El futuro de Java SE

Aurelio Garcia-Ribeyro
Director Senior
Grupo de Plataforma Java, Oracle
@aureliog

JavaYourNext
(Cloud)

Ago 2019

Disculpas a los puristas

Algunos términos en esta presentación no han sido traducidos del inglés.

Términos que el uso y costumbre –al menos en algunos países hispanohablantes- permiten usar en inglés, nombres propios, y conceptos que no supimos traducir sin cambiar el significado han sido dejados en el idioma original.

Por política de Oracle Descargos de Responsabilidad Legal no pueden ser traducidos.

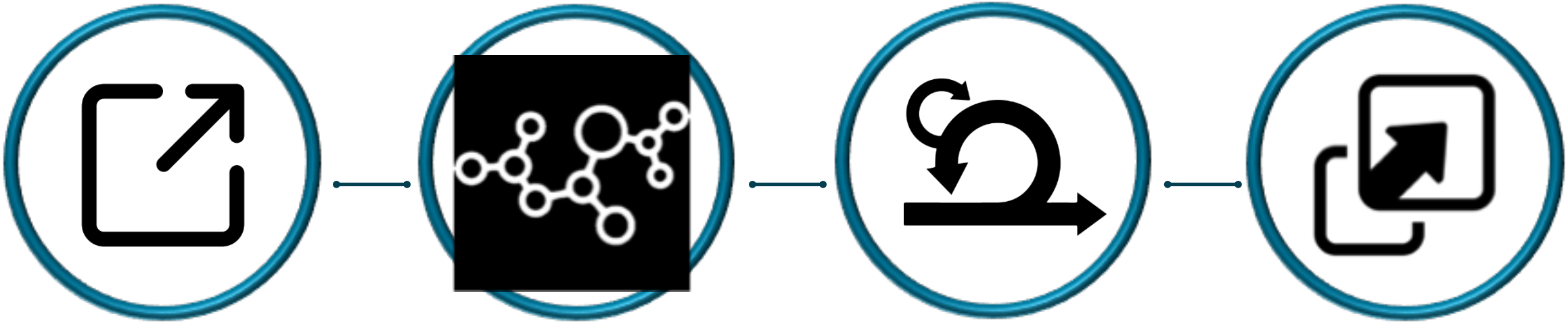
Safe Harbor Statement

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.

Introducción

Aurelio Garcia-Ribeyro

- Director Senior – Gerencia de Producto, Grupo de Plataforma Java, Oracle
- A cargo de los requerimientos de JDK Oracle desde que ingresé a Oracle a través de la adquisición de Sun Microsystems en el 2010
- Antes de unirme a Oracle trabajé en Sun Microsystems para el grupo de gerencia de producto de Java SE
- MBA de MIT Sloan y Bachiller en Ingeniería de Sistemas de la Universidad de Lima

