A Java™ 7 State of the Union

Thorbiörn Fritzon
Lead Technologist

A Big Release

Modularizing the entire JVM

Multi-Language Support

A New Java

Major Performance Upgrade

Modules

Hello World

```
public class Hello {
    public static void main( String[] argv ) {
      System.out.println( "hello, world" );
    }
}
```

Hello World

```
$ time java Hello
hello, world

real  0m0.161s
user  0m0.091s
sys   0m0.046s
```

Hello World

```
$ time python -c 'print "Hello, world\n"'
Hello, world

real  0m0.017s
user  0m0.009s
sys   0m0.008s
```

Hello World

```
$ java -verbose:class Hello | wc -l
     343
```

?

Java 7

?

Java 7
JSR-277

?

Java 7

~~JSR-277~~

?

Java 7

~~JSR-277~~

JSR-294

?

Java 7

~~JSR-277~~

JSR-294

OSGi

?

Java 7

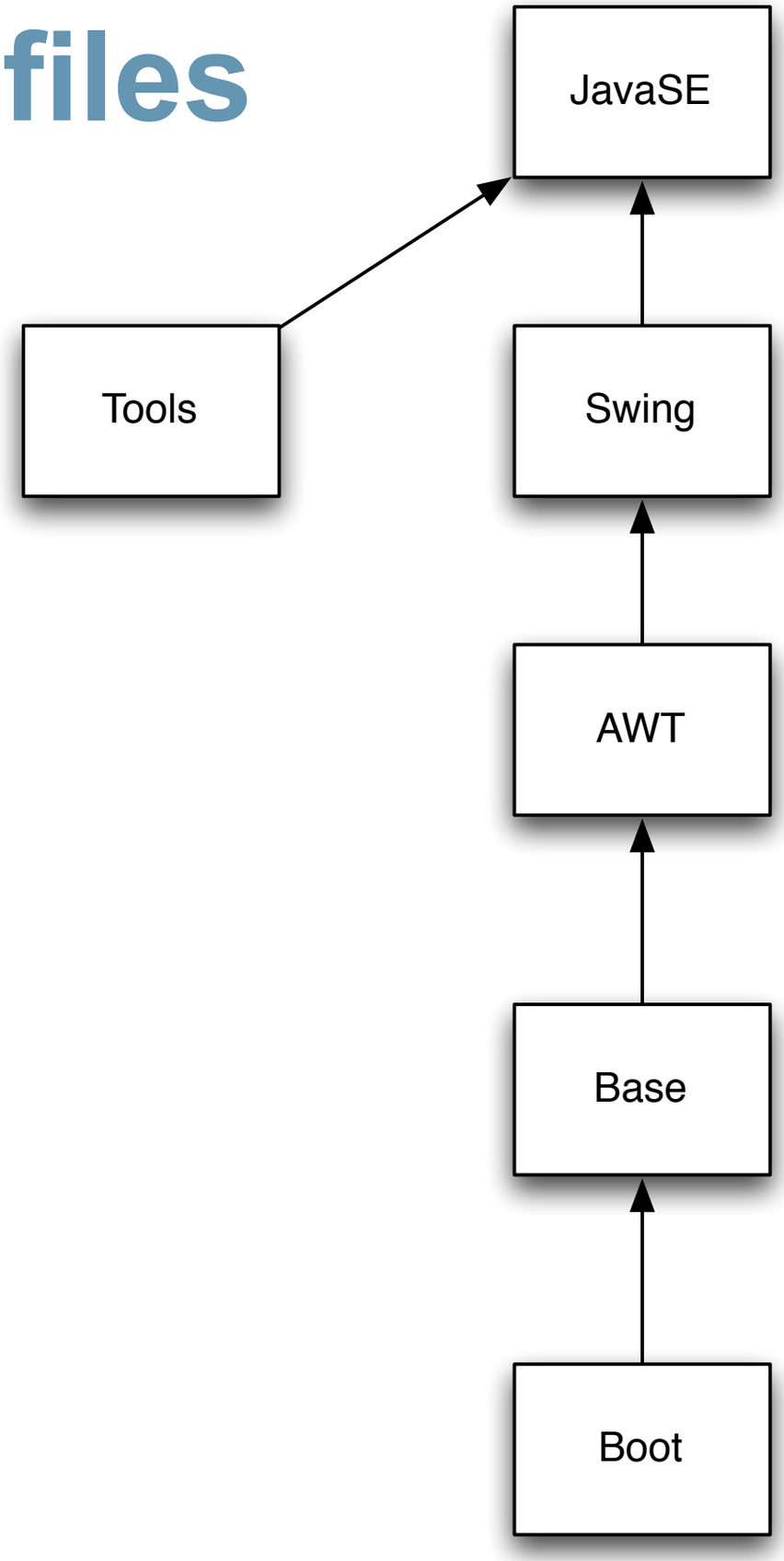
~~JSR-277~~

JSR-294

OSGi

Project Jigsaw

Current Profiles



Project Jigsaw

- Simple, low-level
- Fully leverage JSR-294
- Available for developer use
- No JSR for now, but fully supported by Sun
- Interoperable with OSGi
- Developed in the open

Requirements

- Integrate with the JVM
- Integrate with the language
- Integrate with native packaging
- Support multi-module packages
- Support “friend” modules

JSR-294 Modules

module-info.java:

```
import P.*;
@Annotation
module M1@1.0 provides M2@2.0, M3@3.0 {
  requires M4@4.0, M5@5.0;
  permits  M6;
}
```

Foo.java:

```
module M;
package P;
public class Foo {...}
```

More Languages

Multi-Language Support

- JSR-292
  - DaVinci-Machine
  - invokedynamic
  - method handles
  - (Tail calls, etc.)
- Will Benefit:
  - JRuby
  - Jython
  - Clojure
  - Scala (JVM Closures)
