KARSTEN SILZ, 4. NOVEMBER 2026

WIE KANN ICH JAVA SCHNELLER STARTEN – UND WANN LOHNT SICH DAS?





atra.consulting

WANN KANN ES SICH LOHNEN, JAVA SCHNELLER ZU STARTEN?

SERVERLESS

VIELE SERVER

SEHR HOHE VERFÜGBARKEIT

1. SPRING BOOT TUNING

2. CLASS DATA SHARING (CDS) & OPENJDK PROJECT LEYDEN

3. OPENJDK PROJECT CRAC

4. GRAALVM NATIVE IMAGE

1. SPRING BOOT TUNING

2. CLASS DATA SHARING (CDS) & OPENJDK PROJECT LEYDEN

3. OPENJDK PROJECT CRAC

4. GRAALVM NATIVE IMAGE

WAS MACHT MICH ZUM EXPERTEN?

2022: BETREUTE ARTIKEL-SERIE ÜBER GRAALVM

SEITDEM: ARTIKEL & VORTRÄGE ZU GRAALVM, LEYDEN & CRAC VERFASST

FOLIEN, CODE, ROI-BEISPIEL & MEHR



HTTPS://ATRA.SOFTWARE/WJ1

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

LOHNT SICH DAS?

SERVERLESS

VIELE SERVER

SEHR HOHE VERFÜGBARKEIT

SERVERLESS

ABRECHNUNG NACH LAUFZEIT IN MS

SCHNELLERER STARTUP = GERINGERE KOSTEN

AWS LAMBDA SNAPSTART: LAMBDA MIT CRAC

VIELE SERVER

VIELE STANDBY-INSTANZEN FÜR LASTSPITZEN:

SCHNELLERER STARTUP =
WENIGER INSTANZEN =
WENIGER SERVIER

WENN ANWENDUNGEN NICHT CPU-LIMITIERT:

WENIGER SPEICHER/APP =

MEHR INSTANZEN/SERVER =

WENIGER SERVER

SEHR HOHE VERFÜGBARKEIT

DOWNTIME PRO JAHR & VERFÜGBARKEITS-LEVEL

99.9%: KNAPP 9H

99.99%: GUT 52 MIN

99.999%: 5 MIN

WAS SIE SOWIESO MACHEN: MEHRERE INSTANZEN, ROLLING UPGRADES

SCHNELLER STARTUP: VERSICHERUNG FÜR GEPLANTE ODER UNGEPLANTE AUSFÄLLE

BEWERTUNG DER VARIANTEN

KOSTEN

EINMAL: BUILD & DEPLOYMENT,
ENTWICKLER-TRAINING

WIEDERKEHREND:
WARTUNG, UPGRADES

DEV EXPERIENCE

LOKALE ENTWICKLUNG
LANGSAMER, PRODUKTIONS-
TROUBLESHOOTING SCHLECHTER

	TUNING	CDS & LEYDEN	CRAC	GRAALVM
ZEIT: SERVERLESS, STANDBY, VERFÜG-BARKEIT				
SPEICHER: WENIGER SERVER				
KOSTEN				
DEV EXPERIENCE				

ROI: GRAALVM ÜBER 3 JAHRE

KOSTEN (PERSONAL & HARDWARE): 56.080 €

SPAREN: 6X EC2 T3.2XLARGE (32 GB & 8 CPUS) FÜR 3 JAHRE

FOLIEN, CODE, ROI-BEISPIEL & MEHR



HTTPS://ATRA.SOFTWARE/WJ1

DAS LOHNT SICH MANCHMAL!

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

WARUM JETZT?

ANWENDUNGEN IMMER MEHR IM CONTAINER – PROGRAMMIERSPRACHE EGAL: JAVASCRIPT, C#, PYTHON, GO, RUST, ...

DOCKERHUB: ÖFFENTLICHE & KOSTENLOSE CONTAINER-IMAGES = BILLIGE INFRASTRUKTUR: DATENBANKEN, WEB-SERVER, PROXIES, CACHES, ...

STANDARDISIERTE KOMMUNIKATION VON ANWENDUNGEN:

REST/GRAPHQL + JSON

CONTAINER + DOCKERHUB + REST/GRAPHQL + JSON =

NEUE KONKURRENZ

MONATLICHE ABRECHNUNG FÜR CONTAINER AUF SERVERN = SICHTBARE SERVER-KOSTEN

JAVA BY DESIGN:

MEHR ZEIT

MEHR CPU

MEHR SPEICHER

JAVA BRAUCHT MEHR SERVER-RESSOURCEN

JAVA TEURER ALS NEUE KONKURRENZ

DARUM JETZT!

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

WIESO LANGSAM?

JAVA-ANWENDUNGEN SIND BYTECODE:

.CLASS-DATEIEN IN JAR-ARCHIVEN AUF LINUX, WINDOWS, MACOS...

...ABER WIRD

INTERPRETIERT

–LANGSAM!

JIT-COMPILER (JUST IN TIME): BYTECODE ZU MASCHINENCODE

STANDARD JIT-COMPILER: "HOTSPOT" SEIT 1999

HOTSPOT-TEILE

PROFILER

VERMERKT, WIE OFT
METHODEN & SCHLEIFEN
DURCHLAUFEN WERDEN

C1 COMPILER

SCHNELLE COMPILIERUNG => NICHT-OPTIMIERTER MASCHINENCODE

MASCHINENCODE ENTHÄLT PROFILING-CODE

C2 COMPILER

LANGSAME COMPILIERUNG =>
OPTIMIERTER MASCHINENCODE

METHOD INLINING, LOOP UNROLLING,
SCALAR REPLACEMENT, VECTORIZATION
OF ARITHMETIC LOOPS, ...

WIE STARTET JAVA?

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

BUILD TIME

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

STARTUP TIME & TIME TO PEAK PERFORMANCE

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

STARTUP TIME

BUILD TIME

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

BUILD TIME

TIME TO PEAK PERFORMANCE

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

JAVA BY DESIGN:

MEHR ZEIT

MEHR CPU

MEHR SPEICHER

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

BUILD TIME

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

BUILD TIME

LÄUFT JEDES MAL, GLEICHES RESULTAT: VIELE JAVA-OBJEKTE

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

MEHR ZEIT FÜR STARTUP

MEHR CPU

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

BUILD TIME

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

BUILD TIME

PROFILING
C1 COMPILER
PROFILING
C2 COMPILER

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

MEHR ZEIT FÜR
TIME TO PEAK

MEHR CPU

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

PROFILER + 2 COMPILER IN APP!

BUILD TIME

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

MEHR SPEICHER

JAVA BY DESIGN:

MEHR ZEIT

MEHR CPU

MEHR SPEICHER

BY DESIGN?

"SIMPLE, OBJECT-ORIENTED, DISTRIBUTED, INTERPRETED, ROBUST, SECURE, ARCHITECTURE-NEUTRAL, PORTABLE, HIGH-PERFORMANCE, MULTITHREADED DYNAMIC LANGUAGE"

ORIGINALE JAVA-BESCHREIBUNG

"SIMPLE, OBJECT-ORIENTED, DISTRIBUTED, INTERPRETED, ROBUST, SECURE, ARCHITECTURE-NEUTRAL, PORTABLE, HIGH-PERFORMANCE, MULTITHREADED DYNAMIC LANGUAGE"

ORIGINALE JAVA-BESCHREIBUNG

BY DESIGN!

JAVA BY DESIGN:

MEHR ZEIT

MEHR CPU

MEHR SPEICHER

DARUM LANGSAM!

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

WIE SCHNELLER?

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

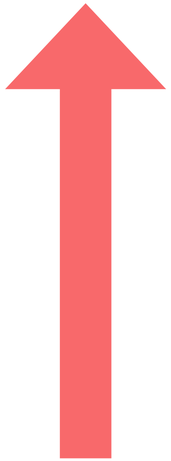
BYTE-CODE

JAR

IMAGE

ARBEIT ZUM BUILD VERLAGERN

BUILD TIME



JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

ERGEBNISSE CACHEN

BUILD TIME

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

INIT JDK & FRAMEWORK

INIT APP

...

MASCHINEN-CODE

RUNTIME

ARBEIT ZUM BUILD VERLAGERN

ERGEBNISSE CACHEN

BEISPIEL: SPRING PETCLINIC

WEB-ANWENDUNG MIT THYMELEAF, SPRING DATA & POSTGRESQL

BENCHMARK

AWS EC2 2 GB RAM, 2 CPUS

LOKALER POSTGRES-SERVER

SPRING BOOT 4.1.2

JAVA 25/27

MESSEN STARTUP TIME

DANN 5 X DIE GLEICHEN 11 SEITEN AUFRUFEN

BENCHMARK MIT DIESEN PARAMETERN:

-XMS256M -XMX768M
-XX:+USEG1GC
--SPRING.PROFILES.
ACTIVE=POSTGRES

3 WARMUP-LÄUFE,
7 BENCHMARK-LÄUFE,
MITTELWERT VON 5 LÄUFEN

ERGEBNISSE GELTEN NUR FÜR PETCLINIC IN MEINER UMGEBUNG!

SCRIPTS SIND IN MEINEM PETCLINIC-FORK – MESSEN SIE FÜR IHRE ANWENDUNG!

./COMPILE-AND-RUN.SH [BASELINE | TUNING | CDS | LEYDEN | CRAC | GRAALVM]

FOLIEN, CODE, ROI-BEISPIEL & MEHR



HTTPS://ATRA.SOFTWARE/WJ1

0:

