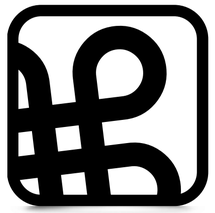


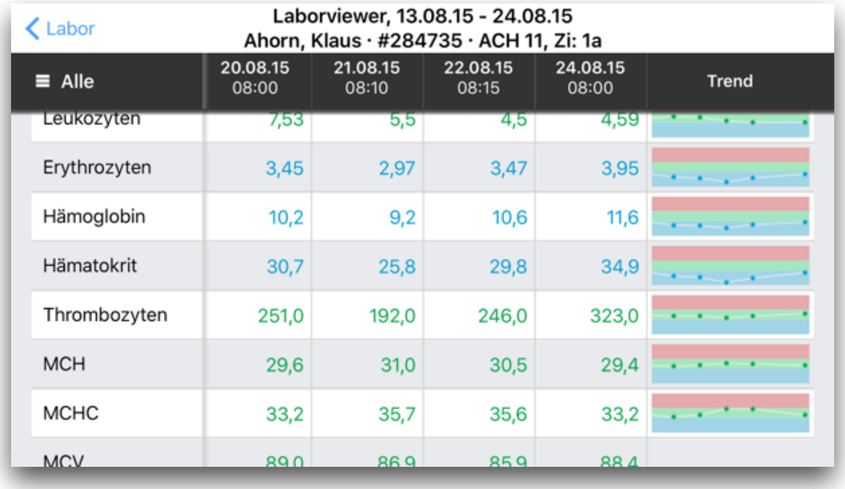
Macoun

Funktionale Programmierung mit Swift

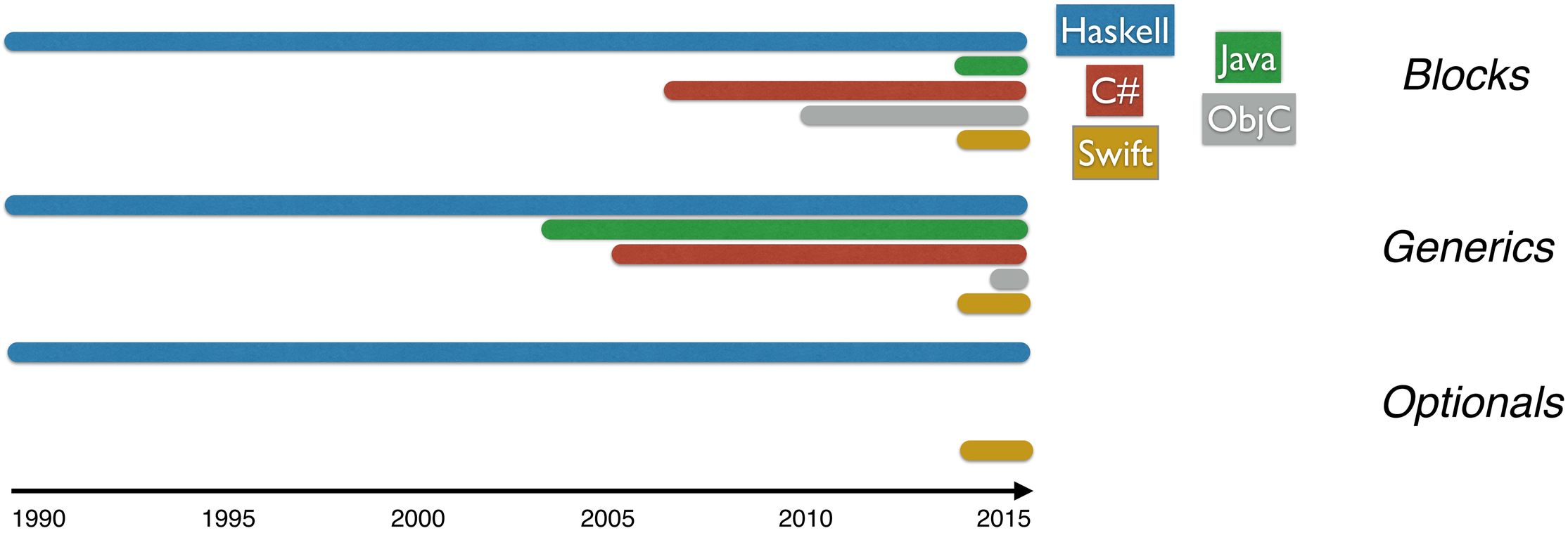
Stefan Wehr
factis research GmbH
factisresearch.com / cpmed.de
wehr@cp-med.com

