Project Panama

Status Update

Java Your Next (Cloud)

Vladimir Ivanov
JVM Compiler team, JPG
Oracle
JVM Compiler team Offsite,
January, 22, 2017

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

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 Panama: What's in there?

http://hg.openjdk.java.net/panama/dev

```
$ hg branches
vectorIntrinsics                47814:8eafa3e1dd22
vectorSnippets                  47748:8f8232cadaff
nicl                            47744:8499209102d4
linkToNative                    47740:8b90565dcbfe
array2                          47726:1fd0974b1d4b
…
```

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

Project Panama: What's in there?

http://hg.openjdk.java.net/panama/dev

```
$ hg branches
vectorIntrinsics          47814:8eafa3e1dd22
vectorSnippets            47748:8f8232cadaff
nicl                      47744:8499209102d4
linkToNative              47740:8b90565dcbfe
array2                    47726:1fd0974b1d4b
…
```

Vector API

JNI 2.0

Arrays 2.0

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

Project Panama: Recent Activity

http://hg.openjdk.java.net/panama/dev

```
$ hg branches
vectorIntrinsics          47814:8eafa3e1dd22
vectorSnippets            47748:8f8232cadaff
nicl                      47744:8499209102d4
linkToNative              47740:8b90565dcbfe
array2                    47726:1fd0974b1d4b
…
```

Vector API

JNI 2.0

Arrays 2.0

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

Better JNI

“Codes like Java, works like native”

```
$ hg branches
vectorIntrinsics        47814:8eafa3e1dd22
vectorSnippets          47748:8f8232cadaff
nicl                    47744:8499209102d4
linkToNative            47740:8b90565dcbfe
array2                  47726:1fd0974b1d4b
```

Vector API

JNI 2.0

Arrays 2.0

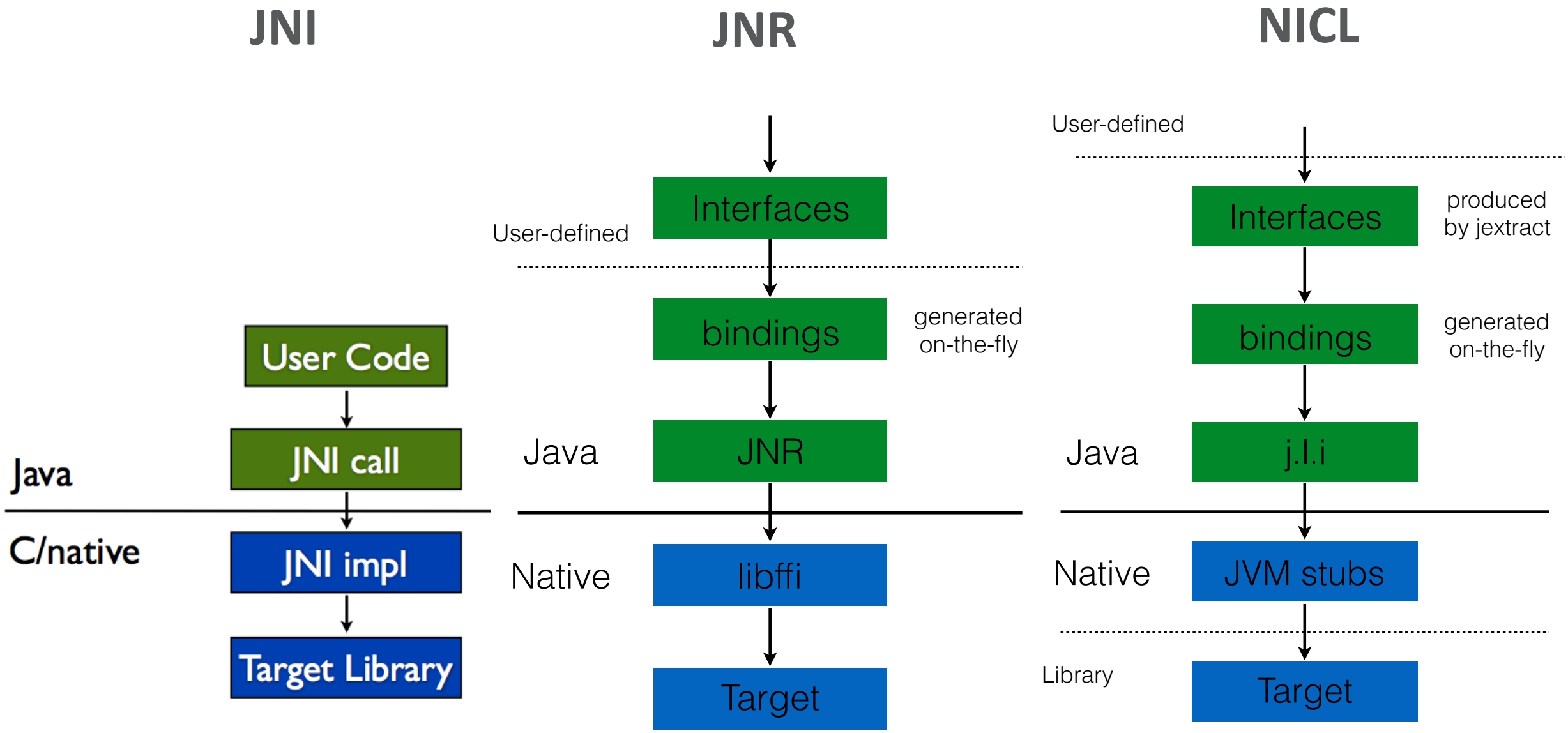
Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

What's in there?

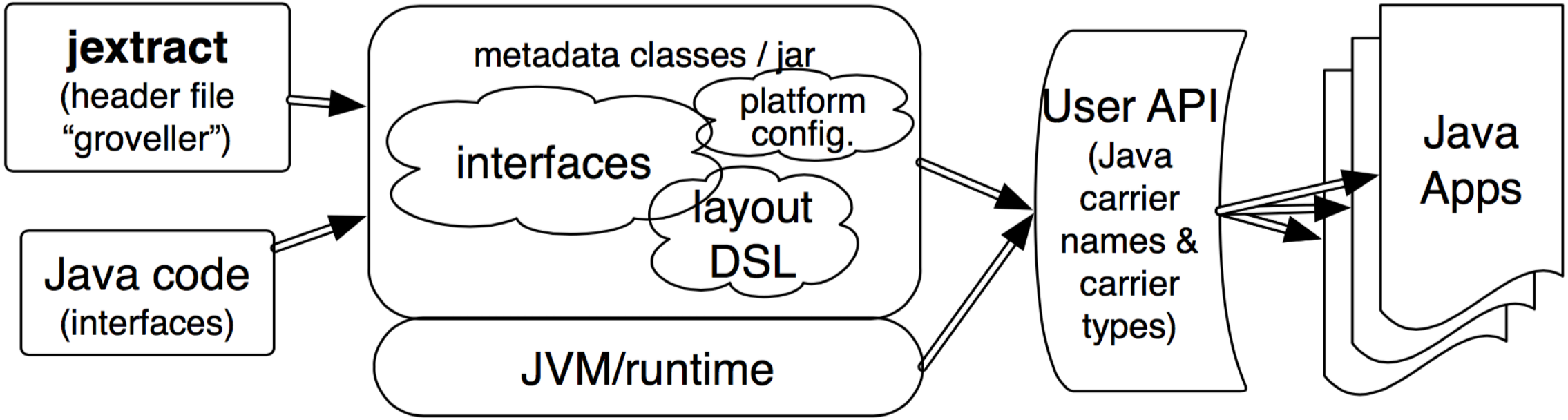
- **nicl**
  - jextract, libclang bindings (Henry, 2014-2016; Maurizio, Sundar, 2017-2018)
  - Universal adapter (Mikael, 2015-2016)
  - Binder, vectors, etc. (Mikael Vidstedt, 2015-2016)
  - MemoryRegion, Scope, Pointer (Henry, Mikael, 2016)

- **linkToNative**
  - MethodHandle-based native invoker (Vladimir Ivanov, 2015)

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



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



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

jextract - Header file -> Java interface

unistd.h

```
#ifndef _UNISTD_H
#define _UNISTD_H

pid_t getpid(void);

#endif // _UNISTD_H
```

unistd.class

```
package mypackage;

@Header(path="unistd.h")
public interface unistd {
  @C(file="unistd.h", line=3, column=1, USR="c:@F@getpid")
  @NativeType(ctype="pid_t (void)", size=1)
  @CallingConvention(1)
  public int getpid();
}
```

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

Example: getpid()

```
// Load native library
Library lib = NativeLibrary.load("c");

// Weave a class for the unistd interface,
// and return an instance
unistd unistd = NativeLibrary.create(lib, unistd.class)

// Call the system getpid() function
int pid = unistd.getpid();
```

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

Native Method Handles

```
MethodType mt = MethodType.methodType(int.class);

MethodHandle mh = NativeLookup.findNative(“getpid", mt);

int pid = mh.invokeExact();
```

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

Native Method Handles

	Java	Native
Construction	Lookup.findVirtual() et al	Lookup.findNative()
Reference (typed)	DirectMethodHandle	NativeMethodHandle
Reference (direct)	MemberName	NativeEntryPoint
Linker	MH.linkToVirtual() et al	MH.linkToNative()
Invocation	indy, MH.invoke(), MH.invokeExact()	

"Making native calls from the JVM" by John Rose
http://cr.openjdk.java.net/~jrose/panama/native-call-primitive.html

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

```
// int pid = mh.invokeExact();

@ 8   LambdaForm$MH::invokeExact_MT (29 bytes)   force inline by annotation
  @ 11   Invokers::checkExactType (17 bytes)   force inline by annotation
    @ 1   MethodHandle::type (5 bytes)   accessor
  @ 15   Invokers::checkCustomized (23 bytes)   force inline by annotation
    @ 1   MethodHandleImpl::isCompileConstant (2 bytes)   (intrinsic)
  @ 25   LambdaForm$NMH::invokeNative_I (27 bytes)   force inline by …
    @ 7   NativeMethodHandle::internalNativeEntryPoint (8 bytes)   force inline …
    @ 23   MethodHandle::linkToNative(L)I (0 bytes)   direct native call
```

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

Native Method Handles

callq 0x1057b2eb0 ; native method entry

	getpid
JNI	13.7 ± 0.5 ns
Direct call	3.4 ± 0.2 ns

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

Calling Convention

Java vs C

hotspot/src/cpu/x86/vm/sharedRuntime_x86_64.cpp:

“The Java calling convention is a "shifted" version of the C ABI. By skipping the first C ABI register we can call non-static jni methods with small numbers of arguments without having to shuffle the arguments at all. Since we control the java ABI we ought to at least get some advantage out of it.“

System V AMD64 ABI

arg	1st	2nd	3rd	4th	5th	6th	…
C	RDI	RSI	RDX	RCX	R8	R9	stack
Java	RSI	RDX	RCX	R8	R9	RDI	stack

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

“Universal Adapter”

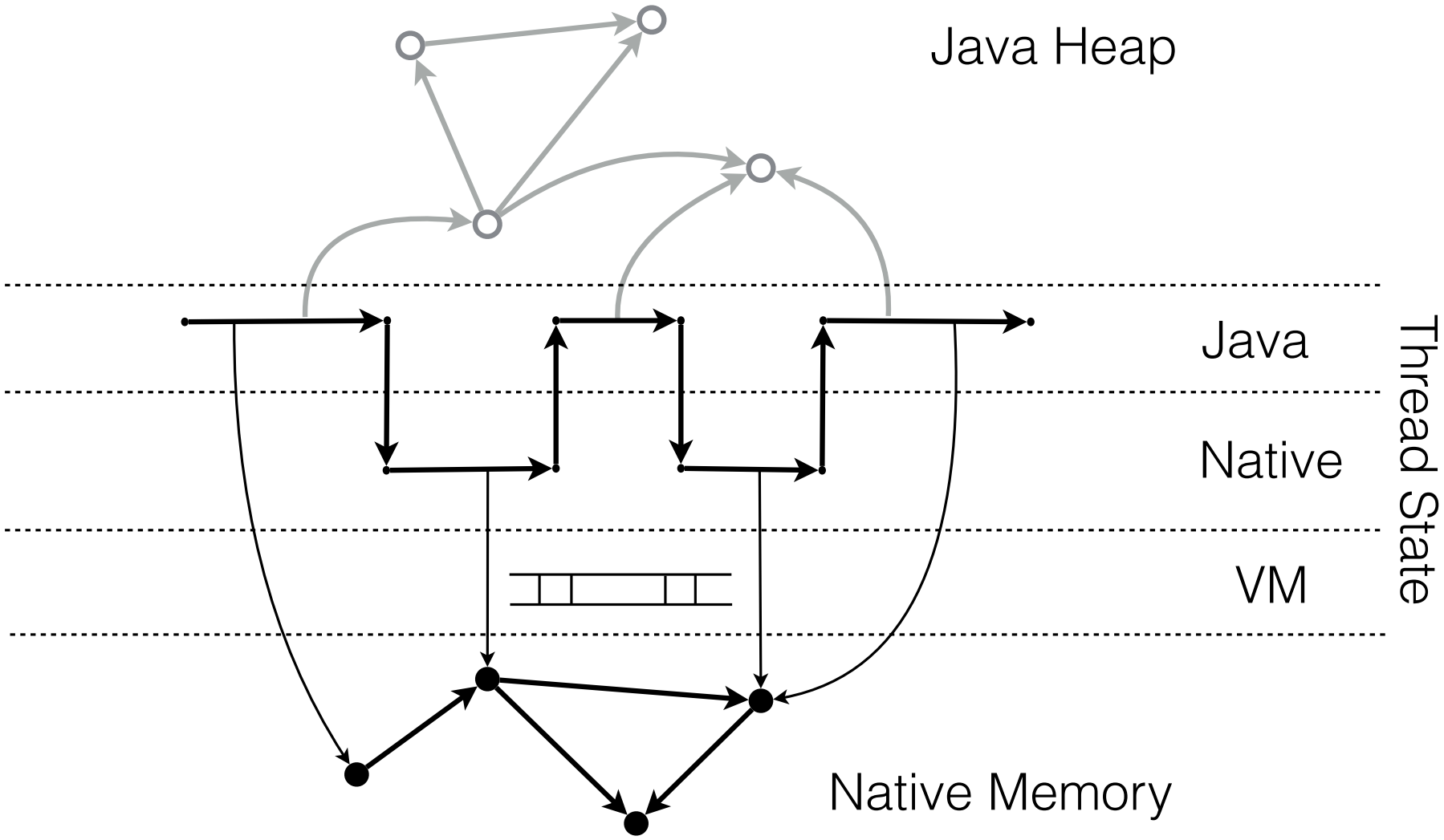
- jdk.internal.nicl.NativeInvoker
  - static native void **invokeNative**(long[] args, long[] rets, long[] recipe, long addr);
  - alternative to MH.linkToNative()
- Very powerful mechanism
  - fully programmable calling conventions through “recipe”
- Rich functionality support
  - varargs
  - struct-by-value argument + return
  - callbacks (function pointers)
- Hard to optimize by JITs

