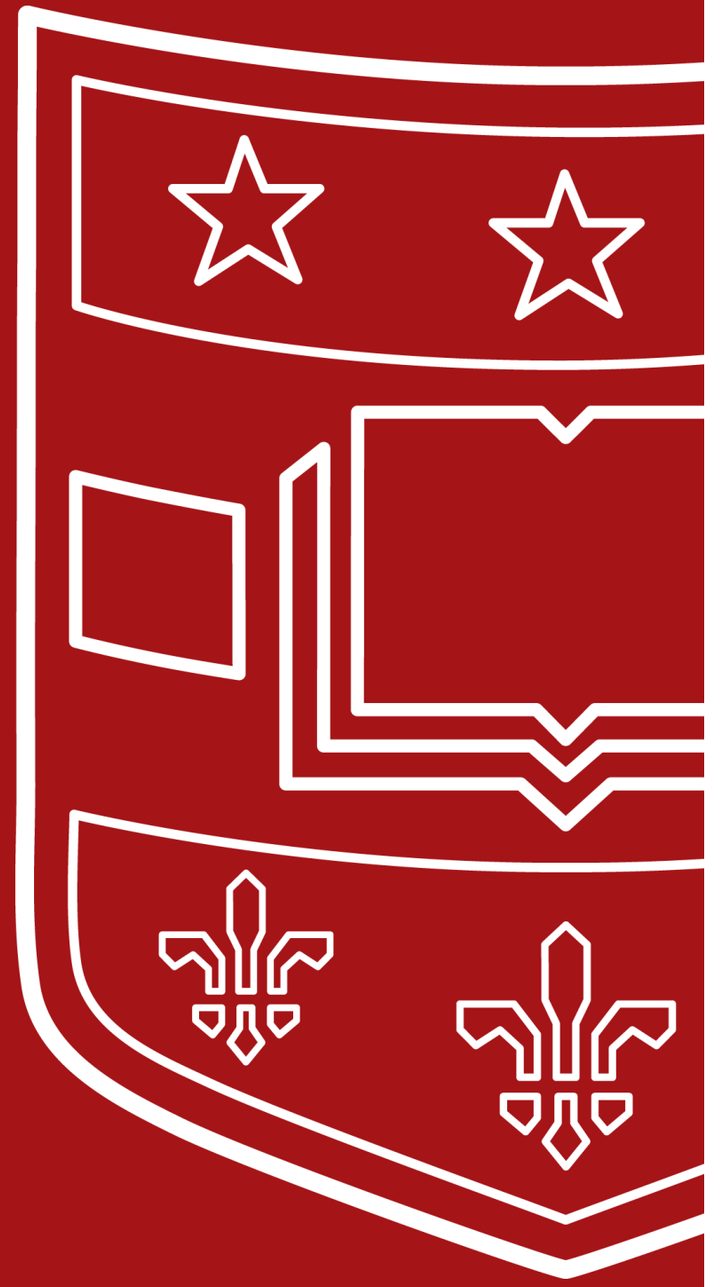
Database Management Systems

- Concurrency Control



Washington University in St. Louis

Review

- Why do we need transactions?
  - What does a transaction contain?
  - What are the four properties of a transaction?

- What is a schedule?
  - What is a conflict?
  - What does it mean for a schedule to be serializable?

Locking

- We wish to create serializable schedules
- Many techniques for this
  - Locking
  - Timestamps
  - Multiversion
- In practice, locking is used most frequently

Binary Locks

- Two states:
  - Locked/Unlocked
  - Cannot access an item while it is locked
- Enforces mutual exclusion

Implementation

- Must maintain a table of locks
  - And a queue…
- Must obey locking rules
- In practice this works fine, but it is too restrictive
  - Why?

Shared / Exclusive Locks

- We wish for our locks to mesh well with conflicts
- Two kinds of locks
- How can we track them?

Shared / Exclusive Locks

- Still must obey locking rules

- Can convert between the two types of locks
  - What must be true?

