Swift Network Programming

with Monads

John Gallagher (@nerdyjkg)
Big Nerd Ranch

Agenda

- What are Monads?
- `Result` and Error Handling
- `Deferred`
- `DeferredTCPSocket`

What is a Monad?

“A monad is just a monoid in the category of endofunctors”

- StackOverflow
(but really *Categories for the Working Mathematician*)

Monad “Protocol”

```
// NOT VALID SWIFT CODE

protocol Monad<T> {
  init(value: T)

  func bind<U>(f: T -> Self<U>) -> Self<U>
}
```

Optional is a Monad

```
extension Optional {
    // init is already defined

    // if `self == nil`, returns nil.
    // otherwise, returns f(self!)
    func bind<U>(f: T -> U?) -> U? {
        if let t = self {
            return f(t)
        } else {
            return nil
        }
    }
}
```

Monadic `map`

```swift
// From the Swift standard library:
enum Optional<T> {
  // . . .

  /// If `self == nil`, returns `nil`.
  /// Otherwise, returns `f(self!)`.
  func map<U>(f: T -> U) -> U?
}
```

Result

Result

```
enum Result<T, E> {
    case Success(T)
    case Failure(E)
}
```

Result

```
protocol ErrorType: Printable {
}

enum Result<T> {
    case Success(T)
    case Failure(ErrorType)
}
```

https://github.com/LlamaKit/LlamaKit/issues/10

https://github.com/rust-lang/rfcs/pull/201

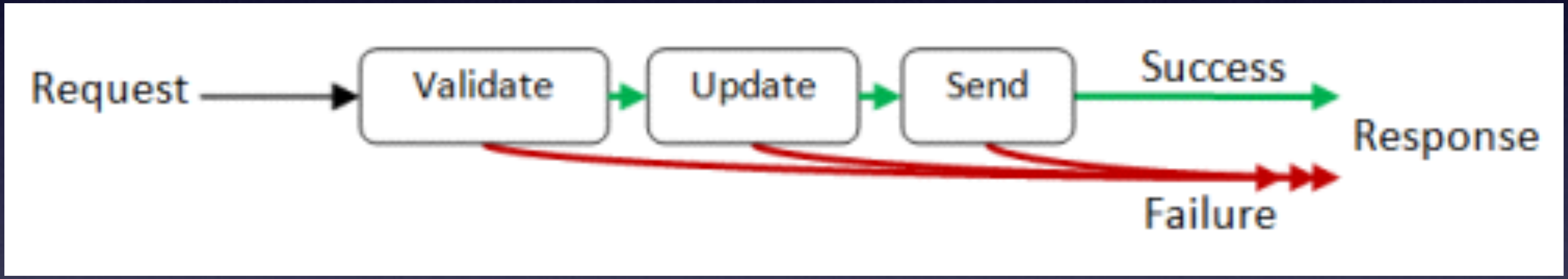
Result

```swift
enum Result<T> {
    case Success(@autoclosure () -> T)
    case Failure(ErrorType)
}
```

Result Monad

```
extension Result {
  func bind<U>(f: T -> Result<U>) -> Result<U> {

    switch self {
    case let .Success(value):
      return f(value)

    case let .Failure(error):
      return .Failure(error)
    }

  }
}
```

Result Train Tracks



http://fsharpforfunandprofit.com/posts/recipe-part2/

Result Example

```
func readString(sock: Socket) -> Result<String>
func parseMessage(str: String) -> Result<Message>

func readMessage(sock: Socket) -> Result<Message> {
    let stringResult = readString(sock)
    return stringResult.bind(parseMessage)
}
```

Deferred

Deferred

```
class Deferred<T> {
    init()
    init(value: T)

    var isFilled: Bool
    func fill(value: T)

    func peek() -> T?
    func upon(block: T -> ())
}
```

Deferred Monad

```
extension Deferred {
    func bind<U>(f: T -> Deferred<U>)
        -> Deferred<U>

    func map<U>(f: T -> U)
        -> Deferred<U>
}
```

