A TYPESCRIPT FAN'S KOTLINJS ADVENTURES

should you make the switch?

@BoyleEamonn

@GarthGilmour

INSTIL

we love typescript

it solves real problems for us

- **TS brings types and compilation to JS**
  - Improves upon the existing strengths of JS
  - Makes us more productive and happy
- **Leverages existing JS frameworks**
- **Interop with JS is a design goal**

we love kotlin

it solves real problems for us

- **Kotlin improves upon Java in many ways**
  - Adds null safety, DSLs, coroutines etc...
  - Makes us more productive and happy
- **Leverages existing JVM frameworks**
- **But interop with Java is a design goal**

Kotlin (JVM)	Kotlin (Android)	Kotlin (JS)	Kotlin (Native)
Java Bytecode	Dalvik Bytecode	JavaScript Interpreter	Machine Code & Runtime
JVM	ART	Browser	OS

JVM

Android

Browser

Native

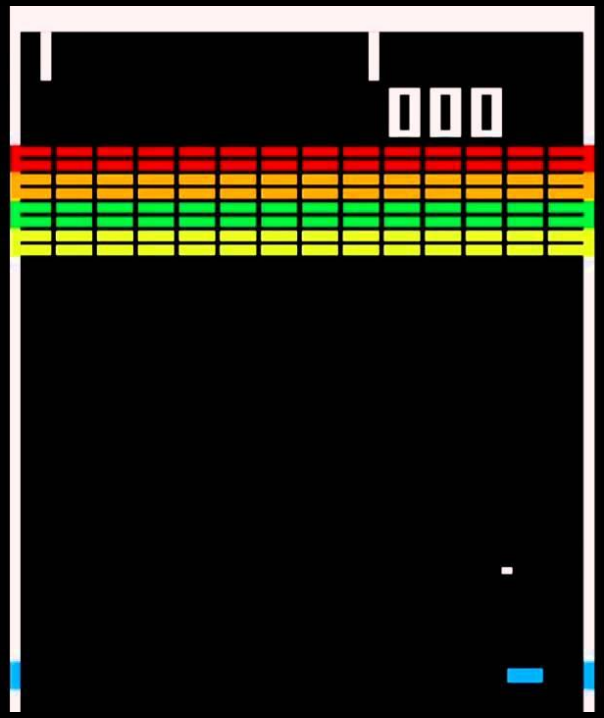
breaking down the problem

experiment: is kotlinjs worth it?
what problem does it solve?

- **Build an app in both TypeScript and KotlinJS**
  - Go beyond “hello world”
- **Incorporate common JS libraries**
- **Compare the experience**
  - Tooling
  - Language features
  - Community

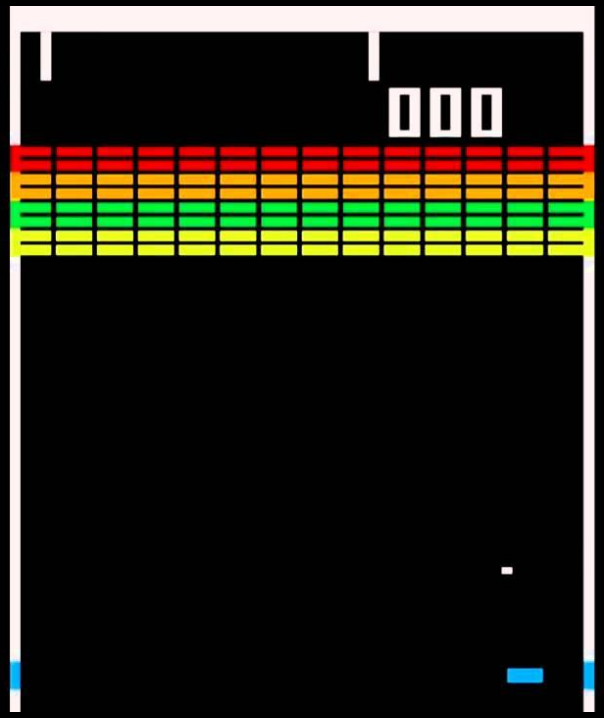
breakout clone

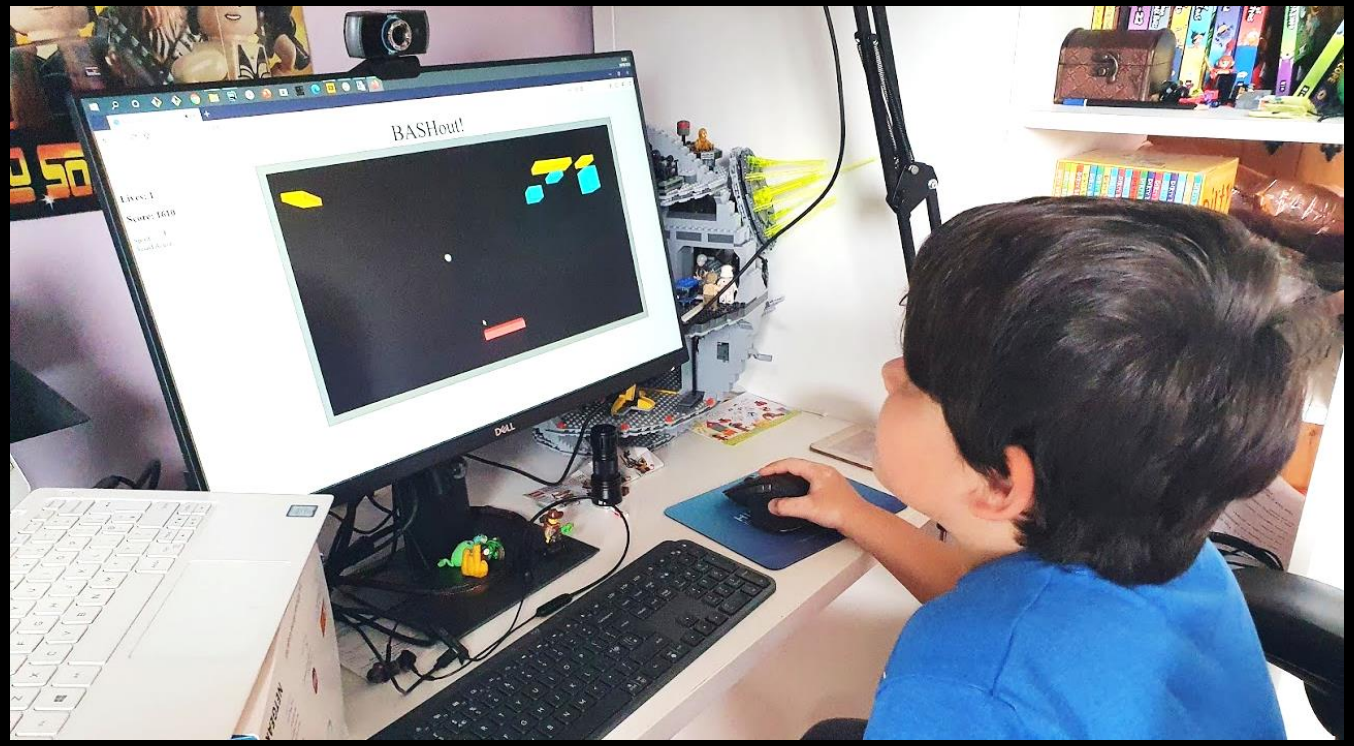
browser based clone



breakout clone

browser based clone





frameworks we used

- **React**
  - https://reactjs.org
- **Redux**
  - https://redux.js.org
- **React-Three-Fiber**
  - https://github.com/react-spring/react-three-fiber

