đột khi X ≠ Y**
  - – T_j không thay đổi dữ liệu đọc của T_i
  - – T_i không sử dụng dữ liệu ghi của T_j
- **Xung đột khi X = Y**

– wi(X) ; rj (Y)

- **Không xung đột khi X ≠ Y, Xung đột khi X = Y**

– wi(X) ; wj (Y)

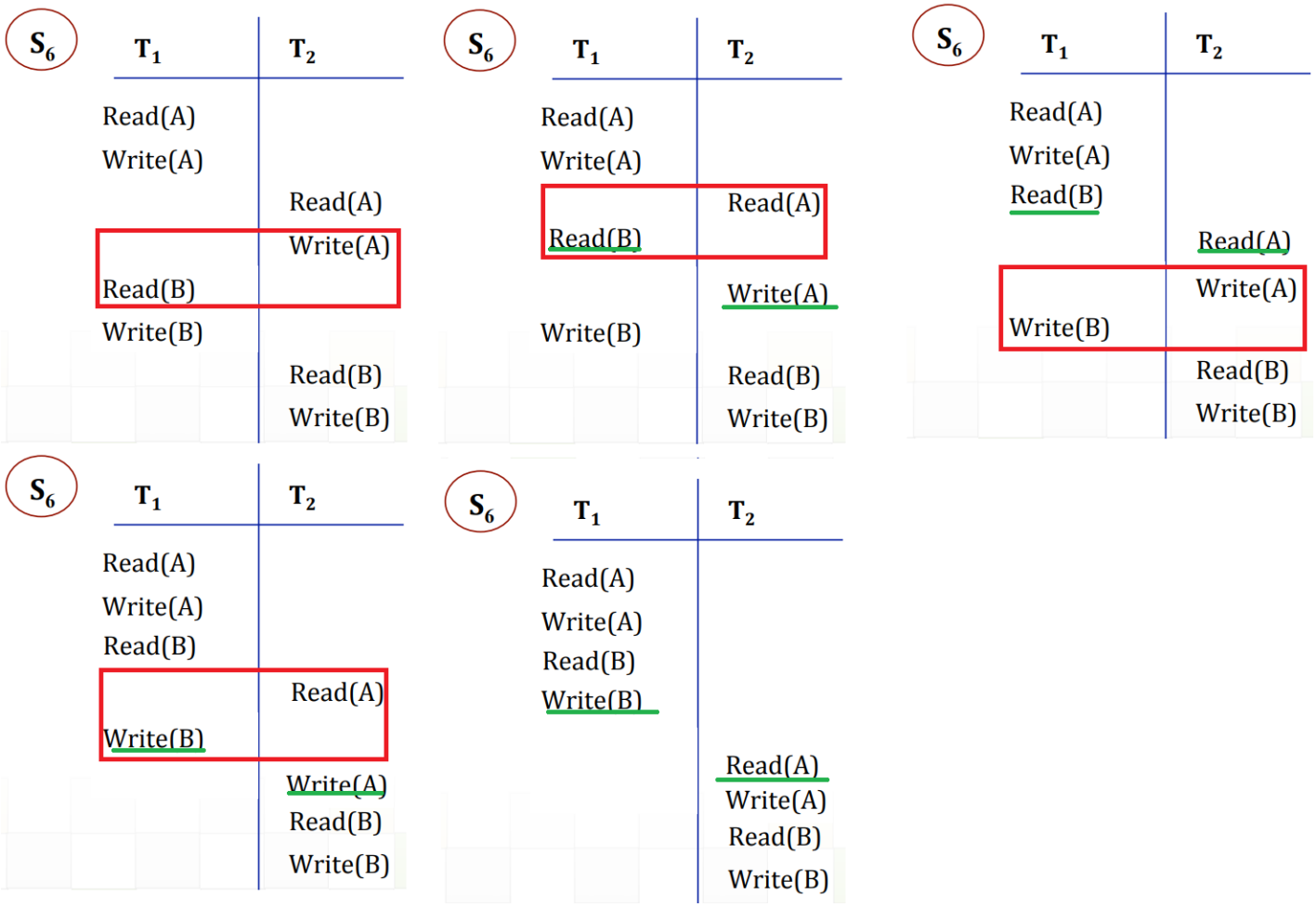
- **Không xung đột khi X ≠ Y, Xung đột khi X = Y**

→ **Tóm lại, hai hành động liên tiếp nhau xung đột nếu chúng:**

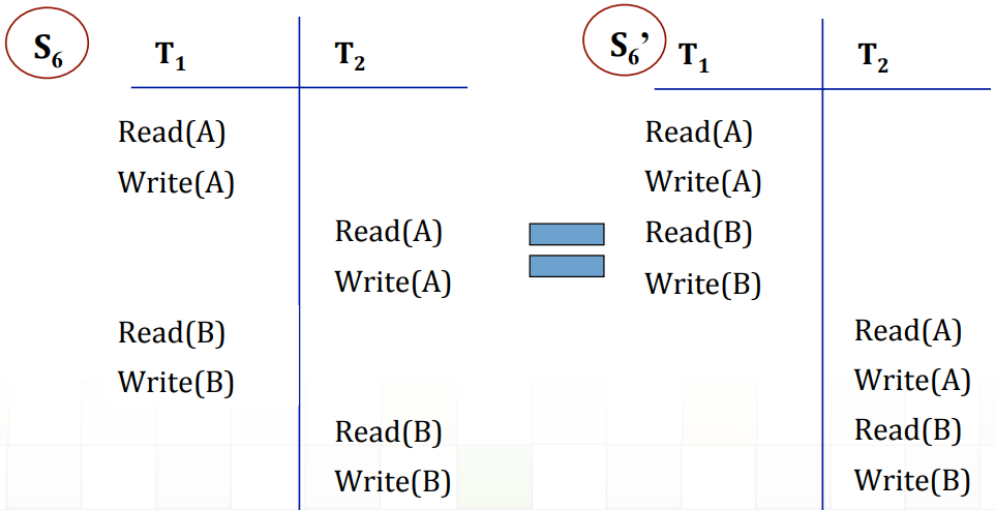
- Thuộc **2 giao tác khác nhau**
- Truy xuất đến **1 đơn vị dữ liệu**
- Trong chúng có **ít nhất một hành động ghi** (write)

→ Hai hành động **xung đột** thì **không thể nào đảo thứ tự của chúng** trong một lịch thao tác.

