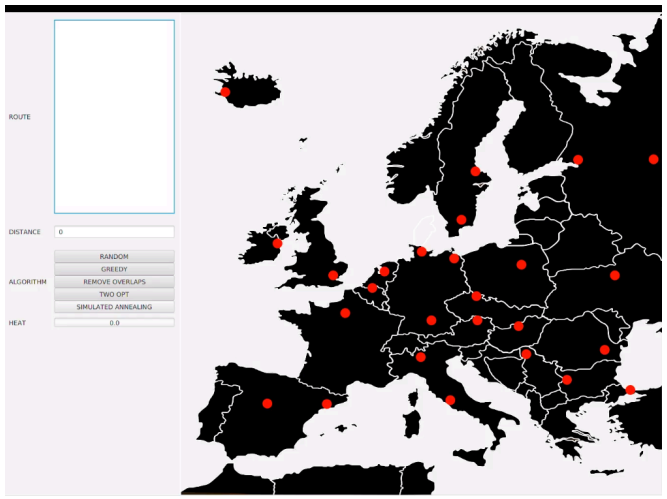
KotlinConf
Amsterdam
3-5 Oct 2018

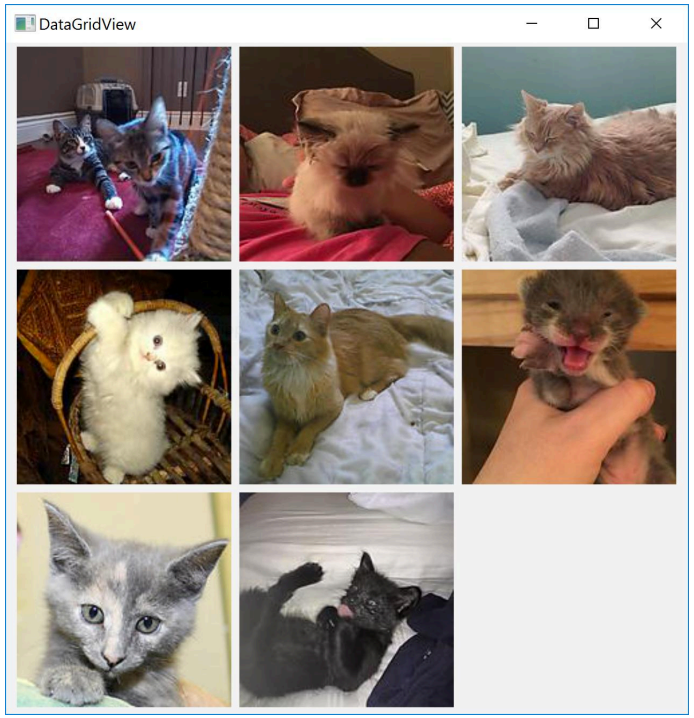
Kotlin: The Next Frontier for Modern (Meta) Programming

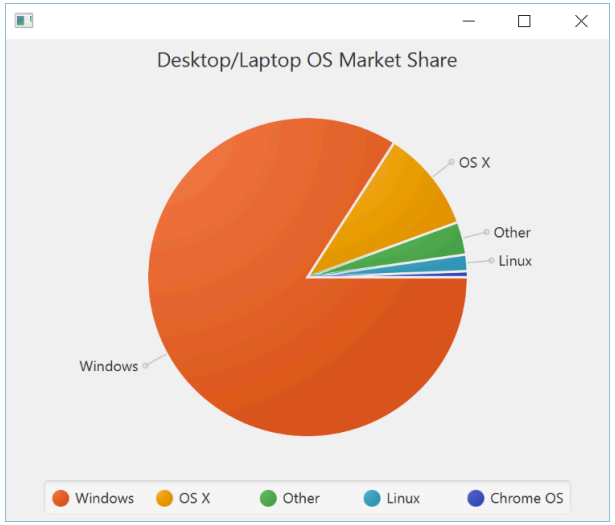
Kotlin, TornadoFX and Metaprogramming

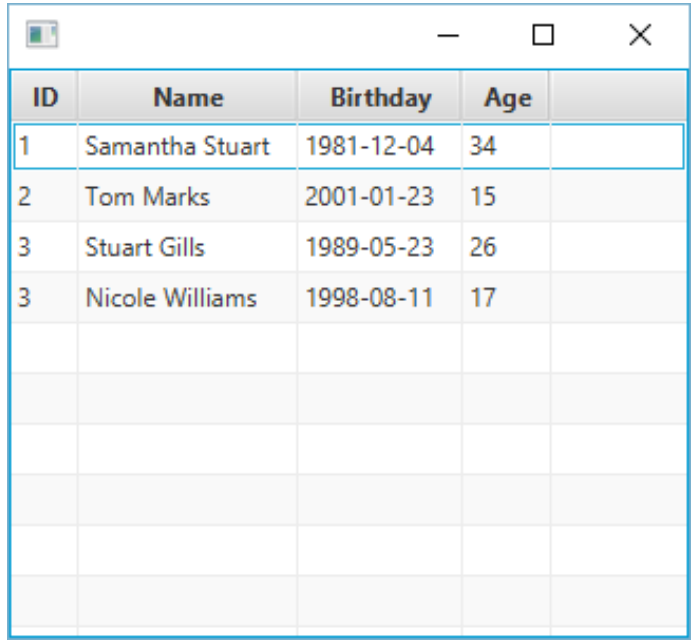
Amanda Hinchman-Dominguez
@hinchman_amanda



