

THE QUEST FOR DYNAMIC LANGUAGE PERFORMANCE ON THE JVM

[NASHORN WAR STORIES]

(from a battle scarred veteran of invokedynamic)

Marcus Lagergren

Oracle

SOFTWARE DEVELOPMENT

gotocon.com



[NASHORN RANTS]

Marcus Lagergren

Oracle

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com



[JAVASCRIPT RANTS]

Marcus Lagergren

Oracle

INTERNATIONAL SOFTWARE DEVELOPMENT CONFERENCE

gotocon.com



"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 DECISION. THE DEVELOPMENT, RELEASE, AND TIMING OF ANY FEATURES OR FUNCTIONALITY DESCRIBED FOR ORACLE'S PRODUCTS REMAINS AT THE SOLE **DISCRETION OF ORACLE."**

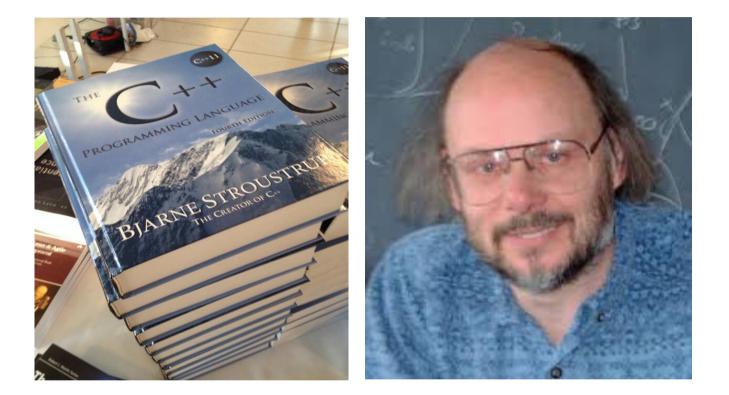




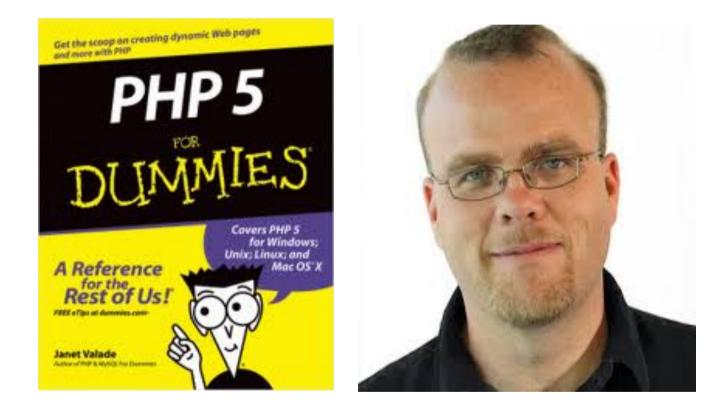
@lagergren

ORACLE























I am here to talk about...





I am here to talk about...

What we've suffered through so far to implement a dynamic language on the JVM





I am here to talk about...

What we've suffered through so far to implement a dynamic language on the JVM



The Nashorn Project



Also – a parade of JavaScript horrors







- What is Nashorn and why?
- The problem of compiling an alien language to Java [sic] bytecode
 - . Types
 - Optimistic assumptions
- . The JVM and its issues





What is Nashorn and why?



What is Nashorn?

- Nashorn is a 100% pure Java runtime for JavaScript
- Nashorn generates bytecode
 - Invokedynamics are everywhere
- Nashorn currently performs somewhere on the order of ~2-10x better than Rhino
- Nashorn is in JDK 8
- Nashorn is 100% ECMAScript compliant
- Nashorn has a well thought through security model



