What the CRaC...

SUPERFAST JVM STARTUP

azul

ABOUTME.

Gerrit Grunwald | Developer Advocate | Azul | X @hansolo_

JAVA IS GREAT...

azul

VIBRANT COMMUNITY...

azul

HUNDREDS OF JUGs...

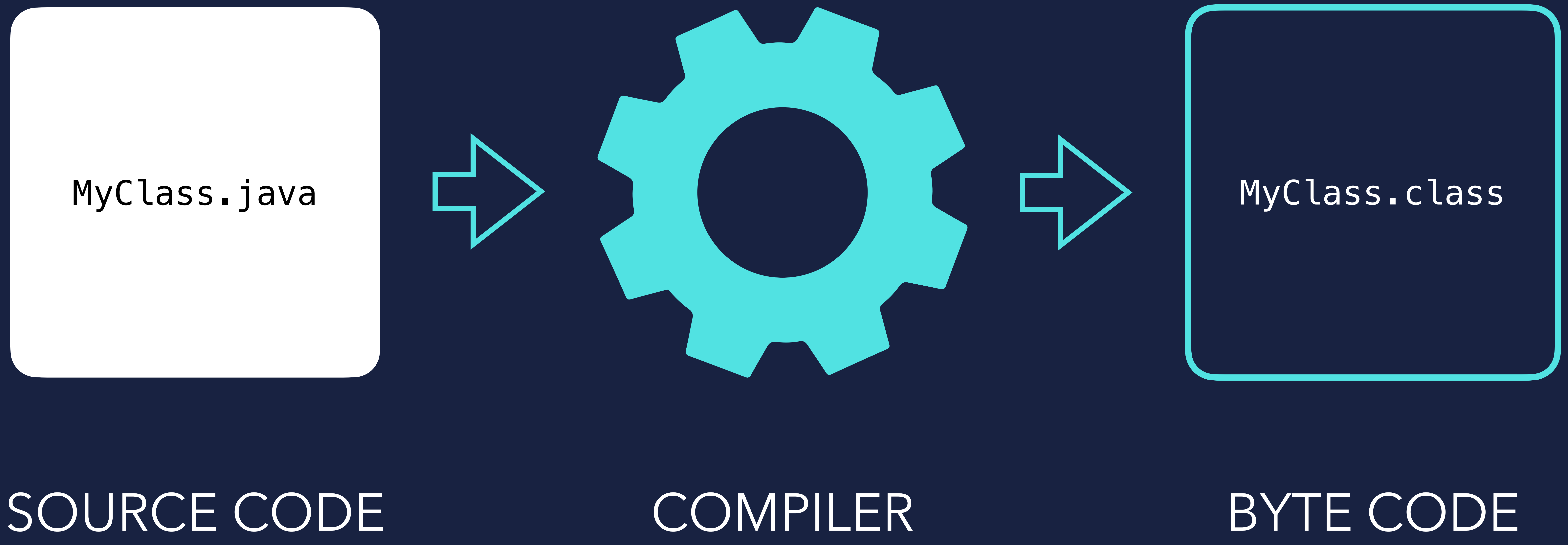
azul

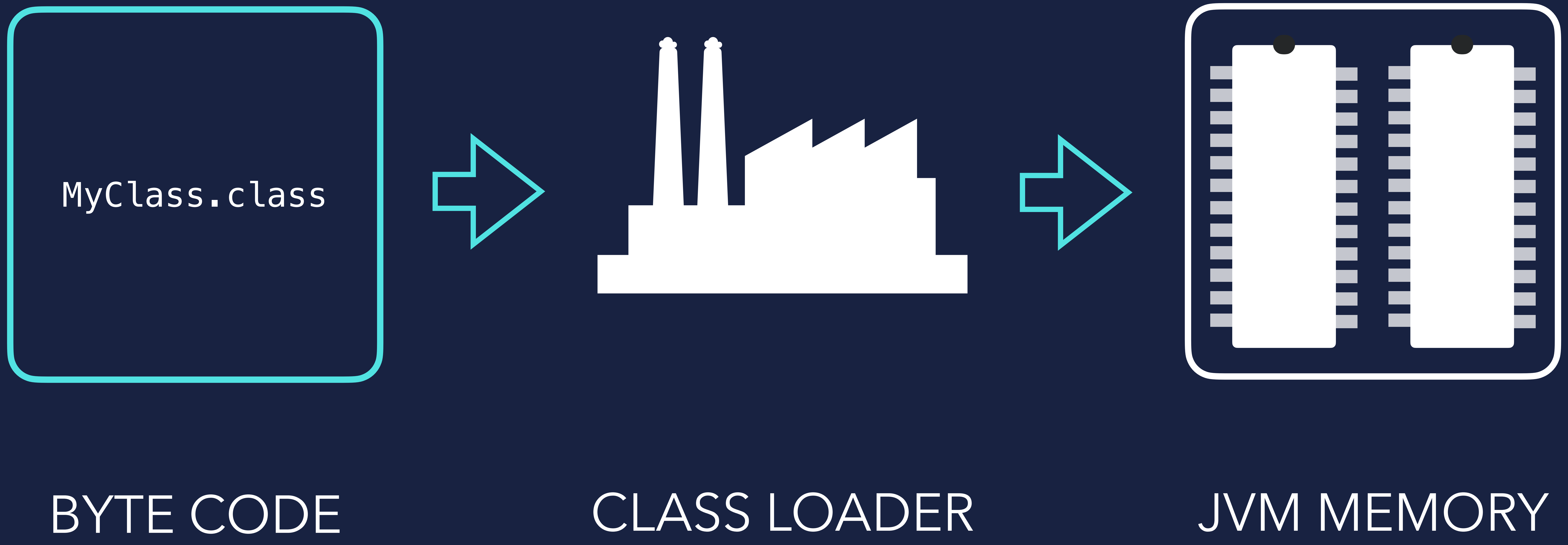
THOUSANDS OF FOSS PROJECTS...

azul

JAVA VIRTUAL MACHINE

HOW DOES IT WORK…

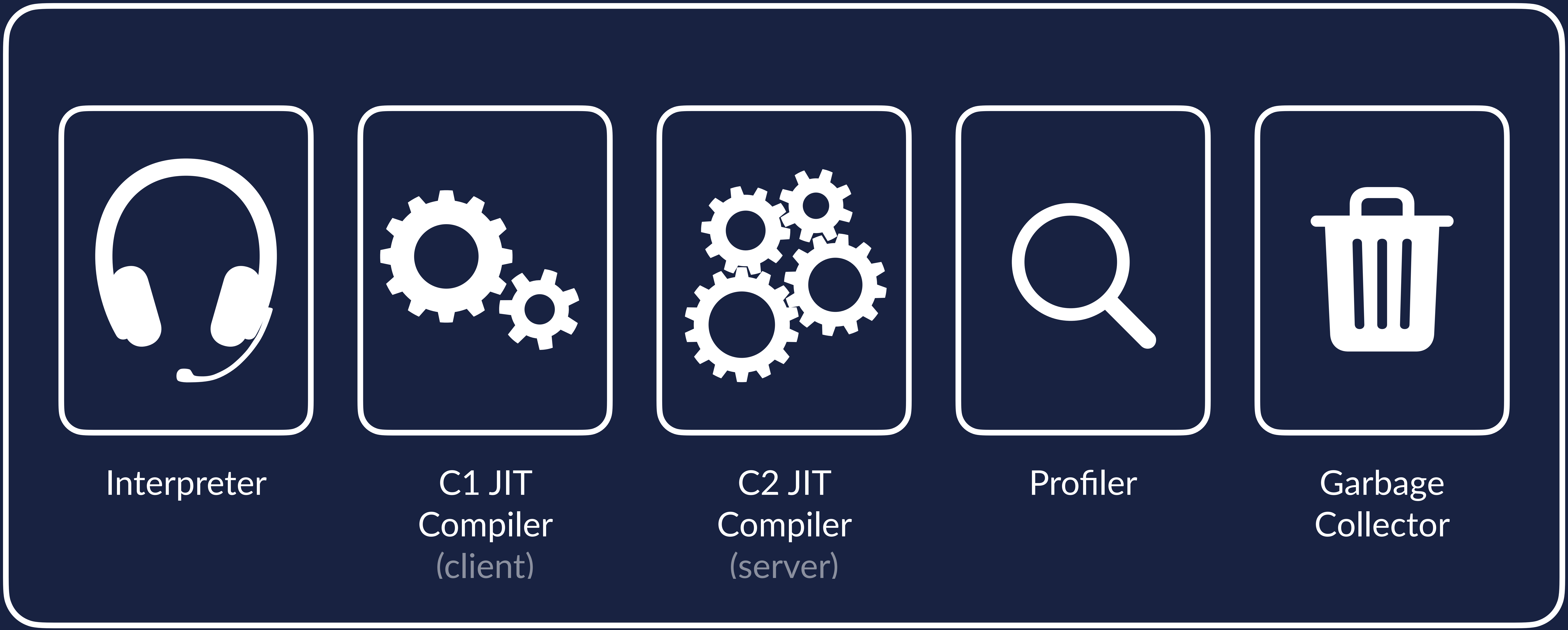

MyClass.java
MyClass.class
SOURCE CODE
COMPILER
BYTE CODE



azul

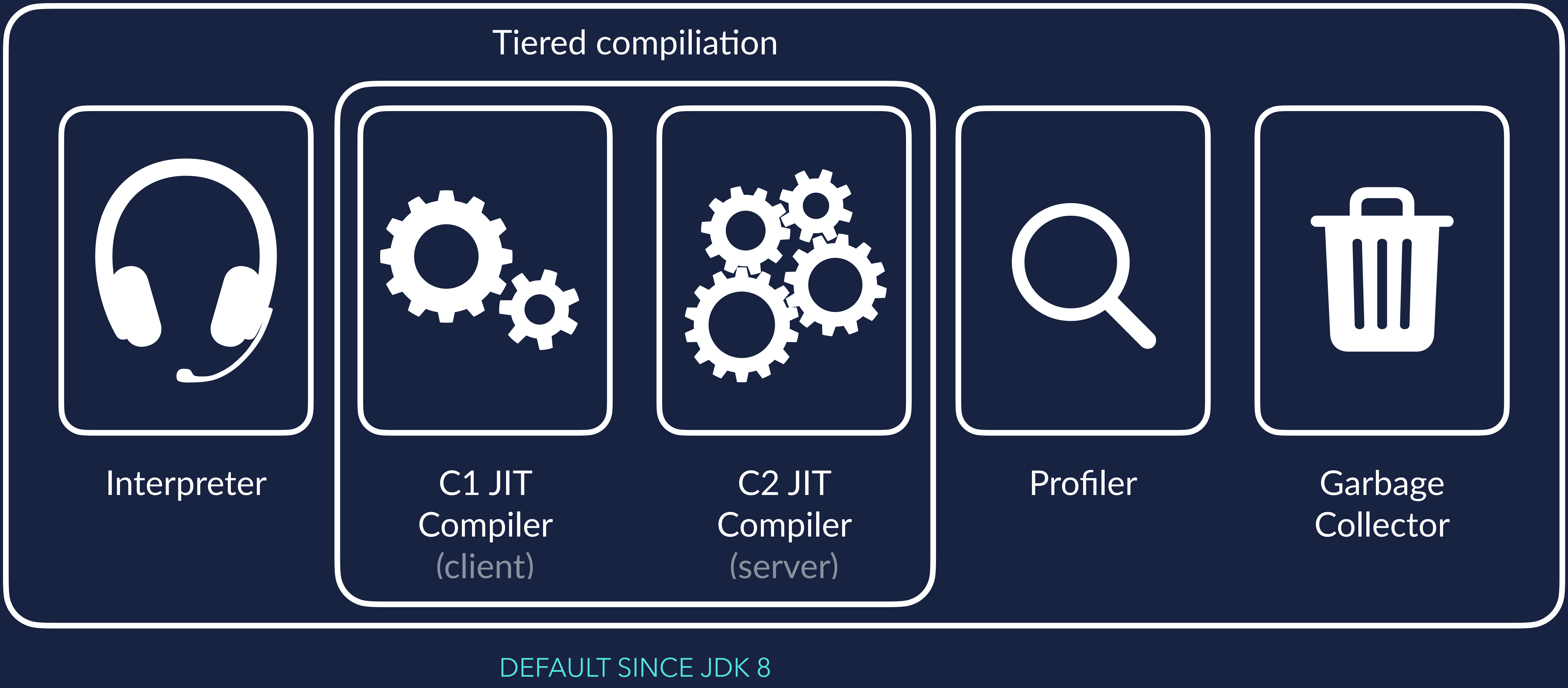

JVM MEMORY
EXECUTION ENGINE

EXECUTION ENGINE

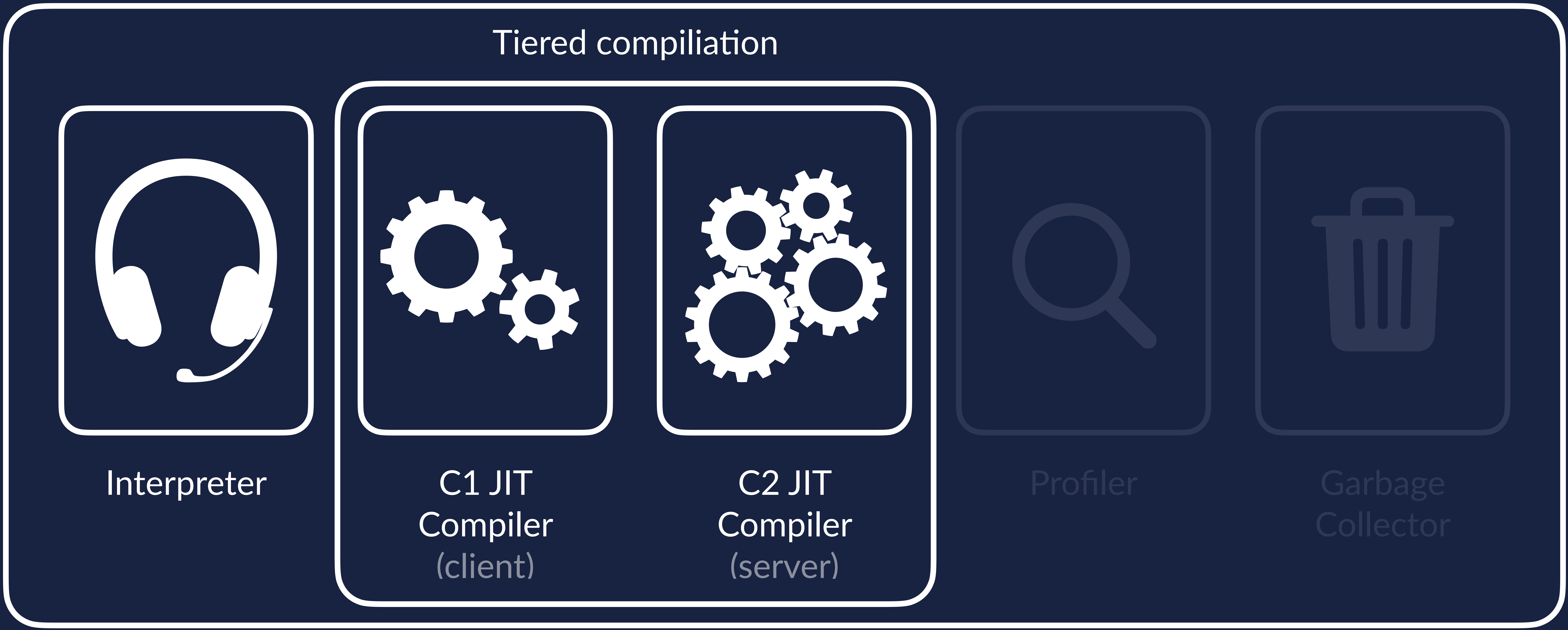


azul

EXECUTION ENGINE



EXECUTION ENGINE



DEFAULT SINCE JDK 8

Converts ByteCode into instruction set of CPU

INTERPRETER

THRESHOLD REACHED

(1000 in JDK 17)

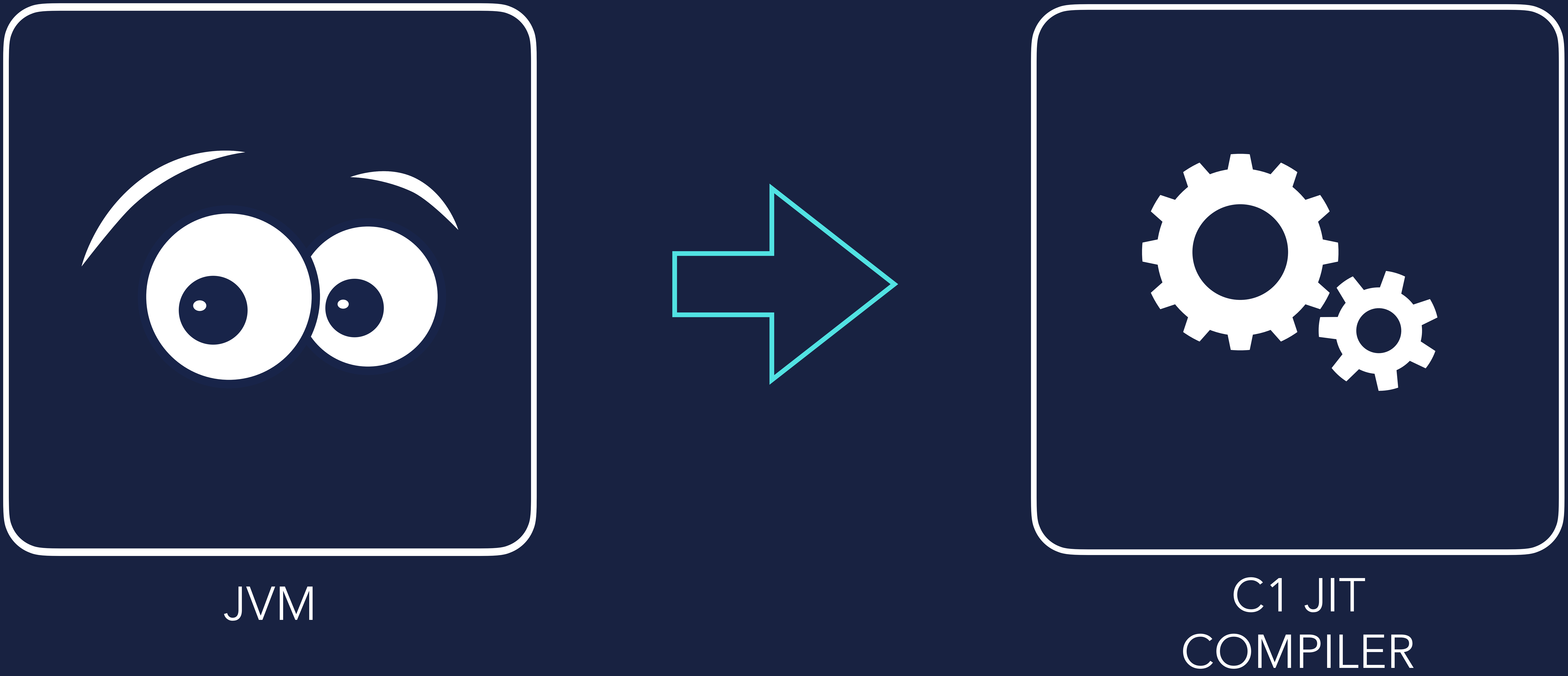
Detects hot spots by counting method calls and loop back edges

JVM

azul

Pass the "hot" code
to C1 JIT Compiler

Compiles code as quickly
as possible with low optimisation



Compiles code as quickly as possible with low optimisation

C1 JIT COMPILER

THRESHOLD REACHED

(5000 in JDK 17)

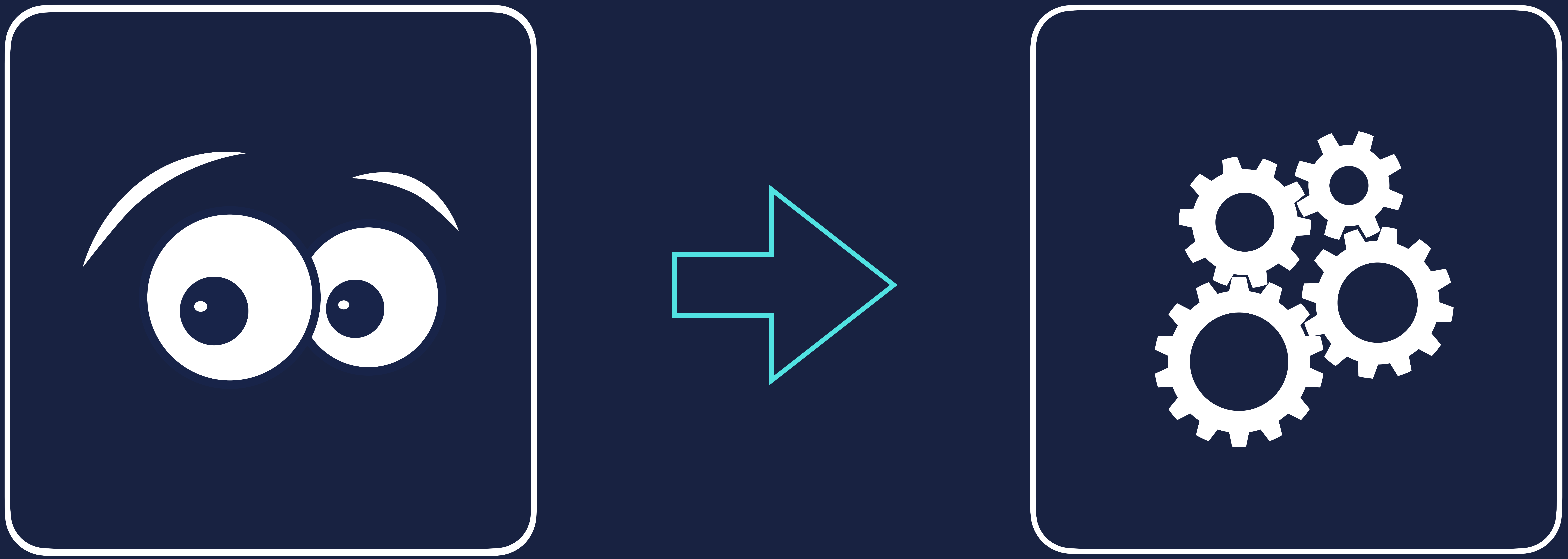
Profiles the running code (detecting hot code)

JVM

azul

Pass the "hot" code to C2 JIT Compiler

Compiles code with best optimisation possible (slower)



