

What the CRaC...

SUPERFAST JVM STARTUP

ABOUTME.



Gerrit Grunwald | Developer Advocate | Azul | X@hansolo_

JAVAIS

GREAT

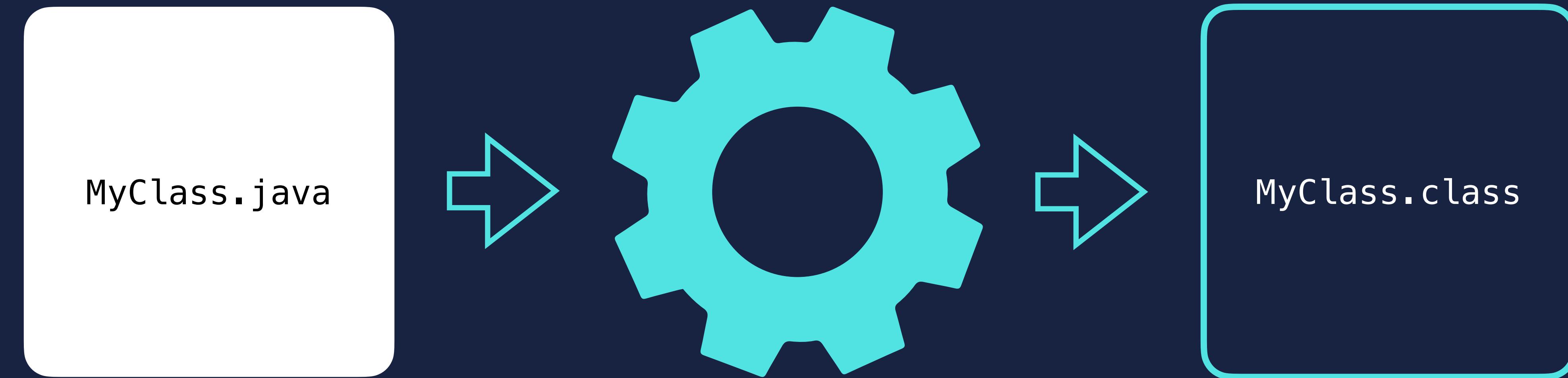
VIBRANT
COMMUNITY. . .

HUNDREDS OF
JUGS....

THOUSANDS OF
FOSS PROJECTS...

JAVA VIRTUAL
MACHINE

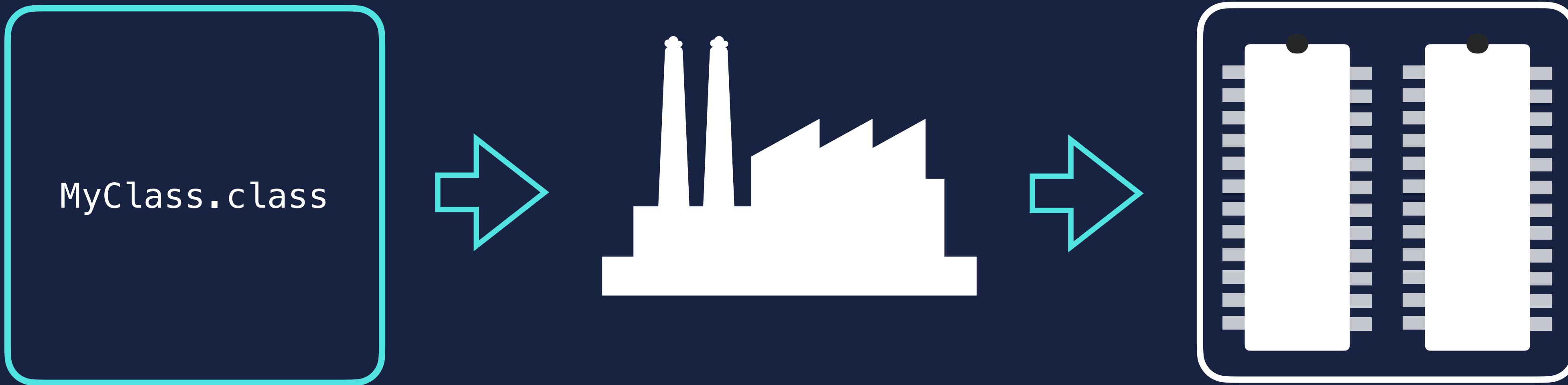
HOW DOES
IT WORK. . .



SOURCE CODE

COMPILER

BYTE CODE



BYTE CODE

CLASS LOADER

JVM MEMORY



JVM MEMORY

EXECUTION ENGINE

EXECUTION ENGINE



Interpreter



C1 JIT
Compiler
(client)



C2 JIT
Compiler
(server)

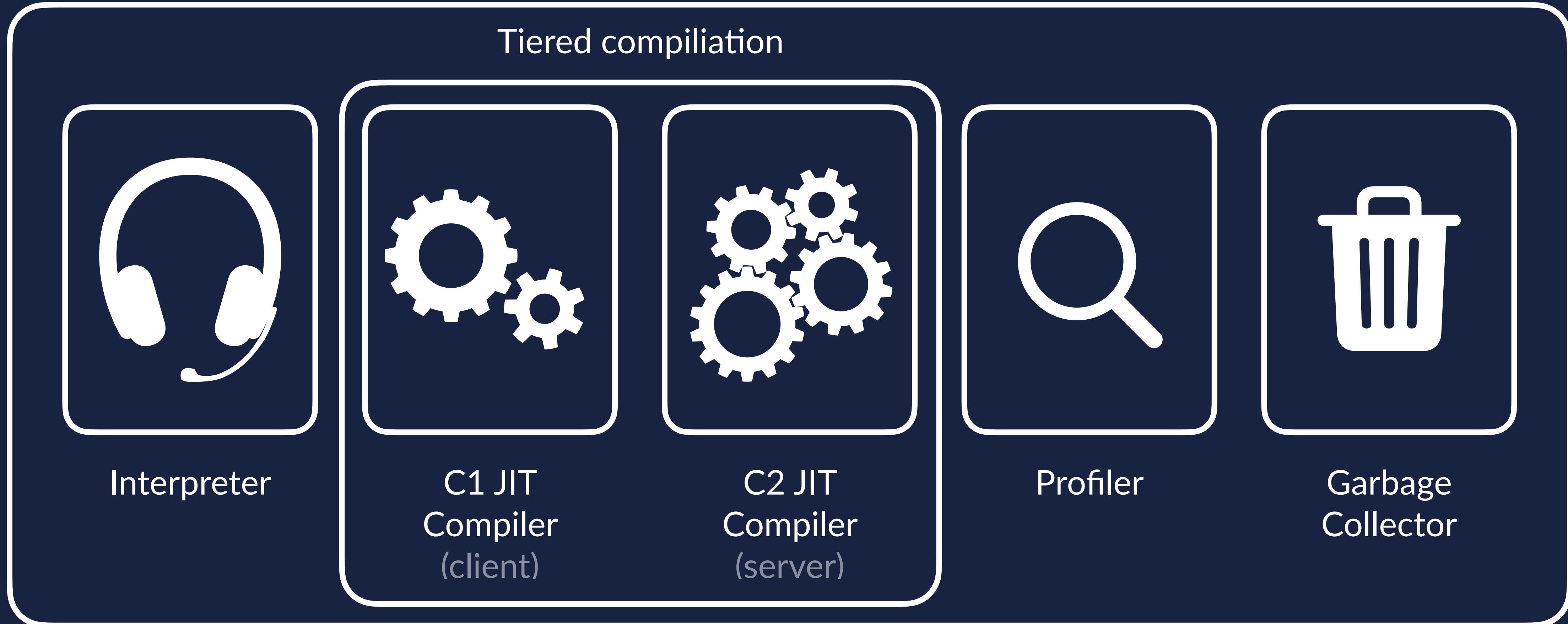


Profiler



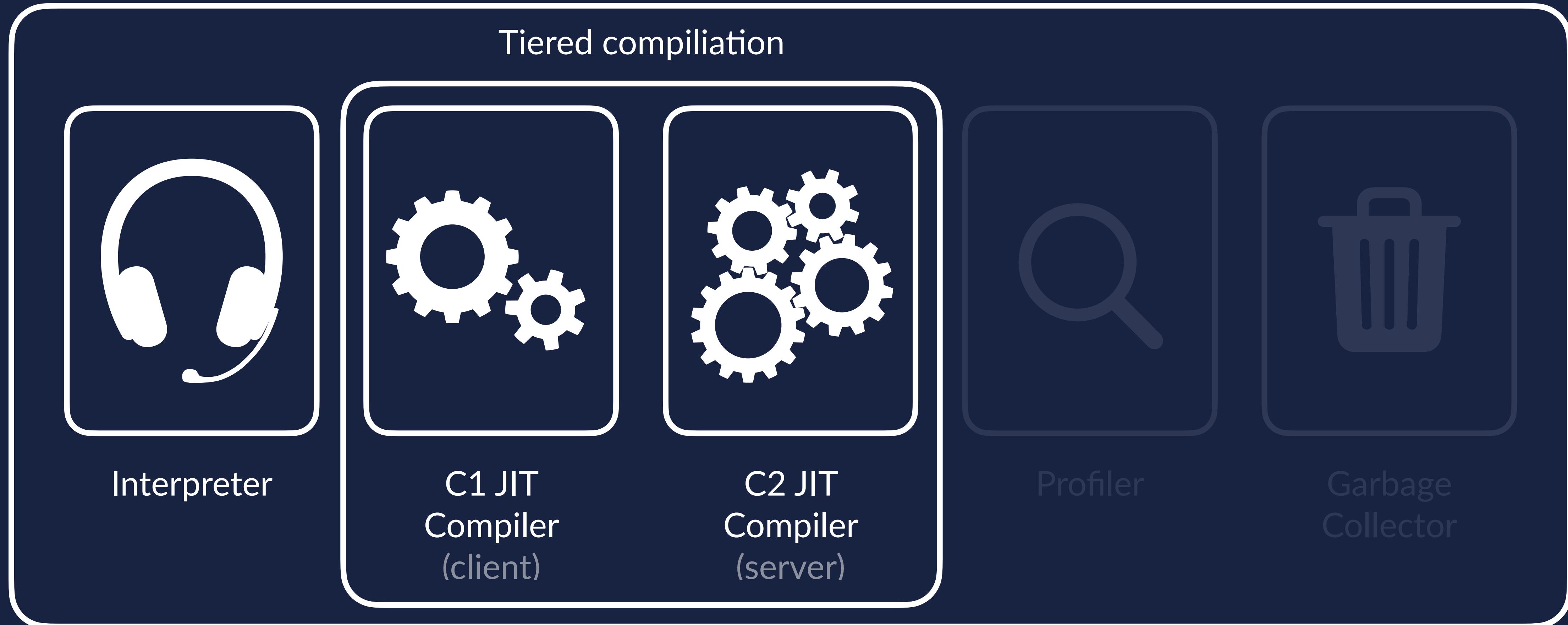
Garbage
Collector

EXECUTION ENGINE



DEFAULT SINCE JDK 8

EXECUTION ENGINE



DEFAULT SINCE JDK 8

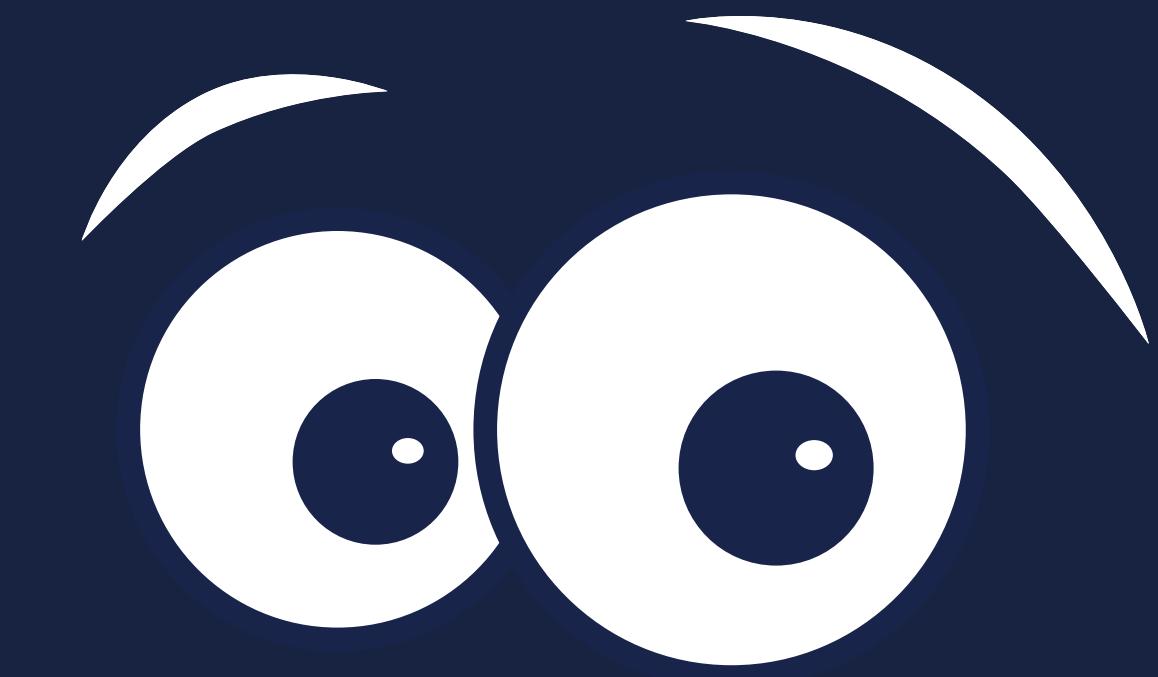
Converts ByteCode into
instruction set of CPU



INTERPRETER

THRESHOLD
REACHED
(1000 in JDK 17)

Detects hot spots by
counting method calls and
loop back edges



JVM

Pass the "hot" code
to C1 JIT Compiler



JVM

Compiles code as quickly
as possible with low optimisation



C1 JIT
COMPILER

Compiles code as quickly
as possible with low optimisation



C1 JIT
COMPILER

THRESHOLD
REACHED
(5000 in JDK 17)

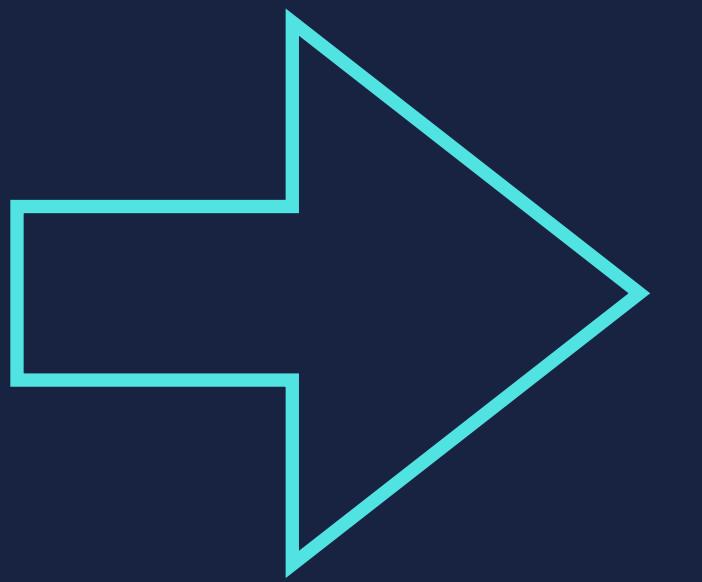


JVM

Pass the "hot" code
to C2 JIT Compiler



JVM



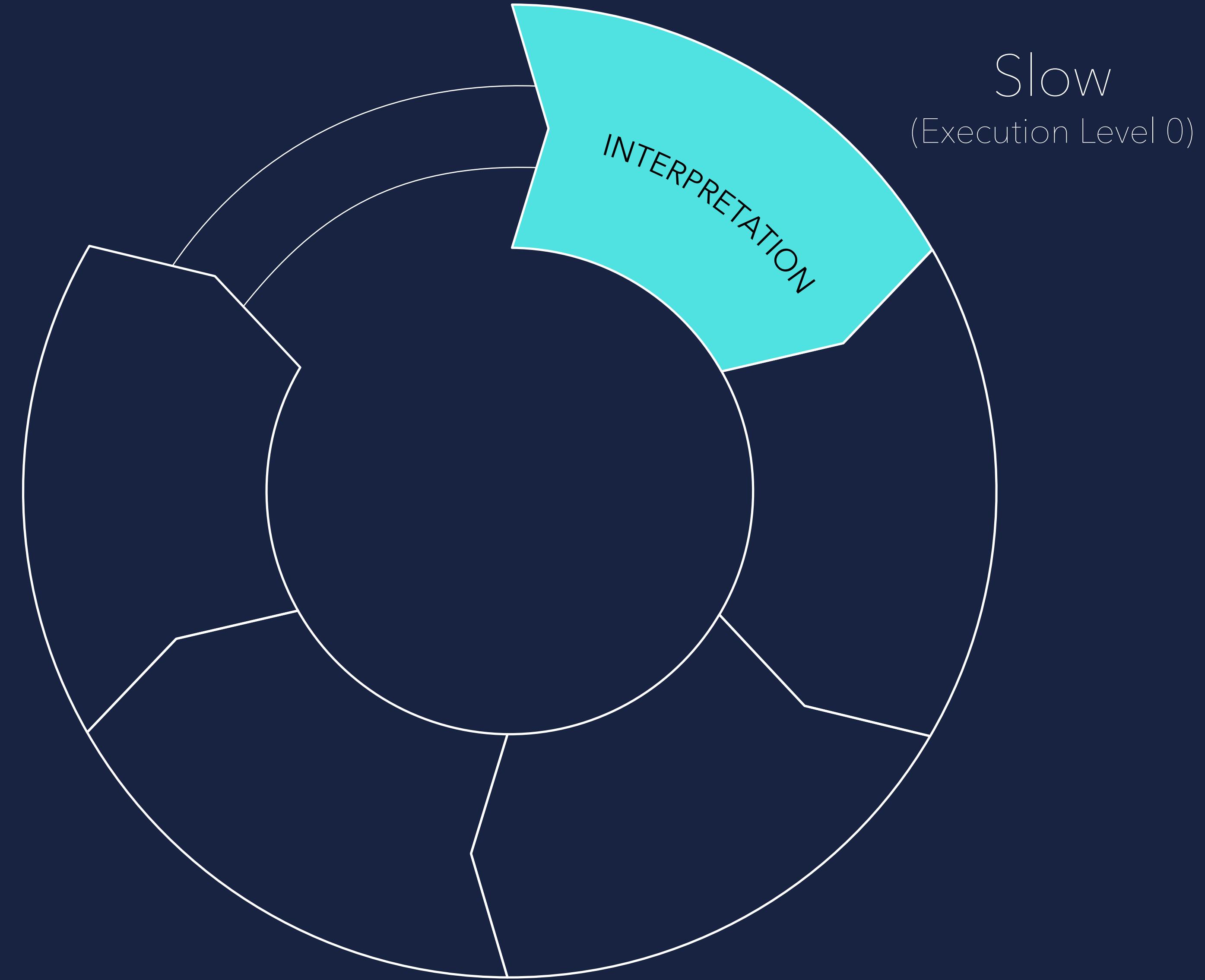
Compiles code with best
optimisation possible (slower)



