Fault tolerance with transactions: past, present and future

Dr Mark Little

Technical Development Manager, Red Hat

Overview

- **Fault tolerance**
- **Transaction fundamentals**
  - What is a transaction?
  - ACID properties
- **Distributed transactions**
- **Web Services**

Fault tolerance

- **Machines and software fail**
  - Fundamental universal law
  - Things get better with each generation, but still statistically significant
- **Failures of centralized systems difficult to handle**
- **Failures of distributed systems are much more difficult**

Fault tolerance techniques

- **Replication of resources**
  - Increase availability
    - Probability is that a critical number of resources remain operational
    - “Guarantee” forward progress
  - Tolerate programmer errors by heterogeneous implementations
- **Spheres of control**
  - “Guarantee” no partial completion of work in the presence of failures

Affect of time

- **Fault tolerance has always been extremely important**
- **Back in the 1980's many different efforts**
  - Emerald, Argus, Arjuna, Camelot/Avalon, Isis, Horus etc.
  - Mostly concentrated around distributed systems
    - Centralized system as degenerate case
- **1990's saw standardization of distributed systems**
  - Ansa, DCE, COM/DCOM, CORBA, J2EE

Is there still research potential?

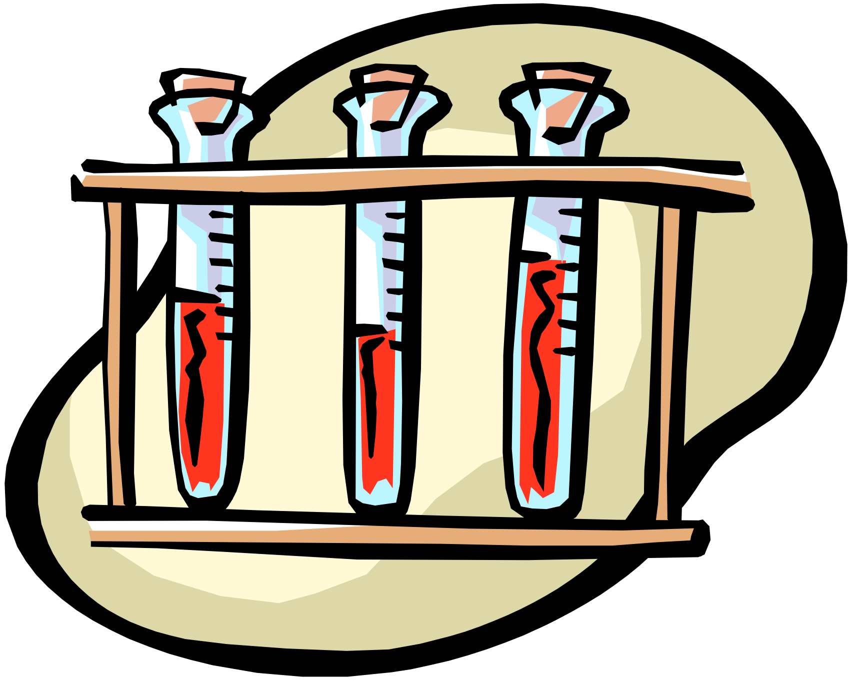
- **What we do is changing**
- **How we do it is changing**
- **Paradigm shifts occurring frequently**
  - Web Services
  - Grid Computing
  - Mobile Computing
  - Large Scale Computing
- **These often require new techniques for fault tolerance**
  - Some research efforts in environments like these started decades ago

What is a transaction?

