

# Valhalla is coming

**Sergey Kuksenko**

Java Platform Group  
Oracle  
June, 2020

## Safe Harbor

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, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Statements in this presentation relating to Oracle's future plans, expectations, beliefs, intentions and prospects are "forward-looking statements" and are subject to material risks and uncertainties. A detailed discussion of these factors and other risks that affect our business is contained in Oracle's Securities and Exchange Commission (SEC) filings, including our most recent reports on Form 10-K and Form 10-Q under the heading "Risk Factors." These filings are available on the SEC's website or on Oracle's website at <http://www.oracle.com/investor>. All information in this presentation is current as of September 2019 and Oracle undertakes no duty to update any statement in light of new information or future events.

# Who am I?

- Java/JVM Performance Engineer at Oracle, @since 2010
- Java/JVM Performance Engineer, @since 2005
- Java/JVM Engineer, @since 1996

# Preface

# 3 Key Principles of Java Evolution

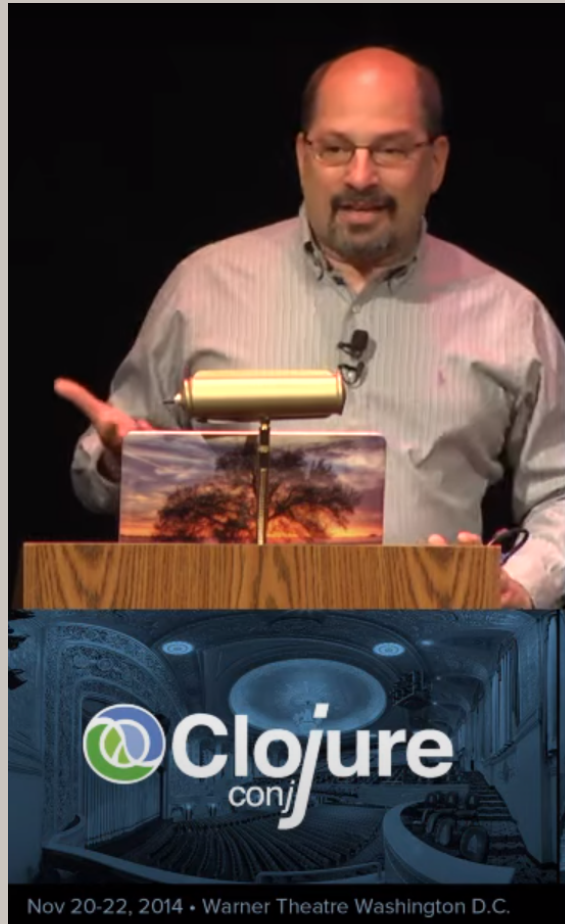
- Compatibility
- Compatibility
- Compatibility

- *If we don't know how to do it right, we won't do it (now).*
- *It's better to leave out something good than to put in something bad. (Because compatibility is forever)*
- *Less is more — The first Java “buzzword” is SIMPLE*

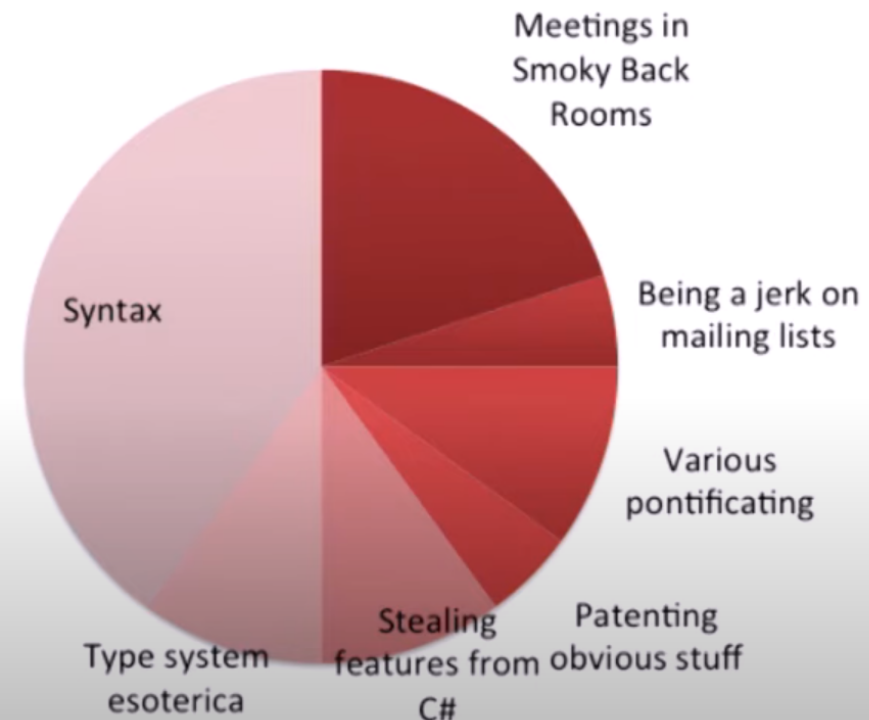
(c) Brian Goetz

# Java Evolution

## Brian Goetz: "Stewardship: the Sobering Parts"



What people think I do (naïve version)



