

Macoun

Neues in Swift

Nikolaj Schumacher

Ablauf

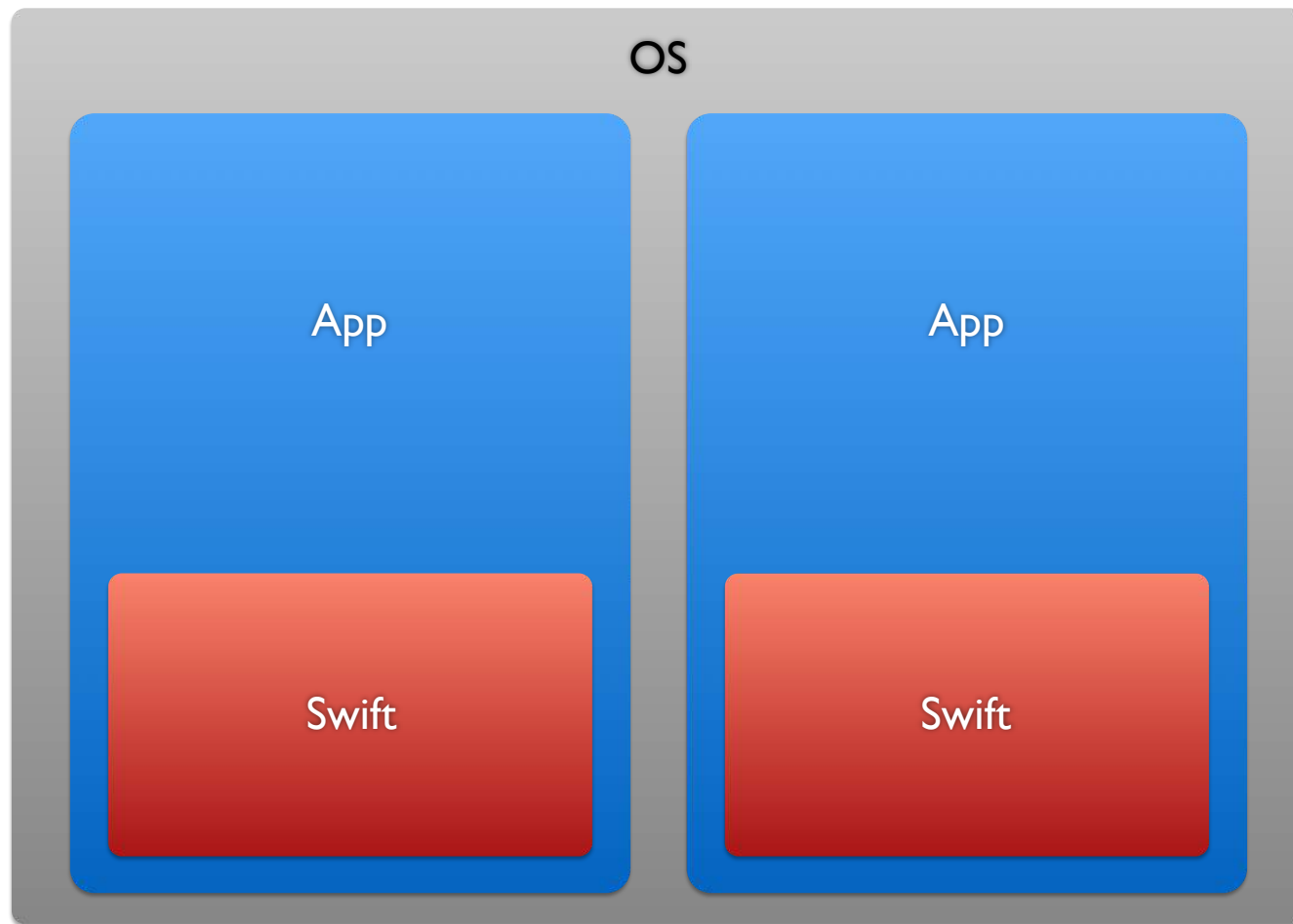
- ABI-Stabilität
- Property Wrapper
- Function Builder
- Strings
- Typen
- dynamische Features
- Collections
- Blitzrunde

