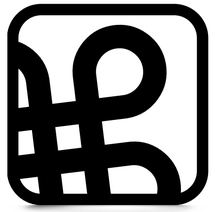
Vorbereitung:

1. Demos, Files im Finder anordnen.

2. Alle Programme die man für Demos benötigt vorher (!!) starten.
Bps: Xcode, Terminal, QuickTime.

3. Präsentation starten, und auf
Signal für Beginn warten.

• Für Wechsel zu Xcode „H“ oder cmd-Tab



Macoun

Einblick in den Swift Compiler

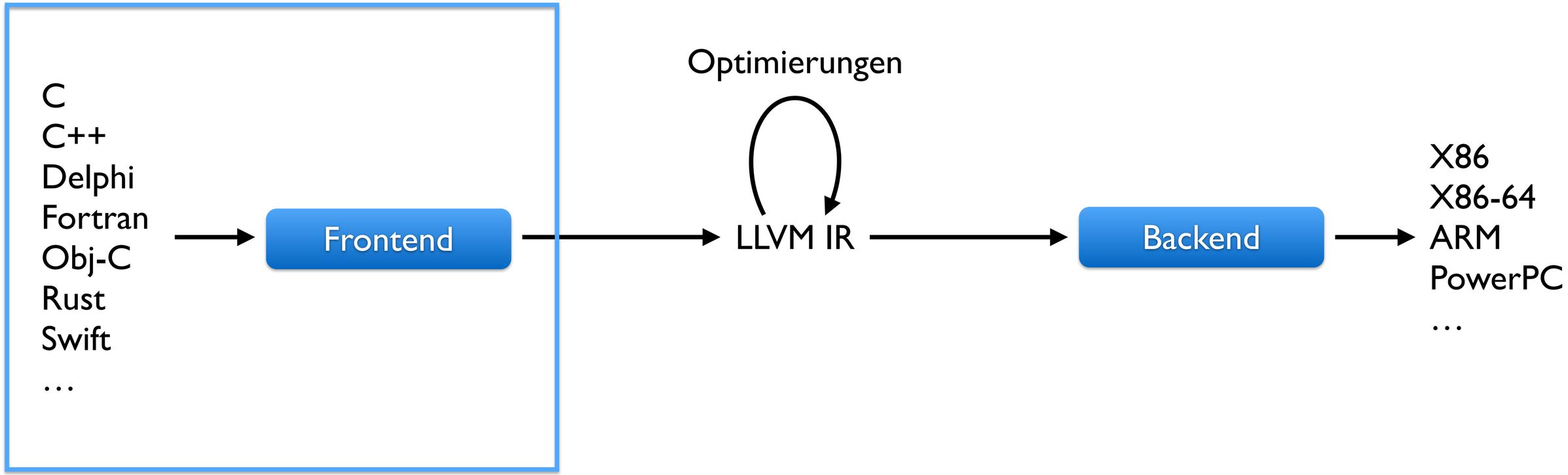
Alex Hoppen
@alex_hoppen

RWTH Aachen / University of Edinburgh

Ablauf

- LLVM
- Swift Compiler
- Bugfix Demo
- Beteiligung im OpenSource-Projekt

LLVM – Ein modularer Compiler



Nur dieser Teil musste neu geschrieben werden

Weitere Infos: http://www.aosabook.org/en/llvm.html

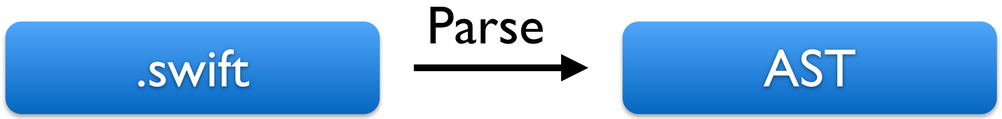
Das Swift Frontend

Linearer
Quellcode

cat demo.swift

```
struct Foo {
   var x: Int
   var y: Int

   func bar() -> Int {
      return x + y
   }
}
```