# What is Valhalla?

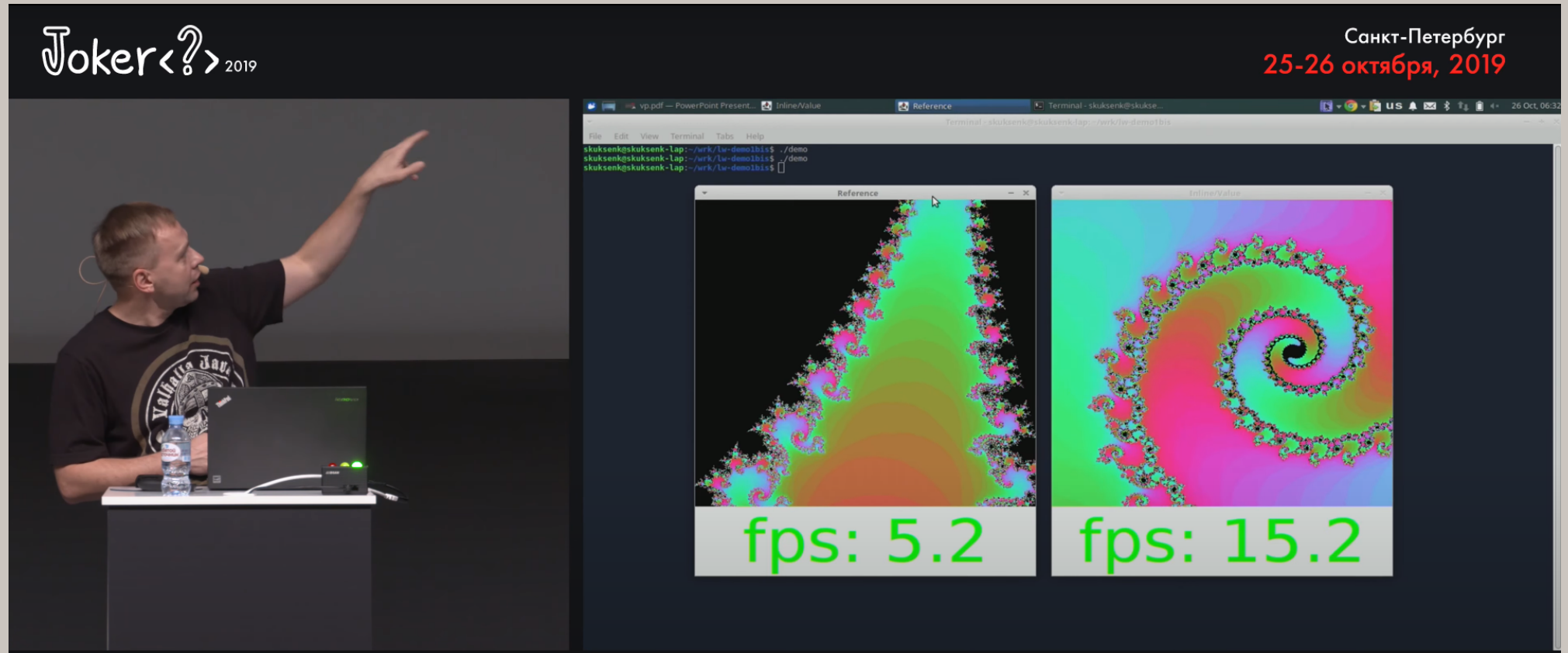


# What is Valhalla?

"Does Java need "inline" types?"

Joker<?> 2019

Санкт-Петербург  
25-26 октября, 2019



The screen shows a terminal window with the following commands and output:

```
skuksen@skuksen-lap:~/wrk/la-demo/ibis$ ./demo
skuksen@skuksen-lap:~/wrk/la-demo/ibis$ ./demo
skuksen@skuksen-lap:~/wrk/la-demo/ibis$
```

Two windows are open, each displaying a fractal image:


- The left window, titled "Reference", shows a fractal image with a frame rate of **fps: 5.2**.
- The right window, titled "InlineValue", shows the same fractal image with a frame rate of **fps: 15.2**.

# What is Valhalla?

Remi Forax: "The sinuous path toward Valhalla"

Joker<?> 2019

Санкт-Петербург  
25-26 октября, 2019



## Benchmark with JMH

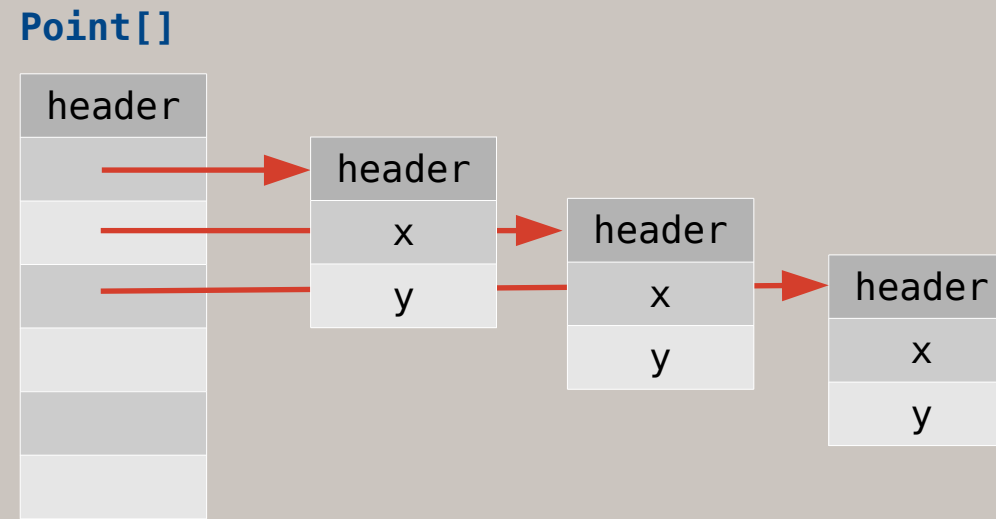
Time to render the Mandelbrot 64x64 with 1000 iterations?

	score	error
primitive	1540 $\mu$ s/op	22 $\mu$ s/op
indirect	2816 $\mu$ s/op	48 $\mu$ s/op
inline	1541 $\mu$ s/op	12 $\mu$ s/op

With `java -XX:+EnableValhalla`

# Inefficiency: references are everywhere

```
class Point {...}
```



# Solution: Inline types

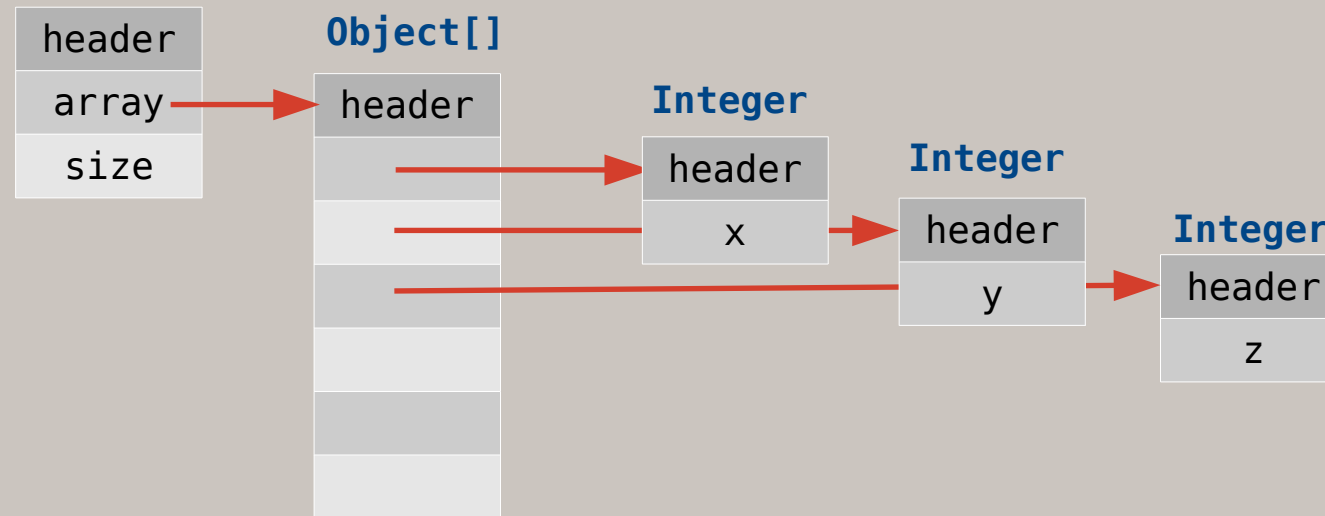
```
inline class Point {...}
```

**Point[]**

header
x
y
x
y
x
y

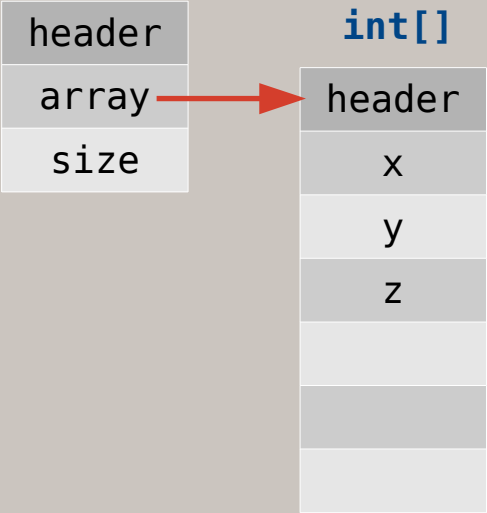
# Inefficiency: boxes are everywhere

`ArrayList<Integer>`



# Solution: Specialized generics

```
ArrayList<int>
```



# Valhalla:

- Inline types
- Specialized generics
- Migration

# Identity, Value objects and other theories.



# Identity

## Identity (object-oriented programming)

From Wikipedia, the free encyclopedia

An **identity** in [object-oriented programming](#), [object-oriented design](#) and [object-oriented analysis](#) describes the property of [objects](#) that distinguishes them from other objects. This is closely related to the philosophical concept of [identity](#).