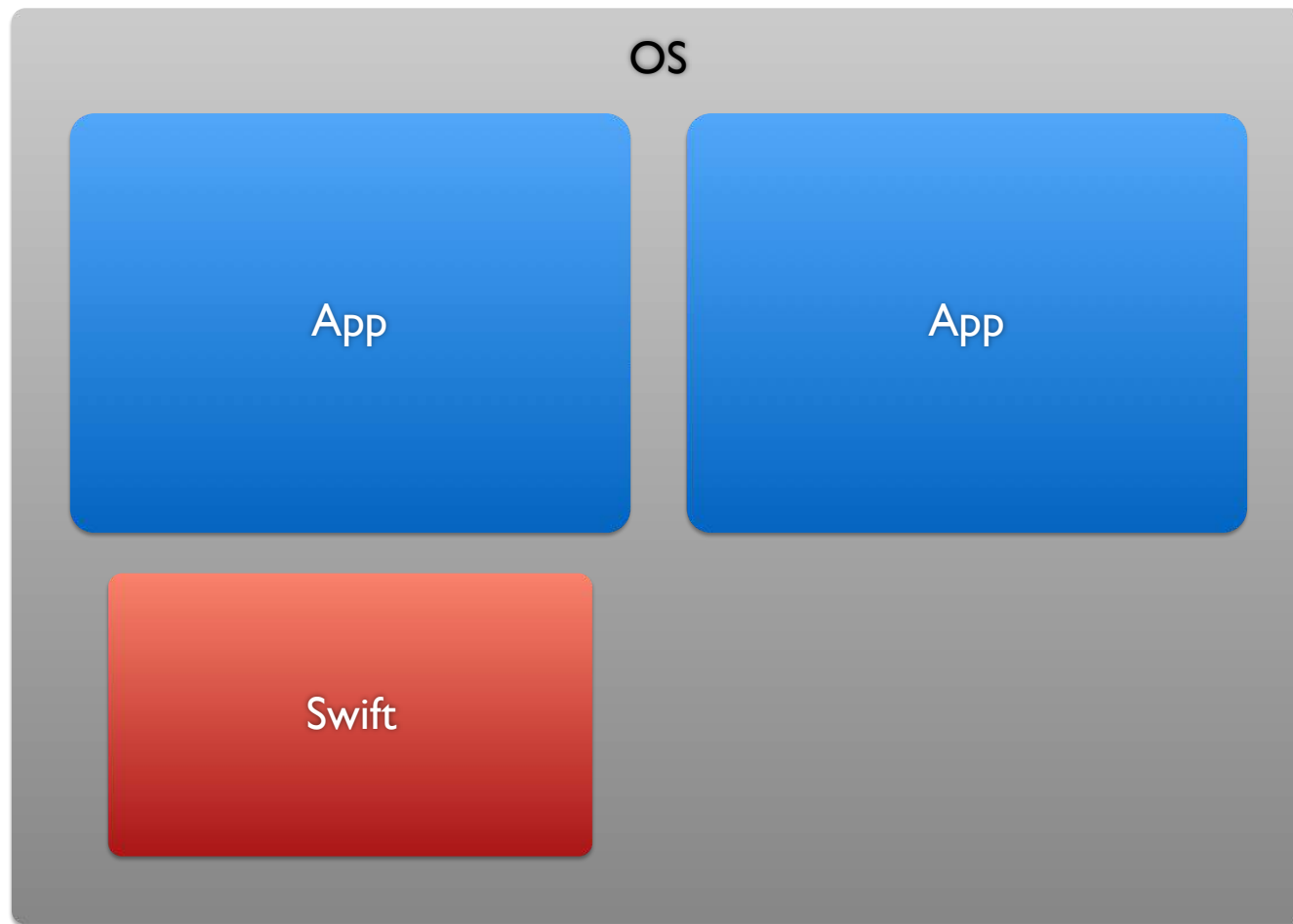
ABI-Stabilität

Was ist ABI?

