



**CloudAPPS**  
MASTER

DESARROLLO Y DESPLIEGUE DE APLICACIONES EN LA NUBE

# REACTOR EN SPRING VS MUTINY EN QUARKUS

CURSO ACADÉMICO 2021/2022  
TRABAJO DE FIN DE MÁSTER

## ÍNDICE

- ▶ Introducción y objetivos
- ▶ Quarkus y Mutiny
- ▶ Comparativa Quarkus y Spring
- ▶ Pruebas de carga
- ▶ Conclusiones
- ▶ Trabajos futuros



# INTRODUCCIÓN Y OBJETIVOS

# ¿POR QUÉ ESTÁ COMPARATIVA?

- ▶ Aparición de nuevos frameworks modernos orientados al Cloud: Quarkus, Micronaut y Helidon
- ▶ Spring es genial por todo lo que aporta, pero ¿es mejor solución en cuanto a la arquitectura, rendimiento o consumo de recursos en todos los apartados?

# ¿POR QUÉ QUARKUS?

- ▶ Micronaut, está apoyado por la gente Grails, un framework que no terminó por llegar a buen puerto
- ▶ Helidon, no nos pareció que estuviera lo suficiente maduro y soportado por Oracle, y todos conocemos a Oracle
- ▶ Quarkus esta apoyado por RedHat, esto da una gran confianza en que el proyecto continúe en el tiempo y se siga desarrollando y mejorando

# QUARKUS Y MUTINY

## CURVA DE APRENDIZAJE Y DOCUMENTACIÓN

- ▶ Rapida, cómoda y agradable
- ▶ Buena documentación y ejemplos
- ▶ Gran variedad de extensiones y librerías

## MUTINY

- ▶ Arquitectura orientada a eventos
- ▶ API sencilla y legible
- ▶ Multi y Uni son los tipos que permiten manejar elementos asincronos



# COMPARATIVA QUARKUS VS SPRING

# SERVIDOR HTTP REST Y PERSISTENCIA

Imperativo

## Imperativo

## Reactiva

### Cliente Rest HTTP

Endpoint Rest que invoca a otro servicio secundario

### Servicio Rest HTTP y Persistencia

Desarrollo de un API (sobre películas) que ofrece diferentes operaciones y conecta con una base de datos

### gRPC

Servicio gRPC que recibe un número, que multiplicado por un número aleatorio

### AMQP

Produce y envía unos pocos mensajes, para luego ser consumidos

### WebSockets

Desarrollo de un chat para usuarios

Endpoint Rest que invoca a otro servicio secundario

Desarrollo de un API (sobre películas) que ofrece diferentes operaciones y conecta con una base de datos

Servicio gRPC que recibe un número, que multiplicado por un número aleatorio

Implementación de un front donde añades una ciudad y devuelve el tiempo hace en mayúsculas o minúsculas

Web donde se visualiza la cantidad de usuarios conectados a la aplicación

# QUARKUS RESTEASY

```
● ● ●

1 @Path("/movies")
2 public class MovieResource {
3
4     @GET
5     @Path("/{movieId}")
6     public Response getMovieById(@PathParam("movieId") Long movieId) {
7         ...
8     }
9
10    @GET
11    public List<MovieResponse> getAllMoviesByPage(@QueryParam("page") Optional<Integer> pageNumber) {
12        ...
13    }
14
15    @POST
16    public MovieResponse saveMovie(@RequestBody MovieRequest movieRequest) {
17        ...
18    }
19 }
```

# SPRING MVC

```
1 @RestController
2 @RequestMapping("/movies")
3 public class MovieController {
4
5     @GetMapping("/{movieId}")
6     public ResponseEntity<MovieResponse> getMovieById(@PathVariable("movieId") Long movieId) {
7         ...
8     }
9
10    @GetMapping
11    public List<MovieResponse> getAllMoviesByPage(@RequestParam("page") Optional<Integer> pageNumber) {
12        ...
13    }
14
15    @PostMapping
16    public MovieResponse saveMovie(@RequestBody MovieRequest movieRequest) {
17        ...
18    }
19 }
```

# QUARKUS PANACHE - ACTIVE RECORD PATTERN

```
1 @Entity(name = "movies")
2 public class Movie extends PanacheEntityBase {
3
4     @Id
5     @GeneratedValue(strategy = GenerationType.IDENTITY)
6     private Long id;
7     private String poster;
8     ...
9
10    public static List<Movie> findByPage(Page page) {
11        return Movie.findAll().page(page).list();
12    }
13 }
```

# QUARKUS PANACHE - REPOSITORY PATTERN

```
● ● ●

1 @ApplicationScoped
2 public class MovieRepository implements PanacheRepository<Movie> {
3
4     public List<Movie> findByPage(Page page) {
5         return findAll().page(page).list();
6     }
7
8 }
```

