Kotlin Compiler

In past, 1.4 and beyond

Simon Ogorodnik
JetBrains

Topics

- Compiler as a opaque box

- Kotlin Compiler Frontend
  - 1.0
  - New Inference in 1.4
  - New Frontend IR

- Kotlin Compiler Backend
  - 1.0
  - New Backend IR

Why?

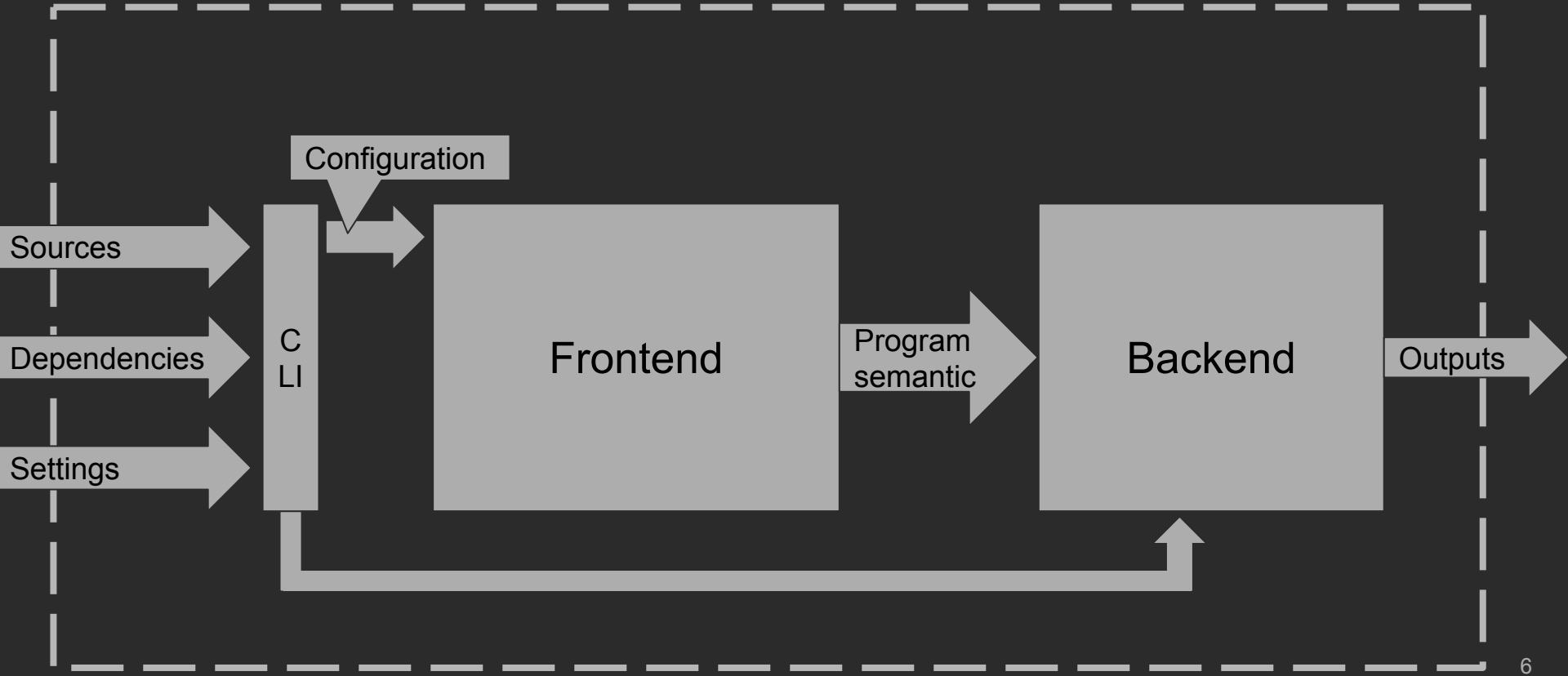
- Major compiler changes
- Complete rewrites of huge compiler parts

- Language evolved since 1.0
- Kotlin Compiler 1.0 wasn’t meant to run fast but to develop fast
- And it is 10 years old

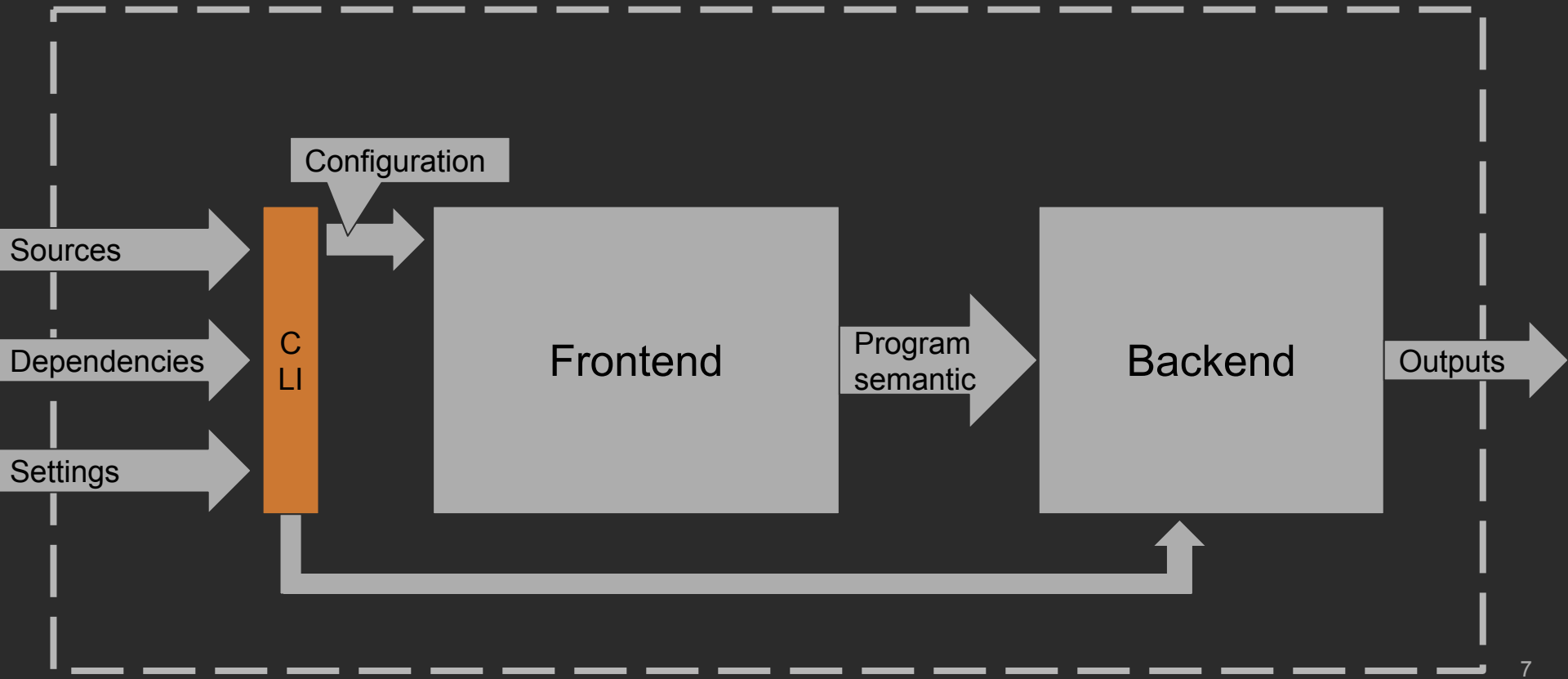
Compiler as a Opaque Box

Compiler as a Opaque Box

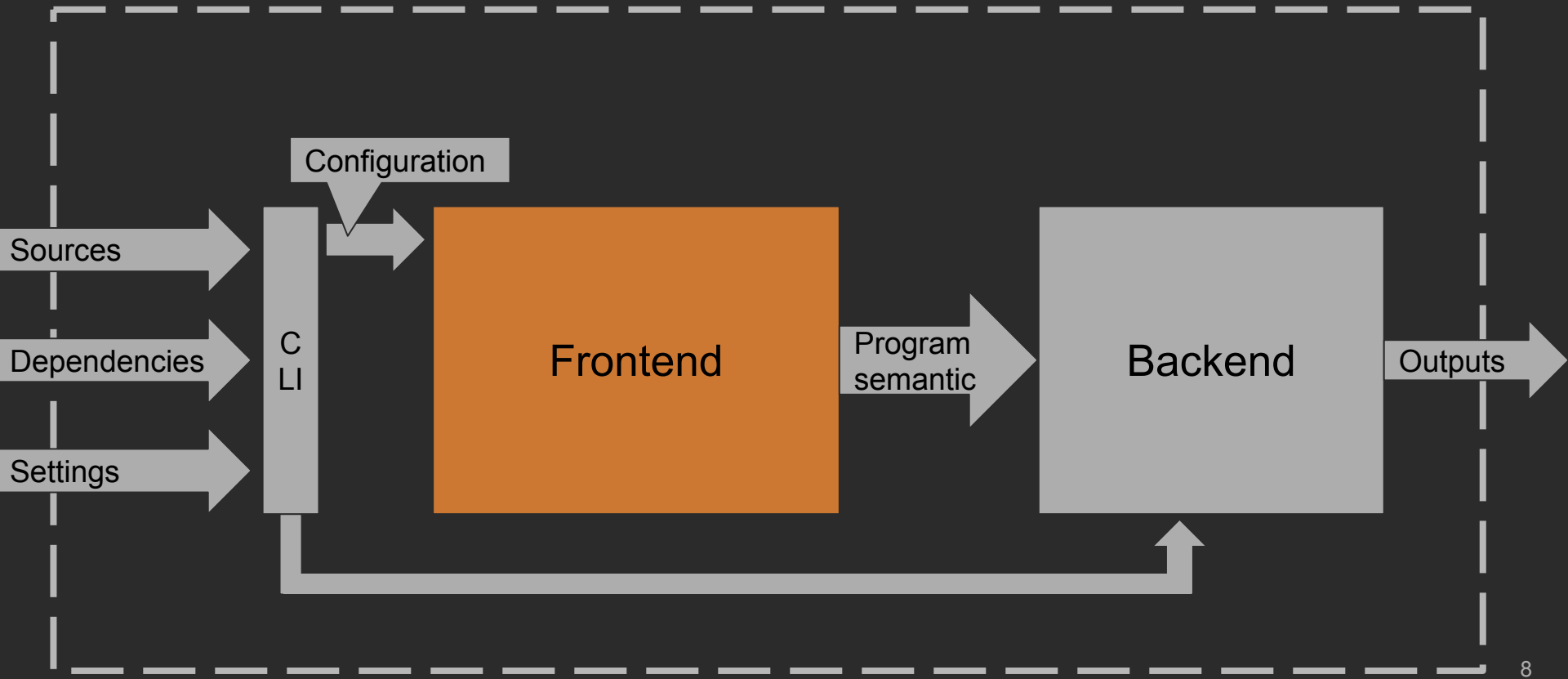
Compiler as Transparent Box



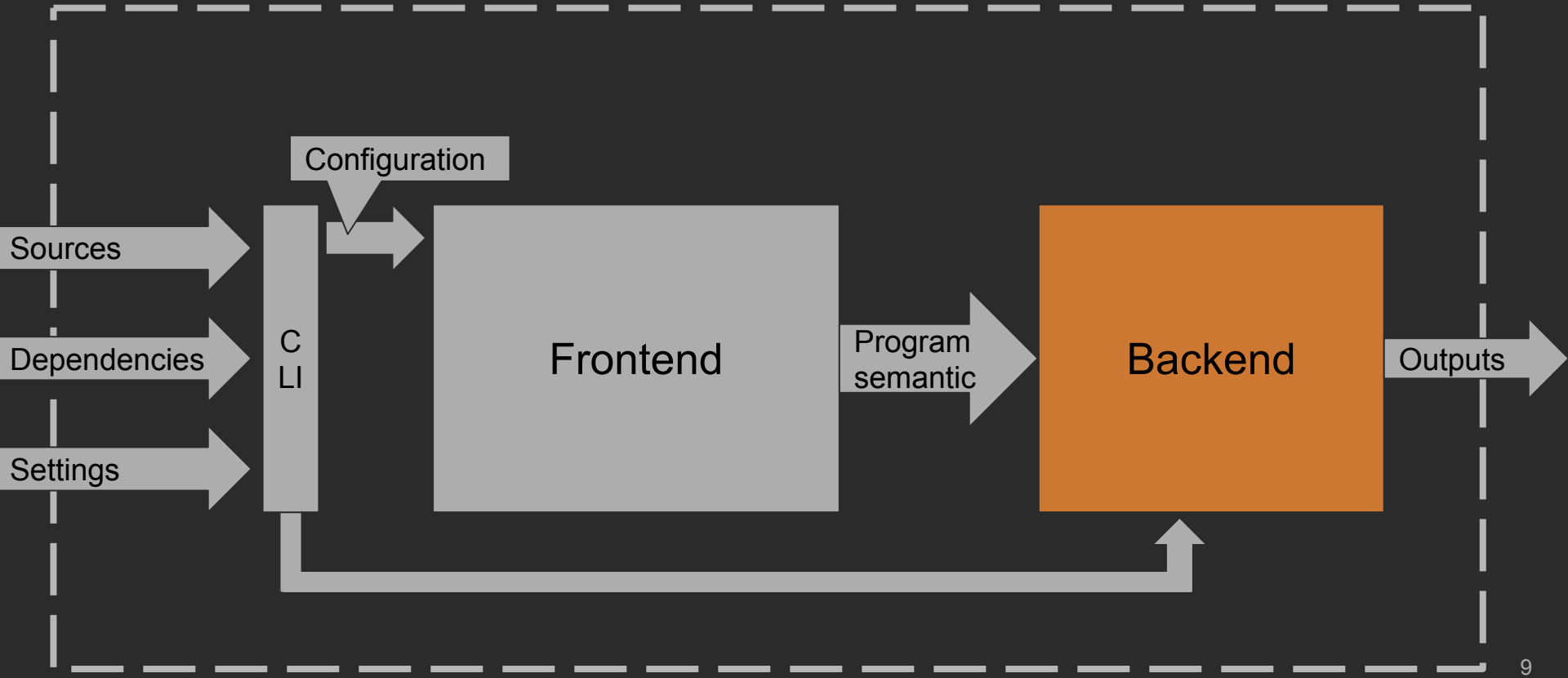
Compiler as Transparent Box



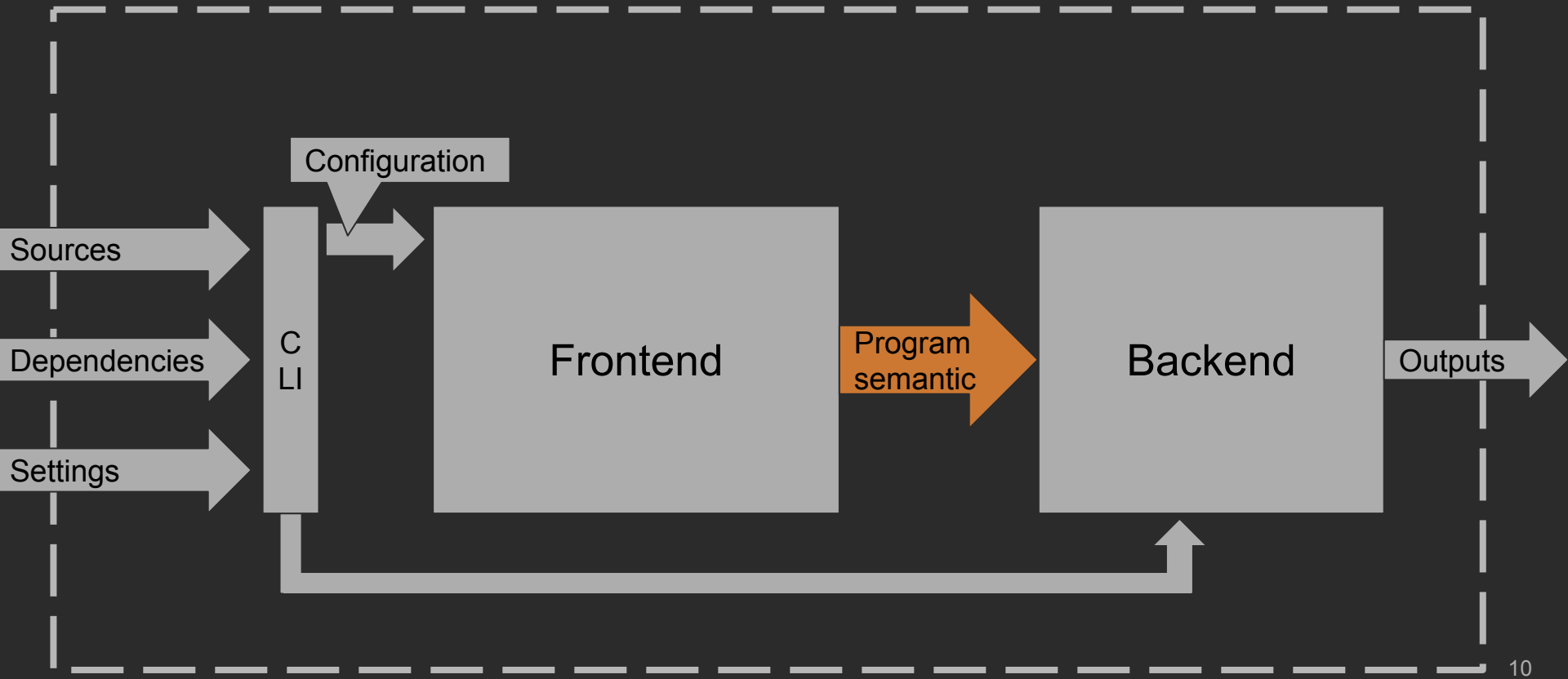
Compiler as Transparent Box



Compiler as Transparent Box



Compiler as Transparent Box



Semantics? Semantics!

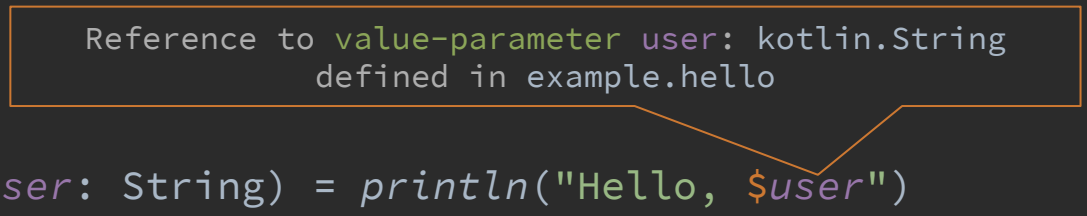
```
fun hello(user: String) = println("Hello, $user")
```

Semantics? Semantics!

```
public fun hello(user: kotlin.String): kotlin.String
        defined in example in file Example.kt
```

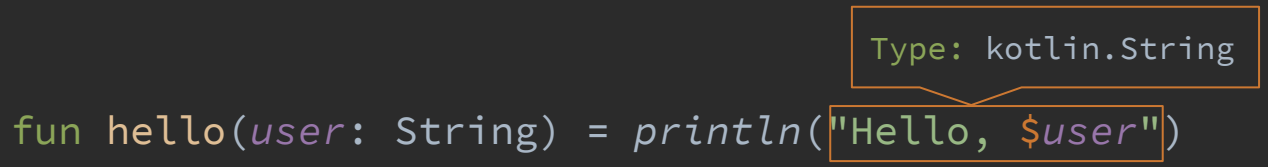
```
fun hello(user: String) = println("Hello, $user")
```

Semantics? Semantics!

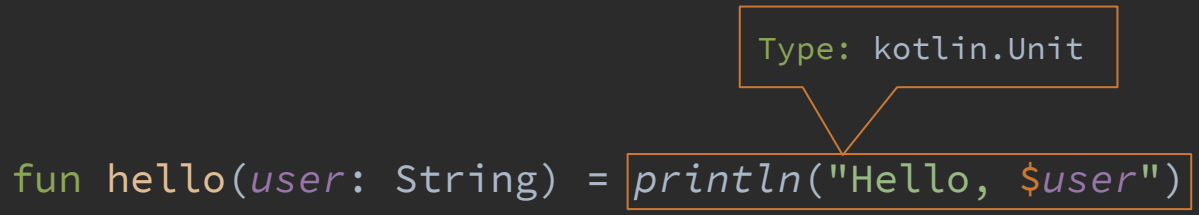


Semantics? Semantics!

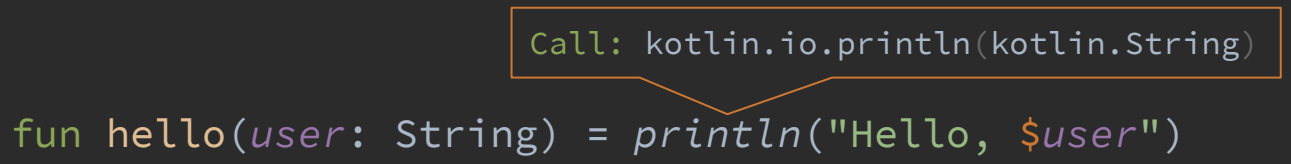
Semantics? Semantics!



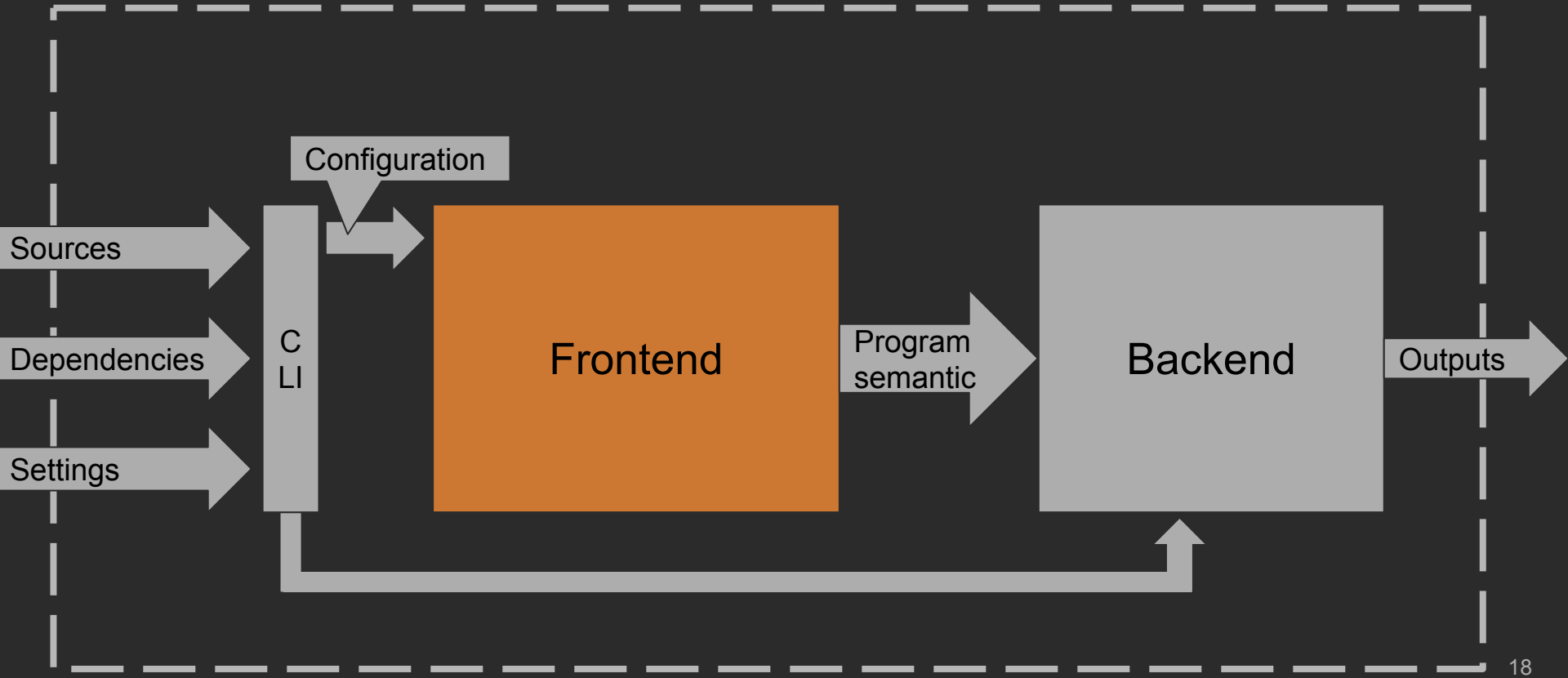
Semantics? Semantics!



Semantics? Semantics!

