# OPTICAL SWIFT

# EMBRACING FUNCTIONAL REFERENCES

# FUNCTIONAL PROGRAMMING

```
SIGABRT
'NSGenericException', reason:
'*** Collection <__NSDictionaryM: 0x17025b8d0> was mutated while being enumerated.'
```

# EXC_BAD_ACCESS
# CoreData

0x3687eec4 -[NSManagedObject(_NSInternalMethods) _setOriginalSnapshot__:] + 40

# LET'S JUST DO IT!

# SWIFT

```
class Person {
    var name = "Oleg"
    var knowledge: KnowledgeBase = .iOS
}
```

```
person.knowledge.skills = .android // Works fine
```

# SWIFT

```swift
struct Person {
    let name: String
    let knowledge: KnowledgeBase
}
```

```swift
person.knowledge.skills = .android // Compile error

/**
    `skills` is a `let` constant.
    `knowledge` is a `let` constant.
    `person` is a `let` constant.
*/
```

# SWIFT

```
let newKnowledge = KnowledgeBase(skills: .android,
                                 experience: knowledge.experience)
let newPerson = Person(name: person.name,
                       knowledge: newKnowledge)
```

# PYTHON

```python
string = '[{"oleg":{"experience":10, "skills":["iOS"]}}]'

person = json.loads(string)[0]['oleg']

oleg = Person("Oleg",
              person['skills'][0], # "iOS"
              person['experience']) # 10
```

# SWIFT

```
let string = "[{\"oleg\":{\"experience\":10, \"skills\":[\"iOS\"]}}]"

if let data = string.data(using: .utf8),
```

# SWIFT

```swift
let string = "[{\"oleg\":{\"experience\":10, \"skills\":[\"iOS\"]}}]"

if let data = string.data(using: .utf8),
    let json = try? JSONSerialization.jsonObject(with: data, options: .init(rawValue: 0)),
```

# SWIFT

```swift
let string = "[{\"oleg\":{\"experience\":10, \"skills\":[\"iOS\"]}}]"

if let data = string.data(using: .utf8),
    let json = try? JSONSerialization.jsonObject(with: data, options: .init(rawValue: 0)),
    let arr = json as? [[String: Any]],
```

# SWIFT

```swift
let string = "[{\"oleg\":{\"experience\":10, \"skills\":[\"iOS\"]}}]"

if let data = string.data(using: .utf8),
    let json = try? JSONSerialization.jsonObject(with: data, options: .init(rawValue: 0)),
    let arr = json as? [[String: Any]],
    let person = arr.first,
```

# SWIFT

```swift
let string = "[{\"oleg\":{\"experience\":10, \"skills\":[\"iOS\"]}}]"

if let data = string.data(using: .utf8),
    let json = try? JSONSerialization.jsonObject(with: data, options: .init(rawValue: 0)),
    let arr = json as? [[String: Any]],
    let person = arr.first,
    let oleg = person["oleg"] as? [String: Any],
```

# SWIFT

```swift
let string = "[{\"oleg\":{\"experience\":10, \"skills\":[\"iOS\"]}}]"

if let data = string.data(using: .utf8),
    let json = try? JSONSerialization.jsonObject(with: data, options: .init(rawValue: 0)),
    let arr = json as? [[String: Any]],
    let person = arr.first,
    let oleg = person["oleg"] as? [String: Any],
    let skills = oleg["skills"] as? [String],
```

# SWIFT

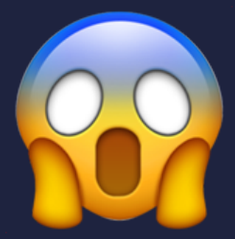
```swift
let string = "[{\"oleg\":{\"experience\":10, \"skills\":[\"iOS\"]}}]"

if let data = string.data(using: .utf8),
    let json = try? JSONSerialization.jsonObject(with: data, options: .init(rawValue: 0)),
    let arr = json as? [[String: Any]],
    let person = arr.first,
    let oleg = person["oleg"] as? [String: Any],
    let skills = oleg["skills"] as? [String],
    let experience = oleg["experience"] as? Int,
```

