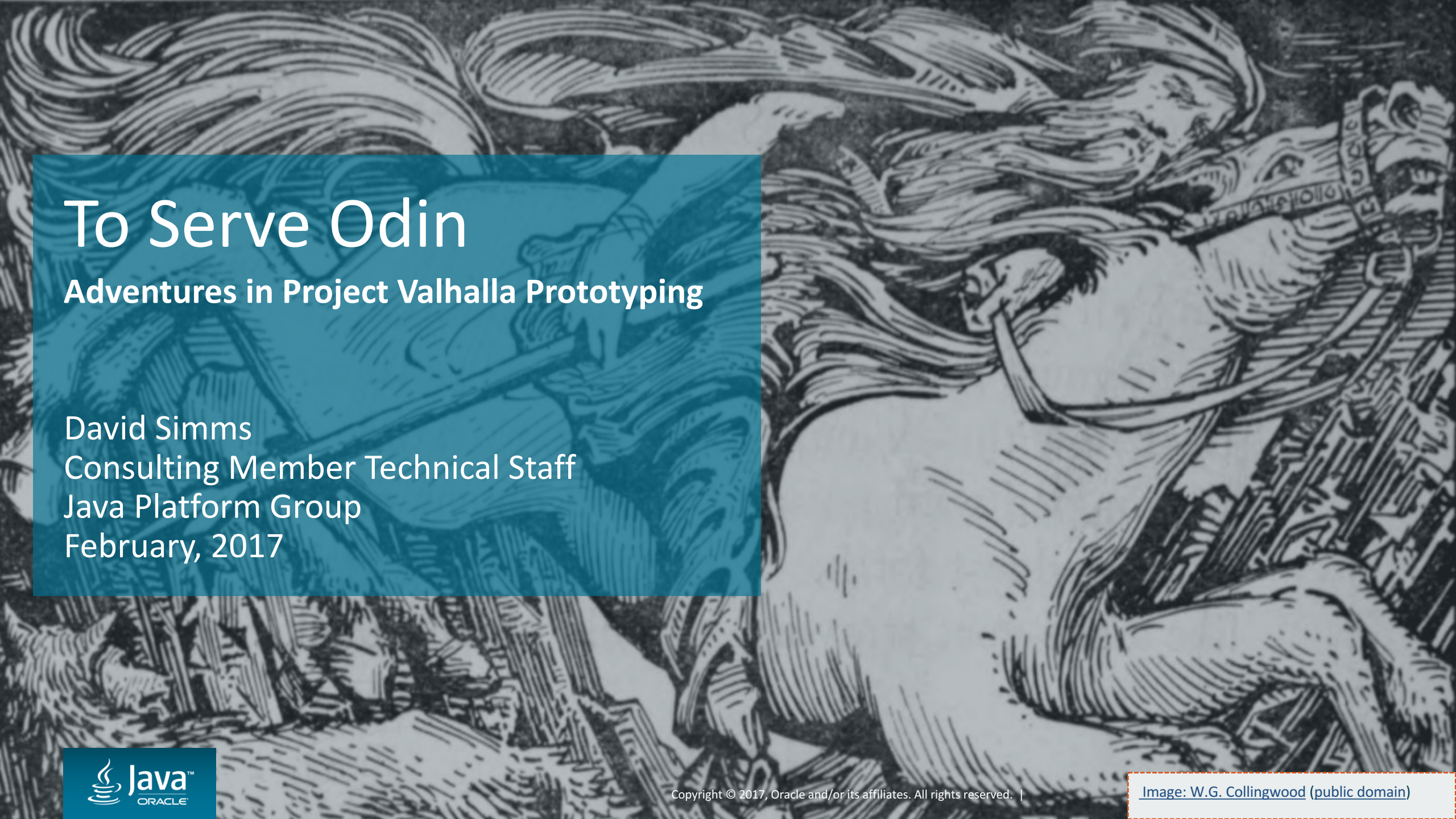

Java™
ORACLE®



To Serve Odin

Adventures in Project Valhalla Prototyping

David Simms
Consulting Member Technical Staff
Java Platform Group
February, 2017

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Image: W.G. Collingwood (public domain)

