Shaping the future of Java, Faster

Georges Saab

Vice President, Java Platform Group

Oracle, Corp

Twitter: ***@gsaab***

Safe Harbor Statement

Communication

Java Magazine
250K+ subscribers

Community

Java User Groups
350+ worldwide

Collaboration

Java Champions
150+ worldwide

Contribution

OpenJDK
470 community participants

#1
Programming
Language

12 Million
Developers
Run Java

38 Billion
Active
Virtual Machines

21 Billion
Cloud Connected
Virtual Machines

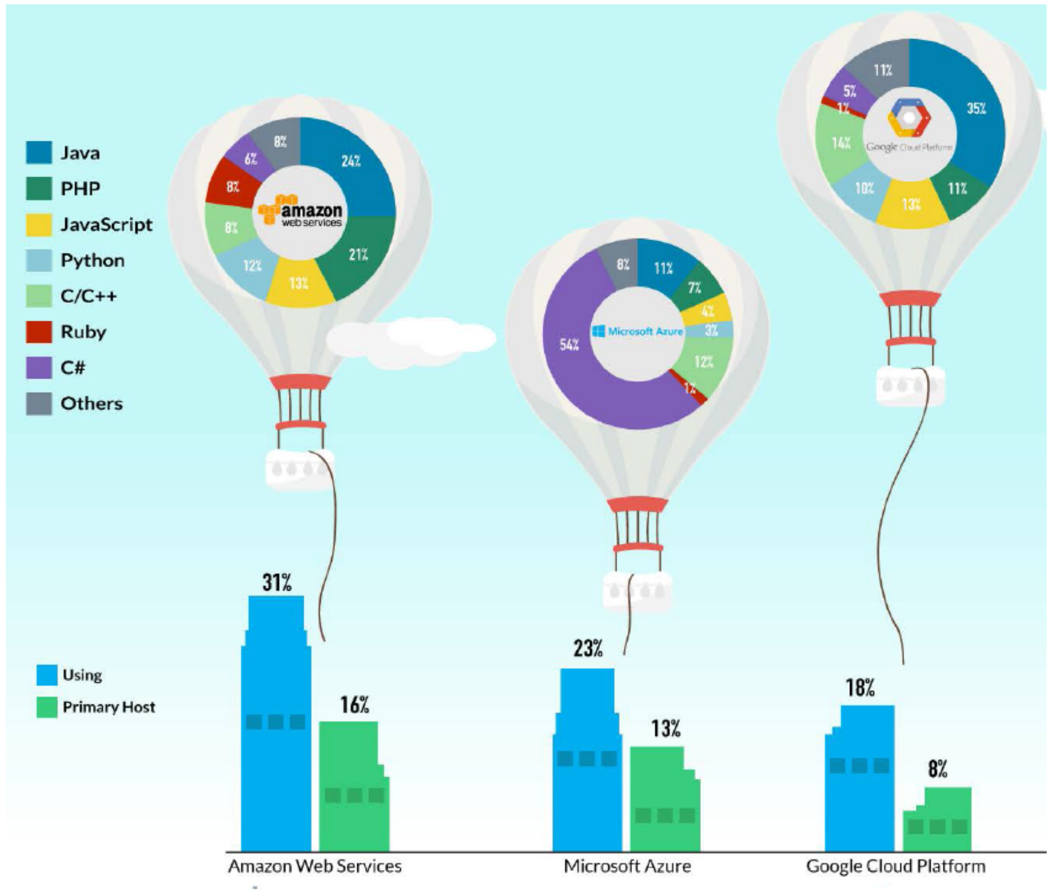
Open

Evolving

Nimble

Scalable

Java SE is #1 Runtime in the Cloud



Source: 2015 Vision Mobile

- #1 Deployment runtime on AWS and Google App Engine and #3 on MS Azure
- Java Runtime is the foundation of the Cloud IaaS, PaaS and SaaS

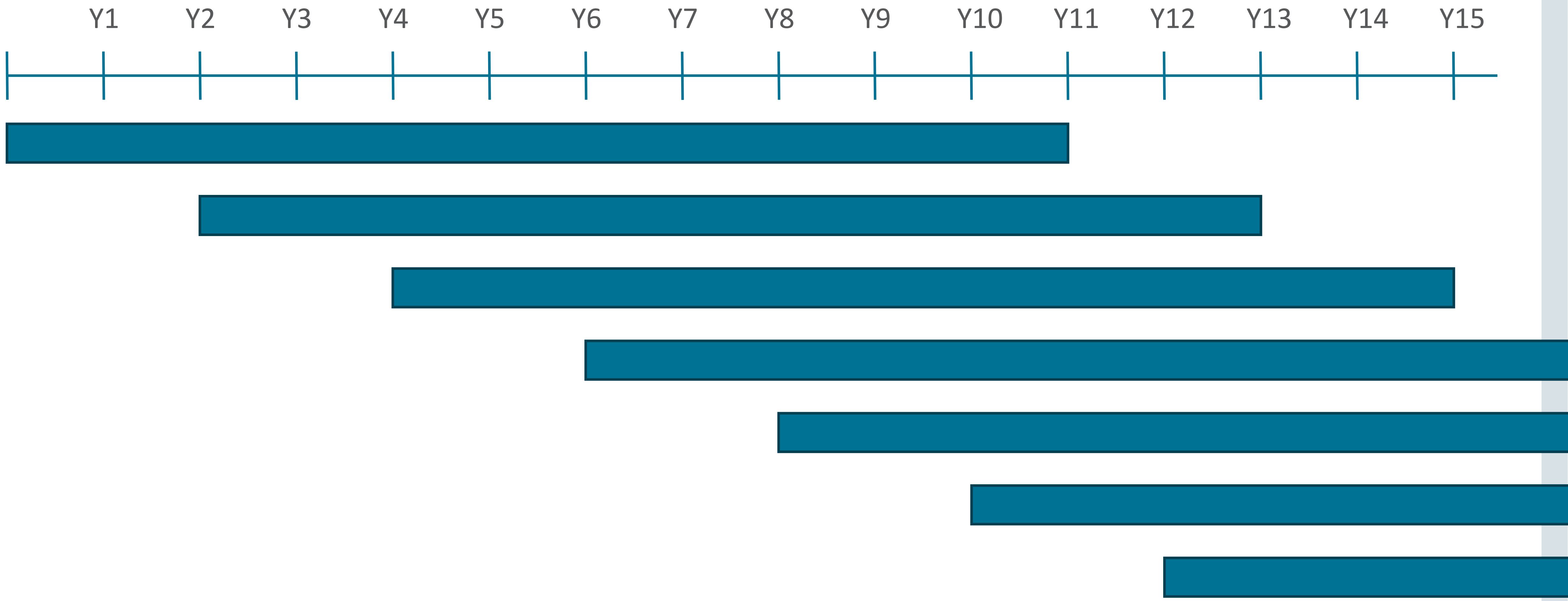
OpenJDK Platform Investments

- Security is our **#1 priority**
- Improving Java developer productivity and compatibility (Amber, Panama, Loom)
- Increasing density (Valhalla)
- Improving startup time (AOT, App CDS)
- Improving predictability (zGC, Shenandoah)
- Simplifying serviceability and profiling (JFR, JMC)

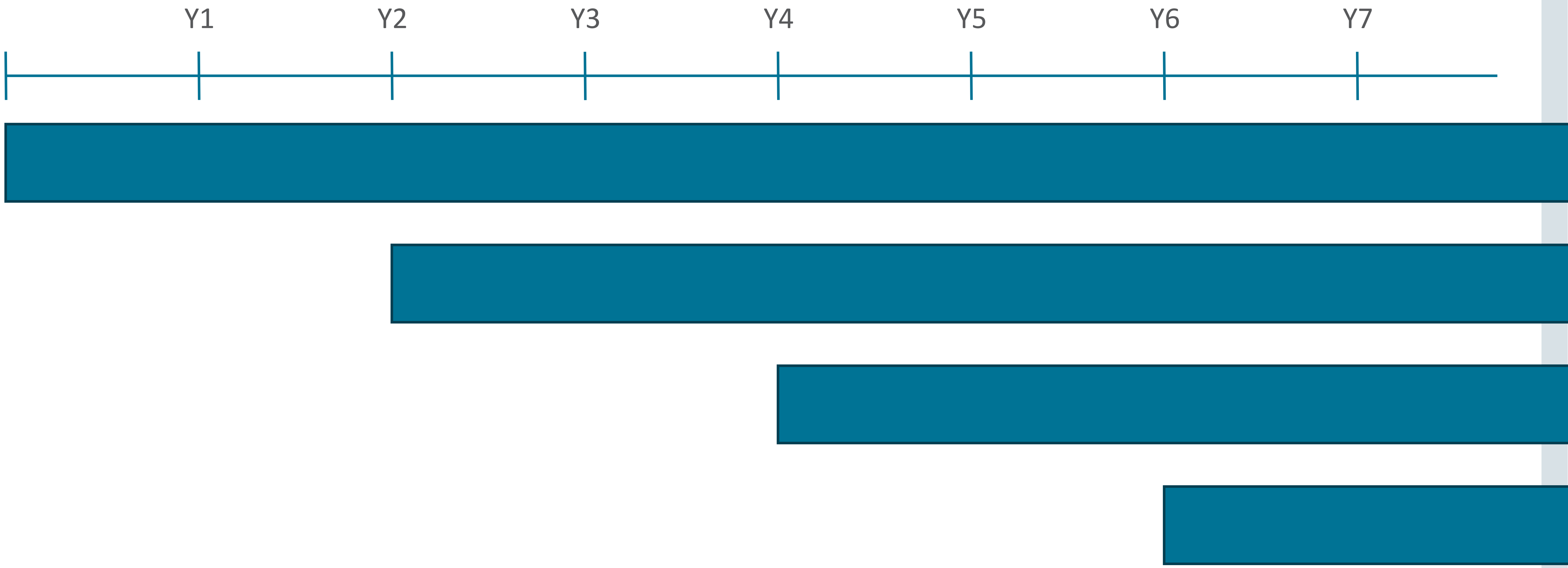
The New Release Model

No more limousines, think trains!

