Shaping the future of Java, Faster

Georges Saab

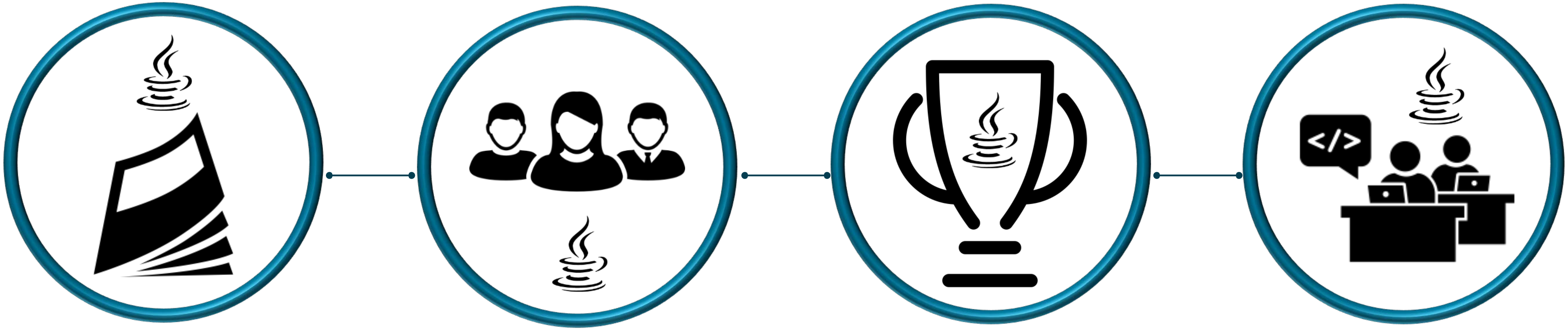
Vice President, Java Platform Group

Oracle, Corp

Twitter: ***@gsaab***

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

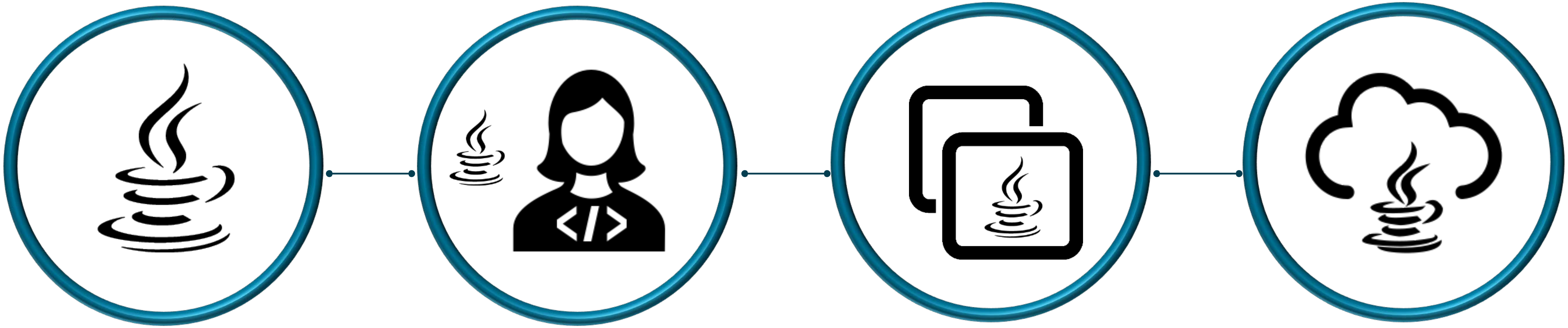


Communication
Java Magazine
250K+ subscribers

Community
Java User Groups
350+ worldwide

Collaboration
Java Champions
150+ worldwide

Contribution
OpenJDK
470 community participants

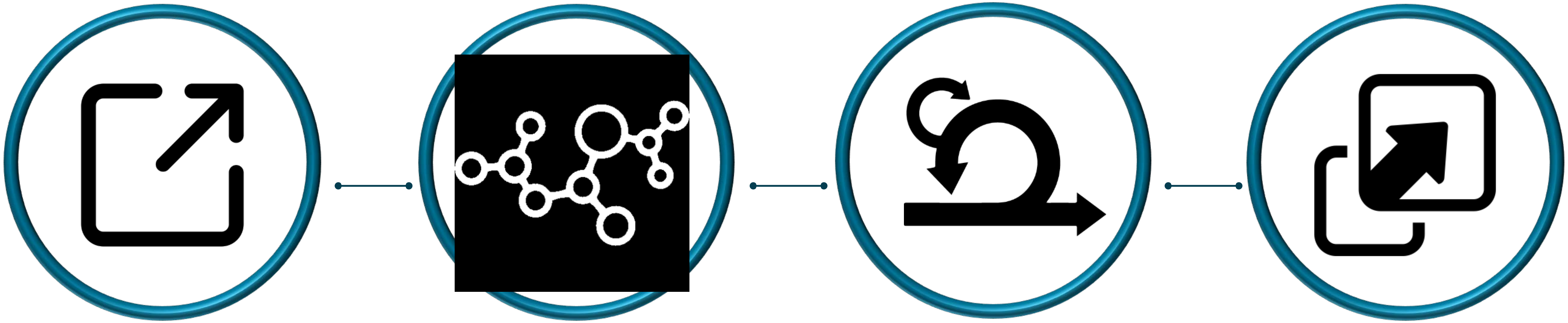