- **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**

ACID Properties

- Atomicity
- Consistency
- Isolation
- Durability



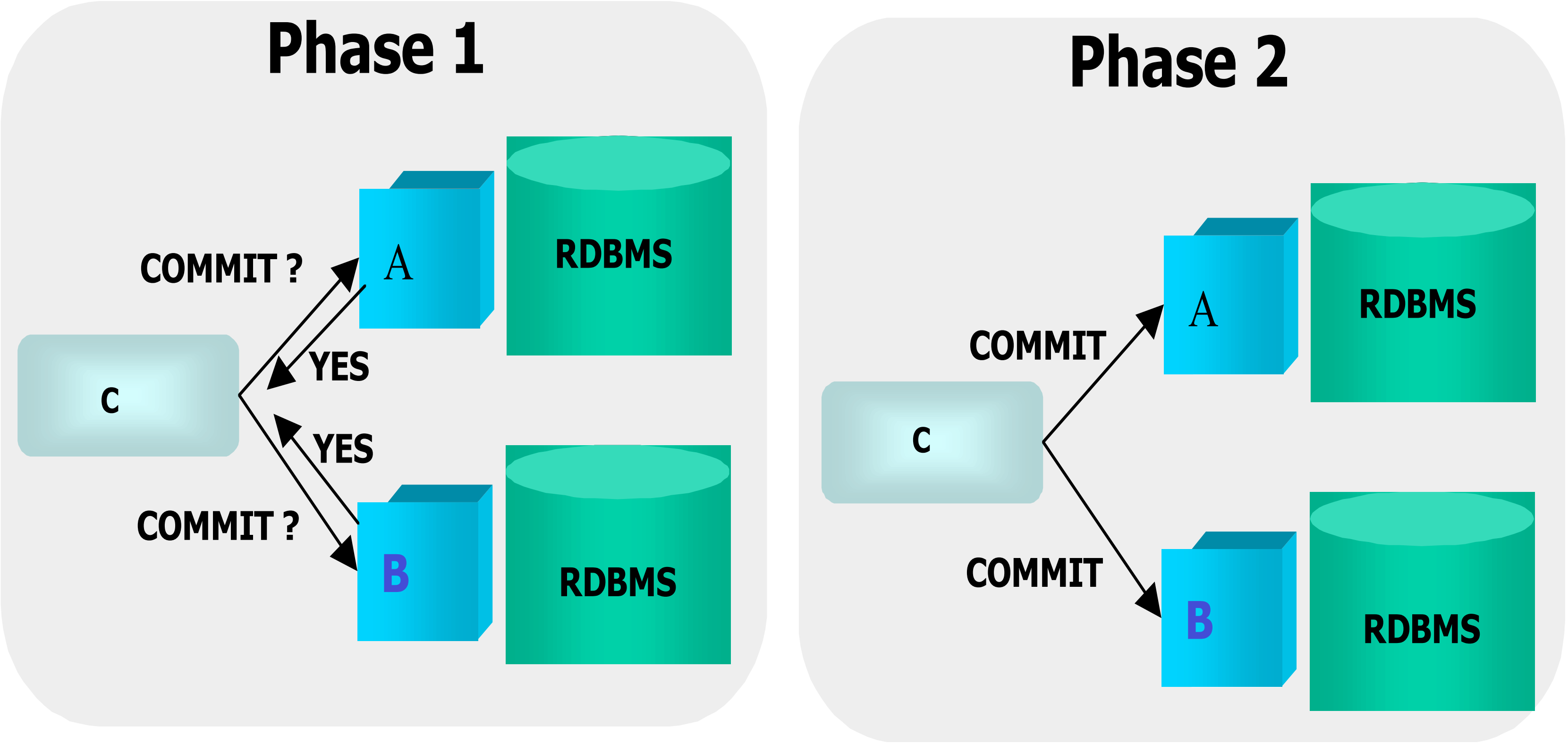
Atomicity

- **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

Two-phase commit

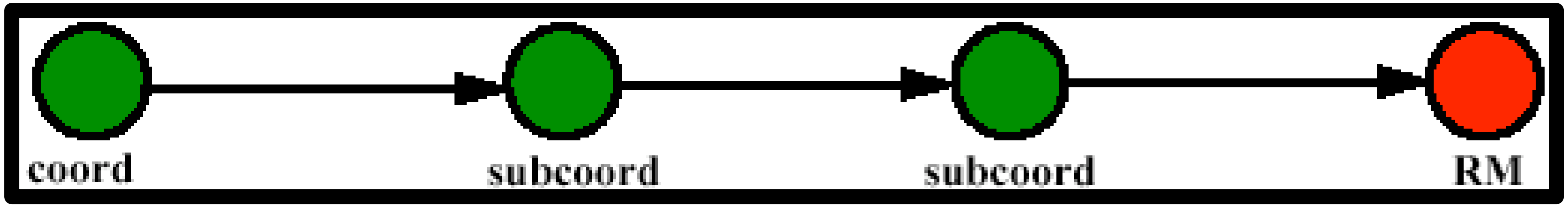


Handling failures

- **Presumed Abort Strategy**
  - can be stated as « when in doubt abort »
  - any failure prior the commit phase lead to abort the transaction