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

Optimize checks

```
void run() {
    nativeFunc1(); // checks & state trans.
    nativeFunc2(); // checks & state trans.
    nativeFunc3(); // checks & state trans.
}
```

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

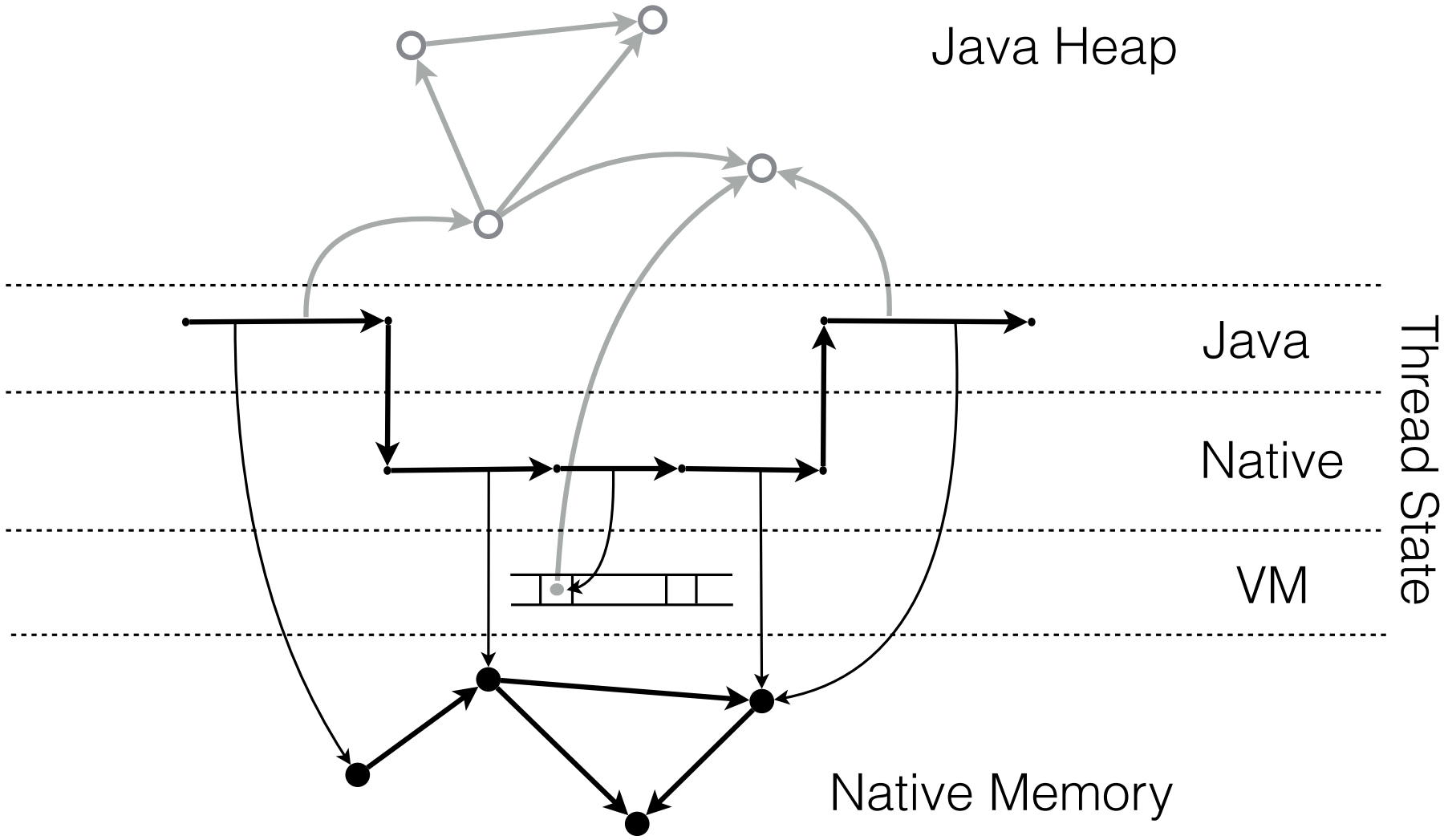


Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Optimize checks

```
void run() {
    if (!performChecks())  jump failed;
    transJavaToNative();
    call nativeFunc1();
    call nativeFunc2();
    call nativeFunc3();
    transNativeToJava();
}
```

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



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

Better JNI

Easier, Safer, Faster

- Native access between the JVM and native APIs
  - Native code via FFIs
  - Native data via safely-wrapped access functions
  - Tooling for header file API extraction and API metadata storage
- Wrapper interposition mechanisms, based on JVM interfaces
  - add (or delete) wrappers for specialized safety invariants
- Basic bindings for selected native APIs

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

Contributors

- John Rose
- Henry Jen
- Mikael Vidstedt
- Maurizio Cimadamore
- Sundararajan Athijegannathan
- Tobi Ajila (IBM)

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

Vector API

```
$ hg branches
vectorIntrinsics        47814:8eafa3e1dd22
vectorSnippets          47748:8f8232cadaff
nicl                    47744:8499209102d4
linkToNative            47740:8b90565dcbfe
array2                  47726:1fd0974b1d4b
```

Vector API

JNI 2.0

Arrays 2.0

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

Motivation

Expose data-parallel operations
through a cross-platform API

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

Motivation

```
Int8Vector x = ..., y = ...; // packed vectors of 8 ints
Int8Vector z = x.add(y);     // element-wise addition
```

```
vpaddd %ymm1,%ymm0,%ymm0
```

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

Goals

1. Maximally expressive and portable API
   - “principle of least astonishment”
   - uniform coverage operations and data types
   - type-safe
2. Performant
   - predictable performance
   - high quality of generated code
   - competitive with existing facilities for auto-vectorization
3. Graceful performance degradation
   - fallback for "holes" in native architectures

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

Draft API (by John Rose, c. 2015)

```
interface Vector<E,S extends Shape<Vector<?,S>>> {
    Vector<E,S> add(Vector<E,S> v2);
}

Vector<Integer,S256Bit> x = ..., y = ...; // vectors of 8 ints
Vector<Integer,S256Bit> z = x.add(y);     // element-wise addition
```

```
vpaddd %ymm1,%ymm0,%ymm0
```

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

Implementation Challenges

1. How to represent vector operations on JVM level?
2. Optimize away vector boxes
   - required for mapping Vector instances to vector registers in generated code
   - Vector<Integer,S256Bits> => ymm regs on x86/AVX
3. Primitive specialization
   - Vector<**Integer**,S256Bits> vs Vector<**int**,S256Bits>
4. Vectorize higher-order operations
   - Vector<E,S> map(UnaryOperator<E> op);
   - “You can fit a universe in a lambda”

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

Vector API: Ultimate Goal

```
Vector<int,?>.map(arr, i -> i + 1)
```

(with **value types**, **generic specialization**, and **crackable lambdas**)

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

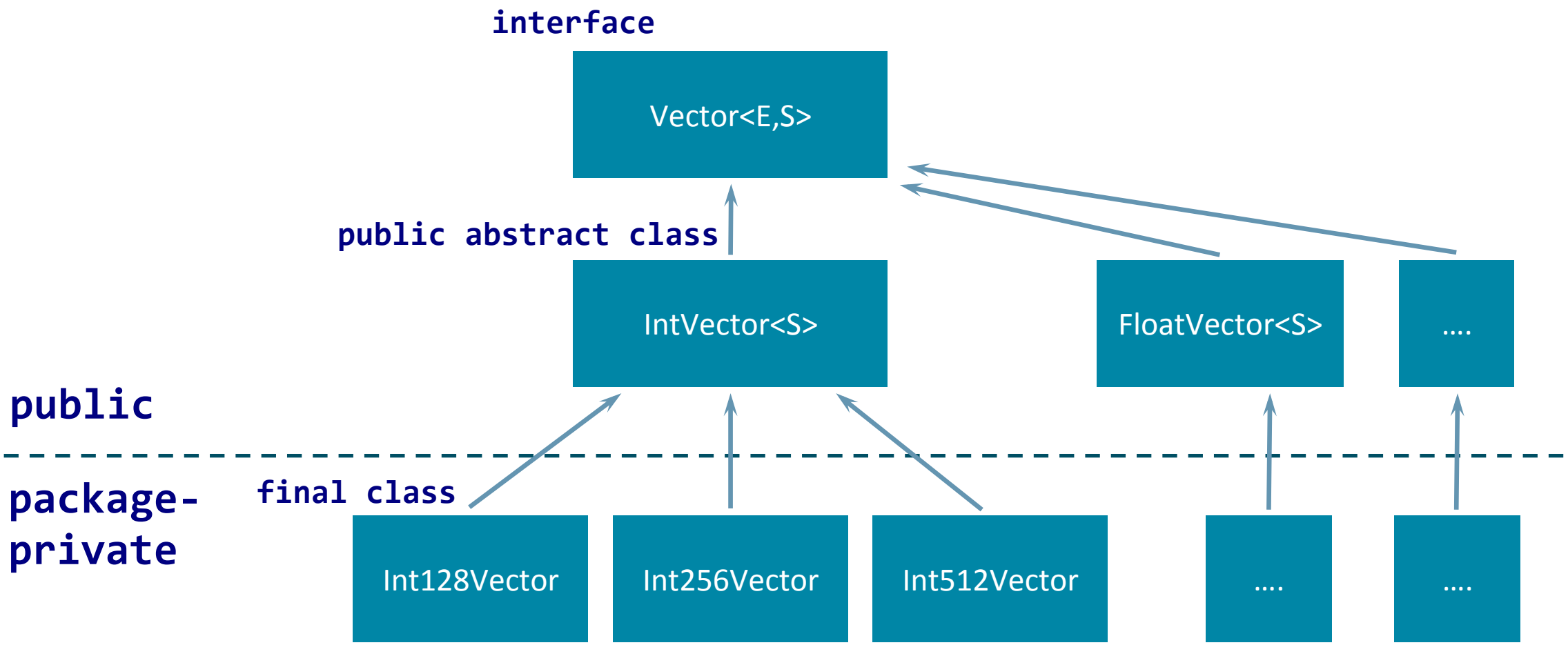
Implementation Challenges

1. **How to represent vector operations on JVM level?**
2. **Optimize away vector boxes**
   - required for mapping Vector instances to vector registers in generated code
   - Vector<Integer,S256Bits> => ymm regs on x86/AVX

3. Primitive specialization
   - Vector<**Integer**,S256Bits> vs Vector<**int**,S256Bits>
4. Vectorize higher-order operations
   - Vector<E,S> map(UnaryOperator<E> op);
   - “You can fit a universe in a lambda”

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

```
package jdk.incubator.vector;
```



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

History

1. Machine Code Snippets + Super-longs (2016-2017)
2. MVT-based Vectors (2017)
3. Intrinsic-backed Typed Vectors (2017-2018)

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

Machine Code Snippets + Super-Longs

Iteration #1
2016

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

Machine Code Snippets

Motivation / Goals

- Use case: prototyping
  1. minimize implementation costs
  2. decent performance
  3. up to a dozen instructions in size

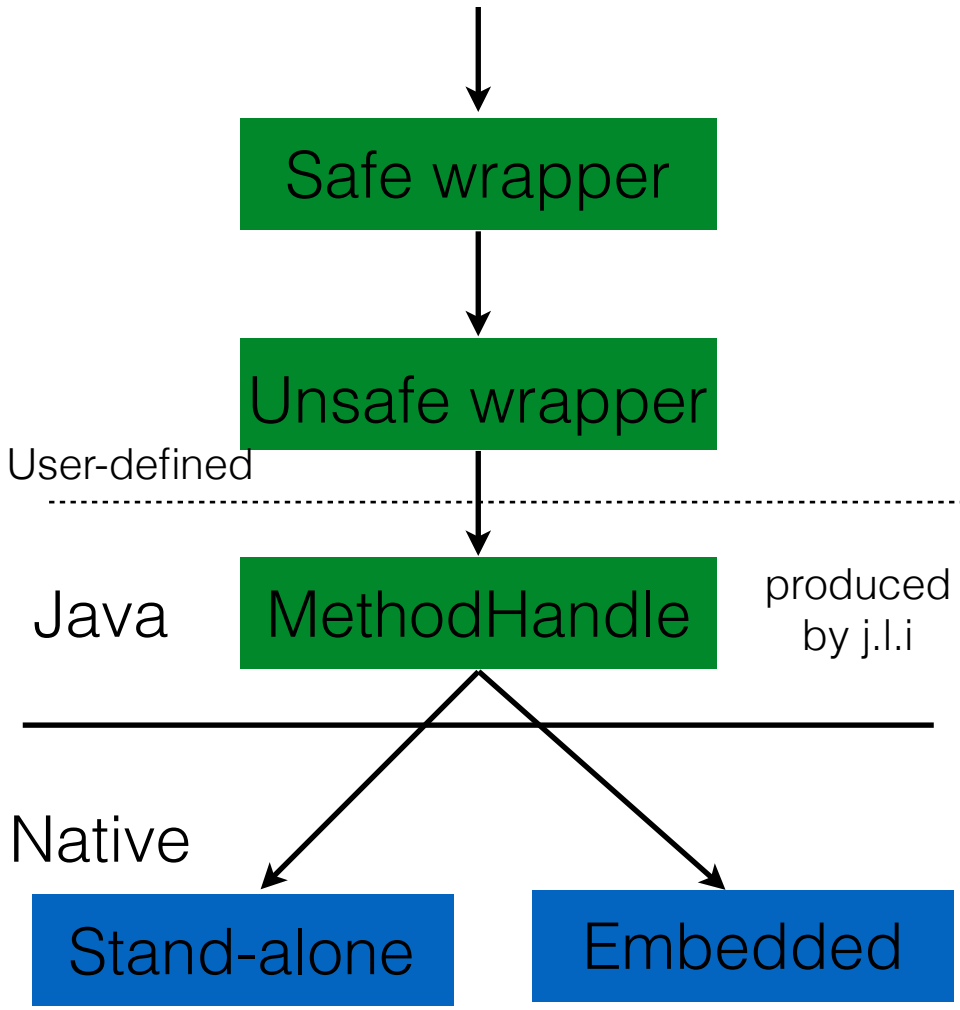
- Existing solutions
  - JVM intrinsics: 1. no / 2. yes
  - JNI / NMH: 1. yes / 2. no

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

Machine Code Snippets

- Wrap raw machine code in a method handle
  - Use snippet "recipe"
  - Let user know about RA decisions

- 2 execution modes
  - optimized:
    - embedded in generated code
  - non-optimized, interpreted
    - invokes stand-alone version



