Backpack to Work:
Towards Backpack in Practice

Edward Z. Yang

- Backpack is a mix-in package system
- It's coming to GHC 8.2 (merge soon!)
- Big idea since the POPL'14 paper:

  Factor the design of mix-in packages into the language-agnostic & the language specific

Motivation

How Backpack works

Evaluation

Motivation: The String problem

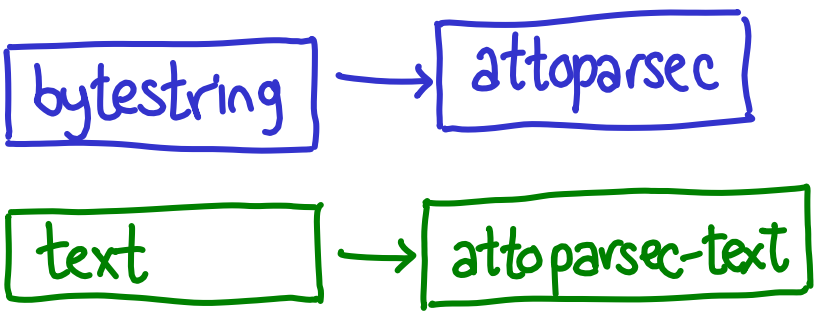
bytestring

text

(and String)

Motivation: The String problem

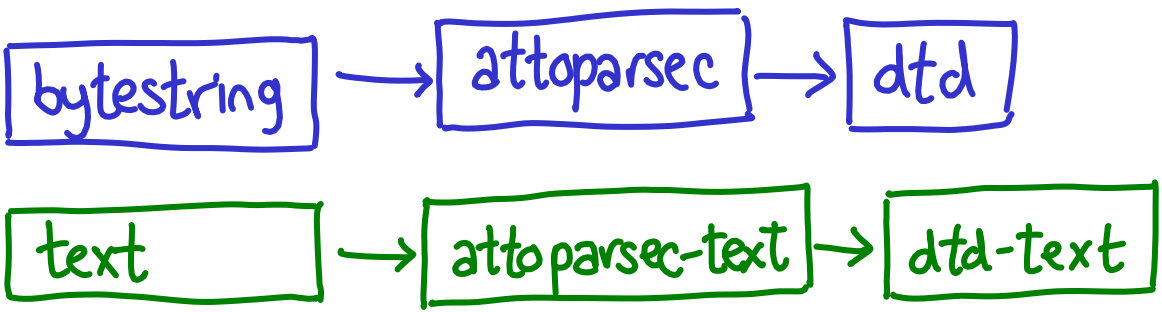
Everything must be implemented twice...



(note: attoparsec-text doesn't exist anymore, it was rolled into attoparsec, as is often done today)

Motivation: The String problem

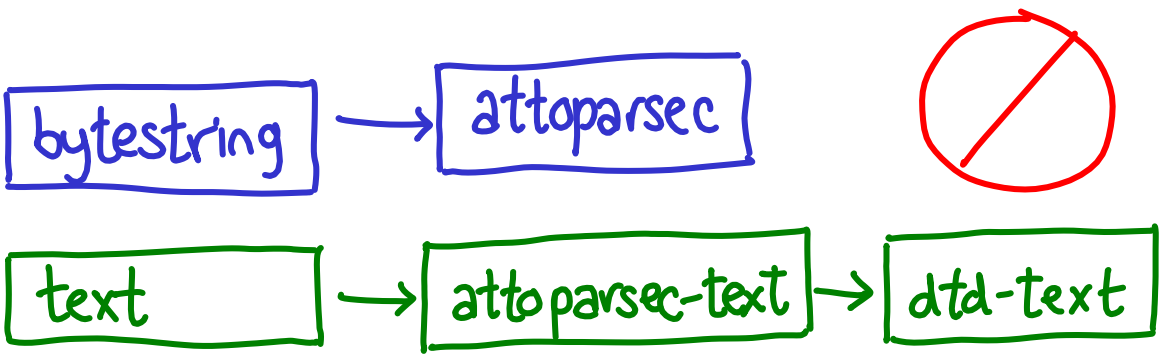
... transitively



Motivation: The String problem

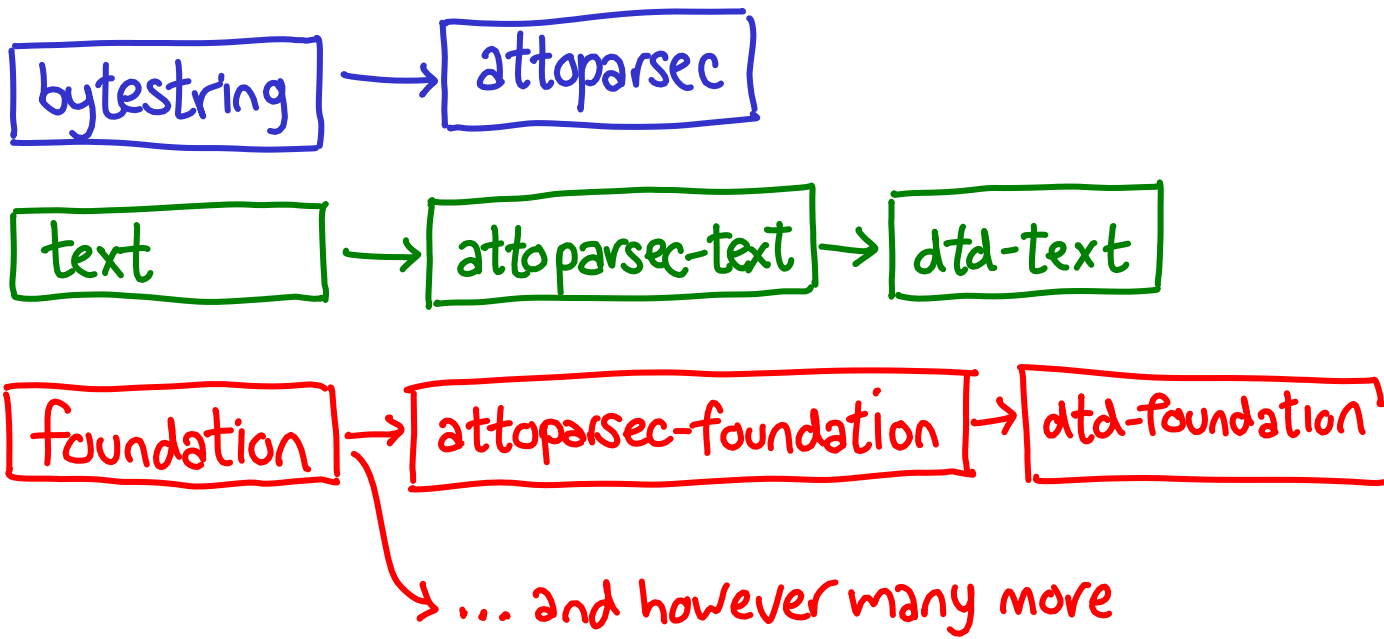
Things are often missing

maybe it doesn't exist!