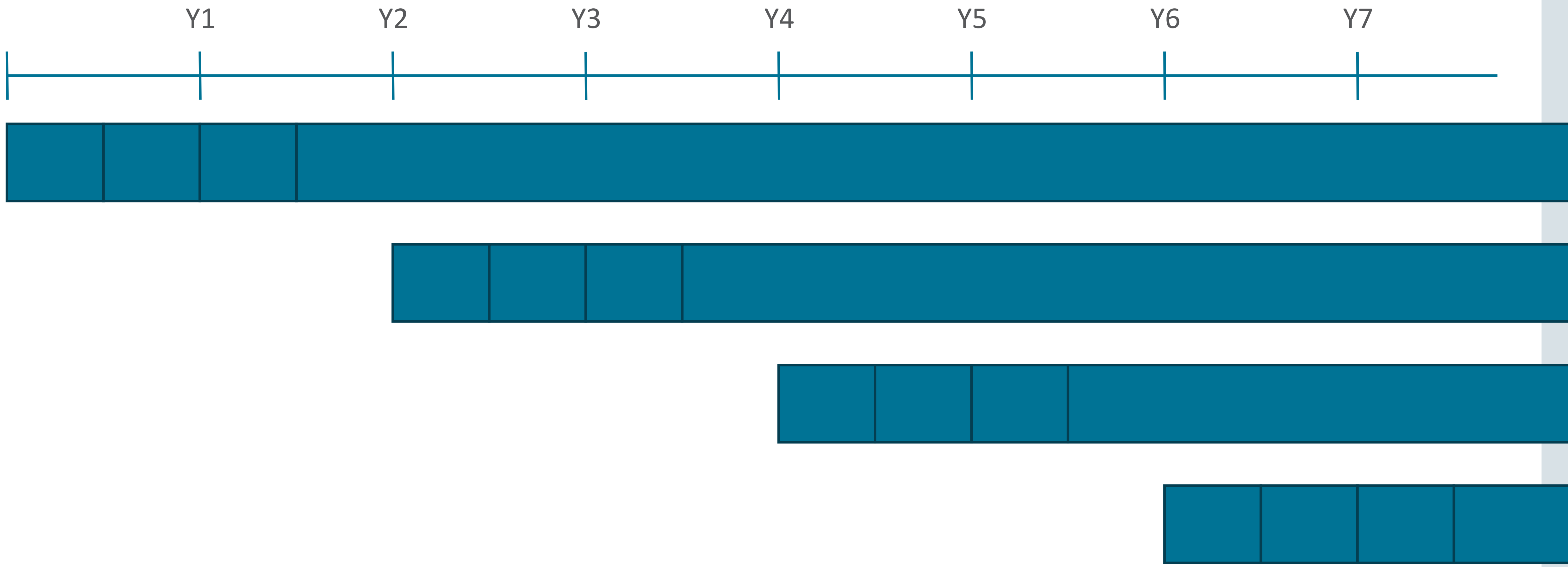
Previous JDK Release Model



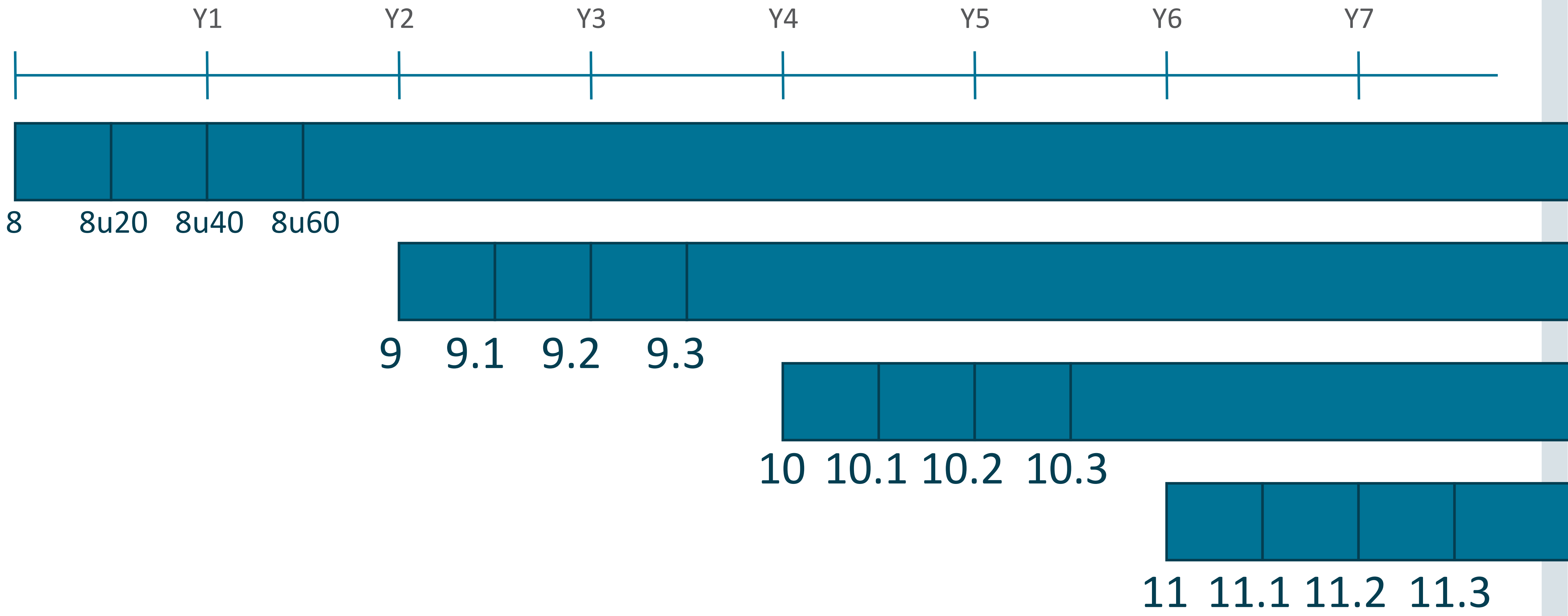
Previous JDK Release Model



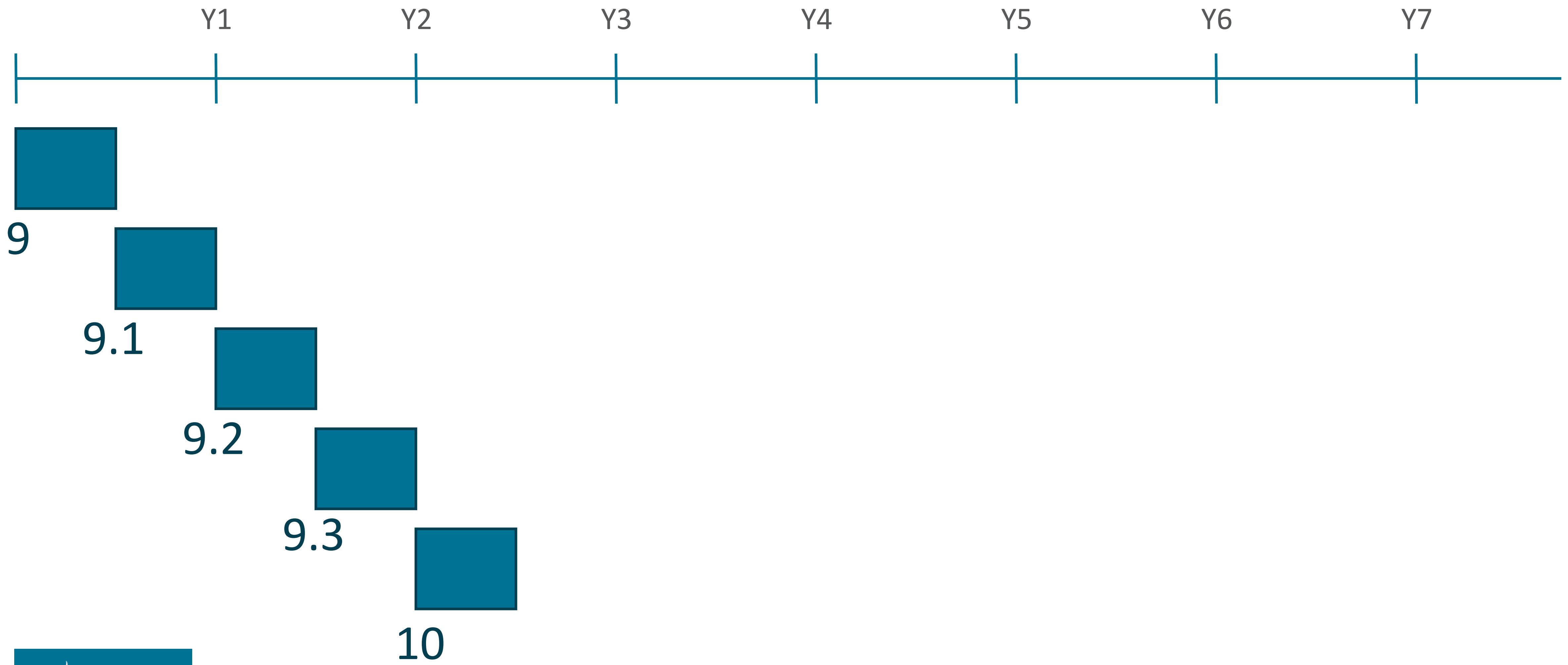
Previous JDK Release Model



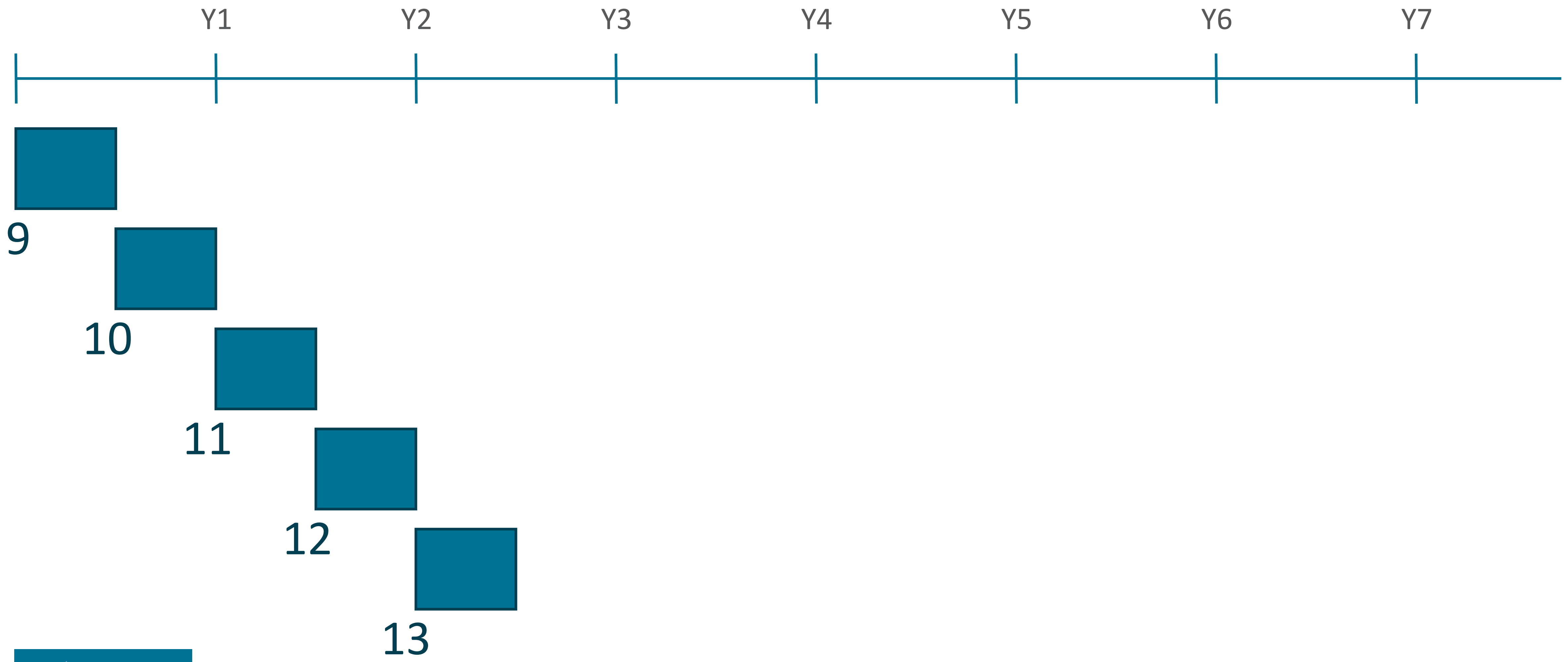
Previous JDK Release Model



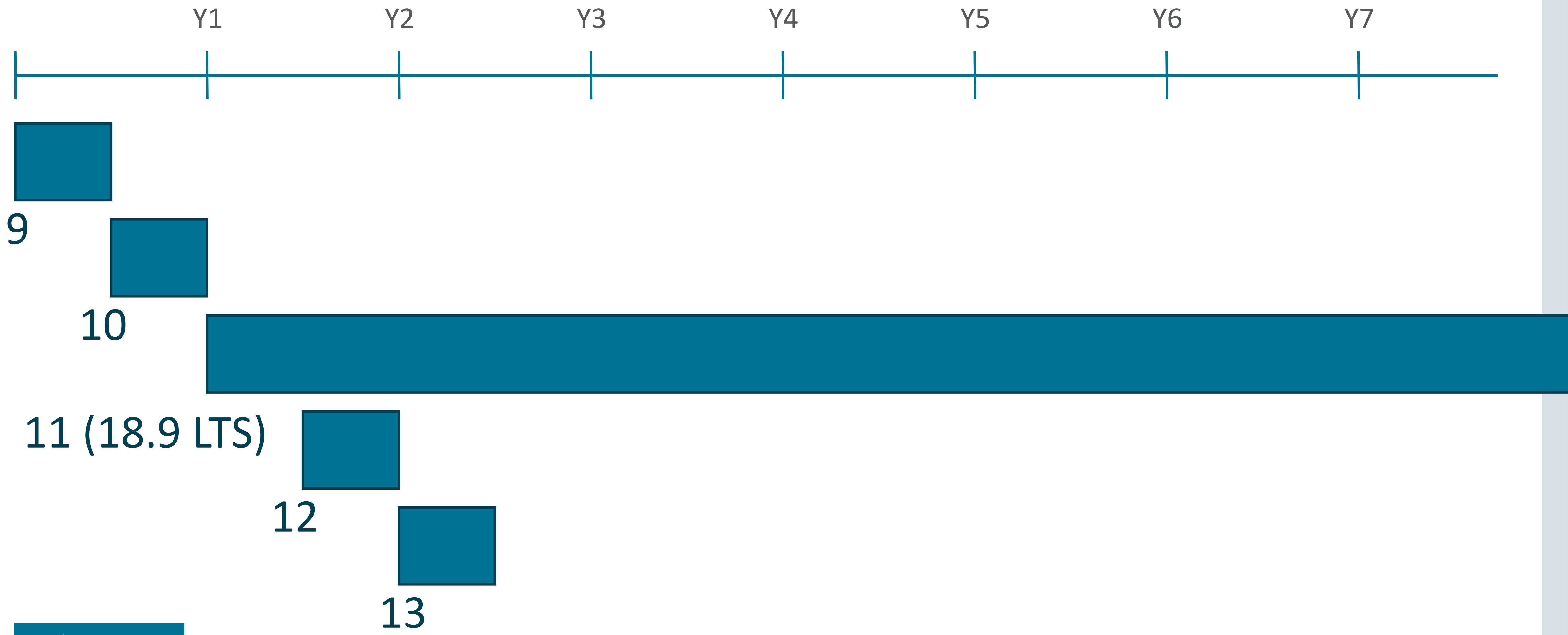
New JDK Release Model – Feature releases every 6 months