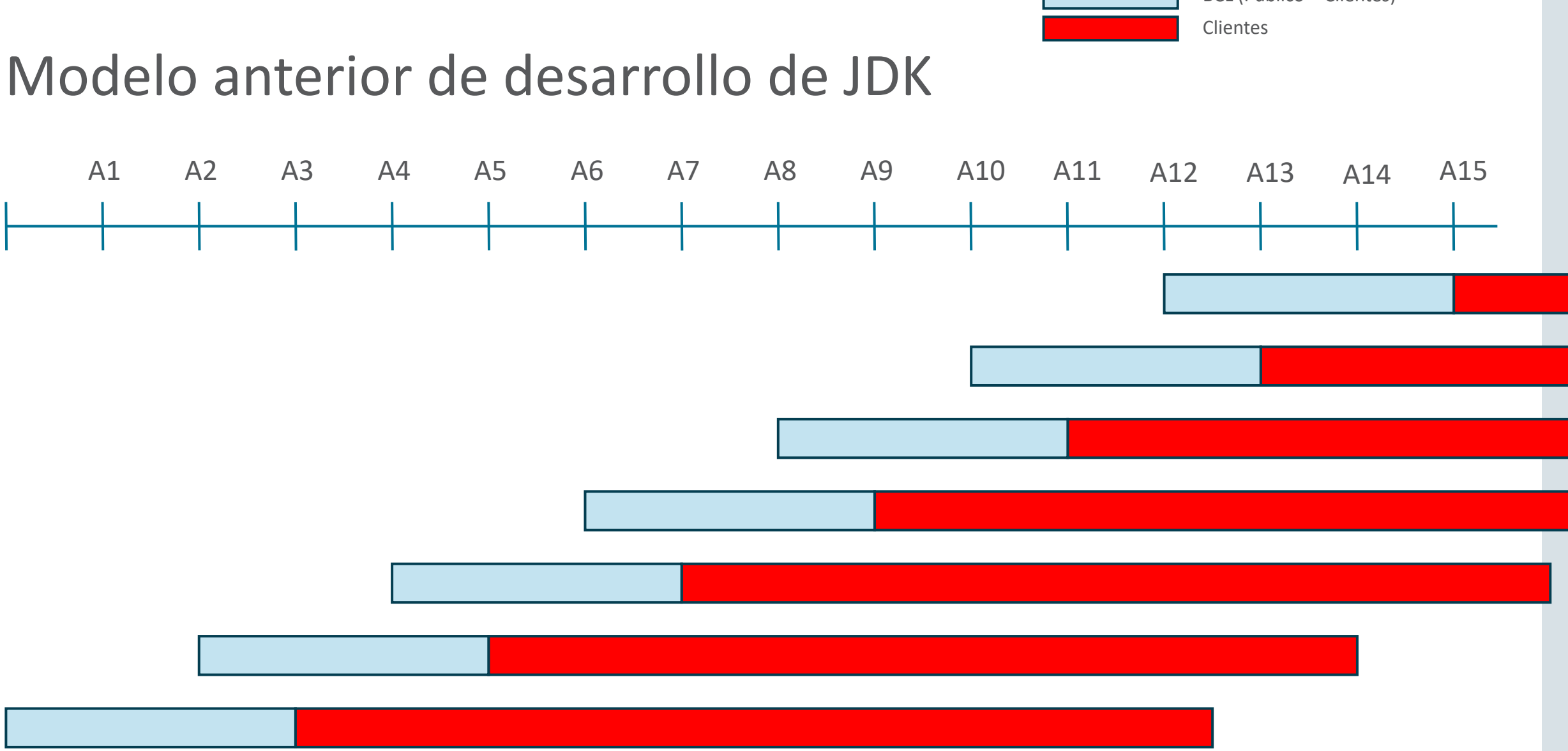


Abierto Evolucionando Ágil Escalable

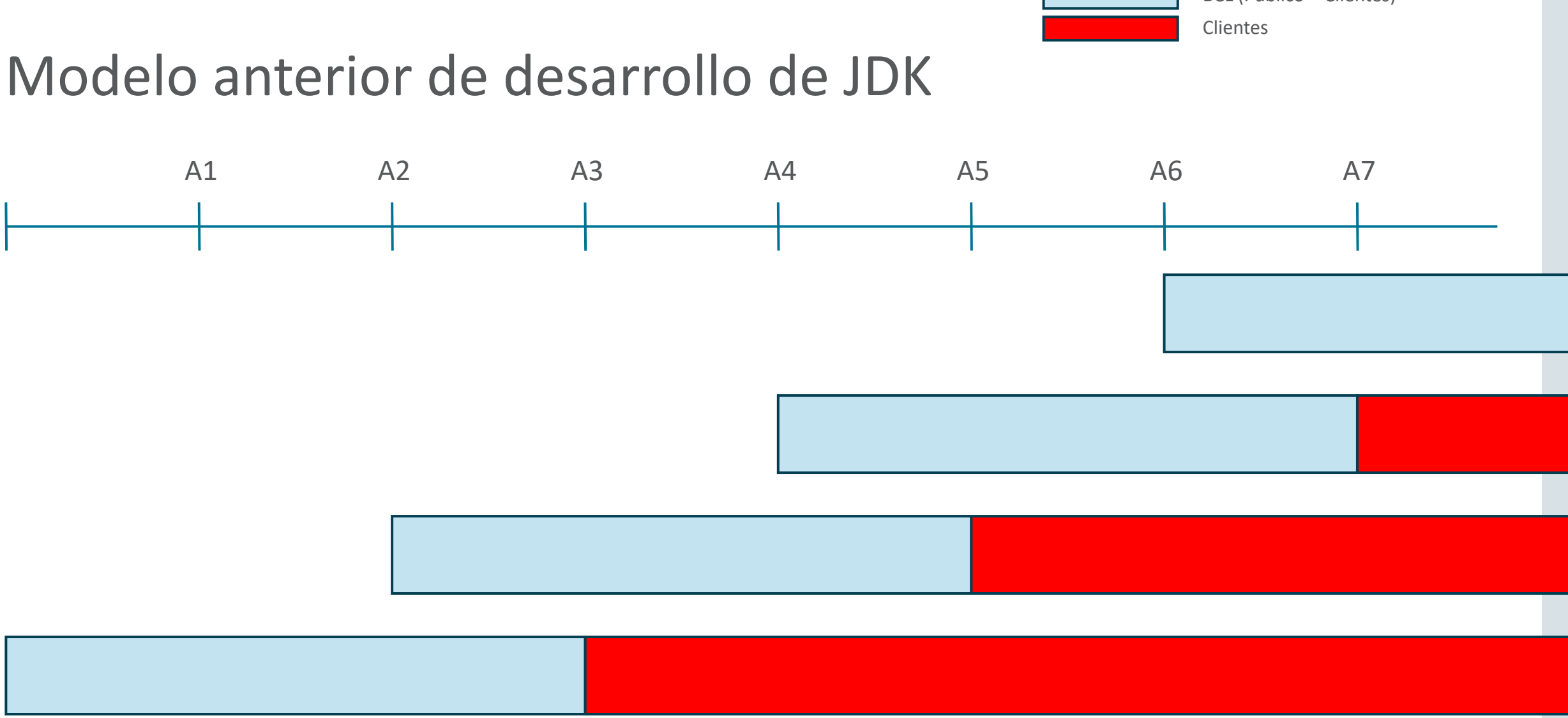
El nuevo modelo de Desarrollo

De Por-Objetivos a Tiempo-Fijo

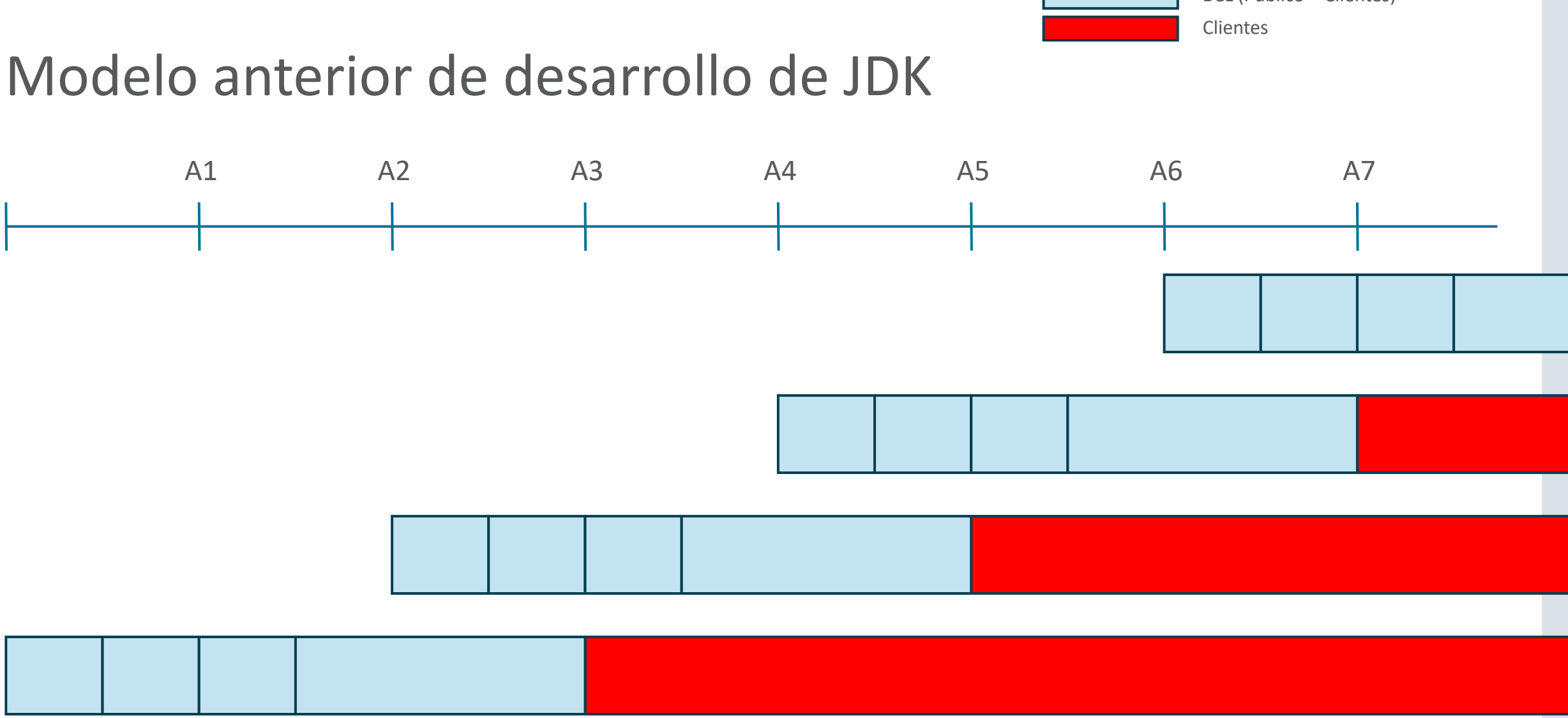
Modelo anterior de desarrollo de JDK



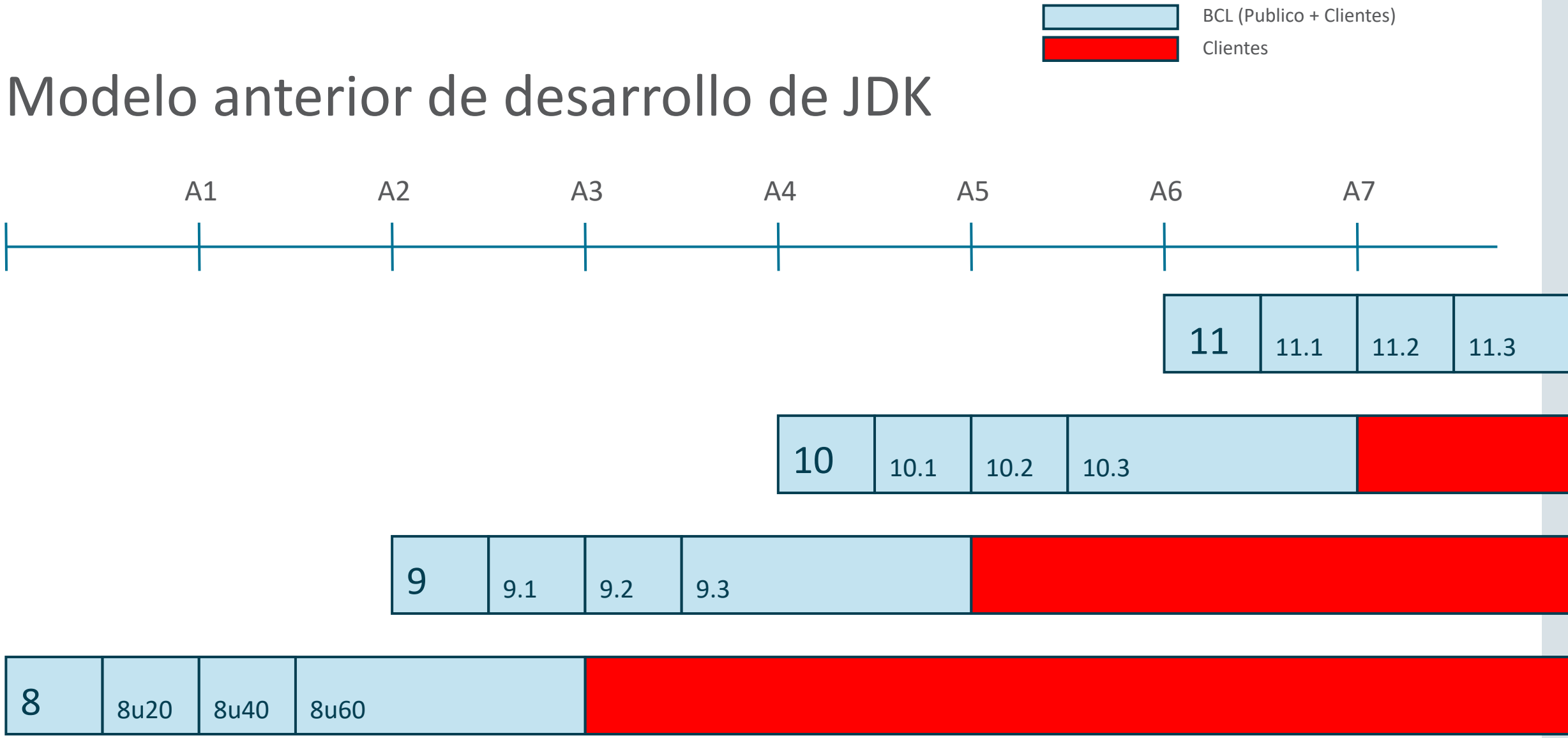
Modelo anterior de desarrollo de JDK



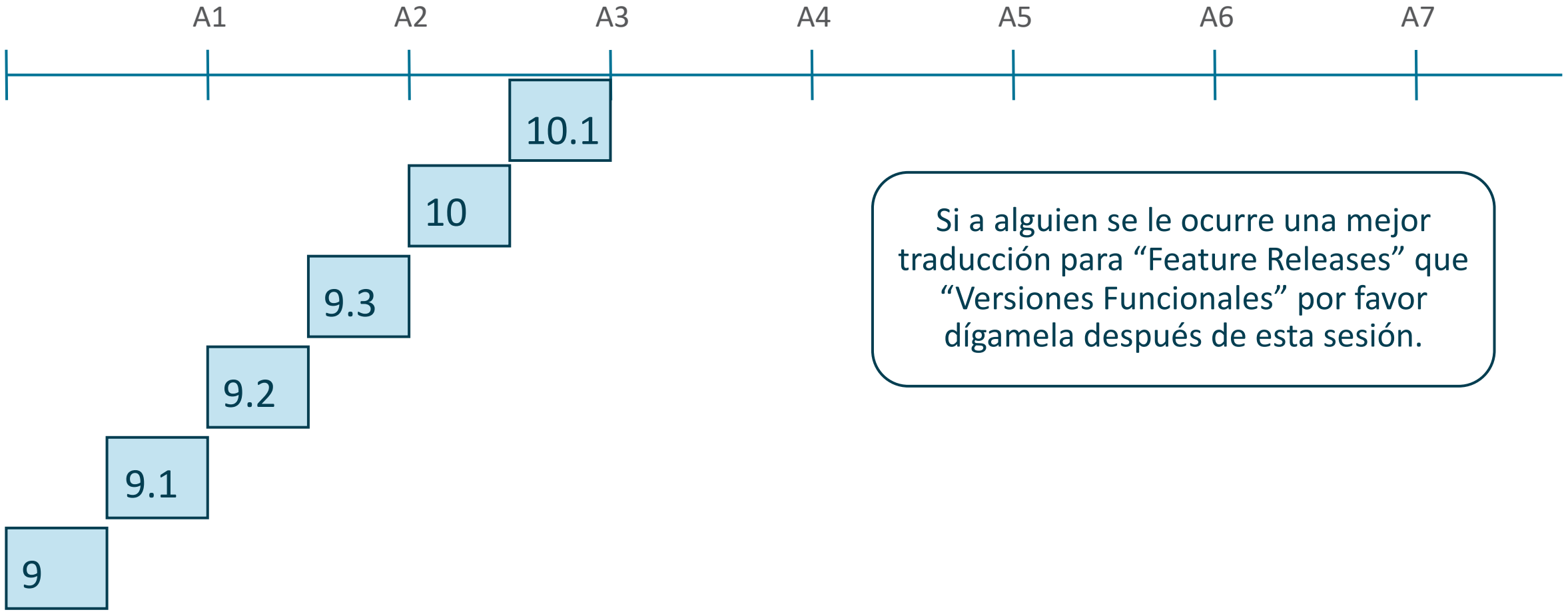
Modelo anterior de desarrollo de JDK



Modelo anterior de desarrollo de JDK

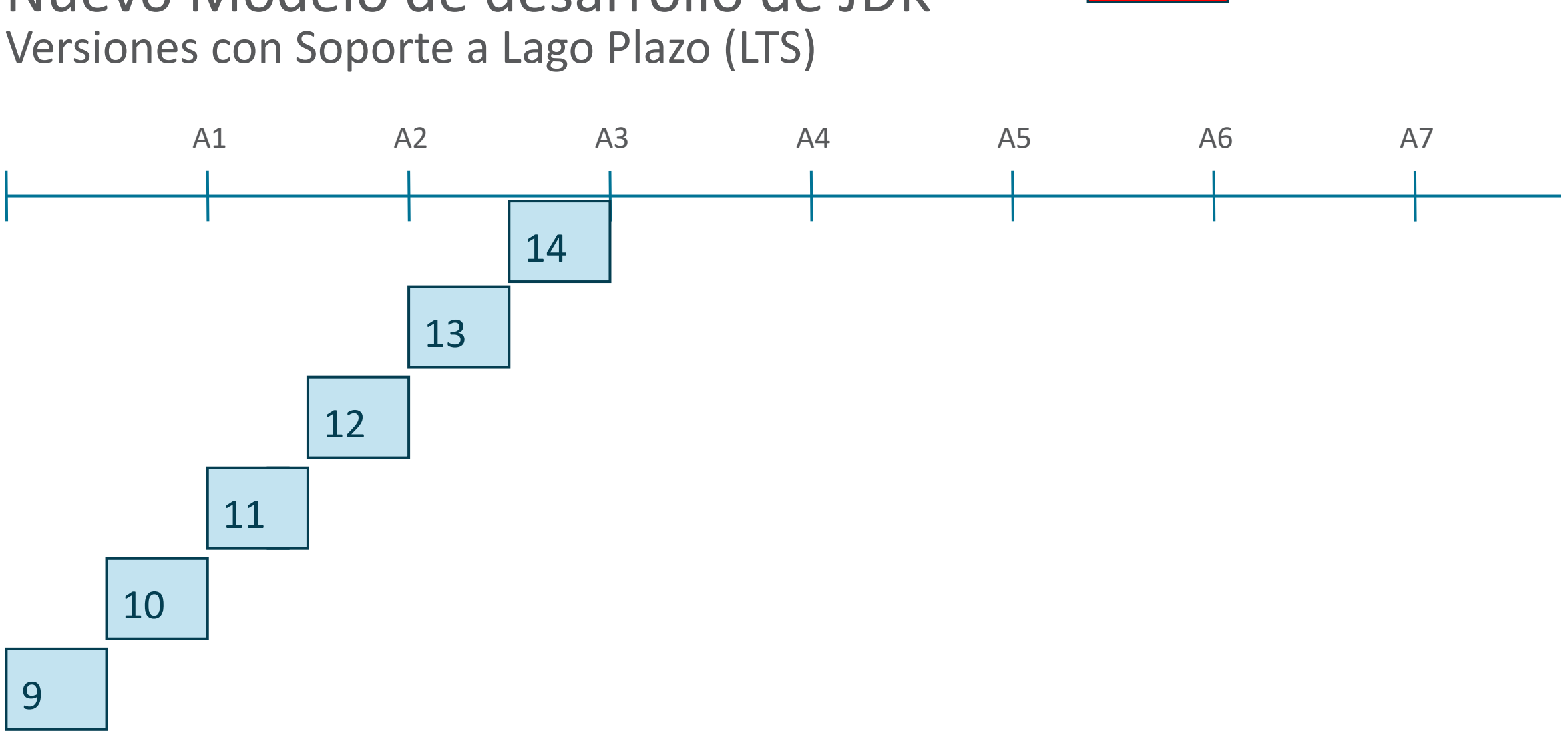


Nuevo Modelo de desarrollo de JDK
Versiones Funcionales cada 6 meses



