

# Quarkus, Micronaut : Java is back

**Kevin Archet**

*Développeur Back*

**Lionel Guez**

*Développeur Back*

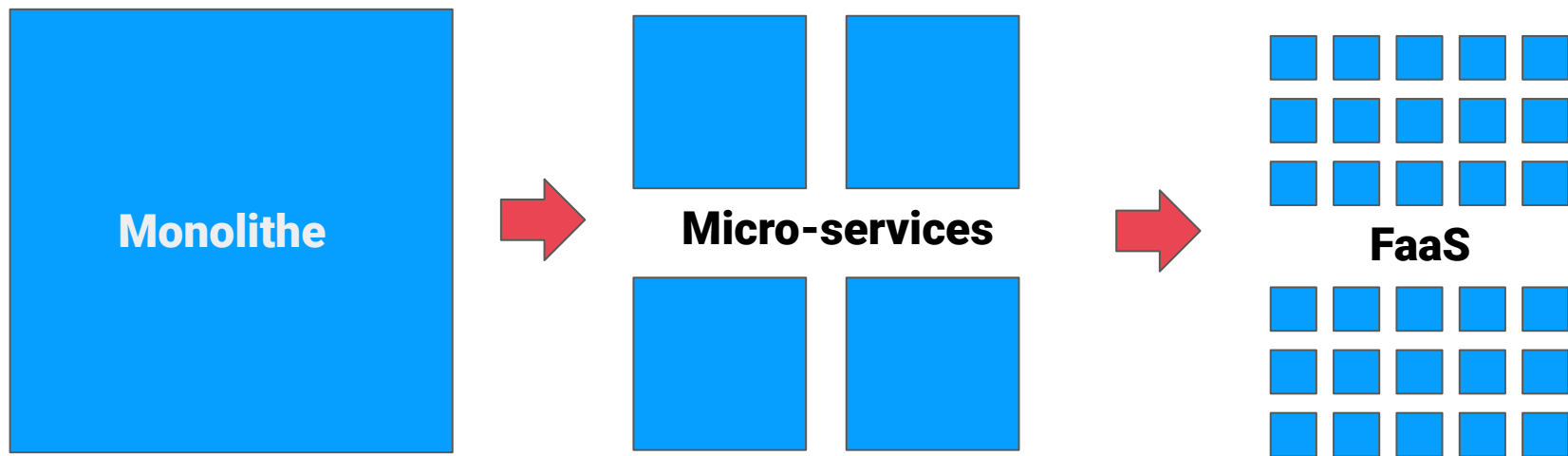
**XEBICON19**



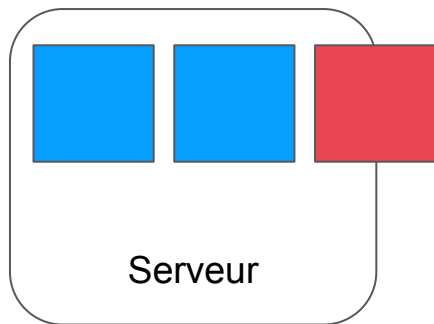
# Problématique : Les nouvelles architectures de développement



