The evolution of a transaction processing system

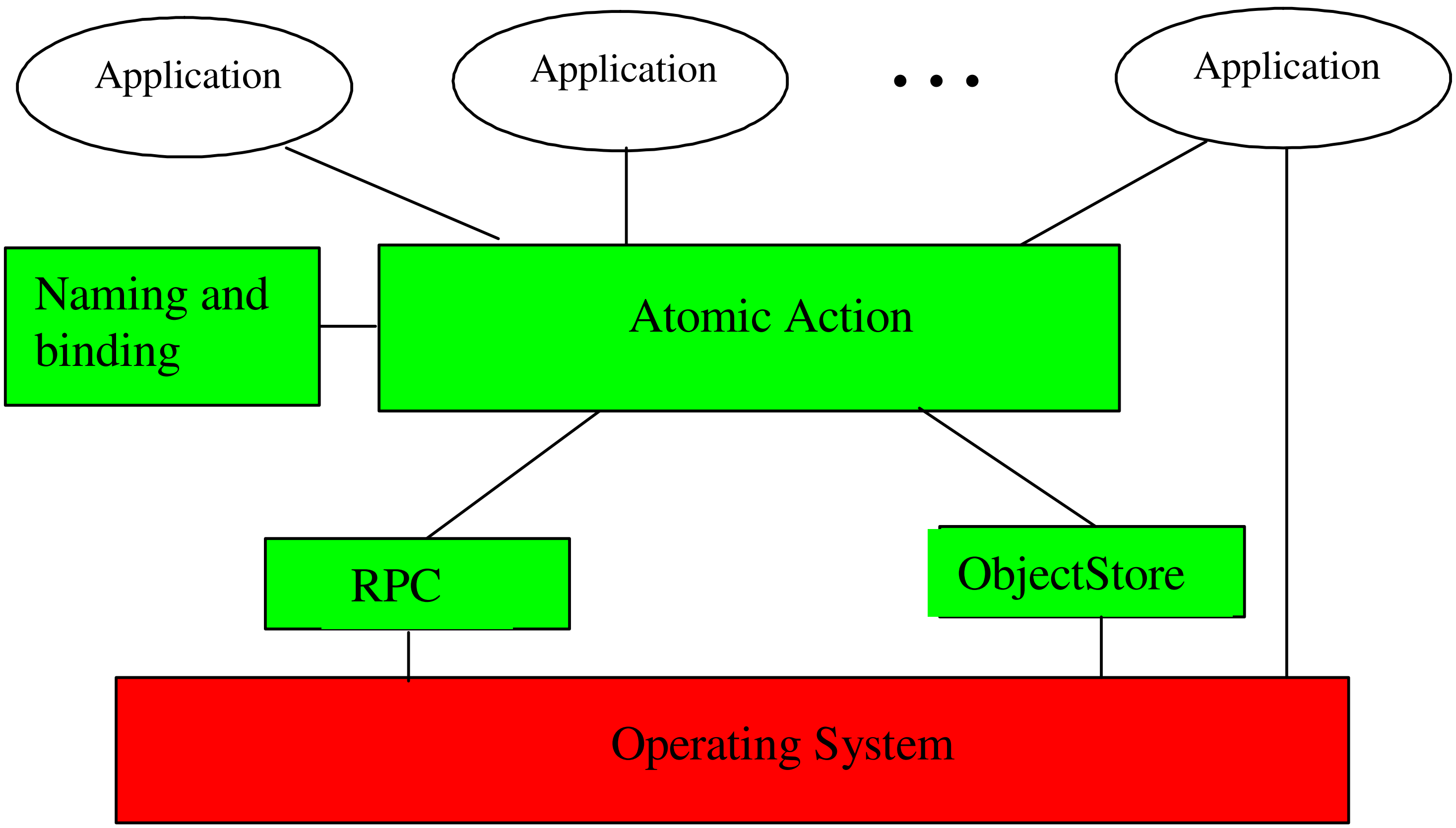
Mark Little
Chief Architect, Arjuna Technologies Ltd
mark.little@arjuna.com

