goto;

GOTO
Guide

LET US HELP YOU

Ask questions through **the app**

★★★★★

also remember to rate session

Download on the App Store

GET IT ON Google Play

THANK YOU!

#GOTOams

TypeScript vs KotlinJS

which child deserves
your love

Gearset

eamonn.boyle@gearset.com

@BoyleEamonn

Eamonn Boyle

Eamonn Boyle

Development Manager

garth

garth.gilmour@instil.co

@GarthGilmour

Garth Gilmour

INSTIL

we love typescript

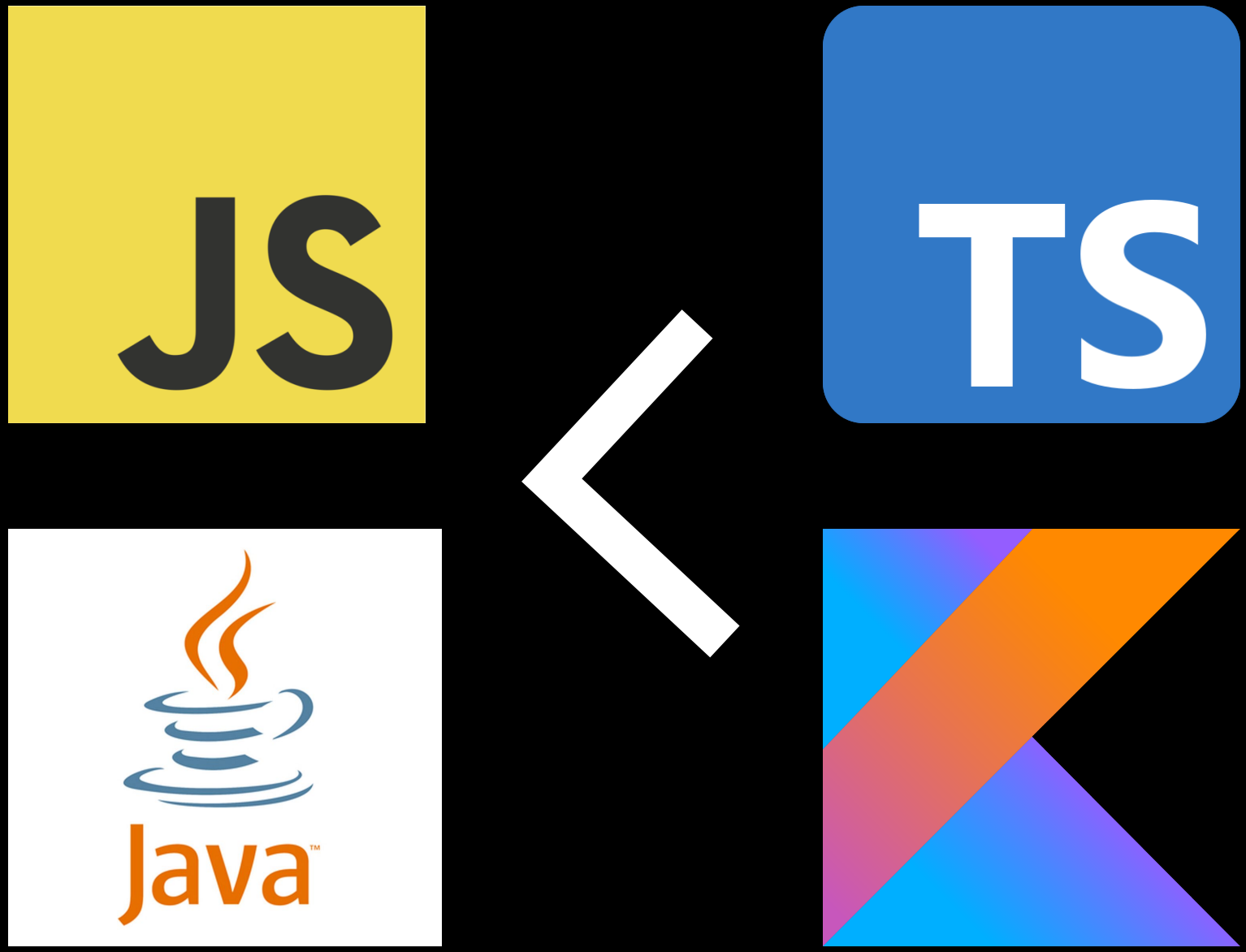
it solves real problems for us

- **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**


JS
TS
Java

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

experiment: is kotlinjs worth it?

what problem does it solve?