Das Swift Frontend



Linearer Quellcode

Baumstruktur

swiftc -dump-parse demo.swift

```
func_decl "bar()" type='<null type>'
  parameter_list
    parameter "self" type='Foo'
  parameter_list
  result
    type_ident
      component id='Int' bind=none
  brace_stmt
    return_stmt
      sequence_expr type='<null>'
        unresolved_decl_ref_expr type='<null>' name=x specialized=no
        unresolved_decl_ref_expr type='<null>' name=+ specialized=no
        unresolved_decl_ref_expr type='<null>' name=y specialized=no
```

swiftc -dump-parse demo.swift

```
func_decl "bar()" type='<null type>'
  parameter_list
    parameter "self" type='Foo'
  parameter_list
  result
    type_ident
      component id='Int' bind=none
  brace_stmt
    return_stmt
      sequence_expr type='<null>'
        unresolved_decl_ref_expr type='<null>' name=x specialized=no
        unresolved_decl_ref_expr type='<null>' name=+ specialized=no
        unresolved_decl_ref_expr type='<null>' name=y specialized=no
```

swiftc -dump-parse demo.swift

```
func_decl "bar()" type='<null type>'
  parameter_list
    parameter "self" type='Foo'
  parameter_list
  result
    type_ident
      component id='Int' bind=none
  brace_stmt
    return_stmt
      sequence_expr type='<null>'
        unresolved_decl_ref_expr type='<null>' name=x specialized=no
        unresolved_decl_ref_expr type='<null>' name=+ specialized=no
        unresolved_decl_ref_expr type='<null>' name=y specialized=no
```

swiftc -dump-parse demo.swift

```
func_decl "bar()" type='<null type>'
  parameter_list
    parameter "self" type='Foo'
  parameter_list
  result
    type_ident
      component id='Int' bind=none
  brace_stmt
    return_stmt
      sequence_expr type='<null>'
        unresolved_decl_ref_expr type='<null>' name=x specialized=no
        unresolved_decl_ref_expr type='<null>' name=+ specialized=no
        unresolved_decl_ref_expr type='<null>' name=y specialized=no
```

swiftc -dump-parse demo.swift

```
func_decl "bar()" type='<null type>'
  parameter_list
    parameter "self" type='Foo'
  parameter_list
  result
    type_ident
      component id='Int' bind=none
  brace_stmt
    return_stmt
      sequence_expr type='<null>'
        unresolved_decl_ref_expr type='<null>' name=x specialized=no
        unresolved_decl_ref_expr type='<null>' name=+ specialized=no
        unresolved_decl_ref_expr type='<null>' name=y specialized=no
```

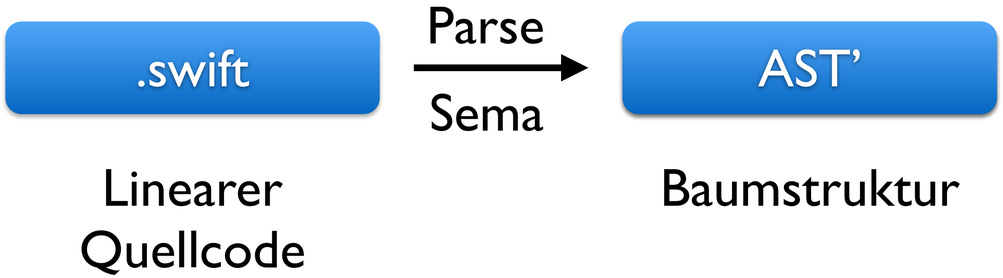
Das Swift Frontend

Linearer Quellcode

Baumstruktur

Baumstruktur

Das Swift Frontend



swiftc -dump-ast demo.swift

```
func_decl "bar()" type='(Foo) -> () -> Int' access=internal
  brace_stmt
    return_stmt
      binary_expr type='Int' location=demo.swift:6:12 nothrow
        declref_expr type='(Int, Int) -> Int' location=demo.swift:6:12
                     decl=Swift.(file).+ specialized=no)
        tuple_expr implicit type='(Int, Int)' location=demo.swift:6:10
          member_ref_expr type='Int' location=demo.swift:6:10
                          decl=demo.(file).Foo.x@demo.swift:2:6
            declref_expr implicit type='Foo' location=demo.swift:6:10
                         decl=demo.(file).Foo.func decl.self@demo.swift:5:7
          member_ref_expr type='Int' location=demo.swift:6:14
                          decl=demo.(file).Foo.y@demo.swift:3:6
            declref_expr implicit type='Foo' location=demo.swift:6:14
                         decl=demo.(file).Foo.func decl.self@demo.swift:5:7
```

