

كلية الهندسة المعلوماتية- جامعة دمشق

قواعد معطيات (2)

المناقلات (المداولات)

Transaction

د. مادلين عبود

الفصل الخامس

# المناقلات

## Transaction

- مفهوم المناقلة
- حالة مناقلة
- الكتلية و الاستمرارية Atomicity and Durability
- التنفيذ بشكل قابل للتسلسل Serializability
- قابلية الاسترجاع والاستعادة Recoverability
- العزل Isolation
- تعريف المناقلات في SQL
- اختبار إمكانية تسلسل التنفيذ

# مفهوم المناقلة

- تعريف المناقلة (وحدة من التعليمات)
- المحافظة على قاعدة معطيات متلائمة **Consistence**، ويجب أن تبقى القاعدة متلائمة في نهاية المناقلة (بعد القيام بعملية Commit)
- ريعان رئيسيان لمفهوم المناقلة في التعامل مع :
  - ★ الأعطال المختلفة مثل أعطال التجهيزات، أو أعطال النظام.
  - ★ التحكم بالوصول المتزامن concurrency control لعدة مناقلات .

# لماذا التحكم بالوصول المتزامن؟؟.

■ الحاجة إلى التحكم بالوصول المتزامن concurrency control لعدة مناقلات .

T1	T2
Read (A) $A := A - N$	
	Read (A) $A := A + M$
Write (A) Read (B)	
	Write (A)
$B := B + N$ Write (B)	

القيمة المخزنة في A غير صحيحة - - - <

# لماذا التحكم بالوصول المتزامن؟؟.

■ تعديلات وسيطة (قراءة معطيات خاطئة)

T1	T2
Read (A) A := A-N Write (A)	
	Read (A) A := A + M Write (A)
Read (A) Transaction fails must change value of A to old value	

◀ T2 قامت بقراءة قيمة وسيطة خاطئة - -

# لماذا الاستعادة ?? Recovery

■ لا يسمح نظام إدارة قواعد المعطيات بأن تطبق جزء من العمليات الموجودة في المناقلة T على قاعدة المعطيات دون تطبيق بقية العمليات نتيجة حصول عطل ما.

■ أنواع الأعطال :

★ عطل في النظام System crash

★ خطأ في المناقلة

★ أخطاء محلية أو شروط استثنائية مكتشفة من قبل المناقلة

★ عطل في القرص

★ مشاكل فيزيائية

■ يقوم مدير عملية الاستعادة recovery manager بحفظ مسار العمليات التالية :

★ Begin transaction

★ Read or Write

★ End transaction

★ Commit transaction

★ Rollback (or Abort)

# خواص قواعد المعطيات ACID

يجب على قاعدة المعطيات، لتحافظ على تكامل المعطيات فيها، أن تؤمن الخواص التالية :

■ **Atomicity** الكتلية

■ **Consistency** الملاءمة

■ **Isolation** العزلة

★ من أجل المناقلتين  $T_i$  and  $T_j$  يبدو لـ  $T_i$  بأن المناقلة  $T_j$  تُنفذ بشكل كامل قبل أن تبدأ

$T_i$ ، أو أنها تبدأ التنفيذ بعد أن ينتهي تنفيذ  $T_j$

■ **Durability** الاستمرارية

## مثال

■ مناقلة لتحويل مبلغ 50 من حساب A إلى حساب B

1. **read**(A)
2.  $A := A - 50$
3. **write**(A)
4. **read**(B)
5.  $B := B + 50$
6. **write**(B)

■ المتطلبات :

Consistency ★

بقاء مجموع الحسابين  $A + B$  ثابت بعد تنفيذ المناقلة.

Atomicity ★

في حال حدث خطأ بين الخطوة 3 و الخطوة 6، يجب على النظام أن يتأكد من عدم إجراء أي تعديل على القاعدة و إلا ستصبح القاعدة غير متلائمة.