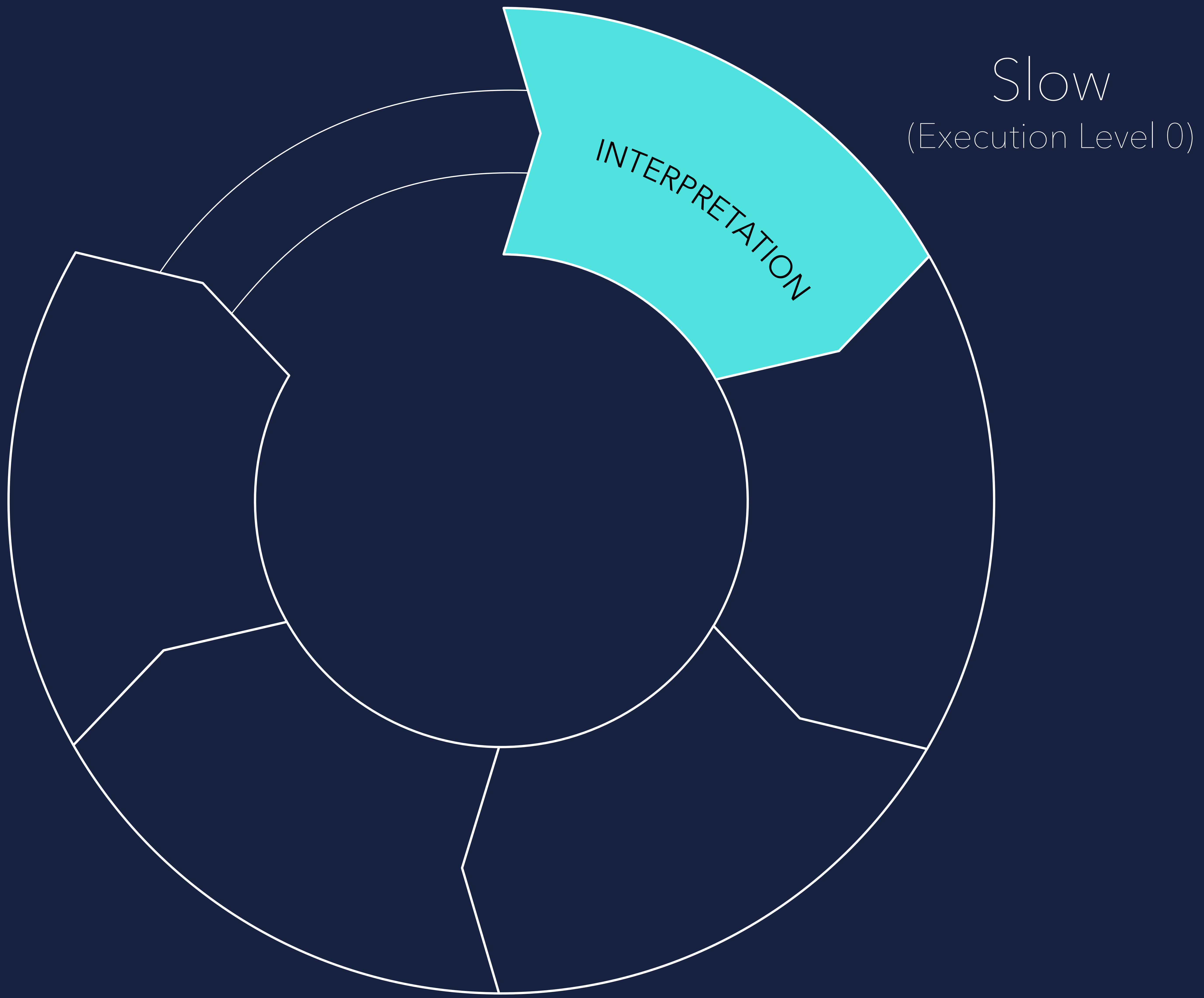
JVM

C2 JIT COMPILER

EXECUTION CYCLE

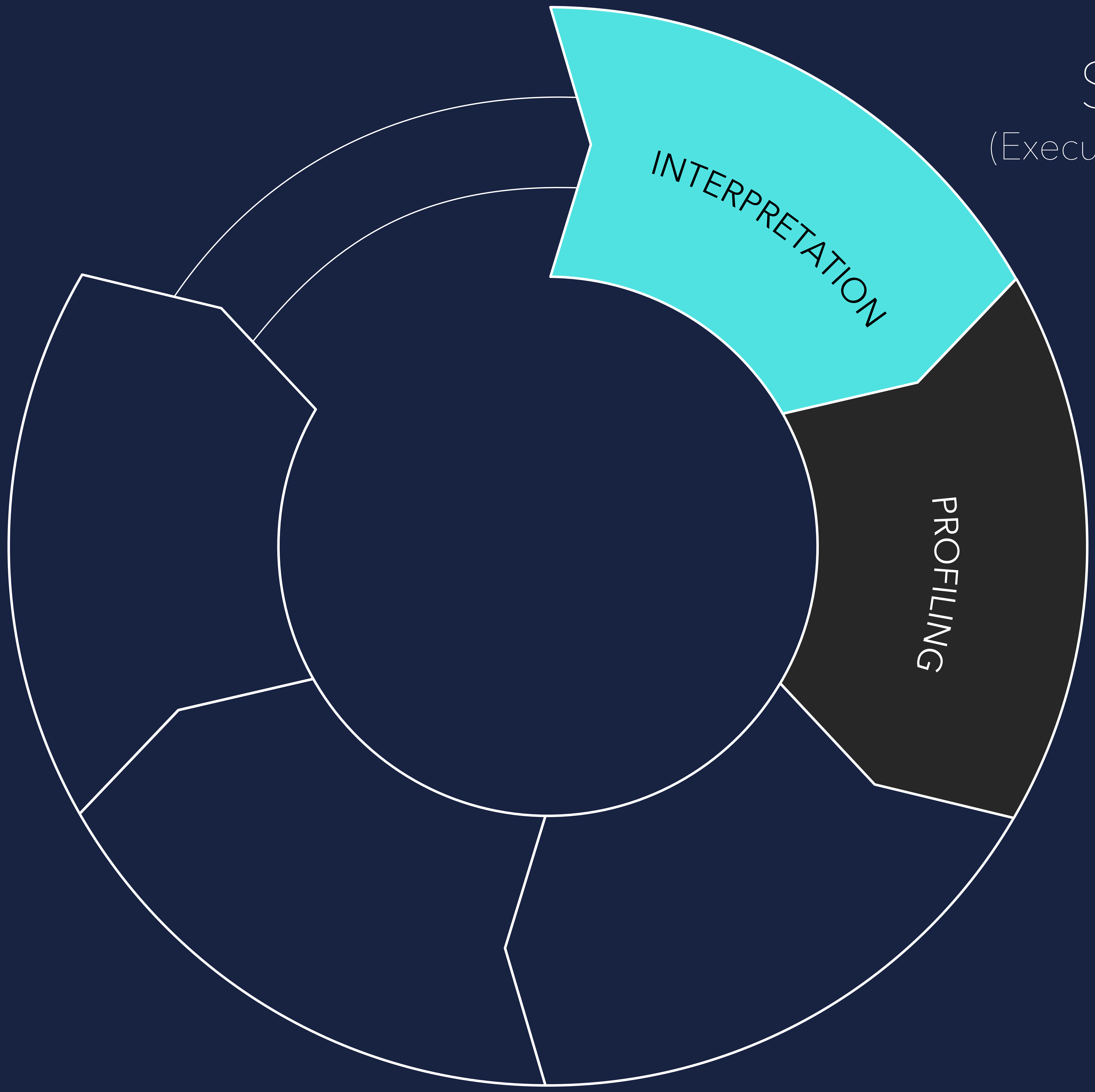
azul

EXECUTION CYCLE



azul

EXECUTION CYCLE

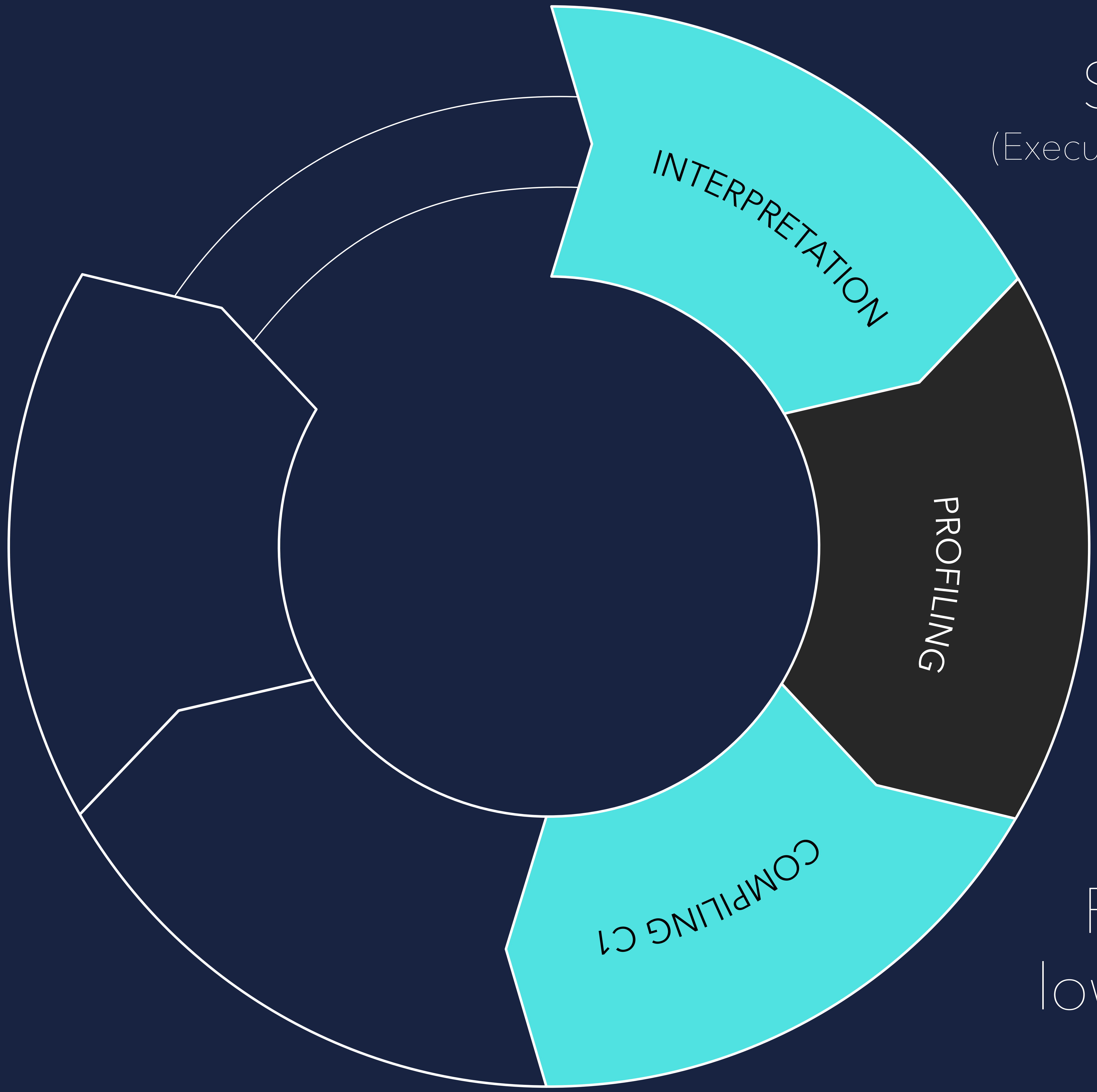


Slow
(Execution Level 0)

Finding
"hot spots"

azul

EXECUTION CYCLE



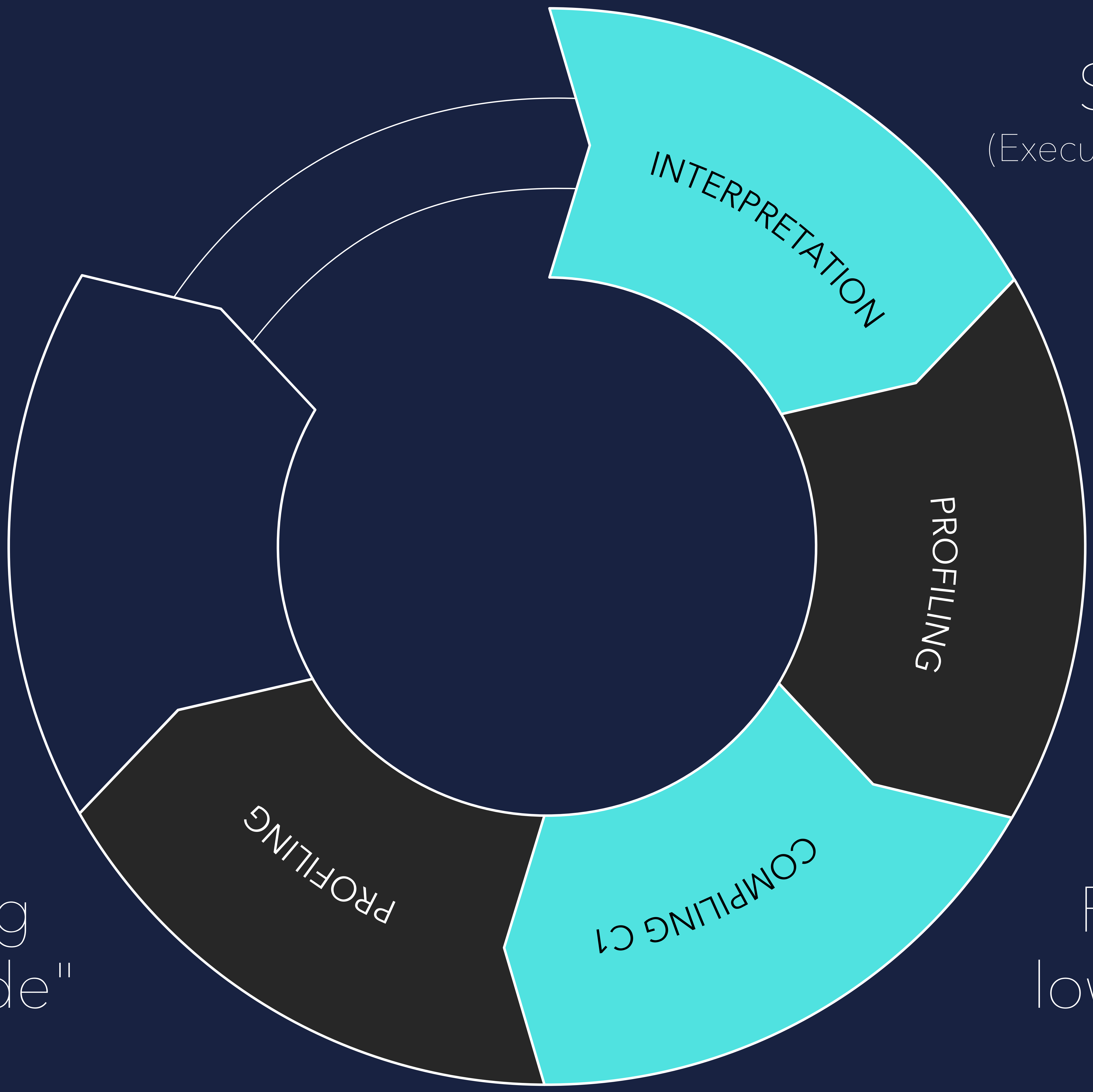
Slow
(Execution Level 0)

Finding "hot spots"

Fast compile, low optimisation
(Execution Level 3)

azul

EXECUTION CYCLE



Slow
(Execution Level 0)

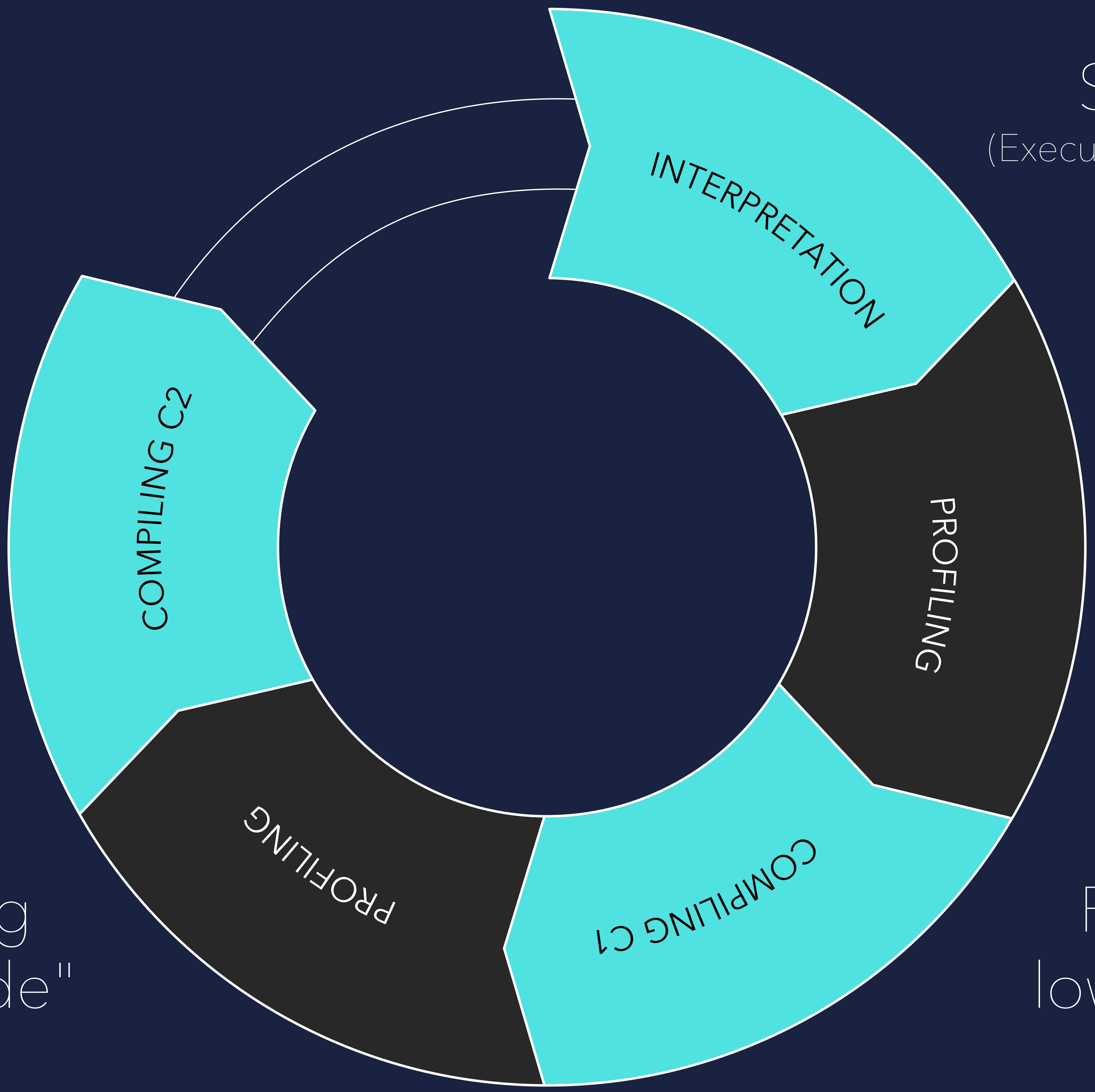
Finding "hot spots"

Fast compile, low optimisation
(Execution Level 3)

Finding "hot code"

azul

EXECUTION CYCLE



Slow
(Execution Level 0)

Finding
"hot spots"

Fast compile,
low optimisation
(Execution Level 3)

Finding
"hot code"

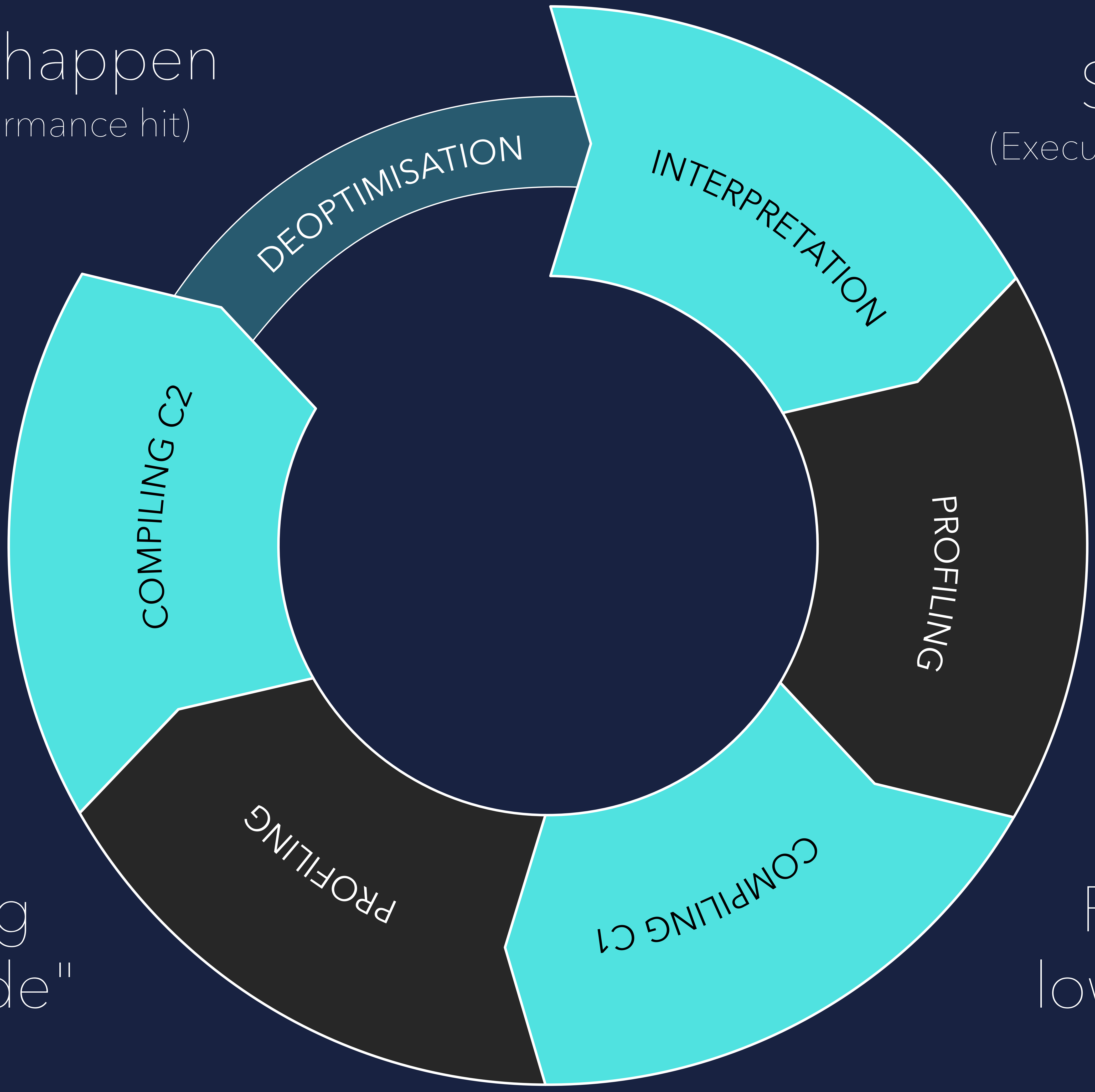
Slower compile,
high optimisation
(Execution Level 4)

azul

EXECUTION CYCLE

Can happen
(performance hit)

Slow
(Execution Level 0)



Finding
"hot spots"

Fast compile,
low optimisation
(Execution Level 3)

Finding
"hot code"

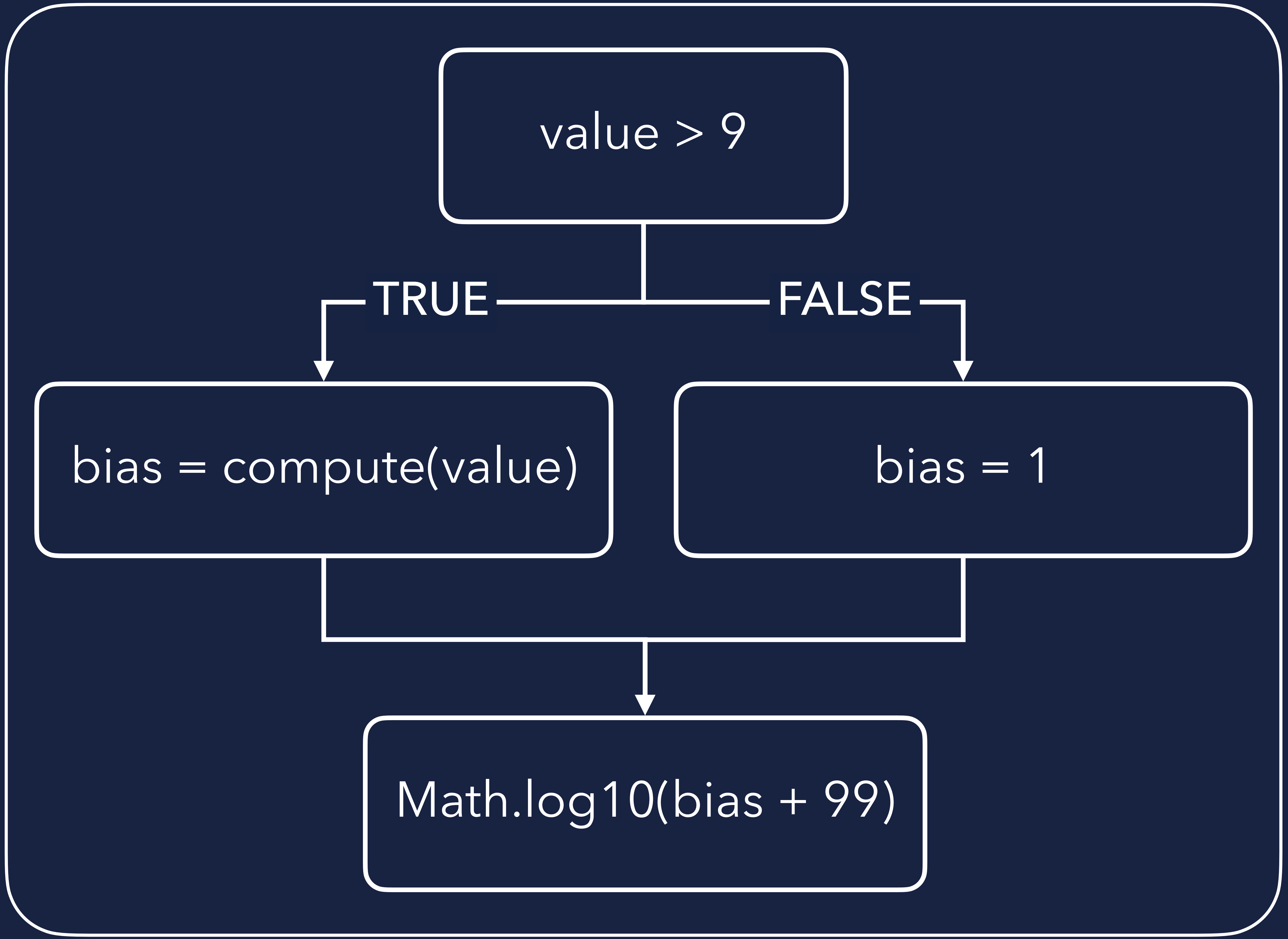
Slower compile,
high optimisation
(Execution Level 4)

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);
}
```

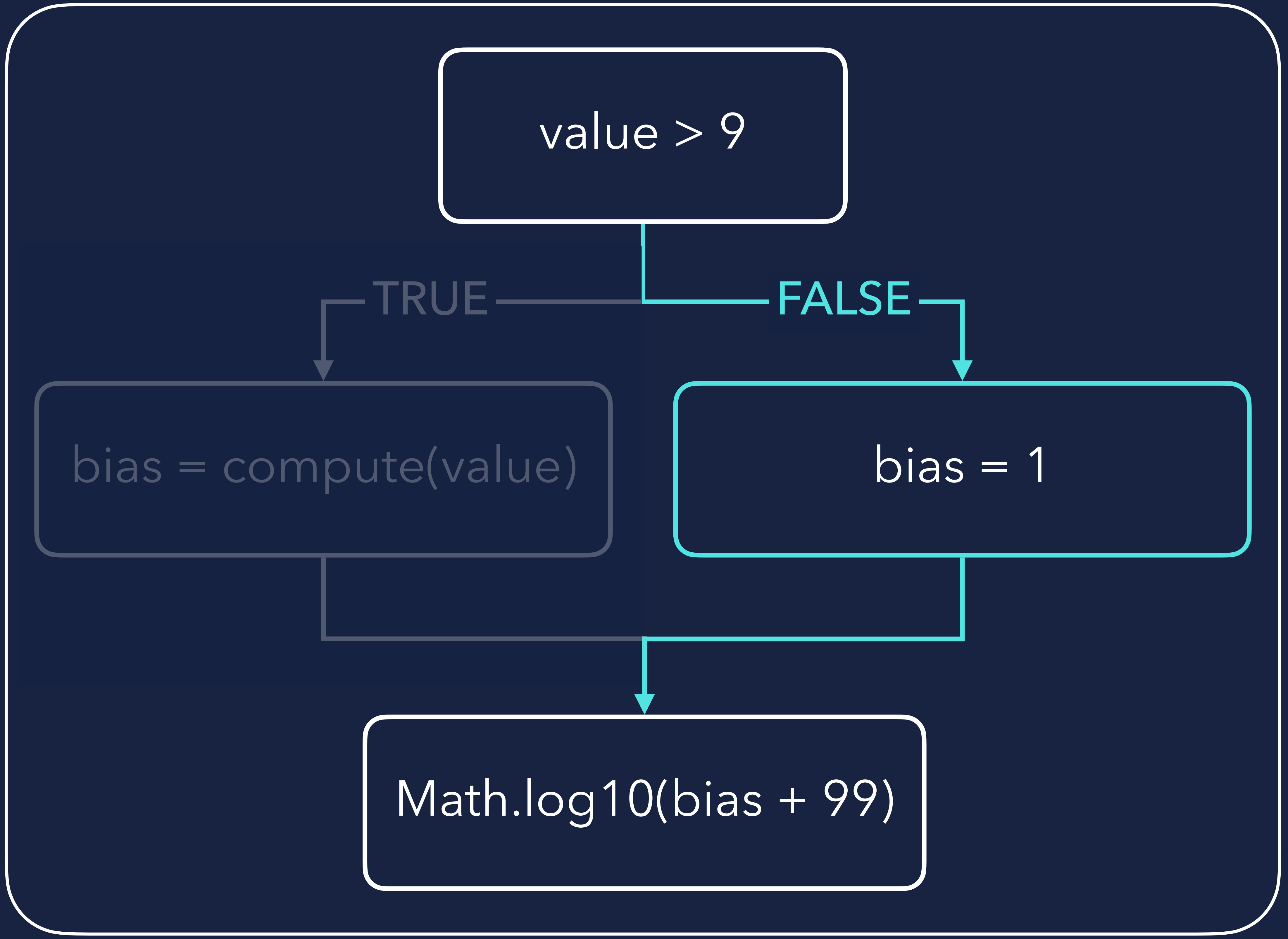


azul

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);
}
```