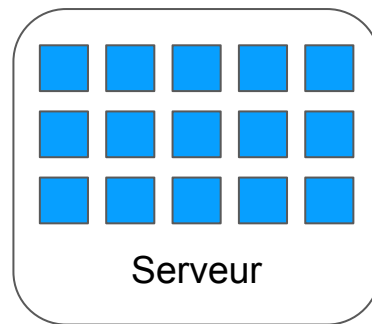
# Evolution architecturale



# Impact de l'empreinte mémoire

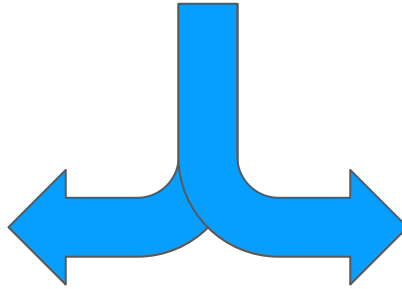


vs



# Impact du temps de démarrage

**Scalabilité  
performante**

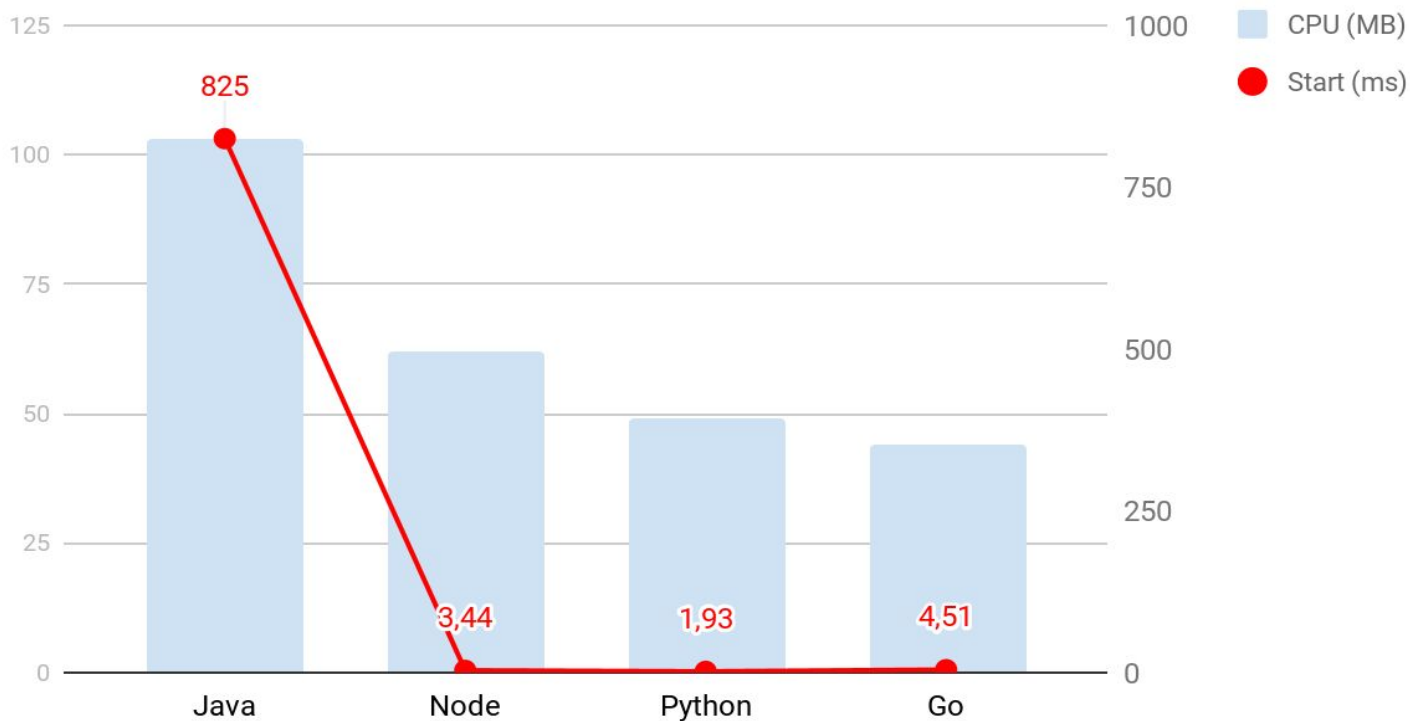


**FaaS**

# Et Java dans tout ça ?

# Java à la traîne

Benchmark "HelloWorld" AWS Lambda

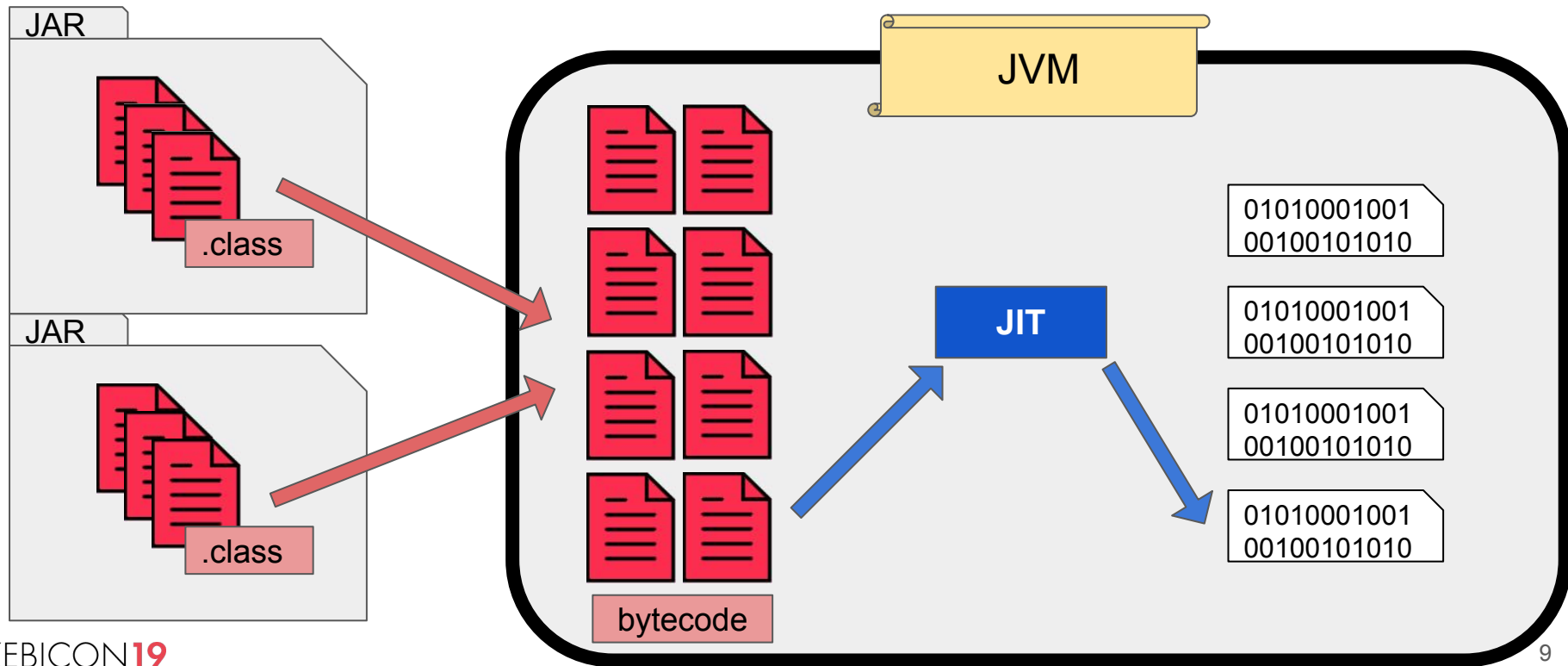


# D'où viennent ces mauvaises performances ?





# Fonctionnement de la JVM



## Mémoire de la JVM

### Mémoire spécifique à la JVM :

- Métadonnées du GC
- Statistiques du JIT
- etc...

### Mémoire dédiée à l'application :

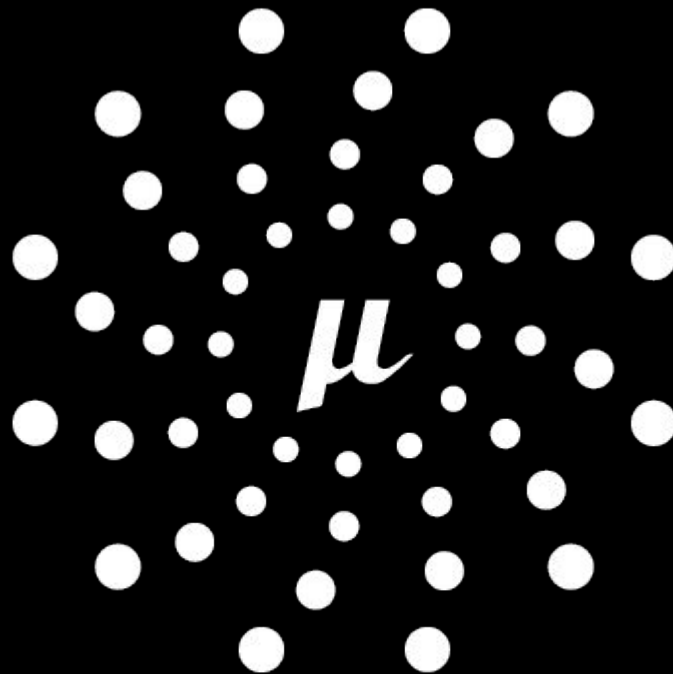
- Le fameux "Heap"  
= Code + mémoire

# Des frameworks qui en rajoutent une couche

- Lecture et parsing des fichiers de config
- Scan complet des classes pour récupérer les métadonnées (annotations, getters...)
- Création du métamodèle
- Préparation de la réflexion
- Création des proxies

