PROGRAMMING PARADIGMS

WHICH ONE IS THE BEST?

@akashivskyy

iOS

PROGRAMMING PARADIGMS

WAY OF LOOKING AT CONTROL FLOW AND EXECUTION OF A PROGRAM

1. OBJECT-ORIENTED PROGRAMMING

PROGRAM IS DEFINED BY
OBJECTS WHICH COMBINE
STATE AND BEHAVIOR

3 ASSUMPTIONS

1. ABSTRACTION
2. ENCAPSULATION
3. INHERITANCE

1. ABSTRACTION

2. ENCAPSULATION

3. INHERITANCE

```swift
protocol Shape {
  var area: Double
}

func printShapeArea(shape: Shape) {
  println("area = \(shape.area)")
}
```

```swift
struct Square: Shape {
  let side: Double
  let area: Double {
    return side * side
  }
}

printShapeArea(Square(side: 4)) // 16.0
```

```swift
struct Circle: Shape {
  var radius: Double
  var area: Double {
    return M_PI * radius * radius
  }
}

printShapeArea(Circle(radius: 2)) // 12.56
```

```swift
struct Plane: Shape {
  var area: Double {
    return Double.infinity
  }
}

printShapeArea(Plane()) // infinity
```

~~1. ABSTRACTION~~

2. ENCAPSULATION

3. INHERITANCE

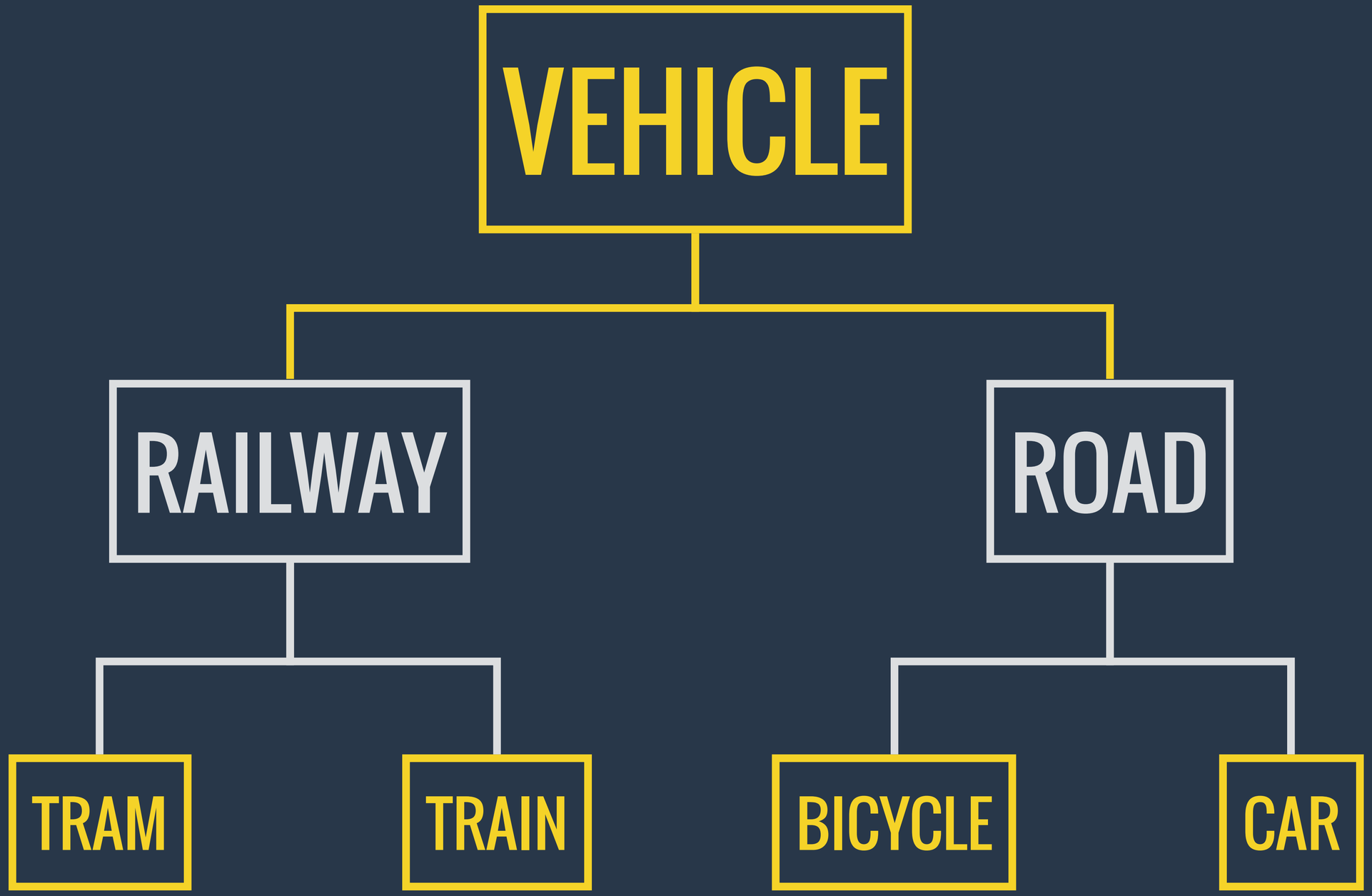
```
class EncryptionAssistant {
  private var key = "420mlg$crub"
  public func encrypt(pass: String) -> String {
    return rsaEncrypt(pass, key)
  }
}
```

```
let assistant = EncryptionAssistant()
assistant.encrypt("secret") // 1Ll00Myn4RtY
assistant.key // compile error!
```

