EAMON SCULLION

KOTLIN AND SPRING BOOT

WHAT IS SPRING BOOT?

SPRING BOOT

- Spring Framework has been around for over 15 years, and with the ease of Spring Boot, it has established itself as the industry standard for enterprise server-side development.
- Difference that you need to know - Spring is an easy way to write Java apps, Spring Boot is an easy way to write Spring apps

JAVA IS THE OBVIOUS LANGUAGE CHOICE, RIGHT?

THE OBVIOUS LANGUAGE CHOICE IS JAVA, RIGHT?

- Java's strength is it's ecosystem and interoperability with modern tools e.g Docker, Kafka, AWS SDK etc
- Any language limitations of Java can be mostly overcome with the top class library available (Project Lombok, Apache Commons, Google Guava)

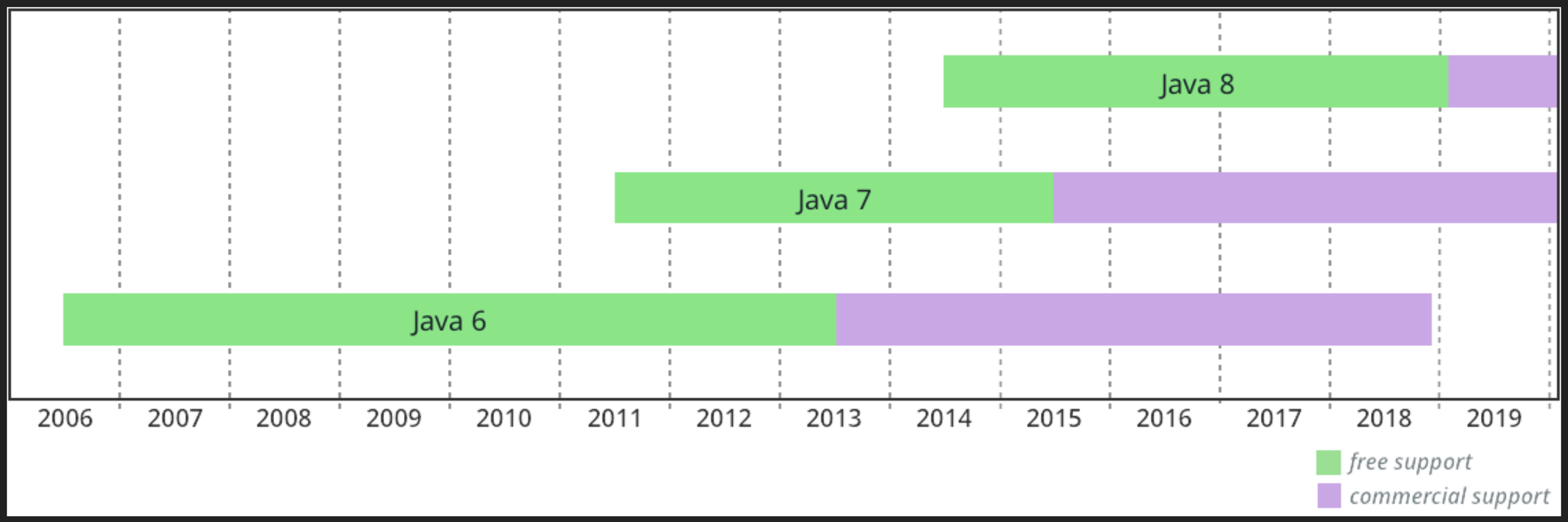
THERE'S ALSO EXCITING FEATURES COMING UP FOR JAVA

- Records - less verbose class definitions - Java's answer to Kotlin Data Classes
  - https://openjdk.java.net/jeps/359
- Project Loom - high-throughput, lightweight threads - Java's answer to Kotlin Coroutines
  - https://wiki.openjdk.java.net/display/loom
- These features are years away from being released

OK, BUT WHAT'S THE CATCH?

JAVA RELEASE TRAIN THEN

- Traditionally Java has been slow to update:

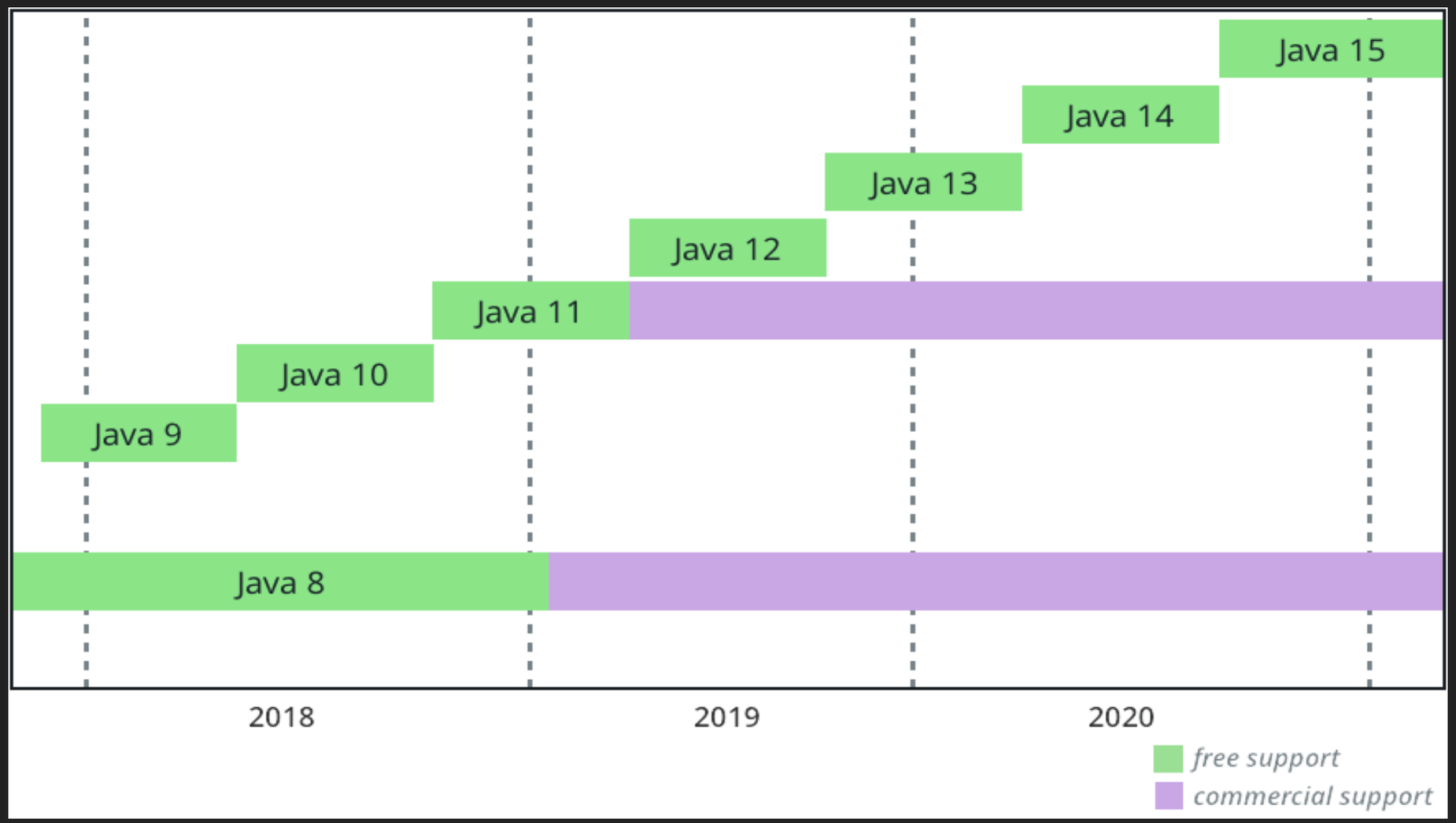


- Big updates would arrive every couple of years, with a lot of thought put into upgrading and backward compatibility.

