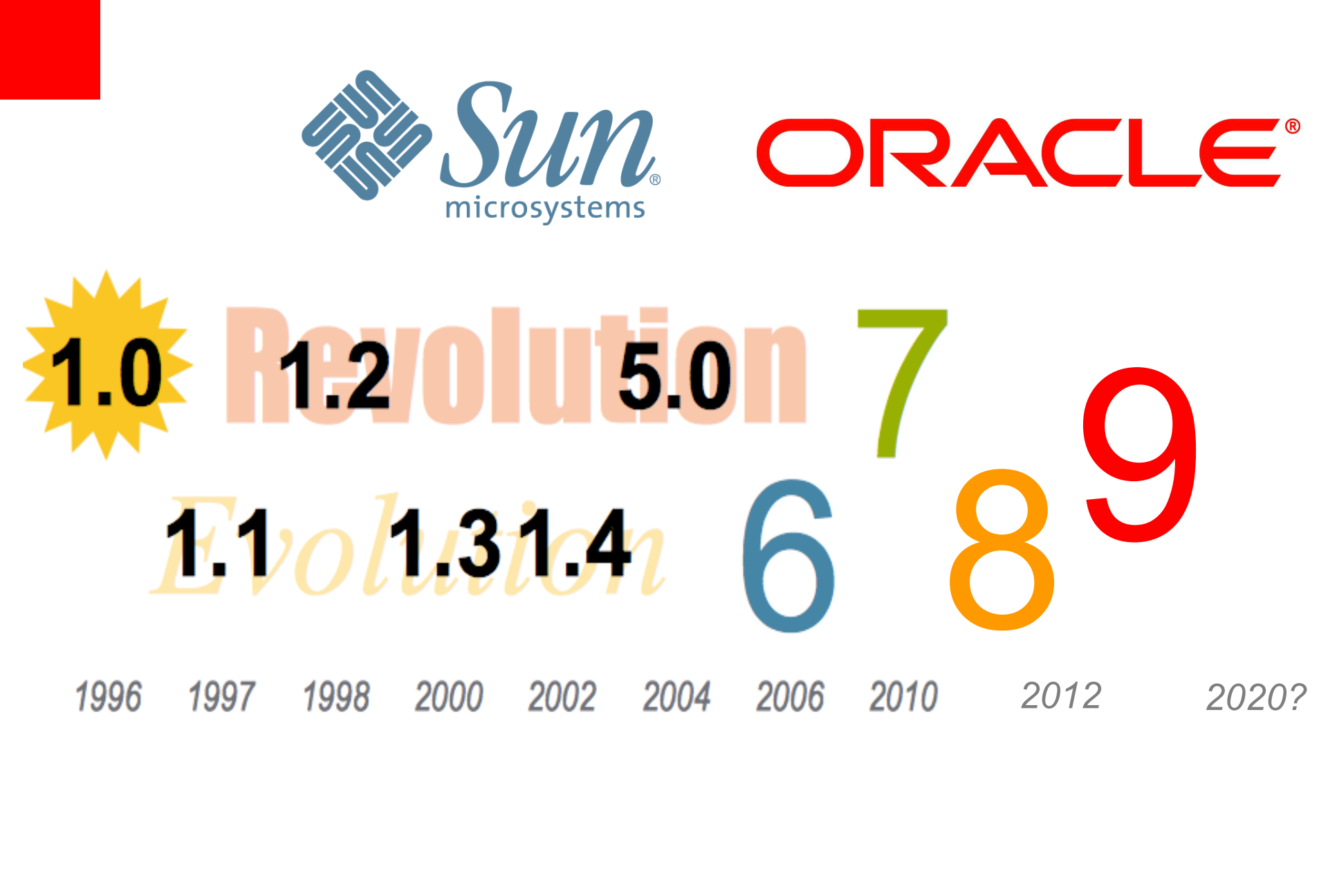
ORACLE®

ORACLE®

To Java SE 8, and Beyond!

Simon Ritter
Technology Evangelist

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.


Sun microsystems
ORACLE
1.0
Revolution
1.2
5.0
7
9
1.1
1.3
1.4
Evolution
6
8
1996
1997
1998
2000
2002
2004
2006
2010
2012
2020?

Priorities for the Java Platforms

- Grow Developer Base
- Grow Adoption
- Increase Competitiveness
- Adapt to change

