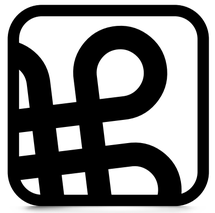
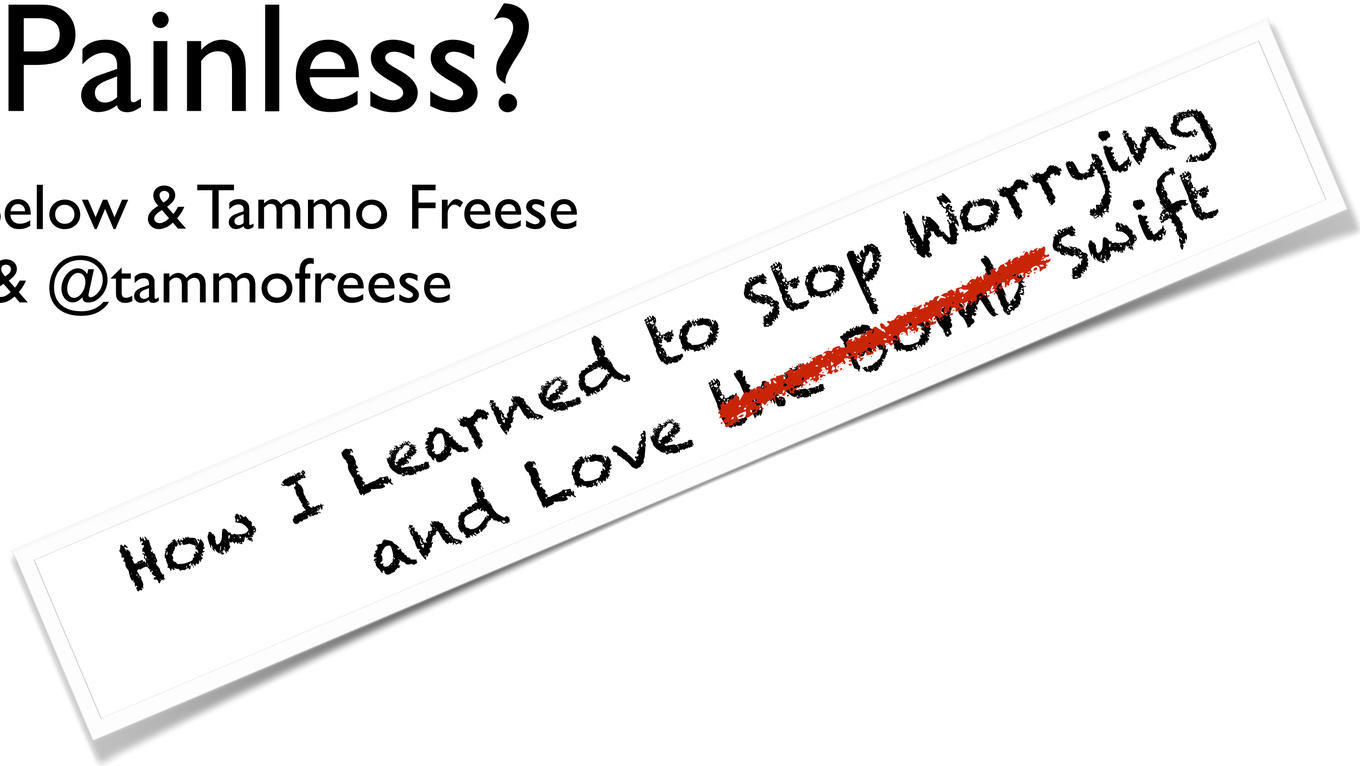