#1
Programming
Language

12 Million
Developers
Run Java

38 Billion
Active
Virtual Machines

21 Billion
Cloud Connected
Virtual Machines



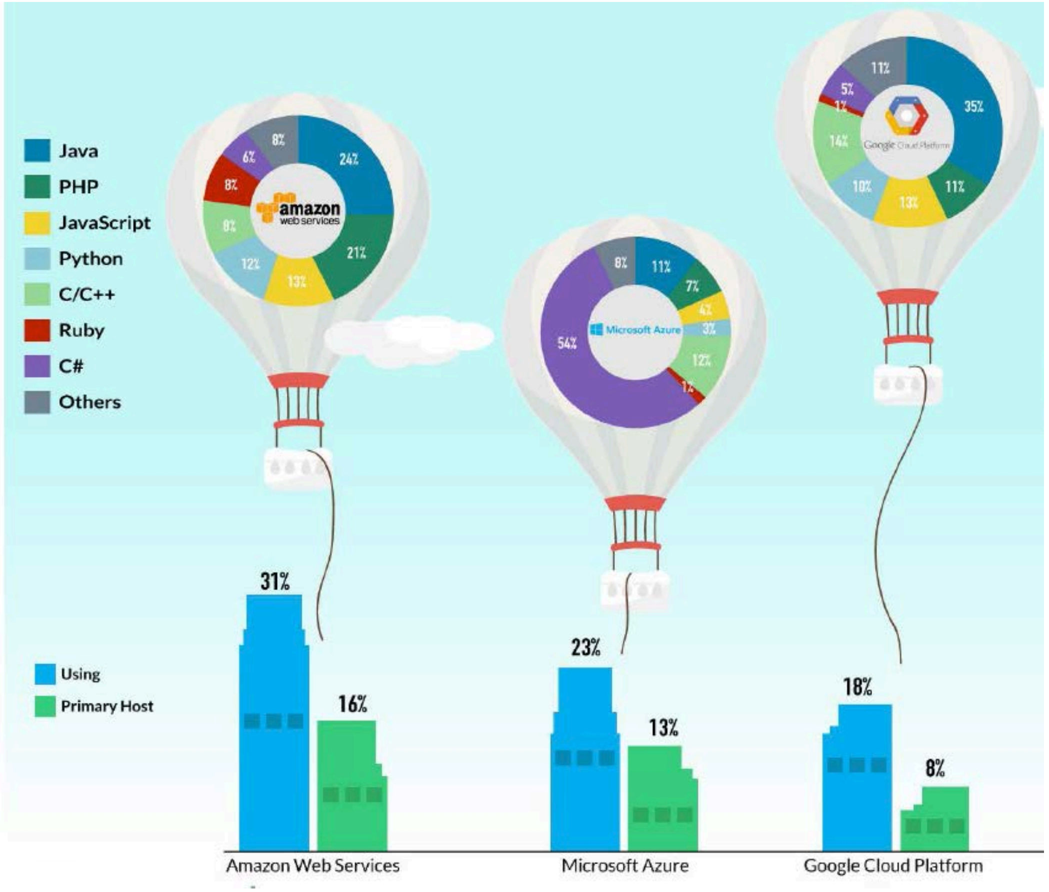
Open

Evolving

Nimble

Scalable

Java SE is #1 Runtime in the Cloud

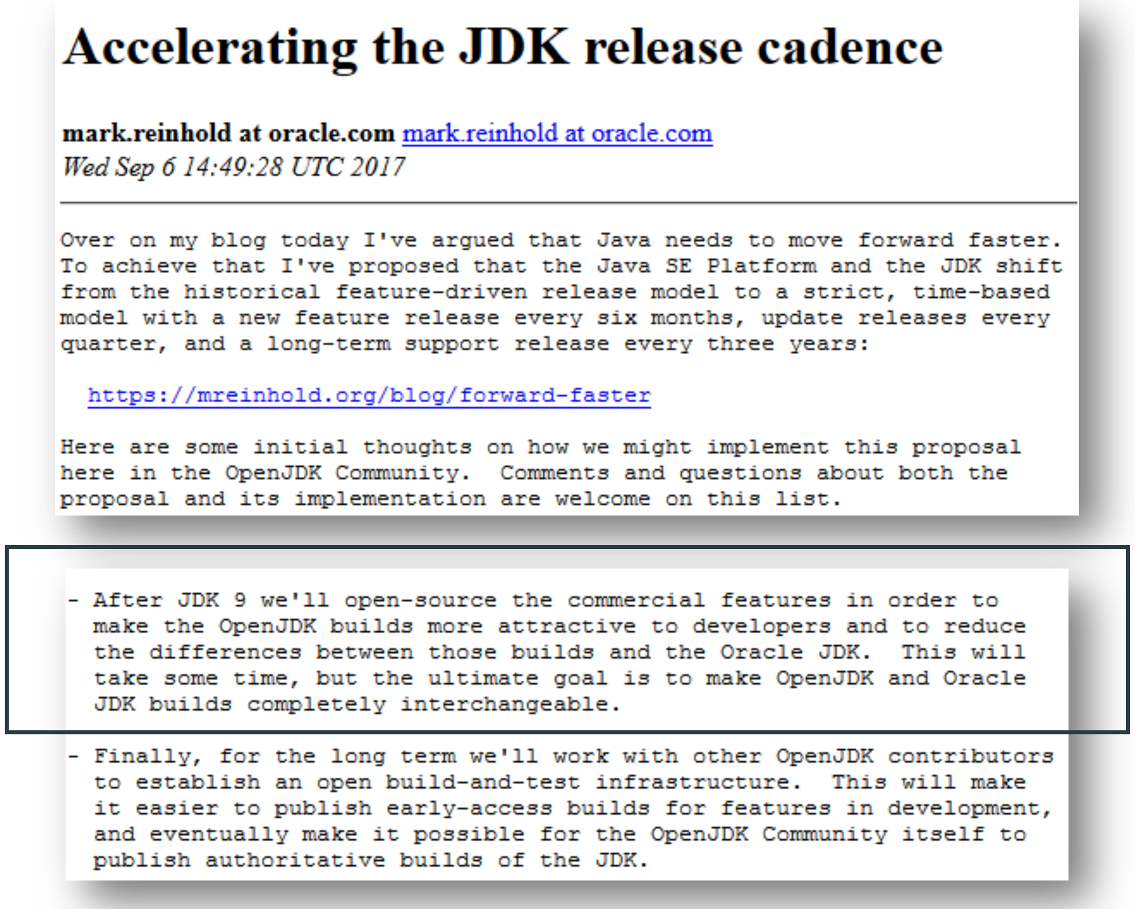


- #1 Deployment runtime on AWS and Google App Engine and #3 on MS Azure
- Java Runtime is the foundation of the Cloud IaaS, PaaS and SaaS

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)

