

# Điều khiển truy xuất đồng thời

## Nội dung chi tiết

### *\* Các vấn đề của Truy xuất đồng thời*

#### *\* Kỹ thuật khóa*

- ❖ Khóa đơn giản
- ❖ Khóa đọc ghi
- ❖ Khóa đa hạt

#### *\* Kỹ thuật nhãn thời gian*

- ❖ Nhãn thời gian toàn phần
- ❖ Nhãn thời gian riêng phần
- ❖ Nhãn thời gian nhiều phiên bản

#### *\* Kỹ thuật lạc quan*

#### *\* Vấn đề khóa chết*

#### *\* Các vấn đề khác*

✱ *Xét 2 giao tác T1 và T2 và đơn vị dữ liệu A vốn có giá trị ban đầu là 50*

- ❖ Giả sử T1 và T2 thực hiện tuần tự (T1 rồi T2 hoặc t2 rồi T1) thì cuối cùng  $A = 80$
- ❖ Nếu chỉ có T1 thì  $A = 60$
- ❖ Nếu chỉ có T2 thì  $A = 70$
- ❖ Giả sử T1 và T2 được thực hiện đồng thời theo lịch sau

A=50	T <sub>1</sub>	T <sub>2</sub>
t <sub>1</sub>	Read(A)	
t <sub>2</sub>		Read(A)
t <sub>3</sub>	A:=A+10	
t <sub>4</sub>		A:=A+20
t <sub>5</sub>	Write(A)	
t <sub>6</sub>		Write(A)

A=60

A=70

## Vấn đề không thể đọc lại

✱ *Xét 2 giao tác T1 và T2 và đơn vị dữ liệu A vốn có giá trị ban đầu là 50*

✱ *Giả sử T1 và T2 được thực hiện đồng thời theo lịch sau*

A=50	T <sub>1</sub>	T <sub>2</sub>
t <sub>1</sub>	Read(A)	
t <sub>2</sub>		<b>Read(A)</b> A=50
t <sub>3</sub>	A:=A-10	
t <sub>4</sub>		Print(A)    A=50
t <sub>5</sub>	Write(A)	
t <sub>6</sub>		<b>Read(A)</b> A=60
t <sub>7</sub>		Print(A)    A=60

✱ *Xét 2 giao tác T1 và T2 được xử lý đồng thời*

- ❖ A là một tập các đơn vị dữ liệu  $a_1, a_2, a_3, a_4, \dots$
- ❖ T1 xử lý trên toàn bộ tập A
- ❖ Khi T1 đang xử lý, T2 thêm hay xóa một hay một số phần tử trong tập A

	T1	T2
t1	Read(A)	
t4	Xử lý 1 trên A	
t3		<b>Thêm ai vào A</b>
t4	Xử lý 2 trên A	
t5		<b>Xóa aj khỏi A</b>
t6	Xử lý 3 trên A	

## Vấn đề đọc dữ liệu rác

✱ *Xét 2 giao tác T1 và T2 được xử lý đồng thời*

- ❖ T2 đã đọc dữ liệu được ghi bởi T1 nhưng sau đó T1 yêu cầu hủy việc ghi

	$T_1$	$T_2$
$t_1$	Read(A)	
$t_2$	$A := A + 10$	
$t_3$	Write(A)	
$t_4$		<b>Read(A)</b>
$t_5$		<b>Print(A)</b>
$t_6$	<b>Abort</b>	

✳ *Các vấn đề của Truy xuất đồng thời*

✳ *Kỹ thuật khóa*

❖ Khóa đơn giản

❖ Khóa đọc ghi

❖ Khóa đa hạt

✳ *Kỹ thuật nhân thời gian*

❖ Nhấn thời gian toàn phần

❖ Nhấn thời gian riêng phần

❖ Nhấn thời gian nhiều phiên bản

✳ *Kỹ thuật lạc quan*

✳ *Vấn đề khóa chết*

✳ *Các vấn đề khác*

7

## Giới thiệu

✳ *Làm thế nào để bộ lập lịch ép buộc 1 lịch phải khả tuần tự?*

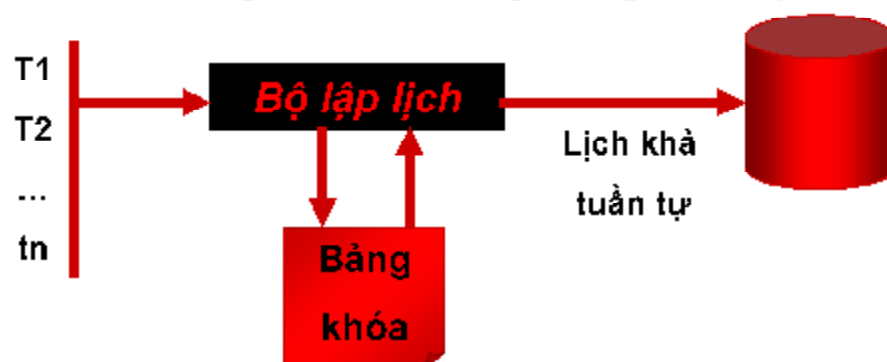
✳ *Bộ lập lịch với cơ chế khóa (locking scheduler)*

❖ Là bộ lập lịch với thêm 2 hành động :

⊗ Lock : Phát khóa

⊗ Unlock : Giải phóng khóa

❖ Các khóa được ghi nhận trong bảng khóa (Lock Table)



## \* 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
  - ⊗ Nếu yêu cầu được chấp thuận thì giao tác mới được phép đọc/ghi lên đơn vị dữ liệu
- ❖ 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

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

Element	Transaction
A	T1

## Kỹ thuật khóa đơn giản (tt)

### \* Qui tắc

- ❖ (1) Giao tác đúng đắn : Việc giao tác  $T_i$  đọc hay ghi trên 1 đơn vị dữ liệu A phải sau khi  $T_i$  phát khóa trên A và trước khi  $T_i$  giải phóng khóa trên A. Phát khóa và giải phóng khóa phải đi đôi với nhau (lock trước, unlock sau)
  - ⊗  $T_i$  : ... l(A) ... r(A) / w(A) ... u(A) ...
- ❖ (2) Lịch thao tác hợp lệ : Khi  $T_i$  đang giữ khóa trên 1 đơn vị dữ liệu A thì không 1  $T_j$  nào khác được phát khóa trên A
  - ⊗ S : ... li(A) ..... ui(A) ...

←————→  
Không được có  $l_j(A)$

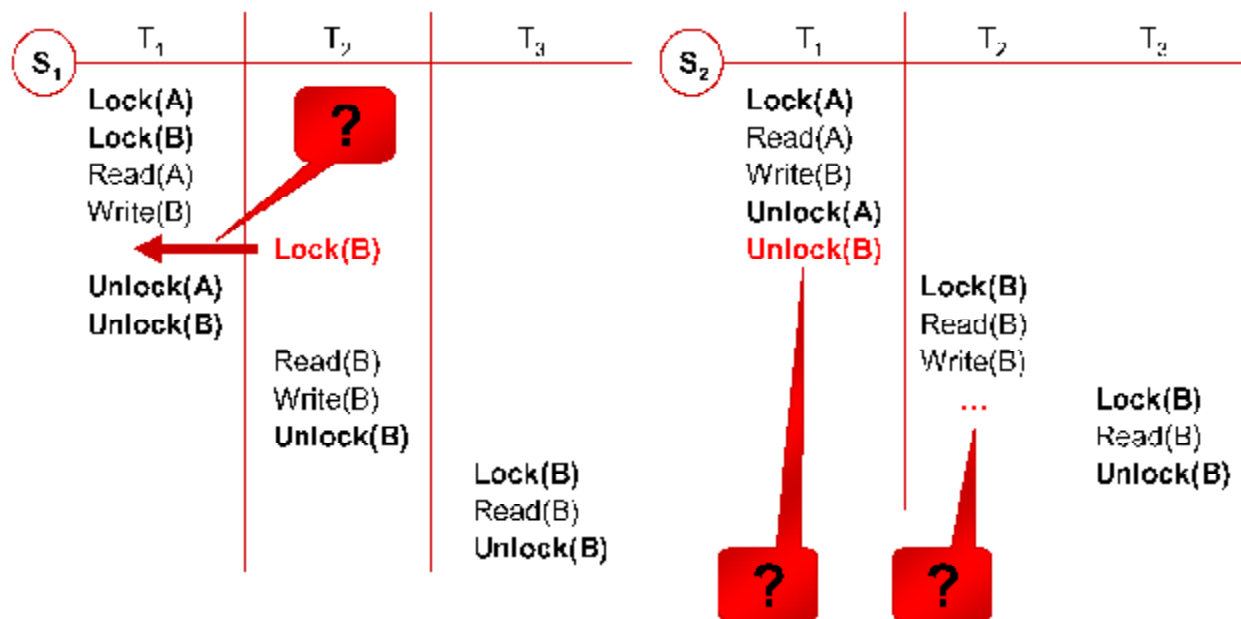
	$T_1$	$T_2$
<b>S</b>	<b>Lock(A)</b> Read(A,t) t:=t+100 Write(A,t) <b>Unlock(A)</b>	<b>Lock(A)</b> Read(A,s) s:=s*2 Write(A,s) <b>Unlock(A)</b> <b>Lock(B)</b> Read(B,s) s:=s*2 Write(B,s) <b>Unlock(B)</b>
	<b>Lock(B)</b> Read(B,t) t:=t+100 Write(B,t) <b>Unlock(B)</b>	

## Ví dụ (tt)

\* Cho biết lịch nào hợp lệ? Giao tác nào là đúng?