  - ... etc.

Language Changes

Language Changes

- Project Coin (Small Language Changes)
- JSR-308 Type Annotations
- Closures

JSR-308 Type Annotations

```
void foo(Bar x) {
    x.fum(); // Potential NullPointerException
}
```

JSR-308 Type Annotations

```
void foo(@NonNull Bar x) {
    x.fum(); // Statically Checked
}
```

```
void foo(Bar x) {
    x.fum(); // Potential NullPointerException
}
```

Better Type Inference (Diamond)

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

Better Type Inference (Diamond)

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

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

Better Literals

- Integrals:

```
>byte a_o_falk= 0b10010;
>int bar= 1_233_755;
```

- Collections:

```
>List<Integer> pi= [3, 1, 4, 1, 5, 9, 2];
>Set<Integer> primes= {2, 3, 5, 7, 11, 13};
>Set<Senator> honestSenators = {};
>Map<Integer, String> platonicSolids =
     { 4 : "tetrahedron",
       6 : "cube",
       8 : "octahedron",
       12 : "dodecahedron" };
```

More on Collections

- Referencing a List item:

```
>List<Integer> l= [ 2, 4, 6, 8 ];
>assert l[2] == 6;
```

- Setting a List item:

```
>l[2]= 42;
```

- Referencing a Map item:

```
>Map<String, Integer> m= { ‘‘one’’ : 1,
                           ‘‘two’’ : 2 };
>assert m[‘‘one’’] == 1;
```

- Setting a Map Item:

```
>m[‘‘three’’]= 3;
```

Simplified Vararg Method Invocation

```
static <T> List<T> asList(T... elements) { ... }

static List<Callable<String>> stringFactories() {
    Callable<String> a, b, c;
    ...
    Note: A.java uses unchecked or unsafe operations.
    return asList(a, b, c);
}
```