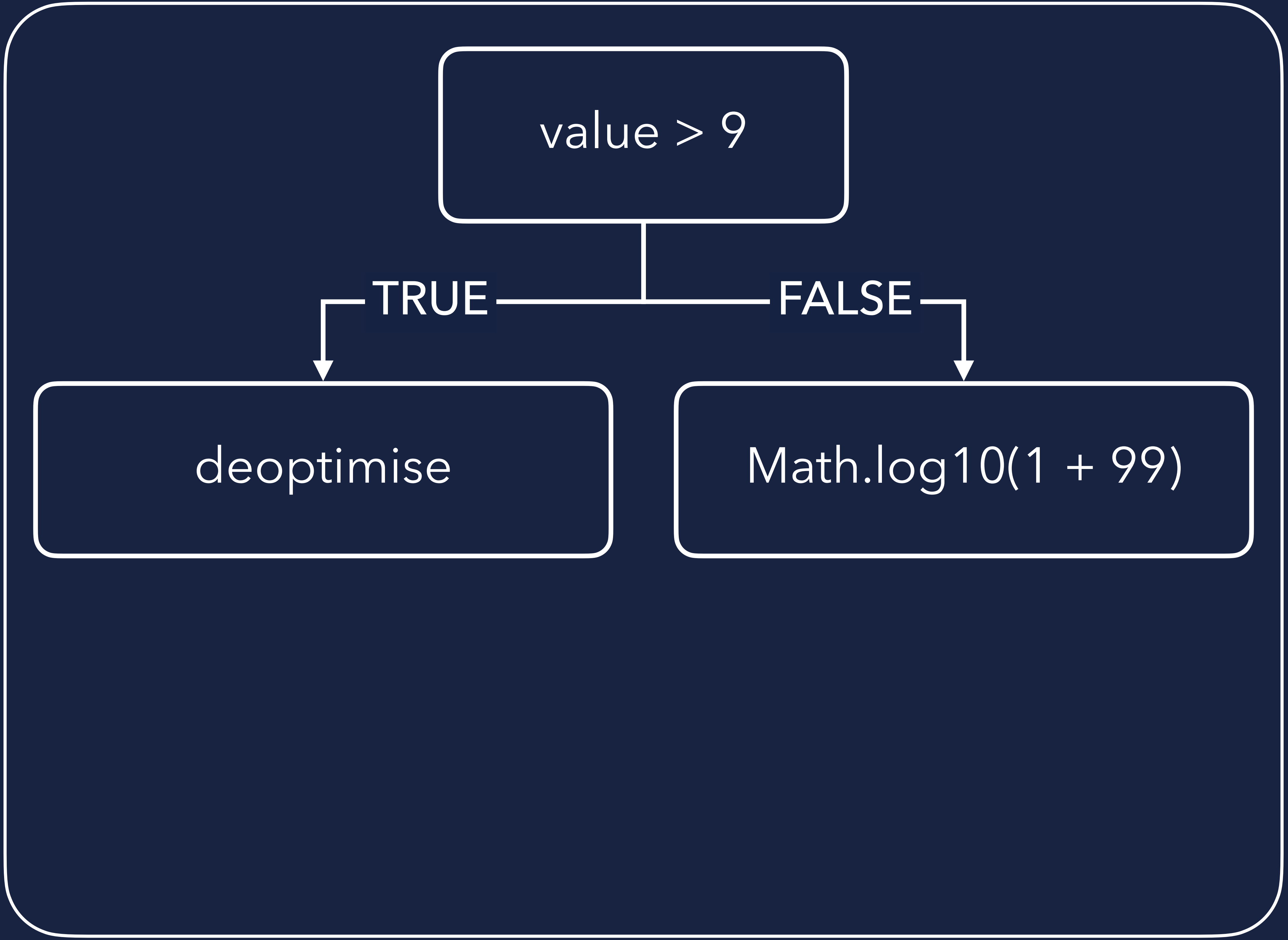
Value was never greater than 9

azul

DEOPTIMISATION

e.g. BRANCH ANALYSIS

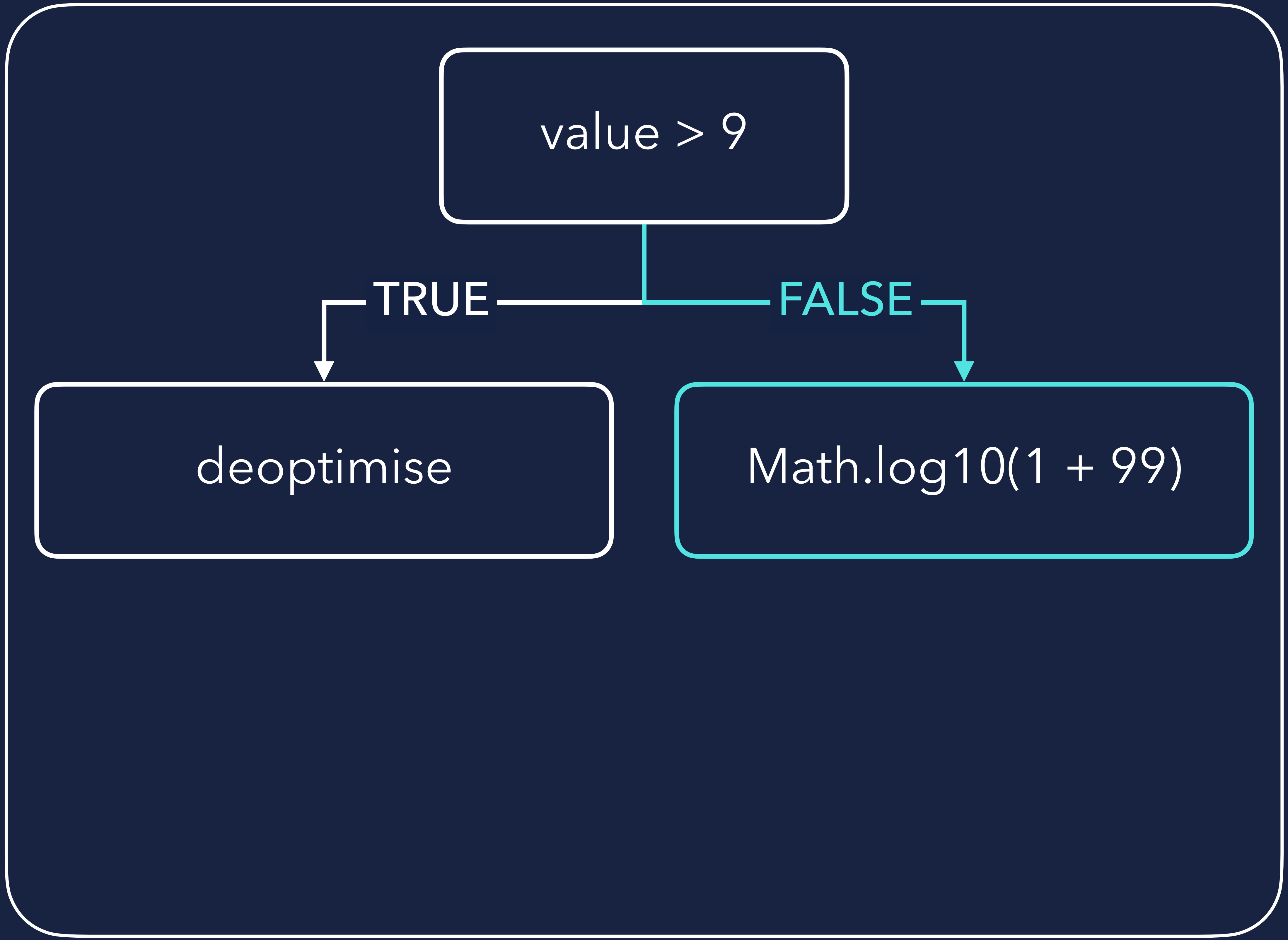
```
int computeMagnitude(int value) {
    if (value > 9) {
        uncommonTrap();
    }
    int bias = 1;
    return Math.log10(bias + 99);
}
```



DEOPTIMISATION

e.g. BRANCH ANALYSIS

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

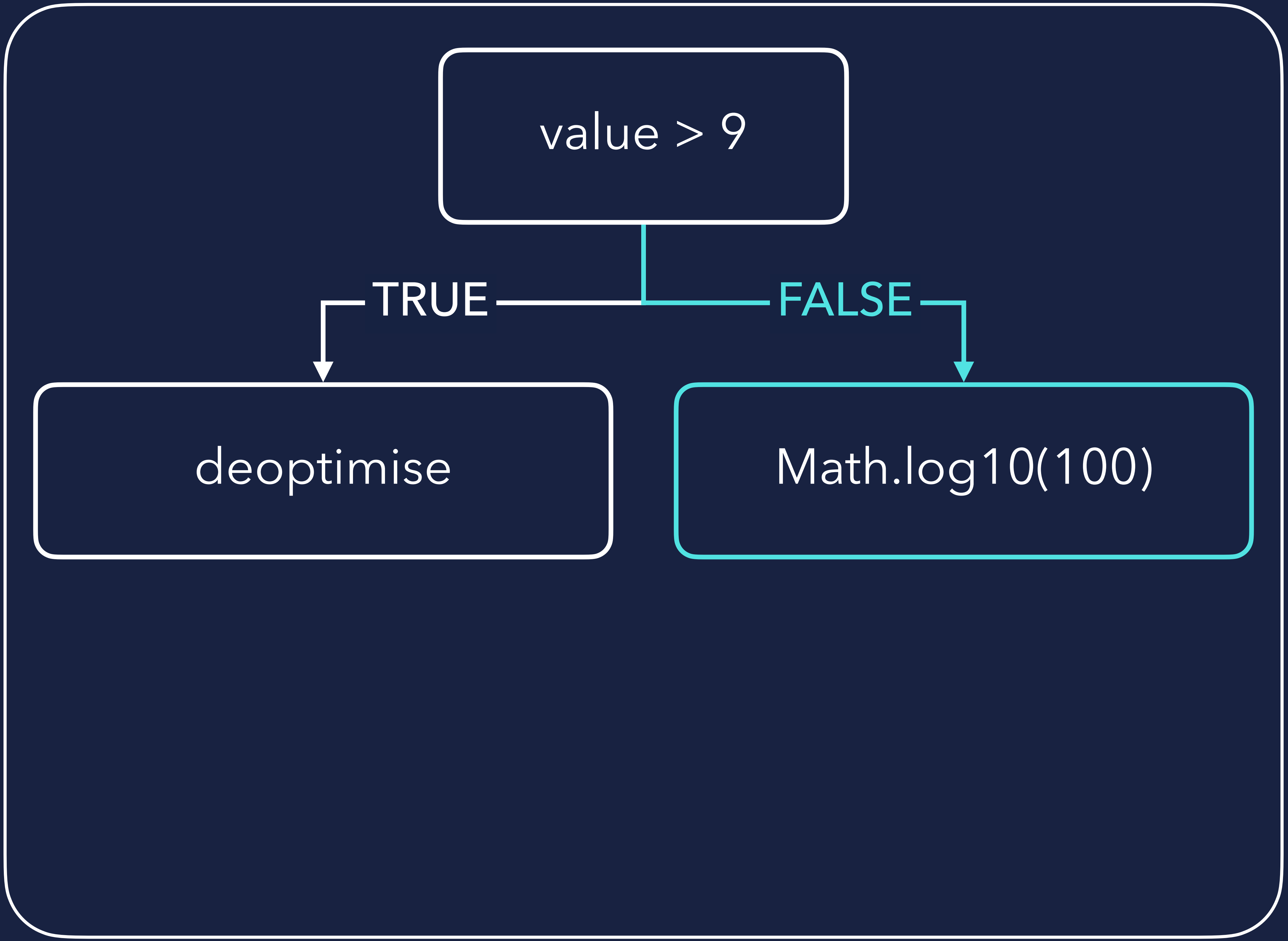


azul

DEOPTIMISATION

e.g. BRANCH ANALYSIS

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

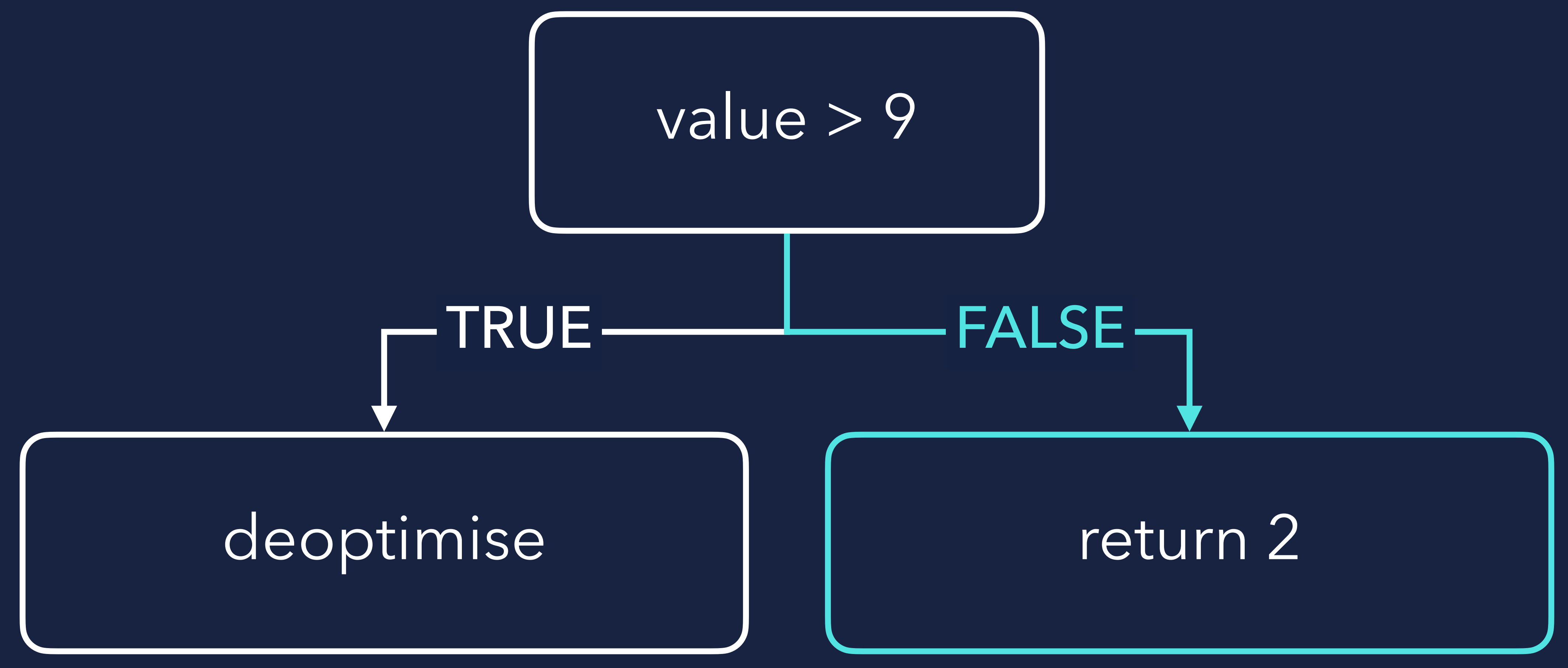


azul

DEOPTIMISATION

e.g. BRANCH ANALYSIS

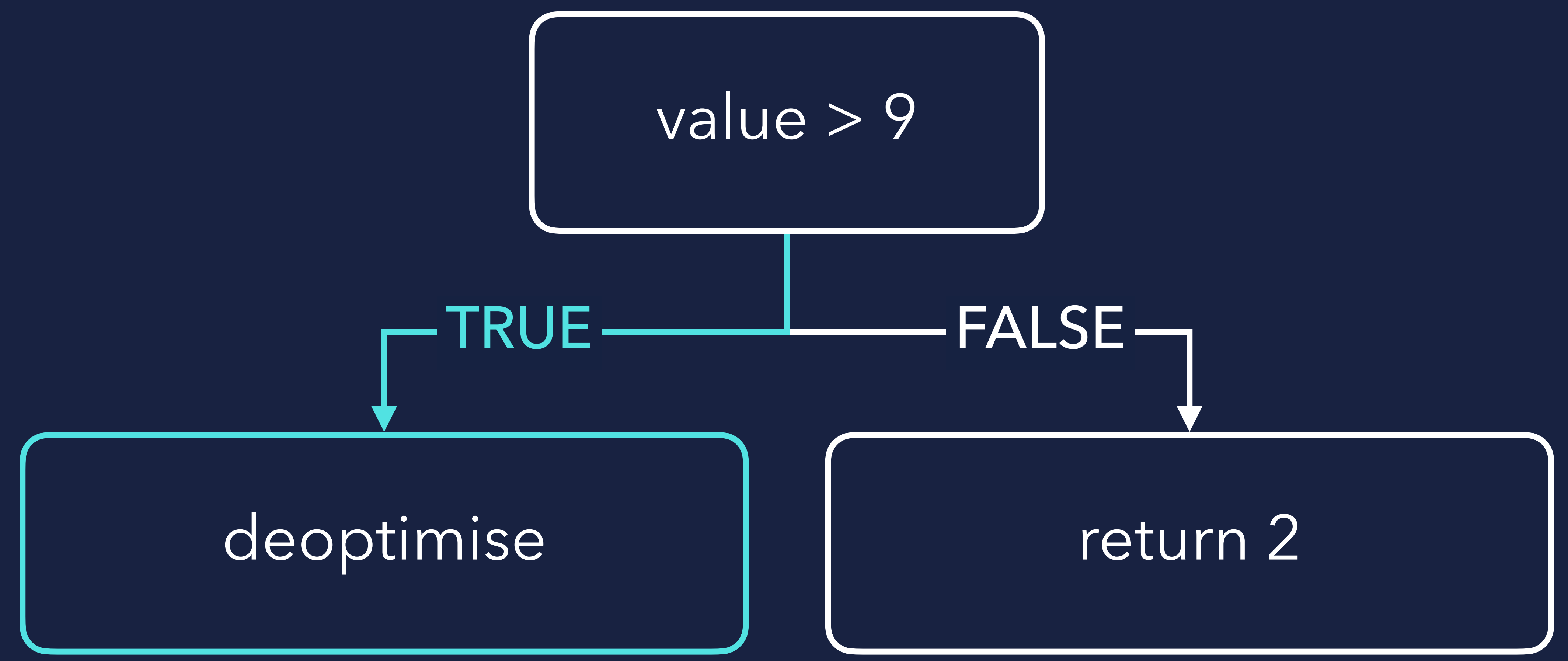
```
int computeMagnitude(int value) {
    if (value > 9) {
        uncommonTrap();
    }
    return 2;
}
```



DEOPTIMISATION

e.g. BRANCH ANALYSIS

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

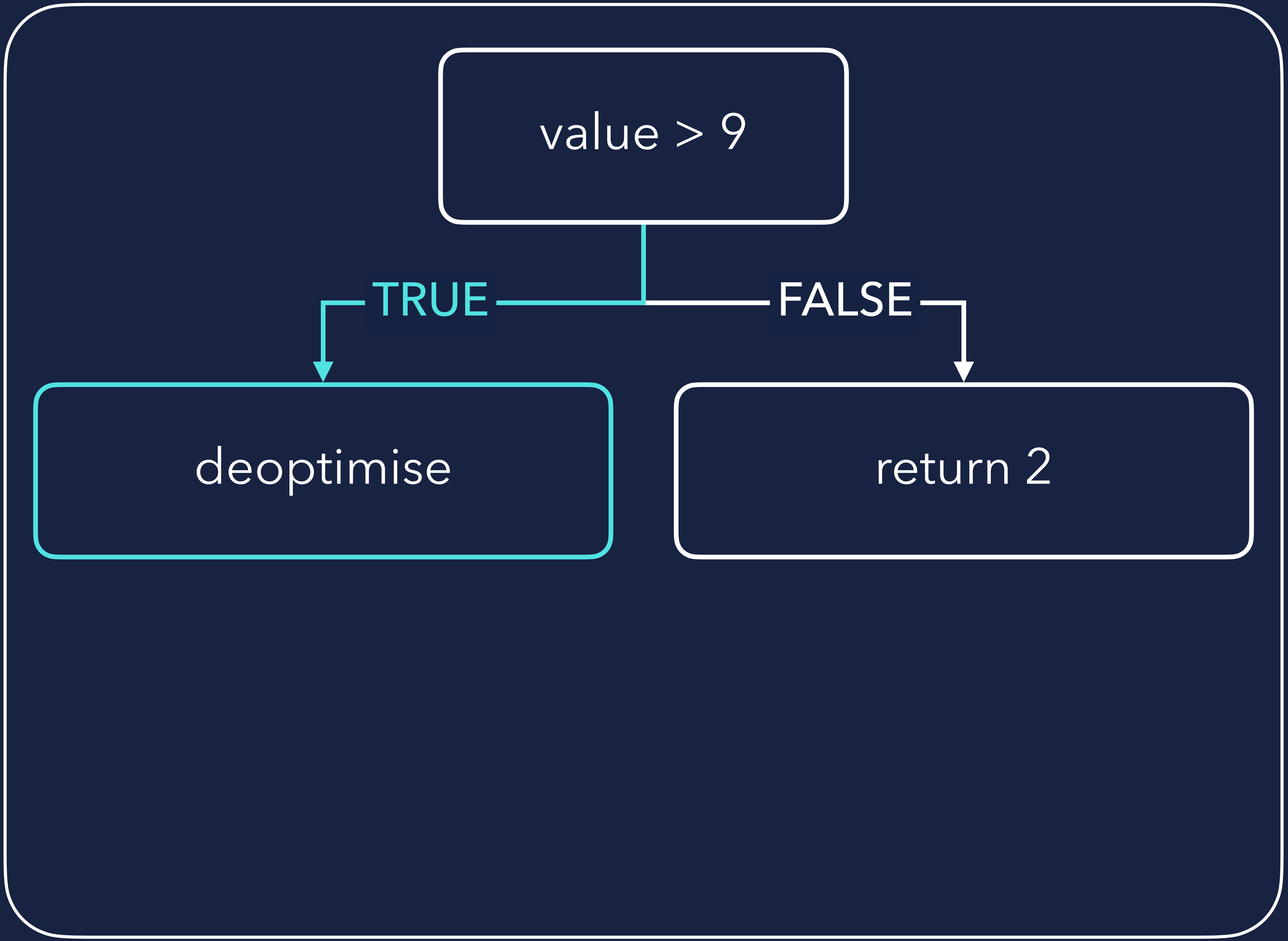


azul

DEOPTIMISATION

e.g. BRANCH ANALYSIS

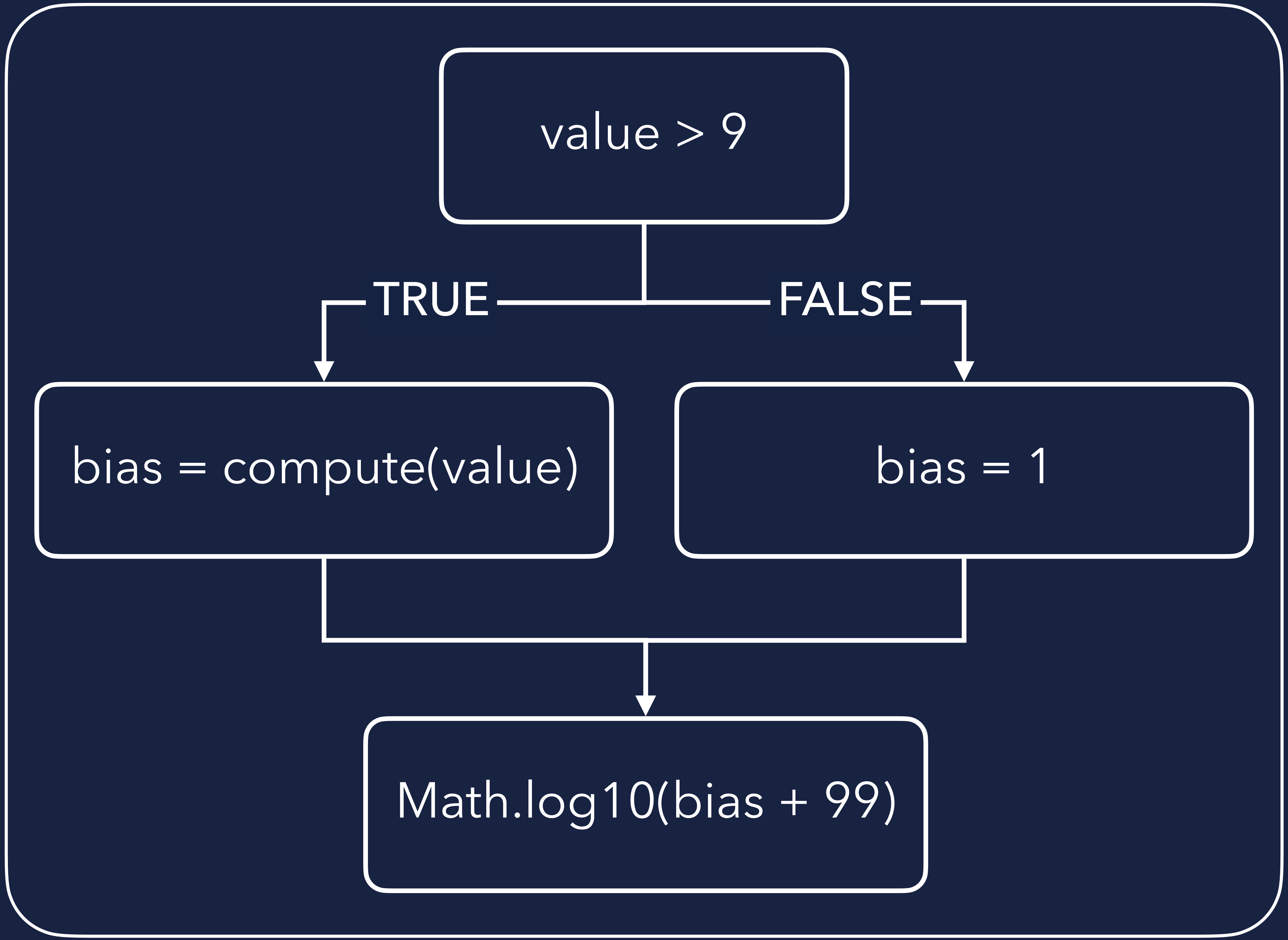
```
int computeMagnitude(int value) {
    if (value > 9) {
        uncommonTrap();
    }
    return 2;
}
```



