



Java 8 to \geq 17 Upgrade - Lessons Learned & Best Practices

Volker Simonis

Principal Software Engineer
Amazon Web Services



“Success and Scale Bring Broad Responsibility”

“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>

The JDK team @ Amazon

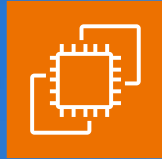


...

The JDK team @ Amazon



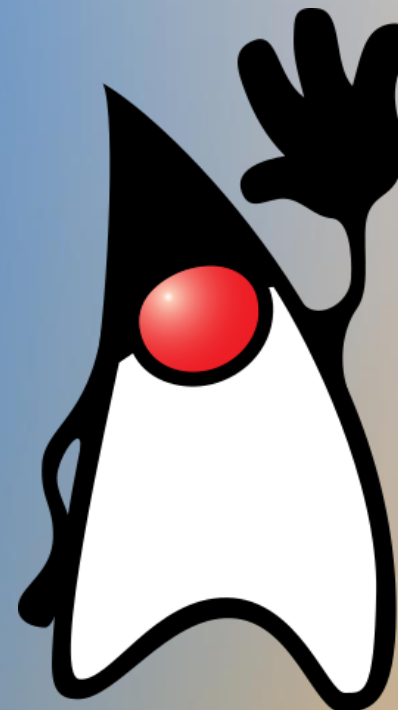
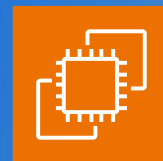
...



The JDK team @ Amazon



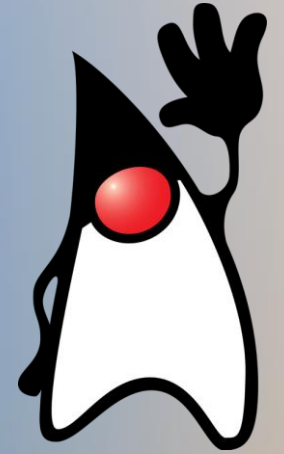
...



What is Amazon Corretto?

- Downstream distribution of OpenJDK
- Quarterly security releases
- Certified
- **No-cost long-term releases** – JDK 8, JDK 11, JDK 17, JDK 21
- Feature release train (JDK 16, JDK 18, JDK 19, JDK 20, JDK 22 ...)

- We launched in Devovx Belgium 2018! 3 binary artifacts
- Now 122 binary artifacts
 - JDK8: 22 artifacts, JDK11: 31, JDK20: 7, JDK17: 31, JDK 21: 31