<b>S<sub>1</sub></b>	$I_1$	$I_2$	$I_3$	<b>S<sub>2</sub></b>	$I_1$	$I_2$	$I_3$
<b>Lock(A)</b> <b>Lock(B)</b> Read(A) Write(B)  <b>Unlock(A)</b> <b>Unlock(B)</b>		<b>Lock(B)</b>  Read(B) Write(B) <b>Unlock(B)</b>	<b>Lock(B)</b> Read(B) <b>Unlock(B)</b>	<b>Lock(A)</b> Read(A) Write(B) <b>Unlock(A)</b> <b>Unlock(B)</b>		<b>Lock(B)</b> Read(B) Write(B)  <b>Lock(B)</b> Read(B) <b>Unlock(B)</b>	

✳ **Cho biết lịch nào hợp lệ? Giao tác nào là đúng?**



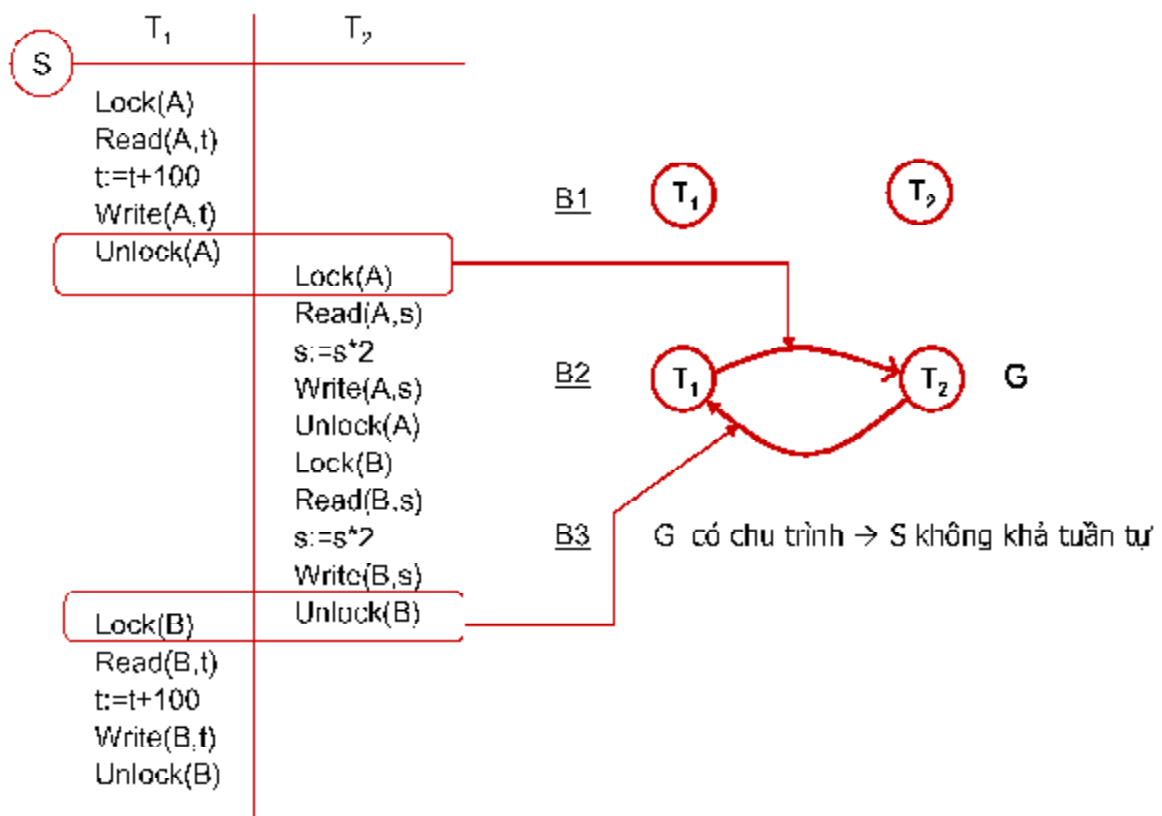
## Kỹ thuật khóa đơn giản (tt)

✳ **Kiểm tra tính khả tuần tự**

- ❖ Input : Lịch S được lập từ n giao tác xử lý đồng thời  $T_1, T_2, \dots, T_n$  theo kỹ thuật khóa đơn giản
- ❖ Output : S khả tuần tự hay không?

✳ **Phương pháp : Xây dựng 1 đồ thị có hướng G**

- ❖ Mỗi giao tác  $T_i$  là 1 đỉnh của đồ thị
- ❖ Nếu một giao tác  $T_j$  phát ra Lock(A) sau một giao tác  $T_i$  phát ra Unlock(A) thì sẽ vẽ cung từ  $T_i$  đến  $T_j$ ,  $i \neq j$
- ❖ S khả tuần tự nếu G không có chu trình



## Kỹ thuật khóa đơn giản (tt)

\* *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

❖ (3) Giao tác **2PL** : Trong 1 giao tác, tất cả các thao tác phát khóa đều xảy ra trước tất cả các thao tác giải phóng khóa

⊛  $T : \dots\dots\dots l(A) \dots\dots\dots u(A) \dots\dots\dots$

Khô  
phó  
kh



phát  
thỏa



\* (3) gọi là *Nghi thức khoá 2 giai đoạn (2PL)*

\* *Định lý : Nếu lịch  $S$  thoả nghi thức 2PL thì  $S$  conflict-serializable*



T <sub>1</sub>
Lock(A)
Read(A)
Lock(B)
Read(B)
B:=B+A
Write(B)
Unlock(A)
Unlock(B)

T <sub>2</sub>
Lock(B)
Read(B)
Lock(A)
Read(A)
Unlock(B)
A:=A+B
Write(A)
Unlock(A)

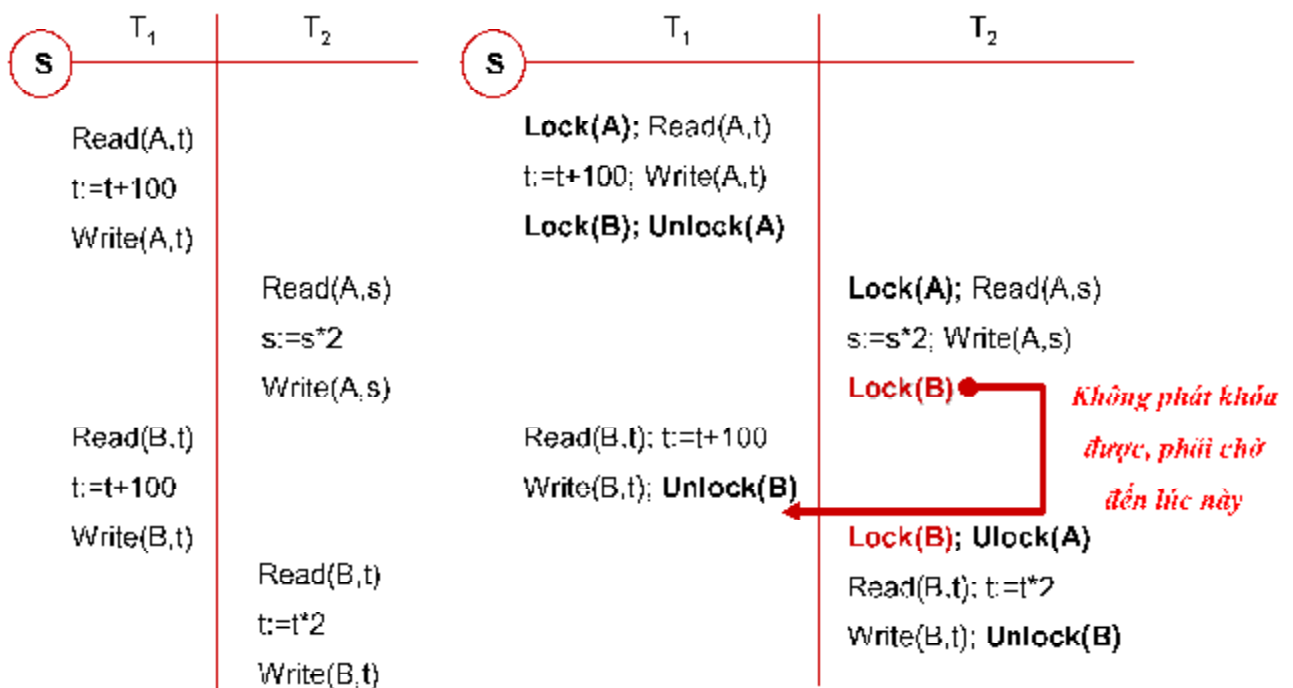
*Thỏa nghi thức  
khóa 2 giai đoạn*

T <sub>3</sub>
Lock(B)
Read(B)
B=B-50
Write(B)
Unlock(B)
Lock(A)
Read(A)
A=A+50
Write(A)
Unlock(A)

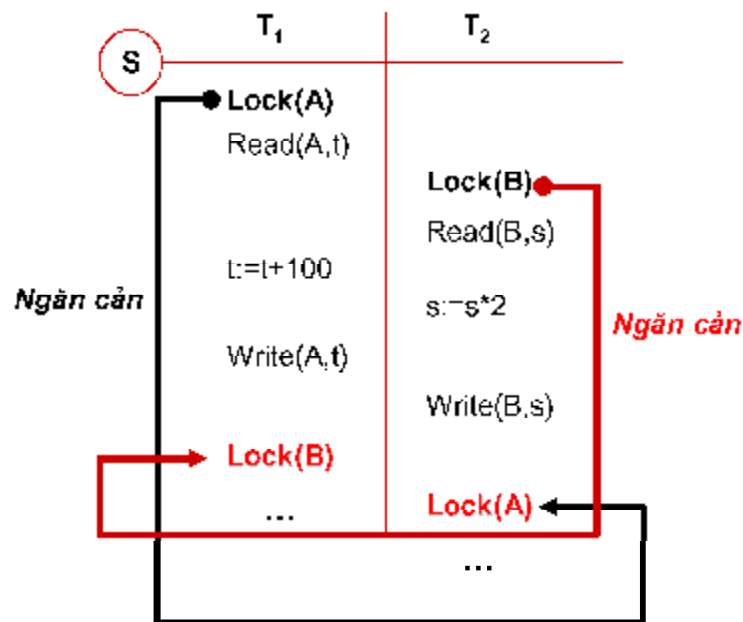
T <sub>4</sub>
Lock(A)
Read(A)
Unlock(A)
Lock(B)
Read(B)
Unlock(B)
Print(A+B)

*Không thỏa nghi thức  
khóa 2 giai đoạn*

## Kỹ thuật khóa đơn giản (tt)



- \* **Chú ý :** Hiện tượng chờ khi cần phát khóa có thể dẫn đến chờ lẫn nhau **vĩnh viễn**



