Lean backends using functional Kotlin

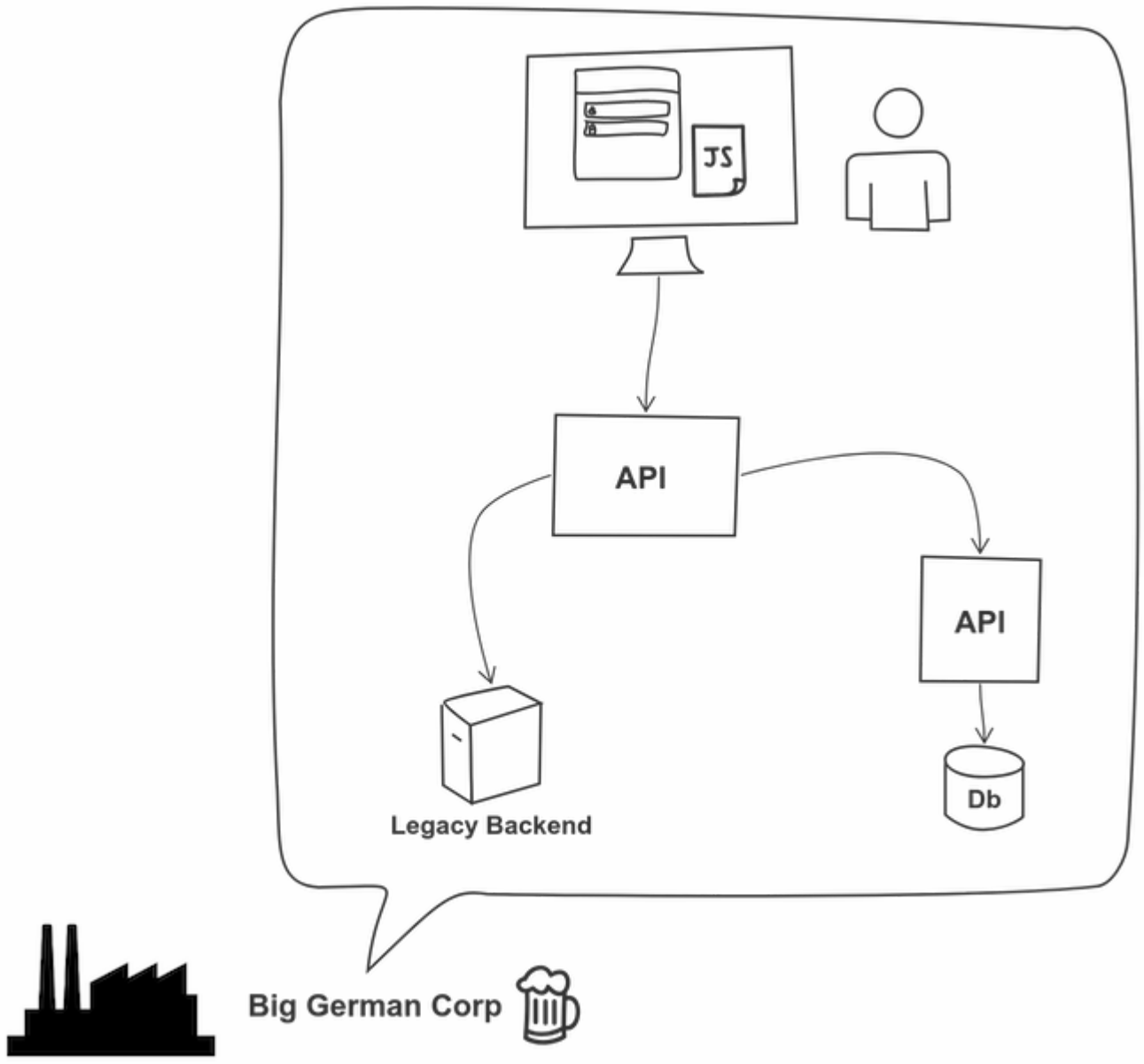
What to expect from this talk

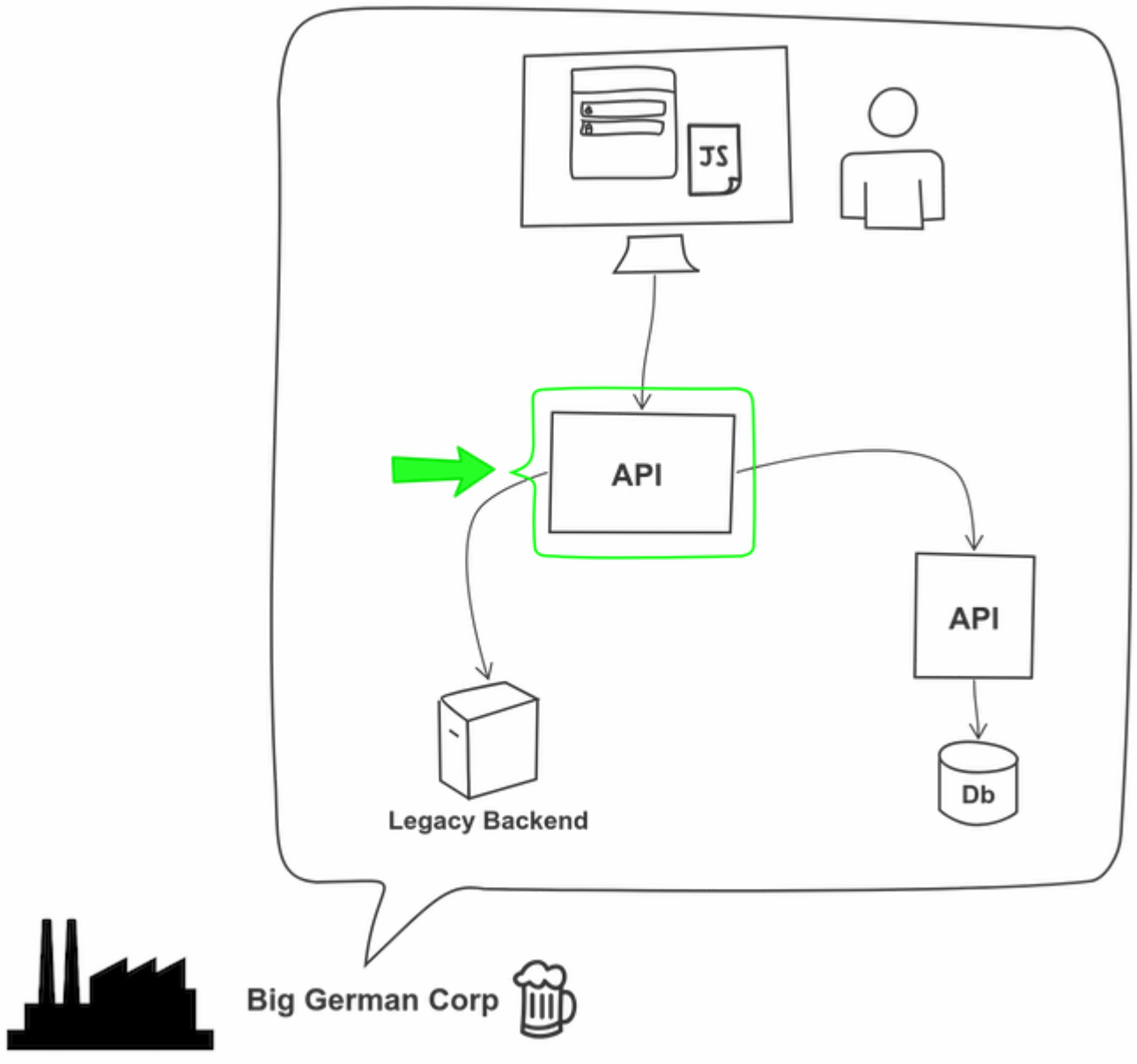
Mario Fernandez

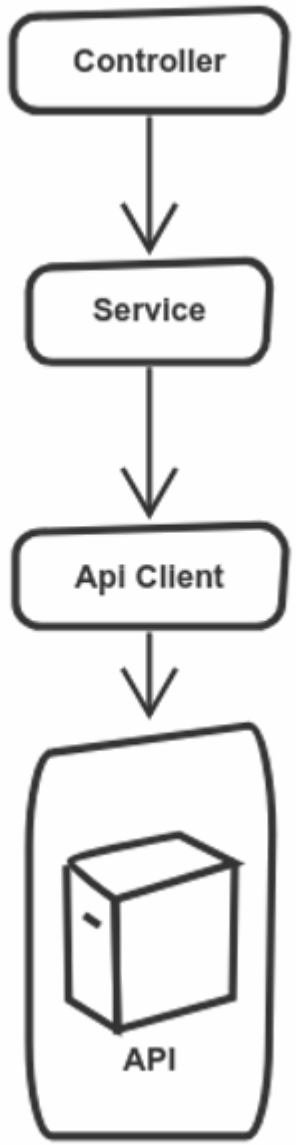
Andrei Bechet

ThoughtWorks

Let's start with some context


JS
API
API
Db
Legacy Backend
Big German Corp


JS
API
API
Db
Legacy Backend
Big German Corp


Controller
Service
Api Client
API

Technologies that we will be mentioning


Kotlin


STRIKT


ARROW


Spring
Boot

Our pain points

I don't know the state of my data

I don't know the state of my data

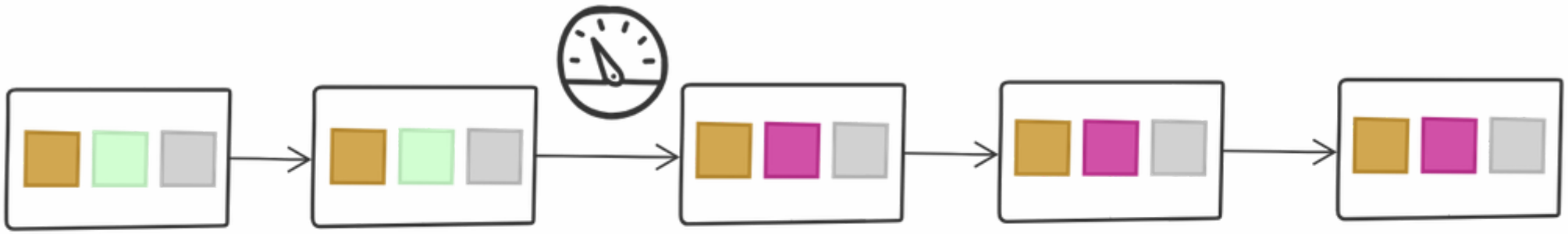
Half of my time I'm just dealing with *null* values

I don't know the state of my data

Half of my time I'm just dealing with *null* values

The other half I'm debugging the 500s thrown by our application


Uncontrolled Data
Dealing with Null
Unhandled Exceptions



Wait, when did this actually change?

Immutability

```
data class TokenAuthentication(
        val id: Id,
        val firstName: FirstName,
        val lastName: LastName,
        val scopes: List<Scope>,
        val expiresAt: LocalDateTime
)
```

```
val scopes: List<Scope> = buildScopes(token)

// List is immutable, it won't compile
scopes.removeAt(1) ❌

// Creates a new list
scopes.filter { it.isAdmin } ✅
```

JSON

```
@JsonIgnoreProperties(ignoreUnknown = true)
data class TokenAuthentication(
        val id: Id,
        @JsonAlias("name_first")
        val firstName: FirstName,
        @JsonAlias("name_last")
        val lastName: LastName,
        @JsonDeserialize(converter = ListSanitizer::class)
        val scopes: List<Scope>,
        val expiresAt: LocalDateTime
)
```

Dealing with change

```
// Will create a new object
fun TokenAuthentication.clearScopes() = copy(scopes = emptyList())
```

```
expectThat(token) {
    get { name }.isEqualTo("google-oauth2|3234123")
    get { authorities.map { it.authority } }.contains("create:recipes")
}
```

```
org.opentest4j.AssertionFailedError:
▼ Expect that Some(TokenAuthentication@52789c41):
  ▼ TokenAuthentication@52789c41:
    Authenticated: true;
    Authorities: profile, create:recipes:
    ▼ name:
      ✗ is equal to "google-oauth2|3234123" : found "google-oauth2|dude"
```

Why Immutability?