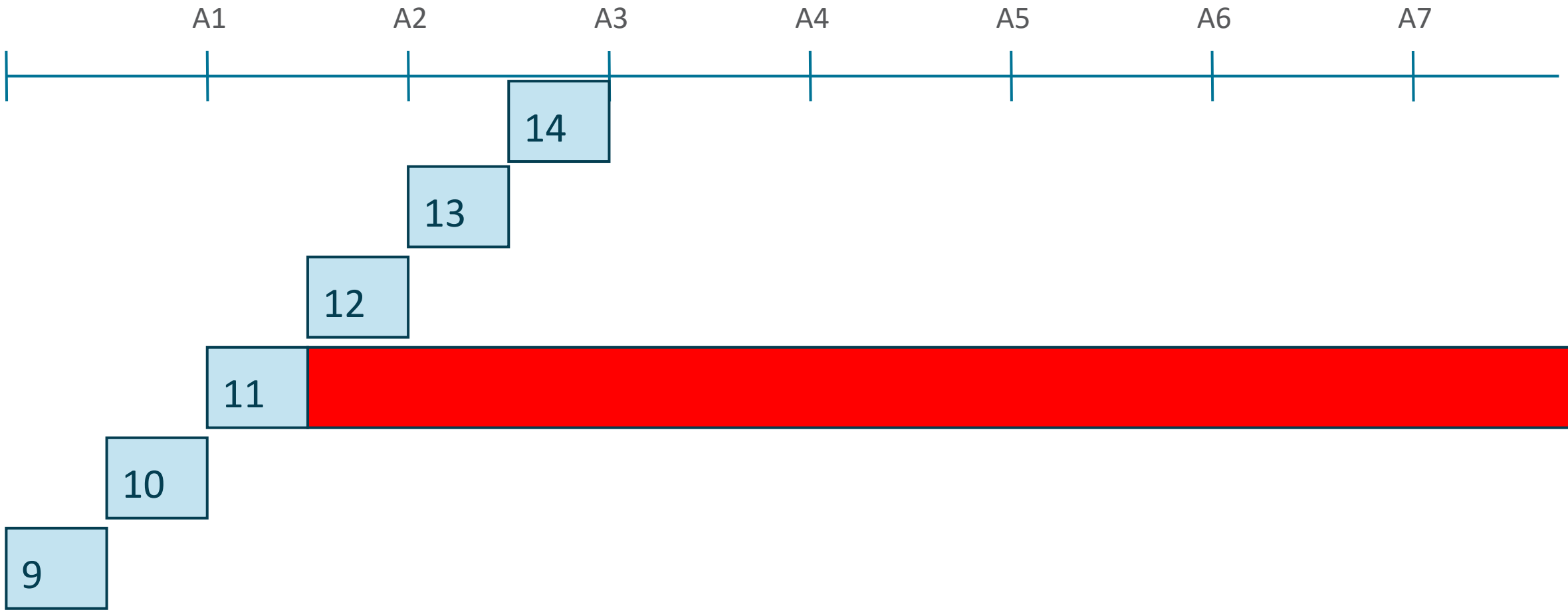
Si a alguien se le ocurre una mejor traducción para “Feature Releases” que “Versiones Funcionales” por favor dígamela después de esta sesión.

Nuevo Modelo de desarrollo de JDK
Versiones con Soporte a Lago Plazo (LTS)



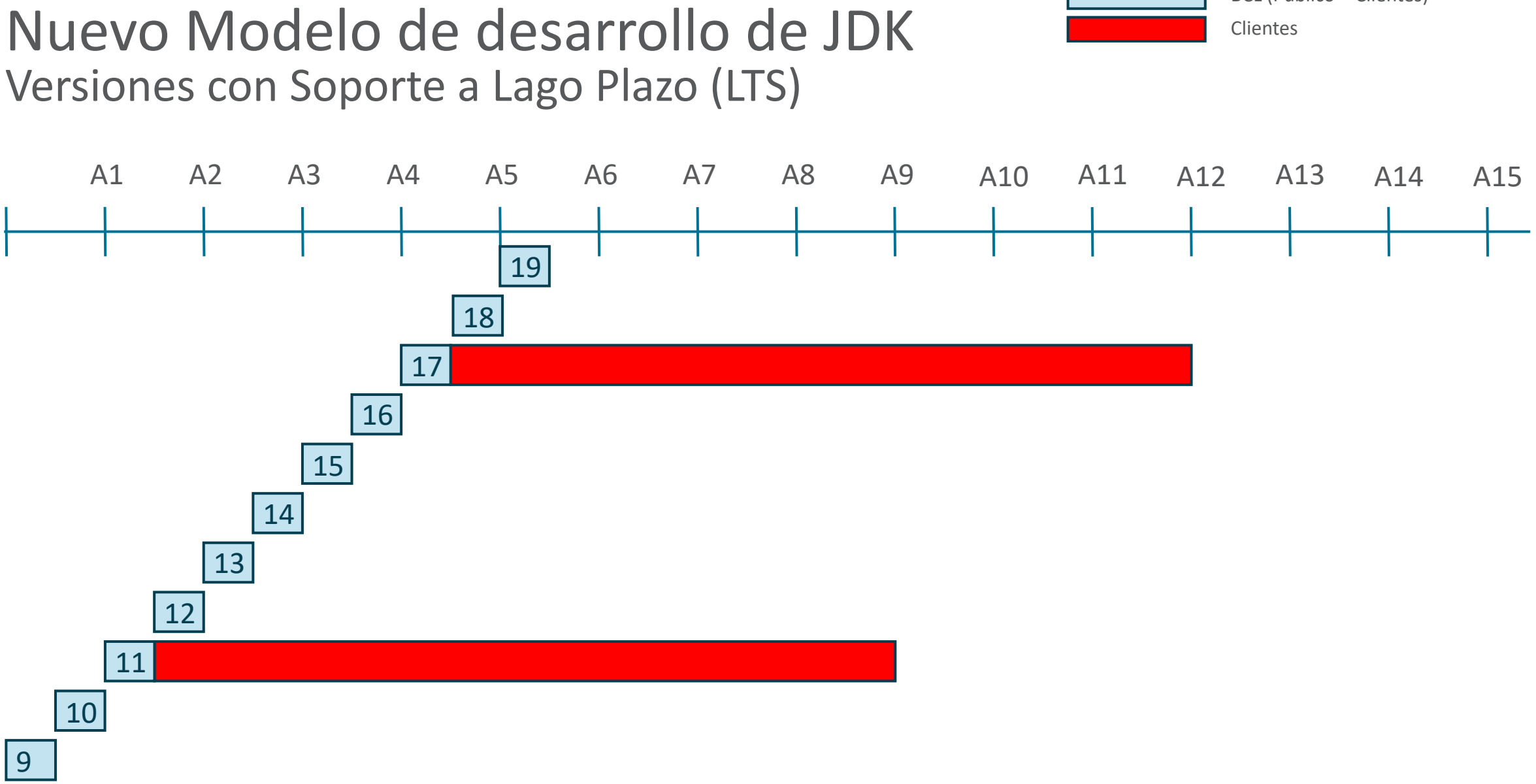
Nuevo Modelo de desarrollo de JDK

Versiones con Soporte a Lago Plazo (LTS)



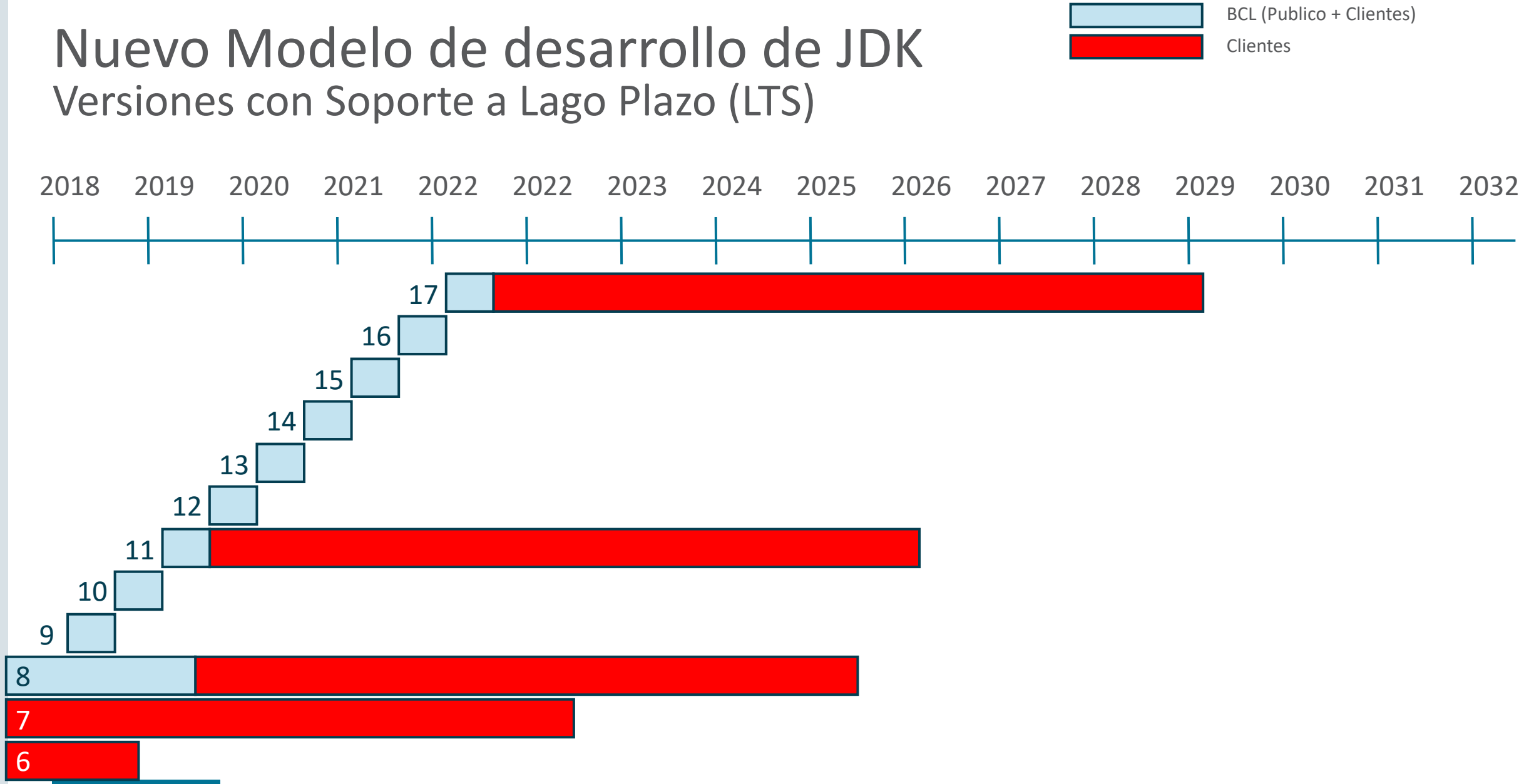
Nuevo Modelo de desarrollo de JDK

Versiones con Soporte a Lago Plazo (LTS)

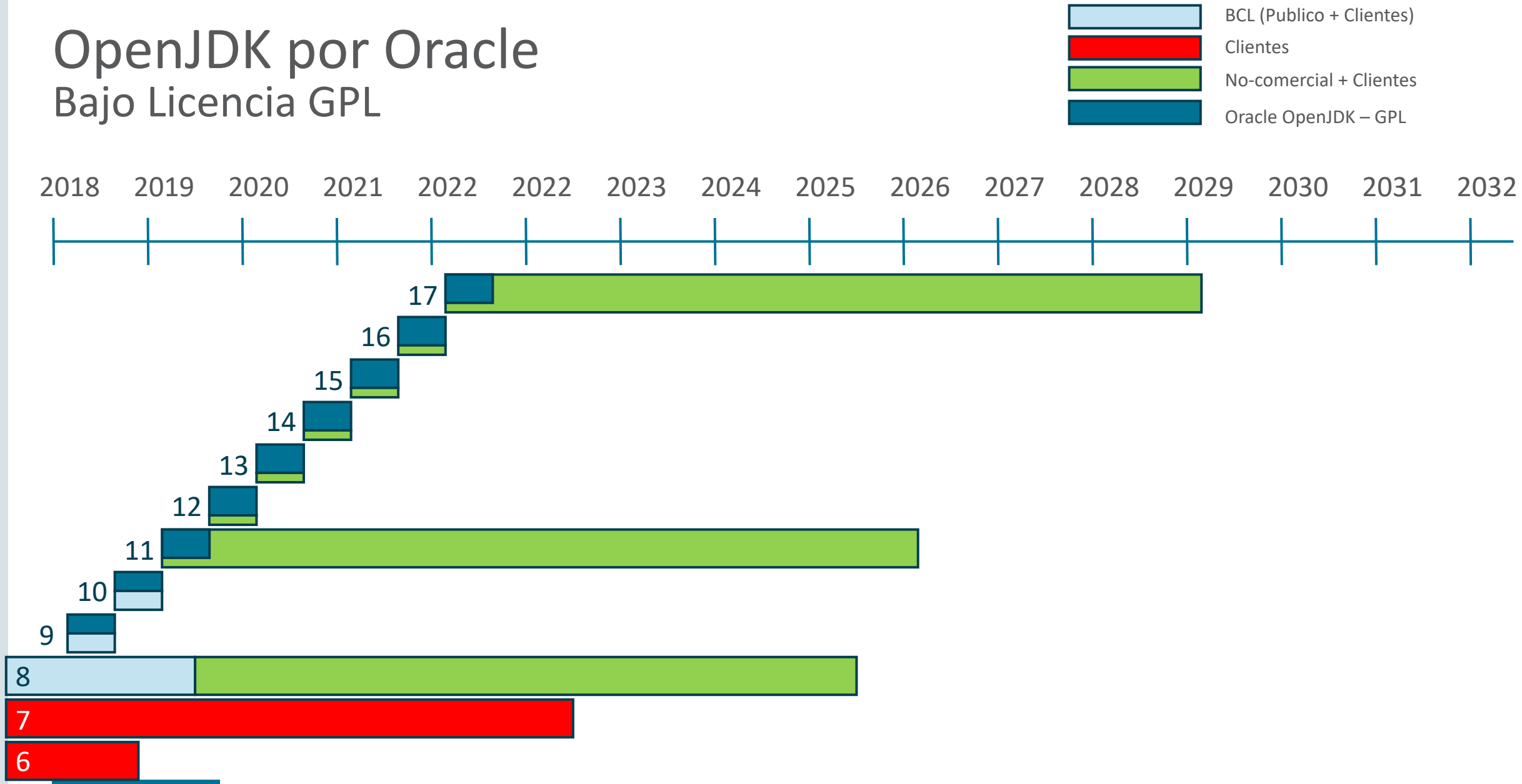


Nuevo Modelo de desarrollo de JDK

Versiones con Soporte a Lago Plazo (LTS)