~~1. ABSTRACTION~~

~~2. ENCAPSULATION~~

3. INHERITANCE


VEHICLE
RAILWAY
ROAD
TRAM
TRAIN
BICYCLE
CAR

```
class Car {
  var color: String = "red"
  var name: String {
    return "\(color) car"
  }
}

class BlueCar: Car {
  override var color = "blue"
}
```

```
Car().name // red car
BlueCar().name // blue car
```

~~1. ABSTRACTION~~

~~2. ENCAPSULATION~~

~~3. INHERITANCE~~

2. IMPERATIVE PROGRAMMING

IMPERATIVE PHRASES WHICH CHANGE THE GLOBAL STATE OF A PROGRAM

```
let numbers = [1, 2, 3, 4, 5, 6]
var sum = 0
var odds: [Int] = []
for number in numbers {
  sum += number
  if number % 2 == 1 {
    odds.append(number)
  }
}
```

```
getRemoteData("url", { data, error in
  if error == nil {
    parseData(data, { parsed, error in
      if error == nil {
        handleParsedData(parsed)
      } else {
        displayError(error)
      }
    })
  } else {
    displayError(error)
  }
})
```

IMPERATIVE
PROGRAMMING IS
THE MOST POPULAR

IMPERATIVE
PROGRAMMING IS
THE EASIEST

IMPERATIVE PROGRAMMING IS THE WORST

1. ERROR-PRONE
2. NOT SCALABLE
3. TOO COMPLICATED

```
getRemoteData("example.com", { data, error in
  if error == nil {
    parseData(data, { parsed, error in
      if error == nil {
        handleParsedData(parsed)
      } else {
        displayError(error)
      }
    })
  } else {
    displayError(error)
  }
})
```

```
getRemoteData("example.com", { data, error in
  if error == nil {
    parseData(data, { parsed, error in
      if error == nil {
        if parsedDataValid(parsed) {
          handleParsedData(parsed)
        }
      } else {
        displayError(error)
      }
    })
  } else {
    displayError(error)
  }
})
```

```
getRemoteData("example.com", { data, error in
  if error == nil {
    parseData(data, { parsed, error in
      if error == nil {
        if parsedDataValid(parsed) {
          saveParsedDataInCache(parsed, { error in
            if error == nil {
              handleParsedData(parsed)
            } else {
              displayError(error)
            }
          })
        }
      } else {
        displayError(error)
```

```
getRemoteData("example.com", { data, error in
  if error == nil {
    parseData(data, { parsed, error in
      if error == nil {
        if parsedDataValid(parsed) {
          saveParsedDataInCache(parsed, { error in
            if error == nil {
              handleParsedData(parsed, { error in
                if error == nil {
                  displaySuccess()
                } else {
                  displayError(error)
                }
              })
            } else {
```