Durability ★

Isolation ★



# حالة المناقلة

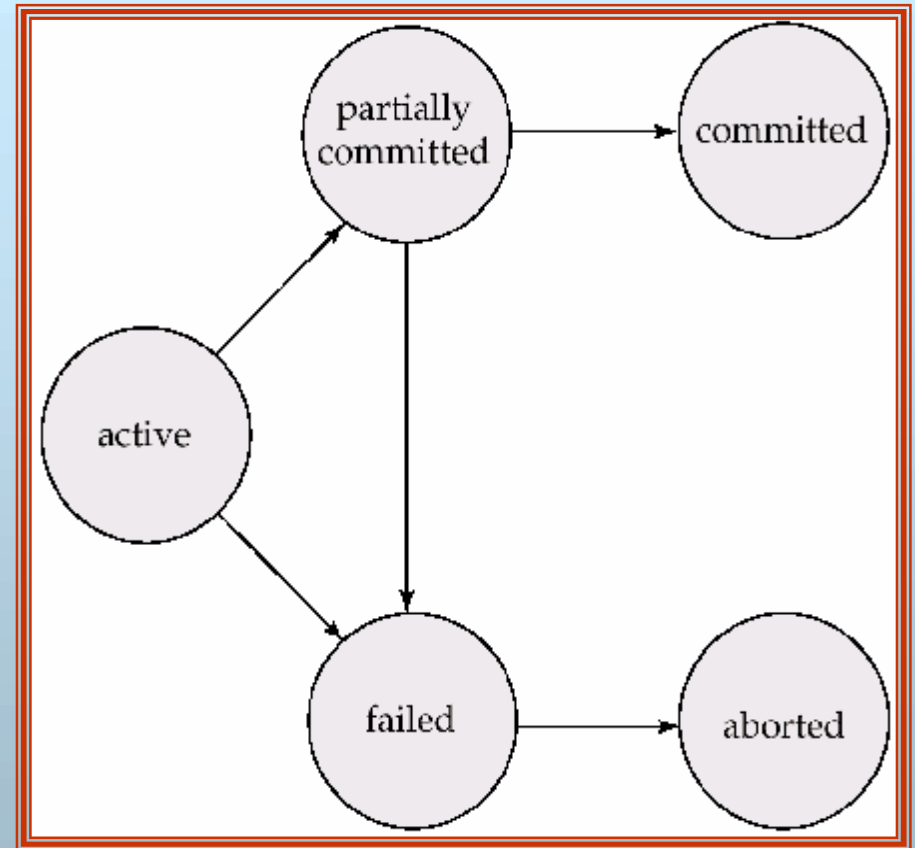
Active ■

Partially committed ■ بعد تنفيذ آخر  
تعليلة ضمن المناقلة

Failed ■ حالة الفشل

Aborted ■ إلغاء التنفيذ، بعد تعليلة rolled back والتي تعيد القاعدة إلى حالتها الأولى ويمكن إعادة تنفيذ المناقلة (حالة خطأ داخلي منطقي)، أو إنهاء المناقلة.

Committed ■



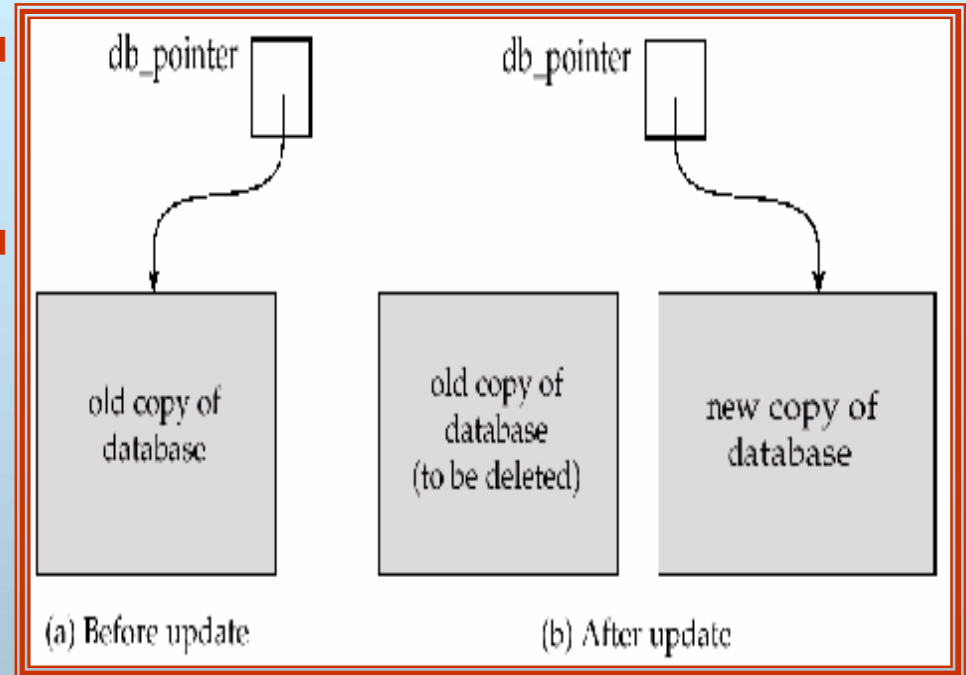
# تحقيق الكتلية والاستمرارية

■ تتحقق الكتلية والاستمرارية بواسطة الأجزاء المتعلقة بإدارة الاستعادة recovery management في قاعدة المعطيات.

■ مخطط قاعدة المعطيات الظل *shadow-database*

★ يفترض بأن تكون مناقلة واحدة فقط في حالة فعالة (active) في نفس الوقت.

★ db\_pointer مؤشر يُوْشِر دوماً إلى نسخة متلائمة من قاعدة المعطيات.



تفترض هذه الطريقة عدم وجود أعطال في الأقراص

هذه الطريقة مفيدة في الحالات البسيطة، ولكنها غير فعالة في قواعد معطيات ضخمة

# التنفيذ المتزامن

■ فوائد تنفيذ مجموعة من المناقلات بشكل متزامن في النظام

★ يزيد من استخدام المعالج والقرص مما يقود إلى معالجة عدد أكبر من المناقلات (throughput).

★ يقلل من متوسط زمن الاستجابة للمناقلات.

■ مخططات التحكم بالوصول المتزامن : تقنيات لتحقيق العُزلة

# مخططات التنفيذ Schedules

## ■ تعريف مخطط التنفيذ Schedules

★ تسلسل يبين الترتيب المنطقي الذي تُنفذ به تعليمات المناقلات المنفذة بشكل متزامن.

■ يحوي مخطط تنفيذ مجموعة من المناقلات جميع التعليمات الموجودة في تلك المناقلات

■ يحافظ المخطط على ترتيب التعليمات التي تظهر في كل مناقلة على حدى

## مثال على مخططات تنفيذ

- لتكن لدينا المناقلة  $T_1$  التي تقوم بتحويل \$50 من الحساب A إلى B، ولتكن المناقلة  $T_2$  التي تقوم بتحويل 10% من رصيد A إلى B.

<T2, T1>

$T_1$	$T_2$
<pre> read(A) A := A - 50 write(A) read(B) B := B + 50 write(B) </pre>	<pre> read(A) temp := A * 0.1 A := A - temp write(A) read(B) B := B + temp write(B) </pre>

<T1, T2>

$T_1$	$T_2$
<pre> read(A) A := A - 50 write(A) read(B) B := B + 50 write(B) </pre>	<pre> read(A) temp := A * 0.1 A := A - temp write(A) read(B) B := B + temp write(B) </pre>

## مثال مخطط تنفيذ

■ المخطط التالي يحوي تداخلاً في تنفيذ التعليمات و يكافئ المخطط السابق

$T_1$	$T_2$
read( $A$ ) $A := A - 50$ write( $A$ )	read( $A$ ) $temp := A * 0.1$ $A := A - temp$ write( $A$ )
read( $B$ ) $B := B + 50$ write( $B$ )	read( $B$ ) $B := B + temp$ write( $B$ )

تحافظ قاعدة المعطيات على ملاءمتها في المخططين السابقين ويبقى مجموع  $A + B$  ثابتاً.

