
# Transactions: Over used or just misunderstood?

Mark Little

# Overview

- Transaction fundamentals
  - What is a transaction?
  - ACID properties
  - Recovery (and why you should care!)
- Why people turn away from transactions
- Why you should not try to roll your own!
- Pseudo transactions and their friends
- Compensating transactions
- When to use transactions and when not to use them



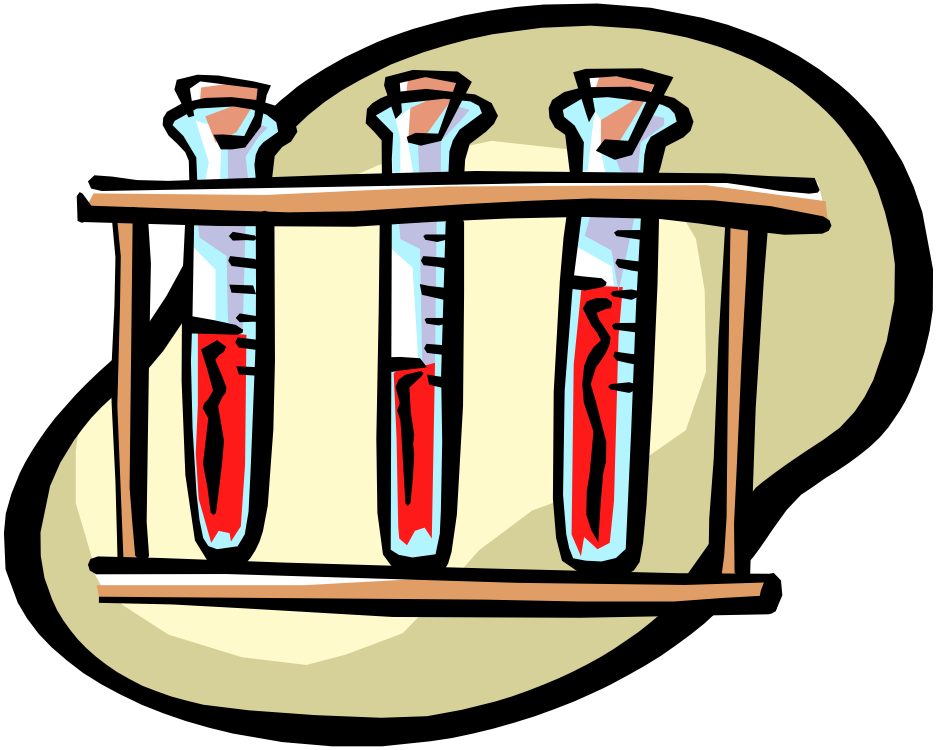
# What is a transaction?

- “Transaction” is an overloaded term
- Mechanistic aid to achieving correctness
- Provides an “all-or-nothing” property to work that is conducted within its scope
  - Even in the presence of failures
- Ensures that shared resources are protected from multiple users
- Complexity is hidden by the transaction system



# ACID Properties

- Atomicity
- Consistency
- Isolation
- Durability





# Atomicity

- Within the scope of a transaction
  - all changes occur together OR no changes occur
- Atomicity is the responsibility of the Transaction Manager

