O'REILLY®
oscon
open source convention

Java™

oscon.com

The State of JDK and OpenJDK

Joseph D. Darcy
Oracle

#include "std_orcl_disclaimer.h"

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.

Outline

- The JDK and OpenJDK
- The JCP
- Compatibility
- JDK 7 Features
- JDK 8 Features
- Q & A

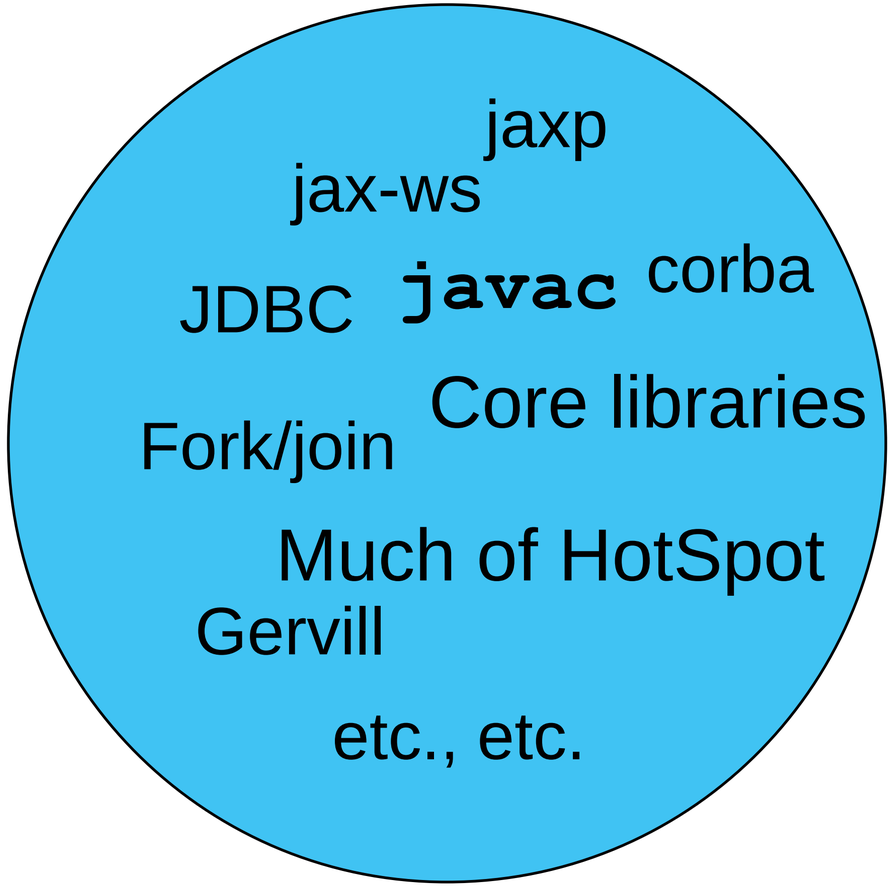
What are JDK and OpenJDK?

- The JDK is an implementation of a Java SE specification (amongst other things)
  - Bundled Webstart / plugin are *not* part of Java SE
- OpenJDK is
  - A community
  - A set of projects
  - A set of code bases

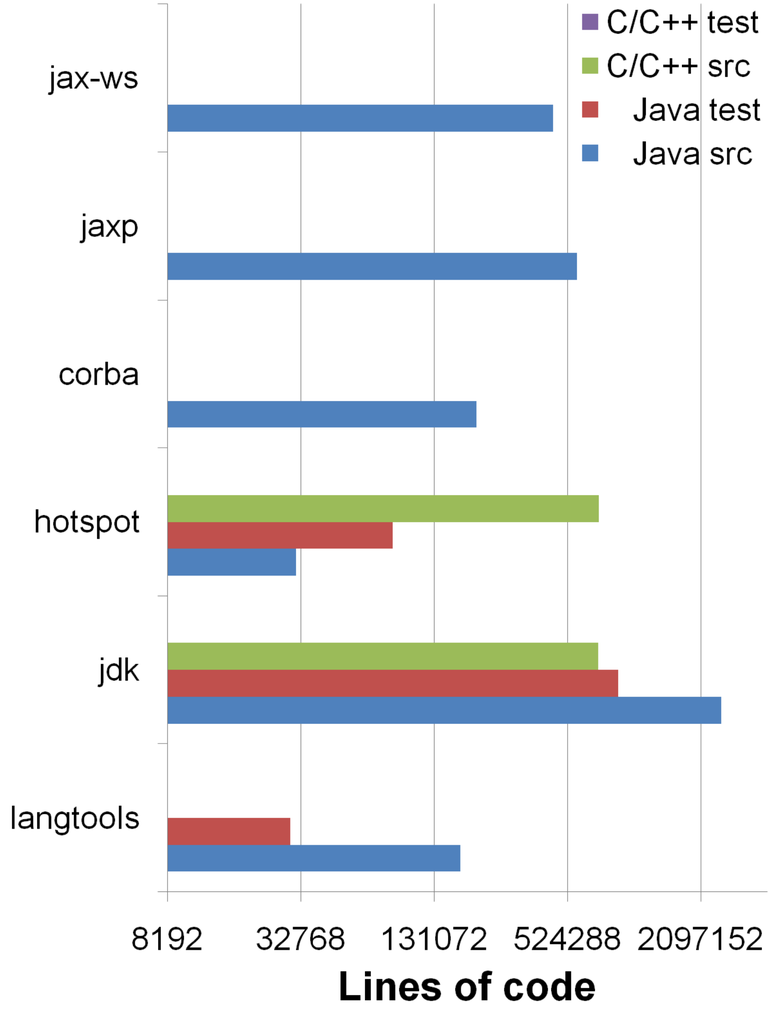
- Oracle and others develop the reference implementation of Java SE specifications in OpenJDK.
- Much of this code is reused in Oracle's JDK product
- Related projects, including experimental projects, also occur as part of OpenJDK
- More on OpenJDK at OSCON:
  OpenJDK – When And How To Contribute To The Java SE Reference Implementation
  Dalibor Topic (Oracle, Corp.)
  3:30pm Tuesday