Thomas Nield





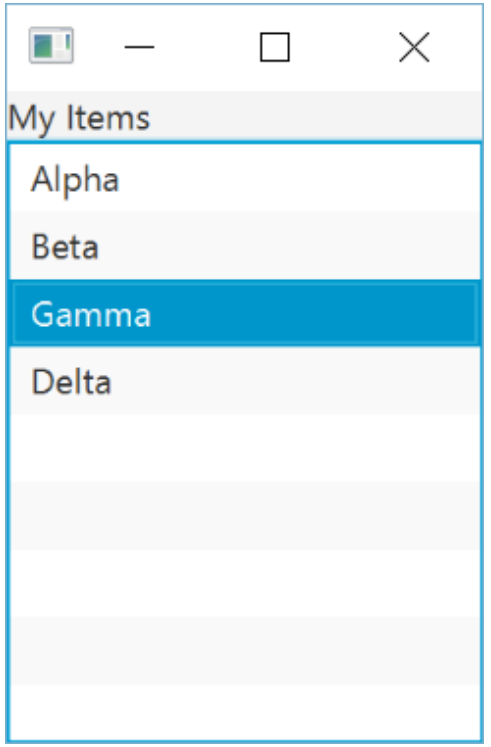


ID	Name	Birthday	Age
1	Samantha Stuart	1981-12-04	34
2	Tom Marks	2001-01-23	15
3	Stuart Gills	1989-05-23	26
3	Nicole Williams	1998-08-11	17

TornadoFX

a JavaFX Framework written in Kotlin

```
import javafx.collections.FXCollections
import tornadofx.*


class MyView : View() {
    val controller: MyController by inject()

    override val root = vbox {
        label("My items")
        listview(controller.values)
    }
}

class MyController: Controller() {
    val values = FXCollections.observableArrayList("Alpha","Beta","Gamma","Delta")
}
```

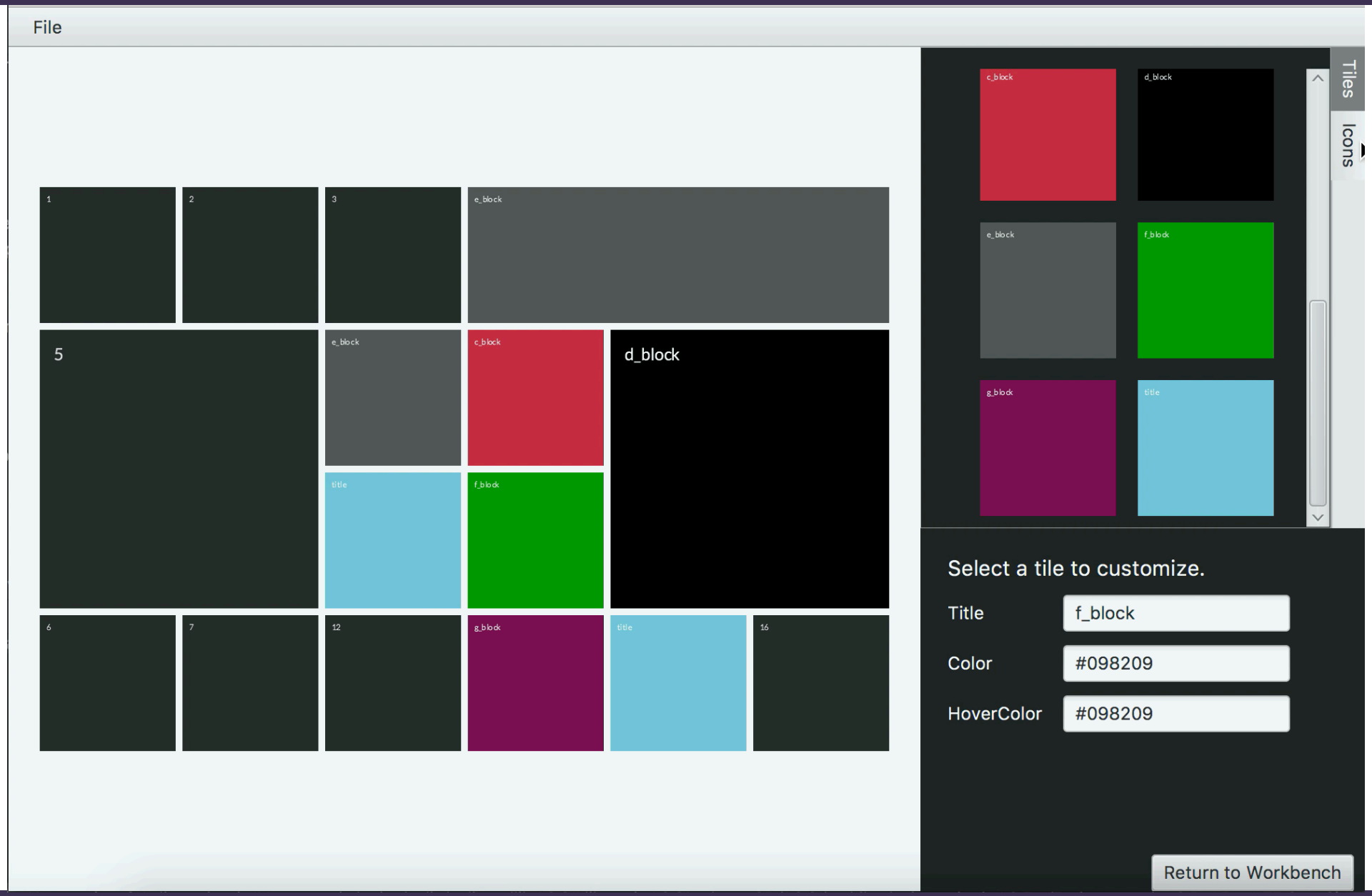


Characteristics of Functional Languages

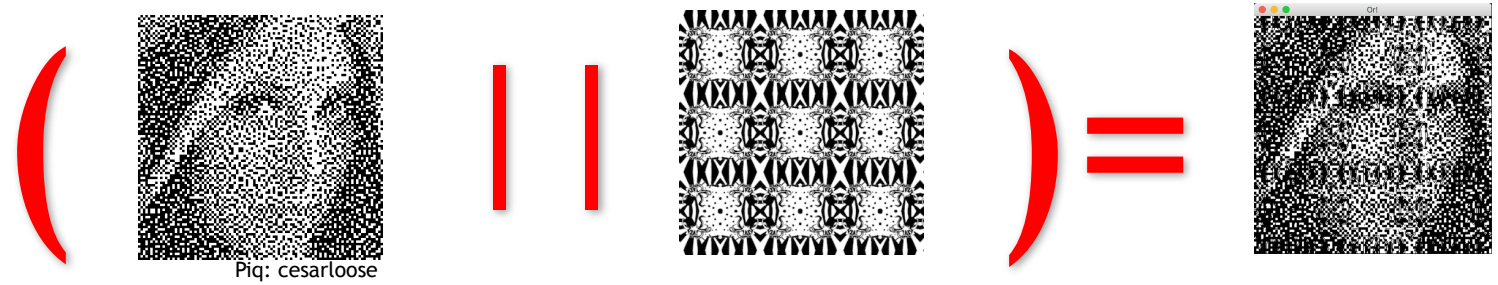
- Functions are treated as first class citizens
- Functions have no side-effects
- Metaprogramming capabilities

Metaprogramming is magic!!