arjuna
middleware for reliability

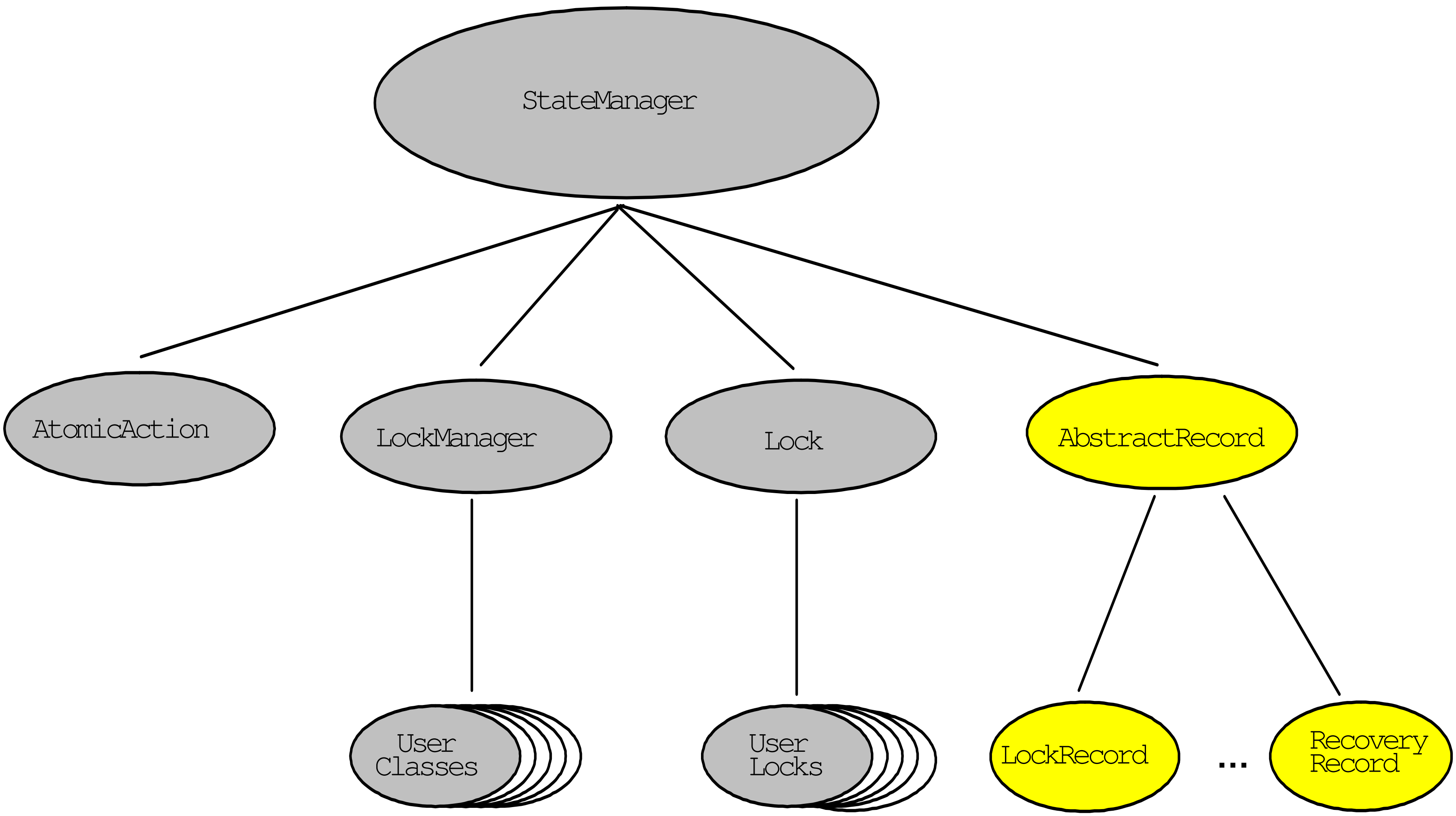
ATS background

- Distributed transaction processing system
  – Began life in C++ back in 1986 at the University of Newcastle upon Tyne, England
    - Exploit object-oriented techniques
  – Pre-CORBA, DCE, COM, …
    - Own RPC and stub-generation mechanisms
- Complete toolkit for development of fault-tolerant applications
  – Persistence, concurrency control,…

The architecture



Class hierarchy



AbstractRecords

- Forms the basic interface for all transaction participants
  - (nested) two-phase commit aware
    - Does not imply a specific implementation.
- Key to the longevity of Arjuna.
  - Many transaction systems then and today tie transaction participants to X/Open XA compliant resources (e.g., databases).