# SWIFT

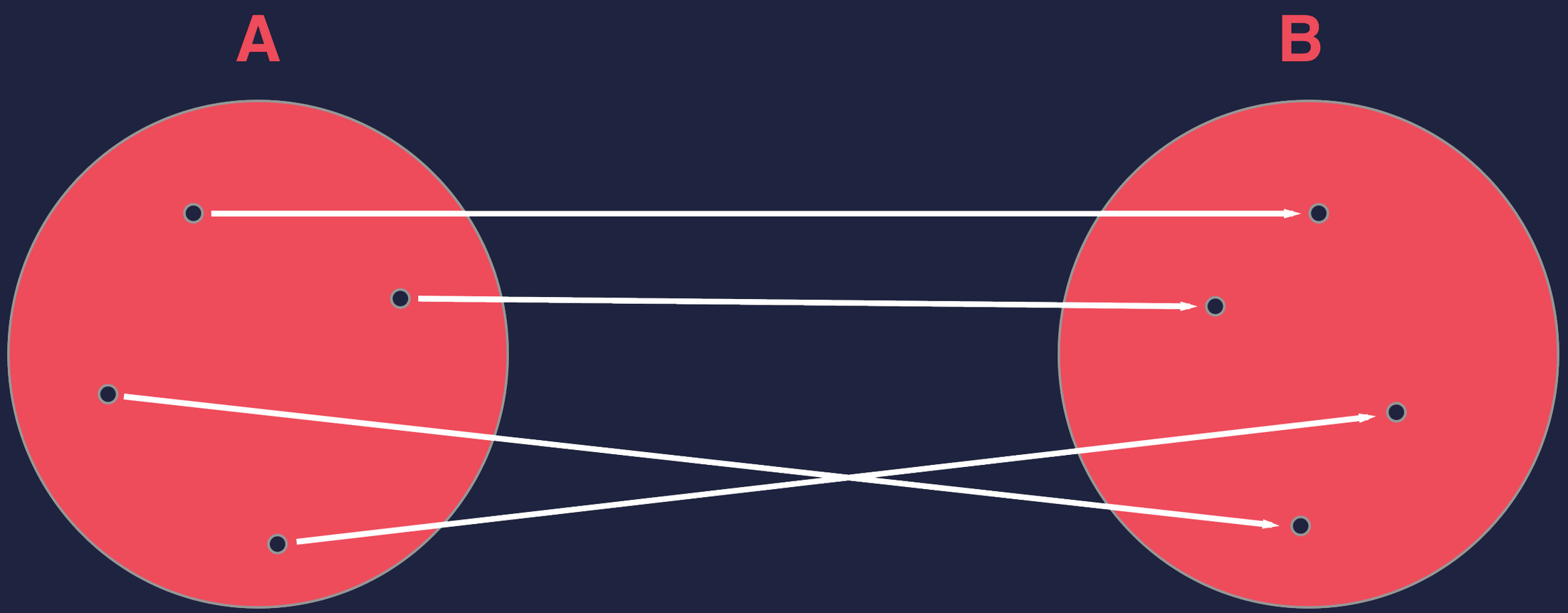
```swift
let string = "[{\"oleg\":{\"experience\":10, \"skills\":[\"iOS\"]}}]"

if let data = string.data(using: .utf8),
    let json = try? JSONSerialization.jsonObject(with: data, options: .init(rawValue: 0)),
    let arr = json as? [[String: Any]],
    let person = arr.first,
    let oleg = person["oleg"] as? [String: Any],
    let skills = oleg["skills"] as? [String],
    let experience = oleg["experience"] as? Int,
    let mainSkill = skills.first {
        let person = Person(name: "Oleg", knowledge: ...)
}
```

# SWIFT

```swift
let string = "[{\"oleg\":{\"experience\":10, \"skills\":[\"iOS\"]}}]"

if let data = string.data(using: .utf8),
    let json = try? JSONSerialization.jsonObject(with: data, options: .init(rawValue: 0)),
    let arr = json as? [[String: Any]],
    let person = arr.first,
    let oleg = person["oleg"] as? [String: Any],
    let skills = oleg["skills"] as? [String],
    let experience = oleg["experience"] as? Int,
    let mainSkill = skills.first {
        let person = Person(name: "Oleg", knowledge: ...)
}
```



