Viewing Note

This presentation is designed with animations and is best experienced in PowerPoint. This PDF version is provided for your convenience for printing and sharing

JOHANNES KEPLER
UNIVERSITÄT LINZ

JOHANNES KEPLER
UNIVERSITÄT LINZ

Simplifying Kotlin Compile-Time Code Instrumentation

Lorenz Bader
Markus Weninger, Institute for System Software

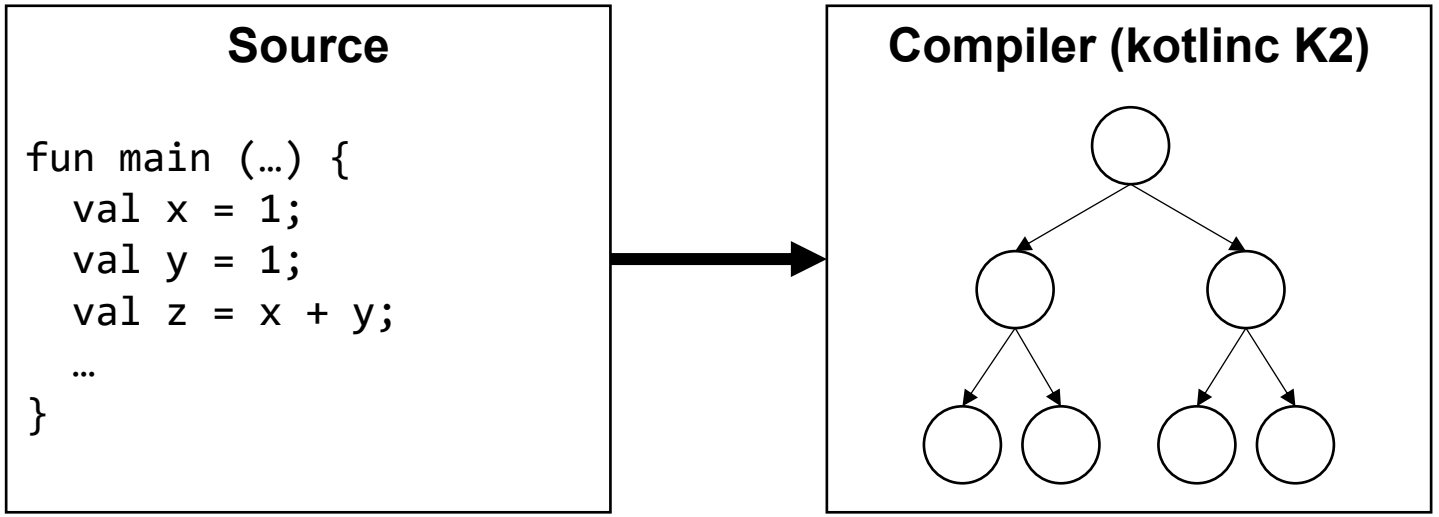
JOHANNES KEPLER
UNIVERSITÄT LINZ
Altenberger Straße 69
4040 Linz, Österreich
jku.at

Introduction

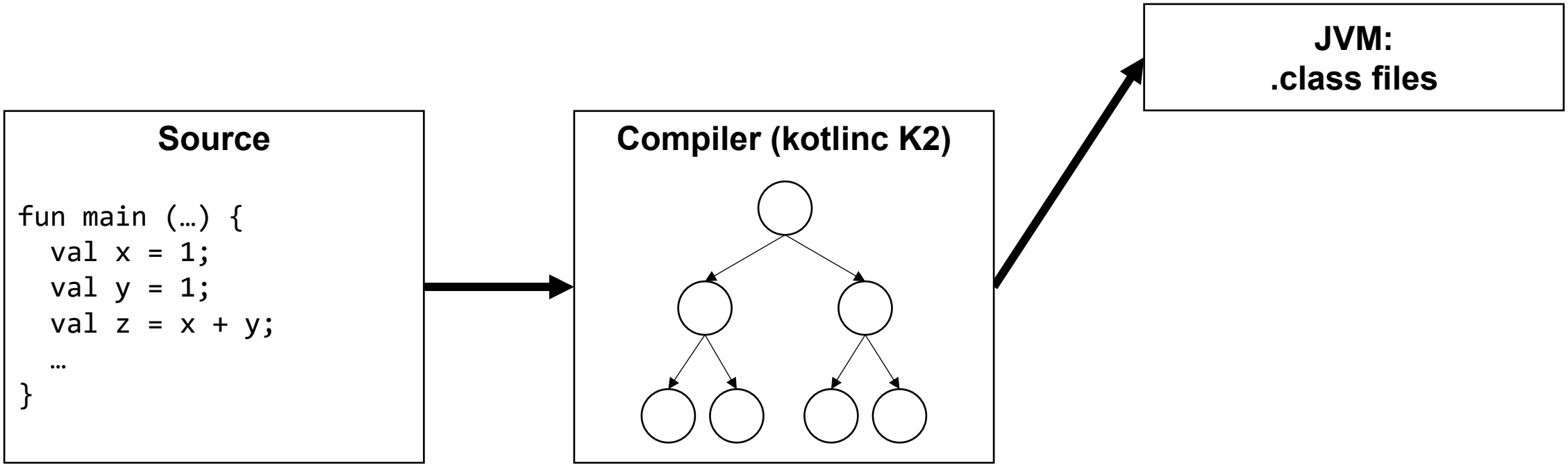
Source

```
fun main (…) {
  val x = 1;
  val y = 1;
  val z = x + y;
  …
}
```