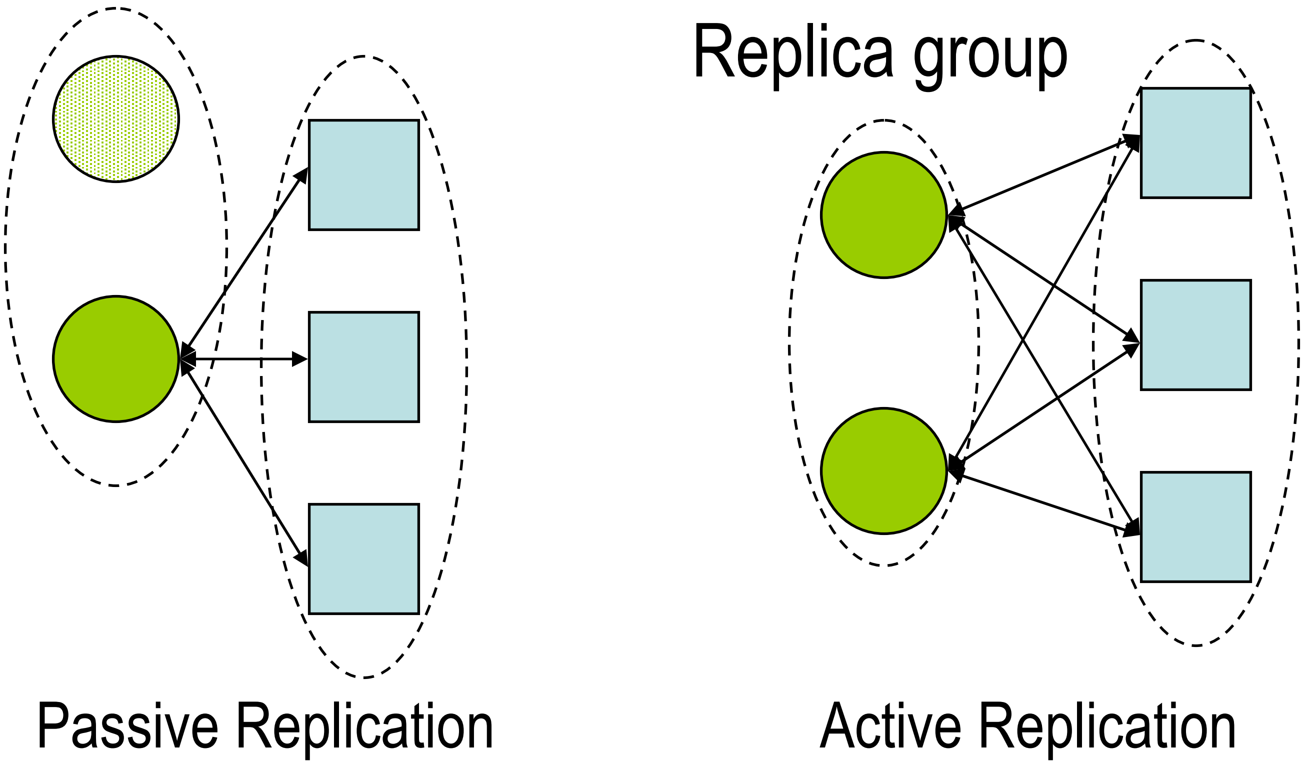
Basics covered

- High availability
- Standards evolution
- Performance, performance, performance!
- Support for multiple models and protocols

High availability

- Active replication
  - Assumes determinism
  - K+1 replicas to tolerate K failures
    - 2K+1 if network can partition
  - Group communication
    - Typically ordered delivery
- Passive replication
  - Does not require determinism
  - K+1 replicas to tolerate K failures
  - Slower fail-over time

Replication protocols



Student registration

- No money to buy
  - Nothing available to by at that time
- Must work on PC, Mac and various Unix workstations
  - 20000 students over 5 days
    - Cannot tolerate failure as student gets no money
- Campus wide
  - 10 servers, with 150 front-ends
- Network can partition

Standards evolution

- 1995 saw release of initial OTS specification from OMG
  - Shares many similarities with Arjuna
    - Generic two-phase participants
    - Optional support for nested transactions
  - Only a two-phase commit protocol engine
    - Persistence and concurrency control elsewhere
- Overlap in several areas
  - Naming and binding
  - RPC

Modifications

- Replace RPC and Naming/Binding modules
  - Slight modifications due to different distribution model
    - E.g., Arjuna had support for passing pointers and associated memory, CORBA IDL did not
- Transaction engine remained *unchanged*
  - Wrap OTS participants in AbstractRecords
    - Benefits from previous 10 years of testing and use

Coordinator performance

- Supported typical optimizations
  - Presumed abort, one-phase commit, read-only participants
- Also supports embedding of coordinator
  - Small footprint
    - Can run in less than 16 Meg
  - Durability and recovery are loaded on demand
  - Log structure is created on demand
    - Implementation is flexible too (no requirement for db, for example)

Coordinator performance

- Different types of participant
  - Recoverable
    - Two phase (and nesting) aware
    - Do not have any persistent state representation
    - Do not require recovery
    - Do not require (transaction) log
  - Durable recoverable
    - Have persistent state representation
    - Require recovery
    - Require (transaction) log

Multiple models and protocols

- Factor out core transaction engine
  - Essentially the same engine that began life in 1986
    - No dependency on any distribution infrastructure
    - Purely local transactions and recovery
  - Hooks for distribution are essentially the interfaces to the RPC, Naming/Binding and Crash Recovery modules
    - Participant implementations are opaque to the transaction engine
    - Context information via XML+SOAP, IIOP, …
- Embedded within
  - HP products (HP-TS, HP-MS, HP-WST)
    - HP proof of concepts technologies (mobile devices)
  - Arjuna products (A-TS, A-MS, A-XTS)

Conclusions and lessons learnt

arjuna
middleware for reliability

- Modularity helped us a lot
- AbstractRecord made it easier to customise
- Customer feedback has been extremely useful
  – "Transaction semantics are great, but relax the properties."
- Standards are good
  – But their lifetimes and impact are sometimes over hyped
- Transactions everywhere is a good idea
  – Just make them cheap to use!