Vorstellung

- Checkpad MED App (seit 2010)
  - mobile Patientenakte
  - Client ~80.000 Zeilen ObjC Code
  - Backend in Haskell
  - http://factisresearch.com / http://cpmed.de
- Funktionaler Programmierer seit 2003
  - Blog: http://funktionale-programmierung.de
  - Konferenz: http://bobkonf.de



Funktionale Programmierung wird Mainstream



Was ist Swift?

Objective-C without the C

Was ist Swift?

~~Objective-C without the C~~

Objective-C with functional programming

Was ist funktionale Programmierung?

unveränderliche Datenstrukturen

statische Typen

Lambda-Kalkül

Generics

formalisierte Semantik

Blocks

Typinferenz

Metaprogrammierung

Mathematik

Automatische Speicherverwaltung

Pattern Matching

interaktive Programmierumgebung

Monaden

Was ist funktionale Programmierung?

unveränderliche Datenstrukturen

statische Typen

Lambda-Kalkül

Generics

formalisierte Semantik

Blocks

Typinferenz

Metaprogrammierung

Mathematik

Automatische Speicherverwaltung

Pattern Matching

interaktive Programmierumgebung

Monaden

Soll ich mich mit funktionaler Programmierung beschäftigen?

Soll ich mich mit funktionaler Programmierung beschäftigen?

Kannst du es dir leisten, dies nicht zu tun??