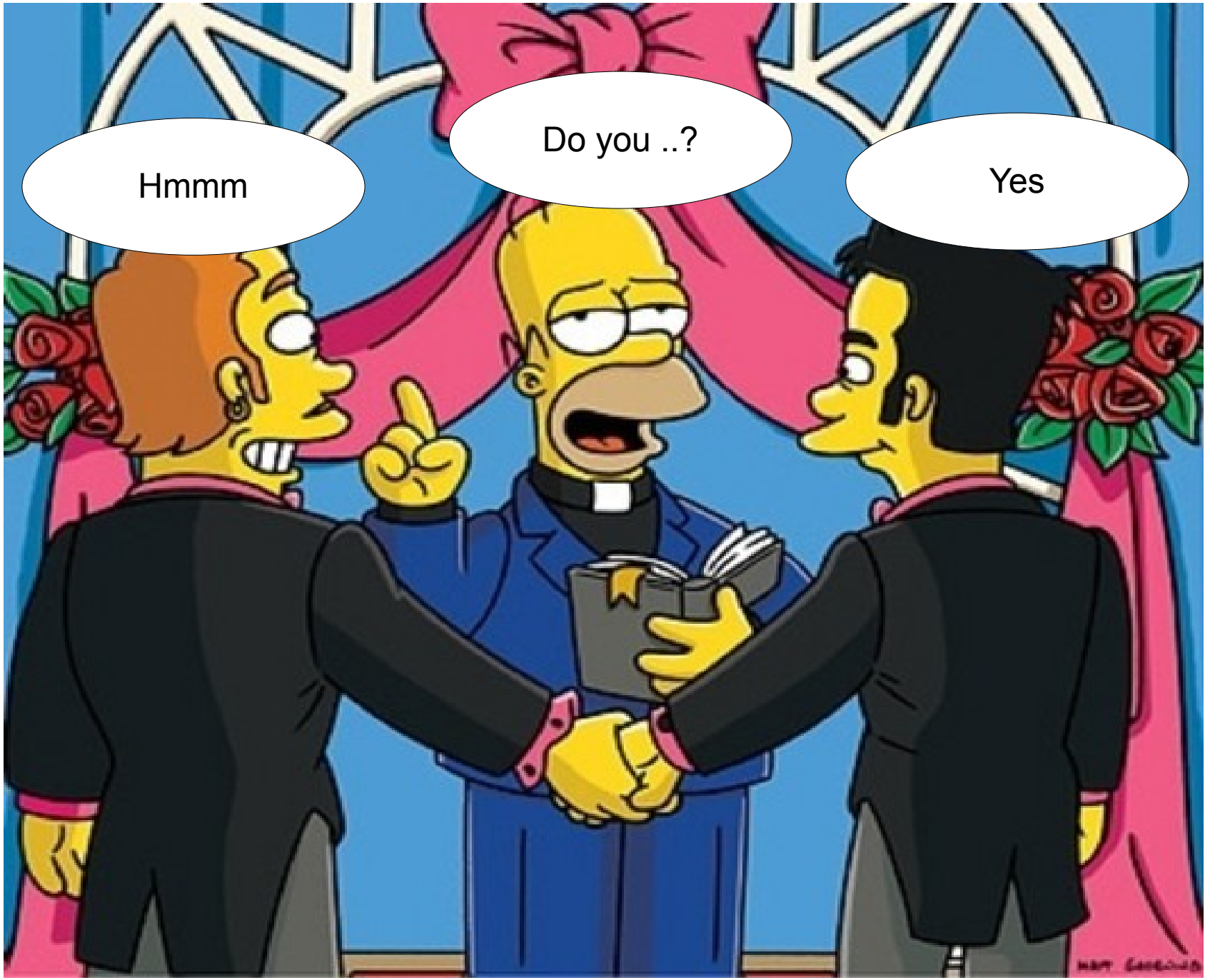
- For example - a money transfer
  - debit removes funds
  - credit add funds
  - no funds are lost!



# Two-phase commit

- Required when there are more than one resource managers (RM) in a transaction
- Managed by the transaction manager (TM)
- Uses a familiar, standard technique:
  - marriage ceremony - Do you? I do. I now pronounce ..
- Two - phase process
  - voting phase - can you do it?
    - Attempt to reach a common decision
  - action phase - if all vote yes, then do it.
    - Implement the decision




Hmmm
Do you ..?
Yes

# The “ins and outs” of Commit

- Most descriptions of the two phase commit protocol focus on the mechanism for achieving consensus
  - voting process managed by a transaction coordinator
  - first phase allows resource managers to checkpoint their work with the intention to commit and report whether that checkpoint was successful
  - the resource manager is **in doubt** or **uncertain**, a state that will be maintained indefinitely until the coordinator communicates an outcome
  - coordinator aggregates the responses of the resource managers and if all resource managers have voted to commit, issues a commit command to each. Otherwise, it issues the rollback command.



