Backpack: Retrofitting Haskell with Interfaces

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

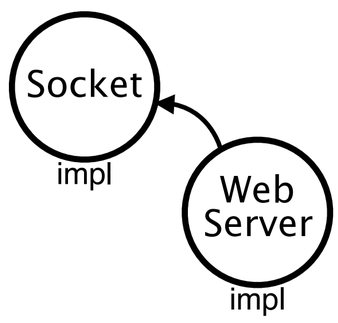
Two kinds of modularity

Incremental Modular Development

impls depend on *impls*; develop *incrementally*

```
module Socket where
  data Sock = MkS Int
  open = λn. MkS n

module WebServer where
  import Socket
  ... open 80 ...
```

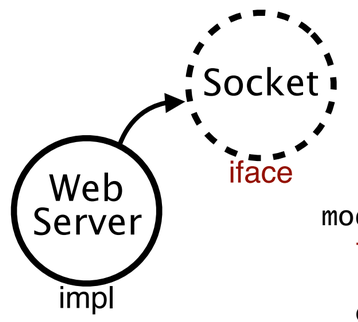


IMD vs. **SMD**

Haskell style | ML style

Separate Modular Development

impls depend on *interfaces*; develop *separately*

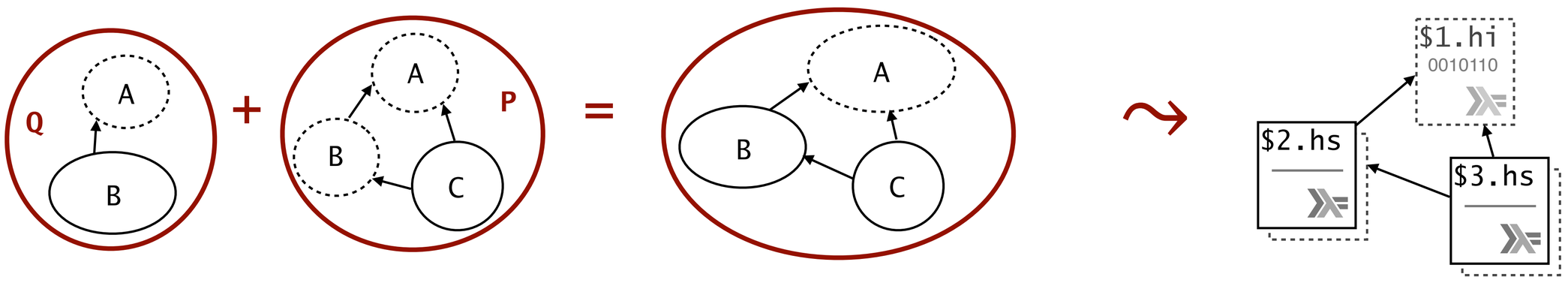



```
module type SOCKET = sig
  type Sock = MkS of int
  val open : int -> Sock
end

module WebServer =
  functor (S : SOCKET) -> struct
    ... S.open 80 ...
  end
```

Goal

★ **Build support for SMD on top of existing Haskell IMD.** ★



Design at package level
- Leave module level alone
- Add interfaces aka signatures

Packages are mixins, *not functors*
- Some modules impl'd; others spec'd
- Modules link (recursively) by name

Elaborate to plain Haskell modules
- Thus works with existing tools
- Preserves code but rewrites imports

Example packages

```
package socket-sig where
  Socket :: [data Socket
             new :: Int -> Socket]

package webserver where
  include socket-sig
  Server = [import Socket
            loop = ... new 80 ...]

package socket-impl where
  Socket = [data Socket = MkS Int
            new = λ n. MkS n]

package client where
  include webserver
  include socket-impl
  Client = [import qualified Server
            main = Server.loop]
```

- **Signature** denotes a "hole" for a module implementation
- **Package inclusion** dumps its contents into scope
- Module **import** sees only the interface of imported module
- Server **implementation** depends on undefined Socket
- **Linking** occurs since two modules named Socket are dumped into same namespace
- Implementation in socket **must match** signature in webserver

- **Elaboration** rewrites module names to semantic identities

elaboration of client package

```
module #S[] where
  data Socket = MkS Int
  new = λn. MkS n
```

```
module #W[#S] where
  import #S[] as Socket
  loop = ... new 80 ...
```

```
module #C[#W[#S]] where
  import qualified
     #W[#S] as Server
  main = Server.loop
```

See our paper in POPL 2014!