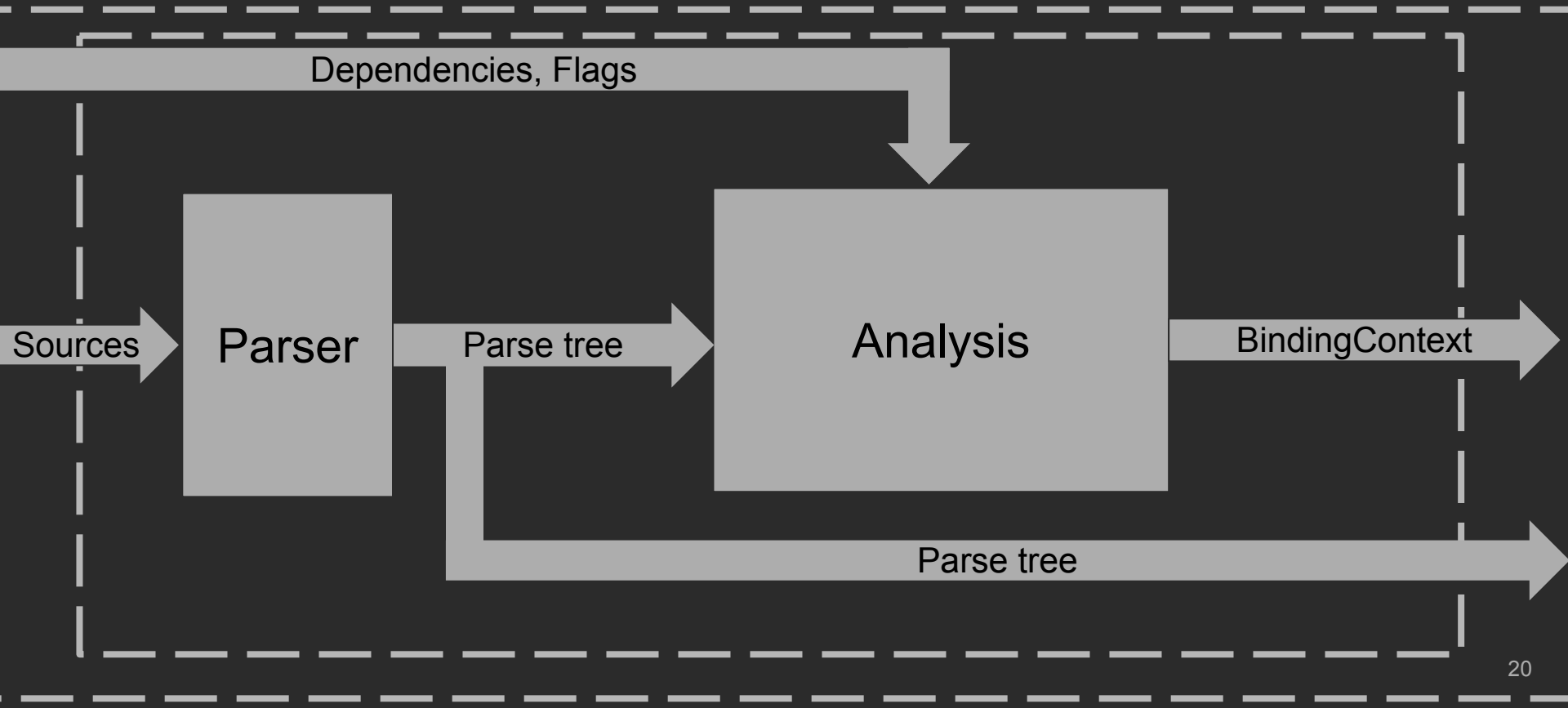


Compiler as Transparent Box

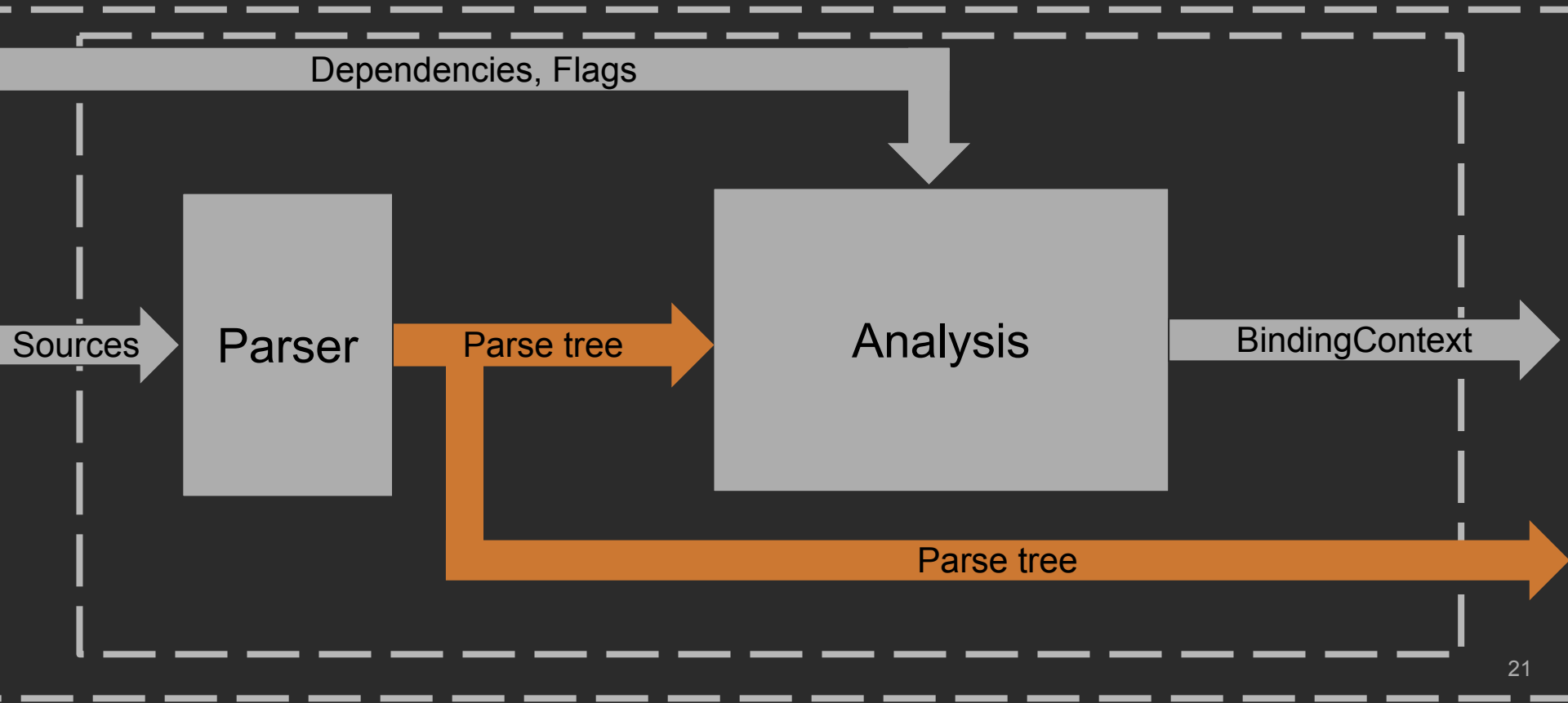


Frontend Timeline

Frontend 1.0 on High-Level



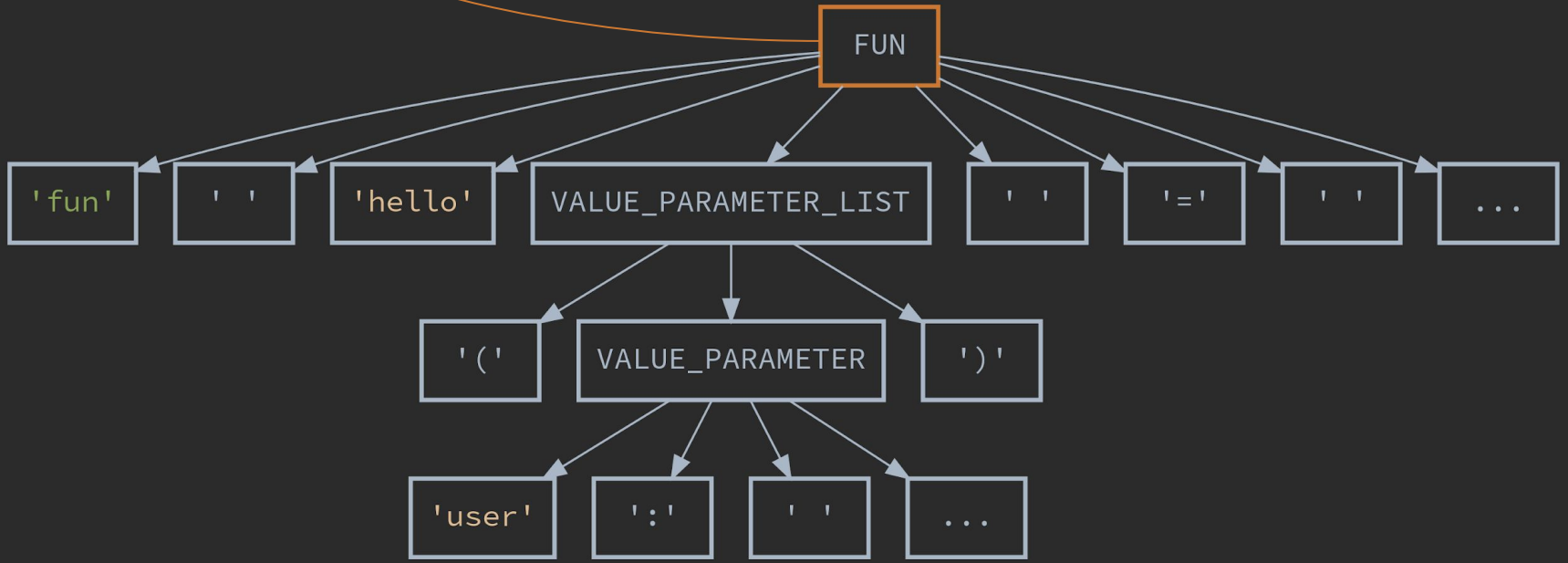
Frontend 1.0 on High-Level



Parse Tree, aka PSI - Program Structure Interface

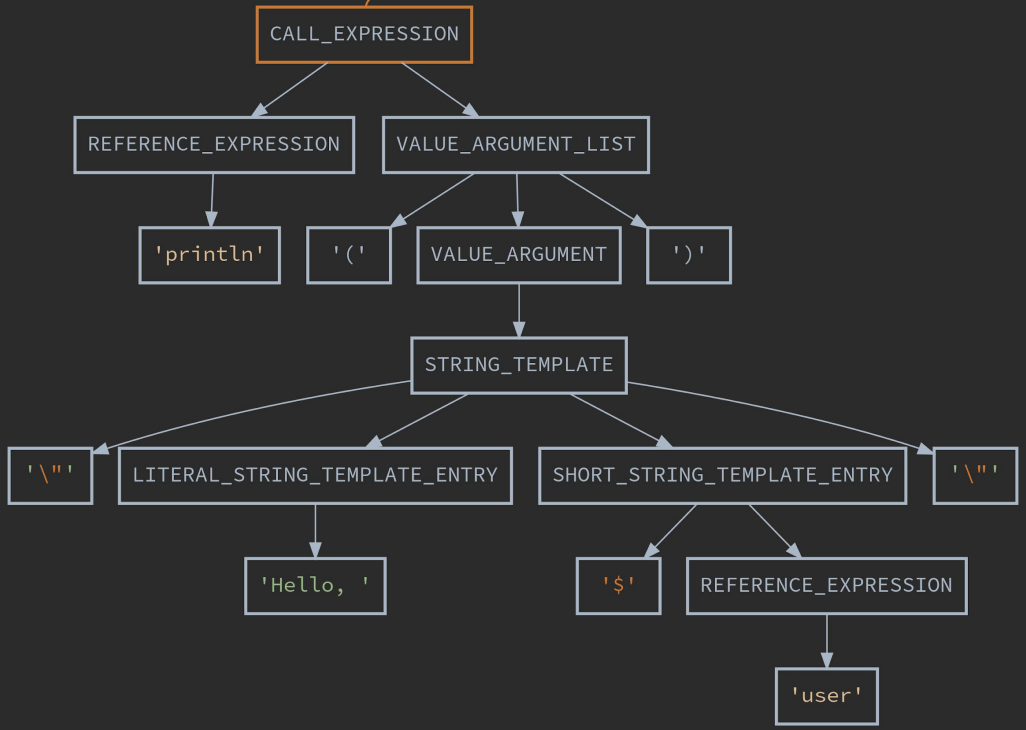
```
fun hello(user: String) = ...
```

NOTE: CST



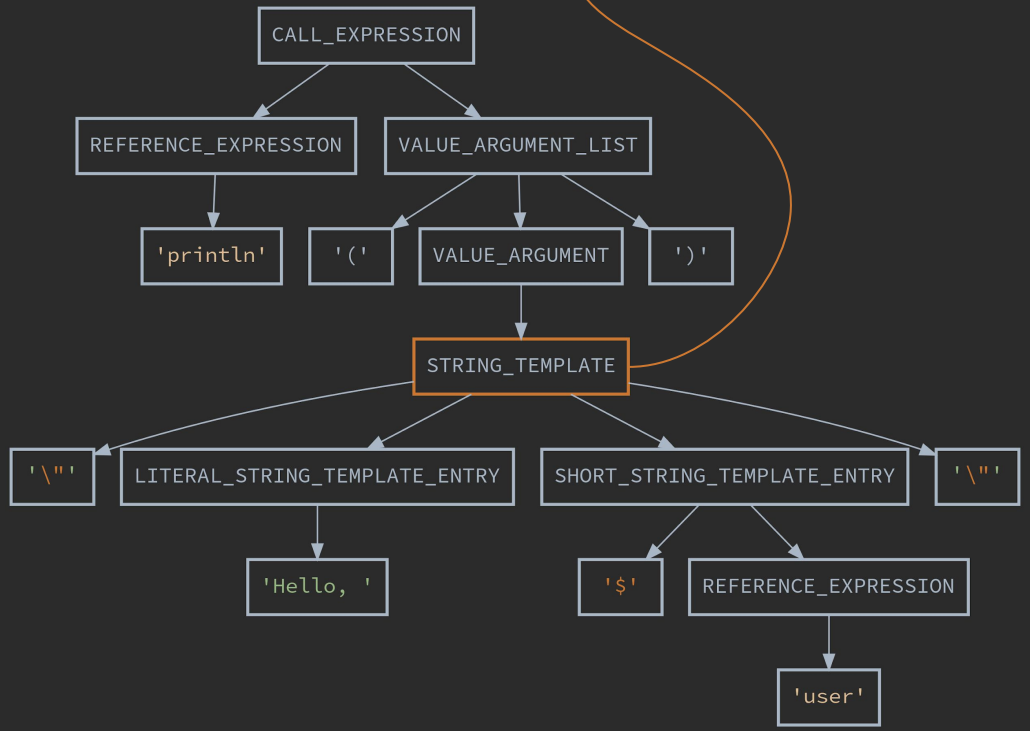
Parse Tree, aka PSI - Program Structure Interface

```
fun hello(user: String) = println("Hello, $user")
```



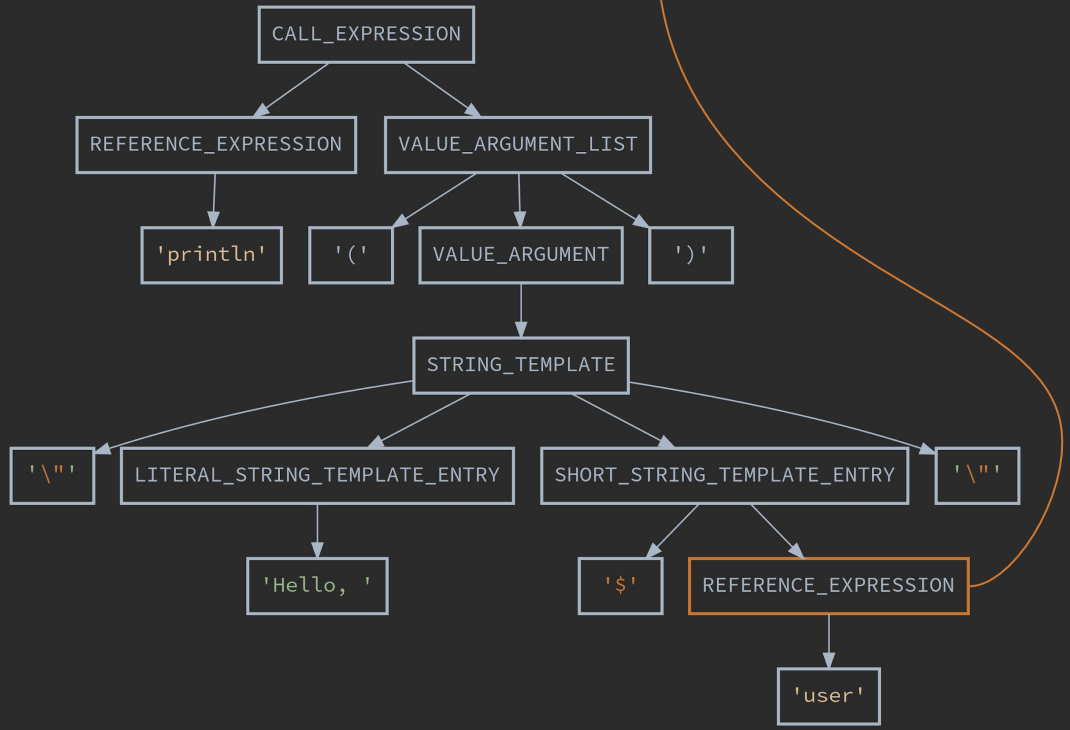
Parse Tree, aka PSI - Program Structure Interface

```
fun hello(user: String) = println("Hello, $user")
```

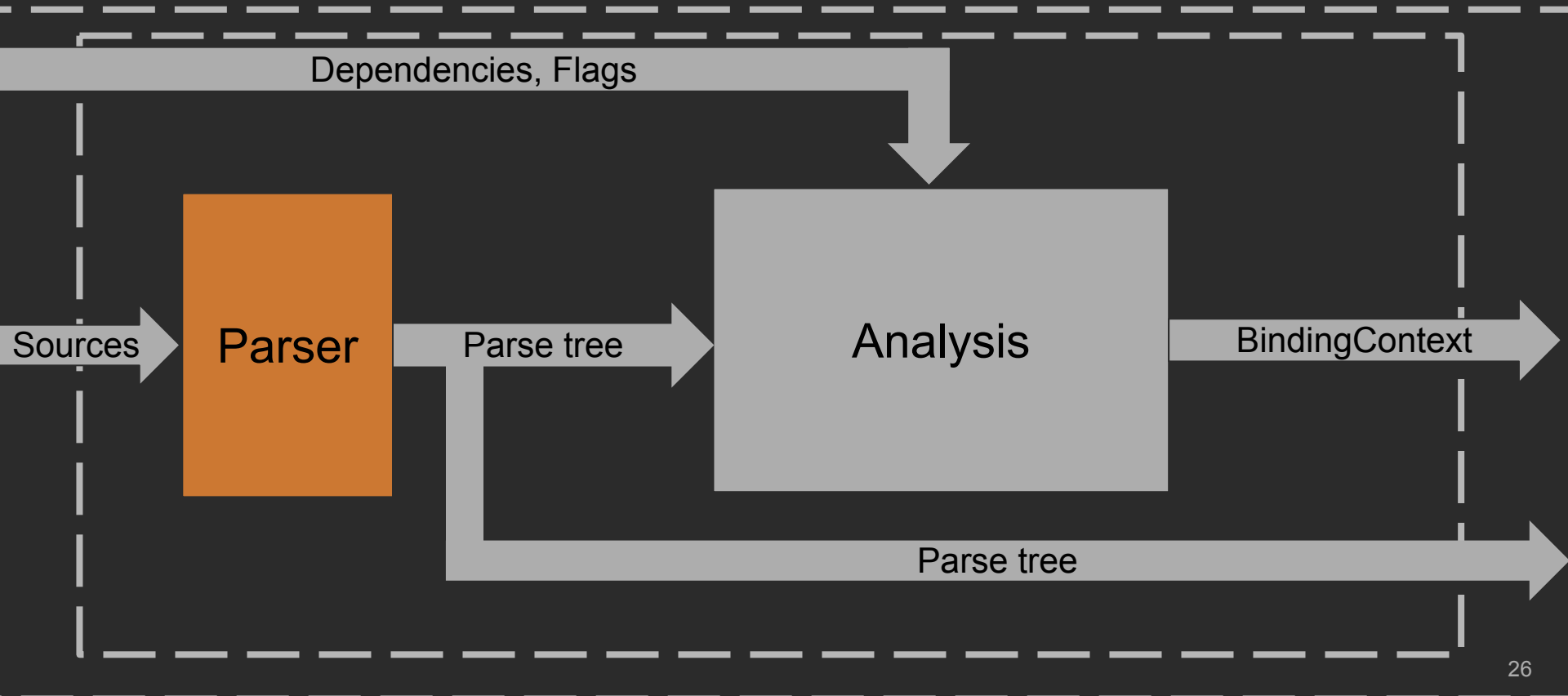


Parse Tree, aka PSI - Program Structure Interface

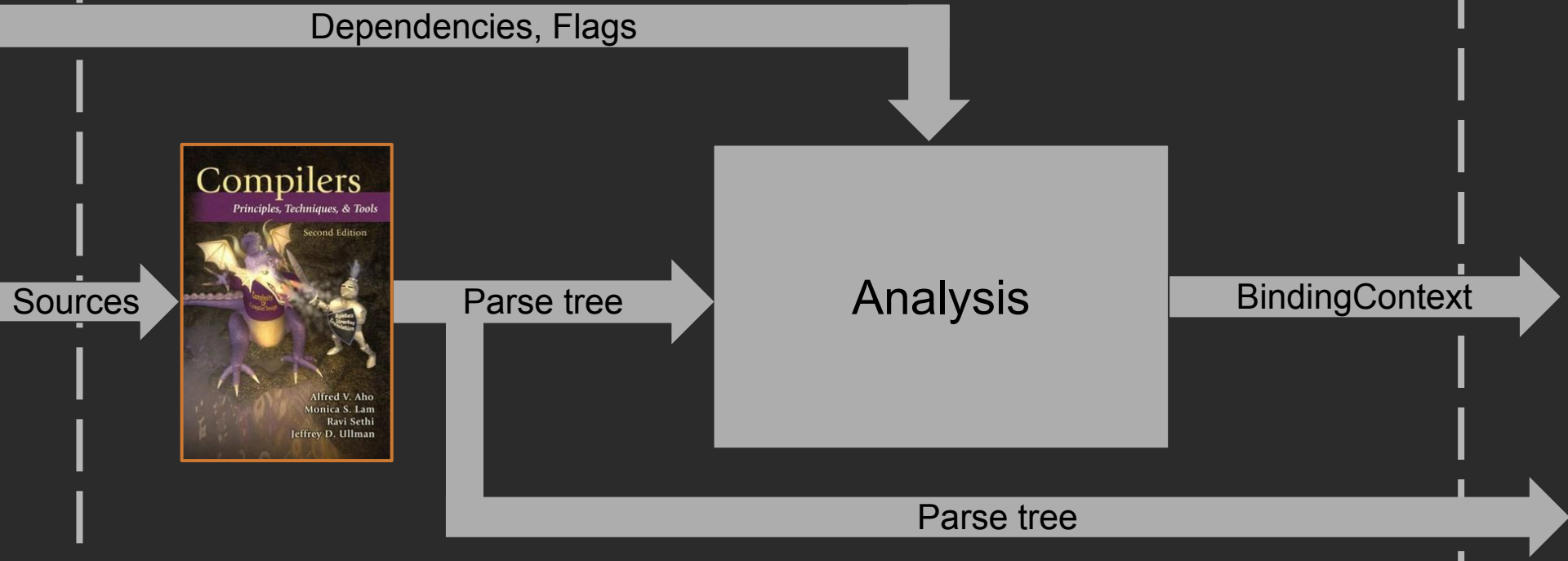
```
fun hello(user: String) = println("Hello, $user")
```



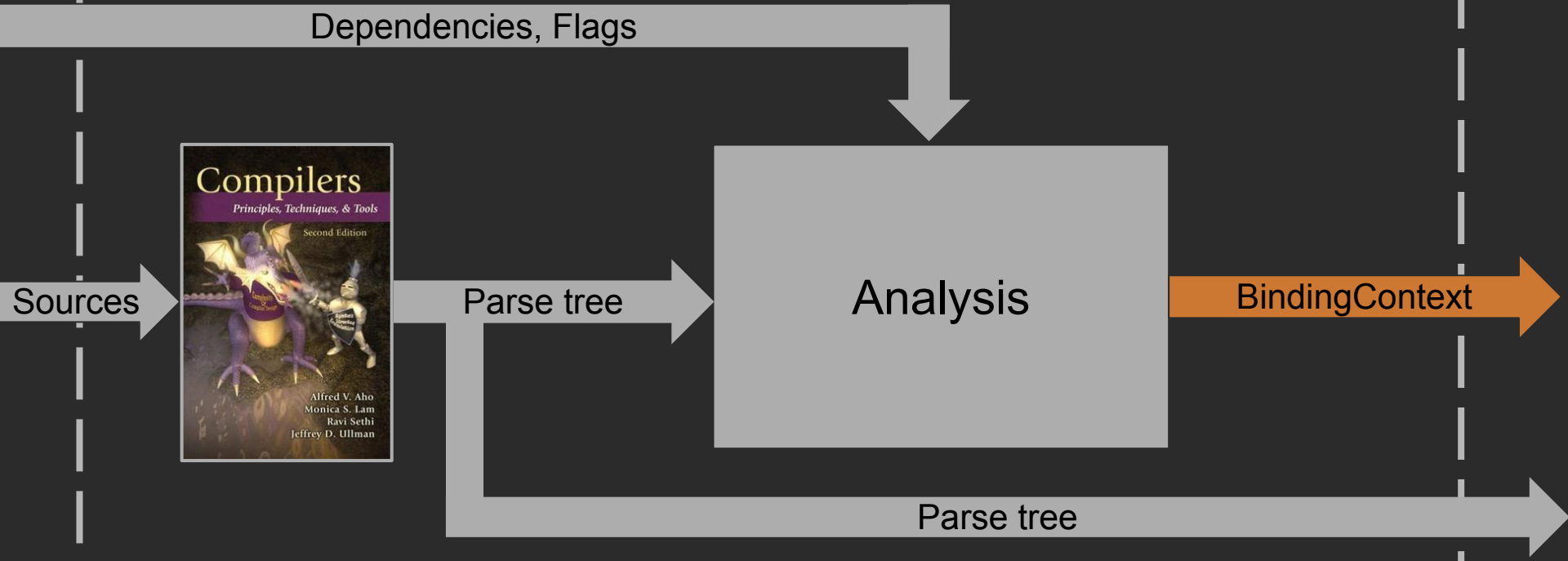
Frontend 1.0 on High-Level



Frontend 1.0 on High-Level



Frontend 1.0 on High-Level

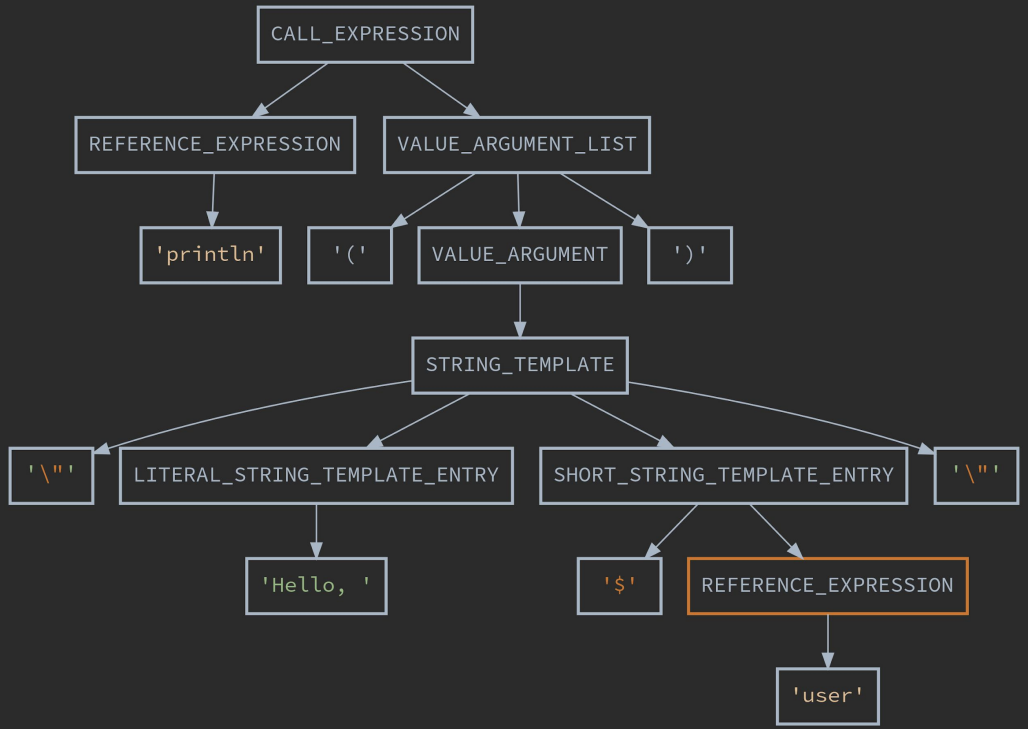


BindingContext?

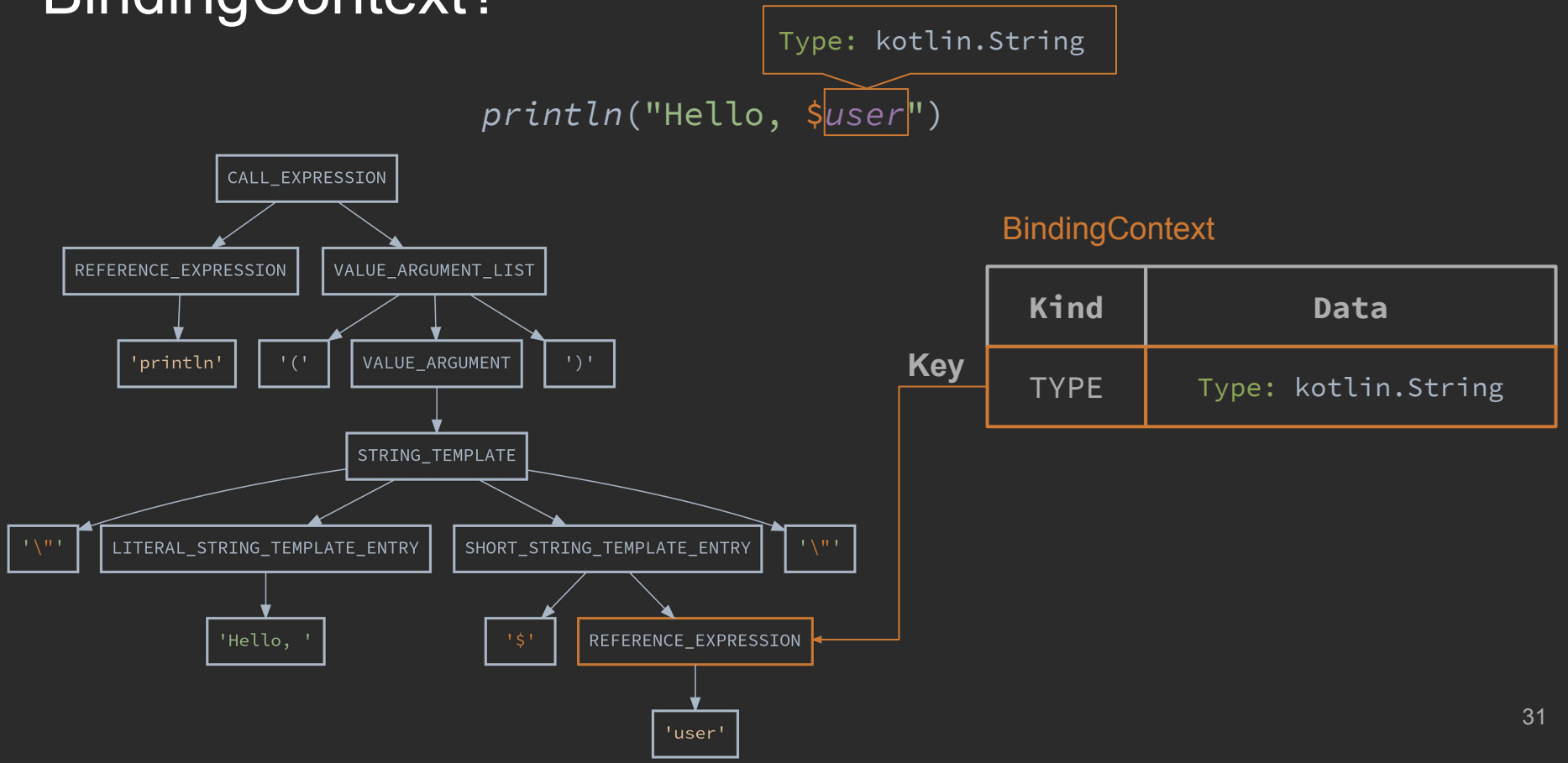
BindingContext?

Type: kotlin.String

```
println("Hello, $user")
```



BindingContext?



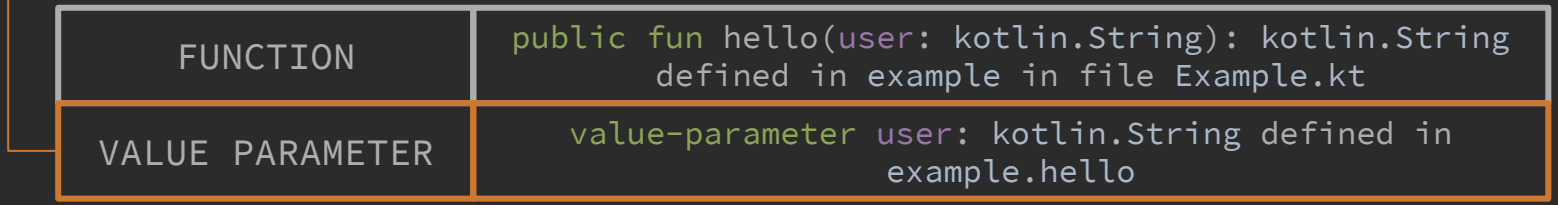
BindingContext?

```
fun hello(user: String) = "Hello, $user"
```

FUNCTION	public fun hello(user: kotlin.String): kotlin.String defined in example in file Example.kt

BindingContext?

```
fun hello(user: String) = "Hello, $user"
```



BindingContext?

```
fun hello(user: String) = "Hello, $user"
```

FUNCTION	public fun hello(user: kotlin.String): kotlin.String defined in example in file Example.kt
VALUE PARAMETER	value-parameter user: kotlin.String defined in example.hello
REFERENCE TARGET	deserialized class String

BindingContext?

```
fun hello(user: String) = "Hello, $user"
```

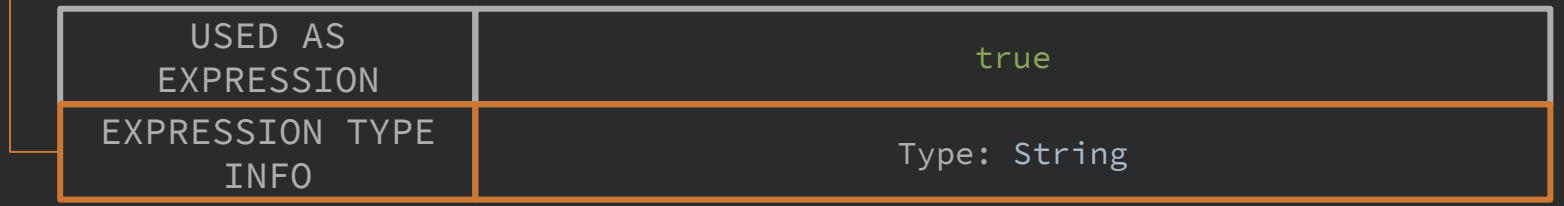
FUNCTION	public fun hello(user: kotlin.String): kotlin.String defined in example in file Example.kt
VALUE PARAMETER	value-parameter user: kotlin.String defined in example.hello
REFERENCE TARGET	deserialized class String
TYPE	Type: String

BindingContext?

```
fun hello(user: String) = "Hello, $user"
```

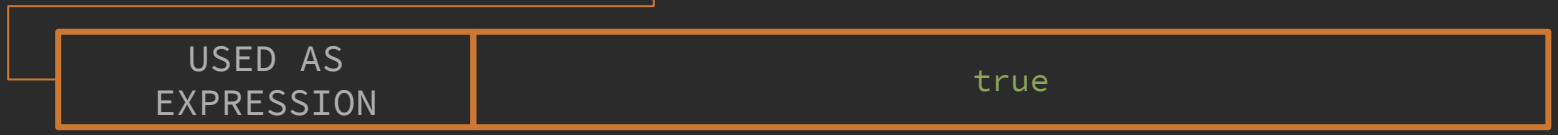
BindingContext?

```
fun hello(user: String) = "Hello, $user"
```



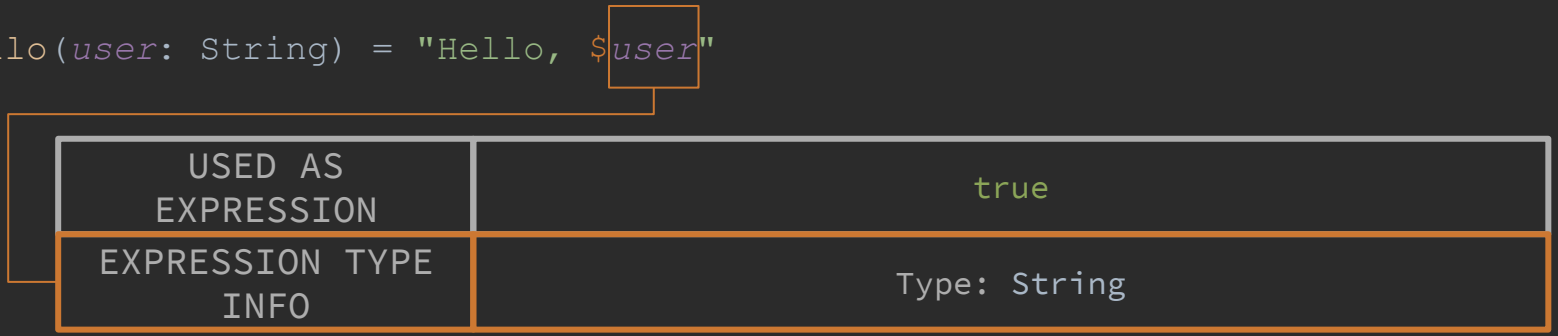
BindingContext?

```
fun hello(user: String) = "Hello, $user"
```



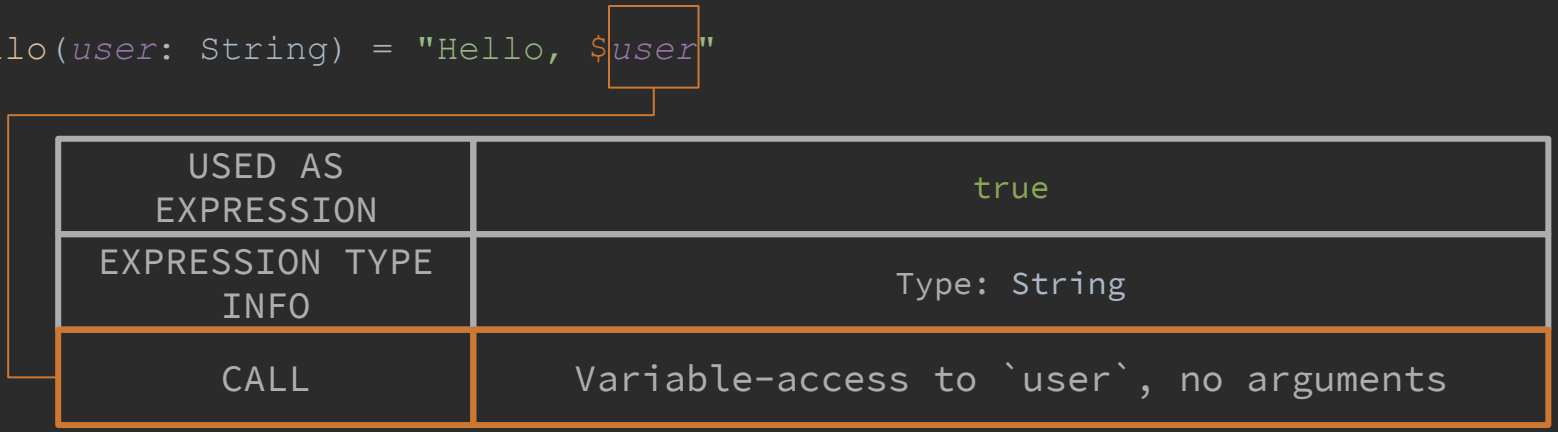
BindingContext?

```
fun hello(user: String) = "Hello, $user"
```



BindingContext?