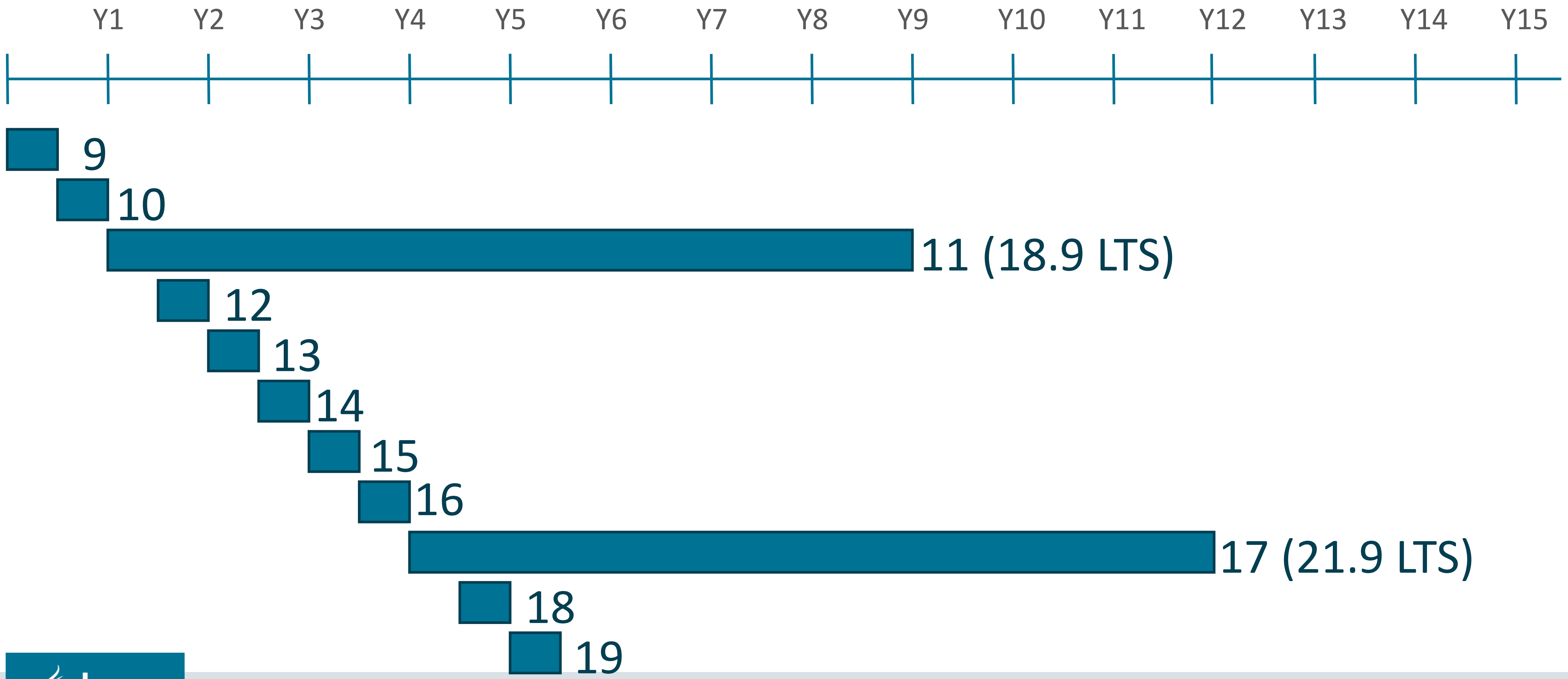
New JDK Release Model



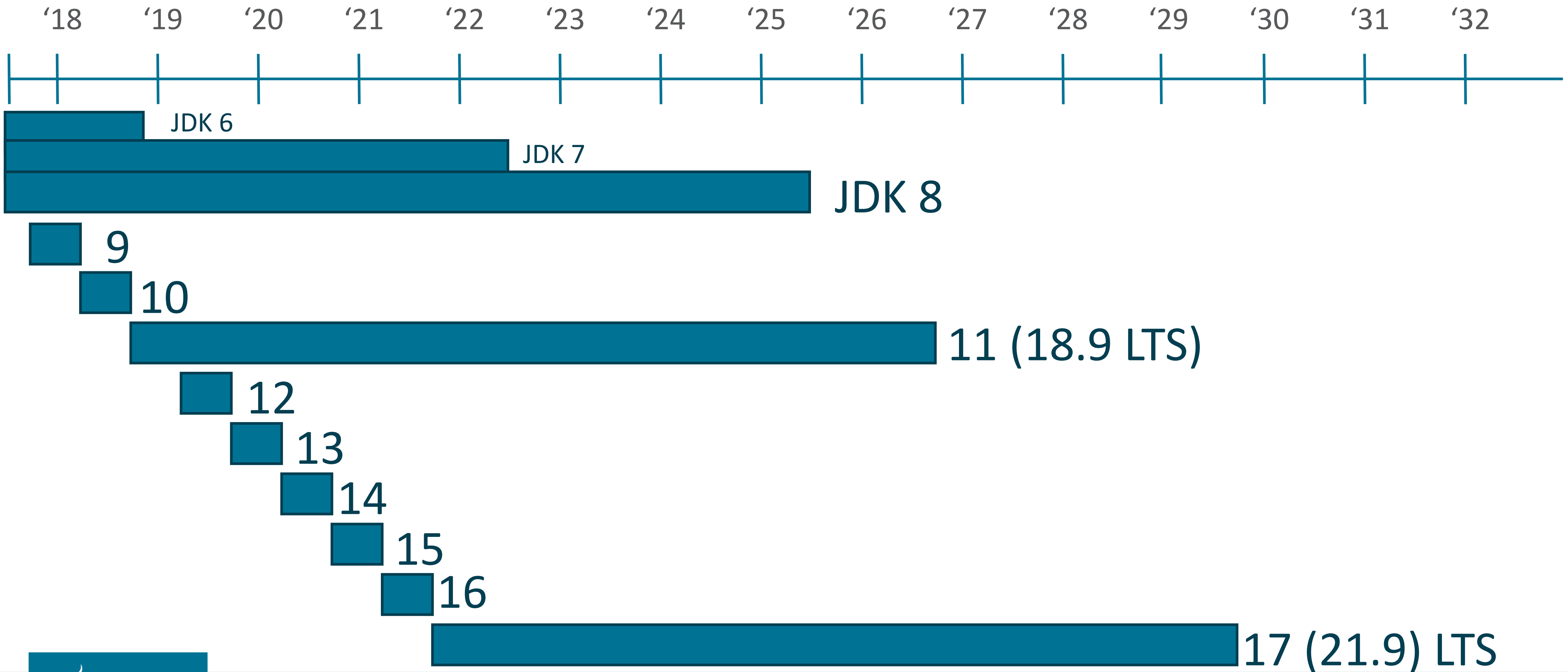
New JDK Release Model - LTS Releases



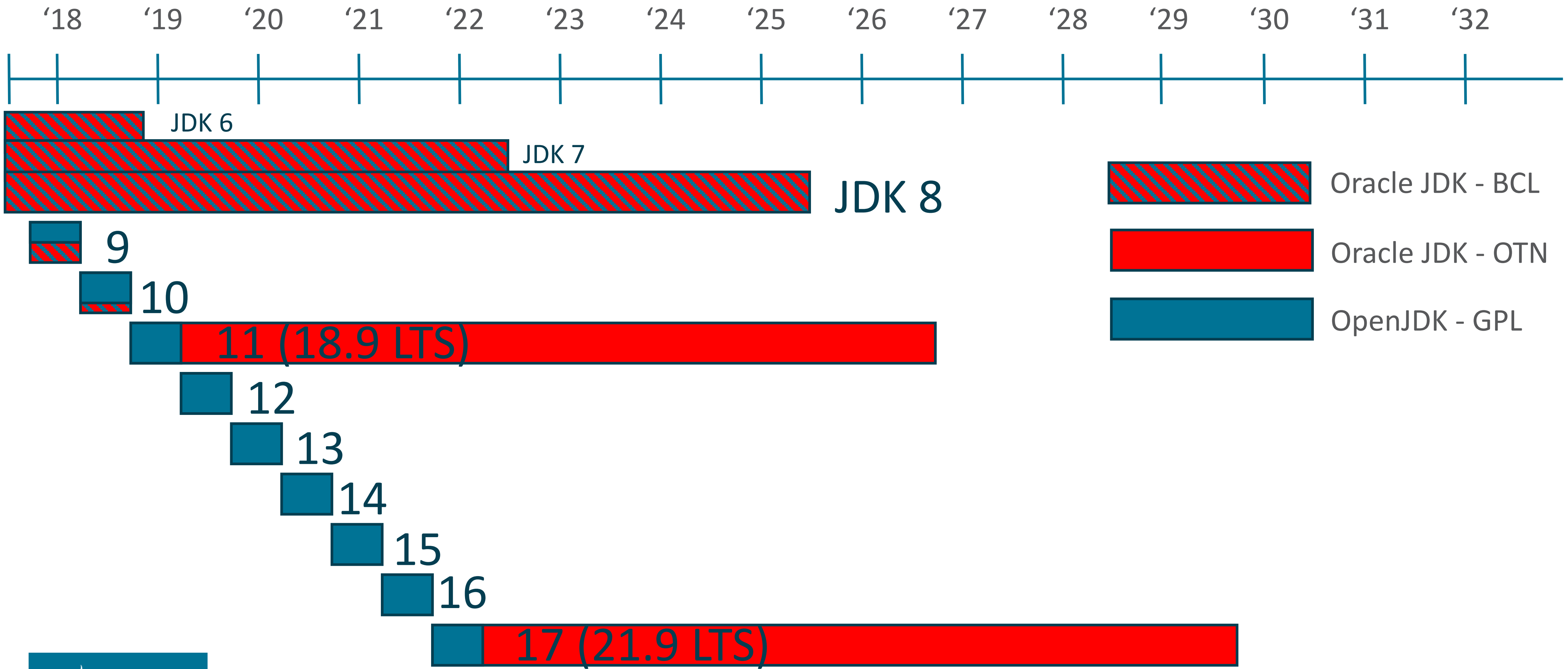
New JDK Release Model - LTS Every 3 years



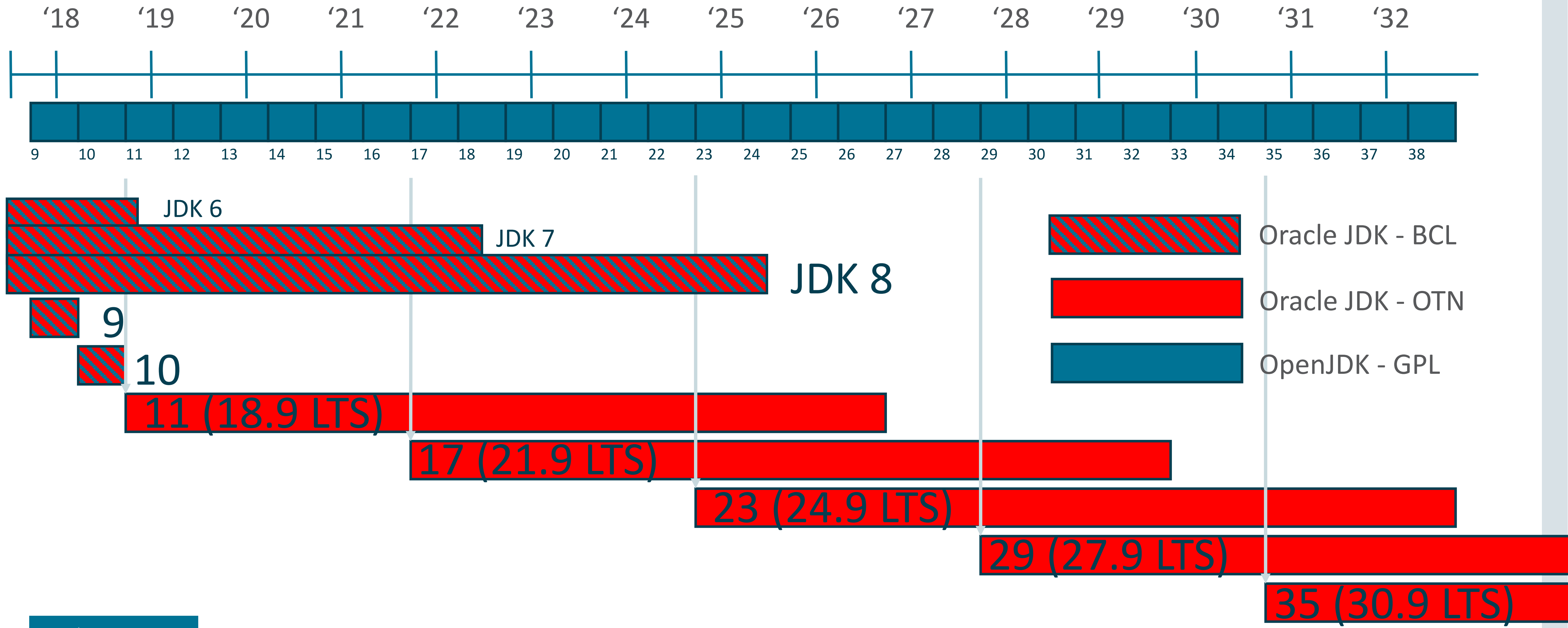
New JDK Release Model – Starting with JDK 9



Oracle JDK & OpenJDK

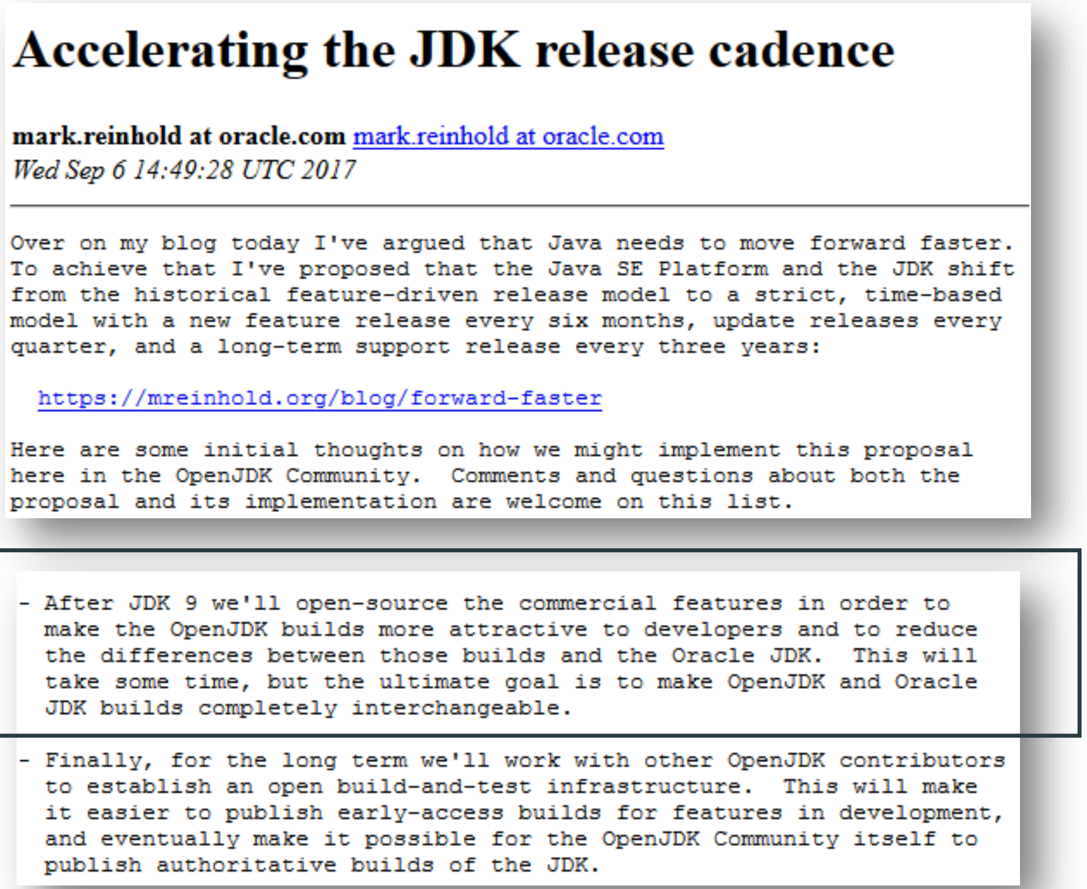


New JDK Release model



New OpenJDK binaries

Moving Java Forward Faster and more open! (*Opener*?)