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.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Project Valhalla

Generic Specialization and Value Types

Image: David Simms (All rights reserved)

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Introduction

- Project Page: http://openjdk.java.net/projects/valhalla/
  - Links to mailing lists, repository(s), OpenJDK Wiki, Presentations etc...
- Brian Goetz: ”Adventures in Parametric Polymorphism” – JVMLS, Aug 2016
  - https://www.youtube.com/watch?v=Tc9vs_HFHVo
  - http://www.oracle.com/technetwork/java/jvmls2016-goetz-3126134.pdf

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Project Goals

Why…Three major goals

- Align JVM memory layout behavior with the cost model of modern hardware
- Extend generics to allow abstraction over all types, including primitives, values, and even void
- Enable existing libraries **especially the JDK** to compatibly evolve to fully take advantage of these features

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

"Valhalla may be motivated by performance considerations, but a better way to view it as enhancing abstraction, encapsulation, safety, expressiveness, and maintainability **without** giving up performance."

– Brian Goetz, Java Language Architect
– http://mail.openjdk.java.net/pipermail/valhalla-spec-experts/

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

A War on Two Main Fronts

What…

- Generic Specialization
  - Just say ’no’ to boxing: ”`java.util.Map<long, U>`”
- Value Types
  - Code your own primitive types, ”Codes like a class, works like an int”
  - Pure data, no identity, logically seen as ”pass by value”
  - No polymorphism
  - Immutable
  - Not nullable

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Generic Specialization

Model 3 Implementation

Image: David Simms (All rights reserved)

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Model 3

…and counting

- Truth is, we are probably up to ”model 9” at this point
  - Always end up back at ”model 3+some-variant” as far as implementation so far...
- Q: Why is this so hard ? A: Object Model still not done…
  - “`Foo<any U>`”, what is the top type, when “`U=int`” ?
  - “`Foo<int>, Foo<String>`” are these “`Foo`” ?
    - Common super type would be ? ”Any” interface
  - Migration and Compatibility
    - Do parameterized types need to box primitives when dealing with existing code ?
  - Nasty cases keep rearing their ugly heads

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Box of “Any”

```
public class Box<any T> {

    T t;

    public Box() {
    }

    public Box(T t) {
        this.t = t;
    }

    T get() { return t; }

    void set(T t) { this.t = t;}

}
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Current Implementation

Model 3 Specializer

- Mostly implemented in Java with the JDK, minor Hotspot VM changes
- At compile time: Javac is free to create a ”template class” with all manor of new prototype class file changes
  - bytecodes, constant pool forms, etc
  - Name mangling scheme: “`Foo<int>`” == “`Foo${I}`” for “specialized class”
- At run time: class load hook within the JVM up-call to the Model 3 Specializer
  - Responsible for converting prototype forms into ”legal” VM class file
  - Dump runtime class generation: “`-Dvalhalla.dumpProxyClasses=<dir>`”

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Realizing Parameterized Types

Constant Pool Specialization

- Transforming bytecode to accommodate all types, is too hard
  - Consider primitive types, value types and objects, bytecode syntax differs for each family.
  - E.g. “`anewarray`” vs “`newarray`”, “`if_acmpeq`” vs “`dcmp…if`”
- Introduced parameterization to the constant pool and specialize the pool
  - New to the constant pool: GenericClass, ParameterizedType, TypeVar, etc…
  - Consolidate required transforms to a few constant pool entries
- Instructions operating on parameterized types, use “generic bytecode” to be specialized later*

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Realizing Parameterized Types

Generic instructions

- **”typed”** bytecode
  - “typed <TypeVarIndex>”
- Prefixes “a” bytecodes, treating those as wildcard instructions
- Transformation when specializing to a concrete type
  - Specializing to int: “typed **<TypeVar>**; areturn” ➔ “nop; nop; nop; ireturn”
  - Attempt to keep the shape of the surrounding bytecode, preserve local/stack size
- *Actual transformation leave some part to the VM: => “typed **I**; areturn”
  - VM Experiment with multiple dispatch tables switched via “typed” bytecode

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Box of “Any” versus Box of int, Constant Pool