Evolving the Language

From "Evolving the Java Language" - JavaOne 2005

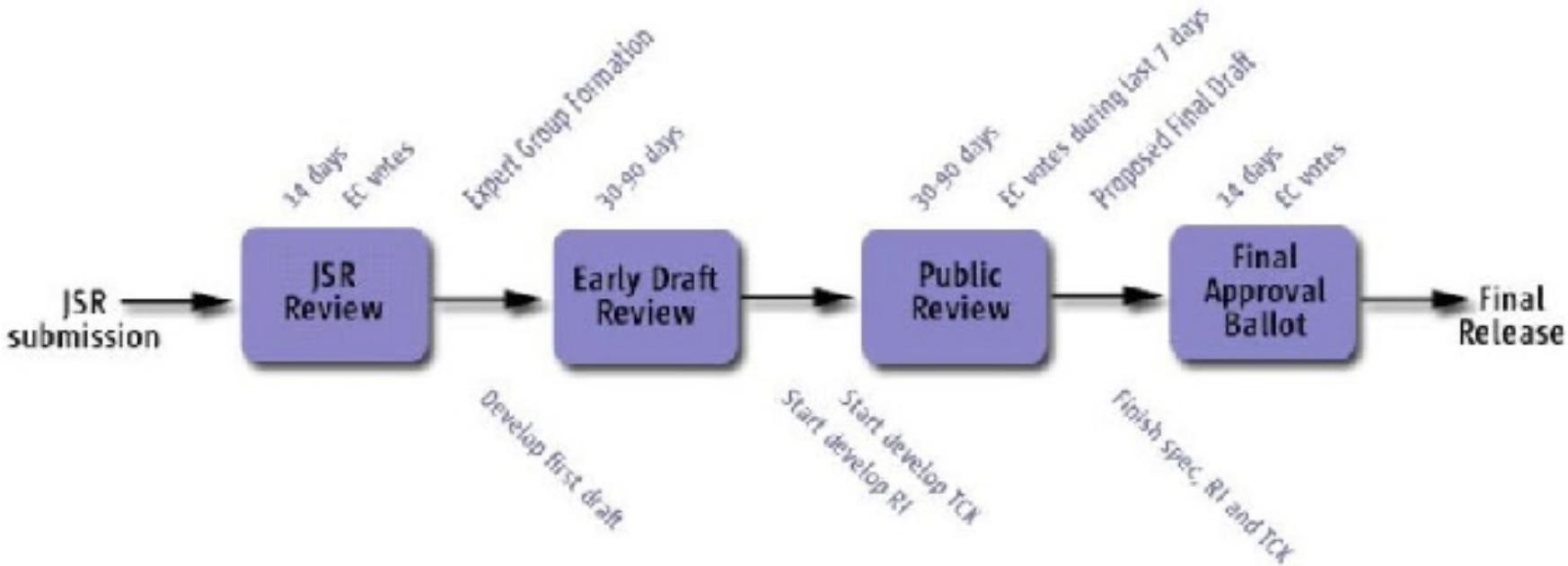
- Java language principles
  - Reading is more important than writing
  - Code should be a joy to read
  - The language should not hide what is happening
  - Code should do what it seems to do
  - Simplicity matters
  - Every "good" feature adds more "bad" weight
  - Sometimes it is best to leave things out
- One language: with the same meaning everywhere
  - No dialects
- We will evolve the Java language
  - But cautiously, with a long term view
  - "first do no harm"

also "Growing a Language" - Guy Steele 1999
"The Feel of Java" - James Gosling 1997

How Java Evolves and Adapts

Of the community, by the community, for the community

Java Community Process



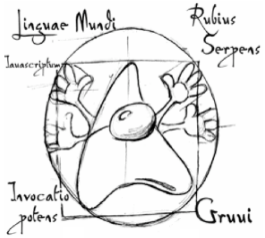
JSR-348: JCP.next

JCP Reforms

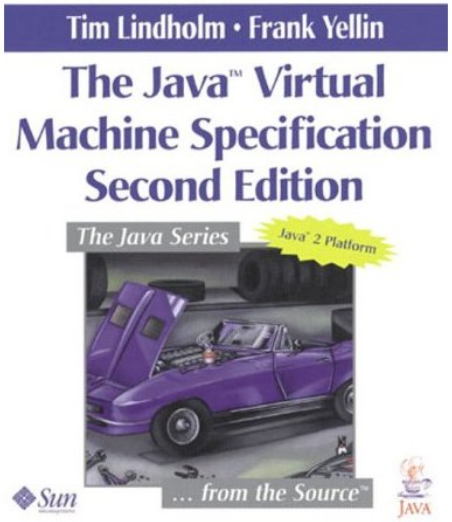
- Developers' voice in the Executive Committee
  - SOUJava
  - Goldman Sachs
  - London JavaCommunity
- JCP starting a program of reform
  - JSR 348: Towards a new version of the JCP

