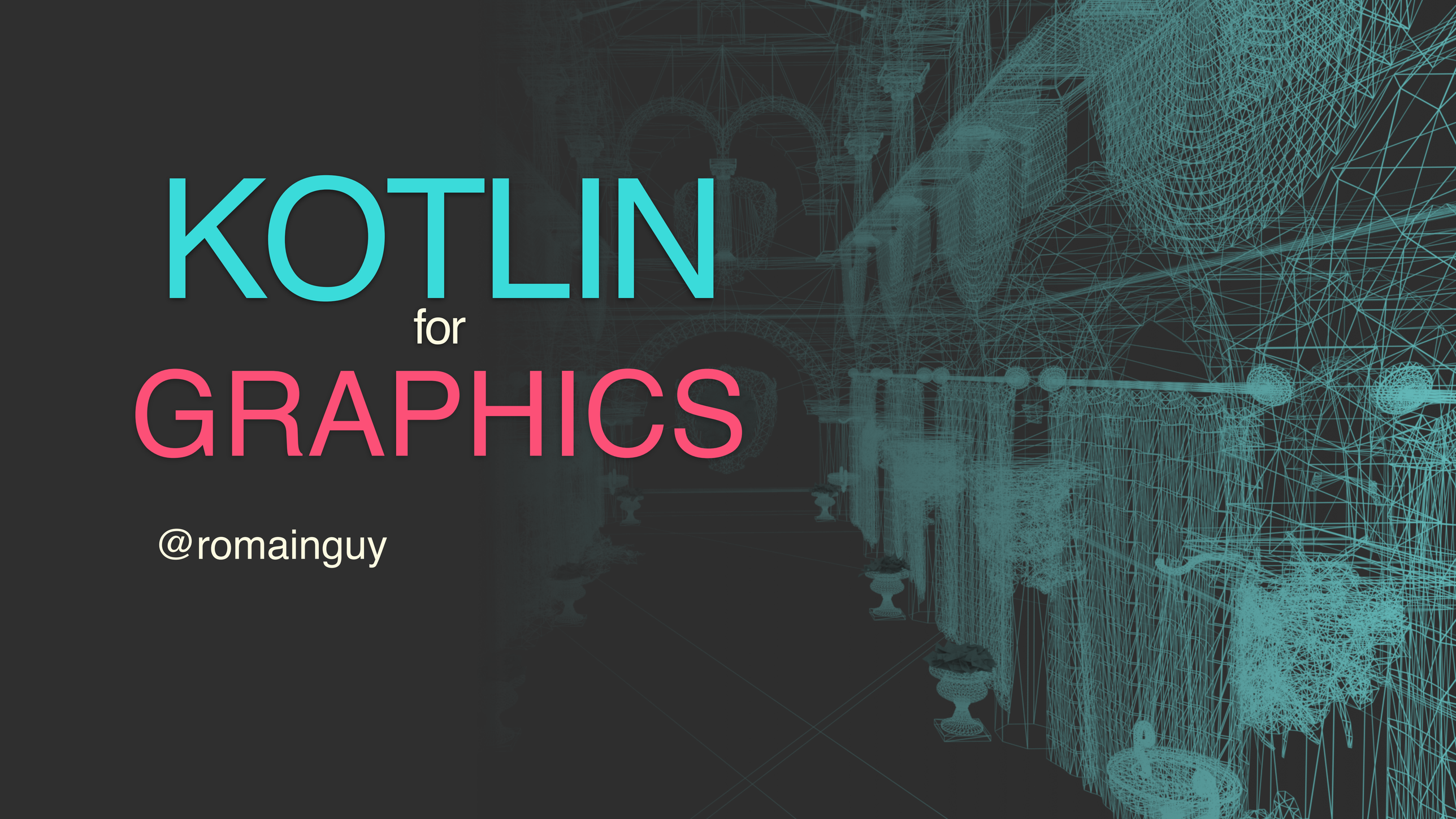
KOTLIN for GRAPHICS

@romainguy



Android KTX

Dealing with legacy

```
// Pack a color int
val color =
    ((a and 0xff) shl 24) or
    ((r and 0xff) shl 16) or
    ((g and 0xff) shl  8) or
    ((b and 0xff)       )

// Unpack a color int
val a = (color shr 24) and 0xff
val r = (color shr 16) and 0xff
val g = (color shr  8) and 0xff
val b = (color       ) and 0xff
```

```
inline val @receiver:ColorInt Int.alpha get() = (this shr 24) and 0xff
inline val @receiver:ColorInt Int.red   get() = (this shr 16) and 0xff
inline val @receiver:ColorInt Int.green get() = (this shr  8) and 0xff
inline val @receiver:ColorInt Int.blue  get() = (this       ) and 0xff
```

```
val a = color.alpha
val r = color.red
val g = color.green
val b = color.blue
```

```
inline operator fun @receiver:ColorInt Int.component1() = this.alpha
inline operator fun @receiver:ColorInt Int.component2() = this.red
inline operator fun @receiver:ColorInt Int.component3() = this.green
inline operator fun @receiver:ColorInt Int.component4() = this.blue
```

```
val (a, r, g, b) = color
```