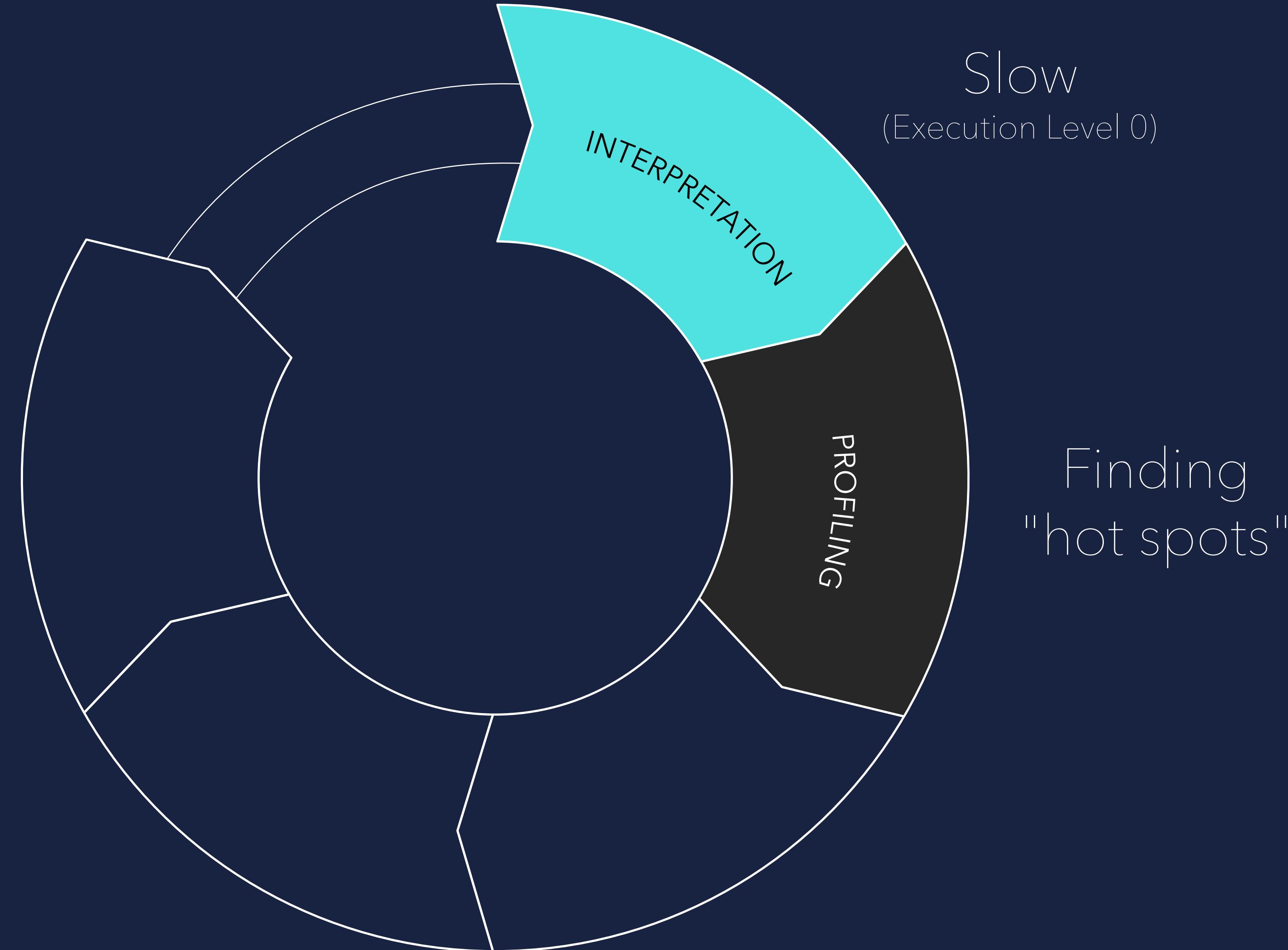
C2 JIT
COMPILER

EXECUTION
CYCLE

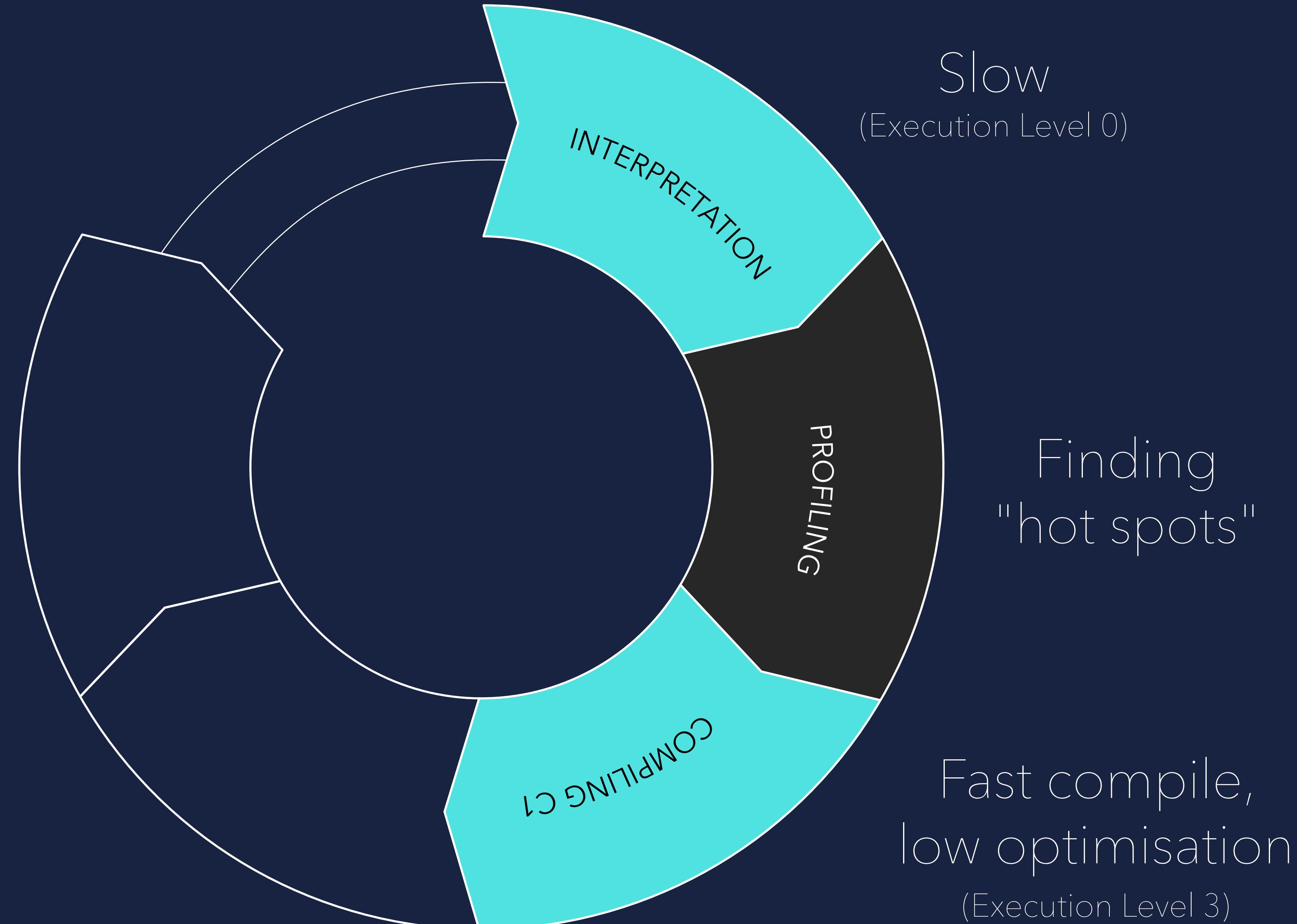
EXECUTION CYCLE



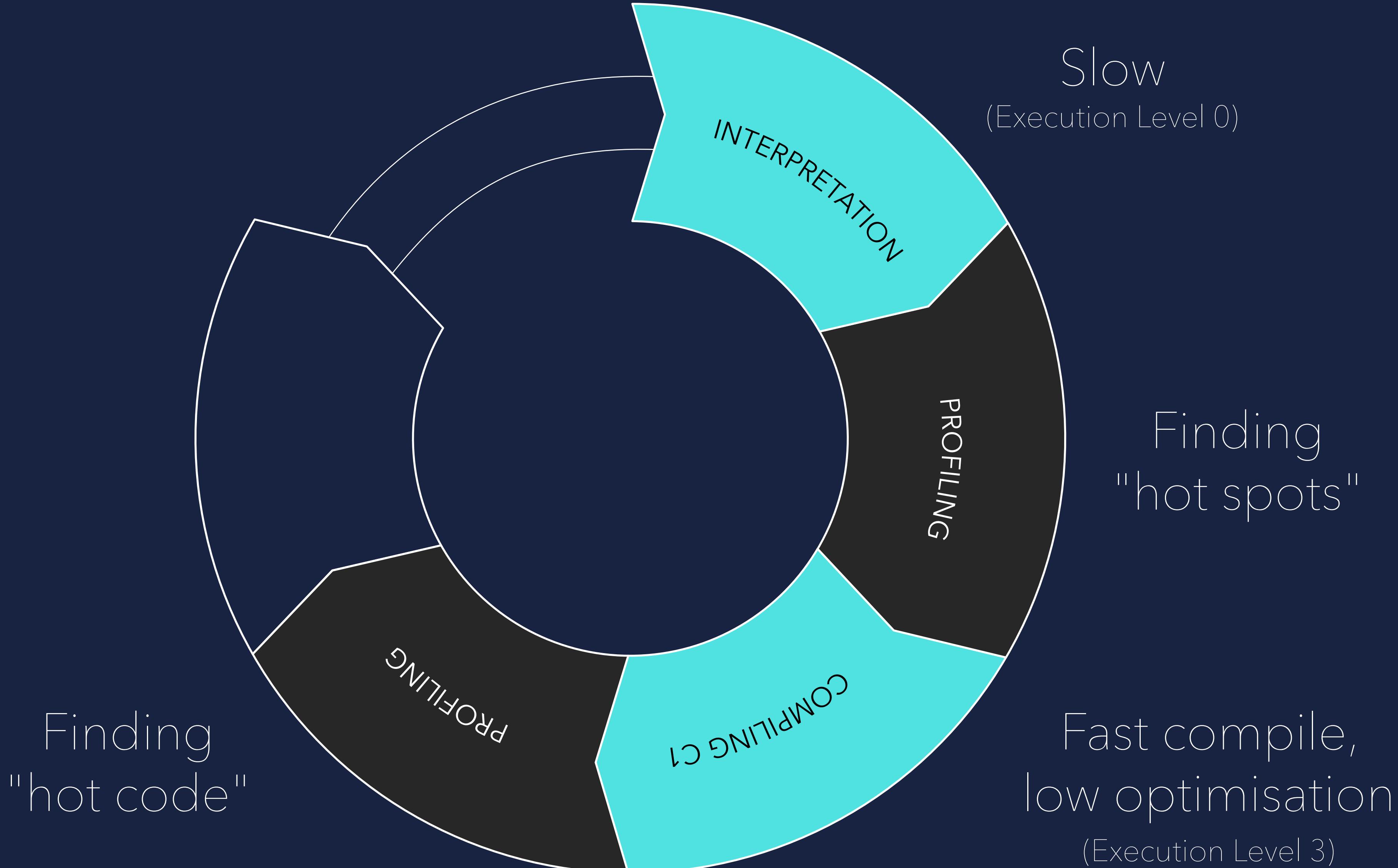
EXECUTION CYCLE



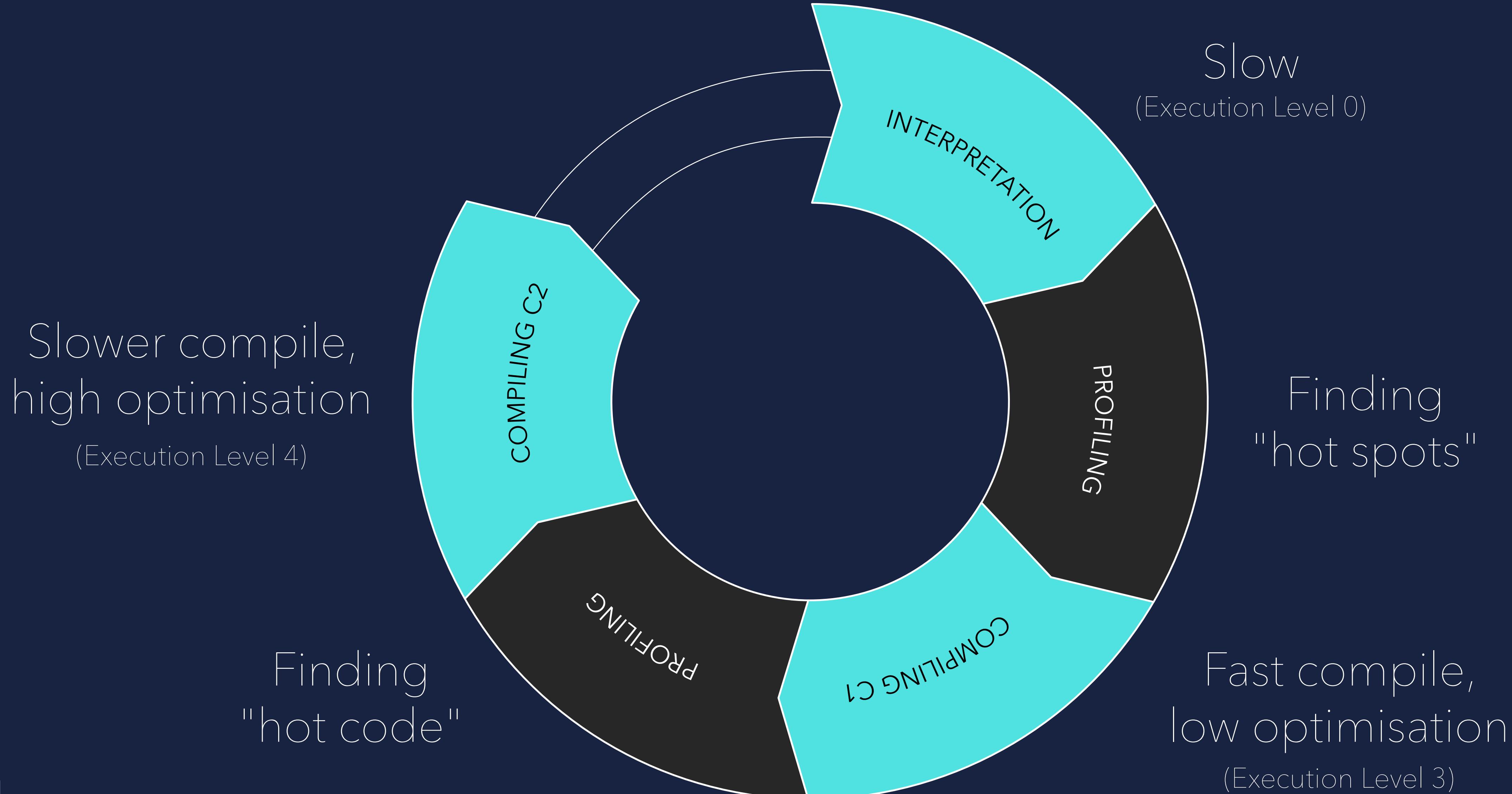
EXECUTION CYCLE



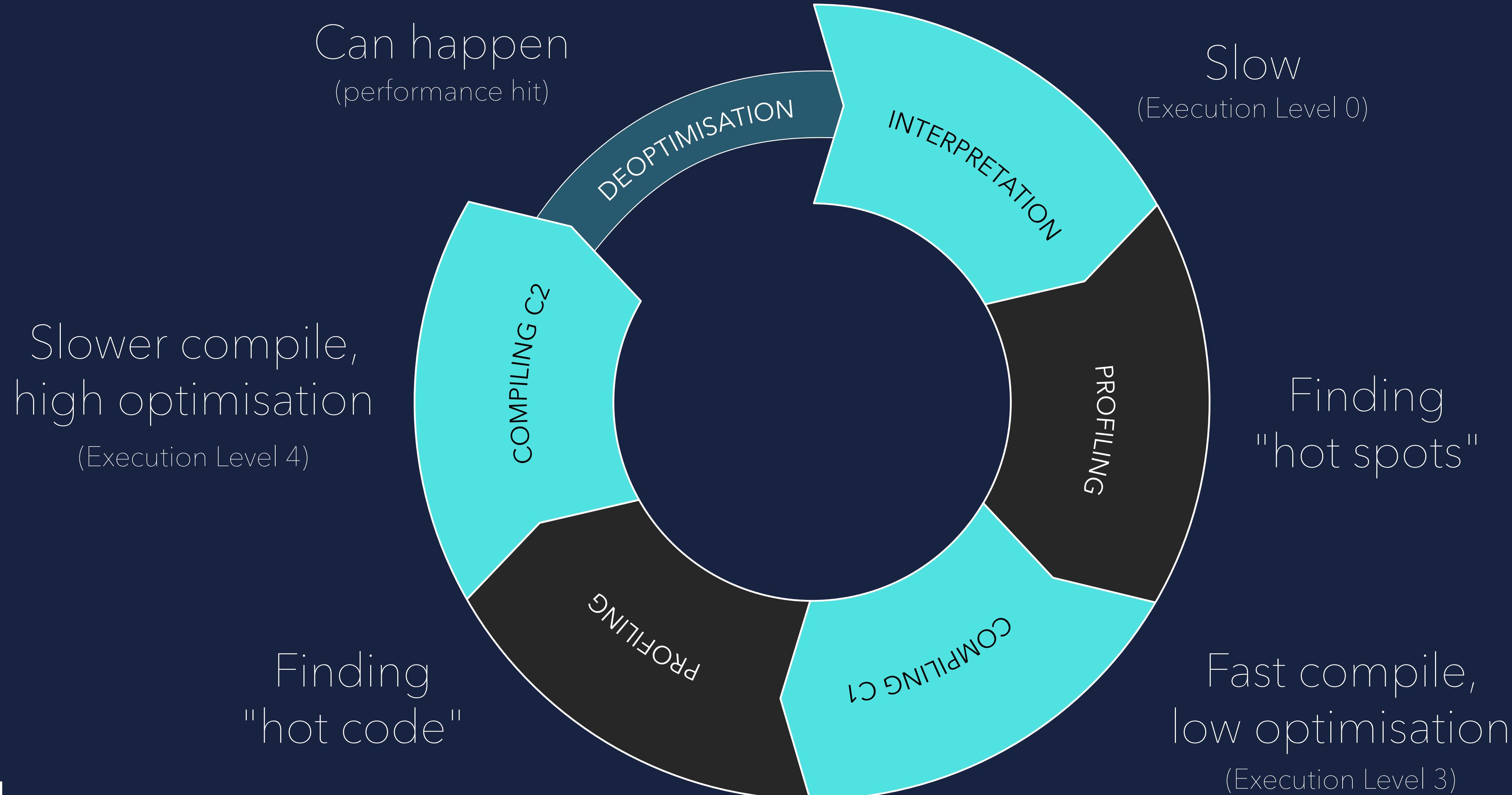
EXECUTION CYCLE



EXECUTION CYCLE



EXECUTION CYCLE

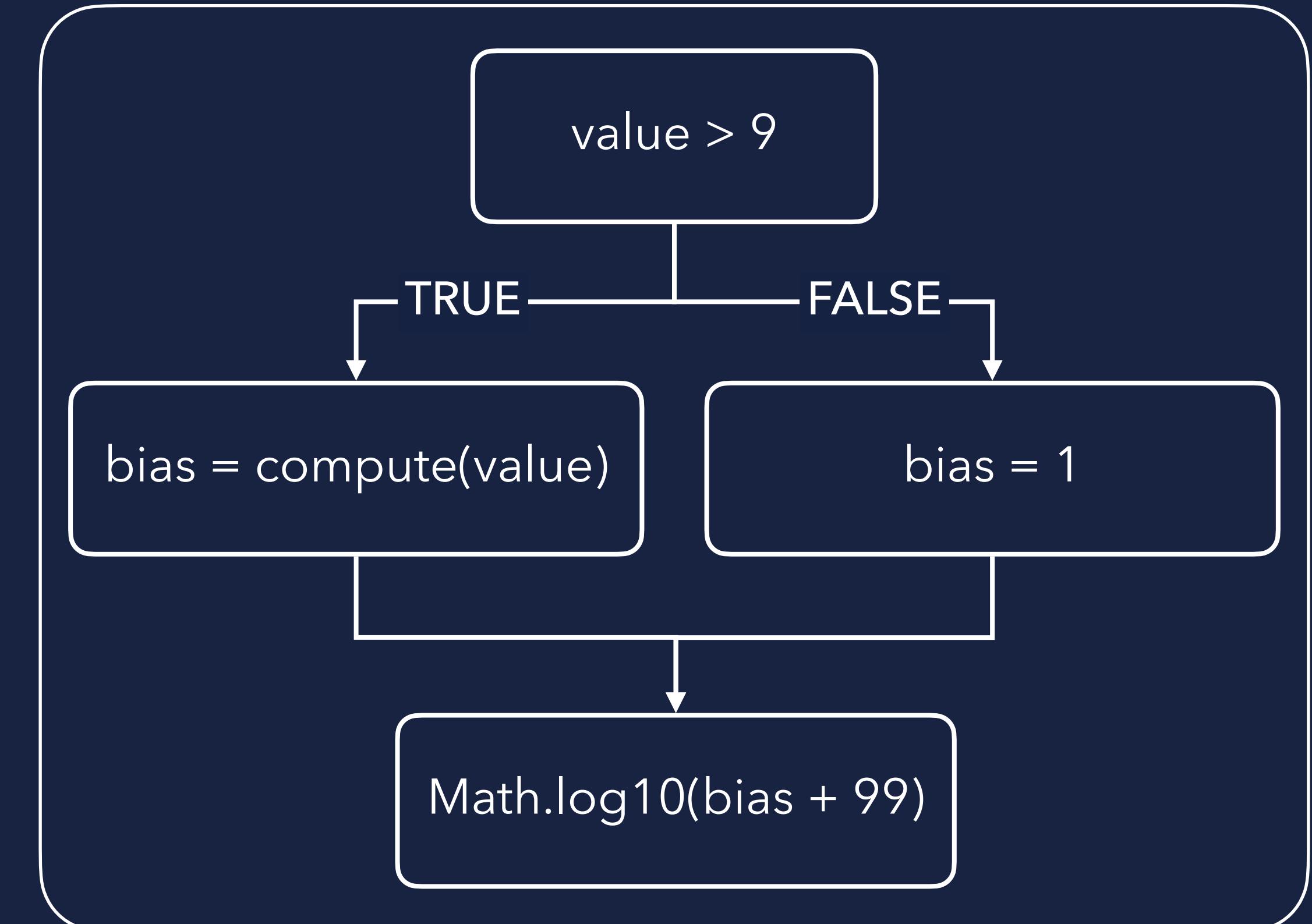


DEOPTIMISATION

DEOPTIMISATION

e.g. BRANCH ANALYSIS

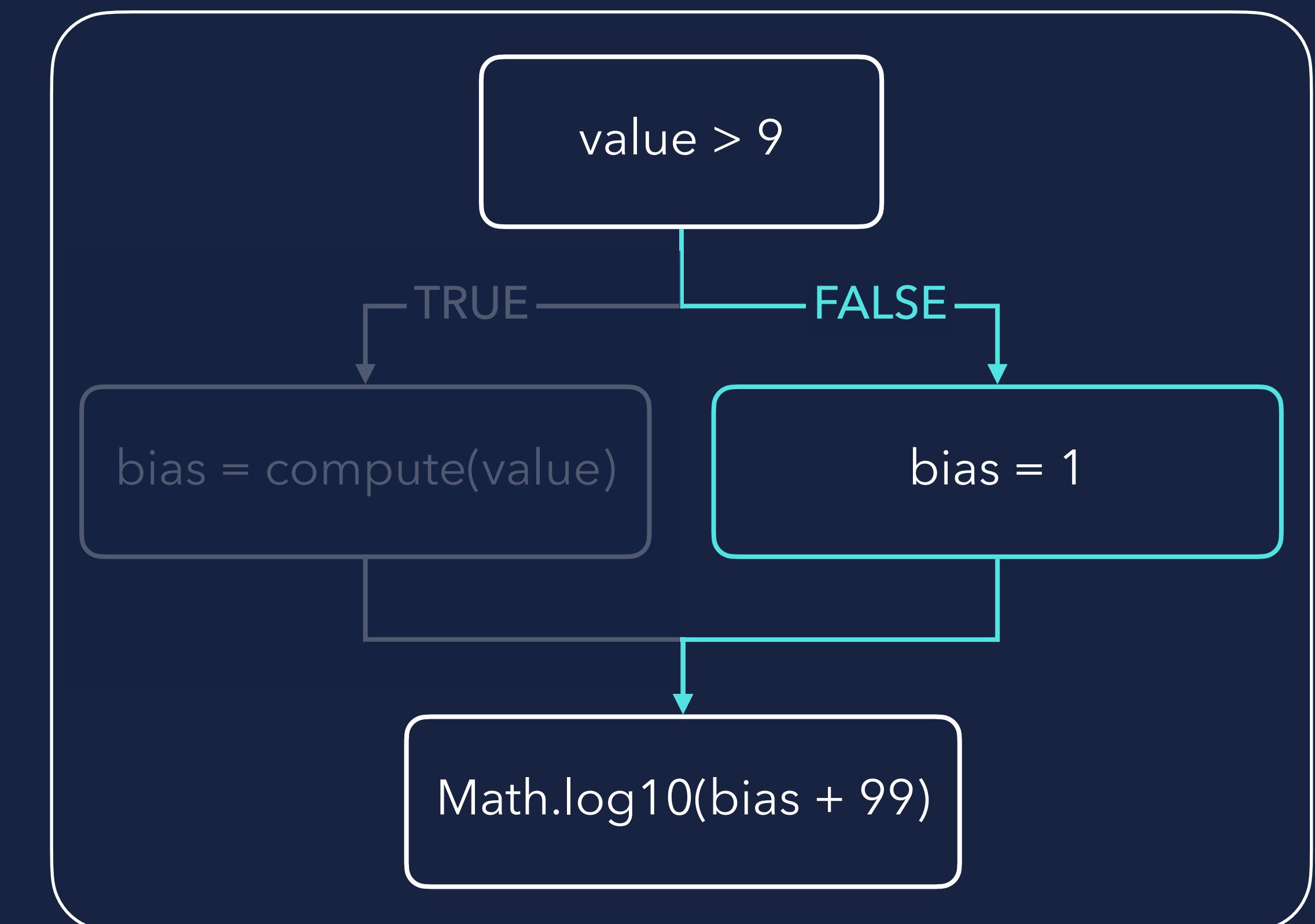
```
int computeMagnitude(int value) {  
    int bias;  
    if (value > 9) {  
        bias = compute(value);  
    } else {  
        bias = 1;  
    }  
    return Math.log10(bias + 99);  
}
```



DEOPTIMISATION

e.g. BRANCH ANALYSIS

```
int computeMagnitude(int value) {  
    int bias;  
    if (value > 9) {  
        bias = compute(value);  
    } else {  
        bias = 1;  
    }  
    return Math.log10(bias + 99);  
}
```

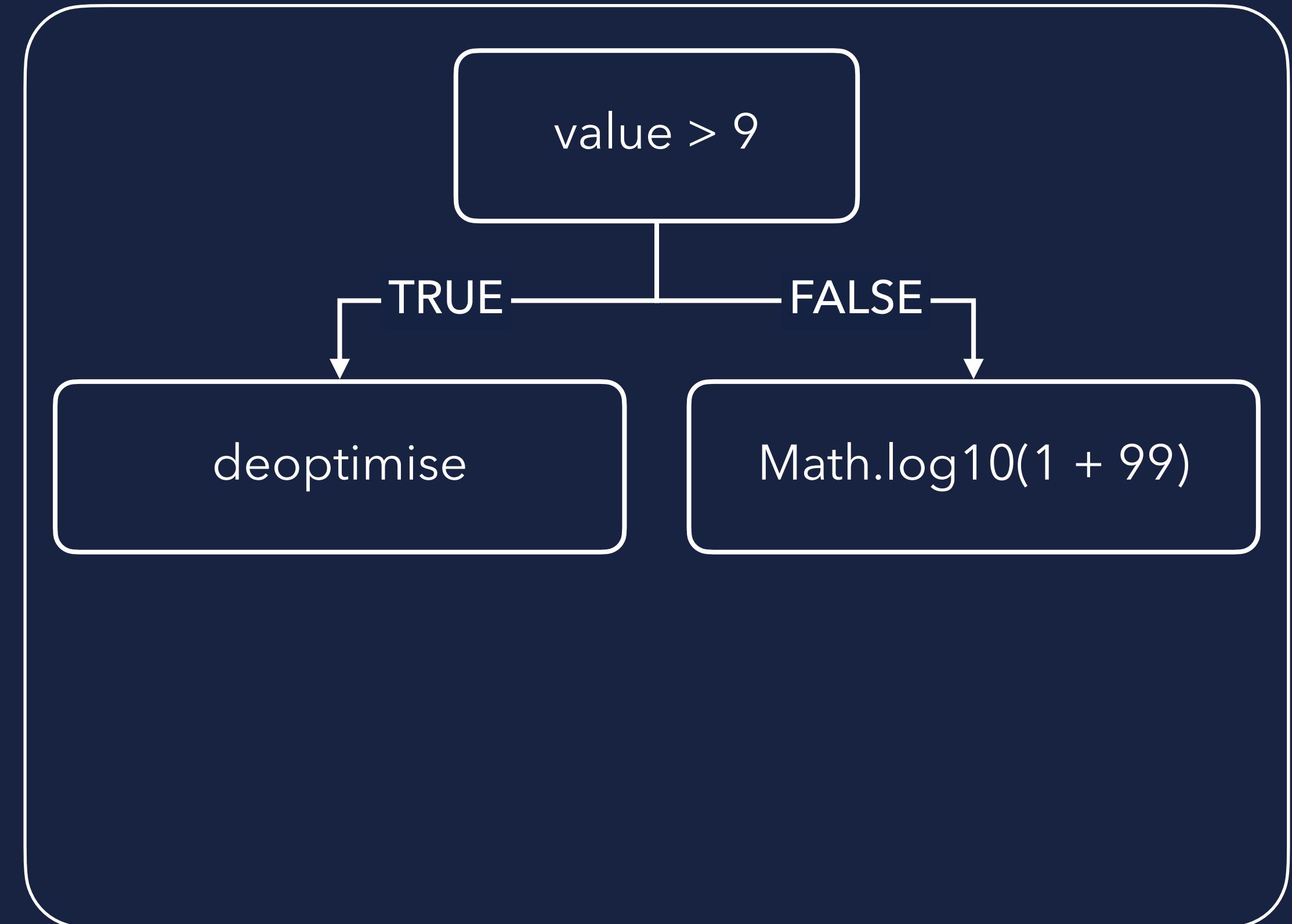


Value was never greater than 9

DEOPTIMISATION

e.g. BRANCH ANALYSIS

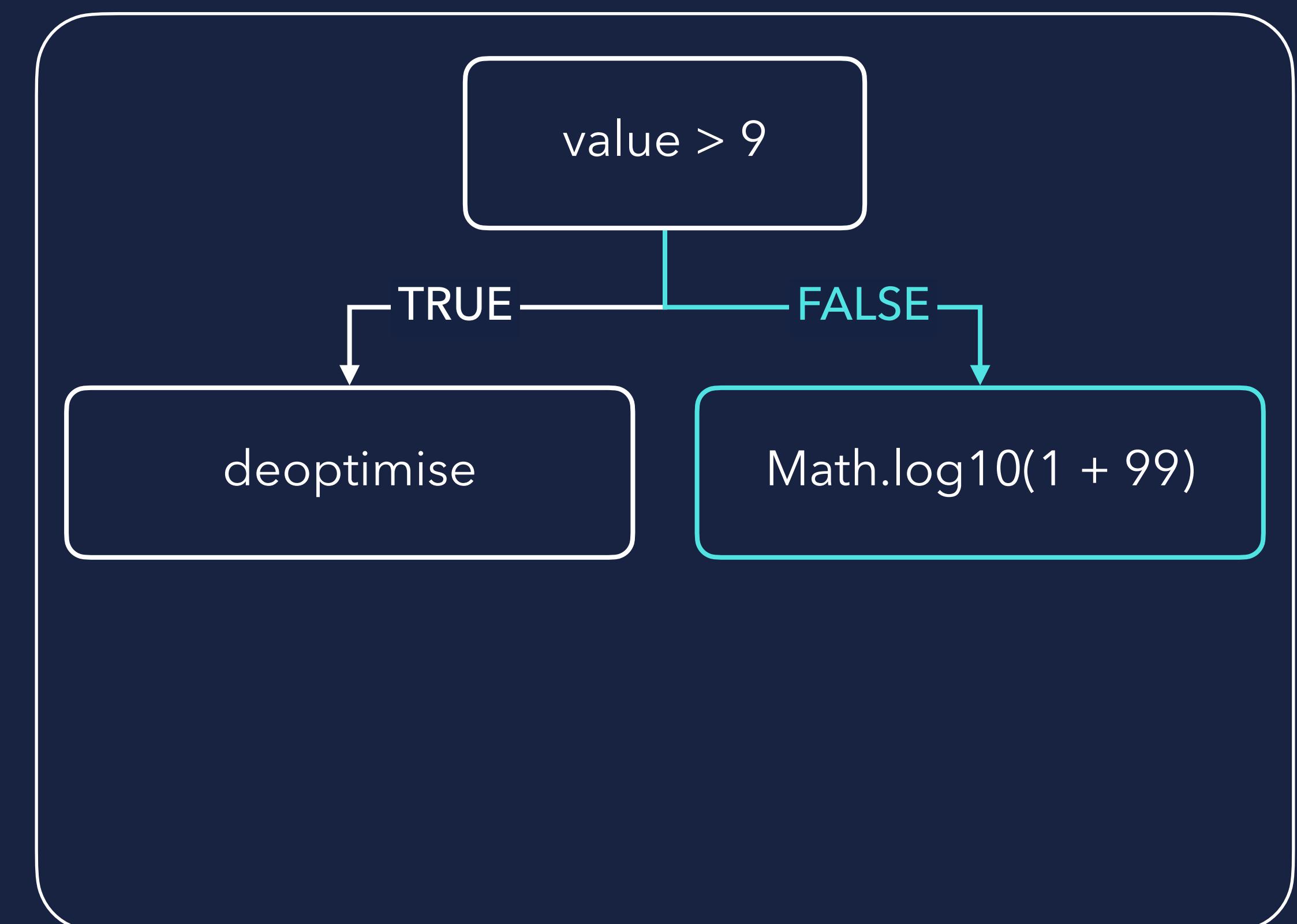
```
int computeMagnitude(int value) {  
    if (value > 9) {  
        uncommonTrap();  
    }  
    int bias = 1;  
    return Math.log10(bias + 99);  
}
```