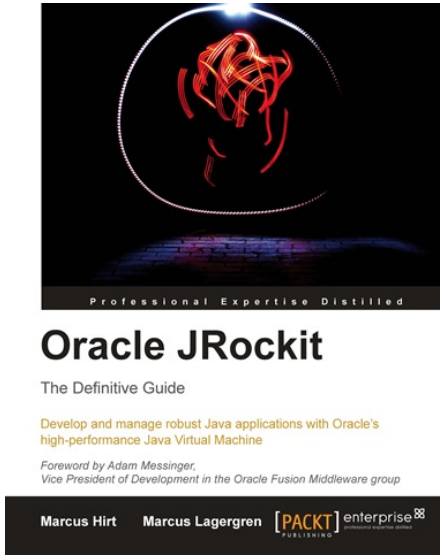
Java SE 7 Release Contents

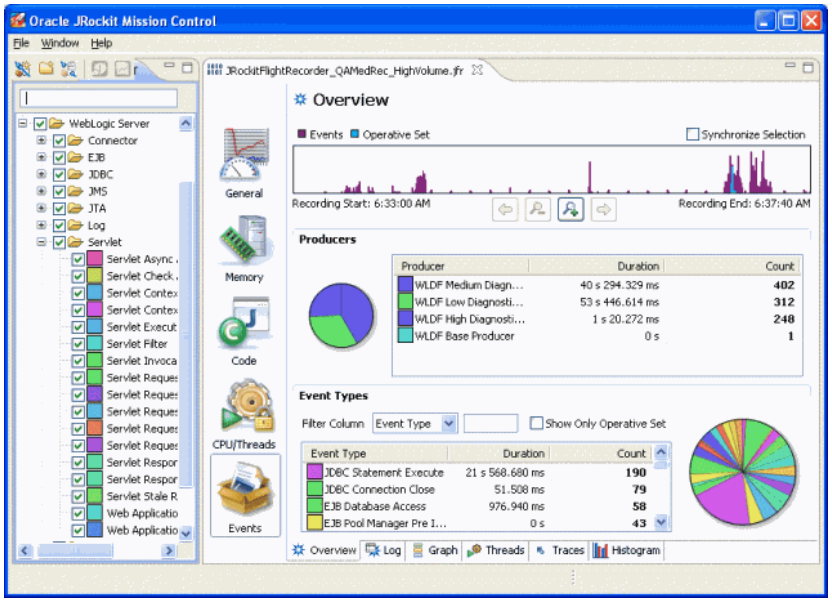
- Java Language
  - Project Coin (JSR-334)
- Class Libraries
  - NIO2 (JSR-203)
  - Fork-Join framework, ParallelArray (JSR-166y)
- Java Virtual Machine