Motivation: The String problem

Difficult to add new string types



Motivation: The String problem

https://hackage.haskell.org/package/unix

Modules

System

- System.Posix
  - System.Posix.ByteString
    - System.Posix.ByteString.FilePath
  - System.Posix.Directory
    - System.Posix.Directory.ByteString
  - System.Posix.DynamicLinker
    - System.Posix.DynamicLinker.ByteString
    - System.Posix.DynamicLinker.Module
      - System.Posix.DynamicLinker.Module.ByteString
    - System.Posix.DynamicLinker.Prim
  - System.Posix.Env
    - System.Posix.Env.ByteString
  - System.Posix.Error
  - System.Posix.Fcntl
  - System.Posix.Files
    - System.Posix.Files.ByteString
  - System.Posix.IO
    - System.Posix.IO.ByteString
  - System.Posix.Process

Motivation: The String problem

System.Posix.Internal

```
getEnv :: String -> IO (Maybe String)
getEnv name = do
  litstring <- withFilePath name c_getenv
  if litstring /= nullPtr
     then liftM Just $ peekFilePath litstring
     else return Nothing
```

Data.ByteString

```
getEnv :: ByteString -> IO (Maybe ByteString)
getEnv name = do
  litstring <- B.useAsCString name c_getenv
  if litstring /= nullPtr
     then liftM Just $ B.packCString litstring
     else return Nothing
```

Motivation: The String problem

Oops

```
putEnv :: String -> IO ()
putEnv keyvalue = do
  s <- newFilePath keyvalue
  -- Do not free `s` after calling putenv.
  -- According to SUSv2, the string passed to putenv
  -- becomes part of the environment. #7342
  throwErrnoIfMinus1_ "putenv" (c_putenv s)
```

```
putEnv :: ByteString -> IO ()
putEnv keyvalue = B.useAsCString keyvalue $ \s ->
  throwErrnoIfMinus1_ "putenv" (c_putenv s)
```

https://github.com/haskell/unix/issues/68

What can we do?

Backpack: The Str signature

exported module

```
name: attoparsec-indef
library
  signatures: Str
  exposed-modules: ...
```

bytestring } Str

text } Str

foundation } Str

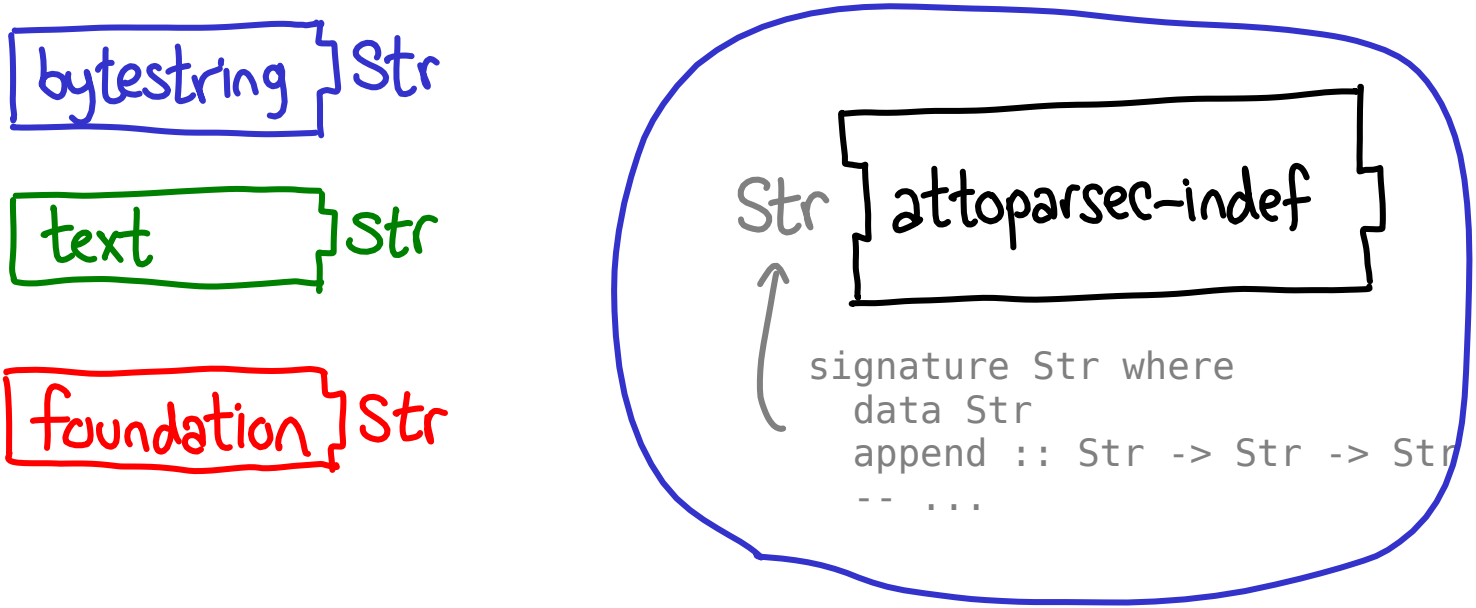
Str } attoparsec-indef

```
signature Str where
  data Str
  append :: Str -> Str -> Str
  -- ...
```

lightweight shims which have
type Str = {ByteString, Text, ...} (could be in other package)

Backpack: The Str signature

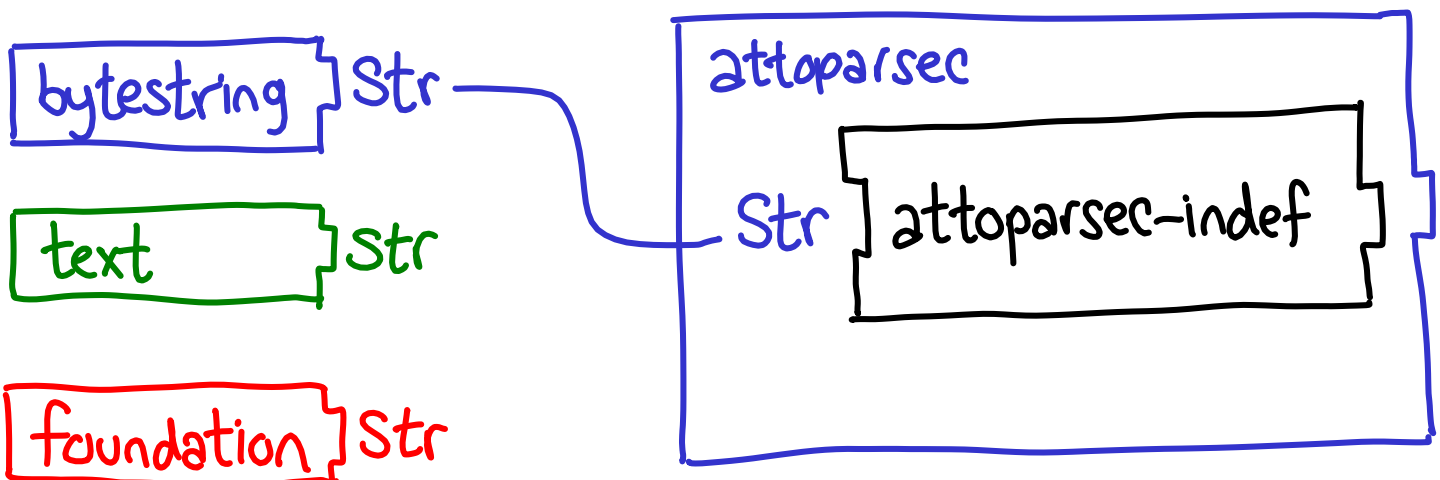
```
name: attoparsec-indef
library
  signatures: Str
  exposed-modules: ...
```



