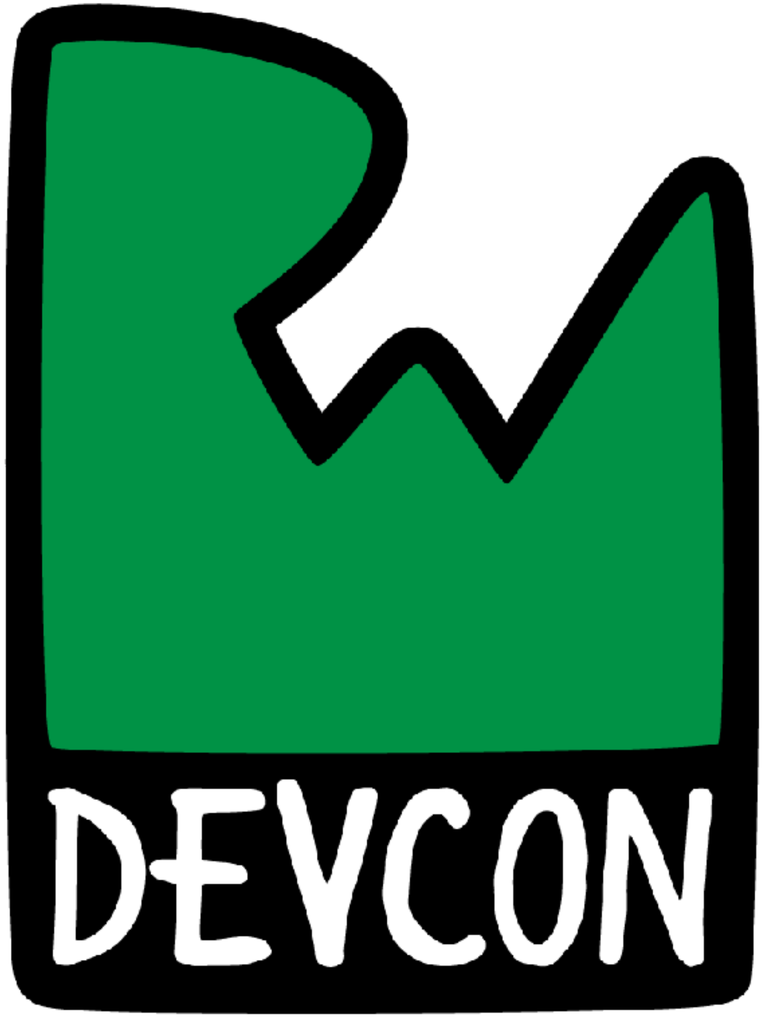
Session 2: Swift 4 Serialization



The Basics and Beyond

Serialization

Domain Objects

Encoding
Serialization
Freeze-drying

Decoding
De-serialization
Re-hydrating

Data
JSON/XML/UDT

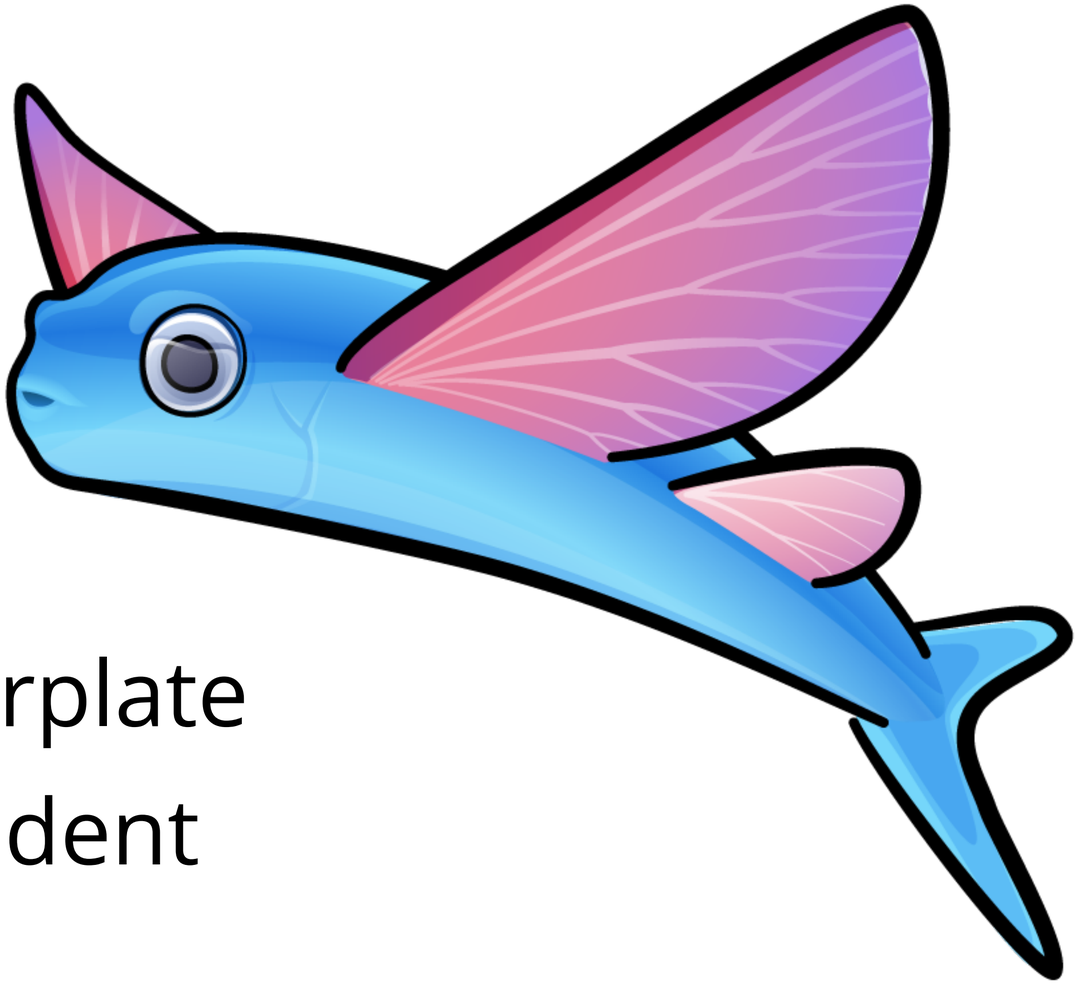
Serialization

SE-0166 Swift Archival & Serialization
SE-0167 Swift Encoders

Authors:
Itai Ferber, Michael LeHew, Tony Parker

Codability

- First class
- Type safe
- Easy to use
- Flexible
- Minimized boilerplate
- Format independent



JSON Format Review

Name	Types	Example
string	unicode	“Hello World”
number	integers, floating point	42
object	{string: value}	{“gravity”: 9.8, “name”: null}
array	[value]	[“cat”, true, null, 42]
value	string, number, object, array, true, false, null	[{“size”: 10}, 10.3, false, “yes”]

Static Reflection

- Member-wise Initializers
- Automatic Equatable Conformance
- Automatic Hashable Conformance
- **Automatic Codable Conformance**

