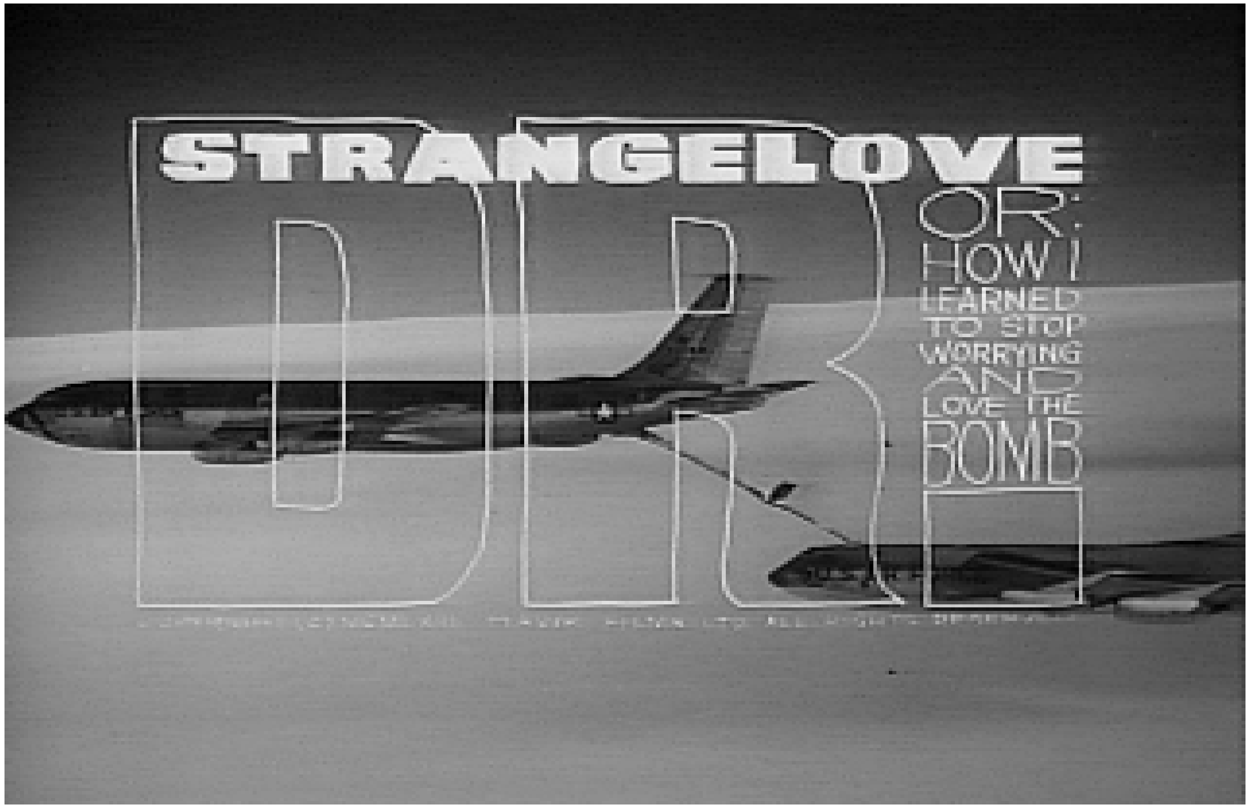
JBoss Transactions

Dr Mark Little,

Director of Standards, Development Manager

Transactions are your friend!

