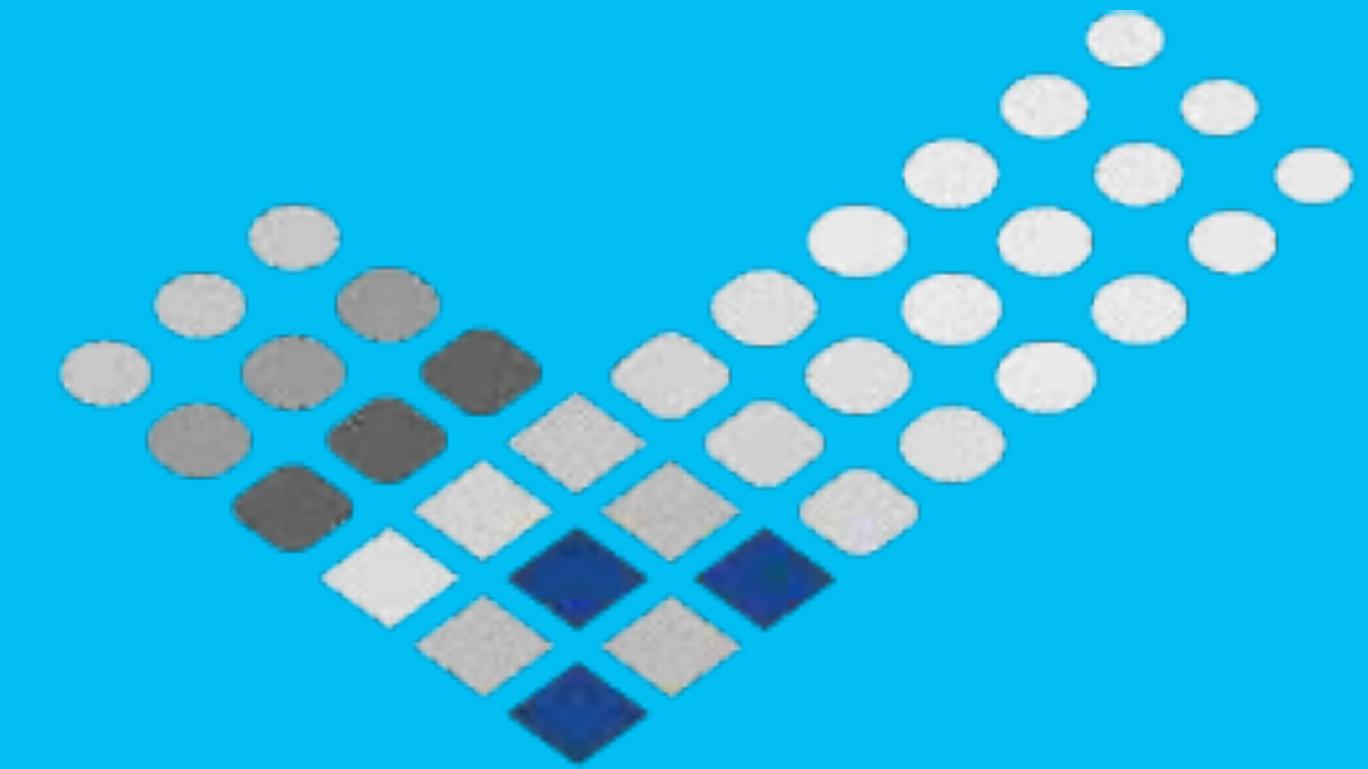


KARSTEN SILZ, 4. NOVEMBER 2025

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

JAVA BY DESIGN:

MEHR ZEIT

MEHR CPU

MEHR SPEICHER

1. SPRING BOOT TUNING

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

3. OPENJDK PROJECT CRAC

4. GRAALVM NATIVE IMAGE

**FOLIEN, CODE,
ROI-BEISPIEL &
MEHR**



HTTPS://BPELI.ATC

**WAS MACHT MICH ZUM
EXPERTEN?**

**2022: BETREUTE ARTIKEL-SERIE
ÜBER GRAALVM**

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

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

LOHNT SICH DAS?

**ENTWICKLER SOLLEN
BUSINESS VALUE
SCHAFFEN...**

NEUE FEATURES

VERBESSERUNGEN

FEHLERBEHEBUNGEN

...NICHT KOSTENEIN-
SPARUNGEN!

**FRAGEN SIE CHEF FÜR
JAVA-STARTUP-
OPTIMIERUNG
UM ERLAUBNIS!**

SERVERLESS

VIELE SERVER

SEHR HOHE VERFÜGBARKEIT

SERVERLESS

ABRECHNUNG NACH LAUFZEIT IN MS

SCHNELLER STARTUP =
GERINGERE KOSTEN

AWS LAMBDA SNAPSTART: LAMBDA MIT CRAC

VIELE SERVER

VIELE STANDBY-
INSTANZEN
FÜR LASTSPITZEN:

SCHNELLERER STARTUP =
WENIGER INSTANZEN =
WENIGER SERVER

**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 UPDATES**

**SCHNELLER STARTUP:
VERSICHERUNG FÜR
GEPLANTE ODER
UNGEPLANTE AUSFÄLLE**

BEWERTUNG DER VARIANTEN

KOSTEN

**EINMAL: BUILD & DEPLOYMENT,
ENTWICKLER-TRAINING**

**WIEDERKEHRENDE:
WARTUNG, UPGRADES**

DEV EXPERIENCE

LOKALE ENTWICKLUNG

LANGSAMER, PRODUKTIONS-

TROUBLESHOOTING SCHLECHTER

TUNING

ZEIT: SERVERLESS,
STANDBY, VERFÜG-
BARKEIT

SPEICHER:
WENIGER SERVER

KOSTEN

DEV EXPERIENCE

CDS & LEYDEN

CRAC

GRAALVM

ROI: GRAALVM ÜBER 3 JAHRE

KOSTEN (PERSONAL & HARDWARE):

56.080 €

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

**ROI-
BEISPIEL**



HTTPS://BPELIATC

**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 -
PROGRAMMERSPRACHE 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/GRAFQL + 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 COMPILERUNG => 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

**QUELL-
CODE**

BUILD TIME

FRAMEWORK

JAR

IMAGE

**BYTE-
CODE**

**KLASSEN-
LISTE**

**INIT JDK &
FRAMEWORK**

**INIT
APP**

...

JIT-COMPILER

**MASCHINEN-
CODE**

RUNTIME

JVM

**STARTUP TIME &
TIME TO PEAK
PERFORMANCE**

JAVA-COMPILER

QUELL-
CODE

BUILD TIME

BYTE-
CODE

RUNTIME

KLASSEN-
LISTE

INIT JDK &
FRAMEWORK

INIT
APP

JVM

JIT-COMPILER

MASCHINEN-
CODE

...

FRAMEWORK

JAR

IMAGE

STARTUP
TIME

JAVA-COMPILER

QUELL-
CODE

BUILD TIME

BYTE-
CODE

RUNTIME

FRAMEWORK

BYTE-
CODE

JAR

IMAGE

TIME TO PEAK
PERFORMANCE

JVM

KLASSEN-
LISTE

INIT JDK &
FRAMEWORK

INIT
APP

...

JIT-COMPILER

MASCHINEN-
CODE

JAVA BY DESIGN:
MEHR ZEIT
MEHR CPU
MEHR SPEICHER

JAVA-COMPILER

QUELL-
CODE

BUILD TIME

BYTE-
CODE

RUNTIME

FRAMEWORK

BYTE-
CODE

JAR

IMAGE

JVM

KLASSEN-
LISTE

INIT JDK &
FRAMEWORK

INIT
APP

...

MASCHINEN-
CODE

LÄUFT JEDES MAL,
GLEICHES RESULTAT:
VIELE JAVA-
OBJEKTE

JIT-COMPILER

MEHR ZEIT FÜR
STARTUP

MEHR CPU

JAVA-COMPILER

**QUELL-
CODE**

BUILD TIME

**BYTE-
CODE**

RUNTIME

FRAMEWORK

**BYTE-
CODE**

JAR

IMAGE

JVM

**KLASSEN-
LISTE**

**INIT JDK &
FRAMEWORK**

**INIT
APP**

...

**PROFILING
C1 COMPILER
PROFILING
C2 COMPILER**

JIT-COMPILER

**MASCHINEN-
CODE**

MEHR ZEIT FÜR
TIME TO PEAK

MEHR CPU

JAVA-COMPILER

**QUELL-
CODE**

BUILD TIME

FRAMEWORK

**BYTE-
CODE**

JAR

IMAGE

**PROFILER +
2 COMPILER
IN APP!**

JIT-COMPILER

**BYTE-
CODE**

RUNTIME

**KLASSEN-
LISTE**

**INIT JDK &
FRAMEWORK**

**INIT
APP**

...

**MASCHINEN-
CODE**

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

BY DESIGN!

JAVA BY DESIGN:

MEHR ZEIT

MEHR CPU

MEHR SPEICHER

DARUM LANGSAM!

LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

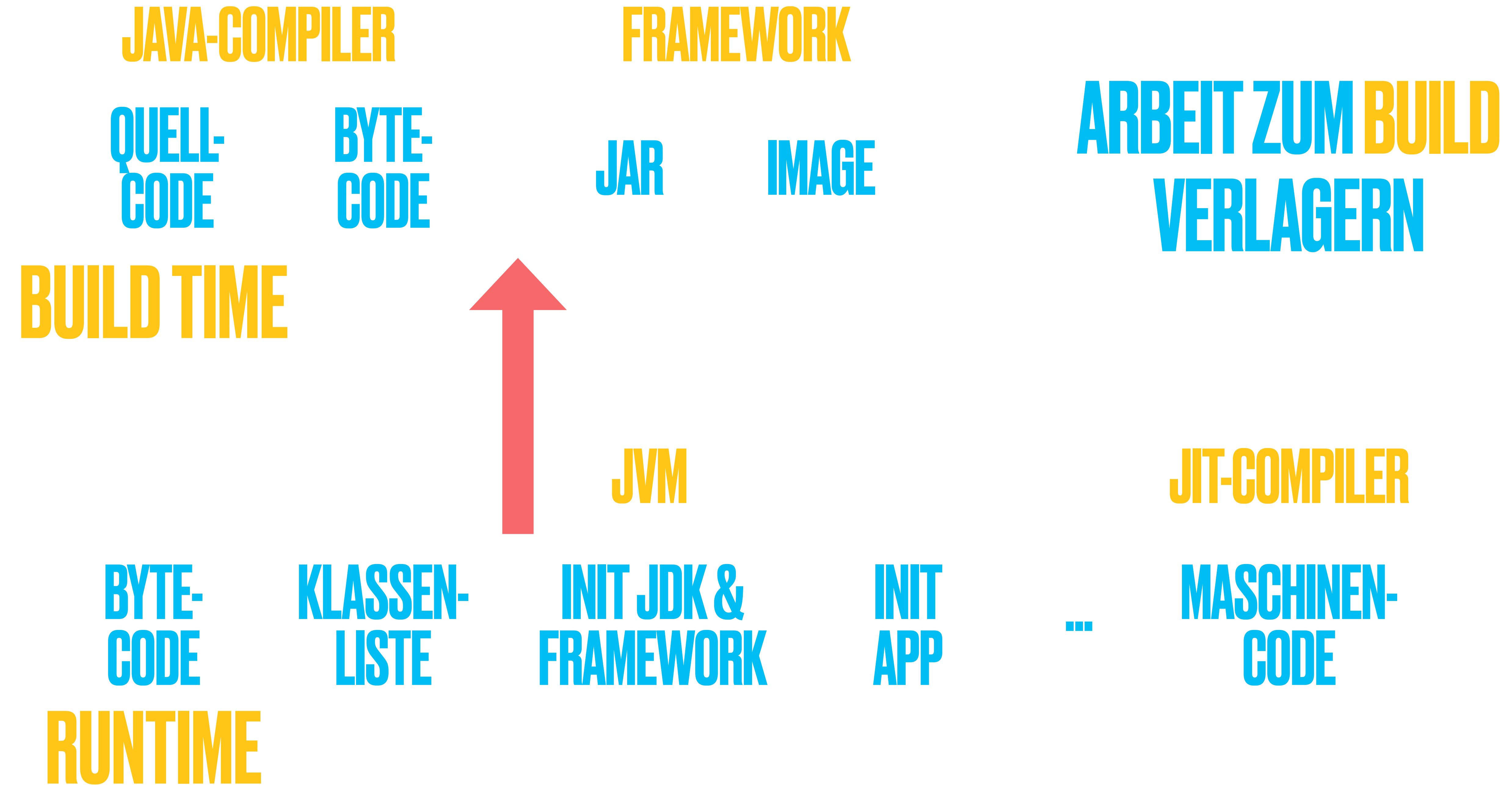
LOHNT SICH DAS?

WARUM JETZT?

WARUM LANGSAM?

WIE SCHNELLER?

WIE SCHNELLER?



JAVA-COMPILER

QUELL-
CODE

BUILD TIME

FRAMEWORK

BYTE-
CODE

JAR

IMAGE

ERGEBNISSE
CACHEN

JIT-COMPILER

BYTE-
CODE

RUNTIME

JVM

KLASSEN-
LISTE

INIT JDK &
FRAMEWORK

INIT
APP

...

MASCHINEN-
CODE

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 4.0 RC1

JAVA 25

MESSEN STARTUP TIME

DANN 5X DIE GLEICHEN
11SEITEN 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]**

CODE

[HTTPS://BETA.LIATC](https://beta.liatc)



0.

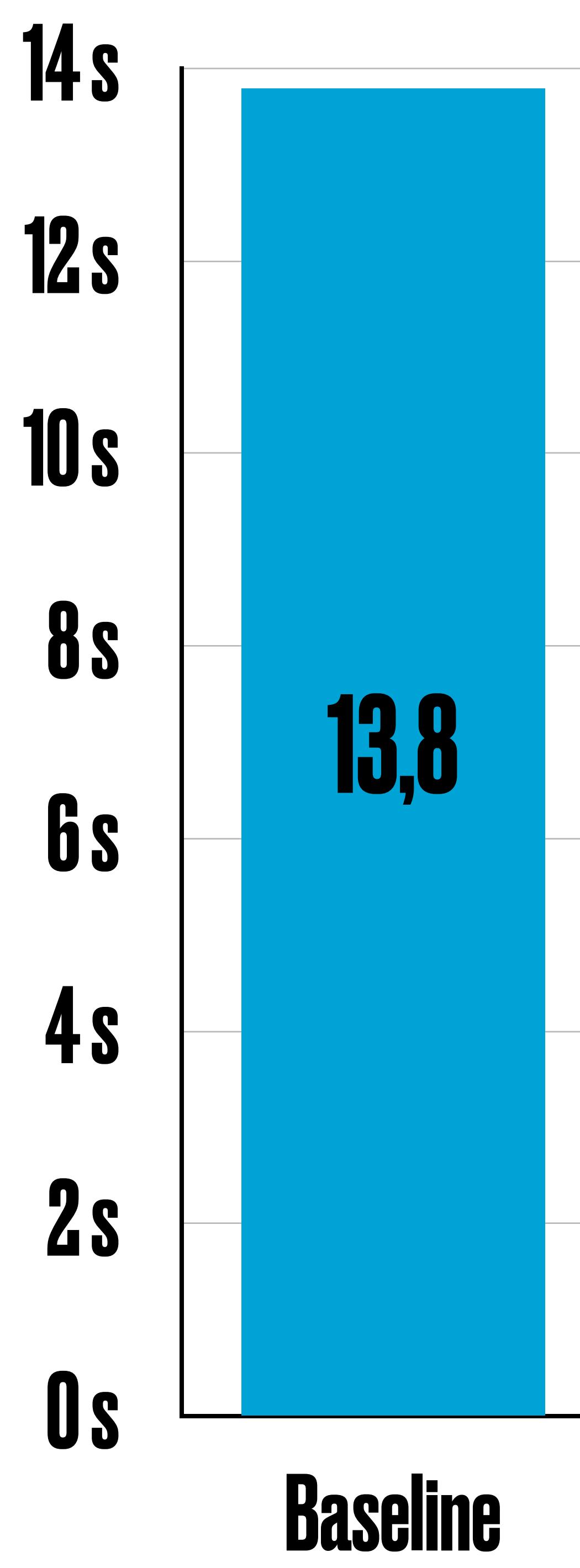
PERFORMANCE BASELINE

SPRING BOOT 3

JDK 17

./GRADLEW CLEAN BOOTJAR

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



500 MB

400 MB

300 MB

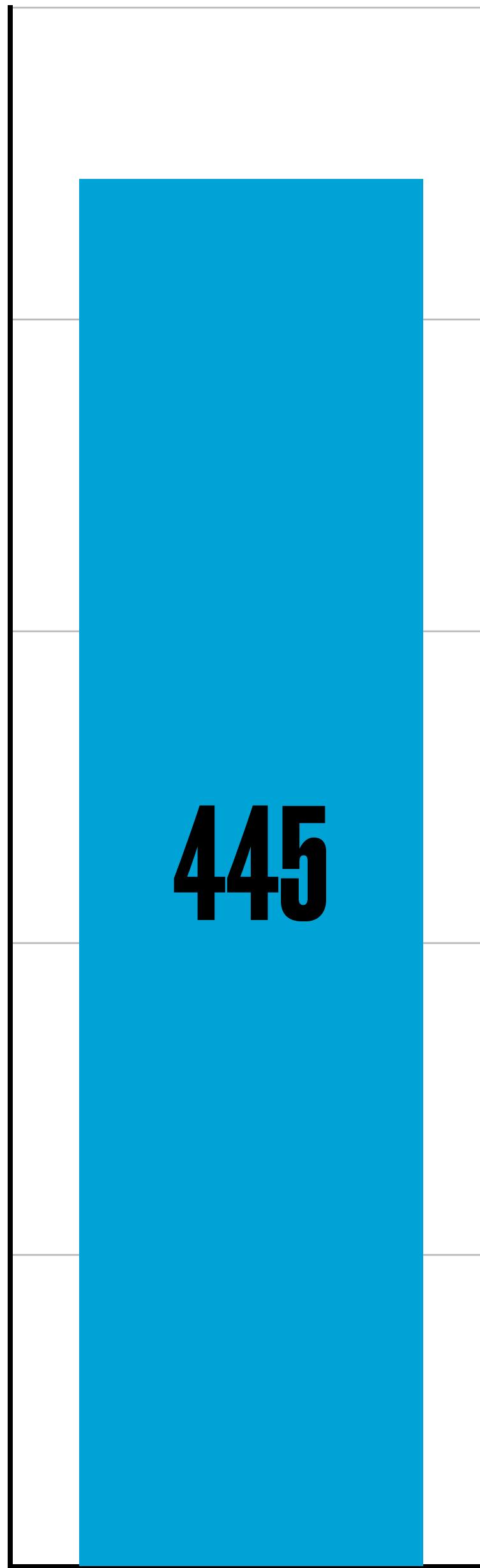
200 MB

100 MB

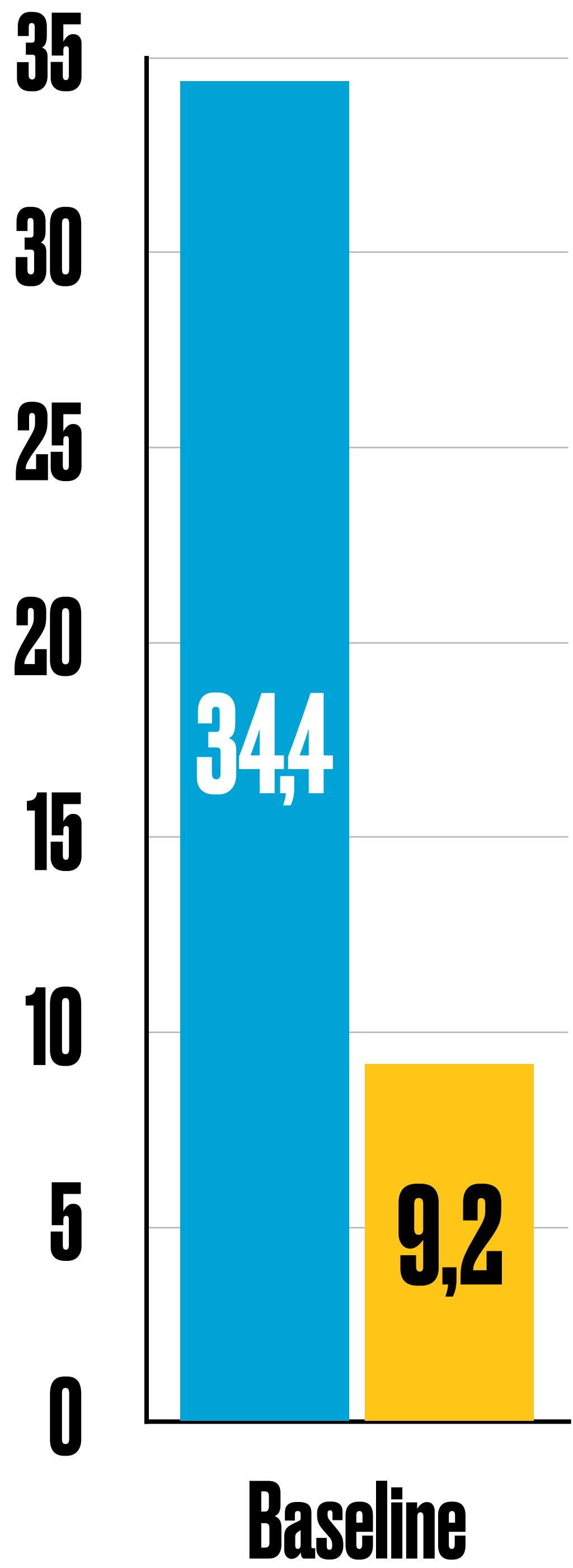
0 MB

Baseline

445



GARBAGE COLLECTIONS: STARTUP BENCHMARK



APP JAR: 67 MB

1.

