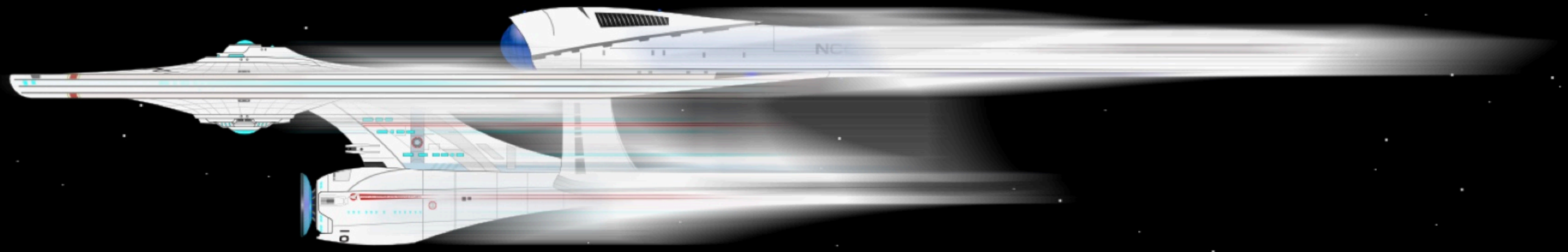
SCOTTY

I NEED WARP SPEED



WAYS TO IMPROVE THE JVM STARTUP / WARMUP

azul

ABOUT ME...

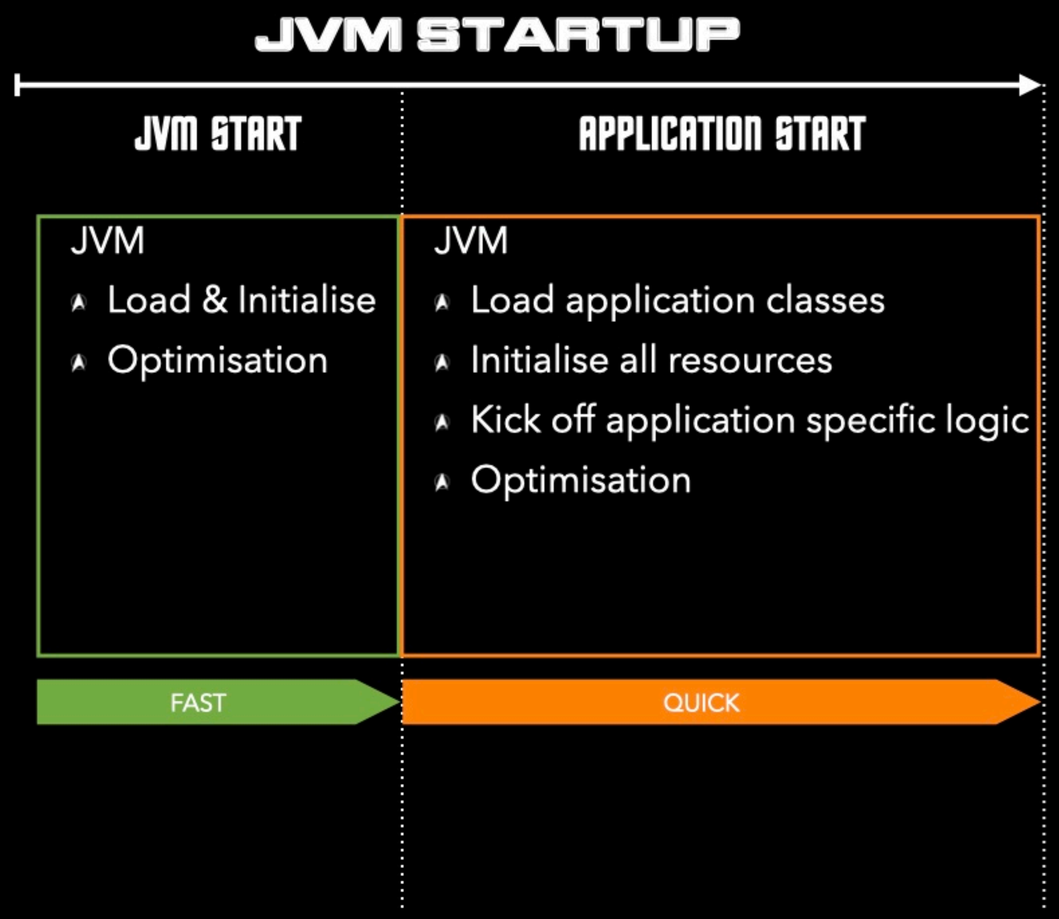
NAME	GERRIT GRUNWALD
CREW	AZUL
RANK	DEVELOPER ADVOCATE
CONTACT	GGRUNWALD@AZUL.COM
DECORATION	JAVA CHAMPION

THE JVM STARTUP...

azul

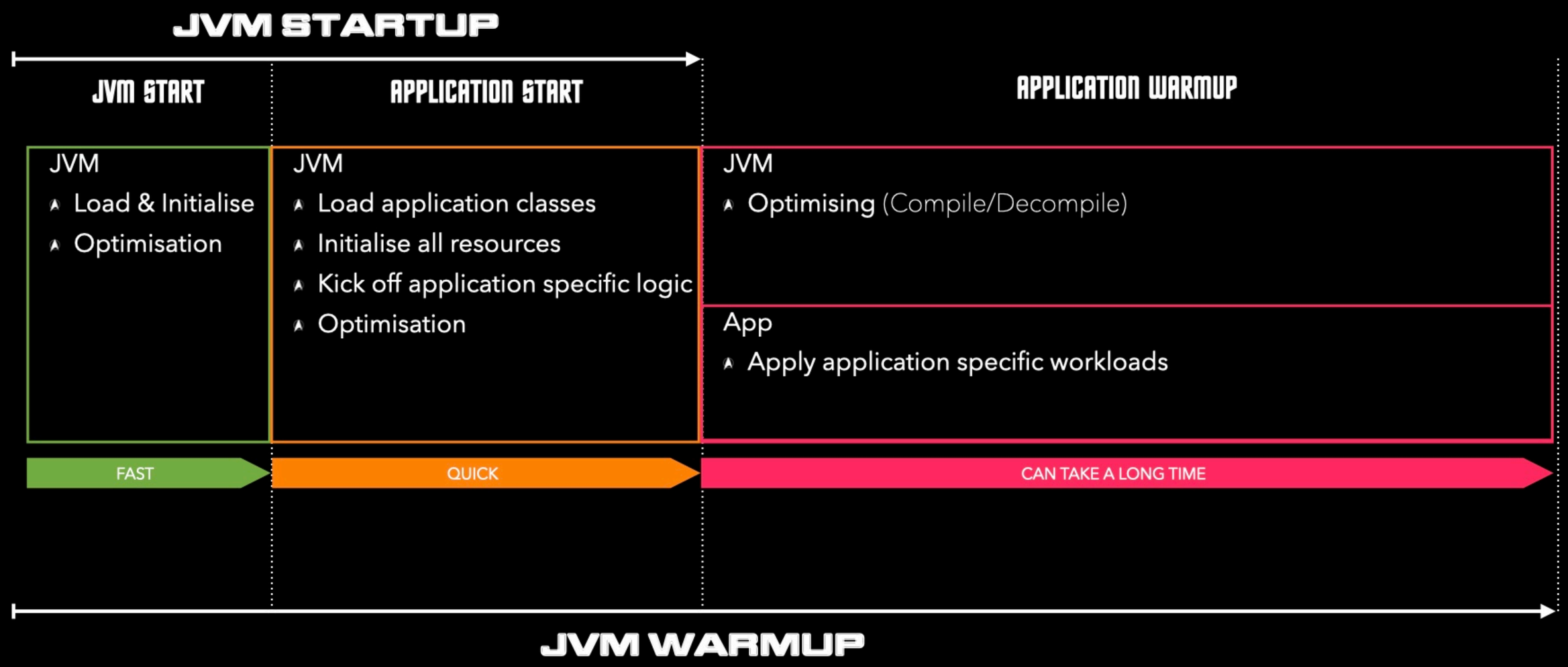
CLARIFICATION

JVM Startup and Warmup

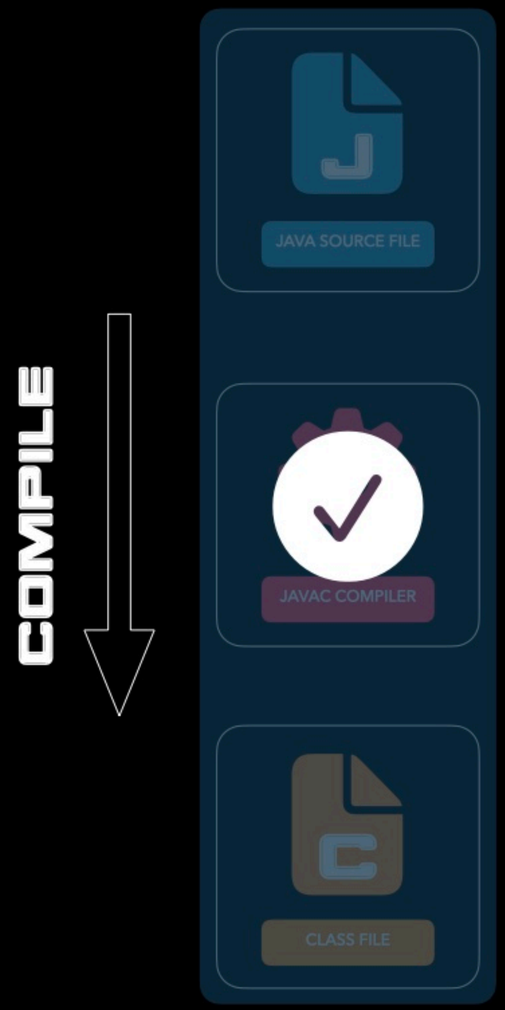


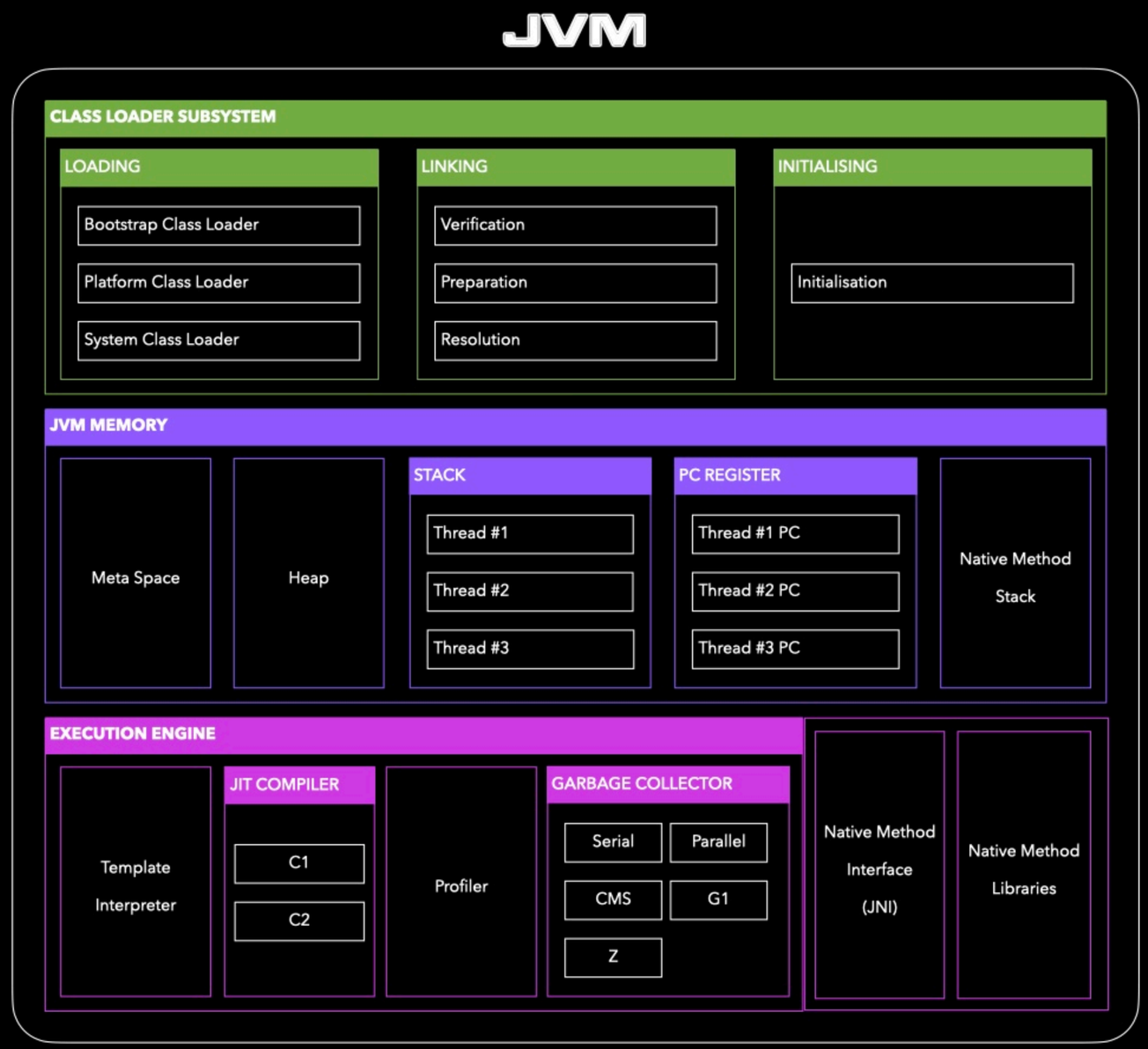
CLARIFICATION

JVM Startup and Warmup



JVM STARTUP





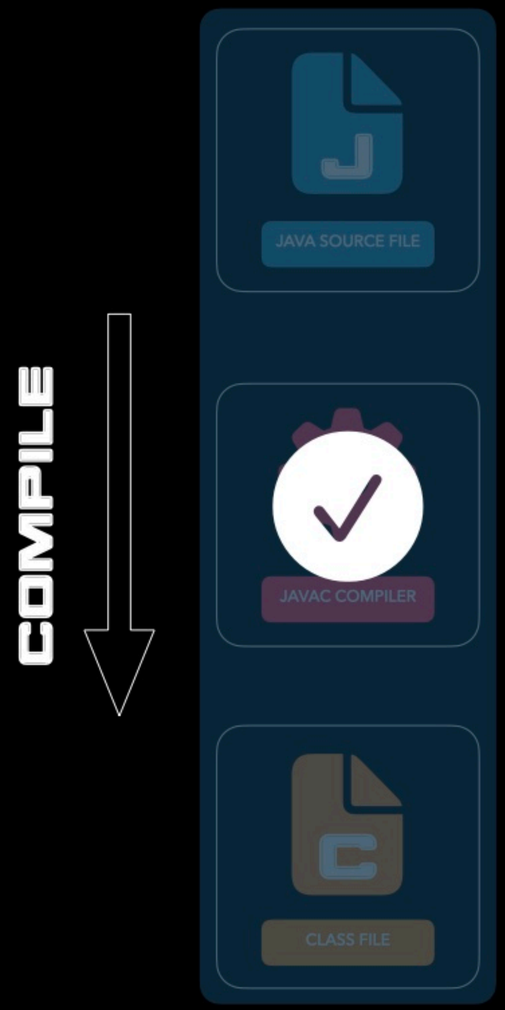
azul

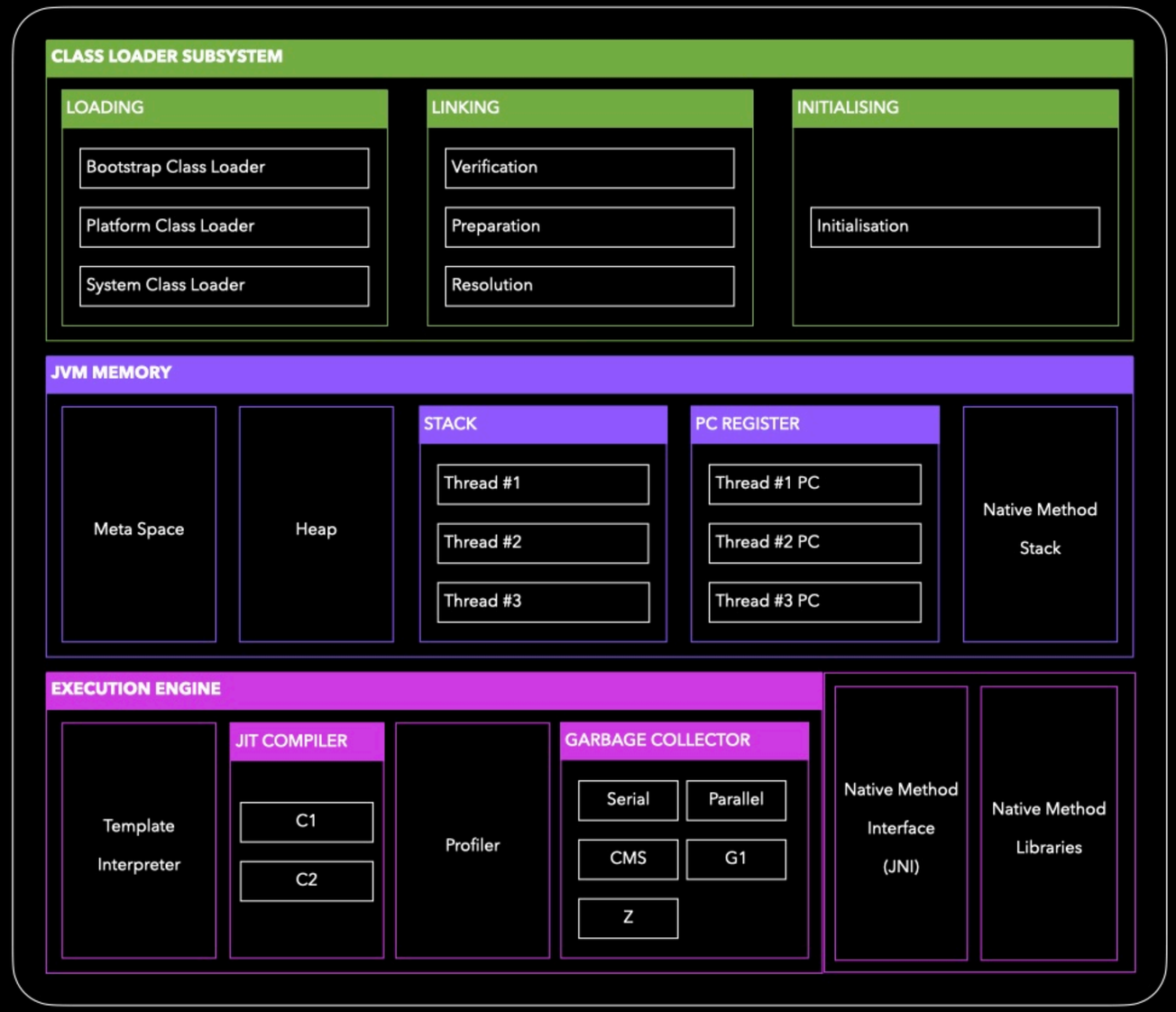
CLASS LOADING...

azul

JVM STARTUP

How does it work...





azul

JVM STARTUP

How does it work...