# Consensus is not enough

- Voting, by itself, is not sufficient to guarantee correctness in the presence of failures
- Coordinator must
  - maintain a transaction log that may be used to reconstruct the state prior to start of transaction
  - be able to to communicate the resolution strategy to each of the participants
- At times coordinator may need to contact resource managers directly
  - Some variations of protocol require resource managers to contact coordinator
- **Two-phase protocol alone does not provide ACID guarantees!**



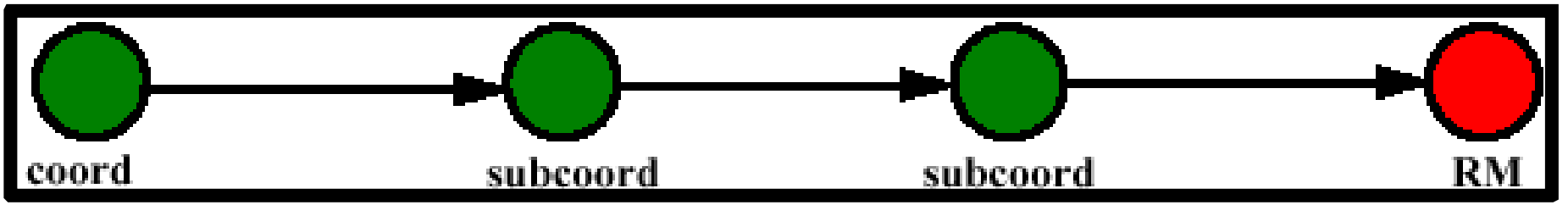
# Heuristics

- Two-phase commit protocol is blocking in order to guarantee atomicity
  - Participants may be blocked for an indefinite period due to failures
- To break the blocking nature, prepared participants may make autonomous decisions to commit or rollback
  - Participant *must* durably record this decision in case it is eventually contacted to complete the original transaction
  - If the decision differs then the coordinator's choice then a possibly non-atomic outcome has happened: a *heuristic outcome*, with a corresponding *heuristic decision*
- Heuristics cannot be resolved automatically
  - But most TPMs will retain as much information as possible to aid resolution



# 2PC: optimizations

- one phase commit
  - no voting if transaction tree is single branch




One Phase Commit

- "read-only"
  - ✓ resource doesn't change any data
  - ✓ can be ignored in second phase of commit



# Consistency

- Transactions scope a set of operations
- Consistency can be violated within a transaction
  - Allowing a debit for an empty account
  - Debit without a credit during a Money Transfer
  - Delete old file before creating new file in a copy
- transaction must be correct according to application rules
- Begin and commit are points of consistency

- Consistency preservation is a property of a transaction, not of the TP system (unlike the A, I, and D of ACID)




# Isolation

- Transaction must operate as a black box to other transactions
  - Some caveats
- Multiple programs sharing data requires concurrency control
  - Locking (one at a time access)
  - Versioning (each program has own copy)
- When using transactions
  - programs can be executed concurrently
  - BUT programs appear to execute serially
- Optimistic and pessimistic



# Durability

- When a transaction commits, its results must survive failures
  - Must be durable recorded prior to commit
  - System waits for disk ack before acking user
- If a transaction rolls back, changes must be undone
  - Before images recorded
  - Undo processing after failure
- Durability is probabilistic
- Durability can be implemented in a number of ways



# Fault tolerance

- Why is it critical?
  - Failures happen
    - Insurance policy
  - Transaction system uses log to drive outcomes
- Beware
  - Some transaction services don’t log
    - To be avoided for real applications
- Please don't encourage transaction usage without logging!
  - There are caveats ...