SPRING BOOT TUNING

VOM SPRING-TEAM

VERLAGERT SCHRITTE ZUM BUILD

REDUZIERT STARTUP TIME

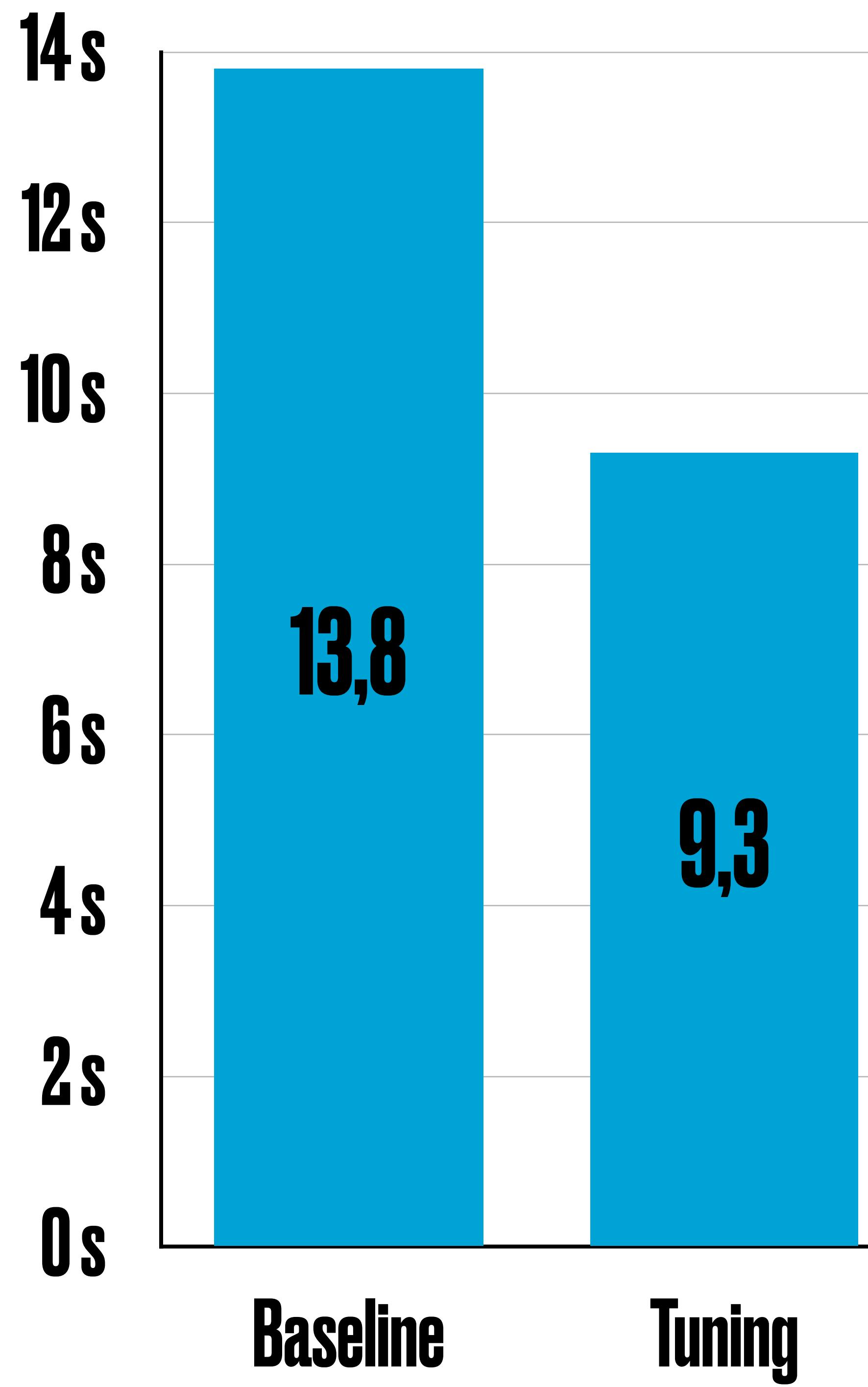
SPRING BOOT 3, JDK 17

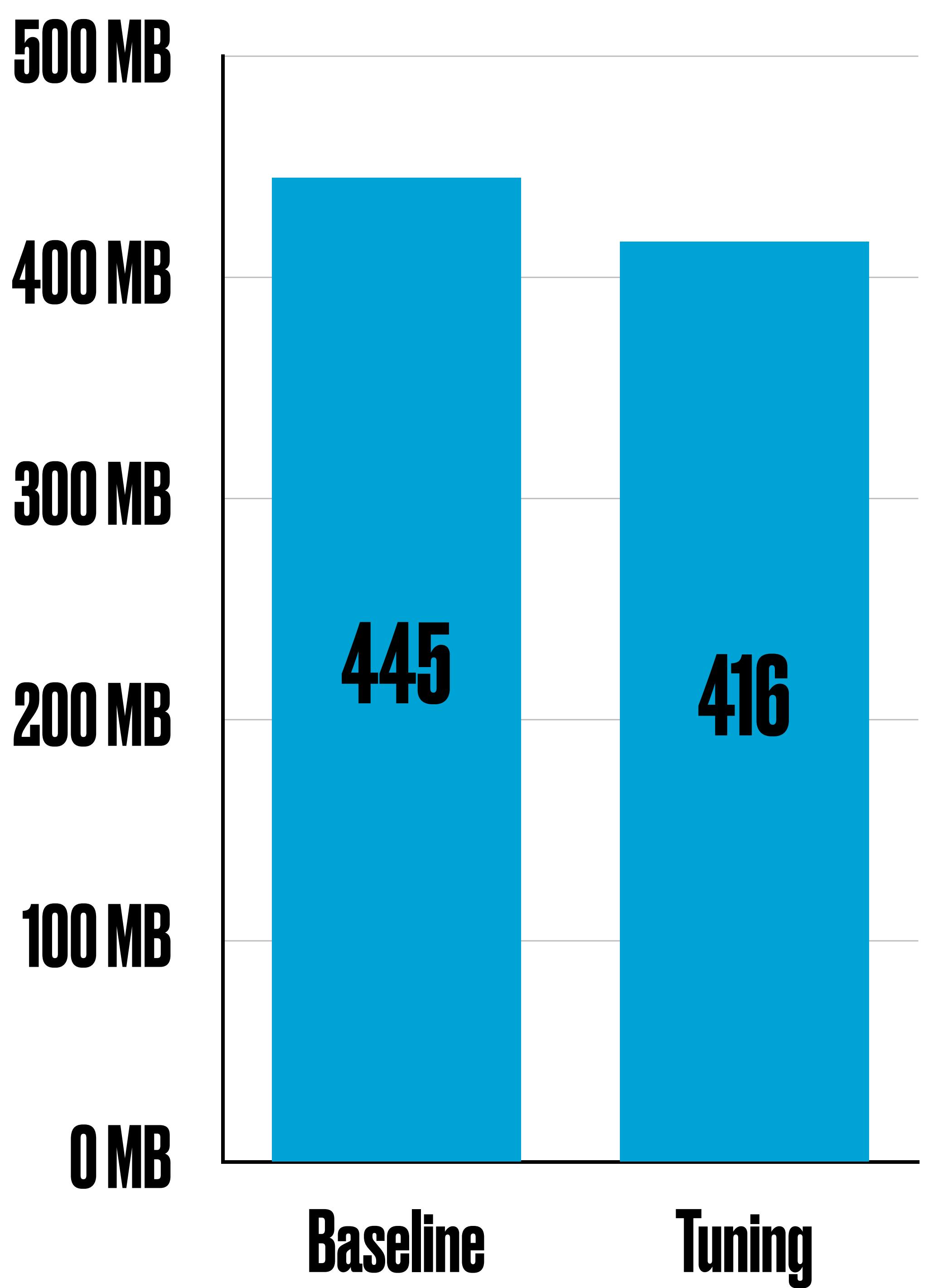
**SPRING-BOOT-INITIALISIERUNG
TEILWEISE IM BUILD**

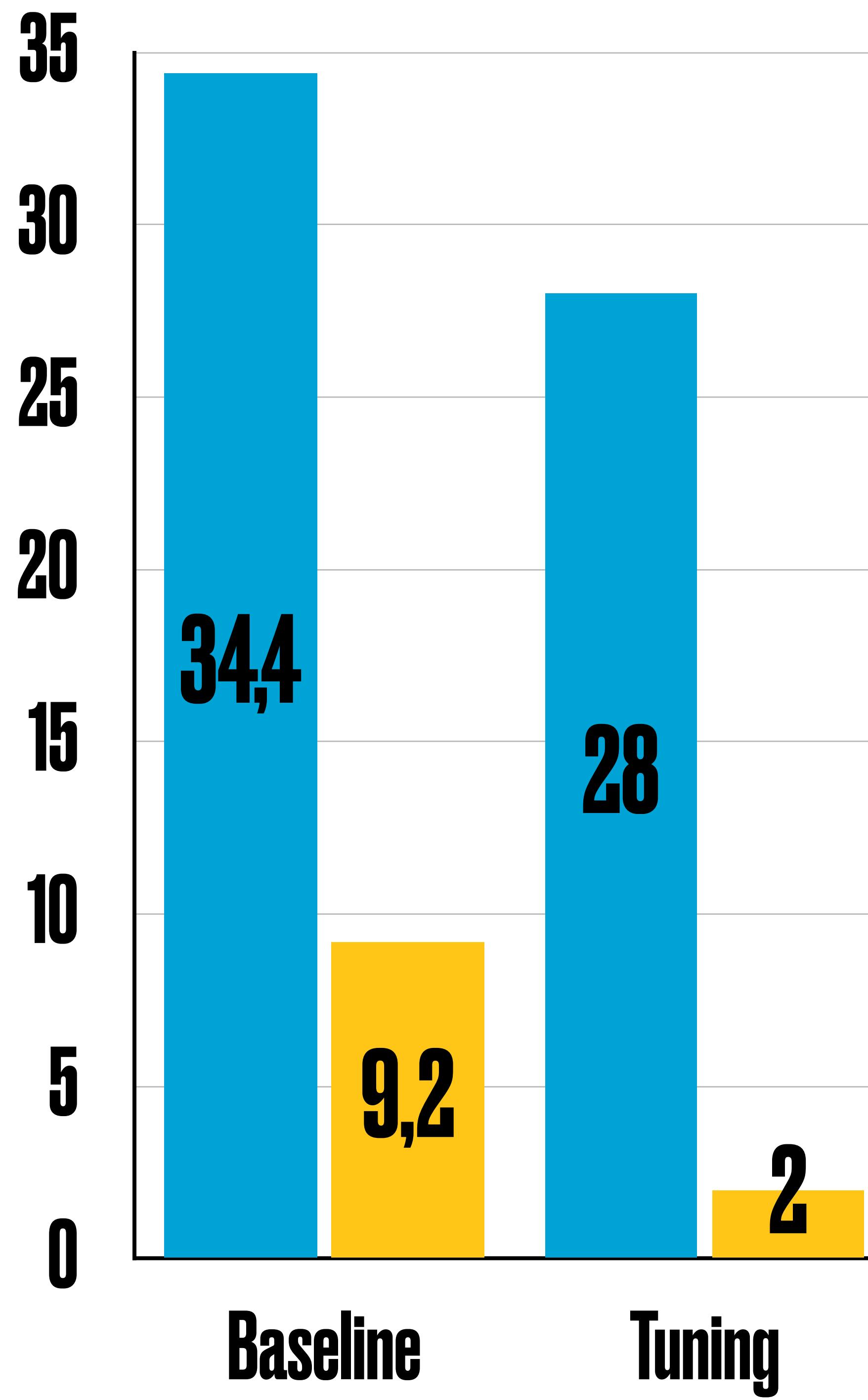
**SPRING AOT ENTFERNT NICHT
BENÖTIGTEN CODE & KONFIGURATION
BOOT-JAR ENTPACKT**

```