Destructure everything

```
// Points
val (x, y) = PointF(1.0f, 2.0f) + PointF(3.0f, 4.0f)

// Rectangles
val (l, t, r, b) = Rect(0, 0, 4, 4) and Rect(2, 2, 6, 6)

// Matrix
val (right, up, forward, eye) = viewMatrix
val (x, y, z) = eye
```

Implement operators

Kotlin	Arithmetic operators	Set operators
`plus`	+	∪ (union)
`minus`	−	− (difference)
`times`	×	
`div`	/	
`and`		∪ (union)
`or`		∩ (intersection)
`xor`		⊖ (symmetric difference)
`not`		U \ (complement)

```
// Intersection
val r = RectF(0.0f, 0.0f, 4.0f, 4.0f) or RectF(2.0f, 2.0f, 6.0f, 6.0f)

// Symmetric difference
val r = Rect(0, 0, 4, 4) xor Rect(2, 2, 6, 6)

// Difference
val path = circle - square

// Offset
val (l, t, r, b) = Rect(0, 0, 2, 2) + 2
val (l, t, r, b) = Rect(0, 0, 2, 2) - Point(1, 2)
```

```
inline operator fun Rect.contains(p: PointF) = contains(p.x, p.y)
```

```
if (PointF(x, y) in path.bounds) {
    // Hit detected
}
```

```
inline operator fun Bitmap.get(x: Int, y: Int) = getPixel(x, y)

inline operator fun Bitmap.set(x: Int, y: Int, @ColorInt color: Int) =
        setPixel(x, y, color)
```

```
val b = getBitmap()
val a = b[1, 1].alpha
if (a < 255) {
    b[1, 1] = 0xff_00_00_00
}
```

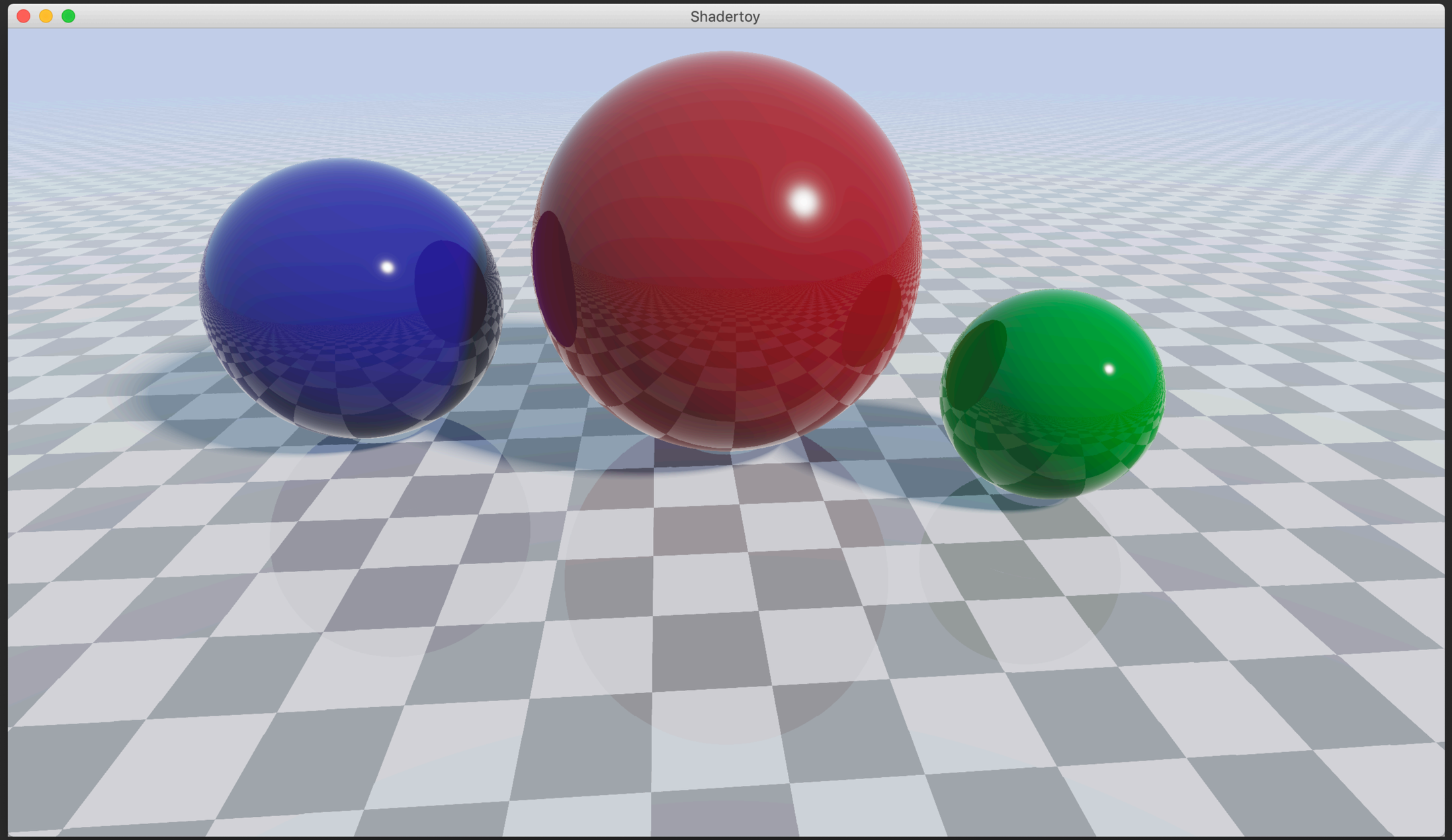
```
inline operator fun Bitmap.get(x: IntRange, y: IntRange): IntArray
```

```
bitmap[16..32, 16..32].forEach {
    val (_, r, g, b) = it
}
```