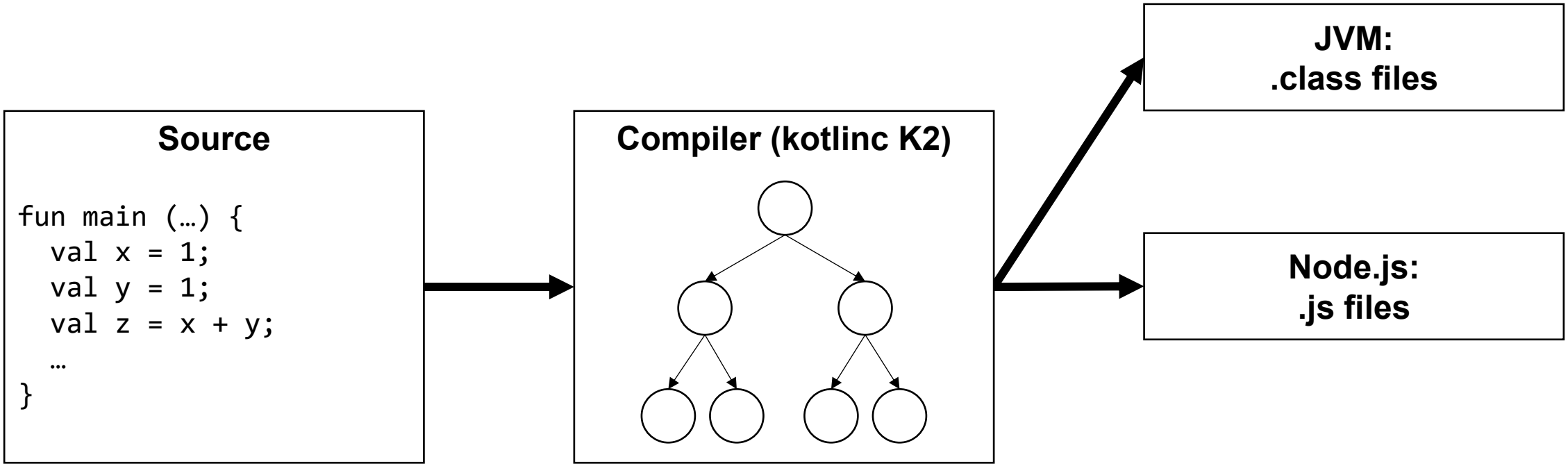
Introduction



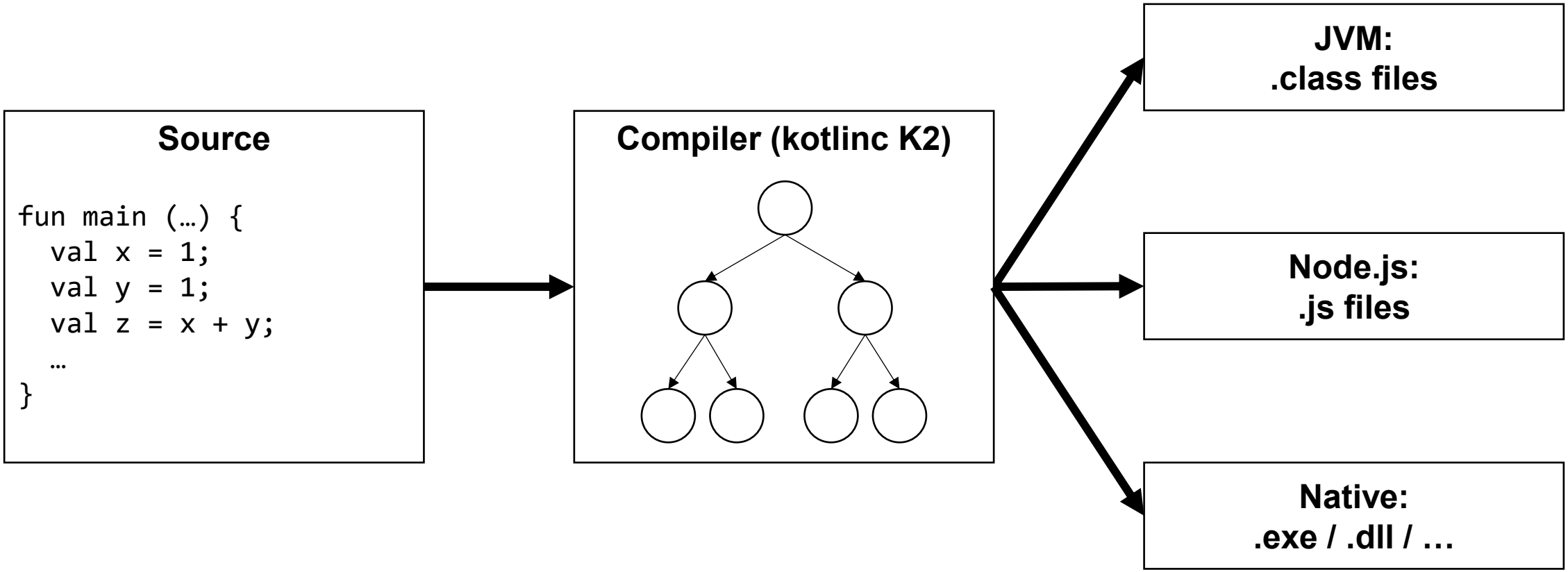
Introduction



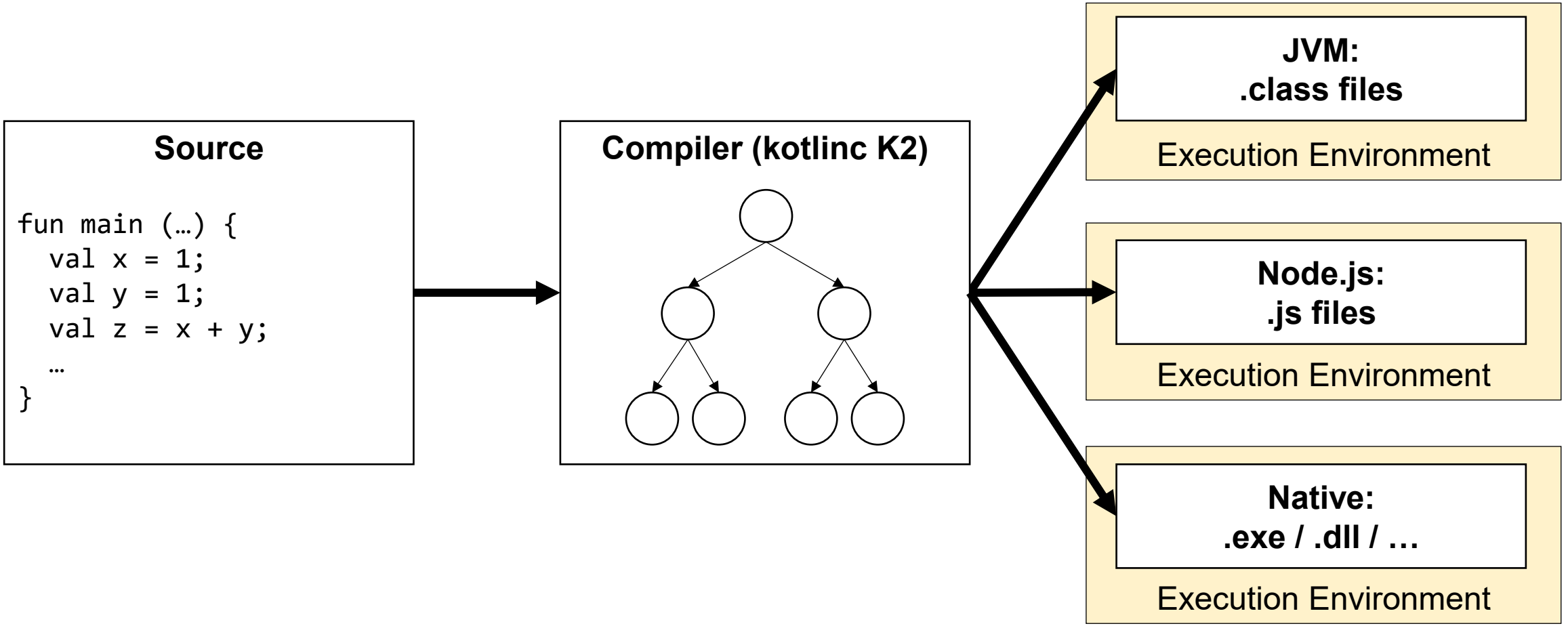
Introduction



Introduction

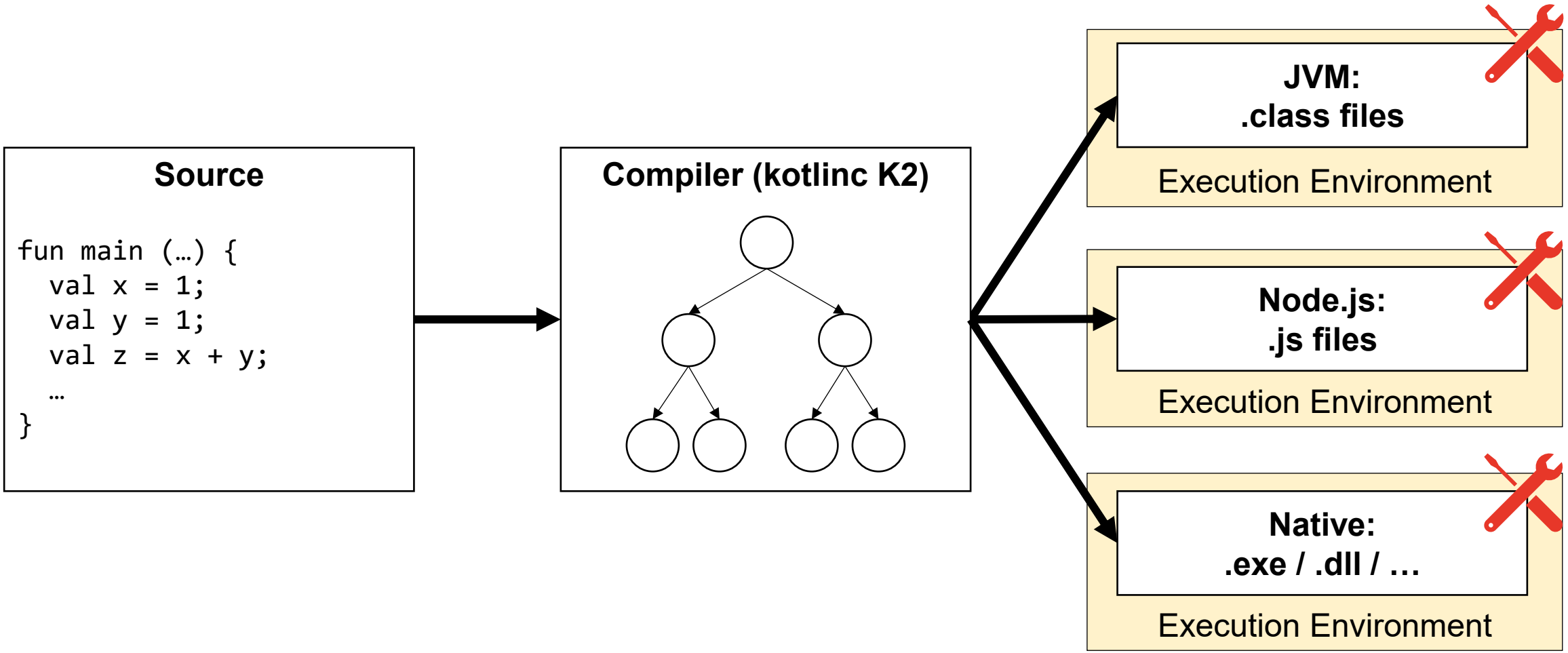


Introduction



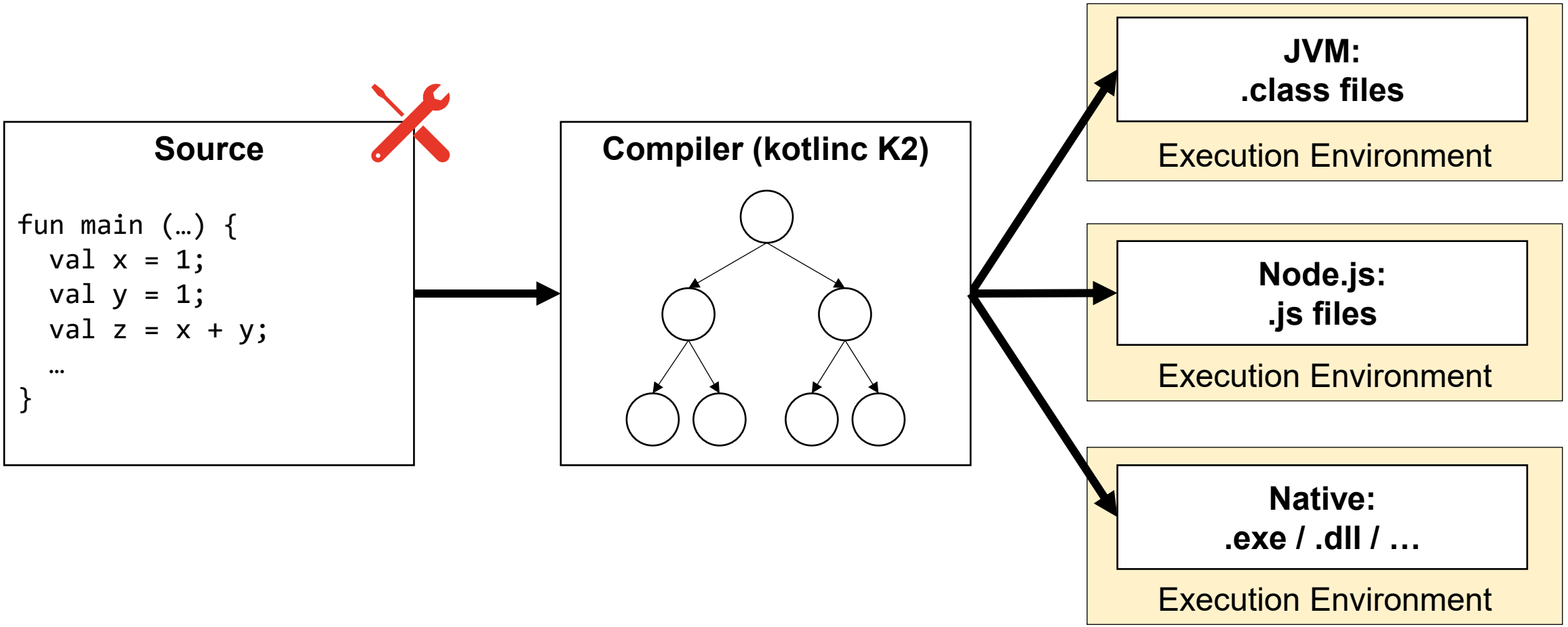
Introduction

(3) Code Instrumentation



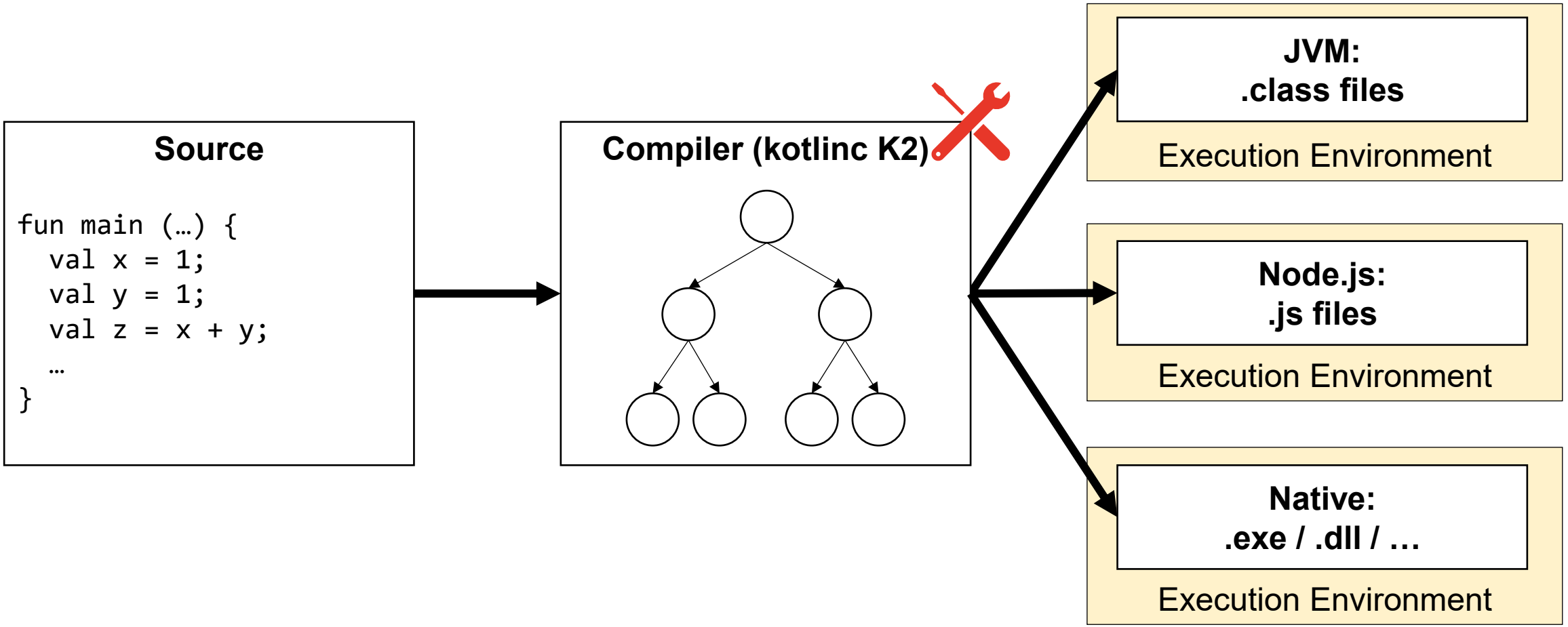
Introduction

(3) Code Instrumentation



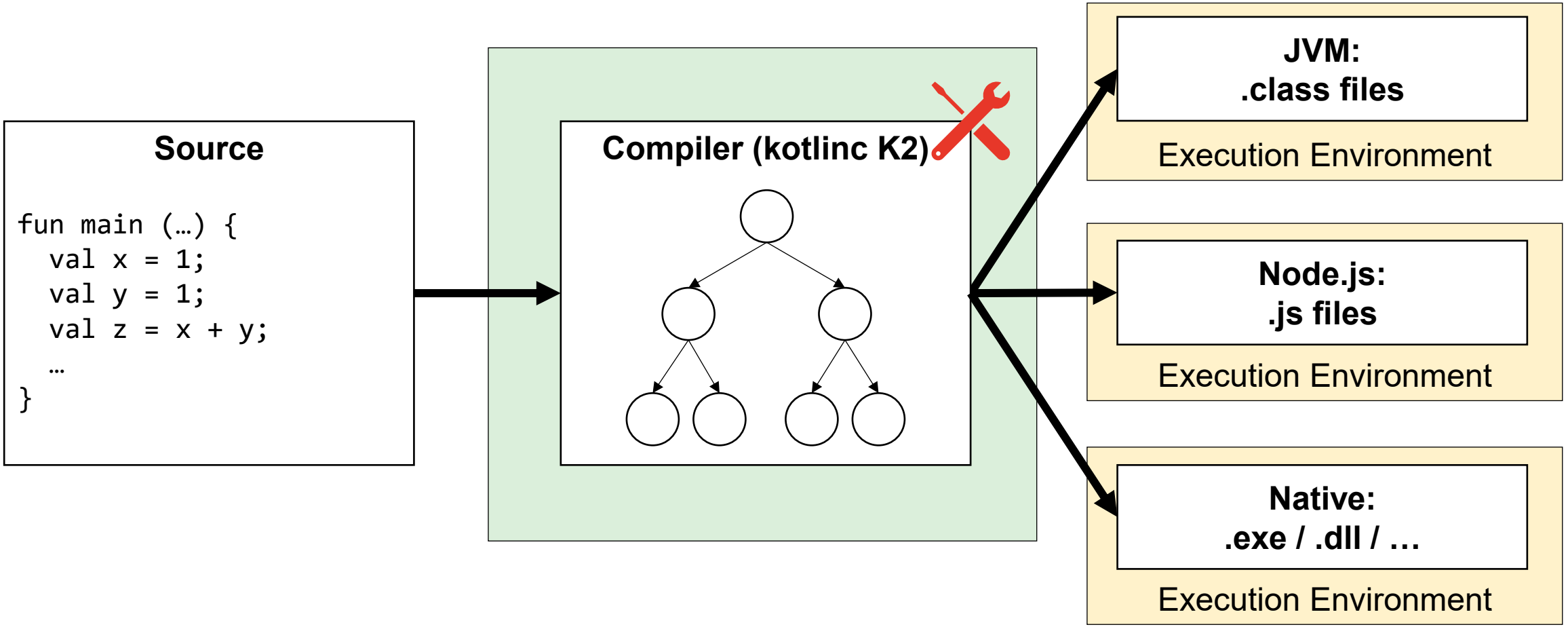
Introduction

(3) Code Instrumentation



Introduction

(3) Code Instrumentation



Introduction

Code Instrumentation ?

Source

```
fun main (…) {
  val x = 1;
  val y = 1;
  val z = x + y;
  …
}
```

Compiler (k

JVM:
ass files

ecution Environment

Node.js:
.js files

Execution Environment

Native:
.exe / .dll / …

Execution Environment

Tracing Performance Metrics in Kotlin Multiplatform Projects via Compile-Time Code Instrumentation

Markus Weninger
markus.weninger@jku.at
Institute for System Software, Johannes Kepler University Linz, Austria

Introduction

Code Instrumentation ?