DEOPTIMISATION

e.g. BRANCH ANALYSIS

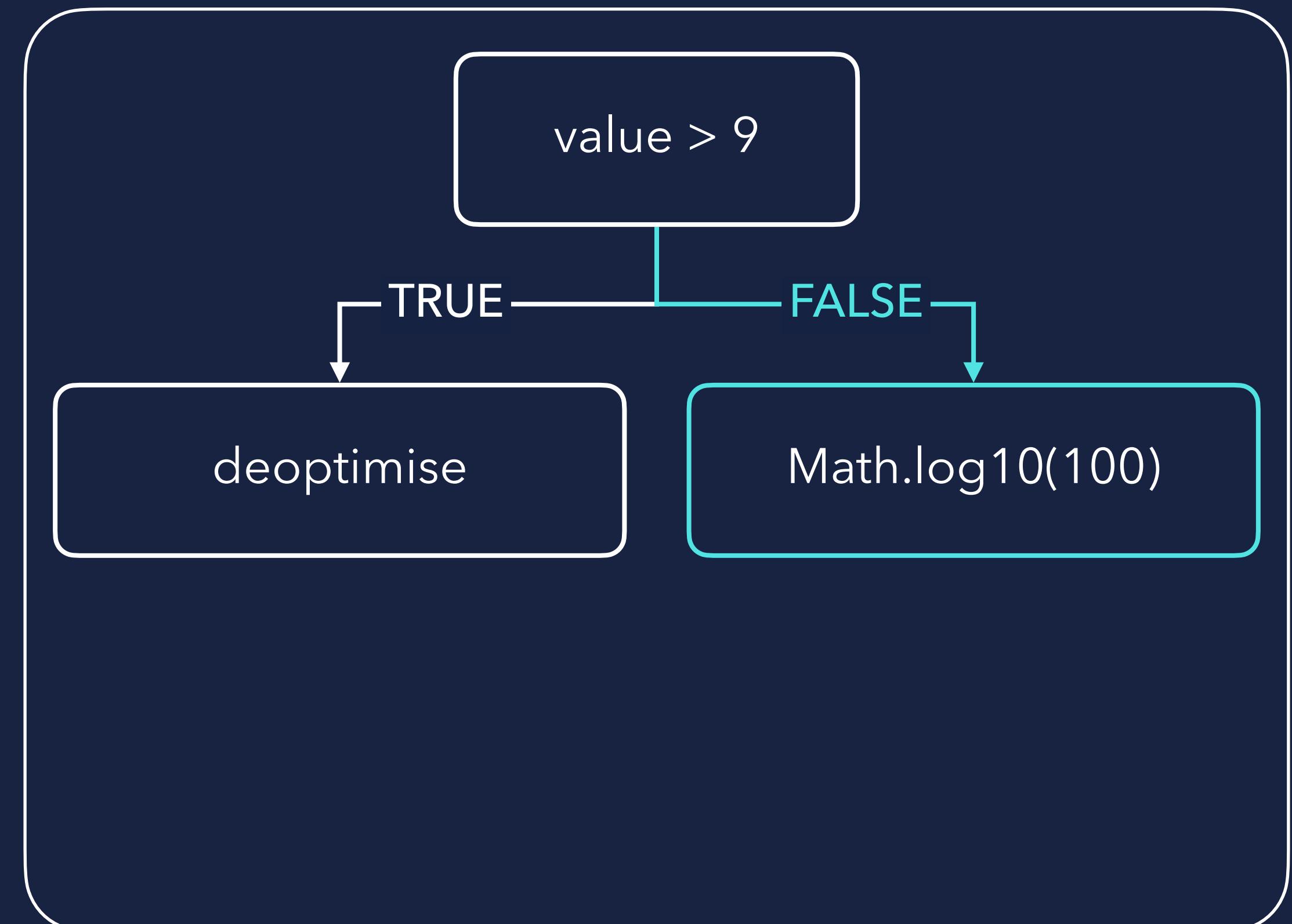
```
int computeMagnitude(int value) {  
    if (value > 9) {  
        uncommonTrap();  
    }  
    int bias = 1;  
    return Math.log10(bias + 99);  
}
```



DEOPTIMISATION

e.g. BRANCH ANALYSIS

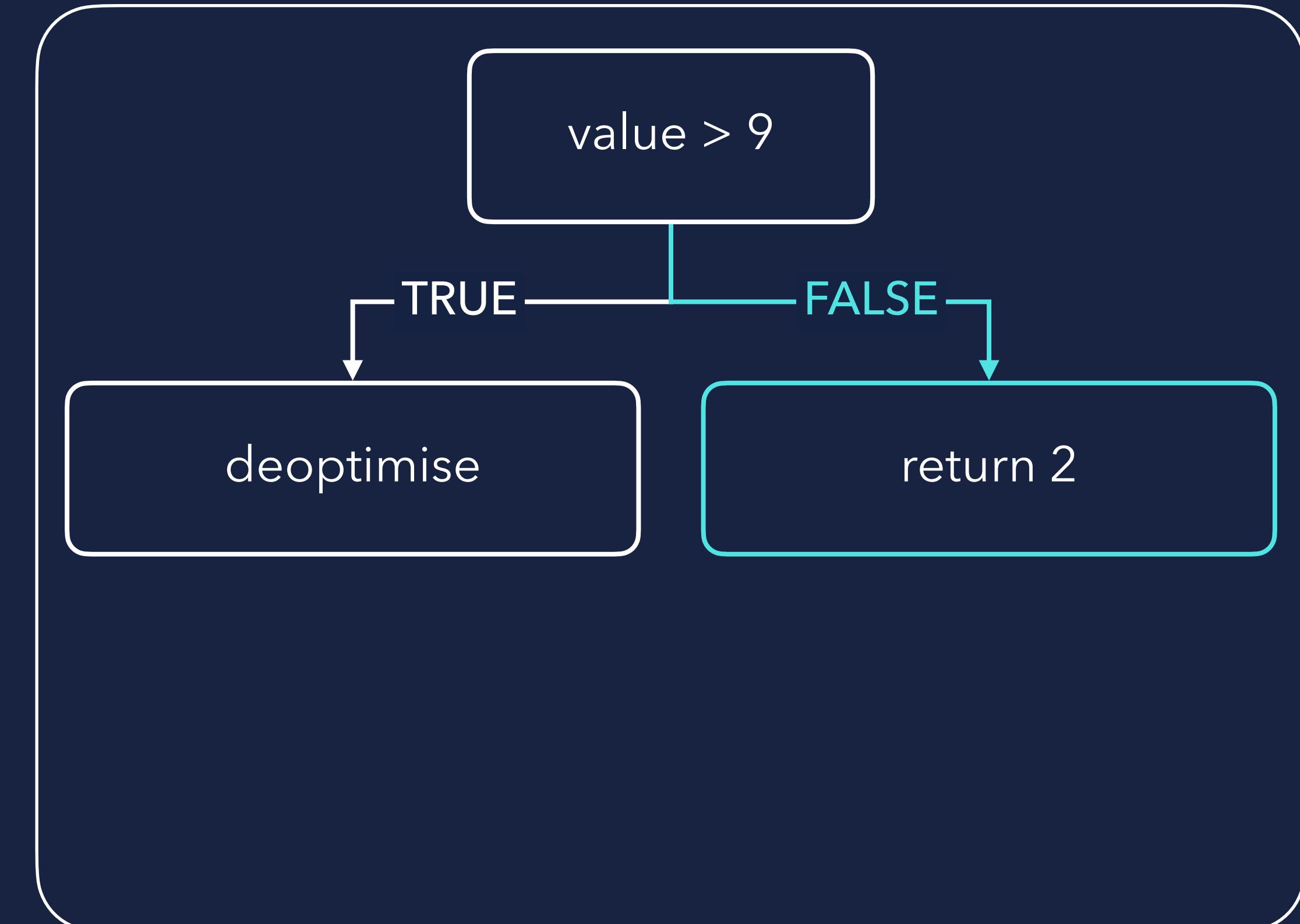
```
int computeMagnitude(int value) {  
    if (value > 9) {  
        uncommonTrap();  
    }  
    return Math.log10(100);  
}
```



DEOPTIMISATION

e.g. BRANCH ANALYSIS

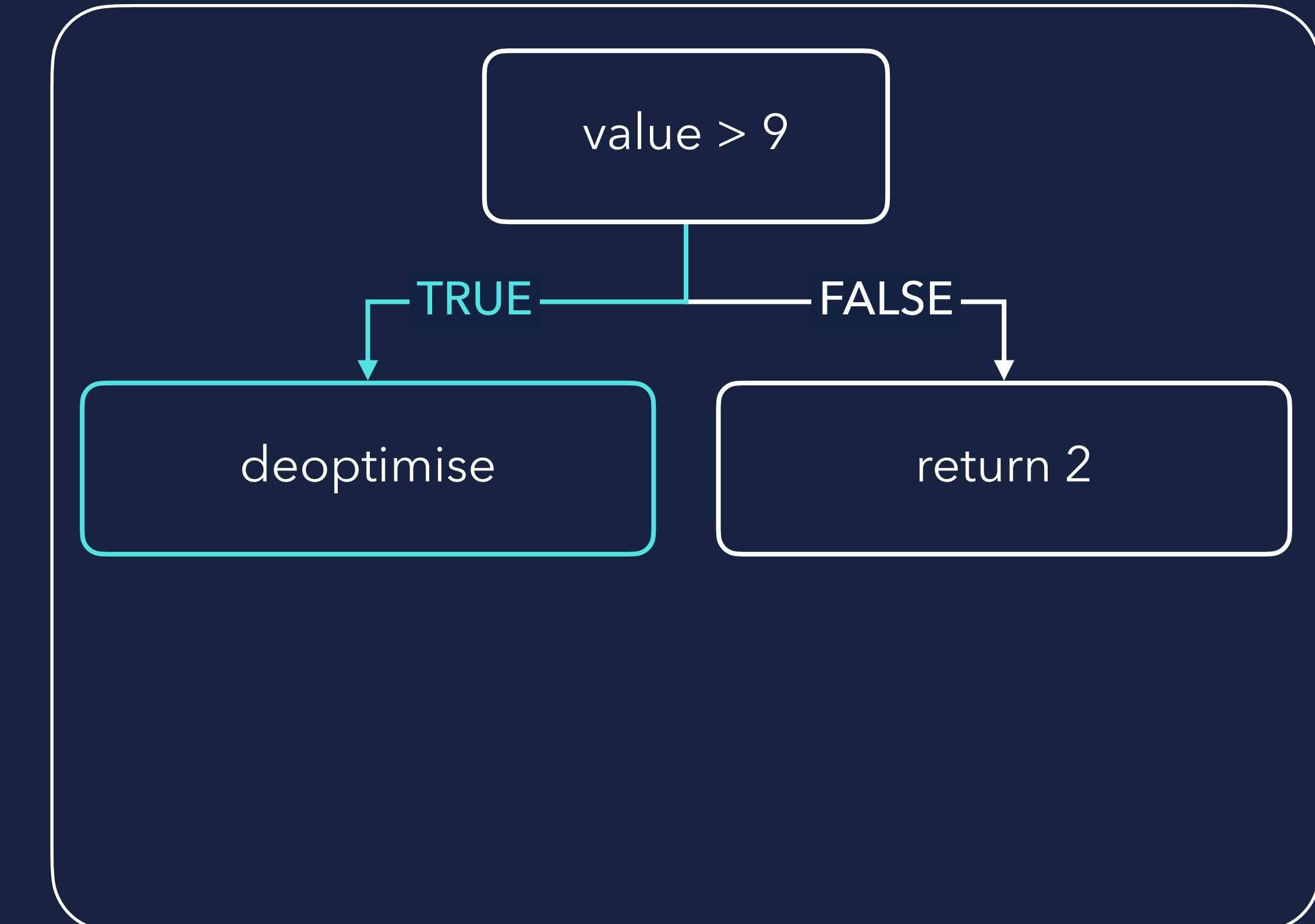
```
int computeMagnitude(int value) {  
    if (value > 9) {  
        uncommonTrap();  
    }  
    return 2;  
}
```



DEOPTIMISATION

e.g. BRANCH ANALYSIS

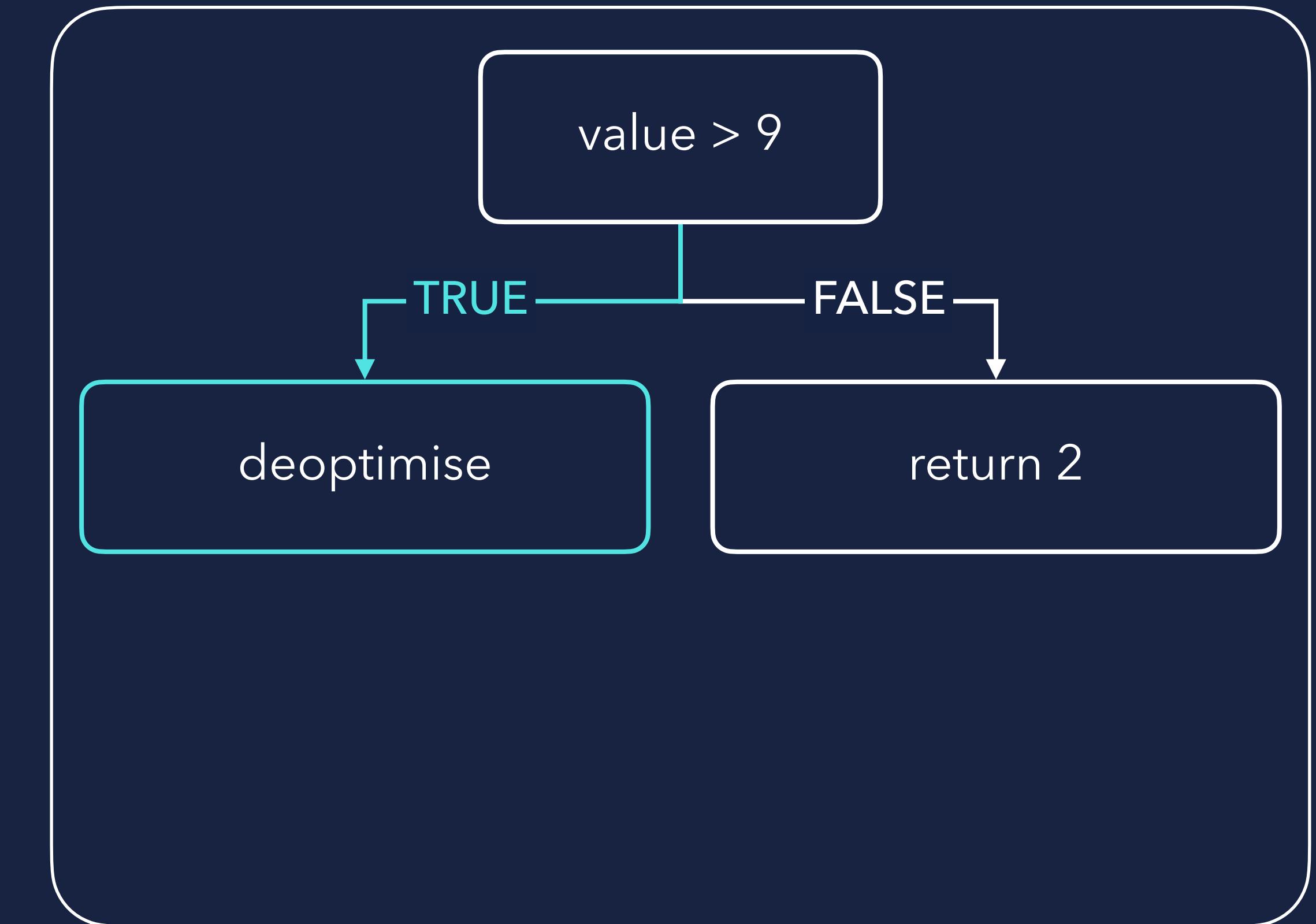
```
int computeMagnitude(int value) {  
    if (value > 9) {  
        uncommonTrap();  
    }  
    return 2;  
}
```



DEOPTIMISATION

e.g. BRANCH ANALYSIS

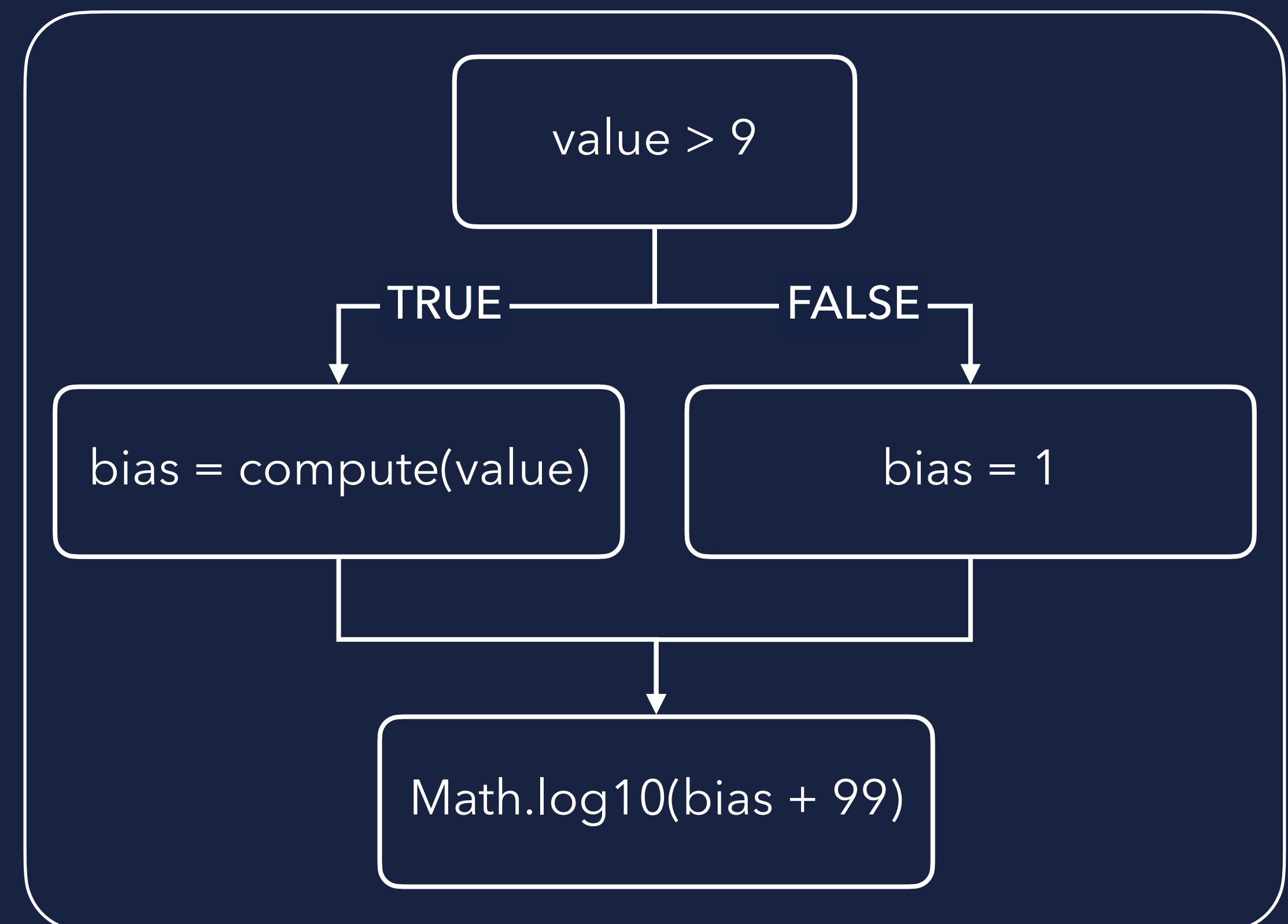
```
int computeMagnitude(int value) {  
    if (value > 9) {  
        uncommonTrap();  
    }  
    return 2;  
}
```



DEOPTIMISATION

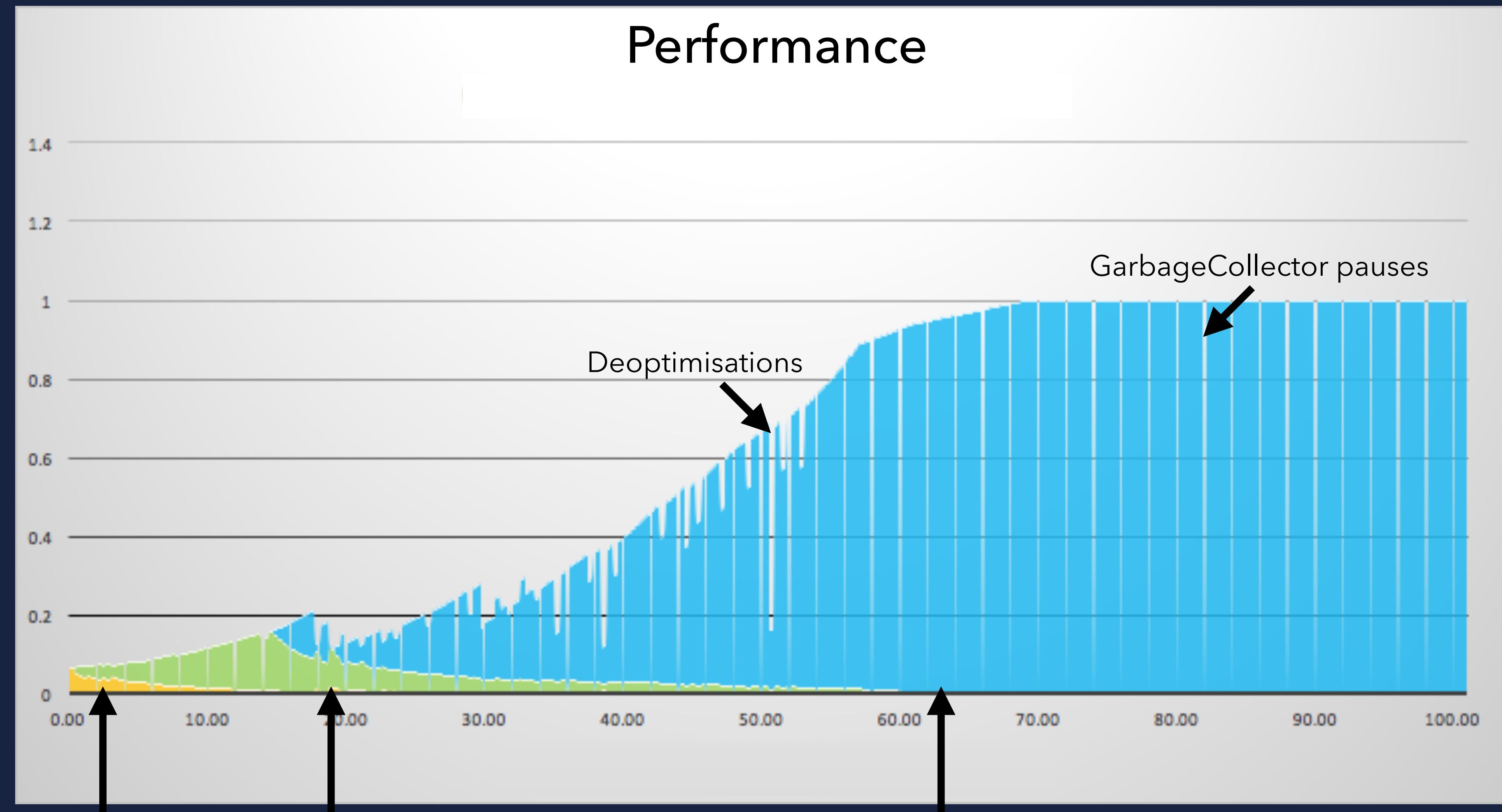
e.g. BRANCH ANALYSIS

```
int computeMagnitude(int value) {  
    int bias;  
    if (value > 9) {  
        bias = compute(value);  
    } else {  
        bias = 1;  
    }  
    return Math.log10(bias + 99);  
}
```



