Java startup time

where we are
and
what is coming

About me

- Java Developer at BVGroup
- Hobbies
  - Cats
  - Cycling

Agenda

- Common Definitions
- JIT - Just in Time Compiler
- AoT - Ahead of Time compilation
- Project CRaC
- Project Leyden
- Conclusion

Disclaimer

- Today we’re talking about many things
- Each topic can have its own talk
- My opinions are not biased

Definitions

- **Cold start** refers to the initial state of a system where the application is loaded for the first time
- **Startup** is the time it takes to get to the first useful unit of work
- **Warm-up** is the time it takes for the application to reach peak performance

Startup and warm-up are an issue for Java applications because Java is highly dynamic

JIT

just-in-time compiler

A bit about JIT

- C1 compiles the code into bytecode
- C2 analyses and optimizes the code
- C2 keeps checking the code and deoptimizes and optimizes again
- can be configured via parameters

Java from start to peak performance

- load JAR files from disk
- uncompress class files
- verify class definitions
- execute in the interpreter
- gather profiling feedback
- compile to machine code
- execute at peak performance

AoT

ahead of time compilation

compile Java code into native code

- instant startup
- no warmup
- low resource usage
- reduced attack surface
- compact packaging
- lower compute costs

AoT

ahead of time compilation

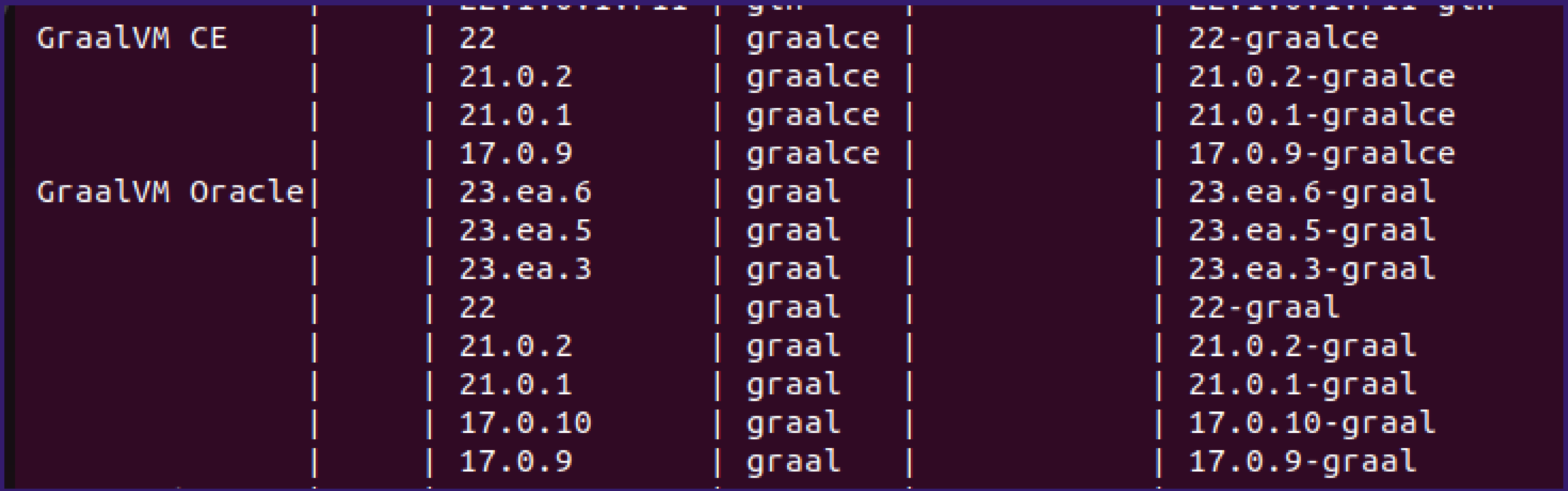
- load executable from disk
- execute at peak perfomance