Easier to reason

Always in a valid state

Can be shared freely

Immutability

```
public static boolean isAdmin(List<Scope> scopes) {
    if(scopes == null) {
      return false;
    }

    Scope adminScope = findAdminScope(scopes);

    if(adminScope == null) {
      return false;
    }

    return adminScope.isValid();
  }
```

The billion dollar mistake

> I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

Nullable types

```
Authorization: Bearer bGciOi...JIUzI1NiIs
```

```
Authorization: Bearer bGciOi...JIUzI1NiIs
```

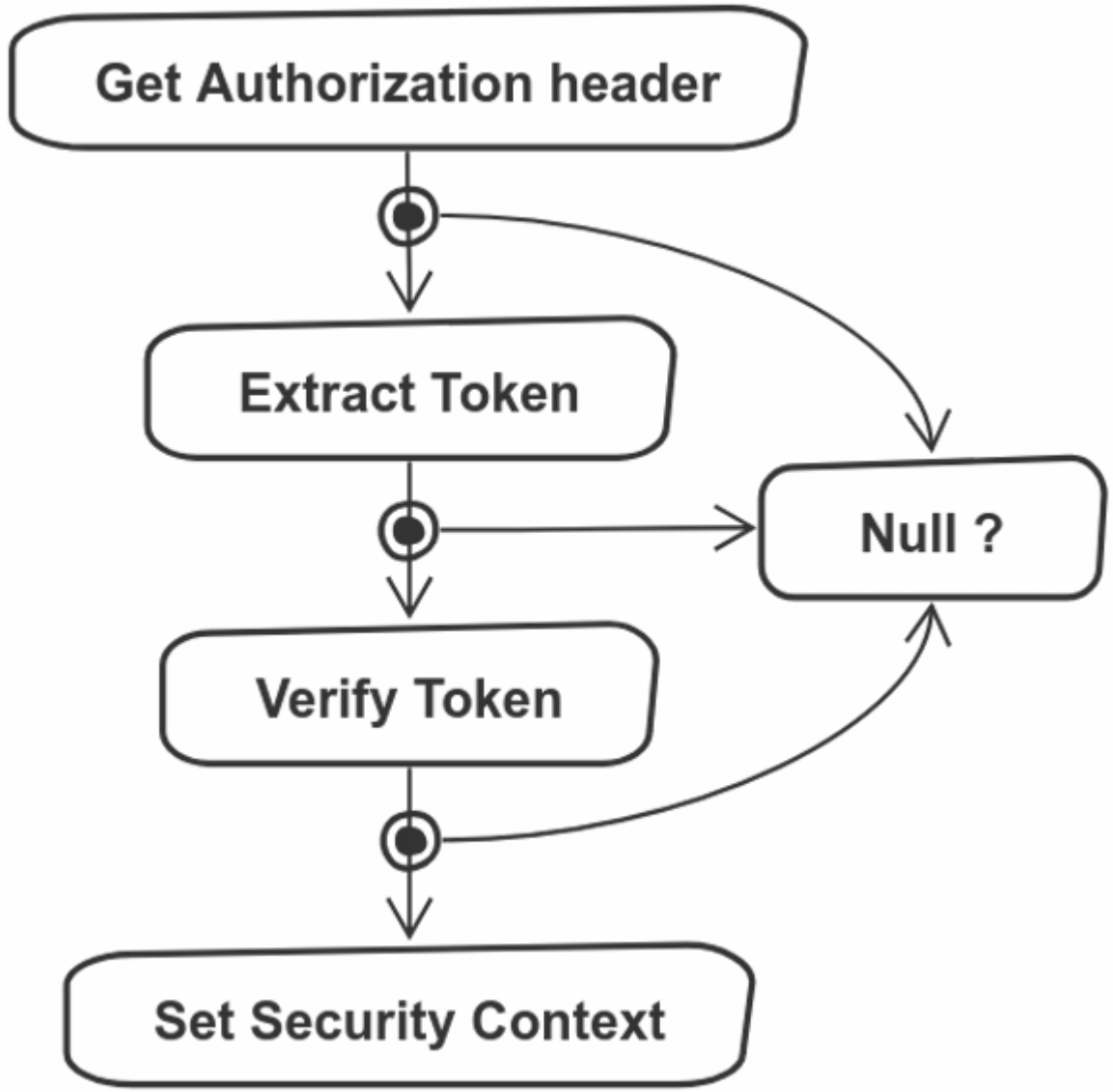
```
fun String.extractToken(): String? = if (startsWith("Bearer"))
    split(" ").last()
else
    null
```

```
Authorization: Bearer bGciOi...JIUzI1NiIs
```

```
fun String.extractToken(): String? = if (startsWith("Bearer"))
    split(" ").last()
else
    null
```

```
header.extractToken()
  ?.let { token -> doStuff(token) }
```

It can get messy


Get Authorization header
Extract Token
Null ?
Verify Token
Set Security Context

```
request.getHeader(Headers.AUTHORIZATION)
  ?.let { header ->
    header.extractToken()
      ?.let { jwt ->
        verifier.verify(jwt)
          ?.let { token ->
            SecurityContextHolder.getContext().authentication = token
          }
      }
  }
```

```
function hell(win) {
 // for listener purpose
 return function() {
   loadLink(win, REMOTE_SRC+'/assets/css/style.css', function() {
     loadLink(win, REMOTE_SRC+'/lib/async.js', function() {
       loadLink(win, REMOTE_SRC+'/lib/easyXDM.js', function() {
         loadLink(win, REMOTE_SRC+'/lib/json2.js', function() {
           loadLink(win, REMOTE_SRC+'/lib/underscode.min.js', function() {
             loadLink(win, REMOTE_SRC+'/lib/backbone.min.js', function() {
               loadLink(win, REMOTE_SRC+'/dev/base_dev.js', function() {
                 loadLink(win, REMOTE_SRC+'/assets/js/deps.js', function() {
                   loadLink(win, REMOTE_SRC+'/src/' + win.loader_path + '/loader.js', function() {
                     async.eachSeries(SCRIPTS, function(src, callback) {
                       loadScript(win, BASE_URL+src, callback);
                     });
                   });
                 });
               });
             });
           });
         });
       });
     });
   });
 };
}
```