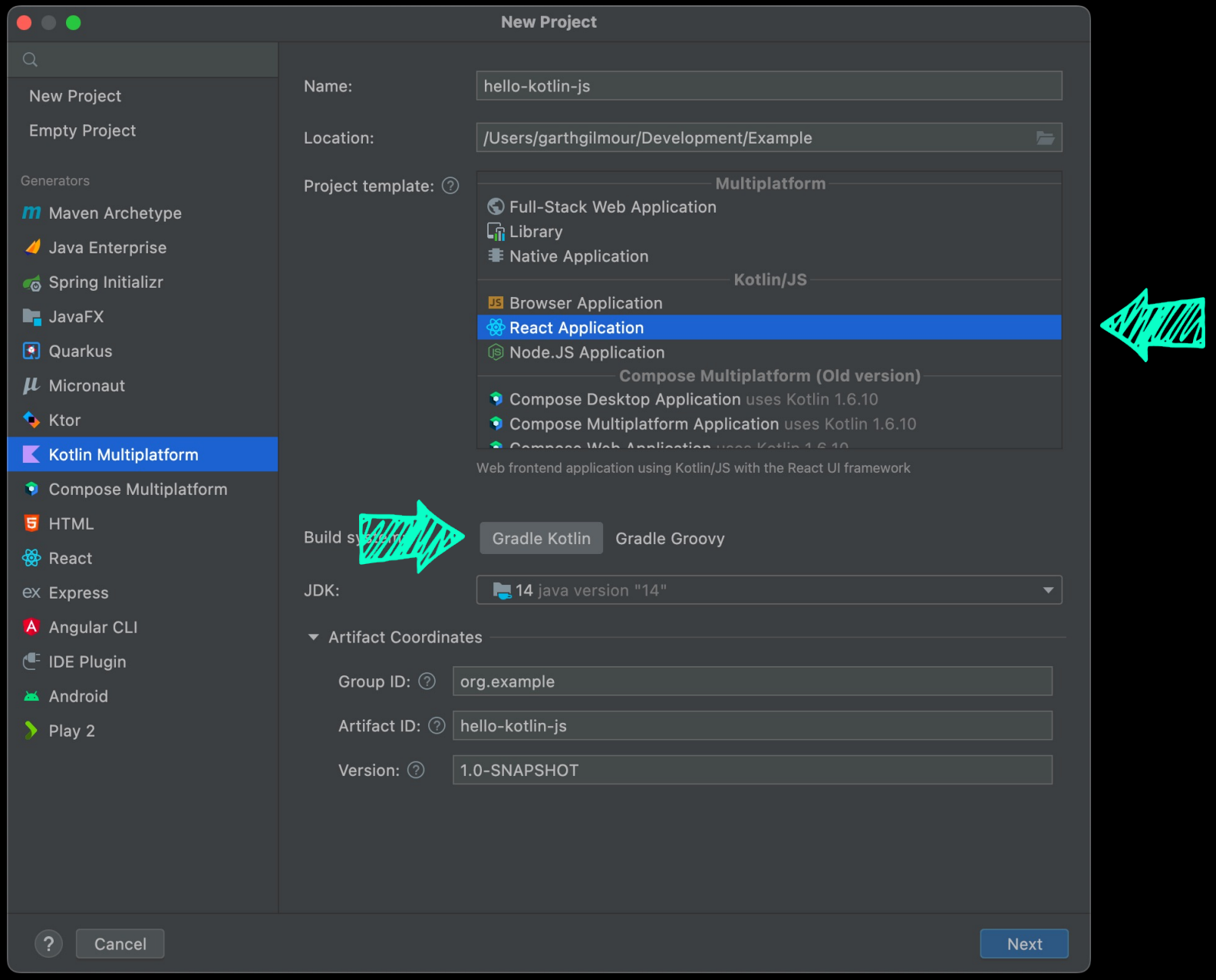
- **We are not using KotlinJS in production**
  - The code written has been for labs, workshops and talks
- **Built several apps both TypeScript and KotlinJS**
  - Various features - REST, Forms, Routing, 3D graphics, React
  - Incorporate common JS libraries
- **Compare the experience**
  - Tooling
  - Language features
  - Community

