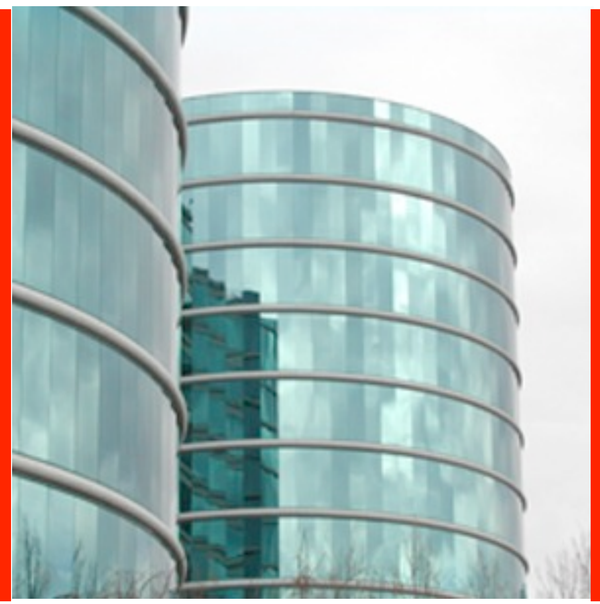


ORACLE®

Java SE: The Road Ahead

Brian Goetz

Java Language Architect

1996

1.1

1996 1997

1.2

1.1

1996 1997 1998

1.2

1.1 1.3

1996 1997 1998 2000

1.0 1.1 1.2 1.3 Revolution

1996 1997 1998 2000

Revolution

Evolution

1.1 1.2 1.3

1996 1997 1998 2000

1.0

Revolution

1.1

1.2

Evolution

1.3

1.4

1996 1997 1998 2000 2002

1.0 1.2 5.0

Revolution

Evolution

1.1 1.3 1.4

1996 1997 1998 2000 2002 2004

Revolution

1.2 5.0

Evolution

1.1 1.3 1.4

6

1996 1997 1998 2000 2002 2004 2006

Sun microsystems

Revolution 7

Evolution 6

1.1 1.2 1.3 1.4 5.0

1996 1997 1998 2000 2002 2004 2006 2010

ORACLE

1.0 1.2 5.0 Revolution 7 ?

1.1 1.3 1.4 Evolution 6

1996 1997 1998 2000 2002 2004 2006 2010

7 & 8 & 9 . . .

7 & 8 & 9 . . .

Productivity

7 & 8 & 9 . . .

Productivity Performance

7 & 8 & 9 . . .

Productivity
Performance
Universality

ORACLE

7 & 8 & 9 . . .

Productivity
Performance
Universality
Modularity

ORACLE

7&8...

Productivity
Performance
Universality
Modularity
Integration

ORACLE

7&8...

Productivity
Performance
Universality
Modularity
Integration
Serviceability

7&9...

Productivity
Performance
Universality
Modularity
Integration
Serviceability

ORACLE

Project Coin (JSR TBD)

openjdk.java.net/projects/coin

http://www.flickr.com/photos/chefranden/908539119/

ORACLE

Project Coin (JSR TBD)

openjdk.java.net/projects/coin

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

http://www.flickr.com/photos/chefranden/908539119/

ORACLE

Project Coin (JSR TBD)

openjdk.java.net/projects/coin

- *Small* language changes that simplify things that programmers do every day
  - Eliminate redundant code
  - Create language mechanisms to replace error-prone idioms
- Small means
  - No type system changes
  - No new keywords
  - Small in implementation, specification, testing
  - Stay away from method resolution rules!

http://www.flickr.com/photos/chefranden/908539119/

ORACLE

Project Coin Features for JDK 7

Project Coin Features for JDK 7

- Diamond
- Try-with-resources
- Improved integral literals
- Strings in switch
- Varargs warnings
- Multi-catch & precise rethrow

```
Map map = new HashMap();
```

```
Map<String,String> map = new HashMap();
```

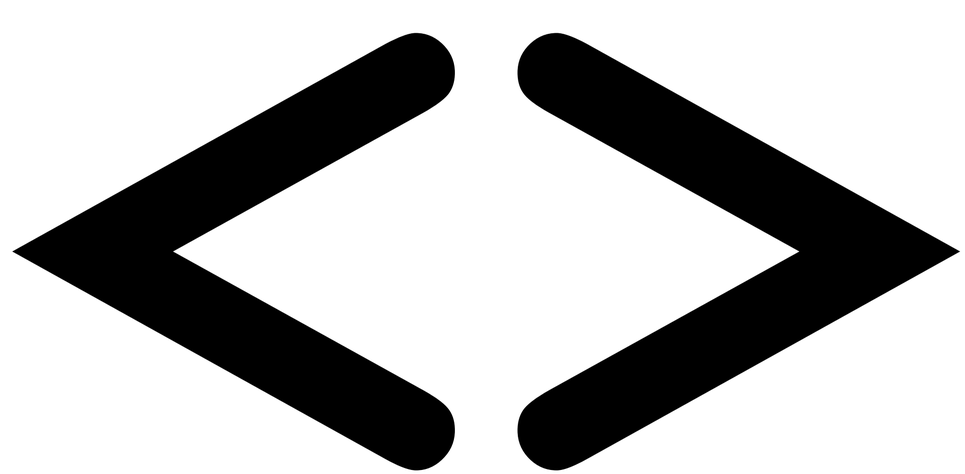
```
Map<String,String> map
    = new HashMap<String,String>();
```

```
Map<String,List<String>> map
    = new HashMap<String,List<String>>();
```

```
Map<String,Map<Integer,String>> map
    = new HashMap<String,Map<Integer,String>>();
```

```
Map<String,Map<Integer,String>> map
    = new HashMap                                    ();
```

```
Map<String,Map<Integer,String>> map
    = new HashMap                                    ();
```



```
Map<String,Map<Integer,String>> map
    = new HashMap<>();
```

Project Coin: Diamond

- Diamond embraces *type inference*
  - No loss of static typing
    - But less “typing” on the part of the programmer
  - Don’t make the programmer say things the compiler can easily deduce
- Type inference is a key productivity theme we will see in JDK 7, 8, and beyond

```
static void copy(File src, File dst)
  throws IOException
{
  InputStream in = new FileInputStream(src);
  OutputStream out = new FileOutputStream(dst);
  try {
    byte[] buf = new byte[BUFSIZ];
    int n;
    while ((n = in.read(buf)) >= 0)
      out.write(buf, 0, n);
  } finally {
    in.close();
    out.close();
  }
}
```

```
static void copy(File src, File dst)
  throws IOException
{
  InputStream in = new FileInputStream(src);
  OutputStream out = new FileOutputStream(dst);
  try {
    byte[] buf = new byte[BUFSIZ];
    int n;
    while ((n = in.read(buf)) >= 0)
      out.write(buf, 0, n);
  } finally {
    in.close();
    out.close();
  }
}
```

```
static void copy(File src, File dst)
  throws IOException
{
  InputStream in = new FileInputStream(src);
  try {
    OutputStream out = new FileOutputStream(dst);
    try {
      byte[] buf = new byte[BUFSIZ];
      int n;
      while ((n = in.read(buf)) >= 0)
        out.write(buf, 0, n);
    } finally {
      out.close();
    }
  } finally {
    in.close();
  }
}
```

```
static void copy(File src, File dst)
  throws IOException
{
  try (InputStream in = new FileInputStream(src);
       OutputStream out = new FileOutputStream(dst))
  {
    byte[] buf = new byte[8192];
    int n;
    while ((n = in.read(buf)) >= 0)
      out.write(buf, 0, n);
  }
}
```

Project Coin: try-with-resources

- Reduces boilerplate coding for a common idiom
  - Further, this common idiom is *very* error-prone
    - So error-prone that most developers get it wrong!
- Idioms that people get wrong all the time are evidence of language or library design failures
  - Not all such failures can be fixed in a compatible way

ORACLE

Project Coin Features beyond Java 7

- Not yet specified, but will likely use the same community process to propose and select features
- Possible candidates include
  - Collection literals
  - Large arrays
  - Multi-line strings
  - Your favorite feature...

ORACLE

7 & 8 & 9 . . .

Productivity
Performance
Universality
Modularity
Integration
Serviceability

7&8...