typecheckable in isolation

Backpack: The Str signature

instantiation gives us the original package

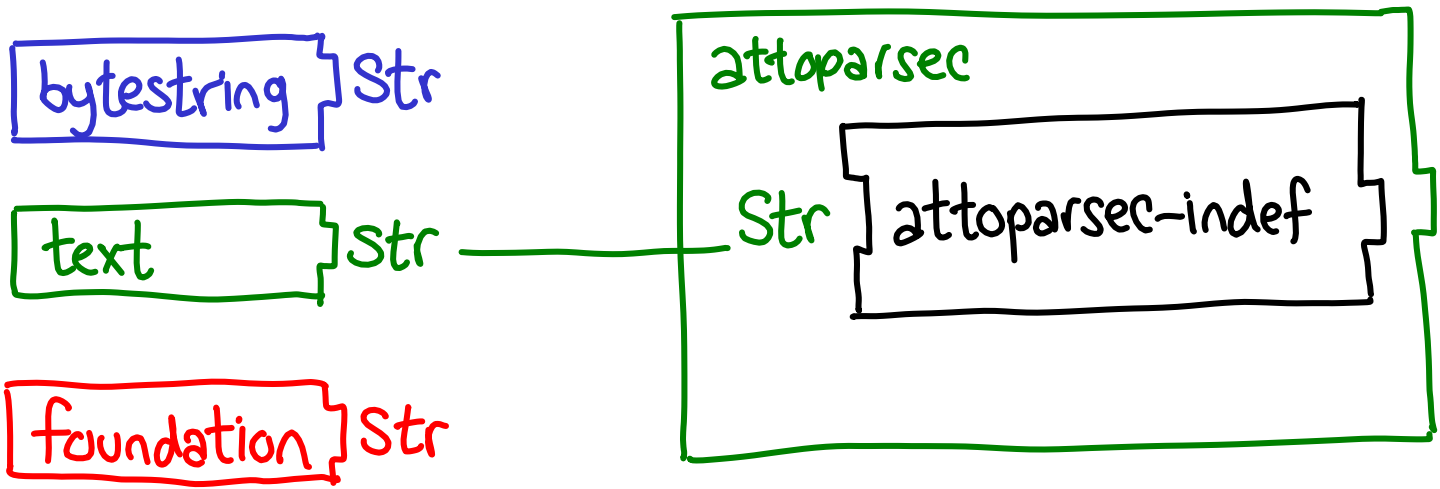



```
name: attoparsec
library
  build-depends: attoparsec-indef,
                 bytestring
```

(no performance cost!)

Backpack: The Str signature

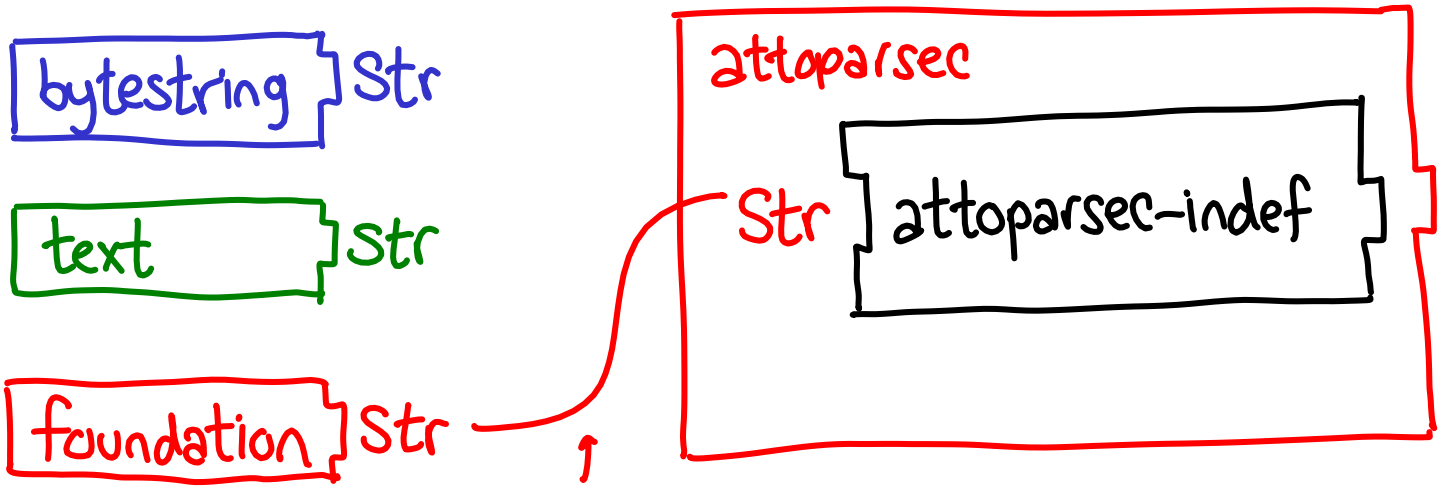
instantiation gives us the original package




```
name: attoparsec-text
library
  build-depends: attoparsec-indef,
                 text
```

Backpack: The Str signature

to add a new type, only need to check you match signature (modulo type class naughtiness)