OpenJDK por Oracle
Bajo Licencia GPL

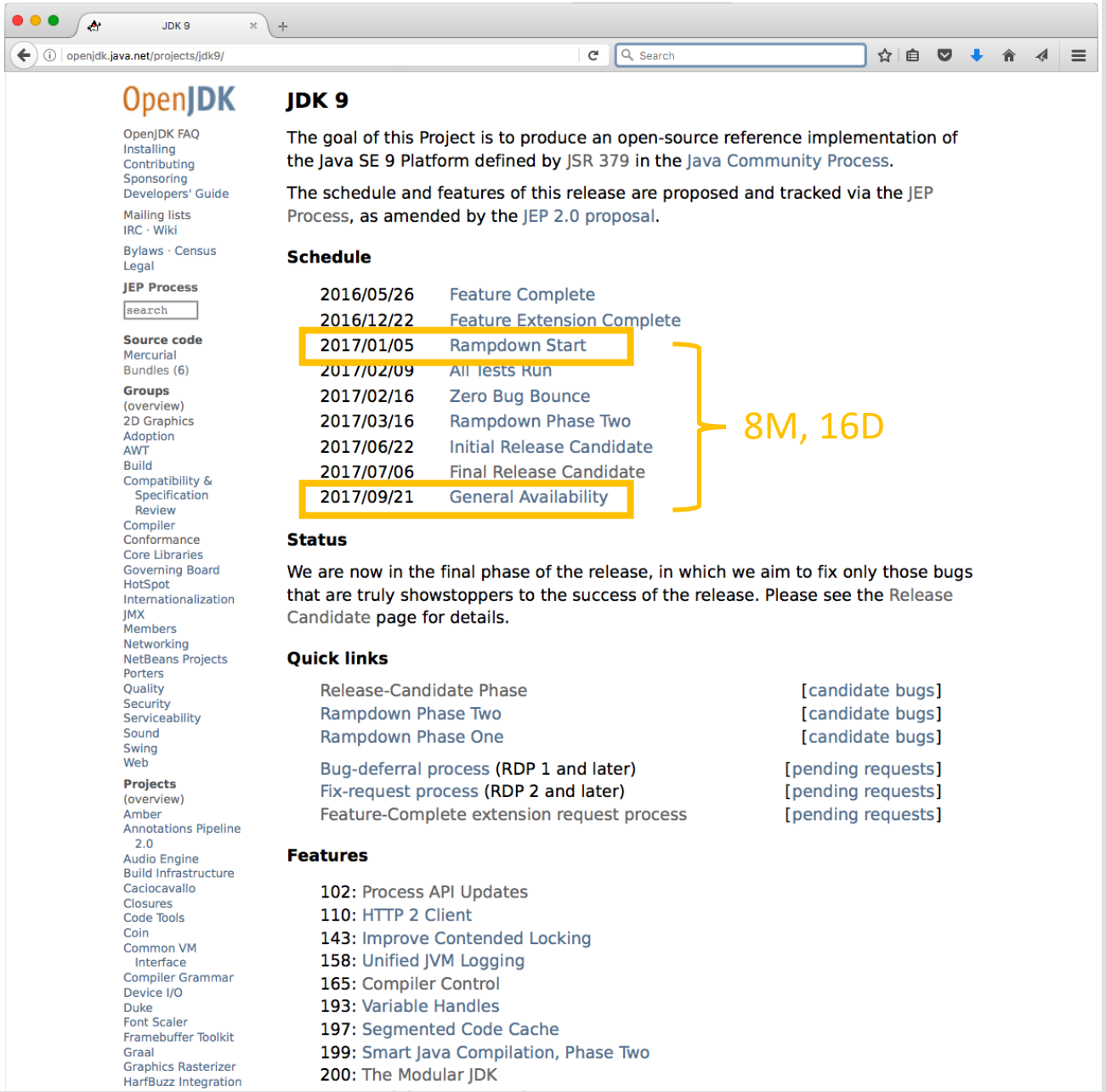


Cambios desde JDK 8

Desde marzo 2014

JDK 9

- GA Septiembre 2017
- La ultima versión “Mayor”
  - Release Driver: Jigsaw
  - 100+ mejoras
- Tiempo desde JDK 8: 3 años y 6 meses



Más información sobre cualquier JEP:
http://openjdk.java.net/jeps/{JEP#}

JDK 9

- GA Septiembre 2017
- La ultima versión “Mayor”
  - Release Driver: Jigsaw
  - 100+ mejoras
- Tiempo desde JDK 8: 3 años y 6 meses

Más información sobre cualquier 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

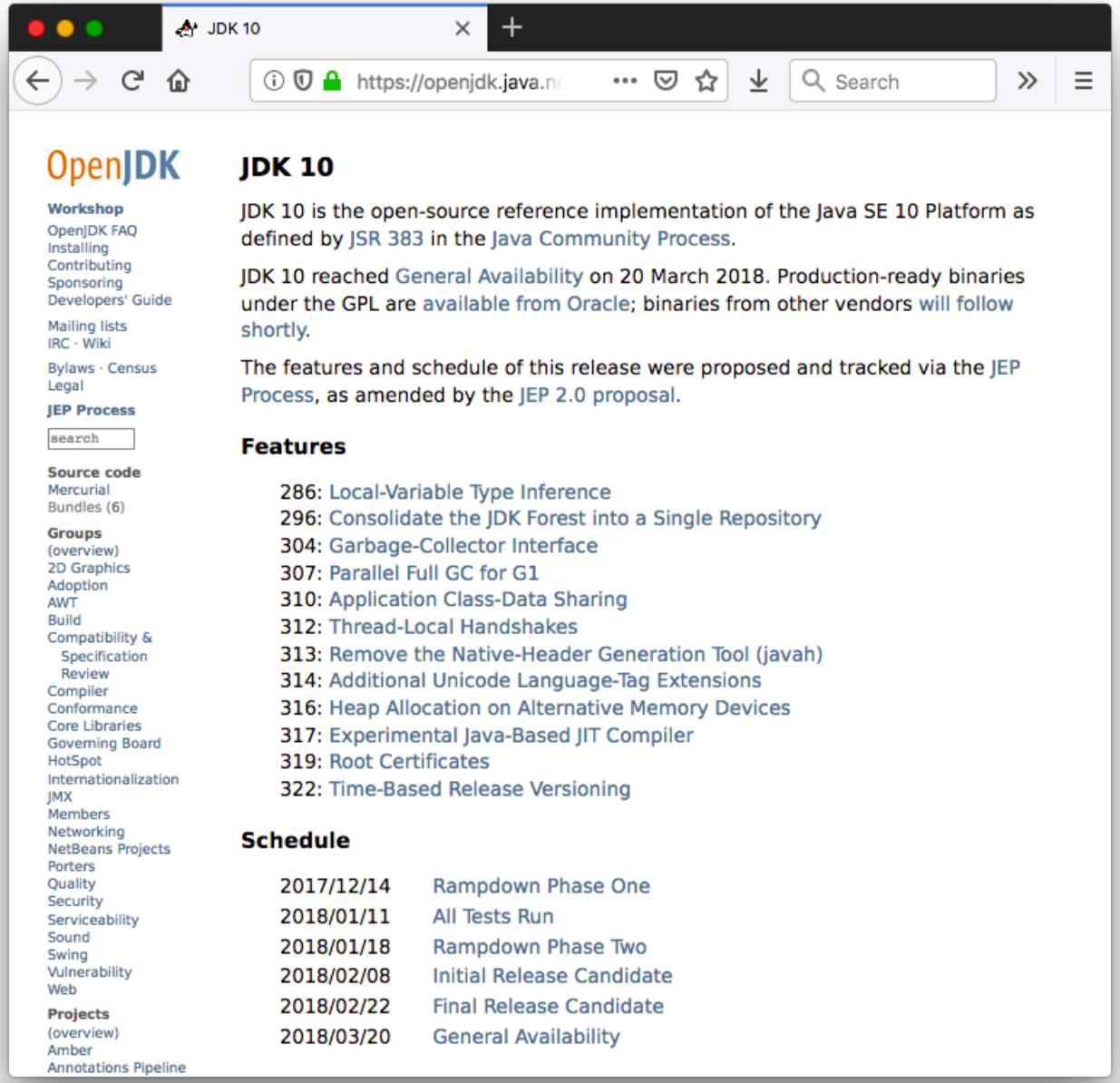
The milestone definitions for JDK 9 are the same as those for JDK 8, with the addition of:

- *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

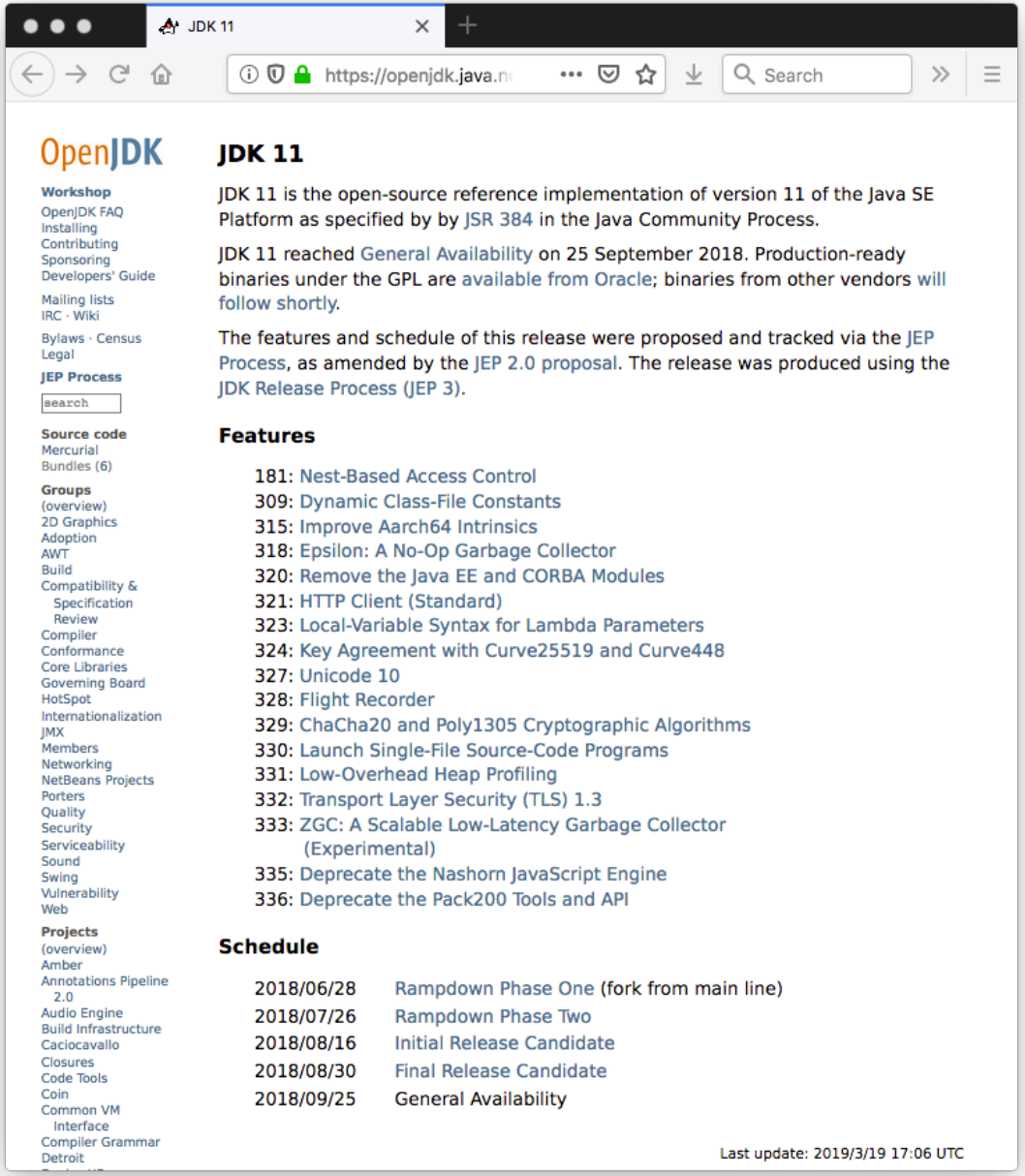
JDK 10 – Marzo 2018

- Primera Versión Funcional
- 12 JEPs (Java Enhancement Proposals)
- 6 meses después de Java 9