All together

```
if (bounds.contains(hit.x, hit.y)) {
    val pixel = Bitmap.createBitmap(width, height, ARGB_8888).let {
        with(Canvas(it)) {
            save()
            scale(2.0f, 2.0f)
            val path = Path().apply { op(path1, path2, DIFFERENCE) }
            drawPath(path, paint)
            restore()
        }
        it.getPixel(hit.x, hit.y)
    }
    val r = Color.red(pixel)
    val g = Color.green(pixel)
    val b = Color.blue(pixel)
}
```

```
if (hit in bounds) {
    val (_, r, g, b) = createBitmap(width, height).applyCanvas {
        withScale(2.0f, 2.0f) {
            drawPath(path1 - path2, paint)
        }
    }[hit.x, hit.y]
}
```


Shadertoy

```
vec3 color = vec3(0.65, 0.85, 1.0) + direction.y * 0.72;

// (distance, material)
vec2 hit = traceRay(origin, direction);
float distance = hit.x;
float material = hit.y;

// We've hit something in the scene
if (material > 0.0) {
    vec3 lightDir = vec3(0.6, 0.7, -0.7);
    vec3 position = origin + distance * direction;
    vec3 v = normalize(-direction);
    vec3 n = normal(position);
    vec3 l = normalize(lightDir);
    vec3 h = normalize(v + l);
    vec3 r = normalize(reflect(direction, n));

    float NoV = abs(dot(n, v)) + 1e-5;
    float NoL = saturate(dot(n, l));
    float NoH = saturate(dot(n, h));
    float LoH = saturate(dot(l, h));
    // ...
}
```

```
var color = Float3(0.65f, 0.85f, 1.0f) + direction.y * 0.72f

// (distance, material)
val hit = traceRay(origin, direction)
val distance = hit.x
val material = hit.y

// We've hit something in the scene
if (material > 0.0f) {
    val lightDir = Float3(0.6f, 0.7f, -0.7f)
    val position = origin + distance * direction
    val v = normalize(-direction)
    val n = normal(position)
    val l = normalize(lightDir)
    val h = normalize(v + l)
    val r = normalize(reflect(direction, n))

    val NoV = abs(dot(n, v)) + 1e-5f
    val NoL = saturate(dot(n, l))
    val NoH = saturate(dot(n, h))
    val LoH = saturate(dot(l, h))
    // ...
}
```

Data first

Math is functional

```
data class Float3(var x: Float = 0.0f, var y: Float = 0.0f, var z: Float = 0.0f)
```

+ operators

```
inline fun abs(v: Float3) = Float3(abs(v.x), abs(v.y), abs(v.z))
inline fun length(v: Float3) = sqrt(v.x * v.x + v.y * v.y + v.z * v.z)
inline fun length2(v: Float3) = v.x * v.x + v.y * v.y + v.z * v.z
inline fun distance(a: Float3, b: Float3) = length(a - b)
inline fun dot(a: Float3, b: Float3) = a.x * b.x + a.y * b.y + a.z * b.z
```

```
fun lookAt(
    eye:    Float3,
    target: Float3,
    up:     Float3 = Float3(z = 1.0f)
): Mat4 {
    val f = normalize(target - eye)
    val r = normalize(f x up)
    val u = normalize(r x f)
    return Mat4(r, u, f, eye)
}

val h = normalize(v + l)
val r = normalize(reflect(direction, n))

val NoV = abs(dot(n, v)) + 1e-5f
val NoL = saturate(dot(n, l))
```