# QUARKUS - SPRING DATA JPA

```
● ● ●  
1 public interface MovieRepository extends JpaRepository<Movie, Long> {  
2  
3     @Modifying  
4     @Query("update Movie m set m.rating = :rating where m.id = :movieId")  
5     void update(@Param("rating") Double rating, @Param("movieId") Long movieId);  
6  
7 }  
8
```

# SPRING DATA JPA

```
● ● ●  
1 public interface MovieRepository extends JpaRepository<Movie, Long> {  
2  
3     @Modifying  
4     @Query("update Movie m set m.rating = :rating where m.id = :movieId")  
5     void update(@Param("rating") Double rating, @Param("movieId") Long movieId);  
6  
7 }  
8
```

# AMQP

## Imperativo

# QUARKUS - QUARKIVERSE RABBITMQ CLIENT

```
● ● ●

1 @Startup
2 @ApplicationScoped
3 public class RabbitConfiguration {
4
5     private final Channel channel;
6
7     @Inject
8     public RabbitConfiguration(RabbitMQClient rabbitMQClient, RabbitProperties rabbitProperties) {
9
10        try {
11
12            // Config exchange, queue and binding
13            Connection connection = rabbitMQClient.connect();
14            channel = connection.createChannel();
15            channel.exchangeDeclare(rabbitProperties.getExchangeName(), BuiltinExchangeType.DIRECT, true);
16            channel.queueDeclare(rabbitProperties.getQueueName(), true, false, false, null);
17            channel.queueBind(rabbitProperties.getQueueName(), rabbitProperties.getExchangeName(), rabbitProperties.getRouterKey());
18
19            // Set listener service class
20            ListenerService listenerService = new ListenerService(channel);
21            channel.basicConsume(rabbitProperties.getQueueName(), true, listenerService);
22
23        } catch (IOException e) {
24            throw new UncheckedIOException(e);
25        }
26
27    }
28
29    @Produces
30    public Channel getChannel() {
31        return this.channel;
32    }
33
34 }
```

# QUARKUS - QUARKIVERSE RABBITMQ CLIENT

```
1 public class ListenerService extends DefaultConsumer {  
2  
3     public ListenerService(Channel channel) {  
4         super(channel);  
5     }  
6  
7     @Override  
8     public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body) {  
9         ...  
10    }  
11 }
```

```
1 byte[] messageBytes = message.getBytes(StandardCharsets.UTF_8);  
2 AMQP.BasicProperties basicProperties = new AMQP.BasicProperties();  
3  
4 channel.basicPublish(  
5     rabbitProperties.getExchangeName(), rabbitProperties.getRouterKey(),  
6     basicProperties, messageBytes);
```

# SPRING AMQP

```
●●●

1 @Configuration
2 public class RabbitConfiguration {
3
4     @Bean
5     public Queue myQueue() {
6         return QueueBuilder.durable(rabbitProperties.getQueueName()).build();
7     }
8
9     @Bean
10    public Exchange exchange() {
11        return new DirectExchange(rabbitProperties.getExchangeName());
12    }
13
14    @Bean
15    public Binding binding(Queue queue, Exchange exchange) {
16        return BindingBuilder
17            .bind(queue)
18            .to(exchange)
19            .with(rabbitProperties.getRouterKey())
20            .noargs();
21    }
22 }
```

## SPRING AMQP



```
1 @RabbitListener(queues = "#{rabbitProperties.getQueueName()}")
2 public void handleDelivery(String message) {
3     log.info("Received: {}", message);
4 }
```



```
1 rabbitTemplate.convertAndSend(rabbitProperties.getQueueName(), message);
```

# SERVIDOR HTTP REST Y PERSISTENCIA

Reactiva

# QUARKUS REACTIVE RESTEASY

```
● ● ●

1 @Path("/movies")
2 public class MovieResource {
3
4     @GET
5     @Path("/{movieId}")
6     public Uni<Response> getMovieById(@PathParam("movieId") Long movieId) {
7
8         return Movie.findById(movieId)
9             .onItem().ifNull().failWith(NotFoundException::new)
10            .onItem().castTo(Movie.class)
11            .map(this::convertToResponse)
12            .map(Response::ok)
13            .onFailure(NotFoundException.class).recoverWithItem(Response.status(NOT_FOUND))
14            .map(Response.ResponseBuilder::build);
15    }
16
17    @GET
18    public Uni<List<MovieResponse>> getAllMoviesByPage(@QueryParam("page") Optional<Integer> pageNumber) {
19        ...
20    }
21
22    @POST
23    public Uni<MovieResponse> saveMovie(@RequestBody MovieRequest movieRequest) {
24        ...
25    }
26
27    ...
28 }
```

