

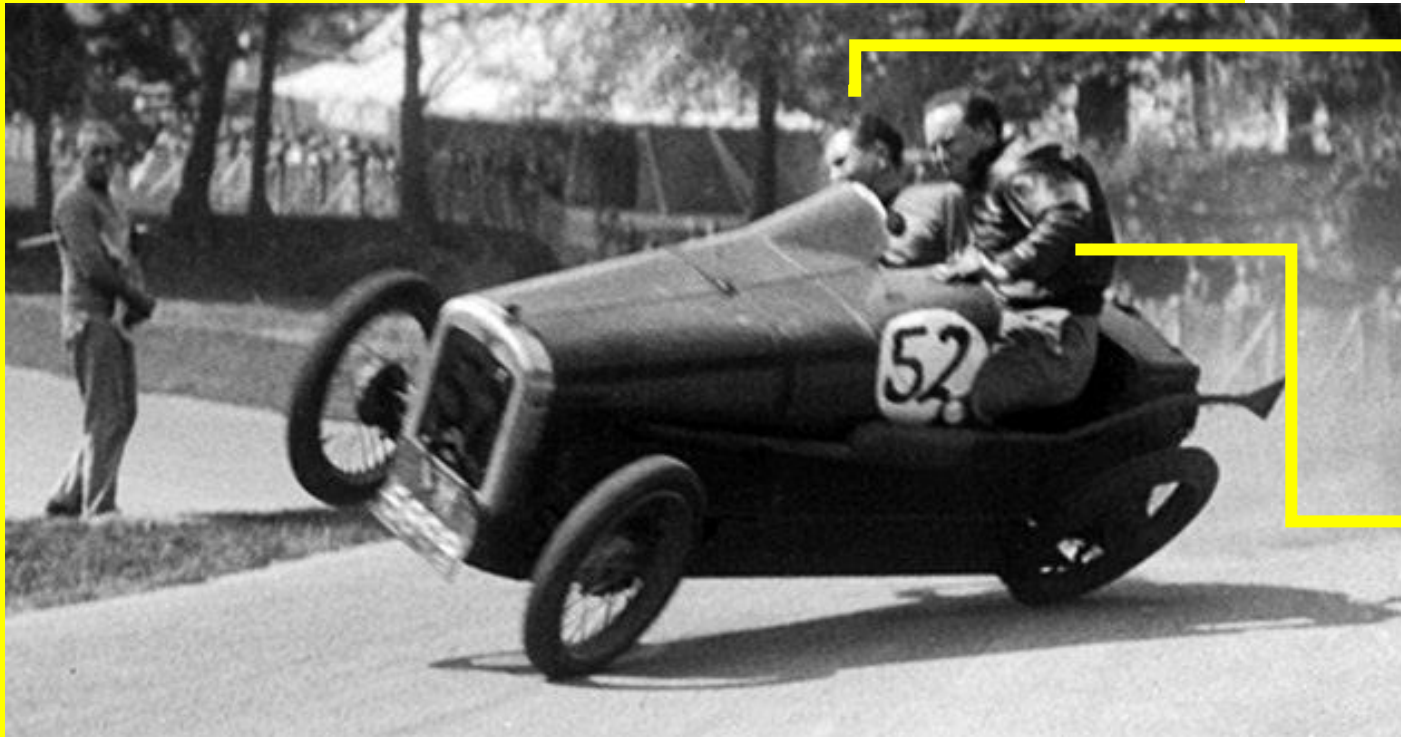






FAST & FURIOUS

Modern Java Development



Drivers



- *Jan Van Wassenhove*
 - Lead Architect Tobania.Development
 - 15 years of working experience
 - Connect on 
 - Instagram   "mITy.John"
- *Sébastien Vanpé*
 - Architect Tobania.Development
 - 20 years of working experience
 - Connect on 

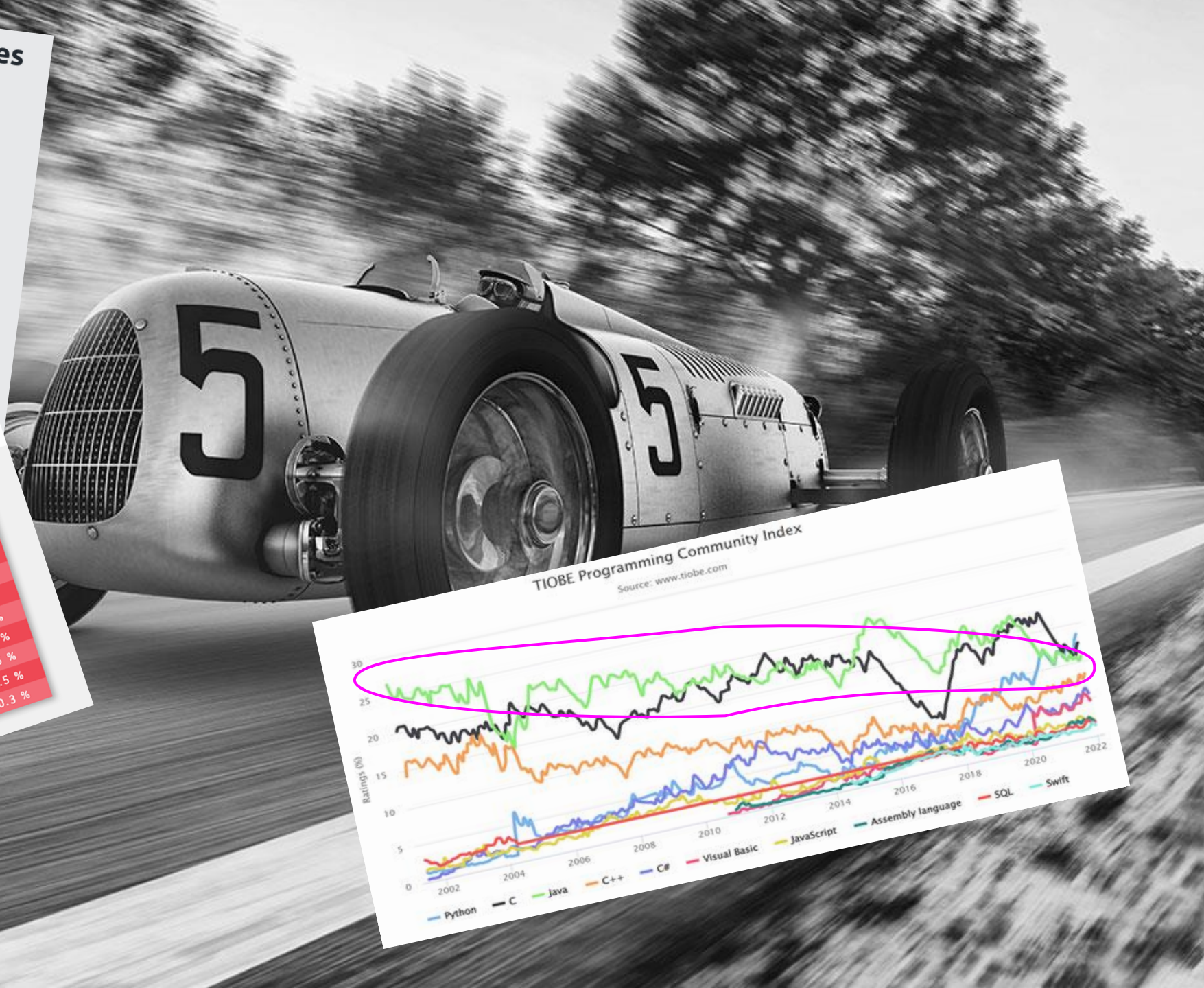


A small throwback in history...