3. DECLARATIVE PROGRAMMING

DECLARE WHAT YOU'RE TRYING TO ACCOMPLISH, NOT HOW TO DO IT

```
let numbers = [1, 2, 3, 4, 5, 6]
var sum = 0
var odds: [Int] = []
for number in numbers {
  sum += number
  if number % 2 == 1 {
    odds.append(number)
  }
}
```

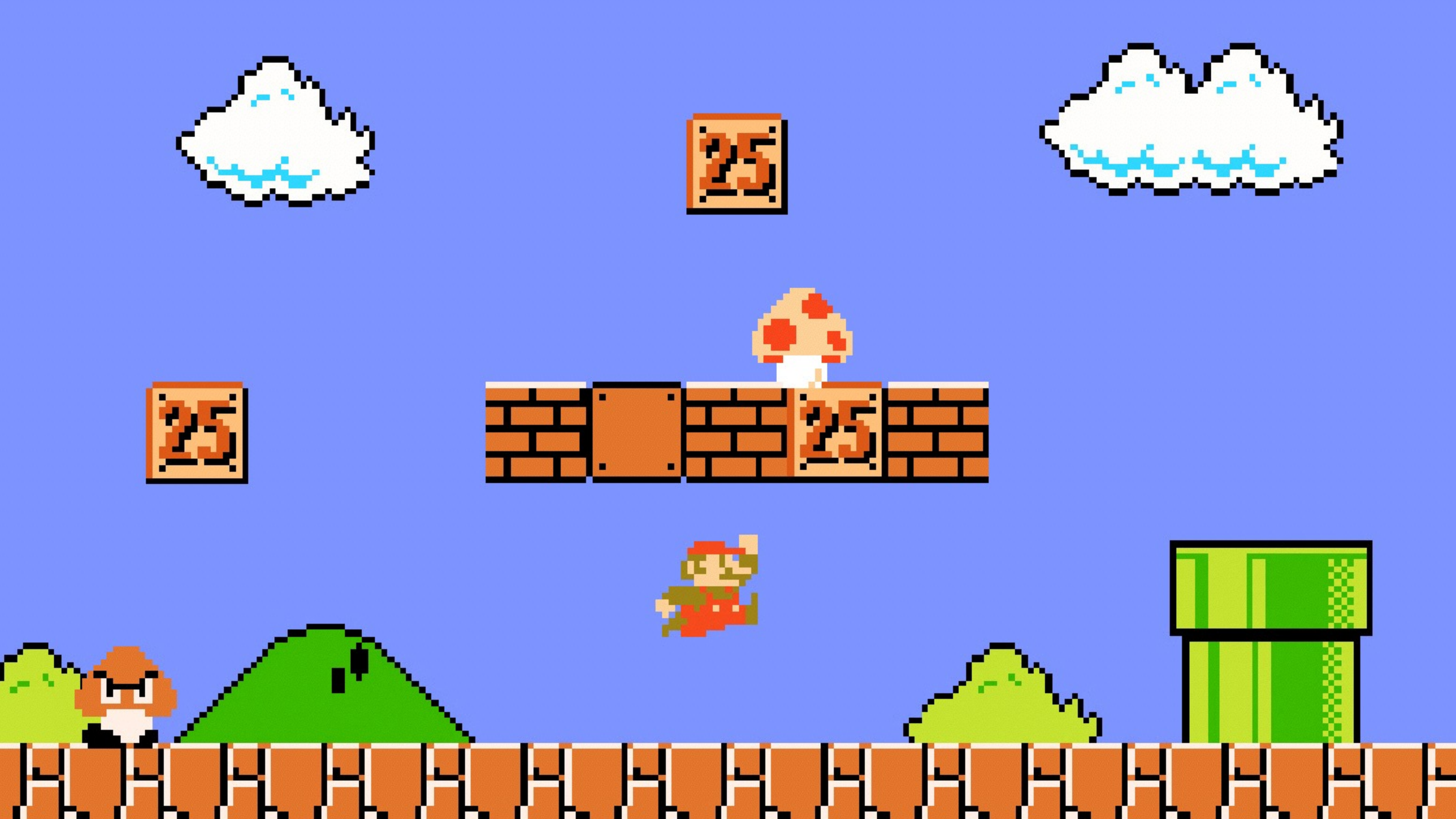
```
var sum = 0
var odds: [Int] = []
let numbers = [1, 2, 3, 4, 5, 6]
for number in numbers {
  sum += number // reduction
  if number % 2 == 1 { // filtration
    odds.append(number)
  }
}
```

```
let numbers = [1, 2, 3, 4, 5, 6]
let sum = reduce(numbers, 0, { memo, number in
  return memo + number
})
let odds = filter(numbers, { number in
  return number % 2 == 1
})
```