swiftc -dump-ast demo.swift

```
func_decl "bar()" type='(Foo) -> () -> Int' access=internal
  brace_stmt
    return_stmt
      binary_expr type='Int' location=demo.swift:6:12 nothrow
        declref_expr type='(Int, Int) -> Int' location=demo.swift:6:12
                     decl=Swift.(file).+ specialized=no)
        tuple_expr implicit type='(Int, Int)' location=demo.swift:6:10
          member_ref_expr type='Int' location=demo.swift:6:10
                          decl=demo.(file).Foo.x@demo.swift:2:6
            declref_expr implicit type='Foo' location=demo.swift:6:10
                         decl=demo.(file).Foo.func decl.self@demo.swift:5:7
          member_ref_expr type='Int' location=demo.swift:6:14
                          decl=demo.(file).Foo.y@demo.swift:3:6
            declref_expr implicit type='Foo' location=demo.swift:6:14
                         decl=demo.(file).Foo.func decl.self@demo.swift:5:7
```

swiftc -dump-ast demo.swift

```
func_decl "bar()" type='(Foo) -> () -> Int' access=internal
  brace_stmt
    return_stmt
      binary_expr type='Int' location=demo.swift:6:12 nothrow
        declref_expr type='(Int, Int) -> Int' location=demo.swift:6:12
                     decl=Swift.(file).+ specialized=no)
        tuple_expr implicit type='(Int, Int)' location=demo.swift:6:10
          member_ref_expr type='Int' location=demo.swift:6:10
                          decl=demo.(file).Foo.x@demo.swift:2:6
            declref_expr implicit type='Foo' location=demo.swift:6:10
                         decl=demo.(file).Foo.func decl.self@demo.swift:5:7
          member_ref_expr type='Int' location=demo.swift:6:14
                          decl=demo.(file).Foo.y@demo.swift:3:6
            declref_expr implicit type='Foo' location=demo.swift:6:14
                         decl=demo.(file).Foo.func decl.self@demo.swift:5:7
```

swiftc -dump-ast demo.swift

```
func_decl "bar()" type='(Foo) -> () -> Int' access=internal
  brace_stmt
    return_stmt
      binary_expr type='Int' location=demo.swift:6:12 nothrow
        declref_expr type='(Int, Int) -> Int' location=demo.swift:6:12
                     decl=Swift.(file).+ specialized=no)
        tuple_expr implicit type='(Int, Int)' location=demo.swift:6:10
          member_ref_expr type='Int' location=demo.swift:6:10
                          decl=demo.(file).Foo.x@demo.swift:2:6
            declref_expr implicit type='Foo' location=demo.swift:6:10
                         decl=demo.(file).Foo.func decl.self@demo.swift:5:7
          member_ref_expr type='Int' location=demo.swift:6:14
                          decl=demo.(file).Foo.y@demo.swift:3:6
            declref_expr implicit type='Foo' location=demo.swift:6:14
                         decl=demo.(file).Foo.func decl.self@demo.swift:5:7
```

Das Swift Frontend

Linearer Quellcode

Baumstruktur

Assembler-artige Sprache mit starkem Swift-Typsystem

swiftc -emit-silgen demo.swift | xcrun swift-demangle

```
sil hidden @demo.Foo.bar () -> Swift.Int : $@convention(method) (Foo) -> Int {
bb0(%0 : $Foo):
  debug_value %0 : $Foo, let, name "self", argno 1, loc "demo.swift":5:7,
      scope 2
  %2 = function_ref @Swift.+ infix (Swift.Int, Swift.Int) -> Swift.Int :
      $@convention(thin) (Int, Int) -> Int, loc "demo.swift":6:12, scope 3
  %3 = struct_extract %0 : $Foo, #Foo.x, loc "demo.swift":6:10, scope 3
  %4 = struct_extract %0 : $Foo, #Foo.y, loc "demo.swift":6:14, scope 3
  %5 = apply %2(%3, %4) : $@convention(thin) (Int, Int) -> Int, loc
      "demo.swift":6:12, scope 3
  return %5 : $Int, loc "demo.swift":6:3, scope 3
}
```