## Nội dung chi tiết

- \* Các vấn đề của Truy xuất đồng thời
- \* Kỹ thuật khóa
  - ❖ Khóa đơn giản
  - ❖ Khóa đọc ghi
  - ❖ Khóa đa hạt
- \* Kỹ thuật nhãn thời gian
  - ❖ Nhãn thời gian toàn phần
  - ❖ Nhãn thời gian riêng phần
  - ❖ Nhãn thời gian nhiều phiên bản
- \* Kỹ thuật lạc quan
- \* Vấn đề khóa chết
- \* Các vấn đề khác

## ※ Vấn đề tồn tại của 2PL

- ❖ Có những tình huống mà  $T_i$  không thực sự cần phải ngăn cản một  $T_j$  truy xuất đơn vị dữ liệu của nó, nhưng theo 2PL vẫn phải ngăn cản
- ❖ Không tối ưu về mặt tốc độ vì có những khoảng chờ không cần thiết, thậm chí gây nên deadlock

## ※ Do đó, bộ lập lịch cần các hành động

- ❖ Khóa đọc (Read lock, Shared lock)
  - ⊛ Ký hiệu : RLock(A) hay rl(A)
- ❖ Khóa ghi (Write lock, Exclusive lock)
  - ⊛ Ký hiệu : WLock(A) hay wl(A)
- ❖ Giải phóng khóa
  - ⊛ Ký hiệu : Unlock(A) hay u(A)

## Kỹ thuật khóa đọc ghi (tt)

### ※ Cho 1 đơn vị dữ liệu A bất kỳ

- ❖ WLock(A)
  - ⊛ Hoặc có 1 khóa ghi duy nhất lên A
  - ⊛ Hoặc không có khóa ghi nào lên A
- ❖ RLock(A)
  - ⊛ Có thể có nhiều khóa đọc được thiết lập lên A
  - ⊛ Ma trận tương thích

	Khóa đọc	Khóa ghi
Khóa đọc		
Khóa ghi		

## **\* Giao tác T muốn Write(A)**

### **❖ Yêu cầu WLock(A)**

- ✱ WLock(A) sẽ được chấp thuận nếu hiện không có khóa nào trên A
- ✱ Từ đó sẽ không có giao tác nào khác nhận được WLock(A) hay RLock(A) cho đến khi T giải phóng khóa trên A

## **\* Giao tác muốn Read(A)**

### **❖ Yêu cầu RLock(A) hoặc WLock(A)**

- ✱ RLock(A) sẽ được chấp thuận nếu A không đang giữ một WLock nào
- ✱ Từ đó sẽ không có giao tác nào khác nhận được WLock(A) cho đến khi T giải phóng khóa trên A. Nhưng không ngăn chặn các thao tác khác cùng xin Rlock(A) nên các giao tác không cần phải chờ nhau khi đọc A

## **\* Sau khi thao tác xong thì giao tác phải giải phóng khóa trên đơn vị dữ liệu A**

## **Kỹ thuật khóa đọc ghi (tt)**

## **\* Quy 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

## \* Quy 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)

## \* Quy tắc

### ❖ (3) - Giao tác 2PL

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

- ⊛  $T : \dots rli(A) \dots wli(A) \dots ui(A) \dots$

Không  
bắt

át ra  
nào

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

- ⊛  $T : \dots rli(A) \dots uli(A)-wli(A) \dots ui(A) \dots$

Chấp nhận giải  
đọc khi nâng

## \* Định lý : $S$ thỏa (1), (2) và (3) $\rightarrow S$ conflict-serializable

	T <sub>1</sub>	T <sub>2</sub>		T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>
<b>S</b>	RLock(A) Read(A) U(A)  WLock(B) Read(B) B:=B+A Write(B) U(B)	RLock(B) Read(B) U(B) WLock(A) Read(A) A:=A+B Write(A) U(A)	<b>S'</b>	RL(A)  WL(B) U(A) U(B) RL(B)  RL(A) WL(C) U(A)  U(B) U(C)	RL(A)  WL(A)  U(A)  RL(B) U(B)  WL(B) U(B)		

## Kỹ thuật khóa đọc ghi (tt)

### \* Khóa cập nhật (update lock)

❖ Giao tác muốn read(A) và sau đó cũng muốn write(A)

⊛ Yêu cầu khóa cập nhật (khóa dự định ghi) : ULock(A)

- ▶ ULock(A) được chấp thuận khi A tự do hoặc đang giữ RLock
- ▶ Khi trên A đang có ULock, không có giao tác nào khác được phát WLock(A) hay ULock(A) nữa

❖ Ma trận tương thích

	Khóa đọc	Khóa ghi	Khóa cập nhật
Khóa đọc	✓	✗	✓
Khóa ghi	✗	✗	✗
Khóa cập nhật	✓	✗	✗

✱ *Các vấn đề của Truy xuất đồng thời*

✱ *Kỹ thuật khóa*

❖ Khóa đơn giản

❖ Khóa đọc ghi

❖ Khóa đa hạt

✱ *Kỹ thuật nhãn thời gian*

❖ Nhãn thời gian toàn phần

❖ Nhãn thời gian riêng phần

❖ Nhãn thời gian nhiều phiên bản

✱ *Kỹ thuật lạc quan*

✱ *Vấn đề khóa chết*

✱ *Các vấn đề khác*

29

## *Kỹ thuật khóa đa hạt*

✱ *Xét ví dụ hệ thống ngân hàng*

❖ Quan hệ TàiKhoản(mãTK, sốDư)

❖ 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

⊗ Khóa tuple hay disk block

▶ 2 tài khoản ở 2 blocks khác nhau có thể được cập nhật cùng thời điểm

▶ Xử lý đồng thời nhanh

❖ Giao tác tính tổng số tiền của các tài khoản

⊗ Khóa relation?

⊗ Khóa tuple hay disk block?

### \* *Phải quản lý khóa ở nhiều mức độ*

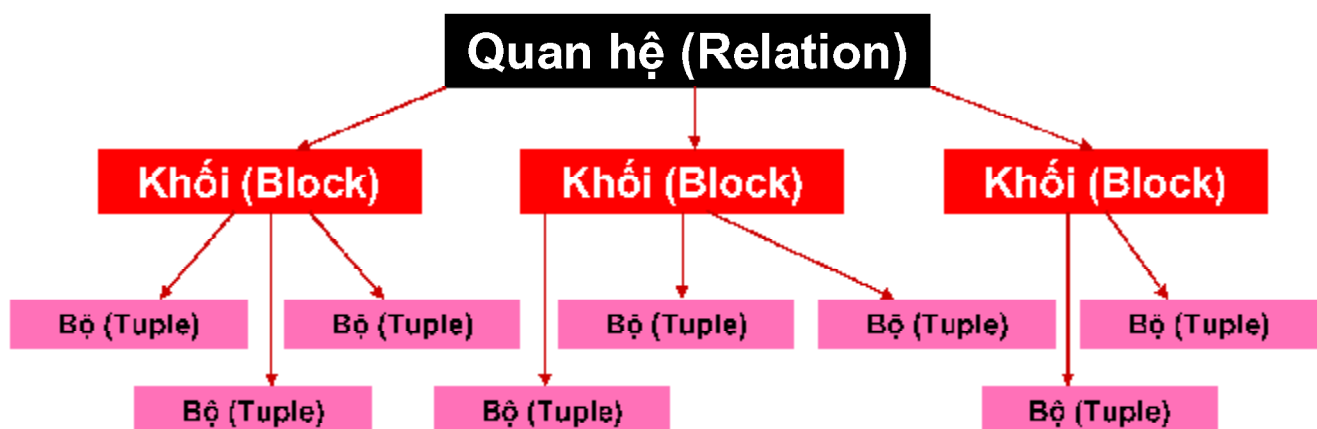
- ❖ Tính chất hạt (granularity) : Tính hạt càng tăng khi đơn vị dữ liệu bị khóa càng lớn
- ❖ Tính đồng thời (concurrency) : Tính đồng thời càng tăng khi đơn vị dữ liệu bị khóa càng nhỏ
- ❖ Tính hạt tăng thì tính đồng thời giảm và ngược lại → phải thỏa hiệp



## Kỹ thuật khóa đa hạt (tt)

### \* *Phân cấp Dữ liệu*

- ❖ Relations là đơn vị dữ liệu khóa lớn nhất
- ❖ Một relation gồm 1 hoặc nhiều blocks (pages)
- ❖ Một block gồm 1 hoặc nhiều tuples





## \* *Gồm các khóa*

### ❖ *Khóa thông thường*

- ⊛ Shared lock: S
- ⊛ Exclusive lock: X

### ❖ *Khóa cảnh báo (warning lock)*

- ⊛ Warning (intention to) shared lock: IS
- ⊛ Warning (intention to) exclusive lock: 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 node dữ liệu

	<i>IS</i>	<i>IX</i>	<i>S</i>	<i>X</i>
<i>IS</i>	✓	✓	✓	✗
<i>IX</i>	✓	✓	✗	✗
<i>S</i>	✓	✗	✓	✗
<i>X</i>	✗	✗	✗	✗

✱ ***Ma trận tương thích giữa node cha và node con***

- ❖ Cho biết các khóa có thể phát trên node con khi node cha đang có một khoá nào đó → Cho biết muốn phát 1 khóa trên node con thì trước đó phải phát khóa nào trên node cha

<i>Node cha</i>	<i>Node con</i>
IS	IS, S
IX	IS, S, IX, X
S	S, IS (không cần thiết)
X	