AWS Graviton2 & Graviton3

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



EUROPE

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



ASIA PACIFIC

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



AFRICA

- Cape Town 3



MIDDLE EAST

- Bahrain 3
- Tel Aviv 3
- UAE 3



S AMERICA

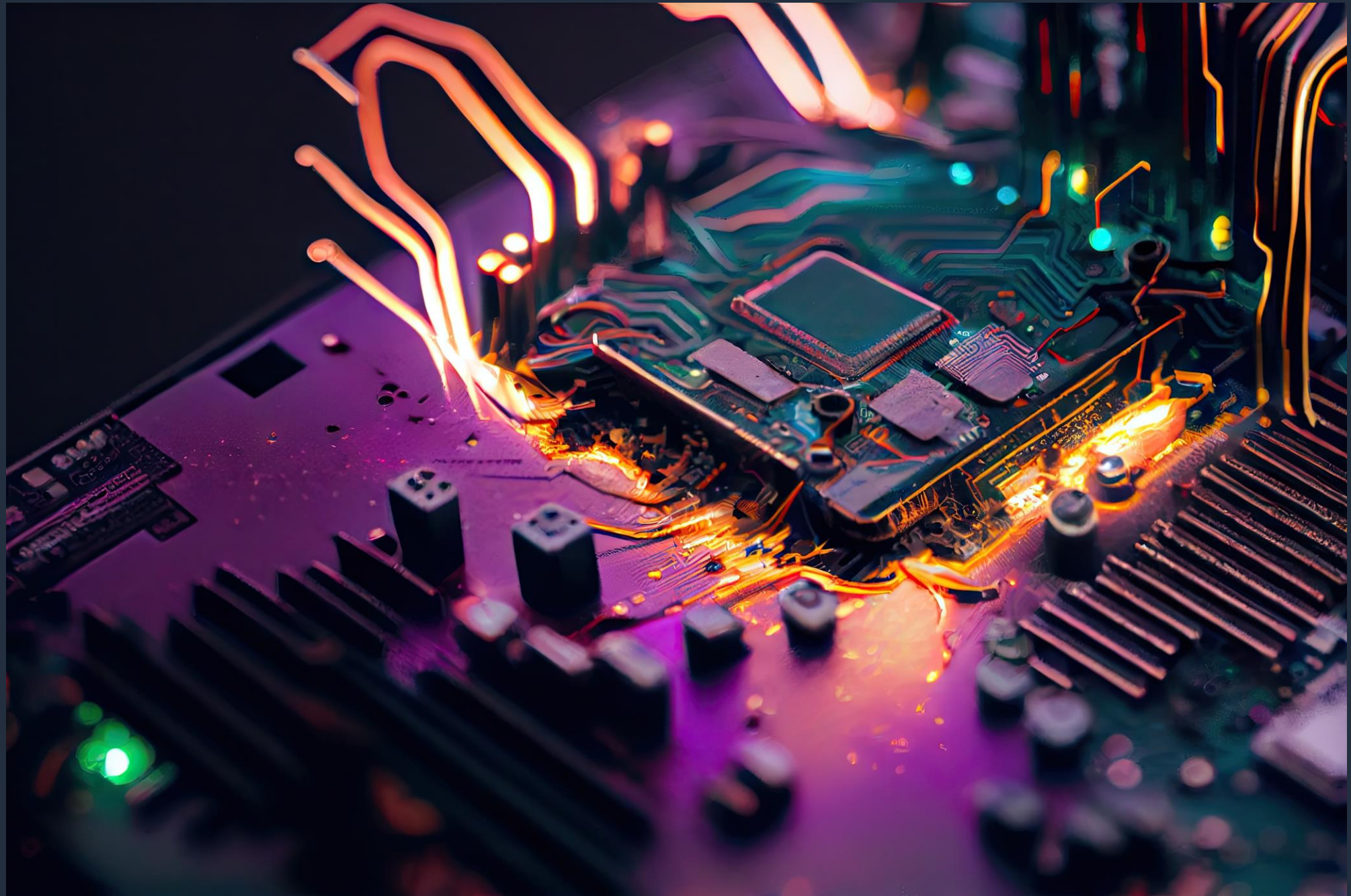
- São Paulo 3



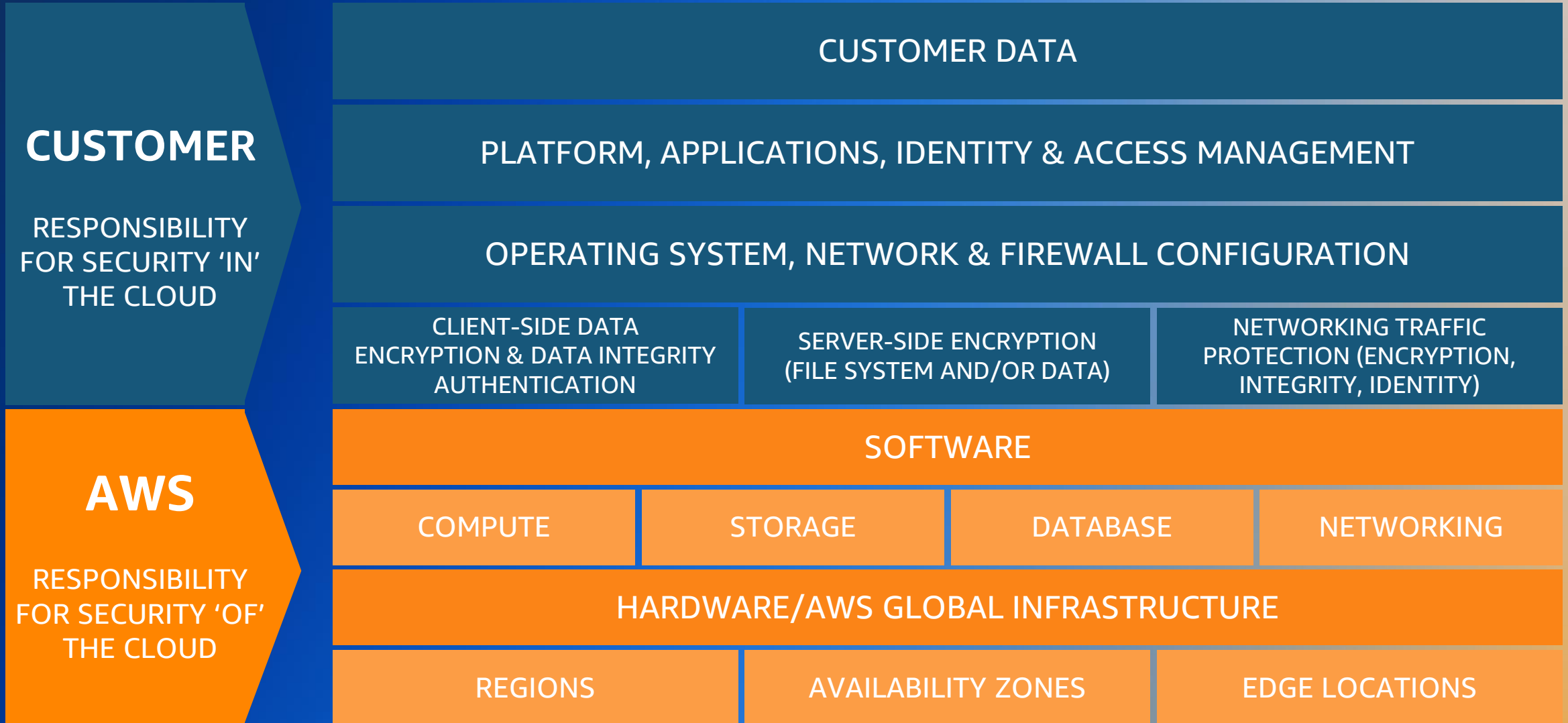
AUSTRALIA & NEW ZEALAND

- Melbourne 3
- Sydney 3
- Auckland

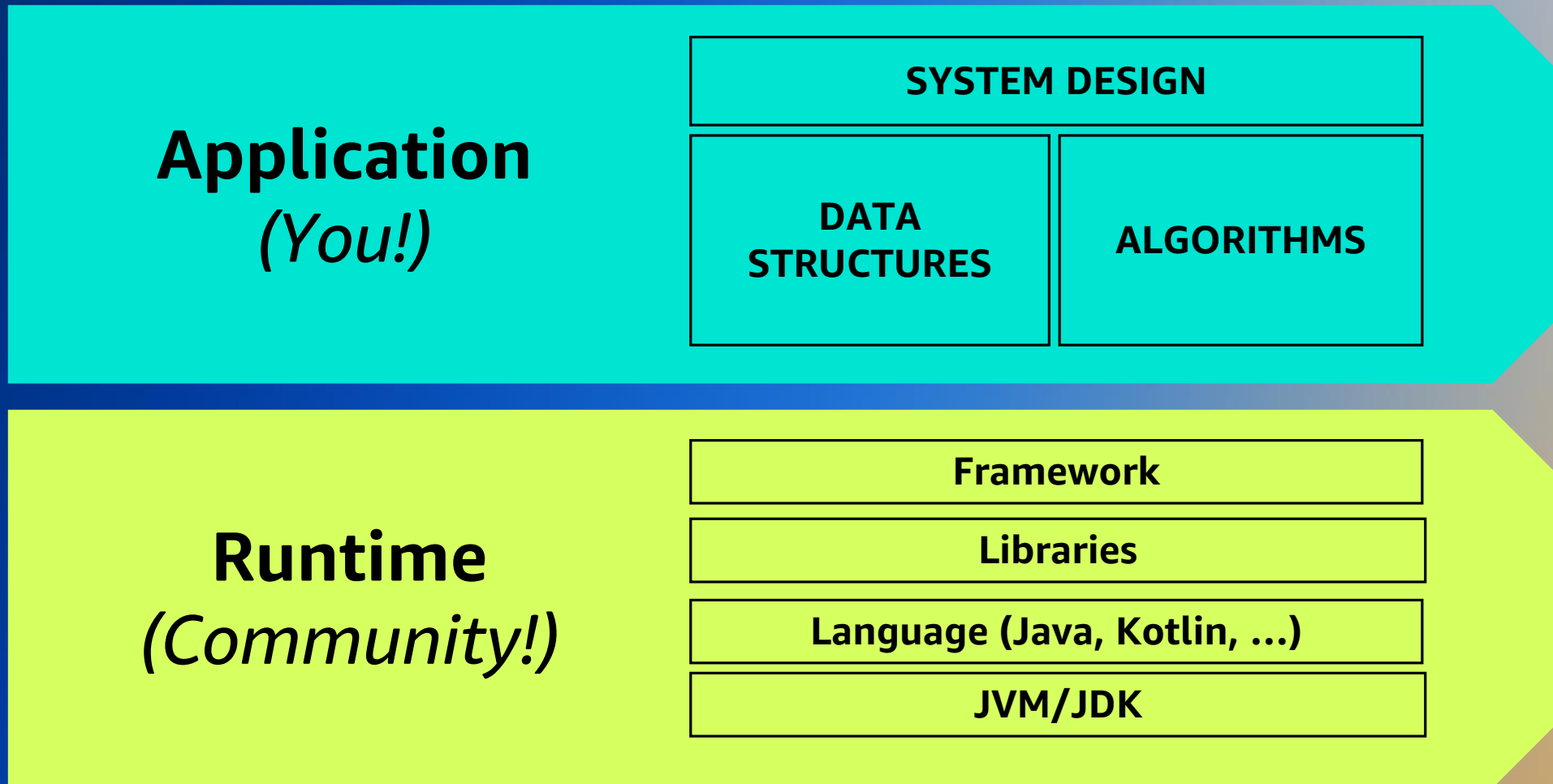
● Available Region ● Announced # Availability Zone



AWS Shared Responsibility Model



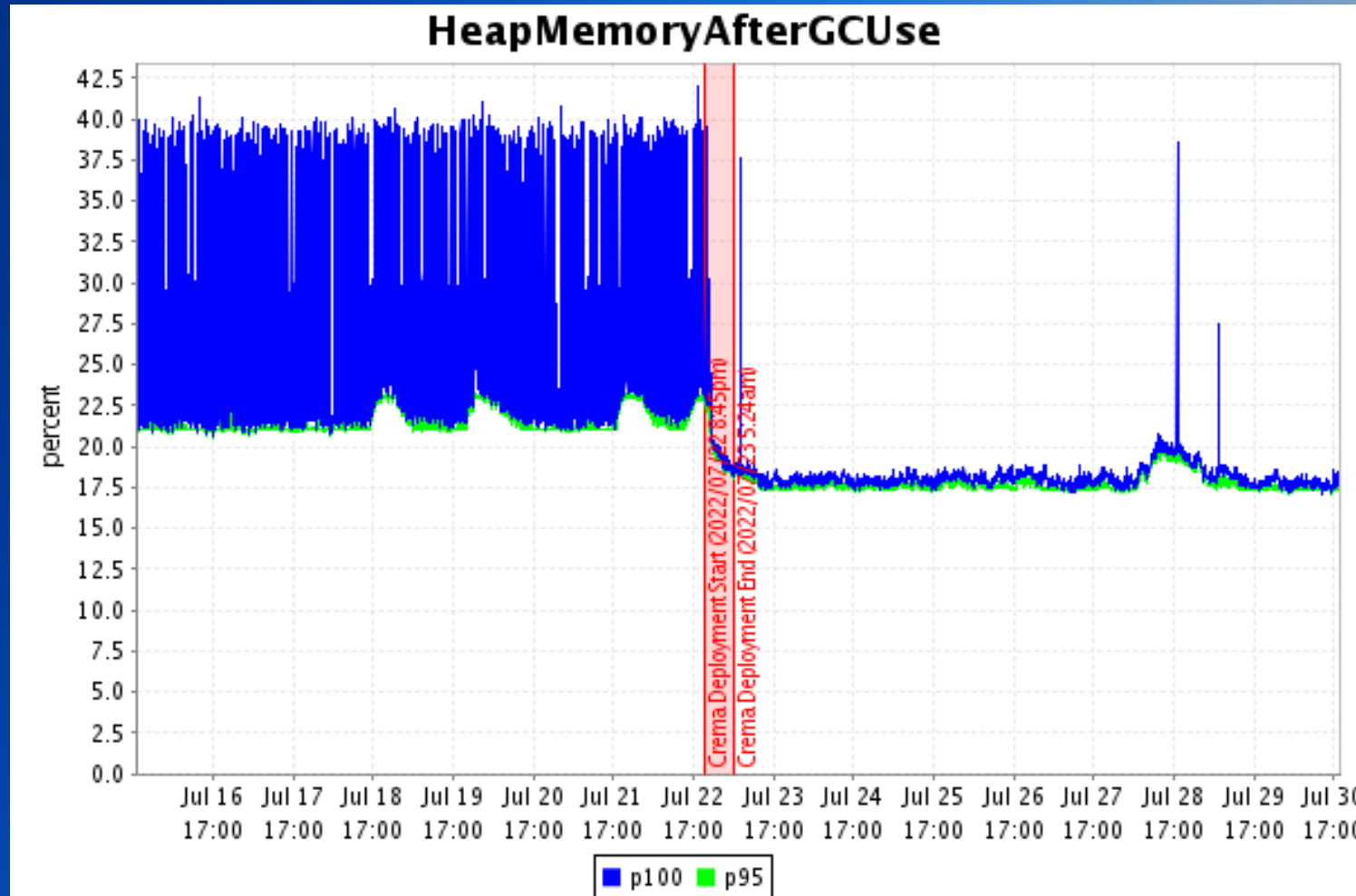
Application Performance Shared Responsibility Model