Accelerating the JDK release cadence

mark.reinhold at oracle.com mark.reinhold at oracle.com
Wed Sep 6 14:49:28 UTC 2017

```
Over on my blog today I've argued that Java needs to move forward faster.
To achieve that I've proposed that the Java SE Platform and the JDK shift
from the historical feature-driven release model to a strict, time-based
model with a new feature release every six months, update releases every
quarter, and a long-term support release every three years:

  https://mreinhold.org/blog/forward-faster

Here are some initial thoughts on how we might implement this proposal
here in the OpenJDK Community.  Comments and questions about both the
proposal and its implementation are welcome on this list.
```

```
- After JDK 9 we'll open-source the commercial features in order to
  make the OpenJDK builds more attractive to developers and to reduce
  the differences between those builds and the Oracle JDK.  This will
  take some time, but the ultimate goal is to make OpenJDK and Oracle
  JDK builds completely interchangeable.

- Finally, for the long term we'll work with other OpenJDK contributors
  to establish an open build-and-test infrastructure.  This will make
  it easier to publish early-access builds for features in development,
  and eventually make it possible for the OpenJDK Community itself to
  publish authoritative builds of the JDK.
```

- New Java feature release will be made every 6 months
- **Oracle will now produce OpenJDK builds**
- **The new OpenJDK builds will be licensed under GPL V2**
  GNU General Public License Version 2 with Class Path Exception (GPL 2 with CPE)
- **Oracle will open source commercial features**
- **Oracle will work with other OpenJDK contributors to make the community infrastructure complete, modern and accessible**

URL: http://mail.openjdk.java.net/pipermail/discuss/2017-September/004281.html

From Oracle JDK to OpenJDK from Oracle

Java™ ORACLE → OpenJDK

What Is Being Open-Sourced in Java

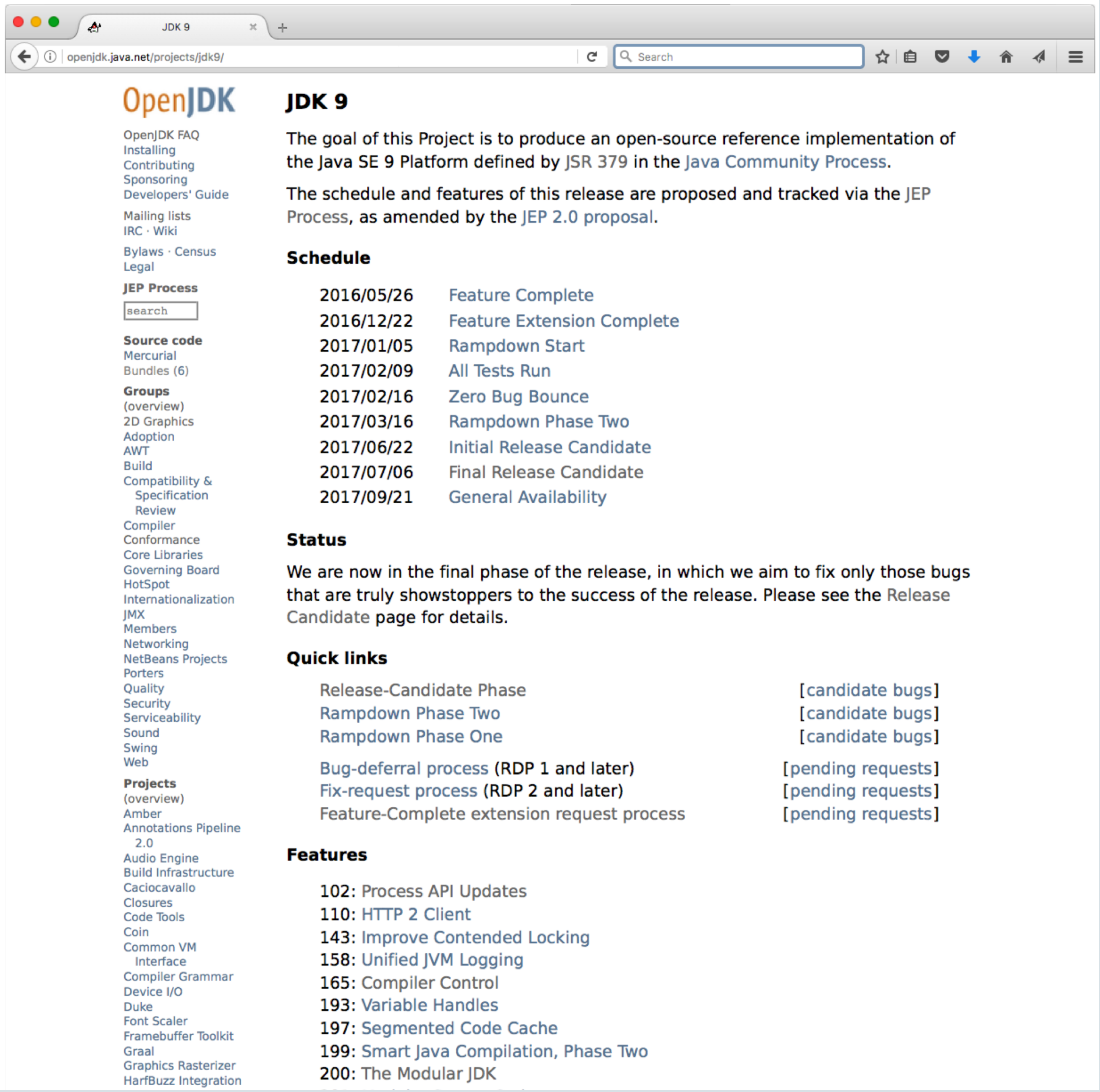
- **Java Mission Control**
  - Monitor and manage Java applications with minimal performance overhead.
- **Java Flight Recorder**
  - Collects diagnostic and profiling data about a running Java application.
- **Application Class Data Sharing**
  - Enables you to place classes from the standard extensions directories and the application class path in the shared archive.
- **Java Usage Tracker**
  - Tracks how the JRE's are being used in your systems.
- **Infrastructure**

Java 9

JDK 9

- Released September 2017
- Last Major Release
  - 100+ features

More information on any JEP:
http://openjdk.java.net/jeps/{JEP#}



OpenJDK

OpenJDK FAQ
Installing
Contributing
Sponsoring
Developers' Guide

Mailing lists
IRC · Wiki

Bylaws · Census
Legal

JEP Process

search

Source code
Mercurial
Bundles (6)

Groups
(overview)
2D Graphics
Adoption
AWT
Build
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
Internationalization
JMX
Members
Networking
NetBeans Projects
Porters
Quality
Security
Serviceability
Sound
Swing
Web

Projects
(overview)
Amber
Annotations Pipeline 2.0
Audio Engine
Build Infrastructure
Caciocavallo
Closures
Code Tools
Coin
Common VM Interface
Compiler Grammar
Device I/O
Duke
Font Scaler
Framebuffer Toolkit
Graal
Graphics Rasterizer
HarfBuzz Integration

JDK 9

The goal of this Project is to produce an open-source reference implementation of the Java SE 9 Platform defined by JSR 379 in the Java Community Process.

The schedule and features of this release are proposed and tracked via the JEP Process, as amended by the JEP 2.0 proposal.

Schedule

2016/05/26	Feature Complete
2016/12/22	Feature Extension Complete
2017/01/05	Rampdown Start
2017/02/09	All Tests Run
2017/02/16	Zero Bug Bounce
2017/03/16	Rampdown Phase Two
2017/06/22	Initial Release Candidate
2017/07/06	Final Release Candidate
2017/09/21	General Availability

Status

We are now in the final phase of the release, in which we aim to fix only those bugs that are truly showstoppers to the success of the release. Please see the Release Candidate page for details.

Quick links

Release-Candidate Phase	[candidate bugs]
Rampdown Phase Two	[candidate bugs]
Rampdown Phase One	[candidate bugs]
Bug-deferral process (RDP 1 and later)	[pending requests]
Fix-request process (RDP 2 and later)	[pending requests]
Feature-Complete extension request process	[pending requests]

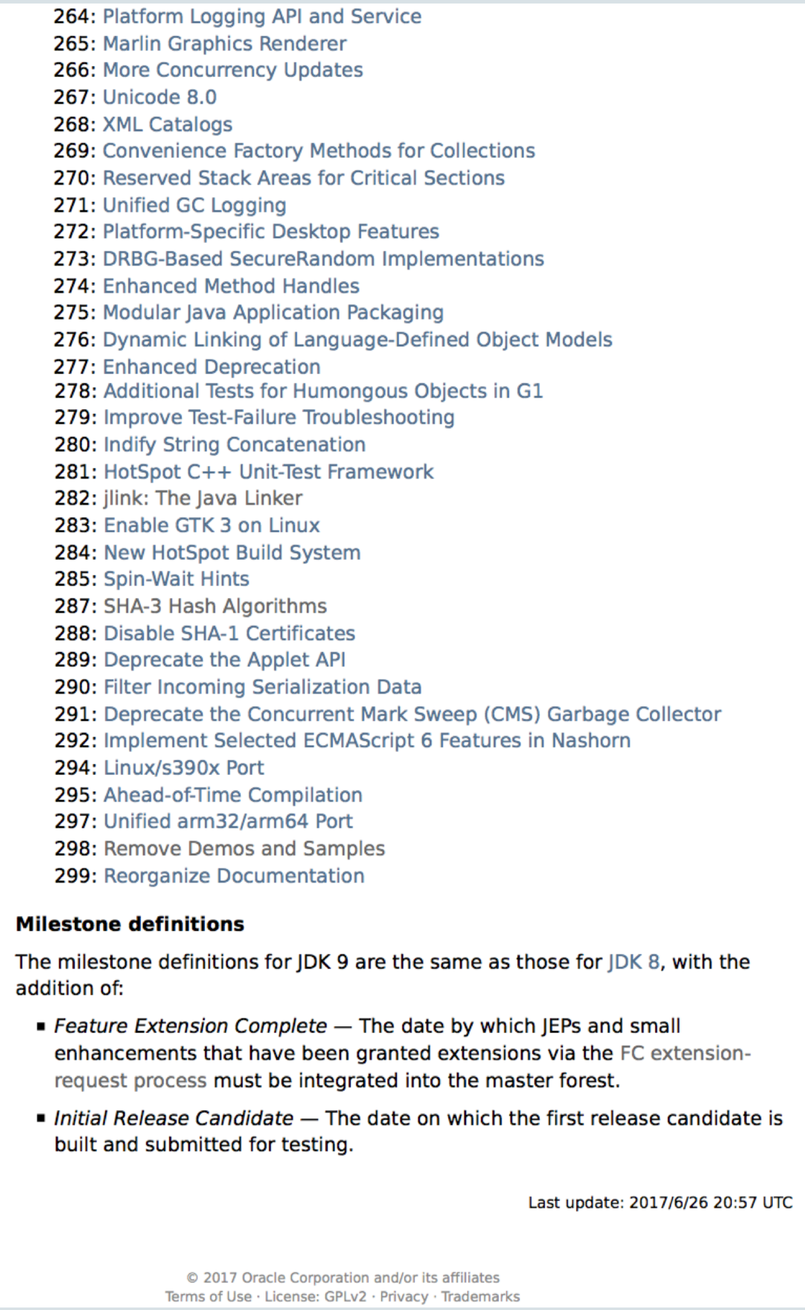
Features

102: Process API Updates
110: HTTP 2 Client
143: Improve Contended Locking
158: Unified JVM Logging
165: Compiler Control
193: Variable Handles
197: Segmented Code Cache
199: Smart Java Compilation, Phase Two
200: The Modular JDK

JDK 9

- Released September 2017
- Last Major Release
  - 100+ features

More information on any JEP:
http://openjdk.java.net/jeps/{JEP#}

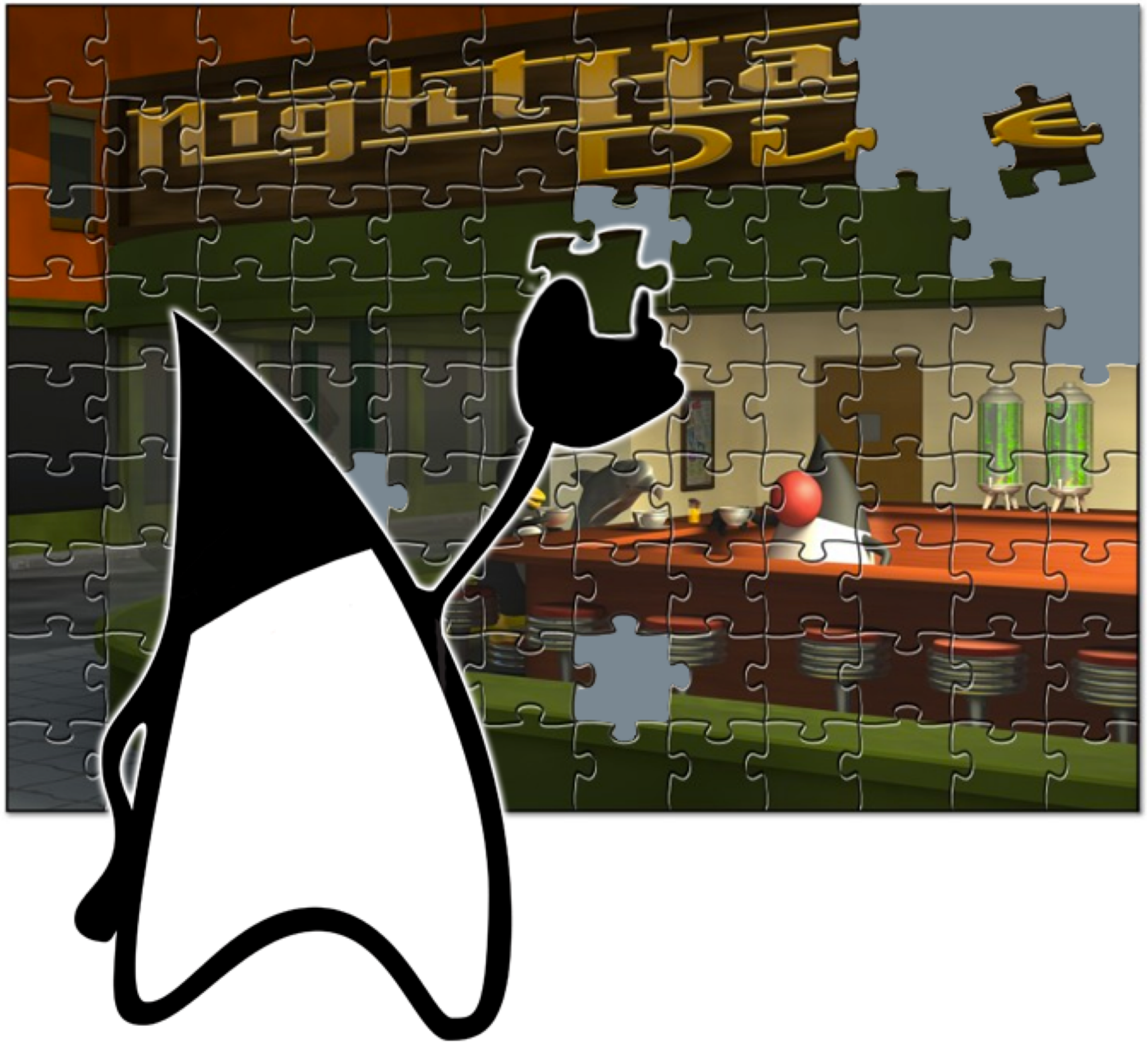

264: Platform Logging API and Service
265: Marlin Graphics Renderer
266: More Concurrency Updates
267: Unicode 8.0
268: XML Catalogs
269: Convenience Factory Methods for Collections
270: Reserved Stack Areas for Critical Sections
271: Unified GC Logging
272: Platform-Specific Desktop Features
273: DRBG-Based SecureRandom Implementations
274: Enhanced Method Handles
275: Modular Java Application Packaging
276: Dynamic Linking of Language-Defined Object Models
277: Enhanced Deprecation
278: Additional Tests for Humongous Objects in G1
279: Improve Test-Failure Troubleshooting
280: Indify String Concatenation
281: HotSpot C++ Unit-Test Framework
282: jlink: The Java Linker
283: Enable GTK 3 on Linux
284: New HotSpot Build System
285: Spin-Wait Hints
287: SHA-3 Hash Algorithms
288: Disable SHA-1 Certificates
289: Deprecate the Applet API
290: Filter Incoming Serialization Data
291: Deprecate the Concurrent Mark Sweep (CMS) Garbage Collector
292: Implement Selected ECMAScript 6 Features in Nashorn
294: Linux/s390x Port
295: Ahead-of-Time Compilation
297: Unified arm32/arm64 Port
298: Remove Demos and Samples
299: Reorganize Documentation

Milestone definitions

The milestone definitions for JDK 9 are the same as those for JDK 8, with the addition of:

- *Feature Extension Complete* — The date by which JEPs and small enhancements that have been granted extensions via the FC extension-request process must be integrated into the master forest.
- *Initial Release Candidate* — The date on which the first release candidate is built and submitted for testing.