AoT
ahead of time compilation

Disadvantages

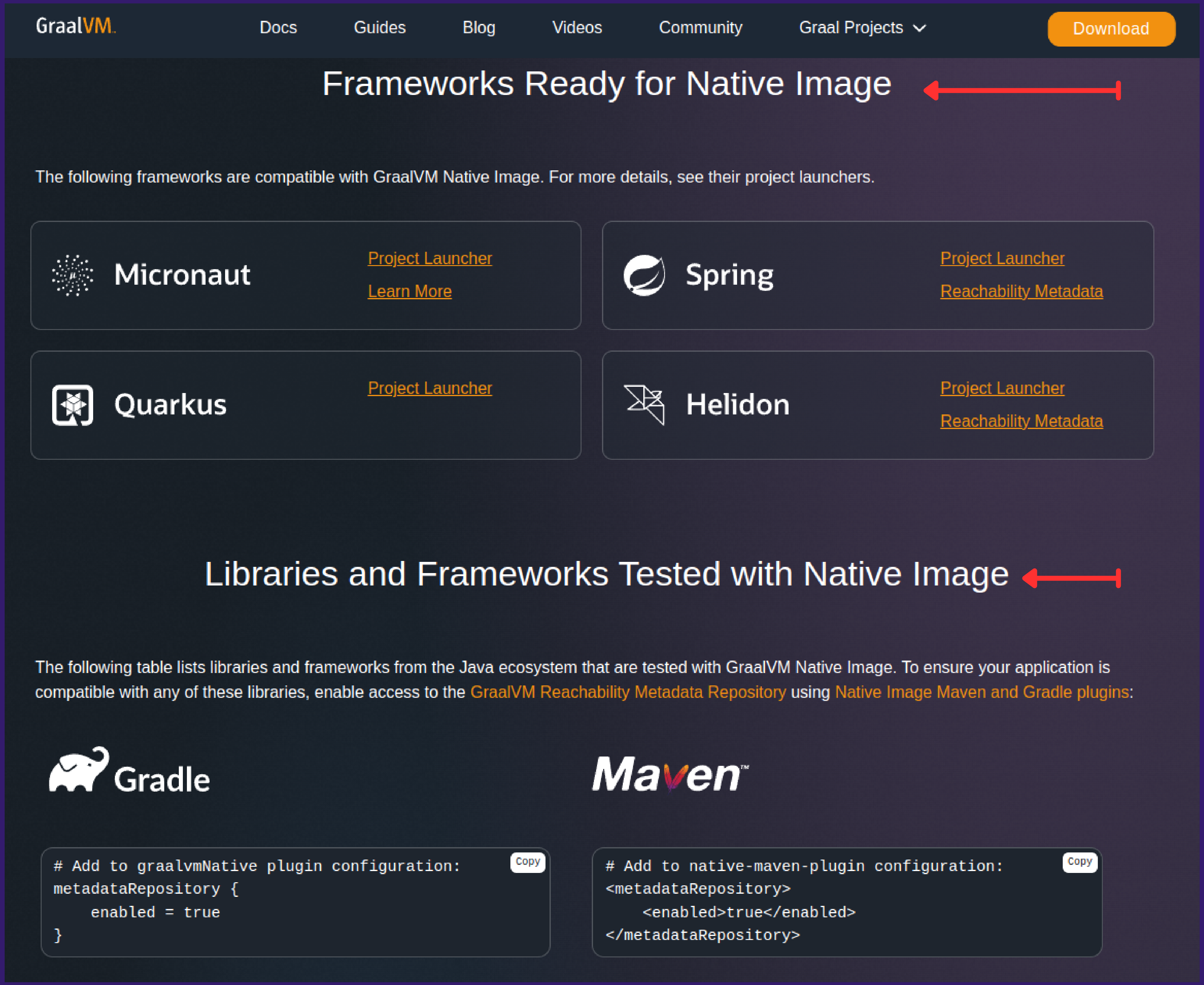
- extra configuration to detect reflection
- not compatible with all libraries
- adapt your pipeline
- longer build times

GraalVM



```
 GraalVM CE     |      | 22             | graalce |            | 22-graalce
                |      | 21.0.2         | graalce |            | 21.0.2-graalce
                |      | 21.0.1         | graalce |            | 21.0.1-graalce
                |      | 17.0.9         | graalce |            | 17.0.9-graalce
 GraalVM Oracle|      | 23.ea.6        | graal   |            | 23.ea.6-graal
                |      | 23.ea.5        | graal   |            | 23.ea.5-graal
                |      | 23.ea.3        | graal   |            | 23.ea.3-graal
                |      | 22             | graal   |            | 22-graal
                |      | 21.0.2         | graal   |            | 21.0.2-graal
                |      | 21.0.1         | graal   |            | 21.0.1-graal
                |      | 17.0.10        | graal   |            | 17.0.10-graal
                |      | 17.0.9         | graal   |            | 17.0.9-graal
```

GraalVM



Frameworks Ready for Native Image

The following frameworks are compatible with GraalVM Native Image. For more details, see their project launchers.