# Macoun

# It will be Swift!
# – and Painless?

Alexander von Below & Tammo Freese
@avbelow & @tammofreese



# It will be Swift

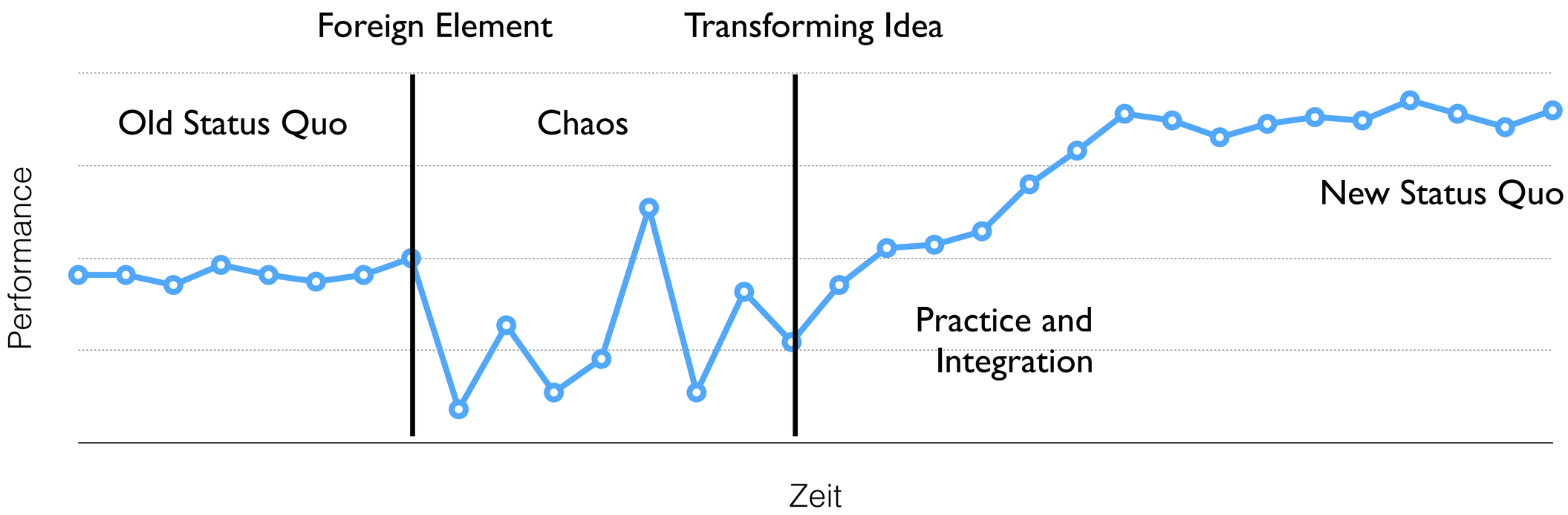
- Objective-C wird sterben
- Ernsthaft
- Xcode Release Notes: "Swift is a complete replacement for both the C and Objective-C languages“
- Und was ist mit C++?

# — and Painless?

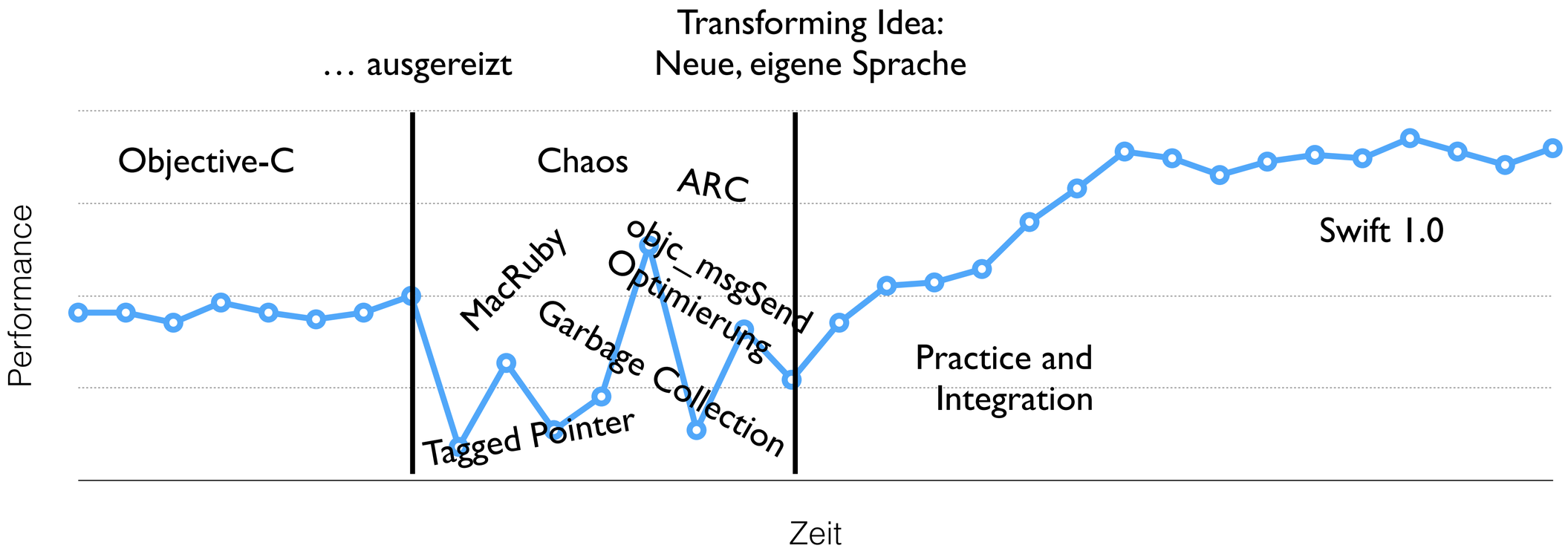
- Neues bringt Vorteile
- Vorteile machen Freude