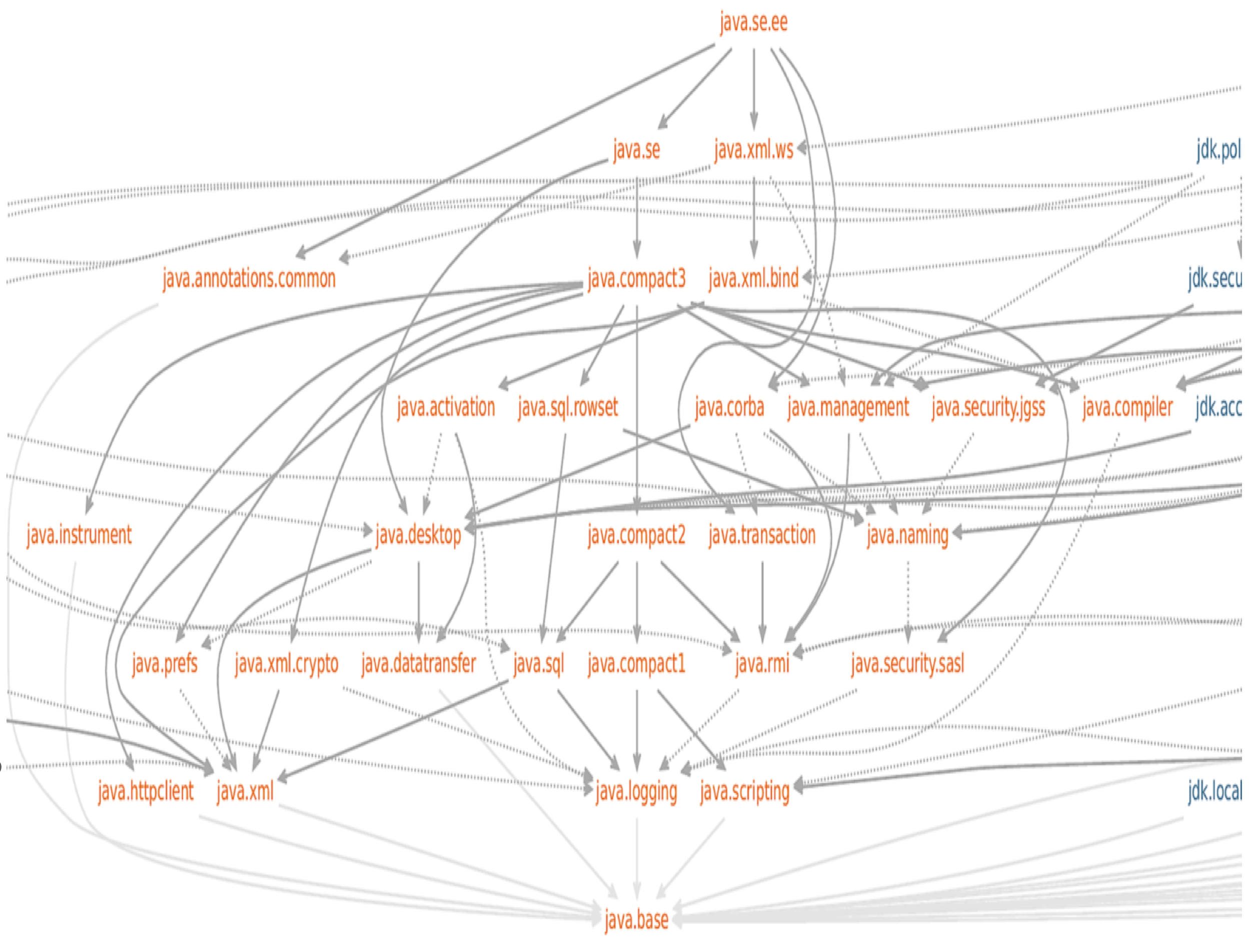
Last update: 2017/6/26 20:57 UTC


nightHawk DU

Project Jigsaw

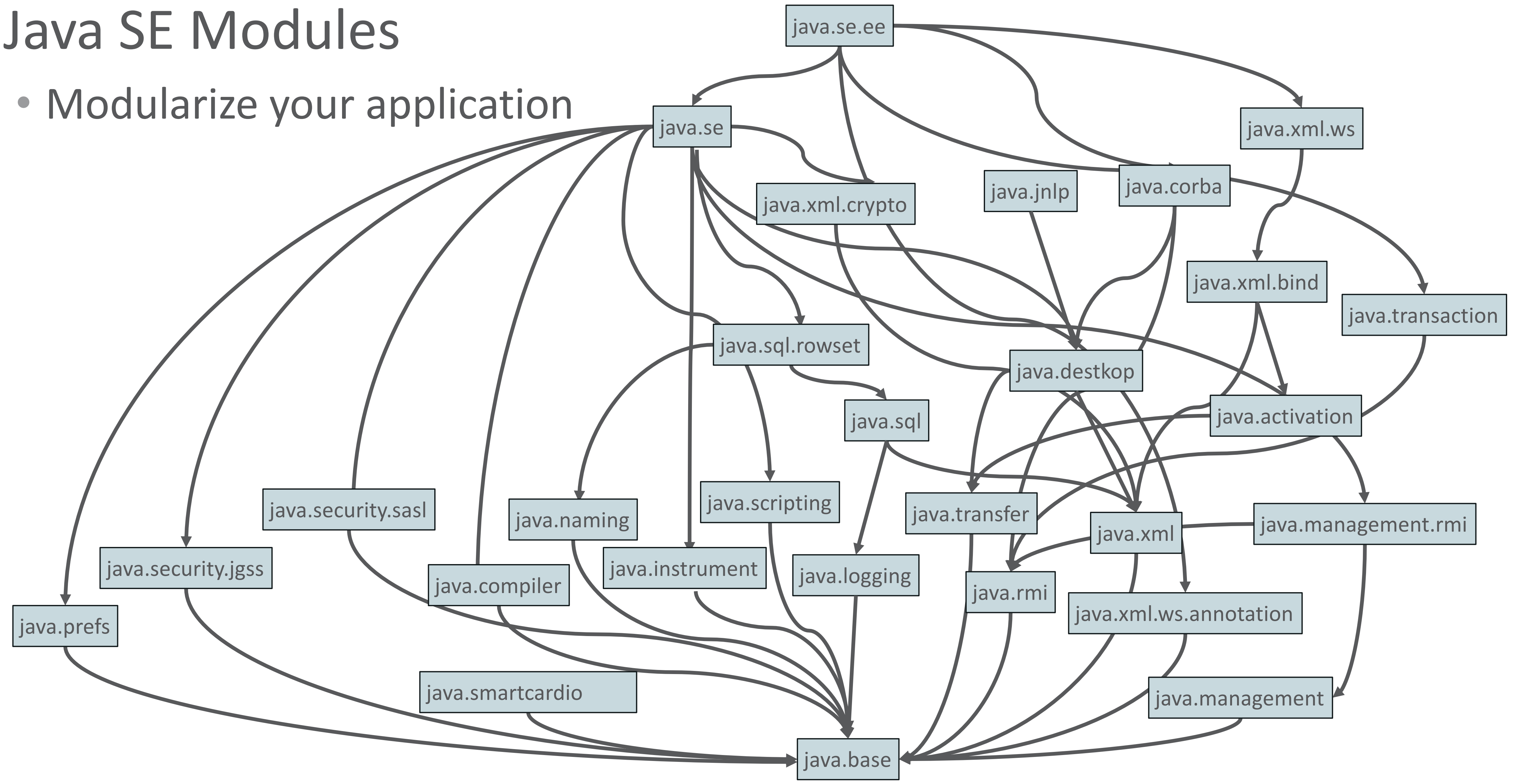
Modularize the Java Platform

- JEP 261: Module System
- JEP 200: The Modular JDK
- JEP 201: Modular Source Code
- JEP 220: Modular Run-Time Images
- Plus
  - JEP 260: Encapsulate Most Internal APIs
  - JEP 282: jlink: The Java Linker