Ví dụ:



Vậy:



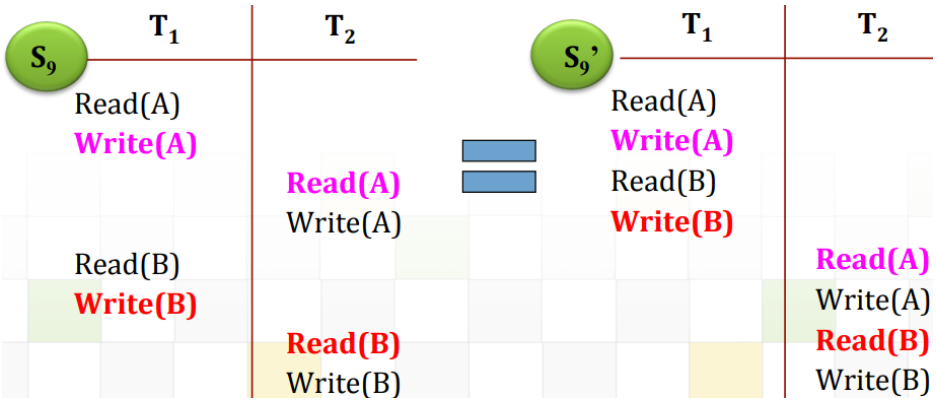
https://www.youtube.com/watch?v=GnU5tA3MReg

– **S và S'** là những lịch thao tác **conflict equivalent** (tương đương xung đột) **nếu S có thể chuyển được thành S' thông qua một chuỗi các hoán vị những thao tác không xung đột.**
– Một lịch thao tác **S là conflict serializable** nếu **S là conflict equivalent** với một lịch thao tác **tuần tự S'** nào đó.

– S là **conflict serializable** thì **S khả tuần tự.**
– S là khả tuần tự thì không chắc S conflict serializable.

2.3.2.4.2 Kiểm tra Conflict Serializability

Cho lịch S_9: S_9 có conflict serializability hay không?



Ý tưởng: Các hành động **xung đột** trong lịch S được thực hiện theo thứ tự nào thì **các giao tác thực hiện chúng trong S'** (kết quả sau hoán vị) **cũng theo thứ tự đó**.

Cho lịch S có **2 giao tác T1 và T2:**

– T1 thực hiện hành động A1
– T2 thực hiện hành động A2
– Ta nói **T1 thực hiện trước hành động T2** trong S, ký hiệu **T1 <S T2**, **khi:**
  • **A1 được thực hiện trước A2** trong S, **A1 không nhất thiết phải liên tiếp A2**
  • **A1 và A2 là 2 hành động xung đột** *(A1 và A2* ***cùng thao tác lên 1 đơn vị dữ liệu*** *và có* ***ít nhất 1 hành động là ghi*** *trong A1 và A2)*

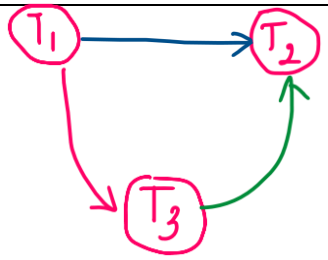
Phương pháp Precedence Graph (Đồ thi ưu tiên):

- Cho lịch S bao gồm các thao tác $T_1, T_2, \ldots, T_n$
- Đồ thị trình tự của S **(Precedence graph)** của S ký hiệu là **P(S)** có:
  – **Đỉnh** là các giao tác T_i **(vẽ trước đầy đủ các đỉnh)**
  – **Cung** đi từ T_i đến T_j nếu $T_i <_S T_j$
  Nếu có nhiều hơn 1 cặp xung đột (nhiều hơn 1 cung giữa 2 đỉnh), ta chỉ vẽ một cung để đỡ rối.
- **S Conflict Serializable khi và chỉ khi P(S) không có chu trình,** điều này có nghĩa là chúng ta có thể xây dựng một lịch trình tuần tự S' tương đương với lịch xung đột S. Lịch tuần tự S' có thể được tìm thấy **bằng cách sắp xếp tô pô của đồ thị P(S). Lịch trình như vậy có thể nhiều hơn 1.**
- Với 2 lịch S và S' được lập từ cùng các giao tác, **S và S' conflict equivalent khi và chỉ khi P(S) = P(S')**
- Thứ tự hình học các đỉnh là **thứ tự của các giao tác** trong lịch tuần tự tương đương với S.

Tham khảo: https://www.youtube.com/watch?v=U3SHusK80q0

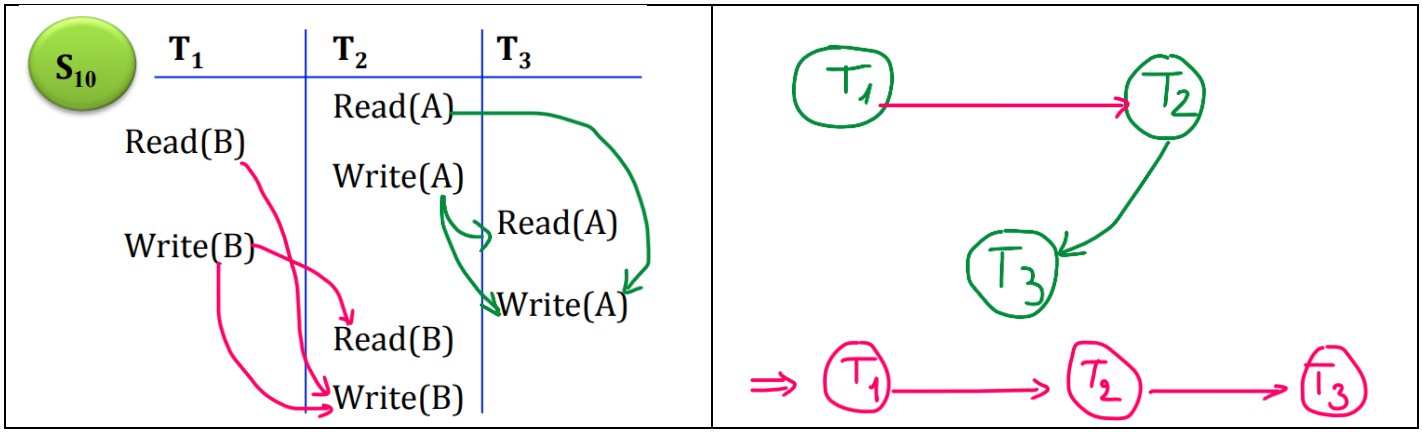
Ví dụ 1: S_1: r1(x);r3(y);w1(x);w2(y);r3(x);w2(x)