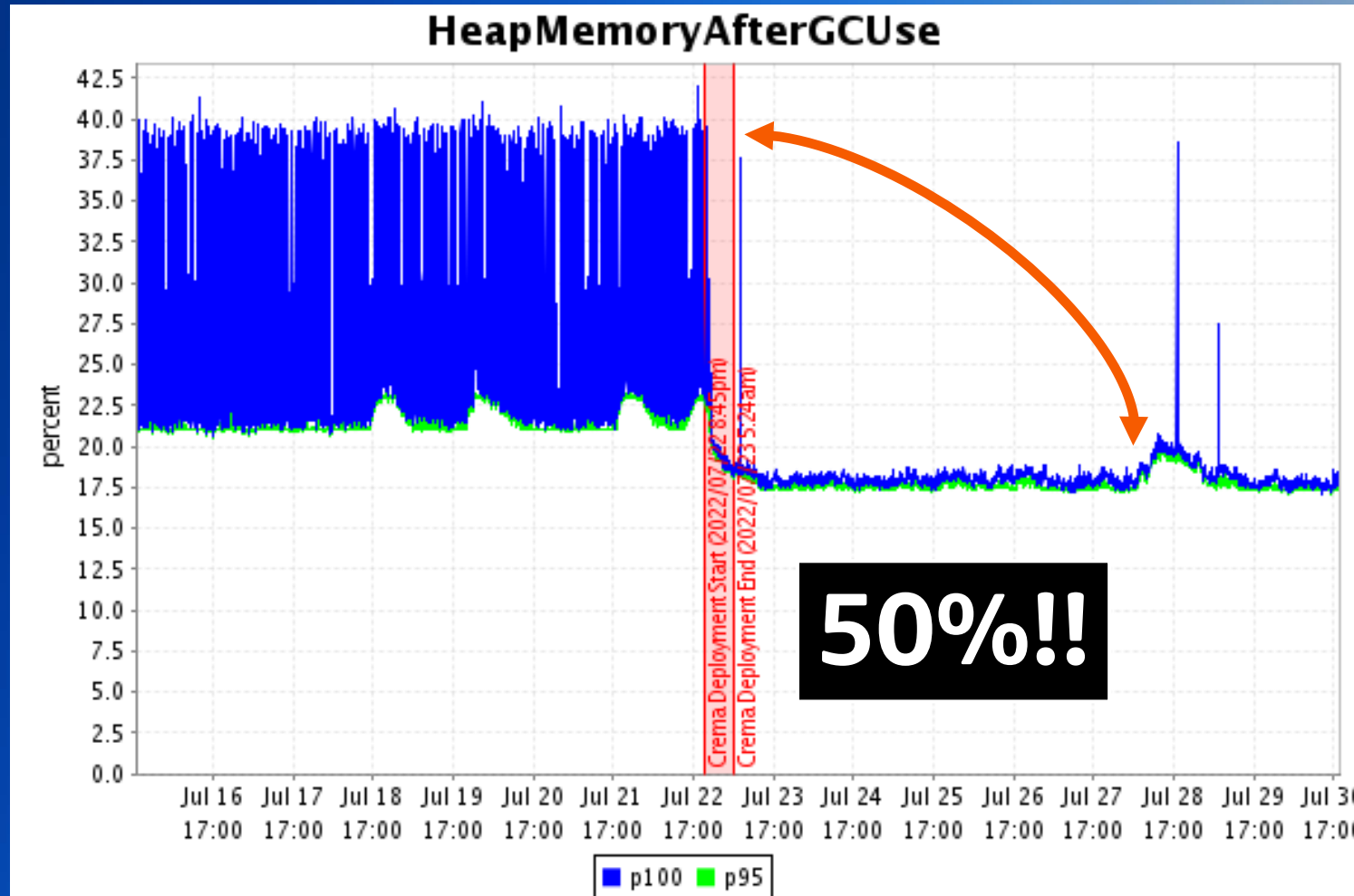
The good, the bad, and the ugly.



Example: Monitoring service 1/5



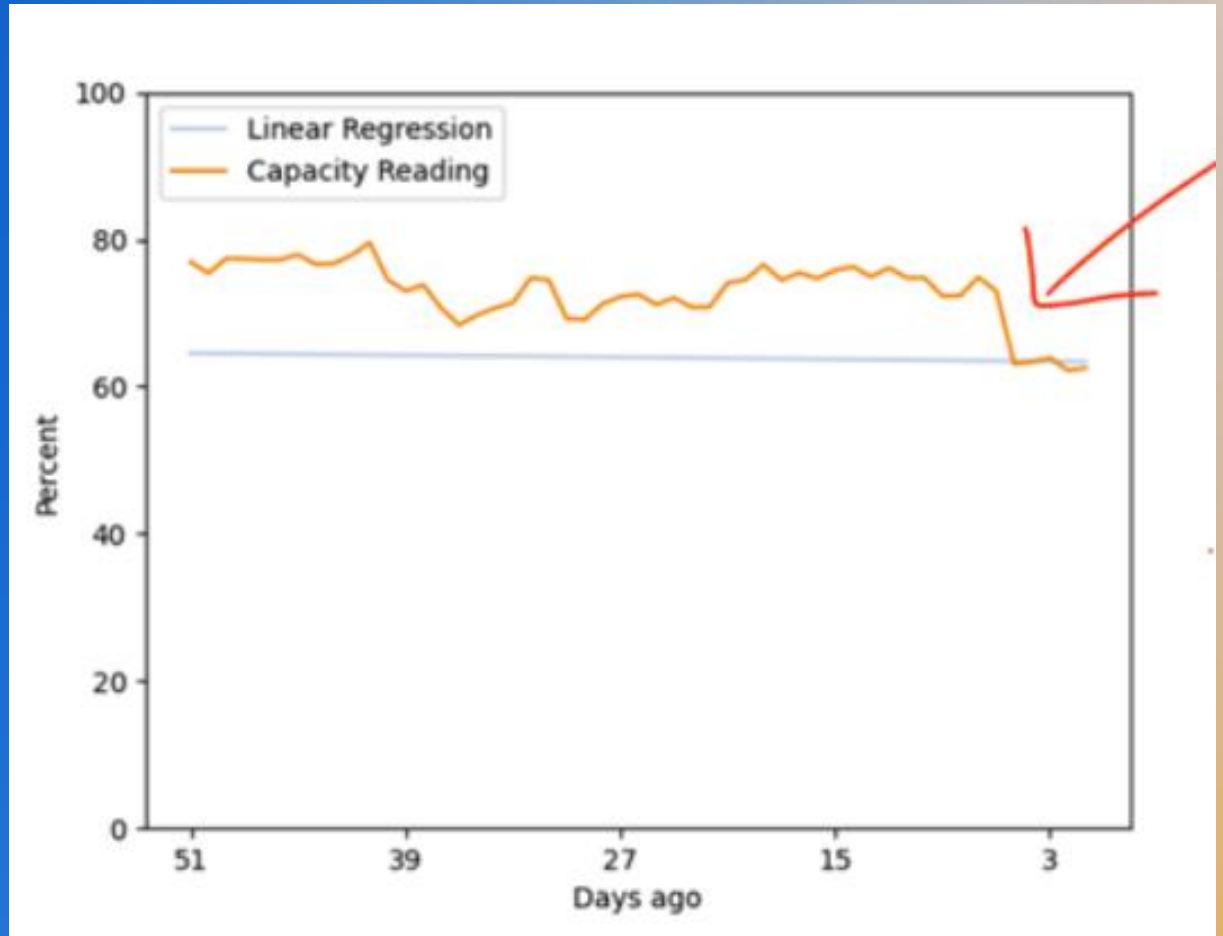
Example: Monitoring service 1/5



Another Monitoring Service 2/5

JDK 8 to JDK 17

- 7% fleet utilization improvement
- Millions in savings
- 98% cold start fault reduction
- 19% p99 PUT latency reduction