- Neues bedeutet Umgewöhnung
- Umgewöhnung ist machmal schmerzhaft

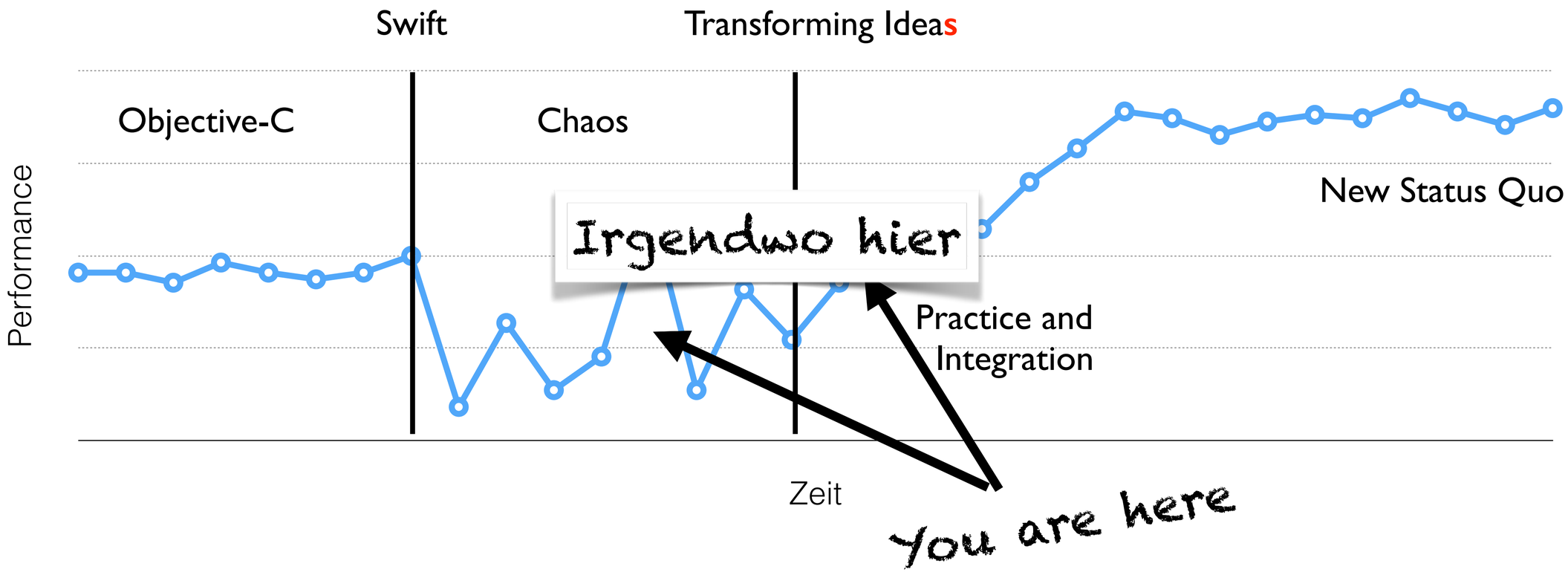
# Satir Change Model



# Swift aus Apples Sicht



# Swift aus unserer Sicht



# Inhalt

- Neue Dinge zeigen: Eine Auswahl
  - Optionale Werte, Strukturen, Aufzählungen, Dictionary, Eigene Operatoren, Funktionstypen, Generics
- Und was ist mit Dynamik?
- Ausklang