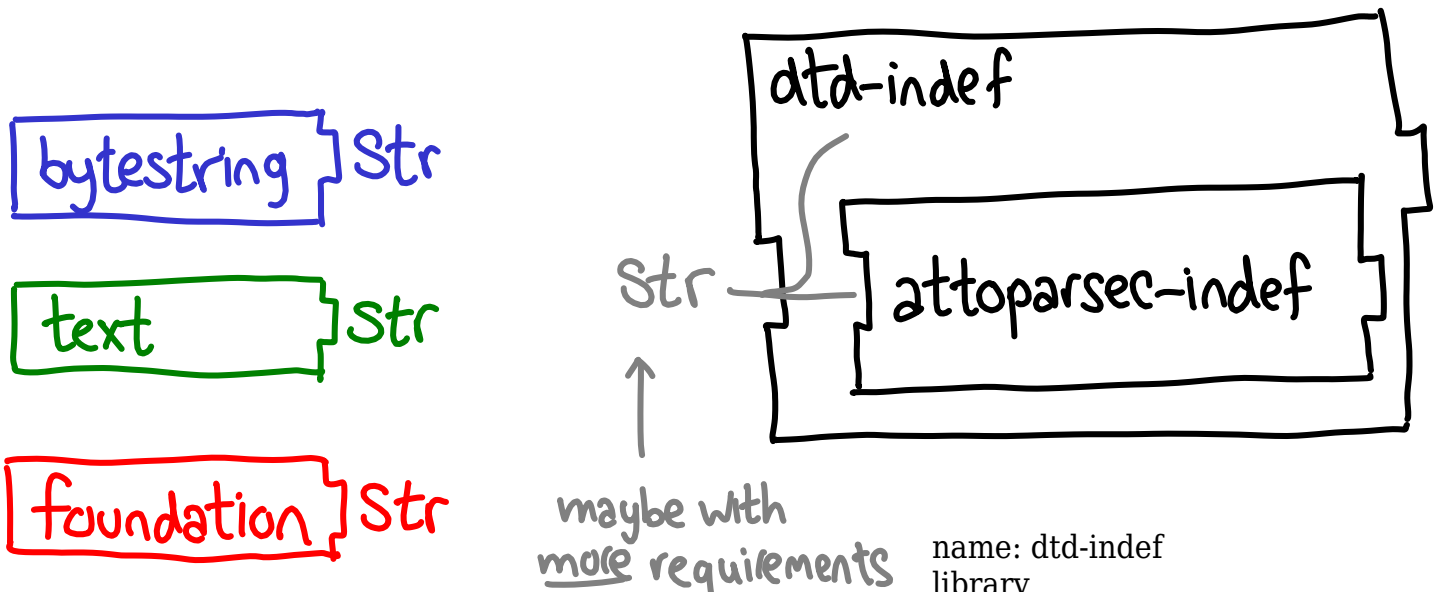


matches?

```
name: attoparsec-foundation
library
  build-depends: attoparsec-indef,
                 foundation
```

Backpack: Instantiation can be deferred

we can build an ecosystem parametric over Str




```
name: dtd-indef
library
  build-depends: attoparsec-indef
  exposed-modules: ...
```

the same as what you'd write normally

Idea: Packages, not modules

[POPL'14]

- A way to provide "strong modularity" without modifying the source language (too much)
- Modularity is for large scale programs; packages are the largest unit of development

  Look at SMLSC, which needs units <u>on top of</u> functors. OPAM ecosystem <u>not</u> fully functorized (MirageOS does better — and they have a DSL for functor application)
- Users can adopt Backpack with only modest changes to their code (BC-ish)
- Promising, under-explored point in the design space (interfaces ≥ version numbers)

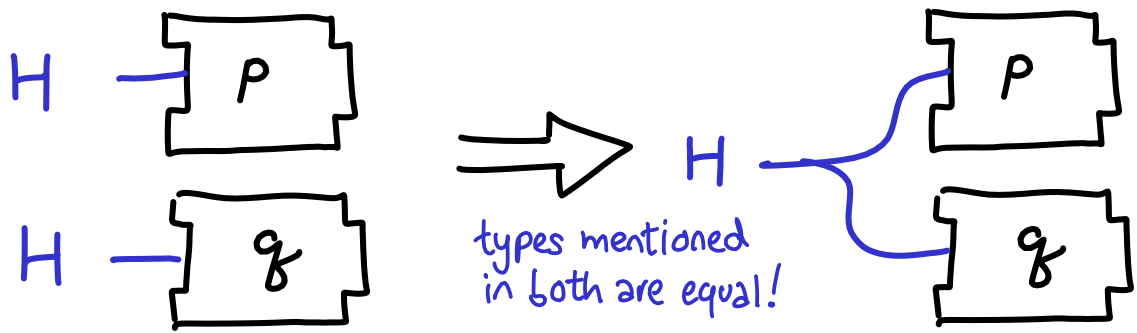
Idea: Mix-ins not functors

[POPL'14]

- Explicitly passing around functors gets old

  mix-ins = WildcardRecords + thinning/renaming

- Explicitly managing sharing constraints gets old



- Packages as "namespace management" well suited for mix-in linking

(Also: mutual recursion! But not for GHC 8.2)

How Backpack Works

The Backpack challenge:

Design an extension to the package manager and compiler interface (ahem) which can support Backpack.

package manager

abstraction barrier

compiler

(sorry Conor!)

package manager

Knows no evil (indifferent to source code)