- **A coordinator or a participant can fail in two ways**
  - it stops running (crashes)
  - it times out waiting for a message it was expecting
- **A recovered coordinator or participant uses information on stable storage to guide its recovery**

2PC: optimizations

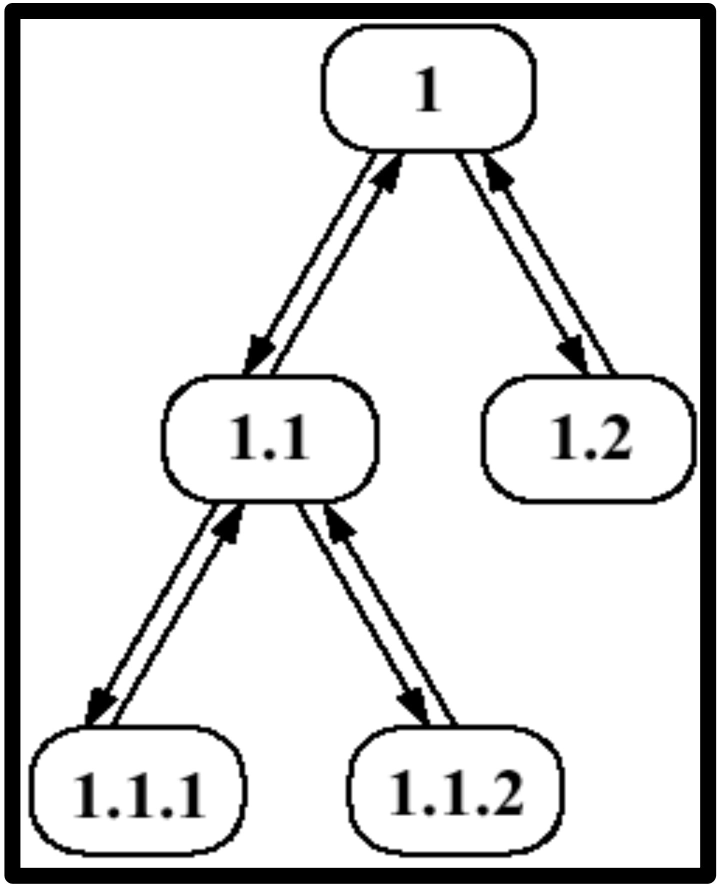
- **one phase commit**
  - no voting if transaction tree is single branch



One Phase Commit

- “read-only”
  - ✓ resource doesn’t change any data
  - ✓ can be ignored in second phase of commit