Deferred Example

```
func connectOverTCP(host: String)
    -> Deferred<Socket>

func handshake(socket: Socket)
    -> Deferred<Connection>

func connect(host: String)
    -> Deferred<Connection>
{
    let socket = connectOverTCP(host)
    return socket.bind(handshake)
}
```

Deferred (real) Example

```
func connectOverTCP(host: String)
    -> Deferred<Result<Socket>>

func handshake(socket: Socket)
    -> Deferred<Result<Connection>>

func connect(host: String)
    -> Deferred<Result<Connection>>
{
    let defSocket = connectOverTCP(host)
    return ???
}
```

Monad Transformer

```
func resultToDeferred<T,U>(r: Result<T>,
                           f: T -> Deferred<Result<U>>)
    -> Deferred<Result<U>>
{
    switch r {
    case let .Success(value):
        return f(value)

    case let .Failure(error):
        return Deferred(value: .Failure(error))
    }
}
```

Deferred (real) Example

```swift
func connectOverTCP(host: String)
    -> Deferred<Result<Socket>>

func handshake(socket: Socket)
    -> Deferred<Result<Connection>>

func resultToDeferred<T,U>(r: Result<T>,
                           f: T -> Deferred<Result<U>>)
    -> Deferred<Result<U>>

func connect(host: String)
    -> Deferred<Result<Connection>>
{
    let defSocket = connectOverTCP(host)
    return defSocket.bind { resultToDeferred($0, handshake) }
}
```

Deferred vs Completion Blocks

- Returning a Deferred can trivially replace completion blocks via upon / uponQueue
- Deferreds can also be combined in interesting ways:

```
// wait for both Deferreds to complete
func both<T,U>(d1: Deferred<T>, d2: Deferred<U>)
    -> Deferred<(T,U)>

// combine an array, waiting for all to complete
func all<T>(deferreds: [Deferred<T>]) -> Deferred<[T]>

// wait for the first of any some Deferreds to complete
func any<T>(deferreds: [Deferred<T>]) -> Deferred<Deferred<T>>
```

DeferredTCPSocket

Asynchronous Socket Programming

- Uses C APIs for creating sockets, connecting
- Uses GCD dispatch sources for async I/O
- Much annoyance using both from Swift

TCPAcceptSocket

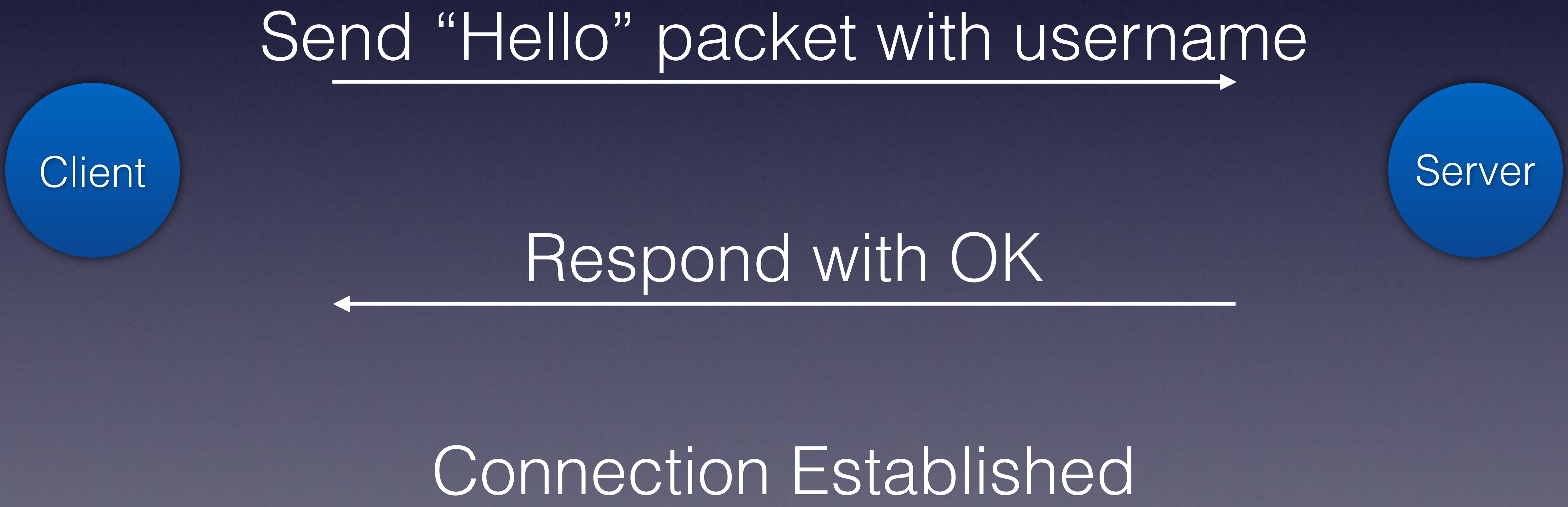
```
class TCPAcceptSocket {

    typealias ConnectionHandler =
        (queue: dispatch_queue_t,
         callback: TCPCommSocket -> ())

    class func accept(
        onPort port: UInt16,
        withConnectionHandler: ConnectionHandler)
      -> Result<TCPAcceptSocket>

    func close()

}
```