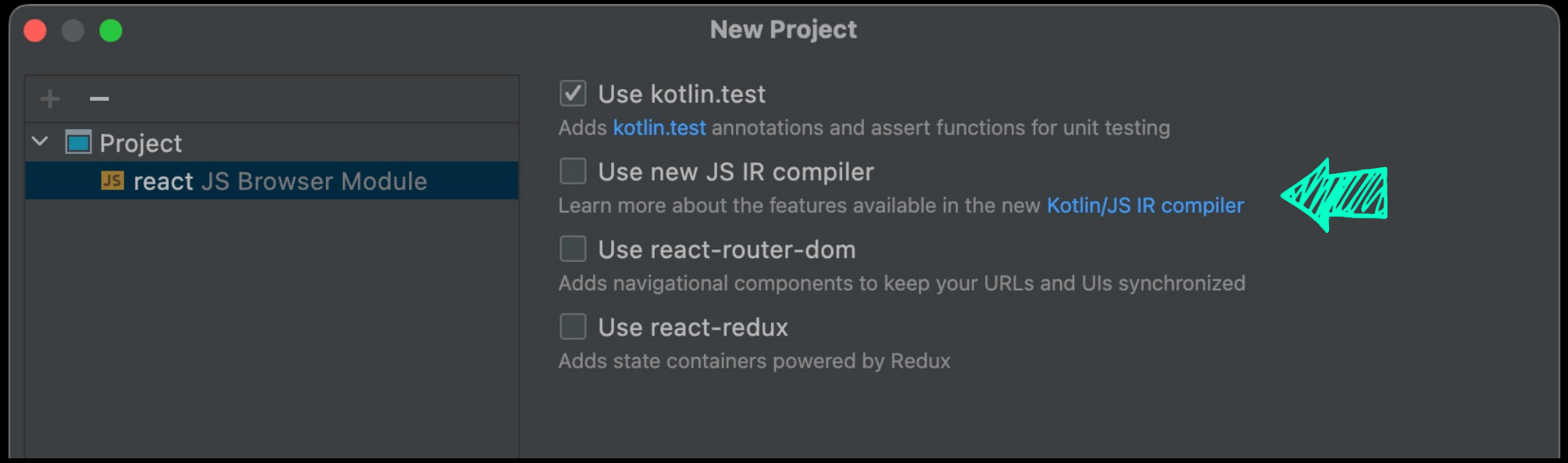
creating a kotlinjs react project

creating new project

the wizard

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

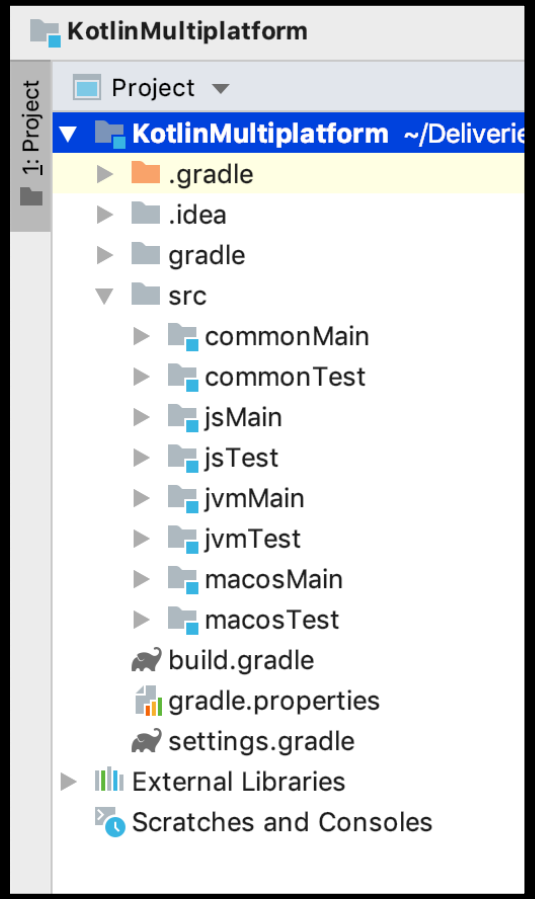

New Project
New Project
Empty Project
Generators
Maven Archetype
Java Enterprise
Spring Initializr
JavaFX
Quarkus
Micronaut
Ktor
Kotlin Multiplatform
Compose Multiplatform
HTML
React
Express
Angular CLI
IDE Plugin
Android
Play 2
Name:
hello-kotlin-js
Location:
/Users/garthgilmour/Development/Example
Project template:
Multiplatform
Full-Stack Web Application
Library
Native Application
Kotlin/JS
Browser Application
React Application
Node.JS Application
Compose Multiplatform (Old version)
Compose Desktop Application uses Kotlin 1.6.10
Compose Multiplatform Application uses Kotlin 1.6.10
Web frontend application using Kotlin/JS with the React UI framework
Build system:
Gradle Kotlin
Gradle Groovy
JDK:
14 java version "14"
Artifact Coordinates
Group ID:
org.example
Artifact ID:
hello-kotlin-js
Version:
1.0-SNAPSHOT
Cancel
Next


New Project
Project
react JS Browser Module
Use kotlin.test
Adds kotlin.test annotations and assert functions for unit testing
Use new JS IR compiler
Learn more about the features available in the new Kotlin/JS IR compiler
Use react-router-dom
Adds navigational components to keep your URLs and UIs synchronized
Use react-redux
Adds state containers powered by Redux

multiplatform libraries

COMMON
KOTLIN

KOTLIN JVM
KOTLIN ANDROID
KOTLIN JS
KOTLIN NATIVE



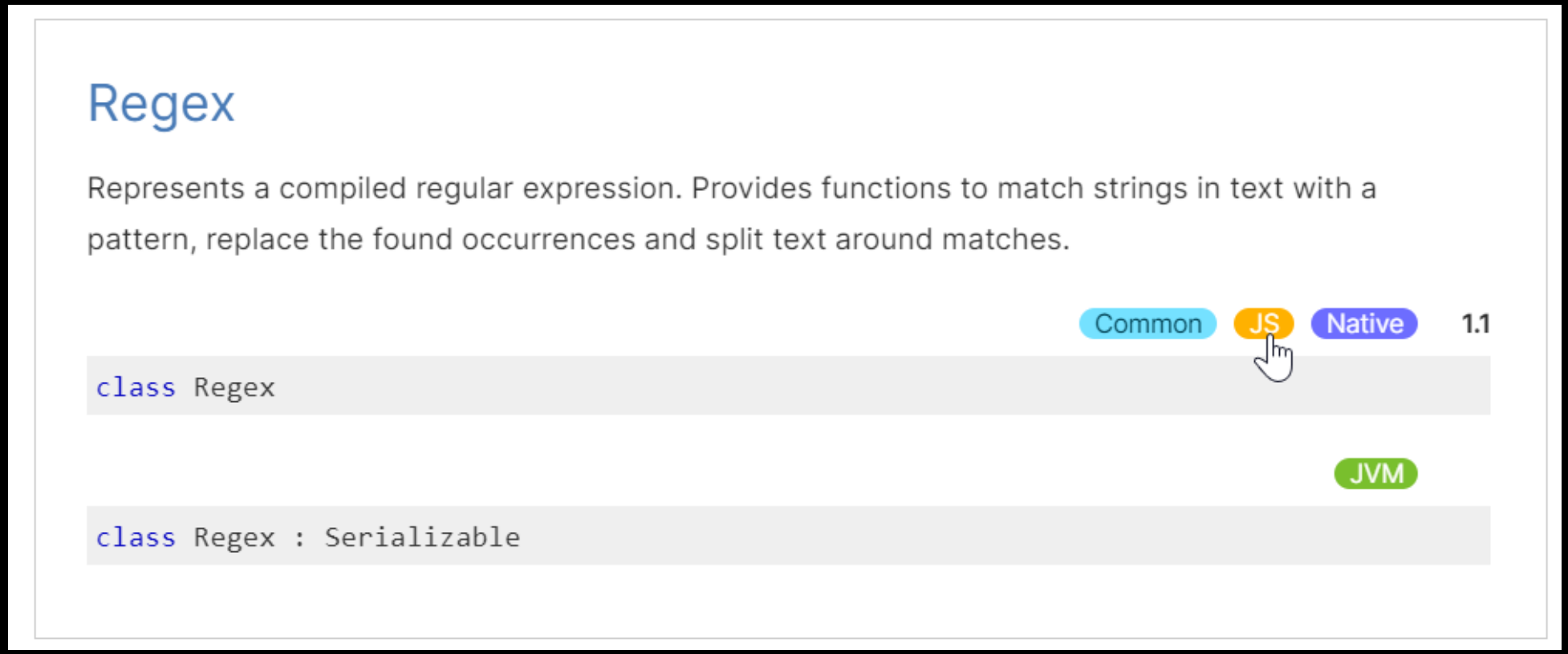
NATIVE ARTEFACT

JS BUNDLE

JAR

standard library documentation

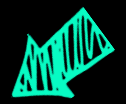
excellent compatibility docs



configuring the kotlin gradle dsl

```
plugins {
    kotlin("js") version "1.6.21"
    kotlin("plugin.serialization") version "1.6.21"
}

group = "org.example"
version = "1.0-SNAPSHOT"

val ktorVersion = "2.0.2"

repositories { mavenCentral() }

dependencies { ... }
```