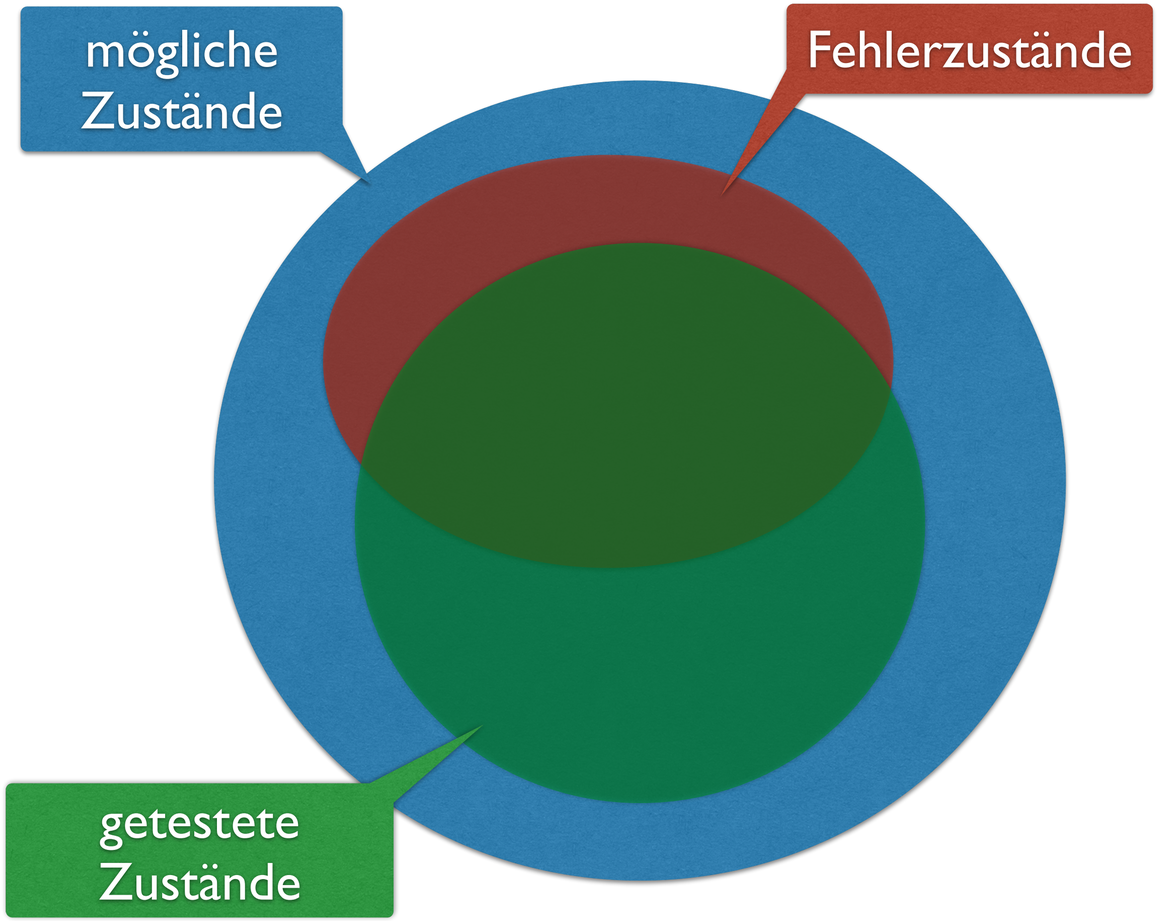
Prominente Benutzer

- Twitter: Scala
- Facebook: Haskell
- Microsoft: F#
- Apple: Swift
- Ericsson: Erlang
- Whatsapp: Erlang