*For a long time, Java was the undisputed **number one** programming language...*

- Perfect for large, monolithic enterprise applications
- Outstanding IDE's
- Very big community and companies support
- Mature, Secure, Stable, and Robust Ecosystem
- JVM based-language (Java, Kotlin, Groovy, and Scala)
- Platform Independent.



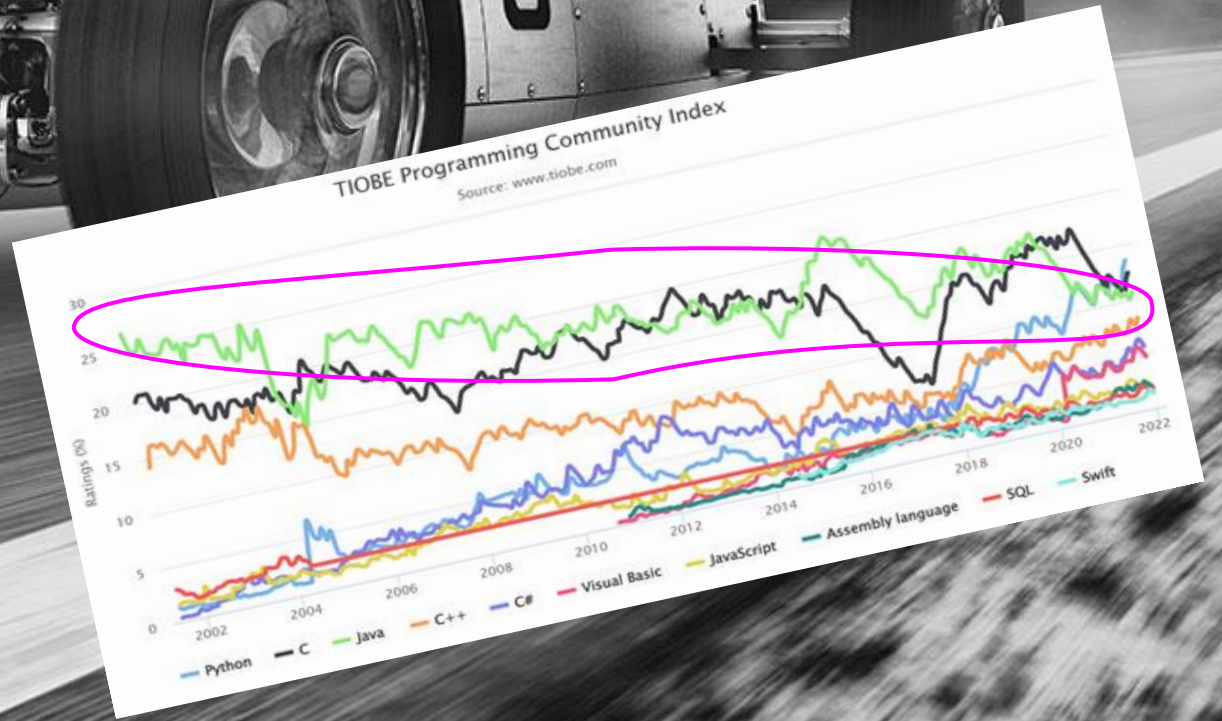


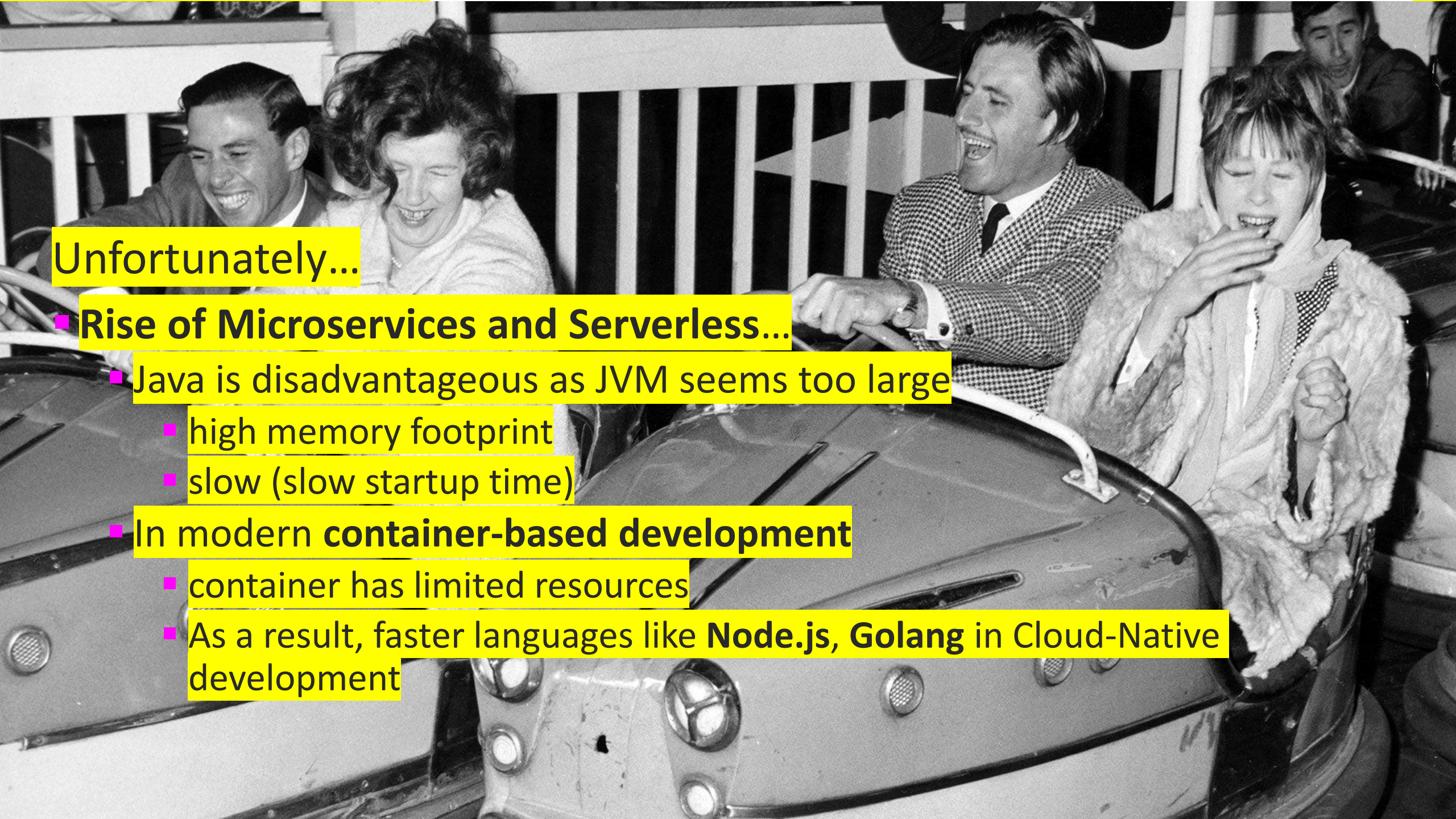
TOP 10 PROGRAMMING LANGUAGES

Worldwide Google Trends, Dec 2016 vs Dec 2015

Rank	Change	Language	Share	Trend
1		JAVA	23.7 %	-0.1 %
2		Python	14.0 %	+2.6 %
3		PHP	9.7 %	-1.0 %
4		C#	8.4 %	-0.4 %
5	▲	Javascript	7.9 %	+0.7 %
6	▼	C++	6.9 %	-0.9 %
7	▼	C	6.8 %	-0.8 %
8	▼	Objective-C	4.6 %	+0.5 %
9	▲	R	3.3 %	+0.3 %
10	▼	Swift	3.1 %	+0.3 %

Source: The PYPL Popularity of Programming Language Index





Unfortunately...

- **Rise of Microservices and Serverless...**
- Java is disadvantageous as JVM seems too large
 - high memory footprint
 - slow (slow startup time)
- In modern **container-based development**
 - container has limited resources
 - As a result, faster languages like **Node.js, Golang** in Cloud-Native development

***Java Developers never RIP,
They just get Garbage Collected***

So finally...

**the Java Community provided a
modern version of Java**







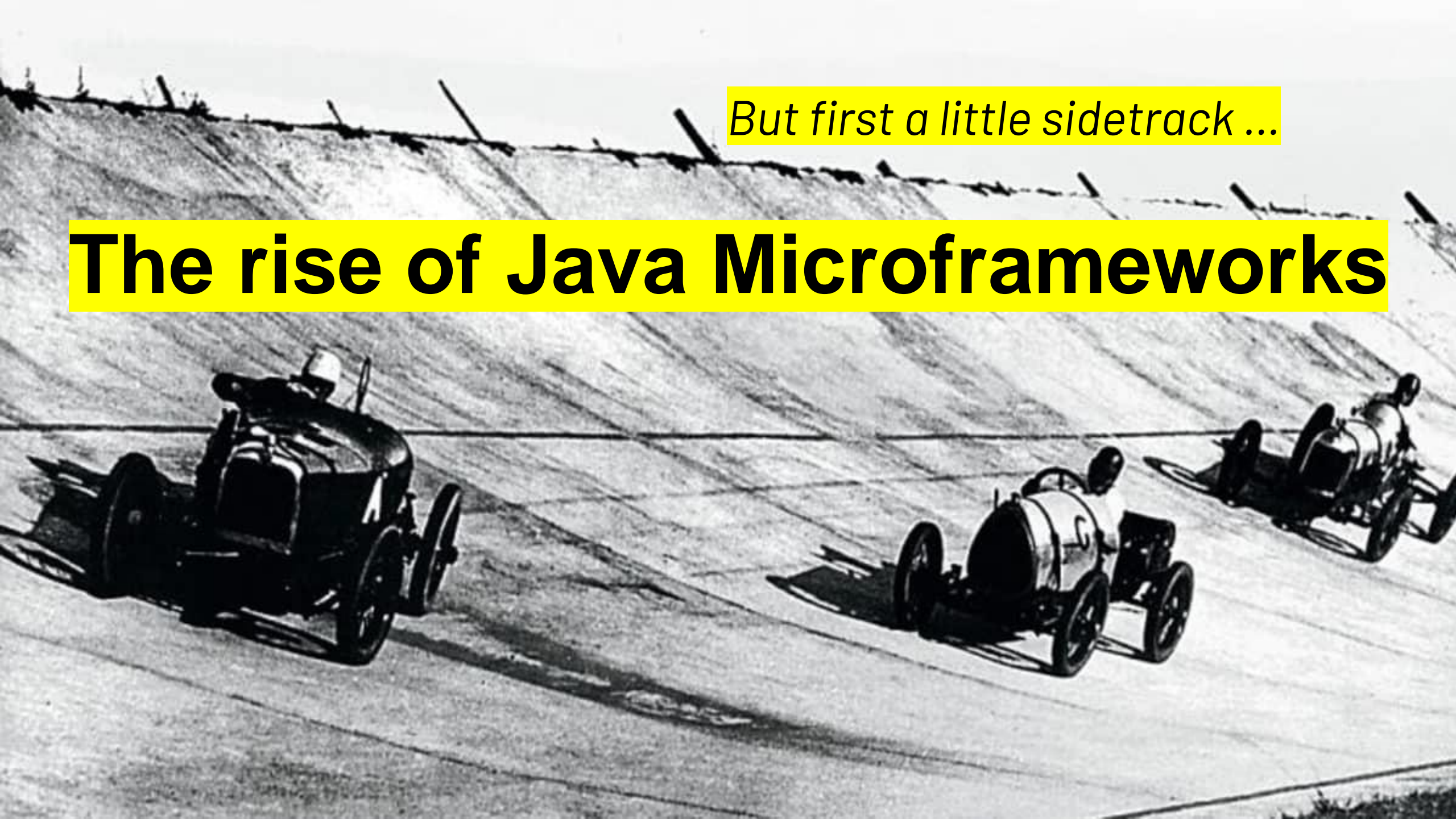
GraalVMTM

A new era of speed arises...



But first a little sidetrack ...

The rise of Java Microframeworks



Moving to microservices



**We can scale...
but our resources
are finite!**



**So how to use
less resources ?**



Microframeworks

- Refers to minimalistic web application frameworks
 - *Without authentication and authorization*
 - *Without database abstraction via object-relational mapping*
 - *Without input validation an input sanitation*
- Examples
 - *Javalin*
 - *Micronaut*
 - *Helidon*
 - *Quarkus*
 - ...



**But less modules, functions
and dependencies are not
enough...**



How do e.g. Spring or Jakarta EE work?

Spring e.g. is an amazing technical achievement and does so many things, but does them at **Runtime**.

- Reads the byte code of every bean it finds.
- Synthesizes new annotations for each annotation on each bean method, constructor, field etc. to support Annotation metadata.
- Builds Reflective Metadata for each bean for every method, constructor, field etc.





So when using reflection...



**... is it possible to have the
same productivity but without
reflection?**



Yes, of course!

With AOT Compilation!



Ahead of Time (AOT) Compilation

“Compiling high level programming language or intermediate representation such as java byte code into native machine code so that the resulting binary file can execute natively.”