Nested transactions

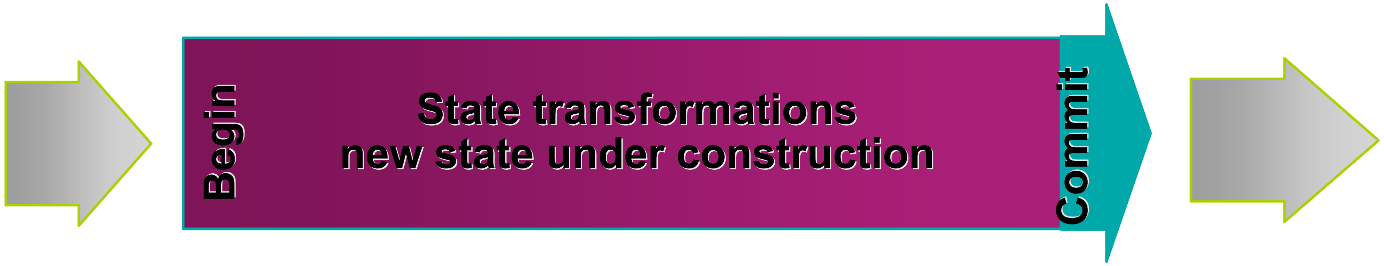


- **a transaction is *nested* when it executes within another transaction**
- **nested transactions live in a tree structure**
  - parents
  - children
- **implement modularity and containment**

Consistency

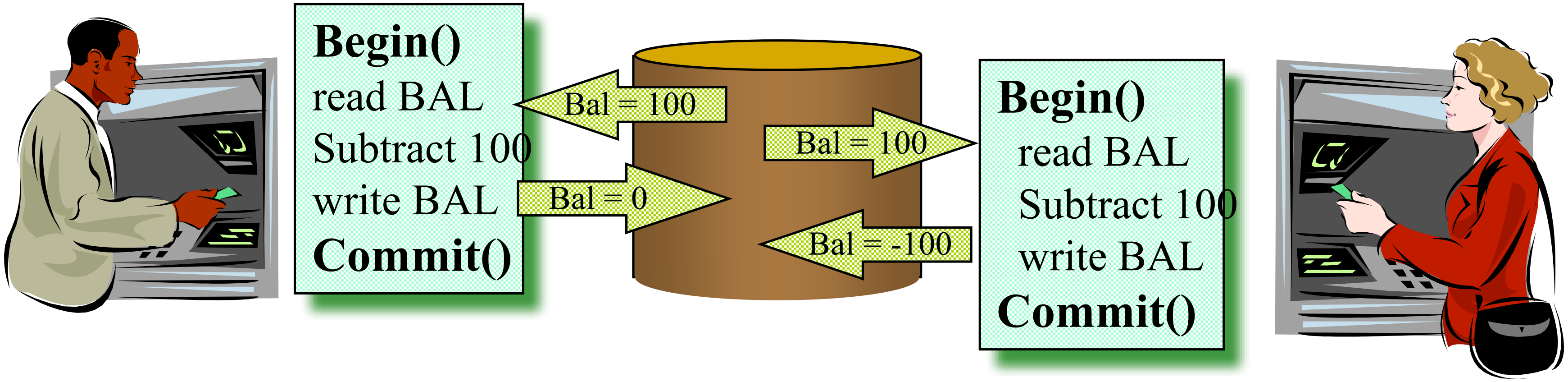
- Transactions scope a set of operations
- Consistency can be violated within a transaction
  - Allowing a debit for an empty account
  - Debit without a credit during a Money Transfer
  - Delete old file before creating new file in a copy
- transaction must be correct according to application rules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