http://hg.openjdk.java.net/jdk7/jdk7

OpenJDK



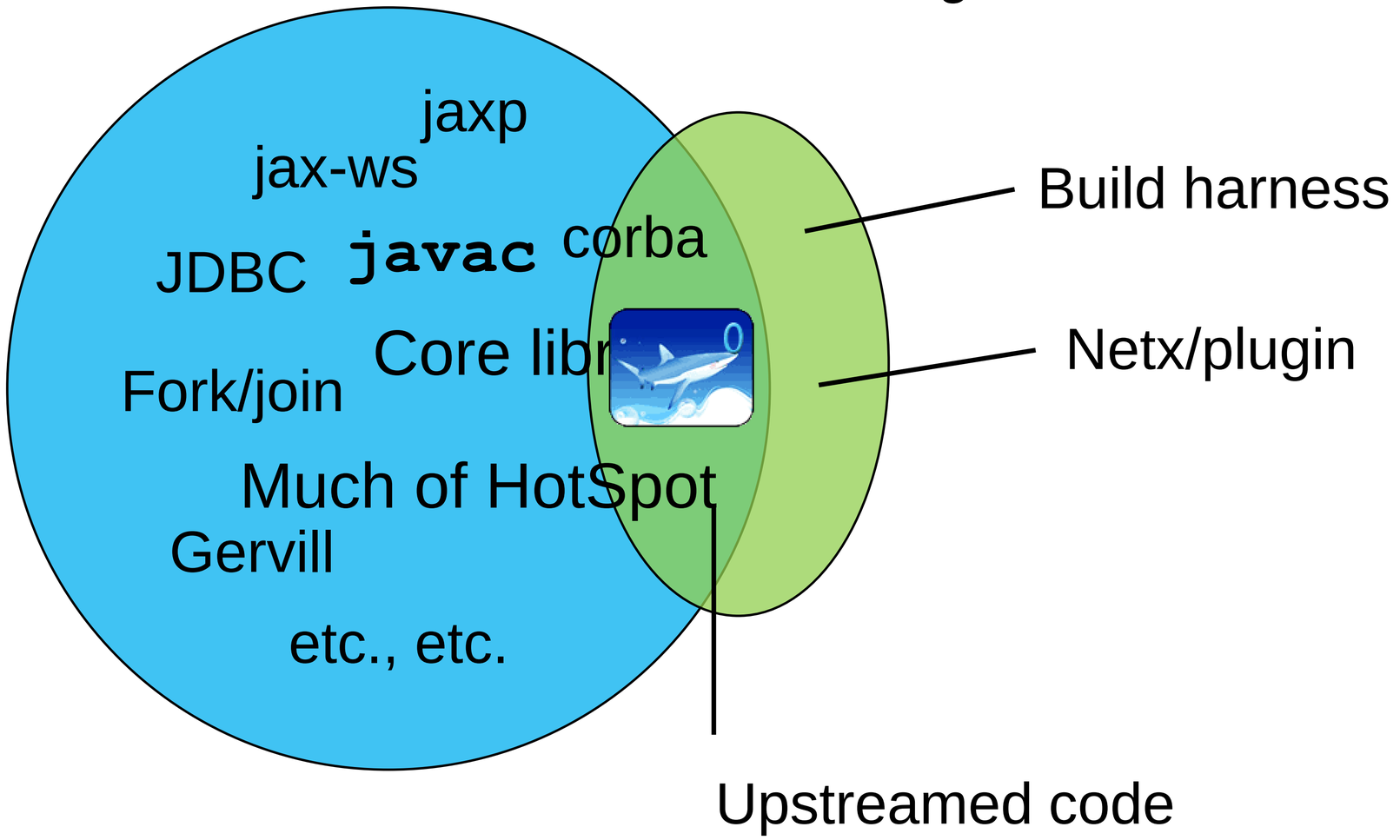
Note: log scale on x-axis.



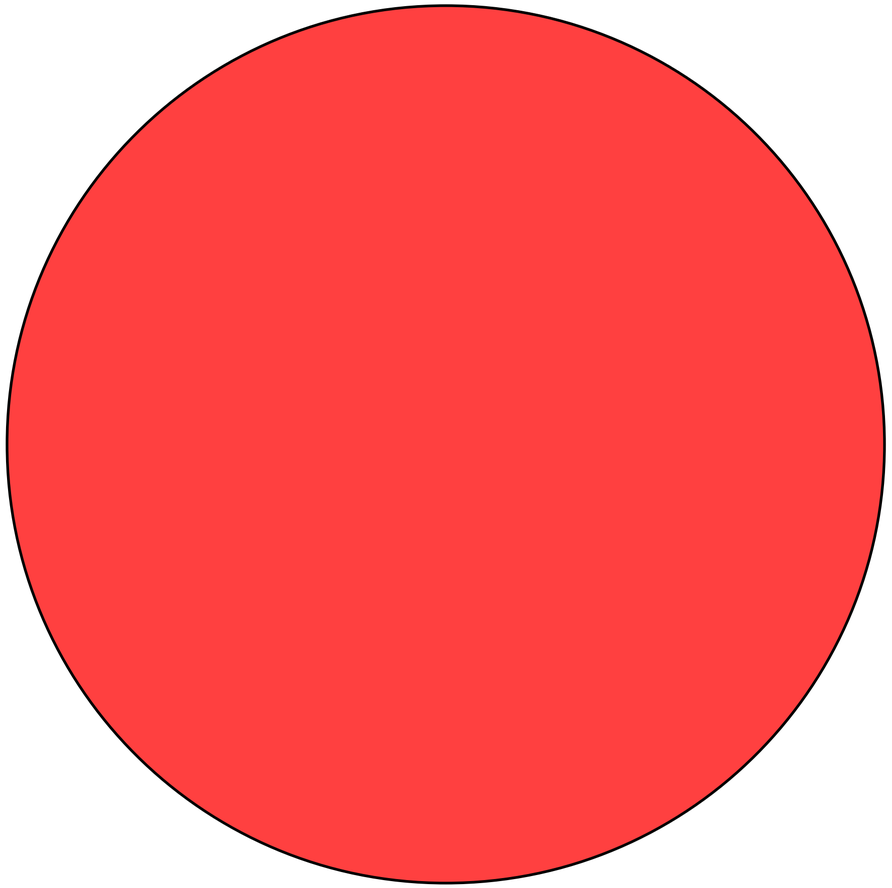
IcedTea — http://icedtea.classpath.org

Note: figure not drawn to scale.