# SPRING WEBFLUX

```
● ● ●

1 @RestController
2 @RequestMapping("/movies")
3 public class MovieController {
4
5     @GetMapping("/{movieId}")
6     public Mono<ResponseEntity<MovieResponse>> getMovieById(@PathVariable("movieId") Long movieId) {
7         return movieRepository.findById(movieId)
8             .map(this::convertToResponse)
9             .map(ResponseEntity::ok)
10            .defaultIfEmpty(ResponseEntity.notFound().build());
11    }
12
13    @GetMapping
14    public Mono<List<MovieResponse>> getAllMoviesByPage(@RequestParam("page") Optional<Integer> pageNumber) {
15        ...
16    }
17
18    @PostMapping
19    public Mono<MovieResponse> saveMovie(@RequestBody MovieRequest movieRequest) {
20        ...
21    }
22    ...
23 }
```

# QUARKUS REACTIVE PANACHE - ACTIVE RECORD PATTERN

```
● ● ●

1 @Entity(name = "movies")
2 public class Movie extends PanacheEntityBase {
3
4     @Id
5     @GeneratedValue(strategy = GenerationType.IDENTITY)
6     private Long id;
7     private String poster;
8     ...
9
10    public static Uni<List<Movie>> findByPage(Page page) {
11        return Movie.findAll().page(page).list();
12    }
13 }
```

# QUARKUS REACTIVE PANACHE - REPOSITORY PATTERN



```
1 @ApplicationScoped
2 public class MovieRepository implements PanacheRepository<Movie> {
3
4     public Uni<List<Movie>> findByPage(Page page) {
5         return findAll().page(page).list();
6     }
7
8 }
```

# SPRING DATA R2DBC

```
1 @Repository
2 public interface MovieRepository extends ReactiveCrudRepository<Movie, Long> {
3
4     Flux<Movie> findAllBy(Pageable pageable);
5
6     @Modifying
7     @Query("update movies m set m.rating = :rating where m.id = :movieId")
8     Mono<Void> update(@Param("rating") Double rating, @Param("movieId") Long movieId);
9
10 }
```

# AMQP

## Reactivio

# QUARKUS SMALLRYE REACTIVE MESSAGING RABBITMQ

```
 1 @Path("/forecast")
 2 public class ForecastResource {
 3
 4     @Channel("request-forecast")
 5     MutinyEmitter<Forecast> processForecast;
 6
 7     @Channel("forecast")
 8     Multi<Forecast> processedForecasts;
 9
10    @POST
11    public Uni<Forecast> processForecast(City city) {
12        return Uni.createFrom().item(city)
13            .map(City::getName)
14            .map(cityName -> new Forecast(UUID.randomUUID().toString(), cityName))
15            .call(forecast -> processForecast.send(forecast));
16    }
17
18    @GET
19    @Produces(MediaType.SERVER_SENT_EVENTS)
20    public Multi<Forecast> getProcessedForecasts() {
21        return processedForecasts;
22    }
23 }
```

```
 1 @Incoming("process-forecast")
 2 @Outgoing("processed-forecast")
 3 public Uni<Forecast> listener(Forecast receivedForecast) {
 4     return Uni.createFrom().item(receivedForecast)
 5         .map(forecast -> {
 6             String weather = getRandomWeather(forecast.getCity());
 7             return new Forecast(forecast.getId(), forecast.getCity(), weather);
 8         })
 9         .onItem().delayIt().by(getRandomDuration());
10 }
```

# SPRING STARTER AMQP Y REACTOR RABBITMQ

```
1 @Bean
2 public Mono<Connection> reactiveRabbitMQConnection(RabbitProperties rabbitProperties) {
3     ConnectionFactory connectionFactory = new ConnectionFactory();
4     connectionFactory.useNio();
5     connectionFactory.setHost(rabbitProperties.getHost());
6     connectionFactory.setPort(rabbitProperties.getPort());
7     connectionFactory.setUsername(rabbitProperties.getUsername());
8     connectionFactory.setPassword(rabbitProperties.getPassword());
9     return Mono.fromCallable(connectionFactory::newConnection).cache();
10 }
```