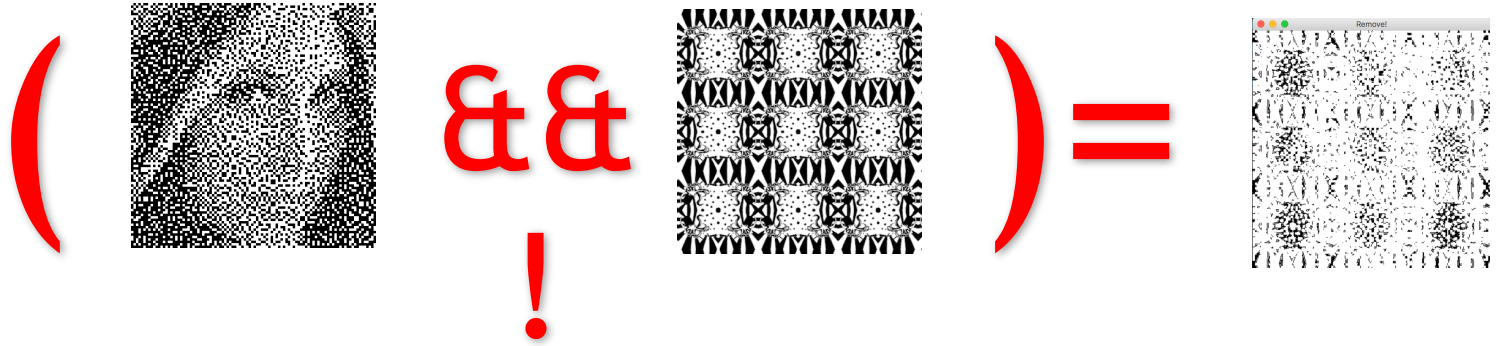
- **Programs that treat programs as data – read, analyze, or transform other programs**
- Metaprogramming can be used to make our lives easier


File
1
2
3
e_block
5
e_block
c_block
d_block
title
f_block
6
7
12
g_block
title
16
c_block
d_block
e_block
f_block
g_block
title
Tiles
Icons
Select a tile to customize.
Title
f_block
Color
#098209
HoverColor
#098209
Return to Workbench

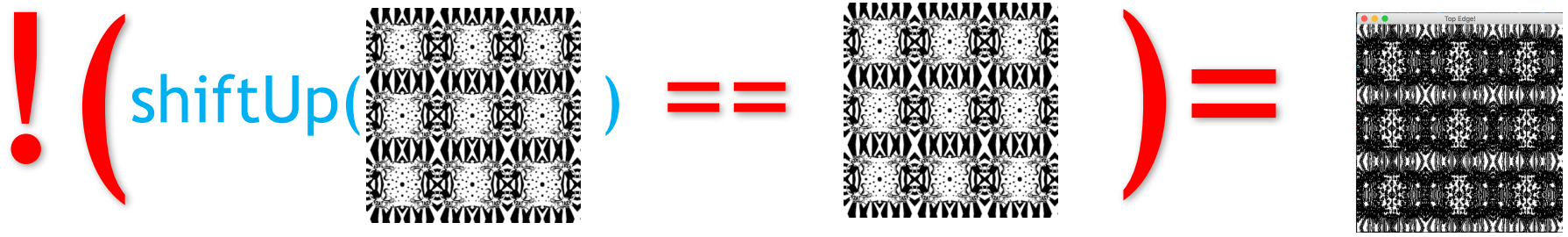
Primitive Filters



or(a, b)

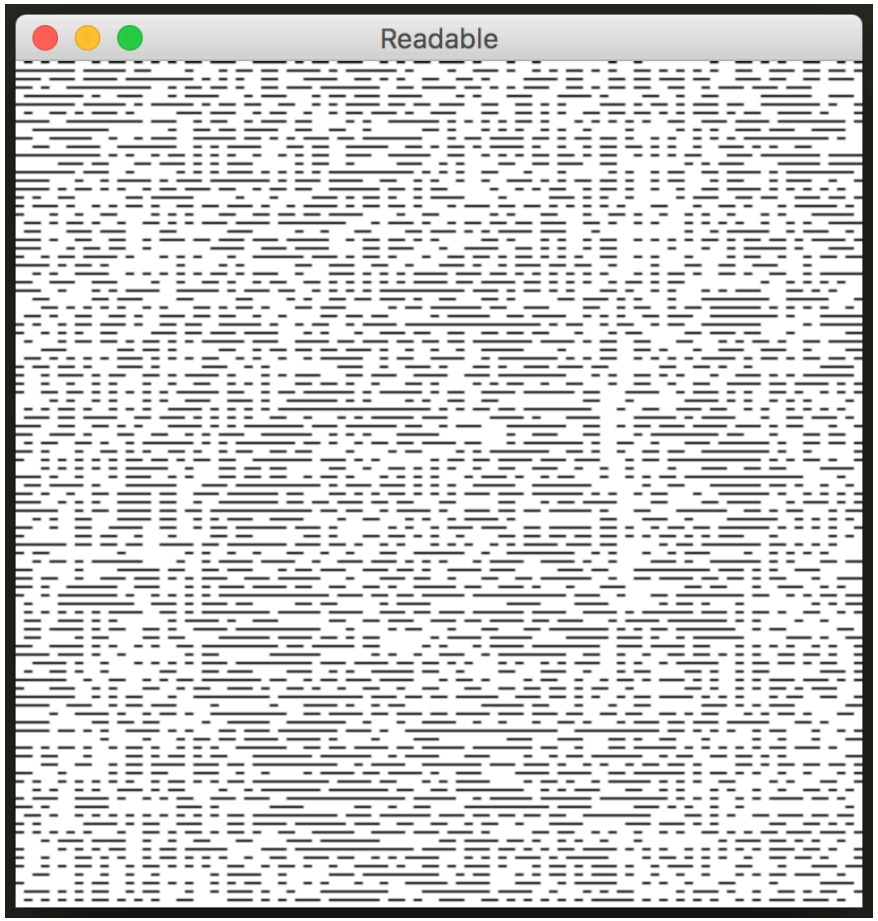


remove(a, b)



topEdge(a)