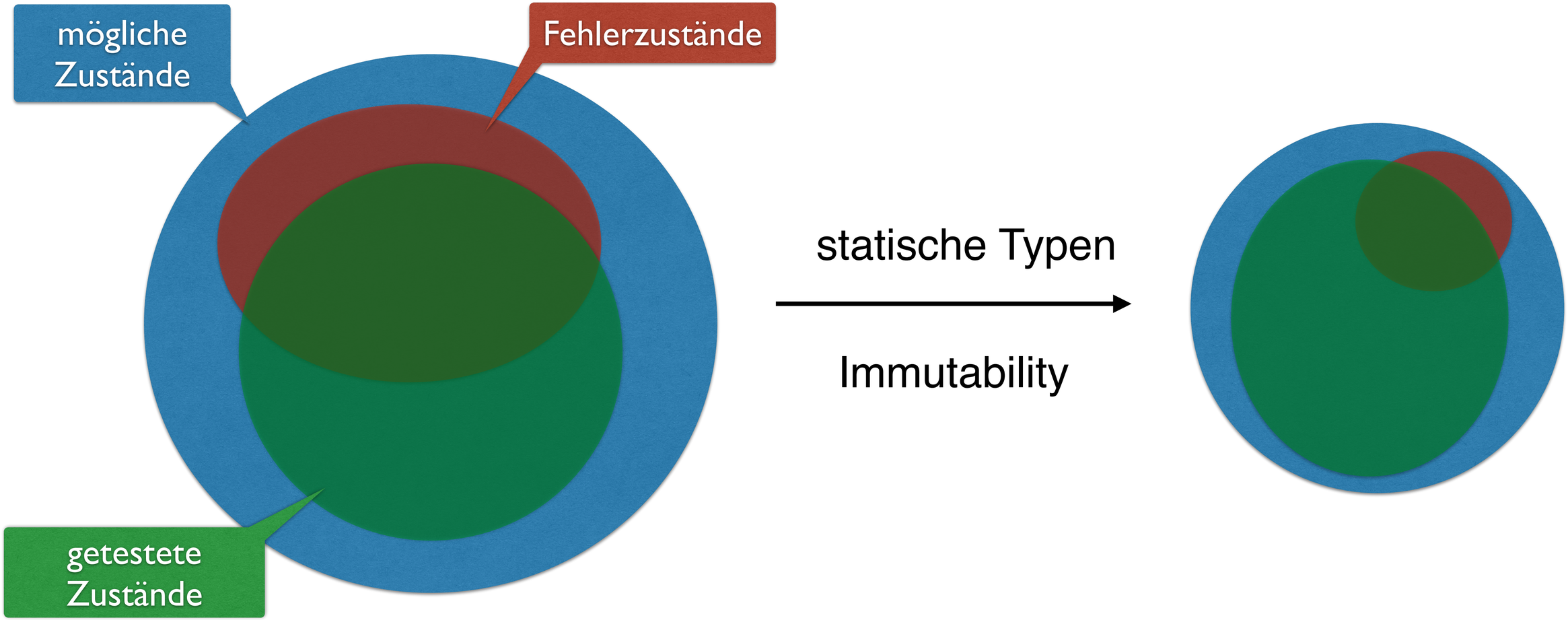
Vorzüge funktionaler Programmierung

- Disziplinierter Umgang mit Seiteneffekten
- Ausdrucksstarkes Typsystem: wenn das Programm kompiliert funktioniert es auch!
- Gute Testbarkeit
- Kurzer, prägnanter und lesbarer Code
- Einfaches Abstrahieren, hoher Wiederverwendungsgrad

Testbarkeit



Testbarkeit



Statische Typen

```
@interface Registry : NSObject
- (NSInteger)getValue:(NSString *)key;
@end

@implementation Registry {
    @private NSMutableDictionary *dictionary;
}

- (NSInteger)getValue:(NSString *)key
{
    // hey, trust me: the dictionary only contains NSNumbers
    NSNumber *n = [self->dictionary valueForKey:key];
    return [n integerValue];
}

- (instancetype)init
{
    self = [super init];
    if (self) {
        self->dictionary = [NSMutableDictionary dictionary];
        [self->dictionary setObject:@1 forKey:@"one"];
        [self->dictionary setObject:@2 forKey:@"two"];

    return self;
}
@end
```

Statische Typen

```
@interface Registry : NSObject
- (NSInteger)getValue:(NSString *)key;
@end

@implementation Registry {
    @private NSMutableDictionary *dictionary;
}

- (NSInteger)getValue:(NSString *)key
{
    // hey, trust me: the dictionary only contains NSNumbers
    NSNumber *n = [self->dictionary valueForKey:key];
    return [n integerValu
}

- (instancetype)init
{
    self = [super init];
    if (self) {
        self->dictionary = [NSMutableDictionary dictionary];
        [self->dictionary setObject:@1 forKey:@"one"];
        [self->dictionary setObject:@2 forKey:@“two"];
        [self->dictionary setObject:@[@3] forKey:@“three"];

    }
    return self;
}
@end
```

Laufzeitfehler: unrecognized selector sent to instance 0x27dc108

Statische Typen

```objc
@interface Registry : NSObject
- (NSInteger)getValue:(NSString *)key;
@end

@implementation Registry {
    @private NSMutableDictionary *dictionary;
}

- (NSInteger)getValue:(NSString *)key
{
    // hey, trust me: the dictionary only contains NSNumbers
    NSNumber *n = [self->dictionary valueForKey:key];
    return [n integerValu
}

- (instancetype)init
{
    self = [super init];
    if (self) {
        self->dictionary = [NSMutableDictionary dictionary];
        [self->dictionary setObject:@1 forKey:@"one"];
        [self->dictionary setObject:@2 forKey:@“two"];
        [self->dictionary setObject:@[@3] forKey:@“three"];

    }
    return self;
}
@end
```

Laufzeitfehler: unrecognized selector sent to instance 0x27dc108

```swift
class Registry {
    private var dictionary: Dictionary<String, Int>

    func getValue(key: String) -> Int? {
        return self.dictionary[key]
    }

    init() {
        dictionary = Dictionary()
        dictionary["one"] = 1
        dictionary["two"] = 2
        dictionary["three"] = [3]
    }
}
```

Kompilierfehler

Immutability

```
class Temperature {
    var celsius: Double
    var fahrenheit: Double {
        get { return (celsius * 9 / 5  + 32) }
        set { celsius = (newValue - 32) * 5 / 9 }
    }
    init(celsius: Double) {
        self.celsius = celsius
    }
}

let house = House()
let temp = Temperature(celsius:21)
house.thermostat.temperature = temp
```