INTERPRETER → C1 → C2

JVM PERFORMANCE GRAPH



JVM

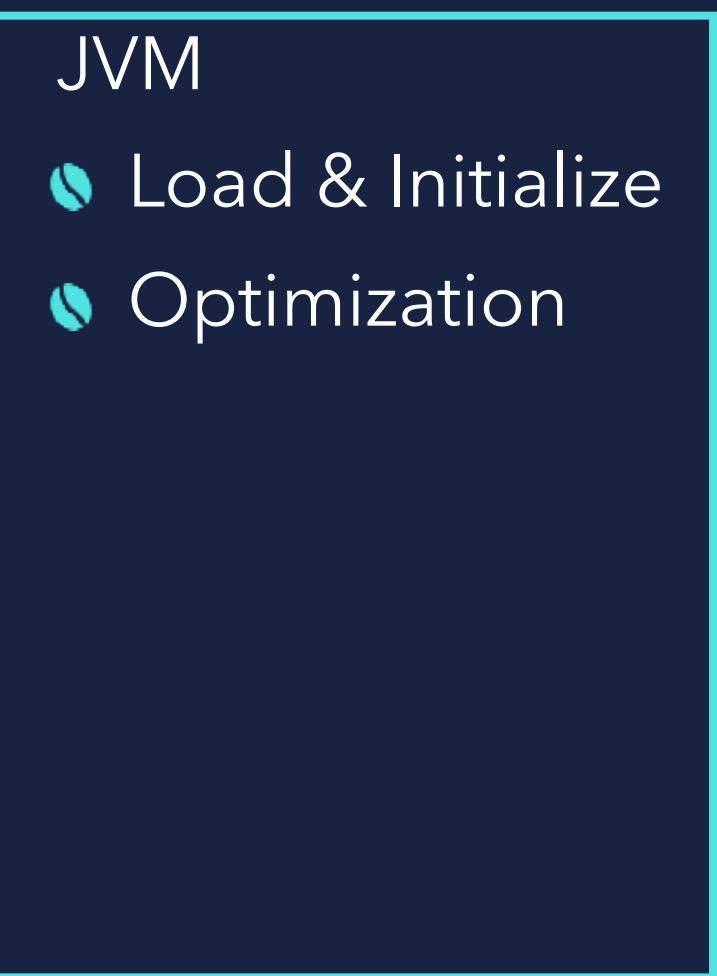
STARTUP

JVM STARTUP

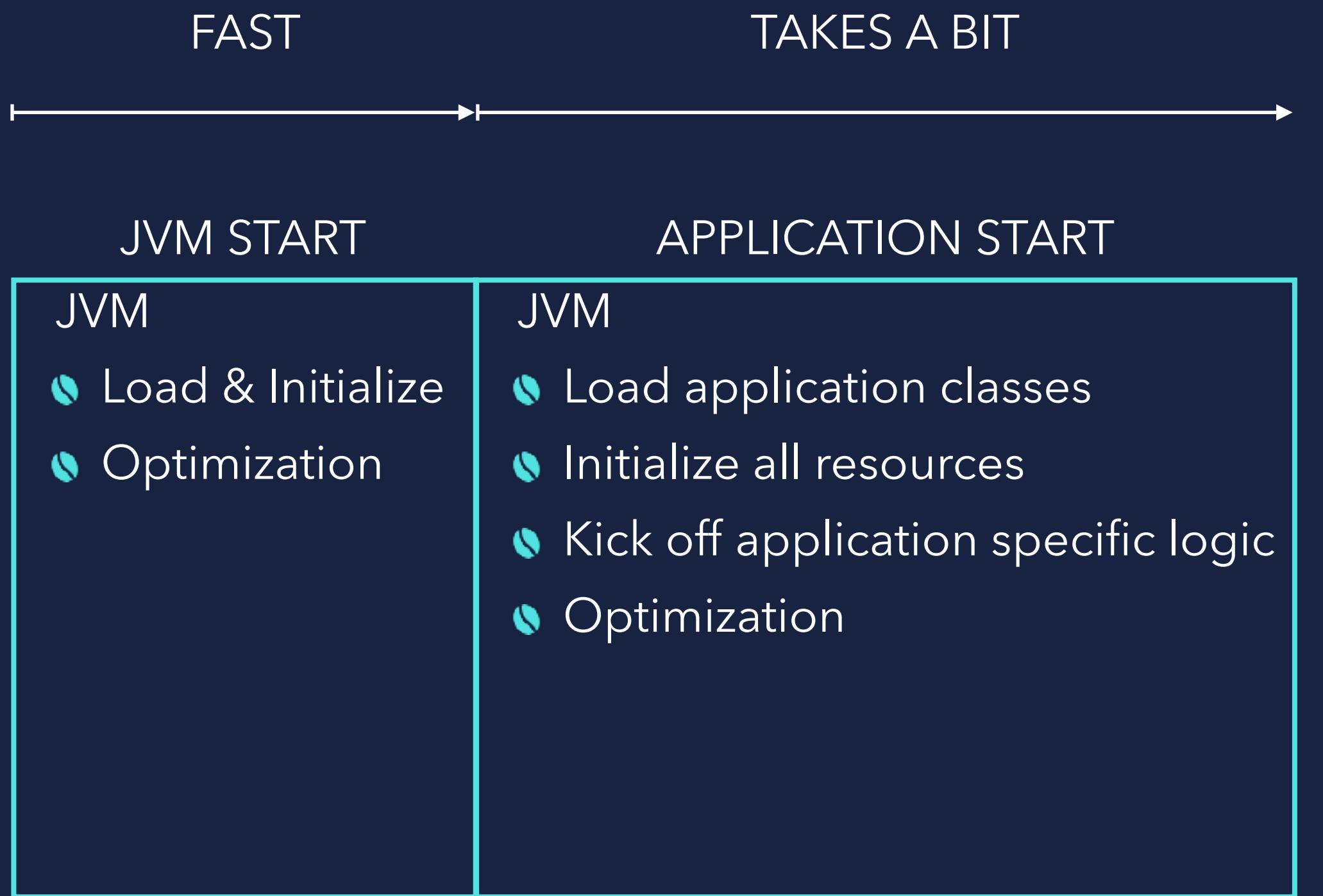
FAST



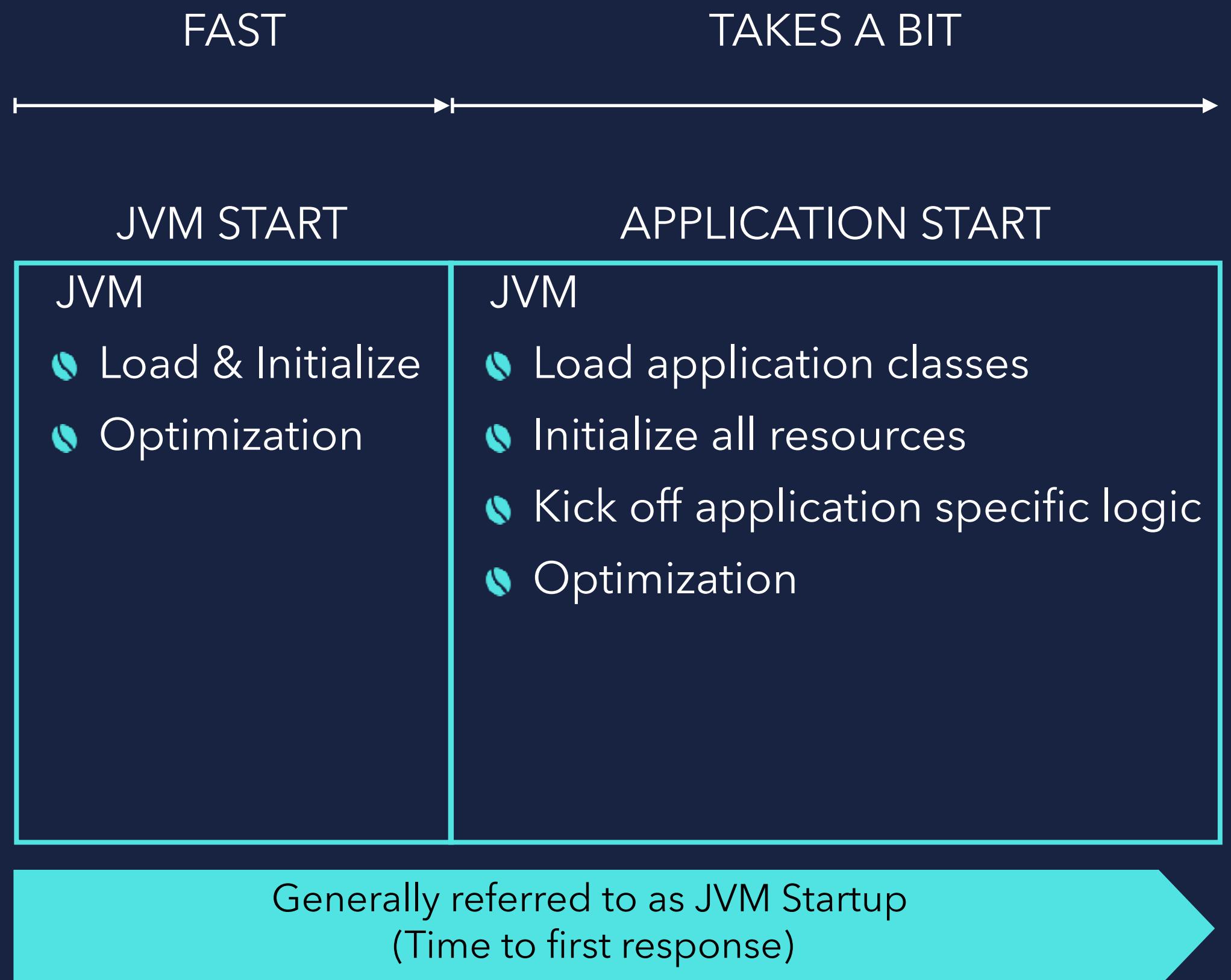
JVM START



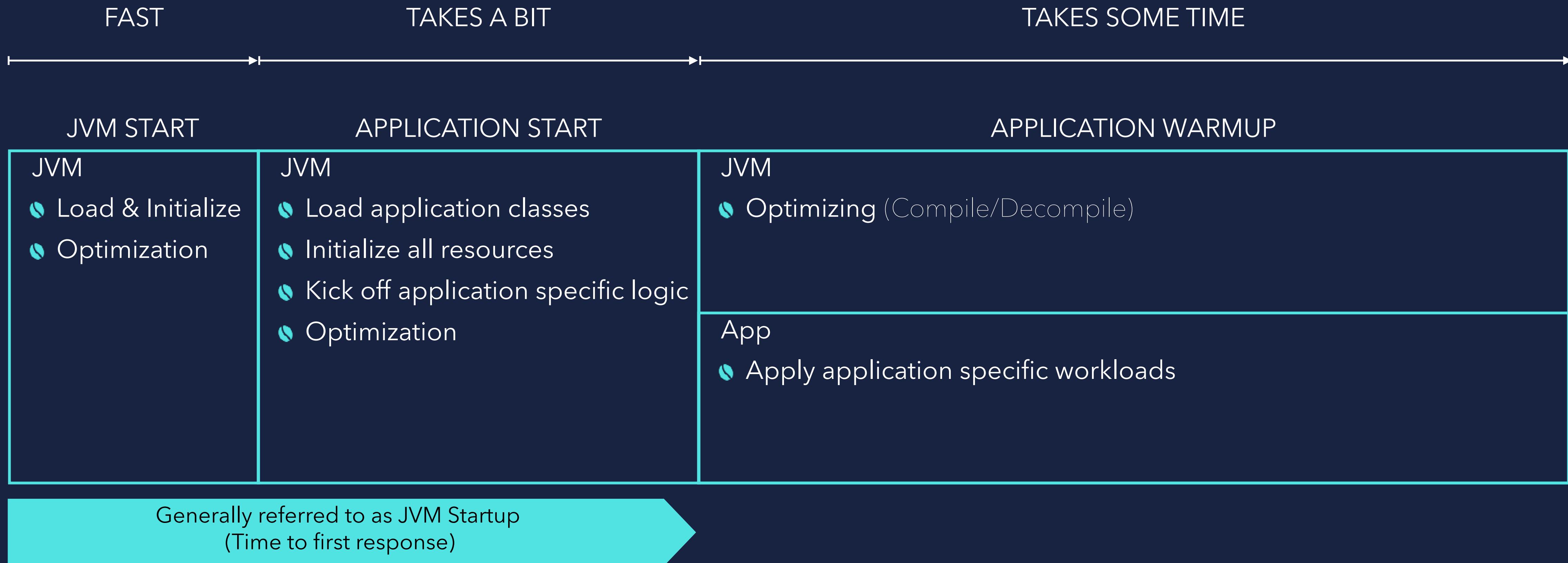
JVM STARTUP



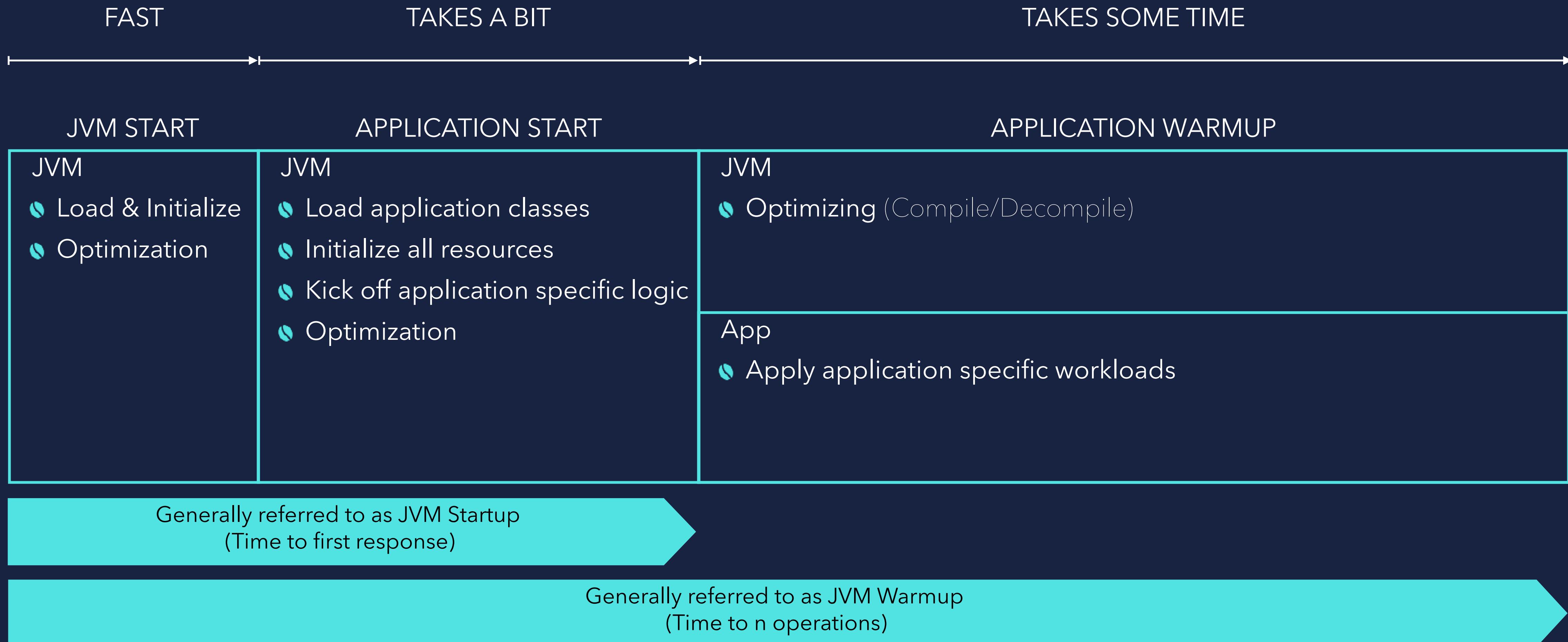
JVM STARTUP



JVM STARTUP



JVM STARTUP



THAT'S

GREAT.

...BUT ...

IT TAKES



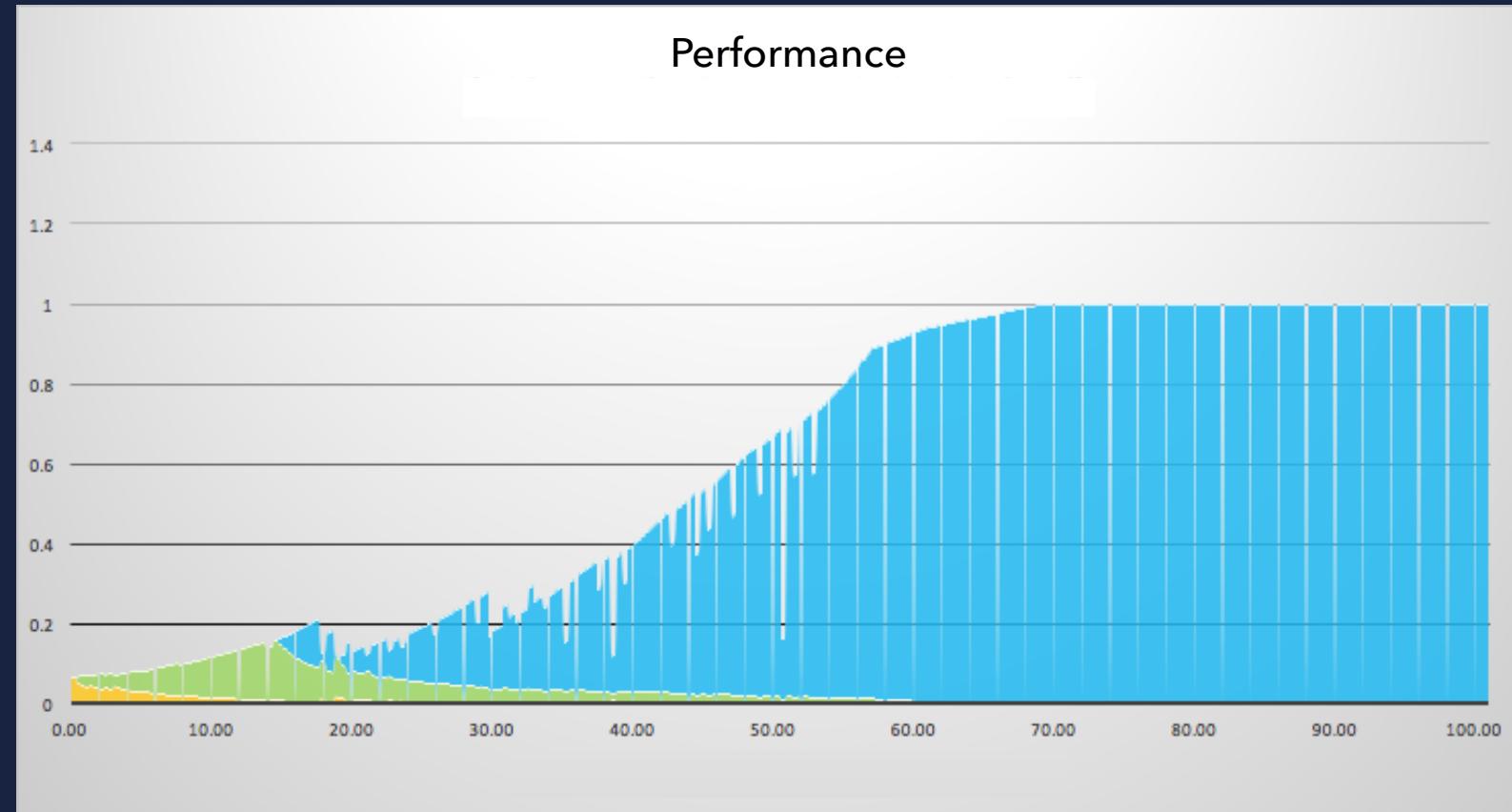
TIME!