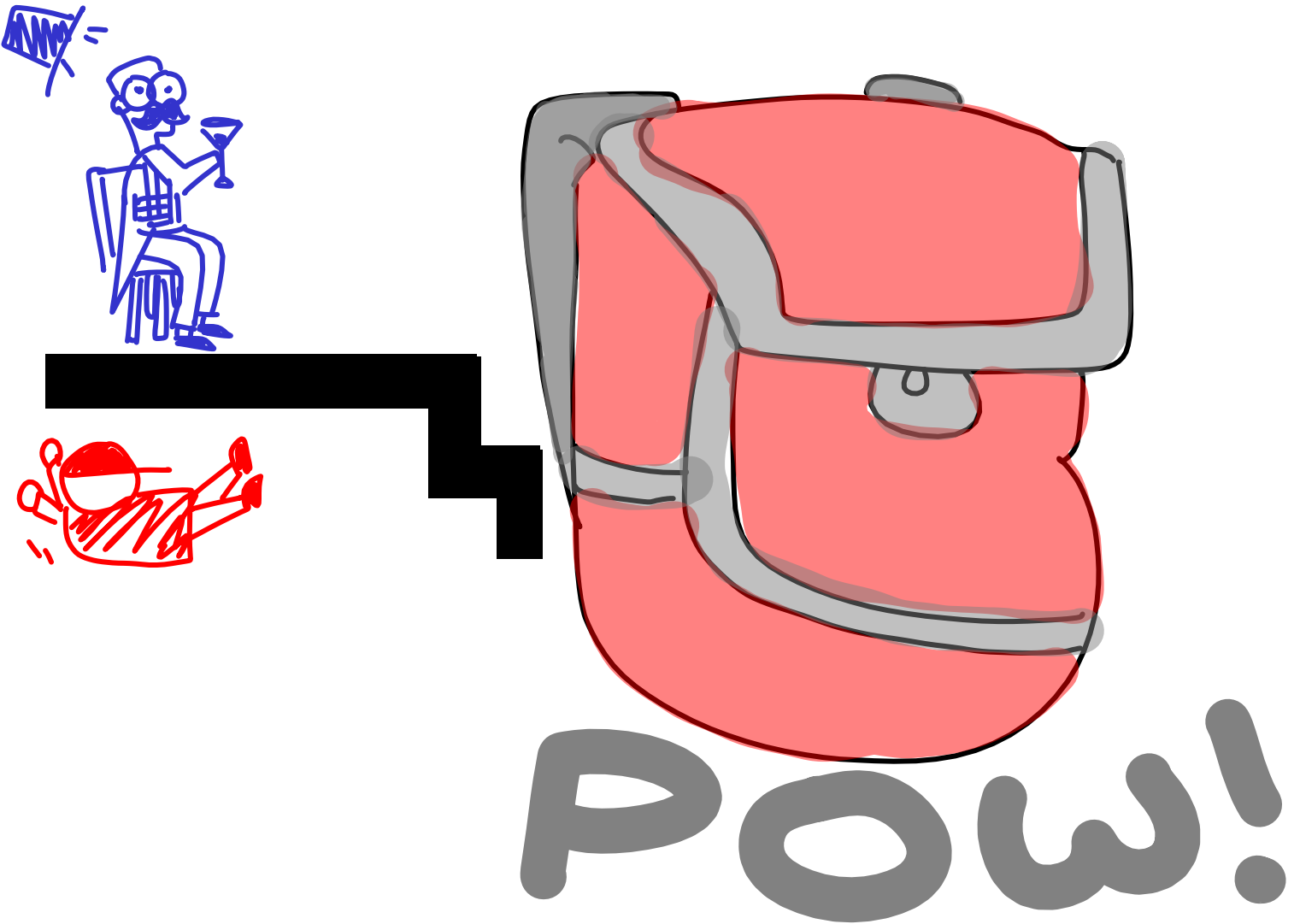
has a global view of the world

abstraction barrier

compiler

does all of the dirty work (understands source)

only looks at one compilation unit at a time


POW!

linking packages together

Modularity at the package level affects both the package manager and the compiler.

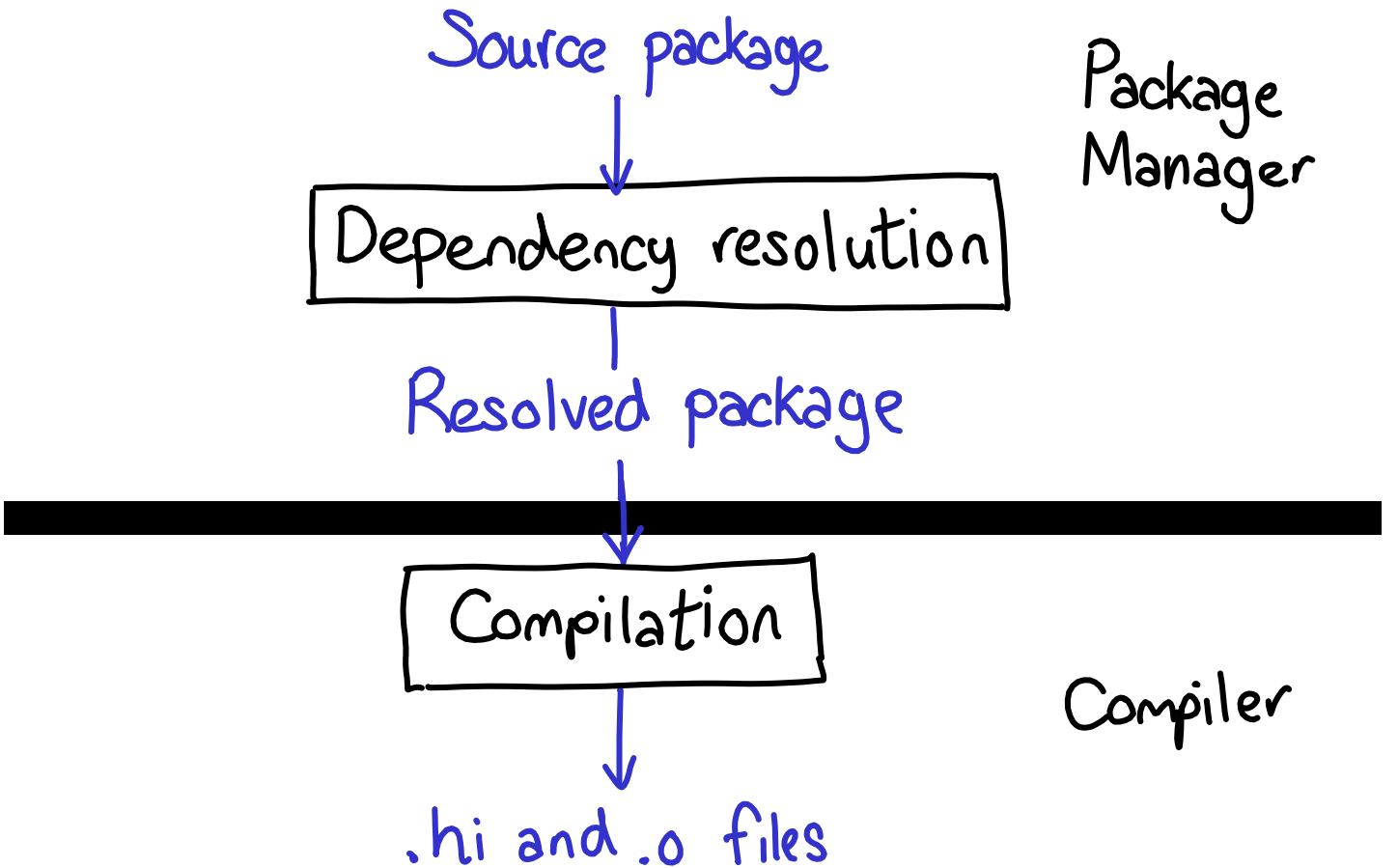
typechecking against interfaces / compiling with implementation

Modularity at the package level affects both the package manager and the compiler.

Backpack'14 demands a <u>lot</u> from the package manager

- Entirely new shaping pre-pass
- Pre-typechecking pass to generate hi files (no cross-mod opt)
- Merging operation on package types

The current abstraction



The current abstraction

cabal install attoparsec

Source package

The current abstraction

cabal install attoparsec

- (cd bytestring-0.9; ./Setup install ...)

- (cd attoparsec-0.13; ./Setup install ...)

