

refs - synchronous & coordinated

atoms - synchronous & uncoordinated

agents - asynchronous (actors)

vars - thread-local state only

It's a mutable reference to an immutable object. They are for coordinating synchronous changes to shared state using software transactional memory.

They are actors, used to send asynchronous changes to shared state?

Create reference:

```
(ref initial-state)
```

Two ways to dereference:

```
(deref reference)
```

```
@reference
```

Point the reference somewhere else:

```
(ref-set reference new-value)
```

(dosync & exprs)

- Atomicity. They are "all or nothing" and cannot be executed only partially.
- Consistency. Validation conditions can be specified causing the entire transaction to fail should any one of them not hold.
- Isolation. Transactions are never visible in a "partially complete" state.

The final part of ACID, durability, is not provided, since software transactions are in-memory only.

Transactions are coordinated if they happen simultaneously from others' perspective.

Coordination is achieved by putting multiple transactions in the same `dosync`.

```
(alter ref update-fn & args)
```

This is preferable to using a `ref-set` with derefing.

It still needs to be wrapped in a `dosync` as normal.

The argument order might work better. `cons` takes the item as the first argument, `conj` takes the item as the second argument.

```
(func-name obj-to-update & args)
```

When the transaction is running it uses its own private copy of any references needed. If any of these references are altered by other transactions while the first one is running, it will have to retry the whole thing from scratch until it succeeds.

This is potentially costly because the altering that other transactions do while the first transaction is running may not actually interfere with the first transaction's business.

By using `commute`, which has the same signature.
However, this is only acceptable if the transactions are commutative.

```
(ref initial-state & validator-fns)
```

```
(atom initial-state & validator-fns)
```