Nguyên lý Hệ quản trị CSDL

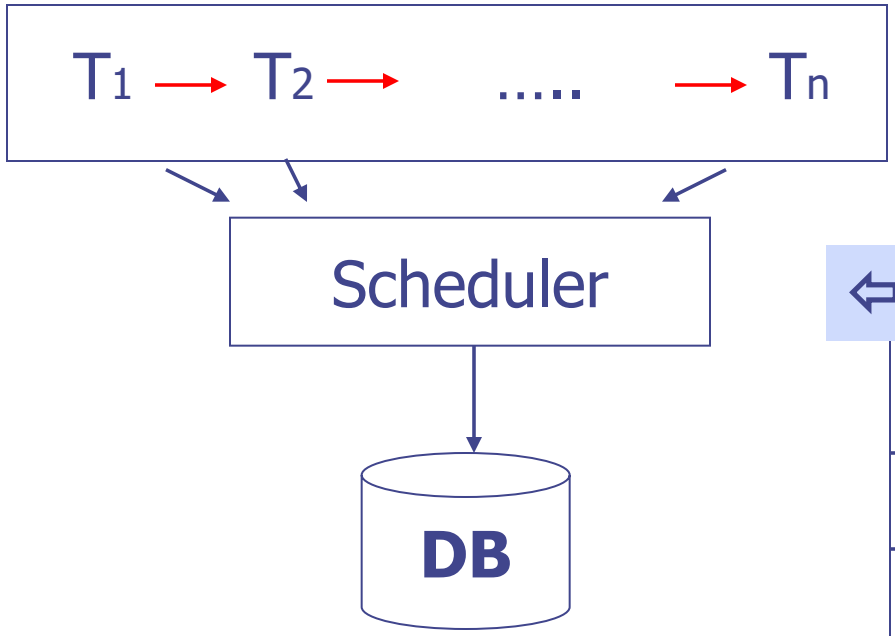
Chương 4: Điều khiển cạnh tranh (Concurency Control)

Tóm tắt nội dung

- Giới thiệu
- Giao thức dựa trên chốt
  - Chốt và cấp chốt
  - Giao thức chốt 2 kỳ
  - Giao thức dựa trên đồ thị
  - Đa hạt
- Giao thức dựa trên tem thời gian
  - Giao thức thứ tự tem thời gian
  - Giao thức viết Thomas
- Giao thức dựa trên tính hợp lệ
- Quản lý deadlock
  - Phòng ngừa deadlock
  - Sơ đồ dựa trên timeout
  - Phát hiện và khôi phục deadlock

Giới thiệu

- Trong chương trước, ta đã có các giải thuật để xác định một lịch trình có khả tuần tự hay không.
- **Nhưng làm thế nào để sinh ra được các lịch trình khả tuần tự?**



⇦ **Các giao thức**

- Dựa trên chốt
- Dựa trên tem thời gian
- Dựa trên tính hợp lệ

Giao thức dựa trên chốt

- Dựa trên sự loại trừ hỗ tương: khi 1 GD đang truy xuất một hạng mục DL, không một GD nào khác được truy xuất hạng mục DL này.

- Phương pháp thực hiện:
  - Mỗi hạng mục DL sẽ có một chốt.
  - Một GD chỉ được truy xuất 1 hạng mục DL nếu nó đang giữ chốt trên hạng mục DL đó

    ⇒ Một GD muốn truy xuất một hạng mục DL phải xin chốt tương ứng trên hạng mục DL đó.

Giao thức dựa trên chốt – Chốt

- Chốt:
  - **S**hare (khóa chia sẻ): chỉ đọc.
  - e**X**clusive (khóa loại trừ): có thể đọc/ghi.
- Các thao tác trên chốt:
  - Xin chốt S trên Q: Lock_S(Q), L_S(Q), LS(Q)
  - Xin chốt X trên Q: Lock_X(Q), L_X(Q), LX(Q)
  - Tháo chốt trên Q: Unlock(Q), U(Q)
- Một GD cần truy xuất hạng mục DL ⇒ xin chốt → được cấp
  → chờ

Requester

Holder	S	X
S	True	False
X	False	False

Ma trận tương thích

Giao thức dựa trên chốt – Chốt

- Sau khi sử dụng xong hạng mục DL thì phải tháo chốt
  - Tháo chốt sớm: cạnh tranh vs. không nhất quán.
  - Tháo chốt trễ: nhất quán vs. ít cạnh tranh + deadlock.

- Ví dụ: Cho 2 lịch trình sau (g/s giá trị A và B trước khi xảy ra các giao dịch lần lượt là 100$ và 200$)

T1
R(B)
B = B – 50
W(B)
R(A)
A = A + 50
W(A)

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

Giao thức dựa trên chốt – Chốt

- Tháo chốt sớm:

T1	T2
L_X(B); R(B) B = B – 50 W(B); U(B)	
	L_S(A); R(A) U(A); L_S(B) R(B); U(B) **Display(A+B)**
L_X(A); R(A) A = A + 50 W(A); U(A)	
Schedule 1	

- Tháo chốt trễ:

T1	T2
L_X(B); R(B) B = B – 50 W(B)	
	L_S(A); R(A) **L_S(B)**; R(B) Display(A+B)
L_X(A); R(A) A = A + 50 W(A) U(A); U(B)	
	U(A); U(B)
Schedule 2	

Giao thức dựa trên chốt – Chốt

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

Giao thức dựa trên chốt – Cấp chốt

- Một GD sẽ được cấp chốt nếu như không có GD nào giữ chốt ở phương thức không tương thích

- Để tránh chết đói, ta có thể dùng giao thức sau:
  - GD $\mathbf{T_i}$ chỉ được cấp chốt trên hạng mục dữ liệu **Q** ở phương thức **M**, khi cả hai điều kiện sau được thỏa:
    - Không có GD khác giữ chốt không tương thích với **M** trên **Q**
    - Không có GD khác đang chờ chốt trên **Q** và đã đưa ra yêu cầu cấp chốt trước $\mathbf{T_i}$

Giao thức dựa trên chốt – Chốt 2 kỳ

- Tính chất:
  - Khả tuần tự xung đột
  - Không tránh được deadlock và cuộn lại hàng loạt
- **Quy tắc**: yêu cầu chốt và tháo chốt của các GD phải chia thành 2 kỳ
  - Kỳ phình to (growing phase): chỉ có thể xin chốt
  - Kỳ thu nhỏ (shrinking phase): chỉ có thể tháo chốt

- Giao thức chốt 2 kỳ **nghiêm ngặt**:
  Chốt X phải được giữ đến cuối GD
- Giao thức chốt 2 kỳ **nghiêm khắc**:
  Tất cả chốt phải được giữ đến cuối GD

} **Tránh cuộn lại hàng loạt**

Giao thức dựa trên chốt – Chốt 2 kỳ

- Ví dụ: **S2** tuân theo GT chốt 2 kỳ còn **S1** thì không

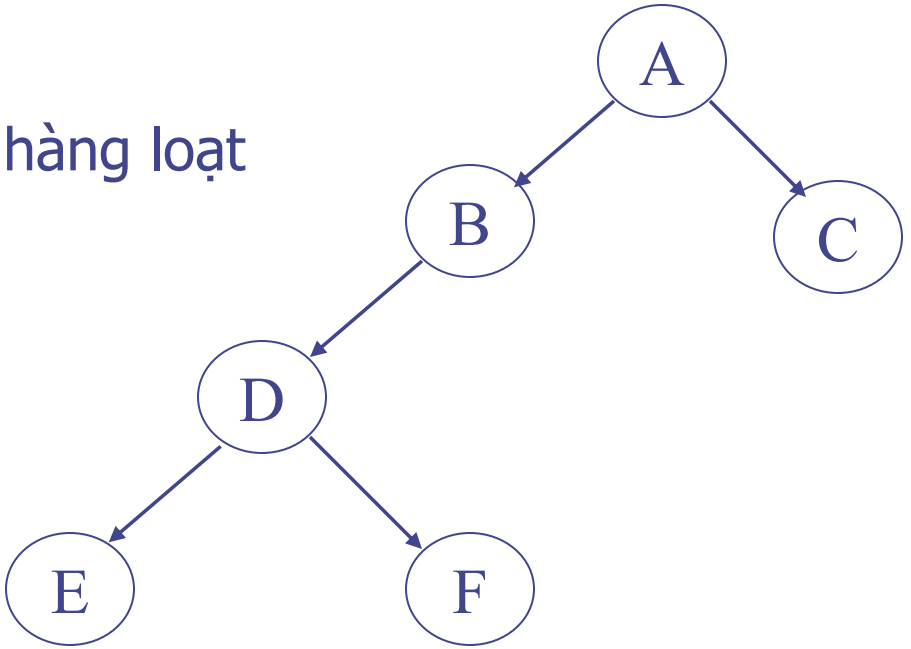
T1	T2
L_X(B); R(B) B = B – 50 W(B); U(B) L_X(A); R(A) A = A + 50 W(A); U(A)	 L_S(A); R(A) U(A); L_S(B) R(B); U(B) Display(A+B)
S1	

T1	T2
L_X(B); R(B) B = B – 50 W(B) **L_X(A)**; R(A) A = A + 50 W(A) U(A); U(B)	 L_S(A); R(A) **L_S(B)**; R(B) Display(A+B) U(A); U(B)
S2	

Giao thức dựa trên chốt – Đồ thị

- Có thể thực hiện nếu biết trước thứ tự mà các hạng mục DL được truy xuất.
  ⇒ Đồ thị CSDL.

