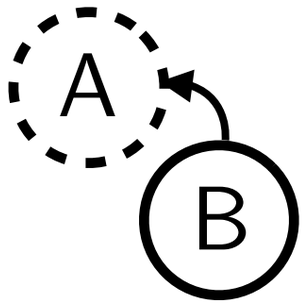
Backpack:
Retrofitting Haskell with Interfaces

Scott Kilpatrick
MPI-SWS

Derek Dreyer MPI-SWS
Simon Peyton Jones Microsoft Research
Simon Marlow Facebook

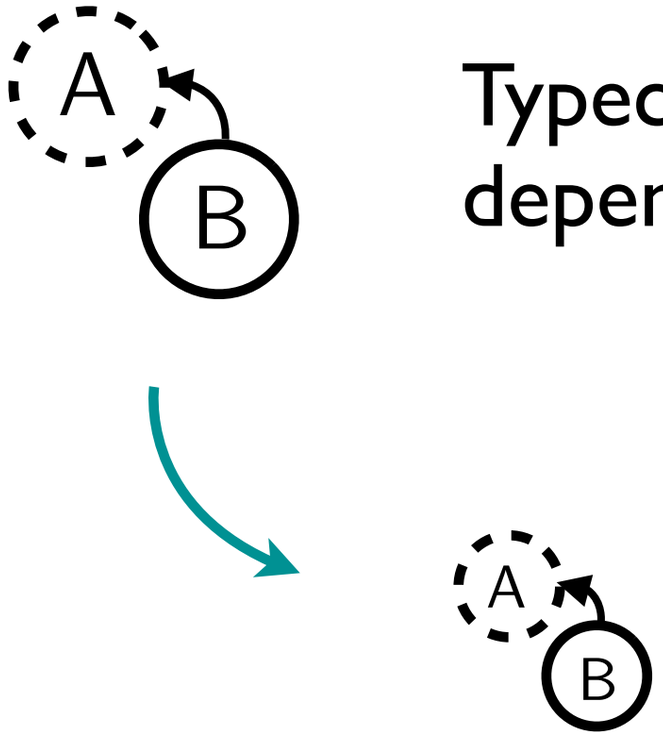
POPL '14
22 January 2014

Strong Modularity



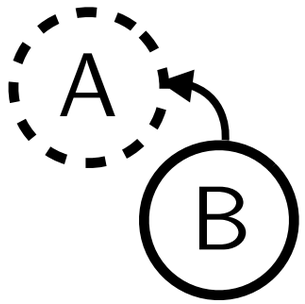
Typecheck B *separately* from A by depending on an interface A.

Strong Modularity

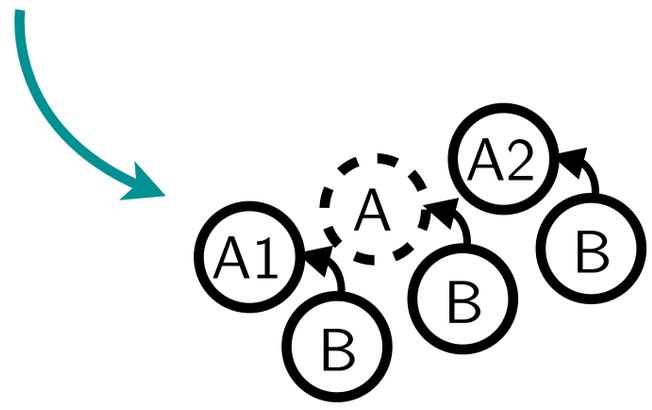


Typecheck B *separately* from A by depending on an interface A.

Strong Modularity

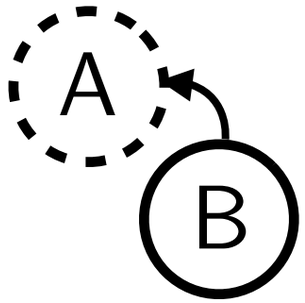


Typecheck B *separately* from A by depending on an interface A.

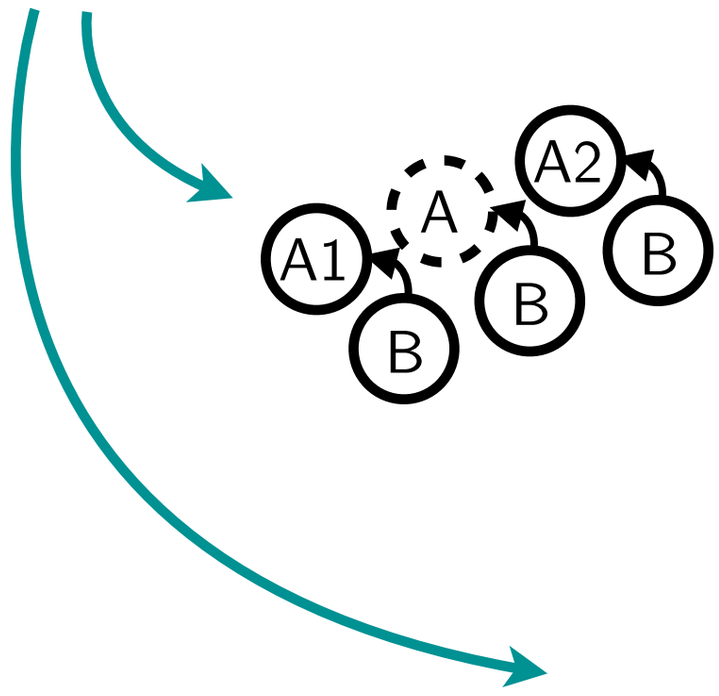


Reuse B by instantiating it with different implementations of A.

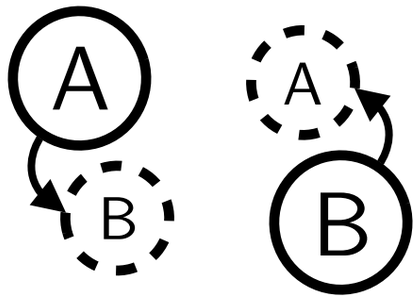
Strong Modularity



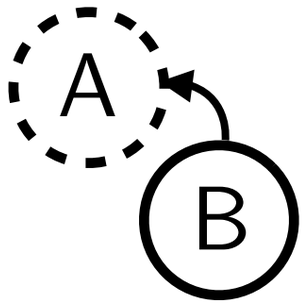
Typecheck B *separately* from A by depending on an interface A.



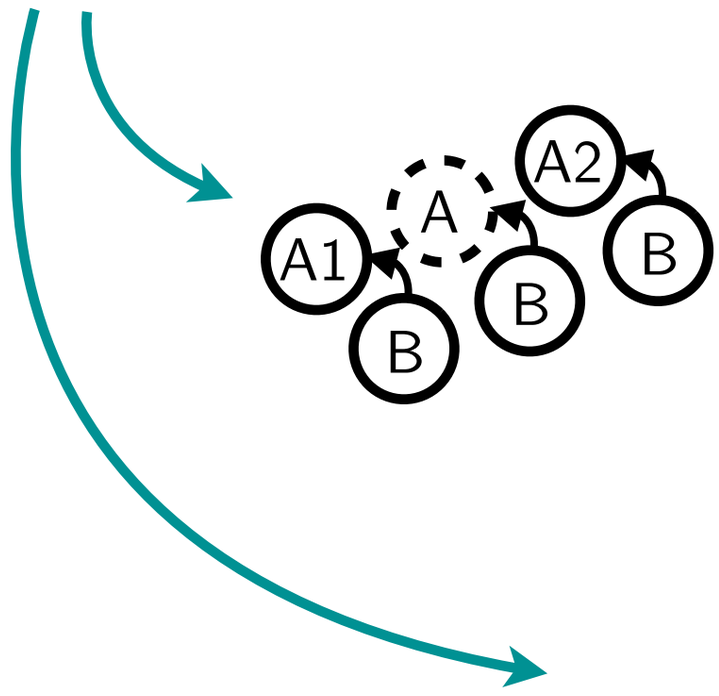
Reuse B by instantiating it with different implementations of A.



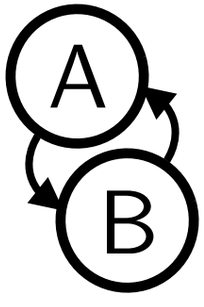
Strong Modularity



Typecheck B *separately* from A by depending on an interface A.



Reuse B by instantiating it with different implementations of A.

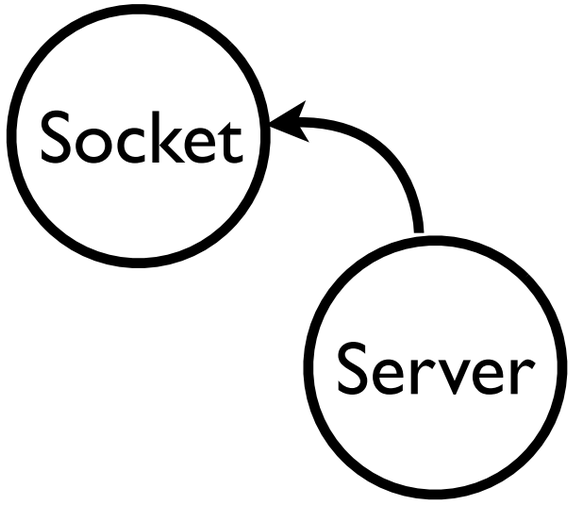


Typecheck mutually dependent modules *before* tying the recursive knot.

Weak Modularity

(implementations depend on implementations)

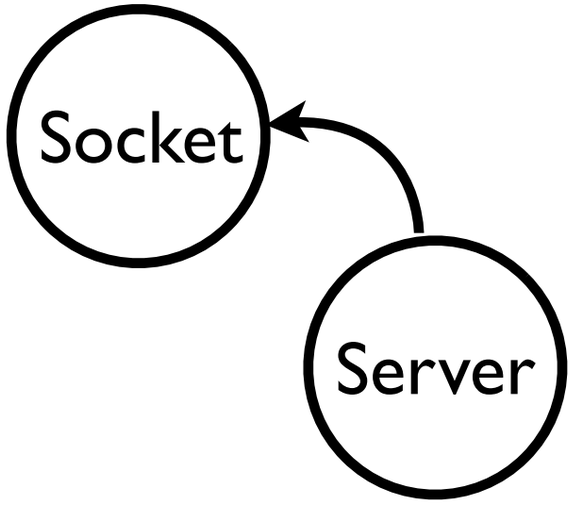
- at Haskell's module level



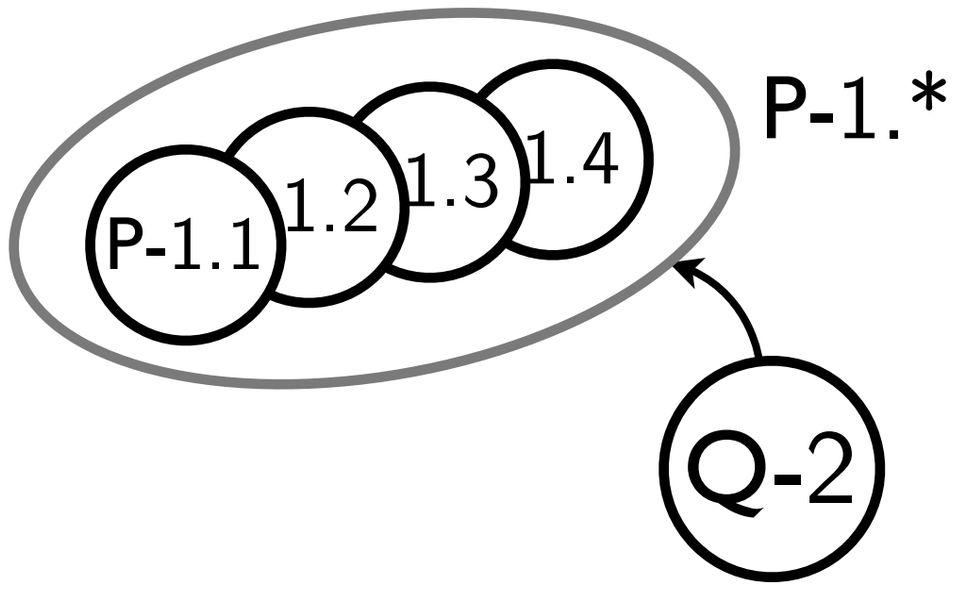
Weak Modularity

(implementations depend on implementations)

- at Haskell's module level



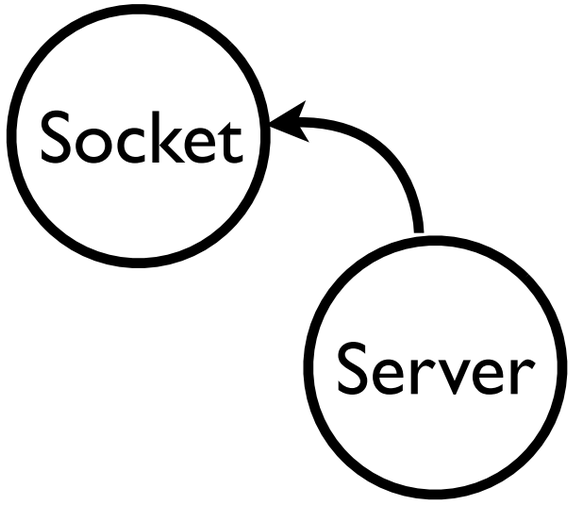
- at Haskell's package level



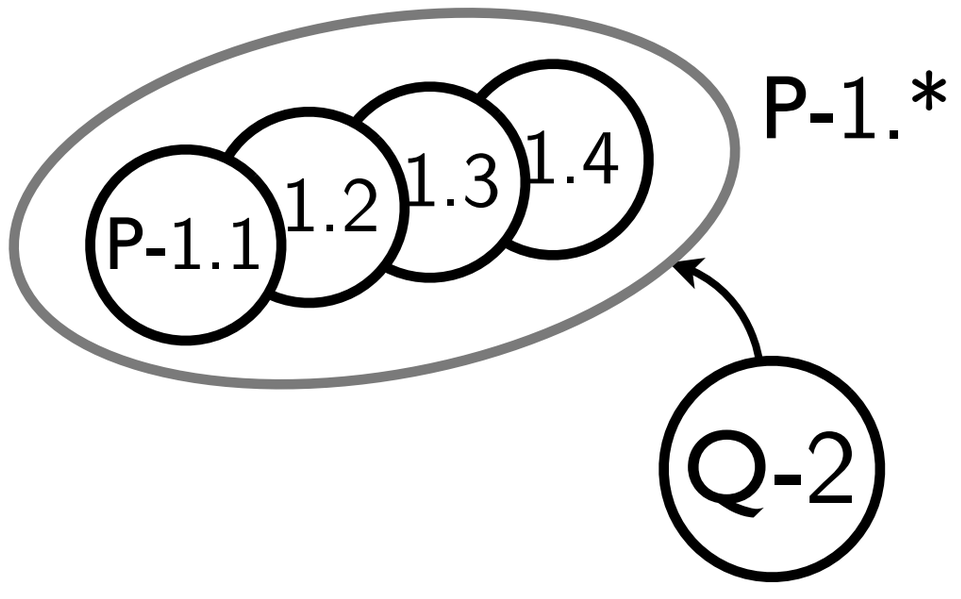
Weak Modularity

(implementations depend on implementations)

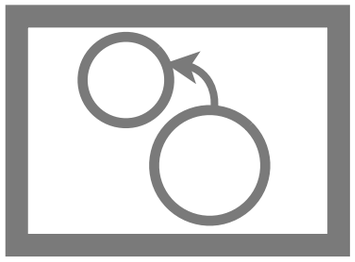
- at Haskell's module level



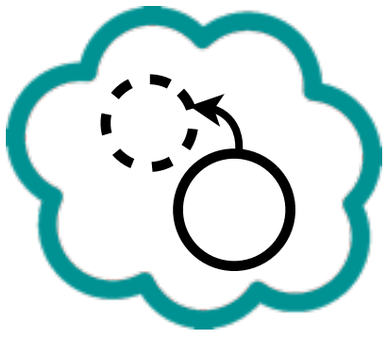
- at Haskell's package level



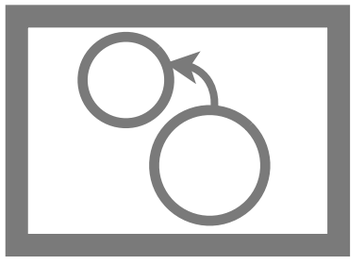
no interfaces ⇒ no strong modularity



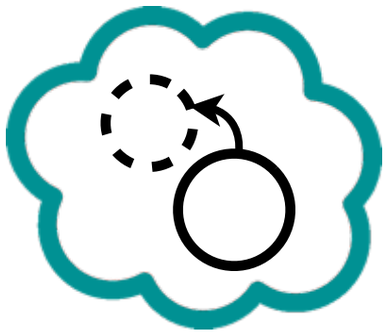
Haskell has
weak modularity.



We want to extend it
with *strong* modularity.

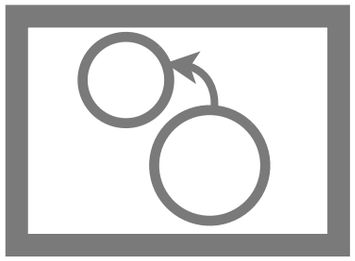


Haskell has
weak modularity.

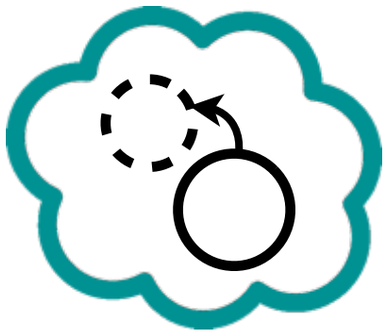


We want to extend it
with *strong* modularity.

Solution:



Haskell has
weak modularity.

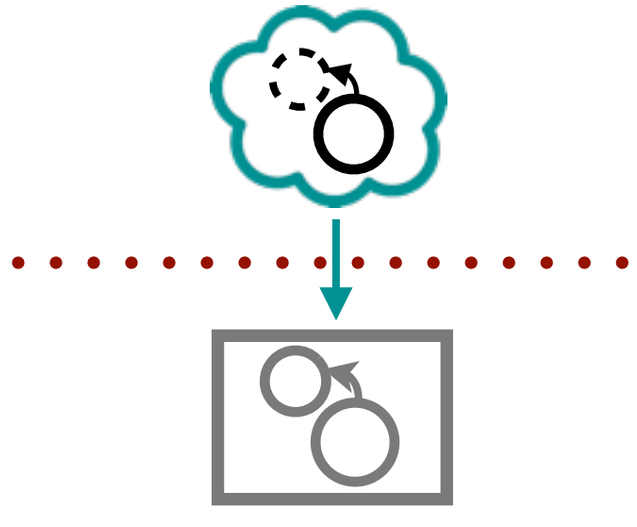


We want to extend it
with *strong* modularity.

Solution:

Retrofit

Design
strong layer
on top of
weak modules!