In philosophy, **identity**, from Latin: *identitas* ("sameness"), is the relation each thing bears only to itself. The notion of identity gives rise to many philosophical problems, including the identity of indiscernibles, and questions about change and personal identity over time.

# Identity

The idea is the following: in a model with object identity, an object has an existence which is independent of its value. Thus two notions of object equivalence exist: two objects can be identical (they are the same object) or they can be equal (they have the same value). This has two implications: one is object sharing and the other one is object updates.

# Value object

## Value object

---

From Wikipedia, the free encyclopedia

In **computer science**, a **value object** is a small **object** that represents a *simple* entity whose equality is not based on identity: i.e. two value objects are *equal* when they *have* the same *value*, not necessarily being the *same object*.<sup>[1][2]</sup>

## ValueObject

14 November 2016



Martin Fowler

<https://martinfowler.com/bliki/ValueObject.html>

# Identity and mutability

Identity enables mutability<sup>(\*)</sup>

*(\*) in order to mutate a field of an object,  
we must know which object we are trying to modify*

# Value object and mutability

<http://wiki.c2.com/?ValueObject>

<http://wiki.c2.com/?ValueObjectHypotheses>

<http://wiki.c2.com/?ValueObjectsCanBeMutable>

<http://wiki.c2.com/?ValueObjectsShouldBeImmutable>

<http://wiki.c2.com/?CanValueObjectsContainReferenceObjects>

# Value object and mutability

<http://wiki.c2.com/?ValueObject>

**C#**



<http://wiki.c2.com/?ValueObjectHypotheses>

<http://wiki.c2.com/?ValueObjectsCanBeMutable>

**Java**



<http://wiki.c2.com/?ValueObjectsShouldBeImmutable>

<http://wiki.c2.com/?CanValueObjectsContainReferenceObjects>

# Identity

The “simplest” identity implementation –  
reference/pointer/address

# Identity gives:

- Extra value

**null**

- Extra attributes

*“each object in Java is associated with a monitor”* (c) JLS

- Performance

e.g. `(x==y || x.equals(y))`



# Identity gives:

- **Not always working optimizations**  
e.g. Escape analysis + scalar replacement
- **Not optimal optimizations**  
e.g. `-XX:+UseStringDeduplication`
- **Extra overhead to preserve identity**  
e.g. ZGC/Shenandoah read barriers

# Does Java already has Value Objects?

# Does Java already has Value Objects<sup>(\*)</sup>?

- `java.lang.String`
- `java.lang.Integer`
- ...

*(\*) - theoretically we may deprive these classes of identity*

# Does Java already has Value Objects?

## Value-based Classes

Some classes, such as `java.util.Optional` and `java.time.LocalDateTime`, are *value-based*.  
Instances of a value-based class:

- are final and immutable (though may contain references to mutable objects);
- have implementations of `equals`, `hashCode`, and `toString` which are computed solely from the instance's state and not from its identity or the state of any other object or variable;
- make no use of identity-sensitive operations such as reference equality (`==`) between instances, identity hash code of instances, or synchronization on an instances's intrinsic lock;
- are considered equal solely based on `equals()`, not based on reference equality (`==`);
- do not have accessible constructors, but are instead instantiated through factory methods which make no committment as to the identity of returned instances;
- are *freely substitutable* when equal, meaning that interchanging any two instances `x` and `y` that are equal according to `equals()` in any computation or method invocation should produce no visible change in behavior.

# Does Java already has Value Objects?

*This implies that the identity of the result of evaluating a lambda expression (or, of serializing and deserializing a lambda expression) is unpredictable, and therefore identity-sensitive operations (such as reference equality ([§15.21.3](#)), object locking ([§14.19](#)), and the `System.identityHashCode` method) may produce different results in different implementations of the Java programming language, or even upon different lambda expression evaluations in the same implementation.*

# What is Inline class?

# Inline Class

```
public inline class Complex {  
  
    private double re, im;  
  
    public Complex(double re, double im) {  
        this.re = re;  
        this.im = im;  
    }  
  
    public double re() { ... }  
    public double im() { ... }  
    public Complex add(Complex c) { ... }  
    public Complex mult(Complex c) { ... }  
}
```

# Inline Class

```
public inline class Complex {  
    private double re, im;  
  
    public Complex(double re, double im) {  
        this.re = re;  
        this.im = im;  
    }  
  
    public double re() { ... }  
    public double im() { ... }  
    public Complex add(Complex c) { ... }  
    public Complex mult(Complex c) { ... }  
}
```

No Identity



# Inline class

- is a class
- no identity → immutable, non-nullable
- that's all

## We didn't say:

- it's allocated on heap or out of heap
- it's allocated on stack or out of stack
- it's inlined into container object or referenced by container object

# Identity-sensitive operations

- Synchronization (`Object::wait`, `Object::notify`)

```
throw new IllegalMonitorStateException();
```

# Identity-sensitive operations

- Equality (==), substitutability check

a == b

- a & b has the same type
- for each field:
  - if inline class – check substitutability
  - if double/float – check Double/Float::equals
  - else – check “==”

