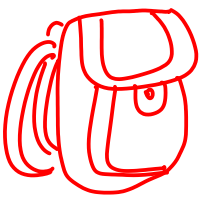
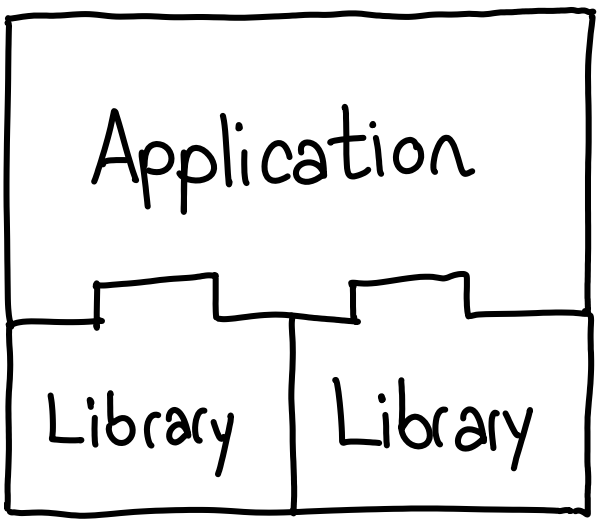
Implementing Backpack in GHC

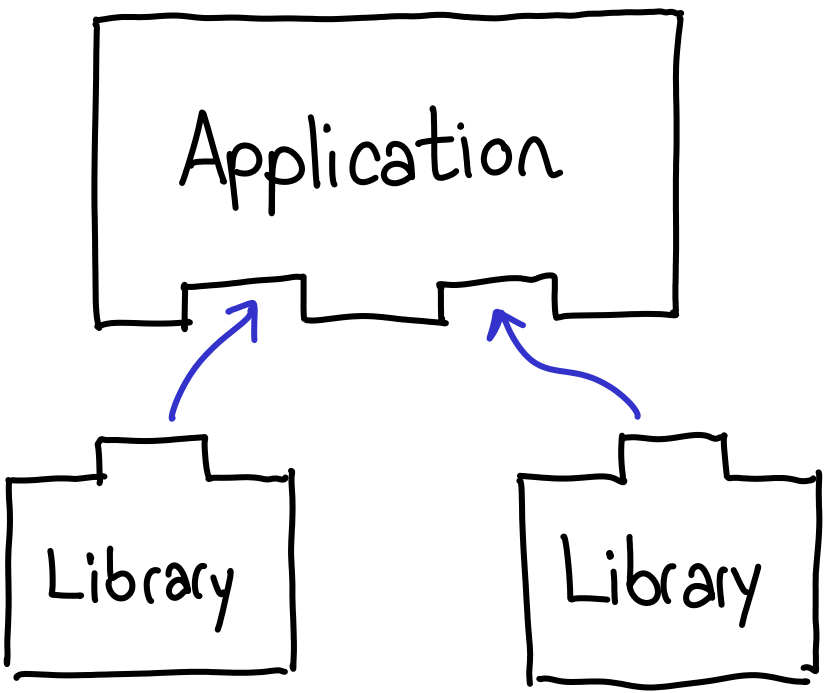


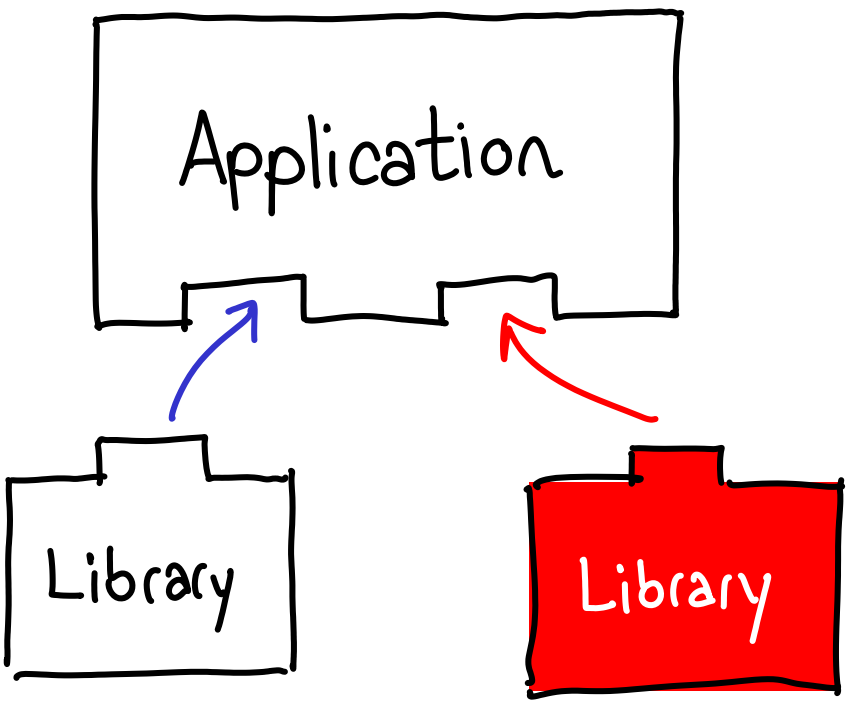
Edward Z. Yang <ezyang@cs.stanford.edu>

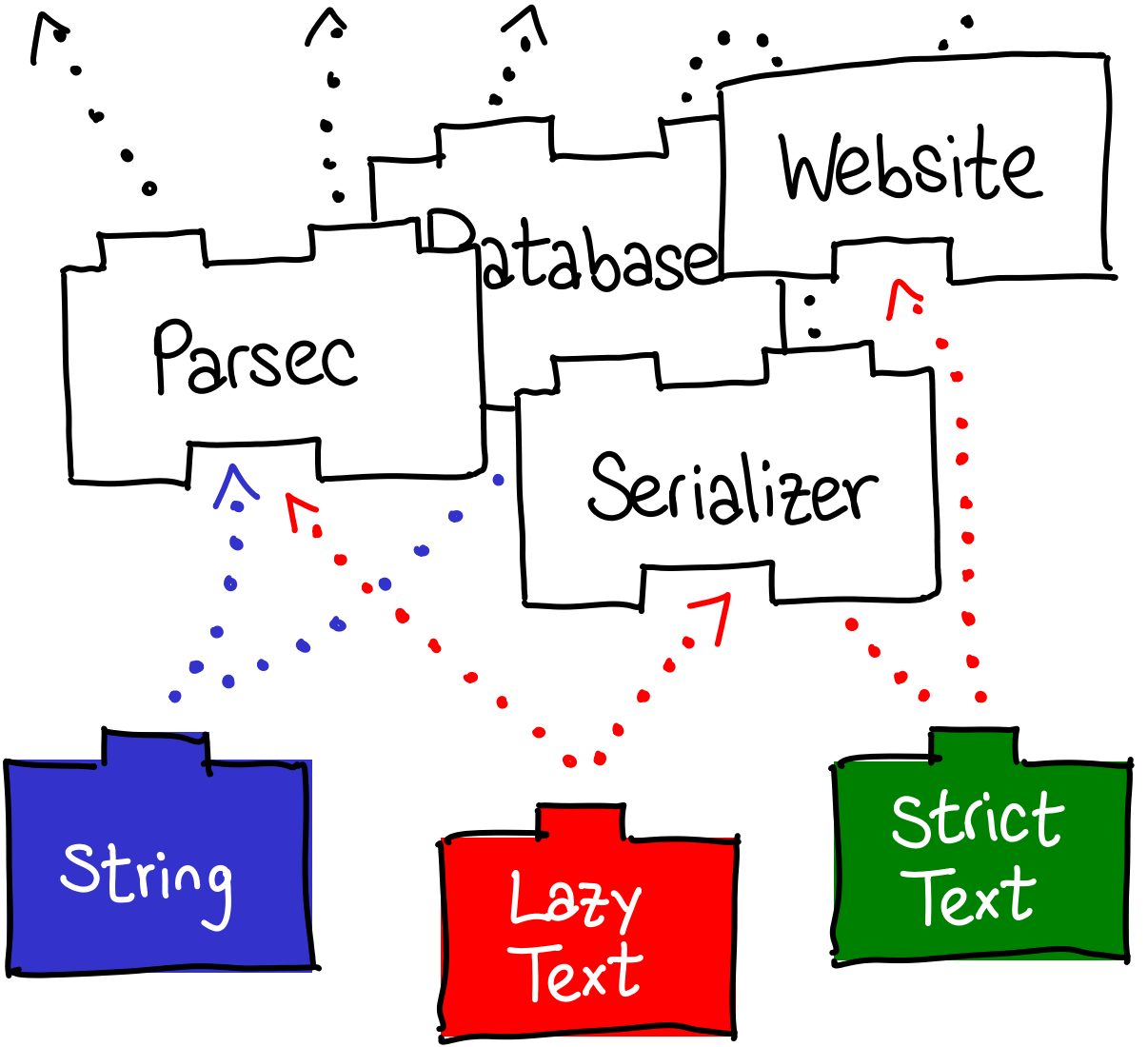
Simon Peyton Jones Scott Kilpatrick
Simon Marlow Derek Dreyer Duncan Coutts