## تابع

$T_1$	$T_2$
$\text{read}(A)$ $A := A - 50$	$\text{read}(A)$ $\text{temp} := A * 0.1$ $A := A - \text{temp}$ $\text{write}(A)$ $\text{read}(B)$
$\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$	$B := B + \text{temp}$ $\text{write}(B)$

■ لا يحافظ المخطط على مجموع  $A + B$

# التسلسلية Serializability

- نفترض أن كل مناقلة على حدى تحافظ على قاعدة المعطيات متلائمة.
- هل يحافظ التنفيذ المتداخل لمجموعة من المناقلات على قاعدة المعطيات متلائمة؟
- نقول عن مخطط تنفيذ (متداخل لعدة مناقلات) أنه متسلسل (قابل للتسلسل) إذا كان مكافئاً لمخطط تنفيذ متعاقب.
- أدت الأشكال المختلفة من مخططات التنفيذ المتكافئة إلى ظهور مفهومين هما :

★ متسلسل تصادمياً Conflict Serializability

★ متسلسل ظاهرياً View Serializability

- سنهمل العمليات التي لا تتعلق بالقراءة والكتابة من وعلى القرص، ونفترض أنه يمكن للمناقلات إجراء العمليات الحسابية على المعطيات الموجودة ضمن buffers بين عمليتي القراءة والكتابة.
- ★ يتكون المخطط المبسط الناتج من عمليات القراءة والكتابة من وعلى القرص فقط.



# متسلسل تصادمياً

## Conflict Serializability

- ليكن لدينا التعليمات التالية  $I_i$  من المناقلة  $T_i$  و  $I_j$  من المناقلة  $T_j$
- نقول أنه يوجد تصادم **conflict** بين التعليمتين  $I_i$  and  $I_j$  إذا وفقط إذا وجدت بعض العناصر مثل  $Q$  ، تتعامل المناقلتان معها من خلال التعليمتين  $I_i$  and  $I_j$  ، وتتضمن إحدى هاتان التعليمتان على الأقل عملية كتابة :

  1.  $I_i = \text{read}(Q)$ ,  $I_j = \text{read}(Q)$ . no conflict.
  2.  $I_i = \text{read}(Q)$ ,  $I_j = \text{write}(Q)$ . conflict.
  3.  $I_i = \text{write}(Q)$ ,  $I_j = \text{read}(Q)$ . conflict
  4.  $I_i = \text{write}(Q)$ ,  $I_j = \text{write}(Q)$ . conflict

- وجود تصادم يستدعي وجود ترتيب زمني بين العمليتين. في حال كانت التعليمتان متعاقبتين ولا وجود لتصادم فيما بينهما، تبقى النتيجة نفسها حتى ولو حصل تبديل في ترتيب تنفيذهم ضمن مخطط التنفيذ Schedule.

# متسلسل تصادمية

## Conflict Serializability

- نقول أن  $S$  and  $S'$  مخططاً تنفيذ متكافئين تصادمية **conflict equivalent** إذا نتج أحدهم عن الآخر بعد إجراء سلسلة من التبديلات بين التعليمات غير المتصادمة
- نقول أن المخطط  $S$  مخططاً متسلسلاً تصادمية **conflict serializable** إذا كان مخططاً مكافئاً تصادمية **conflict equivalent** لمخطط تنفيذ متعاقب.
- مثال : مخطط التنفيذ (1) هو مخطط غير متسلسل تصادمية :

$T_3$	$T_4$
read(Q)	write(Q)
write(Q)	

مخطط تنفيذ (1)

عدم إمكانية الحصول على مخطط تنفيذ متعاقب  $\langle T_3, T_4 \rangle$  أو  $\langle T_4, T_3 \rangle$

# متسلسل تصادميةً

## Conflict Serializability

$T_1$	$T_2$
read(A) write(A)	read(A) write(A)
read(B) write(B)	read(B) write(B)

المخطط (2)

■ يمكن تحويل المخطط (2) إلى مخطط تنفيذ متعاقب  $\langle T_1, T_2 \rangle$  بعد القيام بسلسلة من التباديل بين التعليمات غير المتصادمة. وبالتالي المخطط (2) هو مخطط تنفيذ متسلسل تصادميةً.

## التحويل إلى مخطط تعاقبي

$T_1$	$T_2$
read(A)	
write(A)	
read(B)	
write(B)	
	read(A)
	write(A)
	read(B)
	write(B)

مخطط تعاقبي

$T_1$	$T_2$
read(A)	
write(A)	
	read(A)
read(B)	
	write(A)
write(B)	
	read(B)
	write(B)

بعد إجراء بعض التبديلات

## متسلسل ظاهرياً View Serializability

■ ليكن لدينا  $S$  and  $S'$  مخططاً تنفيذ لنفس مجموعة المناقلات. نقول أن  $S$  and  $S'$  متكافئين ظاهرياً view equivalent إذا تحققت الشروط التالية :

★ من أجل كل حد من المعطيات  $Q$ ،

➤ إذا جرت عملية قراءة للقيمة الابتدائية لـ  $Q$  في المناقلة  $T_i$  ضمن المخطط  $S$ ، يجب أن تقوم نفس المناقلة  $T_i$  في المخطط  $S'$  أيضاً بقراءة للقيمة البدائية لـ  $Q$ .

★ من أجل كل حد من المعطيات  $Q$ ،

➤ إذا احتوت المناقلة  $T_i$  تنفيذ تعليمة  $read(Q)$  في المخطط  $S$  لقيمة ناتجة من المناقلة  $T_j$ ، يجب أن تحوي نفس المناقلة  $T_i$  في المخطط  $S'$  أيضاً قراءة لقيمة  $Q$  الناتجة من المناقلة  $T_j$ .