CLASS LOADER SUBSYSTEM

LOADING

- Bootstrap Class Loader
- Platform Class Loader
- System Class Loader

LINKING

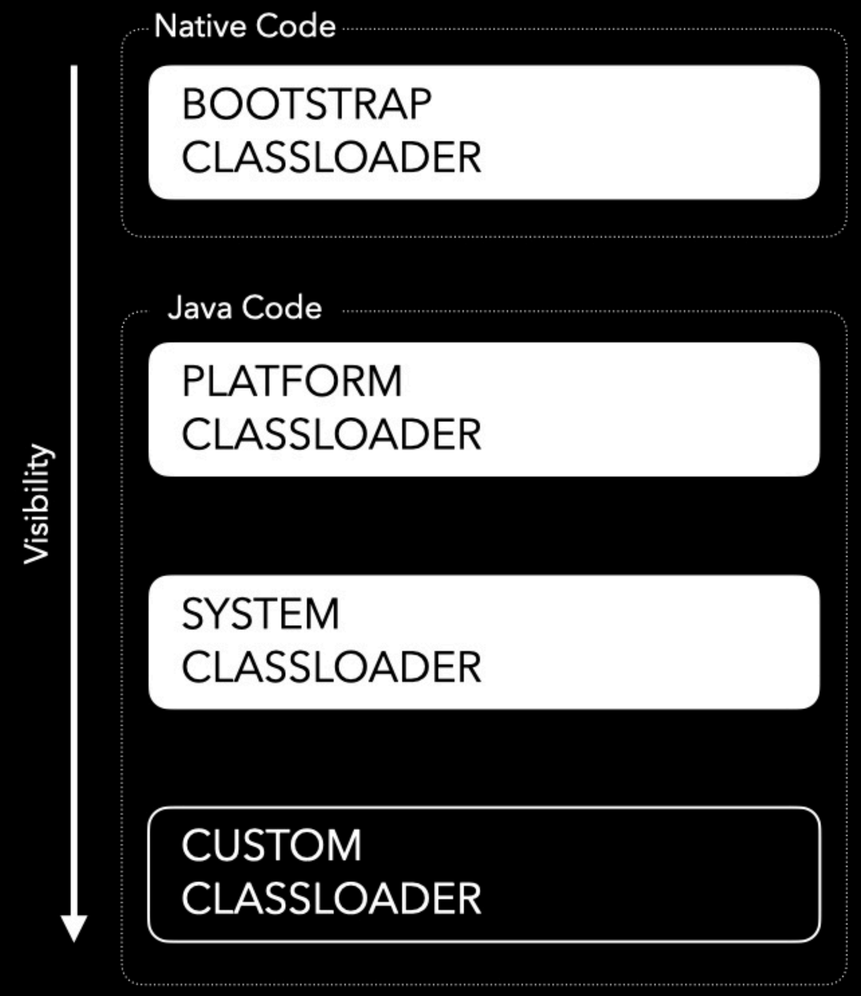
- Verification
- Preparation
- Resolution

INITIALISING

- Initialisation

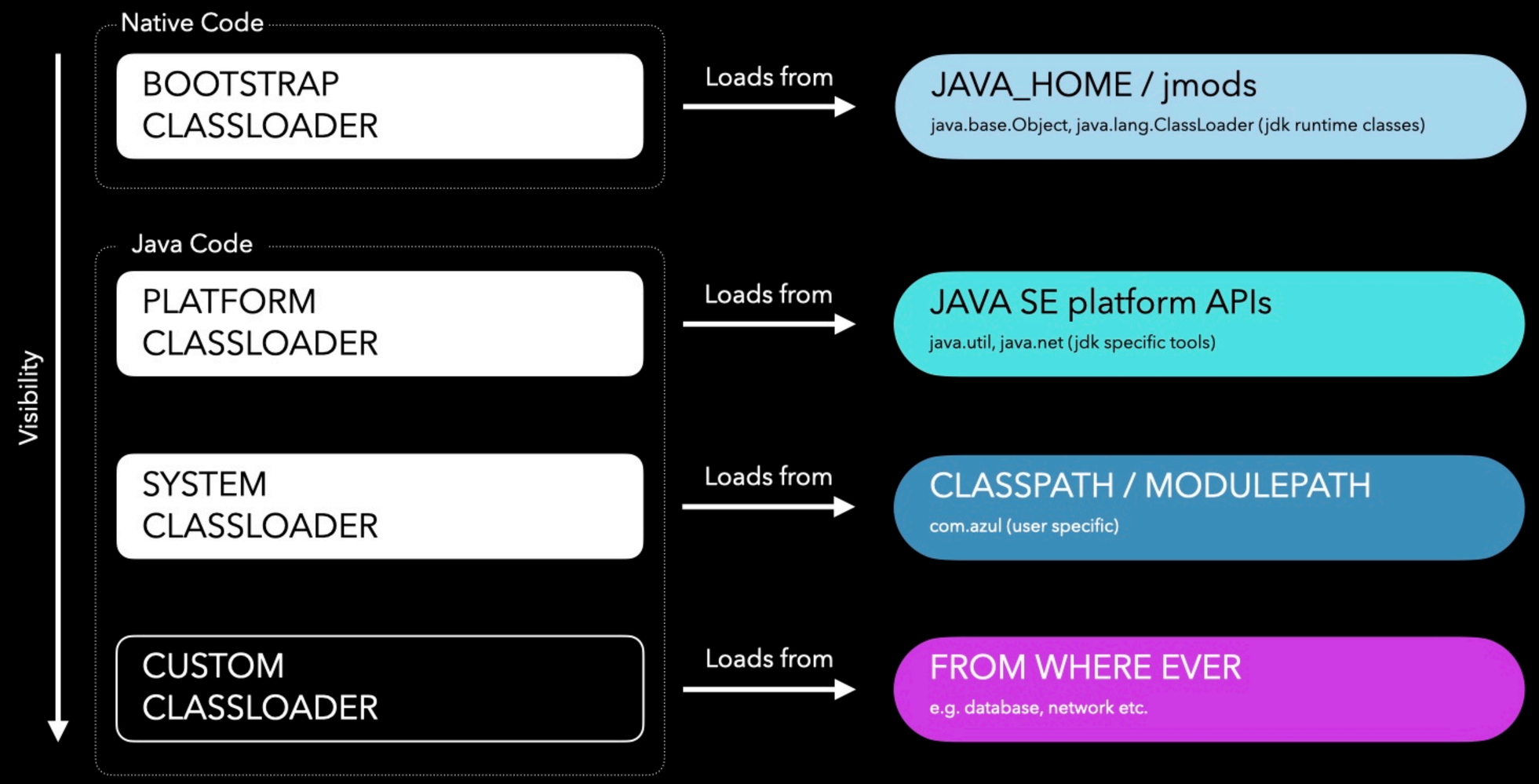
JVM STARTUP

Delegation and Visibility



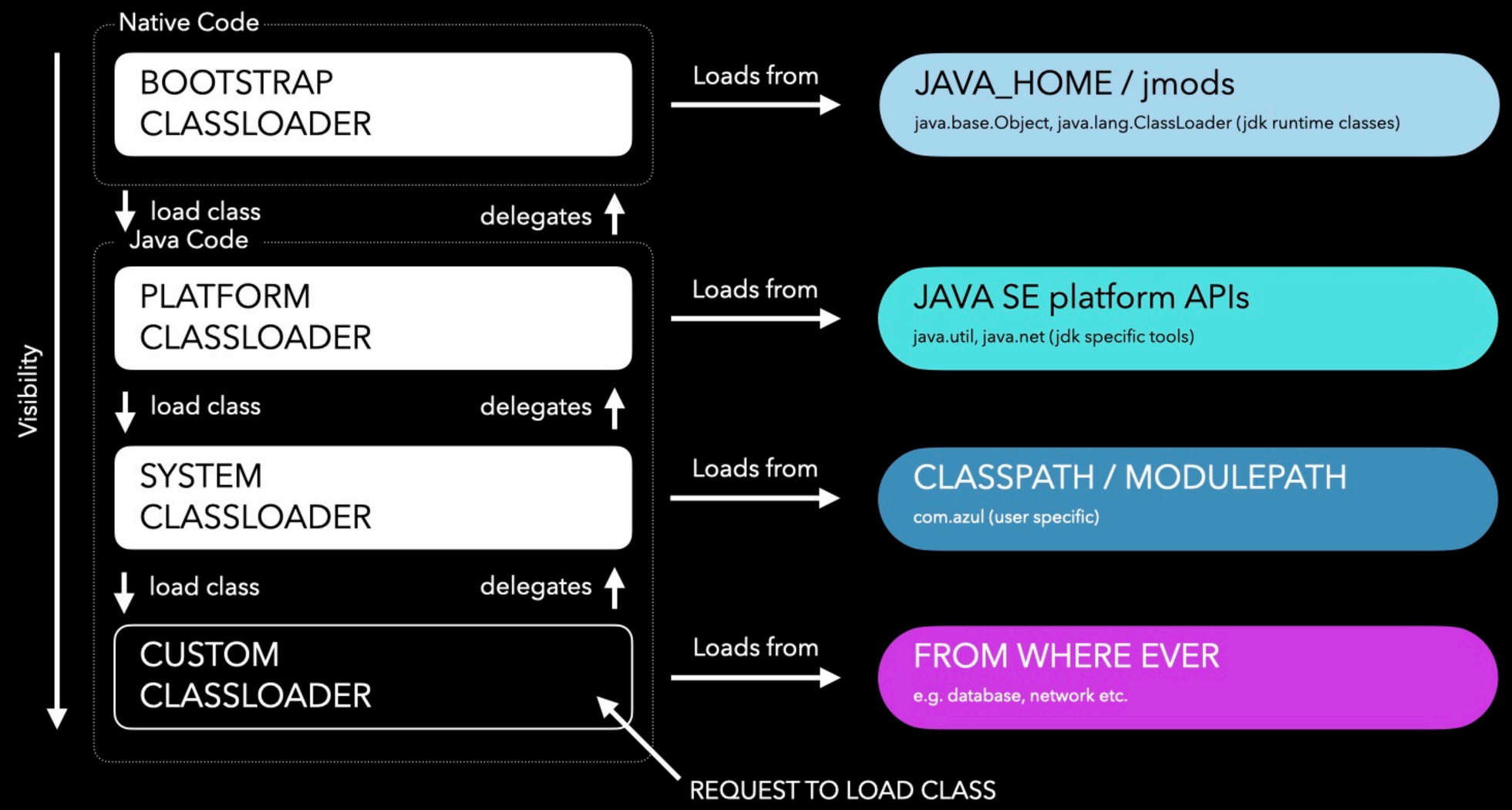
JVM STARTUP

Delegation and Visibility



JVM STARTUP

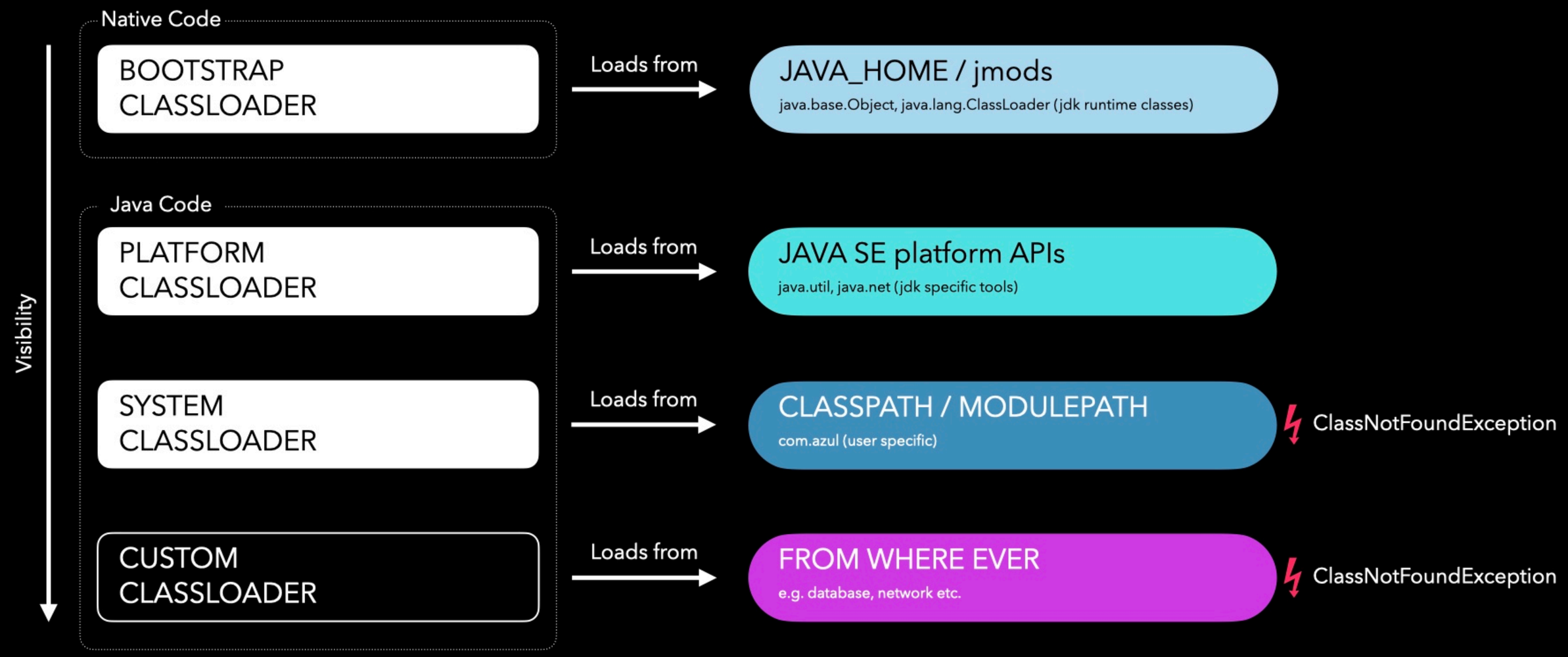
Load request comes in



azul

JVM STARTUP

Delegation and Visibility



azul

JVM STARTUP

On demand class loading

LOADING

- Find the binary file
- Decompress it from archive
- Load the file

→

LINKING

- Verification
  (might kick off loading of other classes)
- Preparation
  (initialisation of static fields with defaults)
- Resolution
  (resolve symbolic references in Constant Pool)

→

INITIALISING

- Execute the class
  <init> / <clinit> method
- Assign constant values
  to static fields

JVM STARTUP

azul

EXECUTION...

azul

JVM STARTUP

azul

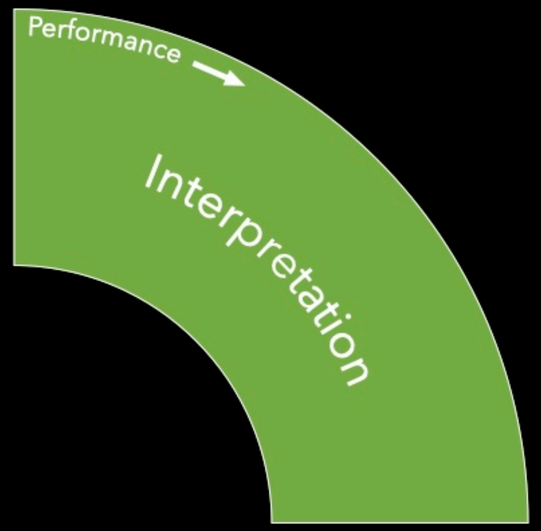
JVM STARTUP

TIERED COMPILATION

JVM STARTUP

Execution Cycle

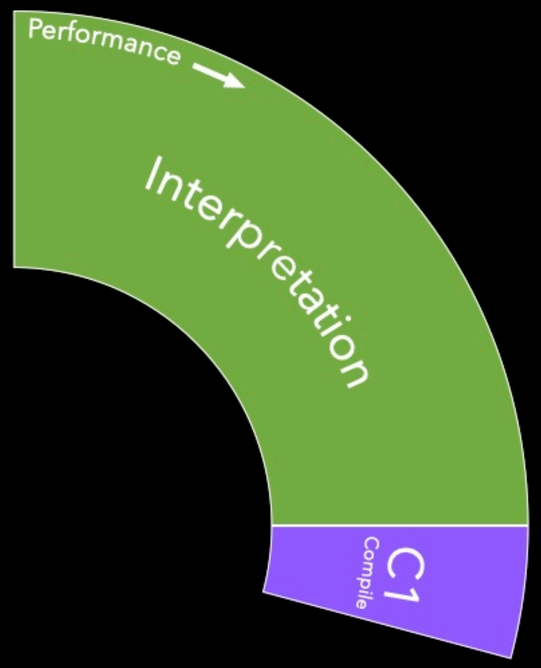
- Interpretation
- Profiling the interpreted code



JVM STARTUP

Execution Cycle

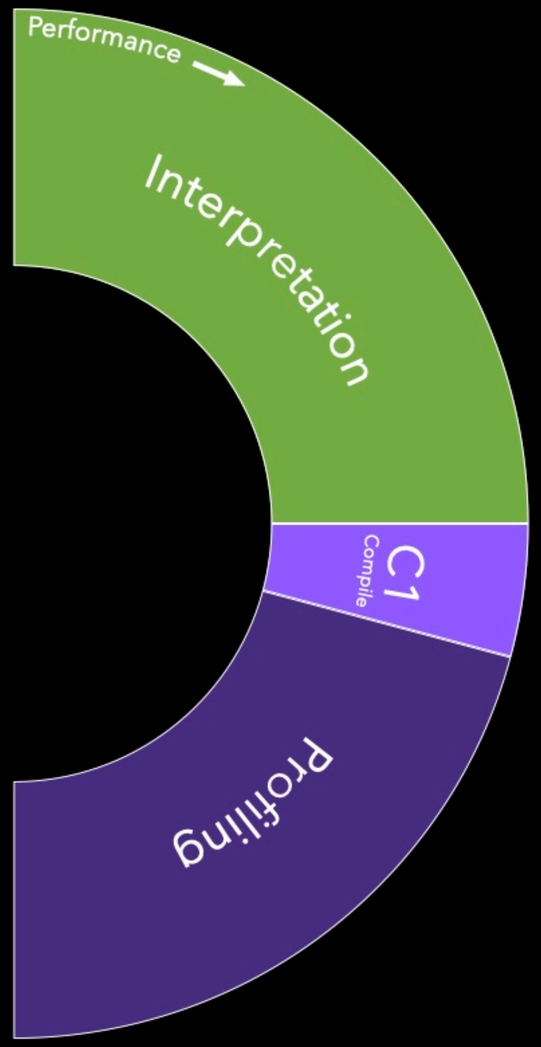
- Interpretation
- Profiling the interpreted code
- Reaching a threshold of e.g. 1000 method calls
- Compiling with C1 (fast, no optimisation)



JVM STARTUP

Execution Cycle

- Interpretation
- Profiling the interpreted code
- Reaching a threshold of e.g. 1000 method calls
- Compiling with C1 (fast, no optimisation)
- Profiling the C1 compiled code



JVM STARTUP

Execution Cycle

- Interpretation
- Profiling the interpreted code
- Reaching a threshold of e.g. 1000 method calls
- Compiling with C1 (fast, no optimisation)
- Profiling the C1 compiled code
- Reaching a threshold of e.g. 5000 method calls
- Compiling with C2 (slower, heavy optimisation)

JVM STARTUP

Execution Cycle

- Interpretation
- Profiling the interpreted code
- Reaching a threshold of e.g. 1000 method calls
- Compiling with C1 (fast, no optimisation)
- Profiling the C1 compiled code
- Reaching a threshold of e.g. 5000 method calls
- Compiling with C2 (slower, heavy optimisation)
- Optimising the C2 compiled code

JVM STARTUP

Execution Cycle

- Interpretation
- Profiling the interpreted code
- Reaching a threshold of e.g. 1000 method calls
- Compiling with C1 (fast, no optimisation)
- Profiling the C1 compiled code
- Reaching a threshold of e.g. 5000 method calls
- Compiling with C2 (slower, heavy optimisation)
- Optimising the C2 compiled code
- De-Optimising and go back to Interpretation

INTERPRETATION...

azul

JVM STARTUP

Interpretation

- Template Interpreter (platform dependent, used as default)

JVM STARTUP

Interpretation

- Template Interpreter (platform dependent, used as default)
- Will be created at JVM startup in around in ms (e.g. MBP 4 Pro ->14ms)

JVM STARTUP

Interpretation

- Template Interpreter (platform dependent)
- Will be created at JVM startup in around in ms (e.g. MBP 4 Pro ->14ms)
- java -XX:+UnlockDiagnosticVMOptions -XX:+PrintInterpreter

JVM STARTUP

Interpretation

Byte Code

```
int add(int, int);
  0: iload_0
  1: iload_1
  2: iadd
  3: ireturn
```

SEGMENTED CODE CACHE

NON METHOD CODE	PROFILED CODE	NON PROFILED CODE
Template: NNN ...		
Template: NNN ...		
Template: NNN ...		
Template: NNN ...		
Template: NNN ...		
Template: NNN ...		