Micronaut	Project Launcher Learn More
Spring	Project Launcher Reachability Metadata
Quarkus	Project Launcher
Helidon	Project Launcher Reachability Metadata

Libraries and Frameworks Tested with Native Image

The following table lists libraries and frameworks from the Java ecosystem that are tested with GraalVM Native Image. To ensure your application is compatible with any of these libraries, enable access to the GraalVM Reachability Metadata Repository using Native Image Maven and Gradle plugins:

Gradle

```
# Add to graalvmNative plugin configuration:
metadataRepository {
    enabled = true
}
```

Maven

```
# Add to native-maven-plugin configuration:
<metadataRepository>
    <enabled>true</enabled>
</metadataRepository>
```

https://www.graalvm.org/native-image/libraries-and-frameworks/

Project CRaC
(Coordinated Restore at Checkpoint)

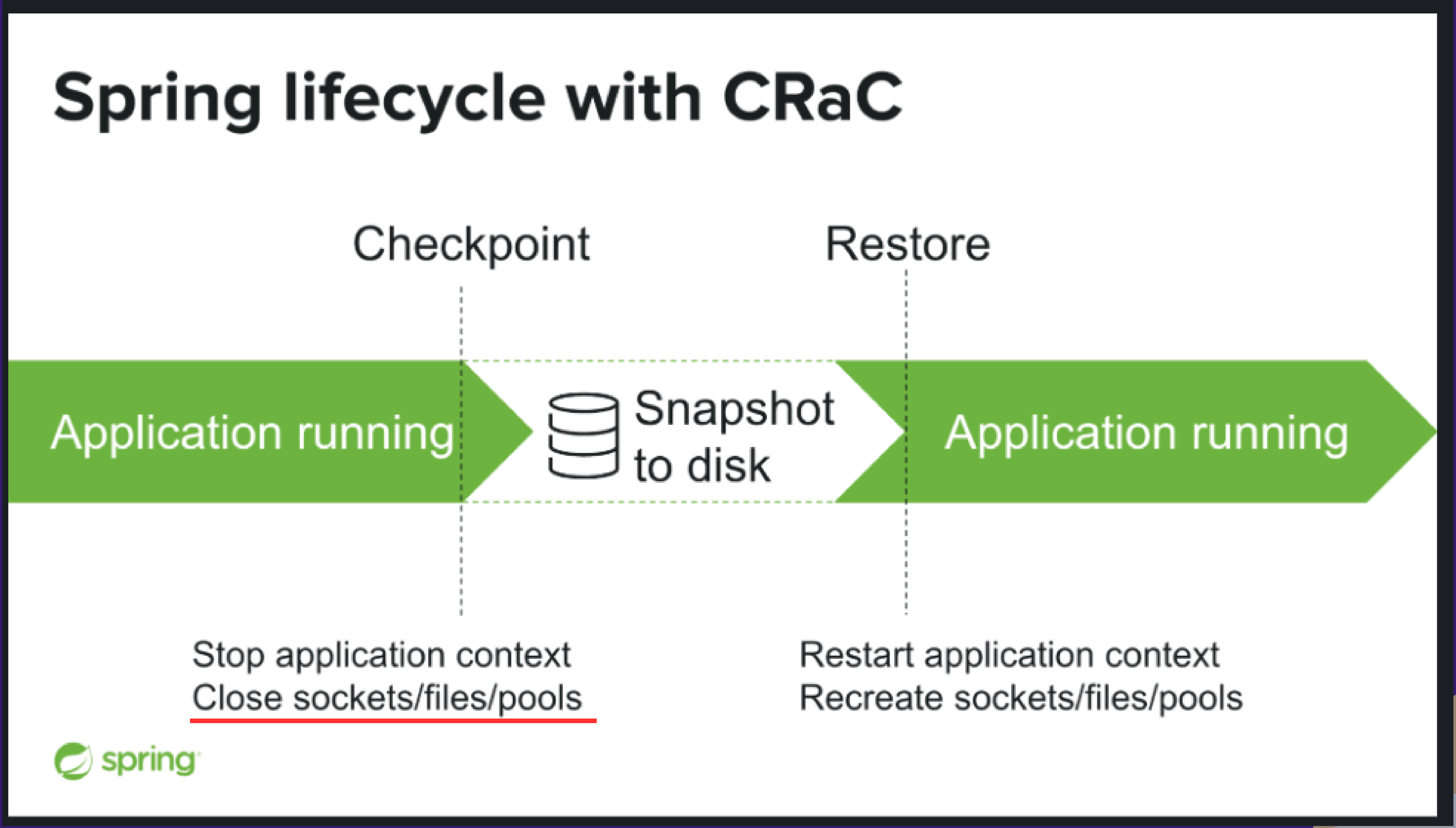
- initiative of Azul Technologies
- it is a project inside openJDK
- available only in Linux
    - based on CRIU
- spring boot from version 3.2 supports CRaC

Project CRaC

Checkpoint/Restore In Userspace, or CRIU (pronounced kree-oo), is a Linux software. It can freeze a running container (or an individual application) and checkpoint its state to disk. The data saved can be used to restore the application and run it exactly as it was during the time of the freeze. Using this functionality, application or container live migration, snapshots, remote debugging, and many other things are now possible.

https://criu.org

Project CRaC and Spring



Spring lifecycle with CRaC

Stop application context
Close sockets/files/pools

Restart application context
Recreate sockets/files/pools

spring

Project CRaC and Spring

From the documentation:

"When the -Dspring.context.checkpoint=onRefresh JVM system property is set, a checkpoint is created automatically at startup during the LifecycleProcessor.onRefresh phase. After this phase has completed, all non-lazy initialized singletons have been instantiated, and InitializingBean#afterPropertiesSet callbacks have been invoked; but the lifecycle has not started, and the ContextRefreshedEvent has not yet been published."