TCPCommSocket

```
class TCPCommSocket {

    class func connectToHost(host: String,
                             serviceOrPort: String)
        -> Deferred<Result<TCPCommSocket>>

    func readData() -> Deferred<Result<NSData>>

    func writeData(data: NSData) -> Deferred<Result<()>>

}
```

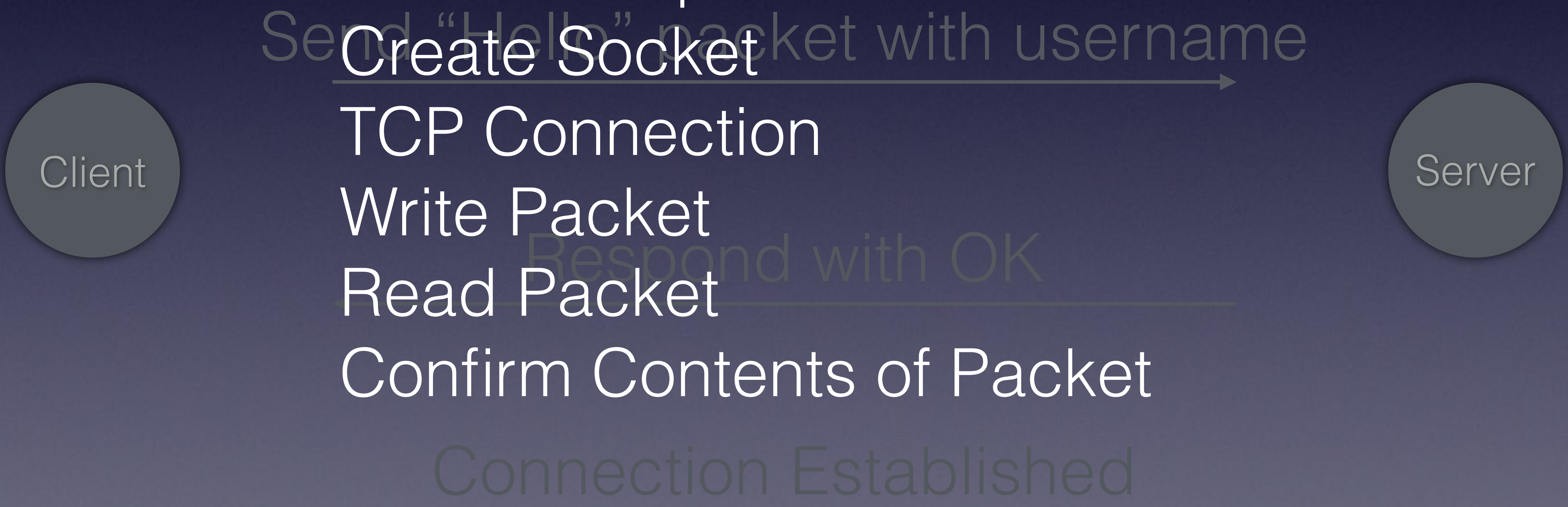
Networking Example

"Just" a "Simple" Handshake



Networking Example

“Just” a “Simple” Handshake



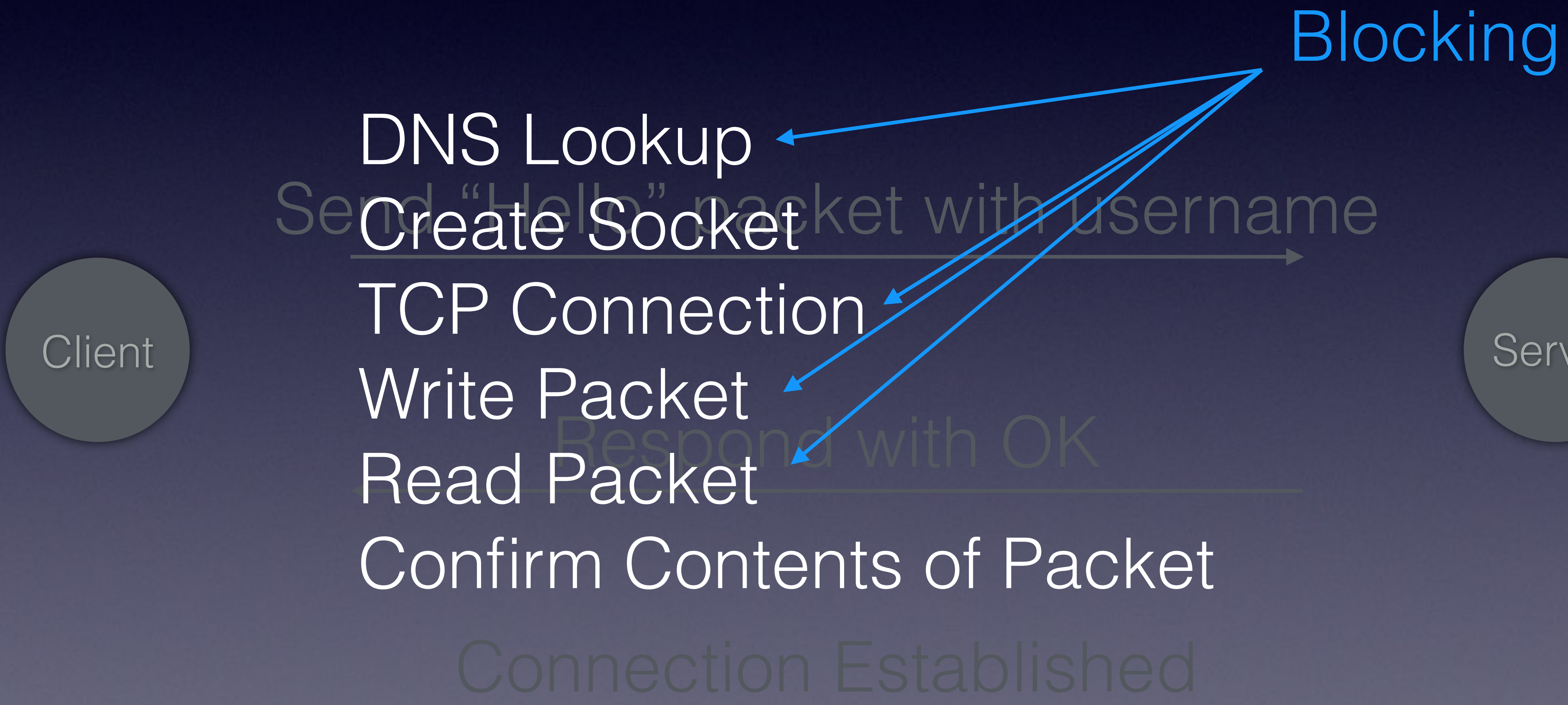
DNS Lookup
Create Socket
TCP Connection
Write Packet
Read Packet
Confirm Contents of Packet