./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
```







APP JAR: 1 MB

LIB JARS: 118 MB

PROBLEME?

NISCHEN-BUGS FÜR SPRING AOT IN SPRING BOOT 3.5

	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

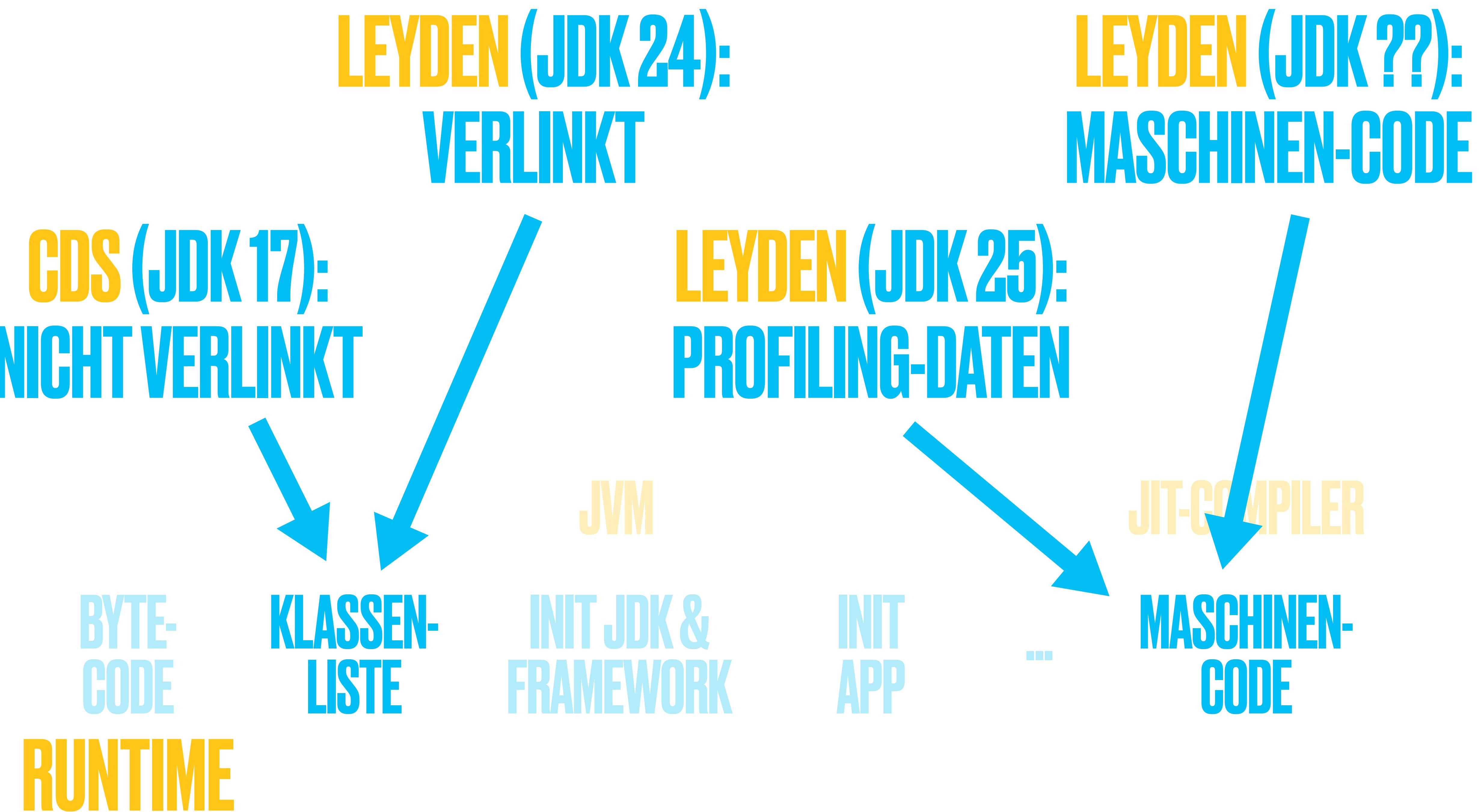
MY-APP.AOT

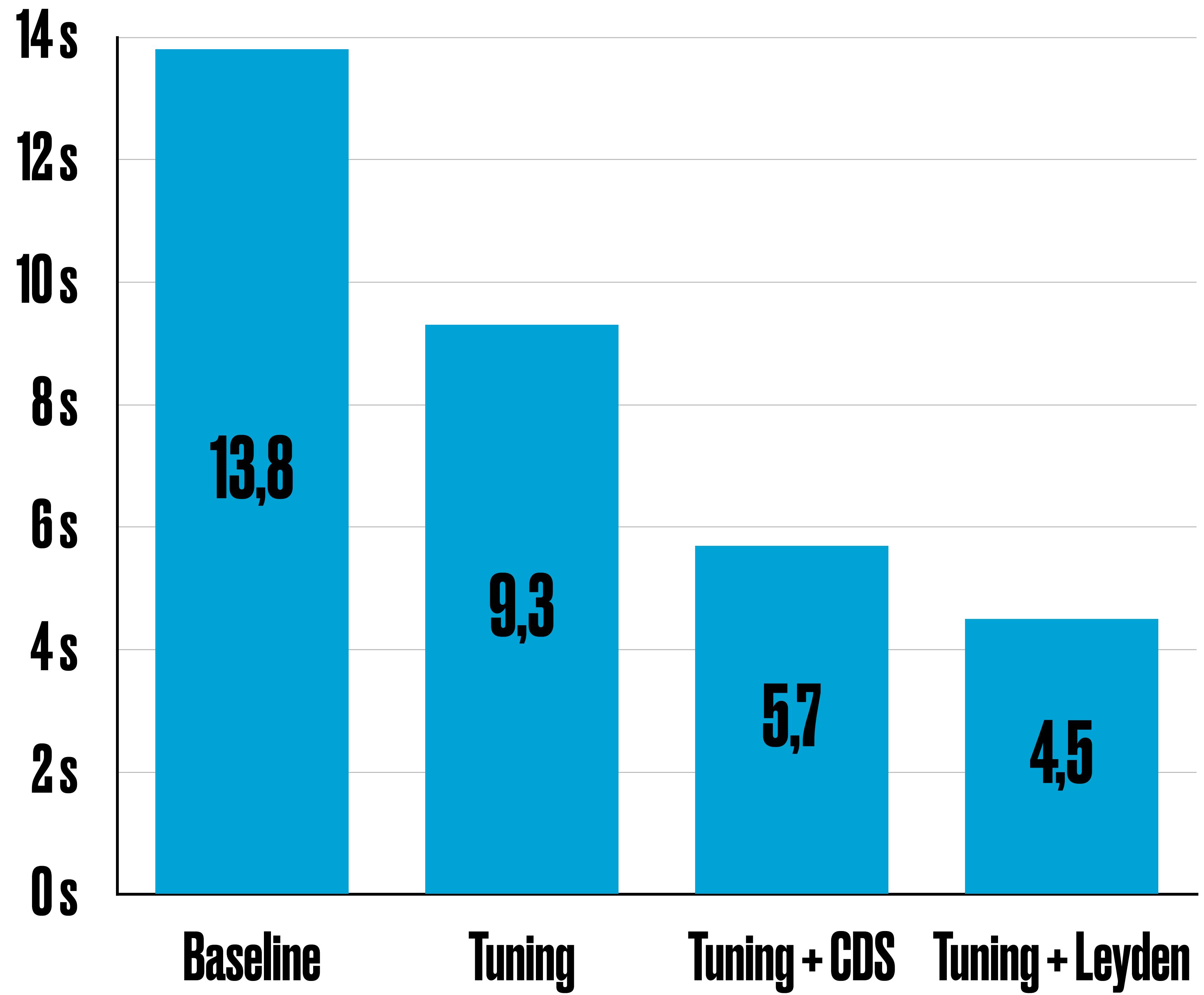
```
./GRADLEW CLEAN BOOTJAR && \
JAVA -DJARMODE=TOOLS \
-JAR BUILD/LIBS/MY-APP-1.4.JAR EXTRACT
```