- Tính chất:
  - Khả tuần tự xung đột.
  - Tránh deadlock và cuộn lại hàng loạt
  - Chỉ sử dụng chốt Exclusive.



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

Giao thức dựa trên chốt – Đồ thị

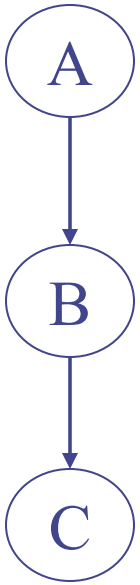
- Quy tắc:
  (1) Mỗi hạng mục có thể chốt nhiều nhất 1 lần
  (2) Chốt đầu tiên bởi T có thể trên bất kỳ hmục DL nào
  (3) Sau đó, Q có thể được chốt bởi T nếu T đã chốt cha Q
  (4) Các hạng mục DL có thể tháo chốt bất kỳ lúc nào

- Giao thức này có thuận lợi hơn giao thức chốt 2 kỳ là ta có thể tháo chốt sớm nhưng nó có hạn chế là tổng chi phí chốt tăng

- Có những GD thỏa giao thức cây nhưng không thỏa giao thức chốt 2 kỳ và ngược lại.

Giao thức dựa trên chốt – Đồ thị

- Ví dụ:



T1	T2
L_X(A)	
L_X(B)	
U(A)	
	L_X(A)
L(C)	
U(B)	
	L_X(B)
	U(A)
	U(B)
U(C)	
Tree, !2PL	

T1	T2
L_X(A)	
	L_X(B)
L(C)	
	U(B)
U(A)	
U(C)	
2PL, !Tree	

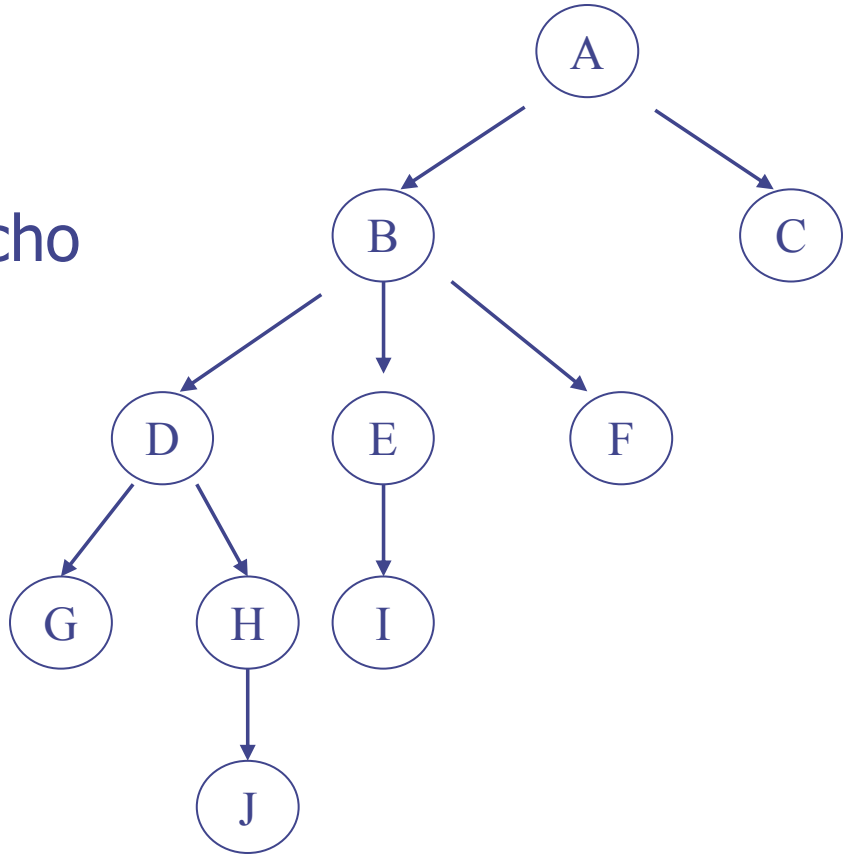
Giao thức dựa trên chốt – Đồ thị

Bài tập

- Cho đồ thị CSDL như sau và hai giao dịch T1 và T2:
  - T1: Truy xuất G, F.
  - T2: Truy xuất F, G.

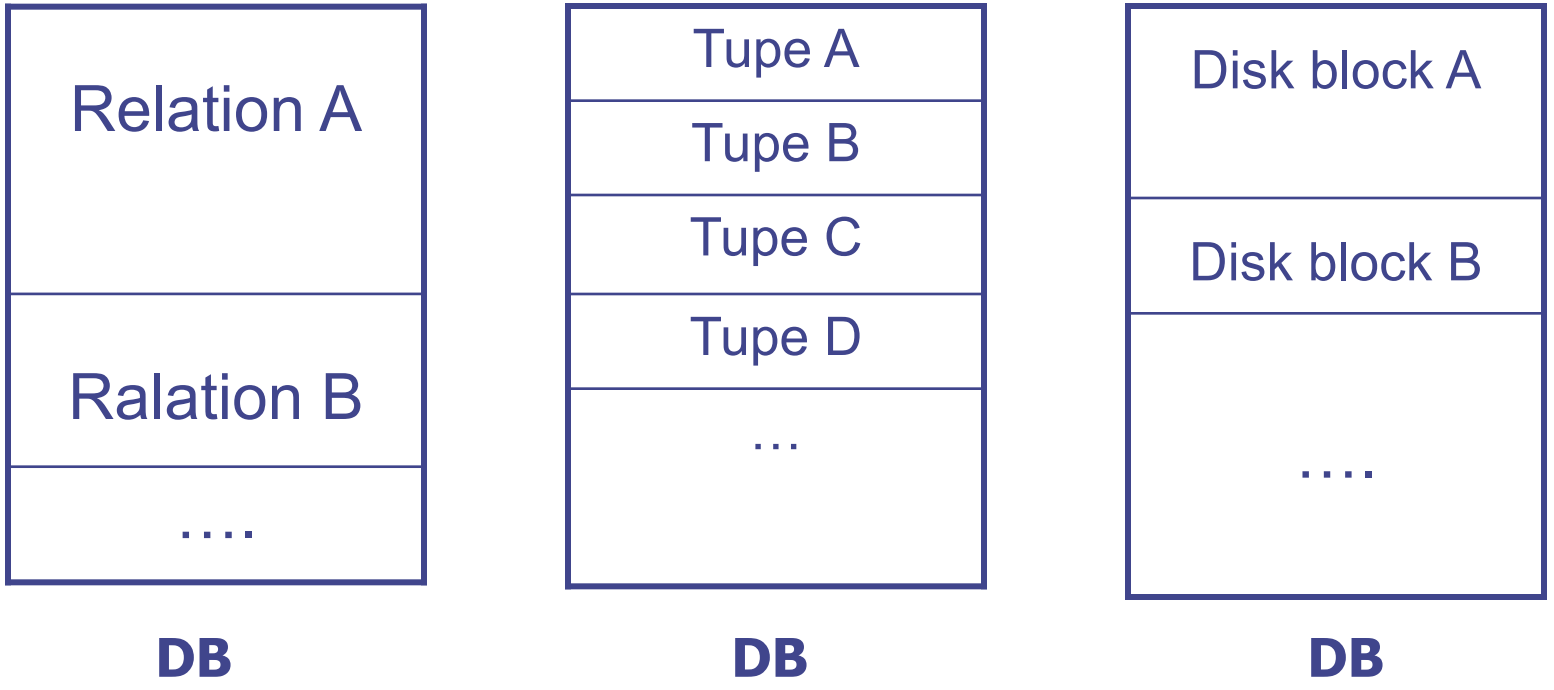
1) Viết các chỉ thị xin/tháo khóa cho các giao dịch trên theo đúng giao thức cây.

2) Tạo một lịch trình cho 2 giao dịch trên và cho biết lịch trình này có bị deadlock hay không?



Giao thức dựa trên chốt – Đa hạt

- Trong các GT dựa trên chốt, một GD phải xin chốt trước khi cập nhật dữ liệu.
- Chúng ta cần chốt những gì?