This will result in

- Short startup time
- Dependency injection at compile time
- Can be run with as little as 15mb Max Heap



GraalVMM™



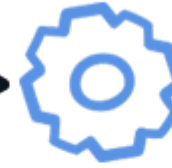
GaalVM

- It's a tool for developers to write and execute Java code
 - It is a Java Virtual Machine (JVM)
 - and Java Development Kit (JDK) created by *Oracle*
- The JDK distribution offering contains
 - **AOT (ahead-of-time)** compilation
 - **Polyglot** programming
 - Compiles to directly Native Code
 - Perfectly suitable for **Cloud-Native development (low memory footprint, first startup time)**
- For existing Java applications, GraalVM can provide benefits by running them faster, providing a faster **Just In Time (JIT)** Compilation

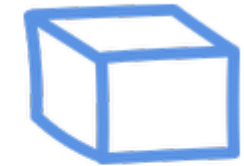


GraalVM objectives

(1) TAKE YOUR APPLICATION



(3) RUN IT!



(2) PRODUCE A NATIVE EXECUTABLE



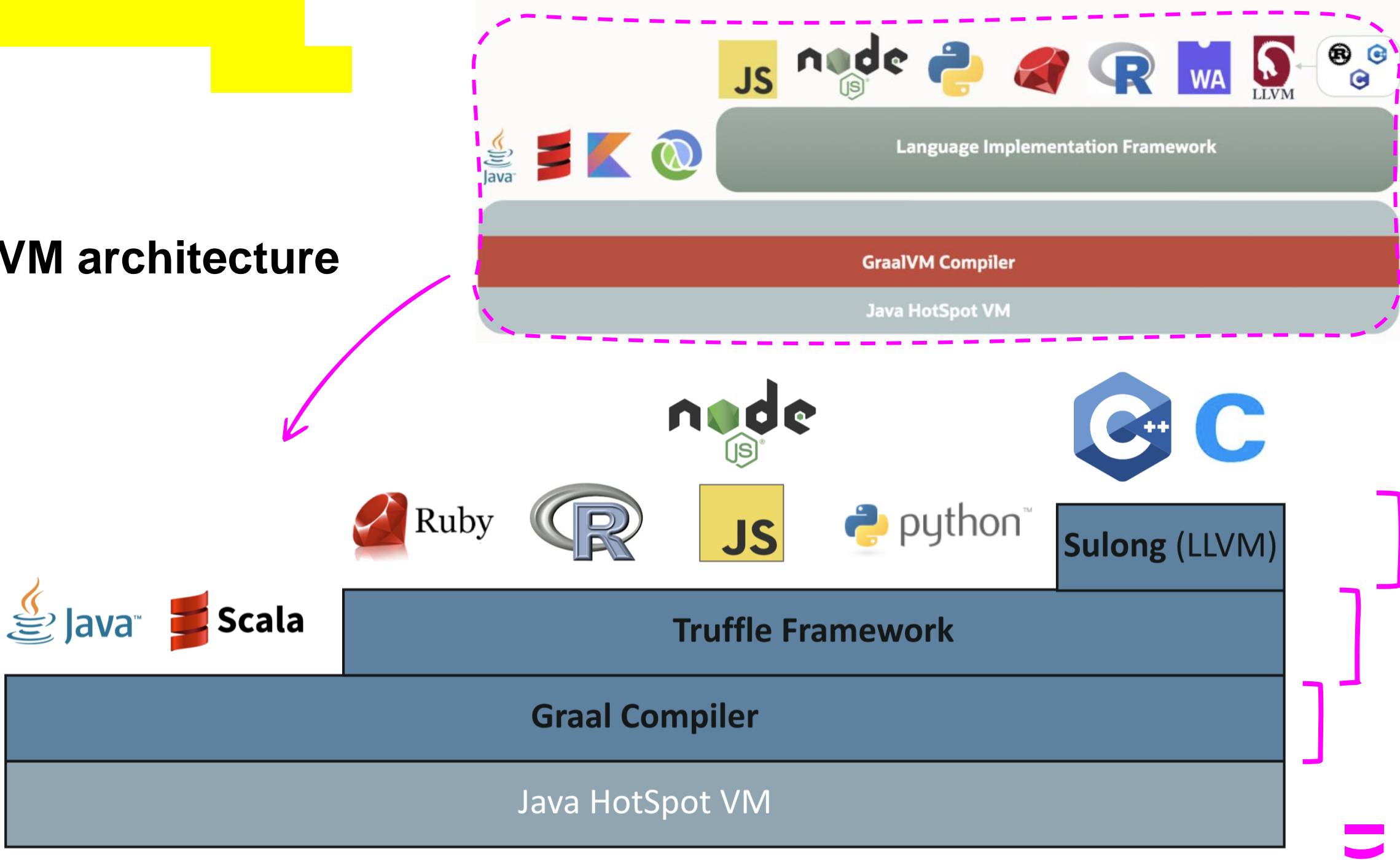
(4) USE IT!

The main objectives of GraalVM are:

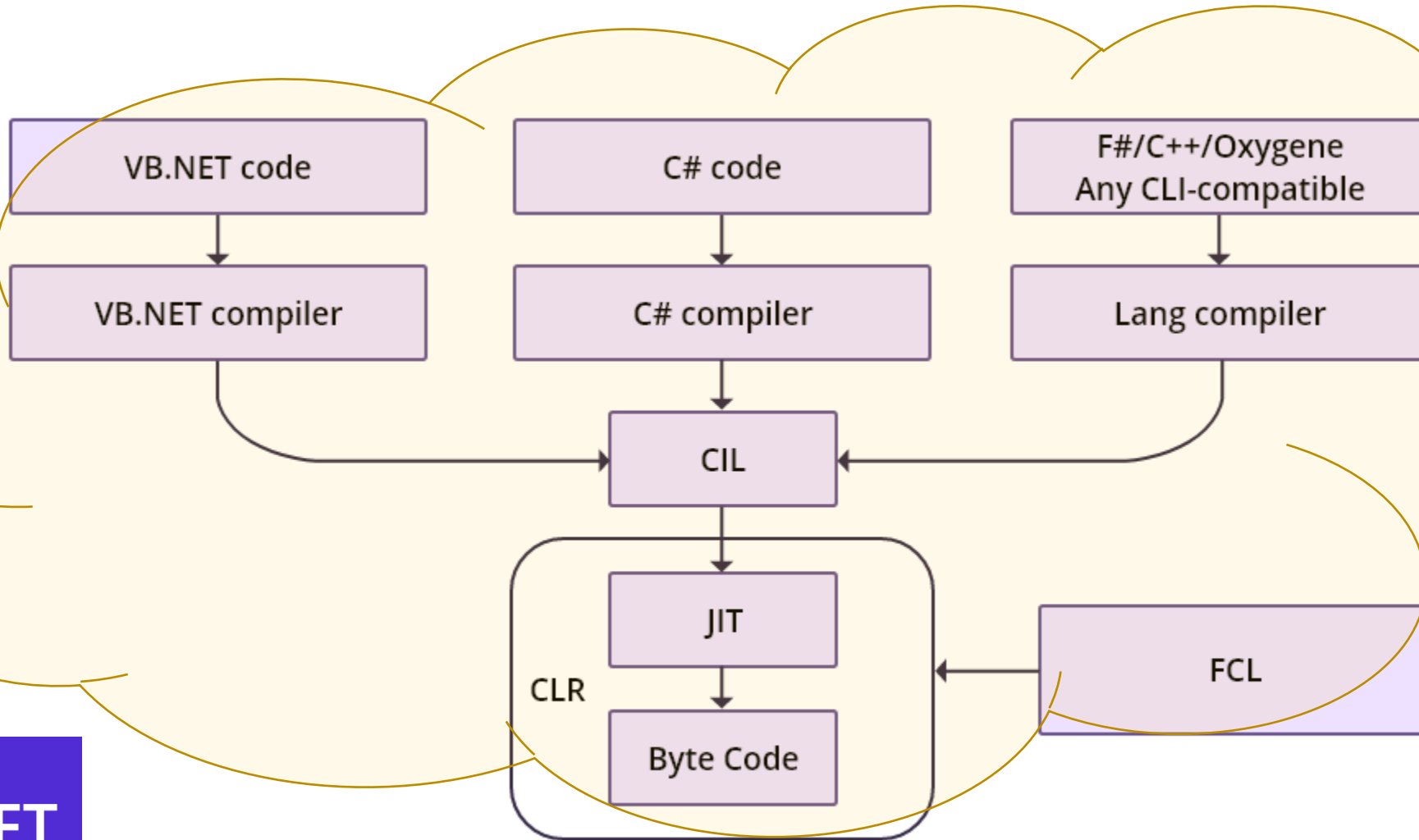
- To write a **compiler** that is faster and easier to maintain
- **Low-footprint, fast-startup** Java for Cloud and Serverless
- Improving the **performance** of languages that run on the JVM
... and so reducing application startup times
- Integrating **multi-language support** into the Java ecosystem, as well as providing a set of programming tools to do so



