U051SS2EU : <@U3JURM9B6> I'd think accidentally calling an accessor and constructor would be a lot less likely than accidentally calling assoc on the wrong object
U3JURM9B6 : <@U051SS2EU> : I can make the constructor private, then have a function (which calls the constructor) do checks beforehand
U04VDQDDY : <@U3JURM9B6> For accessing object properties (as opposed to array elements), consider `goog.object/get` instead of `aget`.
U050ECB92 : <@U3JURM9B6> creating custom types that have their own field accessors completely negates the value of generic data access from the clojure std library
U050ECB92 : your code will essentially be a non-reusable DSL
U050ECB92 : I firmly believe you are going down the wrong path, and that the advice you're getting about specific implementation is misguided
U050ECB92 : you should test against invalid / incorrect data (even from your own code). Have you considered writing generative tests for your datastructures? (either from test.check or from spec)
U050ECB92 : test against bad data while allowing bad data to exist
U050ECB92 : Lots of things that are common in other languages (e.g. privileged data) are the exact wrong approach in Clojure.
U050ECB92 : when I say 'generic data access' here is an example of what I mean:
U050ECB92 : ```(get-in order [:line-items 2 :product :picture :url])
```
If you make a custom type, you'll need to do:

U050ECB92 : ```(-&gt; order
  (get-in [:line-items 2 :product])
  (.-picture)
  (get :url))
```

U050ECB92 : You will infect the codebase with specificity
U050ECB92 : and negate a whole lot of the benefits of "the clojure way"
U1WMPA45U : is it possible to tell if you're already in a `clojure.core.async/go` block inside a `defmacro`?
U11BV7MTK : you could call `<!`
U11BV7MTK : the code of that is literally throw an error
U11BV7MTK : the go macro totally rewrites everything in it. so if that throws an error then you are not in a go block. otherwise have it take from a channel that has a single value ready to give
U11BV7MTK : super hacky
U1WMPA45U : our devs went a bit bonkers with nested `go`s
U0NCTKEV8 : the go macro actually macro expands everything before it does its thing, and the way it macro expands has some differences from the way the compiler does it