KARSTEN SILZ, FEBRUARY 16, 2023

WHEN IS NATIVE JAVA WITH GRAALVM WORTHWHILE FOR ME?

LJC LIVE FEBRUARY 2023

METRO BANK

WHEN IS NATIVE JAVA WITH GRAALVM WORTHWHILE FOR ME?

75% OF YOU: NOT NOW

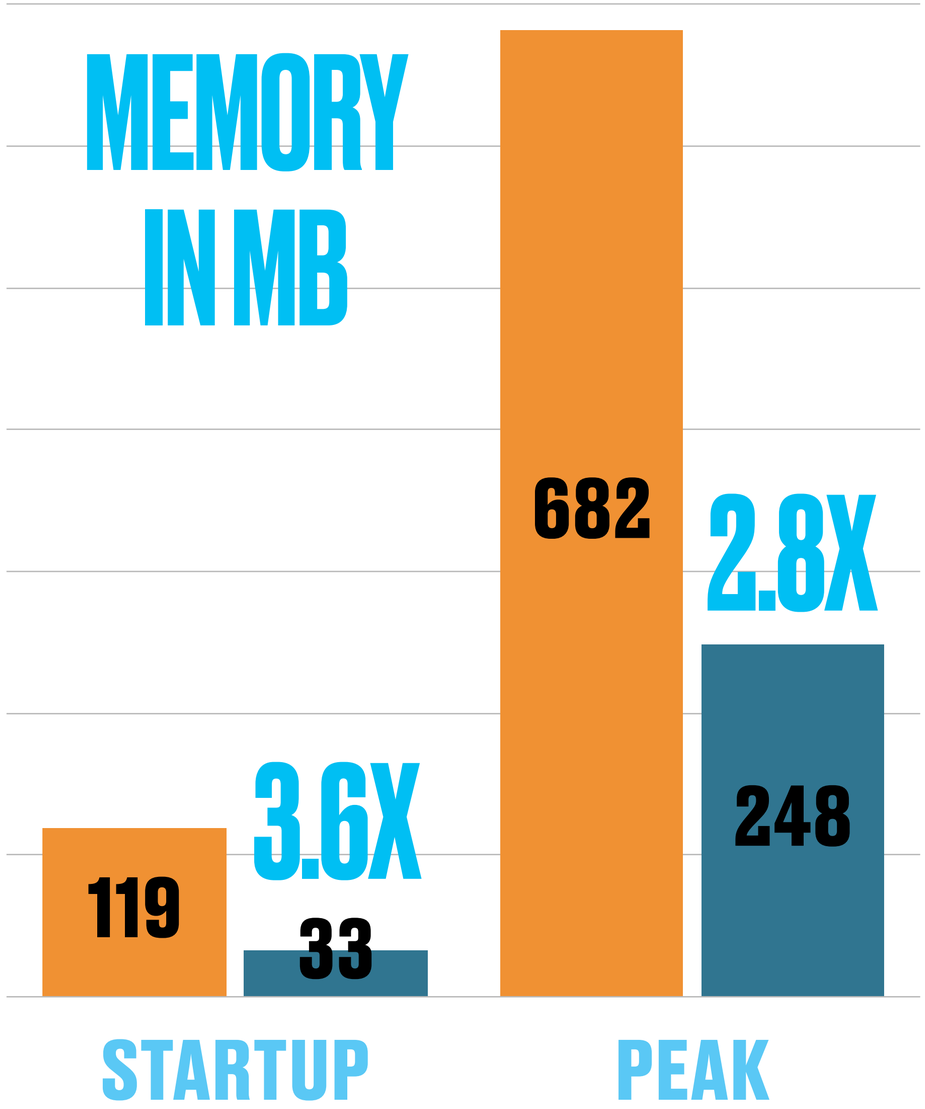

The End

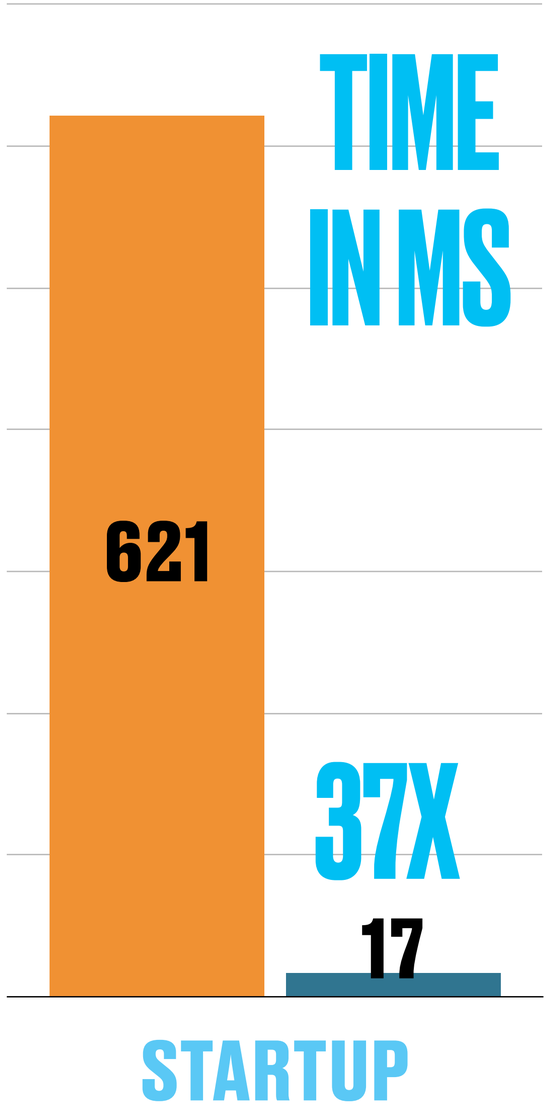
TALK SUMMARY

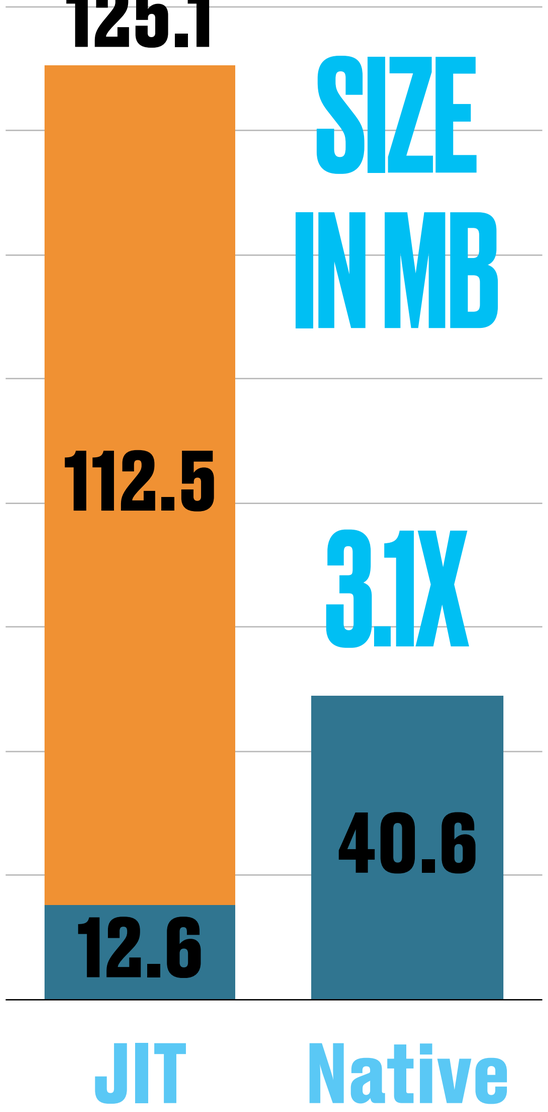
NATIVE JAVA:
SPEND DEVELOPER TIME TO SAVE OPERATIONAL COSTS

FASTER STARTUP
LESS MEMORY
SMALLER FILES
BETTER SECURITY

SPRING BOOT 3: JIT VS. NATIVE (LOWER = BETTER)







SWEET SPOT: MICROSERVICES ON K8S – MORE PODS PER NODE

SOMETIMES DOESN'T WORK

OFTEN MAKES NO BUSINESS SENSE

HARDER TO LEARN & USE, LOW JOB DEMAND

WHY AM I AN EXPERT?

I GOT DATA TO BACK UP MY CLAIMS

EDITOR OF 6-PART INFOQ ARTICLE SERIES ON NATIVE JAVA

JAVA EXPERTS HELPED ME

EVALUATE & RECOMMEND TECHNOLOGIES FOR 20 YEARS

BETTER PROJECTS FASTER

SLIDES

ADDITIONAL INFORMATION

SAMPLE CODE REPOS

BPF.LI/LAD

AGENDA

PROBLEM? AND WHY NOW?

NATIVE JAVA SOLUTION

JIT JAVA SOLUTION

WHEN WORTHWHILE?

PROBLEM? AND WHY NOW?

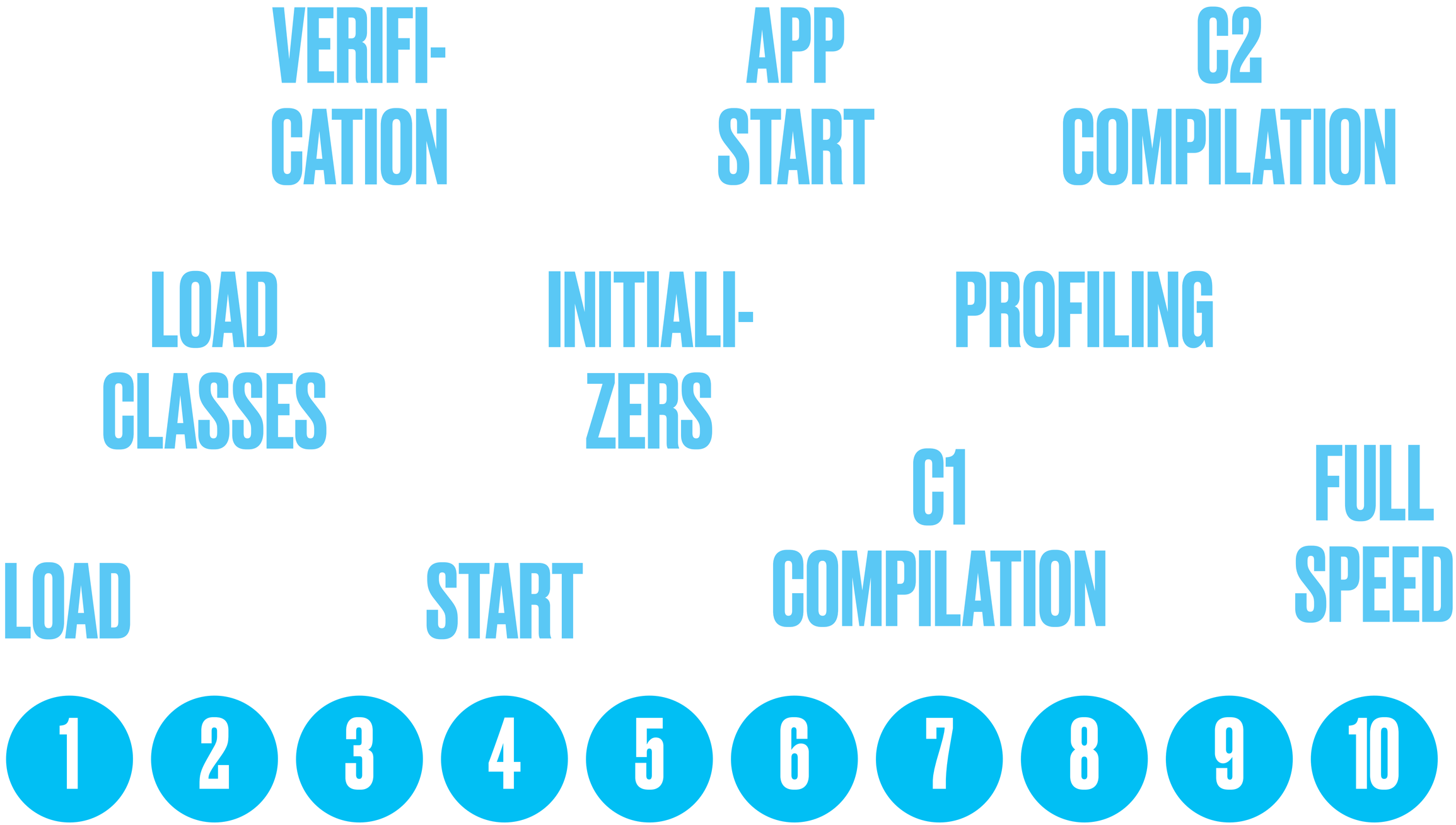
"THE LONG-TERM PAIN POINTS OF JAVA'S SLOW STARTUP TIME, SLOW TIME TO PEAK PERFORMANCE, AND LARGE FOOTPRINT"

JAVA LANGUAGE ARCHITECT MARK REINHOLD,
ORACLE, APRIL 2020

SLOW STARTUP,
SLOW TIME TO PEAK
PERFORMANCE,
LARGE FOOTPRINT

WHY SLOW?

JAVA: RUNTIME


VERIFI-
CATION
APP
START
C2
COMPILATION
LOAD
CLASSES
INITIALI-
ZERS
PROFILING
C1
COMPILATION
FULL
SPEED
LOAD
START
1
2
3
4
5
6
7
8
9
10

10 STEPS =
A LOT

C1 + C2 COMPILATION: HOTSPOT JIT COMPILER - ALSO USES MEMORY

WHY CLASS FILES & JIT COMPILER?