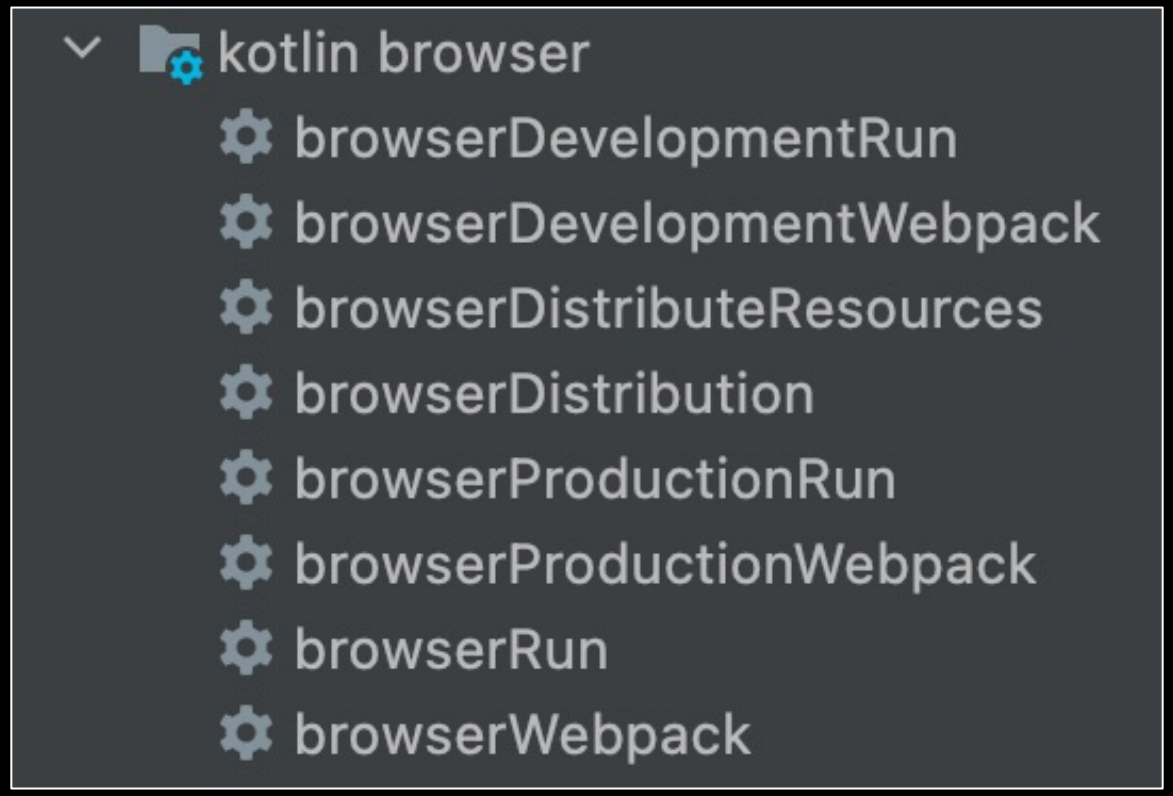
configuring the kotlin gradle dsl



There is a new IR compiler

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

tasks in the kotlin gradle dsl



add multiplatform and kotlinjs packages

kotlinjs, multiplatform and npm

```
dependencies {
    implementation("org.jetbrains.kotlin-wrappers:kotlin-react:17.0.2-pre.290-kotlin-1.6.10")
    implementation("org.jetbrains.kotlin-wrappers:kotlin-react-dom:17.0.2-pre.290-kotlin-1.6.10")
    implementation("org.jetbrains.kotlin-wrappers:kotlin-react-router-dom:6.3.0-pre.340-compat")
    implementation("org.jetbrains.kotlin-wrappers:kotlin-react-css:17.0.2-pre.290-kotlin-1.6.10")

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

what’s the code like

standard main function

```
fun main() {

    val container = document.createElement("div")
    document.body!!.appendChild(container)

    val app = App.create()
    render(app, container)
}
```

what's the code like

easy access to browser apis

```
fun main() {

    val container = document.createElement("div")
    document.body!!.appendChild(container)

    val app = App.create()
    render(app, container)
}
```

what's the code like

easy access to react library