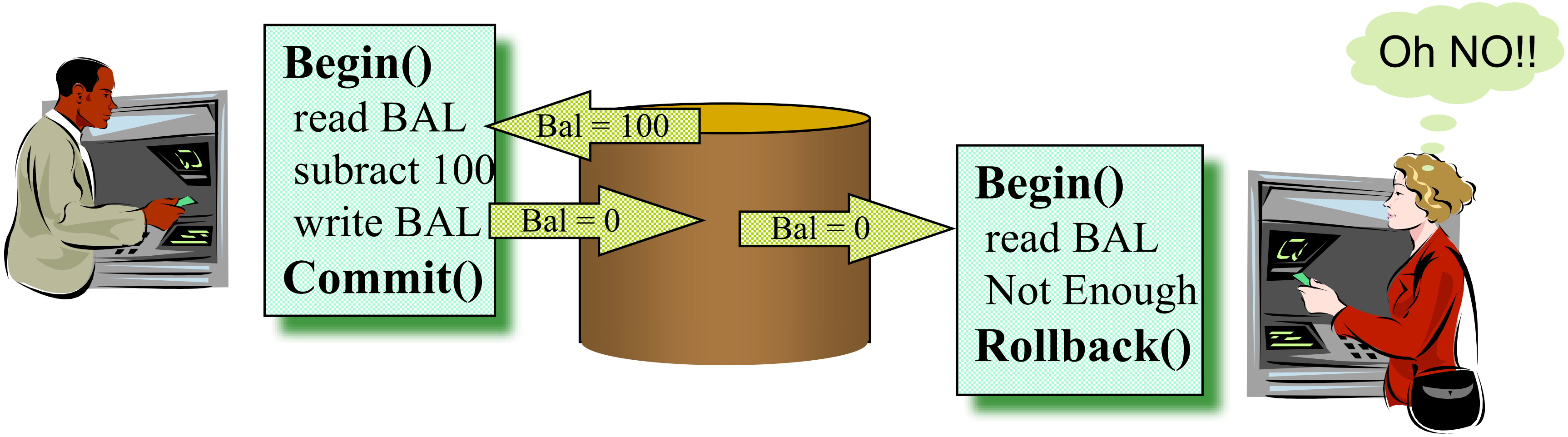
- **Running programs concurrently on same data can create concurrency anomalies**
  - the shared checking account example



Isolation

- **Transaction must operate as a black box to other transactions**
- **Multiple programs sharing data requires concurrency control**
- **When using transactions**
  - programs can be executed concurrently
  - BUT programs appear to execute serially

Isolation



Begin()
read BAL
subract 100
write BAL
Commit()
Bal = 100
Bal = 0
Bal = 0
Begin()
read BAL
Not Enough
Rollback()
Oh NO!!

Durability

- **When a transaction commits, its results must survive failures**
  - must be durably recorded prior to commit
  - system waits for disk ack before acking to user
- **If a transaction rolls back, changes must be undone**
  - before images recorded
  - undo processing after failure

