



Show-stoppers for Transactional Memory

Dave Dice – blogs.sun.com/dave

J2SE Core Engineering

SunLabs Scalable Synchronization Group

PPoPP Panel 2007-3-15

Concurrency

- Here today
- Explicit thread-level parallelism
 - **not** a future
 - a remedy with side-effects
 - brings hope of performance
 - and promise of complexity
 - end of the lay-z-boy programming era (David Patterson)

Human scalability

- Today:
 - lots of available cores
 - small concurrency priesthood
- Programs - programmers
- Reduce **complexity**
 - Eliminate common sources of errors
 - Think sequentially, execute concurrently
 - At least raise the abstraction level above locks

TM Critique

- Restrictions (as of today)
 - large/long transactions
 - IO and irrevocable state
- Single-threaded latency ?
 - yes, it's important
- Missing infrastructure:
 - debugging, performance profiling
- Open issues:
 - atomicity, nesting, exceptions

Better than locks ?

- Wish: synchronized ~~(Lock)~~ {...}
- **Not** a drop-in-replacement
- decreased complexity; added constraints
- Better but not good enough
- Transactions won't displace locks
 - incremental adoption
- We'll end up with both
 - lock-aware transactions?

A useful addition?



Shared Mutable State

- Minimize shared mutable state
- Locks and transactions : immutable view
- Eliminate shared data
- **Message passing:** MPI, Erlang, etc
- 1 thread per address space
- Same programming model inter- & intra-node
- Can't express common concurrency bugs
- Can you express large systems?
 - old-school distributed programming

Where does this take us?

- Locks + transactions + message passing
- Keep the lock abstraction
 - Transparently **Commute** to transactions
 - Revert to actual locks only as needed
 - Complexity of coarse-grained locking
 - Possibly better performance