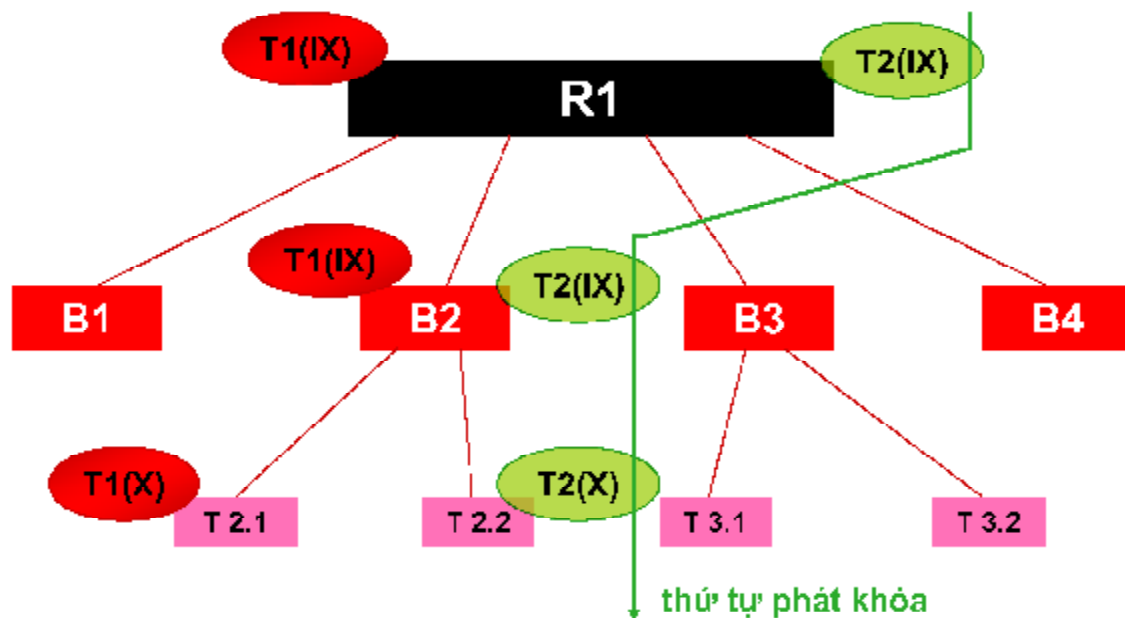
Không cần thiết vì dùng IX là đủ

## Kỹ thuật khóa đa hạt (tt)

✱ ***Quy tắc :***

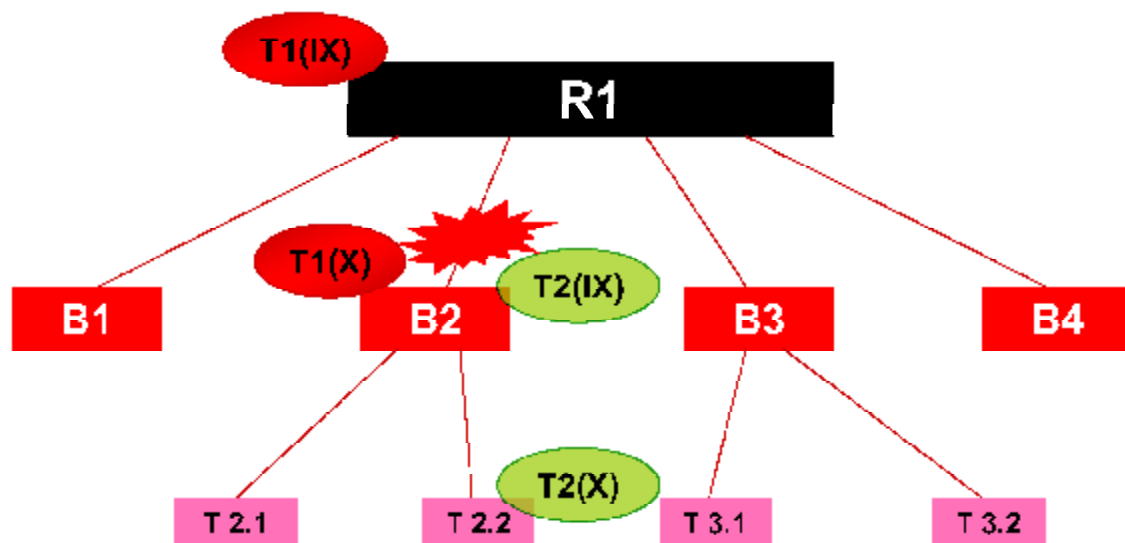
- ❖ (1) Thỏa ma trận tương thích trên cùng một node
- ❖ (2) Khóa nút gốc của cây trước
- ❖ (3) Thỏa ma trận tương thích giữa node cha và node con
- ❖ (4)  $T_i$  thỏa 2PL
- ❖ (5)  $T_i$  có thể giải phóng nút Q khi không có nút con nào của Q bị khóa bởi  $T_i$
- ❖ (6) Trong trường hợp thêm mới hay xóa bỏ một node Q thì cha của Q phải được khóa bằng X trước → tránh Phantom

- \* *T2 có thể truy xuất T2.2 bằng khóa X được không?*
- \* *T2 sẽ có những khóa gì?*

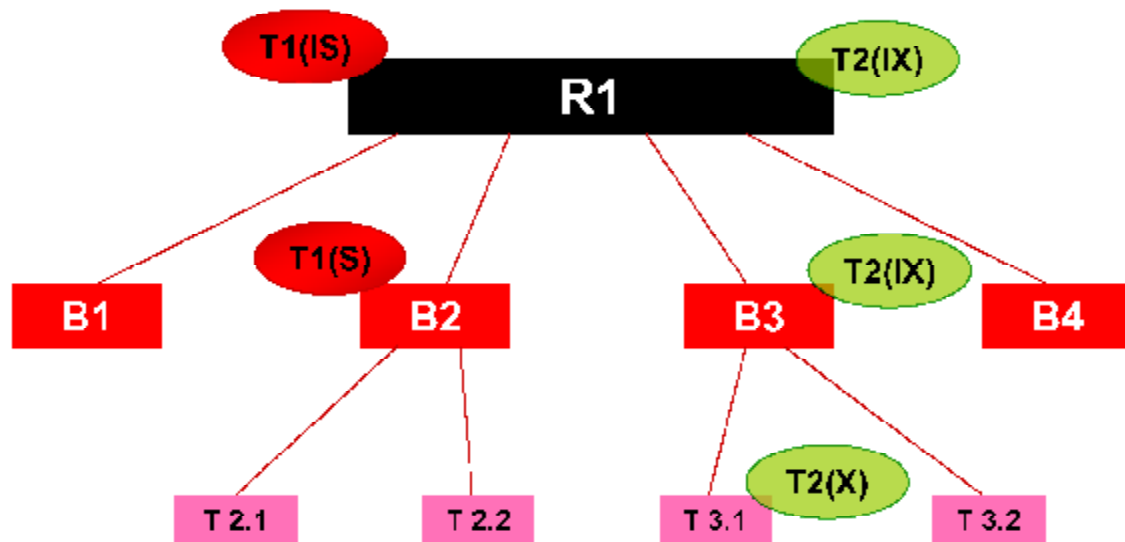


## Bài tập (tt)

- \* *T2 có thể truy xuất f2.2 bằng khóa X được không?*
- \* *T2 sẽ có những khóa gì?*



- \* *T2 có thể truy xuất f3.1 bằng khóa X được không?*
- \* *T2 sẽ có những khóa gì?*



## Nội dung chi tiết

- \* *Các vấn đề của Truy xuất đồng thời*
- \* *Kỹ thuật khóa*
  - ❖ Khóa đơn giản
  - ❖ Khóa đọc ghi
  - ❖ Khóa đa hạt
- \* *Kỹ thuật nhãn thời gian*
  - ❖ Nhãn thời gian toàn phần
  - ❖ Nhãn thời gian riêng phần
  - ❖ Nhãn thời gian nhiều phiên bản
- \* *Kỹ thuật lạc quan*
- \* *Vấn đề khóa chết*
- \* *Các vấn đề khác*

## ※ Ý tưởng

- ❖ Mặc định cho rằng tất cả mọi thao tác của các giao tác dù diễn ra đồng thời cũng không gây mất nhất quán dữ liệu
- ❖ Nếu lỡ có thao tác mang nguy cơ mất nhất quán dữ liệu → Hủy giao tác tương ứng rồi chạy lại giao tác ấy sau

## ※ Chọn một thứ tự thực hiện nào đó cho các giao tác bằng cách gán nhãn thời gian (timestamping)

- ❖ Mỗi giao tác  $T$  sẽ có 1 nhãn thời gian, ký hiệu  $TS(T)$ , nhãn này gán ngay lúc  $T$  bắt đầu bằng 1 trong 2 cách :
  - ⊛ Đồng hồ máy tính
  - ⊛ Bộ lập lịch tự đếm
- ❖ Thứ tự của các nhãn tăng dần,  $T_i$  trễ hơn  $T_j$  thì  $TS(T_i) > TS(T_j)$ , kết quả là 1 lịch tương đương với lịch tuần tự theo trình tự gán nhãn thời gian

## 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  $TS(X) \leq TS(T)$  thì cho  $T$  đọc  $X$  và gán  $TS(X) = TS(T)$
    - ⊛ Ngược lại  $T$  bị hủy (abort)
  - ❖ **Nếu  $T$  muốn ghi  $X$** 
    - ⊛ Nếu  $TS(X) \leq TS(T)$  thì cho  $T$  ghi  $X$  và gán  $TS(X) = TS(T)$
    - ⊛ Ngược lại  $T$  bị hủy (abort)

$T_1$	$T_2$	A	B	
$TS(T_1)=100$	$TS(T_2)=200$	$TS(A)=0$	$TS(B)=0$	
Read(A)		$TS(A)=100$		$TS(A) \leq TS(T_1) : T_1$ đọc được A
	Read(B)		$TS(B)=200$	$TS(B) \leq TS(T_2) : T_2$ đọc được B
$A=A*2$				
Write(A)		$TS(A)=100$		$TS(A) \leq TS(T_1) : T_1$ ghi lên A được
	$B=B+20$			
	Write(B)		$TS(B)=200$	$TS(B) \leq TS(T_2) : T_2$ ghi lên B được
Read(B)				$TS(B) > TS(T_1) : T_1$ không đọc được B

*T1 bị hủy, sau đó khởi động lại T1 với một nhãn thời gian mới, lúc đó T2 đã chạy rồi nên  $TS_{mới}(T1) > TS(T2)$ , lịch này khi đó tương đương 1 lịch tuần tự mà T2 làm trước T1*