# SPRING STARTER AMQP Y REACTOR RABBITMQ

```
●●●

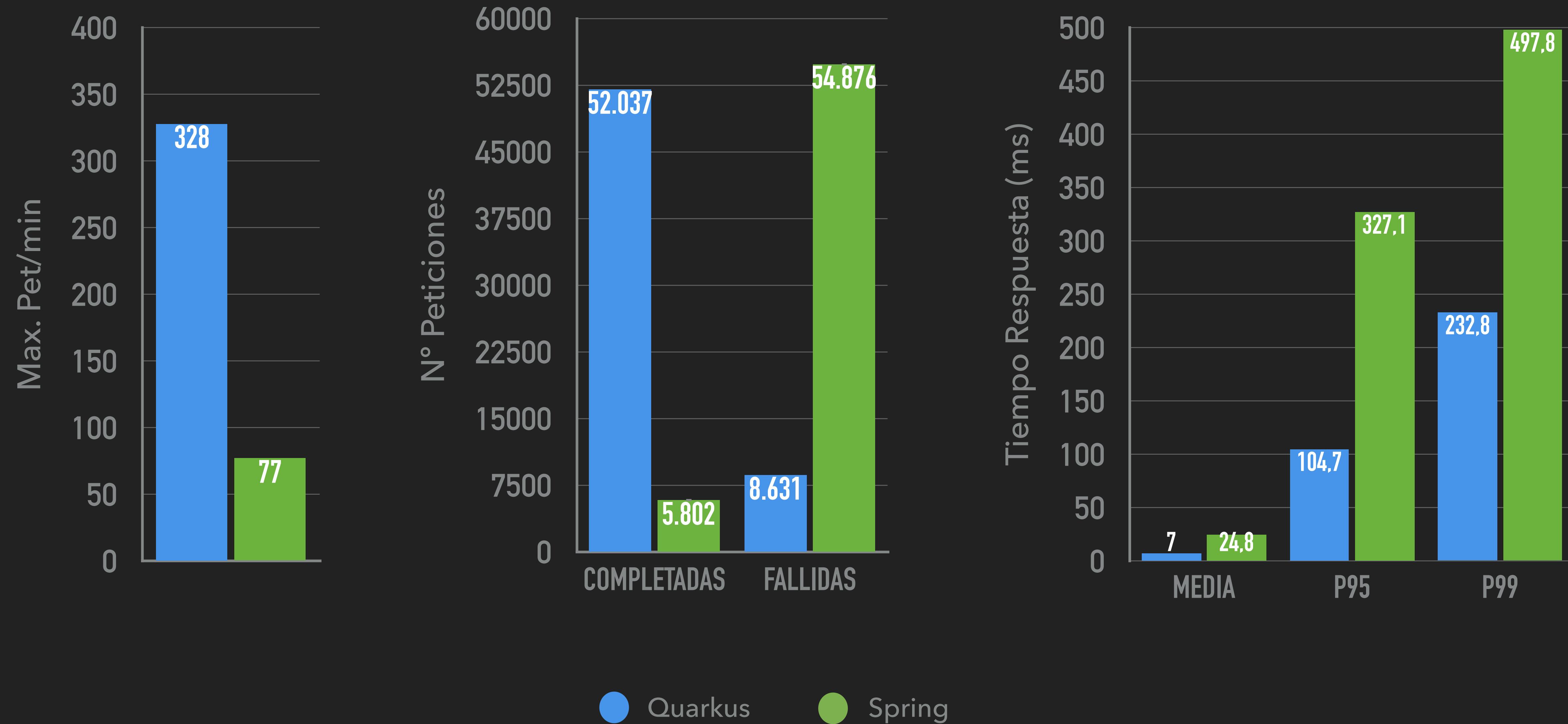
1 @Bean
2 public Sender sender(Mono<Connection> reactiveRabbitMQConnection) {
3     return RabbitFlux.createSender(new SenderOptions().connectionMono(reactiveRabbitMQConnection));
4 }
5
6 @Bean
7 public Receiver receiver(Mono<Connection> reactiveRabbitMQConnection) {
8     return RabbitFlux.createReceiver(new ReceiverOptions().connectionMono(reactiveRabbitMQConnection));
9 }
10
11 @Bean
12 public Flux<Delivery> processForecastListener(final Receiver receiver, Binding processForecastBinding) {
13     return receiver.consumeAutoAck(processForecastBinding.getDestination());
14 }
15
16 @Bean
17 public Flux<Delivery> processedForecastListener(final Receiver receiver, Binding processedForecastBinding) {
18     return receiver.consumeAutoAck(processedForecastBinding.getDestination());
19 }
```