★ من أجل كل حد من المعطيات  $Q$ ،

➤ إذا قامت مناقلة بآخر عملية كتابة لـ  $Q$  في المخطط  $S$ ، يجب أن تُنفذ نفس المناقلة آخر عملية كتابة لـ  $Q$  في المخطط  $S'$ .

■ كما نرى فإن تكافؤ المظهر يعتمد فقط على عمليات القراءة والكتابة.

# متسلسل ظاهرياً

## View Serializability

- نقول أن المخطط  $S$  مخططاً **متسلسلاً ظاهرياً** إذا كان مخططاً مكافئاً ظاهرياً  $\text{view equivalent}$  لمخطط متعاقب  $\text{serial schedule}$ .
- كل مخطط متسلسل تصادمية  $\text{conflict serializable schedule}$  هو أيضاً مخطط متسلسل ظاهرياً  $\text{view serializable}$ .
- المخطط (3) هو مخطط متسلسل ظاهرياً  $\text{view-serializable}$  ولكنه غير متسلسل تصادمية  $\text{not conflict serializable}$ .

$T_3$	$T_4$	$T_6$
read(Q)	write(Q)	
write(Q)		
		write(Q)

المخطط (3)

## مفاهيم أخرى من التسلسلية

- هل المخطط متسلسل تصادمية؟
- هل المخطط متسلسل ظاهرياً؟
- هل النتيجة مكافئة لتنفيذ متسلسل؟

$T_1$	$T_5$
$\text{read}(A)$ $A := A - 50$ $\text{write}(A)$	$\text{read}(B)$ $B := B - 10$ $\text{write}(B)$
$\text{read}(B)$ $B := B + 50$ $\text{write}(B)$	$\text{read}(A)$ $A := A + 10$ $\text{write}(A)$

- المخطط التالي يؤدي إلى نفس النتيجة في حال جرى تنفيذ المناقلتين بشكل متعاقب، وهو بنفس الوقت ليس مكافئاً تصادميةاً أو مكافئاً ظاهرياً لتنفيذ متعاقب.
- يتطلب تحديد تكافؤ تنفيذ المخطط لتنفيذ مخطط متعاقب تحليل العمليات الأخرى غير الكتابة والقراءة، وهو بشكل عام صعب التحقيق وباهظ الثمن.

# إمكانية الاستعادة Recoverability

نحتاج إلى تحديد تأثير عطل أثناء تنفيذ مناقلة على التنفيذ المتزامن لعدة مناقلات.

## ■ المخطط القابل للاستعادة Recoverable schedule

★ في حال احتوت المناقلة  $T_j$  على عملية قراءة لمجموعة حدود كتبت في مناقلة  $T_i$  سابقاً، يجب أن تُنفذ عملية COMMIT في المناقلة  $T_i$  قبل أن تنفذ في المناقلة  $T_j$ .

$T_8$	$T_9$
read(A)	
write(A)	
	read(A)
read(B)	

هل المخطط التالي قابل للاستعادة إذا جرى تنفيذ تعليمية Commit في المناقلة  $T_9$  فوراً بعد عملية القراءة ؟


في حال حصل عطل في تنفيذ المناقلة  $T_8$  وجرى تنفيذ تعليمية Rollback،  $T_9$  انتهى تنفيذها بشكل طبيعي وتكون قد استخدمت قيمة غير صحيحة للمتحول A وتظهر قاعدة المعطيات بشكل غير متلائم.

يجب على قاعدة المعطيات التأكد من أن المخططات التي تُنفذها هي مخططات قابلة للاستعادة.



## إمكانية الاستعادة

- **Cascading rollback** العودة بشكل شلال : يمكن أن يؤدي عطل في مناقلة إلى إيقاف تنفيذ مخطط لعدة مناقلات والعودة لسلسلة من المناقلات rollbacks .
- لنفترض أنه لدينا مخطط التنفيذ التالي حيث لم يتم تسجيل commit لأية مناقلة من المناقلات.



$T_{10}$	$T_{11}$	$T_{12}$
read(A) read(B) write(A)	read(A) write(A)	read(A)

إذا فشل تنفيذ  $T_{10}$  يجب أيضاً إيقاف تنفيذ والعودة rolled back لـ  $T_{11}$  ,  $T_{12}$

# إمكانية الاستعادة

## ■ مخططات التنفيذ لمناقلات غير مترابطة بشكل شلال - Cascadeless schedules

- ★ المخططات التي تؤدي إلى عودة بشكل شلال غير مرغوب بها
- ★ يكون مخطط تنفيذ مجموعة من المناقلات غير مترابط بشكل شلال إذا كان
  - من أجل كل زوج من المناقلات  $T_i$  and  $T_j$  ، وجدت عملية قراءة في مناقلة  $T_j$  لحدود جرى كتابتها في المناقلة  $T_i$  ، فيجب أن تظهر عملية الـ Commit في المناقلة  $T_i$  قبل عملية القراءة في المناقلة  $T_j$ .
- كل مخطط غير مترابط بشكل شلال هو أيضاً قابل للاستعادة

من المستحسن التعامل فقط مع المخططات غير مترابطة بشكل شلال.

# تتفيذ العزلة Isolation

■ لتحقيق العزلة في تنفيذ المناقلات

★ إقفال قاعدة المعطيات بشكل كامل عند تنفيذ مناقلة <==> أداء غير فعال

★ يجب أن يكون مخطط التنفيذ مخططاً متسلسلاً تصادميةً أو متسلسلاً ظاهرياً، وقابل للاستعادة ومن الأفضل أن يكون غير مترابط بشكل شلال من أجل الحفاظ على أن تبقى قاعدة المعطيات متلائمة.