  - The DaVinci Machine project (JSR-292)
  - InvokeDynamic bytecode
- Miscellaneous things
- JSR-336: Java SE 7 Release Contents



JVM Convergence







The (Performance) Free Lunch Is Over

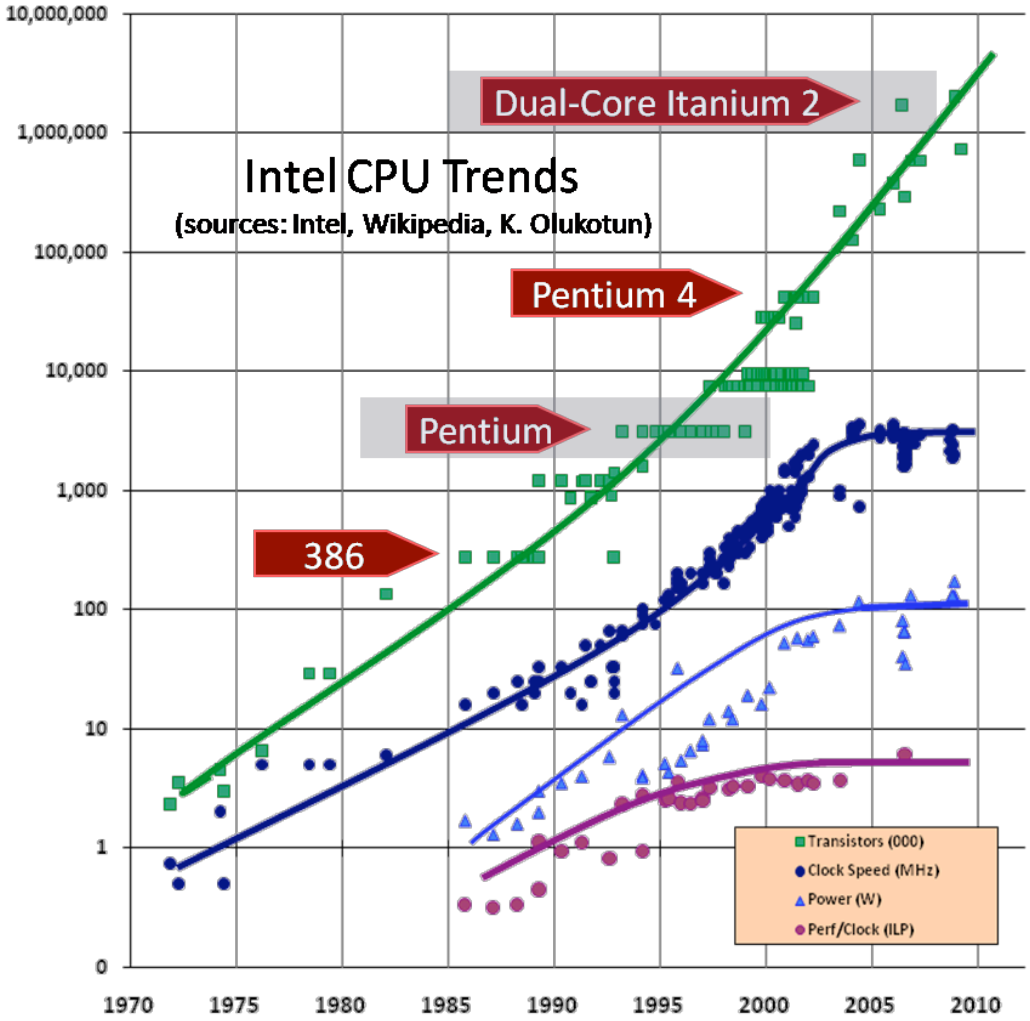
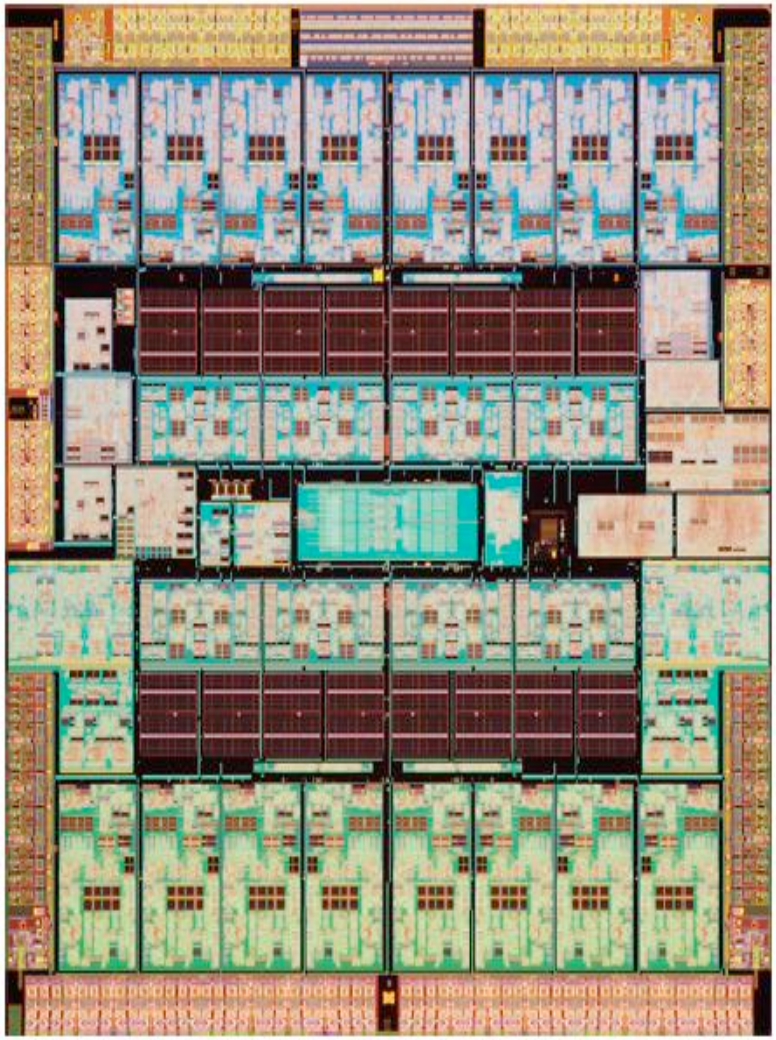