- Application Binary Interface
- Call Conventions
- Metadata

Vorteile





Nachteile

iOS 11

App

Swift 4.2

App

Swift 4.0



OS 12.2

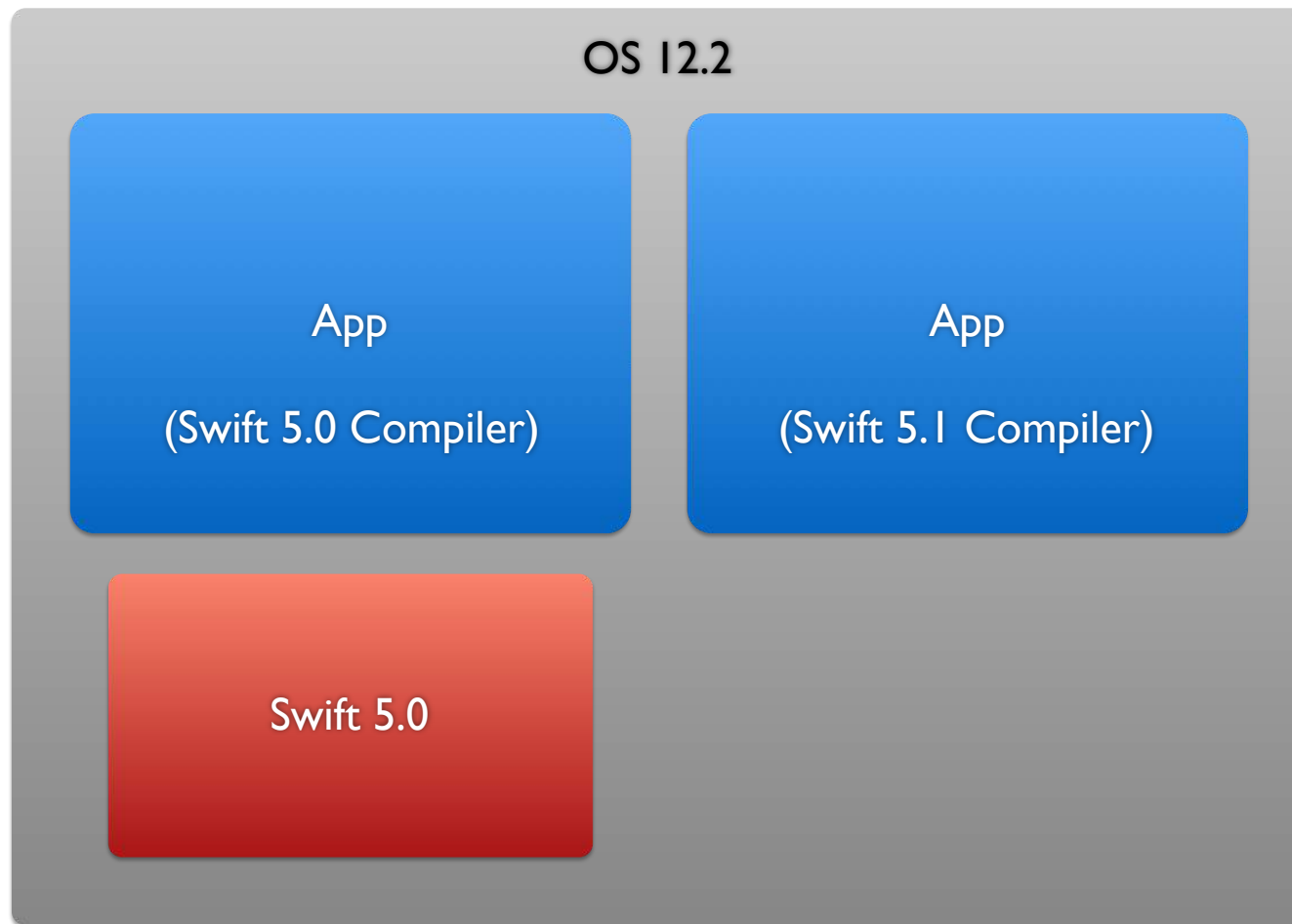
App

(Swift 5.0 Compiler)

App

(Swift 5.1 Compiler)

Swift 5.0



Nachteile ABI-Stabilität

- nur bestimmte Swift Features verfügbar
- Compiler-Features 👍
- Standardbibliothek 👎

Swift 5.0

Swift 5.1

nur Swift 5.1

Modul-Stabilität

- seit Swift 5.1
- Abwärtskompatibilität
- für binäre Frameworks

Beispiele

- Property-Typ ändern ✗
- Property zu Struct hinzufügen ✓
- Größe des Typs zur Laufzeit bestimmt