# Neue Dinge

# Optionale Werte

- Neu: Objektreferenz ist nicht optional
  `var view: UIView`
- Kann optional gemacht werden:
  `var view: UIView?`
- Auch für einfache Typen:
  `var delay: NSTimeInterval?`

# Optionale Werte: Vorteile

- Fehlervermeidung
- Ausdrucksmächtigkeit: Bessere APIs!

```
class TodoList {
    // ...
    func indexOfItem(item: Item) -> Int? {
    }
    // ...
}
```

# Optionale Werte verwenden

Optional Binding

Optional Chaining

```
if let naturalIndexOfItem = list.indexOfItem(item)?.successor() {
    println("Item \(naturalIndexOfItem)")
} else {
    println("Item not in list")
}
```

# Optionale Werte: Umgewöhnung

- Mehr Überlegen beim Code Design nötig: Optional oder nicht?
- In Objective-C: Spezialwerte wie NSNotFound (`Int.max`)

```
let list = [item] as NSArray

let indexOfItem = list.indexOfObject(item)
if indexOfItem != NSNotFound {
    let naturalIndexOfItem = indexOfItem.successor()
    println("Item \(naturalIndexOfItem)")
} else {
    println("Item not in list")
}
```

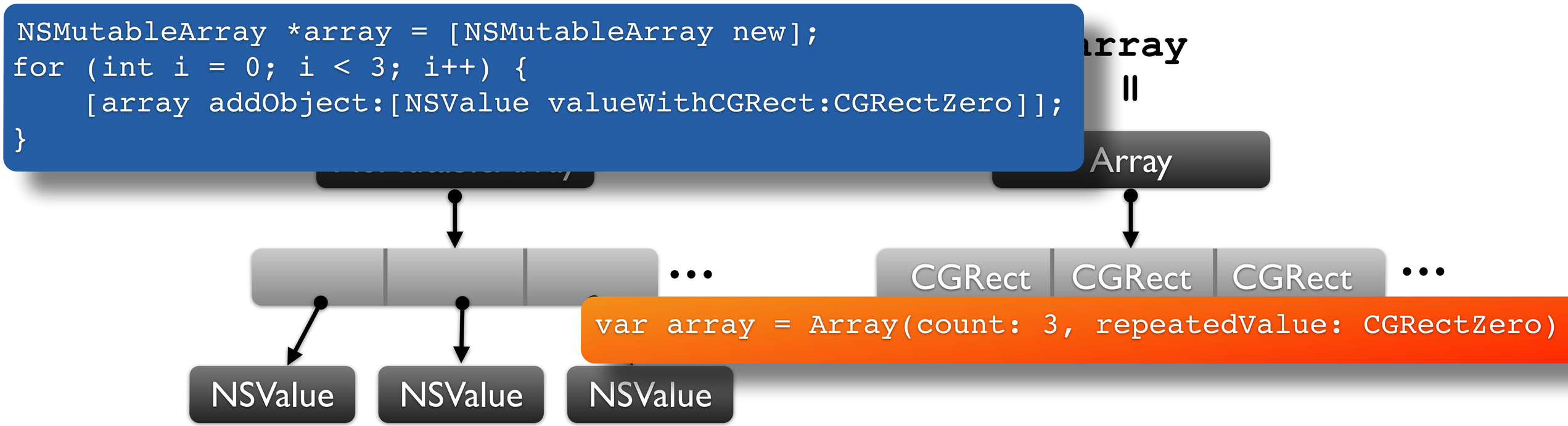
# Strukturen

- Struktur: Durch ihren Wert definiert, keine Identität
- `struct` in C: Keine Methoden, keine Initializer
- `struct` in Swift: Methoden, Initializer, kann Protokolle implementieren