DEOPTIMISATION

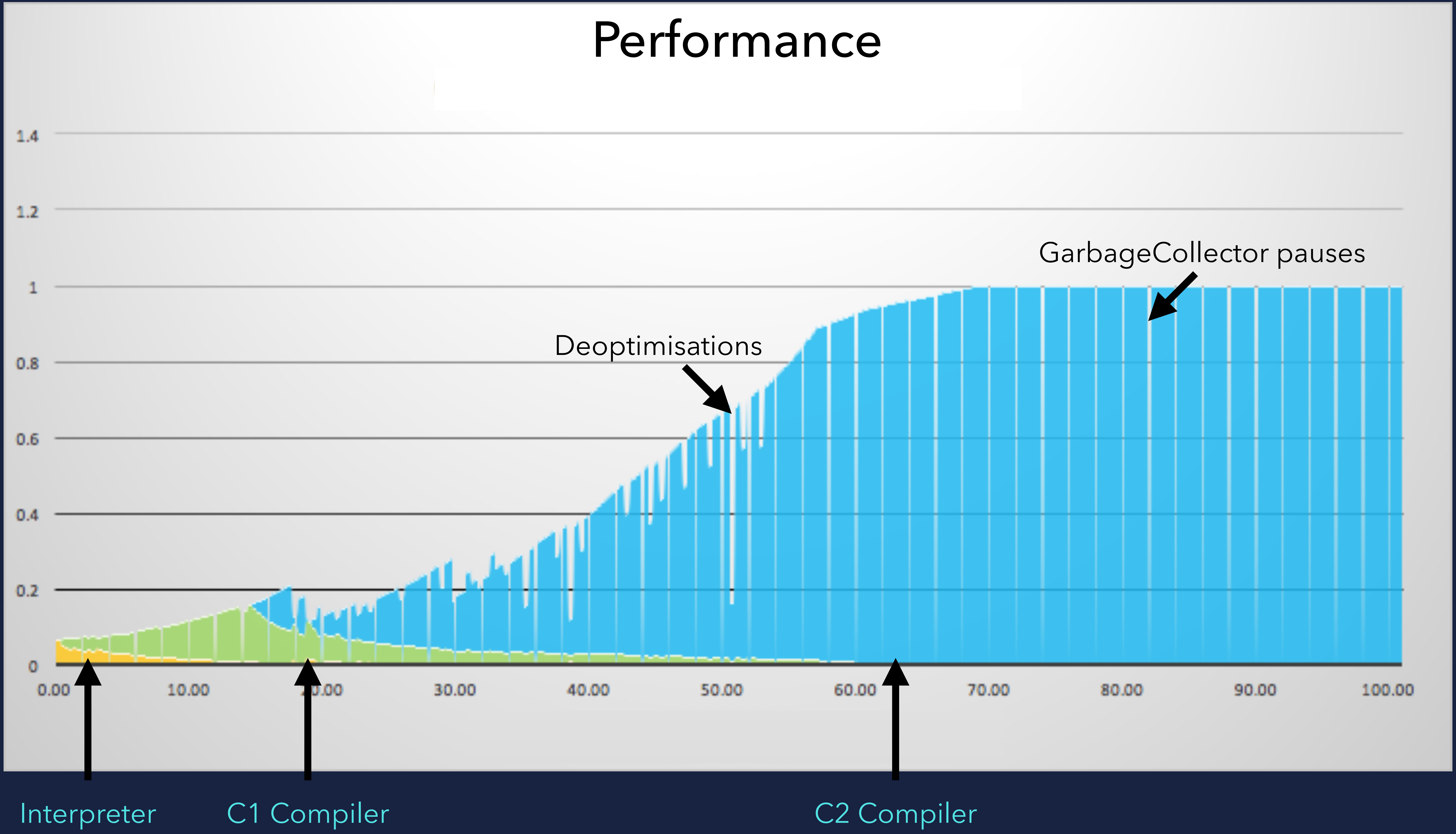
e.g. BRANCH ANALYSIS

```
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



azul

JVM STARTUP

azul

JVM STARTUP

FAST

JVM START

JVM

- Load & Initialize
- Optimization

azul

JVM STARTUP

FAST	TAKES A BIT
JVM START	**APPLICATION START**
JVM • Load & Initialize • Optimization	JVM • Load application classes • Initialize all resources • Kick off application specific logic • Optimization

JVM STARTUP

FAST	TAKES A BIT
JVM START	**APPLICATION START**
JVM - Load & Initialize - Optimization	JVM - Load application classes - Initialize all resources - Kick off application specific logic - Optimization

Generally referred to as JVM Startup
(Time to first response)

JVM STARTUP

FAST	TAKES A BIT	TAKES SOME TIME
JVM START	**APPLICATION START**	**APPLICATION WARMUP**
JVM • Load & Initialize • Optimization	JVM • Load application classes • Initialize all resources • Kick off application specific logic • Optimization	JVM • Optimizing (Compile/Decompile) App • Apply application specific workloads

Generally referred to as JVM Startup
(Time to first response)

JVM STARTUP

FAST	TAKES A BIT	TAKES SOME TIME
JVM START	APPLICATION START	APPLICATION WARMUP

JVM START

JVM
- Load & Initialize
- Optimization

APPLICATION START

JVM
- Load application classes
- Initialize all resources
- Kick off application specific logic
- Optimization

APPLICATION WARMUP

JVM
- Optimizing (Compile/Decompile)

App
- Apply application specific workloads

Generally referred to as JVM Startup
(Time to first response)

Generally referred to as JVM Warmup
(Time to n operations)

THAT'S GREAT...

azul

...BUT...

azul

...IT TAKES TIME !

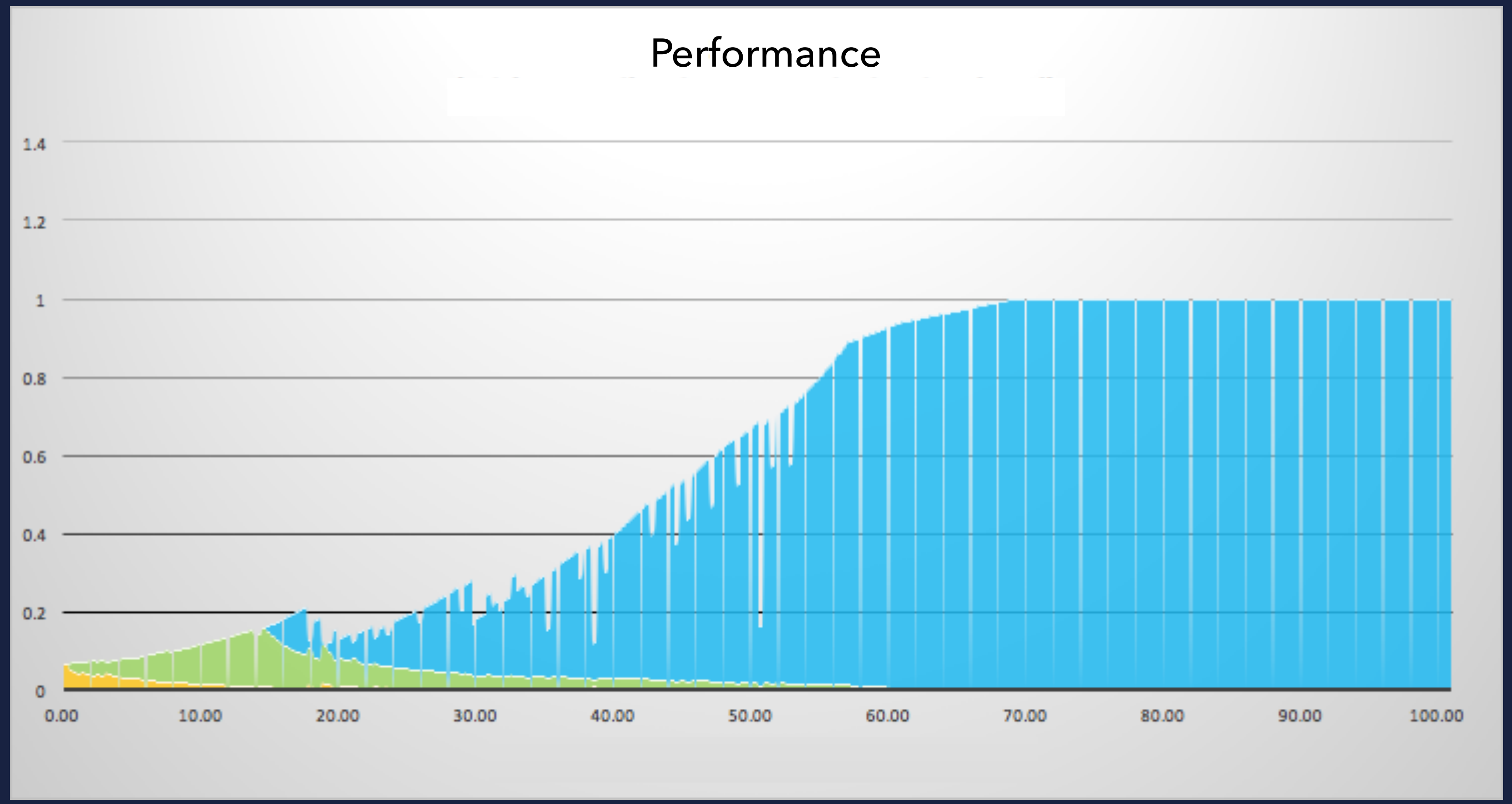
azul

MICROSERVICE ENVIRONMENT

azul

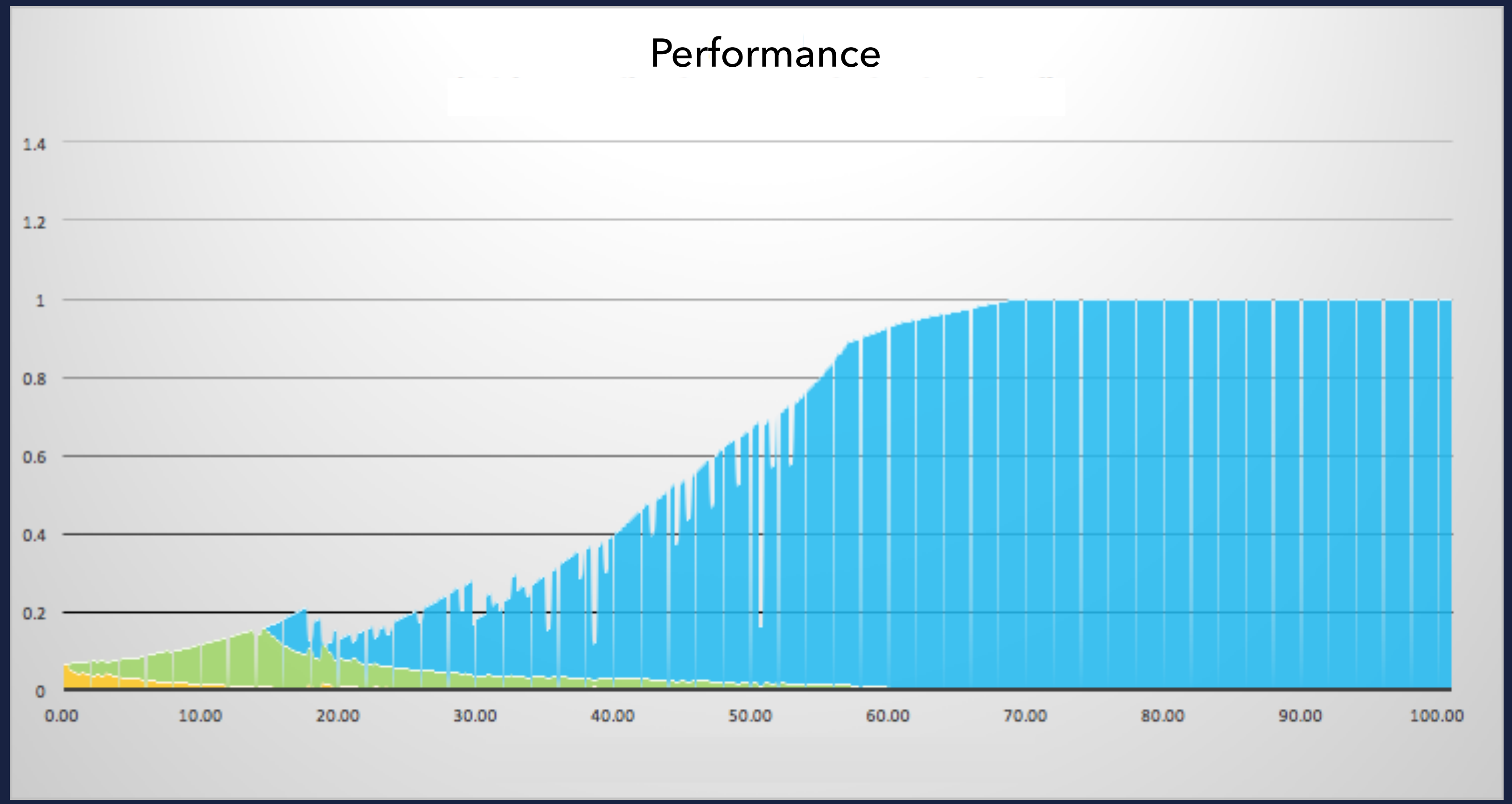
MICROSERVICE ENVIRONMENT

FIRST RUN



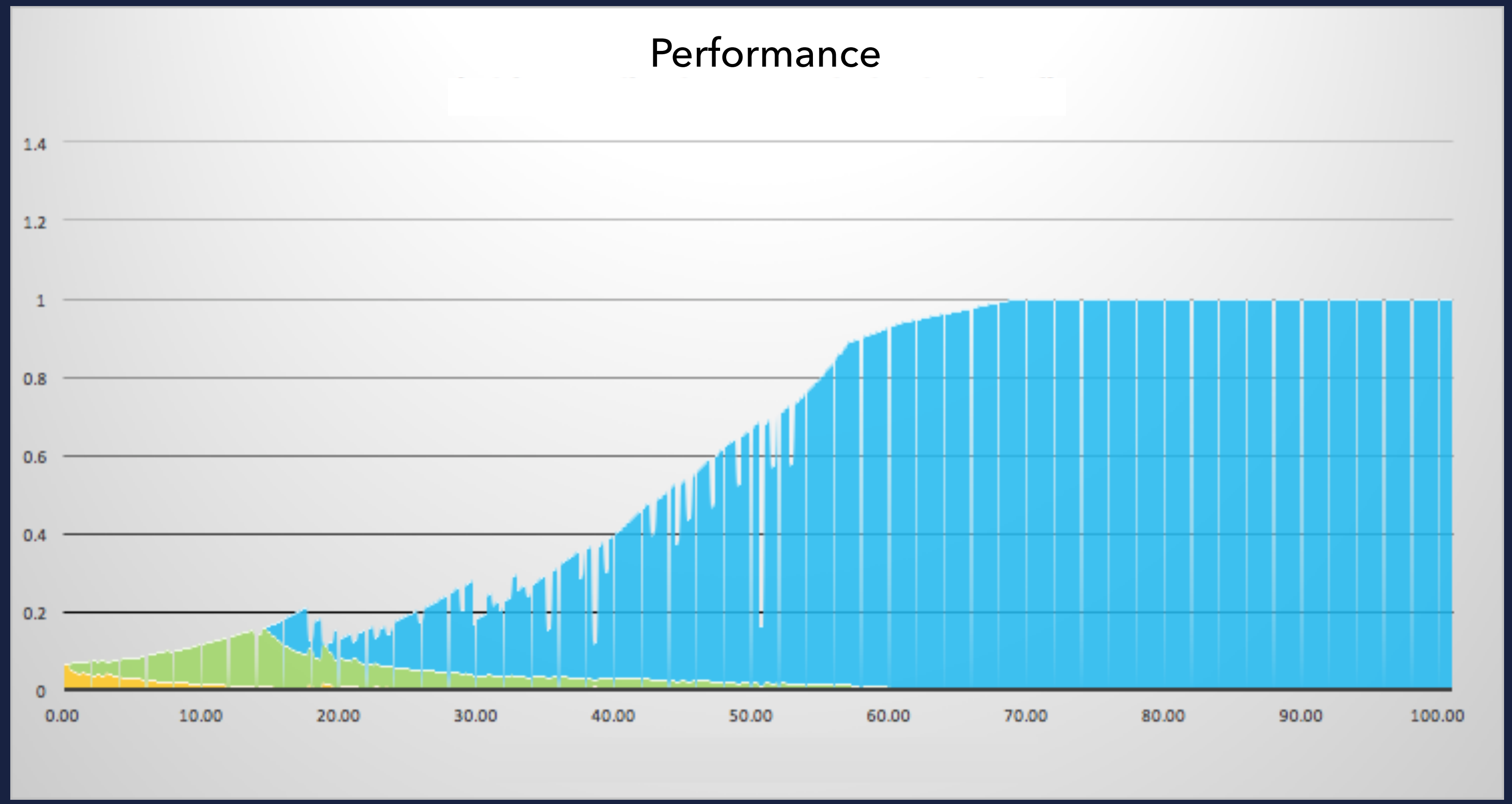
JVM STARTUP

SECOND RUN



JVM STARTUP

THIRD RUN

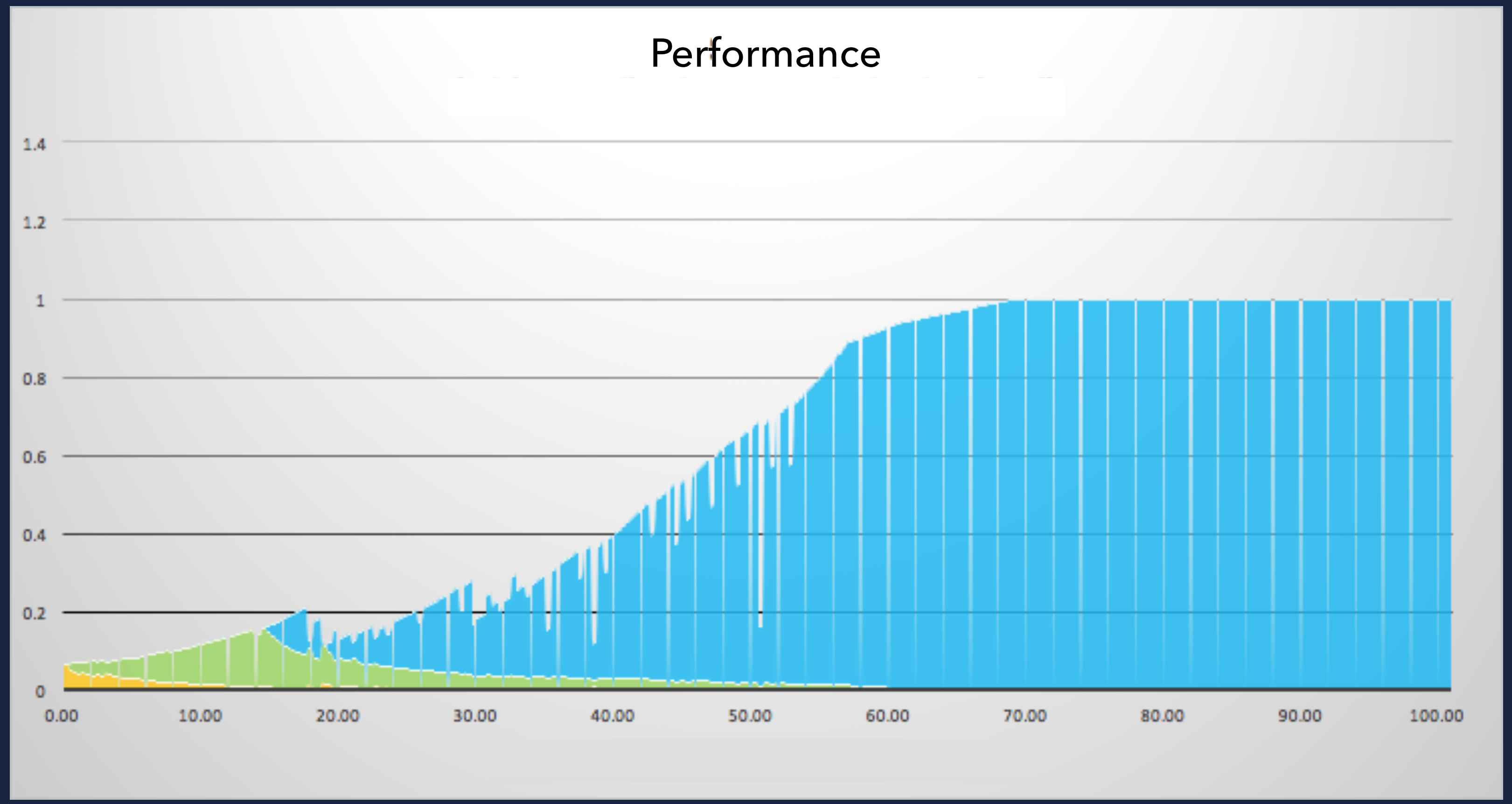


JVM STARTUP

azul

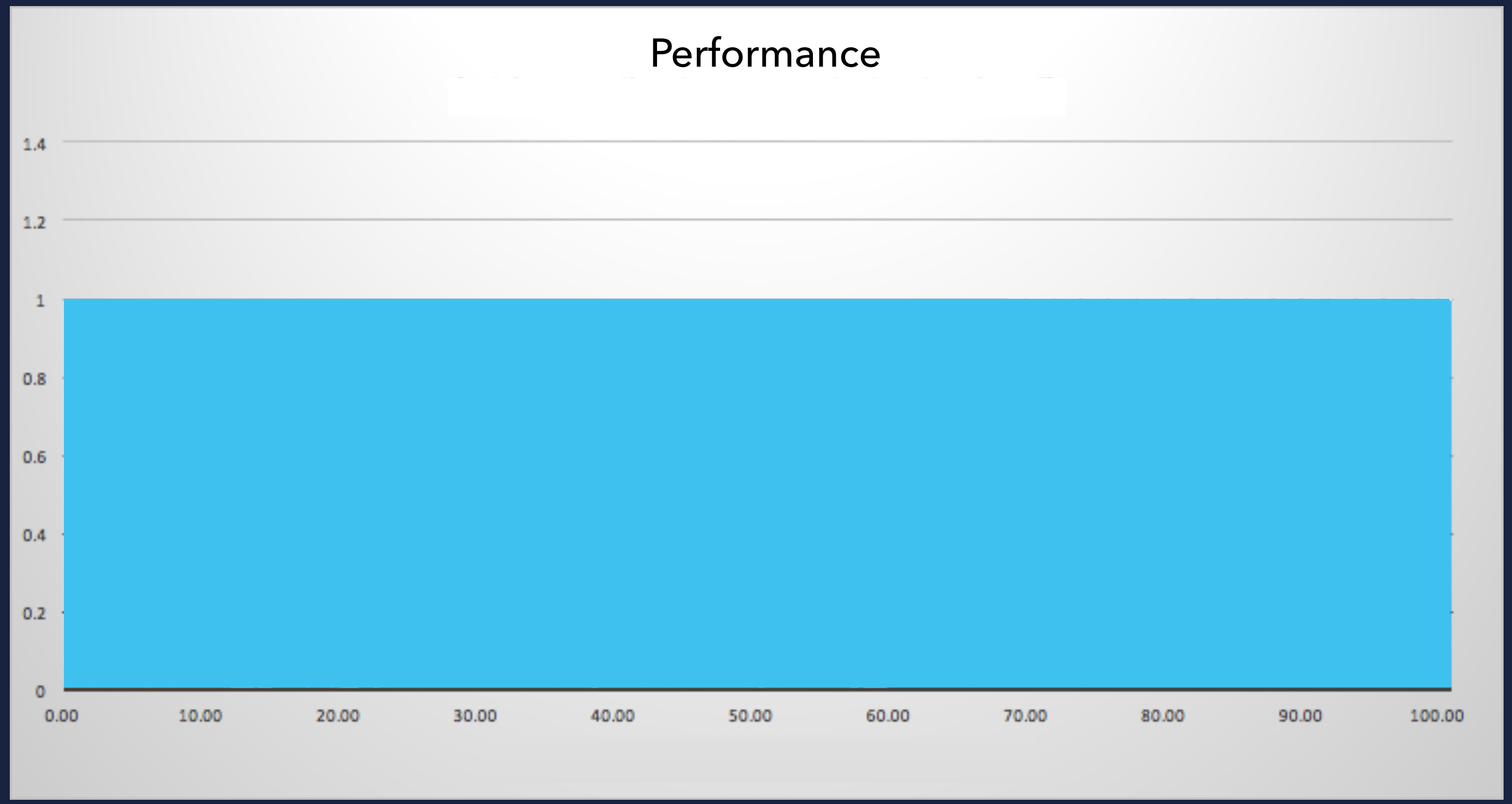
WOULDN'T IT BE GREAT…?

FIRST RUN



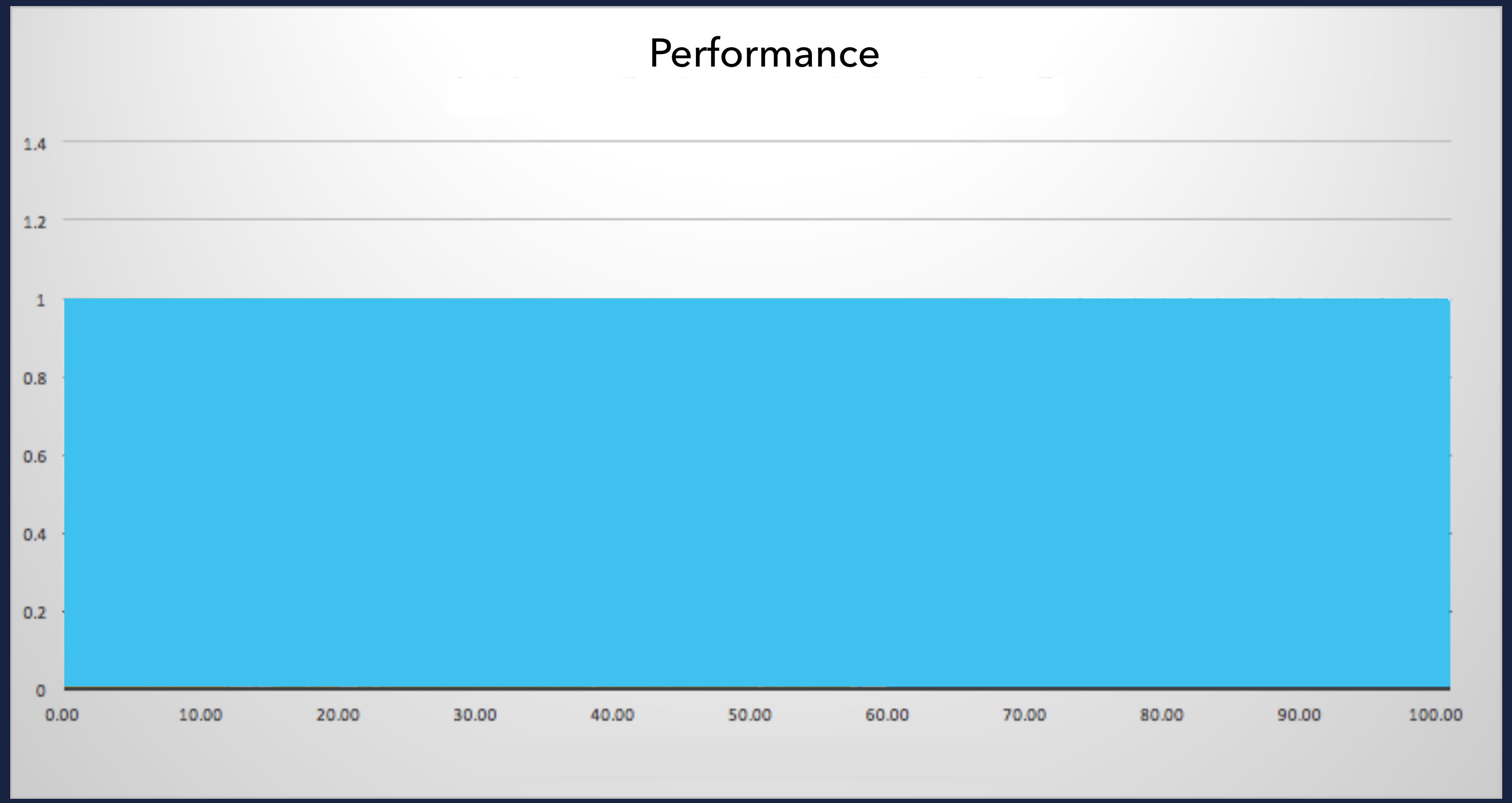
JVM STARTUP

SECOND RUN



NO STARTUP OVERHEAD

THIRD RUN



NO STARTUP OVERHEAD

azul

SOLUTIONS…?

azul

CLASS DATA SHARING

azul

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

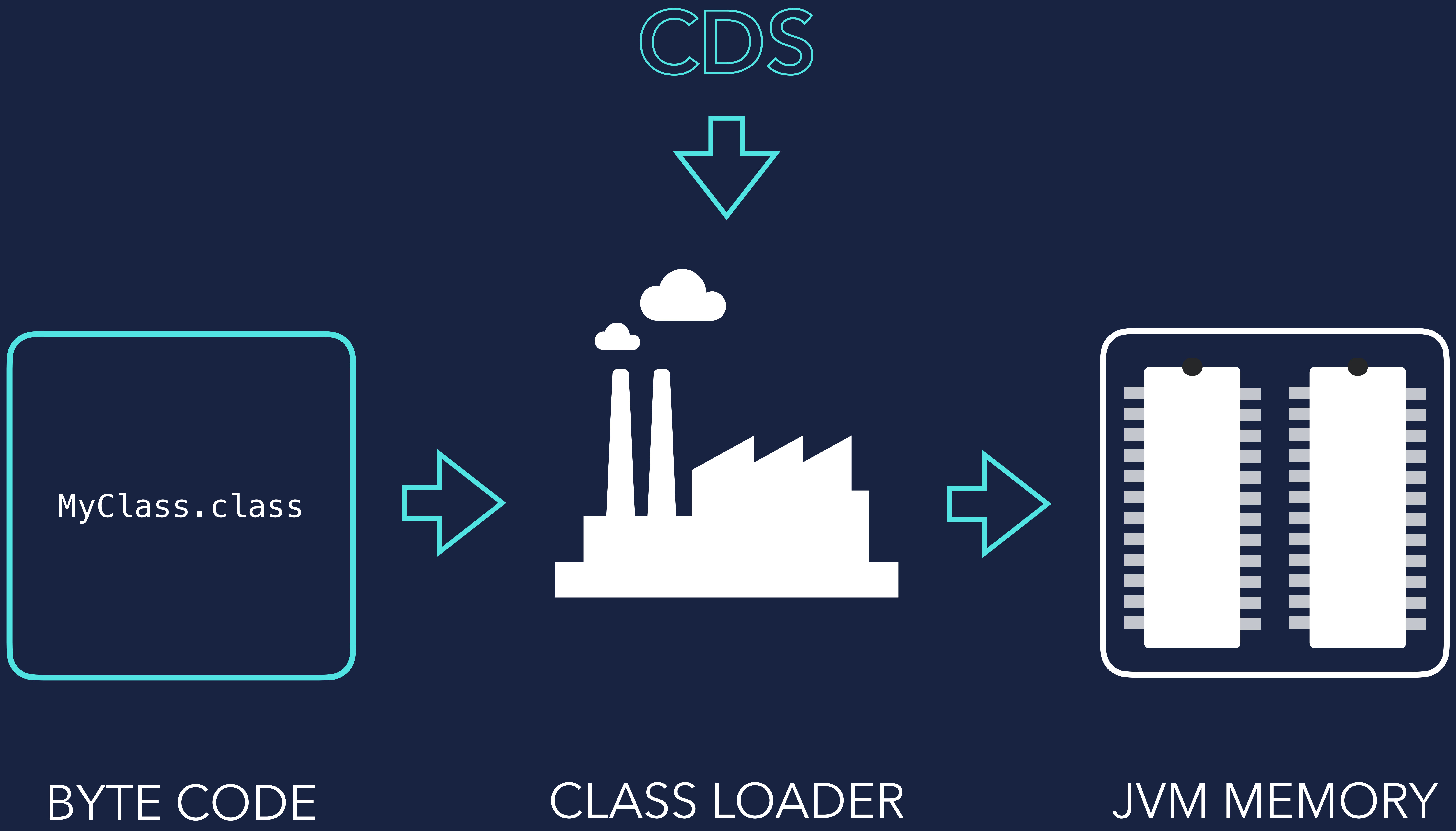
WHAT ABOUT CDS?

- 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

azul


CDS
MyClass.class
BYTE CODE
CLASS LOADER
JVM MEMORY

AHEAD OF TIME COMPILATION

azul

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

azul