@frozen

SE-0260

Swift 5.1

- struct oder enum
- Property zu Struct hinzufügen ✗
- Property-Typ ändern ✗
- Größe des Typs bleibt gleich (optimierbar)

zukunftsicheres Switch

SE-0192

Swift 5.0

```
enum Kuchen {  
    case kirsch  
    case streusel  
  
    // + zukünftige Cases  
}
```

```
func iss(kuchen: Kuchen) {  
  
    switch kuchen {  
        case .kirsch:  
            print("Kirsch")  
        case .streusel:  
            print("Streusel")  
        default:  
            print("unbekannt")  
    }  
}
```


zukunftsicheres Switch

```
enum Kuchen {  
    case kirsch  
    case streusel  
  
    // + zukünftige Cases  
}
```

```
func iss(kuchen: Kuchen) {  
  
    switch kuchen {  
        case .kirsch:  
            print("Kirsch")  
        case .streusel:  
            print("Streusel")  
        @unknown default:  
            print("unbekannt")  
    }  
}
```

Property Wrapper

SE-0158

Swift 5.1

Motivation

Wiederholten Code beim Zugriff auf Properties vermeiden

Motivation

```
struct S {  
    var startDate: Date { return Date(timeIntervalSince1970: _startDate) }  
    private var _startDate: Double  
  
    var endDate: Date { return Date(timeIntervalSince1970: _endDate) }  
    private var _endDate: Double  
}
```

Motivation

```
struct S {  
    var startDate: Date { return Date(timeIntervalSince1970: _startDate) }  
    private var _startDate: Double  
  
    var endDate: Date { return Date(timeIntervalSince1970: _endDate) }  
    private var _endDate: Double  
}
```

Beispiel User Defaults

```
struct LoginSettings {  
    @UserDefaults("stay_logged_in", default: false)  
    var stayLoggedIn: Boolean  
}
```

Beispiel User Defaults

```
@propertyWrapper
struct UserDefaults<T> {
    let key: String
    let `default`: T

    var wrappedValue: T {
        get { UserDefaults.standard.object(forKey: key) as? T ?? self.default }
        set { UserDefaults.standard.set(newValue, forKey: key) }
    }
}
```

Beispiel User Defaults

```
struct LoginSettings {  
    @UserDefaults("stay_logged_in", default: false)  
    var stayLoggedIn: Boolean  
  
    func printSettings() {  
        print("\(stayLoggedIn.key): \(stayLoggedIn)")  
    }  
}
```

UserDefaults<Bool>

Bool

Arbeit für den Compiler

```
let loginSettings = LoginSettings()  
print(loginSettings.stayLoggedIn)
```

```
let loginSettings = LoginSettings()  
print(loginSettings  
      ._stayLoggedIn  
      .wrappedValue)
```

LoginSettings

stayLoggedIn: Bool

property

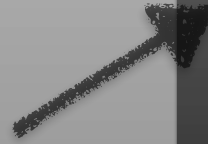
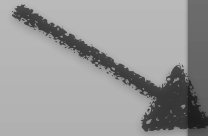
\$stayLoggedIn

projected value

private

_stayLoggedIn: UserDefault

backing storage



Projected Value

```
@propertyWrapper
struct UserDefault<T> {
    ...

    var projectedValue: Self {
        return self
    }
}
```

Projected Value

```
@propertyWrapper struct UserDefault<T> {  
    ...  
    class Projection {  
        ...  
        let key: String  
        func addObserver() {...}  
    }  
  
    var projectedValue: Projection {  
        return ProjectedValue(self)  
    }  
}
```

Projected Value

```
let loginSettings = LoginSettings()  
  
loginSettings.$stayLoggedIn.addObserver {  
    print("\(stayLoggedIn.key) changed to \(stayLoggedIn)")  
}
```



UserDefaults.Projection



Bool

Beispiel IUO

```
@propertyWrapper class LateInit<T> {  
    var realValue: T? = nil  
  
    var wrappedValue: T {  
        get {  
            guard let realValue = realValue else { fatalError() }  
            return realValue  
        }  
        set { realValue = newValue}  
    }  
}
```

weitere Beispiele

- Lazy
- Atomarer Variablen
- Thread-lokale Variablen
- Copy-on-Write
- Validierung
- Observer/Subscribe

weitere Beispiele

SwiftUI

Function Builder

Swift 5.1*

Swift 5.2?