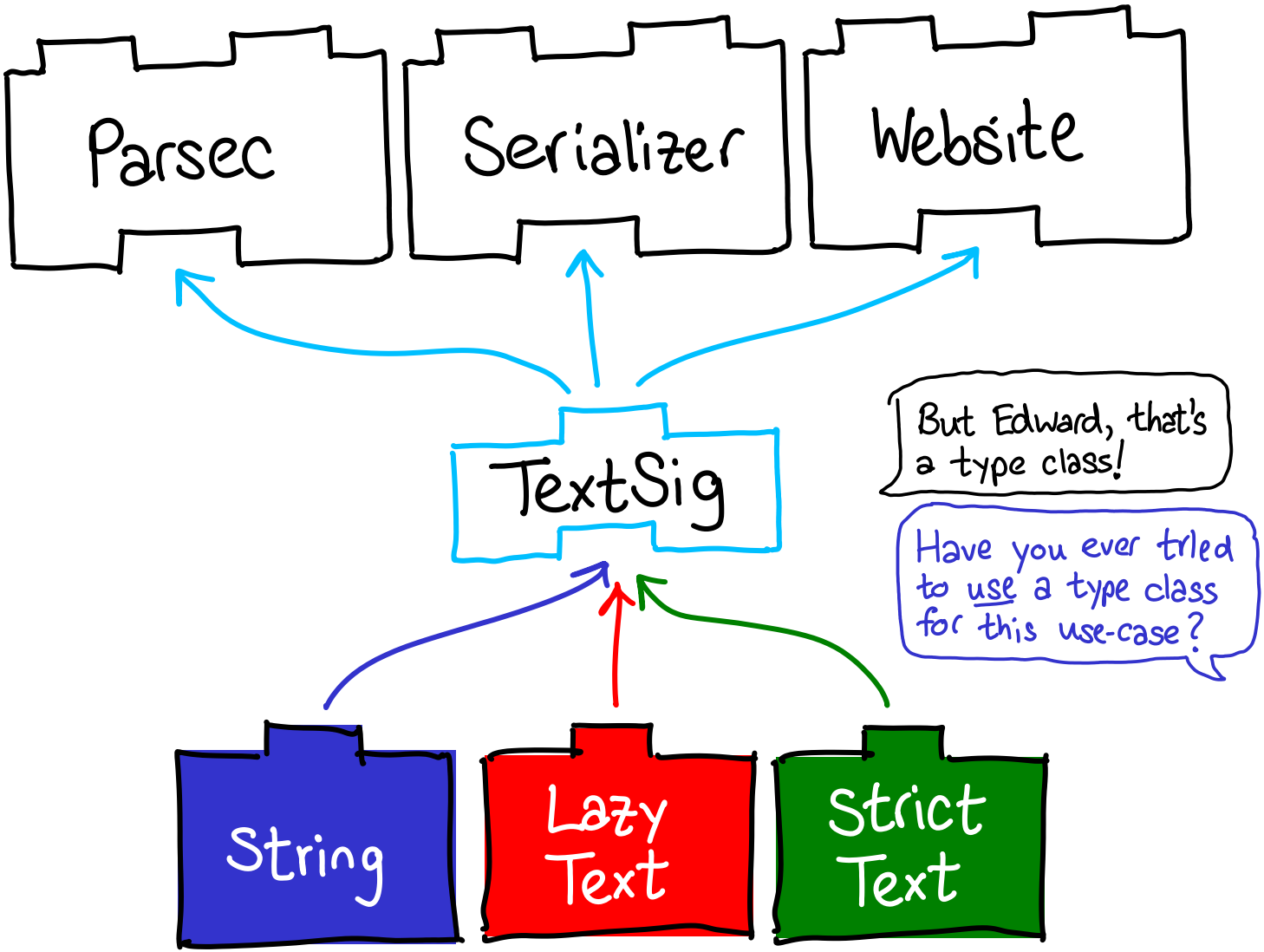
Why should you care about about module systems?

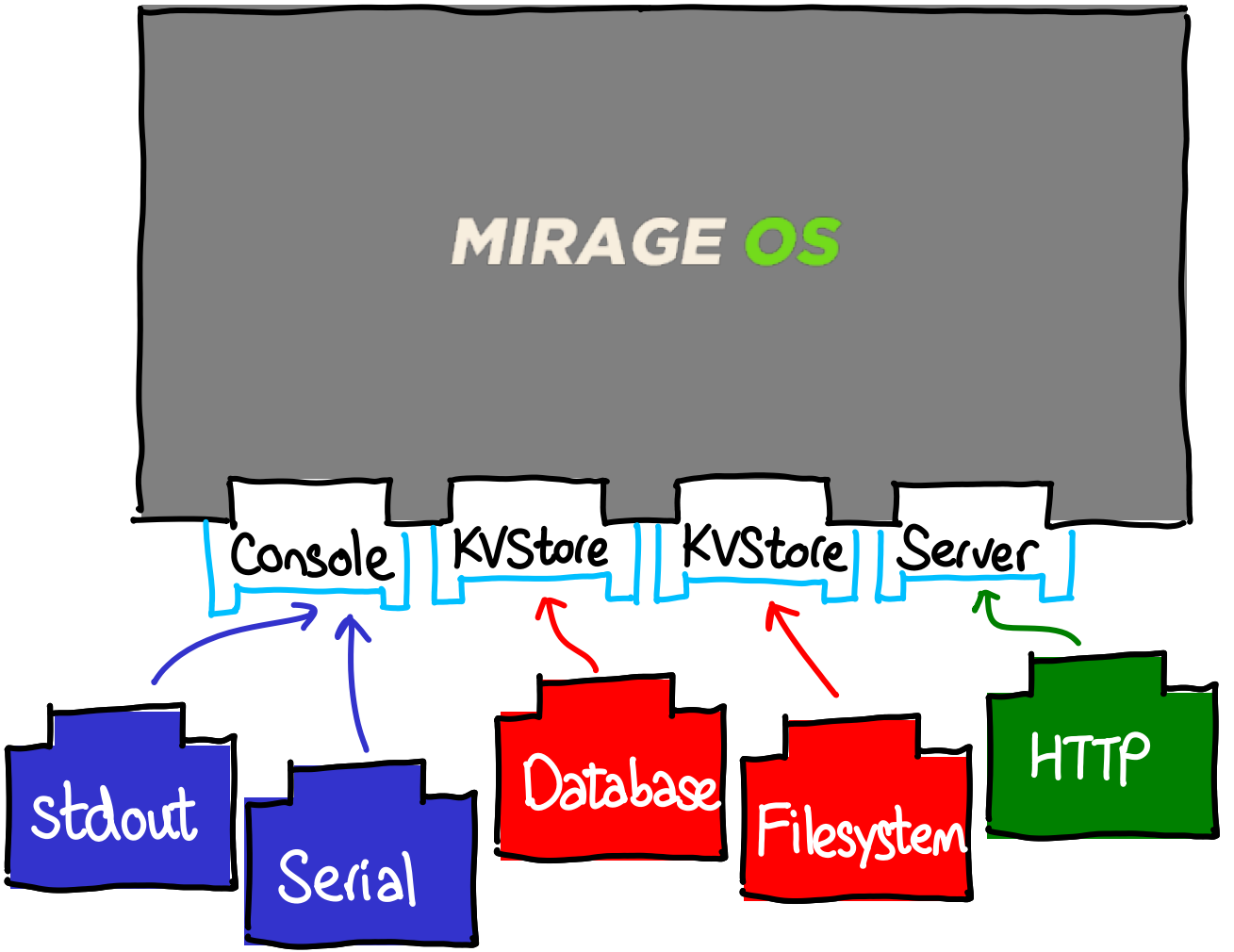

Application
Library
Library


Application
Library
Library


Application
Library
Library


Website
Database
Parsec
Serializer
String
Lazy Text
Strict Text


Parsec
Serializer
Website
TextSig
But Edward, that's a type class!
Have you ever tried to use a type class for this use-case?
String
Lazy Text
Strict Text


MIRAGE OS
Console
KVStore
KVStore
Server
stdout
Serial
Database
Filesystem
HTTP

```
$ cabal install hledger-0.18
```

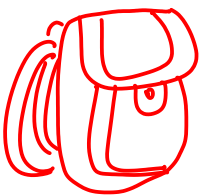
```
$ cabal install hledger-0.18
Resolving dependencies...
cabal: Could not resolve dependencies:
rejecting: hledger-0.19.4/installed-402..., 
0.18.2, 0.18.1 (global constraint requires =
trying: hledger-0.18
trying: regexpr-0.5.4/installed-0da...
trying: process-1.1.0.2/installed-7b6...
rejecting: haskeline-0.7.0.3/installed-414..
0.7.0.0 (conflict: hledger => haskeline==0.6
rejecting: haskeline-0.6.4.7 (conflict: proc
unix==2.6.0.1/installed-ccb..., haskeline =>
rejecting: haskeline-0.6.4.6 (conflict: rege
```

```
$ cabal install hledger-0.18
Resolving dependencies...
Downloading hledger-0.18...
Configuring hledger-0.18...
Building hledger-0.18...

 Hledger/Cli/Options.hs:20:7:
     Couldn't match expected type `LibType' w
     In the first argument of `libfoo' namel
Failed to install hledger-0.18
```

Yes, Stable Distributions help, but we also need tools to wrangle the wild-wild west.