## Nội dung chi tiết

### \* Các vấn đề của Truy xuất đồng thời

#### \* Kỹ thuật khóa

- ❖ Khóa đơn giản
- ❖ Khóa đọc ghi
- ❖ Khóa đa hạt

#### \* Kỹ thuật nhãn thời gian

- ❖ Nhãn thời gian toàn phần
- ❖ Nhãn thời gian riêng phần
- ❖ Nhãn thời gian nhiều phiên bản

#### \* Kỹ thuật lạc quan

#### \* Vấn đề khóa chết

#### \* Các vấn đề khác

## ※ Nhận xét

- ❖ Kỹ thuật nhãn thời gian toàn phần không phân biệt thao tác ghi với thao tác đọc
- ❖ Thực tế trong xử lý đồng thời thao tác ghi và đọc có vai trò khác xa nhau
- ❖ Do đó cần quan tâm sự khác biệt này

## ※ Nhãn của đơn vị dữ liệu X được tách ra thành 2 nhãn

### ❖ RT(X) - read

- ⊛ Ghi nhận TS(T) với T là giao tác gần nhất đọc X thành công

### ❖ WT(X) - write

- ⊛ Ghi nhận TS(T) với T là giao tác gần nhất ghi X thành công

## Nhãn thời gian riêng phần (tt)

## ※ Bộ lập lịch sẽ gán gán các nhãn thời gian TS, RT và WT

## ※ Thi giao tác cần truy xuất dữ liệu, xử lý theo 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  $RT(X) = \text{Max}(RT(X), TS(T))$
- ⊛ Ngược lại hủy T và khởi động lại T với TS(T) mới

### ❖ 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 thì bỏ qua thao tác ghi này của T (không hủy T)
- ⊛ Ngược lại hủy T và khởi động lại T với TS(T) mới

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

## Ví dụ (tt)

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

**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**



✱ *Các vấn đề của Truy xuất đồng thời*

✱ *Kỹ thuật khóa*

- ❖ Khóa đơn giản
- ❖ Khóa đọc ghi
- ❖ Khóa đa hạt

✱ *Kỹ thuật nhãn thời gian*

- ❖ Nhãn thời gian toàn phần
- ❖ Nhãn thời gian riêng phần

❖ Nhãn thời gian nhiều phiên bản

✱ *Kỹ thuật lạc quan*

✱ *Vấn đề khóa chết*

✱ *Các vấn đề khác*

48

## Nhãn thời gian nhiều phiên bản

✱ *Ý tưởng*

- ❖ Cho phép thao tác read3(A) 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*

✱ *Ưu :*

- ❖ Giao tác ghi ít bị hủy
- ❖ Thao tác đọc thì luôn đọc được và không bao giờ bị hủy

✱ *Khuyết :*

- ❖ Tốn bộ nhớ lưu các phiên bản (nên giải phóng các phiên bản quá cũ)
- ❖ Tốn chi phí tìm kiếm phiên bản phù hợp

✱ *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 tuần tự**

✱ *Một phiên bản mới của X sẽ được tạo khi hành động ghi X thành công*

## **Nhân thời gian nhiều phiên bản (tt)**

✱ *Khi một giao tác T có nhu cầu truy xuất đơn vị dữ liệu X, xử lý theo quy tắc sau :*

❖ **Nếu T muốn đọc X**

⊛ Tìm i là giá trị lớn nhất sao cho  $WT(X_i) \leq TS(T)$

⊛ Cho T đọc  $X_i$  và gán  $RT(X_i) = \text{Max}(RT(X_i), TS(T))$

❖ **Nếu T muốn ghi X**

⊛ Tìm i là giá trị lớn nhất sao cho  $WT(X_i) \leq TS(T)$

⊛ Nếu  $RT(X_i) > TS(T)$  thì hủy T

⊛ Ngược lại thì

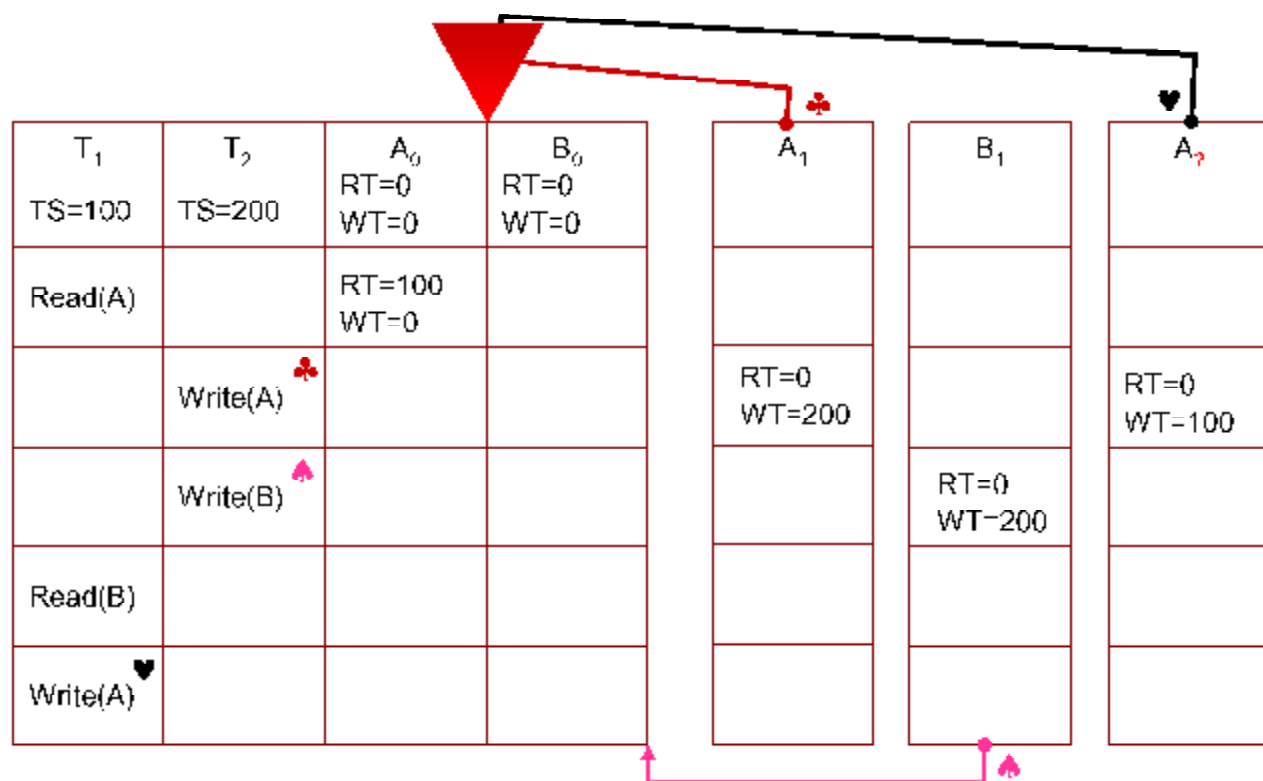
▶ **Tạo phiên bản  $X_{i+1}$  và cho T ghi giá trị vào phiên bản này**

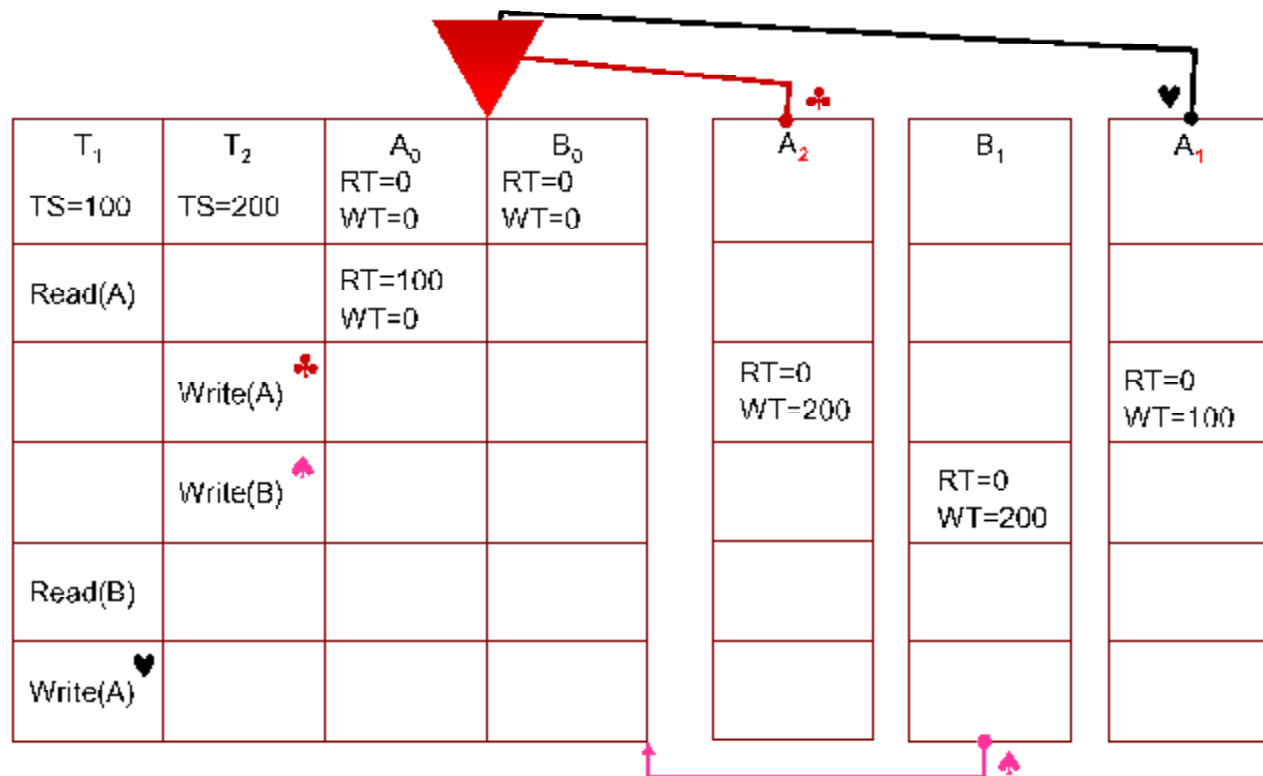
▶ **Gán  $RT(X_{i+1}) = 0$**

▶ **Gán  $WT(X_{i+1}) = TS(T)$**

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





## Nội dung chi tiết

✱ *Các vấn đề của Truy xuất đồng thời*

✱ *Kỹ thuật khóa*

- ❖ Khóa đơn giản
- ❖ Khóa đọc ghi
- ❖ Khóa đa hạt

✱ *Kỹ thuật nhãn thời gian*

- ❖ Nhãn thời gian toàn phần
- ❖ Nhãn thời gian riêng phần
- ❖ Nhãn thời gian nhiều phiên bản