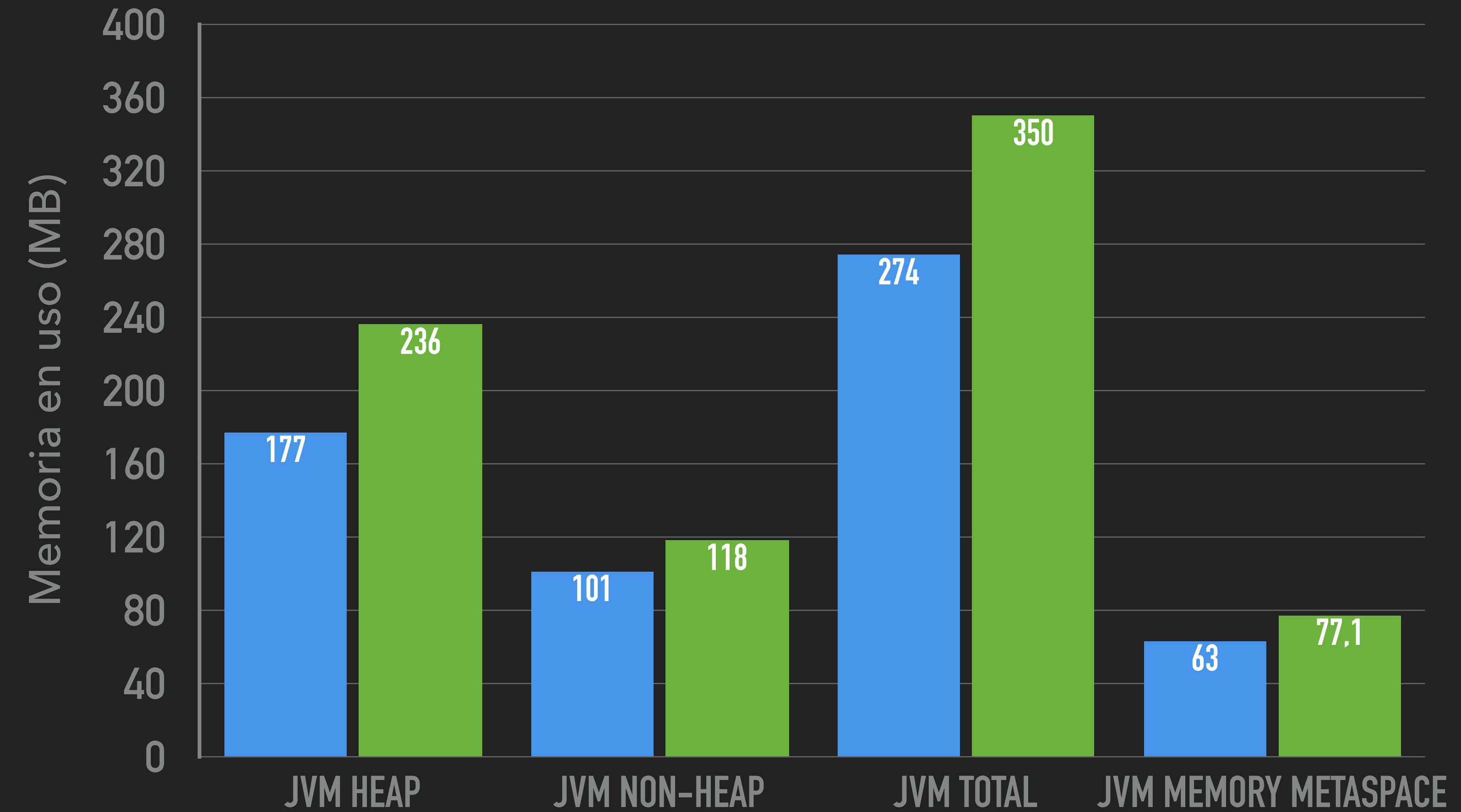