declarative UI

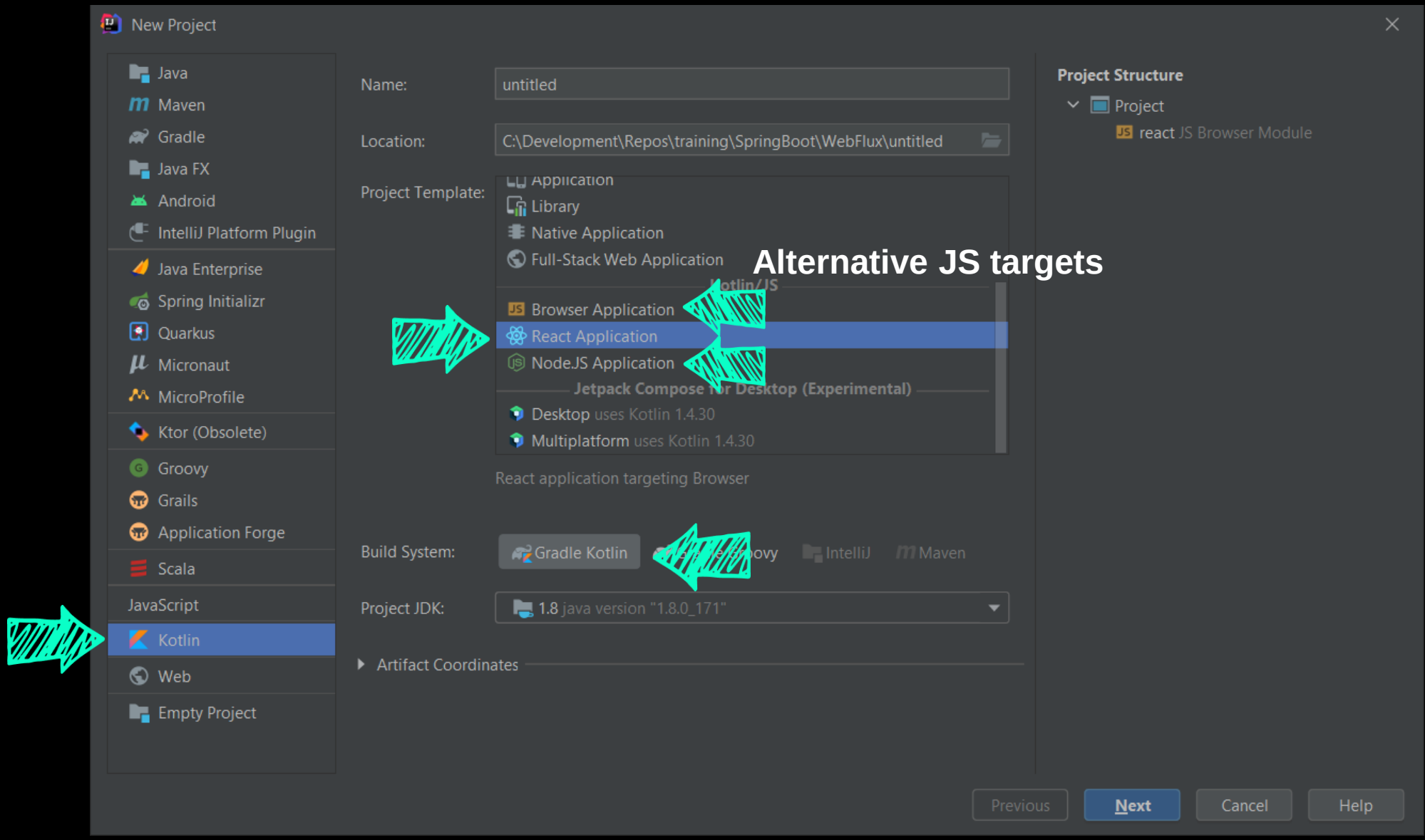
state separation

```
const geometry = new THREE.BoxGeometry(BAT_WIDTH, 1, 1);

export const Bat: FC = () => {
  const position = useSelector((state: State) => state.batPosition);

  return (
    <mesh position={[position, 0, BAT_Z]} geometry={geometry}>
      {woodMaterial}
    </mesh>
  );
};
```

creating a kotlinjs react project

creating new project
the wizard

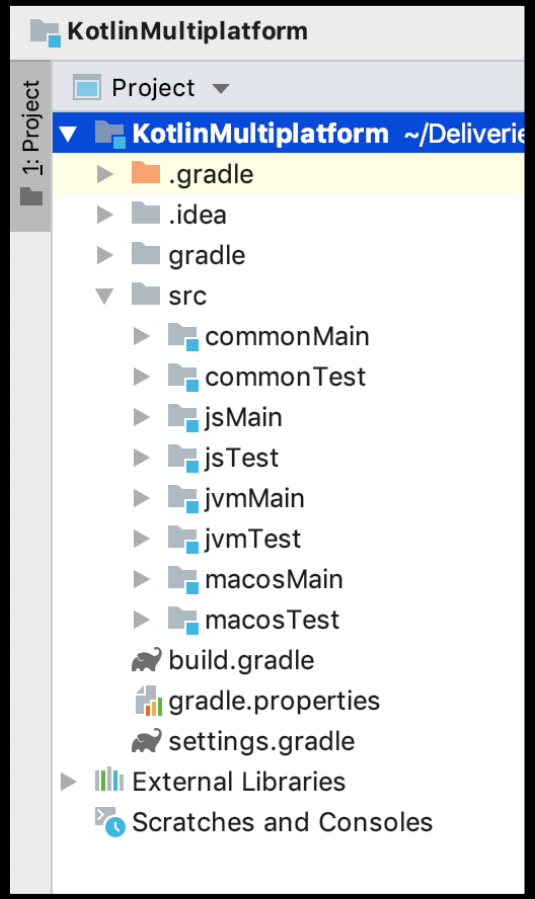
- Template project from IntelliJ
- Grade project using Kotlin DSL
- Easily integrate NPM packages



multiplatform libraries

COMMON KOTLIN

KOTLIN JVM
KOTLIN ANDROID
KOTLIN JS
KOTLIN NATIVE



NATIVE ARTEFACT

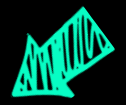
JS BUNDLE

JAR

kotlin js plugin

```
plugins {
    kotlin("js") version "1.5.0"
}

// ...

kotlin {
    js(LEGACY) {
        binaries.executable()
        browser {
            commonWebpackConfig {
                cssSupport.enabled = true
            }}}}
```