✱ *Kỹ thuật lạc quan*

✱ *Vấn đề khóa chết*

✱ *Các vấn đề khác*

## **\* Ý tưởng**

- ❖ Cho phép các giao tác truy xuất dữ liệu 1 cách tự do
- ❖ Kiểm tra tính khả tuần tự của các giao tác
  - ⊛ Trước khi ghi, thực hiện bước Kiểm tra hợp lệ
    - ▶ tập hợp các đơn vị dữ liệu của 1 giao tác sẽ được so sánh với tập đơn vị dữ liệu của những giao tác khác
  - ⊛ Nếu không hợp lệ, sẽ có giao tác phải rollback

## **Kỹ thuật lạc quan**

### **\* Một giao tác có 3 giai đoạn**

#### **❖ (1) Đọc vào Read set - RS(T)**

- ⊛ Đọc tất cả các đơn vị dữ liệu có dùng đến trong giao tác
- ⊛ Tính toán rồi lưu trữ vào bộ nhớ phụ gọi là Read set
- ⊛ Không sử dụng cơ chế khóa

#### **❖ (2) Kiểm tra hợp lệ - Validate**

- ⊛ Kiểm tra tính khả tuần tự

#### **❖ (3) Ghi từ Write set - WS(T)**

- ⊛ Nếu (2) hợp lệ thì ghi xuống CSDL, tập hợp các dữ liệu sẽ ghi gọi là Write set

**\* Trong lịch phát sinh, nếu  $T_1, T_2, \dots, T_n$  là thứ tự hợp lệ thì kết quả sẽ tương đương lịch tuần tự  $\{T_1, T_2, \dots, T_n\}$**

## **\* Bộ lập lịch xem xét 3 tập hợp**

### **❖ START**

- ⊛ Tập các giao tác đã bắt đầu nhưng chưa kiểm tra hợp lệ xong
- ⊛ START(T) ghi nhận thời điểm bắt đầu của T

### **❖ VAL**

- ⊛ Tập các giao tác được kiểm tra hợp lệ nhưng chưa hoàn tất ghi
  - ▶ *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

## **Kỹ thuật lạc quan (tt)**

### **\* Vấn đề 1**

- ❖ *T đã kiểm tra hợp lệ xong và bắt đầu ghi WS(T)***
- ❖ *T chưa hoàn tất ghi thì U lại bắt đầu đọc***
- ❖ *Nếu  $RS(U) \cap WS(T) \neq \emptyset$*** 
  - ⊛ *U có thể không đọc được giá trị X mà T chưa ghi kịp*
  - ⊛ *U lẽ ra phải bắt đầu khi T đã hoàn tất việc ghi*
- ❖ *Quyết định : Rollback U***

## ※ Vấn đề 2

- ❖ *T đã kiểm tra hợp lệ xong và bắt đầu ghi WS(T)*
- ❖ *T chưa hoàn tất ghi thì U bắt đầu kiểm tra hợp lệ*
- ❖ *Nếu  $WS(U) \cap WS(T) = \{X\}$*

⊛ *Nghĩa là cả T và U đều sẽ ghi X. Nếu tuân tự thì T ghi X trước và U ghi X sau (vì T đã kiểm tra hợp lệ xong trước)*

⊛ *Nhưng U có thể kiểm tra hợp lệ rất nhanh và tiến hành ghi trước khi T hoàn tất ghi, và có thể U sẽ ghi X trước khi T kịp ghi X*

⊛ *U lẽ ra phải kiểm tra hợp lệ sau khi T đã hoàn tất ghi*

- ❖ **Quyết định : Rollback U**

## Kỹ thuật lạc quan (tt)

### ※ Qui tắc : từ vấn đề 1 và vấn đề 2, ta có quy tắc

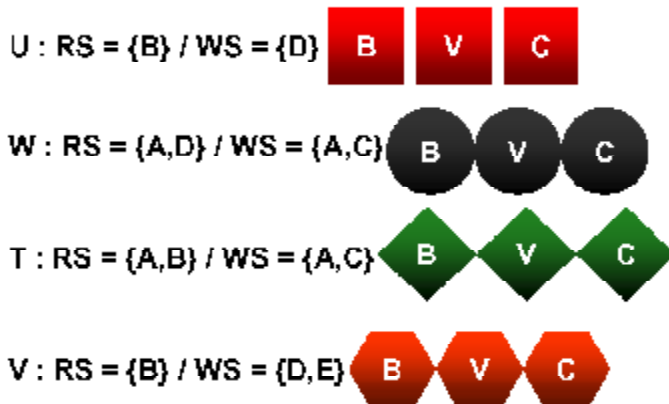
- ❖ (1) - *Nếu có T đã kiểm tra hợp lệ xong nhưng chưa ghi hoàn tất mà U bắt đầu*

⊛ *Kiểm tra nếu  $RS(U) \cap WS(T) \neq \emptyset$  thì hủy U*

- ❖ (2) - *Nếu có T đã kiểm tra hợp lệ xong nhưng ghi chưa hoàn tất mà U kiểm tra hợp lệ*

⊛ *Kiểm tra nếu  $WS(U) \cap WS(T) \neq \emptyset$  thì hủy U*

### ※ Hai quy tắc trên được áp dụng khi U kiểm tra hợp lệ



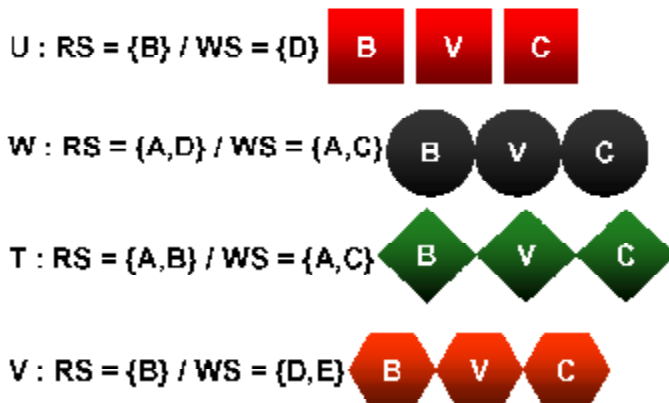
Khi U kiểm tra hợp lệ, U thấy :

Khi U bắt đầu và khi U kiểm tra hợp lệ, không có giao tác nào kiểm tra hợp lệ xong trước đó

→ U kiểm tra hợp lệ thành công và ghi D



## Ví dụ



Khi T kiểm tra hợp lệ, T thấy :

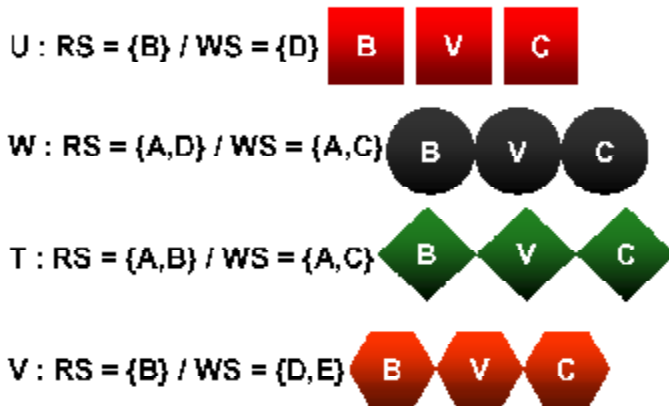
- Khi T bắt đầu, chưa có giao tác nào kiểm tra hợp lệ xong

- Khi T kiểm tra hợp lệ, U đã kiểm tra hợp lệ xong nhưng chưa hoàn tất nên kiểm tra WS(U) và RS(T) thấy không giao

→ T kiểm tra hợp lệ thành công và ghi A, C





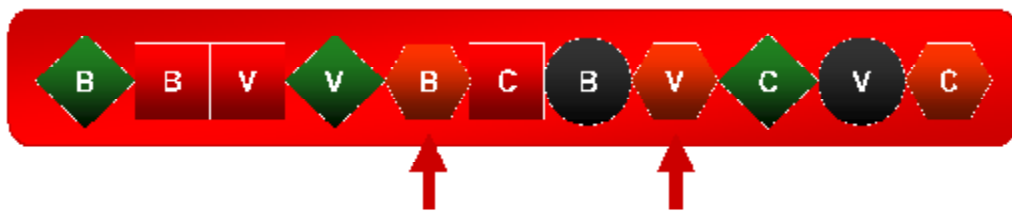


Khi V kiểm tra hợp lệ, V thấy :

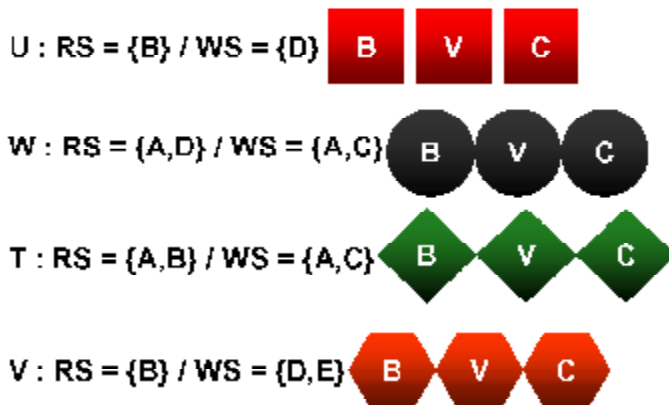
- V bắt đầu khi U đã kiểm tra hợp lệ nhưng chưa hoàn tất nên kiểm tra RS(V) và WS(U) thấy không giao