# FUNCTION



```
let square: (Double) -> Double = { x in x * x }
```

```
let personName: (Person) -> String = { person in person.name }
```

```
extension Array {
    func sort(compare: (Element, Element) -> Bool) -> [Element] { ... }
}
```

```
func derivative(f: (Double) -> Double) -> (Double) -> Double
```

```
derivative(square) // { x in 2 * x }
```

# COMPOSITION

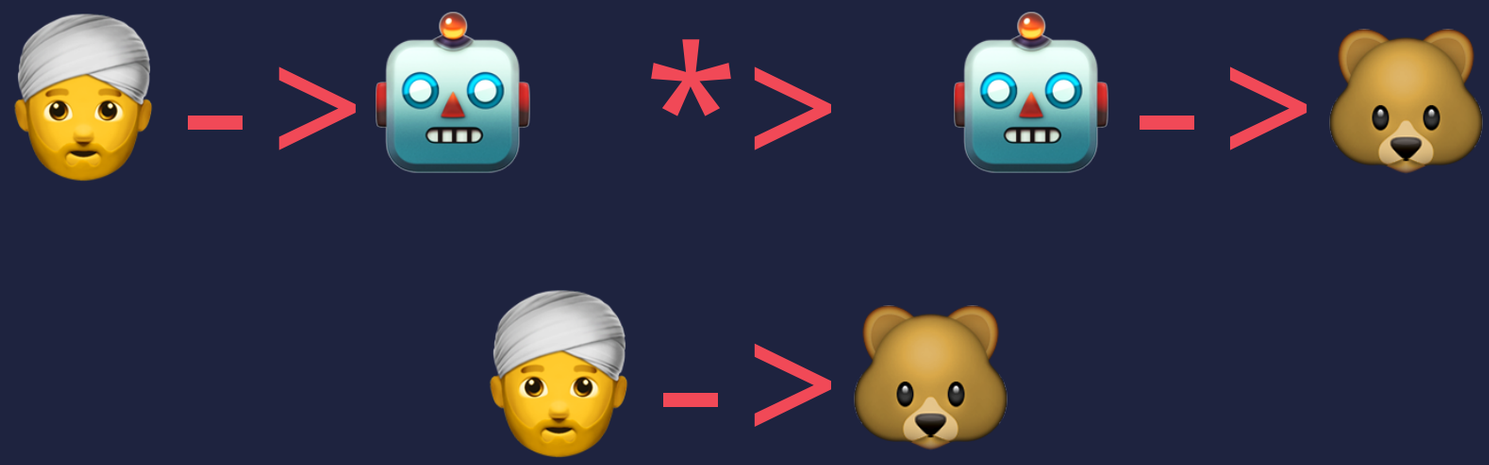
```
func *> <A, B, C>(f: @escaping (A) -> B, g: @escaping (B) -> C) -> (A) -> C {
    return { g(f($0)) }
}
```

# COMPOSITION

```
func *> <A, B, C>(f: @escaping (A) -> B, g: @escaping (B) -> C) -> (A) -> C {
    return { g(f($0)) }
}
```

# COMPOSITION

```
func *> <A, B, C>(f: @escaping (A) -> B, g: @escaping (B) -> C) -> (A) -> C {
    return { g(f($0)) }
}
```



# (Data) -> Any?

```
func parseJSON(data: Data) -> [[String: Any]]?
```

parseJSON

# (Data) -> [[String: Any]]?

```
func parseJSON(data: Data) -> [[String: Any]]?
func first<T>(arr: [T]) -> T?
```

```
parseJSON *> first
```