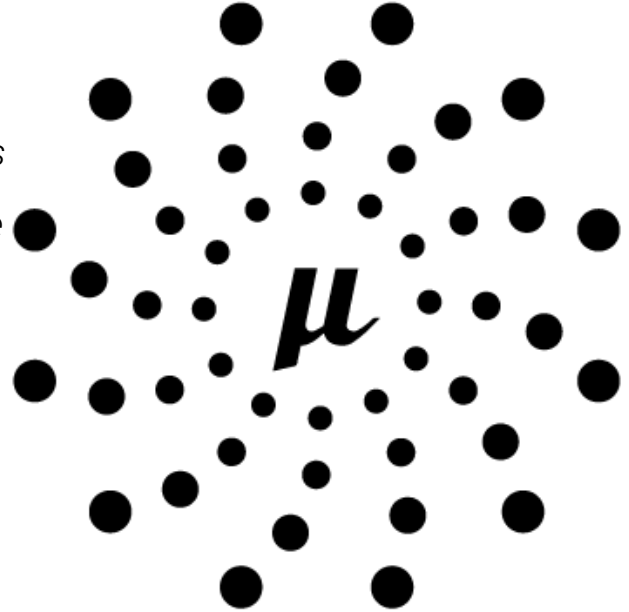
# Comment garder Java et les avantages de son écosystème ?

# Micronaut



# Micronaut : Présentation

- *"Micronaut is a modern, JVM-based, full stack microservices framework designed for building modular, easily testable microservice applications."*
- Micronaut un certain nombre de fonctionnalités haut niveau :
  - Injection de dépendances
  - Intégration de serveur web
  - Programmation orientée aspect
- .... Sans le surcoût de la réflexion Java, des proxies, etc...
- => Grâce à la génération de code au moment du build



# Micronaut : caractéristiques

- Framework disponible pour : Java, Groovy & Kotlin
- Fonctionne avec : Gradle & Maven
- Intègre des services typiques des microservices (Service Discovery...) et des facilités pour faire du FAAS
- Configuration centralisée
- Supporte la programmation réactive
- Intègre le server http Netty
- Dispose de connecteurs vers
  - Hibernate, MongoDB, Redis, etc....
  - Kafka, RabbitMQ,...

# Exemple de code

```
@Controller("/users")
public class UserController {

    private UserService userService;

    @Inject // L'annotation est optionnelle
    public UserController(UserService userService){
        this.userService = userService;
    }

    @Post(value = "{name}/{firstName}", produces = MediaType.TEXT_PLAIN)
    public void createUser(@PathVariable String name, @PathVariable String firstName){
        System.out.println(name + " ; " + firstName);
        userService.createUser(new User(firstName,name));
    }

    @Get(produces = MediaType.TEXT_PLAIN)
    public String allUsers(){
        //return "everyone";
        return userService.allUsers().stream().map(User::toString).collect(Collectors.joining(" ; "));
    }
}
```

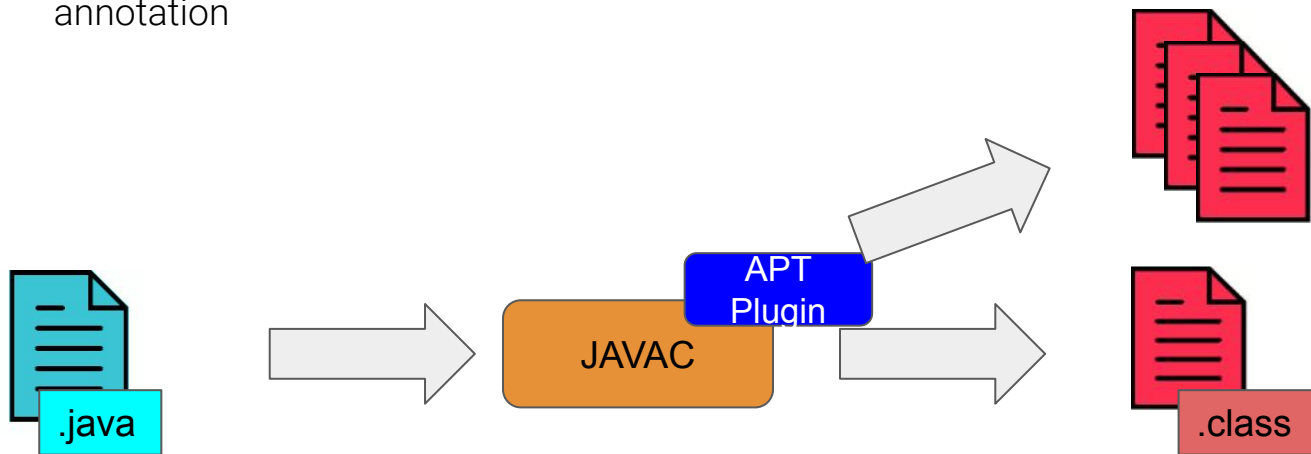


# Exemple de code (Micronaut Data)

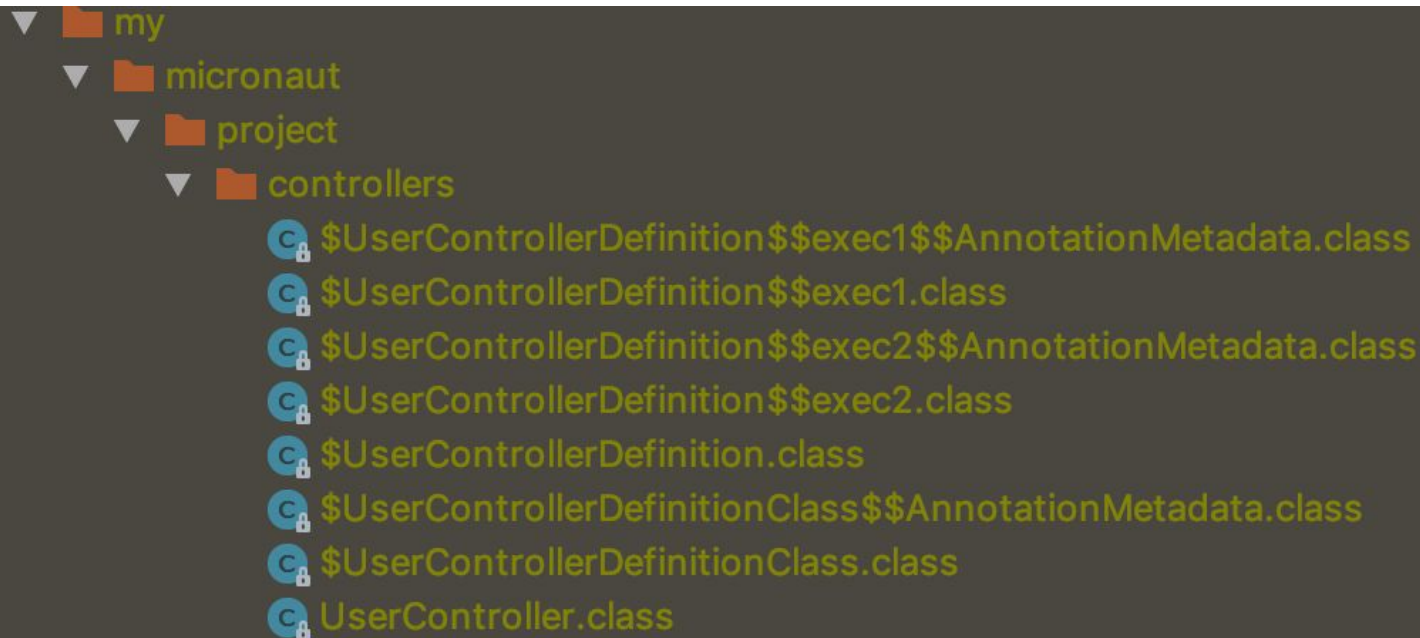
```
@Repository
public interface UserRepository extends CrudRepository<User, Long> {
    User findByFirstName(String firstName);
}
```