- V kiểm tra hợp lệ khi T kiểm tra hợp lệ xong nhưng chưa hoàn tất nên kiểm tra WS(T) và WS(V) thấy không giao

→ V kiểm tra hợp lệ thành công và ghi A, C

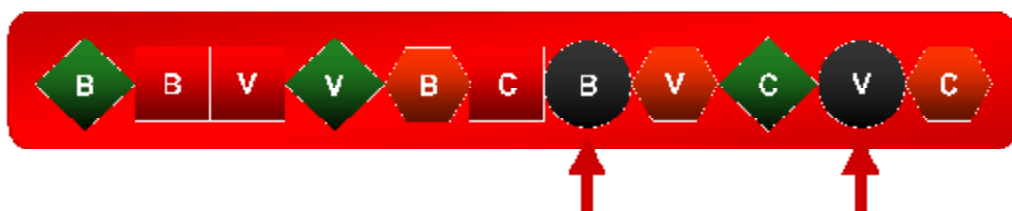


## Ví dụ



Khi W kiểm tra hợp lệ, W thấy :

Khi W bắt đầu, T đã kiểm tra hợp lệ xong nhưng chưa ghi, xét RS(w) và WS(T) thấy giao ở A → W tự hủy



### ✳ *Kỹ thuật khóa*

- ❖ Ưu : Không gây tình trạng rollback
- ❖ Khuyết : Có thể gây chờ vĩnh viễn (Deadlock)

### ✳ *Kỹ thuật Nhãn thời gian*

- ❖ Ưu : Không gây deadlock
- ❖ Khuyết :
  - ⊛ Tồn bộ nhớ lưu nhãn thời gian (và các phiên bản khác nhau với nhãn thời gian nhiều phiên bản)
  - ⊛ Xử lý rollback không đơn giản (có thể gây rollback dây chuyền)

### ✳ *Kỹ thuật Optimistic*

- ❖ Ưu : Xử lý đơn giản (phép giao tập hợp)
- ❖ Khuyết :
  - ⊛ Tồn không gian lưu RS và WS
  - ⊛ Rollback quá nhiều → Giả thiết lạc quan (thống kê thực tế)

## Nội dung chi tiết

### ✳ *Các vấn đề của Truy xuất đồng thời*

#### ✳ *Kỹ thuật khóa*

- ❖ Khóa đơn giản
- ❖ Khóa đọc ghi
- ❖ Khóa đa hạt

#### ✳ *Kỹ thuật nhãn thời gian*

- ❖ Nhãn thời gian toàn phần
- ❖ Nhãn thời gian riêng phần
- ❖ Nhãn thời gian nhiều phiên bản

#### ✳ *Kỹ thuật lạc quan*

#### ✳ *Vấn đề khóa chết*

#### ✳ *Các vấn đề khác*

## ※ **Khái niệm**

- ❖ Khi xử lý đồng thời, không tránh khỏi việc transaction này phải chờ đợi transaction khác
- ❖ Nếu vì lý do gì đó mà hai transaction lại chờ lẫn nhau vĩnh viễn, không cái nào trong hai có thể hoàn thành được thì ta gọi đó là hiện tượng Dead Lock

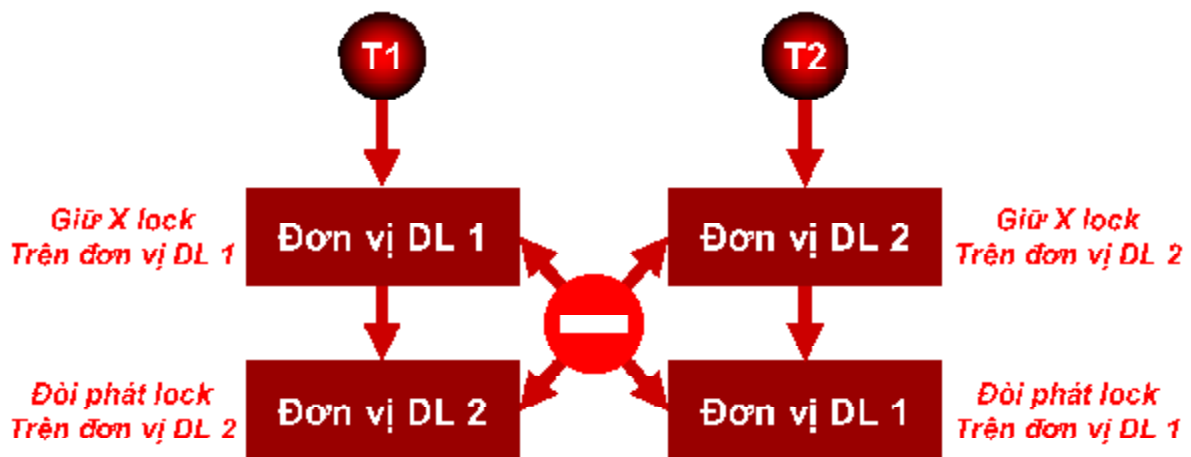
## ※ **Phân loại**

- ❖ Cyclic Deadlock
- ❖ Conversion Deadlock

## Dead lock (tt)

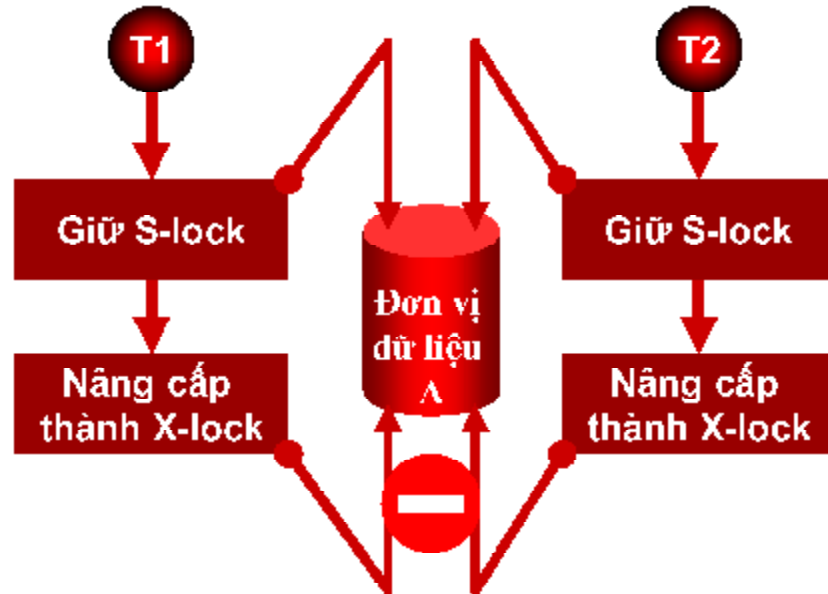
## ※ **Phân loại**

- ❖ Cyclic Deadlock



## \* **Phân loại**

### ❖ **Conversion Deadlock**



## Giải quyết Deadlock

### \* **Phát hiện**

#### ❖ Cho phép trạng thái deadlock xảy ra và sau đó cố gắng khôi phục lại hệ thống

- ✧ Chọn 1 giao tác để rollback

#### ❖ Phương pháp

- ✧ Đồ thị chờ (wait-for graph)

### \* **Ngăn ngừa**

#### ❖ Quản lý các giao tác sao cho không bao giờ có deadlock

#### ❖ Phương pháp

- ✧ Sắp thứ tự tài nguyên (resource ordering)
- ✧ Timeout
- ✧ Wait-die
- ✧ Wound-wait

## \* **Đồ thị gồm**

- ❖ **Đỉnh** là các giao tác đang giữ khóa hoặc đang chờ khóa
- ❖ **Cung đi từ đỉnh T sang U khi**
  - ⊛ U đang giữ khóa trên đơn vị dữ liệu A
  - ⊛ T đang chờ khóa trên A
  - ⊛ T không thể khóa đơn vị dữ liệu A nếu U không giải phóng khóa

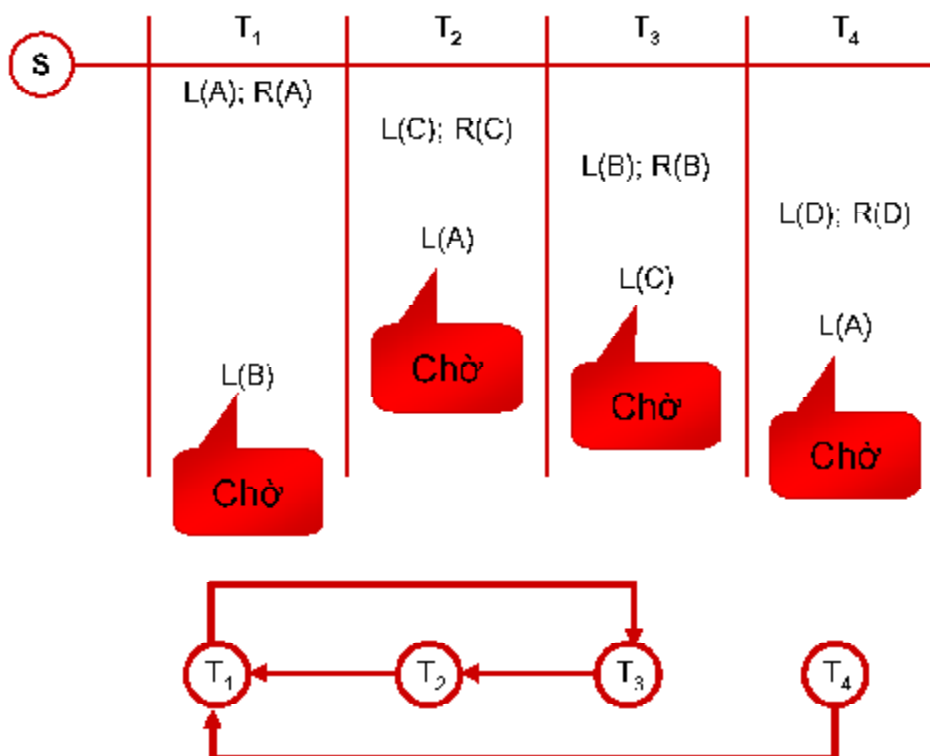
## \* **Nếu đồ thị chờ không có chu trình**