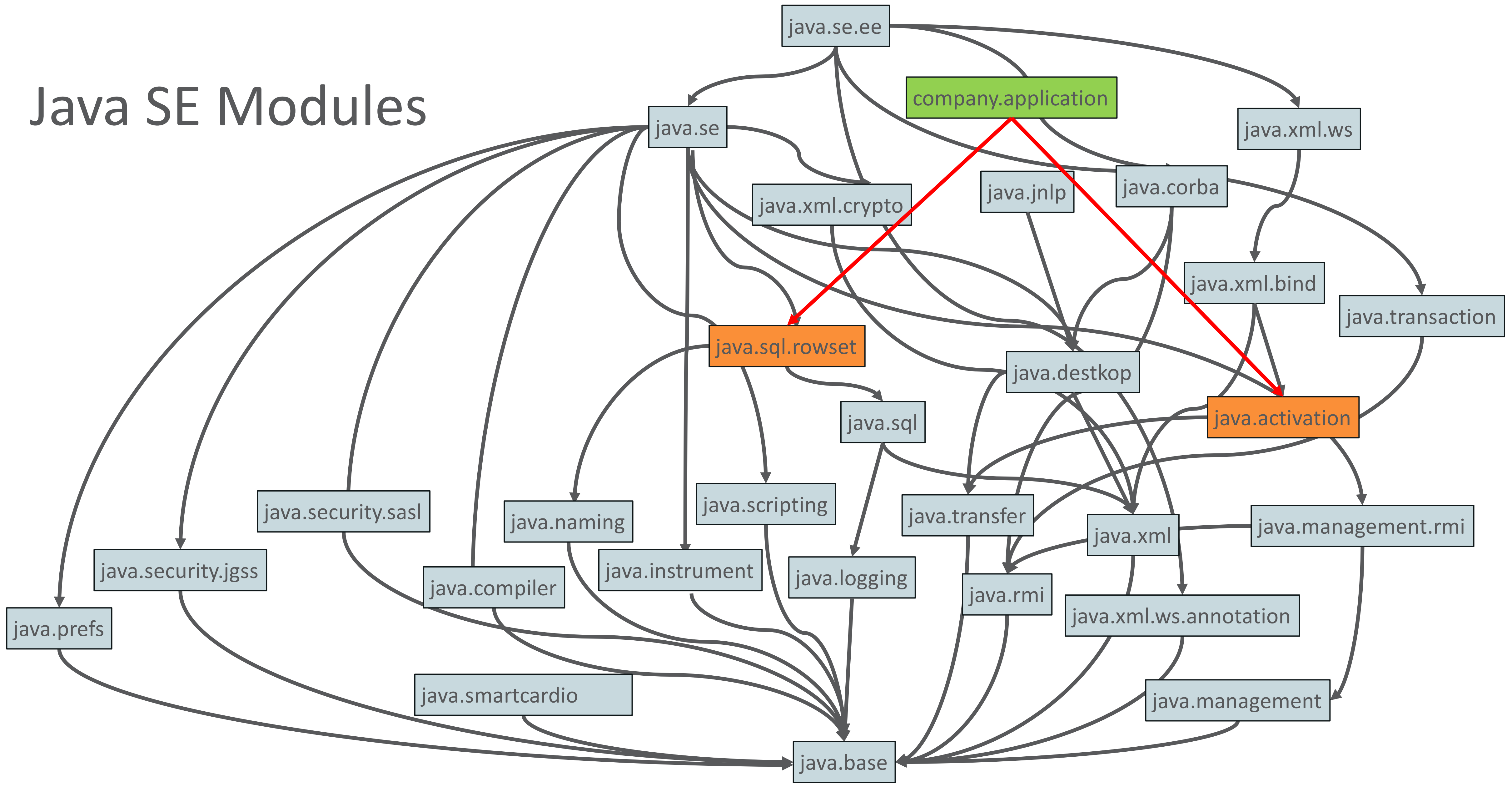


Java SE Modules

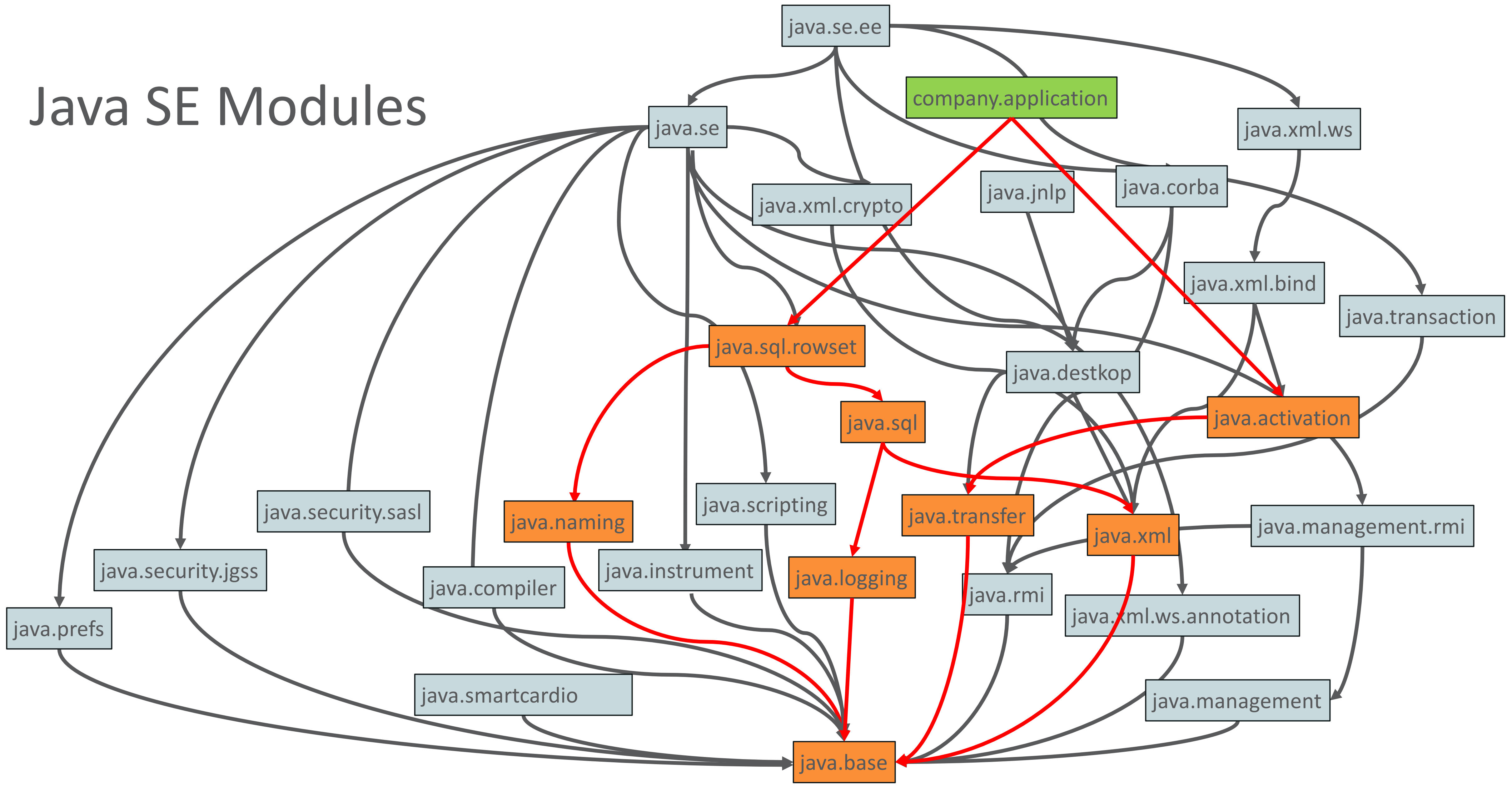
- Modularize your application



Java SE Modules



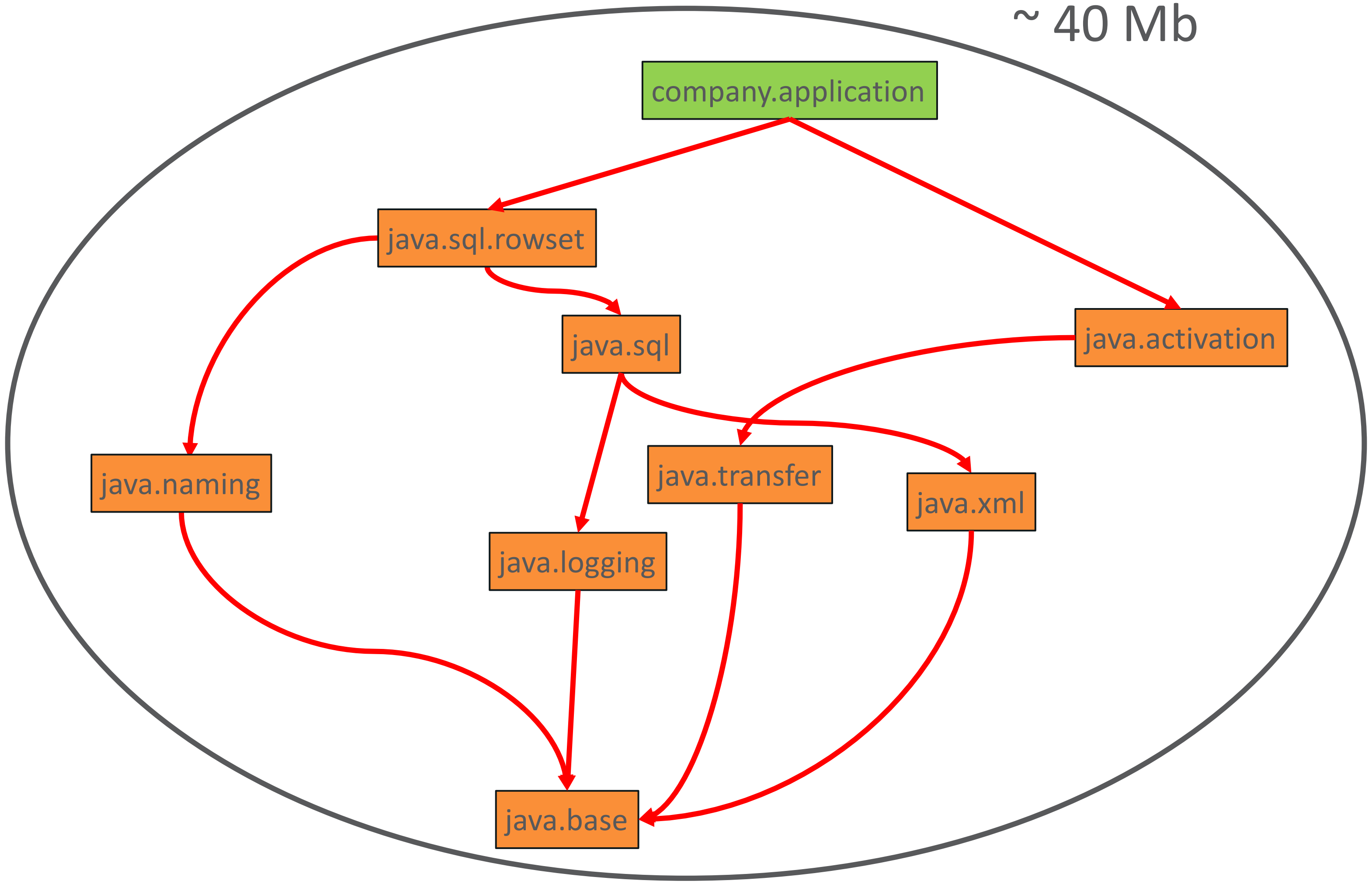
Java SE Modules



Java Custom Runtime

- Includes the Modular Application

Custom Image
~ 40 Mb



JDK 9 Jigsaw Security

Module boundaries enforced by the JVM

- • Encapsulate implementation--internal classes inside modules
- – Share them with other implementation modules only as needed
- Massive maintainability improvement
- Simpler compatibility upgrade path
  - **We and You** can now hide and preclude access to unsupported internal APIs and implementation
- Will also significantly improve Security
  - Enable developers to create customized runtime that removed unused security sensitive APIs

Java 9: Ahead of Time (AOT) Java Compiler

- The unification of static and dynamic compilation
  - Static compilation – faster startup, lower memory usage, but limited in optimizing code generation
  - Dynamic profiling based compilation – slow startup but optimum code generation
- New AOT Compiler to statically compile Java classes to native shared libraries
  - Reduces startup time **and** improve density to close the gap against native service
- Compile Java packages to native shared libraries
- JVM was modified to load native shared libraries on startup
  - JVM internal structures, which describe compiled code, are split to describe compiled code in code cache and in a shared library
  - AOT compiled code is dynamically linked to Java methods after its class is initialized

New world, new deployment option

Containers

In a World of Containers We Expect…

- Safety and security becoming increasingly more important
- Sprawl
  - Many instances
  - Mix of different applications
  - Heterogeneous machines
  - Heterogeneous container configurations

Java in a World of Containers

Java's characteristics make it ideal for a container environment

- Managed language/runtime
- Hardware and operating system agnostic
- Safety and security enforced by JVM
- Reliable: Compatibility is a key design goal
- Runtime adaptive: JVM ensures stable execution when environment changes
- Rich ecosystem

Java in a World of Containers

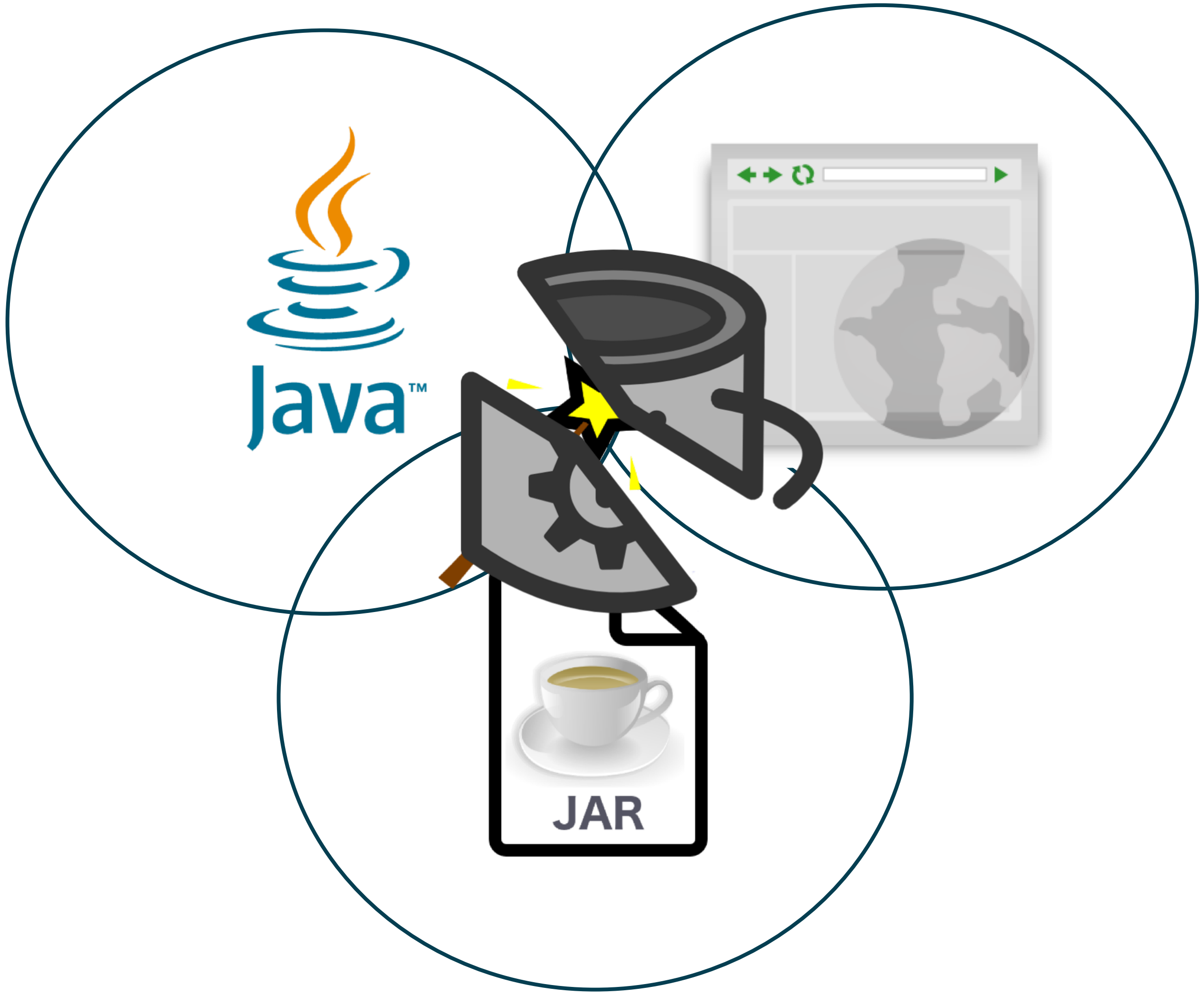
Java's characteristics make it ideal for a container environment

- Managed language/runtime
- Hardware and operating system agnostic
- Safety and security enforced by JVM
- Reliable: Compatibility is a key design goal
- Runtime adaptive: JVM ensures stable execution when environment changes
- Rich ecosystem
-