Two Phase Locking

- All locks must be acquired before first unlock statement
  - Two phases: acquisition and release

- Guarantees serializability

Two Phase Locking

T_1	T_2

read_lock(Y);
read_item(Y);
unlock(Y);
write_lock(X);
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);
	read_lock(X);
	read_item(X);
	unlock(X);
	write_lock(Y);
	read_item(Y);
	$Y := X + Y$;
	write_item(Y);
	unlock(Y);

T_1'

read_lock(Y);
read_item(Y);
write_lock(X);
unlock(Y)
read_item(X);
$X := X + Y$;
write_item(X);
unlock(X);

T_2'

read_lock(X);
read_item(X);
write_lock(Y);
unlock(X)
read_item(Y);
$Y := X + Y$;
write_item(Y);
unlock(Y);

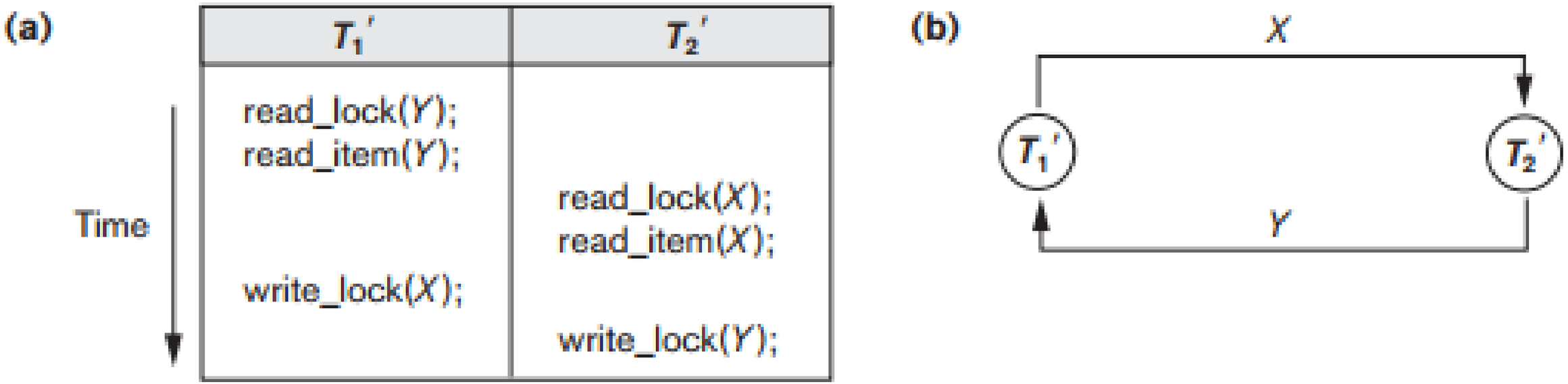
Types of Two Phase

- Basic
- Conservative
- Strict

Issues with Two Phase

- Cannot generate all possible serializable schedules
- Reduces Concurrency
- Can lead to deadlock
- Can lead to starvation

Deadlock


(a)
T_1'
T_2'
read_lock(Y);
read_item(Y);
read_lock(X);
read_item(X);
Time
write_lock(X);
write_lock(Y);
(b)
X
T_1'
T_2'
Y

Deadlock Prevention

- Timestamp based
  - Wait-die: younger transaction dies, older can wait
  - Wound-wait: Older can abort younger, younger simply wait

- Problems?

Deadlock Prevention

- Non-timestamp based
  - No-wait: abort if unavailable
  - Cautious wait

Deadlock Detection

- Create a graph:
  - Nodes are transactions
  - Edges occur when transactions must wait
  - When do we know when deadlock has occurred?

- Performance considerations?

Starvation

- Occurs when a transaction has to wait a long time
  - Perhaps forever?
  - When does this happen?

- Solutions:
  - Time scaling priority
  - FCFS

Granularity

- What should we be locking?
  - Fields?
  - Columns?
  - Pages?
  - Tables?