MISSION & INCENTIVES

JAVA THEN (E.G., 2010)

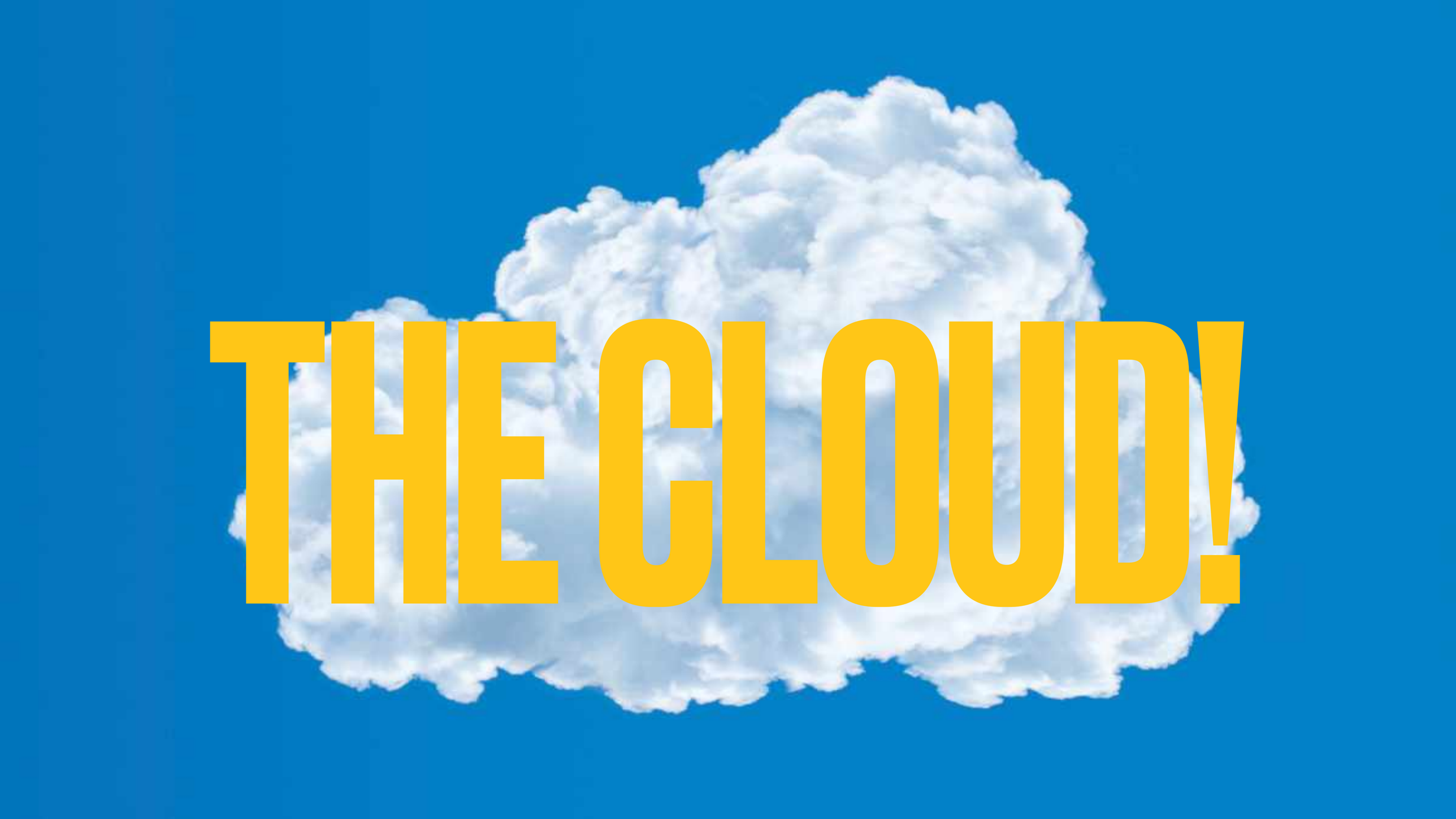
MISSION: RUN EVERYWHERE

- PLATFORM-INDEPENDENT BYTE CODE IN CLASS FILES

INCENTIVE: SELL TO ENTERPRISES

- OPTIMIZE FOR LONG-RUNNING PROCESSES & LARGE HEAPS (JAVA EE APP SERVERS)

WHAT CHANGED?


THE CLOUD!

SOMEBODY ELSE RUNS MY SERVERS...

...AND I MAY PAY BY RESOURCE USAGE!

JAVA NOW

MISSION: RUN EVERYWHERE

PLATFORM-INDEPENDENT BYTE CODE IN CLASS FILES

MISSION: RUN IN LINUX CONTAINERS

NATIVE EXECUTABLES ARE BETTER

INCENTIVE: SELL TO ENTERPRISES

OPTIMIZE FOR LONG-RUNNING PROCESSES & LARGE HEAPS (JAVA EE APP SERVERS)

INCENTIVE: RUN IN CLOUD

PAY LESS FOR LOWER MEMORY & CPU USAGE AND SMALLER FILE SIZES

JAVA APPLICATIONS: HOW MANY CPU CORES? (NEW RELIC, APRIL 2022)

OUTSIDE: 58% >=4 CORES

IN: 62% 1-2 CORES,

74% 1-4 CORE

JAVA APPLICATIONS: HOW MUCH MEMORY? (NEW RELIC, APRIL 2022)

OUTSIDE: 47% > 2 GB

IN: 73% <= 2 GB

JAVA APPLICATIONS IN CONTAINERS: FEWER CPU CORES, LESS MEMORY TODAY

SECTION SUMMARY

JAVA HAS...

SLOW STARTUP,
SLOW TIME TO PEAK
PERFORMANCE,
LARGE FOOTPRINT

...BECAUSE IT WAS TO RUN...

LONG-RUNNING PROCESSES WITH LARGE HEAPS EVERYWHERE

THAT IS MORE EXPENSIVE IN THE CLOUD

AGENDA

~~PROBLEM? AND WHY NOW?~~

NATIVE JAVA SOLUTION

JIT JAVA SOLUTION

WHEN WORTHWHILE?

NATIVE JAVA SOLUTION

GRAALVM

JDK

OPENJDK WITH GRAALVM JAVA COMPILERS (WRITTEN IN JAVA), REPLACES HOTSPOT JIT COMPILER (C++)

TRUFFLE

RUNS OTHER LANGUAGES (JAVASCRIPT, PYTHON, ...)

NATIVE IMAGE

AHEAD-OF-TIME (AOT) JAVA COMPILER FOR NATIVE EXECUTABLES – THINK C++ COMPILER

FASTER STARTUP
LESS MEMORY
SMALLER FILES
BETTER SECURITY

HOW CAN NATIVE IMAGE DO THIS?

THROW UNUSED STUFF AWAY & DO MORE AT BUILD TIME


easy

NATIVE JAVA: RUNTIME

LOAD

SOME INITIALIZERS

FULL SPEED

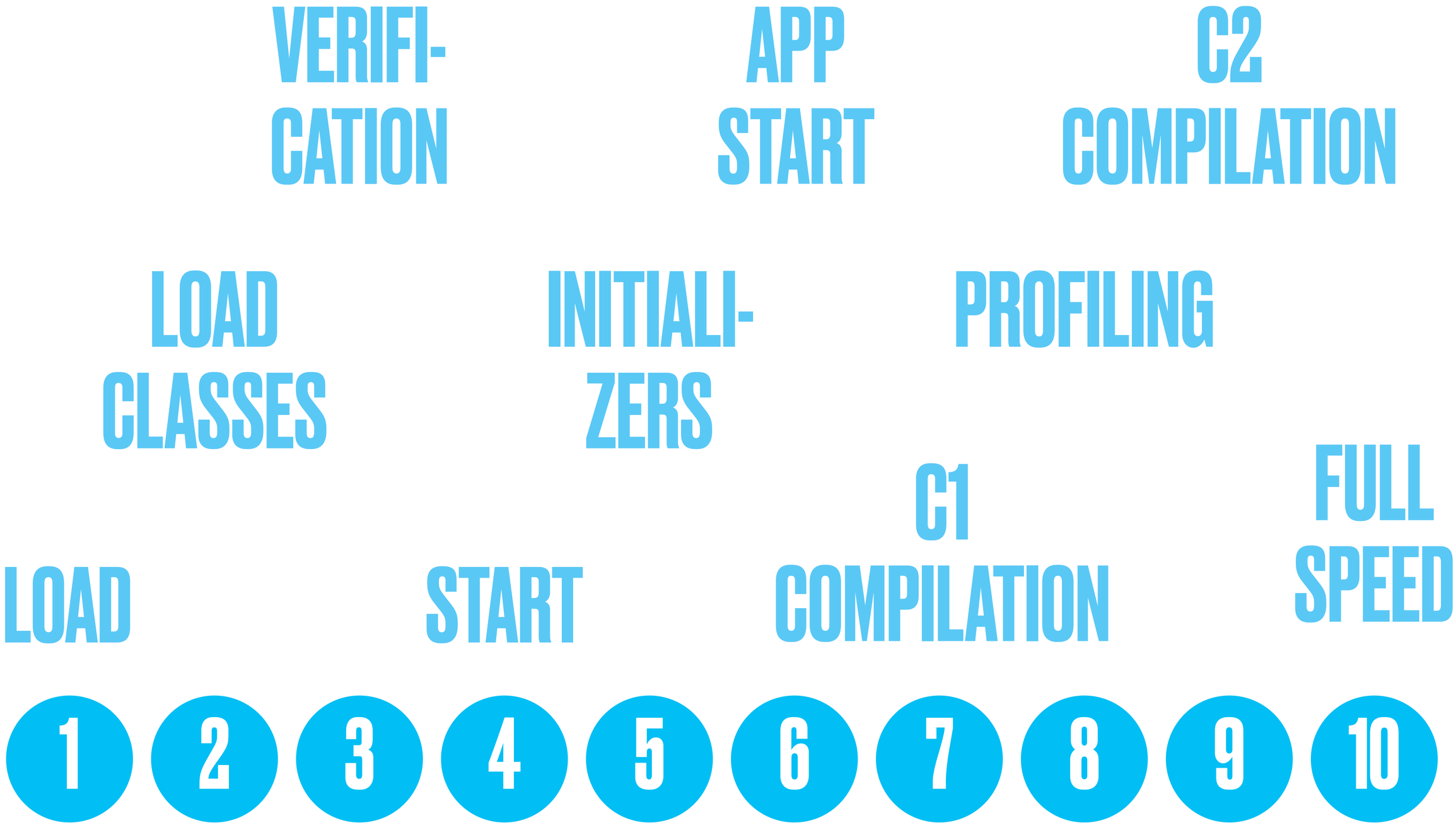
1

2

3

3 STEPS =
VERY QUICK

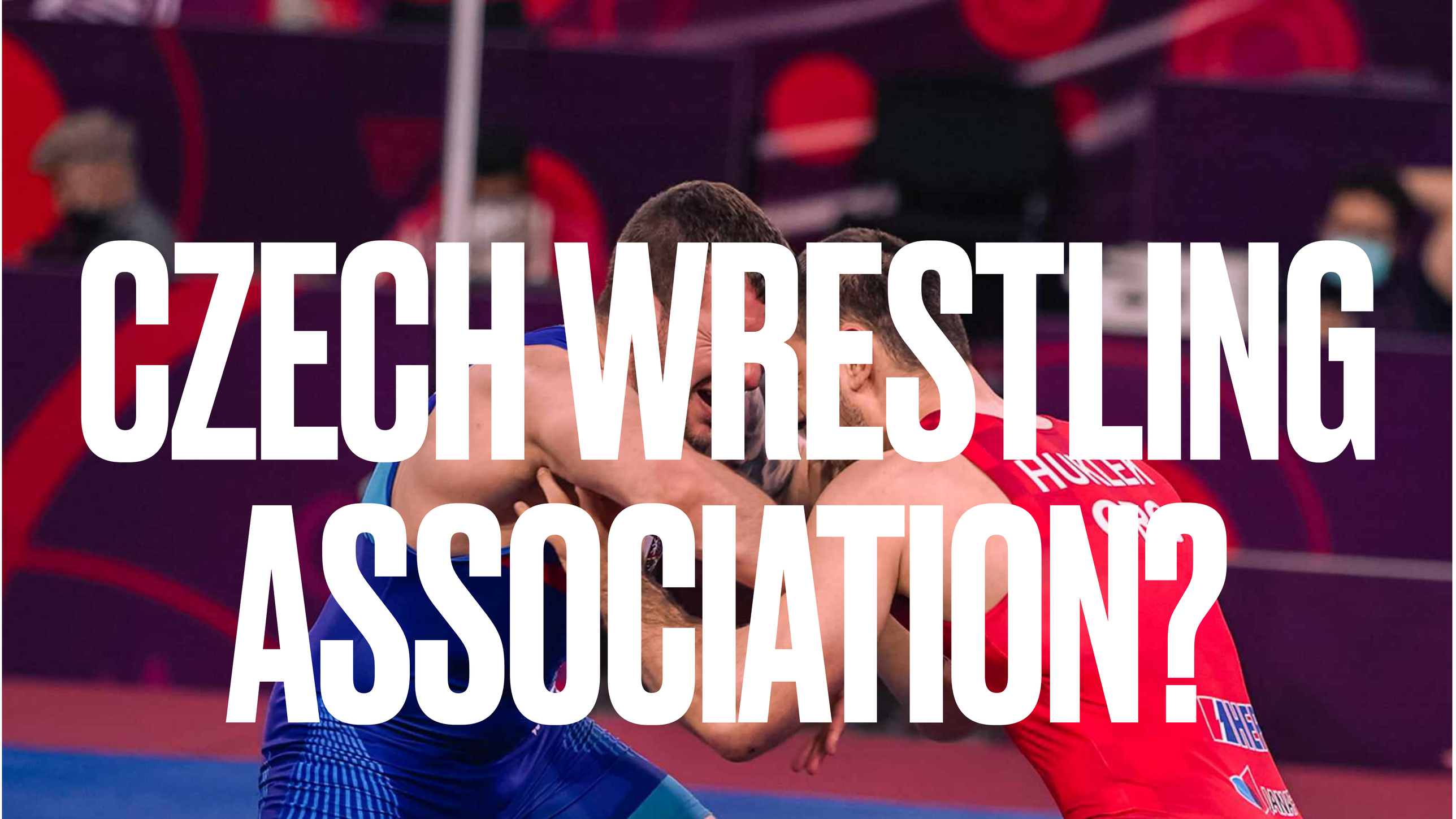
JIT JAVA: RUNTIME


VERIFI-
CATION
APP
START
C2
COMPILATION
LOAD
CLASSES
INITIALI-
ZERS
PROFILING
C1
COMPILATION
FULL
SPEED
LOAD
START
1
2
3
4
5
6
7
8
9
10