# تعريف المناقلة في لغة SQL

- تحوي لغة التعامل مع المعطيات DML تعابير لتحديد مجموعة التعليمات المكونة لمناقلة
- تبدأ المناقلة في لغة SQL بشكل ضمني (ليست بحاجة لتصريح)
- تنتهي المناقلة في لغة SQL بـ :
- ★ Commit work لتسجيل المناقلة الحالية (تسجيل التعديلات على قاعدة المعطيات) والبدء في مناقلة جديدة.
- ★ Rollback work لإنهاء عمل المناقلة الحالية والعودة إلى الحالة السابقة لقاعدة المعطيات

# اختبار التسلسلية – مخطط متسلسل تصادفياً

■ لنأخذ مجموعة من مخططات التنفيذ لمجموعة من المناقلات  $T_1, T_2, \dots, T_n$

■ رسم بيان موجه لتحديد السوابق (بيان السوابق) **precedence graph** :

★ العقد : أسماء المناقلات

★ رسم رابطة (قوس) من  $T_i$  إلى  $T_j$  ( $T_i \rightarrow T_j$ ) في حال كانت المناقلتان متصادمتين، ويتم ذلك في حال تحقيق أحد الشروط التالية:

➤ تنفذ المناقلة  $T_i$  تعليمة  $write(Q)$  قبل أن تنفذ  $T_j$  تعليمة  $read(Q)$

➤ تنفذ المناقلة  $T_i$  تعليمة  $read(Q)$  قبل أن تنفذ  $T_j$  تعليمة  $write(Q)$

➤ تنفذ المناقلة  $T_i$  تعليمة  $write(Q)$  قبل أن تنفذ  $T_j$  تعليمة  $write(Q)$

★ في حال وجد قوس  $T_i \rightarrow T_j$  في بيان السوابق، يعني ذلك أنه في جميع المخططات المتعاقبة  $S'$  المكافئة لـ  $S$  يجب أن تنفذ  $T_i$  قبل  $T_j$

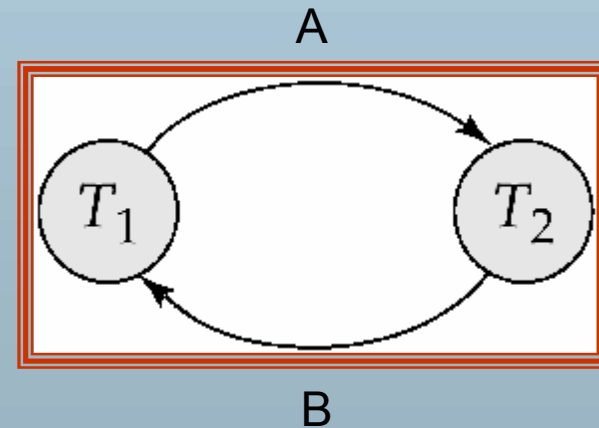
★ يمكن وضع اسم على القوس (اسم حد التصادم)

■ مثال :

Read(A); A:= A-50	
	Read(A) Temp :=A*0.1; A :=A- temp Write(A) Read(B)
Write(A); Read(B) B := B+50 Write(B)	
	B:=B+temp Write(B)

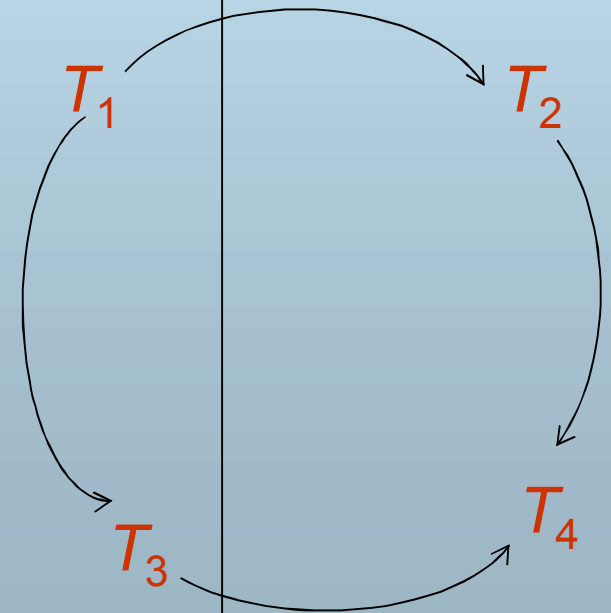
T1

T2

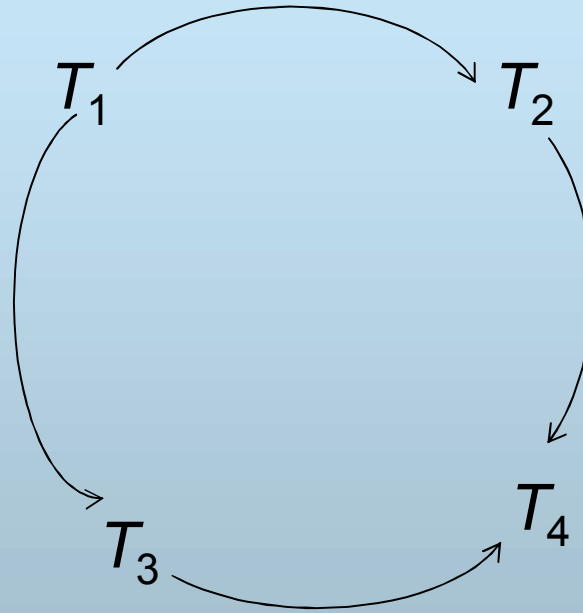


# مثال 1

$T_1$	$T_2$	$T_3$	$T_4$	$T_5$
read(Y) read(Z)	read(X)			read(V) read(W) read(W)
	read(Y) write(Y)	write(Z)		
read(U)			read(Y) write(Y) read(Z) write(Z)	
read(U) write(U)				



