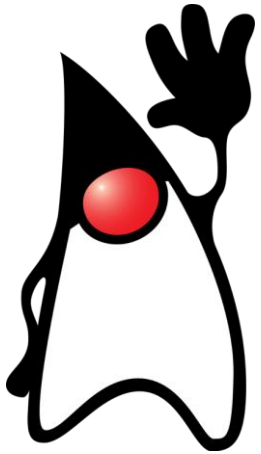
Java at Amazon

Volker Simonis

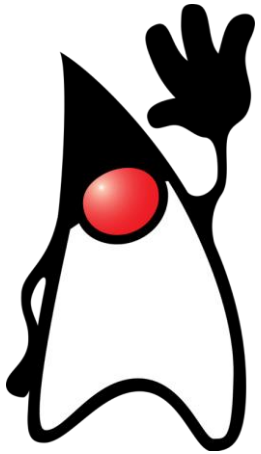
Principal Engineer
Amazon Web Services

How it all started..



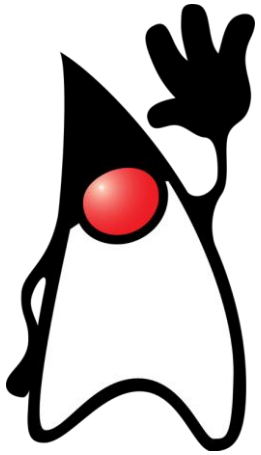
- In 2017 I met this crazy guy (Yishai ☺) at JavaOne and thought..
  - ..do they really want to create yet another OpenJDK distro?

How it all started..



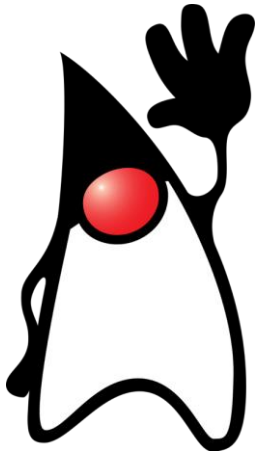
- In 2017 I met this crazy guy (Yishai ☺) at JavaOne and thought..
  - ..do they really want to create yet another OpenJDK distro?
- At FOSDEM 2018 I met Yishai again and I thought..
  - ..this men is quite persistent

How it all started..



- In 2017 I met this crazy guy (Yishai ☺) at JavaOne and thought..
  - ..do they really want to create yet another OpenJDK distro?
- At FOSDEM 2018 I met Yishai agein and I thought..
  - ..this men is quite persistent
- At Devoxx 2018 Yishai & James (the one and only Gosling) announced Amazon Corretto
  - ..and I thought – they're really serious about it!

How it all started..



- In 2017 I met this crazy guy (Yishai ☺) at JavaOne and thought..
  - ..do they really want to create yet another OpenJDK distro?
- At FOSDEM 2018 I met Yishai agein and I thought..
  - ..this men is quite persistent
- At Devoxx 2018 Yishai & James (the one and only Gosling) announced Amazon Corretto
  - ..and I thought – they're really serious about it!
- In 2019 I joined the Amazon Corretto team..

Success and Scale Bring Broad Responsibility

We are big, we impact the world, and we are far from perfect. We must be humble and thoughtful about even the secondary effects of our actions. Our local communities, planet, and future generations need us to be better every day.

Amazon's leadership principles

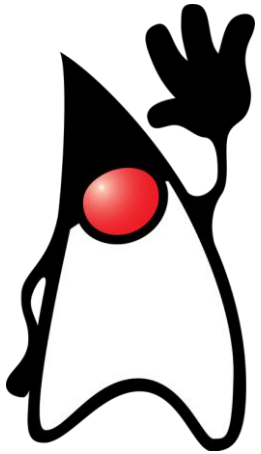
https://amazon.jobs/content/en/our-workplace/leadership-principles

Amazon is a Java shop

- A big amount of internal and external services are written in Java:

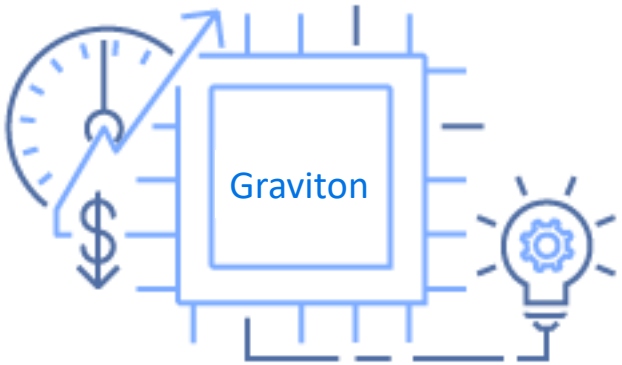
- A lot of customer code that runs on AWS is written in Java:

What is Amazon Corretto?



- Downstream distribution of OpenJDK
- Quarterly security releases
- No-cost long-term releases – JDK 8, JDK 11, JDK 17, JDK 21
- Feature release train (… JDK 22, JDK23 …)
- Drop-in OpenJDK replacement
- Certified using the Java Technical Compatibility Kit (TCK) to ensure it meets the Java SE standard
- Is available on Linux, Windows, and macOS (x86_64 & aarch64)

Corretto runs great on ARM & AWS Graviton processors



AWS Graviton2, Graviton3 & Graviton4

Custom AWS silicon with 64-bit Arm cores

Available in e.g. Amazon EC2, AWS Lambda, AWS Fargate, Amazon Relational Database Service

Up to 40% better price performance over comparable current generation x86-based instances

Up to **60% less energy** for the same performance than comparable EC2 instances

AWS Global Infrastructure Regions & Availability Zones

N AMERICA

- Canada Central 3
- Canada West 3
- GovCloud US-East 3
- GovCloud US-West 3
- Northern California 3
- Northern Virginia 6
- Ohio 3
- Oregon 4
- Mexico

EUROPE

- Frankfurt 3
- Ireland 3
- London 3
- Milan 3
- Paris 3
- Spain 3
- Stockholm 3
- Zurich 3
- Germany

ASIA PACIFIC

- *Beijing 3
- *Ningxia 3
- Hong Kong 3
- Hyderabad 3
- Jakarta 3
- Mumbai 3
- Osaka 3
- Seoul 4
- Singapore 3
- Tokyo 4
- Malaysia
- Taipei
- Thailand