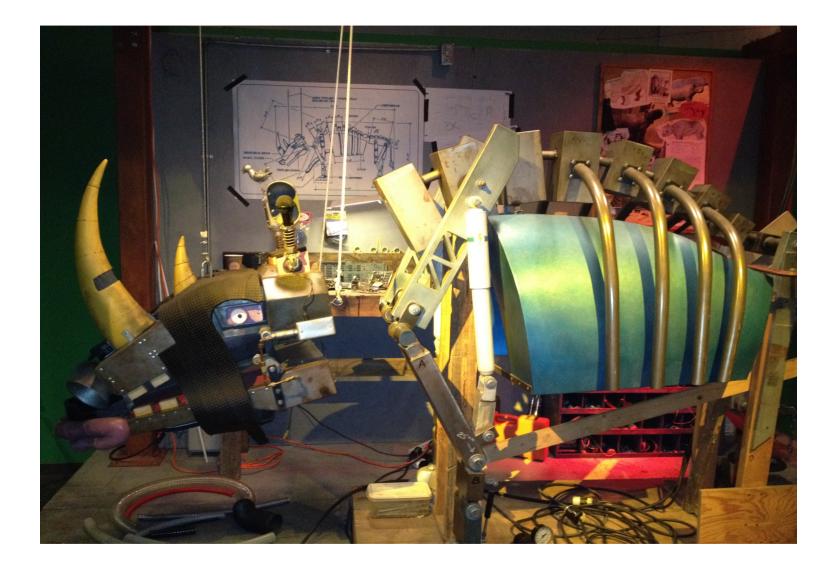


- Started as an invokedynamic POC.
- Rhino is still alive today after ~18 years. Why?
 - JSR-223
- Nashorn is now mature and replaces Rhino for Java 8



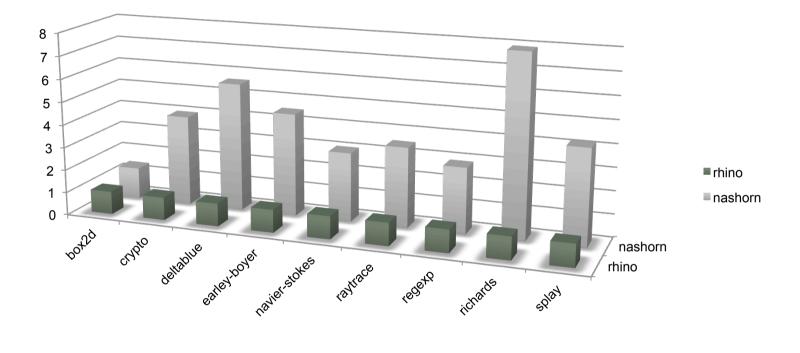






ORACLE













When is Nashorn available?

- Nashorn is part of OpenJDK8
 - Already available in JDK 8 builds.

```
> jjs
jjs> var x = "hello";
jjs> print(x);
hello
jjs>
```





Compiling an alien language to Java [sic] bytecode







. Scala is fairly good fit



- . Scala is fairly good fit
 - Yes I know: hard tail call optimization, interface injection etc.



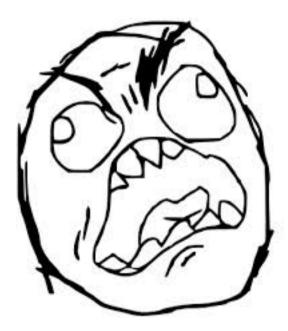
- . Scala is fairly good fit
 - Yes I know: hard tail call optimization, interface injection etc.
- Ruby and JavaScript are pretty bad fits



- Scala is fairly good fit
 - Yes I know: hard tail call optimization, interface injection etc.
- Ruby and JavaScript are pretty bad fits
 - No types
 - Things change at runtime. A lot.
 - Invokedynamic certainly alleviates a lot of the pain, but plenty of stuff remains to be solved







JavaScript!

Was it deliberately designed to make every efficient representation useless?





jjs> Array.prototype[1] = 17;





jjs> Array.prototype[1] = 17; 17 jjs>





jjs> Array.prototype[1] = 17;

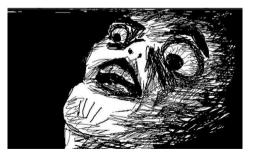
17

jjs> print([,,,]);





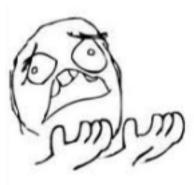
jjs> Array.prototype[1] = 17; 17 jjs> print([,,,]); ,17, jjs>





Let's talk about JavaScript - Numbers

- Numbers in JavaScript have no fixed ranges
- . "Intish". "Doublish".
- Not very nice for strongly typed bytecode
 - · Overflows must be handled
- Conservative: At least they tend to fit in Java doubles.