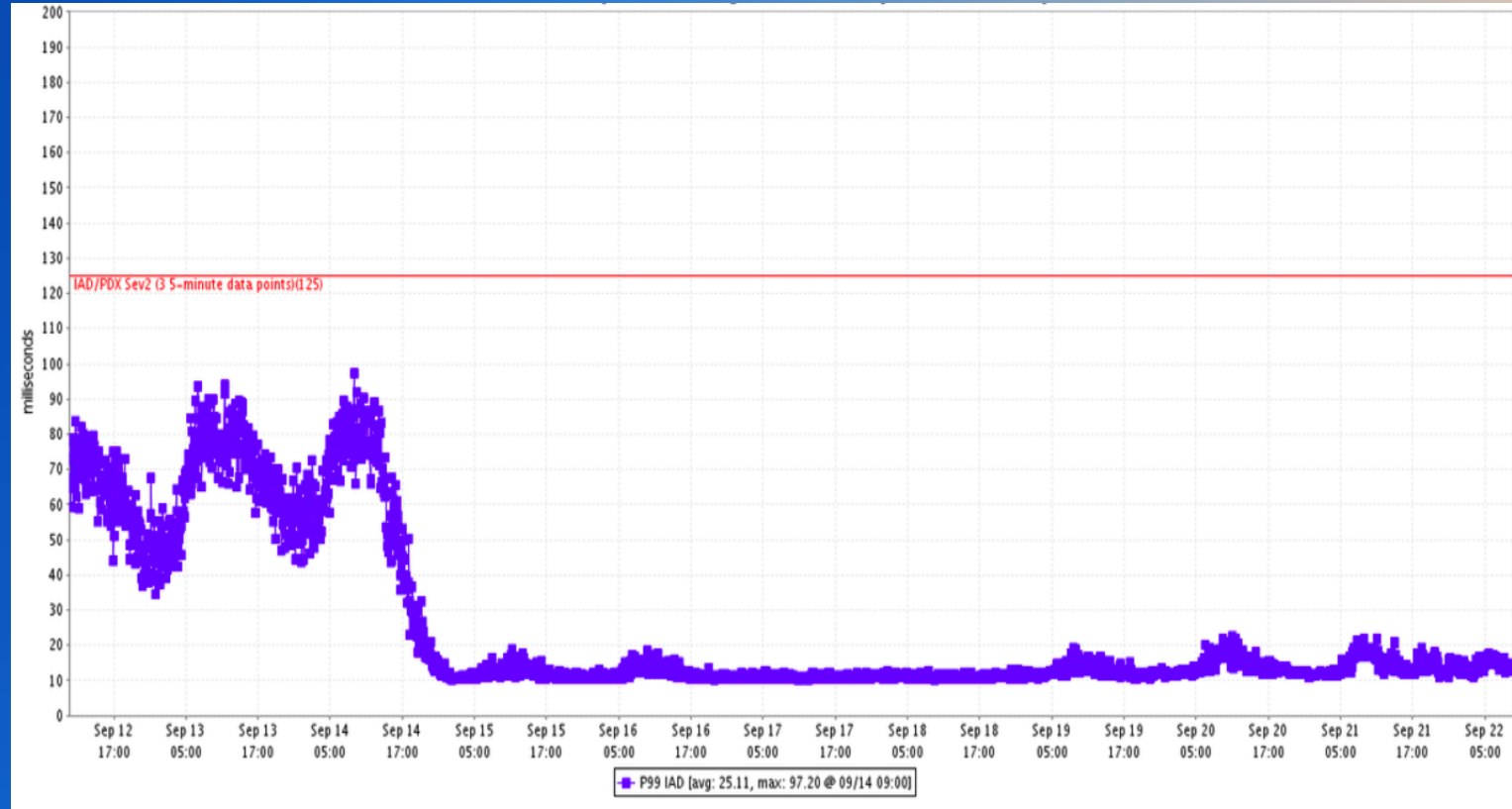


Proxy Service Improvement 3/5

JDK 8 to 17

75% lower end to end latency

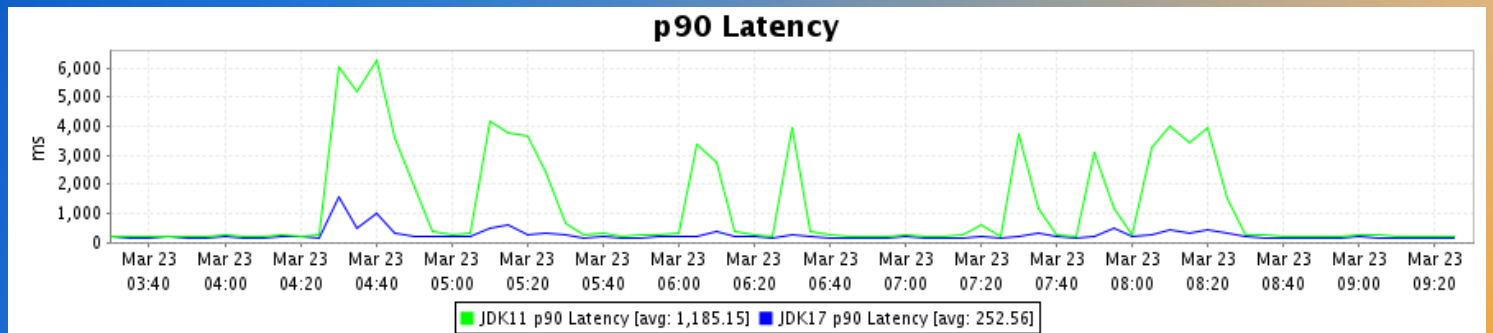
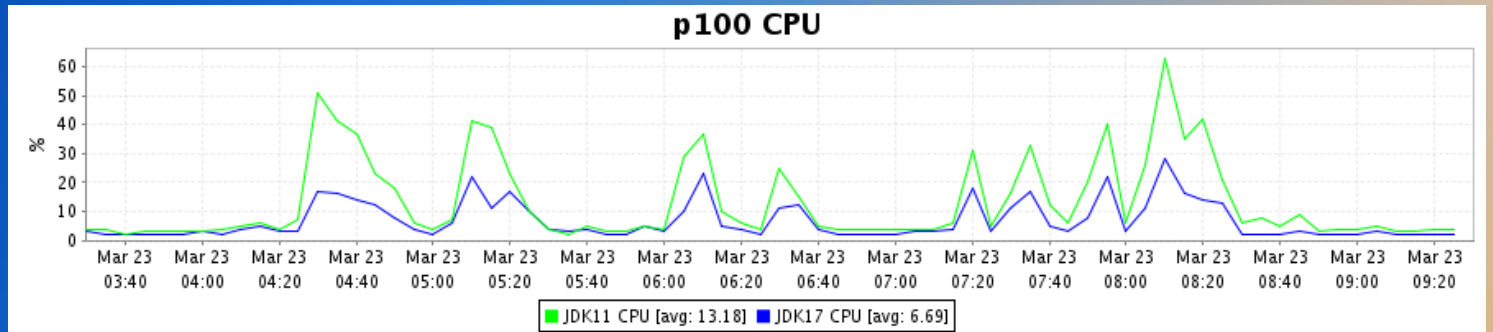
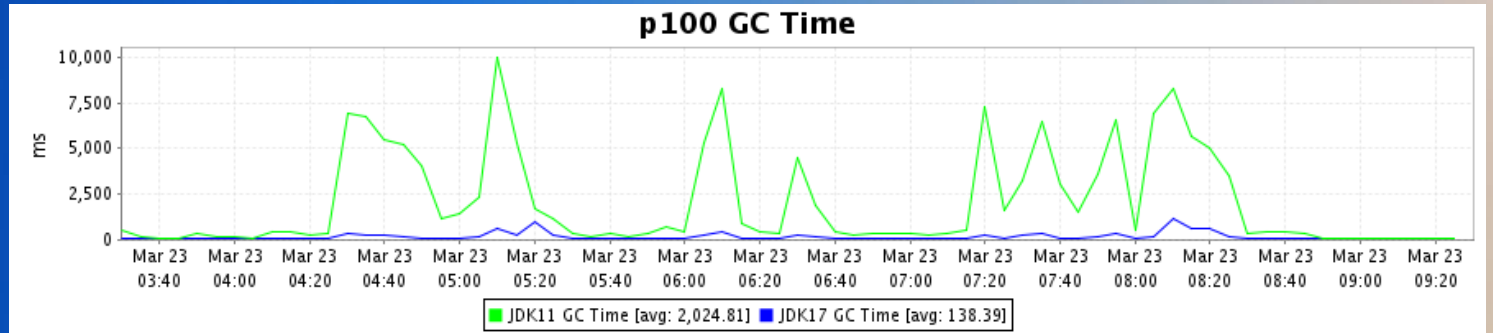
30-40% capacity improvement