MIDDLE EAST

- Bahrain 3
- Tel Aviv 3
- UAE 3
- Kingdom of Saudi Arabia

S AMERICA

- São Paulo 3

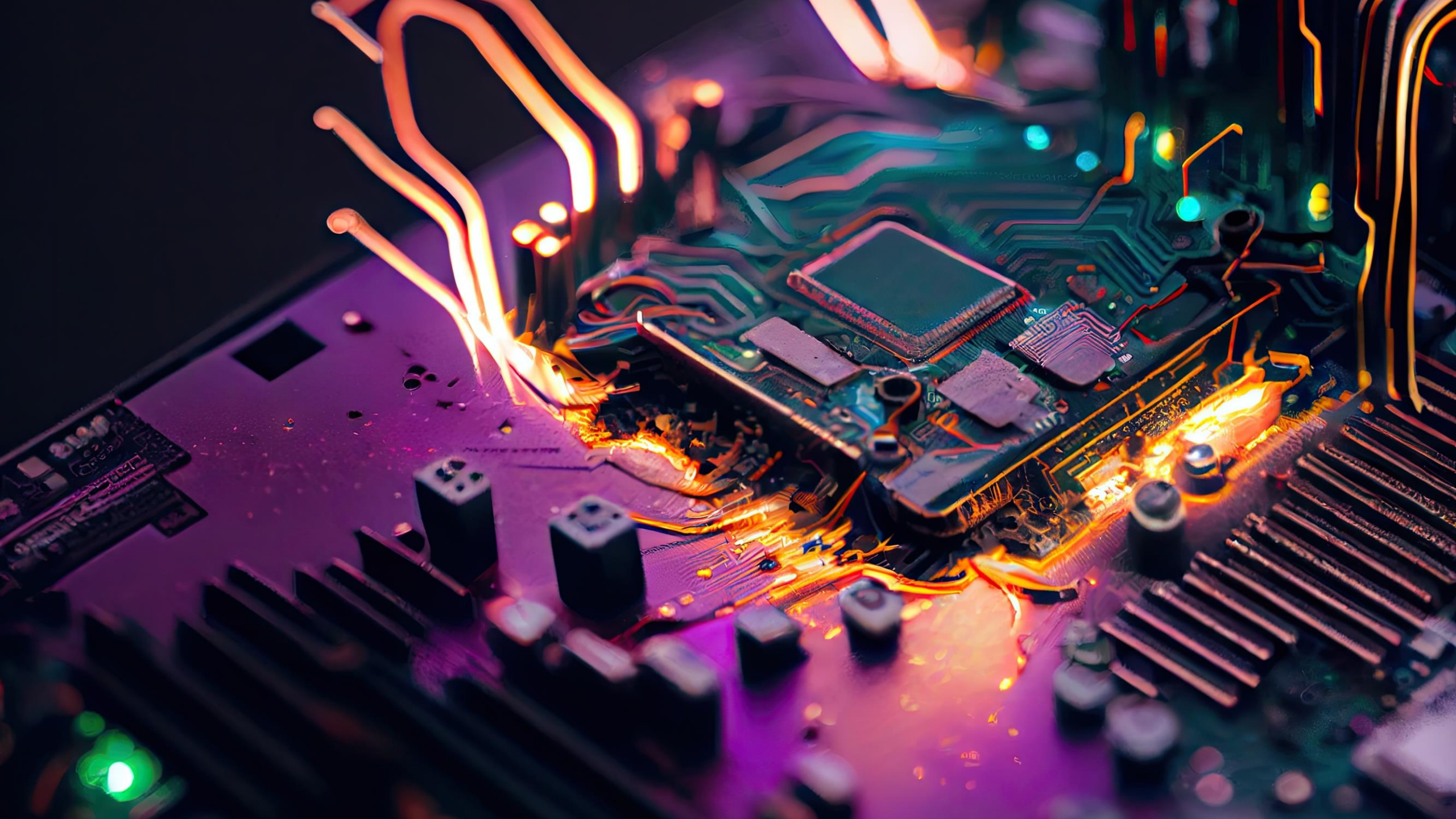
AFRICA

- Cape Town 3

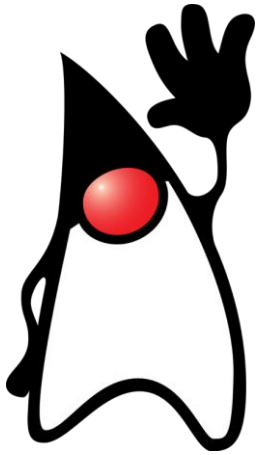
AUSTRALIA & NEW ZEALAND

- Melbourne 3
- Sydney 3
- Auckland

Available Region | Announced | # Availability Zone

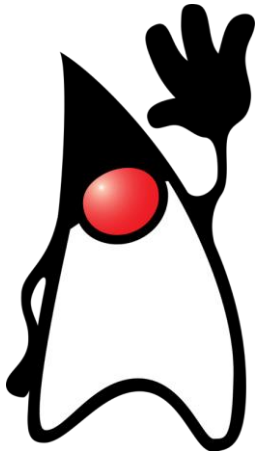


Amazon in the OpenJDK..



- Member of the OpenJDK Vulnerability Group
- Active contributor to the JDK 8u, 11u, 17u and 21u LTS projects
- Maintainer of JMH (Java Micro Harness) and JOL (Java Object Layout)
- Maintainer of Shenandoah and creator of Generational Shenandoah (JEP 404)
- Develops Project Lilliput (i.e. “Compact Object Headers”, JEP 450)
- Contributors to the Leyden and CRaC projects

..and the Java Community

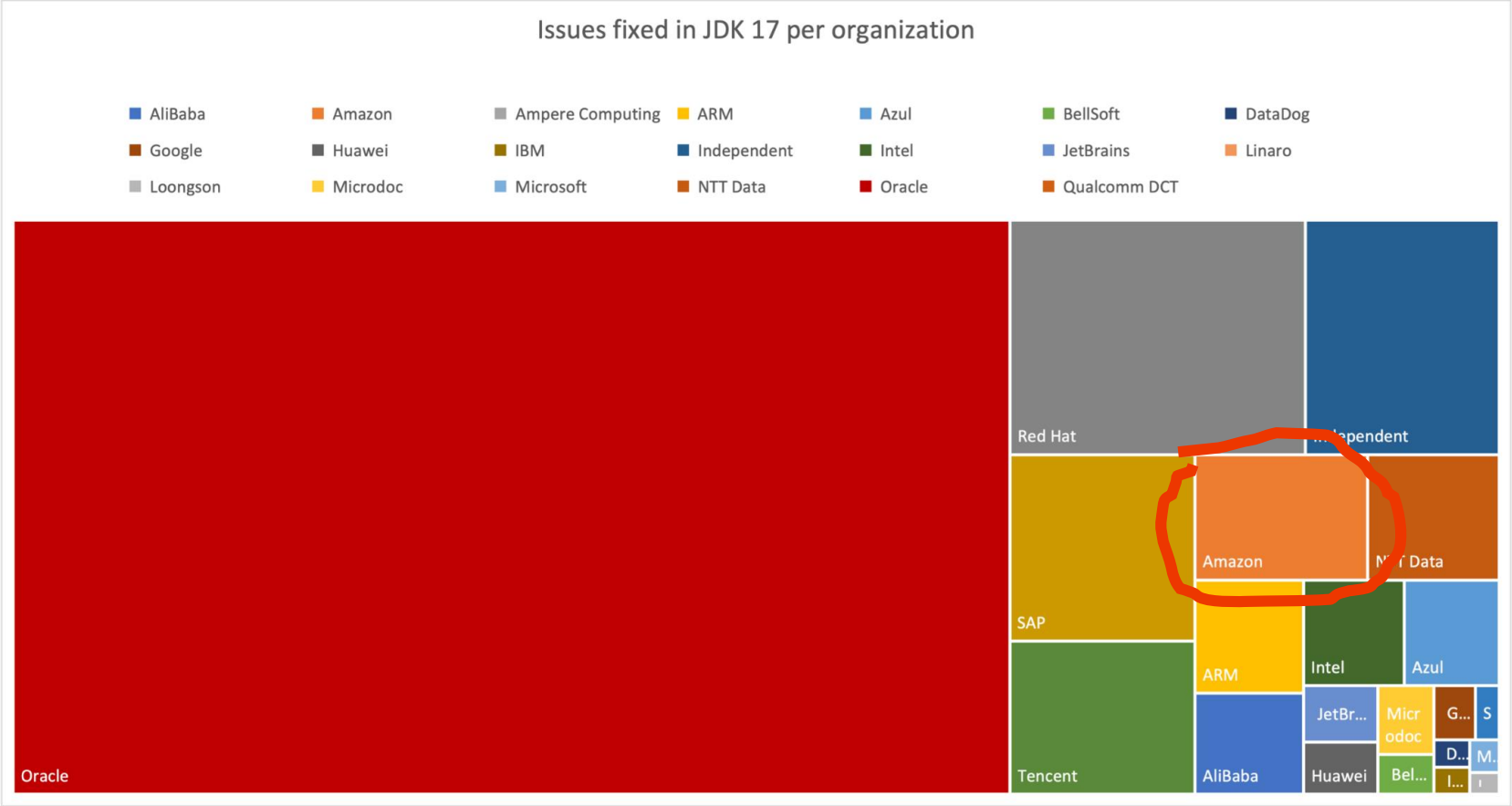


- Maintainer of AsyncProfiler
- Creator of the Heapothesys/Extremem GC benchmark suite
- Creator of the Hotpatch for the Apache Log4Shell vulnerability
- Member of the GraalVM Advisory Board and GraalVM Vulnerability Group
  - Co-maintainer of GraalVM and GraalJS 23.1 for OpenJDK 21 (LTS)

