redhat.

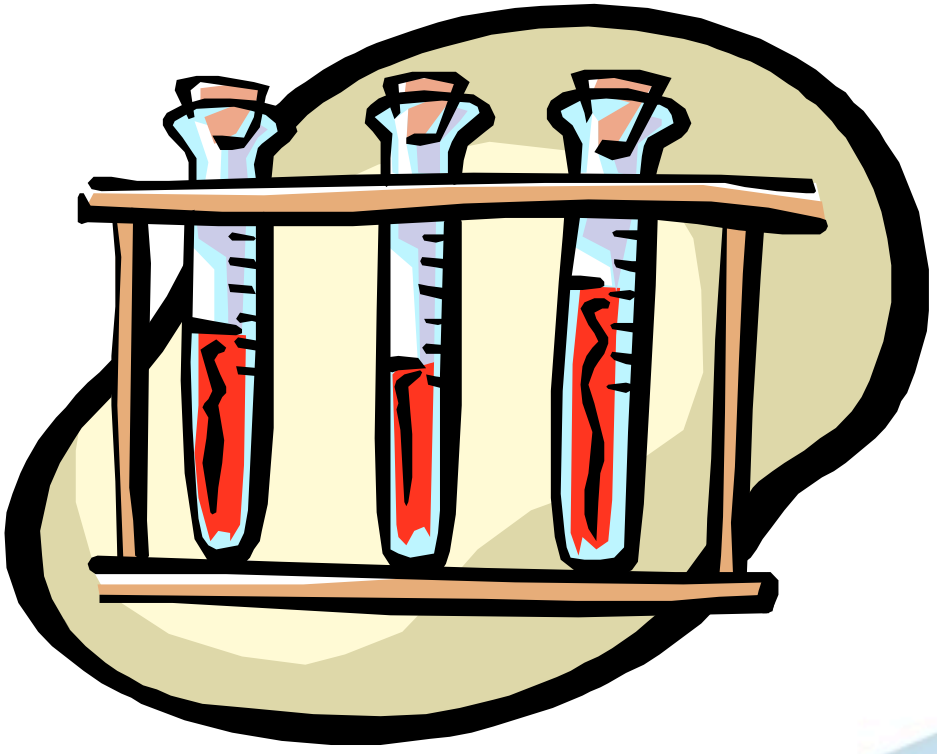
Java. Cloud. Leadership.

Transactions: Over used or just misunderstood?

Dr Mark Little
Red Hat, Inc.
25/9/2012

ACID Properties

- Atomicity
- *Consistency*
- Isolation
- Durability



Atomicity: two-phase commit

- Required when there is more than one resource managers (RM) in a transaction
- Managed by the transaction manager (TM)
- Uses a familiar two-phase technique:

Consensus is not enough

- Coordinator must:
  - maintain a transaction log
  - be able to to communicate the resolution strategy to each of the participants
- During recovery, coordinator may need to contact resource managers directly
  - Some variations of protocol require resource managers to contact coordinator

Isolation

- Programs can be executed concurrently
  - BUT they appear to execute serially

  - P1 : z := 10; x := x+1; y := y+1
  - P2 : w := 7: x := x * 2; y := y * 2

  - (z := 10 || w := 7); x := x+1; y := y+1; x := x * 2; y := y * 2
  - (z := 10 || w := 7); x := x+1; x := x * 2; y := y * 2; y := y +1

Durability

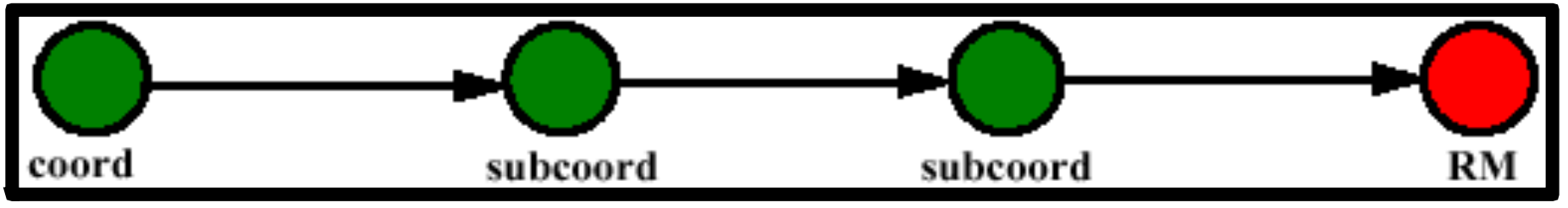
- When a transaction commits, its results must survive failures
  - Must be durably recorded prior to commit
  - System waits for disk ack before acking user
- If a transaction rolls back, changes are undone
- Durability is probabilistic
  - It does not have to be disk based!