Source: https://dev.karakun.com/java/2018/06/25/java-releases.html

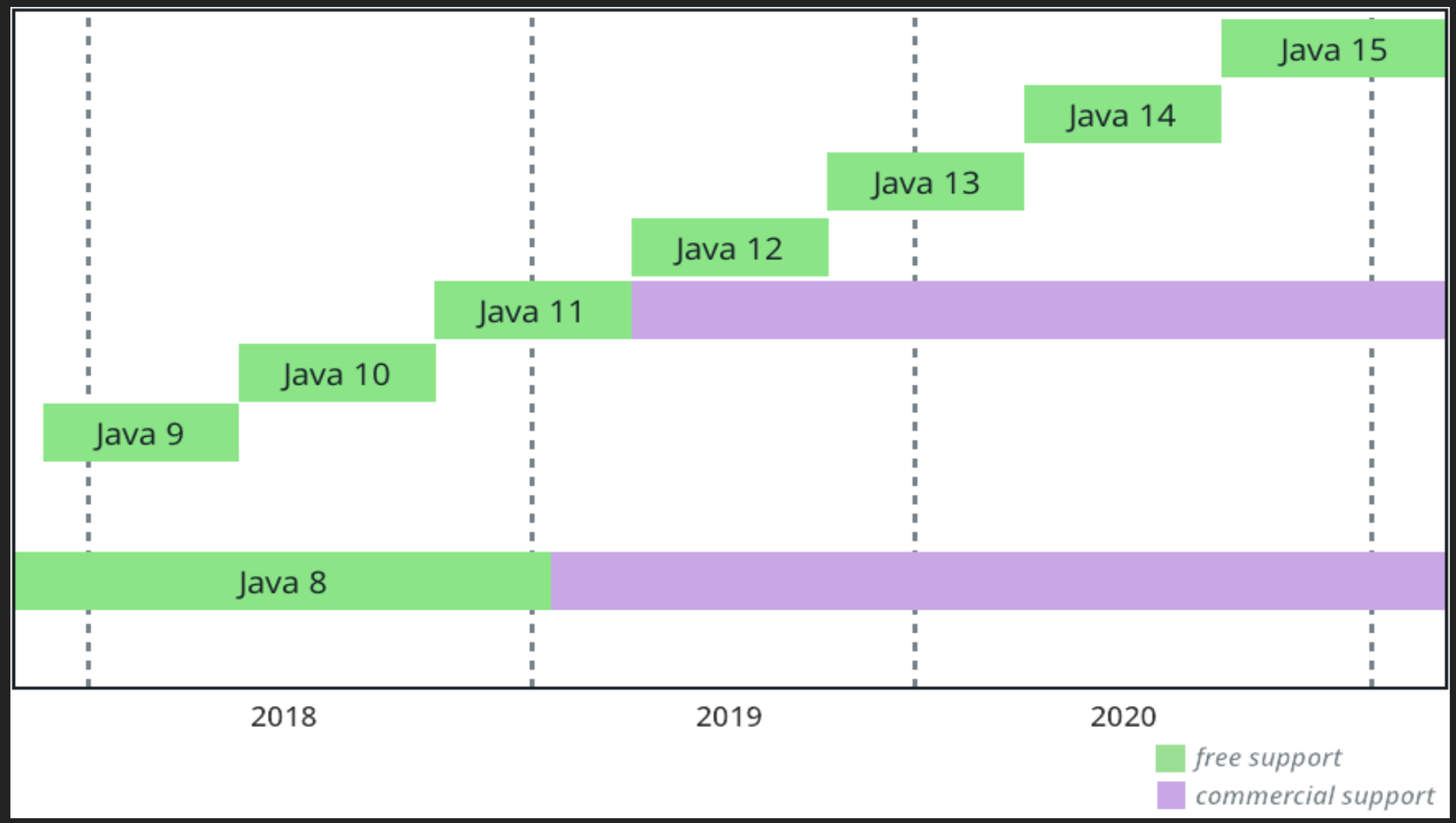
JAVA RELEASE TRAIN NOW

- Java 9 introduced Project Jigsaw - a more modular and "agile" approach to the JDK, allowing smaller, faster updates to be rolled out:



JAVA RELEASE TRAIN NOW

- Java 9 introduced Project Jigsaw - a more modular and "agile" approach to the JDK, allowing smaller, faster updates to be rolled out:

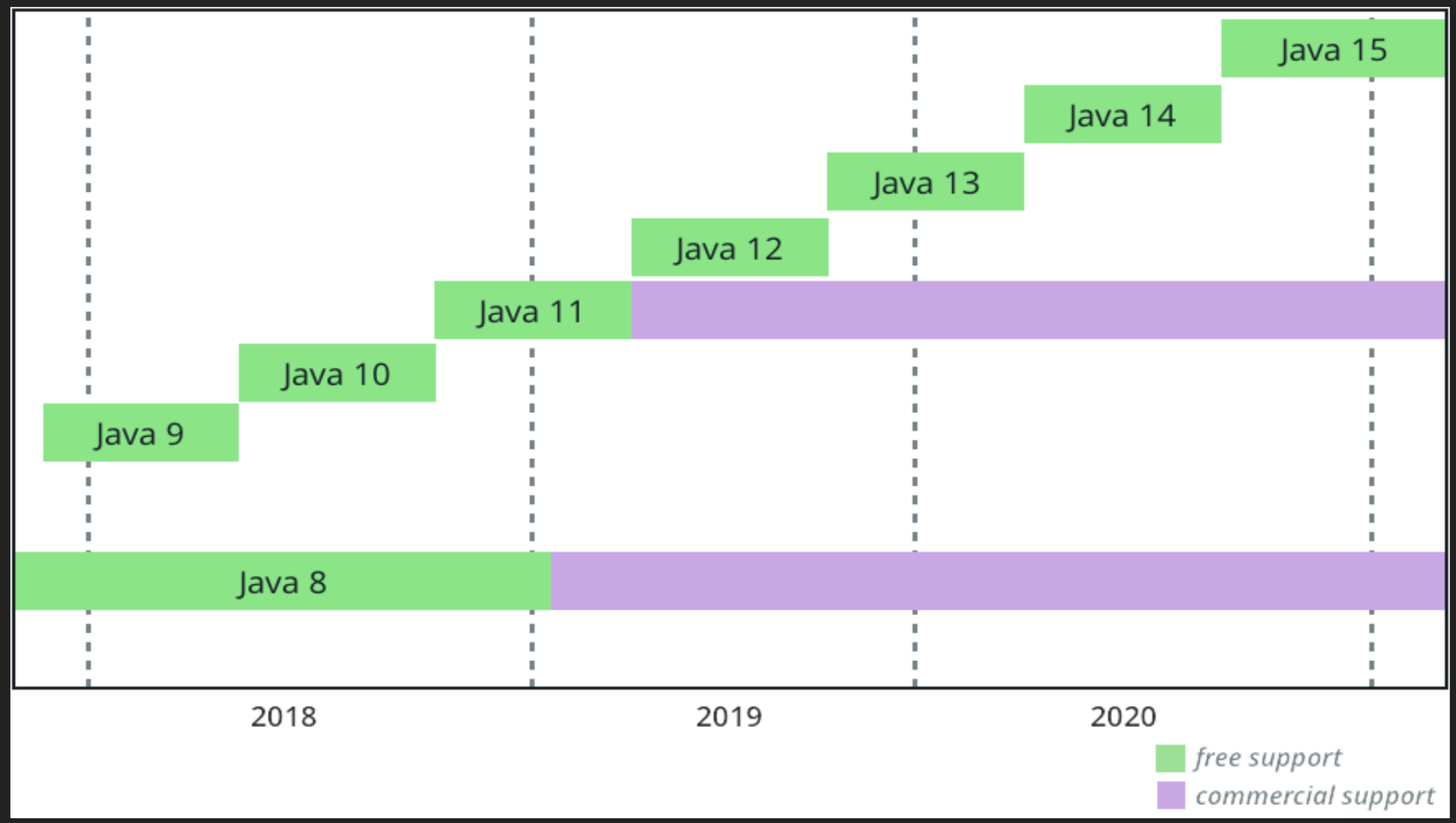


Lambdas, New Stream, Time and Concurrency APIs

JAVA RELEASE TRAIN NOW

- Java 9 introduced Project Jigsaw - a more modular and "agile" approach to the JDK, allowing smaller, faster updates to be rolled out:

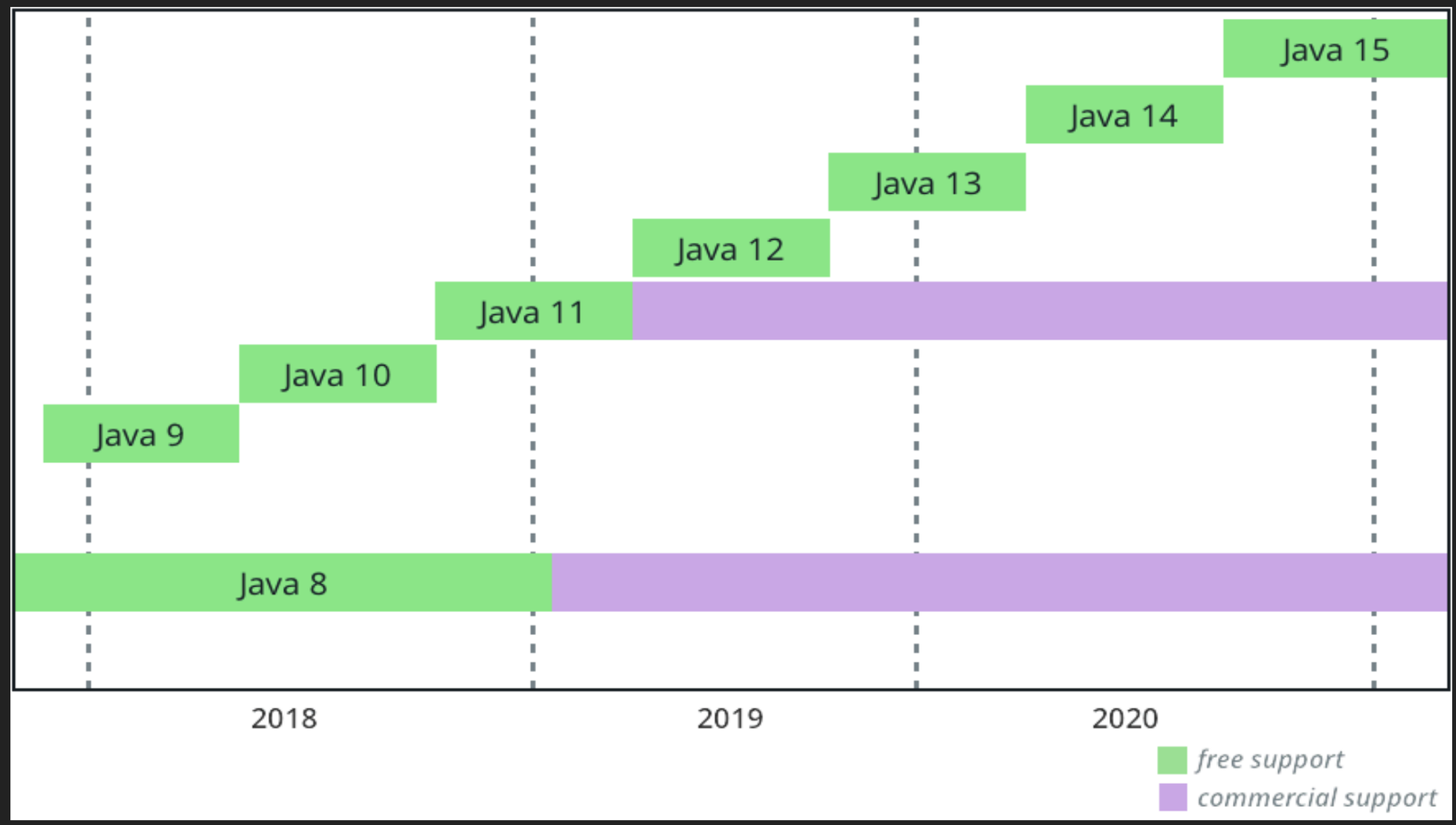
Modular JDK, JShell



JAVA RELEASE TRAIN NOW

- Java 9 introduced Project Jigsaw - a more modular and “agile” approach to the JDK, allowing smaller, faster updates to be rolled out:

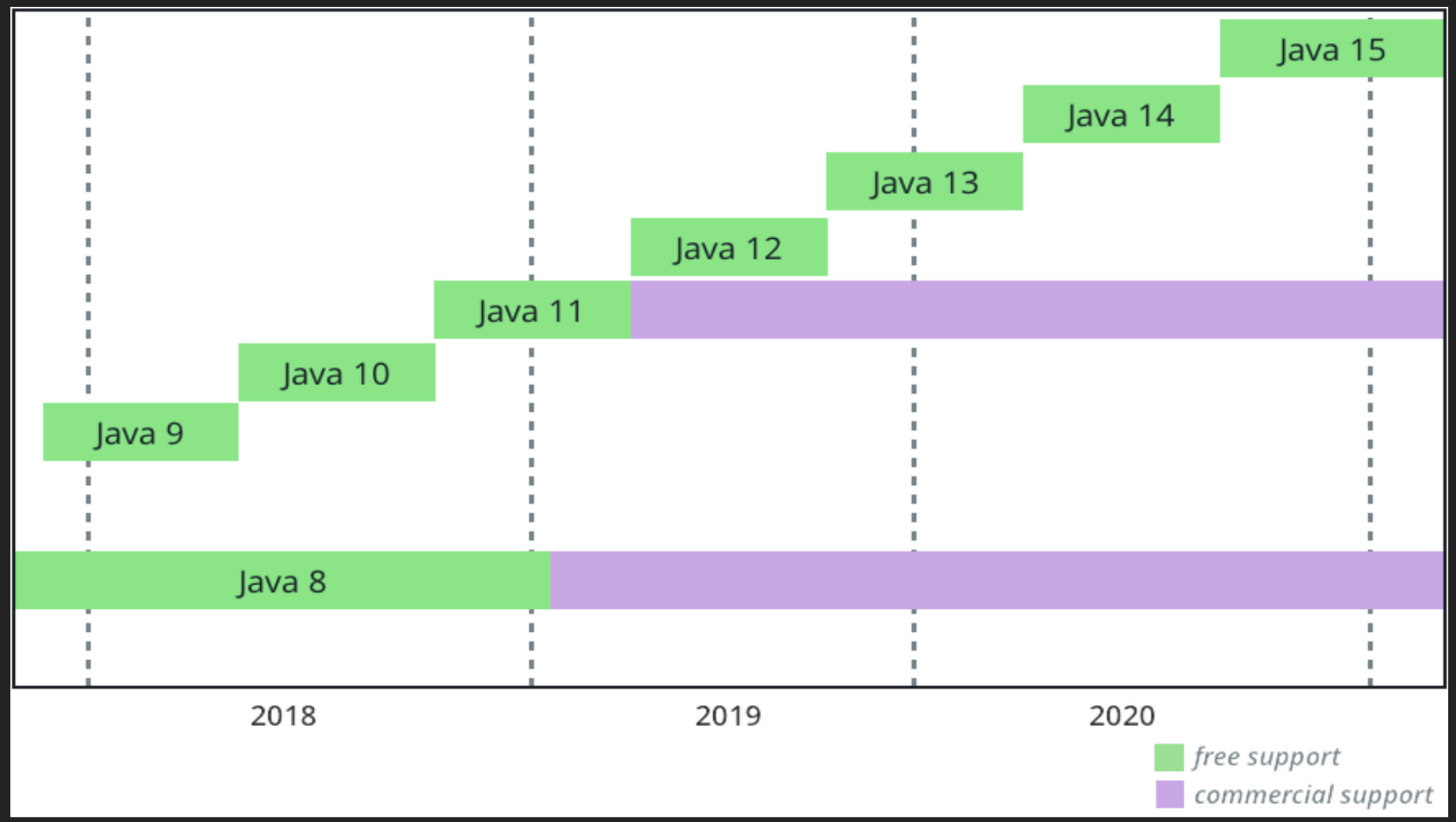
Local variable type inference (var)



JAVA RELEASE TRAIN NOW

- Java 9 introduced Project Jigsaw - a more modular and “agile” approach to the JDK, allowing smaller, faster updates to be rolled out:

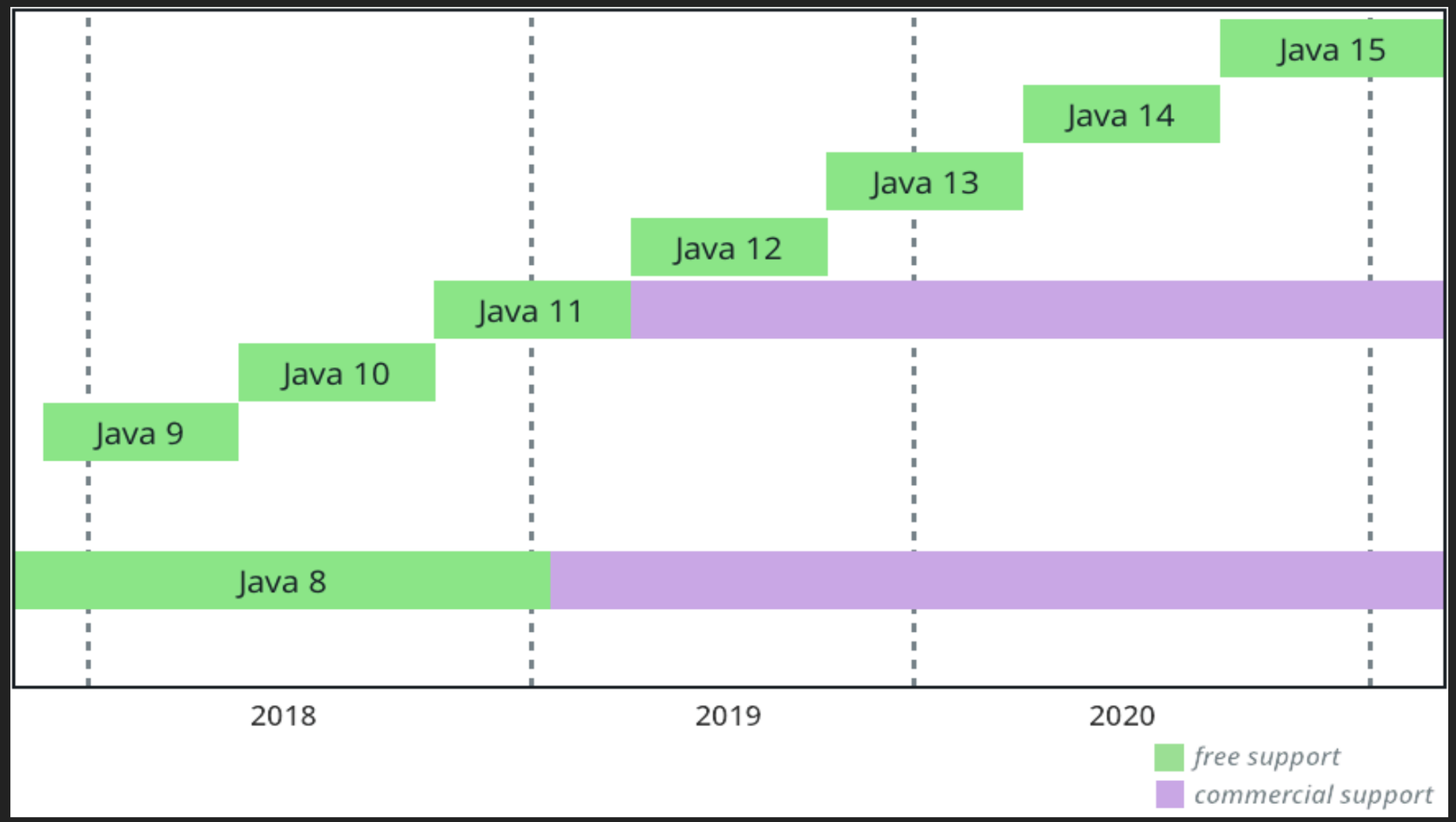
Minor API improvements



JAVA RELEASE TRAIN NOW

- Java 9 introduced Project Jigsaw - a more modular and "agile" approach to the JDK, allowing smaller, faster updates to be rolled out:

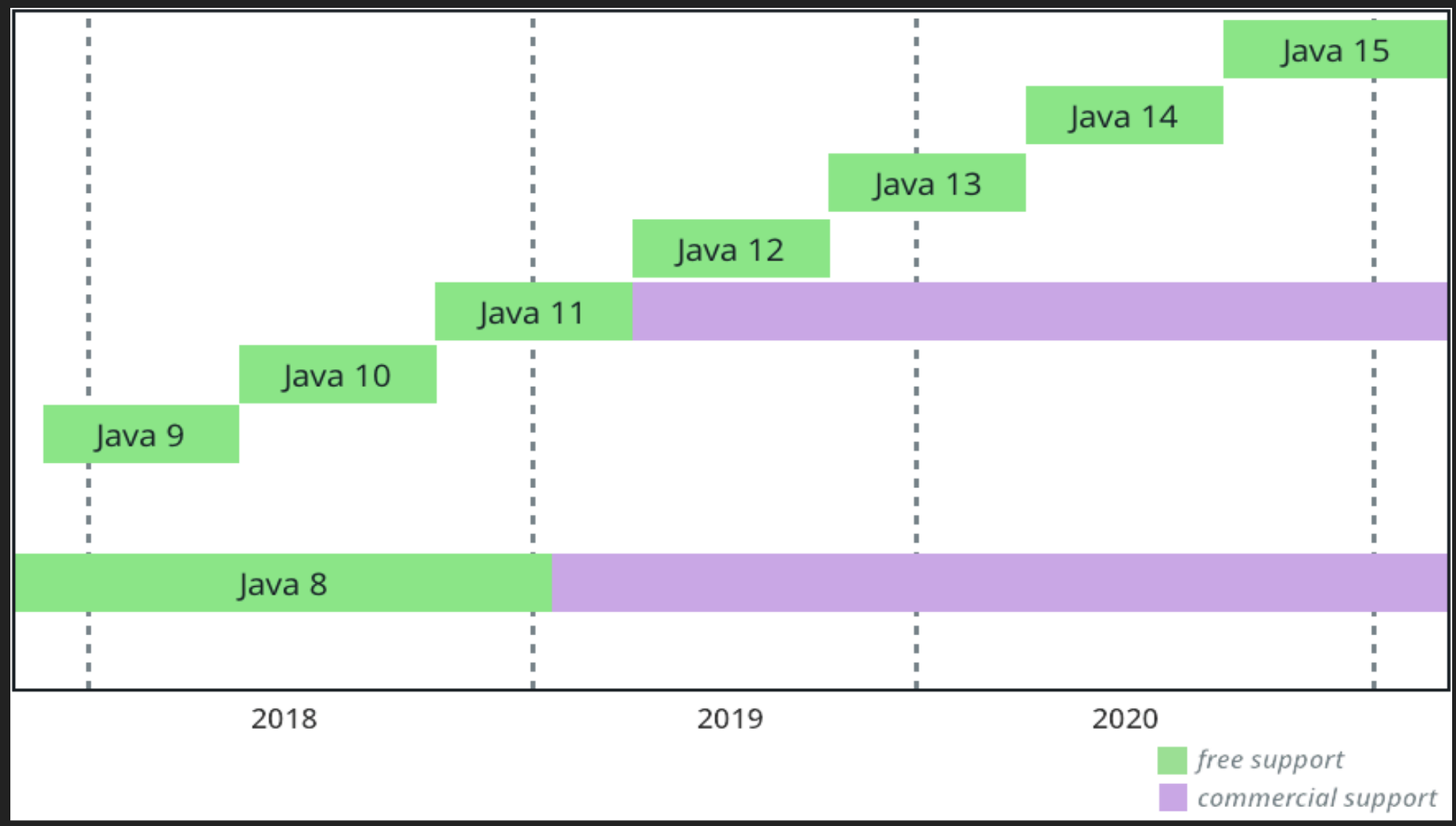
Underlying JDK & JVM improvements



JAVA RELEASE TRAIN NOW

- Java 9 introduced Project Jigsaw - a more modular and "agile" approach to the JDK, allowing smaller, faster updates to be rolled out:

Switch expression improvements, Text blocks



NEW RELEASE TRAIN HAS ITS ADVANTAGES AND DISADVANTAGES

- Upgrading is smoother when you move beyond JDK 9+, but updates are smaller.
- Oracle JDK no longer free to use in production
- Have to use OpenJDK (has feature parity with Oracle JDK)
- https://adoptopenjdk.net/

SO WHAT IS KOTLIN AND HOW CAN IT HELP?

WHAT IS KOTLIN?

- Created by JetBrains in 2011
- JVM language which compiles to Java bytecode (like Scala, Groovy, Clojure)
- Can run on:
  - Android - Officially supported language in 2017
  - Server (JVM, NodeJS)
  - Browser (Transpile to JS)
  - iOS (Through kotlin native)
- Full intro talk can be found here: https://github.com/escullion/kotlin-101-presentation

WHY USE KOTLIN?

- **Concise** - drastically reduce the amount of boilerplate code
- **Safe** - avoid entire classes of errors (e.g null pointer exception)
- **Interoperable** - leverage existing libraries for JVM, Android and the browser
- **Tool-friendly** - choose any existing Java IDE or build directly from command line

SAME AS JAVA

- ✓ Statically typed
- ✓ Use the full JVM ecosystem
- ✓ Development tools (IDEs, build tools etc.)

DIFFERENT FROM JAVA

- Null-safe
- Concise and simple (much less boilerplate)
- Proper functions
- No primitive types
- No checked exceptions
- Syntax

WHAT MAKES IT A GOOD FIT FOR SPRING BOOT?

WHY USE KOTLIN FOR SPRING BOOT?

- Going to cover only some of the many features which make backend developer lives much easier and productive.
- Disclaimer: Some of these problems can be solved with Java libraries (Apache commons, Project Lombok)
- In general though - Kotlin has more flexibility (and less ceremony required) when it comes to underlying APIs

CLASS DEFINITIONS

CLASS DEFINITIONS

- So we have a Java class called person:

```
public class Person {
```

- We need to set up variables, constructor, getters, setters, equals, toString, hashCode, lets see how that looks..