swiftc -emit-silgen demo.swift | xcrun swift-demangle

```
sil hidden @demo.Foo.bar () -> Swift.Int : $@convention(method) (Foo) -> Int {
bb0(%0 : $Foo):
  debug_value %0 : $Foo, let, name "self", argno 1, loc "demo.swift":5:7,
      scope 2
  %2 = function_ref @Swift.+ infix (Swift.Int, Swift.Int) -> Swift.Int :
      $@convention(thin) (Int, Int) -> Int, loc "demo.swift":6:12, scope 3
  %3 = struct_extract %0 : $Foo, #Foo.x, loc "demo.swift":6:10, scope 3
  %4 = struct_extract %0 : $Foo, #Foo.y, loc "demo.swift":6:14, scope 3
  %5 = apply %2(%3, %4) : $@convention(thin) (Int, Int) -> Int, loc
      "demo.swift":6:12, scope 3
  return %5 : $Int, loc "demo.swift":6:3, scope 3
}
```

swiftc -emit-silgen demo.swift | xcrun swift-demangle

```
sil hidden @demo.Foo.bar () -> Swift.Int : $@convention(method) (Foo) -> Int {
bb0(%0 : $Foo):
  debug_value %0 : $Foo, let, name "self", argno 1, loc "demo.swift":5:7,
      scope 2
  %2 = function_ref @Swift.+ infix (Swift.Int, Swift.Int) -> Swift.Int :
      $@convention(thin) (Int, Int) -> Int, loc "demo.swift":6:12, scope 3
  %3 = struct_extract %0 : $Foo, #Foo.x, loc "demo.swift":6:10, scope 3
  %4 = struct_extract %0 : $Foo, #Foo.y, loc "demo.swift":6:14, scope 3
  %5 = apply %2(%3, %4) : $@convention(thin) (Int, Int) -> Int, loc
      "demo.swift":6:12, scope 3
  return %5 : $Int, loc "demo.swift":6:3, scope 3
}
```

swiftc -emit-silgen demo.swift | xcrun swift-demangle

```
sil hidden @demo.Foo.bar () -> Swift.Int : $@convention(method) (Foo) -> Int {
bb0(%0 : $Foo):
  debug_value %0 : $Foo, let, name "self", argno 1, loc "demo.swift":5:7,
      scope 2
  %2 = function_ref @Swift.+ infix (Swift.Int, Swift.Int) -> Swift.Int :
      $@convention(thin) (Int, Int) -> Int, loc "demo.swift":6:12, scope 3
  %3 = struct_extract %0 : $Foo, #Foo.x, loc "demo.swift":6:10, scope 3
  %4 = struct_extract %0 : $Foo, #Foo.y, loc "demo.swift":6:14, scope 3
  %5 = apply %2(%3, %4) : $@convention(thin) (Int, Int) -> Int, loc
      "demo.swift":6:12, scope 3
  return %5 : $Int, loc "demo.swift":6:3, scope 3
}
```

swiftc -emit-silgen demo.swift | xcrun swift-demangle

```
sil hidden @demo.Foo.bar () -> Swift.Int : $@convention(method) (Foo) -> Int {
bb0(%0 : $Foo):
  debug_value %0 : $Foo, let, name "self", argno 1, loc "demo.swift":5:7,
      scope 2
  %2 = function_ref @Swift.+ infix (Swift.Int, Swift.Int) -> Swift.Int :
      $@convention(thin) (Int, Int) -> Int, loc "demo.swift":6:12, scope 3
  %3 = struct_extract %0 : $Foo, #Foo.x, loc "demo.swift":6:10, scope 3
  %4 = struct_extract %0 : $Foo, #Foo.y, loc "demo.swift":6:14, scope 3
  %5 = apply %2(%3, %4) : $@convention(thin) (Int, Int) -> Int, loc
      "demo.swift":6:12, scope 3
  return %5 : $Int, loc "demo.swift":6:3, scope 3
}
```