Image courtesy of Herb Sutter



SPARC T1 (2005)

8 x 4 = **32**

SPARC T2 (2007)

8 x 8 = **64**

SPARC T3 (2011)

16 x 8 = **128**

Multi-core Clients

Phone ... Tablet ... Desktop

2 ... 4 ..... 8 .....

2002 2004 2006 2008 2010 2012

Big Disclaimer

The syntax used in the following slides may change

Caveat emptor

```
class Student {
   String name;
   int gradYear;
   double score;
}

Collection<Student> students = ...;
```

```
Collection<Student> students = ...;

double max = Double.MIN_VALUE;

for (Student s : students) {
    if (s.gradYear == 2011)
        max = Math.max(max, s.score);
}
```

```
Collection<Student> students = ...;

double max = Double.MIN_VALUE;

for (Student s : students) {
    if (s.gradYear == 2011)
        max = Math.max(max, s.score);
}
```

```
Collection<Student> students = ...;

max = students.filter(new Predicate<Student>() {
                 public boolean op(Student s) {
                     return s.gradYear == 2011;
                 }
             }).map(new Extractor<Student, Double>() {
                 public Double extract(Student s) {
                     return s.score;
                 }
             }).reduce(0.0, new Reducer<Double, Double>() {
                 public Double reduce(Double max, Double score) {
                   return Math.max(max, score);
                 }
             });
```

Inner Classes Are Imperfect Closures

- Bulky syntax
- Unable to capture non-final local variables
- Transparency issues
  - Meaning of return, break, continue, this
- No non-local control flow operators

Single Abstract Method (SAM) Types

- Lots of examples in the Java APIs
  - **`Runnable, Callable, EventHandler, Comparator`**

```
foo.doSomething(new CallbackHandler() {
  public void callback(Context c) {
    System.out.println(c.v());
  }
});
```

- Noise:Work ratio is 5:1
- Lambda expressions grow out of the idea of making callback objects easier

```
Collection<Student> students = ...;

max = students.filter((Student s) -> s.gradYear == 2011)
              .map((Student s) -> s.score)
              .reduce(0.0,
                (Double max, Double score) ->
                   Math.max(max, score));

max = students.filter(s -> s.gradYear == 2011)
              .map(s -> s.score)
              .reduce(0.0, Math#max);

max = students.parallel()
              .filter(s -> s.gradYear == 2011)
              .map(s -> s.score)
              .reduce(0.0, Math#max);
```