Backpack



a new module system for Haskell

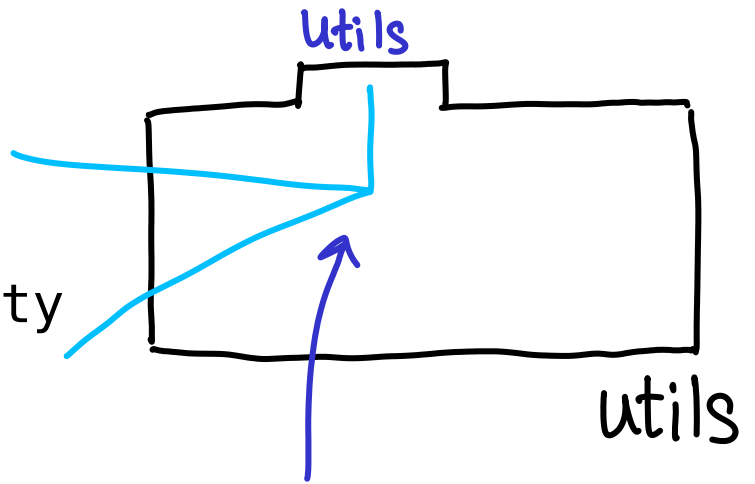
there a lot of things I could talk about...

but let me tell
you how it works...

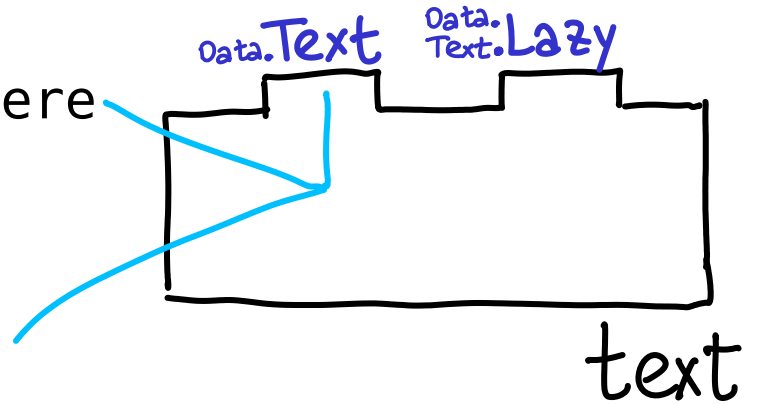
did you spend too long on the intro again?

```
module Utils where
  import Data.Text
  blank = putStr empty
```

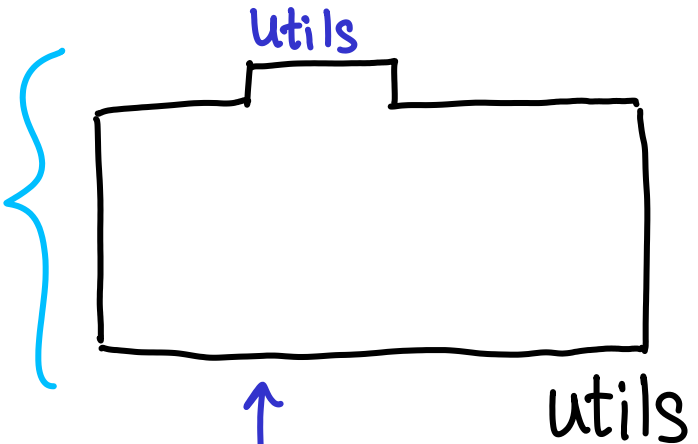
what a useless function!



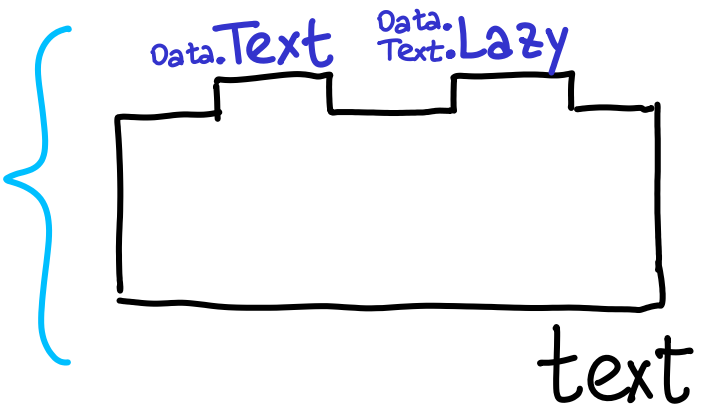

```
module Data.Text where
  data Text = ...
  empty = Text ...
  putStr s = ...
```



```
name: utils
build-depends:
  text
exposed-modules:
  Utils
```