"Why not just use the ML module system?" (*i*.e. functors)

"Why not just use the ML module system?" (*i*.e. functors)

good point

- a lot of work on the ML module system
- interfaces via signatures, reuse via functors

"Why not just use the ML module system?" (*i*.e. functors)

good point

- a lot of work on the ML module system
- interfaces via signatures, reuse via functors

---

but

"Why not just use the ML module system?" (*i*.e. functors)

good point

- a lot of work on the ML module system
- interfaces via signatures, reuse via functors

but

- functors won't work for us
  - incompatible with recursive linking
  - unclear how to incorporate into weak modules

"Why not just use the ML module system?" (*i*.e. functors)

good point

- a lot of work on the ML module system
- interfaces via signatures, reuse via functors

but

- functors won't work for us
  - incompatible with recursive linking
  - unclear how to incorporate into weak modules
- functors are ill-suited for separate compilation
  - ML langs require additional system on top

Introducing Backpack

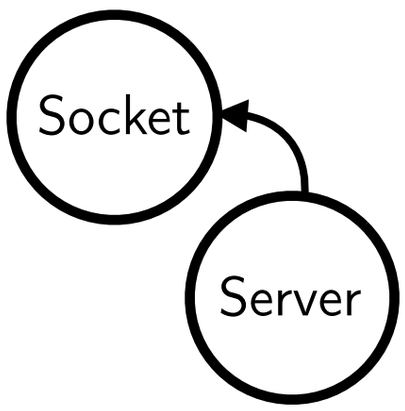
- Retrofits Haskell with strong modularity
  - Designed at *package* level
  - Employs simplified *mixin* design
  - Defined as *elaboration* into weak Haskell modules
  - Separate *typechecking*, not separate compilation

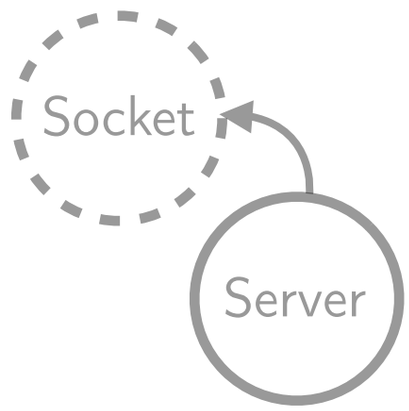
Outline

- Language tour
- Semantics
- Future work

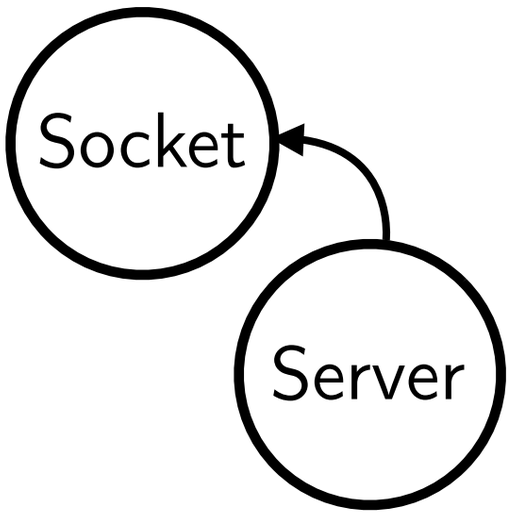
modules in today's Haskell

interfaces





Modules in Today's Haskell



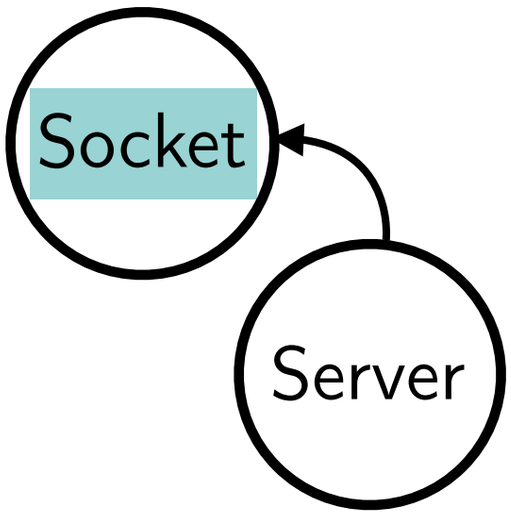
Socket.hs

```
module Socket where
   data SocketT = ...
   open = ...
```

Server.hs

```
module Server where
   import Socket
   data ServerT = ...SocketT...
```

Modules in Today's Haskell



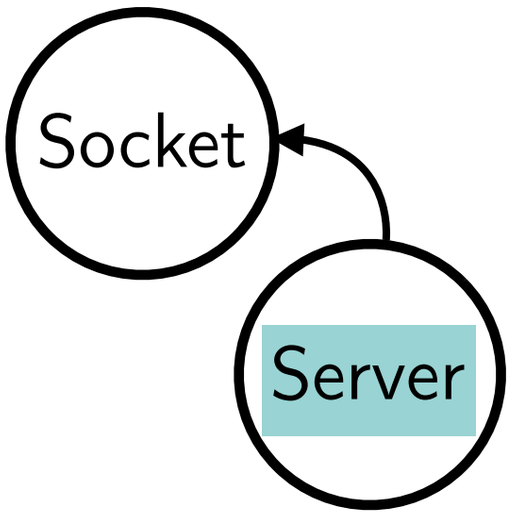
Socket.hs

```
module Socket where
   data SocketT = ...
   open = ...
```

Server.hs

```
module Server where
   import Socket
   data ServerT = ...SocketT...
```

Modules in Today's Haskell



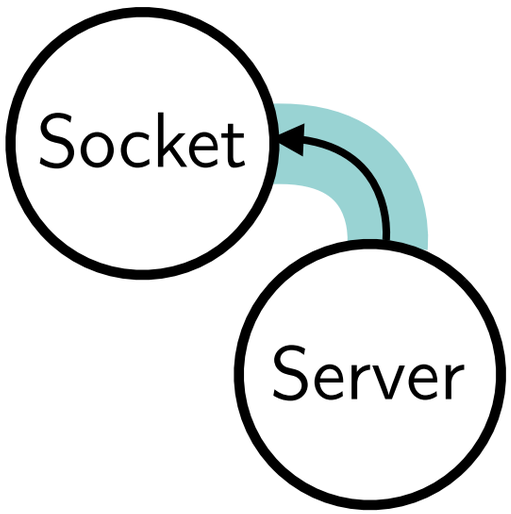
Socket.hs

```
module Socket where
   data SocketT = ...
   open = ...
```

Server.hs

```
module Server where
   import Socket
   data ServerT = ...SocketT...
```

Modules in Today's Haskell



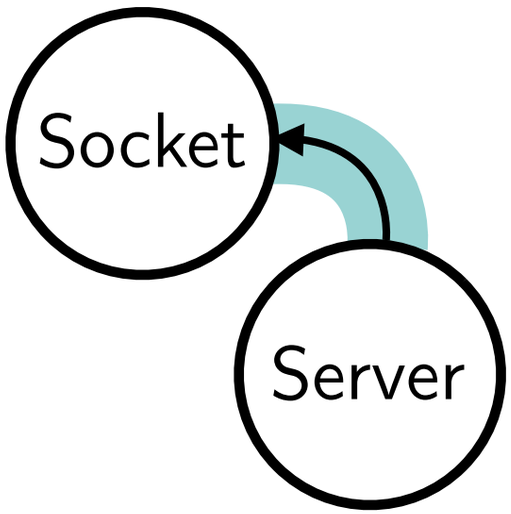
Socket.hs

```
module Socket where
   data SocketT = ...
   open = ...
```

Server.hs

```
module Server where
   import Socket
   data ServerT = ...SocketT...
```

Modules in Today's Haskell



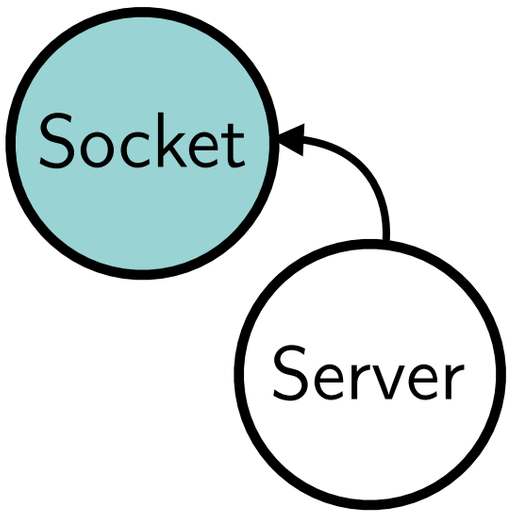
Socket.hs

```
module Socket where
   data SocketT = ...
   open = ...
```

Server.hs

```
module Server where
   import Socket
   data ServerT = ...SocketT...
```

Modules in Today's Haskell



Socket.hs

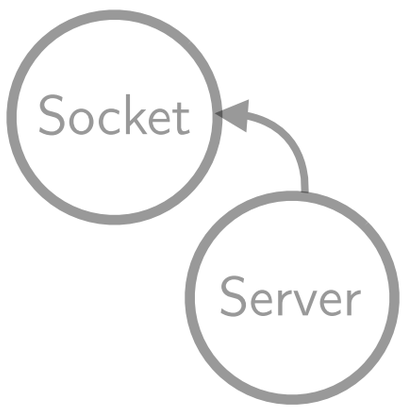
```
module Socket where
   data SocketT = ...
   open = ...
```

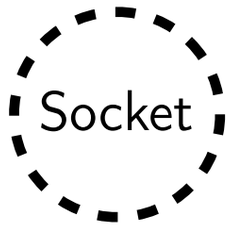
Server.hs

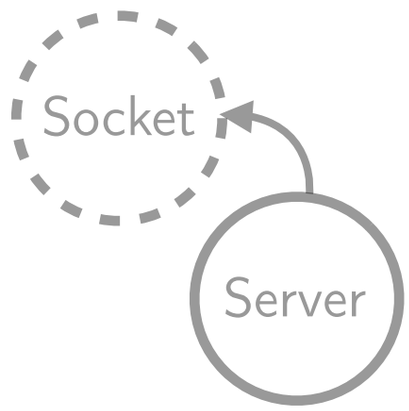
```
module Server where
   import Socket
   data ServerT = ...SocketT...
```

modules in today's Haskell

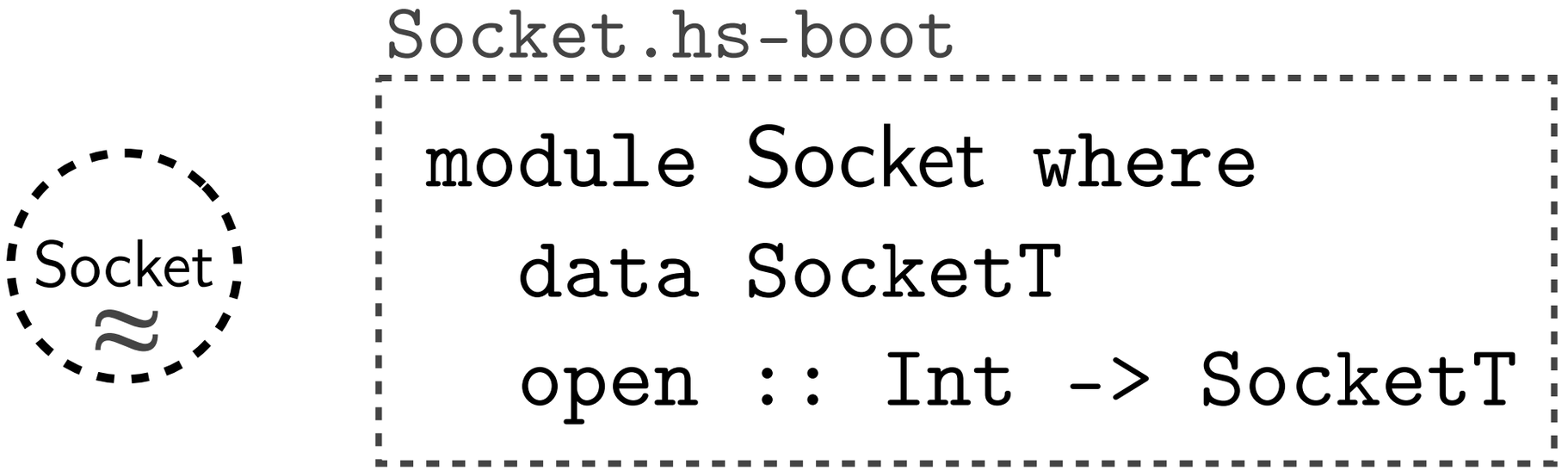
interfaces





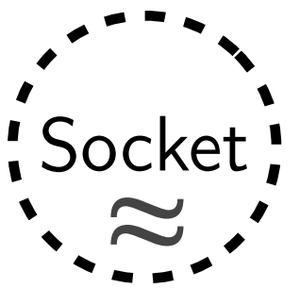


Boot Files: Almost Interfaces



- implementation mechanism in GHC compiler
- used as “forward declaration” for recursive modules

Boot Files: Almost Interfaces

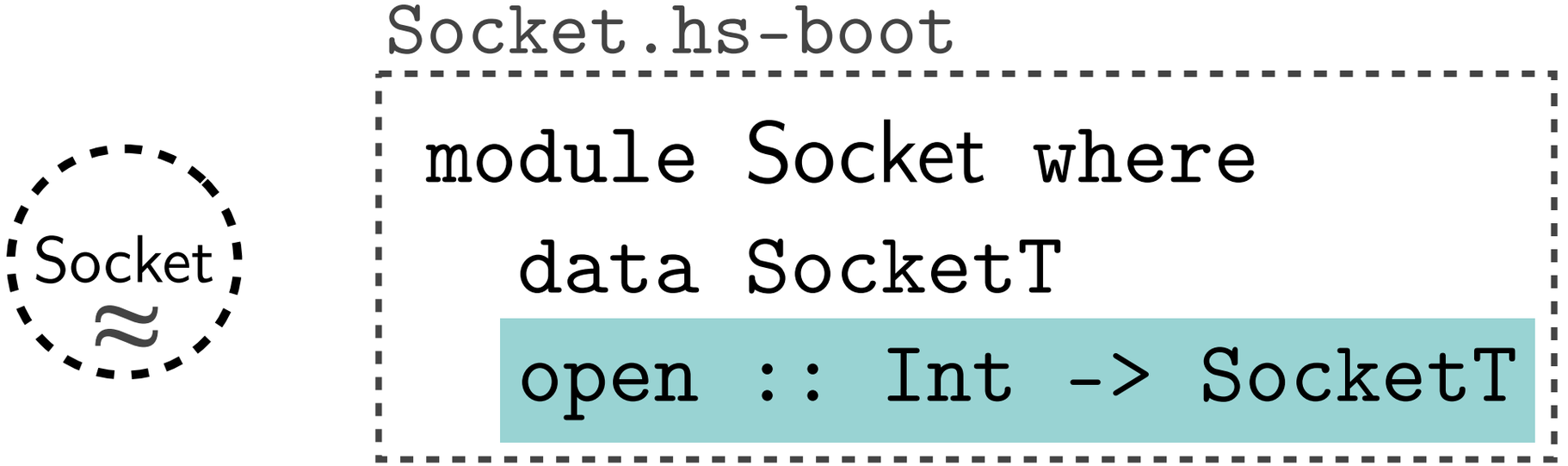


Socket.hs-boot

```
module Socket where
  data SocketT
  open :: Int -> SocketT
```

- implementation mechanism in GHC compiler
- used as “forward declaration” for recursive modules

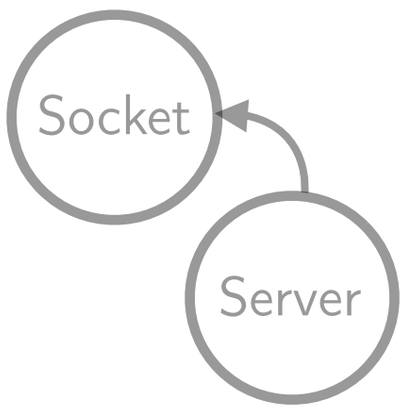
Boot Files: Almost Interfaces

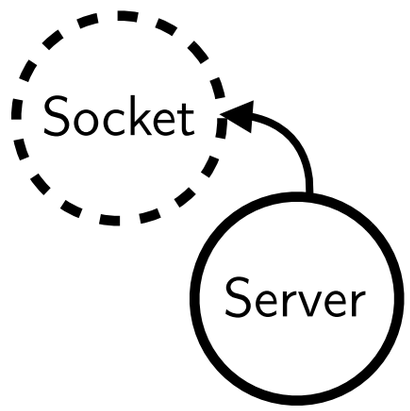


- implementation mechanism in GHC compiler
- used as “forward declaration” for recursive modules

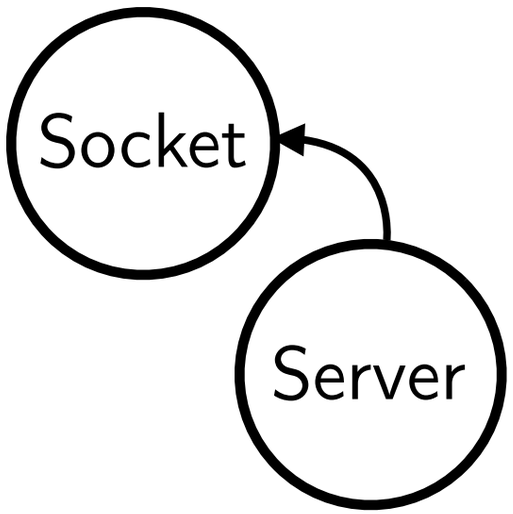
modules in today's Haskell

interfaces





Packages in Backpack



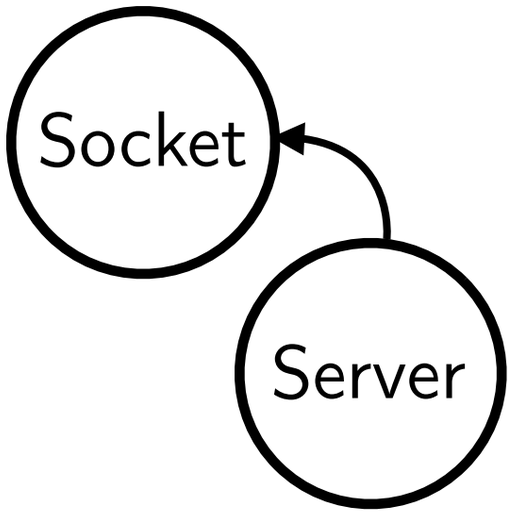
Socket.hs

```
module Socket where
  data SocketT = ...
  open = ...
```

Server.hs

```
module Server where
  import Socket
  data ServerT = ...SocketT...
```

Packages in Backpack

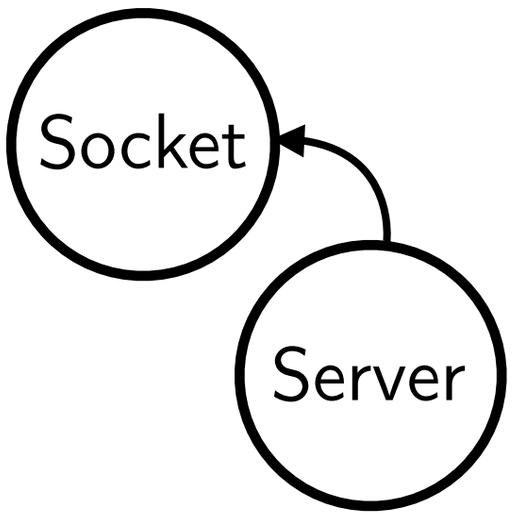


package complete-server **where**

$$\text{Socket} = \begin{bmatrix}\texttt{data SocketT = ...}\\ \texttt{open = ...}\end{bmatrix}$$

$$\text{Server} = \begin{bmatrix}\texttt{import Socket}\\ \texttt{data ServerT = ...SocketT...}\end{bmatrix}$$

Packages in Backpack

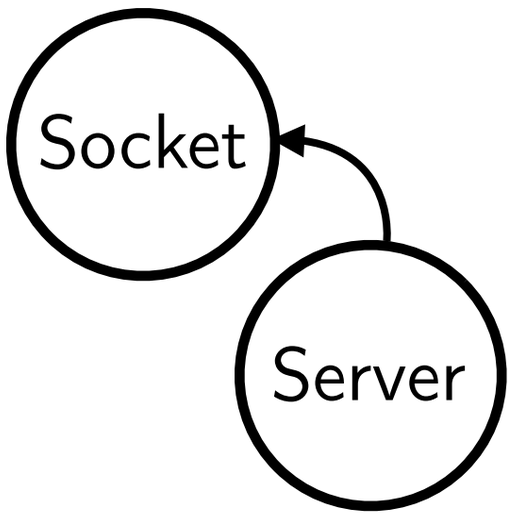


package complete-server **where**

Socket = [data SocketT = ...
open = ...]

Server = [import Socket
data ServerT = ...SocketT...]