```
//class Box<any T>

#8 = TypeVar              // T/Ljava/lang/Obj
#11 = TypeVar             // T/_
#12 = ParameterizedType   // LBox<T/_>
#13 = Class               // "Box<T>"
#15 = NameAndType         // t:T
#16 = Fieldref            // "Box<T>".t:T
#24 = MethodDescriptor    // (T/Ljava/lang/Ob
#29 = MethodDescriptor    // ()T/Ljava/lang/O

LBox;:
  Tvar  Flags  Bound
  T     [ANY]  Ljava/lang/Object;
```

```
//class Box<int>

#8 = Utf8           I
#11 = Utf8          I
#12 = Utf8          LBox$${I};
#13 = Class         // "Box$${I}"
#15 = NameAndType   // t:I
#16 = Fieldref      // "Box$${I}".t:I
#24 = Utf8          (I)V
#29 = Utf8          ()I
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Box of “Any” versus Box of int, Bytecode

```
//class Box<any T>.set(T);

public void set(T);
    descriptor: (TT;)V
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=2, args_size=2
         0: aload_0
         1: typed   // T/Ljava/lang/Object;
         4: aload_1
         5: putfield // Field t:T
         8: return
      LineNumberTable:
        line 9: 0
    Signature: #40
// (TT;)V
```

```
//class Box<any T>.set(T);

public void set(T);
    descriptor: (I)V
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=2, args_size=2
         0: aload_0
         1: nop
         2: nop
         3: nop
         4: iload_1
         5: putfield  // Field t:I
         8: return
      LineNumberTable:
        line 9: 0
    Signature: #40
// (TT;)V
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

“Any” interface and Arrays 2.0

Type injection for kicks

- Model 3 experiment with common super type “any interface”: “Foo<any T>” = “Foo<any>”
- Experiment with Arrays 2.0 ideas, all arrays implement “Arrayish<any T>”
  - Which is itself a generic “any” type…
  - …which needs specialization…
  - …at VM boot time, before the runtime specializer can be run
- Interface dispatch via “extra super” feature
  - VM allowed an extra type to be injected into klass at runtime, kind of a “trait”
- Probably won’t survive ☺

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Arrayish

```
public interface Arrayish<any T> {

    default int arraySize();

    default T arrayGet(int index);

    default void arraySet(int index, T element);
}
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Nestmates

One compilation unit, many “crass”, aren’t we in the same class, can I see your bits ?

- Specialization creates multiple types at runtime from a single class
  - “crass” has been a term thrown around
- Difference in Java versus VM access rules
  - Javac today generates bridges to enable access for inner/outer classes
  - Not specific to Valhalla, moving out into JDK10
  - http://openjdk.java.net/jeps/181
- Allow class file to describe “nest members” (Valhalla prototype attribute)

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Value Types

A bucket of bytecodes

Image: David Simms (All rights reserved)

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Point Value

```
public __ByValue final class Point {

    final int x;
    final int y;

    private Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() { return x; }
    public int getY() { return y; }

    public boolean isSamePoint(Point that) {
        return this.getX() == that.getX() && this.getY() == that.getY();
    }

    public String toString() {
        return "Point: x=" + getX() + " y=" + getY();
    }

    public static Point createPoint(int x, int y) {
        return __Make Point(x, y);
    }
}
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Value Types

Repository State, what is working today

- Q Type descriptors
- “Level-0” prototype bytecodes
  - Prototype Javac support, you can take it out for a drive
- Flattened all the things: arrays, fields, compositions
- Interpreter
  - Calls to C++ code instead of assembler or code generation
- Initial C2 support
- Naïve heap allocation

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Value Types

Repository State, what is not working just yet

- No verifier support, must run with "`-noverify`"
- X86_64 only
- No optimization, no C1
- Primitive fields only
- Object Model – still a can of worms
  - Common super type ? Common descriptor ?
  - Implement Interface ?
  - Boxed values passed through pre-existing code, what is "synchronized(boxedValue)" ?

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

bytecodes.hpp

