Introduction to Object-oriented Programming

What, Why, and a Bit of How

Author: Torey Hickman

Phase 1: Day 6

Intro to OOP

- What is OOP
- Benefits
- How to basics

Procedural Programming

- We've been writing in a procedural style

Procedural Programming

- We've been writing in a procedural style
  - Variables hold data
  - Methods perform tasks
  - Code steps through the methods

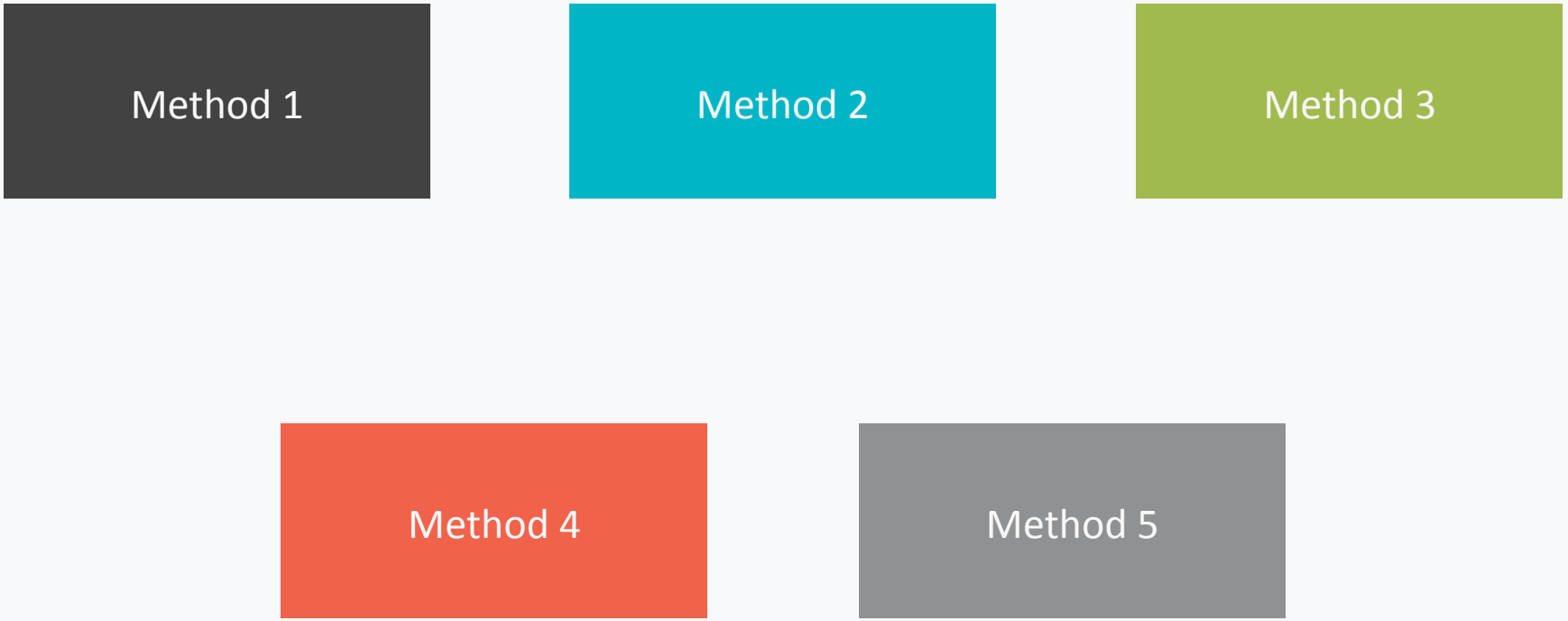
Procedural Programming

```
def birthday(birthday_girl)
  birthday_girl[:age] += 1
end

def die(mortal)
  mortal[:alive] = false
end

torey = { age: 33,
          alive: true }
100.times { birthday(torey) }
die(torey)
```