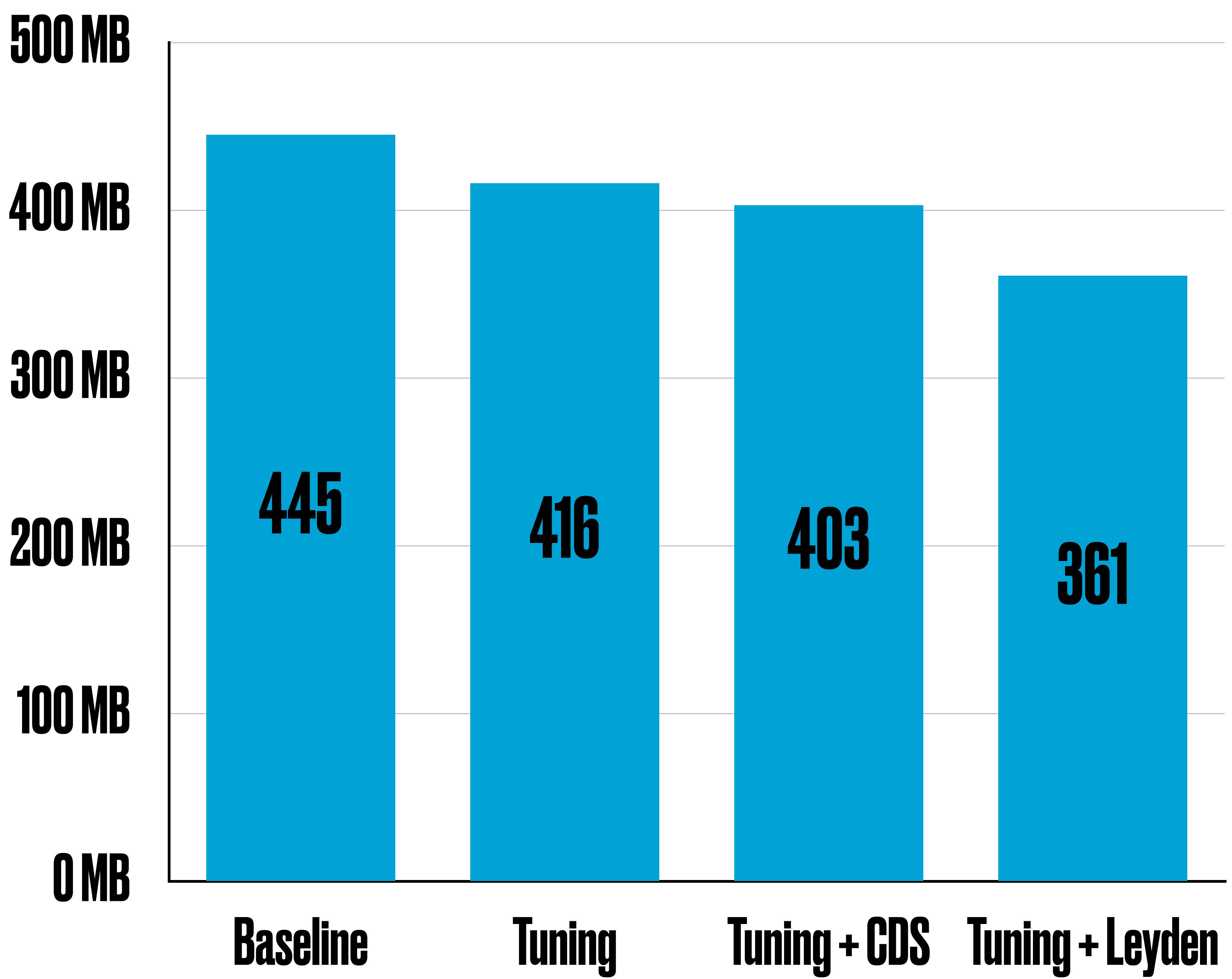
```
JAVA -DSPRING.AOT.ENABLED=TRUE \
-XX:AOTCACHEOUTPUT=MY-APP.AOT \
-JAR MY-APP-1.4/MY-APP-1.4.JAR
```

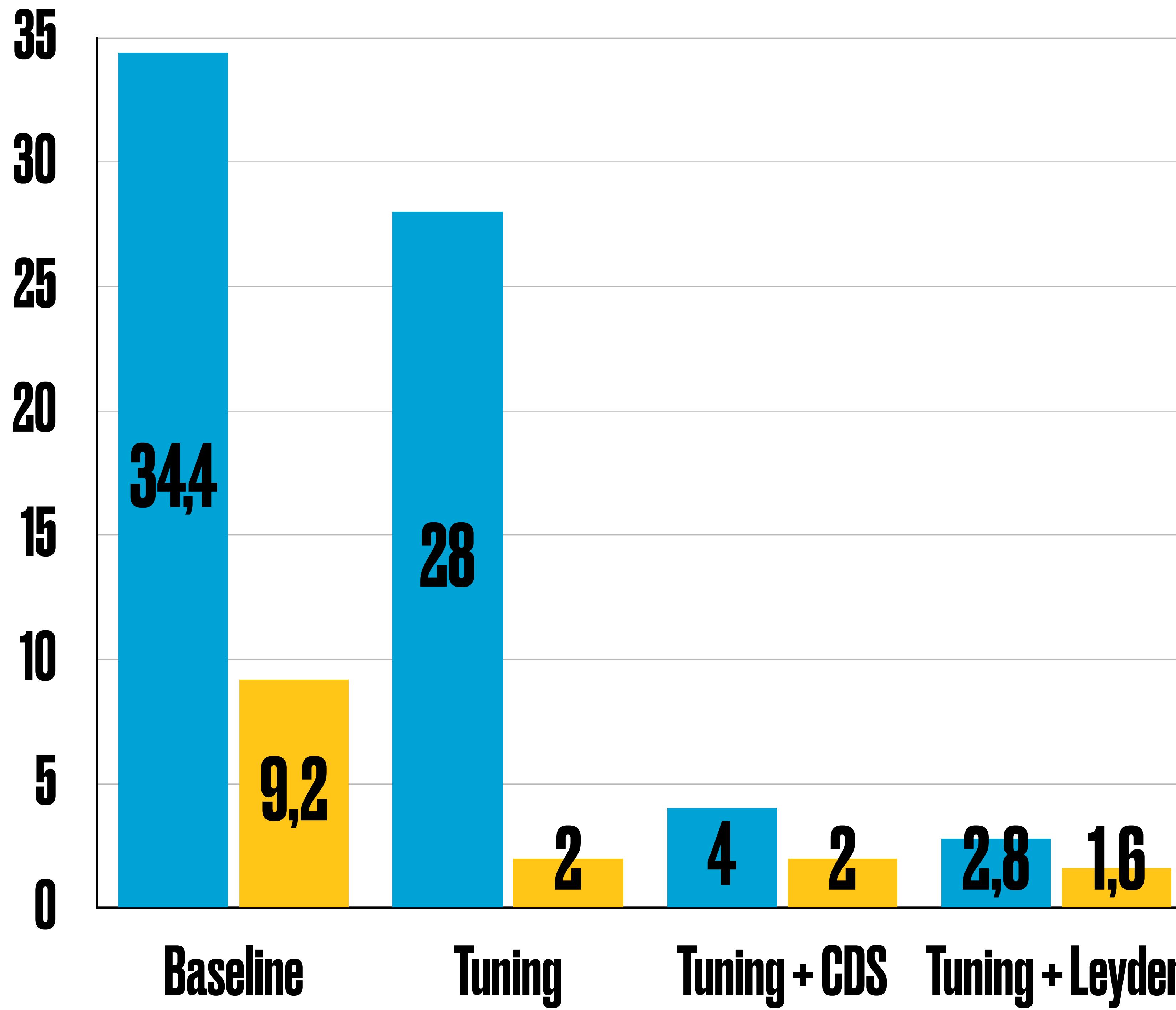
```
JAVA -DSPRING.AOT.ENABLED=TRUE \
-XX:AOTCACHE=MY-APP.AOT \
-JAR MY-APP-1.4/MY-APP-1.4.JAR
```

**WELCHE DATEN
IM CACHE?**









CDS CACHE. 102 MB

LEYDEN CACHE. 132 MB

PROBLEME?