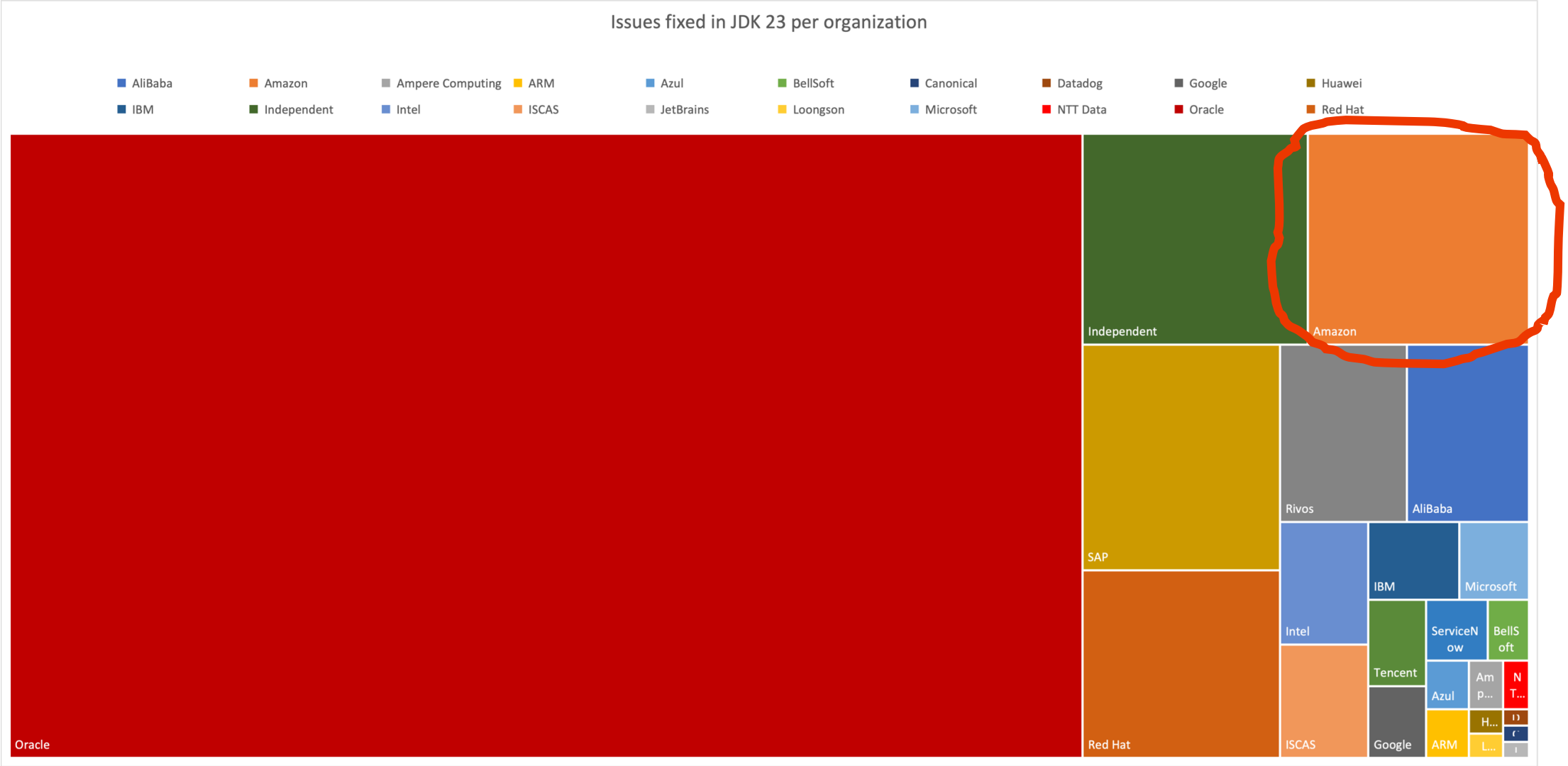
We started small..

We started small..

But constantly getting better..



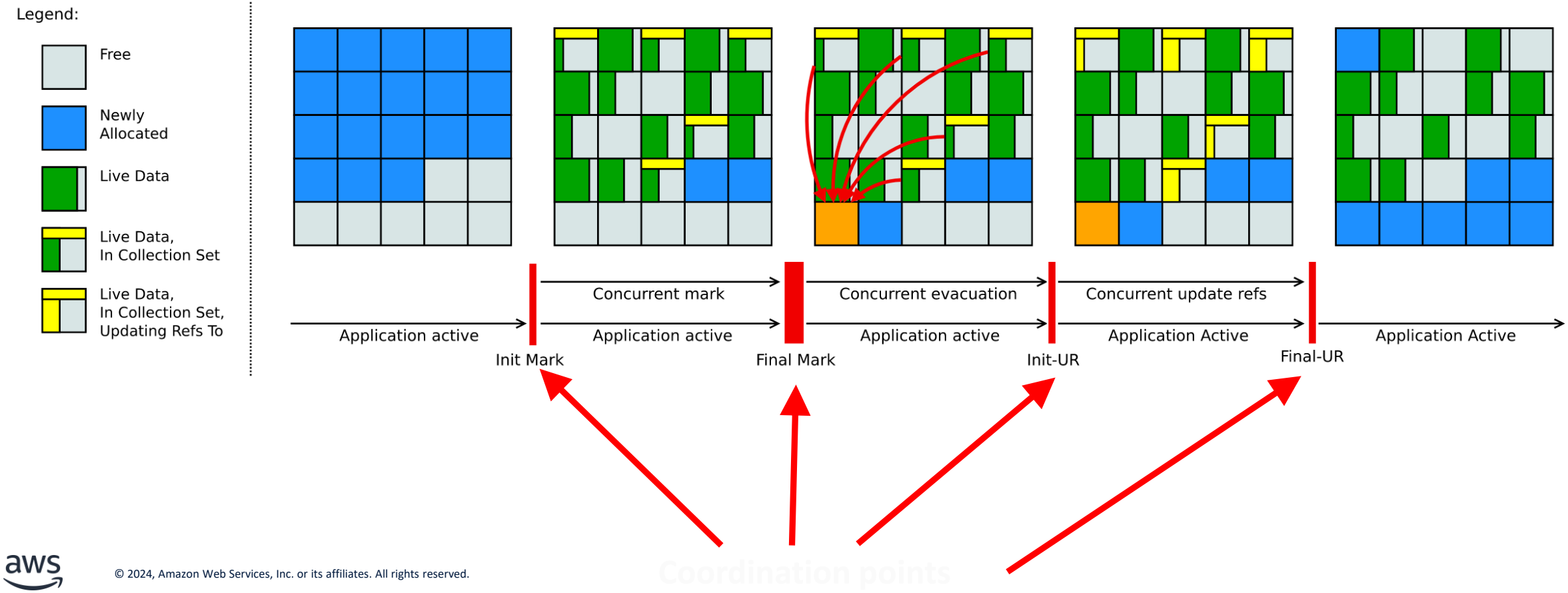
But constantly getting better and better..



Generational Shenandoah

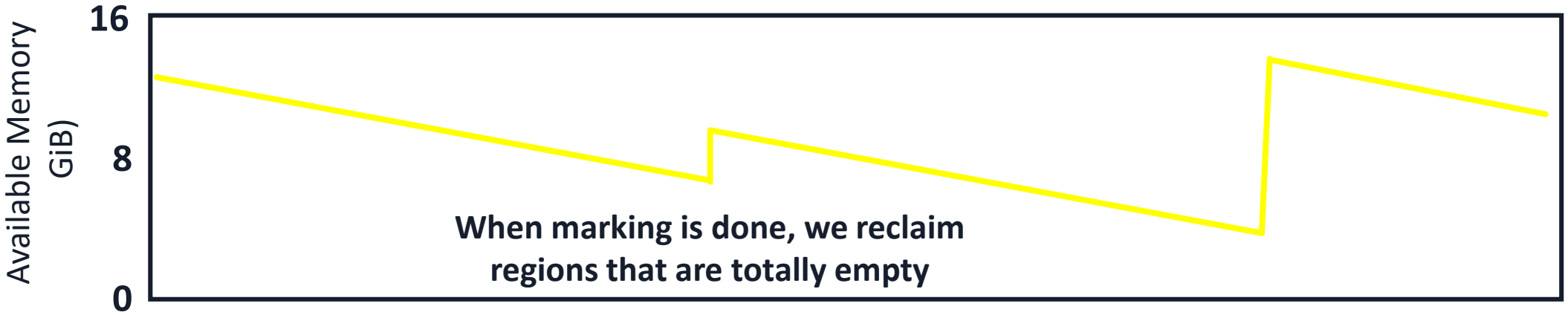
Pauses in Shenandoah GC

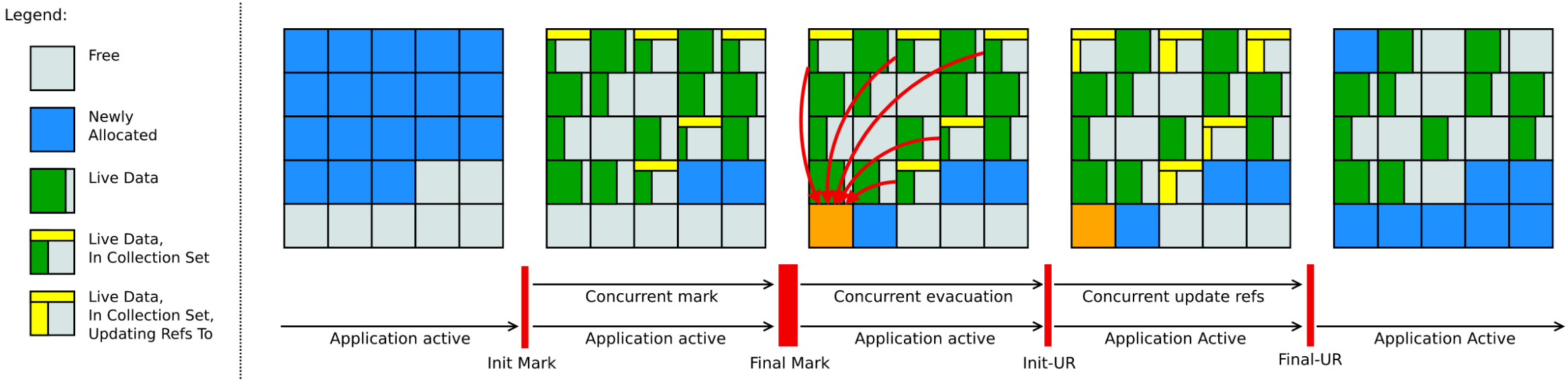
We describe Shenandoah GC as pause-less because there is no stop-the-world phase (except brief coordination points)