Aliasing & swizzling

```
vec4 v = vec4(…)

// Use XYZ for coordinates
vec3 position = v.xyz
// Use RGB for color data
vec3 color = v.rgb

// Swizzling
vec3 reverseColor = v.bgr
vec3 grayColor = v.ggg
vec4 twoPositions = v.xyxy
```

```
// In Float4
inline var xy: Float2
    get() = Float2(x, y)
    set(value) {
        x = value.x
        y = value.y
    }

inline var rg: Float2
    get() = Float2(x, y)
    set(value) {
        x = value.x
        y = value.y
    }
```

```
enum class VectorComponent {
    X, Y, Z, W,
    R, G, B, A,
    S, T, P, Q
}

operator fun get(index: VectorComponent) = when (index) {
    VectorComponent.X, VectorComponent.R, VectorComponent.S -> x
    VectorComponent.Y, VectorComponent.G, VectorComponent.T -> y
    VectorComponent.Z, VectorComponent.B, VectorComponent.P -> z
    else -> throw IllegalArgumentException("Unknown index")
}

operator fun get(
        index1: VectorComponent,
        index2: VectorComponent,
        index3: VectorComponent
): Float3 {
    return Float3(get(index1), get(index2), get(index3))
}
```

```
val v = Float4(…)

val position = v.xyz
val color = v.rgb

val reverseColor = v[B, G, R]
val grayColor = v[G, G, G]
val twoPositions = v[X, Y, X, Y]
```

Row major

Column major

right up forward

$$\begin{bmatrix} X_1 & X_2 & X_3 & X_t \\ Y_1 & Y_2 & Y_3 & Y_t \\ Z_1 & Z_2 & Z_3 & Z_t \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

translation

```
val transform: Mat4

// 3rd column, first element
val x3 = transform[2, 0]
val x3 = transform.z.x

// Math notation
// Row-major, 1-based
val x3 = transform(1, 3)
```

Where there is one
there are many

```
if (color.r > 0.0f &&
    color.g > 0.0f &&
    color.b > 0.0f) {
    // ...
}
```

```
if (all(color gt Float3(0.0f))) {
    // ...
}

// any() for ||
if (any(color lt black)) {
    // ...
}
```

https://github.com/google/filament

Local extension functions

```
private fun createMesh() {
    data class Vertex(val x: Float, val y: Float, val z: Float, val n: Float3)

    fun ByteBuffer.put(v: Vertex): ByteBuffer {
        putFloat(v.x)
        putFloat(v.y)
        putFloat(v.z)
        v.n.forEach { putFloat(it) }
        return this
    }

    val vertexData = ByteBuffer.allocate(vertexCount * vertexSize)
            .order(ByteOrder.nativeOrder())
            // Face -Z
            .put(Vertex(-1.0f, -1.0f, -1.0f, Float3(0.0f, 0.0f, -1.0f)))
            .put(Vertex(-1.0f,  1.0f, -1.0f, Float3(0.0f, 0.0f, -1.0f)))
            // ...

    // Build mesh with vertexData
}
```

1.3

Unsigned Integers

UInt

UInt represents natural numbers

```kotlin
@ColorInt
operator fun Bitmap.get(x: Int, y: Int): Int

// Can x and y be < 0?
// Will it throw, clamp or wrap around?
myBitmap[-1, -1]
```

```
@ColorInt
operator fun Bitmap.get(x: UInt, y: UInt): Int

// No more ambiguity
myBitmap[-1, -1]
```

Conversion of signed constants to unsigned ones is

Kotlin

Signed	Byte	Short	Int	Long
Unsigned		Char	UInt	ULong

JVM/ART

Signed	byte	short	int	long
Unsigned		char		

```
val a: UInt = //...
val b: UInt = //...

val c = a + b
```

```
1: iload           4    // load a
2: iload           5    // load b
3: iadd
4: invokestatic  #13 // kotlin/UInt."constructor-impl":(I)I
5: istore_3
```

```
4: invokestatic  #13 // kotlin/UInt."constructor-impl":(I)I
```

```
public static int constructor-impl(int);
  Code:
     0: iload_0
     1: ireturn
```

```
1: iload          4    // load a
2: iload          5    // load b
3: isub
4: invokestatic  #13 // kotlin/UInt."constructor-impl":(I)I
5: istore_3
```

```
1: iload          4    // load a
2: iload          5    // load b
3: imul
4: invokestatic  #13 // kotlin/UInt."constructor-impl":(I)I
5: istore_3
```

```
1: iload          4    // load a
2: iload          5    // load b
3: invokestatic   #91  // kotlin/UnsignedKt."uintDivide-J1ME1BU":(II)I
4: invokestatic   #13  // kotlin/UInt."constructor-impl":(I)I
5: istore_3
```