Productivity

Performance

Universality

Modularity

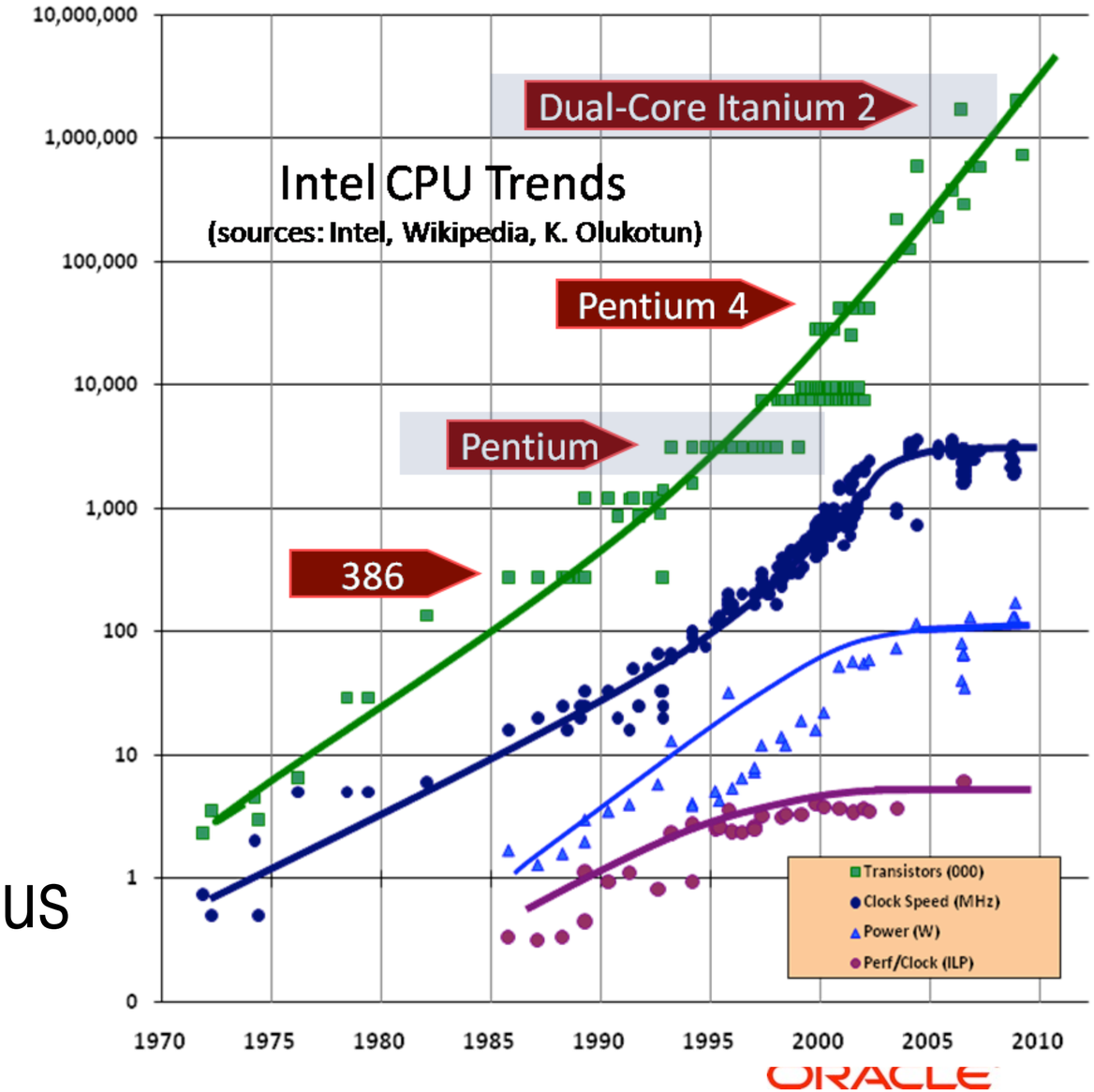
Integration

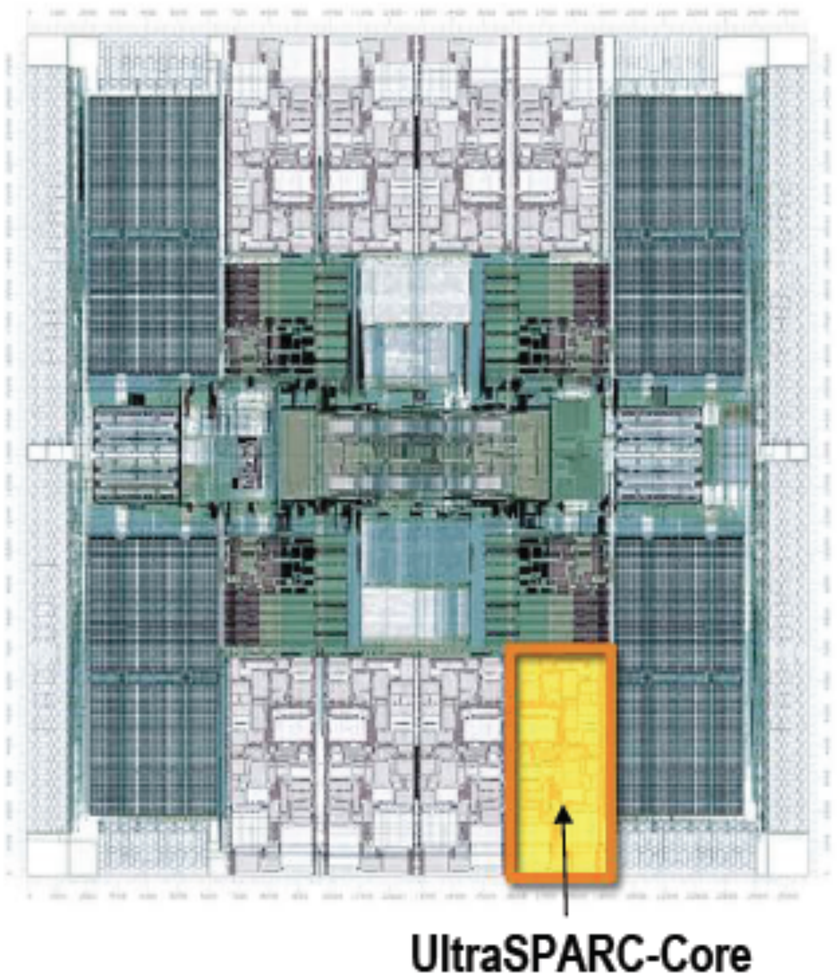
Serviceability

ORACLE

Moore's Law

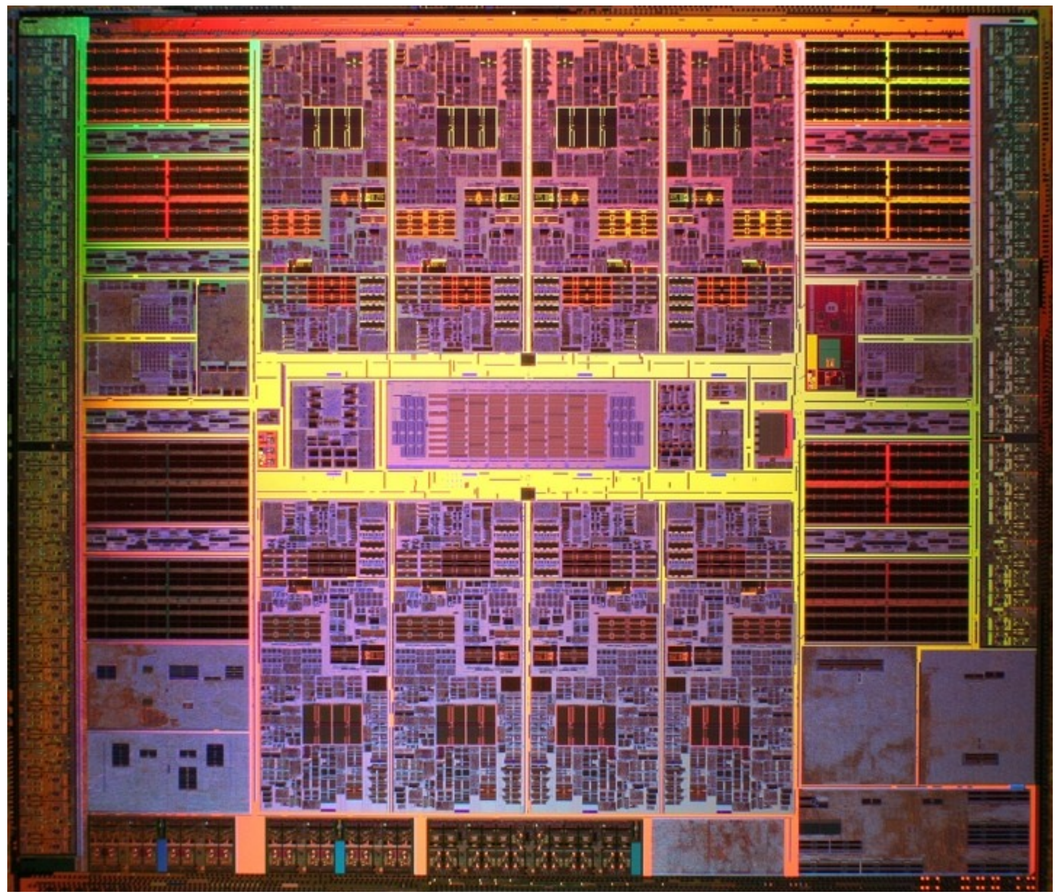
- Number of transistors continues to grow exponentially
- Sequential processor speed does not!
  - Techniques for increasing sequential performance have been mined out
- Going forward, Moore's law gives us *more* cores, not *faster* cores





Niagara 1 (2005)

8 x 4 = **32**

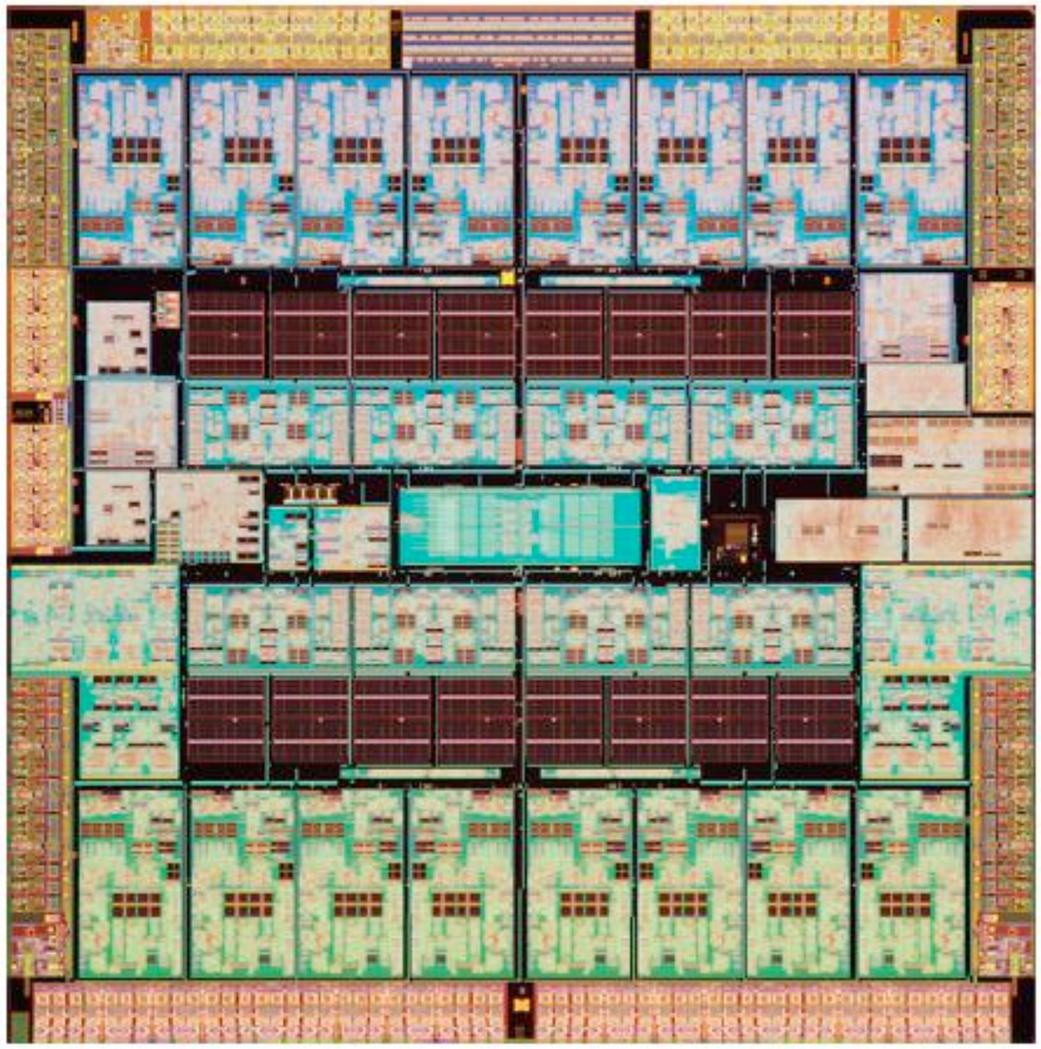


Niagara 1 (2005)

8 x 4 = **32**

Niagara 2 (2007)

8 x 8 = **64**



Niagara 1 (2005)