OpenJDK



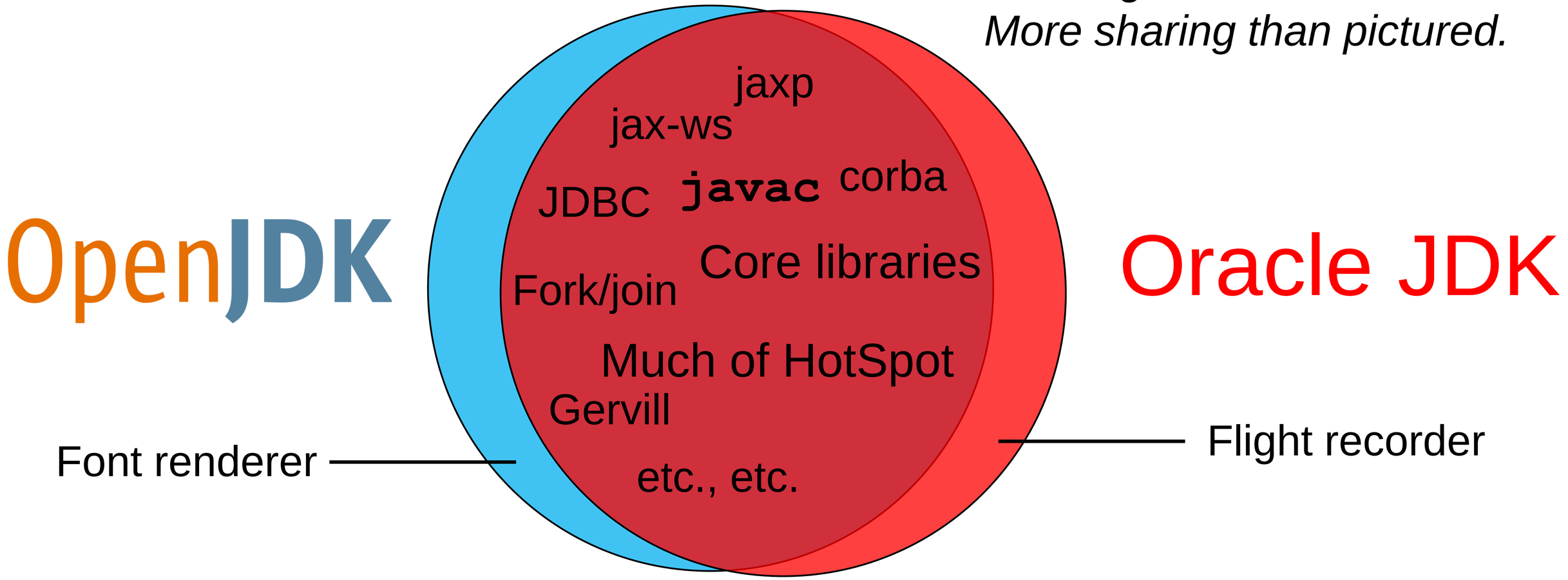
Graphic from http://today.java.net/pub/a/today/2009/05/21/zero-and-shark-openjdk-port.html



Oracle JDK

“We have a lot in common.”

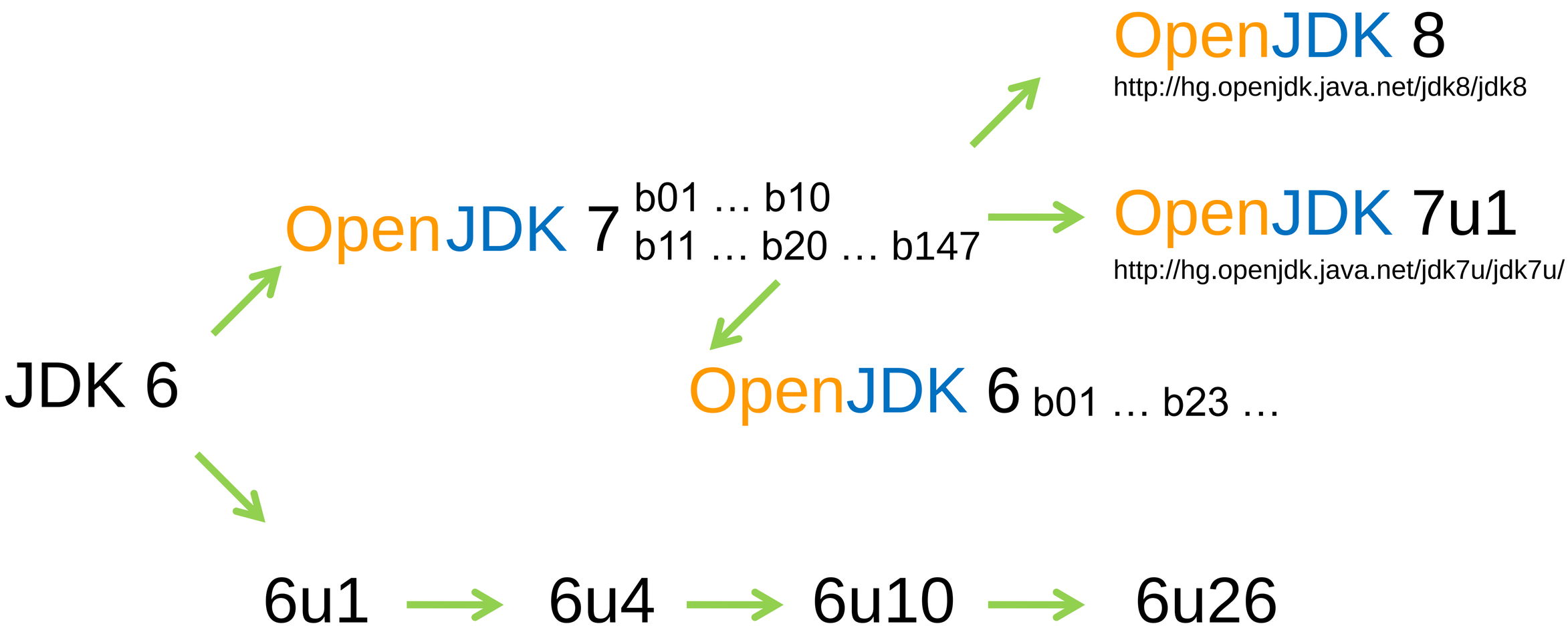
Note: figure not drawn to scale.
More sharing than pictured.



