Clojure's Concurrency Types
Refs, Atoms, Agents and Vars

Alex Redington
lovemachine@thinkrelevance.com
@holy_chao

Programming with Values

- Functions of immutable data, returning immutable data
- Values never change (e.g. java Strings)
- Made performant by Persistent Collections (e.g. lists, vectors, maps, sets)
- Allows for change without destructive modification

State vs Identity

- State: A immutable value representing a snapshot of some point in time. May be a compound of (many) other immutable values. E.G.: Lists, Vectors, Maps, Sets, Numbers, Strings, Keywords, Symbols, etc. CANNOT be changed.

- Identity: A series of states sequentially ordered in time. Can be changed with rigorously defined semantics. These are Refs, Atoms, Agents and Vars.

Syncronous?

- A synchronous identity can be updated from one state to another state synchronously. The update will block until the identity has successfully transitioned to the new state. The update will return the new state at the end of the update.
- The thread issuing the update stops until the update has completed successfully.
- Asynchronous identities can be updated, but do not block the updating thread's execution. Don't return the post-update value.

Coordinated?

- Coordinated identities allow for “all or nothing” semantics. Updates occur within transactions.
- If any of the identities taking part in a transaction has its value altered during the course of the transaction, the transaction fails and is retried.
- Independent identities do not coordinate with other identities, and therefore do not need to have their updates placed in transactions.

Unified update model

- All identities start with an initial value, e.g.: (def zot (atom []))
- All shared identities can be updated with a specific update function (swap! zot conj :a)
- All shared identities can be read with deref, or the @ reader macro (println @zot) ;; ”[:a]”

Compare with locking

- No locking orders
- No explicit acquisition and subsequent release of locks
- No potential for deadlocks
- Simpler (Arf = WoofBark^2)
- Still not easy

Vars: Isolated identities

- Vars are boring. Clojure code use them all the time (created with def and with-local-vars)
- Can have a root binding (not required!)
- Can have a per-thread binding (also not required!)
- When derefed, per-thread binding wins if present, followed by the root binding if there is no thread binding.
- Read with deref/var-get/@, altered with var-set, or alter-var-root

Atoms: Synchronous, Independent

- Read with deref/@
- Updated with swap! and reset!
- No way to update multiple atoms with all-or-nothing semantics

Refs: Synchronous, Coordinated

- Read with deref/@
- Updated with alter, commute, and ref-set
- Must be updated within dosync. If multiple refs are read and set within one dosync txn, Clojure's STM will sync the reads and updates to those refs.
- Ideal when multiple identities need to interact transactionally.

Agents: Asynchronous, Independent

- Read with deref/@
- Updated with send, and send-off
- Send will execute the update function in a thread pool, use if the update will not block.
- Send-off will execute the update function in a new thread, use if the update may block.