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

VPADDD

VEX.NDS.256.66.0F.WIG FE /r VPADDD *ymm1, ymm2, ymm3/m256*	RVM	V/V	AVX2	Add packed doubleword integers from *ymm2, ymm3/m256* and *store in ymm1*.

```
static final MethodHandle MHm256_vpaddd =
    MachineCodeSnippet.make("mm256_vpaddd",
            MethodType.methodType(Long4.class, Long4.class, Long4.class),
            requires(AVX2),
            << vpaddd %ymm?,%ymm?,%ymm? >>);


static Long4 vpaddd(Long4 v1,Long4 v2) {
    try {
        return MHm256_vpaddd.invokeExact(v1, v2);
    } catch (Throwable e) { throw new Error(e); }
}
```

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

VPADDD

VEX.NDS.256.66.0F.WIG FE /r VPADDD *ymm1, ymm2, ymm3/m256*	RVM	V/V	AVX2	Add packed doubleword integers from *ymm2, ymm3/m256* and *store in ymm1*.

```
static final MethodHandle MHm256_vpaddd =
    MachineCodeSnippet.make("mm256_vpaddd",
            MethodType.methodType(Long4.class, Long4.class, Long4.class),
            requires(AVX2),
            << vpaddd %ymm?,%ymm?,%ymm? >>);
```

```
static Long4 vpaddd(Long4 v1,Long4 v2) {
    try {
        return MHm256_vpaddd.invokeExact(v1, v2);
    } catch (Throwable e) { throw new Error(e); }
}
```

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

JVM vs Hardware

Impedance Mismatch

size (bits)	8	16	32	64	128	256	512	…
x86 regs	AL	AX	EAX	RAX	XMM0	YMM0	ZMM0	-
JVM	B	S	I	J	-	-	-	…

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

Super-Longs

- java.lang.Long2 / Long4 / Long8
  - represent 128/256/512-bit values
- “well-known“ to the JVM
  - special treatment in the JVM
  - C2 knows how to map the values to appropriate vector registers
- 2 ways to define operations on vectors
  - methods on Long2 / Long4 / Long8
  - user-defined machine code snippets

```
// 128-bit vector.
public /* value */ final class Long2 {
    private final long l1, l2; // FIXME

    private Long2() { throw new Error(); }

    @HotSpotIntrinsicCandidate
    public static native Long2 make(long lo, long hi);

    @HotSpotIntrinsicCandidate
    public native long extract(int i);

    @HotSpotIntrinsicCandidate
    public boolean equals(Long2 v) { ... }

    ...
```

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

JVM vs Hardware

Super-Longs

size (bits)	8	16	32	64	128	256	512	…
x86 regs	AL	AX	EAX	RAX	XMM0	YMM0	ZMM0	-
JVM	B	S	I	J	j.l.Long2	j.l.Long4	j.l.Long8	…

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

Vector API: Implementation

```
interface Vector<E,S extends Shape<Vector<?,S>>> {
    Vector<E,S> add(Vector<E,S> v2);
}

abstract class IntVector
       <S extends Vector.Shape<Vector<?, S>>>
      implements Vector<Integer, S> {…}


final class Int256Vector extends IntVector<S256Bit> {
    private final Long4 vec;

    public Vector<Integer,S256Bit> add(Vector<Integer,S256Bit> v) {
        Int256Vector v256i = (Int256Vector) v;
        return new Int256Vector(VecUtils.vpaddd(this.vec, v256i.toLong4()));
    }
```

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

Vector API: Implementation

```
interface Vector<E,S extends Shape<Vector<?,S>>> {
    Vector<E,S> add(Vector<E,S> v2);
}

abstract class IntVector
       <S extends Vector.Shape<Vector<?, S>>>
      implements Vector<Integer, S> {…}


final class Int256Vector extends IntVector<S256Bit> {
    private final Long4 vec;

    public Vector<Integer,S256Bit> add(Vector<Integer,S256Bit> v) {
        Int256Vector v256i = (Int256Vector) v;
        return new Int256Vector(VecUtils.vpaddd(this.vec, v256i.toLong4()));
    }
```

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

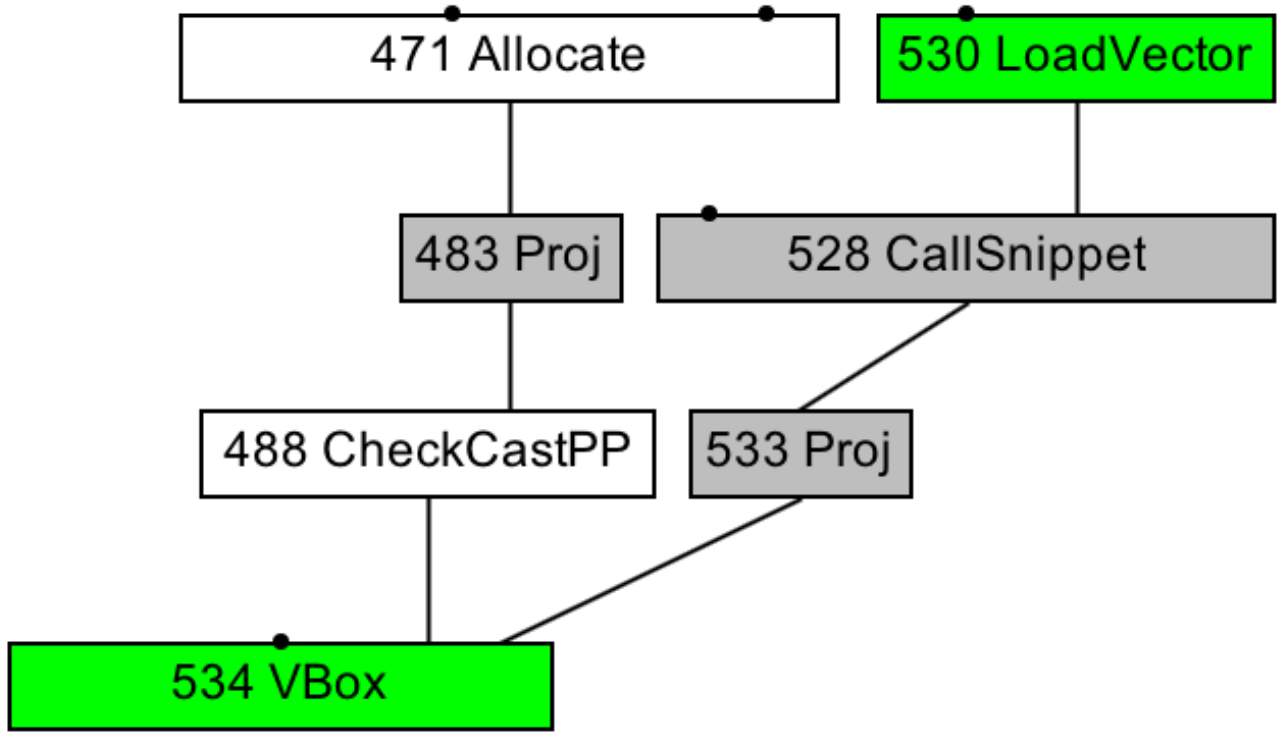
Vector API: Implementation

```
interface Vector<E,S extends Shape<Vector<?,S>>> {
    Vector<E,S> add(Vector<E,S> v2);
}

abstract class IntVector
       <S extends Vector.Shape<Vector<?, S>>>
      implements Vector<Integer, S> {…}


final class Int256Vector extends IntVector<S256Bit> {
    private final Long4 vec;

    public Vector<Integer,S256Bit> add(Vector<Integer,S256Bit> v) {
        Int256Vector v256i = (Int256Vector) v;
        return new Int256Vector(VecUtils.vpaddd(this.vec, v256i.toLong4()));
    }
```

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

Snippet Result Boxing

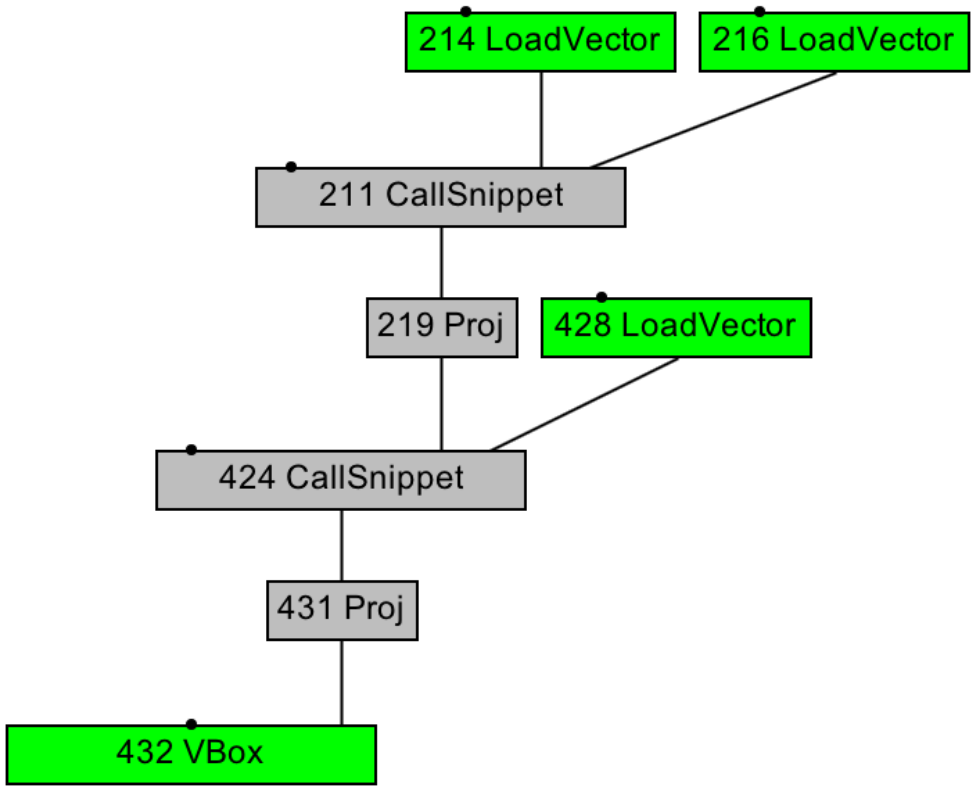
C2 IR



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

Nested Vector Operations

C2 IR



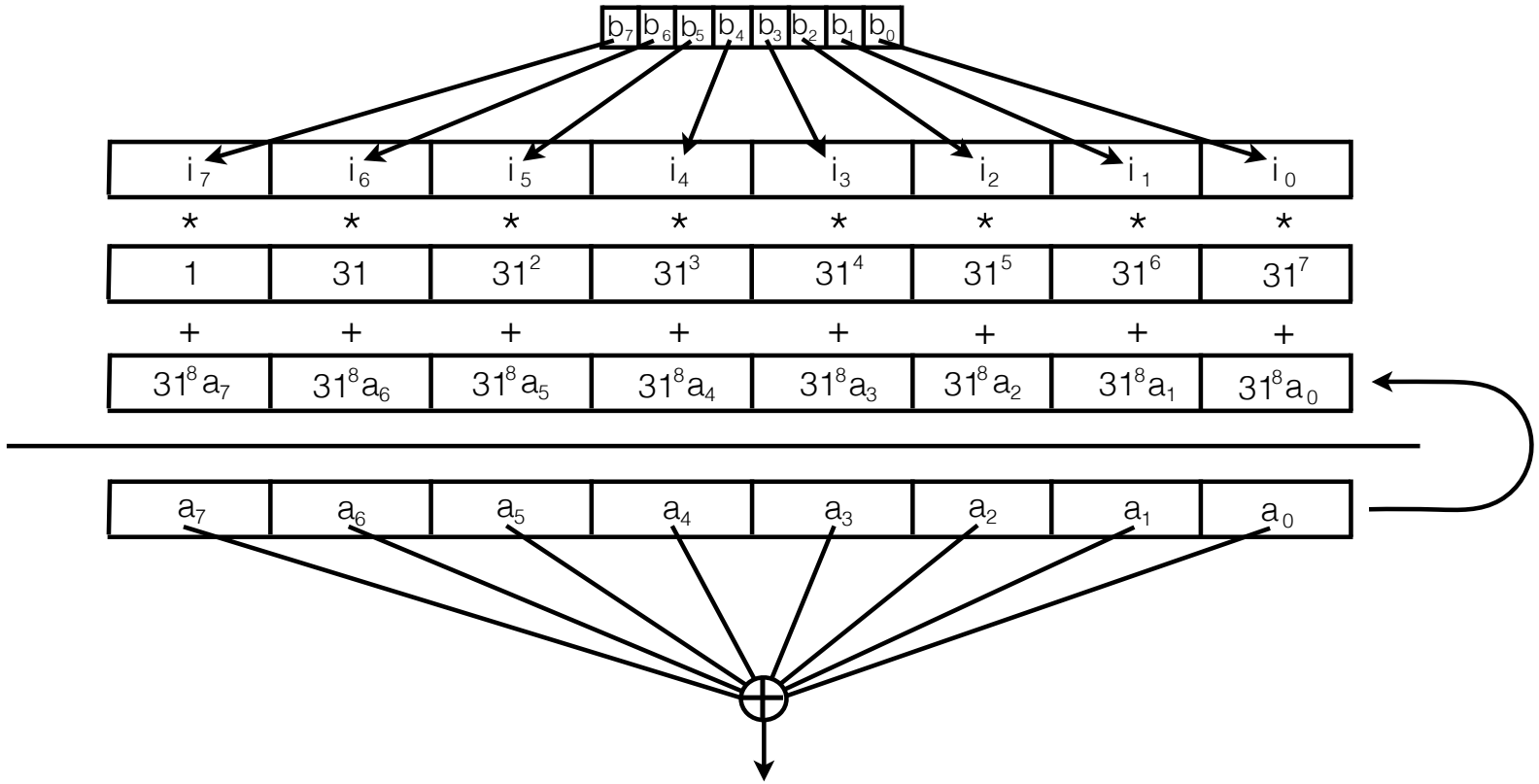
Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Use Case: Hash