Source

```
fun main (…) {
  val x = 1;
  val y = 1;
  val z = x + y;
  …
}
```

Compiler (k

JVM:
ass files

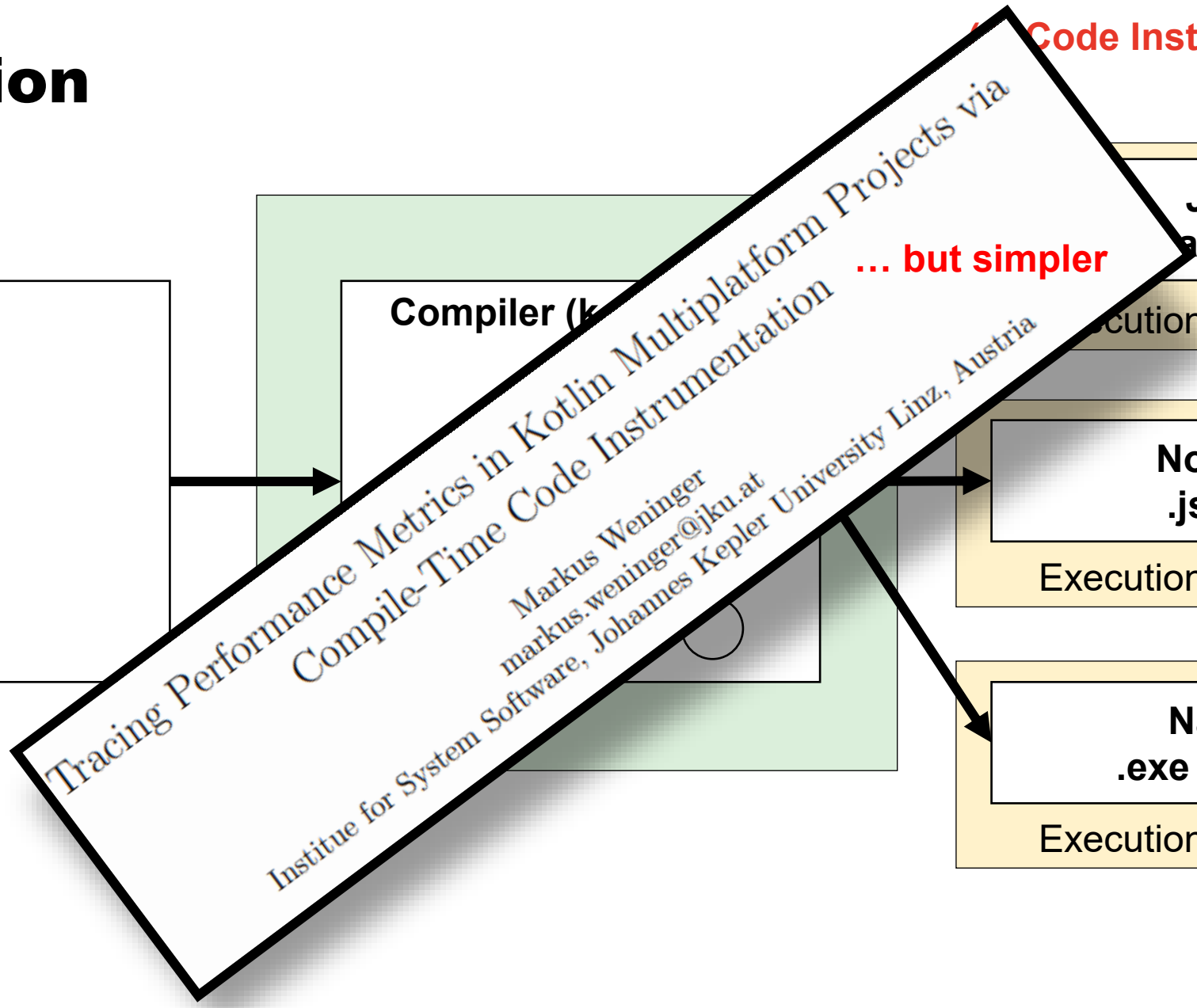
ecution Environment

Node.js:
.js files

Execution Environment

Native:
.exe / .dll / …

Execution Environment

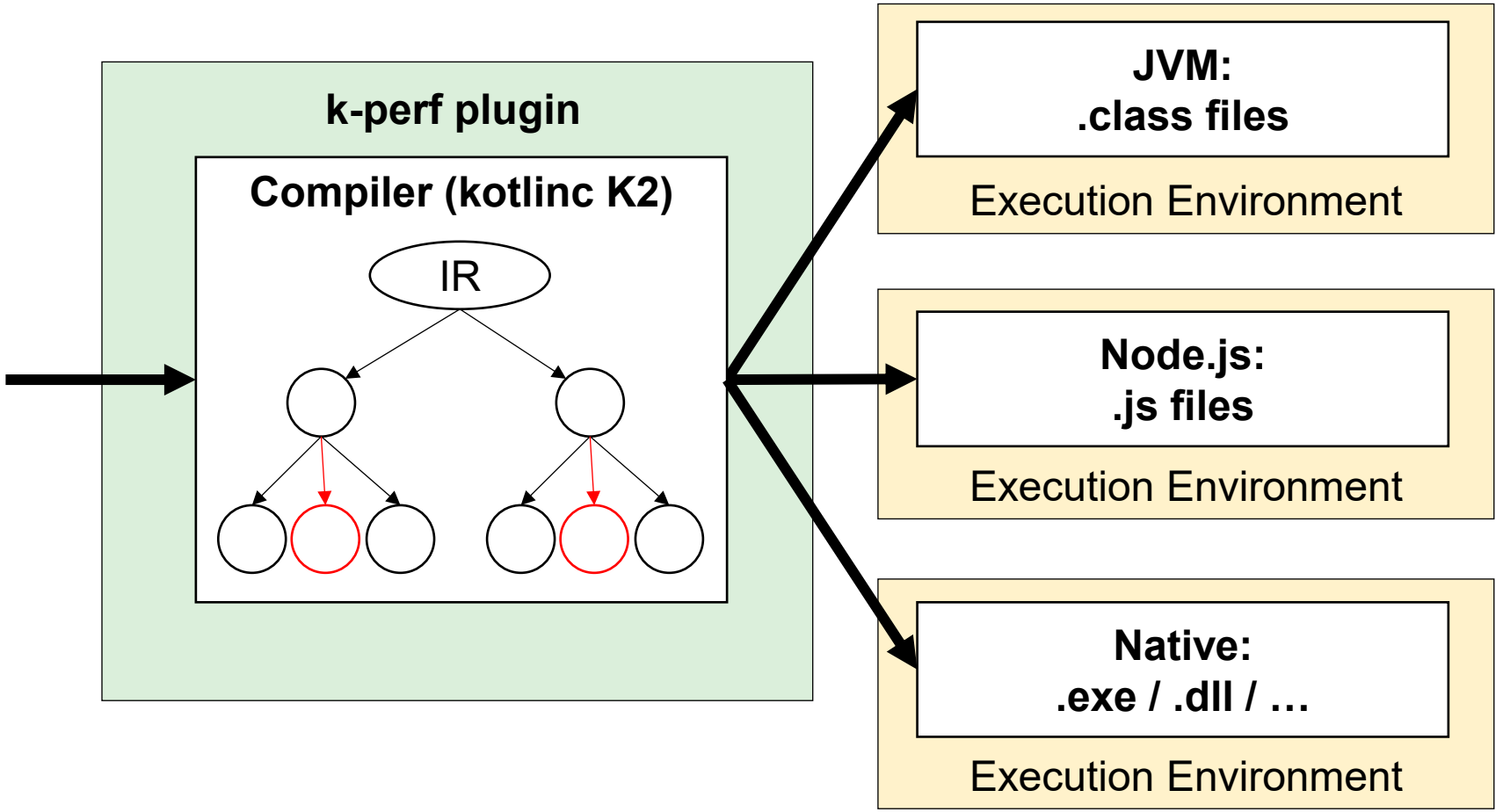


… but simpler

K-perf Plugin

Source

```
fun main () {
  val start = _enter(1)
  try {
    val x = 1
    val y = 1
    val z = x + y
  } finally {
    _exit(start)
  }
}
```

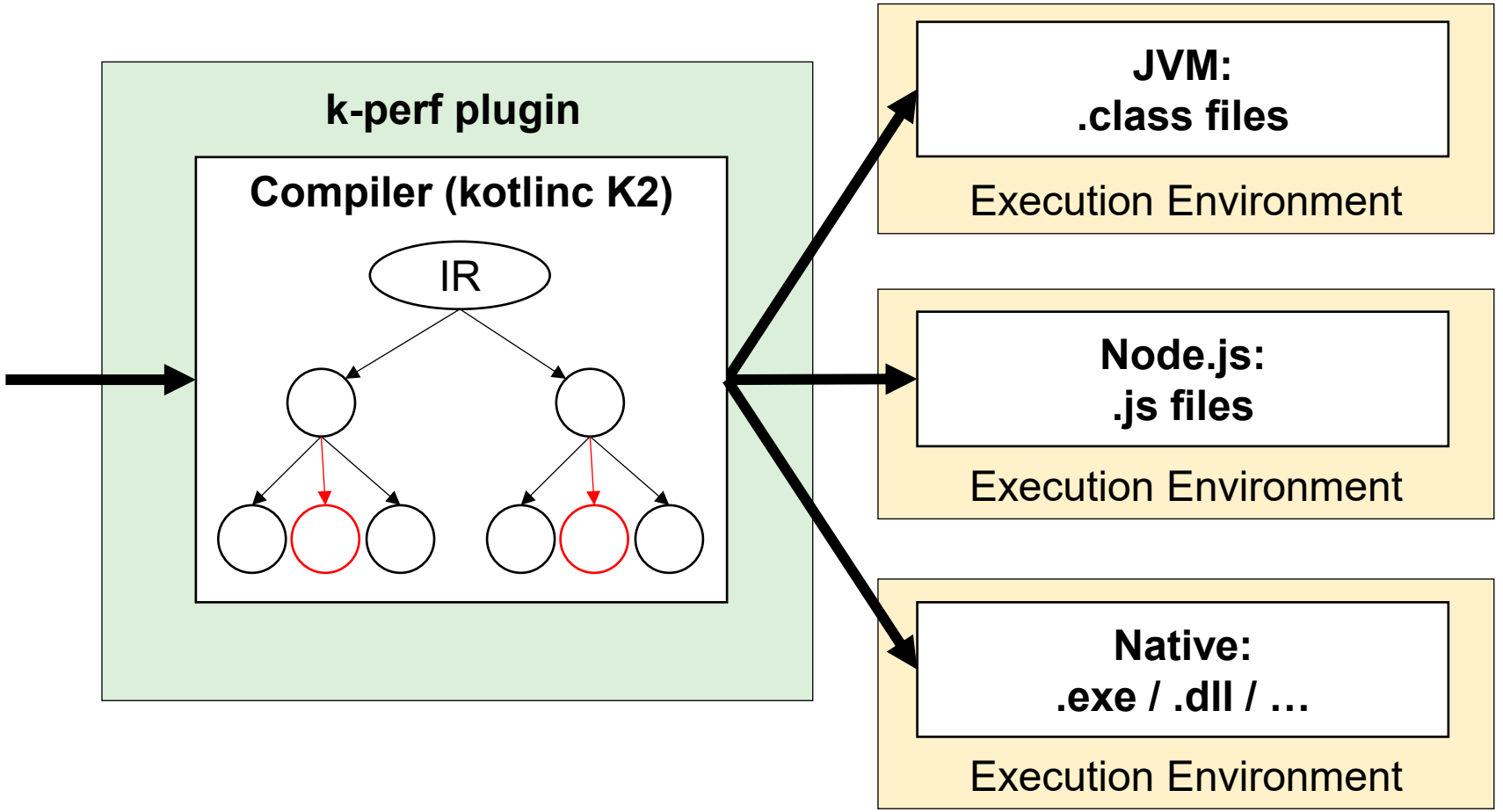


K-perf Plugin

Added to all methods

Source

```
fun main () {
  val start = _enter(1)
  try {
    val x = 1
    val y = 1
    val z = x + y
  } finally {
    _exit(start)
  }
}
```



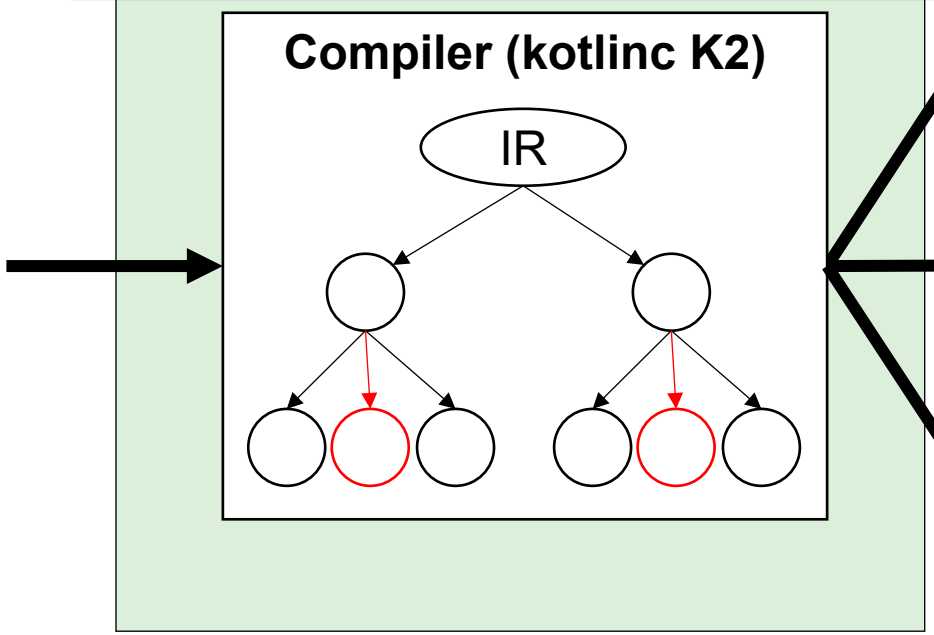
K-perf Plugin

Added to all methods

Source

```
fun main () {
  val start = _enter(1)
  try {
    val x = 1
    val y = 1
    val z = x + y
  } finally {
    _exit(start)
  }
}
```

```
>;1       // fun 1 entered
>;2       // fun 2 entered
<;230     // fun 2 left after 0.23ms
>;2       // fun 2 entered
<;250     // fun 2 left after 0.25ms
<;7500    // fun 1 left after 7.5ms
```



JVM:
.class files

Execution Environment

Node.js:
.js files

Execution Environment

Native:
.exe / .dll / …

Execution Environment

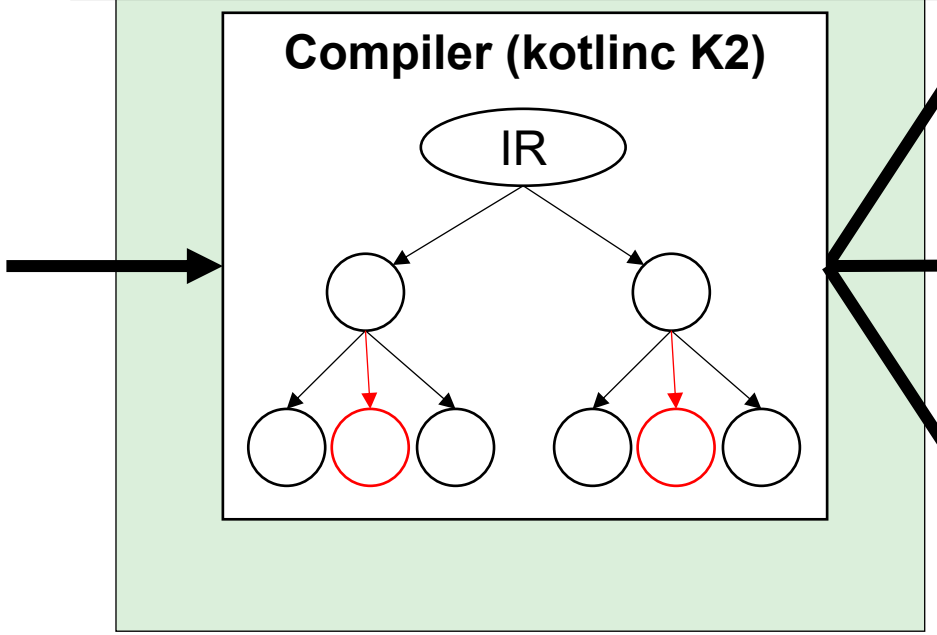
K-perf Plugin

Added to all methods

Source

```
fun main () {
  val start = _enter(1)
  try {
    val x = 1
    val y = 1
    val z = x + y
  } finally {
    _exit(start)
  }
}
```

```
>;1       // fun 1 entered
>;2       // fun 2 entered
<;230     // fun 2 left after 0.23ms
>;2       // fun 2 entered
<;250     // fun 2 left after 0.25ms
<;7500    // fun 1 left after 7.5ms
```



JVM:
.class files

Execution Environment

Node.js:
.js files

Execution Environment

Native:
.exe / .dll / …

Execution Environment

We want to simplify the generation of those additional IR nodes

K-perf Plugin

K-perf Plugin

```
val bufferedTraceFileSink : Sink =
  fs.sink(Path(traceName)).buffered()
```

K-perf Plugin

Sink ~ File

```
val bufferedTraceFileSink : Sink =
  fs.sink(Path(traceName)).buffered()
```

K-perf Plugin

Sink ~ File

```
val bufferedTraceFileSink : Sink =
  fs.sink(Path(traceName)).buffered()
```

```
val bufferedTraceFileSink = pluginContext.irFactory.buildField {
    name = Name.identifier("bufferedTraceFileSink")
    type = rawSinkClass.defaultType
    isFinal = false
    isStatic = true
}.apply {
    initializer = DeclarationIrBuilder(pluginContext, firstFile.symbol).run {
        irExprBody(irCall(bufferedFunc).apply {
            extensionReceiver = irCall(sinkFunc).apply {
                dispatchReceiver = irCall(systemFileSystem.owner.getter!!)
                putValueArgument(
                    0,
                    irCall(pathConstructionFunc).apply {
                        putValueArgument(0, irGetField(null, bufferedTraceFileName))
                    })
            }
        })
    }
}
```

K-perf Plugin

Sink ~ File

```
val bufferedTraceFileSink : Sink =
  fs.sink(Path(traceName)).buffered()
```

```
val bufferedTraceFileSink = pluginContext.irFactory.buildField {
    name = Name.identifier("bufferedTraceFileSink")
    type = rawSinkClass.defaultType
    isFinal = false
    isStatic = true
}.apply {
    initializer = DeclarationIrBuilder(pluginContext, firstFile.symbol).run {
        irExprBody(irCall(bufferedFunc).apply {
            extensionReceiver = irCall(sinkFunc).apply {
                dispatchReceiver = irCall(systemFileSystem.owner.getter!!)
                putValueArgument(
                    0,
                    irCall(pathConstructionFunc).apply {
                        putValueArgument(0, irGetField(null, bufferedTraceFileName))
                    })
            }
        })
    }
}
```

K-perf Plugin

Sink ~ File

```
val bufferedTraceFileSink : Sink =
  fs.sink(Path(traceName)).buffered()
```

```
val bufferedFunc = pluginContext.referenceFunctions(
   CallableId(
      FqName("kotlinx.io"),
      Name.identifier("buffered")
   )
).single { it.owner.extensionReceiverParameter!!.type
== sinkFunc.owner.returnType }
```

```
val rawSinkClass = …
```

```
val sinkFunc = …
```

```
val systemFileSystem = …
```

```
val pathConstructionFunc = …
```

```
val bufferedTraceFileSink = pluginContext.irFactory.buildField {
   name = Name.identifier("bufferedTraceFileSink")
   type = rawSinkClass.defaultType
   isFinal = false
   isStatic = true
}.apply {
   initializer = DeclarationIrBuilder(pluginContext, firstFile.symbol).run {
      irExprBody(irCall(bufferedFunc).apply {
         extensionReceiver = irCall(sinkFunc).apply {
            dispatchReceiver = irCall(systemFileSystem.owner.getter!!)
            putValueArgument(
               0,
               irCall(pathConstructionFunc).apply {
                  putValueArgument(0, irGetField(null, bufferedTraceFileName))
               })
         }
      })
   }
}
```

K-perf Plugin

Sink ~ File

```
val bufferedTraceFileSink : Sink =
  fs.sink(Path(traceName)).buffered()
```

```
val bufferedFunc = pluginContext.referenceFunctions(
   CallableId(
      FqName("kotlinx.io"),
      Name.identifier("buffered")
   )
).single { it.owner.extensionReceiverParameter!!.type
== sinkFunc.owner.returnType }
```

```
val rawSinkClass = …
```

```
val sinkFunc = …
```

```
val systemFileSystem = …
```

```
val pathConstructionFunc = …
```

```
val bufferedTraceFileSink = pluginContext.irFactory.buildField {
   name = Name.identifier("bufferedTraceFileSink")
   type = rawSinkClass.defaultType
   isFinal = false
   isStatic = true
}.apply {
   initializer = DeclarationIrBuilder(pluginContext, firstFile.symbol).run {
      irExprBody(irCall(bufferedFunc).apply {
         extensionReceiver = irCall(sinkFunc).apply {
            dispatchReceiver = irCall(systemFileSystem.owner.getter!!)
            putValueArgument(
               0,
               irCall(pathConstructionFunc).apply {
                  putValueArgument(0, irGetField(null, bufferedTraceFileName))
               })
}
```