There is a new IR compiler



kotlin gradle dsl

```
$ ./gradlew tasks -all
…
Kotlin browser tasks
--------------------
browserDevelopmentRun - start development webpack dev server
browserDevelopmentWebpack - build webpack development bundle
browserDistributeResources
browserDistribution
browserProductionRun - start production webpack dev server
browserProductionWebpack - build webpack production bundle
browserRun
browserWebpack
```

add packages from multiple sources

kotlinjs, multiplatform and npm

```
dependencies {
    testImplementation(kotlin("test-js"))

    implementation("io.ktor:ktor-client-core:$ktorVersion")

    implementation("org.jetbrains:kotlin-react:17.0.1-pre.148-kotlin-1.4.30")
    implementation("org.jetbrains:kotlin-react-dom:17.0.1-pre.148-kotlin-1.4.30")
    implementation("org.jetbrains:kotlin-redux:4.0.5-pre.148-kotlin-1.4.30")
    implementation("org.jetbrains:kotlin-react-redux:7.2.2-pre.148-kotlin-1.4.30")

    implementation(npm("bootstrap", "4.6.0"))
    implementation(npm("jquery", "1.9.1 - 3"))
    implementation(npm("popper.js", "^1.16.1"))
    implementation(npm("lodash", "^4.7.0"))
}
```

get down to coding

main

```
import kotlinx.browser.document
import kotlinx.browser.window
import react.dom.render
import react.router.dom.browserRouter

fun main() {
    window.onload = {
        render(document.getElementById("root")) {
            browserRouter {
                // ...
            }
        }
    }
}
```

get down to coding

easy access to standard browser apis

```
import kotlinx.browser.document
import kotlinx.browser.window
import react.dom.render
import react.router.dom.browserRouter

fun main() {
    window.onload = {
        render(document.getElementById("root")) {
            browserRouter {
                // ...
            }
        }
    }
}
```

Moved from `kotlin.browser` to `kotlinx.browser` in 1.4

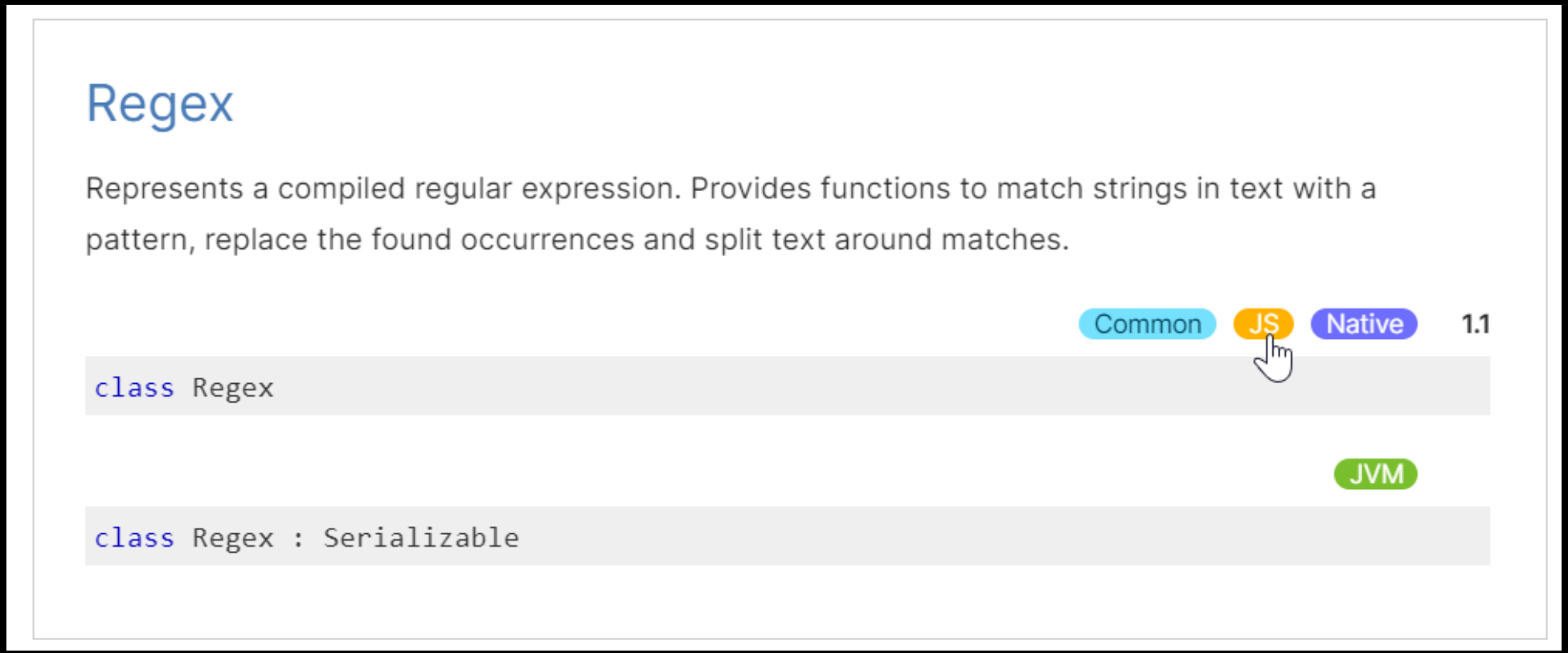
get down to coding

easy access to react library

```
import kotlinx.browser.document
import kotlinx.browser.window
import react.dom.render
import react.router.dom.browserRouter

fun main() {
    window.onload = {
        render(document.getElementById("root")) {
            browserRouter {
                // ...
            }
        }
    }
}
```

standard library documentation

excellent compatibility docs



standard library documentation

excellent compatibility docs

HashMap

Hash table based implementation of the MutableMap interface.

Common Native 1.2

```
class HashMap<K, V> : MutableMap<K, V>
```

JVM 1.1

```
typealias HashMap<K, V> = HashMap<K, V>
```

JS 1.1

```
open class HashMap<K, V> :
    AbstractMutableMap<K, V>,
    MutableMap<K, V>
```