SERGIO LEONE
CLINT EASTWOOD
ELI WALLACH
LEE VAN CLEEF
THE GOOD
THE UGLY
AND THE BAD

# Anti-transaction sentiments

- “They add overhead for very little benefit.”
  - “Failures don't happen that often.”
    - “What's the worst that can happen?”
  - They don't scale.
- “The are hard to use and understand.”
- “I don't need a database or XA.”
- “Transaction monitors are too expensive.”
  - “As well as too bloated.”
- “I don't need atomicity.”
  - “I need consensus but not the overhead of 2PC.”
  - “I'm not interested in distributed transactions!”

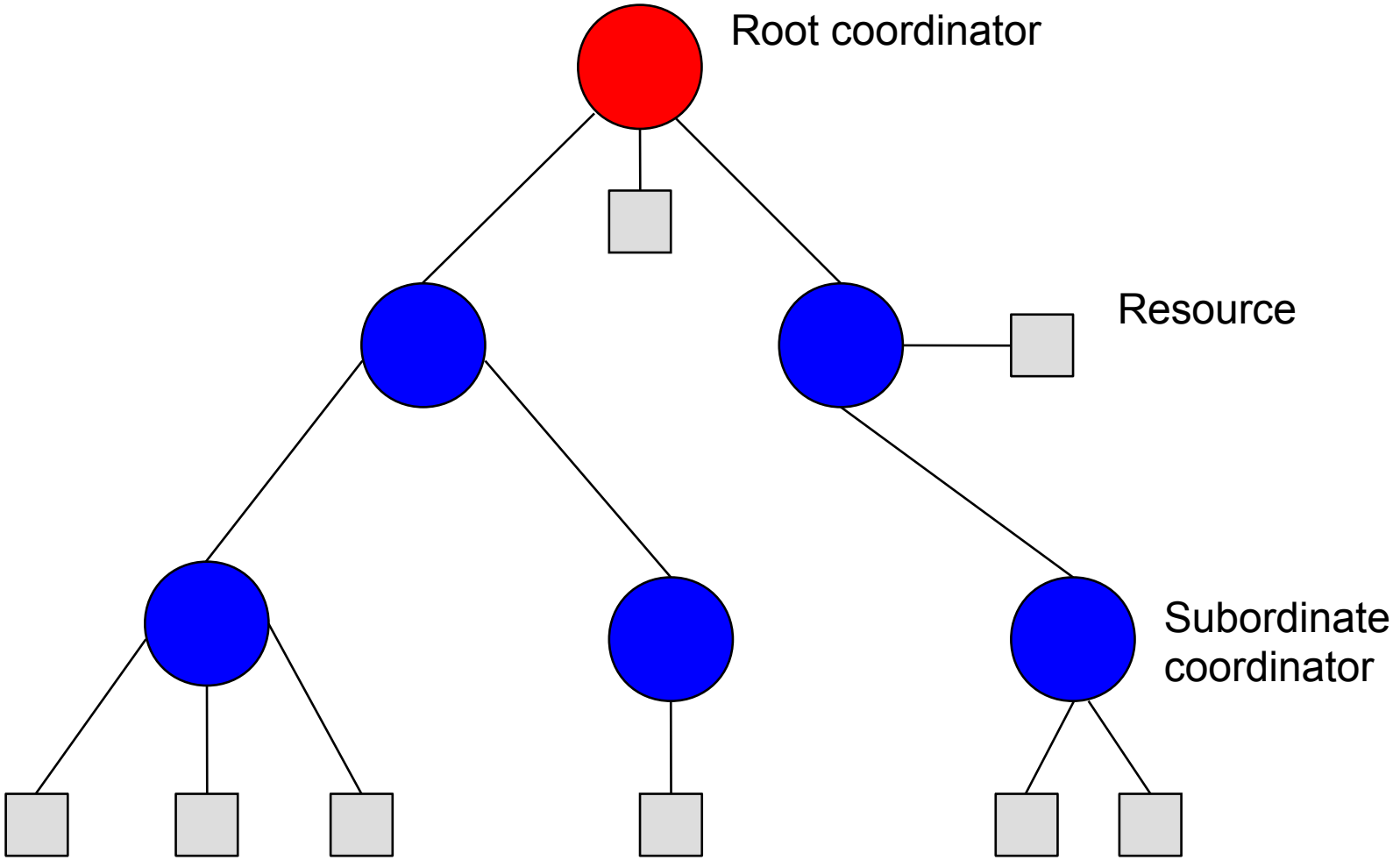


# “They add overhead for no benefit”

- Many customers regularly run with multiple one-phase only participants in the same transaction
  - And still get atomicity and recoverability
  - Because failures are rare
- Lack of education on the problem
  - Often surprised when risks are explained
    - http://news.cnet.com/8301-30685_3-10370026-264.html
  - But still want atomicity and recoverability with no impact on performance
    - Plus want to keep using same multiple one-phase participants
- Try achieving consensus reliably without a multi-phase protocol!
  - Other protocols exist, e.g., gossip
  - Rolling your own is harder than it sounds



# Consider “distributed” consensus






# Pseudo-transactions

- Let's pretend that resources are 2PC
- Let's make the application responsible for logging
- Let's make the application responsible for recording undo work
- Let's make the application responsible for atomicity
- Let's make the application responsible for driving recovery
- Let's hope that concurrent accesses don't occur
- Let's hope that cascading rollback doesn't occur
- Let's hope!



# “Too expensive and bloated”

- OSS implementations have been improving
  - Free is pretty cheap!
- OSS TPMs with support for recovery now exist
  - JBossTS (ex HP-TS) now in OSS
    - Limited footprint
- RMs support for 2PC
  - MySQL
  - Derby (aka CloudScape)
  - Various JMS implementations



# “I don't need atomicity”

- Forward compensation transactions may offer a medium-term solution
  - Do-undo