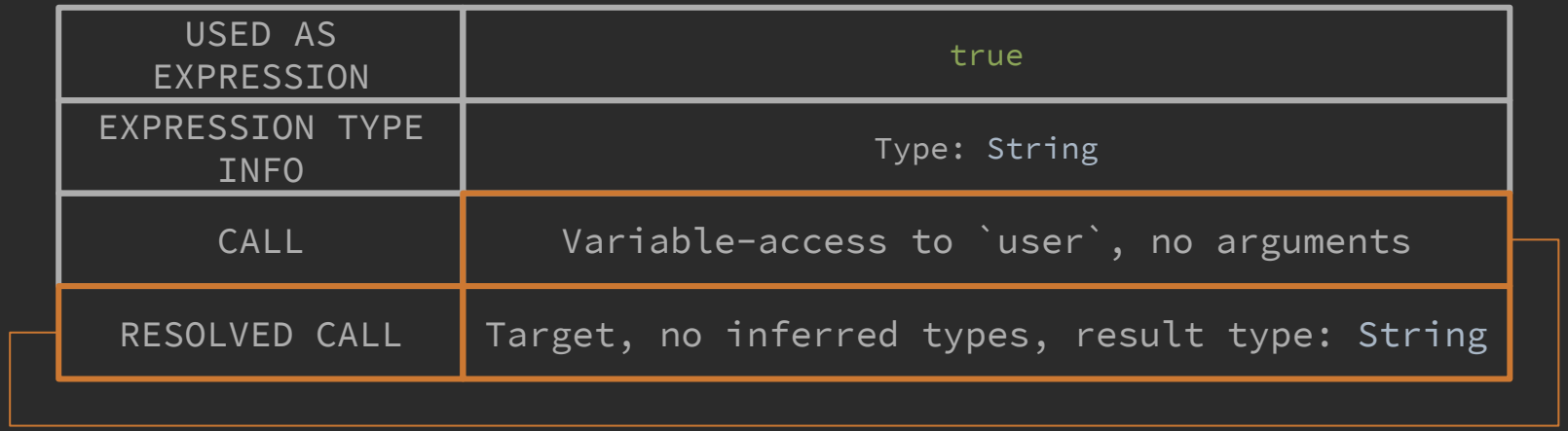
```
fun hello(user: String) = "Hello, $user"
```



USED AS EXPRESSION	true
EXPRESSION TYPE INFO	Type: String
CALL	Variable-access to `user`, no arguments

BindingContext?

```
fun hello(user: String) = "Hello, $user"
```



BindingContext?

```
fun hello(user: String) = "Hello, $user"
```

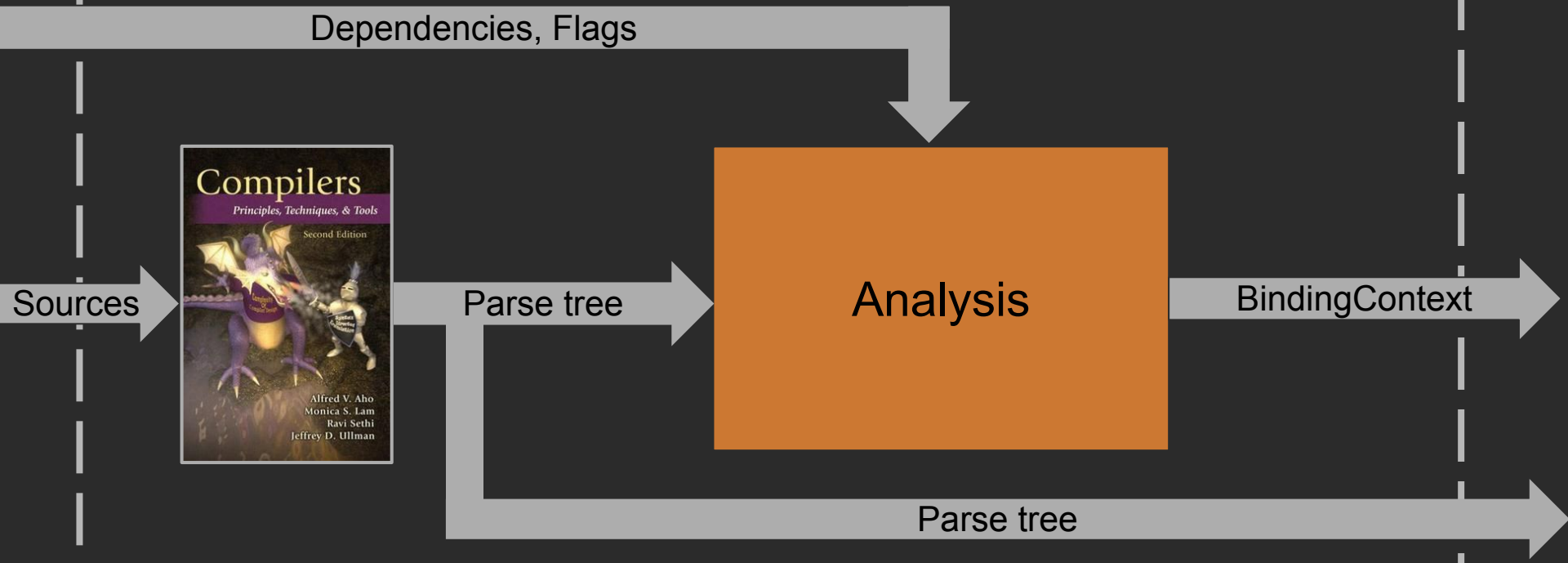
USED AS EXPRESSION	true
EXPRESSION TYPE INFO	Type: String
CALL	Variable-access to `user`, no arguments
RESOLVED CALL	Target, no inferred types, result type: String
REFERENCE TARGET	value-parameter user: kotlin.String defined in example.hello

BindingContext?

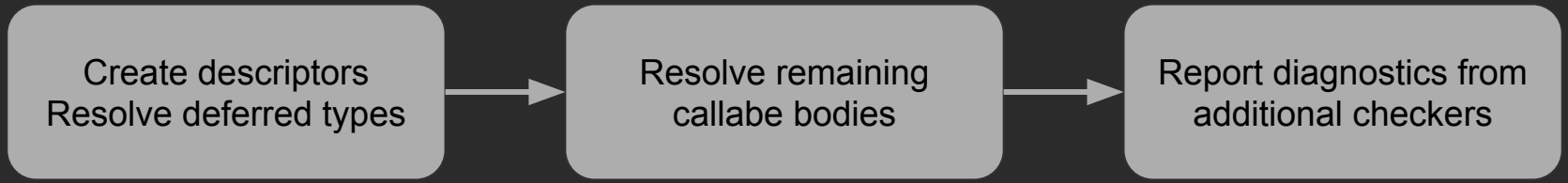
```
fun hello(user: String) = "Hello, $user"
```

USED AS EXPRESSION	true
EXPRESSION TYPE INFO	Type: String
CALL	Variable-access to `user`, no arguments
RESOLVED CALL	Target, no inferred types, result type: String
REFERENCE TARGET	value-parameter user: kotlin.String defined in example.hello

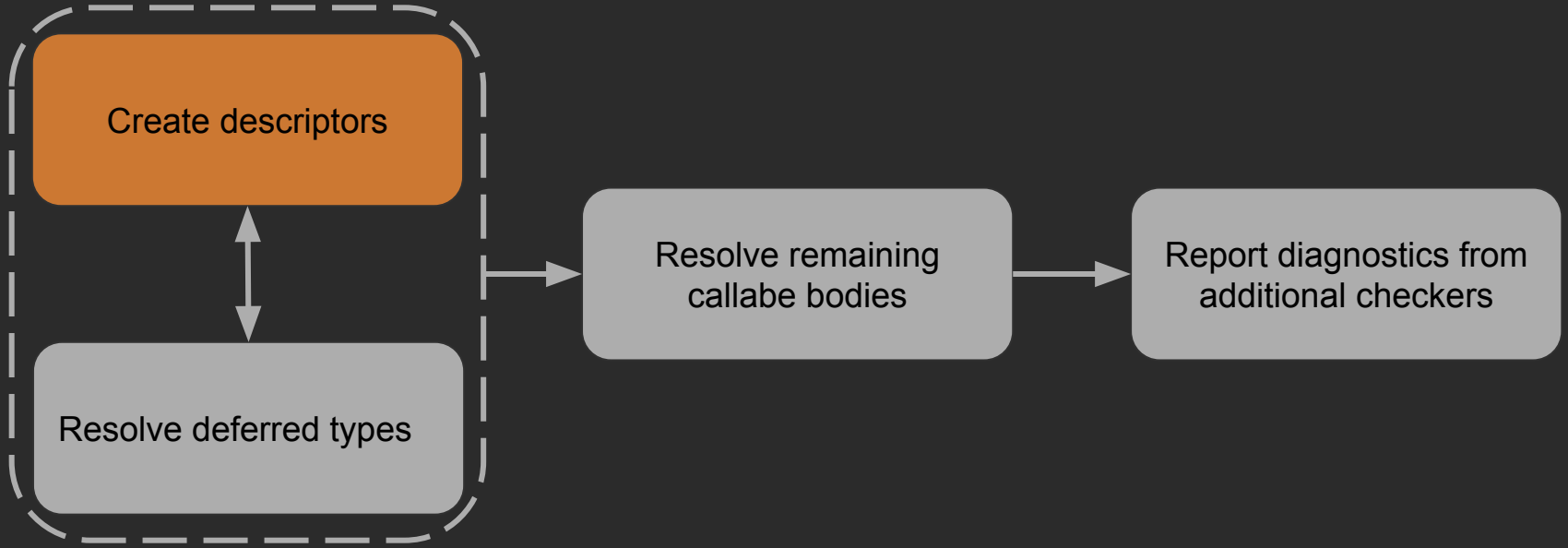
Frontend 1.0 on High-Level



Analysis


Create descriptors
Resolve deferred types
Resolve remaining
callabe bodies
Report diagnostics from
additional checkers

Analysis

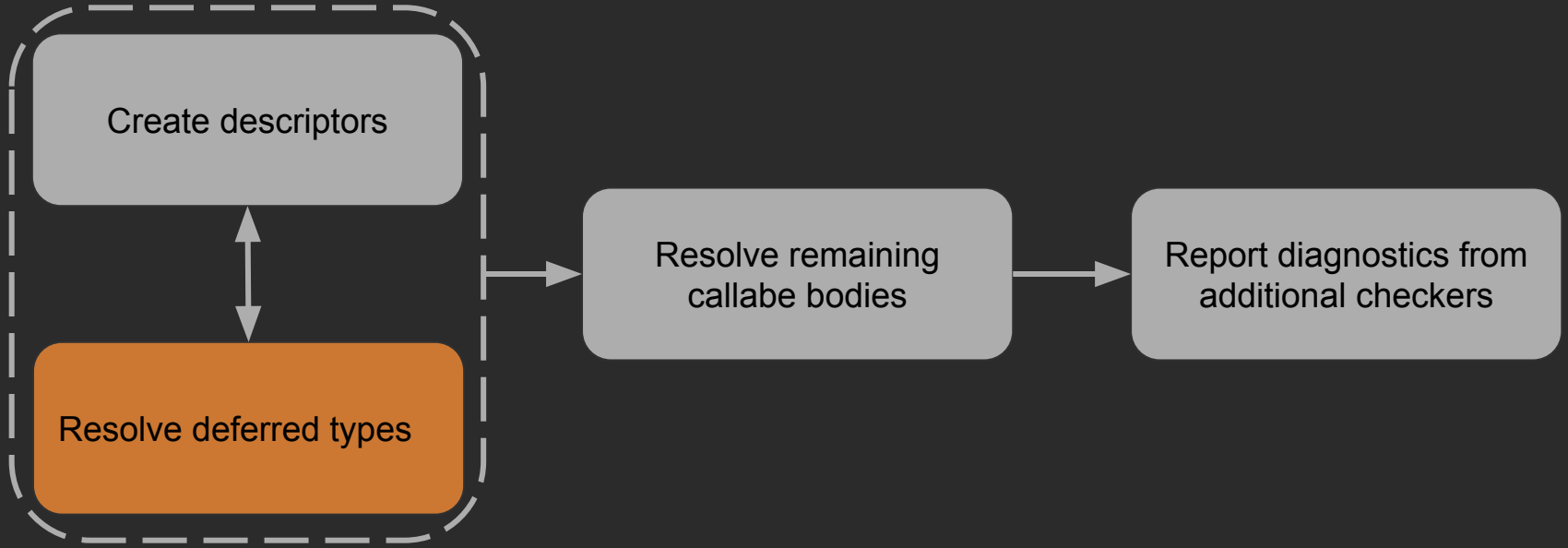

Create descriptors
Resolve deferred types
Resolve remaining callabe bodies
Report diagnostics from additional checkers

Declaring. DeclarationDescriptor

```
fun hello(user: String) = "Hello, $user"
```

FUNCTION	public fun hello(user: kotlin.String): kotlin.String defined in example in file Example.kt
VALUE PARAMETER	value-parameter user: kotlin.String defined in example.hello
REFERENCE TARGET	deserialized class String
TYPE	Type: String

Analysis


Create descriptors
Resolve deferred types
Resolve remaining callabe bodies
Report diagnostics from additional checkers

Lazy Analysis

```
fun hello() = helloWorld

val helloWorld = "Hello, world"
```

Lazy Analysis

```
fun hello(): ??? = helloWorld

val helloWorld = "Hello, world"
```

Lazy Analysis

```
fun hello(): ??? = helloWorld :???

val helloWorld = "Hello, world"
```

Lazy Analysis

```
fun hello(): ??? = helloWorld :???

val helloWorld: ??? = "Hello, world"
```

Lazy Analysis

```
fun hello(): ??? = helloWorld :???

val helloWorld: ??? = "Hello, world" :String
```

Lazy Analysis

```
fun hello(): ??? = helloWorld :???

val helloWorld: String = "Hello, world" :String
```

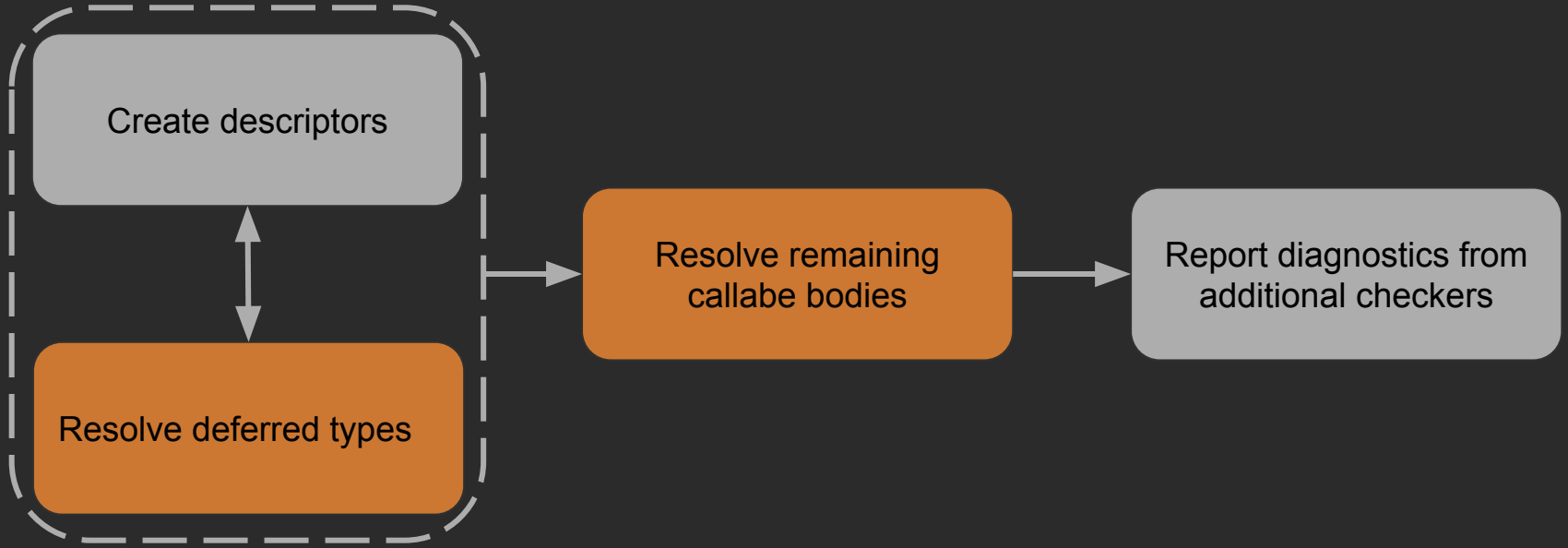
Lazy Analysis

```
fun hello(): ??? = helloWorld :String

val helloWorld: String = "Hello, world" :String
```

Lazy Analysis

```
fun hello(): String = helloWorld :String

val helloWorld: String = "Hello, world" :String
```

Analysis


Create descriptors
Resolve deferred types
Resolve remaining callabe bodies
Report diagnostics from additional checkers

How to Resolve a Call!

```
fun hello(name: String) = println(name)

class Hello {
    fun hello(user: String) = "Hello, $user"

    fun hey() {
        hello("Me")
    }
}
```

How to Resolve a Call!

```
fun hello(name: String) = println(name)

class Hello {
    fun hello(user: String) = "Hello, $user"

    fun hey() {
        hello("Me")
    }
}
```

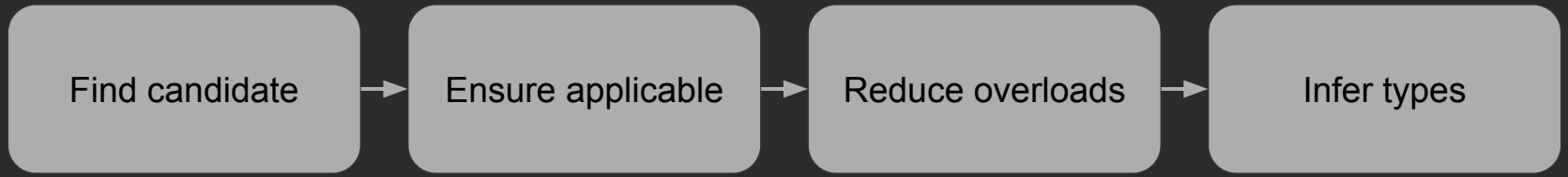
How to Resolve a Call!

```
fun hello(name: String) = println(name)

class Hello {
    fun hello(user: String) = "Hello, $user"

    fun hey() {
        hello("Me")
    }
}
```

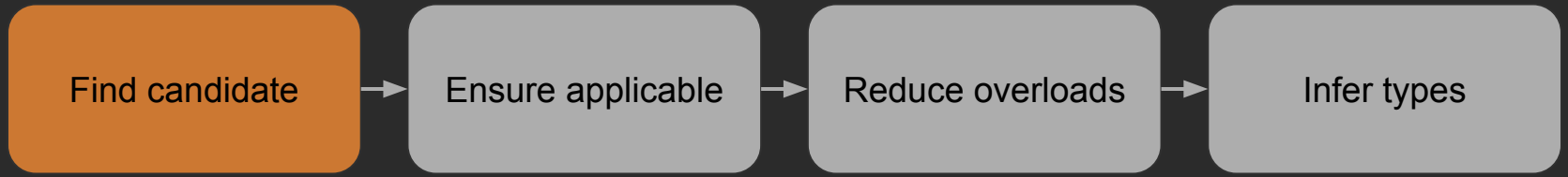
How to Resolve a Call!

```
fun hello(name: String) = println(name)

class Hello {
    fun hello(user: String) = "Hello, $user"

    fun hey() {
        hello("Me")
    }
}
```

How to Resolve a Call!



How to Resolve a Call!



Scopes

Package: helloworld

```kotlin
// FILE: hello.kt

package helloworld

import helloworld.world

fun hello(user: String) = "Hello, $user"
```

```kotlin
// FILE: world.kt

package helloworld

fun world() = "World"
```

Scopes

```kotlin
// FILE: hello.kt

package helloworld

import helloworld.world

fun hello(user: String) = "Hello, $user"
```

```kotlin
// FILE: world.kt

package helloworld

fun world() = "World"
```

Scopes

```kotlin
// FILE: hello.kt

package helloworld

import helloworld.world

fun hello(user: String) = "Hello, $user"
```

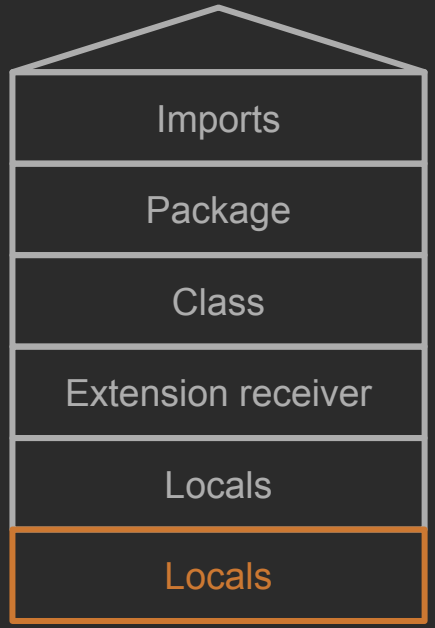
```kotlin
// FILE: world.kt

package helloworld

fun world() = "World"
```

Scopes

```kotlin
// FILE: hello.kt

package helloworld

import helloworld.world

fun hello(user: String) = "Hello, $user"
```

```kotlin
// FILE: world.kt

package helloworld

fun world() = "World"
```

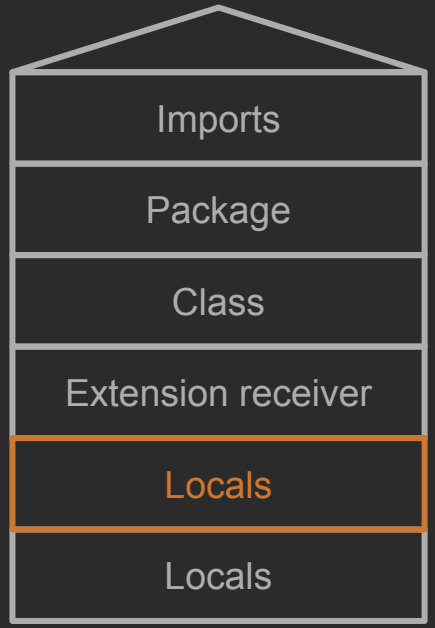
The Tower




```kotlin
// hello.kt
package example
import other.hello
class Hello {
    fun hello(d: D) = …
    fun World.hey() {
        fun hello(b: B) = …
        run {
            fun hello(a: A) = …
            hello(F())
        }
    }
}
```

```kotlin
// samePackage.kt
package example
class World {
    fun hello(c: C) = …
}
fun hello(e: E) = …

// other.kt
package other

fun hello(f: F) = …
```

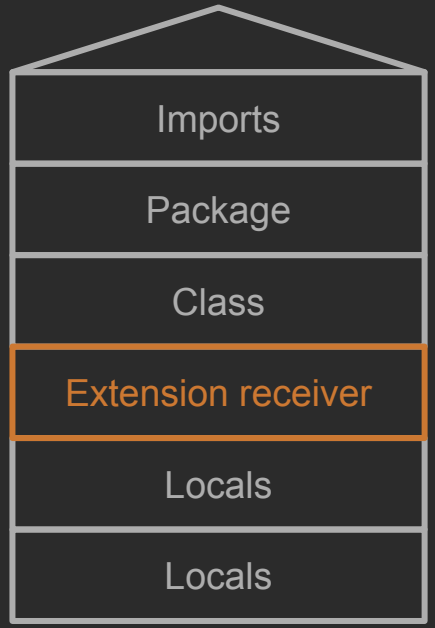
The Tower




```kotlin
// hello.kt
package example
import other.hello
class Hello {
    fun hello(d: D) = …
    fun World.hey() {
        fun hello(b: B) = …
        run {
            fun hello(a: A) = …
            hello(F())
        }
    }
}
```

```kotlin
// samePackage.kt
package example
class World {
    fun hello(c: C) = …
}
fun hello(e: E) = …

// other.kt
package other

fun hello(f: F) = …
```

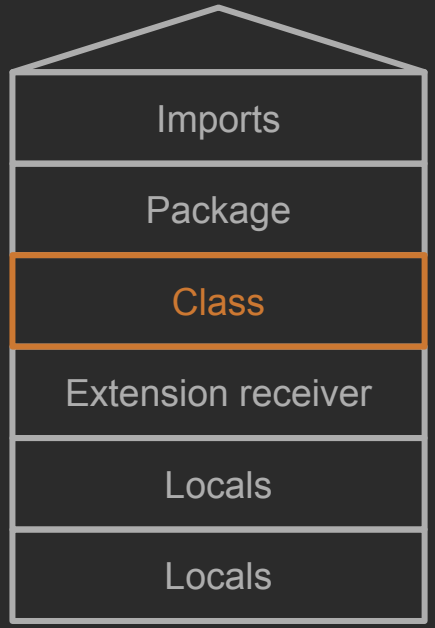
The Tower




```
// hello.kt
package example
import other.hello
class Hello {
    fun hello(d: D) = …
    fun World.hey() {
        fun hello(b: B) = …
        run {
            fun hello(a: A) = …
            hello(F())
        }
    }
}
```

```
// samePackage.kt
package example
class World {
    fun hello(c: C) = …
}
fun hello(e: E) = …

// other.kt
package other

fun hello(f: F) = …
```

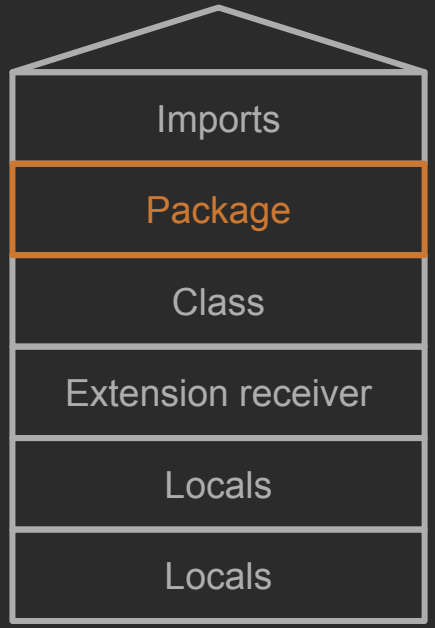
The Tower




```kotlin
// hello.kt
package example
import other.hello
class Hello {
    fun hello(d: D) = …
    fun World.hey() {
        fun hello(b: B) = …
        run {
            fun hello(a: A) = …
            hello(F())
        }
    }
}
```

```kotlin
// samePackage.kt
package example
class World {
    fun hello(c: C) = …
}
fun hello(e: E) = …

// other.kt
package other

fun hello(f: F) = …
```

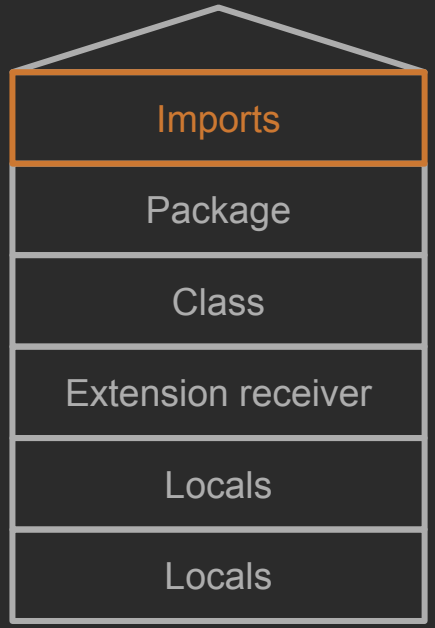
The Tower




```kotlin
// hello.kt
package example
import other.hello
class Hello {
    fun hello(d: D) = …
    fun World.hey() {
        fun hello(b: B) = …
        run {
            fun hello(a: A) = …
            hello(F())
        }
    }
}
```

```kotlin
// samePackage.kt
package example
class World {
    fun hello(c: C) = …
}
fun hello(e: E) = …

// other.kt
package other

fun hello(f: F) = …
```

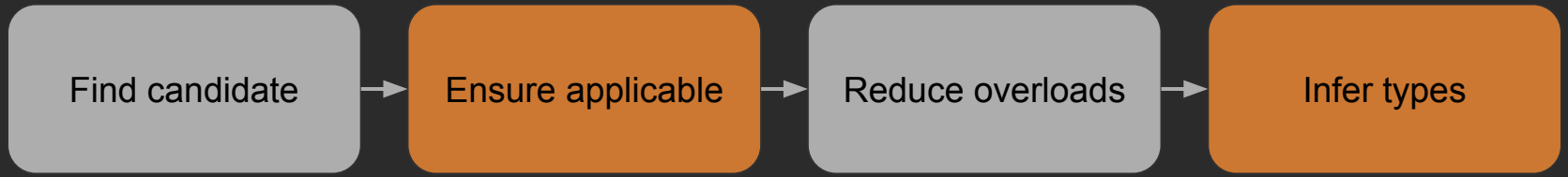
The Tower




```
// hello.kt
package example
import other.hello
class Hello {
    fun hello(d: D) = …
    fun World.hey() {
        fun hello(b: B) = …
        run {
            fun hello(a: A) = …
            hello(F())
        }
    }
}
```

```
// samePackage.kt
package example
class World {
    fun hello(c: C) = …
}
fun hello(e: E) = …

// other.kt
package other

fun hello(f: F) = …
```

How to Resolve Call!