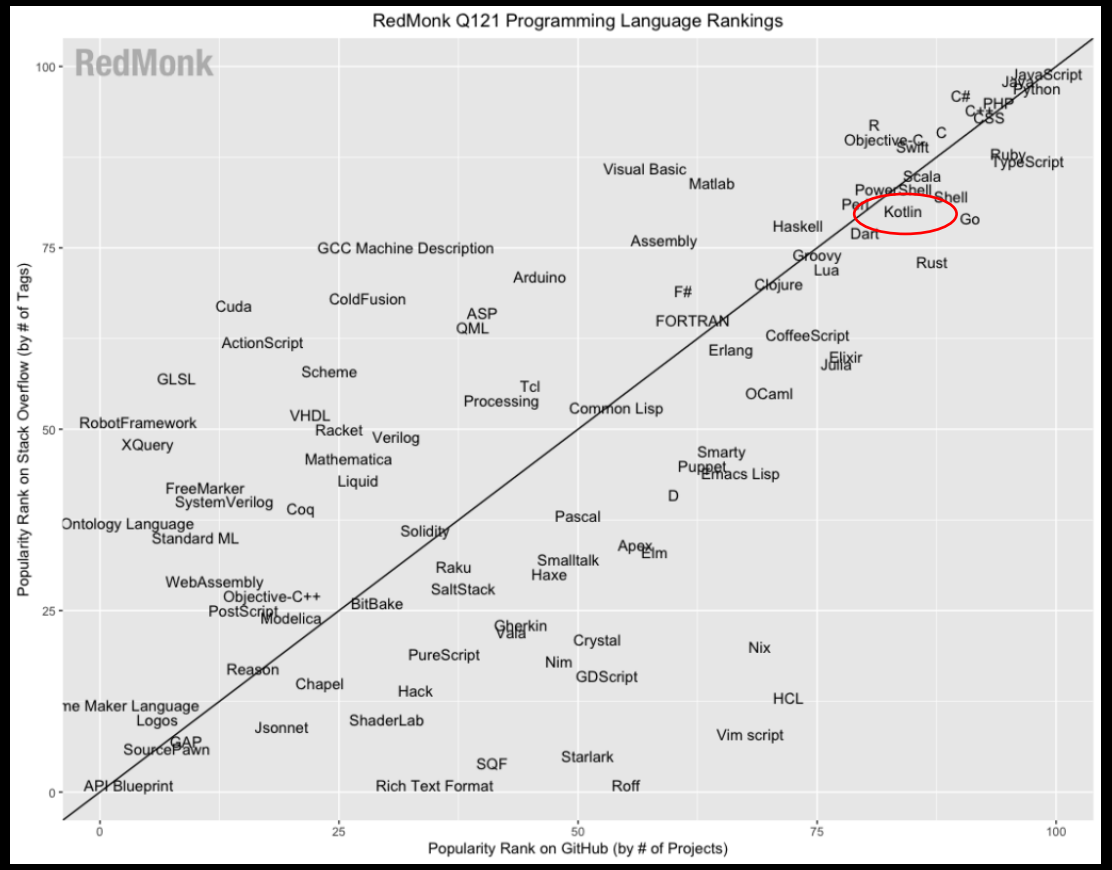
Round 1

community

its popularity continues to grow
rising star on language rankings

The RedMonk Programming Language Rankings: January 2021

https://redmonk.com/sogrady/2021/03/01/language-rankings-1-21/



…
18 Kotlin
…

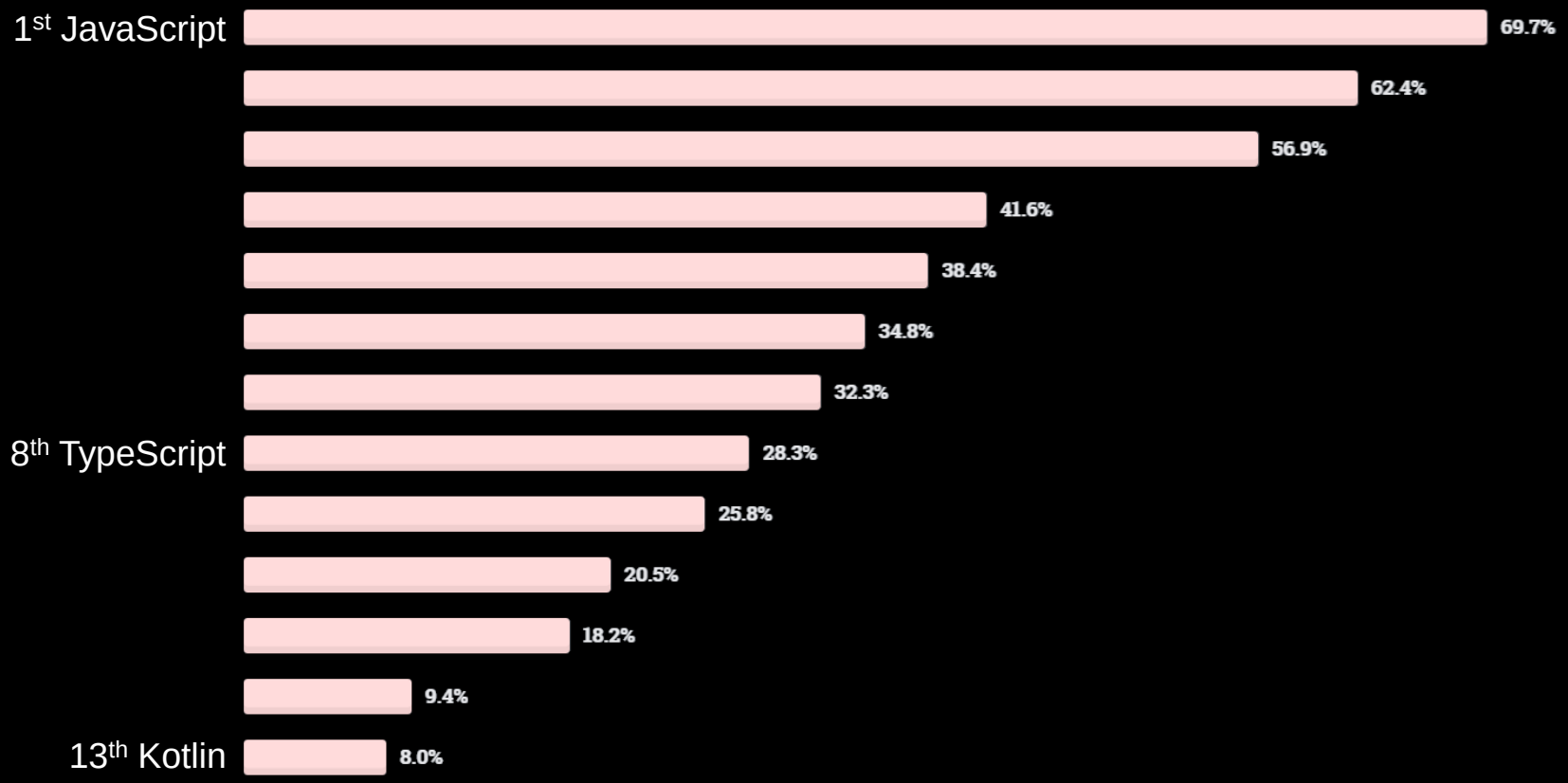
its popularity continues to grow

rising star on language rankings

Stack Overflow Developer Survey 2020

Most Used Language

https://insights.stackoverflow.com/survey/2020



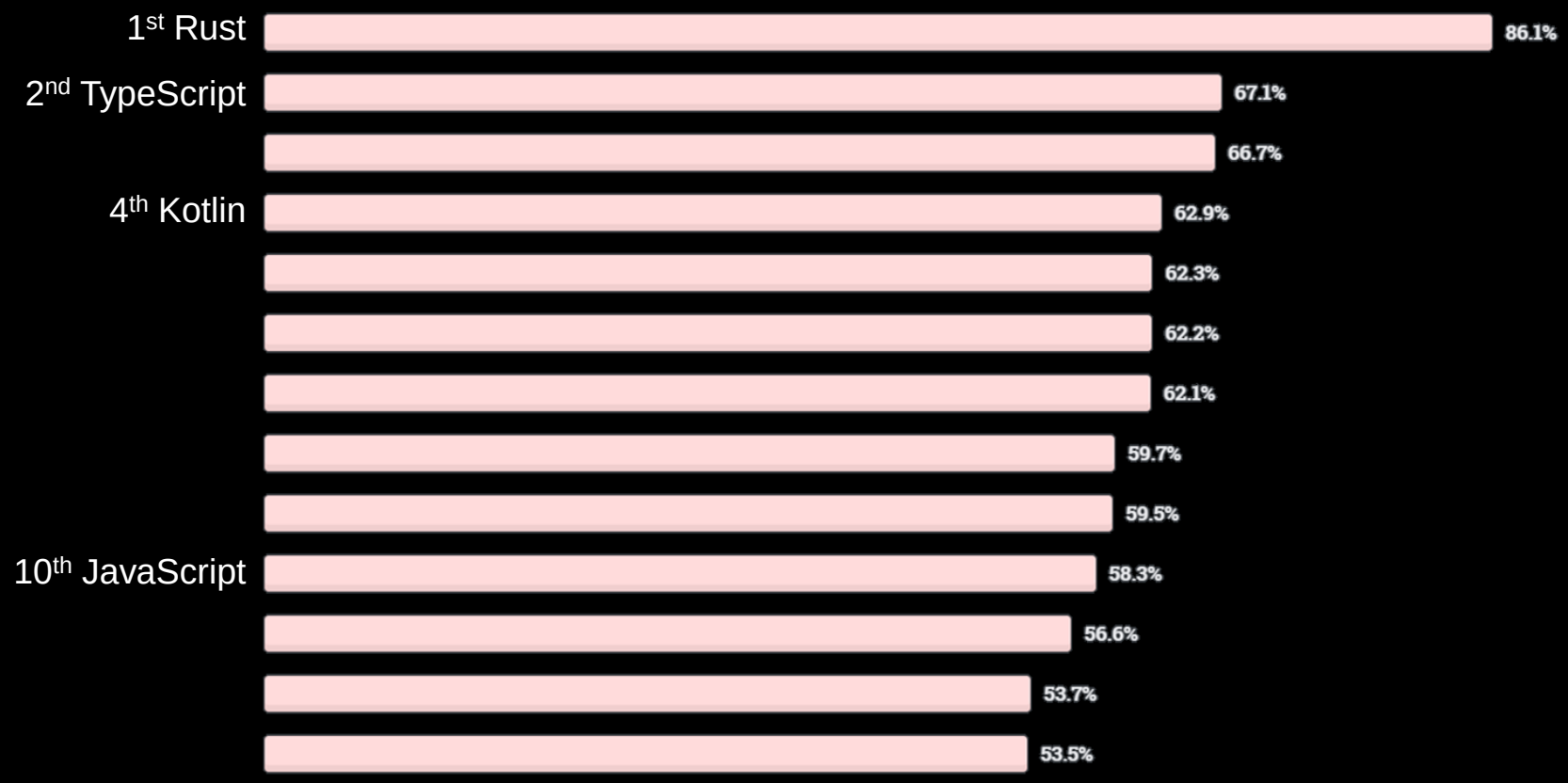
its popularity continues to grow

loved by its users

Stack Overflow Developer Survey 2020

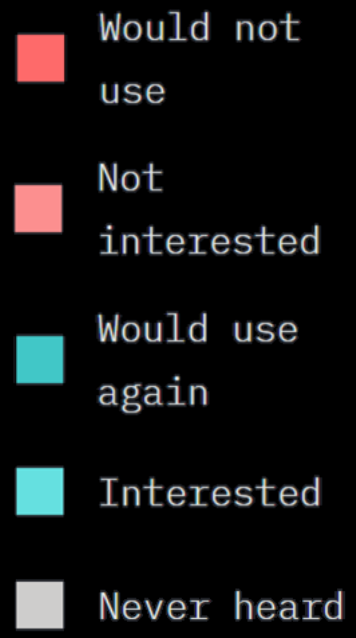
Most Loved Language

https://insights.stackoverflow.com/survey/2020

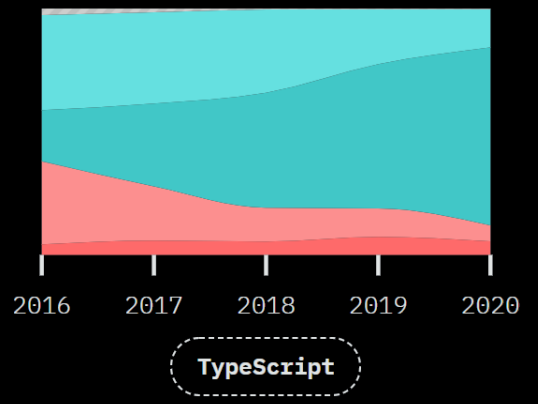


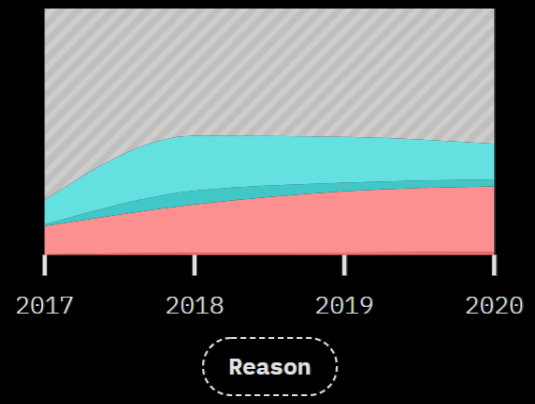
TS is good at retaining adopters

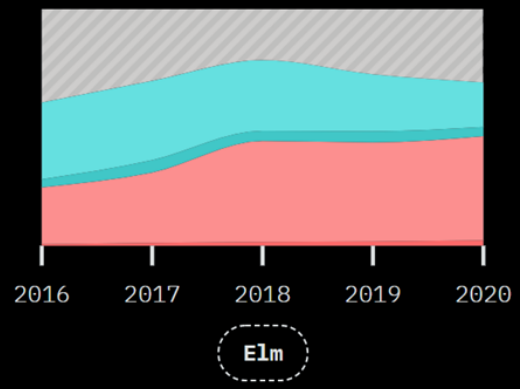
https://2020.stateofjs.com/en-US/