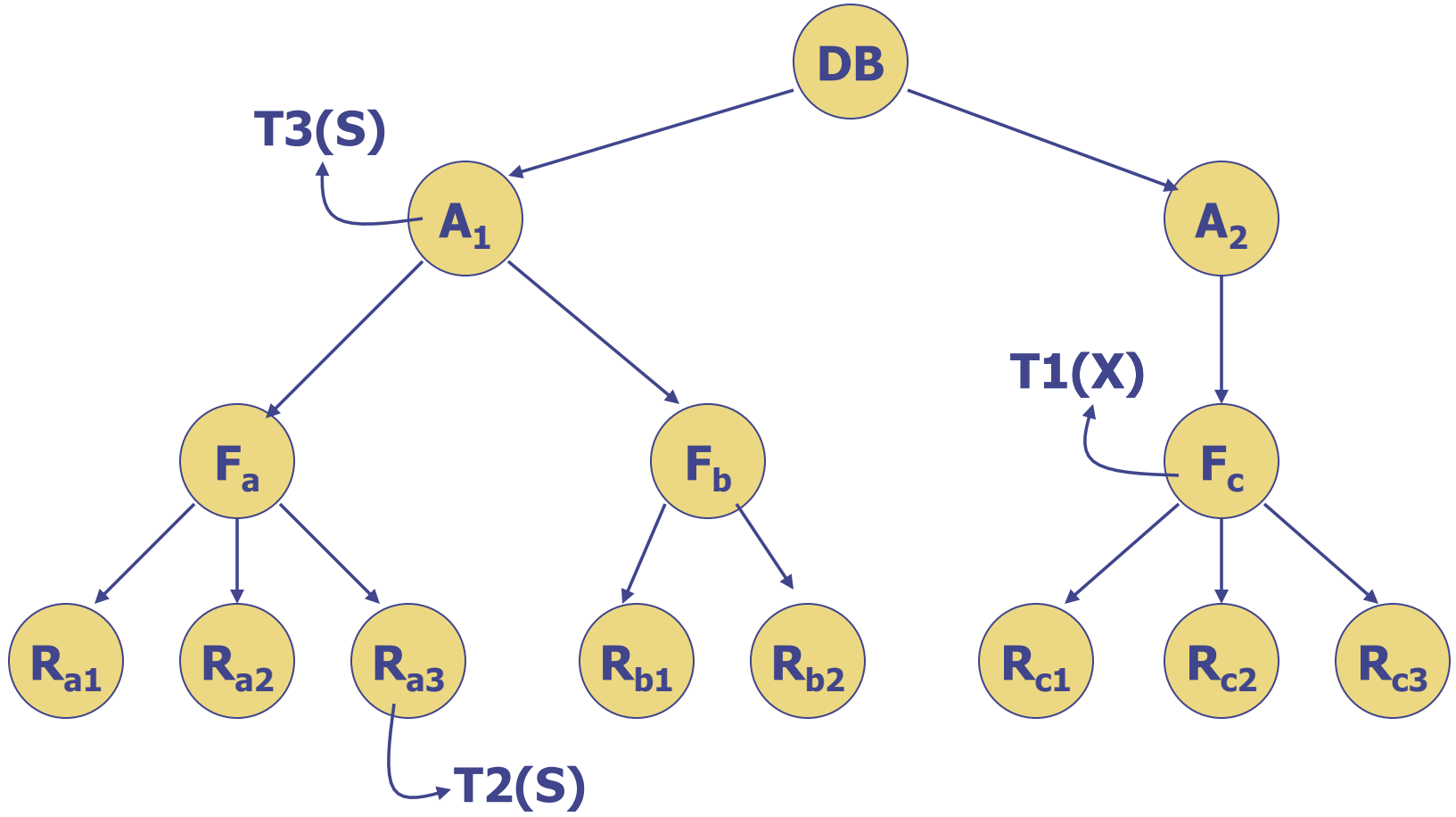


Giao thức dựa trên chốt – Đa hạt

- Nên chốt những đối tượng lớn hay nhỏ?
  - Nếu chốt đối tượng lớn (bảng, DB): cần ít khóa nhưng ít cạnh tranh
  - Nếu chốt đối tượng nhỏ (tupe): cho phí chốt cao nhưng tăng tính cạnh tranh

- Có một giải pháp thực hiện song song cả hai cách: Đa hạt (multiple granularity).

Giao thức dựa trên chốt – Đa hạt



Giao thức dựa trên chốt – Đa hạt

- Để tăng hiệu quả cấp chốt, bổ sung thêm các khóa tăng cường (intensive lock).