*experimentell

Function Builder

```
func build(@MyBuilder builder: () -> Int) -> [Int] {...}
```

```
let array = build {
```

```
1
```

```
2
```

```
3
```

```
random()
```

```
}
```



unused expression



missing return

Arbeit für den Compiler

```
let array = build {  
  1  
  2  
  3  
  random()  
}
```

```
let array = build {  
  return MyBuilder.buildBlock(  
    1,  
    2,  
    3,  
    random()  
  )  
}
```

Strings

UTF-8

Swift 5.0

- Strings nutzen intern UTF-8 statt UTF-16
- weniger Konvertierung (C-Aufrufe)

Raw Strings

SE-0200

Swift 5.0

```
let path = "c:\\komische\\Windows\\Pfade\\"  
let re = NSRegularExpression(pattern: "\\b\\w*-\\d*\\b", options: [])  
let swift = "let swift = \"swift\""
```

Lösung

Escape-Level erhöhen

Raw Strings

```
let path = #"c:\komische\Windows\Pfade\"#  
let re = NSRegularExpression(pattern: #"\\b\\w*\\d*\\b"#, options: [])  
let swift = #"let swift = "swift" "#
```


Raw Strings

```
let swift = ##"let swift = #"swift"# "##
```

```
let swift = #####"let swift = #"swift"# "####
```

```
let swift = #####" "  
let swift = #"swift"#  
"####
```

String Interpolationen

SE-0228

Swift 5.0

```
let time = TimeInterval(1.0 / 3.0 * 60.0 * 60.0)
```

```
print("Kirschkuchen in \(time) Stunden")
```

Kirschkuchen in 0.333333 Stunden

String Interpolationen

```
let time = TimeInterval(1.0 / 3.0 * 60.0 * 60.0)

extension TimeInterval {
    func toString() -> String {
        let formatter = RelativeDateTimeFormatter()
        formatter.locale = Locale(identifier: "de")
        let string = formatter.localizedString(fromTimeInterval: time)
    }
}

print("Kirschkuchen in \(time.toString())")
```

Kirschkuchen in 20 Minuten

```
func toString() -> String {
```

Lösung

Overloading für String-Interpolationen

String Interpolationen

```
extension DefaultStringInterpolation {  
    mutating func appendInterpolation(_ time: TimeInterval) {  
        let formatter = RelativeDateTimeFormatter()  
        formatter.locale = Locale(identifier: "de")  
        appendLiteral(formatter.localizedString(fromTimeInterval: time))  
    }  
}  
  
let time = TimeInterval(1.0 / 3.0 * 60.0 * 60.0)  
print("Kirschkuchen \$(time)")
```

String Interpolationen

```
extension DefaultStringInterpolation {  
    mutating func appendInterpolation(time: TimeInterval, locale: String) {  
        let formatter = RelativeDateTimeFormatter()  
        formatter.locale = Locale(identifier: locale)  
        appendLiteral(formatter.localizedString(fromTimeInterval: time))  
    }  
}  
  
let time = TimeInterval(1.0 / 3.0 * 60.0 * 60.0)  
print("Kirschkuchen \$(time: time, locale: "de")")
```

Typen

Result

SE-0235

Swift 5.0

- ähnlich zu Optional
- expliziter Fehlerursache statt nil

Result

```
enum Result<Success, Failure> where Failure : Error {  
    case success(Success)  
    case failure(Failure)  
}
```

Result

```
enum HttpError {  
    case notFound  
    case notAuthorized  
}  
  
typealias HttpResult = Result<String, HttpError>  
  
func fetch(url: NSURL) -> HttpResult {...}
```

Opaque Typen

nur Swift 5.1

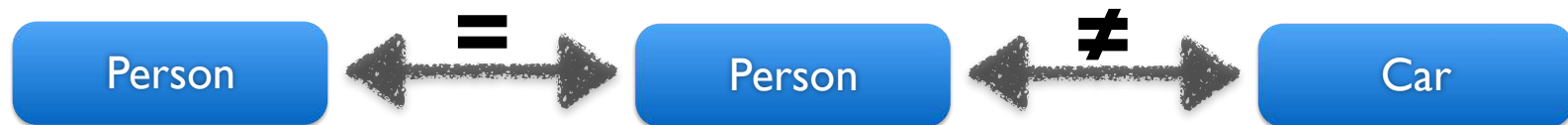
SE-0244

Protocol 'Equatable' can only be used as a generic constraint because it has Self or associated type requirements