Priorities of Oracle's JDK Group

- Keep Java vibrant
- Enable revenue
- Increase efficiency of Java developers

OpenJDK History and Release Genealogy

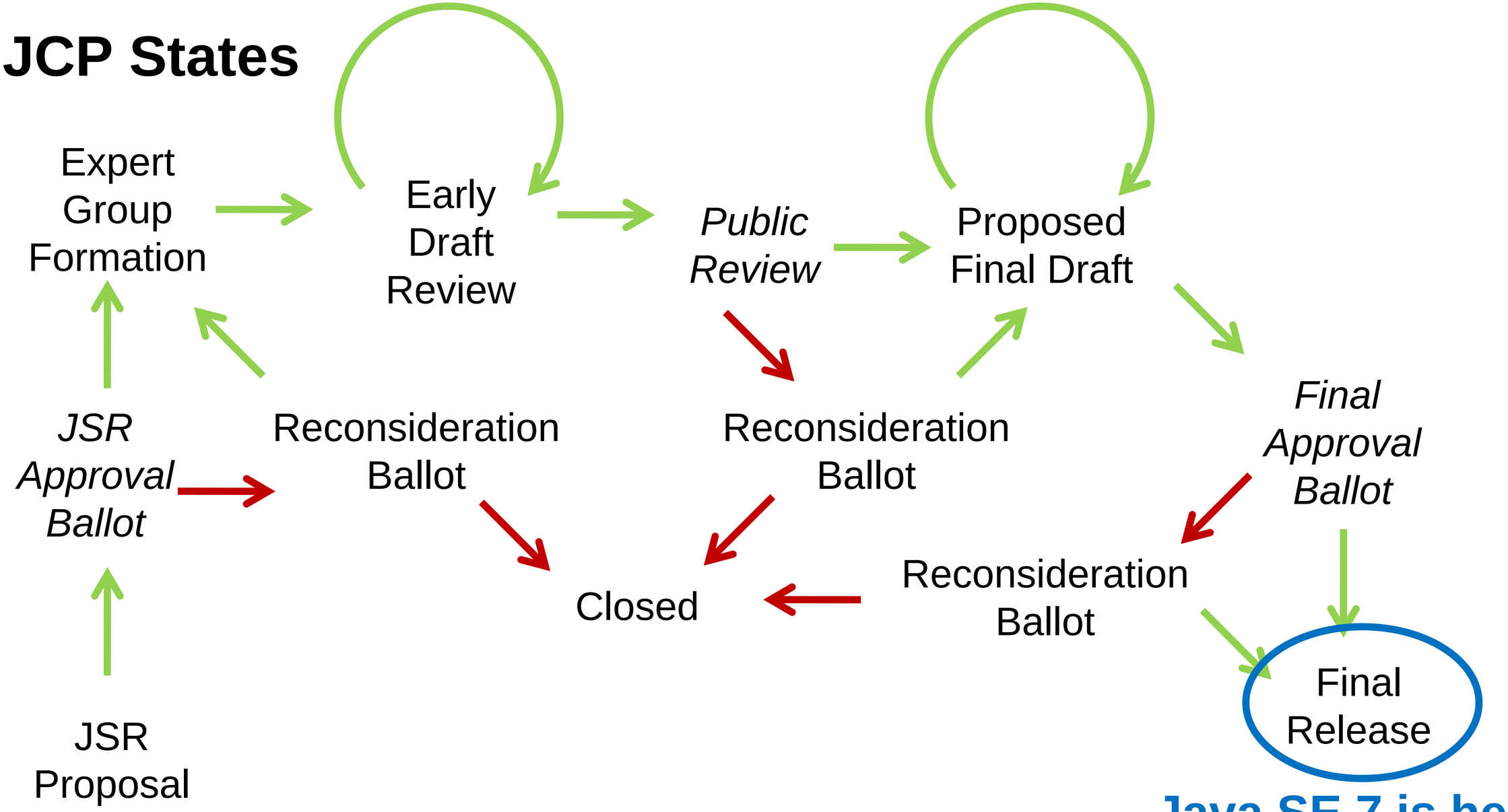


What is the JCP? The Java Community Process

JCP Triad for the Java SE 7 Umbrella JSR 336

Specification	Reference Implementation (RI)	Technology Compatibility Kit (TCK)
Java Language Specification Java Virtual Machine Spec. **`java.*`** **`javax.*`** …	Build of OpenJDK 7 on Linux and windows http://jdk7.java.net/java-se-7-ri/	JCK 7

JCP States



Java SE 7 is here!

Java SE 7 JSRs

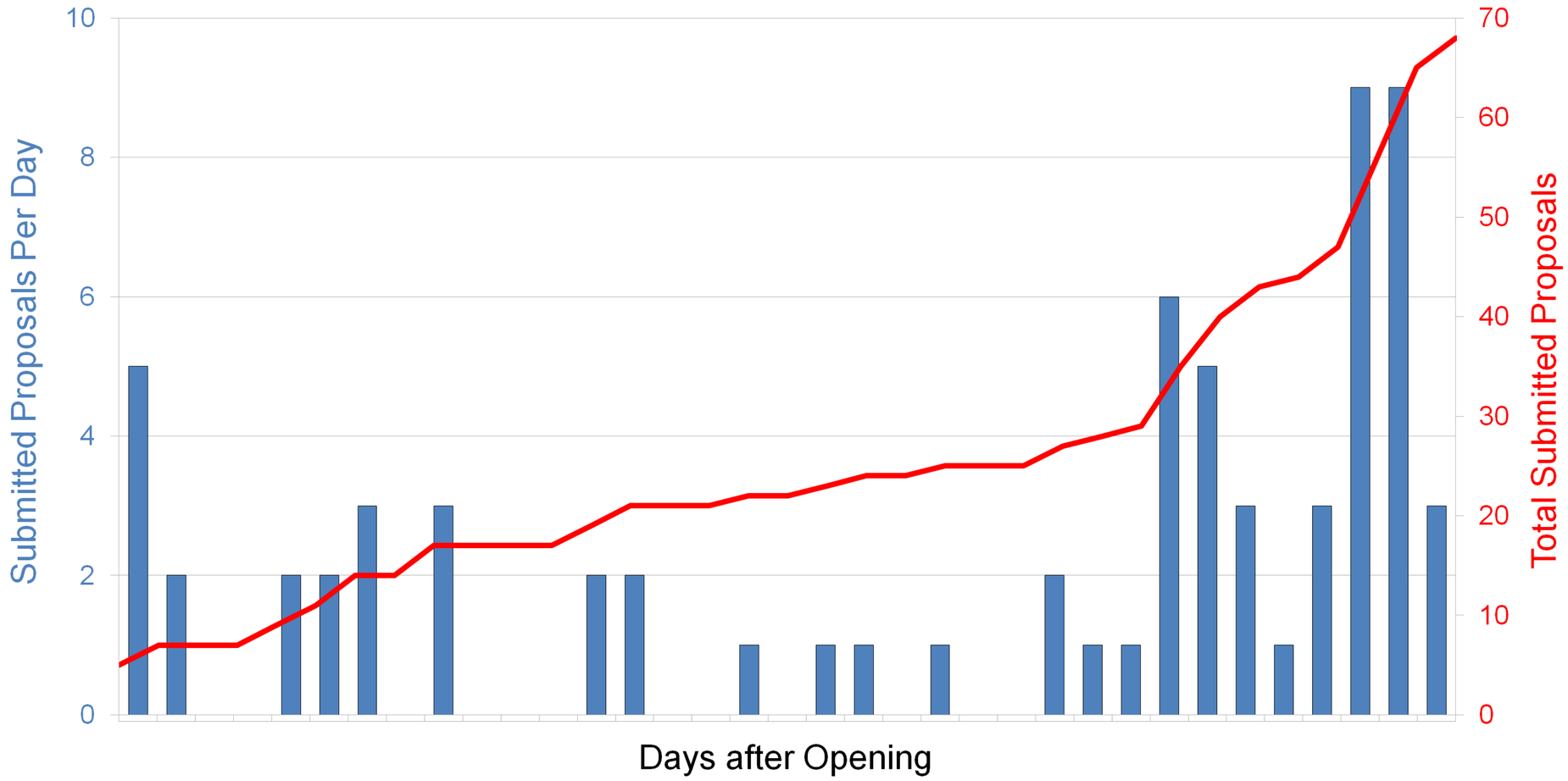
- JSR 336: Java™ SE 7 Release Contents
  - JSR 203: More New I/O APIs for the Java™ Platform ("NIO.2")
  - JSR 292: Supporting Dynamically Typed Languages on the Java™ Platform (**`invokedynamic`**)
  - JSR 334: Small Enhancements to the Java™ Programming Language (Project Coin)
- Maintenance
  - JSR 114: JDBC Rowset Implementations
  - JSR 269: Pluggable Annotation Processing API
  - JSR 901: Java Language Specification
  - JSR 924: JVM Specification
  - …