JDK 11 – Septiembre 2018

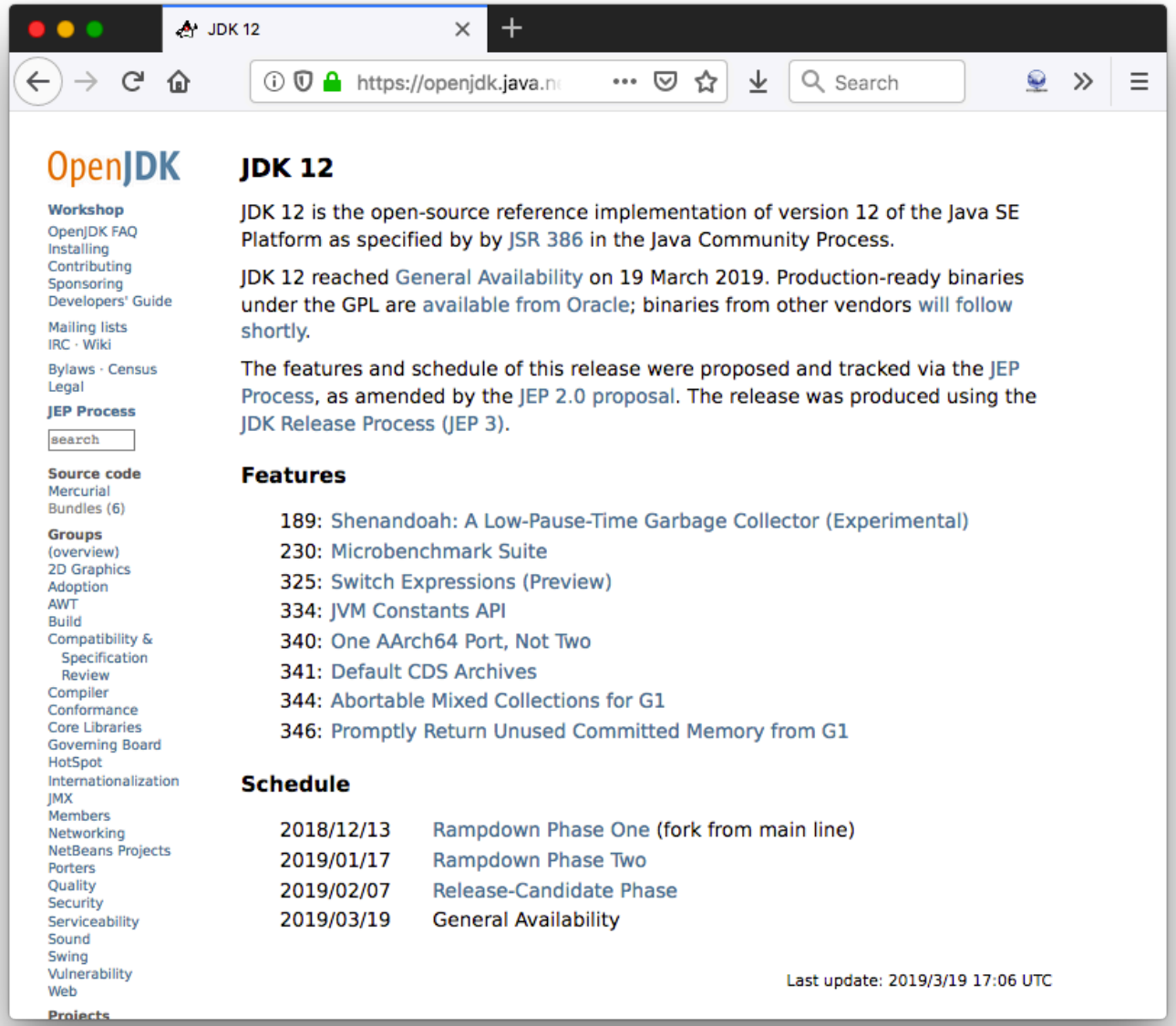
- 17 JEPs
  - TLS 1.3, la menor demora entre la aprobación de una especificación y su inclusión en una versión de JDK
    - Definido por RFC 8446 en Ago 2018
- Primera versión LTS bajo el nuevo modelo
- 6 meses después de Java 10



JDK 12 – Marzo 2019

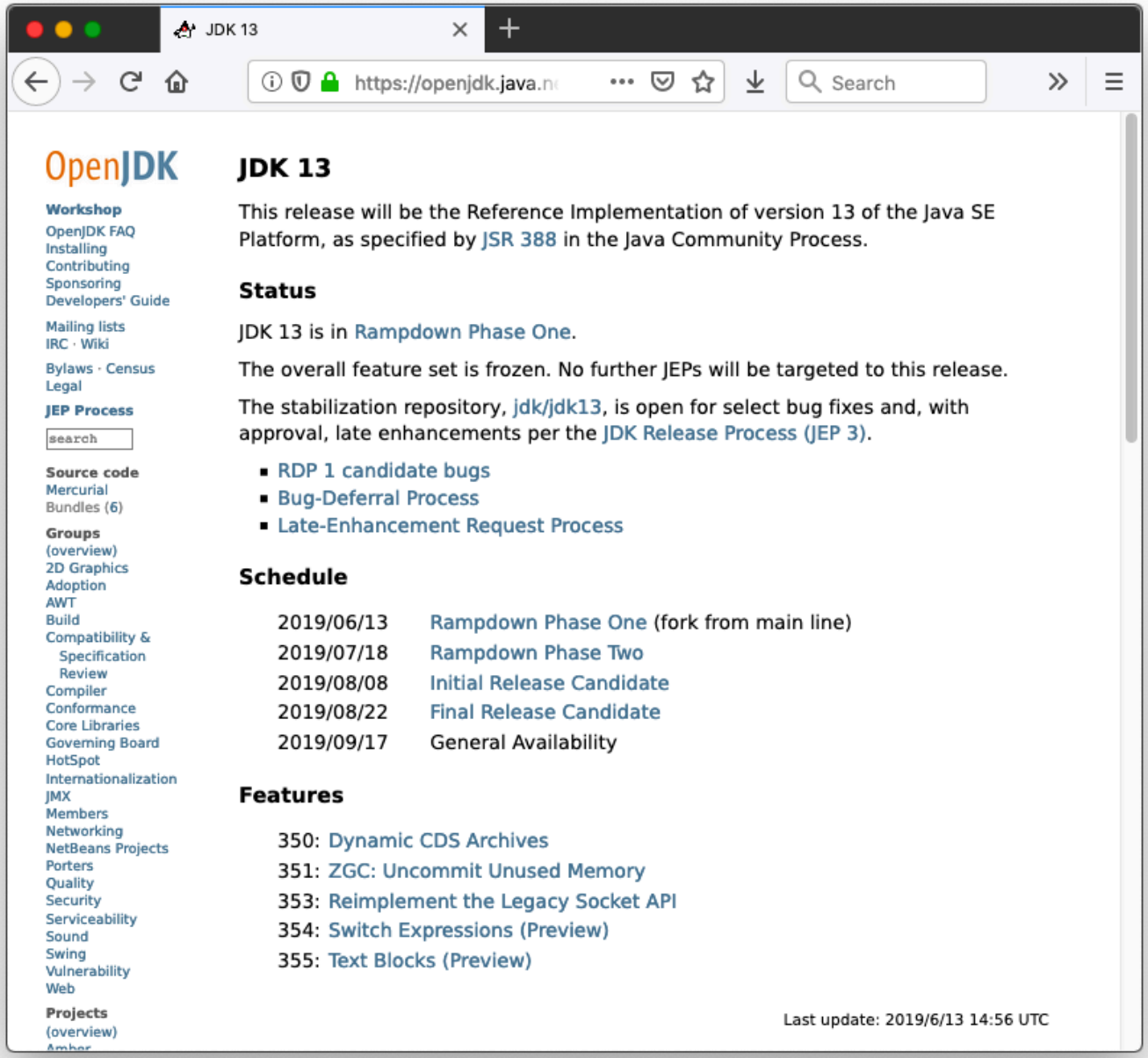
- 8 JEPs
  - Primera Funcionalidad Preliminar (Switch Expressions)
- 6 meses después de Java 11

¿Alguien empieza a notar un patrón aquí?



JDK 13 – Septiembre 2019

- 5 JEPs
- 2 Funcionalidades Preliminares
  - Switch Expression continua siendo preliminar con algunos cambios
  - Text Blocks
- 6 meses después de Java 12



Quien hace Java: Cambios resueltos en JDK 11

Quien hace Java: Cambios resueltos en JDK 12

Proyectos en OpenJDK

Trabajando en las mejoras para las siguientes versiones

Siguientes ~~Desafíos~~ Oportunidades

- Contenedores
- Predictibilidad
- Performance
- Optimización de Data
- Aceleración de HW
- Escalabilidad
- Mejoras continuas al lenguaje

Panama

Portola

Valhalla

Loom

Amber

openjdk.java.net

Proyecto Amber

- Completos
  - **Local-Variable Type Inference (var) *(JDK 10)***
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - Switch Expressions [Funcionalidad Preliminar] (JDK 12)
  - Text Blocks [Funcionalidad Preliminar] (JDK 12)
- En desarrollo
  - Mejoras para Lambda
  - Pattern Matching
  - Concise Method Bodies

```
URL url = new URL("http://www.oracle.com/"); ]

URLConnection conn = url.openConnection();

Reader reader = new BufferedReader(
      new InputStreamReader(conn.getInputStream()));
```

Proyecto Amber

- Completos
  - **Local-Variable Type Inference (var) *(JDK 10)***
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - Switch Expressions [Funcionalidad Preliminar] (JDK 12)
  - Text Blocks [Funcionalidad Preliminar] (JDK 12)
- En desarrollo
  - Mejoras para Lambda
  - Pattern Matching
  - Concise Method Bodies

```
var url = new URL("http://www.oracle.com/"); ]

var conn = url.openConnection();

var reader = new BufferedReader(
       new InputStreamReader(conn.getInputStream()));
```

Proyecto Amber

- Completos
  - Local-Variable Type Inference (var) *(JDK 10)*
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - **Switch Expressions [Funcionalidad Preliminar] (JDK 12)**
  - Text Blocks [Funcionalidad Preliminar] (JDK 12)
- En desarrollo
  - Mejoras para Lambda
  - Pattern Matching
  - Concise Method Bodies

```
int numLetters;
switch (day) {
    case MONDAY:
    case FRIDAY:
    case SUNDAY:
        numLetters = 6;
        break;
    case TUESDAY:
        numLetters = 7;
        break;
    case THURSDAY:
    case SATURDAY:
        numLetters = 8;
        break;
    case WEDNESDAY:
        numLetters = 9;
        break;
    default:
        throw new IllegalStateException(”wat: " + day);
}
```

Proyecto Amber

- Completos
  - Local-Variable Type Inference (var) *(JDK 10)*
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - **Switch Expressions [Funcionalidad Preliminar] (JDK 12)**
  - Text Blocks [Funcionalidad Preliminar] (JDK 12)
- En desarrollo
  - Mejoras para Lambda
  - Pattern Matching
  - Concise Method Bodies

```
int numLetters = switch (day) {
        case MONDAY, FRIDAY, SUNDAY   -> 6;
        case TUESDAY                  -> 7;
        case THURSDAY, SATURDAY       -> 8;
        case WEDNESDAY                -> 9;
};

 /*The cases of a switch expression must
 be exhaustive; for all possible values there
 must be a matching switch label.

 In practice this normally means that a
 default clause is required
 (except for enums)*/
```

Proyecto Amber

- Completos
  - Local-Variable Type Inference (var) *(JDK 10)*
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - Switch Expressions [Funcionalidad Preliminar] (JDK 12)
  - **Text Blocks [Funcionalidad Preliminar] (JDK 12)**
- En desarrollo
  - Mejoras para Lambda
  - Pattern Matching
  - Concise Method Bodies

```
String html = "<html>\n" +
              "    <body>\n" +
              "        <p>Hello World.</p>\n" +
              "    </body>\n" +
              "</html>\n";
```