**PRODUKTIONSNÄHER 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 WERDEN**

TUNING

ZEIT: SERVERLESS,
STANDBY, VERFÜG-
BARKEIT



CDS & LEYDEN

SPEICHER:
WENIGER SERVER

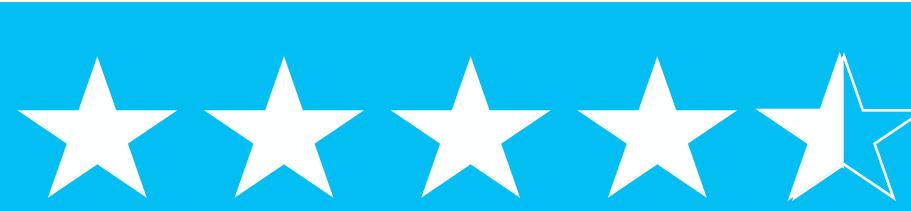
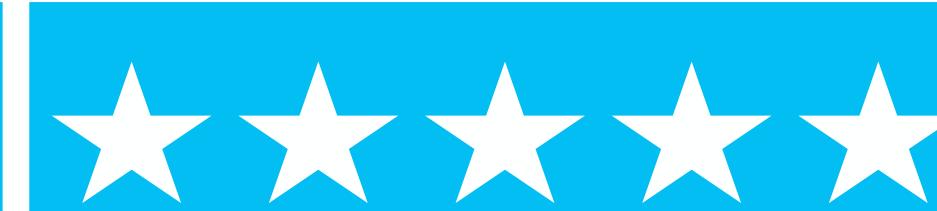


CRAC

KOSTEN



DEV EXPERIENCE



GRAALVM

3:

PROJECT CRAC

“COORDINATED RESTORE AT CHECKPOINT”

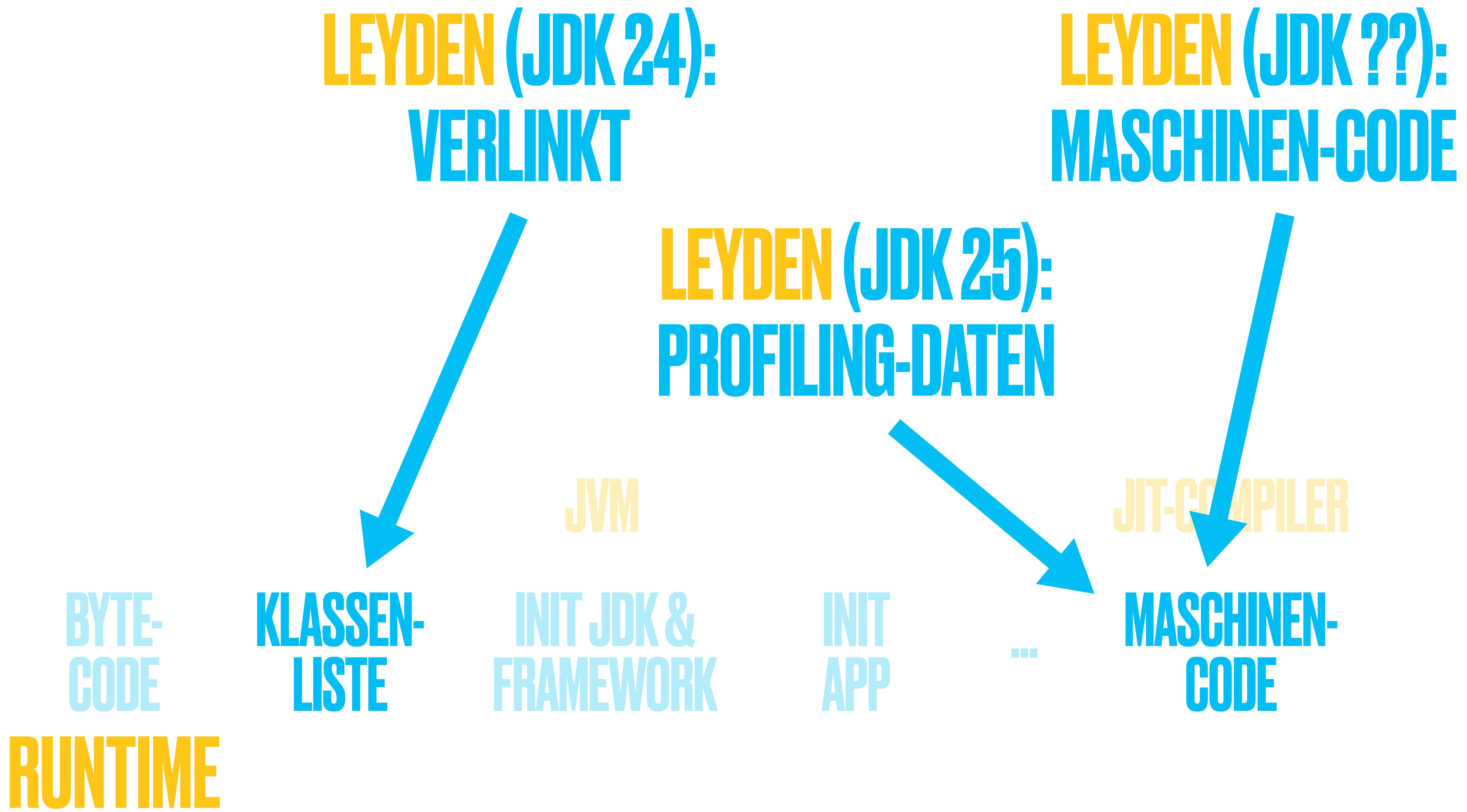
OPEN-JDK-PROJEKT, INITIAL VON AZUL.

CACHET ERGEBNISSE

REDUZIERT STARTUP TIME & TIME TO PEAK

JDK 17 (TEST: 24)

LEYDEN VS. CRAC



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

JVM

JIT-COMPILER

BYTE-
CODE

KLASSEN-
LISTE

INIT JDK &
FRAMEWORK

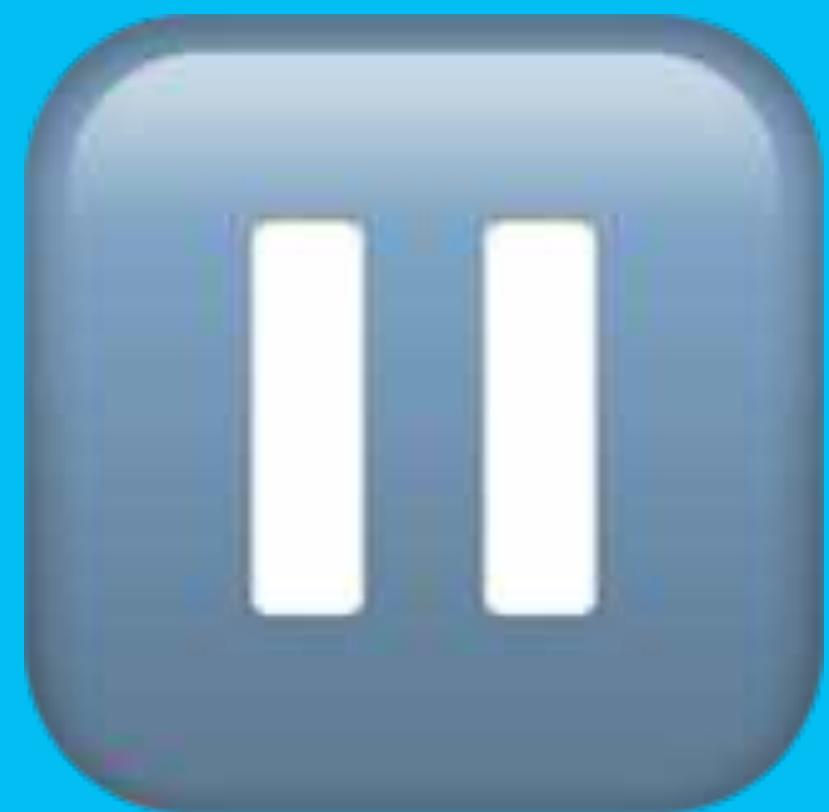
INIT
APP

...

MASCHINEN-
CODE

RUNTIME

TRAININGSLAUF:



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

PRODUKTION:

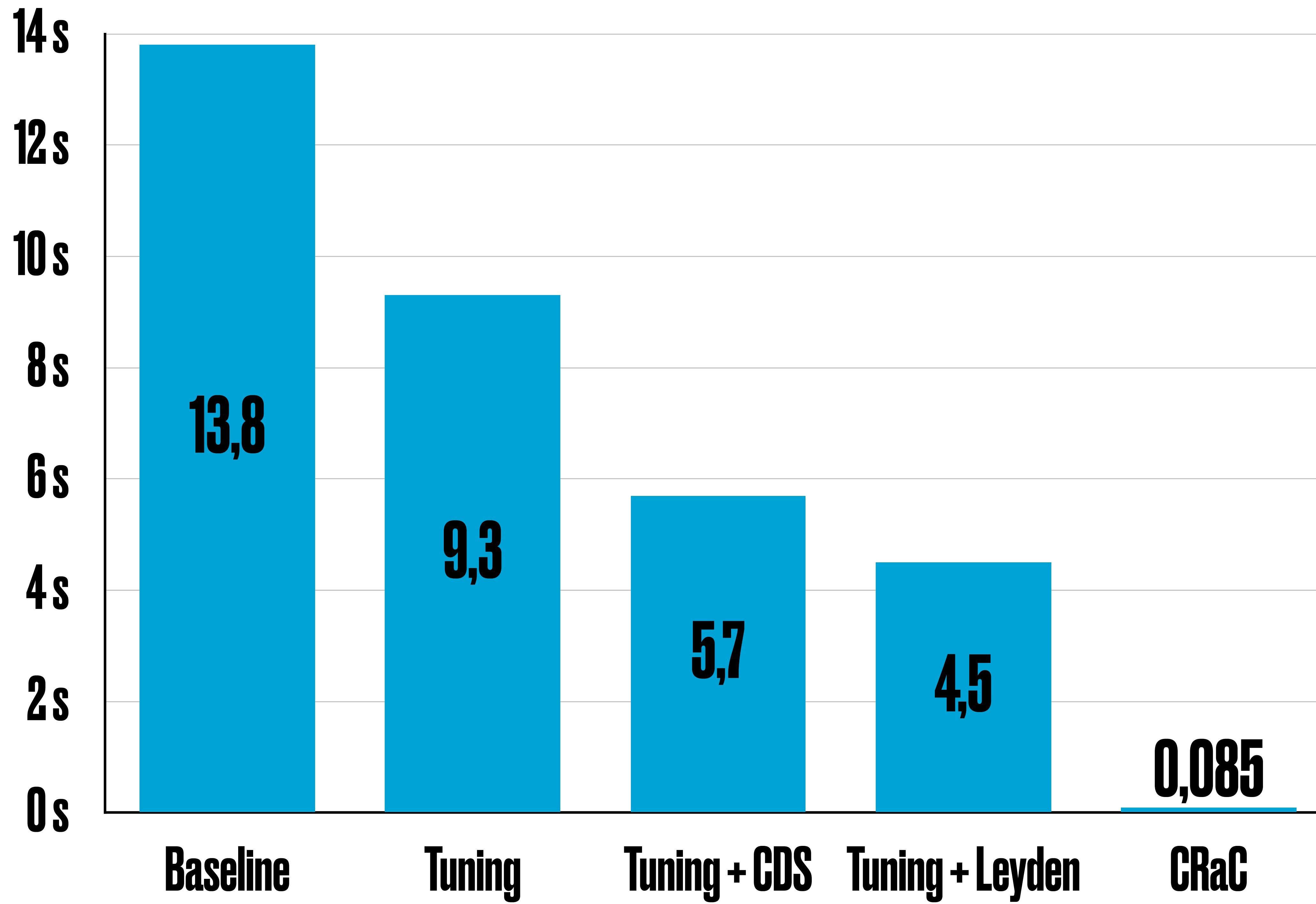


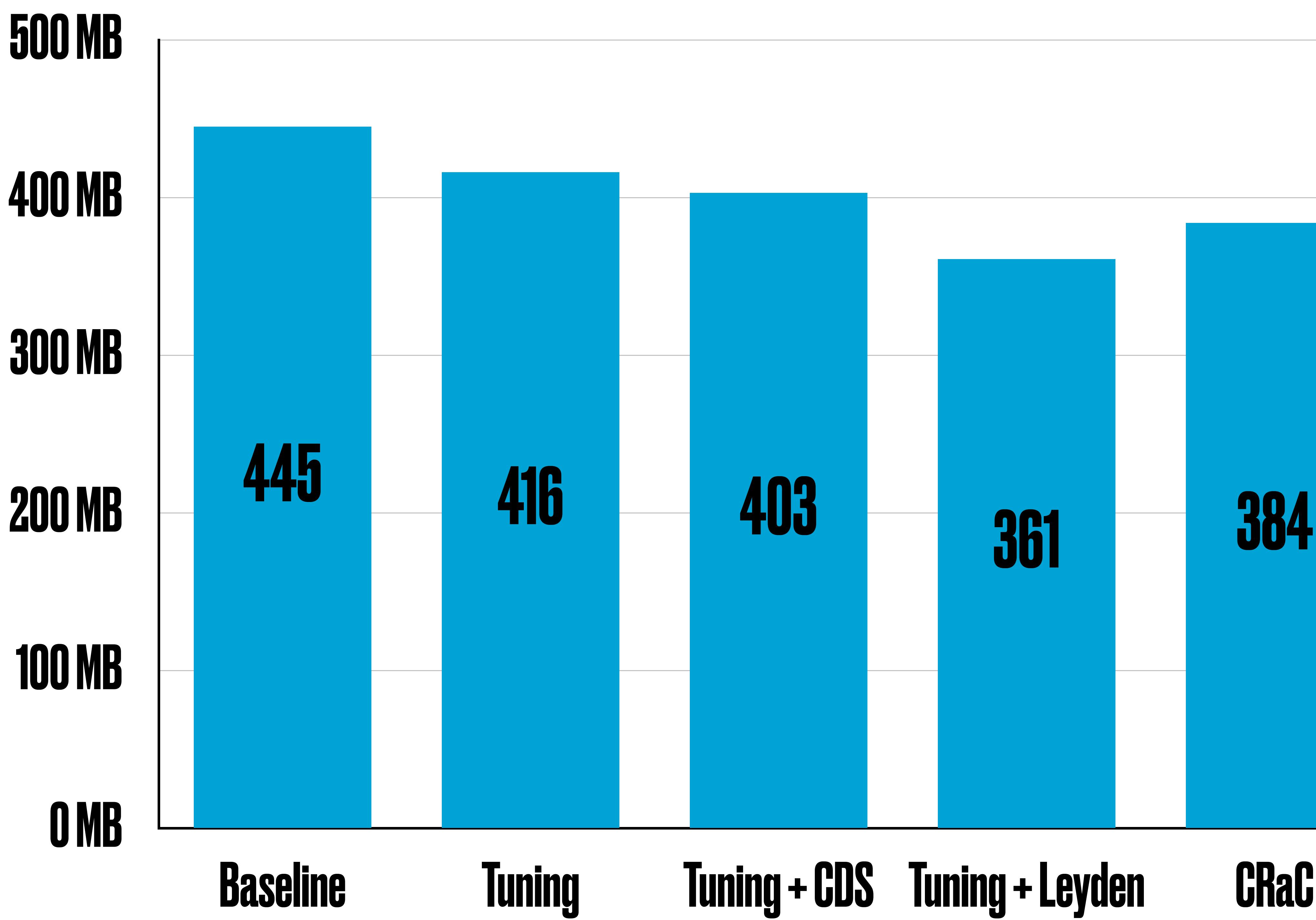
```
./GRADLEW CLEAN BOOTJAR -P CRAC=TRUE
```

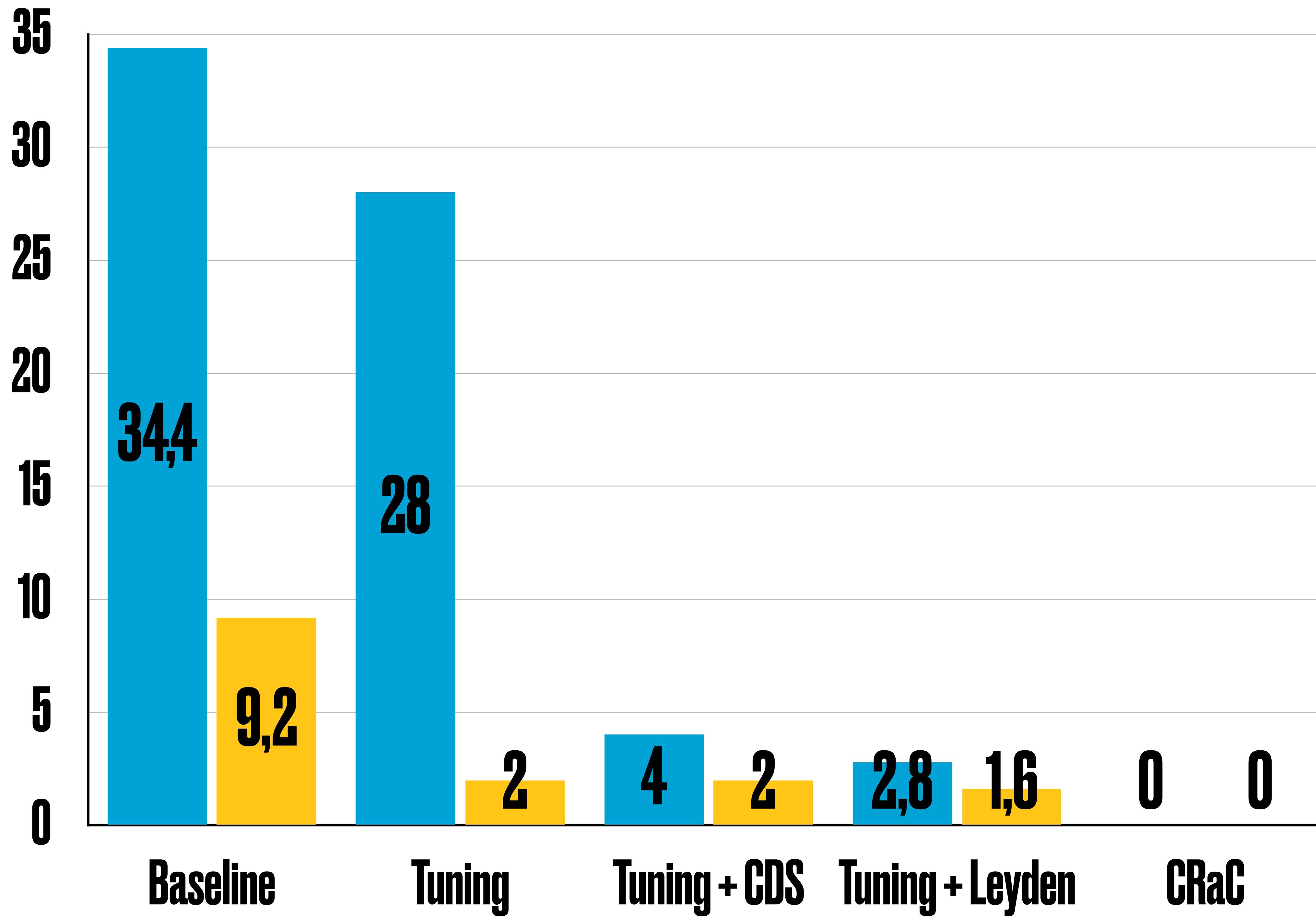
```
JAVA -XX:CRACENGINE=WARP \
-XX:CRACCHECKPOINTTO=MY-APP-CRAC
-JAR BUILD/LIBS/MY-APP-1.4.JAR
```

```
JCMD [PID] JDK.CHECKPOINT
```

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







SNAPSHOT: 229 MB

PROBLEME?

"CRAC TRIES TO MAKE THINGS RUN FASTER POTENTIALLY AT THE EXPENSE OF CORRECTNESS, 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

**ZEIT: SERVERLESS,
STANDBY, VERFÜG-
BARKEIT**



**SPEICHER:
WENIGER SERVER**



KOSTEN



DEV EXPERIENCE



CDS & LEYDEN



CRAC



GRAALVM



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: 24)**

**"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
COMPILERUNGS-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)

