An Examination of the Transition of the Arjuna Distributed Transaction Processing Software from Research to Products

Mark Little

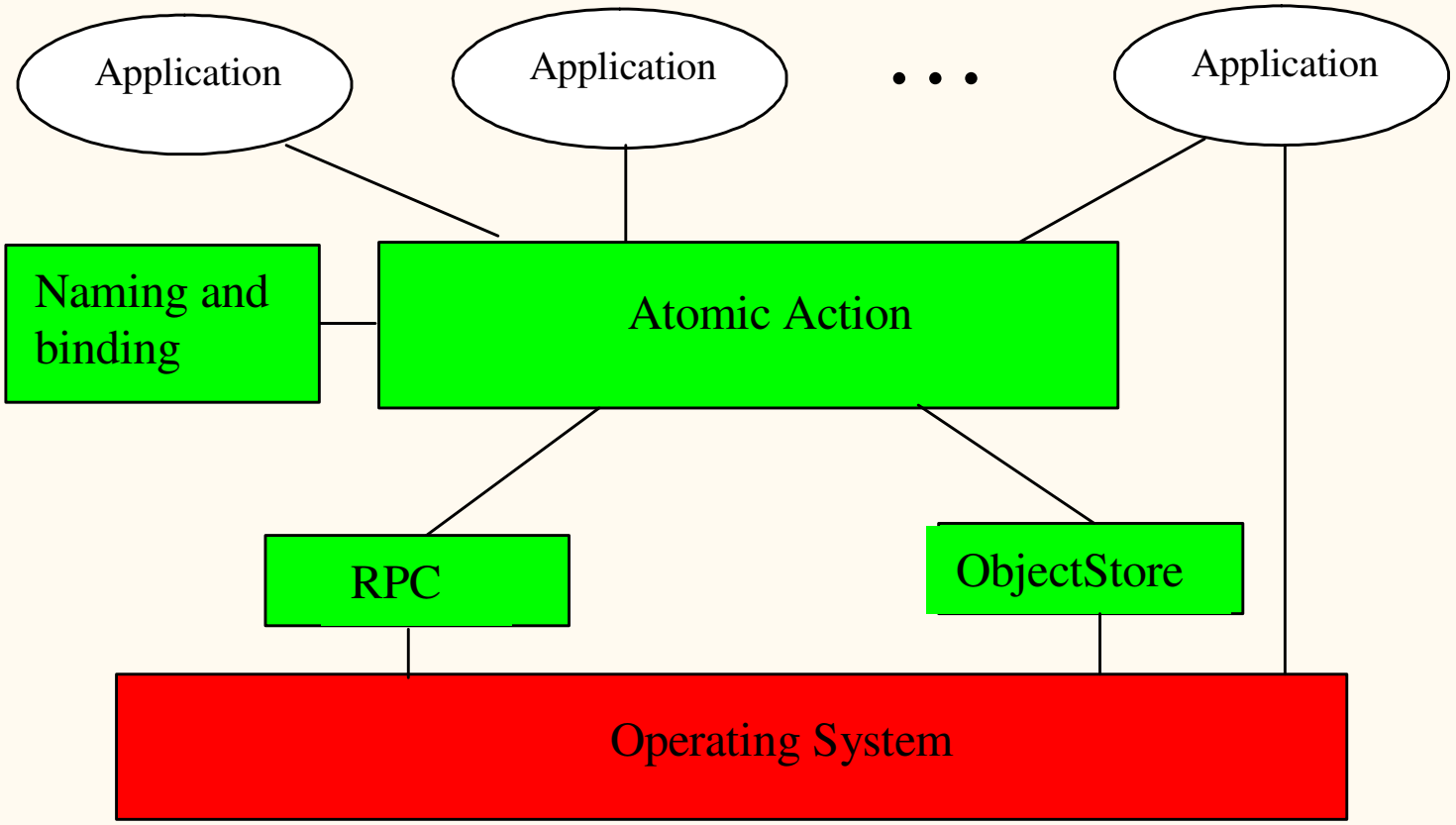
Head of Transactions Technology, Arjuna Technologies Ltd