```
struct Location :{ Codable
  var longitude, latitude: Double
}
```

Now Swift: Standard Library + Compiler Magic

Future Swift: Macros

{ “longitude": 38.8031375, “latitude": -77.0675817 }

Under The Hood

```
typealias Codable = Decodable & Encodable
```

```
protocol Decodable {
  init(from decoder: Decoder) throws
}
```

```
protocol Encodable {
  func encode(to encoder: Encoder) throws
}
```

Encoder & Decoder

KeyedDecodingContainer<Key>

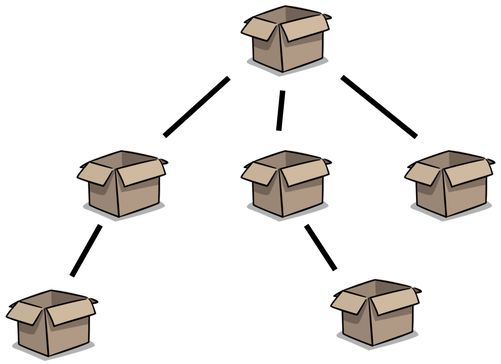
KeyedEncodingContainer<Key>

UnkeyedDecodingContainer

UnkeyedEncodingContainer

SingleValueDecodingContainer

SingleValueEncodingContainer



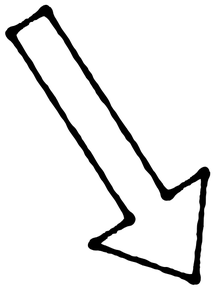
Coding Keys

KeyedDecodingContainer<Key>

```
… where Key: CodingKey
```

A CodingKeys enumeration backed by a String is generated for each stored property and is used to implement Decodable and Encodable.

You provide your own to customize behavior.