We first have to manually look up all classes, functions, properties, … we need in the generation process

K-perf Plugin

Sink ~ File

```
val bufferedTraceFileSink : Sink =
  fs.sink(Path(traceName)).buffer
```

```
val bufferedFunc = pluginContext.referenc
  CallableId(
    FqName("kotlinx.io"),
    Name.identifier("buffered")
  )
).single { it.owner.extensionReceiverP
== sinkFunc.owner.returnType }
```

```
val rawSinkClass = ...
```

```
val sinkFunc = ...
```

```
val systemFileSystem = ...
```

```
val pathConstructionFunc = ...
```

Future Work

```
redTraceFileSink = pluginContext.irFactory.buildField {
ntifier("bufferedTraceFileSink")
Type

der(pluginContext, firstFile.symbol).run {
dFunc).apply {
Call(sinkFunc).apply {
irCall(systemFileSystem.owner.getter!!)
t(

structionFunc).apply {
gument(0, irGetField(null, bufferedTraceFileName))
```

o manually look up all classes, functions,
, … we need in the generation process

K-perf Plugin

Sink ~ File

```
val bufferedTraceFileSink : Sink =
  fs.sink(Path(traceName)).buffer
```

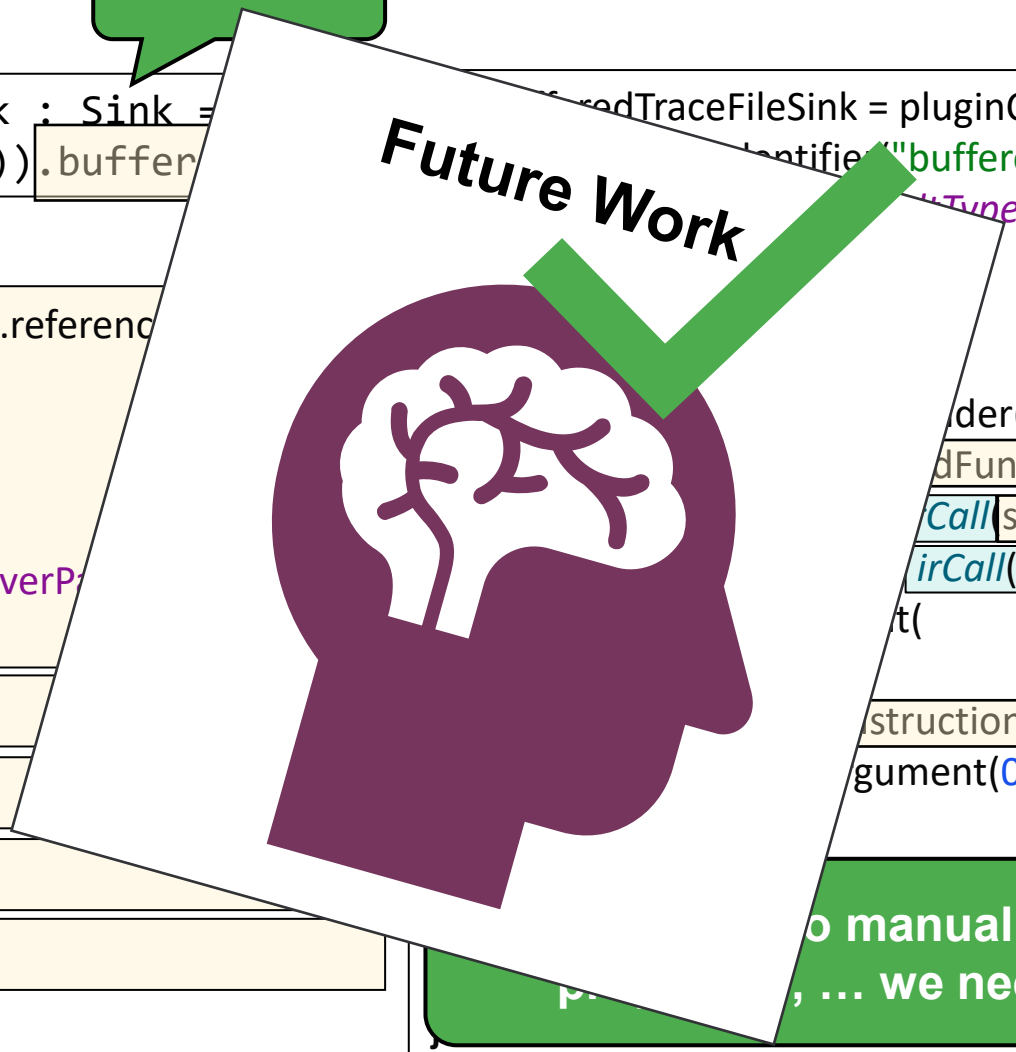
```
val bufferedFunc = pluginContext.referenc
  CallableId(
    FqName("kotlinx.io"),
    Name.identifier("buffered")
  )
).single { it.owner.extensionReceiverP
== sinkFunc.owner.returnType }
```

```
val rawSinkClass = ...
```

```
val sinkFunc = ...
```

```
val systemFileSystem = ...
```

```
val pathConstructionFunc = ...
```




```
redTraceFileSink = pluginContext.irFactory.buildField {
  ntifie("bufferedTraceFileSink")
  uType

der(pluginContext, firstFile.symbol).run {
dFunc).apply {
Call(sinkFunc).apply {
irCall(systemFileSystem.owner.getter!!)
t(

structionFunc).apply {
gument(0, irGetField(null, bufferedTraceFileName))
```

o manually look up all classes, functions,
, … we need in the generation process

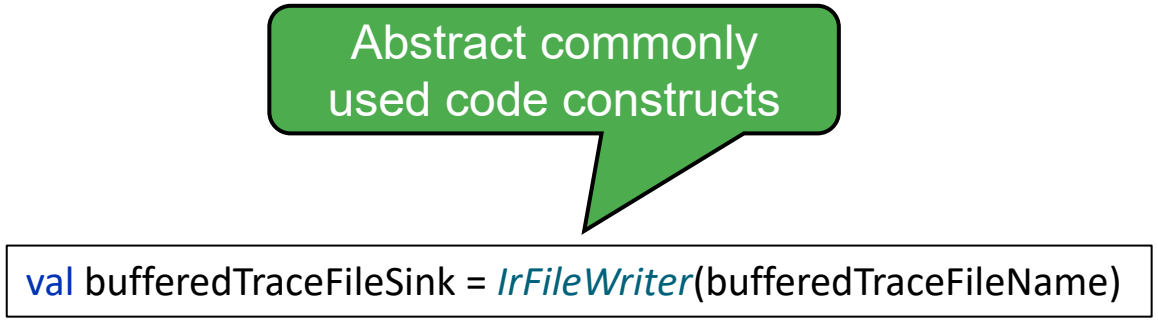
Goal of the KIRHelperKit

```
val bufferedTraceFileSink = pluginContext.irFactory.buildField {
    name = Name.identifier("bufferedTraceFileSink")
    type = rawSinkClass.defaultType
    isFinal = false
    isStatic = true
}.apply {
    initializer = DeclarationIrBuilder(pluginContext, firstFile.symbol).run {
        irExprBody(irCall(bufferedFunc).apply {
            extensionReceiver = irCall(sinkFunc).apply {
                dispatchReceiver = irCall(systemFileSystem.owner.getter!!)
                putValueArgument(
                    0,
                    irCall(pathConstructionFunc).apply {
                        putValueArgument(0, irGetField(null, bufferedTraceFileName))
                    })
            }
        })
    }
}
```

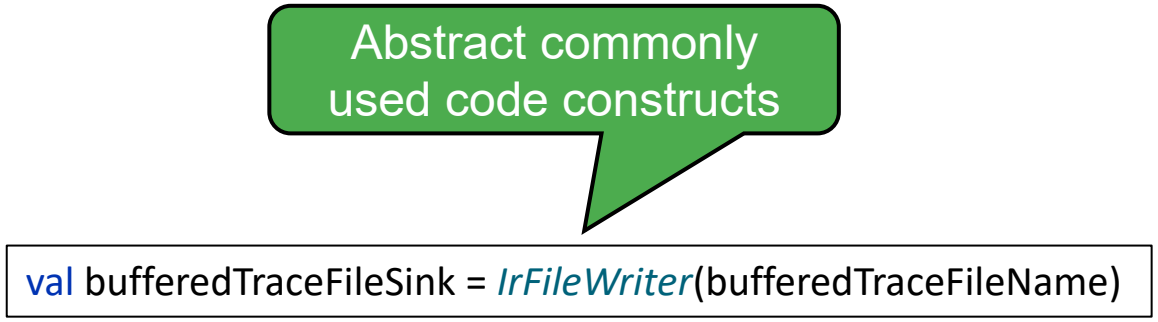
Goal of the KIRHelperKit

```
val bufferedTraceFileSink = IrFileWriter(bufferedTraceFileName)
```

Goal of the KIRHelperKit



Goal of the KIRHelperKit



Streamline lookup operations

Goal of the KIRHelperKit

Abstract commonly used code constructs

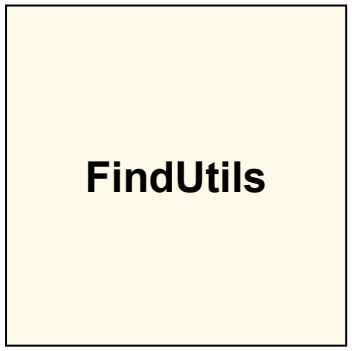
```
val bufferedTraceFileSink = IrFileWriter(bufferedTraceFileName)
```

Streamline lookup operations

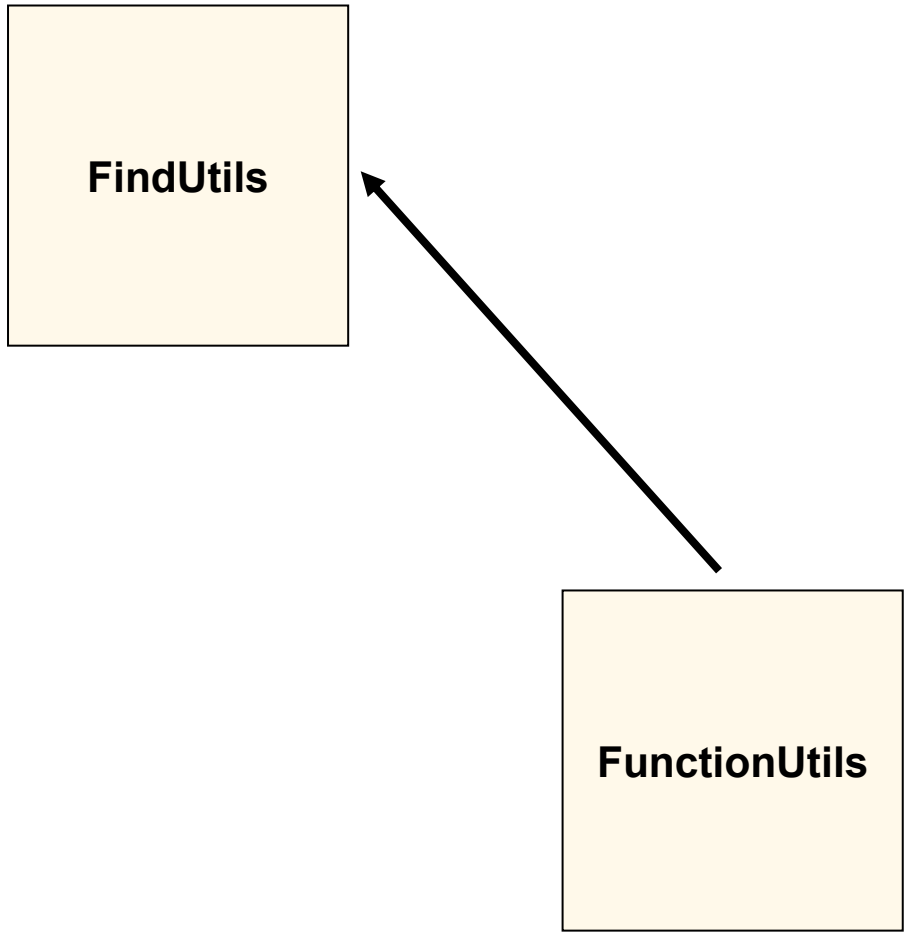
Reduce complexity of function call generation

KIRHelperKit

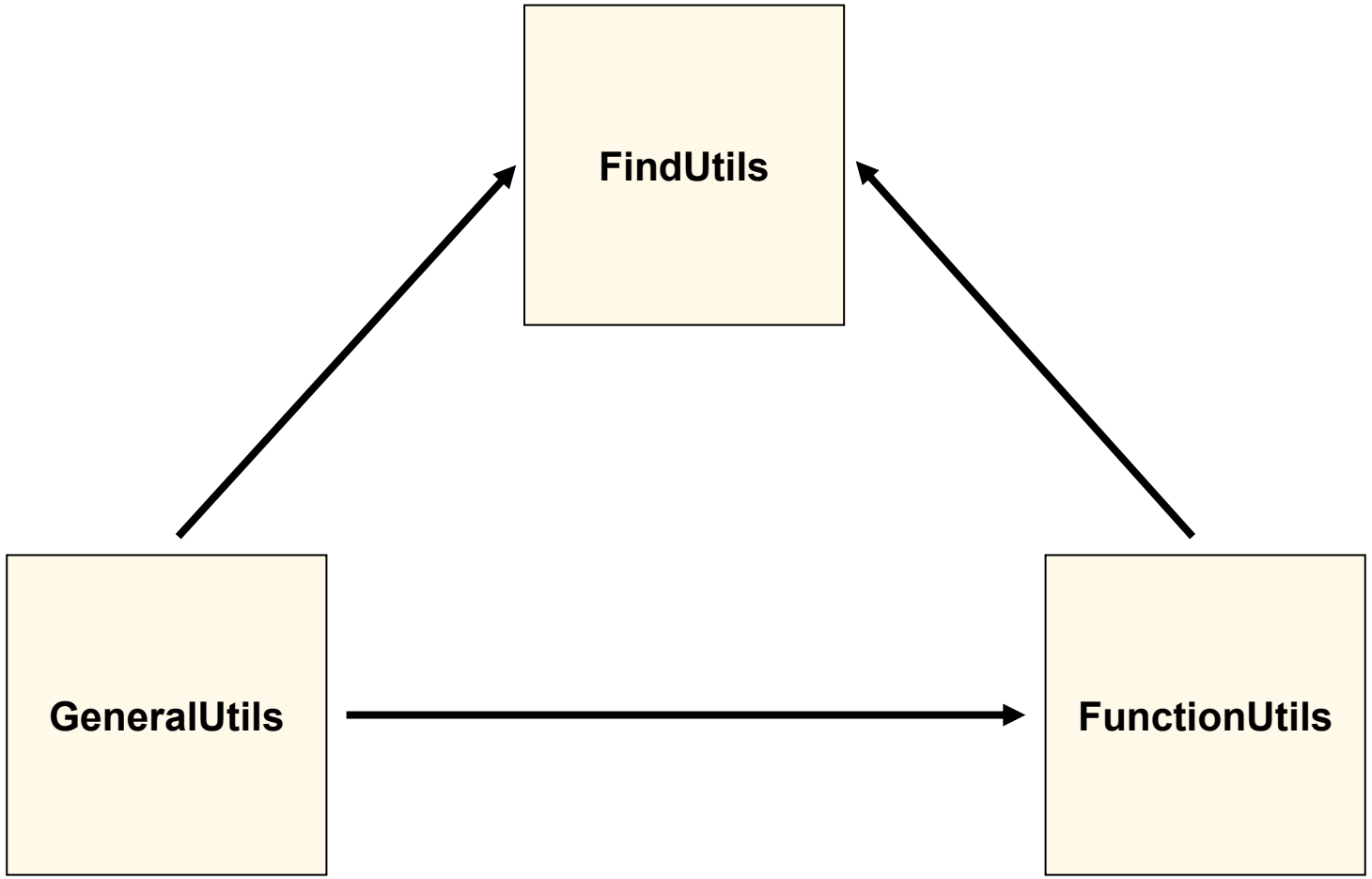
KIRHelperKit



KIRHelperKit



KIRHelperKit



KIRHelperKit

FindUtils

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
))
```

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
))
```

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
))
```

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
))
```