# (Data) -> KnowledgeBase?

```
func parseJSON(data: Data) -> [[String: Any]]?
func first<T>(arr: [T]) -> T?
```

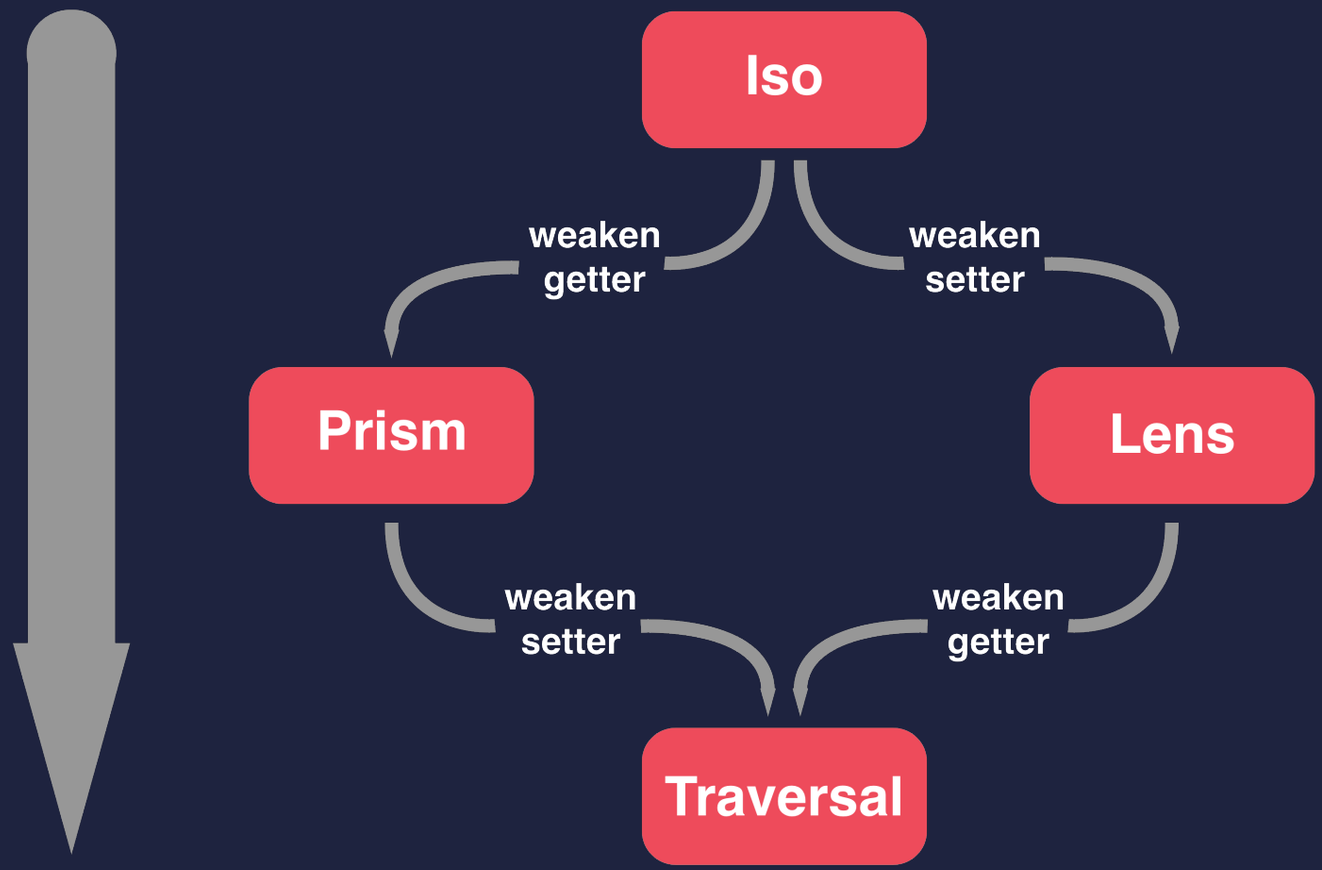
```
parseJSON *> first *> KnowledgeBase.init
```