Option Datatype

Datatype? 🤔

A digression about Functional Programming

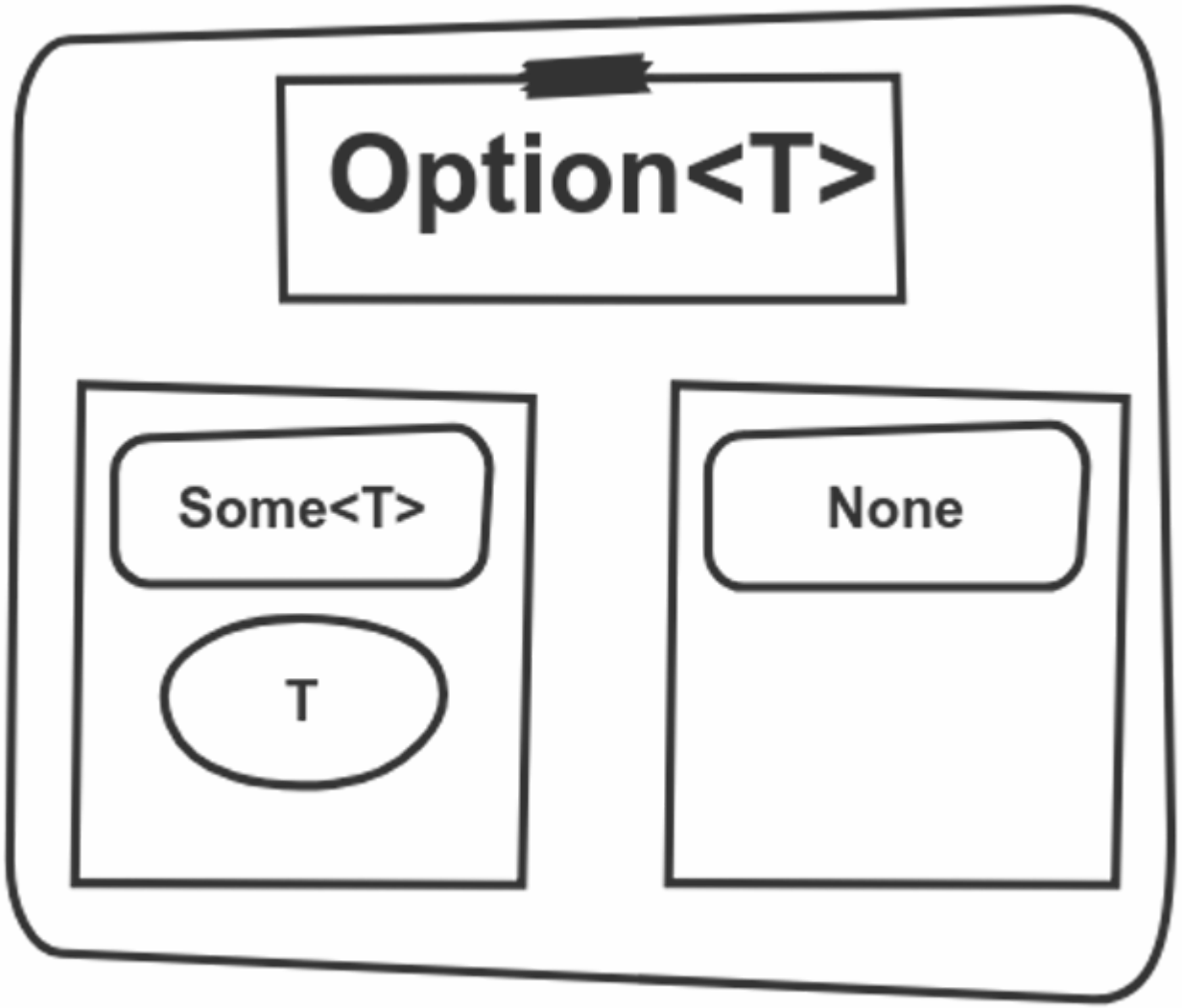
Datatype? 🤔

A datatype is an abstraction that encapsulates one reusable coding pattern.

```
interface Operations {
    fun <A, B> Option<A>.map(f: (A) -> B): Option<B>
    fun <A, B> Option<A>.flatMap(f: (A) -> Option<B>): Option<B>
}
```

hackernoon.com/kotlin-functors-applicatives-and-monads-in-pictures-part-1-3-c47a1b1ce251

A digression about Functional Programming


Option<T>
Some<T>
T
None

```
fun String.extractToken(): String? = if (startsWith("Bearer"))
    split(" ").last()
else
    null
```

```
fun String.extractToken(): Option<String> = startsWith("Bearer ")
        .maybe { split(" ").last() }
```

Let's try our previous example with *Option*

```
request.getHeader(Headers.AUTHORIZATION)
  .toOption()
  .flatMap { header ->
    header.extractToken()
      .flatMap { jwt ->
        verifier
          .verify(jwt)
          .map { token ->
            SecurityContextHolder.getContext().authentication = token
        }
      }
  }
```

Not much of an improvement 😔

Non-nested syntax thanks to arrow

```
Option.fx {
    val (header) = request.getHeader(Headers.AUTHORIZATION).toOption()
    val (jwt) = header.extractToken()
    val (token) = verifier.verify(jwt)
    SecurityContextHolder.getContext().authentication = token
}
```

```
@Test
fun `verify does not work with a invalid jwt token`() {
    expectThat(
      RemoteVerifier(keySet).verify(jwt)
    ).isEmpty()
}
```

Why Option?