JVM STARTUP

Interpretation

Byte Code

```
int add(int, int);
 0: iload_0
 1: iload_1
 2: iadd
 3: ireturn
```

SEGMENTED CODE CACHE

NON METHOD CODE

```
Template: iload_0
mov     -0x8(%r14), %eax
movzb1   0x1(%r13), %ebx
inc      %r13
mov      $0xff40,%r10
jmpq     *(%r10,%rbx,8)
```

```
Template: NNN
...
```

```
Template: NNN
...
```

```
Template: NNN
...
```

PROFILED CODE

NON PROFILED CODE

JVM STARTUP

Interpretation

Byte Code

```
int add(int, int);
  0: iload_0
  1: iload_1
  2: iadd
  3: ireturn
```

SEGMENTED CODE CACHE

NON METHOD CODE

```
Template: NNN
...
```

```
Template: NNN
...
```

```
Template: iload_1
mov     -0x8(%r14), %eax
movzb1   0x1(%r13), %ebx
inc      %r13
mov      $0xff40,%r10
jmpq     *(%r10,%rbx,8)
```

```
Template: NNN
...
```

PROFILED CODE

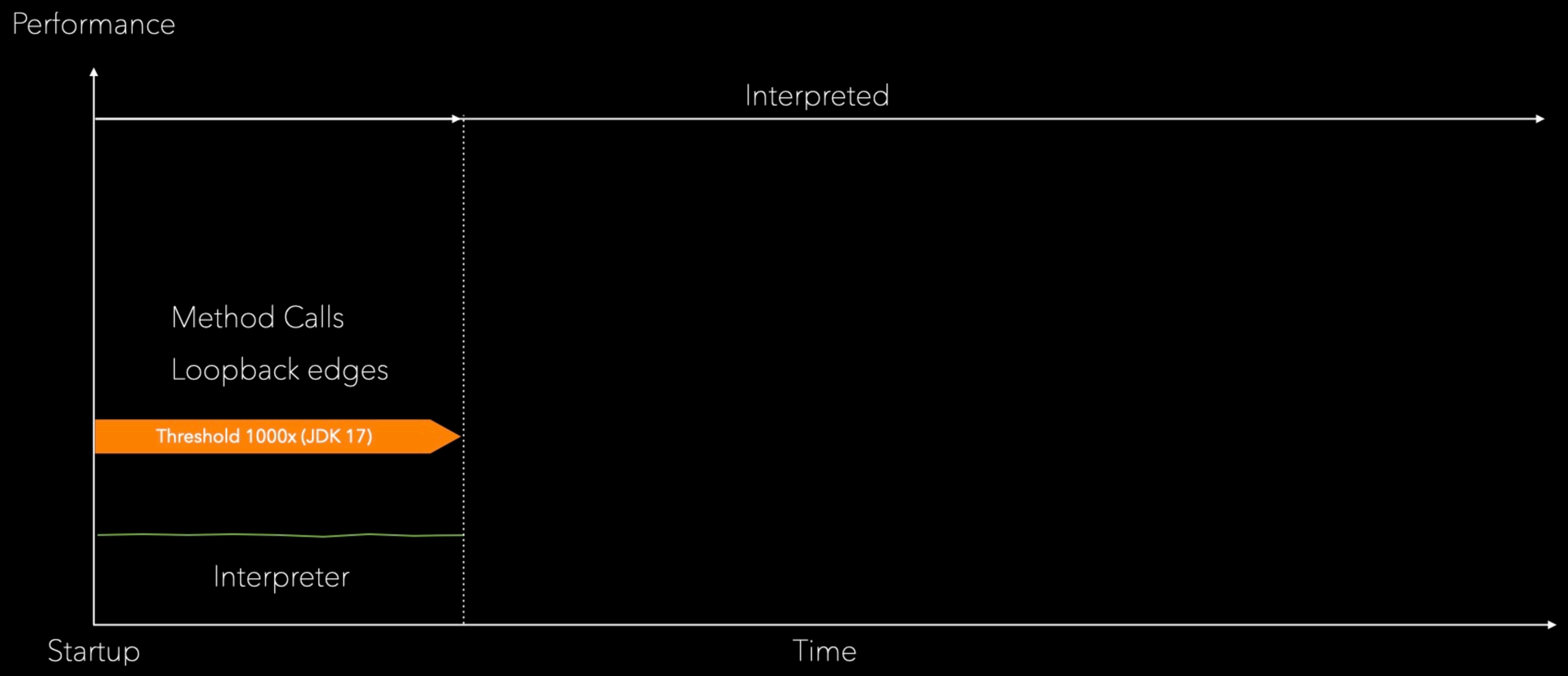
NON PROFILED CODE

TIERED COMPILATION...

azul

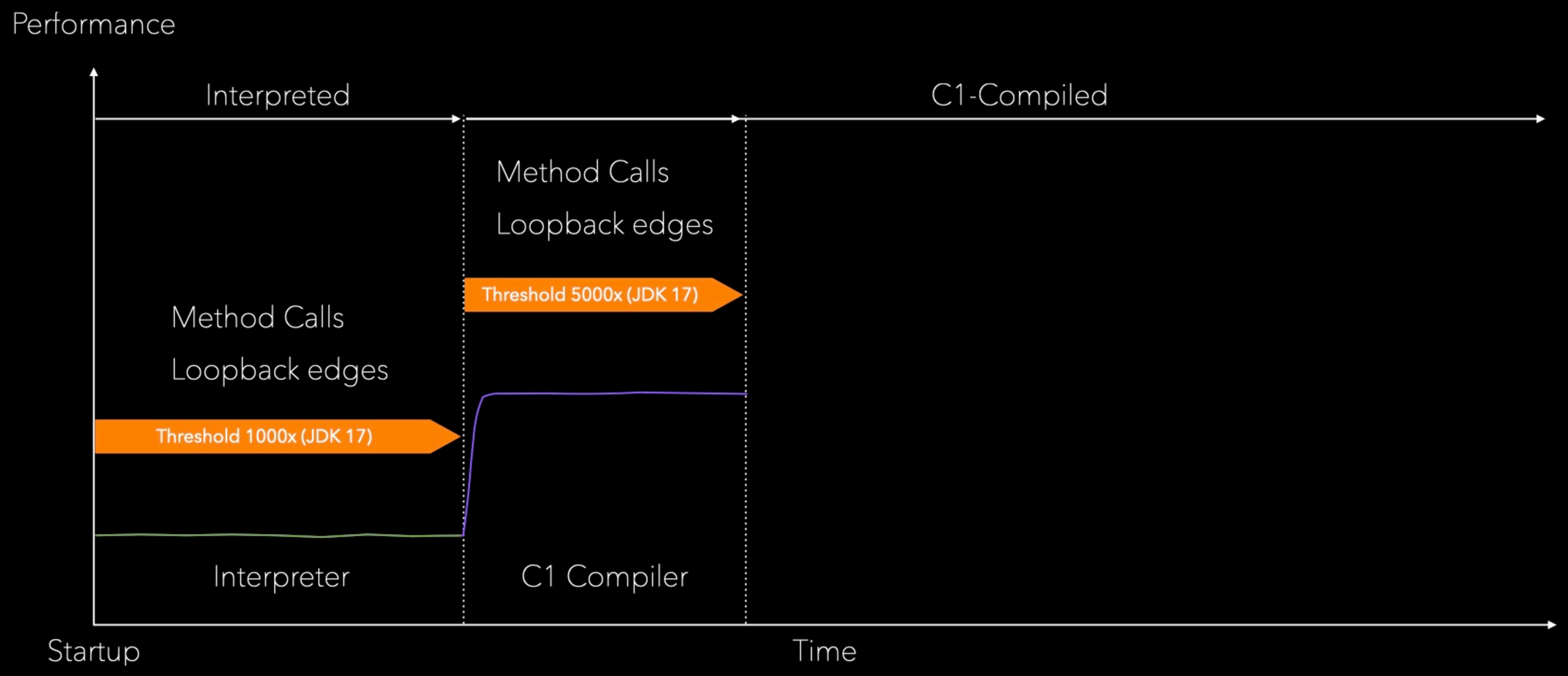
JVM STARTUP

Execution - Interpretation



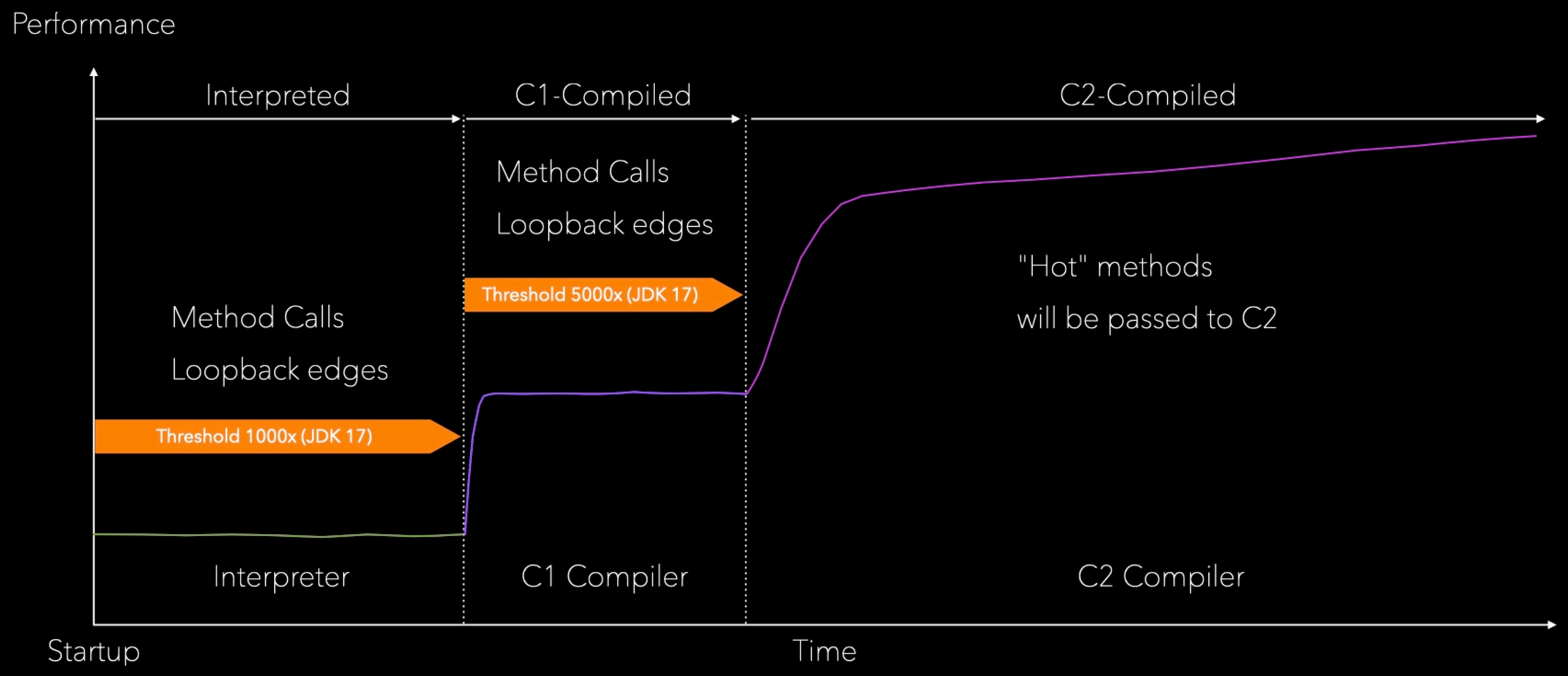
JVM STARTUP

Tiered Compilation



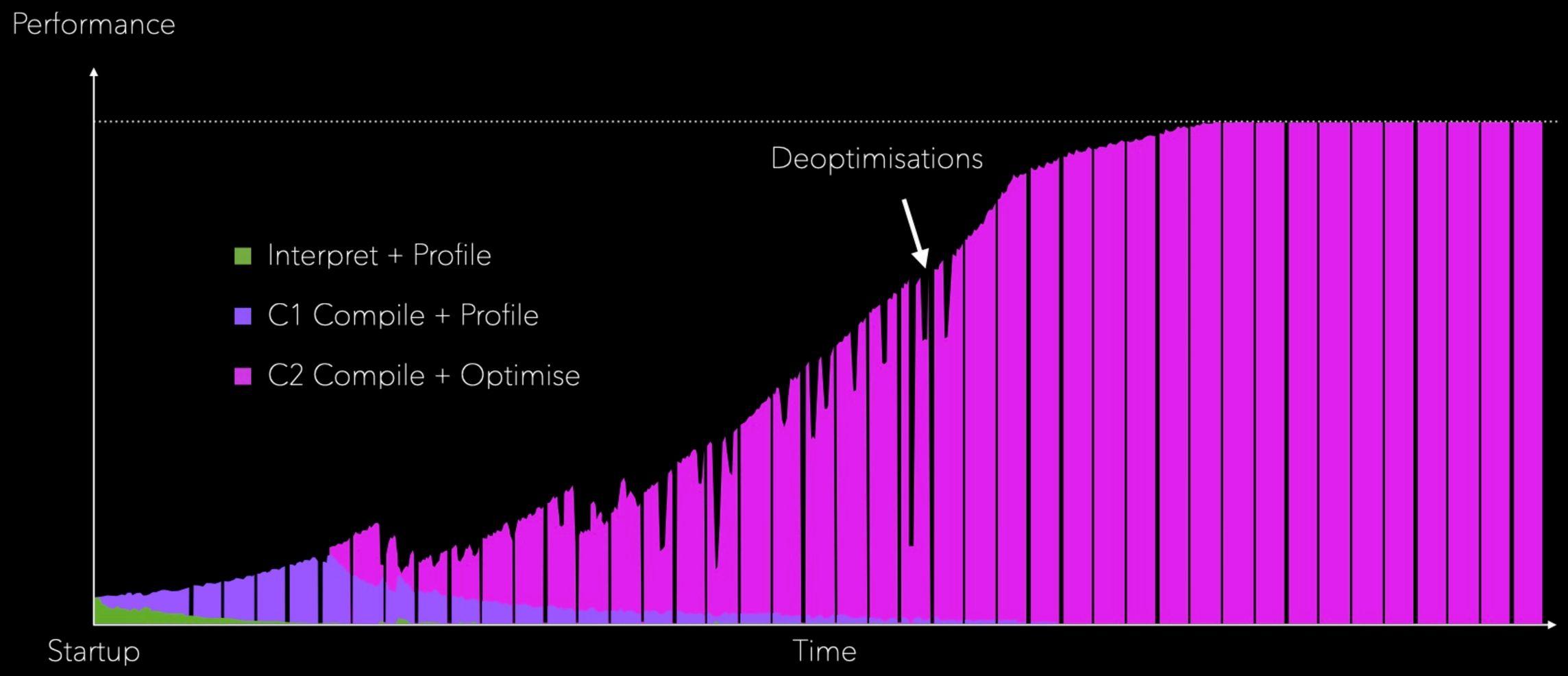
JVM STARTUP

Tiered Compilation



JVM STARTUP

JVM Performance Graph



azul

DEOPTIMISATION...