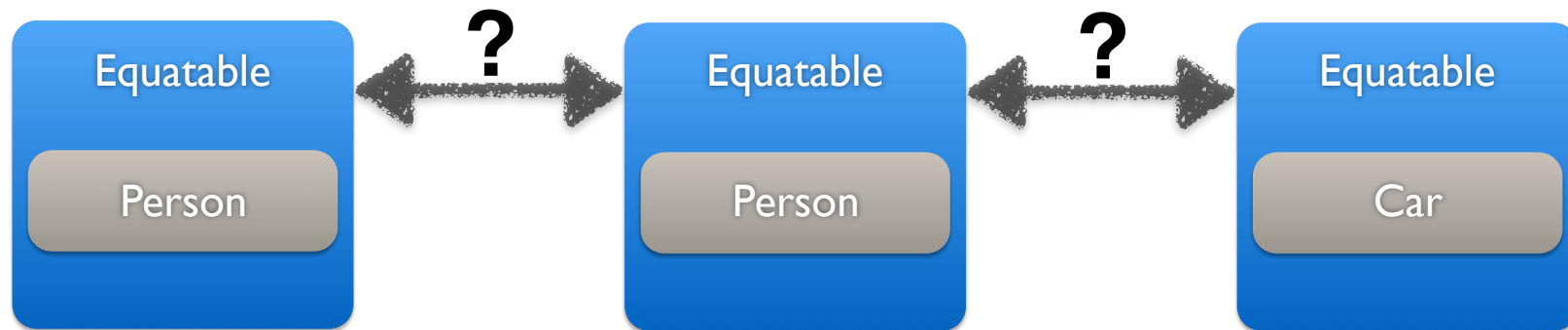
Opaque Typen

```
struct Person {  
    var name: String  
  
    var id: Equatable {  
        return name  
    }  
}
```

Equatable



Equatable



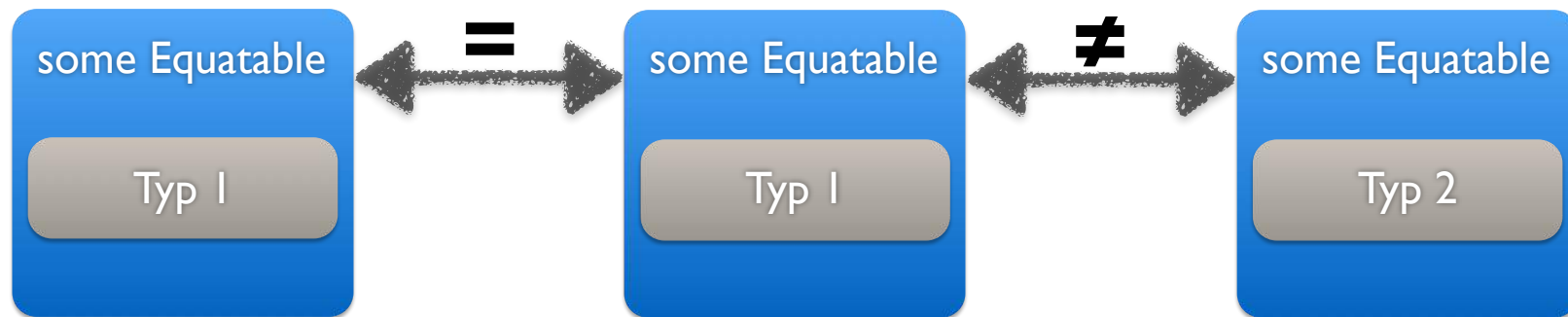
some

- garantiert gleichen konkreten Type
- (aber nicht für zukünftige Versionen)
- tatsächlicher Typ unbekannt

Opaque Typen

```
struct Person {  
    var name: String  
  
    var id: some Equatable {  
        return name  
    }  
}
```


some Equatable



dynamische Features

KeyPath member lookup

SE-0252

Swift 5.1

- dynamisch Properties zu Typen hinzufügen
- Erweiterung zu `@dynamicMemberLookup` (Swift 4.2)
- Key-Path statt String

```
struct Person {  
    let name: String  
}  
  
struct Car {  
    let licensePlate: String  
}
```

```
struct Person { let name: String }  
struct Car { let licensePlate: String }
```

```
@dynamicMemberLookup
```

```
struct TimeTraveler<T> {  
    let traveler: T
```

```
    subscript<U>(dynamicMember keyPath: KeyPath<T, U>) -> U {  
        traveler[keyPath: keyPath]  
    }  
}
```

```
struct Person { let name: String }
struct Car { let licensePlate: String }

@dynamicMemberLookup
struct TimeTraveler<T> {
    let traveler: T

    subscript<U>(dynamicMember keyPath: KeyPath<T, U>) -> U {
        traveler[keyPath: keyPath]
    }
}

let person = TimeTraveler(traveler: Person(name: "Marty"))
print(person.name)

let car = TimeTraveler(traveler: Car(licensePlate: "OUTATIME"))
print(car.licensePlate)
```