```
val App = FC<Props>("Application") {
    BrowserRouter {
        div {
            p { Link {  to = "/hello" +"Simple Component" } }
        }
        div {
            Routes {
                Route {
                    // …
                }
            }
        }
}
```

Round 1

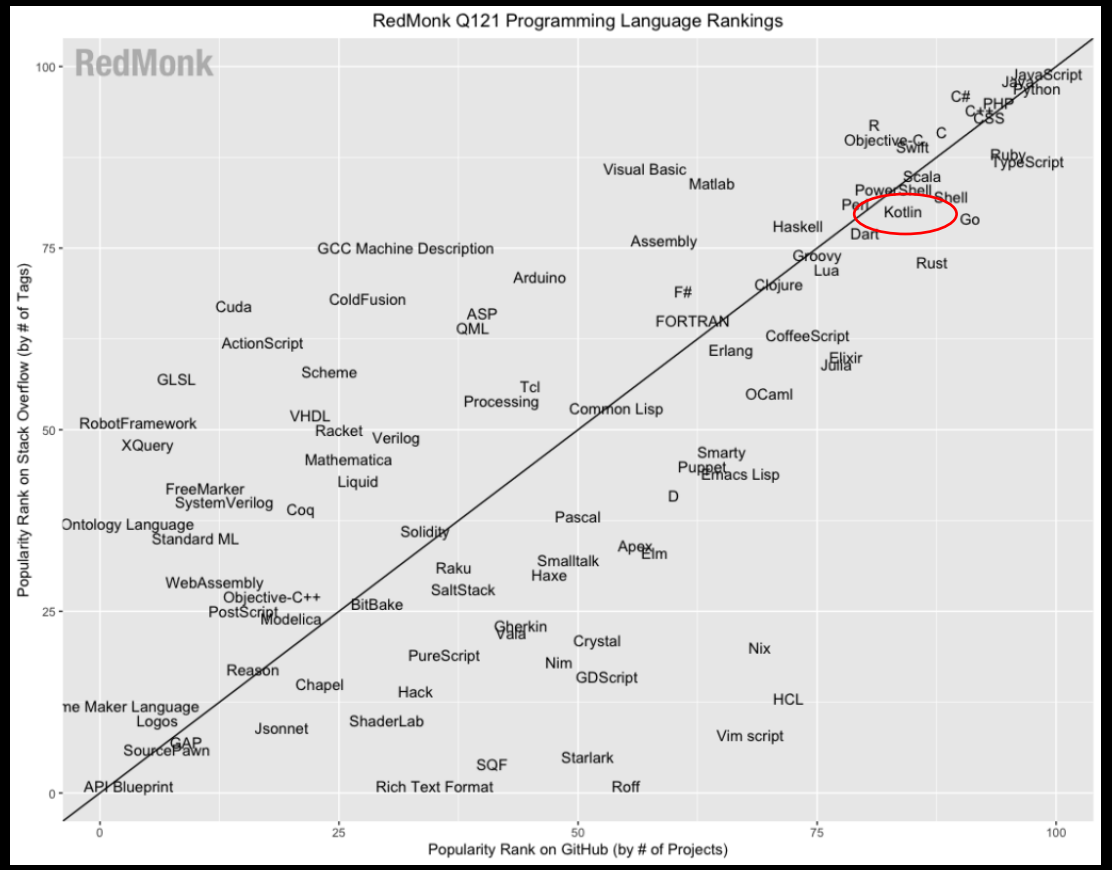
community

its popularity continues to grow

rising star on language rankings

The RedMonk Programming Language Rankings: January 2022

https://redmonk.com/sogrady/2022/03/28/language-rankings-1-22/



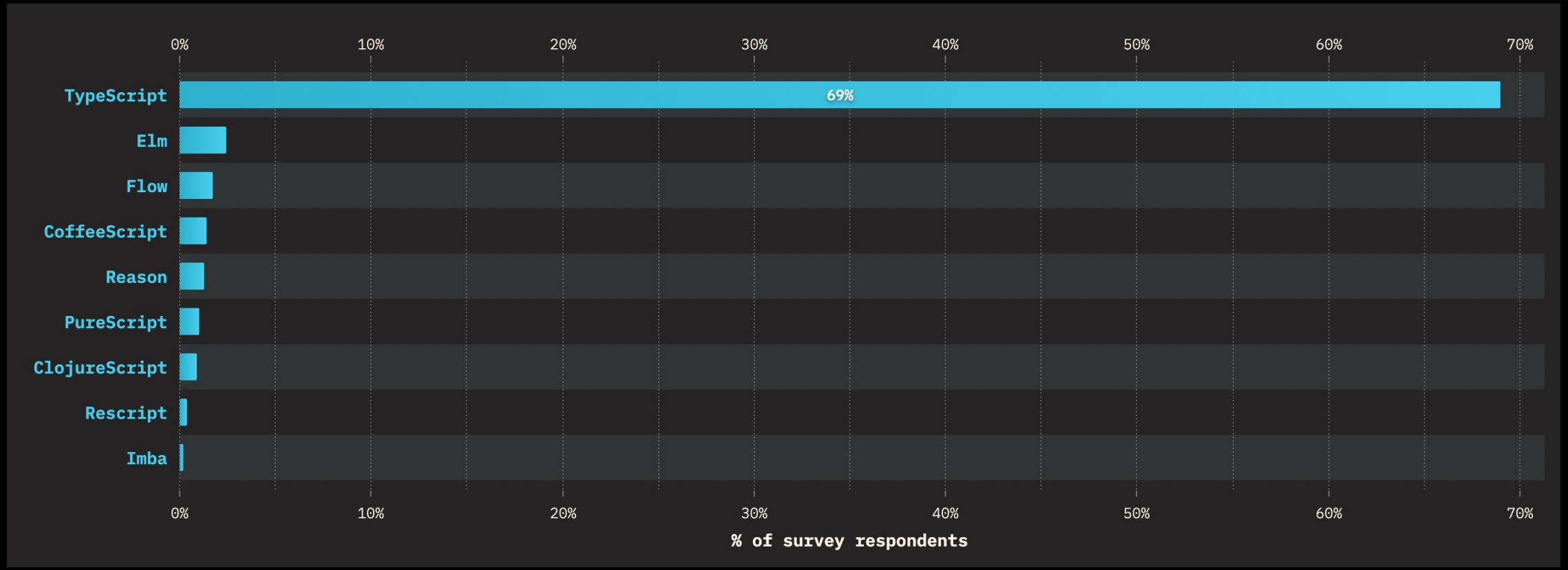
…

18 Kotlin

…

TS is the largest alternative to JS

https://2021.stateofjs.com/en-US/



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


```
export function getString(): string;