Immutability

```
class Temperature {
    var celsius: Double
    var fahrenheit: Double {
        get { return (celsius * 9 / 5  + 32) }
        set { celsius = (newValue - 32) * 5 / 9 }
    }
    init(celsius: Double) {
        self.celsius = celsius
    }
}

let house = House()
let temp = Temperature(celsius:21)
house.thermostat.temperature = temp

temp.celsius = 250
house.oven.temperature = temp
```

Jetzt wird's heiß!

```
class Temperature {
    let celsius: Double
    let fahrenheit: Double {
        get { return (celsius * 9 / 5  + 32) }
    }
    init(celsius: Double) {
        self.celsius = celsius
    }
}

let house = House()
house.thermostat.temperature = Temperature(celsius:21)
house.oven.temperature = Temperature(celsius:250)
```

Immutability

```
class Temperature {
    var celsius: Double
    var fahrenheit: Double {
        get { return (celsius * 9 / 5  + 32) }
        set { celsius = (newValue - 32) * 5 / 9 }
    }
    init(celsius: Double) {
        self.celsius = celsius
    }
}

let house = House()
let temp = Temperature(celsius:21)
house.thermostat.temperature = temp

temp.celsius = 250
house.oven.temperature = temp
```

Jetzt wird's heiß!

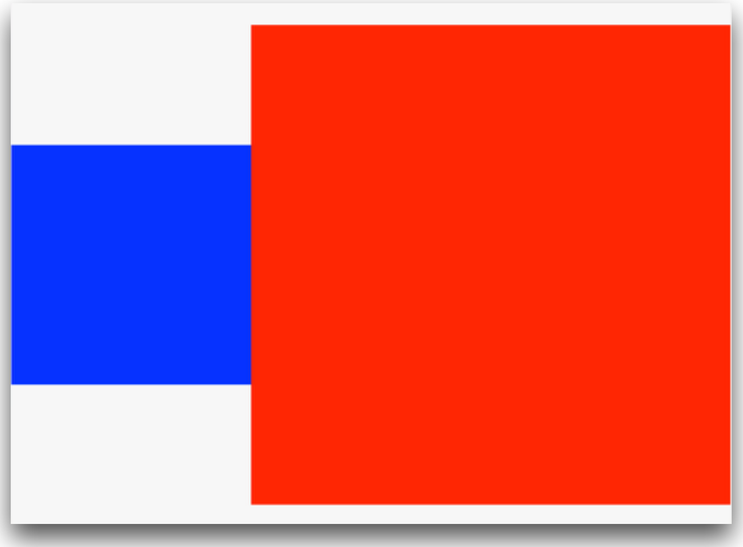
```
class Temperature {
    let celsius: Double
    let fahrenheit: Double {
        get { return (celsius * 9 / 5  + 32) }
    }
    init(celsius: Double) {
        self.celsius = celsius
    }
}

let house = House()
house.thermostat.temperature = Temperature(celsius:21)
house.oven.temperature = Temperature(celsius:250)
```

- WWDC 2015: Building Better Apps with Value Types in Swift
  "Swift supports rich first-class **value types** in the form of powerful structs, which provide new ways to architect your apps. Learn about the differences between reference and value types, how **value types help you elegantly solve common problems around mutability and thread safety**, and discover how Swift's unique capabilities might change the way you think about abstraction."
- WWDC 2014: Advanced iOS Application Architecture and Patterns
  "Learn how to manage complexity in large codebases by clearly defining where truth resides, by **controlling state with Swift's powerful value types and immutability**, and by **thinking in terms of composition**."

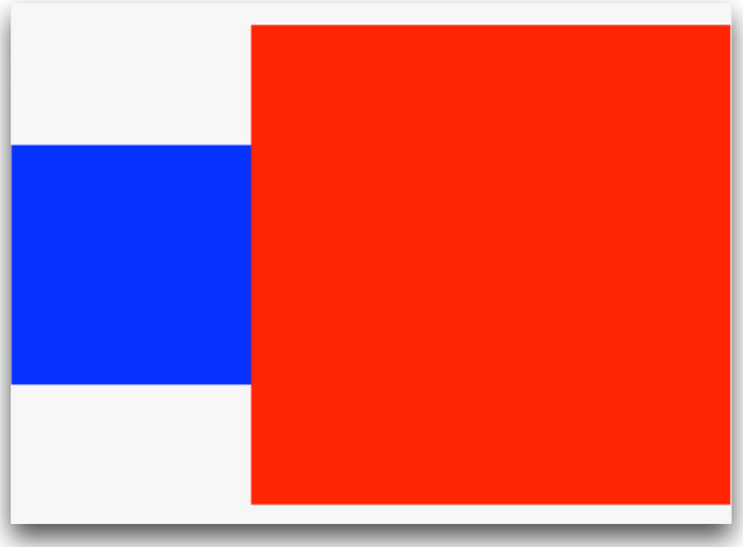
Demo Time!

