CSE 486/586 Distributed Systems
Concurrency Control --- 2

Steve Ko
Computer Sciences and Engineering
University at Buffalo

Midterm Review

- Please check your midterm
  - Any mistake?
  - Any re-grading request?
- Please come down row-by-row
- Please return your exam after you're done.

CSE 486/586 Administrivia

- PA3 deadline: 3/29 (Friday)
- PA2 grading
  - Will be done sometime next week (we wanted to grade the midterm first.)
- Tech Talk: Chris Buryta (Streamline Social) on their virtualization tools for social applications
  - Today at 6pm
  - Davis 338A
- Anonymous feedback form still available.
- Please come talk to me!

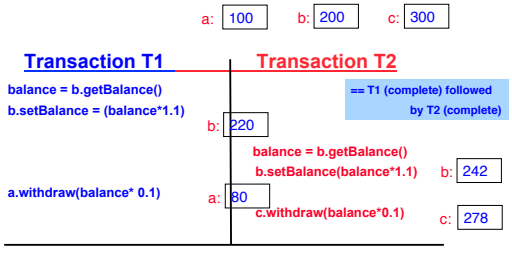
Recap: Conflicting Operations

- Two operations are said to be in conflict, if their *combined effect* depends on the order they are executed, e.g., read-write, write-read, write-write (all on same variables). NOT read-read, not on different variables.

Operations of different transactions		*Conflict*	*Reason*
read	*read*	No	Because the effect of a pair of *read* operations does not depend on the order in which they are executed
read	*write*	Yes	Because the effect of a *read* and a *write* operation depends on the order of their execution
write	*write*	Yes	Because the effect of a pair of *write* operations depends on the order of their execution

Recap: Serial Equivalence

- An interleaving of the operations of 2 or more transactions is said to be *serially equivalent* if the combined effect is the same as if these transactions had been performed sequentially (in some order).

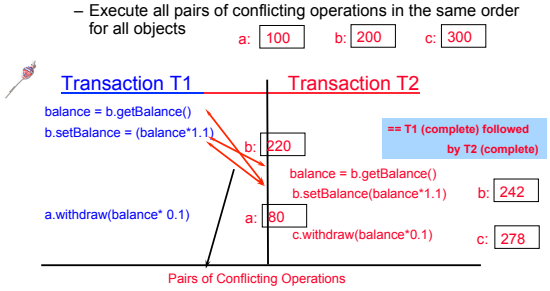


Recap: Serial Equivalence

- How to provide serial equivalence with conflicting operations?
  - Execute all pairs of conflicting operations in the same order for all objects

Recap: Serial Equivalence

- How to provide serial equivalence with conflicting operations?
  - Execute all pairs of conflicting operations in the same order for all objects



Implementing Transactions

- Two things we wanted to take care of (from the last lecture)
  - Performance: interleaving of operations
  - Failure: intentional (abort()), unintentional (e.g., process failure)
- Interleaving must satisfy serial equivalence
- What about failures?
  - Should be able to rollback as if no transaction has happened.

Handling Abort()

- What can go wrong?

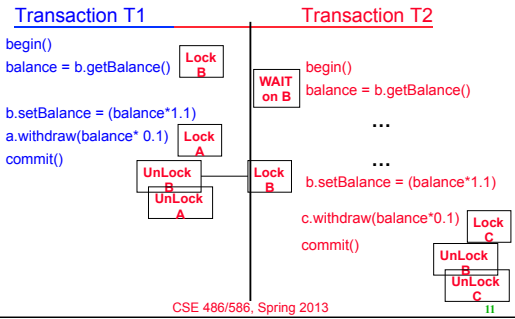
Transaction *V*: *a.withdraw(100); b.deposit(100)*		**Transaction *W*:** *aBranch.branchTotal()*	
a.withdraw(100);	$100		
		total = a.getBalance()	$100
b.deposit(100)	$300		
		total = total+b.getBalance() *total = total+c.getBalance()* ...	$400

Strict Executions of Transactions

- Transactions should *delay both their read and write operations* on an object
  - Until all transactions that previously wrote that object have either committed or aborted
- How do we implement serial equivalence & strict executions? Many ways
- One example: optimistic concurrency control: optimistically perform a transaction → if it's OK to commit then commit → if not, abort and retry
  - Examples: Dropbox, Google Docs, Wikipedia, Dynamo, etc.
- We'll see how to do this with locks

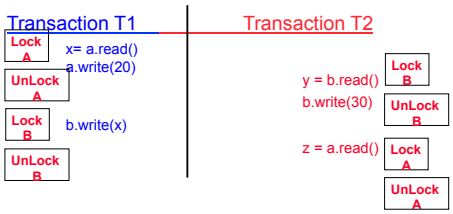
Using Exclusive Locks

- Exclusive Locks



How to Acquire/Release Locks

- Can't do it naively



Using Exclusive Locks

- Two phase locking
  - To satisfy serial equivalence
  - First phase (growing phase): new locks are acquired
  - Second phase (shrinking phase): locks are only released
  - A transaction is not allowed to acquire any new lock, once it has released any one lock
- Strict two phase locking
  - To handle abort() (failures)
  - Locks are only released at the end of the transaction, either at commit() or abort()

Summary

- Strict Execution
  - Delaying both their read and write operations on an object until all transactions that previously wrote that object have either committed or aborted
- Strict execution with exclusive locks
  - Strict 2PL

Acknowledgements

- These slides contain material developed and copyrighted by Indranil Gupta (UIUC).