GraalVM architecture



Does it ring a bell?



Let's compile!

A word on Graal AOT & JIT compilation



Graal JIT Compilation

Just In Time (JIT) compilation

- a way of executing computer code that involves compilation during execution of a program
- It runs complex optimizations to generate high-quality machine code

1. Compiling source code into binary presentation (JVM bytecode).

Source code
System.out.println("Hello World");



2. To be able to run a Java program, the JVM interprets the bytecode.

Byte code
6a 61 76 61 20
c3 a9 20 66 6f
64 61



*3. The JVM runs another compiler which will now compile our bytecode into the machine code that can be run by the processor: **JIT***

Machine code
01101010
01100001
0100000

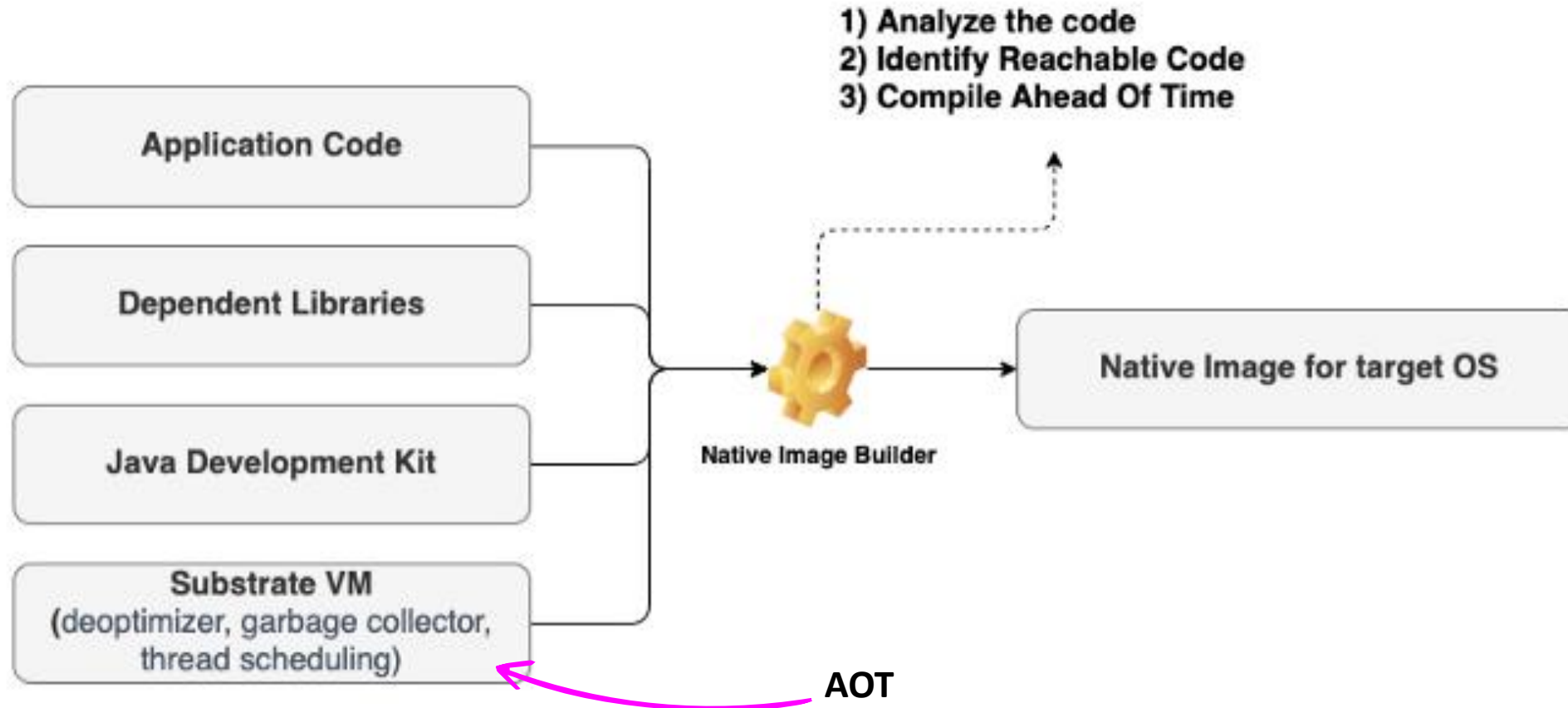


Optimizations to generate high-quality machine code



Graal AOT Compilation

AOT Compilation “*compiling a higher-level programming language into a lower-level language before execution of a program, usually at build-time, to reduce the amount of work needed to be performed at run time.*”



GraalVM – Limitations Native Image

- **Dynamic Class Loading**
 - Deploying jars, wars, etc. at runtime impossible.
- **Reflection**
 - Requires registration via native-image CLI/API.
- **Dynamic Proxy**
 - No agents: JMX, JRebel, Byteman, profilers, tracers, etc.





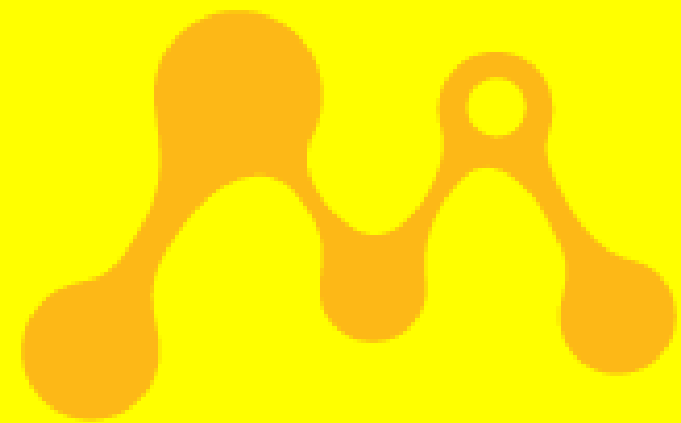
GraalVM

The word “Graal” comes from old French (or Dutch?) for “Grail”

The “**Graal**” Oracle project started out as a research project inside **Oracle Labs**, attempting to make a Java **compiler** while being **fast and easy** to maintain.