# Strukturen: Vorteile

- Speicherverwaltung einfacher

```
NSMutableArray *array = [NSMutableArray new];
for (int i = 0; i < 3; i++) {
    [array addObject:[NSValue valueWithCGRect:CGRectZero]];
}
```



# Strukturen: Mehr Vorteile

- Strukturen wirken in Objective-C wie Fremdkörper:
  ```
  view.frame = CGRectOffset(view.frame, 10, 0);
  ```
- In Swift viel besser integriert:
  ```
  view.frame.offset(dx: 10, dy: 0)
  ```

# Strukturen: Noch mehr Vorteile

- Durch höhere Ausdrucksstärke: Viel mehr structs!
  - `String`
  - `Array`
  - `Dictionary`
- `Int` (!)

# Strukturen: Nachteile!

- Mehr Überlegen beim Code Design nötig: Struktur oder Klasse?
- Objective-C-Ballast:
  `NSURL`, `NSIndexPath`, `NSValue`, `CLLocation` weiterhin Objekte

# Aufzählungen

- enum in Swift kann Methoden und Initialisierer haben
- enum kann Protokolle implementieren
- enum kann pro Fall Werte speichern

```
enum SignalStrength {
   case Unknown
   case RSSI(Int)
}
```

# Aufzählungen: Vorteile

- Funktionen können als Methoden definiert werden:
- Assoziierte Werte eröffnen neue Anwendungsfelder

```
if (UIInterfaceOrientationIsLandscape(interfaceOrientation)) {
    // …
 }
```

```
 if interfaceOrientation.isLandscape() {
   // …
 }
```

# Aufzählungen: Nachteile

- Mehr Überlegung beim Code-Design nötig: enum oder Optional?

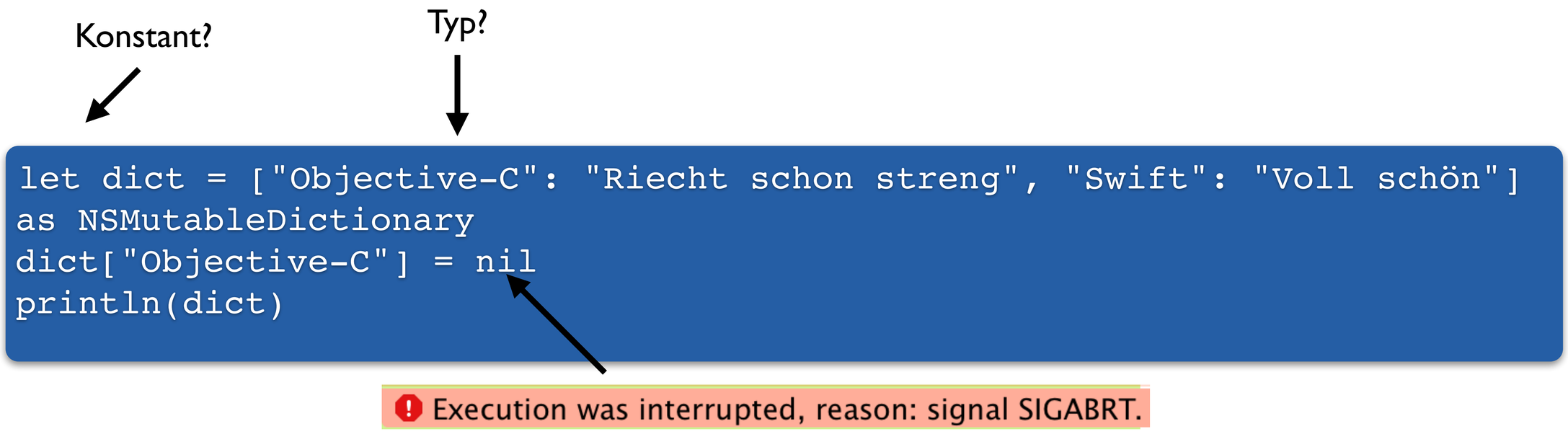
# Dictionary

- Schon erwähnt: struct (Speichermanagement einfacher)
- Einfacher zu verwenden: var ist änderbar, let nicht!
- Einfache Typen für Schlüssel und Werte möglich
- nil zuweisen zum Löschen von Werten aus Dictionary

```
var dict = ["Objective-C": "Riecht schon streng", "Swift": "Voll schön"]
dict["Objective-C"] = nil
println(dict) // "[Swift: Voll schön]"
```