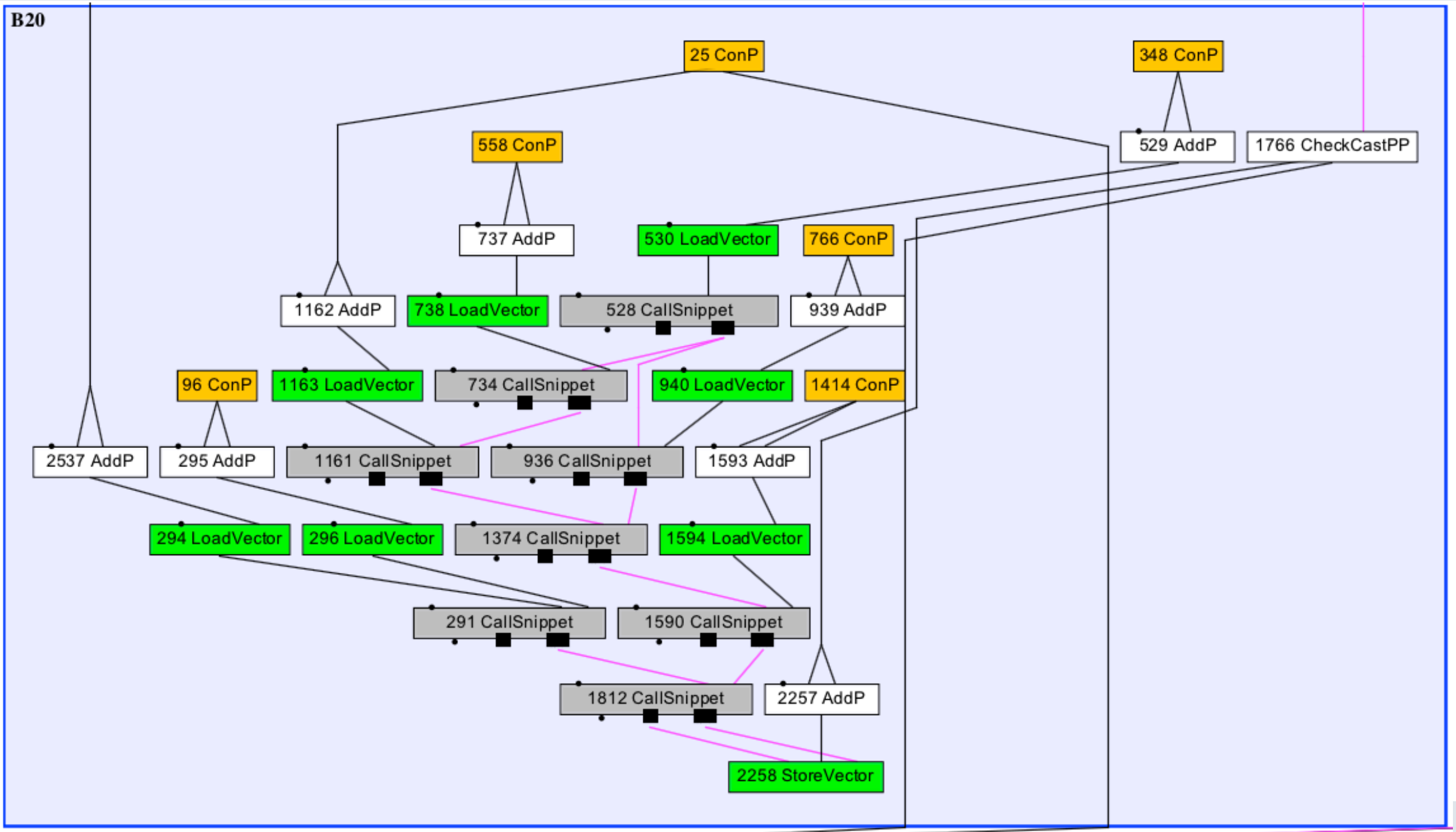
$$h_{i+1} = 31*h_i + v_i$$

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

Use Case: Vectorized Hash

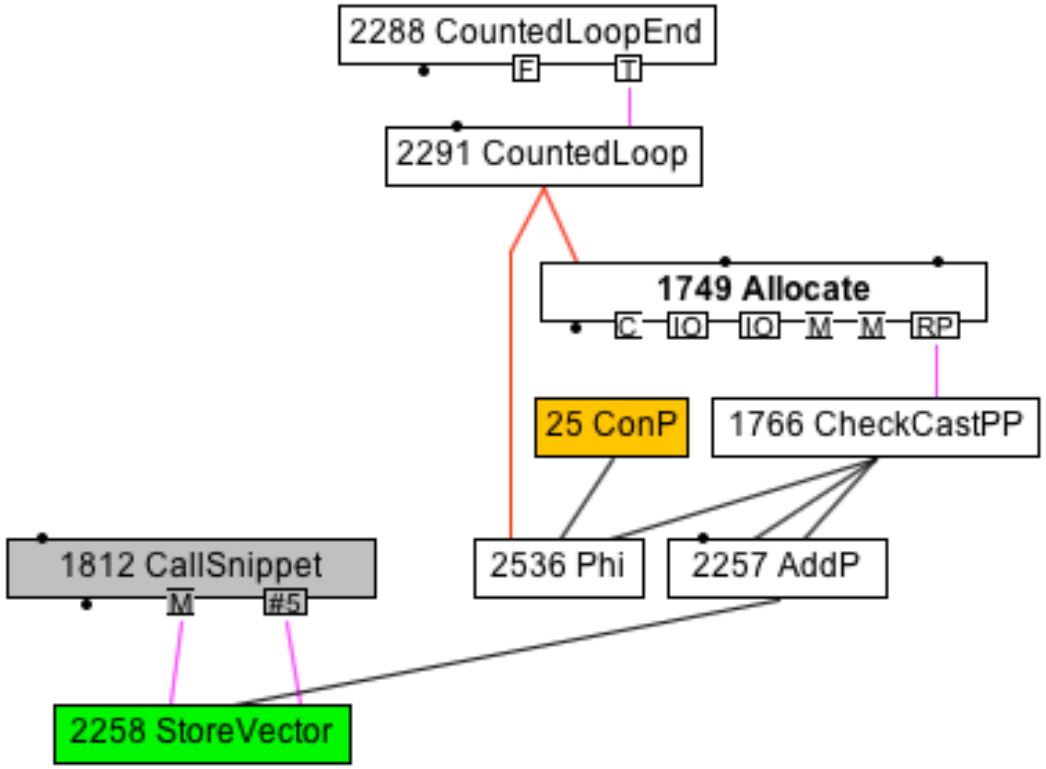


Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |



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

```
Long4 acc = Long4.ZERO;
for (; len >= 8; off += 8, len -
    long v = U.getLong(...);
    acc = vhash8(acc, v);
}
```



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

Escape Analysis in C2

Limitations

"The test case shows limitation of the current EA implementation. Objects will not be eliminated if there is merge point in which it is undefined which object is referenced."

JDK-6853701

"[The] address may point to more then one object. This may produce the false positive result (set not scalar replaceable) since the flow-insensitive escape analysis can't separate the case when stores overwrite the field's value from the case when stores happened on different control branches."

hotpsot/src/share/vm/opto/escape.cpp#l1733

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

Escape Analysis in C2

Observations

- Phi node blocks allocation elimination
- No vector load hoisting out of a loop
- No constant folding of vector values
- Repeated unboxing
- Box allocation placement

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

Machine Code Snippets + Super-Longs

Summary

- Pros
  - excellent for exploratory development
  - snippets cover all exectution modes

- Cons
  - liminations of C2 escape analysis
  - default implementations are needed anyway
    - for running on H/W w/o proper support (SSE vs AVX, etc)
  - 2 JVM features instead of 1
    - Machine Code Snippets are a full-blown feature on their own

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

MVT-based Vectors

Iteration #2
2016-2017

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

Vector Box Elimination

- Crucial for decent performance
- Escape Analysis in C2
  - doesn't cover all the cases (e.g., non-trivial control flow)
  - brittle (depends on inlining decisions; easy for a user to leak an instance)

- Value Types for the rescue!
  - (but in a longer term… not there yet)
  - represent super-longs & typed vectors as value types
  - let the JIT-compiler do the rest

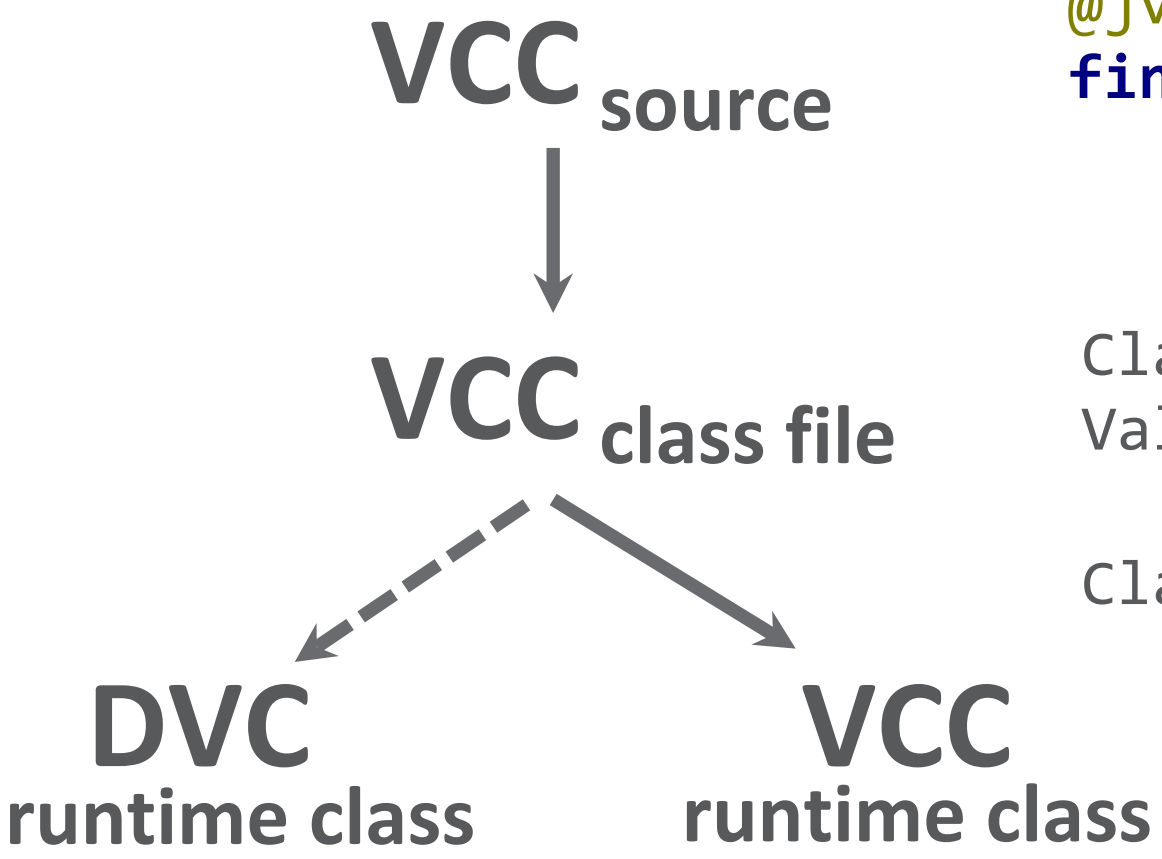
Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Minimal Value Types (MVT)

- First version
  - “minimized but viable subset of value-type functionality“
  - no language/tooling support
- Value operations are provided through method handles

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

Minimal Value Types (MVT)




```
@jvm.internal.value.ValueCapableClass
final class Int256Vector
        implements IntVector<S256Bit>
```

```
Class<?> VCC = Int256Vector.class;
ValueType<?> VT =
    ValueType.forClass(VCC);
Class<?> DVC = VT.valueClass();
```

VCC = Value Capable Class
DVC = Derived Value Class

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

Limitations: no VT-in-VT support

No place for raw vs typed vectors separation anymore.

size	8	16	32	64	128	256	512	…
x86	AL	AX	EAX	RAX	XMM0	YMM0	ZMM0	-
JVM	B	S	I	J	**Int128Vector** **Long128Vector** **Float128Vector** …	**Int256Vector** **Long256Vector** **Float256Vector** …	**Int512Vector** **Long512Vector** **Float512Vector** …	…

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

MethodHandle-based Vector API

- Vector operations as method handles
  - MVT will only expose value operations as MethodHandles
  - wrapping value types into ordinary classes doesn't work
    - no benefits for box elimination

- 2 ways to compose vector operations
  - method handle combinators
  - bytecode

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

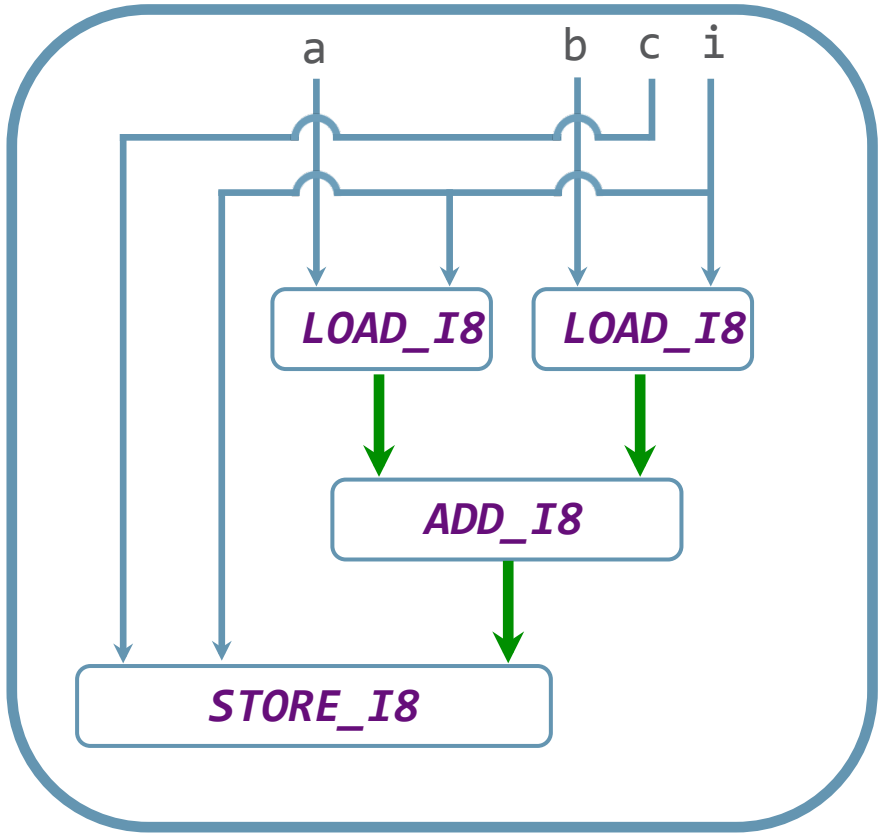
Example

```
// c[i:i+7] = a[i:i+7] + b[i:i+7]
void vectorized_add(int[] a, int[] b, int[] c, int i) {
    Int256Vector va = LOAD_I8(a, i);
    Int256Vector vb = LOAD_I8(b, i);
    Int256Vector vc = ADD_I8(va, vb);
    STORE_I8(c, i, vc);
}

MethodHandle LOAD_I8  = … // MT(int[],int)DVC
MethodHandle ADD_I8   = … // MT(DVC,DVC)DVC
MethodHandle STORE_I8 = … // MT(int[],int,DVC)void
                                VCC = IntVector256
                                DVC = IntVector256$Value
```

