Ruby

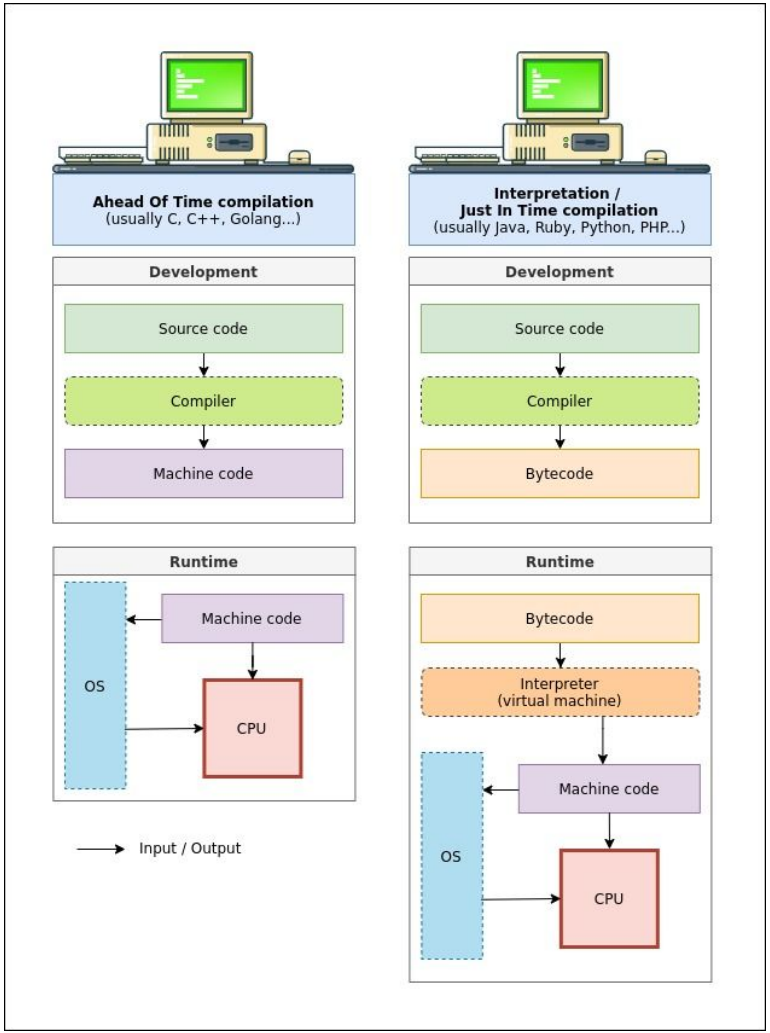
Ruby Programming Language

Ruby Programming Language

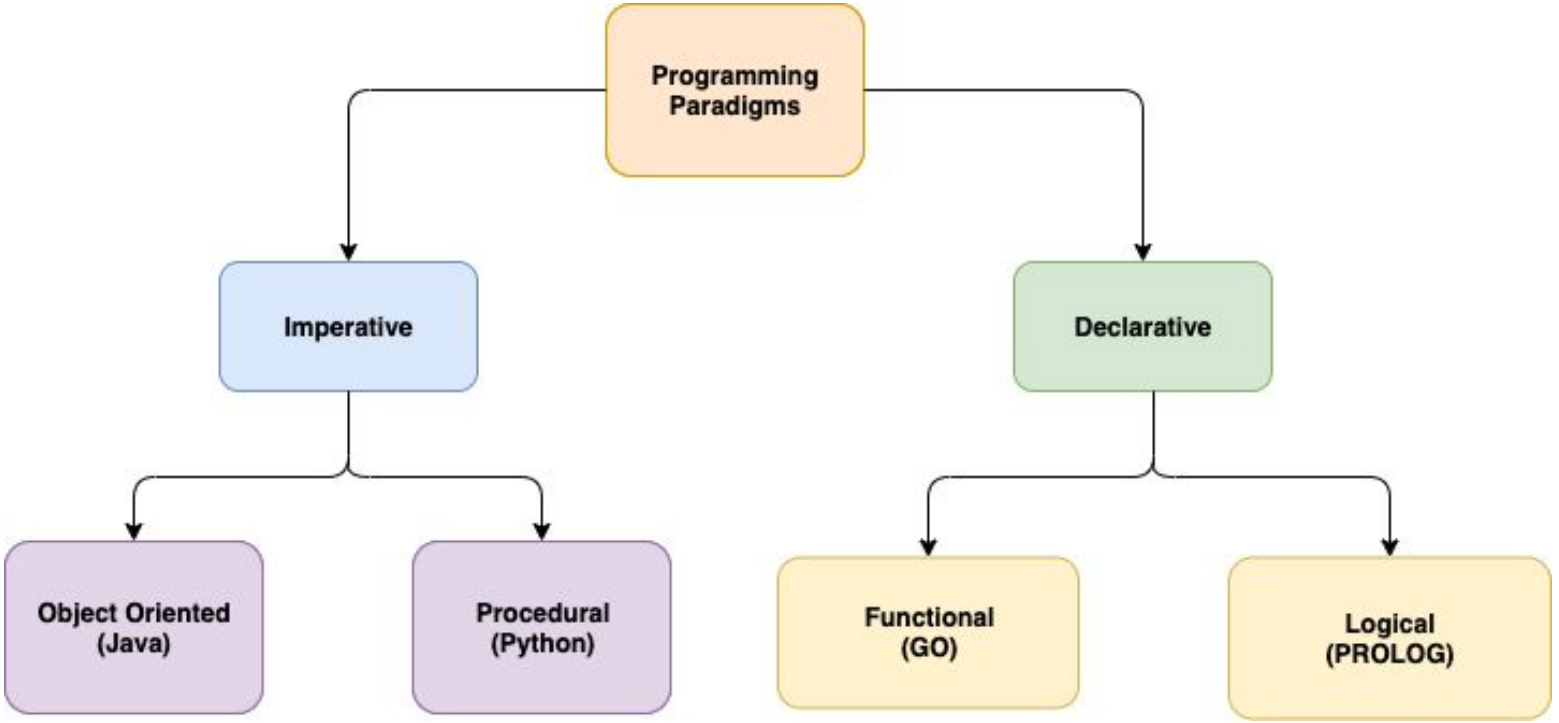
- Interpreted
- High-level
- Dynamically typed
- Imperative
- Multi-paradigm
  - Procedural
  - Functional
  - Object-oriented

Interpreted vs Compiled

- Translate source code into some efficient intermediate representation or object code and immediately execute that