CLASS DEFINITIONS

- Constructor:

```
public Person(String name, int age) {
    this.name = name;
    this.age = age;
}
```

CLASS DEFINITIONS

- Getters and setters:

```java
public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getAge() {
    return age;
}

public void setAge(int age) {
    this.age = age;
}
```

CLASS DEFINITIONS

- Equals and hashCode:

```java
@Override
public boolean equals(Object o) {
    if (this == o) return true;
    if (o == null || getClass() != o.getClass()) return false;

    Person person = (Person) o;

    if (name != null ? !name.equals(person.name) : person.name != null) return false;
    if (age != 0 ? age != person.age : person.age != 0) return false;
}

@Override
public int hashCode() {
    int result = name != null ? name.hashCode() : 0;
    result = 31 * result + age;
    return result;
}
```

CLASS DEFINITIONS

- toString:

```
@Override
public String toString() {
    return "Person{" +
        "name='" + name + '\'' +
        ", age='" + age + '\'' +
        '}';
}
```

CLASS DEFINITIONS

- We have to do this setup a LOT in a regular project.
- Project Lombok helps with this, but wouldn't it be good to have something built into the language to handle this?

CLASS DEFINITIONS IN KOTLIN

- Class definitions in kotlin:

```
data class Person(val name: String, val age: Int)
```

- Generates constructor, getters, setters, equals, hashcode, toString and copy

CLASS DEFINITIONS IN KOTLIN – BONUS POINTS

- Easy immutable objects using “val” and “copy”:

```
val joe = Person(name = "joe", age = 30)
val olderJoe = joe.copy(age = 31)
```

- Very important for ensuring objects can only have a valid state

CLASS DEFINITIONS IN KOTLIN – BONUS POINTS

- Destructuring declarations:

```
val jane = Person(name = "Jane", age = 20)
val (name, age) = jane

println("$name, $age years of age") // prints "Jane, 35 years of age"
```

- Improved readability and productivity for any work that involves retrieving specific fields from an object, e.g performing validation, logging out data

NULL SAFETY

NULL SAFETY

- Java requires “safe” programming - making sure to null check things before you can use them to avoid null pointer exceptions:

```
if (person != null) {
    // use person for something
    person.getAge();
}
```

- Java 8 Optionals do help with this, but are only recommended to be used as method return types to indicate “no result”

NULL SAFETY

- Kotlin has a number of ways to handle null pointers.

```
var a: String = "abc"
a = null // compilation error
```

- The compiler won’t actually let you set the value of something to null by default!

NULL SAFETY

- We can declare a variable as nullable using “?”

```
var b: String? = "abc"
b = null // ok
print(b)
```

- This will force you to handle the variable safely when it is used

NULL SAFETY

- Safe calls:

```
val l = b.length // error: variable 'b' can be null
```

- To access this safely, we once again use the “?” operator:

```
val a = "Kotlin"
val b: String? = null
println(b?.length)
println(a?.length)
```

- This returns b.length if b is not null, otherwise it will return null (without throwing a null pointer exception)

ELVIS OPERATOR

- Allows you to specify what to return if the result is null:

```
val l = b?.length ?: -1
```

- This returns length if b is not null, otherwise it will return -1

```
val name = node.getName() ?: throw IllegalArgumentException("name expected")
```

- Useful for performing validation

WHY'S IT CALLED THE ELVIS OPERATOR?


Why ?: is called the
Elvis operator

SO WHAT IF YOU STILL WANT TO USE NULL?

- If you **really** want the risk of null pointer exceptions, you can use the non-null assertion operator (!!):

```
val l = b!!.length
```

- Like in Java, this will throw a null pointer exception if b is null

EXTENSION FUNCTIONS

EXTENSION FUNCTIONS

- These allow you to extend functionality on top of an existing API, without affecting the underlying code:

```
private fun LegacyService.newFeature() {
    // perform functionality
}
```

- Good when working with legacy code that would be risky to change, or adding functionality to an external API that you don’t have control of

FP SUPPORT

FUNCTIONAL PROGRAMMING

- Great support for immutability (has built in read-only collections and mutable collections)
- Higher-order functions and lambdas - functions can be used as parameters and as return type
- No need for Java 8 Streams - every collection has built in support for functional operations

```
val numbers = listOf("one", "two", "three", "four")
val longerThan3 = numbers.filter { it.length > 3 }
```

DOMAIN SPECIFIC LANGUAGES (DSL)

KOTLIN DSL

- Allow you to write type-safe, statically-typed builders in Kotlin.
- Allows you make use of language features to interface with other languages (most common usage is to make underlying APIs easier to work with).
- Later, we will be looking at a few more examples later (Gradle Kotlin DSL, Functional Router)

COROUTINES

COROUTINES

- Lightweight threads that allow you to write non-blocking asynchronous code, in an imperative way!
- Coroutines can run in parallel, wait for each other and communicate.
- Very cheap - we can create thousands with little performance impact.
- Official Coroutines support in Spring Framework 5.2 back in April

COROUTINES EXAMPLE

- Here we have an endpoint:

```
@GetMapping("/{id}")
suspend fun findOne(@PathVariable id: String): UserWithDetails {
    val user: User = userRepository.findOne(id) ?:
    throw CustomException("This user does not exist")
    return withDetails(user)
}
```

- “suspend” modifier indicates that a function can be suspended, while awaiting for a result or action, freeing up coroutines for other processes.
- In this example, “withDetails” is also a suspending function

COROUTINES EXAMPLE

- Here we are making a few external API requests and awaiting the result:

```
private suspend fun withDetails(user: User): UserWithDetails {
    val userDetail1 = client.get().uri("/userdetail1/${user.login}")
            .accept(APPLICATION_JSON)
            .awaitExchange().awaitBody<UserDetail1>()
    val userDetail2 = client.get().uri("/userdetail2/${user.login}")
            .accept(APPLICATION_JSON)
            .awaitExchange().awaitBody<UserDetail2>()
    return UserWithDetails(user, userDetail1, userDetail2)
}
```

- awaitExchange() and awaitBody() suspend the function until the body is returned.
- Coroutines are sequential by default, so we could refactor this further to make both requests happen concurrently

KOTLIN GRADLE

WHY USE GRADLE?

- Gradle is a flexible build tool which allows you to build, run and package your app.
- Gradle is usually written using Groovy (dynamic language), and can be hard for beginners to use (easy to read, harder to write).
- Usually ends up as trial-and-error until it magically works!

WHY USE KOTLIN FOR GRADLE?

