

1. Implicit Sharing

The sad story

Defensive Copying

Inefficiency

Race Conditions

Locks

More Inefficiency

Deadlock

Complexity

Bugs!

2. Inheritance All Up In Your Business

One superclass — choose well!

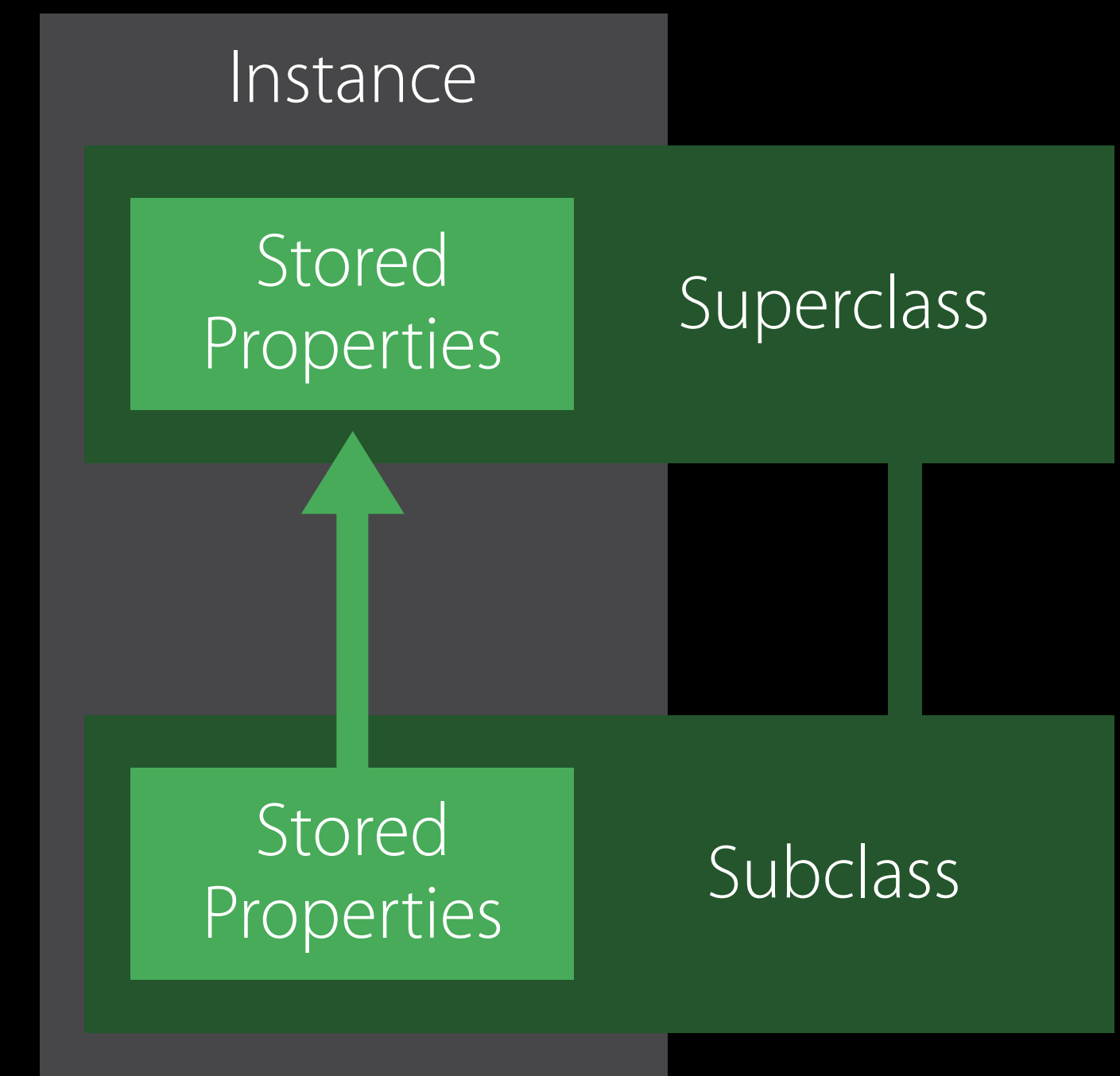
Single Inheritance weight gain

No retroactive modeling

Superclass may have stored properties

- You must accept them
- Initialization burden
- Don't break superclass invariants!

Know what/how to override (and when not to)



3. Lost Type Relationships


```
class Ordered {  
    func precedes(other: Ordered) -> Bool  
}  
  
func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {  
    var lo = 0, hi = sortedKeys.count  
    while hi > lo {  
        let mid = lo + (hi - lo) / 2  
        if sortedKeys[mid].precedes(k) { lo = mid + 1 }  
        else { hi = mid }  
    }  
    return lo  
}
```

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool {  
    }  
}
```

```
func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {  
    var lo = 0, hi = sortedKeys.count  
    while hi > lo {  
        let mid = lo + (hi - lo) / 2  
        if sortedKeys[mid].precedes(k) { lo = mid + 1 }  
        else { hi = mid }  
    }  
    return lo  
}
```

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool {  }  
}
```

```
func binarySearch(sortedKeys: [Ordered], forKey k: Ordered) -> Int {  
    var lo = 0, hi = sortedKeys.count  
    while hi > lo {  
        let mid = lo + (hi - lo) / 2  
        if sortedKeys[mid].precedes(k) { lo = mid + 1 }  
        else { hi = mid }  
    }  
    return lo  
}
```