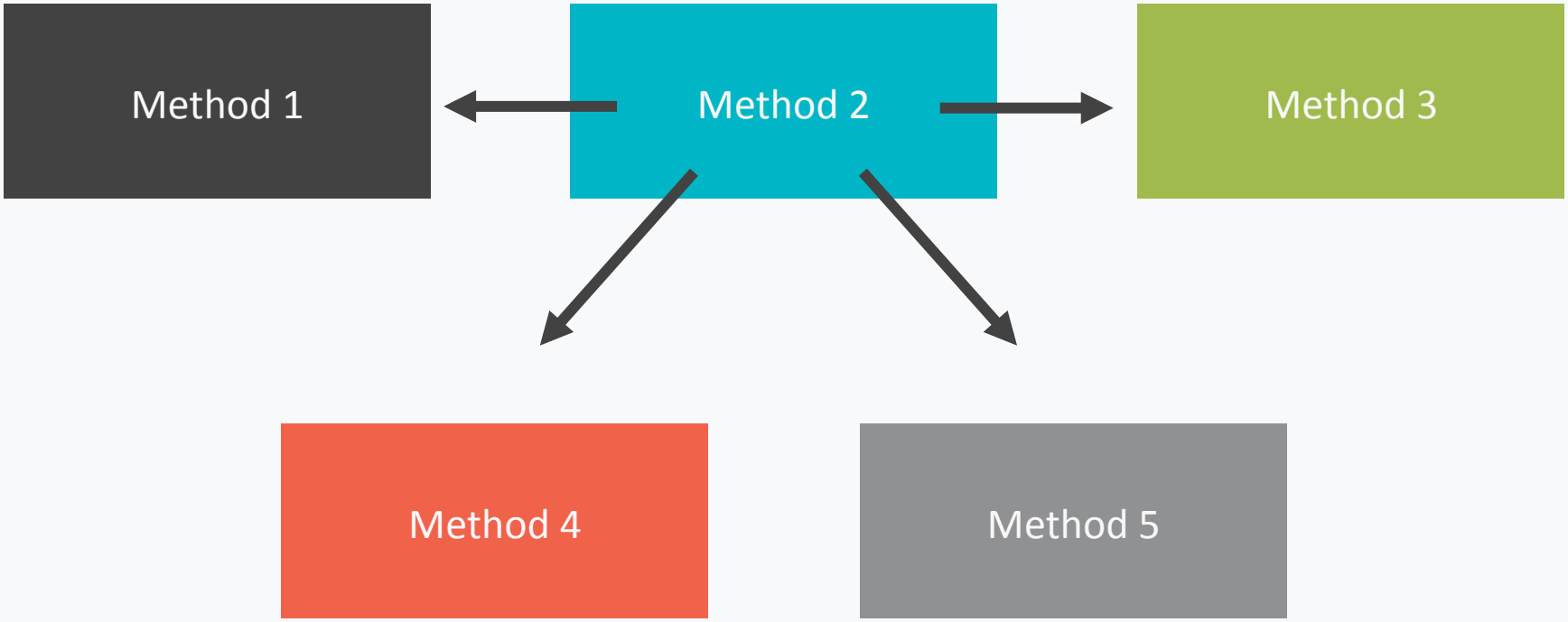
Procedural Programming

- No link between methods and data
- Open season on calling methods

Procedural Programming

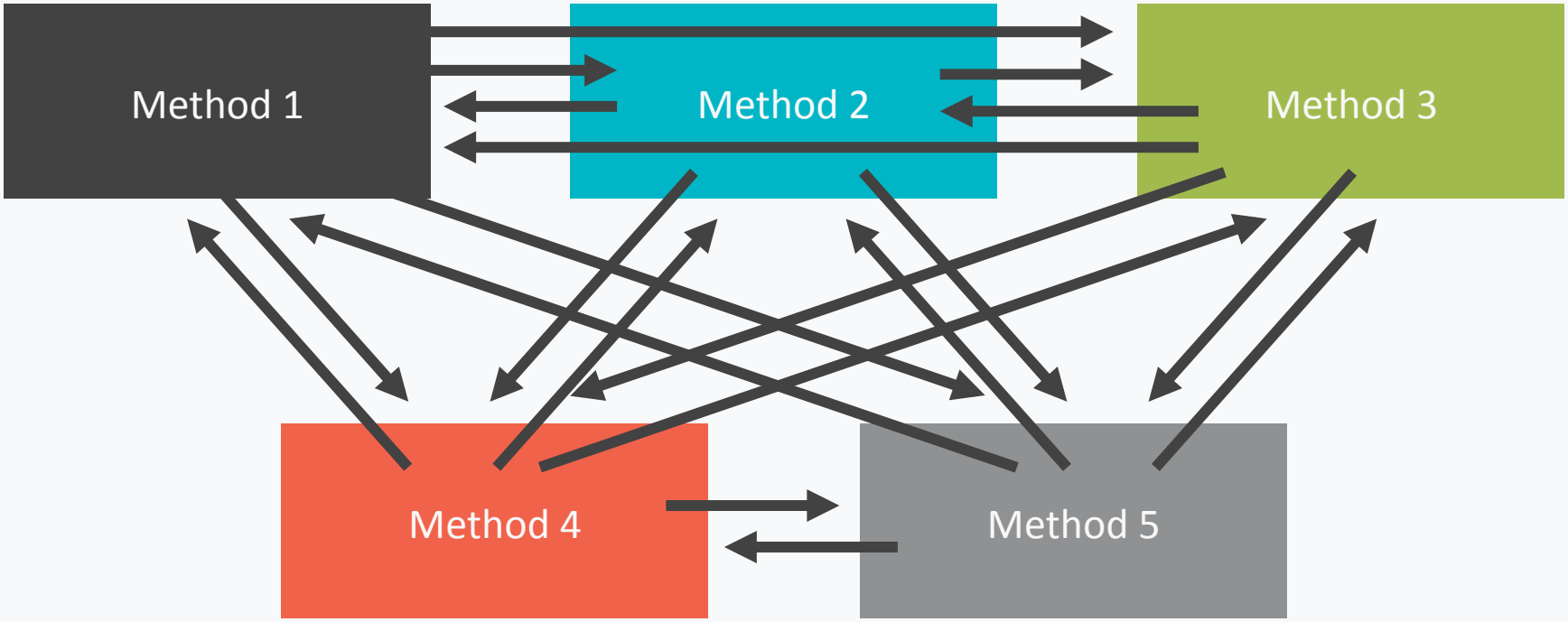


Procedural Programming



passing data as necessary

Procedural Programming



passing data as necessary

Object-oriented Programming

- Alternative style for organizing code

Object-oriented Programming

- Alternative style for organizing code
  - Encapsulate data and related behavior
  - Limit interfaces

Encapsulation

- Classes: data and behavior
- Modules: behavior

Classes: Data and Behavior

- Encapsulate related data and behavior
- Instantiate objects of the class

Procedural & OO Programming

```
def birthday(birthday_girl)
  birthday_girl[:age] += 1
end

def die(mortal)
  mortal[:alive] = false
end

torey = { age: 33,
          alive: true }

birthday(torey)
die(torey)
```

```
class Person
  def initialize(age, alive = true)
    @age = age
    @alive = alive
  end

  def birthday
    self.age = age + 1
  end

  def die
    self.alive = false
  end

private
  attr_accessor :age
  attr_writer   :alive
end

torey = Person.new(33)
torey.birthday
torey.die
```

Classes: Data and Behavior

- Person class
  - Data: age and alive status
  - Behavior: have a birthday, die

Modules: Behavior

- Encapsulate related behaviors
- Independent, or add behavior to classes

Modules: Add Behaviors

```
module LifeCycle
  def birthday
    self.age = age + 1
  end

  def die
    self.alive = false
  end
end
```

Modules: Add Behaviors

```
module LifeCycle
  def birthday
    self.age = age + 1
  end

  def die
    self.alive = false
  end
end
```