UInt


UInt

Inline Classes

Inline classes are a zero-cost* abstraction

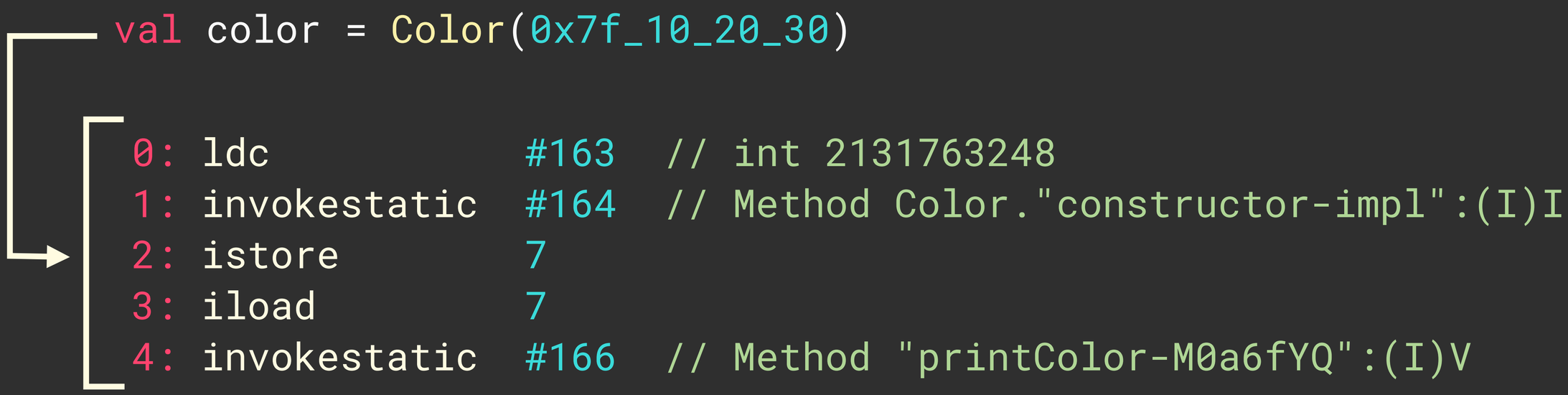
* When used right

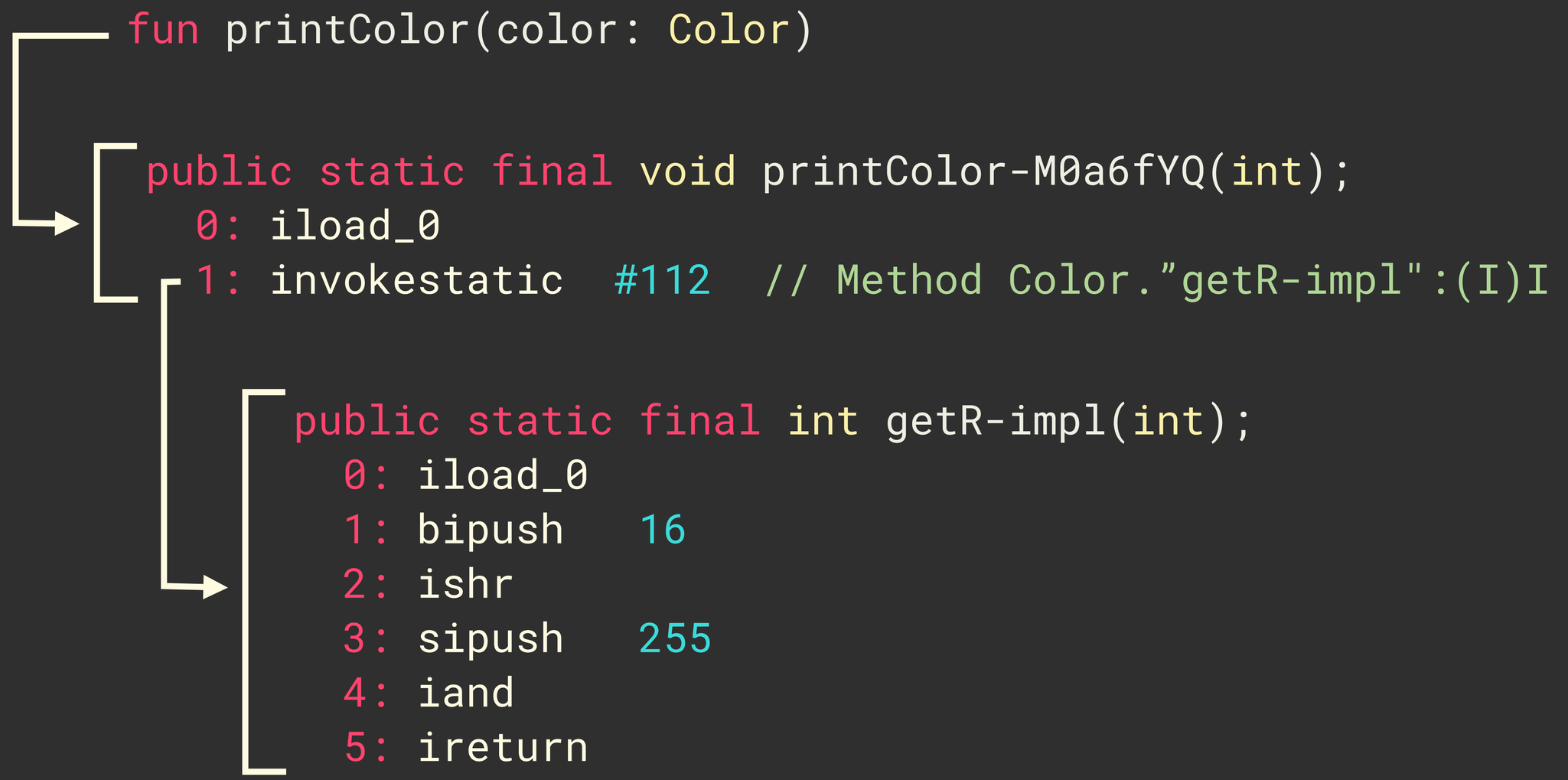
- Wraps a single value
- The underlying value is read-only
- Single constructor
- Can only implement interfaces
- `equals`/`hashcode`/`toString` for free
- Cannot be used as vararg*