PERFORMANCE BASELINE

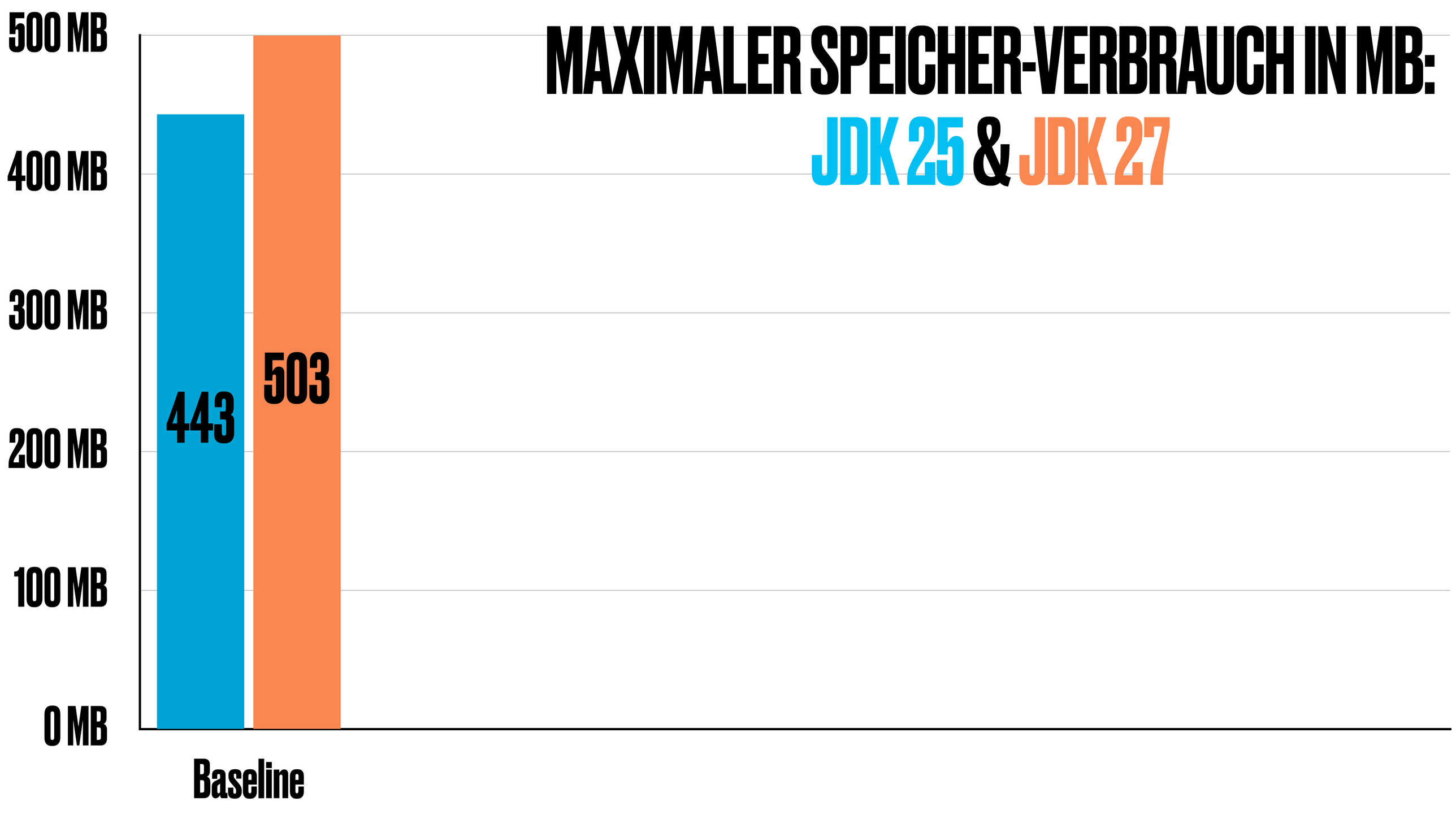
SPRING BOOT 4.1

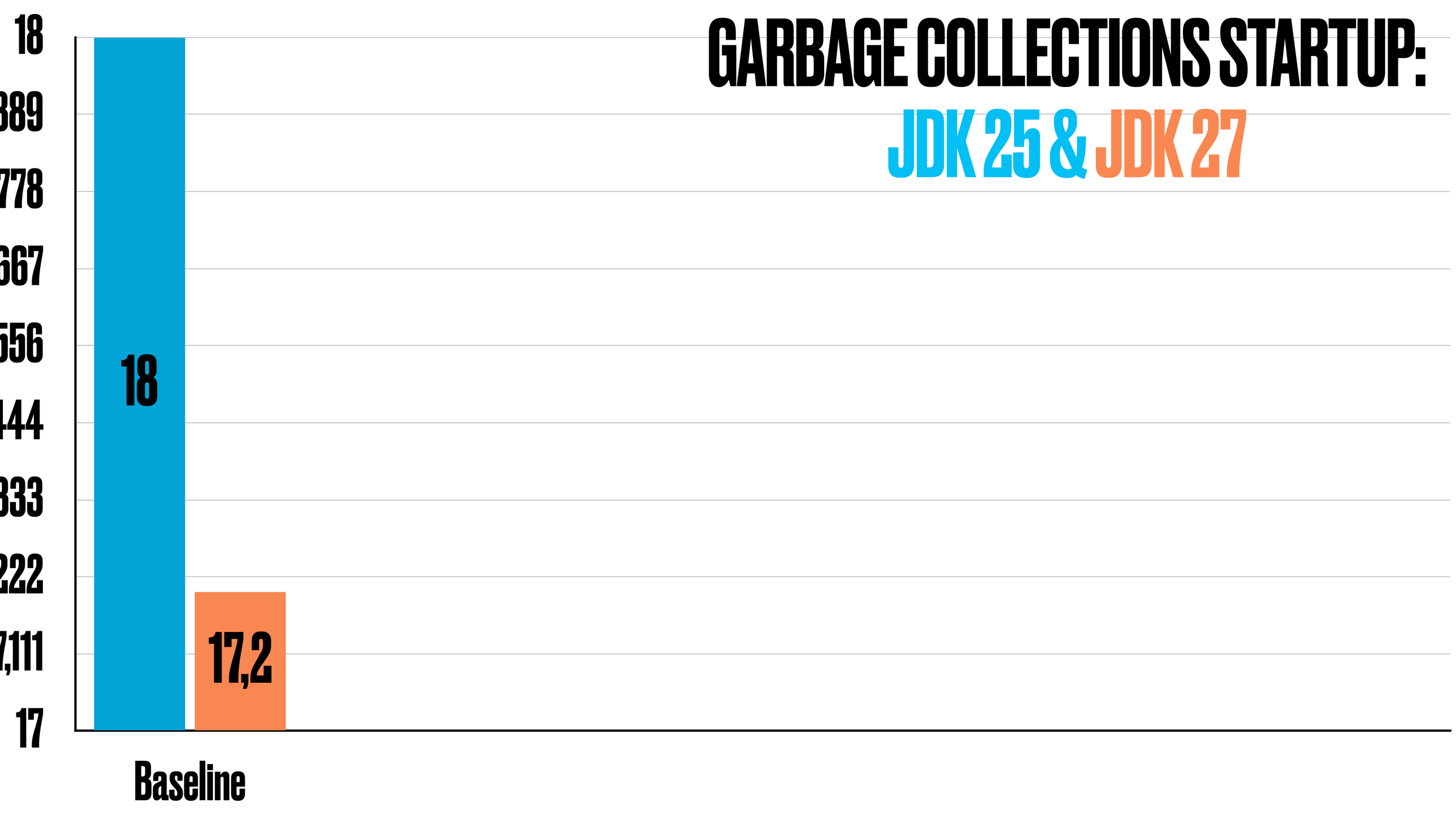
JDK 25/27

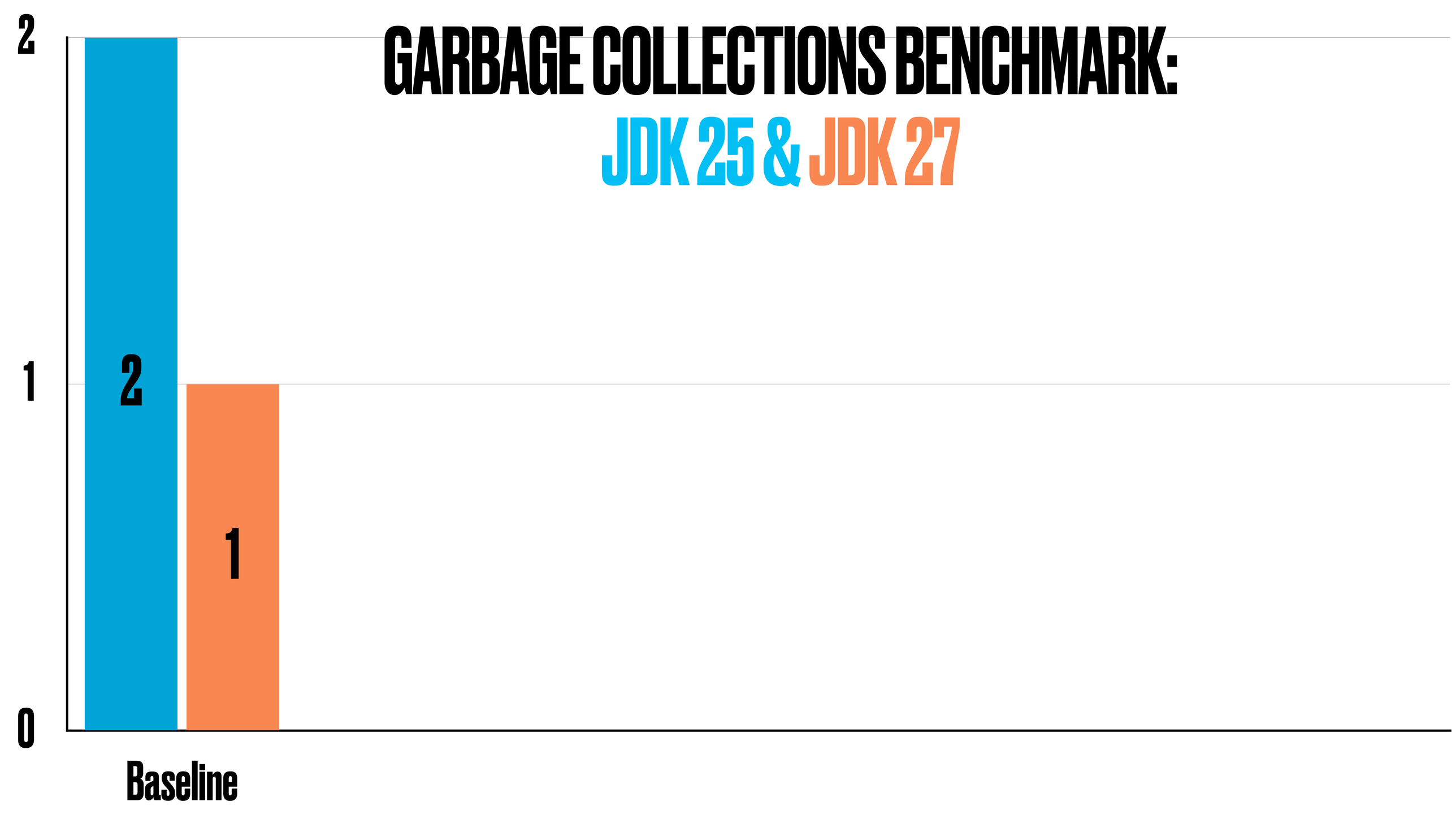
```
./GRADLEW CLEAN BOOTJAR

JAVA -JAR BUILD/LIBS/MY-APP-1.4.JAR
```


STARTUP-ZEIT IN SEKUNDEN:
JDK 25 & JDK 27
16 s
14 s
12 s
10 s
8 s
6 s
4 s
2 s
0 s
14,6
15,3
Baseline


MAXIMALER SPEICHER-VERBRAUCH IN MB:
JDK 25 & JDK 27
500 MB
400 MB
300 MB
200 MB
100 MB
0 MB
443
503
Baseline


GARBAGE COLLECTIONS STARTUP:
JDK 25 & JDK 27
18
389
778
667
556
444
333
222
7,111
17
18
17,2
Baseline


GARBAGE COLLECTIONS BENCHMARK:
JDK 25 & JDK 27
2
1
0
2
1
Baseline

FÜR JDK 25/27:
APP JAR: 64 MB

1:

SPRING BOOT TUNING

VOM SPRING-TEAM

VERLAGERT SCHRITTE ZUM BUILD

REDUZIERT STARTUP TIME

SPRING BOOT 3, JDK 17

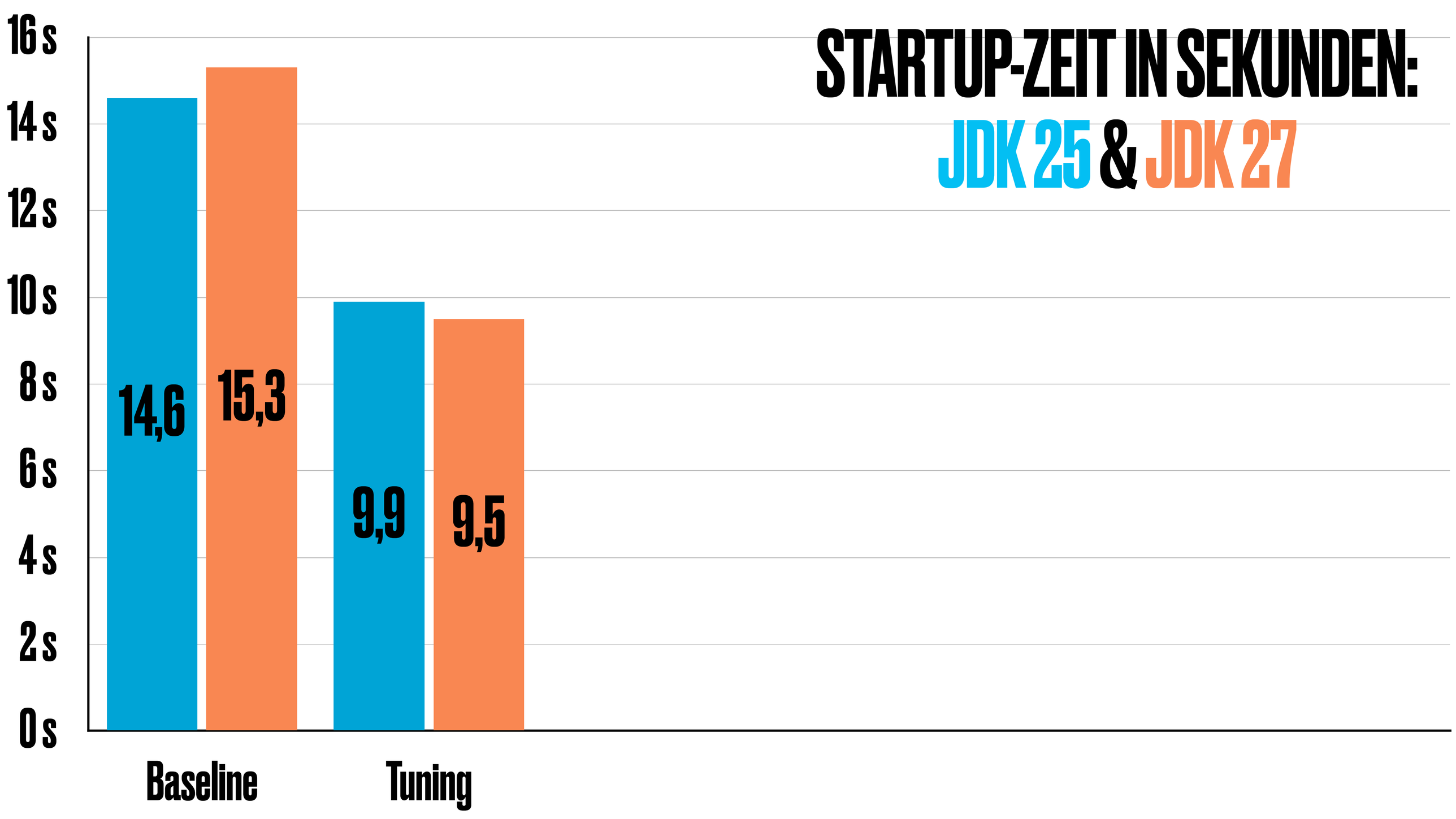
SPRING-BOOT-INITIALISIERUNG TEILWEISE IM BUILD

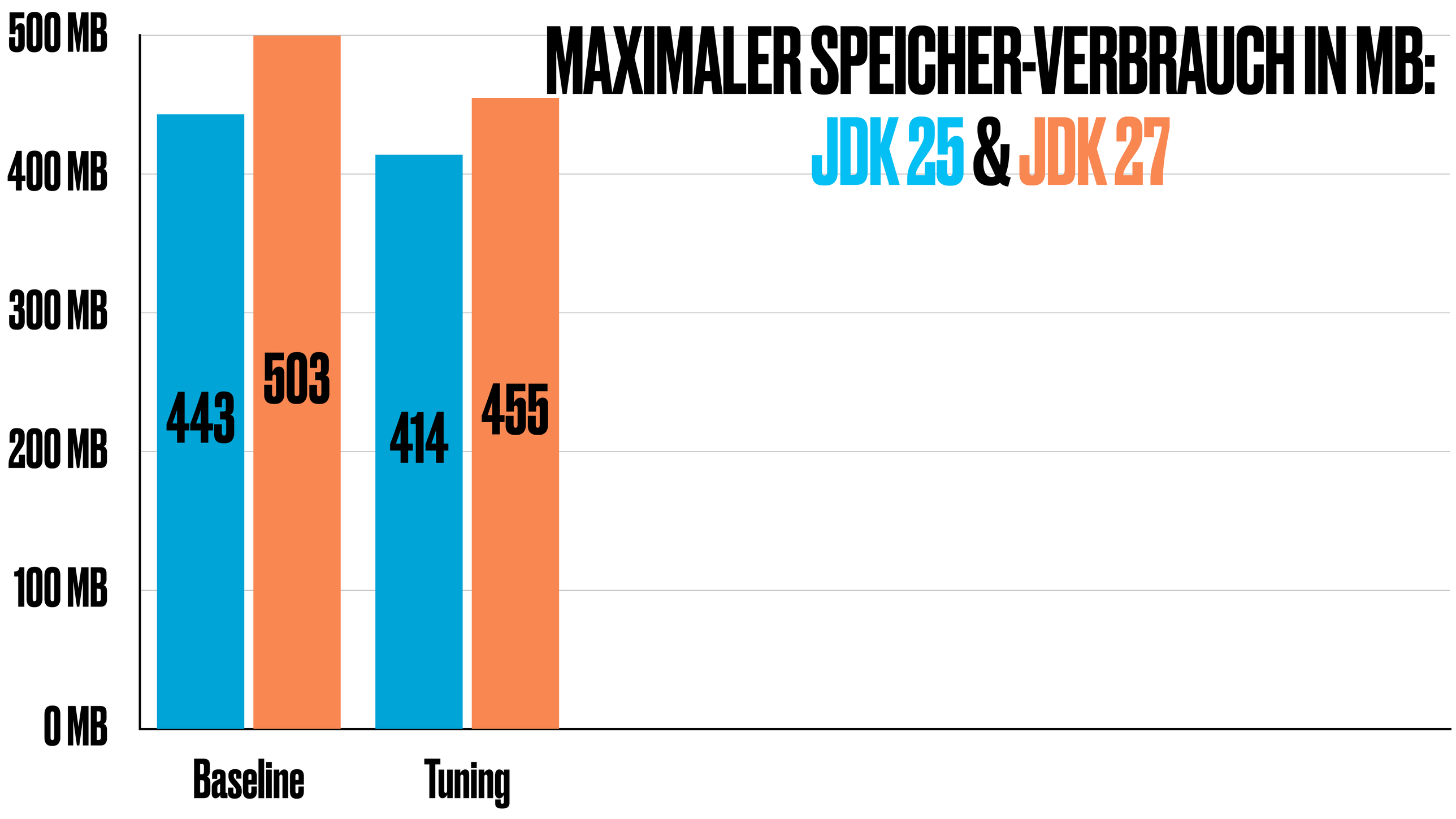
SPRING AOT ENTFERNT NICHT BENÖTIGTEN CODE & KONFIGURATION

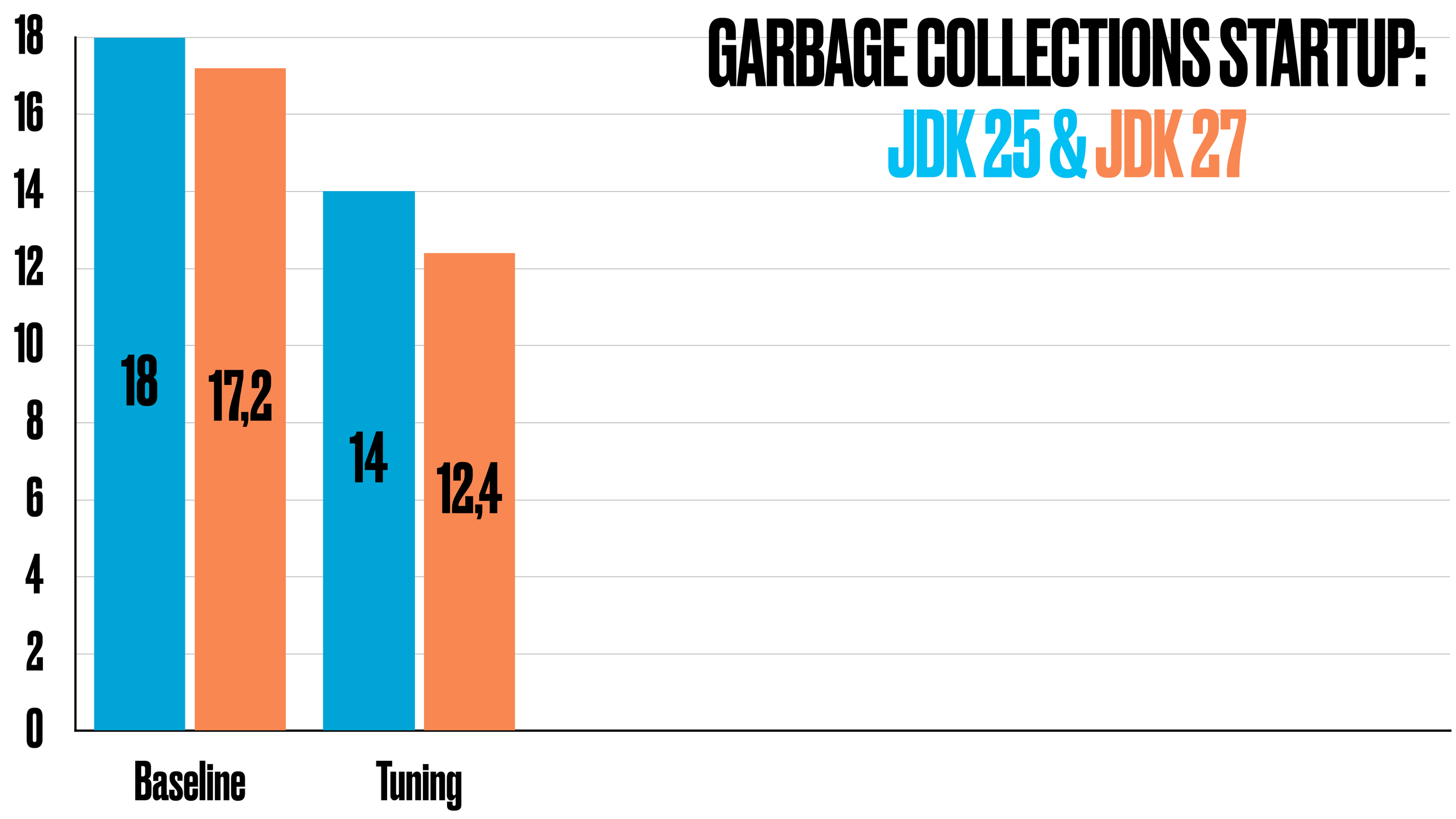
BOOT-JAR ENTPACKT

```
./GRADLEW CLEAN BOOTJAR && \
  JAVA -DJARMODE=TOOLS \
  -JAR BUILD/LIBS/MY-APP-1.4.JAR EXTRACT

JAVA -DSPRING.AOT.ENABLED=TRUE \
  -JAR MY-APP-1.4/MY-APP-1.4.JAR
```

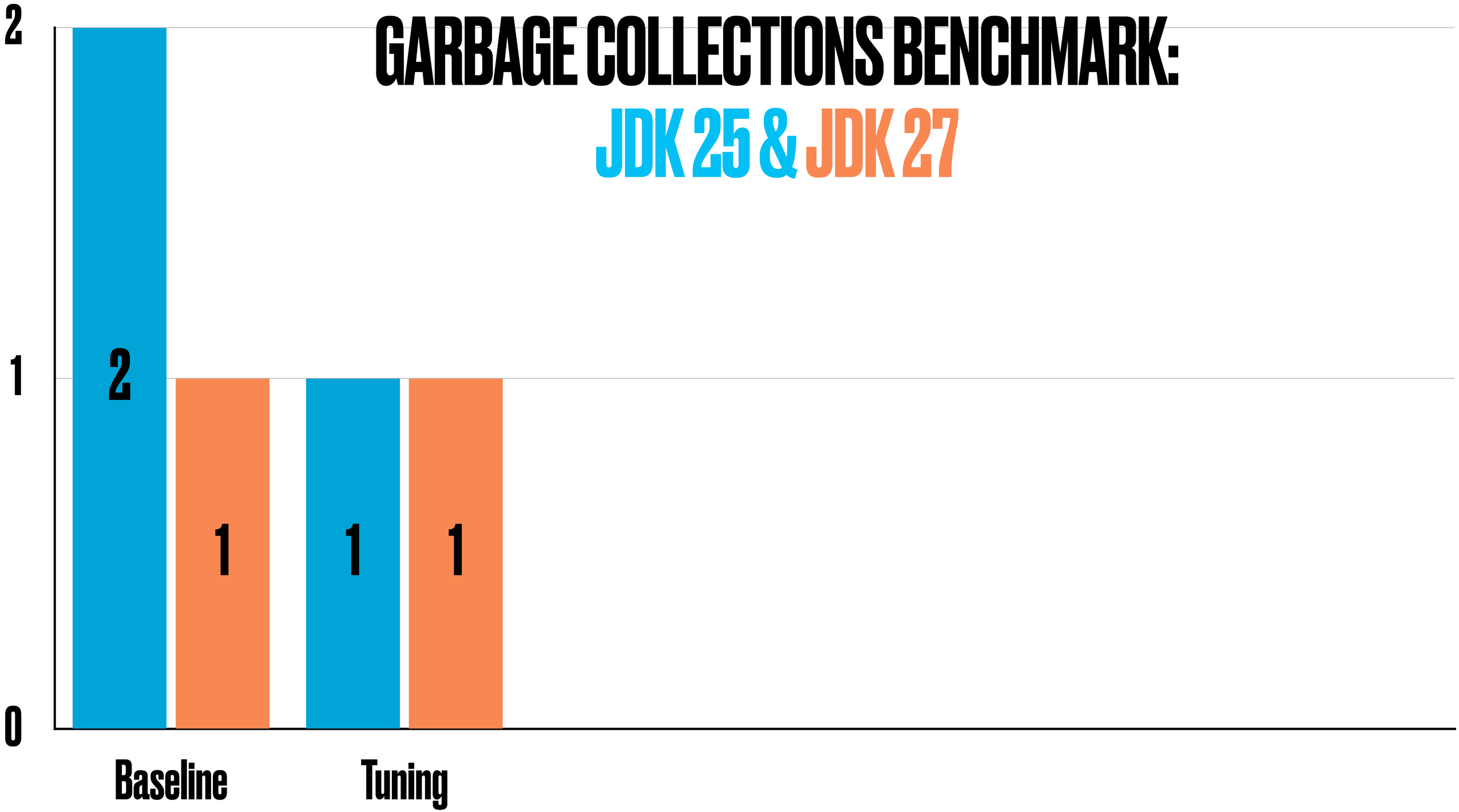

STARTUP-ZEIT IN SEKUNDEN:
JDK 25 & JDK 27
16 s
14 s
12 s
10 s
8 s
6 s
4 s
2 s
0 s
14,6
15,3
9,9
9,5
Baseline
Tuning


MAXIMALER SPEICHER-VERBRAUCH IN MB:
JDK 25 & JDK 27
500 MB
400 MB
300 MB
200 MB
100 MB
0 MB
443
503
414
455
Baseline
Tuning


GARBAGE COLLECTIONS STARTUP:
JDK 25 & JDK 27
18
16
14
12
10
8
6
4
2
0
18
17,2
14
12,4
Baseline
Tuning

GARBAGE COLLECTIONS BENCHMARK:

JDK 25 & JDK 27



FÜR JDK 25/27:

APP JAR: 1 MB

LIB JARS: 63 MB

	TUNING	CDS & LEYDEN	CRAC	GRAALVM
ZEIT: SERVERLESS, STANDBY, VERFÜG-BARKEIT				
SPEICHER: WENIGER SERVER				
KOSTEN				
DEV EXPERIENCE				

2:

CLASS DATA SHARING (CDS) & LEYDEN

CDS IN JAVA EINGEBAUT, LEYDEN OPEN-JDK-PROJEKT VON ORACLES JAVA-TEAM

CACHET ERGEBNISSE

REDUZIERT STARTUP TIME & TIME TO PEAK

CDS: JDK 17, LEYDEN: JDK 24

BRAUCHT ANWENDUNGS-TRAININGSLAUF

SCHREIBT BEI INITIALISIERUNG
ANFALLENDE DATEN IN CACHE-DATEI

ANWENDUNG LÄDT
CACHE-DATEI IN PRODUKTION

CDS:

CACHE-DATEI

MY-APP.JSA

```
./GRADLEW CLEAN BOOTJAR && \
  JAVA -DJARMODE=TOOLS \
  -JAR BUILD/LIBS/MY-APP-1.4.JAR EXTRACT

JAVA -DSPRING.AOT.ENABLED=TRUE \
  -XX:ARCHIVECLASSESATEXIT=MY-APP.JSA
  -JAR MY-APP-1.4/MY-APP-1.4.JAR

JAVA -DSPRING.AOT.ENABLED=TRUE \
  -XX:SHAREDARCHIVEFILE=MY-APP.JSA
  -JAR MY-APP-1.4/MY-APP-1.4.JAR
```

PROJECT LEYDEN: AOT CACHE

MY-APP.AOT

```
./GRADLEW CLEAN BOOTJAR && \
  JAVA -DJARMODE=TOOLS \
  -JAR BUILD/LIBS/MY-APP-1.4.JAR EXTRACT

JAVA -DSPRING.AOT.ENABLED=TRUE \
  -XX:AOTCACHEOUTPUT=MY-APP.AOT
  -JAR MY-APP-1.4/MY-APP-1.4.JAR

JAVA -DSPRING.AOT.ENABLED=TRUE \
  -XX:AOTCACHE=MY-APP.AOT
  -JAR MY-APP-1.4/MY-APP-1.4.JAR
```

WELCHE DATEN IM CACHE?