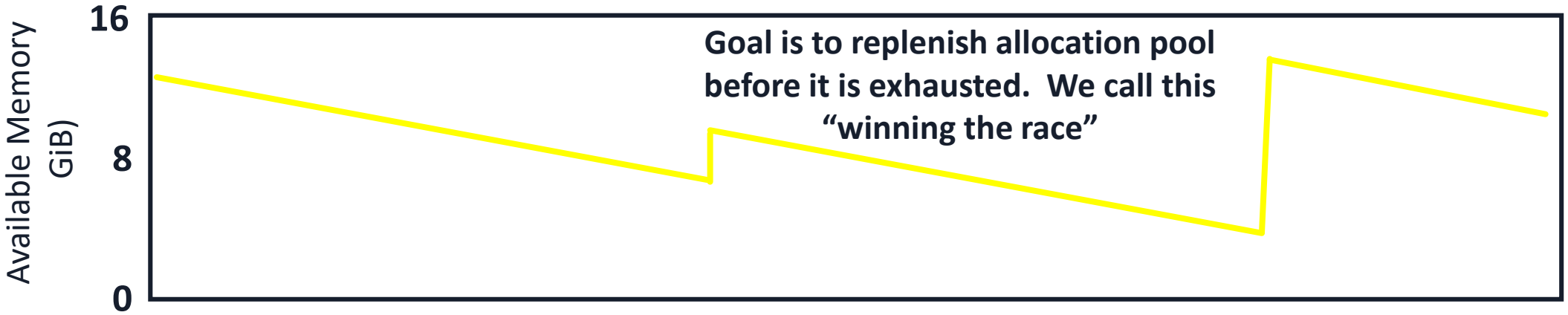
Shenandoah GC Running Well

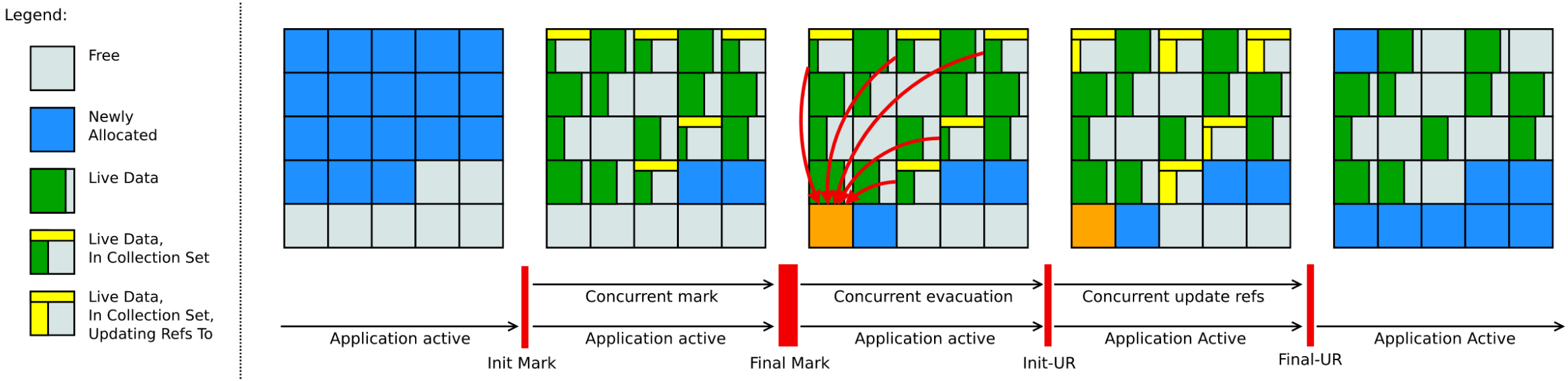
After updating refs, we reclaim all evacuated regions