8 x 4 = **32**

Niagara 2 (2007)

8 x 8 = **64**

Rainbow Falls

16 x 8 = **128**

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

```
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 == 2010)
        max = Math.max(max, s.score);
}
```

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

double max
    = students.filter(new Predicate<Student>() {
          public boolean op(Student s) {
              return s.gradYear == 2010;
          }
      }).map(new Extractor<Student,Double>() {
          public Double extract(Student s) {
              return s.score;
          }
}).max();
```

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

double max
    = students.filter(new Predicate<Student>() {
          public boolean op(Student s) {
              return s.gradYear == 2010;
          }
      }).map(new Extractor<Student,Double>() {
          public Double extract(Student s) {
              return s.score;
          }
      }).max();
```

```
double max
    = students.filter(#{ Student s -> s.gradYear == 2010 })
              .map(   #{ Student s -> s.score })
              .max();
```

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

double max
    = students.filter(new Predicate<Student>() {
          public boolean op(Student s) {
              return s.gradYear == 2010;
          }
      }).map(new Extractor<Student,Double>() {
          public Double extract(Student s) {
              return s.score;
          }
      }).max();

double max                  // Lambda expressions
    = students.filter(#{ Student s -> s.gradYear == 2010 })
              .map(   #{ Student s -> s.score })
              .max();
```

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

double max
    = students.filter(new Predicate<Student>() {
          public boolean op(Student s) {
              return s.gradYear == 2010;
          }
      }).map(new Extractor<Student,Double>() {
          public Double extract(Student s) {
              return s.score;
          }
      }).max();

double max                  // Lambda expressions
    = students.filter(#{ s -> s.gradYear == 2010 })
              .map(   #{ s -> s.score })
              .max();
```

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

double max                  // Lambda expressions
    = students.filter(#{ s -> s.gradYear == 2010 })
              .map(   #{ s -> s.score })
              .max();
```

Lambda expressions

- Lambda expressions are lexically scoped anonymous methods
  - Not members of any class
- Encourage libraries to support internal iteration
  - So libraries can manage parallelism
- Big challenge: don't want to have two different kinds of libraries
  - Want lambda expressions to "just work" with existing APIs that use function-like types such as Runnable or ActionListener

SAM types

- We define a *SAM type* as an interface or abstract class with a Single Abstract Method
  - Runnable, Callable, TimerTask, ActionListener are all SAM types
  - Many APIs already use SAM types extensively
- We define a *SAM-conversion* that converts a lambda expression to any compatible SAM type
  - So you can pass a lambda to a library that expects, say, Callable
- In such a context, the type of the lambda *is* Callable
  - This is *target typing*
  - Enables inference of lambda parameter types

```
interface Collection<T> {

    public int size();
    ...
```

```
interface Collection<T> {

    public int size();
    ...
    Collection<T> filter(Predicate<T> p)
        default Collections.<T>filter;

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

    <V extends Comparable<? super V> V max()
        default Collections.<V>max;
}
```

```
interface Collection<T> {     // Extension methods

    public int size();
    ...
    Collection<T> filter(Predicate<T> p)
        default Collections.<T>filter;

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

    <V extends Comparable<? super V> V max()
        default Collections.<V>max;
}
```

Extension methods

- Compatible mechanism for adding methods to interfaces
  - Provide a new signature AND a default implementation
- Implementations can inherit or override default
  - These are *virtual* extension methods
- Less problematic than multiple inheritance, mixins, traits

```
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person x, Person y) {
        return x.getLastName().compareTo(y.getLastName());
    }
});
```

```
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person x, Person y) {
        return x.getLastName().compareTo(y.getLastName());
    }
});
```

Lambda expression + SAM conversion

```
Collections.sort(people,
    #{ Person x, Person y
        -> x.getLastName().compareTo(y.getLastName())};
});
```

```
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person x, Person y) {
        return x.getLastName().compareTo(y.getLastName());
    }
});
```

Lambda expression + SAM conversion

```
Collections.sort(people,
    #{ Person x, Person y
        -> x.getLastName().compareTo(y.getLastName())};
});
```

Better libraries

```
Collections.sortBy(people, #{ Person p -> p.getLastName() });
```

```
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person x, Person y) {
        return x.getLastName().compareTo(y.getLastName());
    }
});
```

Lambda expression + SAM conversion

```
Collections.sort(people,
    #{ Person x, Person y
        -> x.getLastName().compareTo(y.getLastName())};
});
```

Better libraries

```
Collections.sortBy(people, #{ Person p -> p.getLastName() });
```

Type inference

```
Collections.sortBy(people, #{ p -> p.getLastName() });
```

```
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person x, Person y) {
        return x.getLastName().compareTo(y.getLastName());
    }
});
```

Lambda expression + SAM conversion

```
Collections.sort(people,
    #{ Person x, Person y
        -> x.getLastName().compareTo(y.getLastName())};
});
```

Better libraries

```
Collections.sortBy(people, #{ Person p -> p.getLastName() });
```

Type inference

```
Collections.sortBy(people, #{ p -> p.getLastName() });
```

Method references

```
Collections.sortBy(people, #Person.getLastName );
```

ORACLE

```
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person x, Person y) {
        return x.getLastName().compareTo(y.getLastName());
    }
});
```

Lambda expression + SAM conversion

```
Collections.sort(people,
    #{ Person x, Person y
        -> x.getLastName().compareTo(y.getLastName())};
});
```

Better libraries

```
Collections.sortBy(people, #{ Person p -> p.getLastName() });
```

Type inference

```
Collections.sortBy(people, #{ p -> p.getLastName() });
```

Method references

```
Collections.sortBy(people, #Person.getLastName );
```

Extension methods

```
people.sortBy(#Person.getLastName);
```

ORACLE

```
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person x, Person y) {
        return x.getLastName().compareTo(y.getLastName());
    }
});
```

```
people.sortBy(#Person.getLastName);
```

```
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person x, Person y) {
        return x.getLastName().compareTo(y.getLastName());
    }
});
```

Lambda expressions
Better libraries
Type inference
Method references
Extension methods

```
people.sortBy(#Person.getLastName);
```

```
Collections.sort(people, new Comparator<Person>() {
    public int compare(Person x, Person y) {
        return x.getLastName().compareTo(y.getLastName());
    }
});
```

Lambda expressions
Better libraries
Type inference
Method references
Extension methods

More concise
More abstract
Less ceremony
More reuse
More object-oriented

```
people.sortBy(#Person.getLastName);
```

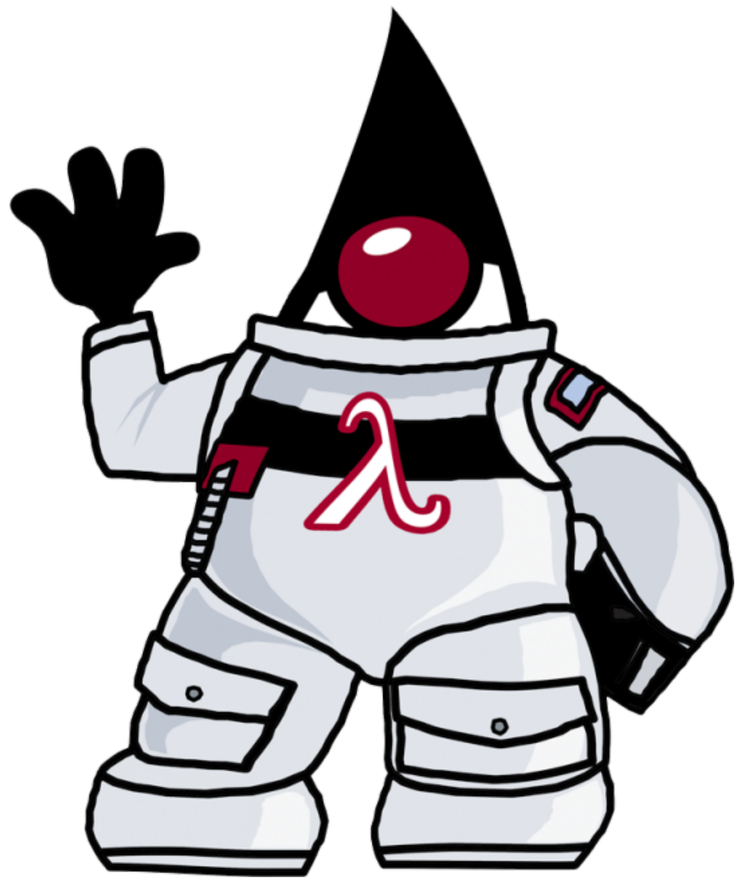
Project Lambda (JSR TBD)

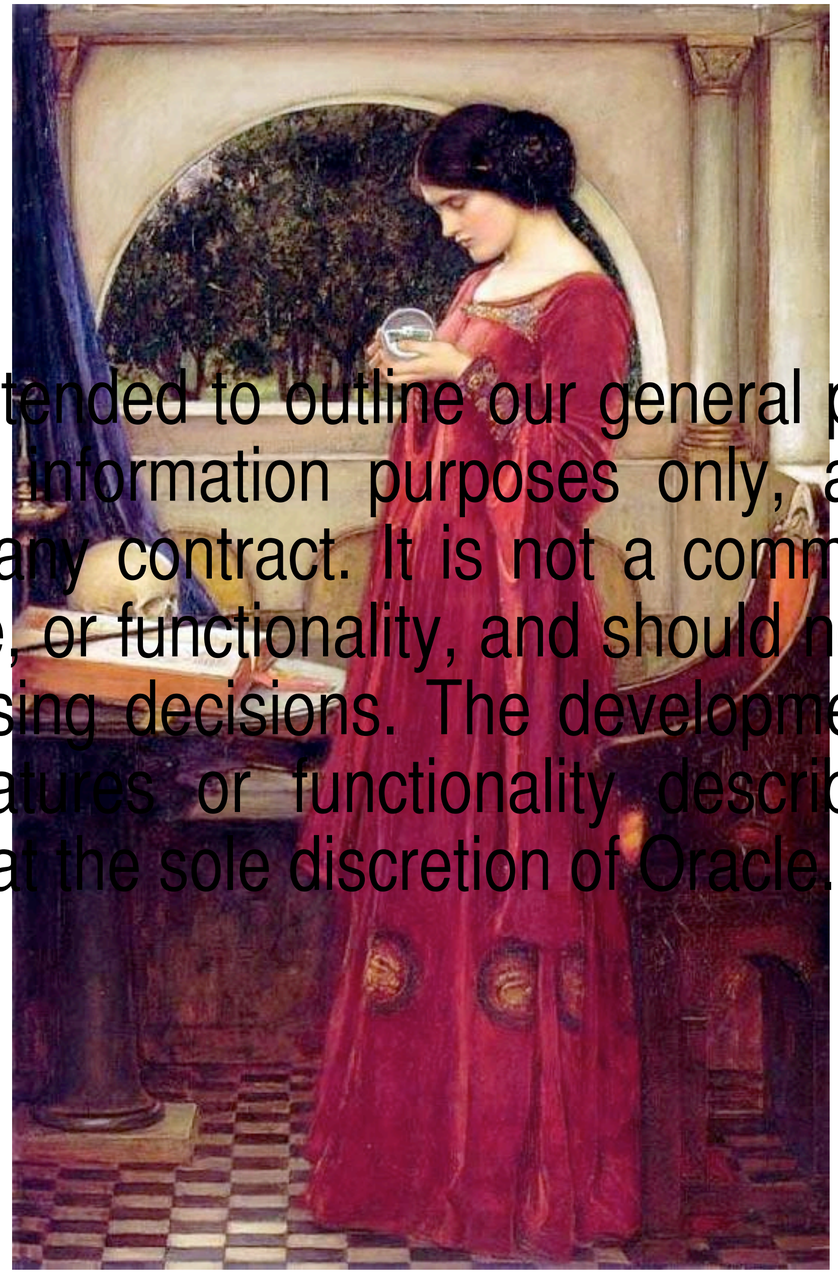
openjdk.java.net/projects/lambda

Project Lambda (JSR TBD)

openjdk.java.net/projects/lambda

- Lambda expressions
- Extension methods for interface evolution
- SAM conversion with target typing
- Method references
- Library enhancements for internal iteration





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.

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
  - a clear semantic model greatly boosts readability
  - every “good” feature adds more “bad” weight
  - sometimes it is best to leave things out
- One language: with same meaning everywhere

Evolving the Language

- We will evolve the Java Language
- But cautiously, with a long-term view
  - we want Java to be around in 2030
  - we can't take a slash-and-burn approach
  - “first do no harm”
- We will add a few selected features periodically
  - aimed at developer productivity
  - while preserving clarity and simplicity