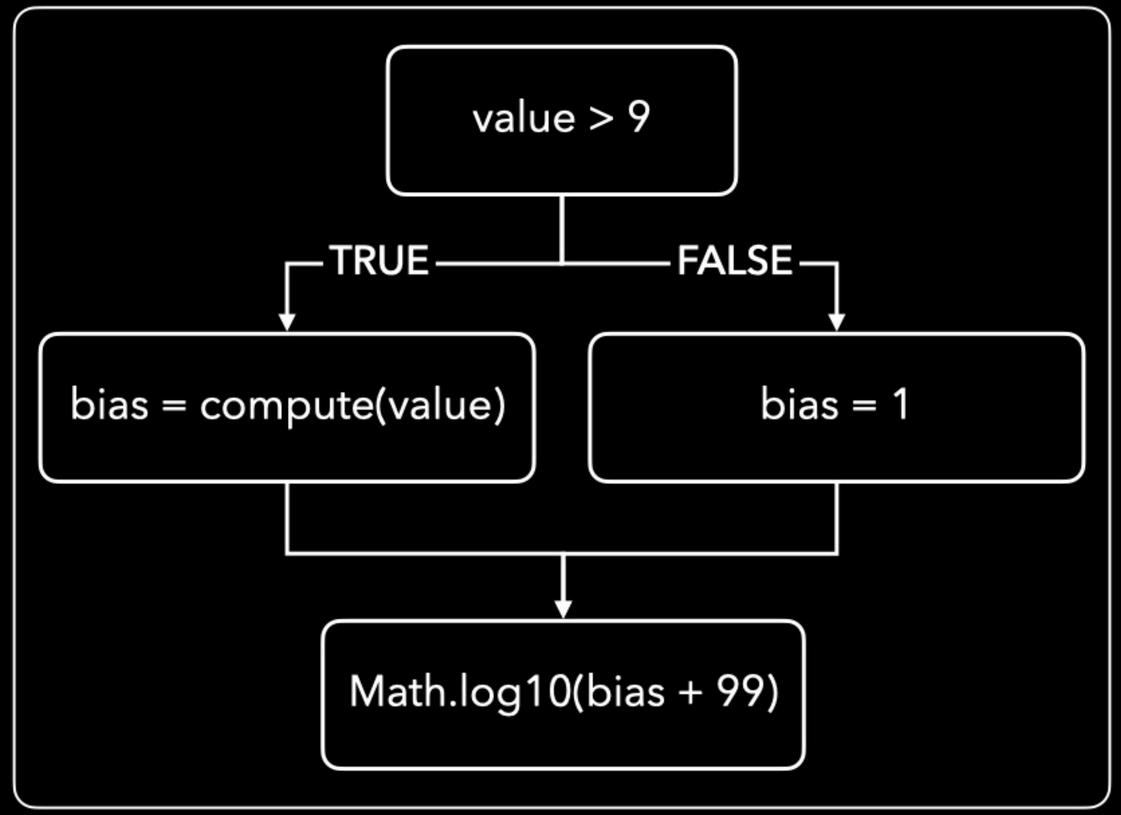
azul

JVM STARTUP

Speculative Optimisation

e.g. BRANCH ANALYSIS

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

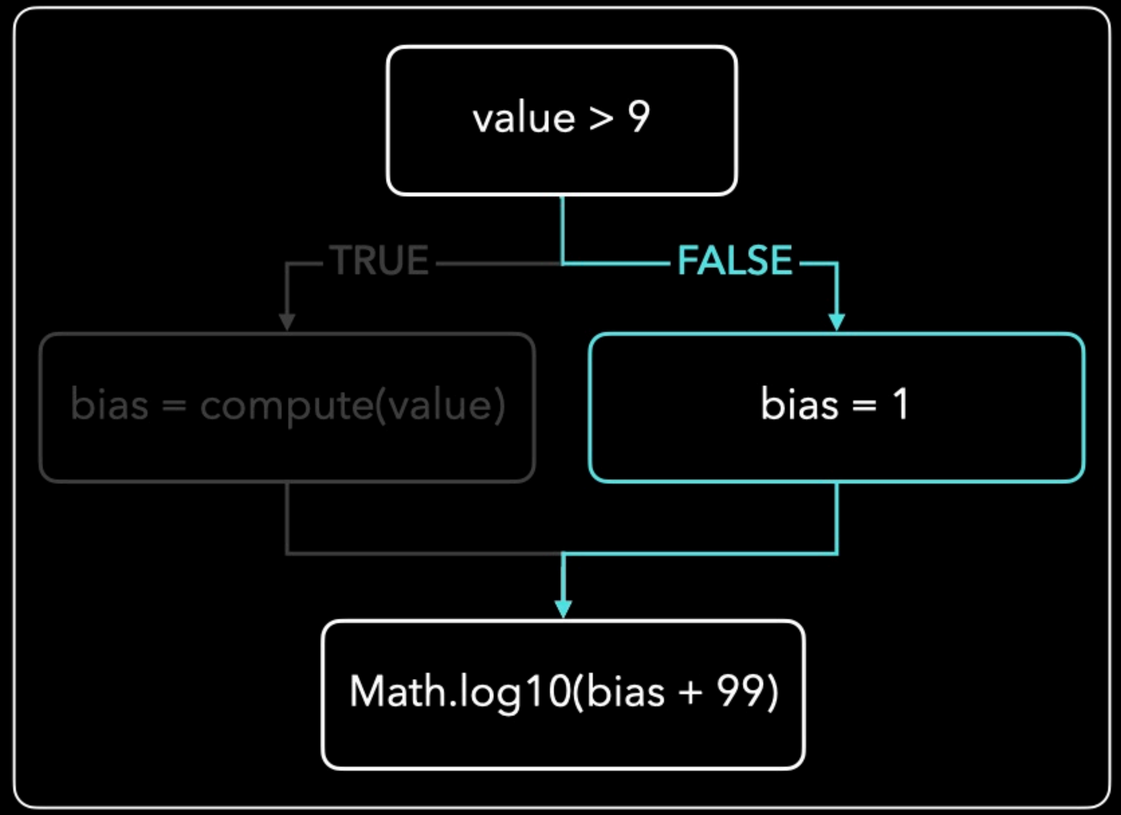


JVM STARTUP

Speculative Optimisation

e.g. BRANCH ANALYSIS

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



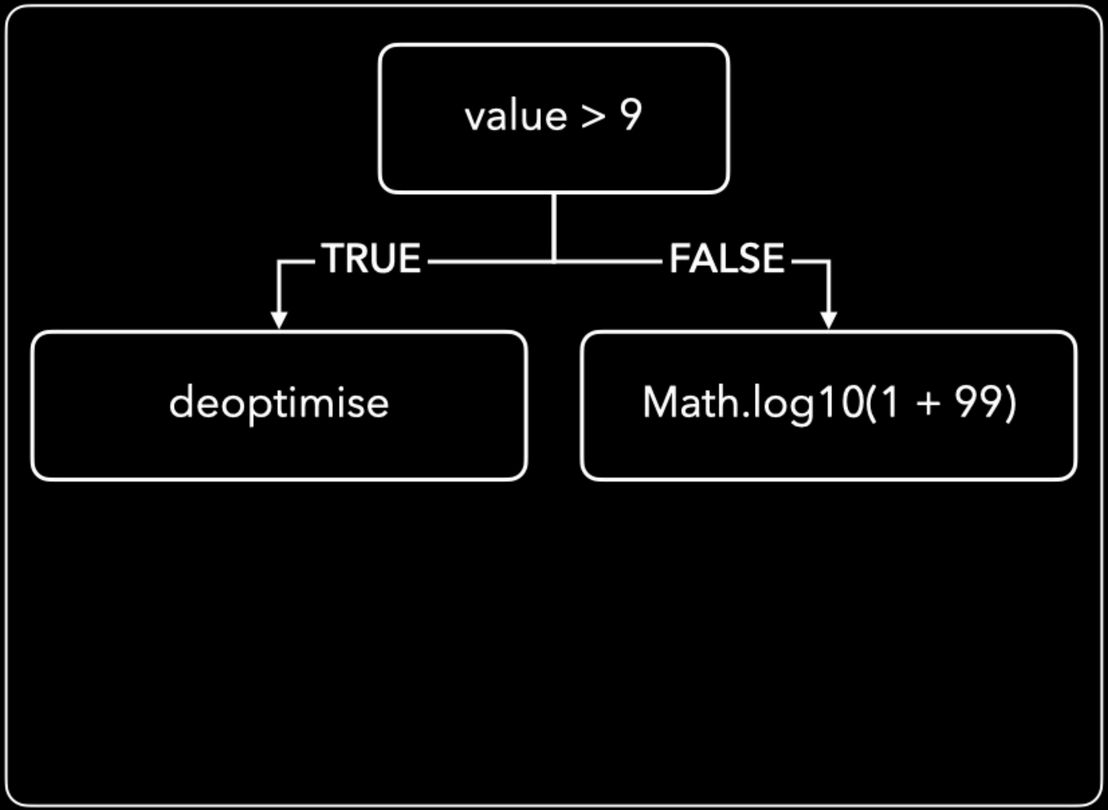
Value never greater than 9

JVM STARTUP

Speculative Optimisation

e.g. BRANCH ANALYSIS

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

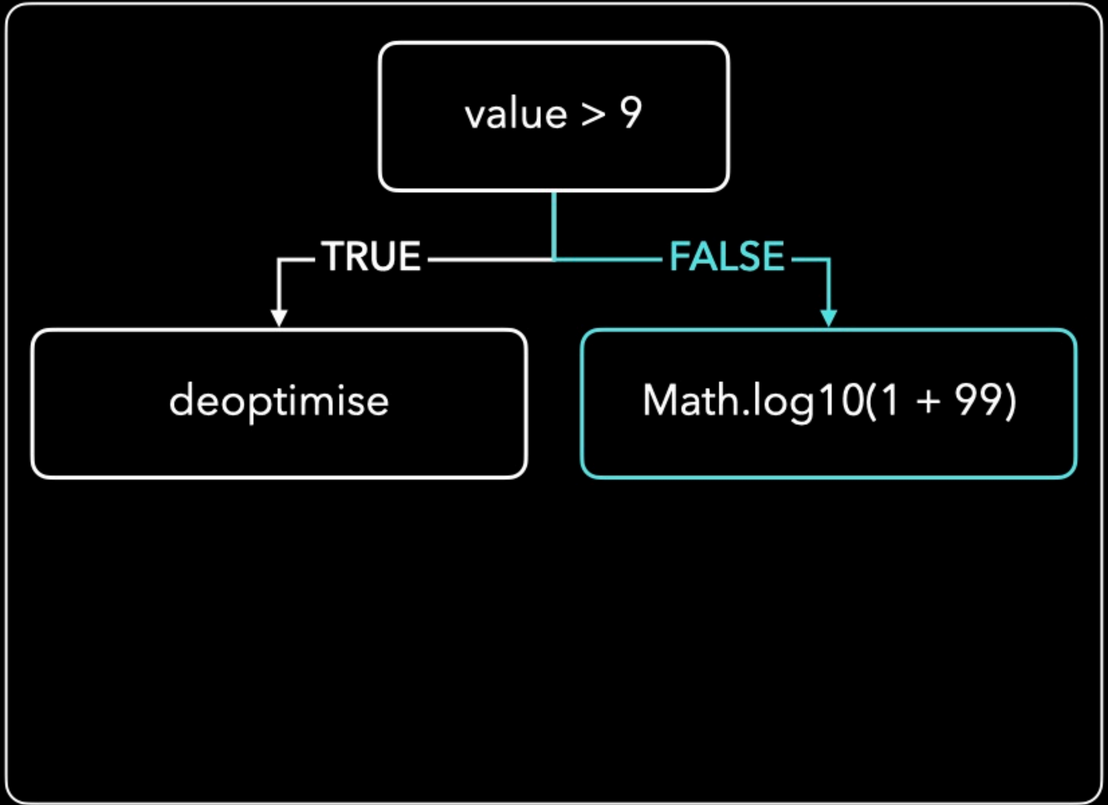


JVM STARTUP

Speculative Optimisation

e.g. BRANCH ANALYSIS

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

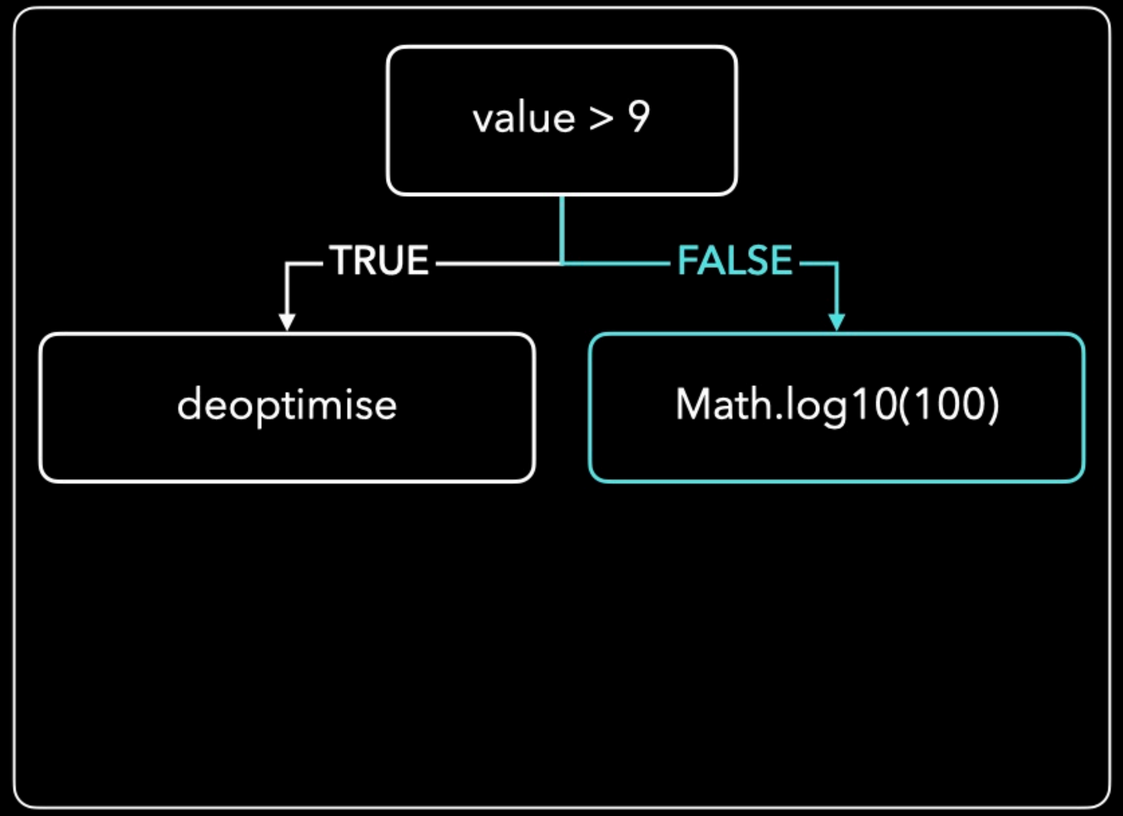


JVM STARTUP

Speculative Optimisation

e.g. BRANCH ANALYSIS

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

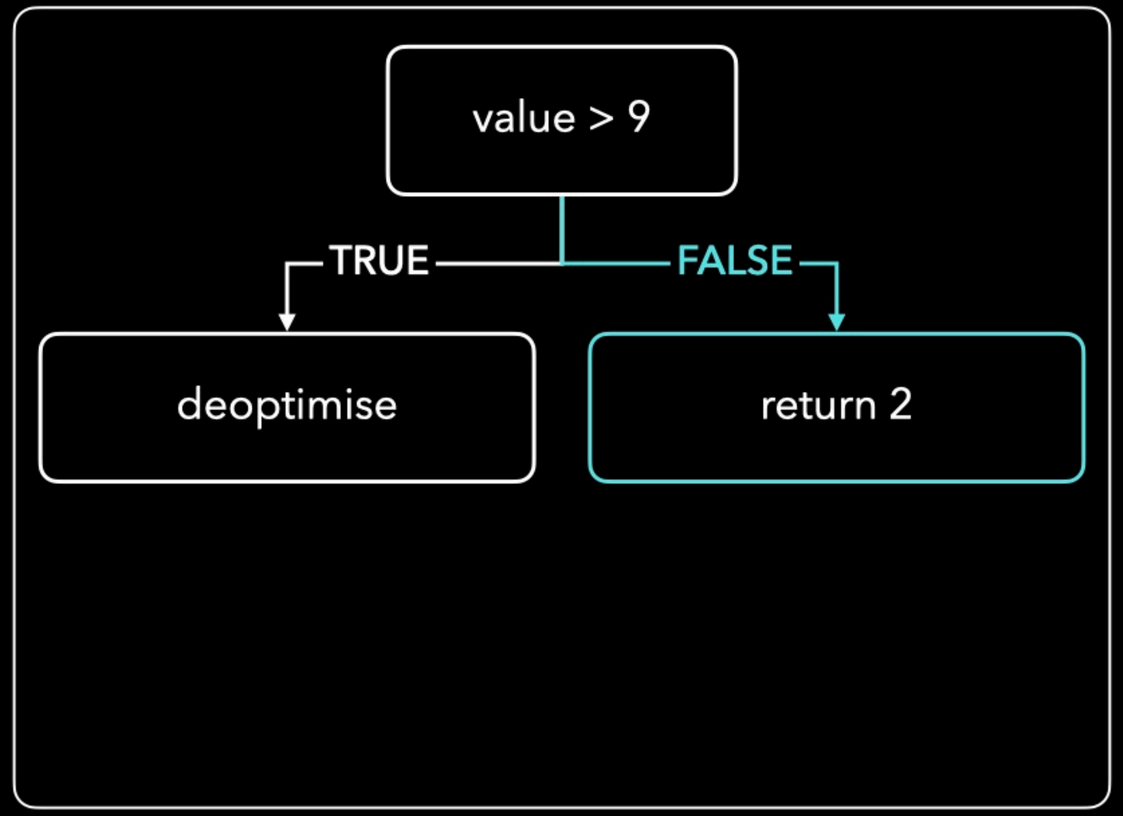


JVM STARTUP

Speculative Optimisation

e.g. BRANCH ANALYSIS

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

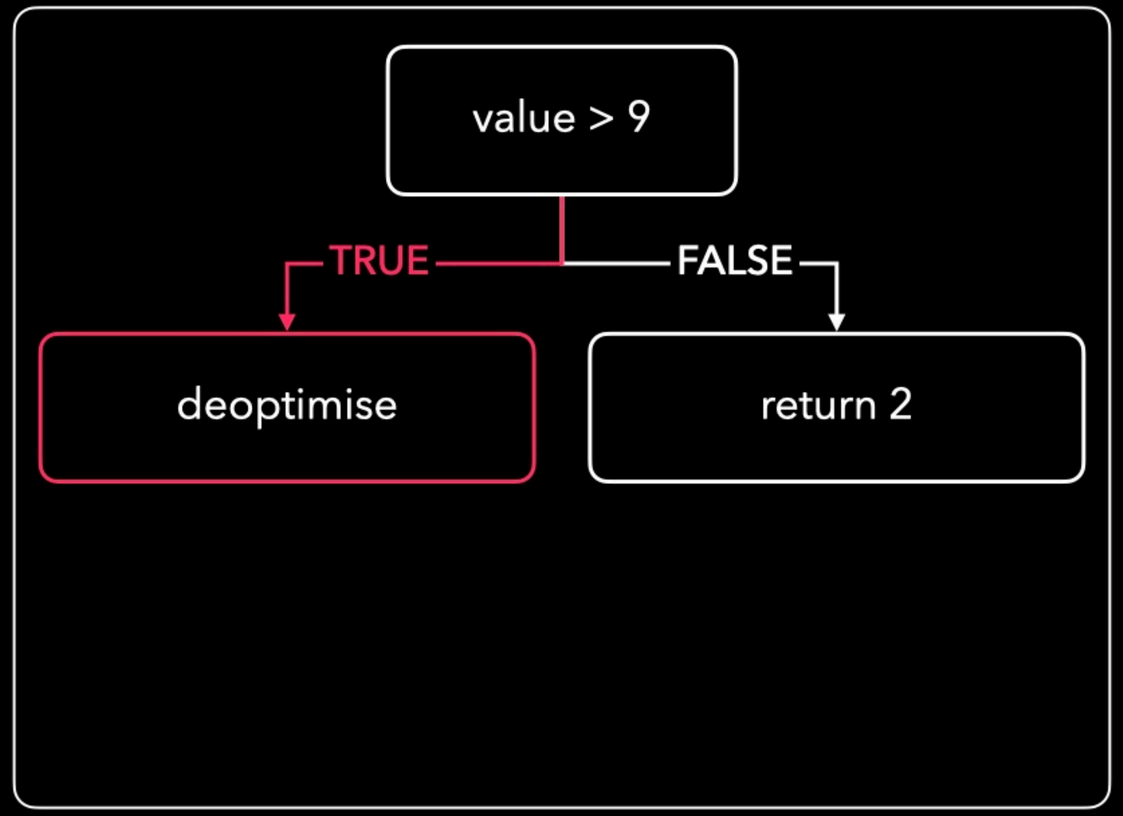


JVM STARTUP

Speculative Optimisation

e.g. BRANCH ANALYSIS

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

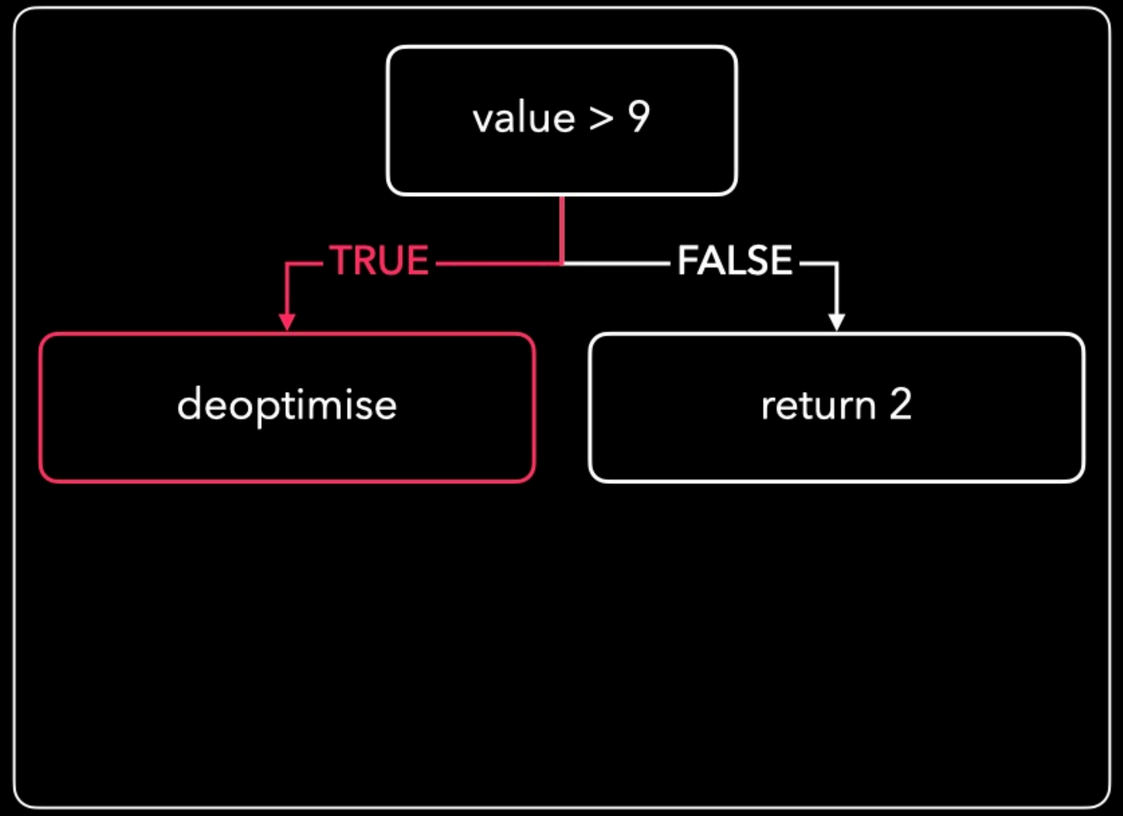


JVM STARTUP

Speculative Optimisation

e.g. BRANCH ANALYSIS

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

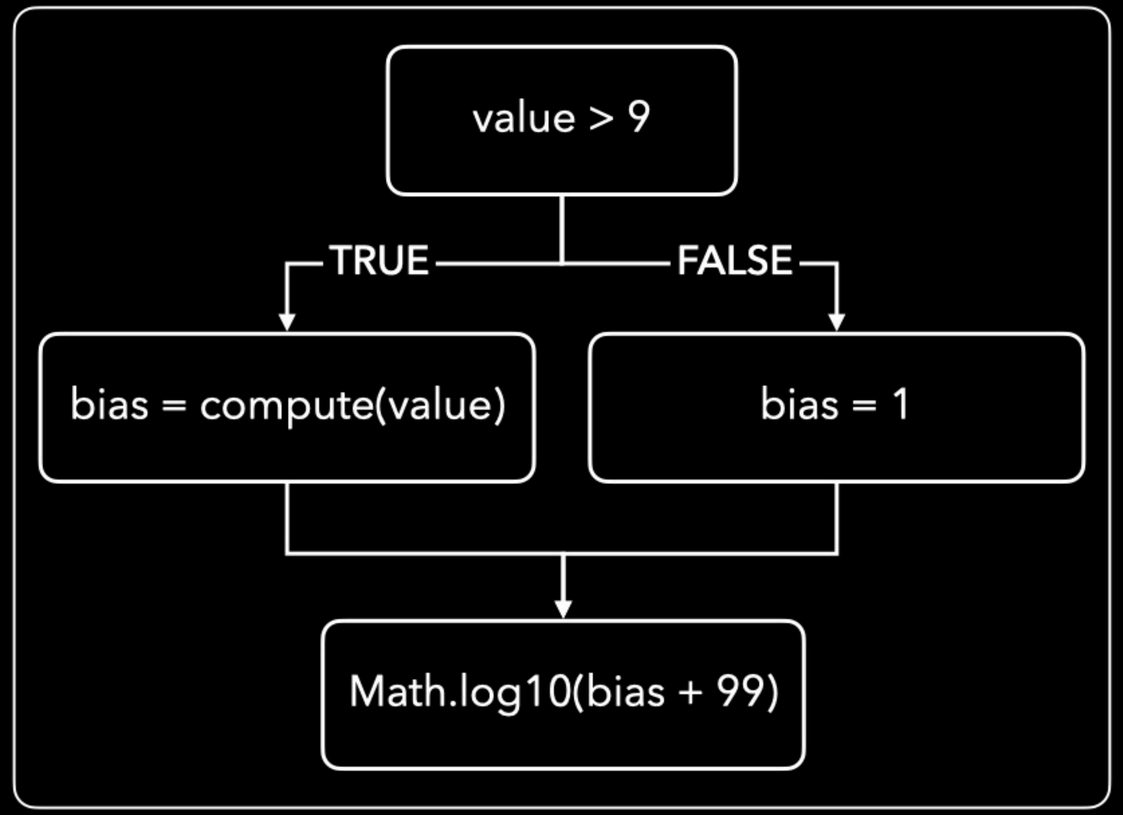


JVM STARTUP

Speculative Optimisation

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

ALL THESE OPTIMISATIONS LEAD TO...

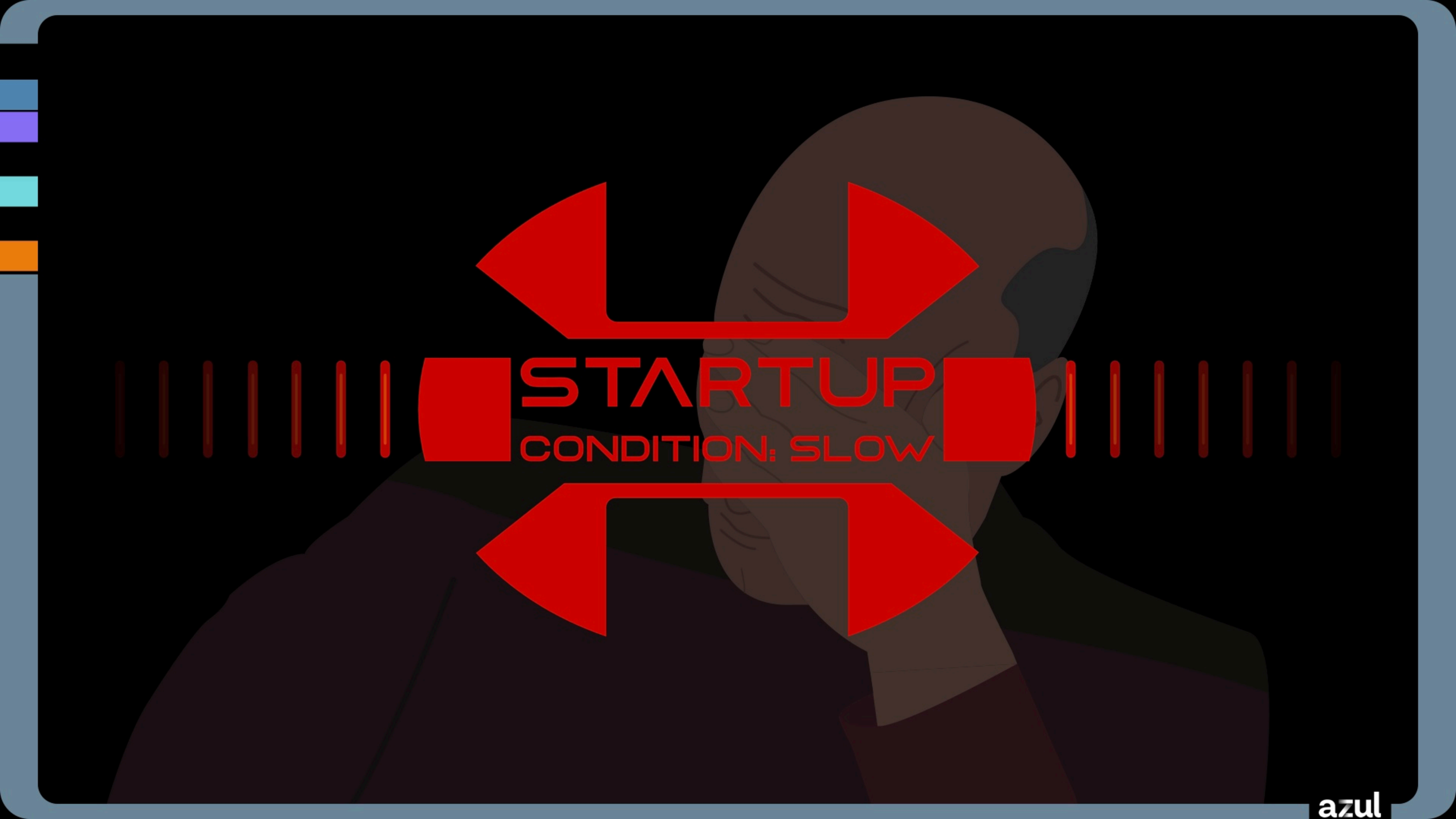
azul

AWESOME PERFORMANCE...

azul

BUT...

azul



azul

SLOW ZONES

azul

SLOW ZONES

Where is the time spent...?

SOLUTIONS...

azul

SOLUTIONS

01 CDS/AppCDS

02 Native Image

03 Leyden

04 CRaC

05 Ready Now

06 Cloud Native Compiler / JIT Server

azul

SOLUTIONS

01 CDS/AppCDS

02 Native Image

03 Leyden

04 CRaC

05 Ready Now

06 Cloud Native Compiler / JIT Server

CDS/APPCDS

Class Data Sharing and Application Class Data Sharing

- Dump internal representation (memory format) of shared classes into a single file
- Shared on each JVM start (CDS) or application start (AppCDS)
- Default CDS archive for JDK runtime classes shipped since JDK 17
- Can reduce time to load classes by up to 2 seconds
- Good explanation can be found at the blog of Ionut Balosin
- Will be replaced by Project Leyden (Cache Data Storage) from JDK24+

SPRINGBOOT SUPPORT...

azul

SPRINBOOT

Create an AppCDS Archive using features from SpringBoot

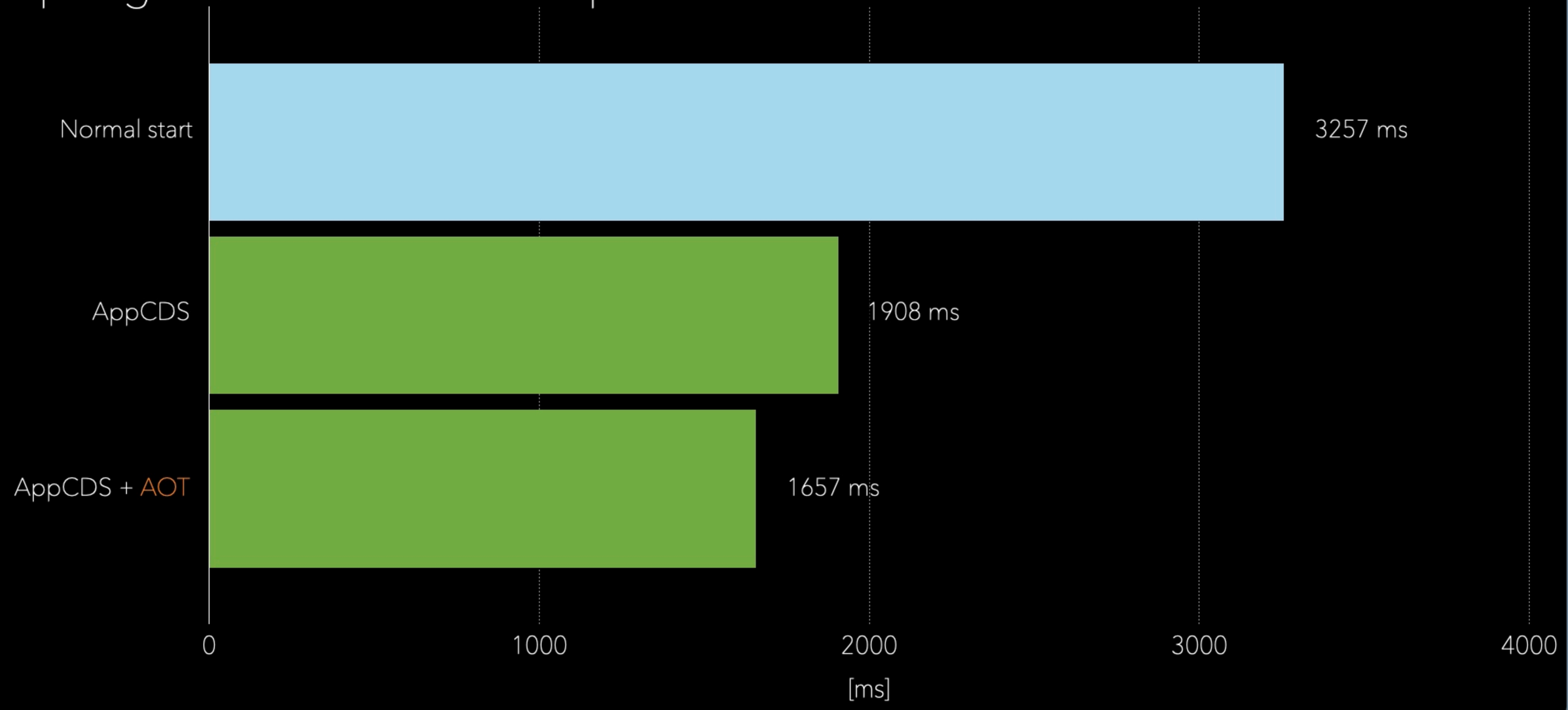
```
add plugin to build.gradle: plugin 'org.springframework.boot.aot'          // Add plugin to build file