Explicit about what can be null

Avoid if-null litter

Compile time checks

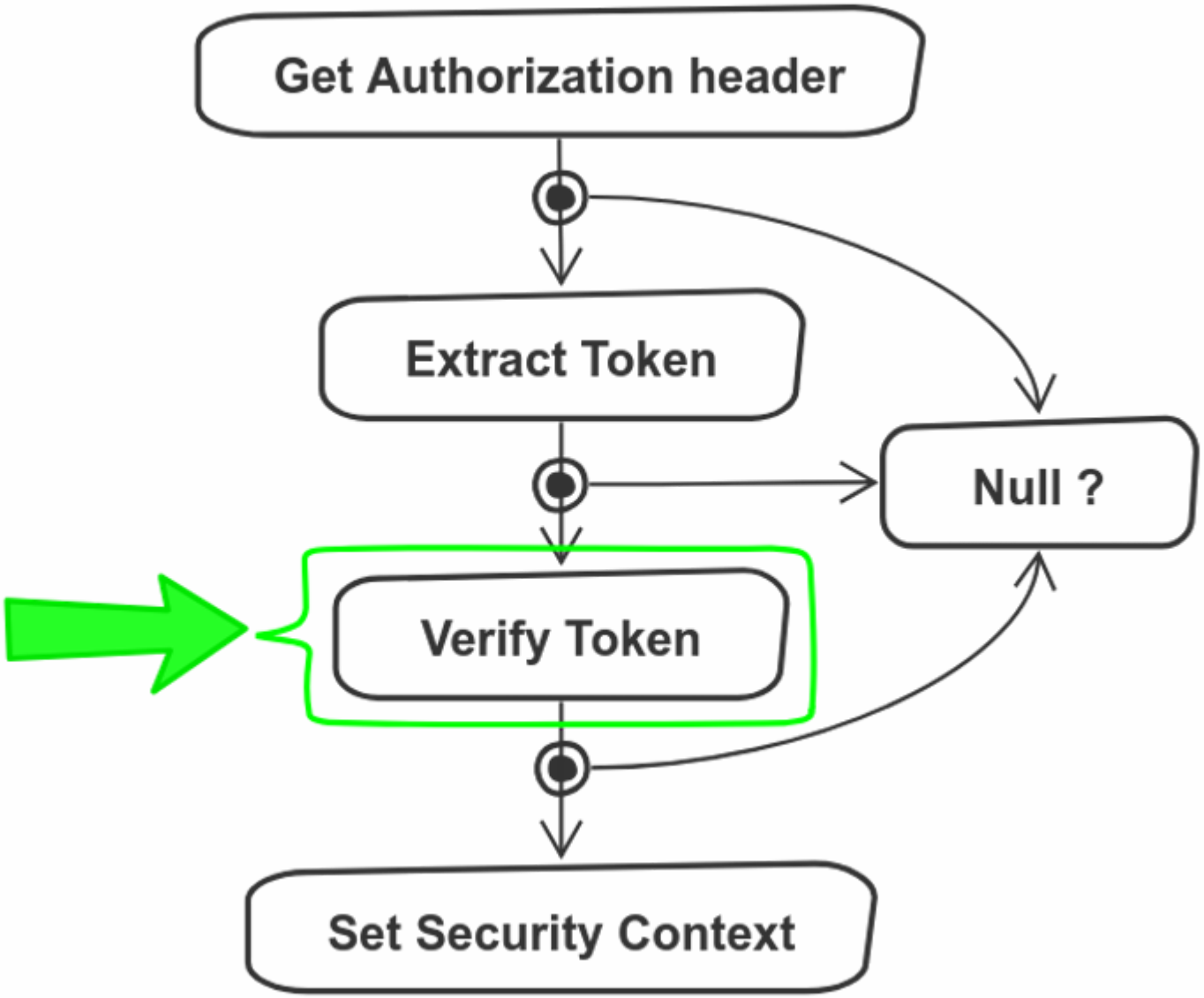
Save a billion dollars 🤓

Immutability

Option

```
com.auth0.jwt.exceptions.JWTDecodeException:
  The string '{"typ":"JWT","alg":"RS256"}' is not a valid token.
      at com.auth0.jwt.impl.JWTParser.convertFromJSON(JWTParser.java:52)
      at com.auth0.jwt.impl.JWTParser.parseHeader(JWTParser.java:33)
      at com.auth0.jwt.JWTDecoder.<init>(JWTDecoder.java:37)
      at com.auth0.jwt.JWT.decode(JWT.java:21)
      at com.auth0.jwt.JWTVerifier.verify(JWTVerifier.java:352)
```

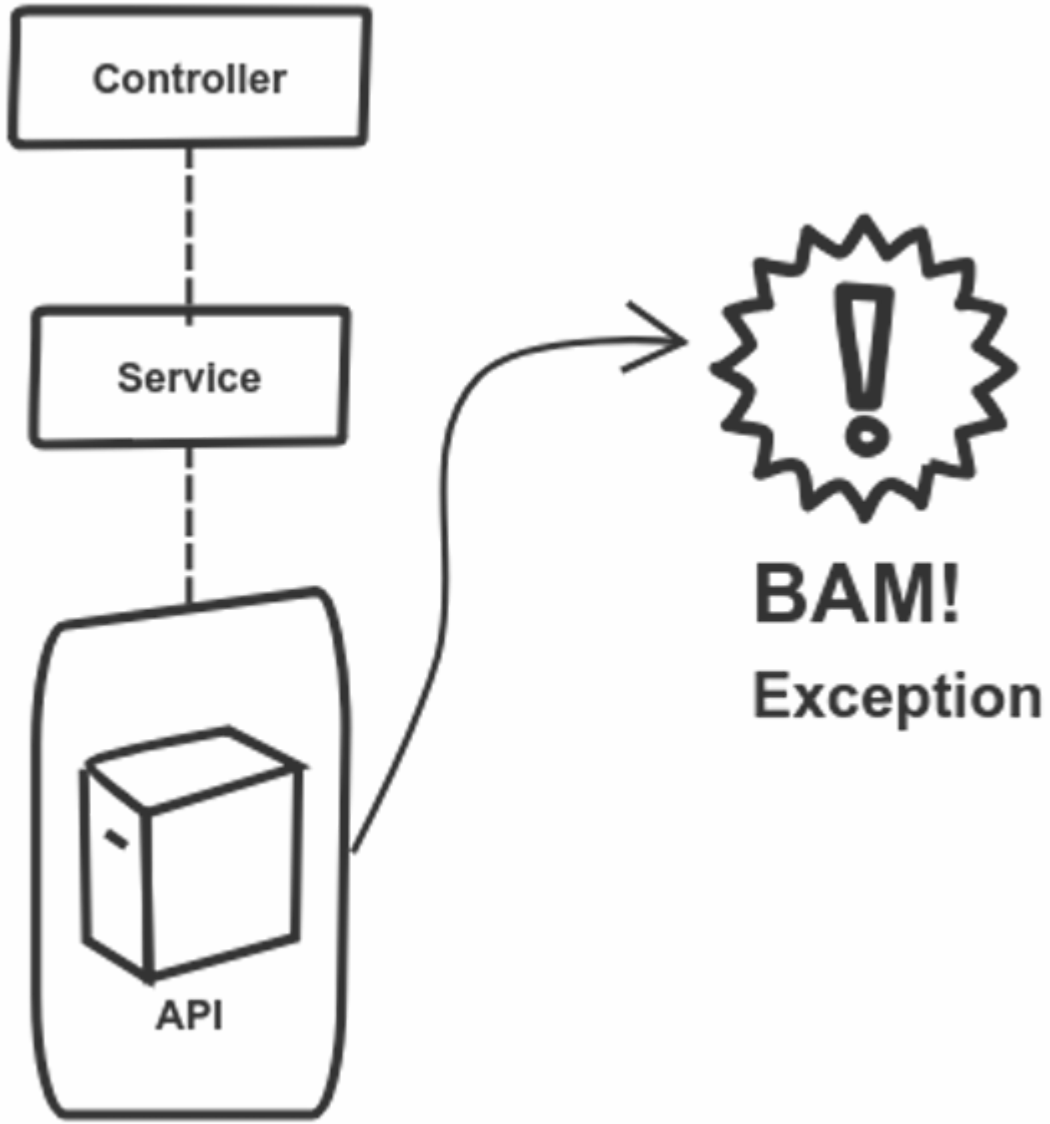
Verifying our token


Get Authorization header
Extract Token
Null ?
Verify Token
Set Security Context

```
interface Verifier {
    fun verify(token: String): TokenAuthentication
}
```