Diagramme

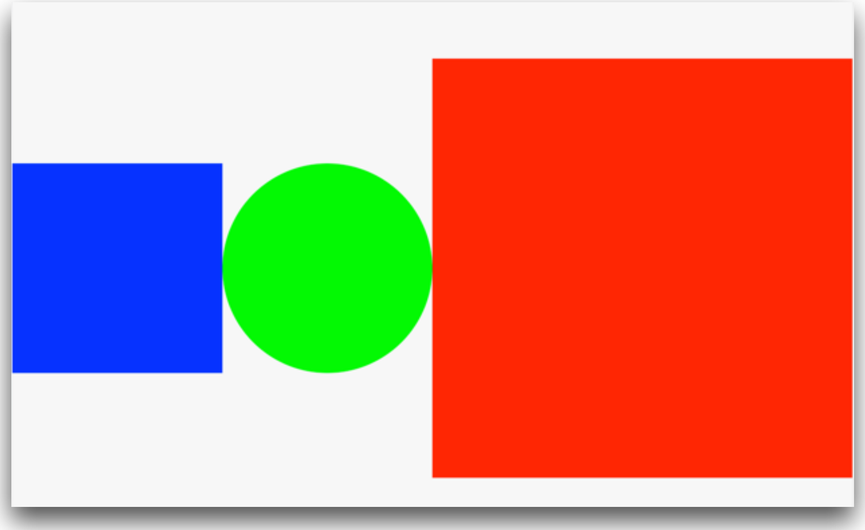


```
NSColor.blueColor().setFill()
CGContextFillRect(ctx,
    CGRectMake(0.0, 37.5, 75.0, 75.0)
NSColor.redColor().setFill()
CGContextFillRect(ctx,
    CGRectMake(75.0, 0.0, 150.0, 150.0))
```

Diagramme



```
NSColor.blueColor().setFill()
CGContextFillRect(ctx,
    CGRectMake(0.0, 37.5, 75.0, 75.0)
NSColor.redColor().setFill()
CGContextFillRect(ctx,
    CGRectMake(75.0, 0.0, 150.0, 150.0))
```



```
NSColor.blueColor().setFill()
CGContextFillRect(ctx, CGRectMake(0.0, 37.5, 75.0, 75.0))
NSColor.greenColor().setFill()
CGContextFillEllipseInRect(ctx,
    CGRectMake(75.0, 37.5, 75.0, 75.0))
NSColor.redColor().setFill()
CGContextFillRect(ctx, CGRectMake(150.0, 0.0, 150.0, 150.0))
```

Demo 1

Funktionale Diagramme (siehe Xcode Playground)

Buch: Functional Programming in Swift (2014)
Chris Eidhof, Florian Kugler, Wouter Swierstra

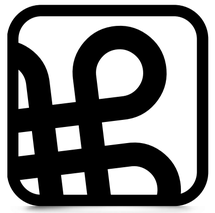
Zusammenfassung

- Die Zukunft ist funktional
- Statische Typen sind deine Freunde
- Immutability ist dein Freund
- Warum warten? Die Zukunft ist jetzt!

Funktionale Programmierung mit Swift

Fragen?

Stefan Wehr
factis research GmbH
wehr@cp-med.com



Macoun

Quellen

- Folie 3; Screenshots “Checkpad MED, iOS App”; factis research GmbH; Genemigung zur Verwendung der Screenshots im Rahmen dieses Vortrags liegt vor.

- Folie 4; Diagramme; Stefan Wehr; public domain.