Let's talk about JavaScript - Numbers

- Double arithmetic is slower than integer arithmetic on modern HW
- But double arithmetic is sometimes *faster* than int arithmetic with the necessary overflow checks.
- · WAT!
- (getting back to that)



Let's talk about JavaScript – Types/Numbers

- HotSpot itself was originally tested and developed with bytecode that came from Java
- Representing everything as Objects to get the bytecode format type agnostic is nowhere near viable, performance wise.
- Boxing
- . Go primitive



- For bytecode performance we should
 - Use whatever static types we have
 - (mostly) done
 - Optimistically assume stuff about types
 - On it

CHALLENGE ACCEPTED





- JavaScript type coercion semantics and literals uses and definitions
- That's all the static type info we're going to get from the compiler
 - . Java int: statically enough for ~, &, |, ^
 - Java double: statically enough for: *, /, -, %
 - Object: binary + and pretty much everything else

•

Callsites, though. How do we deal with parameter types?

int square(int x) {
return x * x;
}
iload_0
dup
imul
ireturn



. But...

function square(x) {

return x * x;

jjs> square(2)

4

jjs> square(2.1)

4.41

```
jjs> square("a")
```

NaN



So conservatively...

square(Ljava/lang/Object;)D
aload_0
// hopefully just unbox:
invokestatic coerce2Double(Ljava/lang/Object;)D
dup
dmul // returns mul result, so always double
dreturn





• Guess again

jjs> square({

. . .

```
valueOf: function() {
```

```
global++;
```

```
return 2 + global; });
```



So conservatively...

square(Ljava/lang/Object;)D
aload_0
// hopefully just unbox:
invokestatic coerce2Double(Ljava/lang/Object;)D
dup
dmul // returns mul result, so always double
dreturn



sigh - well at least the return value HAS to be double

```
square(Ljava/lang/Object;)D
aload_0
invokestatic coerce2Double(Ljava/lang/Object;)D
aload_0
invokestatic coerce2Double(Ljava/lang/Object;)D
dmul // returns mul result, so always double
dreturn
```



JavaScript has a lot of magic in its number coercion

```
var dict = Object.create(null);
var key = `valueOf';
//later
dict[key] = formatHarddriveFunction;
//much later
dict++;
```





++[[]][+[]]+[+[]]

"10"



Brendan

ORACLE

ORACLE

```
Fibbonacci calculator
function fib( ) {
    for( =[+[],++[[]][+[]],+[], ], [++[++[++[[]][+[]]]
      [+[]]][+[]]]=((( [++[++[++[[]]][+[]]][+[]]][+[]]]-
       (++[[]][+[]])) \& (((--[[]][+[]]) >>> (++[[]][+[]]))))
      === ( [++[++[++[[]]]+[]]] [+[]]] [+[]]] -
      (++[[]][+[]])))?([++[++[[]][+[]]][+[]]]=
      ++[[]][+[]], [++[++[++[[]][+[]]][+[]]][+[]]]-
       (++[[]][+[]])):+[]; [++[++[++[[]][+[]]][+[]]]
      [+[]]] --; [+[]] = ([++[[]][+[]]] =
      [++[++[[]]][+[]]][+[]]] = [+[]] + [++[[]]][+[]]]) -
      [+[]]);
    return _[++[++[[]][+[]]];
```

- We can, and do, use static callsite types though.
- (ignore int overflows for a bit)

```
// Even if square is replaced, callsite type is not
// It always takes a number, always returns a number
var a = b * square(17.0);
```

ORACLE

- We can, and do, use static callsite types though.
- (ignore int overflows for a bit)

```
// Even if square is replaced, callsite type is not
// It always takes a number, always returns a number
var a = b * square(17.0);
```

```
square(D)D
dload 0
dup
dmul
dreturn
```



- We can, and do, use static callsite types though.
- (ignore int overflows for a bit)

```
// Even if square is replaced, callsite type is not
// It always takes a number, always returns a number
var a = b * square(17.0);
square = function(x) { return x + "string"; }
square(D)D
dload 0
dup
dmul
dreturn
```

ORACLE

square(Ljava/lang/Object;)Ljava/lang/Object;

aload O

ldc "string"

JS_ADD(Ljava/lang/Object;Ljava/lang/Object);Ljava/lang/Object;

areturn



square(Ljava/lang/Object;)Ljava/lang/Object;

aload O

ldc "string"

JS_ADD(Ljava/lang/Object;Ljava/lang/Object);Ljava/lang/Object;

areturn

```
revert_square(D)D
```

dload 0

```
coerceToJSObject(D)Ljava/lang/Object; # param filter
```

```
invokedynamic square(Ljava/lang/Object;)Ljava/lang/Object;
```

```
coerceToDouble(Ljava/lang/Object;)D
```

dreturn



square(Ljava/lang/Object;)Ljava/lang/Object;

aload O

ldc "string"