String / Int

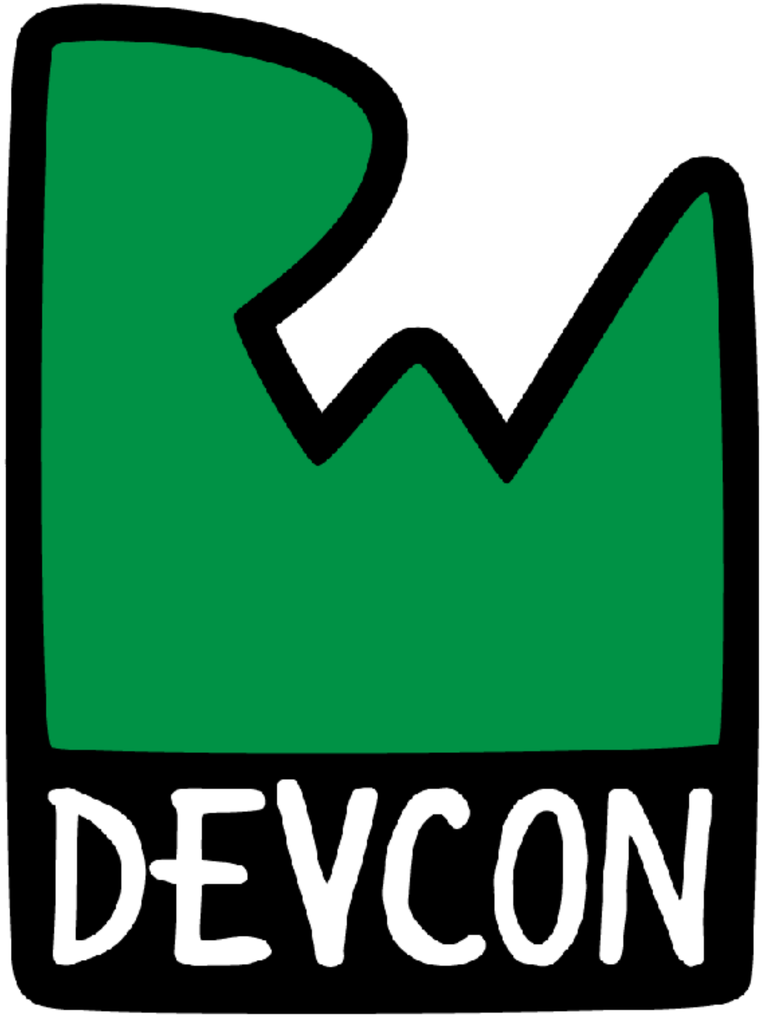
Compiler Generated Code

- **CodingKeys** case for each stored properties
- **init(from decoder:)**
- **encode(to encoder:)**

Requires **Codable** conformance part of declaration, not as an extension.

Demo 1

Session 2: Swift 4 Serialization



Errors And Polymorphism

Missing Values (nil)

When the compiler sees an optional, it uses encode / decode if present.

This means that the key is dropped completely for a missing value.

JSON `null`

⚠️ **Plist** `<string>$null</string>`

Help Stop Martian Accidents

```
protocol Length {}
struct Feet: Length {}
struct Meters: Length {}
```

```
struct Distance<Units: Length>: Codable, Equatable {
  var value: Double
  …
}
```