```
void foo(List<Object> ls) {
  if (ls instanceof List<Integer>) { ... }
}
```

```
void foo(List<Object> ls) {
  if (ls instanceof List<Integer>) { ... }
}
```

✗ illegal generic type for instanceof

```
void foo(List<Object> ls) {
  if (ls instanceof List<Integer>) { ... }
}
```

✗ illegal generic type for instanceof

```
int sum(List<Integer> ls) {
  int s = 0;
  for (Integer v : ls)
      s += v;
  return s;
}
```

```
void foo(List<Object> ls) {
  if (ls instanceof List<Integer>) { ... }
}
```

✗ illegal generic type for instanceof

```
int sum(List<Integer> ls) {
  int s = 0;
  for (Integer v : ls)
      s += v;
  return s;
}
```

```
int sum(List<String> ls) {
  int s = 0;
  for (String v : ls)
      s += Integer.parseInt(v);
  return s;
}
```

```
void foo(List<Object> ls) {
  if (ls instanceof List<Integer>) { ... }
}
```

✗ illegal generic type for instanceof

```
int sum(List<Integer> ls) {
  int s = 0;
  for (Integer v : ls)
      s += v;
  return s;
}
```

```
int sum(List<String> ls) {
  int s = 0;
  for (String v : ls)
      s += Integer.parseInt(v);
  return s;
}
```

✗ name clash: sum(List<String>) and sum(List<Integer>) have the same erasure

Language Futures: Reification

```
void foo(List<Object> ls) {
  if (ls instanceof List<Integer>) { ... }
}
```

✗ illegal generic type for instanceof

```
int sum(List<Integer> ls) {
  int s = 0;
  for (Integer v : ls)
      s += v;
  return s;
}
```

```
int sum(List<String> ls) {
  int s = 0;
  for (String v : ls)
      s += Integer.parseInt(v);
  return s;
}
```

✗ name clash: sum(List<String>) and sum(List<Integer>) have the same erasure

Language Futures: Reification

```
List<int> = new ArrayList<>;
```

Language Futures: Reification

```
List<int> = new ArrayList<>;
```

Language Futures

Language Futures

```
class Node {

    private Node parent;
    Node getParent() { return parent; }

    private Node leftChild;
    Node getLeftChild() { return leftChild; }

    private Node rightChild;
    Node getRightChild() { return rightChild; }

}
```

Language Futures: Value Classes

```
value class Node {

    private Node parent;
    Node getParent() { return parent; }

    private Node leftChild;
    Node getLeftChild() { return leftChild; }

    private Node rightChild;
    Node getRightChild() { return rightChild; }

}
```

Language Futures: Properties

```
value class Node {

    Node property parent;
    Node property leftChild;
    Node property rightChild;