A READABLE Horizontal Edge



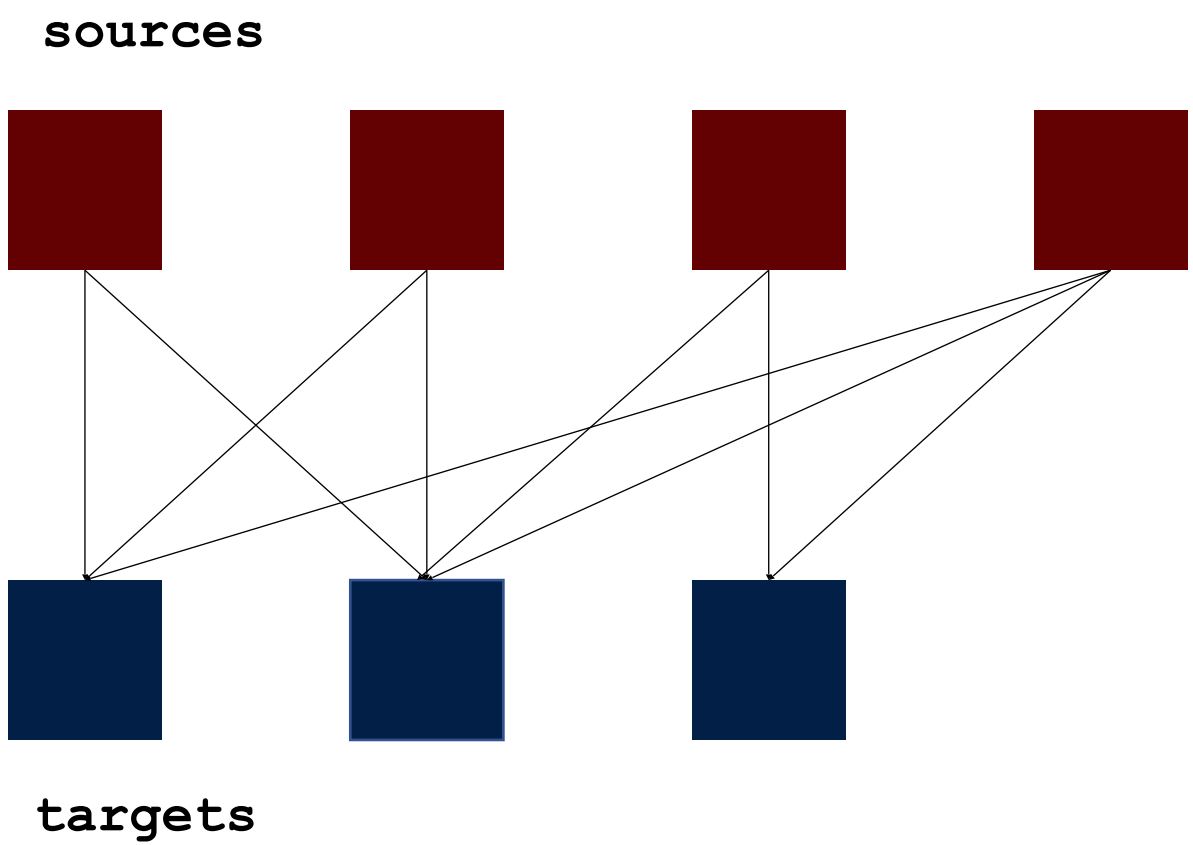

```
private fun horizontalEdge(image: PixelReader): WritableImage {
    val topEdge = topEdge(image).pixelReader
    val bottomEdge = bottomEdge(image).pixelReader
    return orFilter(topEdge, bottomEdge)
}
```

A MEMORY-OPTIMAL Horizontal Edge

```
private fun horizontalEdge(image: PixelReader): WritableImage {
    val a = topEdge(image).pixelReader
    val b = bottomEdge(image).pixelReader
    for (x in 0 until width) {
        for (y in 0 until height) {
            resultWriter.setColor(x, y,
                    (or (and(a.getColor(x, y),
                            not(a.getColor(x, y))),
                        (and(b.getColor(x,y),
                            not(b.getColor(x,y)))))))
        }
    }
    return result
}
```

Crosscutting = (Scattering && Tangling)

- **Scattering** – when a source element is related to multiple target elements
- **Tangling** – when a target element is related to multiple source elements
- **Crosscutting occurs if there exists both scattering & tangling**
  - "spaghetti code"



Metaprogramming
A solution to crosscutting

- We sacrifice one domain design decision for another
- Metaprogramming comes in different forms
  - **Wizards** - “Monkey patching”
  - **Aspect-Oriented Programming** (AOP)
  - **Domain Specific Languages** (DSLs)

Flickr: Susan Lucas Hhoffman

[app] ~/repos/kotlin-android-avocado-facts/app/src/main/res/layout/activity_main.xml

Palette

Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy

Ab TextView
Button
ImageView
RecyclerView
<fragment>
ScrollView
Switch

Component Tree

RelativeLayout
avocadoFactScroll (Scrc
Ab avocadoFact (TextVie
avocadoIcon (ImageView
Ab avocado_title (TextView)

Nexus 4 | 28 | AppTheme | Default (en-us) | 74%

8:00

Avocado fact of the day:

Astronauts do not eat avocados while in space, this is partially because avocados do not have efficient rockets to reach the International Space Program.

Avocado fact of the day:

124

-102

Astronauts do not eat avocados while in space, this is partially because avocados do not have efficient rockets to reach the International Space Program.