Lambda Expression Examples

```
(Context c) -> System.out.println(c.v());

c -> System.out.println(c.v());  // Inferred

int x -> x + 1
```

Target Types

- Rule #1: Only in a context where it can be converted to a SAM type

```
CallBackHandler cb =
    (Context c) -> System.out.println(c.v());

x -> x + 1;

Runnable r = () -> System.out.println("Running”);
executor.submit(() -> System.out.println("Running”));

Object o = () -> 42;   // Illegal, not a SAM type
```

Lambda Bodies

- Rule #2: A list of statements just like in a method body, except no break or continue at the top level. The return type is inferred from the unification of the returns from the set of return statements
- Rule #3: ‘this’ has the same value as ‘this’ immediately outside the Lambda expression
- Rule #4: Lambdas can use ‘effectively final’ variables as well as final variables (compiler inferred)

```
Collection<Student> students = ...;

double max =          // Lambda expressions
  students.filter(Students s -> s.gradYear == 2010})
          .map(Students s -> s.score })
          .reduce(0.0, Math#max);


interface Collection<T> {
   int add(T t);
   int size();
   void clear();
   ...
}
```

How to extend an interface in Java SE 8

tells us this method extends the interface

```
public interface Set<T> extends Collection<T>
{

    public int size();

  ...    // The rest of the existing Set methods


    public extension T reduce(Reducer<T> r)
      default Collections.<T>setReducer;
}
```

Implementation to use if none exists for the implementing class

```
Collection<Student> students = ...;

double max =          // Lambda expressions
  students.filter(Students s -> s.gradYear == 2010)
          .map(Students s -> s.score )
          . reduce(0.0, Math#max);

interface Collection<T> {  // Default methods
   extension Collection<E> filter(Predicate<T> p)
        default Collections.<T>filter;

   extension <V> Collection<V> map(Extractor<T,V> e)
        default Collections.<T>map;

   extension <V> V reduce()
        default Collections.<V>reduce;
}
```

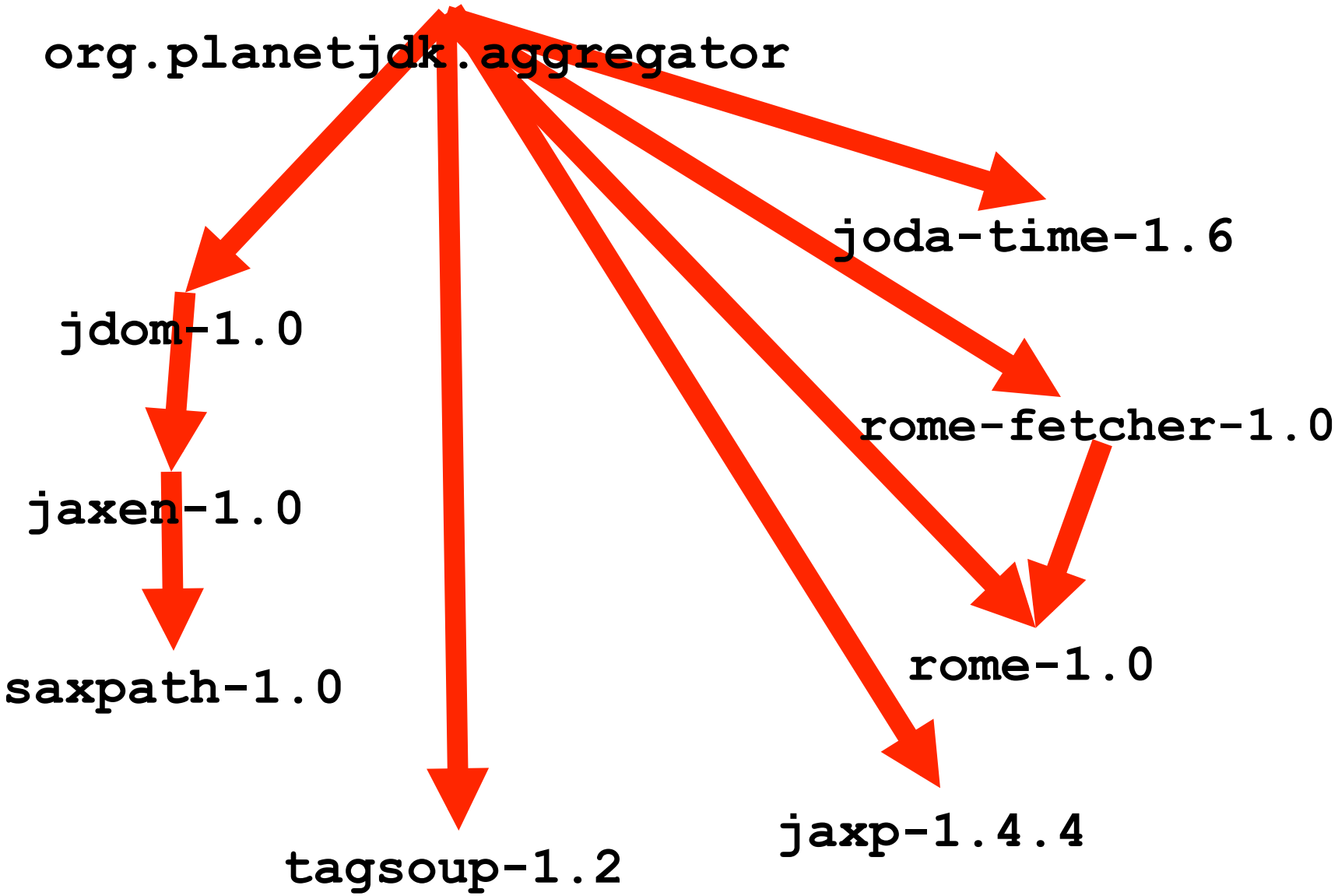
```
$ java org.planetjdk.aggregator.Main
```

```
$ java -cp $APPHOME/lib/jdom-1.0.jar:
$APPHOME/lib/jaxen-1.0.jar:
$APPHOME/lib/saxpath-1.0.jar:
$APPHOME/lib/rome.jar-1.0.jar:
$APPHOME/lib/rome-fetcher-1.0.jar:
$APPHOME/lib/joda-time-1.6.jar:
$APPHOME/lib/tagsoup-1.2.jar:
org.planetjdk.aggregator.Main
```

```
$ java -cp $APPHOME/lib/jdom-1.0.jar:
$APPHOME/lib/jaxen-1.0.jar:
$APPHOME/lib/saxpath-1.0.jar:
$APPHOME/lib/rome.jar-1.0.jar:
$APPHOME/lib/rome-fetcher-1.0.jar:
$APPHOME/lib/joda-time-1.6.jar:
$APPHOME/lib/tagsoup-1.2.jar:
org.planetjdk.aggregator.Main
```