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

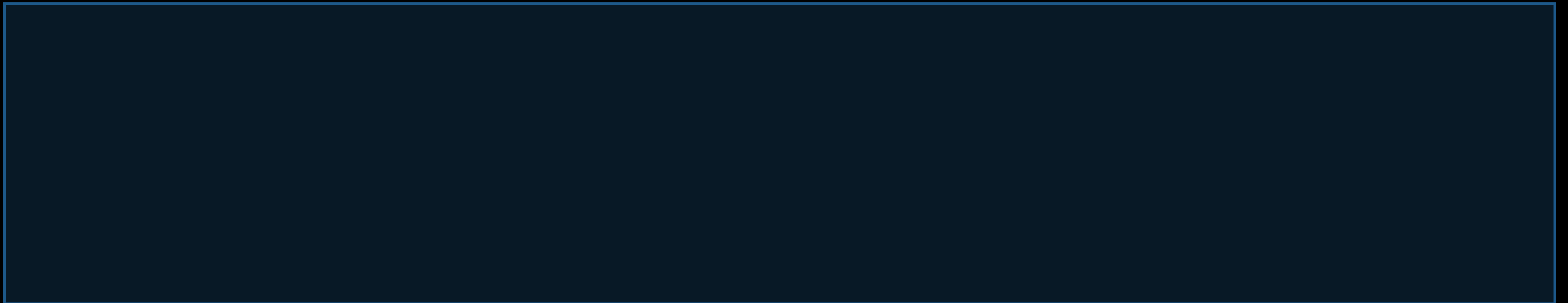


3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```


3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```



3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return value < other.value  
    }  
}
```

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return value < other.value  
    }  
}
```

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return value < other.value  
    }  
}
```

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return value < other.value  
    }  
}
```

'Ordered' does not have a member
named 'value'

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return value < other.value  
    }  
}
```

'Ordered' does not have a member
named 'value'

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Label : Ordered { var text: String = "" ... }
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return value < other.value  
    }  
}
```

'Ordered' does not have a member
named 'value'

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Label : Ordered { var text: String = "" ... }
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return value < other.value  
    }  
}
```


3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Label : Ordered { var text: String = "" ... }
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return value < (other as! Number).value  
    }  
}
```

3. Lost Type Relationships

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Label : Ordered { var text: String = "" ... }
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return value < (other as! Number).value  
    }  
}
```

as! ASubclass



A sign that a type relationship was lost
Usually due to using classes for abstraction

A Better Abstraction Mechanism

Supports value types (and classes)

Supports static type relationships (and dynamic dispatch)

Non-monolithic

Supports retroactive modeling

Doesn't impose instance data on models

Doesn't impose initialization burdens on models

Makes clear what to implement

Swift Is a Protocol-Oriented
Programming Language

Start with a Protocol

Your first stop for new abstractions

Starting Over with Protocols

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```

Starting Over with Protocols

```
class Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```


Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```

Starting Over with Protocols

error: protocol methods may not have bodies

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool { fatalError("implement me!") }  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```

error: method does not override any
method from its superclass

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool  
}
```

```
class Number : Ordered {  
    var value: Double = 0  
    override func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool  
}
```

```
struct Number : Ordered {  
    var value: Double = 0  
    func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool  
}
```

```
struct Number : Ordered {  
    var value: Double = 0  
    func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool  
}
```

```
struct Number : Ordered {  
    var value: Double = 0  
    func precedes(other: Ordered) -> Bool {  
        return self.value < (other as! Number).value  
    }  
}
```


Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool  
}
```

```
struct Number : Ordered {  
    var value: Double = 0  
    func precedes(other: Number) -> Bool {  
        return self.value < other.value  
    }  
}
```

Starting Over with Protocols

```
protocol Ordered {
```

```
    func precedes(other: Ordered) -> Bool
```

```
}
```

```
struct Number : Ordered {
```

```
    var value: Double = 0
```

```
    func precedes(other: Number) -> Bool {
```

```
        return self.value < other.value
```

```
    }
```

```
}
```

protocol requires function 'precedes' with type '(Ordered) -> Bool'
candidate has non-matching type '(Number) -> Bool'

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Ordered) -> Bool  
}
```

```
struct Number : Ordered {  
    var value: Double = 0  
    func precedes(other: Number) -> Bool {  
        return self.value < other.value  
    }  
}
```

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Self) -> Bool  
}
```

```
struct Number : Ordered {  
    var value: Double = 0  
    func precedes(other: Number) -> Bool {  
        return self.value < other.value  
    }  
}
```

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Self) -> Bool  
}
```

"Self" requirement

```
struct Number : Ordered {  
    var value: Double = 0  
    func precedes(other: Number) -> Bool {  
        return self.value < other.value  
    }  
}
```

Starting Over with Protocols

```
protocol Ordered {  
    func precedes(other: Self) -> Bool  
}  
  
struct Number : Ordered {  
    var value: Double = 0  
    func precedes(other: Number) -> Bool {  
        return self.value < other.value  
    }  
}
```