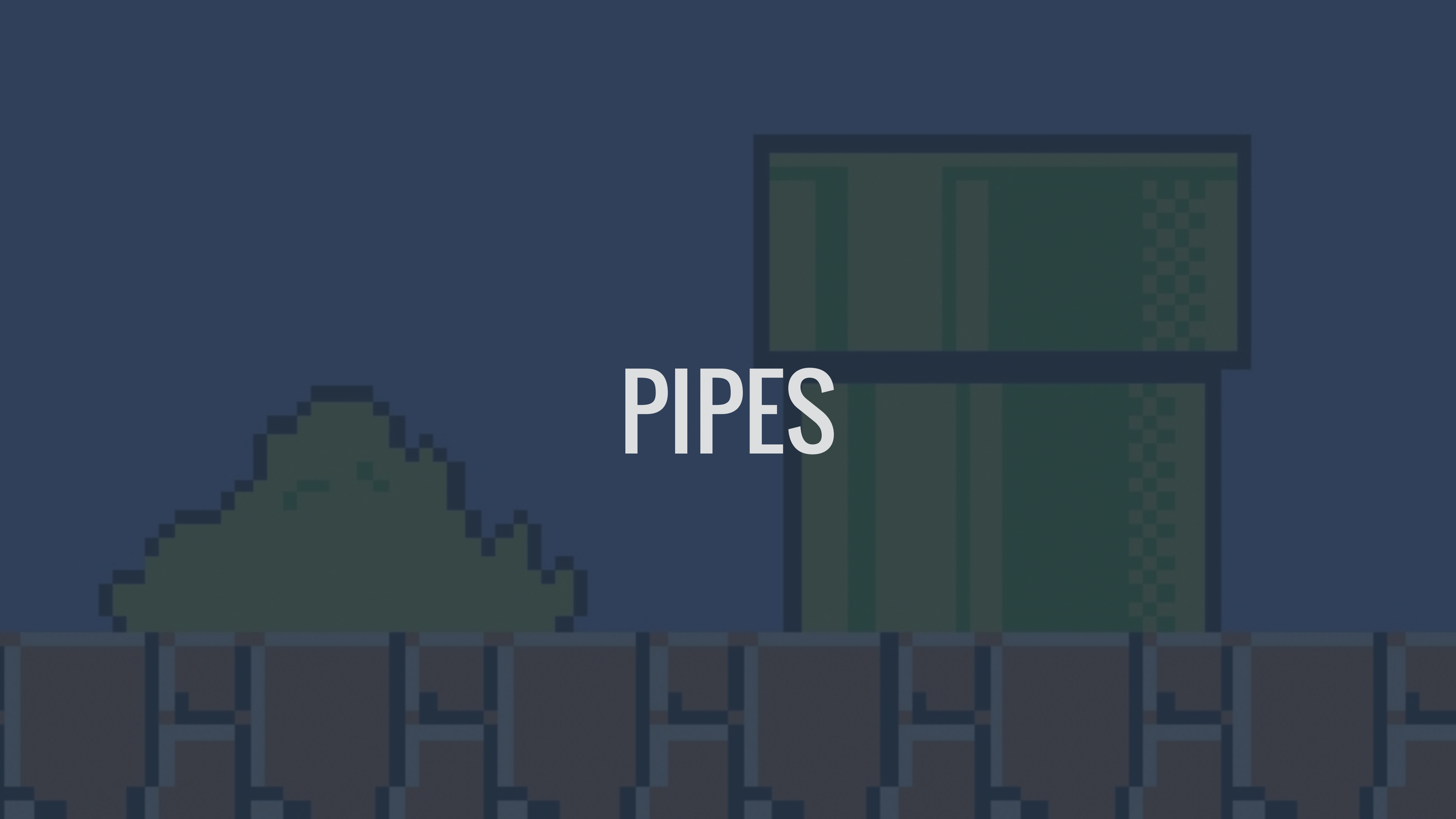
```
let numbers = [1, 2, 3, 4, 5, 6]
let sum = reduce(numbers, 0, +)
let odds = filter(numbers, { $0 % 2 == 1 })
```

```
getRemoteData("example.com",
  if error == nil {
    parseData(data, { parsed,
      if error == nil {
        if parsedDataValid(pars
          saveParsedDataInCach
            if error == nil {
              handleParsedData(
                if error == nil {
                  displaySuccess
                } else {
                  displayError(er
                }
              })
            } else {
              displayError(error)
            }
          })
        }
      } else {
        displayError(error)
      }

      se {
      splayError(error)
      }
    })
```