# Identity-sensitive operations

- `System::identityHashCode`
  - compute over fields

# Identity-sensitive operations

- Weak references

JDK-8244968 [valhalla] WeakHashMap needs to handle keys which are inline objects

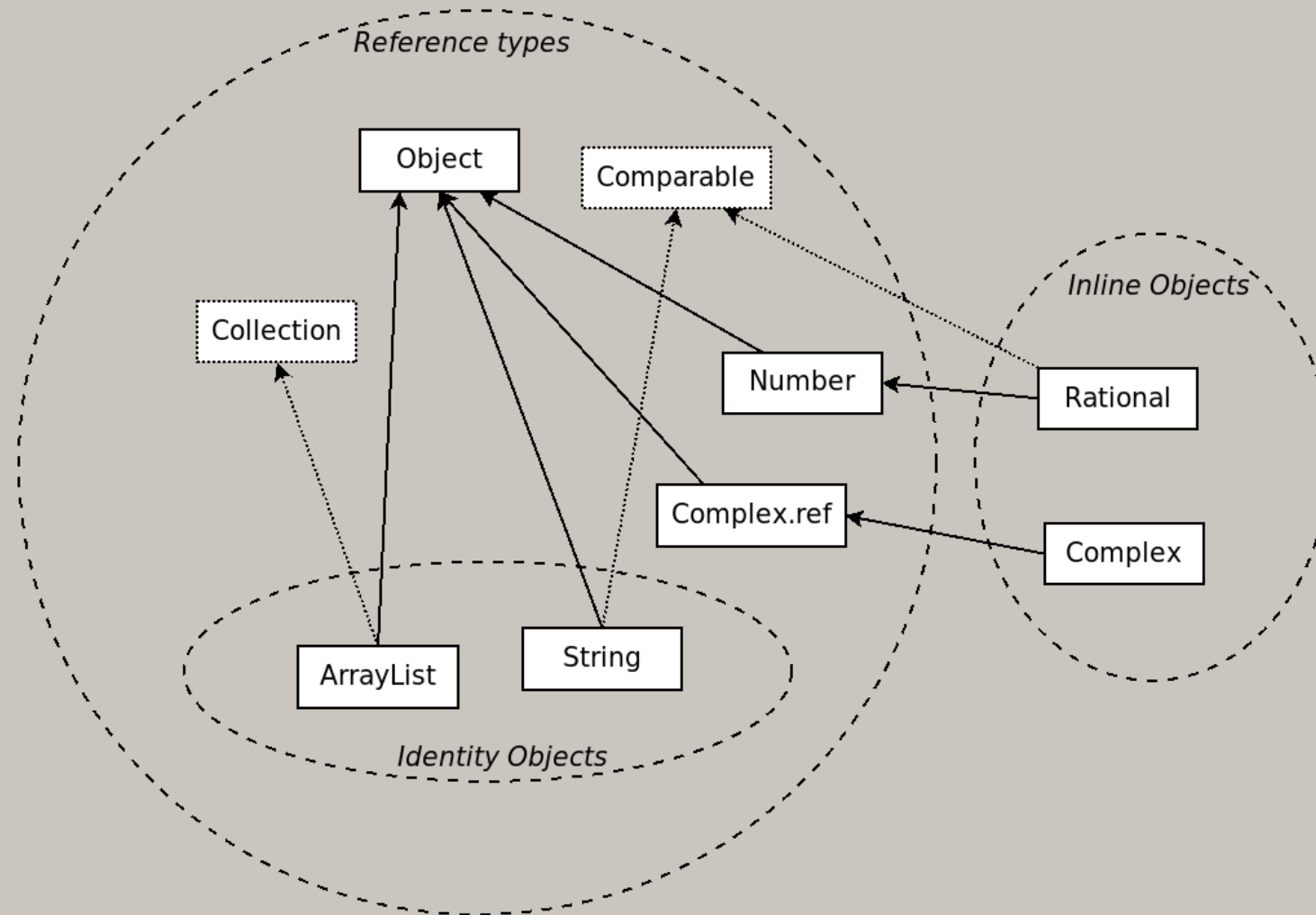
# Class hierarchy

# Inline class

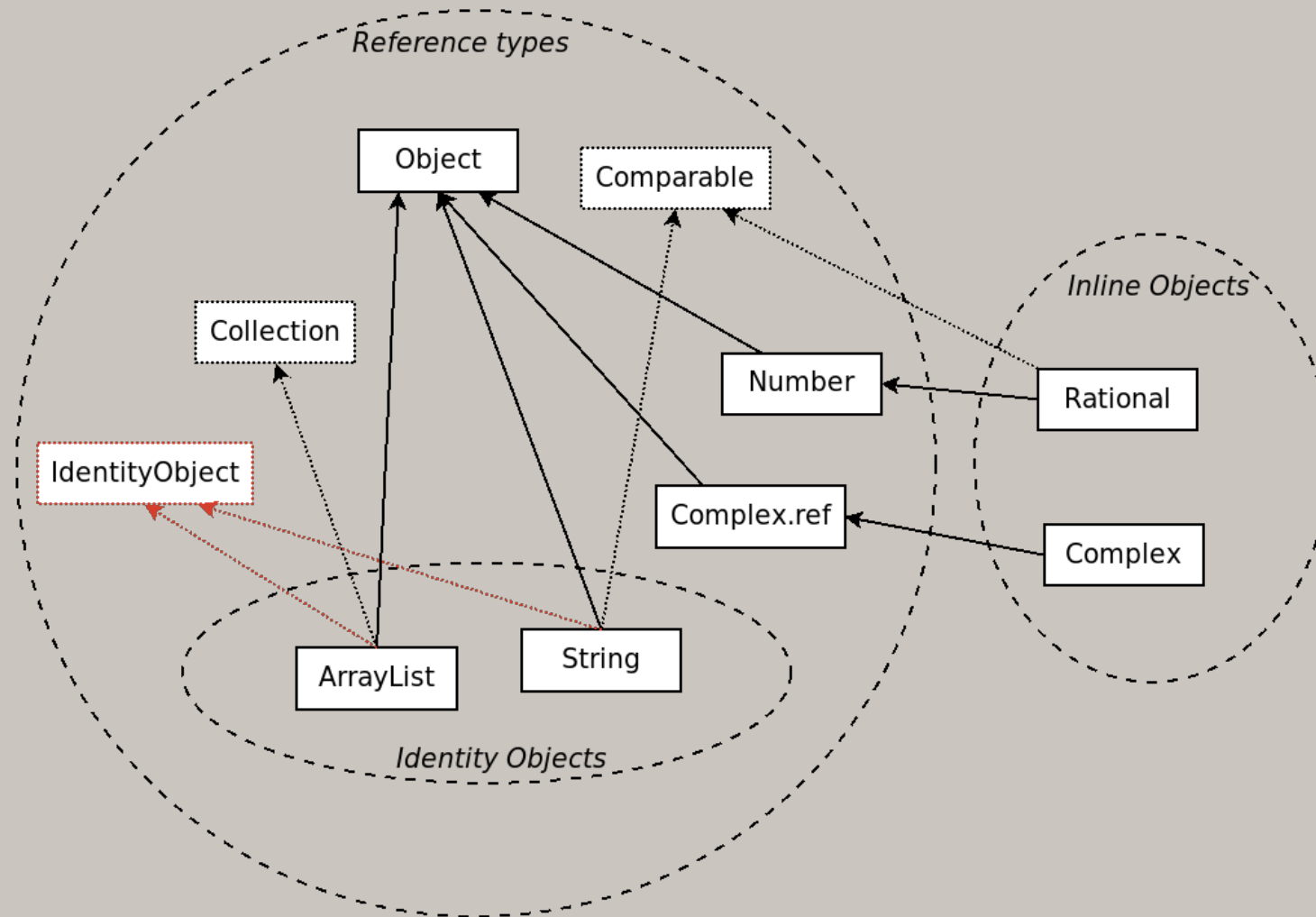
- is final
- can implement interfaces
- extends “well-formed” abstract class:
  - no fields
  - empty no-arg constructor
  - no synchronized methods
  - has “well-formed” super



# Class hierarchy



# Class hierarchy



# IdentityObject

```
if (x instanceof IdentityObject) {  
    synchronized(x) { ... }  
}
```

...

```
void runWithLock(IdentityObject lock, Runnable r) {  
    synchronized (lock) {  
        r.run();  
    }  
}
```

Wait, wait!