Packages in Backpack

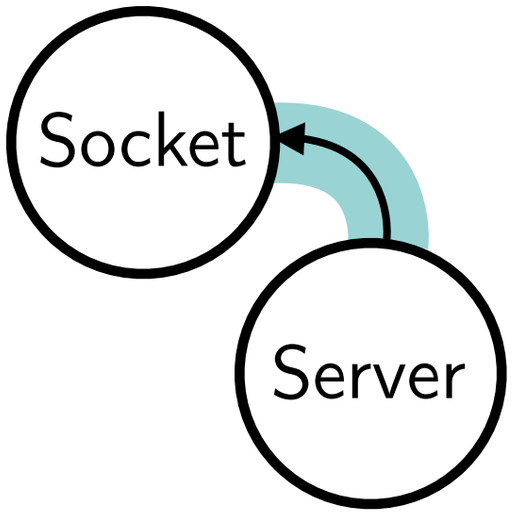


package complete-server **where**

$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$

$$\text{Server} = \begin{bmatrix} \texttt{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Packages in Backpack

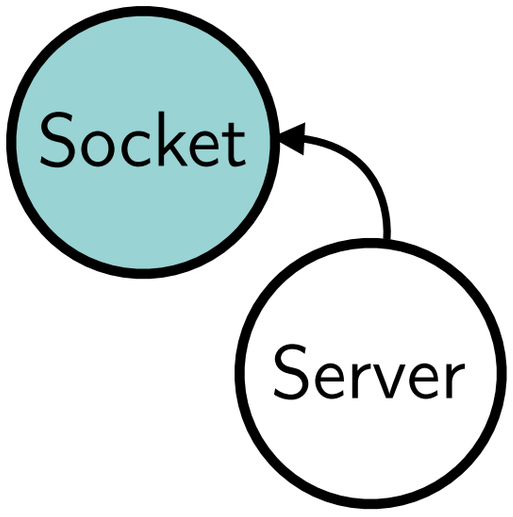


package complete-server **where**

$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$

$$\text{Server} = \begin{bmatrix} \texttt{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Packages in Backpack

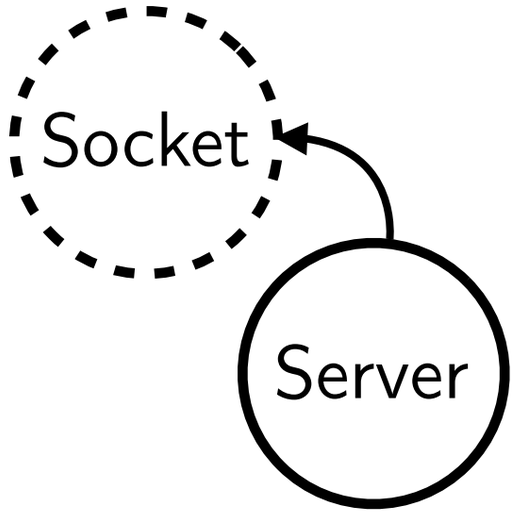


package complete-server **where**

$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$

$$\text{Server} = \begin{bmatrix} \texttt{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Packages in Backpack
(With Holes)

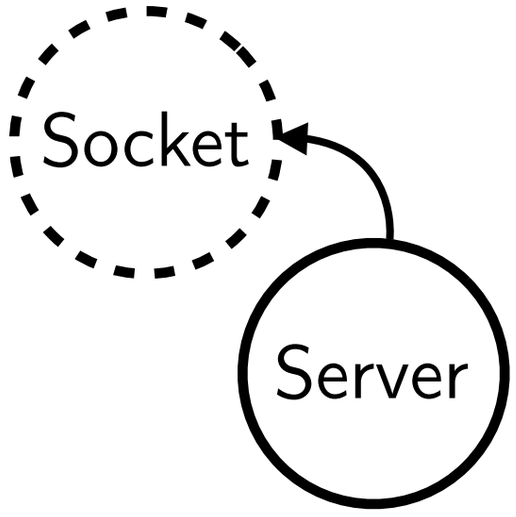


package partial-server **where**

$$\text{Socket} :: \begin{bmatrix} \texttt{data SocketT} \\ \texttt{open :: Int -> SocketT} \end{bmatrix}$$

$$\text{Server} = \begin{bmatrix} \text{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Packages in Backpack
(With Holes)

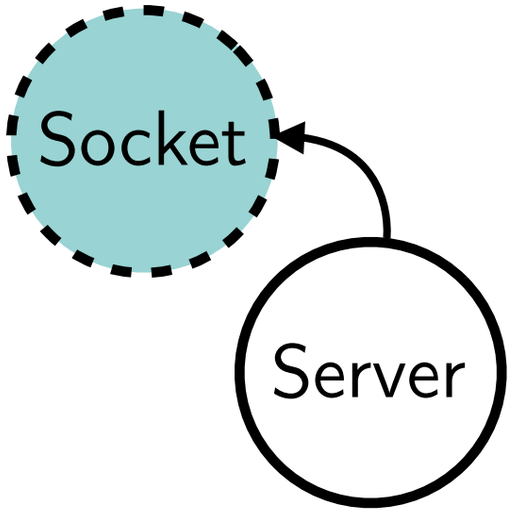


package partial-server **where**

Socket :: [data SocketT
open :: Int -> SocketT]

Server = [import Socket
data ServerT = ...SocketT...]

Packages in Backpack
(With Holes)

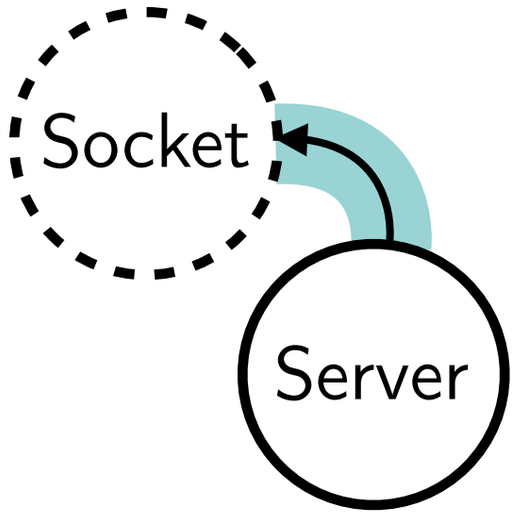


package partial-server **where**

Socket :: [data SocketT
open :: Int -> SocketT]

Server = [import Socket
data ServerT = ...SocketT...]

Packages in Backpack
(With Holes)



package partial-server **where**

Socket :: [data SocketT
open :: Int -> SocketT]

Server = [import Socket
data ServerT = ...SocketT...]

Package Inclusion

package complete-server **where**

$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$

$$\text{Server} = \begin{bmatrix} \texttt{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Package Inclusion

package socketimpl **where**

$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$

package complete-server **where**

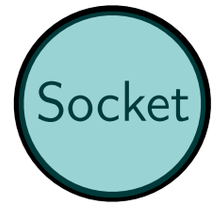
include socketimpl

$$\text{Server} = \begin{bmatrix} \text{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Package Inclusion

package socketimpl **where**

$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$



package complete-server **where**

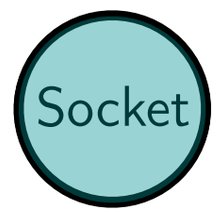
include socketimpl

$$\text{Server} = \begin{bmatrix} \text{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Package Inclusion

package socketimpl **where**

$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$



package complete-server **where**

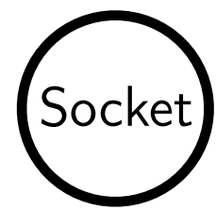
include socketimpl

$$\text{Server} = \begin{bmatrix} \text{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Package Inclusion

package socketimpl **where**

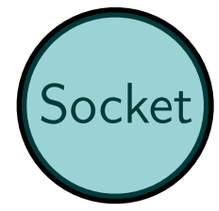
$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$



package complete-server **where**

include socketimpl

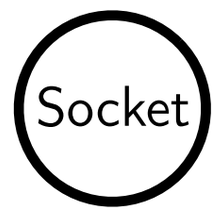
$$\text{Server} = \begin{bmatrix} \text{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$



Package Inclusion

package socketimpl **where**

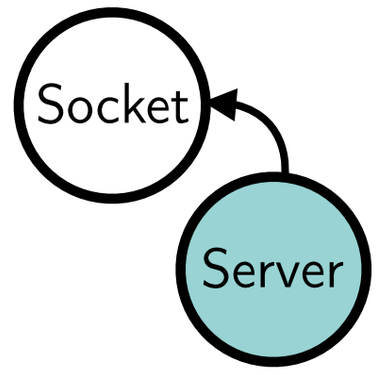
$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$



package complete-server **where**

include socketimpl

$$\text{Server} = \begin{bmatrix} \text{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$



Package Inclusion

package socketimpl **where**

$$\text{Socket} = \begin{bmatrix} \texttt{data SocketT = ...} \\ \texttt{open = ...} \end{bmatrix}$$

package complete-server **where**

include socketimpl

$$\text{Server} = \begin{bmatrix} \text{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Package Inclusion

package socketsig **where**

$$\text{Socket} :: \begin{bmatrix} \texttt{data SocketT} \\ \texttt{open :: Int -> SocketT} \end{bmatrix}$$

package partial-server **where**

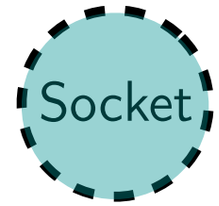
include socketsig

$$\text{Server} = \begin{bmatrix} \text{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$

Package Inclusion

package socketsig **where**

Socket :: [data SocketT
open :: Int -> SocketT]



package partial-server **where**

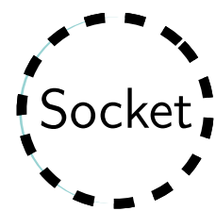
include socketsig

Server = [import Socket
data ServerT = ...SocketT...]

Package Inclusion

package socketsig **where**

Socket :: [data SocketT
open :: Int -> SocketT]



package partial-server **where**

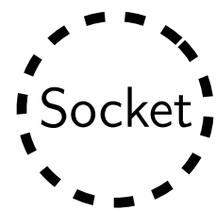
include socketsig

Server = [import Socket
data ServerT = ...SocketT...]

Package Inclusion

package socketsig **where**

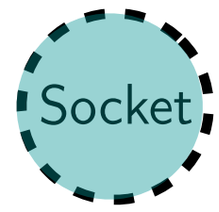
Socket :: [data SocketT
open :: Int -> SocketT]



package partial-server **where**

include socketsig

Server = [import Socket
data ServerT = ...SocketT...]



Package Inclusion

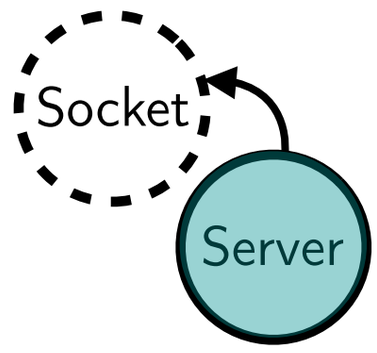
package socketsig **where**

$$\text{Socket} :: \begin{bmatrix} \texttt{data SocketT} \\ \texttt{open :: Int -> SocketT} \end{bmatrix}$$

package partial-server **where**

include socketsig

$$\text{Server} = \begin{bmatrix} \text{import Socket} \\ \texttt{data ServerT = ...SocketT...} \end{bmatrix}$$



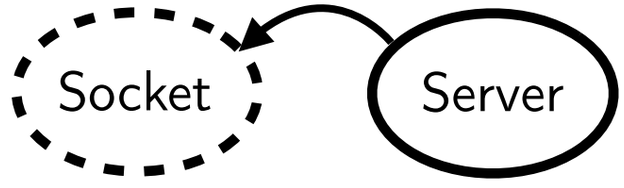
Linking

package socketsig **where**

Socket :: [`data SocketT`
`open :: Int -> SocketT`]

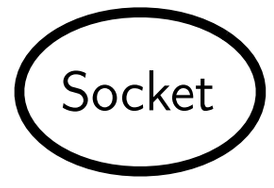
package partial-server **where**
include socketsig

Server = [import Socket
`data ServerT = ...SocketT...`]



package socketimpl **where**

Socket = [`data SocketT = ...`
`open =` $\lambda n.$ `...`]



package main **where**
include partial-server
include socketimpl

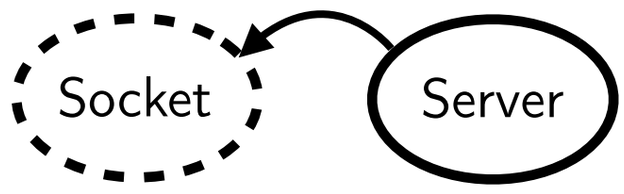
Linking

package socketsig **where**

Socket :: [`data SocketT`, `open :: Int -> SocketT`]

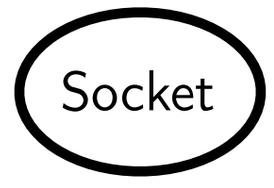
package partial-server **where**
include socketsig

Server = [import Socket, `data ServerT = ...SocketT...`]



package socketimpl **where**

Socket = [`data SocketT = ...`, `open = λn. ...`]



package main **where**
include partial-server
include socketimpl

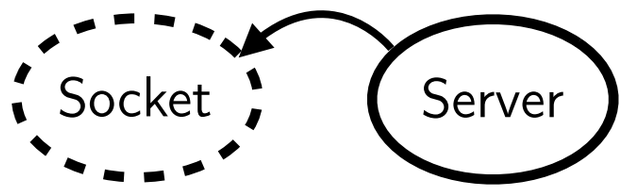
Linking

package socketsig **where**

Socket :: [`data SocketT`, `open :: Int -> SocketT`]

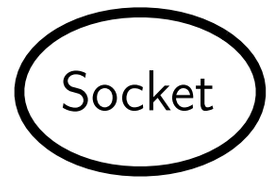
package partial-server **where**
include socketsig

Server = [import Socket, `data ServerT = ...SocketT...`]

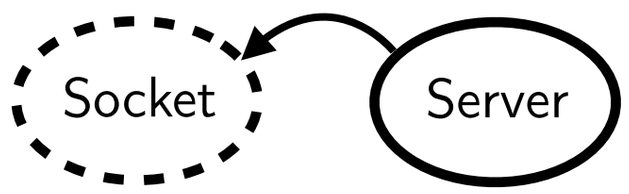


package socketimpl **where**

Socket = [`data SocketT = ...`, `open =` $\lambda n.$ `...`]



package main **where**
include partial-server
include socketimpl



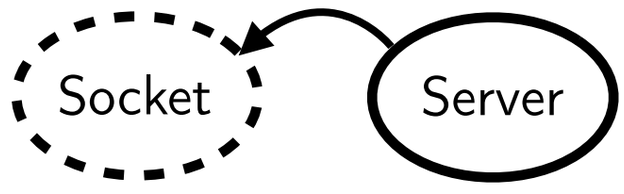
Linking

package socketsig **where**

Socket :: [data SocketT
open :: Int -> SocketT]

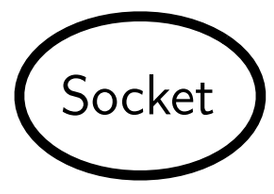
package partial-server **where**
include socketsig

Server = [import Socket
data ServerT = ...SocketT...]

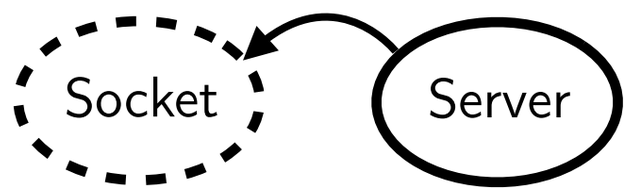


package socketimpl **where**

Socket = [data SocketT = ...
open = $\lambda n.$ ...]



package main **where**
include partial-server
include socketimpl



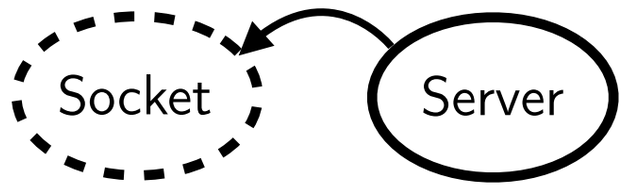
Linking

package socketsig **where**

Socket :: $\begin{bmatrix}\texttt{data SocketT} \\ \texttt{open :: Int -> SocketT}\end{bmatrix}$

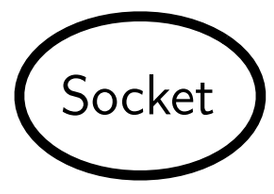
package partial-server **where**
include socketsig

Server = $\begin{bmatrix}\text{import Socket} \\ \texttt{data ServerT = ...SocketT...}\end{bmatrix}$

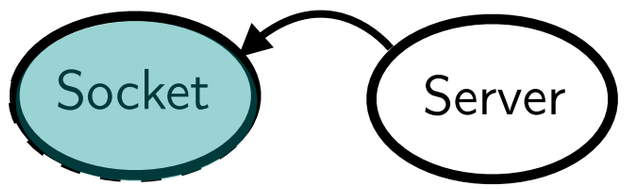


package socketimpl **where**

Socket = $\begin{bmatrix}\texttt{data SocketT = ...} \\ \texttt{open} = \lambda n.\ \ldots\end{bmatrix}$



package main **where**
include partial-server
include socketimpl



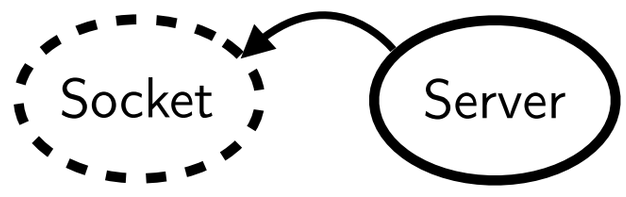
Linking

package socketsig **where**

$$\text{Socket} :: \begin{bmatrix}\texttt{data SocketT} \\ \texttt{open :: Int -> SocketT}\end{bmatrix}$$

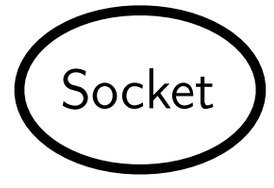
package partial-server **where**
include socketsig

$$\text{Server} = \begin{bmatrix}\text{import Socket} \\ \texttt{data ServerT = ...SocketT...}\end{bmatrix}$$

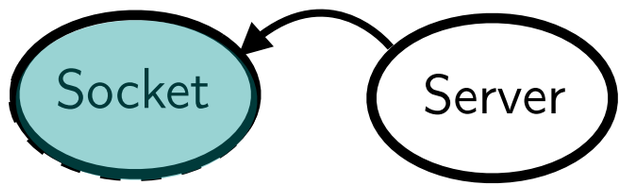


package socketimpl **where**

$$\text{Socket} = \begin{bmatrix}\texttt{data SocketT = ...} \\ \texttt{open = } \lambda n.\ \ldots\end{bmatrix}$$



package main **where**
include partial-server
include socketimpl



$$\begin{bmatrix}\texttt{data SocketT = ...} \\ \texttt{open = } \lambda n.\ \ldots\end{bmatrix} \leq \begin{bmatrix}\texttt{data SocketT} \\ \texttt{open :: Int -> SocketT}\end{bmatrix}$$

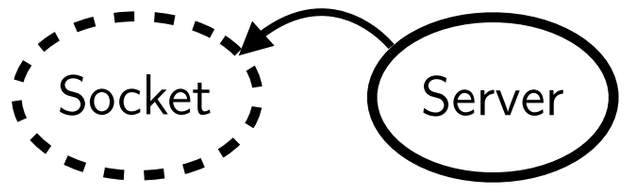
Linking

package socketsig **where**

Socket :: $\begin{bmatrix}\texttt{data SocketT} \\ \texttt{open :: Int -> SocketT}\end{bmatrix}$

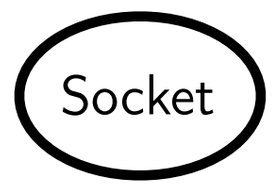
package partial-server **where**
include socketsig

Server = $\begin{bmatrix}\text{import Socket} \\ \texttt{data ServerT = ...SocketT...}\end{bmatrix}$

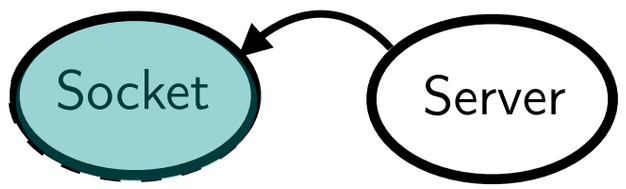


package socketimpl **where**

Socket = $\begin{bmatrix}\texttt{data SocketT = ...} \\ \texttt{open} = \lambda n.\ \ldots\end{bmatrix}$



package main **where**
include partial-server
include socketimpl



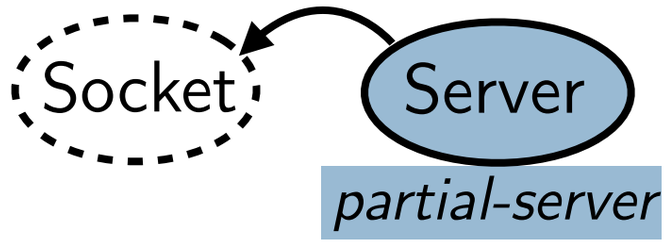
Reuse

package server-linked-1 **where**
  include partial-server
  include socketimpl-1