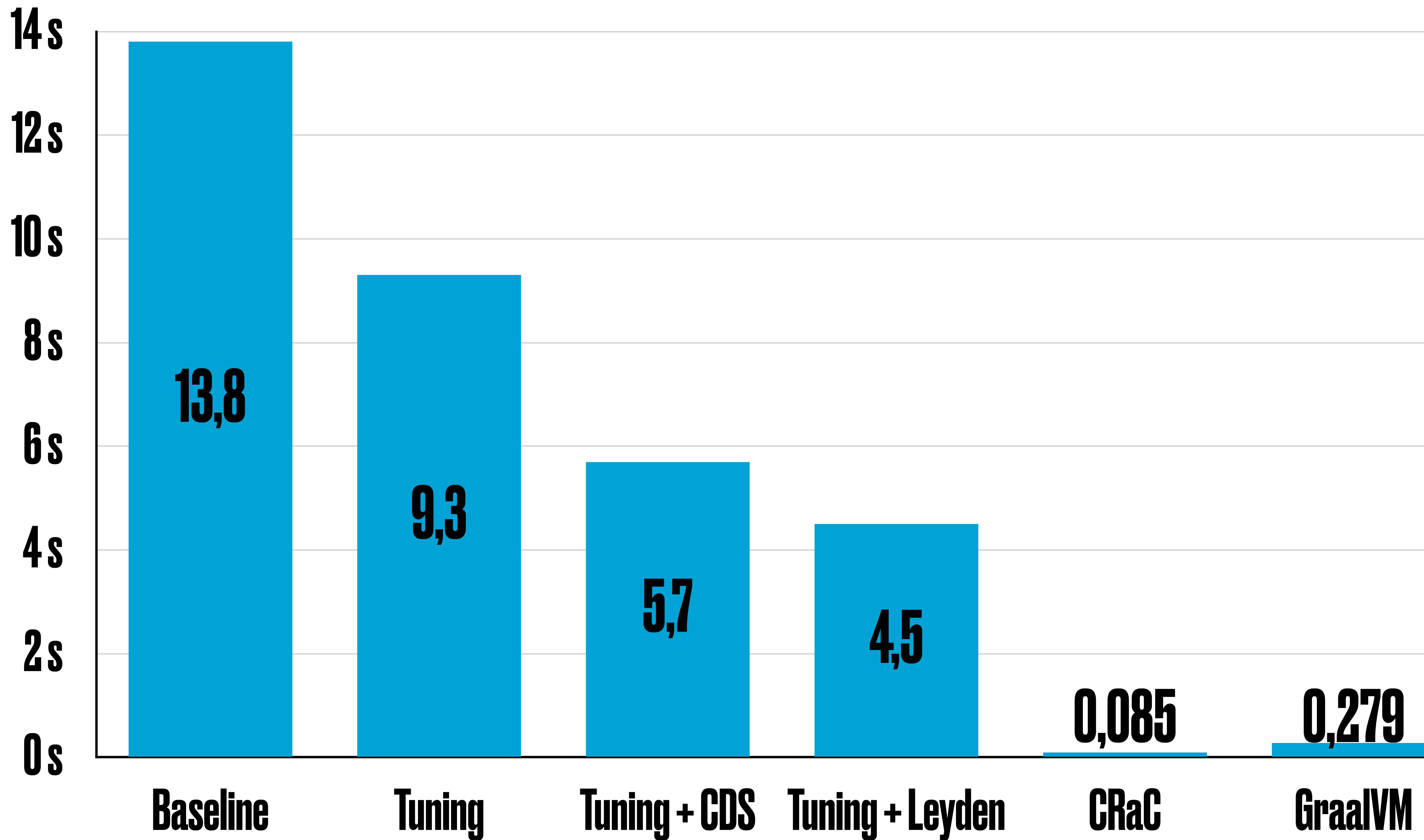
**SEPARATER DOWNLOAD, ABER RELEASES
SYNCHRON MIT OPENJDK**

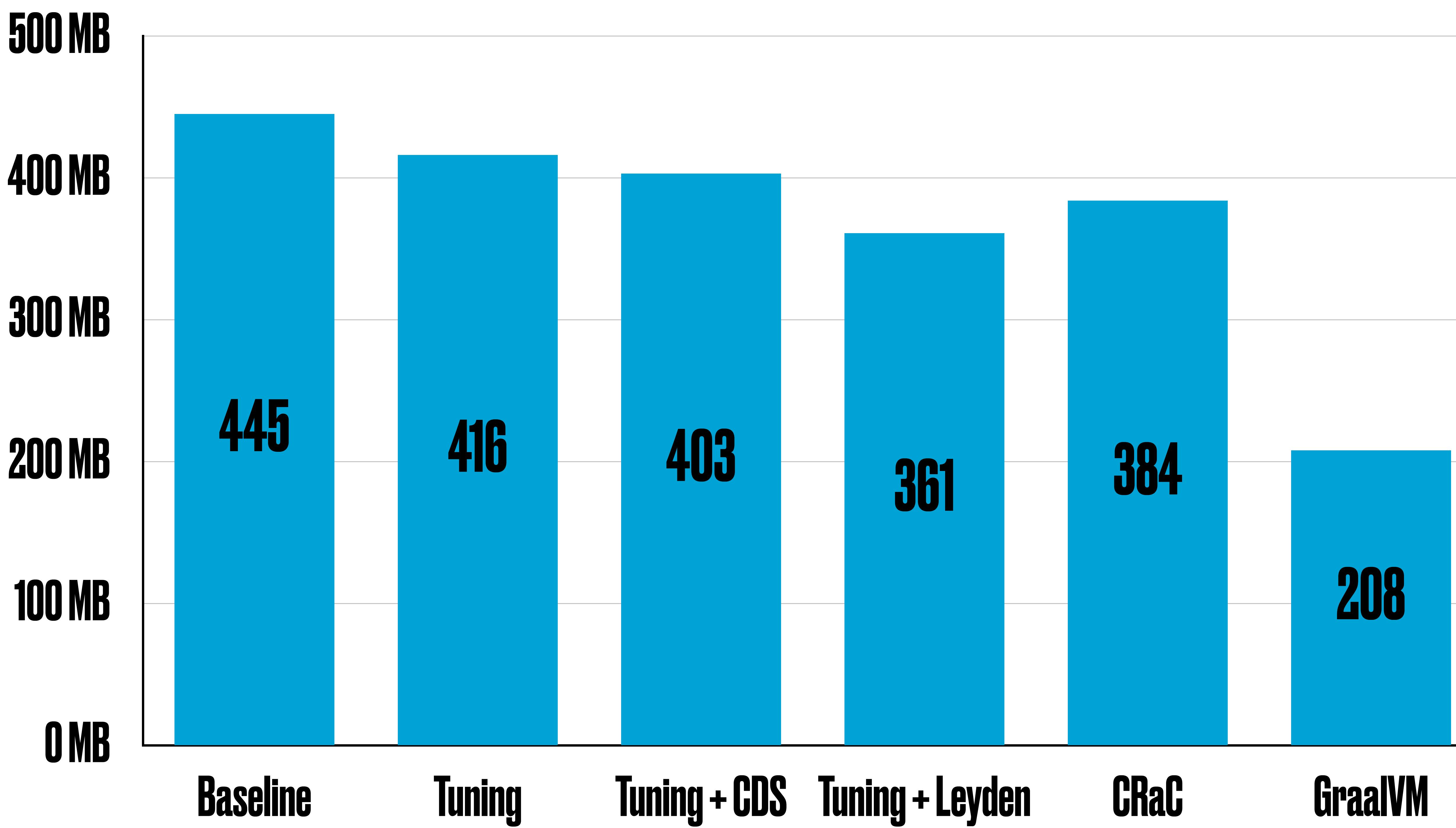
```
./GRADLEW NATIVECOMPILE --PGO-INSTRUMENT
```

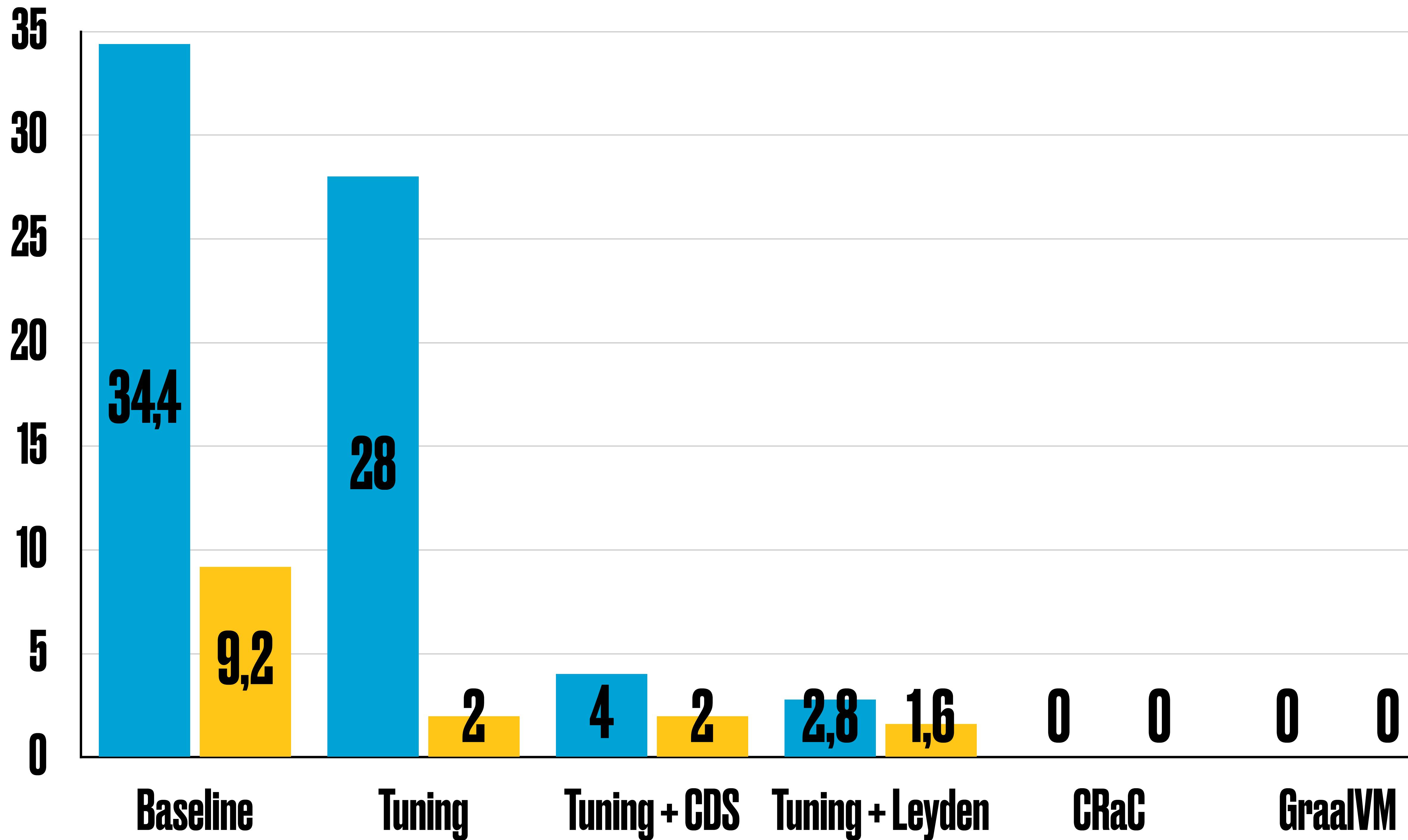
```
./BUILD/NATIVE/NATIVECOMPILE/MY-APP-  
INSTRUMENTED
```

```
./GRADLEW NATIVECOMPILE
```

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







LINUX APP: 172 MB

NATIVE IMAGE:

9:43 MN & 5:34 MN

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

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

JAVA BY DESIGN:
MEHR ZEIT
MEHR CPU
MEHR SPEICHER

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://BPELI.ATC