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

MVT: Method Handle Combinators

```
MethodHandle mh1 = MethodHandles.collectArguments(
                        ADD_I8, 1, LOAD_I8);

MethodHandle mh2 = MethodHandles.collectArguments(
                        mh1, 0, LOAD_I8);

MethodHandle mh3 = MethodHandles.collectArguments(
                        STORE_I8, 2, mh2);

MethodHandle mh4 = MethodHandles.permuteArguments(
        mh3, methodType(void.class,
                        int[].class, int[].class,
                        int[].class, int.class),
        2, 3, 0, 3, 1, 3);

VADD_MH = mh4;
```



VADD_MH

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

MVT: Bytecode Generation

```
MethodHandleBuilder.loadCode(LOOKUP, ”loop_body",
     methodType(void.class, int[].class, int[].class,
                                int[].class, int.class),
    CODE -> { CODE
        .ldc(LOAD_I8) // constant pool patch
        .checkcast(MethodHandle.class)
        .load(0)      // a: int[]
        .load(3)      // i: int
        .invokevirtual(MethodHandle.class, "invokeExact",
                       "([II)QIntVector256$Value;", false)
        .store(4)    // va: QIntVector256$Value
```

DVC = IntVector256$Value

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

Q-typed bytecode

```
VADD = MethodHandleBuilder.loadCode(LOOKUP, "vadd",
    methodType(void.class, int[].class, int[].class, int[].class, int.class),
    CODE -> { CODE
        // va = loadI8(a,i)
        .ldc(LOAD_I8)
        .checkcast(MethodHandle.class)
        .load(0) // a: int[]
        .load(3) // i: int
        .invokevirtual(MethodHandle.class, "invokeExact",
                       LOAD_I8.type().toMethodDescriptorString(), false)
        .store(4) // va: QIntVector256

        // vb = loadI8(b,i)
        .ldc(LOAD_I8)
        .checkcast(MethodHandle.class)
        .load(1) // b: int[]
        .load(3) // i: int
        .invokevirtual(MethodHandle.class, "invokeExact",
                       LOAD_I8.type().toMethodDescriptorString(), false)
        .store(5) // vb: QIntVector256

        // vc = add(va,vb)
        .ldc(ADD_I8)
        .checkcast(MethodHandle.class)
        .load(4)  // va: QIntVector256
        .load(5)  // vb: QIntVector256
        .invokevirtual(MethodHandle.class,  "invokeExact",
                       ADD_I8.type().toMethodDescriptorString(), false)
        .store(6) // vc: QIntVector256

        .ldc(STORE_I8)
        .checkcast(MethodHandle.class)
        .load(2) // c: int[]
        .load(3) // i: int
        .load(6) // vc: QIntVector256
        .invokevirtual(MethodHandle.class, "invokeExact",
                       STORE_I8.type().toMethodDescriptorString(), false)
        .return_();
});
```

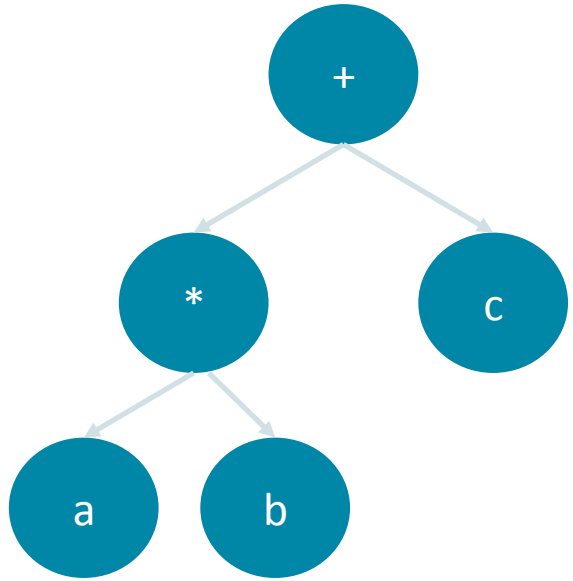
Method handle combinators

```
MethodHandle ADD_I8   = vectorOperation(Op.ADD, I8_DVC);
MethodHandle LOAD_I8  = vectorOperation(Op. ARRAY_LOAD, I8_DVC);
MethodHandle STORE_I8 = vectorOperation(Op.ARRAY_STORE, I8_DVC);

// ADD_I8 #0 (LOAD_I8 #1 #2)
MethodHandle mh1 = MethodHandles.collectArguments(ADD_I8, 1, LOAD_I8);

// ADD_I8 (LOAD_I8 #0 #1) (LOAD_I8 #2 #3)
MethodHandle mh2 = MethodHandles.collectArguments(mh1, 0, LOAD_I8);

// STORE_I8 #0 #1 (ADD_I8 (LOAD_I8 #2 #3) (LOAD_I8 #4 #5))
MethodHandle mh3 = MethodHandles.collectArguments(STORE_I8, 2, mh2);

// STORE_I8 #2 #3 (ADD_I8 (LOAD_I8 #0 #3) (LOAD_I8 #1 #3))
MethodHandle mh4 = MethodHandles.permuteArguments(mh3,
        methodType(void.class,
                   int[].class, int[].class, int[].class, int.class),
        2, 3, 0, 3, 1, 3);

VADD_MH = mh4;


static void add(int[] a, int[] b, int[] c) {
    try {
        for (int i = 0; i < a.length; i += 8) {
            VADD_MH.invokeExact(a, b, c, i);
        }
    } catch (Throwable e) {
        throw new Error(e);
    }
}
```

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

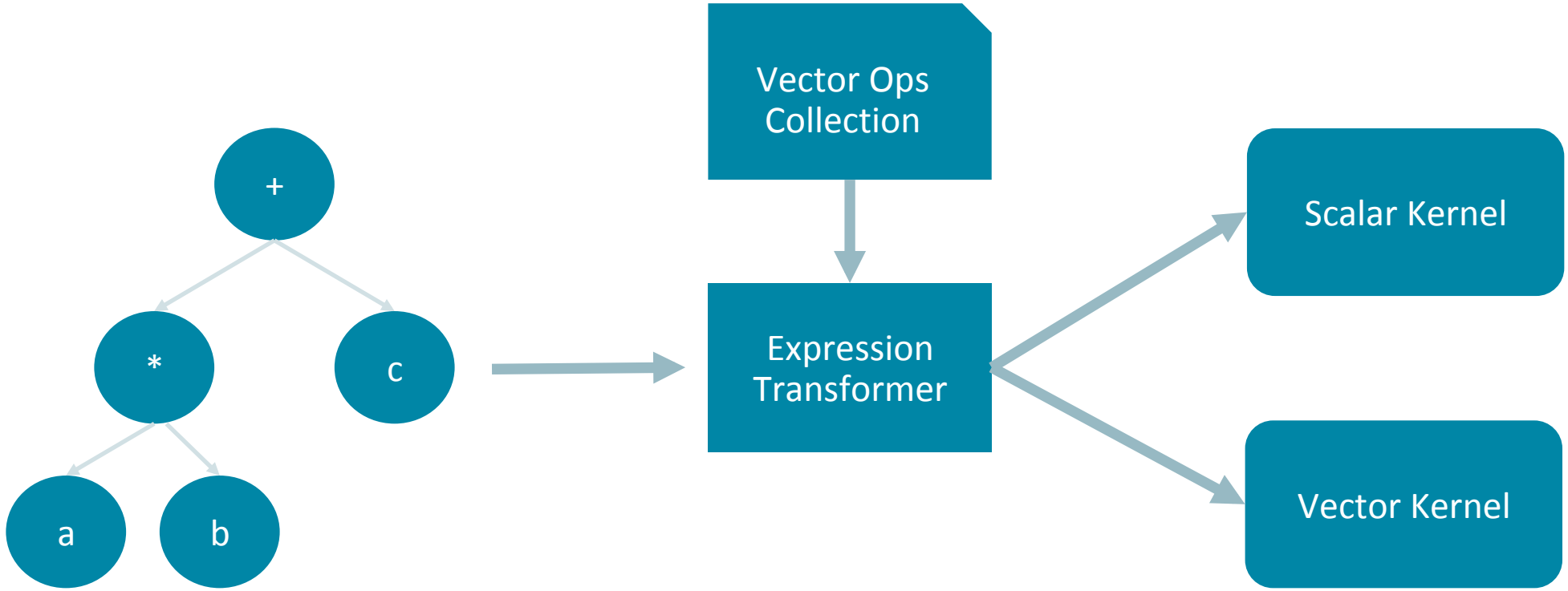
Vector API: Expression Language

```
(int a, int b, int c) -> {
      a * b + c;
}
```



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

Vector API: Expression Language



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

Vector API: Expression Language

```
MethodHandle addMH =
  Builder.builder(IntVector256.class) // Return type
         .bind(Int256Vector.class, // a
               Int256Vector.class, // b
               Int256Vector.class, // c
               (a, b, c, B) -> B.return_(a.mul(b).add(c)))
         // Instantiate with a method handle mapping
         .build(opsProvider);
```

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

Summary

- Pros
  - excellent JVM support
    - boxing problem solved
    - no implicit boxing
      - only vbox/vunbox bytecodes trigger vector boxing/unboxing
- Cons
  - too low-level API for users
    - exposing MVT-based API to users is not feasible
      - too verbose, too many implementation details
    - needs some wrapping, but naïve approaches don’t play well with MVT design
      - vector wrapping defeats MVT
  - Expression Language (EL) as a interim solution

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

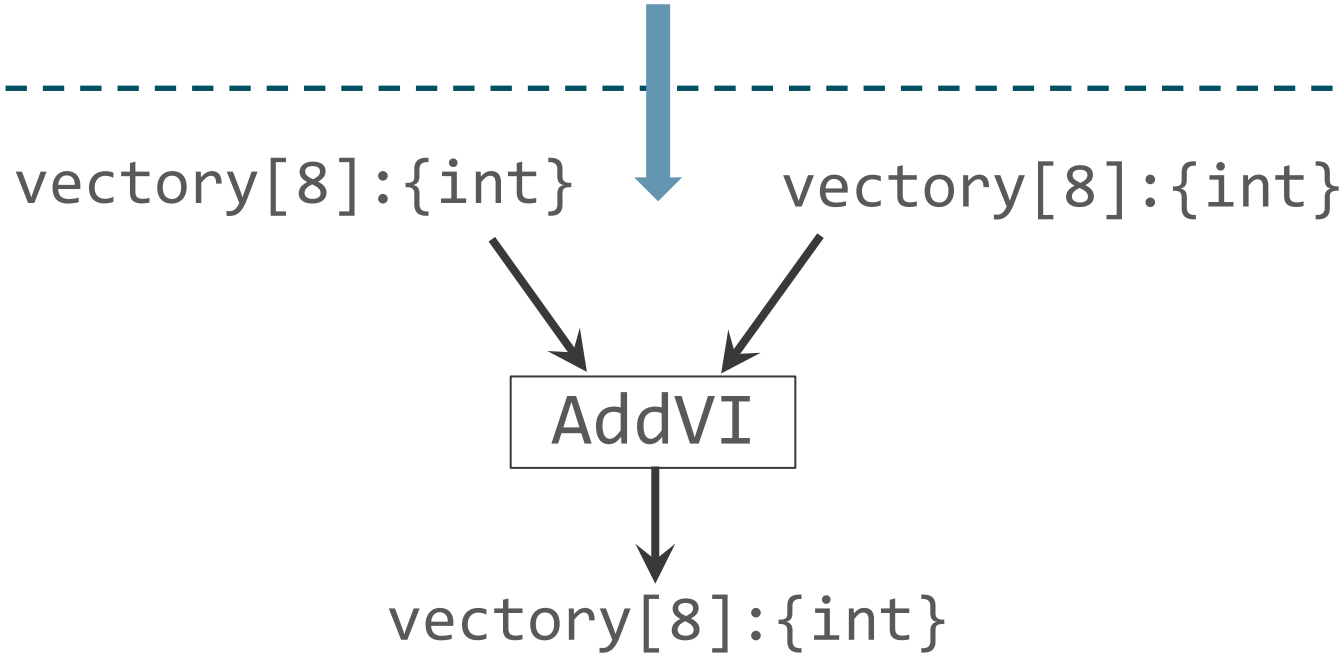
Intrinsic-backed Typed Vectors

Iteration #3

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

```
Int256Vector x = ..., y = ...;
Int256Vector z = x.add(y);
```

Java



C2 IR

vpaddd %ymm1,%ymm0,%ymm0

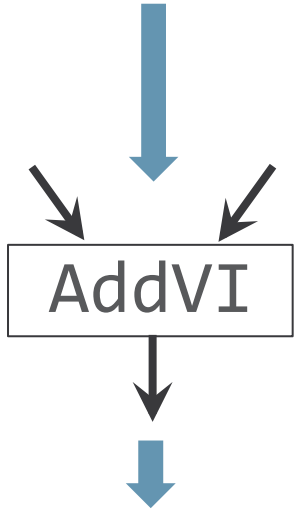
Machine code

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

```
IntVector<S256Bit> z =
    x.add(y);
```

```
Vector<Integer,S256Bit> z =
    x.add(y);
```

```
Int256Vector z = x.add(y);
```



vpaddd %ymm1,%ymm0,%ymm0

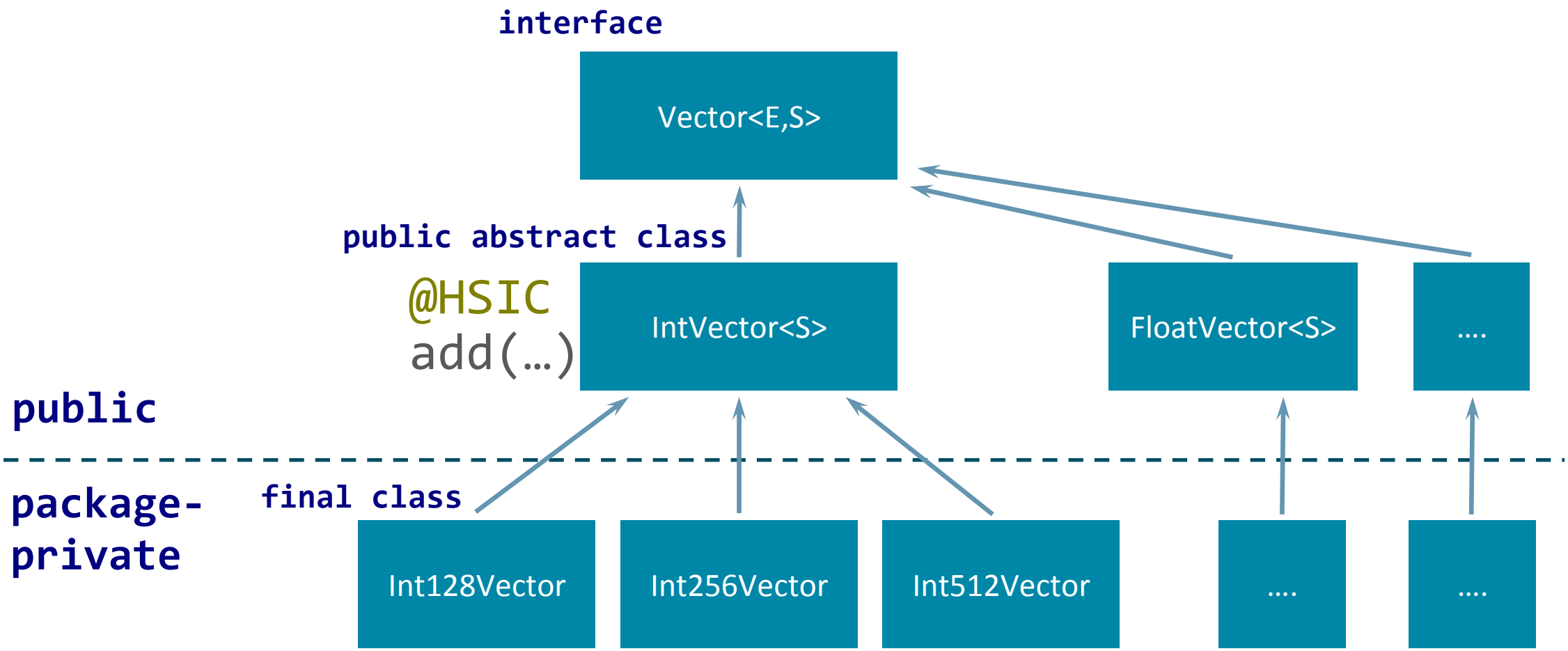
Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

```java
package jdk.incubator.vector;

interface Vector<E,S extends Shape<Vector<?,S>>> {
    Vector<E,S> add(Vector<E,S> v2);
}

public abstract class IntVector <S extends …> implements Vector<Integer,S> {
    @HotSpotIntrinsicCandidate
    public IntVector<S> add(Vector<Integer,S> o) {
        return bOp(o, (i, a, b) -> (int) (a + b)); // default implementation
    }
}
```

```java
final class Int256Vector extends IntVector<S256Bit> {
    private final int[] vec = new int[8];

    Int256Vector bOp(Vector<Integer, S256Bit> o, FBinOp f) { … }
}
```