build petclinic jar e.g. gradlew clean build                                // Build petclinic jar
java -Djarmode=tools -jar petclinic.jar extract --destination extracted     // Extract jar to destination
cd extracted                                                                // Open to destination folder

// Create AppCDS archive
java -DSpring.aot.enabled=true -XX:ArchiveClassesAtExit=app-cds.jsa -Dspring.context.exit=onRefresh -jar
petclinic.jar

// Start using the AppCDS archive
java -Dspring.aot.enabled=true -XX:SharedArchiveFile=app-cds.jsa -jar petclinic.jar
```

CDS/APPCDS

SpringBoot PetClinic Startup



azul

SOLUTIONS

01 CDS/AppCDS

02 Native Image

03 Leyden

04 CRaC

05 Ready Now

06 Cloud Native Compiler / JIT Server

NATIVE IMAGE

Ahead of Time compilation

- No interpretation, analysis of hot spots and no runtime compilation
- Compile directly to native executable ahead of time
- No runtime optimisation possible (Profile Guided Optimisation* can help)
- Needs to know all classes that will be used at compile time (Problem reflection etc.)
- Native executable will only contain used classes
- Very long compile process
- Debugging is hard (No debugging of Java code because it is native code)

* GraalVM Enterprise only

NATIVE IMAGE

Ahead of Time compilation

GraalVM Community

- Free to use in production
- Doesn't have support for PGO
- Lower performance

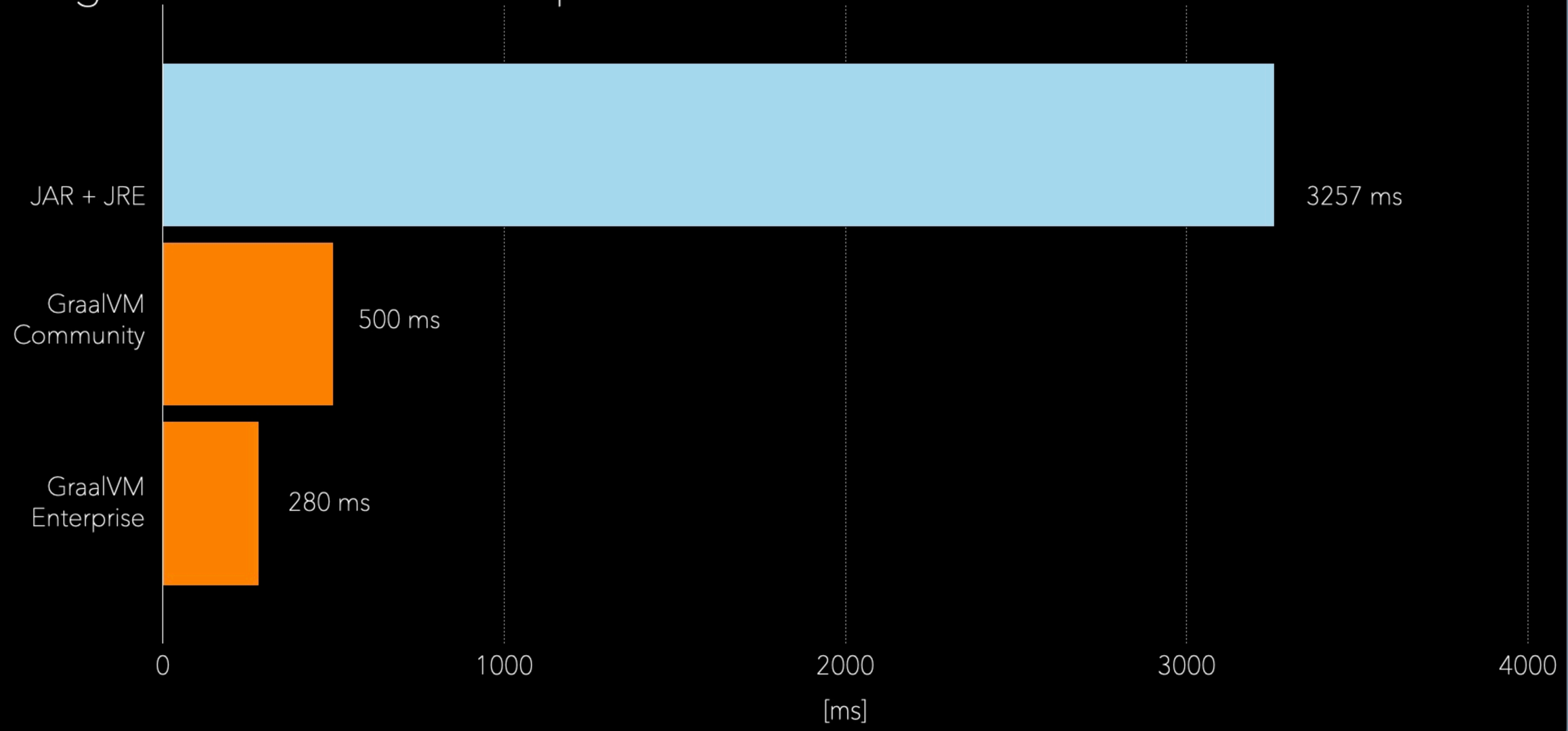
GraalVM Enterprise

- Only free to use in production for 3 years
- Has support for PGO
- Higher performance
- Support is questionable at the moment
  (no official support from Oracle since JDK 25)
- Future is a bit uncertain at the moment

⚠️ ATTENTION ⚠️

NATIVE IMAGE

SpringBoot PetClinic Startup



azul

SOLUTIONS

01 CDS/AppCDS

02 Native Image

03 Leyden

04 CRaC

05 Ready Now

06 Cloud Native Compiler / JIT Server

LEYDEN

Ahead of Time Class Loading and Linking (and later compilation)

- Available from JDK 25+
- Ahead-of-Time Class Loading and Linking (JEP 483)
- Command-Line Ergonomics (JEP 514) and Profiling (JEP 515)
- Cache Data Storage enhances AppCDS by caching more than just classes (JEP 483)
- Do a training run which will create an aot cache file
- Run the app
- Benefits from more CPUs

SPRINGBOOT SUPPORT...

azul

SPRINGBOOT

Support for Ahead-of-Time Processing

- Create exploded JAR with java -Djarmode=tools -jar App.jar extract
- Compile with
  - Maven -> mvn -Pnative package
  - Gradle -> plugin 'org.springframework.boot.aot'
- Run with -Dspring.aot.enabled=true
- Classpath is fixed and fully defined at build time ⚠️
- Beans defined in application cannot change at runtime ⚠️