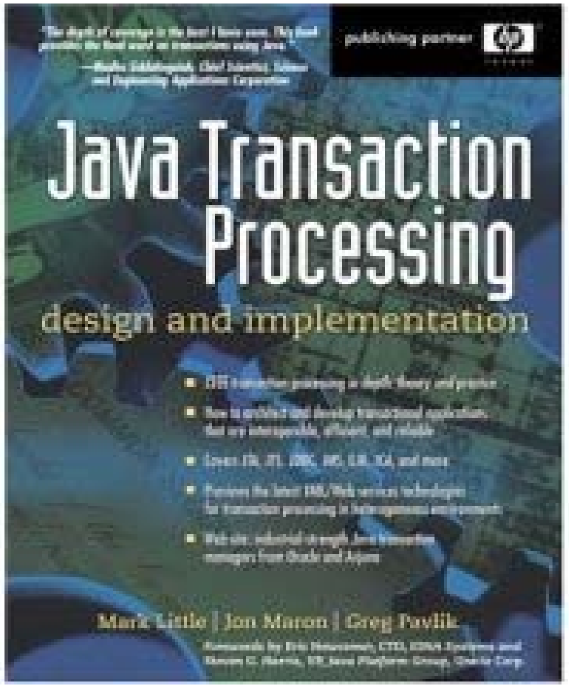


What this talk will cover

- Background
- ArjunaCore
  - ✓ Transaction engine
- JTA
  - ✓ JDBC driver
- JTS
- WS-T
- Summary

What this talk won't cover

- Transaction processing basics
  - ✓ There are enough good books out there to do the job


publishing partner

Java Transaction Processing

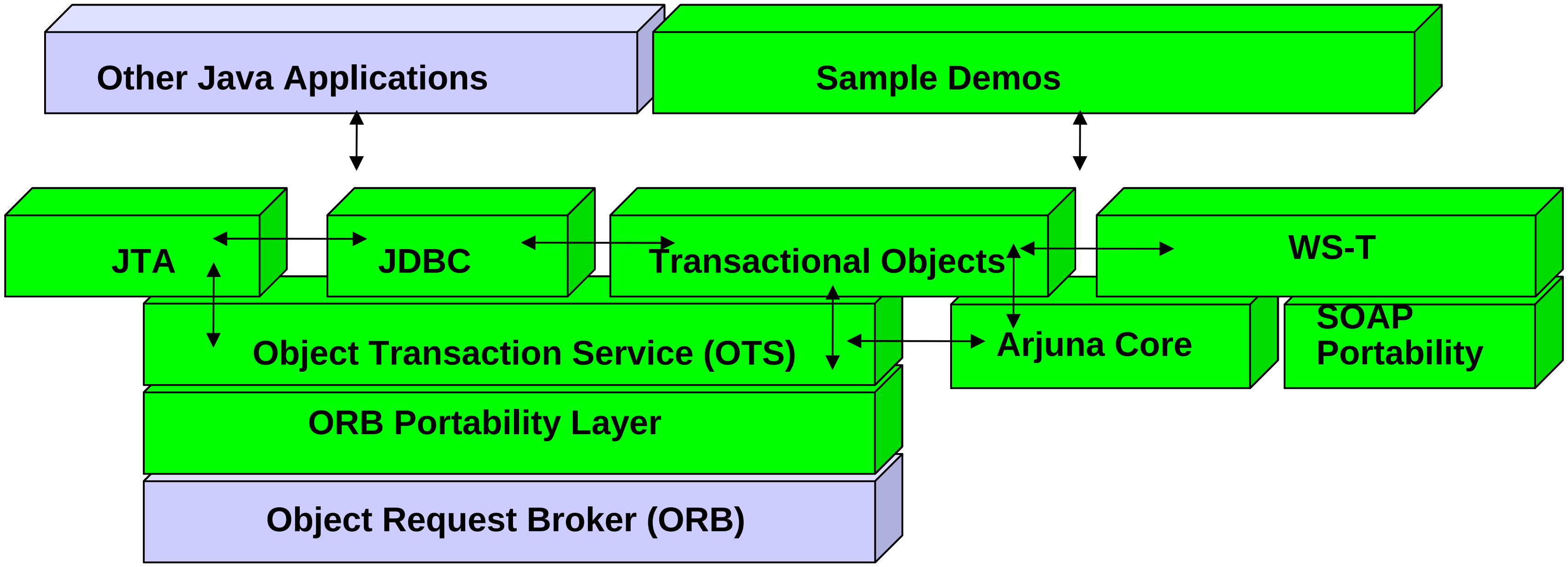
design and implementation

Mark Little | Jon Maron | Greg Pavlik

What is JBoss Transactions?