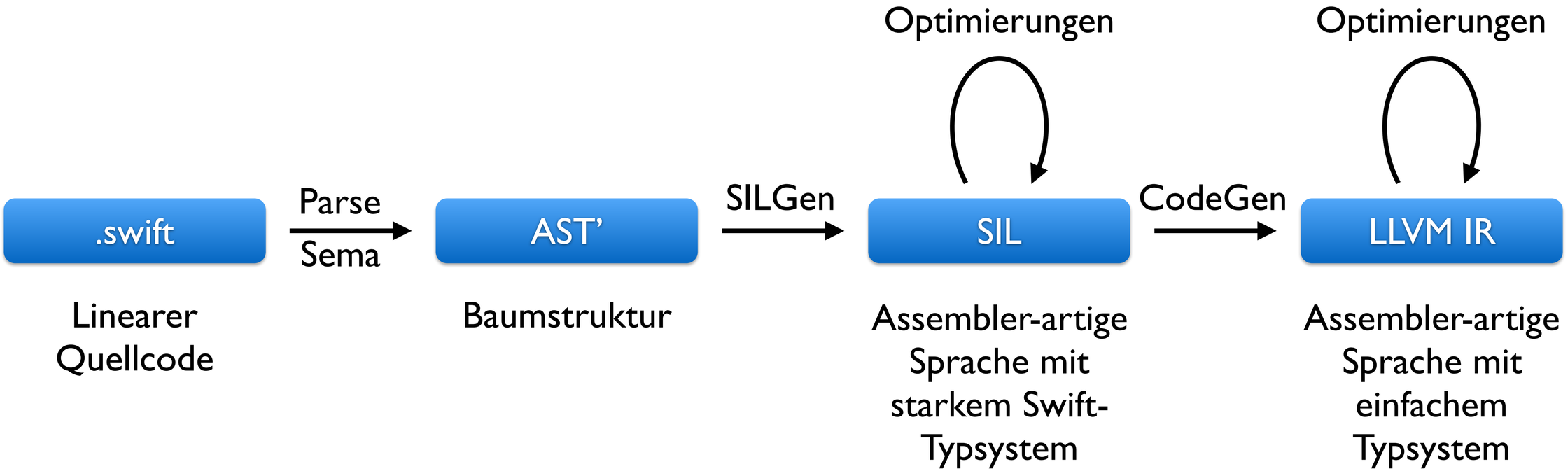
swiftc -emit-silgen demo.swift | xcrun swift-demangle

```
sil hidden @demo.Foo.bar () -> Swift.Int : $@convention(method) (Foo) -> Int {
bb0(%0 : $Foo):
  debug_value %0 : $Foo, let, name "self", argno 1, loc "demo.swift":5:7,
      scope 2
  %2 = function_ref @Swift.+ infix (Swift.Int, Swift.Int) -> Swift.Int :
      $@convention(thin) (Int, Int) -> Int, loc "demo.swift":6:12, scope 3
  %3 = struct_extract %0 : $Foo, #Foo.x, loc "demo.swift":6:10, scope 3
  %4 = struct_extract %0 : $Foo, #Foo.y, loc "demo.swift":6:14, scope 3
  %5 = apply %2(%3, %4) : $@convention(thin) (Int, Int) -> Int, loc
      "demo.swift":6:12, scope 3
  return %5 : $Int, loc "demo.swift":6:3, scope 3
}
```

Das Swift Frontend

Weitere Infos: https://github.com/apple/swift/blob/master/docs/SIL.rst#guaranteed-optimization-and-diagnostic-passes

Das Swift Frontend



swiftc -emit-ir demo.swift | xcrun swift-demangle

```
define hidden i64 @demo.Foo.bar () -> Swift.Int(i64, i64) #0 {
entry:
  %2 = call { i64, i1 } @llvm.sadd.with.overflow.i64(i64 %0, i64 %1)
  %3 = extractvalue { i64, i1 } %2, 0
  %4 = extractvalue { i64, i1 } %2, 1
  br i1 %4, label %6, label %5

; <label>:5
  ret i64 %3

; <label>:6
  call void asm sideeffect "", "n"(i32 0)
  call void @llvm.trap()
  unreachable
}
```

swiftc -emit-ir demo.swift | xcrun swift-demangle

```
define hidden i64 @demo.Foo.bar () -> Swift.Int(i64, i64) #0 {
entry:
  %2 = call { i64, i1 } @llvm.sadd.with.overflow.i64(i64 %0, i64 %1)
  %3 = extractvalue { i64, i1 } %2, 0
  %4 = extractvalue { i64, i1 } %2, 1
  br i1 %4, label %6, label %5

; <label>:5
  ret i64 %3

; <label>:6
  call void asm sideeffect "", "n"(i32 0)
  call void @llvm.trap()
  unreachable
}
```

swiftc -emit-ir demo.swift | xcrun swift-demangle

```
define hidden i64 @demo.Foo.bar () -> Swift.Int(i64, i64) #0 {
entry:
  %2 = call { i64, i1 } @llvm.sadd.with.overflow.i64(i64 %0, i64 %1)
  %3 = extractvalue { i64, i1 } %2, 0
  %4 = extractvalue { i64, i1 } %2, 1
  br i1 %4, label %6, label %5

; <label>:5
  ret i64 %3

; <label>:6
  call void asm sideeffect "", "n"(i32 0)
  call void @llvm.trap()
  unreachable
}
```