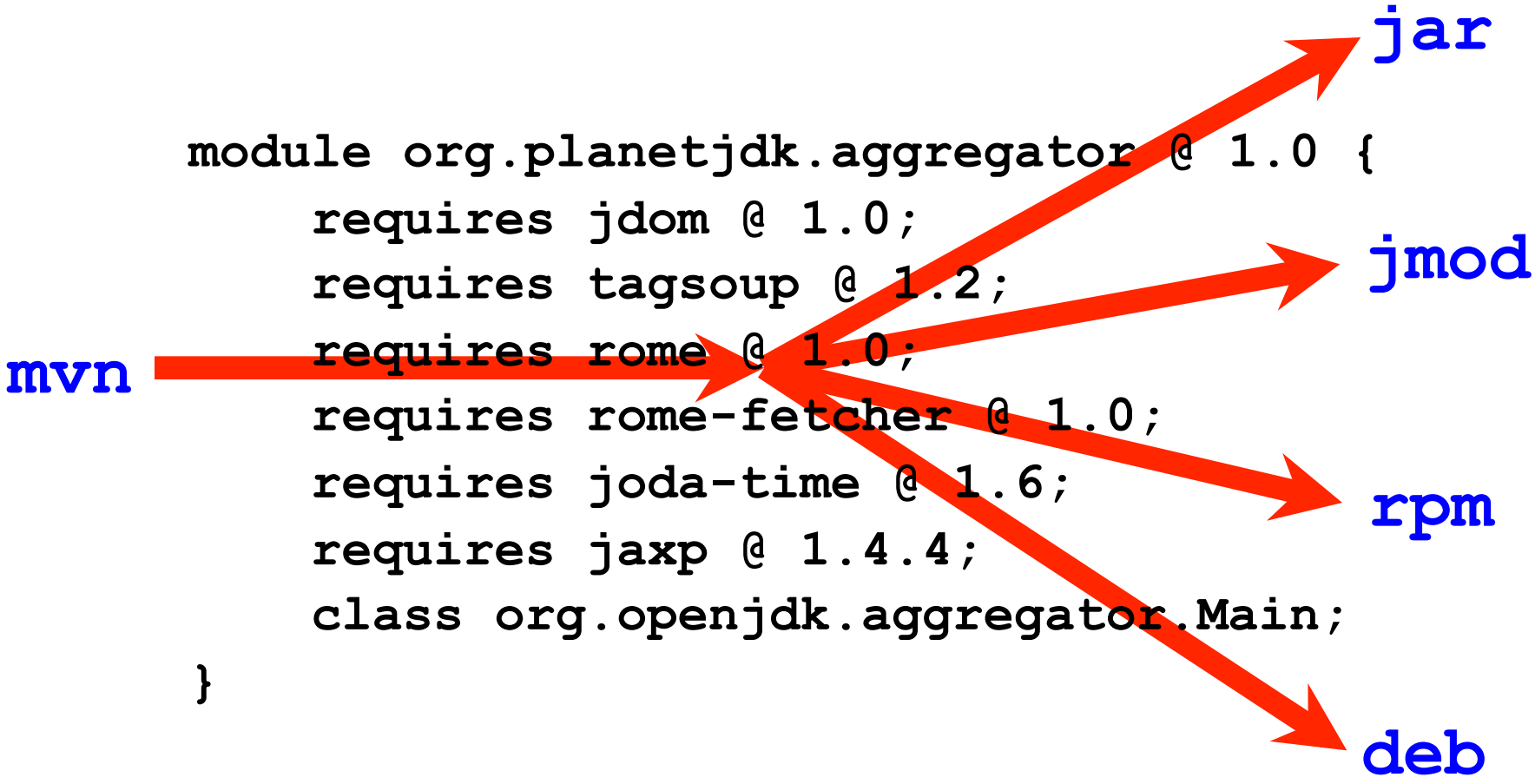
module-info.java

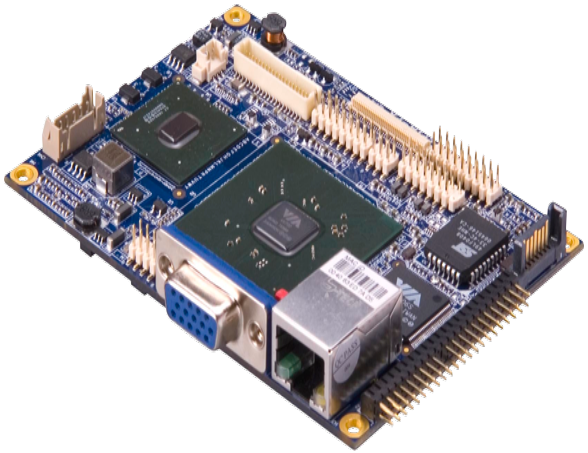
```
module org.planetjdk.aggregator @ 1.0 {
    requires jdom @ 1.0;
    requires tagsoup @ 1.2;
    requires rome @ 1.0;
    requires rome-fetcher @ 1.0;
    requires joda-time @ 1.6;
    requires jaxp @ 1.4.4;
    class org.openjdk.aggregator.Main;
}
```


org.planetjdk.aggregator
joda-time-1.6
jdom-1.0
rome-fetcher-1.0
jaxen-1.0
rome-1.0
saxpath-1.0
jaxp-1.4.4
tagsoup-1.2


classpath

```
module org.planetjdk.aggregator @ 1.0 {
    requires jdom @ 1.0;
    requires tagsoup @ 1.2;
    requires rome @ 1.0;
    requires rome-fetcher @ 1.0;
    requires joda-time @ 1.6;
    requires jaxp @ 1.4.4;
    class org.openjdk.aggregator.Main;
}
```





http://www.flickr.com/photos/thatguyfromcchs08/2300190277
http://www.flickr.com/photos/viagallery/2290654438

Java SE Profiles and Modules

- Rules for creating modules of the Java SE platform
  - Java SE base profile
  - Java SE base module
  - Component modules for separable technologies

JDK 8 – Proposed Content

Theme	Description/Content
Project Jigsaw	• Module system for Java applications and for the Java platform
Project Lambda	• Closures and related features in the Java language (JSR 335) • Bulk parallel operations in Java collections APIs (filter/map/reduce)
Oracle JVM Convergence	• Complete migration of performance and serviceability features from JRockit, including Mission Control and the Flight Recorder
JavaFX 3.0	• Next generation Java client, Multi-touch
JavaScript	• Next-gen JavaScript-on-JVM engine (Project Nashorn) • JavaScript/Java interoperability on JVM
Device Support	• Camera, Location, Compass and Accelerometer
Developer Productivity	• Annotations onTypes (JSR 308), Minor language enhancements
API and Other Updates	• Enhancements to Security, Date/Time (JSR 310), Networking, Internationalization, Accessibility, Packaging/Installation

Additional Disclaimers

- Some ***ideas*** for the Java Platform are shown on the following slides
- Large R&D effort required
- Content and timing highly speculative
- Some things will turn out to be bad ideas
- New ideas will be added
- Still, Java's future is bright (in our humble opinion)!

Java SE 9 (and beyond…)

Interoperability	• Multi-language JVM • Improved Java/Native integration
Cloud	• Multi-tenancy support • Resource management
Ease of Use	• Self-tuning JVM • Language enhancements
Advanced Optimizations	• Unified type system • Data structure optimizations
Works Everywhere and with Everything	• Scale down to embedded, up to massive servers • Support for heterogenuous compute models

Vision: Interoperability

- Improved support for non-Java languages
  - Invokedynamic (done)
  - Java/JavaScript interop (in progress – JDK 8)
  - Meta-object protocol (JDK 9)
  - Long list of JVM optimizations (JDK 9+)
- Java/Native
  - Calls between Java and Native without JNI boilerplate (JDK 9)

Vision: Cloud

- Multi-tenancy (JDK 8+)
  - Improved sharing between JVMs in same OS
  - Per-thread/threadgroup resource tracking/management
- Hypervisor aware JVM (JDK 9+)
  - Co-operative memory page sharing
  - Co-operative lifecycle, migration

Vision: Language Features

- Large data support (JDK 9)
  - Large arrays (64 bit support)
- Unified type system (JDK 10+)
  - No more primitives, make everything objects
- Other type reification (JDK 10+)
  - True generics
  - Function types
- Data structure optimizations (JDK 10+)
  - Structs, multi-dimensional arrays, etc
  - Close last(?) performance gap to low-level languages