The “**VM**” in “GraalVM” comes from the fact that it **runs inside the JVM**.





MICROPROFILE™

OPTIMIZING ENTERPRISE JAVA



Microprofile



- A community-driven **specification**
- designed to provide a **baseline platform definition**
 - optimizes the **Enterprise** Java for microservices architecture
 - delivers application **portability** across multiple MicroProfile runtimes
- The founding vendors of MicroProfile offered their own microservices frameworks:
 - Open Liberty (*IBM*)
 - WildFly Swarm (*Red Hat*) → Thorntail → **Quarkus**
 - TomEE (*Tomitribe*)
 - Payara Micro (*Payara*)





QUARKUS



Quarkus



- From Redhat
- Similar to Spring Boot (but faster)
- A K8S native Java framework
 - uses **GraalVM** instead of traditional OpenJDK
 - applications that are smaller:
 - have faster startup time and are more suitable for container-based development

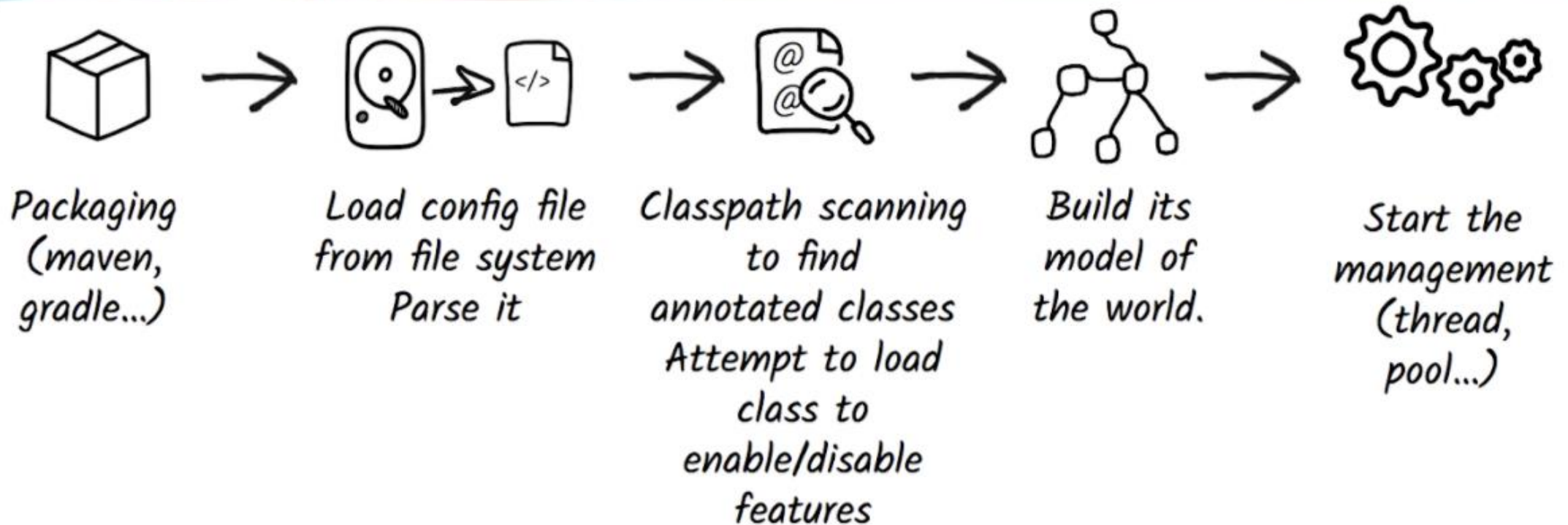




How does a regular framework start?