JS_ADD(Ljava/lang/Object;Ljava/lang/Object);Ljava/lang/Object;

areturn

revert_square(D)D
dload 0
coerceToJSObject(D)Ljava/lang/Object; # param filter
Nok dy amic sq rafe(Lj (a/lang Obj C)) jav / ang Obj ct DE
coerceToDouble(Ljava/lang/Object;)D
dreturn





Static compile time types bring us performance, [But they are too rare to take us all the way]



Type Specialization

```
function am3(i,x,w,j,c,n) {
 var this array = this.array;
 var w array = w.array;
 var xl = x \& 0 x 3 fff, xh = x >> 14;
  while (--n \ge 0) {
    var l = this array[i]&0x3fff;
    var h = this array[i++] >> 14;
    var m = xh*l+h*xl;
    l = xl*l+((m&0x3fff) <<14)+w array[j]+c;</pre>
    c = (1>>28) + (m>>14) + xh*h;
    w array[j++] = l&Oxffffff;
  return c;
```



Type Specialization – Prove ints

```
function am3(i,x,w,j,c,n) {
 var this array = this.array;
 var w array = w.array;
  var xl = x \& 0 x 3 fff, xh = x >> 14;
  while (--n \ge 0) {
    var l = this array[i]&0x3fff;
    var h = this_array[i++]>>14;
    var m = xh*l+h*xl;
    l = xl*l+((m&0x3fff)<<14)+w array[j]+c;</pre>
    c = (1>>28) + (m>>14) + xh*h;
    w array[j++] = l&0xfffffff;
  return c;
```



Type Specialization – Prove doubles

```
function am3(i,x,w,j,c,n) {
 var this array = this.array;
 var w array = w.array;
  var xl = x \& 0 x 3 fff, xh = x >> 14;
  while(--n >= 0) {
    var l = this array[i]&0x3fff;
    var h = this array[i++]>>14;
    var m = xh*l+h*xl;
    l = xl*l+((m&0x3fff) <<14)+w array[j]+c;</pre>
    c = (1>>28) + (m>>14) + xh*h;
    w array[j++] = l&0xfffffff;
  return c;
```



Static range analysis – fold doubles to ints

```
function am3(i,x,w,j,c,n) {
 var this array = this.array;
 var w array = w.array;
 var xl = x&0x3fff, xh = x>>14; // xl = max 32 bits, xh: 18 bits
 while(--n >= 0) {
   var l = this array[i]&0x3fff; // l max 12 bits
   var h = this_array[i++]>>14; // h max (32-14) = 18 bits
   var m = xh*l+h*xl; // will never overflow
   l = xl*l+((m&0x3fff) <<14)+w array[j]+c;</pre>
    c = (1>>28) + (m>>14) + xh + h;
   w array[j++] = l&0xfffffff;
 return c;
```

Static range analysis

```
function am3(i,x,w,j,c,n) {
 var this array = this.array;
 var w array = w.array;
 var x1 = x&0x3fff, xh = x>>14: // x1 = max 32 bits, xh 18 bits
                       while -
   var l = this array[i]&0x3fff; // l max 12 bits
   var h = this array[i++]>>14; // h max (32-14) = 18 bits
   var m = xh*l+h*xl,
                                         ver overflow
    1 = xl*l+((m&0x3fff) <<14)+w array[j]+c;</pre>
    c = (1>>28) + (m>>14) + xh + h;
   w array[j++] = l&0xfffffff;
 return c;
```

ORACLE



Do we need our own inlining as well?



Do we need our own inlining as well?

We can statically prove a few primitive numbers from callsites to am3.

Not from all of them.

Runtime callsite is really: (Ljava/lang/Object;IILjava/lang/Object;III)I Statically unprovable, though





- Just ignore all primitive types use boxing everywhere and axxx instructions
 - Way too slow. The JVM is nowhere near being able to cope with that amount of boxing, and probably never will





- Just ignore all primitive types use boxing everywhere and axxx instructions
 - Way too slow. The JVM is nowhere near being able to cope with that amount of boxing, and probably never will
 - Use what primitives we can
 - Definitely gives us performance, depending on the amount of statically provable primitives