- The basis for all of the Web Services transactions specifications and standards
  - Extended transactions
- May provide a viable upgrade path for multiple 1PC participants too
  - Not quite ACID, but better than manual/ad hoc



# “I don't need a database or XA”

- Transactions don't require a database
  - Databases are one of the reasons people equate 2PC with all ACID semantics
- Transactions existed before XA
  - Participants do not have to be XA-aware
  - 2PC is independent of XA
- Durability could be through replication
  - It's all probabilistic anyway
  - File system is also sufficient
- Recoverable transactions



# “They're hard to use and understand”

- They don't have to be intrusive
  - Annotations based
- Most applications using transactions don't know they are using transactions
  - JEE, Web Services
- Software Transactional Memory
- Aspect Oriented Programming



# When to use transactions

- When you need ACID semantics!
- Or ...
  - When you have a need to guarantee consensus in the presence of failures
    - Consensus is not easy to achieve when failures happen
    - Local or distributed cases
  - When you need isolation and consistency across failures
- Relaxing ACID semantics is possible with some TPMs
- Recoverable transactions may be sufficient
  - Automatically promote when needed



# When not to use transactions

- When all you want is consensus
  - Without reliability
  - Even with reliability, some transaction systems may be overkill
    - “Good enough” may be sufficient
- When you will only ever have a single resource
  - Most modern databases come with them build in
  - Though 1PC optimizations can make overhead negligible
- When guarantees are too strong for your needs



# When not to fudge the issue!

- If you want ACID semantics then use an implementation that provides them all
- If you want to relax the semantics then use an implementation that allows that to happen
- Don't use an “ACID” transaction system that doesn't provide for ACID
  - Stay away from pseudo-transactions!
- Don't expect ACID guarantees when your application doesn't uphold its end of the contract
  - Multiple 1PC resources are asking for trouble!