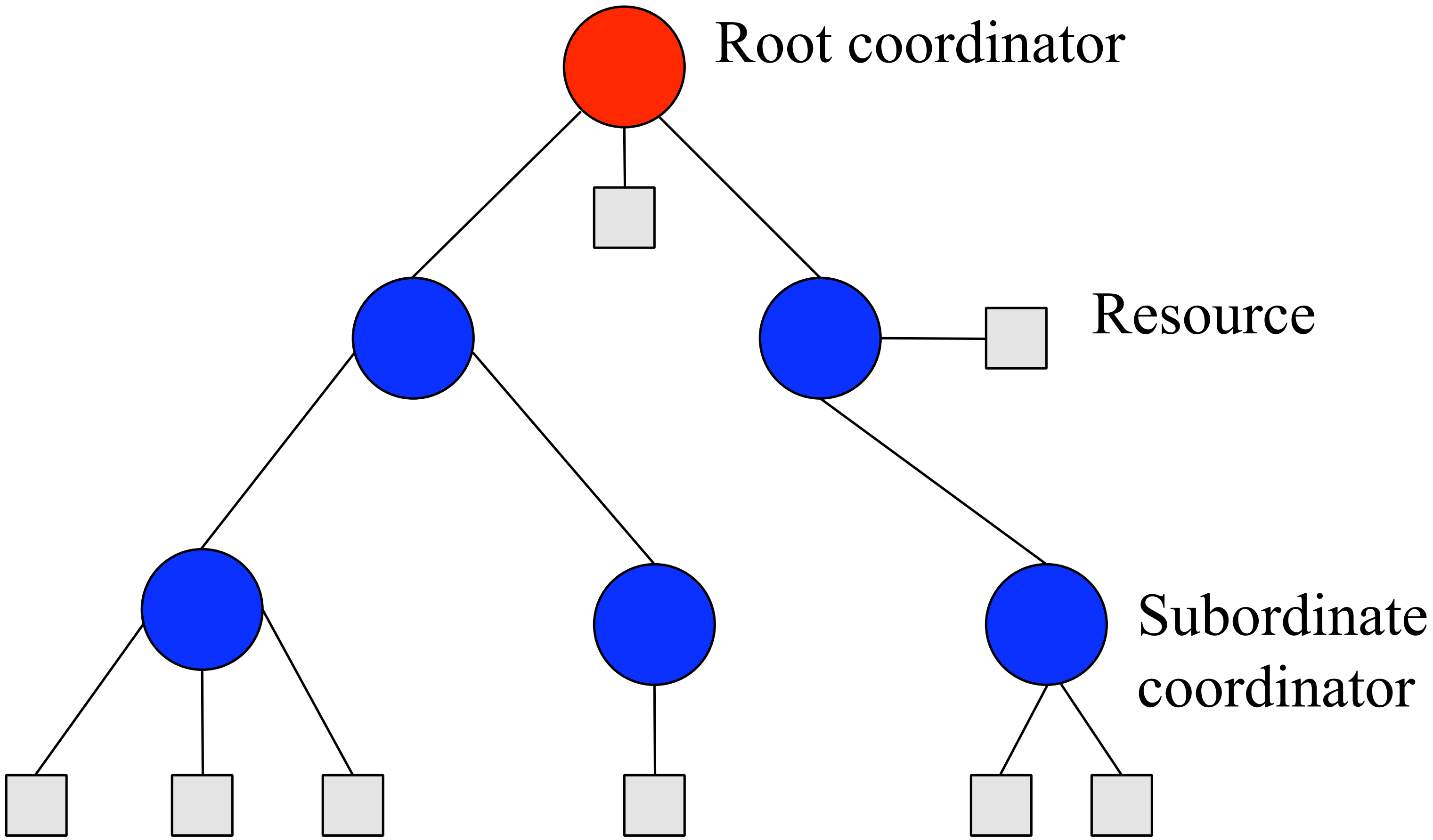
Heuristics

- **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*.

Interposition

- **Allows a subordinate coordinator to be created**
- **Interposed coordinator registers with transaction originator**
  - Form tree with parent coordinator
  - Application resources register locally

Interposition



Web Services and SOA

- **Transactions today imply all ACID properties**
- **Good for “short” durations**
  - Application specific
- **Long-running transactions may impose constraints**
  - Hours, days, months, …
  - Retain resources for duration of transaction

Web Services transactions

- **Business-to-business interactions may be complex**
  - involving many parties
  - spanning many different organisations
  - potentially lasting for hours or days
- **Cannot afford to lock resources on behalf of an individual indefinitely**
- **May need to undo only a subset of work**