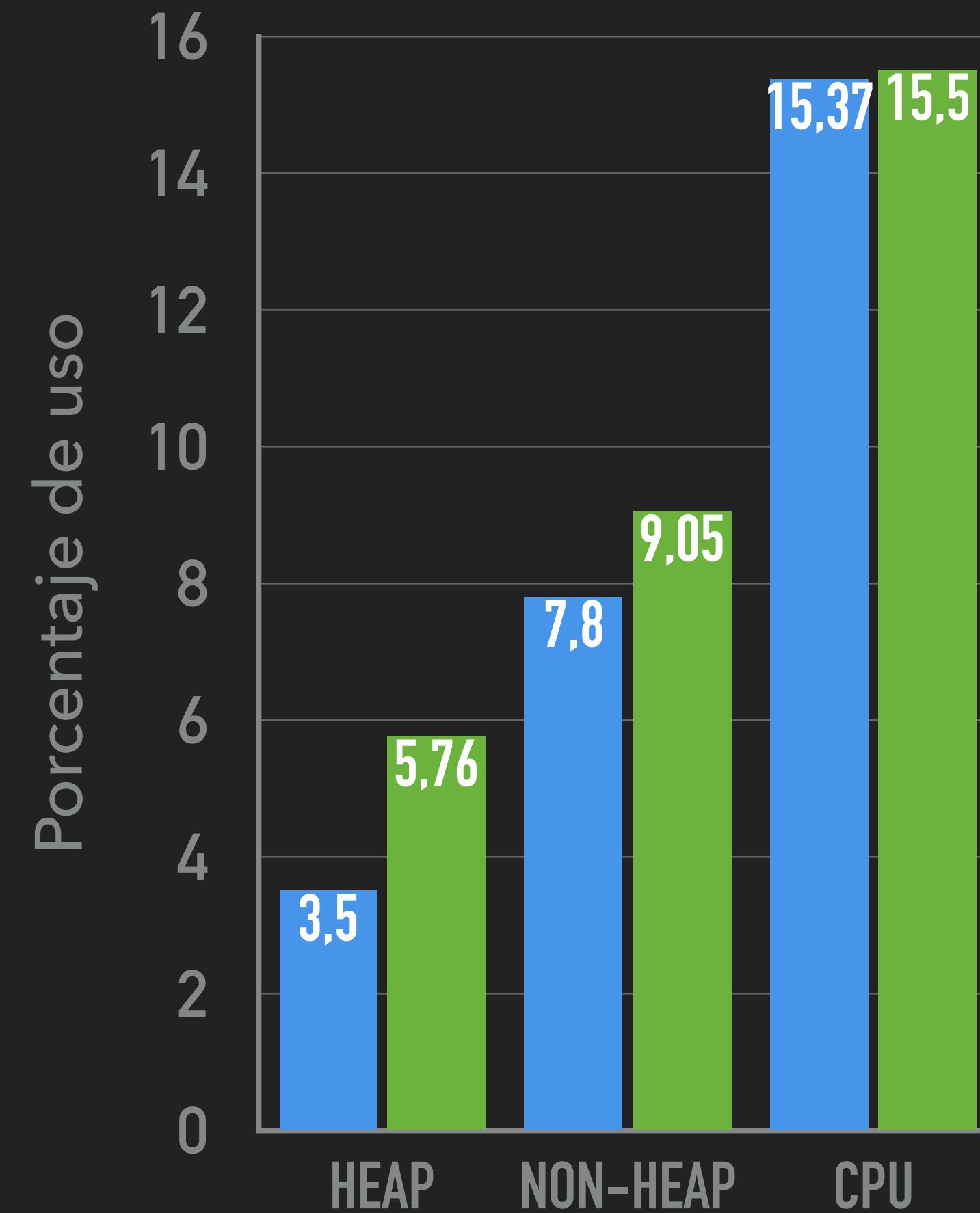