Type Inference

```
fun <T> id(t: T): T = t

id("")
```

Type Inference

```
fun <T> id(t: T): T = t

id("")
```

```
T <: Any?
```

Type Inference

```
fun <T> id(t: T): T = t

id("")
```

```
T <: Any?
T :> Nothing
```

Type Inference

```
fun <T> id(t: T): T = t

id("")

T <: Any?
T :> Nothing
T :> String
```

Type Inference

```
fun <T> id(t: T): T = t

id("")
```

```
T <: Any?
T :> Nothing
T :> String

=>
T <:> String
```

Type Inference, Expression Level

```
fun <T> materialize(): T = TODO()

val x: Int = materialize()
```

Type Inference, Expression Level

```
fun <T> materialize(): T = TODO()

val x: Int = materialize()
```

```
T <: Any?
T :> Nothing
```

Type Inference, Expression Level

```
fun <T> materialize(): T = TODO()

val x: Int = materialize()
```

~~T <: Any?~~
T :> Nothing
T <: Int

=> T <:> Int

Smart-Casts

```
fun test(x: Any) {
    if (x is String) {
        x.length
    }
    x.length
}
```

Data-flow

Smart-Casts

```
fun test(x: Any) {
    if (x is String) {
        x.length
    }
    x.length
}
```

Data-flow

Smart-Casts

```
fun test(x: Any) {
    if (x is String) {
        x.length
    }
    x.length
}
```

Data-flow

Smart-Casts

```
fun test(x: Any) {
    if (x is String) {
        x.length
    }
    x.length
}
```

Data-flow

Smart-Casts

```
fun test(x: Any) {
    if (x is String) {
        x.length
    }
    x.length
}
```

Data-flow

Smart-Casts

```
fun test(x: Any) {
    if (x is String) {
        x.length
    }
    x.length
}
```

Data-flow

Smart-Casts: Quiz

```
interface A {
    fun foo()
}

fun bar(x: Any, b: Boolean) {
   while (true) {
       if (b) {
           x as A
           x.foo()
           break
       }
   }
}
```

Smart-Casts: Quiz

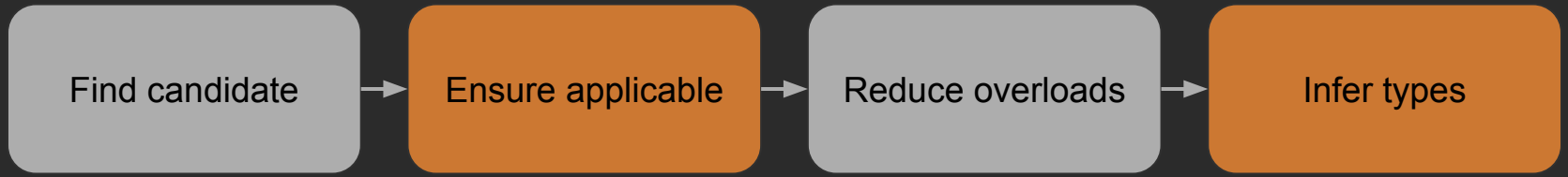
```
interface A {
    fun foo()
}

fun bar(x: Any, b: Boolean) {
   while (true) {
       if (b) {
           x as A
           break
       }
   }
   x.foo()
}
```

Frontend Timeline

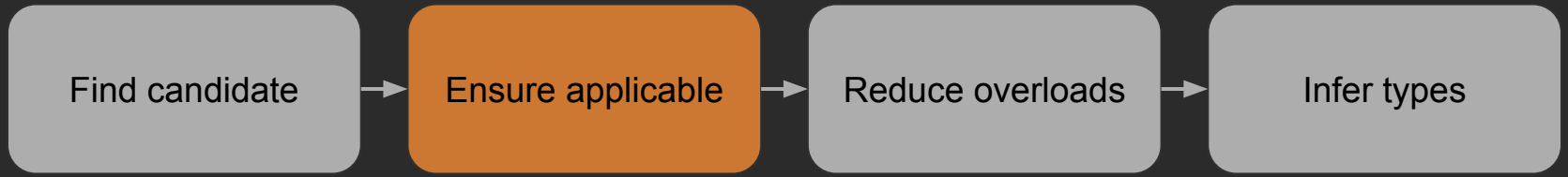
New Inference. Goals

- Call resolve unification
- Conversion improvements
- Better non-denotable types handling

New Inference



New Inference



New Inference. Conversions

KT-7770, KT-11129

```
interface Function1<in P, out R> { ... }
fun interface Action<U> {
   fun perform(data: U)
}

fun <T> enqueue(action: Action<T>) = ...

enqueue { it: String -> println(it) }
```

New Inference. Conversions

KT-7770, KT-11129

```
interface Function1<in P, out R> { ... }
fun interface Action<U> {
    fun perform(data: U)
}

fun <T> enqueue(action: Action<T>) = ...

enqueue { it: String -> println(it) }
```

New Inference. Conversions

KT-7770, KT-11129

```
interface Function1<in P, out R> { ... }
fun interface Action<U> {
    fun perform(data: U): Unit // Function1<U, Unit>
}

fun <T> enqueue(action: Action<T>) = ...

enqueue { it: String -> println(it) }
```

New Inference. Conversions

KT-7770, KT-11129

```
interface Function1<in P, out R> { ... }
fun interface Action<U> {
   fun perform(data: U): Unit // Function1<U, Unit>
}

fun <T> enqueue(action: Action<T>) = ...

enqueue { it: String -> println(it) }

Action<T> -> Function1<T, Unit>
 => P <:> T
    R :> Unit
```

New Inference. Conversions

KT-7770, KT-11129

```
interface Function1<in P, out R> { ... }
fun interface Action<U> {
   fun perform(data: U): Unit // Function1<U, Unit>
}

fun <T> enqueue(action: Action<T>) = ...

enqueue { it: String -> println(it) }
```

```
Action<T> -> Function1<T, Unit>
 => P <:> T
    R :> Unit
```

```
Function1<String, R>
 => P <: String
```

New Inference. Conversions

KT-7770, KT-11129

```
interface Function1<in P, out R> { ... }
fun interface Action<U> {
   fun perform(data: U): Unit // Function1<U, Unit>
}

fun <T> enqueue(action: Action<T>) = ...

enqueue { it: String -> println(it) }
```

```
Action<T> -> Function1<T, Unit>
 => P <:> T
    R :> Unit

 => P <:> String, R <:> Unit, T <:> String
```

```
Function1<String, R>
 => P <: String
```

New Inference. Intersection Types

```
fun <T> select(a: T, b: T): T = TODO()

val n = select(5, 5f)
```

New Inference. Intersection Types

```
fun <T> select(a: T, b: T): T = TODO()

val n = select(5, 5f)

T <: Any?
T :> Nothing
```

New Inference. Intersection Types

```
fun <T> select(a: T, b: T): T = TODO()

val n = select(5, 5f)

T <: Any?
T :> Nothing

T :> Int
T :> Float
```

New Inference. Intersection Types

```
fun <T> select(a: T, b: T): T = TODO()

val n = select(5, 5f) // Any

T <: Any?
T :> Nothing

T :> Int
T :> Float

=> T <:> Any // Old inference
```

New Inference. Intersection Types

```
fun <T> select(a: T, b: T): T = TODO()

val n = select(5, 5f) // Comparable<*> & Number

T <: Any?
T :> Nothing

T :> Int
T :> Float

=> T <:> Any // Old inference

T <:> CommonSupertype(Int, Float)
T <:> Comparable<*> & Number
```

New Inference. Intersection Types

```
fun <T> select(a: T, b: T): T = TODO()

val n = if (true) 5 else 5f // Implicit cast to Any in old inference
```

New Inference. Intersection Types

```
fun <T> select(a: T, b: T): T = TODO()

val n: Comparable<*> & Number = if (true) 5 else 5f
```

New Inference. System Joining

```
fun <T> id(t: T) = t
fun <R> materialize(): R = TODO()


fun foo(): String = id(materialize())
```

New Inference. System Joining

```
fun <T> id(t: T) = t
fun <R> materialize(): R = TODO()


fun foo(): String = id(materialize())


R <: Any?, not enough
```

New Inference. System Joining

```
fun <T> id(t: T) = t
fun <R> materialize(): R = TODO()


fun foo(): String = id(materialize())


R <: Any?, not enough

T <:> String, join system
```

New Inference. System Joining

```
fun <T> id(t: T) = t
fun <R> materialize(): R = TODO()


fun foo(): String = id(materialize())
```

R <: Any?, not enough

T <:> String, join system

R <: T => R <:> String

New Inference. System Joining

```
class Some(l: List<String>) {
    val s: Set<String> = l.toCollection(newSetFromMap(IdentityHashMap()))
}

fun <T, C : MutableCollection<in T>> Iterable<T>.toCollection(out: C): C = …
fun <K> newSetFromMap(map: MutableMap<K, Boolean>): MutableSet<K> = …
```

New Inference. System Joining

```
class Some(l: List<String>) {
    val s: Set<String> = l.toCollection(newSetFromMap(IdentityHashMap()))
}

fun <T, C : MutableCollection<in T>> Iterable<T>.toCollection(out: C): C = …
fun <K> newSetFromMap(map: MutableMap<K, Boolean>): MutableSet<K> = …
```

New Inference. System Joining

```
class Some(l: List<String>) {
    val s: Set<String> = l.toCollection(newSetFromMap(IdentityHashMap()))
}

fun <T, C : MutableCollection<in T>> Iterable<T>.toCollection(out: C): C = …
fun <K> newSetFromMap(map: MutableMap<K, Boolean>): MutableSet<K> = …

C <: Set<String>
```

New Inference. System Joining

```
class Some(l: List<String>) {
    val s: Set<String> = l.toCollection(newSetFromMap(IdentityHashMap()))
}

fun <T, C : MutableCollection<in T>> Iterable<T>.toCollection(out: C): C = …
fun <K> newSetFromMap(map: MutableMap<K, Boolean>): MutableSet<K> = …

C <: Set<String>
C :> MutableSet<K>
```

New Inference. System Joining

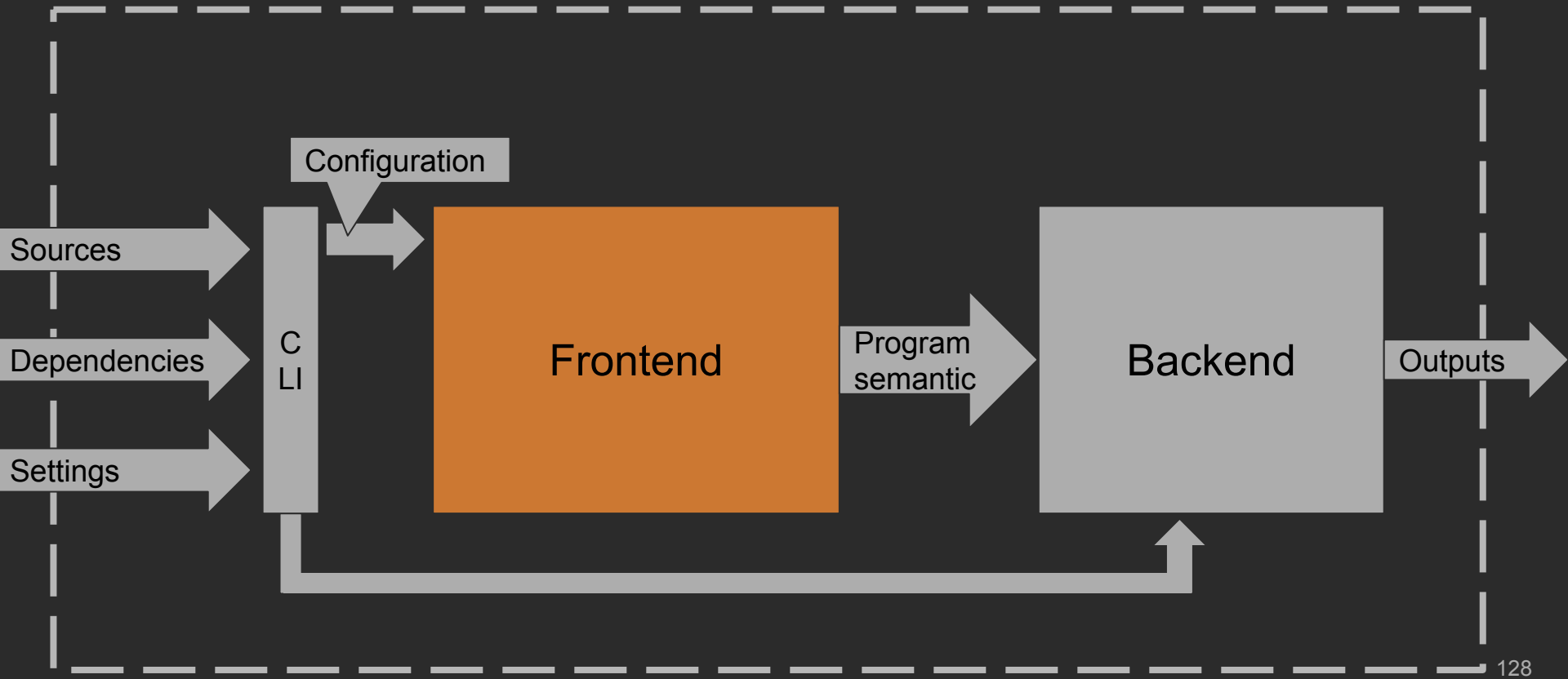
```
class Some(l: List<String>) {
    val s: Set<String> = l.toCollection(newSetFromMap(IdentityHashMap()))
}

fun <T, C : MutableCollection<in T>> Iterable<T>.toCollection(out: C): C = …
fun <K> newSetFromMap(map: MutableMap<K, Boolean>): MutableSet<K> = …

C <: Set<String>
C :> MutableSet<K>

  => MutableSet<K> <: Set<String>
```

New Inference. System Joining

```
class Some(l: List<String>) {
   val s: Set<String> = l.toCollection(newSetFromMap(IdentityHashMap()))
}

fun <T, C : MutableCollection<in T>> Iterable<T>.toCollection(out: C): C = …
fun <K> newSetFromMap(map: MutableMap<K, Boolean>): MutableSet<K> = …

C <: Set<String>
C :> MutableSet<K>

  => MutableSet<K> <: Set<String>
  => K <:> String
```

Frontend Timeline

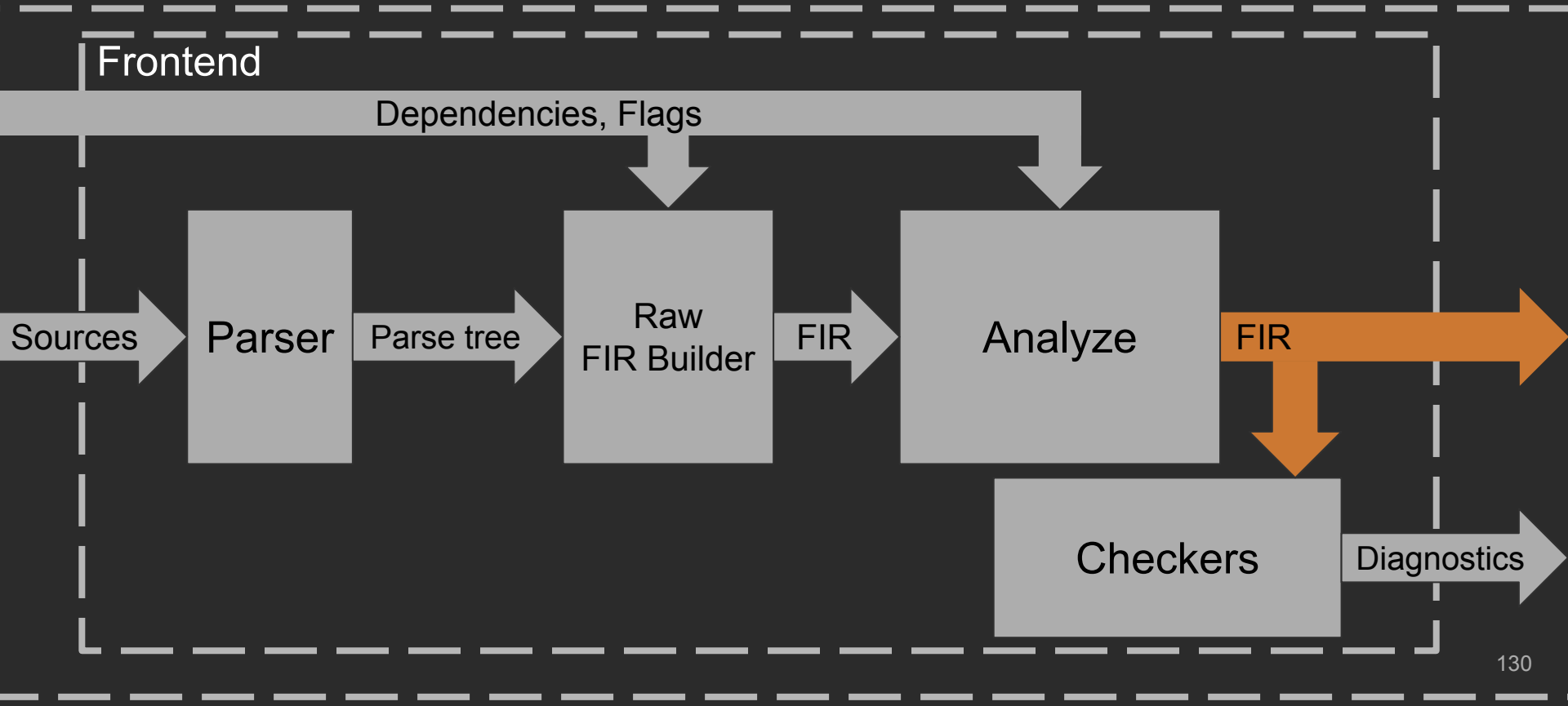
Compiler as a Transparent Box



New Frontend. Goals

- More traditional compiler approach
- Better performance
- Single main data structure to hold semantics

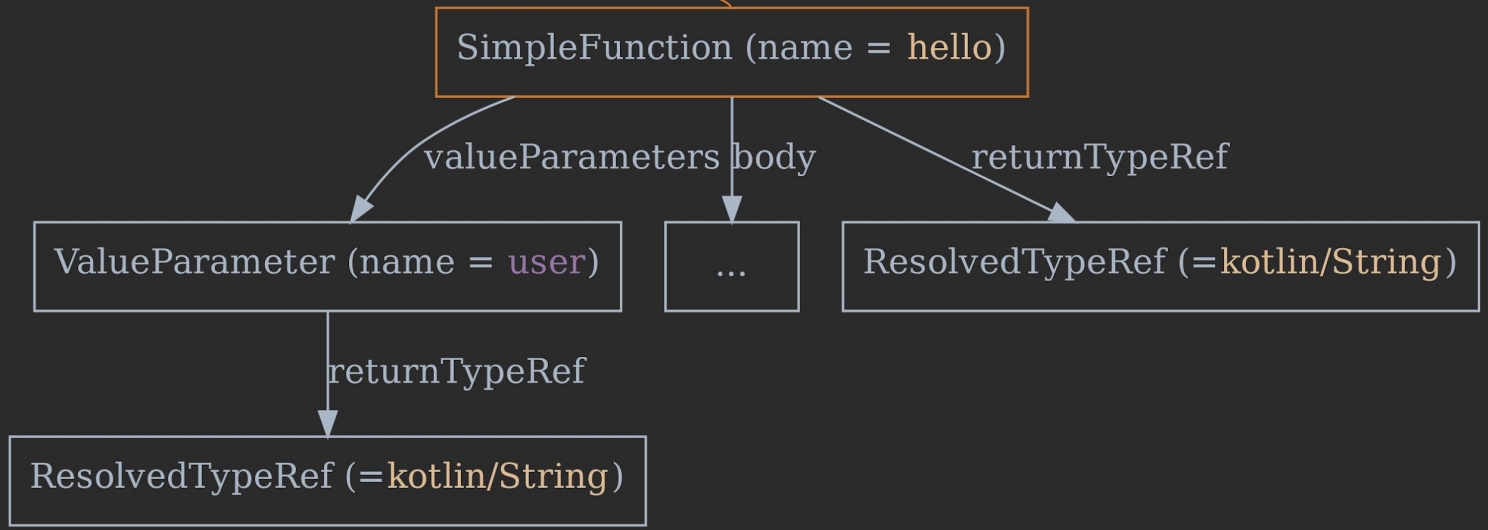
FIR/Frontend as a Transparent Box



FIR? Frontend Intermediate Representation! A Tree!

```
fun hello(user: String) = "Hello, $user"
```

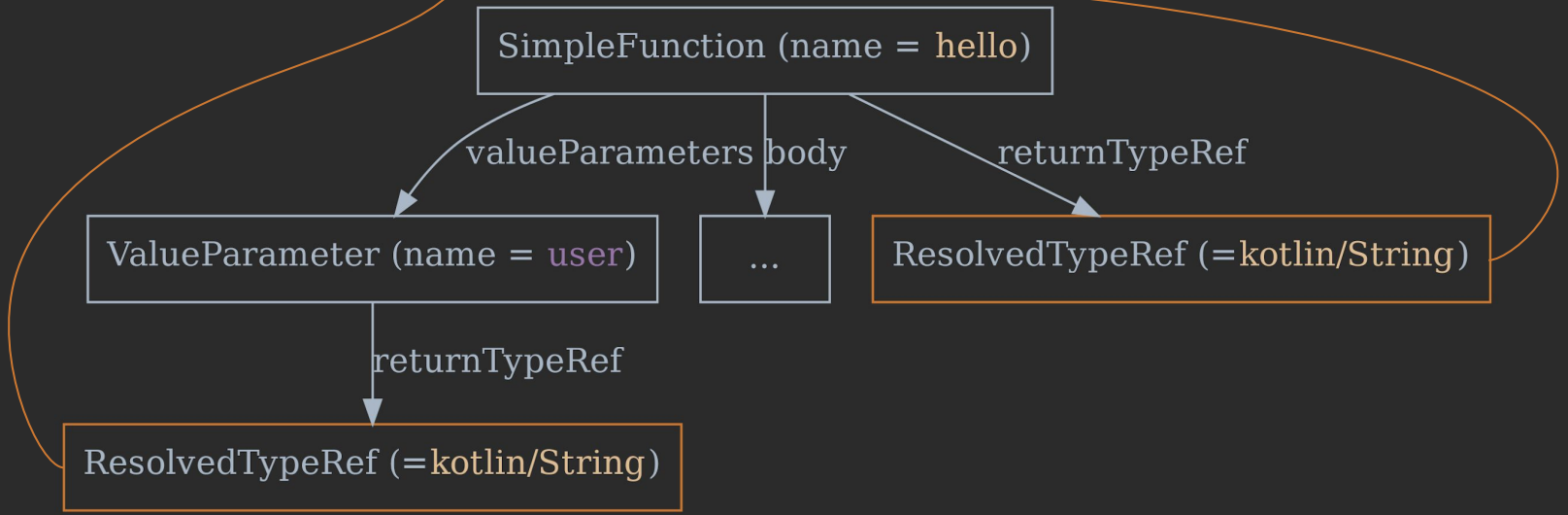
NOTE: AST



FIR? Frontend Intermediate Representation! A Tree!

```
fun hello(user: String) = "Hello, $user"
```

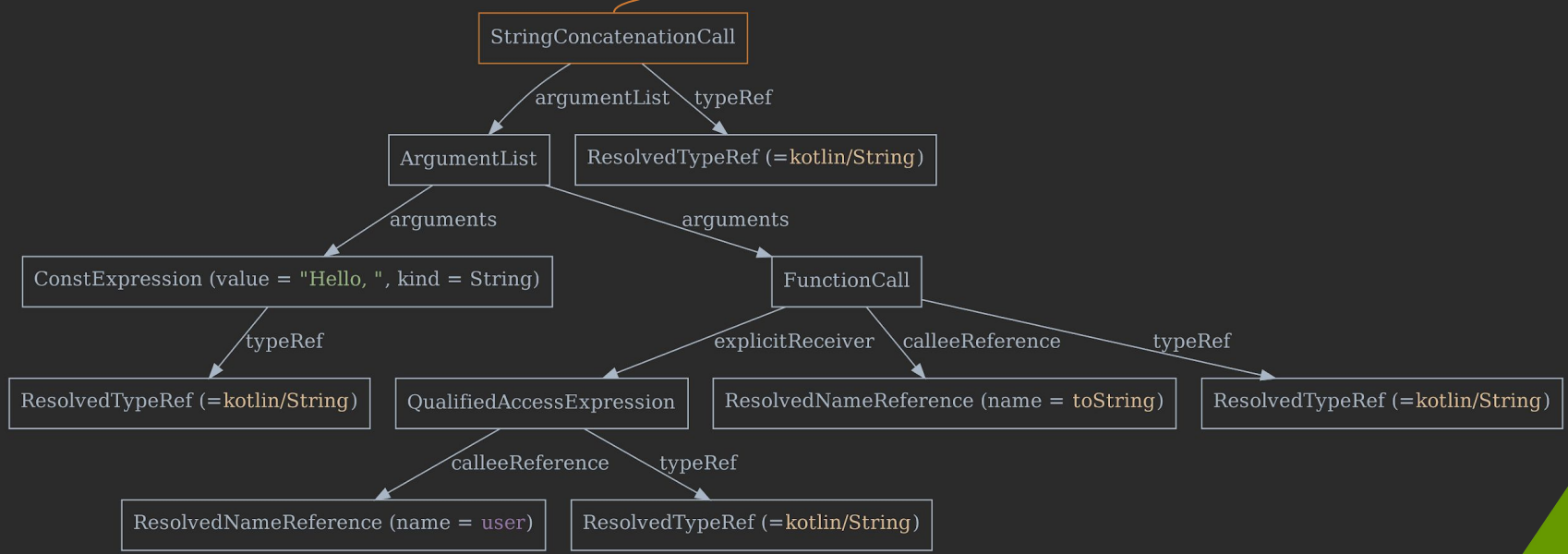
NOTE: AST



FIR? Frontend Intermediate Representation! A Tree!

```
fun hello(user: String) = "Hello, $user"
```

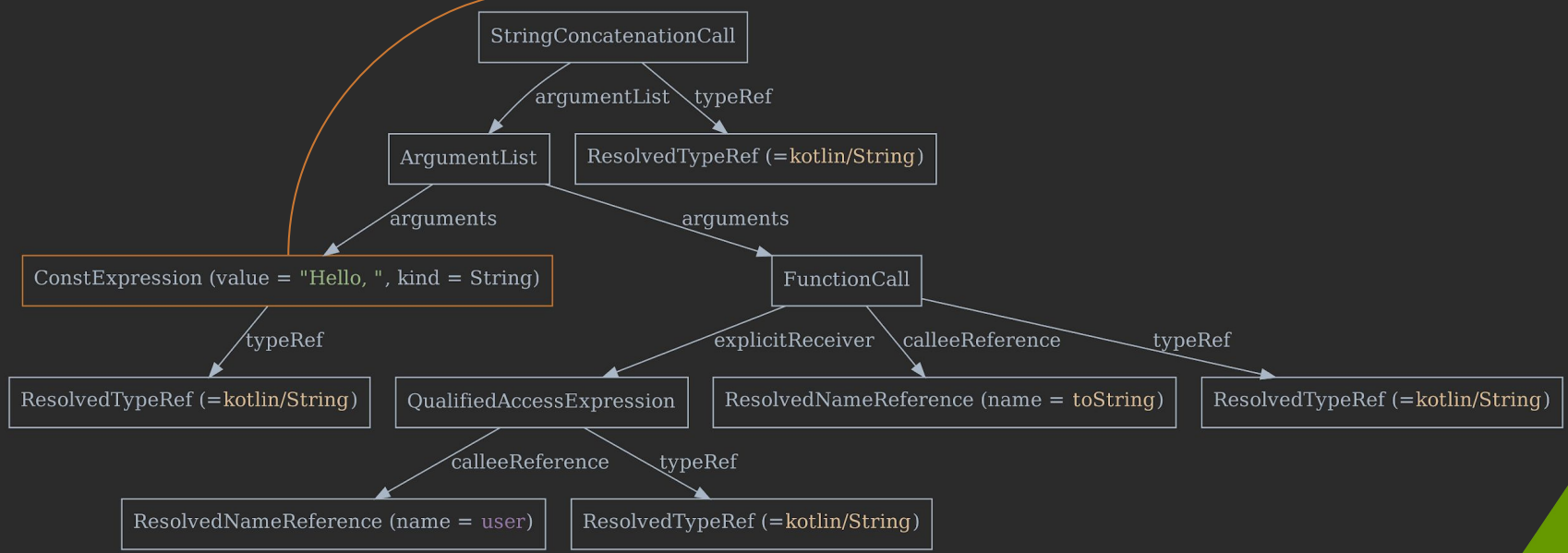
NOTE: AST



FIR? Frontend Intermediate Representation! A Tree!

```
fun hello(user: String) = "Hello, $user"
```