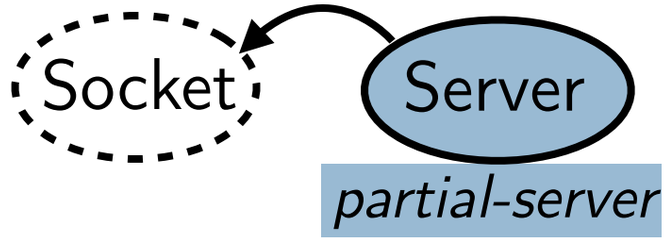
package server-linked-2 **where**
include partial-server
include socketimpl-2

Reuse

package server-linked-1 **where**
  include partial-server
  include socketimpl-1

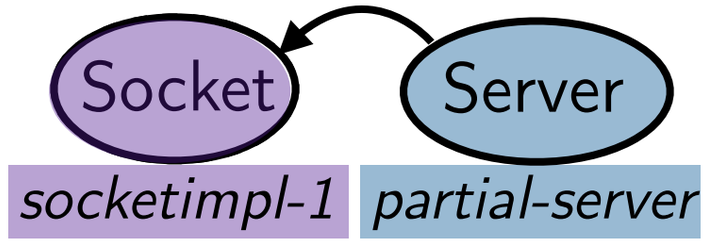


package server-linked-2 **where**
  include partial-server
  include socketimpl-2

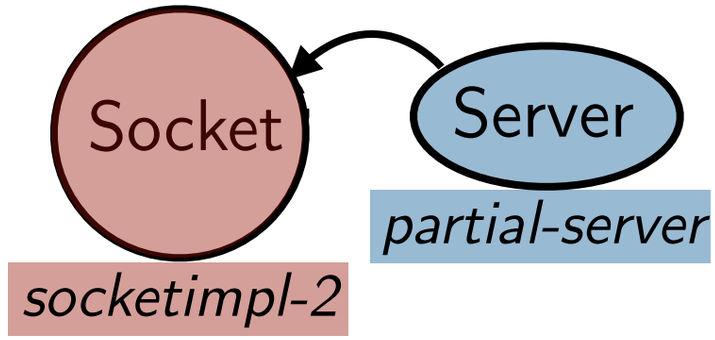


Reuse

package server-linked-1 **where**
  include partial-server
  include socketimpl-1

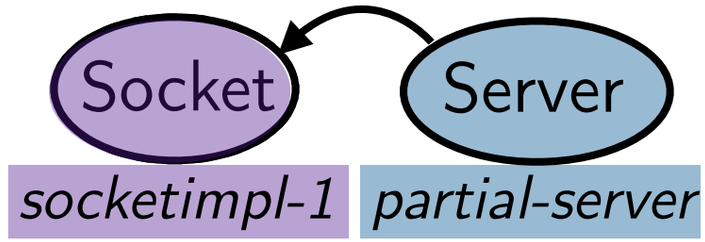


package server-linked-2 **where**
  include partial-server
  include socketimpl-2

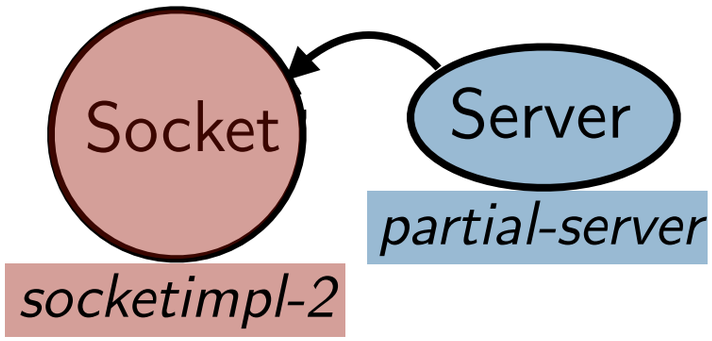


Reuse

package server-linked-1 **where**
  include partial-server
  include socketimpl-1



package server-linked-2 **where**
  include partial-server
  include socketimpl-2

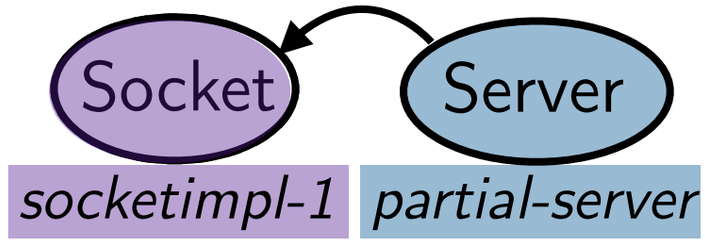


package multi **where**

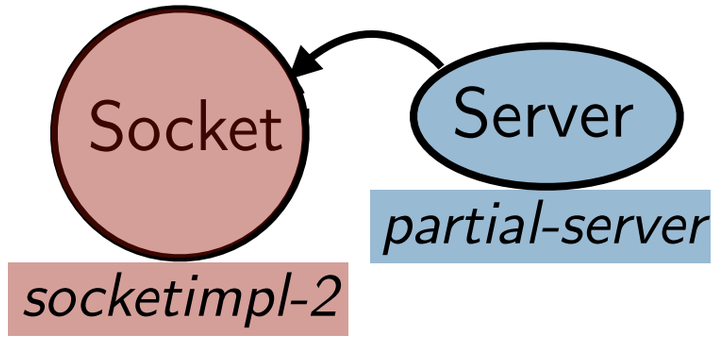
$$\begin{array}{lcl} \text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\ \text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\ \text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix} \end{array}$$

Reuse

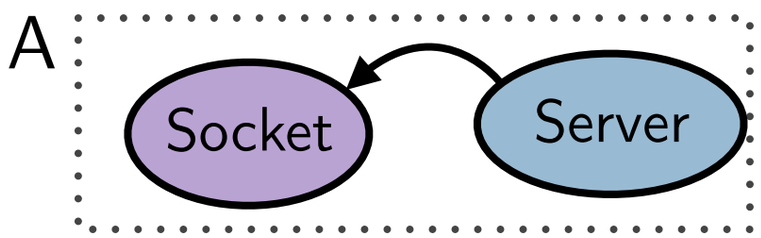
package server-linked-1 **where**
  include partial-server
  include socketimpl-1



package server-linked-2 **where**
  include partial-server
  include socketimpl-2

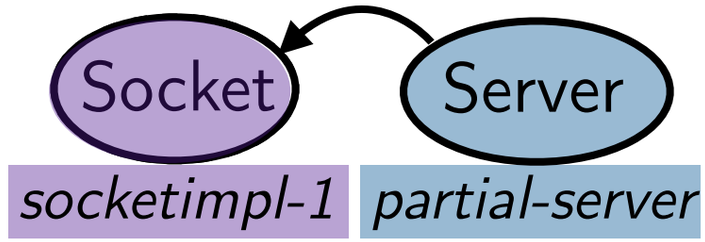


package multi **where**
  A = {**include** server-linked-1}
  B = {**include** server-linked-2}
  Main = [import qualified A.Server
  import qualified B.Server
  ...]

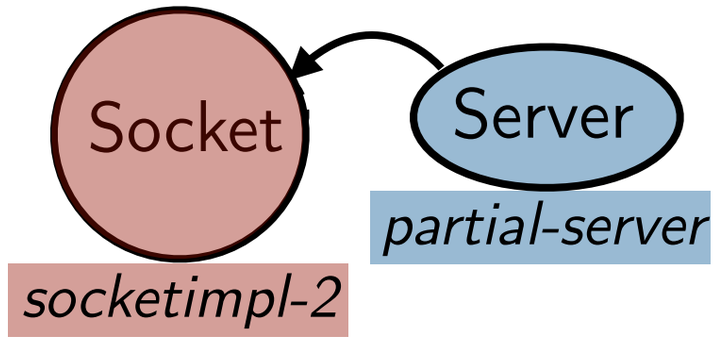


Reuse

package server-linked-1 **where**
  include partial-server
  include socketimpl-1

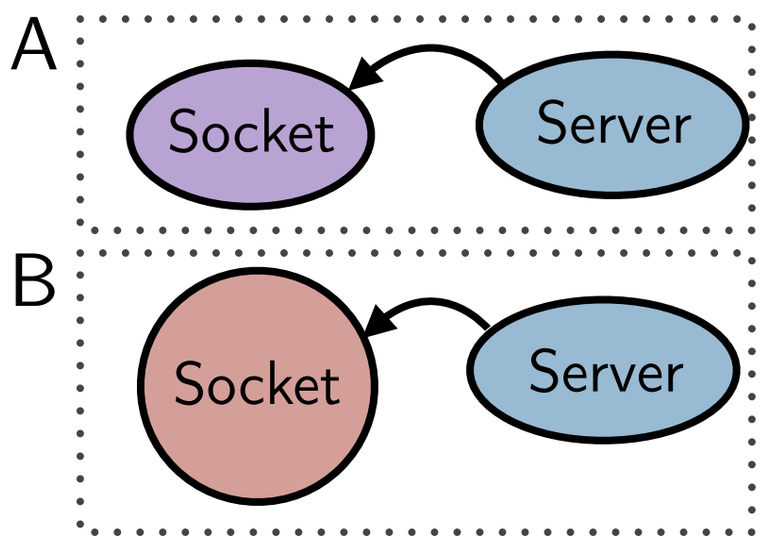


package server-linked-2 **where**
  include partial-server
  include socketimpl-2



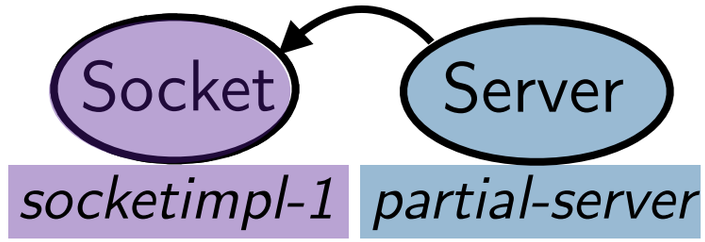
package multi **where**

$$\begin{aligned} A &= \{\textbf{include}\ \text{server-linked-1}\} \\ B &= \{\textbf{include}\ \text{server-linked-2}\} \\ \text{Main} &= \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix} \end{aligned}$$

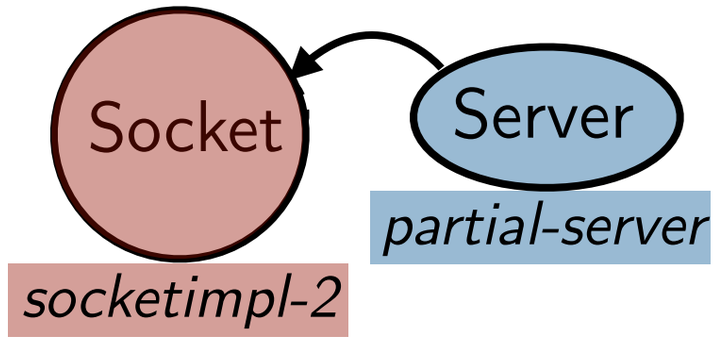


Reuse

package server-linked-1 **where**
  include partial-server
  include socketimpl-1

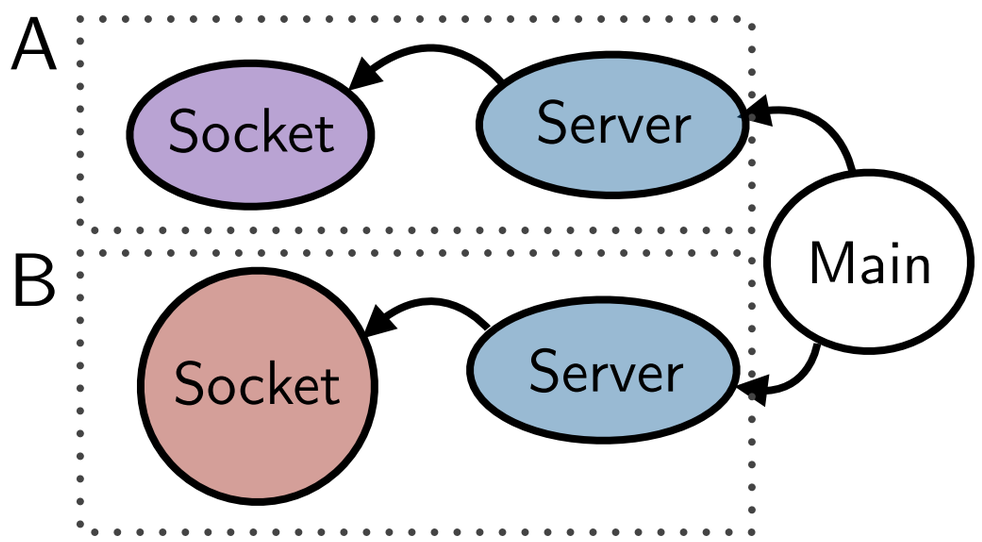


package server-linked-2 **where**
  include partial-server
  include socketimpl-2



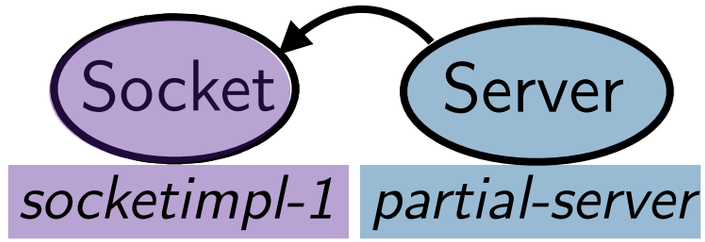
package multi **where**

$$\begin{aligned} A &= \{\textbf{include}\ \text{server-linked-1}\} \\ B &= \{\textbf{include}\ \text{server-linked-2}\} \\ \text{Main} &= \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix} \end{aligned}$$

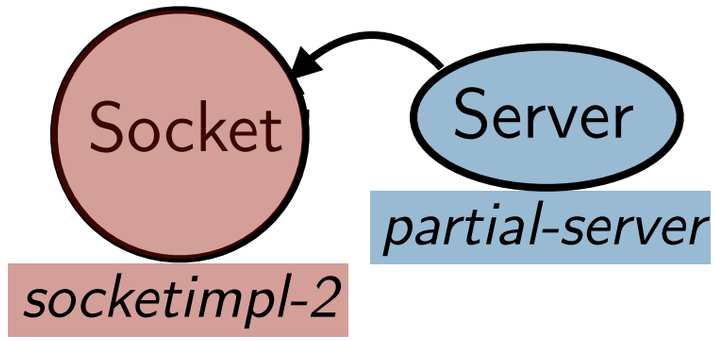


Reuse

package server-linked-1 **where**
  include partial-server
  include socketimpl-1

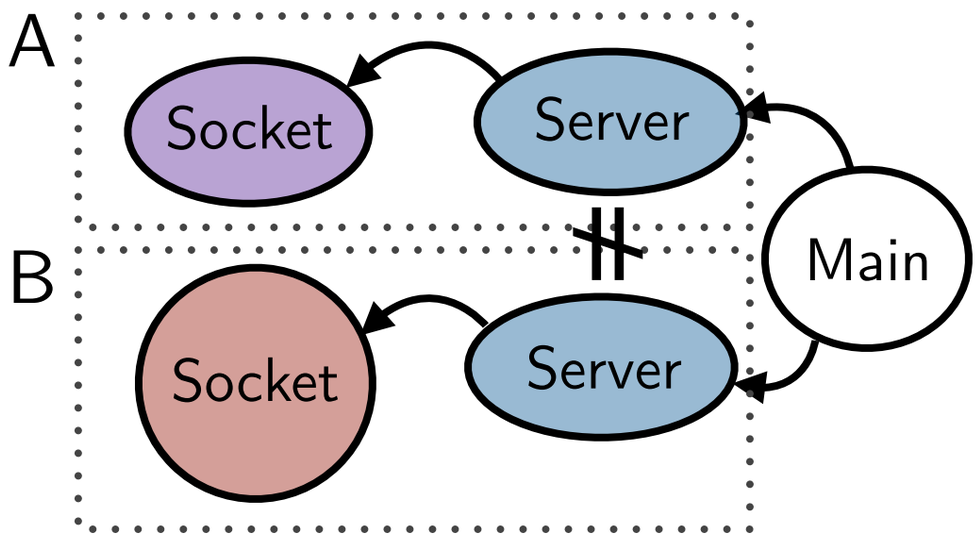


package server-linked-2 **where**
  include partial-server
  include socketimpl-2



package multi **where**

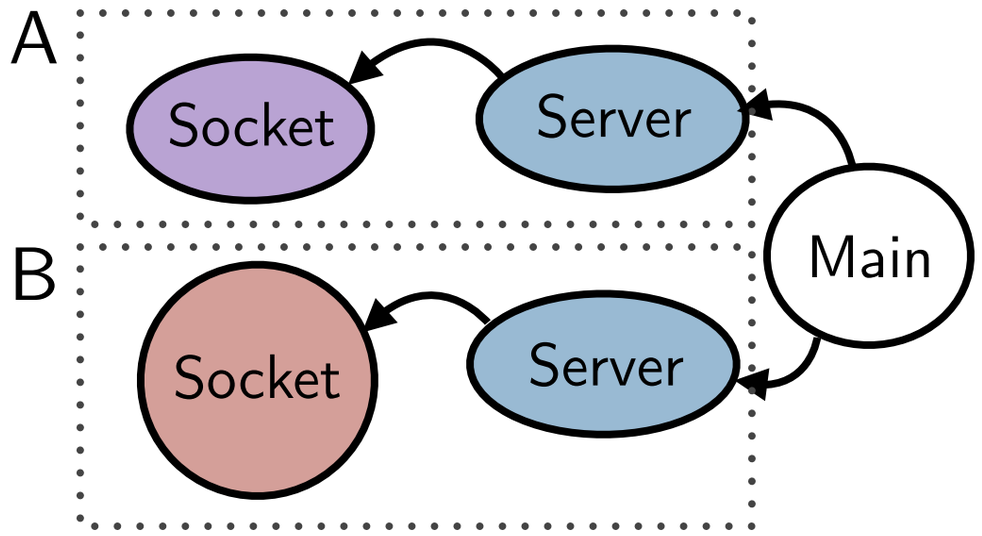
$$\begin{aligned} A &= \{\textbf{include}\ \text{server-linked-1}\} \\ B &= \{\textbf{include}\ \text{server-linked-2}\} \\ \text{Main} &= \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix} \end{aligned}$$



Shared Reuse

package multi **where**

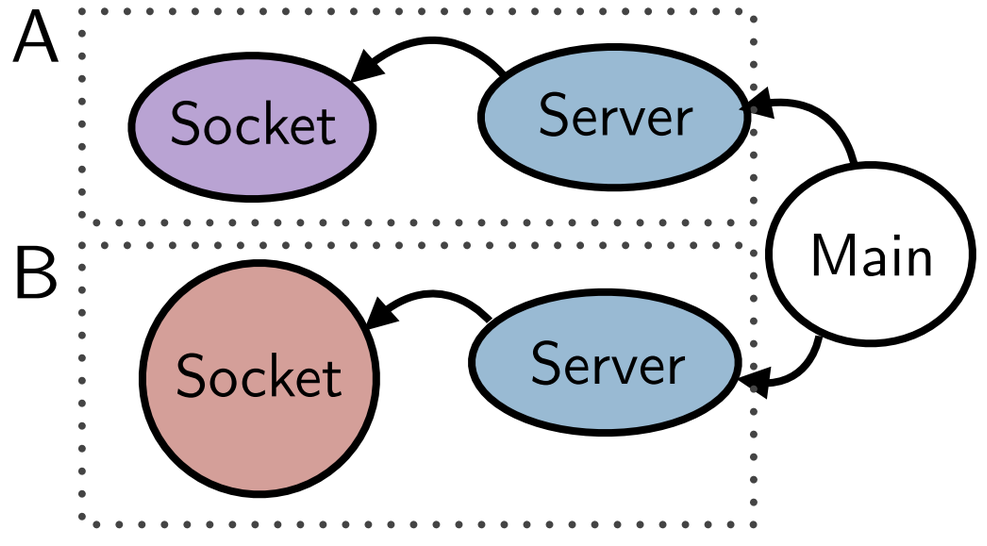
$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