}
```

7&9...

Productivity

Performance

Universality

Modularity

Integration

Serviceability

ORACLE

7 & 8 & 9 . . .

Productivity

Performance

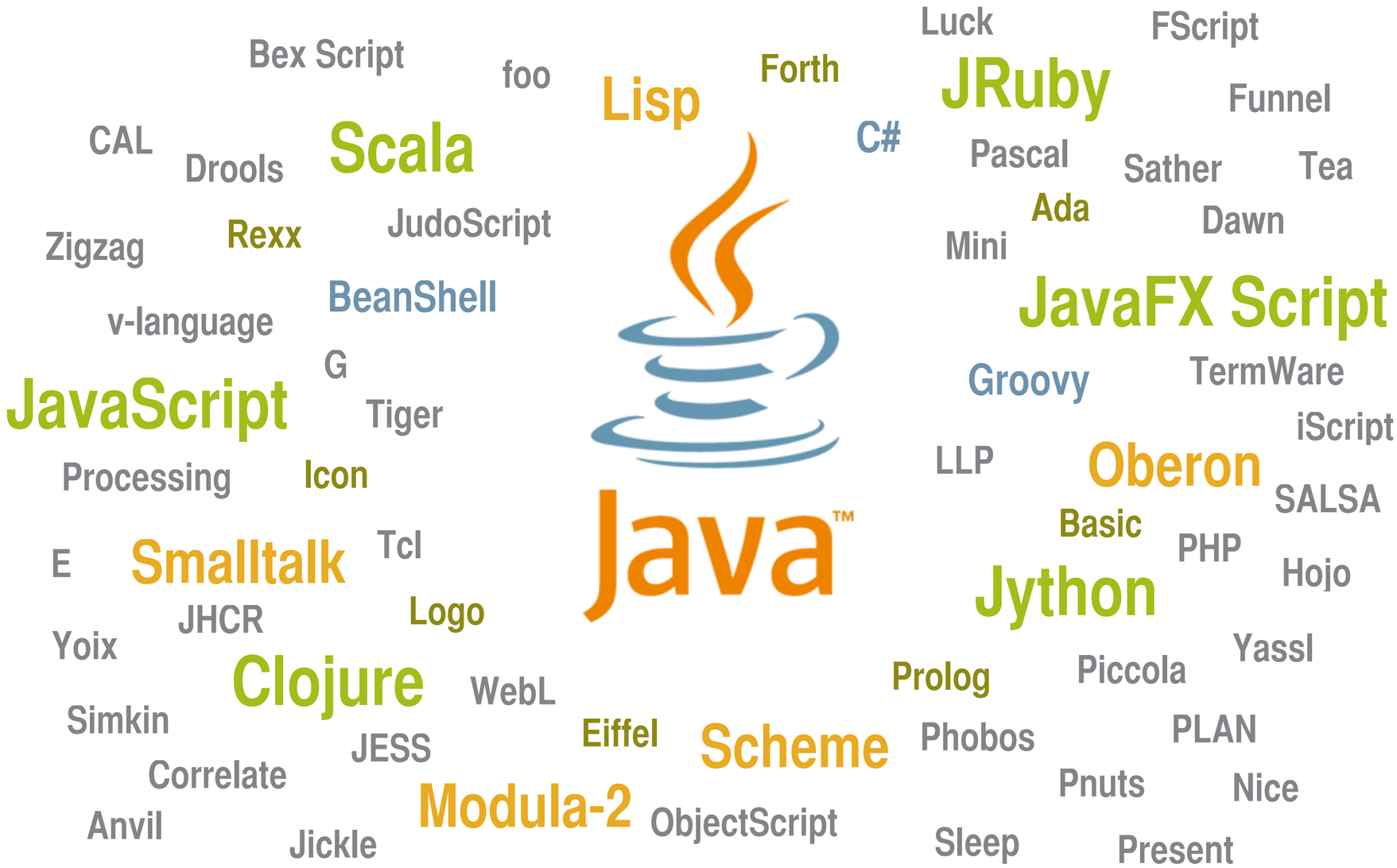
Universality

Modularity

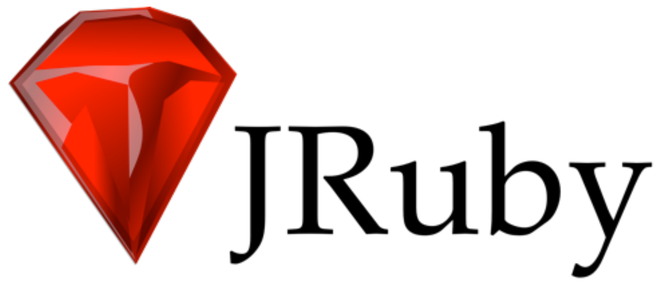
Integration

Serviceability

ORACLE

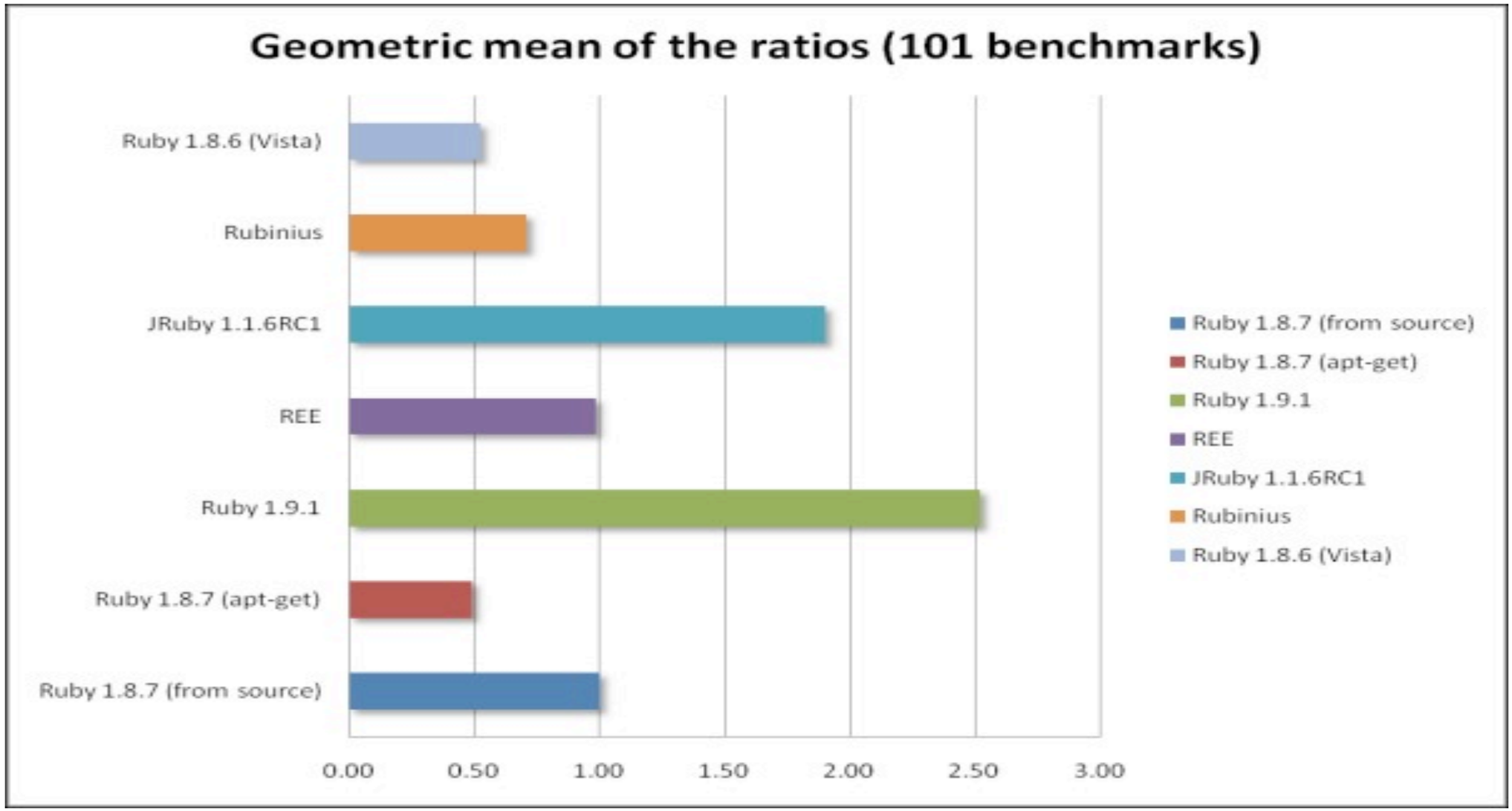


ORACLE



ORACLE

The Great Ruby Shootout



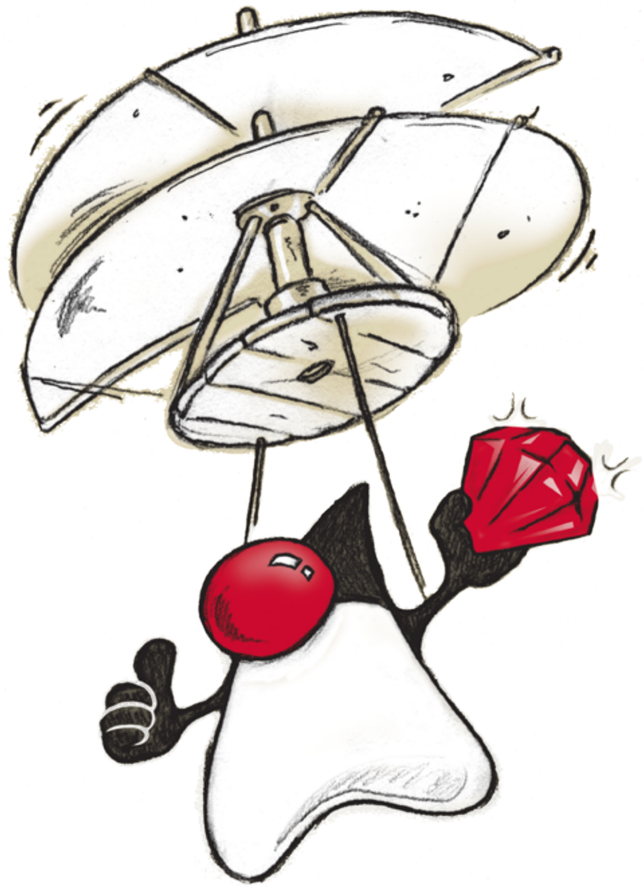
ORACLE

The DaVinci Machine (JSR 292)

openjdk.java.net/projects/mlvm

The DaVinci Machine (JSR 292)

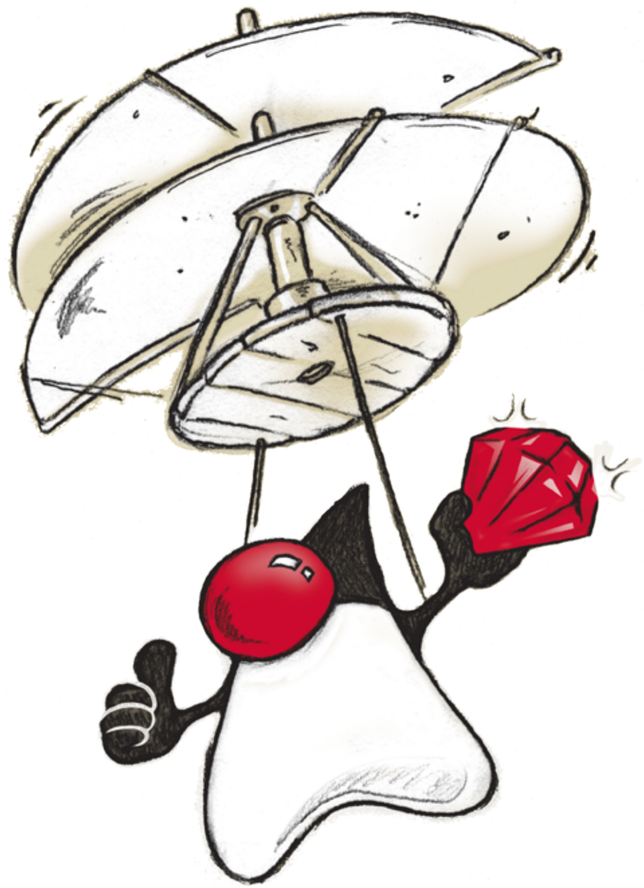
openjdk.java.net/projects/mlvm



The DaVinci Machine (JSR 292)

openjdk.java.net/projects/mlvm

- Invokedynamic bytecode
- Method handles
- *Interface injection*
- *Tail calls*
- *Coroutines*
- *HotSwap*
- *Advanced arrays*



ORACLE

7 & 8 & 9 . . .

Productivity

Performance

Universality

Modularity

Integration

Serviceability

ORACLE

7 & 8 & 9 . . .

Productivity
Performance
Universality
Modularity
Integration
Serviceability

$

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

```
$  java  -cp $APPHOME/lib/jdom-1.0jar\
:$APPHOME/lib/jaxen-1.0.jar\
:$APPHOME/lib/saxpath-1.0.jar\
:$APPHOME/lib/rome-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 $APPHOME/lib/jdom-1.0jar\
:$APPHOME/lib/jaxen-1.0.jar\
:$APPHOME/lib/saxpath-1.0.jar\
:$APPHOME/lib/rome-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