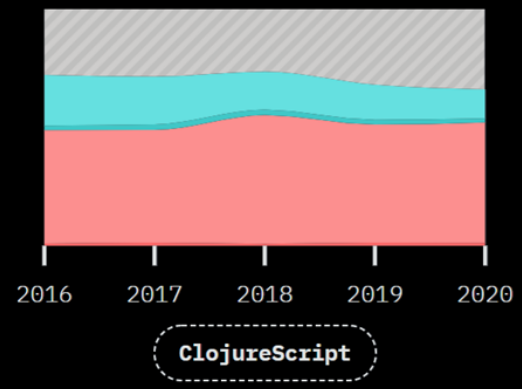


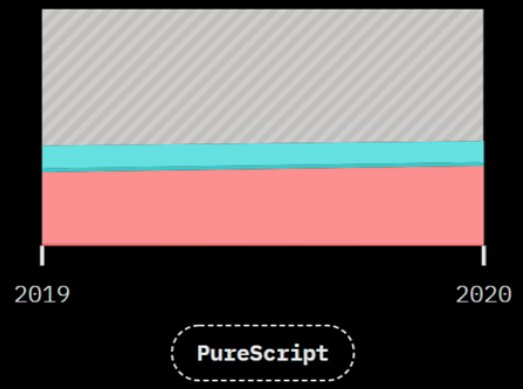
TypeScript in 2020
72.2% would use again
15.5% interested











community, maturity and support

typescript > kotlin

- **TypeScript is more popular than KotlinJS**
- **As a superset of JS, reusing knowledge and assets is easier**
  - And the transition for JS developers to TS is easier
- **TypeScript is well established in the JavaScript world**
  - Many libraries include TypeScript definitions
  - DefinitelyTyped contains many more

Round 2

interop with javascript

importing npm js packages
gradle dsl

- Only a few first class wrappers provided
- It is easy to add NPM packages yourself

```
dependencies {
    . . .
    implementation(npm("react-three-fiber", "4.2.20"))
    implementation(npm("react-use-gesture", "7.0.15"))
    implementation(npm("three", "0.119.1"))
}
```

- But how easy is it to consume that code in Kotlin?