* Unless you are UInt/ULong

```
inline class Color(private val c: Int) {
    val a: Int get() = (c shr 24) and 0xff
    val r: Int get() = (c shr 16) and 0xff
    val g: Int get() = (c shr  8) and 0xff
    val b: Int get() = (c       ) and 0xff
}
```

```
fun printColor(color: Color) {
    println("""
        red   = ${color.r / 255f}
        green = ${color.g / 255f}
        blue  = ${color.b / 255f}
        alpha = ${color.a / 255f}
    """.trimIndent())
}

val color = Color(0x7f_10_20_30)
printColor(color)
```

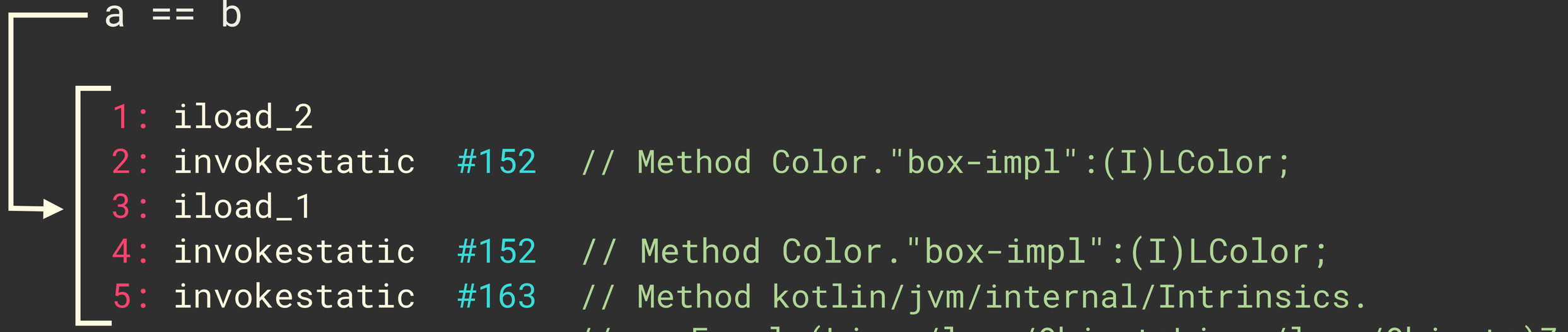

val color = Color(0x7f_10_20_30)
0: ldc #163 // int 2131763248
1: invokestatic #164 // Method Color."constructor-impl":(I)I
2: istore 7
3: iload 7
4: invokestatic #166 // Method "printColor-M0a6fYQ":(I)V