- JBoss Transactions 4.2.1
  - ✓ Next generation of JBoss transaction service
  - ✓ Based on
    - JTA 1.0.1
    - JTS 1.0 (OTS 1.4)
    - WS-Coordination, WS-Atomic Transaction, WS-Business Activity
      - ✓ Demonstrated interoperability with IBM and MSFT
  - ✓ Used at HP World for Web Services seminars
  - ✓ Licensed to TIBCO, webMethods, Mizuho and others
    - Does not require an application server to run
  - ✓ I18N and L10N

JBossTS Components



🟩 JBossTS Package

↔ Interact with each other

ArjunaCore

- Stand-alone transaction engine
- Full failure recovery
- ACID properties can be relaxed
  - ✓ Gray's matrix of transaction models
  - ✓ Does not restrict to XA
- Designed to be used stand-alone
  - ✓ Own set of APIs
- Similar to what MSFT are doing with Indigo

Failure recovery

- Automatic failure recovery daemon
  - ✓Runs periodically
  - ✓Can be driven directly
- Recover inflight transactions
- Recover resources
  - ✓different recovery mechanisms required for each resource type
  - ✓different mechanisms can be easily added

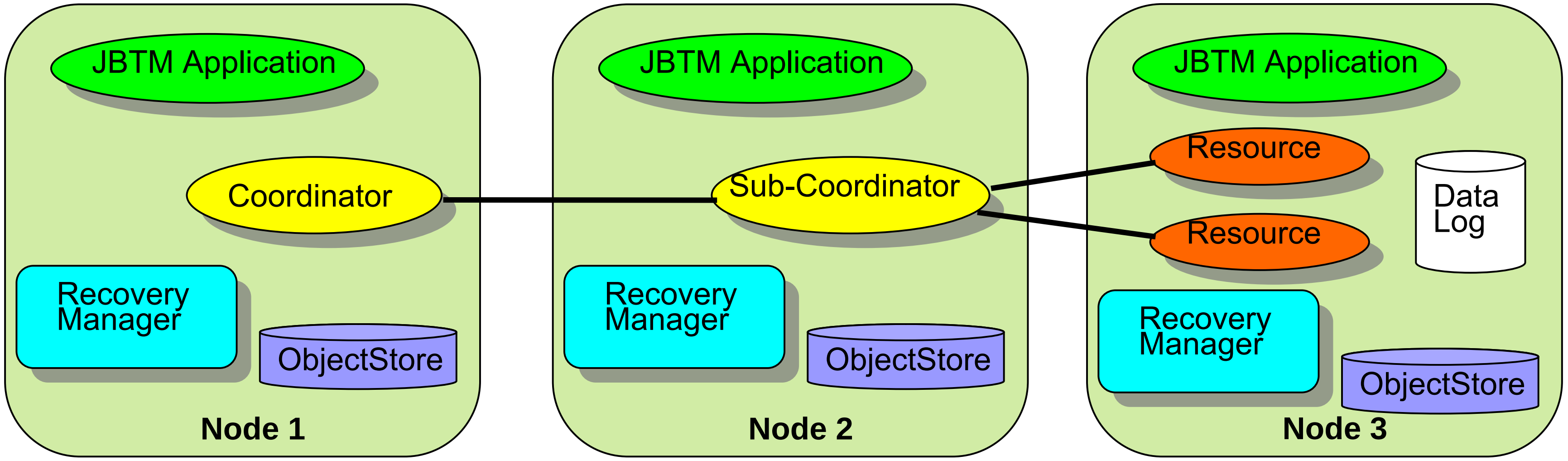
JBoss World 2006 LAS VEGAS

Why do I care about recovery?