```
_vload                  = 203, // 0xcb
_vstore                 = 204, // 0xcc
_vaload                 = 205, // 0xcd
_vastore                = 206, // 0xce
_vnew                   = 207, // 0xcf
_vreturn                = 210, // 0xd2
_vgetfield              = 211, // 0xd3
_typed                  = 212, // 0xd4
_invokedirect           = 213, // 0xd5
_vdefault               = 214, // 0xd6
_vwithfield             = 215, // 0xd7
_vbox                   = 216, // 0xd6
_vunbox                 = 217, // 0xd7

// JVM Internal…
_fast_qgetfield,
_fast_qputfield,
                                        // 15 value-type bytecodes, yippie…
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Value Bytecodes

Some points of interest

- "`vnew`" – offers atomic construction
  - Address current issues with "`new`" and "`invoke init()`", needs all args on stack
- "`vdefault`" – offers simple "all fields are zero" construction
  - Provides some efficiency compare to "`vnew`"
- "`vwithfield`" – C.O.W. field setter
  - Combine with "`vdefault`" to help code patterns like "`p.x += 3`"
- "`invokedirect`" – monomorphic method invocation
- Are not atomic by default*
  - `vaload, vastore, qgetfield, qputfield`

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Current Work

...and what's next

Image: David Simms (All rights reserved)

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

“typed” Bytecode Experiments

Current Work

- Rename a2b, semantically similar “MH.asType()”
- Replace most of the “v” bytecodes
  - since there are so few free bytecodes
- “extended bytecodes”
  - allow “typed” to switch “BCSet”
  - allows alias existing bytecode
    - Consider “Point”, its 2 int fields can be aliased as a long
    - E.g. “typed Point; aload_0” ➔ “nop; nop; nop; lload_0”

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Thread Local Buffering

Current Work

- Remove naïve heap allocation, reduce GC pressure in general
- Thread local value buffer pages
- Interpreter frame activation records its current
- Simple push/pop model on frame entry/return
- Spill to heap
- Investigating cost of TLGC
  - ‘Cause “jmp” (why couldn’t Java bytecode rid us of “goto” ?!)
  - Looks at Stack/LVT entries, copy those, toss everything else

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Reference Fields in Values

Current Work

- Re-enable embedding oops
- oopmap generation is already there
  - will need further adjustment with value thread local buffering, expose inner oop refs
  - Conditional allocation, could be heap oop, could be value buffer.
- Some further GC barrier considerations
  - GC write barriers don't care if destination is a buffer, so we are "mostly good"
  - Avoid root scan pollution, live Stack/LVT entries (common for TLGC of value buffers)
  - Ensure klass mirror referenced / unreferenced appropriately
- Refactor "`ValueKlass::value_store()`" et al.