- Translate source code from a high-level programming language to a low-level programming language that can be executed



Programming Paradigms



Imperative vs Declarative

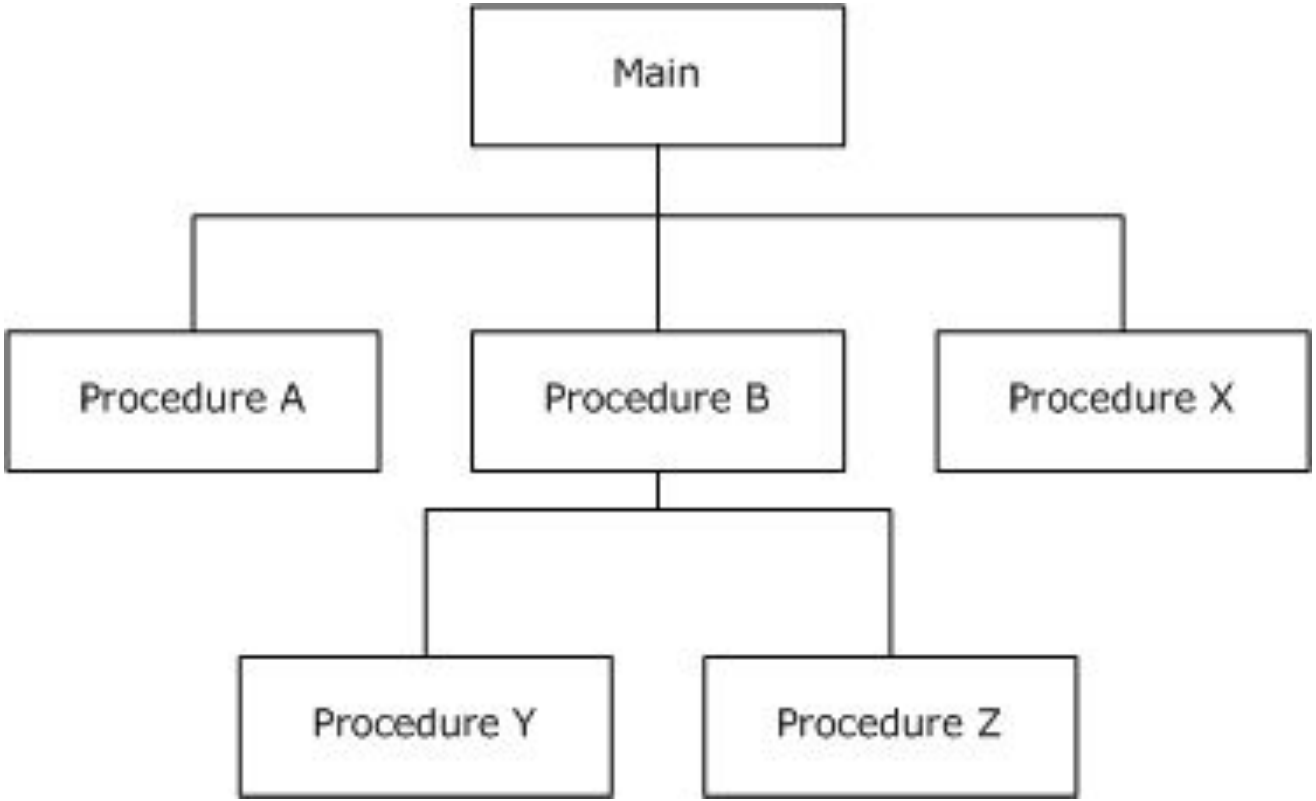
- Everything is an expression
- Instructions
- The system is stupid, you are smart

```
i = 0
sum = 0
while i < 10 do
  sum = sum + i
end
puts sum
```