# Micronaut : sous le capot

- Sous le capot, Micronaut utilise un “Annotation Processor”
  - Fonctionnalité standard du compilateur Java
  - Ajoute une sorte de plug-in au compilateur
  - Permet, entre autre, de générer des classes lorsque le compilateur rencontre une annotation



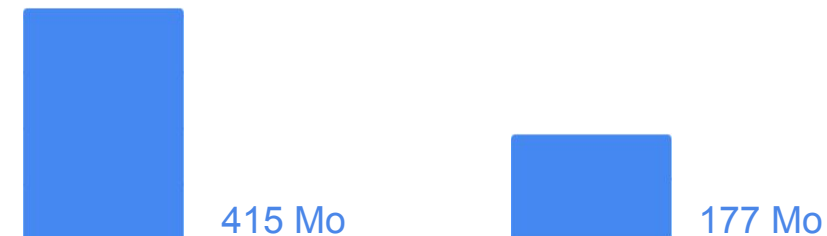
# Micronaut : sous le capot



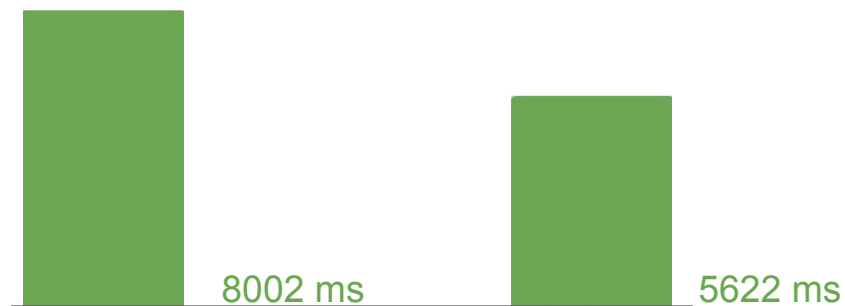
```
▼ my
  ▼ micronaut
    ▼ project
      ▼ controllers
        $UserControllerDefinition$$exec1$$AnnotationMetadata.class
        $UserControllerDefinition$$exec1.class
        $UserControllerDefinition$$exec2$$AnnotationMetadata.class
        $UserControllerDefinition$$exec2.class
        $UserControllerDefinition.class
        $UserControllerDefinitionClass$$AnnotationMetadata.class
        $UserControllerDefinitionClass.class
        UserController.class
```

# Performances

Mémoire



Temps de démarrage

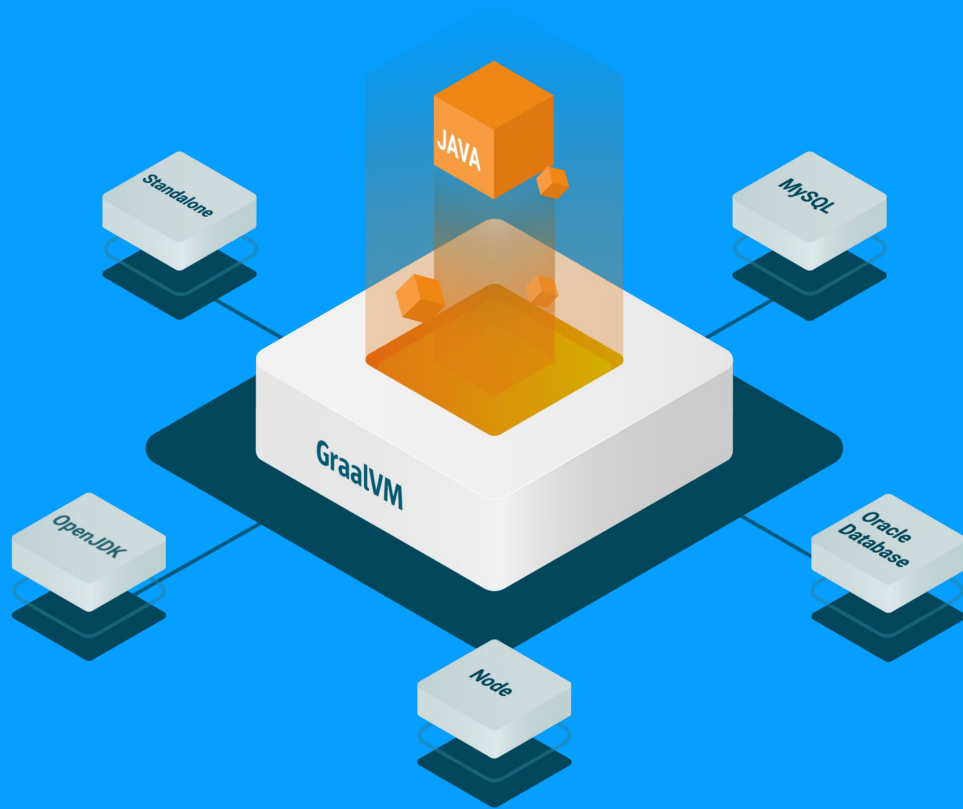


Stack Classique

Micronaut

(Spring + Spring Data)