- Since Kotlin Gradle DSL is type safe, this gives the following benefits:
  - Instant feedback when you make a mistake
  - Full IDE support (Auto completion, warning, errors)
- There's also a benefit to writing your build scripts in the same language as your app, easier for developers to learn.
- Migration guide: https://guides.gradle.org/migrating-build-logic-from-groovy-to-kotlin/

ALRIGHT.. LETS START LOOKING AT SPRING BOOT

CONTROLLER VS FUNCTIONAL ROUTING

ANNOTATED CONTROLLER VS FUNCTIONAL ROUTING

- Controller is the more traditional support, but node developers might be more familiar with functional routing.
- Controller has a straight forward setup:

```
@RestController
class ReportGeneratorController(private val reportGeneratorService: ReportGeneratorService) {

    @GetMapping( ...value: "/")
    fun generateReport(): String {
        return reportGeneratorService.generateReport()
    }
}
```

- If there is one constructor, it will automatically autowire the required objects (in this case reportGeneratorService)

FUNCTIONAL ROUTING

- Here we are using a Kotlin DSL called Kofu (an extension of Spring Fu).
- We will go through each section
- Fast start up time:

```
Started DemoApplicationKt in 0.999 seconds
```

```kotlin
val app = application(WebApplicationType.SERVLET) {
    logging {
        level = LogLevel.DEBUG
    }
    beans {
        bean<SampleService>()
    }
    webMvc {
        port = if (profiles.contains("test")) 8181 else 8080
        router {
            val service = ref<SampleService>()
            GET("/") {
                ok().body(service.generateMessage())
            }
            GET("/api") {
                ok().body(Sample(service.generateMessage()))
            }
        }
        converters {
            string()
            jackson()
        }
    }
}

data class Sample(val message: String)

class SampleService {
    fun generateMessage() = "Hello world!"
}

fun main(args: Array<String>) {
    app.run(args)
}
```

FUNCTIONAL ROUTING

- First we declare the type of web app that we want (Servlet or Reactive):

```
val app = application(WebApplicationType.SERVLET)
```

- Then we can set up properties about our app (other options are available, like security)

```
logging { this: LoggingDsl
    level = LogLevel.DEBUG
}
beans { this: BeanDefinitionDsl
    bean<SampleService>()
}
```

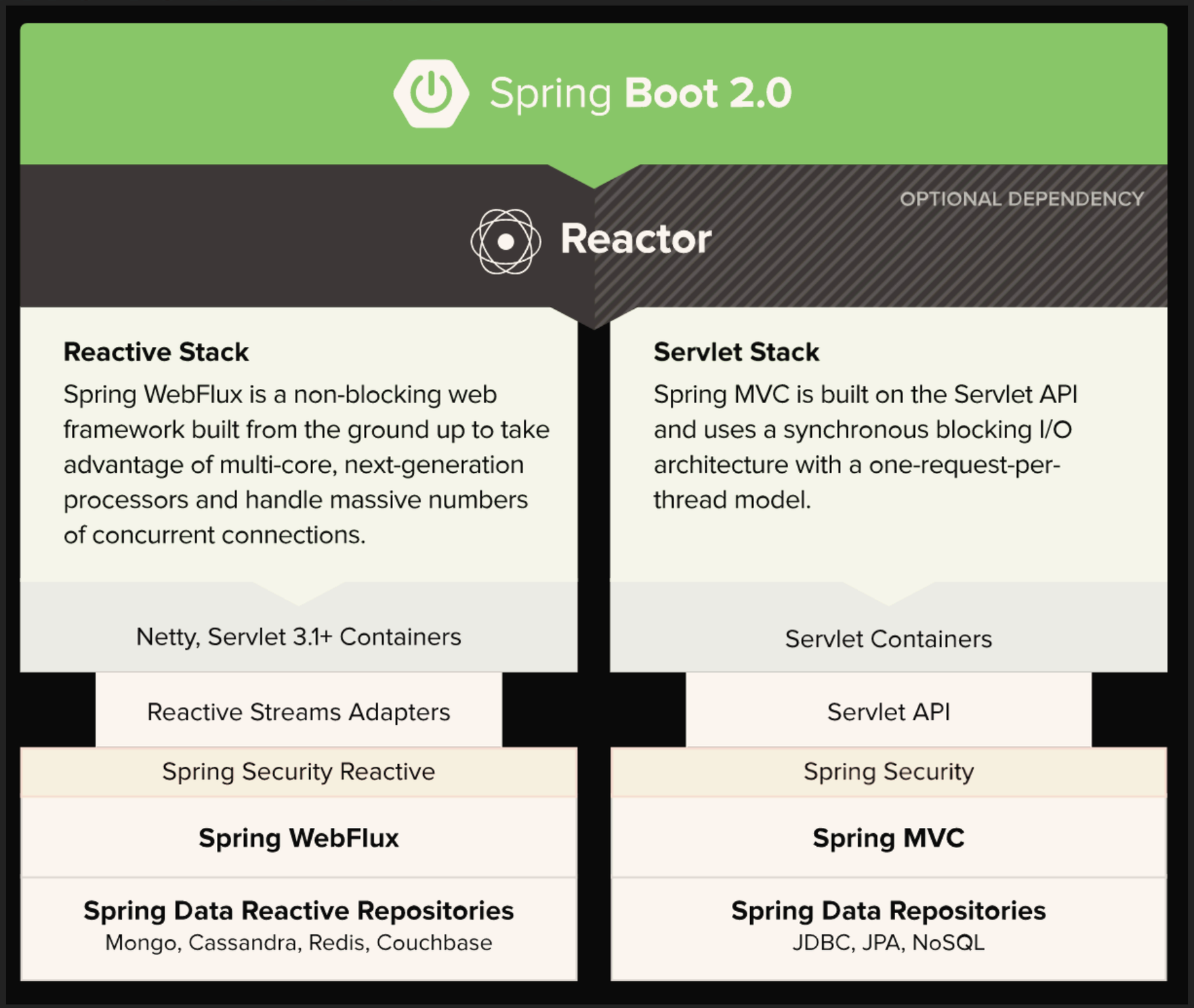
FUNCTIONAL ROUTING

- Finally, we set up our endpoints and what format they can return (if we were using Reactive, we would use webFlux instead of webMvc):

```
webMvc { this: WebMvcServerDsl
port = if (profiles.contains("test")) 8181 else 8080
router { this: RouterFunctionDsl
    val service = ref<SampleService>()
    GET( pattern: "/") { it: ServerRequest
        ok().body(service.generateMessage())
    }
    GET( pattern: "/api") { it: ServerRequest
        ok().body(Sample(service.generateMessage()))
    }
}
converters { this: WebMvcServerDsl.WebMvcConverterDsl
    string()
    jackson()
}
```

SPRING MVC VS WEBFLUX

SPRING MVC VS WEBFLUX



SPRING MVC

- Spring MVC runs on Servlet stack which is **blocking**.
- It runs on a request-response model, threads are created by each new request (and will be blocked while they are waiting).
- Scaling is generally achieved by increasing the size of the thread pool.
- Can use a lot of reactive libraries to help (WebClient, RxJava etc), but still limited by the underlying Servlet API.