ImageView

Attributes

id	avocadoFact
layout_width	match_parent
layout_height	166dp
Layout_Margir	[?, ?, ?, ?, ?]
Padding	[?, ?, ?, ?, ?]
Theme	
elevation	
text	@string/avocadoFac
textColor	#222222
textSize	20sp
accessibilityLi	
accessibilityTr	
accessibilityTr	
allowUndo	
alpha	
autoLink	[]
autoSizeMaxT	
autoSizeMinTe	
autoSizePrese	
autoSizeStepG	
autoSizeTextT	
autoText	
autofillHints	
background	
backgroundTi	
backgroundTi	
breakStrategy	
bufferType	
capitalize	
clickable	
contentDescri	
contextClickal	

Design | Text

1: Project | 7: Structure | Captures | 2: Favorites | Build Variants | Gradle | Device File Explorer

activity_main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">

    <ScrollView
        android:id="@+id/avocadoFactScroll"
        android:layout_width="wrap_content"
        android:layout_height="155dp"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="124dp"
        android:paddingHorizontal="50dp"
        android:scrollbarStyle="outsideOverlay">

        <TextView
            android:id="@+id/avocadoFact"
            android:layout_width="match_parent"
            android:layout_height="166dp"
            android:text="Astronauts do not eat avocados while in space, this is ..."
            android:textColor="#222222"
            android:textSize="20sp" />
    </ScrollView>

    <ImageView
        android:id="@+id/avocadoIcon"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true"
        android:contentDescription="Avocado guy"
        android:paddingBottom="0dp"
        android:visibility="visible"
        app:srcCompat="@drawable/avocado" />

    <TextView
        android:id="@+id/avocado_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

Aspect-Oriented Programming (AOP)

- AOP is how Java approaches crosscutting
- Draws out a higher level of abstraction to act as a weaver between concerns

Flickr: Chris Radcliffe

Memoization - optimization technique

```
class Memoize<in A, out B>(private val f: (A) -> B) : (A) -> B {
    private val values = mutableMapOf<A, B>()
    override fun invoke(x: A): B {
        return values.getOrPut(x) { f(x) }
    }
}

private fun <A, B> ((A) -> B).memoize(): (A) -> B = Memoize( f: this)

private val memoizedHorizontalEdge = { x: PixelReader -> horizontalEdge(x) }.memoize()
```

Java Annotations API

```
public class PlayerStats extends Fragment {
  ...
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    UUID player = (UUID) getArguments().getPlayerId(PLAYER_ID);
    mPlayer = PlayerRepo.get(getActivity()).getPlayer(playerId);
  }
  ...
}
```

How does Java achieve AOP?

1. Declare the **@interface**
2. Specify the **@Retention** and its targets
3. Add elements

```
package test.annotation.example;

import java.lang.annotation.*;
import java.lang.reflect.*;
import java.lang.util.*;

@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD})
public @interface Transaction {
    String description() default "";
    String transaction() default "";
    String return();
}
```

How does Java achieve AOP?

- Dynamic introspection prime for reflection
- Runtime annotation carries through to the JVM

```
public class BankTransactions {
    @Logging(transactionType = "withdrawal")
    public void withdrawal(int accountId, float amount, Date time) {
        // remove money from an account
    }

    @Logging(transactionType = "deposit")
    public void deposit(int accountId, float amount, Date time) {
        // deposit money into an account
    }

    @Logging(transactionType = "transfer")
    public void transfer(int accountId, int secondaryAccountId,
                         float amount, Date time) {
        // transfer money into another account
    }
}
```

Current Shortcomings in Aspect-Oriented Programming

- Annotations are only a form of metadata
  - No direct effect on the annotated code
  - Annotation processing occurs only at designated annotations
  - No dynamic generation for elements
- True metaprogramming doesn’t play nicely with static types
  - Generating code without compiling risks type-checking
  - True metaprogramming does not respect encapsulation
- Statically-typed languages overcomplicate

Imperative v. Declarative Programming

Salting your food

- **An imperative approach (HOW)**: I see the salt is out of my reach. If I reach over the beans and the sauce and avoiding knocking the drinks over I could probably get it without brushing over my own food.
- **A declarative approach (WHAT)**: Pass the salt please.
  - **Execution is more efficient**
  - **Reduced bugs**

Flickr: Joe King

Domain-Specific Languages (DSLs)

- Regular DSLs
  - Cares about a specific concern and nothing else
  - Not a general purpose language & difficult to combine with
- Kotlin creates internal DSLs
  - Uses its own language
  - Retains key advantages of DSLs

Type-Safe CSS

- Blocks, or { }, can be passed when the terminal parameter is a function

  op -> ()

- Type-safe builders allows Kotlin-based DSLs to create complex hierarchies in a declarative way

```
class Styles : Stylesheet() {
    // holds class-level properties that can easily be retrieved
    companion object {
        val loginScreen by cssclass()
        val workbenchScreen by cssclass()
        val tileGUI by cssclass()
        val grid by cssclass()
    }

    // apply styling to classes
    init {

        select(loginScreen) { this: CssSelectionBlock
            padding = box(15.px)
            vgap = 7.px
            hgap = 10.px
        }

        workbenchScreen { this: CssSelectionBlock
            backgroundColor += Color.rgb( red: 34,  green: 34,  blue: 34)
        }

        tileGUI { this: CssSelectionBlock
            backgroundColor += Color.rgb( red: 34,  green: 34,  blue: 34)
        }

        grid { this: CssSelectionBlock
            backgroundColor += Color.WHITE
        }
```

Higher Order Functions and Lambdas

Function as a Parameter

```
operator fun Selectable.invoke(op: CssSelectionBlock.() -> Unit): CssSelection {
    val selection = CssSelection(toSelection(), op)
    addSelection(selection)
    return selection
}
```

```
op: CssSelectionBlock.() -> Unit
```

Higher Order Functions and Lambdas

Function as a Parameter

```
operator fun Selectable.invoke(op: CssSelectionBlock.() -> Unit): CssSelection {
    val selection = CssSelection(toSelection(), op)
    addSelection(selection)
    return selection
}
```

Return type

```
op: CssSelectionBlock.() -> Unit
```

Receiver type

Parameter types

Higher Order Functions and Lambdas

Function as a Parameter

```
operator fun Selectable.invoke(op: CssSelectionBlock.() -> Unit): CssSelection {
    val selection = CssSelection(toSelection(), op)
    addSelection(selection)
    return selection
}
```