Project CRaC and Spring

creating an automatic checkpoint:

```
java
-Dspring.context.checkpoint=onRefresh
-XX:CRaCCheckpointTo=./tmp_checkpoint
-jar spring-petclinic-3.2.0.jar
```

starting from the checkpoint

```
java -XX:CRaCRestoreFrom=./tmp_checkpoint
```

Project CRaC

starting the application:

```
java
-XX:CRaCCheckpointTo=./tmp_checkpoint
-jar spring-petclinic-3.2.0.jar
```

creating an manual checkpoint:

```
jcmd spring-petclinic-3.2.0.jar JDK.checkpoint
```

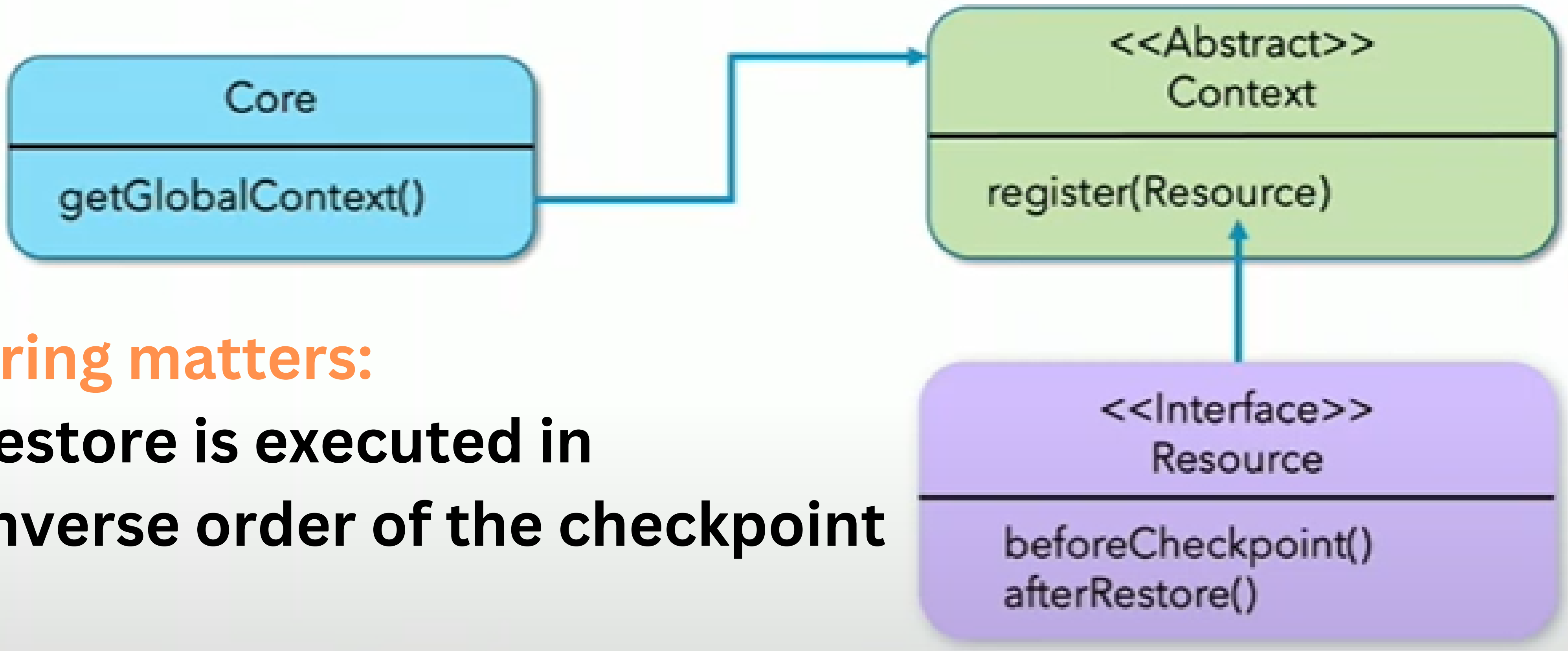
starting from the checkpoint

```
java -XX:CRaCRestoreFrom=./tmp_checkpoint
```

