U051SS2EU : that has nothing to do with it
U051SS2EU : def creates a new var in your namespace
U051SS2EU : oh - I see, you create the gensym def once
U051SS2EU : that guarantees that pdiff-once is a race condition if it's used in two threads though
U0BKWMG5B : What's the purpose of the macro?
U051SS2EU : that's another good question
U2TCUSM2R : The purpose of the macro is to automate three calls in one: creating the agent, calling pdiff, and returning the contents of the agent
U051SS2EU : if two threads use that macro, the second one will replace the data used by the first
U051SS2EU : it's extremely unsafe
U0BKWMG5B : Okay, but why not:```
(defn pdiff-once [poly order]
  (let [tape (agent (i/int-map))]
    (pdiff poly tape order)
    (await tape)
    @tape))
```

U0CGFT70T : why this: ```router=&gt; (if-let [{params :params} {}] (str "params:" params ".") "NOT-defined") "params:."```

U0BKWMG5B : <@U0CGFT70T> because {} is not falsey.
U2TCUSM2R : weavejester: let me try that. I'm not sure it'll work, but I'm relatively new to agents.
U0BKWMG5B : `(if-let [params (:params {})] ...)`
U0CGFT70T : <@U0BKWMG5B> : thanks :slightly_smiling_face:
U0BKWMG5B : With if-let, the expression on the right needs to be `nil` or `false` to fail
U0CGFT70T : trying to hit else if params not defined
U0CGFT70T : <@U0BKWMG5B> ok...
U0BKWMG5B : <@U2TCUSM2R> You could also use a promise.
U0BKWMG5B : It depends what `pdiff` looks like, but a promise is more usual.
U051SS2EU : I assumed an agent was being used because there were multiple alterations
U051SS2EU : but doing that in a new thread makes using an agent problematic...
U2TCUSM2R : It works
U0BKWMG5B : The name `pdiff-once` also suggests a memoize, but it depends what you're trying to do.
U2TCUSM2R : But `await` is not working
U0BKWMG5B : Could you give us an idea of what `pdiff` is doing, <@U2TCUSM2R> ?
U2TCUSM2R : `send` calls inside pdiff have not finished by the time `pdiff-once` returns
U2TCUSM2R : it's just associng values with `send`
U051SS2EU : if await returns, that means those calls weren't even made before await was
U0BKWMG5B : `pdiff` is parallel diff I assume.
U051SS2EU : you need some other way to know pdiff is done
U2TCUSM2R : This is exactly what I said previously...
U051SS2EU : you said "await isn't working" which I might have misinterpreted
U2TCUSM2R : I said, "I think `await` isn't doing anything since it can't know whether `pdiff` has finished"
U0BKWMG5B : Would it be possible to post the definition of `pdiff` as well?
U2TCUSM2R : It's not going to make a difference
U051SS2EU : it could, if there was a way to ensure you don't return from pdiff until it makes all it's send calls for example
U0BKWMG5B : I'm not clear on what `pdiff` is doing, or why you're using an agent. I assume it's doing something in parallel.
U5YHX0TQV : <@U2TCUSM2R> do you get your functionality working without all the parallel stuff? Just plain idiomatic single-threaded clojure
U2PGHFU5U : Does anyone know how to obtain the `Set-Cookie` header from `http-kit`, when you pass the `Cookie` header in the call as well? E.g.,
```
(defn visit-url [{:keys [cookies url] :as context}]
  (let [result-chan (chan)
        check-result (fn [{:keys [status] :as response}]
                       ;; TODO: get new cookies here....:/ not visible in response
                       (log/error "RESPONSE" response) ; => no `Set-Cookie`

```
                (go (&gt;! result-chan (= status 200))))]
  (http/get url
            {:headers          {"Accept" "text/html"
                                "Cookie" cookies} ; cookies is string of earlier obtained cookies
             :follow-redirects false}
             check-result)
  result-chan))
```

U0K1UT6PQ : question: how to find a dependency a leiningen plugin pulls in at runtime? I've run `lein deps :tree` on the most obvious dependencies, and I don't see where a particular version of ring is coming from. I've sprinkled `:exclusions` all round. Yet I find the dependency in `target/stale/leiningen.core.classpath.extract-native-dependencies` ...
U0K1UT6PQ : (context: trying to upgrade gorilla repl to 1.9, but it barfs on an old version or ring)
U0K1UT6PQ : happens when I do `lein gorilla :port 9000`, yet neither gorilla-repl nor lein-gorilla seem to need it
U06F82LES : have you tried deleting `target`?
U0K1UT6PQ : I've deleted all the content yes
U06F82LES : `lein deps :tree` should work, unless gorilla-repl does something funky
U06F82LES : you can issue a global exclusion for ring
U06F82LES : <https://github.com/technomancy/leiningen/blob/master/sample.project.clj#L86>
U0K1UT6PQ : hm. the problem being that lein-gorilla does need ring, just not that old one ...