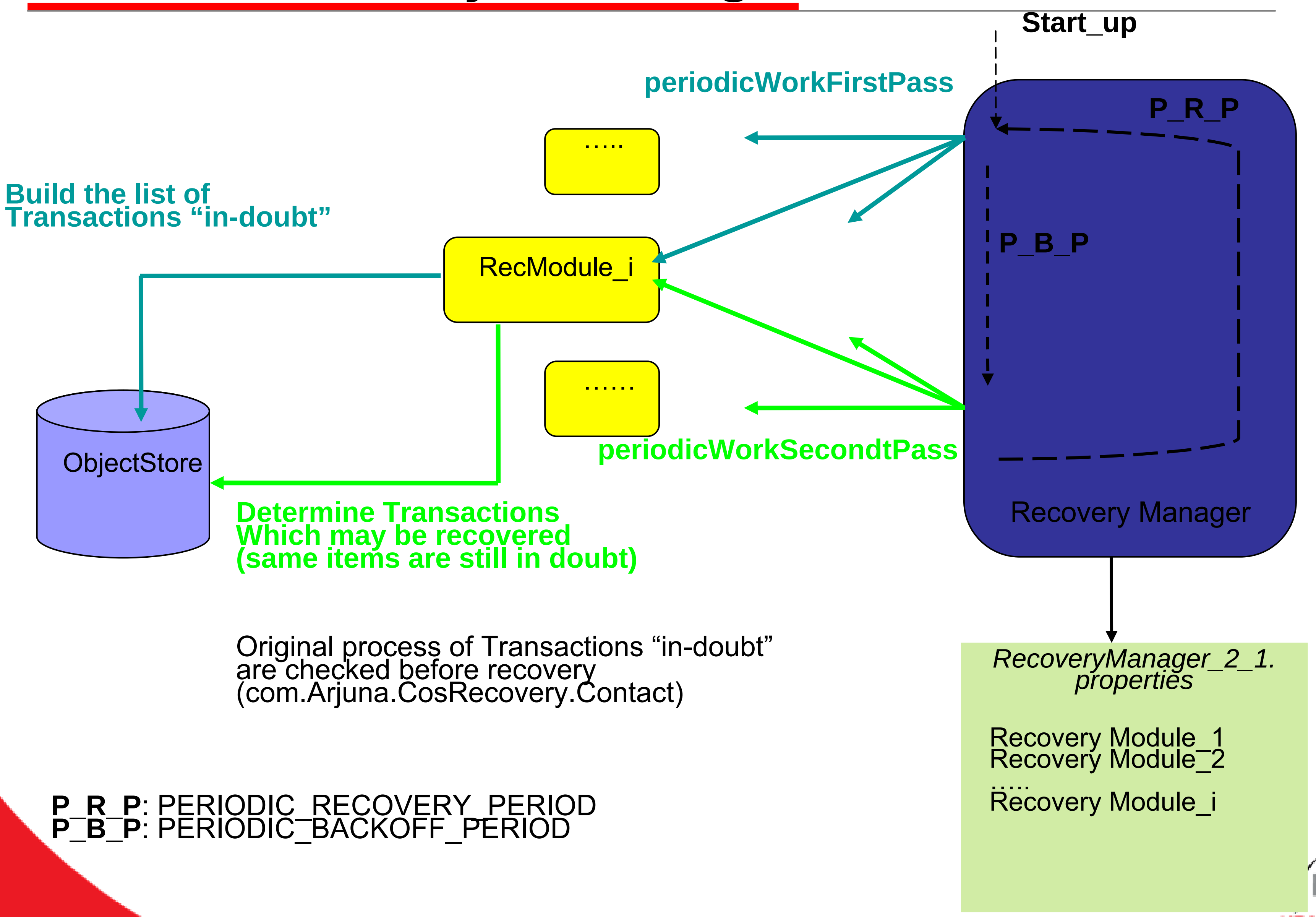
- Why is it critical?
  - ✓ Sh*t happens!
    - Insurance policy
  - ✓ Infrastructure uses log to drive atomic outcomes
- Beware
  - ✓ Many times you must enable logs
  - ✓ Some transaction services don’t log
    - To be avoided for real applications

The Recovery Manager

- Recovery requires a Recovery Manager process
  - ✓ Stand-alone
  - ✓ Embedded (application server)



The Recovery Manager in action



Original process of Transactions "in-doubt" are checked before recovery (com.Arjuna.CosRecovery.Contact)

P_R_P: PERIODIC_RECOVERY_PERIOD
P_B_P: PERIODIC_BACKOFF_PERIOD

Further features

- Many configuration options
  - ✓Transaction nesting
- Checked transactions
  - ✓Per transaction basis
- Last resource commit optimization
- Asynchronous commit protocol
  - ✓Prepare and commit
- Transaction management tools
  - ✓Heuristic resolution

JEE support

- Local and remote JTA implementations
- World’s first JTS implementation
  - ✓Used to push the OTS specification
  - ✓Completely multi-thread aware
- Portable to a number of ORBs
  - ✓E.g., Orbix 2k, JacORB, JDK ORB, ...
- Distributed failure recovery
- Sub-transaction aware resources

What is JTS?