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

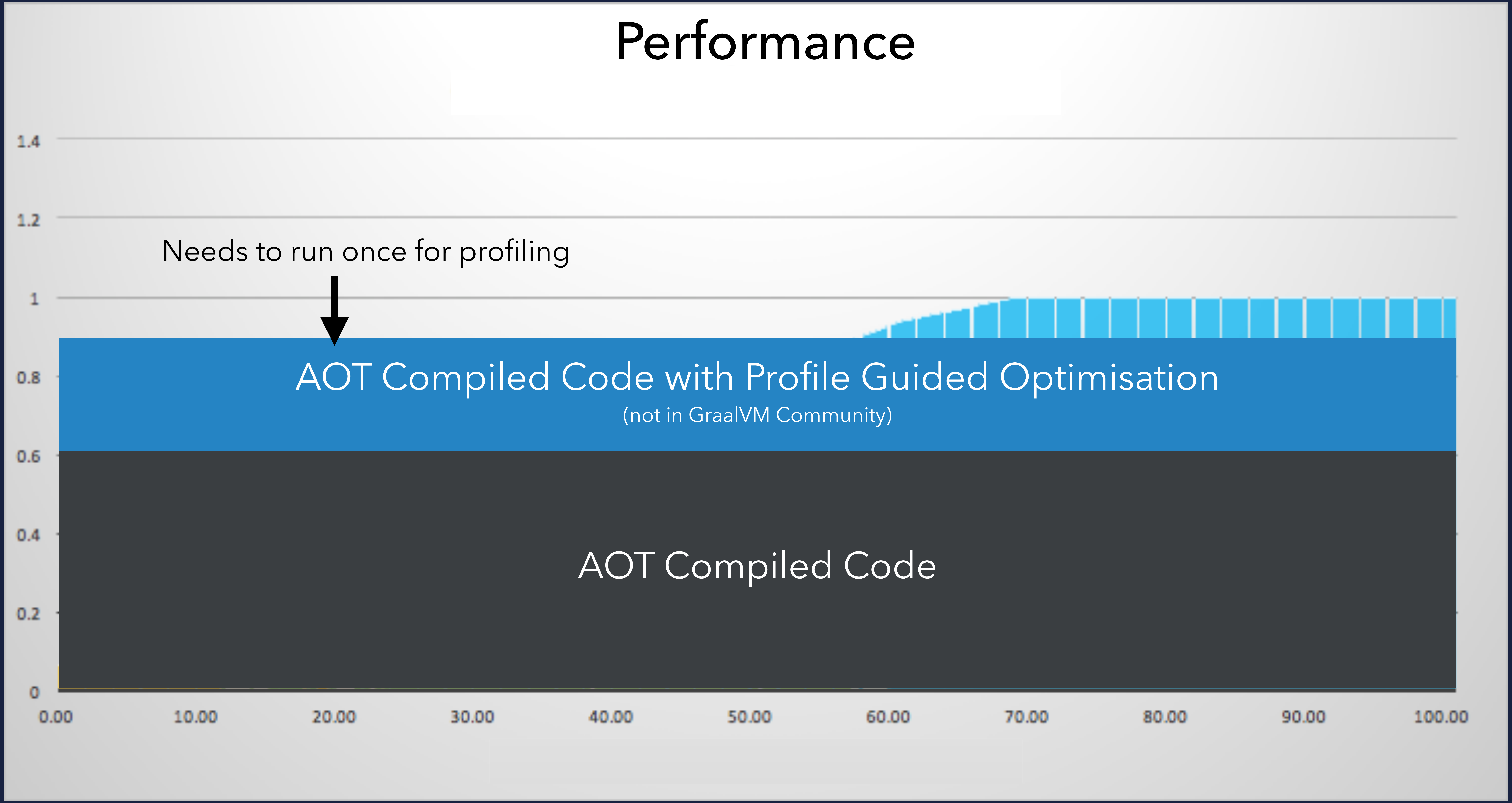
NOT SO FAST...

- 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



azul

JVM PERFORMANCE

Metrics	Spring Boot JVM	Quarkus JVM	Spring Boot Native	Quarkus Native
Startup time (sec)	1.865	1.274	0.129	0.110
Build artifact time (sec)	1.759	5.243	113	91
Artifact size (MB)	30.0	31.8	94.7	80.5
Loaded classes	8861	8496	21615	16040
CPU usage max(%)	100	100	100	100
CPU usage average(%)	82	73	94	92
Heap size startup (MB)	1048.57	1056.96	-	-
Used heap startup (MB)	83	62	12	58
Used heap max (MB)	780	782	217	529
Used heap average (MB)	675	534	115	379
RSS memory startup (MB)	494.04	216.1	90.91	71.92
Max threads	77	47	73	42
Requests per Second	7887.29	9373.38	5865.02	4932.04

https://www.baeldung.com/spring-boot-vs-quarkus

JVM PERFORMANCE

	JVM		NATIVE IMAGE	
Metrics	Spring Boot JVM	Quarkus JVM	Spring Boot Native	Quarkus Native
Startup time (sec)	1.865	1.274	0.129	0.110
Build artifact time (sec)	1.759	5.243	113	91
Artifact size (MB)	30.0	31.8	94.7	80.5
Loaded classes	8861	8496	21615	16040
CPU usage max(%)	100	100	100	100
CPU usage average(%)	82	73	94	92
Heap size startup (MB)	1048.57	1056.96	-	-
Used heap startup (MB)	83	62	12	58
Used heap max (MB)	780	782	217	529
Used heap average (MB)	675	534	115	379
RSS memory startup (MB)	494.04	216.1	90.91	71.92
Max threads	77	47	73	42
Requests per Second	7887.29	9373.38	5865.02	4932.04

https://www.baeldung.com/spring-boot-vs-quarkus

JVM PERFORMANCE

	JVM		NATIVE IMAGE	
Metrics	Spring Boot JVM	Quarkus JVM	Spring Boot Native	Quarkus Native
Startup time (sec)	1.865	1.274	0.129	0.110
Build artifact time (sec)	1.759	5.243	113	91
Artifact size (MB)	30.0	31.8	94.7	80.5
Loaded classes	8861	8496	21615	16040
CPU usage max(%)	100	100	100	100
CPU usage average(%)	82	73	94	92
Heap size startup (MB)	1048.57	1056.96	-	-
Used heap startup (MB)	83	62	12	58
Used heap max (MB)	780	782	217	529
Used heap average (MB)	675	534	115	379
RSS memory startup (MB)	494.04	216.1	90.91	71.92
Max threads	77	47	73	42
Requests per Second	7887.29	9373.38	5865.02	4932.04
	100%	100%	74%	53%

https://www.baeldung.com/spring-boot-vs-quarkus

AOT VS JIT

AOT

- Limited use of method inlining
- No runtime bytecode generation
- Reflection is possible but complicated
- Unable to use speculative optimisations
- Must be compiled for least common denominator
- Overall performance will typically be lower
- Deployed env != Development env.

- 'Full speed' from the start
- No overhead to compile code at runtime
- Small memory footprint

JIT

- Can use aggressive method inlining at runtime