Project CRaC

Using CRaC API

- `Resource` objects need to be registered with a `Context` so that they can receive notifications
- There is a global `Context` accessible via the static `getGlobalContext()` method of the `Core` class

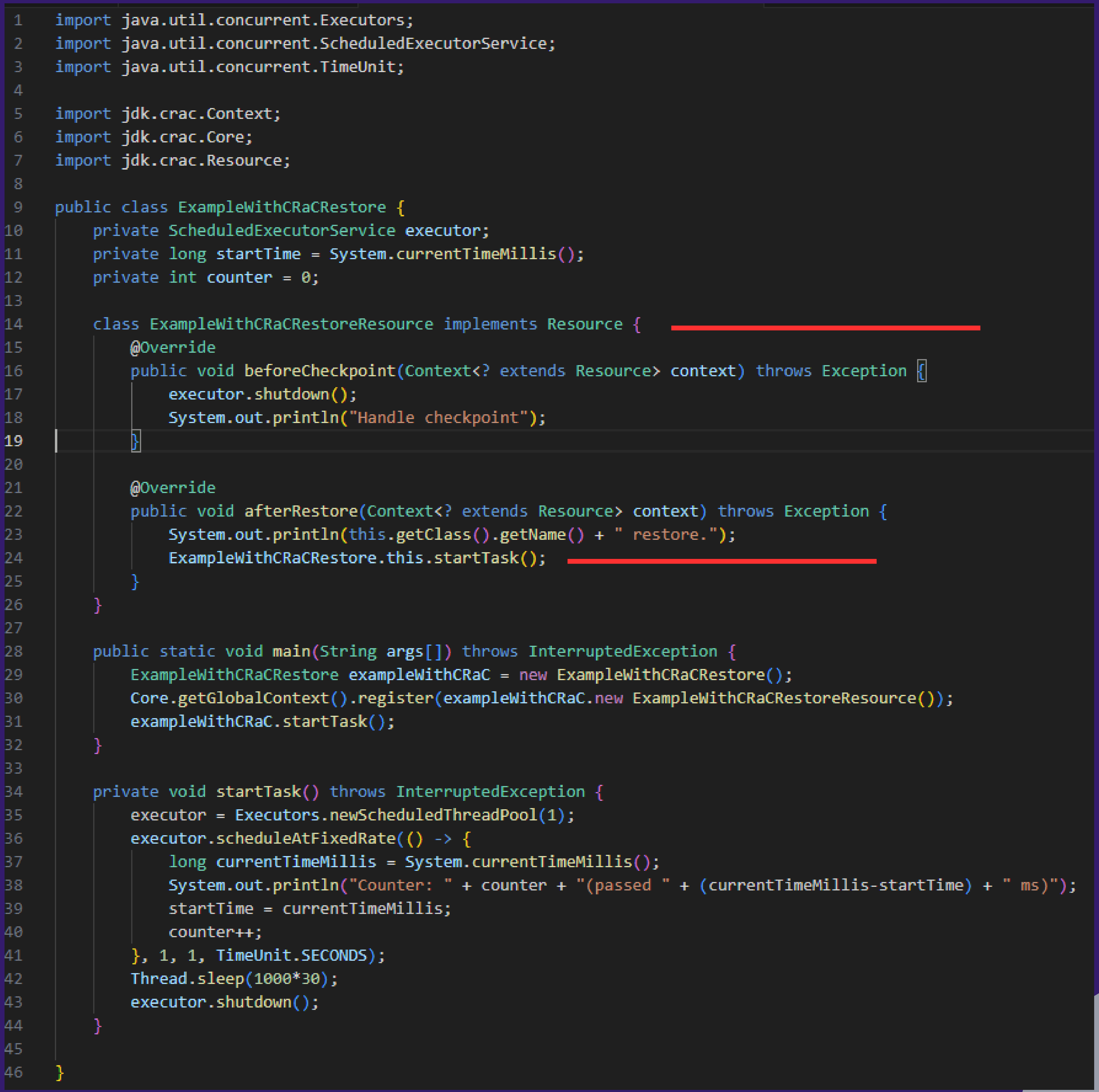


Ordering matters:

the restore is executed in the inverse order of the checkpoint

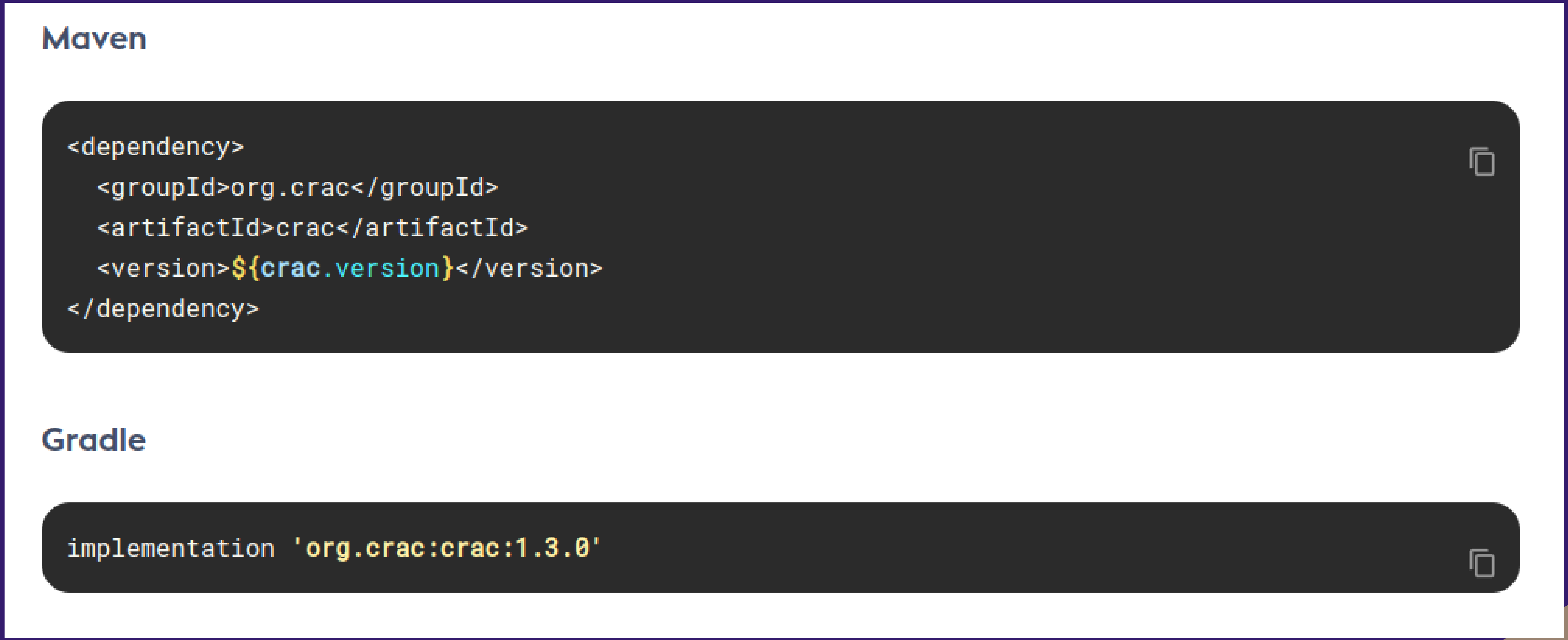
azul

Project CRaC



```java
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import jdk.crac.Context;
import jdk.crac.Core;
import jdk.crac.Resource;

public class ExampleWithCRaCRestore {
    private ScheduledExecutorService executor;
    private long startTime = System.currentTimeMillis();
    private int counter = 0;

    class ExampleWithCRaCRestoreResource implements Resource {
        @Override
        public void beforeCheckpoint(Context<? extends Resource> context) throws Exception {
            executor.shutdown();
            System.out.println("Handle checkpoint");
        }

        @Override
        public void afterRestore(Context<? extends Resource> context) throws Exception {
            System.out.println(this.getClass().getName() + " restore.");
            ExampleWithCRaCRestore.this.startTask();
        }
    }

    public static void main(String args[]) throws InterruptedException {
        ExampleWithCRaCRestore exampleWithCRaC = new ExampleWithCRaCRestore();
        Core.getGlobalContext().register(exampleWithCRaC.new ExampleWithCRaCRestoreResource());
        exampleWithCRaC.startTask();
    }

    private void startTask() throws InterruptedException {
        executor = Executors.newScheduledThreadPool(1);
        executor.scheduleAtFixedRate(() -> {
            long currentTimeMillis = System.currentTimeMillis();
            System.out.println("Counter: " + counter + "(passed " + (currentTimeMillis-startTime) + " ms)");
            startTime = currentTimeMillis;
            counter++;
        }, 1, 1, TimeUnit.SECONDS);
        Thread.sleep(1000*30);
        executor.shutdown();
    }

}
```