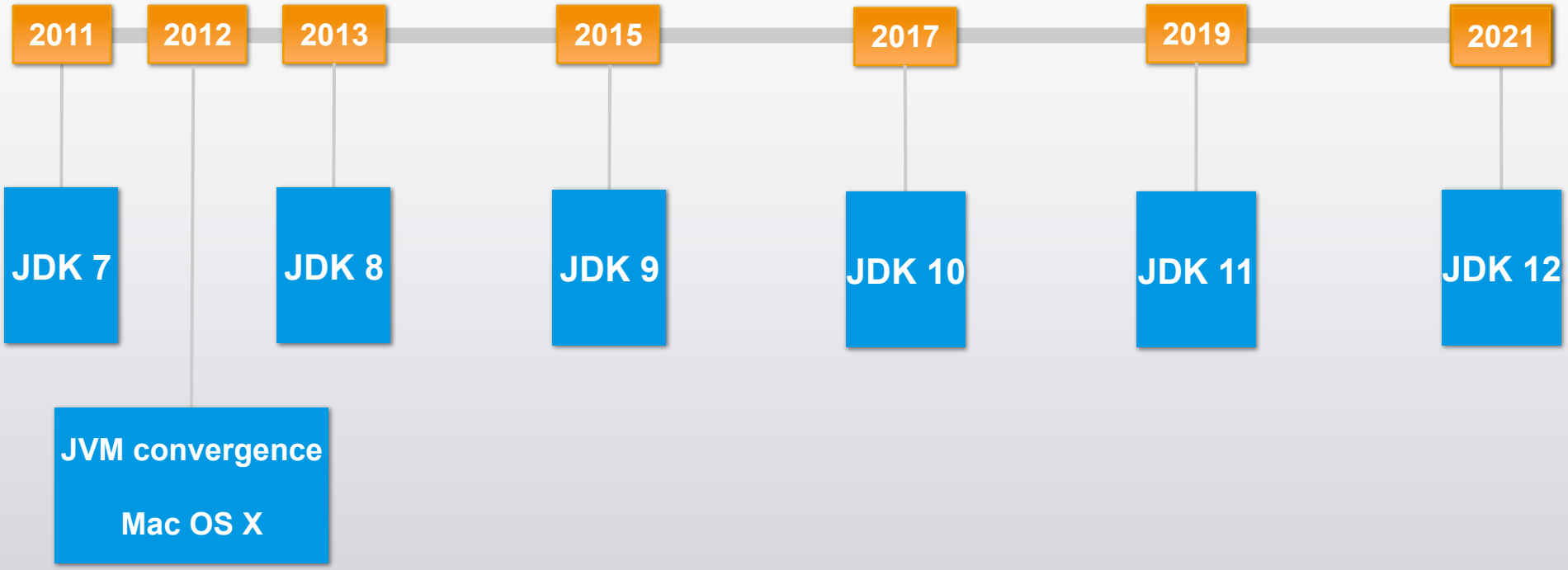
Vision: Integration

- Modern device support (JDK 8+)
  - Multitouch (JDK 8)
  - Location (JDK 8)
  - Sensors – compass, accelerometer, temperature, pressure, ... (JDK 8+)
- Heterogenous compute models (JDK 9+)
  - Java language support for GPU, FPGA, offload engines, remote PL/SQL...

The Path Forward

- Open development
  - Prototyping and R&D in OpenJDK
  - Cooperate with partners, academia, greater community
- Work on next JDK, future features in parallel
- 2-year cycle for Java SE releases

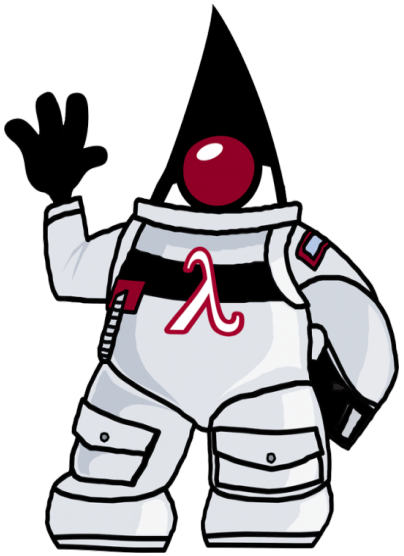
Java SE 2012 to Java 12



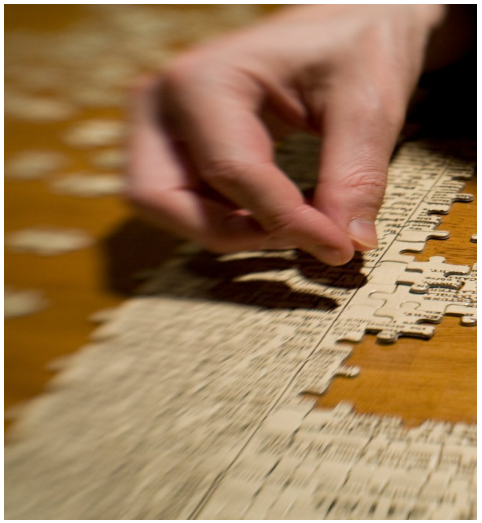
Conclusions

- The Java platform will continue to evolve
- Java SE 8 will add some nice, big features
- Expect to see more in Java SE 9 and beyond
- Java is not the new Cobol

Further Information



- Project Lambda
  - openjdk.java.net/projects/lambda



- Project Jigsaw
  - openjdk.java.net/projects/jigsaw

ORACLE®