Simplified Vararg Method Invocation

```
Note: A.java enables unsafe generic array creation.
static <T> List<T> asList(T... elements) { ... }

static List<Callable<String>> stringFactories() {
    Callable<String> a, b, c;
    ...
    return asList(a, b, c);
}
```

```
static <T> List<T> asList(T... elements) { ... }

static List<Callable<String>> stringFactories() {
    Callable<String> a, b, c;
    ...
    Note: A.java uses unchecked or unsafe operations.
    return asList(a, b, c);
}
```

JSR-292 Support

- The Dynamic interface has an infinite number of methods:
  - `Object x= Dynamic.∞();`
- Method Handles targets can be called directly:
  - `MethodHandle mh= ...;`
  - `mh.invoke();`
- Exotic Identifiers:
  - `int #”The answer to the ultimate question of life, the universe and everything”= 42;`
- A new wildcard type
  - `Dynamic x= (any type can go here);`
  - `Object y= x.foo(“ABC”).bar(42).baz();`

Strings in a Switch-statement

```
String s= ...;
switch(s) {
 case ”foo”: foo();
 case ”bar”: bar();
 case ”fie”:
 case ”fum”:
  fieFum();
}
```

Safe Re-Throw

```
void someMethod() throws X1,X2 {
  try { /* Something that can throw X1,X2 */ }
  catch (Throwable e) {
    logger.log(e);
    throw e; // Error: Unreported exception Throwable
 }
}
```

Safe Re-Throw

```
void m() throws X1,X2 {
    try { /* Something that can throw X1,X2 */ }
    catch (final Throwable e) {
        logger.log(e);
        throw e; // Compiles OK; can throw X1,X2
    }
}
```

```
void someMethod() throws X1,X2 {
  try { /* Something that can throw X1,X2 */ }
  catch (Throwable e) {
    logger.log(e);
    throw e; // Error: Unreported exception Throwable
 }
}
```

Automatic Resource Block Mgmt

```
// Start a block using two resources, which will automatically
// be cleaned up when the block exits scope
try (BufferedOutputStream bos = ...) {
    ... // Perform action with bos
} catch (IOException e) {
    ...
}
```

Closures

Closure:

> “…a first-class function with free variables that are bound in the lexical environment.”

– Wikipedia

Closure:

> “…a first-class function with free variables that are bound in a lexical environment.

Function type variables

– Wikipedia

Lambda Expressions

- **Lisp:** `(lambda (x) (f x))`
- **Haskell:** `\x -> f(x)`
- **Scala:** `(x:int => f(x))`
- **Groovy:** `(x -> f(x))`
- **Clojure:** `(fn [x] (f x))`

Java 7 Straw Man

```
#()(42)

#(int x)(f(x))

#(int x)(x * x)

#(int x, int y)(x * y)
```

```
#(int x, int y) {
    int z= expensiveComputation(x, y);
    if( z < 0 ) return x;
    if( z > 0 ) return y;
    return 0;
}
```

BGGA

```
{ => return 42; }
{ int x => return f(x); }
{ int x => return x*x; }
{ int x, int y => return x * y; }
```

```
{ int x, int y =>
    int z= expensiveComputation(x, y);
    if( z < 0 ) return x;
    if( z > 0 ) return y;
    return 0;
}
```

Java 7 Straw Man: Function Types

```
#int() fortyTwo= #()(42);

#int(int) doubler = #(int x)(x + x);

#int(int,int) multiplier = #(int x,
int y)(x * y);
```

Function Conversion

```
Thread t= new Thread(new Runnable() {
     public void run() {
          doStuff();
     }
});
```

Function Conversion

```
Thread t= new Thread(new Runnable() {
     public void run() {
          doStuff();
     }
});
```

```
Thread t=
     new Thread(#() { doStuff(); })
```

Function Conversion

```
Thread t= new Thread(new Runnable() {
     public void run() {
          doStuff();
     }
});
```

```
Thread t=
     new Thread(#() { doStuff(); })
```