JAVA-COMPILER

FRAMEWORK

QUELL-CODE

BYTE-CODE

JAR

IMAGE

BUILD TIME

JVM

JIT-COMPILER

BYTE-CODE

KLASSEN-LISTE

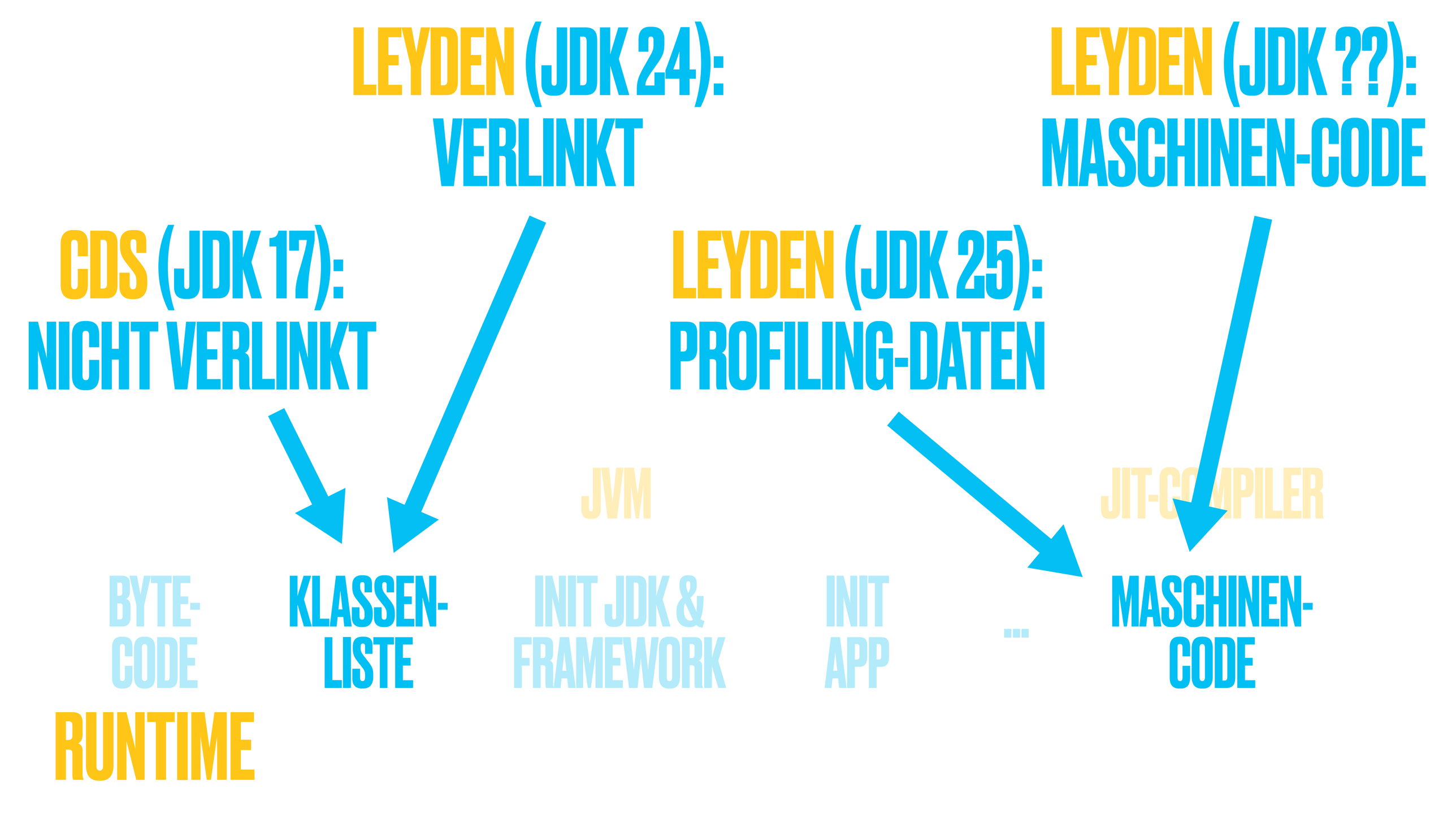
INIT JDK & FRAMEWORK

INIT APP

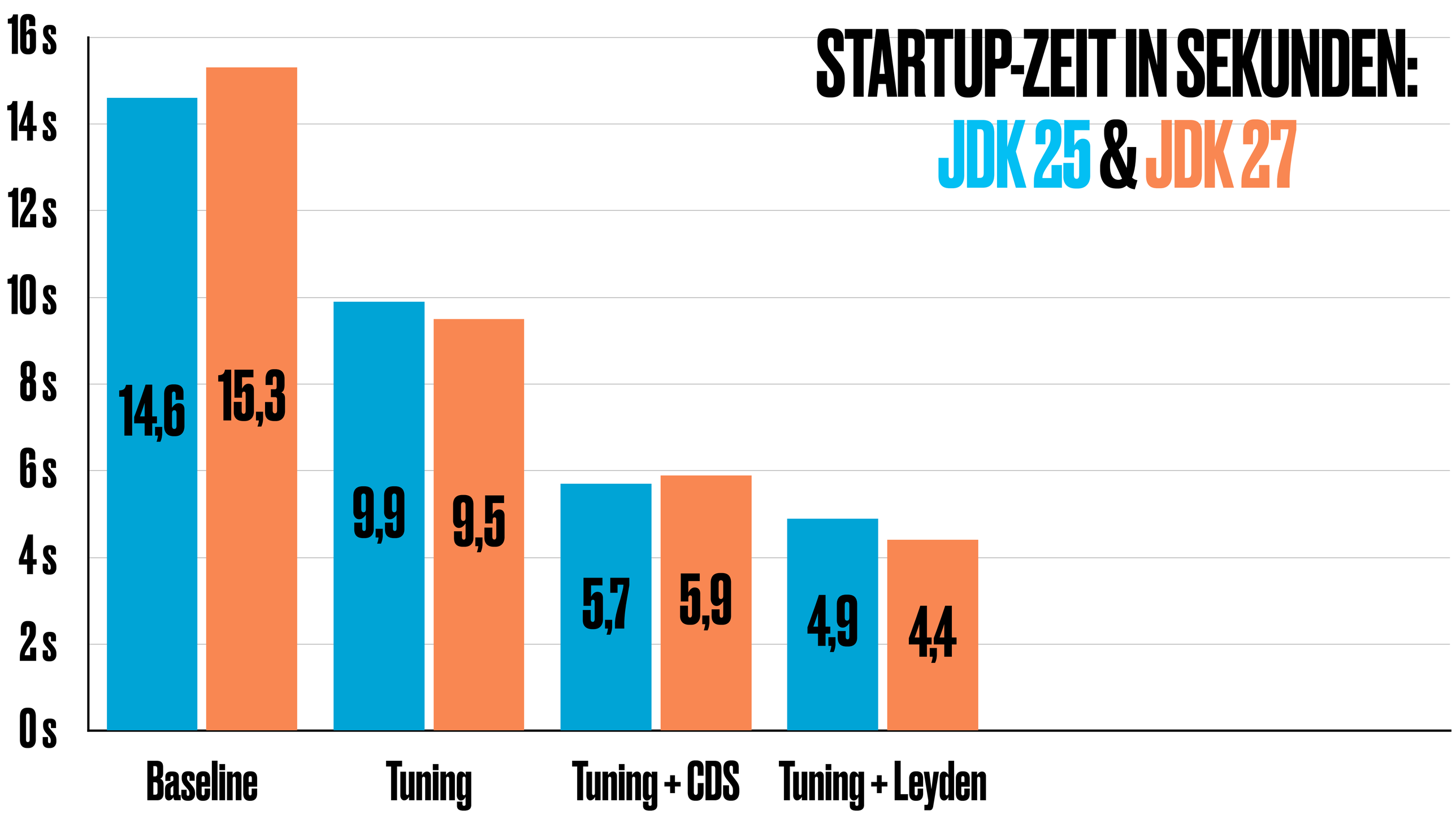
...