Java™ ORACLE

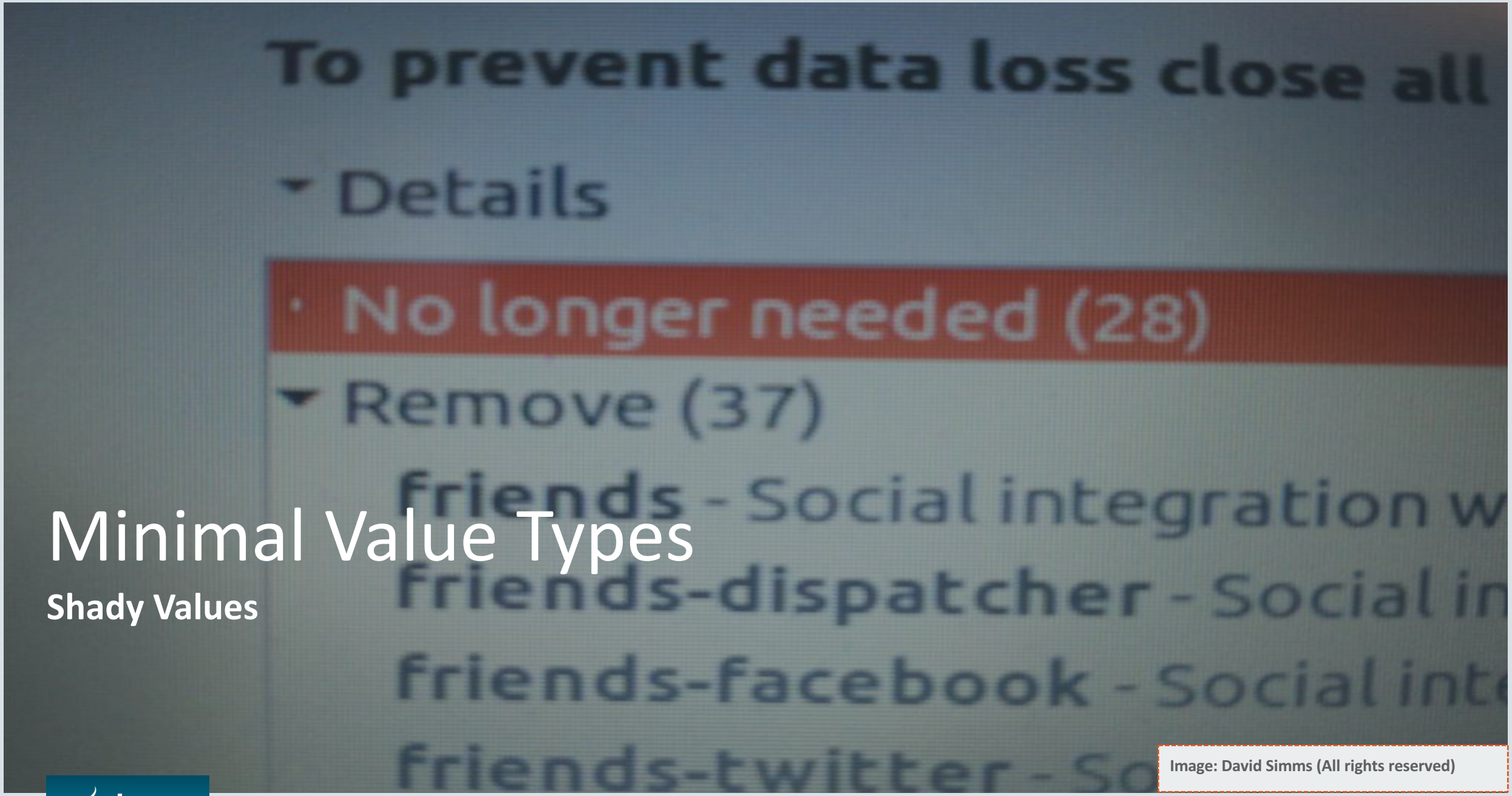
Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Current Work

- C2
  - Optimizations
  - Integration with value buffering: i2c, deoptimization, etc
- Verifier sanity check
  - Look for obvious holes in the current byte-code, but deferring implementation
- Rebasing to JDK10
  - Last sync to JDK9, was when ? “Merge Bankruptcy”
  - Some of ideas won’t be coming

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |



Minimal Value Types

Shady Values

Image: David Simms (All rights reserved)

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Minimal Value Types

A path to release

- Up to 15 new byte codes + unfinished type system = shipping with JDK10 ?
  - Ah, yeah, no. Doh !
  - Changes to JLS and JVMS need to be pretty much set in stone…
  - …and the path for getting that done is non-trivial in itself
- Folks have waited a long time, and you have something that walks at least
  - True, and we want to get those bits out the door…

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

How do you release a specification compliant JDK/JVM then ?