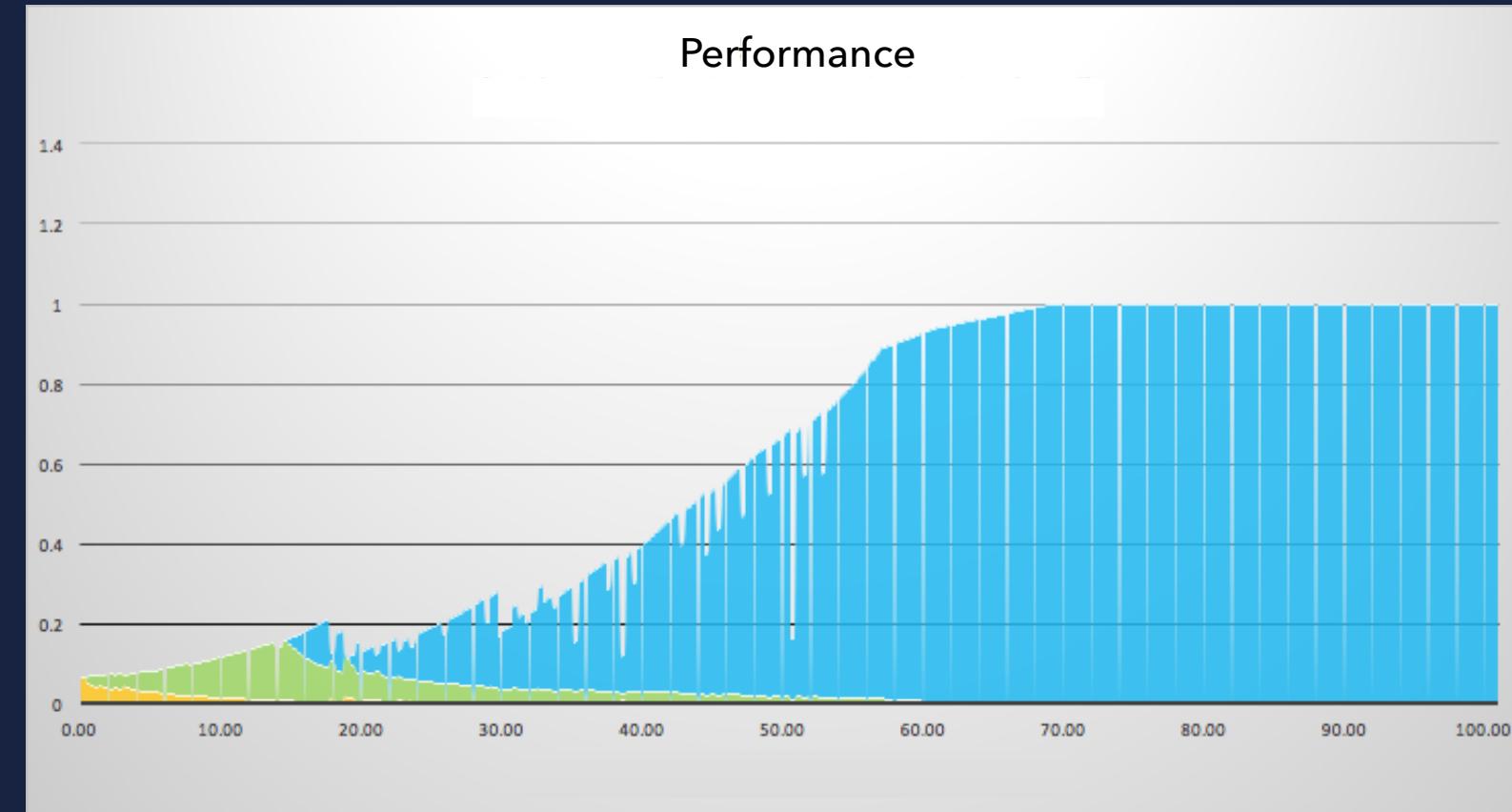
MICROSERVICE ENVIRONMENT

MICROSERVICE ENVIRONMENT

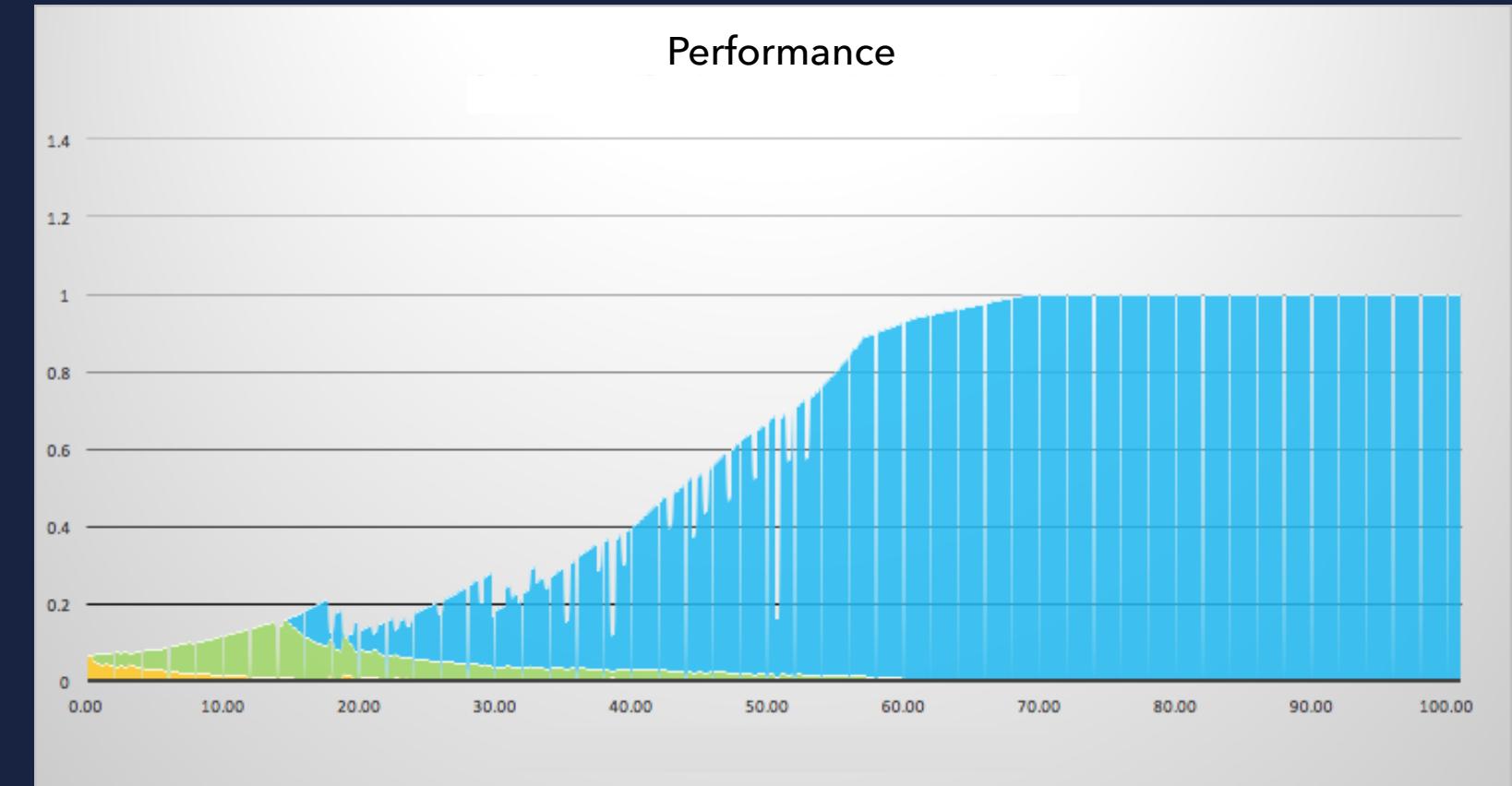
FIRST RUN



SECOND RUN



THIRD RUN



JVM STARTUP

JVM STARTUP

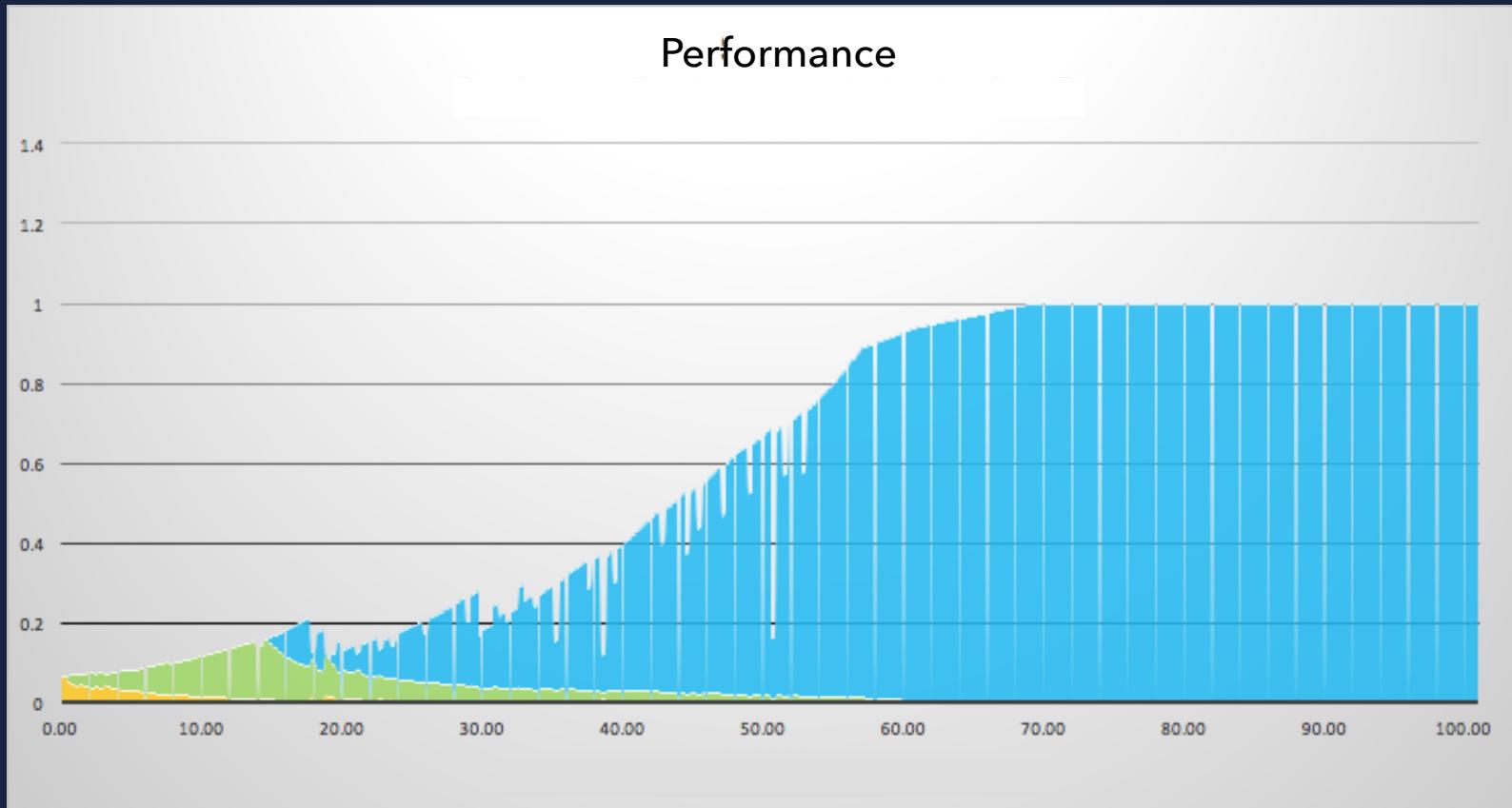
JVM STARTUP



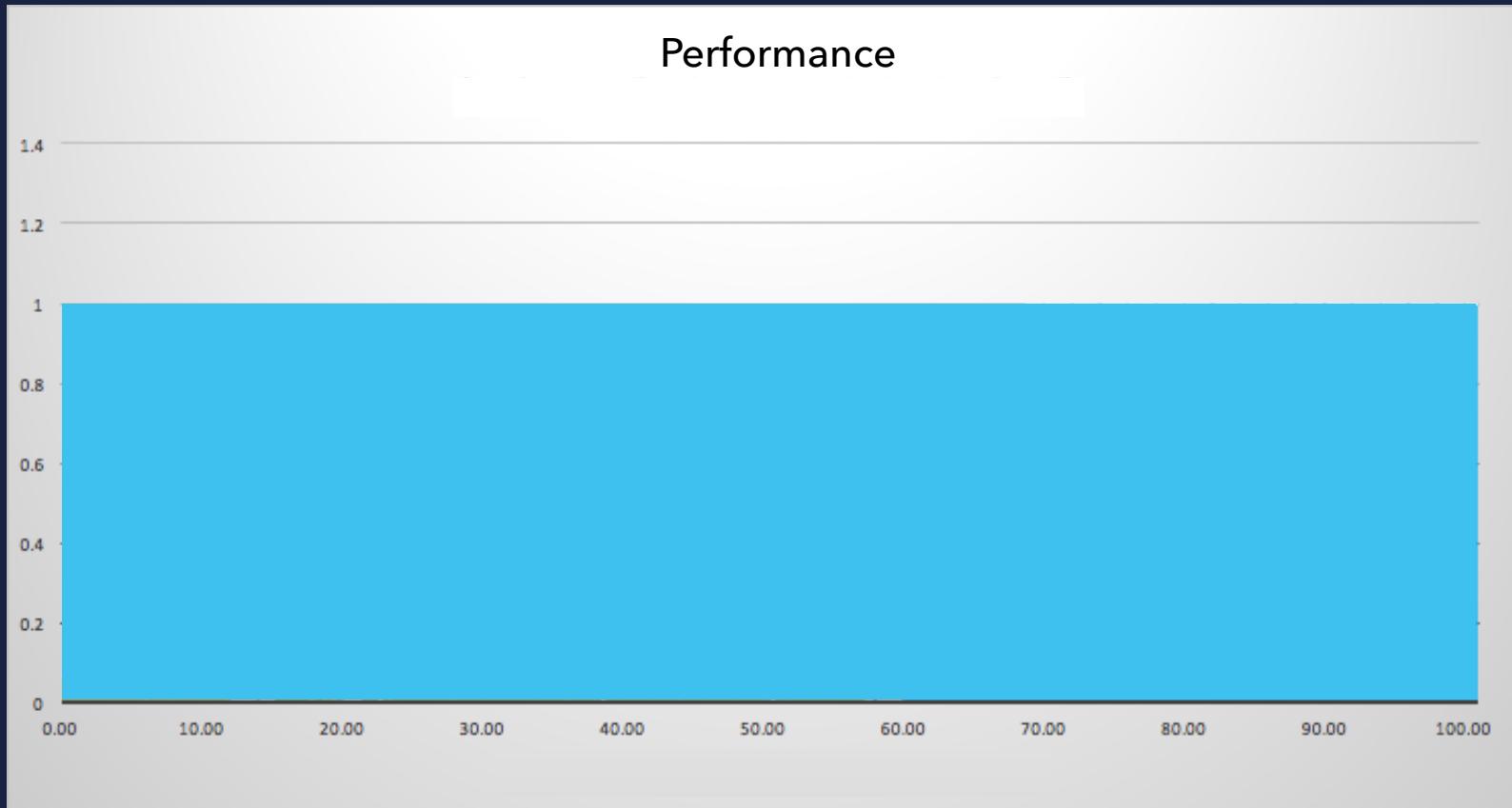
azul

WOULDN'T IT BE GREAT...?

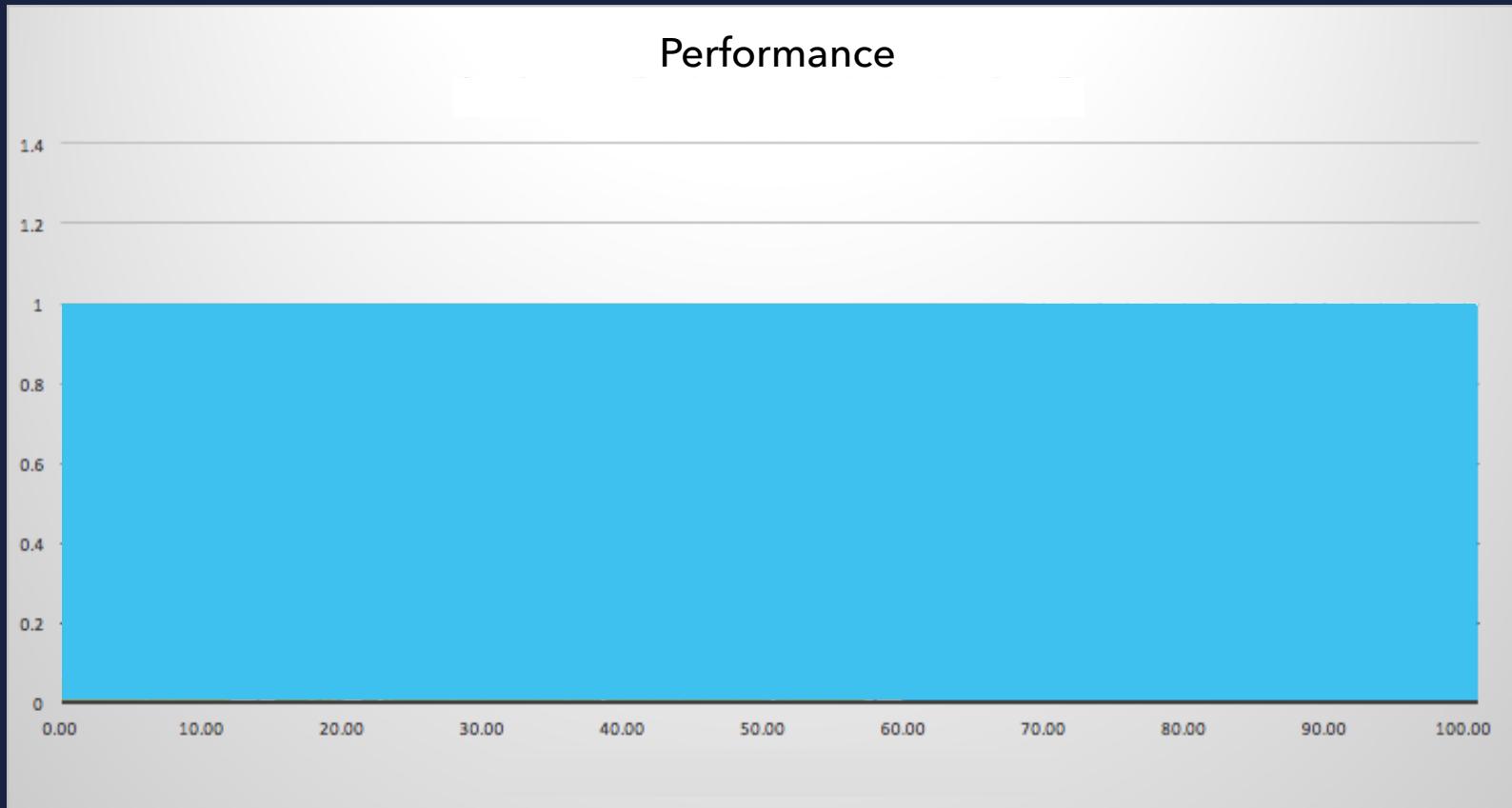
FIRST RUN



SECOND RUN



THIRD RUN



JVM STARTUP

NO STARTUP OVERHEAD

NO STARTUP OVERHEAD

SOLUTIONS...?

CLASS DATA
SHARING

WHAT ABOUT CDS?

- Dump internal class representations into file

WHAT ABOUT CDS?

- Dump internal class representations into file
- Shared on each JVM start (CDS)

WHAT ABOUT CDS?

- Dump internal class representations into file
- Shared on each JVM start (CDS)
- No optimization or hotspot detection

WHAT ABOUT CDS?

- Dump internal class representations into file
- Shared on each JVM start (CDS)
- No optimization or hotspot detection
- Only reduces class loading time

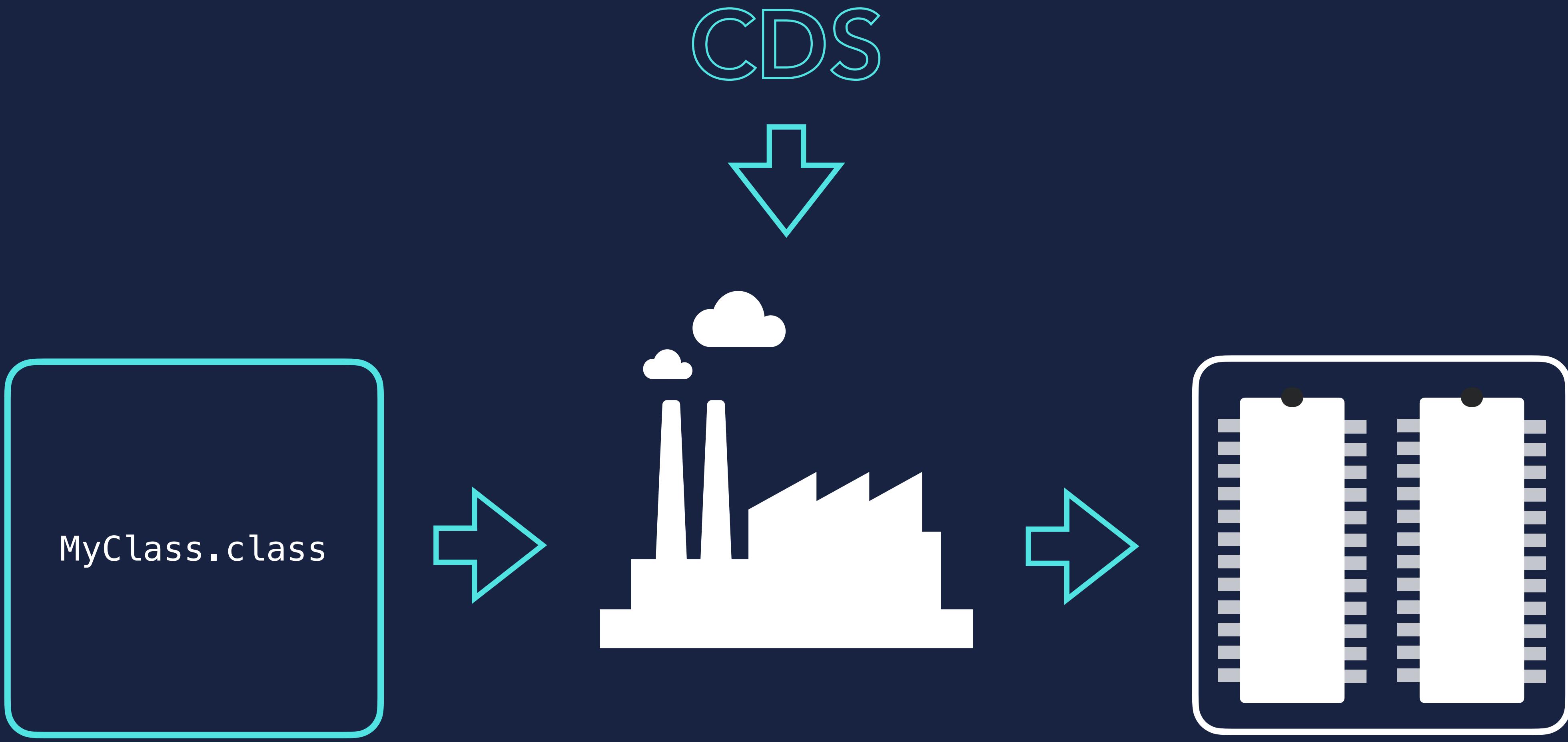
WHAT ABOUT CDS?

- Dump internal class representations into file
- Shared on each JVM start (CDS)
- No optimization or hotspot detection
- Only reduces class loading time
- Startup up to 2 seconds faster

WHAT ABOUT CDS?

- Dump internal class representations into file
- Shared on each JVM start (CDS)
- No optimization or hotspot detection
- Only reduces class loading time
- Startup up to 2 seconds faster
- Good info from Ionut Balosin





BYTE CODE

CLASS LOADER

JVM MEMORY

AHEAD OF TIME
COMPILATION

WHY NOT USE AOT?

- No interpreting bytecodes

WHY NOT USE AOT?

- No interpreting bytecodes
- No analysis of hotspots

WHY NOT USE AOT?

- No interpreting bytecodes
- No analysis of hotspots
- No runtime compilation of code

WHY NOT USE AOT?

- No interpreting bytecodes
- No analysis of hotspots
- No runtime compilation of code
- Start at 'full speed', straight away

WHY NOT USE AOT?

- No interpreting bytecodes
- No analysis of hotspots
- No runtime compilation of code
- Start at 'full speed', straight away
- GraalVM native image does that

PROBLEM SOLVED...?

NOT SO FAST ...

- AOT is, by definition, static

NOT SO FAST ...

- AOT is, by definition, static
- Code is compiled before it is run

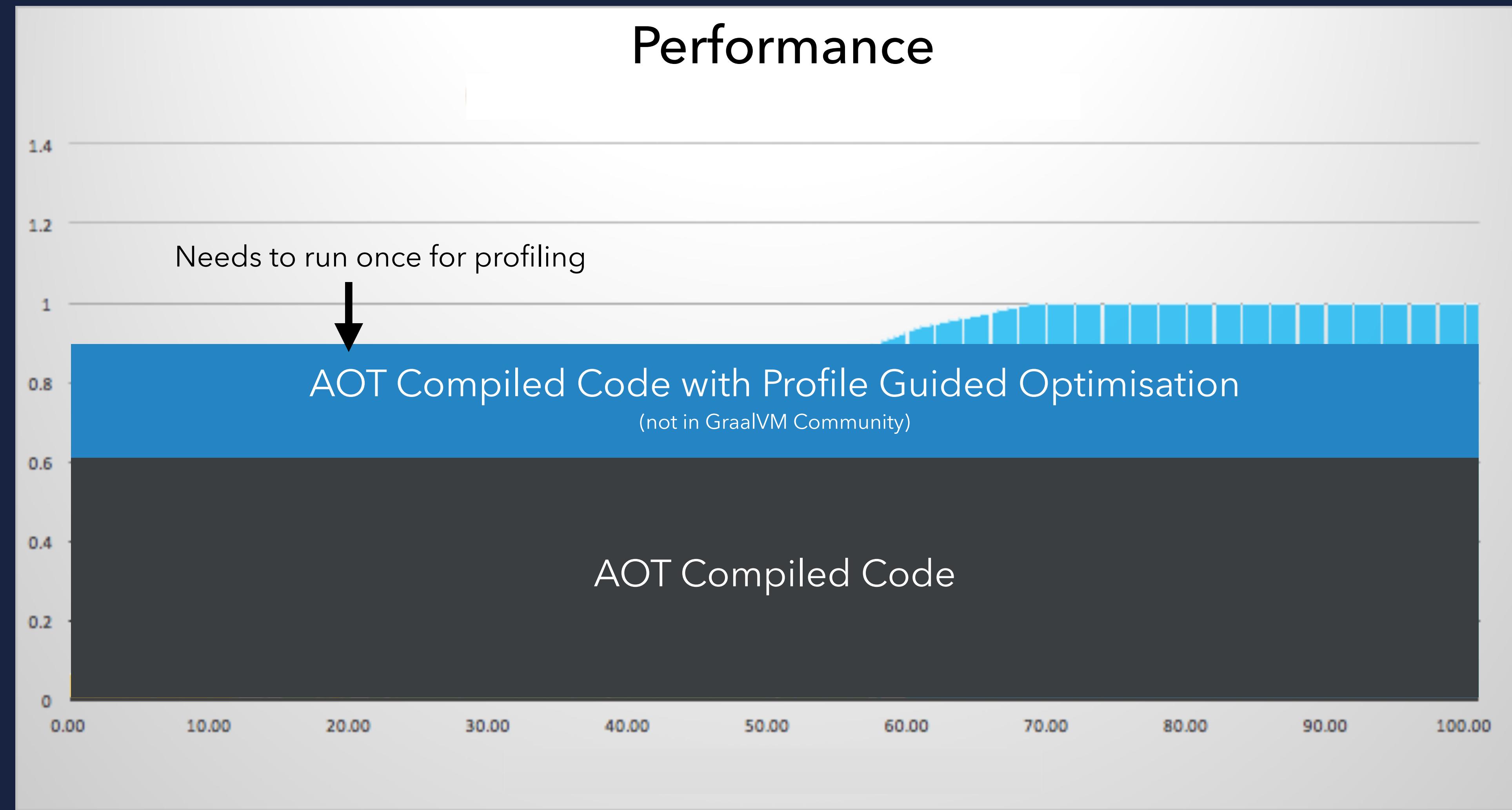
NOT SO FAST ...

- AOT is, by definition, static
- Code is compiled before it is run
- Compiler has no knowledge of how the code will actually run

NOT SO FAST...

- AOT is, by definition, static
- Code is compiled before it is run
- Compiler has no knowledge of how the code will actually run
- Profile Guided Optimisation (PGO) can partially help

JVM PERFORMANCE GRAPH



A DIFFERENT
APPROACH



CRIU

CHECKPOINT RESTORE IN USERSPACE

CRIU

- Linux project



CRIU

- Linux project
- Part of kernel >= 3.11 (2013)



CRIU



- Linux project
- Part of kernel >= 3.11 (2013)
- Freeze a running container/application

CRIU



- Linux project
- Part of kernel >= 3.11 (2013)
- Freeze a running container/application
- Checkpoint its state to disk

CRIU



- Linux project
- Part of kernel >= 3.11 (2013)
- Freeze a running container/application
- Checkpoint its state to disk
- Restore the container/application from the saved data.

CRIU



- Linux project
- Part of kernel >= 3.11 (2013)
- Freeze a running container/application
- Checkpoint its state to disk
- Restore the container/application from the saved data.
- Used by/integrated in OpenVZ, LXC/LXD, Docker, Podman and others

CRIU

- Heavily relies on /proc file system



CRIU



- Heavily relies on `/proc` file system
- It can checkpoint:
 - Processes and threads
 - Application memory, memory mapped files and shared memory
 - Open files, pipes and FIFOs
 - Sockets
 - Interprocess communication channels
 - Timers and signals

CRIU



- Heavily relies on `/proc` file system
- It can checkpoint:
 - Processes and threads
 - Application memory, memory mapped files and shared memory
 - Open files, pipes and FIFOs
 - Sockets
 - Interprocess communication channels
 - Timers and signals
- Can rebuild TCP connection from one side only

CRIU
CHALLENGES

CRIU CHALLENGES



- Restart from saved state on another machine
(open files, shared memory etc.)

CRIU CHALLENGES



- Restart from saved state on another machine
(open files, shared memory etc.)
- Start multiple instances of same state on same machine
(PID will be restored which will lead to problems)

CRIU CHALLENGES



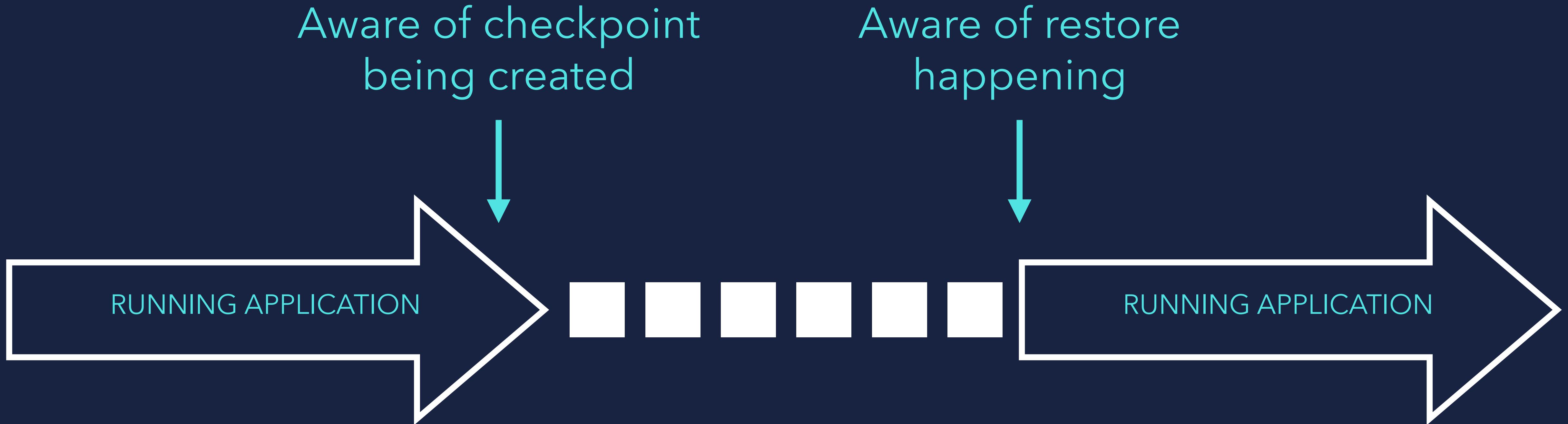
- Restart from saved state on another machine
(open files, shared memory etc.)
- Start multiple instances of same state on same machine
(PID will be restored which will lead to problems)
- A Java Virtual Machine would assume it was continuing its tasks
(very difficult to use effectively, e.g. running applications might have open files etc.)

CRAAC

Coordinated Restore at Checkpoint

CRaC

A way to solve the problems when checkpointing a JVM
(e.g. no open files, sockets etc.)



CRaC

- Comes with a simple API

CRaC

- Comes with a simple API
- Creates checkpoints using code or jcmd

CRaC

- Comes with a simple API
- Creates checkpoints using code or jcmd
- Throws CheckpointException

(in case of open files/sockets)

CRaC

- Comes with a simple API
- Creates checkpoints using code or jcmd
- Throws CheckpointException
 - (in case of open files/sockets)
- Heap is cleaned, compacted
 - (using JVM safepoint mechanism -> JVM is in a safe state)



Additional command line parameters

START

```
>java -XX:CRaCCheckpointTo=PATH -jar app.jar
```

RESTORE

```
>java -XX:CRaCRestoreFrom=PATH
```

openjdk.org/projects/cracker

Lead by Anton Kozlov (Azul)



CRAC API

CRaCAPI

- Resource interface (can be notified about a Checkpoint and Restore)

<<interface>>

Resource

beforeCheckpoint()

afterRestore()

CRaCAPI

- Resource interface (can be notified about a Checkpoint and Restore)
- Classes in application code implement the Resource interface

<<interface>>

Resource

beforeCheckpoint()

afterRestore()

CRaCAPI

- Resource interface (can be notified about a Checkpoint and Restore)
- Classes in application code implement the Resource interface
- Application receives callbacks during checkpointing and restoring

<<interface>>

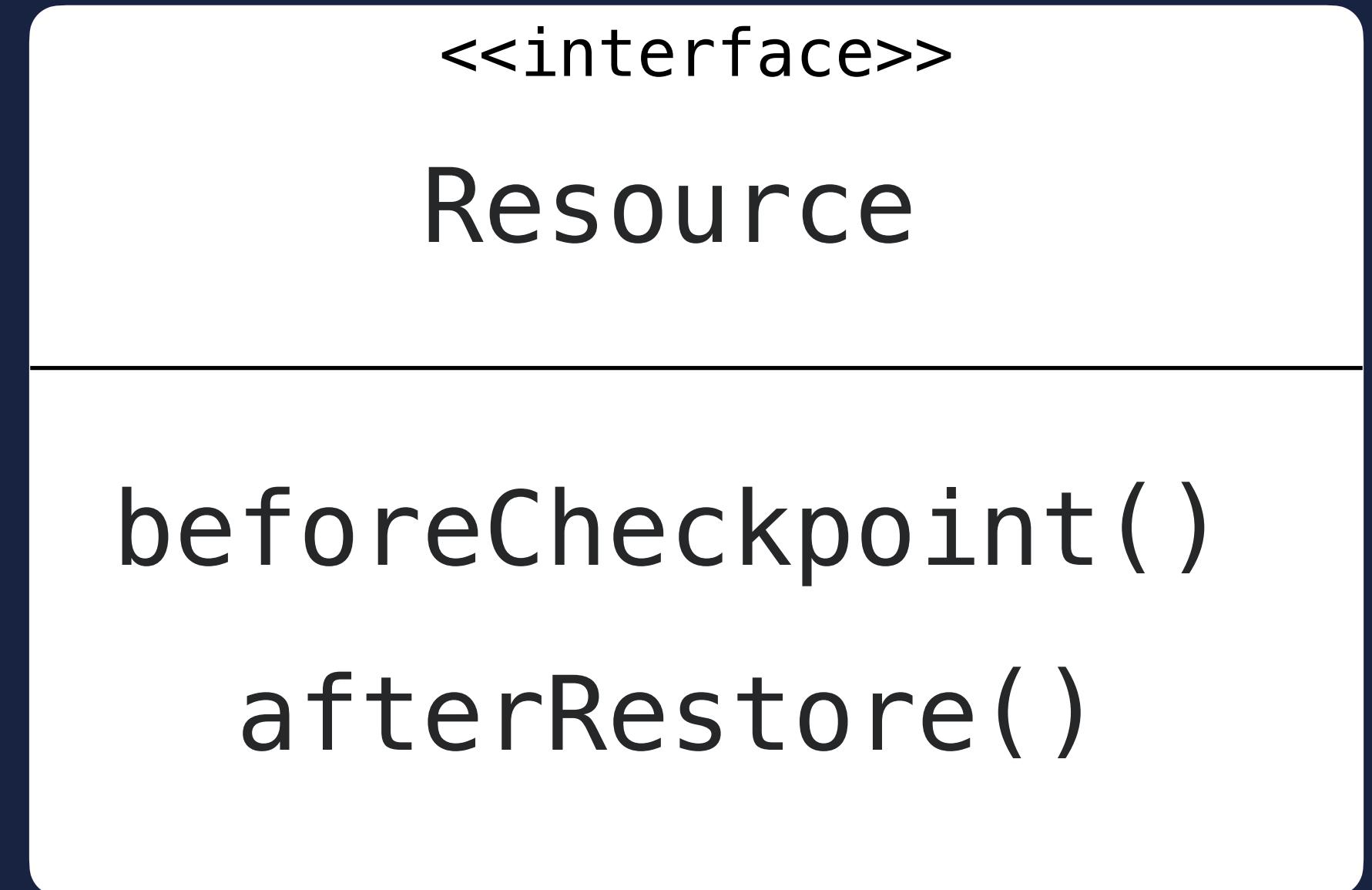
Resource

beforeCheckpoint()

afterRestore()