Project CRaC



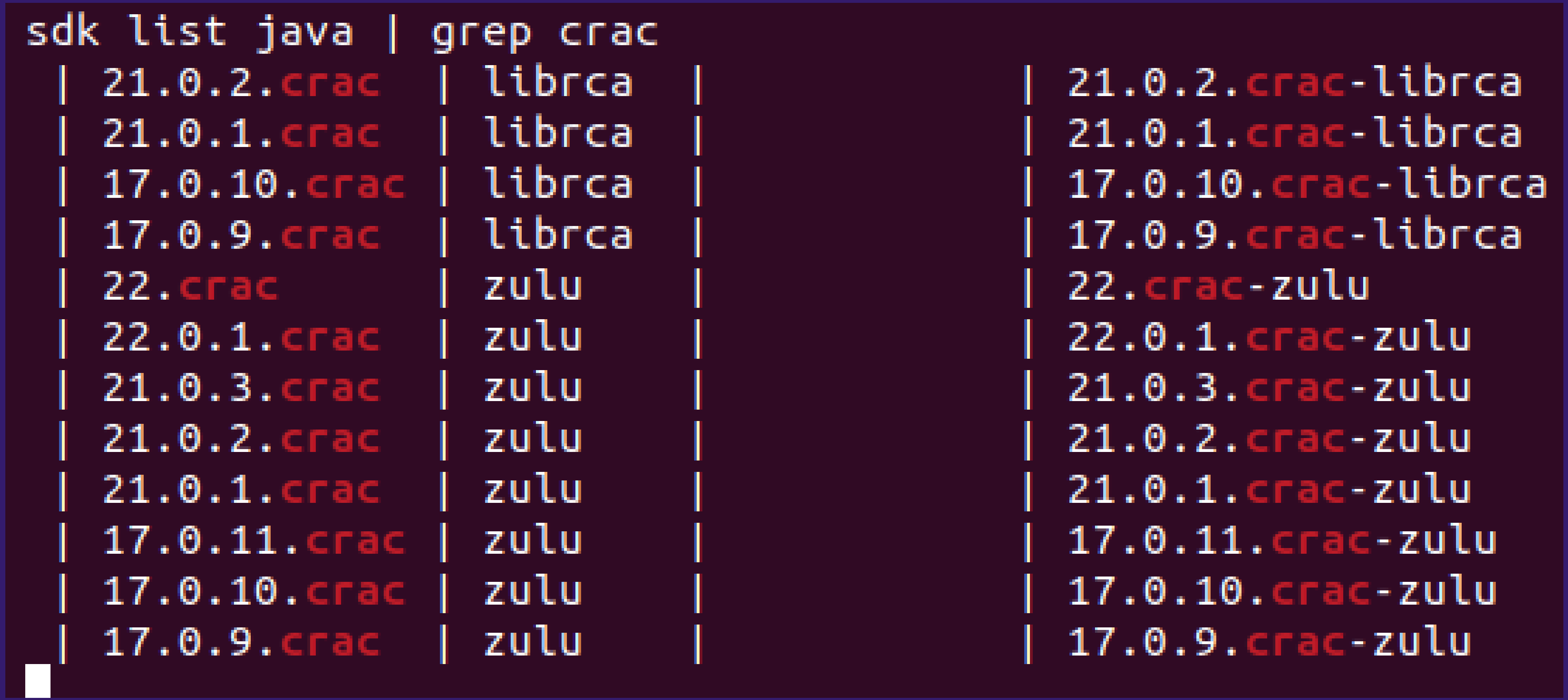
Maven

```xml
<dependency>
  <groupId>org.crac</groupId>
  <artifactId>crac</artifactId>
  <version>${crac.version}</version>
</dependency>
```