# PRUEBAS DE CARGA

# RESULTADOS DE RENDIMIENTO

Imperativo



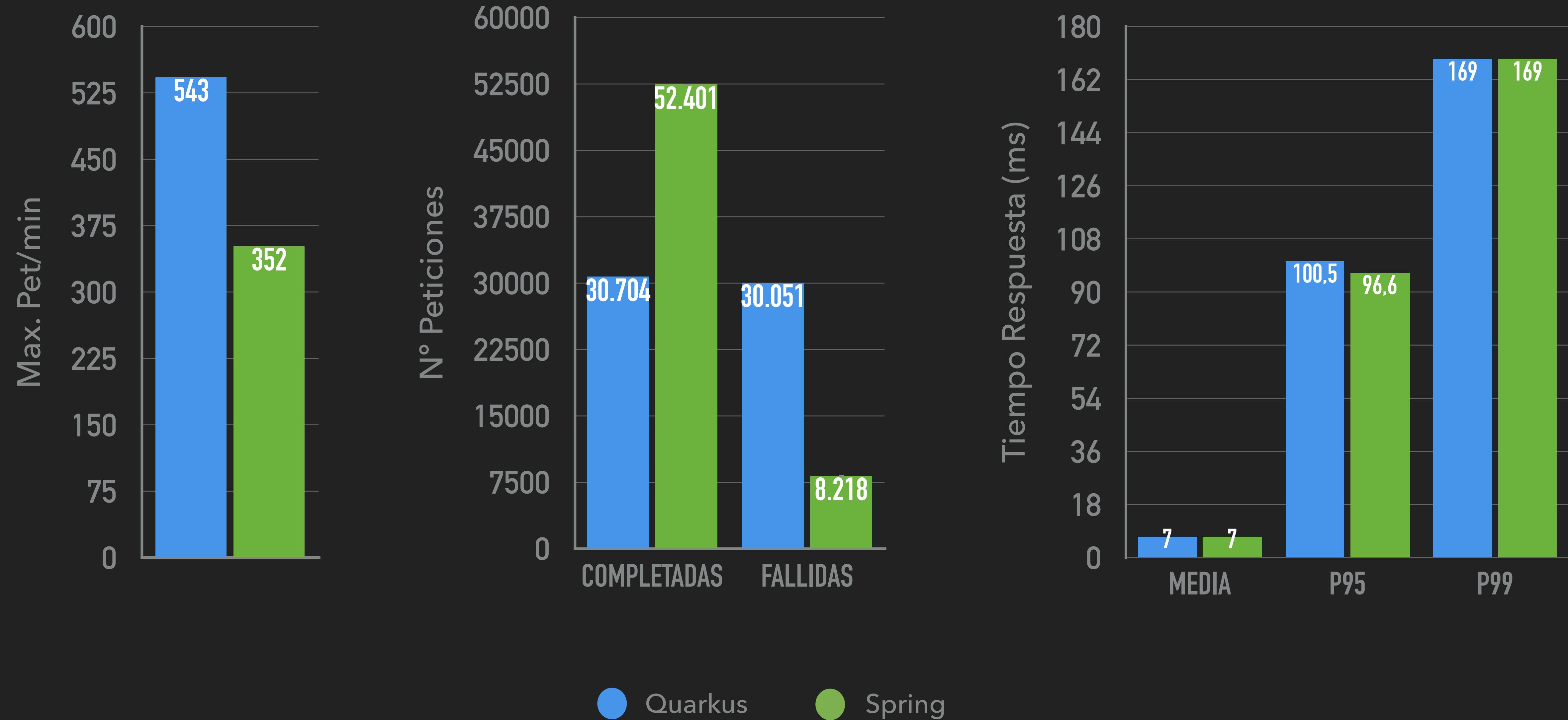


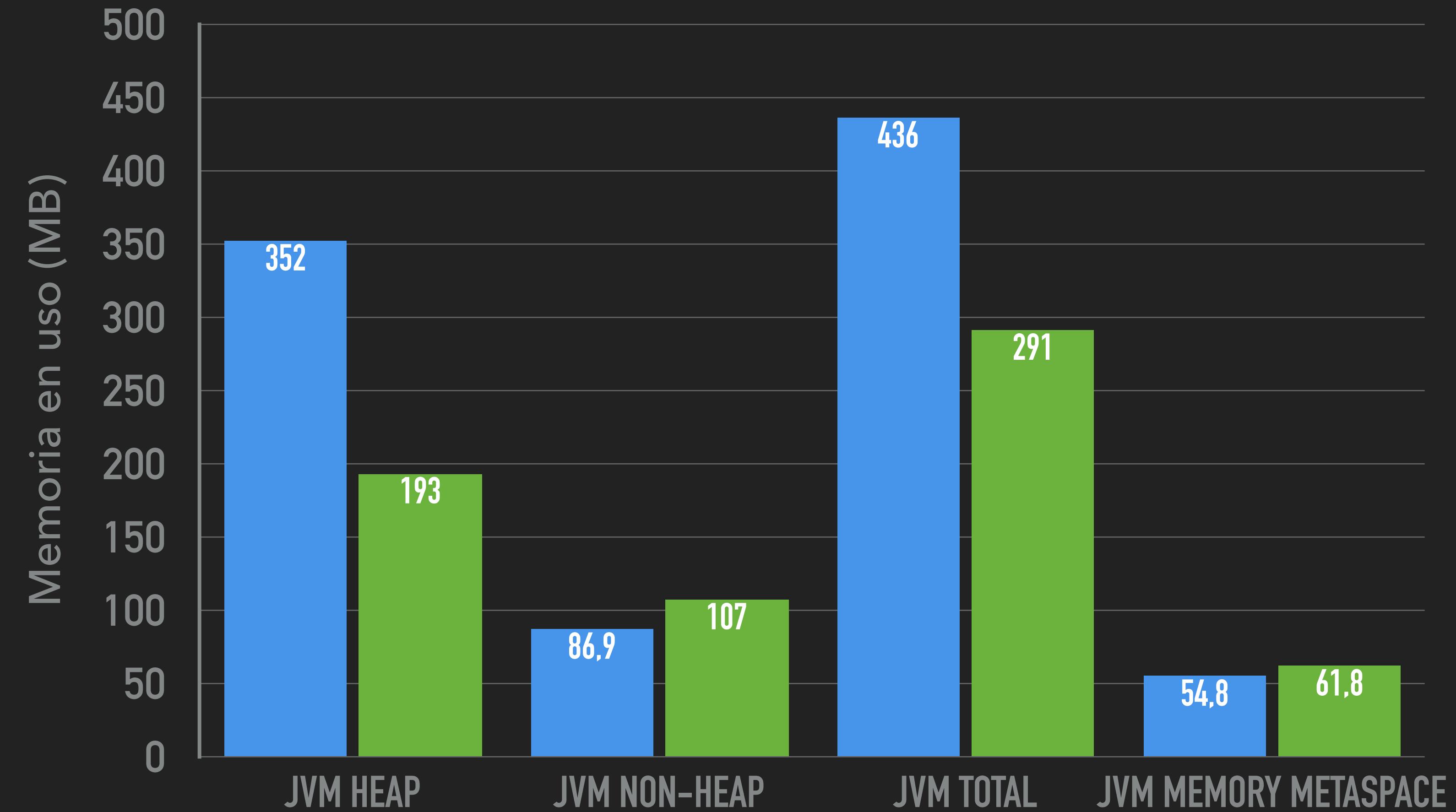
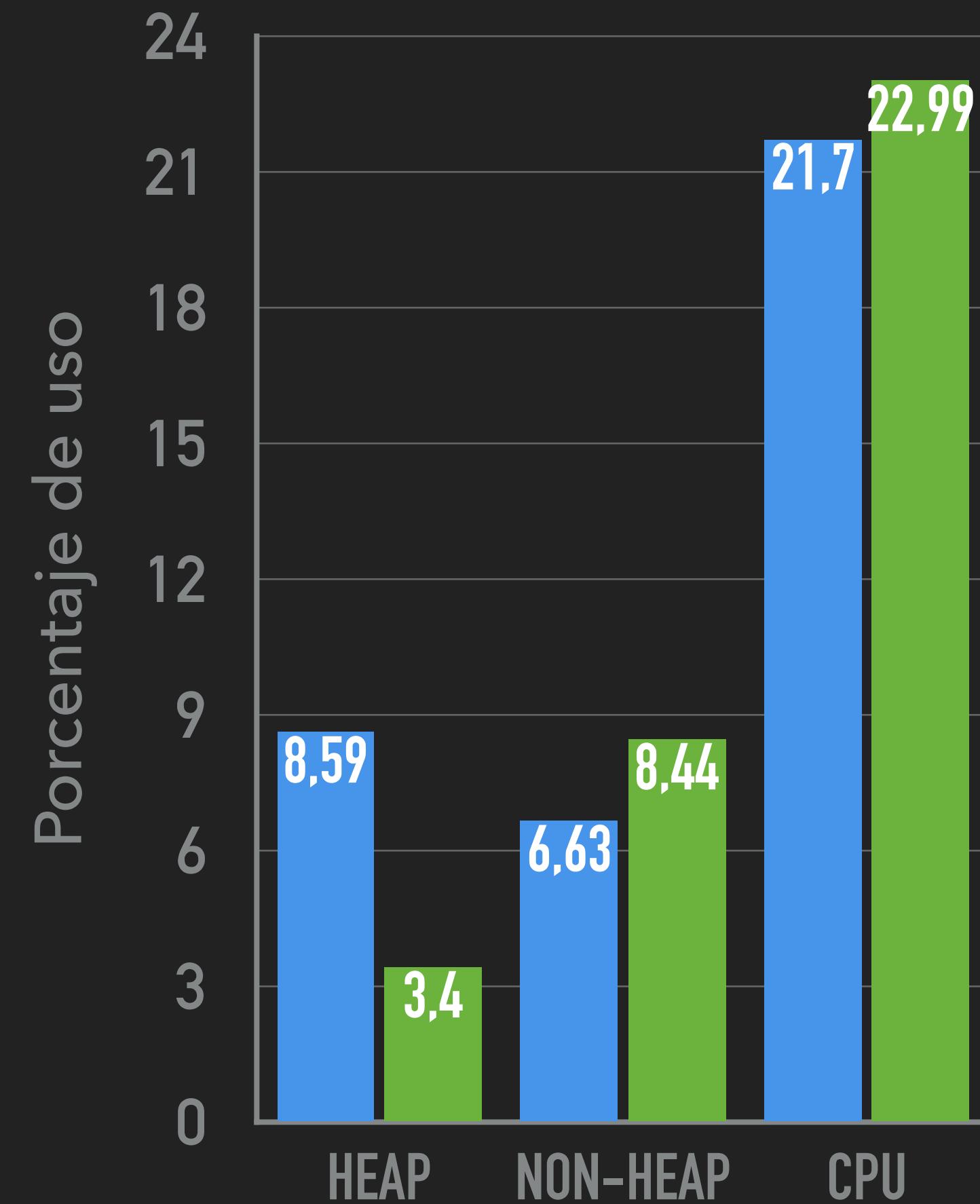
Quarkus

Spring

# RESULTADOS DE RENDIMIENTO

Reactivo





● Quarkus

● Spring



# CONCLUSIONES

## EXPERIENCIA PERSONAL

- ▶ Buen rendimiento
- ▶ Fácil aprendizaje
- ▶ Soluciones sencillas, limpias y diferentes a Spring
- ▶ Pese a su madurez las sensaciones están por encima de lo esperado
- ▶ Buena opción para aprender y llevar a proyectos productivos
- ▶ Muy rápido y sencillo crear imágenes nativas y manifiestos para Kubernetes



# TRABAJOS FUTUROS

## SIGUIENTES PASOS...

- ▶ Realizar mas pruebas de carga:
  - Sobre otros sistemas operativos y arquitecturas
  - Sobre el resto de tecnologías y analizar su comportamiento
  - Usando otras herramientas como k6 o wrk2
- ▶ Probar Quarkus en Kubernetes con imágenes JVM y nativas



# DUDAS O PREGUNTAS

# ¡MUCHAS GRACIAS!

AUTORES: IZAN SANTANA ALONSO Y DIEGO FERNANDEZ ACEVES  
TUTOR: MICAEL GALLEGO CARRILLO