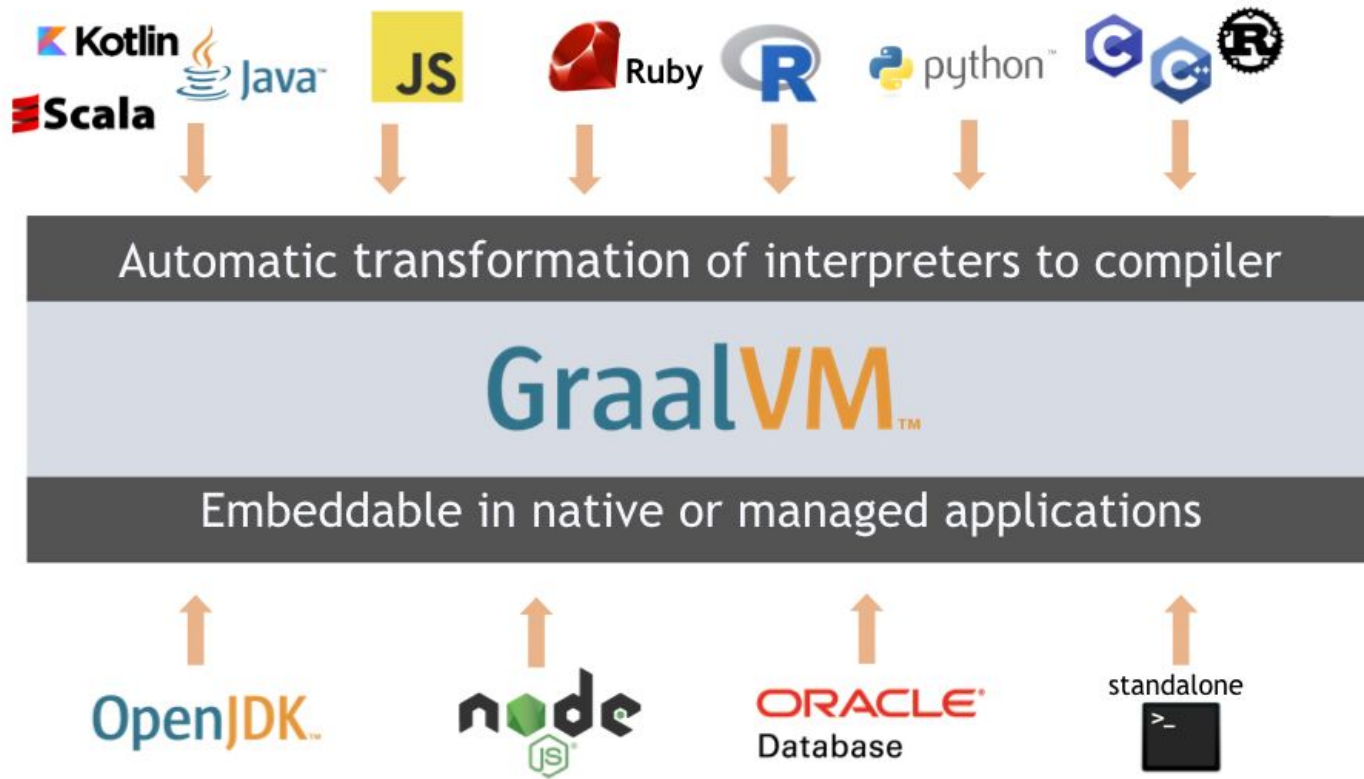
# GraalVM



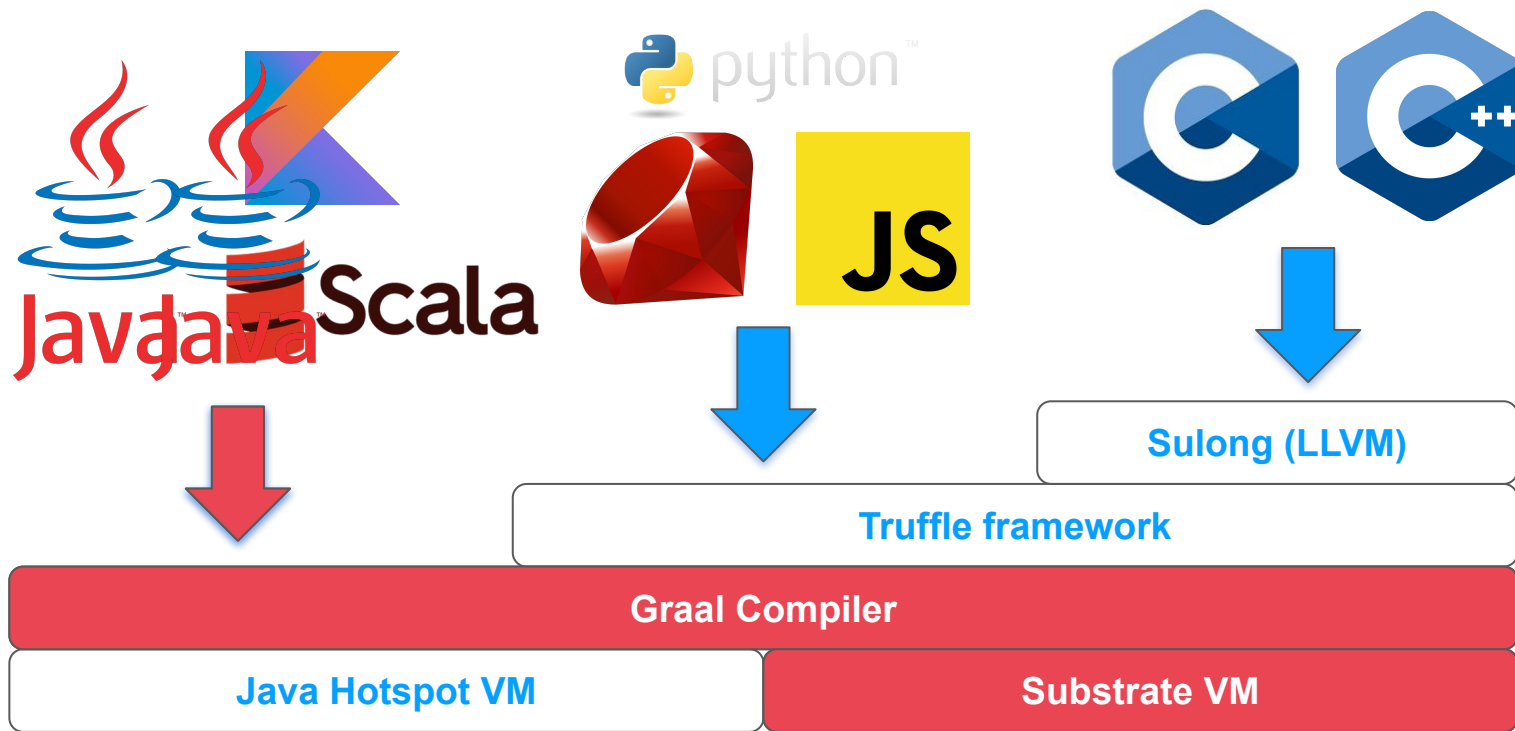
# En bref

- Nouvelle machine virtuelle universelle proposée par Oracle
- 8 ans de développement
- Sortie il y a 1 an