Arbeit für den Compiler

```
person.name
```

```
person[dynamicMember: \Person.name]
```

@dynamicCallable

SE-0216

Swift 5.0

- Typen können aufgerufen werden
- wie Closures

@dynamicCallable

```
@dynamicCallable
struct Callable {

    func dynamicallyCall(withArguments args: [Int]) {
        ...
    }

    func dynamicallyCall(withKeywordArguments args: KeyValueCollection<String, Int>) {
        ...
    }

}
```

Arbeit für den Compiler

```
foo(arg1: 1, arg2: 2)
```

```
foo(1, 2, 3)
```

```
foo.dynamicallyCall(  
  withKeywordArguments: [  
    "arg1": 1,  
    "arg2": 2  
  ]  
)  
  
foo.dynamicallyCall(withArguments:  
  [1, 2, 3]  
)
```

Collections

entfernte Features

- Int statt IndexOffset
- SubSequence
- customization points

SE-0241

SE-0234

SE-0232

Diff

SE-0240

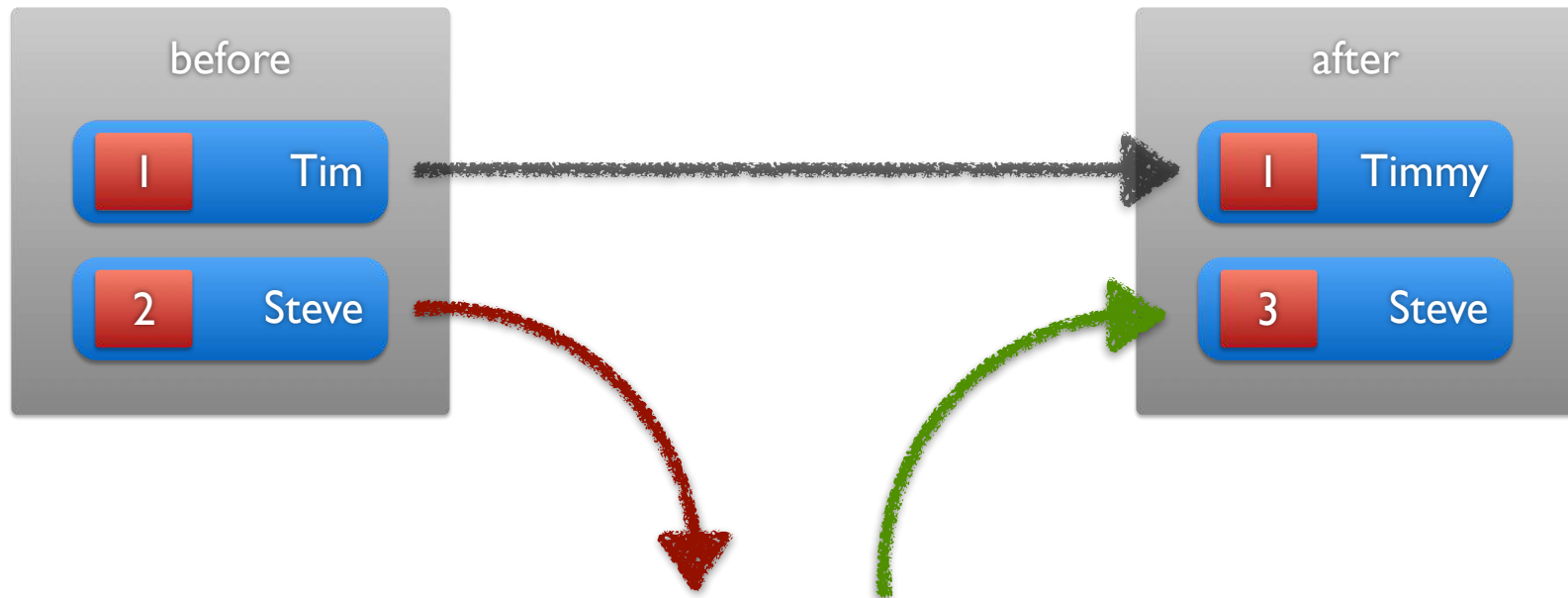
nur Swift 5.1

```
struct Person{
    var id: Int
    var name: String
}
let before = [
    Person(id: 1, name: "Tim"),
    Person(id: 2, name: "Steve")
]
let after = [
    Person(id: 1, name: "Timmy"),
    Person(id: 3, name: "Steve")
]
```

```
let diffs =
    after.difference(from: before) {
        $0.id == $1.id
    }

switch diff.first() {
case let .insert(index, element, _):
    // eingesetzt
case let .remove(index, element, _):
    // entfernt
}
```