25
25
25

PIPES



DOWNLOAD PARSE SAVE IN CACHE DISPLAY

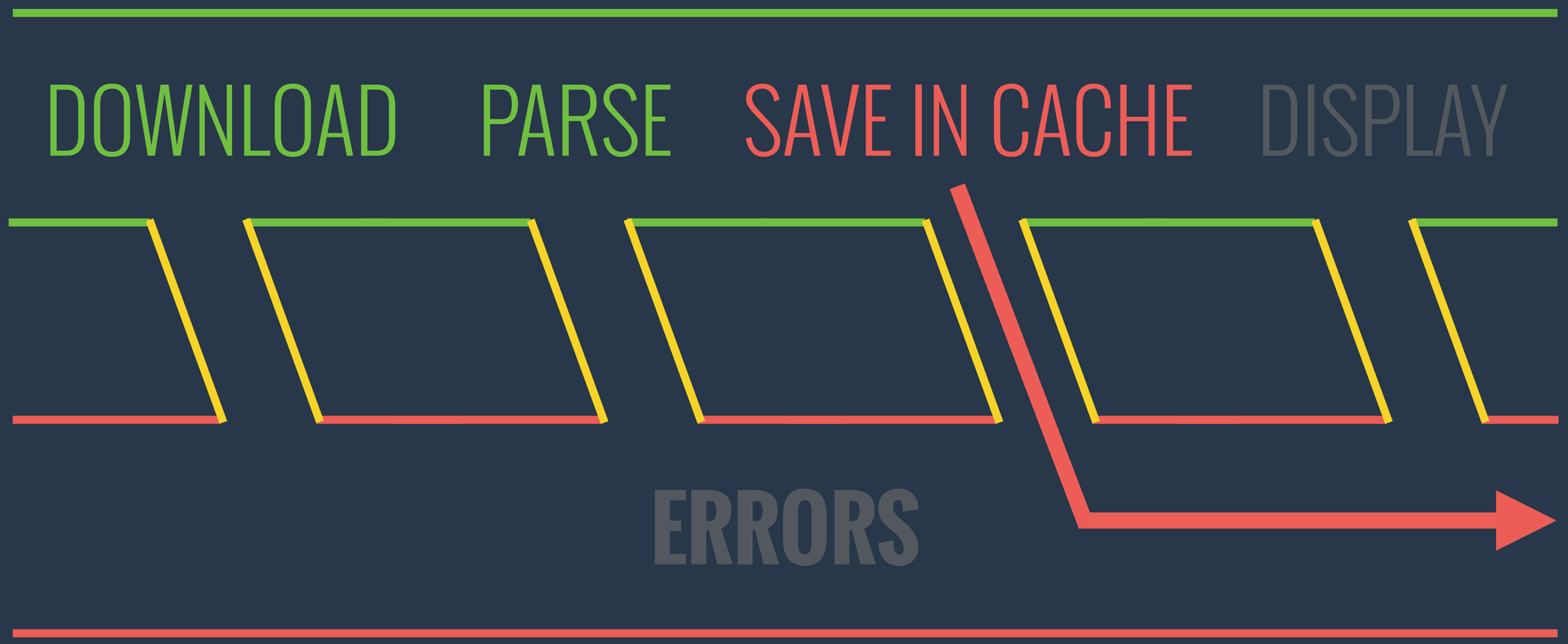
ERRORS


DOWNLOAD
PARSE
SAVE IN CACHE
DISPLAY
ERRORS


DOWNLOAD
PARSE
SAVE IN CACHE
DISPLAY
ERRORS


DOWNLOAD
PARSE
SAVE IN CACHE
DISPLAY
ERRORS


DOWNLOAD
PARSE
SAVE IN CACHE
DISPLAY
ERRORS


DOWNLOAD
PARSE
SAVE IN CACHE
DISPLAY
ERRORS

```
getRemoteData("example.com")
  .then({ data in parseData(data) })
  .filter({ parsed in parsedDataValid(parsed) })
  .then({ parsed in saveInCache(parsed) })
  .then({ parsed in handleParsedData(parsed) })
  .error({ error in displayError(error) })
```

```
getRemoteData("example.com")
  .then({ data in parseData(data) })
  .filter({ parsed in parsedDataValid(parsed) })
  .filter({ parsed in !alreadyInCache(parsed) })
  .then({ parsed in saveInCache(parsed) })
  .then({ parsed in handleParsedData(parsed) })
  .error({ error in displayError(error) })
```

```
getRemoteData("example.com")
  .then({ data in parseData(data) })
  .filter({ parsed in parsedDataValid(parsed) })
  .filter({ parsed in !alreadyInCache(parsed) })
  .then({ parsed in saveInCache(parsed) })
  .then({ parsed in handleParsedData(parsed) })
  .error({ error in displayError(error) })
```

DECLARATIVE PROGRAMMING IS MUCH SIMPLER

DECLARATIVE PROGRAMMING IS MUCH SAFER

DECLARATIVE PROGRAMMING IS MORE SCALABLE

WHICH PARADIGM IS THE BEST?

1. OBJECT-ORIENTED
2. IMPERATIVE
3. DECLARATIVE

~~1. OBJECT-ORIENTED~~

2. IMPERATIVE

3. DECLARATIVE

~~1. OBJECT-ORIENTED~~

~~2. IMPERATIVE~~

3. DECLARATIVE

~~1. OBJECT-ORIENTED~~

~~2. IMPERATIVE~~

3. DECLARATIVE

TOGETHER

THANK YOU

ADRIAN KASHIVSKYY

@akashivskyy

github.com/akashivskyy/talks