Shared Reuse

package multi **where**

$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



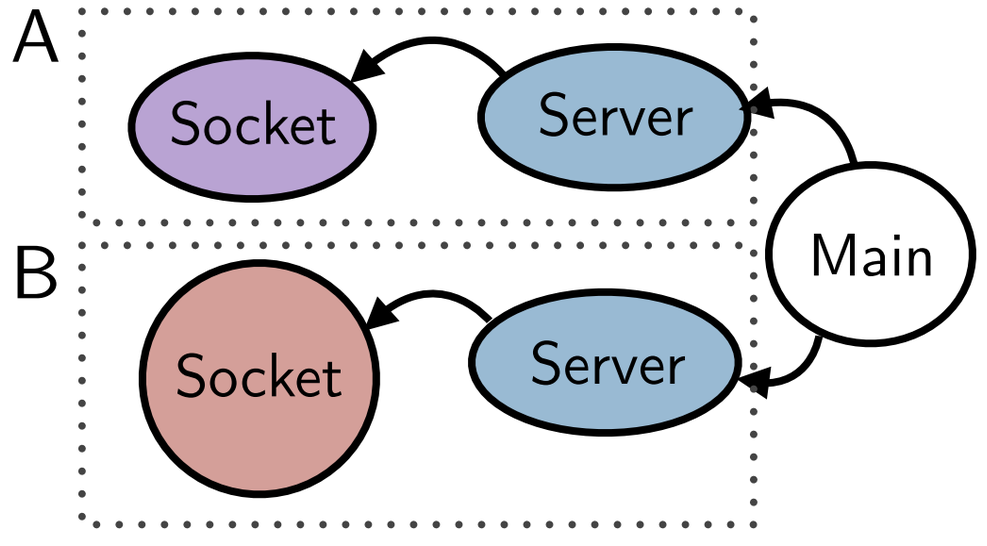
package multi-shared **where**

$$
\begin{array}{lcl}
\text{C} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots \end{bmatrix}
\end{array}
$$

Shared Reuse

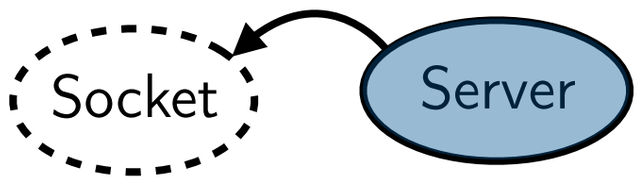
package multi **where**

$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



package multi-shared **where**

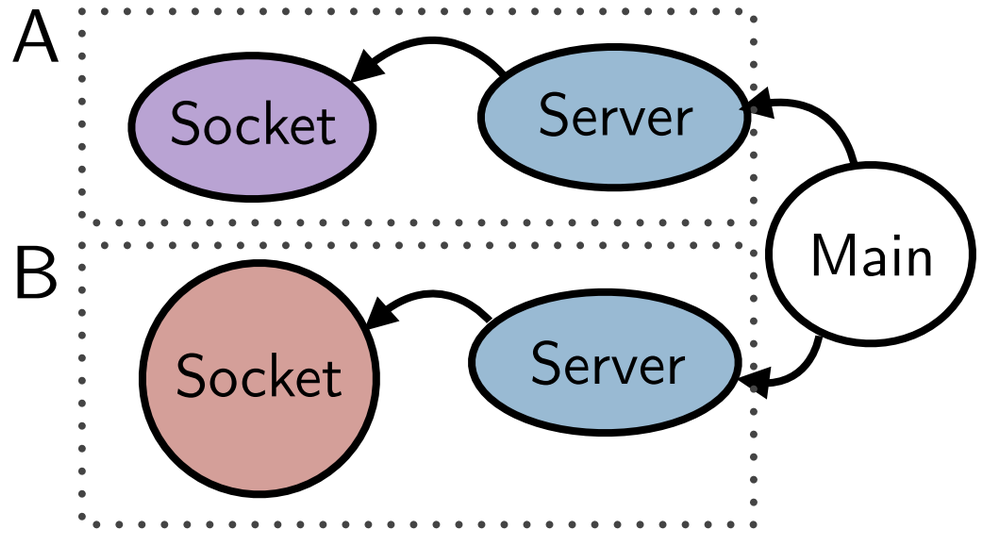
$$
\begin{array}{lcl}
\text{C} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



Shared Reuse

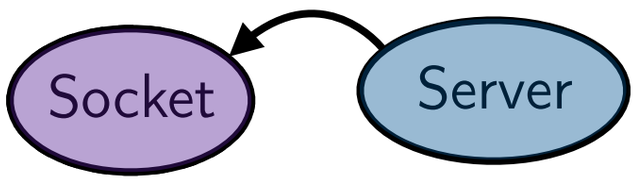
package multi **where**

$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



package multi-shared **where**

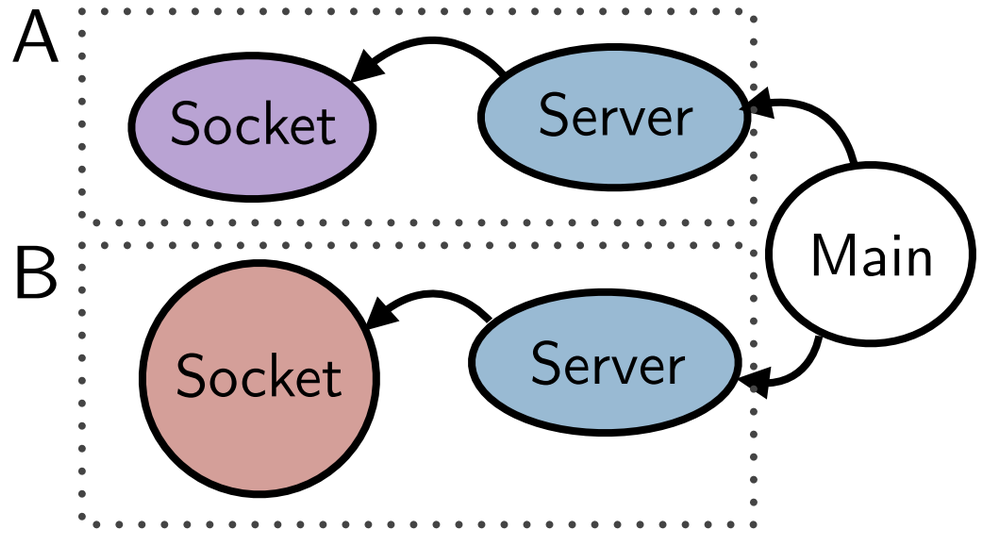
$$
\begin{array}{lcl}
\text{C} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



Shared Reuse

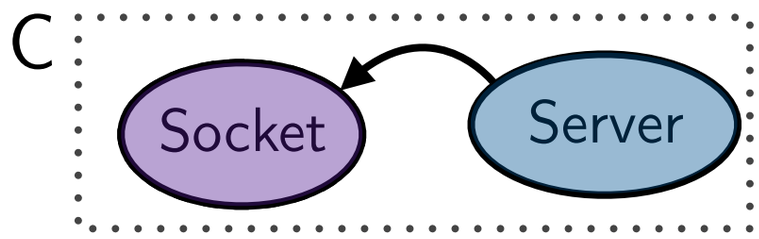
package multi **where**

$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



package multi-shared **where**

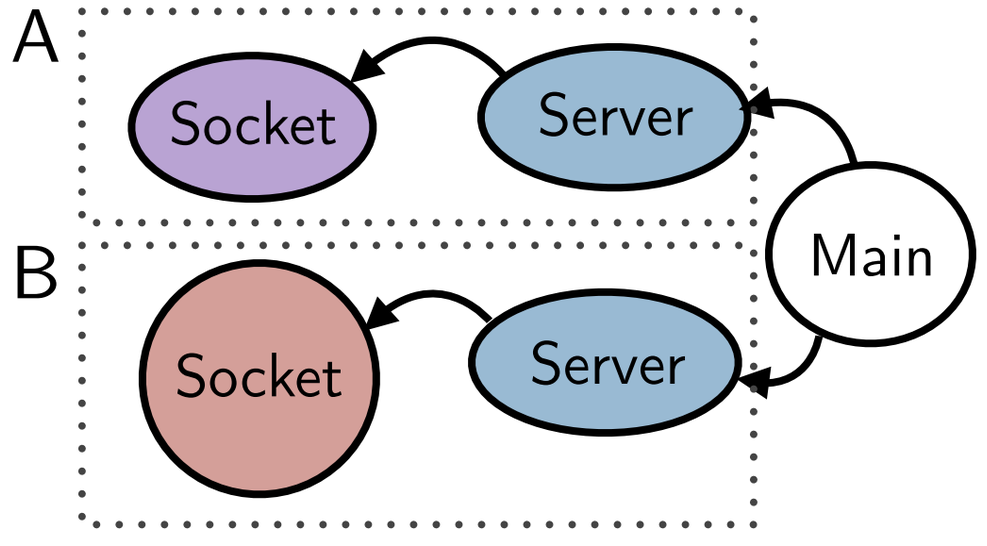
$$
\begin{array}{lcl}
\text{C} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



Shared Reuse

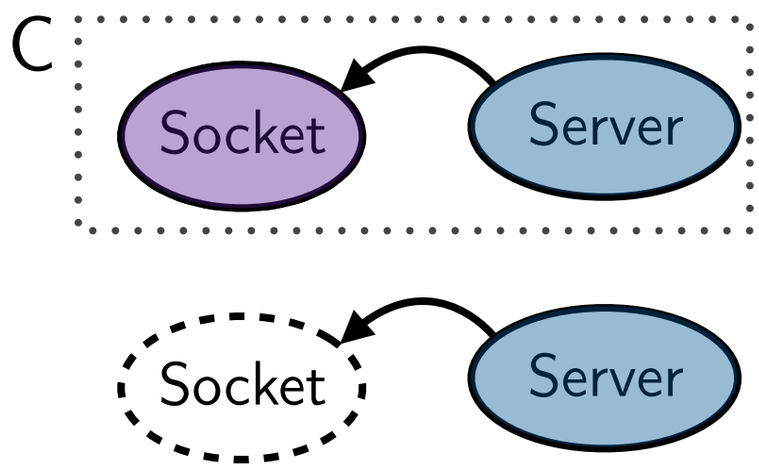
package multi **where**

$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix}\text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots\end{bmatrix}
\end{array}
$$



package multi-shared **where**

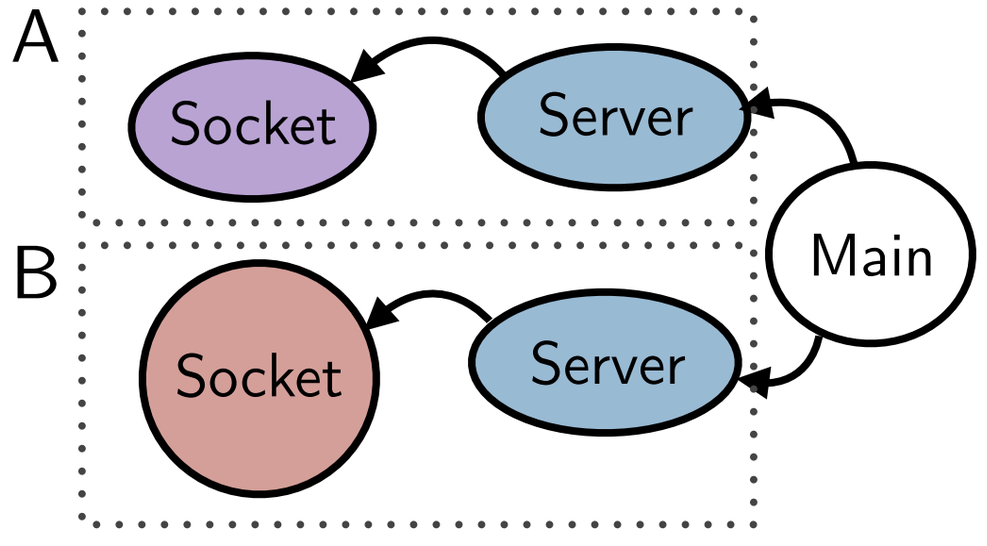
$$
\begin{array}{lcl}
\text{C} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} & = & \begin{bmatrix}\text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots\end{bmatrix}
\end{array}
$$



Shared Reuse

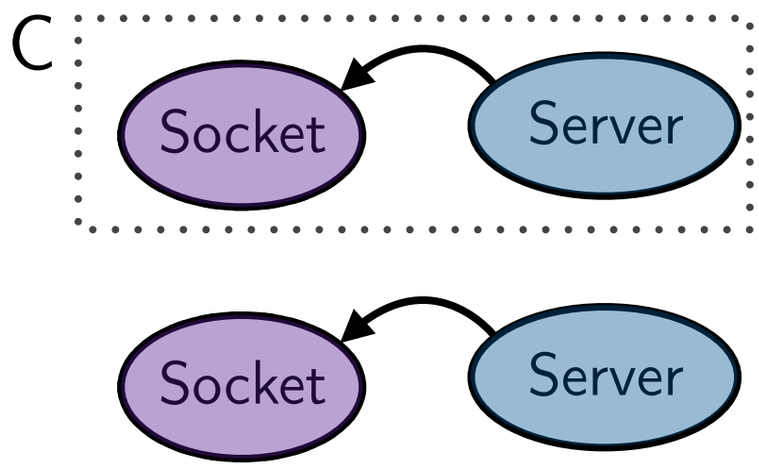
package multi **where**

$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



package multi-shared **where**

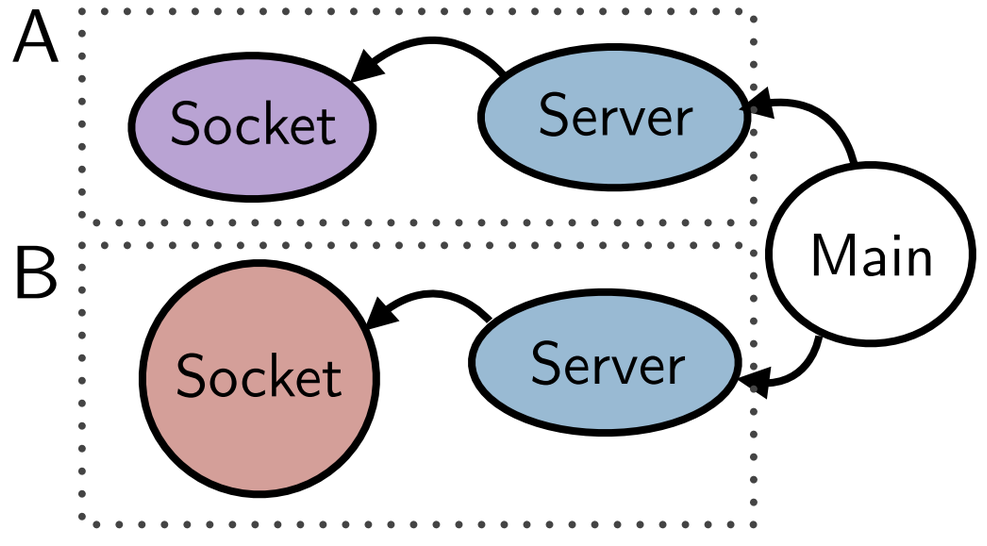
$$
\begin{array}{lcl}
\text{C} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



Shared Reuse

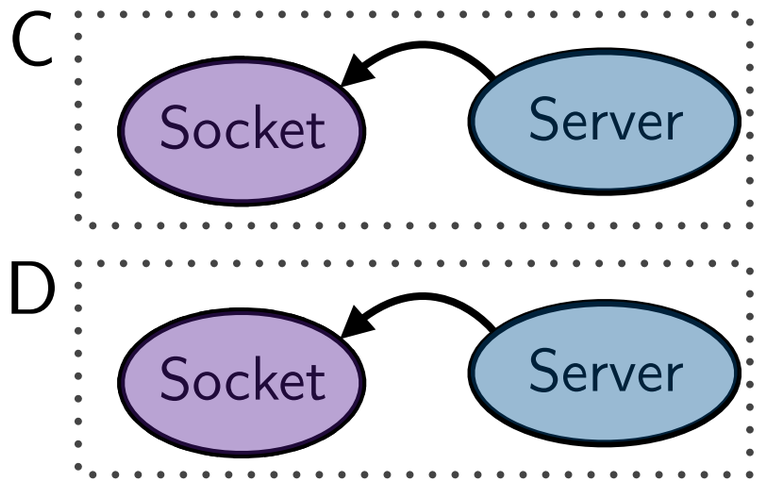
package multi **where**

$$
\begin{aligned}
\text{A} &= \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} &= \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} &= \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{aligned}
$$



package multi-shared **where**

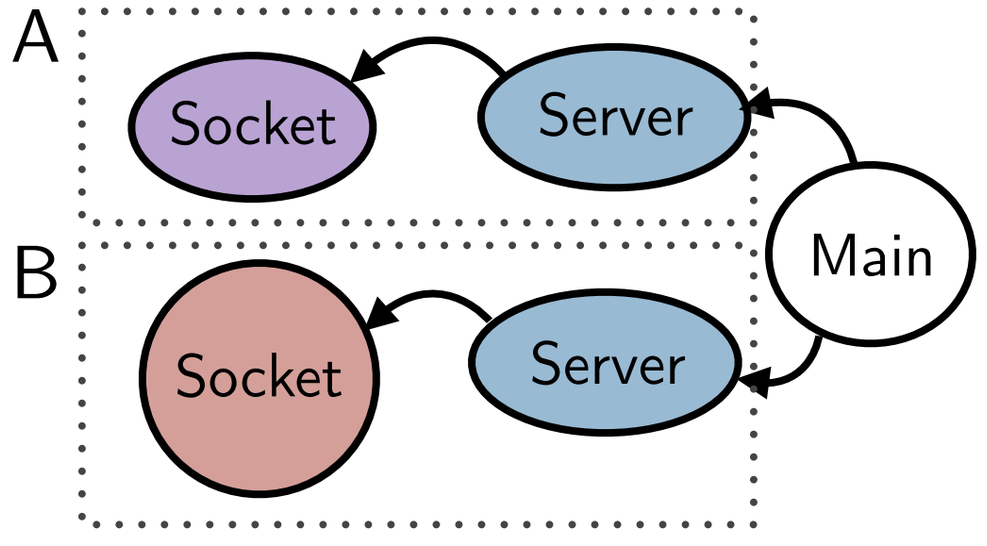
$$
\begin{aligned}
\text{C} &= \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} &= \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} &= \begin{bmatrix} \text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots \end{bmatrix}
\end{aligned}
$$



Shared Reuse

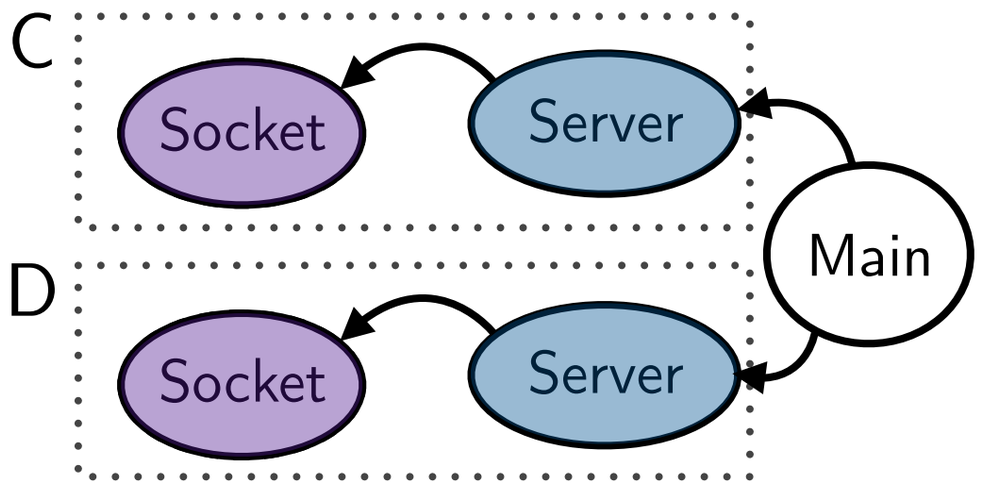
package multi **where**

$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



package multi-shared **where**

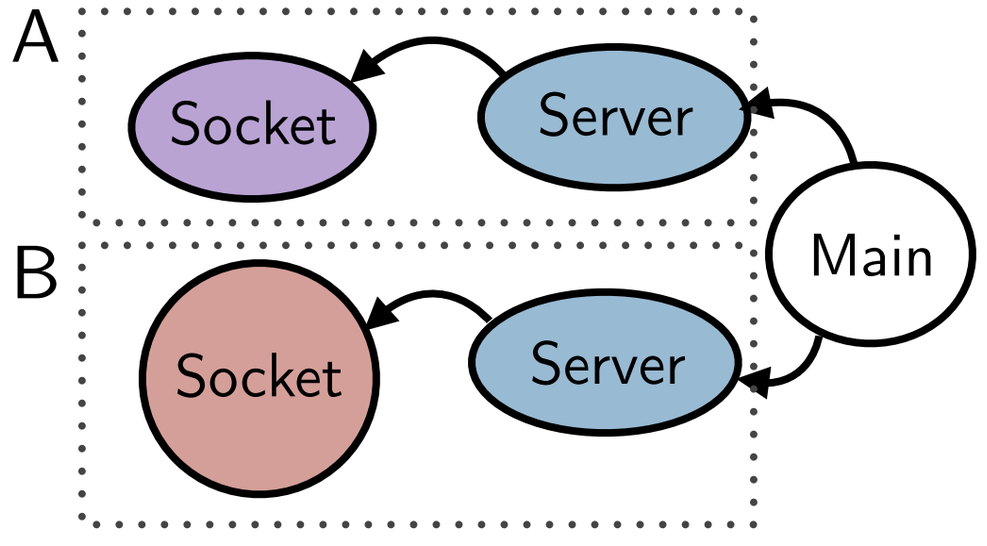
$$
\begin{array}{lcl}
\text{C} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