Analytics 4/5

JDK 11 to 17

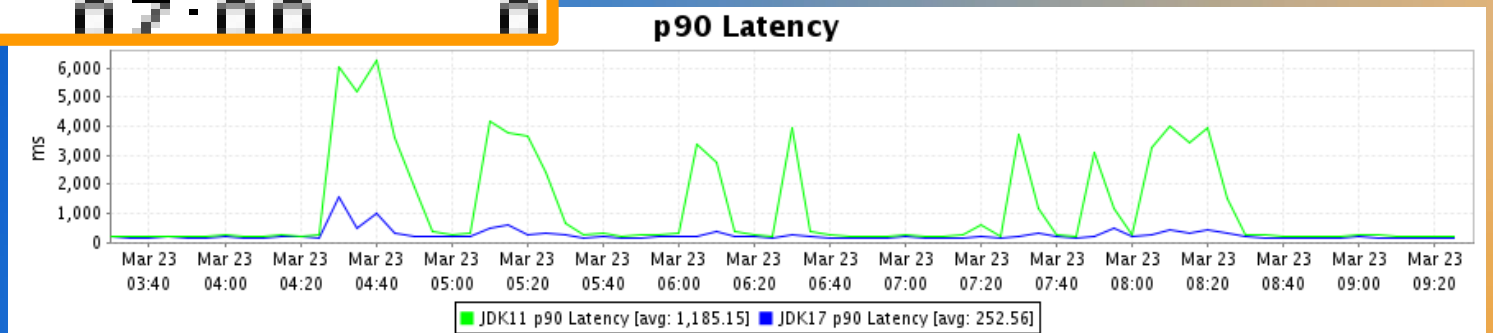
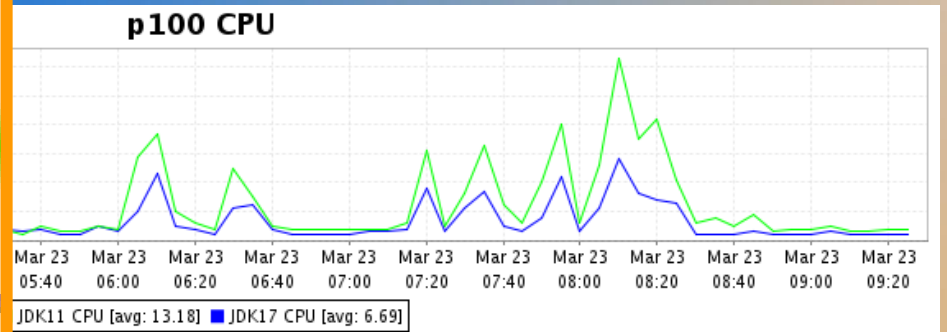
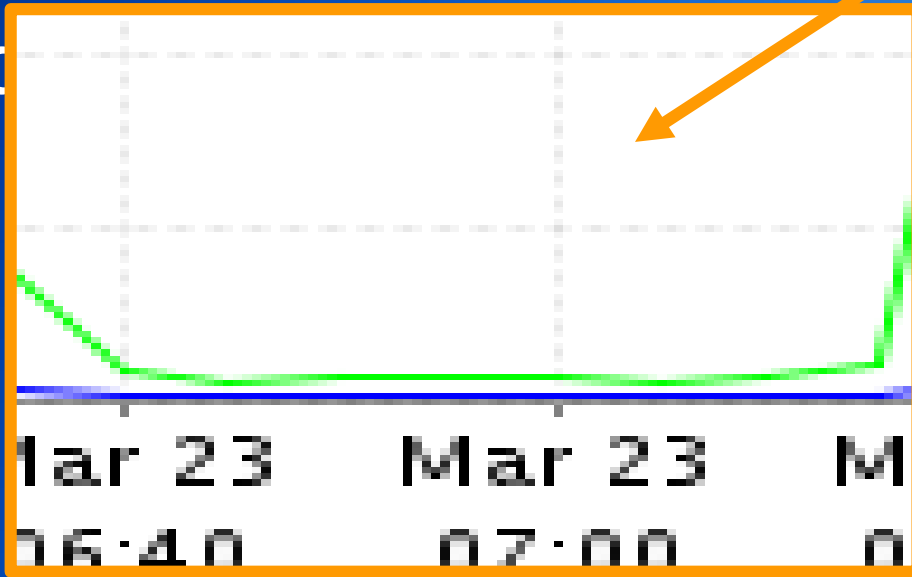
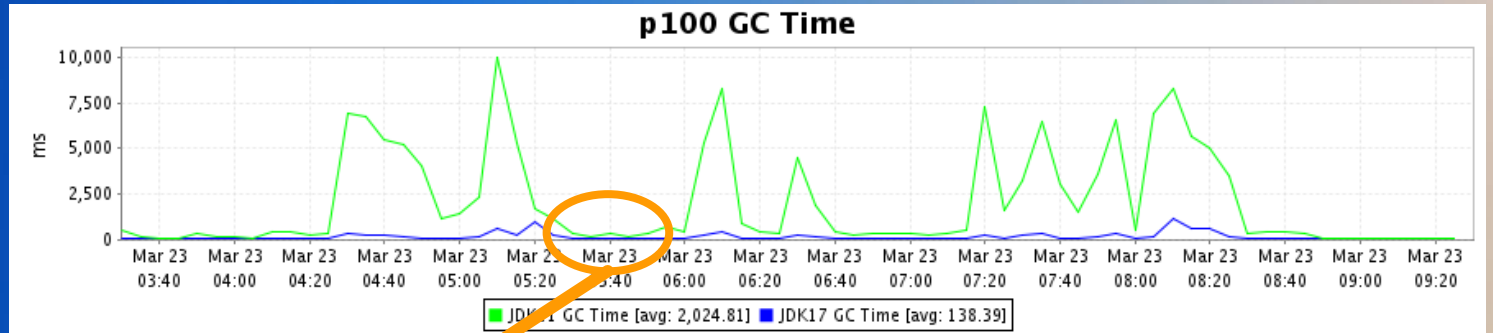
Always compare
under high load



Analytics 4/5

JDK 11 to 17

Always compare
under high load



Proxy Service – No Improvement 5/5

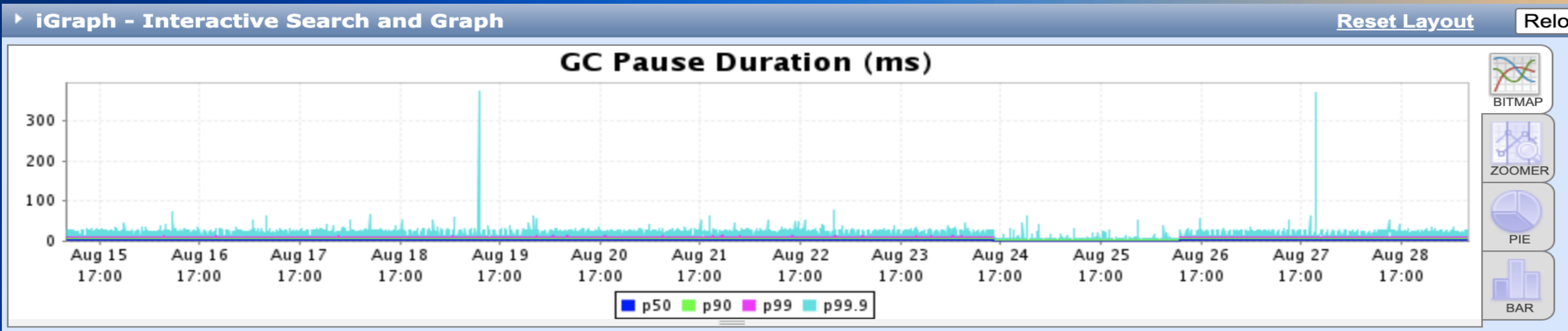
JDK 11 to 17

Is GC the limit for a low latency service?

Proxy Service – No Improvement 5/5

JDK 11 to 17

Is GC the limit for a low latency service?