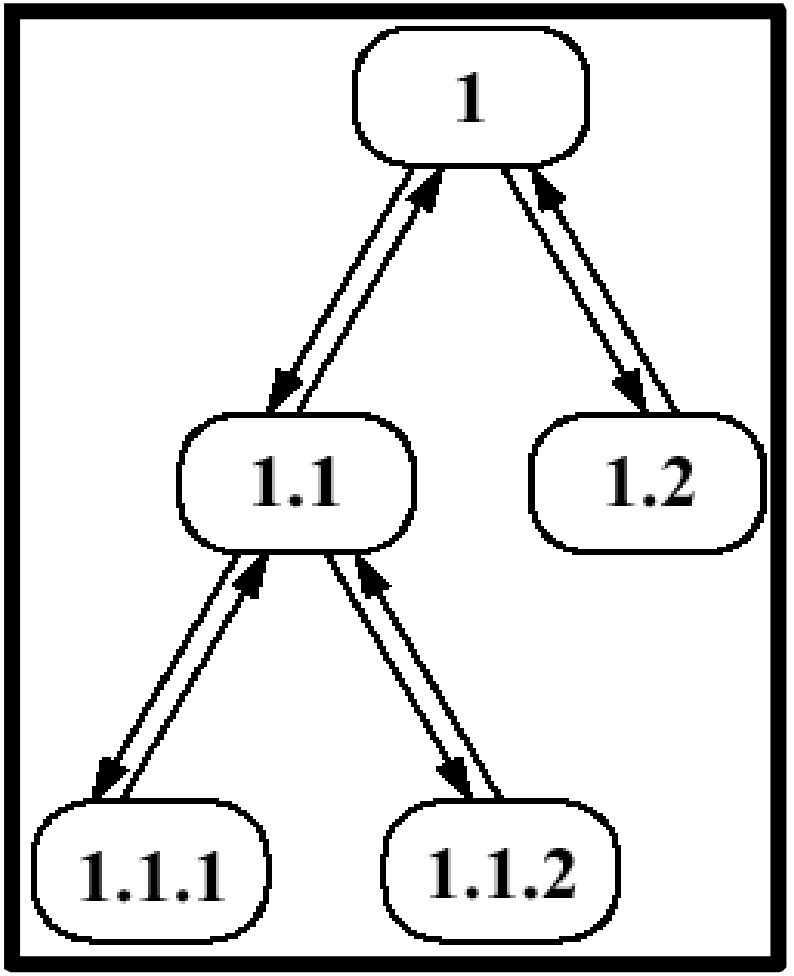


# Transactions and multi-core

- Fault tolerance is important in local and distributed systems
- Multi-core systems encourage concurrent applications
  - Sharing data
- Cores may fail independently
- Cores may have on-chip persistence
- Similar problems to those which encouraged distributed transactions



# Nested transactions




- A transaction is *nested* when it executes within another transaction
- Nested transactions live in a tree structure
  - parents
  - children
- Implement modularity and containment
  - Relax ACID



# Relaxing isolation

- Internal isolation or resources should be a decision for the service provider
  - E.g., commit early and define compensation activities
  - However, it does impact applications
    - ✓ Some users may want to know a priori what isolation policies are used
- Undo can be whatever is required
  - Before and after image
  - Entirely new business processes



# Relaxing atomicity

- Sometimes it may be desirable to cancel some work without affecting the remainder
  - ✓E.g., prefer to get airline seat now even without travel insurance
- Similar to nested transactions
  - ✓Work performed within scope of a nested transaction is provisional
  - ✓Failure does not affect enclosing transaction
- However, nested transactions may be too restrictive
  - ✓Relaxing isolation



# Conclusions

- Imagine doing this ad hoc!
- Atomicity requires durability if failures are to be survived
  - Consensus requires a protocol that terminates
    - Typically in finite time
- Failures require recovery
  - Typically automatic
- Distributed nature complicates things
- Two-phase commit not just for distributed cases
- Many cases of 1 resource becoming 2 or more
  - Transactions take care of that transition opaquely