org.planetjdk.aggregator

joda-time-1.6

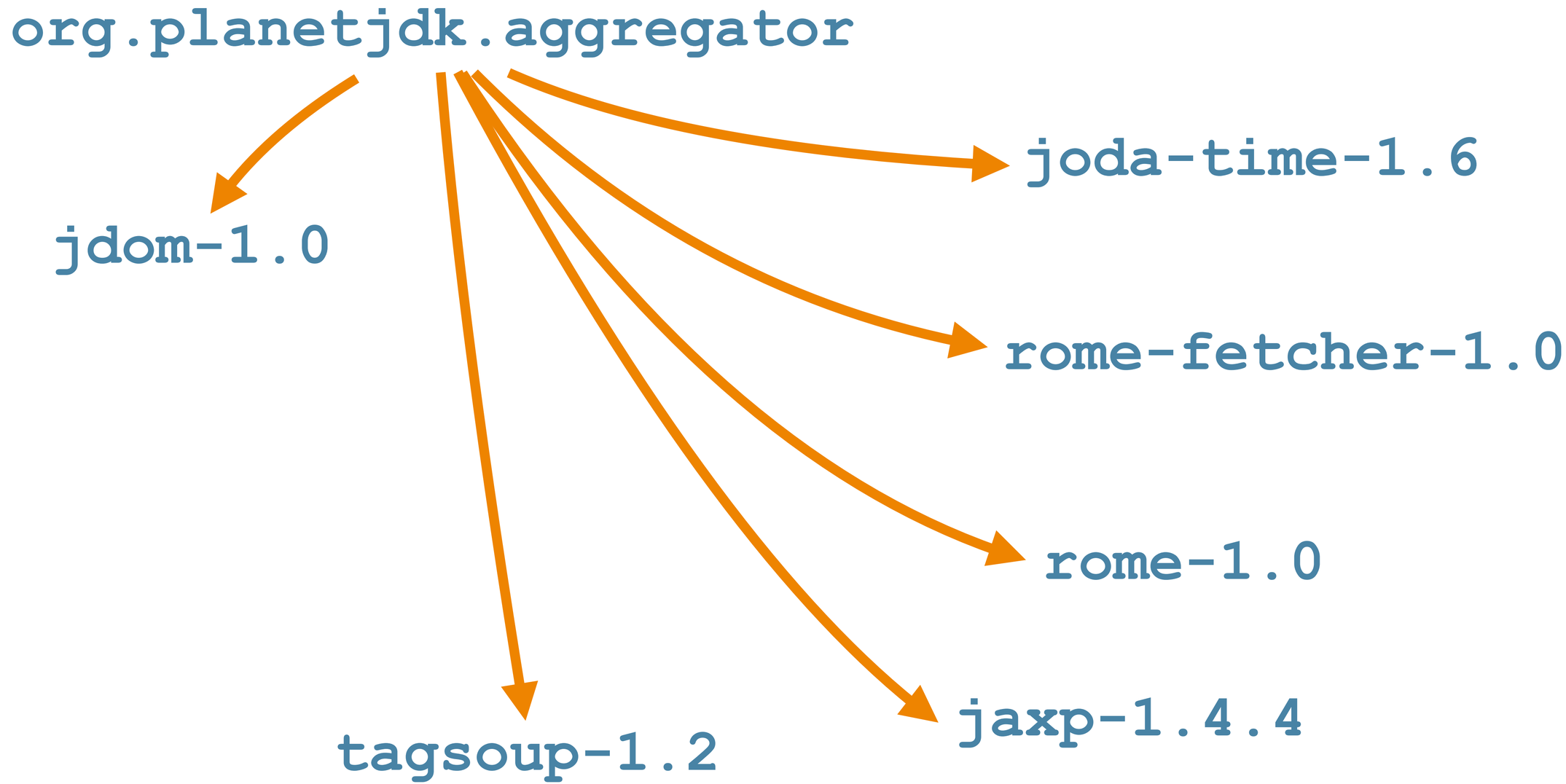
jdom-1.0

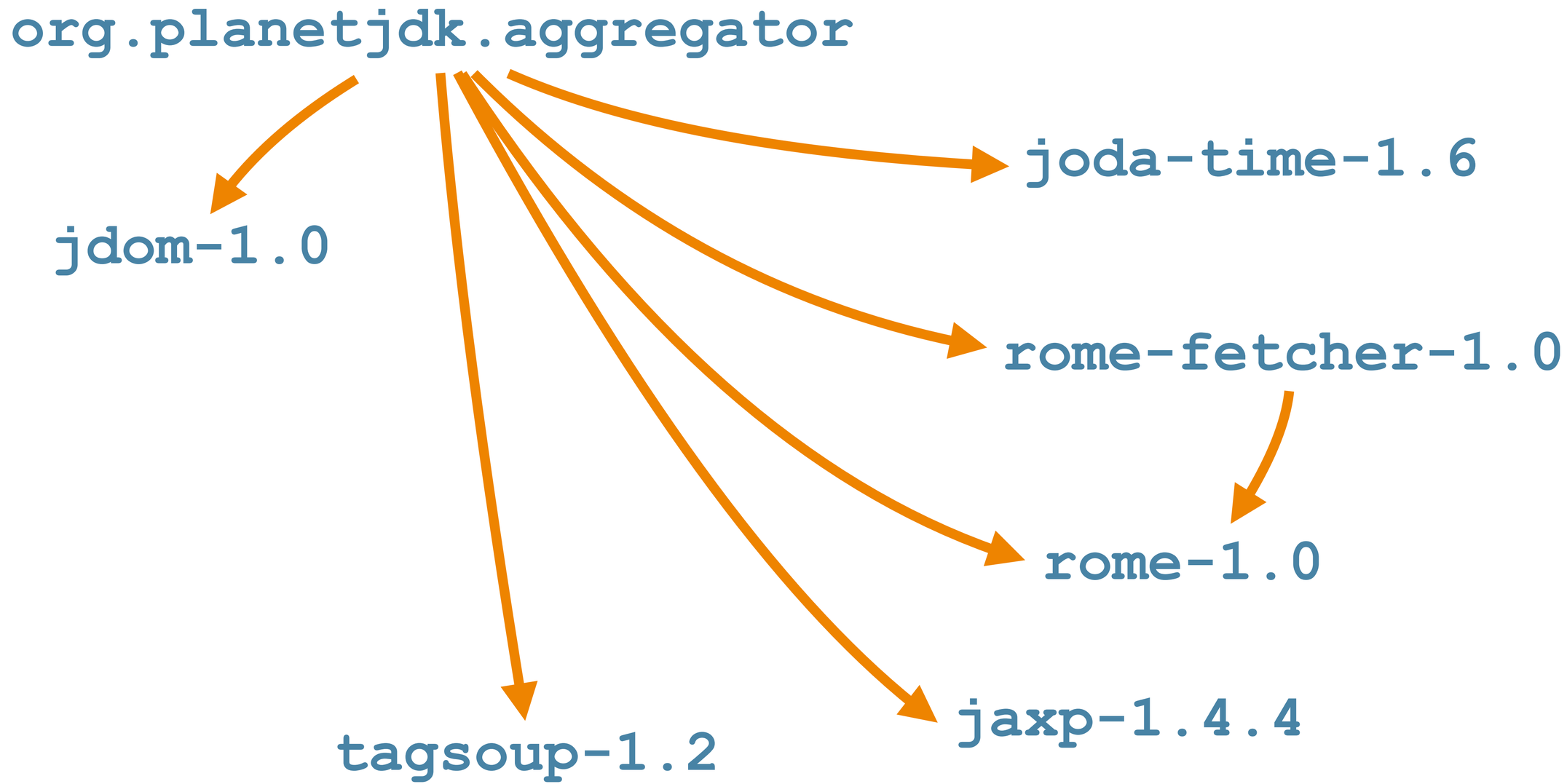
rome-fetcher-1.0

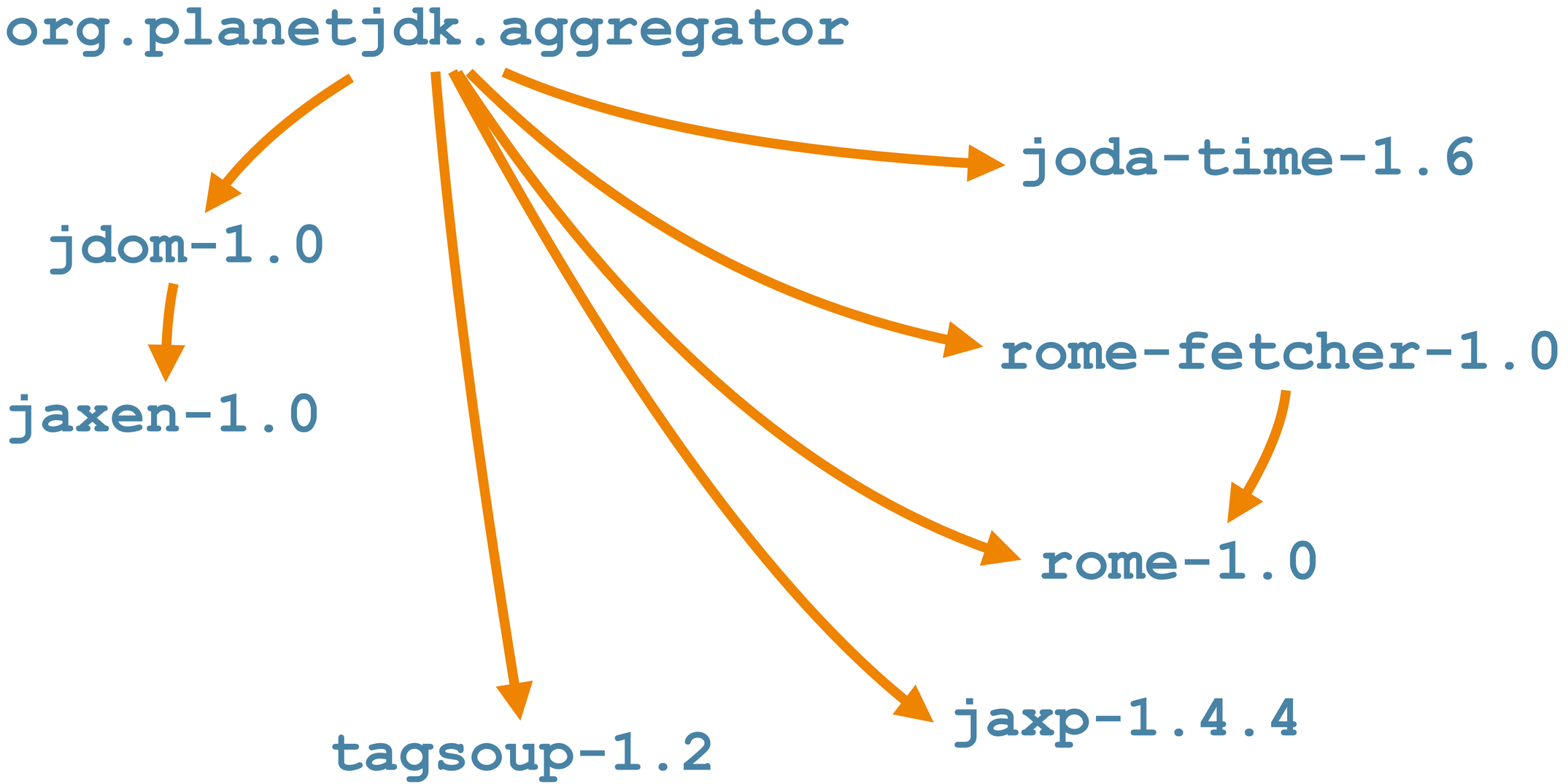
rome-1.0

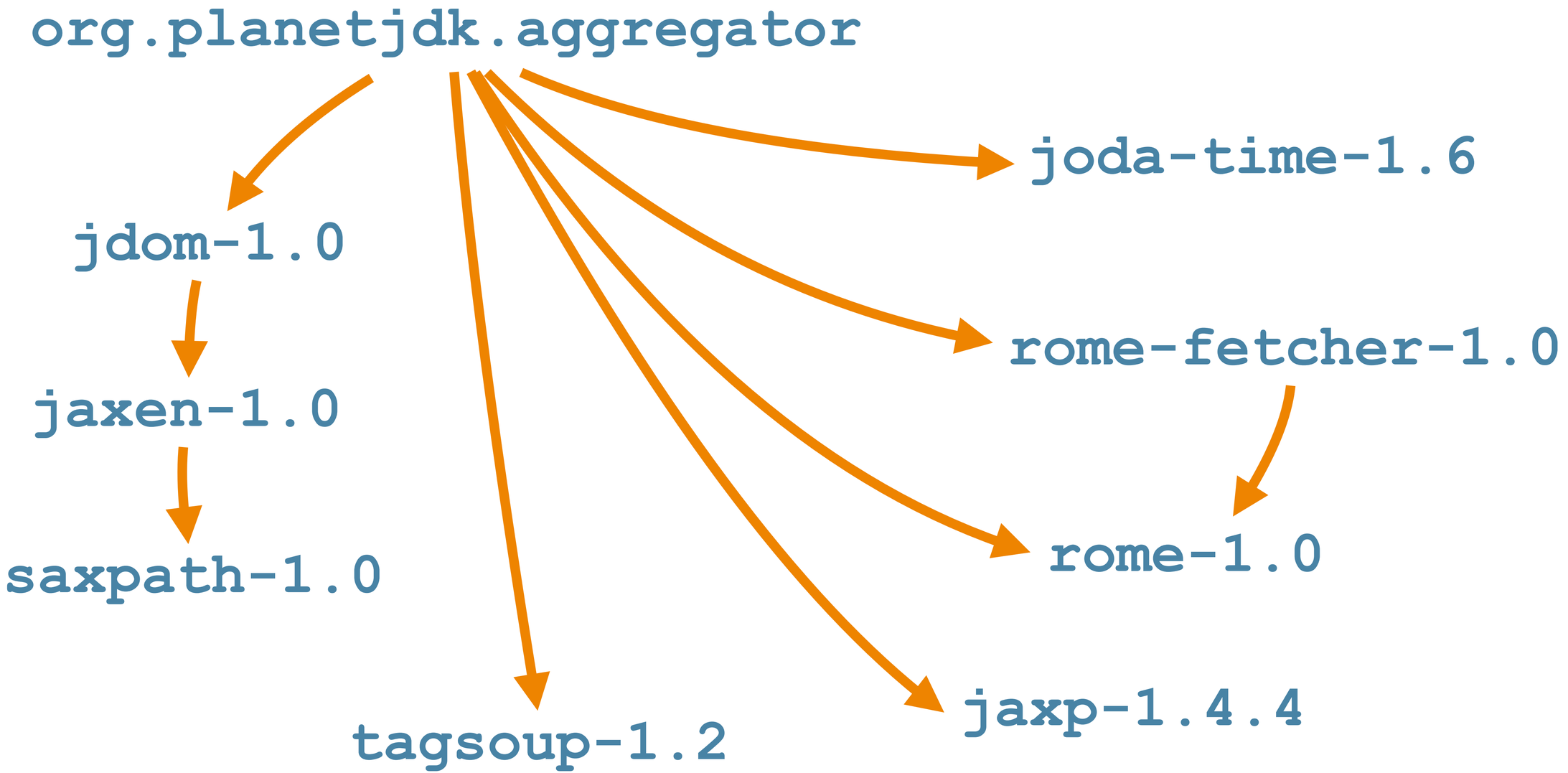
jaxp-1.4.4

tagsoup-1.2









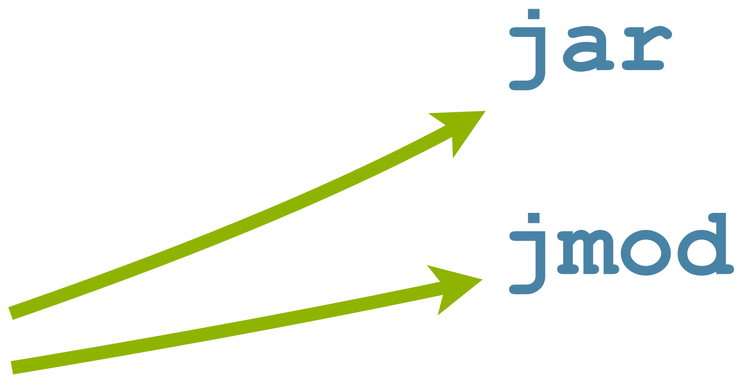
```
// module-info.java

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;
}
```

```
// 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;
}
```

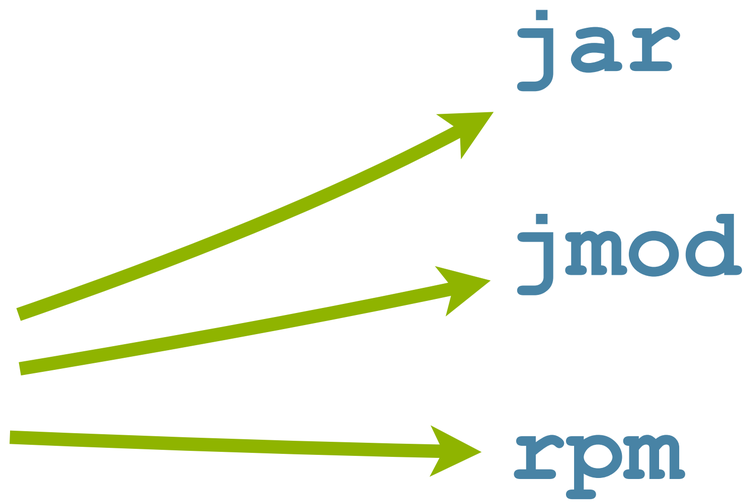
jar

```
// 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;
}
```

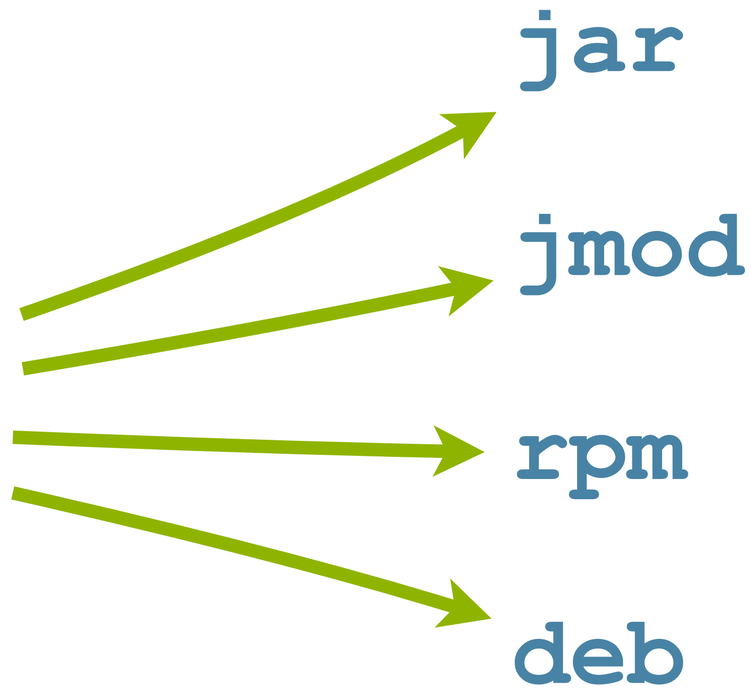
```
// module-info.java

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;
}
```



```
// 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;
}
```



```
// 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;
}
```



mvn

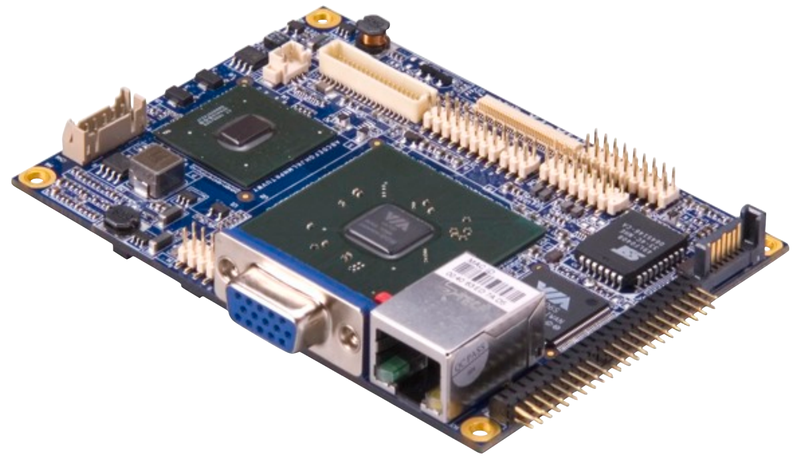