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.
```

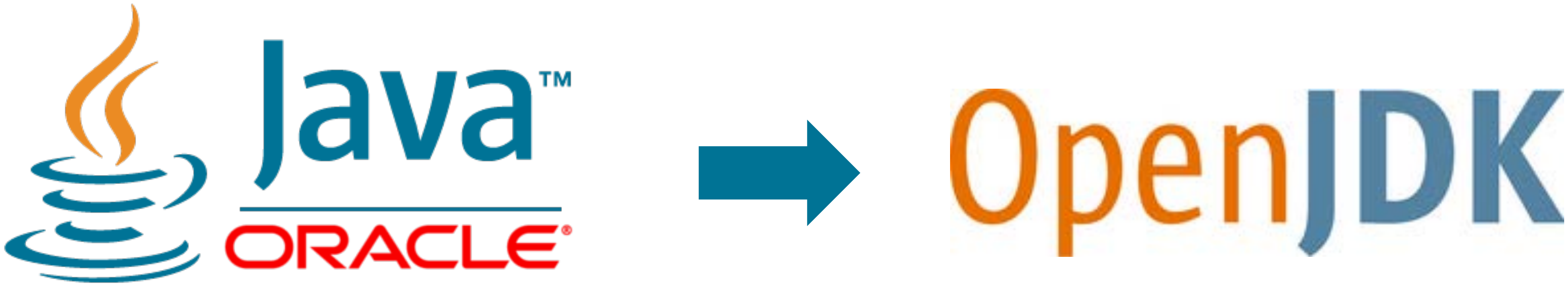
- **New Java feature release will be made every 6 months**
- Oracle will now produce OpenJDK builds
- The new OpenJDK builds will be licensed under GPL V2

  GNU General Public License Version 2 with Class Path Exception (GPL 2 with CPE)

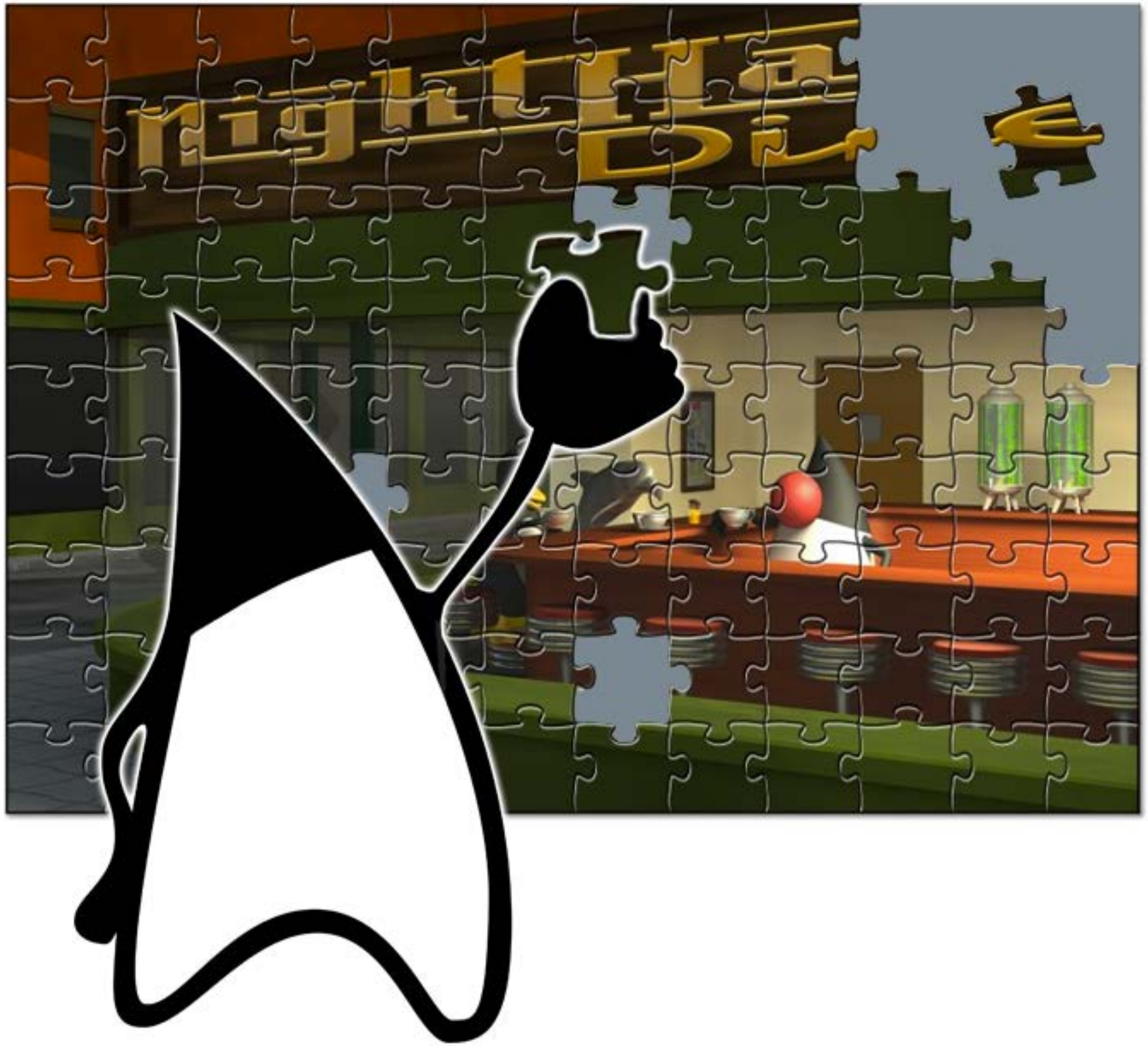
- Oracle will open source commercial features