# بيان تمثيل المخطط 1



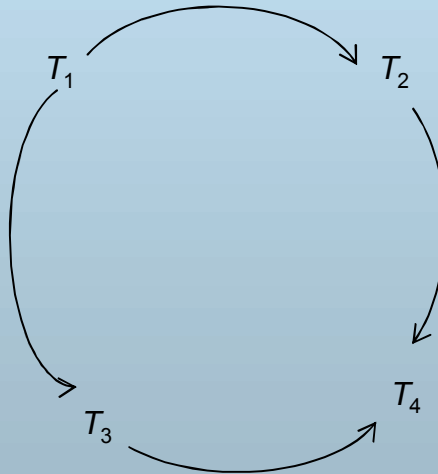
# اختبار كون مخطط تنفيذ متسلسلاً تصادميةً

■ نقول عن مخطط تنفيذ أنه متسلسل تصادميةً إذا و فقط إذا لم يحوي بيان السوابق الممثل له حلقات acyclic graph

■ تحتاج خوارزمية كشف حلقة في بيان مؤلف من  $n$  عقدة إلى  $n^2$  عملية

■ مثال : ترتيب متسلسل للمخطط A

$$T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4 .$$

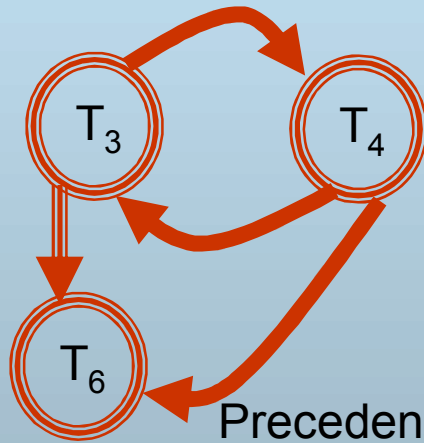




# اختبار كون مخطط تنفيذ متسلسلاً ظاهرياً

■ ليكن لدينا مخطط التنفيذ التالي :

$T_3$	$T_4$	$T_6$
read(Q)	write(Q)	write(Q)
write(Q)		



مخطط تنفيذ متسلسل ظاهرياً وغير متسلسل تصادمية

مكافئ ظاهرياً لتنفيذ  $T_3 \rightarrow T_4 \rightarrow T_6$

نلاحظ أنه يمكن حذف القوس  $T_4 \rightarrow T_3$  حيث أن القيمة الناتجة من  $T_4$ ,  $T_3$  غير مستخدمة بأية مناقلة لاحقاً. تدعى التعليمة  $write(Q)$  في المناقلتين  $T_4$ ,  $T_3$  كتابة غير مستخدمة  
 نحتاج لتطوير طريقة  
 لتحديد الحاجة إلى وجود  
 أقواس في البيان

# بيان السوابق المعدل

■ تطوير بيان السوابق ليكون مناسباً لاختبار كون مخطط تنفيذ متسلسلاً ظاهرياً (بيان السوابق المعدل):

1. نضيف بشكل وهمي مناقلتين (مناقلة البداية، مناقلة النهاية) لمخطط التنفيذ هما  $T_b$ ,  $T_f$  حيث تحوي  $T_b$  تعليمات كتابة  $write(Q)$  لكل حد جرى استخدامه في المخطط، و تحوي  $T_f$  تعليمات قراءة  $read(Q)$  لكل حد جرى استخدامه في المخطط .
2. إضافة قوس من  $T_i$  إلى  $T_j$  مع لاحقة 0 ( $T_i 0 \rightarrow T_j$ ) في حال جرى قراءة لقيمة حد  $Q$  في المناقلة  $T_j$  كتبت قبلاً في  $T_i$  .
3. حذف جميع الأقواس المرتبطة بالمناقلات غير المستخدمة (نقول أن مناقلة  $T_i$  غير مستخدمة في حال لم يتواجد قوس يربطها بالمناقلة النهائية  $T_f$  في بيان السوابق)

## بيان السوابق المعدل

4. من أجل كل عنصر مثل Q حيث :

- a. تقوم المناقلة  $T_j$  بقراءة قيم Q المكتوبة قبلاً في المناقلة  $T_i$  نفذ ما يلي :
- b. تنفذ المناقلة  $T_k$  عملية  $write(Q)$  و تختلف  $T_k$  عن  $T_b$

1. إذا  $T_b = T_i$  و كان  $T_f \text{ not} = T_j$ ، قم بإضافة قوس  $T_k \rightarrow 0$  في بيان السوابق المعدل.

2. إذا  $T_b \text{ not} = T_i$  و كان  $T_f = T_j$ ، قم بإضافة قوس  $T_k \rightarrow 0$  في بيان السوابق المعدل.

3. إذا  $T_b \text{ not} = T_i$  و كان  $T_f \text{ not} = T_j$ ، قم بإضافة القوسين  $T_k \rightarrow p$  و  $T_j \rightarrow p$  في بيان السوابق المعدل. حيث p عدد صحيح  $0 < p$ .

إذا كتبت مناقلة  $T_i$  معطيات قامت مناقلة أخرى  $T_j$  بقراءتها، وقامت مناقلة  $T_k$  بكتابة نفس المعطيات، فيجب تنفيذ هذه المناقلة  $T_k$  قبل المناقلة  $T_i$  أو بعد  $T_j$ .

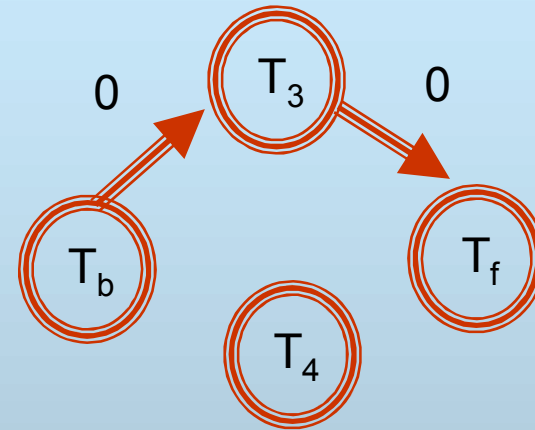
الحالتان 1 و 2 هما حالات خاصة ناتجة عن كون  $T_b$  و  $T_f$  هما المناقلتان الأولى والأخيرة في التسلسل.