Dependency resolution

↓

Resolved package

The current abstraction

cabal install attoparsec

- (cd bytestring-0.9; ./Setup install ...)
  - ghc --make Data.ByteString
    -this-package-id bytestring-0.9-aaa

- (cd attoparsec-0.13; ./Setup install ...)
  - ghc --make Data.Attoparsec ...
    -this-package-id attoparsec-0.13-bbb
    -package-id bytestring-0.9-aaa

Compilation

The current abstraction

cabal install attoparsec

- (cd bytestring-0.9; ./Setup install ...)
  - ghc --make Data.ByteString
    -this-package-id bytestring-0.9-aaa
    - ghc -c Data/ByteString.hs ''
- (cd attoparsec-0.13; ./Setup install ...)
  - ghc --make Data.Attoparsec Data.Attoparsec.Types
    -this-package-id attoparsec-0.13-bbb
    -package-id bytestring-0.9-aaa
    - ghc -c Data/Attoparsec/Types.hs ''
    - ghc -c Data/Attoparsec.hs

The current abstraction

Design a language to express these calls

cabal install attoparsec

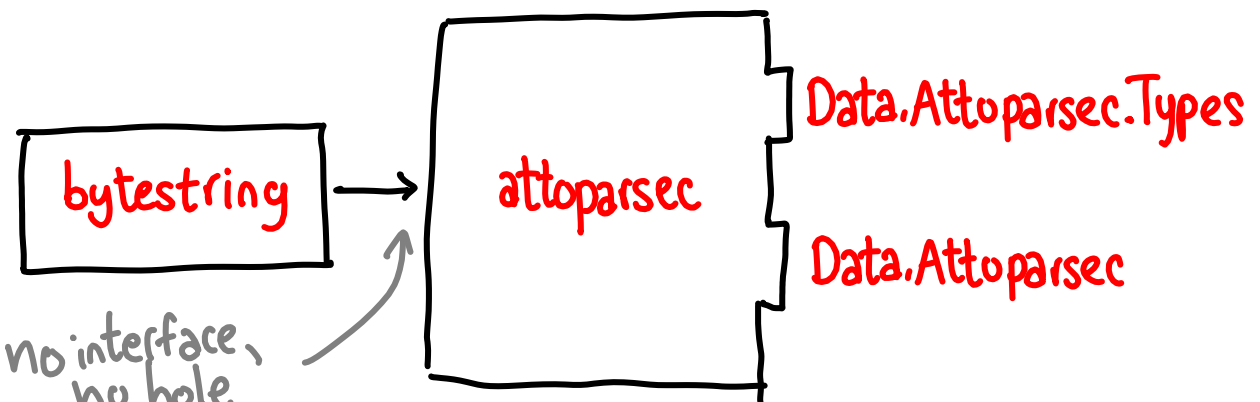
- (cd bytestring-0.9; ./Setup install ...)
  - ghc --make Data.ByteString -this-package-id bytestring-0.9-aaa
    - ghc -c Data/ByteString.hs ''
- (cd attoparsec-0.13; ./Setup install ...)
  - ghc --make Data.Attoparsec Data.Attoparsec.Types -this-package-id attoparsec-0.13-bbb -package-id bytestring-0.9-aaa
    - ghc -c Data/Attoparsec/Types.hs ''
    - ghc -c Data/Attoparsec.hs

Pre-Backpack components

(simplified names)

component attoparsec
  dependency bytestring
  module Data.Attoparsec.Types {...}
  module Data.Attoparsec {...}

opaque!



Principle: Package Manager is source-code independent.

Long live stratification!

<u>Corollary 1</u>: Signatures should be tracked per-package, not per-module.

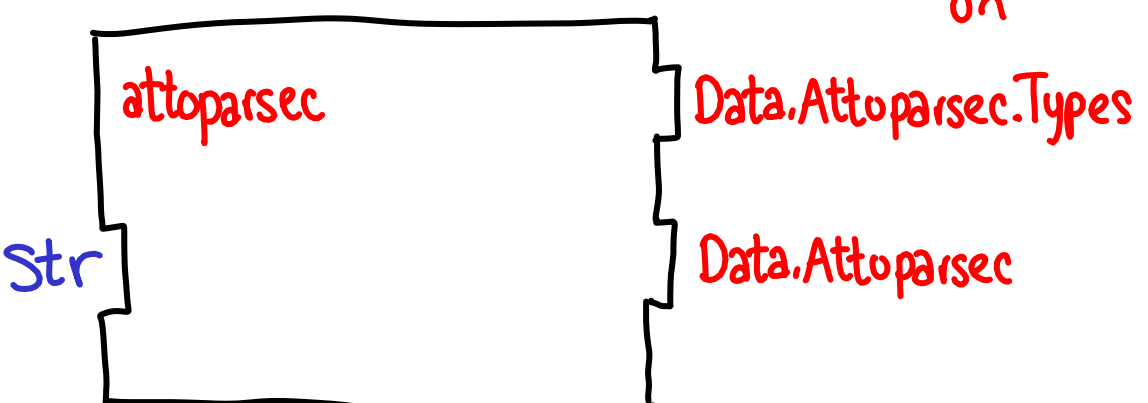
UX: Users don't have to look at import graphs to determine signatures.

Adding signatures

package granularity

component attoparsec-indef <Str>
  signature Str {...}
  module Data.Attoparsec.Types {...}
  module Data.Attoparsec {...}

always depend on



<u>Corollary 2</u>: Mix-in linking can be separated into two algorithms: one language-agnostic, the other language-specific.

Language agnostic mix-in linking

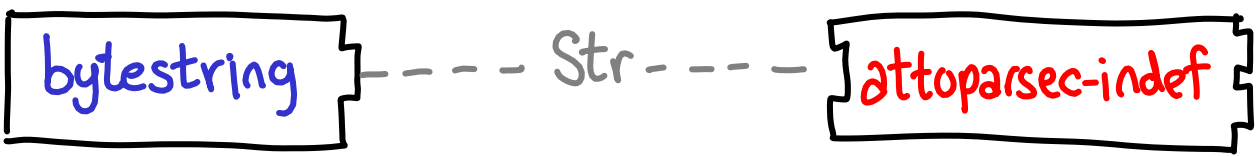
```
package attoparsec-indef
  Str :: [ ... ]
  Data.Attoparsec.Types = [ ... ]
  Data.Attoparsec = [ ...]

package bytestring
  Str = [ ... ]

package attoparsec
  include bytestring
  include attoparsec-indef
```

component attoparsec
  dependency bytestring[]
  dependency
    attoparsec-indef
      [Str = bytestring[]:Str]

functor application!



Very simple: just connect the dots!