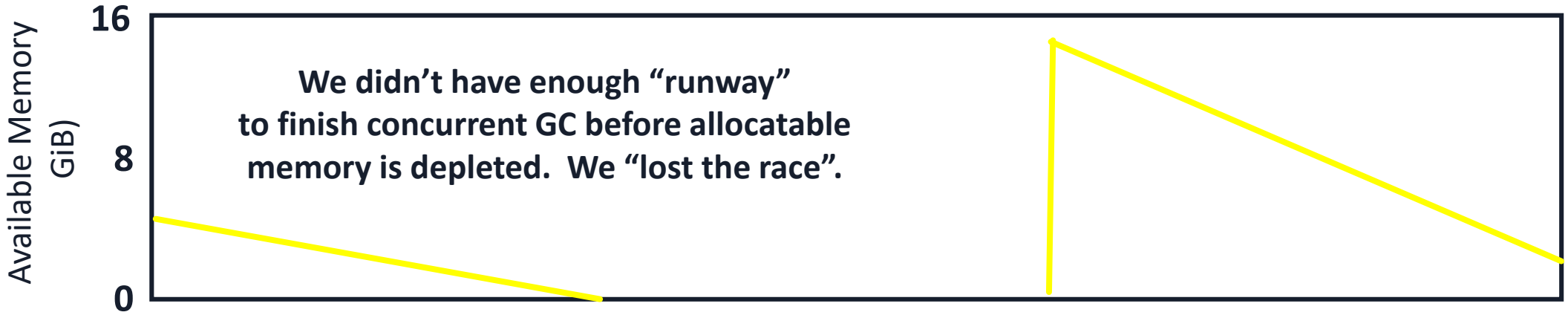
Shenandoah GC Running Well

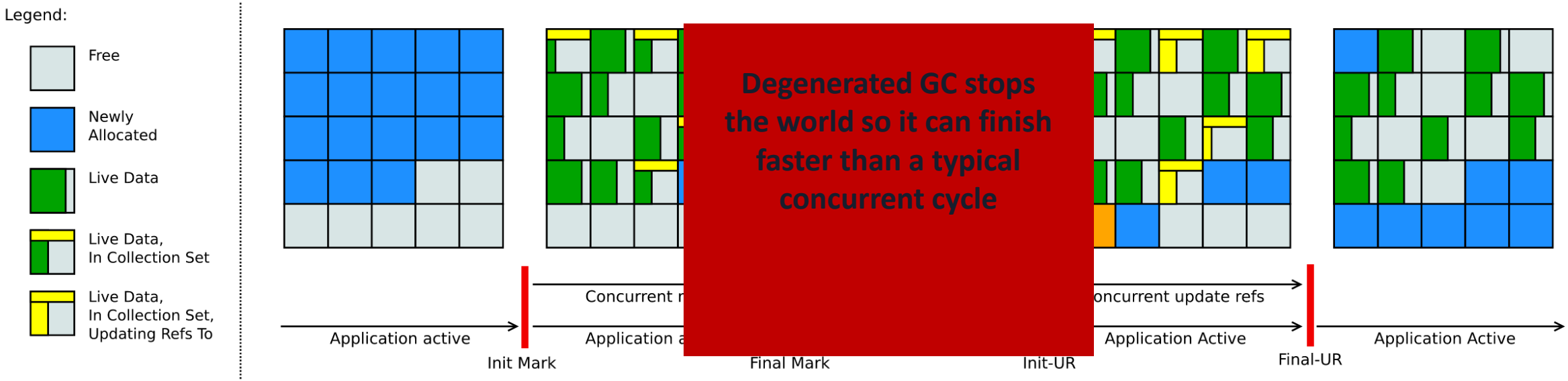




Shenandoah GC Not Running Well

Higher allocation rate here due to pent up demand

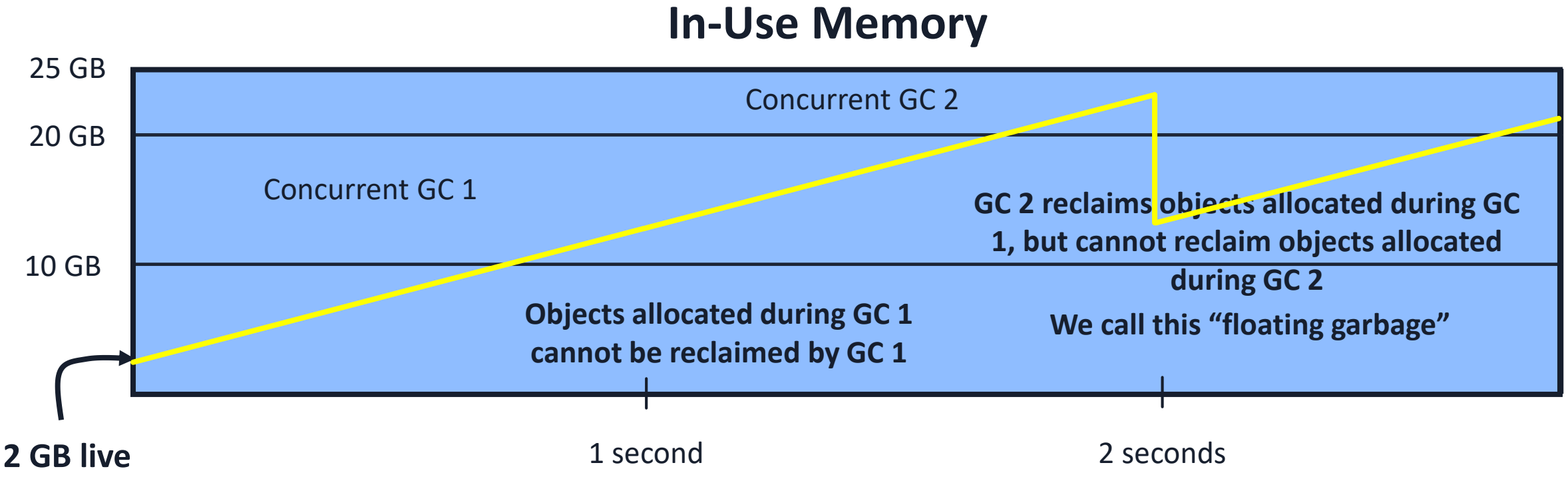




Motivating Generational Shenandoah

Sample workload: 2 GB live, 10 GB/s allocation

Assume 1 second GC time with traditional Shenandoah



Motivating Generational Shenandoah

Sample workload: 2 GB live, 10 GB/s allocation

Assume 1 second GC time with traditional Shenandoah:

- **Traditional Shenandoah requires 22 GB heap size**

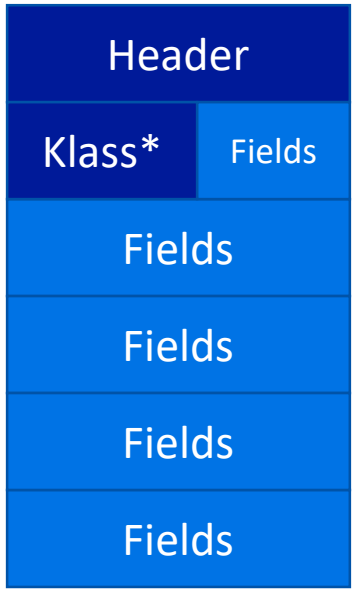
Now assume ¼ second young-gen GC time with GenShen:

- **Generational Shenandoah requires just 7 GB heap size**

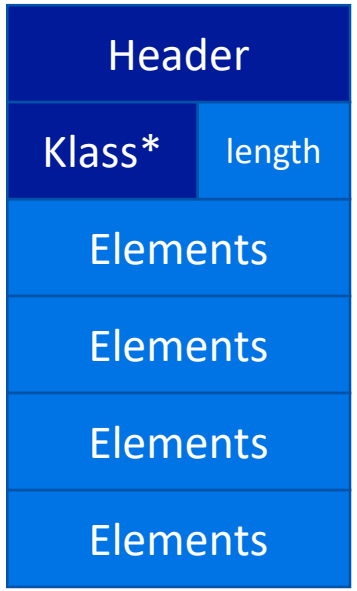
Project Lilliput

Object Headers – Current State

Java Object



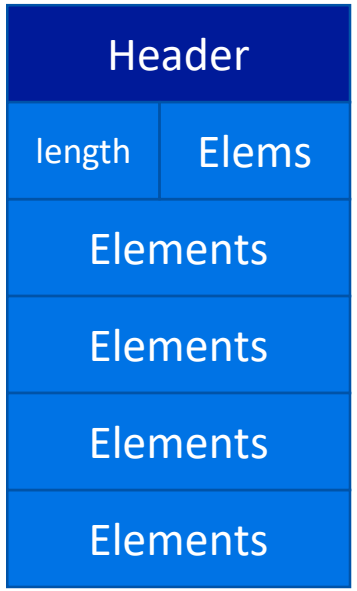
Java Array



Object Headers – Lilliput 1

Java Object

Java Array



What’s in the Object Header?

```
Mark Word (normal):
 64                      39                                  8    3  0
  [.......................HHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHH.AAAA.TT]
           (Unused)                  (Hash Code)         (GC Age)(Tag)
```

```
Class Word (compressed):
 32                                 0
  [CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC]
      (Compressed Class Pointer)
```

Insight:

- Most objects never get hashed
- Most objects never get locked

What’s in the Object Header?

```
Mark Word (overwritten):
 64                                                             2 0
  [ppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppppTT]
                      (Native Pointer)                         (Tag)
```

-> Pointer into stack (for stack-locking) (tag = 00)

-> Pointer to ObjectMonitor (for monitor-locking) (tag = 10)

-> Pointer to forwarded object (for GC forwarding) (tag = 11)

Lilliput 1

```
Header (compact):
 64                      42                                        7    3 0
  [CCCCCCCCCCCCCCCCCCCCCCCHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHVVVVAAAASTT]
   (Compressed Class Pointer)                (Hash Code) (GC Age)^(Tag)
                                                  (Self Forwarded Tag)
```

- 22 Klass bits
- 31 Hash bits
- 4 Valhalla reserved bits
- 4 GC age bits
- 2+1 Tag bits

Lilliput Dependencies

Lightweight Locking

- Don’t use object header for stack-locking
- Fixed in JDK21

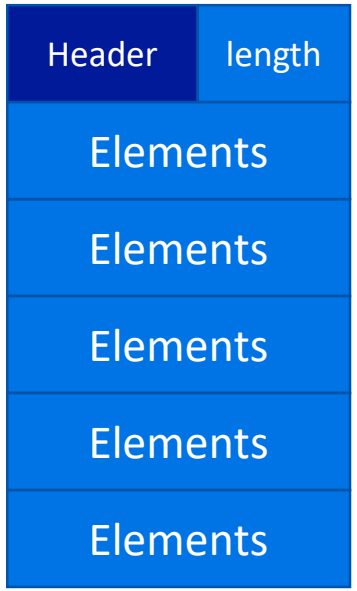
ObjectMonitor mapping

- Use side-table to map object -> ObjectMonitor
- Fixed in JDK24

Object Headers – Lilliput 2

Java Object

Java Array



Lilliput 2 – The Plan

```
Header (Lilliput 2):
 32                    11   7   3  0
  [CCCCCCCCCCCCCCCCCCCCHHVVVVAAAASTT]
       (Class Pointer)    ^(Age)^(Tag)
                 (Hash-Code) (Self Forwarded Tag)
```

- 19 Klass bits
- 2 Hash control bits
- 4 Valhalla reserved bits
- 4 GC age bits
- 2+1 Tag bits

AsyncProfiler

What is Async profiler?

Open source profiler for Java
https://github.com/async-profiler/async-profiler

Developed by JDK team at Amazon

Apache-2.0 license

Used in

- IntelliJ IDEA
- Grafana Pyroscope
- Datadog
- Elastic APM and more

Why Async profiler?