Gradle

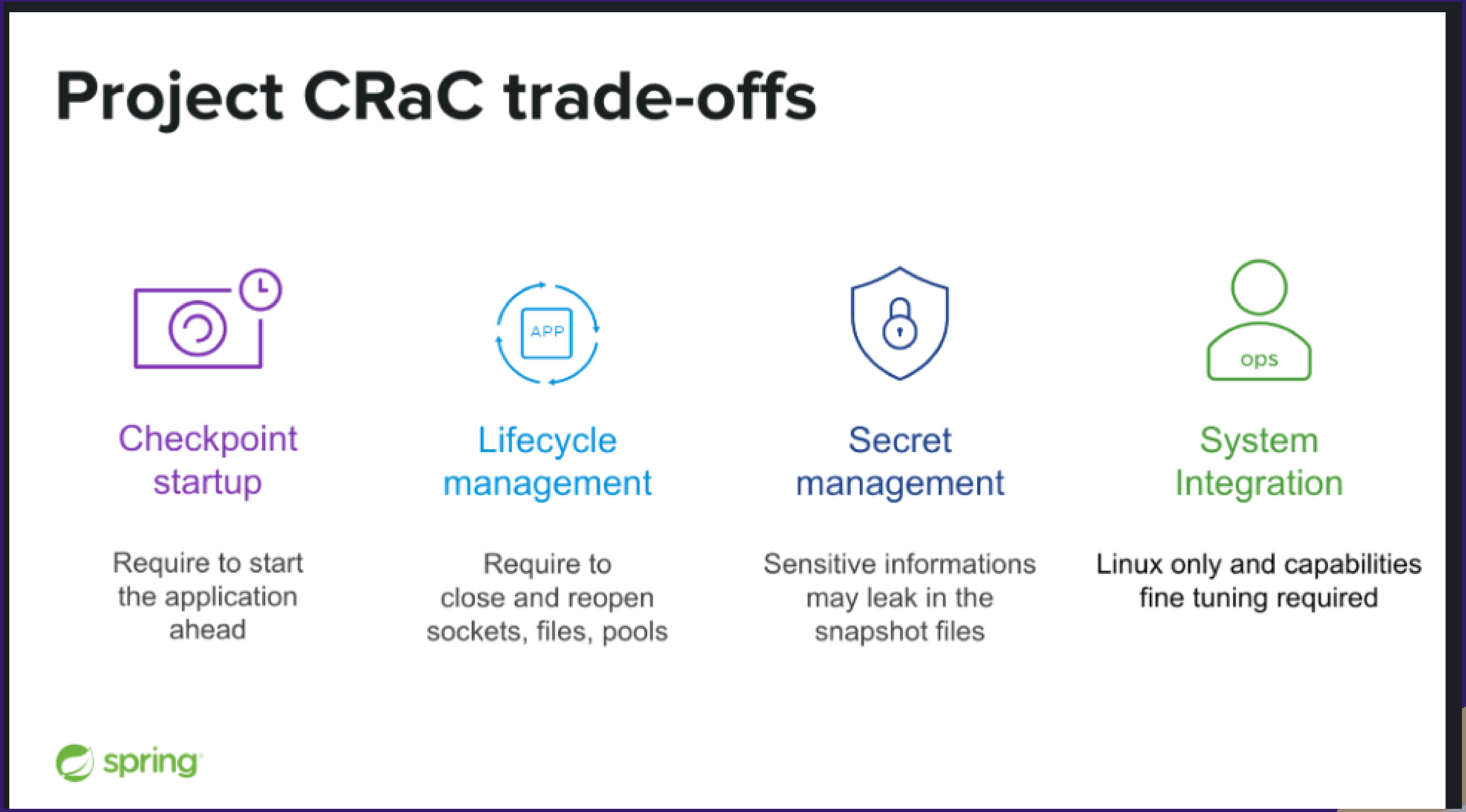
```groovy
implementation 'org.crac:crac:1.3.0'
```

Project CRaC



```
sdk list java | grep crac
 | 21.0.2.crac  | librca  |             | 21.0.2.crac-librca
 | 21.0.1.crac  | librca  |             | 21.0.1.crac-librca
 | 17.0.10.crac | librca  |             | 17.0.10.crac-librca
 | 17.0.9.crac  | librca  |             | 17.0.9.crac-librca
 | 22.crac      | zulu    |             | 22.crac-zulu
 | 22.0.1.crac  | zulu    |             | 22.0.1.crac-zulu
 | 21.0.3.crac  | zulu    |             | 21.0.3.crac-zulu
 | 21.0.2.crac  | zulu    |             | 21.0.2.crac-zulu
 | 21.0.1.crac  | zulu    |             | 21.0.1.crac-zulu
 | 17.0.11.crac | zulu    |             | 17.0.11.crac-zulu
 | 17.0.10.crac | zulu    |             | 17.0.10.crac-zulu
 | 17.0.9.crac  | zulu    |             | 17.0.9.crac-zulu
```

Project CRaC and Spring



Project CRaC trade-offs

Checkpoint startup

Require to start the application ahead

Lifecycle management

Require to close and reopen sockets, files, pools

Secret management

Sensitive informations may leak in the snapshot files

System Integration

Linux only and capabilities fine tuning required

spring

Project CRaC and AWS Lambda

```
...
  import org.crac.Resource;
  import org.crac.Core;
  ...
public class CRaCDemo implements RequestStreamHandler, Resource {
    public CRaCDemo() {
      Core.getGlobalContext().register(this);
    }
    public String handleRequest(String name, Context context) throws IOException {
      System.out.println("Handler execution");
      return "Hello " + name;
    }
    @Override
    public void beforeCheckpoint(org.crac.Context<? extends Resource> context)
        throws Exception {
      System.out.println("Before checkpoint");
    }
    @Override
    public void afterRestore(org.crac.Context<? extends Resource> context)
        throws Exception {
      System.out.println("After restore");
```

Project Leyden

- developed by Java core team of Oracle
- not officially available

From documentation:

A condenser is a program transformer that runs in a phase between compile time and run time. It transforms a program into a new, faster, and potentially smaller program while preserving the meaning given to the original program by the Java Platform Specification

Conclusion

current solutions are cumbersome and need to follow an easy path to work

but,

don’t give up on Java if startup time is any issue for your project

Thank you

https://sferrazjr.github.io/startup-time-talk