A.(B) ⟶ C

Return type

op: CssSelectionBlock.() -> Unit

Receiver type

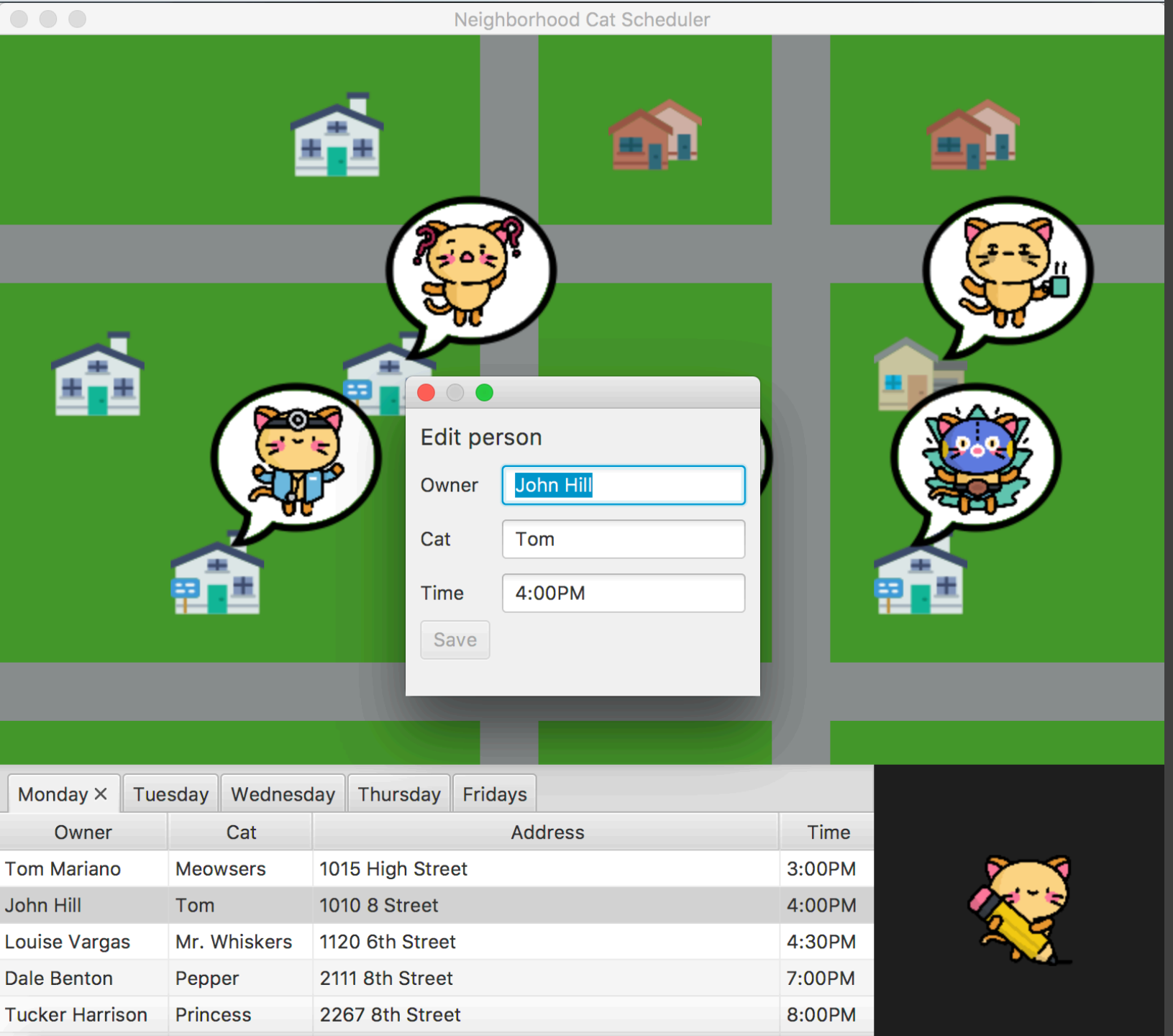
Parameter types

or

(A, B) ⟶ C

Reified Generics

- Function bytecode copies into every place where the function is called
- When the function is called the compiler **modifies the generated bytecode to use the corresponding class directly**
- Can be used
  - Type checks/casts
  - Use the Kotlin reflection API
  - As a type argument to call other functions



Scopes

- simple constructs used to pass objects like models to other classes
- Makes the View/ Controller unique to a smaller subset of instances in an application

Passing Scopes to Components

```
@JvmStatic
@JvmOverloads
fun <T : Component> find(componentType: Class<T>, scope: Scope = DefaultScope): T = find(componentType.kotlin, scope)

inline fun <reified T : Component> find(scope: Scope = DefaultScope): T = find(T::class, scope)
```

Generic-Class-as-a-Parameter Pattern

```
@JvmStatic
@JvmOverloads
fun <T : Component> find(componentType: Class<T>, scope: Scope = DefaultScope): T = find(componentType.kotlin, scope)
```