really easy
external declarations

```
@file:JsModule("react-three-fiber")
@file:JsNonModule

...

external val Canvas: RClass<RProps>

external fun extend(objects: Any)

external fun useFrame(callback: (dynamic, Double) -> Unit)

external fun useThree(): dynamic

external interface PointerEvent {
    val uv: Vector2
}
```

Specify the NPM package

Define any items you wish to use

dukat

automatic generation

- Converts TypeScript `d.ts` files to Kotlin external declarations
- Still experimental
- At time of writing, on hold until IR compiler stabilises

https://kotlinlang.org/docs/js-external-declarations-with-dukat.html

dukat

usage

- **Command line tool**
  - Installable in isolation via npm

```
$ npm install dukat
```

- **Simply point to a .d.ts file and it will do the rest**

```
$ dukat index.d.ts index.module_my-library.kt
```


```ts
export function getString(): string;

export function setString(input: string): void;
```

```
external fun getString(): String

external fun setString(input: String)
```

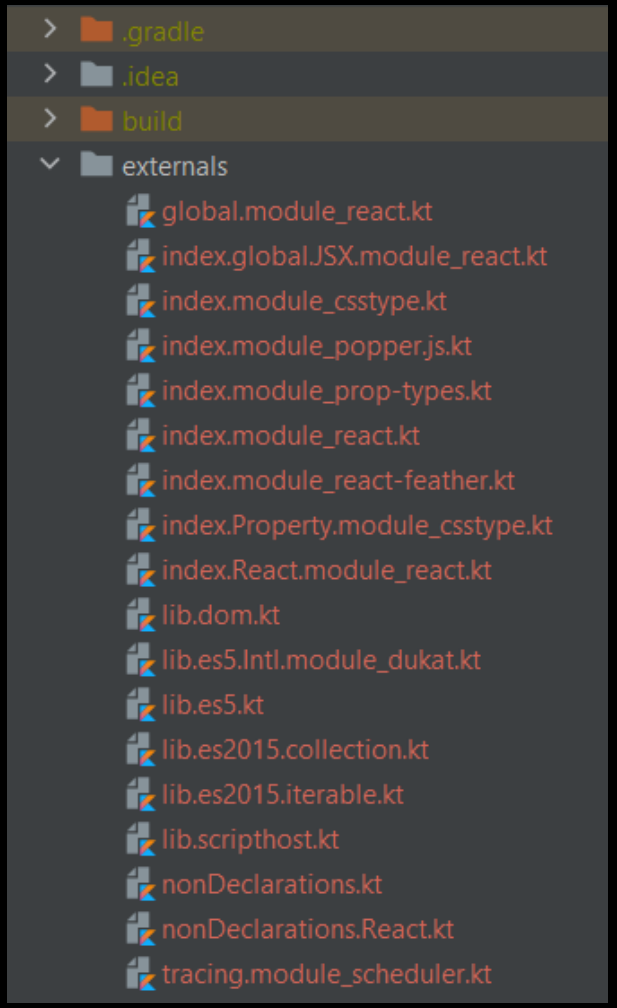
dukat

dukat

usage

- **Integrated into Gradle**
  - Generates definitions for configured packages

```
Dukat tasks
-----------
generateExternals
generateExternalsIntegrated
```

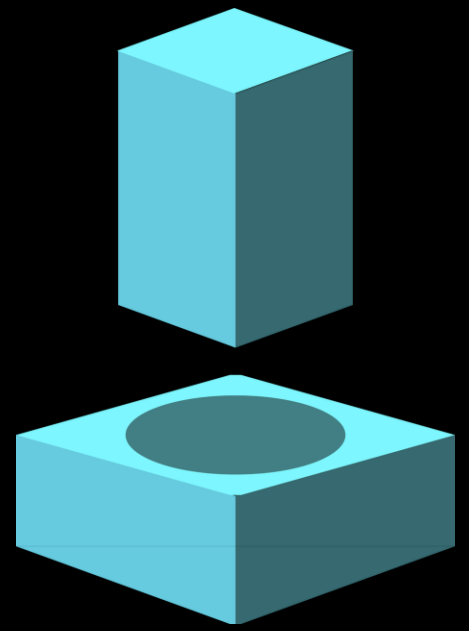


```ts
export interface BasicInterface {
    readonly field1: number;
    method1(): boolean;
}

export function buildInterface(): BasicInterface;

export type ReadOnlyBasicInterface = Readonly<BasicInterface>;
```

```ts
external interface BasicInterface {
    var field1: Number
    fun method1(): Boolean
}

external fun buildInterface(): BasicInterface

typealias ReadOnlyBasicInterface = Readonly<BasicInterface>
```

not a silver bullet

square peg and a round hole



- It’s a good help but has issues
- Limited by differences in the languages
- Not a seamless workflow
- This situation may improve as the tool and Kotlin evolves

dynamic

get out of jail type

- **KotlinJS supports a `dynamic` type**
  - You can use it in place of any type
  - Assign it a value of any type
  - Access and use members with any name
- **Basically switches off type checking**
- **This can be used to quickly patch over APIs**

```
external fun useFrame(callback: (dynamic, Double) -> Unit)
```

jsObject
object literals in kotlinjs

- Object literals are common in JavaScript
- KotlinJS has a helper function to create objects

```
Block(jsObject {
    position = brick.location.toVector3()
    color = brick.color
})
```

- This is strongly typed (inferred) but requires fields to be `var`

js
embedding javascript

- **We can embed JavaScript directly with the `js` function**
  - The code can even use Kotlin variables
- **Must be a compile time constant**
  - Cannot be a runtime evaluated expression

```
private fun getToken(): String? {
    val tokenKey = TokenKey
    return js("""
        localStorage.getItem(tokenKey)
    """)
}
```

interop with javascript

typescript > kotlinjs

- **As a superset, TypeScript has to win this one**
- **The type system is geared to support JavaScript**
  - Lots of libraries already provide TypeScript definition files
- **However, writing external declaration in KotlinJS is easy**
- **Dukat exists but has fundamental limitations**
  - You may have to write custom translation code on top

Round 3

jsx vs dsl

```tsx
export const Hud: FC = () => {
  // ...

  return (
    <div>
      <h2>Lives: {lives}</h2>
      <h2>Score: {score}</h2>
      <div>
        <label>Speed:</label>
        <input type="range" min={1} max={50} value={speedScalar * 10}
               onChange={e => /* ... */}/>
      </div>
      <div>
        <label>Sound Active:</label>
        <input type="checkbox" checked={soundActive}
               onChange={e => /* ... */}/>
      </div>
    </div>
  );
};
```

JSX embeds markup inside JS / TS code

```
val Hud = FC {
    // ...

    div {
        h2 { +"Lives: $lives" }
        h2 { +"Score: $score" }
        div {
            label { +"Speed:" }
            input(type = InputType.range) {
                attrs {
                    min = "1"
                    max = "50"
                    value = (speedScalar * 10).toString()
                    onChangeFunction = {/* ... */}
                }}}
        div {
            label { +"Sound Active:" }
            input(type = InputType.checkBox) {
                attrs {
                    checked = soundActive
                    onChangeFunction = {/* ... */}
                }}}}
}
fun RBuilder.Hud() = child(Hud)
```

In Kotlin a DSL provides equivalent functionality

kotlin DSL's are very cool

a major advantage

Extension function

```
inline fun RBuilder.div(
  classes: String? = null,
  block: RDOMBuilder<DIV>.() -> Unit
): ReactElement
```