تطبيق القاعدة (3) لا يعني بأن  $T_k$  تأتي قبل  $T_i$  وبعد  $T_j$ ، وإنما لدينا أن نختار إحدى الحالتين.

## مثال : اختبار كون مخطط تنفيذ متسلسل ظاهرياً

$T_3$	$T_4$
read(Q)	write(Q)
write(Q)	

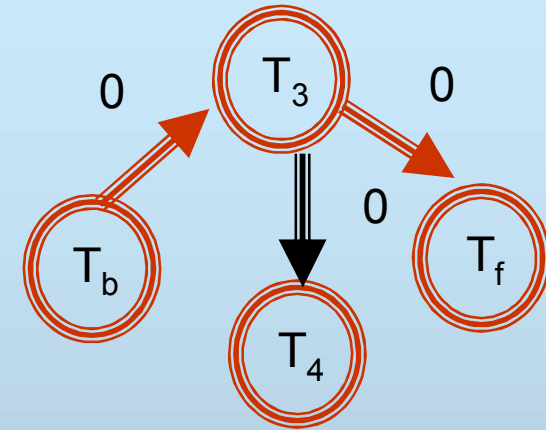
■ لنأخذ مخطط التنفيذ التالي



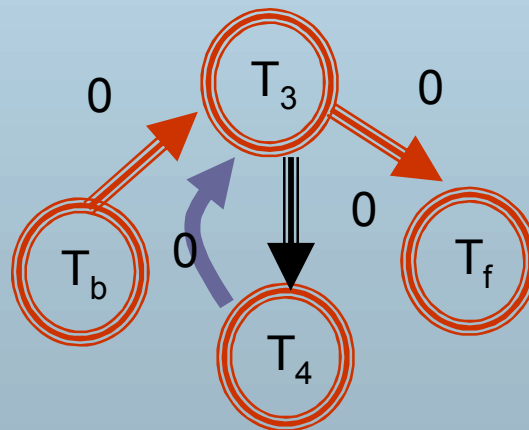
بيان السوابق بعد تطبيق الخطوتين 1،2،3

## مثال (تابع) اختبار كون مخطط تنفيذ متسلسل ظاهرياً

$T_3$	$T_4$
read(Q)	
write(Q)	write(Q)



تطبيق الخاصة (4,1) حيث  $T_3$  تقرأ معطيات كتبت في  $T_4$  و  $T_b$  تكتب نفس المعطيات إضافة  $T_4 \rightarrow T_3$



تطبيق الخاصة (4,2) حيث  $T_f$  تقرأ معطيات كتبت في  $T_3$  و  $T_4$  تكتب نفس المعطيات إضافة  $T_4 \rightarrow T_3$

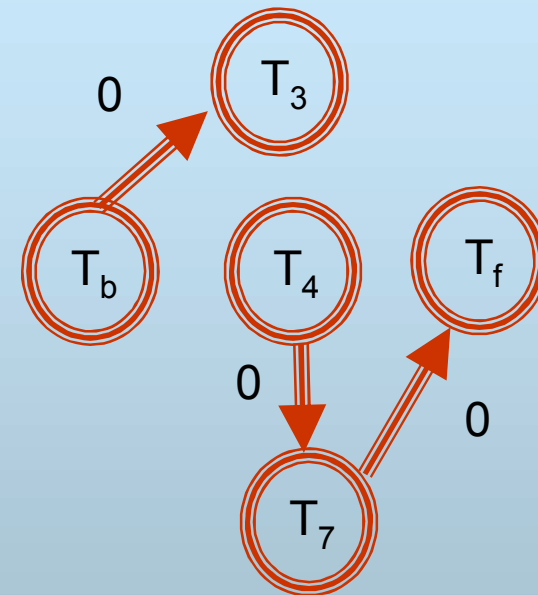
نتيجة

بيان يحوي حلقة وهو غير متسلسل ظاهرياً  
إذا وجد بيان لمخطط تنفيذ لا يحوي حلقة ← يكون  
مخطط التنفيذ متسلسل ظاهرياً

## مثال : اختبار كون مخطط تنفيذ متسلسل ظاهرياً

$T_3$	$T_4$	$T_7$
read(Q)	write(Q)	read(Q)
write(Q)		write(Q)