- Express logic statements
- Outcome
- The system is smart, you don’t care

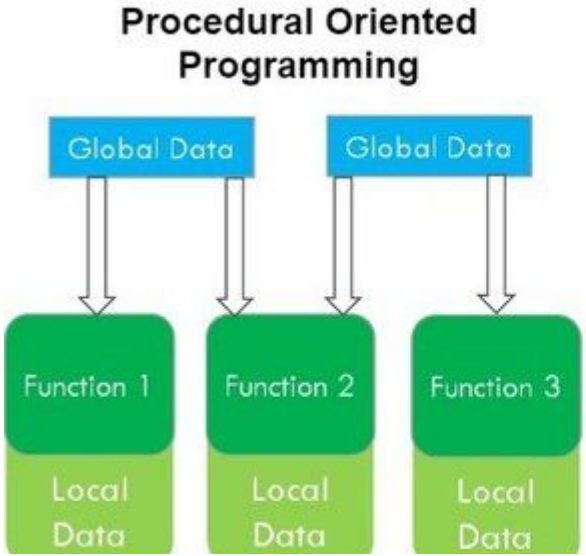
```
puts 10.times.sum
```

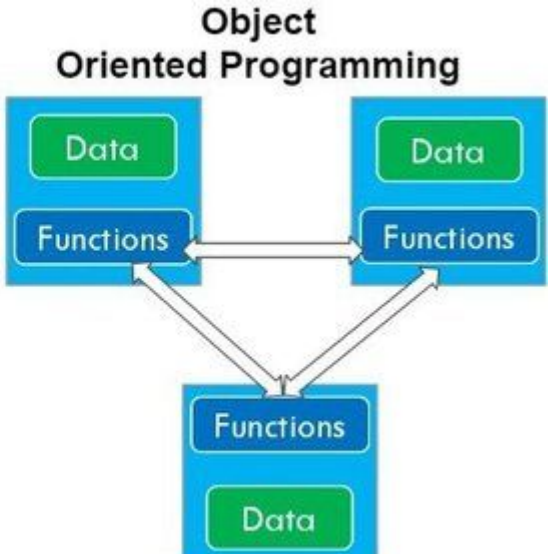
Procedural Programming



Object-Oriented Programming

- Abstraction
  - Class
  - Object
  - Interface
- Polymorphism
- Inheritance
- Encapsulation

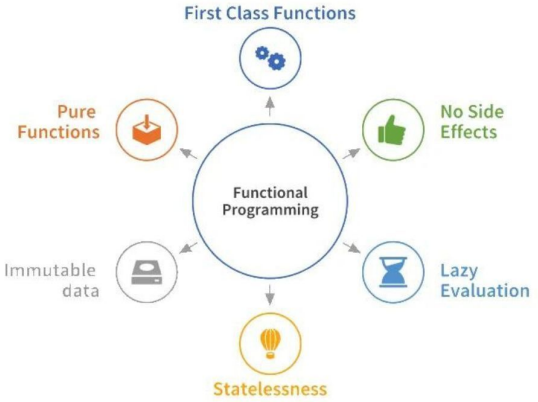




Functional Programming

- Function as first-class citizen
- Higher-order function
- Pure function
  - Immutable states
  - Deterministic