SPRINGBOOT

Create an AOT Cache using features from SpringBoot

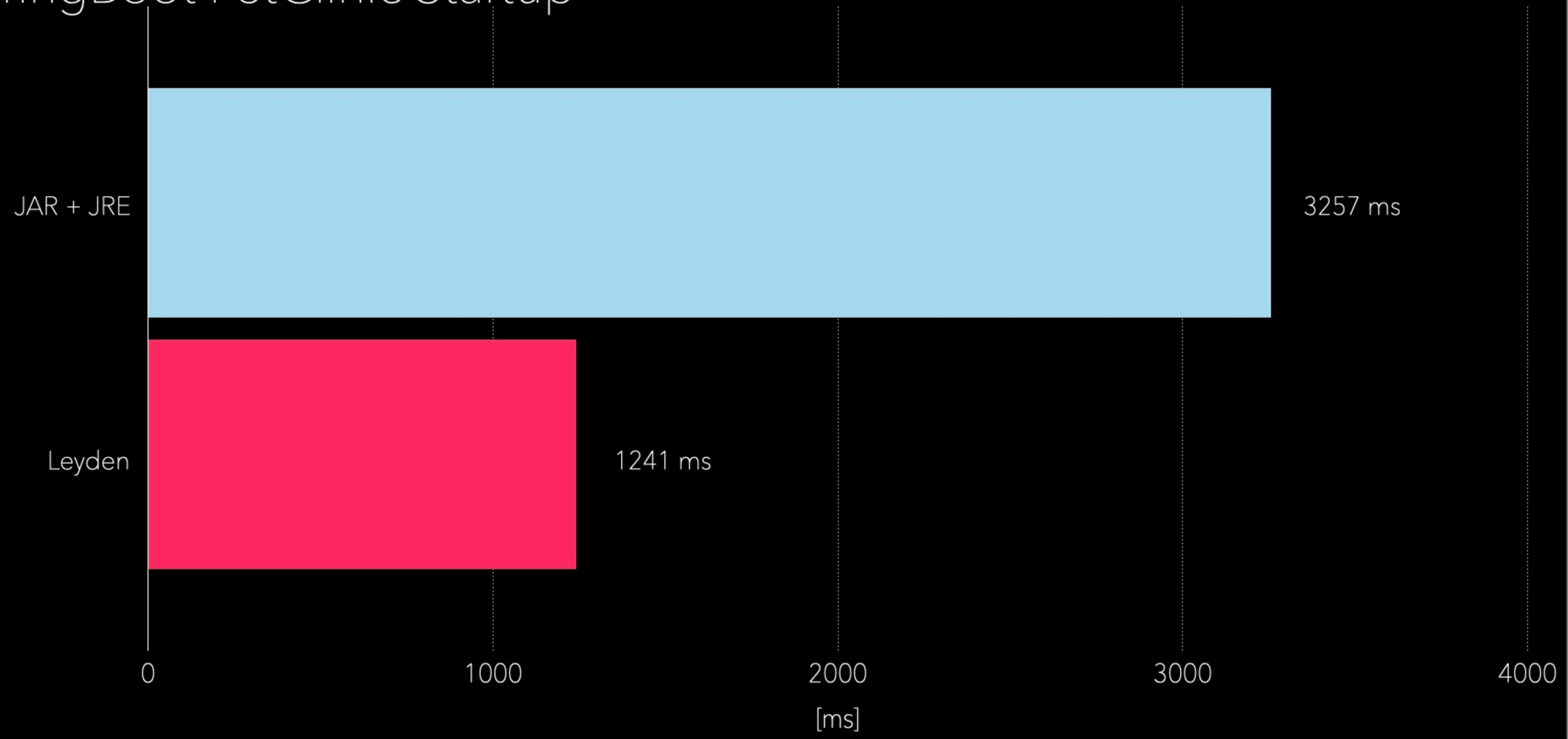
```
add plugin to build.gradle: plugin 'org.springframework.boot.aot'          // Add plugin to build file
build petclinic jar e.g. gradlew clean build                                // Build petclinic jar
java -Djarmode=tools -jar petclinic.jar extract --destination extracted     // Extract jar to destination
cd extracted                                                                // Open to destination folder
java -Dspring.aot.enabled=true -XX:AOTCacheOutput=petclinic.aot -jar petclinic.jar   // Create aot cache
java -Dspring.aot.enabled=true -XX:AOTCache=petclinic.aot -jar petclinic.jar         // Start using the aot cache
```

⚠️ Make sure to use the same GC for training and production run ⚠️

e.g. -XX:+UseG1GC -XX:G1HeapRegionSize=1048576

LEYDEN

SpringBoot PetClinic Startup



azul

SOLUTIONS

01 CDS/AppCDS

02 Native Image

03 Leyden

04 CRaC

05 Ready Now

06 Cloud Native Compiler / JIT Server

CRAC

Coordinated Restore at Checkpoint

- Needs a specific build of OpenJDK
- Run application
- Create a CheckPoint using jcmd
- Store the JVM with it's current state to disk
- Restore the JVM at a later point in time from the stored CheckPoint
- Based on CRIU and/or Warp (only Azul Zulu and Zing)
- Size of CheckPoint depends on heap size

CRAC

Distributions with Support for CRaC

- Azul Zulu (CRIU + Warp) JDK 17, 21, 25
- Azul Zing (CRIU + Warp) JDK 17, 21
- BellSoft Liberica (CRIU) JDK 17, 21
- IBM Semeru (InstantOn CRIU) JDK11, 17, 21, 25
- Alibaba DragonWell (CRIU) JDK 11
- Canonical JVM (CRIU , JDK 17, from Ubuntu 26+)

SPRINGBOOT SUPPORT...

azul

SPRINGBOOT

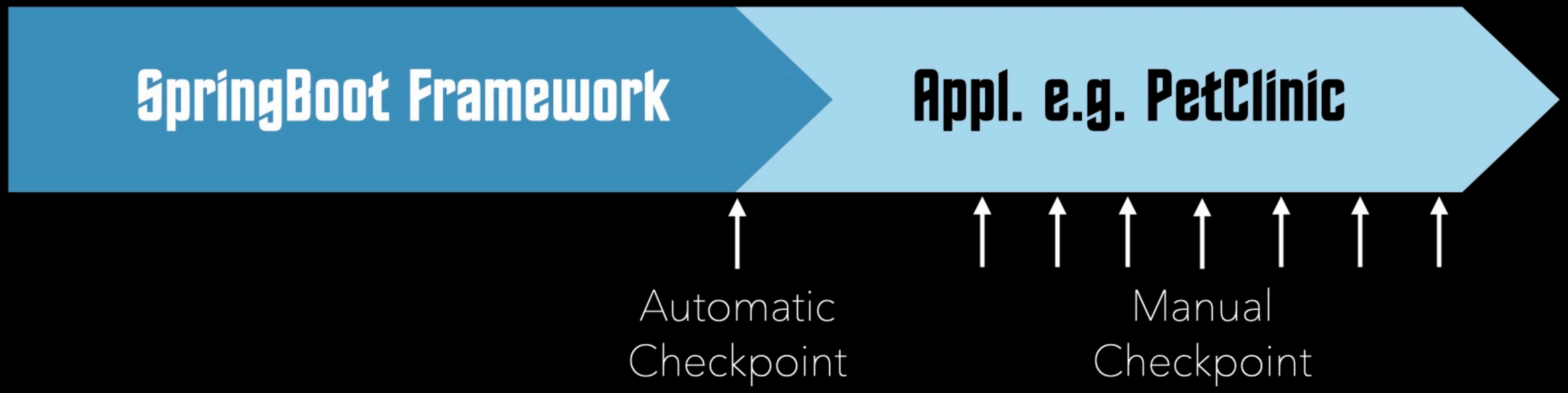
Support for CRaC

- Add dependency to org.crac 1.5.0
- Run with -Dspring.context.checkpoint=onRefresh
- Creates a checkpoint after the start of the SpringBoot framework
- Right before the application will be started
- No code changes needed !!!
- Referred to as "Auto"

SPRINGBOOT

Support for CRaC

-Dspring.context.checkpoint=onRefresh



CRAC

SpringBoot PetClinic Startup

azul

SOLUTIONS

01 CDS/AppCDS

02 Native Image

03 Leyden

04 CRaC

05 Ready Now

06 Cloud Native Compiler / JIT Server

azul

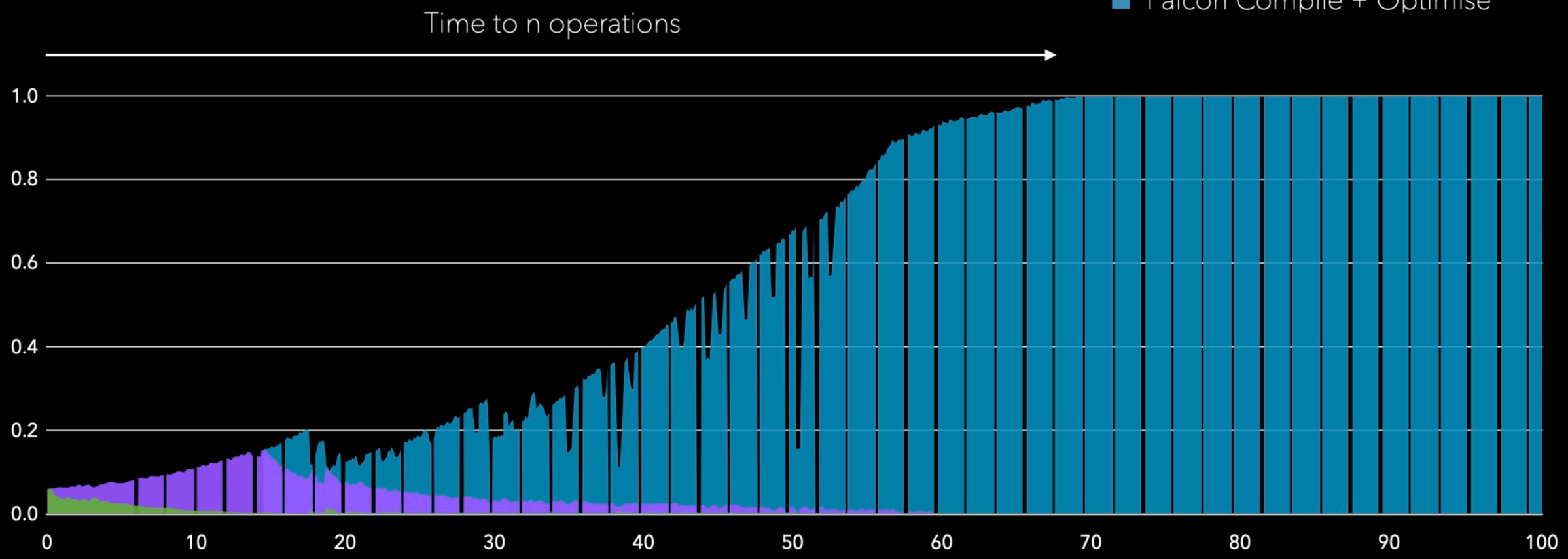
AZUL ZING READY NOW

Store Compiler Optimisations

- Store all optimisations applied to methods by the Zing Falcon compiler in a profile
- The profile will be loaded at startup
- Tries to compile all methods from the profile directly at startup (usually 90%)
- No Interpretation and C1 compilation needed for methods in the profile
- Drastic improvement in JVM warmup
- Can be combined with CRaC to also solve the JVM startup

AZUL ZING READY NOW

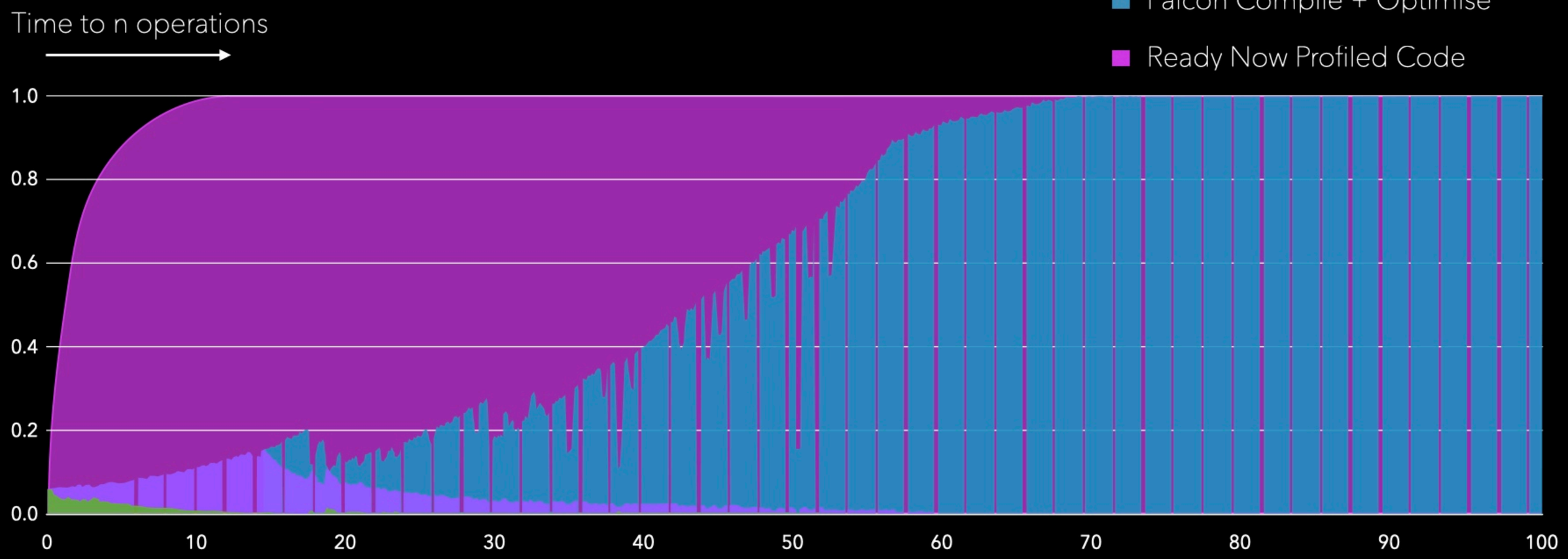
JVM WarmUp Performance



azul

AZUL ZING READY NOW

JVM WarmUp Performance



azul

SOLUTIONS

01 CDS/AppCDS

02 Native Image

03 Leyden

04 CRaC

05 Ready Now

06 Azul Cloud Native Compiler / IBM JIT Server

azul

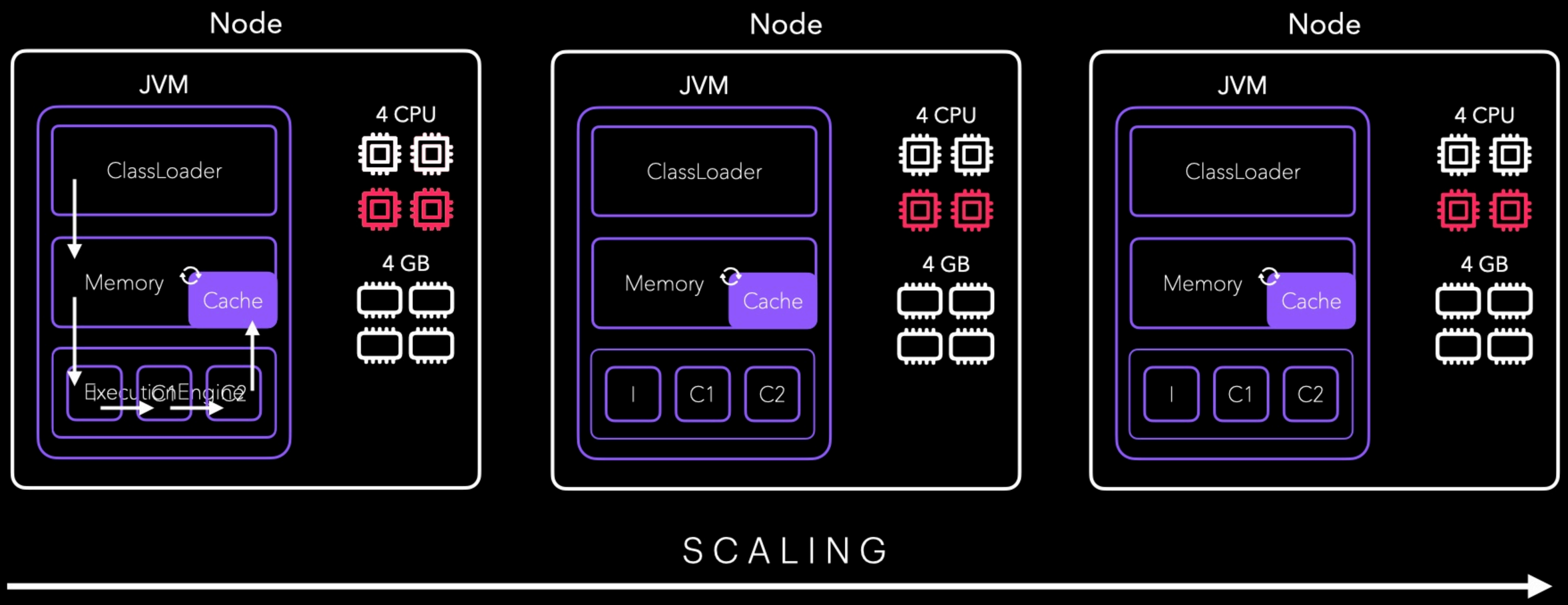
AZUL CNC / IBM JITSERVER

Compile centralised using Cloud Native Compiler

- Using a centralised JIT compiler
- Interpretation and C1 compilation happens locally (1st run only, after that ReadyNow profile)
- Code compilation and optimisation is done on the centralised JIT compiler
- If the centralised JIT Compiler is not available, the compilation will be done locally
- The centralised JIT Compiler keeps the compiled and optimised code in it's code cache
- First compile takes a bit longer but after that it get's faster