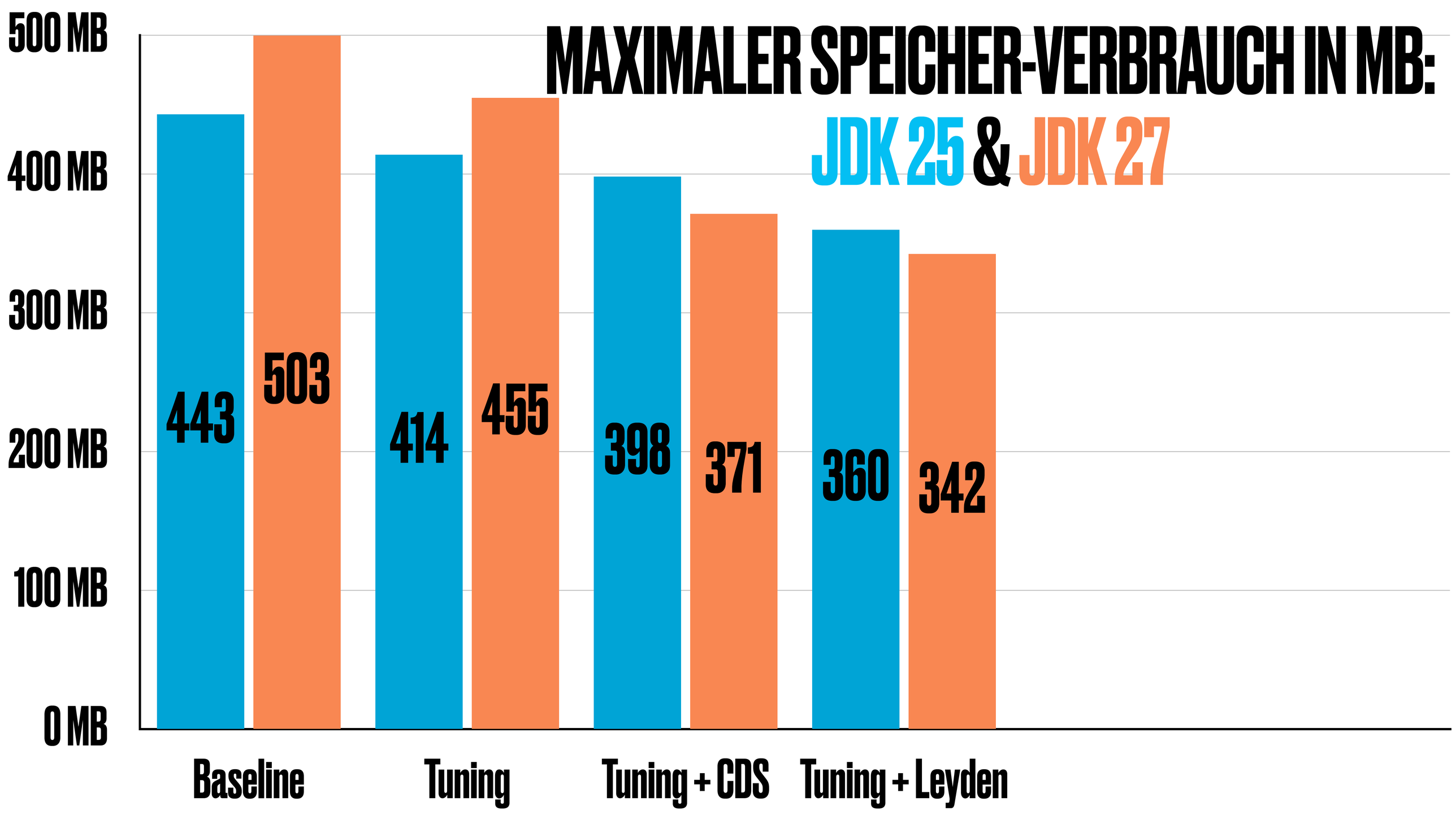
MASCHINEN-CODE

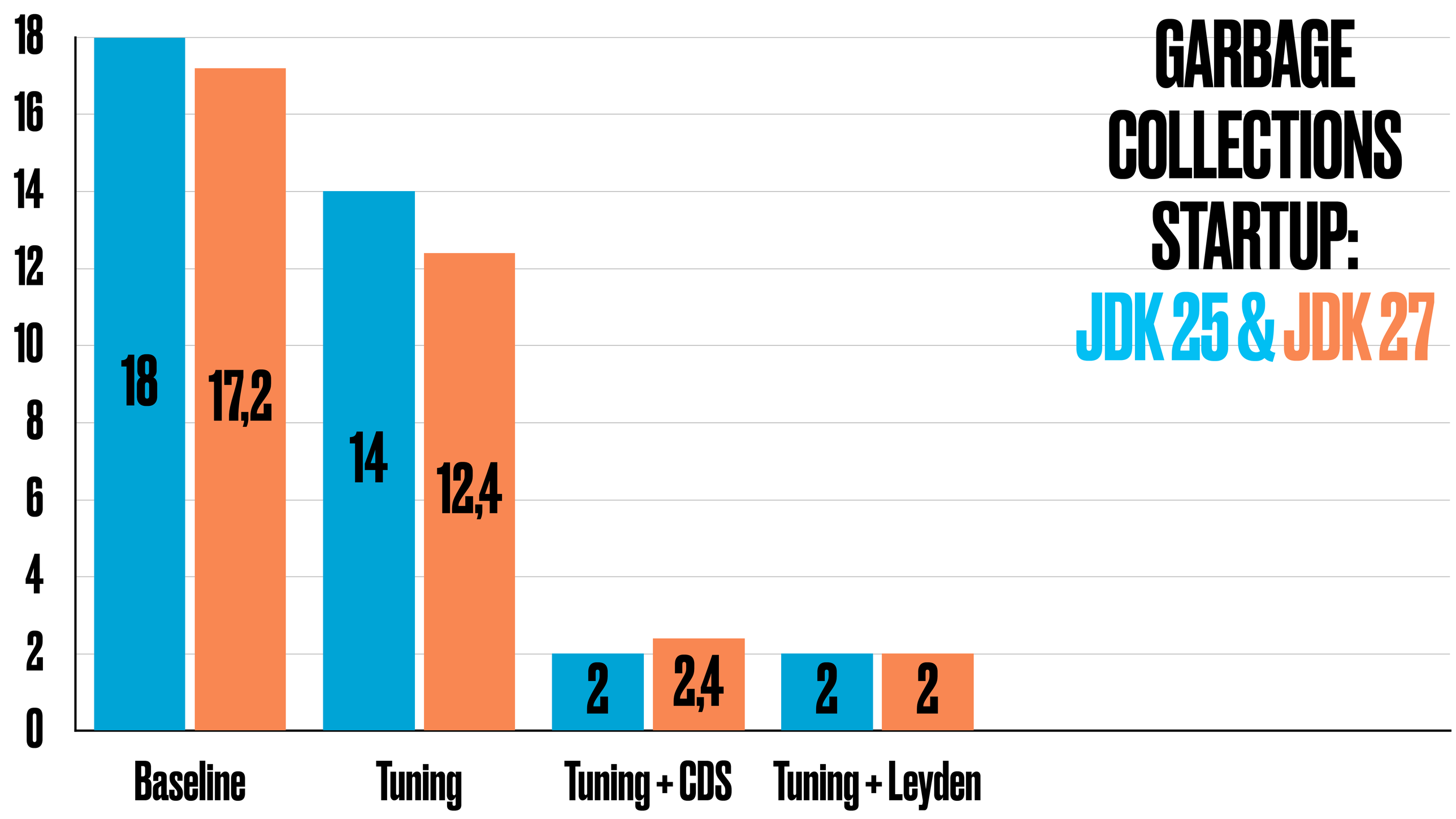
RUNTIME


LEYDEN (JDK 24):
VERLINKT
LEYDEN (JDK ??):
MASCHINEN-CODE
CDS (JDK 17):
NICHT VERLINKT
LEYDEN (JDK 25):
PROFILING-DATEN
JVM
JIT-COMPILER
BYTE-CODE
KLASSEN-LISTE
INIT JDK & FRAMEWORK
INIT APP
...
MASCHINEN-CODE
RUNTIME

STARTUP-ZEIT IN SEKUNDEN:
JDK 25 & JDK 27

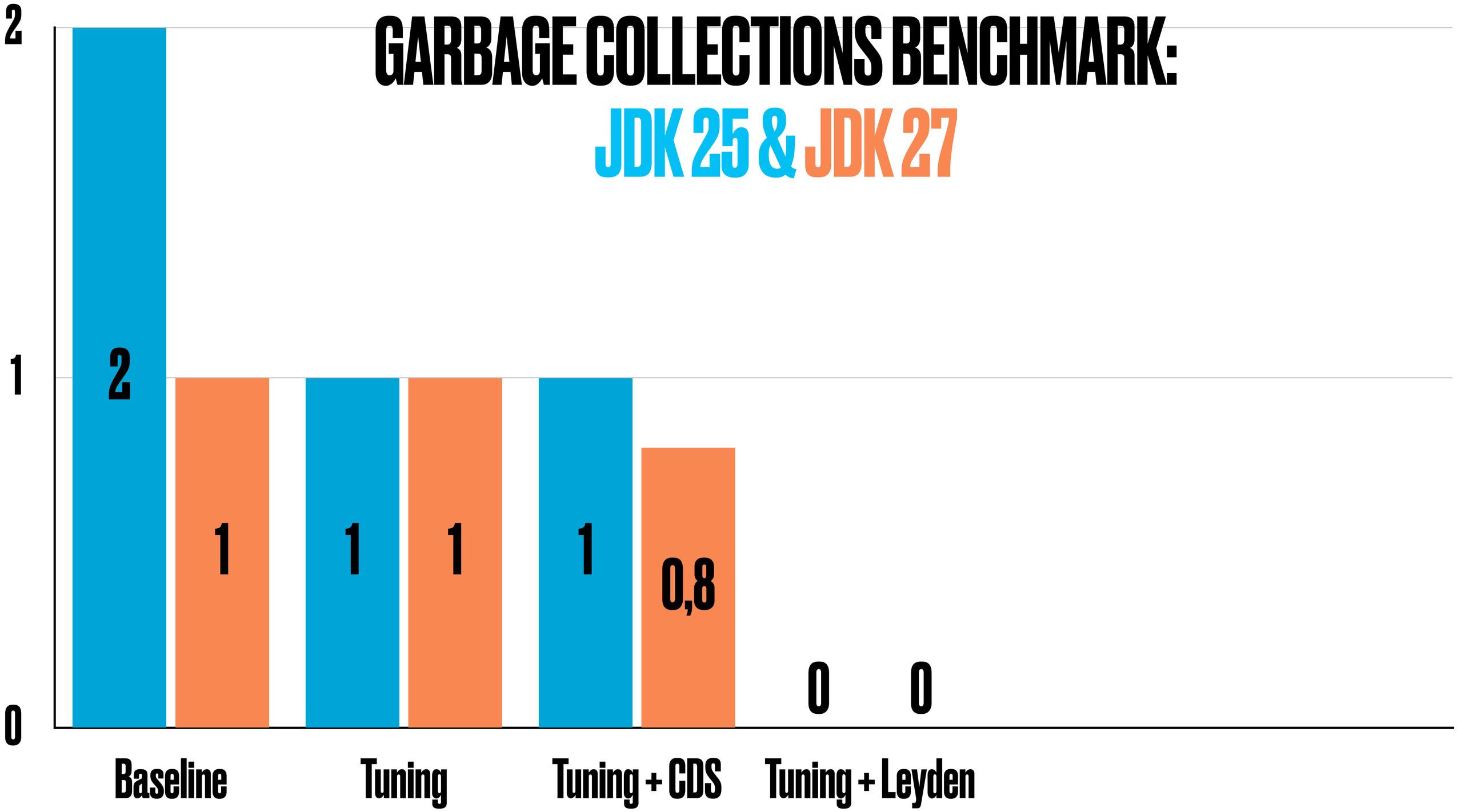



MAXIMALER SPEICHER-VERBRAUCH IN MB:
JDK 25 & JDK 27
500 MB
400 MB
300 MB
200 MB
100 MB
0 MB
443
503
414
455
398
371
360
342
Baseline
Tuning
Tuning + CDS
Tuning + Leyden


GARBAGE COLLECTIONS STARTUP:
JDK 25 & JDK 27
18
16
14
12
10
8
6
4
2
0
18
17,2
14
12,4
2
2,4
2
2
Baseline
Tuning
Tuning + CDS
Tuning + Leyden

GARBAGE COLLECTIONS BENCHMARK:

JDK 25 & JDK 27



FÜR JDK 25/27:

CDS CACHE: 106/105 MB

LEYDEN CACHE: 134 / 143 MB

PROBLEME?

PRODUKTIONSNAHER TRAININGSLAUF

CACHE-DATEI WÄHREND DES BUILDS & IN CONTAINER-IMAGE (CLOUD NATIVE BUILDPACKS IN SPRING BOOT)...

...ODER NACH BUILD & IN PRODUKTION IN CONTAINER GEMAPPT

CACHE-DATEI IN JEDEM BUILD & FÜR JEDES BETRIEBSSYSTEM BAUEN, WEIL SONST UNGÜLTIG

LEYDEN CACHE-BERECHNUNG BRAUCHT SPEICHER

LEYDEN FUNKTIONIERT NICHT MIT USER-DEFINED CLASS LOADER (Z.B. QUARKUS)

LEYDEN WIRD BESSER WERDERN

	TUNING	CDS & LEYDEN	CRAC	GRAALVM
ZEIT: SERVERLESS, STANDBY, VERFÜG-BARKEIT				
SPEICHER: WENIGER SERVER				
KOSTEN				
DEV EXPERIENCE				

3:

PROJECT CRAC

"COORDINATED RESTORE AT CHECKPOINT"

OPEN-JDK-PROJEKT, INITIAL VON AZUL

CACHET ERGEBNISSE

REDUZIERT STARTUP TIME & TIME TO PEAK

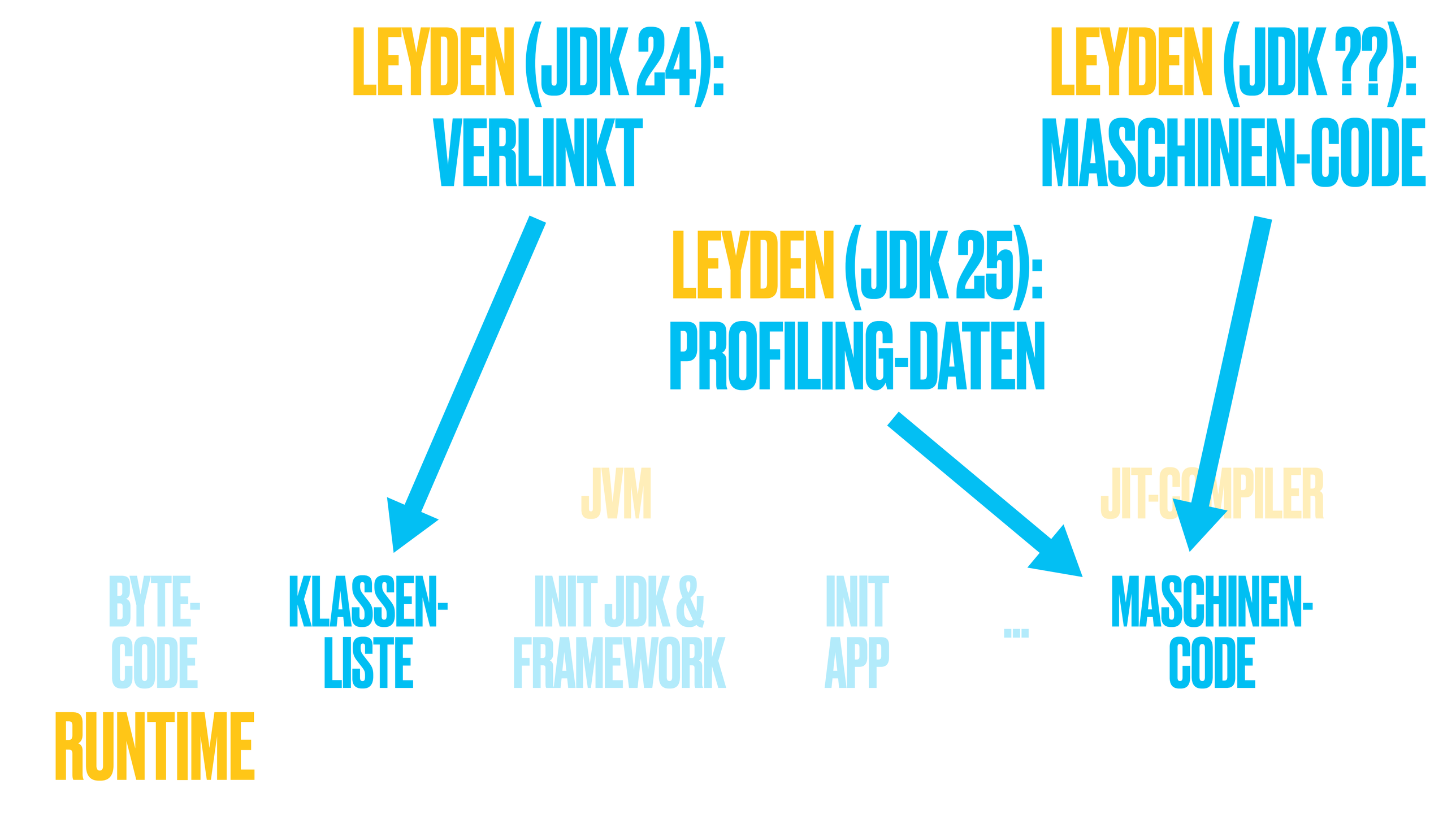
JDK 17 (TEST: 25)

LEYDEN VS. CRAC

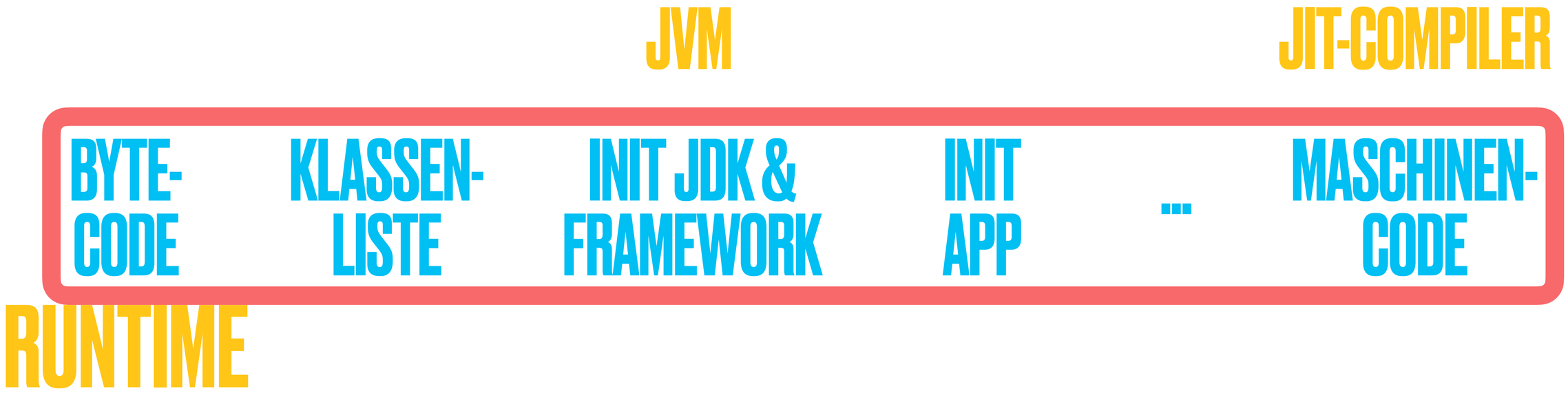
LEYDEN (JDK 24): VERLINKT

LEYDEN (JDK ??): MASCHINEN-CODE

LEYDEN (JDK 25): PROFILING-DATEN



KOMPLETTEN APPLIKATION-SPEICHER ALS MEMORY-SNAPSHOT SPEICHERN & SPÄTER LADEN



TRAININGSLAUF:



NUTZER/APPLIKATION LÖSEN
CHECKPOINT IM TRAININGSLAUF AUS:
ERZEUGT SNAPSHOT ALS MEMORY DUMP
VON ALLEN DATEN DER ANWENDUNG,
BIBLIOTHEKEN, JVM & HOTSPOT

PRODUKTION:

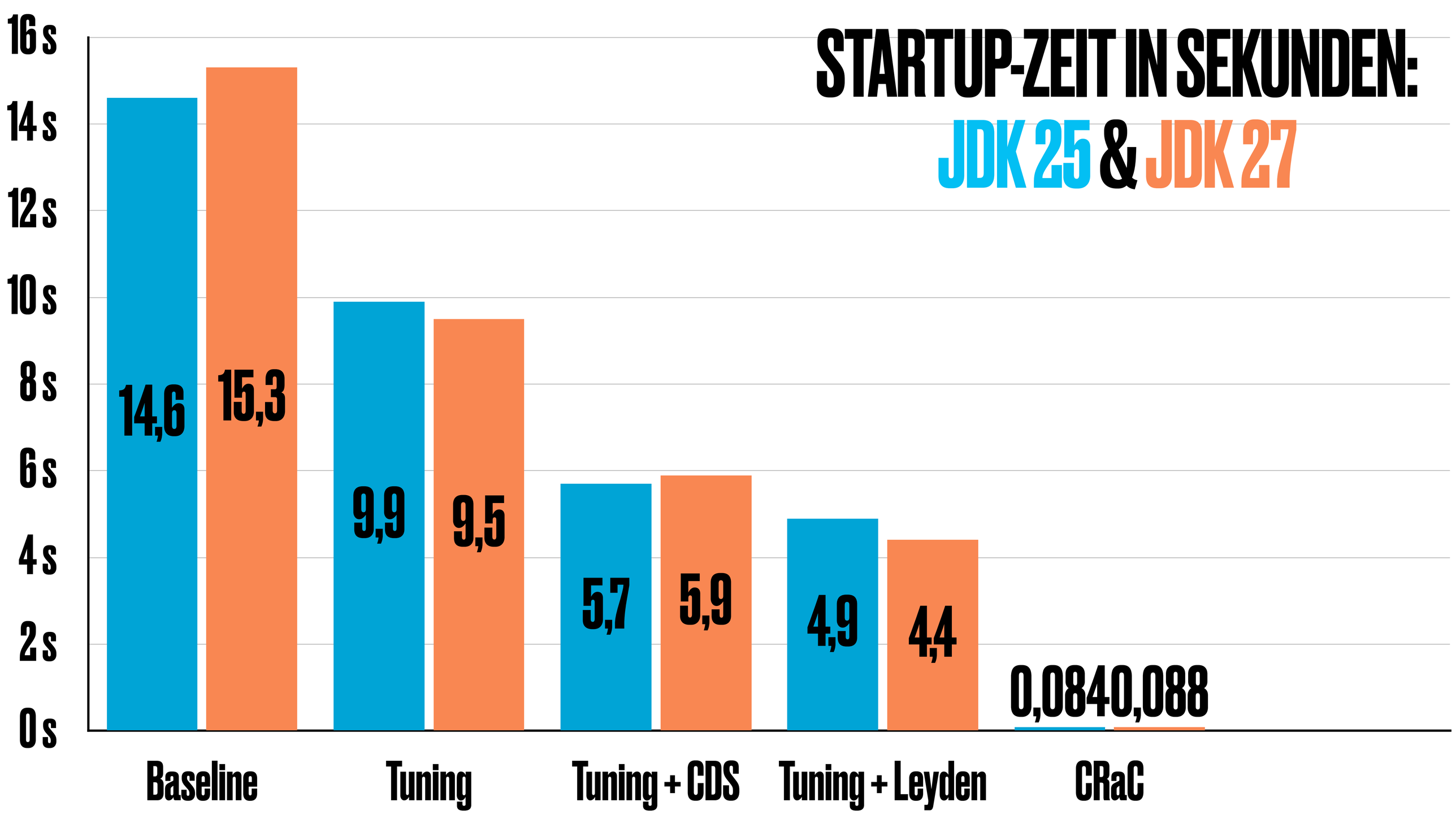


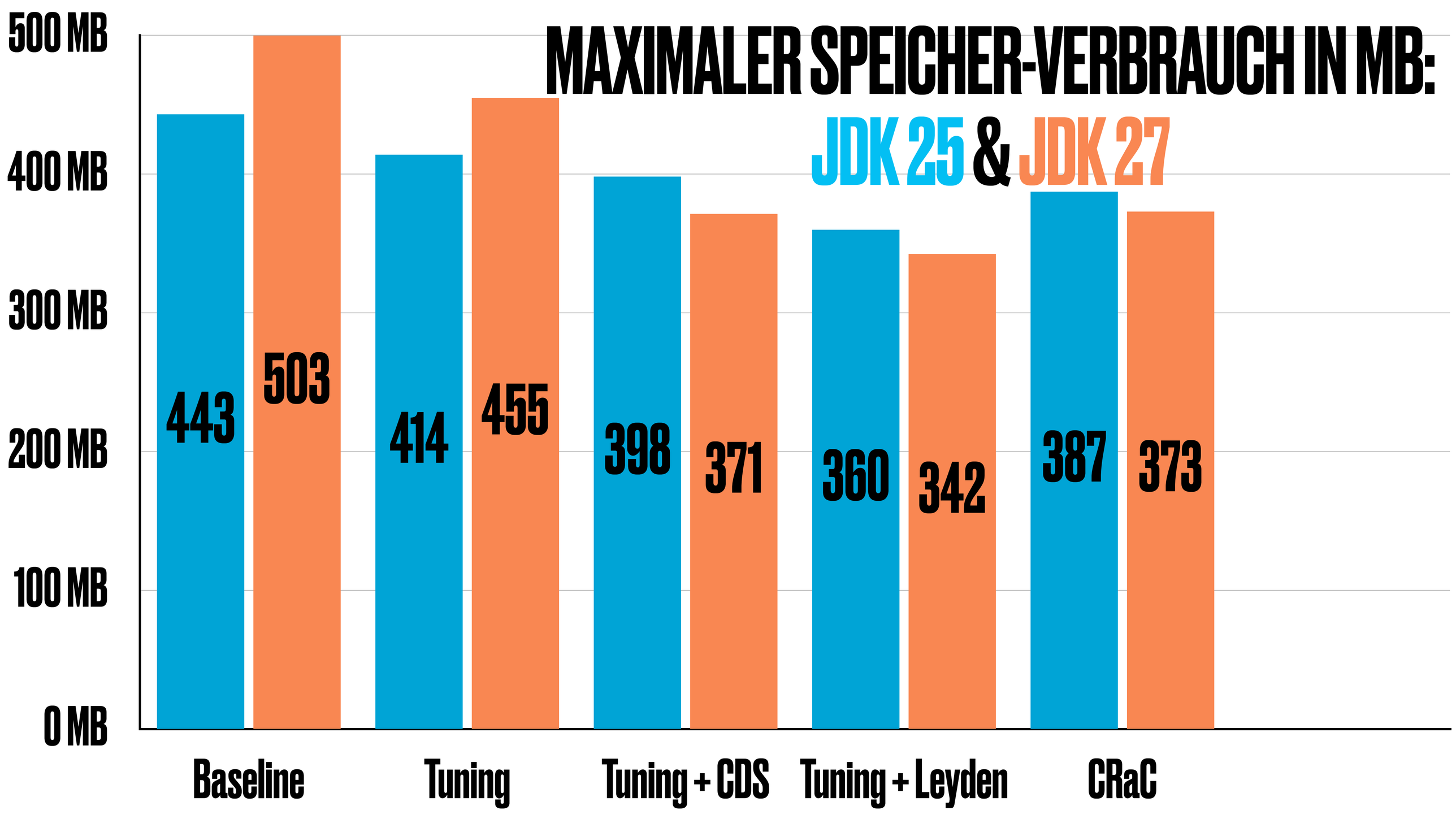
```
./GRADLEW CLEAN BOOTJAR -PCRAC=TRUE

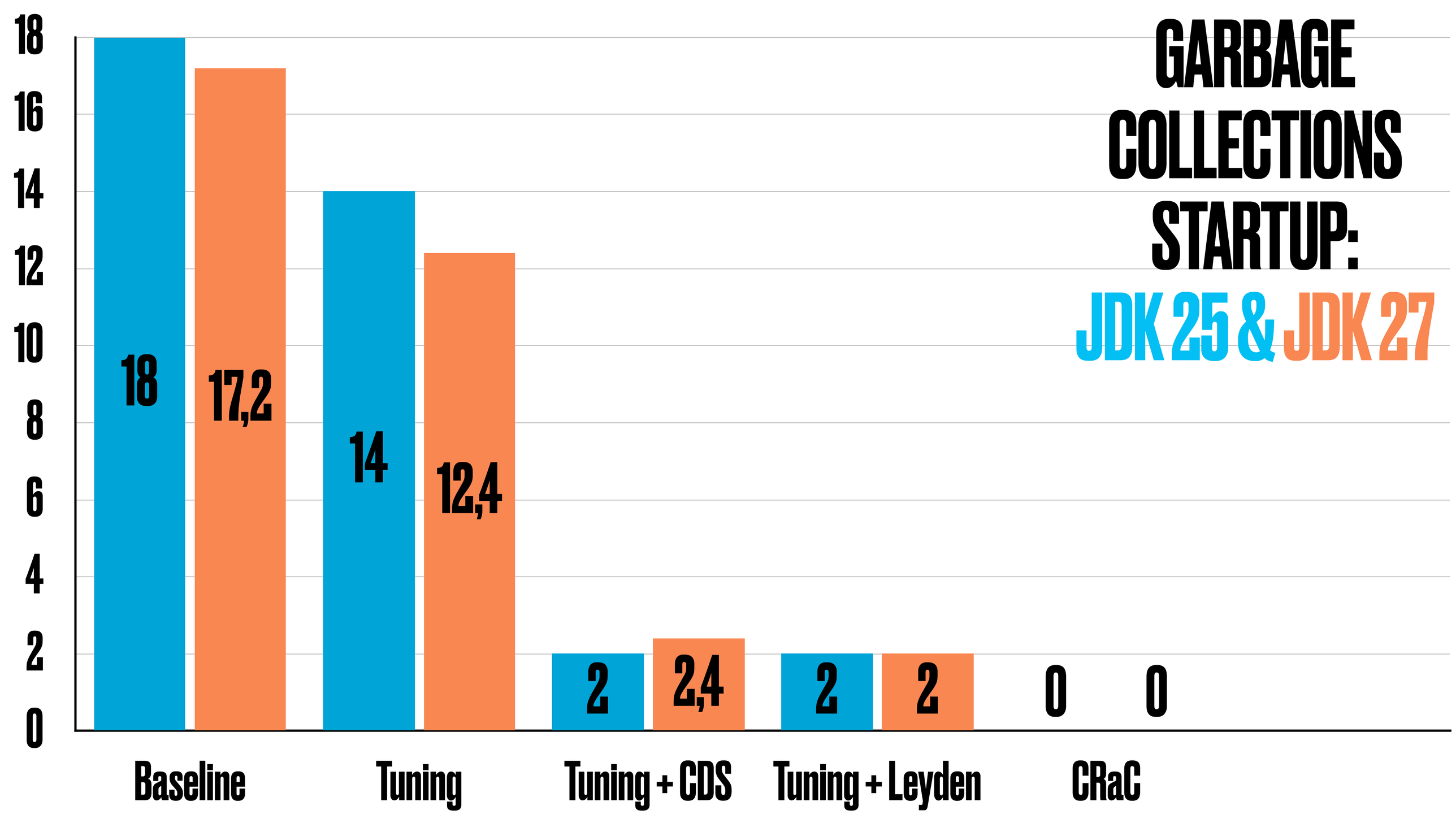
JAVA -XX:CRACENGINE=WARP \
  -XX:CRACCHECKPOINTTO=MY-APP-CRAC
  -JAR BUILD/LIBS/MY-APP-1.4.JAR

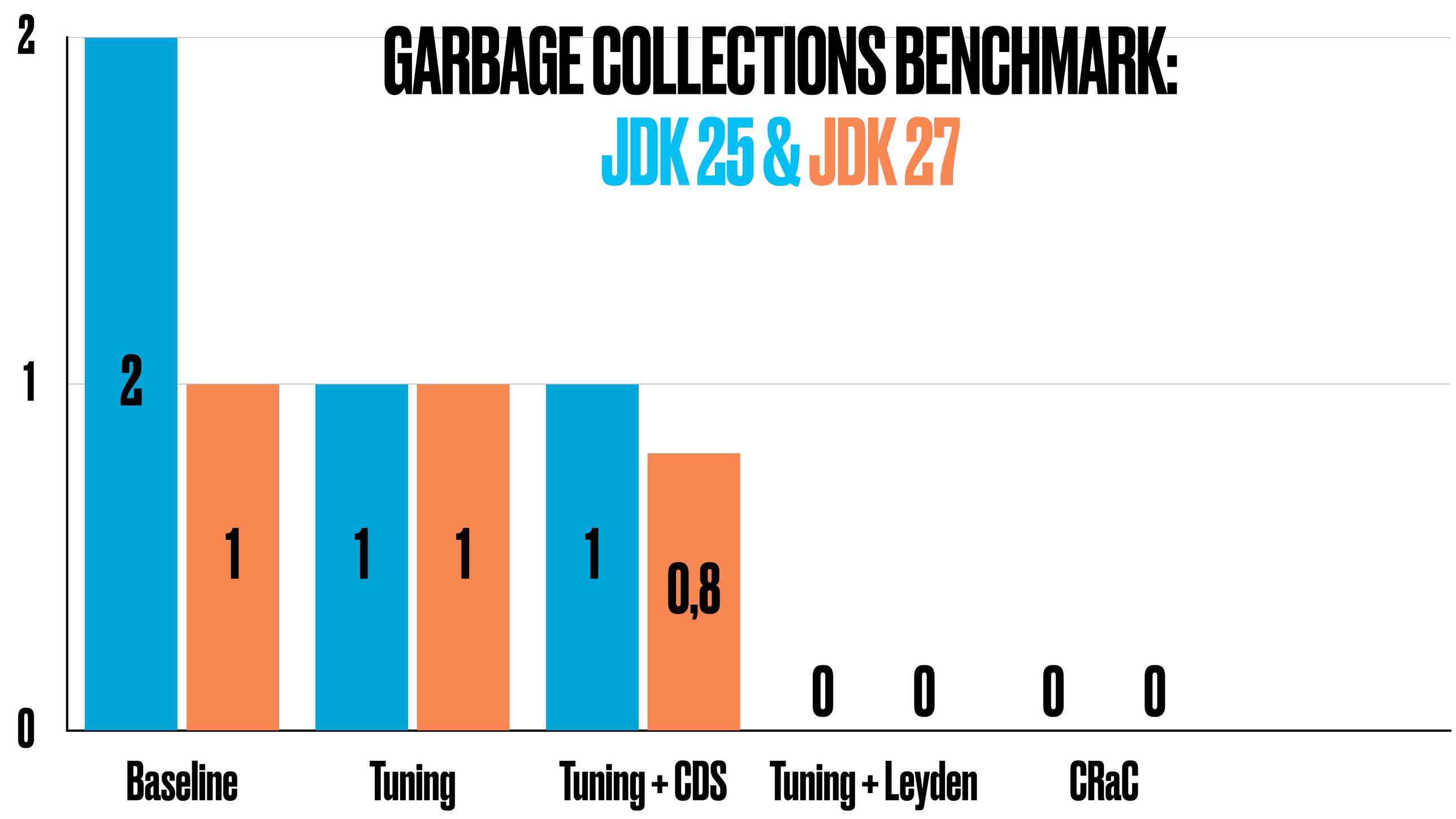
JCMD [PID] JDK.CHECKPOINT

JAVA -XX:CRACRESTOREFROM=MY-APP-CRAC
```