- ❖ Các giao tác có thể hoàn tất

## \* **Ngược lại**

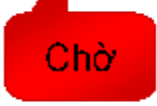

- ❖ Không một giao tác nào trong chu trình có thể tiếp tục thực hiện → **deadlock**

## Ví dụ



- \* **Áp đặt một thứ tự nào đó lên các đơn vị dữ liệu**
  - ❖ Nếu các giao tác thực hiện khóa những đơn vị dữ liệu theo thứ tự này
  - ❖ Thì không có deadlock xảy ra trong khi chờ đợi
- \* **Ví dụ : Giả sử quy ước các đơn vị dữ liệu phải được khóa theo thứ tự Alphabet**
- \* **Các giao tác cần phát khóa như sau :**
  - ❖ T1 : l(A); r(A); l(B); w(B); u(A); u(B);
  - ❖ T2 : l(A); l(C); r(C); w(A); u(C); u(A);
  - ❖ T3 : l(B); r(B); l(C); w(C); u(B); u(C);
  - ❖ T4 : l(A); l(D); r(D); w(A); u(D); u(A);
- \* **Hãy vẽ đồ thị chờ cho lịch sau và nhận xét**

## Ví dụ (tt)

T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>
L(A); R(A)	l (A)	L(B); R(B)	L(A)
		L(C); W(C)	
L(B); R(B)		U(B); U(C)	
U(A); U(B)	L(A); L(C)		
	R(A); W(C)		
	U(C); U(A)		
			L(A); L(D)
			R(D); W(A)
			U(D); U(A)

### \* Quy tắc :

- ❖ Giới hạn các giao tác chỉ được thực hiện trong 1 khoảng thời gian nào đó
- ❖ Nếu giao tác vượt quá thời gian này
- ❖ Thì giao tác phải bị rollback

## Wait-die

### \* Quy tắc

- ❖ Mỗi giao tác sẽ được gán một nhãn ghi nhận thứ tự xuất hiện, kí hiệu :  $ts(T)$
- ❖ Xét 2 giao tác T và U
  - ⊛ U đang giữ khóa trên đơn vị dữ liệu A
  - ⊛ T muốn khóa đơn vị dữ liệu A
  - ⊛ Nếu  $ts(T) < ts(U)$  thì T sẽ chờ (wait) U
  - ⊛ Ngược lại T sẽ bị hủy (die) và bắt đầu làm lại ở 1 thời điểm khác

### ※ **Quy tắc :**

- ❖ Mỗi giao tác sẽ được gán một nhãn ghi nhận thứ tự xuất hiện, kí hiệu:  $ts(T)$
- ❖ Xét 2 giao tác T và U
  - ⊛ U đang giữ khóa trên đơn vị dữ liệu A
  - ⊛ T muốn khóa đơn vị dữ liệu A
  - ⊛ Nếu  $ts(T) < ts(U)$  thì T buộc U rollback và trao khóa lại cho T (wound)
  - ⊛ Ngược lại T sẽ chờ (wait) U

## Nhận xét

### ※ **Timeout**

- ❖ Đơn giản
- ❖ Khó chọn được khoảng thời gian timeout thích hợp
- ❖ Có hiện tượng starvation (Giao tác lập đi lập lại quá trình: bắt đầu, deadlock, rollback)

### ※ **Resource ordering**

- ❖ Không thực tế
- ❖ Chờ đợi nhiều → tiềm ẩn của deadlock



### ✱ *Wait-die và Wound-wait*

- ❖ Không có Starvation
- ❖ Wound-wait ít rollback các giao tác hơn wait-die
- ❖ Dễ cài đặt hơn Đồ thị chờ
- ❖ Có thể rollback những giao tác không gây ra deadlock

### ✱ *Đồ thị chờ*

- ❖ Nếu đồ thị quá lớn sẽ tốn nhiều thời gian phân tích
- ❖ Rất phức tạp khi CSDL phân tán
- ❖ Giảm tối thiểu rollback các giao tác (Chỉ rollback 1 trong những giao tác gây ra deadlock)

## Nội dung chi tiết

### ✱ *Các vấn đề của Truy xuất đồng thời*

#### ✱ *Kỹ thuật khóa*

- ❖ Khóa đơn giản
- ❖ Khóa đọc ghi
- ❖ Khóa đa hạt

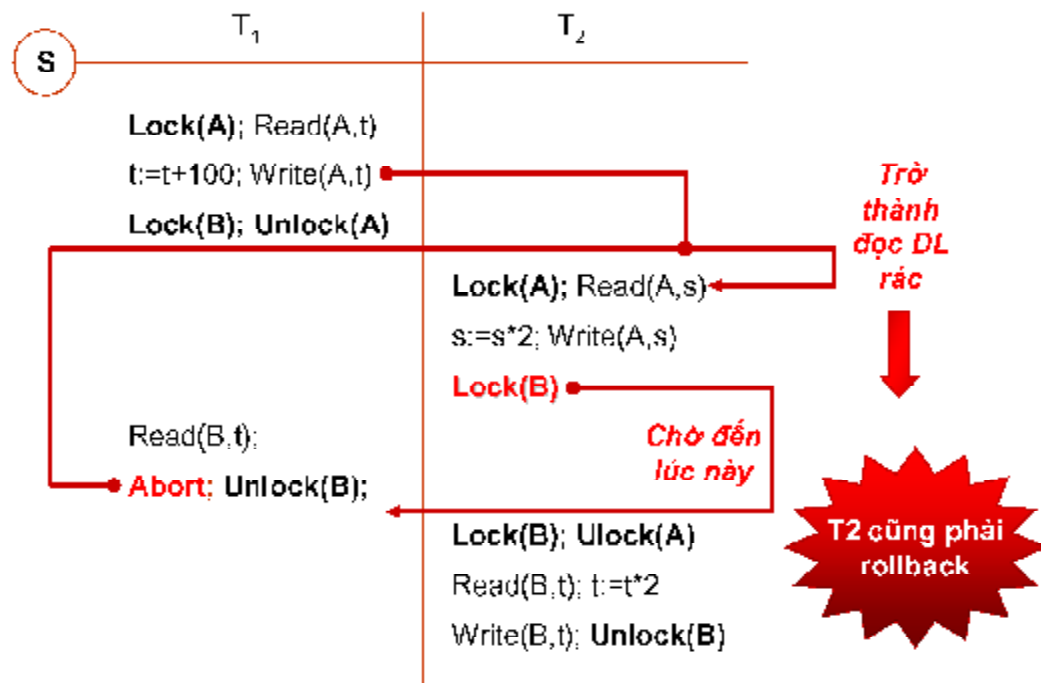
#### ✱ *Kỹ thuật nhãn thời gian*

- ❖ Nhãn thời gian toàn phần
- ❖ Nhãn thời gian riêng phần
- ❖ Nhãn thời gian nhiều phiên bản

#### ✱ *Kỹ thuật lạc quan*

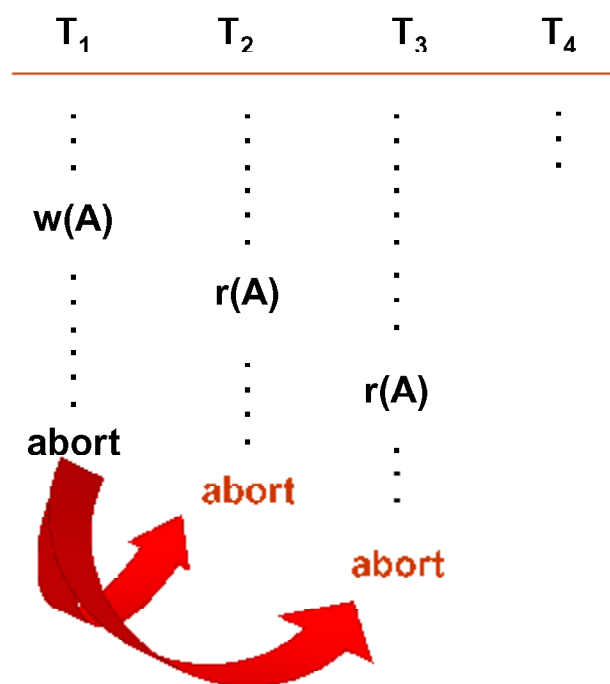
#### ✱ *Vấn đề khóa chết*

#### ✱ *Các vấn đề khác*



## Quay lui dây chuyền

✳ *Một cách tổng quát*



### \* **Khái niệm :**

❖ Lịch khả phục hồi là một lịch thao tác  $S$  mà khi xét một cặp giao tác  $T_i$  và  $T_j$  bất kỳ trong  $S$  :

- ⊗ Nếu  $T_j$  đọc dữ liệu sau khi  $T_i$  ghi
- ⊗ Thì  $T_i$  phải được hoàn tất (commit -  $c_i$ ) trước khi  $T_j$  hoàn tất

### \* **Ví dụ :**



❖  $S1 : w1(A); w1(B); w2(A); r2(B); c1; c2;$

- ⊗  $S1$  khả tuần tự và khả phục hồi

❖  $S2 : w2(A); w1(B); w1(A); r2(B); c1; c2;$

- ⊗  $S2$  không khả tuần tự nhưng khả phục hồi



❖  $S3 : w1(A); w1(B); w2(A); r2(B); c2; c1;$

- ⊗  $S3$  khả tuần tự nhưng không khả phục hồi

## Lịch không quay lui dây chuyền

### \* **Khái niệm :**

❖ Lịch không quay lui dây chuyền (cascadeless schedule) là lịch mà trong đó các giao tác chỉ đọc những đơn vị dữ liệu  $X$  mà giao tác sau cùng ghi  $X$  đã commit

\* **Nhận xét :** Các lịch không quay lui dây chuyền đều là lịch khả phục hồi

### \* **Ví dụ :**

❖  $S1 : w1(A); w1(B); w2(A); c1; r2(B); c2;$

Vì  $c1$  trước  $r2(B)$ , mà  $r2(B)$  chắc chắn trước  $c2 \rightarrow c1$  chắc chắn trước  $c2$