CRaCAPI

- Resource interface (can be notified about a Checkpoint and Restore)
- Classes in application code implement the Resource interface
- Application receives callbacks during checkpointing and restoring
- Makes it possible to close/restore resources (e.g. open files, sockets)



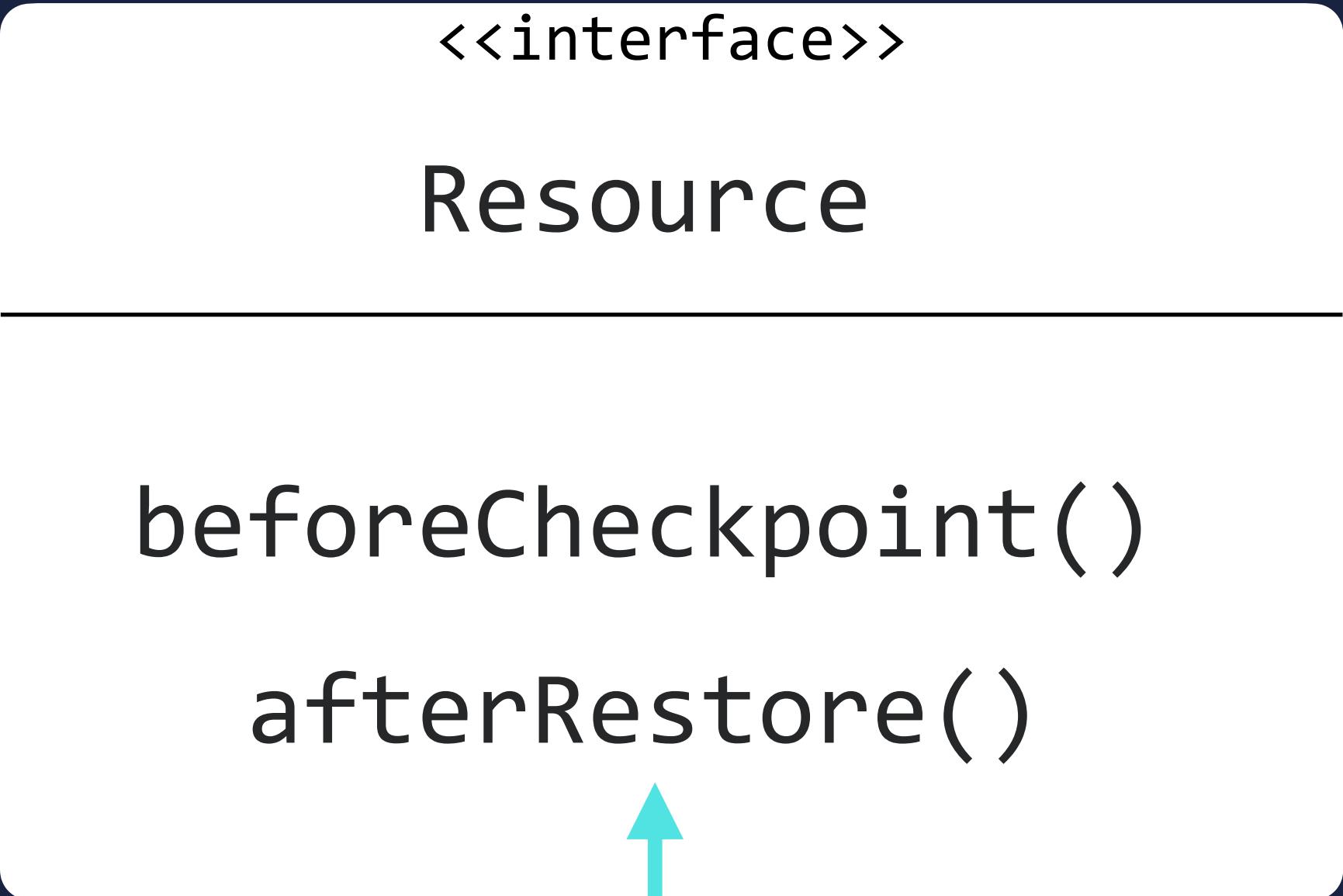
CRaCAPI

- Resource objects need to be registered with a Context so that they can receive notifications

CRaCAPI

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

CRaCAPI



CREATING

A

CHECKPOINT

CREATING A CHECKPOINT

FROM THE COMMAND LINE:

```
>jcmd YOUR_AWESOME.jar JDK.checkpoint
```

```
>jcmd PID JDK.checkpoint
```

CREATING A CHECKPOINT

FROM CODE:

```
Core.checkpointRestore();
```

WHEN?

WHEN TO CHECKPOINT ?

- Start your app with -XX:+PrintCompilation

WHEN TO CHECKPOINT ?

- Start your app with -XX:+PrintCompilation
- Apply typical workload to your app

WHEN TO CHECKPOINT ?

- Start your app with -XX:+PrintCompilation
- Apply typical workload to your app
- Observe the moment the compilations are ramped down

WHEN TO CHECKPOINT ?

- Start your app with -XX:+PrintCompilation
- Apply typical workload to your app
- Observe the moment the compilations are ramped down
- Create the checkpoint

CRAC OVERVIEW

CRaC OVERVIEW

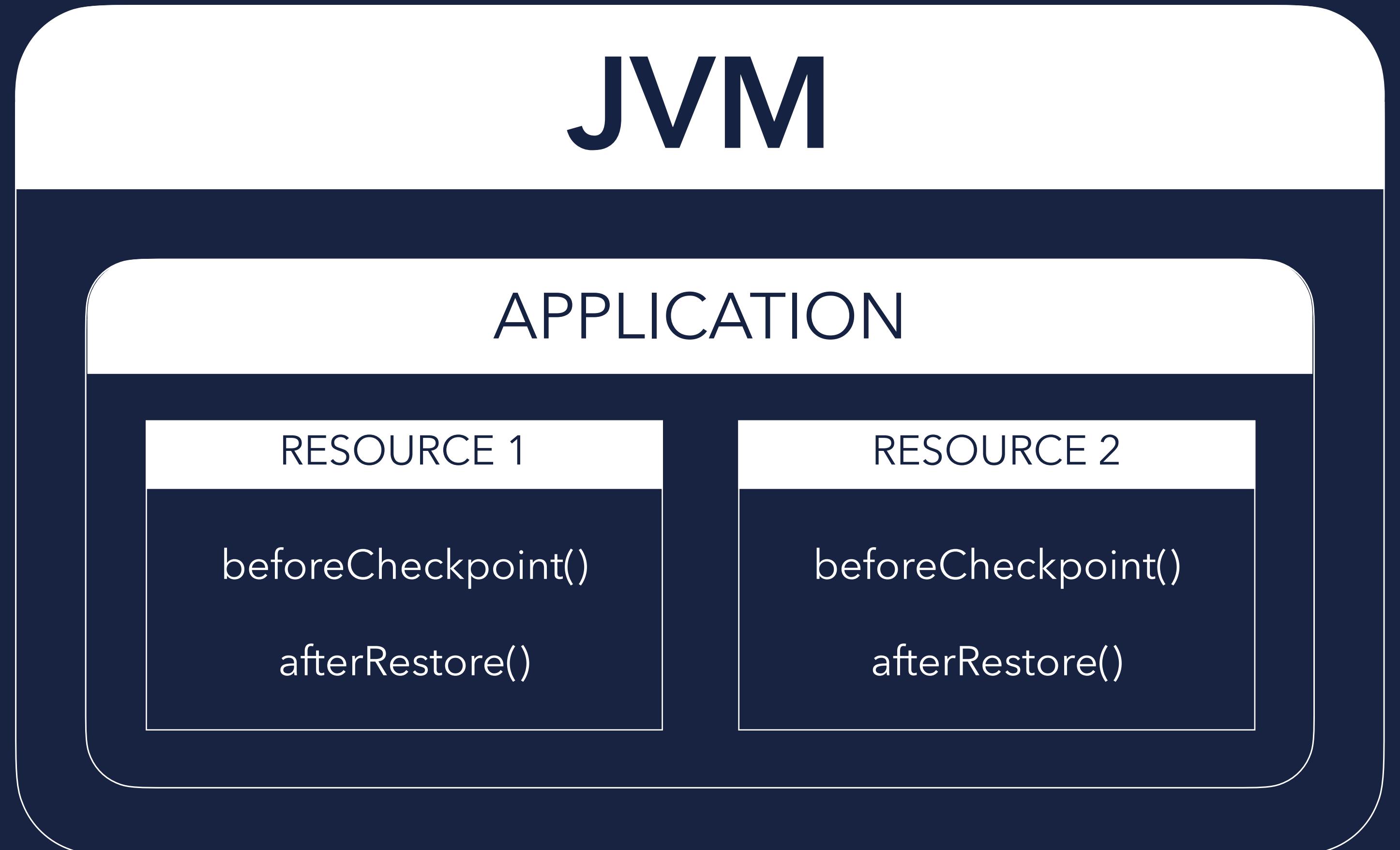
JVM

CRaC OVERVIEW

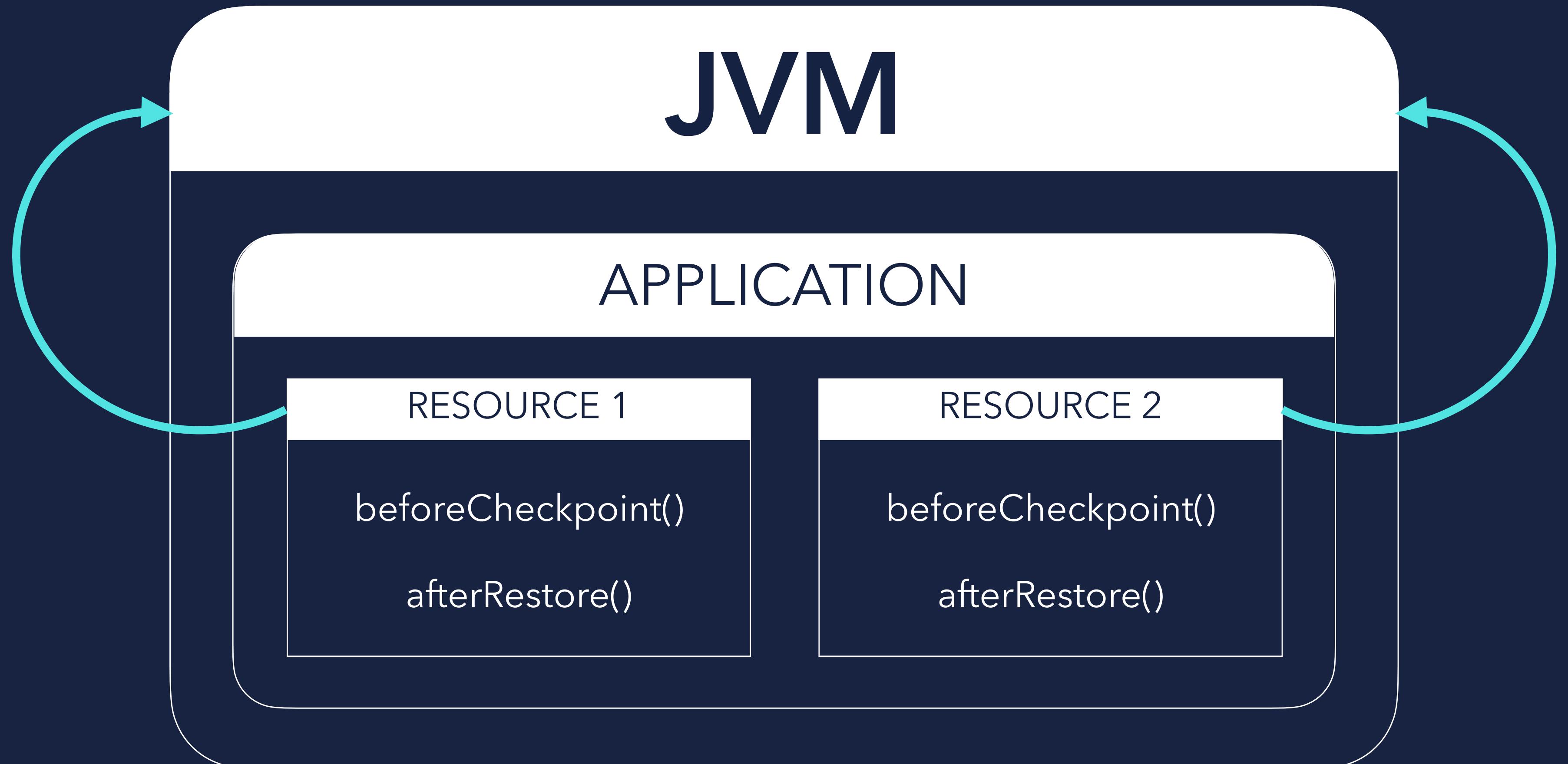


JVM startup...