Proyecto Amber

- Completos
  - Local-Variable Type Inference (var) *(JDK 10)*
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - Switch Expressions [Funcionalidad Preliminar] (JDK 12)
  - **Text Blocks [Funcionalidad Preliminar] (JDK 12)**
- En desarrollo
  - Mejoras para Lambda
  - Pattern Matching
  - Concise Method Bodies

```
String html = """
              <html>
                  <body>
                      <p>Hello World.</p>
                  </body>
              </html>
              """;
```

Proyecto Amber

- Completos
  - Local-Variable Type Inference (var) *(JDK 10)*
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - Switch Expressions [Funcionalidad Preliminar] (JDK 12)
  - **Text Blocks [Funcionalidad Preliminar] (JDK 12)**
- En desarrollo
  - Mejoras para Lambda
  - Pattern Matching
  - Concise Method Bodies

```
String html = """
              <html>
                  <body>
                      <p>Hello World.</p>
                  </body>
              </html>
              """;
```

Proyecto Amber

- Completos
  - Local-Variable Type Inference (var) *(JDK 10)*
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - Switch Expressions [Funcionalidad Preliminar] (JDK 12)
  - Text Blocks [Funcionalidad Preliminar] (JDK 12)
- En desarrollo
  - Mejoras para Lambda
  - **Pattern Matching**
  - Concise Method Bodies

```
String formatted = "unknown";
if (constant instanceof Integer) {
    int i = (Integer) constant;
    formatted = String.format("int %d", i);
}
else if (constant instanceof Byte) {
    byte b = (Byte) constant;
    formatted = String.format("byte %d", b);
}
else if (constant instanceof Long) {
    long l = (Long) constant;
    formatted = String.format("long %d", l);
}
else if (constant instanceof Double) {
    Double d = (Double) constant;
    formatted = String.format(“double %f", d);
}
// Short, Character, Float, Boolean
else if (constant instanceof String) {
    String s = (String) constant;
    formatted = String.format("String %s", s);
}
```

Proyecto Amber

- Completos
  - Local-Variable Type Inference (var) *(JDK 10)*
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - Switch Expressions [Funcionalidad Preliminar] (JDK 12)
  - Text Blocks [Funcionalidad Preliminar] (JDK 12)
- En desarrollo
  - Mejoras para Lambda
  - **Pattern Matching**
  - Concise Method Bodies

```
String formatted;
switch (constant) {
    case Integer i:
        formatted = String.format("int %d", i);
        break;
    case Byte b:
        formatted = String.format("byte %d", b);
        break;
    case Long l:
        formatted = String.format("long %d", l);
        break;
    case Double d:
        formatted = String.format("double %f", d);
        break;
    case String s:
        formatted = String.format("String %s", s);
         break;
    // Short, Character, Float, Boolean
    default:
        formatted = "unknown";
}
```

Proyecto Amber

- Completos
  - Local-Variable Type Inference (var) *(JDK 10)*
  - Local-Variable Syntax for Lambda Parameters (JDK 11)
  - **Switch Expressions [Funcionalidad Preliminar] (JDK 12)**
  - Text Blocks [Funcionalidad Preliminar] (JDK 12)
- En desarrollo
  - Mejoras para Lambda
  - **Pattern Matching**
  - Concise Method Bodies

```
String formatted =
    switch (constant) {
      case Integer i -> String.format("int %d", i);
      case Byte b    -> String.format("byte %d", b);
      case Long l    -> String.format("long %d", l);
      case Double d  -> String.format("double %f", d);
      case String s  -> String.format("String %s", s);
     // Short, Character, Float, Boolean
      default -> "unknown";
    }
}
```

Advertencia:
La siguiente sección no ha sido traducida, y la veremos a la velocidad del estornudo….
(toda la información esta en OpenJDK 9)

Sección bonus

Solo para los que todavia tienen Java 9 en el futuro

Behind the scenes improvements

Goodness you get for free just by updating to JDK 9
No need for user to change anything to benefit from these

JEP 254: Compact Strings

core-libs / java.lang

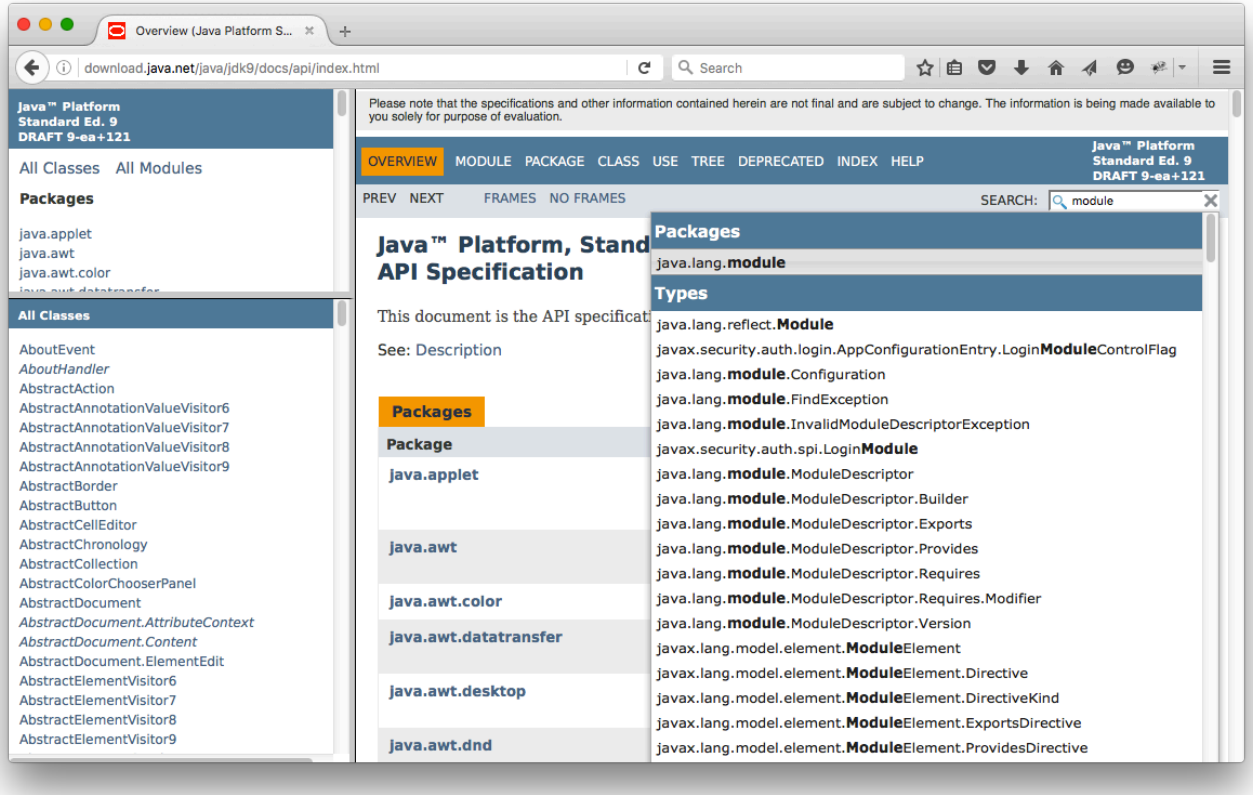
- Adopt a more space-efficient internal representation for strings

- Less memory used for storing strings
- String class stores characters in a char array, using two bytes (sixteen bits) for each character
- Change the internal representation of the String class to a byte array plus an encoding-flag field

JEP 225: Javadoc Search

tools / javadoc(tool)

- Add a search box to generated API documentation that can be used to search for program elements and tagged words and phrases within the documentation



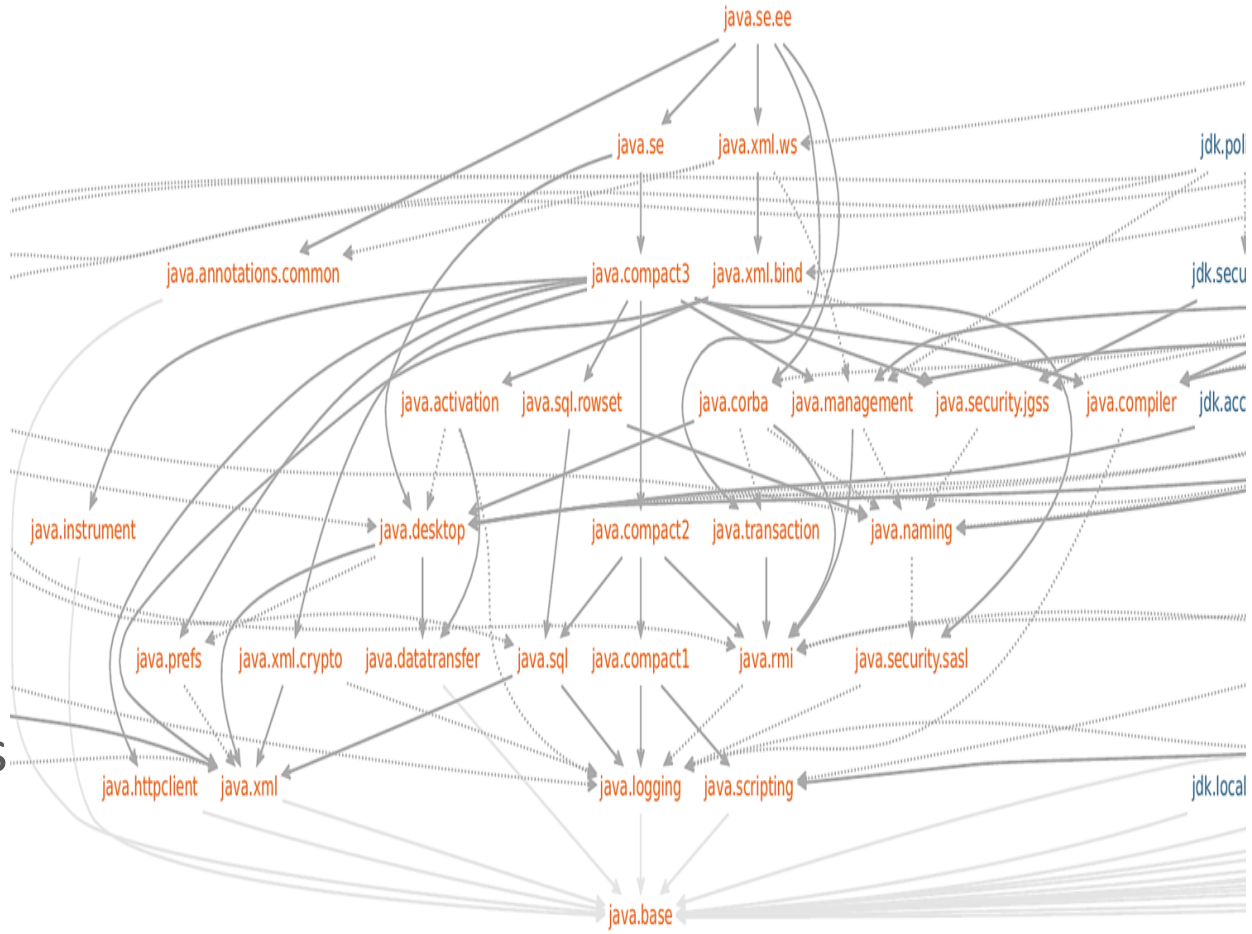
New features and functionality

New tools and capabilities likely to be useful to most developers
Will have to choose to use these