Accurate: no safepoint bias

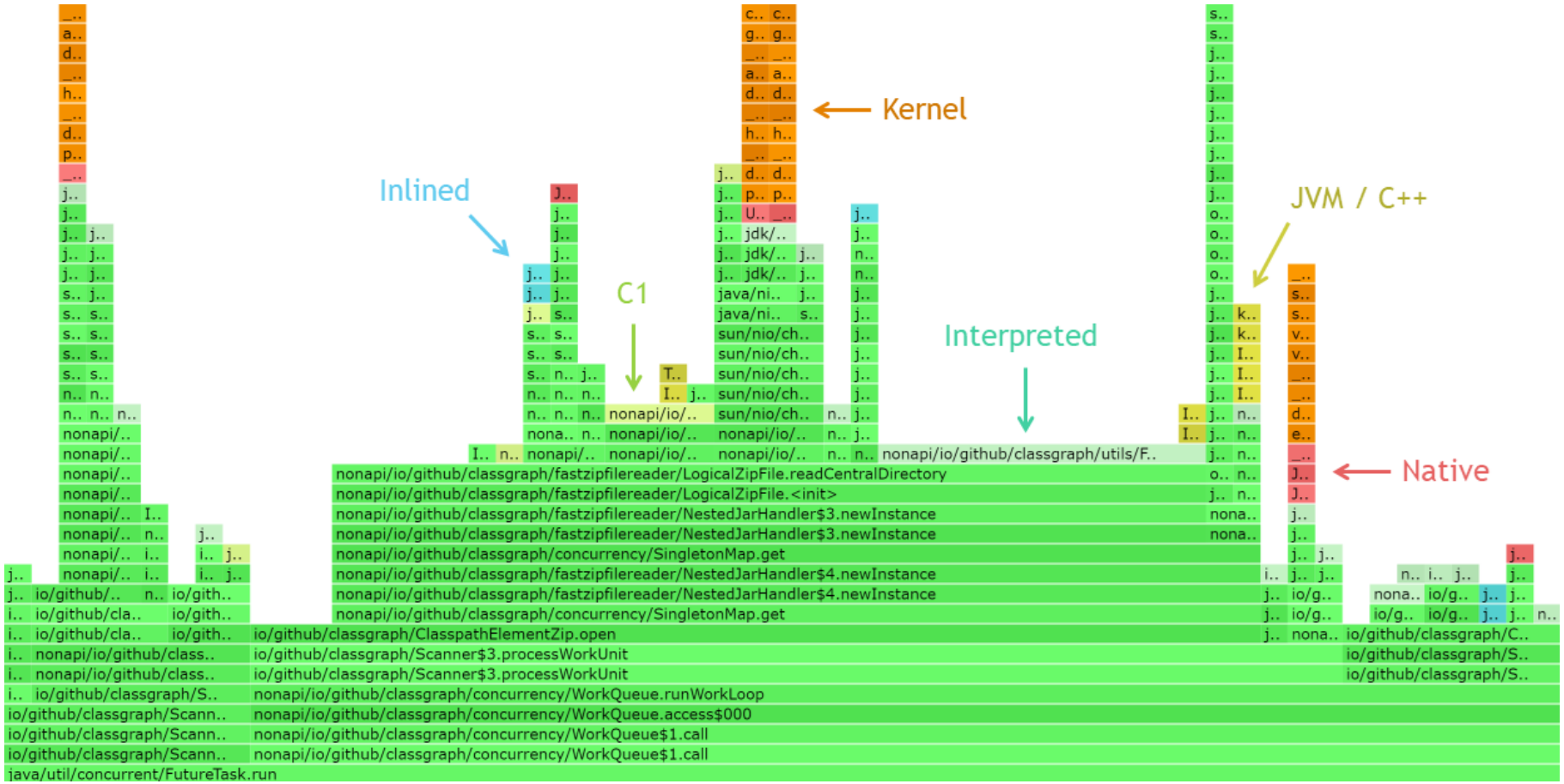
Low overhead

- +1% CPU usage +50 MB RAM
- Controllable overhead
- Suitable for production use

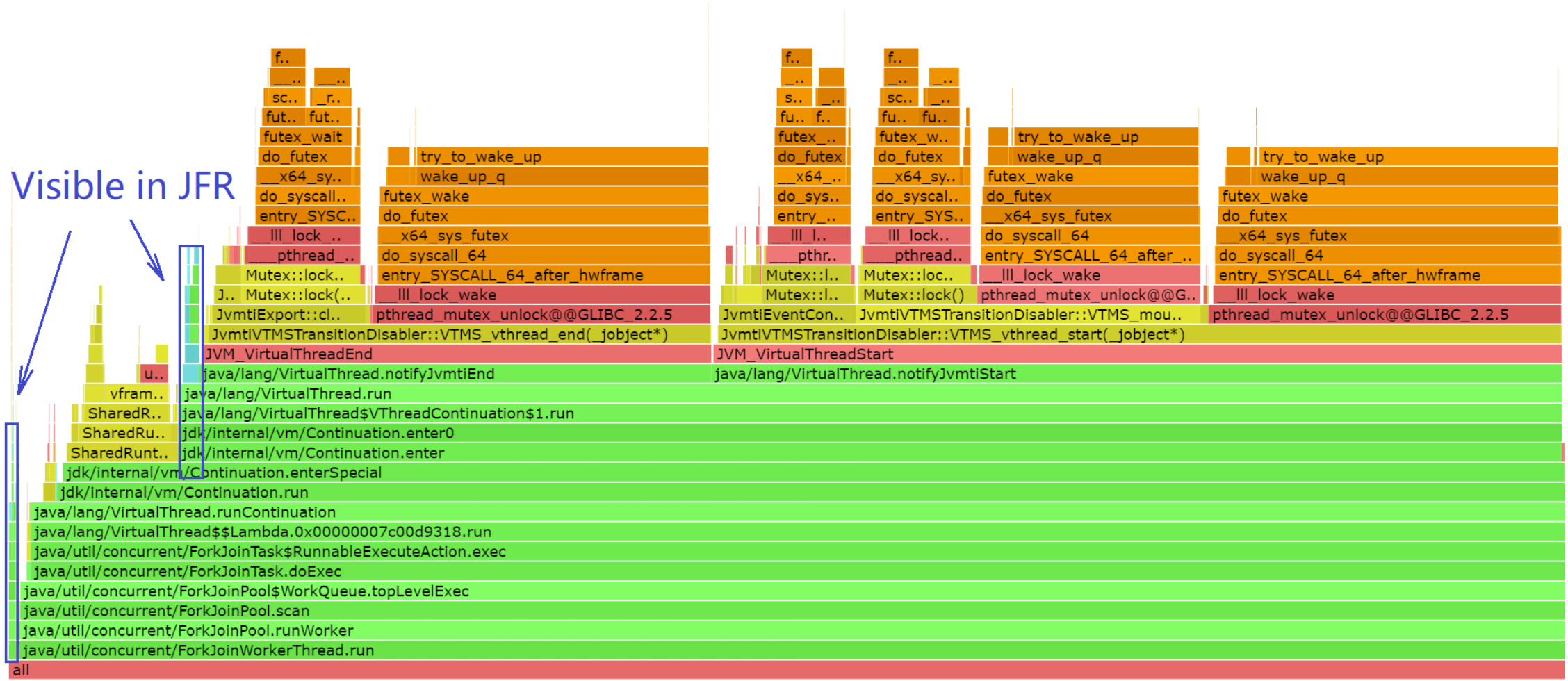
Supports CPU, Wall clock and Allocation profiling

Why Async profiler?

Comprehensive: Java + JVM + Native + Kernel stacks



Async profiler vs. JDK Flight Recorder



Async profiler + JFR

JFR compatible output format

- Viewable in JDK Mission Control, IntelliJ IDEA, etc.

Raw events with timestamps

Combined recordings with --jfrsync option

JMH/JOL

JMH – Java Micro Harness (github.com/openjdk/jmh)

- De-facto standard for Java benchmarking
- Used in the OpenJDK Project (see test/micro/ directory)

JMH – Java Micro Harness (github.com/openjdk/jmh)

- De-facto standard for Java benchmarking
- Used in the OpenJDK Project (see test/micro/ directory)

```
public class JMHSample_01_HelloWorld {
    @Benchmark
    public void wellHelloThere() {
        // this method was intentionally left blank.
    }
}
```

JMH – Java Micro Harness (github.com/openjdk/jmh)

- De-facto standard for Java benchmarking
- Used in the OpenJDK Project (see test/micro/ directory)

```
public class JMHSample_01_HelloWorld {
    @Benchmark
    public void wellHelloThere() {
        // this method was intentionally left blank.
    }
}
```

```
$ mvn clean install
$ java -jar target/benchmarks.jar JMHSample_01
```

JMH – Java Micro Harness (github.com/openjdk/jmh)

```
# Run progress: 0.00% complete, ETA 00:08:20
# Fork: 1 of 5
# Warmup Iteration   1: 2020531759.278 ops/s
# Warmup Iteration   2: 1552635209.690 ops/s
# Warmup Iteration   3: 1269440482.962 ops/s
# Warmup Iteration   4: 1215019056.877 ops/s
# Warmup Iteration   5: 1549509310.024 ops/s
Iteration   1: 1747935918.929 ops/s
Iteration   2: 1730435918.372 ops/s
Iteration   3: 1635819935.621 ops/s
Iteration   4: 1682616358.254 ops/s
Iteration   5: 1844876106.873 ops/s
...

Benchmark                                  Mode  Cnt          Score          Error  Units
JMHSample_01_HelloWorld.wellHelloThere    thrpt   25  1787425383.507 ± 46345392.961  ops/s
```