Type parameter used in receiver and return types

Type parameter declaration - any Component may be passed

Passing Scopes with Reified Generics

“reified” declares that this type parameter will not be erased at runtime

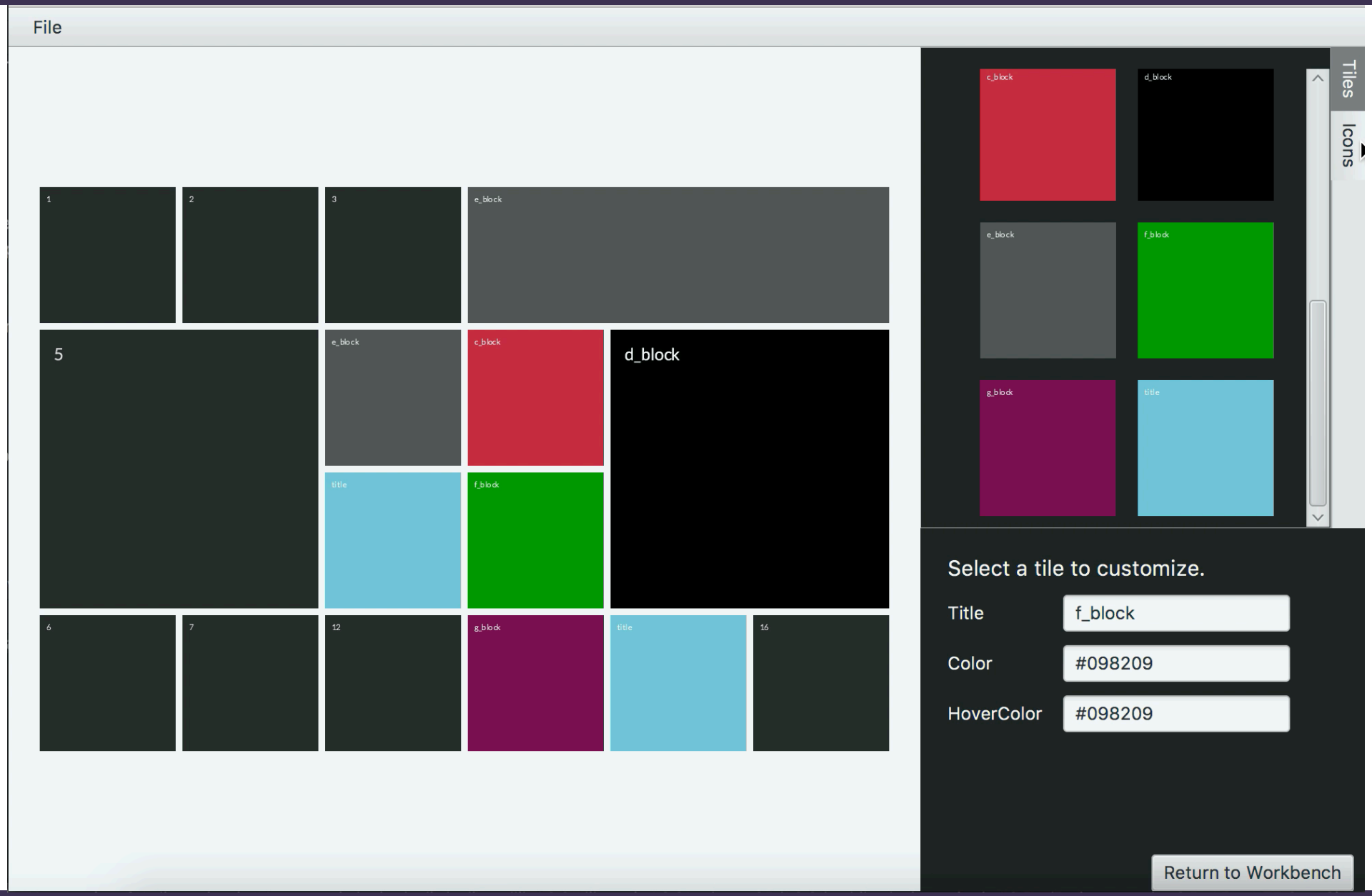
```
inline fun <reified T : Component> find(scope: Scope = DefaultScope): T = find(T::class, scope)
```

Passing Scopes to Components

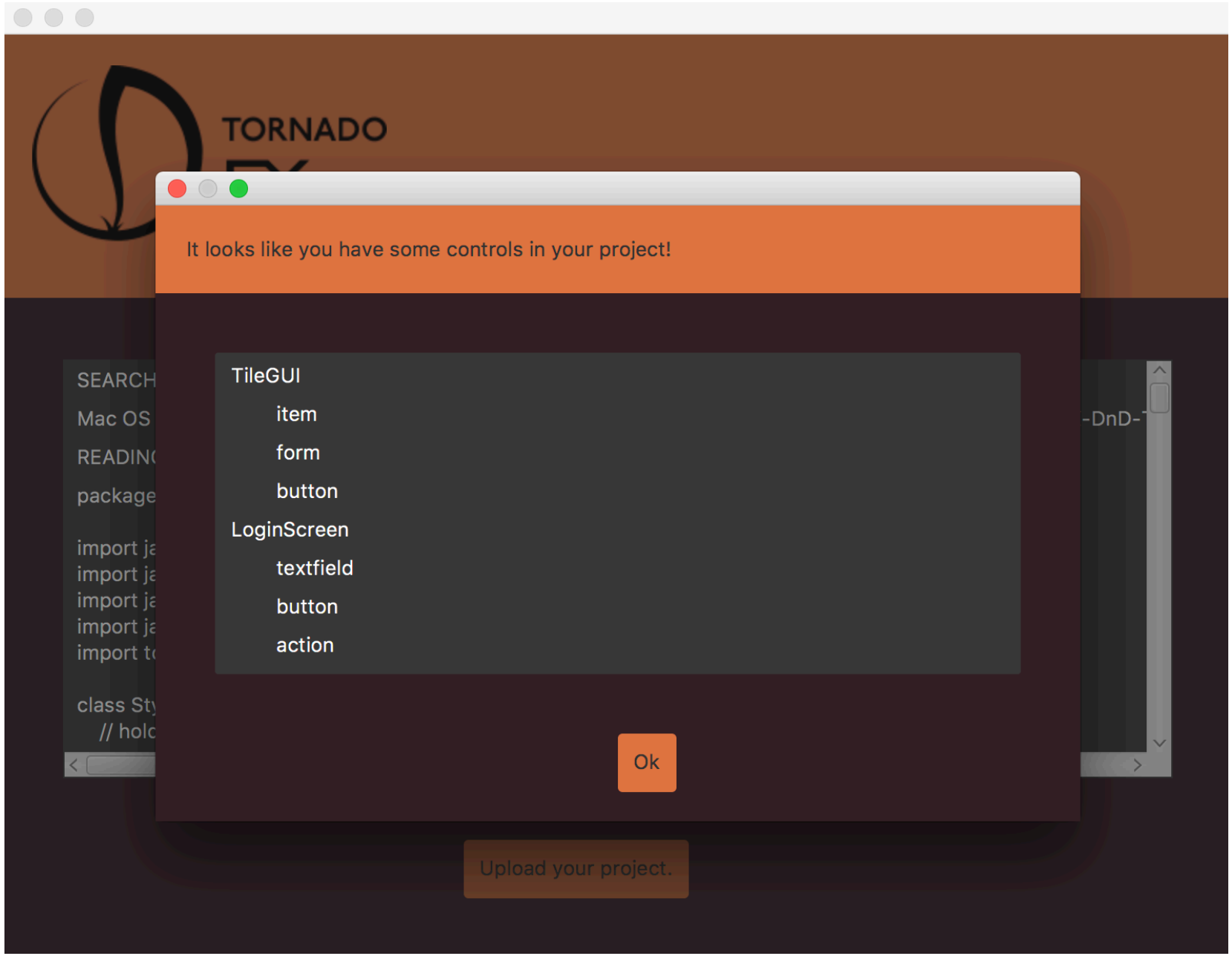
```
@JvmStatic
@JvmOverloads
fun <T : Component> find(componentType: Class<T>, scope: Scope = DefaultScope): T = find(componentType.kotlin, scope)

inline fun <reified T : Component> find(scope: Scope = DefaultScope): T = find(T::class, scope)
```