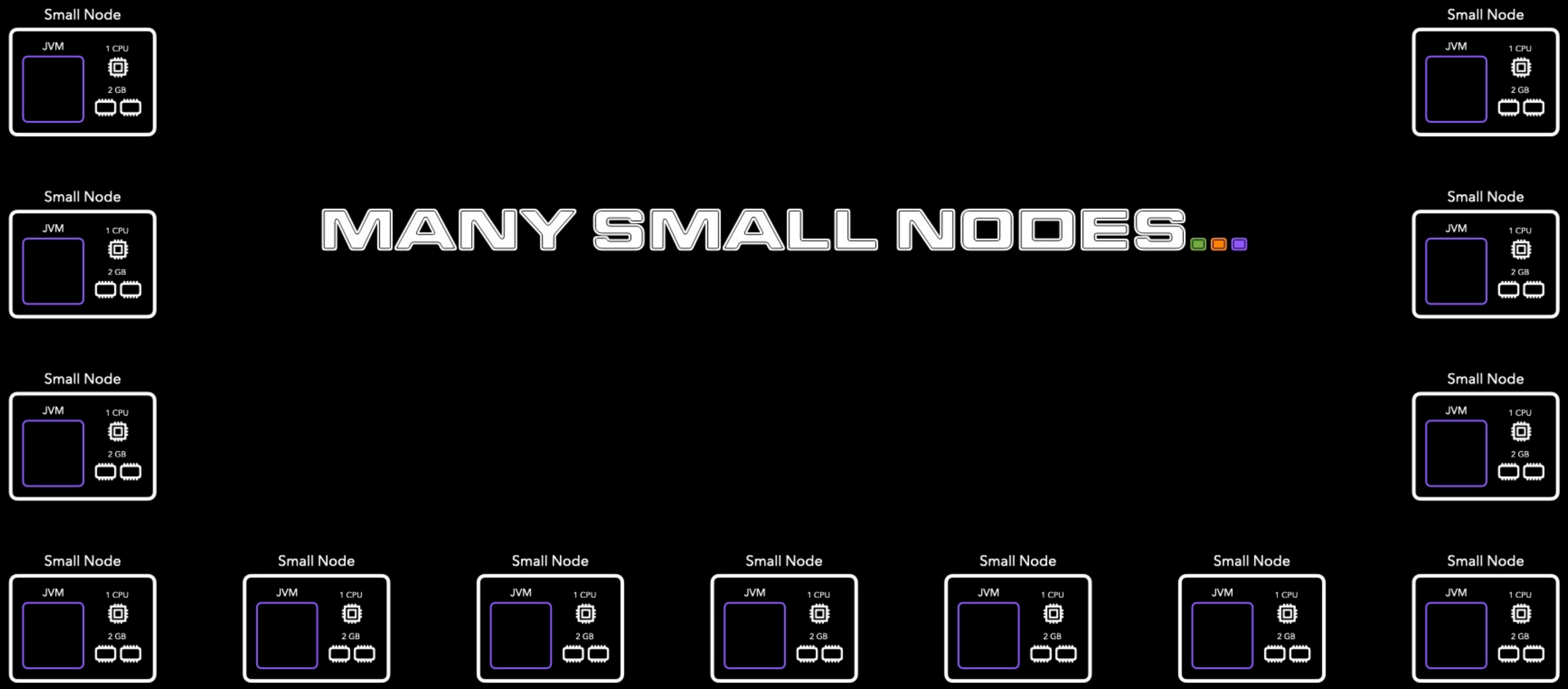
AZUL CNC / IBM JITSERVER

Local Compiler per Node



AZUL CNC / IBM JITSERVER

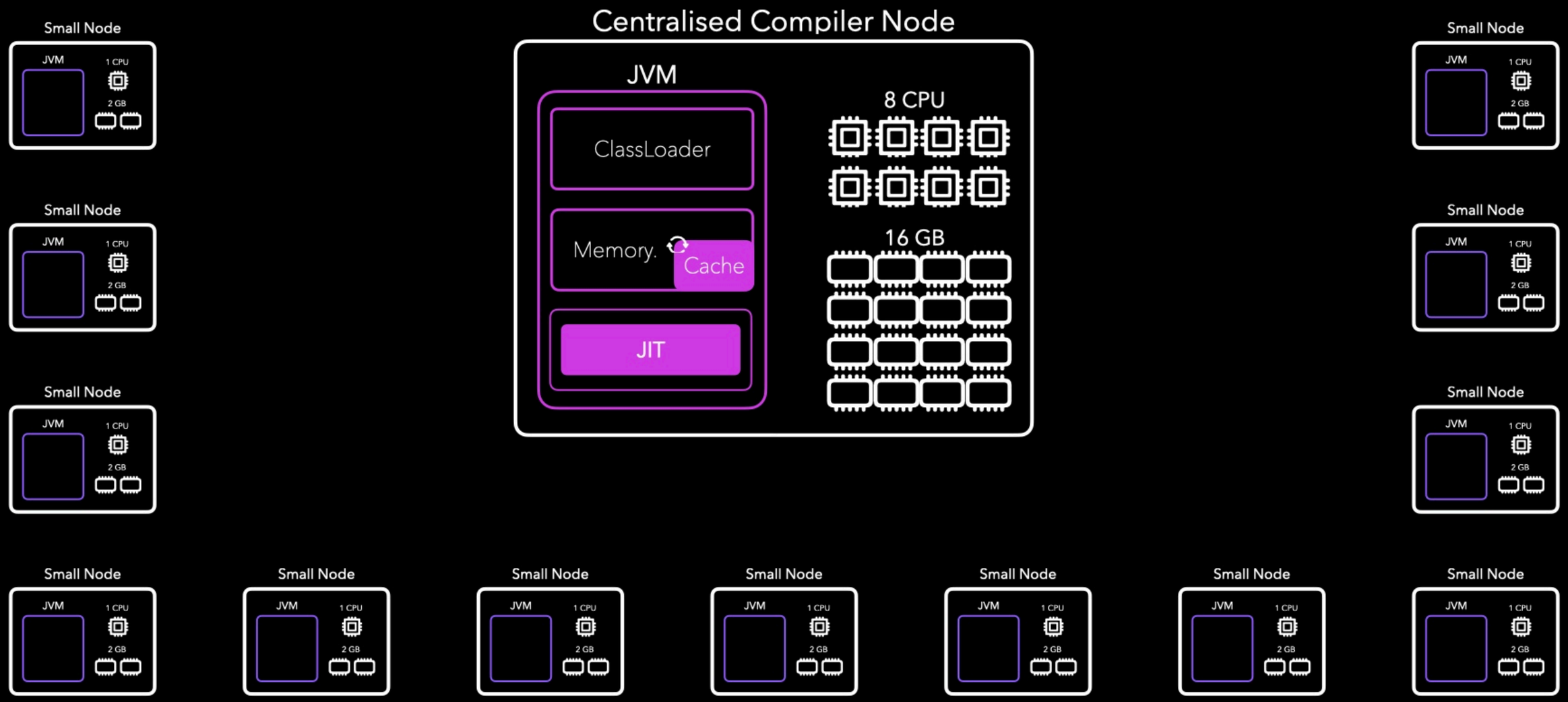
Centralised Compiler



azul

AZUL CNC / IBM JITSERVER

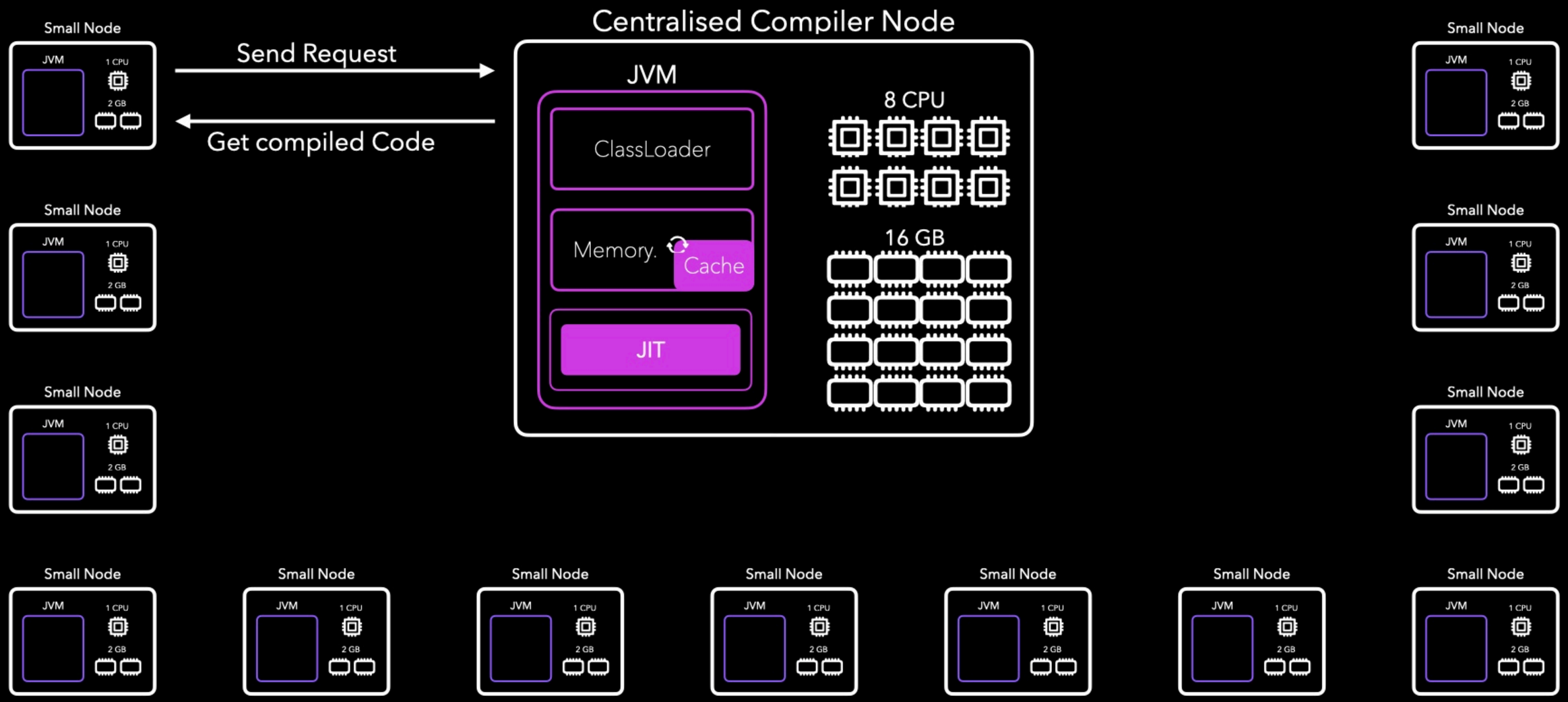
Centralised Compiler



azul

AZUL CNC / IBM JITSERVER

Centralised Compiler



azul

TESTS...

azul

TESTS

Setup

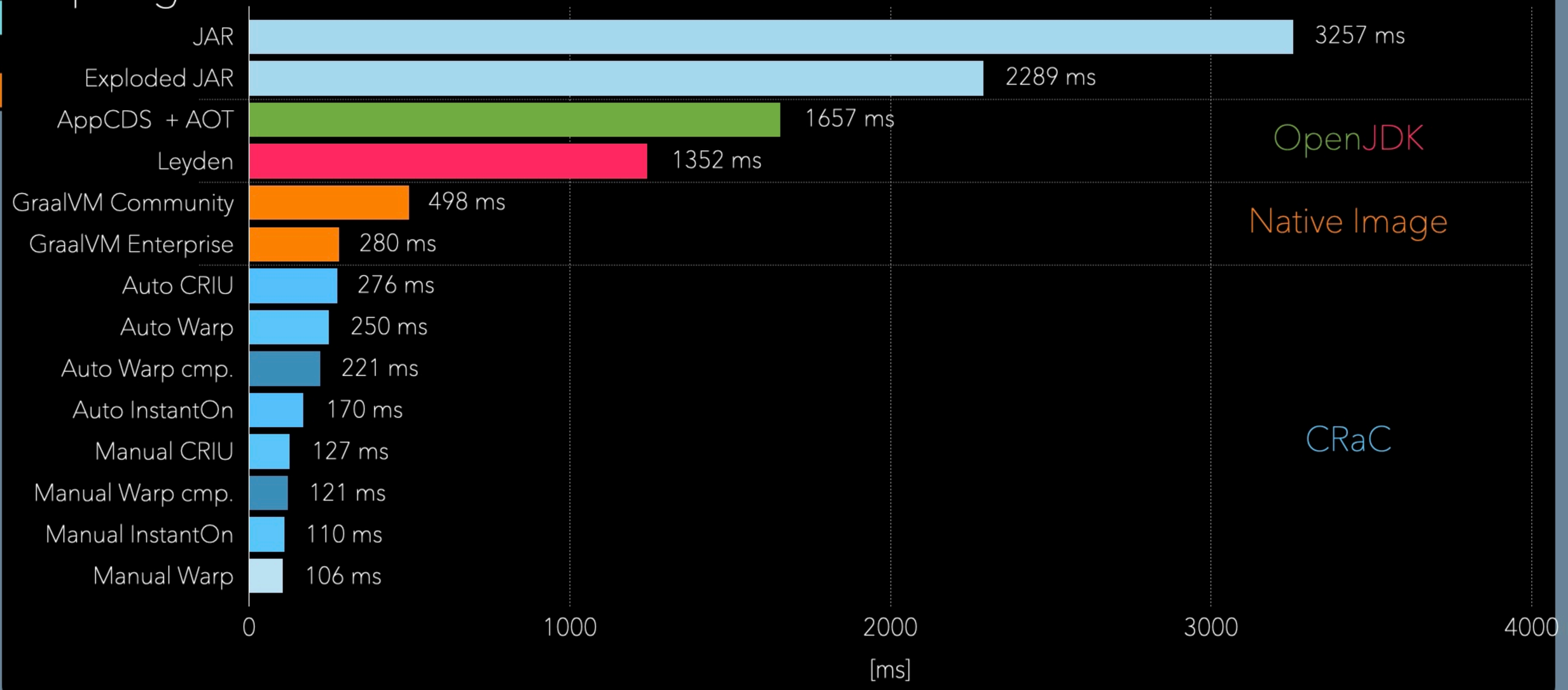
- SpringBoot 3.3 PetClinic
- Macbook Pro M4 Pro on MacOS 26
- Parallels Desktop 26.1.1
- 2 CPUs, 4 GB RAM
- Ubuntu 24.04 for ARM64
- Azul Zulu 25.0.0, Semeru 25.0.0, GraalVM Community 25 and GraalVM 25
- No extensive warm up
- Restart Virtual Machine before each test

STARTUP...

azul

STARTUP

SpringBoot PetClinic



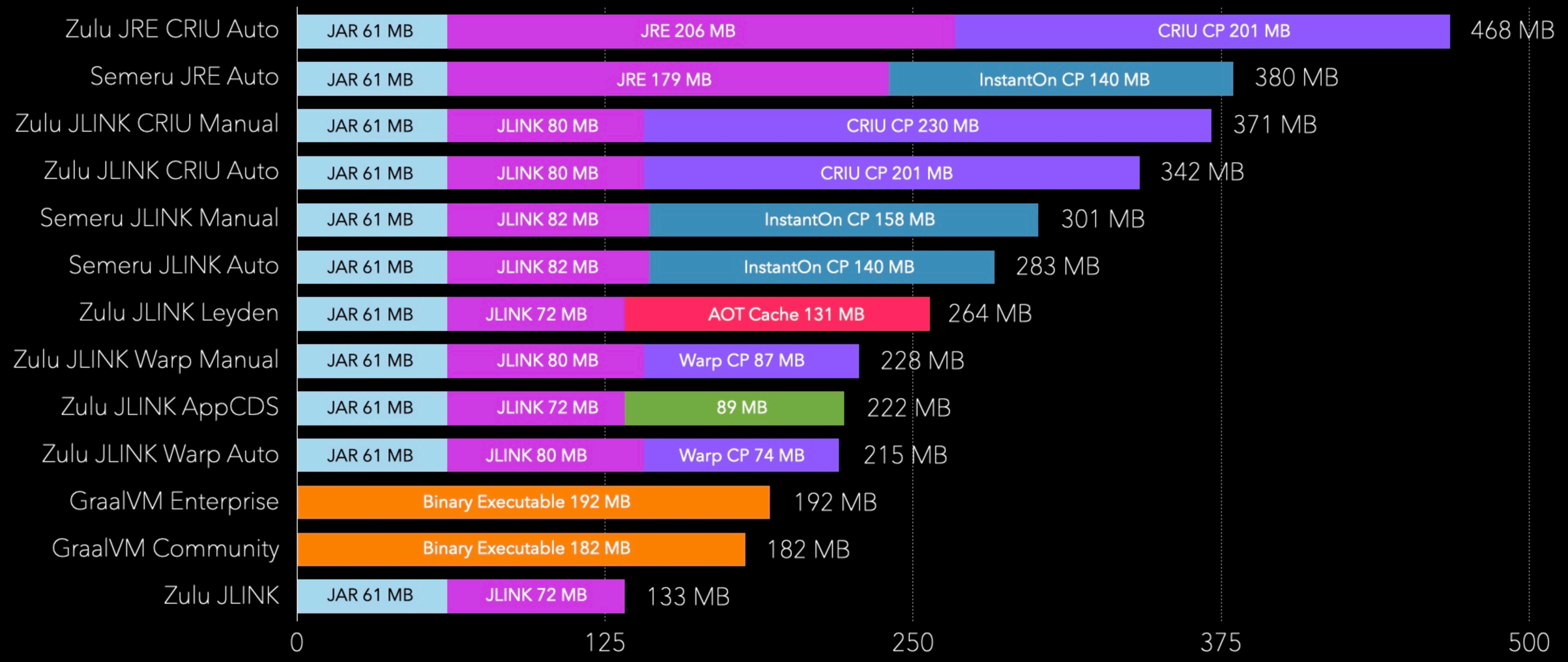
azul

MEMORY FOOTPRINT...

azul

MEMORY FOOTPRINT

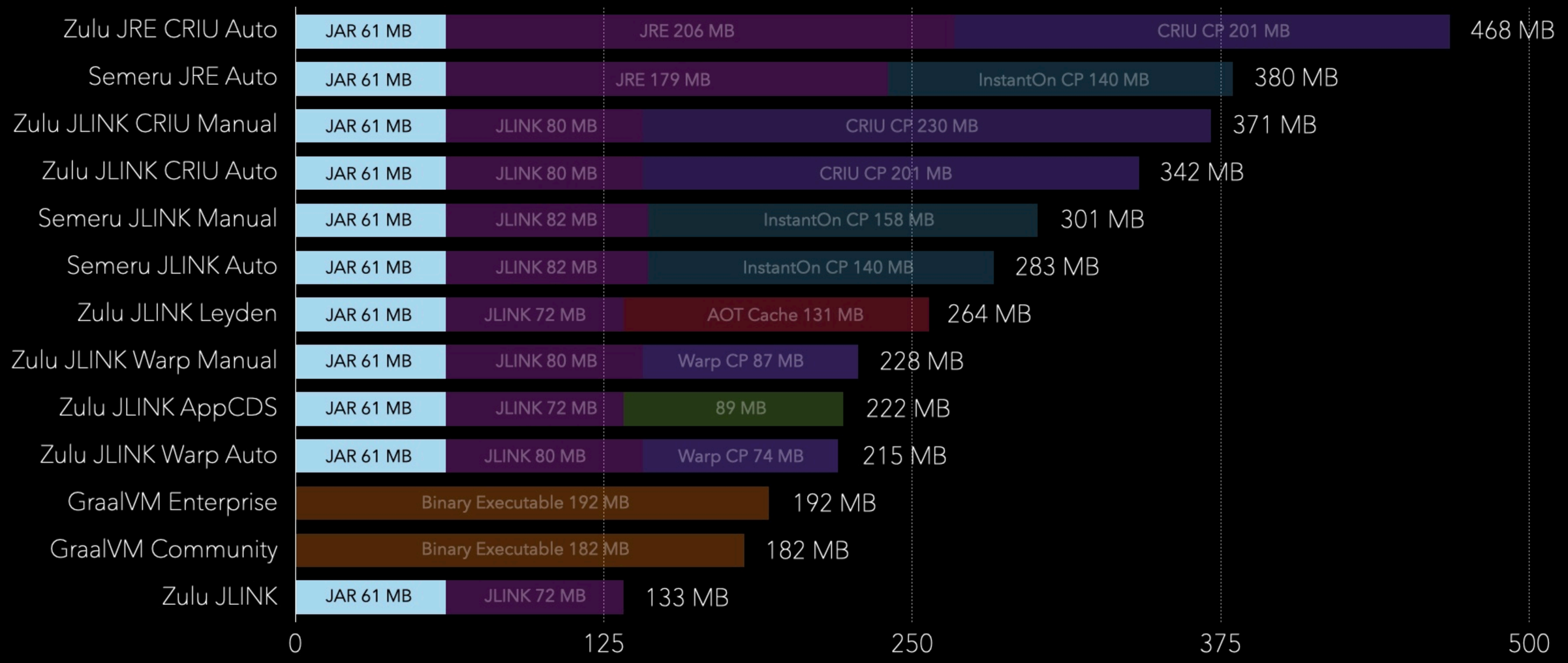
SpringBoot PetClinic (Deployment artifact + Runtime + CheckPoint/AOT Cache)



azul

MEMORY FOOTPRINT

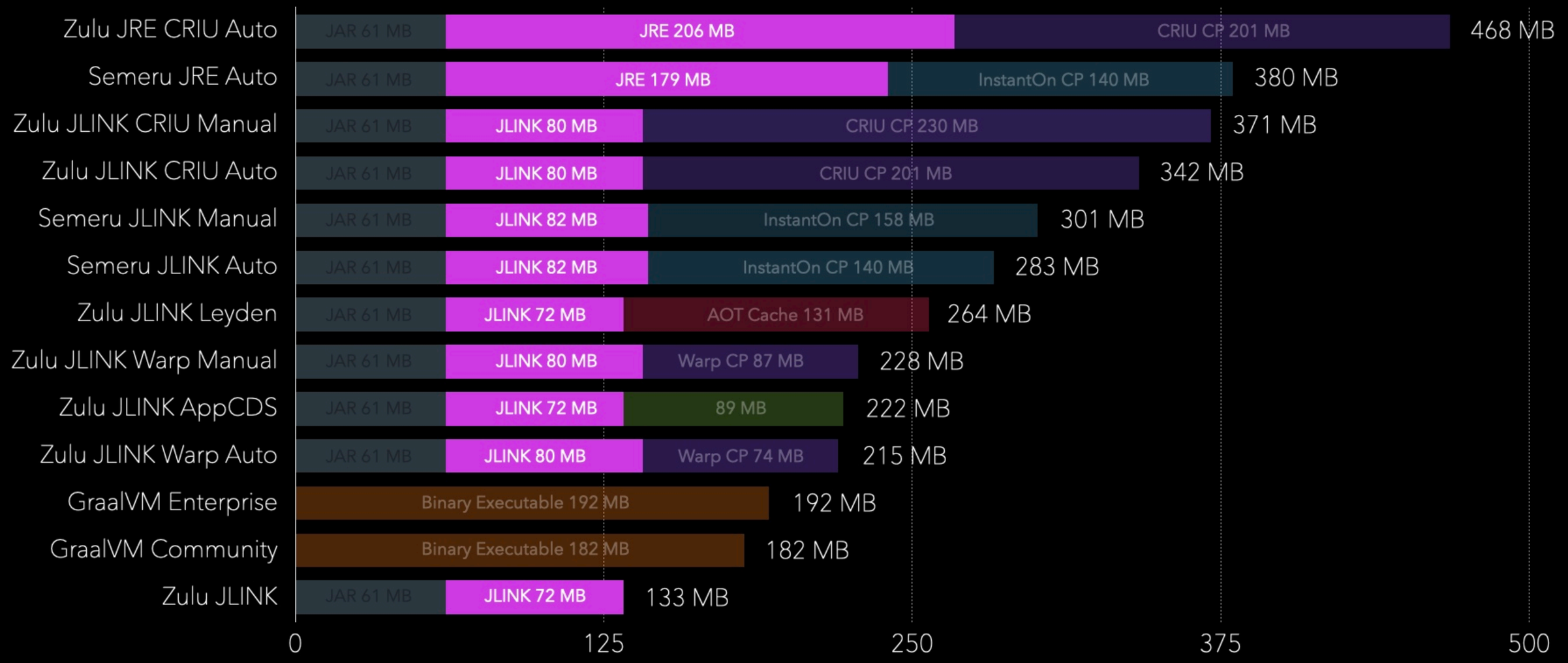
SpringBoot PetClinic (Deployment artifact + Runtime + CheckPoint/AOT Cache)