Networking Example

“Just” a “Simple” Handshake

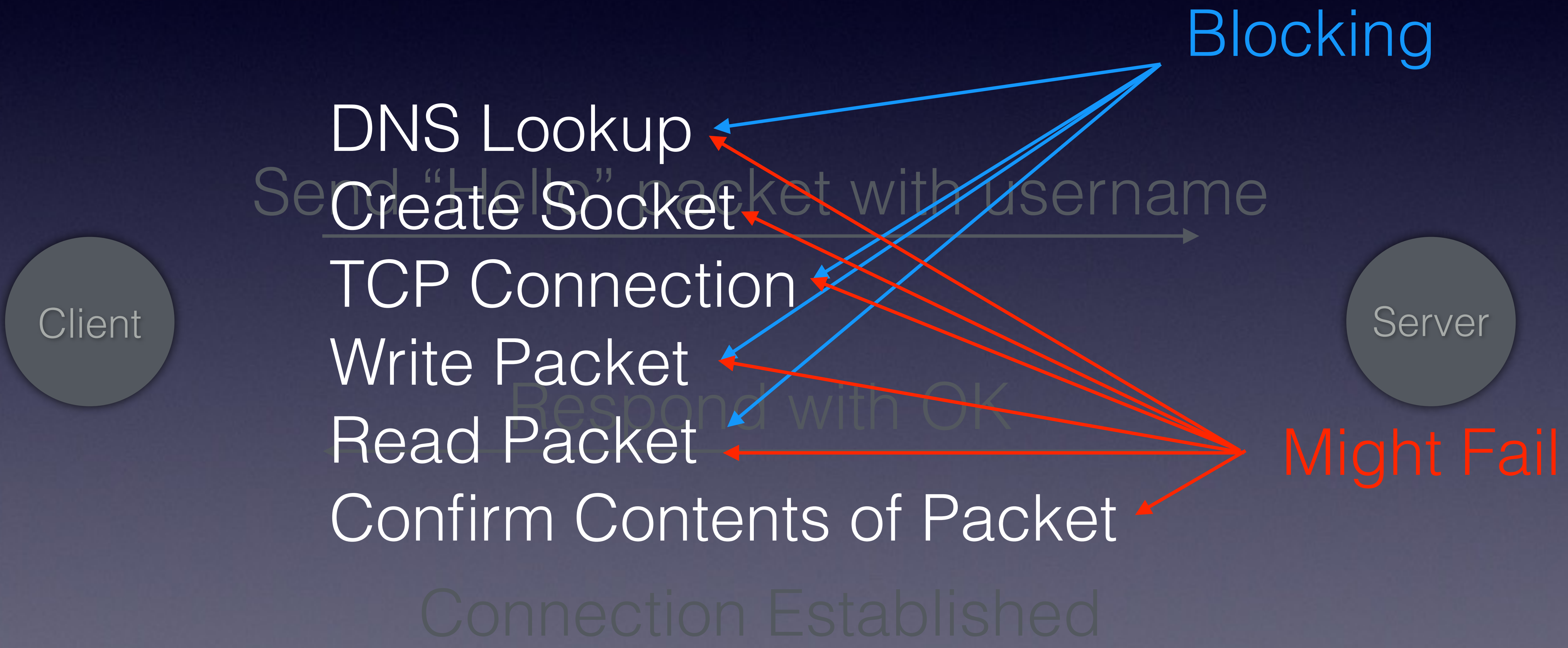
Blocking

DNS Lookup
Create Socket
TCP Connection
Write Packet
Read Packet
Confirm Contents of Packet