OpenJDK and JCP

- JDK 7
  http://openjdk.java.net/projects/jdk7/
  - Da Vinci Machine Project
    mlvm-dev@openjdkjava.net
  - Project Coin
    coin-dev@openjdk.java.net
- JDK 8
  http://openjdk.java.net/projects/jdk8/
  - Project Lambda
    lambda-dev@openjdk.java.net
  - Project Jigsaw
    jigsaw-dev@openjdk.java.net

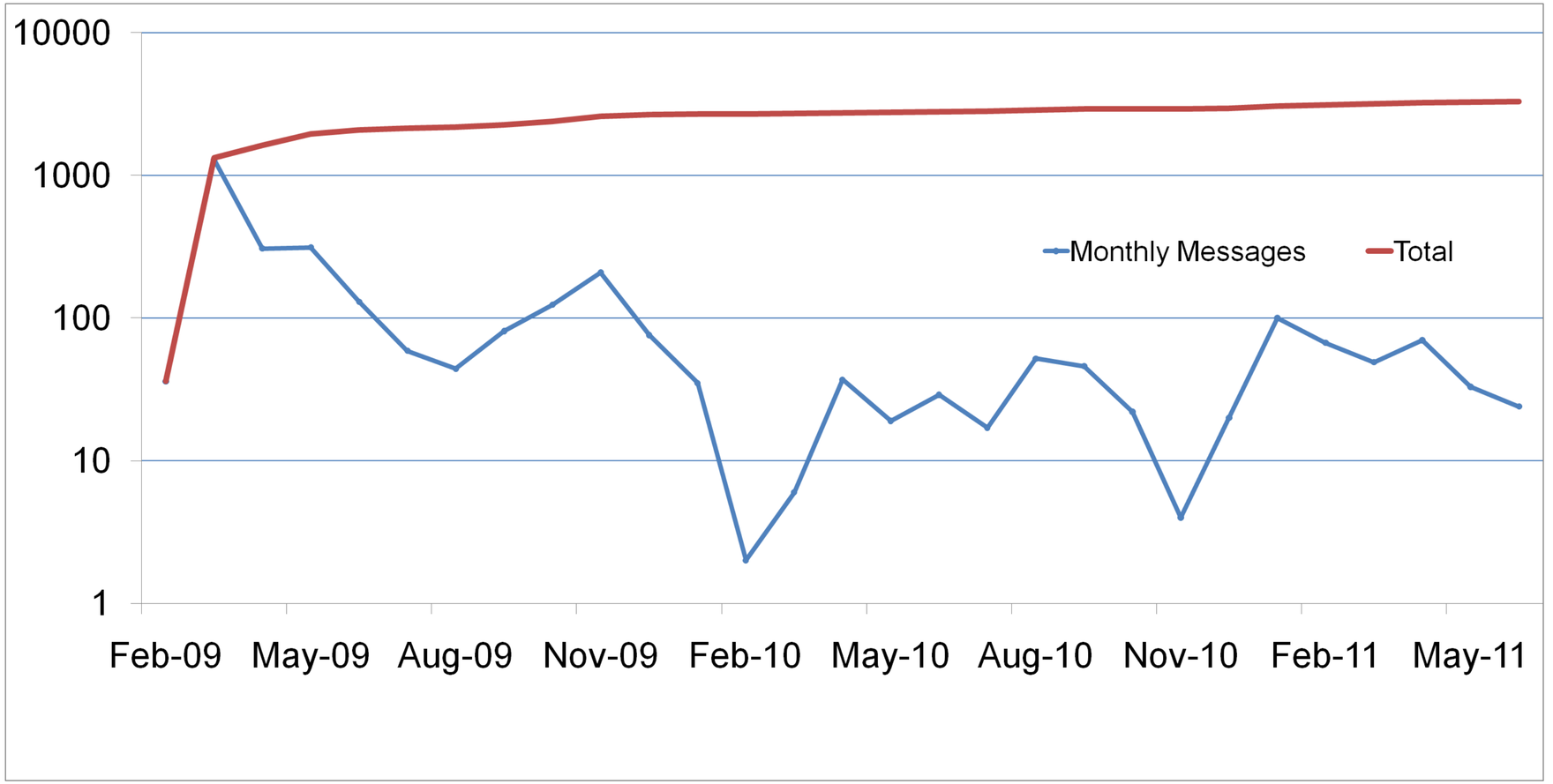
- JSR 336: Java™ SE 7 Release Contents
  - JSR 292: Supporting Dynamically Typed Languages on the Java™ Platform
  - JSR 334: Small Enhancements to the Java™ Programming Language
- JSR 337: Java™ SE 8 Release Contents
  - JSR 335: Lambda Expressions for the Java™ Programming Language
  - JSR *TBD*

Project Coin Proposals



`coin-dev` traffic

Note: log scale on y-axis.