Language specific mix-in linking

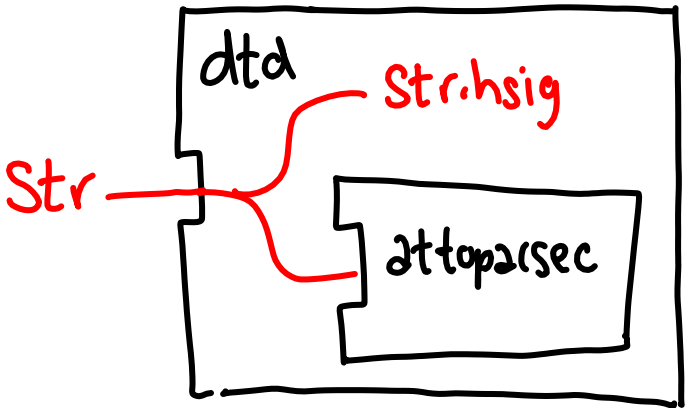
occurs when tc'ing a signature

```
package attoparsec
  Str :: [
    data Str
    concat :: [Str] -> Str
  ]
  Data.Attoparsec = ...

package dtd
  include attoparsec
  Str :: [
    data Str
    length :: Str -> Int
  ]
  ...
```

```
data Str
concat :: [Str] -> Str
length :: Str -> Int
```

merge

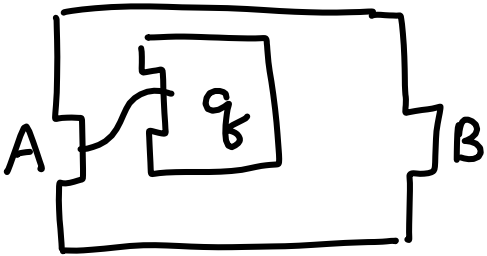


Corollary 3: Compiler only needs to

- ▷ Check if dependency is well-typed
- ▷ Merge signatures
- ▷ Instantiate components (lazily!)

(All per-module operations)

The new abstraction



all required signatures of the component

component r <A>

dependency q[A=<A>]

signature A {...}

module B {...}

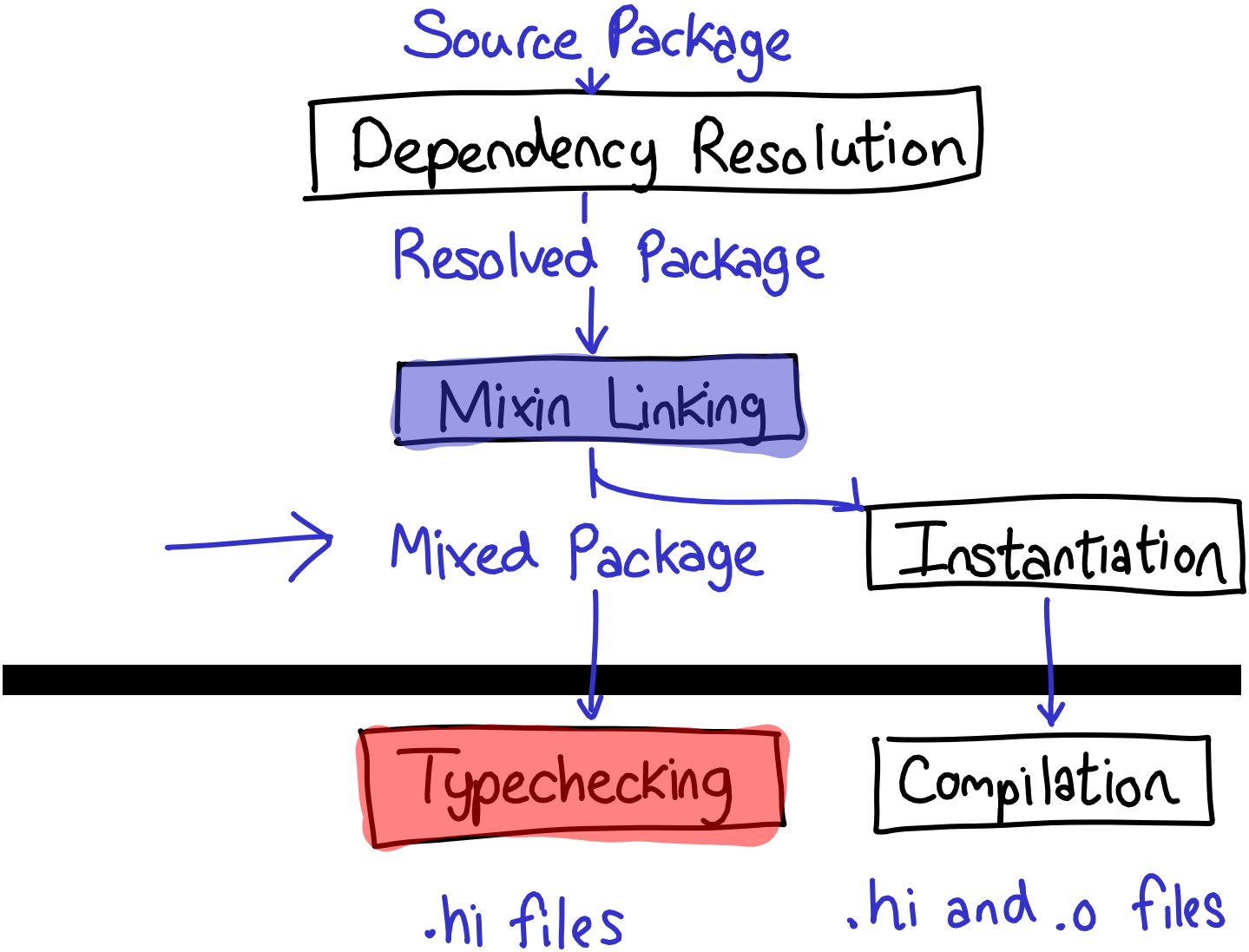
dependency specifies how requirements are instantiated (can just pass requirement through)

signature can add to requirements of component

The new abstraction

cabal install r

- (cd q; ./Setup install ...)
- (cd r; ./Setup install ...)
  - ghc --make A B
    -this-package-id r[A=<A>]
    -package-id q[A=<A>]
    - ghc -c A.hsig " # typechecks A.hsig
      # merges it w/ q[A=<A>]:A
    - checks q[A=<A>] well-t'd
    - ghc -c B.hs " # typechecks B.hs


Source Package
Dependency Resolution
Resolved Package
Mixin Linking
Mixed Package
Instantiation
Typechecking
Compilation
.hi files
.hi and .o files

Evaluation

What has been done

- GHC/Cabal/cabal-install all feature complete
  need code review and testing
- Case studies of varying sizes
  - tagstream-conduit (Str, Parser)
  - ghc-simple (GHC)
  - binary (all build-depends)
  - unix (FilePath, String)

unix

```
signature Str where
import Foreign.C.String (CString)

data Str

useAsOSString :: Str -> (CString -> IO a) -> IO a
newOSString   :: Str -> IO CString
packOSString  :: CString -> IO Str

break :: (Char -> Bool) -> Str -> (Str, Str)
tail :: Str -> Str
head :: Str -> Char
unpack :: Str -> String
append :: Str -> Str -> Str
pack :: String -> Str
```

```haskell
import Str

getEnv :: Str -> IO (Maybe Str)
getEnv name = do
  litstring <- withOSString name c_getenv
  if litstring /= nullPtr
     then liftM Just $ packOSString litstring
     else return Nothing
```

```
library unix-indef
    build-depends:
        base        >= 4.5     && < 4.10,
        time        >= 1.2     && < 1.7
    exposed-modules:
        System.Posix.Env
    required-signatures:
        Str
    -- ...
```

The tagstream-conduit package

[Tags:bsd3, library]

Tag-stream is a library for parsing HTMLXML to a token stream. It can parse unstructured and malformed HTML
provides an Enumeratee which can parse streamline html, which means it consumes constant memory. You can
`tests/Tests.hs` module to see what it can do.