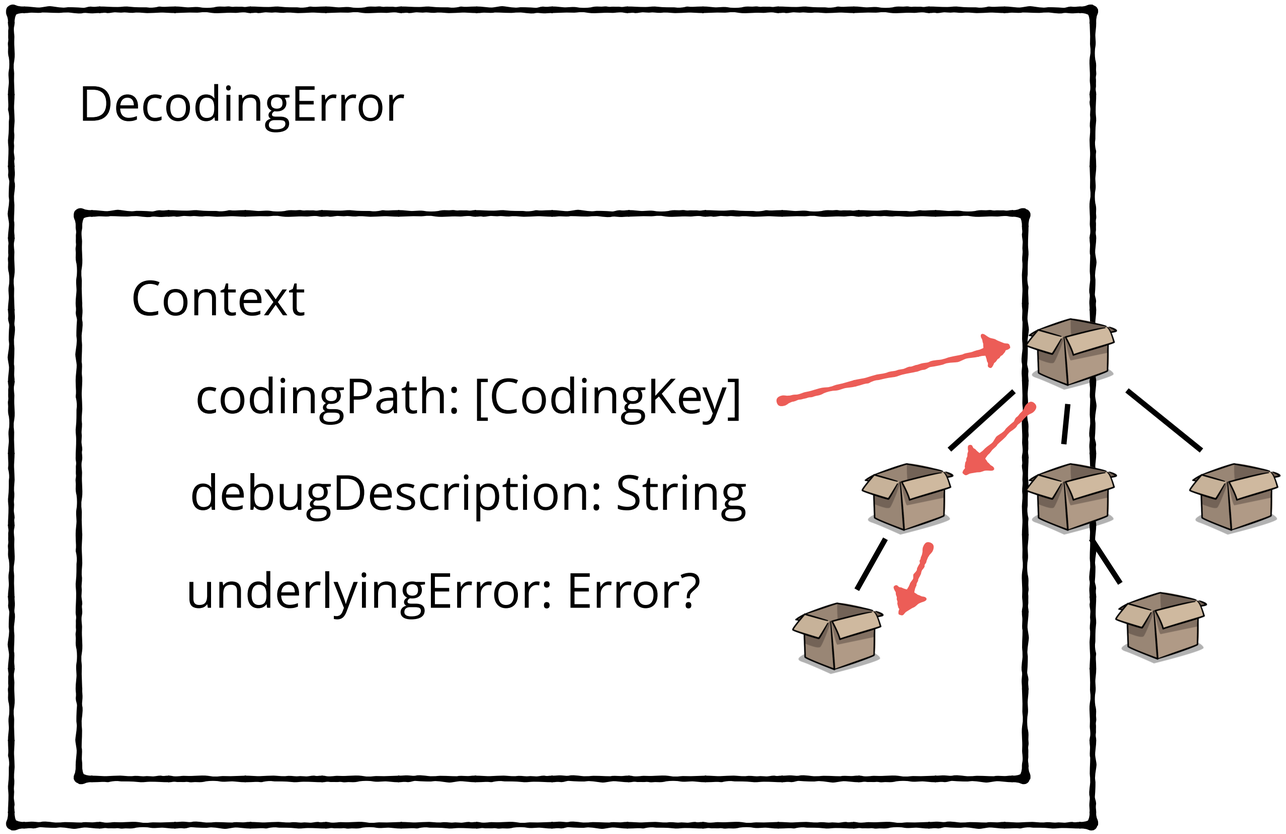
```
let measurements: [Distance<Meters>] = [3.0, 4.25, 0.25]
let encoder = JSONEncoder()
let data = try encoder.encode(measurements)
```

```
[{"value":3},{"value":4.25},{"value":0.25}]
```

```
let decoder = JSONDecoder()
let restored = try decoder.decode([Distance<Feet>].self, from: data)
```

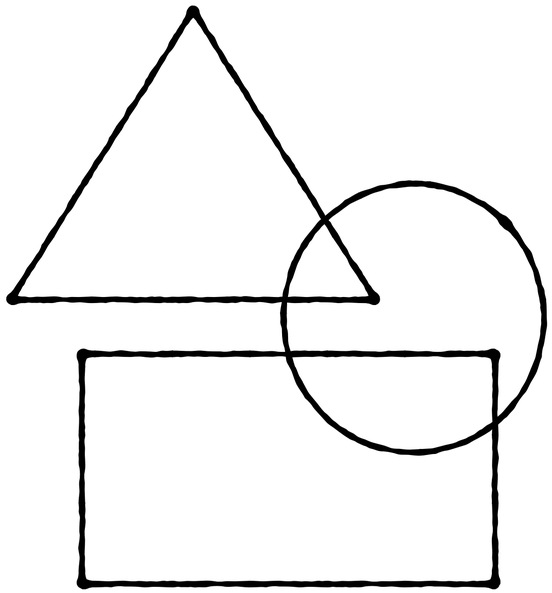
Handling Errors

- DecodingError: Error
  - typeMismatch
  - valueNotFound
  - keyNotFound
  - dataCorrupted
- EncodingError: Error
  - invalidValue

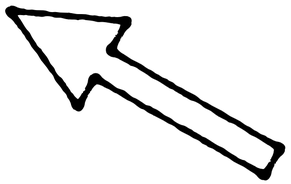


Polymorphic Collections

```
protocol Shape {...}

struct Circle: Shape, Codable {...}
struct Rectangle: Shape, Codable {...}
struct Triangle: Shape, Codable {...}

let shapes: [Shape] = []
```



```
enum AnyShape: Codable {
  case circle(Circle)
  case square(Square)
  case rectangle(Rectangle)
}
```



But, cannot be automatically implemented.