T1	T2	T3
R(X)		
		R(Y)
W(X)		
	W(Y)	
		R(X)
	W(X)	

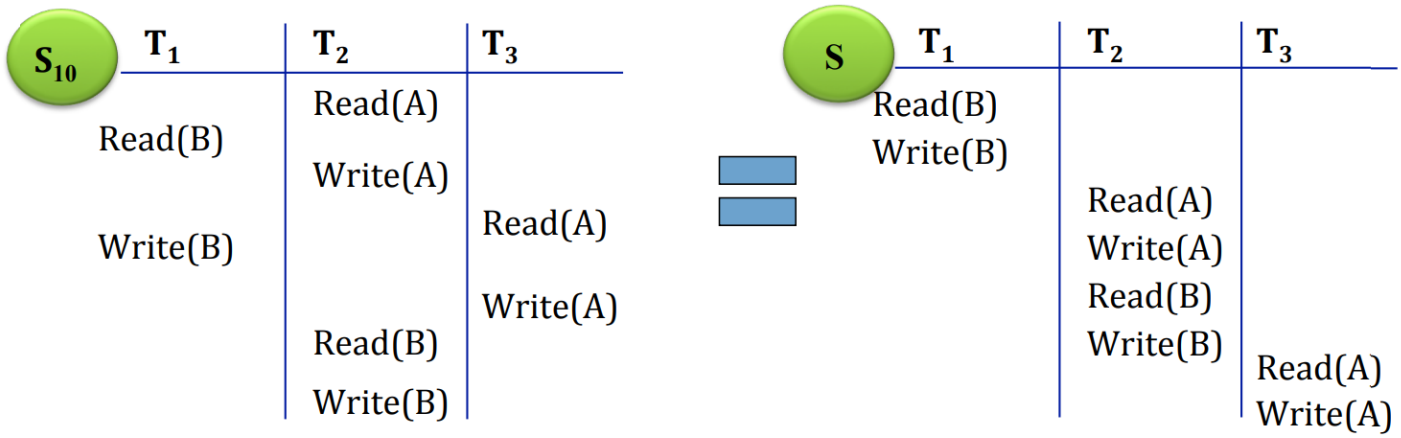


Lịch tuần tự là 1→ 3 → 2

Ví dụ 2: S_{10} có Conflict Serializability hay không?

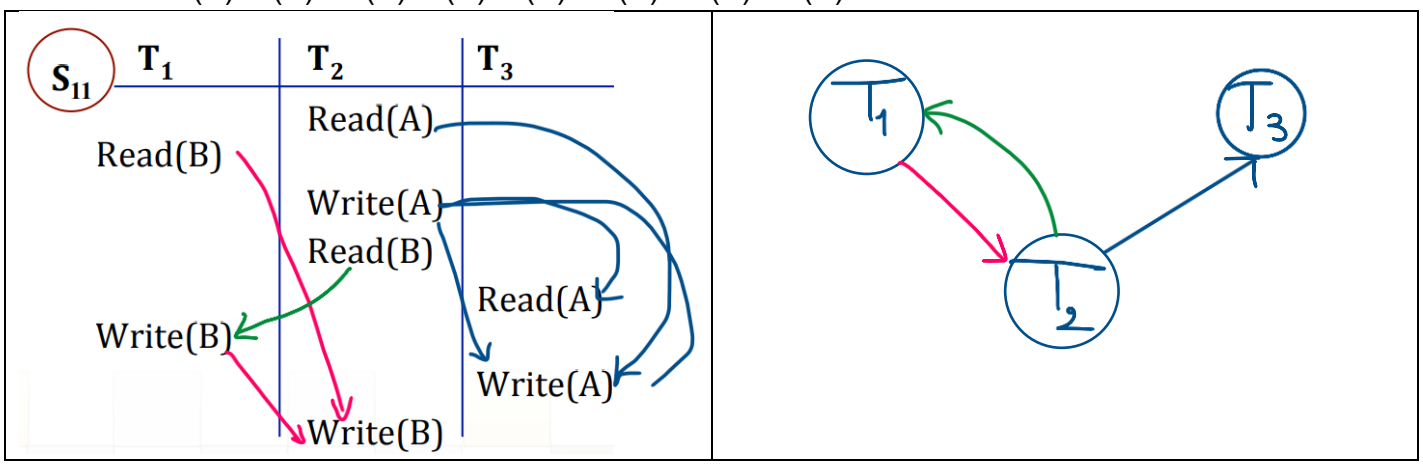


P(S10) không có chu trình → S10 conflict-serializable theo thứ tự T1 → T2 → T3.



Ví dụ 3: S_{11} có Conflict Serializability hay không?

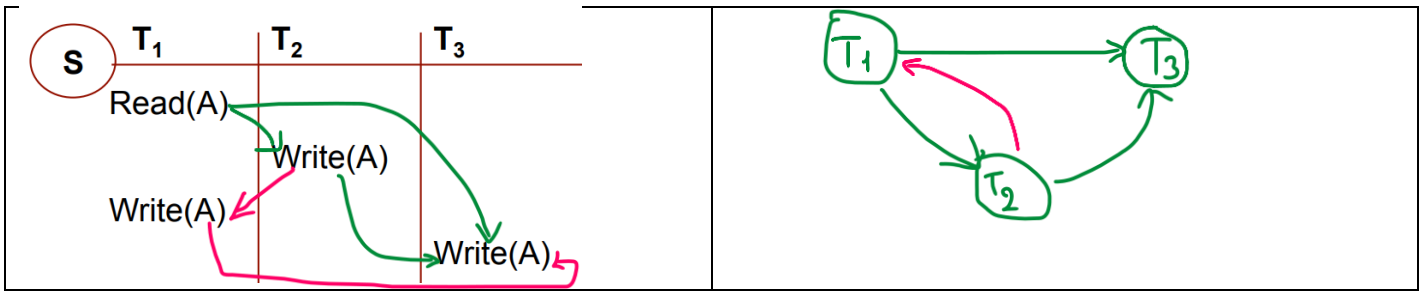
S_{11}: r2(A) r1(B) w2(A) r2(B) r3(A) w1(B) w3(A) w2(B)