VS



Procedure vs Function vs Method

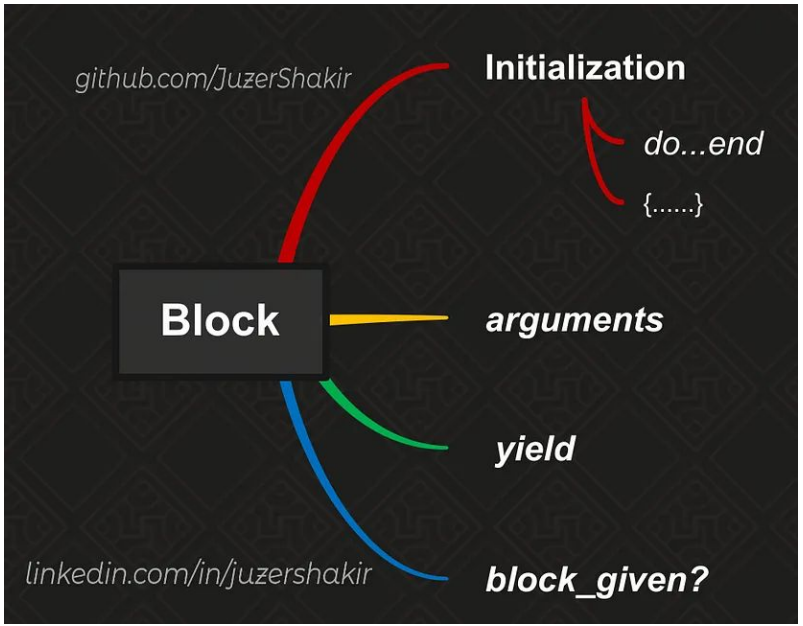
- Procedure DOES something
- Function computes the result by the given arguments
- Method is associated with an object

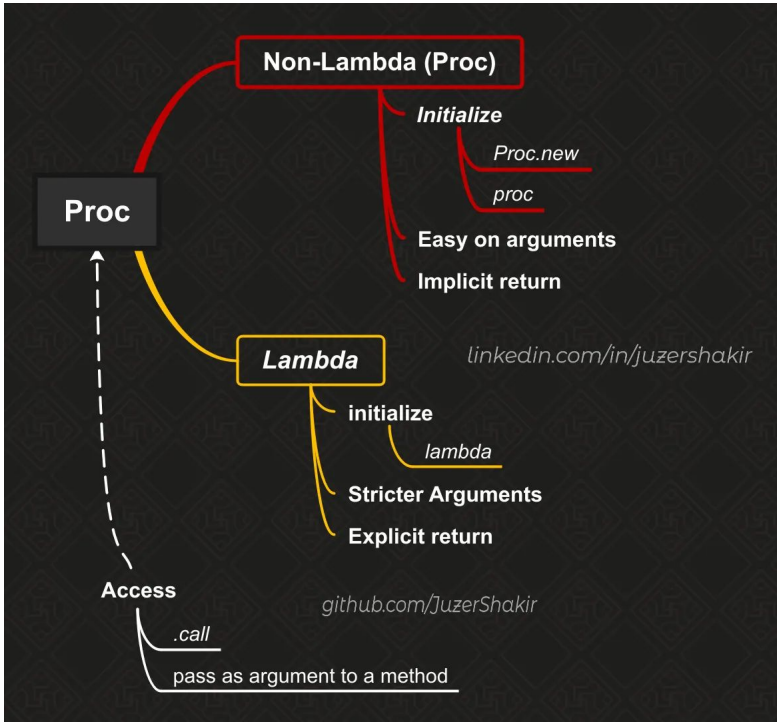
```ruby
def print_sum_of_values
  n = 10
  sum = n.times.sum
  print sum
end

print_sum_of_values
```

```ruby
def sum(n)
  return n.times.sum
end

print sum(10)
```

```ruby
class Number
  def initialize(val)
    @val = val
  end

  def sum
    n.times.sum
  end
end

number = Number.new(10)
print number.sum
```

Block vs Proc vs Lambda






```ruby
## block
10.times { |x| puts x }
10.times do |x|
  puts x
end
```

```ruby
## proc
a = proc { |x| puts x}

10.times(&a)
```

```ruby
## lambda
a = lambda { |x| puts x }
a = ->(x) { puts x }

10.times(&a)
```

Class

- Constructor
- Destructor
- Method
- Getter/Setter
- Access Specifier
- Instance Variable
- Class Variable