- Oracle will work with other OpenJDK contributors to make the community infrastructure complete, modern and accessible

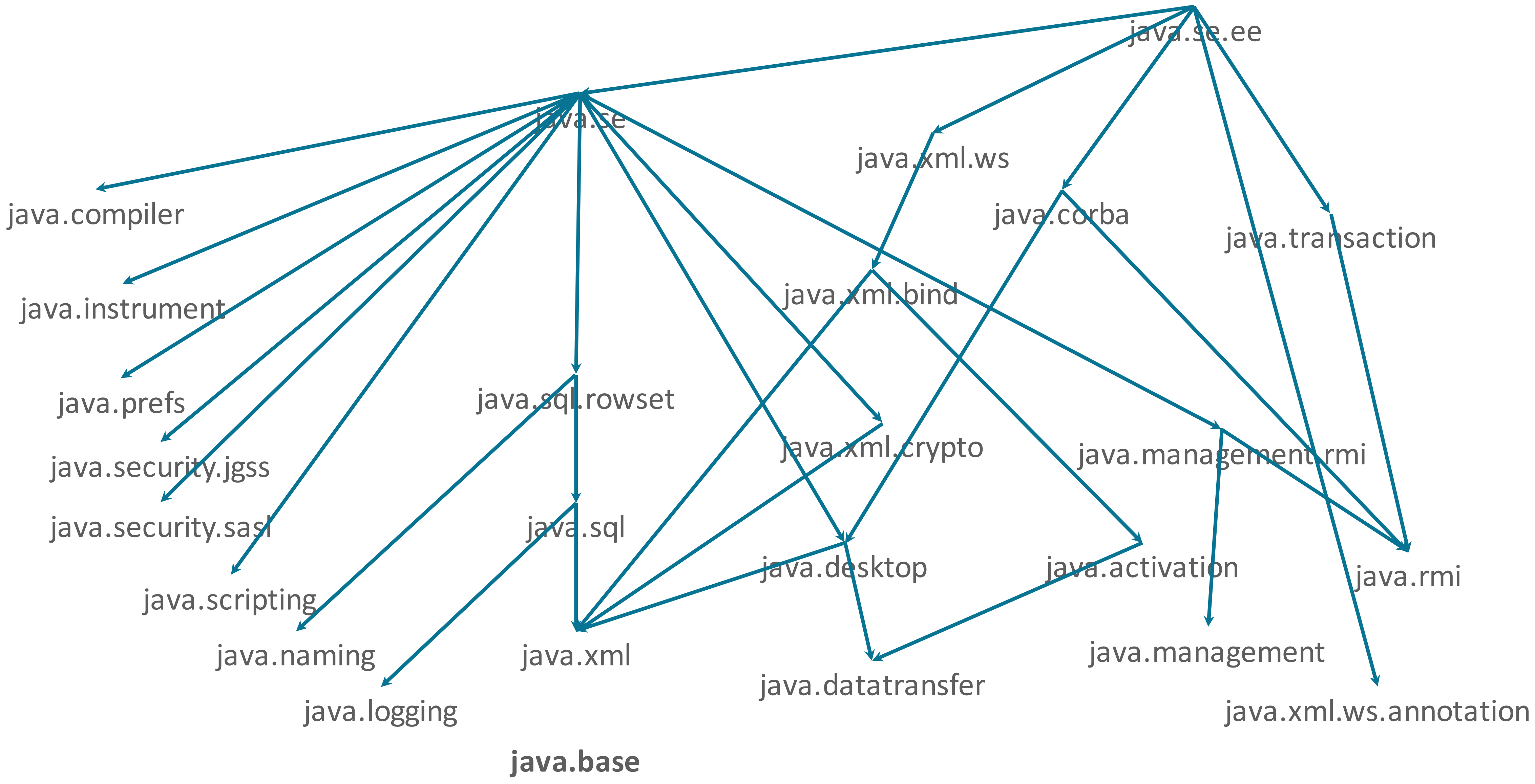
URL: http://mail.openjdk.java.net/pipermail/discuss/2017-September/004281.html

From Oracle JDK to OpenJDK from Oracle



Java 9




java.se.ee
java.se
java.xml.ws
java.corba
java.transaction
java.compiler
java.instrument
java.xml.bind
java.prefs
java.sql.rowset
java.xml.crypto
java.management.rmi
java.security.jgss
java.security.sasl
java.sql
java.desktop
java.activation
java.rmi