Handeled automatically for interfaces with one method

Variable Capture (Currying)

```
public #int(int) adder(int x) {
    return #(int y)(x + y);
}
```

```
#int(int) a42= adder(42);
```

```
assert a42(0) == 42
```

Shared Variables

```
shared int count= 0;
Collections.sort(data,
    #(String a, String b) {
        count++;
        return a.length() - b.length();
    });
System.out.println( ‘‘comparisons: ’’ +
                    count );
```

Instance Capture

```
class CountingSorter {
    int count= 0;
    void sort(List<String> data) {
        Collections.sort(data,
                #(String a, String b) {
                    count++;
                    return a.length() -
                            b.length();
                });
    }
}
```

Instance Capture, cont.

```
CountingSorter cs= new CountingSorter();
cs.sort(data);
cs.sort(moreData);
cs.sort(yetMoreData);
System.out.println( “Comparisons total: ” +
                    count );
```

Collection Functions in Haskell

- **Map**

```
>map (\x -> x*x) [1..9]
  –[1, 4, 9, 16, 25, 36, 49, 64, 81]
```

- **Filter**

```
>filter (\x -> mod x 2 == 0) [1..9]
  –[2, 4, 6, 8]
```

- **Reduce**

```
>foldl (\x y -> x * y) 1 [1..5]
  –120
>foldl (*) 1 [1..5]
  –120
```

Extension Methods

```
Set<Integer> s= ...;

Set sqr= s.map(#(int x)(x * x));

int sum= s.reduce(#(int x, int b)(x+b), 0);

Set even= s.filter( #(int x)(x % 2 == 0) );
```

Define f/m/r in the Collections Class

```
class Collections {
    ...
    static <T> Set<T> filter(Set<T> s, #boolean(T) pred) {
        ...
    }

    static <S,T> Set<S> map(Set<T> s, #S(T) func) {
        ...
    }

    static <T> T reduce(Set<T> s, #T(T,T) op, T base) {
        ...
    }
}
```

Import into *Set* Interface

```
interface Set<T> extends Collections<T> {

    ...

    Set<T> filter(#boolean(T))
            import static Collections.filter;

    <S> map(#S(T))
            import static Collections.map;


    T reduce(#T(T,T), T)
            import static Collections.reduce;
}
```

Result

```
int sum= s.reduce(#(int x, int b)(x+b), 0);
```

Will become:

```
int sum= Collections.reduce(
                s,
                #(int x, int b)(x+b),
                0
            );
```

