What are Clojure's four tools for concurrency?

refs - synchronous & coordinated
atoms - synchronous & uncoordinated
agents - asynchronous (actors)
vars - thread-local state only

What is a ref?

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

What are agents?

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

Reference or dereference objects.

"Re-assign" the reference.

Create reference:

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

Two ways to dereference:

```
(deref reference)
@reference
```

Point the reference somewhere else:

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

How do you execute a transaction?

```
(dosync & exprs)
```

What properties to transactions guarantee?

What property is not guaranteed?

- 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.

What does it mean for two transactions to be coordinated?

How can this be achieved?

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

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

What's the best way to update a ref in a single step?

```
(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.

Why might one use `conj` instead of `cons` when working with a list?

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

What is the signature of a suitable update function for `alter`?

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

What is the fundamental approach of STM's Multiversion Concurrency Control approach?

Why does it get expensive?

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

How can one improve upon the performance of `alter`?

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

Add a validation function to a ref to maintain transaction consistency.

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

Create an atom.

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