swiftc -emit-ir demo.swift | xcrun swift-demangle

```
define hidden i64 @demo.Foo.bar () -> Swift.Int(i64, i64) #0 {
entry:
  %2 = call { i64, i1 } @llvm.sadd.with.overflow.i64(i64 %0, i64 %1)
  %3 = extractvalue { i64, i1 } %2, 0
  %4 = extractvalue { i64, i1 } %2, 1
  br i1 %4, label %6, label %5

; <label>:5
  ret i64 %3

; <label>:6
  call void asm sideeffect "", "n"(i32 0)
  call void @llvm.trap()
  unreachable
}
```

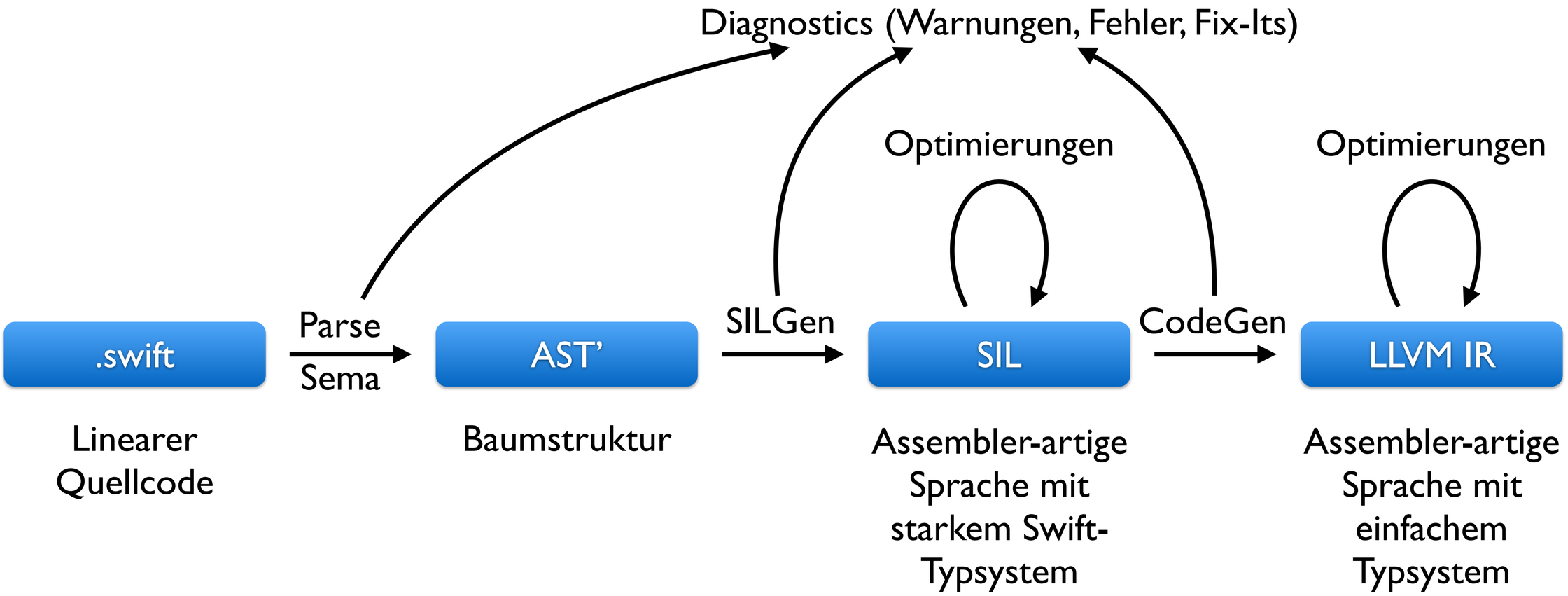
swiftc -emit-ir demo.swift | xcrun swift-demangle

```
define hidden i64 @demo.Foo.bar () -> Swift.Int(i64, i64) #0 {
entry:
  %2 = call { i64, i1 } @llvm.sadd.with.overflow.i64(i64 %0, i64 %1)
  %3 = extractvalue { i64, i1 } %2, 0
  %4 = extractvalue { i64, i1 } %2, 1
  br i1 %4, label %6, label %5

; <label>:5
  ret i64 %3

; <label>:6
  call void asm sideeffect "", "n"(i32 0)
  call void @llvm.trap()
  unreachable
}
```