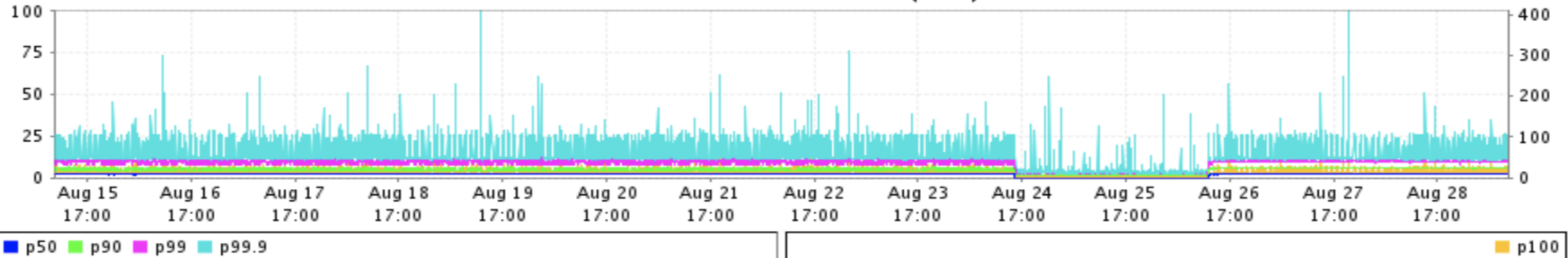
Proxy Service – No Improvement 5/5

iGraph - Interactive Search and Graph

[Reset Layout](#)

[Relo](#)

GC Pause Duration (ms)

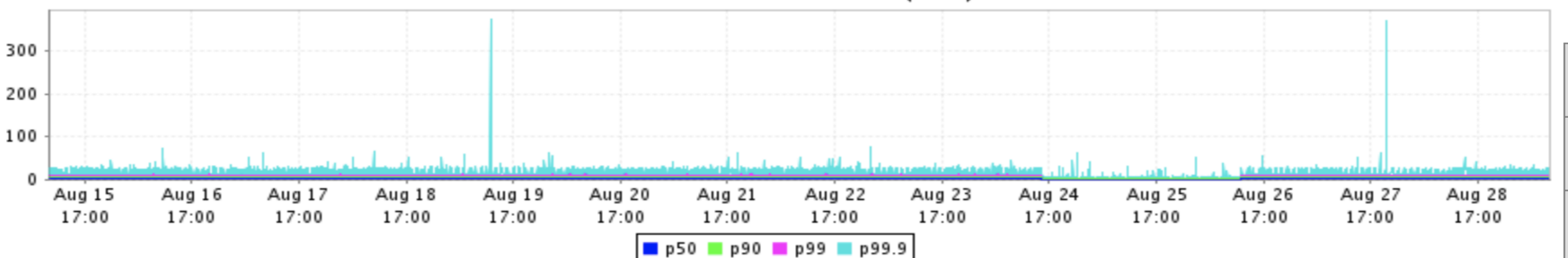


iGraph - Interactive Search and Graph

[Reset Layout](#)

[Relo](#)

GC Pause Duration (ms)



Proxy Service – No Improvement 5/5

iGraph - Interactive Search and Graph

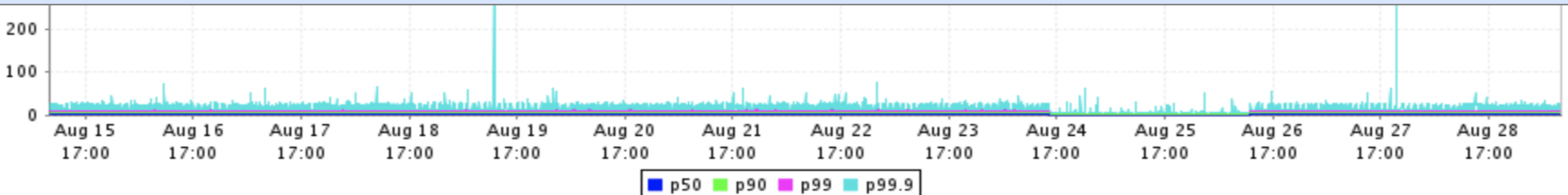
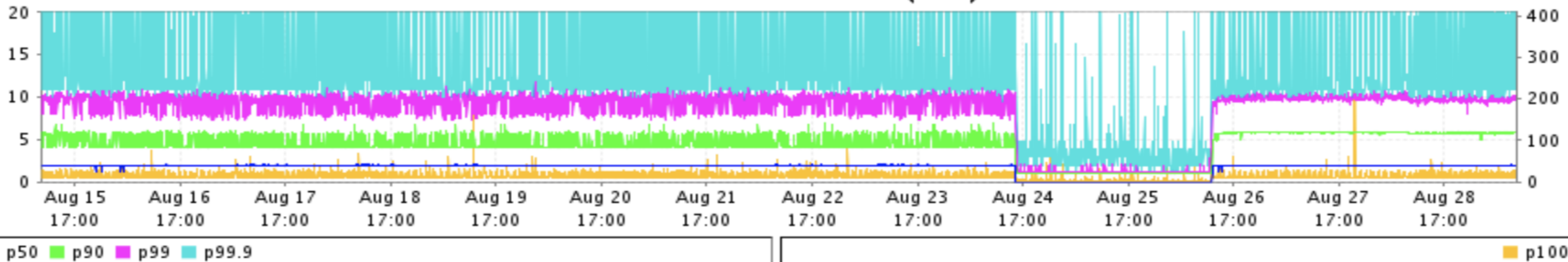
Reset Layout

Relo

GC Pause Duration (ms)



GC Pause Duration (ms)



Remember...

**Friends don't let
friends run JDK8
(or JDK11)**



The good, the **bad**, and the ugly



The good, the **bad** (not really that bad), and the ugly



JDK 17 upgrade regression case study 1

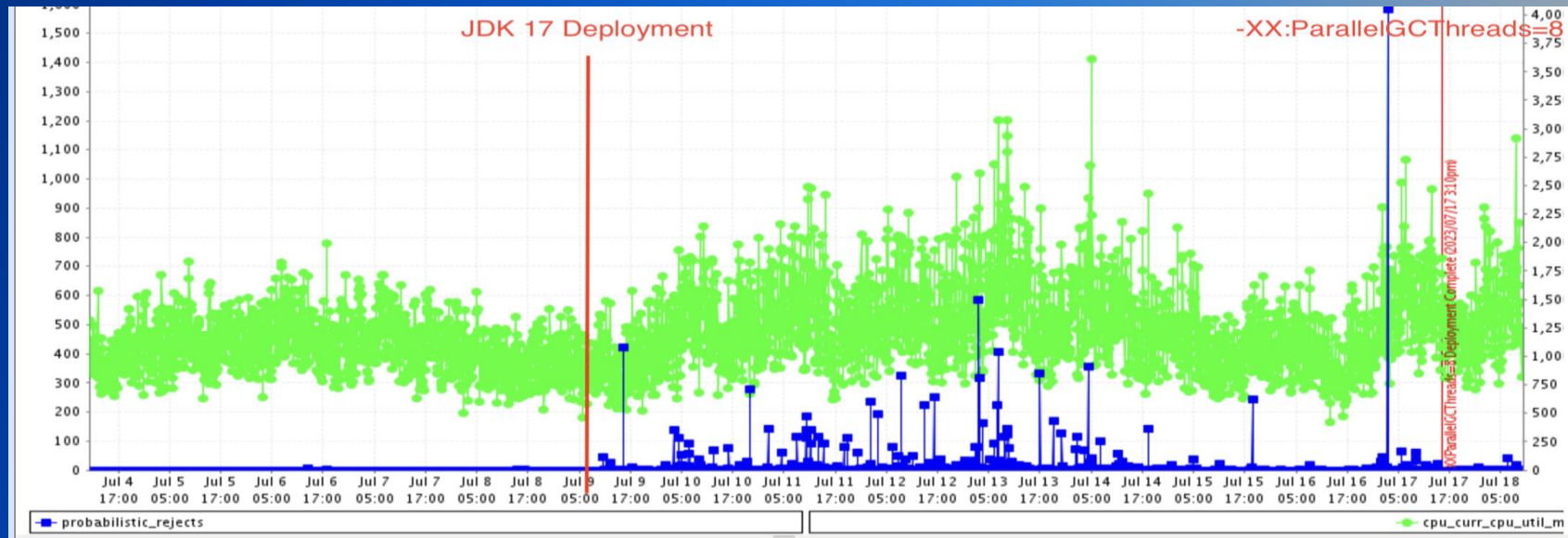
Previously accessible methods became unreferenceable in JDK 17.

```
java.lang.reflect.InaccessibleObjectException:  
Unable to make field private final {type} accessible:  
module java.base does not "opens {module}" to unnamed module {module}
```

- Test your application, fix flakey tests
- Scan logs for exceptions
- Gradual rollout

JDK 17 upgrade regression case study 2

- ParallelGC parallelized the reference processing using more CPU during full pause
- Triggered CPU spikes because failed to take into account other JVMs on host.
- Had to explicitly set `-XX:ParallelGCThreads` when sharing a host.