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

```
external fun getString(): String

external fun setString(input: String)
```

dukat

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**

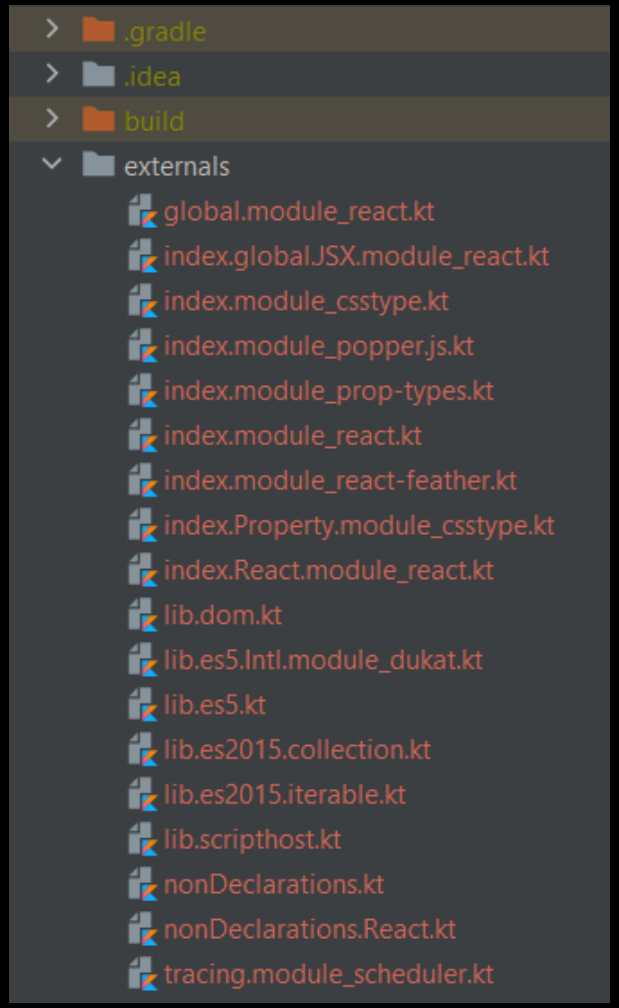
```
$ dukat index.d.ts index.module_my-library.kt
```

dukat

usage

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

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



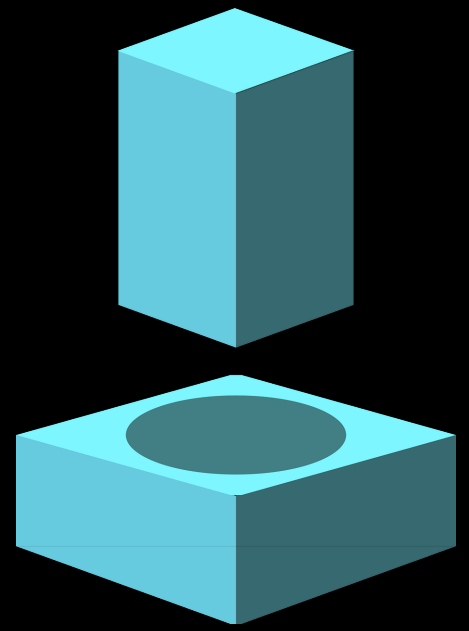
```ts
export interface BasicInterface {
    readonly field1: number;
    method1(): boolean;
}

export function buildInterface(): BasicInterface;

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

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

external fun buildInterface(): BasicInterface

typealias ReadOnlyBasicInterface = Readonly<BasicInterface>
```

dukat

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)
```

jso

object literals in kotlinjs

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

```
Block(jso {
    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

```
export const TaskItem: FC<Props> = (props) =>
    <tr key={props.taskIndex}>
        <td>{props.task.text}</td>
        <td>
            <span onClick={props.onToggle}>
                {pickIcon(props.task.done)}
            </span>
        </td>
    </tr>
```

JSX embeds markup
inside JS / TS code

```
val TaskItem = functionalComponent<TaskItemProps>("TaskItem") { props ->
    tr {
        td { +props.task.text }
        td {
            span {
                +pickIcon(props.task.done)
            }
            attrs {
                onClickFunction = { props.onToggle() }
            }
        }
    }
}

fun RBuilder.TaskItem(task: Task) = child(TaskItem) {
    attrs {
        this.task = task
    }
}
```

In Kotlin a DSL provides equivalent functionality

kotlin DSL's are very cool

a major advantage

```
val TaskItem = functionalComponent<TaskItemProps>("TaskItem") { props ->
    tr {
        td { +props.task.text }
        td {
            span {
                +pickIcon(props.task.done)
            }
            attrs {
                onClickFunction = { props.onToggle() }
            }
        }
    }
}

fun RBuilder.TaskItem(task: Task) = child(TaskItem) {
    attrs {
        this.task = task
    }
}
```

JSX is clearly a lot simpler / shorter

Then they rewrote it
(without telling anyone)

`kotlin-react` was split into two parts: `kotlin-react` and `kotlin-react-legacy`

`kotlin-react` only supports the new DSL for React elements (`ChildrenBuilder`, aka "no attrs"), while `kotlin-react-legacy` provides the familiar `RBuilder` DSL.

If you are migrating from an earlier version and are not interested in migrating to the new API, you should **replace** the `kotlin-react` dependency with `kotlin-react-legacy` in your project.

If you are migrating from an earlier version and would like to gradually migrate to the new API, you should **add** the `kotlin-react-legacy` dependency to your project.