Requester

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

Ma trận cạnh tranh khóa

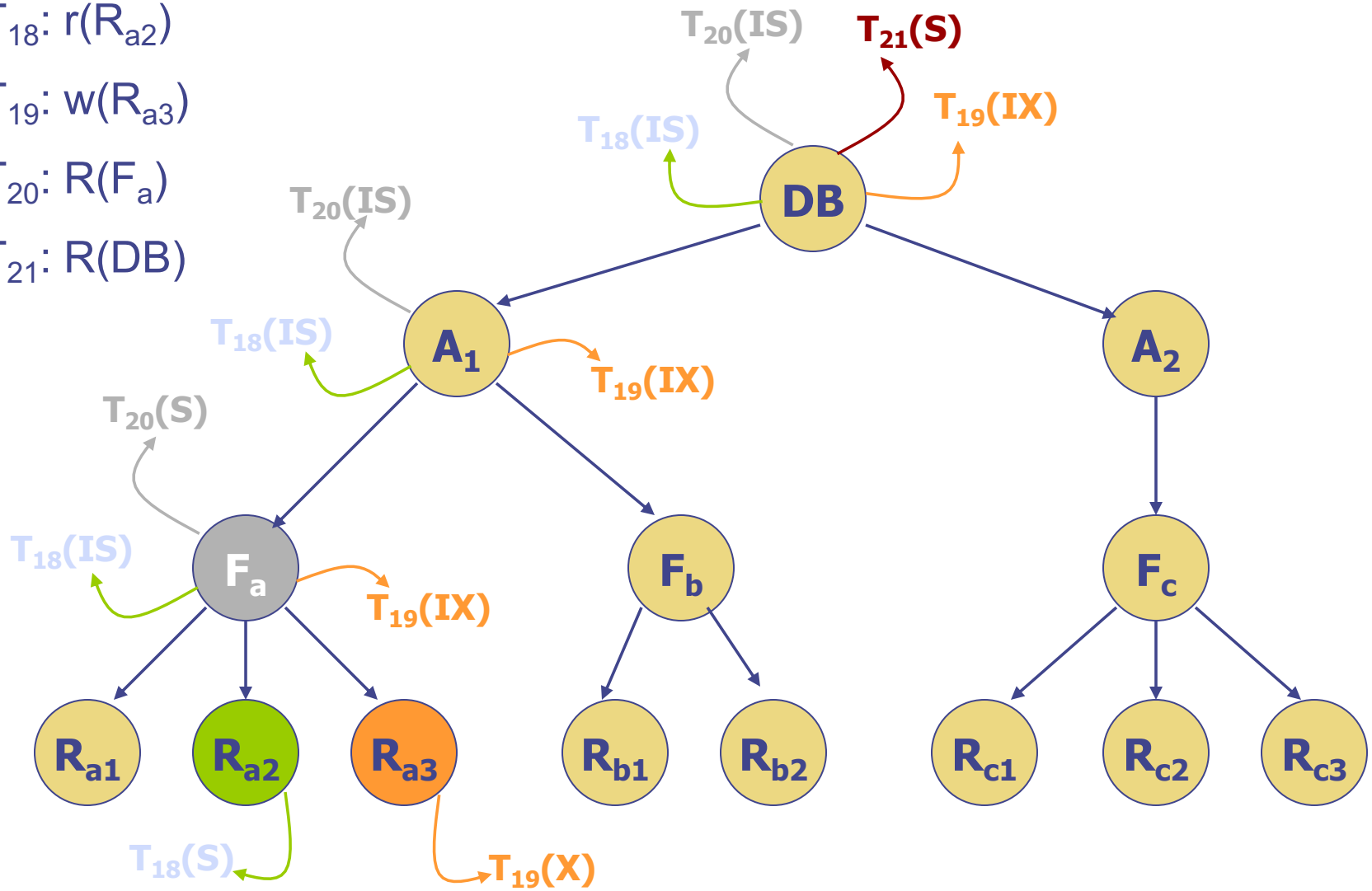
Giao thức dựa trên chốt – Đa hạt

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

⇒ Tậu chốt: Top-Down, tháo chốt: Bottom-Up

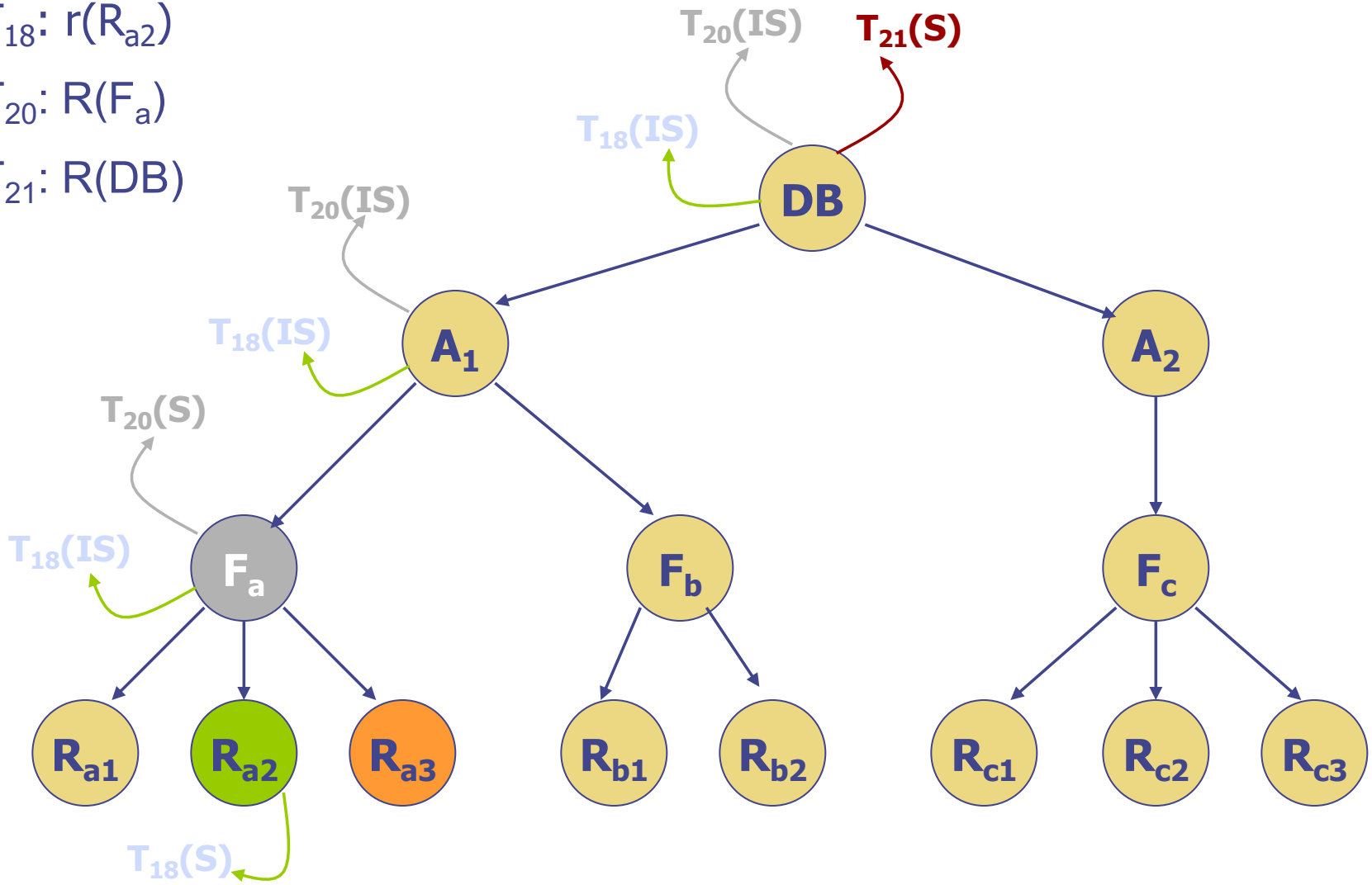
Giao thức dựa trên chốt – Đa hạt

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



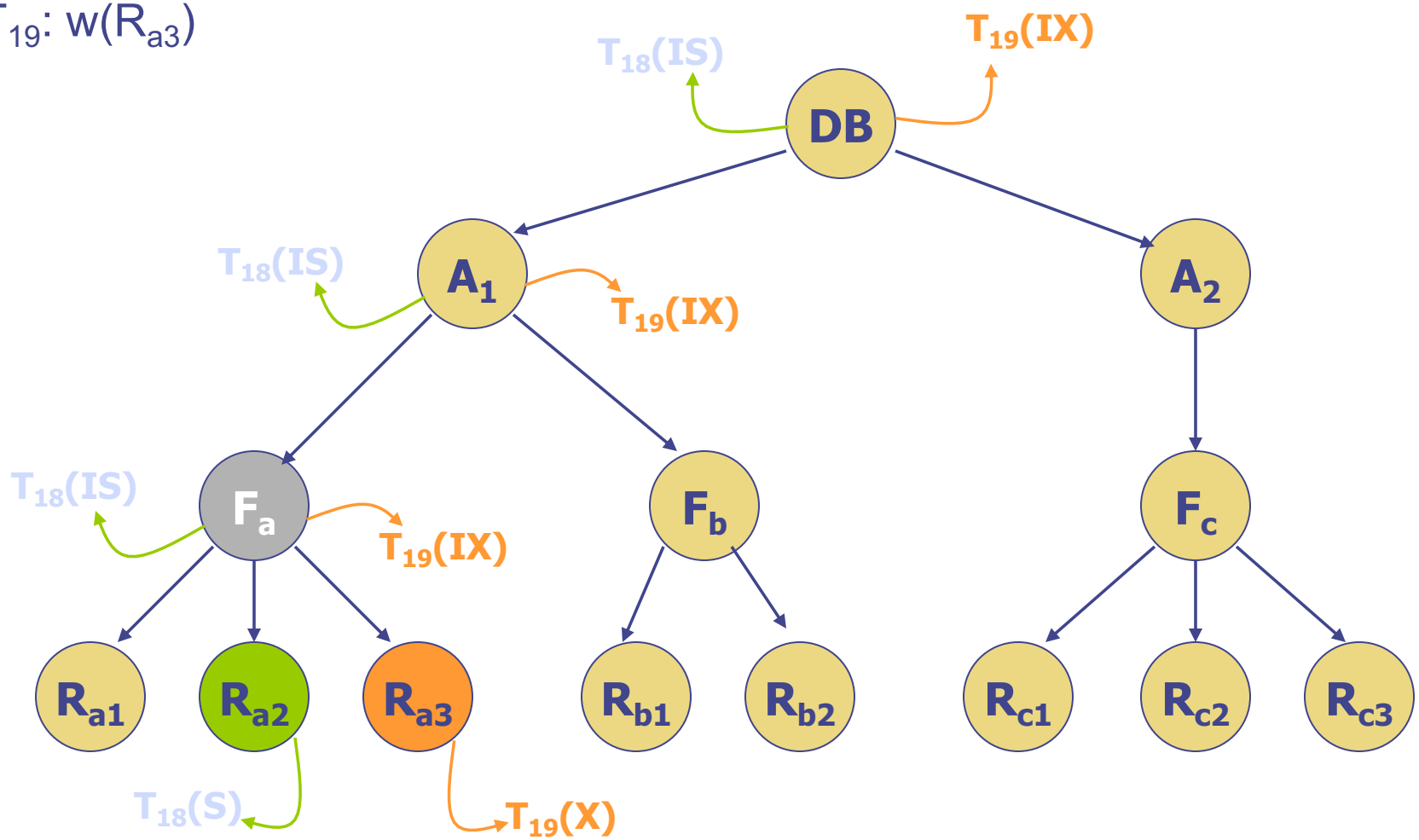
Giao thức dựa trên chốt – Đa hạt

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



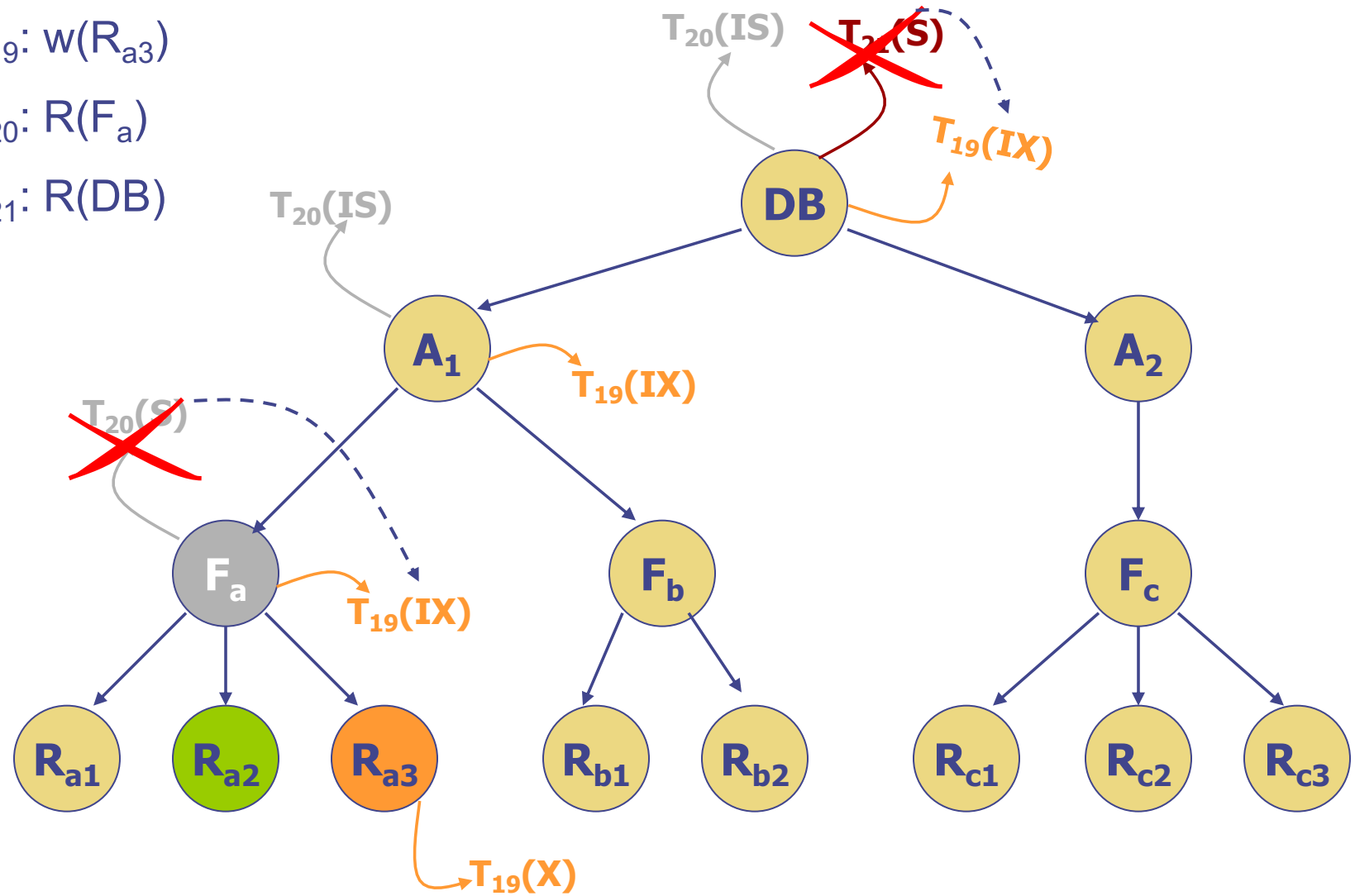
Giao thức dựa trên chốt – Đa hạt