The good, the bad, and the ugly



The good, the bad, and the **ugly** (not really that ugly)



Java 17 still has bugs!

- [JDK-8305994](#): Guarantee eventual async monitor deflation
Moving monitor cleanup out of safepepoints (Java 12/15)
- [JDK-8313678](#): SymbolTable can leak Symbols during cleanup
Triggered by Groovy/LambdaForms (introduced in Java 12)
- We still find a couple of these every month!
This is why Update releases are so important to us and you.
- Both fixed upstream - backport to 17.

Java 21?

How many bugs are slated for a fix in Java 21.0.1?

Report bugs upstream!



Java 21?

How many bugs are slated for a fix in Java 21.0.1?

23/104/271 – 21.0.1

Report bugs upstream!



Java 21?

How many bugs are slated for a fix in Java 21.0.1?

23/104/271 – 21.0.1

300 – 21.0.2

274 – 21.0.3

297 – 21.0.4

Report bugs upstream!



JDK 17/21 Simplified Migration Guide

- Upgrade your dependencies!
 - Especially test frameworks: e.g. at least Mockito 5.x.
 - Libraries, e.g., need at least Lombok 1.18.22, Guice 5.1.0.
- Strong encapsulation.
 - No access to what were "public" JDK implementation classes.
- JDK 17/21 GC behavior is different (better!).
 - Default collector is G1 (not Parallel) starting in JDK 11.
 - Concurrent Mark-Sweep (CMS) collector is gone.
 - New collectors: Shenandoah and ZGC.

It gets complex when you have a lot of “stuff”

Developers

Services and Applications

Dependencies

Open Source / 3rd party

Internal / 1st Party

Custom tooling (build/deploy/test)

How do you future proof?

This could be controversial

- Use less dependencies, a lot less
- Remove unsupported dependencies
- Use libraries that only use public supported APIs
- If you can avoid
 - Mocks
 - Aspect oriented programming
- Relentlessly keep your code and dependencies up to date

How do you future proof?

- Test
 - Unit tests (non mocks)
 - Integration tests that cover all mainline, and critical edge case scenarios
 - Load tests – Observe latency, error rates, anomalies
 - Fix Flakey tests

How do you future proof?

- Metrics
 - Measure success rates, Latency, Memory, Memory/action, CPU
 - Observe and fix anomalies
- Log analytics, eliminate anomalies
- Pipelines, Beta, Fractional Deployments, Auto-Rollbacks, Canaries.

How do you future proof?

Don't leave broken windows behind.

Use the opportunity to modernize your application.

Upgrades become a lot less scary.



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 Code Transformation](#)

Getting Started with Amazon Q Developer Agent for code transformation

By Vinicius Senger

The screenshot displays the Amazon Q Developer interface within a code editor. The main window is titled "Code Transformation plan by Amazon Q". It shows a chat window on the left with instructions to enter the JDK path and a terminal window at the bottom showing the transformation progress. The transformation progress is divided into three steps: Step 1 (Update JDK version, dependencies and related code), Step 2 (Upgrade deprecated code), and Step 3 (Finalize code changes and generate transformation summary). The chat window also provides information about the transformation process, including the number of lines of code, dependencies to be replaced, and files to be changed.

Code Transformation plan by Amazon Q

Amazon Q reviewed your code and generated a transformation plan. Amazon Q will suggest code changes according to the plan, and you can review the updated code before accepting changes to your files.

- Lines of code in your application: 554
- Dependencies to be replaced: 7
- Deprecated code instances to be replaced: 0
- Files to be changed: 6

Planned transformation changes

Amazon Q will use the proposed changes as guidance during the transformation. The final code updates might differ from this plan. [Read more.](#)

Step 1 - Update JDK version, dependencies and related code

Amazon Q will attempt to update the JDK version and change the following dependencies and related code.

Transformation Progress Time elapsed: 5 min 42 sec

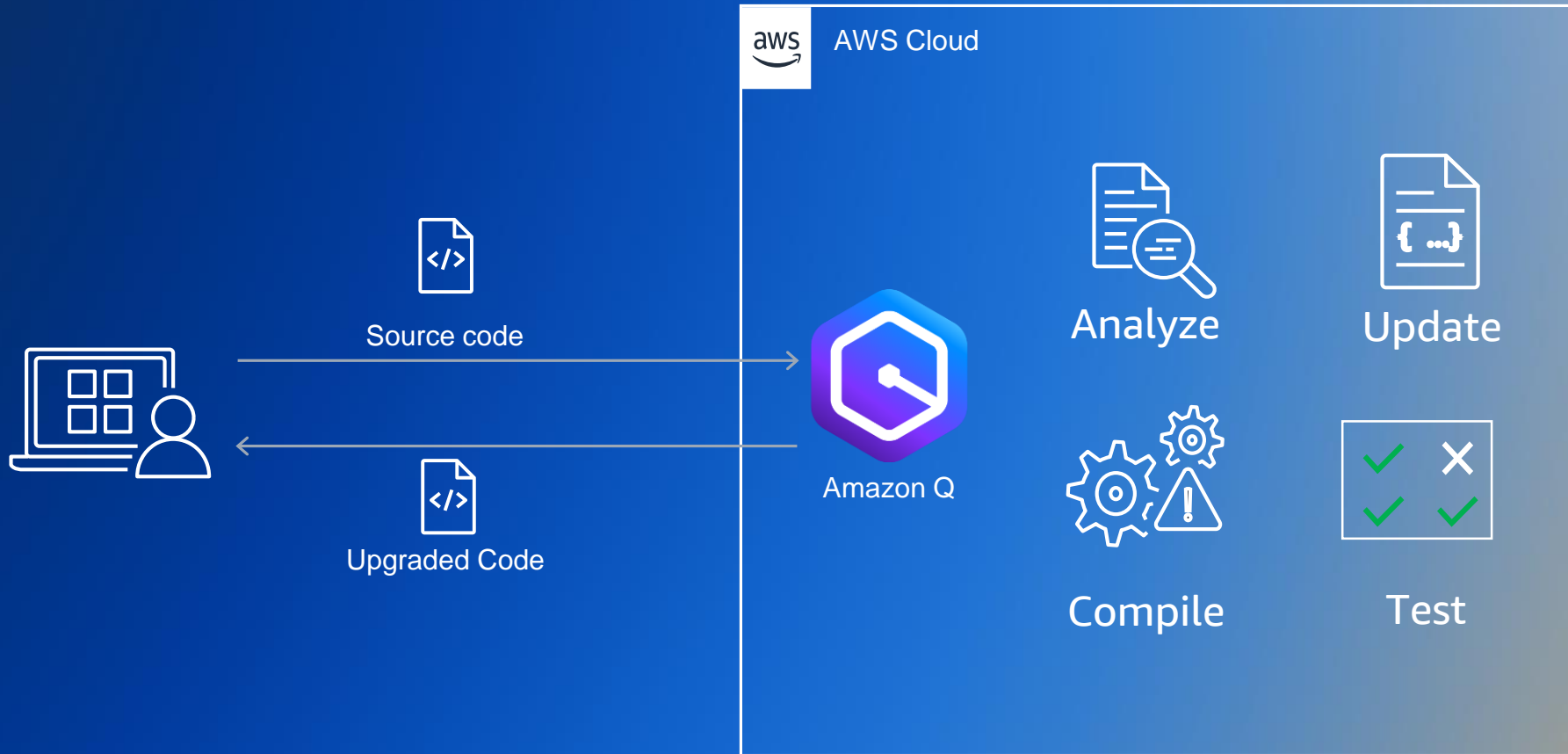
- ✓ Waiting for job to start
- ✓ Build uploaded code in secure build environment
- ✓ Generate transformation plan
- Transform your code to Java 17 using transformation plan
 - ✓ Step 1 - Update JDK version, dependencies and related code [finished on 5/7/2024, 11:59:24 AM] 1 min 33 sec
 - ✓ Step 2 - Upgrade deprecated code [finished on 5/7/2024, 12:00:07 PM] 42 sec