STARTUP-ZEIT IN SEKUNDEN:
JDK 25 & JDK 27
16 s
14 s
12 s
10 s
8 s
6 s
4 s
2 s
0 s
14,6
15,3
9,9
9,5
5,7
5,9
4,9
4,4
0,084
0,088
Baseline
Tuning
Tuning + CDS
Tuning + Leyden
CRaC


MAXIMALER SPEICHER-VERBRAUCH IN MB:
JDK 25 & JDK 27
500 MB
400 MB
300 MB
200 MB
100 MB
0 MB
443
503
414
455
398
371
360
342
387
373
Baseline
Tuning
Tuning + CDS
Tuning + Leyden
CRaC


GARBAGE COLLECTIONS STARTUP:
JDK 25 & JDK 27
18
16
14
12
10
8
6
4
2
0
18
17,2
14
12,4
2
2,4
2
2
0
0
Baseline
Tuning
Tuning + CDS
Tuning + Leyden
CRaC


GARBAGE COLLECTIONS BENCHMARK:
JDK 25 & JDK 27
2
1
0
2
1
1
1
1
0,8
0
0
0
0
Baseline
Tuning
Tuning + CDS
Tuning + Leyden
CRaC

FÜR JDK 25/27:

SNAPSHOT: 229/381 MB

PROBLEME?

"CRAC TRIES TO MAKE THINGS RUN FASTER POTENTIALLY AT THE EXPENSE OF CORRECT-NESS, AND I DON'T LIKE THAT TRADE-OFF."

BRIAN GOETZ, JAVA LANGUAGE ARCHITECT BEI ORACLE

VOR SNAPSHOT-SPEICHERN ALLE DATEIEN & PORTS SCHLIEßEN...

...UND NACH SNAPSHOT-LADEN WIEDER ÖFFNEN

CRAC BRAUCHT SUPPORT VON JDK, FRAMEWORK, BIBLIOTHEKEN & ANWENDUNG

JDK: AZUL, BELLSOFT LIBERICA, CANONICAL/UBUNTU

FRAMEWORKS: SPRING BOOT 3.2+, QUARKUS 2.10.0, MICRONAUT 3.7, HELIDON 4.2

CRAC FUNKTIONIERT NUR IN LINUX – NOOP-IMPLEMENTIERUNG IN WINDOWS & MACOS

BRAUCHT CRAC JAVA-BIBLIOTHEK

SICHERHEITSRISIKO SNAPSHOT: PASSWÖRTER, SECRETS, BYTECODE & MASCHINENCODE IM KLARTEXT

VERSCHLÜSSELUNG FÜR SNAPSHOTS GEPLANT

	TUNING	CDS & LEYDEN	CRAC	GRAALVM
ZEIT: SERVERLESS, STANDBY, VERFÜG-BARKEIT				
SPEICHER: WENIGER SERVER				
KOSTEN				
DEV EXPERIENCE				

4:
GRAALVM NATIVE IMAGE

PROJEKT VON ORACLE LABS

VERLAGERT SCHRITTE ZUM BUILD

REDUZIERT STARTUP TIME & TIME TO PEAK

SPRING BOOT 3,
GRAALVM JDK 17 (HIER: 25)

"JAVA MINUS
JIT-COMPILER"

ORACLE LABS !=
ORACLE JAVA-TEAM

AHEAD-OF-TIME-COMPILER (AOT)

“NATIVE IMAGE” BAUT AUSFÜHRBARE PROGRAMME FÜR LINUX, WINDOWS & MACOS

BENÖTIGT SPRING AOT

NATIVE IMAGE ENTFERNT BEIM BUILD ALLE NICHT BENÖTIGTEN KLASSEN & RESSOURCEN

PROFILE-GUIDED OPTIMIZATION (PGO): TRAININGSLAUF MIT INSTRUMENTIERTER ANWENDUNG, DIE PROFILING-DATEN FÜR ZWEITEN COMPILIERUNGS-LAUF SAMMELT

MEHR SICHERHEIT: KEIN BÖSARTIGER BYTECODE LADBAR ZUR LAUFZEIT, SICHERHEITSLÜCKEN IN UNBENUTZTEM CODE ENTFERNT

ANWENDUNGEN KLEINER ALS JAR + JRE

ZUR LAUFZEIT IMMER NOCH JVM ("SUBSTRATE VM") FÜR THREADS & GARBAGE COLLECTION

GRAALVM BRAUCHT SUPPORT VON FRAMEWORK, BIBLIOTHEKEN & ANWENDUNG

FRAMEWORKS: SPRING BOOT 3.0+, QUARKUS, MICRONAUT, HELIDON

BIBLIOTHEKEN: ÖFFENTLICHES KONFIGURATIONS-REPOSITORY

GRAALVM JDK = OPENJDK + NATIVE IMAGE
+ GRAAL JIT-COMPILER
(HOTSPOT-ERSATZ)