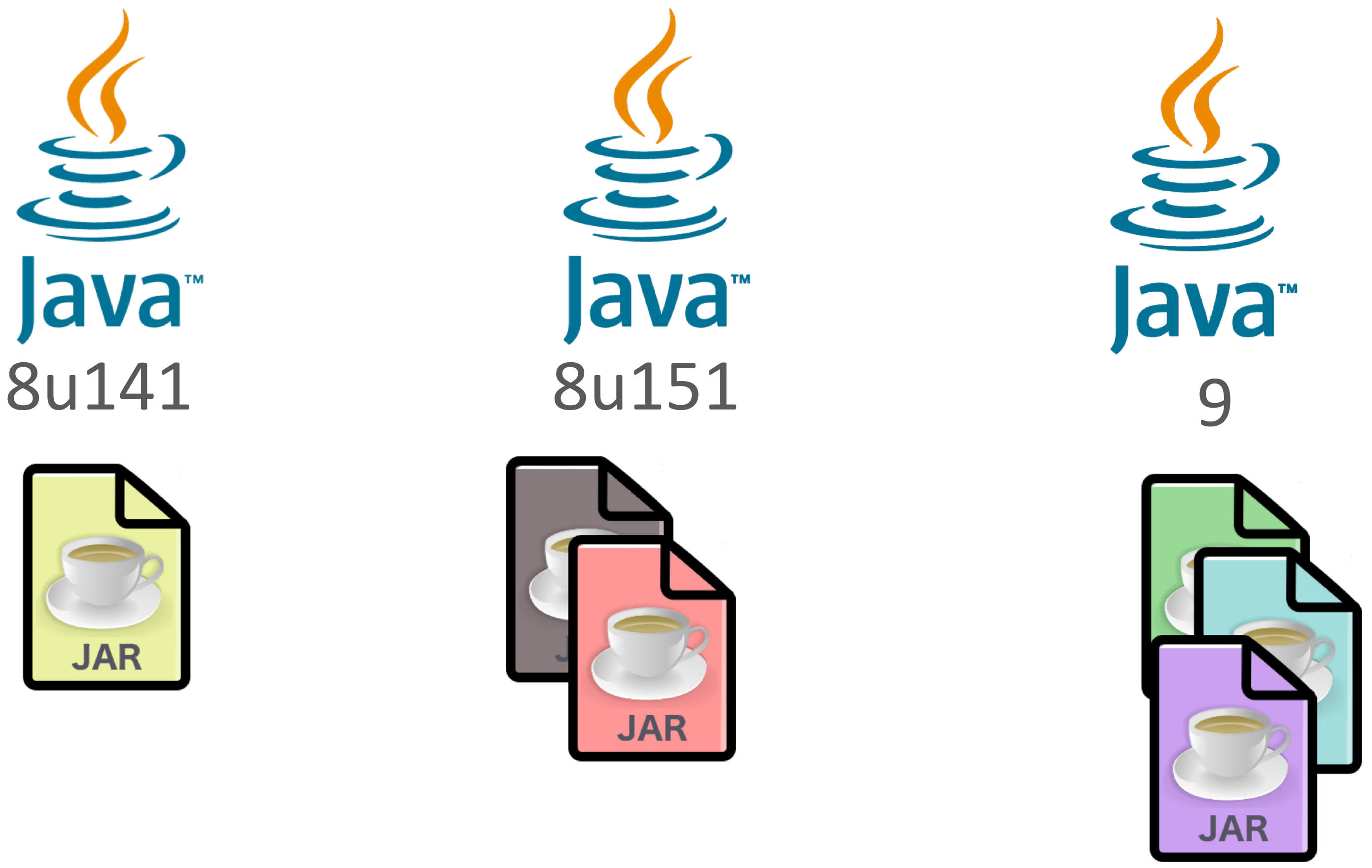
New world, new deployment option

Modern Browsers

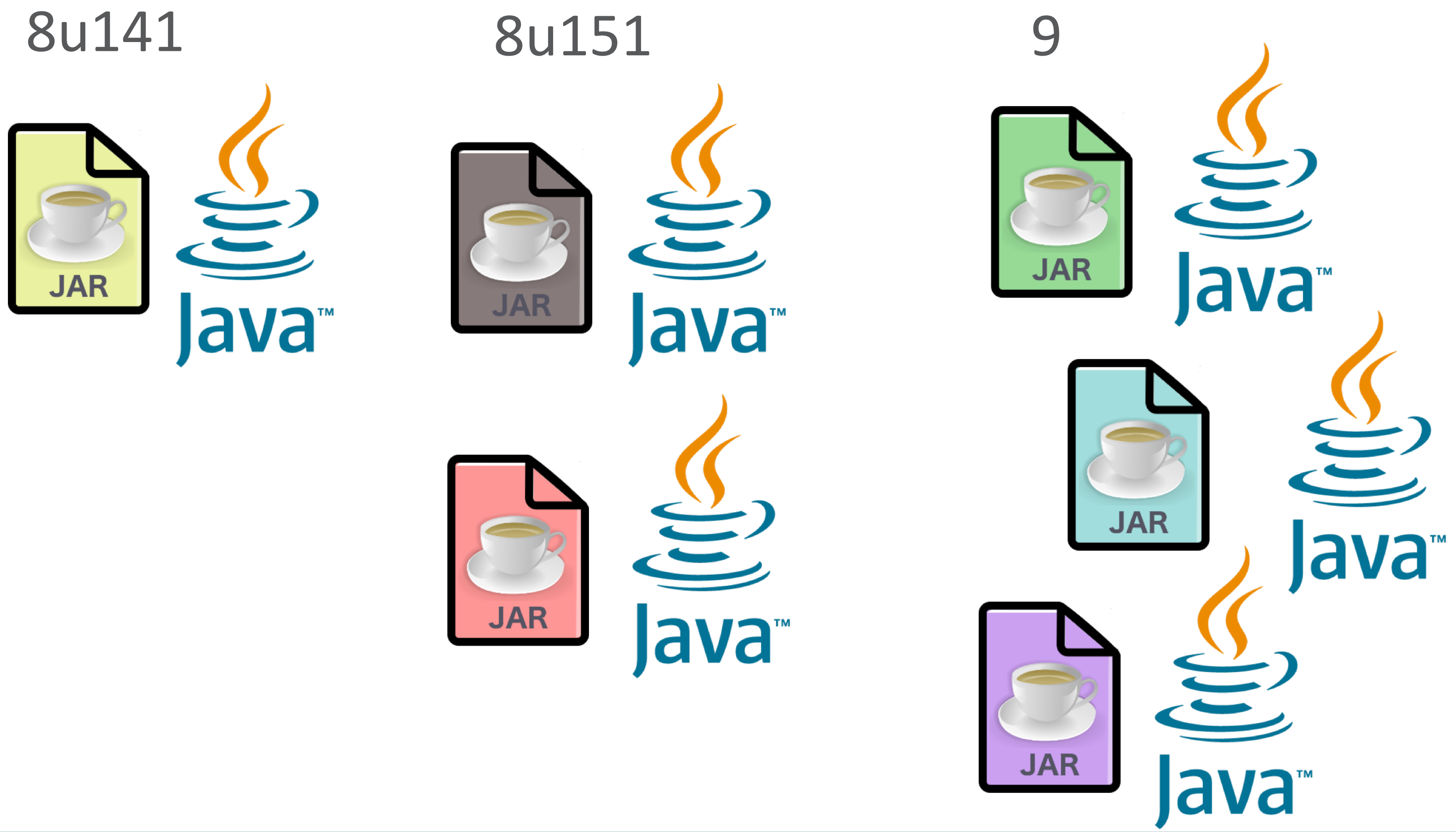
Java on the Browser: 3 Way-conversation



Bring-your-own-Java: More control, less surprises



Bring-your-own-Java: More control, less surprises

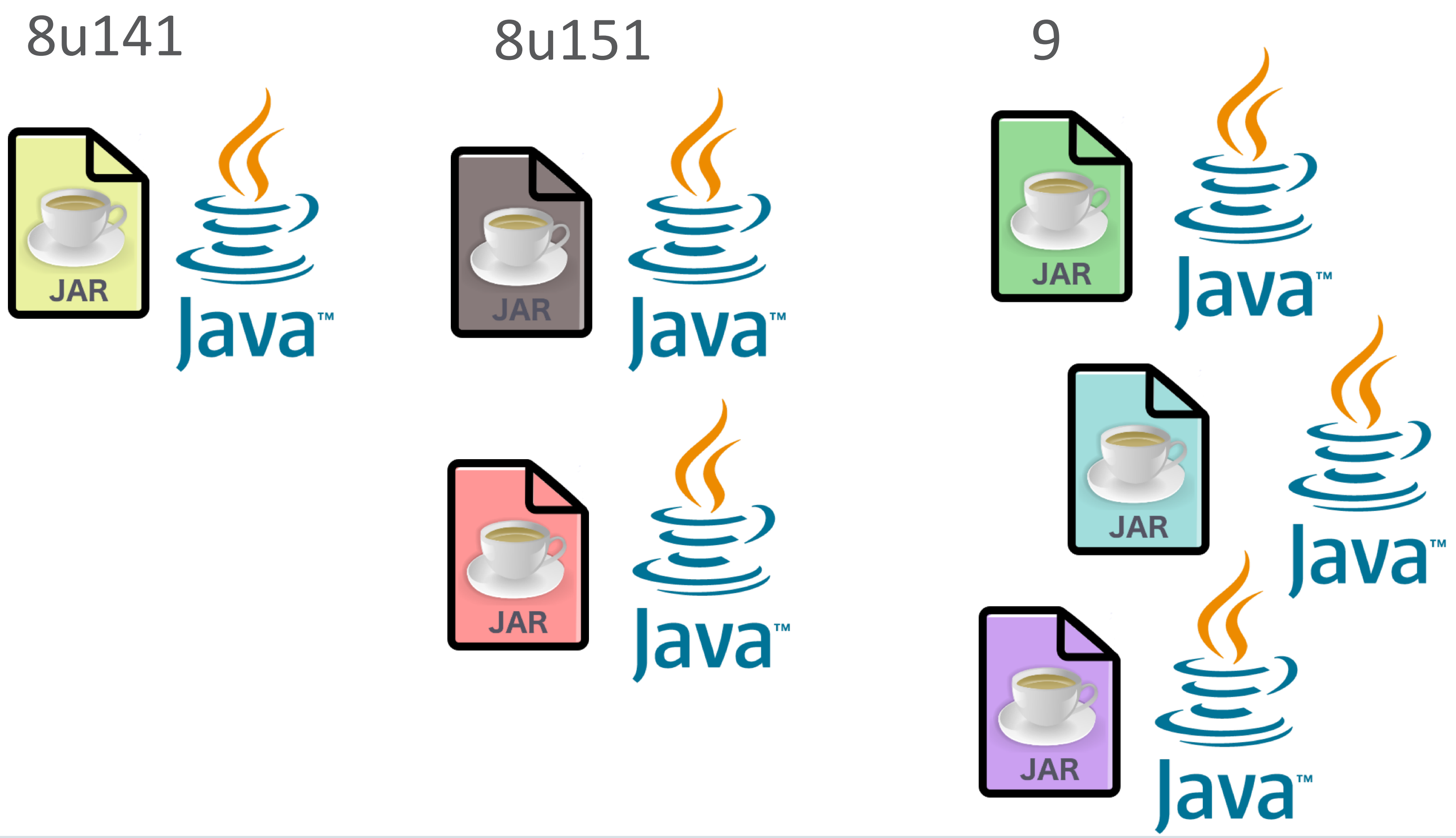


JEP 282: jlink: The Java Linker

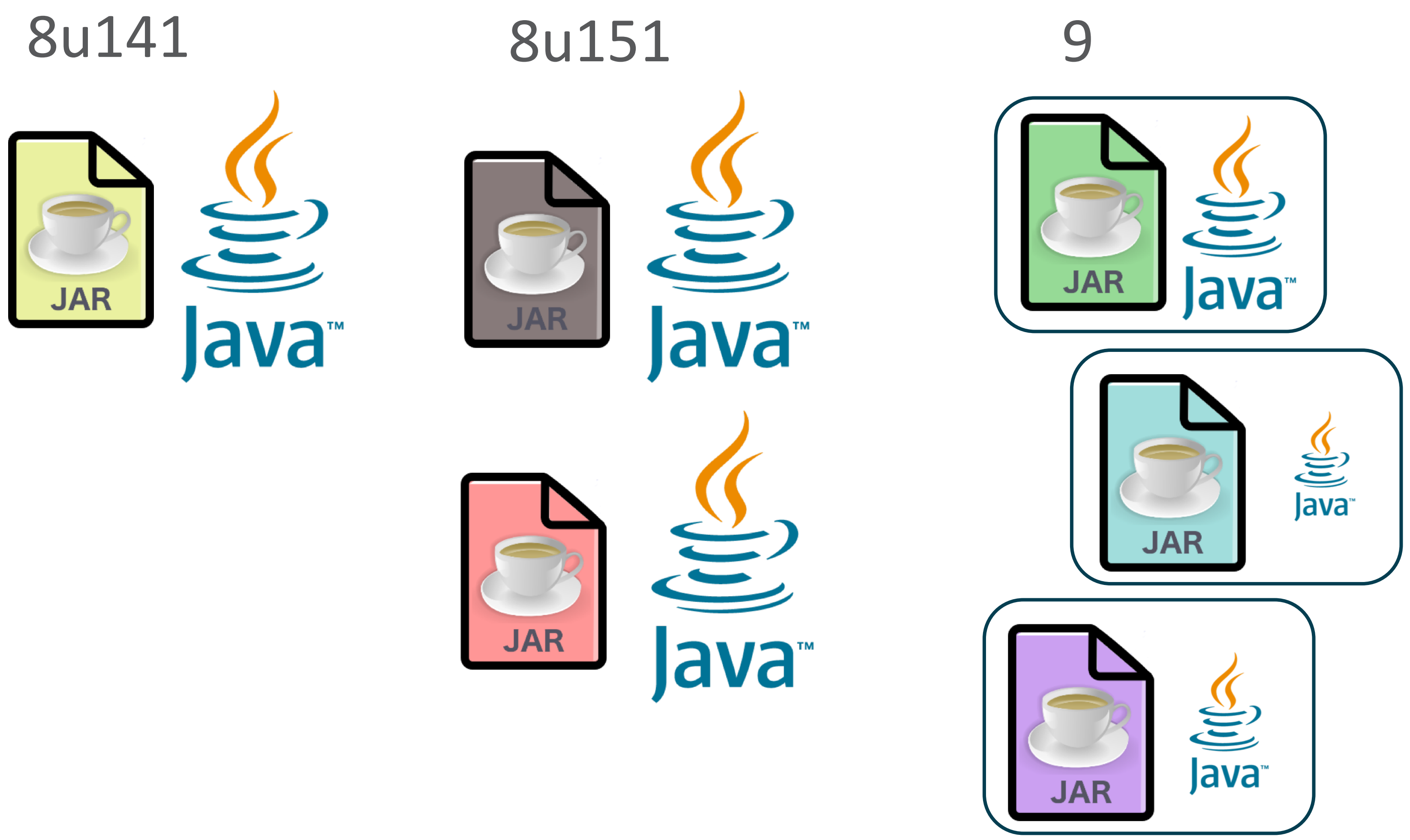
tools / jlink

- Create a tool that can assemble and optimize a set of modules and their dependencies into a custom run-time image as defined in JEP 220. Define a plugin mechanism for transformation and optimization during the assembly process, and for the generation of alternative image formats
- **Create a custom runtime optimized for a single program**
- JEP 261 defines *link time* as an optional phase between the phases of compile time and run time. Link time requires a linking tool that will assemble and optimize a set of modules and their transitive dependencies to create a run-time image or executable