Step 3 - Finalize code changes and generate transformation summary

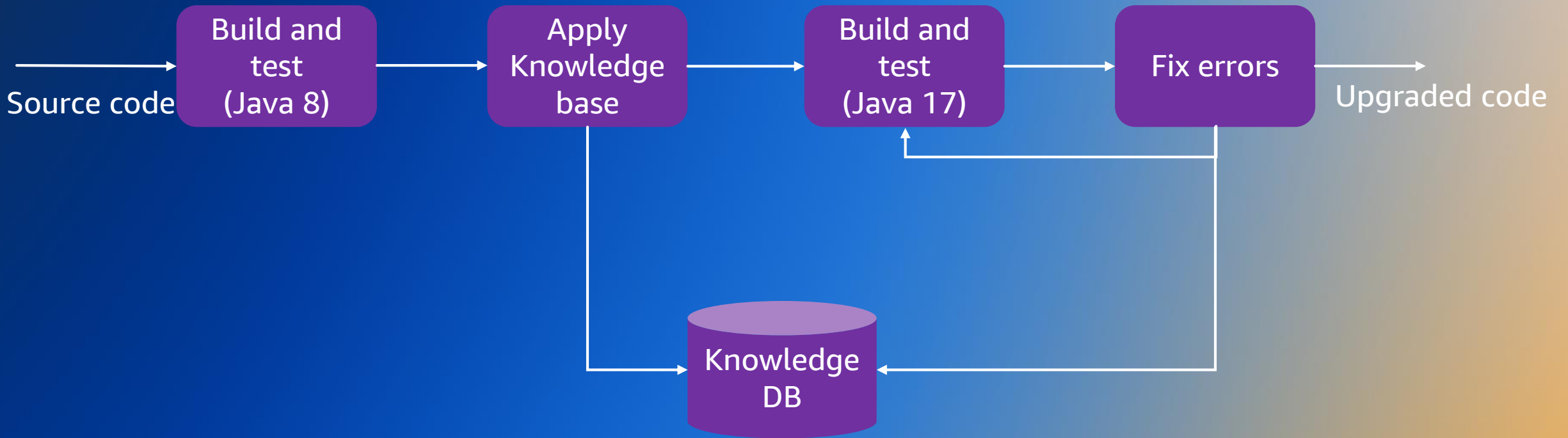
- Building in Java 17 environment
 - Transformation step started



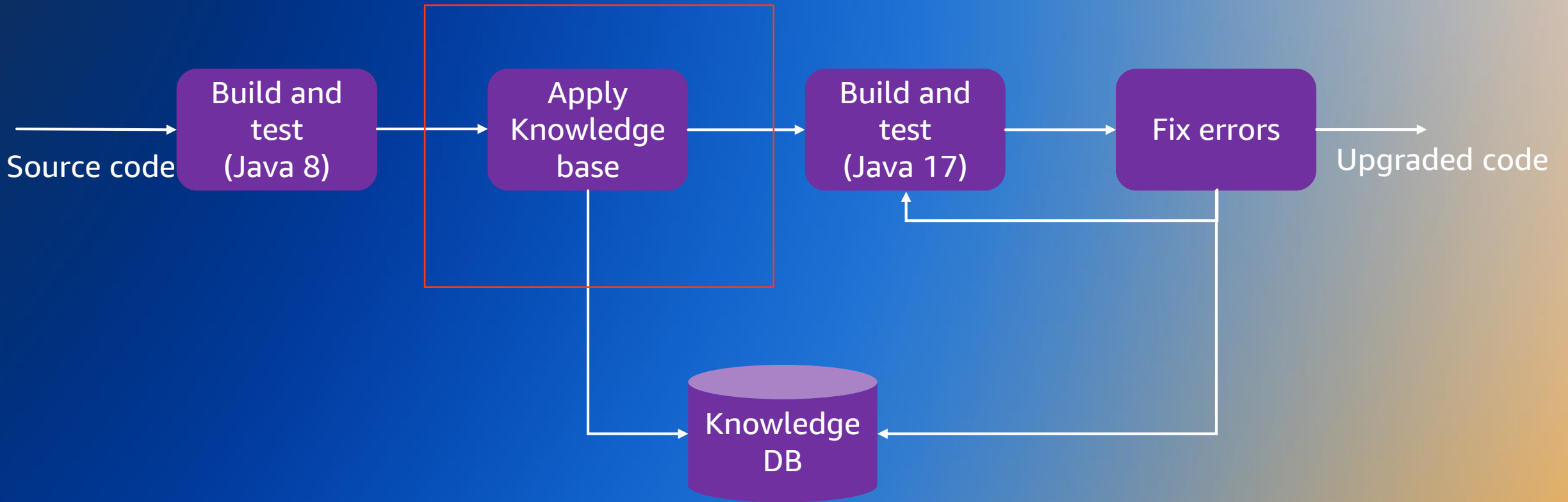
Introducing Amazon Q Code Transformation



Code Transformation: How it works



Code Transformation: How it works



Knowledge Base (1/2)

MINING RECOMMENDED VERSIONS OF DEPENDENCIES

```
<dependency>  
  <groupId>com.fasterxml.jackson.core</groupId>  
  <artifactId>jackson-core</artifactId>  
-   <version>2.9.4</version>  
+   <version>2.12.5</version>  
</dependency>
```



Knowledge Base (2/2)

MINING GENERAL COMPILATION ERRORS AND FIXES

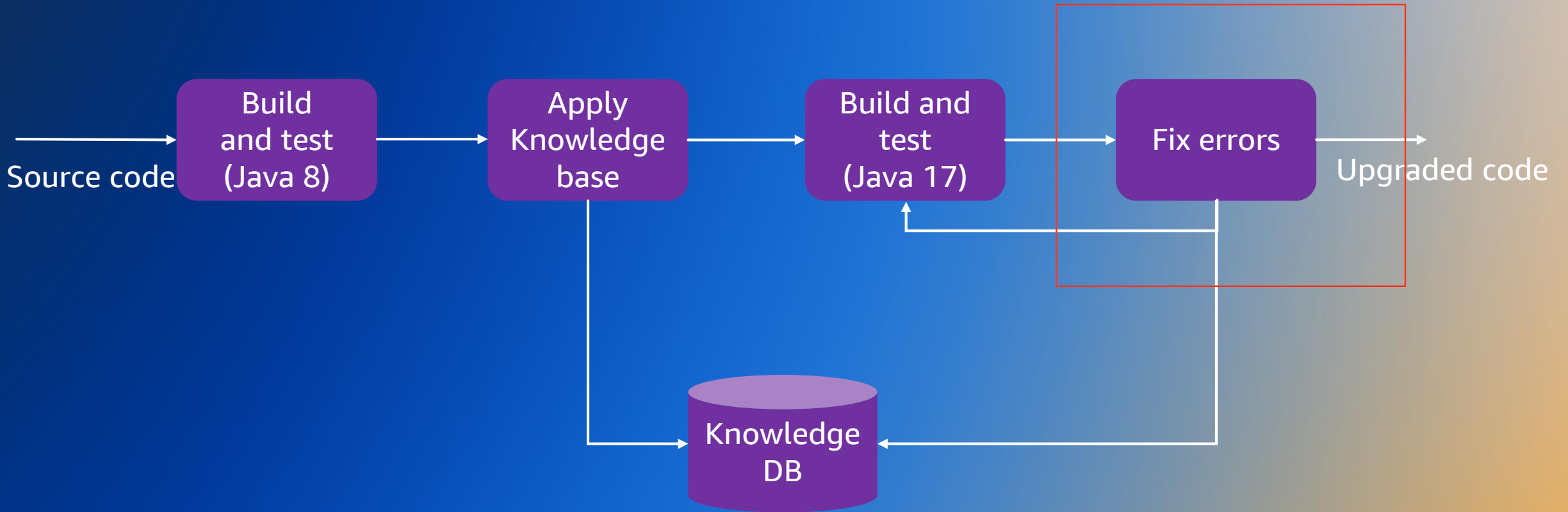
Error: ERROR] [...] package org.junit.runner does not exist

Fix: Add dependency Junit. Example:

```
<dependency>  
  <groupId>junit</groupId>  
  <artifactId>junit</artifactId>  
  <scope>test</scope>  
</dependency>
```



Code Transformation: How it works



Fixing errors using LLMs (1/2)

ERROR RESOLUTION AFTER KNOWLEDGE BASE APPLICATION

COMPILATION ERROR

src/main/java/com/intuit/benten/flickr/utils/SearchUtils.java:[16,11] exception
java.io.UnsupportedEncodingException is never thrown in body of corresponding try statement

```
public class SearchUtils {  
    public static String generateGoogleSearchUrl(String searchString) {  
        try {  
            String uriQuery = UriUtils.encode(searchString, StandardCharsets.UTF_8.name());  
            return SearchItems.GOOGLE_SEARCH_URL + uriQuery;  
        } catch (UnsupportedEncodingException e) {  
            e.printStackTrace();  
            return "";  
        }  
    }  
}
```

Fixing errors using LLMs

LLM CODE UPDATE THAT FIXES THE ERROR

```
public class SearchUtils {  
    public static String generateGoogleSearchUrl(String searchString) {  
        String uriQuery = UriUtils.encode(searchString, StandardCharsets.UTF_8.name());  
        return SearchItems.GOOGLE_SEARCH_URL + uriQuery;  
    }  
}
```





Thank you!