```
// module-info.java

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;
}
```

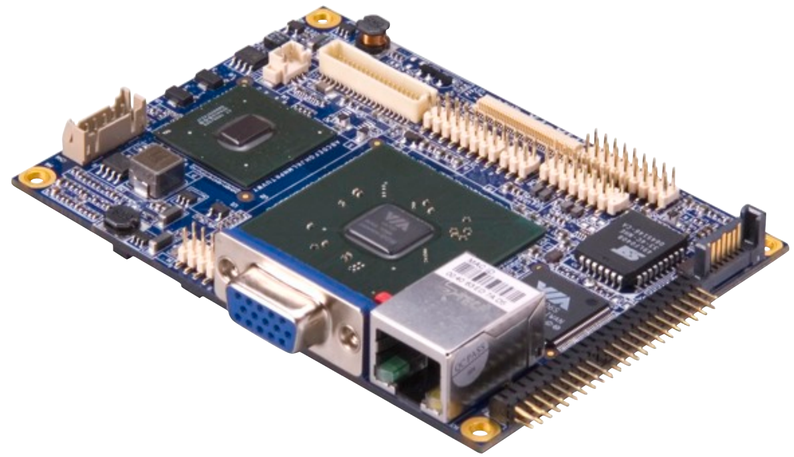
jar

jmod

rpm

deb

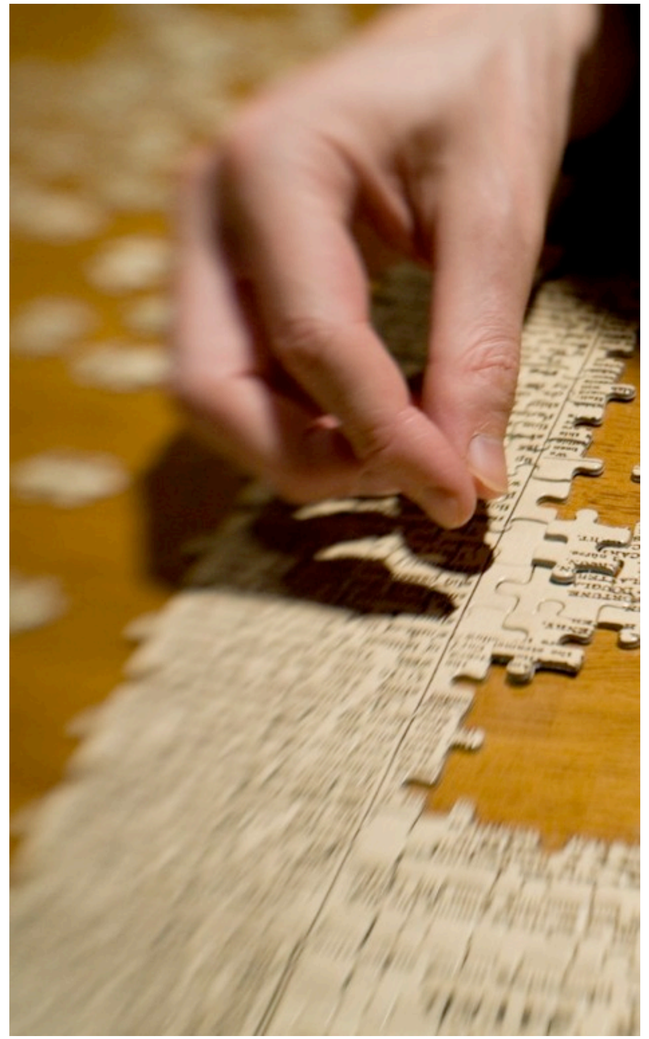




http://www.flickr.com/photos/thatguyfromcchs08/2300190277
http://www.flickr.com/photos/viagallery/2290654438

Project Jigsaw

openjdk.java.net/projects/jigsaw



http://www.flickr.com/photos/lizadaly/2944362379

7 & 8 & 9 . . .

Productivity
Performance
Universality
Modularity
Integration
Serviceability

7&8...

Productivity
Performance
Universality
Modularity
Integration
Serviceability

ORACLE

HotSpot and JRockit

- Oracle now has two VMs, HotSpot and JRockit
  - Each have their respective strengths
  - Would be inefficient to continue development on both
- Goal: single JVM with best features of each
- Plan: merge best features of JRockit into HotSpot
  - There were already lots of features in common!