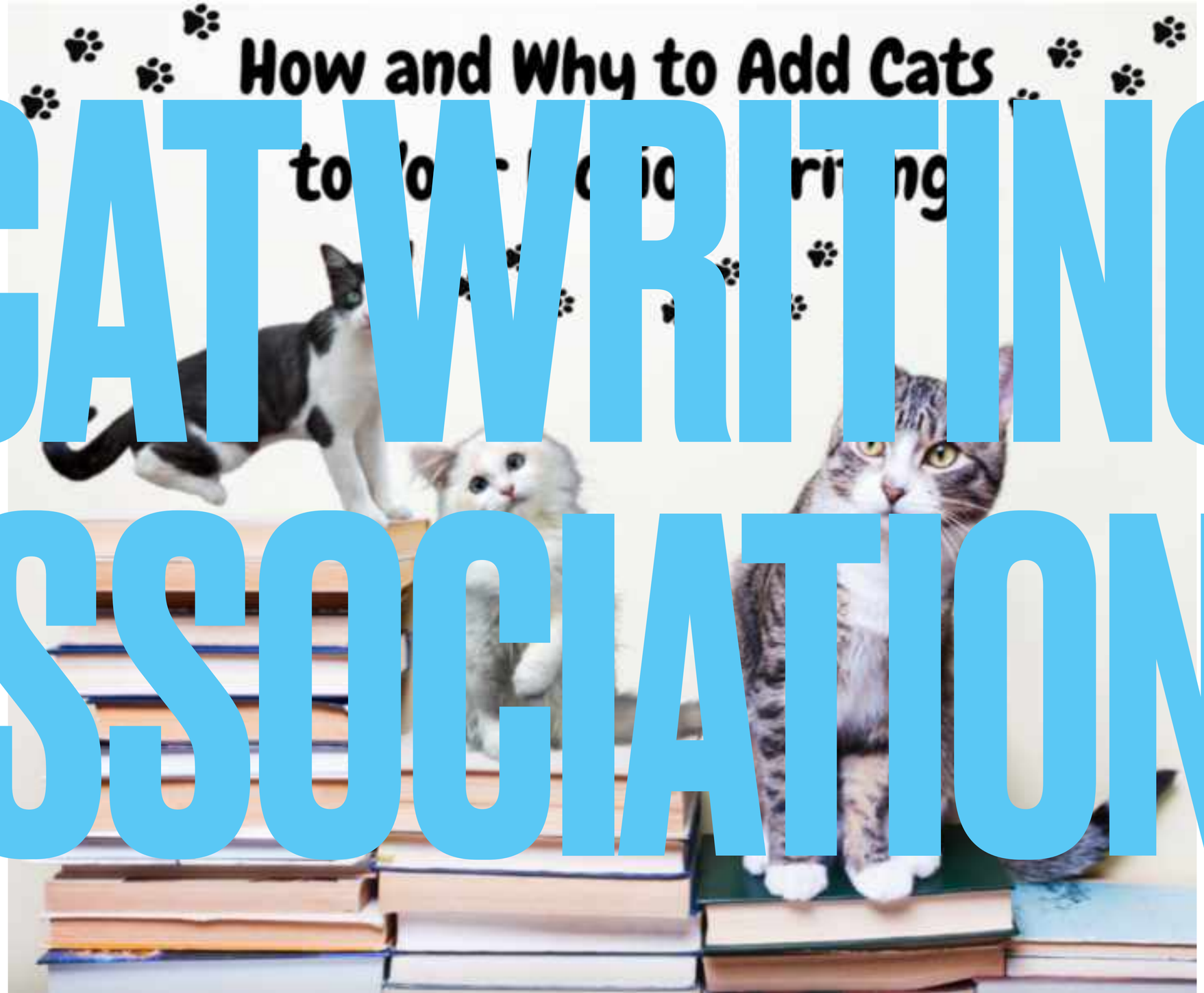
WHAT'S THE CATCH?

WE MUST USE THE CWA

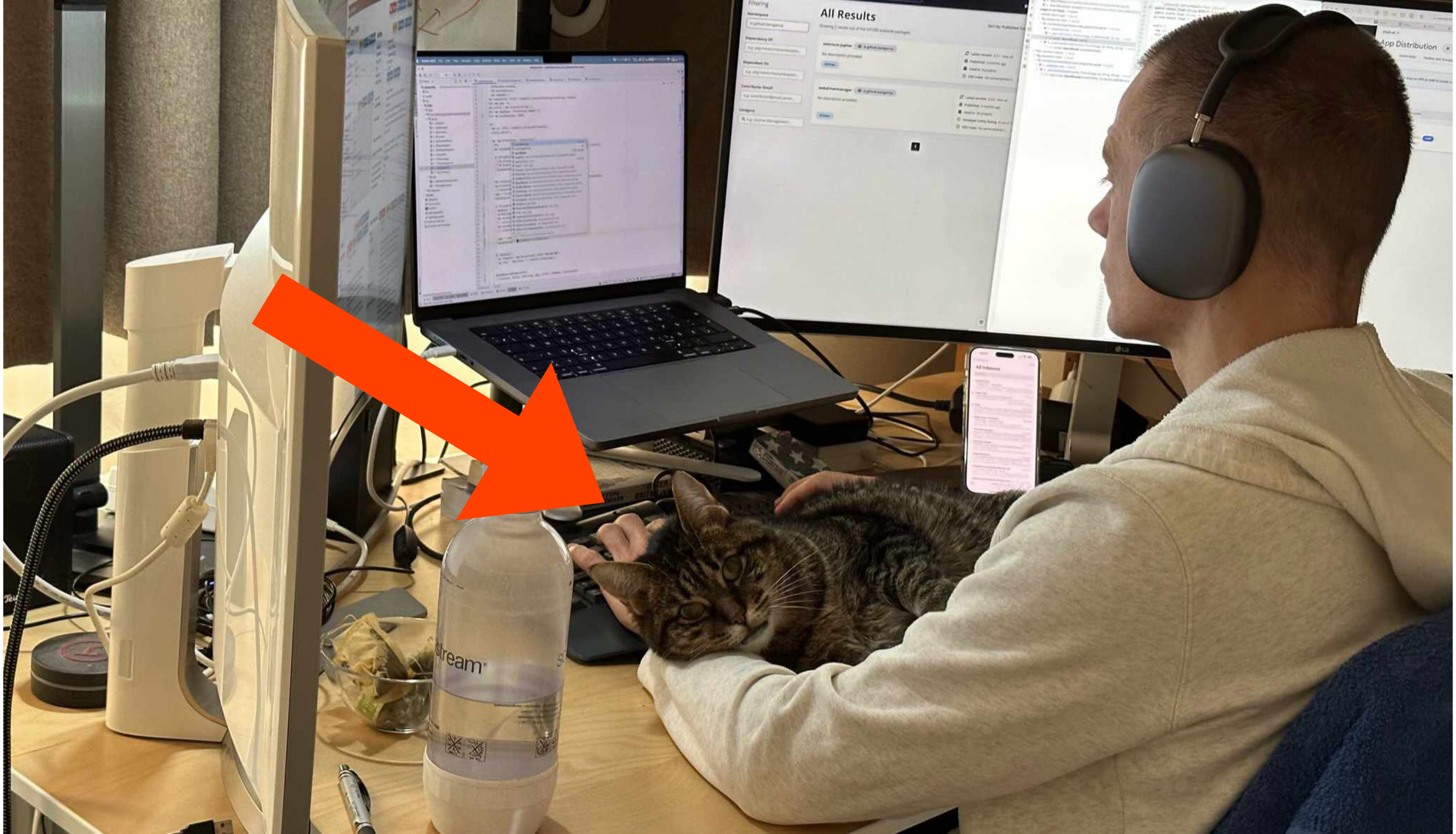
CWA?



CZECH WRESTLING ASSOCIATION?



CAT WRITING ASSOCIATION?


All Results
App Distribution
stream

CLOSED WORLD ASSUMPTION!

CWA=

CONSTRAINT

CLOSED-WORLD ASSUMPTION: AT BUILD TIME

ALL CODE KNOWN

OUR APPLICATION, THE FRAMEWORK & THE JDK

INCLUDE ALL RESOURCES

ALL PROPERTY FILES & RESOURCE BUNDLES WE WANT TO LOAD

WHY IS THAT A CONSTRAINT?

NATIVE JAVA CONSTRAINTS

JAVA FEATURES IMPOSSIBLE

CONSTRUCT NEW CLASSES & METHODS AT RUNTIME

LOAD ARBITRARY BYTECODE

SOME "RUNTIME LINKING & VISIBILITY" SCENARIOS

MORE CUMBERSOME

ENSURE ALL CLASSES (ESPECIALLY IN REFLECTION) ARE INCLUDED

ENSURE ALL RESOURCES ARE INCLUDED


NO NATIVE
MINECRAFT!

STILL THERE

REFLECTION & CLASS LOADING

YES, POSSIBLE IN NATIVE JAVA...

...BUT CONFIGURATION REQUIRED TO INCLUDE CLASSES

VIRTUAL MACHINE

SUBSTRATE VM: GARBAGE COLLECTION, THREAD MANAGEMENT

NO JIT COMPILER!

HOW DOES NATIVE IMAGE WORK?

REGULAR JAVA COMPILER

PARSING

OPTIMIZATION

LEXICAL ANALYSIS

SEMANTIC ANALYSIS

BYTECODE GENER.

1

2

3

4

5

5 STEPS: OK

NATIVE IMAGE: STARTS WITH BYTECODE

BUILD-TIME INITIALIZATION

OPTIMIZATION

POINT-TO-POINT ANALYSIS

LOOP TO 2)

HEAP SNAPSHOTTING

IMAGE GENER.

HINTS

1 2 3 4 5 6 7

NORMAL COMPILATION + 7 STEPS: SLOOOW

HOW SLOW?

MY DEMO MICROSERVICE

WHAT

CONVERTS IMAGES IN CURRENT DIRECTORY TO ONE-PAGE PDFS

USES ITEXT PDF LIBRARY

BENCHMARK: 80 PNG, EACH ABOUT 26.5 MB, 2.1 GB IN TOTAL

HOW

TWICE: SPRING BOOT 3 & QUARKUS

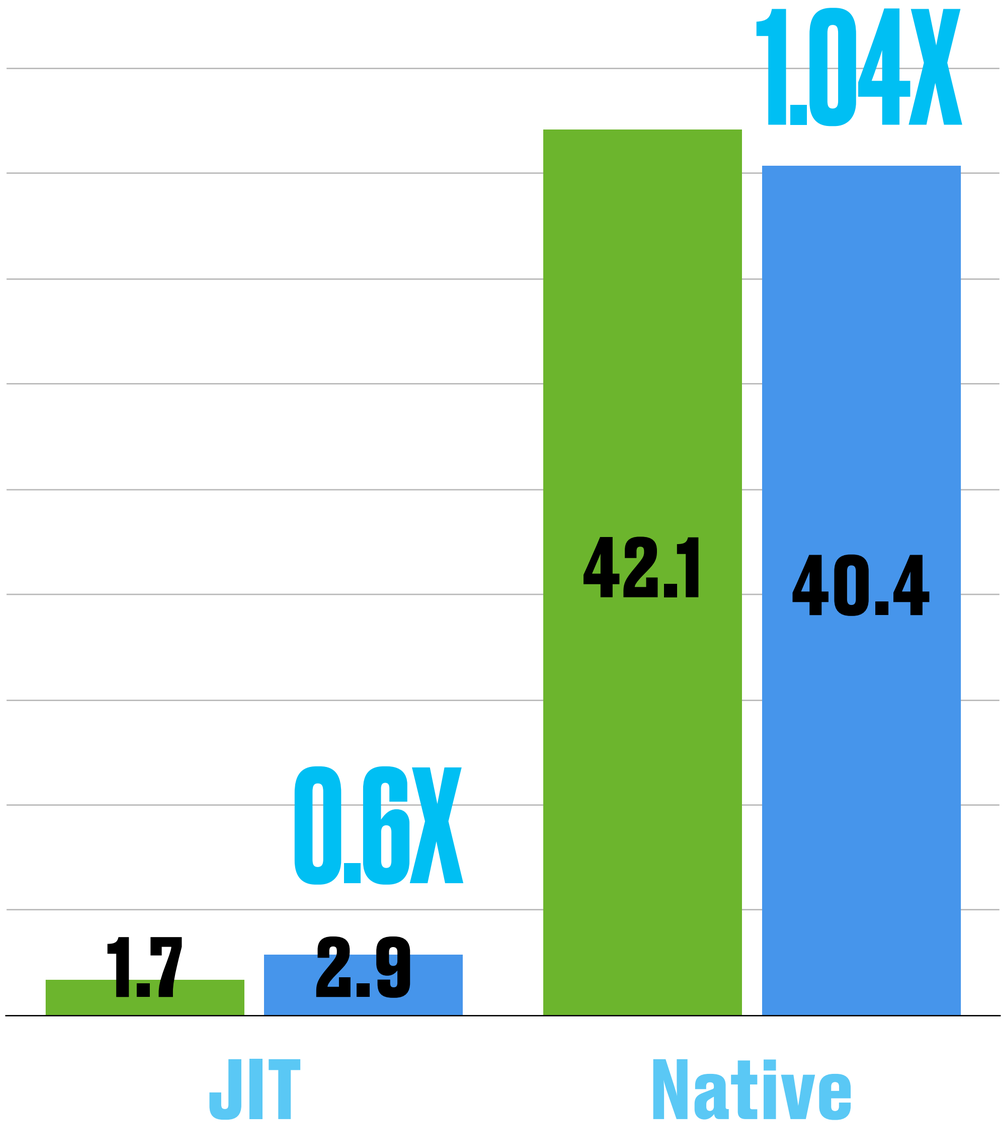
BOTH USE SAME CONVERTER CLASS

REPO LINKS ON TALK PAGE

LOWER = BETTER

IN ALL CHARTS

SPRING BOOT VS. QUARKUS: BUILD TIME (IN SECONDS)



NATIVE IMAGE BUILD TIME: 25X OF JIT JAVA WITH SPRING BOOT!

IT ISN'T AS BAD AS IT LOOKS

NATIVE JAVA DEVELOPMENT

MOST OF THE TIME

DEVELOPERS CODE & DEBUG
WITH JIT JAVA IN IDE

CI/CD PIPELINE BUILDS NATIVE
JAVA EXECUTABLES

SOMETIMES

DEVELOPERS BUILD LOCALLY
BEFORE „BIG MERGE“

BUT DEBUGGING NATIVE PROBABLY
REQUIRES NATIVE BUILDS!

MORE ON NATIVE IMAGE COMPILER

TOOLS

I USE GRAALVM JAVA SDK FOR BUILD

NATIVE IMAGE IS AN ADDITIONAL DOWNLOAD THEN

NO CROSS-COMPILATION

ONLY BUILDS NATIVE EXECUTABLE FOR HOST PLATFORM (E.G., FOR WINDOWS ON WINDOWS)

WINDOWS & MAC: USE DOCKER LINUX IMAGE TO BUILD FOR LINUX

ARE YOU OK IF YOUR APP DOESN'T DO CLASS LOADING OR REFLECTION?

NOPE

DEPENDENCIES...

...ARE LIKELY REASON FOR NATIVE JAVA TO BREAK

HOW DEPENDENCIES BREAK NATIVE JAVA

REFLECTION

IF CLASS/METHOD/FIELD ONLY USED VIA REFLECTION, NATIVE IMAGE REMOVES IT

BREAKS AT RUNTIME - MAYBE AFTER DAYS!

FIX: MARK IT FOR INCLUSION

BUILD-TIME INITIALIZATION

SOME CLASS INITIALIZATION BREAKS AT BUILD TIME

NATIVE IMAGE DEFAULTS TO INITIALIZATION AT RUNTIME – BUT QUARKUS TO BUILD TIME

HOW TO FIX DEPENDENCIES?