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

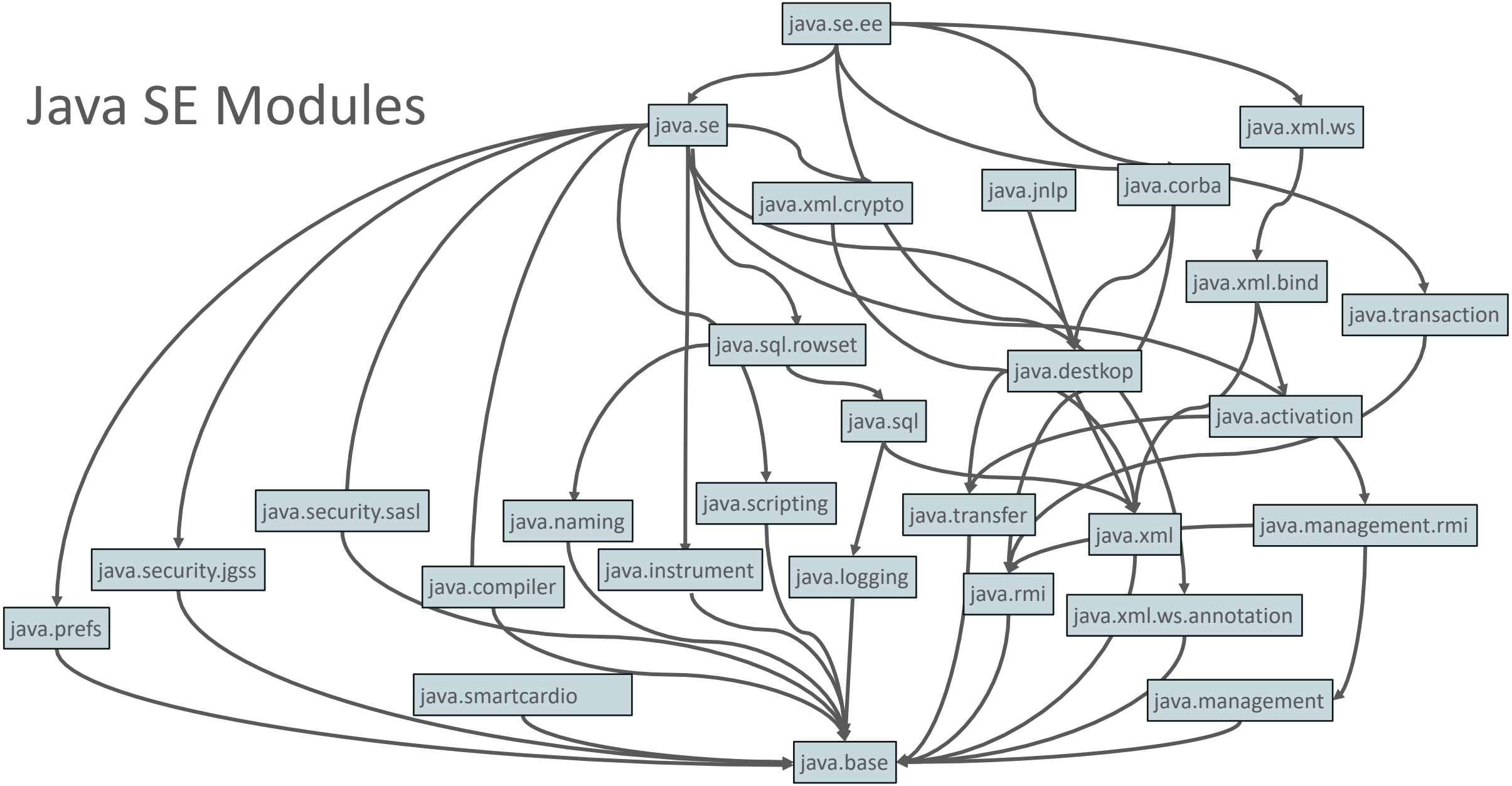


JEP 282: jlink: The Java Linker

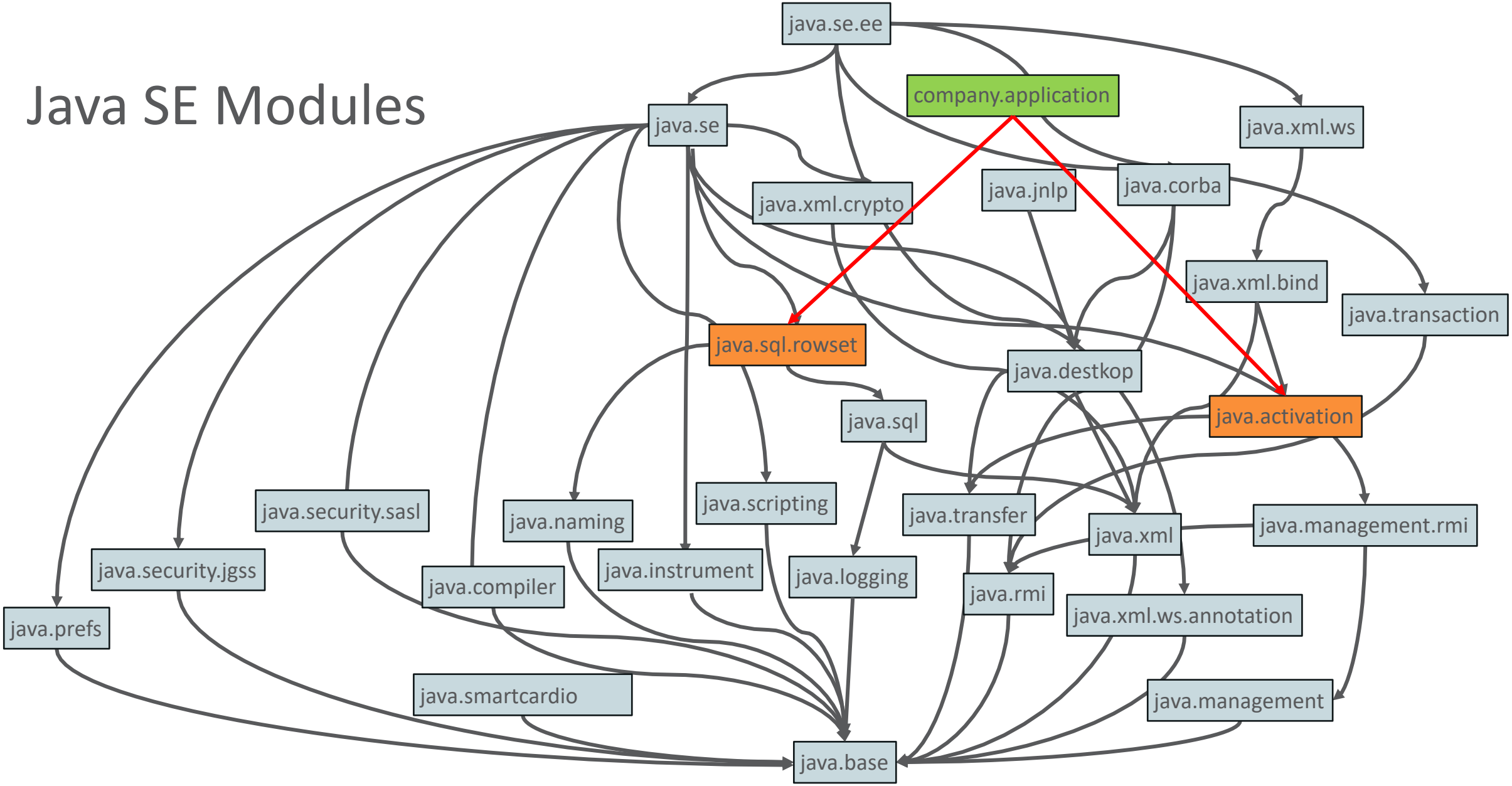
tools / jlink

- 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

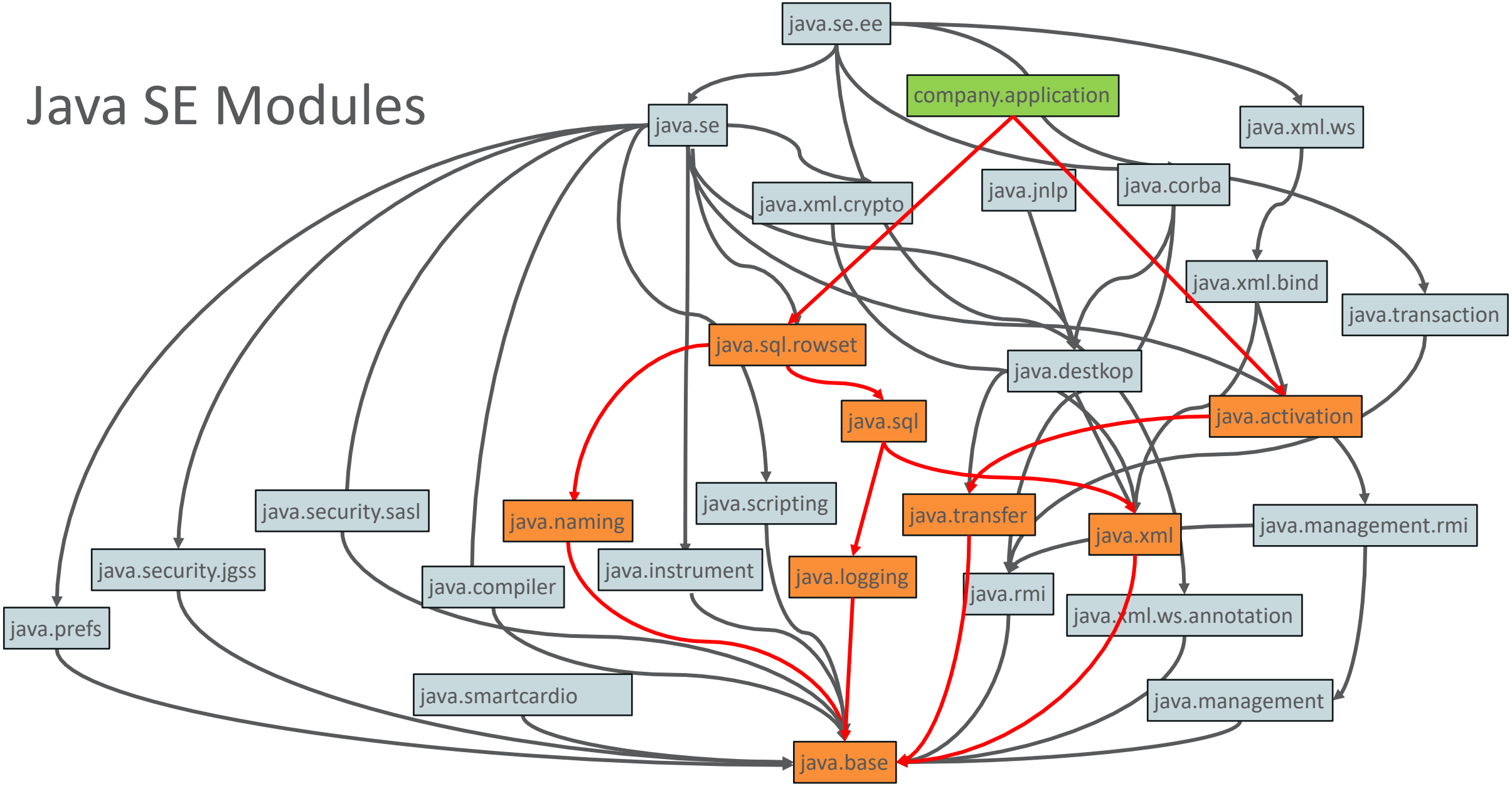
Java SE Modules



Java SE Modules

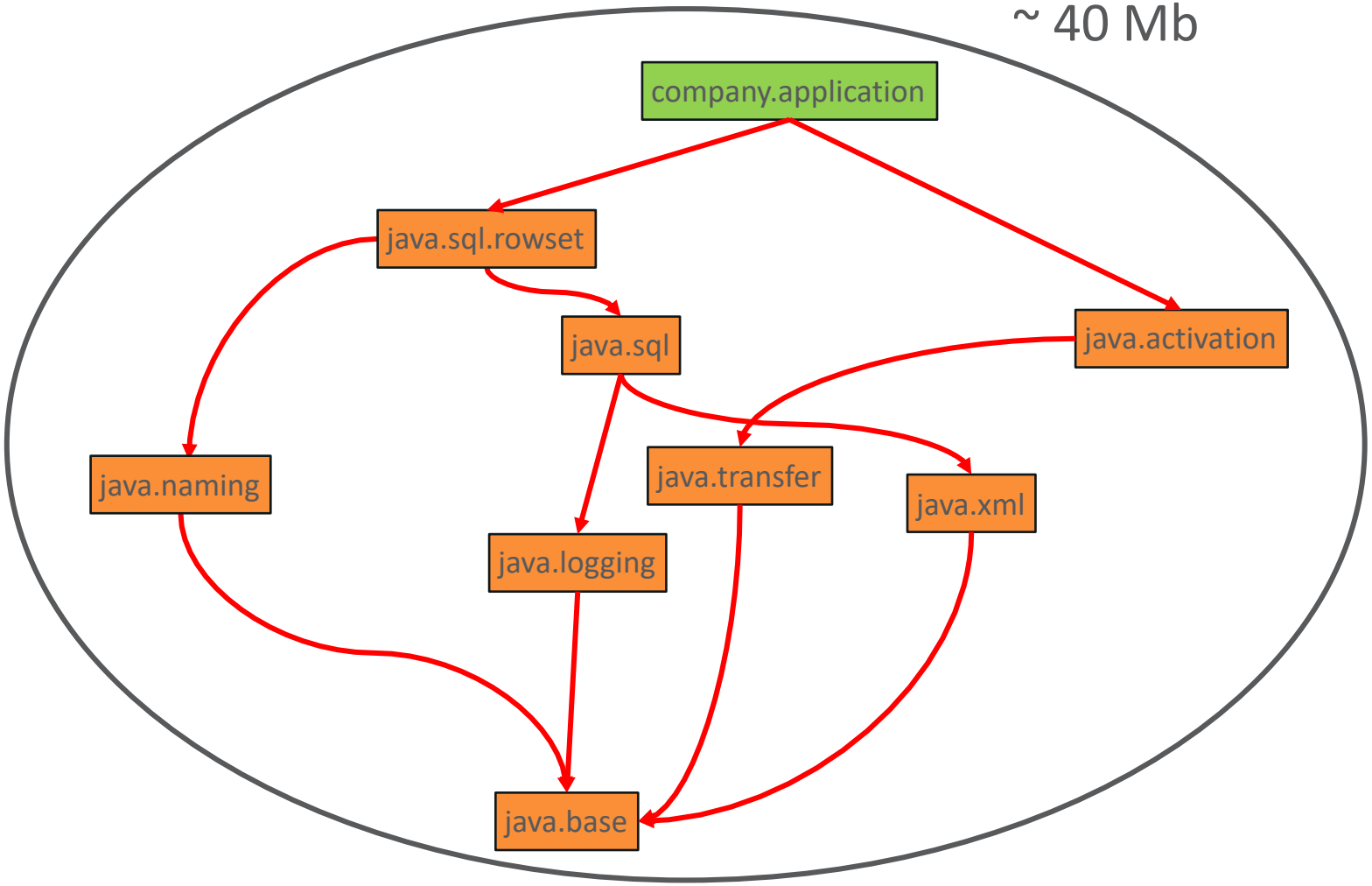


Java SE Modules



Java Custom Runtime

- Includes the Modular Application



JEP 277: Enhanced Deprecation

core-libs / java.lang

@Deprecated(since=9, forRemoval=true)