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

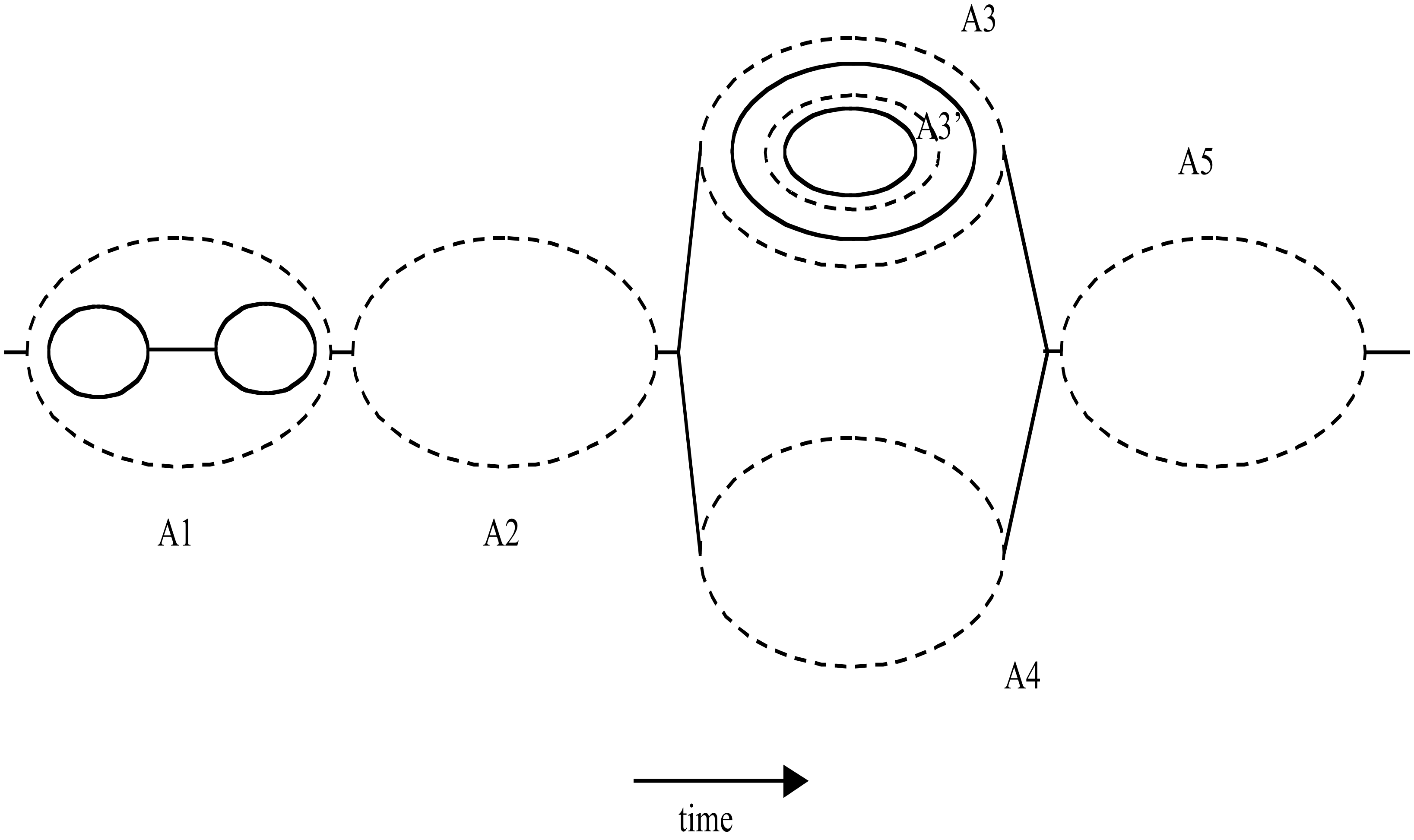
Relaxing atomicity

- **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

Structuring transactions

- **Could structure transactional applications from short-duration transactions**
  - Release locks early
  - Resulting application may still be required to appear to have “ACID” properties
    - May require application specific means to restore consistency
- **A transactional workflow system could be used to script the composition of these transactions**

Structuring transactions



Extended transaction models

- **There are a number of such models**
  - Sagas
  - Compensations
  - Epsilon Serialisability
  - Versioning Schemes
  - Nested top-level transactions
  - Open-nested transactions
  - Glued transactions
  - Coloured actions

Future directions

- **One size does not fit all!**
- **Business domains will impose different requirements on implementers**
  - Essentially construct domain-specific models
  - Real-time
- **The range and requirements for such extended models are not yet known**
  - Do not restrict implementations because we don't know what we want yet
- **Still a very active area of research and development**

Any questions?