- Just ignore all primitive types use boxing everywhere and axxx instructions
 - Way too slow. The JVM is nowhere near being able to cope with that amount of boxing, and probably never will
 - Use what primitives we can
 - Definitely gives us performance, depending on the amount of statically provable primitives
- Add static range checking
 - Gives us another 30% or so





- Just ignore all primitive types use boxing everywhere and axxx instructions
 - Way too slow. The JVM is nowhere near being able to cope with that amount of boxing, and probably never will
 - Use what primitives we can
 - Definitely gives us performance, depending on the amount of statically provable primitives
- Add static range checking
 - Gives us another 30% or so
- Augment CFG with usedef chains to establish param types

ORACL



But soon... static analysis won't get us further unless we build our own native JavaScript runtime





But soon... static analysis won't get us further unless we build our own native JavaScript runtime

Become adaptive/dynamic/optimistic



Statically provable callsites for am3

- (Object, int, Object, Object, double, int, Object)Object
- · (Object, Object, Object, Object, double, int, int)Object
- (Object, Object, double, Object, double, Object, double)Object
- (Object, Object, Object, Object, double, int, int)Object
- (Object, int, int, Object, double, int, Object)Object
- (Object, int, Object, Object, Object, int, Object)Object

- · (Object, int, int, Object, int, int, int)Object
- (Object, int, int, Object, int, int, int) Object
- (Object, int, int, Object, int, int, int)Object
- (Object, int, int, Object, int, int, int) Object
- (Object, int, int, Object, int, int, int)Object
- · (Object, int, int, Object, int, int, int) Object

- · (Object, int, int, Object, int, int, int) Object
- (Object, int, int, Object, int, int, int)Object
- (Object, int, int, Object, int, int, int) Object
- (Object, int, int, Object, int, int, int)Object
- (Object, int, int, Object, int, int, int) Object
- · (Object, int, int, Object, int, int, int)Object
- We know this when linking at runtime

- · (Object, int, int, Object, int, int, int) Object
- · (Object, int, int, Object, int, int, int)Object
- · (Object, int, int, Object, int, int, int)Object
- · (Object, int, int, Object, int, int, int)Object
- · (Object, int, int, Object, int, int, int) Object
- · (Object, int, int, Object, int, int, int)Object

- We know this when linking at runtime
- Use this signature to generate an optimistic version of am3, guard the types
- Just because it's int right now, doesn't mean it's not undefined later. Guard required.

- · (Object, int, int, Object, int, int, int) Object
- · (Object, int, int, Object, int, int, int)Object
- · (Object, int, int, Object, int, int, int)Object
- · (Object, int, int, Object, int, int, int)Object
- · (Object, int, int, Object, int, int, int)Object
- · (Object, int, int, Object, int, int, int)Object

- We know this when linking at runtime
- Use this signature to generate an optimistic version of am3, guard the types
- Just because it's int right now, doesn't mean it's not undefined later. Guard required.
- x2 Performance



We really want to use ints where we can

- x++ pessimistic: x is double (if no static range analysis can prove otherwise)
- Having a double as a loop counter is slow
 - Loop unrolling doesn't work for non integer strides
 - Factor ~50 in improvement if replacing with ints

```
function f() {
  var x = 0;
  while (x < y) {
     x++;
  }
  return x;
}</pre>
```



We really want to use ints where we can

- All non-bitwise arithmetic can potentially overflow
- The + operator is the worst, as it can take any object
- Experiment: TypeScript frontend
 - A lot more performance with no further mods
 - Nashorn performs well with known primitive int types

```
function f() {
  var x = 0;
  while (x < y) {
    x++; // dadd? iadd with overflow check?
  }
  return x;
}</pre>
```



Using ints, problem 1 of 2 – Overflow check overhead

```
static int addExact(int x, int y) {
    int result = x + y;
    if ((x ^ result) & (y ^ result) < 0) {
        throw new ArithmeticException("int overflow")
    }
    return result;
}
function f() {
    var x = 0;
    while (x < y) {
        x = addExact(x, 1);
    }
    return x;
}</pre>
```

This is actually pretty much as slow as the dadd alone Not sometimes, but often.