CRaC OVERVIEW

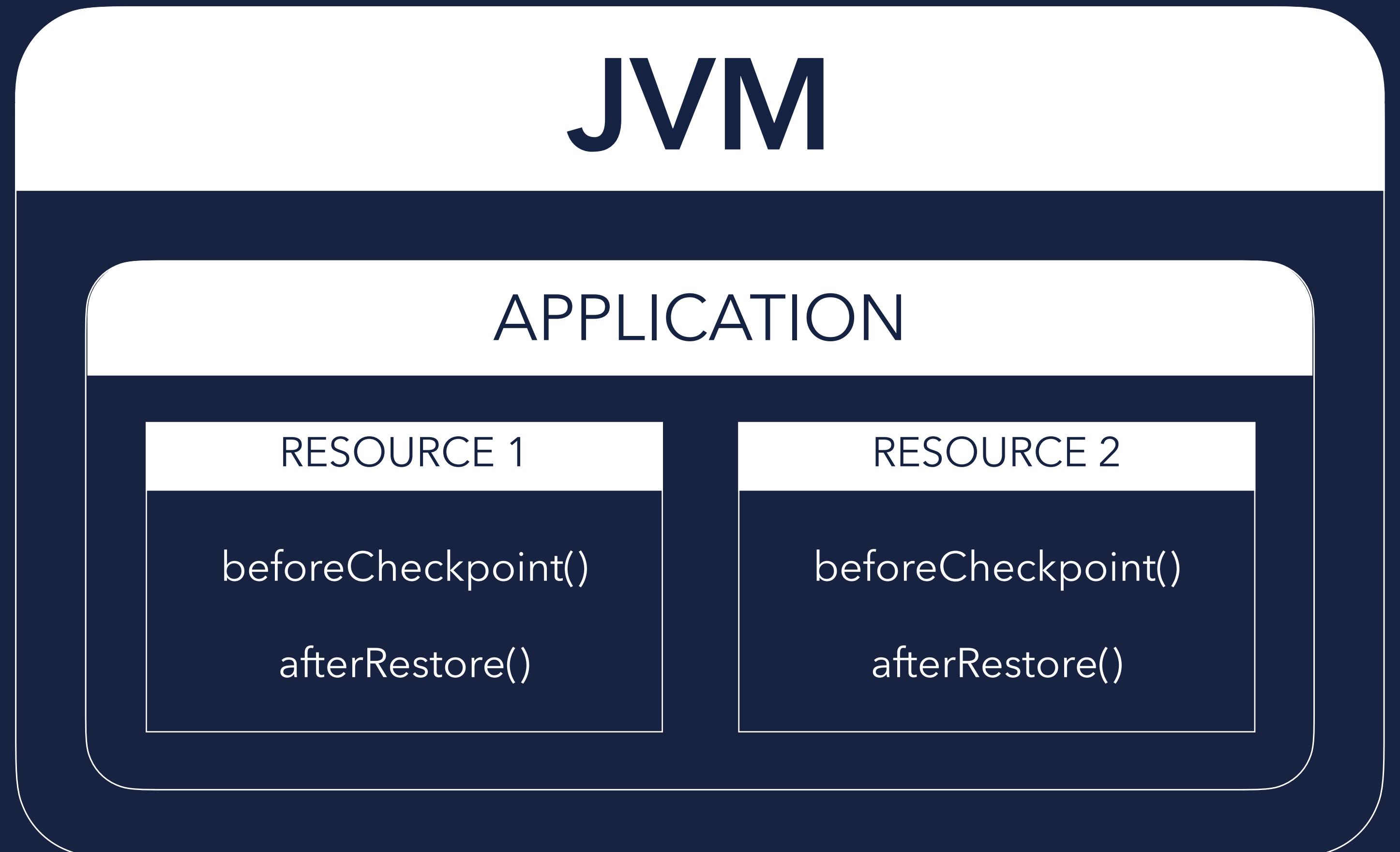


CRaC OVERVIEW



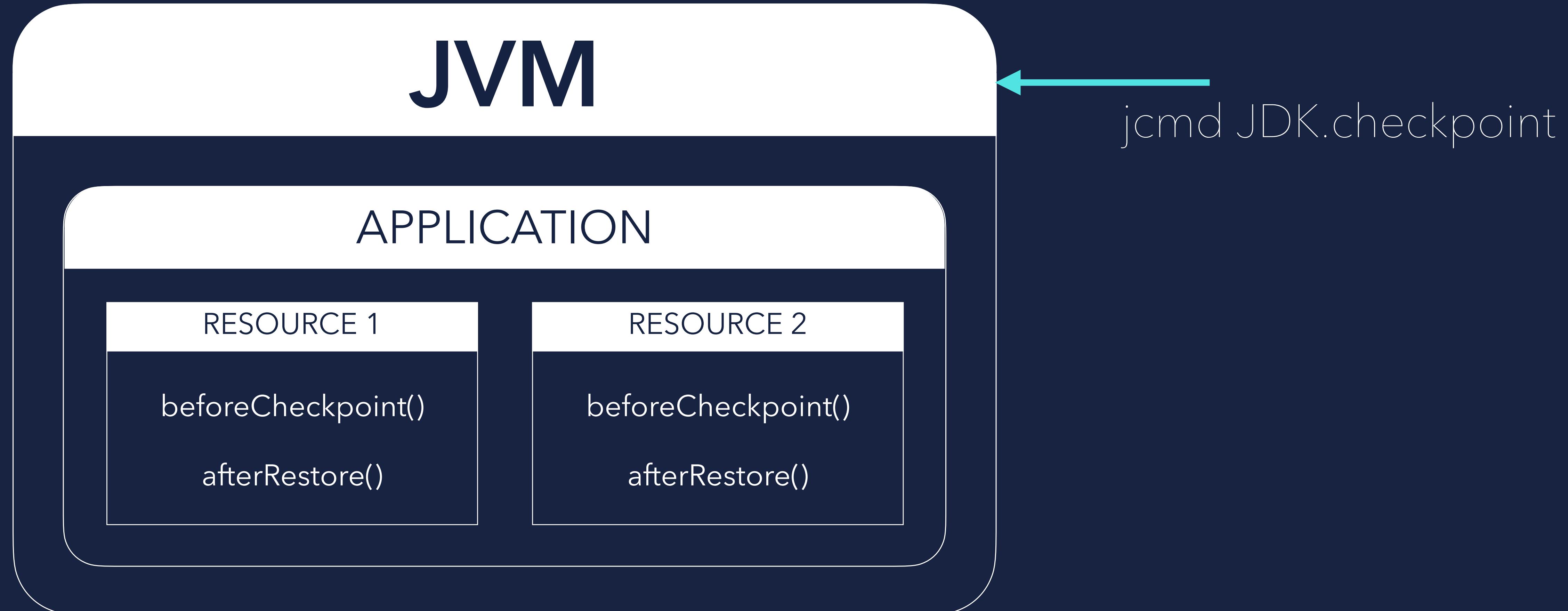
Register resources
in global context

CRaC OVERVIEW

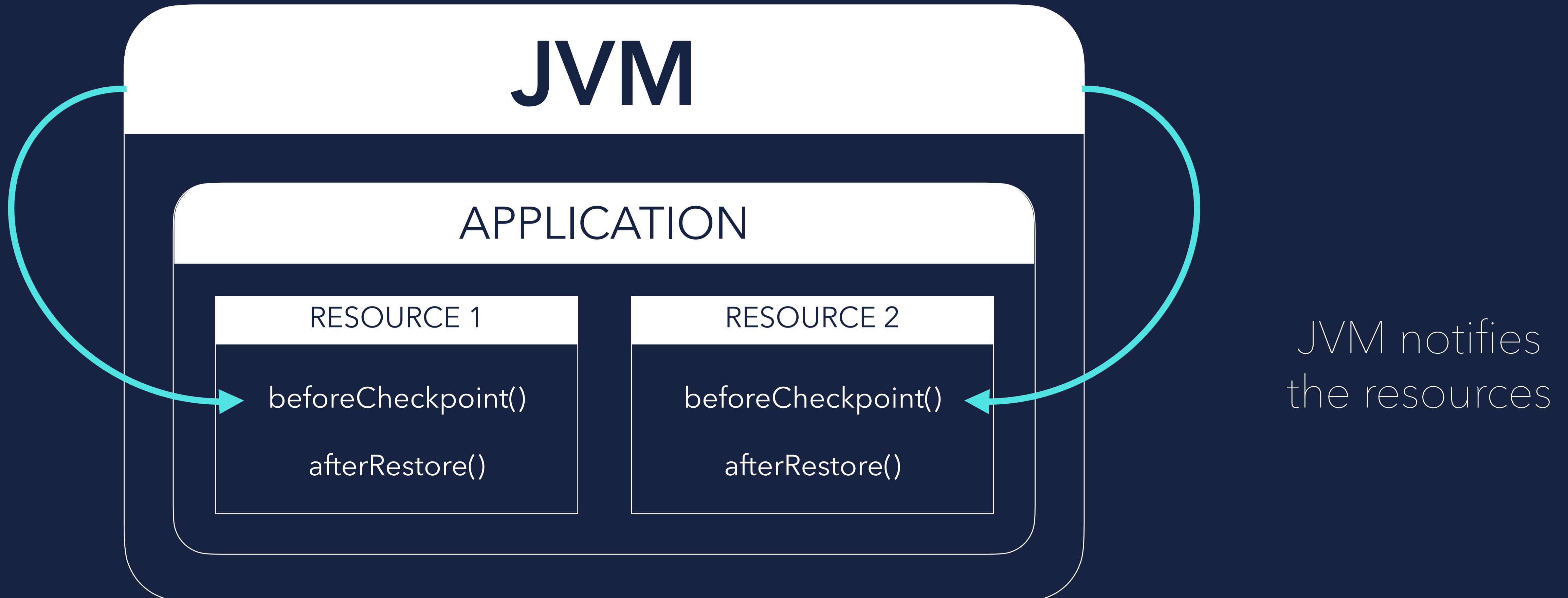


Warmup the application

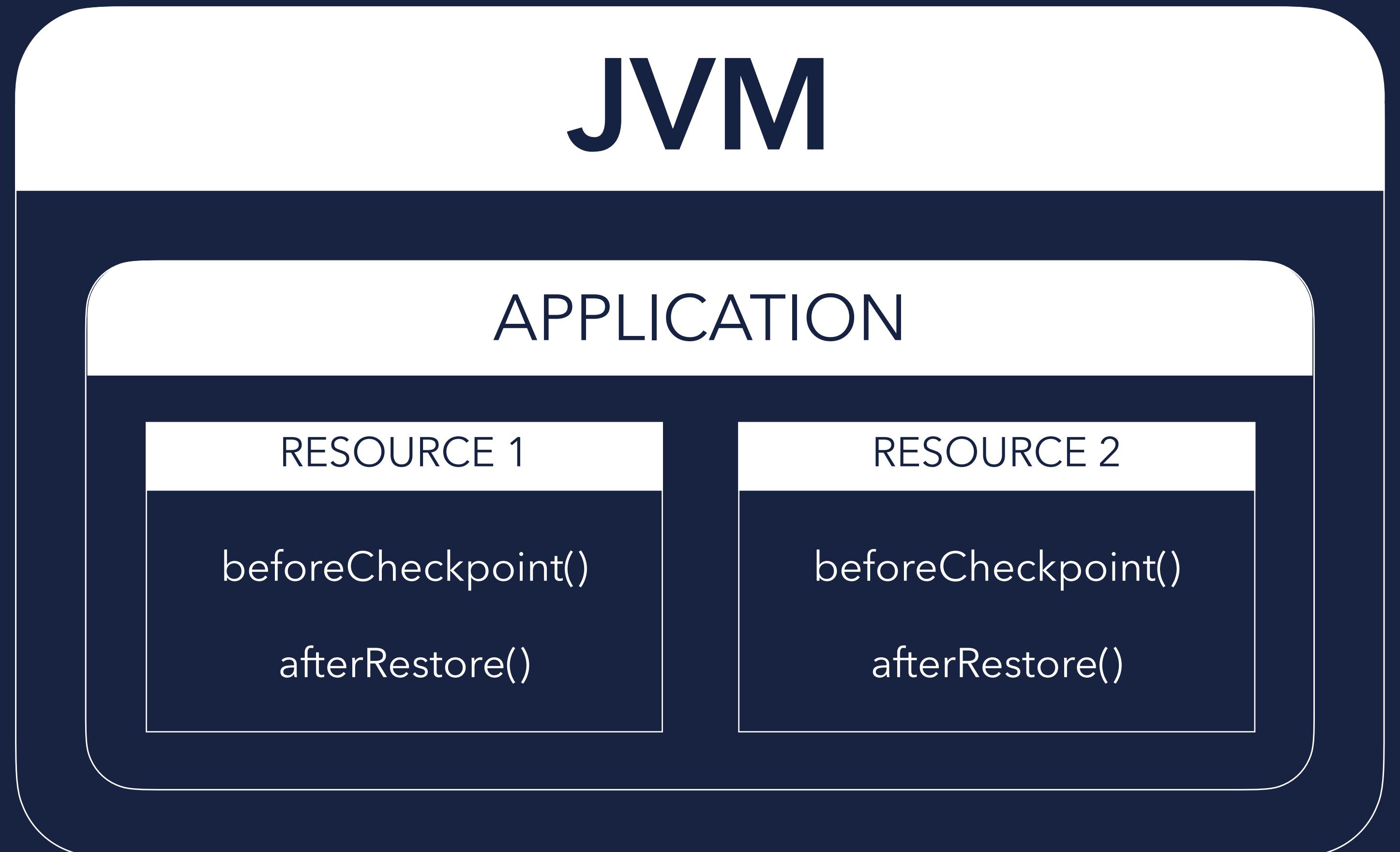
CRaC OVERVIEW



CRaC OVERVIEW

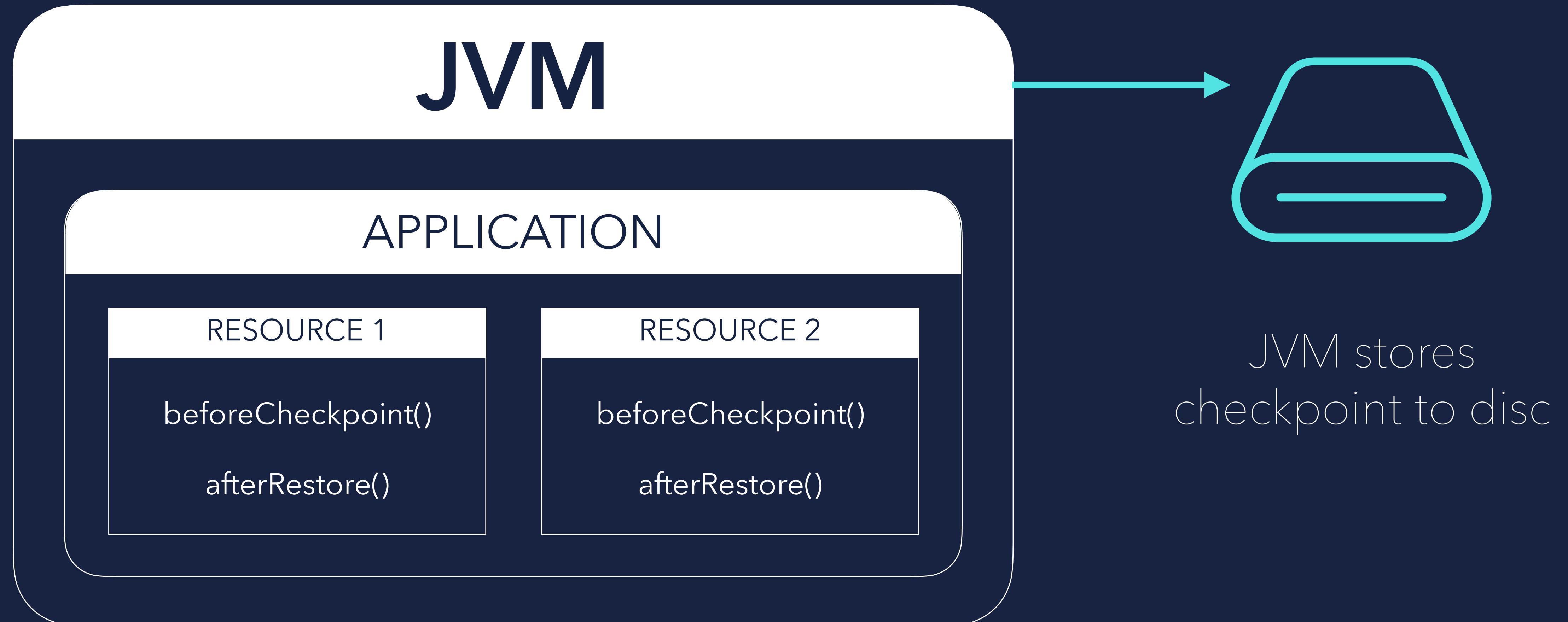


CRaC OVERVIEW

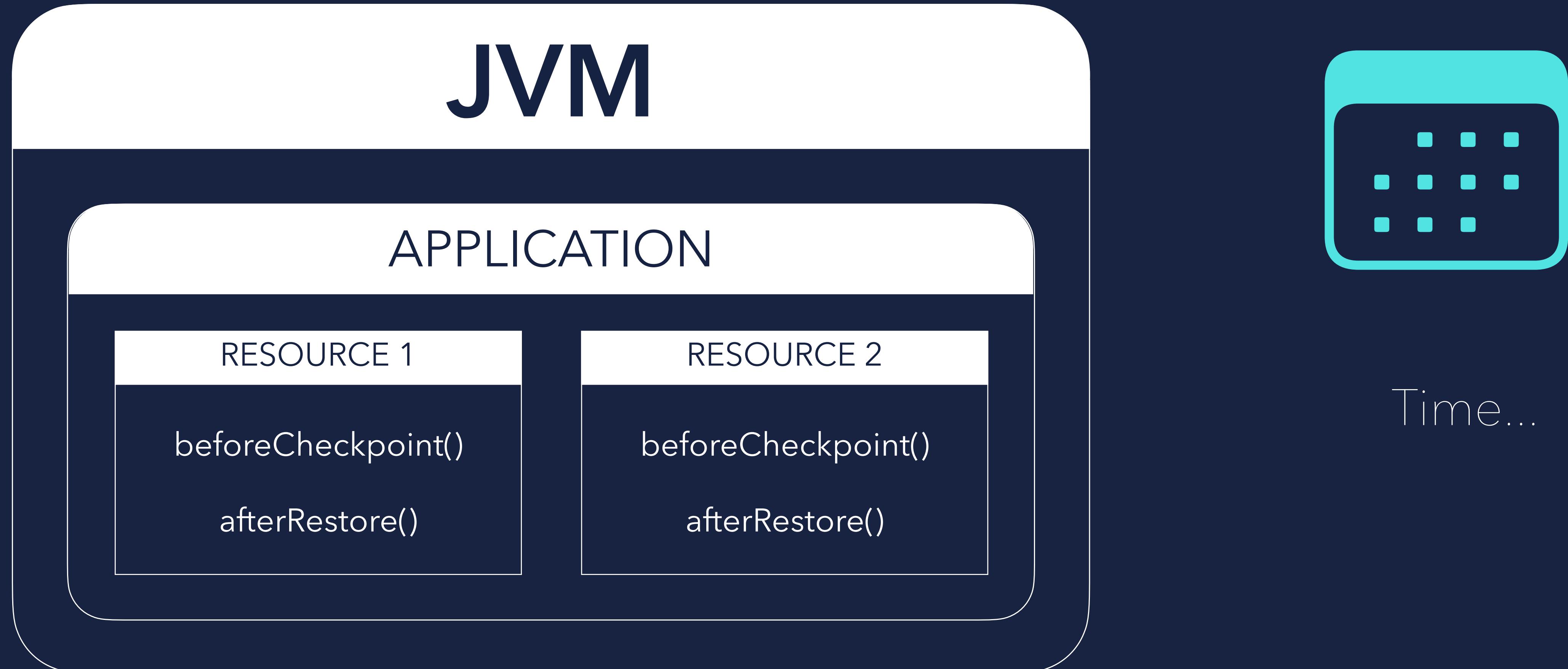


Application closes
open resources

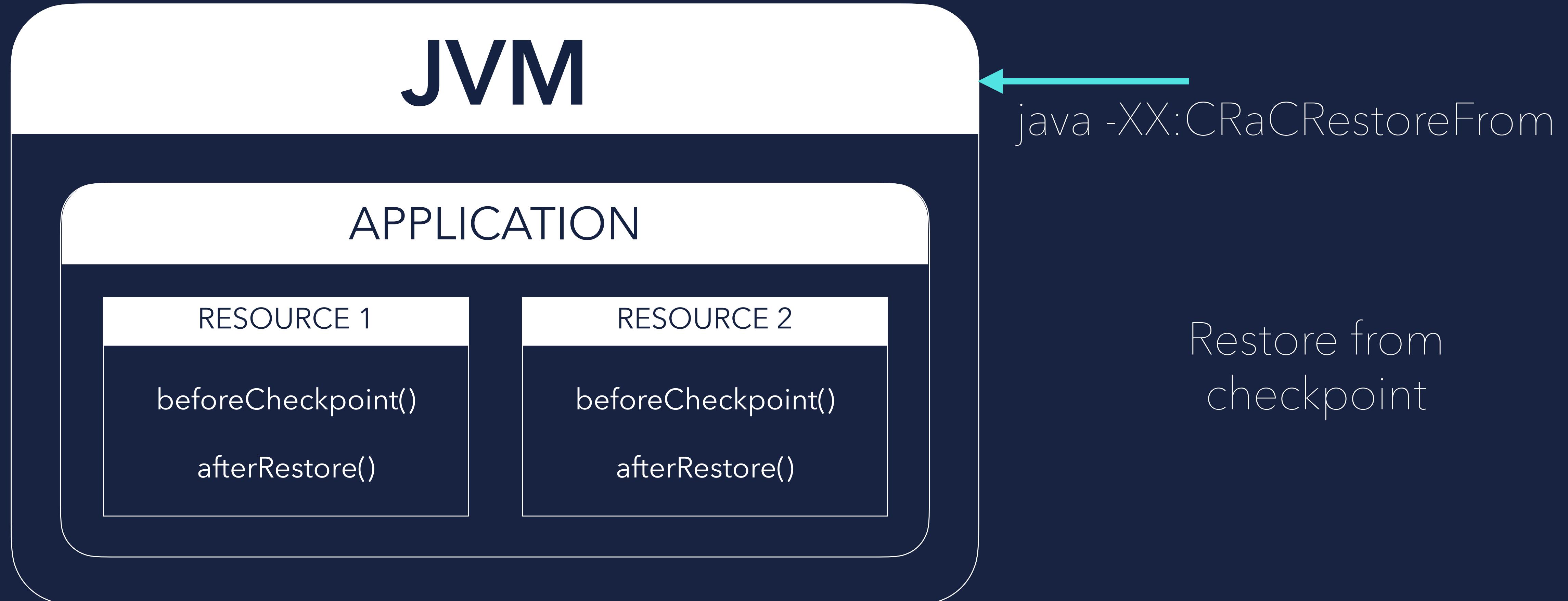
CRaC OVERVIEW



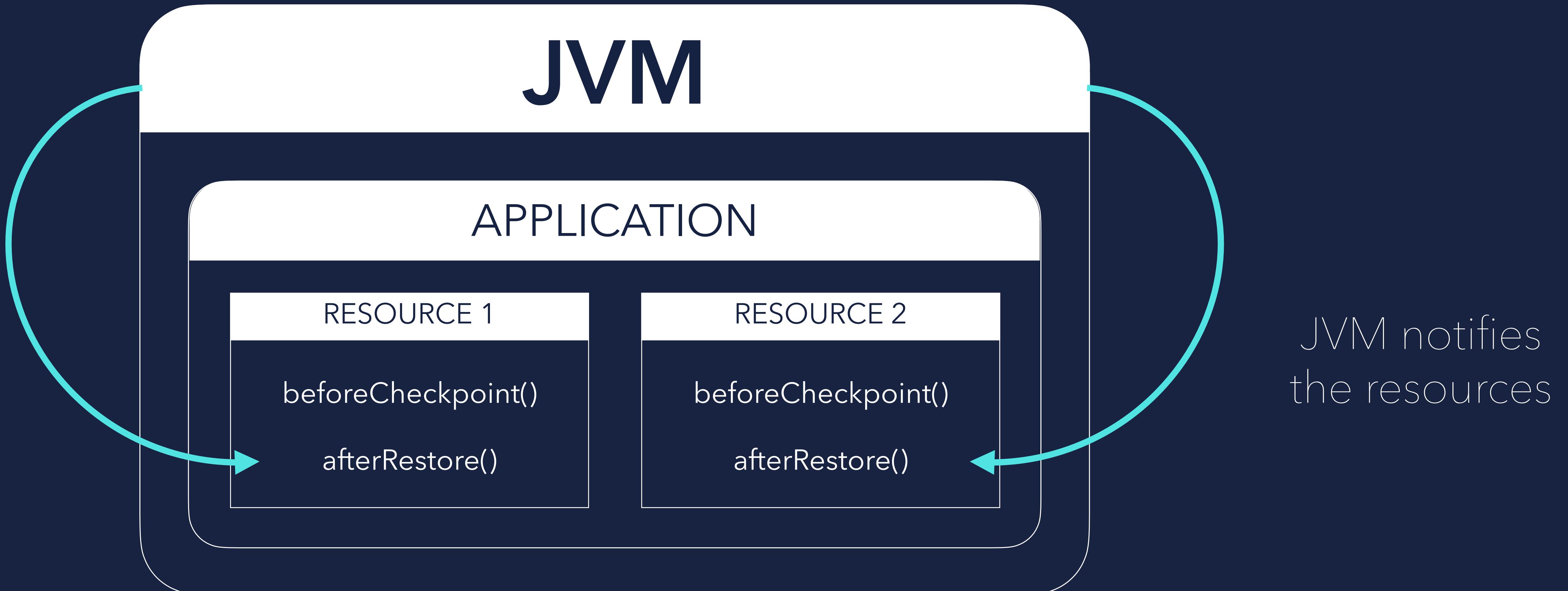
CRaC OVERVIEW



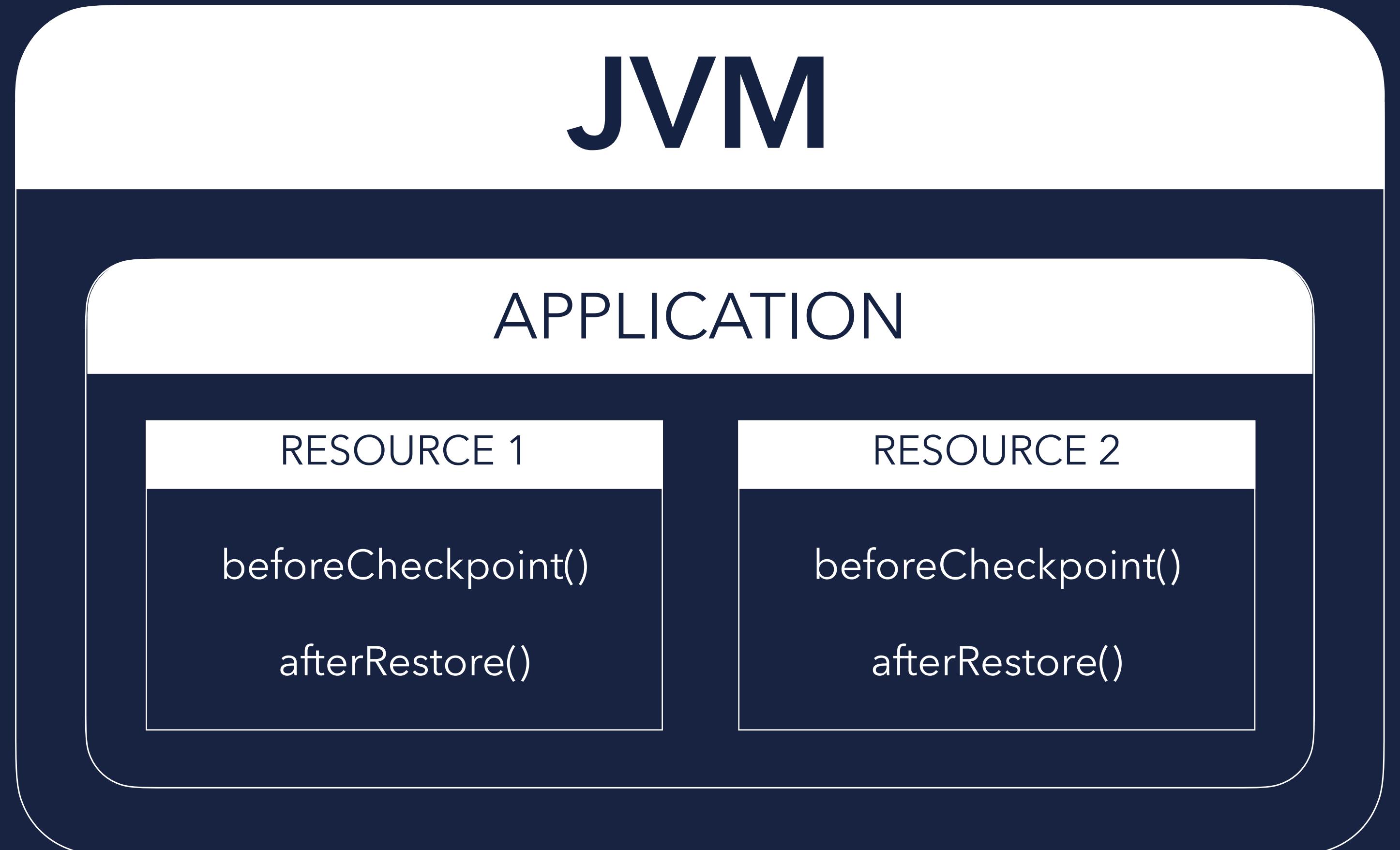
CRaC OVERVIEW



CRaC OVERVIEW

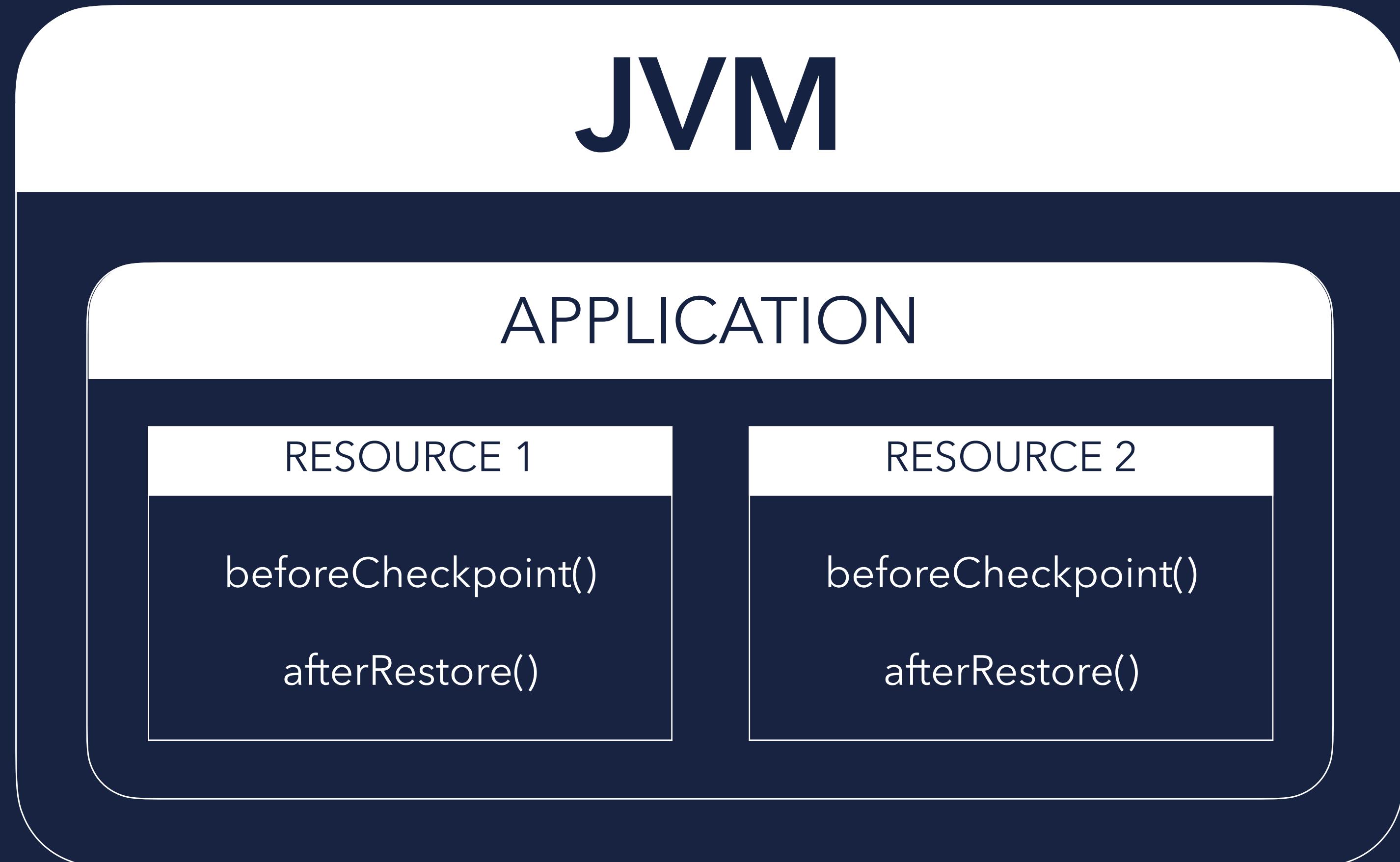


CRaC OVERVIEW



Application
re-open resources

CRaC OVERVIEW



No JVM startup and
no application warmup !!!

TYPIICAL

USAGE

TYPICAL USAGE...

- Run app in a docker container

TYPICAL USAGE...

- Run app in a docker container
- Create checkpoint (store in container or external volume)

TYPICAL USAGE...

- Run app in a docker container
- Create checkpoint (store in container or external volume)
- Commit the state of container (only if checkpoint in container)

TYPICAL USAGE...

- Run app in a docker container
- Create checkpoint (store in container or external volume)
- Commit the state of container (only if checkpoint in container)
- Start the container (point jvm to container or external volume)

LINUX ONLY
X64 / ARCH64

WINDOWS

MACOS?

ORG.CRAC

ORG.CRAC

- Designed to provide smooth CRaC adoption

ORG.CRAC

- Designed to provide smooth CRaC adoption
- Total mirror of jdk.crac api at compile-time

ORG.CRAC

- Designed to provide smooth CRaC adoption
- Total mirror of jdk.crac api at compile-time
- Can be used with any OpenJDK implementation

ORG.CRAC

- Designed to provide smooth CRaC adoption
- Total mirror of jdk.crac api at compile-time
- Can be used with any OpenJDK implementation
- Detects CRaC implementation at runtime

ORG.CRAC

- Designed to provide smooth CRaC adoption
- Total mirror of jdk.crac api at compile-time
- Can be used with any OpenJDK implementation
- Detects CRaC implementation at runtime
- No CRaC support -> won't call CRaC specific code

ORG.CRAC

- Designed to provide smooth CRaC adoption
- Total mirror of jdk.crac api at compile-time
- Can be used with any OpenJDK implementation
- Detects CRaC implementation at runtime
- No CRaC support -> won't call CRaC specific code
- CRaC support -> will forward all CRaC specific calls to jdk.crac

ORG.CRAC



```
implementation 'org.crac:crac:1.4.0'
```

Maven

```
<dependency>
  <groupId>org.crac</groupId>
  <artifactId>crac</artifactId>
  <version>1.4.0</version>
</dependency>
```

ORG.CRAC

github.com/CRaC/org.crac

azul



FRAMEWORK SUPPORT?

FRAMEWORK SUPPORT?

• Quarkus (rudimentary support)

FRAMEWORK SUPPORT?

- Quarkus (rudimentary support)
- Micronaut (good support)

FRAMEWORK SUPPORT?

- Quarkus (rudimentary support)
- Micronaut (good support)
- Spring 6.1+ / SpringBoot 3.2+ (full support)

DEMO



SPRINGBOOT 3.2
PETCLINIC

NORMAL

START

NORMAL START

```
> java -jar spring-petclinic-3.2.0.jar
```

START APPLICATION

NORMAL START

```
> java -jar spring-petclinic-3.2.0.jar
```



```
:: Built with Spring Boot :: 3.2.0
```

```
...
```

```
2023-11-29T11:57:27.579+01:00  INFO 3839 --- [           main] o.s.d.j.r.query.QueryEnhancerFactory      : Hibernate is in classpath; If applicable, HQL parser will be used.  
2023-11-29T11:57:28.549+01:00  INFO 3839 --- [           main] o.s.b.a.e.web.EndpointLinksResolver        : Exposing 13 endpoint(s) beneath base path '/actuator'  
2023-11-29T11:57:28.625+01:00  INFO 3839 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer    : Tomcat started on port 8080 (http)  
with context path ''  
2023-11-29T11:57:28.639+01:00  INFO 3839 --- [           main] o.s.s.petclinic.PetClinicApplication       : Started PetClinicApplication in 4.619 seconds (process running for 5.051)  
Started up in 4099ms with PID: 3839
```

START FROM

AUTO

CHECKPOINT

AUTO CHECKPOINT

- Feature in SpringBoot 3.2

AUTO CHECKPOINT

- Feature in SpringBoot 3.2
- Start with -Dspring.context.checkpoint=onRefresh

AUTO CHECKPOINT

- Feature in SpringBoot 3.2
- Start with -Dspring.context.checkpoint=onRefresh
- Creates automatic checkpoint after start of SpringBoot framework

AUTO CHECKPOINT

- Feature in SpringBoot 3.2
- Start with -Dspring.context.checkpoint=onRefresh
- Creates automatic checkpoint after start of SpringBoot framework
- Right before the application will be started

AUTO CHECKPOINT

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

START APPLICATION AND CREATE CHECKPOINT

AUTO CHECKPOINT

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

> java -XX:CRaCRestoreFrom=./tmp_auto_checkpoint

2023-11-29T12:01:37.698+01:00  WARN 15261 --- [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool      : HikariPool-1 - Thread starvation or clock leap detected (housekeeper delta=1h26m17s198ms377μs333ns).
2023-11-29T12:01:37.790+01:00  INFO 15261 --- [           main] o.s.c.support.DefaultLifecycleProcessor : Restarting Spring-managed lifecycle beans after JVM restore
2023-11-29T12:01:37.811+01:00  INFO 15261 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http)
with context path ''
2023-11-29T12:01:37.834+01:00  INFO 15261 --- [           main] o.s.s.petclinic.PetClinicApplication   : Restored PetClinicApplication in 0.956 seconds (process running for 0.958)
Started up in 697ms with PID: 15261
```

RESTORE FROM CHECKPOINT

START FROM

MANUAL

CHECKPOINT

MANUAL CHECKPOINT

- Start application with -xx:CracCheckpointTo=Path

MANUAL CHECKPOINT

- Start application with -xx:CracCheckpointTo=Path
- Warm up your application

MANUAL CHECKPOINT

- Start application with -xx:CracCheckpointTo=Path
- Warm up your application
- Create checkpoint using jcmd

MANUAL CHECKPOINT

- Start application with -xx:CracCheckpointTo=Path
- Warm up your application
- Create checkpoint using jcmd
- Checkpoint now also contains application

MANUAL CHECKPOINT

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

START APPLICATION

MANUAL CHECKPOINT

```
> java -XX:CRaCCheckpointTo=./tmp_manual_checkpoint -jar spring-petclinic-3.2.0.jar  
...  
2023-11-29T11:57:28.625+01:00  INFO 3839 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http)  
with context path ''  
2023-11-29T11:57:28.639+01:00  INFO 3839 --- [           main] o.s.s.petclinic.PetClinicApplication      : Started PetClinicApplication in  
4.619 seconds (process running for 5.051)  
Started up in 4099ms with PID: 3839
```