- Can use runtime bytecode generation
- Reflection is simple
- Can use speculative optimisations
- Can even optimise for Haswell, Skylake, Ice Lake etc.
- Overall performance will typically be higher
- Deployed env. == Development env.

- Requires more time to start up (but will be faster)
- Overhead to compile code at runtime
- Larger memory footprint

JIT DISADVANTAGES

- Requires more time to start up

(requires many slow operations to happen before optimisation and faster execution can happen)

JIT DISADVANTAGES

- Requires more time to start up

  (requires many slow operations to happen before optimisation and faster execution can happen)
- CPU overhead to compile code at runtime

JIT DISADVANTAGES

- Requires more time to start up
  (requires many slow operations to happen before optimisation and faster execution can happen)
- CPU overhead to compile code at runtime
- Larger memory footprint

AZUL PRIME READY NOW

READY NOW

- Part of Azul Prime JVM

READY NOW

- Part of Azul Prime JVM
- Creates profile at runtime (optimizations and constraints)

READY NOW

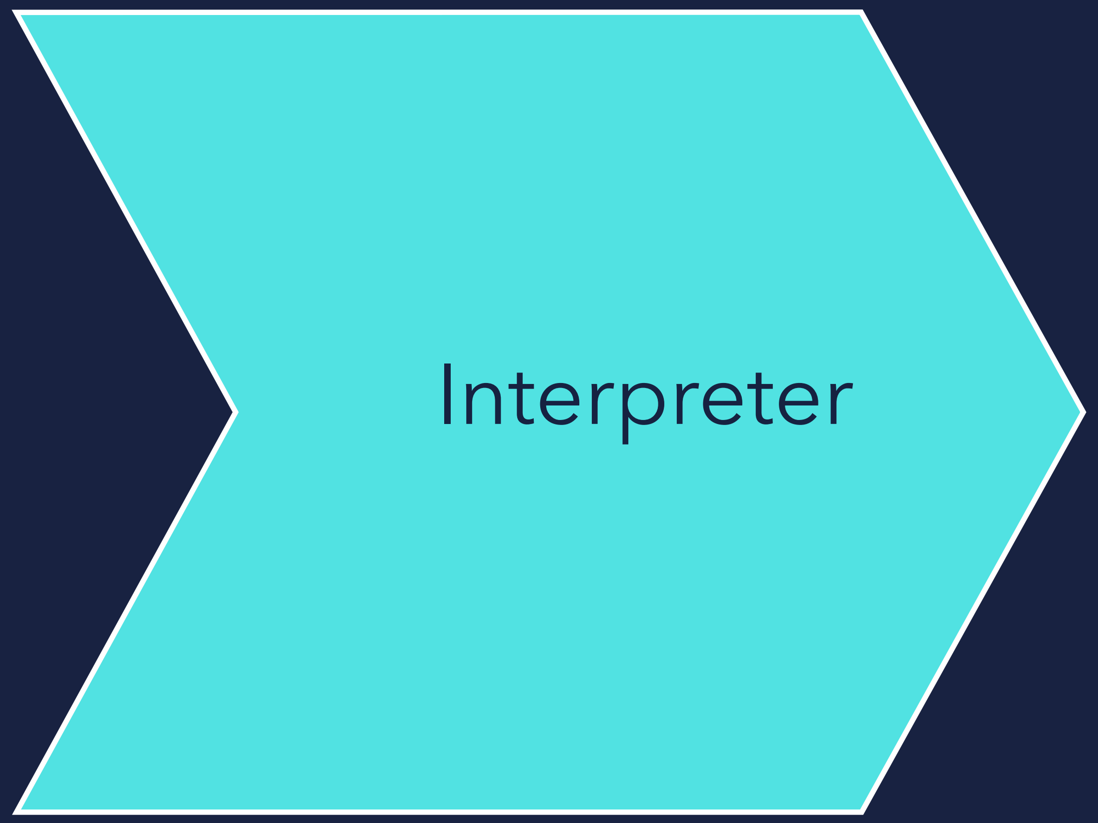
- Part of Azul Prime JVM
- Creates profile at runtime (optimizations and constraints)
- Compile everything from the profile (at startup)

READY NOW

- Part of Azul Prime JVM
- Creates profile at runtime (optimizations and constraints)
- Compile everything from the profile (at startup)
- JVM can further optimize

READY NOW

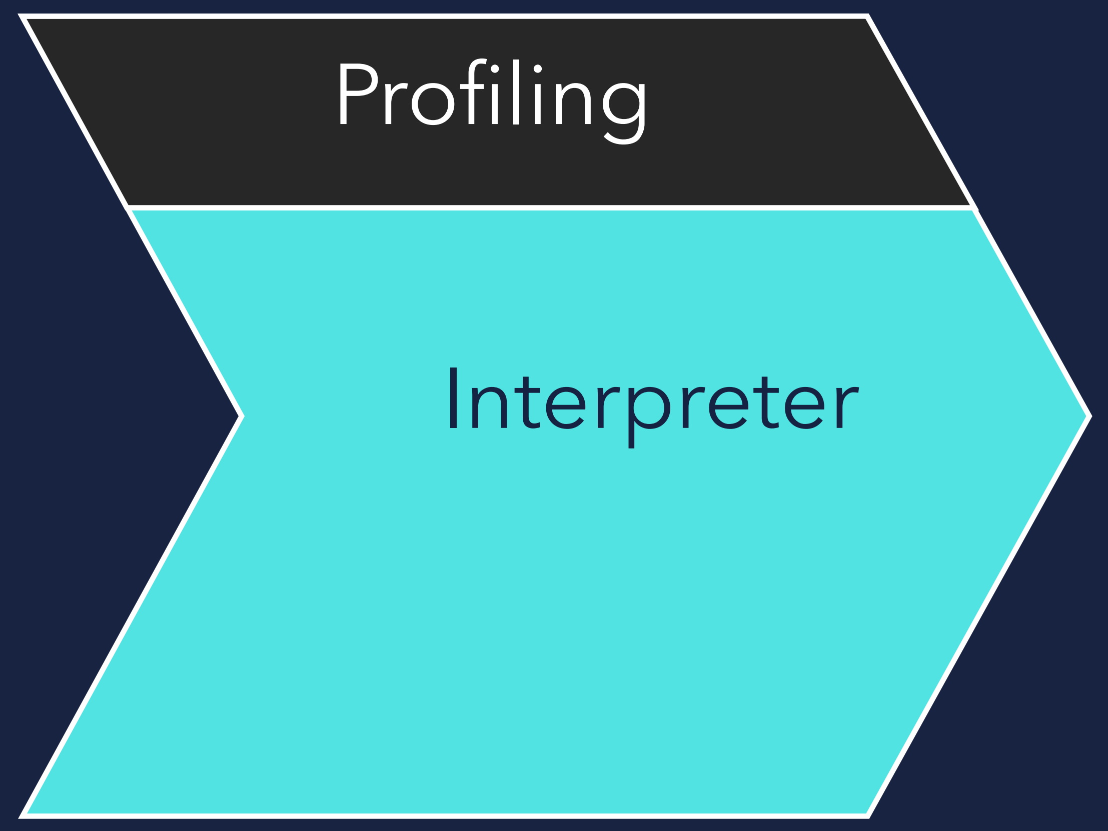
FIRST STARTUP...



Prime will store all optimizations & constraints to ReadyNow profile

READY NOW

FIRST STARTUP...



Prime will store all optimizations & constraints to ReadyNow profile

READY NOW

FIRST STARTUP...

Prime will store all optimizations & constraints to ReadyNow profile

READY NOW

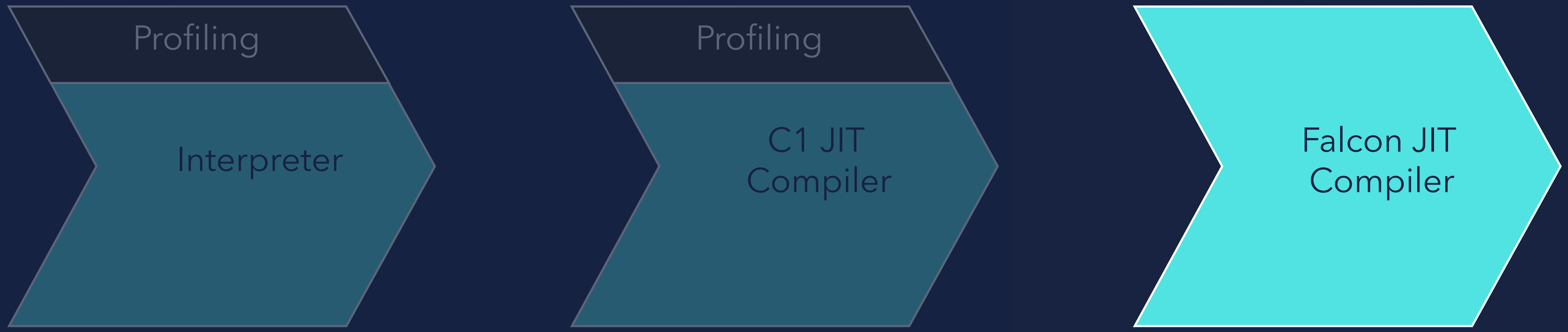
FIRST STARTUP...

Prime will store all optimizations & constraints to ReadyNow profile

azul

READY NOW

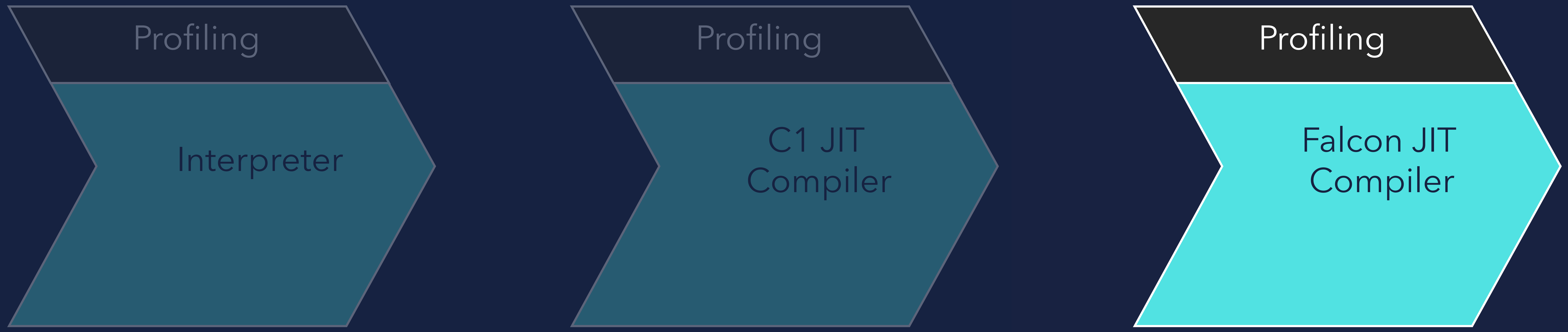
FIRST STARTUP...



Prime will store all optimizations & constraints to ReadyNow profile

READY NOW

FIRST STARTUP...



Prime will store all optimizations & constraints to ReadyNow profile

READY NOW

NEXT STARTUP...

Everything in the ReadyNow profile will directly be compiled

A DIFFERENT APPROACH

azul

CRIU

CHECKPOINT RESTORE IN USERSPACE

azul

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

azul

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)

azul

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.)

CRaC

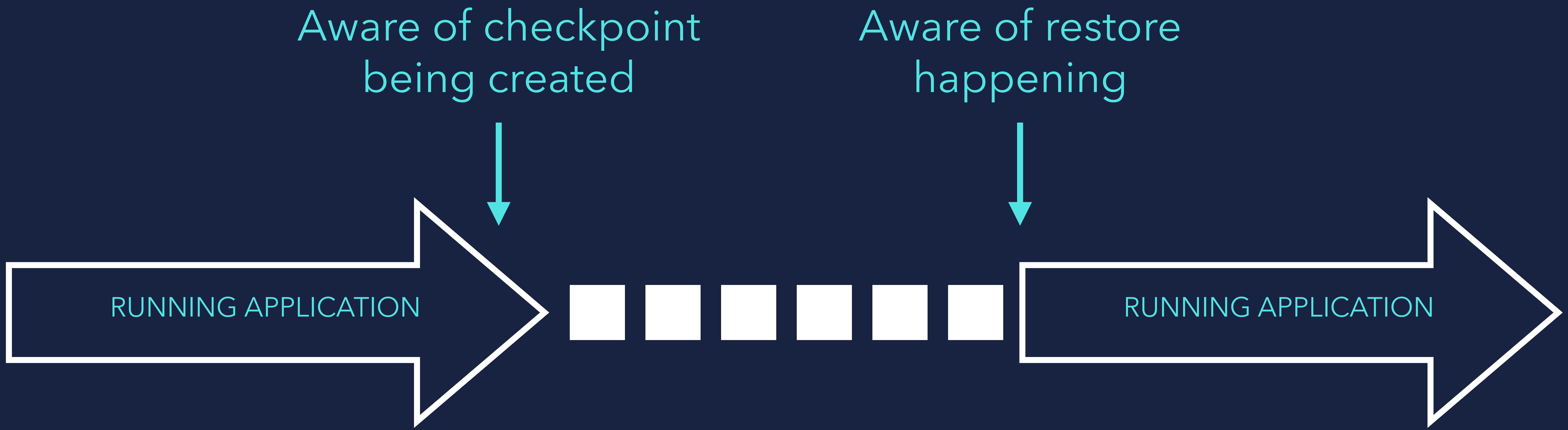