# Architecture générale



# Architecture détaillée





Graal Compiler



Compilation ahead-of-time



SubstrateVM

inutile

- Analyse de tous les chemins empruntés pendant

- Classes non utilisées
- Méthodes
- Attributs



Images natives



MacOS

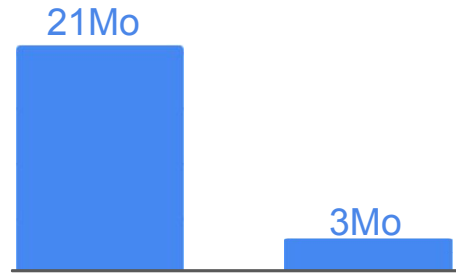


# Avantages

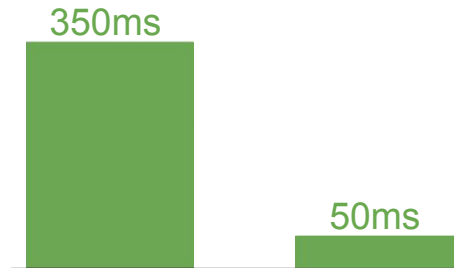
- Code léger
- Binaire
- Pas de temps de lancement de la JVM
- Pas de métaspace



Mémoire



Temps de démarrage

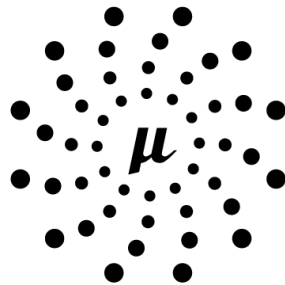


# Imp

- Class
- Invoke
- Finaliz
- Secur
- JVMT

```
[
  {
    "name" : "java.lang.Class",
    "allDeclaredConstructors" : true,
    "allPublicConstructors" : true,
    "allDeclaredMethods" : true,
    "allPublicMethods" : true,
    "allDeclaredClasses" : true,
    "allPublicClasses" : true
  },
  {
    "name" : "java.lang.String",
    "fields" : [
      { "name" : "value", "allowWrite" : true },
      { "name" : "hash" }
    ],
    "methods" : [
      { "name" : "<init>", "parameterTypes" : [] },
      { "name" : "<init>", "parameterTypes" : ["char[]"] },
      { "name" : "charAt" },
      { "name" : "format", "parameterTypes" : ["java.lang.String", "java.lang.Object[]"] }
    ]
  },
  {
    {
      "name" : "java.lang.String$CaseInsensitiveComparator",
      "methods" : [
        { "name" : "compare" }
      ]
    }
  }
]
```

g



Graeme Rocher

@graemerocher

Suivre



Support for [@graalvm](#) native image is coming in the next version of [@micronautfw](#) for [@java](#) and [@kotlin](#)

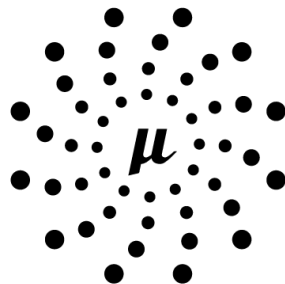
Startup time: 21ms

Memory Consumption: 20mb

Full reflection-free Dependency Injection and AOP.



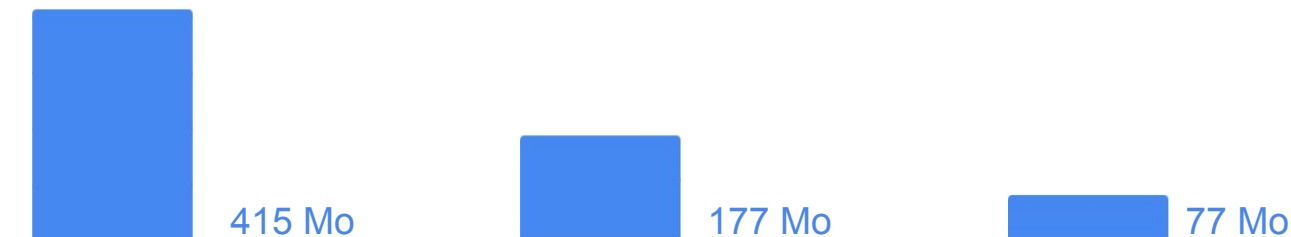
# GraalVM & Micronaut



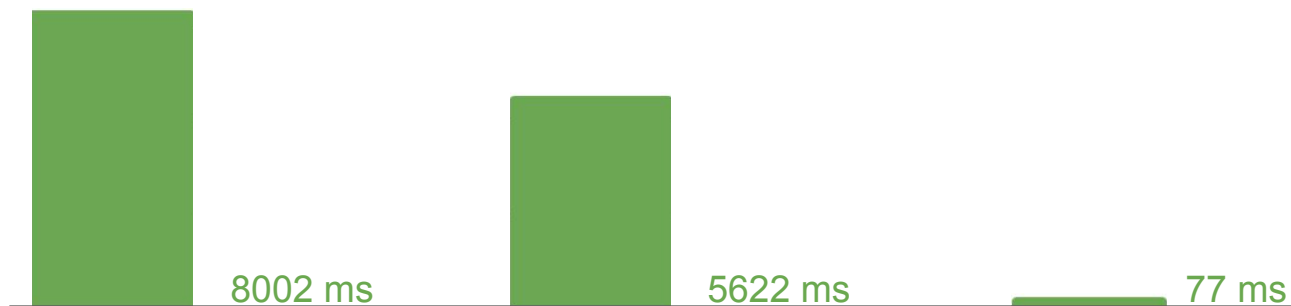
- Meilleur des deux mondes
- Micronaut était naturellement adapté à l'approche GraalVM
- Performance impressionnante
- Encore en "experimental"
- Beaucoup de bibliothèques sont non compatibles

# Performances

Mémoire



Temps de démarrage



Stack Classique  
(Spring + Spring Data)