- The Java^{TM} language mapping of the OMG's Object Transaction Service (OTS)
- supports multiple transaction models
  - ✓ flat
  - ✓ nested (optional)
- interoperability between OTS and X/Open DTP model
- flexible transaction propagation
  - ✓ implicit
  - ✓ explicit

Why do I need to know this?

- EJB™ architecture mandates JTS for interoperability
  - ✓ Actually only on-the-wire message formats required
  - ✓ Unfortunately OTS-to-OTS interoperability is still difficult to achieve
- The OTS specification is more complete than the JTA
  - ✓ Read both together to get an idea of how distributed transactions work

JTS component

- Supports distributed two-phase commit
  - ✓ One-phase commit optimisation
- Failure recovery automatically completes transactions
  - ✓ Driven from resource side as well as from transaction manager
- Fast, in-process transaction management
  - ✓ Separate transaction server possible

Nested transactions

- Optional part of JTS specification
  - ✓ few sub-transaction-aware resources
- Registered resources are only informed of transaction termination
- No two-phase commit for sub-transactions
  - ✓ can result in heuristic-like outcomes
  - ✓ Implementation specific extensions

Why use nested transactions?

- Fault isolation
  - ✓Sub-transaction work can be rolled back independently of enclosing transaction
    - can try alternate work
- Modularity
  - ✓Objects can be responsible for their own transactionality irrespective of client