P(S_{11}) **có chu trình → S_{11} không conflict-serializable.**

2.3.2.4.3 View Serializability

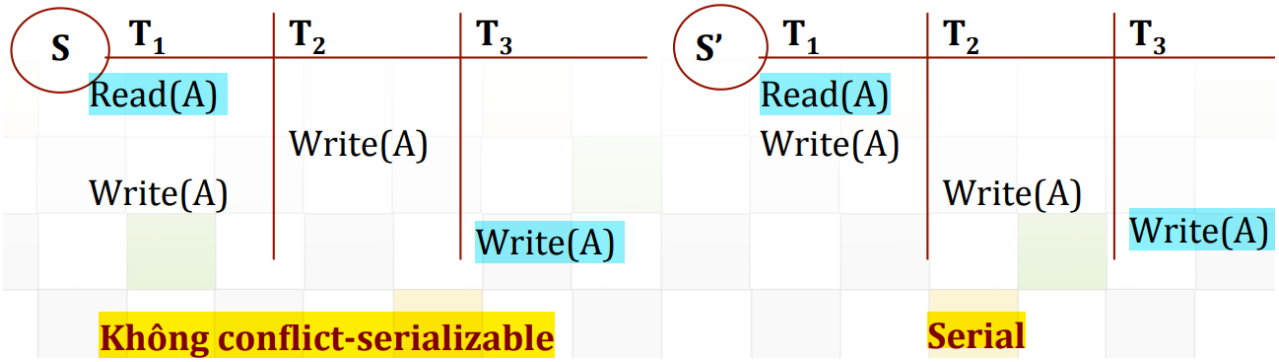
Xét lịch S



P(S) có chu trình → **S không conflict-serializable.**

Vậy S khả tuần tự hay không?

So sánh lịch S và 1 lịch tuần tự S'



Không conflict-serializable **Serial**

– Giả sử **trước khi lịch S thực hiện**, có **giao tác Tb thực hiện việc ghi A** và **sau khi S thực hiện có giao tác Tf thực hiện việc đọc A.**

– Nhận xét lịch S và S':

- **Đều có T1 thực hiện read(A)** từ giao tác Tb → **Kết quả đọc luôn giống nhau**
- **Đều có T3 thực hiện việc ghi cuối cùng lên A.** T2, T3 không có lệnh đọc A

→ Dù S hay S' được thực hiện thì kết quả đọc A của Tf luôn giống nhau

→ Kết quả của S và S' giống nhau → **S vẫn khả tuần tự.**

Khả tuần tự View (View-serializability)

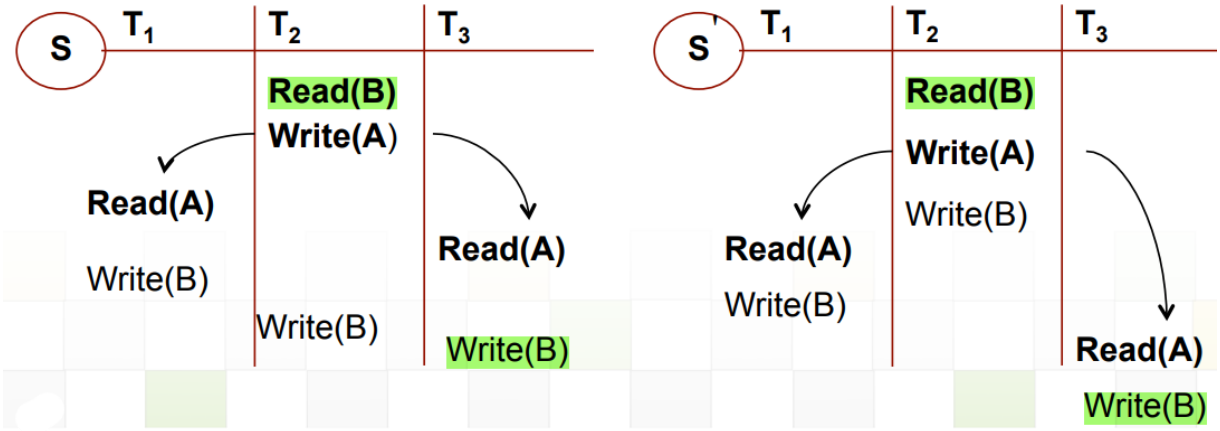
– Một lịch S được gọi là khả tuần tự view nếu **tồn tại một lịch tuần tự S'** được tạo từ các giao tác của S sao cho **S và S' đọc và ghi những giá trị giống nhau.**

– Lịch S được gọi là khả tuần tự view khi và chỉ khi nó **tương đương view (view-equivalent)** với một lịch tuần tự S'.

Ví dụ: Cho lịch S và S' như sau:

S : r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)

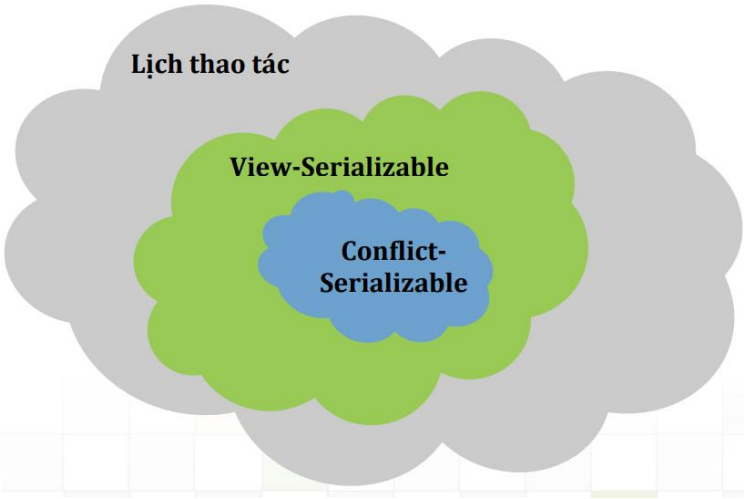
S': r2(B) w2(A) w2(B) r1(A) w1(B) r3(A) w3(B)



→ **S khả tuần tự view.**

View-equivalent: S và S' là những lịch view-equivalent nếu **thỏa các điều kiện sau:**

- Nếu trong **S** có **Ti đọc giá trị ban đầu của A** thì **nó cũng đọc giá trị ban đầu của A** trong **S'.**
- Nếu **Ti đọc giá trị của A được ghi bởi Tj trong S** thì **Ti cũng phải đọc giá trị của A được ghi bởi Tj trong S'.**
- Với mỗi dvdl A, **giao tác thực hiện lệnh ghi cuối cùng lên A (nếu có) trong S thì giao tác đó cũng phải thực hiện lệnh ghi cuối cùng lên A trong S'**.