```
> jcmd 3839 JDK.checkpoint
```

CREATE CHECKPOINT

MANUAL CHECKPOINT

```
> java -XX:CRaCRestoreFrom=./tmp_manual_checkpoint
```

RESTORE FROM CHECKPOINT

MANUAL CHECKPOINT

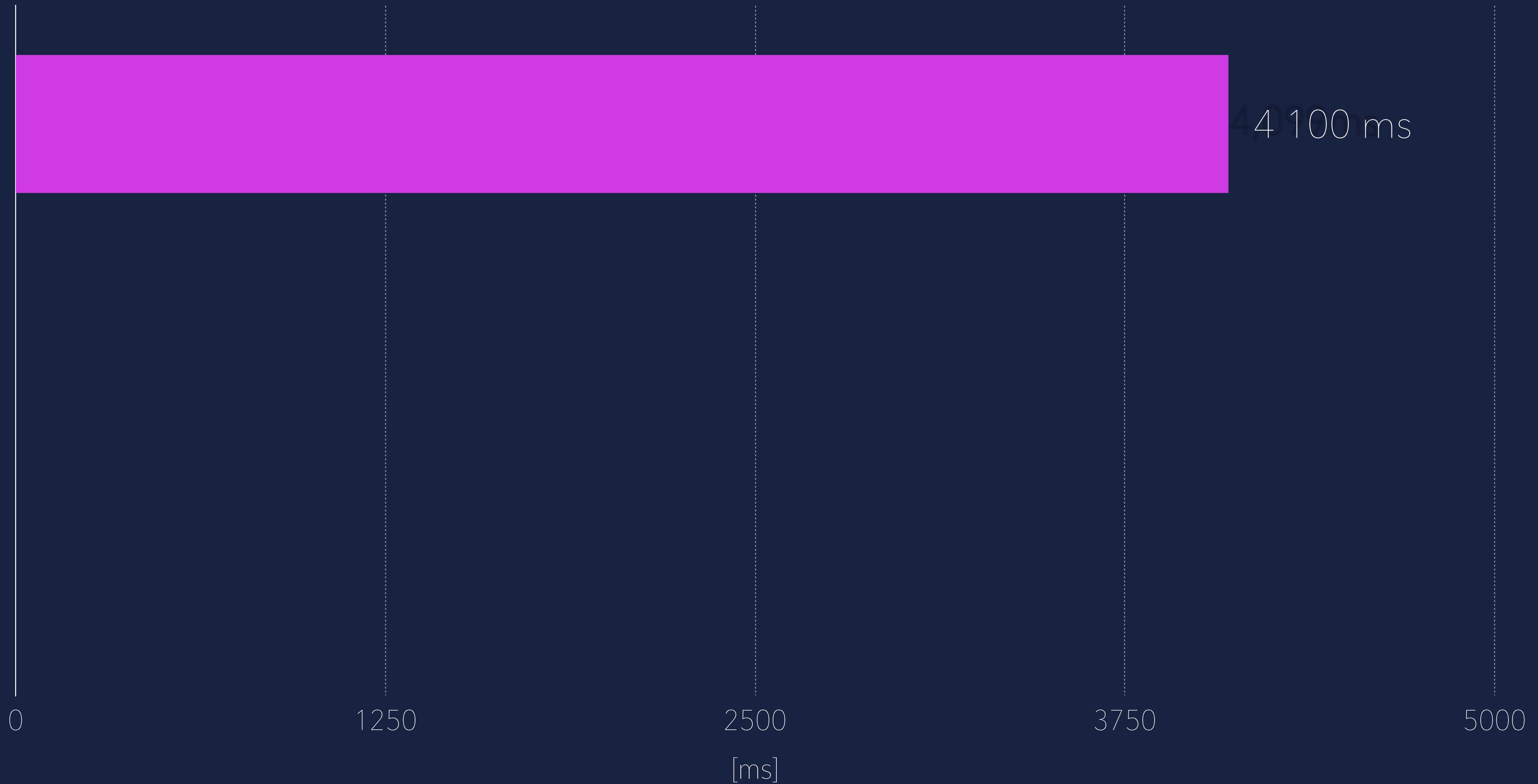
```
> java -XX:CRaCRestoreFrom=./tmp_manual_checkpoint
```

```
2023-11-29T12:04:32.626+01:00  WARN 15512 --- [l-1 housekeeper] com.zaxxer.hikari.pool.HikariPool      : HikariPool-1 - Thread starvation or clock leap detected (housekeeper delta=1h28m32s17ms487μs256ns).  
2023-11-29T12:04:32.634+01:00  INFO 15512 --- [Attach Listener] o.s.c.support.DefaultLifecycleProcessor : Restarting Spring-managed lifecycle beans after JVM restore  
2023-11-29T12:04:32.642+01:00  INFO 15512 --- [Attach Listener] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path ''  
2023-11-29T12:04:32.644+01:00  INFO 15512 --- [Attach Listener] o.s.c.support.DefaultLifecycleProcessor : Spring-managed lifecycle restart completed (restored JVM running for 301 ms)
```

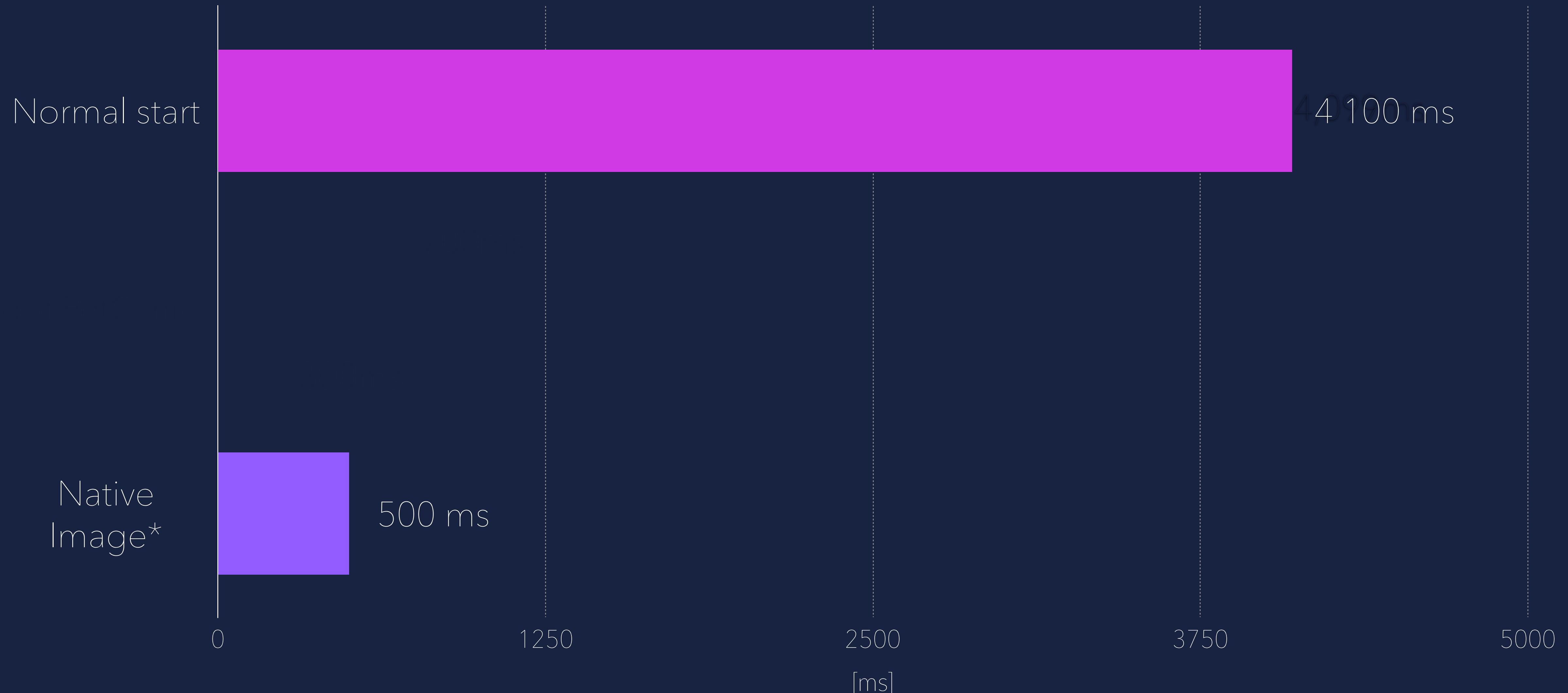
SpringBoot 3.2 PetClinic Demo

Normal start

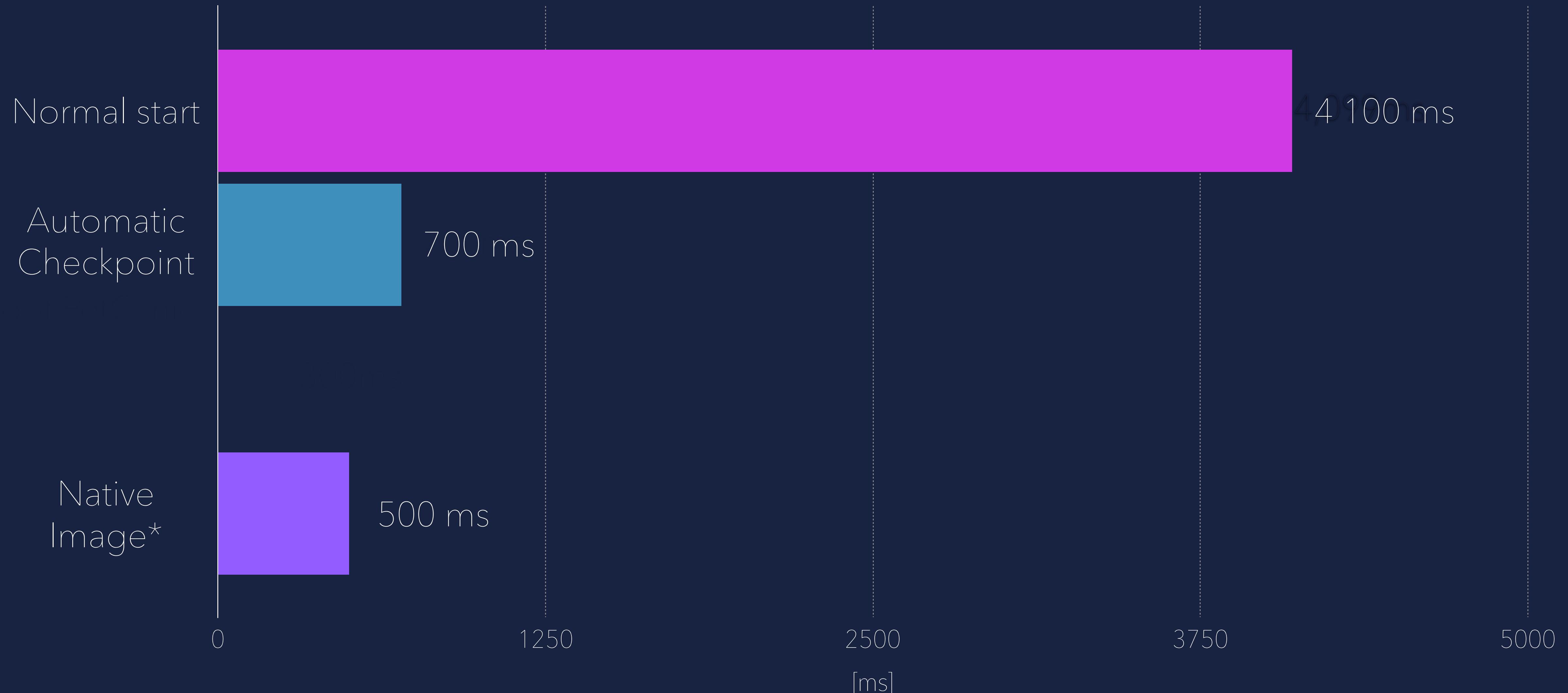
449100ms



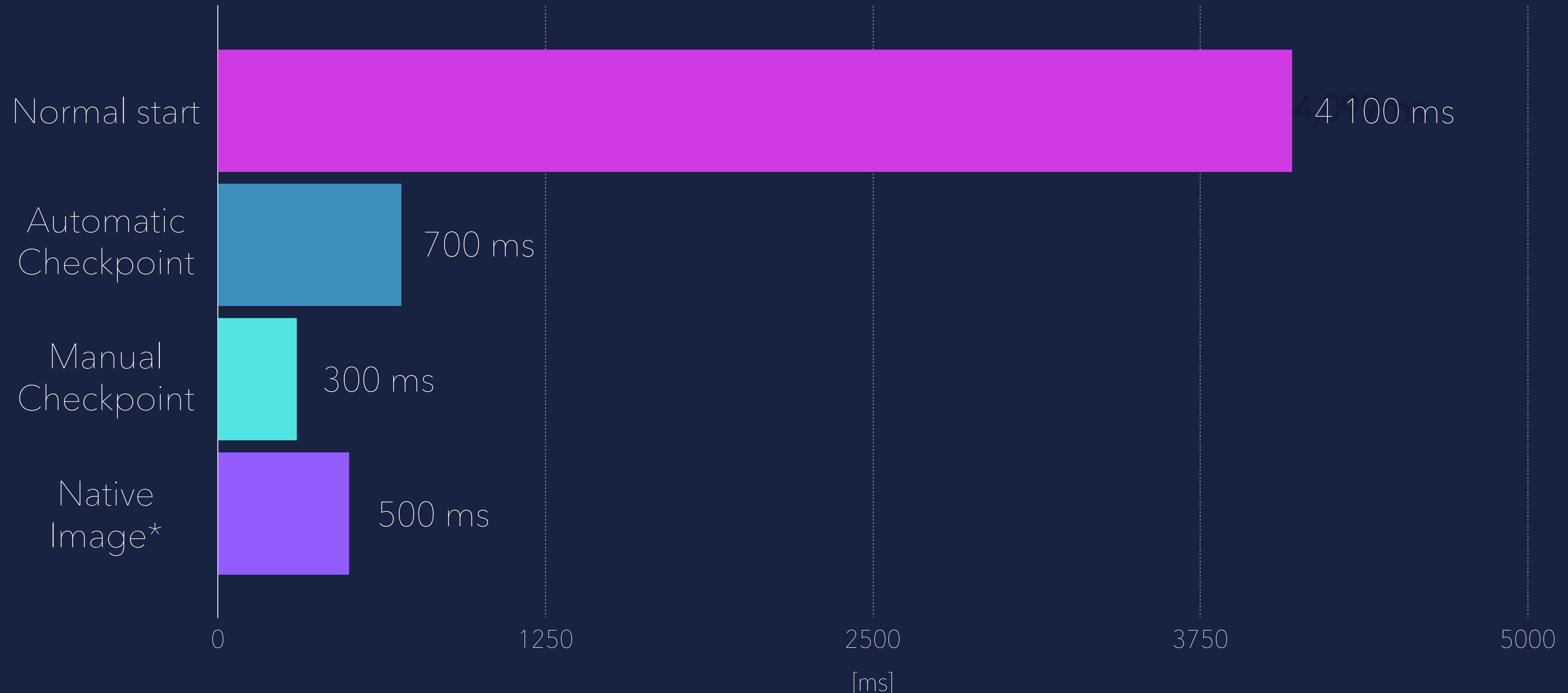
SpringBoot 3.2 PetClinic Demo



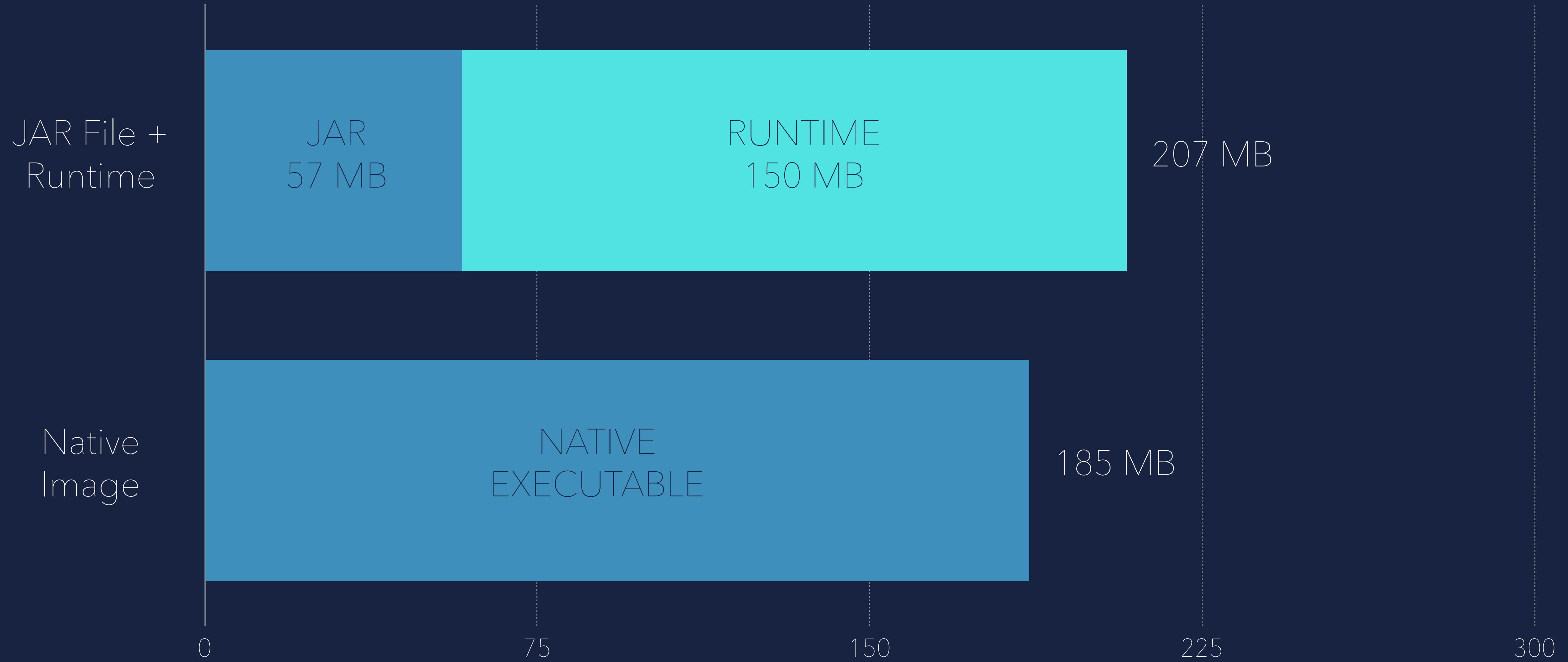
SpringBoot 3.2 PetClinic Demo



SpringBoot 3.2 PetClinic Demo



Memory Footprint (deployment artifact)



SUMMARY...

SUMMARY

...

- CRaC is a way to pause and restore a JVM based application

SUMMARY

...

- CRaC is a way to pause and restore a JVM based application
- It doesn't require a closed world as with a native image

SUMMARY

...

- CRaC is a way to pause and restore a JVM based application
- It doesn't require a closed world as with a native image
- Extremely fast time to full performance level

SUMMARY

...

- CRaC is a way to pause and restore a JVM based application
- It doesn't require a closed world as with a native image
- Extremely fast time to full performance level
- No need for hotspot identification, method compiles, recompiles and deoptimisations

SUMMARY . . .

- CRaC is a way to pause and restore a JVM based application
- It doesn't require a closed world as with a native image
- Extremely fast time to full performance level
- No need for hotspot identification, method compiles, recompiles and deoptimisations
- Improved throughput from start

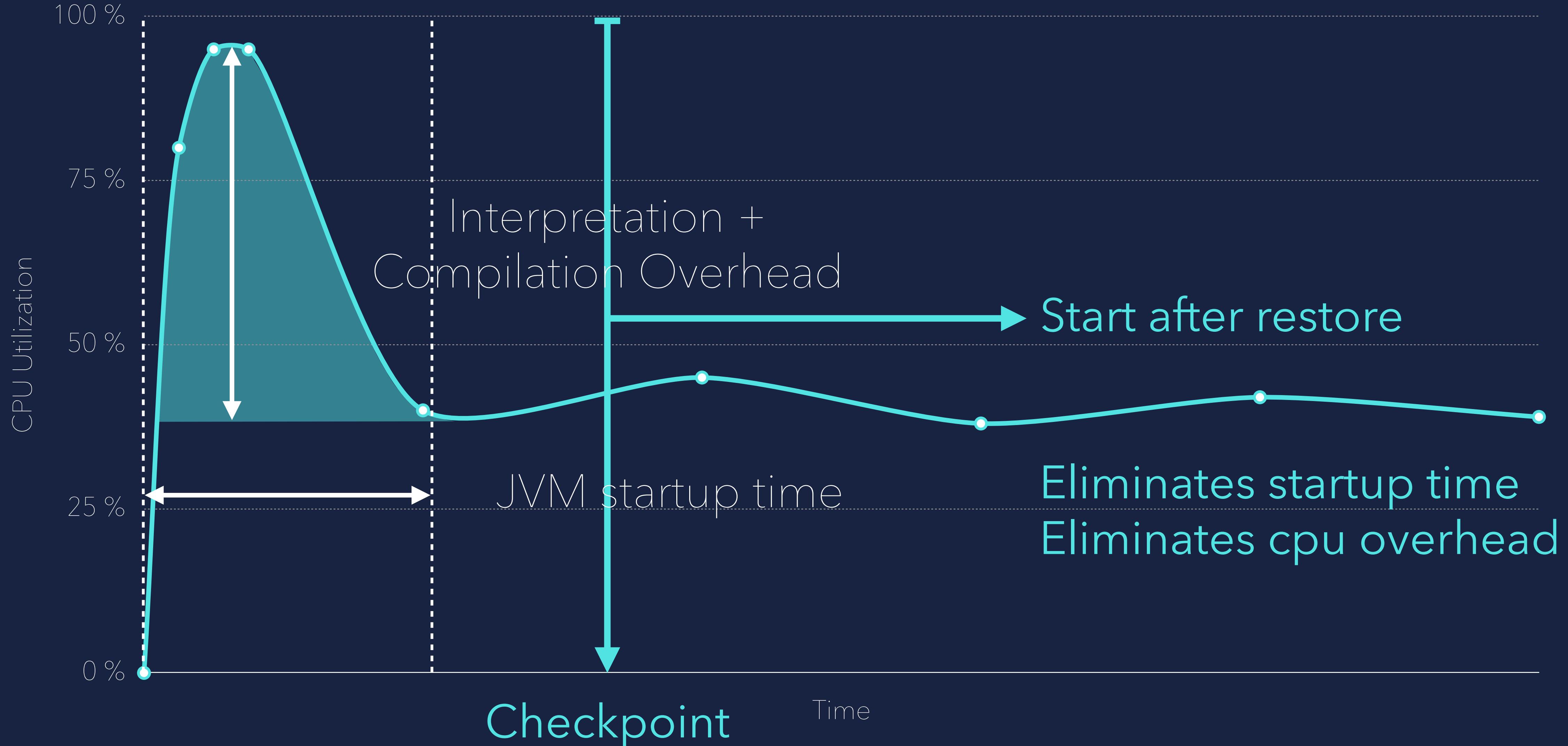
SUMMARY . . .

- CRaC is a way to pause and restore a JVM based application
- It doesn't require a closed world as with a native image
- Extremely fast time to full performance level
- No need for hotspot identification, method compiles, recompiles and deoptimisations
- Improved throughput from start
- CRaC is an OpenJDK project

SUMMARY . . .

- CRaC is a way to pause and restore a JVM based application
- It doesn't require a closed world as with a native image
- Extremely fast time to full performance level
- No need for hotspot identification, method compiles, recompiles and deoptimisations
- Improved throughput from start
- CRaC is an OpenJDK project
- CRaC can save infrastructure cost

INFRASTRUCTURE COST



WANNA

KNOW MORE ?

INFORMATION

ooo

github.com/CRaC



DOWNLOAD

...

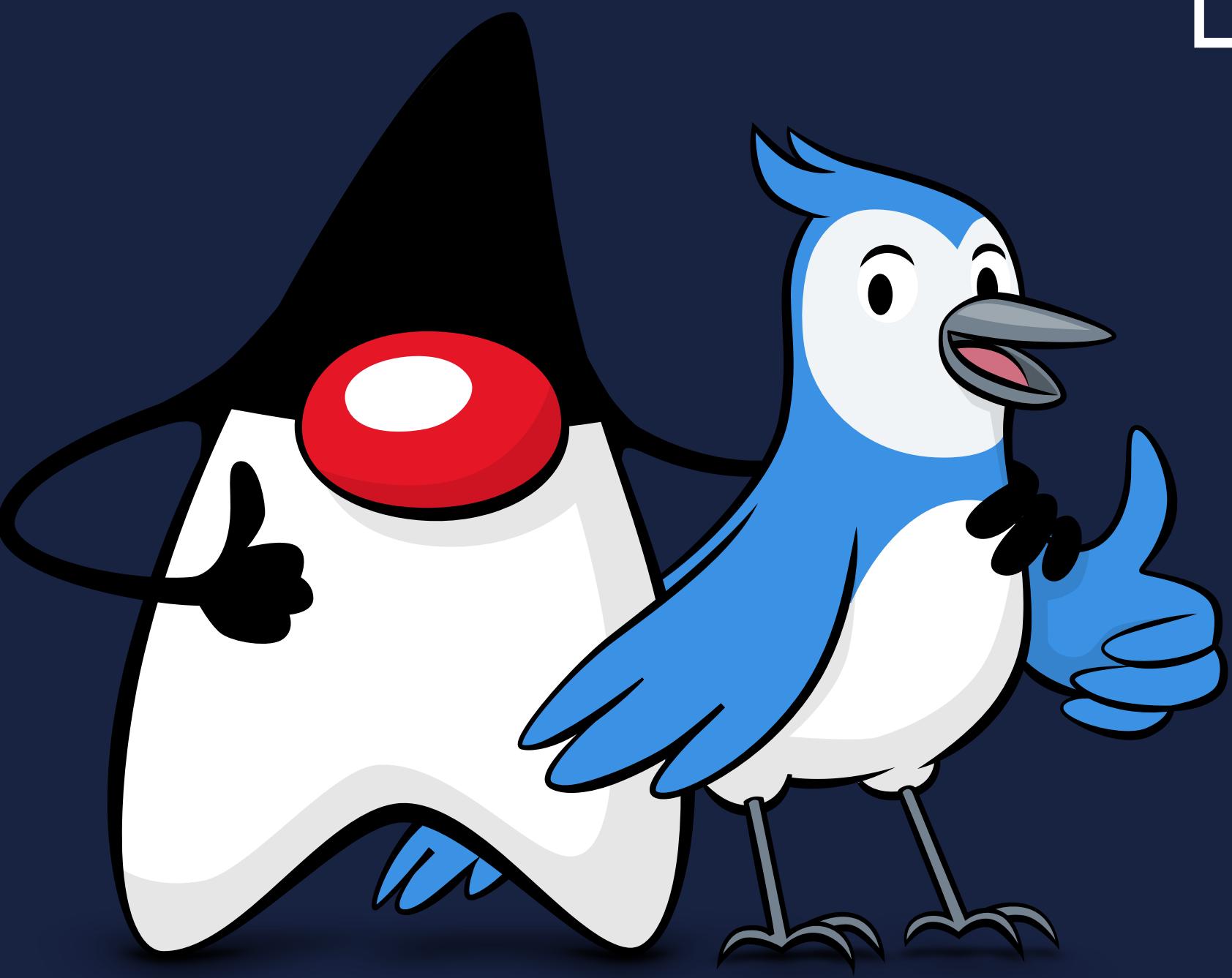
azul.com

JDK 17 / 21 LINUX X64 / AARCH64

azul



THANK



YOU