- T_{18}: $r(R_{a2})$
- T_{19}: $w(R_{a3})$



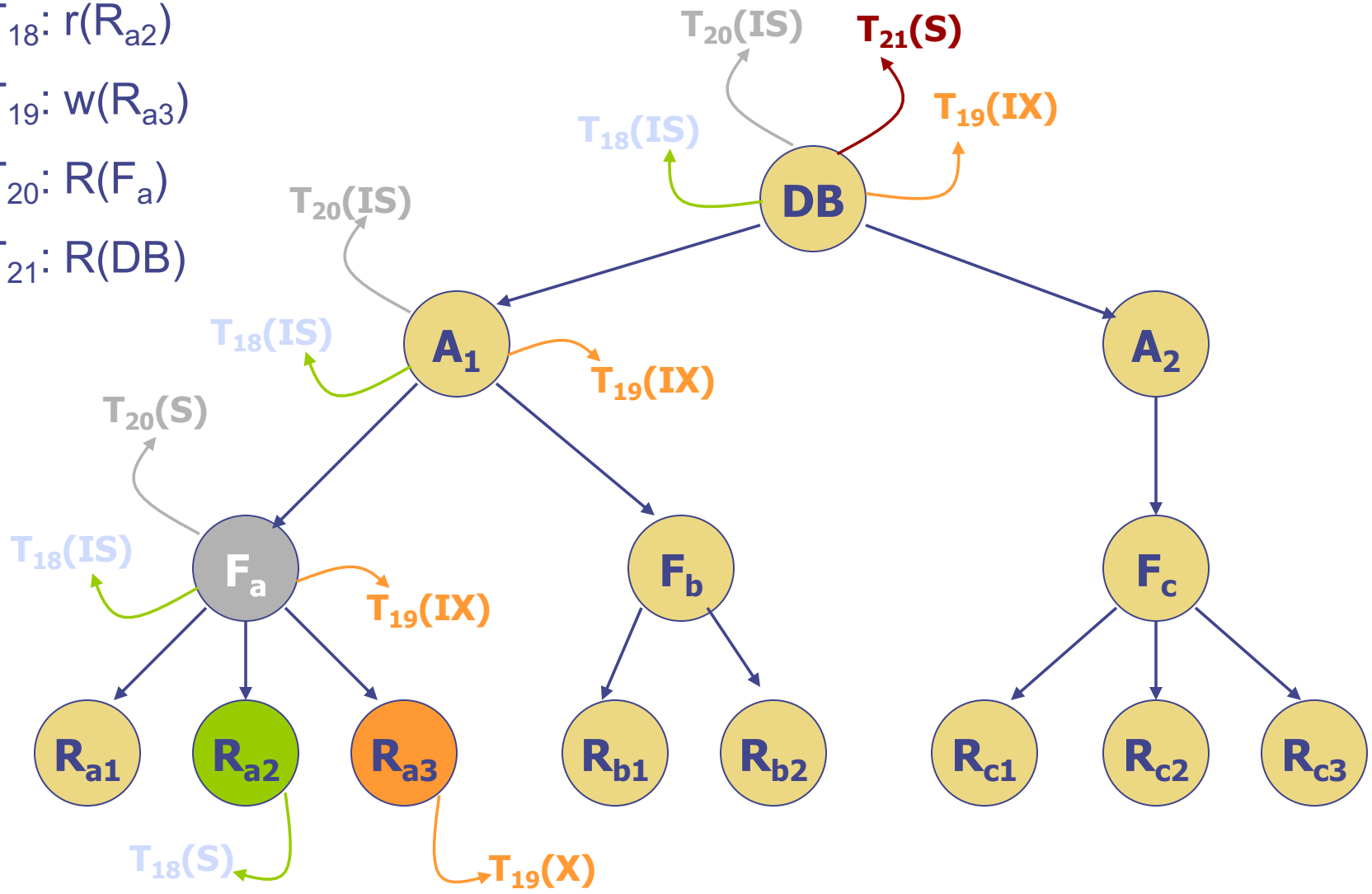
Giao thức dựa trên chốt – Đa hạt

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



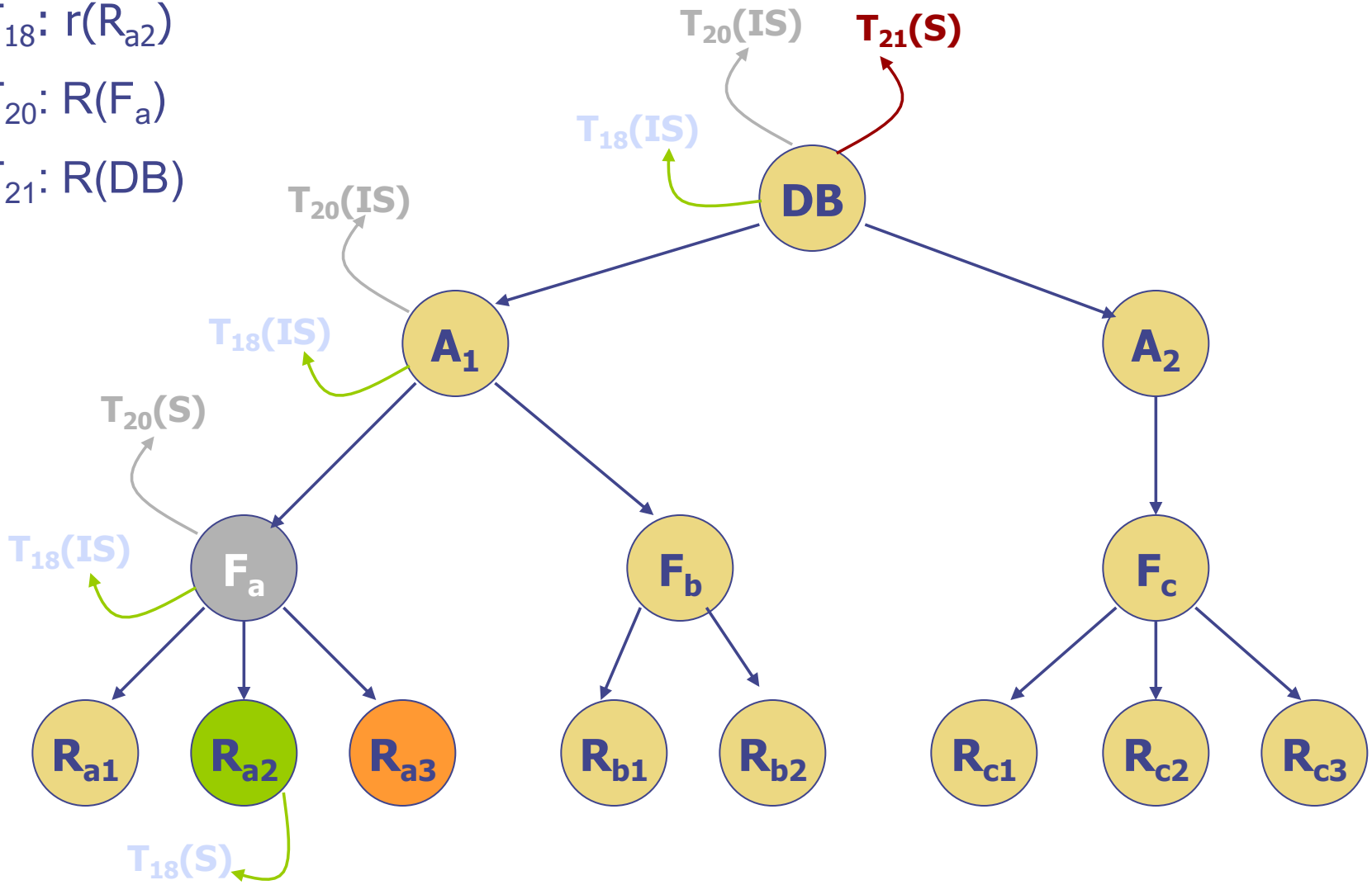
Multiple Granularity

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



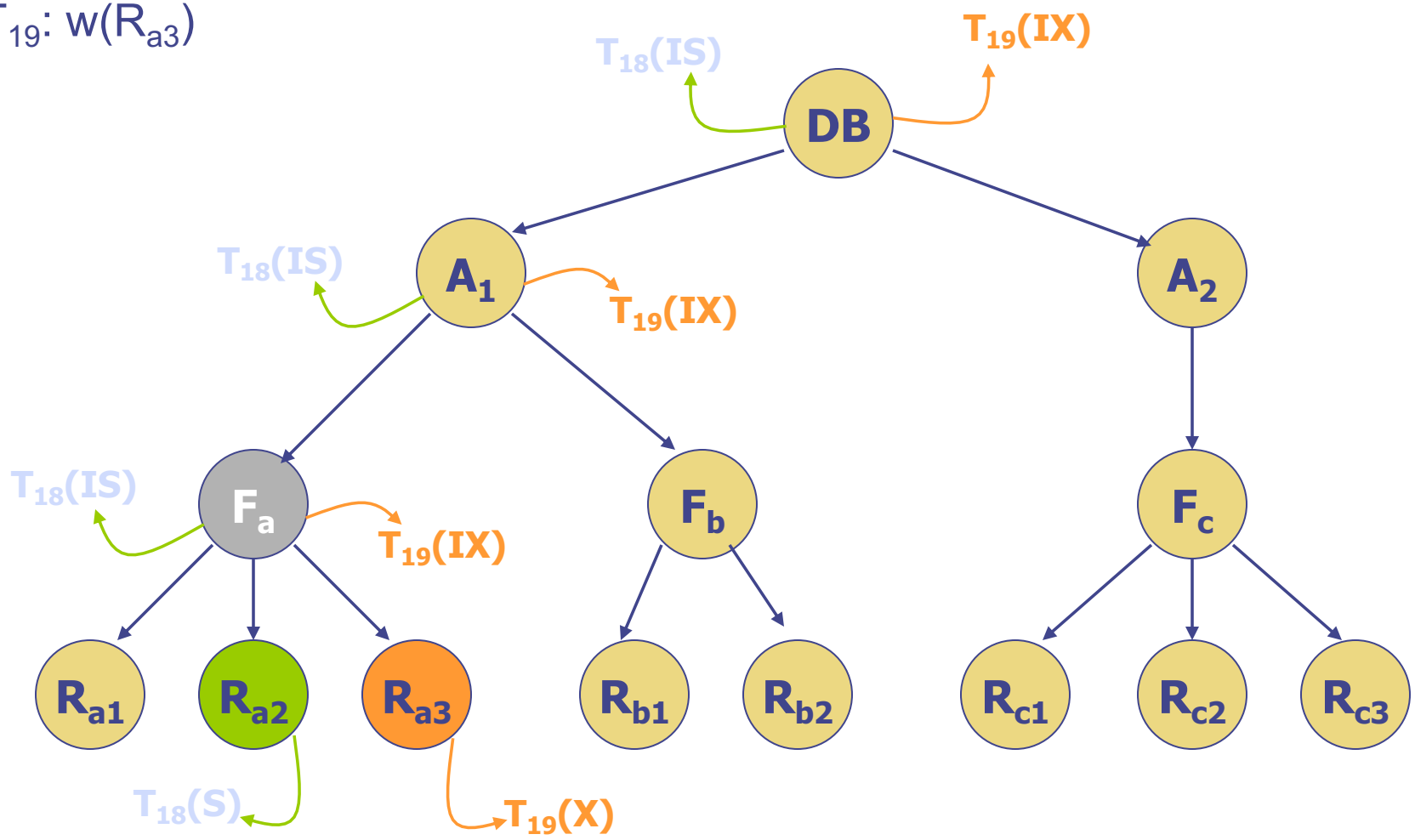
Multiple Granularity

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



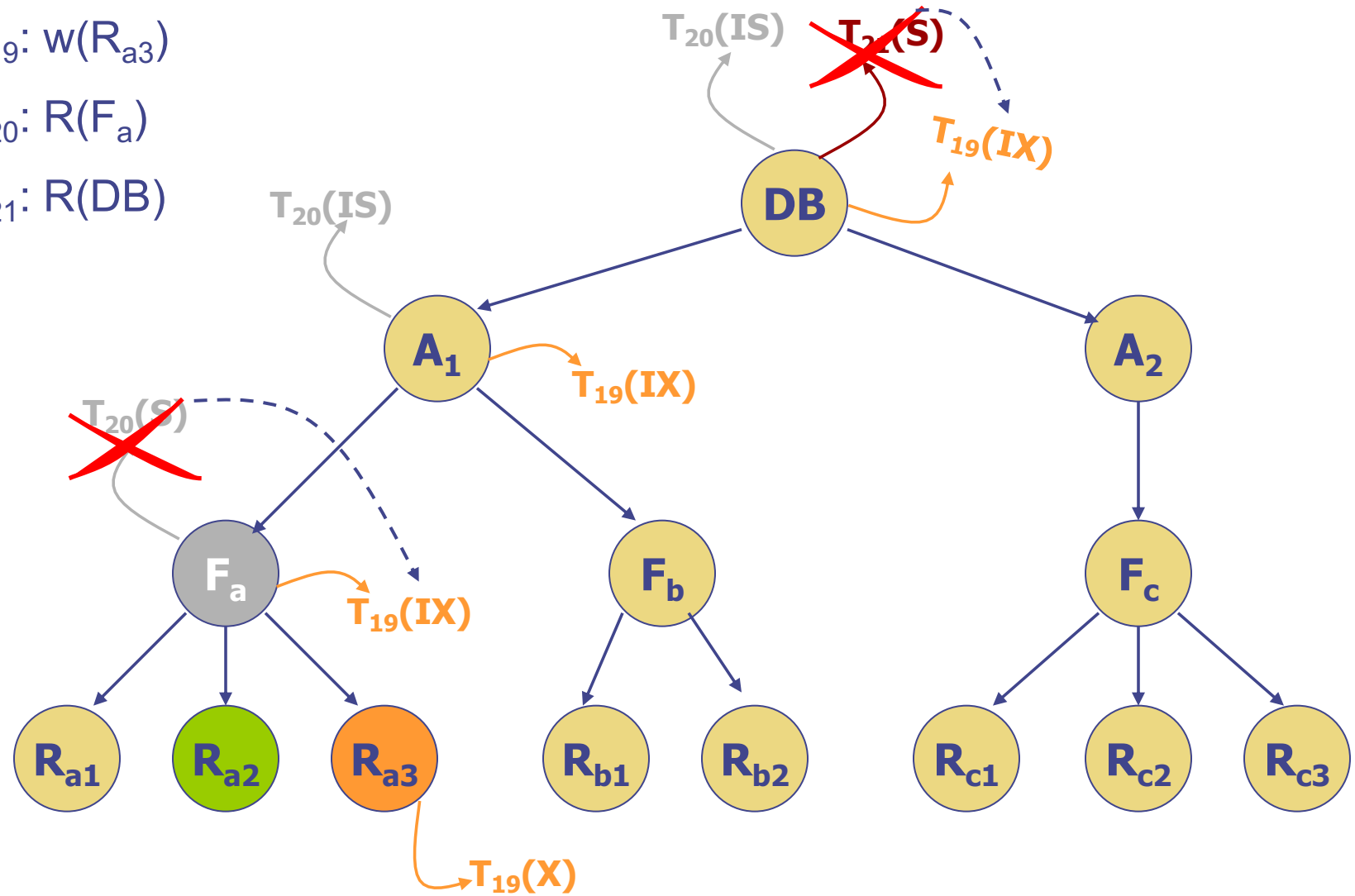
Multiple Granularity

- T_{18}: $r(R_{a2})$
- T_{19}: $w(R_{a3})$



Multiple Granularity

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



Giao thức dựa trên tem thời gian

- Đảm bảo các Read và Write xung đột thực hiện theo thứ tự tem thời gian.
- **Tính chất:** Khả tuần tự xung đột và không deadlock.
- **Thực thi:**
  - Mỗi giao dịch T_i được kết hợp một tem tgian $TS(T_i)$
    (bằng đồng hồ hệ thống hay bộ đếm logic)
  - Nếu $TS(T_i) < TS(T_j)$, lịch trình sinh ra tương đương với một lịch trình tuần tự, trong đó $T_i <_s T_j$
  - Mỗi hạng mục dữ liệu Q được kết hợp 2 tem thời gian:
    - W_Timestamp(Q), ký hiệu W_TS(Q) hoặc WT(Q)
    - R_Timestamp(Q), ký hiệu R_TS(Q) hoặc RT(Q)

    Các tem thời gian này được cập nhật khi giá trị của Q được đọc hoặc ghi.