Solution: Intrinsify math operations

- . Java 8: addExact/subExact/mulExact
- Intrinsify them
- Basically and addExact is just

add eax, edx jo fail ret fail: //slow stuff

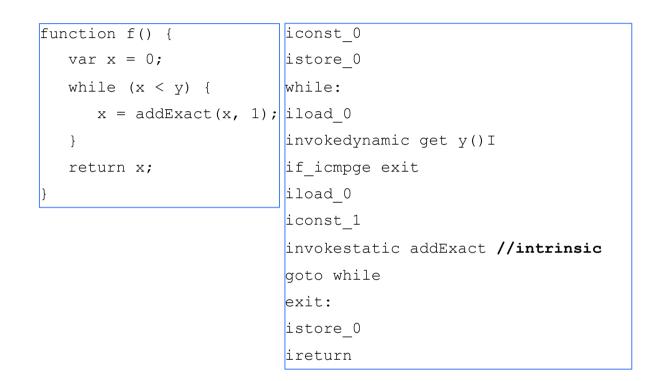
- < 10-15% slower than just the iadd when it doesn't fault</p>
- . Twice the speed of the non-intrinsified version with ${\tt xors}$
- Only slightly faster than dadd, but enables everything

ORACLE

Solution: Intrinsify math operations







This is almost native-fast with add intrinsic and the int specialization.

ORACLE

```
function f() {
                         iconst 0
  var x = 0;
                         istore 0
                         invokedynamic get y()I //check primitive
  while (x < y) {
     x = addExact(x, 1); istore 1
                         while:
                         iload 0
   return x;
                         iload 1 // y
                         if icmpge exit
                         iload 0
                         iconst 1
                         invokestatic addExact //intrinsic
                          goto while
                         exit:
                         istore 0
                          ireturn
```

(One more optimization: is y loop invariant? It may be a getter with side effects or anything as this is JavaScript hell... Hotspot won't be able to tell with the indy)



iconst 0 istore 0 invokedynamic get y()I //check primitive istore 1 while: iload O iload_1 // y if_icmpge exit iload_0 iconst 1 invokestatic addExact //intrinsic goto while exit: istore 0 ireturn

Native-fast



We really want to use ints where we can

Very common instance of same problem.

```
function f() {
   return 17 + array[3];
}
...
bipush 17
aload 2 //scope
invokedynamic get:array(Ljava/lang/Object;)Ljava/lang/Object;
aload 2
iconst_3
invokedynamic getElem(Ljava/lang/Object;I)Ljava/lang/Object;
invokedynamic ADD:OIO_I(ILjava/lang/Object;)Ljava/lang/Object;
areturn
```



We really want to use ints where we can

Very common instance of same problem.

```
function f() {
   return 17 + array[3];
}
...
bipush 17
aload 2 //scope
invokedynamic get:array(Ljava/lang/Object;)Ljava/lang/Object;
aload 2
iconst_3
invokedynamic getElem(Ljava/lang/Object;I)I
invokestatic Math.addExact
ireturn
```



Using ints problem 2 of 2 – erroneous assumptions

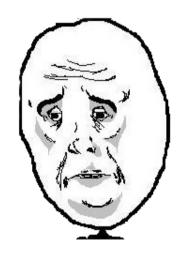
- So what do we do if we overflow or miss an assumption?
- Bytecode is strongly typed, so we can't reuse the same code
- . Throw errors or add guards/version code



Using ints problem 2 of 2 – erroneous assumptions

- So what do we do if we overflow or miss an assumption?
- Bytecode is strongly typed, so we can't reuse the same code
- Throw errors or add guards/version code

```
if (x < y) {
    x &= 1;
    if (x < 2) {
        x *= 2;
        if (k) {
            x += "string"
            //keep branching
        }
    }
return x; //hope this is an int</pre>
```





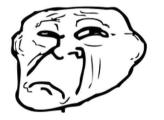


So add a catch block, take a continuation and jump to a less specialized version of the code





So add a catch block, take a continuation and jump to a less specialized version of the code





Continuations, you say?

Start out with

AL	OAD w_array
IL	OAD j
IN	VOKEDYNAMIC dyn:getElem(I)I
IA	DD

ORACLE

Continuations, you say?

Mark callsite optimistic, tag it with a program point

ALOAD w_array
ILOAD j
INVOKEDYNAMIC dyn:getElem(I)I [optimistic pp 17]
IADD





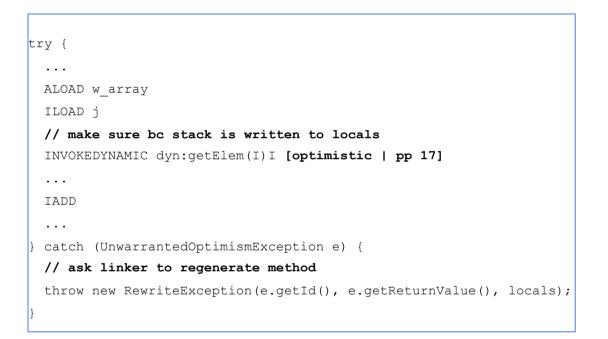
Add a return value filter throwing an Exception if we return a non-int type

public class UnwarrantedOptimismException extends Exception	ı {
<pre>public int getProgramRestartPointId() { };</pre>	
<pre>public Object getReturnedValue() { };</pre>	
}	



Continuations, you say?