JOL – Java Object Layout (github.com/openjdk/jol)

- Uses Unsafe, JVMTI, and Serviceability Agent (SA) to decode object layout & footprint

```
java.lang.String object internals:
OFF  SZ      TYPE DESCRIPTION               VALUE
  0   8           (object header: mark)     0x0000000000000001 (non-biasable; age: 0)
  8   4           (object header: class)    0x0000ec20
 12   4       int String.hash               0
 16   1      byte String.coder              0
 17   1   boolean String.hashIsZero         false
 18   2           (alignment/padding gap)
 20   4    byte[] String.value              []
Instance size: 24 bytes
Space losses: 2 bytes internal + 0 bytes external = 2 bytes total
```

Remember…

Friends don’t let friends run JDK8

Remember…

Friends don’t let friends run JDK8 (or JDK11, 17.. :)

Automatic Upgrade & Migration Tools

- Tools do static & dynamic code analysis
- They have a knowledge base of known upgrade issues.
- They can use LLMs to detect patterns not in the database.
- Some well-known tools:
  - OpenRewrite
  - Eclipse Migration Toolkit for Java
  - Windup/ Migration Toolkit for Runtimes
  - Amazon Q Developer Agent for Code Transformation

Amazon Q Developer

Reimagines the experience across the entire software development lifecycle (SDLC)

Helps developers and IT professionals build and manage secure, scalable, and highly available applications

Helps you write, debug, test, optimize, and upgrade your code faster

Converses with you to explore new AWS capabilities, learn unfamiliar technologies, and architect solutions

Amazon Q is built with security and privacy in mind from the start, making it easier for organizations to use generative AI safely.

Amazon Q Developer supports developers across the SDLC

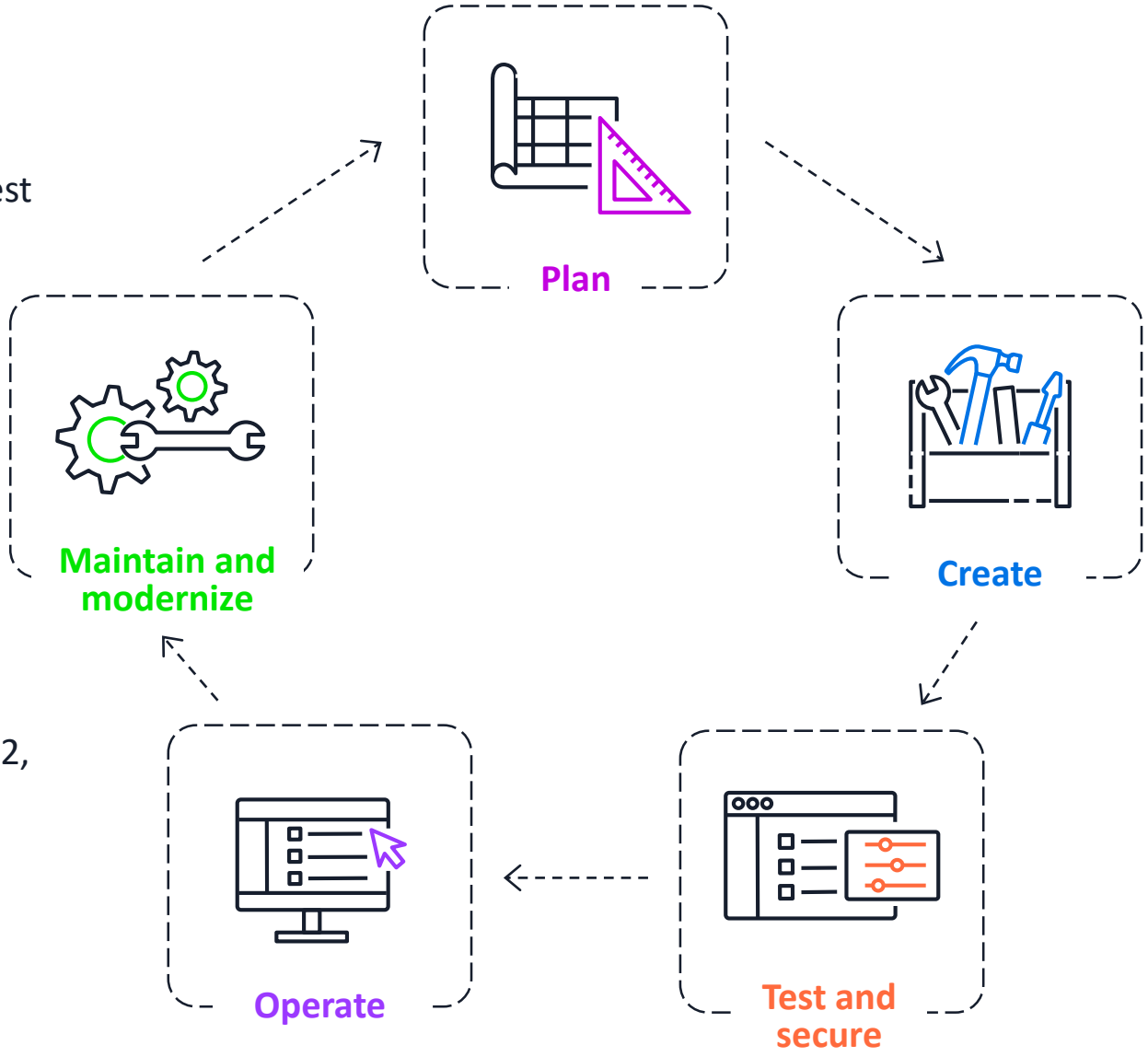
Plan
- Amazon Q Developer in the AWS Management Console (best practices, AWS WAF, Amazon EC2 instance optimization)
- Business-specific application
- Explain code with conversational coding

Create
- Inline coding companion in IDE and CLI
- Software development
- Conversational coding

Test and secure
- Unit test generation
- Security scanning and remediation

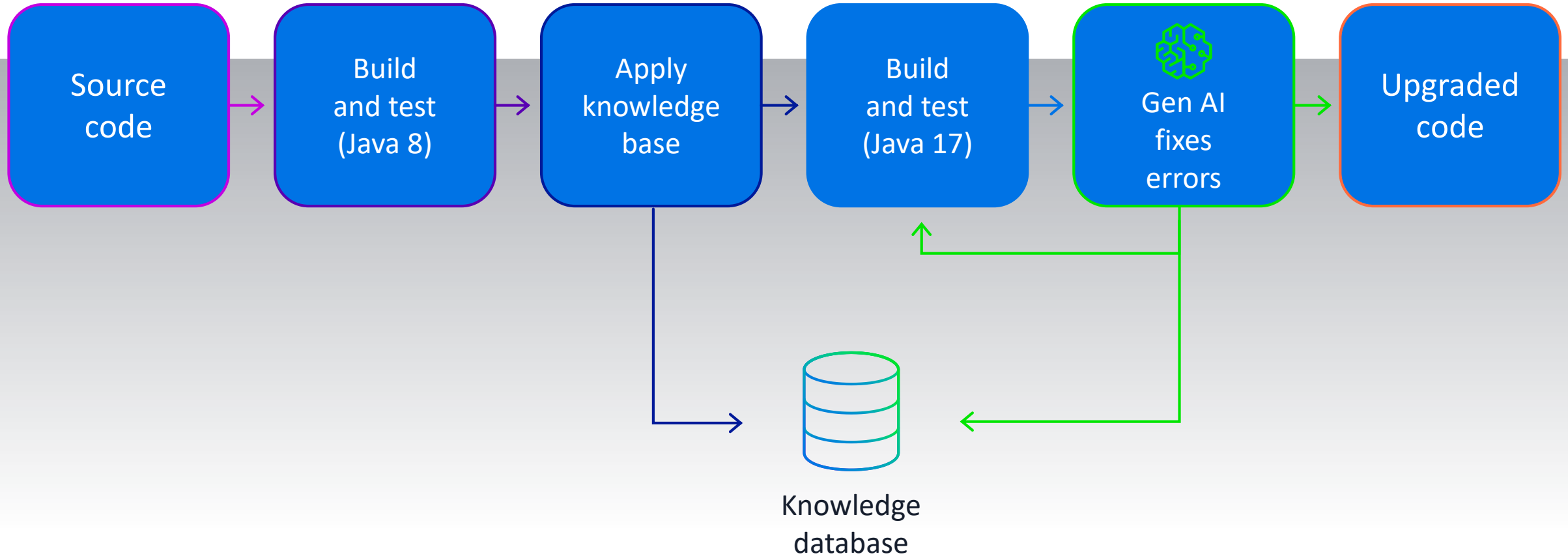
Operate
- Troubleshoot errors (Amazon S3, AWS Lambda, Amazon EC2, Amazon Elastic Container Service)
- VPC Reachability Analyzer
- Debug and optimize code with conversational coding
- Helps you optimize your AWS resources and costs [NEW]