Handling Inheritance

```
class Sample: Codable {
  var date: Date
}
```

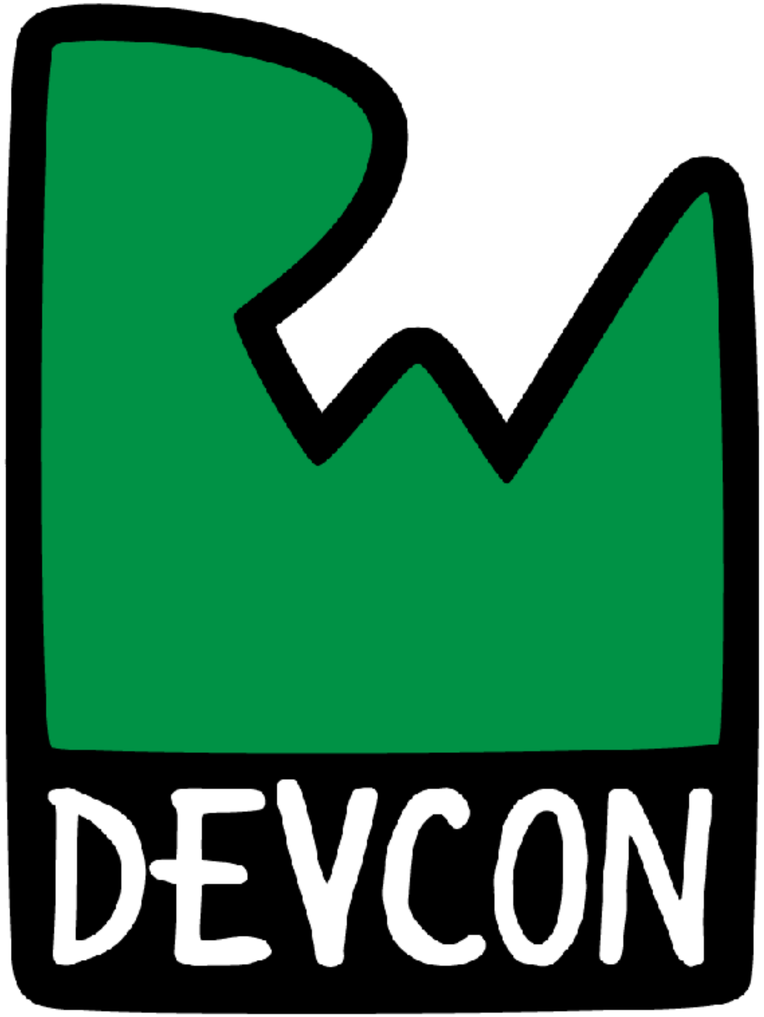
```
class RockSample: Sample {
  var mass: Double
  var rockType: RockType
}
```

```
{
  "date" : 540596433.141518
} "type" : "igneous",
  "sample" : {
    "date" : 540596433.141518
  }
}
```

- Override Codable methods
- Use the superContainer from the subclass

Demo 2

Session 2: Swift 4 Serialization



Dates, Relations, Coders

Date Handling

```
class Sample: Codable, Equatable {
  var date: Date
  init(date: Date) {
    self.date = date
  }
}
```

JSON

```
{"date":541016052.12442803}
```

Property List

```
<dict>
  <key>date</key>
  <date>2018-02-22T18:20:00Z</date>
</dict>
```

```
let encoder = JSONEncoder()
encoder.dateEncodingStrategy = .iso8601

let decoder = JSONDecoder()
decoder.dateDecodingStrategy = .iso8601
```

JSON

```
{"date":"2018-04-07T18:30:34Z"}
```

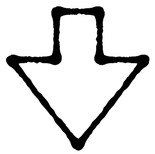
Date Handling

```
extension DateFormatter {
  // Handles "2018-04-07"
  static var yearMonthDay: DateFormatter = {
    let formatter = DateFormatter()
    formatter.dateFormat = "yyyy-MM-dd"
    formatter.locale = Locale(identifier: "en_US_POSIX")
    return formatter
  }()
}
```

```
let encoder = JSONEncoder()
encoder.dateEncodingStrategy = .formatted(.yearMonthDay)

let decoder = JSONDecoder()
decoder.dateDecodingStrategy = .formatted(.yearMonthDay)
```