# (Data) -> Person?

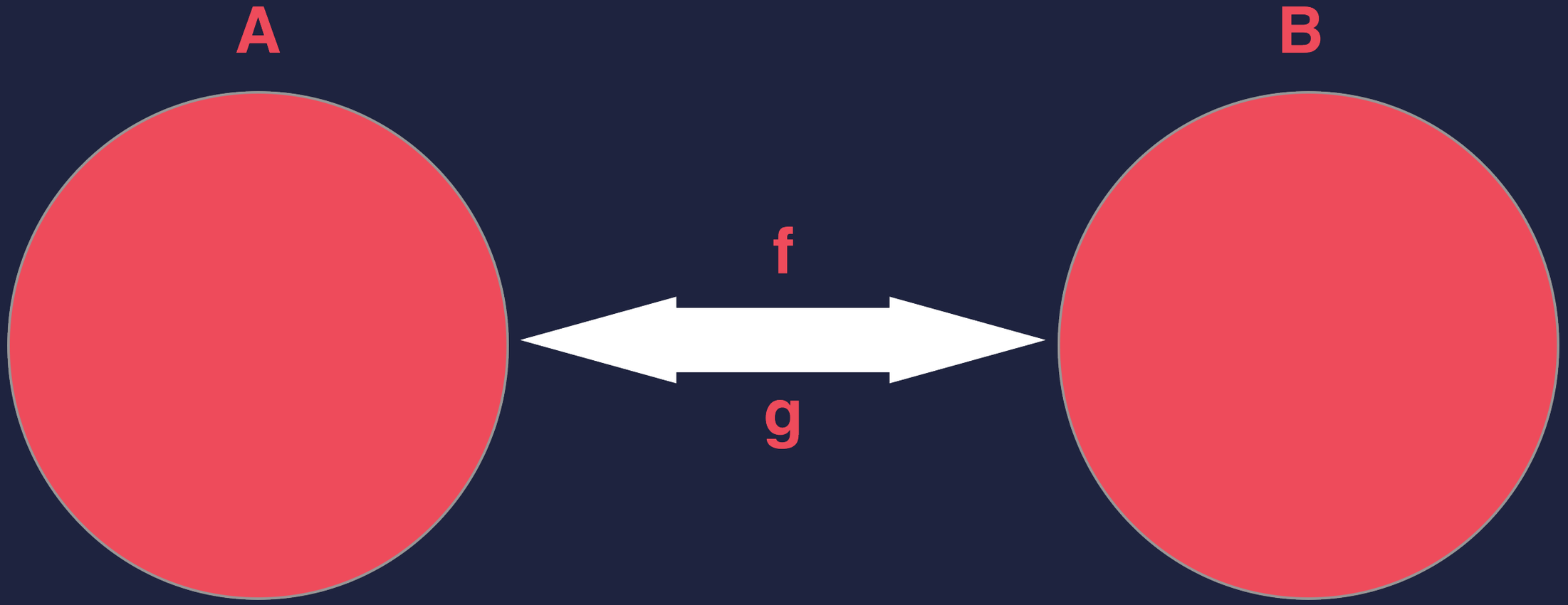
```
func parseJSON(data: Data) -> [[String: Any]]?
func first<T>(arr: [T]) -> T?
```

```
parseJSON *> first *> KnowledgeBase.init *> Person.init
```

# OPTICS HIERARCHY



# ISOMORPHISM



# ISO

```
struct Iso<A, B> {
    let get: (A) -> B
    let reverseGet: (B) -> A
}
```

# ISO

```
let olegToMonkey = Iso<👳, 🙈>(
    get: { oleg in return oleg.bananify() },
    reverseGet: { monkey in return monkey.iOSify() }
)
```

# ISO

```
class PersonManagedObject: NSManagedObject {
    @NSManaged var name: String!
    @NSManaged var skills: String!
    @NSManaged var experience: Int!
}
```