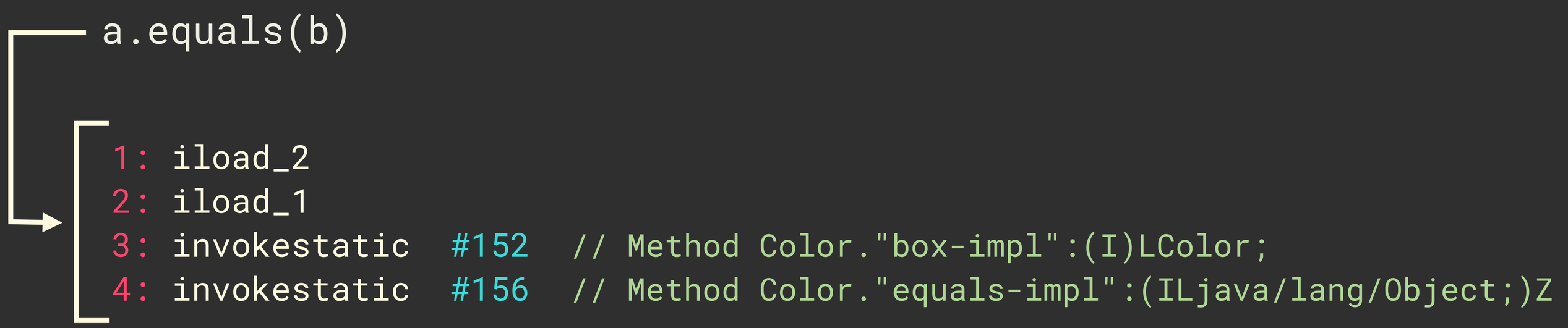

fun printColor(color: Color)
public static final void printColor-M0a6fYQ(int);
0: iload_0
1: invokestatic #112 // Method Color.”getR-impl":(I)I
public static final int getR-impl(int);
0: iload_0
1: bipush 16
2: ishr
3: sipush 255
4: iand
5: ireturn

Be careful with auto-boxing

- Nullable types
- Inside collections, arrays, etc.
- When Object or Any is expected

```
val a = Color(0x7f_10_20_30)
val b = Color(0x7f_30_20_10)

println(a == b)
println(a.equals(b))
```


a == b
1: iload_2
2: invokestatic #152 // Method Color."box-impl":(I)LColor;
3: iload_1
4: invokestatic #152 // Method Color."box-impl":(I)LColor;
5: invokestatic #163 // Method kotlin/jvm/internal/Intrinsics.
//.areEqual:(Ljava/lang/Object;Ljava/lang/Object;)Z


a.equals(b)
1: iload_2
2: iload_1
3: invokestatic #152 // Method Color."box-impl":(I)LColor;
4: invokestatic #156 // Method Color."equals-impl":(ILjava/lang/Object;)Z