Micronaut

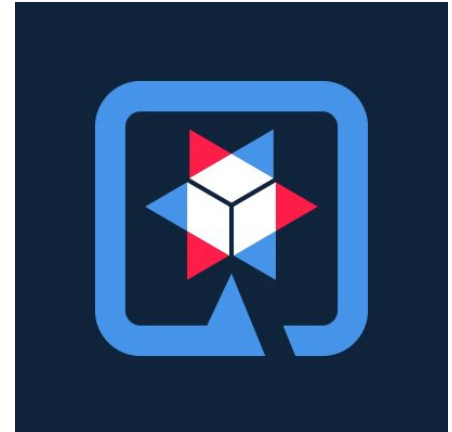
Micronaut +  
GraalVM

# Quarkus



# Quarkus : Présentation

- Stack applicative Java
- Sorti en 2019
- Développé par RedHat
- Objectifs
  - Accélérer/simplifier le développement d'applications
  - Rendre les applications Java adaptées au micro-services et Faas





# Quarkus : Caractéristiques générales

- Facilités de développement
  - Hot Reload
  - Configuration centralisée
  - Utilise les standards
  - Offre de nouvelles bibliothèques
- Repose sur MicroProfile
- Accepte la programmation impérative et réactive
- Compatible Maven/Gradle
- Compatible Java 8 et Kotlin

# Exemple de code

```
@Path("user")
public class UserController {

    @Inject
    private UserService userService;

    @POST
    @Produces(MediaType.TEXT_PLAIN)
    @Path("{name}/{firstName}")
    public void createUser(@PathParam("name") String name, @PathParam("firstName") String firstName){
        System.out.println(name + " ; " + firstName);
        userService.createUser(new User(firstName,name));
    }

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String allUsers() { return userService.allUsers().stream().map(User::toString).collect(Collectors.joining(" ; ")); }
}
```

# Hibernate with Panache

```
@Entity
public class User extends PanacheEntity {

    public String firstName;
    public String lastName;

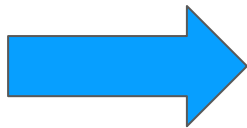
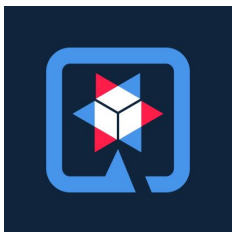
    public User() {
    }

    public User(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}
```

```
@ApplicationScoped
public class UserRepository {

    @Transactional
    public void createUser(User user){
        user.persist();
    }

    public List<User> allUsers(){
        return User.listAll();
    }
}
```

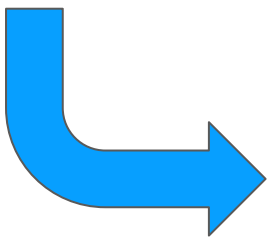


## Compilation “augmentée”

- Passer tout ce qui est au run time au build time
- Élimine le code mort d'un point de vue framework



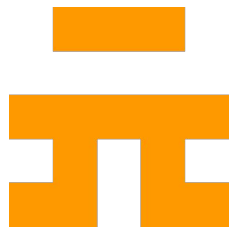
**Extensions**



```
<dependency>  
  <groupId>io.quarkus</groupId>  
  <artifactId>quarkus-resteasy</artifactId>  
</dependency>
```



Prometheus



JDBC

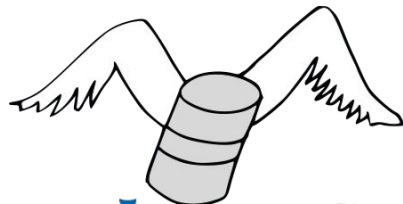


kubernetes



by Iev Polyakov (com)

Netty.fr



Flyway™  
by boxfuse™



Infinispan

VERT.X



APACHE

kafka™

REST  
Eas



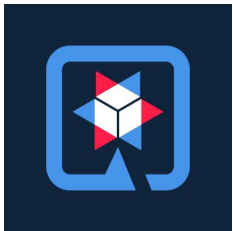
MICROPROFILE™  
OPTIMIZING ENTERPRISE JAVA



SMALLRYE



Apache  
Camel



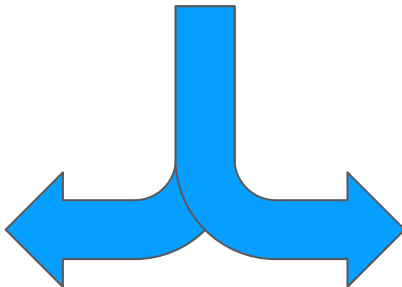
## Framework build-time

- Fait tout au moment de la compilation
- Élimine le code mort d'un point de vue framework



**Extensions**

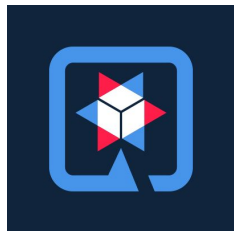
GraalVM™



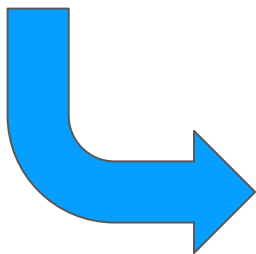
**HotspotVM**

# La symbiose

- Liste les classes pour la réflexion
- Liste les ressources
- A déjà éliminé du code mort
- Génération de l'image native



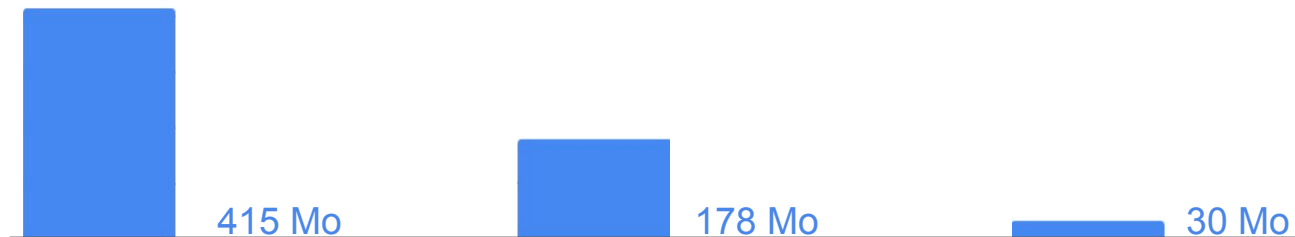
GraalVM™



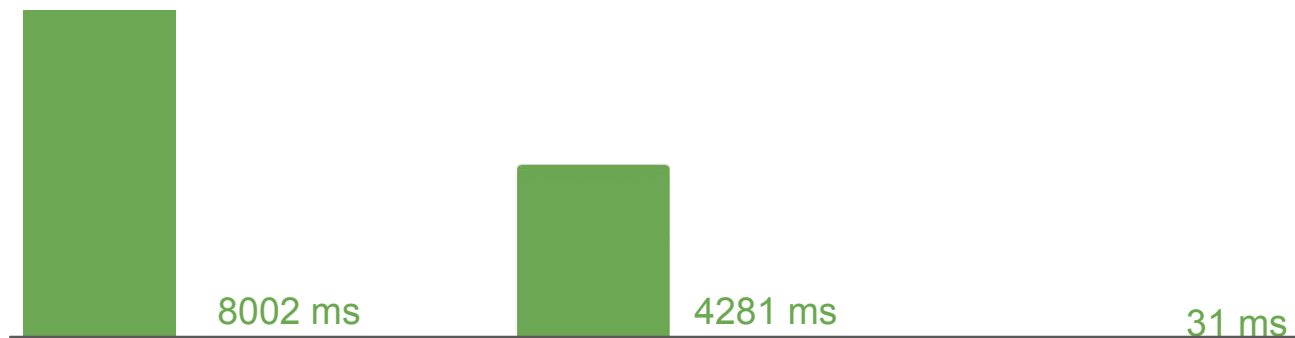
**Gain supplémentaire en mémoire et temps de démarrage**