# ISO

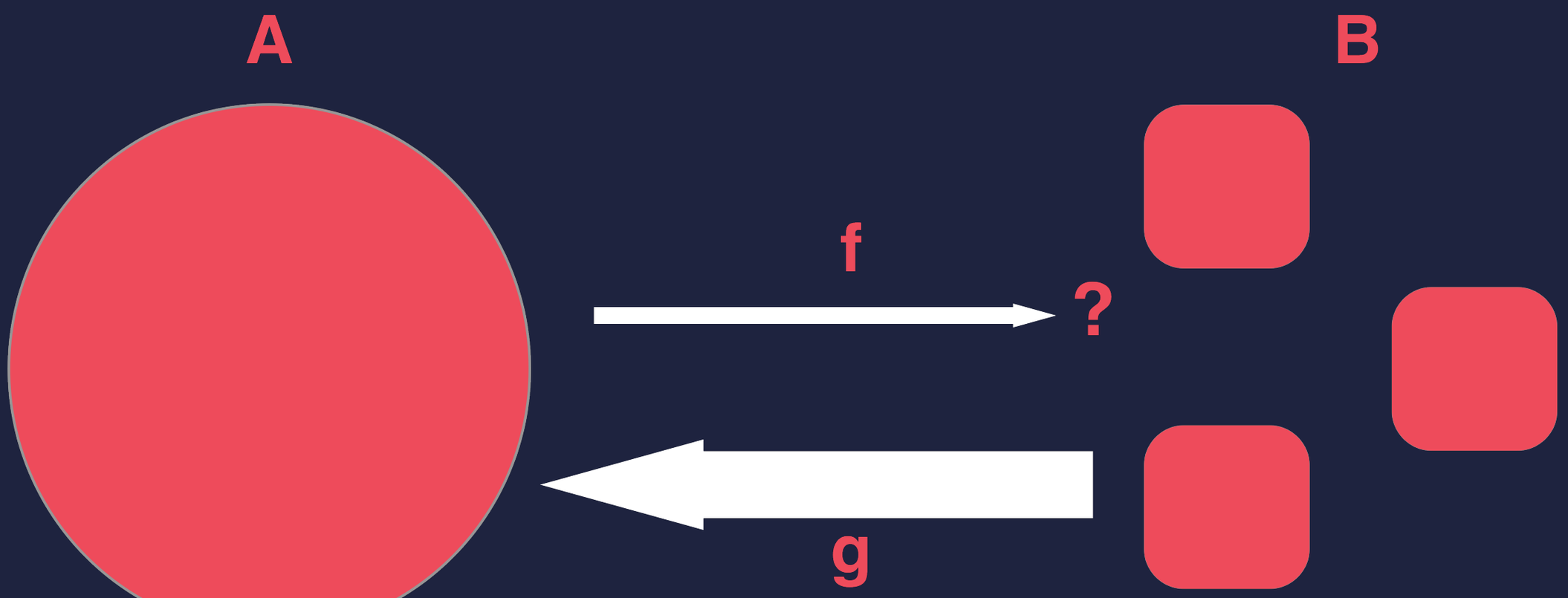
```
let personToManagedObject = Iso<Person, PersonManagedObject>(
    get: { PersonManagedObject(name: $0.name,
                               skills: $0.knowledge.skills,
                               experience: $0.knowledge.experience) },
    reverseGet: { Person(name: $0.name, knowledge: ...) }
)
```

```
prefix func ~ <A, B>(iso: Iso<A, B>) -> Iso<B, A>
func * <A, B, C>(lhs: Iso<A, B>, rhs: Iso<B, C>) -> Iso<A, C>
```

```
let managedObjectToMonkey = ~personToManagedObject * personToMonkey

// Iso<PersonManagedObject, Person>  *  Iso<Person, 🙈>
```

```
let managedObjectToMonkey = ~personToManagedObject * personToMonkey

// Iso<PersonManagedObject, Person>  *  Iso<Person, 🙈>

// Iso<PersonManagedObject, 🙈>
```

# PRISM



# PRISM

```
struct Prism<A, B> {
    let tryGet: (A) -> B?
    let reverseGet: (B) -> A
}
```

```
let olegToCool = Prism<👳, 😎>(
    get: { oleg in oleg.wearGlasses() }, // Cool ???
    set: { coolOleg in coolOleg.removeGlasses() }
)
```

# PRISM

```
func toJSON(data: Data) -> Any?

let dataToJSON = Prism<Data, Any>(
    tryGet: {
        try? JSONSerialization.jsonObject(with: data,
                                          options: .none)
    },
    reverseGet: {
        try! JSONSerialization.data(withJSONObject: $0,
                                    options: .none)
    }
)
```