Giao thức dựa trên tem thời gian

Giao thức tem thời gian:

1. G/s T_i phát ra Read(Q):
   a. Nếu $TS(T_i) < W\_TS(Q)$: vứt bỏ Read(Q) và T_i cuộn lại.
      (giá trị T_i cần đọc đã bị giao dịch khác ghi đè lên)
   b. Ngược lại, thực hiện Read(Q), cập nhật
      $R\_TS(Q) = \max(TS(T_i), R\_TS(Q))$

2. G/s T_i phát ra Write(Q):
   a. Nếu $TS(T_i) < R\_TS(Q)$: vứt bỏ Write(Q) và T_i cuộn lại.
      (giá trị T_i đang ghi là cần thiết trước đó và đã "bị bỏ qua")
   b. Nếu $TS(T_i) < W\_TS(Q)$: vứt bỏ Write(Q) và T_i cuộn lại.
      (T_i đang ghi một giá trị "quá hạn")
   c. Cập nhật $W\_TS(Q) = TS(T_i)$

Giao thức dựa trên tem thời gian

- Ví dụ 1:

T_{14}	T_{15}	A	B
14	**15**	R_TS = 0 W_TS = 0	R_TS = 0 W_TS = 0
R(B)			R_TS = 14
	R(B)		R_TS = 15
	W(B)		W_TS = 15
R(A)		R_TS = 14	
	R(A)	R_TS = 15	
	W(A)	W_TS = 15	

⇒ **Thỏa giao thức tem thời gian?**

Giao thức dựa trên tem thời gian

- Ví dụ 2:

T_1	T_2	P	Q
1	2	R_TS = 0 W_TS = 0	R_TS = 0 W_TS = 0
W(P)		W_TS = 1	
	R(P)	R_TS = 2	
	R(Q)		R_TS = 2
W(Q)			

$TS(T_1) = 1$, R_TS(Q) = 2 ➔ $TS(T_1) <$ R_TS(Q)

⇒ **Hoạt động Write(Q) bị vứt bỏ và T_1 bị cuộn lại**
(không thỏa GT tem thời gian)

Giao thức dựa trên tem thời gian

T_1	T_2	T_3	A	B	C
200	150	175	R_TS = 0 W_TS = 0	R_TS = 0 W_TS = 0	R_TS = 0 W_TS = 0
R(B)				R_TS = 200	
	R(A)		R_TS = 150		
		R(C)			**R_TS = 175**
W(B)				W_TS =200	
W(A)			W_TS = 200		
	W(C)				
		W(A)			

Cuộn lại T_2

Cuộn lại T_3

Giao thức viết Thomas

- Là sự cải tiến của giao thức Tem thời gian nhằm loại bỏ trường hợp cuộn lại vô ích khi có "viết mù"

T1	T2	Q
R(Q)		R_TS = 1
	W(Q)	W_TS = 2
W(Q)		

$TS(T_1) = 1 < W\_TS(Q) = 2$

$\Rightarrow$ Vứt bỏ W(Q), cuộn lại T_1

Giao thức Viết Thomas:

1. **G/s T_i Read(Q):** giống giao thức Tem thời gian.
2. **G/s T_i Write(Q):**
   a. Nếu TS(Ti) < R_TS(Q): vứt bỏ Write(Q) và Ti cuộn lại.
   b. Nếu TS(Ti) < W_TS(Q): bỏ qua Write(Q).
   c. Cập nhật W_TS(Q) = TS(Ti)

Giao thức dựa trên tính hợp lệ

- Thích hợp đối với hệ thống có sự xung đột thấp (đa số giao dịch là chỉ đọc)
- Tổng chi phí để điều khiển cạnh tranh thấp do không giám sát tất cả các chỉ thị của GD mà chỉ thực hiện kiểm tra vào các "kỳ" nhất định
- **Tính chất:**
  - Giao thức này sinh ra các lịch trình Khả tuần tự xung đột
  - Tránh cuộn lại hàng loạt

Giao thức điều khiển cạnh tranh **"lạc quan"**

Giao thức dựa trên tính hợp lệ

- Một giao dịch T_i, có 2 (chỉ đọc) hoặc 3 kỳ:
  - Kỳ đọc: đọc các hạng mục DL, cập nhật trên biến tạm
  - Kỳ hợp lệ: kiểm tra tính hợp lệ để xác định có thể cập nhật từ biến lên CSDL hay không.
  - Kỳ viết: nếu Kỳ hợp lệ thành công ⇒ cập nhật;
    ngược lại ⇒ cuộn lại.

- Để phục vụ cho Kỳ hợp lệ, mỗi GD sẽ được gán các tem thời gian:
  - **Start(T_i)**: Thời điểm khởi động T_i
  - **Validation(T_i)**: Thời gian T_i hoàn thành bắt đầu Kỳ đọc, bắt đầ kỳ hợp lệ (kiểm thử).
  - **Finish(T_i)**: Thời điểm T_i kết thúc Kỳ viết.
  - **TS(T_i)**: tem thời gian của T_i (thường = Valiadation(T_i))

Giao thức dựa trên tính hợp lệ

- **Kiểm tra tính hợp lệ của T_j:** Với tất cả các T_i đã bàn giao hoặc trong thời kỳ hợp lệ, một trong các điều kiện sau phải thỏa:

1. $Finish(T_i) < Start(T_j)$
   (T_i thực hiện trước và kết thúc trước khi T_j, GD đến sau, bắt đầu ⇒ thứ tự tuần tự được đảm bảo)
2. $WS(T_i) \cap RS(T_j) = \varnothing \ \wedge$
   $Finish(T_i) < Validation(T_j)$
   (Thao tác viết của T_i không ảnh hưởng đến thao tác đọc của T_j)
3. $WS(T_i) \cap RS(T_j) = \varnothing \ \wedge$
   $WS(T_i) \cap WS(T_j) = \varnothing \ \wedge$
   $Validation(T_i) < Validation(T_j)$

$RS(T_i)$: tập các hạng mục dữ liệu được đọc bởi T_i

$WS(T_i)$: tập các hạng mục dữ liệu được ghi bởi T_i

Giao thức dựa trên tính hợp lệ

Ví dụ: Kiểm tra tính hợp lệ của các GD T_1, T_2, T_3:
$R_1(A, B)$ $R_2(B, C)$ $R_3(C)$ V_1 V_2 V_3 $W_1(A)$ $W_2(B)$ $W_3(C)$

- **Validate(T_1): OK** (không có GD nào validated hoặc committed)
- **Validate(T_2):** (T1 validated)
  $WS(T_1) \cap RS(T_2) = \varnothing \ \wedge \ WS(T_1) \cap WS(T_2) = \varnothing \ \wedge$
  $val(T_1) < val(T_2)$
  $\Rightarrow$ **OK**
- **Validate(T_3):** (T_1 và T_2 validated)
  - $WS(T_1) \cap RS(T_3) = \varnothing \ \wedge \ WS(T_1) \cap WS(T_3) = \varnothing \ \wedge$
    $val(T_1) < val(T_3)$ ➔ OK
  - $WS(T_2) \cap RS(T_3) = \varnothing \ \wedge \ WS(T_2) \cap WS(T_3) = \varnothing \ \wedge$
    $val(T_2) < val(T_3)$ ➔ OK

  $\Rightarrow$ **OK**

Giao thức dựa trên tính hợp lệ

Xét tính hợp lệ của các GD trong các lịch trình sau:

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

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

Deadlock

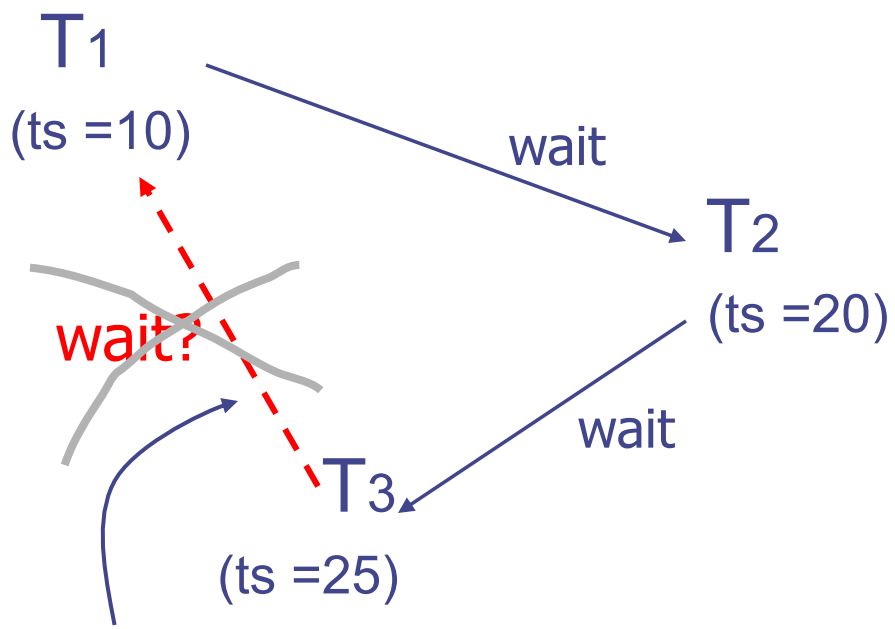
- **Deadlock:** hệ thống $\exists \{T_1, T_2, ..., T_n\}$ sao cho T_1 chờ T_2, T_2 chờ T_3, ...T_{n-1} chờ T_n, T_n chờ T_1
  - ⇒ Không GD nào tiến triển được
  - ⇒ Phải "tẩy rửa" một vài GD tham gia vào deadlock

- Xử lý deadlock trong hệ thống:
  - Ngăn ngừa deadlock: phù hợp khi xác suất deadlock cao
    - Giao thức điều khiển CT ngăn ngừa deadlock (cây,...)
    - Wait-die
    - Wound-wait
    - **Time-out**
  - Phát hiện: phù hợp khi xác suất deadlock thấp
    - Wait-for graph

Ngăn ngừa deadlock – Wait-die

Wait-die:

- Một GD T_i được gán 1 tem thời gian $TS(T_i)$ khi bắt đầu
- T_i có thể chờ T_j nếu $TS(T_i) < TS(T_j)$
  Ngược lại, T_i "chết" và cuộn lại