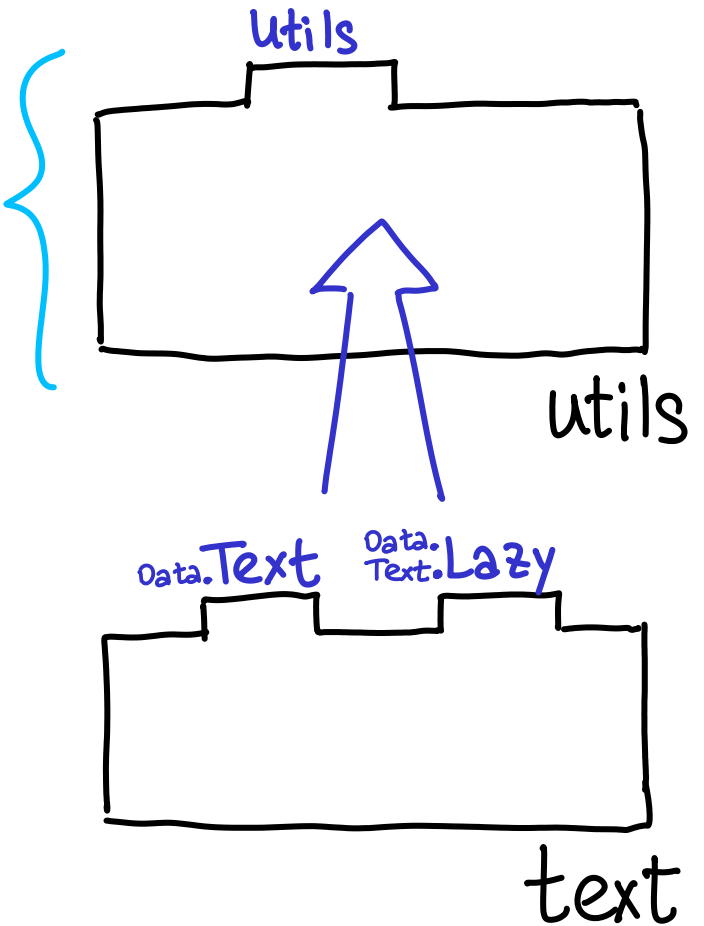


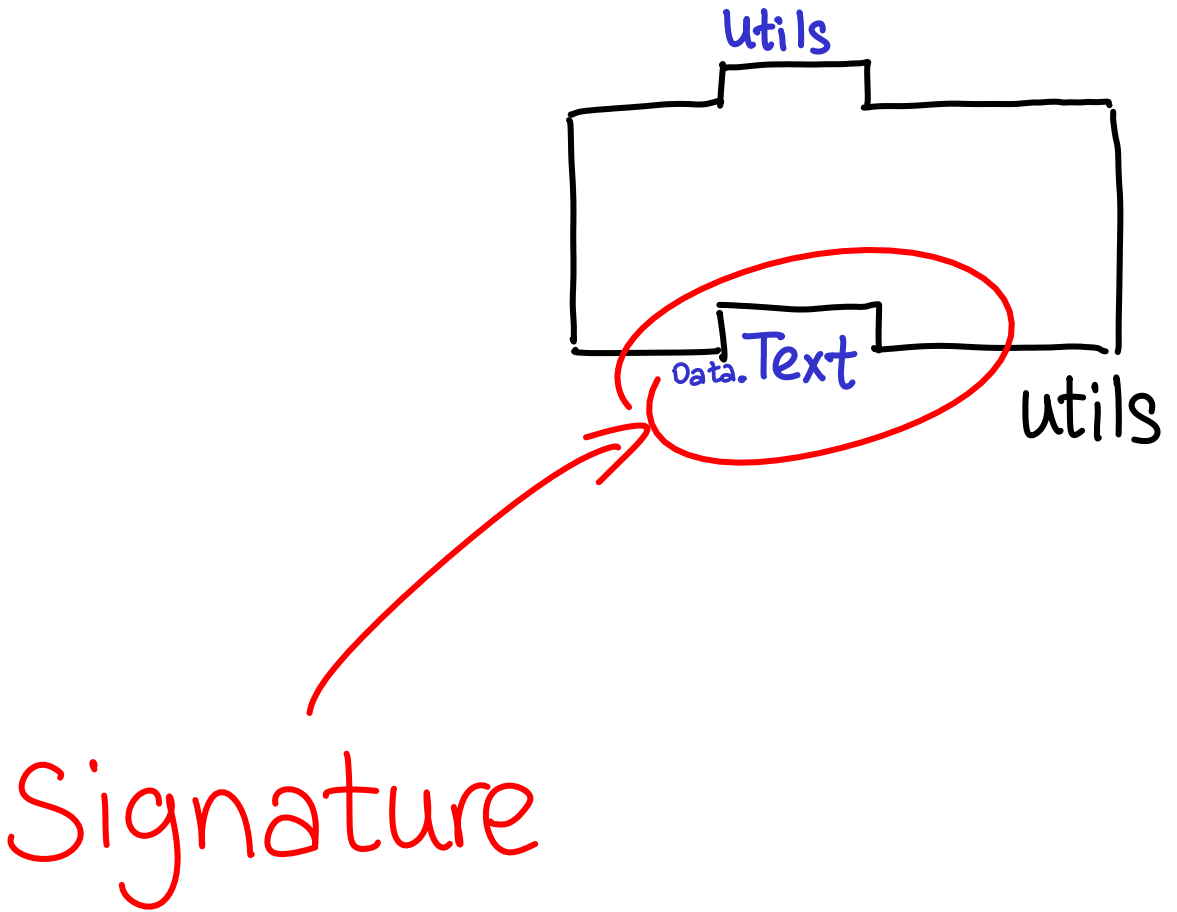

```
name: text
exposed-modules:
  Data.Text
  Data.Text.Lazy
```



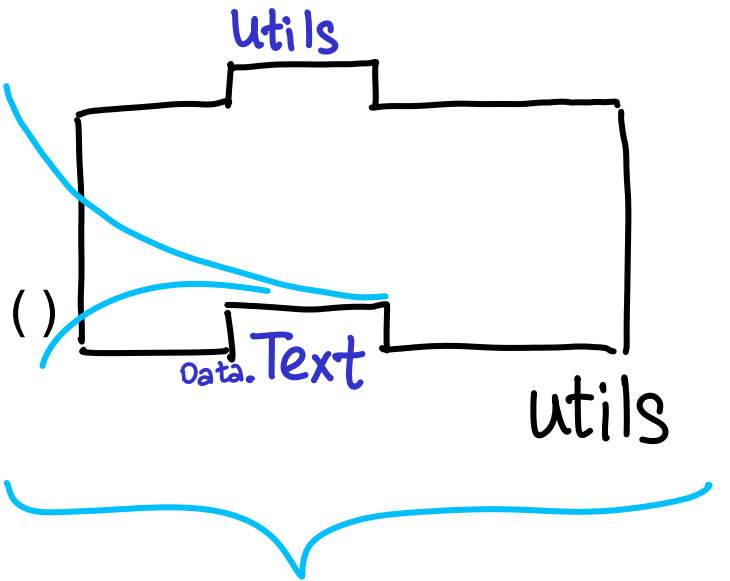
```
name: utils
build-depends:
  text
exposed-modules:
  Utils
```

hard-coded!

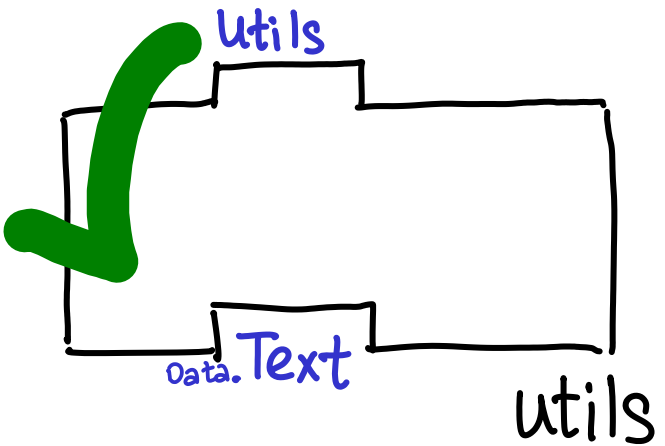



Utils
Data.Text
utils
Signature

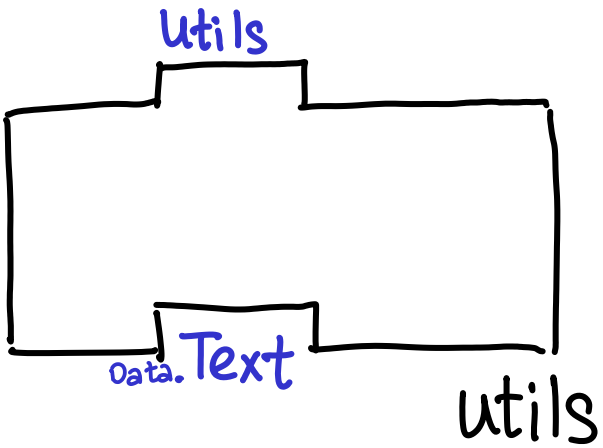
```
module Data.Text where
  data Text
  empty :: Text
  putStr :: Text -> IO ()
```

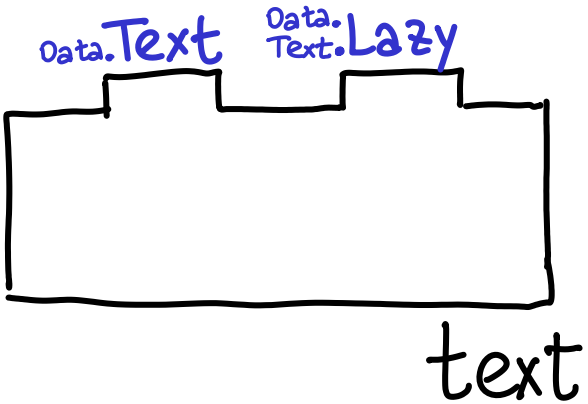



```
name: utilities
required-signatures:
  Data.Text
exposed-modules:
  Utils
```




```
[1 of 1] Checking Utils (Utils.hs, nothing)
```


Utils
Data.Text
utils


Data.Text
Data.Text.Lazy
text

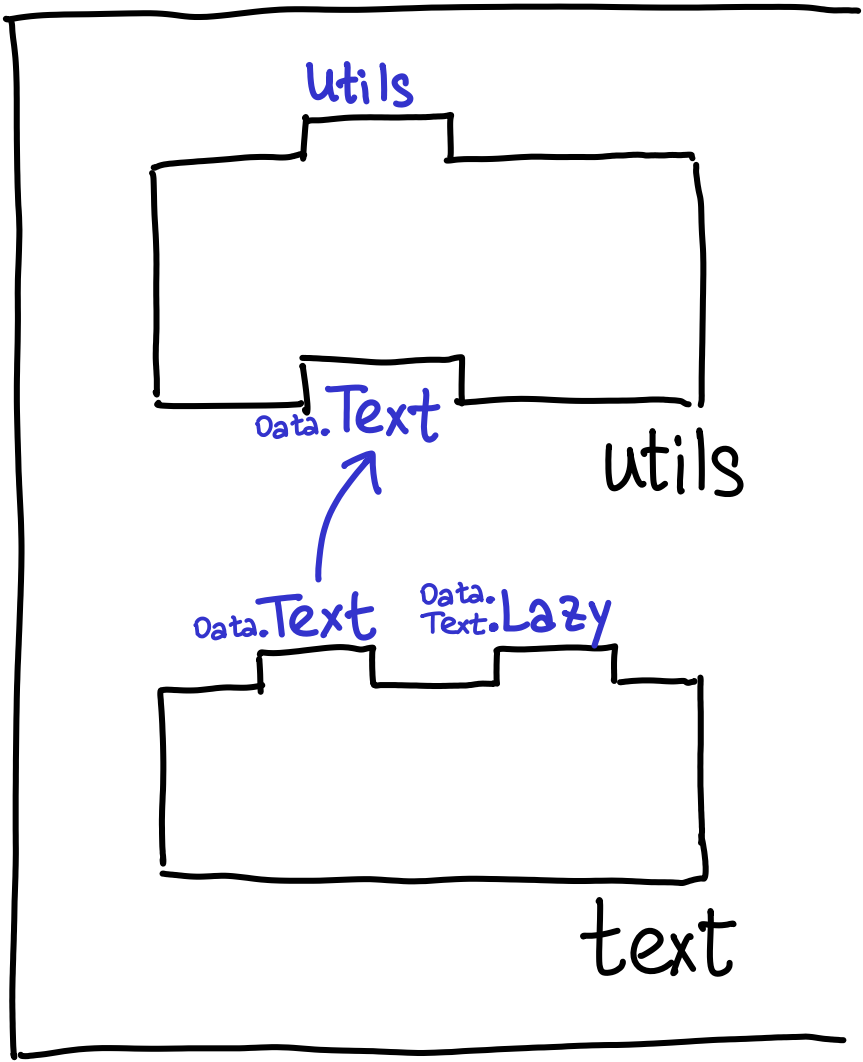
Mix-in linking

configuration by convention

```
name: utils-strict
build-depends:
  utils,
  text
```