# PRISM

```
func convert(...) // Any -> Array<Any>?

func fromAny<T>() -> Prism<Any, T> {
    return Prism<Any, T>(
        tryGet: { $0 as? T },
        reverseGet: { $0 as Any }
    )
}
```

# PRISM

```
func first<T>(_ arr: [T]) -> T?

func first<T>() -> Prism<[T], T> {
    return Prism<[T], T>(
        tryGet: { $0.first },
        reverseGet: { [$0] }
    )
}
```

# PRISM

```
let dictionaryToPerson = Prism<[String: Any], Person>(
    get: {
        guard let person = $0["oleg"] as? [String: Any],
            let experience = person?["experience"] as? Int,
            let skills = person?["skills"] as? [Skill],
            let mainSkill = skills?.first.flatMap { Skill(rawValue: $0) } else {
            return nil
        }
        return Person(name: "Oleg", skill: mainSkill, experience: experience)
    },
    reverseGet: {
        ["oleg": ["experience": $0.knowledge.experience, "skills": [$0.knowledge.skill]]]
    }
)
```

# PRISM

```
func * <A, B, C>(lhs: Prism<A, B>, rhs: Prism<B, C>) -> Prism<A, C>
func * <A, B, C>(lhs: Prism<A, B>, rhs: Iso<B, C>) -> Prism<A, C>
```

# Prism<Data, Any>

```
let dataToPerson = dataToAny
```

# Prism<Data, _>

```
let dataToPerson = dataToAny * fromAny()
```

# Prism<Data, _>

```
let dataToPerson = dataToAny * fromAny() * first()
```

# Prism<Data, Person>

```
let dataToPerson = dataToAny * fromAny() * first() * dictionaryToPerson
```

# Prism<Data, Person>

```
let dataToPerson = dataToAny * fromAny() * first() * dictionaryToPerson

let person = dataToPerson.get(jsonData)
print(person)
// "Person(name: "Oleg", knowledge: ...)\n"
```

# Prism<Data, Person>

```
let data = dataToPerson.reverseGet(address!)

String(data: data, encoding: .utf8)
// [{"oleg":{"experience":10,"skills":["iOS"]}}]
```

# LENS

```
struct Lens<Whole, Part> {
    let get: (Whole) -> Part
    let set: (Part, Whole) -> Whole
}
```

# LENS

```
let olegToKnowledge = Lens<Person, KnowledgeBase>(
    get: { oleg in oleg.knowledge },
    set: { newKnowledge, oleg in Person(name: oleg.name, knowledge: newKnowledge) }
)
```

# LENS

```
let olegToKnowledge = Lens<Person, KnowledgeBase>(
    get: { oleg in oleg.knowledge },
    set: { newKnowledge, oleg in Person(name: oleg.name, knowledge: newKnowledge) }
)
```

```
let knowledgeToSkills = Lens<Knowledge, Skill>(
    get: { knowledge in knowledge.skills },
    set: { newSkills, knowledge in
        Knowledge(experience: knowledge.experience, skills: newSkills)
    }
)
```

# LENS

```
let olegSkills = olegToKnowledge * knowledgeToSkills

olegSkills.get() // .iOS
olegSkills.set(oleg, .android) // Person("Oleg", .android)
```

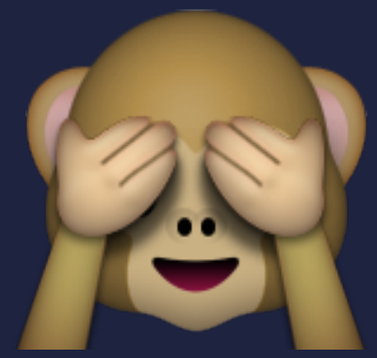
# OPTICS

```
appEnvironment
    * Application.topViewController
    * ViewController.collectionView
    * CollectionView.cells
    * first
    * Cell.title
    < "New title"

// AppEnvironment(...)
```

# WHAT NOW?

# SOURCEKIT

# SOURCEKITTEN

# THANK YOU! 

GITHUB.COM/S2DENTIK

TWITTER.COM/CULEVAALEXANDRU

CULEVALEX@GMAIL.COM