That signature is not quite telling the truth

```java
/**
 * Perform the verification against the given Token
 *
 * @param token to verify.
 * @return a verified and decoded JWT.
 * @throws AlgorithmMismatchException
 * @throws SignatureVerificationException
 * @throws TokenExpiredException
 * @throws InvalidClaimException
 */
public DecodedJWT verifyByCallingExternalApi(String token);
```


Controller
Service
API
BAM!
Exception

Exceptions make the flow implicit

Exceptions force you to be aware of the internal implementation

kotlinlang.org/docs/reference/exceptions.html

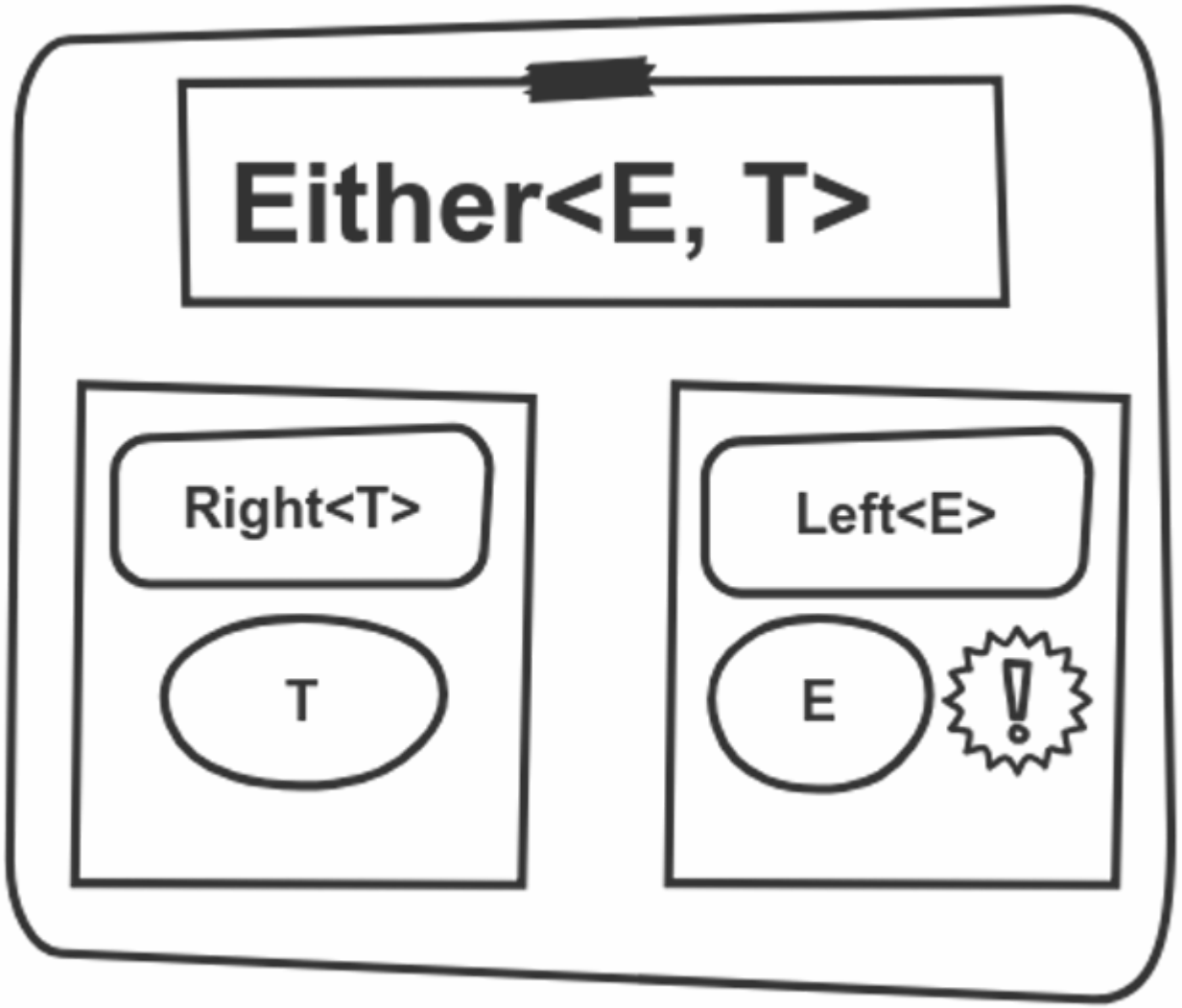
```
@ExceptionHandler(JWTVerificationException::class)
fun handleException(exception: JWTVerificationException):
  ResponseEntity<ErrorMessage> {
    return ResponseEntity
      .status(HttpStatus.BAD_GATEWAY)
      .body(ErrorMessage.fromException(exception))
}
```

Either DataType

Option and Either are quite similar

A digression about Functional Programming

```
interface Operations {
    fun <T, A, B> Either<T, A>.map(f: (A) -> B): Either<T, B>
    fun <T, A, B> Either<T, A>.flatMap(f: (A) -> Either<T, B>):
      Either<T, B>
}
```


Either<E, T>
Right<T>
T
Left<E>
E
!

```
interface Verifier {
    fun verify(token: String):
      Either<JWTVerificationException, TokenAuthentication>
}
```

Isolating the problematic code

```
private fun JWTVerifier.unsafeVerify(token: String) = try {
    verifyByCallingExternalApi(token).right()
} catch (e: JWTVerificationException) {
    e.left()
}
```