If you are migrating from an earlier version and would like to migrate to the new API at once, resolve all the compilation errors you encounter. Good luck :)

```
val TaskItem = FC<TaskItemProps>("TaskItem") { props ->
    tr {
        td { +props.task.text }
        td {
            onClick = { props.onToggle() }
            span {
                +pickIcon(props.task.done)
            }
        }

    }
}
```

```
val TaskItem = FC<TaskItemProps>("TaskItem") { props ->
    tr {
        td { +props.task.text }
        td {
            onClick = { props.onToggle() }
            span {
                +pickIcon(props.task.done)
            }
        }

    }
}
```

attrs section is gone

Many attributes are cleaner

Builder functions not required

The typing is ... imperfect

```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

```ts
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

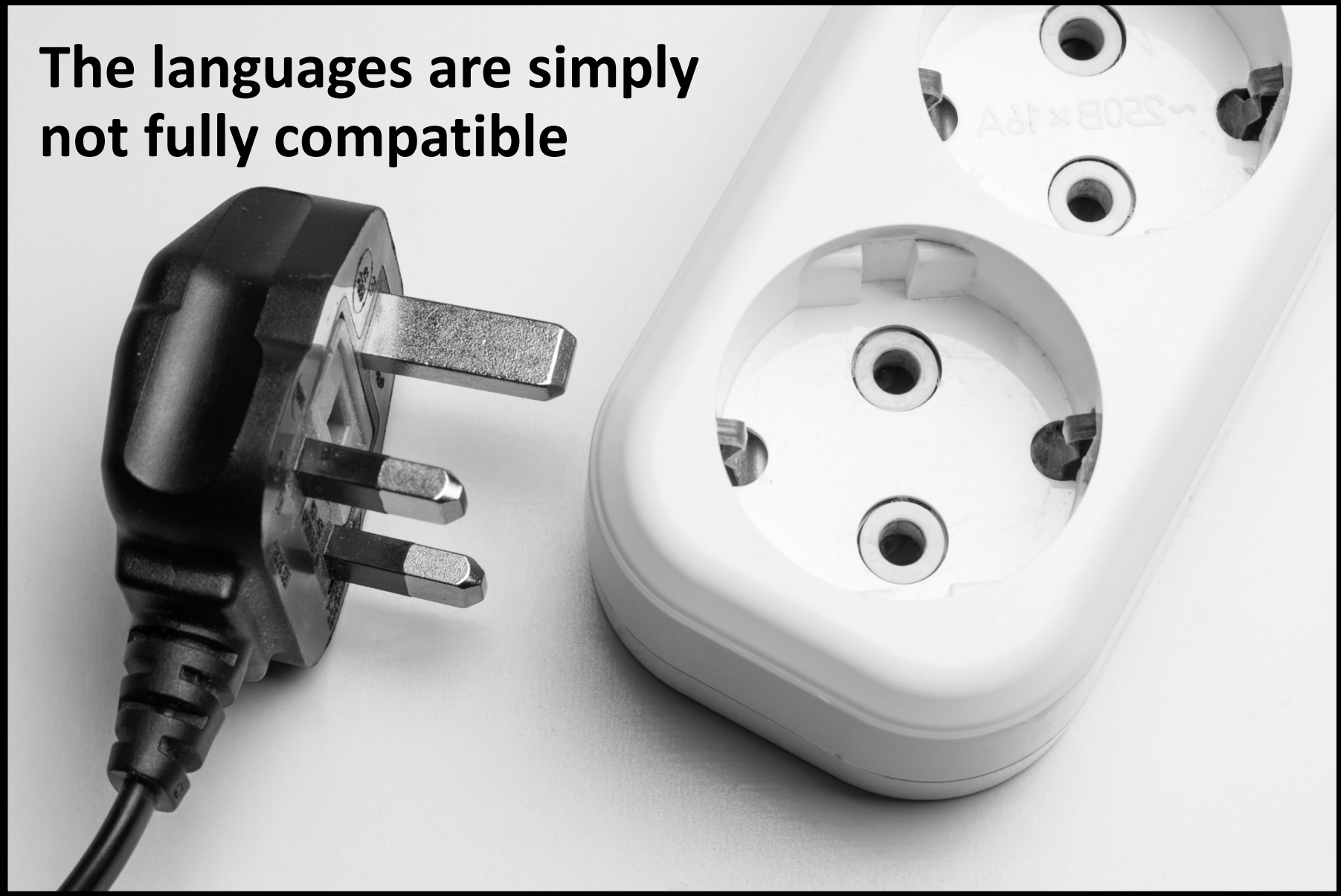
mapped and conditional types

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

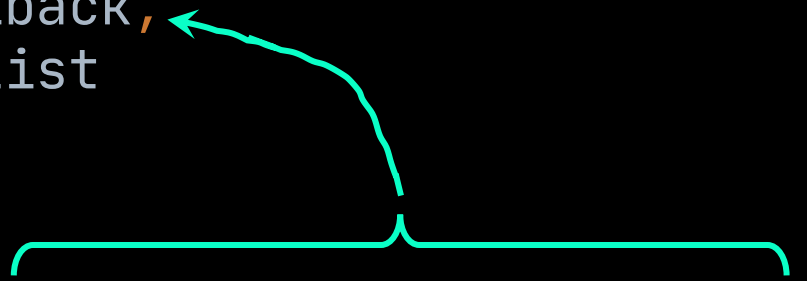
The languages are simply not fully compatible



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
) {
    // ...
}
```

useEffect

or other patterns

```
useEffect { }

useEffect(dep1, dep2) {  }

useEffectOnce { }

useEffectOnce {
    cleanup {
        // Clean up logic goes here
    }
}
```

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

```
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

```kotlin
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;
```