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
))
```

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
)).single { func ->
    func.valueParameters.size == 1 &&
    func.valueParameters[0].type ==
        pluginContext.irBuiltIns.intType
}
```

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
)).single { func ->
    func.valueParameters.size == 1 &&
    func.valueParameters[0].type ==
        pluginContext.irBuiltIns.intType
}
```

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
)).single { func ->
    func.valueParameters.size == 1 &&
    func.valueParameters[0].type ==
        pluginContext.irBuiltIns.intType
}
```

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
)).single { func ->
    func.valueParameters.size == 1 &&
    func.valueParameters[0].type ==
        pluginContext.irBuiltIns.intType
}
```



```
val addFn = pluginContext.findFunction(
    "java/util/concurrent/atomic/
    AtomicInteger.getAndAdd(int)"
)
```

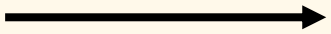
KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
)).single { func ->
    func.valueParameters.size == 1 &&
    func.valueParameters[0].type ==
        pluginContext.irBuiltIns.intType
}
```



```
val addFn = pluginContext.findFunction(
    "java/util/concurrent/atomic/
    AtomicInteger.getAndAdd(int)"
)
```

Search based on simple String

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
)).single { func ->
    func.valueParameters.size == 1 &&
    func.valueParameters[0].type ==
        pluginContext.irBuiltIns.intType
}
```

```
val addFn = pluginContext.findFunction(
    "java/util/concurrent/atomic/
    AtomicInteger.getAndAdd(int)"
)
```

Search based on simple String

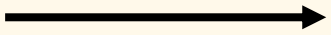
KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
)).single { func ->
    func.valueParameters.size == 1 &&
    func.valueParameters[0].type ==
        pluginContext.irBuiltIns.intType
}
```



```
val addFn = pluginContext.findFunction(
    "java/util/concurrent/atomic/
    AtomicInteger.getAndAdd(int)"
)
```

Search based on simple String

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
)).single { func ->
    func.valueParameters.size == 1 &&
    func.valueParameters[0].type ==
        pluginContext.irBuiltIns.intType
}
```

```
val addFn = pluginContext.findFunction(
    "java/util/concurrent/atomic/
    AtomicInteger.getAndAdd(int)"
)
```

Search based on simple String

KIRHelperKit

FindUtils

AtomicInteger.*getAndAdd*(Int)

Goal: find this function

```
val addFn =
pluginContext.referenceFunctions(CallableId(
    FqName("java.util.concurrent.atomic"),
    FqName("AtomicInteger"),
    Name.identifier("getAndAdd")
)).single { func ->
    func.valueParameters.size == 1 &&
    func.valueParameters[0].type ==
        pluginContext.irBuiltIns.intType
}
```

```
val addFn = pluginContext.findFunction(
    "java/util/concurrent/atomic/
    AtomicInteger.getAndAdd(int)"
)
```

Search based on simple String

KIRHelperKit

FindUtils

Global Lookup

KIRHelperKit

FindUtils

Global Lookup

```
pluginContext.findClass("package/class")
```

KIRHelperKit

FindUtils

Global Lookup

```
pluginContext.findClass("package/class")
```

```
pluginContext.findConstructor("package/class(p)")
```

KIRHelperKit

FindUtils

Global Lookup

```
pluginContext.findClass("package/class")
```

```
pluginContext.findConstructor("package/class(p)")
```

```
pluginContext.findFunction("package/class.fun(p)")
```

KIRHelperKit

FindUtils

Global Lookup

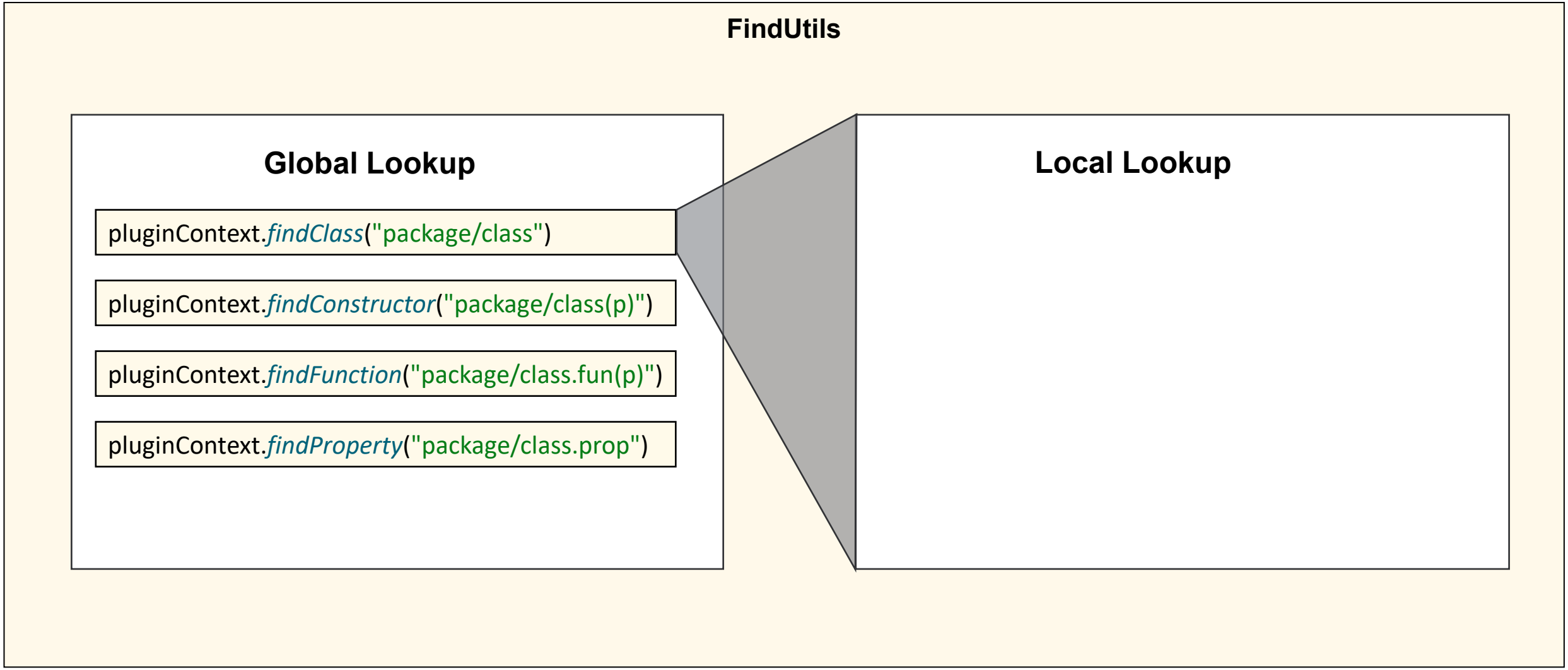
```
pluginContext.findClass("package/class")
```

```
pluginContext.findConstructor("package/class(p)")
```

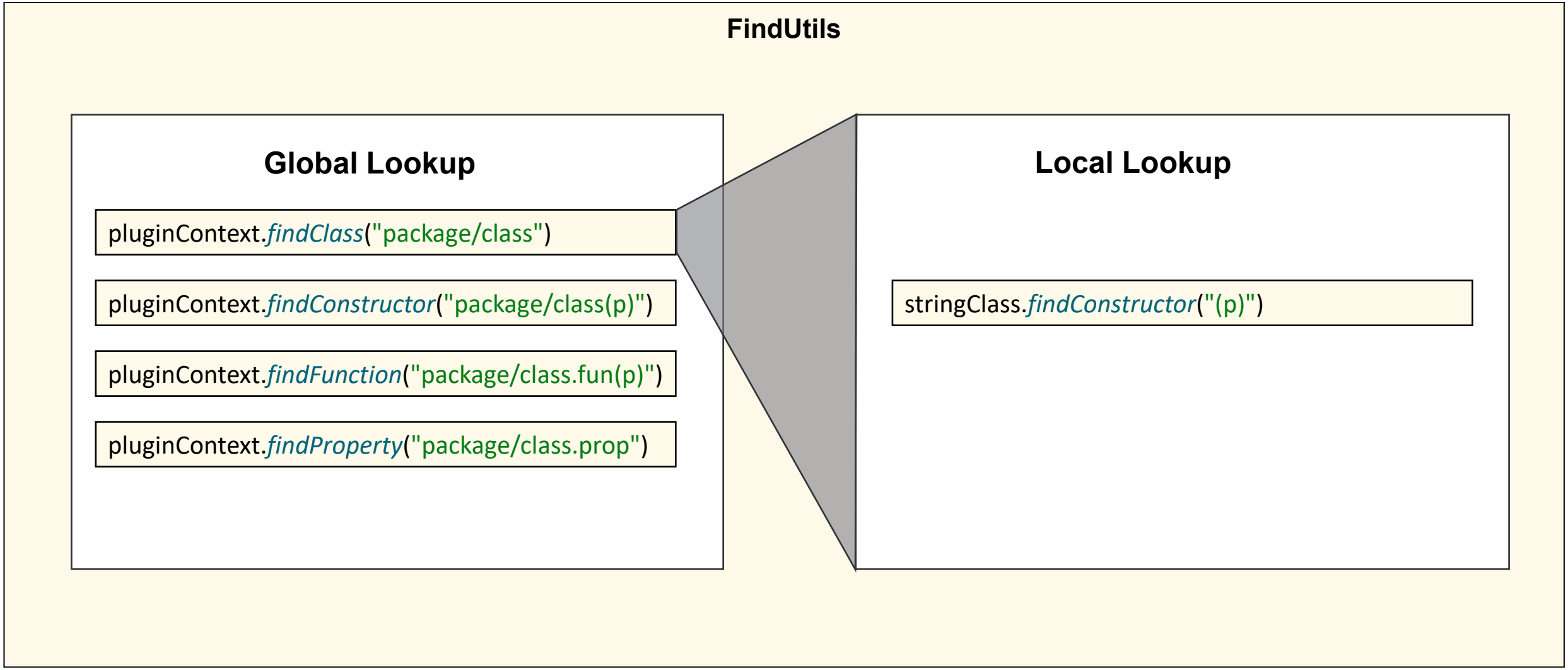
```
pluginContext.findFunction("package/class.fun(p)")
```