Operating with Either

```
override fun verify(token: String)
        : Either<JWTVerificationException, TokenAuthentication> {
    val key = key(keySet)
    val algorithm = algorithm(key)
    val verifier = verifier(algorithm, leeway)
    return verifier
            .unsafeVerify(token)
            .map { it.asToken() }
}
```

```
Either.fx<JWTVerificationException, TokenAuthentication> {
    // Either<Throwable, ResponseEntity<UnprocessedResponse>>
    val response = unsafeRequest()
    val (body) = response
            .mapLeft { JWTVerificationException(it) }
    body.map()
}
```

```
@GetMapping("{id}")
fun recipe(@PathVariable id: Int): ResponseEntity<RecipeDetails> {
    return when (val result = repository.find(id)) {
        is Either.Left -> ResponseEntity.status(result.a).build()
        is Either.Right -> ResponseEntity.ok(result.b)
    }
}
```

```kotlin
@Test
fun `verify works if the expiration is not taken into account`() {
    val hundredYears = 3600L * 24 * 365 * 100
    val verifier = RemoteVerifier(keySet, hundredYears)

    expectThat(verifier.verify(jwt)).isRight().and {
        get { name }
          .isEqualTo("google-oauth2|111460419457288935787")
        get { authorities.map { it.authority } }
          .contains("create:recipes")
    }
}
```

Result

kotlin-stdlib

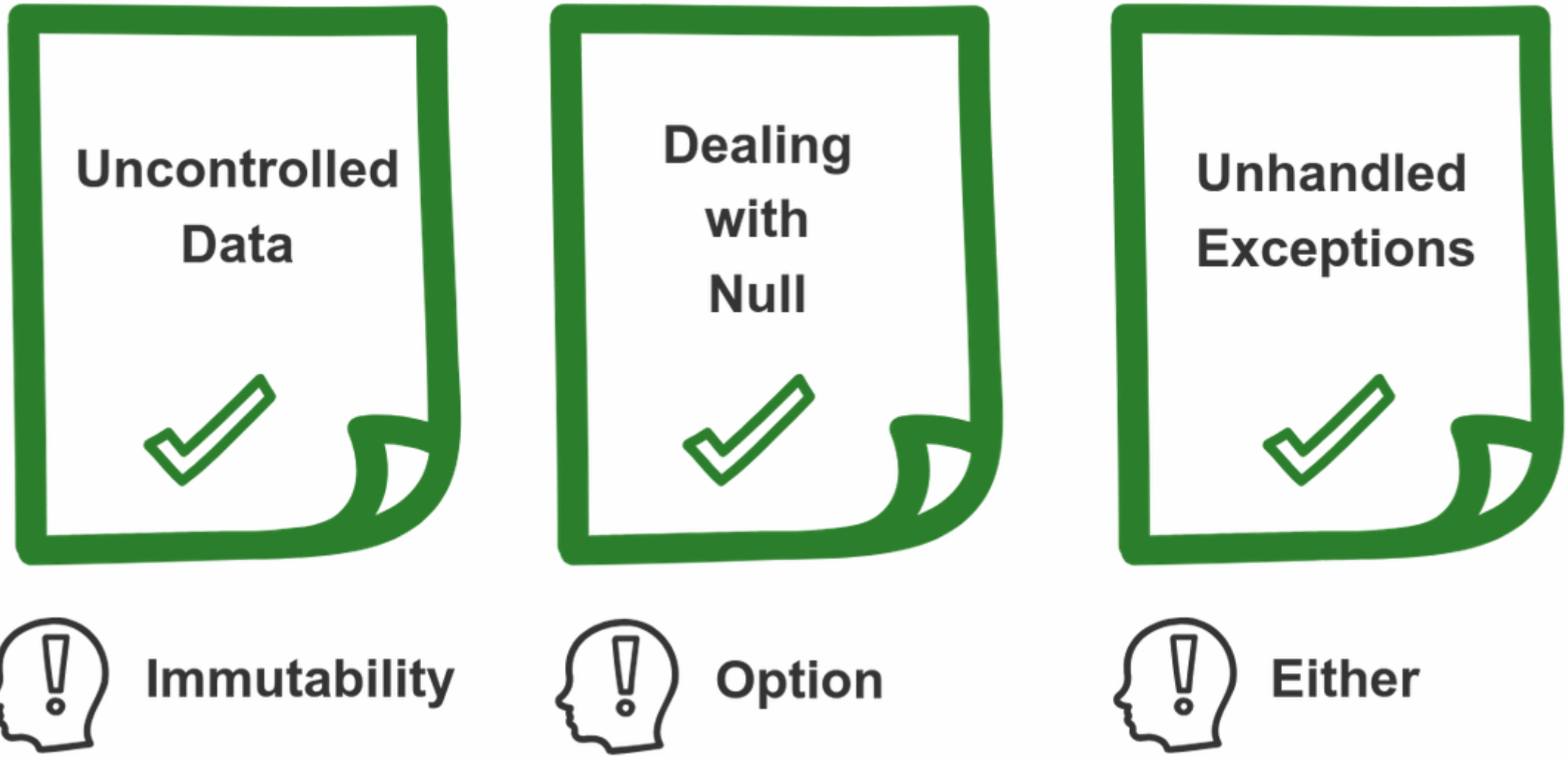
```
fun unsafeOp() =
        runCatching {
            doStuff()
        }.getOrElse { exception -> handle(exception) }
```

Why Either?