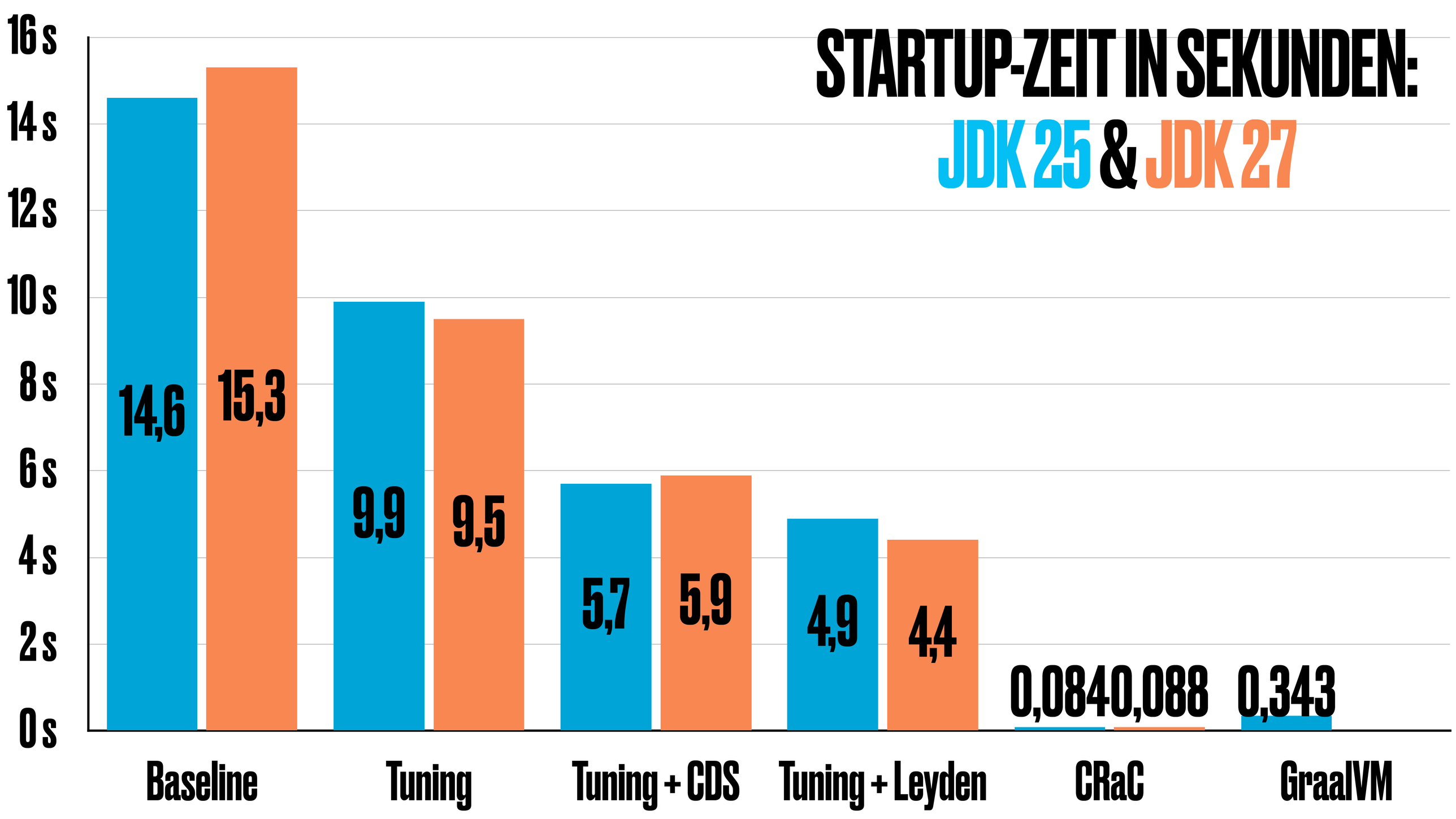
SEPARATER DOWNLOAD, ABER RELEASES
SYNCHRON MIT OPENJDK

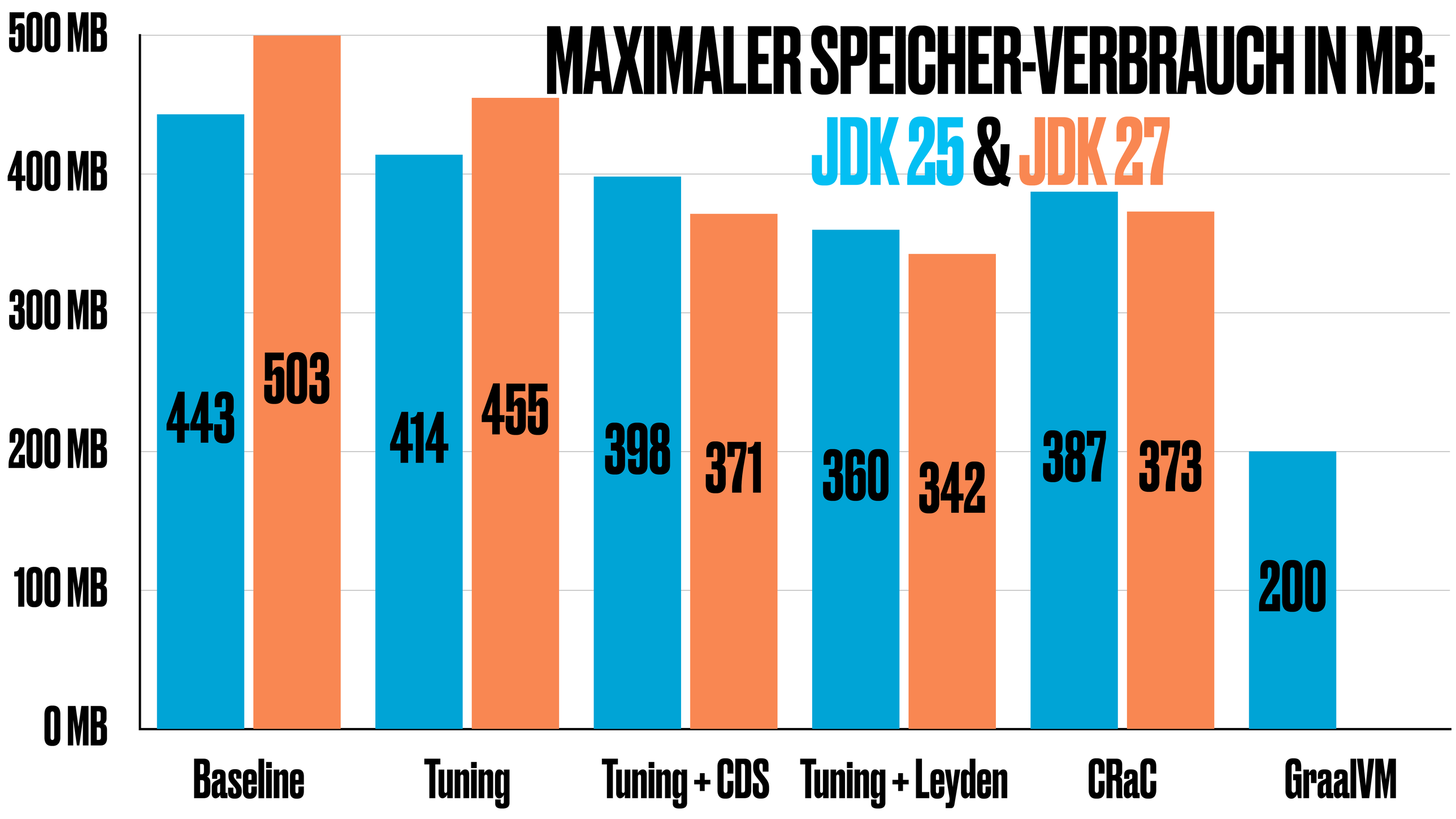
```
./GRADLEW NATIVECOMPILE --PGO-INSTRUMENT

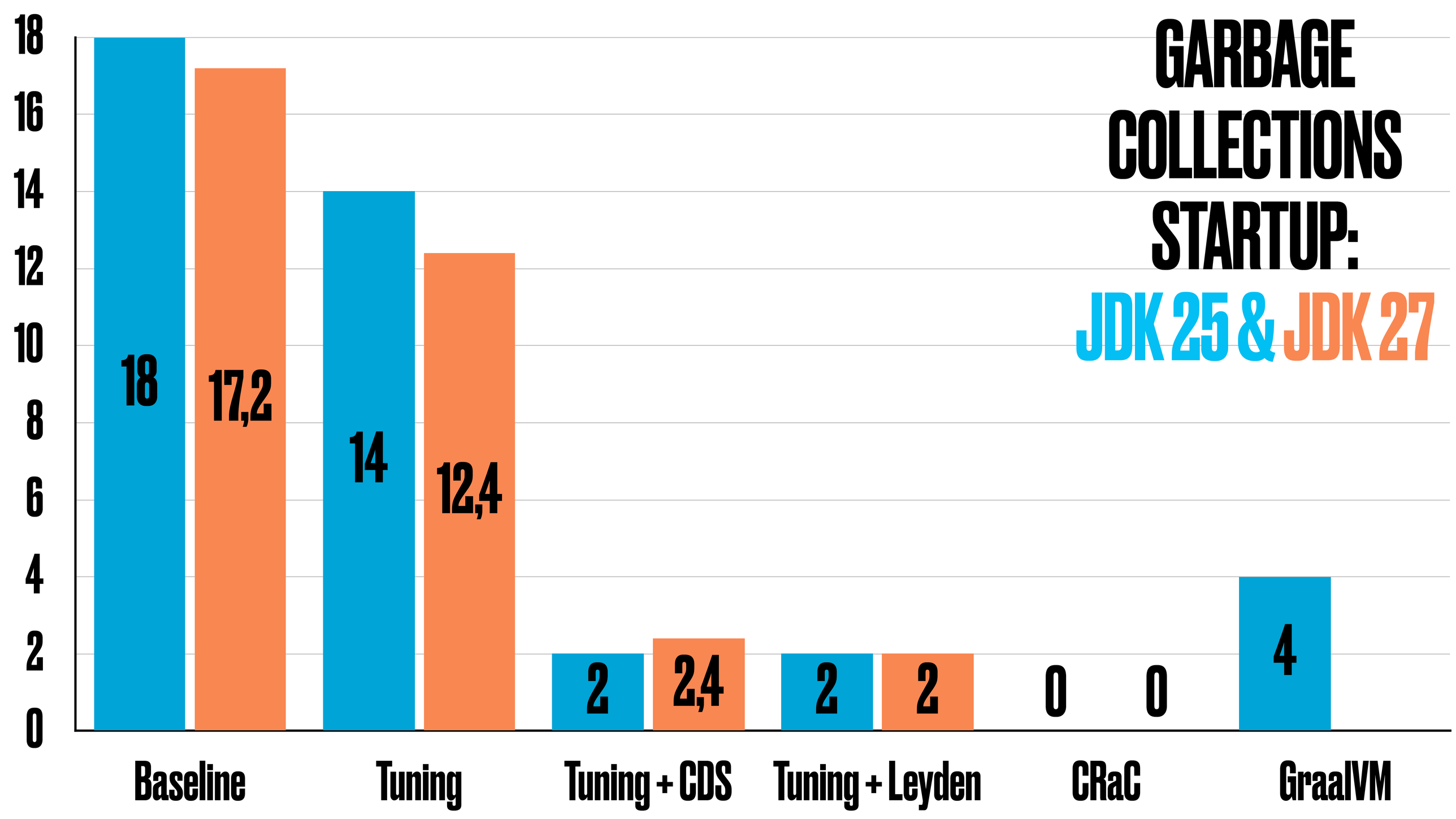
./BUILD/NATIVE/NATIVECOMPILE/MY-APP-
INSTRUMENTED

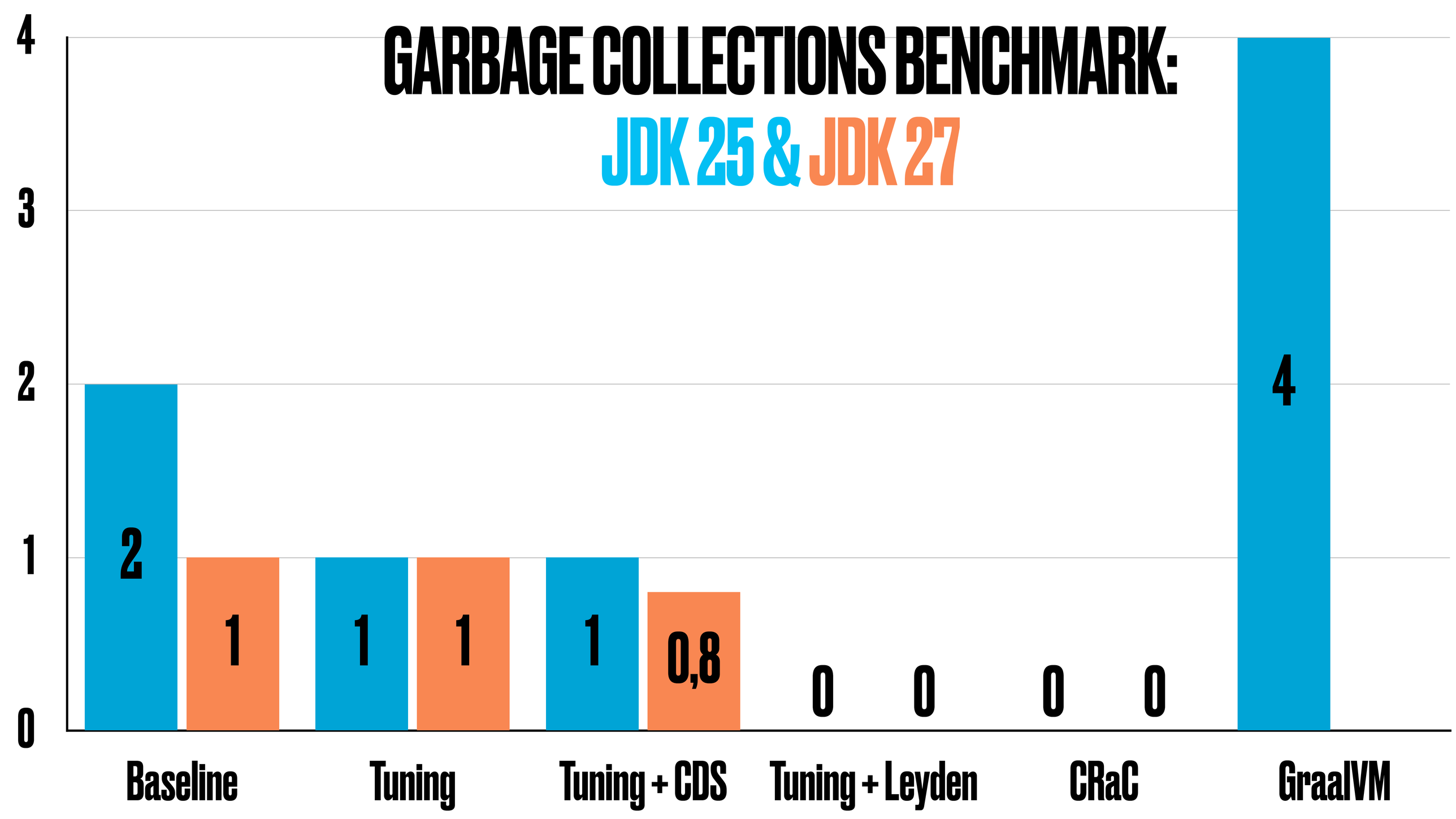
./GRADLEW NATIVECOMPILE

./BUILD/NATIVE/NATIVECOMPILE/MY-APP
```


STARTUP-ZEIT IN SEKUNDEN:
JDK 25 & JDK 27
16 s
14 s
12 s
10 s
8 s
6 s
4 s
2 s
0 s
14,6
15,3
9,9
9,5
5,7
5,9
4,9
4,4
0,084
0,088
0,343
Baseline
Tuning
Tuning + CDS
Tuning + Leyden
CRaC
GraalVM


MAXIMALER SPEICHER-VERBRAUCH IN MB:
JDK 25 & JDK 27
500 MB
400 MB
300 MB
200 MB
100 MB
0 MB
443
503
414
455
398
371
360
342
387
373
200
Baseline
Tuning
Tuning + CDS
Tuning + Leyden
CRaC
GraalVM


GARBAGE COLLECTIONS STARTUP:
JDK 25 & JDK 27
18
16
14
12
10
8
6
4
2
0
18
17,2
14
12,4
2
2,4
2
2
0
0
4
Baseline
Tuning
Tuning + CDS
Tuning + Leyden
CRaC
GraalVM


GARBAGE COLLECTIONS BENCHMARK:
JDK 25 & JDK 27
4
3
2
1
0
2
1
1
1
1
0,8
0
0
0
0
4
Baseline
Tuning
Tuning + CDS
Tuning + Leyden
CRaC
GraalVM

FÜR JDK 25:

LINUX APP: 185 MB

LINUX APP: 172 MB

NATIVE IMAGE:

12:06 MIN & 7:41 MIN

MIT 32 GB RAM & 8 CPUS

PROBLEME?

EINIGE JAVA-FEATURES UNMÖGLICH

ANDERE FUNKTIONIEREN NUR TEILWEISE

REFLECTION & EXPLIZITES KLASSEN-LADEN BRAUCHEN KONFIGURATION

GRAALVM KANN KONFIGURATION AUTOMATISCH DURCH "MITHÖREN BEI JIT-JAVA" ERZEUGEN

NERVIG

LANGE BUILD-ZEITEN & VIEL SPEICHER

EINE ANWENDUNG PRO BETRIEBSSYSTEM

NATIVES DEBUGGEN AUF MAC/WINDOWS

EXTREM ZÄH, WEIL ANWENDUNG IN LINUX-CONTAINER LÄUFT

	TUNING	CDS & LEYDEN	CRAC	GRAALVM
ZEIT: SERVERLESS, STANDBY, VERFÜG-BARKEIT				
SPEICHER: WENIGER SERVER				
KOSTEN				
DEV EXPERIENCE				

ZEIT SPAREN: CRAC & GRAALVM

SPEICHER SPAREN: GRAALVM

SO SCHNELLER!

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

ZUSAMMENFASSUNG

WANN KANN ES SICH LOHNEN, JAVA SCHNELLER ZU STARTEN?

SERVERLESS

VIELE SERVER

SEHR HOHE VERFÜGBARKEIT

1. SPRING BOOT TUNING

2. CLASS DATA SHARING (CDS) & OPENJDK PROJECT LEYDEN

3. OPENJDK PROJECT CRAC

4. GRAALVM NATIVE IMAGE


The End

MEIN VORTRAG HÖRT NICHT AUF, WENN ICH AUFHÖRE VORZUTRAGEN!

FOLIEN, CODE, ROI-BEISPIEL & MEHR



HTTPS://ATRA.SOFTWARE/WJ1