Send a message to the caller to regenerate the method





Continuations, you say?

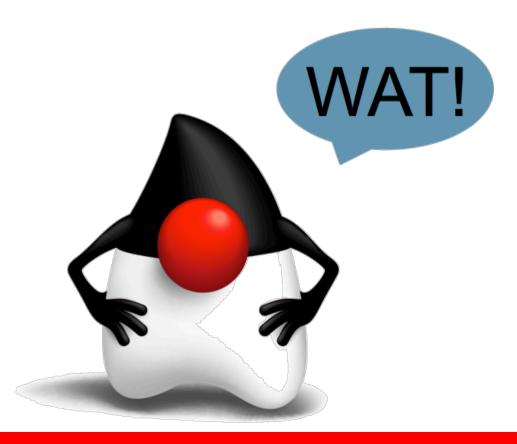
- We know when we are relinking a rewritable method
- Add a MethodHandles.catchException for RewriteException
- Catch triggers recompilation, with the failed callsite made more pessimistic.
- Also generates and invokes a "rest of" method

```
restOfMethod(RewriteException e) {
    // store to locals e.getLocals();
    // ...
    // all code after invokedynamic that failed with
    // maximum pessimism
    // (can never throw UnwarrantedOptimismException)
    return pessimisticReturnValue;
}
```





The JVM situation







- Java 7
 - Pretty quickly started giving us the infamous NoClassDefFoundError bug
 - Circumvented by running with everything in bootclasspath (Eww...)
- Java 8
 - A lot of C++ was reimplemented as LambdaForms
 - Initially, 10% of Java 7 performance. 🛞



₹ J New	_configuration [Java Application]		
V 🖉 jdk.nashorn.tools.Shell at localhost:54014			
	Thread [main] (Suspended (breakpoint at line 618 in NativeMath))		
	NativeMath.round(Object, Object) line: 618		
	LambdaForm\$DMH.invokeStaticInit_LL_L(Object, Object, Object) line: not available		
	LambdaForm\$DMH.invokeSpecial_LLL_L(Object, Object, Object, Object) line: not available		
	LambdaForm\$NamedFunction.invoke_LLL_L(MethodHandle, Object[]) line: 1102		
	LambdaForm\$DMH.invokeStatic_LL_L(Object, Object, Object) line: not available		
	LambdaForm\$NamedFunction.invokeWithArguments(Object) line: 1136		
	LambdaForm.interpretName(LambdaForm\$Name, Object[]) line: 625		
<pre>print(Math.round(0.5));</pre>	LambdaForm.interpretWithArguments(Object) line: 604		
$p_{\text{LIIIC}}(\text{Machi.LOund}(0.5)),$	LambdaForm\$LFI.interpret_L(MethodHandle, Object, Object, Object) line: not available		
	LambdaForm\$NamedFunction.invoke_LLL_L(MethodHandle, Object[]) line: 1102		
	LambdaForm\$DMH.invokeStatic_LL_L(Object, Object, Object) line: not available		
	LambdaForm\$NamedFunction.invokeWithArguments(Object) line: 1136		
	LambdaForm.interpretName(LambdaForm\$Name, Object[]) line: 625		
	LambdaForm.interpretWithArguments(Object) line: 604		
	LambdaForm\$LFI.interpret_L(MethodHandle, Object, Object, double) line: not available		
	LambdaForm\$DMH.invokeSpecial_LLLD_L(Object, Object, Object, Object, double) line: not available		
	LambdaForm\$NFI.invoke_LLLD_L(MethodHandle, Object[]) line: not available		
	LambdaForm\$DMH.invokeStatic_LL_L(Object, Object, Object) line: not available		
	LambdaForm\$NamedFunction.invokeWithArguments(Object) line: 1136		
	LambdaForm.interpretName(LambdaForm\$Name, Object[]) line: 625		
	LambdaForm.interpretWithArguments(Object) line: 604		
	LambdaForm\$LFI.interpret_L(MethodHandle, Object, Object, Object, double) line: not available		
	LambdaForm\$NFI.invoke_LLLD_L(MethodHandle, Object[]) line: not available		
WTF?	LambdaForm\$DMH.invokeStatic_LL_L(Object, Object, Object) line: not available		
	LambdaForm\$NamedFunction.invokeWithArguments(Object) line: 1136		
	LambdaForm.interpretName(LambdaForm\$Name, Object[]) line: 625		
	LambdaForm.interpretWithArguments(Object) line: 604		
	LambdaForm\$LFI.interpret_L(MethodHandle, Object, Object, double) line: not available		
	LambdaForm\$MH.linkToCallSite(Object, Object, double, Object) line: not available		
	Script\$b.runScript(ScriptFunction, Object) line: 1		
	LambdaForm\$DMH.invokeStatic_LL_L(Object, Object, Object) line: not available		
	LambdaForm\$MH.invokeExact_MT(Object, Object, Object, Object) line: not available		
	FinalScriptFunctionData(ScriptFunctionData).invoke(ScriptFunction, Object, Object) line: 512		
	ScriptFunctionImpl(ScriptFunction).invoke(Object, Object) line: 202		
	ScriptRuntime.apply(ScriptFunction, Object, Object) line: 346		
	Shell.apply(ScriptFunction, Object) line: 385		
	Shell.runScripts(Context, ScriptObject, List < String>) line: 314		
	Shell.run(InputStream, OutputStream, OutputStream, String[]) line: 178		
	Shell.main(InputStream, OutputStream, OutputStream, String[]) line: 142		
	Shell.main(String[]) line: 121		