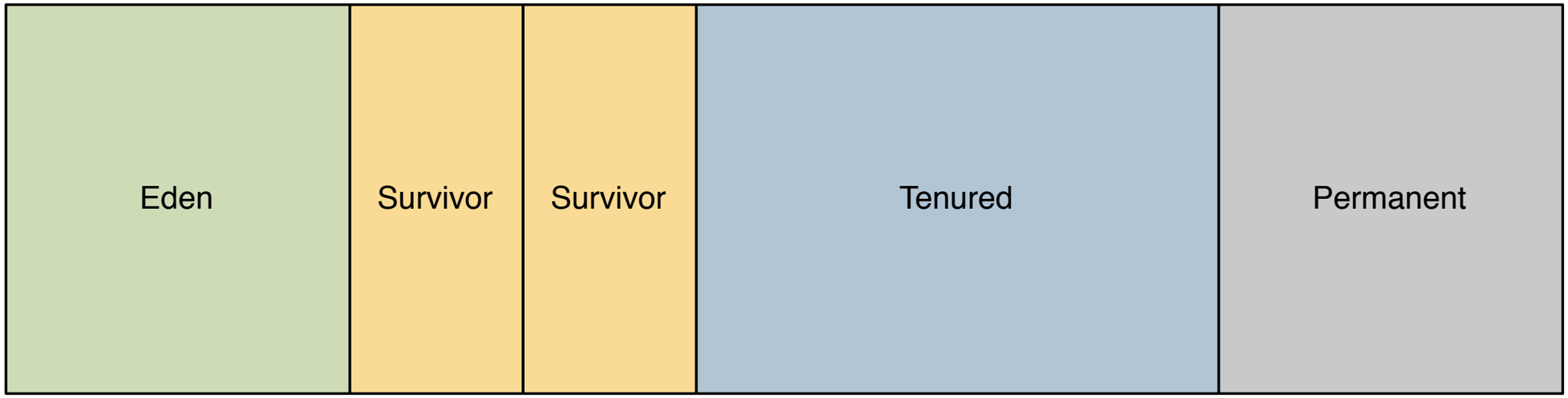
Performance

Performance

- Compressed 64-bit pointers
- G1

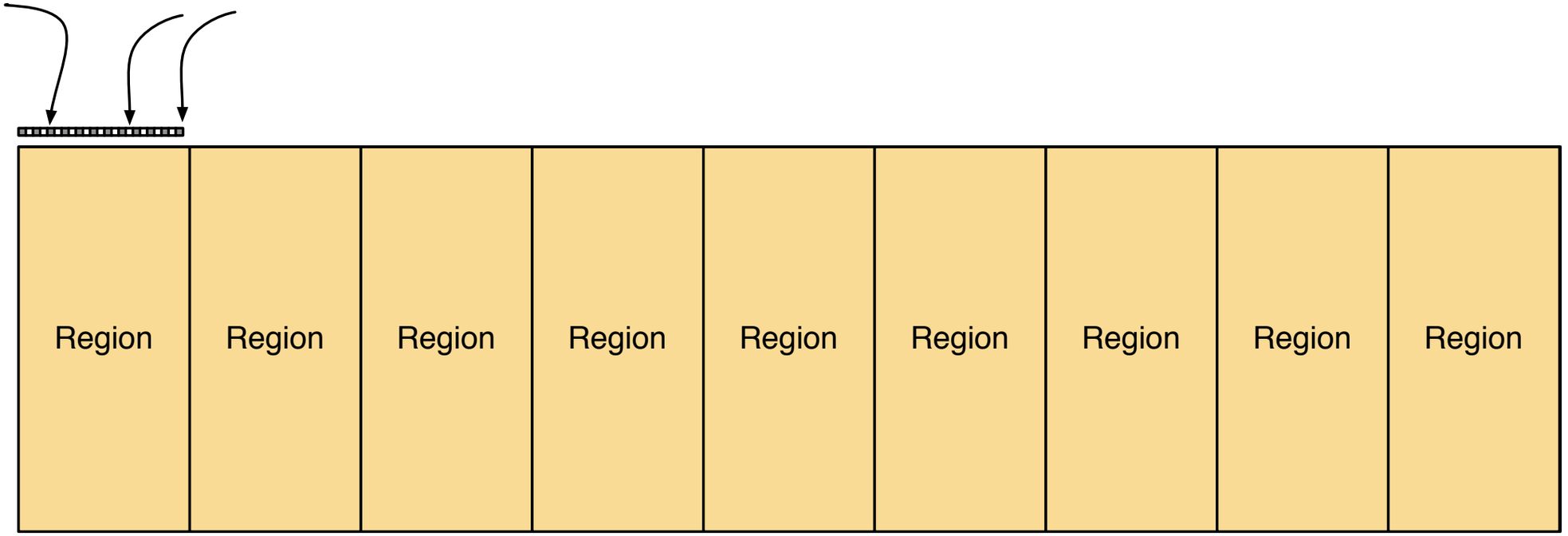
Performance

- Compressed 64-bit pointers
- G1



Performance

- Compressed 64-bit pointers
- G1



ORACLE®


R.I.P
Sun
microsystems
1982-2010
The Network IS

?

Thorbiörn Fritzon
<fritzon@sun.com>