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Shady Values

Keep it in the shadows…

- The answer to quicken the path to release: make no specification changes
- Hide all the new JVM toys under MethodHandles
  - All new bytecodes become JVM internal, not visible to specification
  - Provide an experimental API that returns a method handle for value type operations
  - Gives end-users enough to play with
- Language: Runtime Annotation `@jvm.internal.value.DeriveValueType`
  - Value Capable Class (VCC), follows same rules as “value-based” classes
- http://cr.openjdk.java.net/~jrose/values/shady-values.html

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Point Value Capable Class

```
@jvm.internal.value.DeriveValueType
public final class Point {

    final int x;
    final int y;

    private Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public int getX() { return x; }
    public int getY() { return y; }
}
```

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Derive Value Type (DVT)

Encountering the `DeriveValueType` annotation

- JVM may or may not validate the VCC
  - Similar rules to Value Types and “value-based classes”
- Derive’s a structurally equivalent value type (DVT) which is field compatible with the VCC (Point$Value)
- User can query an API if there is a DVT class associated with VCC
  - j.l.i.MethodHandles support DVT/array classes
- Further API for obtaining and using the DVT, and boxing via VCC
- Supports data only, method invocation limited to VCC
  - Current limitations: primitives only, no composition (MVT1.0)

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

jdk.experimental.value.ValueType API

```
public class ValueType<T> {

    // Query methods...
    static boolean classHasValueType(Class<?> x);
    static <T> ValueType<T> forClass(Class<T> x);

    // Class/type query...Useful for j.l.i.MethodHandles
    Class<T> boxClass();
    Class<?> sourceClass();
    Class<?> valueClass();
    Class<?> arrayValueClass();
    Class<?> arrayValueClass(int dims);

    // Operations…next slide…
```

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

jdk.experimental.value.ValueType API

```
    // Operations…

    MethodHandle defaultValueConstant();
    MethodHandle substitutabilityTest();
    MethodHandle substitutabilityHashCode();
    MethodHandle findWither(String name, Class<?> type);
    MethodHandle unbox();
    MethodHandle box();
    MethodHandle newArray();
    MethodHandle arrayGetter();
    MethodHandle arraySetter();
    MethodHandle newMultiArray(int dims);

}
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Example: Flat array storage…

```
Class<?> VCC = Point.class;
MethodHandles.Lookup lookup = MethodHandles.lookup();

ValueType<?> VT = ValueType.forClass(VCC);
Class<?> vTArrayClass = VT.arrayValueClass();
MethodHandle setMh = MethodHandles.arrayElementSetter(vTArrayClass);
MethodHandle getMh = MethodHandles.arrayElementGetter(vTArrayClass);

//Setup an array...it will be flattened, important to me here...
int arrSize = w * h;
Object arr = MethodHandles.arrayConstructor(vTArrayClass).invoke(arrSize);
for (int i = 0 ; i < arrSize; i++) {
  // Construct VCC, arraySetter will unbox for me...
  Point p = new Point(i, 0);
  setMh.invoke(arr, i, p);
}
// Give me a point...
Point apoint = getMh.invoke(arr, (w-1));
```

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

MVT 1.1

Moving forward, stretch goals…

- Open up experimental access to “Valhalla Value Types” (VVT)
  - ability to dynamically generate byte codes which refer directly to QTypes
  - allow methods that operate directly on the value type
- Experiment layout optimizations
- Allow composition / references
- Explore “typed acmp”
- Common descriptor, “P-Type” or “U-Type” experiments

Java™ ORACLE

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Safe Harbor Statement

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.

Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |


Java™
ORACLE®