Diff



Diff

```
struct Person : Identifiable {  
    var id: Int  
    var name: String  
}  
let before = [  
    Person(id: 1, name: "Tim"),  
    Person(id: 2, name: "Steve")  
]  
let after = [  
    Person(id: 1, name: "Timmy"),  
    Person(id: 3, name: "Steve")  
]
```

```
let diffs =  
    after.difference(from: before) {  
        $0.id == $1.id  
    }  
  
before.applying(diffs)  
// == after  
  
after.applying(diffs.inverse())  
// == before
```

Kleinigkeiten

- schnellere, unsichere Initialisierung
- compactMapValues
- KeyValuePairs statt DictionaryLiteral

SE-0245

nur Swift 5.1

SE-0218

Swift 5.0

SE-0214

Swift 5.0

Blitzrunde

implicit Returns

SE-0255

Swift 5.1

```
struct Angle {  
    var degrees: Double  
    var radians: Double { return degrees / 180.0 * .pi }  
}
```

SIMD

SE-0251

nur Swift 5.1

```
let x = SIMD4<Int>(1, 2, 3, 4)
let y = SIMD4<Int>(1, 2, 2, 4)

let isEqual: SIMDMask<SIMD4<Int.SIMDMaskScalar>> = x .== y
```

Struct Init mit Default

SE-0242

Swift 5.1

```
struct WallClock {  
    var hours: Int  
    var minutes: Int  
    var seconds: Int = 0  
    var milliseconds: Int = 0  
}
```

```
WallClock(hours: 12, minutes: 30, seconds: 30, milliseconds: 500)  
WallClock(hours: 12, minutes: 30, seconds: 30)  
WallClock(hours: 12, minutes: 30)
```

statische Subscripts

SE-0254

Swift 5.1

```
enum Kuchen: CaseIterable {  
    case kirsch  
  
    static subscript(index: Int) -> Kuchen? {  
        return self.allCases[index]  
    }  
}
```

Kuchen[0]

Self expression

SE-0068

Swift 5.1

```
class C {  
    class func staticFunc() {}  
  
    func instanceFunc() {  
        type(of: self).staticFunc()  
    }  
}
```

```
class C {  
    class func staticFunc() {}  
  
    func instanceFunc() {  
        Self.staticFunc()  
    }  
}
```

Identity Key Path

SE-0227

Swift 5.0

```
\.self
```

Identifiable

SE-0261

nur Swift 5.1

```
public protocol Identifiable {  
    associatedtype ID : Hashable  
  
    var id: Self.ID { get }  
}
```


Codable Ranges

SE-0239

Swift 5.0

```
extension Range: Codable  
  
extension ClosedRange: Codable  
  
extension PartialRangeFrom: Codable  
  
extension PartialRangeThrough: Codable  
  
extension PartialRangeUpTo: Codable
```

Flatten try?

SE-0230

Swift 5.0

```
func doSomething() -> Int?? {  
    return try? getOptional()  
}
```

```
func doSomething() -> Int? {  
    return try? getOptional()  
}
```

BinaryInteger.isMultiple

SE-0225

Swift 5.0

```
42.isEven
```

```
43.isOdd
```

```
42.isMultiple(of: 3)
```

#if swift<

SE-0224

Swift 5.0

```
#if swift(>=5.1)
#else
    #warning("Swift zu alt")
#endif
```

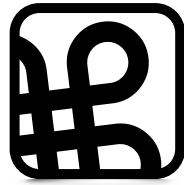
```
#if swift(<5.1)
    #warning("Swift zu alt")
#endif
```

Fragen?



<https://github.com/nschum/talks>

Vielen Dank



Macoun