- Một lịch giao tác S là view-serializable: Nếu **S là view-equivalent với một Lịch giao tác tuần tự S'** nào đó

- Nếu S là conflict-serializable → S view-serializable. Không có chiều ngược lại.

2.3.2.4.4 Kiểm tra View Serializability

Phương pháp 1: Đồ thị phức (PolyGraph)

- Cho 1 Lịch giao tác S
- **Thêm 1 giao tác cuối Tf** vào trong S sao cho Tf thực hiện việc **đọc hết tất cả đơn vị dữ liệu** ở trong S

S = … w1(A)……w2(A) **rf(A)**

- **Thêm 1 giao tác đầu tiên Tb** vào trong S sao cho Tb thực hiện việc **ghi các giá trị ban đầu cho tất cả đơn vị dữ liệu**

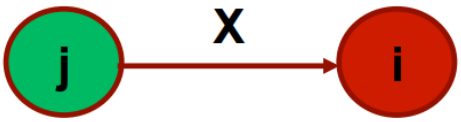
S = **wb(A)**… w1(A)……w2(A)…

- Vẽ đồ thị phức (PolyGraph) cho S, ký hiệu G(S) với

– Đỉnh là các giao tác Ti (**bao gồm cả Tb và Tf**)

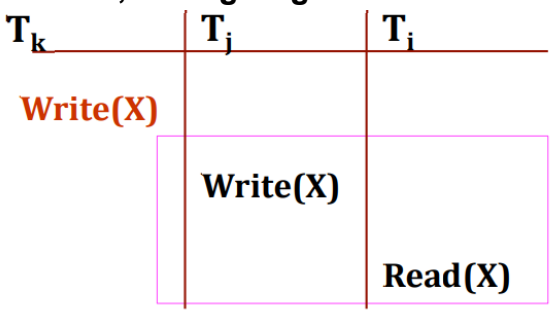
– Cung:

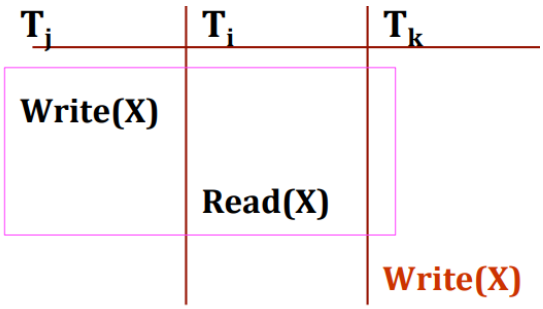
• (1) Nếu giá trị mà ri(X) đọc được là do Tj ghi **(ri(X) có gốc là wj(X))** thì **vẽ cung đi từ Tj đến Ti.** *(mỗi thao tác đọc đơn vị dữ liệu X ứng với thao tác ghi X gần nhất với thao tác đọc đang xét).*

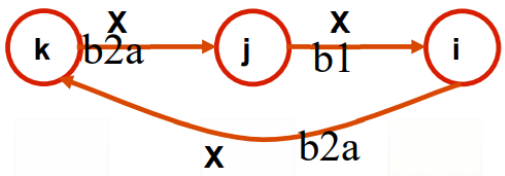


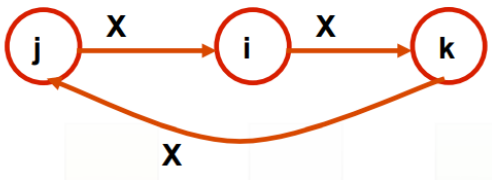
• (2) Với mỗi **wj(X) … ri(X),** xét **wk(X) khác Tb** sao cho **Tk không chèn vào giữa Tj và Ti.**

• **(2a)** Nếu **Tj ≠ Tb và Ti ≠ Tf** thì vẽ cung Tk → Tj và Ti → Tk **(1 cung từ k đến giao tác chứa write, 1 cung từ giao tác chứa read đến k).**

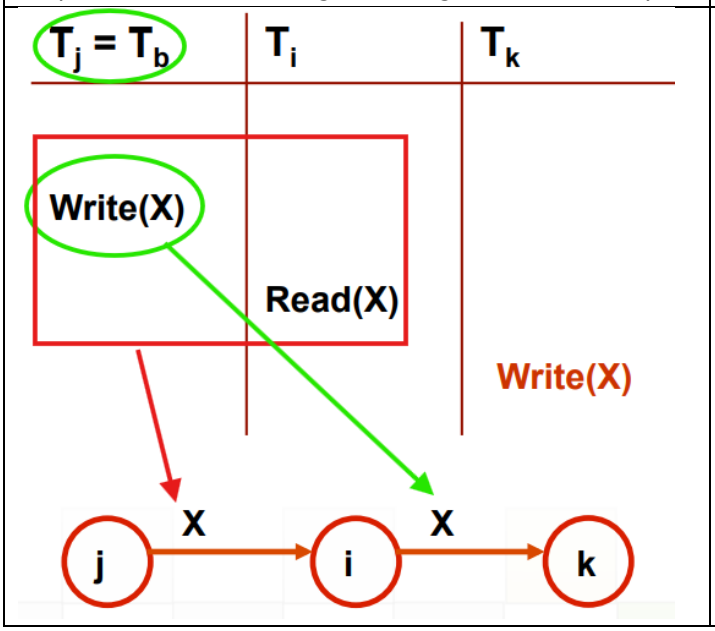
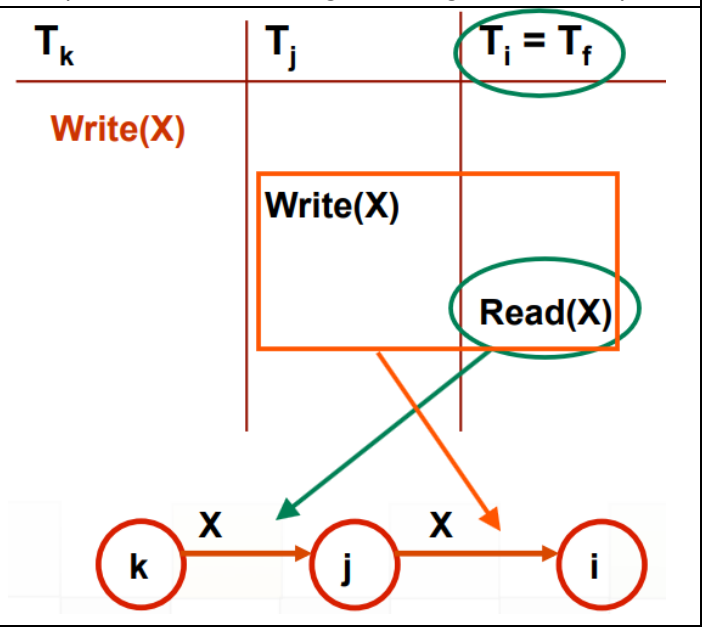