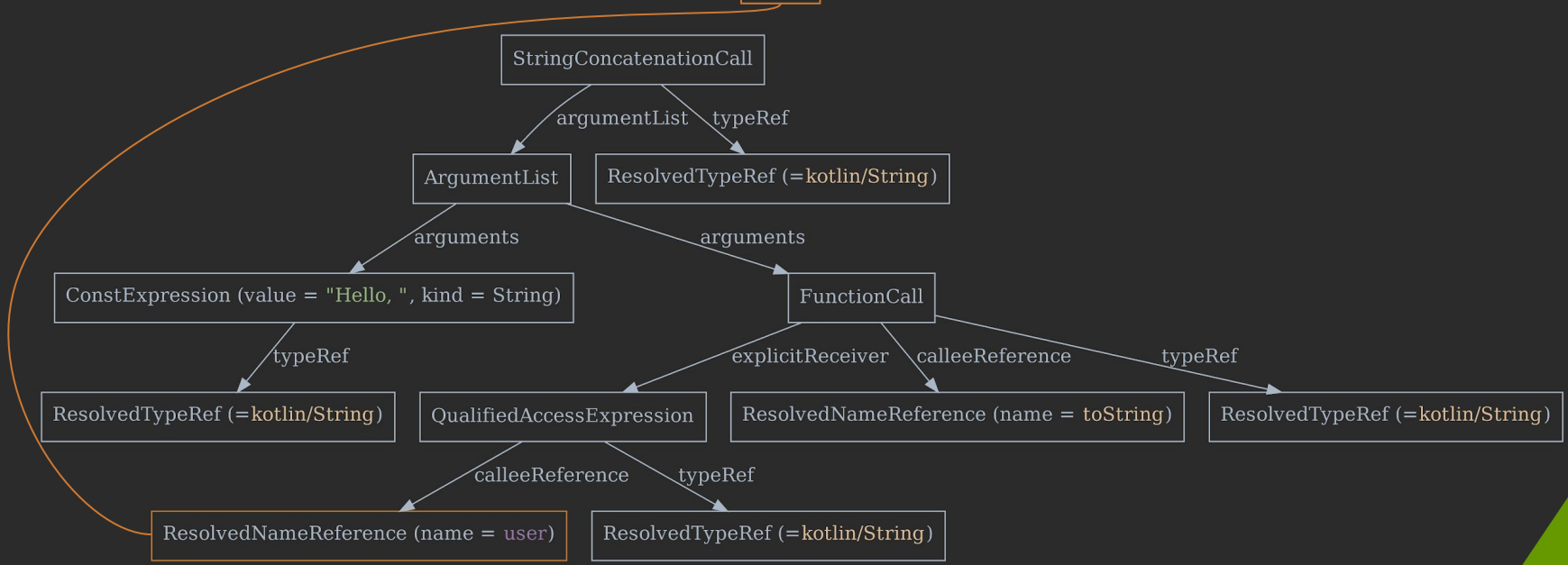
NOTE: AST



FIR? Frontend Intermediate Representation! A Tree!

```
fun hello(user: String) = "Hello, $user"
```

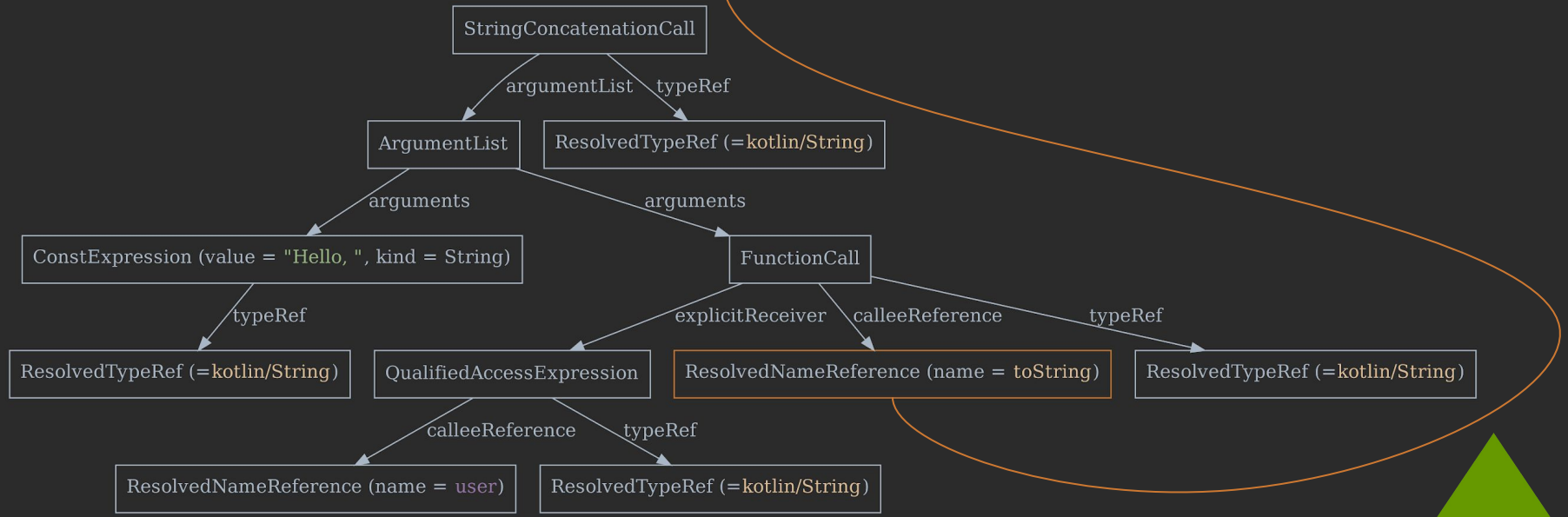
NOTE: AST



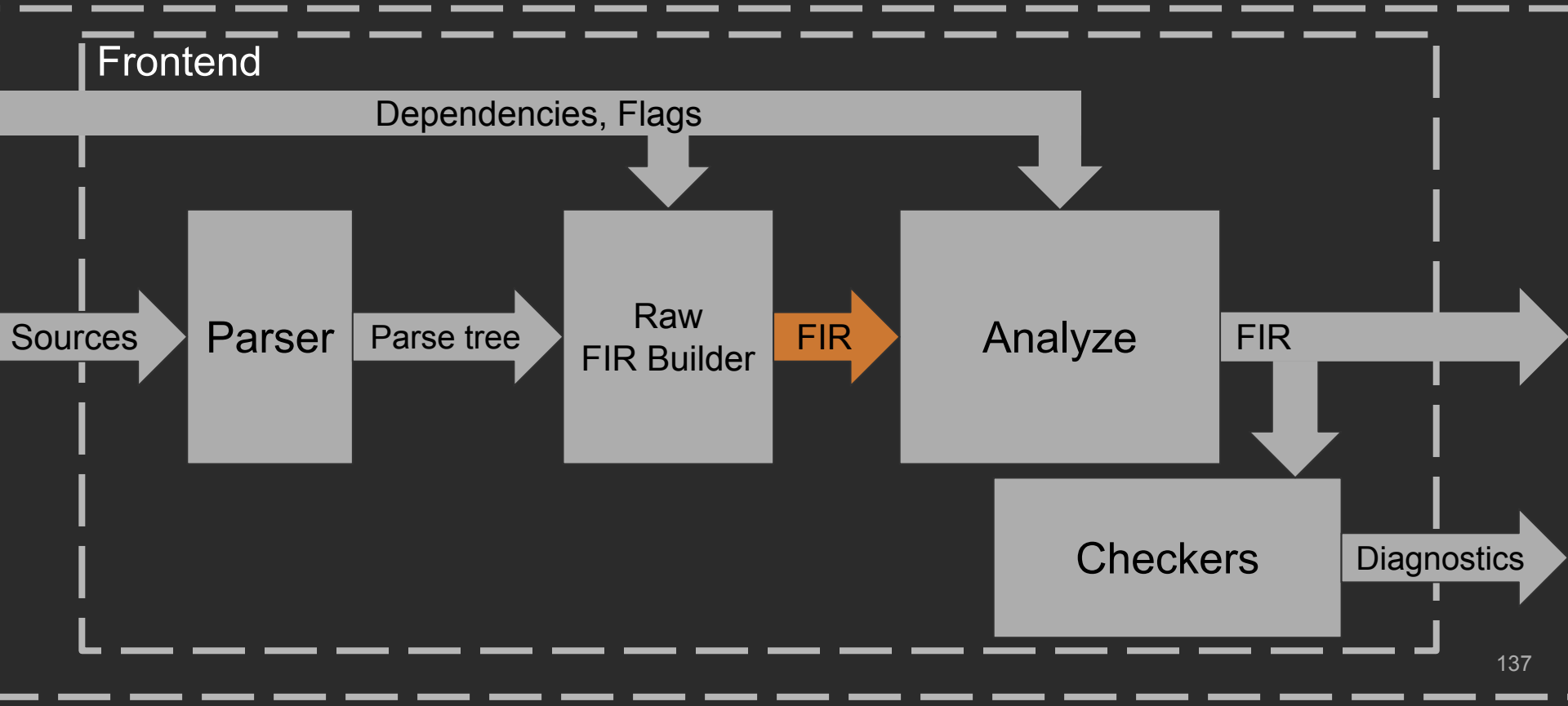
FIR? Frontend Intermediate Representation! A Tree!

```
fun hello(user: String) = "Hello, $user"
```

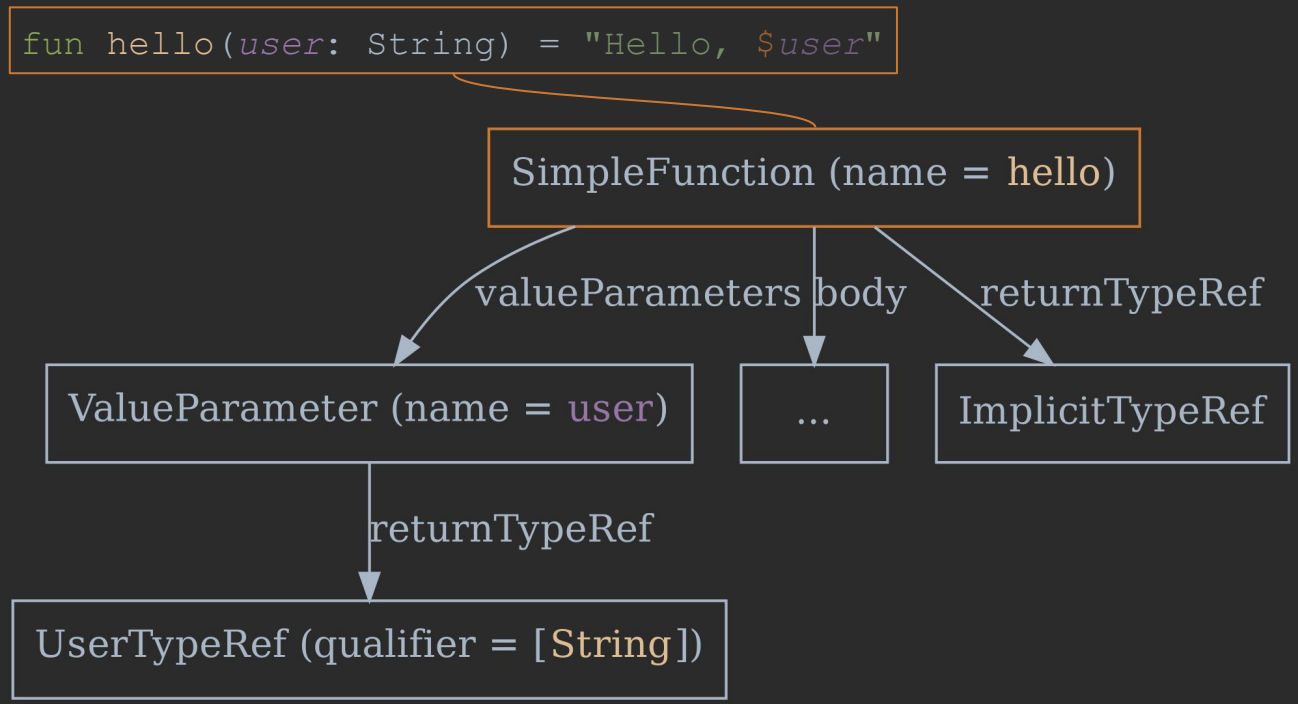
NOTE: AST



FIR/Frontend as a Transparent Box

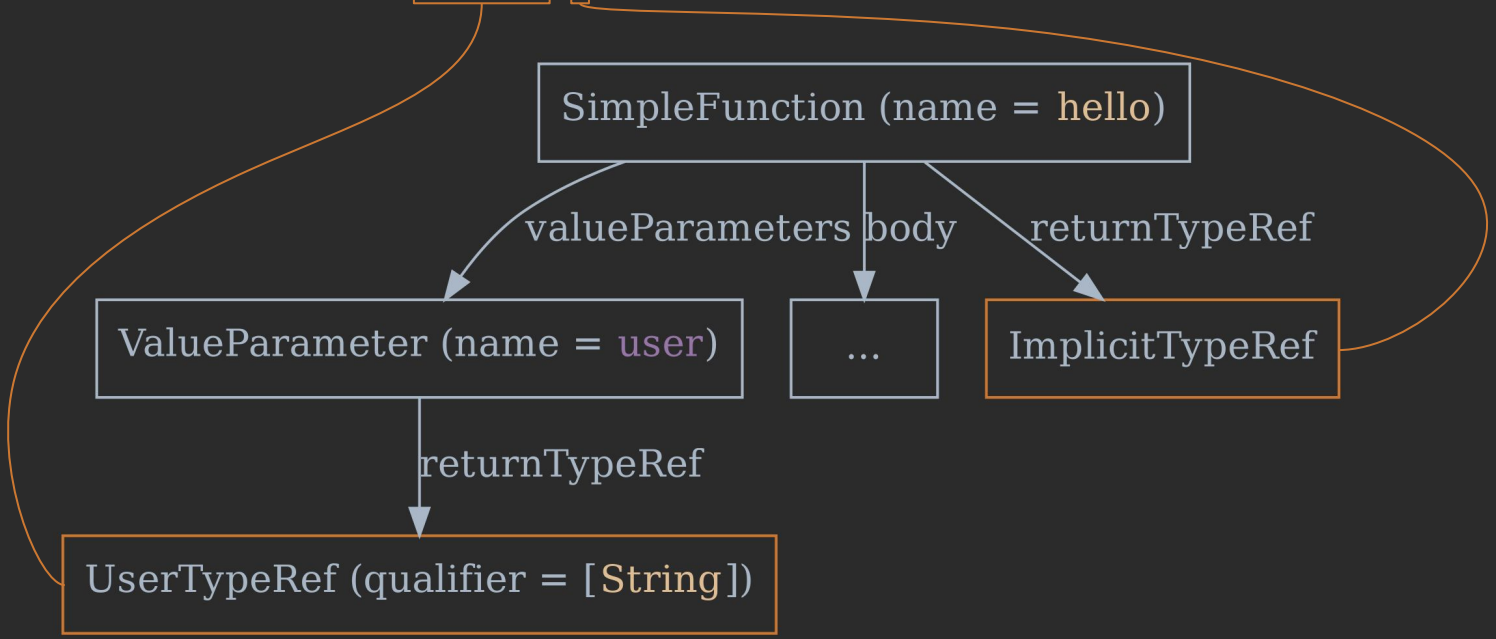


Raw FIR


fun hello(user: String) = "Hello, $user"
SimpleFunction (name = hello)
valueParameters
body
returnTypeRef
ValueParameter (name = user)
...
ImplicitTypeRef
returnTypeRef
UserTypeRef (qualifier = [String])

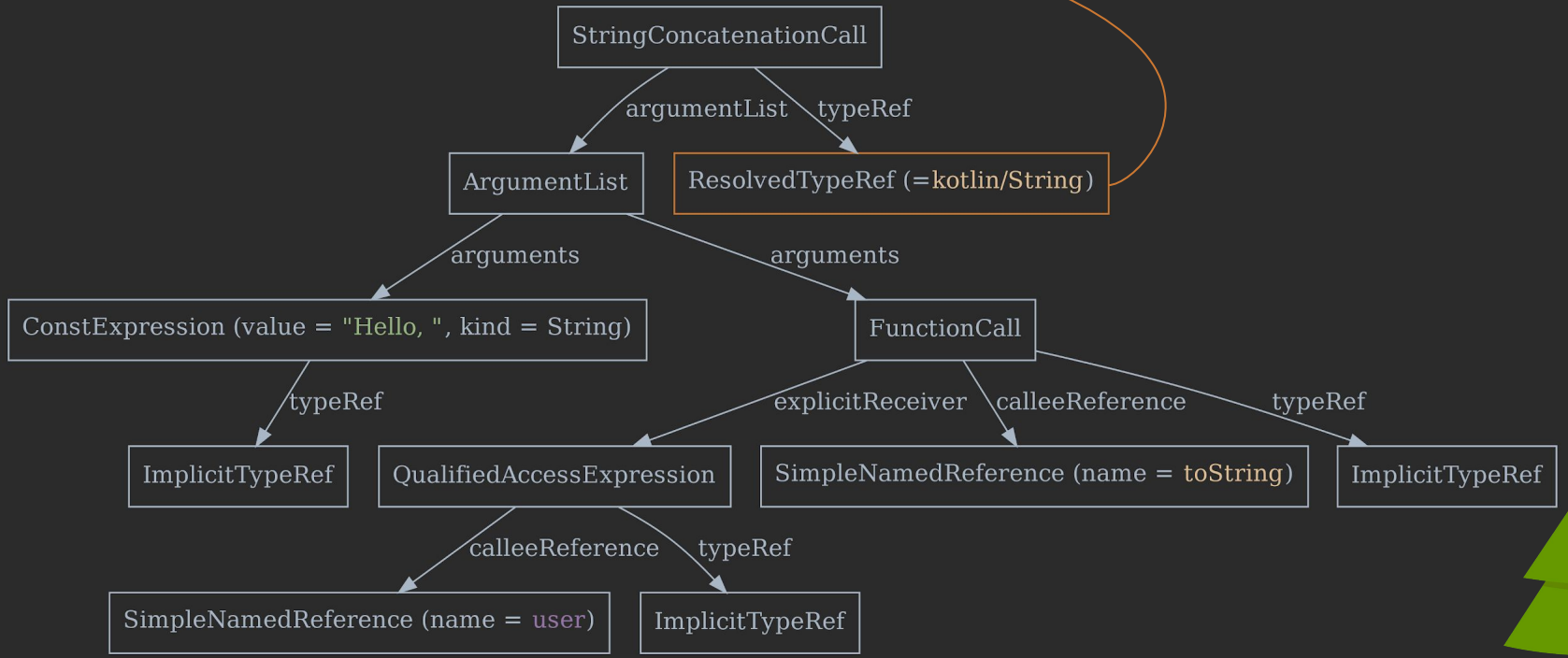
Raw FIR

```
fun hello(user: String) = "Hello, $user"
```



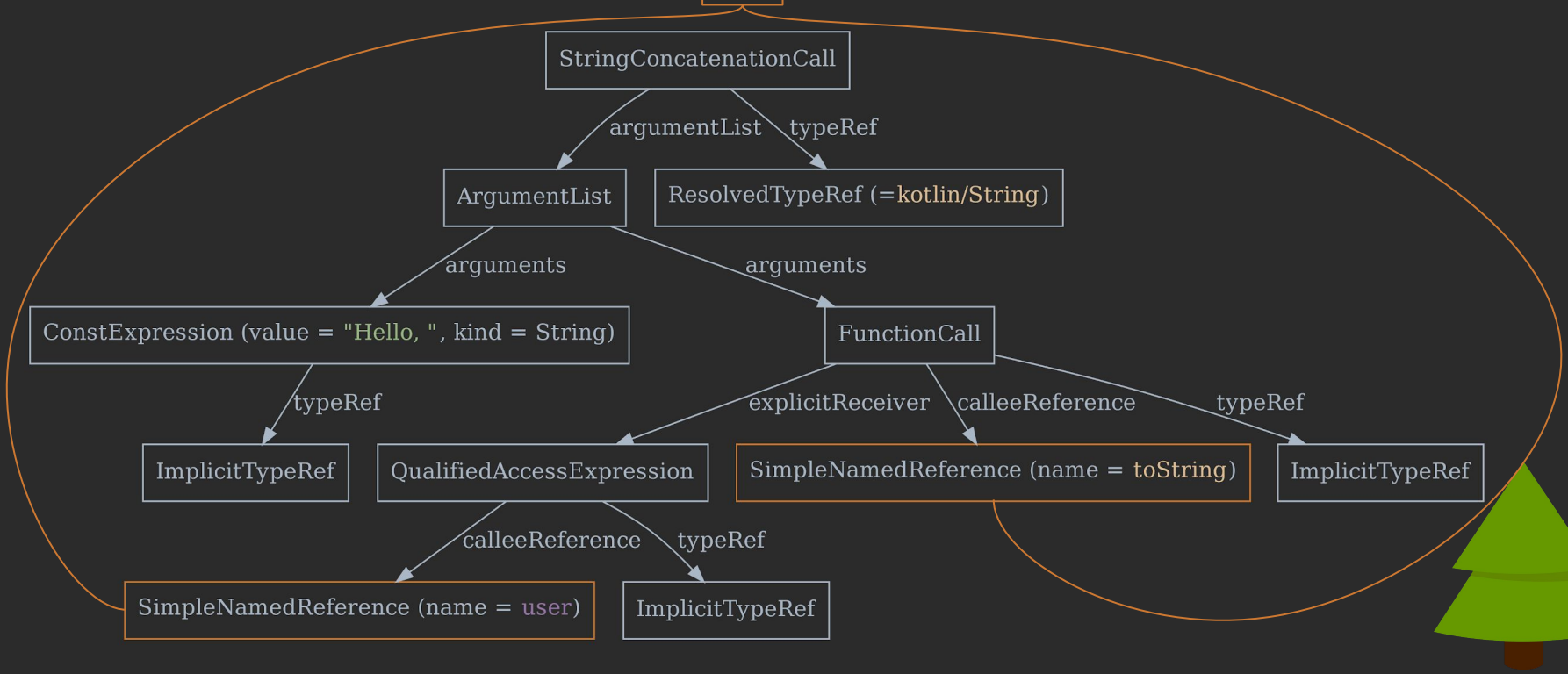
Raw FIR

```
fun hello(user: String) = "Hello, $user"
```



Raw FIR

```
fun hello(user: String) = "Hello, $user"
```



Desugaring

- IF, ?: to when
- For loop to while
- Destructuring declarations

Desugaring

```
if (b) {
    println("Hello")
}
```

```
when {
    b -> println("Hello")
}
```

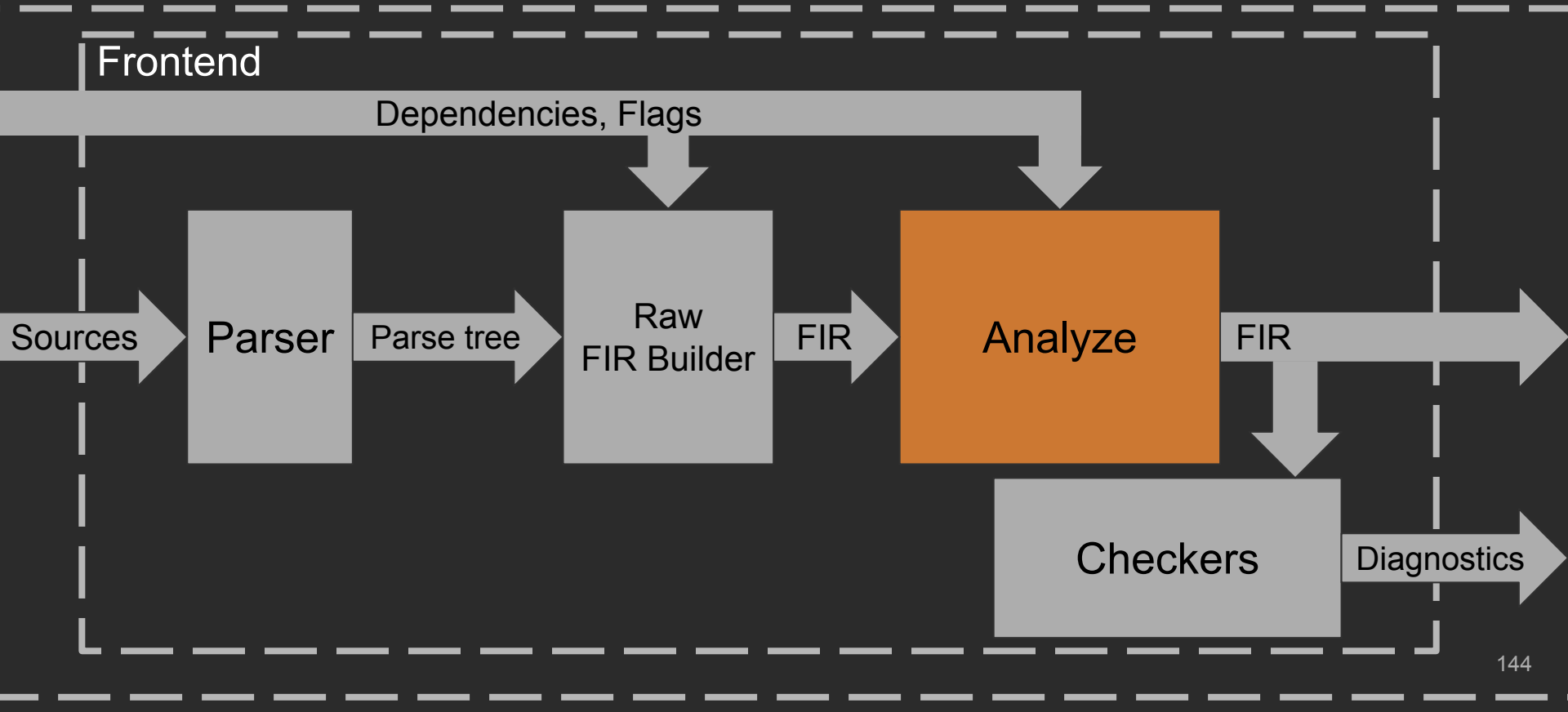
```
for (s in list) {
    println(s)
}
```

```
val <iterator> = list.iterator()
while (<iterator>.hasNext()) {
    val s = <iterator>.next()
    println(s)
}
```

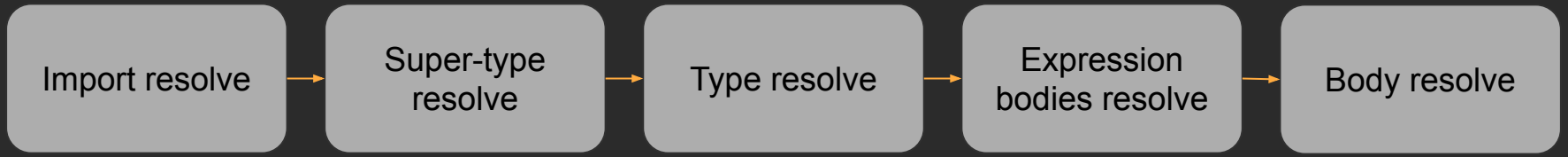
```
val (a, b) = "a" to "b"
```

```
val <destruct> = "a" to "b"
val a = <destruct>.component1()
val b = <destruct>.component2()
```

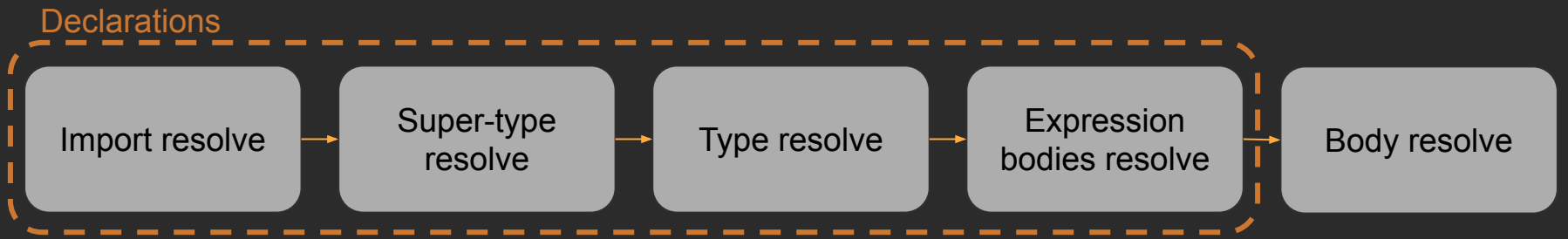
FIR/Frontend as a Transparent Box



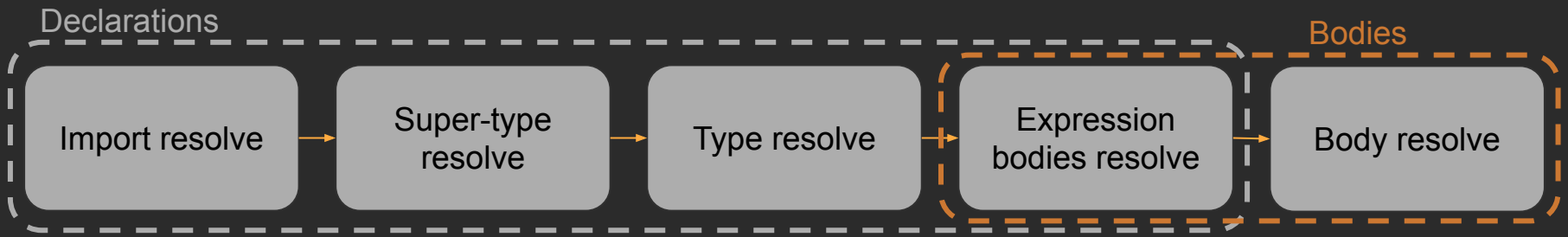
Batch Analysis



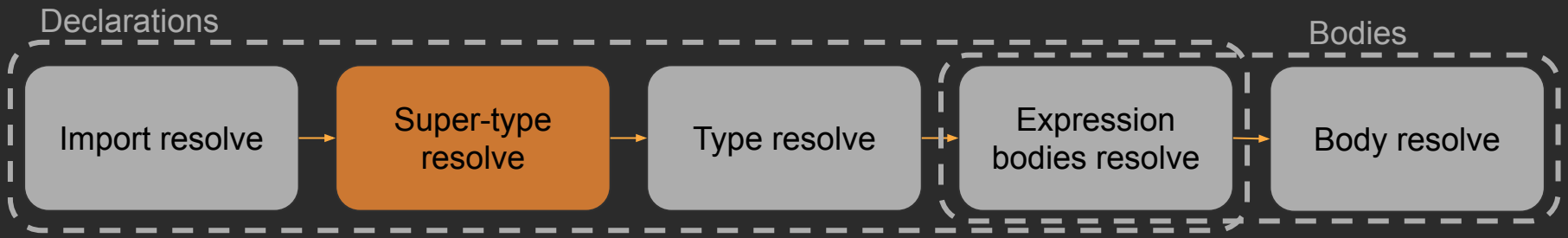
Batch Analysis



Batch Analysis



Batch Analysis



Supertype Resolve

```
// FILE: TopLevel.kt
class TopLevel : Middle() {
    class Nested : BaseNested()
}

// FILE: Middle.kt
open class Middle : Base() {}

// FILE: Base.kt
open class Base {
    open class BaseNested
}
```