```
pluginContext.findProperty("package/class.prop")
```

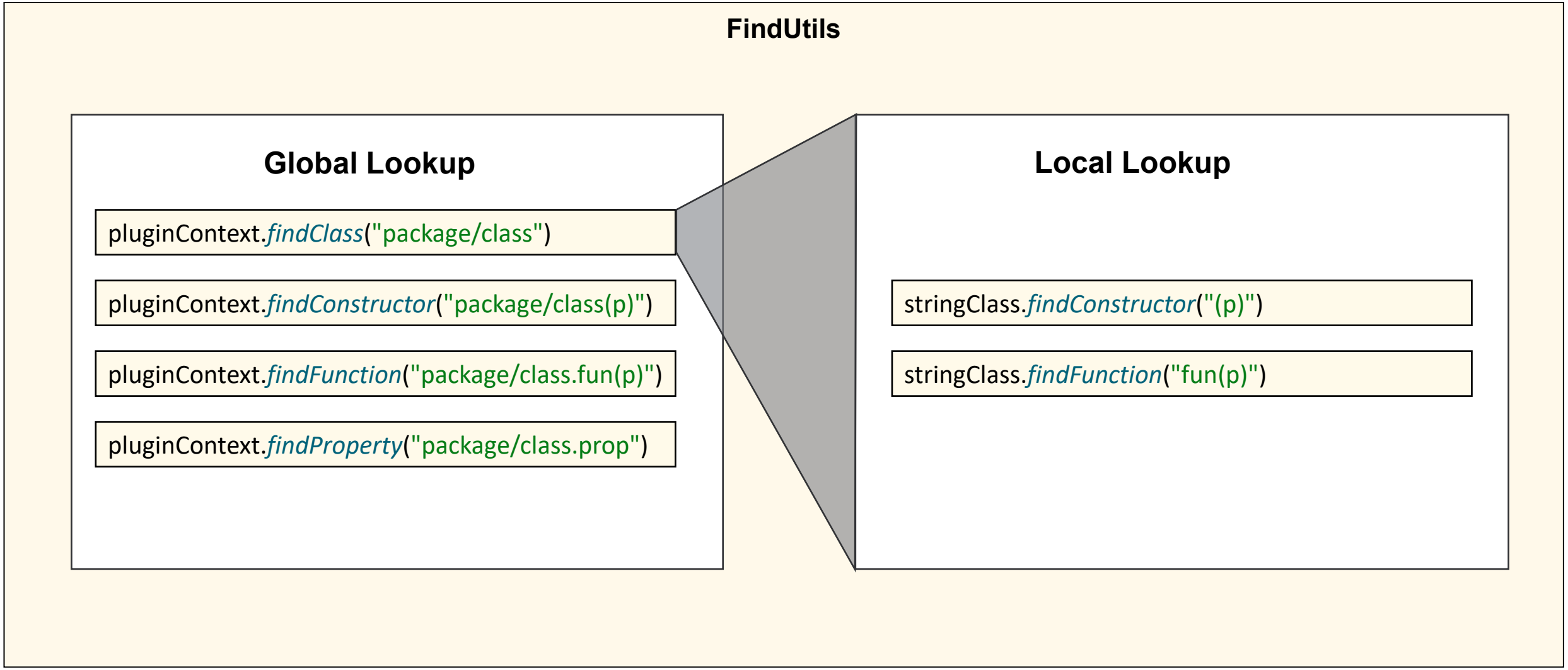
KIRHelperKit



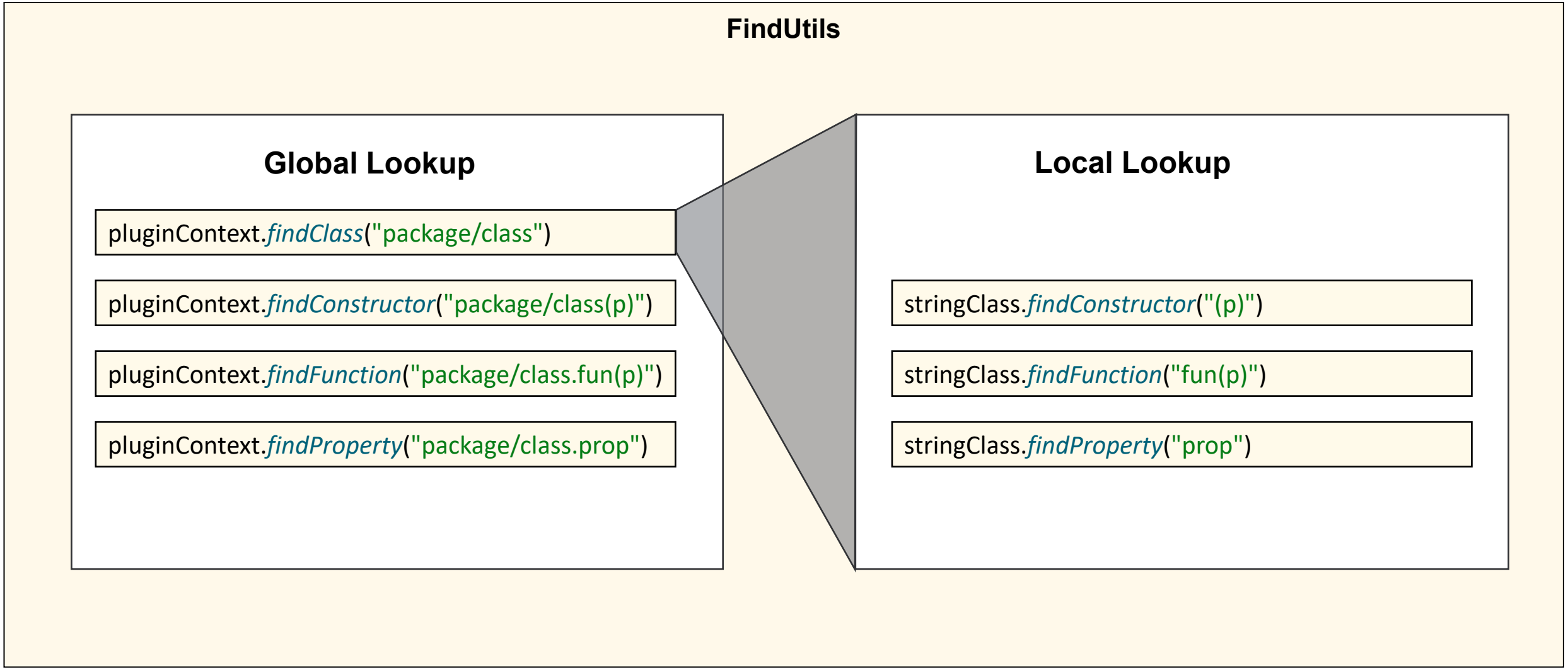
KIRHelperKit



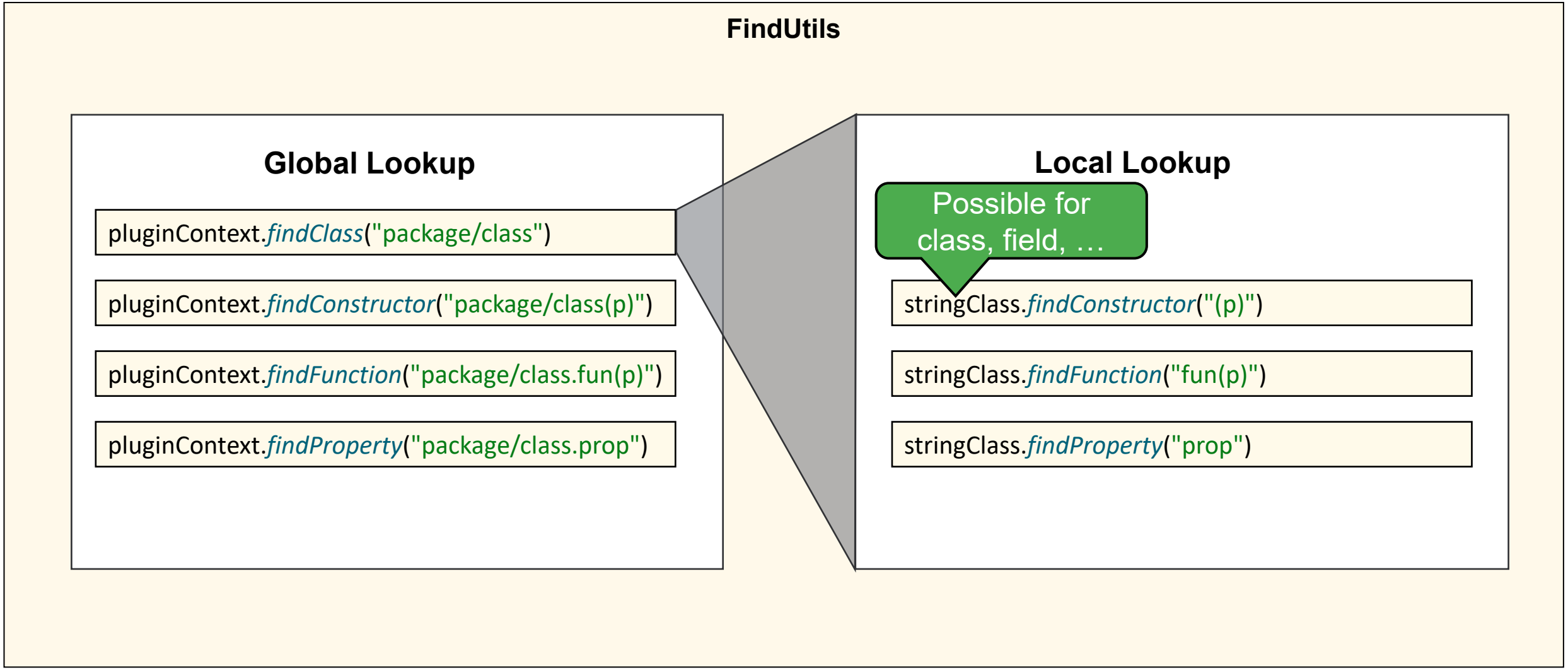
KIRHelperKit



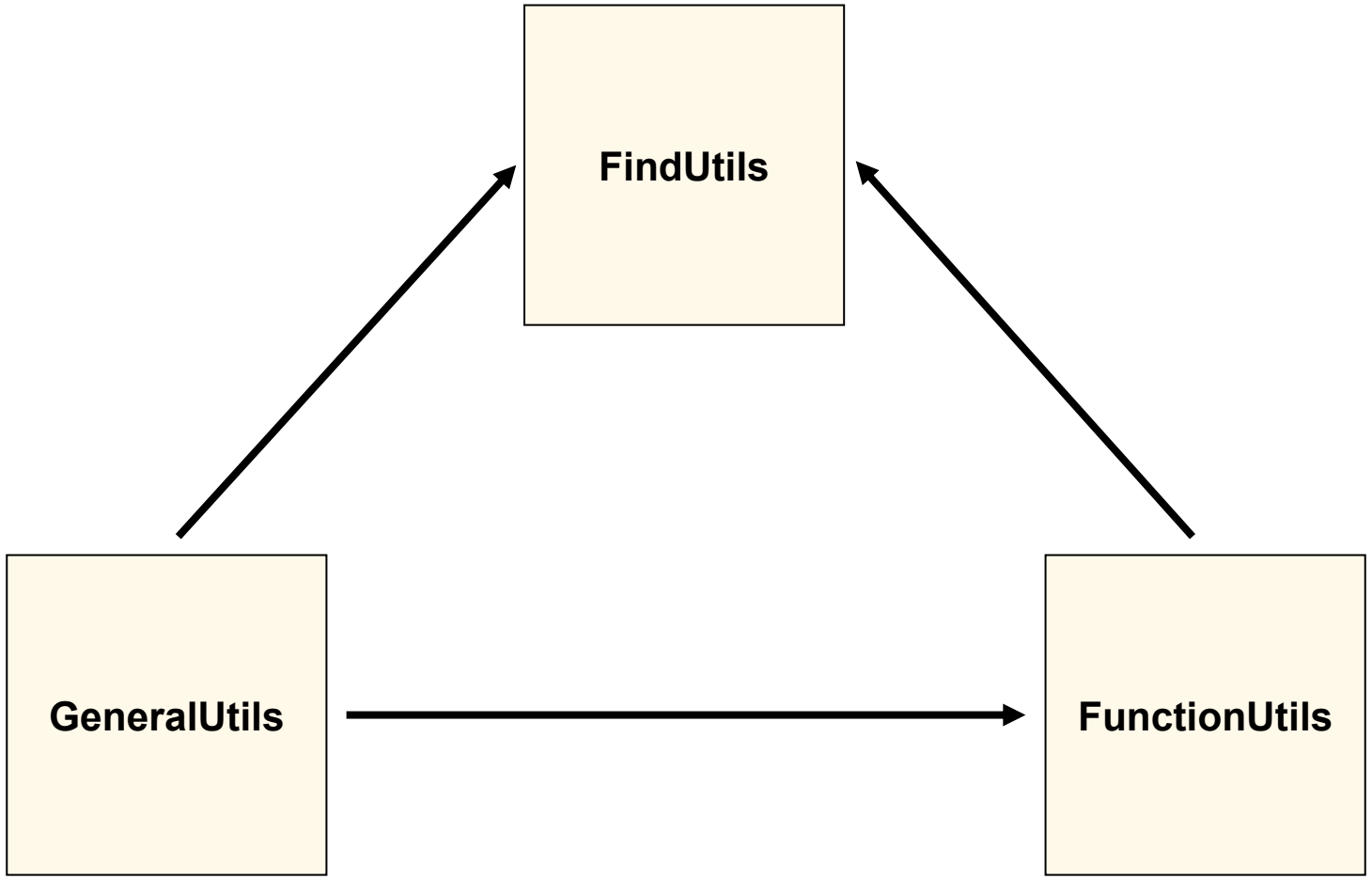
KIRHelperKit



KIRHelperKit



KIRHelperKit



KIRHelperKit

FunctionUtils

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = …
val toStringCall = irCall(toStringFn).apply {
    dispatchReceiver = irCall(addFn).apply {
        dispatchReceiver = irGetField(null, counter)
        putValueArgument(0, irInt(2))
    }
}
```

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = …
val toStringCall = irCall(toStringFn).apply {
    dispatchReceiver = irCall(addFn).apply {
        dispatchReceiver = irGetField(null, counter)
        putValueArgument(0, irInt(2))
    }
}
```

Recursive structure

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = …
val toStringCall = irCall(toStringFn).apply {
    dispatchReceiver = irCall(addFn).apply {
        dispatchReceiver = irGetField(null, counter)
        putValueArgument(0, irInt(2))
    }
}
```

Recursive structure

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = …
val toStringCall = irCall(toStringFn).apply {
   dispatchReceiver = irCall(addFn).apply {
      dispatchReceiver = irGetField(null, counter)
      putValueArgument(0, irInt(2))
   }
}
```

Recursive structure

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = ...
val toStringCall = irCall(toStringFn).apply {
    dispatchReceiver = irCall(addFn).apply {
        dispatchReceiver = irGetField(null, counter)
        putValueArgument(0, irInt(2))
    }
}
```

Recursive structure

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = ...
val toStringCall = irCall(toStringFn).apply {
    dispatchReceiver = irCall(addFn).apply {
        dispatchReceiver = irGetField(null, counter)
        putValueArgument(0, irInt(2))
    }
}
```

Recursive structure

→

```
val toStringCall = counter.call("getAndAdd(int)", 2).call("toString()")
```

KIRHelperKit

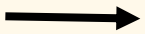
FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = ...
val toStringCall = irCall(toStringFn).apply {
    dispatchReceiver = irCall(addFn).apply {
        dispatchReceiver = irGetField(null, counter)
        putValueArgument(0, irInt(2))
    }
}
```

Recursive structure



```
val toStringCall = counter.call("getAndAdd(int)", 2).call("toString()")
```

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = ...
val toStringCall = irCall(toStringFn).apply {
    dispatchReceiver = irCall(addFn).apply {
        dispatchReceiver = irGetField(null, counter)
        putValueArgument(0, irInt(2))
    }
}
```

Recursive structure

→

```
val toStringCall = counter.call("getAndAdd(int)", 2).call("toString()")
```

KIRHelperKit

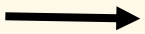
FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = ...
val toStringCall = irCall(toStringFn).apply {
    dispatchReceiver = irCall(addFn).apply {
        dispatchReceiver = irGetField(null, counter)
        putValueArgument(0, irInt(2))
    }
}
```

Recursive structure



```
val toStringCall = counter.call("getAndAdd(int)", 2).call("toString()")
```

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = ...
val toStringCall = irCall(toStringFn).apply {
  dispatchReceiver = irCall(addFn).apply {
    dispatchReceiver = irGetField(null, counter)
    putValueArgument(0, irInt(2))
  }
}
```

Recursive structure

→

```
val toStringCall = counter.call("getAndAdd(int)", 2).call("toString()")
```

Automatic conversion of parameters

KIRHelperKit

FunctionUtils

```
val couter = AtomicInteger()
counter.getAndAdd(2).toString()
```

Goal: generate these function calls

```
val toStringFn = ...
val addFn = ...
val toStringCall = irCall(toStringFn).apply {
    dispatchReceiver = irCall(addFn).apply {
        dispatchReceiver = irGetField(null, counter)
        putValueArgument(0, irInt(2))
    }
}
```

Recursive structure

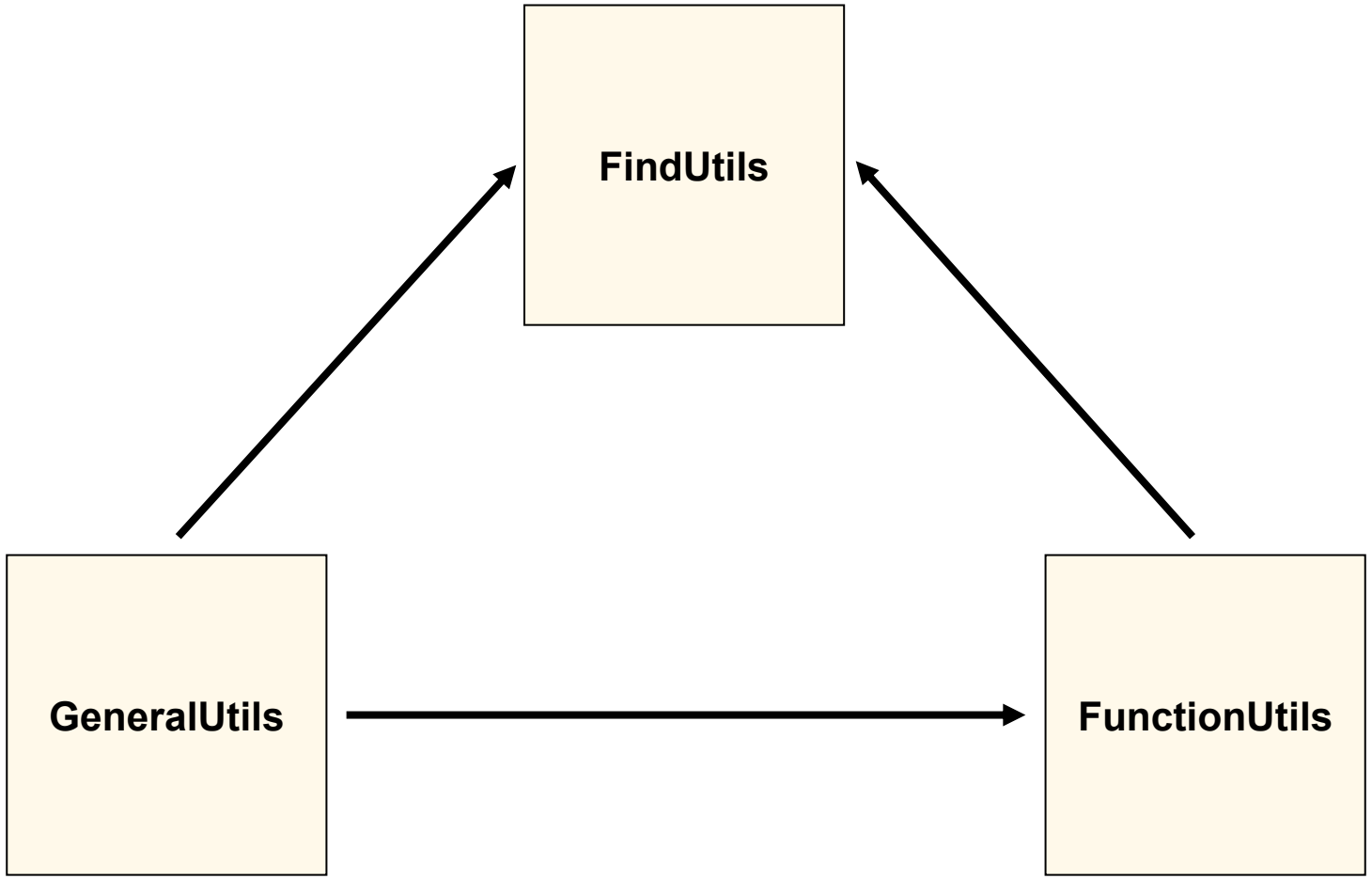
→

```
val toStringCall = counter.call("getAndAdd(int)", 2).call("toString()")
```

No more need for manual lookup

Automatic conversion of parameters

KIRHelperKit



KIRHelperKit

GeneralUtils

KIRHelperKit

GeneralUtils

```
IrPluginContext.generateField ("fieldName", init)
```

KIRHelperKit

GeneralUtils

```
IrPluginContext.generateField ("fieldName", init)
```

```
IrStringBuilder()
```

KIRHelperKit

GeneralUtils

```
IrPluginContext.generateField ("fieldName", init)
```

```
IrStringBuilder()
```

```
IrFileReader("fileName")
```

KIRHelperKit

GeneralUtils

```
IrPluginContext.generateField ("fieldName", init)
```

```
IrStringBuilder()
```

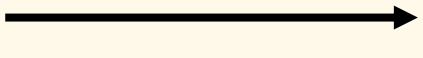
```
IrFileReader("fileName")
```

```
IrFileWriter("fileName")
```

KIRHelperKit

GeneralUtils

IrPluginContext.*generateField* ("fieldName", init)