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

```
package jdk.incubator.vector;
```



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

```
IntVector<S128Bit> z =
    x.add(y);
```

```
IntVector<S256Bit> z =
    x.add(y);
```

```
@HotSpotIntrinsicCandidate
IntVector::add(Vector)IntVector
```

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

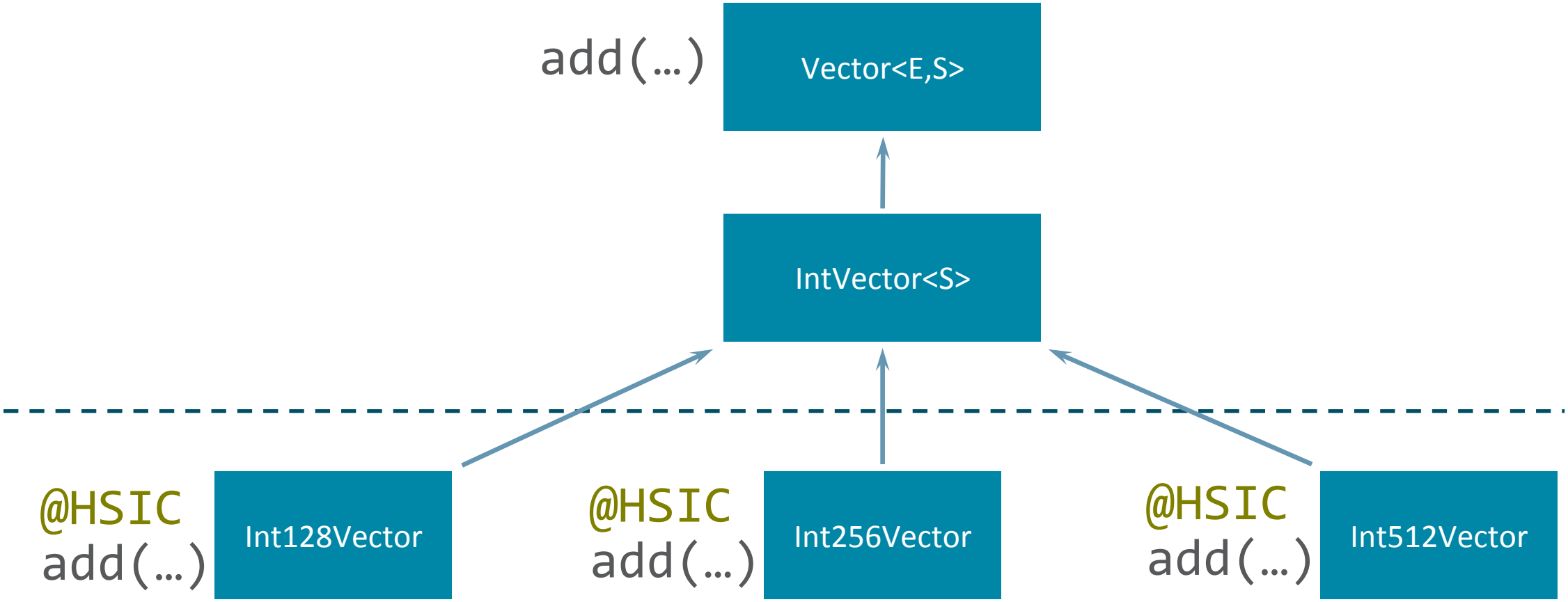
@HSIC IntVector::add(Vector)IntVector

Successful intrinsification requires:

1. Exact type of receiver
   - IntVector => Int256Vector
2. Exact type of the argument
   - Vector => Int256Vector
3. Both arguments have the same type

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

```
package jdk.incubator.vector;
```



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

hotspot/src/share/vm/classfile/vmSymbols.hpp

```
do_intrinsic(_VectorAddInt, jdk_incubator_vector_IntVector, …
```

```
do_intrinsic(_VectorAddInt128, jdk_incubator_vector_Int128Vector, …
do_intrinsic(_VectorAddInt256, jdk_incubator_vector_Int256Vector, …
do_intrinsic(_VectorAddInt512, jdk_incubator_vector_Int256Vector, …
```

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

panama/hotspot/src/share/vm/classfile/vmSymbols.hpp

```
/* _VectorLength should be first one in list as it is used as marker for beginning of Vector API methods */
do_intrinsic(_VectorLength, jdk_incubator_vector_Vector, length_name, void_int_signature, F_R)
do_intrinsic(_VectorConstantMask, jdk_incubator_vector_VectorSpecies, constant_mask_name, vector_constant_mask_sig, F_R)
do_intrinsic(_VectorTrueMask, jdk_incubator_vector_VectorSpecies, true_mask_name, vector_make_mask_sig, F_R)
do_intrinsic(_VectorFalseMask, jdk_incubator_vector_VectorSpecies, false_mask_name, vector_make_mask_sig, F_R)
do_intrinsic(_VectorMaskAllTrue, jdk_incubator_vector_VectorMask, all_true_name, void_boolean_signature, F_R)
do_intrinsic(_VectorMaskAnyTrue, jdk_incubator_vector_VectorMask, any_true_name, void_boolean_signature, F_R)
do_intrinsic(_VectorMaskRebracket, jdk_incubator_vector_VectorMask, rebracket_method_name, vector_mask_rebracket_sig, F_R)
do_intrinsic(_VectorZeroFloat, jdk_incubator_vector_FloatVector_FloatSpecies, zero_vector_name, vector_float_zero_sig, F_R)
do_intrinsic(_VectorBroadcastFloat, jdk_incubator_vector_FloatVector_FloatSpecies, broadcast_vector_name, vector_float_broadcast_sig,
do_intrinsic(_VectorZeroInt, jdk_incubator_vector_IntVector_IntSpecies, zero_vector_name, vector_int_zero_sig, F_R)
do_intrinsic(_VectorBroadcastInt, jdk_incubator_vector_IntVector_IntSpecies, broadcast_vector_name, vector_int_broadcast_sig, F_R)
do_intrinsic(_VectorZeroDouble, jdk_incubator_vector_DoubleVector_DoubleSpecies, zero_vector_name, vector_double_zero_sig, F_R)
do_intrinsic(_VectorBroadcastDouble, jdk_incubator_vector_DoubleVector_DoubleSpecies, broadcast_vector_name, vector_double_broadcast_s
do_intrinsic(_VectorLoadFloat, jdk_incubator_vector_FloatVector_FloatSpecies, load_vector_name, vector_float_load_sig, F_R)
do_intrinsic(_VectorStoreFloat, jdk_incubator_vector_FloatVector, store_vector_name, vector_float_store_sig, F_R)
do_intrinsic(_VectorLoadDouble, jdk_incubator_vector_DoubleVector_DoubleSpecies, load_vector_name, vector_double_load_sig, F_R)
do_intrinsic(_VectorStoreDouble, jdk_incubator_vector_DoubleVector, store_vector_name, vector_double_store_sig, F_R)
do_intrinsic(_VectorLoadInt, jdk_incubator_vector_IntVector_IntSpecies, load_vector_name, vector_int_load_sig, F_R)
do_intrinsic(_VectorStoreInt, jdk_incubator_vector_IntVector, store_vector_name, vector_int_store_sig, F_R)
do_intrinsic(_VectorLoadLong, jdk_incubator_vector_LongVector_LongSpecies, load_vector_name, vector_long_load_sig, F_R)
do_intrinsic(_VectorStoreLong, jdk_incubator_vector_LongVector, store_vector_name, vector_long_store_sig, F_R)
do_intrinsic(_VectorLoadShort, jdk_incubator_vector_ShortVector_ShortSpecies, load_vector_name, vector_short_load_sig, F_R)
do_intrinsic(_VectorStoreShort, jdk_incubator_vector_ShortVector, store_vector_name, vector_short_store_sig, F_R)
do_intrinsic(_VectorLoadByte, jdk_incubator_vector_ByteVector_ByteSpecies, load_vector_name, vector_byte_load_sig, F_R)
do_intrinsic(_VectorStoreByte, jdk_incubator_vector_ByteVector, store_vector_name, implCompress_signature, F_R)
do_intrinsic(_VectorEqualInt, jdk_incubator_vector_IntVector, equal_method_name, vector_cmp_sig, F_R)
do_intrinsic(_VectorLessThanInt, jdk_incubator_vector_IntVector, lessthan_method_name, vector_cmp_sig, F_R)
do_intrinsic(_VectorBlendInt, jdk_incubator_vector_IntVector, blend_method_name, vector_int_blend_sig, F_R)
…
do_intrinsic(_VectorAddFloat, jdk_incubator_vector_FloatVector, add_method_name, vector_float_bin_op_sig, F_R)
/* _VectorAddFloat should be kept as the last one here because it is tagged as ending the list of intrinsics */
```

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

How to reduce the number of JVM intrinsics?

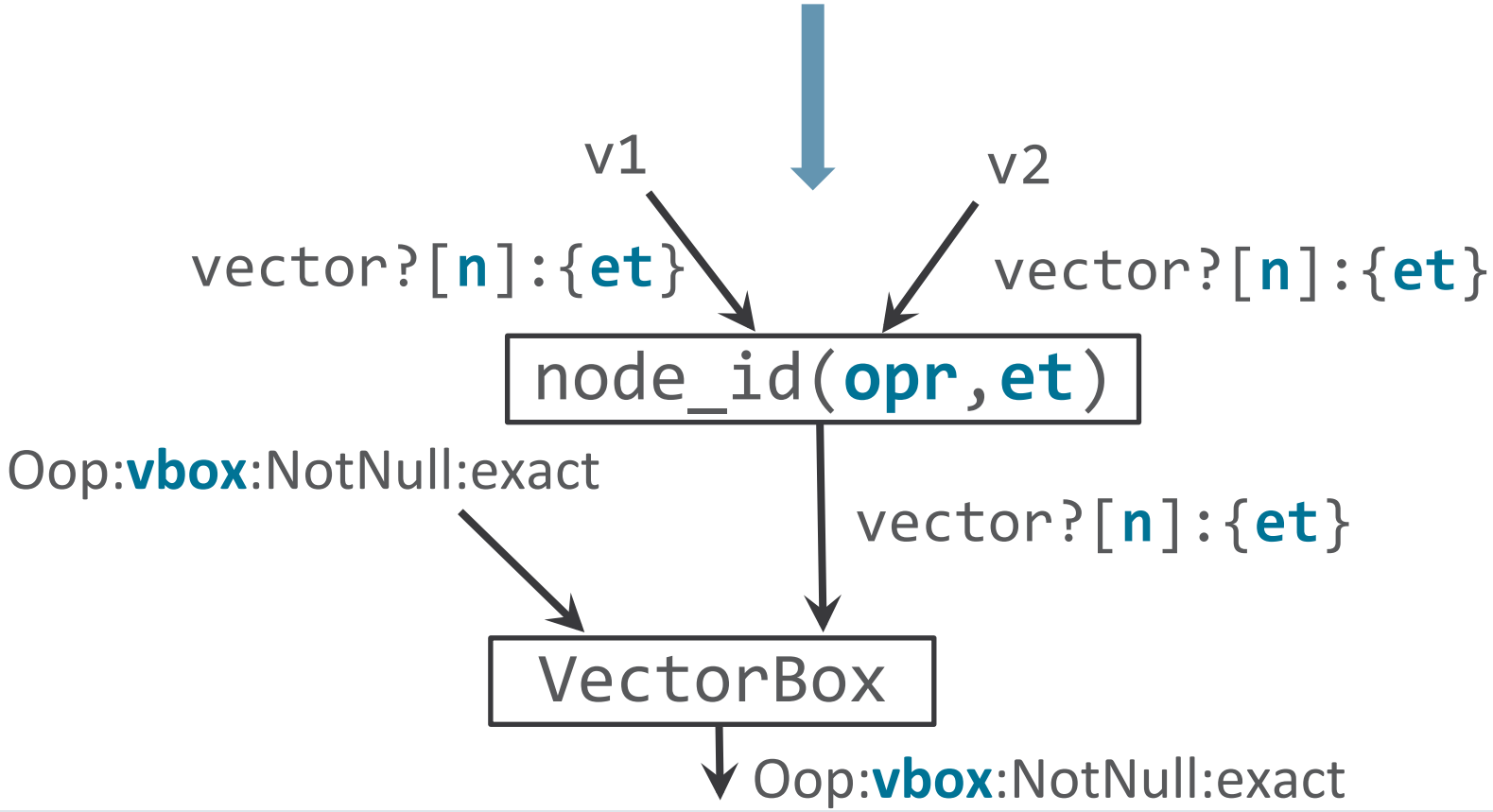
```
do_intrinsic(_VectorAddInt,
             jdk_incubator_vector_IntVector,
             add_method_name,
             vector_int_bin_op_sig, F_R)

do_intrinsic(_VectorSubFloat,
             jdk_incubator_vector_FloatVector,
             sub_method_name,
             vector_float_bin_op_sig, F_R)

do_intrinsic(_VectorMulDouble,
             jdk_incubator_vector_DoubleVector,
             mul_method_name,
             vector_double_bin_op_sig, F_R)
```