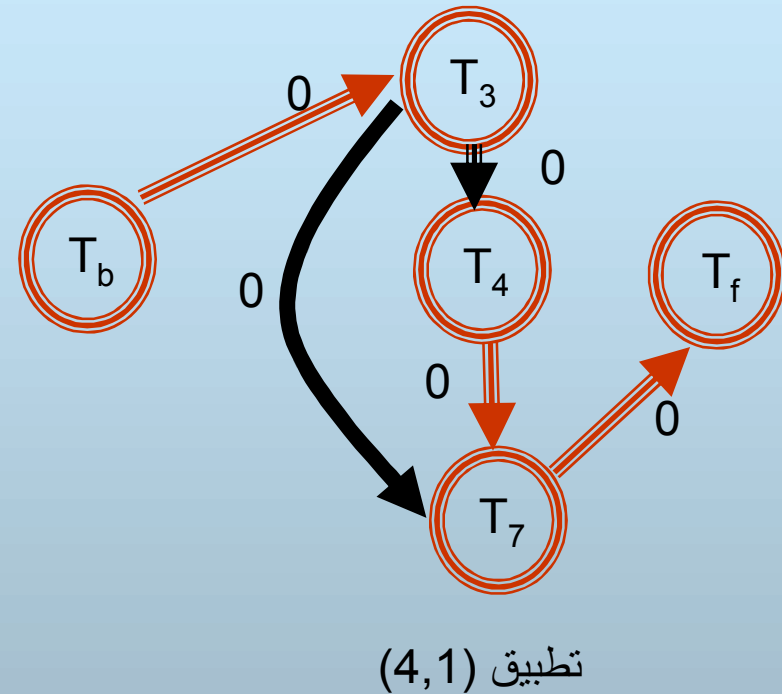
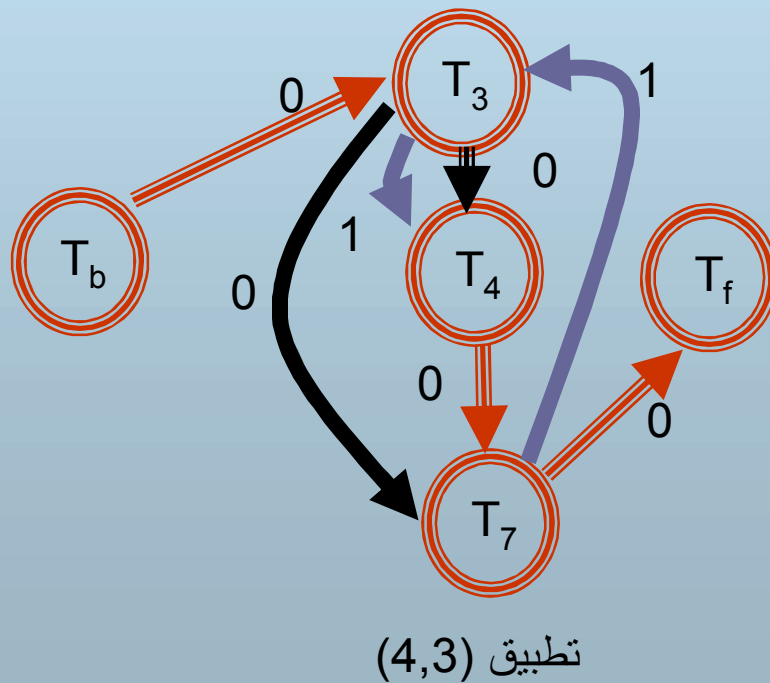
■ لنأخذ مخطط التنفيذ التالي



بيان السوابق بعد تطبيق الخطوتين 1،2،3

## مثال : اختبار كون مخطط تنفيذ متسلسل ظاهرياً

$T_3$	$T_4$	$T_7$
read(Q)	write(Q)	read(Q)
write(Q)		write(Q)



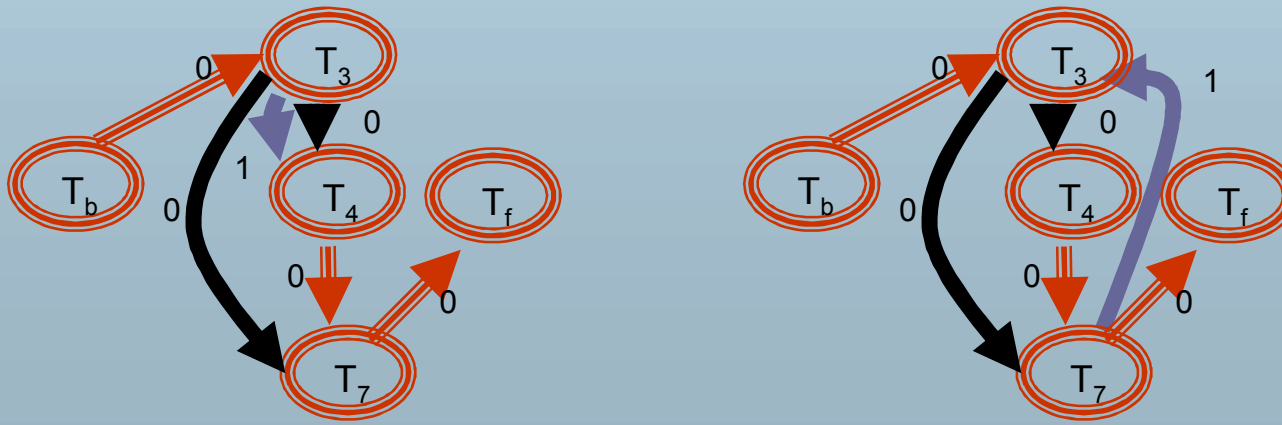
نتيجة : بيان يحوي حلقة وهو  
متسلسل ظاهرياً

## اختبار التسلسلية

■ كيف يمكن تحديد كون مخطط تنفيذ متسلسلاً ظاهرياً؟

يرتبط الجواب بكيفية إنشاء بيان السوابق. (في حال لم يتضمن أحد أشكال بيان السوابق الناتج عن مخطط تنفيذ حلقات نقول أن المخطط متسلسل ظاهرياً).

بفرض عدد مرات تطبيق الخاصة 3-4 هو  $n$  يكون لدينا  $2^n$  بيان (مؤلف من أخذ أحد أطراف الأزواج الناتجة من تطبيق الخاصة 3-4) وبالتالي يكفي أن لا يحوي أحد تلك البيانات على حلقة ليكون مخطط التنفيذ متسلسل ظاهرياً ومكافئ لمخطط متسلسل ناتج عن الفرز الطوبولوجي للبيان غير الحاوي على حلقات.





# التحكم بالوصول المتزامن واختبار التسلسلية

- اختبار تسلسلية مخطط تنفيذ بعد تنفيذه عملياً، عملية تكون متأخرة جداً.
- الهدف : تطوير بروتوكول للتحكم بالوصول المتزامن يؤمن التسلسلية. لن نقوم باختبار بيان السوابق بعد انشائه وإنما سنجعل البروتوكول يتجنب تنفيذ مخطط غير متسلسل.

**End of Chapter**