Coordinated Restore at Checkpoint

azul

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)

CRaC

Additional command line parameters

START

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

RESTORE

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

openjdk.org/projects/crac

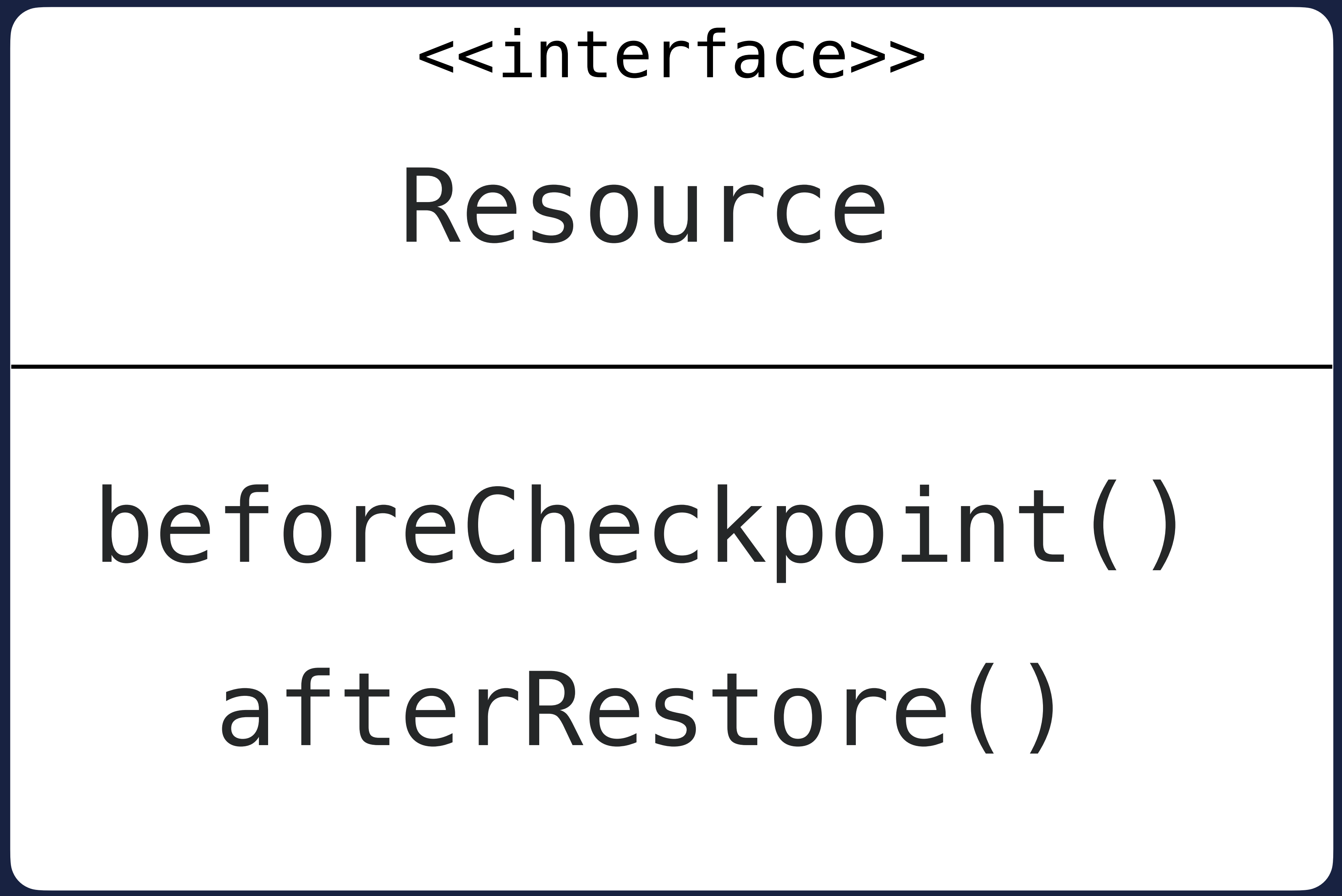
Lead by Anton Kozlov (Azul)

CRaC API

azul

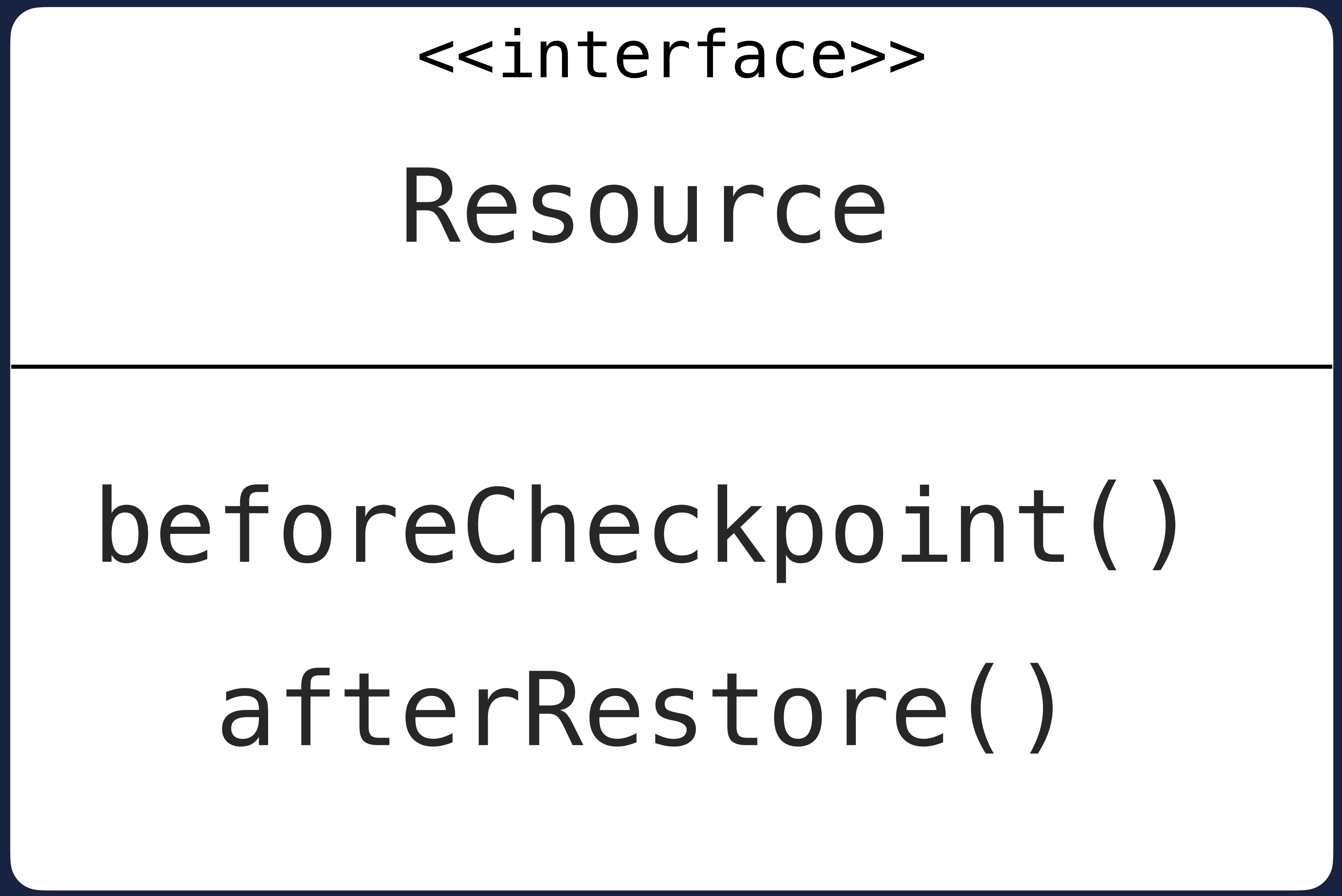
CRaC API

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



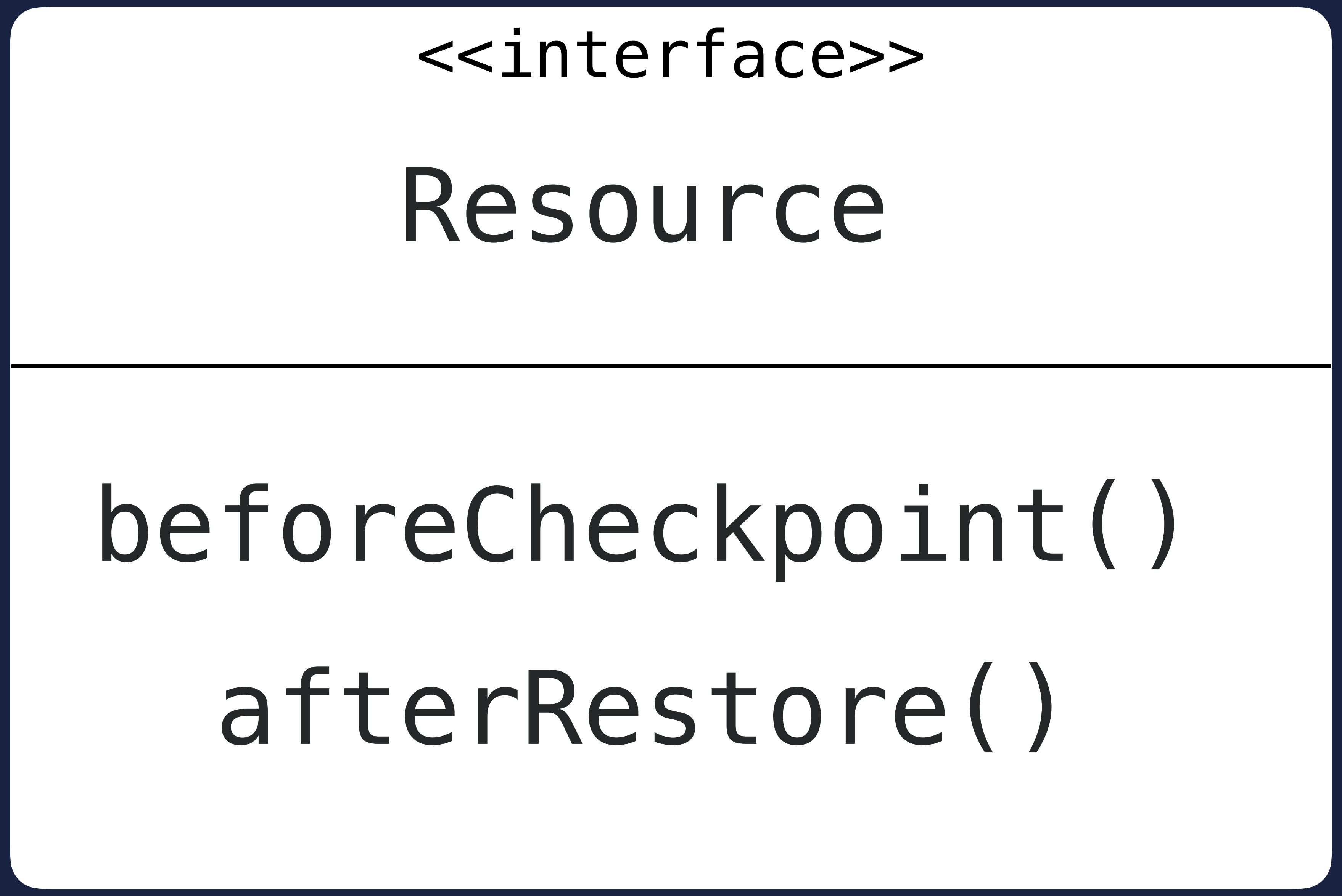
CRaC API

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



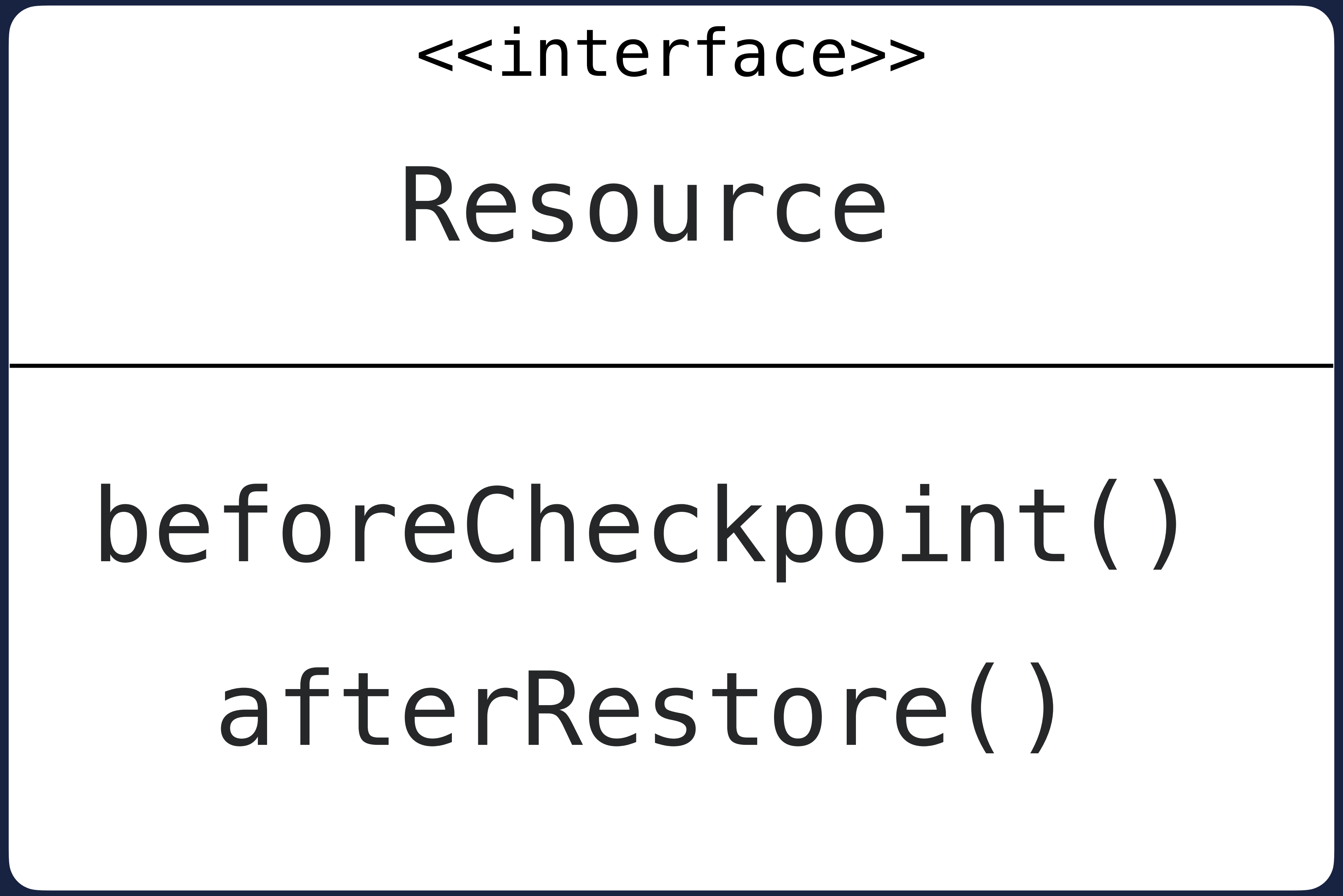
CRaC API

- 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



CRaC API

- 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)



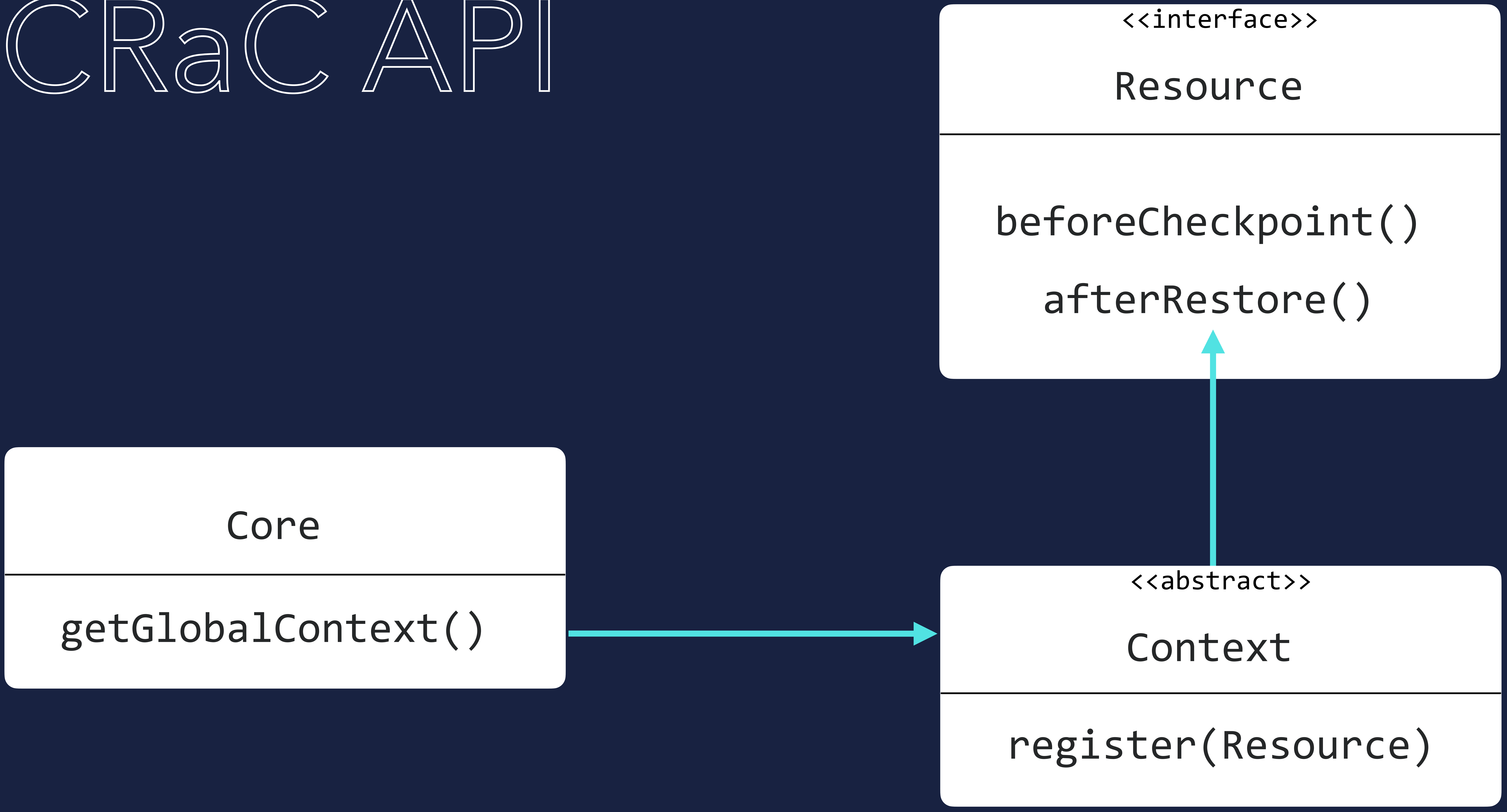
CRaC API

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

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 method Core.getGlobalContext()

CRaC API



azul

CREATING A CHECKPOINT

azul

CREATING A CHECKPOINT

FROM THE COMMAND LINE:

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

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

azul

CREATING A CHECKPOINT

FROM THE CODE:

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

WHEN ?

azul

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

azul

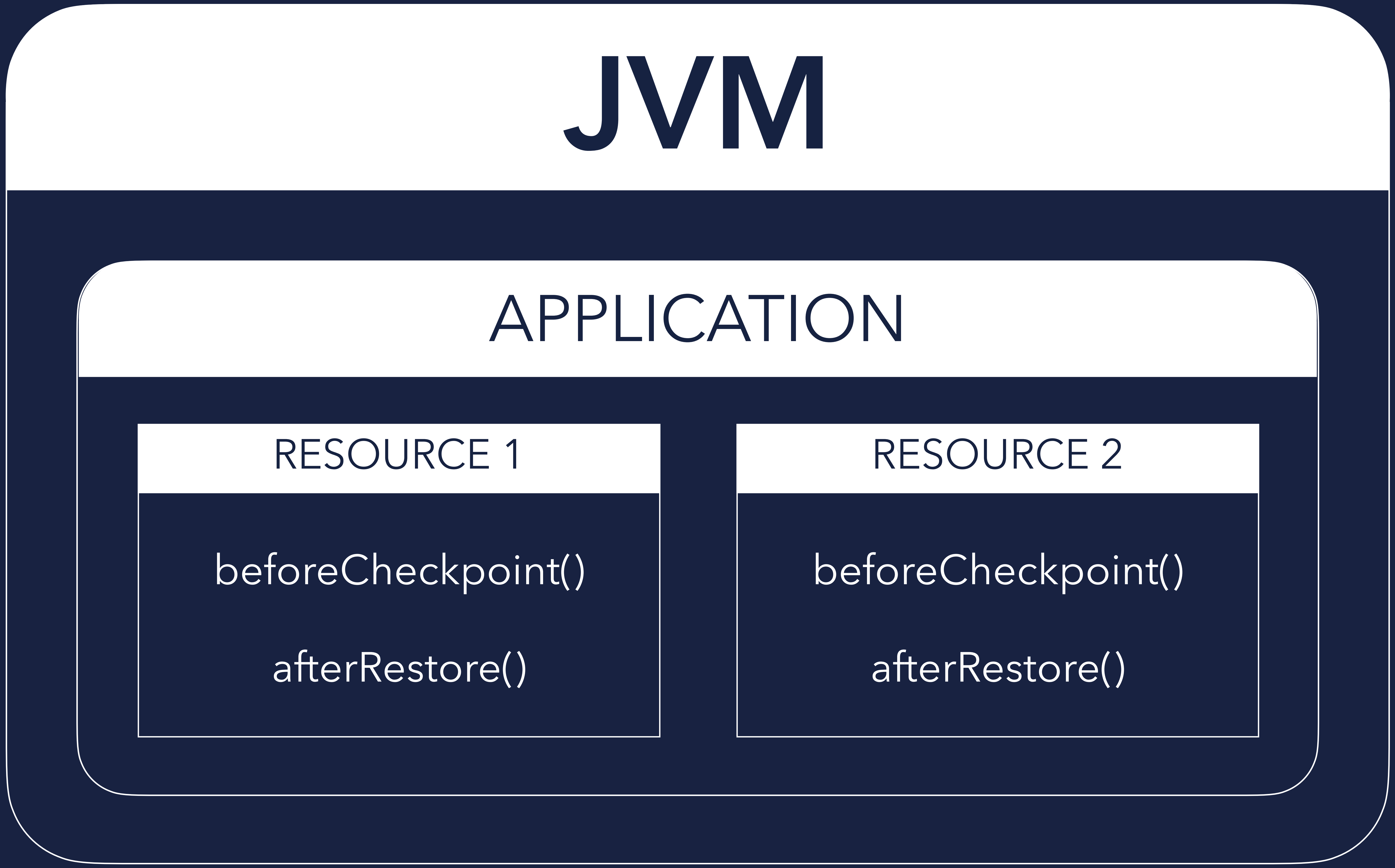
CRaC OVERVIEW

JVM

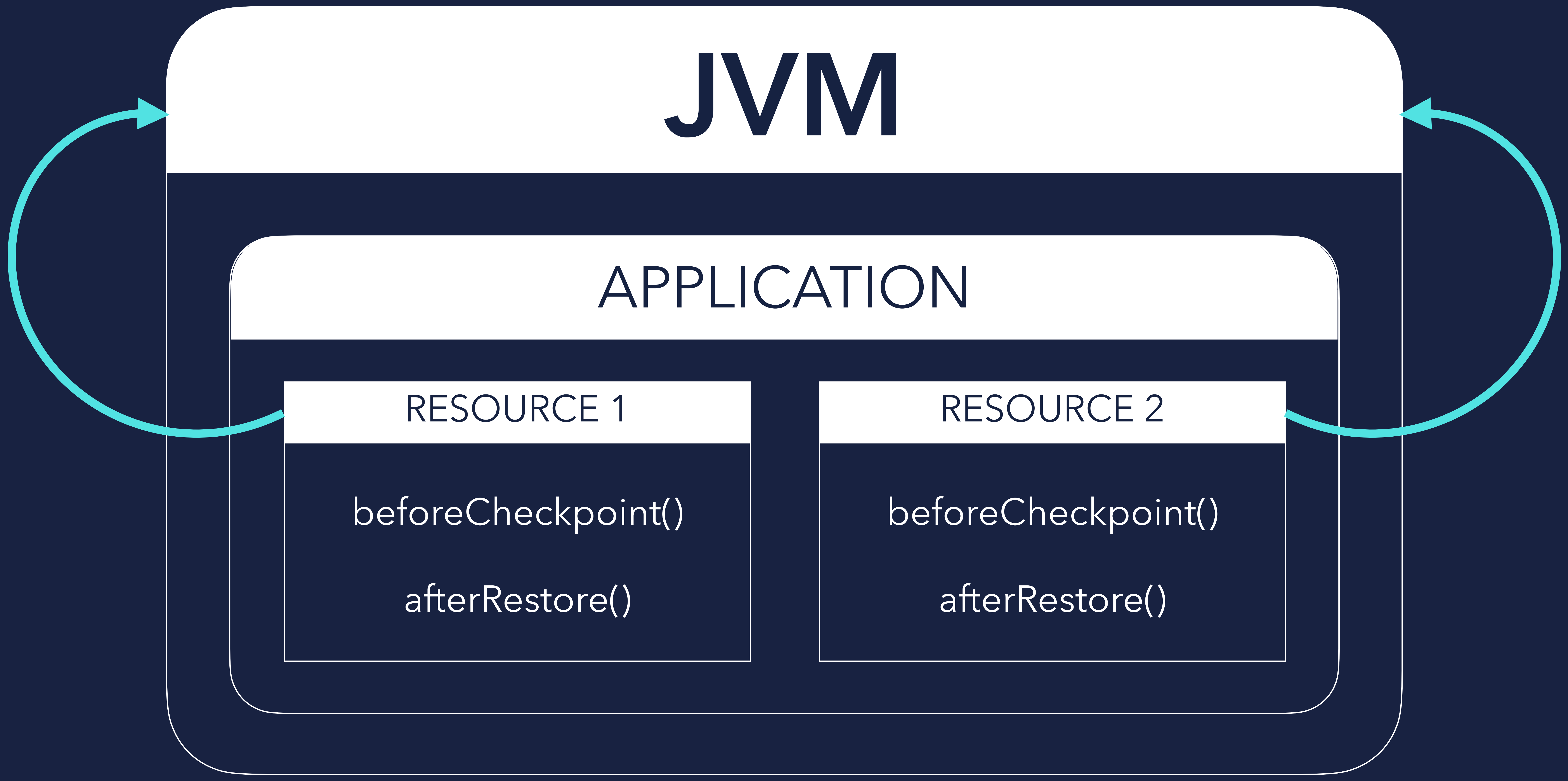
CRaC OVERVIEW

JVM startup...

CRaC OVERVIEW



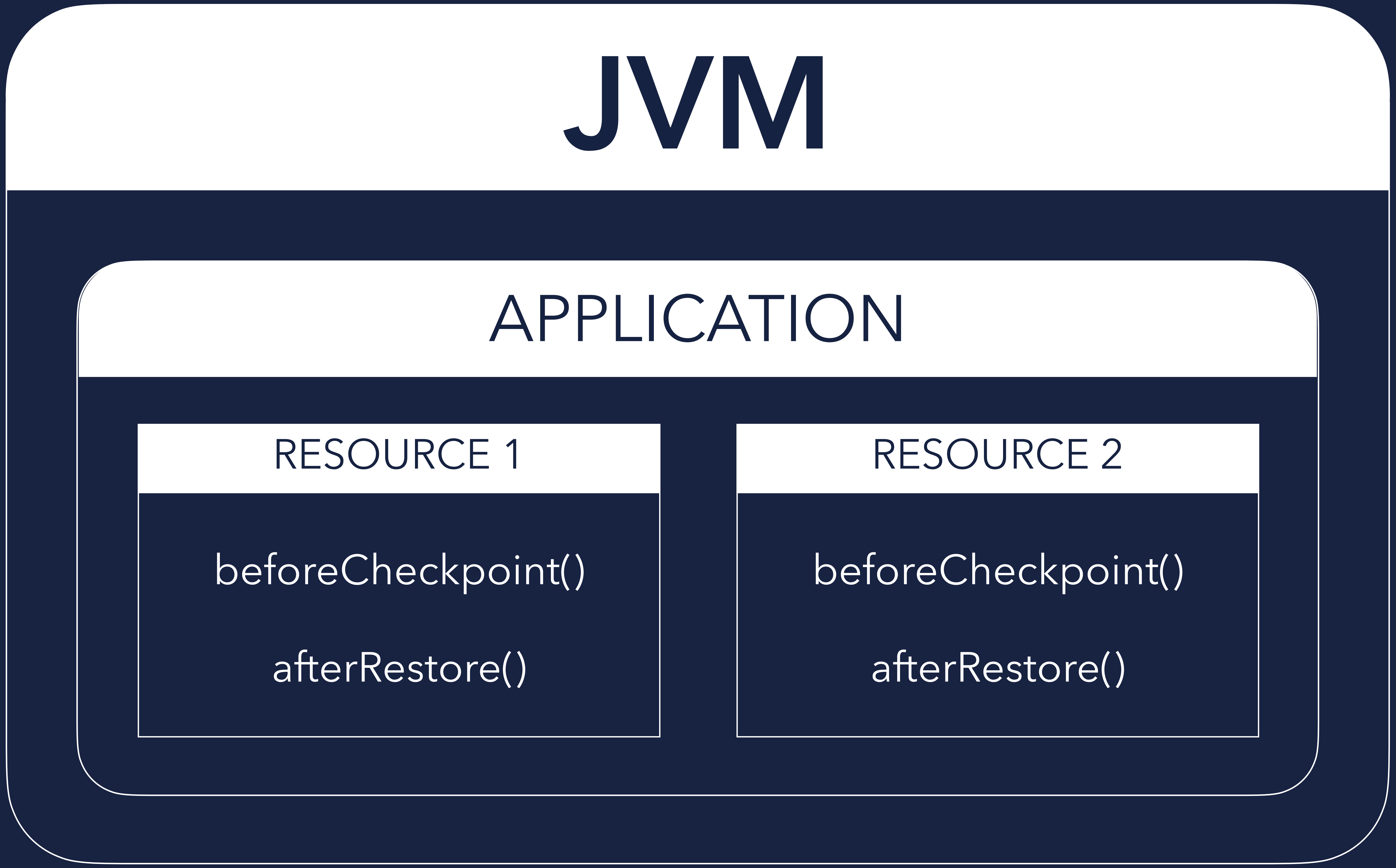
CRaC OVERVIEW



Register resources
in global context

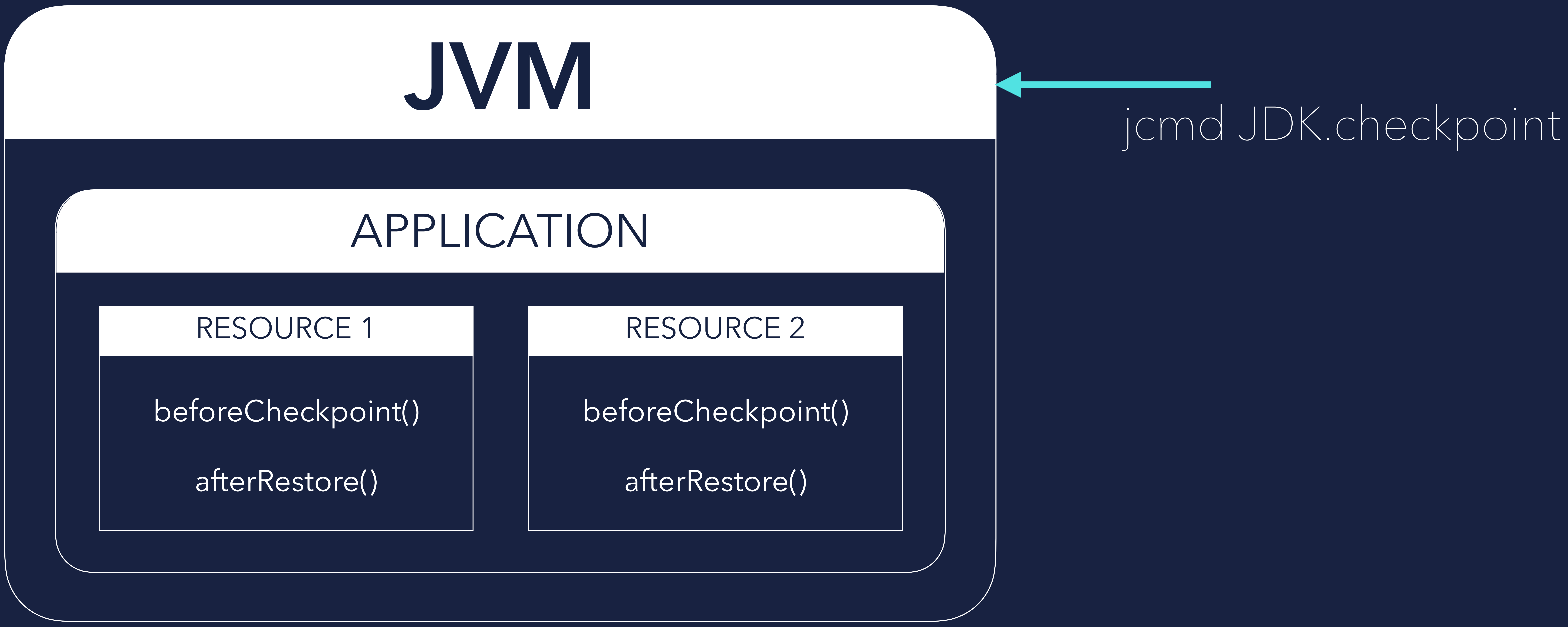
azul

CRaC OVERVIEW



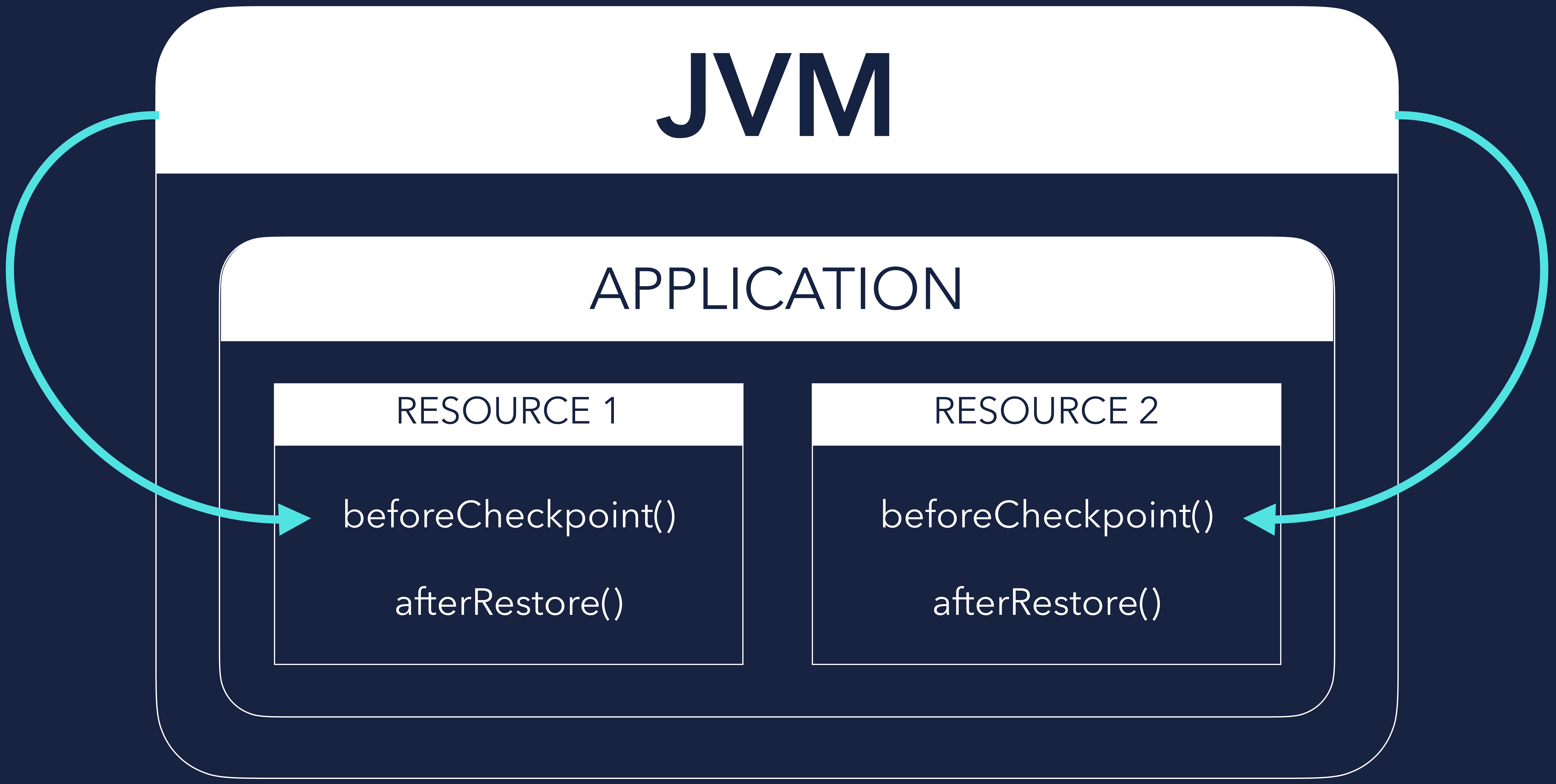
Warmup the application

CRaC OVERVIEW



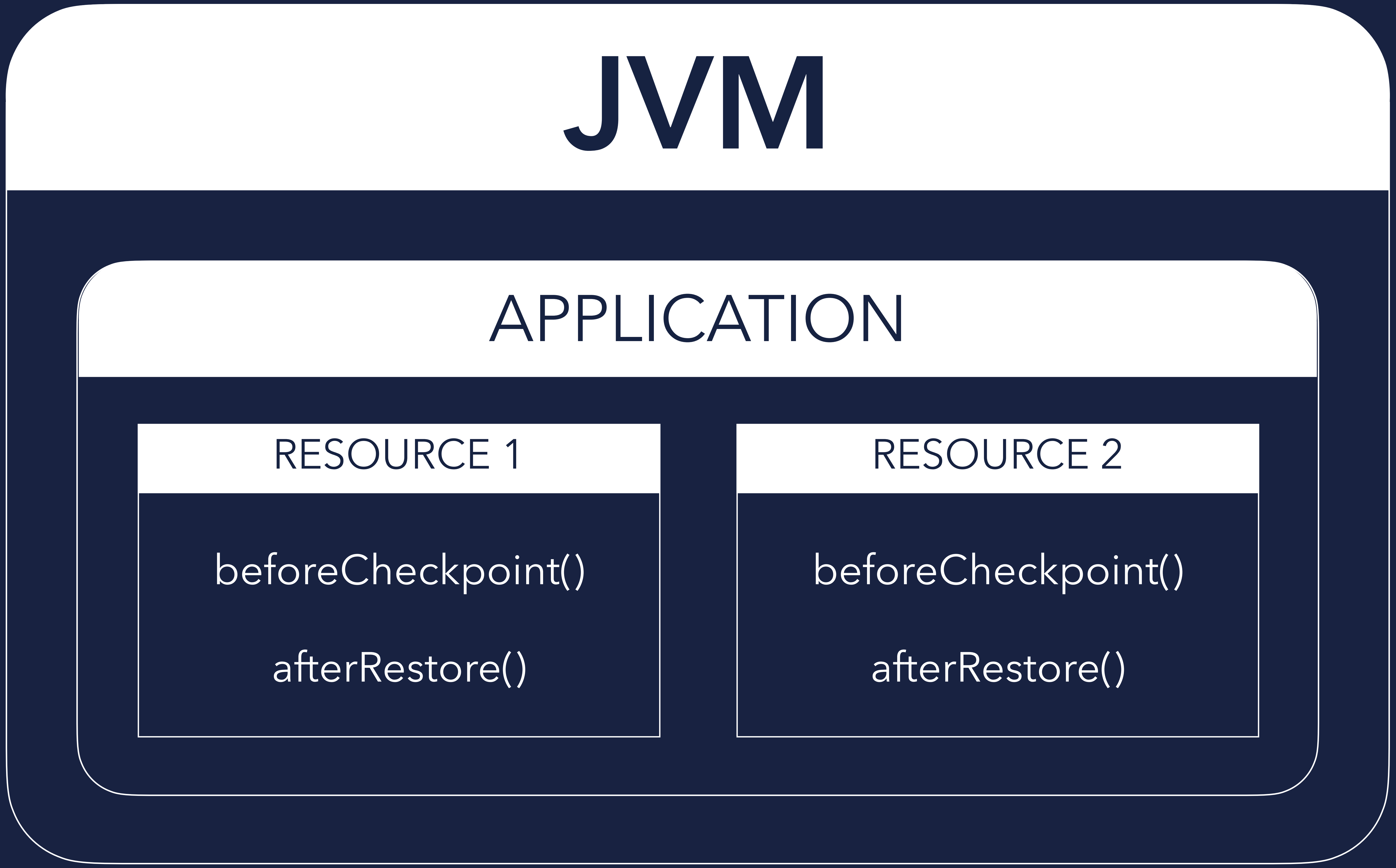
azul

CRaC OVERVIEW



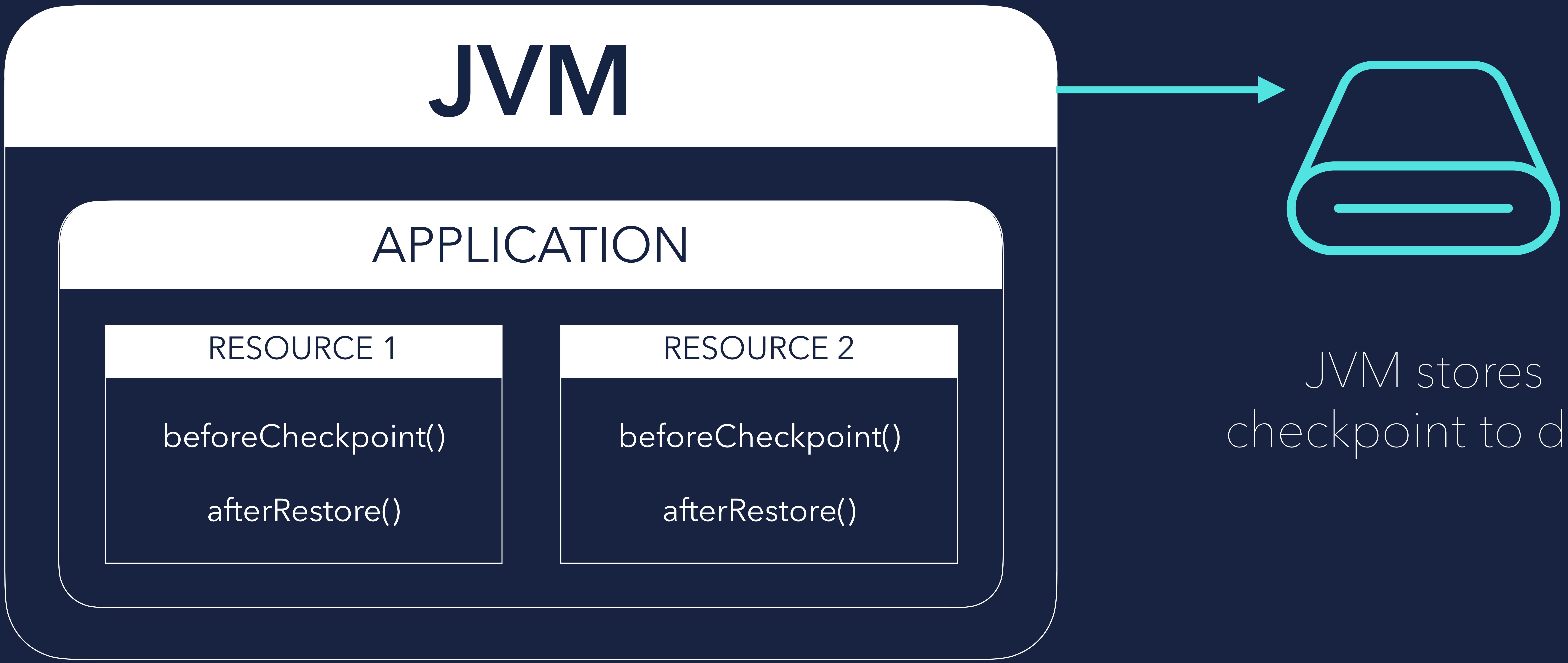
JVM notifies
the resources

CRaC OVERVIEW



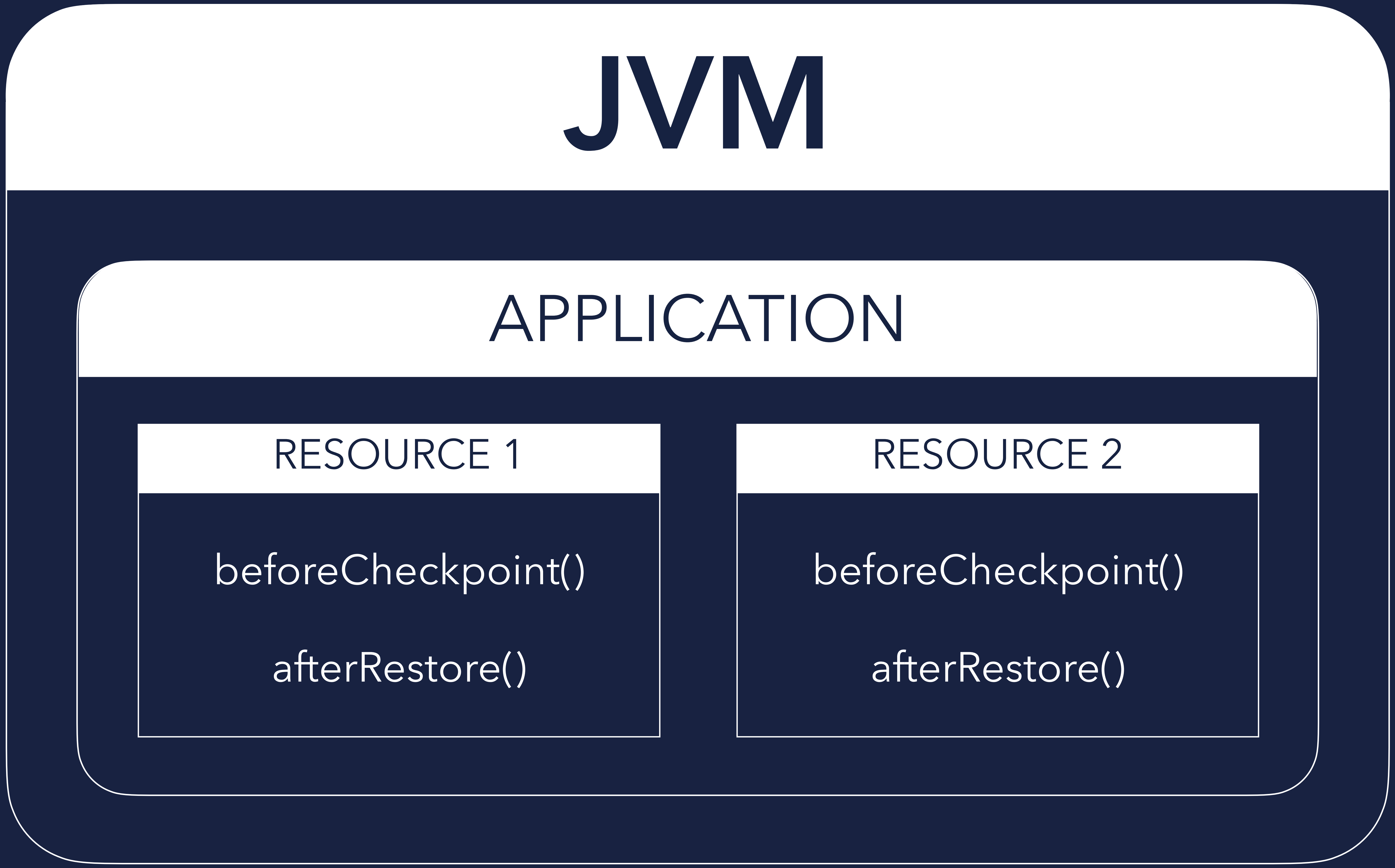
Application closes open resources

CRaC OVERVIEW



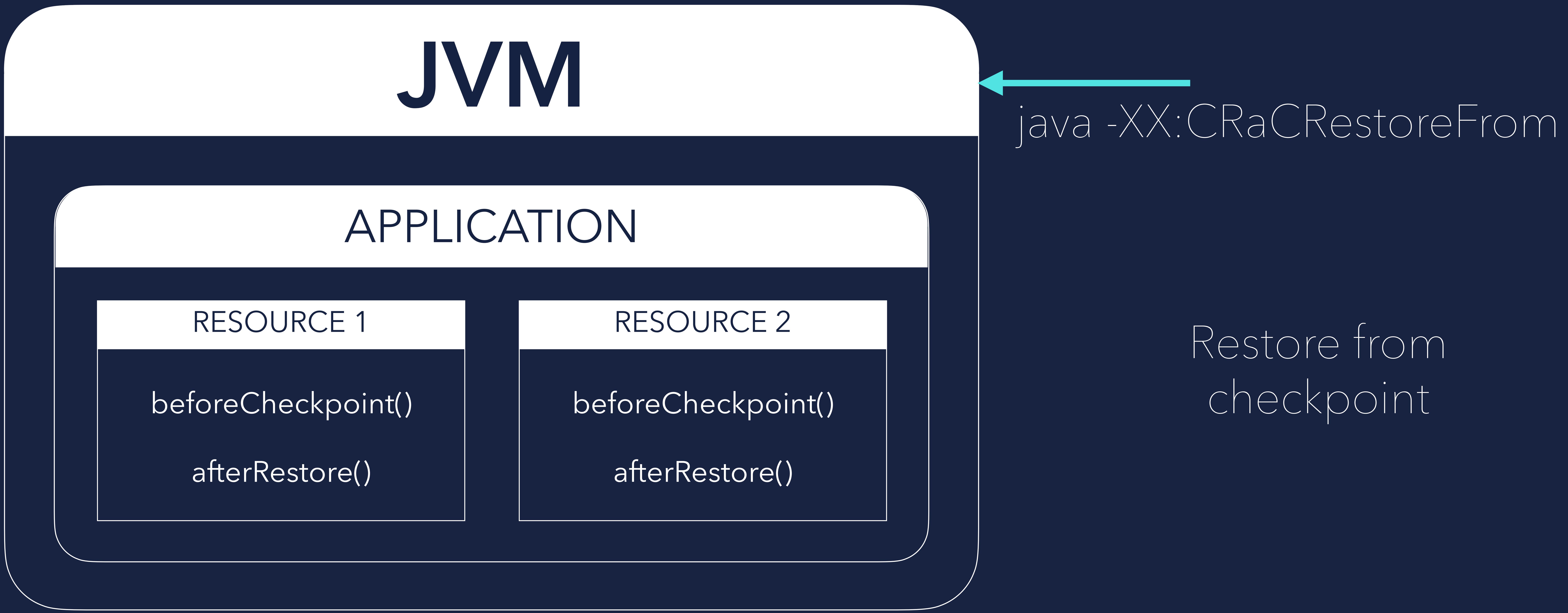
JVM stores checkpoint to disc

CRaC OVERVIEW



Time...

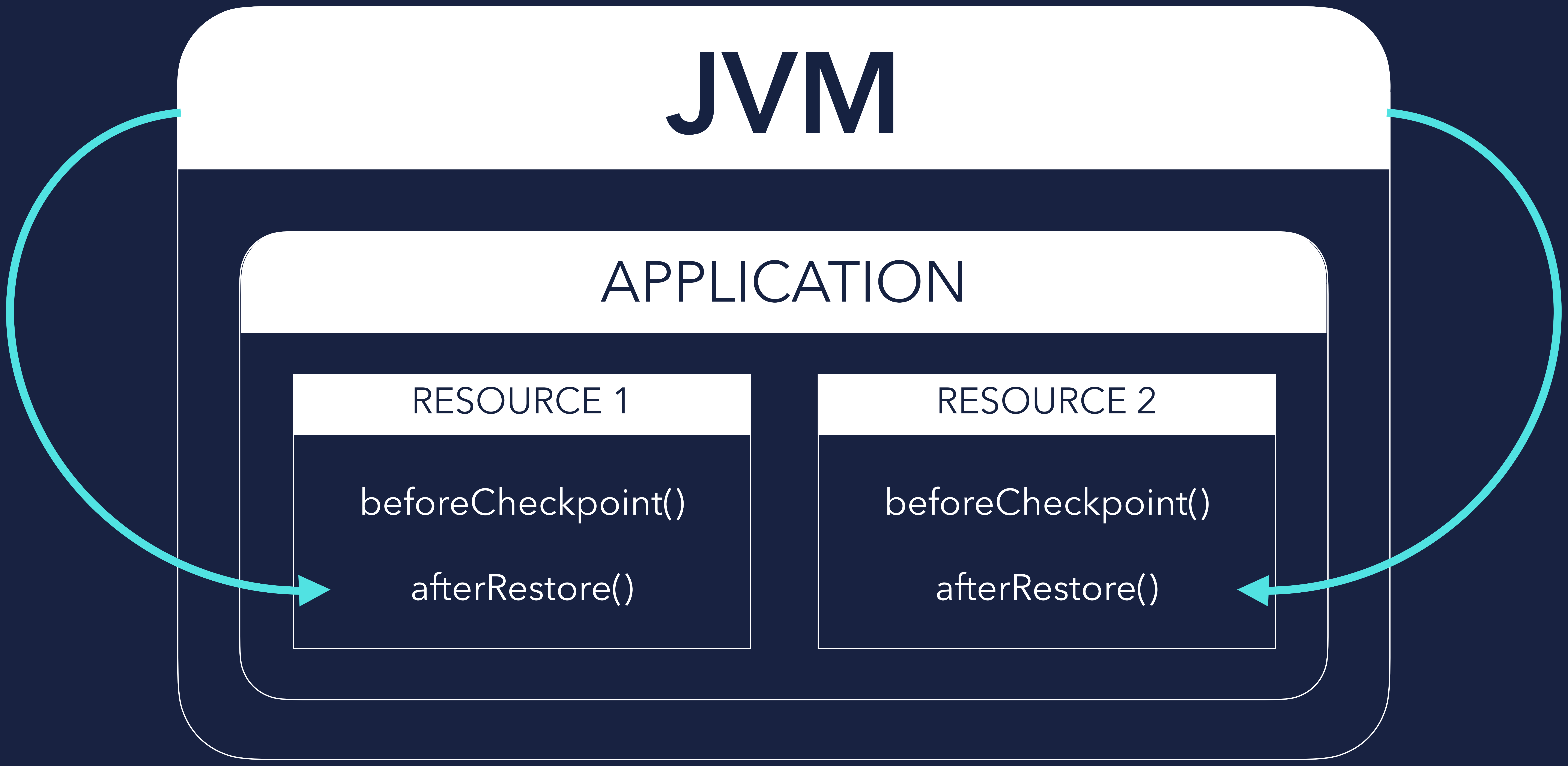
CRaC OVERVIEW



Restore from checkpoint

azul

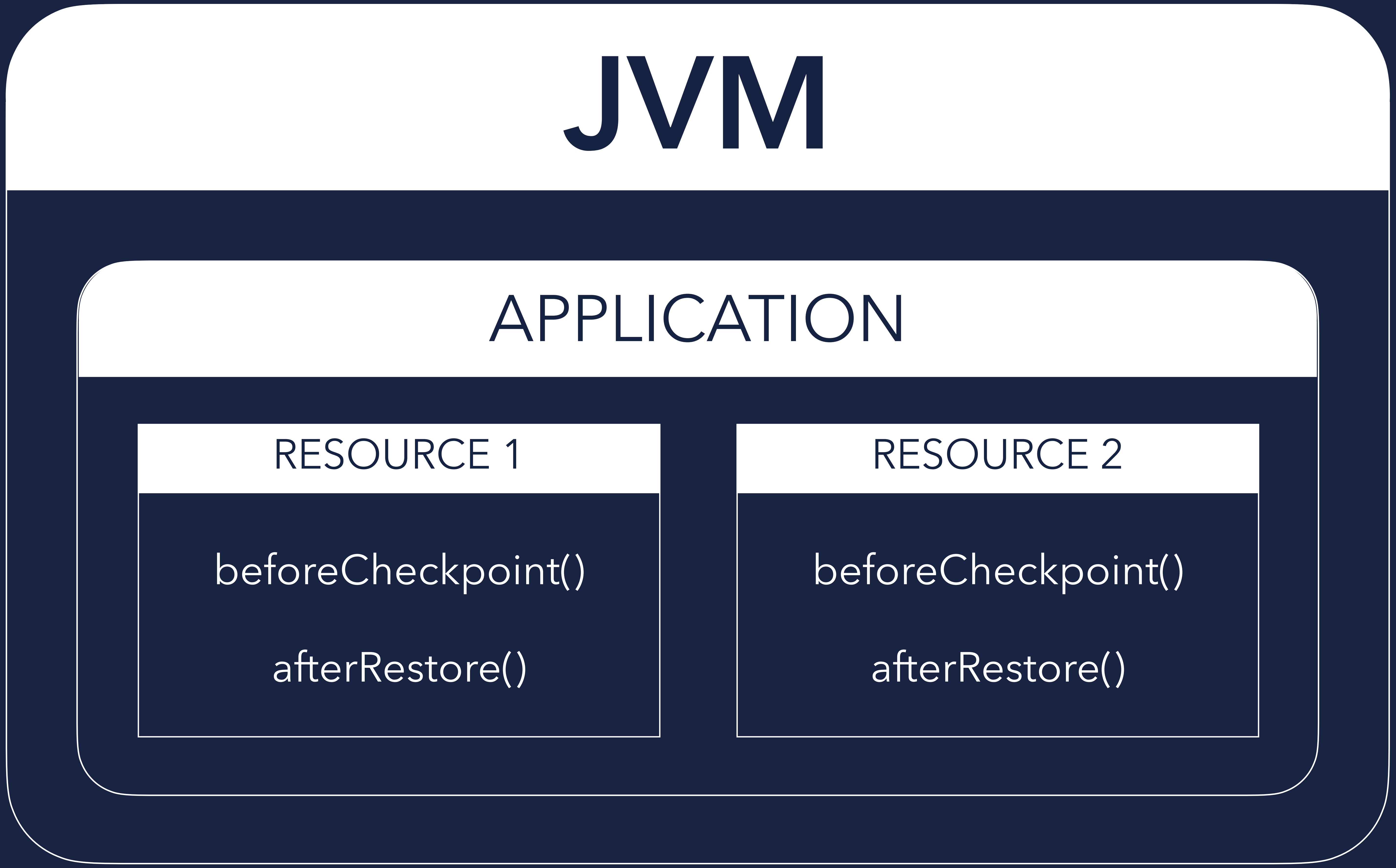
CRaC OVERVIEW



JVM notifies
the resources

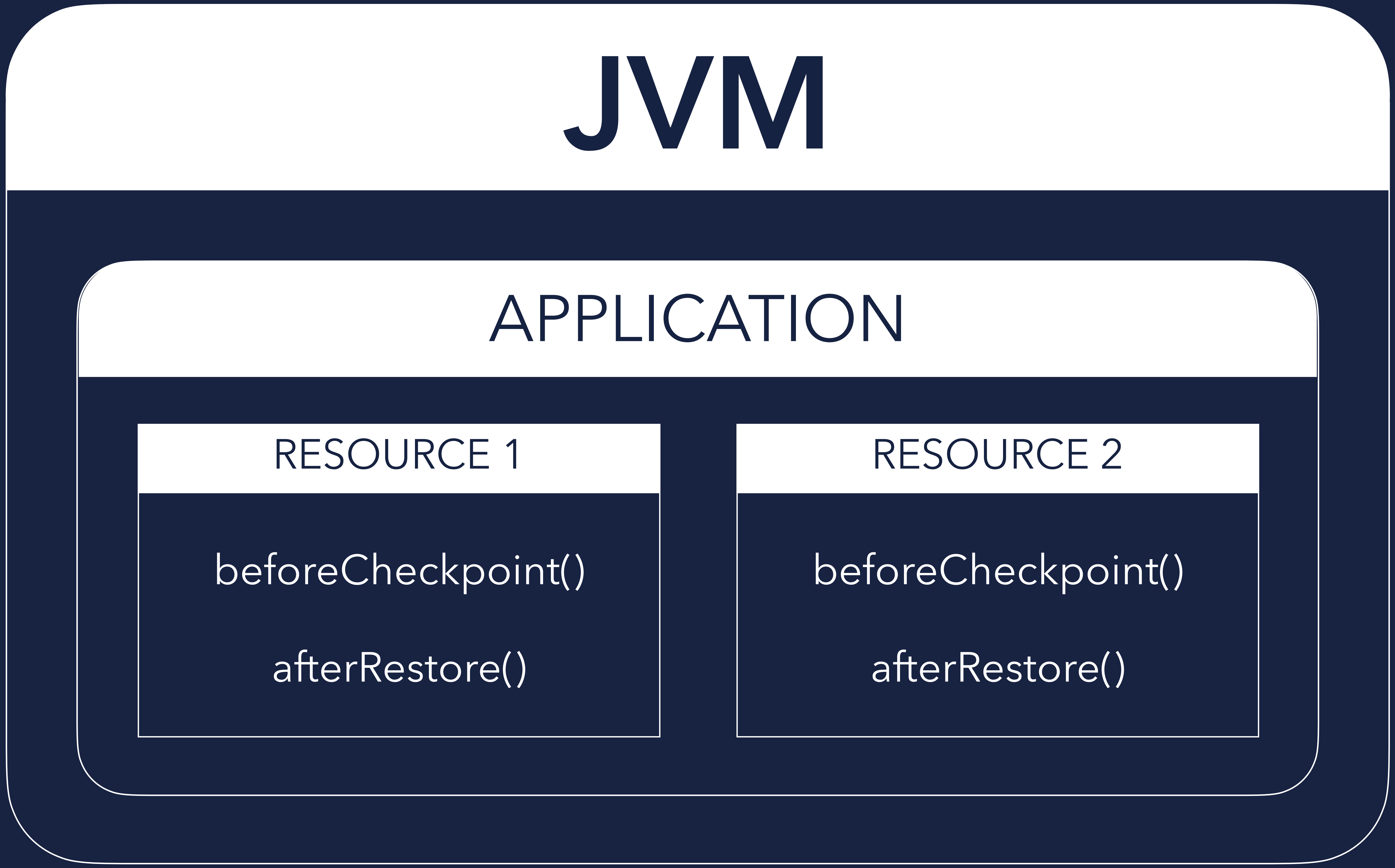
azul

CRaC OVERVIEW



Application
re-open resources

CRaC OVERVIEW



No JVM startup and
no application warmup !!!

TYPICAL 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 / AARCH64

azul

WINDOWS
MACOS ?

azul

ORG.CRAC

azul

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'
```