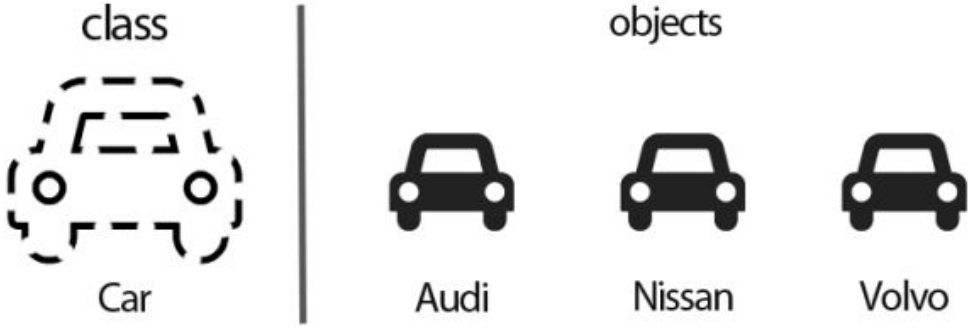
- Method Naming

```ruby
class Square
  class InvalidSideError < StandardError; end

  def initialize(side)
    @side = side
    raise InvalidSideError unless valid_side?
  end

  def area
    side * side
  end

  attr_reader :side

  private

  attr_writer :side

  def valid_side?
    side.is_a?(Numeric) && side.positive?
  end
end

square = Square.new(15) #<Square:0x00007f7a296e61f8 @side=15>
square.side # 15
square.area # 225
square.valid_side? # NoMethodError: private method `valid_side?'
                   # called for #<Square:0x00007f7a279ba840 @side=15>
Square.new(-1) # Square::InvalidSideError: Square::InvalidSideError
```

Object aka Instance

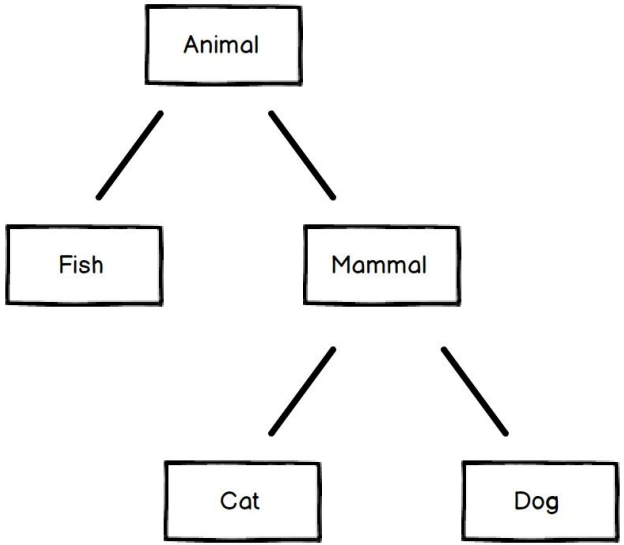
In ruby everything is object except:

- Code Blocks
- Methods
- Operators



Class Inheritance

- Inherit methods and variables
- Call `super` methods
- Override parent methods



Module

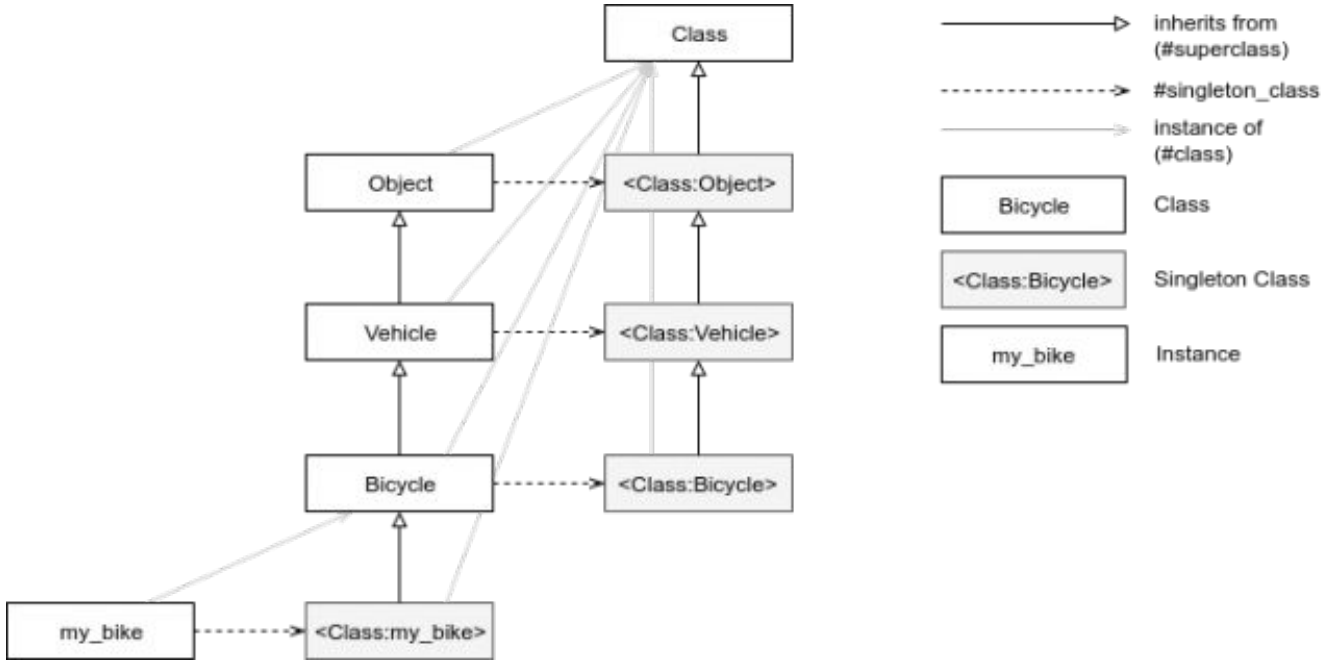
- No instance, similar to interface
- Mixin pattern
- Can be included, prepended and extended

```ruby
class MyArray
  include MyEnumerable

  def initialize(arr); @arr = arr; end
  def size; @arr.size; end
  def each(&block); @arr.each(&block); end
  def first; @arr.first; end
end
```

```ruby
module MyEnumerable
  def each(&block); raise NotImplementedError; end
  def size; raise NotImplementedError; end
  def first; raise NotImplementedError; end

  def each_with_index(&block)
    idx = 0
    each do |el|
      yield el, idx
      idx += 1
    end
  end

  def reduce(acc = nil, &block)
    each do |el|
      acc = acc ? yield(acc, el) : el
    end
    acc
  end

  def map(&block)
    result = Array.new(size)

    each_with_index { |el, idx| result[idx] = yield el }
    result
  end
end
```