# Dictionary: Umgewöhnung

- Passt nicht gut zu NSDictionary / NSMutableDictionary



# Eigene Operatoren

```
infix operator ∈ {}
func ∈(candidate: Int, array: [Int]) -> Bool {
    return contains(array, candidate)
}

let numbers = [4, 8, 15, 16, 23, 42]
if 42 ∈ numbers {
   println("Die Antwort")
}
```

# Eigene Operatoren: Umgewöhnung

- With Great Power Comes Great Responsibility
- Risiko: Code durch DSL mit Operatoren nicht mehr lesbar
- Beispiel: Wollen wir so etwas wirklich?

```
let constraint = view1.al_left == view2.al_right * 2.0 + 10.0
```

# Funktionstypen

- Beispiel Objective-C: Was ist foo?

```
int (^foo)(int);
```

- Beispiel Swift: Was ist foo?

```
var foo: (Int -> Int)
```

# Funktionstypen

- Beispiel Objective-C: Was ist foo?

```
int (^(^foo)(int (^)(int)))(int);
```

- Beispiel Swift: Was ist foo?

```
var foo: (Int -> Int) -> (Int -> Int)
```

# Generics

- Probleme des ∈-Operators:
  - Nur für Int Elemente (String? Double?)
  - Nur für Array (Range?)

```
infix operator ∈ {}
func ∈(candidate: Int, array: [Int]) -> Bool {
    return contains(array, candidate)
}
```

# Generics

- Alle Elementtypen unterstützen:

```
func ∈<T: Equatable>(candidate: T, array: [T]) -> Bool {
    return contains(array, candidate)
}

let languages = ["Go", "Swift", "Objective-C"]
assert("Swift" ∈ languages)
```

# Generics

- Alle Sequenztypen unterstützen:

```
func ∈<S: SequenceType where S.Generator.Element:
Equatable>(candidate: S.Generator.Element, sequence: S) -> Bool {
    return contains(sequence, candidate)
}
assert(42 ∈ 1...100)
```

# Generics: Umgewöhnung

- Objective-C hat keine Generics
- Erhöhter Schwierigkeitsgrad: Keine Generics von Protokollen

# Dynamik

# Message Passing: Ein Rückblick

- Objective-C ist eine hochdynamische Sprache
- Es werden erst Nachrichten geschickt, dann Methoden aufgerufen

# objc_msgSend

- Jeder Methodenaufruf in Objective-C läuft über `objc_msgSend`

```
id objc_msgSend(id self, SEL op, ...)
```

```
[self foo];
```

```
Lloh0:
   adrp  x8, L_OBJC_SELECTOR_REFERENCES_@PAGE
Lloh1:
   add x8, x8, L_OBJC_SELECTOR_REFERENCES_@PAGEOFF
Lloh2:
   ldr x1, [x8]
Ltmp4:
   bl  _objc_msgSend
```

# objc_msgSend Roadmap

- Check for ignored selectors (GC) and short-circuit.
- Check for nil target.
- If nil & nil receiver handler configured, jump to handler
- If nil & no handler (default), cleanup and return.
- Find the IMP on the class of the target and jump to it
- Search the class's method cache for the method IMP
- If found, jump to it.
- Not found: lookup the method IMP in the class itself
- If found, jump to it.
- If not found, jump to forwarding mechanism.