Properties

Versions	0.2.1, 0.2.2, 0.3.0, 0.3.1, 0.3.2, 0.4.0, 0.5.0, 0.5.1, 0.5.2, 0.5.3, 0.5.4, 0.5.4.1, 0.5.5, 0.5.5.1, 0.5.
Dependencies	attoparsec (>=0.10), base (==4.*), blaze-builder, bytestring, case-insensitive, conduit (>=1.2), c data-default (>=0.5.0), resourcet, text, transformers (>=0.2), xml-conduit (>=1.1.0.0) [details]
License	BSD3
Author	yihuang
Maintainer	yi.codeplayer@gmail.com
Stability	Unknown
Category	Web, Conduit
Home page	http://github.com/yihuang/tagstream-conduit
Source repository	head: git clone git://github.com/yihuang/tagstream-conduit
Uploaded	Fri Sep 5 04:04:27 UTC 2014 by YiHuang
Distributions	Arch:0.5.5.3, Debian:0.5.5.3, FreeBSD:0.5.5.3, LTSHaskell:0.5.5.3, NixOS:0.5.5.3, Stackage:0.
Downloads	27513 total (42 in the last 30 days)

Text.HTML.TagStream.Text

Documentation

```
type Token = Token' Text
```

```
type Attr = Attr' Text
```

```
quoted :: Char -> Parser Text
```

```
quotedOr :: Parser Text -> Parser Text
```

```
attrValue :: Parser Text
```

```
attrName :: Parser Text
```

```
tagEnd :: Parser Bool
```

```
attr :: Parser Attr
```

```
attrs :: Parser ([Attr], Bool)
```

```
comment :: Parser Token
```

Text.HTML.TagStream.ByteString

Documentation

```
type Token = Token' ByteString
```

```
type Attr = Attr' ByteString
```

```
quoted :: Char -> Parser ByteString
```

```
quotedOr :: Parser ByteString -> Parser ByteString
```

```
attrValue :: Parser ByteString
```

```
attrName :: Parser ByteString
```

```
tagEnd :: Parser Bool
```

```
attr :: Parser Attr
```

```
attrs :: Parser ([Attr], Bool)
```

```
comment :: Parser Token
```

component

requirements

```
library entities
  signatures: Str, Builder, Parser
  exposed-modules: Entities
  hs-source-dirs: entities
  build-depends:       base >= 4 && < 5
                     , bytestring
                     , text
                     , case-insensitive
                     , transformers >= 0.2
                     , conduit >= 1.2
                     , conduit-extra >= 1.1.0
                     , resourcet
                     , attoparsec >= 0.10
                     , blaze-builder
                     , xml-conduit >= 1.1.0.0
                     , data-default >= 0.5.0
                     , types
```

```
signature Str where

import Data.String

data Str

instance Monoid Str
instance IsString Str
instance Eq Str

drop         :: Int -> Str -> Str
decodeEntity :: Str -> Maybe Str
uncons       :: Str -> Maybe (Char, Str)
cons         :: Char -> Str -> Str
break        :: (Char -> Bool) -> Str -> (Str, Str)
append       :: Str -> Str -> Str
unpack       :: Str -> String
singleton    :: Char -> Str
concat       :: [Str] -> Str
empty        :: Str
null         :: Str -> Bool
```

```
signature Parser where

import Control.Applicative
import Str
```

parameterized over another signature

```
data Parser a
instance Functor Parser
instance Applicative Parser
instance Monad Parser
instance Alternative Parser

anyChar  :: Parser Char
takeTill :: (Char -> Bool) -> Parser Str
char     :: Char -> Parser Char
satisfy  :: (Char -> Bool) -> Parser Char
string   :: Str -> Parser Str
skipSpace :: Parser ()
takeRest :: Parser Str
parseOnly :: Parser a -> Str -> Either String a
```

```haskell
module Entities where

import Prelude hiding (break, drop, concat, null)
-- ...

import Str as T
import Builder
import Parser

type Token = Token' Str
type Attr = Attr' Str

quoted :: Char -> Parser Str
quoted q = append <$> takeTill (in2 ('\\',q))
                      <*> ( char q *> pure ""
                        <|> char '\\' *> atLeast 1 (quoted q)

quotedOr :: Parser Str -> Parser Str
quotedOr p = maybeP (satisfy (in2 ('"','\''))) >>=
               maybe p quoted
```

```haskell
{-# LANGUAGE OverloadedStrings #-}
module Str.ByteString(module Str.ByteString,
                      module Data.ByteString.Char8) where

import           Data.ByteString (ByteString)
import           Data.ByteString.Char8
import           Data.Conduit
import qualified Data.Conduit.List as CL
import           Data.Default
import           Data.Text.Encoding
import qualified Text.XML.Stream.Parse as XML

type Str = ByteString

decodeEntity :: Str -> Maybe Str
decodeEntity entity =
          fmap encodeUtf8
          $ CL.sourceList ["&",entity,";"]
          $= XML.parseBytes def { XML.psDecodeEntities = XM
          $$ XML.content
```

Full code:
https://github.com/yihuang/tagstream-conduit/pull/18

Package manager versus Compiler

- Version-based dependency resolution
- Anti-modular language features
  (type classes, open type families)

These are package manager scale problems.

- Backpack is a mix-in package system
- It's coming to GHC 8.2 (merge soon!)
- Big idea since the POPL'14 paper:

  Factor the design of mix-in packages into the language-agnostic & the language specific

https://github.com/ezyang/ghc-proposals/blob/backpack/proposals/0000-backpack.rst

Backup slide: Why not typeclasses

- Type parameter plumbing / constraints
- Ambiguity
- Performance overhead

  (Even with transitive specialization, you will compile the <u>same code</u> repeatedly)

- No structural subtyping

  (Must commit to interface a priori)

- How will you parametrize existing code?