Shared Reuse

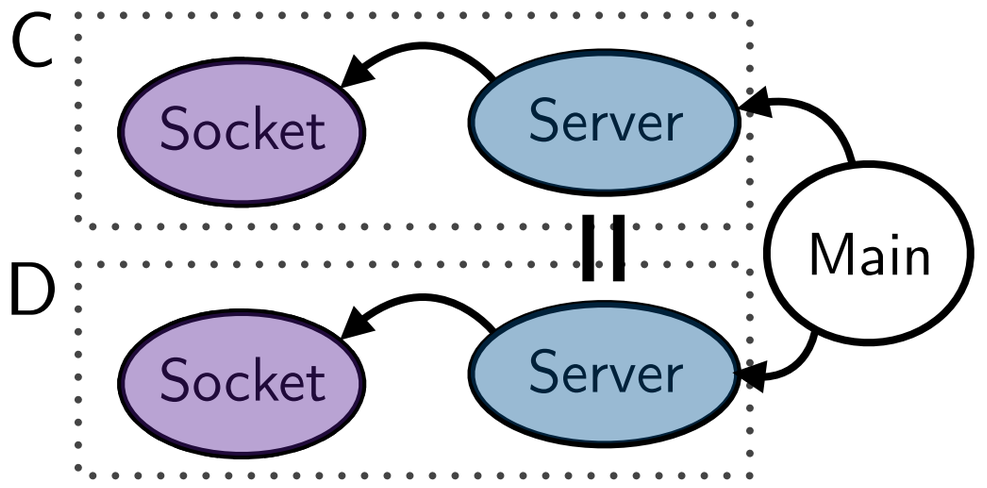
package multi **where**

$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



package multi-shared **where**

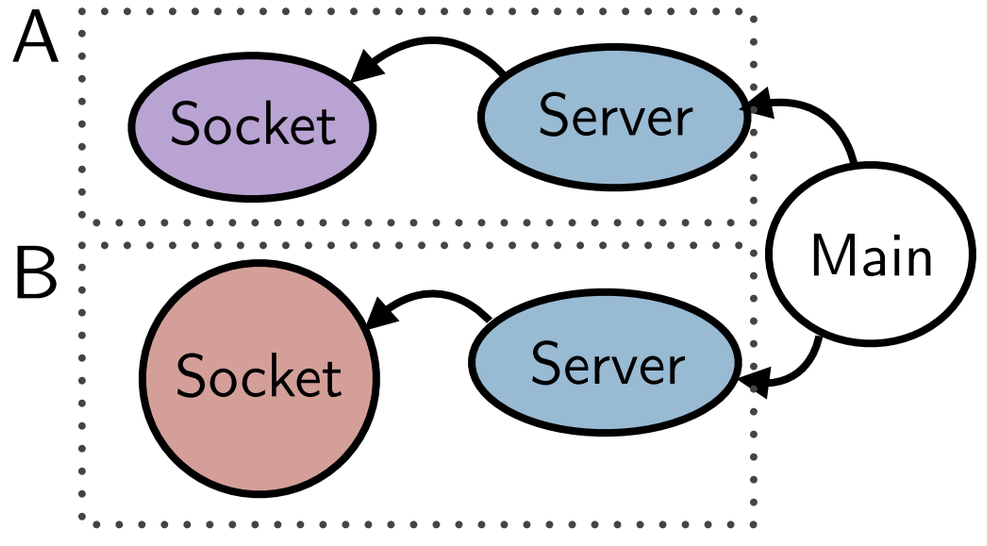
$$
\begin{array}{lcl}
\text{C} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



Shared Reuse

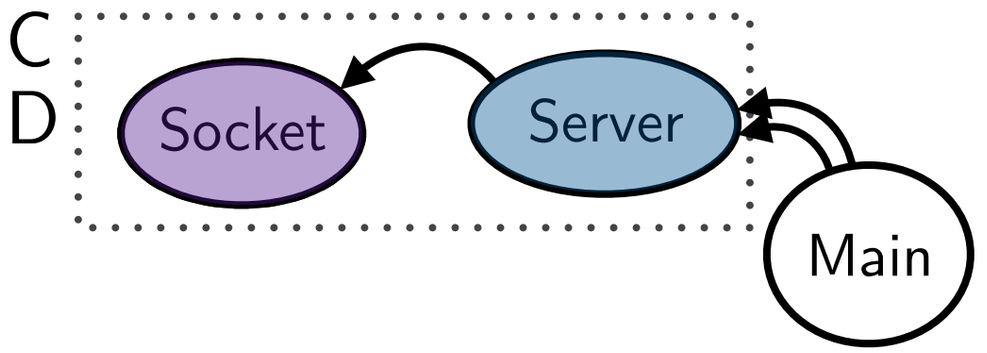
package multi **where**

$$
\begin{array}{lcl}
\text{A} & = & \{\textbf{include}\ \text{server-linked-1}\} \\
\text{B} & = & \{\textbf{include}\ \text{server-linked-2}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified A.Server} \\ \text{import qualified B.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



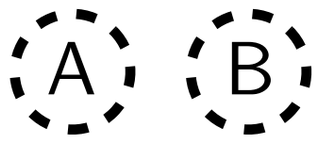
package multi-shared **where**

$$
\begin{array}{lcl}
\text{C} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{D} & = & \{\textbf{include}\ \text{partial-server}; \textbf{include}\ \text{socketimpl-1}\} \\
\text{Main} & = & \begin{bmatrix} \text{import qualified C.Server} \\ \text{import qualified D.Server} \\ \ldots \end{bmatrix}
\end{array}
$$



Recursive Linking

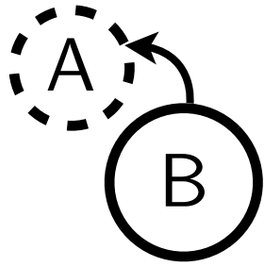
package ab-sigs **where**
  A :: $[S_A]$
  B :: $[S_B]$



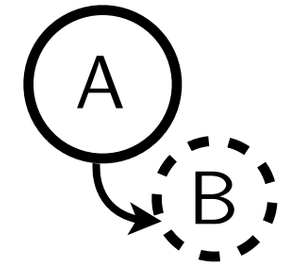
Recursive Linking

package ab-sigs **where**
A :: $[S_A]$
B :: $[S_B]$

package b-from-a **where**
include ab-sigs
B = [import A; ...]



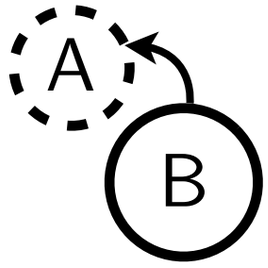
package a-from-b **where**
include ab-sigs
A = [import B; ...]



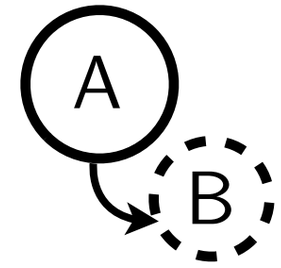
Recursive Linking

package ab-sigs **where**
A :: $[S_A]$
B :: $[S_B]$

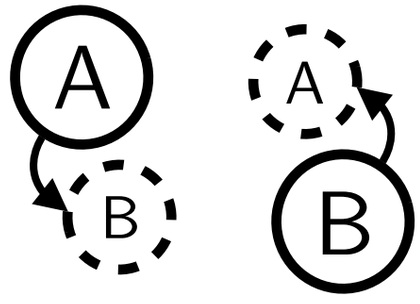
package b-from-a **where**
include ab-sigs
B = [import A; ...]



package a-from-b **where**
include ab-sigs
A = [import B; ...]



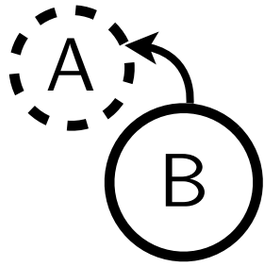
package ab-rec-sep **where**
include a-from-b
include b-from-a



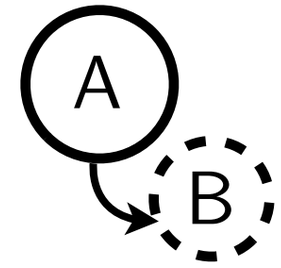
Recursive Linking

package ab-sigs **where**
A :: $[S_A]$
B :: $[S_B]$

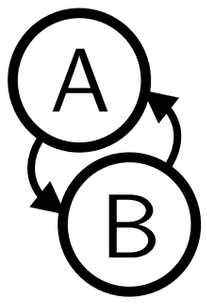
package b-from-a **where**
include ab-sigs
B = [import A; ...]



package a-from-b **where**
include ab-sigs
A = [import B; ...]



package ab-rec-sep **where**
include a-from-b
include b-from-a



Backpack Syntax

Package Names	$P \in PkgNames$
Module Path Names	$p \in ModPaths$
Package Respositories	$R ::= D_1, \ldots, D_n$
Package Definitions	$D ::= \textsf{package}\ P\ t\ \textsf{where}\ B_1, \ldots, B_n$
Bindings	$B ::= p = [M] \mid p :: [S] \mid p = p \mid \textsf{include}\ P\ t\ r$
Thinning Specs	$t ::= (p_1, \ldots, p_n)$
Renaming Specs	$r ::= \langle p_1 \mapsto p'_1, \ldots, p_n \mapsto p'_n \rangle$
Module Expressions	$M ::= imports; exports; defns$
Signature Expressions	$S ::= imports; decls$

interfaces, reuse, and recursive linking ⇒ strong modularity

Backpack Syntax

Package Names	$P \in PkgNames$
Module Path Names	$p \in ModPaths$
Package Respositories	$R ::= D_1, \ldots, D_n$
Package Definitions	$D ::= \textbf{package}\ P\ t\ \textbf{where}\ B_1, \ldots, B_n$
Bindings	$B ::= p = [M] \mid p :: [S] \mid p = p \mid \textbf{include}\ P\ t\ r$
Thinning Specs	$t ::= (p_1, \ldots, p_n)$
Renaming Specs	$r ::= \langle p_1 \mapsto p'_1, \ldots, p_n \mapsto p'_n \rangle$
Module Expressions	$M ::= imports; exports; defns$
Signature Expressions	$S ::= imports; decls$

interfaces, reuse, and recursive linking ⇒ strong modularity

Other features:

- *Aliasing*: module name aliases

Backpack Syntax

Package Names	P	$\in\ PkgNames$
Module Path Names	p	$\in\ ModPaths$
Package Respositories	R	$::= D_1, \ldots, D_n$
Package Definitions	D	$::=$ **package** $P\ t$ **where** $B_1, \ldots, B_n$
Bindings	B	$::= p = [M] \mid p :: [S] \mid p = p \mid$ **include** $P\ t\ r$
Thinning Specs	t	$::= (p_1, \ldots, p_n)$
Renaming Specs	r	$::= \langle p_1 \mapsto p'_1, \ldots, p_n \mapsto p'_n \rangle$
Module Expressions	M	$::= imports; exports; defns$
Signature Expressions	S	$::= imports; decls$

interfaces, reuse, and recursive linking ⇒ strong modularity

Other features:

- *Aliasing*: module name aliases
- *Thinning*: control module exposure

Backpack Syntax

Package Names	P	$\in\ PkgNames$
Module Path Names	p	$\in\ ModPaths$
Package Respositories	R	$::= D_1, \ldots, D_n$
Package Definitions	D	$::=$ **package** $P\ t$ **where** $B_1, \ldots, B_n$
Bindings	B	$::= p = [M] \mid p :: [S] \mid p = p \mid$ **include** $P\ t\ r$
Thinning Specs	t	$::= (p_1, \ldots, p_n)$
Renaming Specs	r	$::= \langle p_1 \mapsto p'_1, \ldots, p_n \mapsto p'_n \rangle$
Module Expressions	M	$::= imports; exports; defns$
Signature Expressions	S	$::= imports; decls$

interfaces, reuse, and recursive linking ⇒ strong modularity

Other features:

- *Aliasing*: module name aliases
- *Thinning*: control module exposure
- *Renaming*: control module naming/linking

Outline

- Language tour
- Semantics
- Future work

Goal 1) Base It on MixML

Start with MixML,* * [Rossberg & Dreyer, ICFP '08]

a minimal, fully expressive core calculus of mixin modules.

Goal 1) Base It on MixML

Start with MixML,* a minimal, fully expressive core calculus of mixin modules.

* [Rossberg & Dreyer, ICFP '08]

but MixML doesn't really work for us:

Goal 1) Base It on MixML

Start with MixML,* * [Rossberg & Dreyer, ICFP '08]

a minimal, fully expressive core calculus of mixin modules.

but MixML doesn't really work for us:

- MixML targets "LTG"

Goal 1) Base It on MixML

Start with MixML,* * [Rossberg & Dreyer, ICFP '08]

a minimal, fully expressive core calculus of mixin modules.

but MixML doesn't really work for us:

- MixML targets "LTG"
- MixML does not support shared reuse

Goal 1) Base It on MixML

Start with MixML,* * [Rossberg & Dreyer, ICFP '08]

a minimal, fully expressive core calculus of mixin modules.

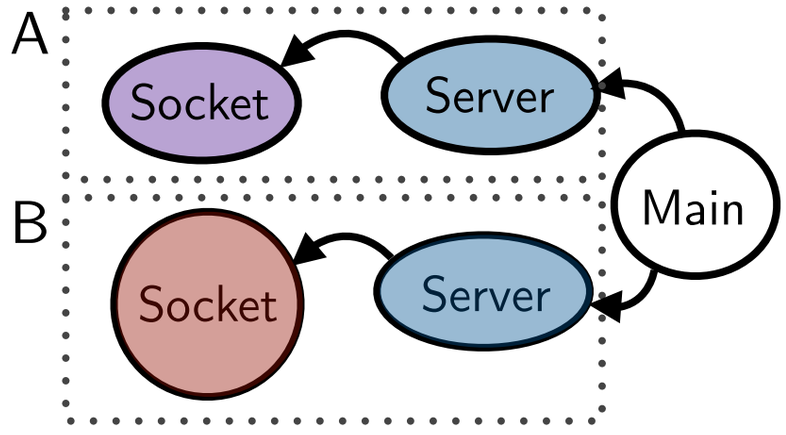
but MixML doesn't really work for us:

- MixML targets "LTG"
- MixML does not support shared reuse
- MixML semantics is pretty complicated

Goal 2) Retrofit Haskell

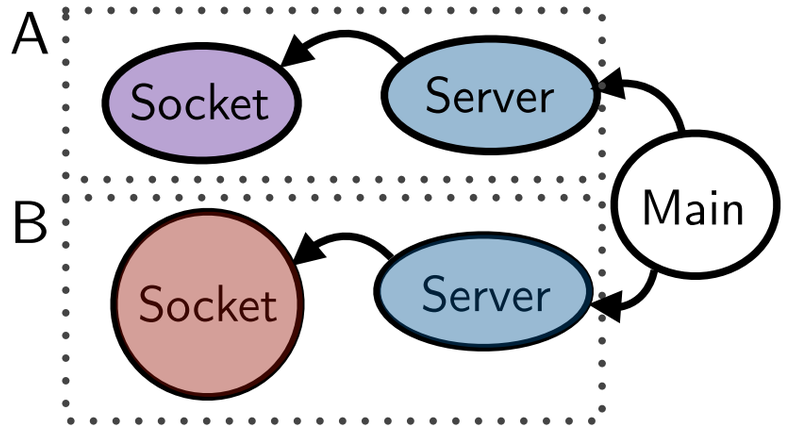
Because we retrofit,
we must "compile" semantics
down to plain Haskell modules.

Type Representation



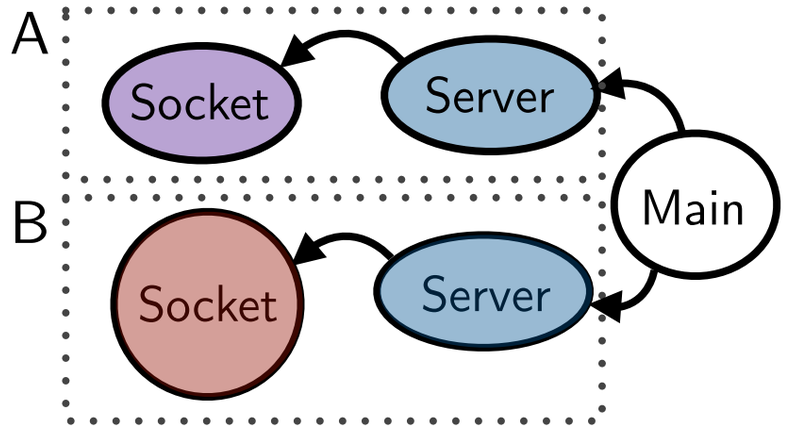
$$\text{Main} = \begin{bmatrix} \texttt{import qualified A.Server} \\ \texttt{import qualified B.Server} \\ \\ \ldots \quad \texttt{A.Server.ServerT} \ \ldots \\ \\ \ldots \quad \texttt{B.Server.ServerT} \ \ldots \end{bmatrix}$$

Type Representation



$$\text{Main} = \left[\begin{array}{l} \texttt{import qualified A.Server} \\ \texttt{import qualified B.Server} \\ \\ \ldots \quad \texttt{A.Server.ServerT} \quad \ldots \\ \\ \ldots \quad \texttt{B.Server.ServerT} \quad \ldots \end{array}\right]$$

Type Representation



$$\text{Main} = \left[\begin{array}{l} \texttt{import qualified A.Server} \\ \texttt{import qualified B.Server} \\ \\ \texttt{...}\ \underbrace{\texttt{Socket}\quad\texttt{Server}}_{\textit{socketimpl-1}\quad\textit{partial-server}}\texttt{.ServerT ...} \\ \\ \texttt{...}\ \underbrace{\texttt{Socket}\quad\texttt{Server}}_{\textit{socketimpl-2}\quad\textit{partial-server}}\texttt{.ServerT ...} \end{array}\right]$$

Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$

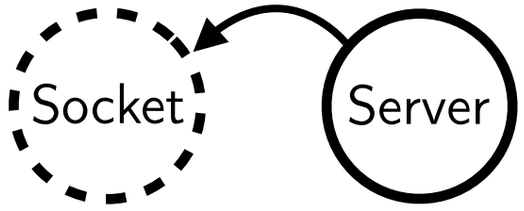
Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$

Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$

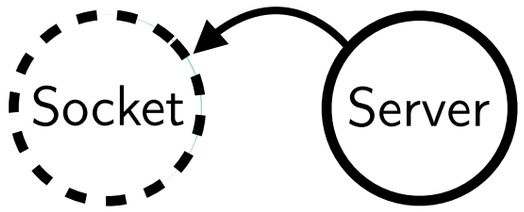
1) Variables for holes.



Module Identities

$$
\begin{array}{lrcl}
\text{Identity Variables} & \alpha, \beta & \in & \mathit{IdentVars} \\
\text{Identity Constructors} & \mathcal{K} & \in & \mathit{IdentCtors} \\
\text{Identities} & \nu & ::= & \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu
\end{array}
$$

1) Variables for holes.

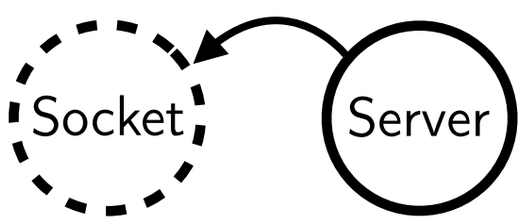


α_{Sock}

Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$

1) Variables for holes.