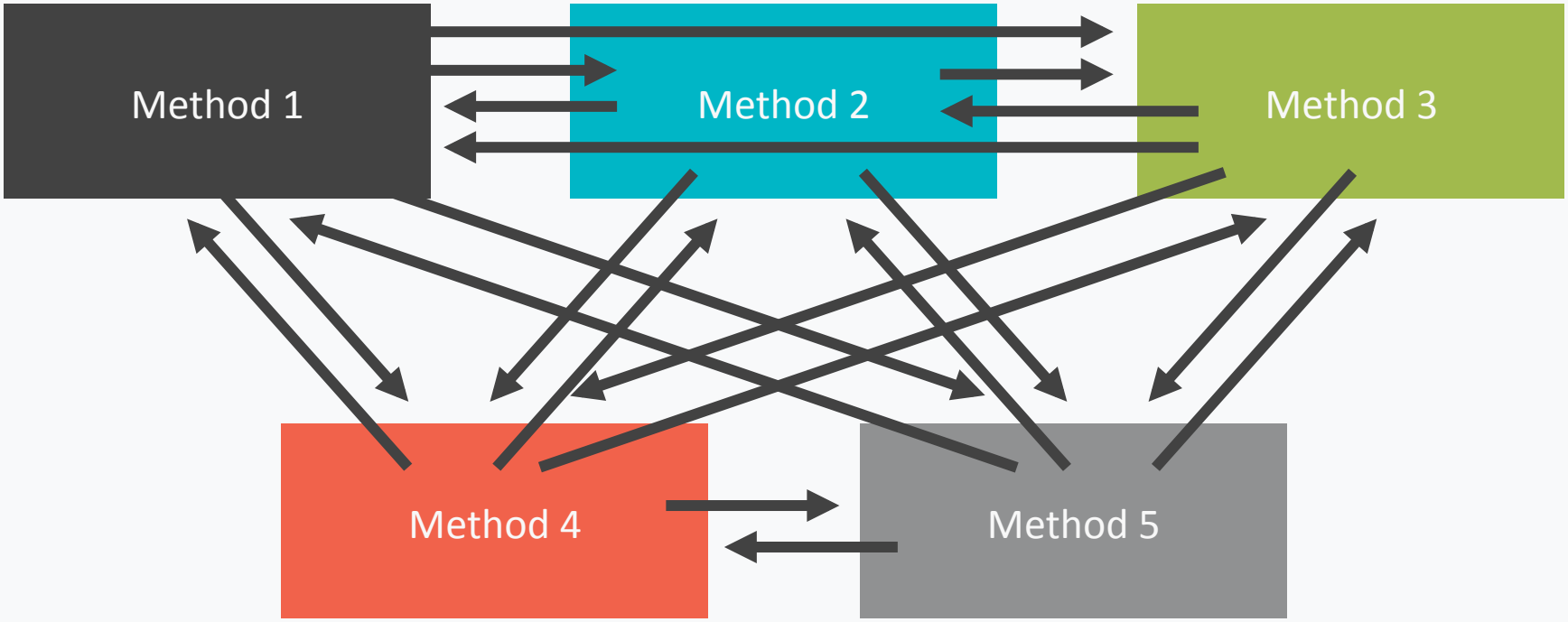
```
class Person
  include LifeCycle

  def initialize(age, alive)
    @age = age
    @alive = alive
  end

private
  attr_accessor :age
  attr_writer   :alive
end

torey = Person.new(33, true)
```

Modules: Add Behaviors

```
module LifeCycle
  def birthday
    self.age = age + 1
  end

  def die
    self.alive = false
  end
end
```

