### Orthogonal Persistence Hypothesis

- If applications developers are provided with a well-implemented and well-supported orthogonally persistent programming platform
- Then a significant increase in developer productivity will ensue
- And operational performance will be satisfactory
- Orthogonality, Completeness, Persistence Independence

Is Java the right language for the OPH?

### Is Java the right language for the OPH?

- **Hopefuls**
  - Resources: SunLabs backing ($)
  - Type Safety
  - Popularity
  - JVM
- **Skeptics**
  - Rapid JDK changes
  - Prototype was complex and unreliable
  - Necessary to capture state (may be complicated in VM) at a checkpoint and then reconstruct upon restart
  - Use as Glueware

### Existing Persistence Options for Java

- Java Object Serialization
- Links to Relational DBs (JDBC)
- Object-Relational Mapping
- Object Database Mapping
- Java Data Objects (JDO)
- Enterprise Java Beans (EJB)

Why are these approaches not good enough?

### Why Not Enough? (Skeptics)

- Java Object Serialization
  - Not Orthogonal (must be serializable)
  - Not Complete (class info not preserved in object state)
  - Fails persistence independence (copy = obj ID text)
  - Standard and customizable – at small scale
- Links to Relational DBs (JDBC)
  - Impedence Mismatch Java – relational
- Automated Object-Relational Mapping
  - Complex and difficult to automate object to relational
- Object Database Mapping
  - Java operations defeat persistence independences
- Java Data Objects (JDO)
  - No persistence independence
- Enterprise Java Beans (EJB)
  - Strict rules for developers = no persistence independence

### Past Failures

- “Host” of previously implemented orthogonally persistent languages lacked conclusive test of OPH
- Insufficient Resources
- Language not popular or type safe

How did PJama get the resources?
Achievements (Hopefuls) and ShortComings (Skeptics)

• Orthogonality
  – "good enough" for many applications
  – Thread
• Persistence Independence
  – "completely achieved" all code runs unchanged
• Durability
  – ARIES recovery works well
  – Other methods conflict with endurance
• Scalability
  – Up to 10GB (no problems anticipated)
• Schema Evolution
  – Permits any change
  – Must stop application to perform change

PJama Failure Tradeoffs

• Specific subset of Java
  – More convincing and deliverable
  – Sun may not see cost-benefit for other subsets
• Focus on a particular application
  – Works well, can deliver as needed
  – Devalue experiment
• Prioritize Requirements
  – Achieve more reliability/functionality
  – May not omit some requirements and still have a sufficient foundation for testing OPH and maintaining support
• Technical Decisions
  – Hindsight required
  – May result in other challenges

Is the complex approach the right approach?
Does this provide much benefit compared to high level statements that can do this?

Achievements (Hopefuls) and ShortComings (Skeptics)

• Platform Migration
  – Possible
  – Stop application and must fit data in memory
• Endurance
  – Stop: above reasons and for garbage collection
  – 6 days ➔ few minutes (threads)
• Openness
  – Demonstrated with some classes (sockets)
  – Left out many core classes
• Transactional
  – Simple transaction facility provided
  – Threads must reach a consistent state before a VM checkpoint
• Performance
  – Relative to some persistent applications, up to 10X faster (no details provided)
  – 15-20% slower than normal execution (what about scalability impact?)

What are they actually gaining?

Industry Obstacles

• Commitment to Existing Practices
• Displaced Problems
• Alternative Solutions
• Dominance of Glueware
• Distribution Drives Application Structure
• Lack of Credibility
• Language Trap

What has changed that makes this easier/harder?

VM Snapshot?

• Hopefuls
  – Migrate to any system
  – Save state
  – Cheap
• Skeptics
  – What about external resources (network, etc)?
  – Persistent bugs as well? Can a DB fix this?

Can we use this idea to make it language independent?
Is OPH still a viable research area?  
still a practical, attainable benefit for developers?  
What has changed that makes this easier/harder?

- Orthogonality
- Completeness providing coherence and comprehensibility to enterprise systems
- Mobility and ease of use/construction

Funding Question

- Is $10M adequate?
- Is there a simpler/cheaper way to show the benefits of OPH besides a multi-million experiment?
- If so (we all mostly thought so), then how can OPH be demonstrably useful?
  - Orthogonality
  - Completeness providing coherence and comprehensibility to enterprise systems
  - Mobility and ease of use/construction