Optional param via default

Last param is a
lambda with receiver

```
// Here, “this” is the containing object
div {
 // Here, “this” is a RDOMBuilder<DIV>
}
```

```
inline fun RBuilder.div(
  classes: String? = null,
  block: RDOMBuilder<DIV>.() -> Unit
): ReactElement
```

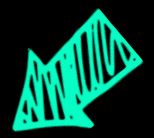
```
// Here, “this” is the containing object
div {
 // Here, “this” is a RDOMBuilder<DIV>
}
```

But some parts are cumbersome...

```
val Hud = FC {
    // ...

    div {
        h2 { +"Lives: $lives" }
        h2 { +"Score: $score" }
        div {
            label { +"Speed:" }
            input(type = InputType.range) {
                attrs {
                    min = "1"
                    max = "50"
                    value = (speedScalar * 10).toString()
                    onChangeFunction = {/* ... */}
                }}}
        // ...
}
fun RBuilder.Hud() = child(Hud)
```

```
val Hud = FC {
    // ...

    div {
        h2 { +"Lives: $lives" }
        h2 { +"Score: $score" }
        div {
            label { +"Speed:" }
            input(type = InputType.range) {
                attrs {
                    min = "1"
                    max = "50"
                    value = (speedScalar * 10).toString()
                    onChangeFunction = {/* ... */}
                }}}
        // ...
    }
fun RBuilder.Hud() = child(Hud)
```

Single ‘bucket’ for attributes



Overloaded operators to attach data

```
val Hud = FC {
    // ...

    div {
        h2 { +"Lives: $lives" }
        h2 { +"Score: $score" }
        div {
            label { +"Speed:" }
            input(type = InputType.range) {
                attrs {
                    min = "1"
                    max = "50"
                    value = (speedScalar * 10).toString()
                    onChangeFunction = {/* ... */}
                }}}
        // ...
}
fun RBuilder.Hud() = child(Hud)
```

```kotlin
val Hud = FC {
    // ...

    div {
        h2 { +"Lives: $lives" }
        h2 { +"Score: $score" }
        div {
            label { +"Speed:" }
            input(type = InputType.range) {
                attrs {
                    min = "1"
                    max = "50"
                    value = (speedScalar * 10).toString()
                    onChangeFunction = {/* ... */}
                }}}
        // ...
}
fun RBuilder.Hud() = child(Hud)
```

Additional extension function required

```
val Hud = FC {
    // ...

    div {
        h2 { +"Lives: $lives" }
        h2 { +"Score: $score" }
        div {
            label { +"Speed:" }
            input(type = InputType.range) {
                attrs {
                    min = "1"
                    max = "50"
                    value = (speedScalar * 10).toString()
                    onChangeFunction = {/* ... */}
                }}}
        // ...
    }
fun RBuilder.Hud() = child(Hud)
```

Attributes must be given as strings*

```ts
interface InputHTMLAttributes<T> extends HTMLAttributes<T> {
    max?: number | string;
    min?: number | string;
    value?: string | ReadonlyArray<string> | number;
    ...
}
```

Type union

```ts
type PropsWithChildren<P> = P & { children?: ReactNode };
```

Type intersection

```
export type InterfaceUnion = First | Second;

export function interfaceUnionInput(input: InterfaceUnion): void;

export function interfaceUnionOutput(): InterfaceUnion;
```

```
// Type exports erased!

external fun interfaceUnionInput(input: First)

external fun interfaceUnionInput(input: Second)

external fun interfaceUnionOutput(): dynamic /* First | Second */
```

Kotlin supports proper overloads

Not supported on the return type

```ts
export type InterfaceIntersection = First & Second;

export function interfaceIntersectionInput(input: InterfaceIntersection): void;

export function interfaceIntersectionOutput(): InterfaceIntersection;
```

```ts
external fun interfaceIntersectionInput(input: First /* First & Second */)

external fun interfaceIntersectionOutput(): First /* First & Second */
```

Intersection dropped

mapped and conditional types

even more power in typescript

```
type Readonly<T> = {
    readonly [P in keyof T]: T[P];
};

type PromiseType<T extends Promise<any>> =
    T extends Promise<infer U> ? U : never;
```

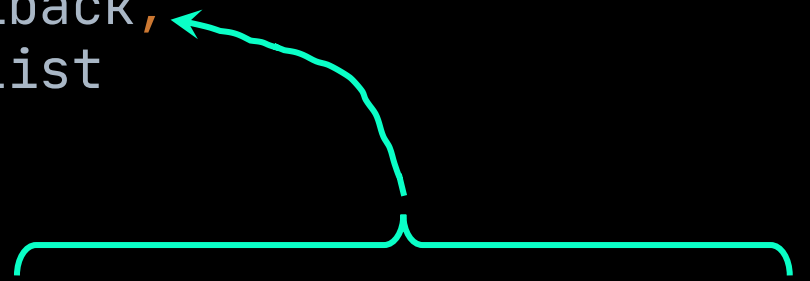
Type conditional

The languages are simply not fully compatible

manual workarounds

useEffect

```
function useEffect(
    effect: EffectCallback,
    deps?: DependencyList
): void;


type EffectCallback = () => (void | Destructor);

type Destructor = () => void;
```



useEffect

union return workaround

```
fun useEffect(
    dependencies: RDependenciesList? = null,
    effect: () -> Unit
) {
    // ...
}

fun useEffectWithCleanup(
    dependencies: RDependenciesList? = null,
    effect: () -> Rcleanup
) {
    // ...
}
```

jsx vs dsl

typescript > kotlin

- **Kotlin's DSL support is a powerful general purpose tool**
  - But JSX is a single purpose solution that suits React better
- **TypeScript's advanced type system is very powerful**
  - Union, intersection and mapped types bring sanity to JS
  - Kotlin types (unsurprisingly) sit awkwardly on top of JS

Round 4

async await vs coroutines

asynchronous programming
promises and async/await

- **Async await is a good async solution in JS & TS**
  - Engineered so it interops with Promises
  - Succinct

```
async function loadMap(url: string): Promise<void> {
  const response = await fetch(url);
  const map = await response.text();

  // ...
}
```

coroutines

kotlin > typescript

- **Kotlin's more general coroutines are better**
  - Works with other patterns than simply async
- **In KotlinJS, it works easily with Promises**

```
suspend fun loadMap(url: String) {
    val response = window.fetch(url).await()
    val map = response.text().await()

    // ...
}
```

coroutines

kotlin > typescript

- **Coroutines are more general and powerful**
  - They can be used with other patterns too
- **With suspend functions we don't need to "await"**

```
suspend fun loadMap(url: String) {
    val map = client.get<String>(url)

    // ...
}
```


Ktor Client

coroutines

kotlin > typescript

- Coroutines can be applied to other patterns too

```
fun infinite() = sequence {
    var count = 0
    while (true) {
        yield(count++)
    }
}
```