Heuristics

- Two-phase commit protocol is blocking
- To break the blocking nature, prepared participants may make autonomous decisions to commit or rollback
  - Participant must durably record this decision
  - If the decision differs then possibly non-atomic outcome
- Heuristics cannot be resolved automatically

Optimisations

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



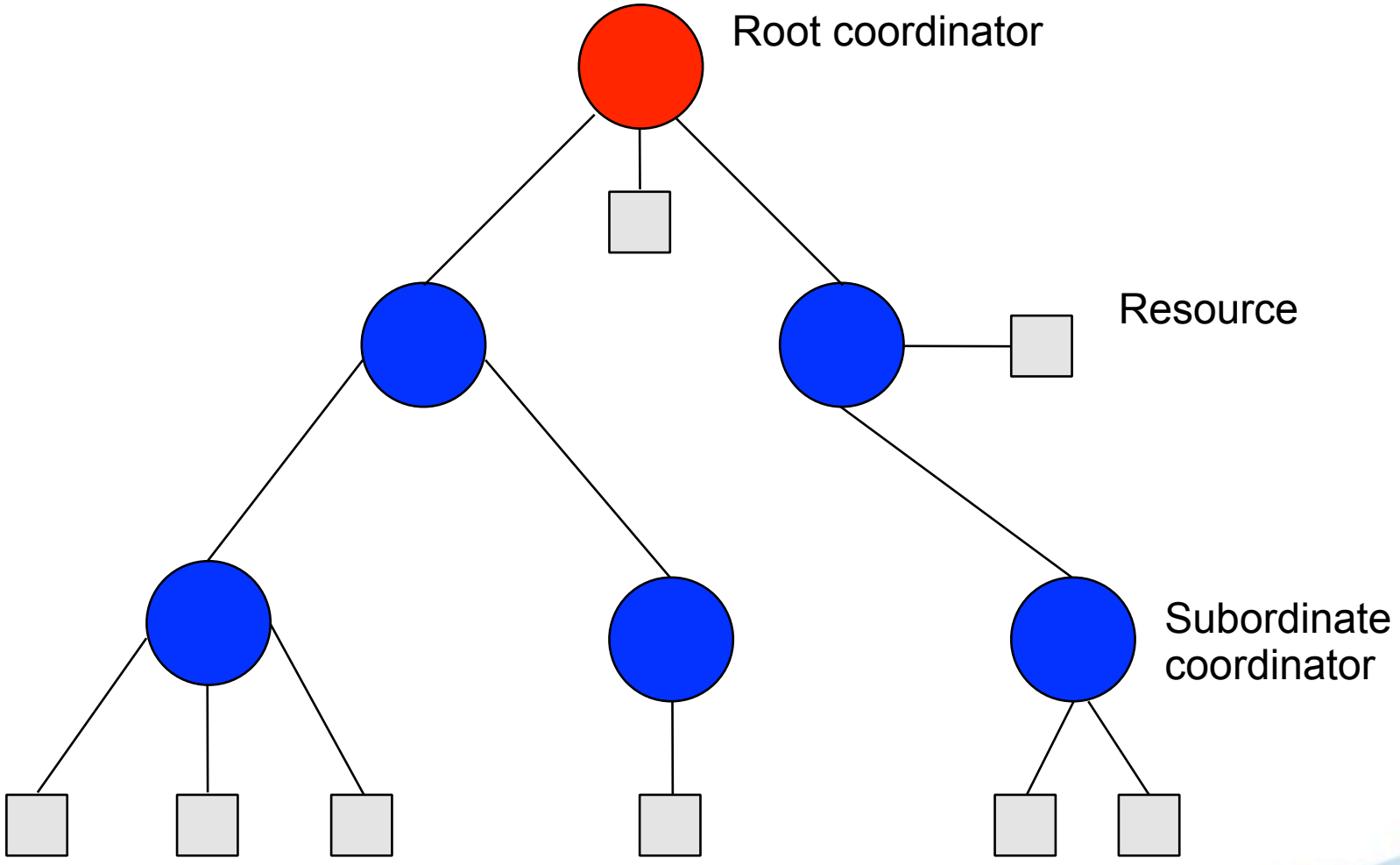
One Phase Commit

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

redhat. JBoss by Red Hat

Java. Cloud. Leadership.