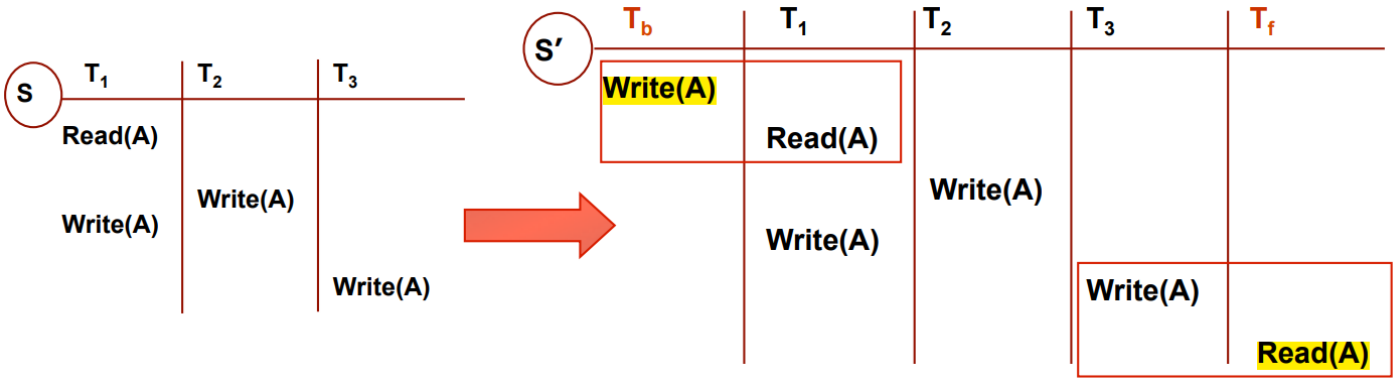


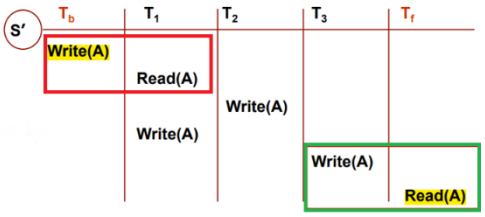


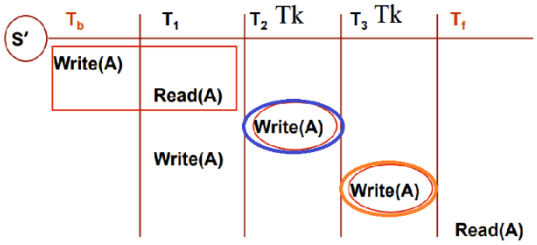
Tk có thể nằm trước Tj hoặc sau Ti.

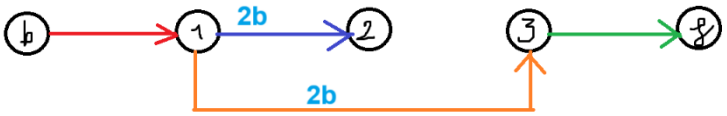
• **(2b)** Nếu **Tj = Tb** (trùng với giao tác bắt đầu là hành động write) thì vẽ cung **Ti → Tk (vẽ cung từ giao tác chứa read đến k).** *(Do bắt buộc không có cung nào trước Tb)*	• **(2c)** Nếu **Ti = Tf** (trùng với giao tác kết thúc là giao tác đọc) thì vẽ cung **Tk → Tj (vẽ cung từ k đến giao tác chứa write).** *(Do bắt buộc không có cung nào sau Tf)*
	

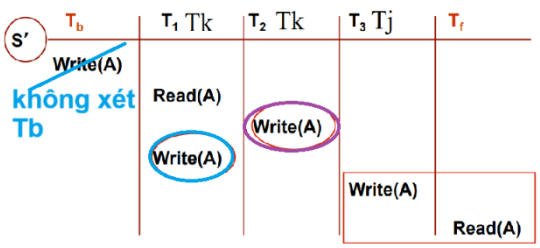
Ví dụ 1:

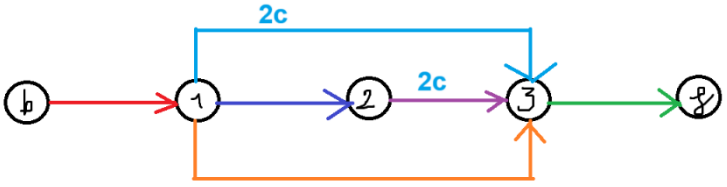


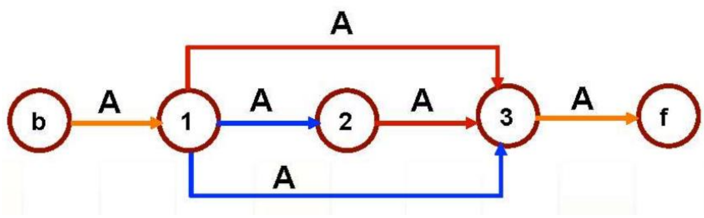








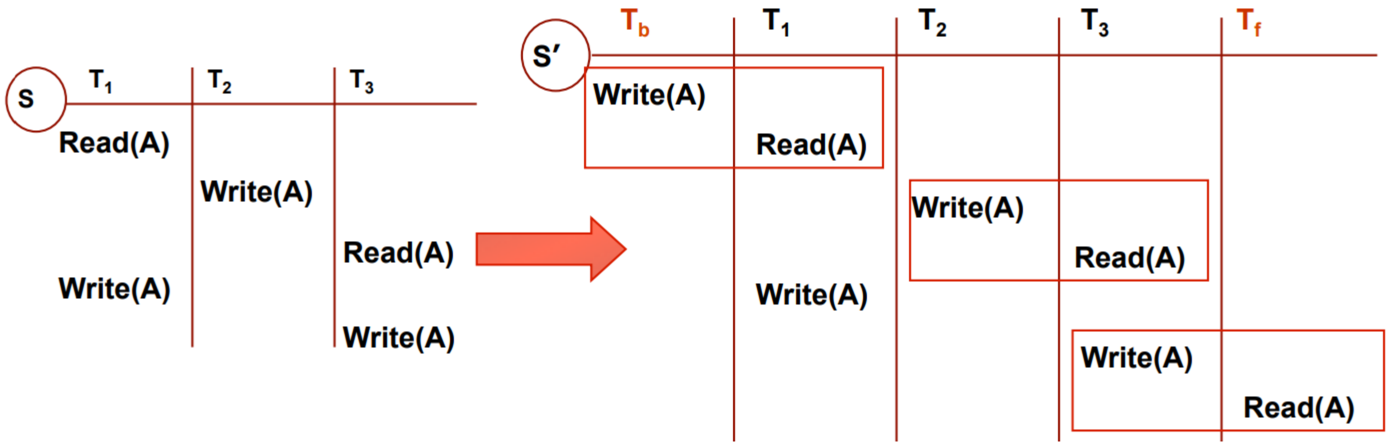


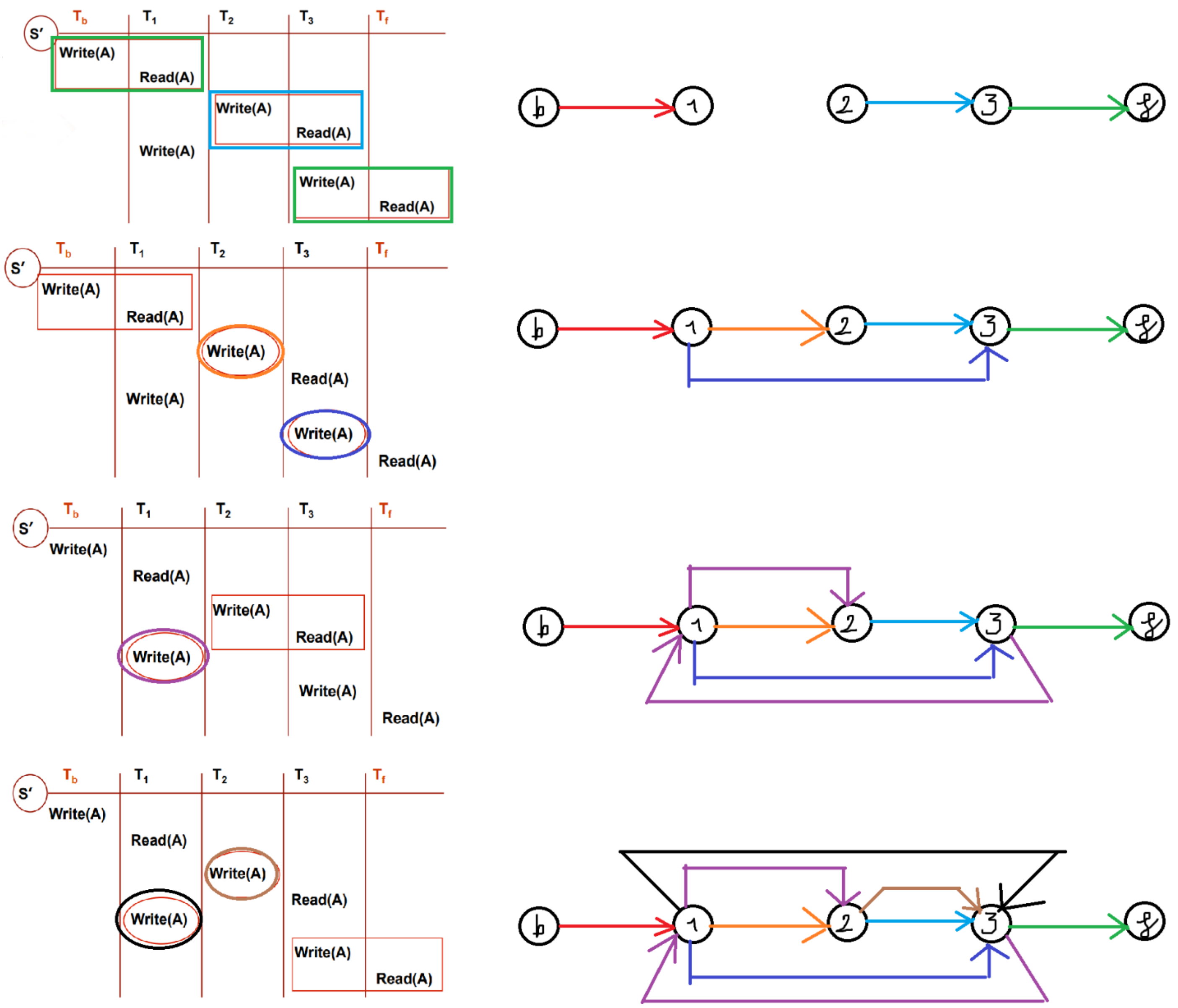


G(S) không có chu trình

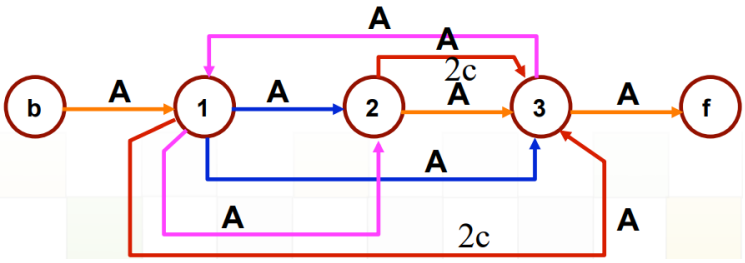
→ S view-serializable theo thứ tự T_b, T_1, T_2, T_3, T_f

Ví dụ 2:



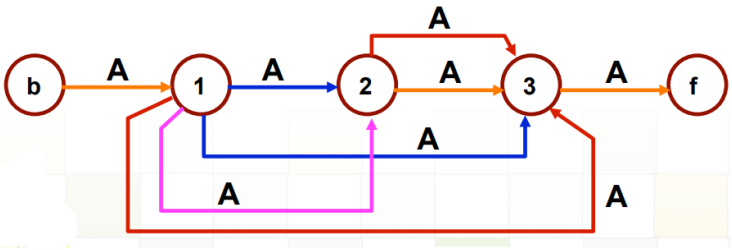


Vẽ lại:



G(S) có chu trình.