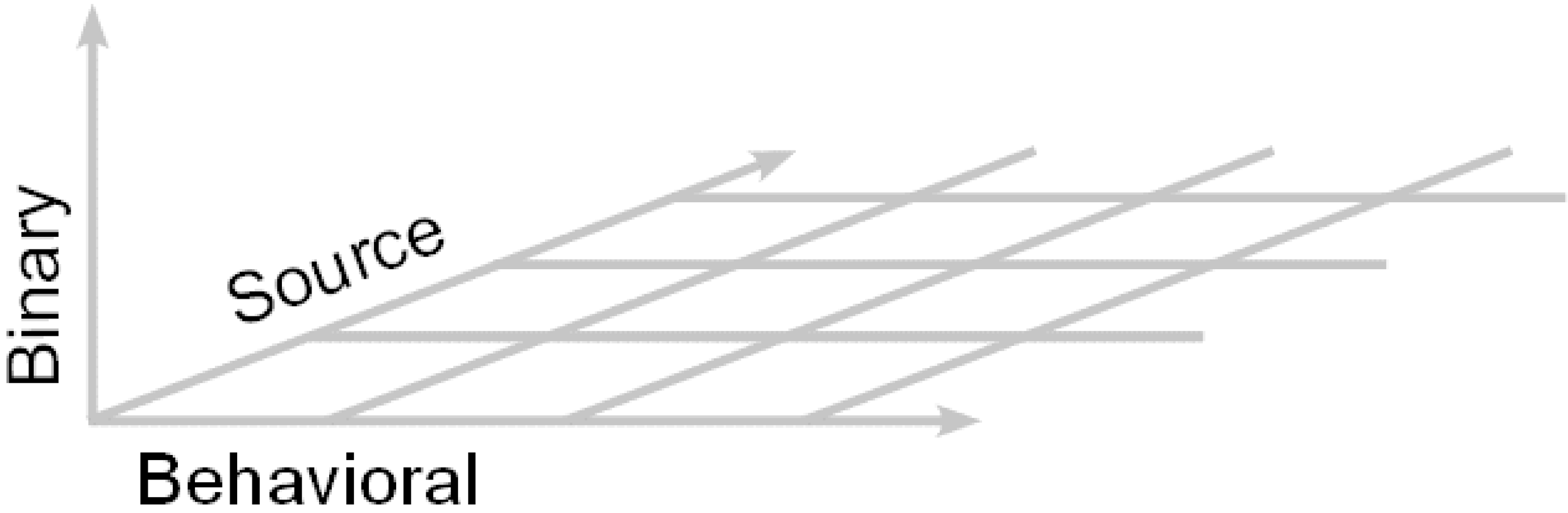


3300+

The JCP at OSCON

- *Java Standards Annoyances*
  Ben Evans (London Java Community) and
  Martijn Verburg (London Java Community), moderators
  3:30 *today*
- *The JCP and the Future of Java*
  Patrick Curran (Java Community Process) and
  Bruno Souza (SouJava)
  2:20pm Tuesday

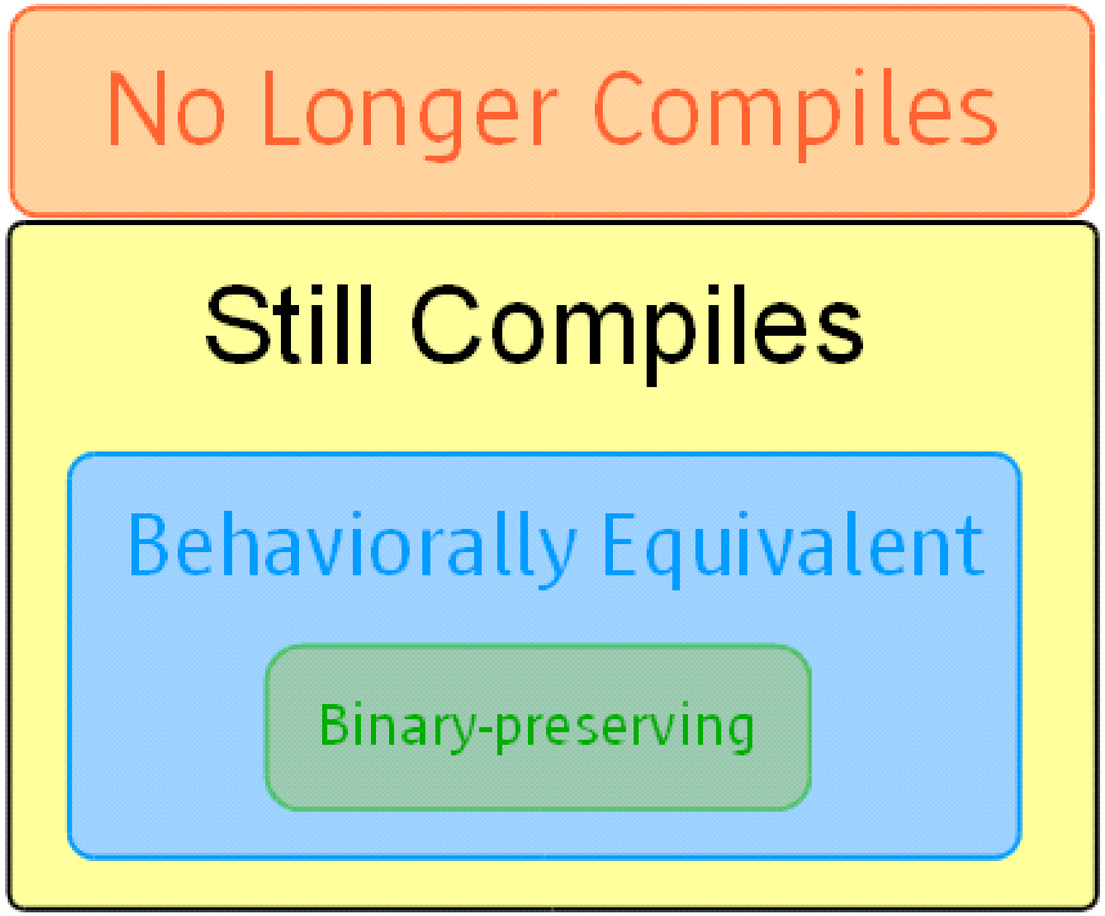
Different kinds of compatibility



JDK Compatible Evolution Policies

- Don't break binary compatibility (as defined in the Java Language Specification).
- Avoid introducing source incompatibilities.
- Manage behavioral compatibility changes.

Source compatibility threat levels



Source compatibility example

```
class PoorlyNamed {
   String[] enum = {"a", "b", "c"};
    boolean assert = false;
}
```

- Stops compiling as of JDK 1.4.x

Another source compatibility example

- Added
  `public BigDecimal(long val)`
- When there was already
  `public BigDecimal(long val)`
- Changes the meaning of **`BigDeicmal(Long.MAX_VALUE)`**
  - A benign change that returns a better answer

Behavioral compatibility example

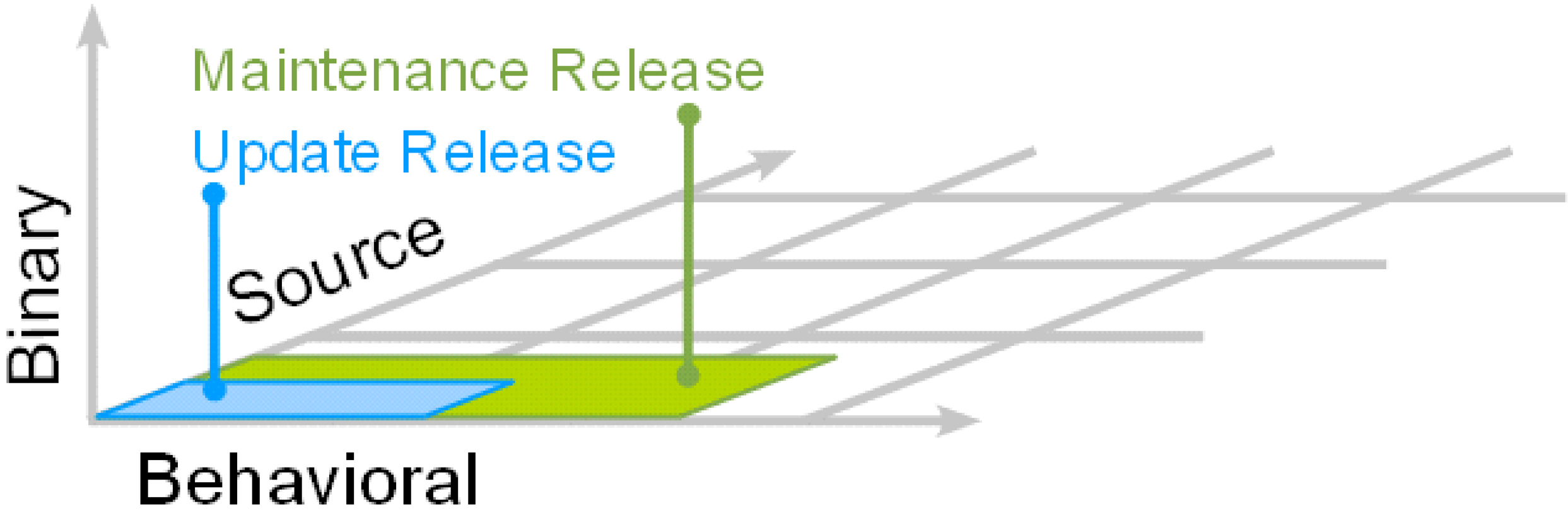
- Expanded values that could be represented, expanded **`toString`** output to include exponents in scientific notation
- Strong input / output properties in a given Java release and across Java releases
- But, databases didn’t recognize the new output format!
- Add a **`toPlainString`** method to provide the old behavior when needed