or you could write a DSL

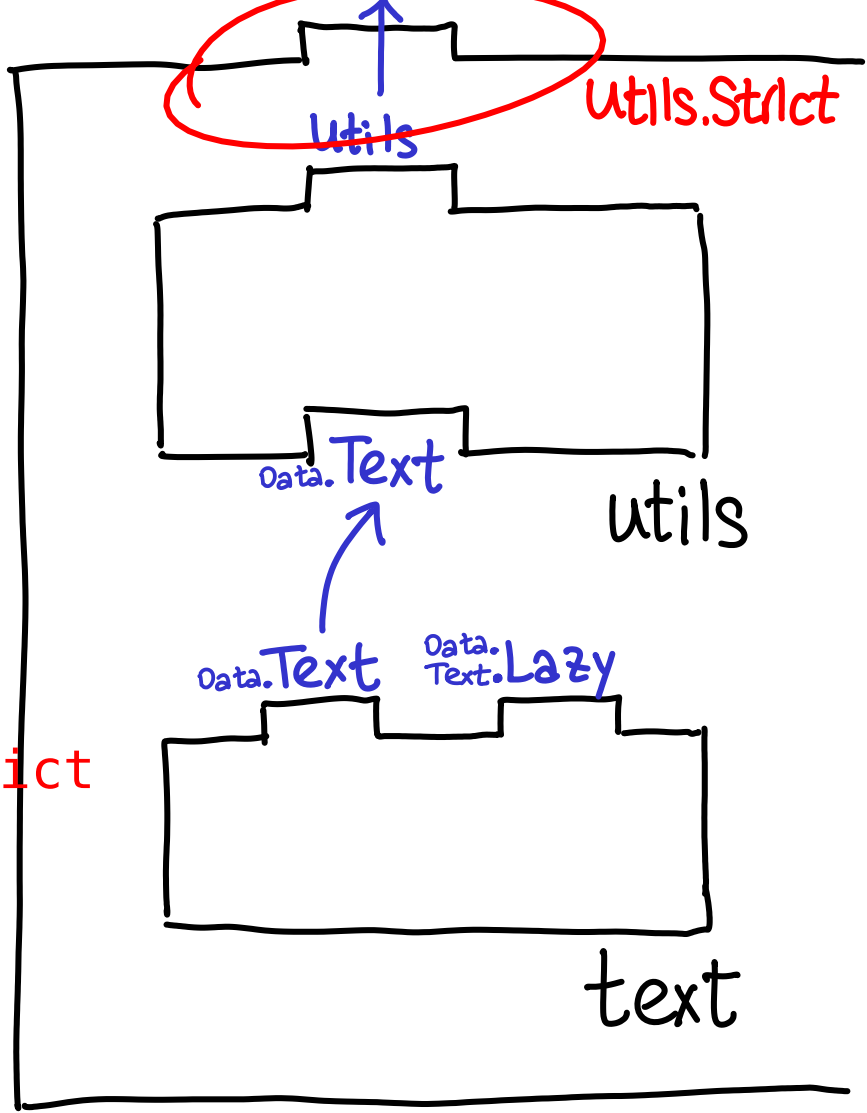
utils-strict



Mix-in linking

configuration by convention

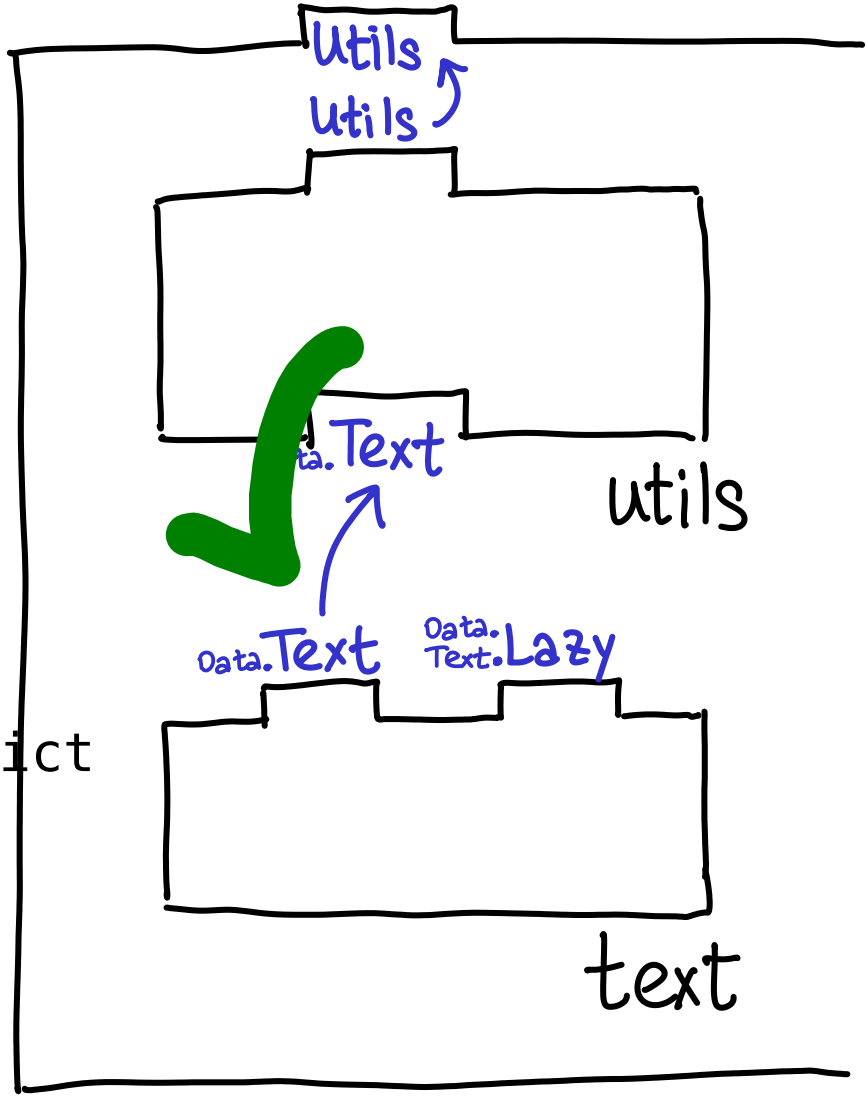
```
name: utils-strict
build-depends:
  utils,
  text
reexported-modules:
  Utils as Utils.Strict
```