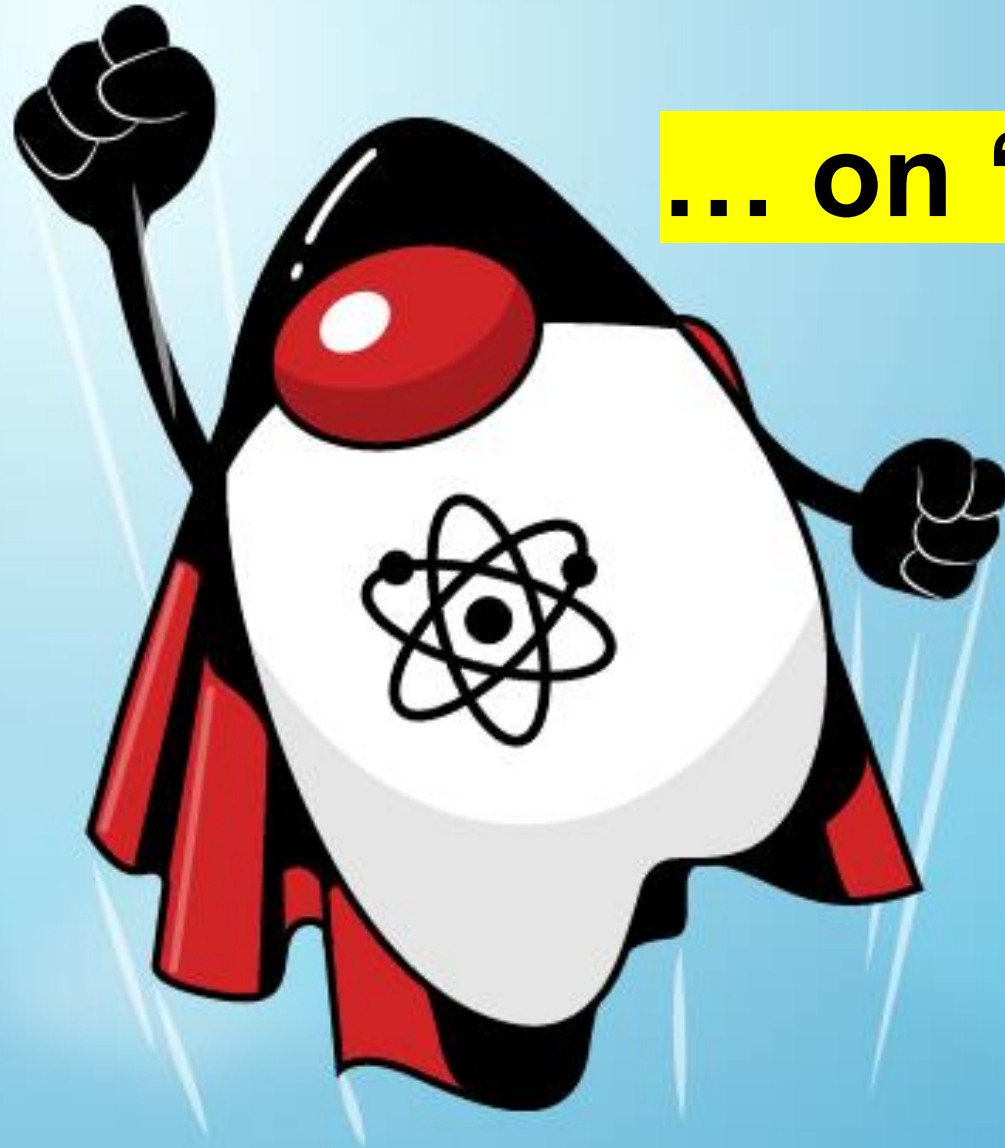
Build Time

Runtime



And in comes the Duke

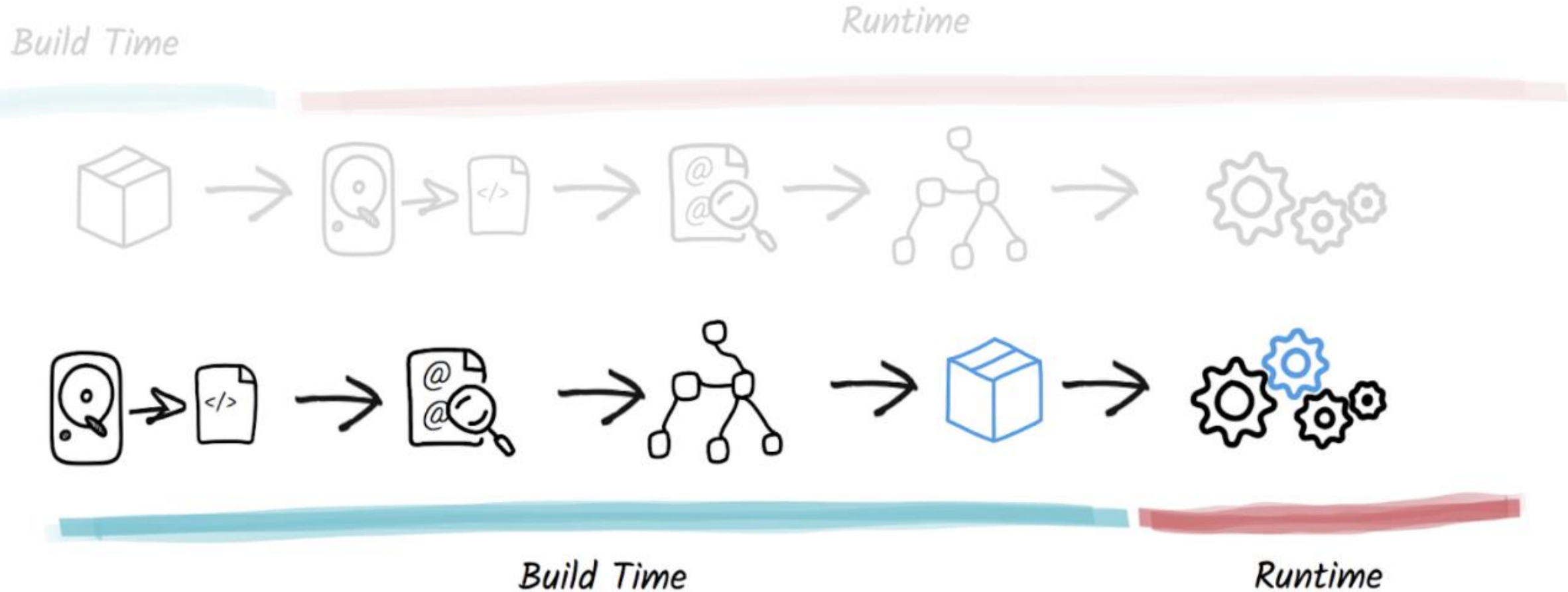
... on "Quarkus"



Supersonic...



The Quarkus way



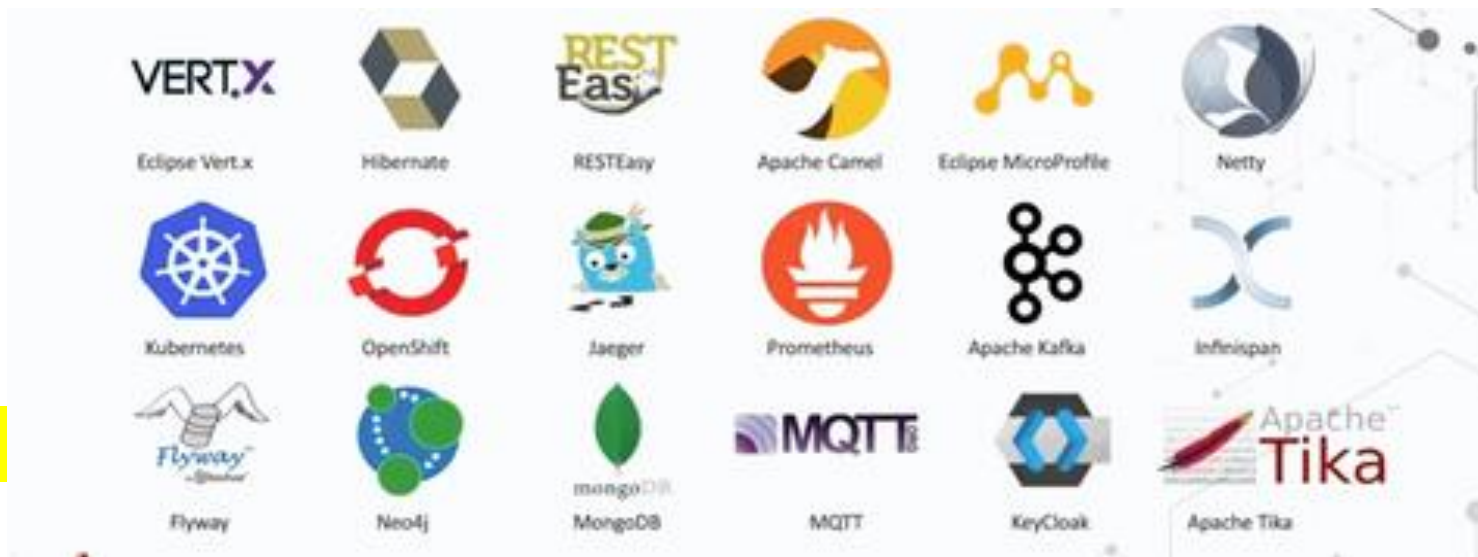
Pros and cons?



- User friendliness (JEE & Spring devs), solid framework, **“best of breed”** framework standards
 - Eclipse MicroProfile
 - Spring Dependency Injection
 - Hibernate ORM

■ Performance

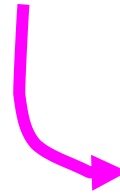
- Fast application start-up time
- Low memory consumption
- Almost immediate scaling of services
- Lower space requirements for native images



Pros and cons?