The Takeaway

- Metaprogramming is a solution to crosscutting
- Common forms of metaprogramming has their shortcomings
- TornadoFX leverages functional Kotlin features to streamline JavaFX

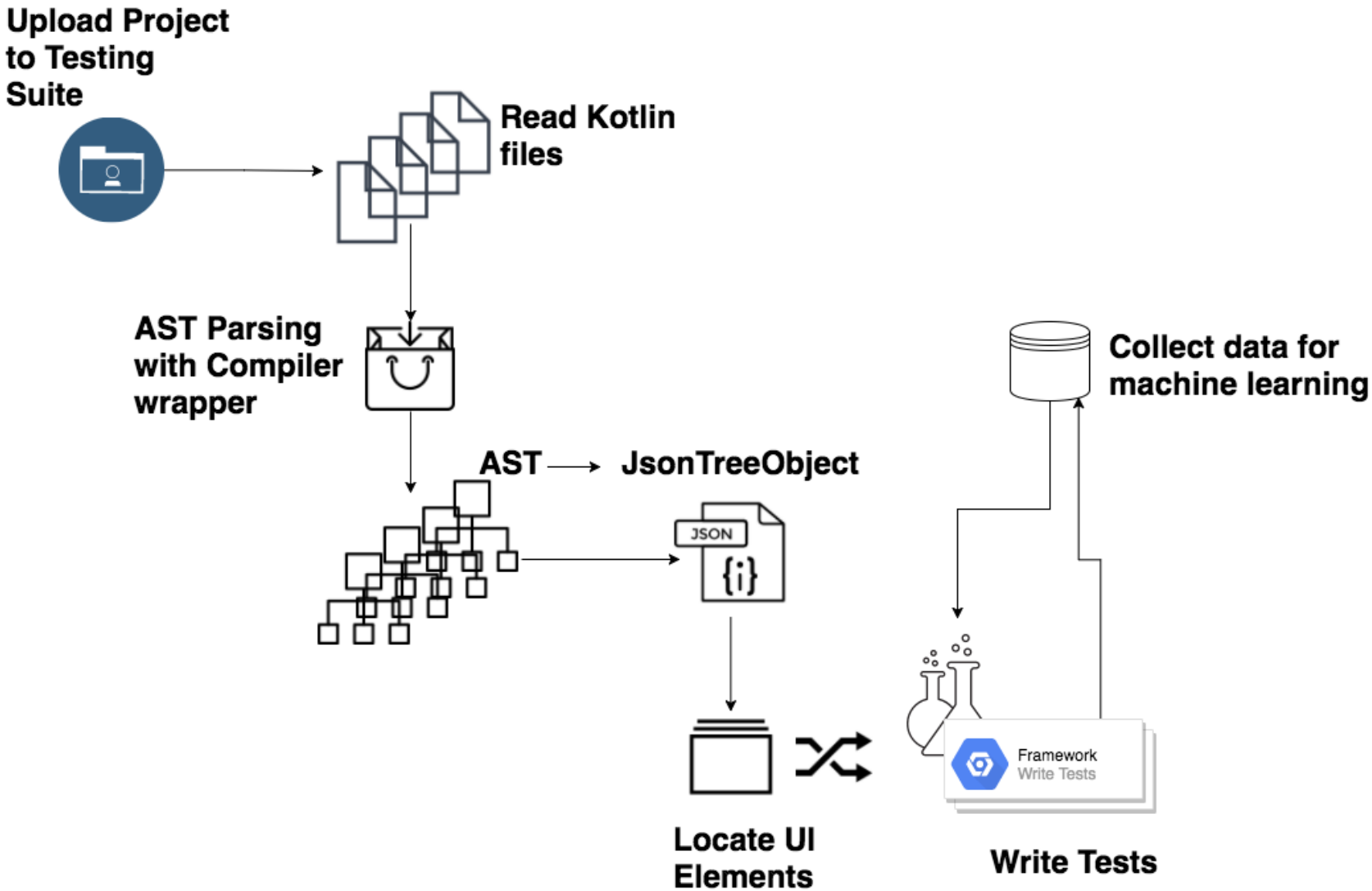

File
1
2
3
e_block
5
e_block
c_block
d_block
title
f_block
6
7
12
g_block
title
16
c_block
d_block
e_block
f_block
g_block
title
Tiles
Icons
Select a tile to customize.
Title
f_block
Color
#098209
HoverColor
#098209
Return to Workbench

TORNADO FX SUITE



Challenges w/ Statically-Typed Metaprogramming

- True reflection does not respect encapsulation
- Generating code at runtime means no compiler for type-checking


Upload Project to Testing Suite
Read Kotlin files
AST Parsing with Compiler wrapper
AST
JsonTreeObject
JSON
{i}
Collect data for machine learning
Framework
Write Tests
Locate UI Elements
Write Tests

Special Thanks

Carl Walker
@bekwaminc

Edvin Syse
@edvinsyse

Thomas Neild
@thomasneild9727

Ruckus T Boom
@ruckustboom

- #TornadoFX channel on Slack
- Kotlin community
- KotlinConf Committee
- Ebad Ahmadzadeh