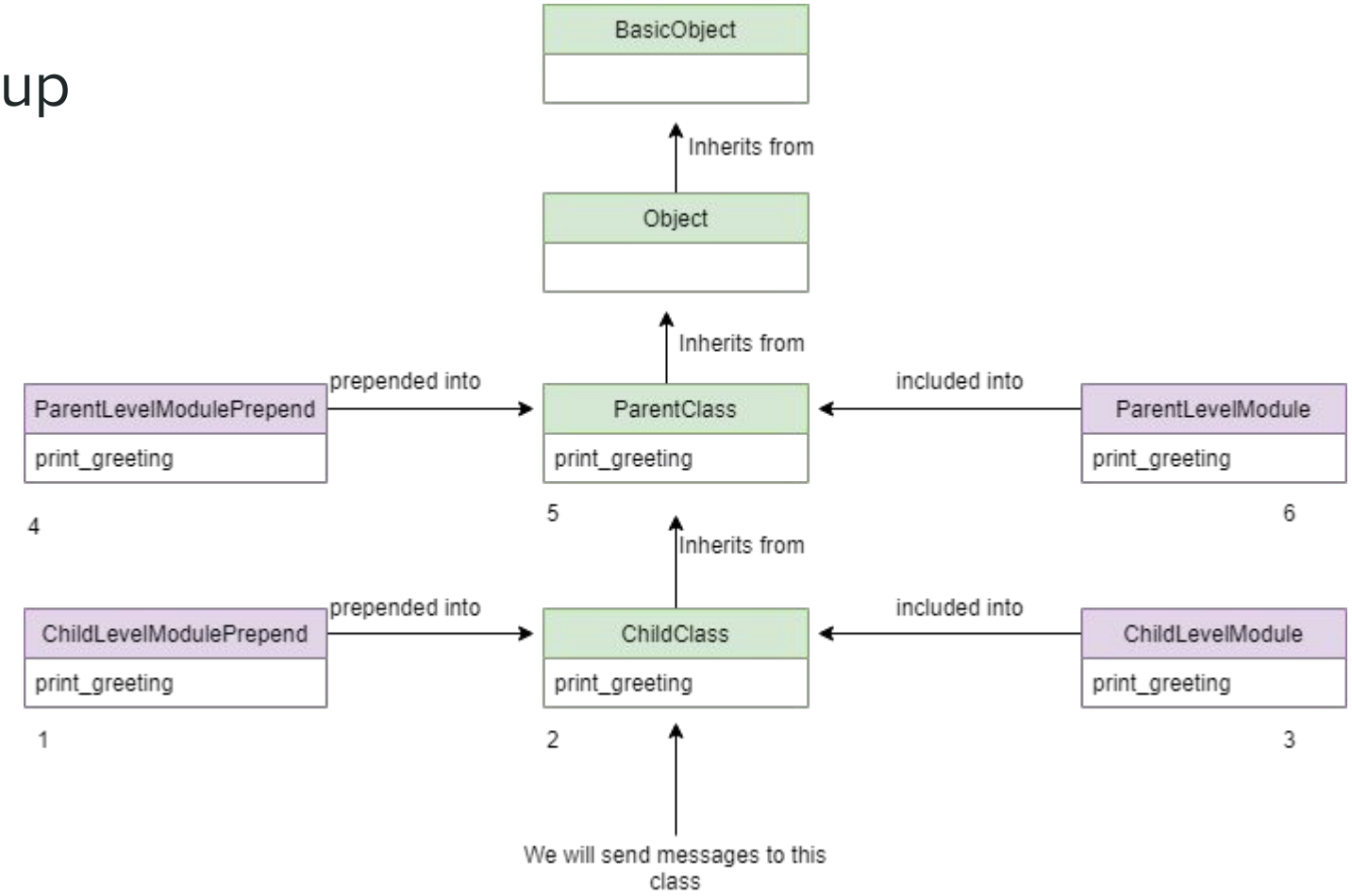
Singleton Class

- Singleton pattern implementation
- Copy of a class for each instance
- Class also has Singleton Class!
- Class methods are instance methods of Singleton Class



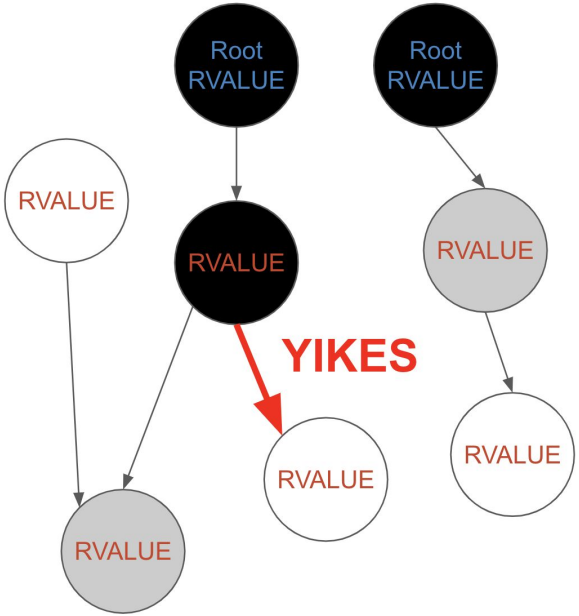
Method Lookup

- method_missing
- instance_eval
- class_eval
- send
- public_send
- define_method



Garbage Collection

- Allocation phase
- Object creation
- Memory management
- Page allocation
- Heap expansion
- Garbage collection trigger
  - Tri-Color marking
  - Sweeping
  - Generations
  - Stops execution