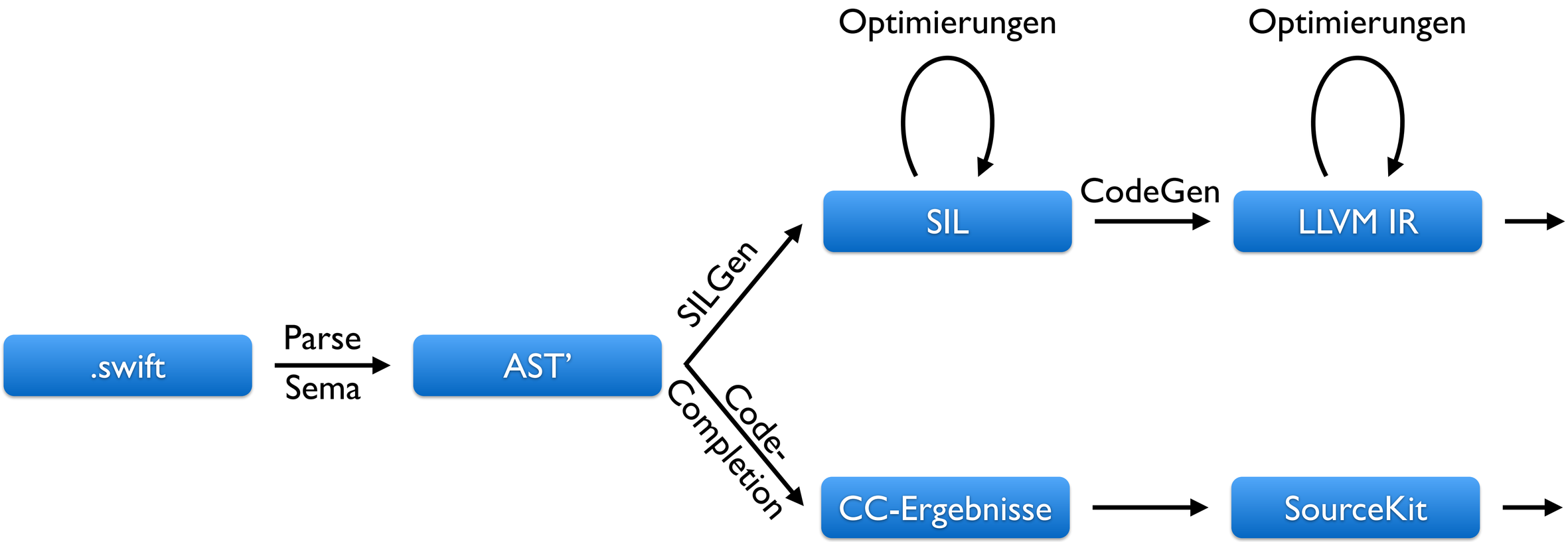
swiftc -emit-ir demo.swift | xcrun swift-demangle

```
define hidden i64 @demo.Foo.bar () -> Swift.Int(i64, i64) #0 {
entry:
  %2 = call { i64, i1 } @llvm.sadd.with.overflow.i64(i64 %0, i64 %1)
  %3 = extractvalue { i64, i1 } %2, 0
  %4 = extractvalue { i64, i1 } %2, 1
  br i1 %4, label %6, label %5

; <label>:5
  ret i64 %3

; <label>:6
  call void asm sideeffect "", "n"(i32 0)
  call void @llvm.trap()
  unreachable
}
```

Das Swift Frontend



Code Completion?



Demo

bugs.swift.org

- JIRA Bugtracker
- Für jeden offen
- Bugname: SR-XXXX
- Swift-Bugs dort eintragen, damit sie auch von Nicht-Apple-Mitarbeitern behoben werden können

Swift Evolution

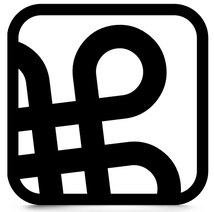
- Entscheidet die nächsten Features für Swift
- Vom Swift Core Team moderiert und gelenkt
  - Aktuell nur Änderungen, die das ABI betreffen
- Für jeden offen

Open Source Projekte

- Swift Compiler (C++)
- Swift Standard Library (Swift)
- Swift Corelibs Foundation (Swift)
    - Core Foundation, libdispatch, XCTest

Fragen?

Vielen Dank



Macoun