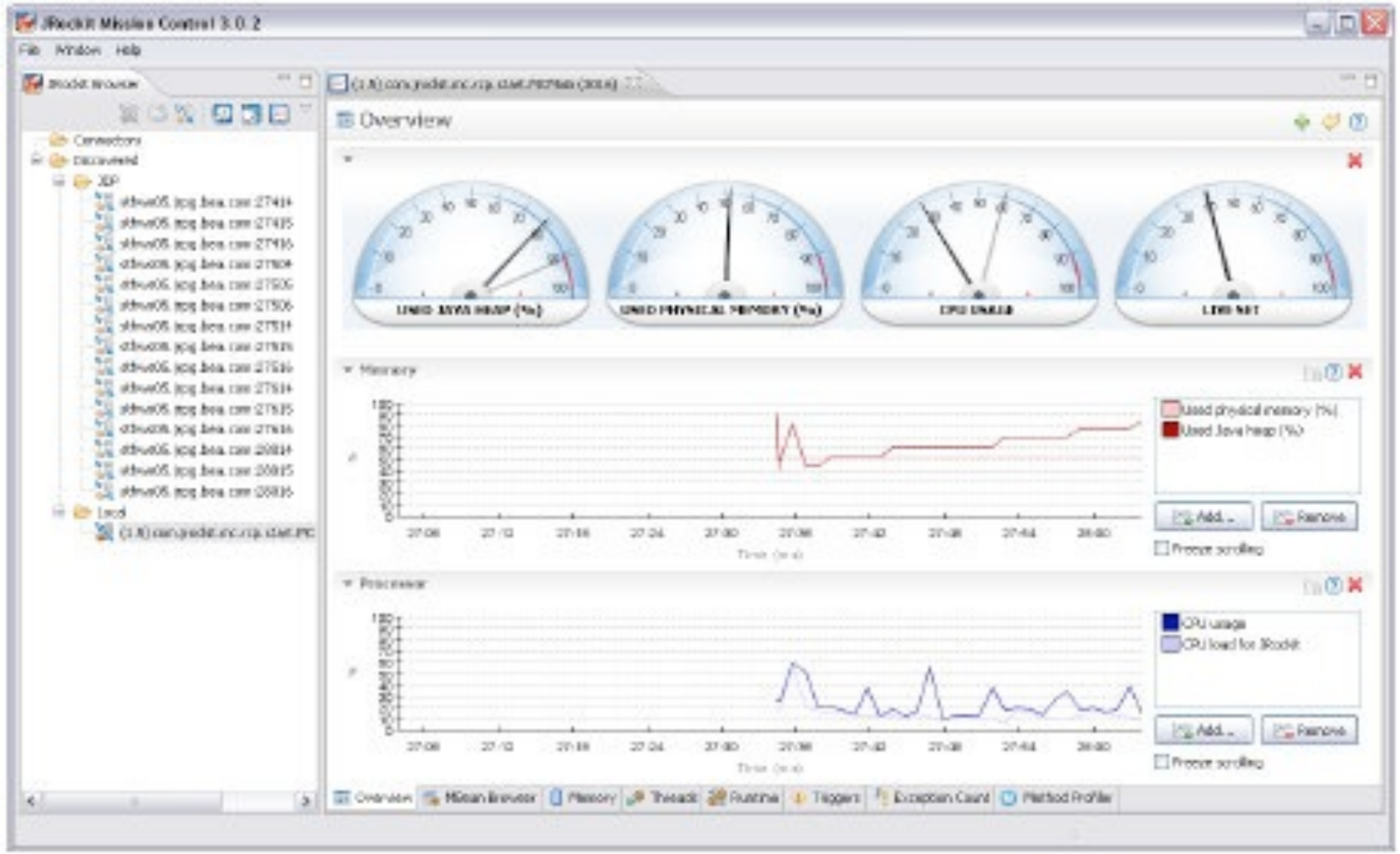
JRockit value-add features

JRockit value-add features

- Flight Recorder
- Mission Control
- Elimination of permanent generation
- Zero-copy IO (via object pinning)
- Native memory tracking
- GC and compiler control APIs
- Virtual Edition
- Soft real-time GC

JVM Convergence

JVM Convergence



7&8...

Productivity
Performance
Universality
Modularity
Integration
Serviceability

ORACLE

7&8...

Productivity
Performance
Universality
Modularity
Integration
Serviceability

7

7

Project Coin (JSR TBD)
InvokeDynamic (JSR 292)
Fork/Join Framework
Project Jigsaw
Project Lambda (JSR TBD)

7

Project Coin (JSR TBD)
InvokeDynamic (JSR 292)
Fork/Join Framework
Project Jigsaw
Project Lambda (JSR TBD)

Strict Verification
Type Annotations (JSR 308)
Parallel Class Loaders
Bulk-Data Operations
Phasers
Transfer Queues
More New I/O (JSR 203)

Unicode 6.0
Enhanced Locales
SDP & SCTP
TLS 1.2
ECC

JDBC 4.1
XRender Pipeline
Swing JDatePicker
Swing JLayer
Swing Nimbus

7

Mid 2012

- Strict Verification
- Type Annotations (JSR 308)
- Parallel Class Loaders
- Bulk-Data Operations
- Phasers
- Transfer Queues
- More New I/O (JSR 203)

Project Coin (JSR TBD)

InvokeDynamic (JSR 292)

Fork/Join Framework

Project Jigsaw

Project Lambda (JSR TBD)

- Unicode 6.0
- Enhanced Locales
- SDP & SCTP
- TLS 1.2
- ECC

- JDBC 4.1
- XRender Pipeline
- Swing JDatePicker
- Swing JLayer
- Swing Nimbus

7

Mid 2012

Project Coin (JSR TBD)
InvokeDynamic (JSR 292)
Fork/Join Framework
Project Jigsaw
Project Lambda (JSR TBD)

Strict Verification
Type Annotations (JSR 308)
Parallel Class Loaders
Bulk-Data Operations
Phasers
Transfer Queues
More New I/O (JSR 203)

Unicode 6.0
Enhanced Locales
SDP & SCTP
TLS 1.2
ECC

JDBC 4.1
XRender Pipeline
Swing JDatePicker
Swing JLayer
Swing Nimbus

7

Project Coin (JSR TBD)
InvokeDynamic (JSR 292)
Fork/Join Framework

Mid 2012

Strict Verification
Type Annotations (JSR 308)
Parallel Class Loaders
Bulk-Data Operations
Phasers
Transfer Queues
More New I/O (JSR 203)

Project Jigsaw
Project Lambda

Unicode 6.0
Enhanced Locales
SDP & SCTP
TLS 1.2
ECC

JDBC 4.1
XRender Pipeline
Swing JDatePicker
Swing JLayer
Swing Nimbus

7

Project Coin (JSR TBD)
InvokeDynamic (JSR 292)
Fork/Join Framework

Mid 2012

- Strict Verification
- ✘ Type Annotations (JSR 308)
- Parallel Class Loaders
- ✘ Bulk-Data Operations
- Phasers
- Transfer Queues
- More New I/O (JSR 203)

Project Jigsaw
Project Lambda

- Unicode 6.0
- Enhanced Locales
- SDP & SCTP
- TLS 1.2
- ECC

- JDBC 4.1
- XRender Pipeline
- ✘ Swing JDatePicker
- Swing JLayer
- Swing Nimbus

7

Project Coin (JSR TBD)
InvokeDynamic (JSR 292)
Fork/Join Framework

Mid 2012

Strict Verification
Parallel Class Loaders
Phasers
Transfer Queues
More New I/O (JSR 203)
Unicode 6.0
Enhanced Locales

SDP & SCTP
TLS 1.2
ECC
JDBC 4.1
XRender Pipeline
Swing JLayer
Swing Nimbus

Project Jigsaw
Project Lambda

Type Annotations (JSR 308)
Bulk-Data Operations
Swing JDatePicker

7

Mid 2012

Project Coin (JSR TBD)
InvokeDynamic (JSR 292)
Fork/Join Framework

Strict Verification
Parallel Class Loaders
Phasers
Transfer Queues
More New I/O (JSR 203)
Unicode 6.0
Enhanced Locales

SDP & SCTP
TLS 1.2
ECC
JDBC 4.1
XRender Pipeline
Swing JLayer
Swing Nimbus

Project Jigsaw
Project Lambda

Type Annotations (JSR 308)
Bulk-Data Operations
Swing JDatePicker
Collection Literals

7

Project Coin (JSR TBD)
InvokeDynamic (JSR 292)
Fork/Join Framework

Mid 2011

Strict Verification
Parallel Class Loaders
Phasers
Transfer Queues
More New I/O (JSR 203)
Unicode 6.0
Enhanced Locales

SDP & SCTP
TLS 1.2
ECC
JDBC 4.1
XRender Pipeline
Swing JLayer
Swing Nimbus

Project Jigsaw
Project Lambda

Type Annotations (JSR 308)
Bulk-Data Operations
Swing JDatePicker
Collection Literals

7

Project Coin (JSR TBD)
InvokeDynamic (JSR 292)
Fork/Join Framework

Mid 2011

Strict Verification
Parallel Class Loaders
Phasers
Transfer Queues
More New I/O (JSR 203)
Unicode 6.0
Enhanced Locales
SDP & SCTP
TLS 1.2
ECC
JDBC 4.1
XRender Pipeline
Swing JLayer
Swing Nimbus

8

Project Jigsaw
Project Lambda

Type Annotations (JSR 308)
Bulk-Data Operations
Swing JDatePicker
Collection Literals

Late 2012

ORACLE

1.0 1.1 1.2 1.3 1.4 5.0

Revolution

Evolution

7

6

?

1996 1997 1998 2000 2002 2004 2006 2010

ORACLE

ORACLE

1.0 1.2 5.0 Revolution 7

1.1 1.3 1.4 Evolution 6

1996 1997 1998 2000 2002 2004 2006 2010

ORACLE

ORACLE

1.0 Revolution 1.2 5.0

Evolution 1.1 1.3 1.4 6 7

1996 1997 1998 2000 2002 2004 2006 2010 2011

ORACLE

ORACLE

1.0 1.2 5.0

Revolution

1.1 1.3 1.4

Evolution

6 7 8

1996 1997 1998 2000 2002 2004 2006 2010 2011 2012

ORACLE

"Plus ça change,
plus c'est la même chose."

Jean-Baptiste Alphonse Karr (1808-1890)

OpenJDK

OpenJDK

GPLv2

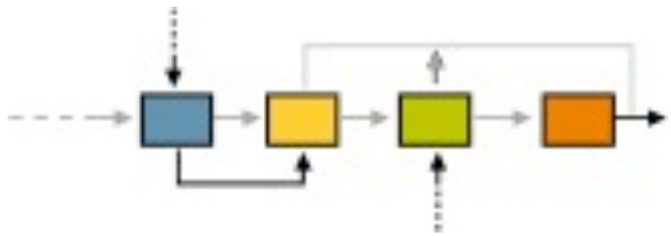
+ Classpath Exception

ORACLE

OpenJDK

GPLv2
+ Classpath Exception

Java Community Process



openjdk.java.net/projects/jdk7
download.java.net/jdk7

ORACLE

The preceding 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.

ORACLE