Supertype Resolve

```
// FILE: TopLevel.kt
class TopLevel : Middle() {
    class Nested : BaseNested()
}

// FILE: Middle.kt
open class Middle : Base() {}

// FILE: Base.kt
open class Base {
    open class BaseNested
}
```

Supertype Resolve

```
// FILE: TopLevel.kt
class TopLevel : Middle() {
    class Nested : BaseNested()
}

// FILE: Middle.kt
open class Middle : Base() {}

// FILE: Base.kt
open class Base {
    open class BaseNested
}
```

Supertype Resolve

```
// FILE: TopLevel.kt
class TopLevel : Middle() {
    class Nested : BaseNested()
}

// FILE: Middle.kt
open class Middle : Base() {}

// FILE: Base.kt
open class Base {
    open class BaseNested
}
```

Supertype Resolve

```
// FILE: TopLevel.kt
class TopLevel : Middle() {
    class Nested : BaseNested()
}

// FILE: Middle.kt
open class Middle : Base() {}

// FILE: Base.kt
open class Base {
    open class BaseNested
}
```

Supertype Resolve

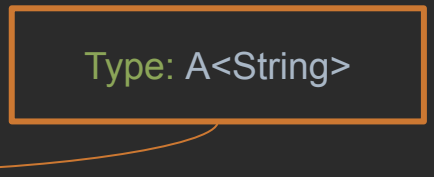
```
// FILE: TopLevel.kt
class TopLevel : Middle() {
    class Nested : BaseNested()
}

// FILE: Middle.kt
open class Middle : Base() {}

// FILE: Base.kt
open class Base {
    open class BaseNested
}
```

Supertype Resolve

```
// FILE: TopLevel.kt
class TopLevel : Middle() {
    class Nested : BaseNested()
}

// FILE: Middle.kt
open class Middle : Base() {}

// FILE: Base.kt
open class Base {
    open class BaseNested
}
```

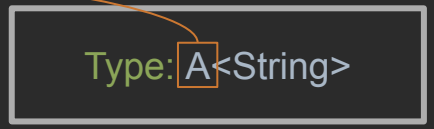
Types in FIR, for MPP

```
// MODULE: A

class A<T> {}
fun get(): A<String> = ...
```



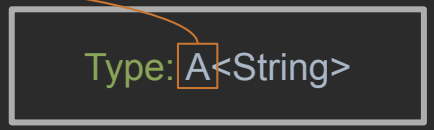
Types in FIR, for MPP

```
// MODULE: A

class A<T> {}
fun get(): A<String> = ...
```



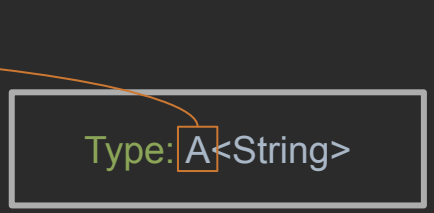
Types in FIR, for MPP

```
// MODULE: A

class A<T> {}
fun get(): A<String> = ...


// MODULE: B
class A<T> {
    fun foo() = ...
}
fun bar() = get().foo() // UNRESOLVED
```



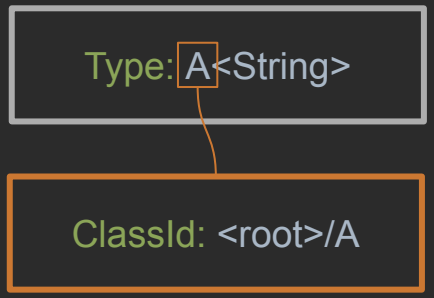
Types in FIR, for MPP

```
// MODULE: A

class A<T> {}
fun get(): A<String> = ...


// MODULE: B
class A<T> {
    fun foo() = ...
}
fun bar() = get().foo() // UNRESOLVED
```



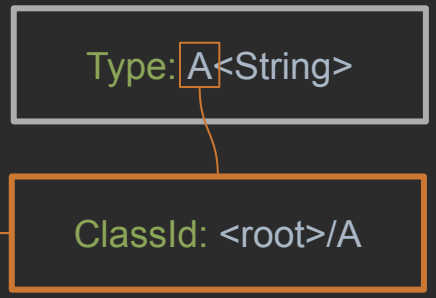
Types in FIR, for MPP

```
// MODULE: A

class A<T> {}
fun get(): A<String> = ...


// MODULE: B
class A<T> {
    fun foo() = ...
}
fun bar() = get().foo() // Works now
```



Types in FIR, for MPP

```
// MODULE: A
class A<T> {}
fun get(): A<String> = ...

// MODULE: B
class A<T> {
    fun foo() = ...
}
fun bar() = get().foo() // Works now
```



Types in FIR, for MPP

```
// MODULE: A
expect class A<T> {}
fun get(): A<String> = …

// MODULE: A_platform
actual class A<T> {
    fun foo() = ...
}

// MODULE: B
fun bar() = get().foo() // Works now
```

Smart-Casts: Quiz

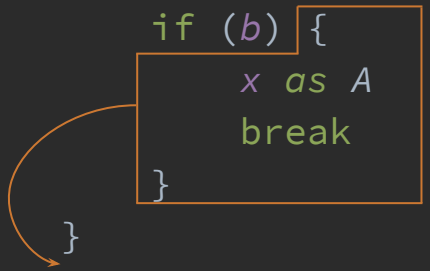
```
interface A {
    fun foo()
}

fun bar(x: Any, b: Boolean) {
   while (true) {
       if (b) {
           x as A
           break
       }
   }
   x.foo()
}
```

Smart-Casts: Quiz

```
interface A {
    fun foo()
}

fun bar(x: Any, b: Boolean) {
   while (true) {
       if (b) {
           x as A
           break
       }
   }
   x.foo()
}
```

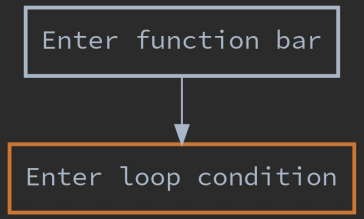
Smart-Casts: Quiz

```
interface A {
    fun foo()
}

fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



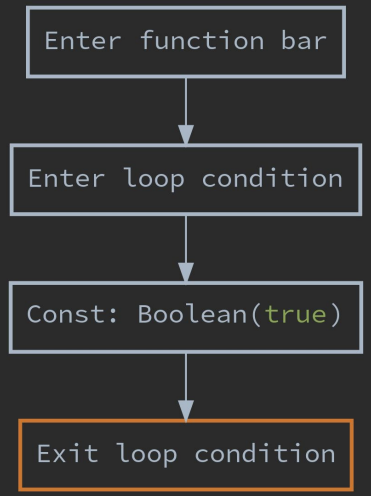
Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



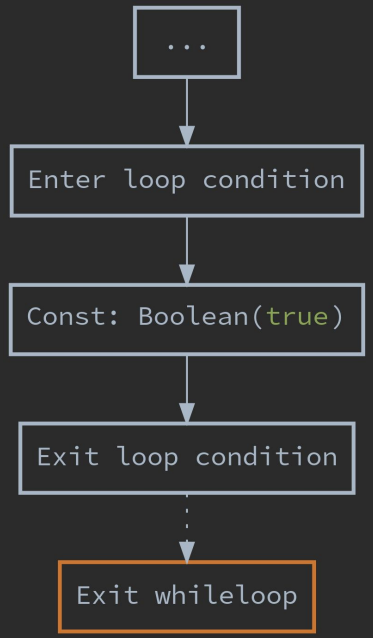
Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



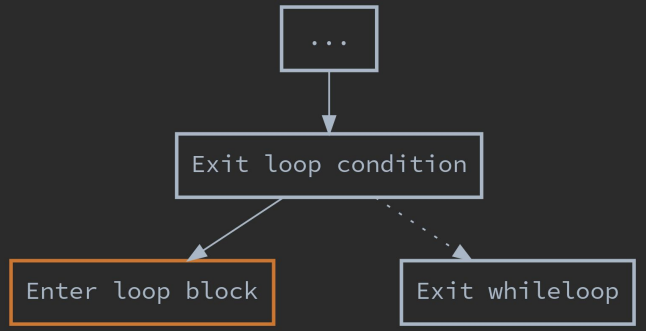
Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



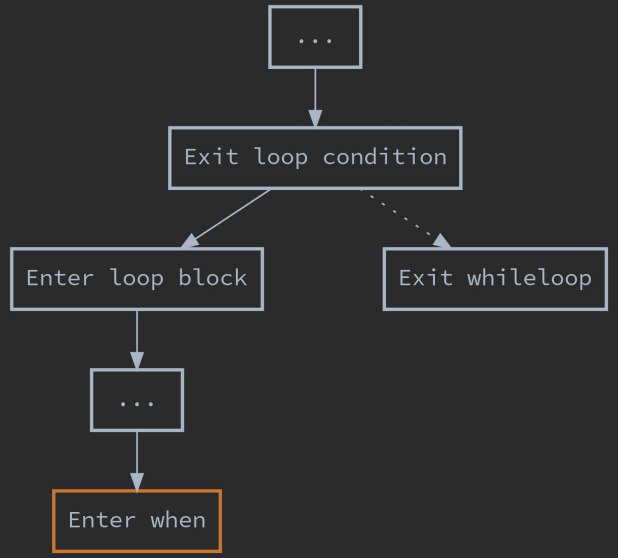
Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



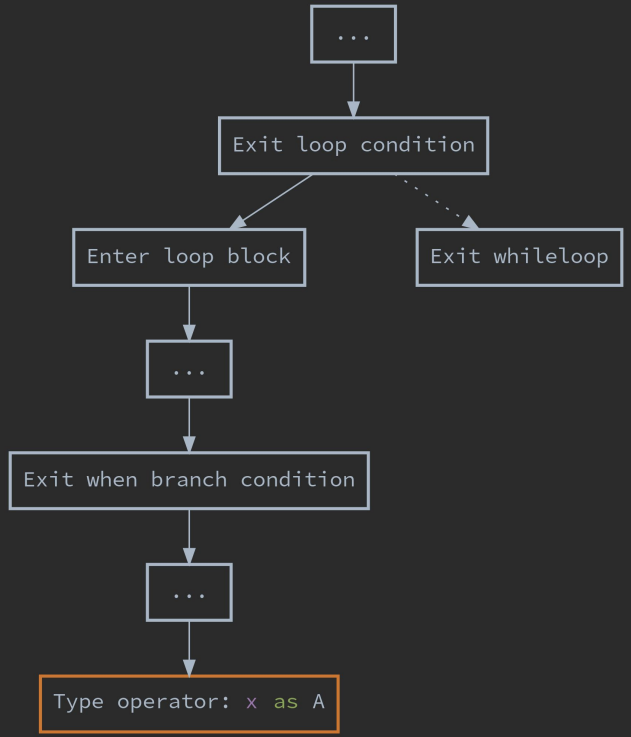
Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



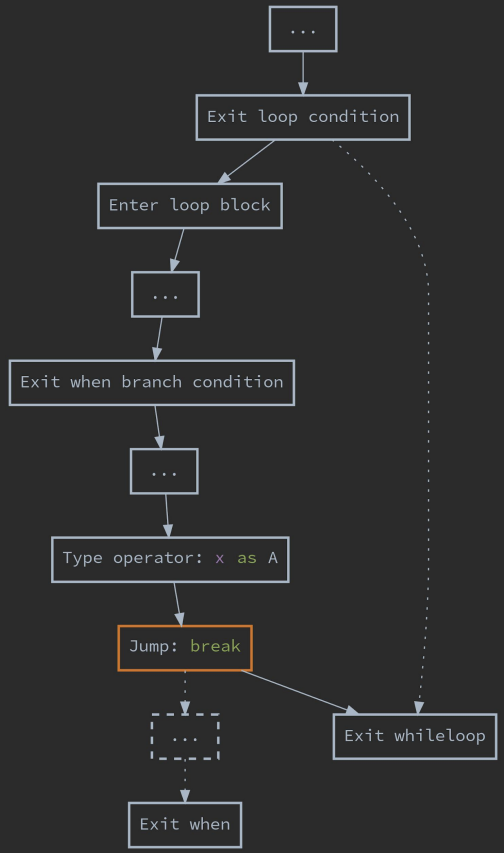
Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



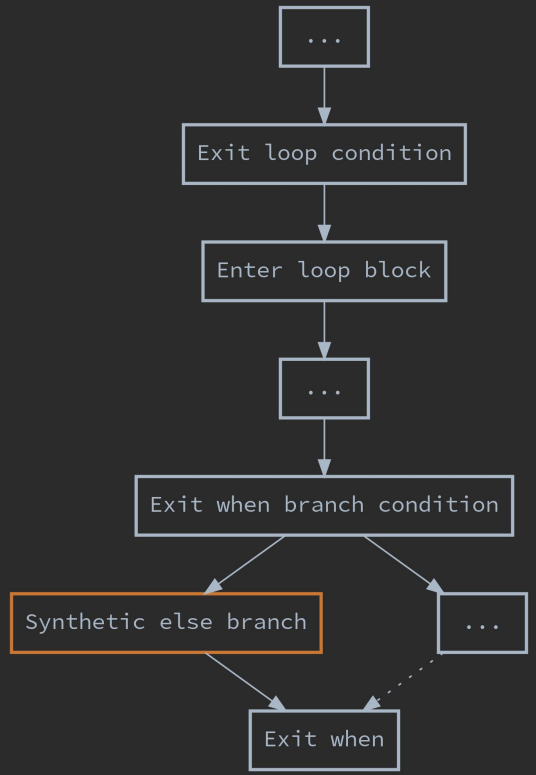
Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



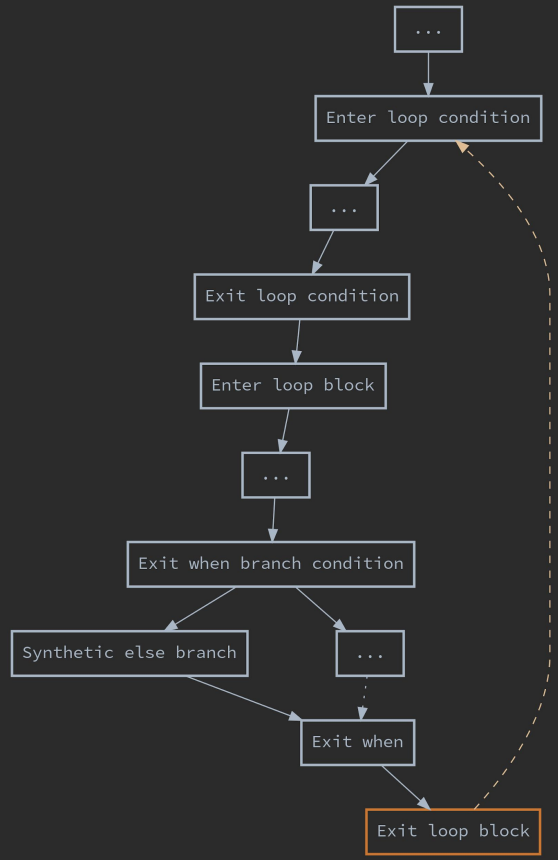
Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



Smarter-Casts and New Data-Flow Analysis

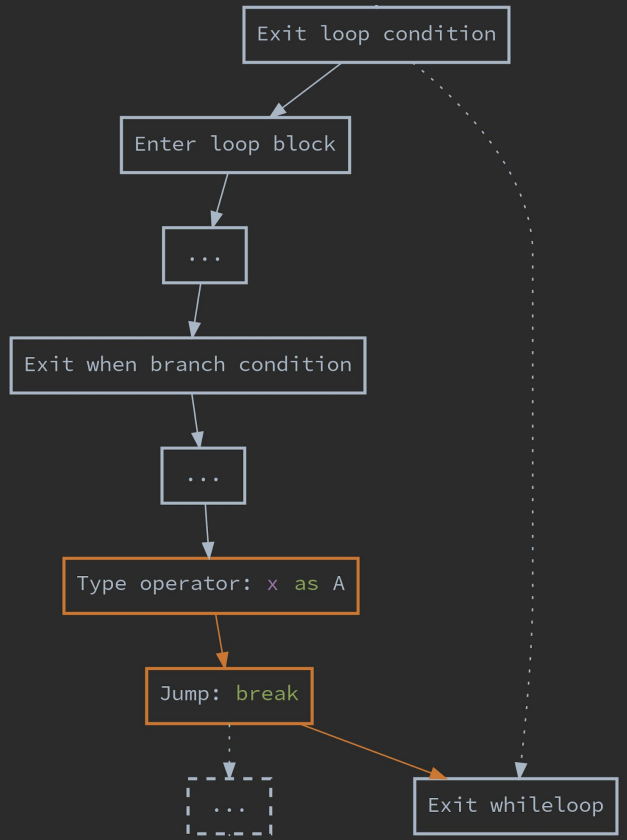
```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```



Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```

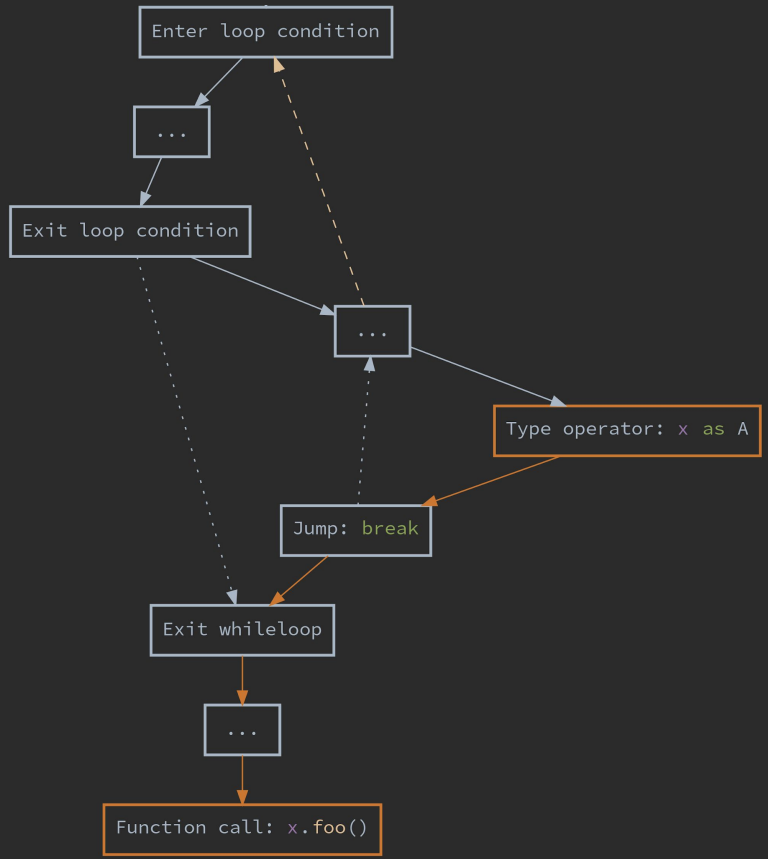
Flow: x is *A*



Smarter-Casts and New Data-Flow Analysis

```
fun bar(x: Any, b: Boolean) {
    while (true) {
        if (b) {
            x as A
            break
        }
    }
    x.foo()
}
```

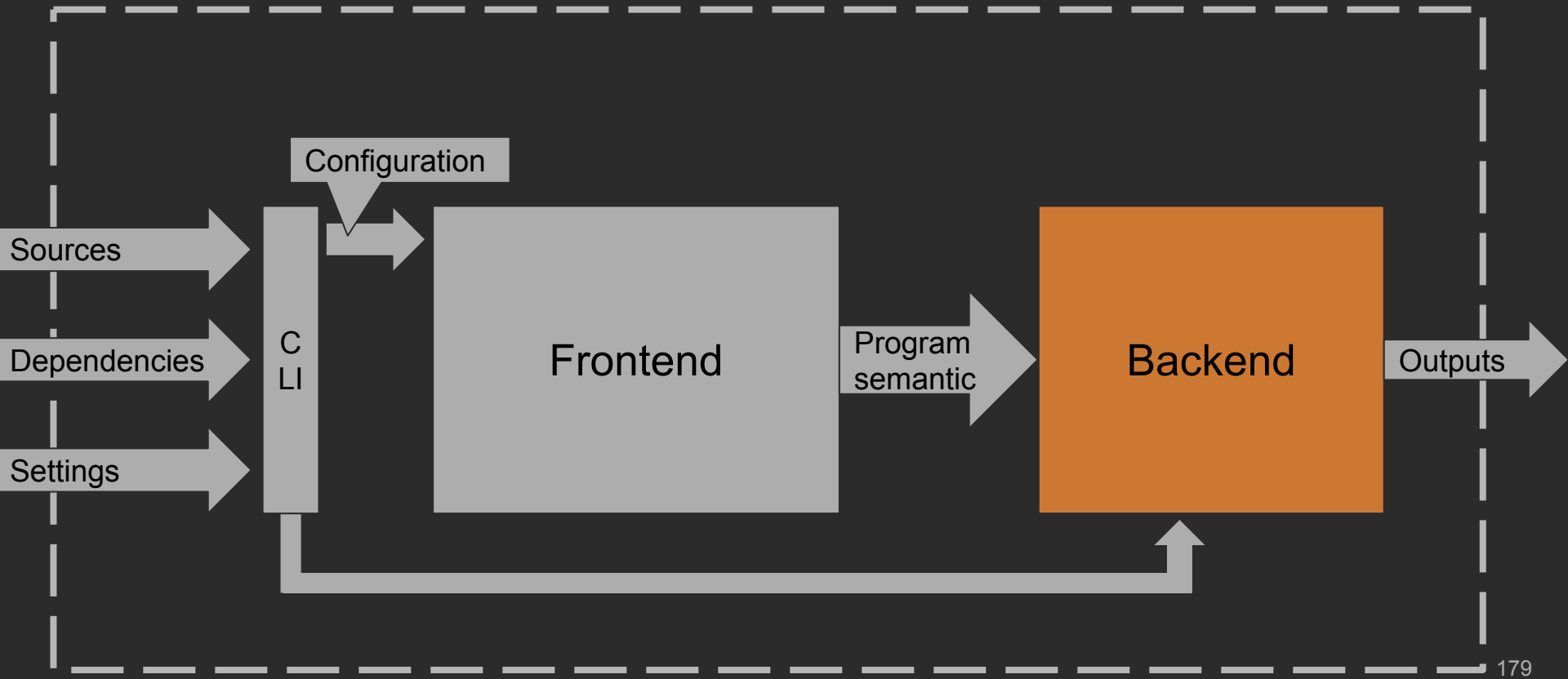
Flow: x is *A*



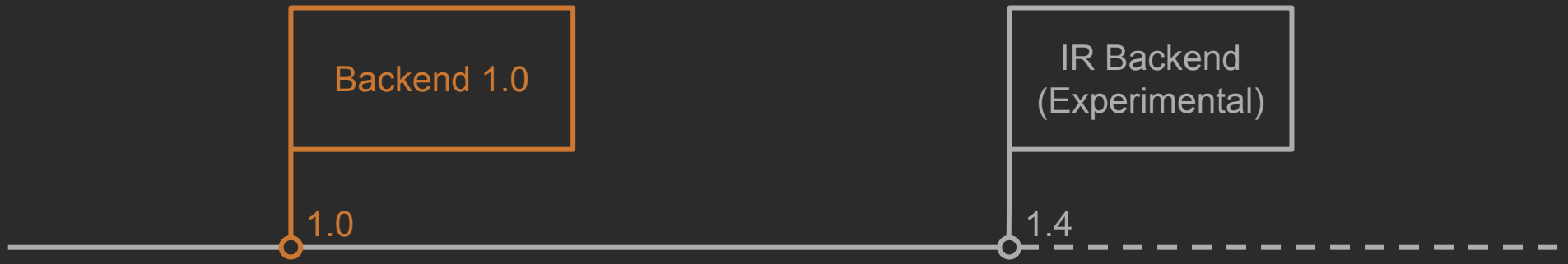
Recap: Frontend

- What’s a compiler frontend
- Kotlin Compiler Frontend 1.0 Architecture
- Type inference and changes to it in 1.4
- New Kotlin Compiler Frontend aka FIR major differences
- New data-flow, control-flow engine

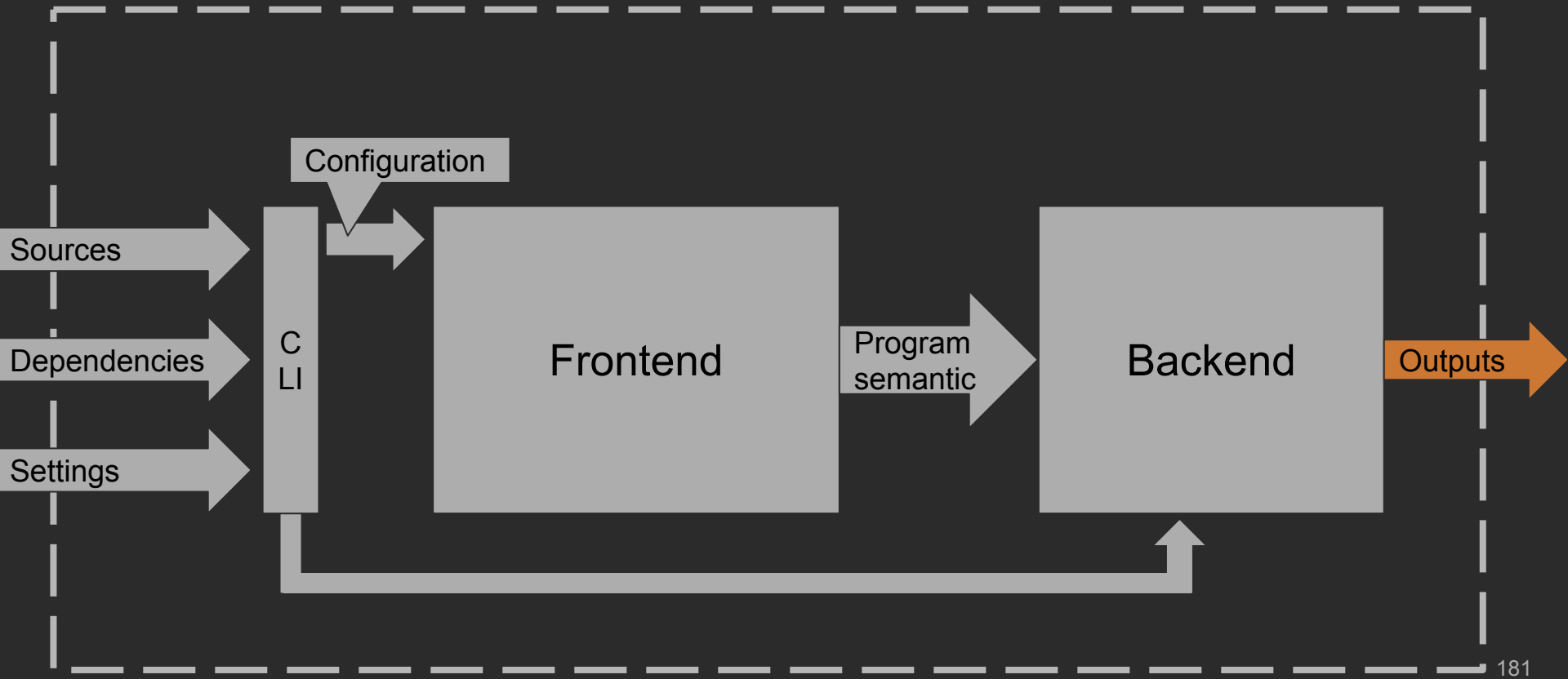
Compiler as a Transparent Box



Backend Timeline



Compiler as a Transparent Box



What Outputs? Let’s take a look on JVM

```
fun hello() = println("Hello, world!")
```

```
public final static hello()V
   L0
    LINENUMBER 1 L0
    LDC "Hello, World"
    ASTORE 0
   L1
    ICONST_0
    ISTORE 1
   L2
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    ALOAD 0
    INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/Object;)V
   L3
   L4
    LINENUMBER 1 L4
    RETURN
   L5
    MAXSTACK = 2
    MAXLOCALS = 2
```

What Outputs? Let’s take a look on JVM

```
fun hello() = println("Hello, world!")
```

```
v1 = "Hello, world!"
```

```
L0
    LINENUMBER 1 L0
    LDC "Hello, World"
    ASTORE 0 // v1 - local variable
```

What Outputs? Let’s take a look on JVM

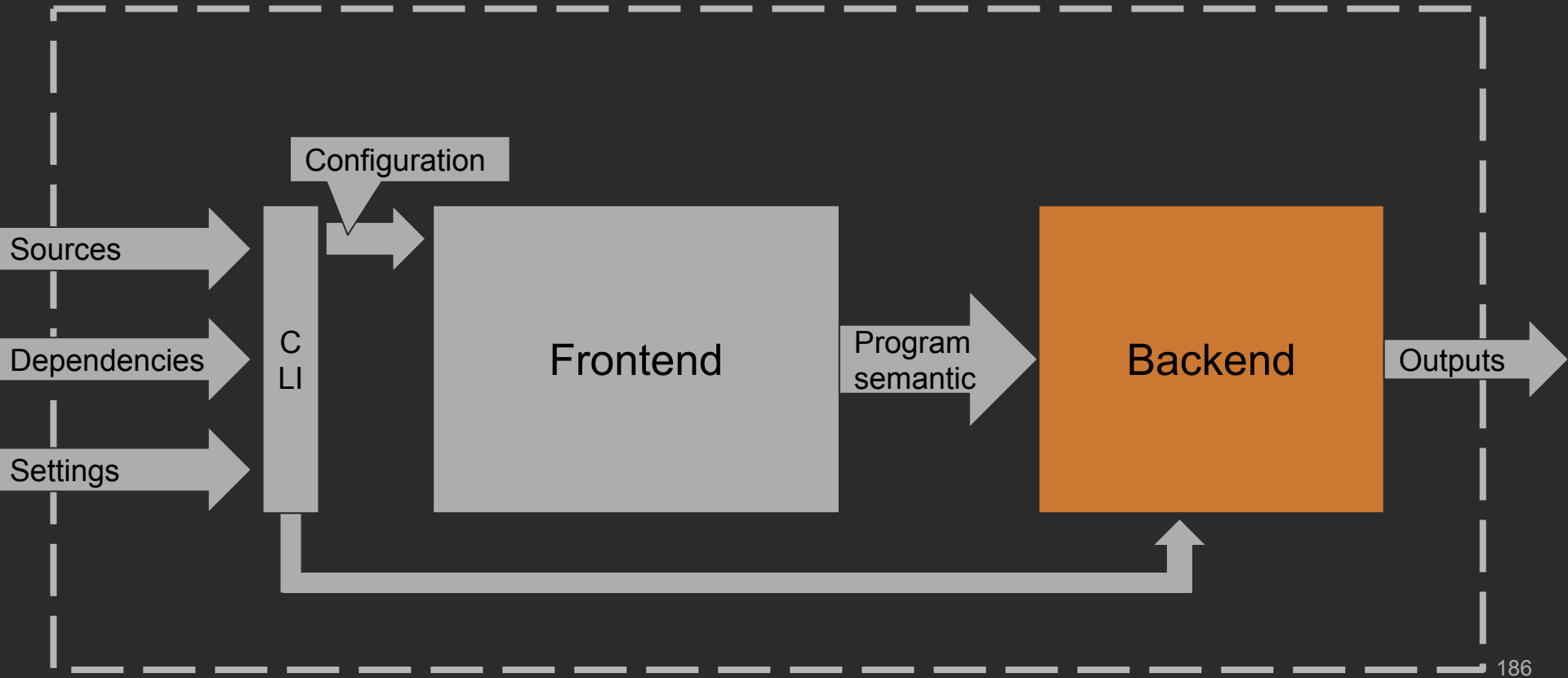
```
fun hello() = println("Hello, world!")

println(v1)

v2 = 0
System.out.println(v1)


L1
    ICONST_0
    ISTORE 1
L2
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    ALOAD 0 // v1 - local variable
    INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/Object;)V
```