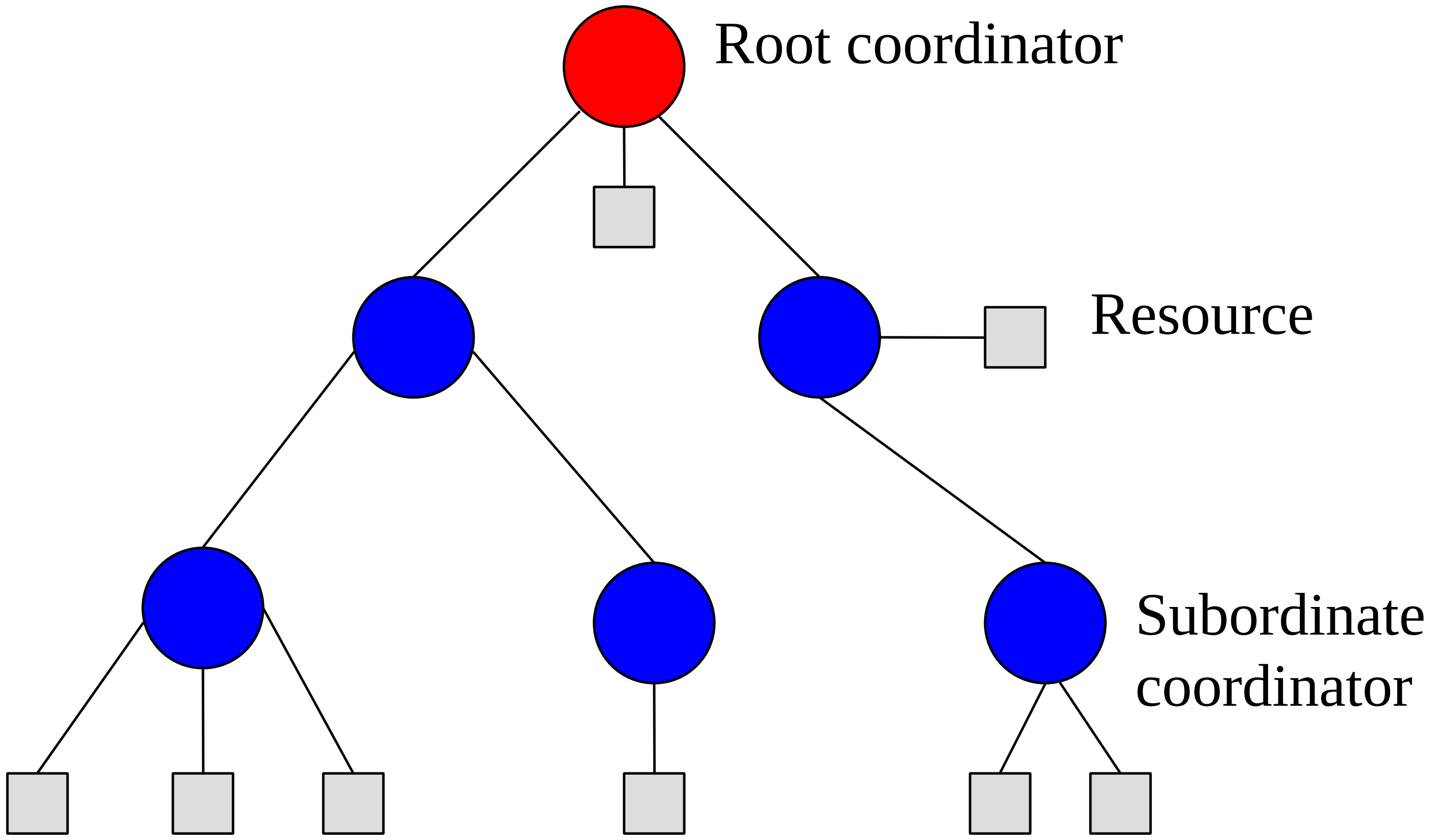
Transaction propagation

- Explicit propagation
  - ✓Context passed as parameter
  - ✓Object implementation responsible for using it when required
- Implicit propagation
  - ✓Transaction context is implicitly passed from client to object
  - ✓All operations are assumed transactional

Interposition

- Allows a subordinate coordinator to be created
- Interposed coordinator registers with transaction originator
  - ✓Form tree with parent coordinator
  - ✓Application resources register locally
- JBossTS supports interposition for implicit and explicit propagation

Interposition



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

WS-AT/WS-BA

- Specifications released by Arjuna, BEA, IBM, IONA and Microsoft
- Separate coordination from transactions
- Define two transaction models
  - ✓ AtomicTransaction
    - Closely coupled, interoperability
  - ✓ Business Activities
    - Compensation based, for long duration activities

AtomicTransaction

- Assume ACID transactions
  - ✓ High degree of trust
  - ✓ Isolation for duration of transaction
  - ✓ Backward compensation techniques
  - ✓ Does not allow heuristic outcomes
- Integration with existing transaction systems
  - ✓ Important to leverage investments
- Interoperability between transaction systems
  - ✓ Something of a holy grail to date

Business Activities

- Workflow-like coordination and management
- Business activity can be partitioned into tasks
  - ✓ Parent and child relationships
    - Select subset of children to complete
    - Parent can deal with child failures without compromising forward progress
- Tasks can dynamically exist a business activity
- Tasks can indicate outcome earlier than termination
  - ✓ Up-calls rather than just down-calls

JBoss 3/WebLogic/JBoss 4*

Application server versus transaction capabilities	Standards compliant	Industry proven	2PC	Failure recovery	Flexible deployment	Distributed transactions	Mgmt tools	Interop	Flexible partici p-ants	Web Services transactions	WS-tx to J2EE tx bridge
JBoss 3	√ (JTA)	X	√	X	X (tied to application server)	X	X	X	X (XA specific)	X	X
WebLogic	√ (JTA)	√	√	√	√ (can run out of application server)	√	√	X	X (XA specific)	X	X
JBoss 4*	√ (JTA and JTS)	√	√	√	√ (can run out of application server)	√	√	√ (via JTS)	√ (not just XA partici pa-nts)	√ (via Arjuna, IBM, MSFT, Oracle specs.)	√

Summary

- Product features
  - ✓ High performance and reliability
  - ✓ Manageability and configurability
  - ✓ Standards compliance
  - ✓ Modular architecture to optimise footprint
  - ✓ Pure Java implementation
- Deployment options
  - ✓ Application server agnostic
  - ✓ Deployable in or outside a J2EE application server

Any questions?