- Revamp the deprecation annotation, and provide tools to strengthen the API life cycle
- Provide better information about the status and intended disposition of APIs in the specification
- Provide a tool to analyze an application's static usage of deprecated APIs
- Provide a tool to detect an application's dynamic usage of of deprecated APIs in order to emit warnings at runtime

JEP 269: Convenience Factory Methods for Collections

core-libs / java.util:collections

- Define library APIs to make it convenient to create instances of collections and maps with small numbers of elements, so as to ease the pain of not having collection literals in the Java programming language

- Decrease the amount of code needed for creating small collections and maps

```
Set<String> alphabet = Set.of("a", "b", "c");
```

JEP 222: jshell: The Java Shell (Read-Eval-Print Loop)

tools / jshell

- Provide an interactive tool to evaluate declarations, statements, and expressions of the Java programming language, together with an API so that other applications can leverage this functionality
- A Read-Eval-Print Loop (REPL) is an interactive programming tool which loops, continually reading user input, evaluating the input, and printing the value of the input or a description of the state change the input caused. Scala, Ruby, JavaScript, Haskell, Clojure, and Python all have REPLs and all allow small initial programs. JShell adds REPL functionality to the Java platform

Adopting new standards

JDK 9 keeping up with improvements in the industry

JEP 287: SHA-3 Hash Algorithms

security-libs / java.security

- Implement the SHA-3 cryptographic hash functions (BYTE-only) specified in NIST FIPS 202

JEP 224: HTML5 Javadoc

tools / javadoc(tool)

- Enhance the javadoc tool to allow generating HTML5 markup.

JEP 110: HTTP/2 Client

core-libs / java.net

- Define a new HTTP client API that implements HTTP/2 and WebSocket, and can replace the legacy HttpURLConnection API
- For JDK 9 this API will be on the incubator modules (JEP 11) with the goal of adding them to the standard on a later release.

- Store Interned Strings in CDS Archives
- Improve Contended Locking
- Compact Strings
- Improve Secure Application Performance
- Leverage CPU Instructions for GHASH and RSA
- Tiered Attribution for javac
- Javadoc Search
- Marlin Graphics Renderer
- HiDPI Graphics on Windows and Linux
- Enable GTK 3 on Linux
- Update JavaFX/Media to Newer Version of GStreamer

Behind the scenes

- **Jigsaw – Modularize JDK**
- Enhanced Deprecation
- Stack-Walking API
- Convenience Factory Methods for Collections
- Platform Logging API and Service
- jshell: The Java Shell (Read-Eval-Print Loop)
- Compile for Older Platform Versions
- Multi-Release JAR Files
- Platform-Specific Desktop Features
- TIFF Image I/O\
- Multi-Resolution Images

New functionality

- Process API Updates
- Variable Handles
- Spin-Wait Hints
- Dynamic Linking of Language-Defined Object Models
- Enhanced Method Handles
- More Concurrency Updates
- Compiler Control

Specialized

- HTTP 2 Client
- Unicode 8.0
- UTF-8 Property Files
- Implement Selected ECMAScript 6 Features in Nashorn
- Datagram Transport Layer Security (DTLS)
- OCSP Stapling for TLS
- TLS Application-Layer Protocol Negotiation Extension
- SHA-3 Hash Algorithms
- DRBG-Based SecureRandom Implementations
- Create PKCS12 Keystores by Default
- Merge Selected Xerces 2.11.0 Updates into JAXP
- XML Catalogs
- HarfBuzz Font-Layout Engine
- HTML5 Javadoc

New standards

- Parser API for Nashorn
- Prepare JavaFX UI Controls & CSS APIs for Modularization
- Modular Java Application Packaging
- New Version-String Scheme
- Reserved Stack Areas for Critical Sections
- Segmented Code Cache
- Ahead-of-Time Compilation
- Indify String Concatenation
- Unified JVM Logging
- Unified GC Logging
- Make G1 the Default Garbage Collector
- Use CLDR Locale Data by Default
- Validate JVM Command-Line Flag Arguments
- Java-Level JVM Compiler Interface
- Disable SHA-1 Certificates
- Simplified Doclet API
- Deprecate the Applet API
- Process Import Statements Correctly
- Annotations Pipeline 2.0
- Elide Deprecation Warnings on Import Statements
- Milling Project Coin
- Filter Incoming Serialization Data

Housekeeping

- Remove GC Combinations Deprecated in JDK 8
- Remove Launch-Time JRE Version Selection
- Remove the JVM TI hprof Agent
- Remove the jhat Tool

Gone

Y ahora.. regresamos a nuestra programa original…

Proyecto Portola

Java en un mundo de contenedores

- Las características de Java lo hacen ideal para ambientes con contenedores
  - Seguro y protegido, alta performance, confiable, rico ecosistema
- Estamos comprometidos en mantener a Java como la mejor opción para aplicaciones en la nube

Proyecto Panama

- Funciones y Data nativos/en otro lenguaje
- Remplazo simple, seguro y de alta performance para JNI
- Acceso directo a funciones de hardware de bajo nivel usando Java
  - Instrucciones Vectoriales, tipos especial de memoria (ej. memoria no-volátil)
- Big Data, Machine Learning

Proyecto Skara

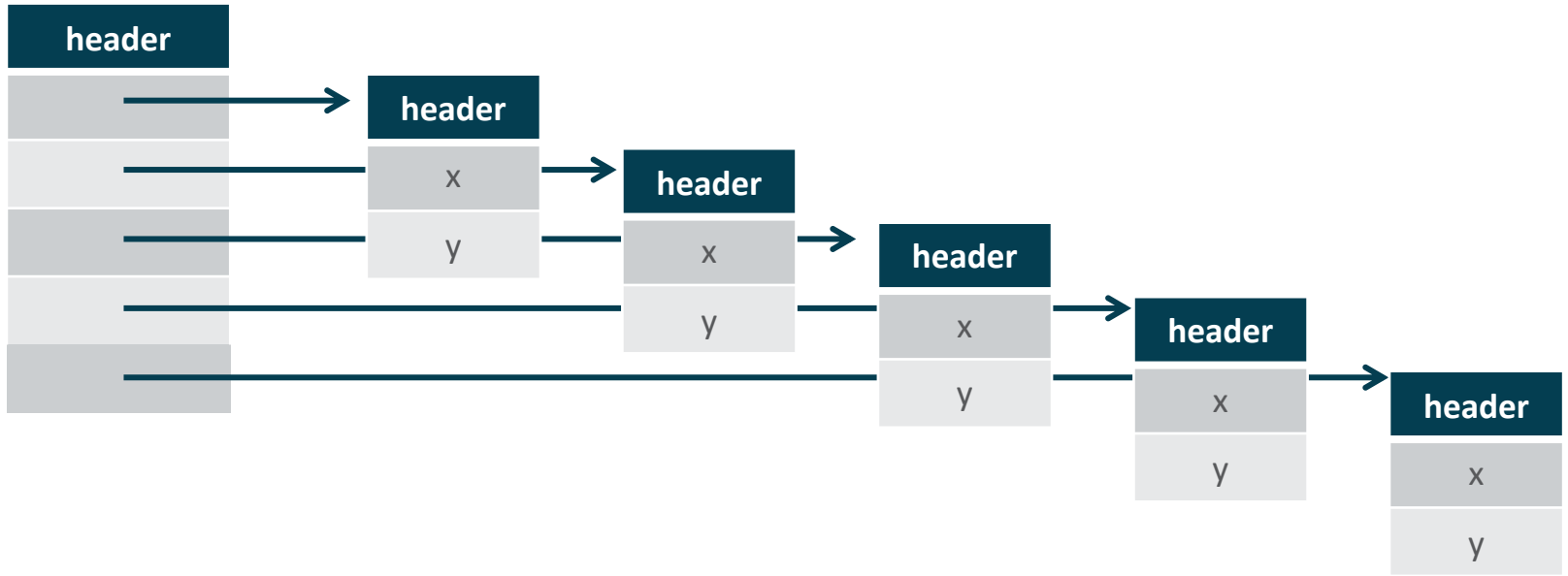
- Investiga alternativas de software de control de versiones para el código fuente del JDK
- Las opciones investigadas están basadas en Git en lugar de Mercurial
- El objetivo es permitir que los colaboradores en OpenJDK sean mas productivos, tanto los expertos como los novatos

Proyecto Valhalla

Data en Memoria con Java hoy

```
final class Point {
    final int x;
    final int y;
}

Point[] pts =
```

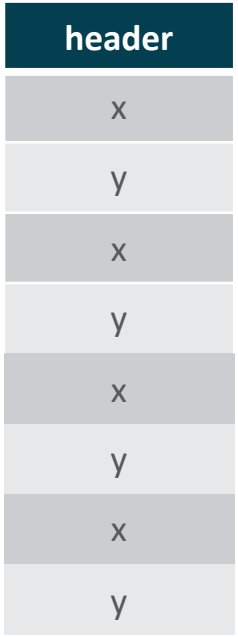


Proyecto Valhalla

Data en Memoria como nos gustaría

```
inline class Point {
    int x;
    int y;
}

Point[] pts =
```



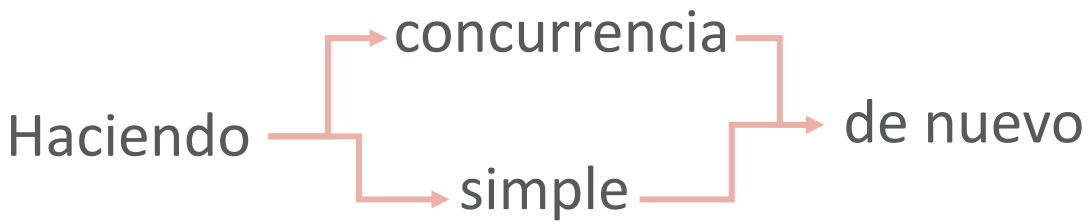
Value Types:

Tan sencillo de usar como una clase, con la performance de una primitiva.

¿Qué haría int?

Proyecto Loom

- Modelo para concurrencia mas sencillo y escalable
  - Hacer llamadas bloqueantes prácticamente sin costo
- “Fibras” (como hilos pero menos costosos) y continuaciones
  - Millones de fibras pueden ser creadas en una sola instancia del JVM



OpenJDK
versiones preliminares

- Futuras Versiones Funcionales
- Proyectos de OpenJDK

Resumen

- El desarrollo de la plataforma Java en OpenJDK se esta volviendo mas abierto
- JDK 11, la primera versión LTS , incluye innovación y mejores en seguridad, productividad y performance
- Oracle ha introducido una suscripción de Java para hacerlo mas asequible y mantener los costos predecibles
- Continuamos trabajando en diversos frentes para traer mejoras a Java cada seis meses

ÚNETE y conviértete en un **COLLABORDOR** de **OpenJDK**

https://openjdk.java.net

MANTENTE INFORMADO

https://blogs.oracle.com/java-platform-group

@OpenJDK @gsaab @mreinhold @BrianGoetz

@BTraTra @DonaldOJDK @MikaelVidstedt

@Sharat_Chander @robilad

@aureliog @SimmsUpNorth

Q & A