Wait, what's that? Quiz

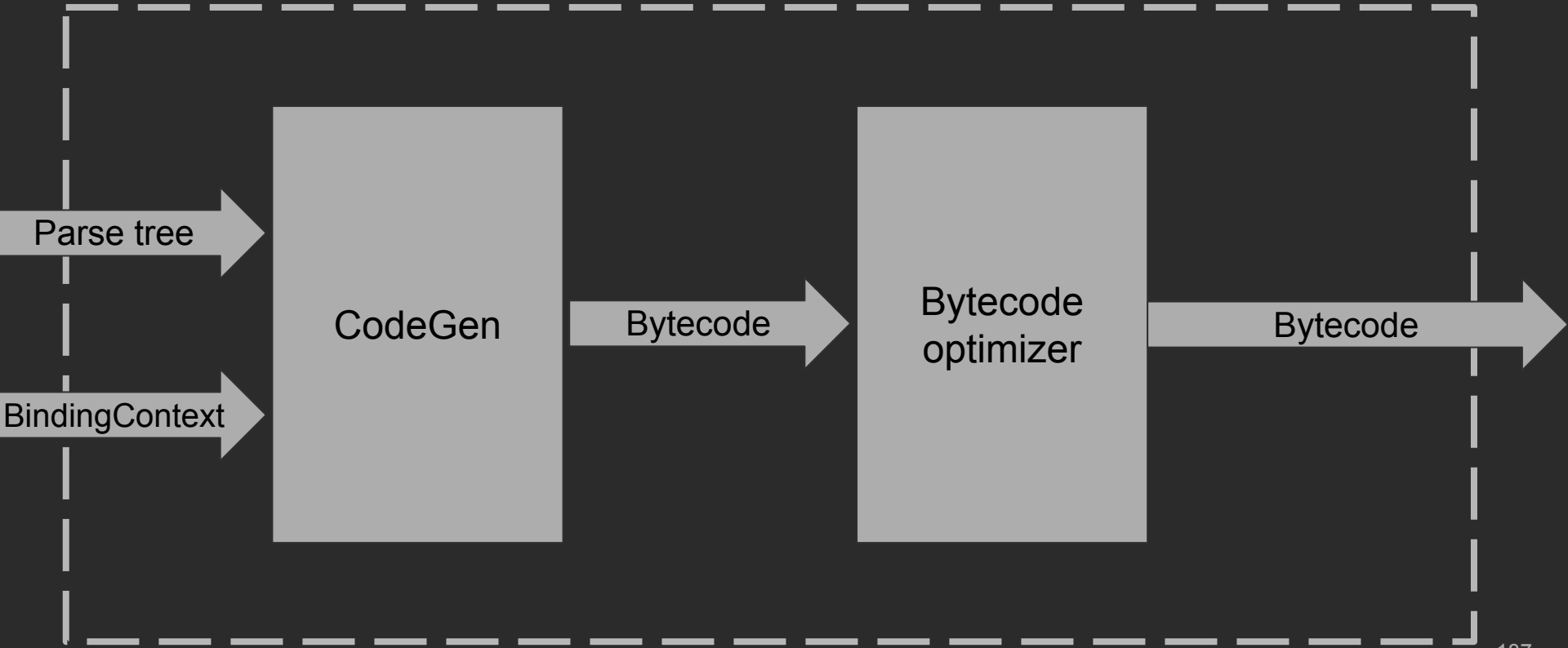
```
fun hello() = println("Hello, world!")
```

```
public final static hello()V
   L0
    LINENUMBER 1 L0
    LDC "Hello, World"
    ASTORE 0
   L1
    ICONST_0
    ISTORE 1
   L2
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    ALOAD 0
    INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/Object;)V
   L3
   L4
    LINENUMBER 1 L4
    RETURN
   L5
    ...
    LOCALVARIABLE $i$f$println I L2 L4 1
    ...
```

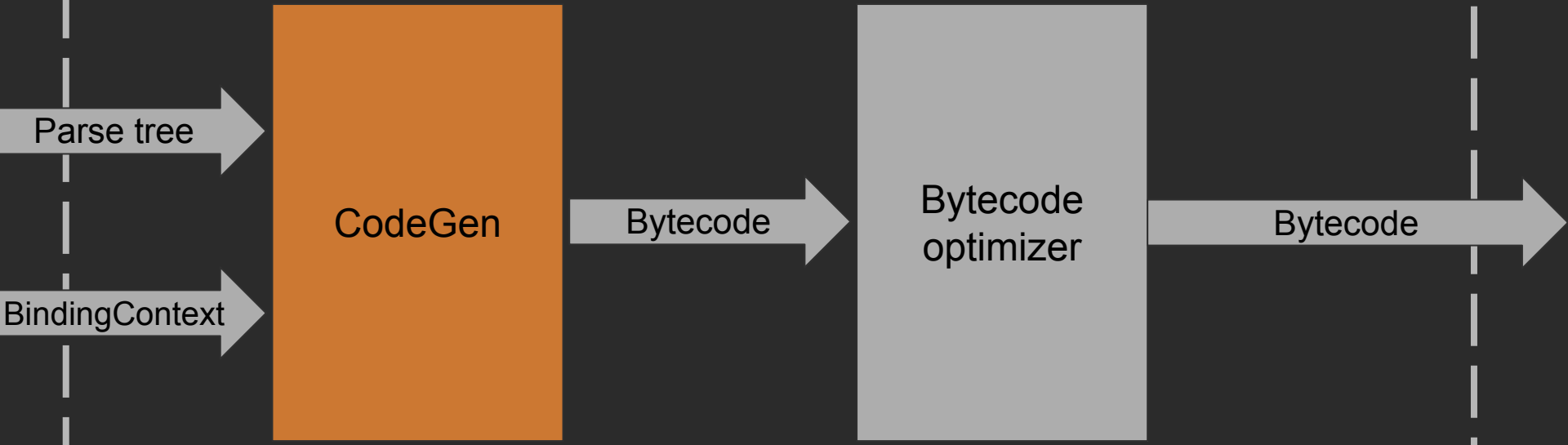
Compiler as a Transparent Box



JVM Backend 1.0



JVM Backend 1.0



JVM CodeGen

- Take PSI
- Take BindingContext
- Generate bytecode directly

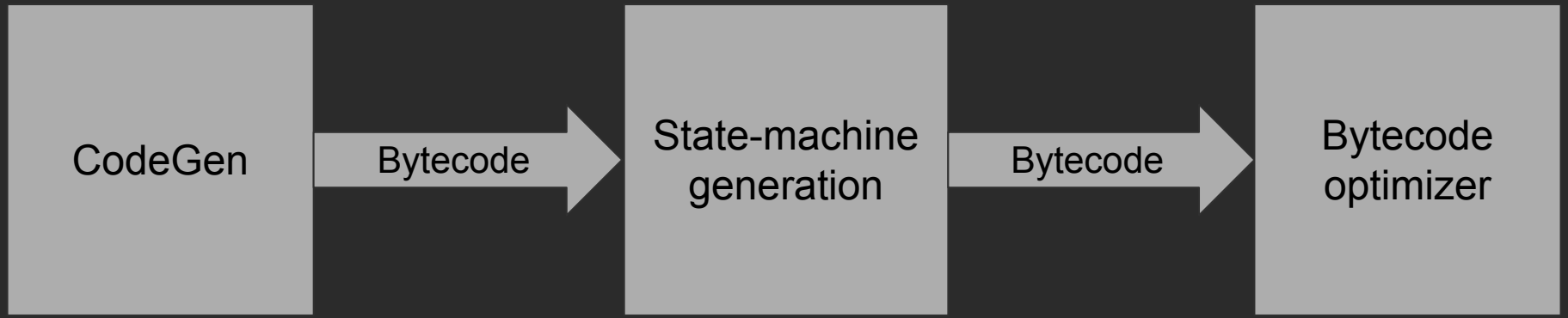
JVM CodeGen

```
fun printLines(lines: List<String>) {
   for (line in lines) {
       println(line)
   }
}
```

```
L1
  ALOAD 0
  INVOKEINTERFACE java/util/List.iterator ()Ljava/util/Iterator;
  ASTORE 2
L2
  ALOAD 2
  INVOKEINTERFACE java/util/Iterator.hasNext ()Z
  IFEQ L3
  ALOAD 2
  INVOKEINTERFACE java/util/Iterator.next ()Ljava/lang/Object;
  CHECKCAST java/lang/String
  ASTORE 1
...
L9
  GOTO L2
```

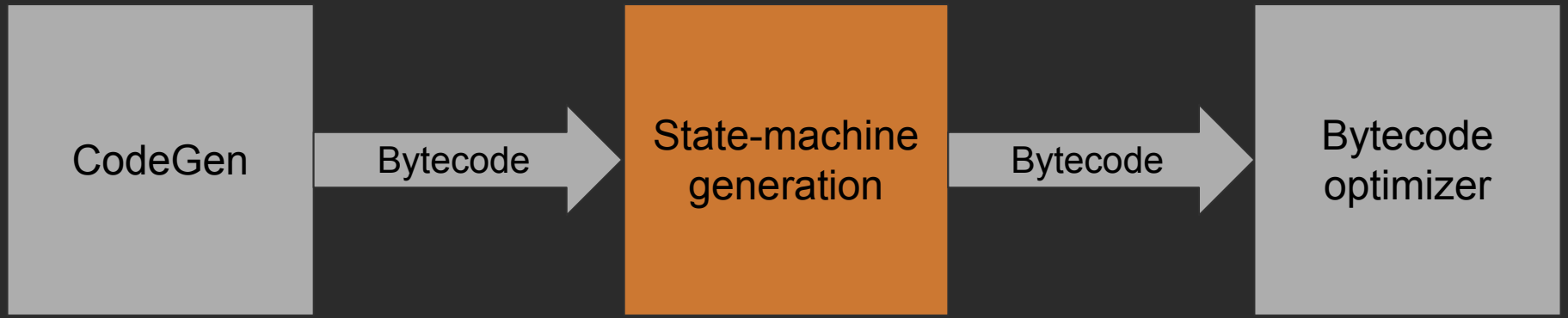
Suspend Function Transformation

- Suspend functions are state-machines
- It is hard to generate such complex bytecode in one pass



Suspend Function Transformation

- Suspend functions are state-machines
- It is hard to generate such complex bytecode in one pass



Bytecode Level Inlining

- Take function bytecode
- Merge into target
- Update local variables

Bytecode Level Inlining

```
fun hello(name: String) = println(name)
```

```
public final static hello(Ljava/lang/String;)V
   L0
    ...
   L2
    ICONST_0
    ISTORE 1
   L3
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    ALOAD 0
    INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/Object;)V
   L4
    NOP
   L5
    ...
   L6
    LOCALVARIABLE $i$f$println I L3 L5 1
    LOCALVARIABLE name Ljava/lang/String; L0 L6 0
```

Bytecode Level Inlining

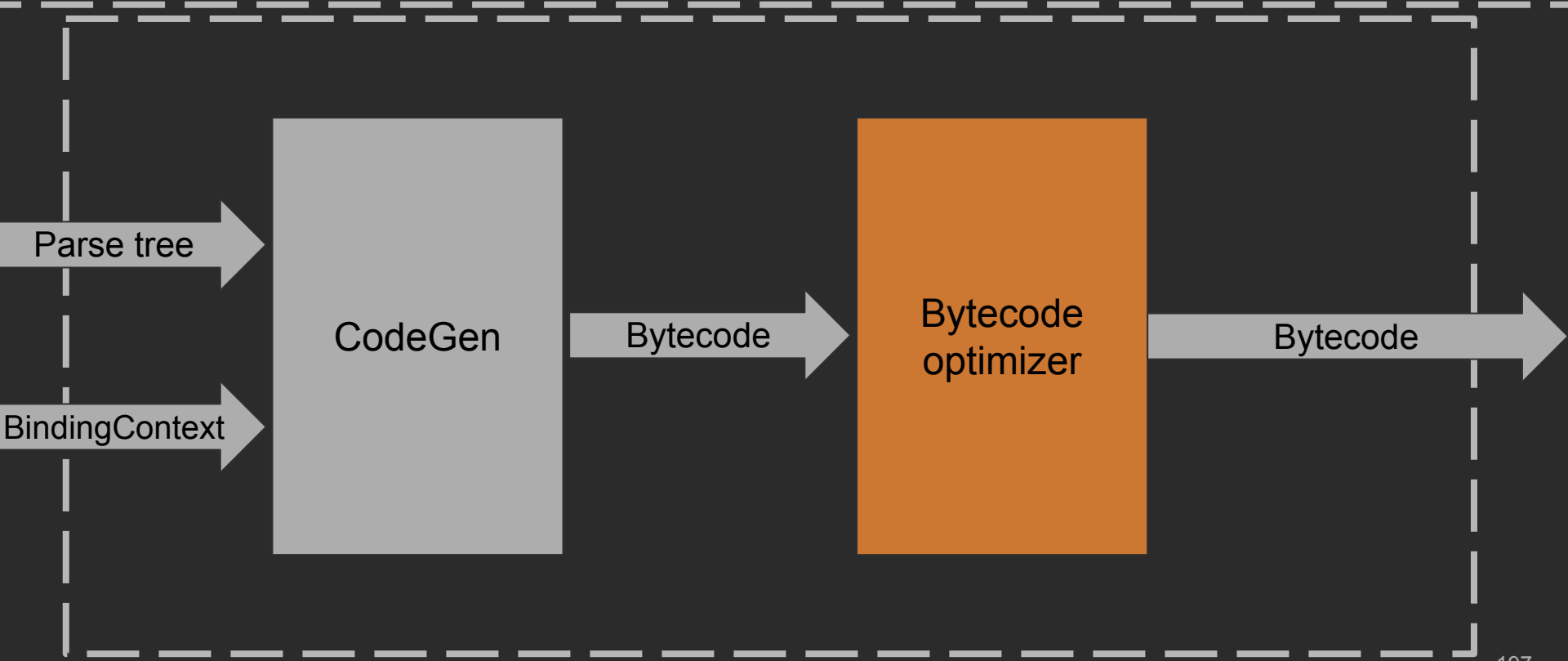
```
public inline fun println(message: Any?) {
    System.out.println(message)
}

public final static println(Ljava/lang/Object;)V
    L0
     LDC 0
     ISTORE 1
    L1
     GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
     ALOAD 0
     INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/Object;)V
    L2
     RETURN
    L3
     LOCALVARIABLE message Ljava/lang/Object; L0 L3 0
     LOCALVARIABLE $i$f$println I L1 L3 1
```

Bytecode Level Inlining

```
fun hello(name: String) = println(name)
```

```
public final static hello(Ljava/lang/String;)V
   L0
    ...
   L2
    ICONST_0
    ISTORE 1
   L3
    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
    ALOAD 0
    INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/Object;)V
   L4
    NOP
   L5
    ...
   L6
    LOCALVARIABLE $i$f$println I L3 L5 1
    LOCALVARIABLE name Ljava/lang/String; L0 L6 0
```

JVM Backend 1.0

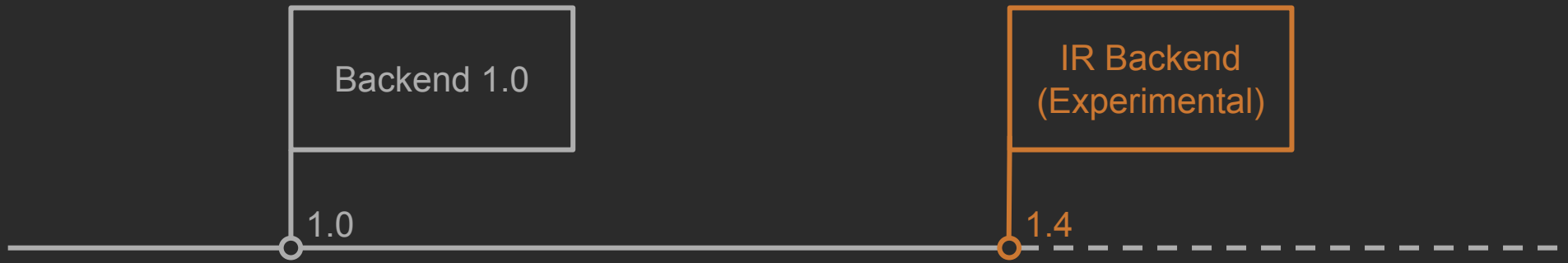


But what about JS and Native?

- JS Backend would like to reuse existing JVM Backend components

- JS text have nothing to do with JVM bytecode
- Kotlin/Native also have nothing to do with JVM bytecode

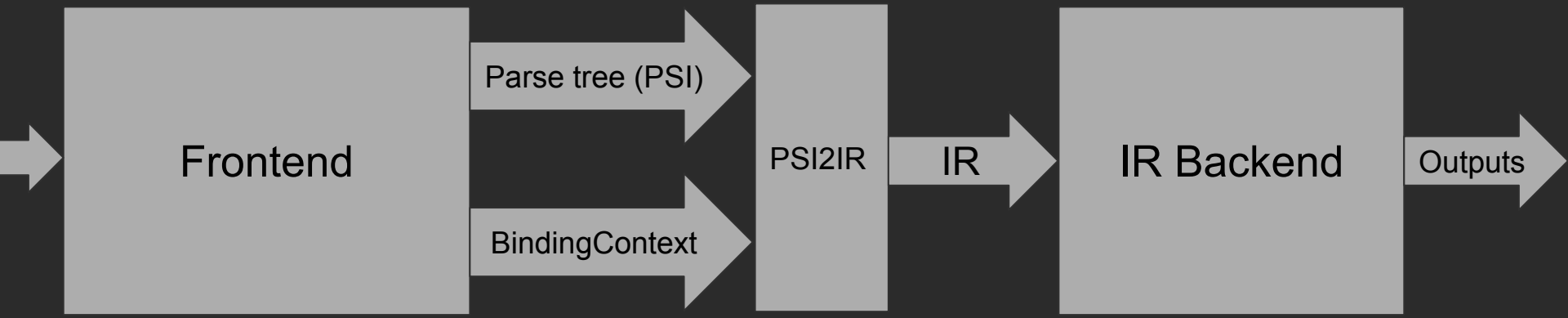
Backend Timeline



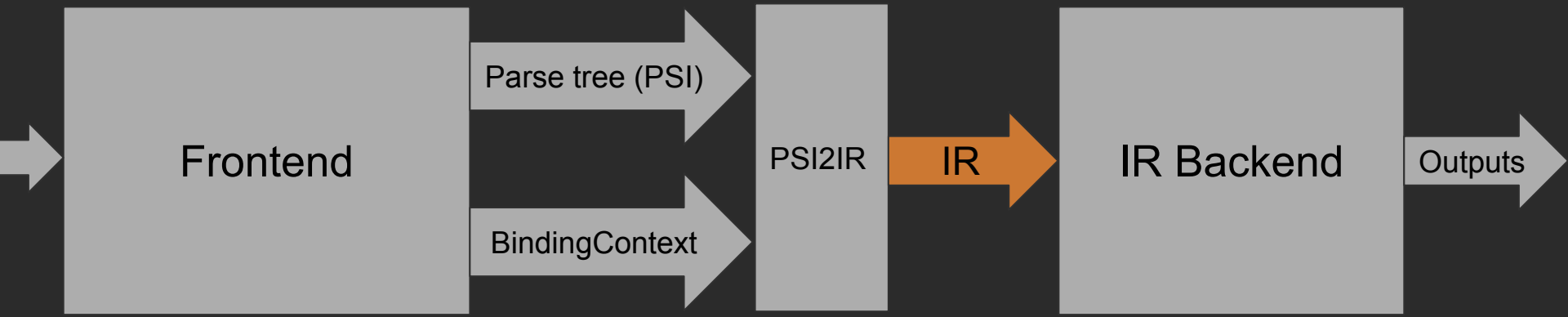
New Backend Goals

- Avoid logic duplication in different backends
- Share high-level optimizations across different backends
- Multi-step transformations suits better for complex constructs
- Pluggability

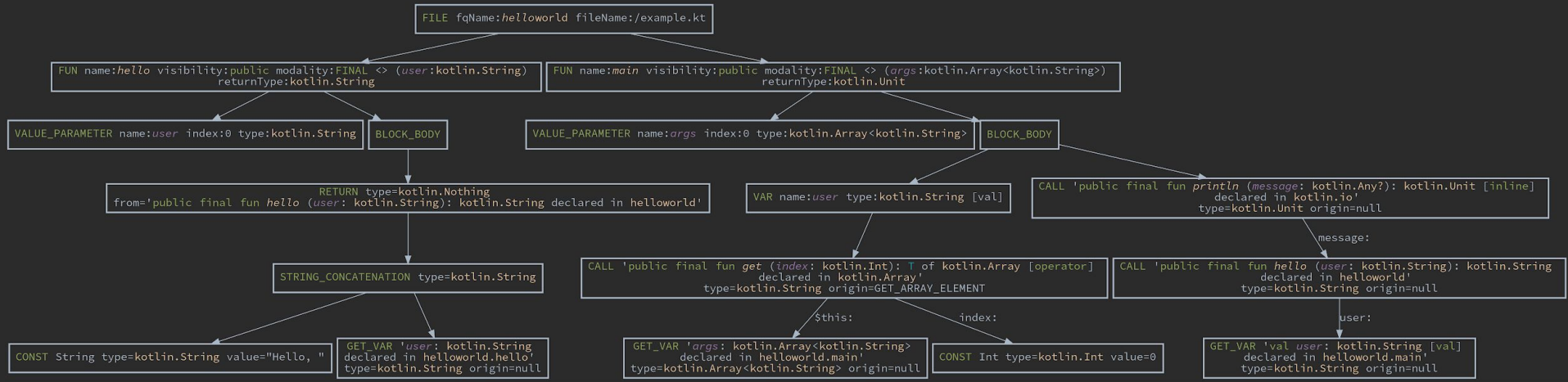
Kotlin Compiler with IR Backend



Kotlin Compiler with IR Backend

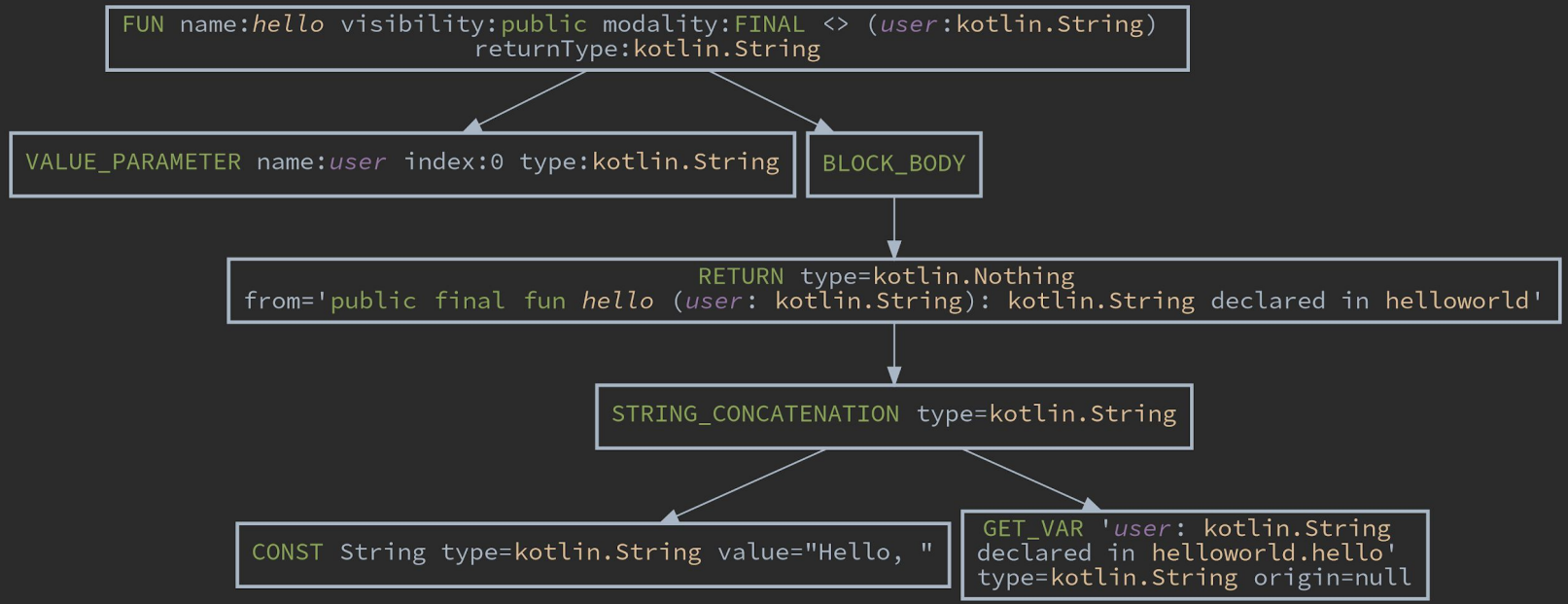


IR? Intermediate Representation! Tree again

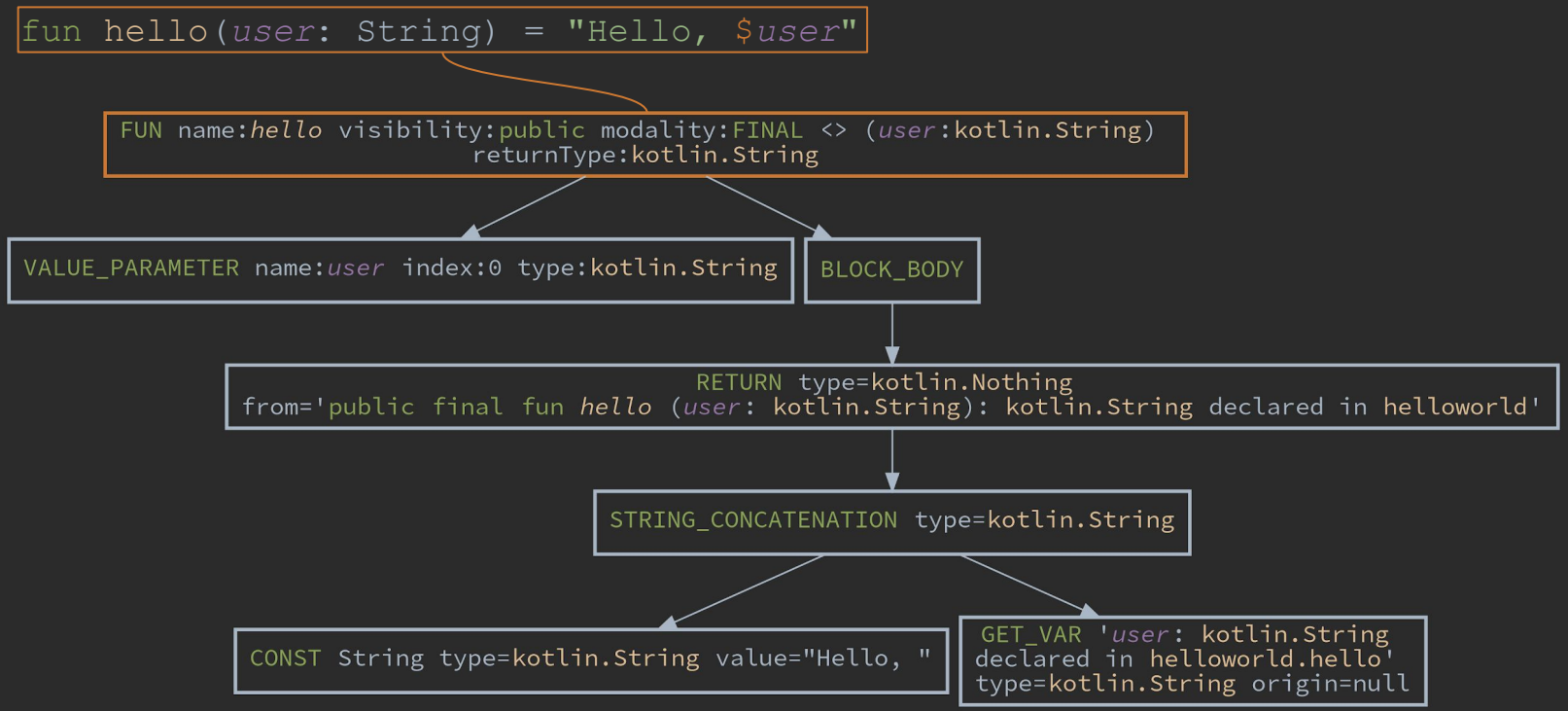


Backend Intermediate Representation: Closer look

```
fun hello(user: String) = "Hello, $user"
```