What about “new Object()”?

# Wait, wait!

What about “new Object()”?

@Deprecated



# Wait, wait!

## What about “new Object()”?

@Deprecated



```
class java.util.Objects {  
    /**  
     * Constructs a new Object implementing the IdentityObject interface.  
     * The object is a unique IdentityObject suitable for all purposes  
     * that previously for which new Object() was used including  
     * synchronization, mutexes and unique placeholders.  
     *  
     * @return a new Object implementing the IdentityObject interface  
     * @since Valhalla  
     */  
    public IdentityObject newIdentity()  
    ...  
}
```

# What about arrays?

The same as before:

`X <: I <: Object`



`X[] <: I[] <: Object[]`

# Boxing VS Inline Widening Conversion



# Value Set (pre Valhalla)

primitive type → primitive values

reference type → null + references to object instances

# Value Set (Valhalla)

primitive type → primitive values

inline type → object instances

identity type → null + references to object instances

reference type → null + ???

# Value Set (Valhalla)

primitive type → primitive values

inline type → object instances

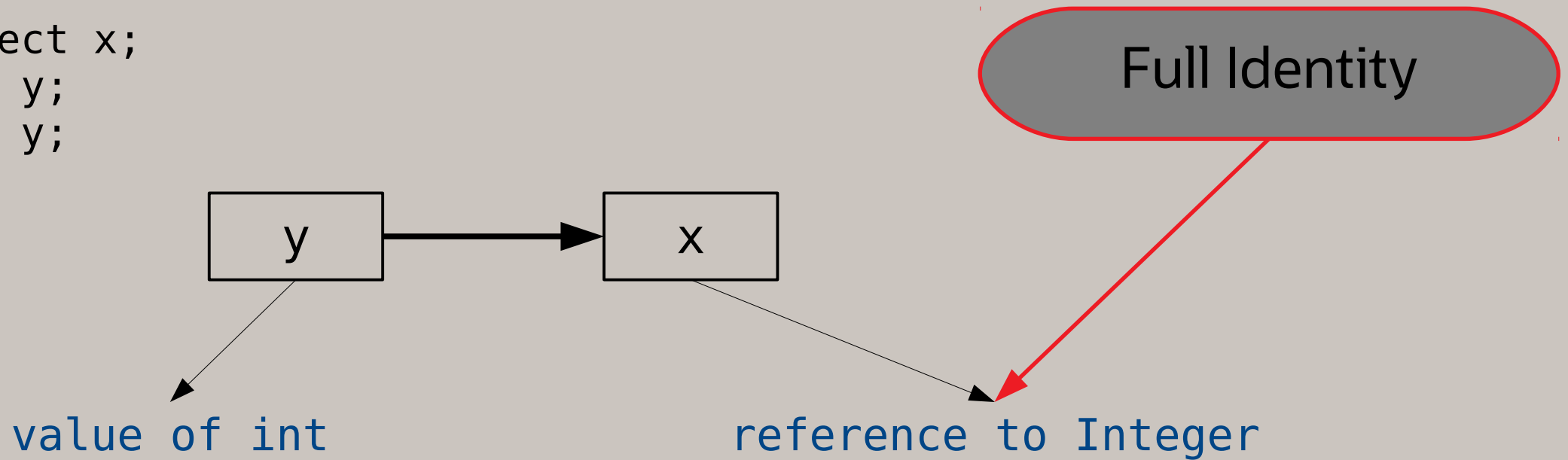
identity type → null + references to object instances

reference type → null + references to

both identity and inline objects

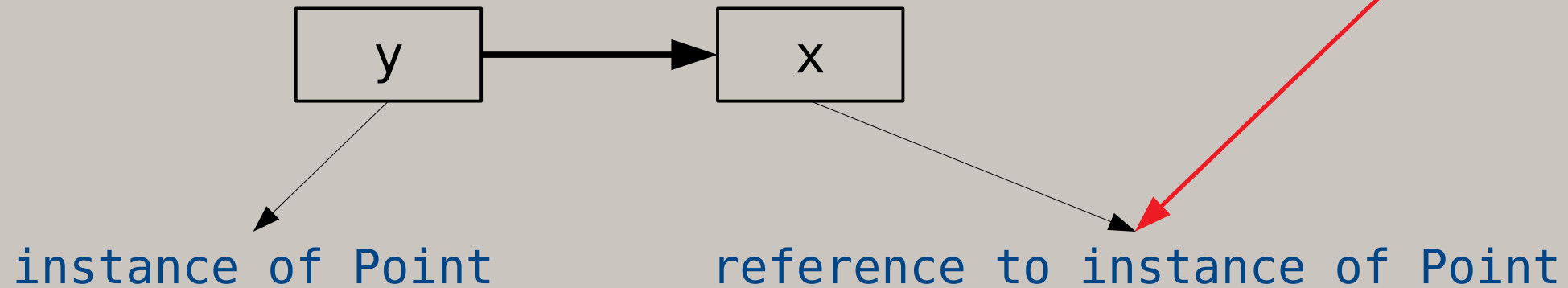
# Boxing

```
Object x;  
int y;  
x = y;
```



# Inline widening

```
Object x;  
Point y;  
x = y;
```



# No more questions

# Reference projection

T – inline type

T . ref – reference projection for T

$\text{ValSet}(T . \text{ref}) = \{ \text{null} \} \cup \{ \mathbf{ref} \ v : v \in \text{ValSet}(T) \}$

# Reference projection

```
inline class V {...}
```

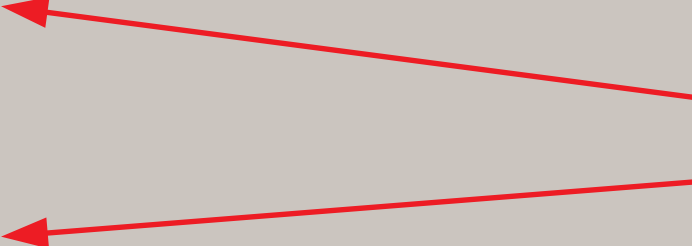
You wrote



---

```
sealed abstract class V.ref permits V.val { }
```

We generated



```
inline class V.val extends V.ref {...}
```



# Reference projection

$T$  – inline type

$T.ref$  – reference projection for  $T$

$T.val$  – value projection for  $T$

$T.ref \rightarrow T.val$  – *inline narrowing conversion*

# Reference projection

S – reference type

S.ref == S

# Default value

# Default value

T – inline type

T.default – default value of T

for each field:

-- 0, 0.0, false, default or null

# Default value

S – reference type

`S.default == null`

# Implementation details

# If you (your JVM) are very lazy

- Allocate all inline classes in heap (as other objects)
- Implement all identity-sensitive operations in a proper way
- Don't forget to make inline types non-nullable
- Some boring stuff with classloading and verification

# What about Hotspot?

- Sometimes allocate inline classes in heap
- Inline it into containers (arrays, classes)
- Locals and parameters - scalarized



# Migration stories

# One migration story

```
Map<K, V>
```

```
V get(Object key);
```

*Returns the value to which the specified key is mapped, or **null** if this map contains no mapping for the key.*

# One migration story

```
Map<K, V>
```

```
V get(Object key);
```

*Returns the value to which the specified key is mapped, or **null** if this map contains no mapping for the key.*



If V – inline?

