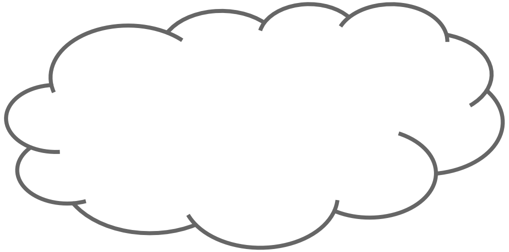


Cloud Haskell

network transport of data types

“Erlang-style concurrent and distributed programming in Haskell.”

Concurrent Haskell

```
forkIO :: IO () -> IO ThreadId
```

Lightweight processes

Message passing (Channels)

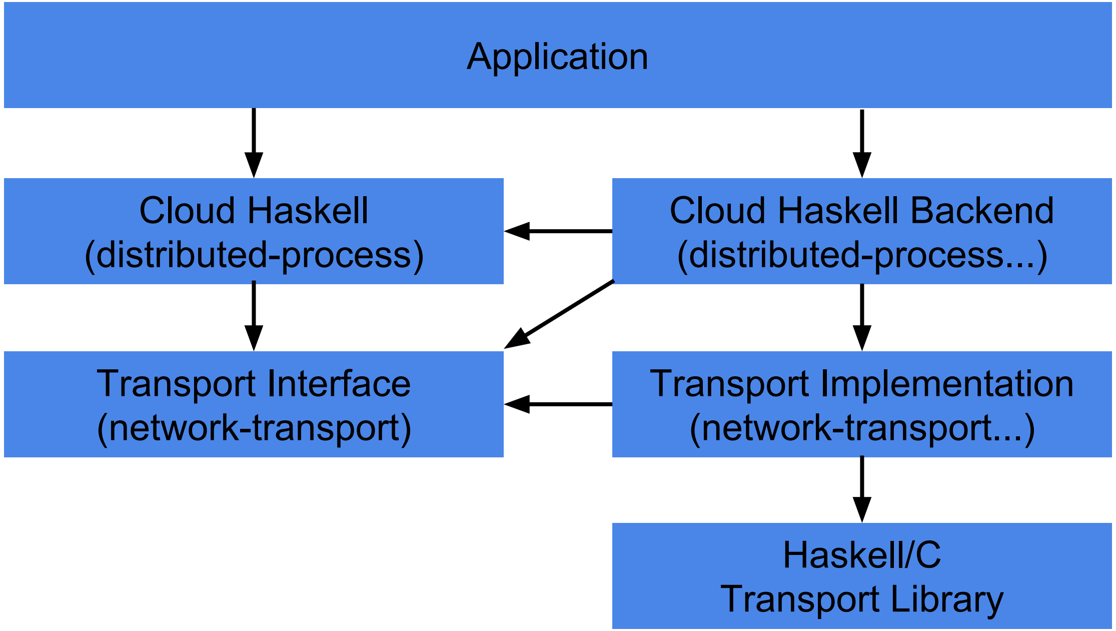
networking?

Network.Socket

```
socket :: Family -> SocketType -> ... -> IO Socket
```

serializing data types?

Cloud Haskell Design



`<- .Backend.`
`SimpleLocalnet`

`<- .TCP`

DSL for distributed computations

The Process Monad

- important / prominent monad

```
newtype Process a = Process
  {unProcess :: ReaderT LocalProcess IO a}
  deriving (Functor, Applicative, Monad,
            MonadIO, MonadReader LocalProcess,
            Typeable, Applicative)
```

Example: Chat.hs

Chat server and client on different machines

Serializable data types

```
data ChatData = MasterInfo ProcessId | ChatMessage String
   deriving (Typeable, Show)
instance Binary ChatData where
   put (MasterInfo  p) = put (0::Word8) >> put p
   put (ChatMessage s) = put (1::Word8) >> put s
   get                 = do val <- getWord8
                            case val of
                               0 -> liftM MasterInfo  get
                               1 -> liftM ChatMessage get
```

```haskell
masterLoop :: Backend -> [NodeId] -> Process ()
masterLoop b _ = forever $
   do slaves <- findSlaves b
      pid    <- getSelfPid
      -- Frequently send master pid to possibly new clients
      forM_ slaves (\x -> send x (MasterInfo pid))
      -- Listen for messages, forward each message to all clients
      msg <- expectTimeout 10 :: Process (Maybe ChatData)
      case msg of
         Nothing  -> return ()
         Just msg -> do forM_ slaves (\c -> send c msg)
                        liftIO $ putStrLn (show msg)
```

```haskell
startCustomSlave :: Backend -> Process () -> IO ()
startCustomSlave backend func = do
   node <- newLocalNode backend
   runProcess node func

sendMsg :: ProcessId -> Process ()
sendMsg pid = forever $ do fromUser <- (liftIO getLine)
                           send pid (ChatMessage fromUser)
```

```haskell
slaveLoop :: Process ()
slaveLoop = do
   pid <- getSelfPid
   register "slaveController" pid
   m <- expect :: Process ChatData
   case m of
      (MasterInfo p) -> do
         liftIO . putStrLn $ "Master pid: " ++ show p
         spawnLocal (sendMsg p) -- Spawn a local process that listens for input
         forever (do msg <- expect :: Process ChatData -- Listen for messages
                     case msg of
                        (ChatMessage s) -> liftIO $ putStrLn s
                         _              -> return () )
      _ -> do liftIO $ putStrLn "error: could not find server"
```

```haskell
main :: IO ()
main = do
   args <- getArgs
   case args of
      ["server", host, port] -> do
         backend <- initializeBackend host port initRemoteTable
         startMaster backend (masterLoop backend)
      ["client", host, port] -> do
         backend <- initializeBackend host port initRemoteTable
         startCustomSlave backend slaveLoop
```

Chat away

```
runhaskell Chat.hs server 127.0.0.1 10001
runhaskell Chat.hs client  127.0.0.1 10002
runhaskell Chat.hs client  127.0.0.1 10003
runhaskell Chat.hs client  127.0.0.1 10004
```

Installing

base < 5:

```
cabal install distributed-process
cabal install distributed-process-simplelocalnet
```

base >= 5:

sandbox…

Download link:

github.com/ksallberg/sthlmhaskell

- installing instructions
- chat server/client code
- pdf tutorial
- relevant articles/papers