Small behavioral compatibility

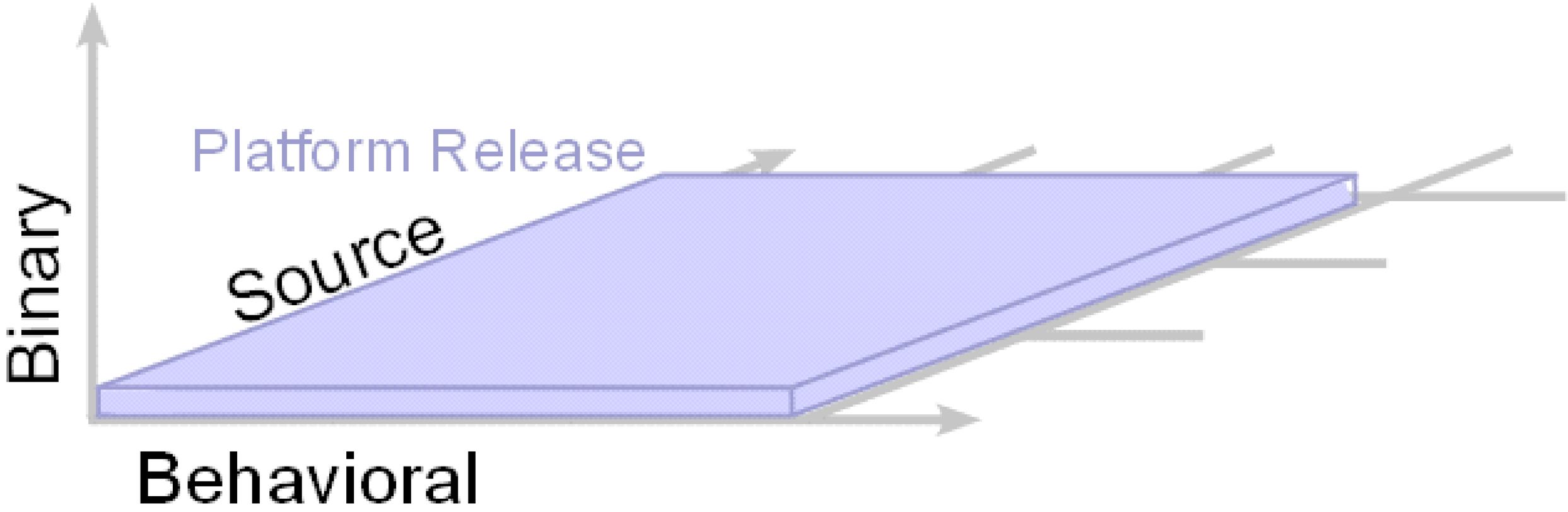
`HashSet.iterator()`:

“Returns an iterator over the elements in this set.
The elements are returned in no particular order.”

Policies vary by category of release



JDK 7 as a platform release



Back to JDK 7

coin, *n*. A piece of small change
coin, *v*. To create new language

“Project Coin in JDK 7 is a suite of small language and library changes to make things programmers do everyday easier.”

Coin Constraints

- *Small* language changes
  - Specification
  - Implementation
  - Testing
- Coordinate with larger language changes, past and future, such as Project Lambda
- Complex language interactions; need to be wary in
  - Specification
  - Implementation

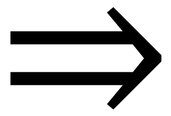
Coin details

- Easier to use generics
  - Diamond
  - Varargs warnings
- More concise error handling
  - Multi-catch
  - `try`-with-resources
- Consistency and clarity
  - Strings in switch
  - Literal improvements

`try`-with-resources desugaring

- **`try(Resource r)`**
  `Block`
- Resource is a type with a **`close`** method. The **`try`**-with-resources statement guarantees **`close`** ose is called when the block exits.

```
try ResourceSpecification
  Block
```



```
{
final VariableModifiers_minus_final R #resource = Expression;
  Throwable #primaryException = null;

  try ResourceSpecification_tail
    Block
  catch (Throwable #t) {
    #primaryException = t;
    throw #t;
  } finally {
    if (#resource != null) {
      if (#primaryException != null) {
        try {
          #resource.close();
        } catch(Throwable #suppressedException) {
          #primaryException.addSuppressed(#suppressedException);
        }
      } else {
        #resource.close();
      }
    }
  }
}
```

Varargs warnings

- Summary: no longer receive uninformative unchecked compiler warnings from calling platform library methods:
  - `<T> List<T> Arrays.asList(T... a)`
  - `<T> boolean Collections.addAll(Collection<? super T> c, T... elements)`
  - `<E extends Enum<E>> EnumSet<E> EnumSet.of(E first, E... rest)`
  - `void javax.swing.SwingWorker.publish(V... chunks)`
  - New `java.lang.SafeVarargs` annotation type to make this assertion

Easy to use, make the compiler do the work!