Makes flow explicit

Interface tells the whole truth

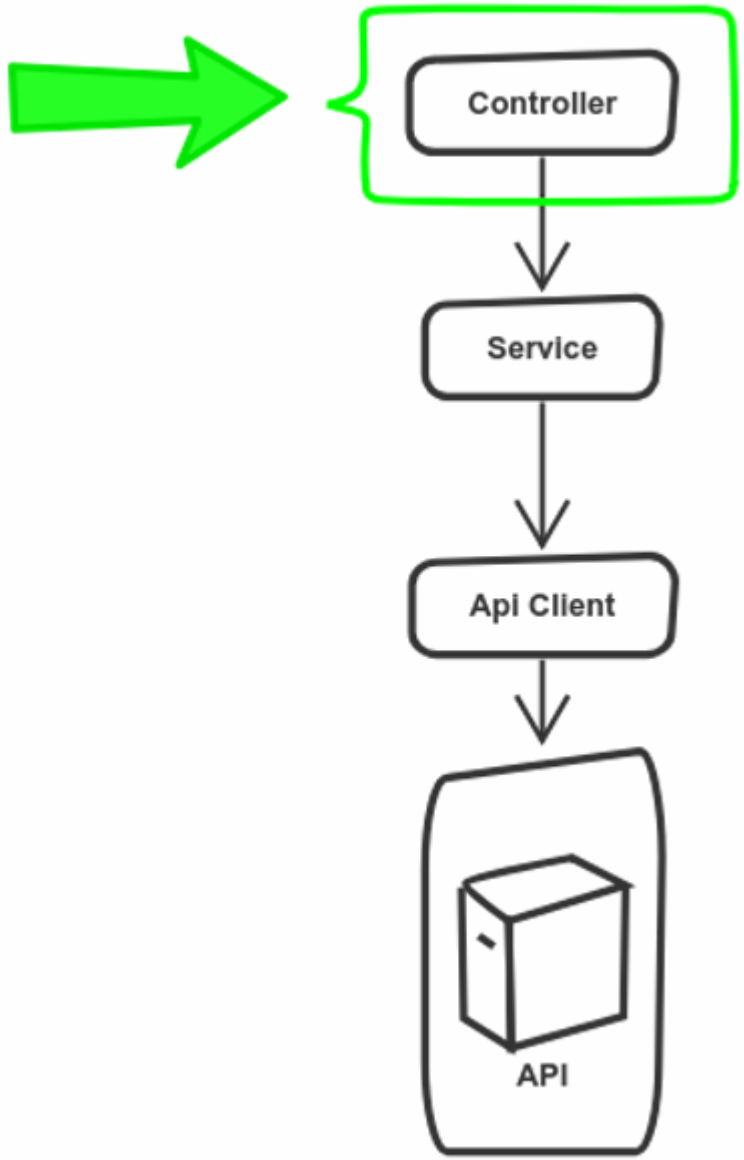
Compile time checks


Uncontrolled Data
Dealing with Null
Unhandled Exceptions
Immutability
Option
Either

What comes next?

Purely functional code


Edge of the World

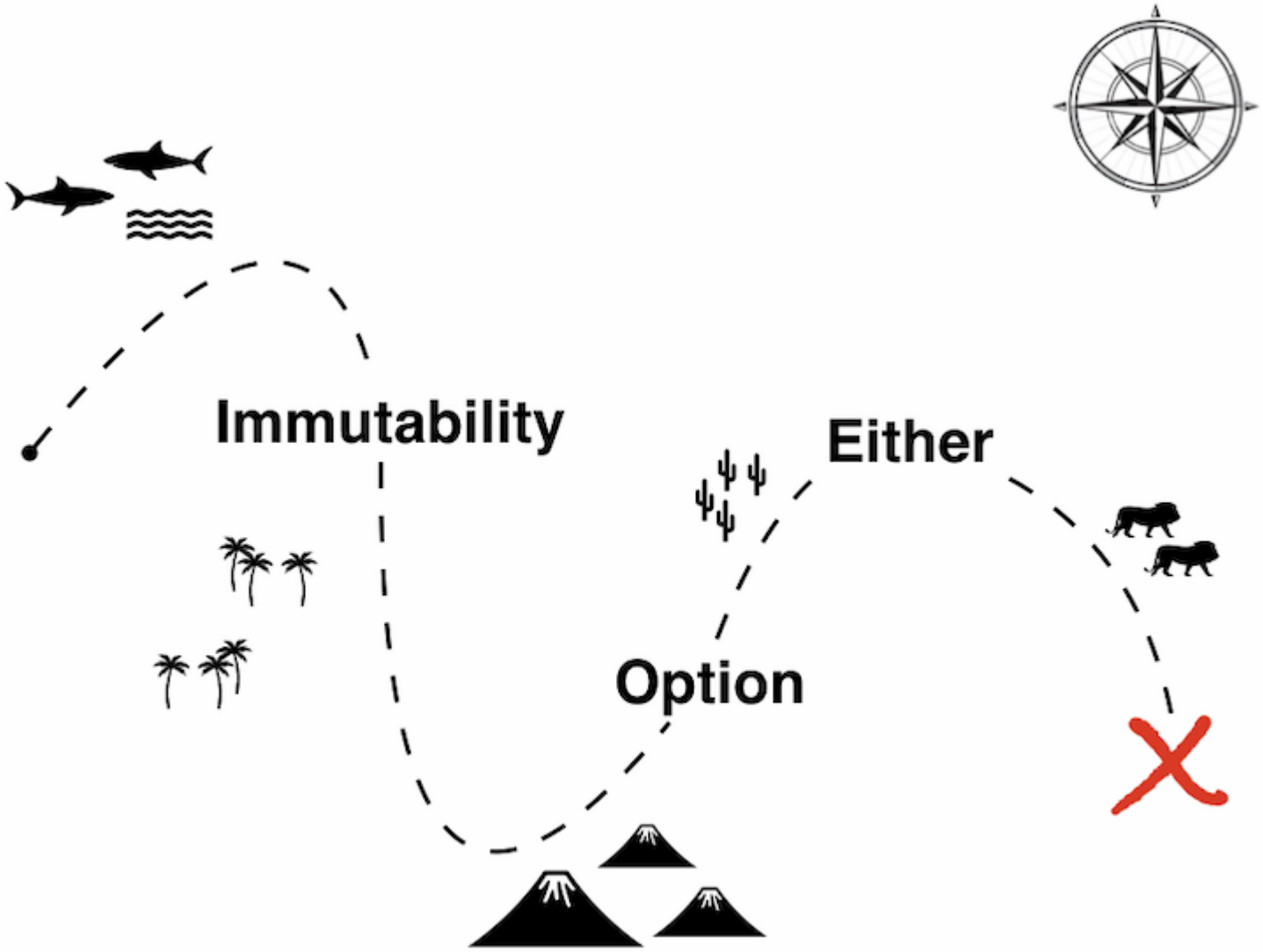

Controller
Service
Api Client
API

IO

```
IO<Either<JWTVerificationException, TokenAuthentication>>
```

We are hitting the limit of what's convenient to do with Kotlin and Arrow here

Wrap Up


Immutability
Option
Either

Backend

At your own pace!

JOIN OUR COMMUNITY

26 years **experience**

42 offices **in 13 countries**

Thought leader **in agile software development and continuous delivery**

6000+ **thoughtworkers worldwide**

300+ **thoughtworkers in Germany**

de-recruiting@thoughtworks.com