# Performances

Mémoire



Temps de démarrage



Stack Classique  
(Spring + Spring Data)

Quarkus

Quarkus +  
GraalVM



# Le revers de la médaille

- Très récent
  - Manque de maturité
- Beaucoup de bibliothèques non compatibles
- Java 8
- Temps de compilation sur GraalVM

# Discussion



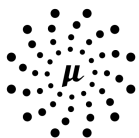
# Comparaison Micronaut/Quarkus

## Points communs

- Globalement, la même idée
  - Se passer au maximum de la réflexion et des proxies
  - Utiliser, si possible, GraalVM
- Performances comparable
- Parfaitement adaptés aux nouvelles architectures

# Comparaison Micronaut/Quarkus

## Différences



Java, Kotlin & Groovy

Basé sur des Annotation Processors

Inspiré par Spring

Compatibilité GraalVM est arrivée après

- *cherche à intégrer des bibliothèques tierces*



Java, Kotlin

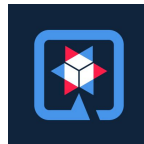
Basé sur des plugins maven

Repose sur Microprofile/J2EE

A été conçu avec GraalVM en tête

- *modifie les bibliothèques tierces (extensions)*

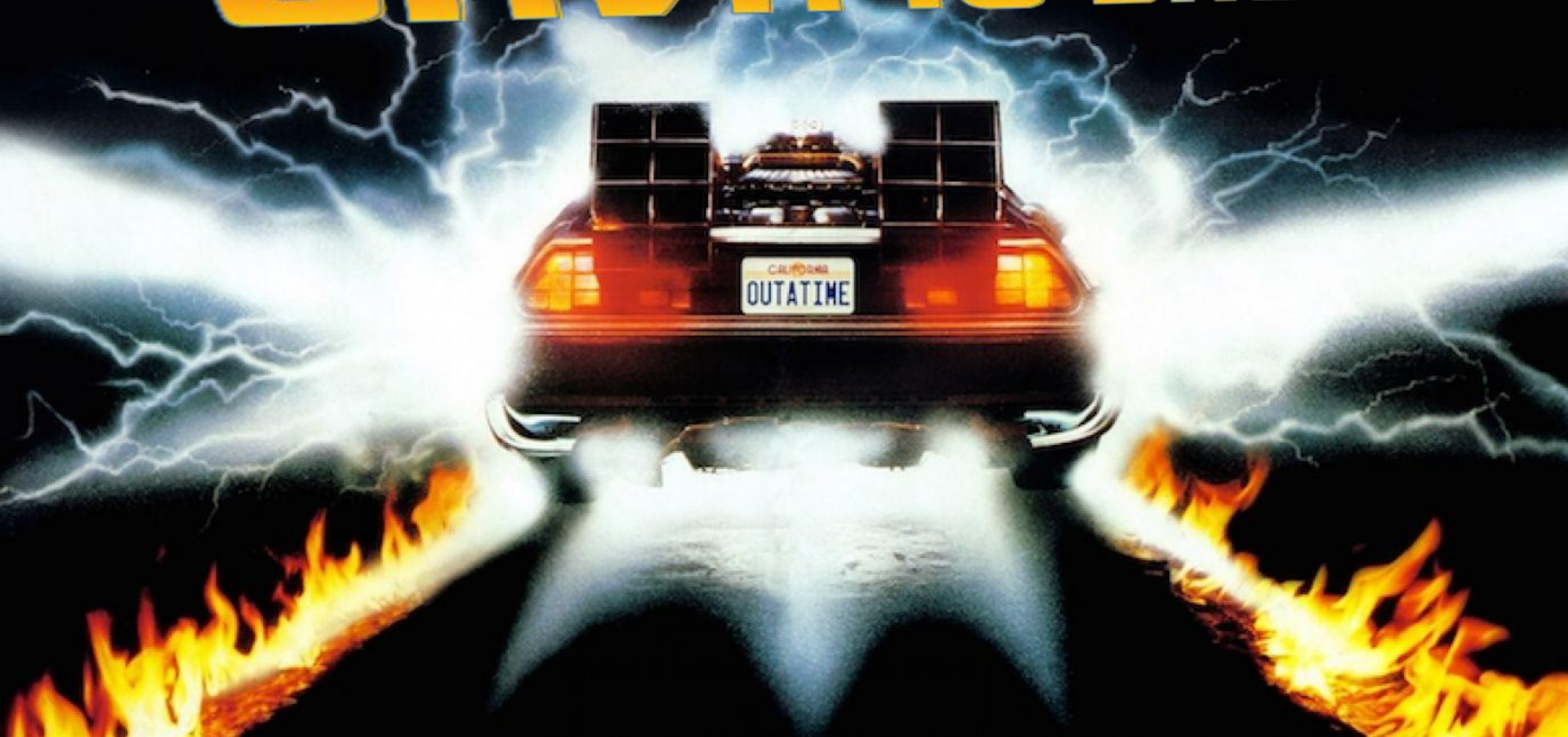
Hot reload



# Take Away

- GraalVM va permettre de compiler nativement du Java (Pivotal commence l'adaptation de Spring pour GraalVM)
- Micronaut et Quarkus font partie des premiers frameworks à initier un nouveau paradigme pour les frameworks Java : moins de réflexion, plus de traitement Ahead of Time
- Les acteurs du monde Java sont en train d'adapter Java aux architectures modernes (microservice, FaaS, Cloud-Native,..)

# JAVA IS BACK



# Des questions ?



@Xebiconfr - #Xebicon19



[https://github.com/LionelGuez/test\\_libraries\\_with\\_graal/](https://github.com/LionelGuez/test_libraries_with_graal/)