java.scripting
java.naming
java.xml
java.management
java.datatransfer
java.logging
java.xml.ws.annotation
java.base

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

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

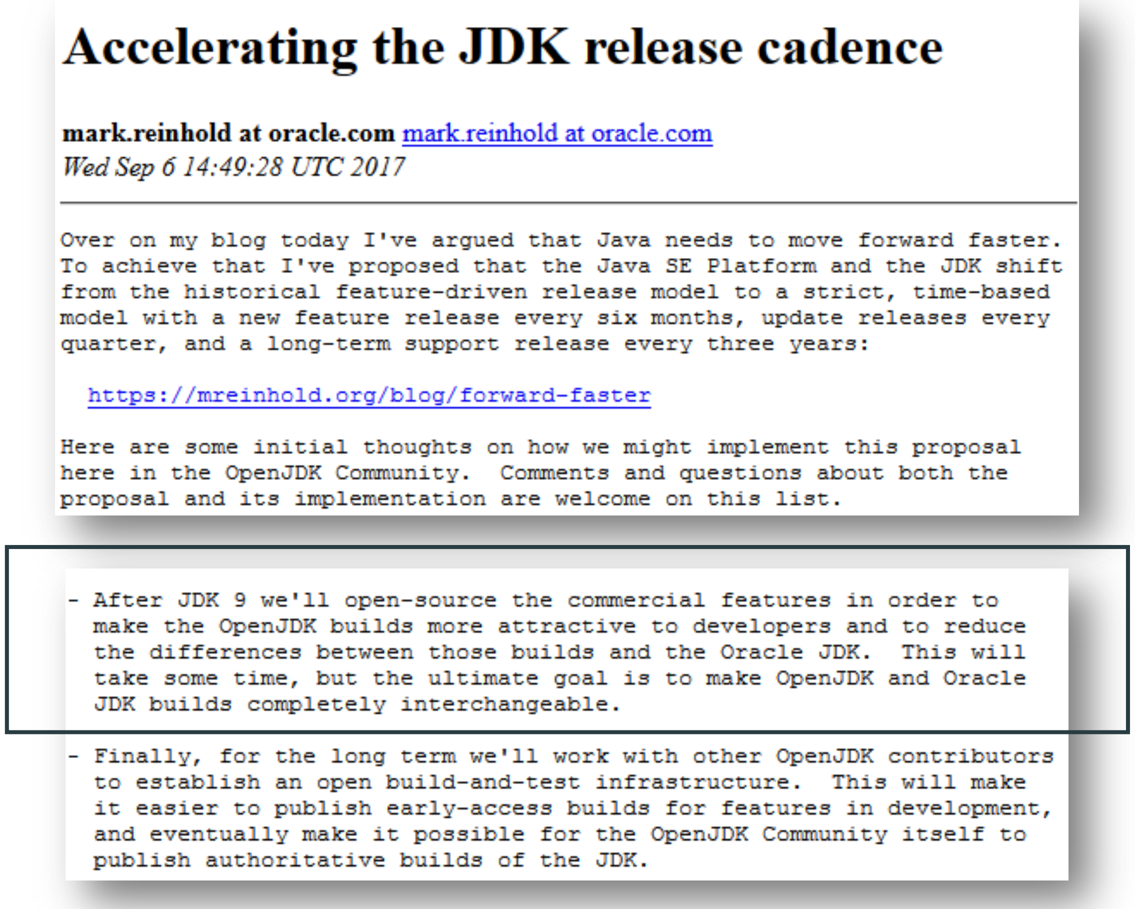
- Managed language/runtime
- Hardware and operating system agnostic
- Safety and security enforced by JVM
- Reliable: Compatibility is a key design goal
- Runtime adaptive: JVM ensures stable execution when environment changes