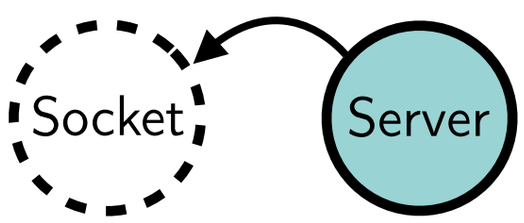
α_{Sock}

2) Fresh token applied to imported mod idents.

Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$

1) Variables for holes.



α_{Sock}

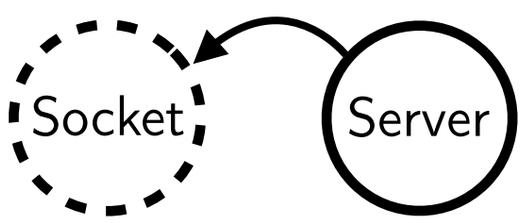
$\mathcal{K}_{\mathsf{Ser}}(\alpha_{\mathsf{Sock}})$

2) Fresh token applied to imported mod idents.

Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$

1) Variables for holes.



α_{Sock}

$\mathcal{K}_{\mathsf{Ser}}(\alpha_{\mathsf{Sock}})$

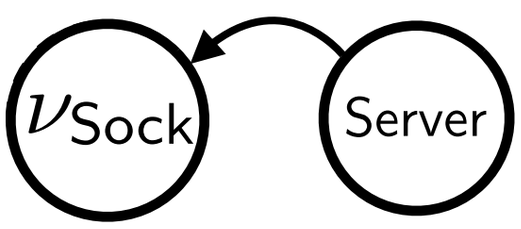
2) Fresh token applied to imported mod idents.

3) Linking is substitution.

Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$

1) Variables for holes.



α_{Sock}

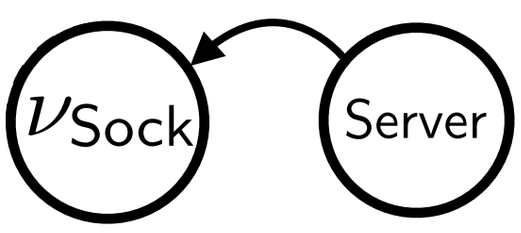
$\mathcal{K}_{\text{Ser}}(\alpha_{\text{Sock}})$

2) Fresh token applied to imported mod idents.

3) Linking is substitution.

Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$



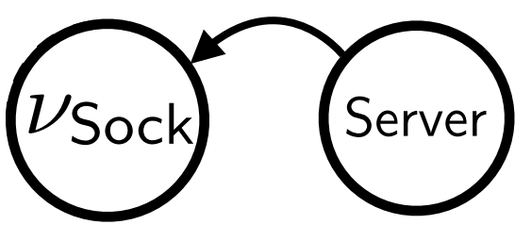
ν_{Sock}

$\mathcal{K}_{\text{Ser}}(\nu_{\text{Sock}})$

1) Variables for holes.
2) Fresh token applied to imported mod idents.
3) Linking is substitution.

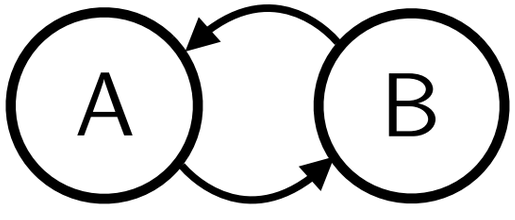
Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$



ν_{Sock}

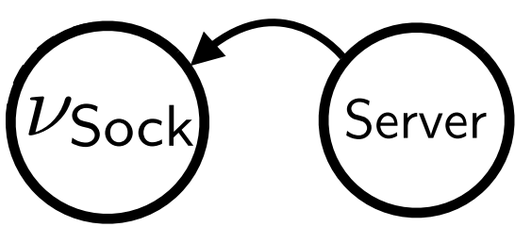
$\mathcal{K}_{\text{Ser}}(\nu_{\text{Sock}})$



1) Variables for holes.
2) Fresh token applied to imported mod idents.
3) Linking is substitution.
4) μ-binders for cyclic graphs (recursion).

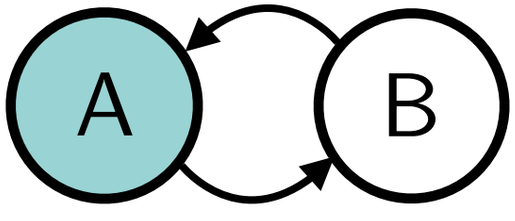
Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$



ν_{Sock}

$\mathcal{K}_{\mathsf{Ser}}(\nu_{\mathsf{Sock}})$



$\mu\alpha.\mathcal{K}_{\mathsf{A}}(\mathcal{K}_{\mathsf{B}}(\alpha))$

$\mu\alpha.\mathcal{K}_{\mathsf{B}}(\mathcal{K}_{\mathsf{A}}(\alpha))$

1) Variables for holes.

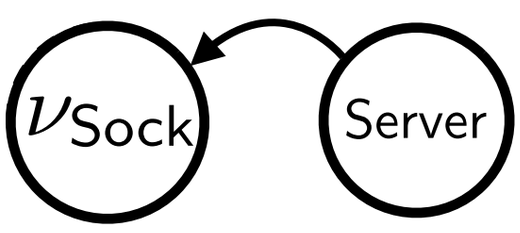
2) Fresh token applied to imported mod idents.

3) Linking is substitution.

4) μ-binders for cyclic graphs (recursion).

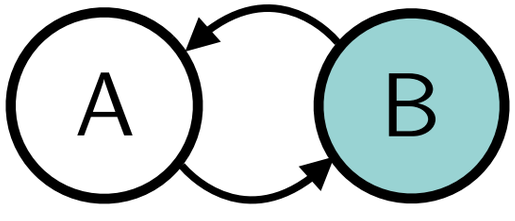
Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$



ν_{Sock}

$\mathcal{K}_{\mathsf{Ser}}(\nu_{\mathsf{Sock}})$



$\mu\alpha.\mathcal{K}_{\mathsf{A}}(\mathcal{K}_{\mathsf{B}}(\alpha))$

$\mu\alpha.\mathcal{K}_{\mathsf{B}}(\mathcal{K}_{\mathsf{A}}(\alpha))$

1) Variables for holes.

2) Fresh token applied to imported mod idents.

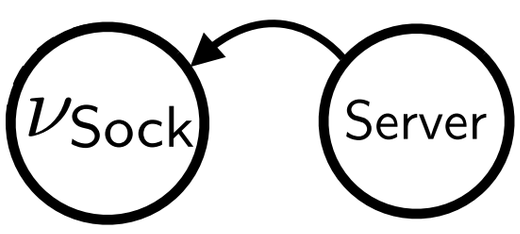
3) Linking is substitution.

4) μ-binders for cyclic graphs (recursion).

Module Identities

Identity Variables	$\alpha, \beta \in IdentVars$
Identity Constructors	$\mathcal{K} \in IdentCtors$
Identities	$\nu ::= \alpha \mid \mathcal{K}\,\overline{\nu} \mid \mu\alpha.\nu$

1) Variables for holes.

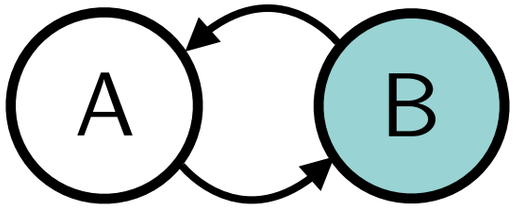


ν_{Sock}

$\mathcal{K}_{\mathsf{Ser}}(\nu_{\mathsf{Sock}})$

2) Fresh token applied to imported mod idents.

3) Linking is substitution.



$\mu\alpha.\mathcal{K}_{\mathsf{A}}(\mathcal{K}_{\mathsf{B}}(\alpha))$

$\mu\alpha.\mathcal{K}_{\mathsf{B}}(\mathcal{K}_{\mathsf{A}}(\alpha))$

4) μ-binders for cyclic graphs (recursion).

module identities enable shared reuse and recursive linking!

module identity

Type System

module identity

Type System

Elaboration

Type System

(two passes on a package)

1) Shaping

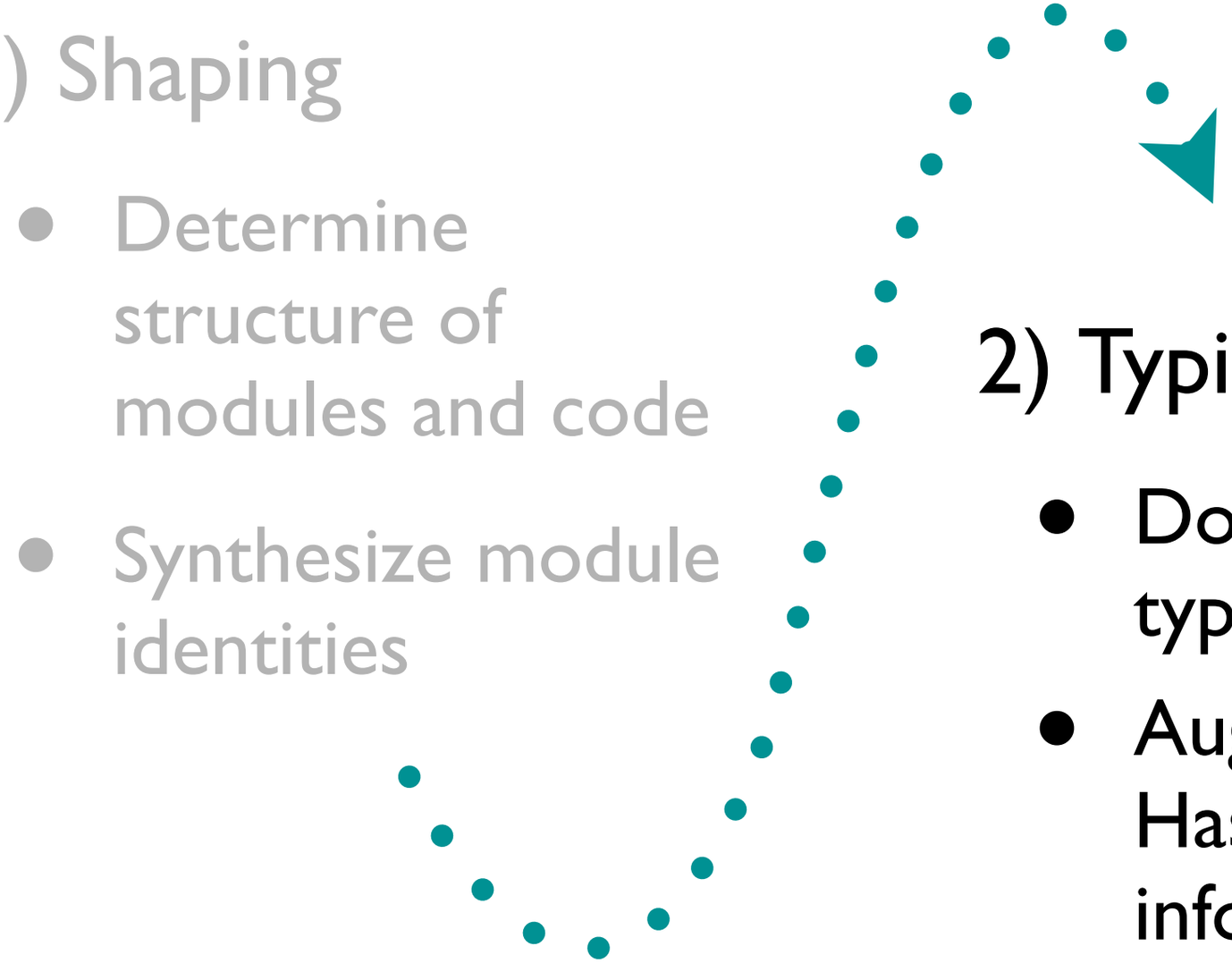
- Determine structure of modules and code
- Synthesize module identities

Type System

(two passes on a package)

1) Shaping

- Determine structure of modules and code
- Synthesize module identities



2) Typing

- Do Haskell typechecking
- Augment shapes with Haskell typing information

Type System: Linking

$$\frac{\Delta \Vdash \overline{B_1} \Rightarrow \tilde{\Xi}_1 \qquad \Delta; \tilde{\Xi}_1 \Vdash B_2 \Rightarrow \tilde{\Xi}_2 \qquad \Vdash \tilde{\Xi}_1 + \tilde{\Xi}_2 \Rightarrow \tilde{\Xi}}{\Delta \Vdash \overline{B_1}, B_2 \Rightarrow \tilde{\Xi}} \text{(SHSEQ)}$$

$$\frac{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1} : \Xi_1 \qquad \Delta; \Xi_1; \tilde{\Xi}_{\mathsf{pkg}} \vdash B_2 : \Xi_2 \qquad \Xi_1 \oplus \Xi_2 = \Xi \text{ defined}}{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1}, B_2 : \Xi} \text{(TYSEQ)}$$

Type System: Linking

$$\frac{\Delta \Vdash \overline{B_1} \Rightarrow \tilde{\Xi}_1 \qquad \Delta; \tilde{\Xi}_1 \Vdash B_2 \Rightarrow \tilde{\Xi}_2 \qquad \Vdash \tilde{\Xi}_1 + \tilde{\Xi}_2 \Rightarrow \tilde{\Xi}}{\Delta \Vdash \overline{B_1}, B_2 \Rightarrow \tilde{\Xi}} \text{(SHSEQ)}$$

$$\frac{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1} : \Xi_1 \qquad \Delta; \Xi_1; \tilde{\Xi}_{\mathsf{pkg}} \vdash B_2 : \Xi_2 \qquad \Xi_1 \oplus \Xi_2 = \Xi \text{ defined}}{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1}, B_2 : \Xi} \text{(TYSEQ)}$$

Type System: Linking

$$\frac{\Delta \Vdash \overline{B_1} \Rightarrow \tilde{\Xi}_1 \qquad \Delta; \tilde{\Xi}_1 \Vdash B_2 \Rightarrow \tilde{\Xi}_2 \qquad \Vdash \tilde{\Xi}_1 + \tilde{\Xi}_2 \Rightarrow \tilde{\Xi}}{\Delta \Vdash \overline{B_1}, B_2 \Rightarrow \tilde{\Xi}} \text{(SHSEQ)}$$

$$\frac{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1} : \Xi_1 \qquad \Delta; \Xi_1; \tilde{\Xi}_{\mathsf{pkg}} \vdash B_2 : \Xi_2 \qquad \Xi_1 \oplus \Xi_2 = \Xi \text{ defined}}{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1}, B_2 : \Xi} \text{(TYSEQ)}$$

Type System: Linking

$$\frac{\Delta \Vdash \overline{B_1} \Rightarrow \tilde{\Xi}_1 \qquad \Delta; \tilde{\Xi}_1 \Vdash B_2 \Rightarrow \tilde{\Xi}_2 \qquad \Vdash \tilde{\Xi}_1 + \tilde{\Xi}_2 \Rightarrow \tilde{\Xi}}{\Delta \Vdash \overline{B_1}, B_2 \Rightarrow \tilde{\Xi}} \text{(SHSEQ)}$$

$$\frac{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1} : \Xi_1 \qquad \Delta; \Xi_1; \tilde{\Xi}_{\mathsf{pkg}} \vdash B_2 : \Xi_2 \qquad \Xi_1 \oplus \Xi_2 = \Xi \text{ defined}}{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1}, B_2 : \Xi} \text{(TYSEQ)}$$

Type System: Linking

unification!

$$\frac{\Delta \Vdash \overline{B_1} \Rightarrow \tilde{\Xi}_1 \qquad \Delta; \tilde{\Xi}_1 \Vdash B_2 \Rightarrow \tilde{\Xi}_2 \qquad \Vdash \tilde{\Xi}_1 + \tilde{\Xi}_2 \Rightarrow \tilde{\Xi}}{\Delta \Vdash \overline{B_1}, B_2 \Rightarrow \tilde{\Xi}} \text{(SHSEQ)}$$

$$\frac{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1} : \Xi_1 \qquad \Delta; \Xi_1; \tilde{\Xi}_{\mathsf{pkg}} \vdash B_2 : \Xi_2 \qquad \Xi_1 \oplus \Xi_2 = \Xi \text{ defined}}{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1}, B_2 : \Xi} \text{(TYSEQ)}$$

Type System: Linking

$$\frac{\Delta \Vdash \overline{B_1} \Rightarrow \tilde{\Xi}_1 \qquad \Delta; \tilde{\Xi}_1 \Vdash B_2 \Rightarrow \tilde{\Xi}_2 \qquad \Vdash \tilde{\Xi}_1 + \tilde{\Xi}_2 \Rightarrow \tilde{\Xi}}{\Delta \Vdash \overline{B_1}, B_2 \Rightarrow \tilde{\Xi}} \text{(SHSEQ)}$$

$$\frac{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1} : \Xi_1 \qquad \Delta; \Xi_1; \tilde{\Xi}_{\mathsf{pkg}} \vdash B_2 : \Xi_2 \qquad \Xi_1 \oplus \Xi_2 = \Xi \text{ defined}}{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1}, B_2 : \Xi} \text{(TYSEQ)}$$

Type System: Linking

$$\frac{\Delta \Vdash \overline{B_1} \Rightarrow \tilde{\Xi}_1 \qquad \Delta; \tilde{\Xi}_1 \Vdash B_2 \Rightarrow \tilde{\Xi}_2 \qquad \Vdash \tilde{\Xi}_1 + \tilde{\Xi}_2 \Rightarrow \tilde{\Xi}}{\Delta \Vdash \overline{B_1}, B_2 \Rightarrow \tilde{\Xi}} \text{(SHSEQ)}$$

$$\frac{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1} : \Xi_1 \qquad \Delta; \Xi_1; \tilde{\Xi}_{\mathsf{pkg}} \vdash B_2 : \Xi_2 \qquad \Xi_1 \oplus \Xi_2 = \Xi \text{ defined}}{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1}, B_2 : \Xi} \text{(TYSEQ)}$$

Type System: Linking

$$\frac{\Delta \Vdash \overline{B_1} \Rightarrow \tilde{\Xi}_1 \qquad \Delta; \tilde{\Xi}_1 \Vdash B_2 \Rightarrow \tilde{\Xi}_2 \qquad \Vdash \tilde{\Xi}_1 + \tilde{\Xi}_2 \Rightarrow \tilde{\Xi}}{\Delta \Vdash \overline{B_1}, B_2 \Rightarrow \tilde{\Xi}} \text{(SHSEQ)}$$

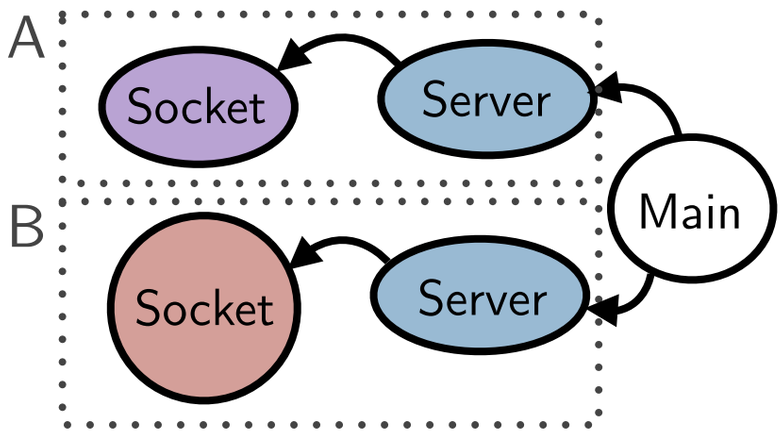
$$\frac{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1} : \Xi_1 \qquad \Delta; \Xi_1; \tilde{\Xi}_{\mathsf{pkg}} \vdash B_2 : \Xi_2 \qquad \Xi_1 \oplus \Xi_2 = \Xi \text{ defined}}{\Delta; \tilde{\Xi}_{\mathsf{pkg}} \vdash \overline{B_1}, B_2 : \Xi} \text{(TYSEQ)}$$