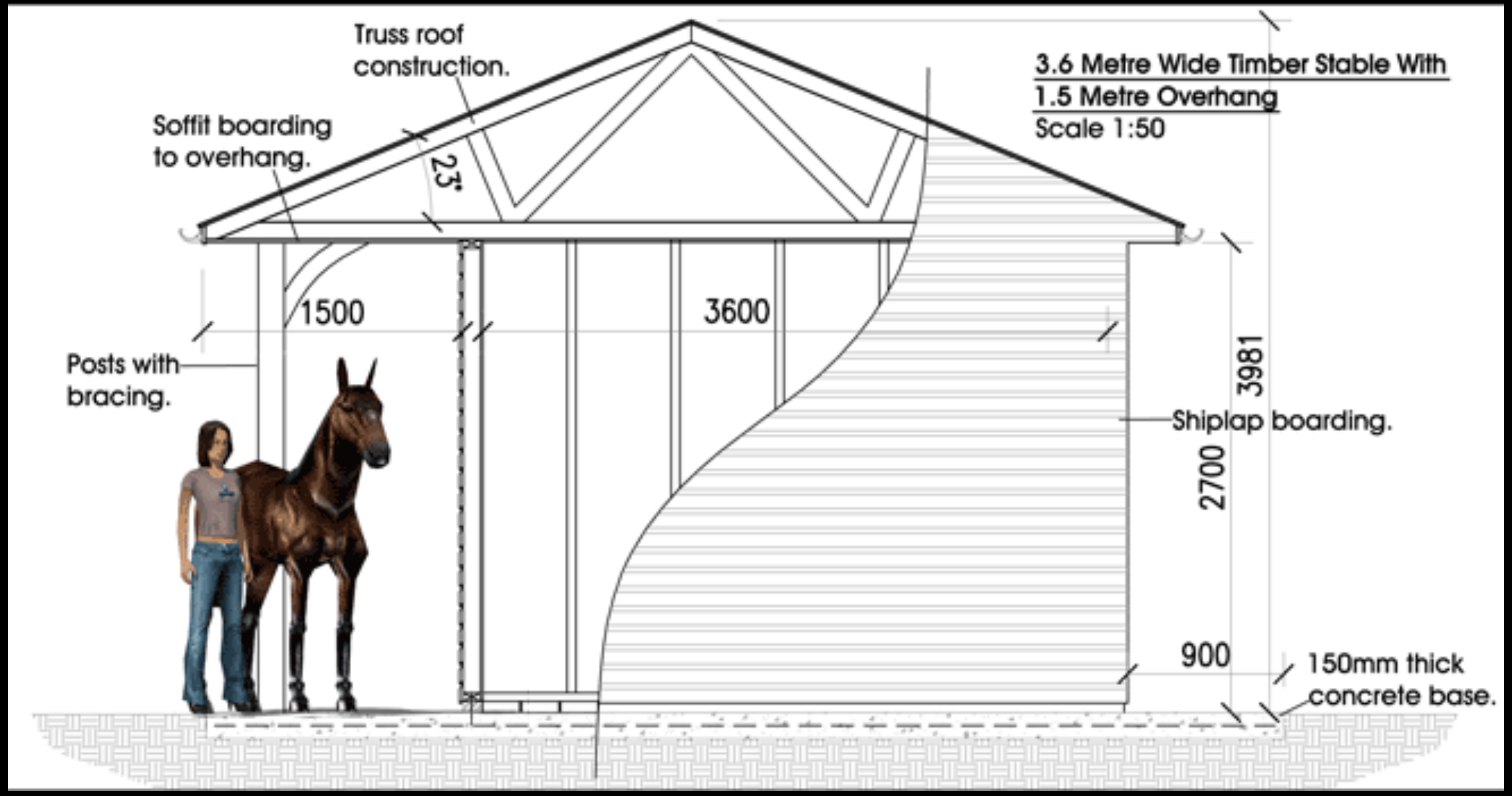
conclusion

they’re both very good

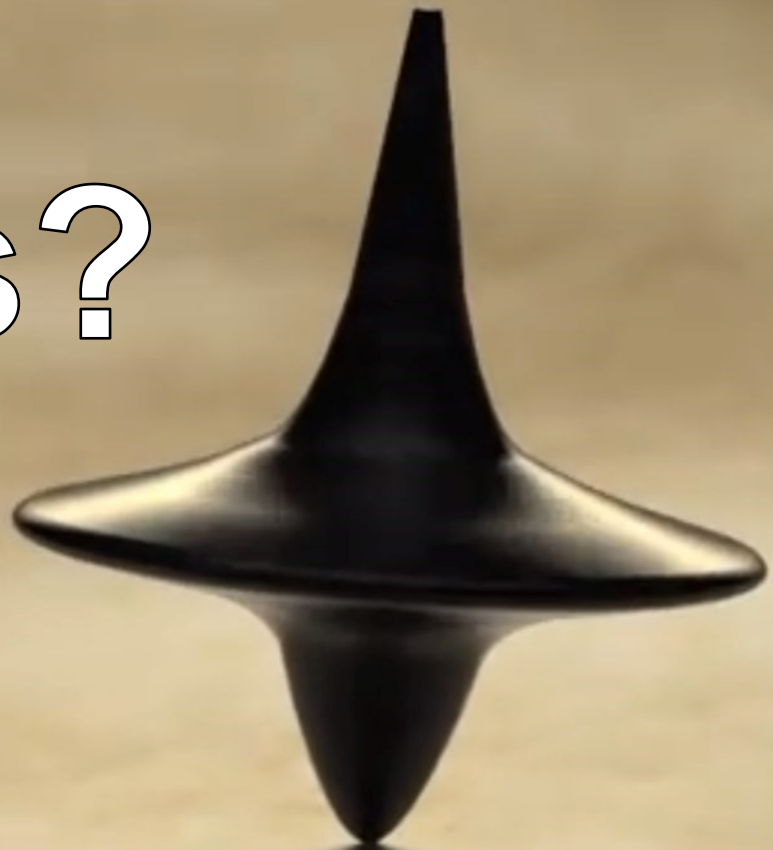
but in different ways


TS
Community
Coroutines
TS
JSX vs React DSL
Multiplatform
TS
Advanced Type System
Extensions
TS
Interop with JS
Expressions
TS
Destructuring
Standard Library
TS
Tooling
Functions

favour stable engineering



Questions?



GOTO
Guide

Remember to **rate this session**

THANK YOU!

#GOTOams