- Type inference in diamond
- Internal compiler *desugaring*
  - multi-catch
  - strings in switch
  - **`try`**-with-resources

A systematic update

- Automated code conversion for “coinification”
- Ran *annotation processors* over the JDK
  - Types to be retrofitted as **`Closeable`**/**`AutoCloseable`**:
    “Project Coin: Bringing it to a Close(able),”
    http://blogs.sun.com/darcy/entry/project_coin_bring_close/
  - Methods and constructors to be annotated with **`@SafeVarargs`**
    “Project Coin: Safe Varargs in JDK Libraries,”
    http://blogs.sun.com/darcy/entry/project_coin_safe_vararg_libraries/
- *Quantitative* language design with an *analytical* approach

Language design in the real world: diamond

- Two inference algorithms were considered for diamond
  - Differed in how constraints were gathered
- Sometimes the 1st algorithm was more useful, *but*
  other times the 2nd algorithm was more useful
- What to do?
  - Is either one any good?
  - How to choose between them?
- Look at relative performance on a body of code
- Both algorithms were equally effective
  - Type arguments eliminated in 90% of constructor calls
  - A slightly different 90% for each algorithm
- Choose the algorithm with better future evolution properties

The importance of source compatibility: more precise rethrow

```
try {
  throw new DaughterOfFoo();
} catch (Foo e) {
  try {
    throw e; // Used to be treated as throwing Foo,
             // would now throw DaughterOfFoo
  } catch (SonOfFoo anotherException) {
    ; // Reachable according to the compiler?
  }
}
```

Should the feature be explicitly requested?

```
try {
  throw new DaughterOfFoo();
} catch (final Foo e) {
  try {
    throw e; // Treating as
             // throwing DaughterOfFoo
  } catch (SonOfFoo anotherException) {
    ; // Not Reachable; delete dead catch block.
  }
}
```

No; problem does not occur in practice

```
try {
  throw new DaughterOfFoo();
} catch (Foo e) {
  try {
    throw e; // Treating as
             // throwing DaughterOfFoo
  } catch (SonOfFoo anotherException) {
    ; // Not Reachable; delete dead catch block.
  }
}
```

NIO.2

- Need better facilities than **`java.io.File`**!
  - **`java.nio.file.Path`**
    - Immutable
    - Create from path String or URI or java.io.File
- **`FileSystem`**
  - Provides a handle to a file system
  - The factory for objects that access the file system
  - **`FileSystems.getDefault()`** returns the default **`FileSystem`**, represents the local/platform file system
- **`FileStore`**: the underlying storage (volume, concrete file system ...)
- Utility methods to operations on files, recurse directories, etc.

`invokedynamic`

- Controllable linkage greatly simplifies implementation of other language runtimes
- Get rid of many levels of indirection and layers of overhead
- A variety of fittings and adapters around **`MethodHandles`**
- Not expected to be used by many programmers directly.

Fork/join

- With hardware trends toward many-core, need support for fine-grained, CPU intensive parallelism!
- Programming model of recursive decomposition:

```
Result solve(Problem p) {
  if (size < sequential_threshold())
    p.solveSerially();
else {
    Result first, second;
    invoke_in_parallel(first = p.solve(p.firstHalf()),
                       second = p.solve(p.secondHalf()));
    return combine(first, second);
}
```

- See **`java.util.concurrent.ForkJoinTask<V>`**
- Performance a function of sequential threshold, but not overly sensitive, profile for fine tuning
  - Code independent of execution topology
  - Use the right tool for the job, supports compute-intensive tasks; other solutions for other situations

Onward to JDK 8!

Project Lambda — http://openjdk.java.net/projects/lambda/

- A *set* of in-progress language changes:*
  - Lambda expressions (closures)

```
{ String x -> x.length() == 0 }
```

  - SAM conversion (Single Abstract Method)

```
Predicate<String> p = { String x -> x.length() == 0 }
```

  - More type inference, e.g. lambda formals

```
Predicate<String> p = { x -> x.length() == 0 }
```

  - Method references

```
Predicate<> p = String->isEmpty
```

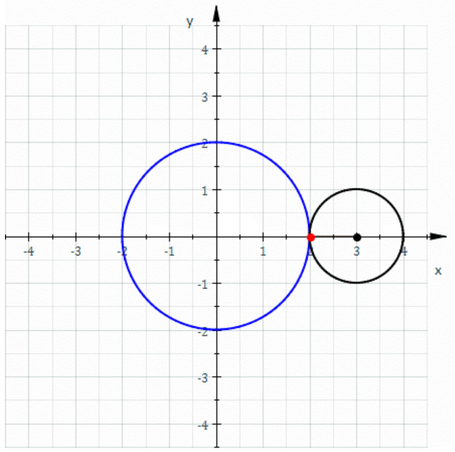
  - Exception transparency (maybe)
  - Virtual extension methods (aka *defender methods*)

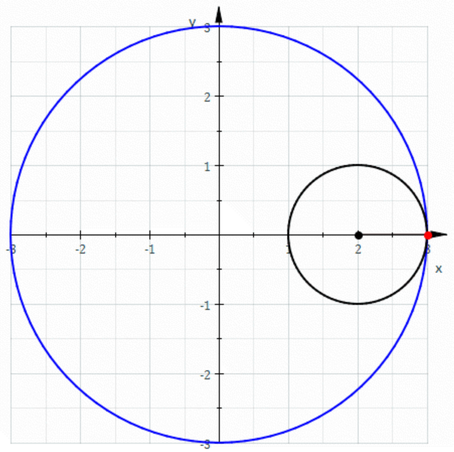
** Syntax subject to change!*

*

Why?

- Provide libraries a path to multicore
  - Internal iteration needed to make data structures parallel-friendly
  - Migrate away from the fundamentally serial for loop





- Empower library developers
  - Easier to evolve the programming model through libraries than through language
  - Enable developers to evolve interface-based APIs over time

Images courtesy http://en.wikipedia.org/wiki/Nephroid, http://en.wikipedia.org/wiki/Deltoid_curve

Why extension methods?

- Adding closures is a *big* language change
- If closures present from the beginning, APIs would look very different
  - So adding closures now makes our APIs show their age!
  - Most important APIs (Collections) are based on interfaces
  - Can’t add methods to interfaces without breaking *source* compatibility
- Adding closures, but not upgrading the APIs to use them effectively, would be silly
  - What no, **`collection.forEach(lambda)`**?
- Therefore, need a mechanism for *interface evolution* to allow existing Collections types (as well as new ones) to benefit from closures.

Virtual extension methods

- Virtual extension methods specified in the interface

```
interface Collection<T> {
    // existing methods, plus
    void forEach(Block<T> block)
        default Collections.<T>forEach;
}
```

- The **`forEach`** method is an *extension method*
  - From caller's perspective, an ordinary virtual method
- **`Collection`** *provides* a *default implementation*
  - Default is only used when implementation classes do not provide a body for the extension method
  - Lots of details to get right!
    - Work in progress to ensure source and binary compatibility when adding extension methods (solutions known when there is global consistency)
    - There exist bad interactions with separate compilation, etc.
  - More strongly avoid binary as opposed to source incompatibilities

Project Jigsaw — http://openjdk.java.net/projects/jigsaw/

- Bringing a standard module system to the Java SE platform!
- Replace “jar hell” with dependencies recorded in source code
- Package modules for download and install, integrate with native packages managers where present
- New optimization points such as module install time
- Provides another mechanism to control the compatible evolution of the platform
- Will be used on the JDK itself

Q & A

http://blogs.oracle.com/darcy

@jddarcy