YOU

- WAIT FOR FIXED DEPENDENCY
- CONFIGURATION HINTS FOR GRAALVM NATIVE IMAGE
- REPLACE DEPENDENCIES

ECOSYSTEM

- "REACHABILITY REPOSITORY" ON GITHUB COLLECTS CONFIGURATION HINTS FROM COMMUNITY
- FRAMEWORK-SPECIFIC CONFIGURATION FEATURES

HOW THIS BREAKS - AND HOW TO FIX IT

QUARKUS: BREAKS

```
return Response.ok(
  jsonParser.fromJson(
    jsonInput,
    Person.class)
  ).build();
```

RETURN PERSON IN HTTP RESPONSE

BREAKS IF PERSON CLASS ISN'T USED ANYWHERE ELSE!

QUARKUS: FIX

```
[
  {
    "name" : "com.acme.Person",
    "allDeclaredConstructors" : true,
    "allPublicConstructors" : true,
    "allDeclaredMethods" : true,
    "allPublicMethods" : true,
    "allDeclaredFields" : true,
    "allPublicFields" : true
  }
]
```

NATIVE IMAGE CONFIG FILE REFLECTION-CONFIG.JSON

SPRING BOOT: BREAKS

```
var result = objectMapper
   .readValue(json,
     new TypeReference
       <List<Customer>>() {}
   );
result.forEach(
   System.out::println
);
```

ITERATE OVER CUSTOMER LIST AND CALL TOSTRING()

BREAKS IF CUSTOMER CLASS ISN'T USED ANYWHERE ELSE!

SPRING BOOT: OTHER FIXES

ANNOTATION

```
@TypeHint(
  types = Customer.class,
  typeNames = "com.acme.Customer")
@SpringBootApplication
public class Application {
```

METHOD CALLS

```
 @Override
 public void registerHints(
   RuntimeHints hints, ClassLoader loader) {
   hints.reflection().registerType(Customer.class,
     MemberCategory.values());
```

USE A FRAMEWORK

FRAMEWORKS WITH GRAALVM PRODUCTION SUPPORT

WHAT FRAMEWORKS DO

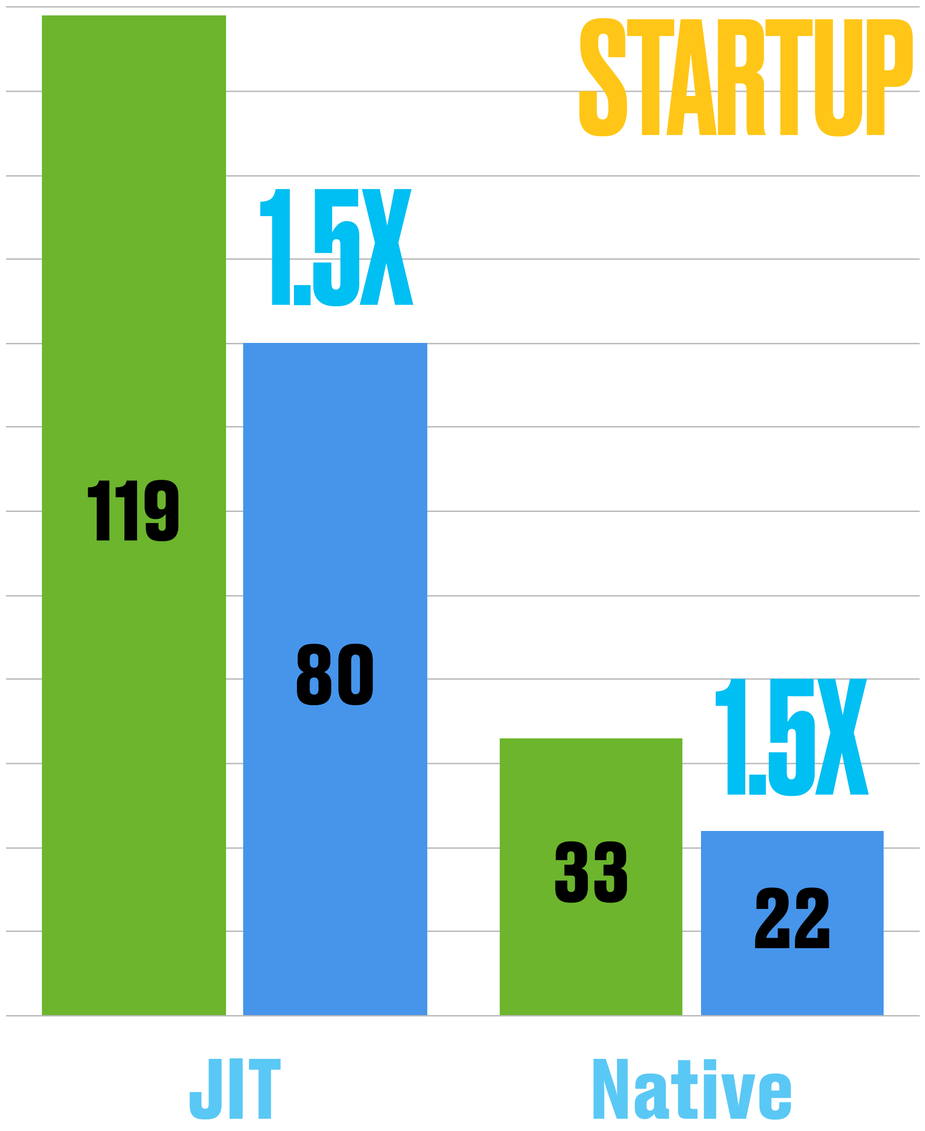
- CONFIGURE NATIVE IMAGE & LIBRARIES
- SHIP WITH NATIVE-READY LIBRARIES
- MAY HAVE ADDITIONAL FEATURES (LIKE HOT CODE RELOAD WITH QUARKUS)

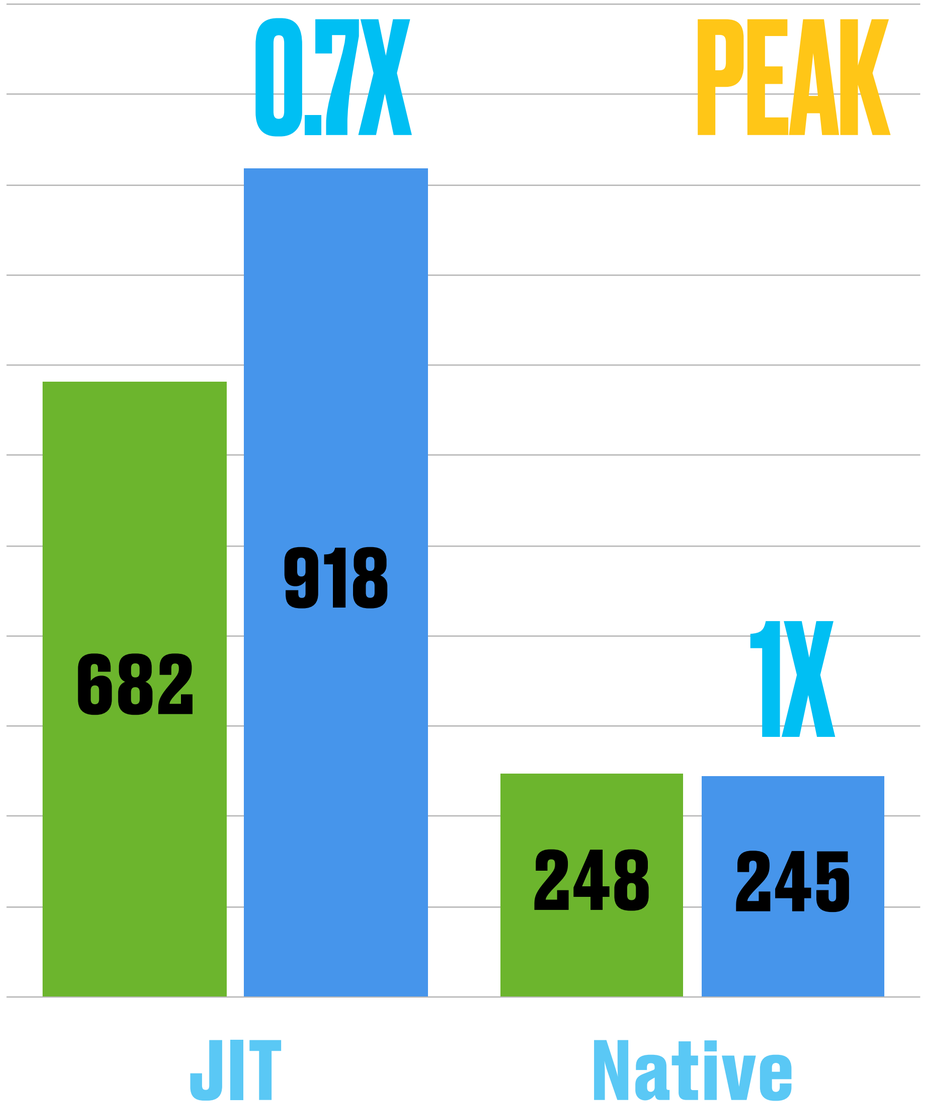
WHICH FRAMEWORKS?

- NEW: QUARKUS & MICRONAUT
  - DON'T USE HELIDON, NOBODY DOES
- OLD: SPRING BOOT 3.0 (NOV 2022)
- JAKARTA EE: NO

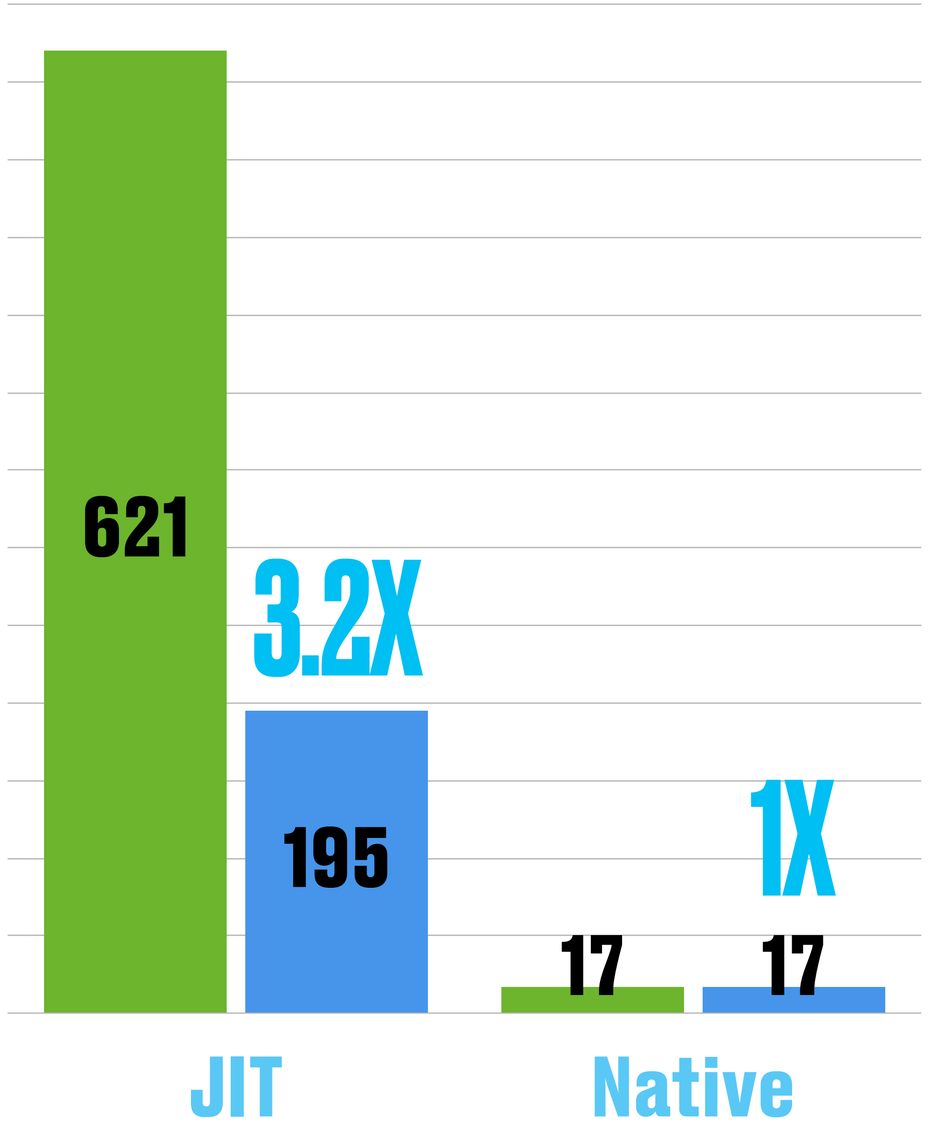
SPRING BOOT 3 VS. QUARKUS

SPRING BOOT VS. QUARKUS: MEMORY (MB)

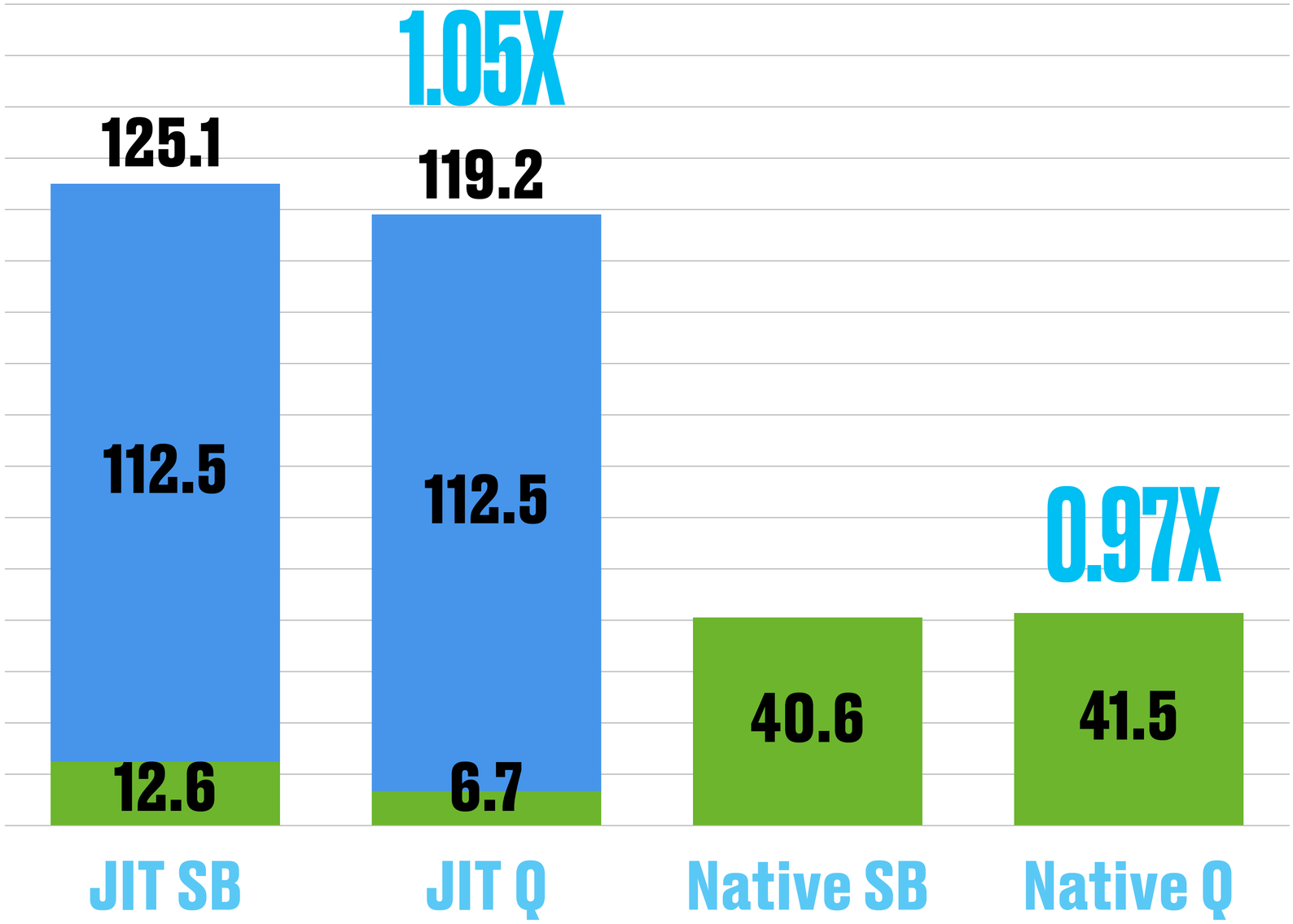




SPRING BOOT VS. QUARKUS: STARTUP TIME (MS)