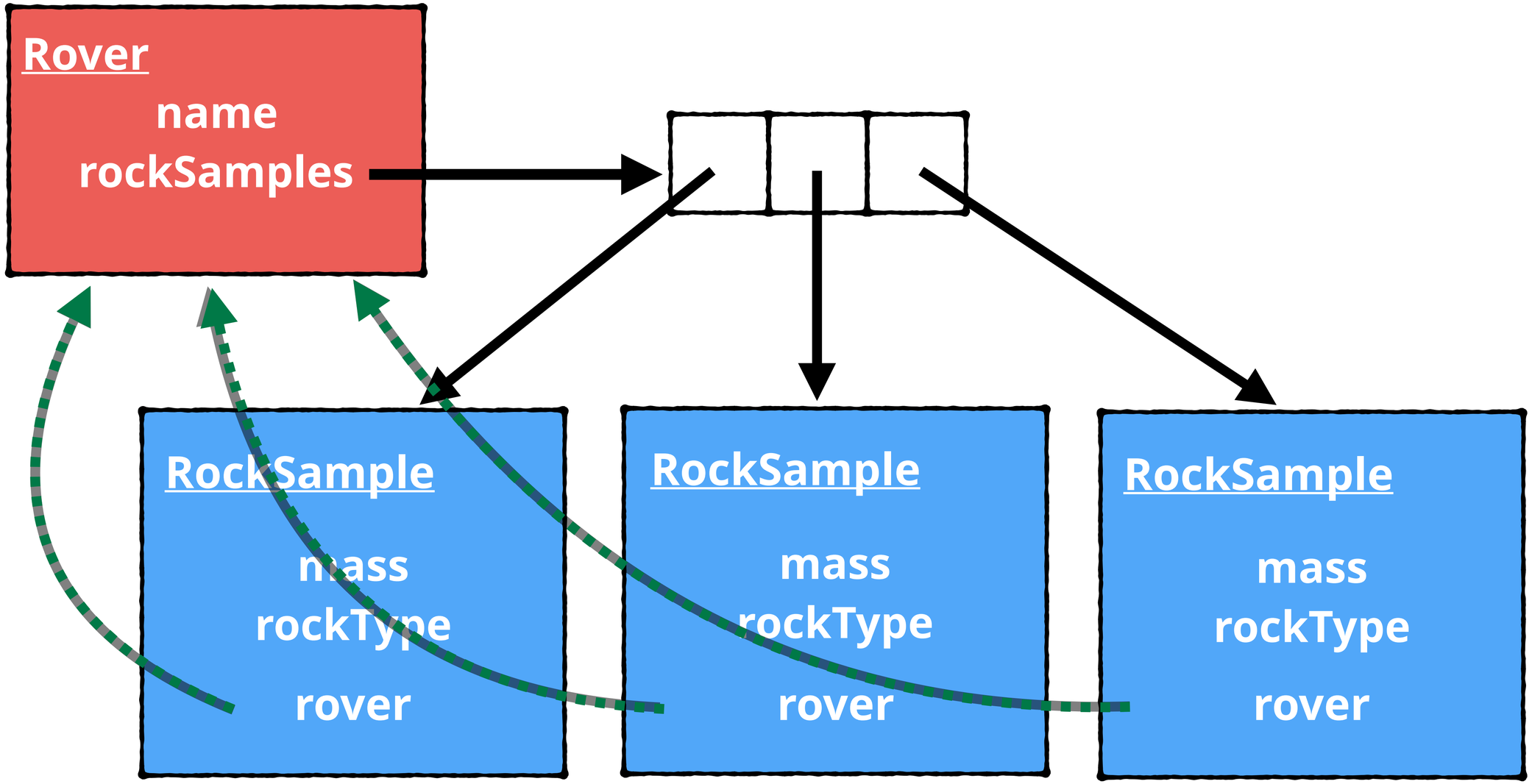
Date Handling

```
required init(from decoder: Decoder) throws {
  let container = try decoder.container(keyedBy: CodingKeys.self)
  let dateString = try container.decode(String.self, forKey: .date)
  guard let date = DateFormatter.yearMonthDay.date(from: dateString) else {
    throw DecodingError.dataCorruptedError(forKey: .date, in: container,
                                           debugDescription: dateString)
  }
  self.date = date
}
```



```
date = try dateStringDecode(forKey: .date,
                            from: container,
                            with: .yearMonthDay)
```