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

ORG.CRAC

github.com/CRaC/org.crac

azul

COMPATIBILITY

azul

COMPATIBILITY…

- Upgrade (Haswell -> restore: Ice Lake, no problem)

COMPATIBILITY...

- Upgrade (Haswell -> restore: Ice Lake, no problem)
- Downgrade (Ice Lake -> restore: Haswell, problematic)

COMPATIBILITY...

- Upgrade (Haswell -> restore: Ice Lake, no problem)
- Downgrade (Ice Lake -> restore: Haswell, problematic)
- Solved in CRaC by specific flag (little drop in performance)

COMPATIBILITY...

- Upgrade (Haswell -> restore: Ice Lake, no problem)
- Downgrade (Ice Lake -> restore: Haswell, problematic)
- Solved in CRaC by specific flag (little drop in performance)
- Node groups stick to same cpu architecture

COMPATIBILITY...

- Upgrade (Haswell -> restore: Ice Lake, no problem)
- Downgrade (Ice Lake -> restore: Haswell, problematic)
- Solved in CRaC by specific flag (little drop in performance)
- Node groups stick to same cpu architecture
- Virtualized Linux environments work on all OS's (as long as cpu architecture is x64/aarch64)

azul

FRAMEWORK SUPPORT ?

azul

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...

azul

SPRINGBOOT 3.2 PETCLINIC

azul

NORMAL START

azul

NORMAL START

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

START APPLICATION

azul

NORMAL START

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

              |\      _,,,--,,_
             /,`.-'`'   ._  \-;;,_
  _______ __|,4-  ) )_   .;.(__`'-'__     ___ __    _ ___ _______
 |       | '---''(_/._)-'(_\_)   |   |   |   |  |  | |   |       |
 |    _  |    ___|_     _|       |   |   |   |   |_| |   |       | __ _ _
 |   |_| |   |___  |   | |       |   |   |   |       |   |       | \ \ \ \
 |    ___|    ___| |   | |      _|   |___|   |  _    |   |      _|  \ \ \ \
 |   |   |   |___  |   | |     |_|       |   | | |   |   |     |_    ) ) ) )
 |___|   |_______| |___| |_______|_______|___|_|  |__|___|_______|  / / / /
 ==================================================================/_/_/_/

:: 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 4997ms with PID: 3839
```

azul

START FROM AUTO CHECKPOINT

azul

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 265ms with PID: 15261
```

RESTORE FROM CHECKPOINT

START FROM MANUAL CHECKPOINT

azul

MANUAL CHECKPOINT

- Start application with `-XX:CracCheckpointTo=Path`

MANUAL CHECKPOINT

- Start application with `-XX:CracCheckpointTo=Path`
- Warm up your application

azul

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

azul

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 4997ms with PID: 3839
```

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

CREATE CHECKPOINT

azul

MANUAL CHECKPOINT

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

RESTORE FROM CHECKPOINT

azul

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 59 ms)
```

spring

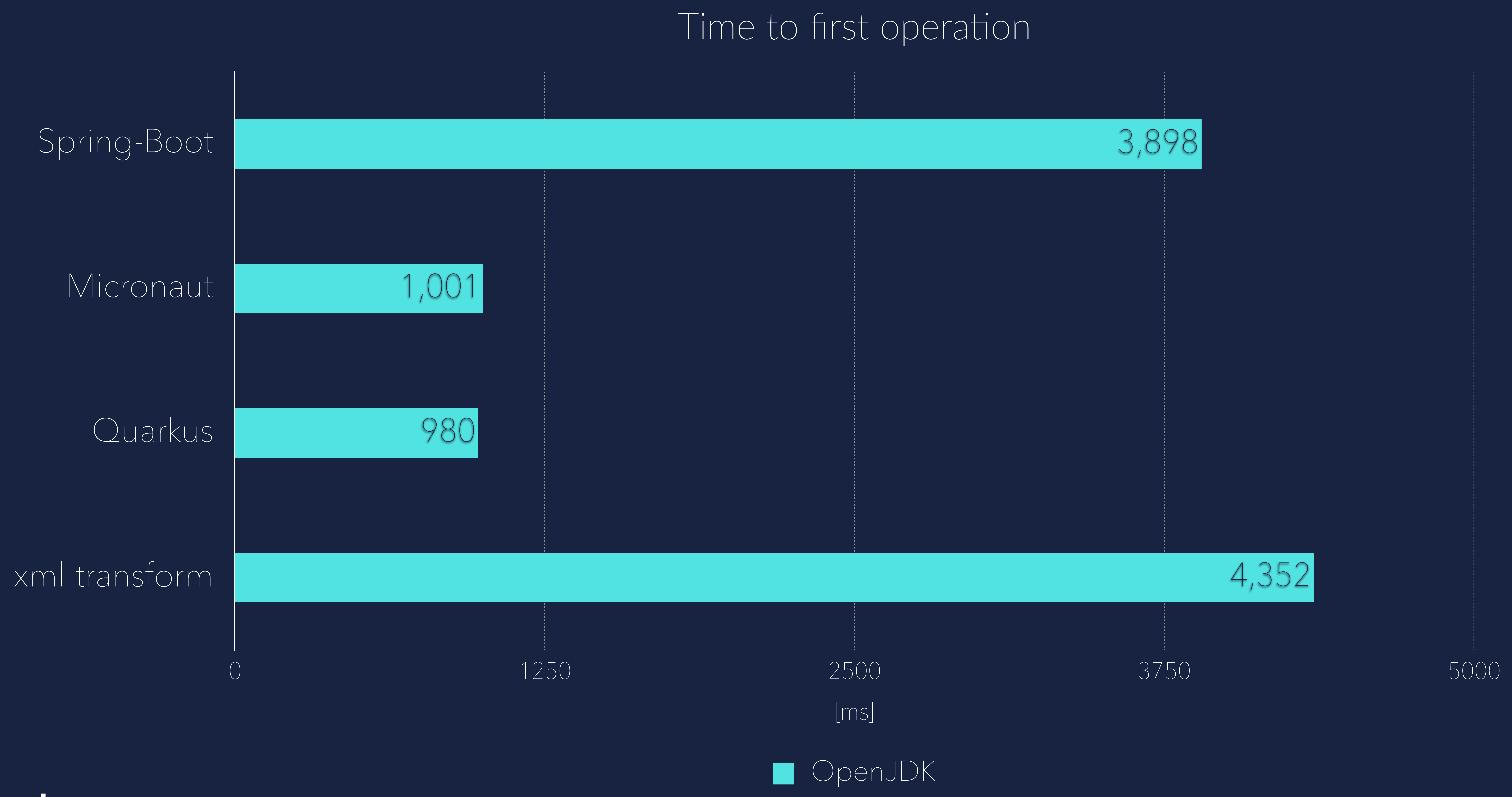
https://github.com/
HanSolo/spring-petclinic

azul

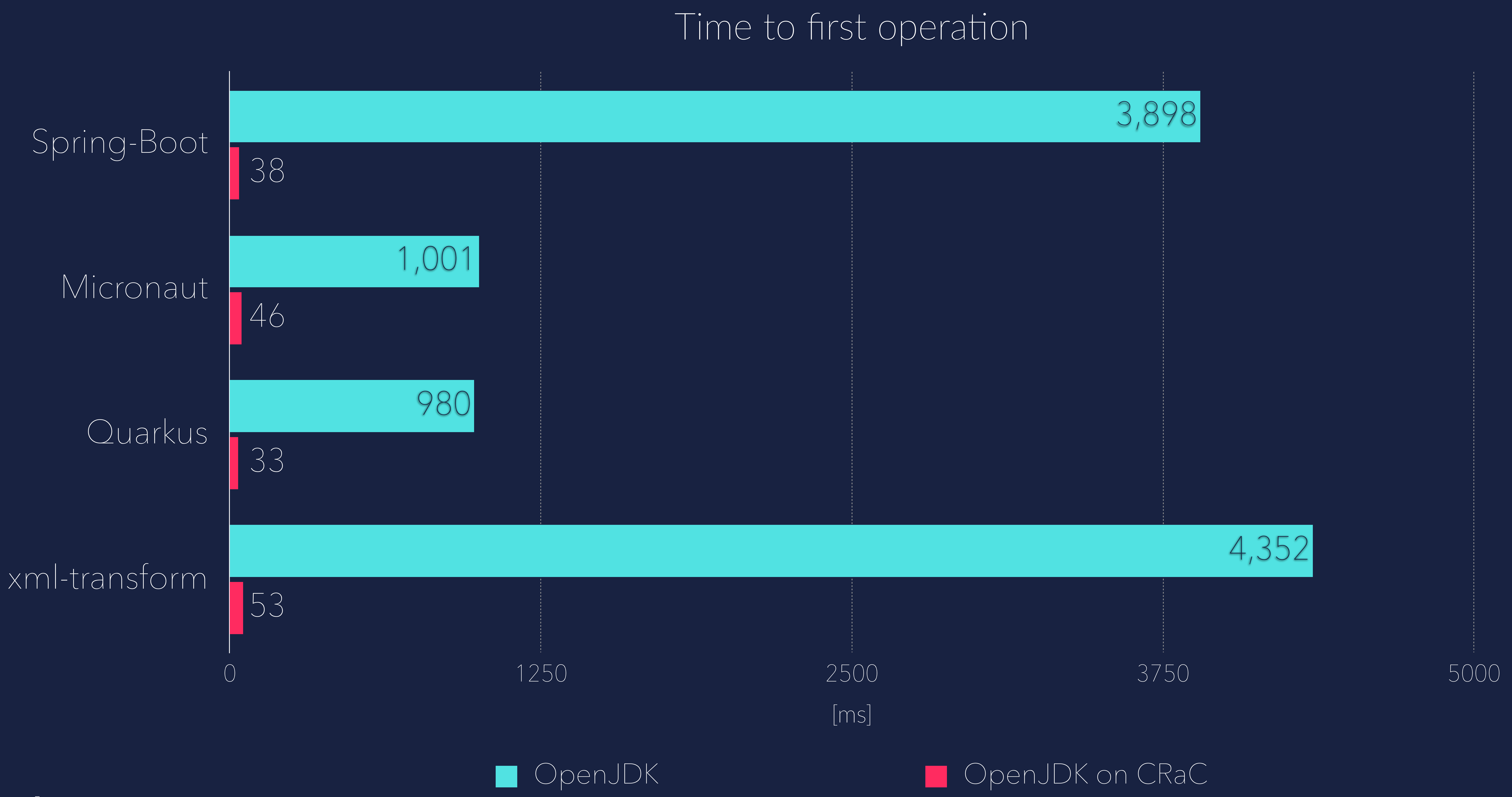
OK...BUT

HOW GOOD IS IT...?

azul

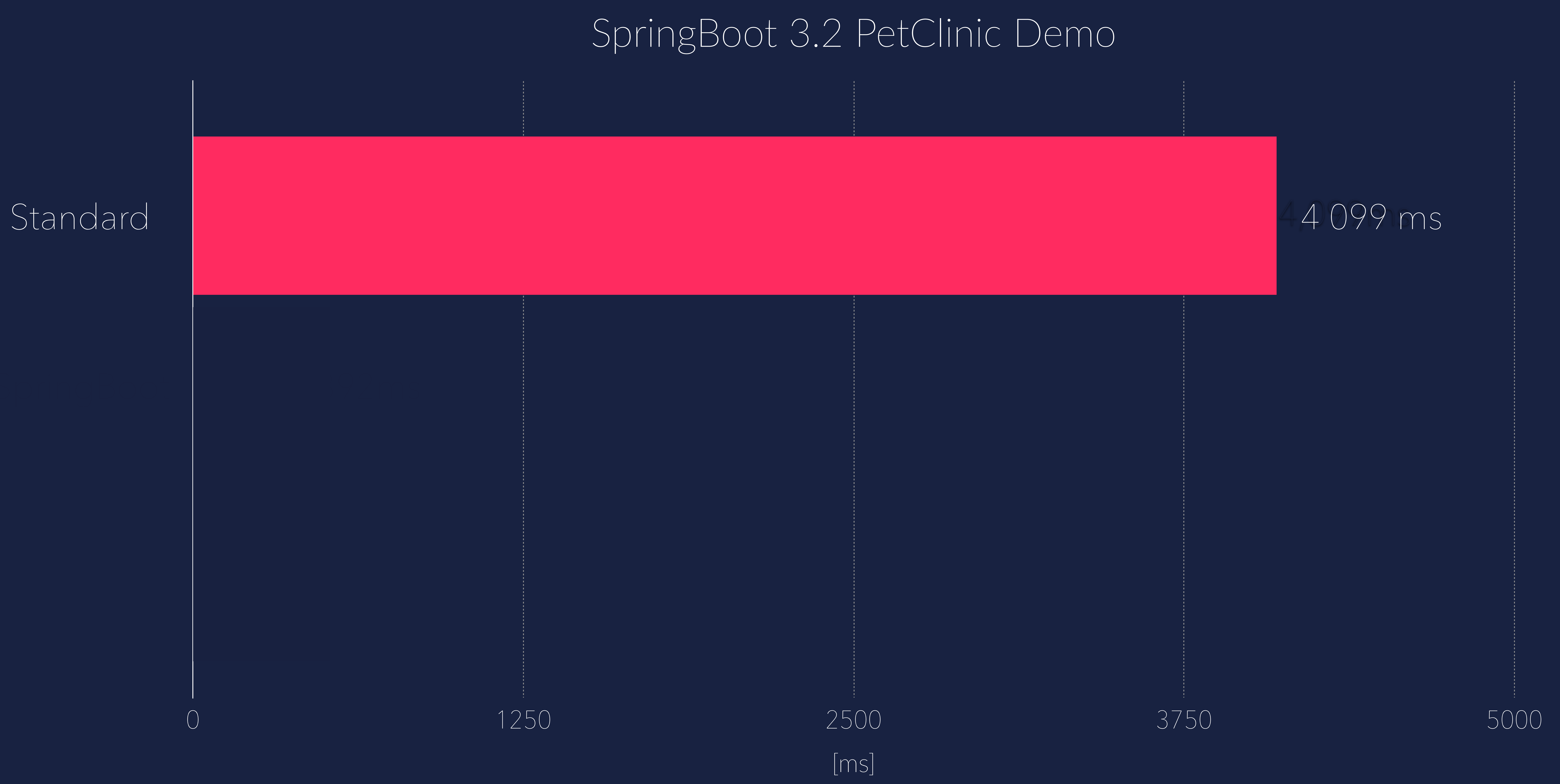

Time to first operation
Spring-Boot
3,898
Micronaut
1,001
Quarkus
980
xml-transform
4,352
0
1250
2500
3750
5000
[ms]
OpenJDK

azul



Time to first operation
Spring-Boot
3,898
38
Micronaut
1,001
46
Quarkus
980
33
xml-transform
4,352
53
0
1250
2500
3750
5000
[ms]
OpenJDK
OpenJDK on CRaC

azul

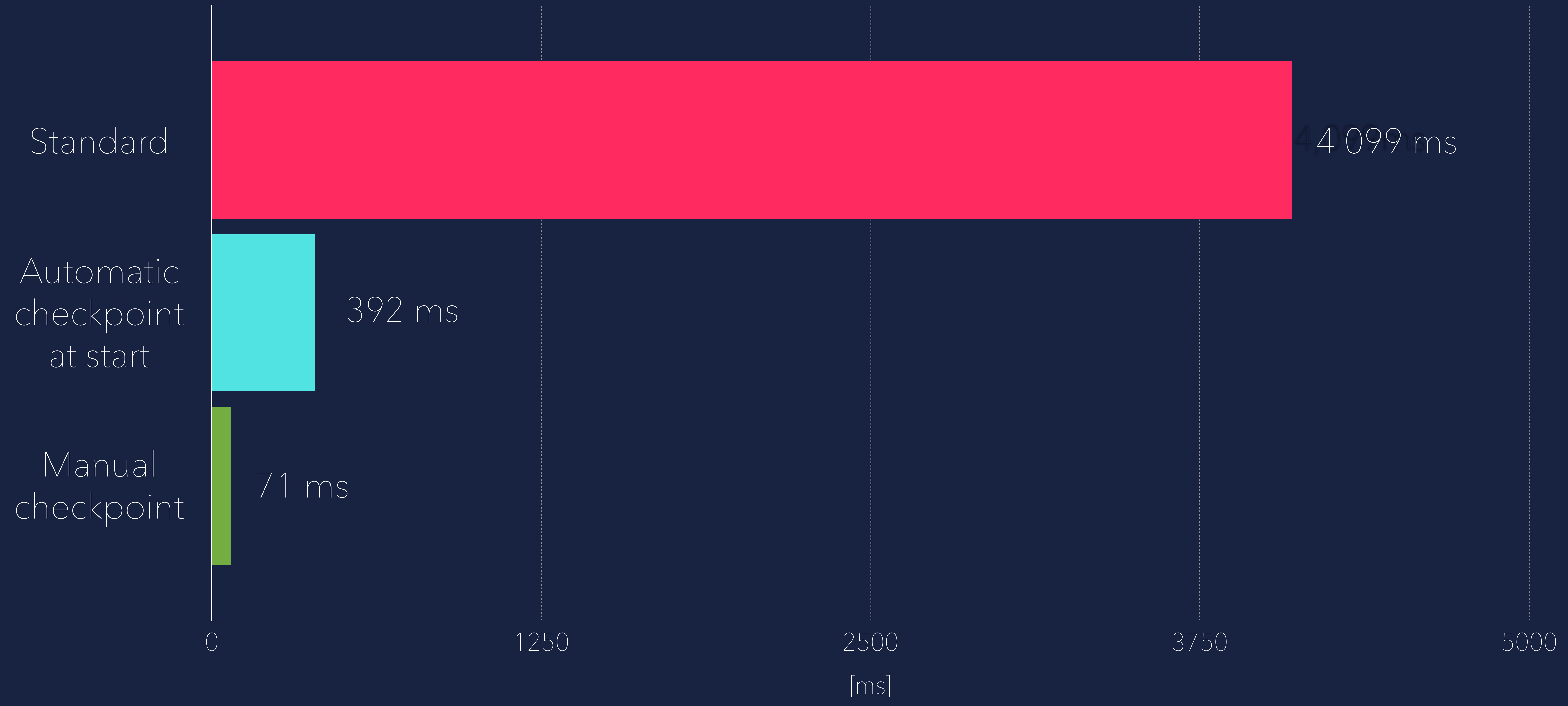


SpringBoot 3.2 PetClinic Demo
Standard
4 099 ms
0
1250
2500
3750
5000
[ms]

azul


SpringBoot 3.2 PetClinic Demo
Standard
4 099 ms
Automatic checkpoint at start
392 ms
0
1250
2500
3750
5000
[ms]
azul

SpringBoot 3.2 PetClinic Demo



THE FUTURE...

azul

THE FUTURE...

- Non privileged mode

THE FUTURE...

- Non privileged mode
- Encryption and compression*

*already works

THE FUTURE...

- Non privileged mode
- Encryption and compression*
- Cloud native storage

*already works

THE FUTURE...

- Non privileged mode
- Encryption and compression*
- Cloud native storage
- Checkpoint after restore

*already works

THE FUTURE...

- Non privileged mode
- Encryption and compression*
- Cloud native storage
- Checkpoint after restore
- Full support on Windows and MacOS

*already works

SUMMARY...

azul

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

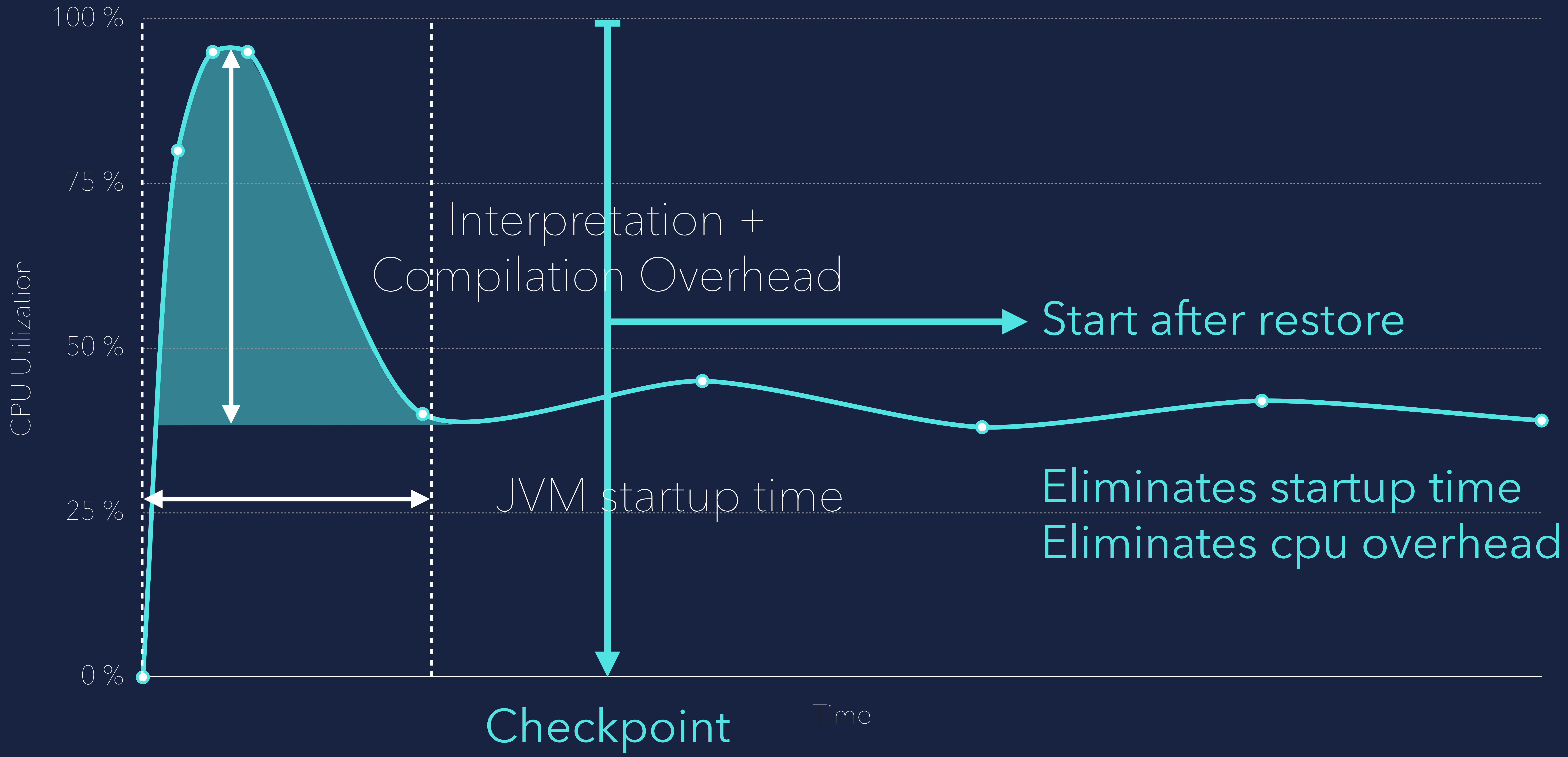
SUMMARY...

- 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



azul

WANNA KNOW MORE ?

azul

INFORMATION...

github.com/CRaC

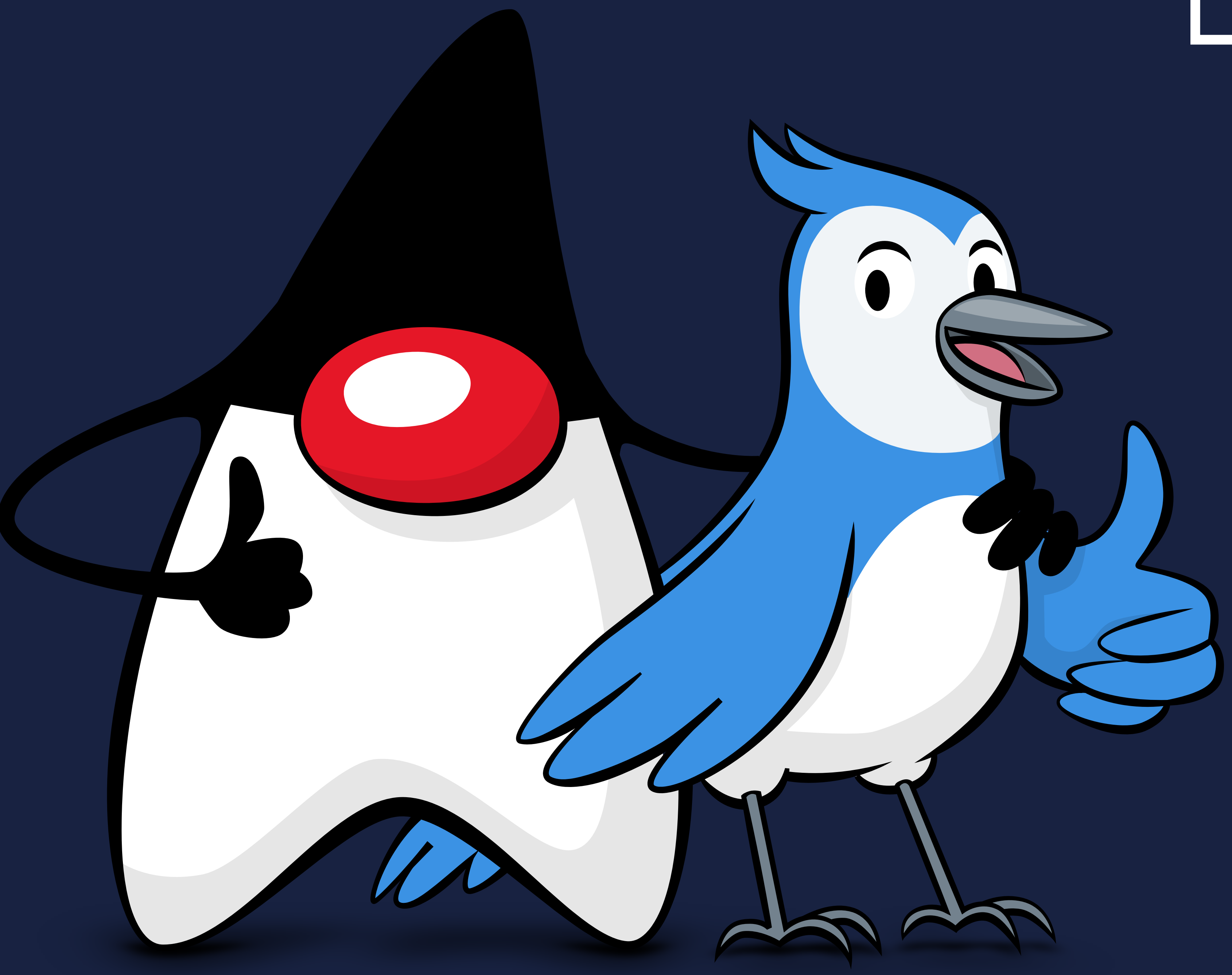
DOWNLOAD...

azul.com

JDK 17.0.9 LINUX X64 / AARCH64

azul

THANK YOU



azul