# One migration story

Map<K, V>

V.ref get(Object key);

*Returns the value to which the specified key is mapped, or **null** if this map contains no mapping for the key.*

# Another migration story

## Optional<T>

*API Note:*

*Optional is primarily intended for use as a method return type where there is a clear need to represent "no result," and where using null is likely to cause errors. A variable whose type is Optional should never itself be null; it should always point to an Optional instance.*

# Another migration story

`Optional<T>` - will be inline

but:

```
Optional<T> op;
```

```
..
```

```
op = null;
```

Can't be inline



# Another migration story

```
inline class V {...}
```

```
V x; ... foo(V a); ... T<V> ...; // usages of V  
// V – just an alias
```

---

```
sealed abstract class V.ref permits V.val { }
```

```
inline class V.val extends V.ref {...}
```

# Another migration story

```
ref-default inline class V {...}
```

```
V x; ... foo(V a); ... T<V> ...; // usages of V  
// V – just an alias
```

---

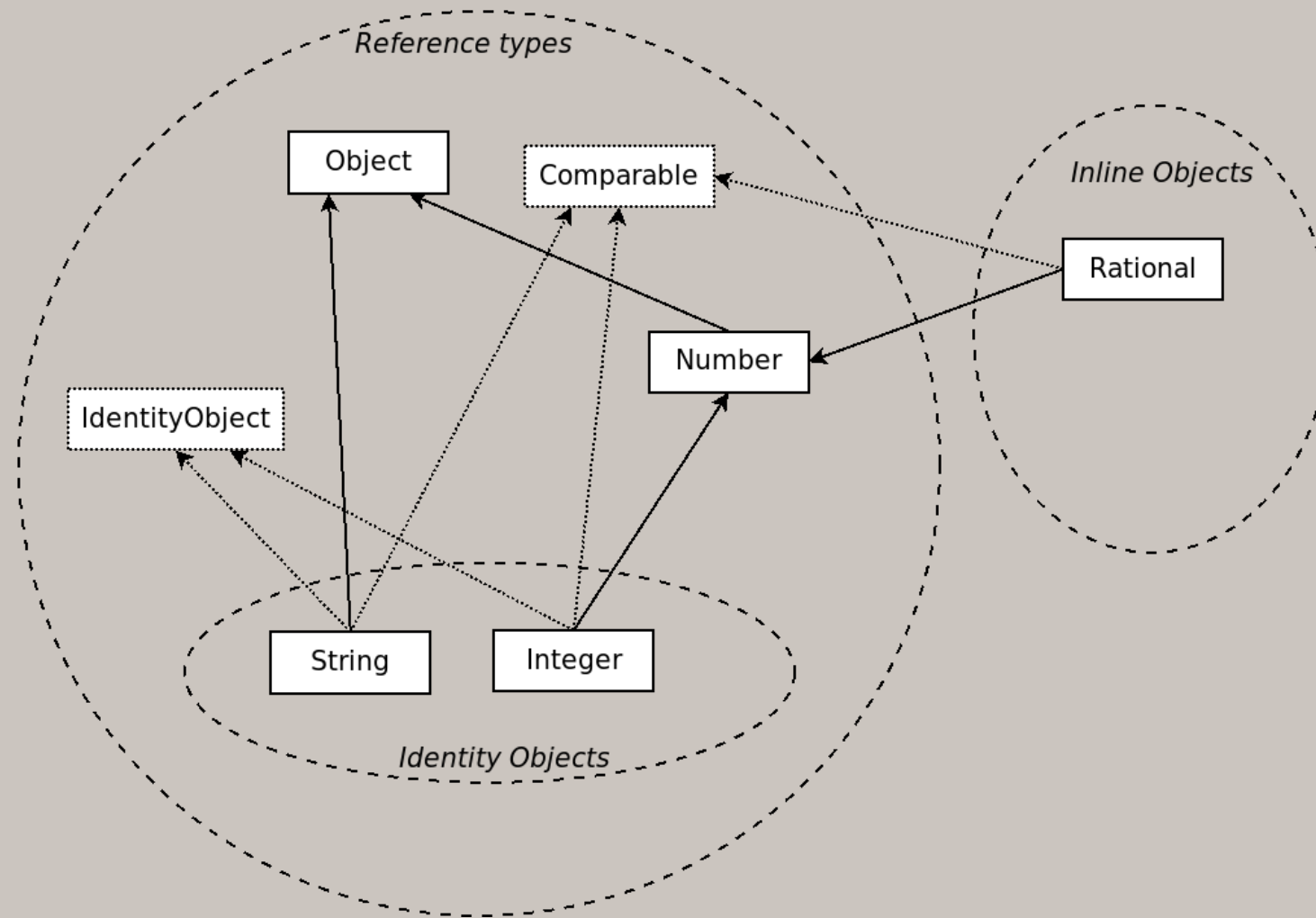
```
sealed abstract class V.ref permits V.val { }
```

```
inline class V.val extends V.ref {...}
```