- Rich ecosystem

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
-

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

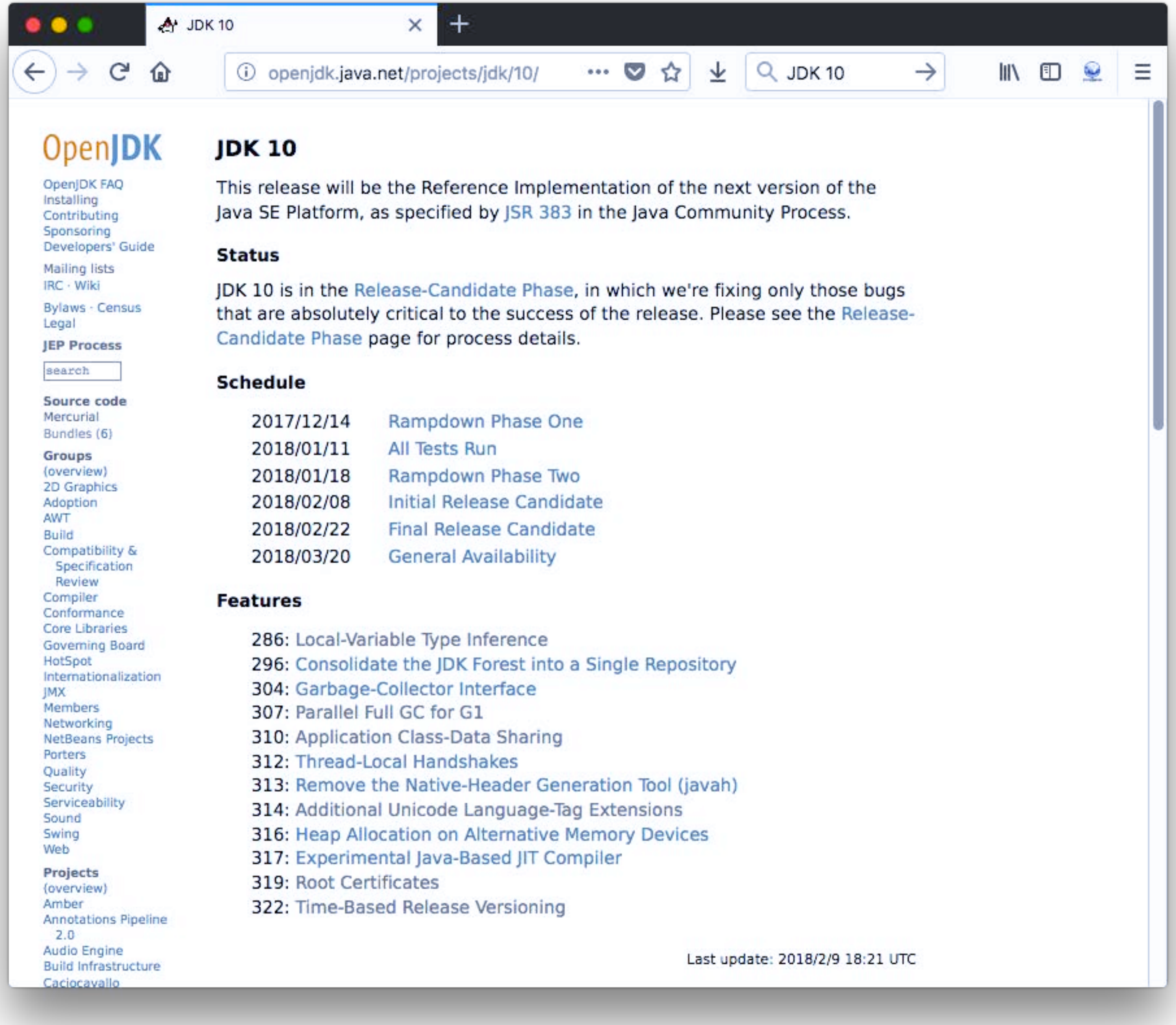
What Is Being Open-Sourced in Java

- **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 10

JDK 10 – Mar 2018

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



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 !

Demo

Local-Variable Type Inference

15+ Years of Type Inference in Java