SPRING BOOT VS. QUARKUS: FILE SIZE (MB)



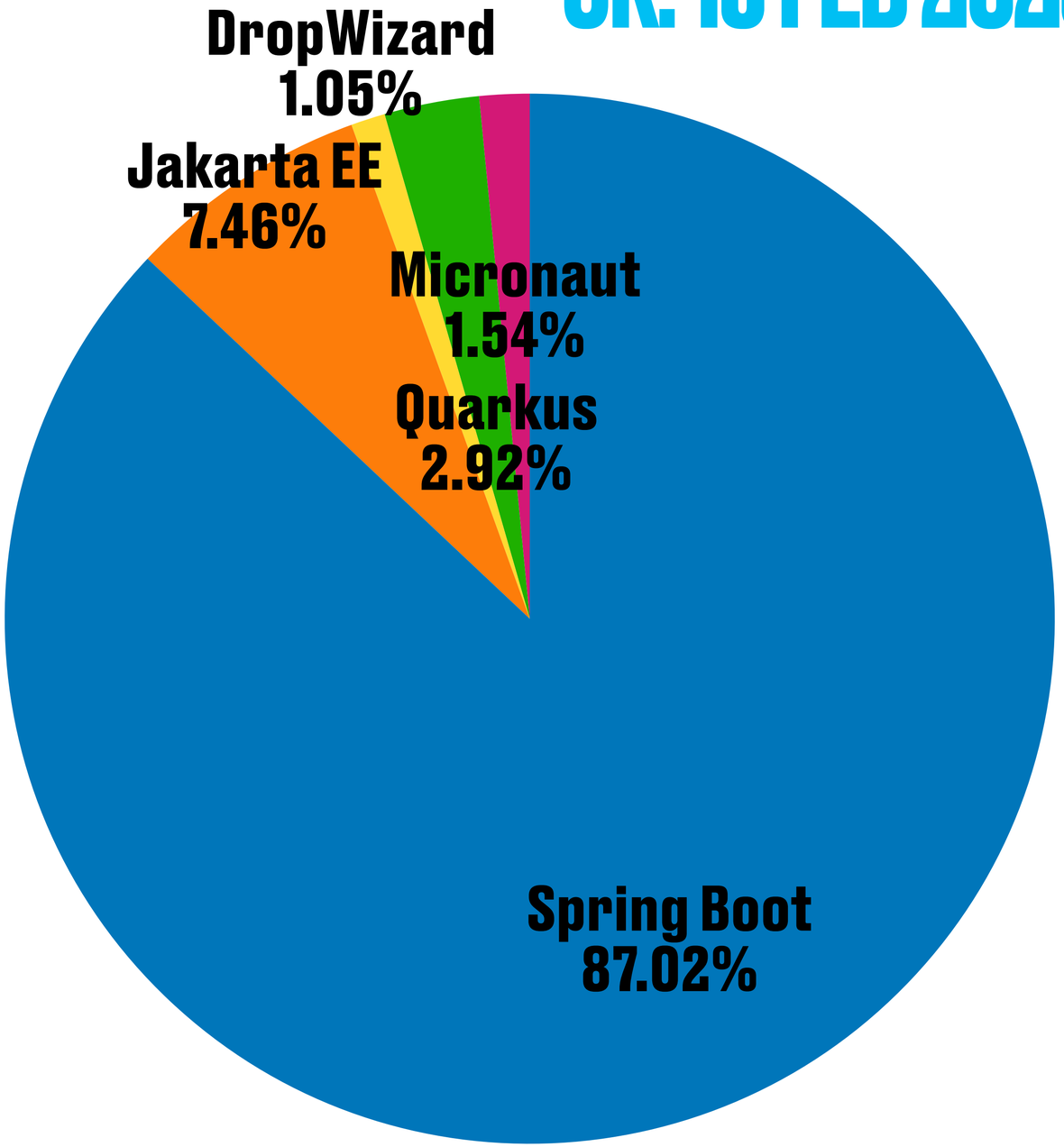
HARDER TO LEARN & USE, LOW JOB DEMAND

HERE'S THE DATA

JOB DEMAND: COUNT MENTIONS IN JOB ADS IN 59 COUNTRIES AT INDEED.COM

NEW FRAMEWORK SHARE GROWS OCT 2021 – DEC 2022: 2.5% => 5.9%

UK: 15 FEB 2023



ZERO MENTIONS OF HELIDON

NEW FRAMEWORK SHARE: 4.5%

NOT ALL NEW FRAMEWORK USERS DEPLOY NATIVE, BUT SOME SPRING BOOT USERS DO

MY NATIVE JAVA JOB DEMAND ESTIMATE: 2/3 OF NEW + 10% OF SPRING BOOT = 4% + 8% = 12%

GOOGLE SEARCHES = INITIAL INTEREST

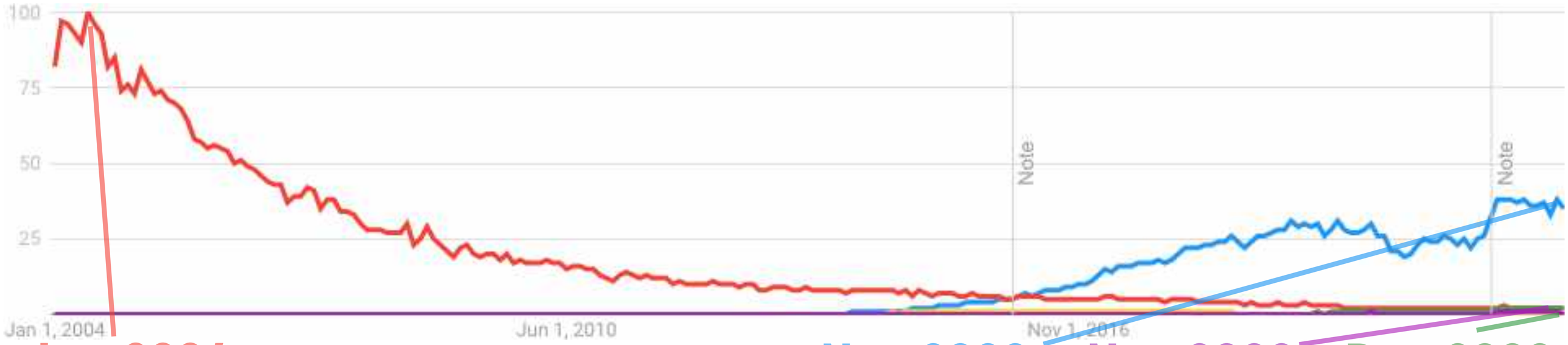
ALL TIME

Micronaut: <1

Jakarta EE: 2

Quarkus: 2

Spring Boot: 35



Jun 2004

100

Nov 2022

38

Nov 2022

1

Dec 2022

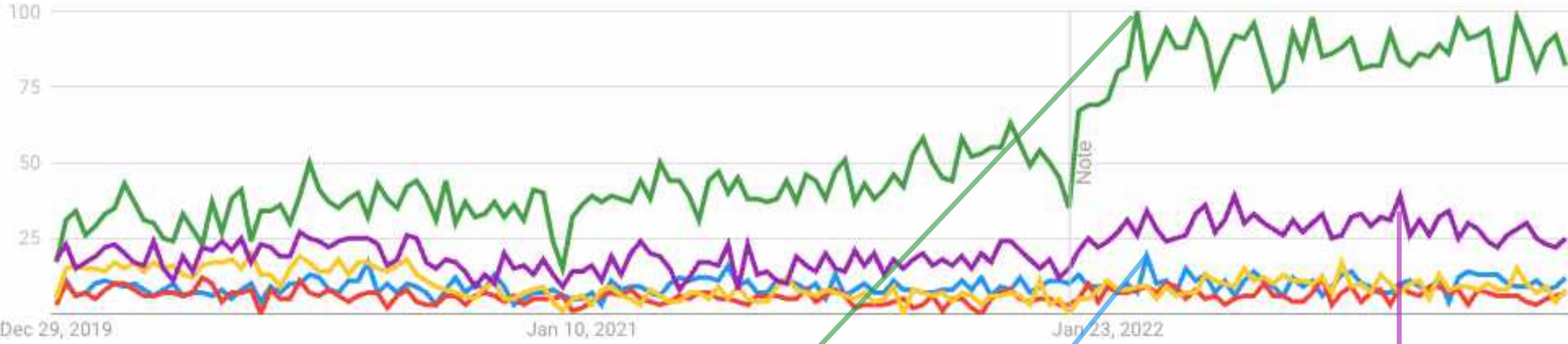
2

NEW FRAMEWORKS, LAST 3 YEARS

Micronaut: 25

Helidon: 12

Quarkus: 82



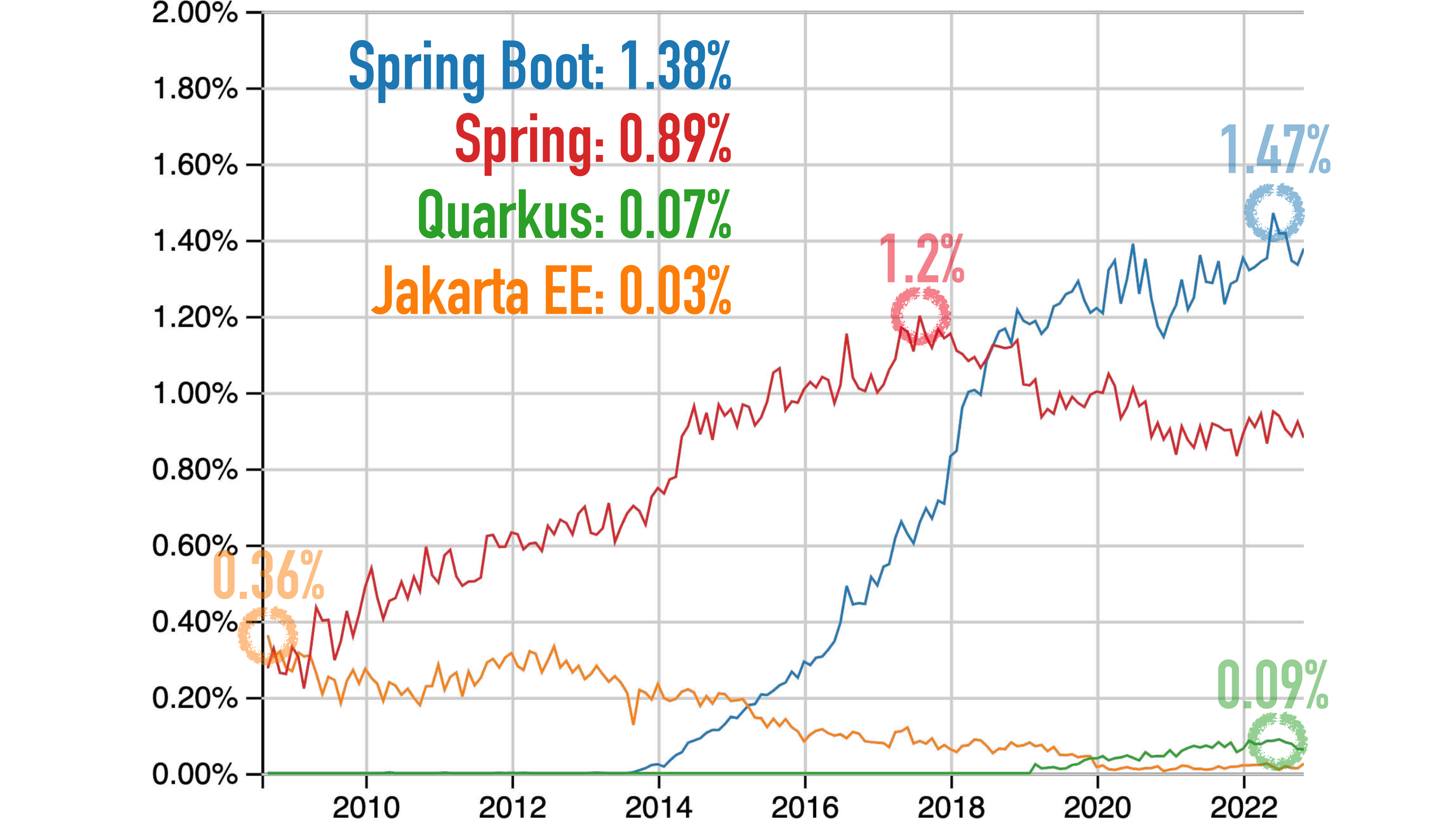
Feb 2022
100

Feb 2022
19

Aug 2022
35

NEW FRAMEWORK INTEREST: <10% OF SPRING BOOT

QUESTIONS AT STACK OVERFLOW = ENGAGEMENT


Spring Boot: 1.38%
Spring: 0.89%
Quarkus: 0.07%
Jakarta EE: 0.03%
2.00%
1.80%
1.60%
1.40%
1.20%
1.00%
0.80%
0.60%
0.40%
0.20%
0.00%
2010
2012
2014
2016
2018
2020
2022
0.36%
1.2%
1.47%
0.09%