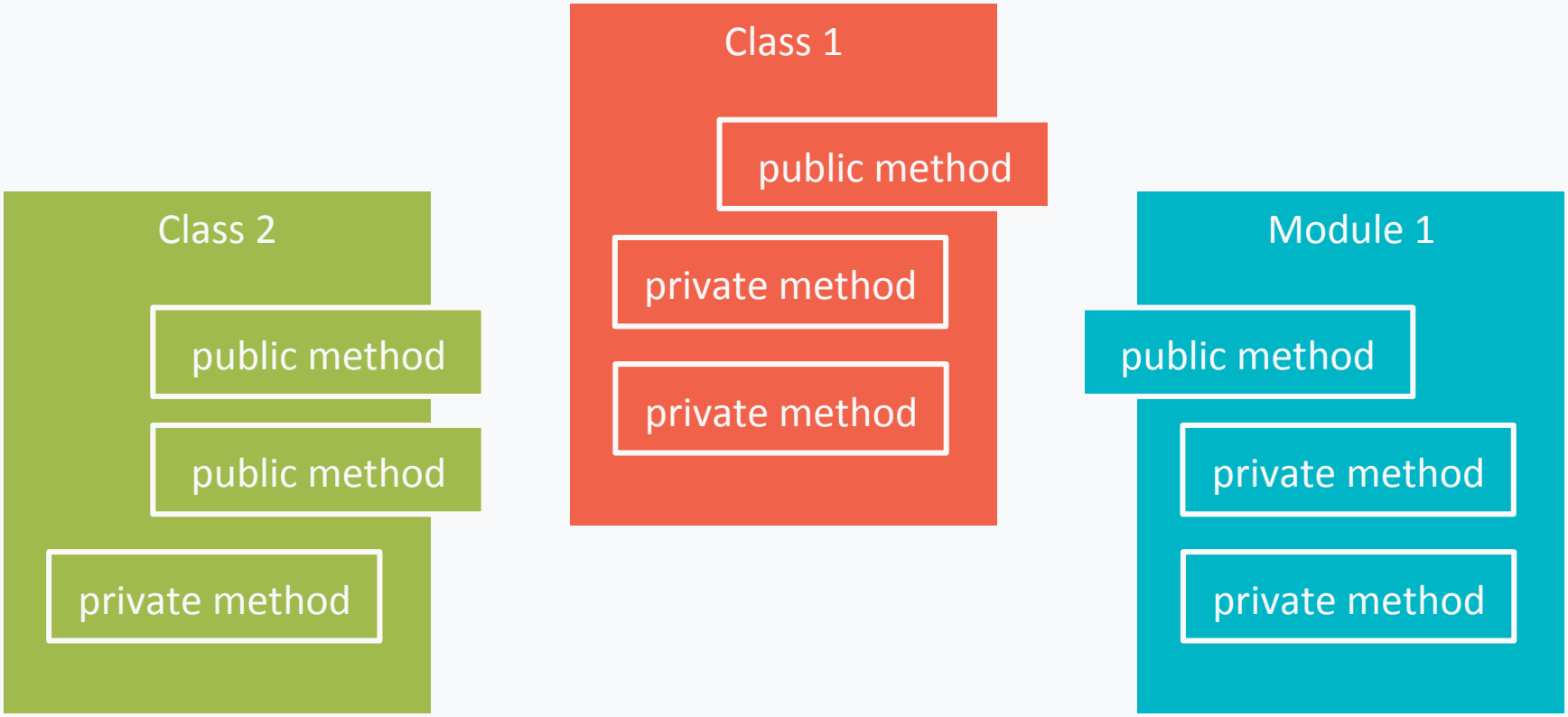
```
class Person
  include LifeCycle

  # …
end

class Dog
  include LifeCycle

  # …
end

torey = Person.new(33, true)
tenley = Dog.new(1, true)
```