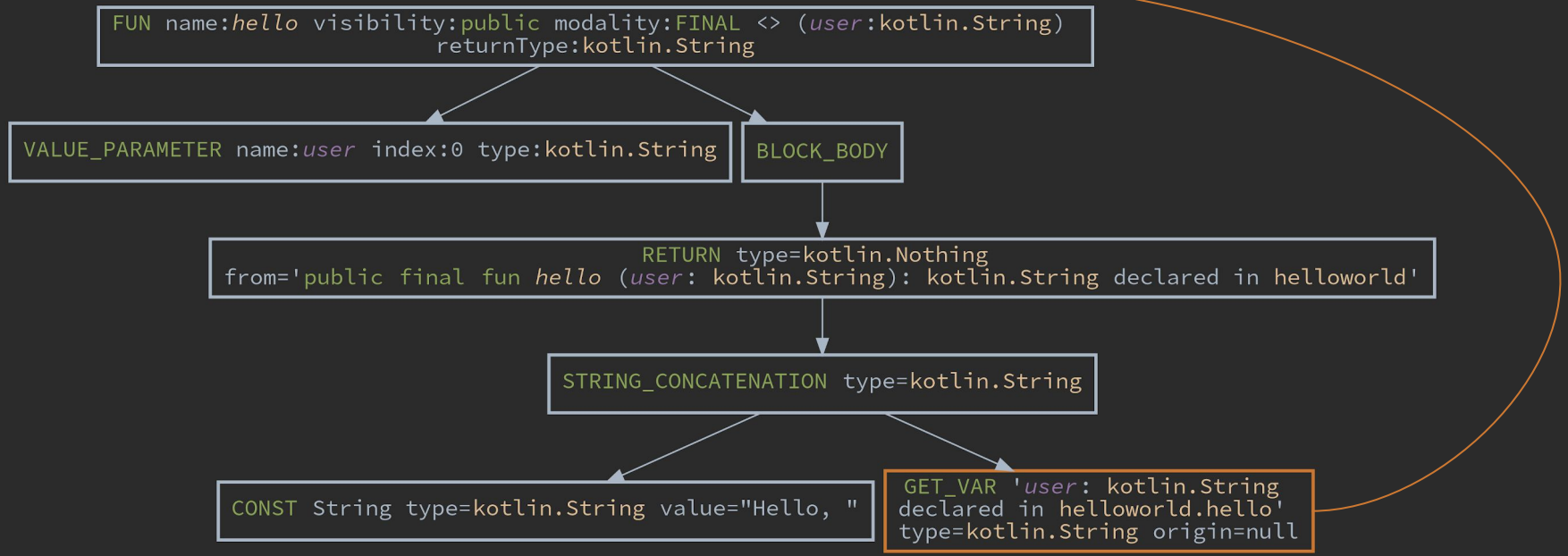


Backend Intermediate Representation: Closer look

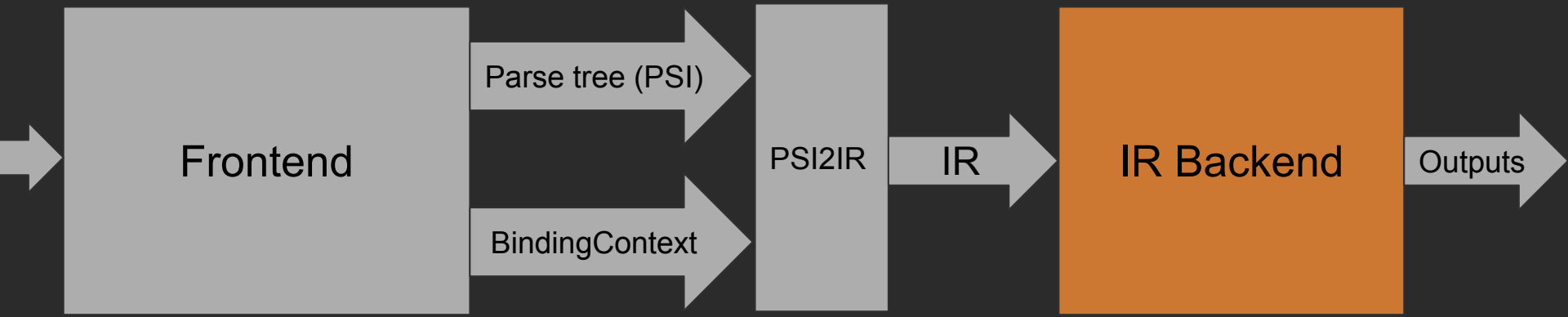


Backend Intermediate Representation: Closer look

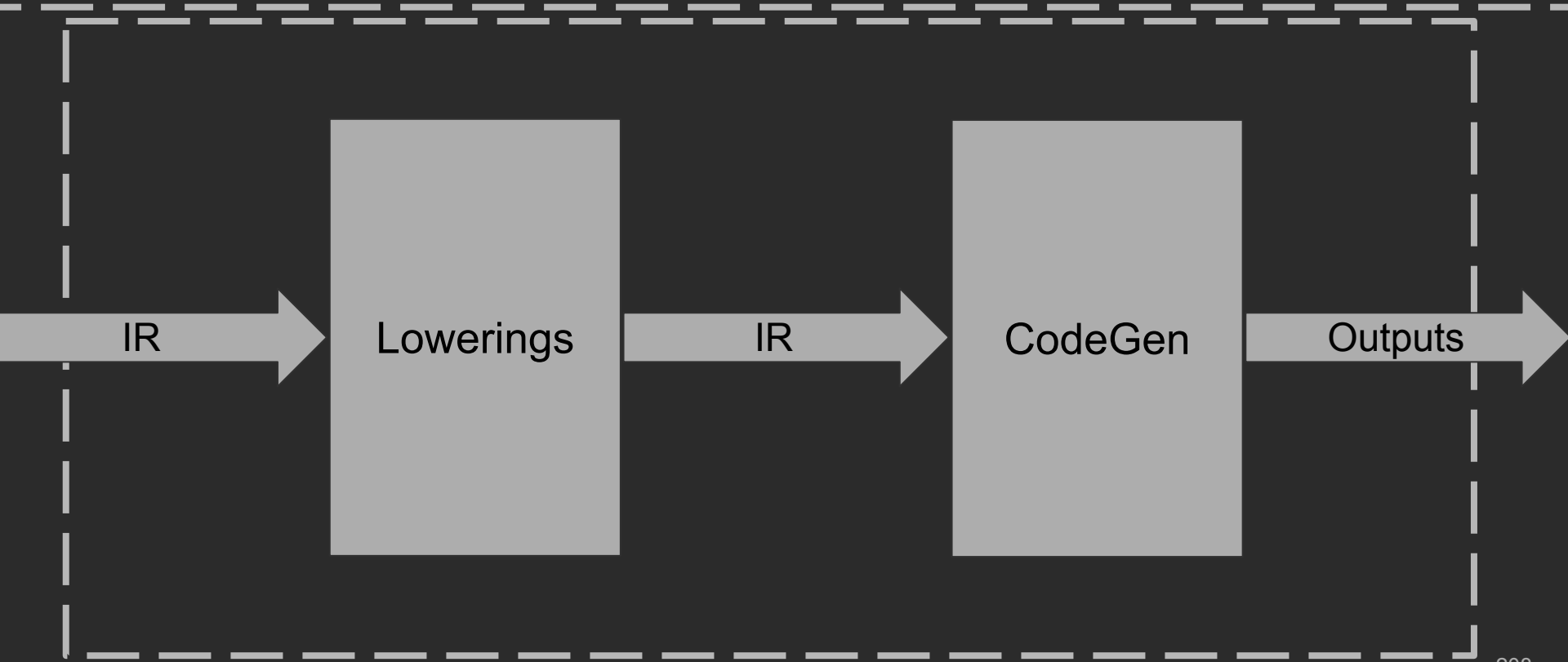
```
fun hello(user: String) = "Hello, $user"
```



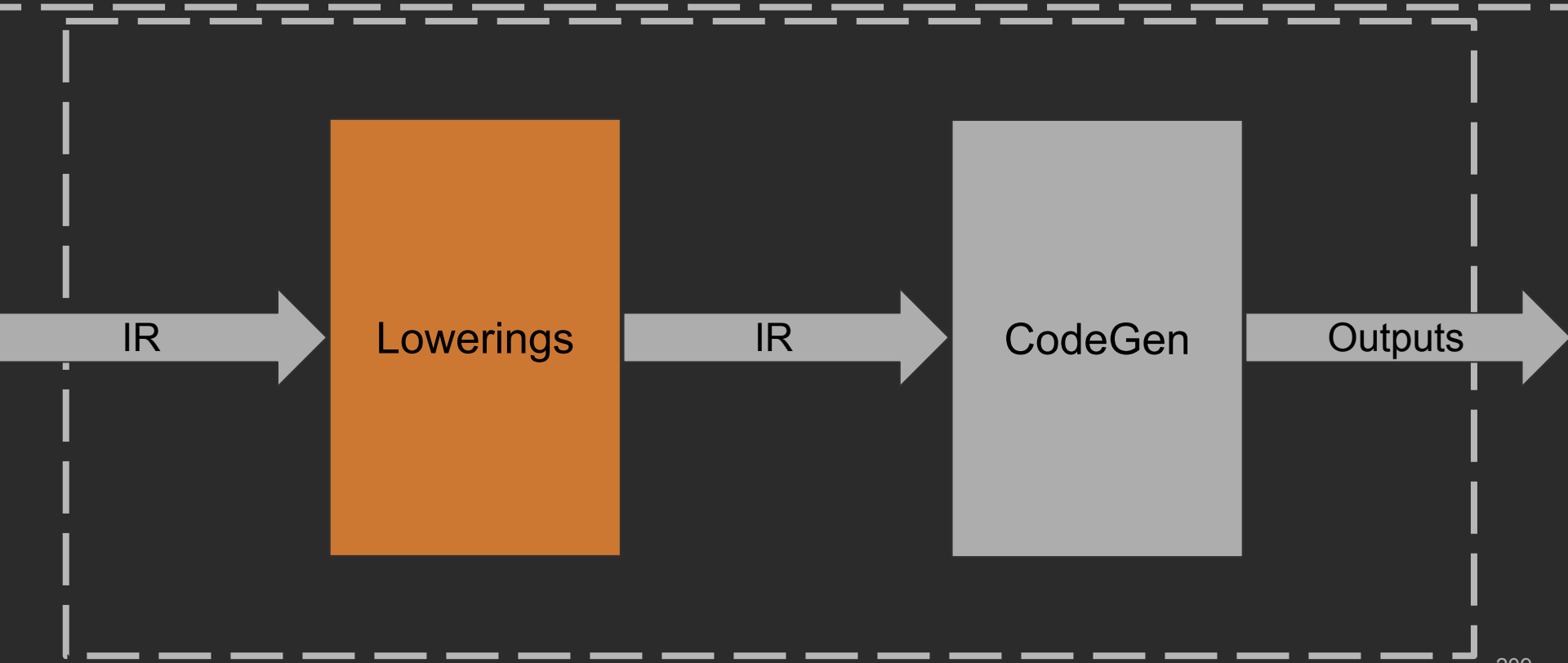
Kotlin Compiler with IR Backend



IR Backend. Pipeline



IR Backend. Pipeline



IR Backend. Lowerings

```
fun interface Action {
   fun perform()
}

fun getHelloAction() = Action { println("Hello!") }
```

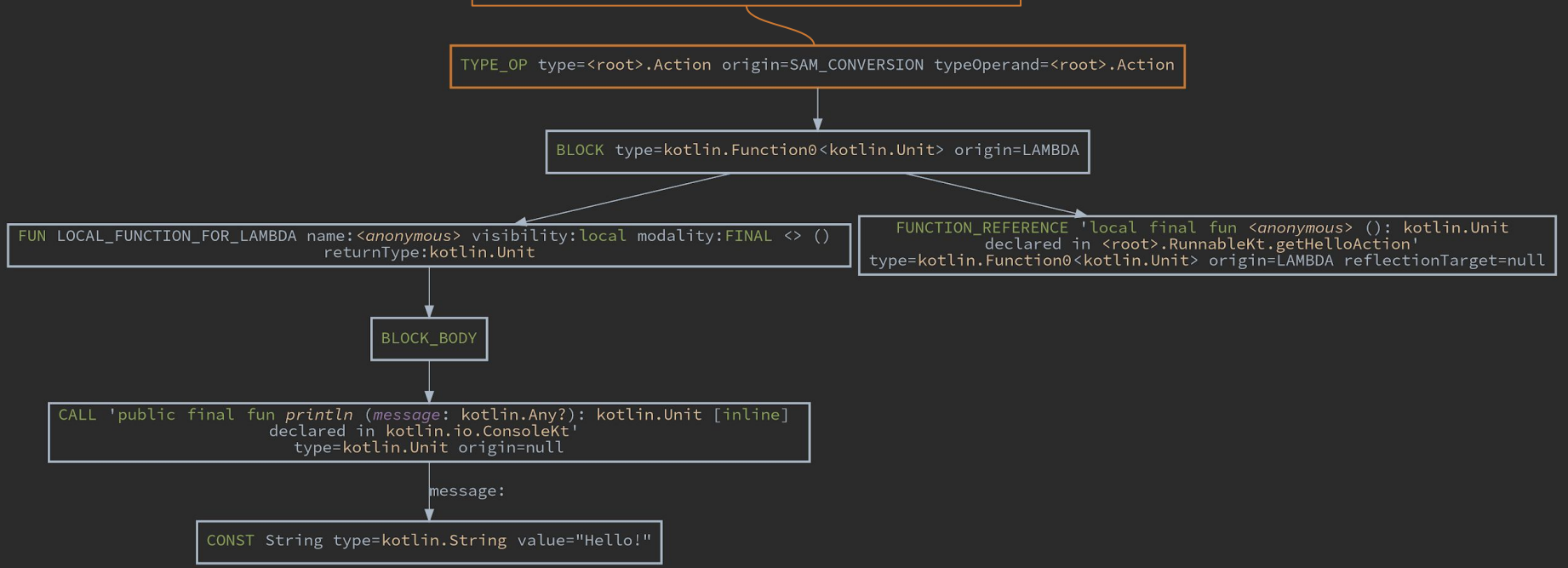
IR Backend. Lowerings

```
fun interface Action {
    fun perform()
}

fun getHelloAction() = Action { println("Hello!") }
```

```
fun getHelloAction() = object : Action {
    override fun perform() {
        println("Hello!")
    }
}
```

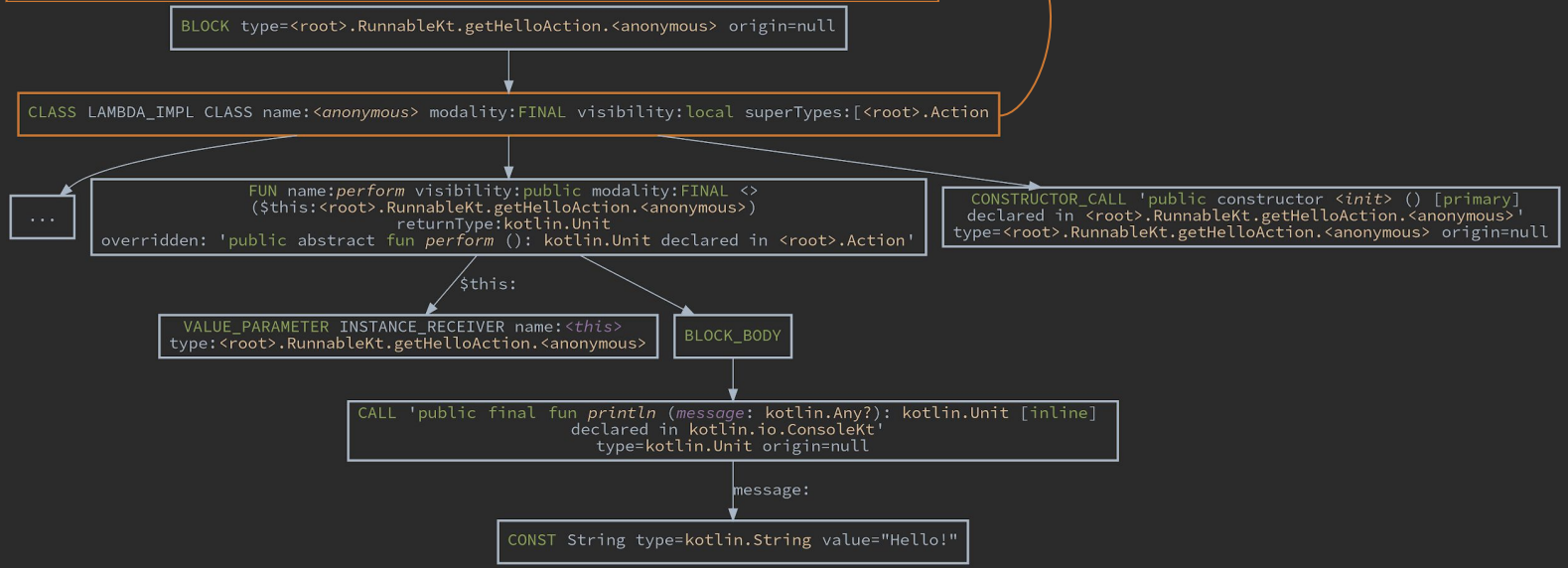
IR Backend. Lowerings

```
fun getHelloAction() = Action { println("Hello!") }
```



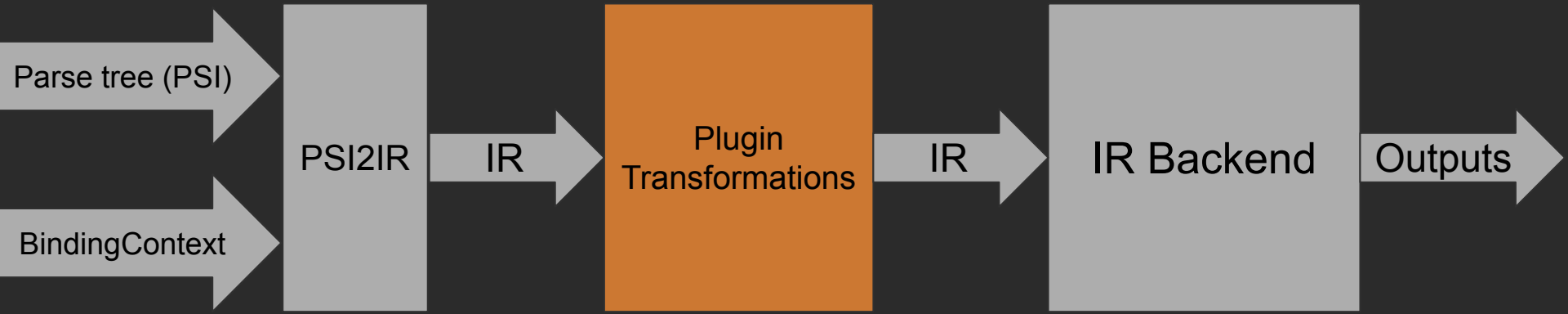
IR Backend. Lowerings

```
fun getHelloAction() = object : Action {
    override fun perform() {
        println("Hello!")
    }
}
```



NOTE: Experimental

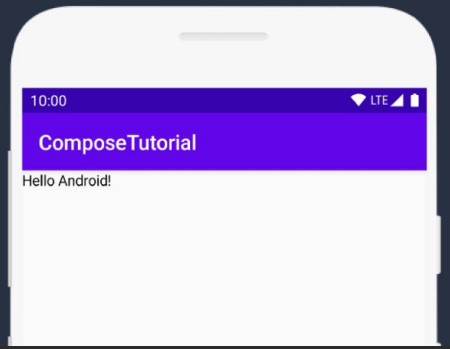
IR Backend. Pluggability



NOTE: Experimental

IR Backend. Pluggability: Jetpack Compose

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            Greeting("Android")
        }
    }
}

@Composable
fun Greeting(name: String) {
    Text (text = "Hello $name!")
}
```



NOTE: Experimental

IR Backend. Pluggability: Jetpack Compose

```
@Composable fun A(x: Int) {
 f(x)
}
```

NOTE: Experimental

IR Backend. Pluggability: Jetpack Compose

```
@Composable fun A(x: Int) {
 f(x)
}
```

```
@Composable fun A(x: Int, $composer: Composer<*>, $changed: Int) {
   $composer.startRestartGroup()
   f(x)
   $composer.endRestartGroup()?.updateScope { next -> A(x, next, $changed or 0b1) }
}
```

KotlinConf 2019: The Compose Runtime, Demystified: https://youtu.be/6BRII5zfCCk

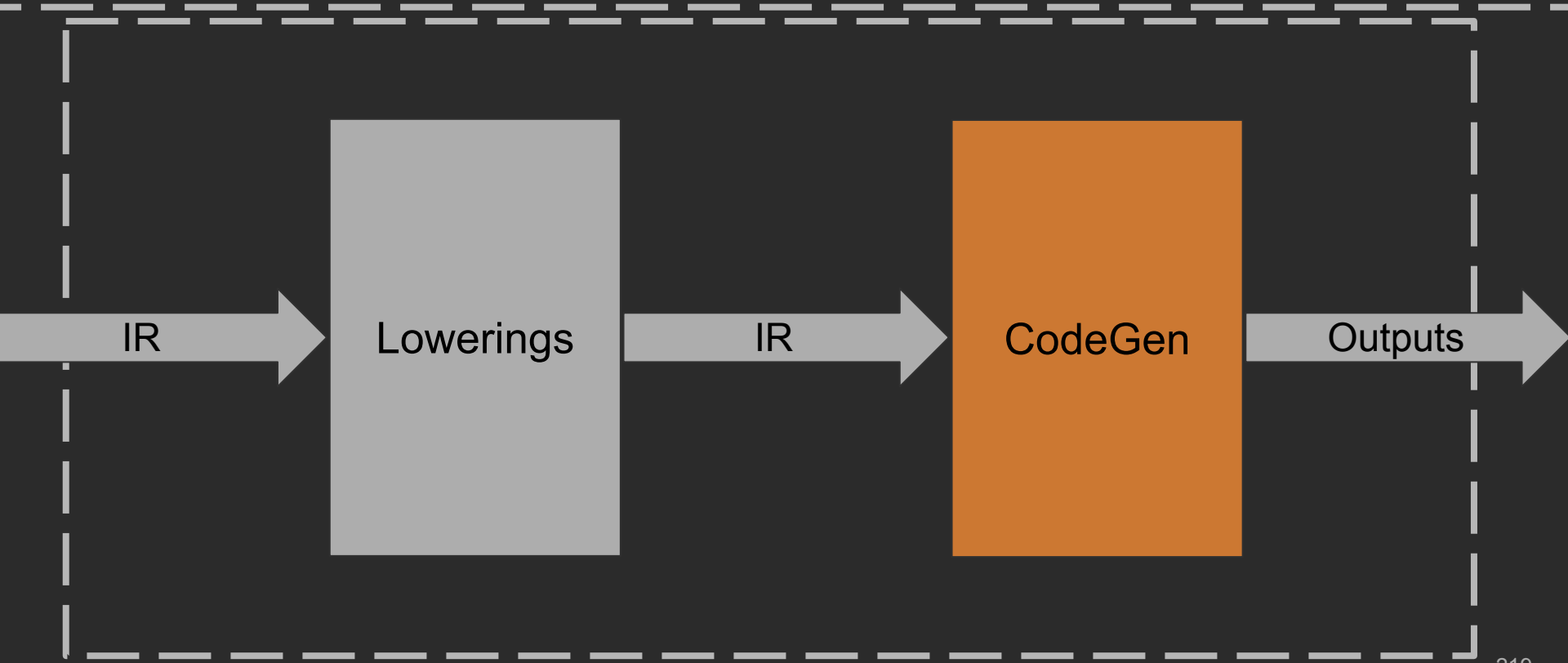
NOTE: Experimental

IR Backend. Pluggability: Power-Assert

```
assert(jane != null && jane.firstName == "Jane")

# Example failure
java.lang.AssertionError: Assertion failed
assert(jane != null && jane.firstName == "Jane")
       |    |
       |    false
       null
```

https://medium.com/@bnorm/soft-assertion-with-kotlin-power-assert-4b61fa763b61

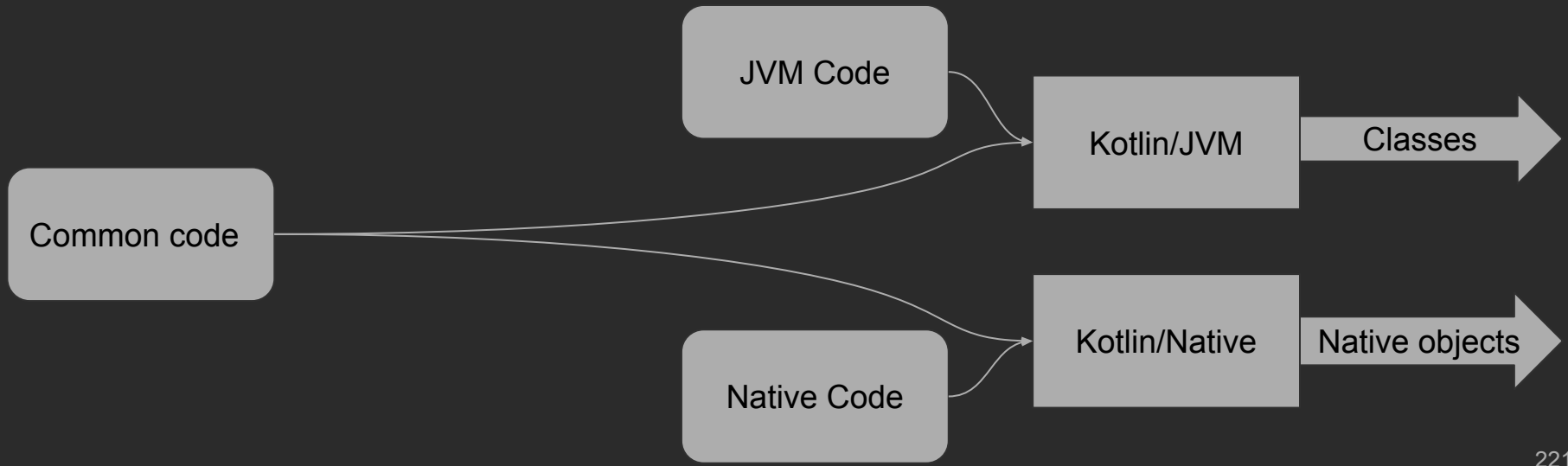
IR Backend. Pipeline



IR Backend. CodeGen

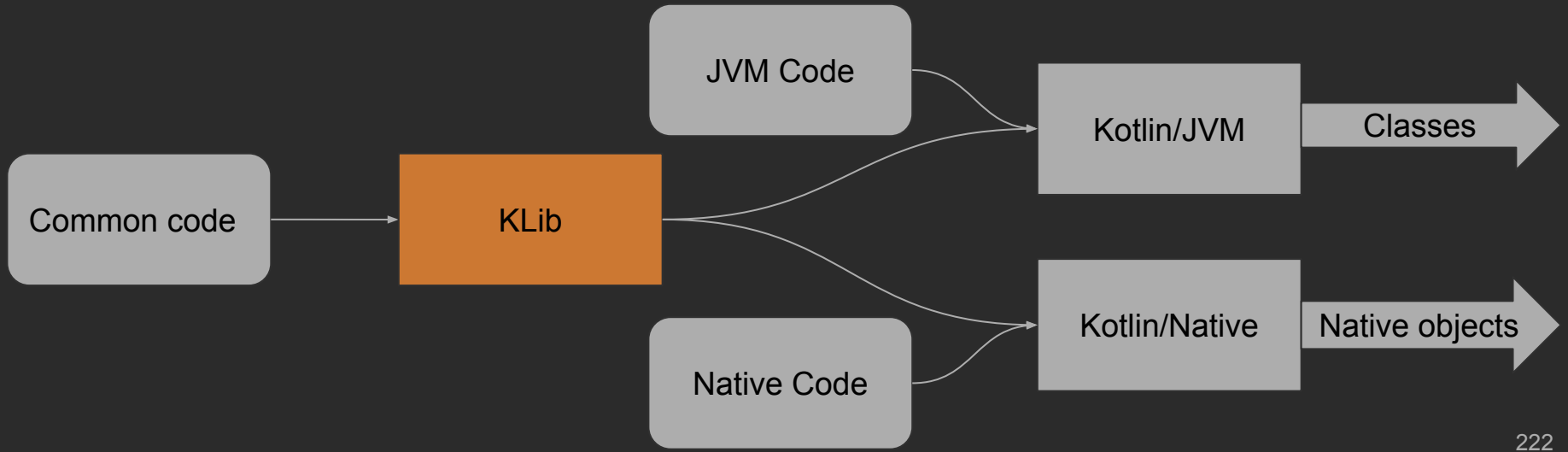
- Take final IR
- Generate bytecode one to one

IR. MPP

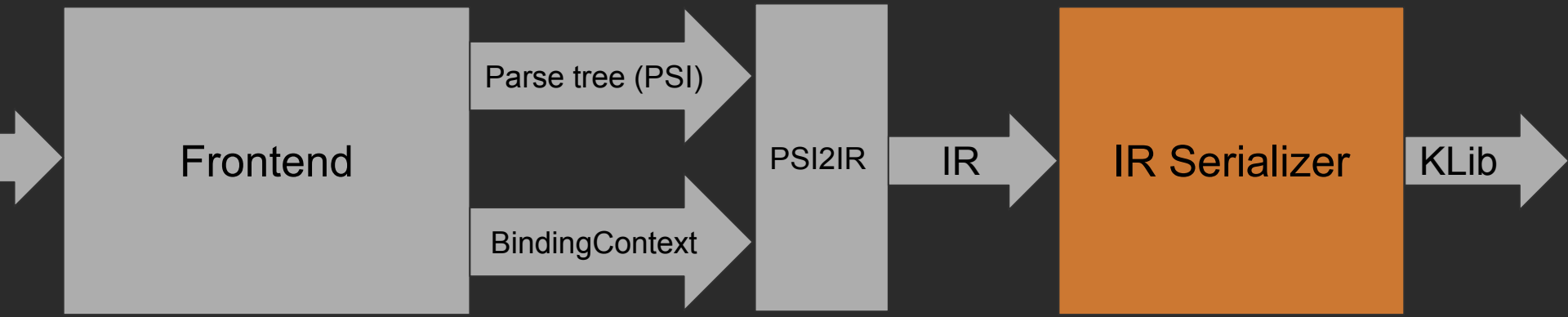

JVM Code
Common code
Native Code
Kotlin/JVM
Kotlin/Native
Classes
Native objects

IR. MPP

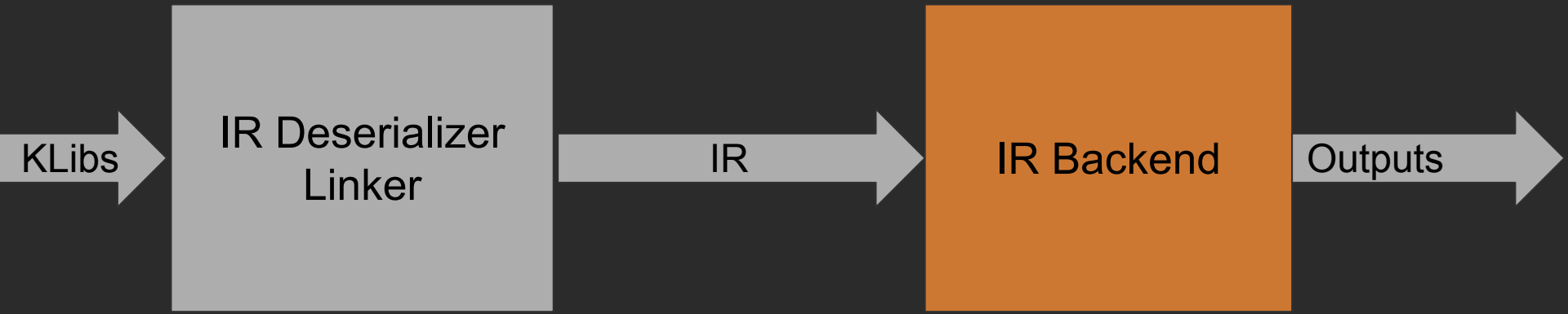
- Compile common modules to KLib
- Link platform modules
- Run target backend to produce final artifacts



IR Backend. KLib



IR Backend. KLib



Summary

- Kotlin Compiler Architecture
- Why and how Kotlin Compiler changes
- Not only language evolving, but also compiler behind it

Questions?

Kotlin Compiler in past, 1.4 and beyond

- Simon Ogorodnik

- Simon.Ogorodnik@jetbrains.com / Simon.Ogorodnik@gmail.com
- semoro @ kotlinlang Slack