NEW FRAMEWORK QUESTIONS: ~7% OF SPRING BOOT

JOB DEMAND FOR NATIVE JAVA INCREASES BUT IS STILL LOW

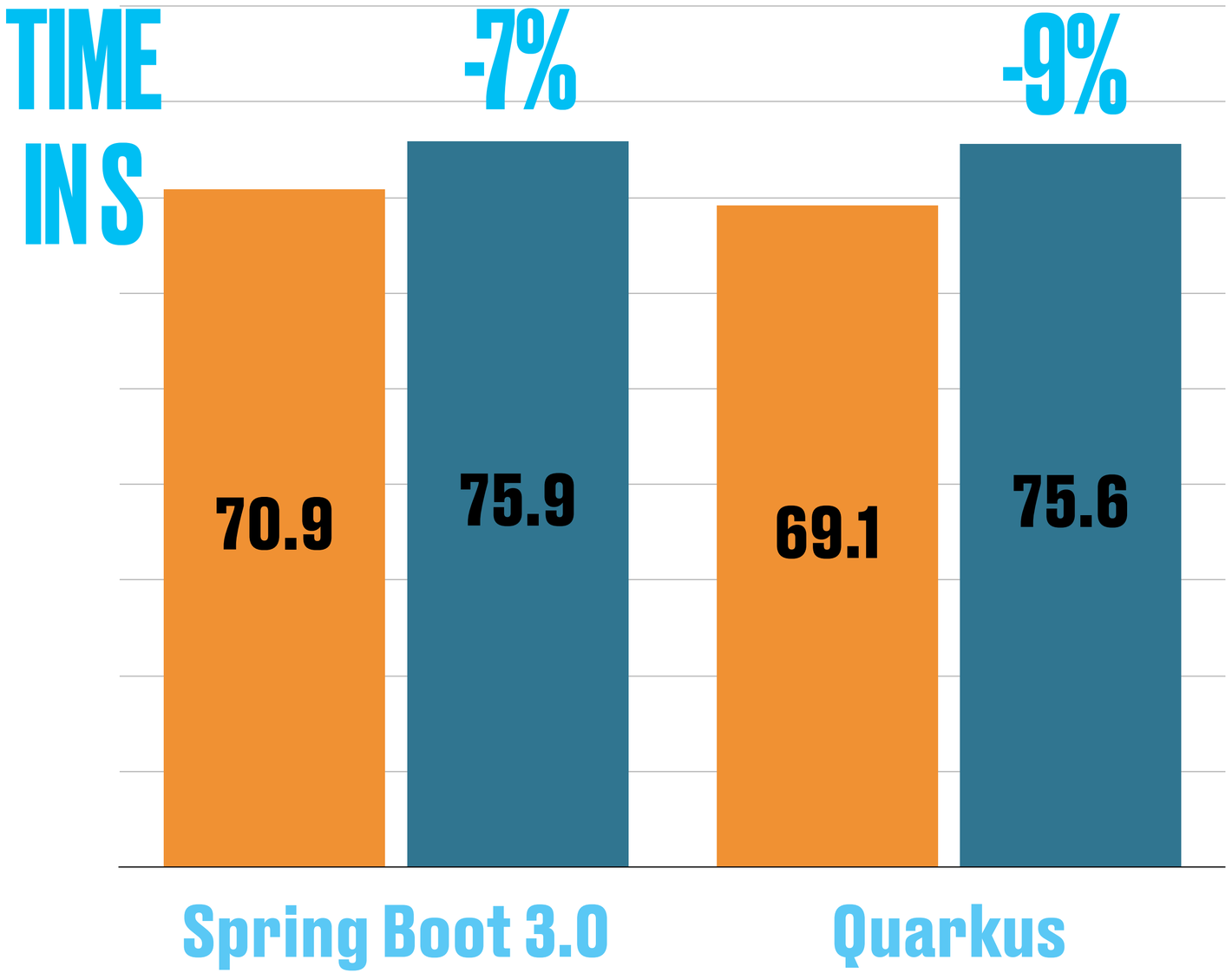
NEW FRAMEWORKS: INITIAL INTEREST & ENGAGEMENT PLATEAUED IN 2022

PEAK PERFORMANCE?

WORSE THAN JIT

– UNLESS YOU PAY

JIT VS. NATIVE: CONVERSION TIME



WHY WORSE?

PEAK PERFORMANCE

GRAALVM COMMUNITY EDITION

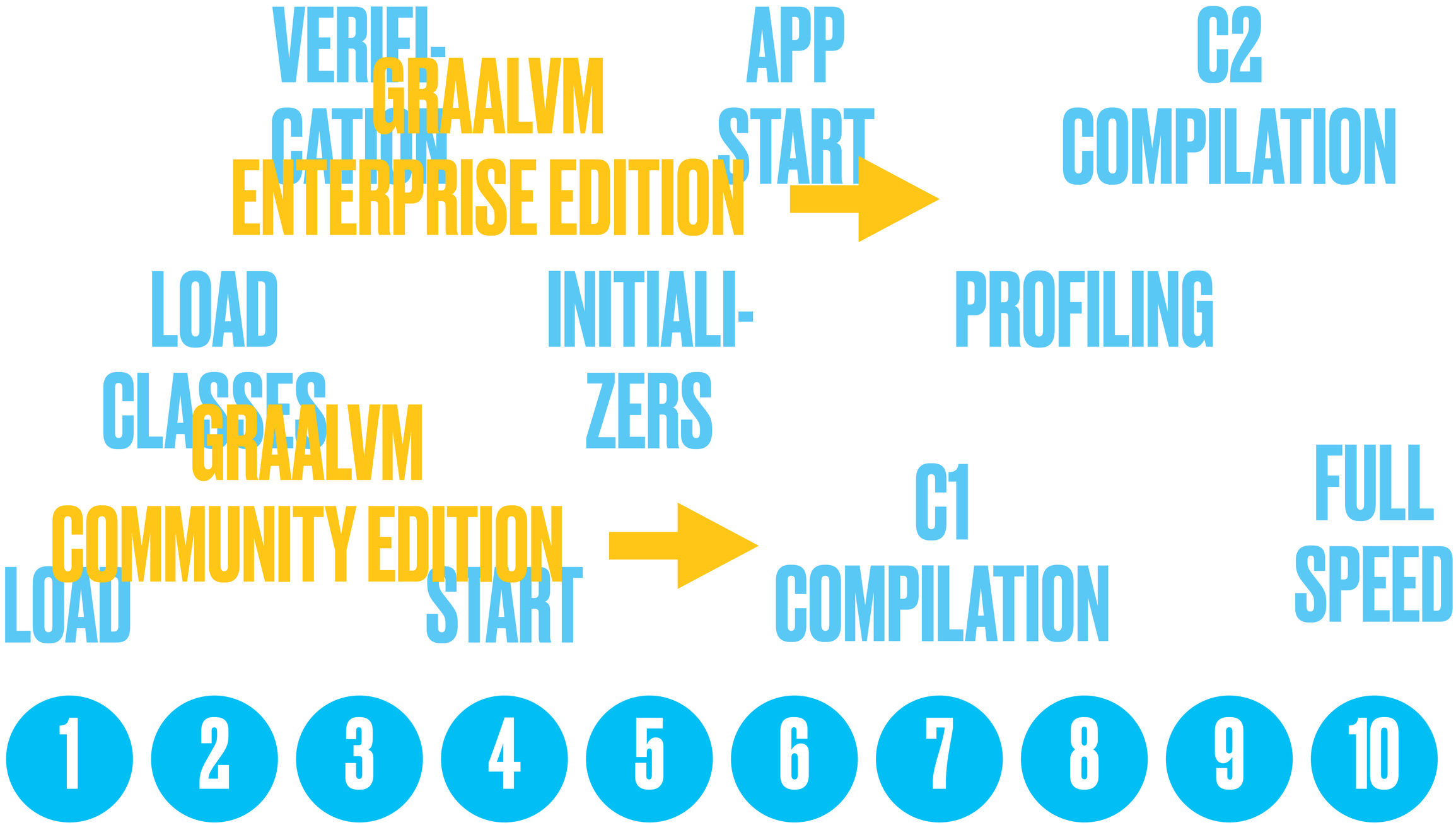
SERIAL GARBAGE COLLECTOR: STOP THE WORLD, SINGLE-THREADED

STANDARD MODE FOR NATIVE IMAGE

GRAALVM ENTERPRISE EDITION: £££

G1 GARBAGE COLLECTOR: DOESN'T STOP THE WORLD, MULTI-THREADED

PROFILE-GUIDED OPTIMIZATION (PGO): AGENT PROFILES AT RUNTIME, NATIVE IMAGE USES THAT


VERIFI-
CATION
GRAALVM
ENTERPRISE EDITION
APP
START
C2
COMPILATION
LOAD
CLASSES
INITIALI-
ZERS
PROFILING
GRAALVM
COMMUNITY EDITION
LOAD
START
C1
COMPILATION
FULL
SPEED
1
2
3
4
5
6
7
8
9
10

MORE CONSTRAINTS

DEBUGGING IS VERY INCONVENIENT

ONLY WHEN RUN IN LINUX

- MAC/WINDOWS: RUN NATIVE IMAGE IN DOCKER LINUX TO BUILD LINUX EXECUTABLE & DOCKER IMAGE
- THAT TAKES VEEERY LONG – AND FOR EVERY CHANGE!

DEBUGGING IN IDE

- EXPERIMENTAL INTELLIJ SUPPORT SINCE JULY 2022
- VS CODE SUPPORT (STABLE?)

WORSE OBSERVABILITY

INCOMPLETE STANDARD SUPPORT

- JXM NOT SUPPORTED AT ALL BUT UNDER CONSTRUCTION
- JAVA FLIGHT RECORDER ONLY PARTIALLY SUPPORTED

POPULAR LIBRARIES DON'T WORK

- MANY USE JAVA AGENTS & RUNTIME INSTRUMENTATION – BOTH DON'T WORK IN NATIVE JAVA
- QUARKUS & SPRING BOOT 3 HAVE THEIR OWN OBSERVABILITY SOLUTION

ECOSYSTEM IMMATURITY

DEMO CALCULATED THUMBNAILS FIRST

NATIVE BUILD FAILED IN QUARKUS BECAUSE THUMBNAIL LIBRARY CLASSES CAN'T BE INITIALIZED AT BUILD TIME (QUARKUS DEFAULT)

FIXED YESTERDAY WITH QUARKUS HINT

MISSING GRAALVM SUPPORT

DEMO COMPILED IN SPRING BOOT...

...BUT LIBRARY USES AWT – DOESN'T WORK NATIVELY ON MAC, SO: PDFS

RELATED: SPRING WEB SERVICES DON'T WORK OUT OF THE BOX (ISSUE WITH FIX)

WHY MORE SECURE?

NATIVE JAVA SECURITY

SMALLER ATTACK SURFACE

REMOVING UNUSED CODE ALSO REMOVES POTENTIAL SECURITY HOLES IN DEPENDENCIES

NO ARBITRARY CODE EXECUTION

MANY JAVA ATTACKS, LIKE LOG4SHELL, GET JAVA TO LOAD AND RUN "ATTACK CODE"

CWA PREVENTS THAT: ONLY RUNS CODE KNOWN AT BUILD TIME

SECTION SUMMARY

NATIVE IMAGE THROWS UNUSED STUFF AWAY & DOES MORE AT BUILD TIME

FASTER STARTUP
LESS MEMORY
SMALLER FILES
BETTER SECURITY

CWA CONSTRAINT MAKES SOME JAVA FEATURES IMPOSSIBLE

DEPENDENCIES MAY BREAK NATIVE JAVA

MUCH LONGER BUILD TIMES, WORSE PEAK PERFOR-MANCE IN FREE VERSION

ALSO WORSE: DEBUGGING, MONITORING, ECOSYSTEM

USE A FRAMEWORK: SPRING BOOT 3.0, QUARKUS, OR MICRONAUT

NATIVE JAVA SOLUTION

AGENDA

~~PROBLEM? AND WHY NOW?~~

~~NATIVE JAVA SOLUTION~~

JIT JAVA SOLUTION

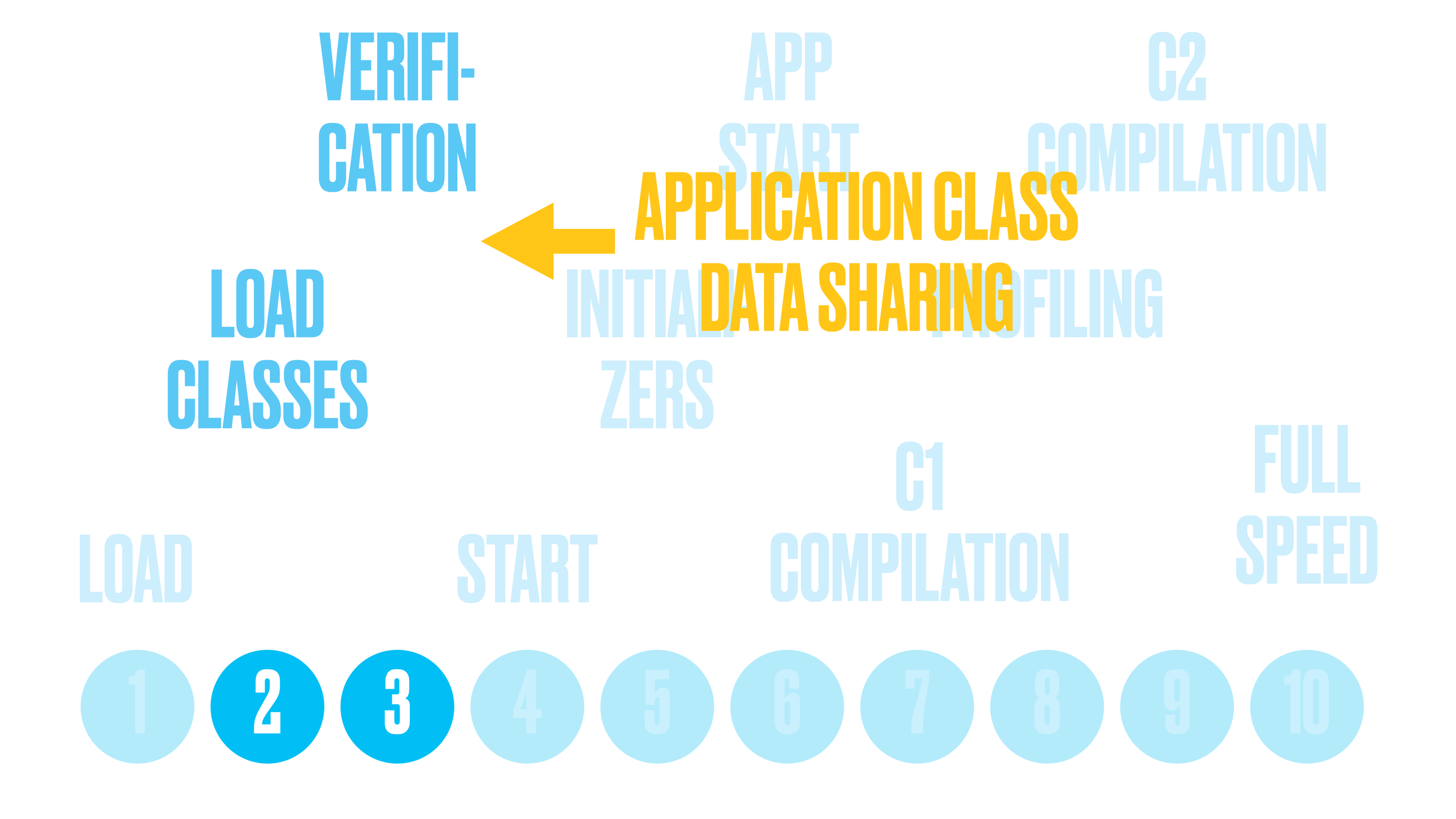
WHEN WORTHWHILE?