Relationships



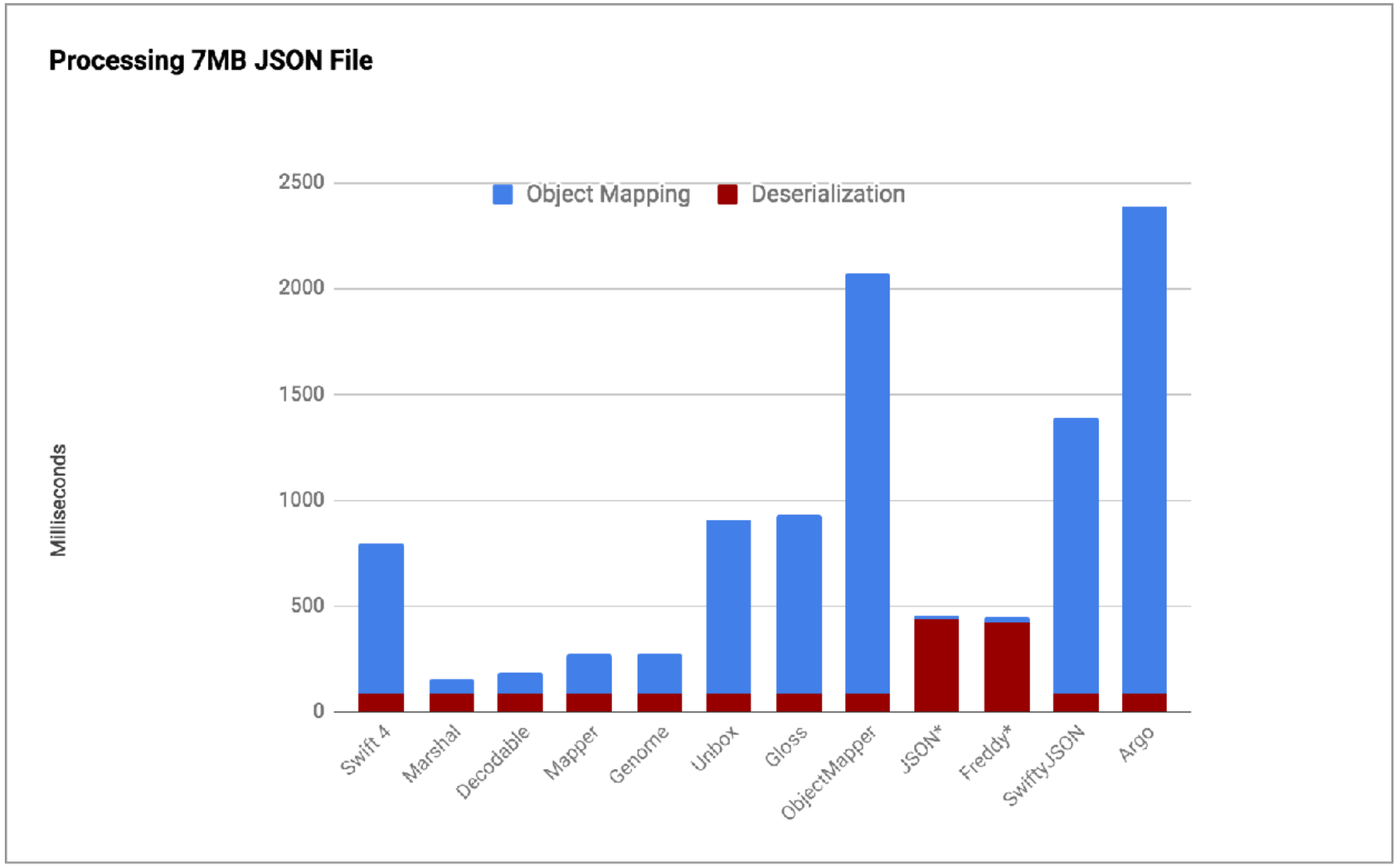
Circular Relationships

- Parent ***(Rover)***
  - Strong references to children
  - Serialize the children
  - Restores the relationship from child back to parent on decode
- Children ***(RockSample)***
  - Keeps weak reference back to parent
  - Don’t encode the parent

Coders

- Coders and Decoders
  - Manage containers (keyed, unkeyed, single value)
  - Maintain coding path
  - Support a userInfo dictionary `[CodingUserInfoKey : Any]`
- Special Capabilities
  - JSON Date Formatting, Data Formatting, Number Handling
  - JSON Pretty Printing
  - JSON Automatic Camel Casing (Swift 4.1)
  - Property List Output Format (binary, xml)