Thread vs Process

Ruby Thread

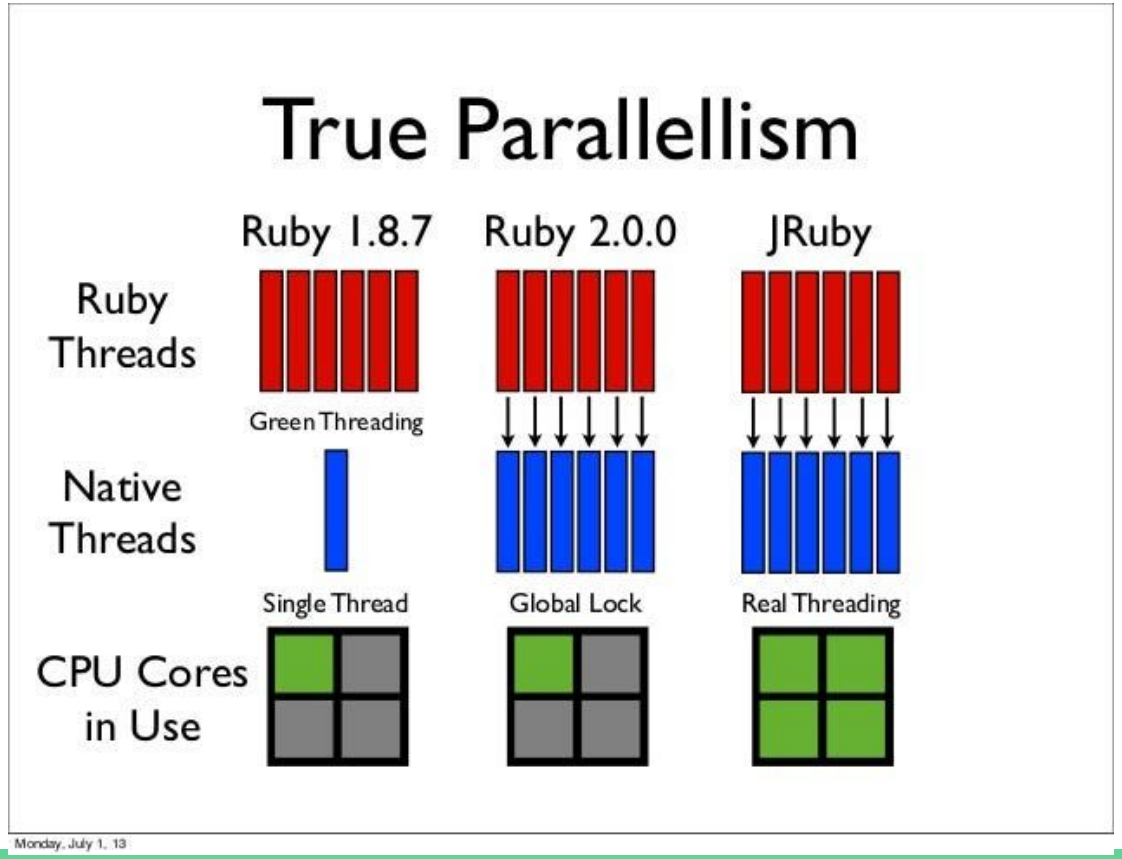
- Managed independently
- Scheduled by VM
- “Sleeps” on I/O operations

```ruby
time = Time.now
a = Thread.new do
  sleep(2)
  puts "A finished after #{Time.now - time}"
end

b = Thread.new do
  sleep(1)
  puts "B finished after #{Time.now - time}"
end

a.join
b.join
puts "All threads finished after #{Time.now - time}"
```

```
B finished after 1.000346246
A finished after 2.000318183
All threads finished after 2.000688865
```

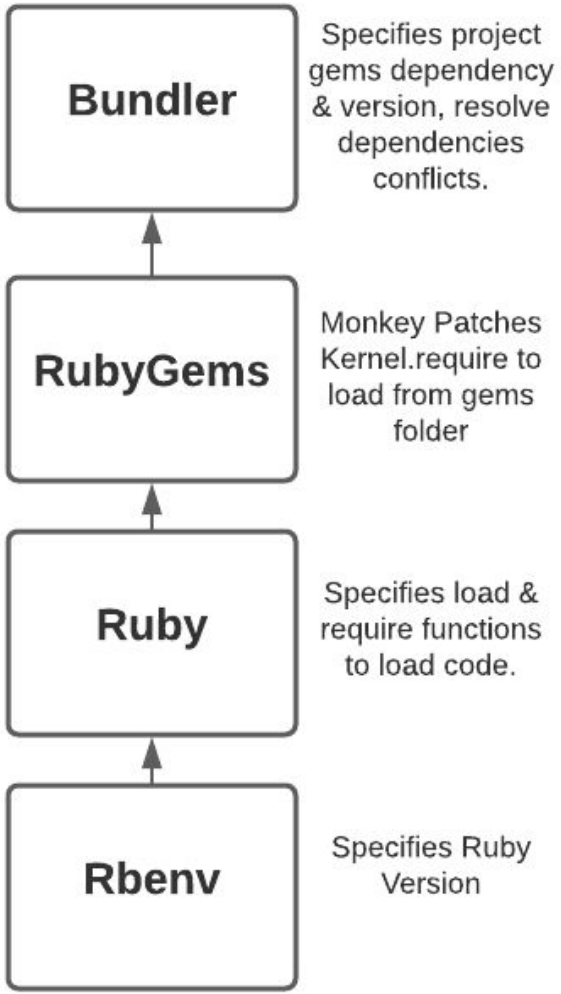
Parallelism and Concurrency

- Thread
- Fiber
- Ractor (Ruby 3.0)
- Process



Gem and Bundler

- Ruby can import files
- Prebuilt modules are called gems
- Bundler manages gems for a project



Thank you for your attention!