JIT JAVA SOLUTION

FASTER STARTUP & LESS MEMORY WITHOUT GRAALVM & CWA?

APPLICATION CLASS
DATA SHARING

JIT JAVA: RUNTIME


VERIFI-
CATION
APP
START
C2
COMPILATION
APPLICATION CLASS
DATA SHARING
LOAD
CLASSES
INITIALIZERS
PROFILING
C1
COMPILATION
FULL
SPEED
LOAD
START
1
2
3
4
5
6
7
8
9
10

APPLICATION CLASS DATA SHARING

WHAT?

LOWER STARTUP TIME: STORE INTERNAL JVM CLASS DATA STRUCTURE AS FILE & LOAD IT ON NEXT RUNS

SAVES MAYBE 10-20% OF STARTUP TIME

HOW?

ENABLED WITH OPENJDK COMMAND LINE PARAMETERS - READY TODAY!

SOME FRAMEWORKS (LIKE QUARKUS) SUPPORT IT, TOO, FOR BIGGER IMPACT

CRAC


NOT THAT
CRACK

OPENJDK PROJECT CRAC: COORDINATED RESTORE AT CHECKPOINT

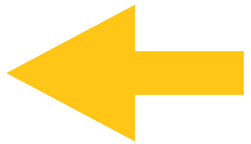
VERIFI-
CATION

APP
START

C2
COMPILATION

LOAD
CLASSES

INITIALI-
ZERS



CRAC

PROFILING

C1
COMPILATION

FULL
SPEED

LOAD

START

1 2 3 4 5 6 7 8 9 10

PROJECT CRAC

WHAT?

LOWER STARTUP TIME: STORE JAVA HEAP WITH "INITIALIZATION RESULTS" AS FILE THE FIRST TIME & LOAD IT ON NEXT RUNS

NEEDS FRAMEWORK SUPPORT: FILES & NETWORK CONNECTIONS CLOSED BEFORE SAVING & REOPENED AFTER LOADING

HOW?

LINUX ONLY: USES CRIU OS FEATURE

APP DETERMINES WHEN SAVE & LOAD

ONLY MICRONAUT PRODUCTION SUPPORTS, AMAZON LAMBDA "SNAPSTART FOR JAVA" USES IT UNDER THE HOOD

CRAC VS. NATIVE IMAGE HEAP SNAPSHOTTING

ADVANTAGE CRAC

- RUN LESS CODE AT RUNTIME: RESULTS FROM ALL INITIALIZERS (NATIVE IMAGE DEFAULTS TO RUNTIME INITIALIZATION) + DYNAMIC INSTANCES
- APPLICATION DETERMINES WHEN TO TAKE SNAPSHOT – STORE EVEN MORE STATE

DISADVANTAGE CRAC

- ONLY MICRONAUT SUPPORTS IT
- HEAP IS SEPARATE FILE (NATIVE IMAGE: PART OF EXECUTABLE)

OPENJDK PROJECT LEYDEN

PROJECT LEYDEN

FIRST PHASE

GOAL: STANDARDIZE NATIVE JAVA

STARTED JUNE 2020 BUT
NOTHING HAPPENED FOR 2 YEARS

SECOND PHASE

GOALS: IMPROVE JIT JAVA &
STANDARDIZE NATIVE JAVA

STARTED AGAIN IN MAY 2022

IN LTS RELEASE: JAVA 25 IN
SEPTEMBER 2025?

SECTION SUMMARY

SOME LOWER JIT JAVA STARTUP TIME WITH APPLICATION CLASS DATA SHARING NOW

HOPEFULLY A LOT LESS STARTUP TIME WITH CRAC IN FUTURE

WAIT & SEE WHAT LEDEN BRINGS

JIT JAVA SOLUTION

AGENDA

~~PROBLEM? AND WHY NOW?~~

~~NATIVE JAVA SOLUTION~~

~~JIT JAVA SOLUTION~~

WHEN WORTHWHILE?

WHEN WORTHWHILE?

NATIVE JAVA:
SPEND DEVELOPER TIME TO SAVE OPERATIONAL COSTS

MY FIVE-STEP PLAN TO INTRODUCE NATIVE JAVA

EVERY STEP IS A "GO/NO GO" DECISION

1. DOES YOUR BOSS CARE?
2. IS THERE AN ROI?
3. WOULD IT WORK?
4. TRY SMALL
5. GO BIG

DOES YOUR BOSS CARE?

DOES YOU BOSS CARE?

YOUR JOB

CREATE BUSINESS VALUE - FEATURES, BUG FIXES

BUSINESS VALUE NOT NECESSARILY EQUALS SAVING COSTS

YOUR BOSS MAY NOT CARE BECAUSE...

FEATURES ARE MORE IMPORTANT NOW

DOESN'T WANT YOU TO DO THE WORK AND OPS TO GET THE CREDIT

DEPARTMENTS PAY IT BUDGET NO MATTER WHAT

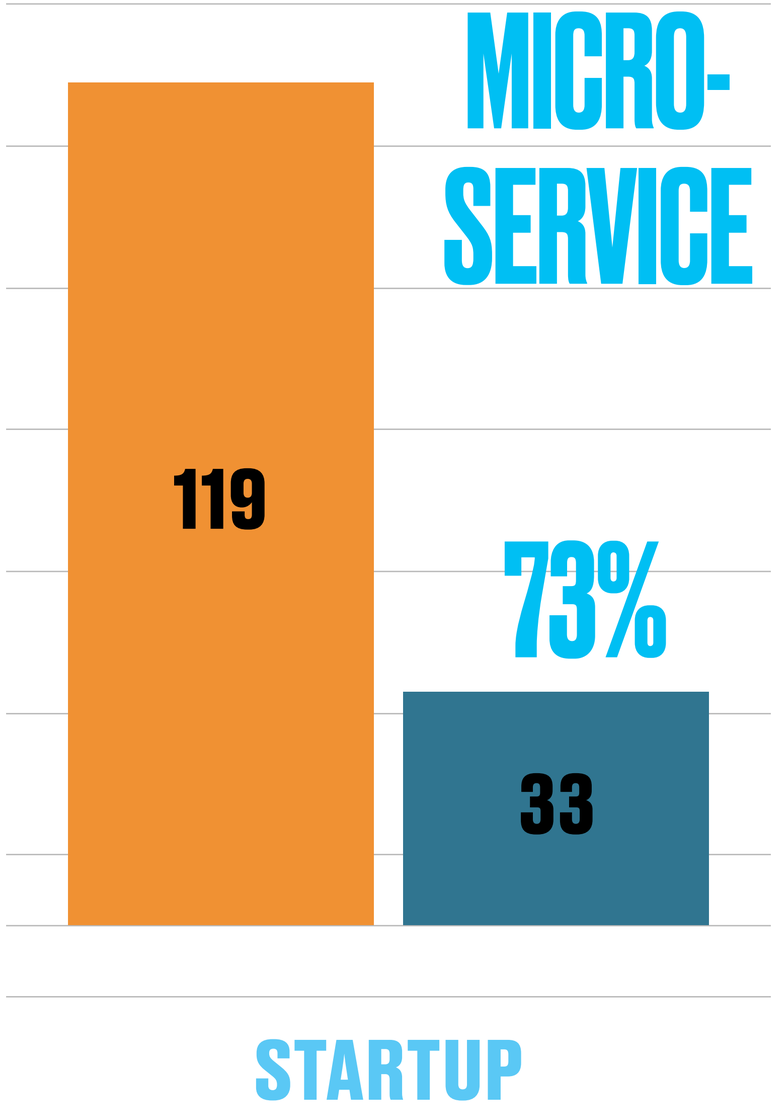
1. DOES YOUR BOSS CARE?
2. IS THERE AN ROI?
3. WOULD IT WORK?
4. TRY SMALL
5. GO BIG

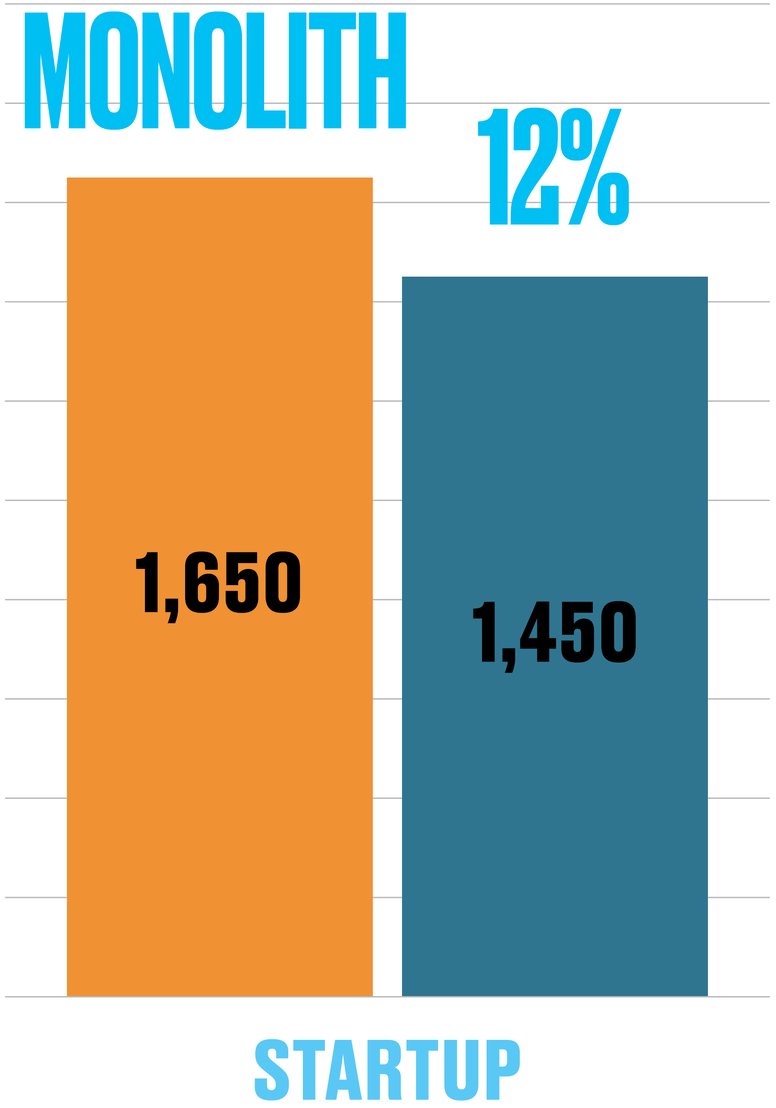
IS THERE AN ROI?