```
val field = pluginContext.generateField("field") {
    //... initializer
}
```

IrStringBuilder()

IrFileReader("fileName")

IrFileWriter("fileName")

KIRHelperKit

GeneralUtils

IrPluginContext.*generateFie*

IrStringBuilder()

IrFileReader("fileName")

IrFileWriter("fileName")

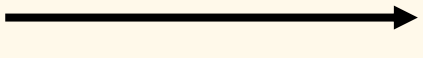
```
val output = pluginContext.irFactory.buildField {
    name = Name.identifier("output")
    type = rawSinkClass.defaultType
    isFinal = false
    isStatic = true
}.apply {
    initializer = DeclarationIrBuilder(pluginContext, firstFile.symbol).run {
        irExprBody(irCall(bufferedFunc).apply {
            extensionReceiver = irCall(sinkFunc).apply {
                dispatchReceiver = irCall(systemFileSystem.owner.getter!!)
                putValueArgument(
                    0,
                    irCall(pathConstructionFunc).apply {
                        putValueArgument(0, irGetField(null, bufferedTraceFileName))
                    })
            }
        })
    }
}
```

rateField("field") {

KIRHelperKit

GeneralUtils

IrPluginContext.*generateField* ("fieldName", init)



```
val field = pluginContext.generateField("field") {
    //... initializer
}
```

IrStringBuilder()

IrFileReader("fileName")

IrFileWriter("fileName")

KIRHelperKit

GeneralUtils

IrPluginContext.*generateField* ("fieldName", init)

```
val field = pluginContext.generateField("field") {
    //... initializer
}
```

IrStringBuilder()

IrFileReader("fileName")

IrFileWriter("fileName")

```
val output = IrFileWriter("fileName")
output.writeData ("Hello World!")
```

KIRHelperKit

GeneralUtils

IrPluginContext.*generateField* ("fieldName", init)

```
val field = pluginContext.generateField("field") {
   //... initializer
}
```

IrStringBuilder()

IrFileReader("fileName")

IrFileWriter("fileName")

Generate file source field

```
val output = IrFileWriter("fileName")
output.writeData ("Hello World!")
```

KIRHelperKit

GeneralUtils

IrPluginContext.*generateField* ("fieldName", init)

```
val field = pluginContext.generateField("field") {
   //... initializer
}
```

IrStringBuilder()

IrFileReader("fileName")

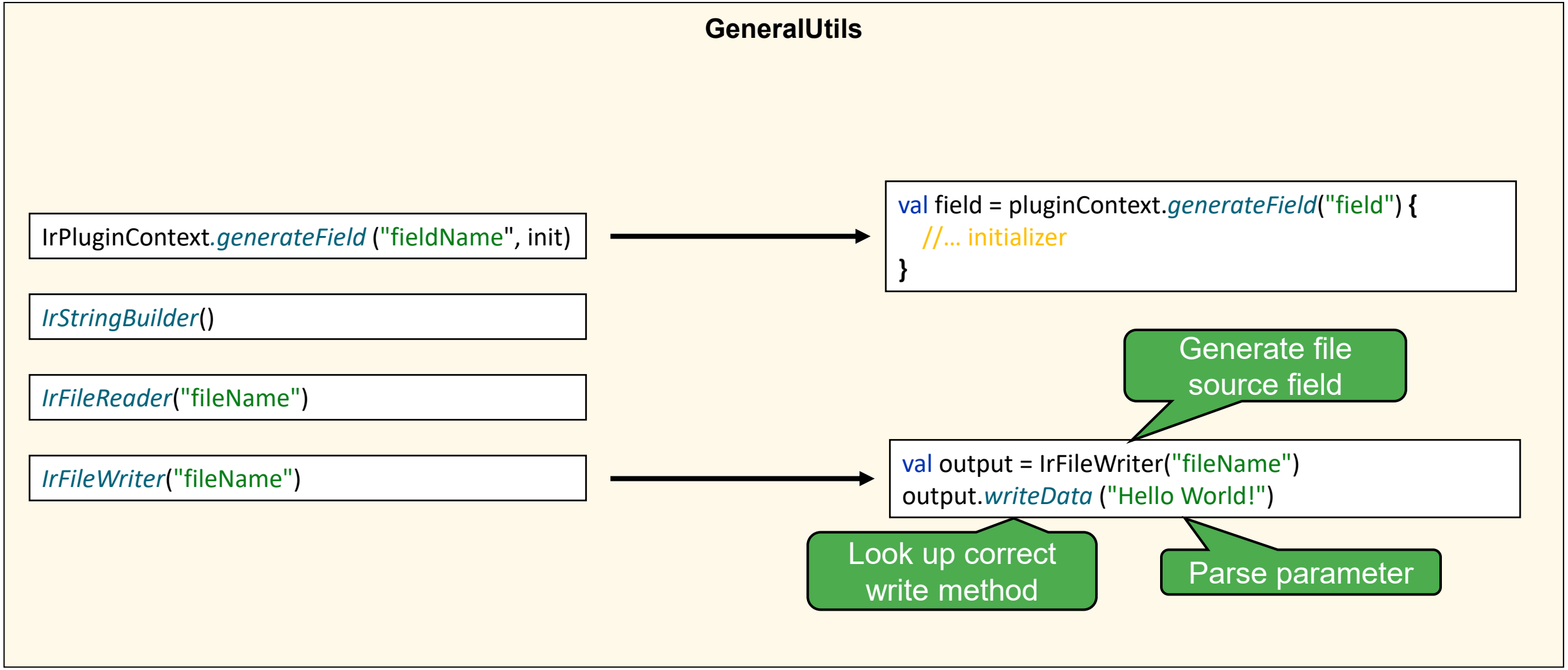
IrFileWriter("fileName")