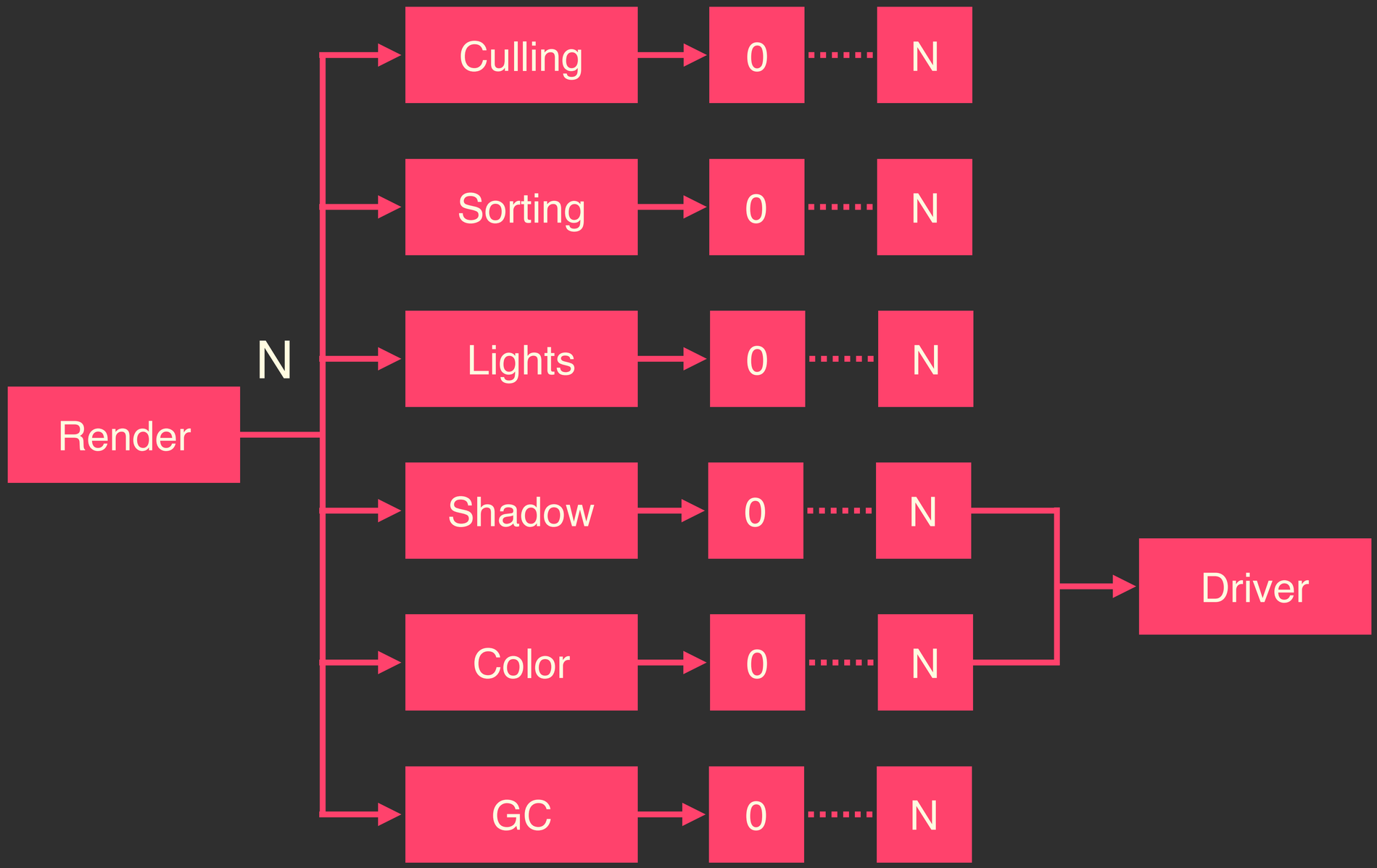
```
inline class Color(private val value: Int) {
    val a: Int get() = (value shr 24) and 0xff
    val r: Int get() = (value shr 16) and 0xff
    val g: Int get() = (value shr  8) and 0xff
    val b: Int get() = (value       ) and 0xff
}
```

```
inline class Color(val value: Int) {
    val a: Int get() = (value shr 24) and 0xff
    val r: Int get() = (value shr 16) and 0xff
    val g: Int get() = (value shr  8) and 0xff
    val b: Int get() = (value       ) and 0xff
}
```

```
val a = Color(0x7f_10_20_30)
val b = Color(0x7f_30_20_10)

println(a.value == b.value)
```

```
var s = “”

// Boxing!
s += a

// No boxing
s += a.toString()
```

Coroutines


Render
N
Culling
Sorting
Lights
Shadow
Color
GC
0
N
Driver

```
private fun startRender(image: BufferedImage, viewer: ImageViewer) {
    GlobalScope.launch {
        while (true) {
            runBlocking {
                renderTiles(image, viewer)
            }
        }
    }
}
```

```
private val NumCpu = Runtime.getRuntime().availableProcessors()
private val TilesDispatcher = newFixedThreadPoolContext(NumCpu * 3, "Renderer")

private suspend fun renderTiles(image: BufferedImage, viewer: ImageViewer) {
    // ...
    coroutineScope {
        repeat(NumCpu) { i ->
            repeat(NumCpu).forEach { j ->
                // ...
                launch(TilesDispatcher) {
                    val pixels = renderTile(x, y, w, h, resolution, time)
                    viewer.pushUpdate(x, height - h - y, w, h, pixels)
                }
            }
        }
    }
}
```

Kotlin scripting

My wishlist

User-defined literals

```
inline class Half(private val s: Short) {
    fun toFloat(): Float = // ...
}

fun Float.toHalf(): Half = // ...

suffix operator fun Float._h() = this.toHalf()
```

```
// pi is of type Half
val pi = 3.1415_h
```

```
// 2.0_s returns 2,000 ms
suffix operator fun Float._ns() = this / 1_000f
suffix operator fun Float._ms() = this
suffix operator fun Float._s()  = this * 1_000f
suffix operator fun Float._m()  = this * 60_000f
suffix operator fun Float._h()  = this * 1_440_000f
```

Short-form constructors

```
data class Vec3(x: Float, y: Float, z: Float)

fun lookAt(pos: Vec3) = // ...

lookAt([1f, 2f, 3f])
```

```
data class Vertex(pos: Float3, normal: Float3)

addVertex([[1f, 2f, 3f], [0f, 0f, 1f]])
```

Linear allocations

Kotlin/GPU

Where to find some code

kotlin-shadertoy
https://goo.gl/TFKUwP

kotlin-math
https://github.com/romainguy/kotlin-math

filament
https://github.com/google/filament
https://google.github.io/filament/Filament.md.html

Talks this week

Mathematical modeling with Kotlin
Today @13:00

Build a game using libGDX and Kotlin
Today @15:15

Porting D3.js to Kotlin Multiplatform
Tomorrow @13:00


Questions?