mark.little@arjuna.com

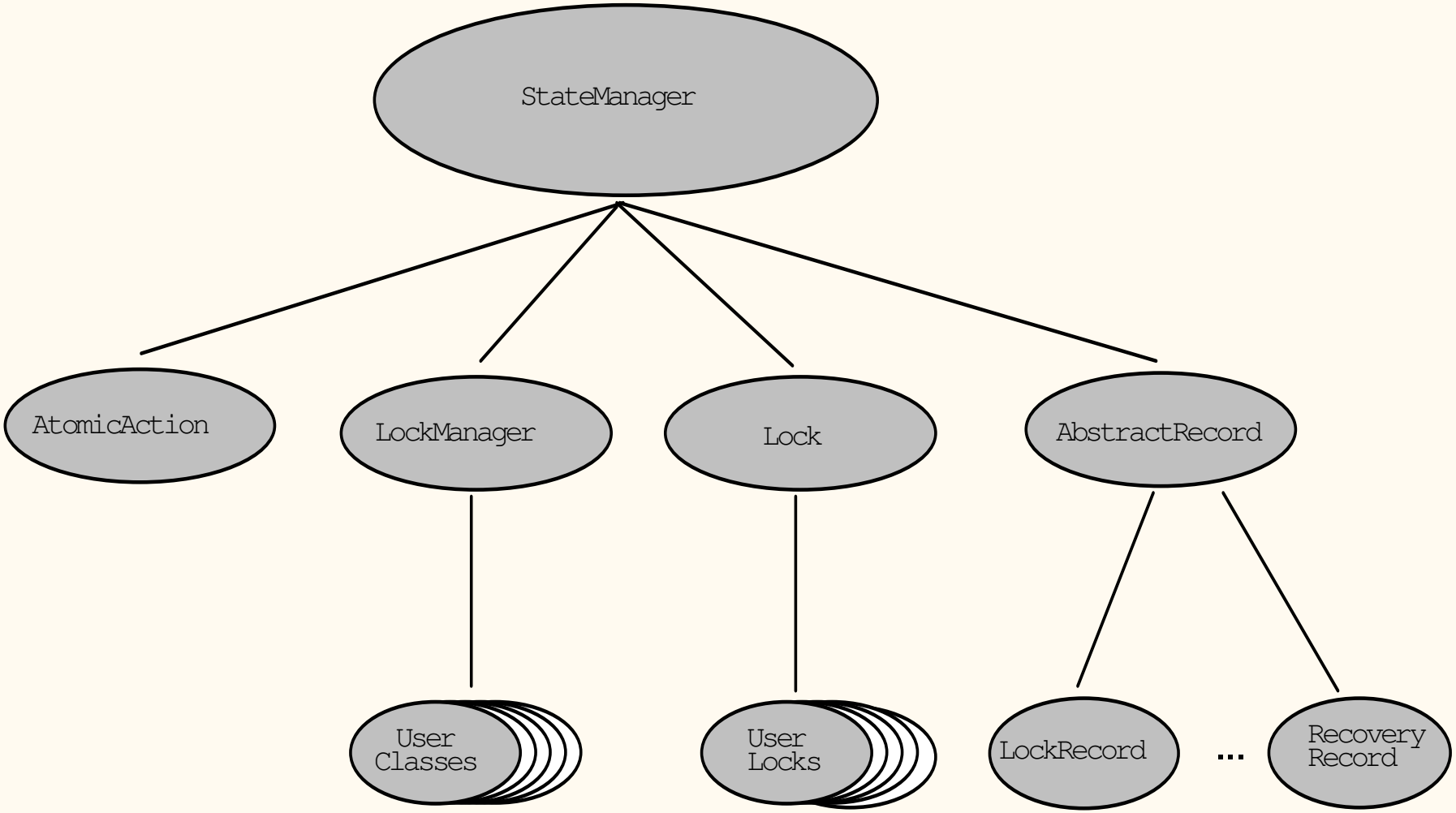
What is Arjuna and where did it all start?

- 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
  - Persistent, concurrency control, replication, …

The original architecture



Class hierarchy



What does this provide?

- Easy to use API for constructing transactional applications.
- AbstractRecord forms the basic interface for all transaction participants
  - (nested) two-phase commit aware but 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).