Mix-in linking

configuration by convention

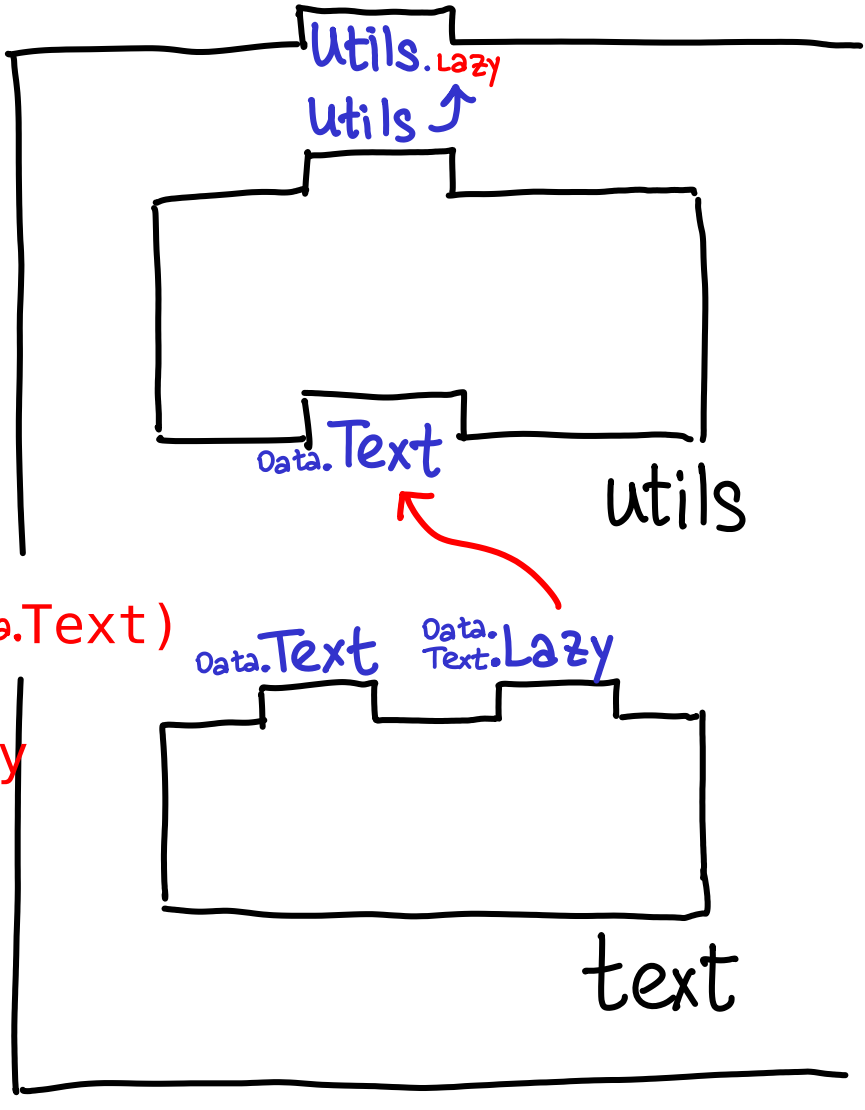
```
name: utils-strict
build-depends:
  utils,
  text
reexported-modules:
  Utils as Utils.Strict
```

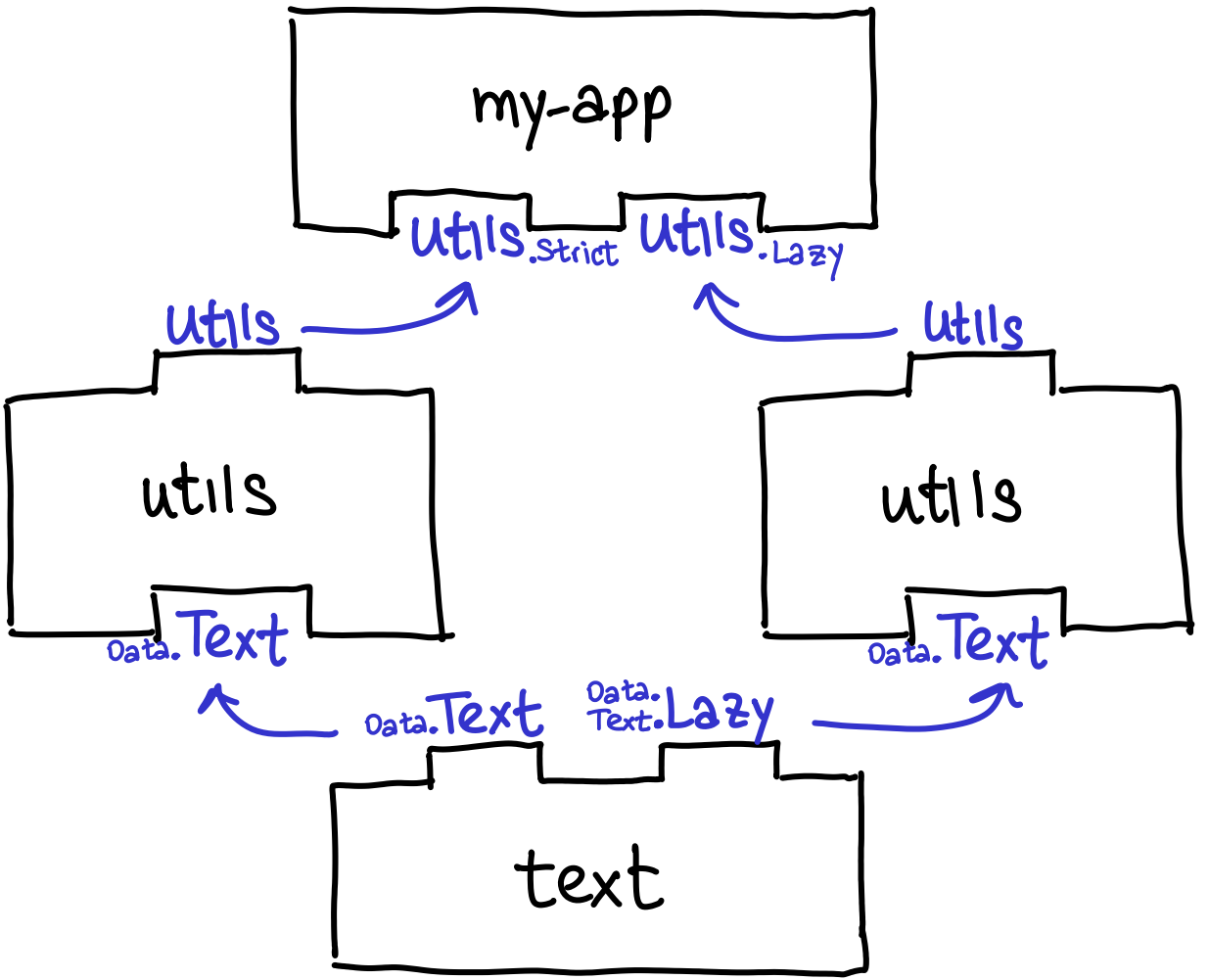


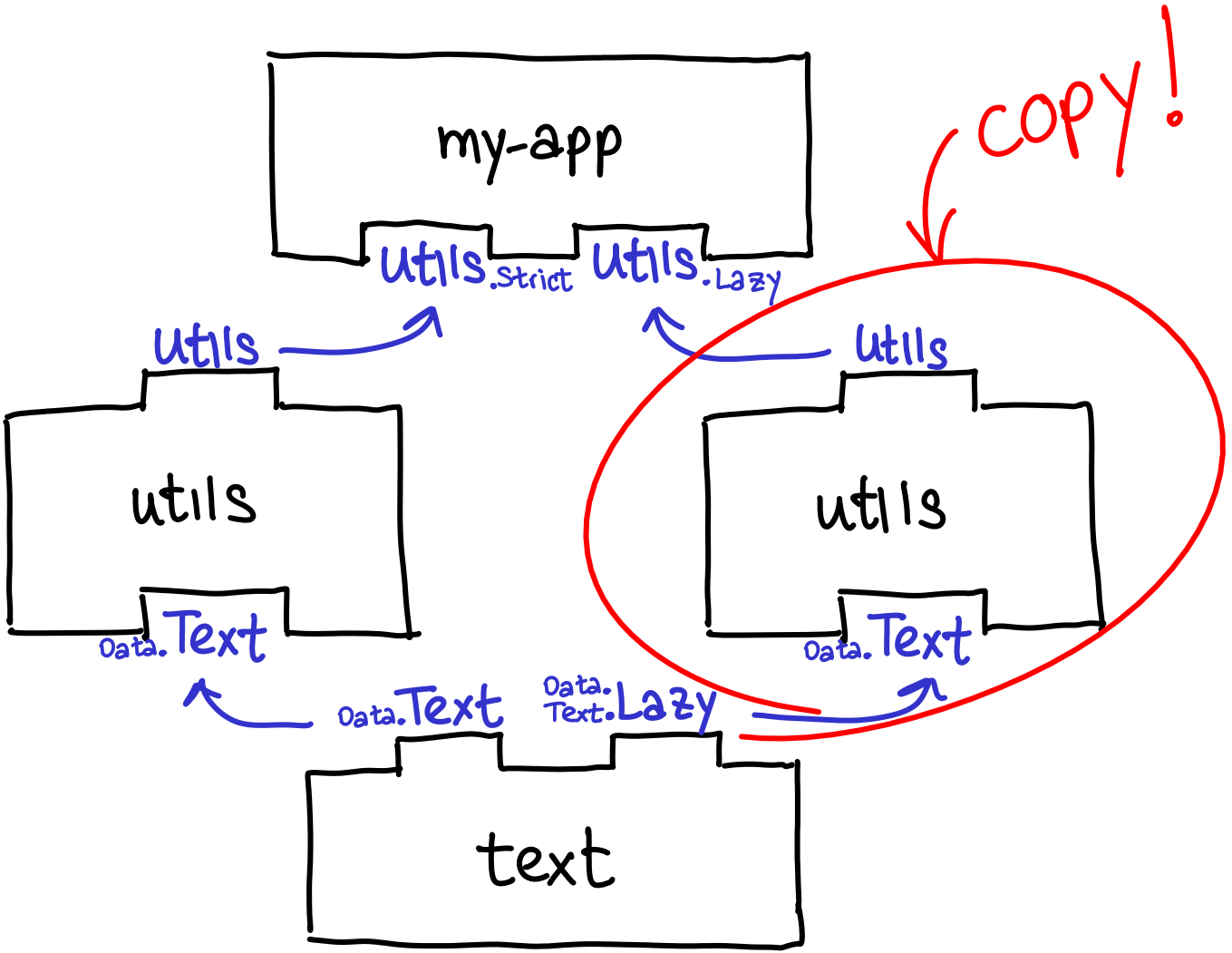
Mix-in linking

configuration by convention

```
name: utils-lazy
build-depends:
  utils,
  text (Data.Text.Lazy as Data.Text)
reexported-modules:
  Utils as Utils.Lazy
```