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

```java
package jdk.incubator.vector;

/*non-public*/ class VectorIntrinsics {

    @HotSpotIntrinsicCandidate
    static
    <V extends Vector<?,?>>
    V binaryOp(int       operatorId,
               Class<V> vectorClass,
               Class<?> elementType,
               int       vlen,
               V v1,
               V v2) {…}
```

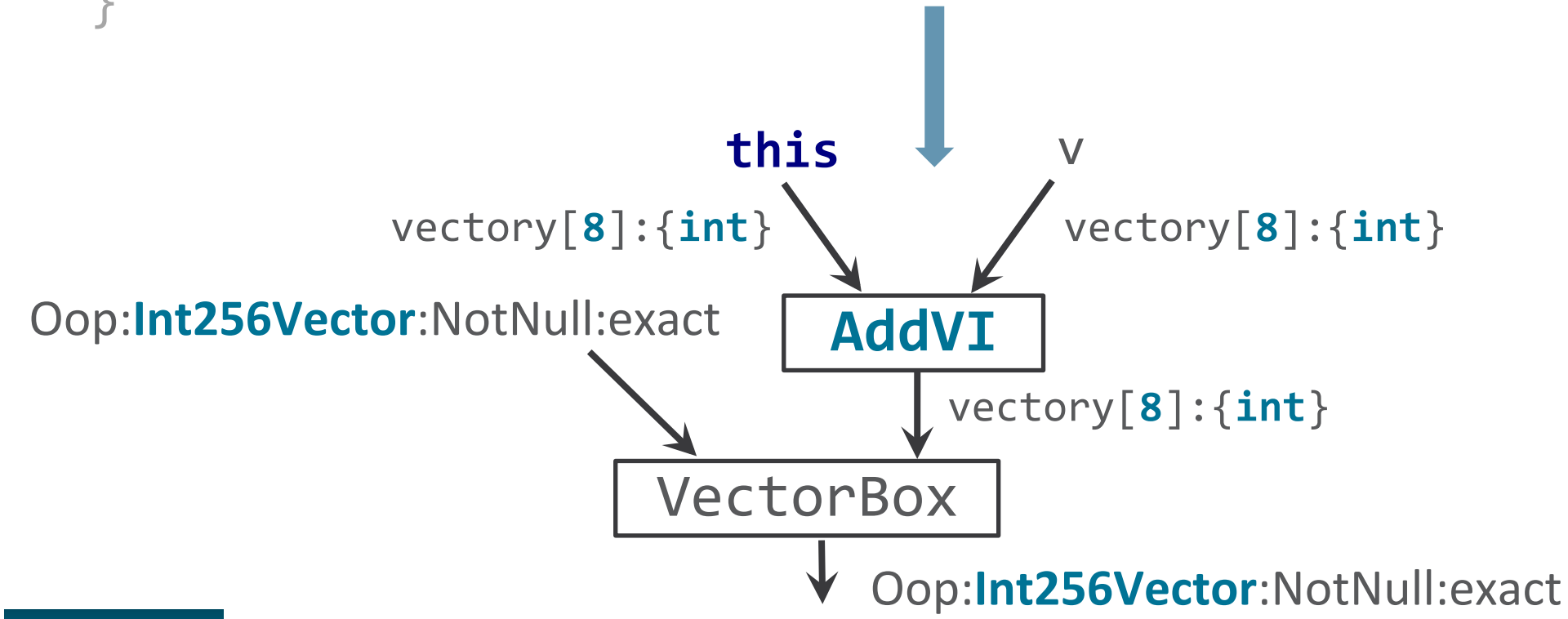
Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

```
@HotSpotIntrinsicCandidate
static <V extends Vector<?,?>>
V binaryOp(int opr, Class<V> vbox, Class<?> e, int n,
           V v1, V v2) {…}
```



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

```
final class Int256Vector extends IntVector<Shapes.S256Bit> {
    @ForceInline
    public Int256Vector add(Vector<Integer,Shapes.S256Bit> v) {
        return (Int256Vector) VectorIntrinsics.binaryOp(
          OP_ADD, Int256Vector.class, int.class, 8 // constants!
          (Int256Vector)this, (Int256Vector)v);
    }
```



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

What about default implementation?

```
@HotSpotIntrinsicCandidate
static <V extends Vector<?,?>>
V binaryOp(…, V v1, V v2,
           BiFunction<V,V,V> defaultImpl) {
    return defaultImpl.apply(v1, v2);
}


public Int256Vector add(Vector<Integer,Shapes.S256Bit> v) {
    return (Int256Vector) VectorIntrinsics.binaryOp(…,
           (v1, v2) -> {
                ((Int256Vector)v1).bOp(v2, (i, a, b) -> (a + b)
           });
}
```

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

VectorIntrinsics

```
1. Vector<E,S> unaryOp  (…, Vector<E,S> v, …)
2. Vector<E,S> binaryOp (…, Vector<E,S> v1, Vector<E,S> v2, …)
3. Vector<E,S> load     (…, Object array, int index, …)
4. void        store    (…, Object array, int index, Vector<E,S> v,…)
5. Vector<E,S> broadcast(…, long bits, …)
6. long        reduction(…, Vector<E,S> v, …)
7. Mask<E,S>   compare  (…, Vector<E,S> v1, Vector<E,S> v2, …)
8. boolean     test     (…, Vector<E,S> v1, Vector<E,S> v2, …)
…
```

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

Benefits

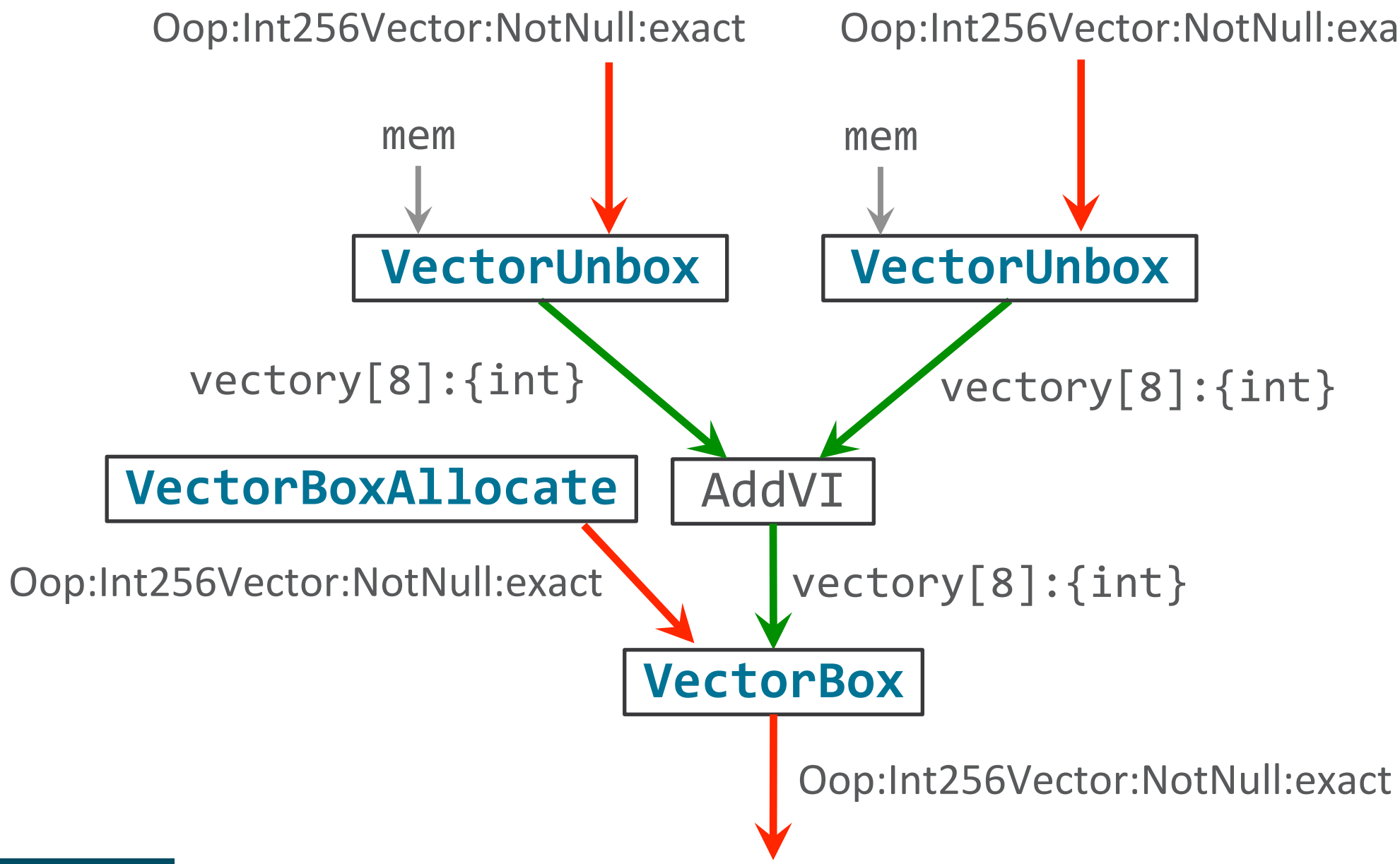
- Reliable intrinsification
  - all information needed is available right away
  - if inlining happens, then the intrinsification will always succeed
    - Vector::add()/IntVector::add() => Int256Vector::add() => VectorIntrinsics::binaryOp(..)

- Small amount of intrinsics needed
  - intrinsic-per-signature (erased)
  - masked operations can be covered with an optional mask argument
    - `Vector<E,S> unaryOp(…, Vector<E,S> v, Mask<E,S> m, …)`

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

Vector Box Elimination

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



C2 IR

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

Vector Box Elimination

Transformations

1. **Box/unbox elimination**

   VectorUnbox (VectorBox **box value**) => **value**

   LoadVector (VectorBox **box value**) => **value**

2. **Merge-through-phi**

   Phi (VectorBox **o1 v1**) (VectorBox **o2 v2**) => VectorBox (**Phi o1 o2**) (**Phi v1 v2**)

3. **Reboxing**

   VectorBox (**Phi o1 o2**) (**Phi v1 v2**) => VectorBox **new_box** (**Phi v1 v2**)

4. **Expansion**

   VectorBox **box value** => StoreVector **box value**

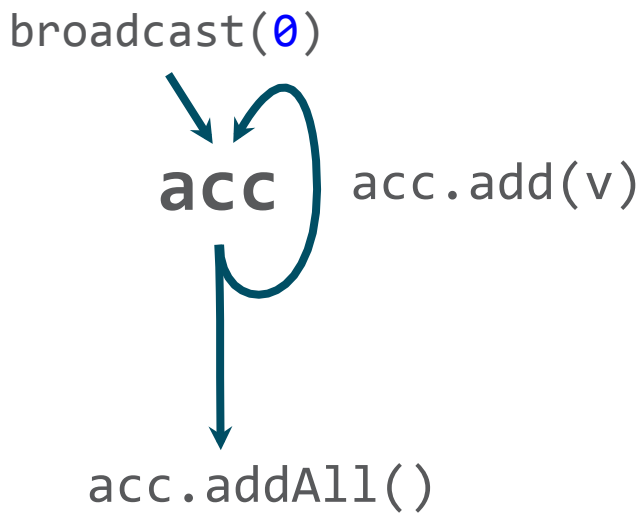
   VectorUnbox mem **box** => LoadVector **box**

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

Merge-through-phi

Example: Reduction

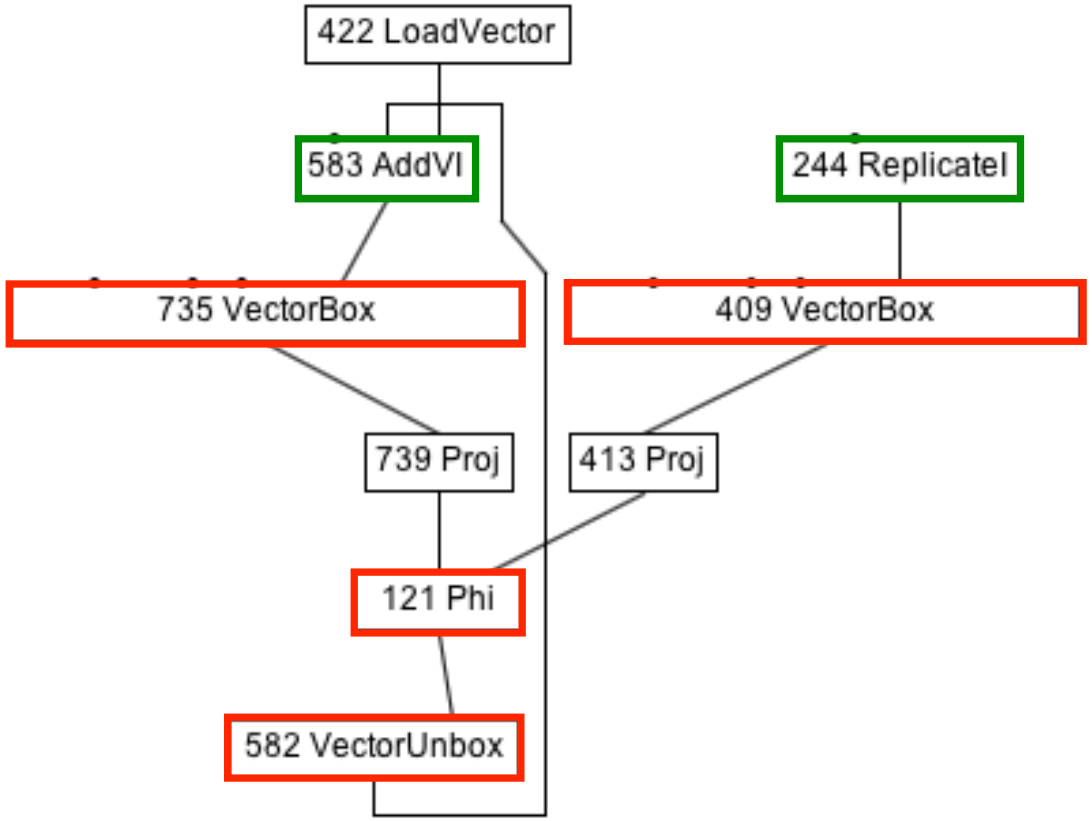
```
<S extends Vector.Shape<Vector<?,?>>>
int arraySum(int[] a, IntVector.IntSpecies<S> species) {
    IntVector<S> acc = species.broadcast(0);
    int step = species.length();

    for (int i = 0; i < a.length; i += step) {
        IntVector<S> v = species.fromArray(a, i);
        acc = acc.add(v);
    }
    return acc.addAll();
}
```