Interposition



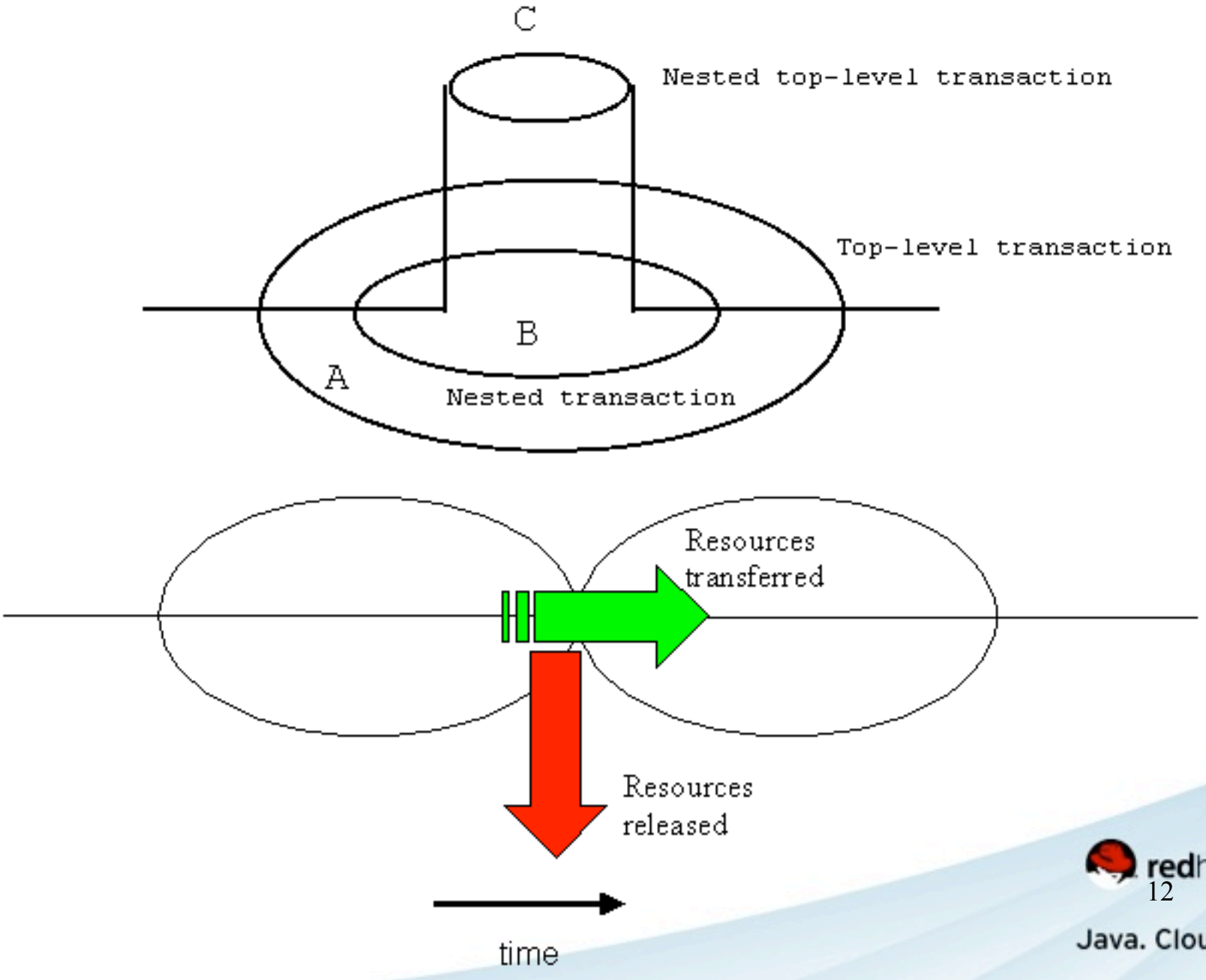
“They add overhead for no benefit”

- Many customers regularly run with multiple one-phase only participants in the same transaction
  - And appear to get atomicity and failure recovery
    - Failures don’t happen that often
- Lack of education on the problem
  - Often surprised when risks are explained
  - But still want atomicity and failure recovery with no impact on performance

“I don't always 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
- May provide a viable upgrade path for multiple 1PC participants too
  - Not quite ACID, but better than manual/ad hoc

Relaxing ACID



Java. Cloud. Leadership.

When to use transactions

- When you need ACID semantics!
- Or ...
  - When you have a need to guarantee consensus in the presence of failures
  - When you need isolation and consistency across failures
- Relaxing ACID semantics is possible
- Recoverable transactions may be sufficient

When not to use transactions

- When all you want is consensus
- When you will only ever have a single resource
  - Though 1PC optimisations can make overhead negligible

When not to cut corners

- 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
- Don't expect ACID guarantees when your application doesn't uphold its end of the contract

Conclusions

- Atomicity requires durability if failures are to be survived
- Failures require recovery
- Two-phase commit not just for distributed cases
- Extended transactions useful for new data strategies
- Many cases of 1 resource becoming 2 or more
  - Transactions take care of that transition opaquely