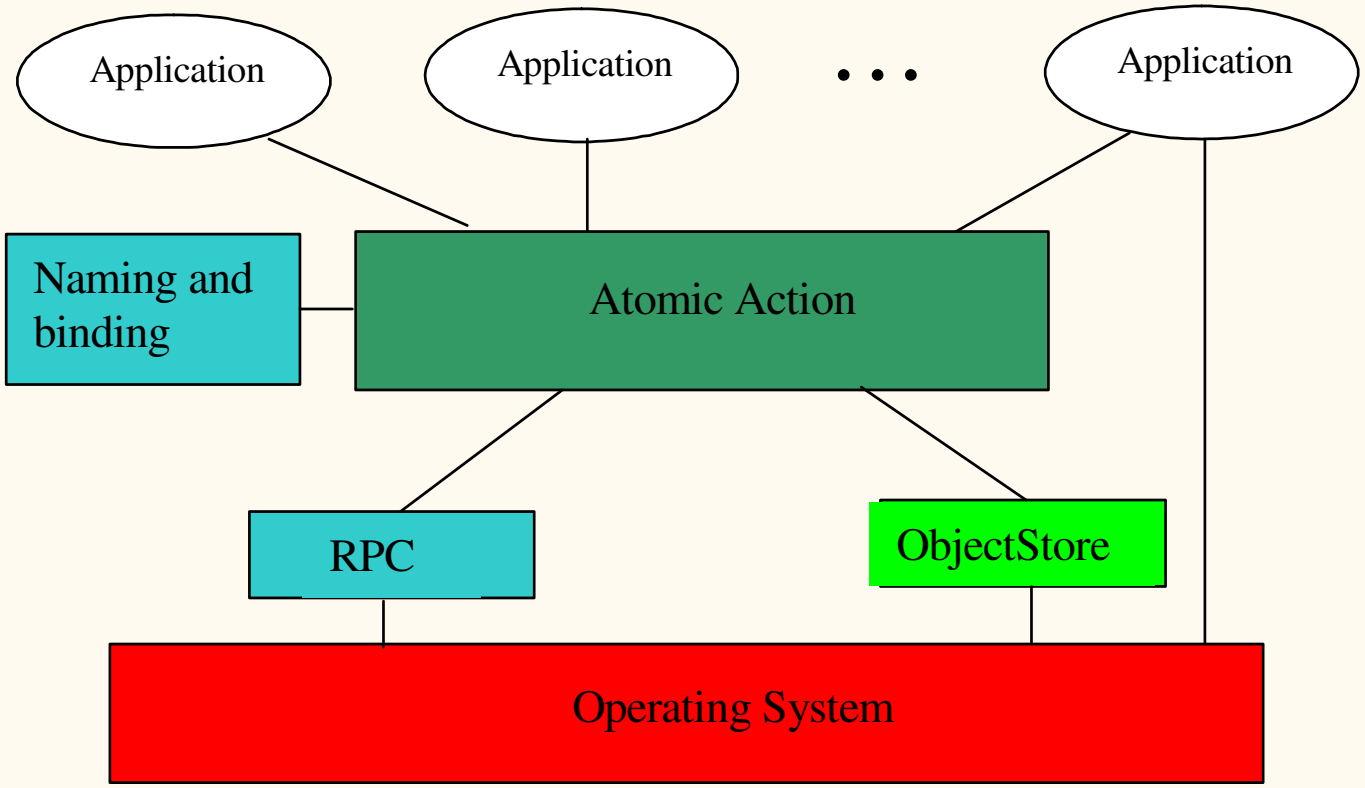
1994 student registration system

- No money to buy in (academic researchers are cheap!)
  - Must work on PCs, MACs, and various Unix workstations
    - 15000 students over 5 days
      - Cannot tolerate failure as student gets no money
  - Campus wide
    - 10 servers, with 120 front-ends
  - Network partition and recovery

When Arjuna met CORBA

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

OTSArjuna



Required modifications

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

Crash recovery

- Crash recovery needed for ACID properties
  - May need to recreate distributed transaction tree (coordinators and participants)
    - Requires knowledge about participants, e.g., where do they reside?
- Original implementation was closely tied to Arjuna's RPC and stub-generator
  - Re-implementation tied to OTS
    - Pragmatic choice based on time constraints and view of future

Productising Arjuna

- JTSArjuna first Java transaction service
  - Marketed by Arjuna Solutions Limited
    - Acquired by Bluestone in 2000, later acquired by Hewlett-Packard in 2001
- Major investment in staff
  - QA (8 from 1 dedicated staff managing > 4000 individual tests)
    - Fewer bugs than might have been expected
  - Development (10 from 2)
  - Training, manuals and whitepapers

Was it worth it?

- What did we gain?
  - A wider audience for our product and ideas ☺
    - More influence in the standards
    - More customer feedback
      - $10 million
    - Interface changes are a no-no!
  - Stress!
    - Moving away from R&D ☹
      - Shorter deadlines mean more focus
- Was it worth it?
  - Yes (I think!)

Where to now?

- Transactions show up in:
  - Web Services
  - Mobile/embedded devices
  - J2EE/CORBA
    - JTS
    - JMS
- Is there some commonality?
  - Two-phase commit

Continued application

- Factor out core transaction engine
  - ArjunaCore
    - Essentially the same engine that began life in 1986
      - Includes toolkit and ObjectStore module
    - Hooks for distribution are essentially the interfaces to the RPC, Naming/Binding and Crash Recovery modules
- Embedded within:
  - HP-TS
  - HP-MS
  - HP-WST

Conclusions and lessons learnt

- Modularity helped us a lot!
- Object-orientation (and specifically AbstractRecord) made it easier to customise
- QA in industry is more heavily emphasised than in academia
  - Important to convincing people to use and invest
- Make any configuration choices easier for non-experts to use
  - E.g., transaction log location or size
- Being ahead of the curve may require staying-power