Backpack

MixML

$$\frac{\begin{array}{rr} \vdash \mathcal{L}_1 \text{ locates } \overline{\alpha_1} & \mathcal{R}_1 \# \Sigma_2 \\ \vdash \mathcal{L}_2 \text{ locates } \overline{\alpha_2} & \mathcal{R}_2 \# \Sigma_1 \\ \vdash (\mathcal{L}_1; \Sigma_1) \rightleftarrows (\mathcal{L}_2; \Sigma_2') \rightsquigarrow \delta & \\ \overline{\alpha_1}, \overline{\alpha_2} \text{ fresh} & \end{array} \qquad \begin{array}{r} \Gamma; \mathcal{R} \uplus \mathcal{R}_1 \uplus \mathcal{L}_1; \overline{\beta_1} \vdash mod_1 : \Sigma_1 \\ \Gamma, \mathrm{X} : |\Sigma_1|; \mathcal{R} \uplus \mathcal{R}_2 \uplus \mathcal{L}_2; \overline{\beta_2} \vdash_{\mathrm{stat}} mod_2 : \Sigma_2' \\ \Gamma, \mathrm{X} : |\delta\Sigma_1|; \mathcal{R} \uplus \mathcal{R}_2 \uplus \delta\mathcal{L}_2; \overline{\beta_2} \vdash mod_2 : \Sigma_2 \\ \vdash \delta\Sigma_1 + \Sigma_2 \Rightarrow \Sigma \end{array}}{\Gamma; \mathcal{R} \uplus \mathcal{R}_1 \uplus \mathcal{R}_2; \overline{\beta_1}, \overline{\beta_2} \vdash (\mathrm{X} = mod_1) \texttt{ with } mod_2 : \Sigma} \text{(LINK)}$$

Elaboration




```
module K_Sock where
  data SockT = ...
```

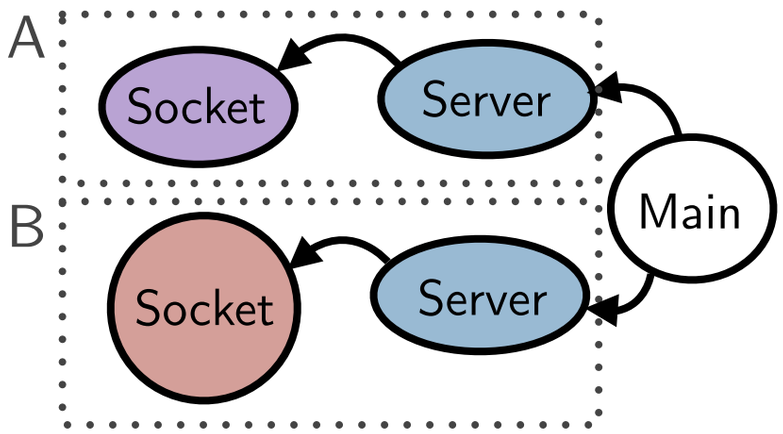
```
module K_Ser(K_Sock) where
  import K_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K^alt_Sock where
  data SockT = ...
```

```
module K_Ser(K^alt_Sock) where
  import K^alt_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K_Main(K_Ser(K_Sock), K_Ser(K^alt_Sock)) where
  import qualified K_Ser(K_Sock) as A.Server(ServerT)
  import qualified K_Ser(K^alt_Sock) as B.Server(ServerT)
  ...
```

Elaboration



- One module file per identity

```
module K_Sock where
  data SockT = ...
```

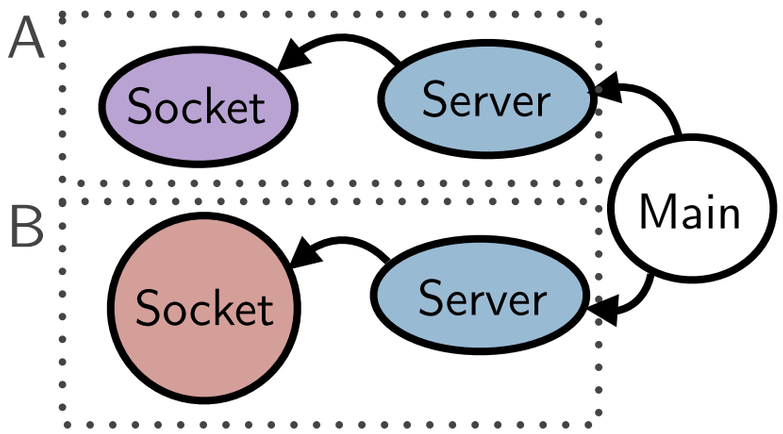
```
module K_Ser(K_Sock) where
  import K_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K^alt_Sock where
  data SockT = ...
```

```
module K_Ser(K^alt_Sock) where
  import K^alt_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K_Main(K_Ser(K_Sock), K_Ser(K^alt_Sock)) where
  import qualified K_Ser(K_Sock) as A.Server(ServerT)
  import qualified K_Ser(K^alt_Sock) as B.Server(ServerT)
  ...
```

Elaboration



- One module file per identity
- Code duplicated (a la C++ templates)

```
module K_Sock where
  data SockT = ...
```

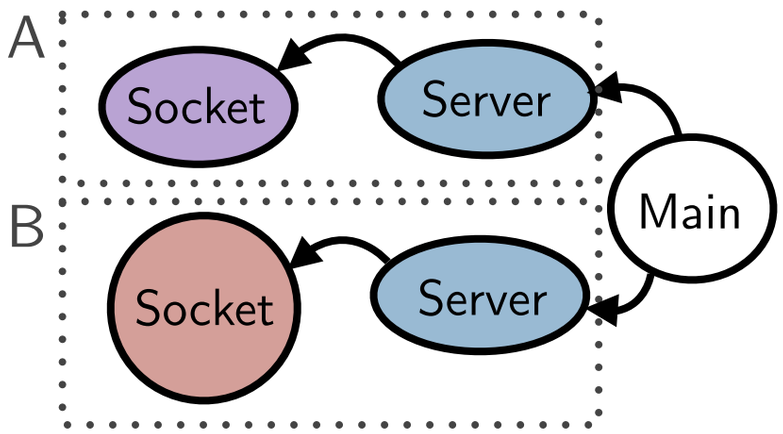
```
module K_Ser(K_Sock) where
  import K_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K^alt_Sock where
  data SockT = ...
```

```
module K_Ser(K^alt_Sock) where
  import K^alt_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K_Main(K_Ser(K_Sock), K_Ser(K^alt_Sock)) where
  import qualified K_Ser(K_Sock) as A.Server(ServerT)
  import qualified K_Ser(K^alt_Sock) as B.Server(ServerT)
  ...
```

Elaboration



- One module file per identity
- Code duplicated (a la C++ templates)
- Code largely preserved

```
module K_Sock where
  data SockT = ...
```

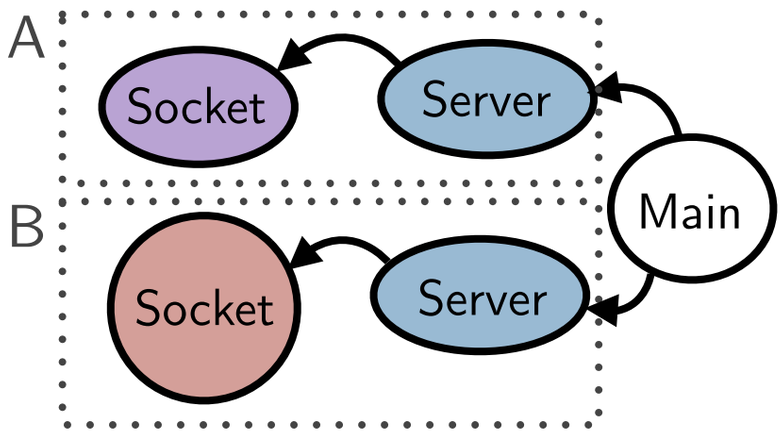
```
module K_Ser(K_Sock) where
  import K_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K^alt_Sock where
  data SockT = ...
```

```
module K_Ser(K^alt_Sock) where
  import K^alt_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K_Main(K_Ser(K_Sock), K_Ser(K^alt_Sock)) where
  import qualified K_Ser(K_Sock) as A.Server(ServerT)
  import qualified K_Ser(K^alt_Sock) as B.Server(ServerT)
  ...
```

Elaboration



- One module file per identity
- Code duplicated (a la C++ templates)
- Code largely preserved
- Rewrite imports to identities

```
module K_Sock where
  data SockT = ...
```

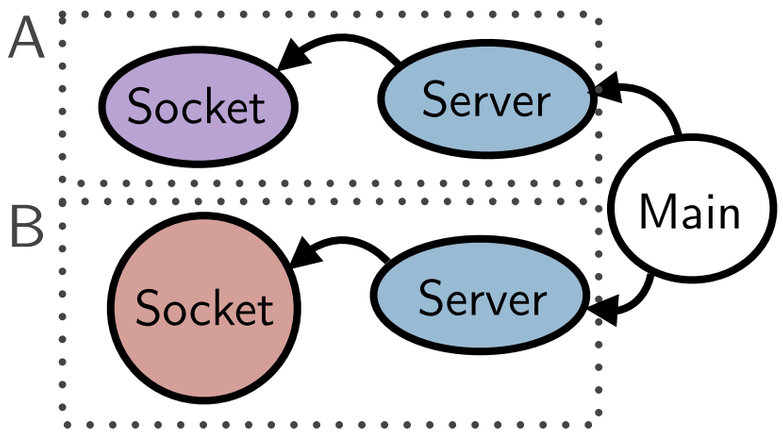
```
module K_Ser(K_Sock) where
  import K_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K^alt_Sock where
  data SockT = ...
```

```
module K_Ser(K^alt_Sock) where
  import K^alt_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K_Main(K_Ser(K_Sock), K_Ser(K^alt_Sock)) where
  import qualified K_Ser(K_Sock) as A.Server(ServerT)
  import qualified K_Ser(K^alt_Sock) as B.Server(ServerT)
  ...
```

Elaboration



- One module file per identity
- Code duplicated (a la C++ templates)
- Code largely preserved
- Rewrite imports to identities

```
module K_Sock where
  data SockT = ...
```

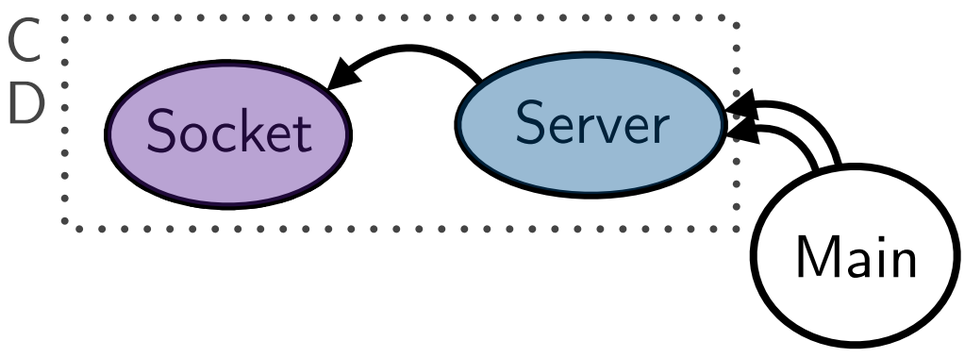
```
module K_Ser(K_Sock) where
  import K_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K^alt_Sock where
  data SockT = ...
```

```
module K_Ser(K^alt_Sock) where
  import K^alt_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K_Main(K_Ser(K_Sock), K_Ser(K^alt_Sock)) where
  import qualified K_Ser(K_Sock) as A.Server(ServerT)
  import qualified K_Ser(K^alt_Sock) as B.Server(ServerT)
  ...
```

Elaboration



- One module file per identity
- Code duplicated (a la C++ templates)
- Code largely preserved
- Rewrite imports to identities

```
module K_Sock where
  data SockT = ...
```

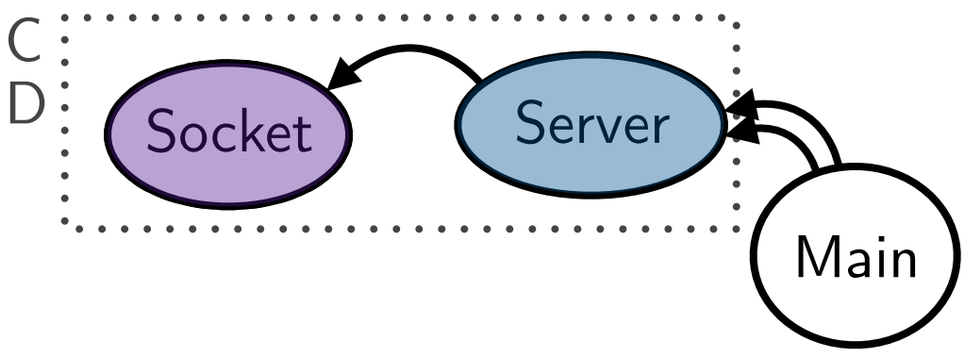
```
module K_Ser(K_Sock) where
  import K_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K^alt_Sock where
  data SockT = ...
```

```
module K_Ser(K^alt_Sock) where
  import K^alt_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K_Main(K_Ser(K_Sock), K_Ser(K^alt_Sock)) where
  import qualified K_Ser(K_Sock) as A.Server(ServerT)
  import qualified K_Ser(K^alt_Sock) as B.Server(ServerT)
  ...
```

Elaboration



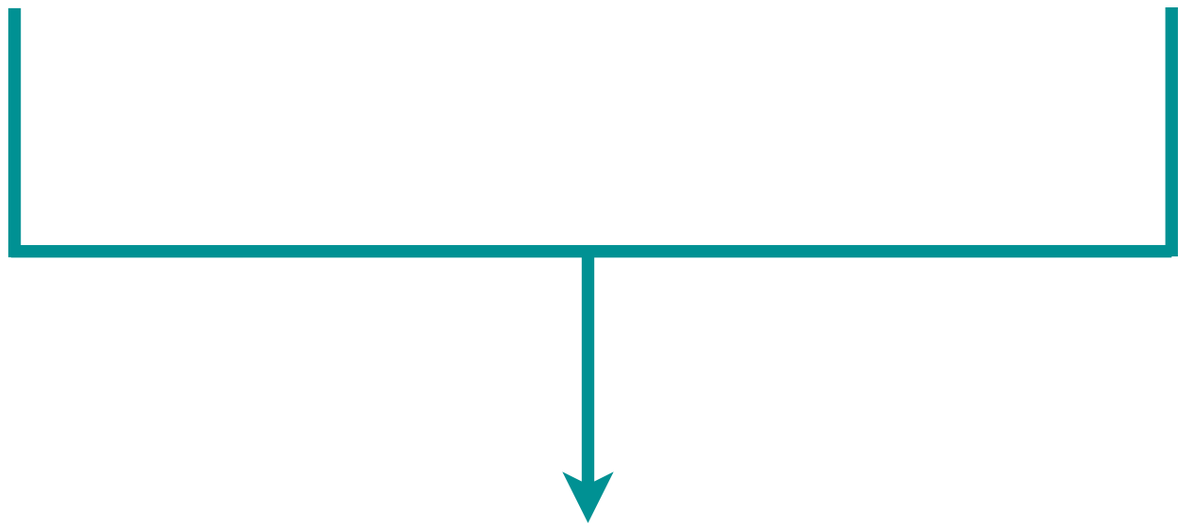
- One module file per identity
- Code duplicated (a la C++ templates)
- Code largely preserved
- Rewrite imports to identities

```
module K_Sock where
  data SockT = ...
```

```
module K_Ser(K_Sock) where
  import K_Sock as Socket(SockT)
  data ServerT = ...SockT...
```

```
module K_Main(K_Ser(K_Sock), K_Ser(K_Sock)) where
  import qualified K_Ser(K_Sock) as C.Server(ServerT)
  import qualified K_Ser(K_Sock) as D.Server(ServerT)
  ...
```

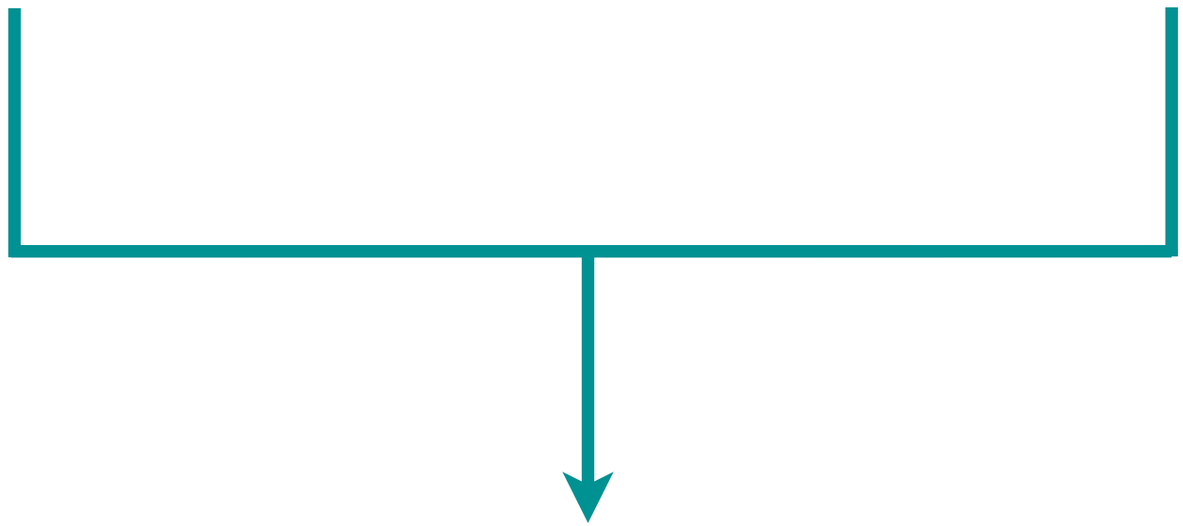
Type System



module identity
simplifies and organizes
the semantics

Type System

Elaboration



module identity
simplifies and organizes
the semantics

What else is in the paper?

- Thinning and renaming
- Formalization of Haskell modules
- Elaboration soundness statement & proof
- Metatheory/axioms for "core language"

What else is in the ~~paper~~?

53-page appendix

- Thinning and renaming
- Formalization of Haskell modules
- Elaboration soundness statement & proof
- Metatheory/axioms for “core language”

Future Work

type classes

&

integration with Cabal package manager

Contributions

- Retrofits Haskell with strong modularity
  - Designed at *package* level
  - Employs simplified *mixin* design
  - Defined as *elaboration* into weak Haskell modules
  - Separate *typechecking*, not separate compilation
- Generic design could work for other weak langs

Targeting an existing, weak module language is not just practical but *interesting!*

Targeting an existing, weak module language is not just practical but *interesting!*

Thanks!

Bonus Materials

Proving soundness was hard.

- First had to formally define "plain Haskell modules" and binary interface files
  - But not Haskell typechecking!
- Only formalization* of Haskell modules with:
  - separately checkable recursive modules
  - conventional metatheory, e.g. *Weakening, Substitution*
- Definitions and metatheory are parameterized by axioms for "core" level of terms and types
  - Could generalize to other "weak" langs with Haskell's module semantics

* others: [Faxén, JFP '02] [Diatchki et al., Haskell '02]

Type Classes

package problematic **where**

```
A :: [data T; ...]
B :: [data T; ...]
C = [import qualified A
     import qualified B
     instance Eq A.T where
       eq t1 t2 = True
     instance Eq B.T where
       eq t1 t2 = False]
```

- Which instances from A and B impls does C see?
  - Elab. of C must specify imported instances
- Package type presumes A.T distinct from B.T
  - Can't use same impl for A and for B

Implementation Check

package `containers-sig-1.x` **where**
  include `prelude-sig-4.x`
  Data.Set :: [import Prelude; ...]

package `containers-impl-1.5` **where**
  include `prelude-sig-4.x`
  HiddenImpl = [...]

$$\text{Data.Set} = \begin{bmatrix} \text{import Prelude} \\ \text{import HiddenImpl} \\ \ldots \end{bmatrix}$$

  implements `containers-sig-1.x`

- Author wants to check that she implements `sig`
- But not all holes in `sig` are implemented in `impl`
- How exactly to define **implements** then?

Conditionals

package `mylib` **where**

```
Foo = [ #ifdef __GHC__
        import GhcStuff
        ...ghc-specific impl...
        #else
        ...default impl...
        #endif ]
```

wrong!
`#if` in program
is not typeable
at package level

right!
conditional at
the level of
package language!

branches might
introduce their
own bindings

package `mylib` **where**

```
Foo = ( case compiler of
          ghc ⇒ [ import GhcStuff
                  ...ghc-specific impl... ]
          _   ⇒ [...default impl...] )
```