- How does this affect concurrency?
- Why not just lock everything on the field level?
- What is the best size?

Granularity

- Two transactions:
  - One wants to update all records in a file, the other wants to update a single record

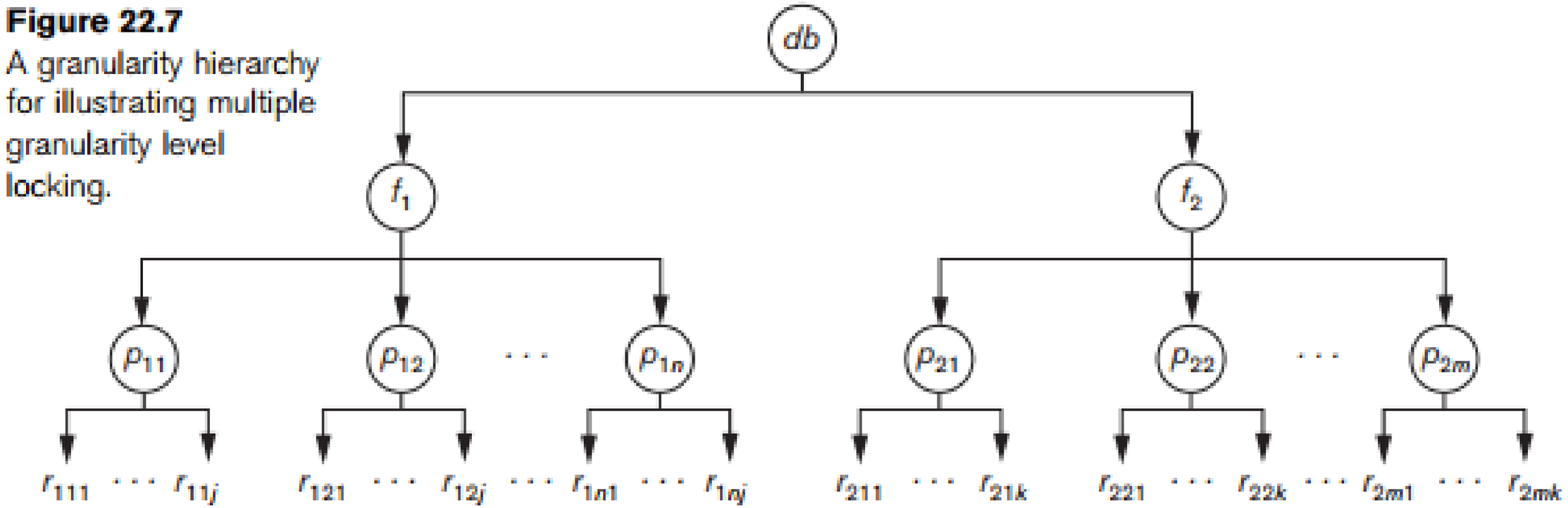


Figure 22.7
A granularity hierarchy for illustrating multiple granularity level locking.

Intention Locks

- What locks would be requested on descendants of the tree?

- Intention Shared/Exclusive

- Shared Intention Exclusive

Intention Locks

- Lock the root first
- Can be locked in S or IS if parent is locked in IS or IX
- Can be locked in X, IX, or SIX if parent is locked in IX or SIX mode
- Can only be unlocked if children are unlocked

Exercises

- You are given the following schedule:

R2(A) R2(Y) R3(Z) W2(Y) R1(A) W3(Z) W2(A) W1(Z) W1(A)

Prove whether or not the schedule is conflict serializable

If the schedule is serializable, determine an equivalent serial schedule and prove that they are equal. If it is not, use the same transactions to create a non-serial schedule that is serializable and prove that it is serializable.

Exercises

- Does strict two phase locking prevent deadlock? If yes, explain how. If no, provide an example of a schedule that obeys two phase locking but also creates deadlock.

- Will basic two phase locking lead to better performance than strict two phase locking (in general)?