Public Interface: Procedural

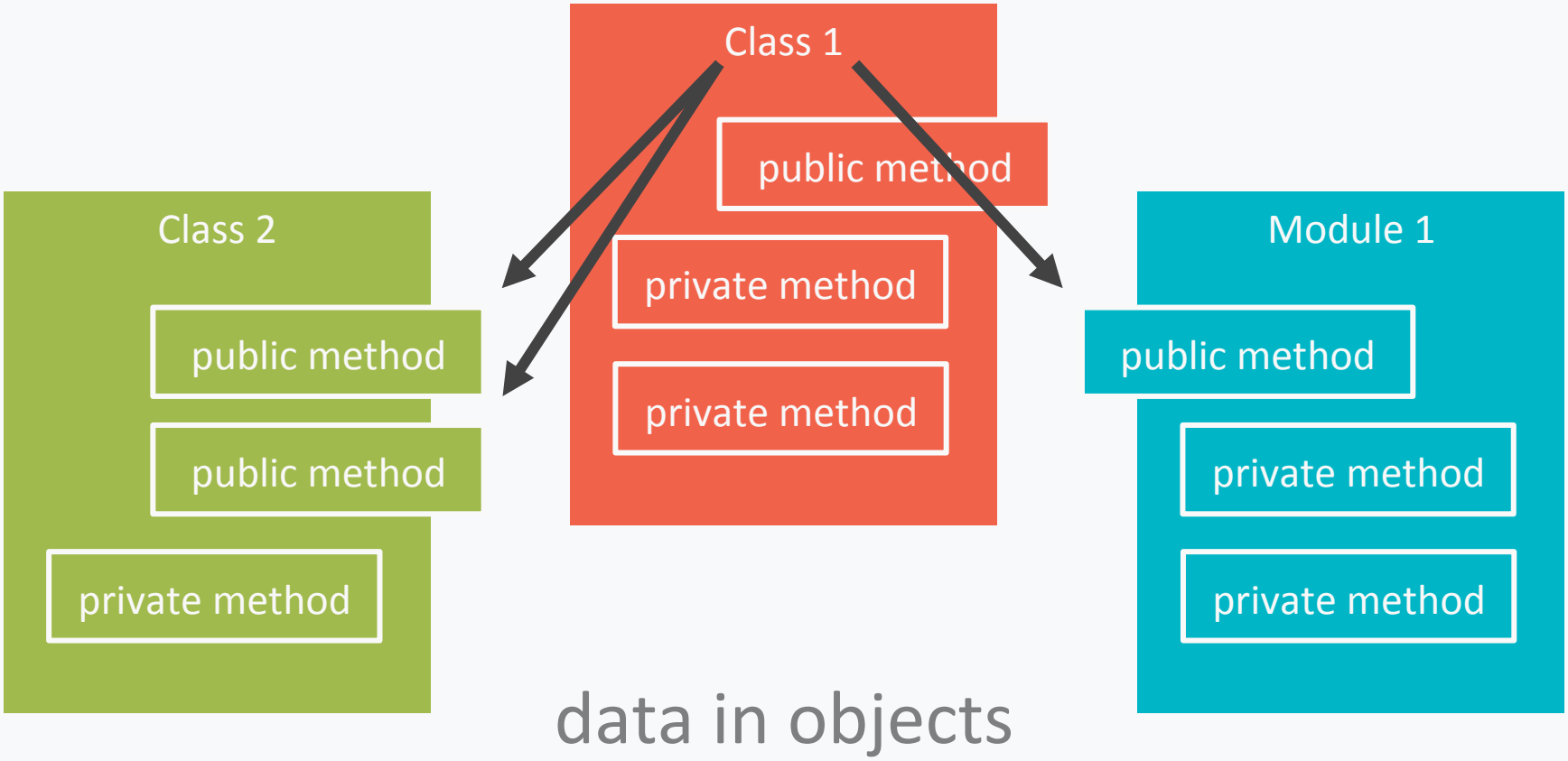


passing data as necessary

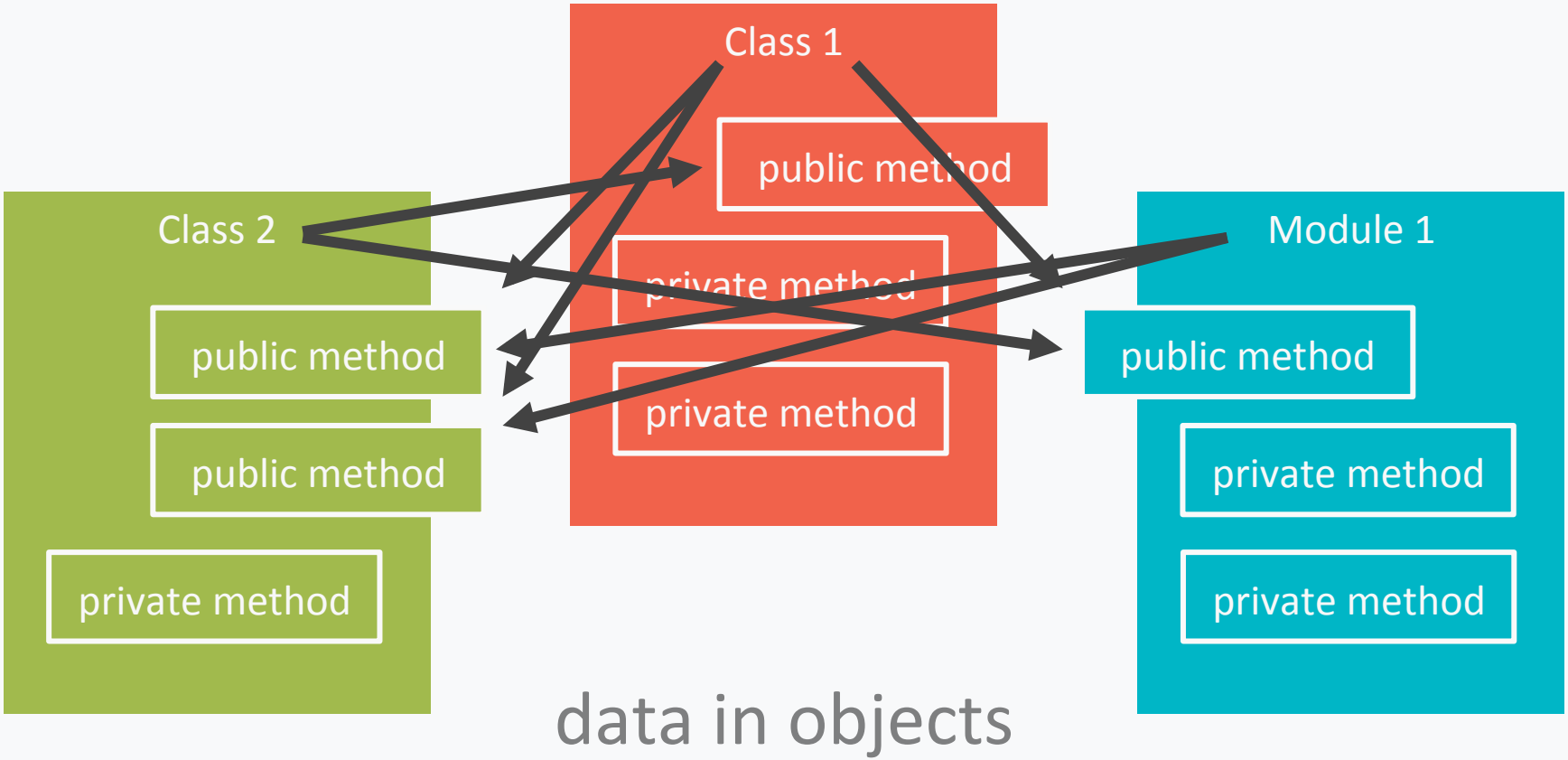
Public Interface: OOP



Public Interface



Public Interface: OOP



Limiting the Interface

```
class Person
  attr_accessor :age
  attr_writer   :alive

  def initialize(age, alive = true)
    @age = age
    @alive = alive
  end

  def birthday
    self.age = age + 1
  end

  def die
    self.alive = false
  end
end
```

```
class Person
  def initialize(age, alive = true)
    @age = age
    @alive = alive
  end

  def birthday
    self.age = age + 1
  end

  def die
    self.alive = false
  end

private
  attr_accessor :age
  attr_writer   :alive
end
```

Benefits: Why OOP

- Easier to change and maintain
- Reuse (libraries)
- Ease of testing