Using Jlink



Using Jlink


8u141
JAR
Java
8u151
JAR
Java
JAR
Java
9
JAR
Java
JAR
Java
JAR
Java

Process API Updates
HTTP/2 Client
Improve Contended Locking
Unified JVM Logging
Compiler Control
Variable Handles
Segmented Code Cache
Smart Java Compilation, Phase Two
The Modular JDK
Modular Source Code
Elide Deprecation Warnings
    on Import Statements
Resolve Lint and Doclint Warnings
Milling Project Coin
Remove GC Combinations Deprecated in JDK 8
Tiered Attribution for javac
Process Import Statements Correctly
Annotations Pipeline 2.0
Datagram Transport Layer Security (DTLS)
Modular Run-Time Images
Simplified Doclet API
jshell: The Java Shell (Read-Eval-Print Loop)
New Version-String Scheme
HTML5 Javadoc
Javadoc Search
UTF-8 Property Files
Unicode 7.0
Add More Diagnostic Commands
Create PKCS12 Keystores by Default
Remove Launch-Time JRE Version Selection
Improve Secure Application Performance
Generate Run-Time Compiler Tests
Test Class-File Attributes Generated by javac
Parser API for Nashorn
Linux/AArch64 Port
Multi-Release JAR Files
Remove the JVM TI hprof Agent
Remove the jhat Tool
Java-Level JVM Compiler Interface
TLS ALPN
Validate JVM Command-Line Flag Arguments
Leverage CPU Instructions for GHASH and RSA
Compile for Older Platform Versions
Make G1 the Default Garbage Collector
OCSP Stapling for TLS
Store Interned Strings in CDS Archives
Multi-Resolution Images
Use CLDR Locale Data by Default
Prepare JavaFX for Modularization
Compact Strings
Merge Selected Xerces Updates into JAXP
BeanInfo Annotations
Update GStreamer in JavaFX/Media
HarfBuzz Font-Layout Engine
Stack-Walking API
Encapsulate Most Internal APIs
Module System
TIFF Image I/O
HiDPI Graphics on Windows and Linux
Platform Logging API and Service
Marlin Graphics Renderer
More Concurrency Updates
Convenience Factory Methods for Collections
Reserved Stack Areas for Critical Sections
Unicode 8.0
XML Catalogs
Unified GC Logging
Platform-Specific Desktop Features
DRBG-Based SecureRandom Implementations
Enhanced Method Handles
Modular Java Application Packaging
Dynamic Linking of Language-Defined
    Object Models
Enhanced Deprecation
Additional Tests for Humongous Objects in G1
Improve Test-Failure Troubleshooting
Indify String Concatenation
HotSpot C++ Unit-Test Framework
jlink: The Java Linker
Enable GTK 3 on Linux
New HotSpot Build System
Spin-Wait Hints
SHA-3 Hash Algorithms
Disable SHA-1 Certificates
Deprecate the Applet API
Filter Incoming Serialization Data
Deprecate the Concurrent Mark Sweep GC
Implement Selected ECMAScript 6 Features
Linux/s390x Port
Ahead-of-Time Compilation
Unified arm32/arm64 Port
Remove Demos and Samples
Reorganize Documentation

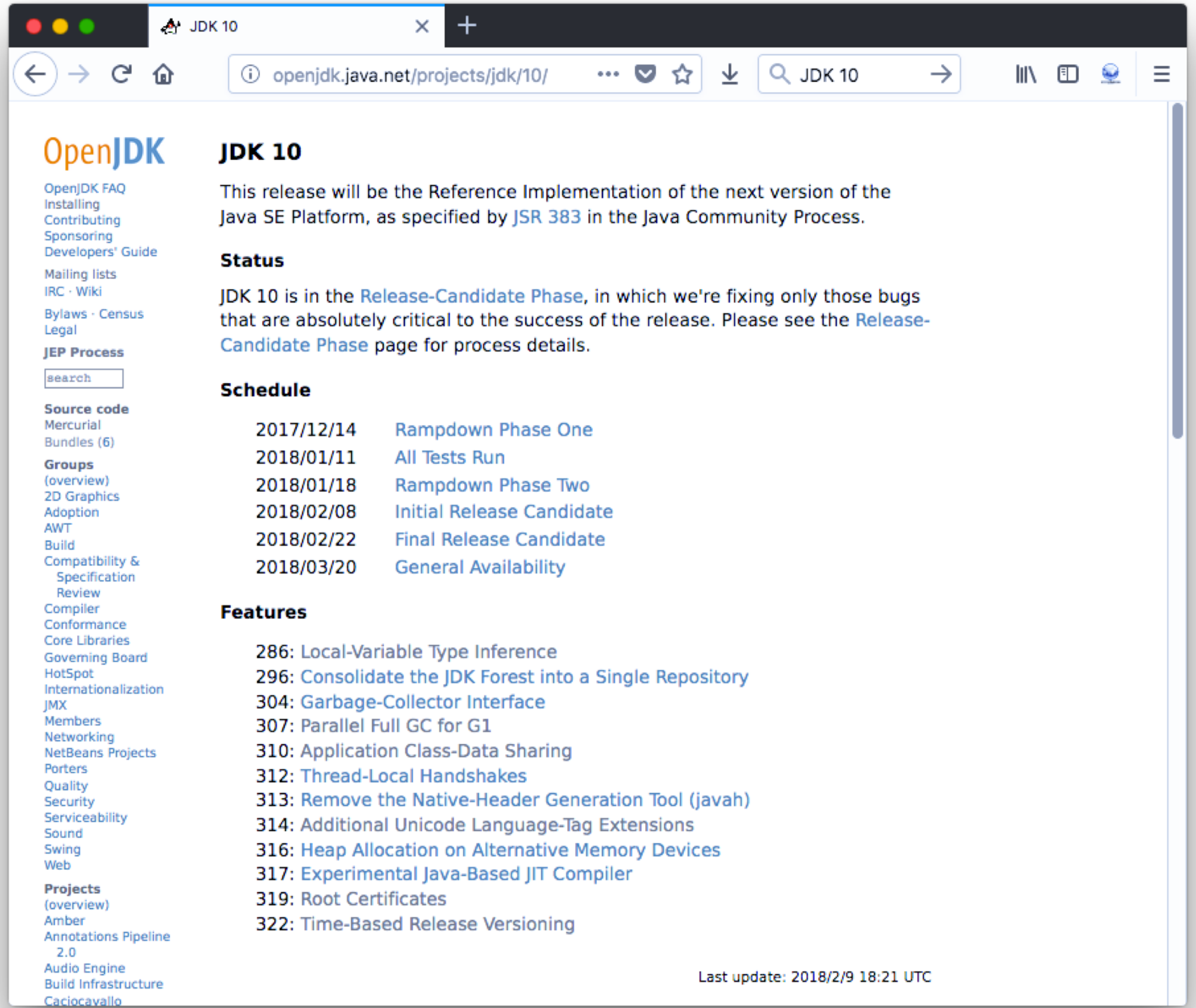
Also opened since JavaOne 2017

- Project ZGC
  - Scalable low latency garbage collector capable of handling heaps ranging from gigabytes to terabytes in size, with GC pause times not exceeding 10ms
- OpenJDK Early Access binaries under GPL
  - Feature releases (e.g. JDK 9, JDK 10, JDK 11)
  - Project-specific binaries e.g. Project Valhalla

Java 10

JDK 10 – Mar 2018

- First feature release
- 12 JEPs
  (Java Enhancement Proposals)



OpenJDK

OpenJDK FAQ
Installing
Contributing
Sponsoring
Developers' Guide

Mailing lists
IRC · Wiki

Bylaws · Census
Legal

JEP Process

search

Source code
Mercurial
Bundles (6)

Groups
(overview)
2D Graphics
Adoption
AWT
Build
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
Internationalization
JMX
Members
Networking
NetBeans Projects
Porters
Quality
Security
Serviceability
Sound
Swing
Web

Projects
(overview)
Amber
Annotations Pipeline 2.0
Audio Engine
Build Infrastructure
Caciocavallo

JDK 10

This release will be the Reference Implementation of the next version of the Java SE Platform, as specified by JSR 383 in the Java Community Process.

Status

JDK 10 is in the Release-Candidate Phase, in which we're fixing only those bugs that are absolutely critical to the success of the release. Please see the Release-Candidate Phase page for process details.

Schedule

2017/12/14	Rampdown Phase One
2018/01/11	All Tests Run
2018/01/18	Rampdown Phase Two
2018/02/08	Initial Release Candidate
2018/02/22	Final Release Candidate
2018/03/20	General Availability

Features

286: Local-Variable Type Inference
296: Consolidate the JDK Forest into a Single Repository
304: Garbage-Collector Interface
307: Parallel Full GC for G1
310: Application Class-Data Sharing
312: Thread-Local Handshakes
313: Remove the Native-Header Generation Tool (javah)
314: Additional Unicode Language-Tag Extensions
316: Heap Allocation on Alternative Memory Devices
317: Experimental Java-Based JIT Compiler
319: Root Certificates
322: Time-Based Release Versioning

Last update: 2018/2/9 18:21 UTC

JEP 286: Local-Variable Type Inference

specification / language

- Enhance the Java Language to extend type inference to declarations of local variables with initializers
- Restricted to local variables with initializers, indexes in the enhanced for-loop, and locals declared in a traditional for-loop
- Not available for method formals, constructor formals, method return types, fields, catch formals, or any other kind of variable declaration

```
ArrayList<String> list = new ArrayList<String>();
Stream<String> stream = list.stream();
```

JEP 286: Local-Variable Type Inference

specification / language

- Enhance the Java Language to extend type inference to declarations of local variables with initializers
- Restricted to local variables with initializers, indexes in the enhanced for-loop, and locals declared in a traditional for-loop
- Not available for method formals, constructor formals, method return types, fields, catch formals, or any other kind of variable declaration

```
var list = new ArrayList<String>();
var stream = list.stream();
```

JEP 310: Application Class-Data Sharing

hotspot / runtime

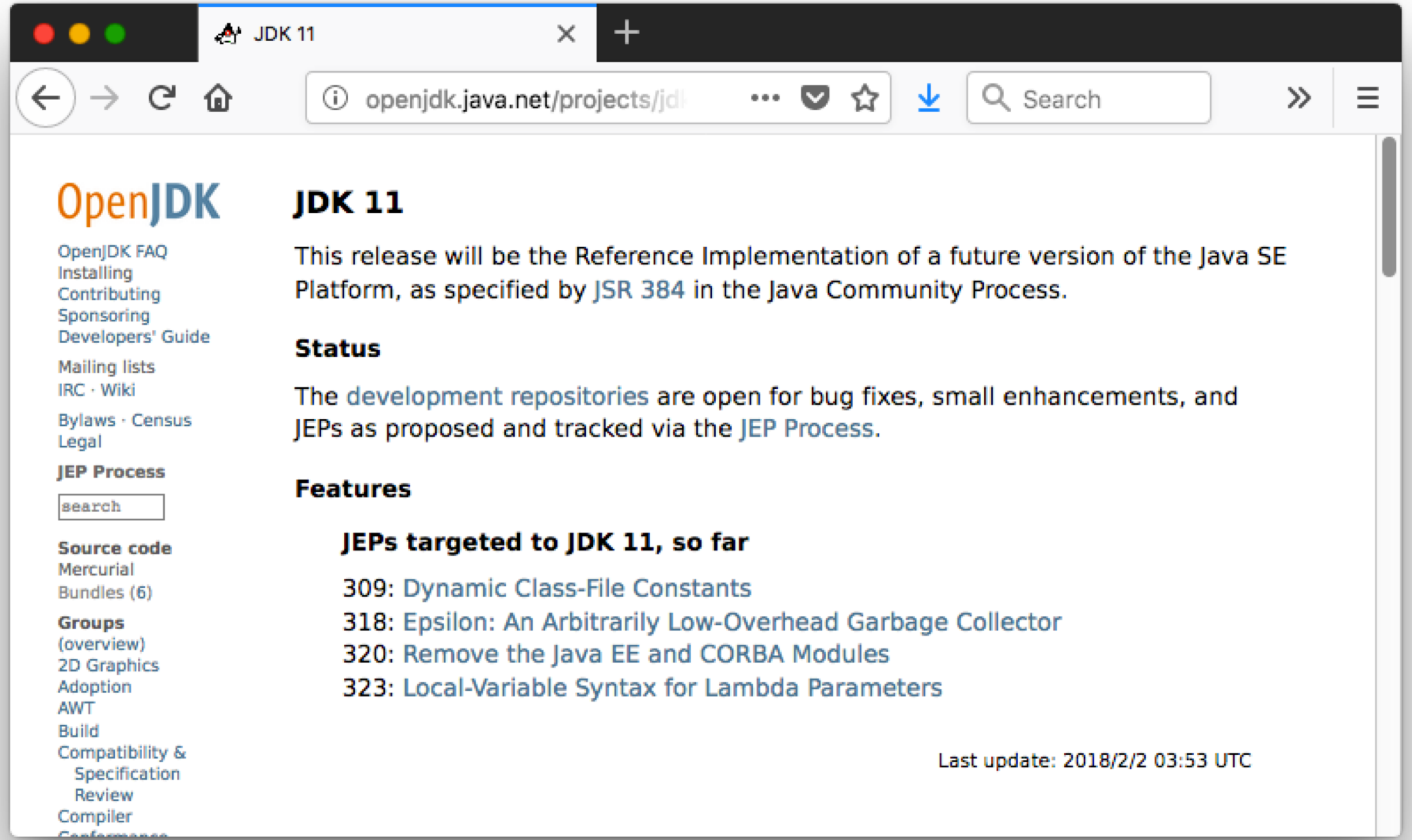
- Extend the existing Class-Data Sharing ("CDS") feature to allow application classes to be placed in the shared archive
- Reduce footprint by sharing common class metadata across different Java processes.
- Improve startup time.

First Oracle JDK commercial feature Open Sourced !

Java 11

JDK 11 – Sep 2018

- 4 JEPs targeted so far...
  - New model calls for JEPS to be targeted only when ready



JDK 11

This release will be the Reference Implementation of a future version of the Java SE Platform, as specified by JSR 384 in the Java Community Process.

Status

The development repositories are open for bug fixes, small enhancements, and JEPs as proposed and tracked via the JEP Process.

Features

JEPs targeted to JDK 11, so far

309: Dynamic Class-File Constants
318: Epsilon: An Arbitrarily Low-Overhead Garbage Collector
320: Remove the Java EE and CORBA Modules
323: Local-Variable Syntax for Lambda Parameters

Last update: 2018/2/2 03:53 UTC

Beyond Java 11

The Next Big Challenge: Object Data layout

- Java is very good at optimizing code, less so at optimizing data
- Java’s type system gives us primitives, objects, and arrays
- But flexibility is not exactly where we need it
- The big problem: object identity
- Project Valhalla – Value Types

Improved Java/Native Interoperability

- Big Data Hadoop and Spark are highly dependent on native libraries
- Meanwhile, Java has significant technical debts in support of foreign calls
- Project Panama - provide an easier, safer and faster JNI
- Project Loom – Lightweight thread and continuation

Summary

- The Java platform development on OpenJDK is becoming more open
  - Contributing all commercial features (zGC, JFR, AppCDS, etc)
  - GPL+CPE build
- The cloud is demanding a faster pace and continuous delivery
  - Uptake new Java releases every 6-months!
- Beyond 10, we have a solid technical roadmap
- Let’s continue to innovate and advance the Java SE Platform on OpenJDK together!

Join and become an OpenJDK contributor

https://openjdk.java.net