"RETURN ON INVESTMENT" – SHOULD BE POSITIVE AFTER 1-3 YEARS

NATIVE JAVA:
SPEND DEVELOPER TIME TO SAVE OPERATIONAL COSTS

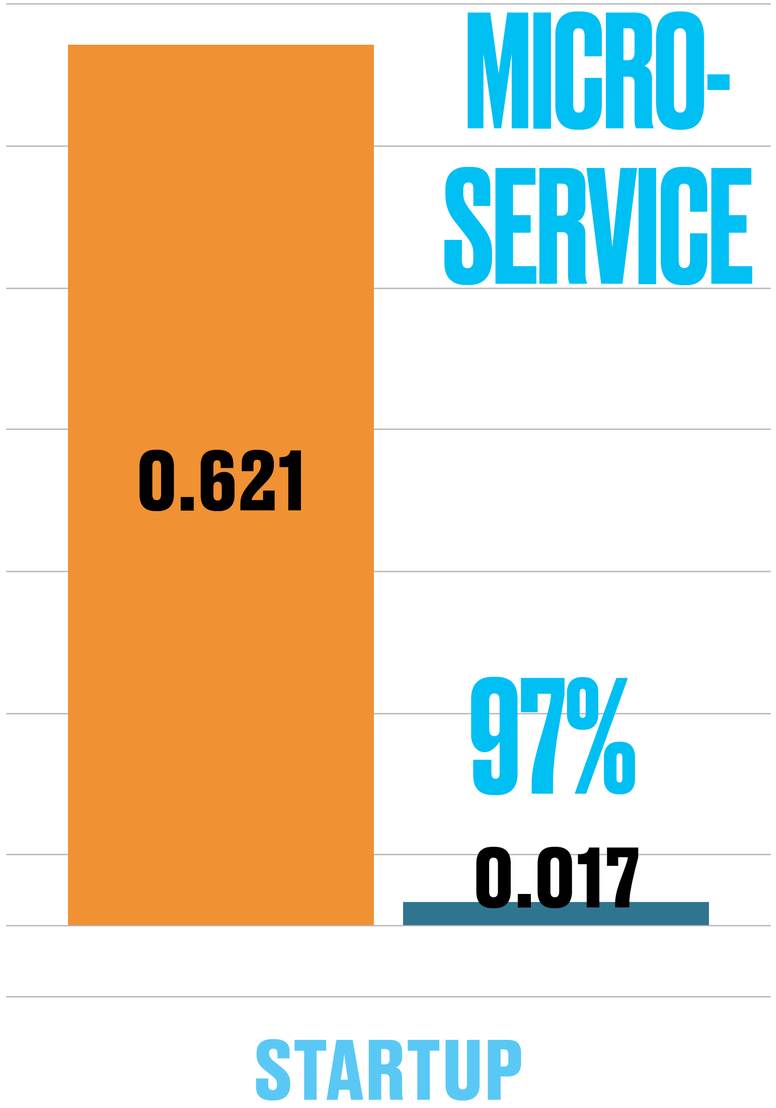
NATIVE SAVINGS: MEMORY

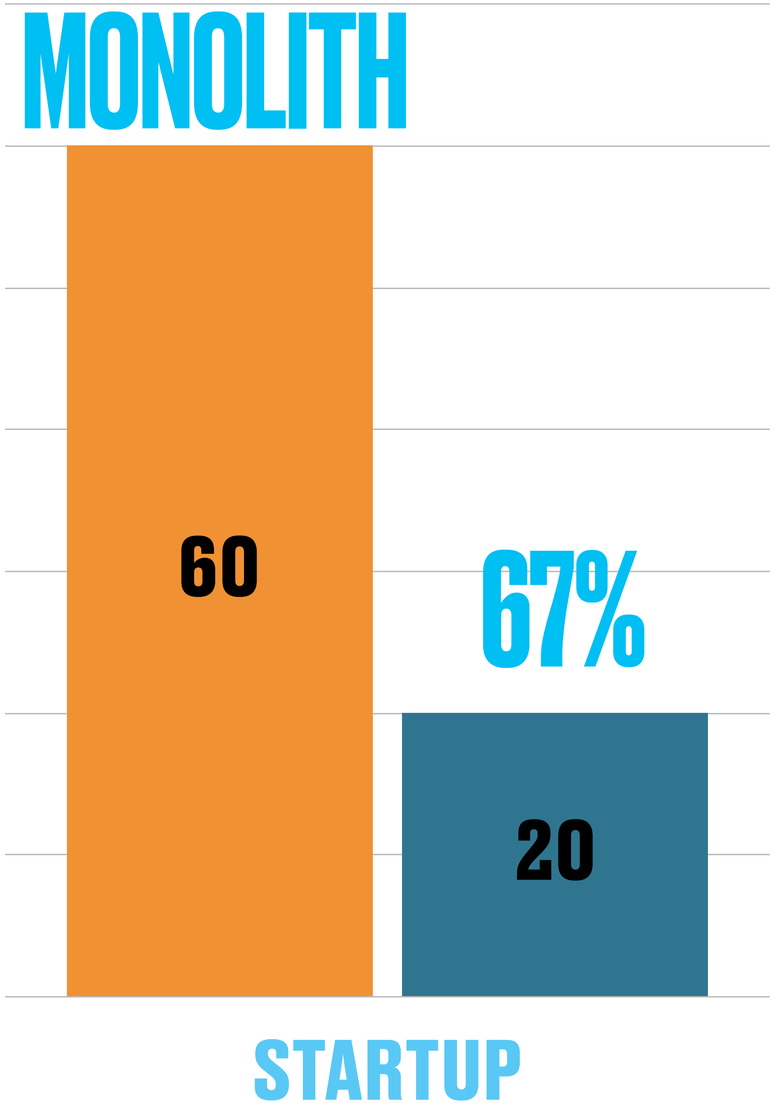




IF YOU PAY FOR SERVER WITH 2 GB RAM, WHAT'S 200 MB MORE HEAP SPACE TO YOU?

NATIVE SAVINGS: STARTUP TIME (IN SECONDS)





WHAT'S 40 SECONDS LESS STARTUP TIME EVERY TWO WEEKS TO YOU?

SWEET SPOT: MICROSERVICES IN KUBERNETES

SAVINGS

INFOQ ARTICLE: SAVED 50% OF KUBERNETES COST BECAUSE MORE PODS IN ONE NODE

IS "50% OF KUBERNETES COST" 10K TO YOU - OR 105M?

CAUTION

WORKS BECAUSE JAVA MICROSERVICES USUALLY WAIT FOR DATABASE AND API CALLS TO COMPLETE

MAY NOT WORK IF YOUR MICROSERVICES ARE CPU-BOUND!

WHAT ARE THE COSTS OF NATIVE JAVA?

COSTS OF NATIVE JAVA

ONE-TIME

NEW FRAMEWORK (QUARKUS) OR UPGRADE (SPRING BOOT 3: JAVA 17, JAKARTA EE 9) - DEPENDENCIES!

TRAIN HOW MANY DEVELOPERS?

DEVELOPER MACHINES, BUILD PIPELINE, CI/CD, OBSERVABILITY, ...

ON-GOING

MORE TIME ON TROUBLESHOOTING: LESS ANSWERS OUT THERE, PROBABLY LESS OBSERVABILITY

DEBUGGING PRODUCTION IS HARDER

SMALLER APP SIZE, SHORTER APP RUNS = HIGHER SAVINGS

YOU STILL ESTIMATE THE ROI – YOU DON'T KNOW

1. DOES YOUR BOSS CARE?
2. IS THERE AN ROI?
3. WOULD IT WORK?
4. TRY SMALL
5. GO BIG

WOULD IT WORK?

WOULD IT WORK?

WHAT YOU KNOW

DOES YOUR APPLICATION DO "FORBIDDEN THINGS" (E.G., CLASS LOADING)?

DOES YOUR OBSERVABILITY WORK IN NATIVE JAVA?

WHAT YOU DON'T KNOW

WILL MY DEPENDENCIES WORK IN JAVA?

1. DOES YOUR BOSS CARE?
2. IS THERE AN ROI?
3. WOULD IT WORK?
4. TRY SMALL
5. GO BIG

TRY SMALL

TRY SMALL

GOT MICROSERVICES

REWRITE ONE/WRITE NEW ONE WITH NATIVE JAVA

SMALL TEAM, LOW COSTS

IS YOUR ROI CALCULATION RIGHT?

GOT MONOLITH

BREAK OFF ONE SMALL PIECE AS MICROSERVICE

THEN: SEE LEFT

1. DOES YOUR BOSS CARE?
2. IS THERE AN ROI?
3. WOULD IT WORK?
4. TRY SMALL
5. GO BIG

GO BIG

GO BIG

ROLL-OUT

CAREFULLY EXPAND NATIVE JAVA FOOTPRINT

ADJUST ALONG THE WAY

DON'T MOVE APPLICATIONS TO NATIVE JAVA IF IT DOESN'T MAKE BUSINESS SENSE

1. DOES YOUR BOSS CARE?
2. IS THERE AN ROI?
3. WOULD IT WORK?
4. TRY SMALL
5. GO BIG

WHEN WORTHWHILE?

AGENDA

~~PROBLEM? AND WHY NOW?~~

~~NATIVE JAVA SOLUTION~~

~~JIT JAVA SOLUTION~~

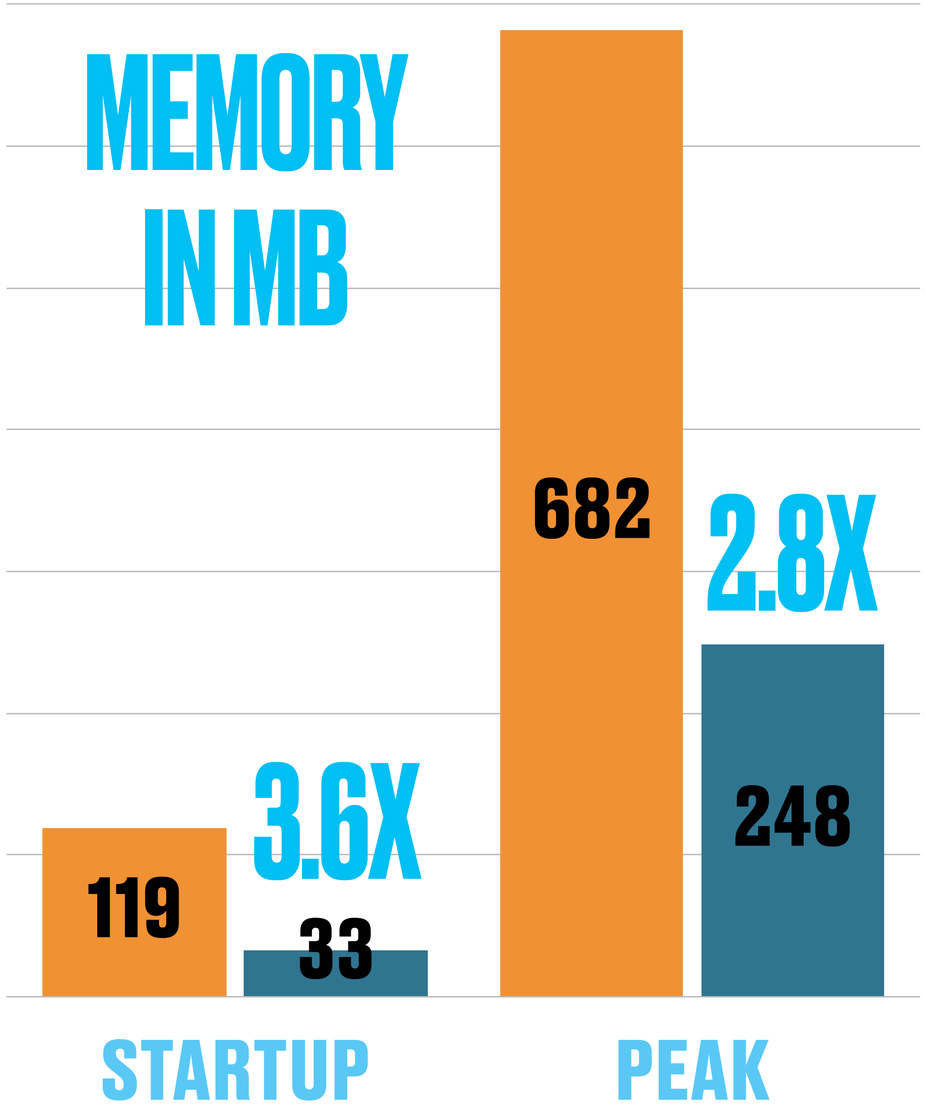
~~WHEN WORTHWHILE?~~

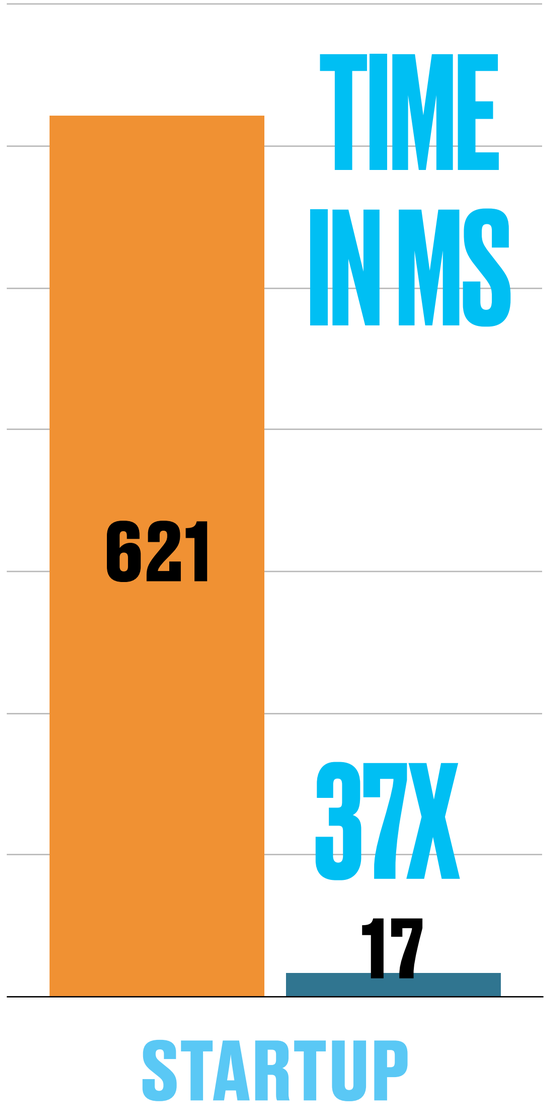
TALK SUMMARY

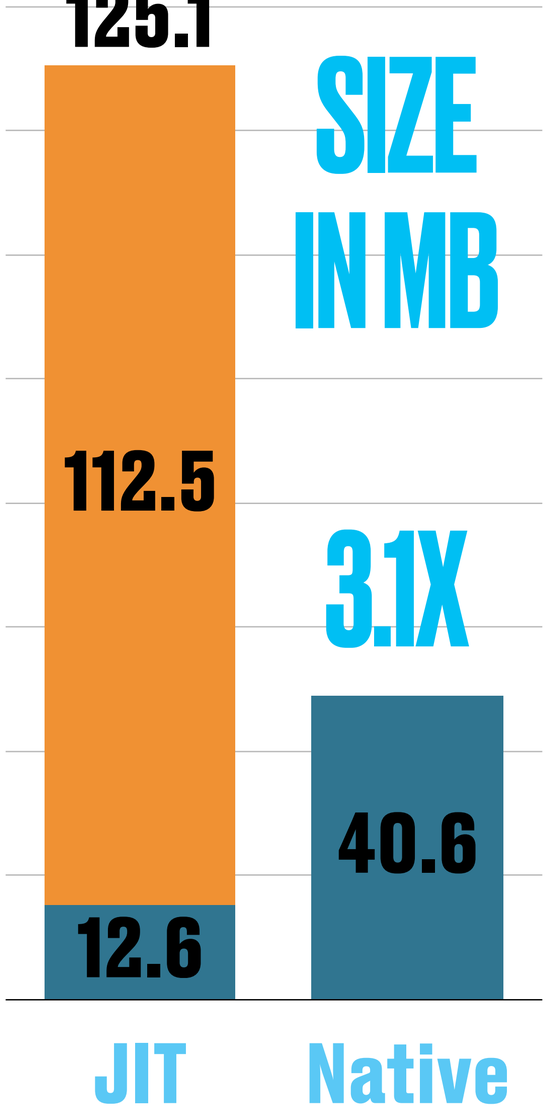
NATIVE JAVA:
SPEND DEVELOPER TIME TO SAVE OPERATIONAL COSTS

FASTER STARTUP
LESS MEMORY
SMALLER FILES
BETTER SECURITY

SPRING BOOT 3: JIT VS. NATIVE (LOWER = BETTER)







SWEET SPOT: MICROSERVICES ON K8S – MORE PODS PER NODE

SOMETIMES DOESN'T WORK

OFTEN MAKES NO BUSINESS SENSE

HARDER TO LEARN & USE, LOW JOB DEMAND


THANK
YOU

QUARKUS TEAM, RED HAT

- BEN EVANS
- DIMITRIS ANDREADIS
- FOLVOS ZAKKAK
- GALDER ZAMARRENO
- HOLLY CUMMINS
- MAX RYDAHL ANDERSEN
- MICHAEL KAM BARBACEC
- PATRICK BAUMGARTNER
- SANNE GRINOVERO

OPENJ9 TEAM, RED HAT

- DAN HEITINGA

GRAALVM TEAM, ORACLE

- ALINA YURENKO

AZUL

- SIMON RITTER


The End

SLIDES

ADDITIONAL INFORMATION

SAMPLE CODE REPOS

BPF.LI/LAD