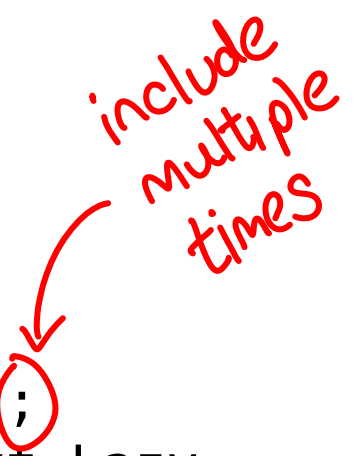



my-app
Utils.Strict
Utils.Lazy
Utils
utils
Data.Text
Utils
utils
Data.Text
Data.Text
Data.Text.Lazy
text


my-app
Utils.Strict
Utils.Lazy
copy!
Utils
utils
Data.Text
Utils
utils
Data.Text
Data.Text
Data.Text.Lazy
text

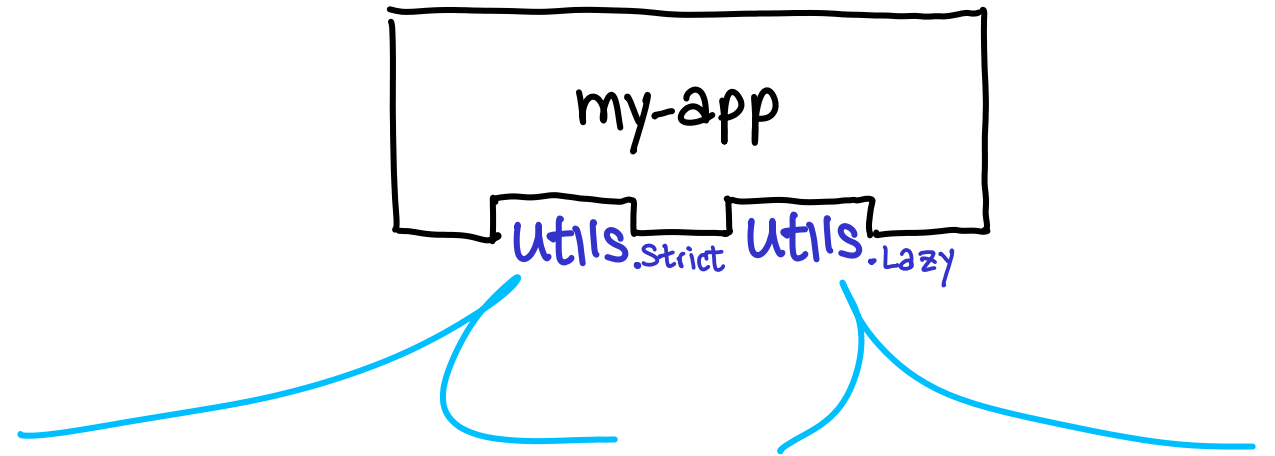
```
name: my-app
build-depends:
  utils
    (Data.Text,
     Utils as Utils.Strict;
     Data.Text as Data.Text.Lazy
     Utils as Utils.Lazy),
  text
```

```
name: my-app
build-depends:
  utils
    (Data.Text,
     Utils as Utils.Strict;
     Data.Text as Data.Text.Lazy
     Utils as Utils.Lazy),
  text
```



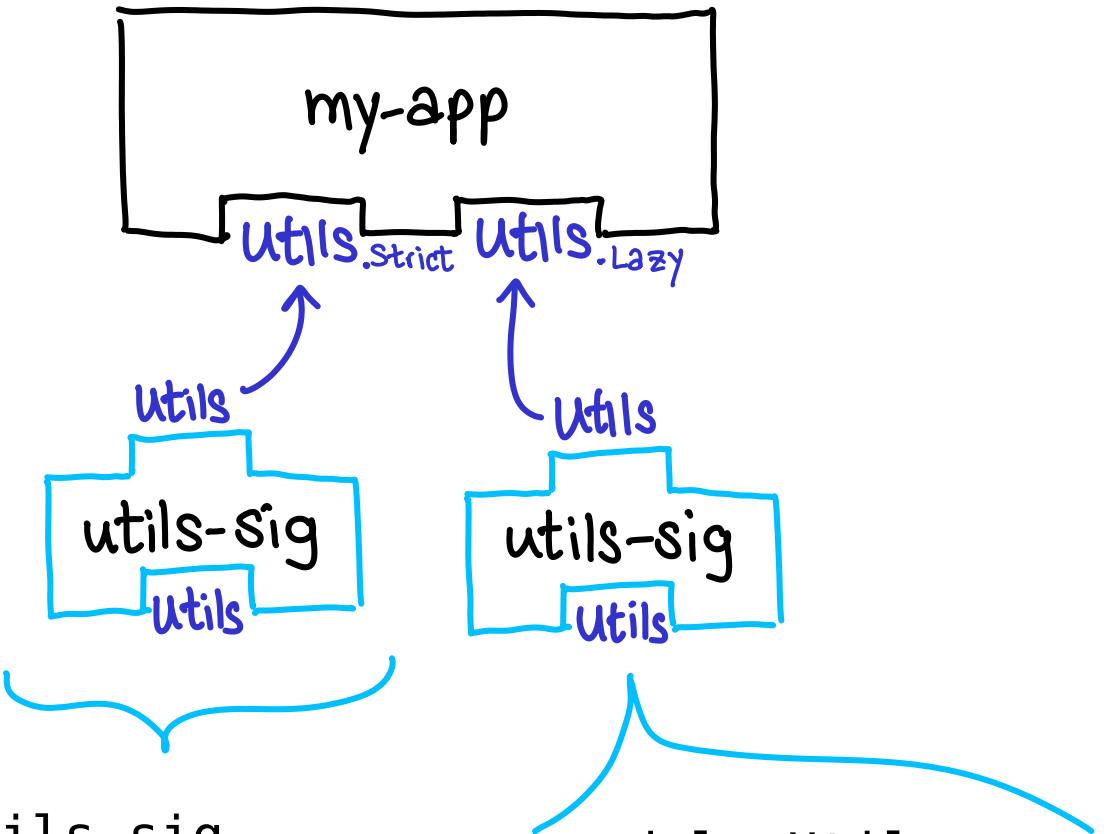
```
name: my-app
build-depends:
  utils
    (Data.Text,
     Utils as Utils.Strict;
     Data.Text as Data.Text.Lazy
     Utils as Utils.Lazy),
  text
```

holes can be renamed too!



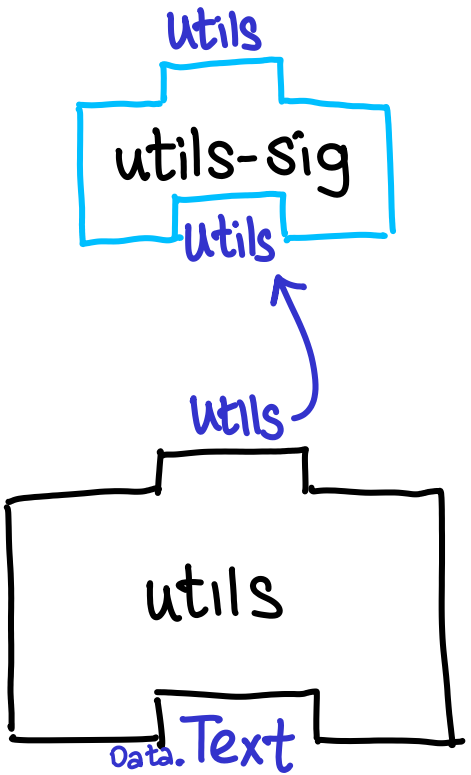

```
module Utils.Strict
  blank :: IO ()
```

```
module Utils.Lazy
  blank :: IO ()
```