Bill Bumgarner http://www.friday.com/bbum/2009/12/18/objc_msgsend-part-1-the-road-map/

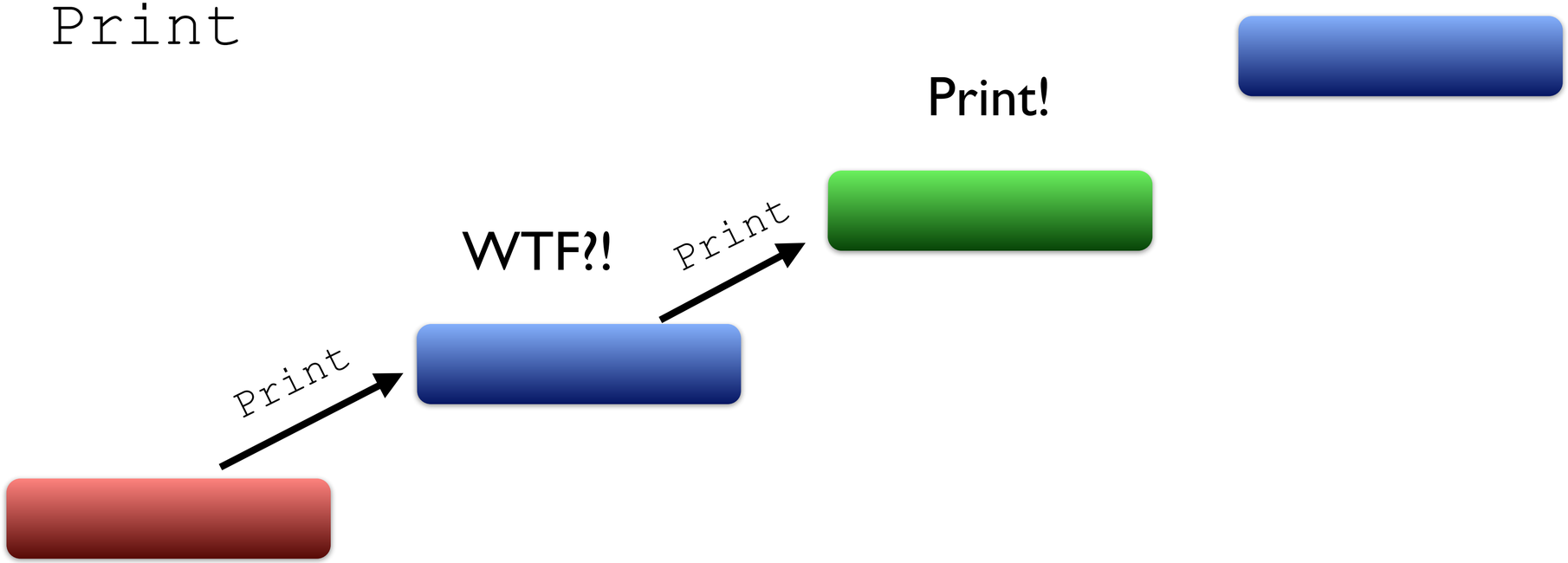
# Swift: Methodenaufruf

- VTable
- Direkter Aufruf
- Sogar Inlining

```
                    bl      $address
```

```
faz.bar()           ldr     r0, [sp, #4] ; Empty Tuple Argument
                    bl      0xa39dc      ; Branch
```

# Responder Chain: Smalltalk Style



# Swift: Dynamik

- Dynamik ist nicht weg
- dynamic für Key-Value-Observing
- @objc
- Objective-C Unterklasse

# Ausklang

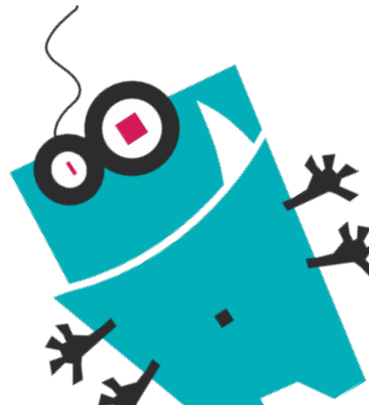
# Was mitnehmen?

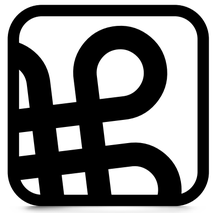
- Wir werden nicht um Swift drumherumkommen
- Sollten wir aber auch nicht wollen
  - Swift hat viel zu bieten, ausprobieren
  - Swift macht es vielen nicht ganz recht: Vielleicht ein gutes Zeichen?
  - Swift wird noch weiterentwickelt

# Was mitnehmen?

# Fragen?

# Vielen Dank





# Macoun