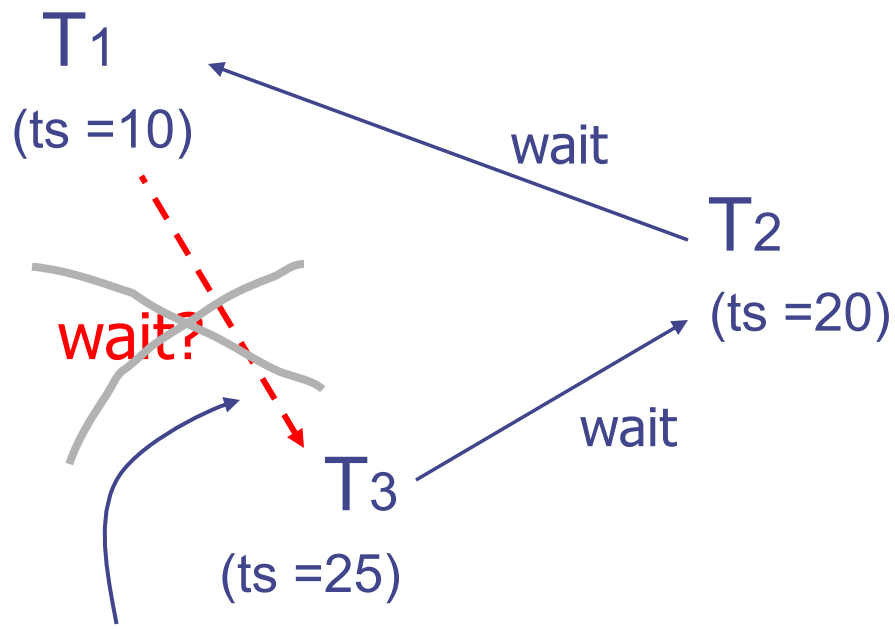


⇒ T3 không chờ được T1 ⇒ T3 chết và cuộn lại

Ngăn ngừa deadlock – Wound-wait

Wound-wait:

- Một GD T_i được gán 1 tem thời gian $TS(T_i)$ khi bắt đầu
- T_i "ép chết" (wound) T_j nếu $TS(T_i) < TS(T_j)$
  Ngược lại, T_i chờ



⇒ T1 "ép chết" T3 ⇒ T3 chết và cuộn lại

Ngăn ngừa deadlock – Wound-wait

- **Chú ý:**
  - Trong cả hai sơ đồ, khi một GD bị cuộn lại sẽ không được gán tem thời gian mới ⇒ tránh "chết đói".
  - Wait-die vs. Wound-wait:

Wait-die	Wound-wait
- GD già phải chờ GD trẻ. ⇒GD già có xu hướng chờ nhiều hơn.	- GD già không bao giờ chờ GD trẻ.
- Một GD T_i chết, sau khi khởi động lại có thể chết tiếp tục. ⇒T_i có thể chết vài lần trước khi tậu một hmục dl cần thiết.	- Một GD T_i bị thương và cuộn lại, sau khi khởi động lại sẽ chờ/ttục. ⇒ Có ít sự cuộn lại hơn trong sơ đồ Wound-wait.

Ngăn ngừa deadlock – Timeout

Timeout:

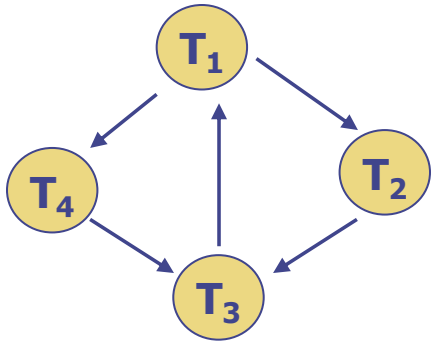
- Sơ đồ trung gian giữa ngăn ngừa và phát hiện deadlock.
- Nếu một GD T_i chờ quá một khoảng thời gian **L**
  ⇒ T_i sẽ bị cuộn lại

- Ưu điểm: đơn giản

- Khuyết điểm:
  - Khó xác định **L**
  - Có thể gây ra chết đói

Phát hiện deadlock

- **Đồ thị chờ** (wait-for graph): G = <V, E>
  - **V** = tập các đỉnh, gồm tất cả các giao dịch trong hệ thống
  - **E** = tập các cung. Nếu $\mathbf{T_i}$ chờ $\mathbf{T_j}$
    ⇒ bổ sung một cung $\mathbf{T_i} \longrightarrow \mathbf{T_j}$ vào **E**



- **Phát hiện deadlock**: nếu đồ thị chờ tại một thời điểm có chu trình ⇒ hệ thống tại thời điểm đó bị deadlock

Phát hiện deadlock

- Bao lâu xây dựng đồ thị chờ một lần?
  - Deadlock thường xảy ra không?
  - Bao nhiêu giao dịch sẽ bị ảnh hưởng bởi deadlock?

- Khôi phục từ deadlock:
  - Chọn nạn nhân: dựa trên các tiêu chí
    - thời gian thực hiện GD
    - số hạng mục dữ liệu sử dụng bởi GD
    - số giao dịch sẽ bị ảnh hưởng
  - Cuộn lại: cuộn lại bao xa
  - Sự chết đói: khi chọn nạn nhân, phải xem số lần đã cuộn lại của giao dịch

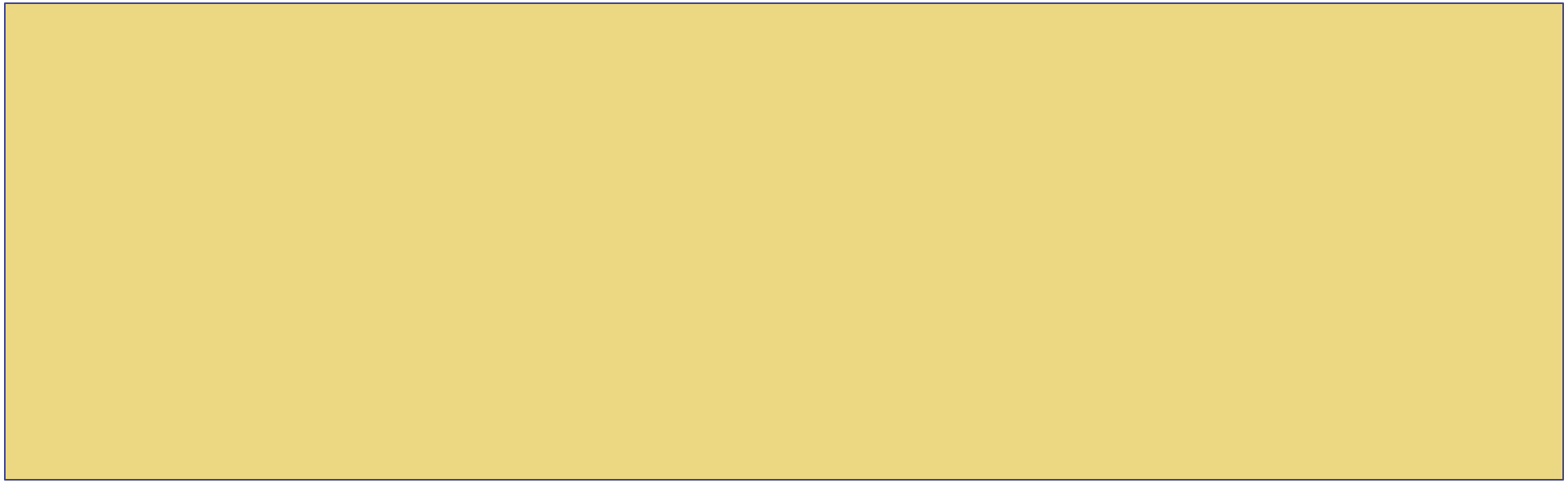
Phát hiện deadlock – Ví dụ

Cho **S = $r_1(A)r_2(B)w_1(C)w_2(D)r_3(C)w_1(B)w_4(D)w_2(A)$**

a. Vẽ đồ thị chờ. Deadlock? Nếu có thì chọn 1 GD để cuộn lại và cho biết trình tự thực hiện các GD. G/s các chỉ thị bị khóa sẽ thực hiện lại theo thứ tự chờ trước thực hiện trước.

b. Mô tả sự hoạt động của **S** trong hệ thống sử dụng sơ đồ Wound-Wait

c. Tương tự câu b, sử dụng sơ đồ Wait-Die.



Phát hiện deadlock – Ví dụ

b. Wait-Die:

S = r1(A)r2(B)w1(C)w2(D)r3(C)w1(B)w4(D)w2(A)

T_1	T_2	T_3	T_4
LS(A) R(A)			
	LS(B) R(B)		
LX(C) W(C)			
	LX(D) W(D)		
		LS(C) ➔ **Die (T_1)**	
LX(B) ➔ **Wait (T_2)**			
			LX(D) ➔ **Die (T_2)**
	LX(A) ➔ **Die (T_1)**		
W(B)			
U(..)			
	…	…	…

Phát hiện deadlock – Ví dụ

c. Wound-Wait:

S = $r_1(A)r_2(B)w_1(C)w_2(D)r_3(C)w_1(B)w_4(D)w_2(A)$

T_1	T_2	T_3	T_4
LS(A) R(A)			
	LS(B) R(B)		
LX(C) W(C)			
	LX(D) W(D)		
		LS(C) ➔ **Wait (T_1)**	
LX(B) W(B)	**Die (T_1 wounds)**		
U(…)			LX(D) W(D)
		(cont) R(C)	
	(restart)