azul

MEMORY FOOTPRINT

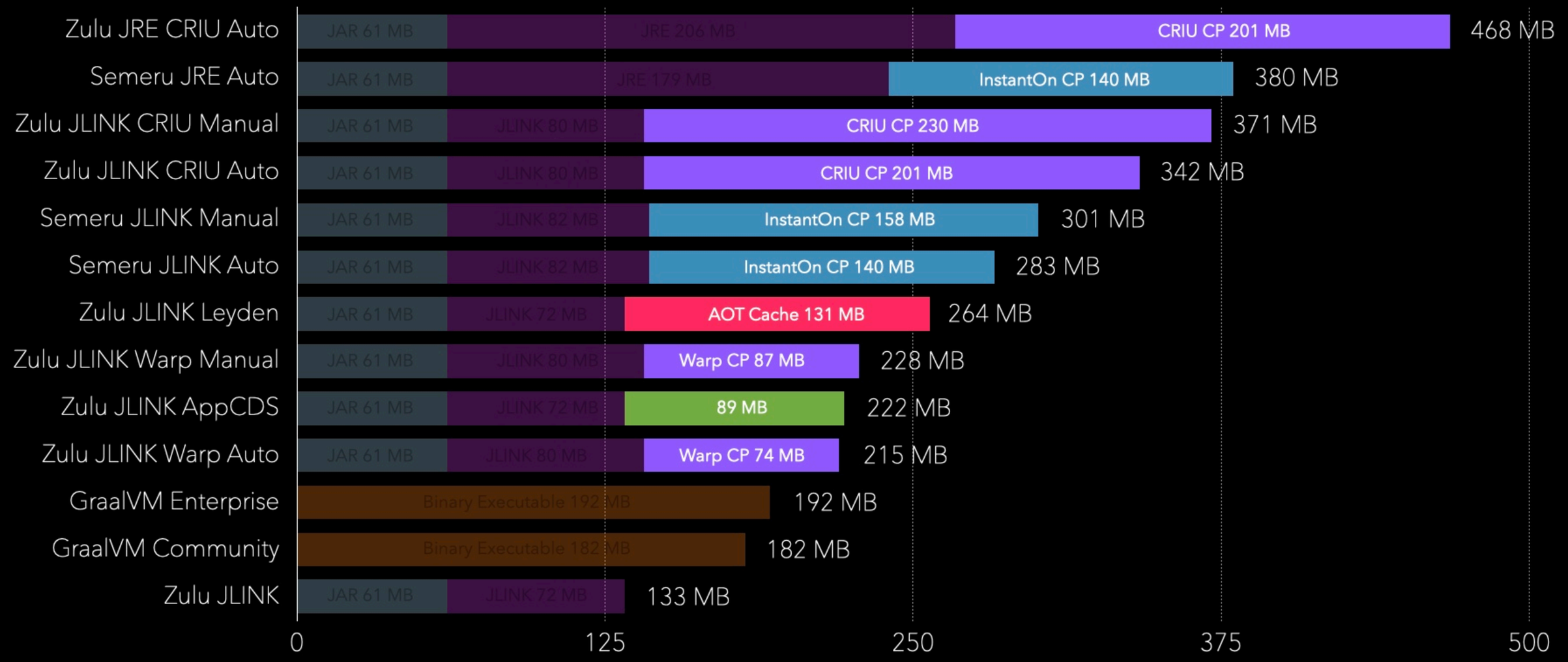
SpringBoot PetClinic (Deployment artifact + Runtime + CheckPoint/AOT Cache)



azul

MEMORY FOOTPRINT

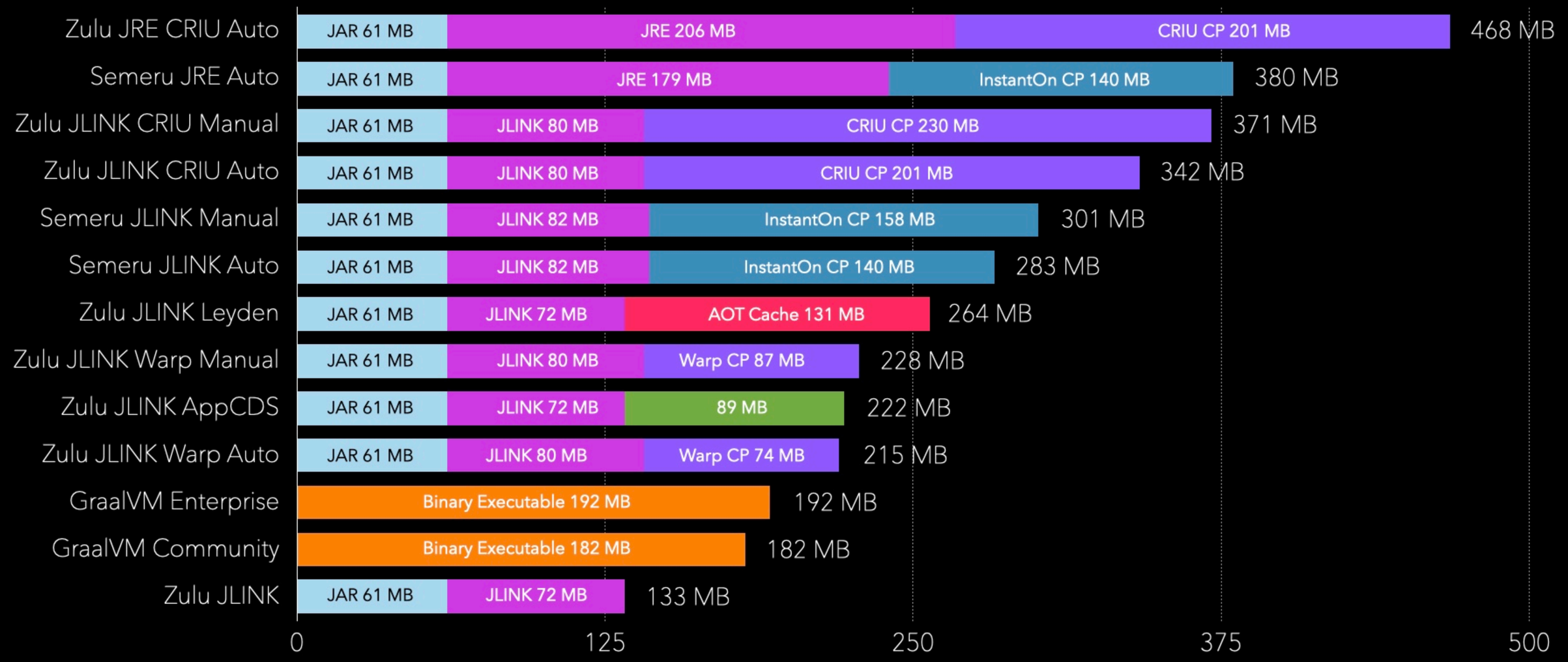
SpringBoot PetClinic (Deployment artifact + Runtime + CheckPoint/AOT Cache)



azul

MEMORY FOOTPRINT

SpringBoot PetClinic (Deployment artifact + Runtime + CheckPoint/AOT Cache)



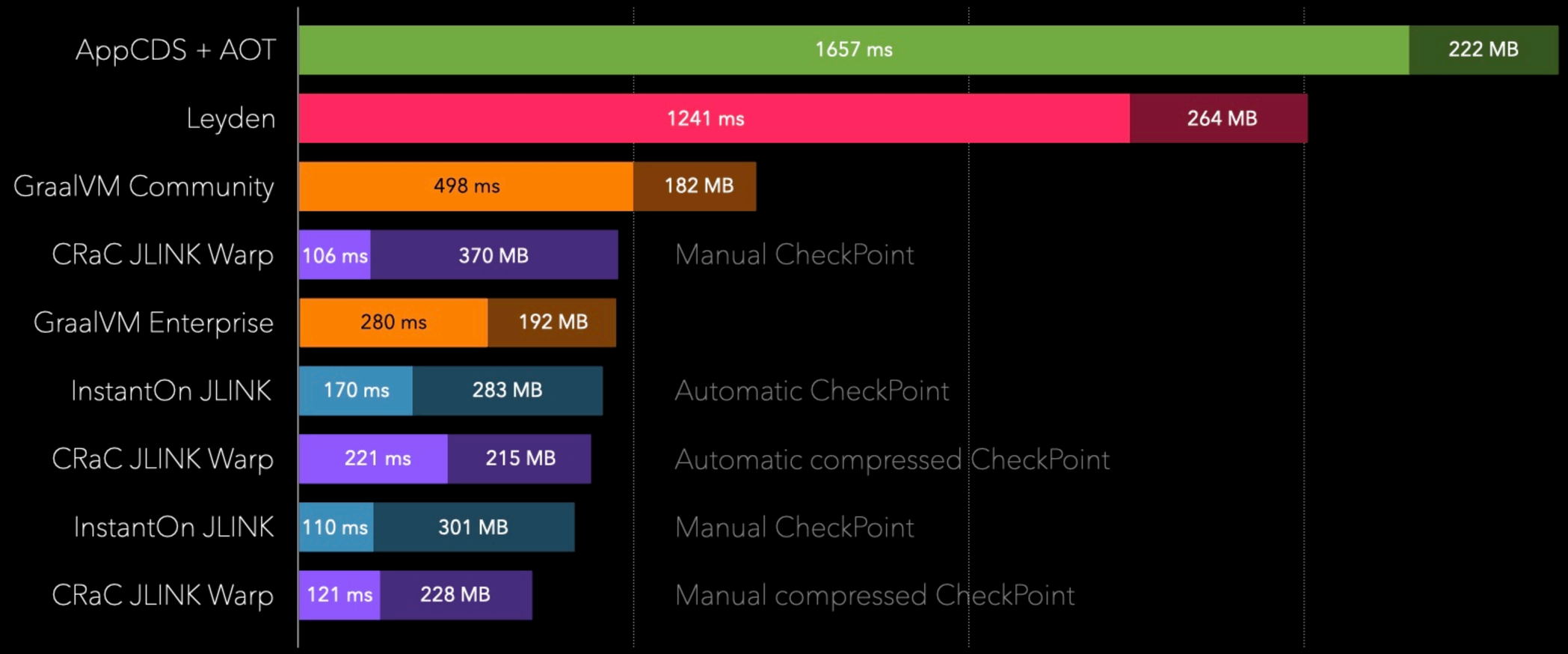
azul

BEST COMBINATIONS...

azul

BEST COMBINATIONS

SpringBoot PetClinic



SPEED + MEMORY FOOTPRINT

azul

WHICH SOLUTIONS IS FOR YOU...?

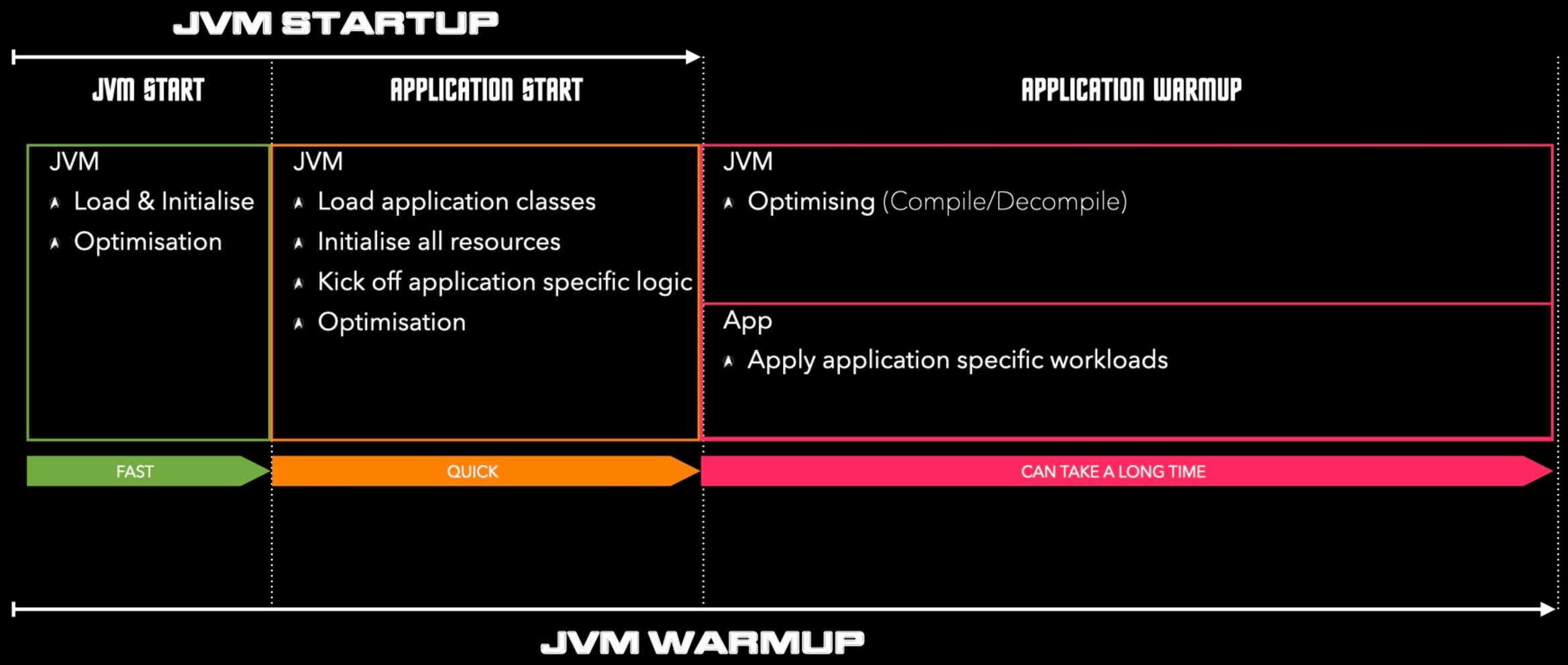
azul

IDENTIFY YOUR PROBLEM...

azul

IDENTIFY YOUR PROBLEM

JVM Startup and Warmup



EXAMPLE STARTUP PROBLEM...

• ONLINE SHOP •

STARTUP PROBLEM

Online Shop on Black Friday

You need to start fast to please the customers

Once they are in the store, they start searching

Performance is not really an issue but startup

- Native Image
- Leyden
- CRaC
- CNC / JITServer

EXAMPLE WARMUP PROBLEM...

- TRADING PLATFORM -

azul

WARMUP PROBLEM

Trading Platform

You need to be as warmed up as possible at 8am

Customers expect the first trade to be as fast as a trade later in the day

Startup is not really an issue but performance

- Ready Now
- Leyden
- CRaC
- CNC / JITServer

EXAMPLE STARTUP + WARMUP PROBLEM...

• GAMING PLATFORM •

azul

STARTUP + WARMUP PROBLEM

Gaming Platform

You need to start fast and be as warm as possible

Gamers expect to jump into a game and be able to compete with others immediately

Startup and performance do really matter

Leyden

CRaC

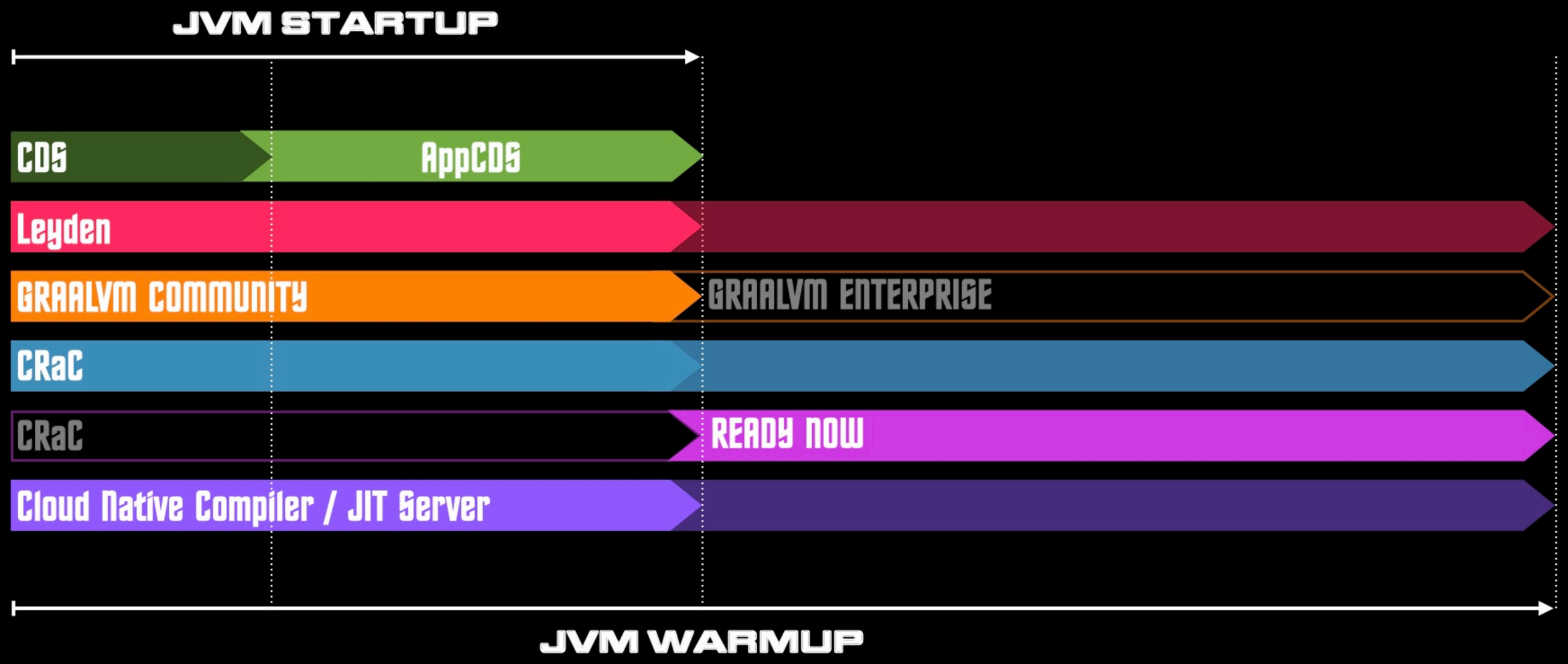
CNC / JITServer

OVERVIEW...

azul

IDENTIFY YOUR PROBLEM

Solutions and their strengths



THANK YOU

azul