JSONDecoder Performance



Source: https://github.com/bwhiteley/JSONShootout

Implementing Custom Coders and Decoders

- What does it take to create one?
  - Fairly straight forward
  - A lot of code (typing)
  - If you have a parser / emitter available adds 1-2k LOC

Example:

https://github.com/jpsim/Yams

http://www.yaml.org/spec/1.2/spec.html

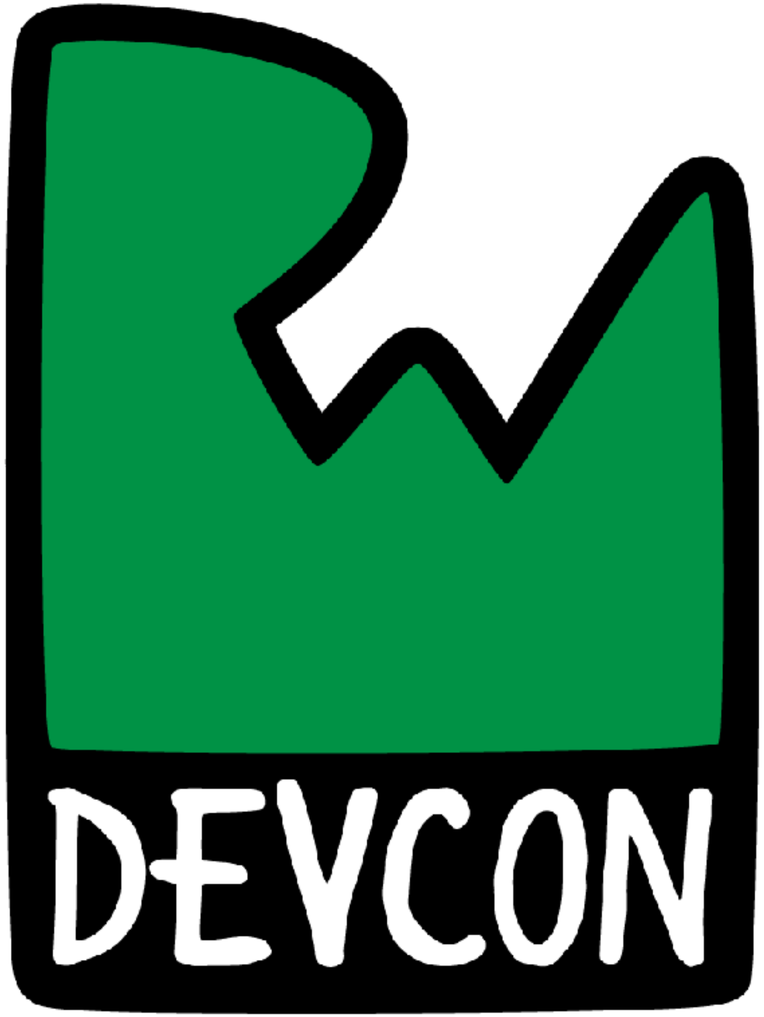
Yams Example

```
import Yams


let photos = [Photo(url: nil, camera: .chemcam, time: Sols(2)),
              Photo(url: URL(string: "https://go.nasa.gov/2omxQlM")!,
                    camera: .navcams, time: Sols(3))]

let encoder = YAMLEncoder()
let yaml = try encoder.encode(photos)
print(yaml)
```

```
- url: null
  camera: chemcam
  time:
    sols: 2e+0
- url: https://go.nasa.gov/2omxQlM
  camera: navcams
  time:
    sols: 3e+0
```

Demo 3

Session 2: Swift 4 Serialization



Conclusion

What You Learned

- **Demo 1**: Codable: The Basics and Beyond
- **Demo 2**: Errors and Polymorphism
- **Demo 3**: Dates, Relationships, Custom Coders

Where To Go From Here?

- SE-0166
  https://github.com/apple/swift-evolution/blob/master/proposals/0166-swift-archival-serialization.md
- SE-0167
  https://github.com/apple/swift-evolution/blob/master/proposals/0167-swift-encoders.md
- Apple Guide
  https://developer.apple.com/documentation/foundation/archives_and_serialization/encoding_and_decoding_custom_types
- Mike Ash's Custom Binary Coder
  https://www.mikeash.com/pyblog/friday-qa-2017-07-28-a-binary-coder-for-swift.html
- Many more... Google: Swift Codable
- Twitter: @rayfix