Networking Example

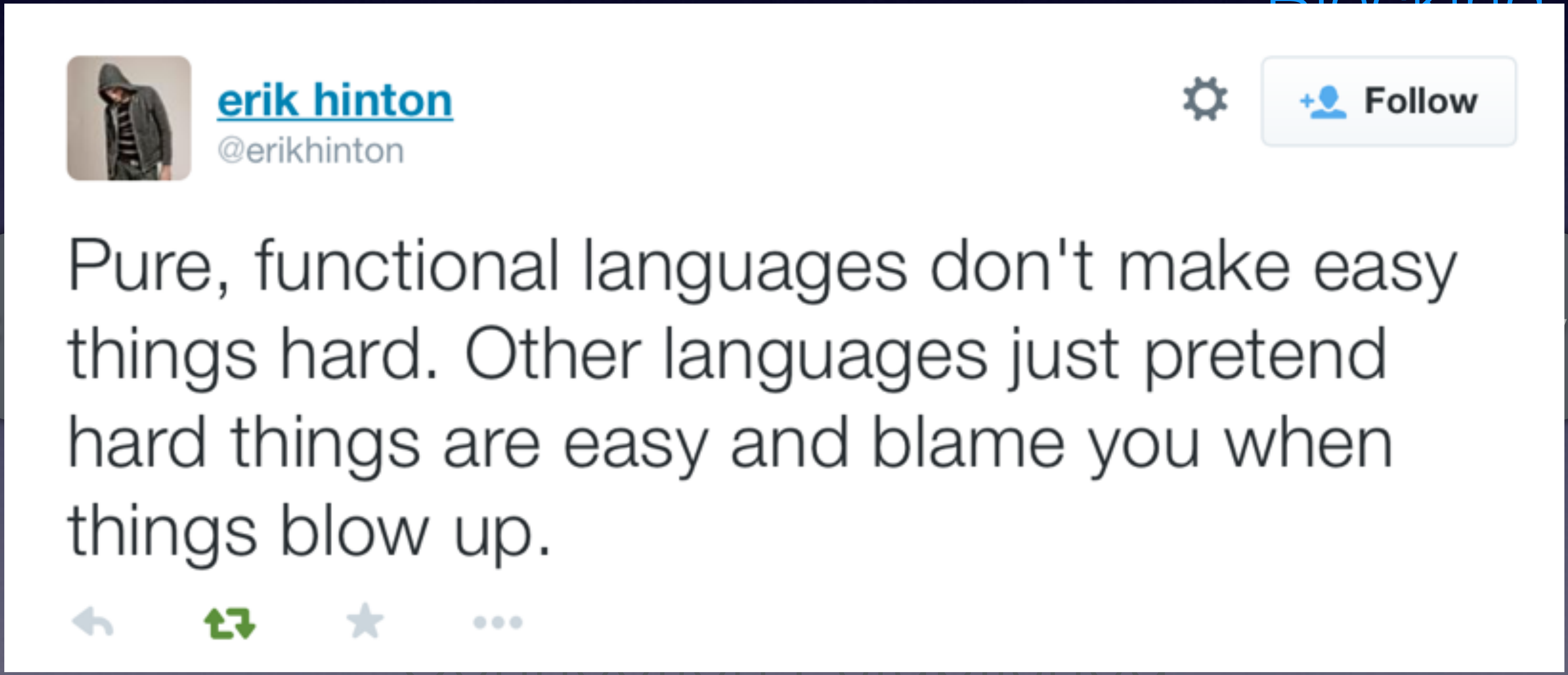
“Just” a “Simple” Handshake



DNS Lookup
Create Socket
TCP Connection
Write Packet
Read Packet
Confirm Contents of Packet

Networking Example

"Just" a "Simple" Handshake



Pure, functional languages don't make easy things hard. Other languages just pretend hard things are easy and blame you when things blow up.

Fail

Handshaking with Monads

```
// Connect to host over TCP
func connect(host: String) -> Deferred<Result<TCPCommSocket>>

// Parse raw socket data into a Message
func parseMessage(data: NSData) -> Result<Message>

// Confirm that `message` is a valid handshake response
func confirmHandshakeResponse(message: Message) -> Result<()>
```

Handshaking with Monads

```
func connectAndHandshake(host: String) ->
    Deferred<Result<TCPCommSocket>> {

// 1. Connect
return connect(host).bind {
  resultToDeferred($0) { (socket: TCPCommSocket) in

    // 2. Send handshake packet
    socket.writeString(helloPacket).bind {
      resultToDeferred($0) { () in

        // 3. Read response
        socket.readData().map { (dataResult: Result<NSData>) in

          // 4. Parse and confirm server's response
          dataResult
              .bind(parseMessage)
              .bind(confirmHandshakeResponse)
              .map { socket }
} } } } } }
```

Conclusions

- Shamelessly steal ideas from other languages
- Use functional programming techniques when it makes sense
- Say goodbye and good riddance to NSError **

Resources

- https://github.com/bignerdranch/Result
- https://github.com/bignerdranch/Deferred
- https://github.com/bignerdranch/DeferredTCPSocket
- https://realworldocaml.org/