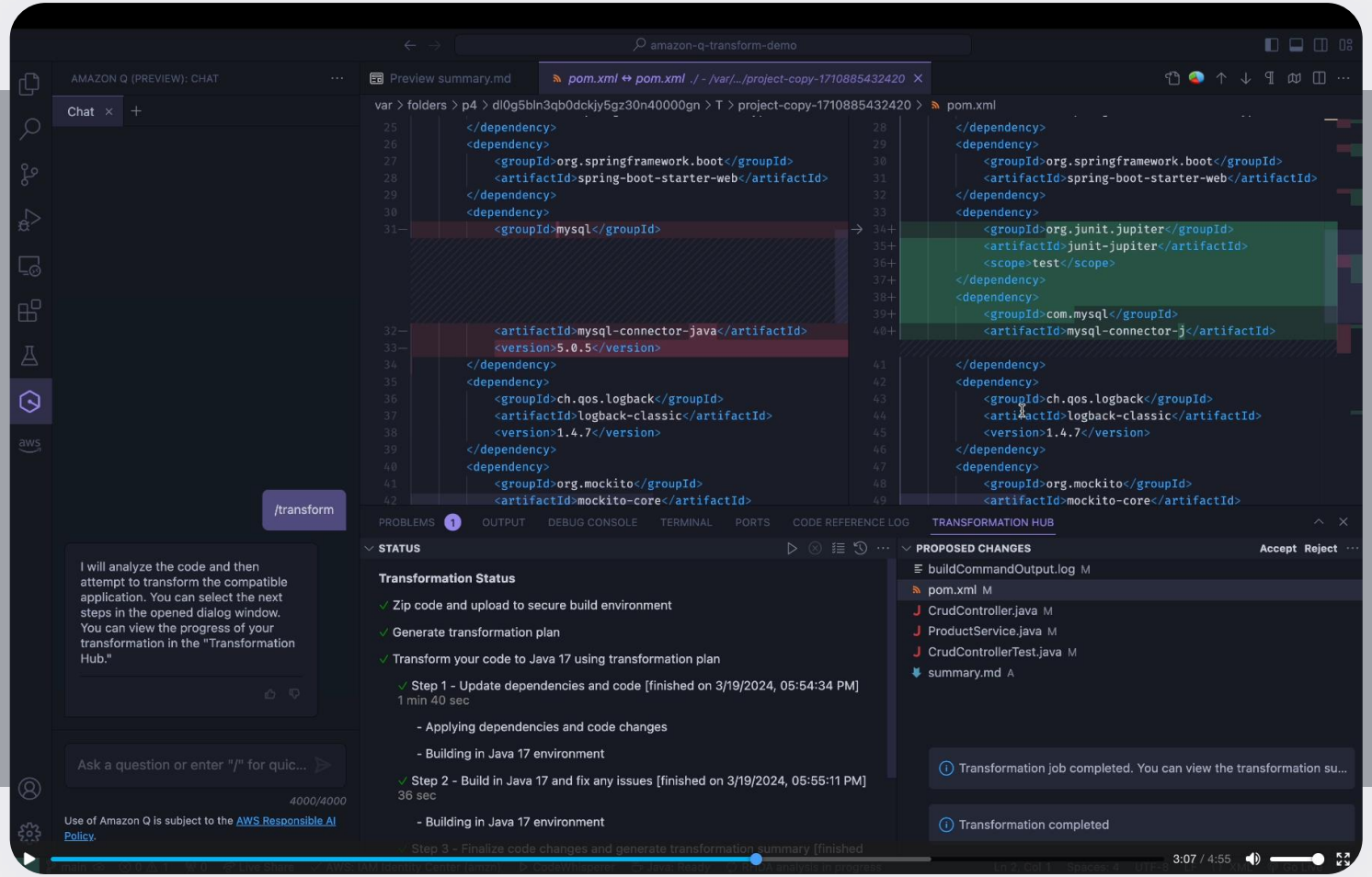
Maintain and modernize
- Update code with Amazon Q Developer agent for code transformation



Maintain and modernize: Amazon Q Developer agent for code transformation

MODERNIZE LANGUAGE VERSIONS IN A FRACTION OF THE TIME





Amazon Q Developer agent for code transformation

Automates the complete process of upgrading and transforming code

Amazon Q Developer agent for code transformation

1,000

Java applications

2

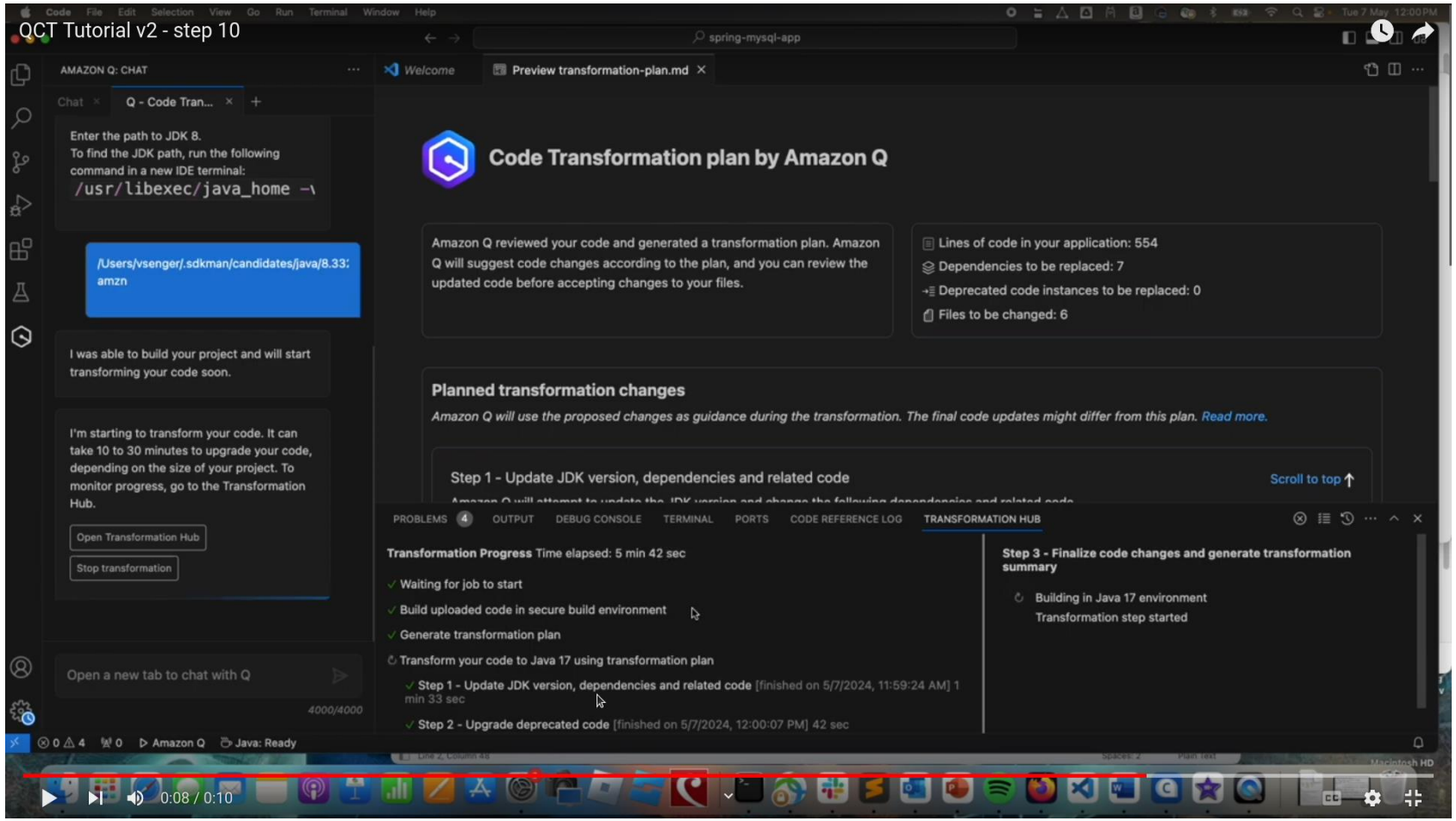
Days

10

Minutes each, on average

Getting Started with Amazon Q Developer Agent for code transformation

By Vinicius Senger



Thank you!

Volker Simonis

simonisv@amazon.de