# Very primitive migration story

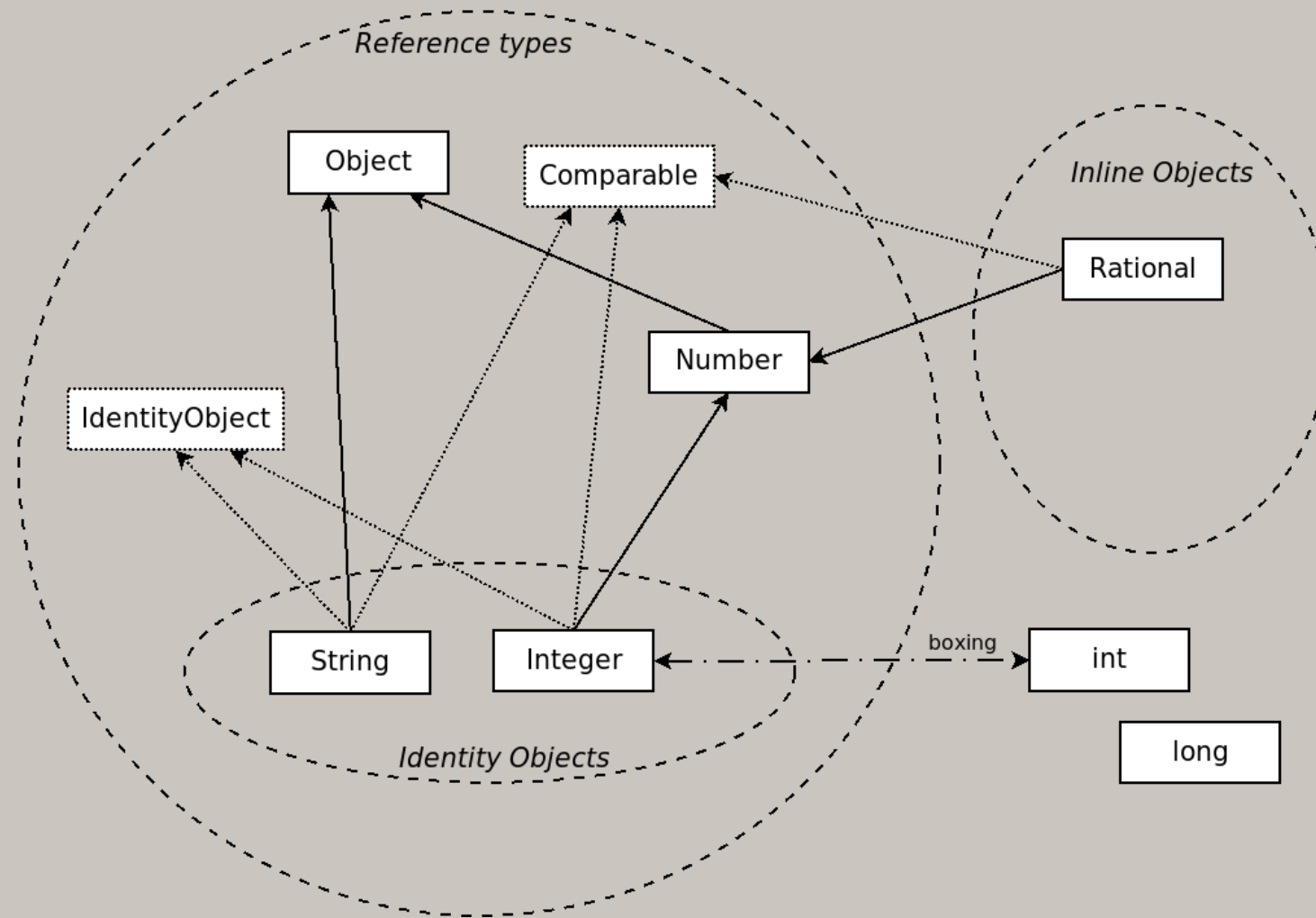
# Class hierarchy



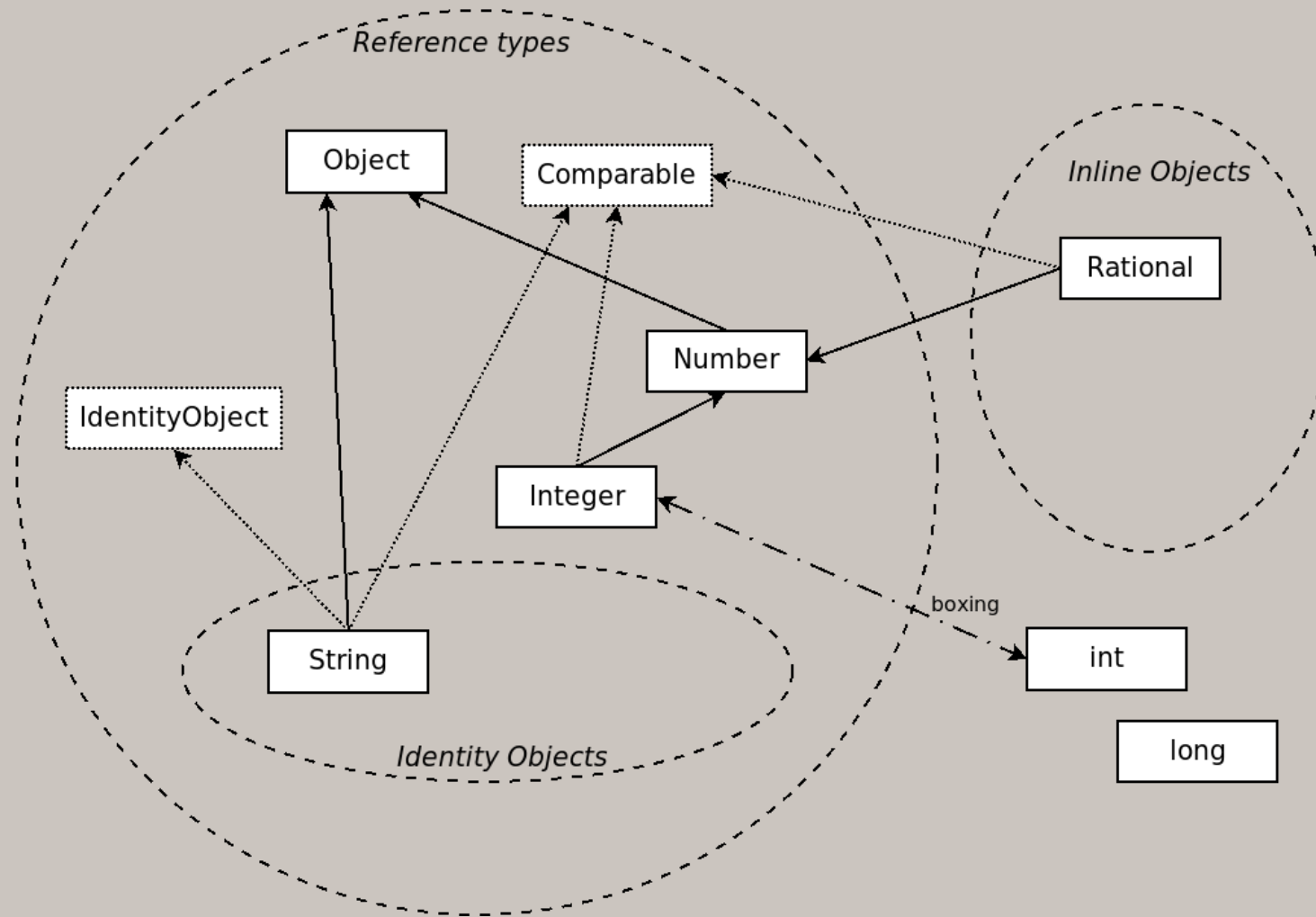
# We forgot primitives

- Not a true OOP
- Incompatible arrays, no generics
- Complex code:
  - e.g. `Arrays::fill` – 9 methods
- Complex libraries:
  - e.g. `Stream<T>`, `IntStream`, `LongStream`, `DoubleStream`