```
List<Block> blocks = List.of(…);
int maxWeight = blocks.stream()
                      .filter(block -> block.getColor() == BLUE)
                      .mapToInt(blue -> blue.getWeight())
                      .max();
```

Local Variable Type Inference in Java

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

Local Variable Type Inference in Java

```
Url url                      = new URL(“http://www.oracle.com/”);
URLConnection conn = url.openConnection();
Reader reader             = new InputStreamReader(conn.getInputStream());

var url          = new URL(“http://www.oracle.com/”);
var conn       = url.openConnection();
var reader     = new InputStreamReader(conn.getInputStream());
```

Application Class-Data Sharing

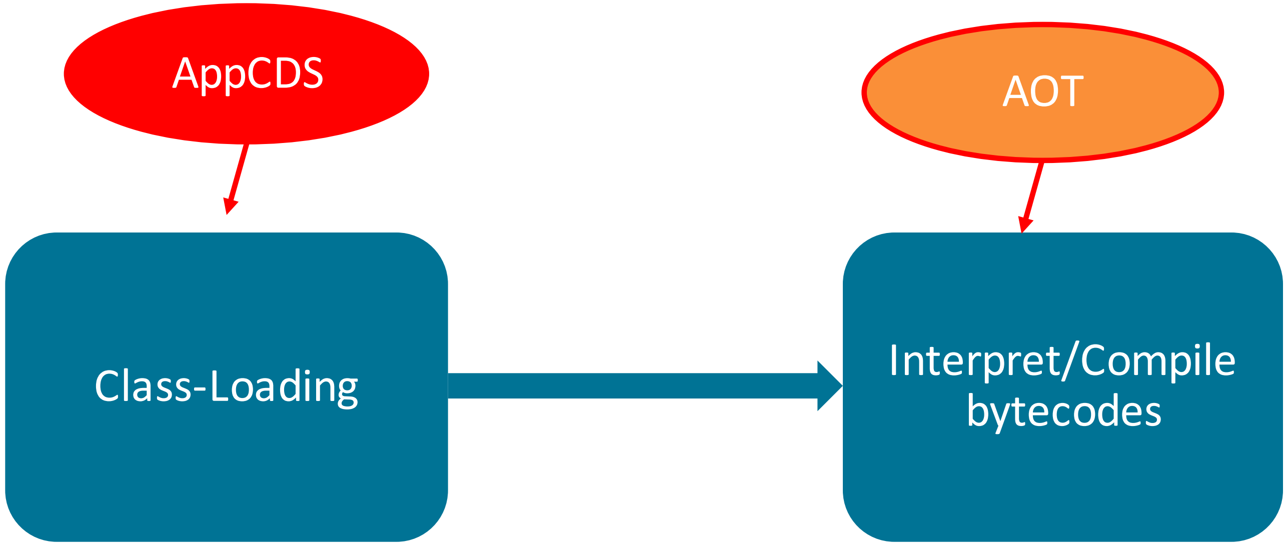
AppCDS Overview

Goal

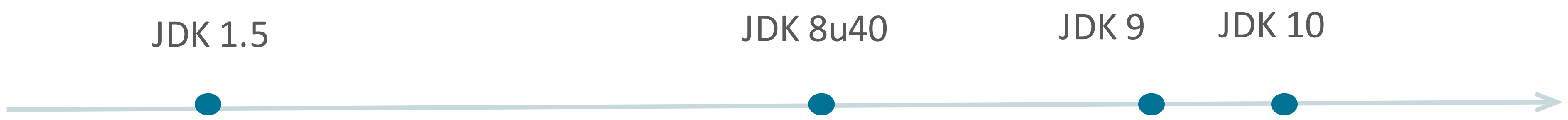
- Improve startup time
- Reduce runtime memory footprint

First Oracle JDK commercial feature Open Sourced !

What consumes additional times during Java Startup?



CDS/AppCDS evolution



JDK 1.5

JDK 8u40

JDK 9

JDK 10

Cache classes from bootstrap classloader only

New feature added

Open sourced!

Dump time process

- Classes are parsed into JVM as class metadata
- Metaspace is splitted into read-only(RO) and read-write (RW) parts
- All loaded class metadata is saved to a file (shared archive)

```
Java -Xshare:dump -XX:+UseAppCDS -XX:SharedArchiveFile=<jsa>  \
        -XX:SharedClassListFile=<classlist> -XX:SharedArchiveConfigFile=<config_file>
```

Runtime process

- Shared archive is memory-mapped into JVM address space
- RO are shared among multiple JVMs
- RW are shared copy-on-write
- Mapped class metadata can be used with minimal processing