G(S) không có chu trình **sau khi bỏ cung** → **S view-serializable**



Phương pháp 2: Đồ thi phụ thuộc (dependency graph)

There's another method to check the view-serializability of a schedule.

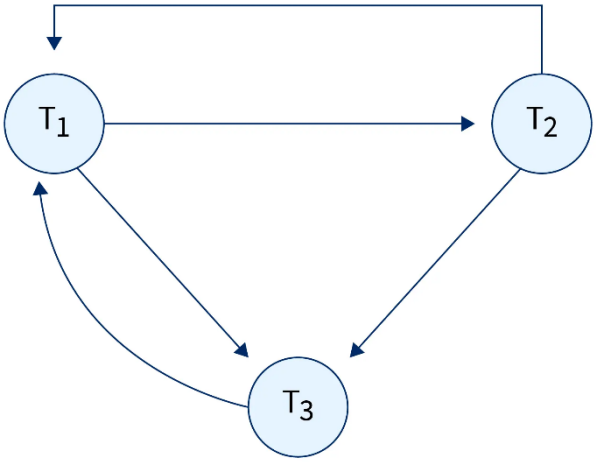
The first step of this method is the same as the previous method i.e. **checking the conflict serializability** of the schedule **by creating the precedence graph.**

T1	T2	T3
R(X)		
	W(X)	
		R(X)
W(X)		
		W(X)

Writing all the conflicting operations:

- R1(X), W2(X) (T1 → T2)
- R1(X), W3(X) (T1 → T3)
- W2(X), R3(X) (T2 → T3)
- W2(X), W1(X) (T2 → T1)
- W2(X), W3(X) (T2 → T3)
- R3(X), W1(X) (T3 → T1)
- W1(X), W3(X) (T1 → T3)

The precedence graph for the above schedule is as follows:



If the precedence graph doesn't contain a loop/cycle, then it is conflict serializable, and thus concludes that the given schedule is consistent.

As the above graph **contains a loop/cycle,** it **does not conflict with serializable.** Thus we need to perform the following steps.

Next, we will check for **blind writes.** If the blind writes don't exist, then the schedule is non-view Serializable. We can simply conclude that the given schedule is inconsistent.

If there exist any blind writes in the schedule, then it may or may not be view serializable.

*** Ghi mù (Blind writes):** If a **write action** is performed on a data item **by a Transaction** (updation), **without performing the reading operation** then it is known as **blind write.**

In the above example, **transaction T2 contains a blind write,** thus we are **not sure** if the given schedule **is view-serializable or not.**

Lastly, we will draw a **dependency graph (đồ thị phụ thuộc)** for the schedule.

Note: The dependency graph is different from the precedence graph.

→ In Dependence Graph method, we find the serial schedule corresponding to Non-conflict Serializable by checking for three conditions

* First Read (R-W)
* First updated Read (W-R)
* Last Write (W-W)

https://www.youtube.com/watch?v=VprVTQ-cH7E

(1) First Read → $T_1 \rightarrow T_2$

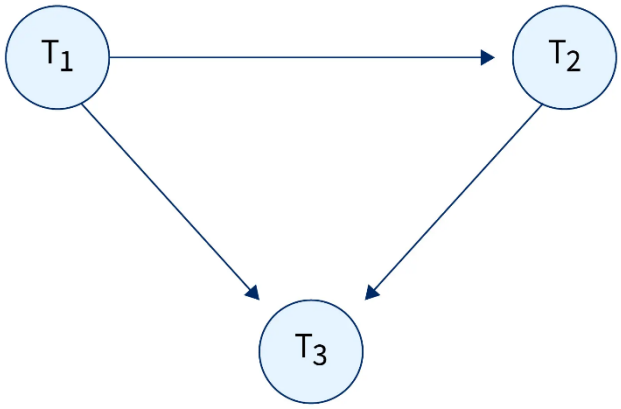
(2) W-R → $T_2 \rightarrow T_3$

(3) W-W → $(T_1, T_2) \rightarrow T_3$

Steps for dependency graph:

- Firstly, T1 reads X, and T2 first updates X. So, **T1 must execute before the T2 operation**. (T1 → T2)
- **T2 updates X before T3**, so we get the dependency as (T2 → T3)
- **T3 performs the final updation** on X thus, **T3 must execute after T1 and T2.** (T1 → T3 and T2 → T3)

The dependency graph is formed as follows:



As **no cycles** exist in the dependency graph for the example, we can say that **the schedule is view serializable.**

If any cycle/ loop exists, then it is not view-serializable and the schedule is inconsistent.

If cycle/loop doesn't exist, then it is view-serializable.

Bài tập:

Conflict-Serializability: Cho các lịch S sau:

1. S: w1(A) r2(A) r3(A) w4(A)
2. S: w3(A) w2(C) r1(A) w1(B) r1(C) w2(A) r4(A) w4(D)
3. S: r1(A) w2(A) w1(A) w3(A)
4. S: r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)
5. S: w1(X) w2(Y) w2(X) w1(X) w3(X)
6. S: r2(A) r1(B) w2(A) r3(A) w1(B) r2(B) w2(B)
7. S: r2(A) r1(B) w2(A) r2(B) r3(A) w1(B) w3(A) w2(B)

Vẽ P(S). S có conflict-serializable không? Nếu có thì S tương đương với lịch tuần tự nào?

View-Serializability: Cho lịch S:

1. S: r2(B) w2(A) r1(A) r3(A) w1(B) w2(B) w3(B)
2. S: w1(A) r3(A) r2(A) w2(A) r1(A) w3(A)
3. S: r2(A) r1(A) w1(C) r3(C) w1(B) r4(B) w3(A) r4(C) w2(D) r2(B) w4(A) w4(B)
4. S: w1(A) r2(A) w2(A) r1(A)
5. S: r1(A) r3(D) w1(B) r2(B) w3(B) r4(B) w2(C) r5(C) w4(E) r5(E) w5(B)
6. S: w1(A) r2(A) w3(A) r4(A) w5(A) r6(A)
7. S: r1(X) r2(X) w1(X) w2(X)
   - Vẽ G(S)
   - S có view-serializable hay không?