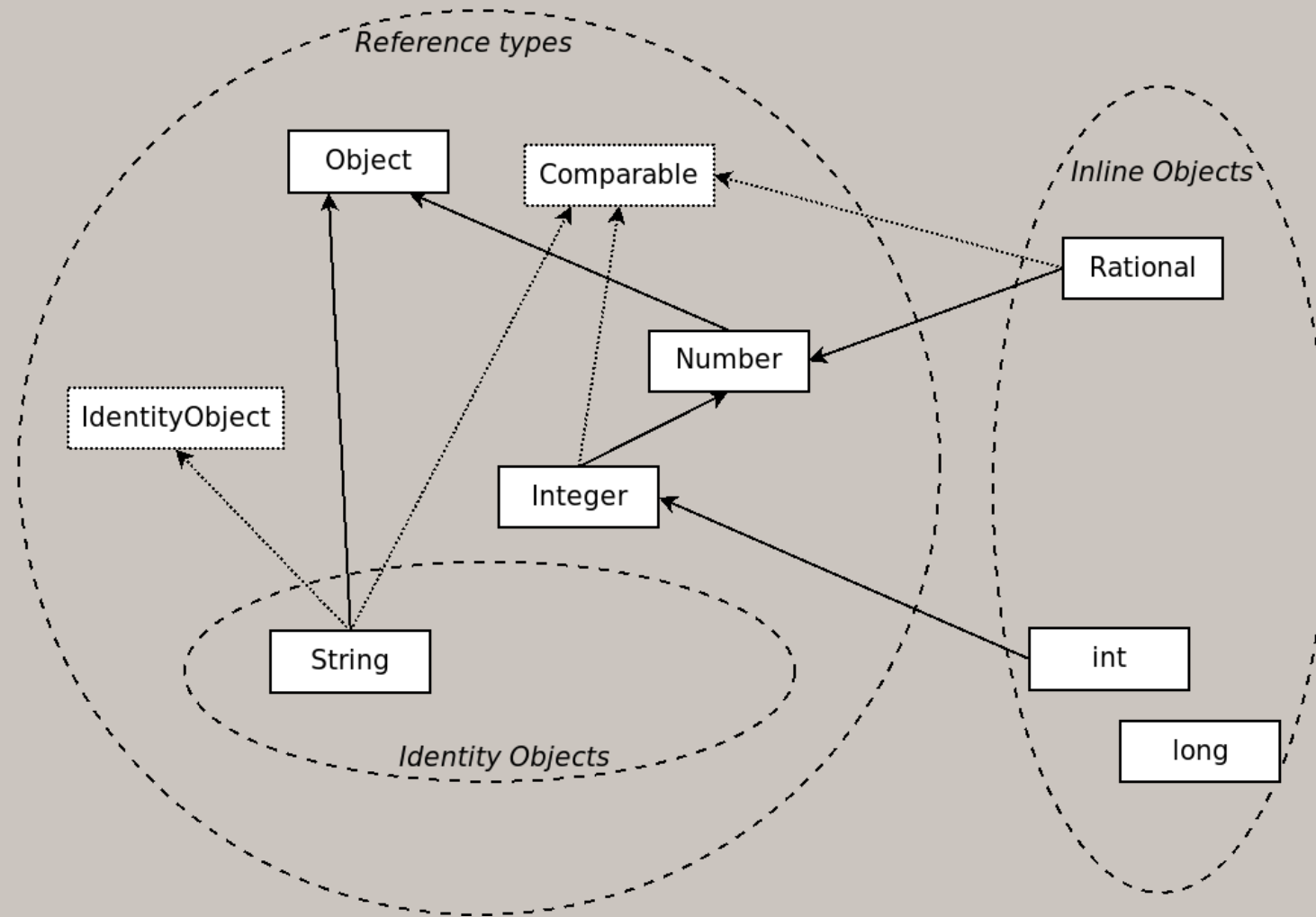
# Type hierarchy



# Type hierarchy



# Class hierarchy



# First step

- Deprecate identity of primitive wrappers

JDK-8236567 diagnostic anti-locked objects, dynamically by instance

JDK-8236568 diagnostic anti-locked objects, by class

JDK-8242263 Diagnose synchronization on primitive wrappers

# Second step

- Implement:

```
inline class int { ... }
```

- Integer == int.ref
- Integer.val == int



# As result

- Make Java true OOP ~~again~~
- Fully covariant arrays:

```
int[] <: Integer[] <: Object[]
```

- Simplify generic specialization:

```
List<int>
```

- ...

As result

at the beginning:

**Inlines – user-defined primitives**

at the end:

**Primitives – built-in inlines**

# Collateral benefits

# Nestmates

JEP 181: Nest-Based Access Control

@since 11

<https://openjdk.java.net/jeps/181>

# Field layout computation overhaul

JDK-8237767 Field layout computation overhaul

@since 15

-XX:+UseEmptySlotsInSupers

# Real objects size (in bytes)

	+UseCompressedOops		-UseCompressedOops	
	Java 14	Java 15	Java 14	Java 15
<code>class L1B0 { long l1; }</code>	24	24	24	24
<code>class L1B1 extends L1B0 { byte b1; }</code>	32	24	32	32
<code>class L2B1 extends L1B1 { long l2; }</code>	40	32	40	40
<code>class L2B2 extends L2B1 { byte b2; }</code>	48	32	48	40
<code>class L3B2 extends L2B2 { long l3; }</code>	56	40	56	48
<code>class L3B3 extends L3B2 { byte b3; }</code>	64	40	64	48
<code>class L4B3 extends L3B3 { long l4; }</code>	72	48	72	56
<code>class L4B4 extends L4B3 { byte b4; }</code>	80	48	80	56

# Specialized generics

# Erased Generics

“Background: how we got the generics we have  
(Or, how I learned to stop worrying and love erasure)”

Brian Goetz, June 2020

<http://cr.openjdk.java.net/~briangoetz/valhalla/erasure.html>



# Erased Generics

“Two Ways to Bake Your Pizza —

Translating Parameterised Types into Java”

Martin Odersky, Enno Runne, and Philip Wadler, 2000

<http://pizzacompiler.sourceforge.net/doc/pizza-translation.pdf>

# Specialization

- Layout specialization
  - `List<int>` - no boxing
- Polymorphic specialization
  - `Predicate<T> == Function<T, boolean>`
  - `HashSet<T> == HashMap<T, void>`

# Reification

- Reified generics preserve full type information
  - `(x instance of List<T>)`
  - `(List<int>)x;`

# Generics

- Java has erased generics → Java will preserve it
- Focus on specialization
  - List<reference type> - erased
  - List<inline type> - specialized
- Reification – not a goal

# Plans

- Two phases:
  - expand expressiveness (List<int>)
  - layout specialization
- No compile time specialization
  - only at runtime and only by JVM decision
- To be continued...

# F.A.Q.

# Q1

- “Complex[size]” vs “double[2\*size]”
- Which layout? Native interop?

# Q1

- “Complex[size]” vs “double[2\*size]”
- Which layout? Native interop?
- Valhalla doesn't specify layout
- @see Panama



# Q2

- Fused String, fused arrays (inject array into object )?

# Q2

- Fused String, fused arrays (inject array into object )?
- Valhalla doesn't do fused arrays
- @see Arrays 2.0

# Q3

- Why did you take name which has different meaning in Kotlin?

# Q3

- Why did you take name which has different meaning in Kotlin?
- Because we can.

# Links

## “State of Valhalla”;

1. The Road to Valhalla
2. Language Model
3. JVM Model
4. Translation scheme

<https://cr.openjdk.java.net/~briangoetz/valhalla/sov/01-background.html>

# Links

- Wiki:

<https://wiki.openjdk.java.net/display/valhalla/Main>

- Mailing lists:

<http://mail.openjdk.java.net/mailman/listinfo/valhalla-dev>

<http://mail.openjdk.java.net/mailman/listinfo/valhalla-spec-observers>

- Repository:

<https://github.com/openjdk/valhalla>

# Thank You

**Sergey Kuksenko**

Java Platform Group  
Oracle