```
Java -Xshare:dump -XX:+UseAppCDS -XX:SharedArchiveFile=<jsa>
```

Improvement measured

- Fit into cloud computing and micro-services!

Software	Startup time Reduced	Footprint(memory) Reduced
WebLogic	19 ~ 37%	(average) 5%
Apache Spark with KMeans workload and 20 slaves	11%	(average) 10%

Also opened since JavaOne 2017

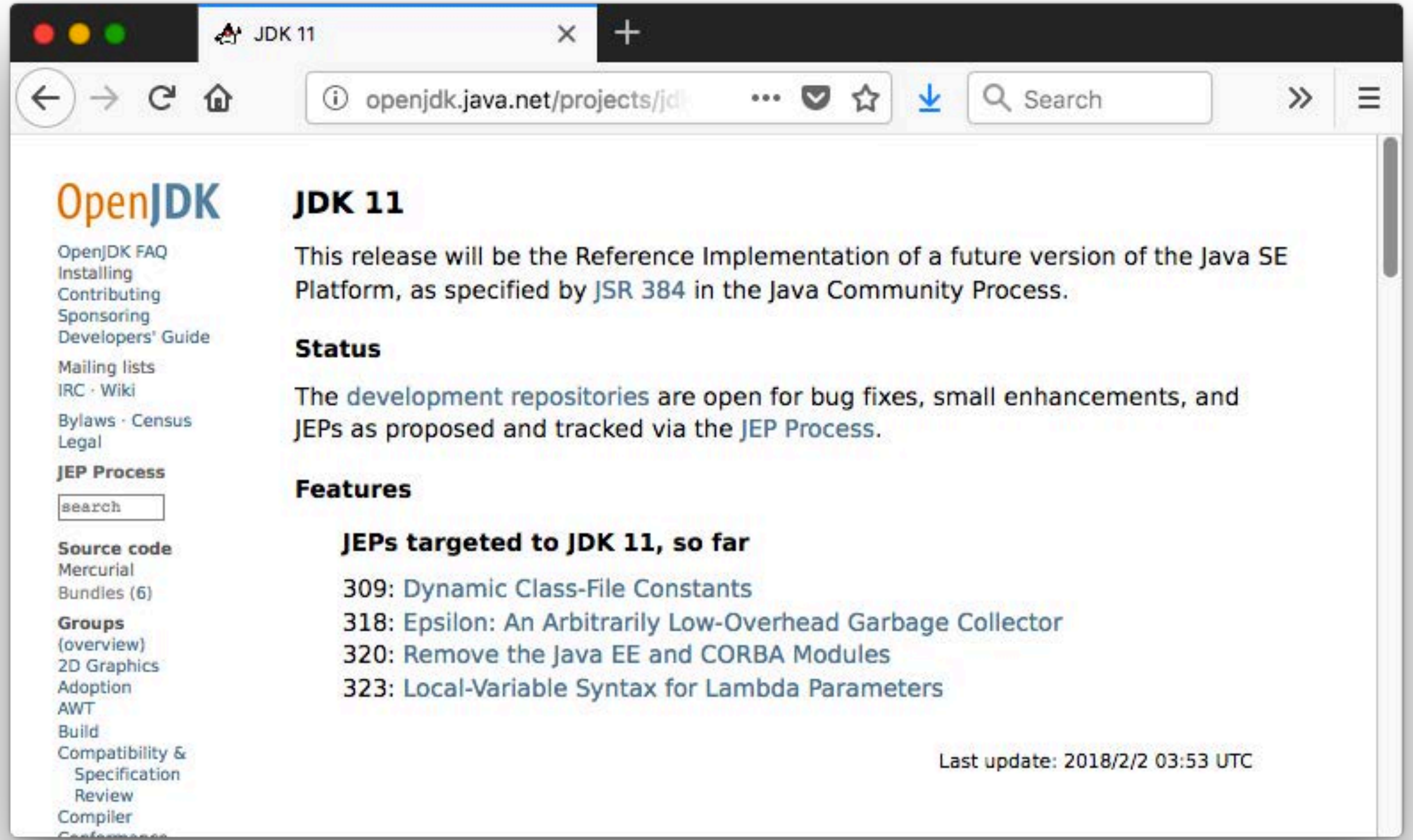
- Project ZGC
  - Scalable low latency garbage collector capable of handling heaps ranging from gigabytes to terabytes in size, with GC pause times not exceeding 10ms

- OpenJDK Early Access binaries under GPL
  - Feature releases (e.g. JDK 9, JDK 10, JDK 11)
  - Project-specific binaries e.g. Project Valhalla

Java 11

JDK 11 – Sep 2018

- 4 JEPs targeted
  - 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

主办方 Geekbang> 极客邦科技 InfoQ

GMTC2018

全球大前端技术大会

大前端的下一站

<<扫码了解更多详情>>