Round
5

elegant syntax

expressions

kotlin > typescript

- **Kotlin doesn't have the ternary, but has more**
  - **when** for basic pattern matching
  - **if** and **when** are expressions
  - **Unit** instead of void
  - Expression bodied functions
- **This creates more symmetry in code**

typescript
chained ternaries

```tsx
const App: FC = () => {
  // ...
  return (
    <div>
      // ...
      {gameState === GameState.Start ? <StartScreen/> :
       gameState === GameState.Playing ? <PlayingGame/> :
       gameState === GameState.Dead ? <DeathScreen/> :
       gameState === GameState.Win ? <WinScreen/> :
       null}
    </div>
  );
};
```

kotlin

when expression

```
val App = FC {
    // ...
    div {
        // ...
        when (gameState) {
            GameState.Start -> StartScreen()
            GameState.Playing -> PlayingScreen()
            GameState.Win -> WinScreen()
            GameState.Dead -> EndScreen()
        }
    }
}
```

destructuring

typescript > kotlin

- **Both languages support object destructuring**
  - Within blocks and in lambda parameters
- **However, Kotlin's is limited**
  - Supported via *`componentN`* methods
  - Fixed order to properties extracted
  - Data classes do this automatically

```
const {value, color} = brick;
```

Arbitrary properties extracted

```
const {value, location} = brick;
```

Destructuring on parameters 

```
export const Brick: FC<Props> = ({index}) => {
}
```

Array destructuring

```
const [first, ...remaining] = bricks;
```

conclusion

they’re both very good

but in different ways

Community

Coroutines

JSX vs React DSL

Multiplatform

Advanced Type System

Extensions

Interop with JS

Expressions

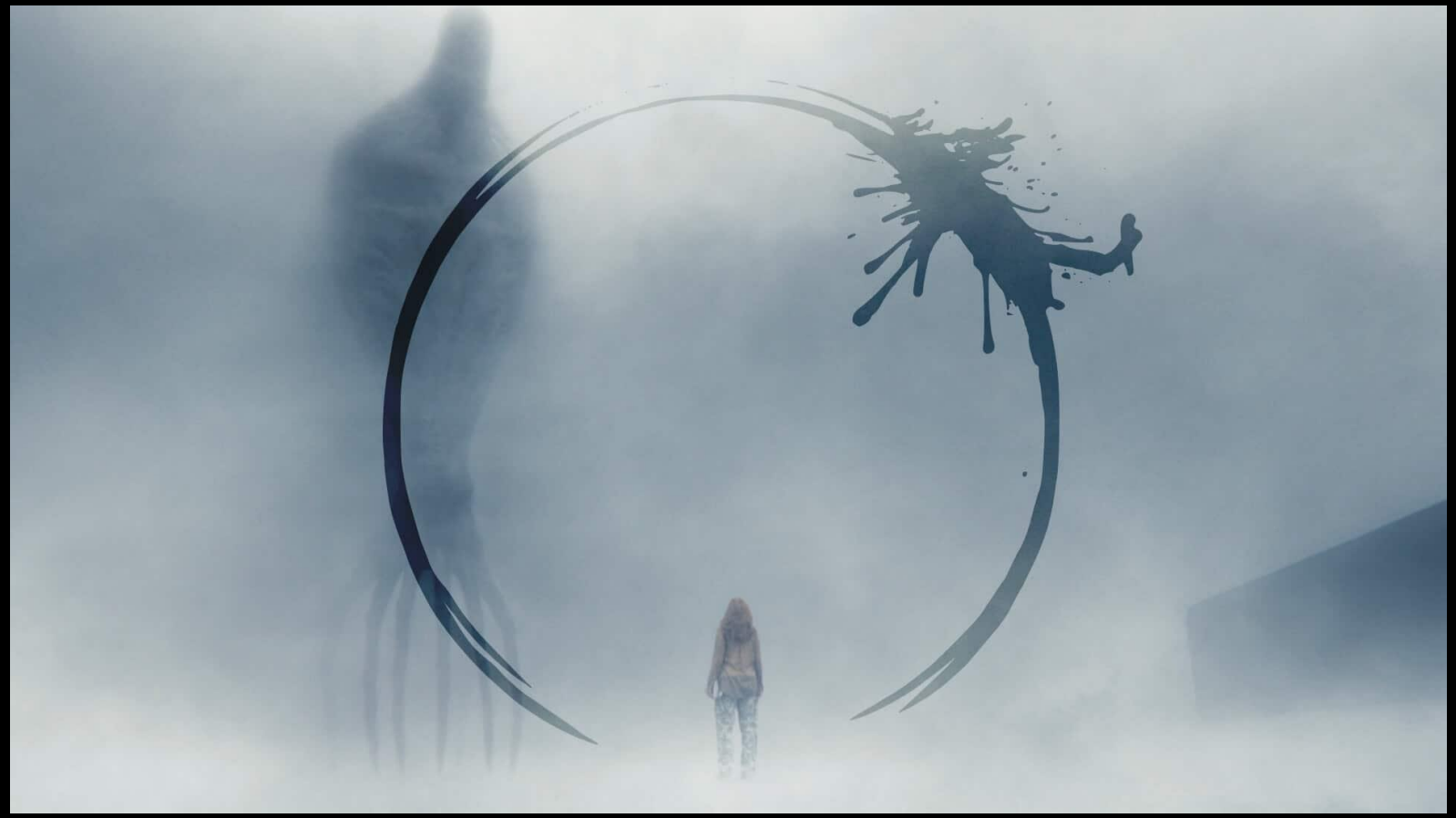
Destructuring

Standard Library

Tooling

Functions

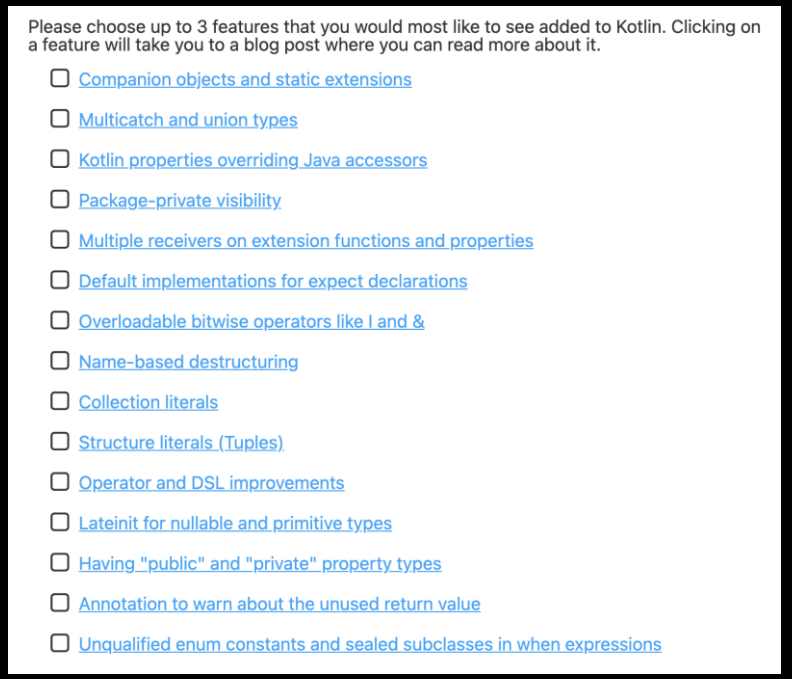
game changers



game changer 1

potential new language features

- Union Types with `|' syntax
- Improvements to Sealed Types
- Extended Operator Overloading
- Name based Destructuring
- Collection Literals
- Structure Literals (Tuples?)



https://compose-web.ui.pages.jetbrains.team/

game changer 2
compose for web

- **Brings Google's UI Toolkit to web**
- **In theory, enables reuse of UI code across stack**

```
Div(attrs = attrs) {
    Label {
        Input(
            type = InputType.Checkbox,
            attrs = {}
        )
        Span {
            content()
        }}}
```

Questions?