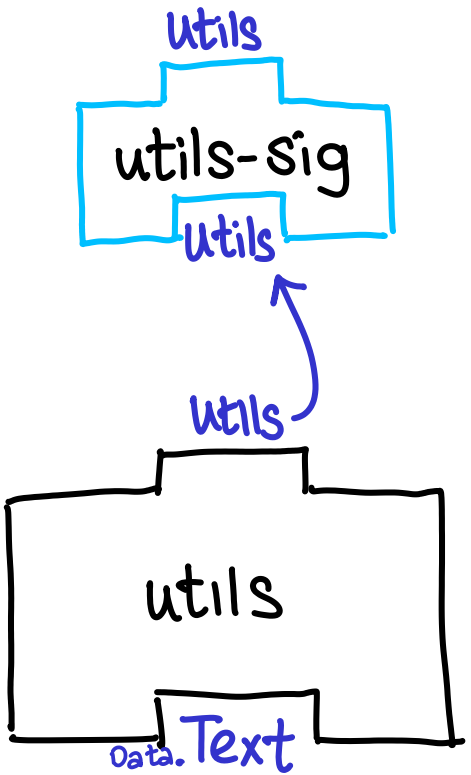



```
name: utils-sig
exported-signatures:
  Utils
```

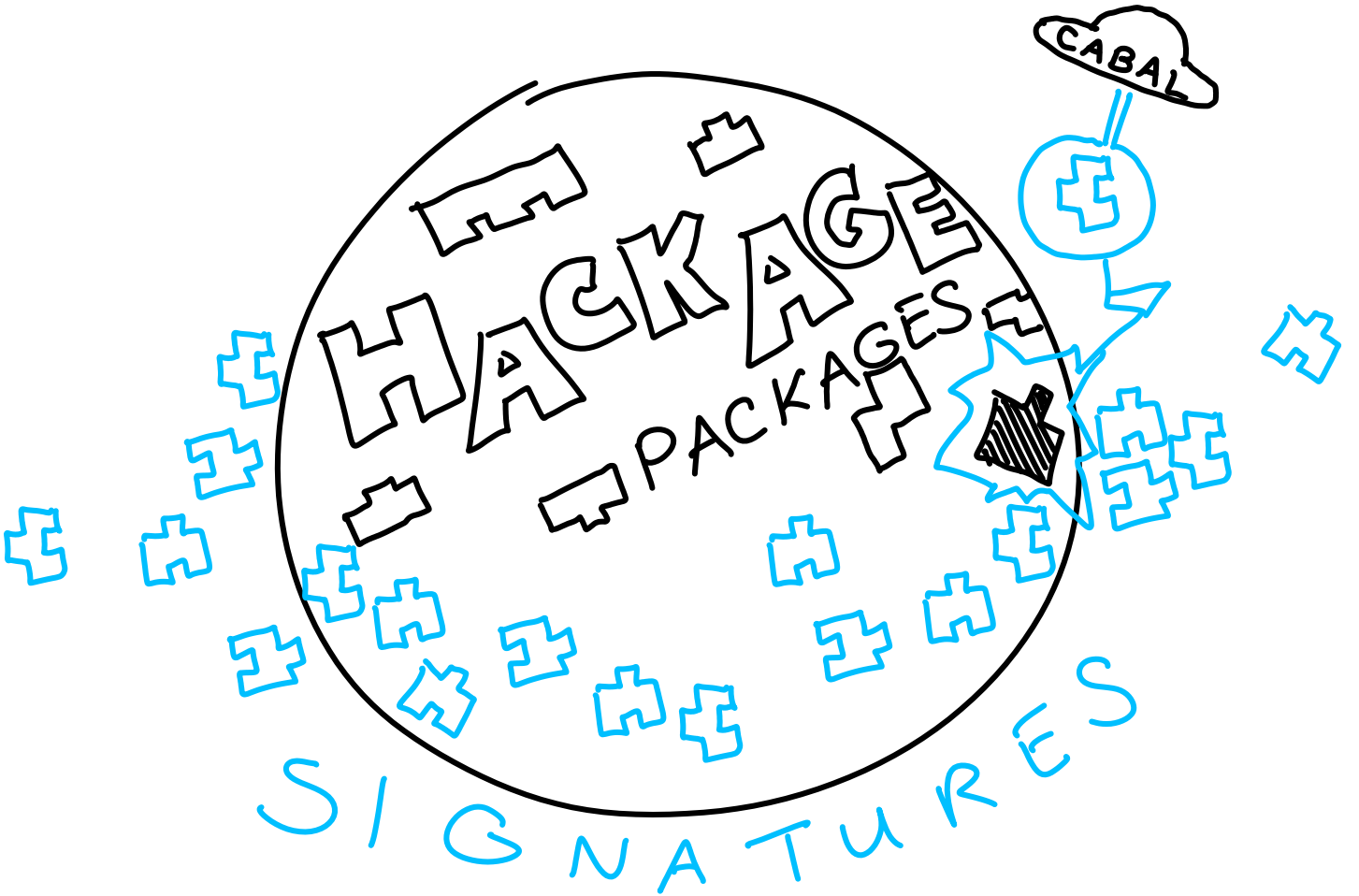
```
module Utils
  blank :: IO ()
```




```
name: utils
exported-modules:
  Utils
implements: utils-sig
build-requires:
  text-sig
```




```
name: utils
exported-modules:
  Utils
implements: utils-sig-3
build-requires:
  text-sig-0.11
```


CABAL
HACKAGE
PACKAGES
SIGNATURES

ok... so what did we just see?

Signature modules

mix-in linking

reexported modules

module renaming

multiple package instances

Signature modules	#9252
mix-in linking	(Cabal)
reexported modules	#8407
module renaming	#9375
multiple package instances	#9267

Signature modules	#9252 60%
mix-in linking	(Cabal) 40%
reexported modules	#8407 100%
module renaming	#9375 90%
multiple package instances	#9267 100%

On the docket

type classes/families

shaping

lots of refactoring

On the docket

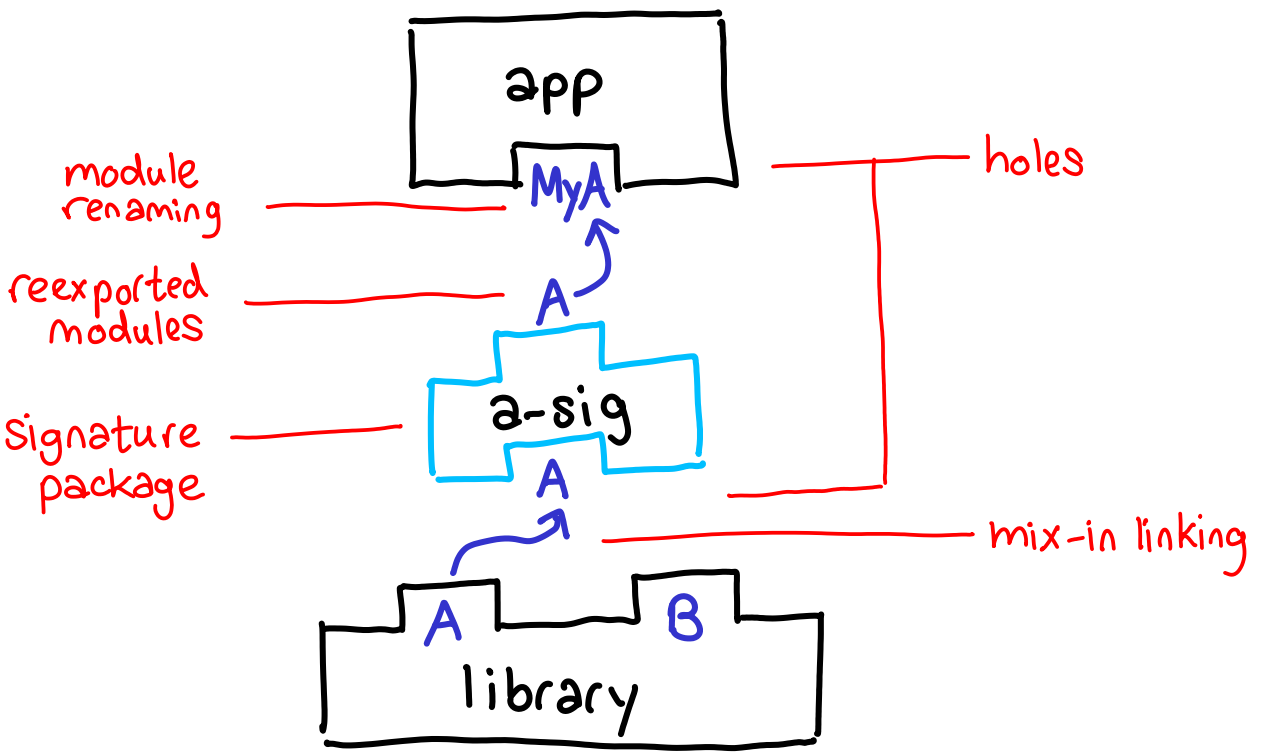
type classes/families

shaping

lots of refactoring

and you!

Thanks!



⇨ see GHC HEAD and blog.ezyang.com for more details ⇦