SPRING MVC

- And the reason its used so much is that it’s genuinely easier to work with and imperative style is easier to read:

```
@RestController
class CustomerController (val repository:CustomerRepository) {

    @GetMapping("/")
    fun findAll() = repository.findAll()

    @GetMapping("/{name}")
    fun findByLastName(@PathVariable name:String)
            = repository.findByLastName(name)
}
```

SPRING WEB FLUX

- Spring WebFlux runs on Reactive stack, which is based on Reactive Streams and allow for more scalability, lack of latency and better stream processing capabilities (Good for using with Kafka).
- Can use the default Java Reactor API which is a more declarative/functional approach, or Kotlin Flow with Coroutines which allows us to write imperative-like code.
- Learning curve can be high, but it’s the kind of change you need to allow you to handle millions of requests, rather than hundreds/thousands

SPRING WEBFLUX WITH JAVA REACTOR API

- Reactor has return types like Mono, Flux to allow us to do non-blocking stream-based work.
- Flux returns either 0 to many elements, Mono returns an element or error:

```
@GetMapping("/{id}")
public Mono<User> findOne(@PathVariable String id) {
    return userRepository
            .findOne(id)
            .switchIfEmpty(Mono.error(
                    new CustomException("This user does not exist");
}
```

SPRING WEBFLUX WITH KOTLIN FLOW EXAMPLE

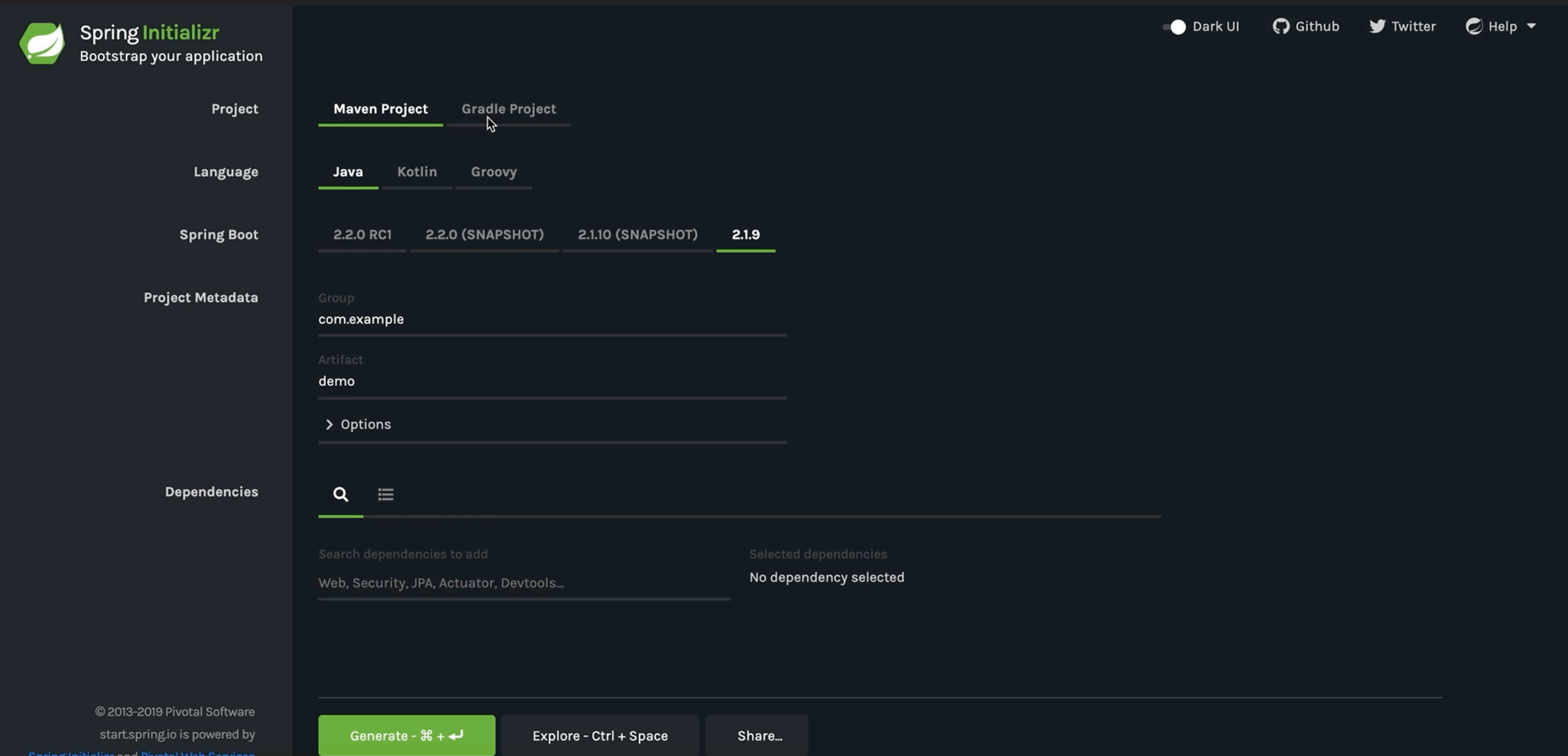
- With Kotlin Flow and Coroutines, we can use WebFlux but still get the simple, easy to read benefits of imperative programming (Flow is the equivalent of Flux):

```
@RestController
class UserController(private val userRepository: UserRepository) {

    @GetMapping("/")
    fun findAll(): Flow<User> =
            userRepository.findAll()

    @GetMapping("/{id}")
    suspend fun findOne(@PathVariable id: String): User? =
            userRepository.findOne(id) ?:
            throw CustomException("This user does not exist")

    @PostMapping("/")
    suspend fun save(user: User) =
            userRepository.save(user)
}
```

SO HOW DO I GET STARTED?

HOW TO GET STARTED

- https://start.spring.io



LINKS

- Kotlin official docs:
  - https://kotlinlang.org/docs/reference/
- Kotlin Flow, WebFlux and Coroutines:
  - https://spring.io/blog/2019/04/12/going-reactive-with-spring-coroutines-and-kotlin-flow
- Spring Official Kotlin docs:
  - https://docs.spring.io/spring/docs/5.2.0.M1/spring-framework-reference/languages.html#kotlin

GENERAL ADVICE

PUT OFF BY LEARNING YET ANOTHER LANGUAGE?

- Kotlin has a very low learning curve for Java developers
- Can start off with writing Kotlin in Java style, then slowly incorporate some of the cool features!
- Easily transferrable skills - Android developers can easily work on server-side, allowing for cross-functional teams

OTHER GENERAL ADVICE (AND WARNINGS)

- Kotlin isn't a silver bullet - some things are easier to do in Java (for cases where there isn't a kotlin library/DSL)
- Kotlin is fully interoperable with Java, but a lot of care has to be taken to ensure null safety - everything in Java is nullable which could cause question mark hell if you aren't careful
- Micro-services should allow you to use the best tools for the job, so it's perfectly fine to use languages based on suitability and the teams skillset.

QUESTIONS?