```
val output = IrFileWriter("fileName")
output.writeData ("Hello World!")
```

Generate file source field

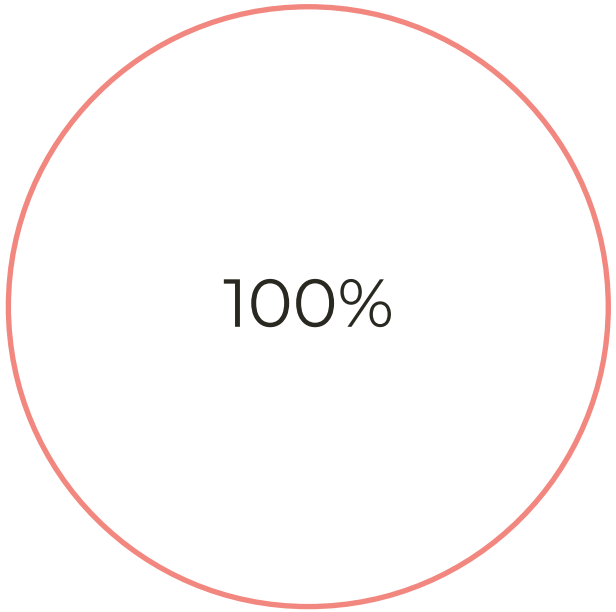
Look up correct write method

KIRHelperKit



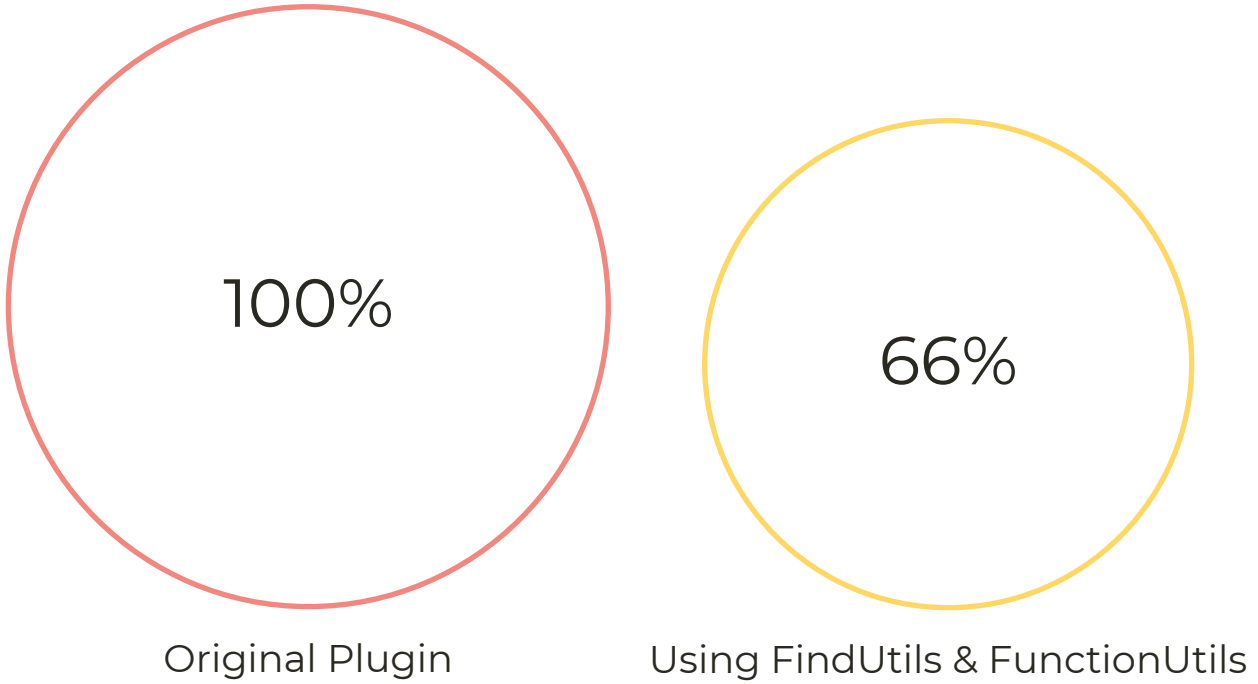
Results: Code Reduction

Results: Code Reduction



Original Plugin

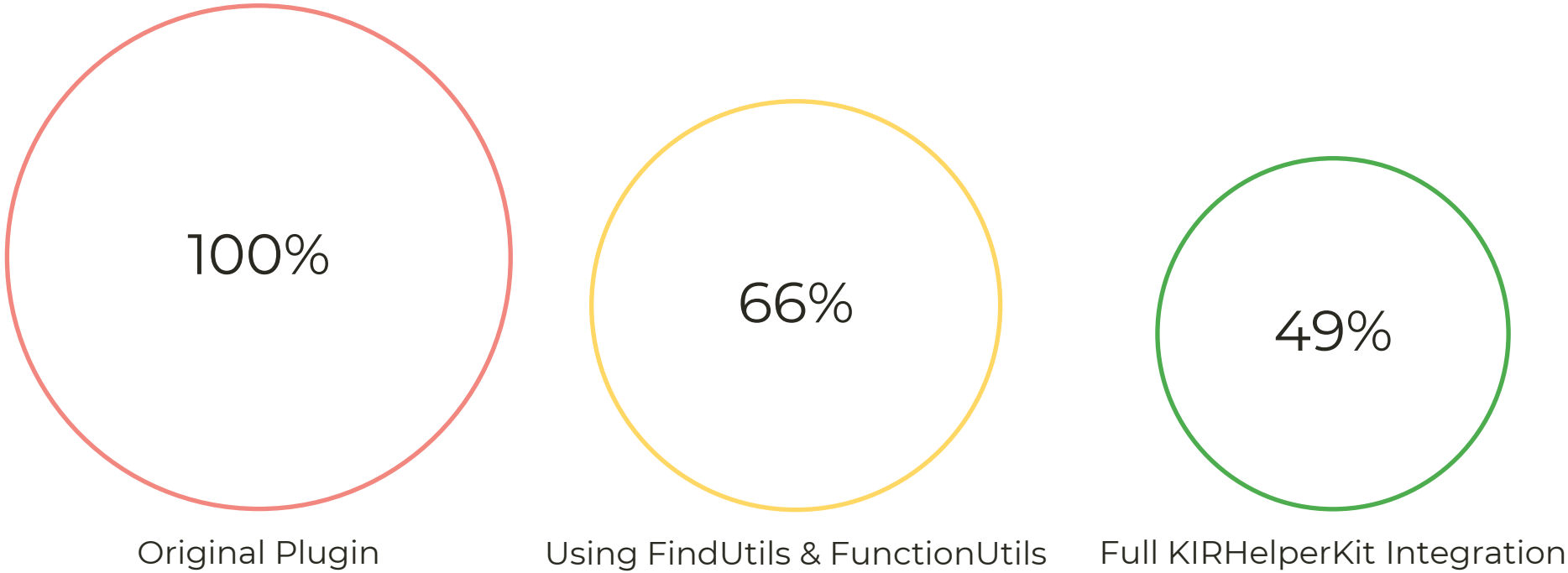
Results: Code Reduction



Original Plugin

Using FindUtils & FunctionUtils

Results: Code Reduction

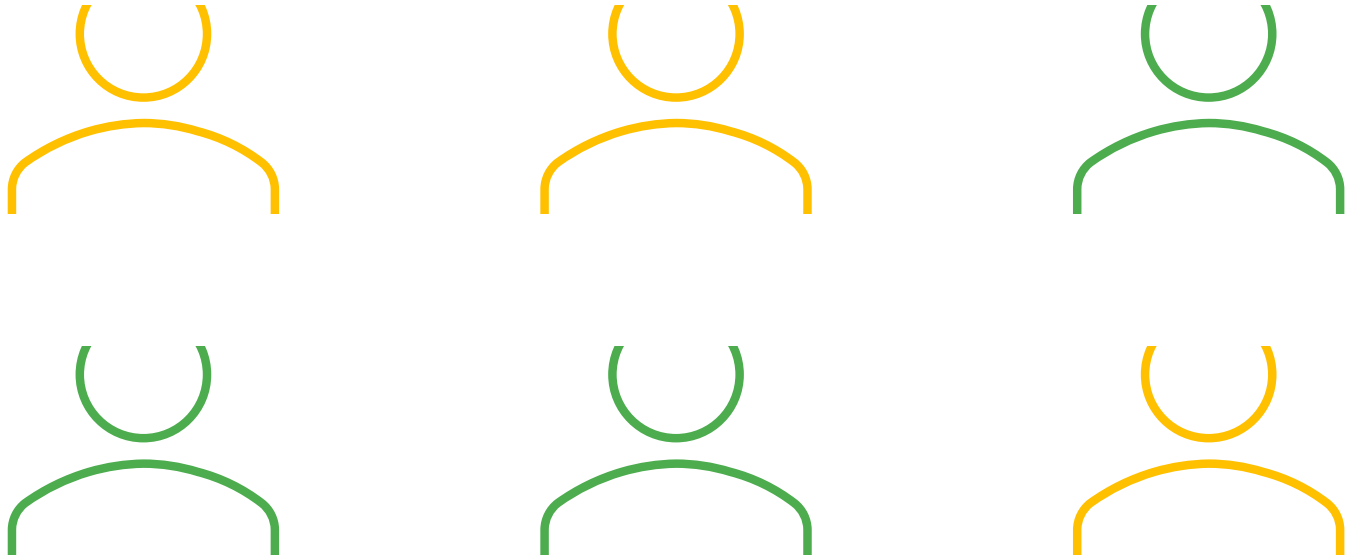


Original Plugin

Using FindUtils & FunctionUtils

Full KIRHelperKit Integration

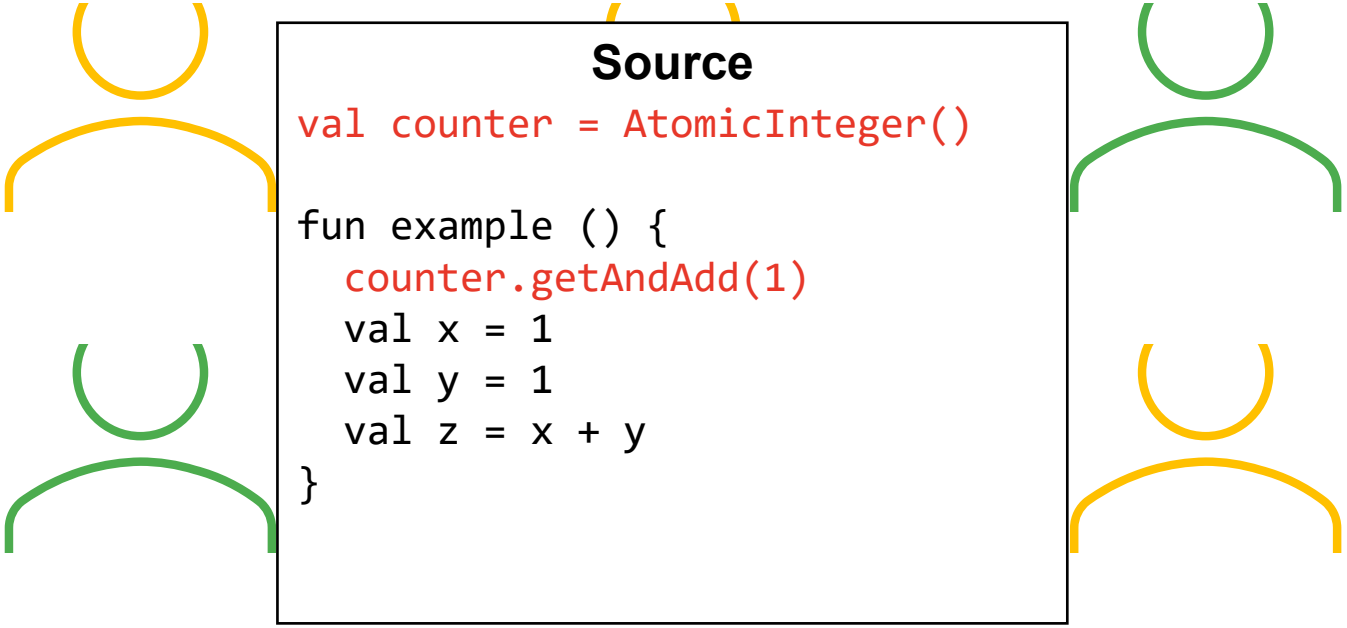
Results: User Study



With IR Experience

Without IR Experience

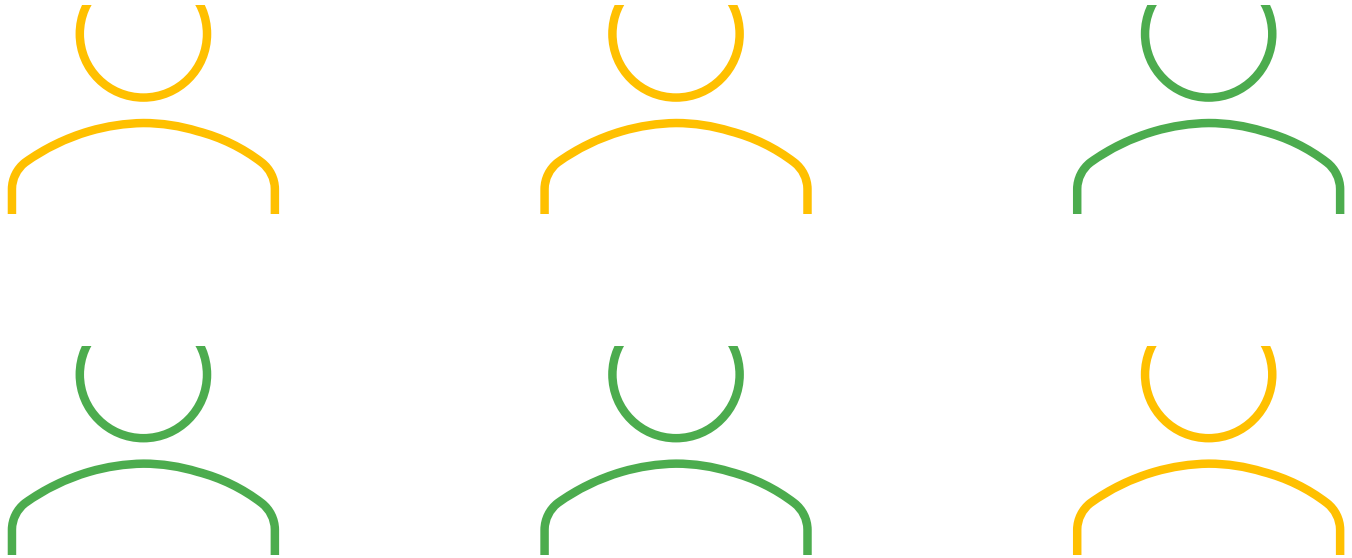
Results: User Study



With IR Experience

Without IR Experience

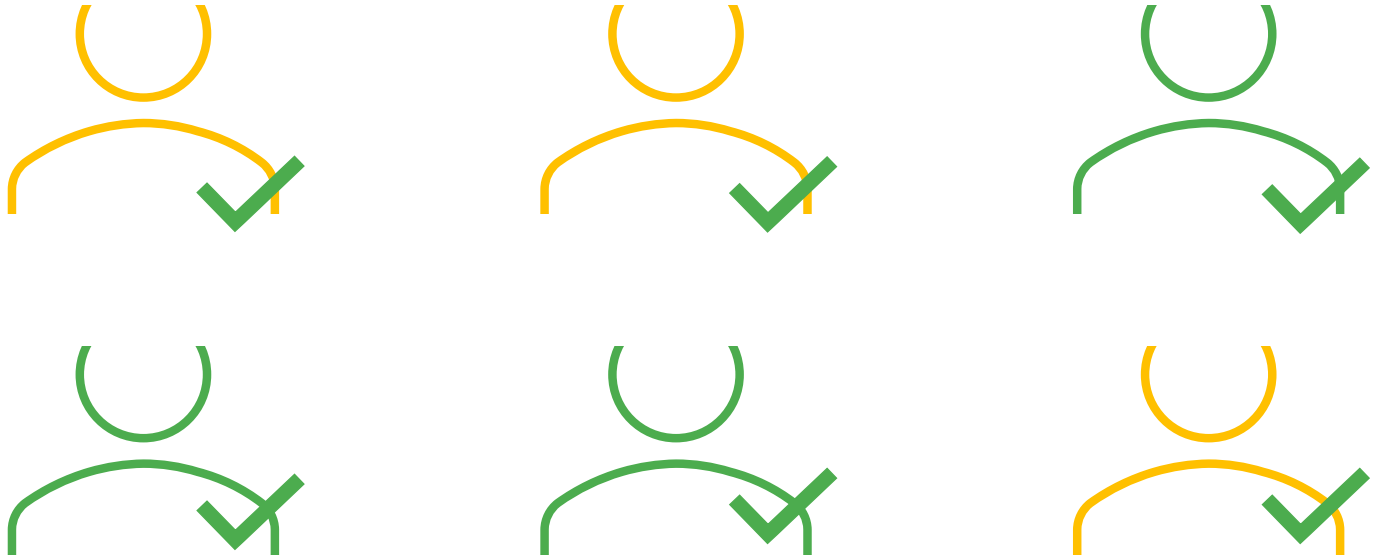
Results: User Study



With IR Experience

Without IR Experience

Results: User Study

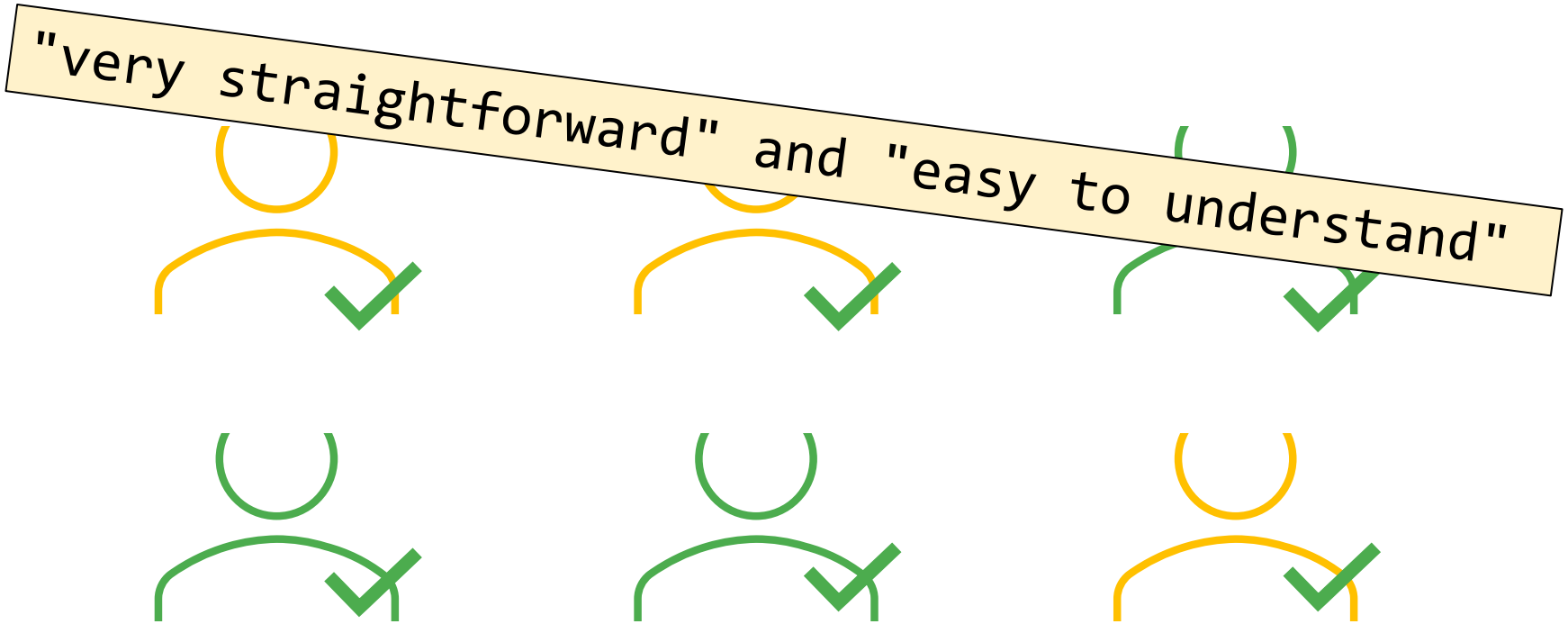


2:17 - 10:30 minutes

With IR Experience

Without IR Experience

Results: User Study

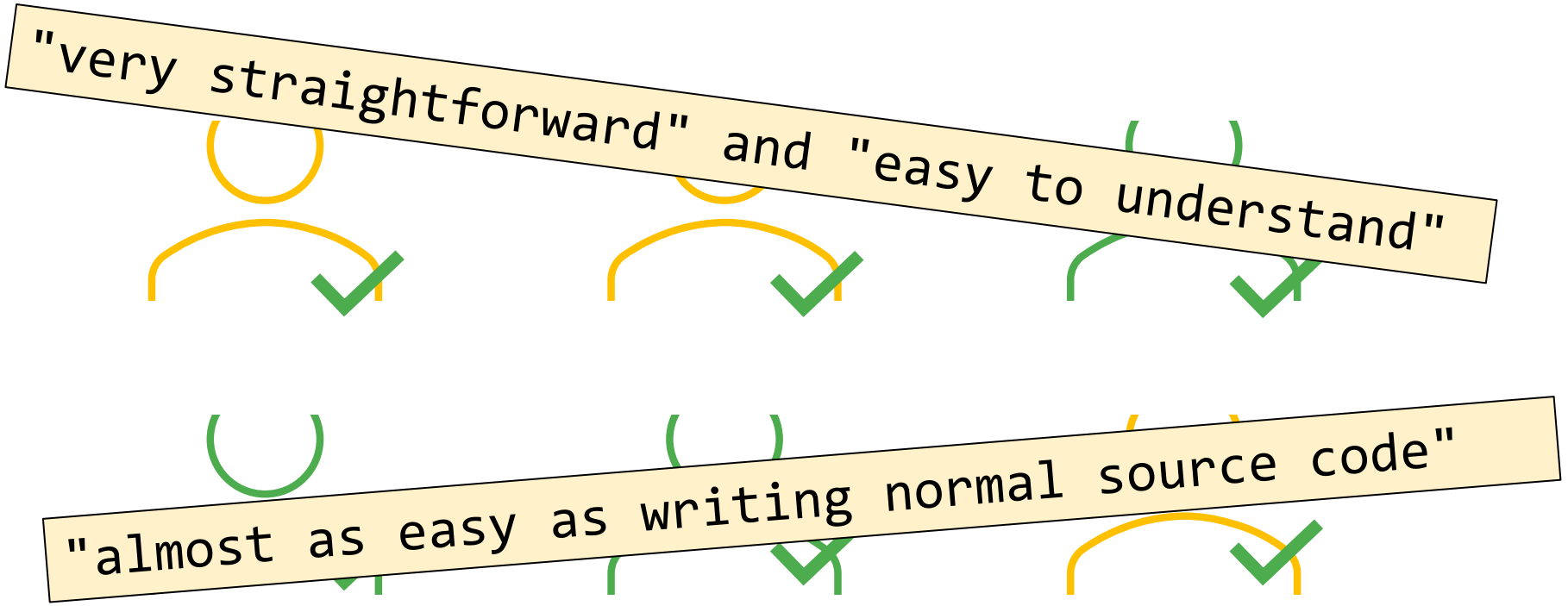


2:17 - 10:30 minutes

With IR Experience

Without IR Experience

Results: User Study



2:17 - 10:30 minutes

With IR Experience

Without IR Experience

User Study / Future Work

User Study / Future Work

Aspect-oriented plugin design

User Study / Future Work

- Aspect-oriented plugin design
- More features for the general utilities

User Study / Future Work

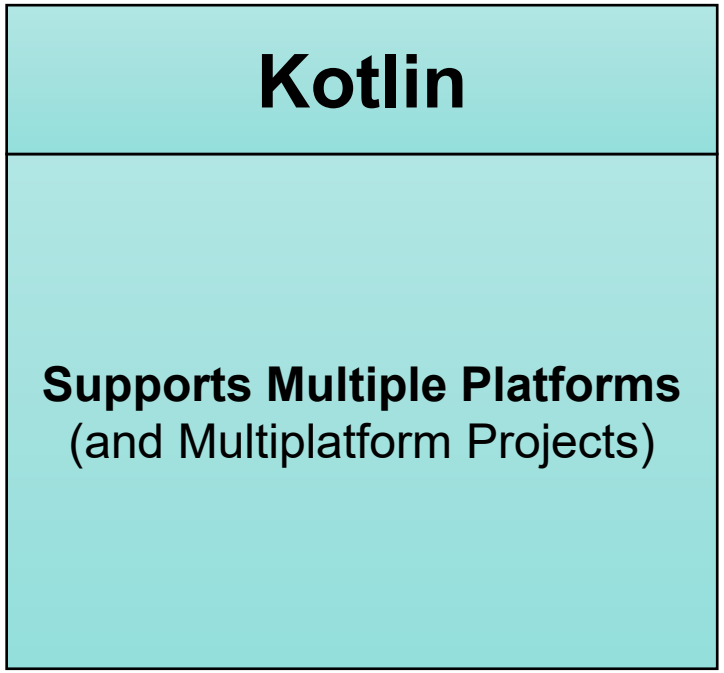
- Aspect-oriented plugin design
- More features for the general utilities
- Analyze other compiler plugins

User Study / Future Work

- Aspect-oriented plugin design
- More features for the general utilities
- Analyze other compiler plugins
- Student / developer feedback

Summary

Summary


Kotlin
Supports Multiple Platforms
(and Multiplatform Projects)

Summary

Kotlin	Kotlin Plugins
Supports Multiple Platforms (and Multiplatform Projects)	Extremly Versatile (but sadly badly documented) **Common Instrumentation for Multiple Targets using a Single Plugin**

Summary

Kotlin	Kotlin Plugins	KIRHelperKit
Supports Multiple Platforms (and Multiplatform Projects)	Extremly Versatile (but sadly badly documented) **Common Instrumentation for Multiple Targets using a Single Plugin**	Simplifies IR Plugin Development **Unified Search API** **Function Call DSL** **Utility Functions**

Summary

Kotlin

Supports Multiple Platforms
(and Multiplatform Projects)

Kotlin Plugins

Extremly Versatile (but sadly badly documented)

Common Instrumentation for Multiple Targets using a Single Plugin

KIRHelperKit

Simplifies IR Plugin Development

Unified Search API
Function Call DSL
Utility Functions

Lorenz Bader
Lorenz.Bader2001@gmail.com

Johannes Kepler University Linz

Markus Weninger
markus.weninger@jku.at

Institute for System Software
Johannes Kepler University Linz

JOHANNES KEPLER UNIVERSITÄT LINZ

JOHANNES KEPLER
UNIVERSITÄT LINZ