- Reducing the dynamic information generated during runtime can lead to problems in some scenarios.
- The severely limited possibilities for introspection may make it **difficult to debug** an application.
- The highly-optimized build process for native images **takes a long time...**





M I C R O N A U T TM



Micronaut



- Introduced in **2018** by the creators of the **Grails** framework
- JVM-based software framework for building:
 - *lightweight, modular applications and microservices*
- **Small memory footprints** and short startup times
- Builds its dependency injection data at compile time.
- Startup time and memory consumption are not tied to the size of an app's codebase
 - development of integration tests much easier and their execution much faster



Micronaut



- Micronaut **analyzes** metadata as soon as the application is compiled.
 - *During this compilation phase Micronaut will*

*Generate an additional set of classes that represent the **state** of the application already **preconfigured***

*This enables **dependency injection** (DI) and **aspect-oriented programming** (AOP) behavior to be applied much more efficiently when the application finally runs*



Micronaut



Quarkus is more reliant on **Jakarta EE** and Eclipse **MicroProfile** APIs,



Micronaut defines its own APIs and is more inspired by the **Spring** and **Grails** frameworks

- API is quite similar to the one in Spring and Grails

When to use?

You need native images

- graalVM / docker / Kubernetes
- Function as service

... but you cannot handle

- Living on the bleeding edge
- Frequent updates

... or you need something special

- Gradle/Groovy support maybe?



Spring Native



GraalVM™





A quick word on Spring & Spring Boot ...

- Spring MVC/**Spring Boot**
 - most dominant (?) Server-Side framework in Java
 - uses the conventional OpenJDK
 - **slowly loses its charm** in Cloud-Native Java Development
- Finally... Spring came up with **Spring Native**,
 - which will use GraalVM for Cloud-Native development

"debatable"



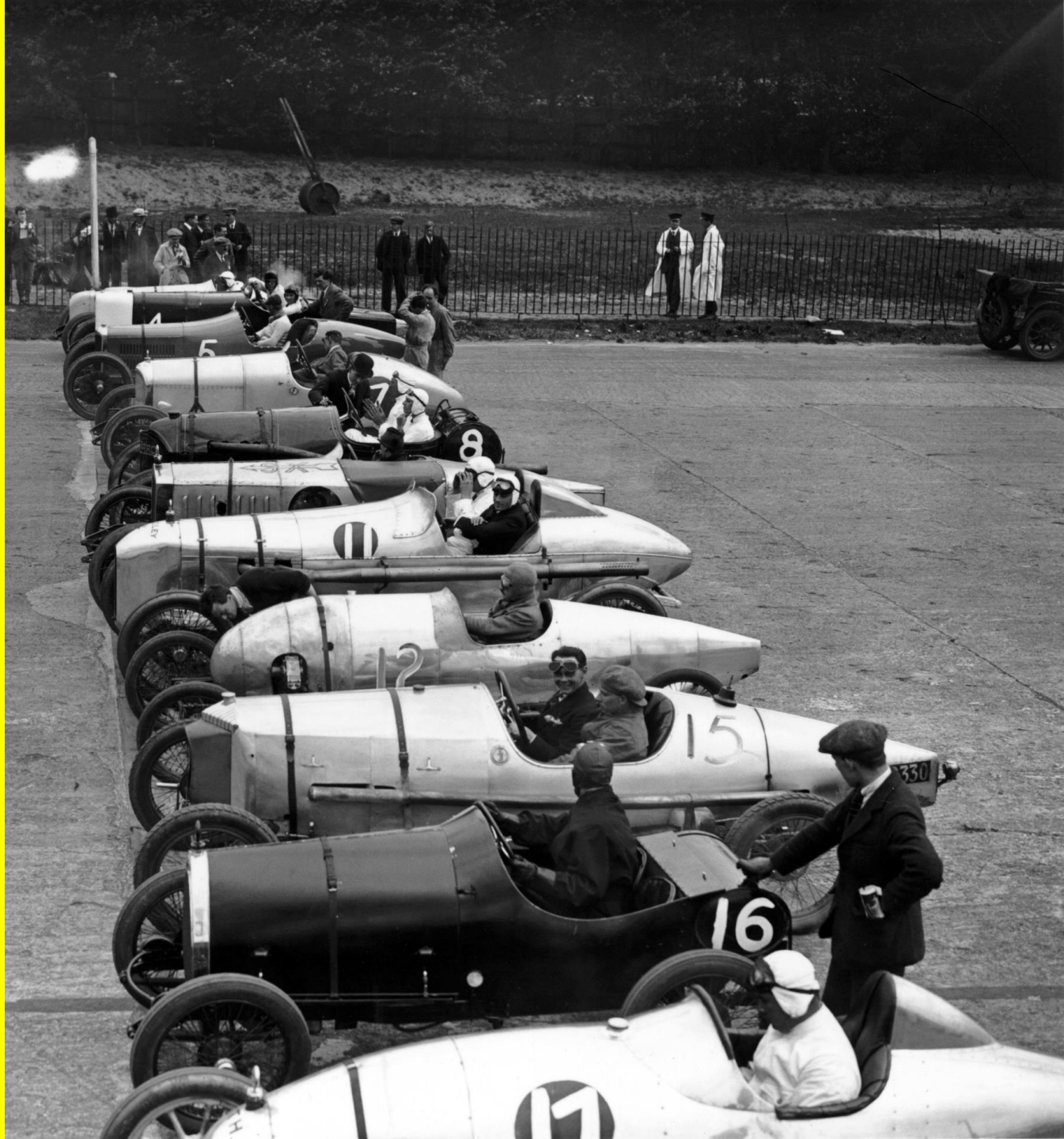
Difference with Spring



- **No class lazy loading** as everything shipped in the executables will be loaded in memory on startup
- **Classpath scanning** is fixed at **build time**
- **Static analysis** of your application from the main entry point is performed at **build time**
- **Removing the unused parts** of the codebase at **build time**
- Configuration is required for reflection, resources, and dynamic proxies



**Let's put them all
next to each other**



Comparisons

Quarkus

- *Is suitable for a **wide range of different application scenarios**.*
- *Other frameworks are more specific to some extent.*
- *Large community*

Open Liberty

- This **IBM framework** allows for the development of microservice applications with Java.
- Like Quarkus, Open Liberty comes with a live reload functionality.



Comparisons

Micronaut

- With the Micronaut framework, microservices and serverless applications can be programmed in Java.
- As with Quarkus, GraalVM is used here.
- Less performant than Quarkus

Spring Native

- Spring is probably the most popular Java framework for web applications.
- Spring Native is based on GraalVM and, in addition to the creation of microservices, it supports reactive programming and live reload. In a performance comparison, **Quarkus beats Spring Native.**
- An existing Spring project can be migrated to Quarkus relatively easily.
- On the other hand, **Quarkus comes with a more steep learning curve.**





Demo Time!

Spring Native

GraaIVM

Quarkus

Demo

- The Demo App : Discount Service
- Meet Graal VM for the first time
- Discount Service on Quarkus
- Discount Service on Spring Native

- Demo is available on [svanpe/the-fast-and-furious\(github.com\)](https://github.com/svanpe/the-fast-and-furious)



Discount Service