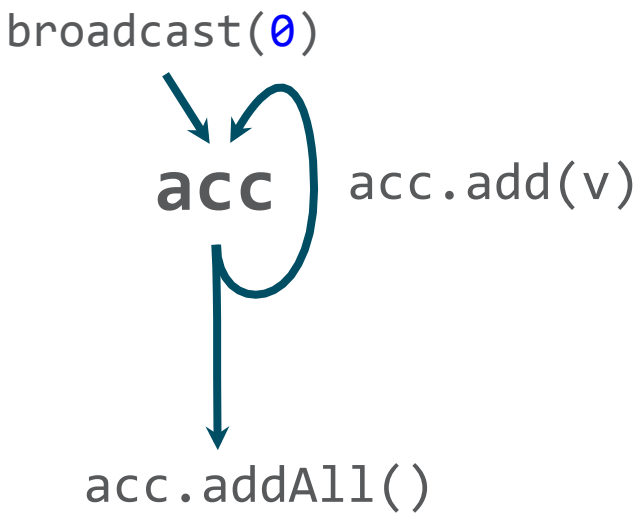


Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Merge-through-phi

Example: Reduction



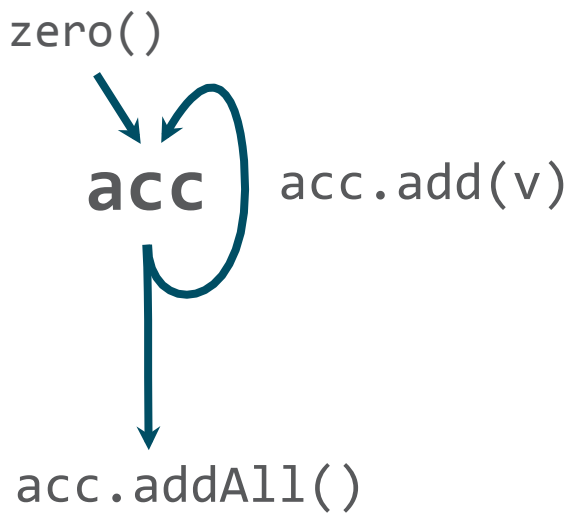


Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Aggressive Unboxing

Example: Reduction

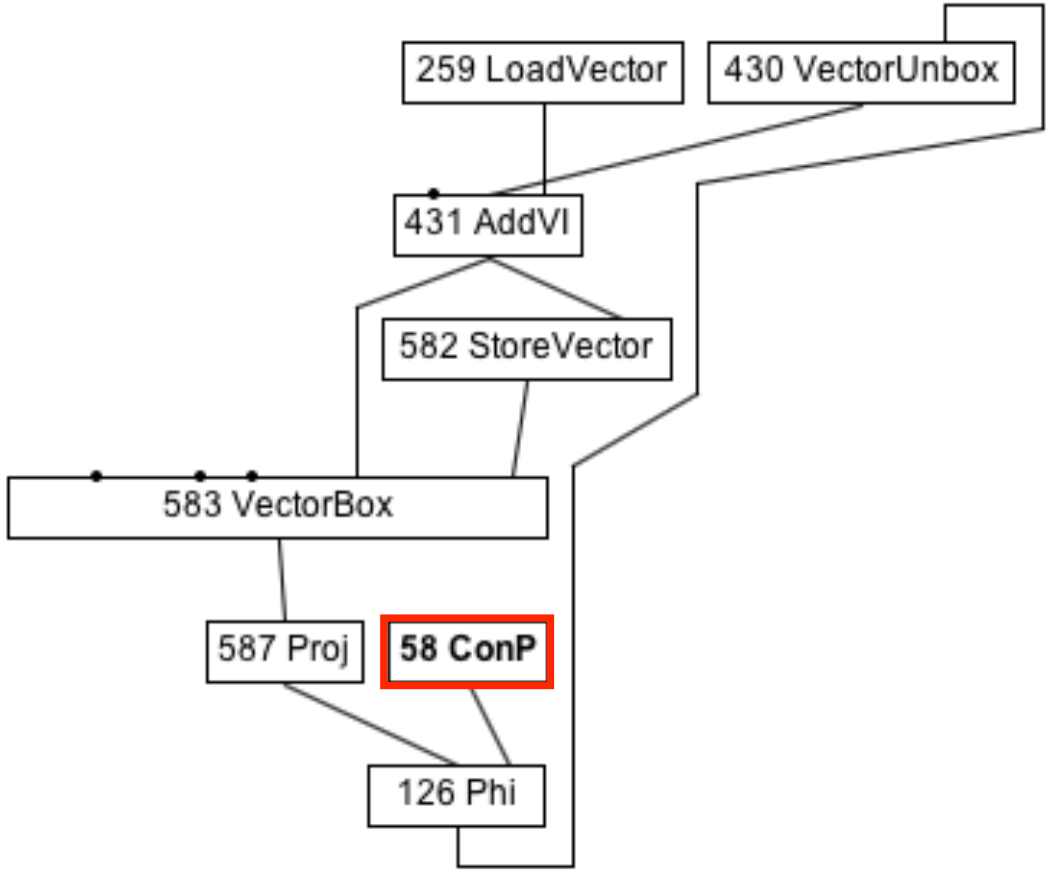
```
<S extends Vector.Shape<Vector<?,?>>>
int arraySum(int[] a, IntVector.IntSpecies<S> species) {
    IntVector<S> acc = species.zero();
    int step = species.length();

    for (int i = 0; i < a.length; i += step) {
        IntVector<S> v = species.fromArray(a, i);
        acc = acc.add(v);
    }
    return acc.addAll();
}
```

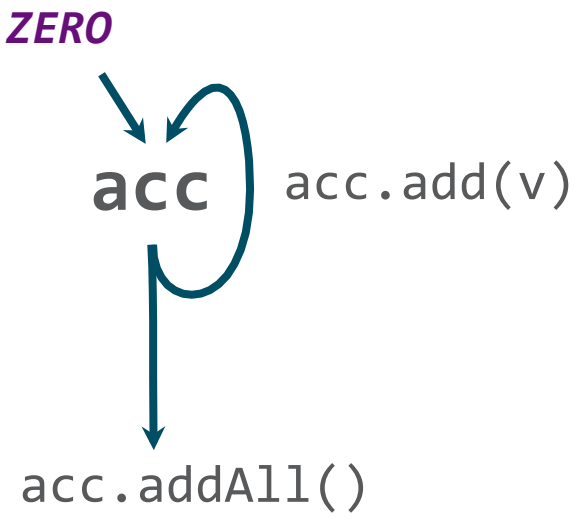


Copyright © 2017, Oracle and/or its affiliates. All rights reserved. |

Aggressive Unboxing

Example: Reduction





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

Reliable allocation materialization on reboxing?

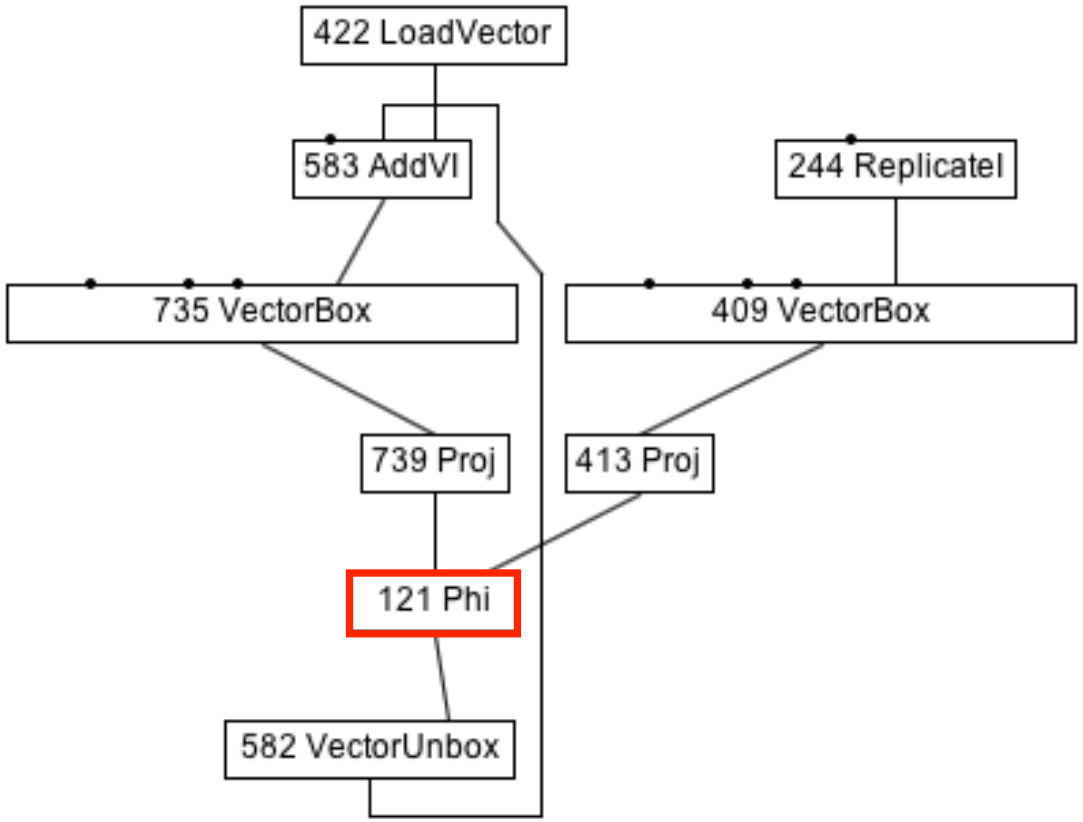
Phi (VectorBox **o1** v1) (VectorBox **o2** v2)

=>

VectorBox (**Phi o1 o2**) (Phi v1 v2)

=>

VectorBox **new_box** (Phi v1 v2)



- Needed for the case when (Phi o1 o2) leaks
  - 1 box instead of 2 allocated

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

Other observations

C2 IR

1. Keeping VectorBox as CallNode can hinder some transformations
   - especially pushing it through Phis
   - no need to keep JVM state once all allocations are materialized
2. Lazily materialize vector stores
   - no need to VectorBox & StoreVector to coexist
   - stores can be inserted when doing VectorBox elimination
3. Turn StoreVectors into initialization stores
   - C2 doesn’t turn vector store into an array into initialization store
   - the most promising way would be to do that manually

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

Vector Box Elimination

Summary

- Rematerialization at safepoints
- Aggressive unboxing
- Expand VectorBoxes and rely on EA pass to get rid of unused boxes

- Open question:
  - How to materialize boxes at use sites?
  - Requires JVM state. Some use sites have JVM state attached (e.g., safepoints), some don't.
  - What to do when one is not available?

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

Vector Box Rematerialization

Transformations

- Can’t rely on EA
  - rematerialization happens during EA only on scalarizable objects

SafePoint … (VectorBox **box** **value**) …

=>

SafePoint … (**SafePointScalarObject** box_klass #value) … **value**

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

Vector Box Rematerialization

Worst Case: Call

```
...
<+48>: vmovdqu %ymm0,(%rsp)        ; spill
... allocation ...
<+67>: callq   _new_instance_Java
<+72>: ...
<+77>: vmovdqu (%rsp),%ymm1        ; fill
...
```

```
PcDesc(pc=<+67> offset=48 bits=4):
...
Locals
  - l0: obj[20]
  ...
Objects
    - 20: Int256Vector stack[0]
.
```

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

Panama

VectorBox / VectorUnbox

- used in Vector API intrinsics
- works on typed vectors
  - 2 objects: typed box + backing array
  - easy access to vector internals in non-optimized case
- **explicitly materializes boxes on intrinsification**
  - enables lazy box materialization
  - requires full JVM state

VBox

- used in Machine Code Snippets
- works on super-longs
  - 1 object: super-long embeds vector values
  - requires some hacks in C2 to fool alias analysis
- works with preallocated boxes
- **rematerialization on safepoints**
- experiment to enable aggressive reboxing during EA
  - status: **FAILED**
  - http://cr.openjdk.java.net/~vlivanov/panama/hbox/

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

Panama

VectorBox / VectorUnbox

…

Vbox

…

Valhalla

ValueTypeNode

- excellent buffer elimination support
  - buffers can reside both on-heap and in thread-local buffer
  - aggressive scalarization
  - rematerialization on safepoints
  - custom calling conventions for Q-types
- works only on Q-types
  - no identity-sensitive operations on buffers allowed
  - explicit boxing/unboxing in bytecode

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

Panama

VectorBox / VectorUnbox

…

Vbox

…

Valhalla

ValueTypeNode

- MVT: 1st iteration
  - works only on Q-type
- **"LWorld"**: 2nd iteration
  - use L-types instead of U-types
  - very close to the world of Vector API!
    - every vector lives in L-typed slot
  - difference: identity-sensitive operations will throw exceptions at runtime

http://mail.openjdk.java.net/pipermail/valhalla-dev/2017-December/003631.html
http://mail.openjdk.java.net/pipermail/valhalla-spec-experts/2017-November/000432.html

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

Other Directions

for improvements

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

Non-intrinsifiable operations

Other directions

- So far, we are focused on intrinsified case
- "Graceful performance degradation"
  - one of the main project goals
- Box elimination optimizations should work irrespective of intrinsification!

- May require changes in memory layout for typed vectors
  - having typed vector wrapper + backing array can hinder C2 ability to see boxed values
  - another approach: mimic ValueTypeNode and keep track of individual elements
    - VectorBox box v_n => VectorBox box $e_1 \ldots e_n$

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

Contributors

- Intel
  - Ian Graves (left Intel)
  - Razvan Lupusoru
  - Vivek Deshpande
  - Shravya Rukmannagari
- Oracle
  - John Rose
  - Paul Sandoz
  - Vladimir Ivanov

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

Materials/Links

- Generalized Intrinsics
  - http://cr.openjdk.java.net/~vlivanov/panama/vector.generalized_intrinsics.01/
- "On vector box elimination"
  - http://mail.openjdk.java.net/pipermail/panama-dev/2017-December/000840.html
- "MVT-based vectors: experiments with JVM support"
  - http://mail.openjdk.java.net/pipermail/valhalla-dev/2017-November/003555.html

Project Valhalla

- Minimal Value Types
  - http://cr.openjdk.java.net/~jrose/values/shady-values.html
- "LWorld prototype - initial brainstorming goals/prototyping steps"
  - http://mail.openjdk.java.net/pipermail/valhalla-dev/2017-December/003631.html
- "abandon all U-types, welcome to L-world (or, what I learned in Burlington)"
  - http://mail.openjdk.java.net/pipermail/valhalla-spec-experts/2017-November/000432.html

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. |