ORACLE

 M. Section 1.00
 M. Section 1.00
 M. Section 2.00
 M. personal compact and an experimental or an experimental state or an experimental personal compact and an experimental and experimental and the second state of the sec a) periodi como laborarigitante en la como da como de la como d Serie Lang. Longers. Longers and Providents - Reservers, he had a provident d) production inter-control of the second Construction descently a) production inter-control entering in the feature interaction of a production production of a production of production of a production of Status, Links, Links, Links, et al. (2014). Status and the status an Salarian intere togetares of ergradies interest. Sale and Statu and J. Latter. Latter for a strain and status with draw a processing of the strain and stra a) Section (1998) Construction (1997) Const personal control total and a second providence of the second person of t THE ADDRESS OF THE ADDRESS OF THE PARTY OF T a) production, inter-control entering of the second sec TATALAND, INDER, LARDAN PROPERTY AND ADDRESS A Torta Canada, Landarda, Landard and Landard and Ballion Stationary and Station Station A. Section (1998) A state in a difference on a specific data (1999) A specific data (1999).
 A. Section (1999) A state in a difference on particular difference on a difference on a specific data (1999).
 A. Section (1999) A state in a difference of the difference Salarana inter- tokan bergeren werten bergeren bergeren beite bergeren berger Territory links to the providence of the links to the second state TATA AND ADDRESS AND ADDRESS OF TAXABLE PARTY ADDRESS train, much a cashed characteristic Section - International Conference on Confere

(b) Special (control of the second a produce come takeners interpretenting metricanteres, produce Senated inter-transference-senated and the contract pro-Second Construction and a second secon

THE LOCAL LANDFORMULA PROPERTY AND ADDRESS AND ADDRESS AND ADDRESS a) proc. Log. 1000. Log. Conference on the control of the contr a) per lang time tamperony termination tamperony a characterization of per lang time tamperony and per language and per language at per language tamperony and per language and per language at per language tamperony and per language and per language at per language tamperony and per language and per language at per language tamperony and per language at per language at per language tamperony and per language at per language at per language tamperony and per language at per language at per language tamperony and per language at per language at per language tamperony and per language at per language at per language tamperony and per language at per language at per language tamperony and per language at per language at per language tamperony and per language at per languag e jans tag tanak kantaparahika partan tanakat taganat padahaken jara tany e jans tag tanak kantapara tanggarakan kantapara jan tet def provinse internet interpretation (anterpretation), provinse
 provinse, interpretation (anterpretation), provinse, provinse
 provinse, interpretation (anterpretation), provinse, provinse
 provinse, interpretation (anterpretation), provinse, provinse a) proc. Log. Log. Log. Control of the processing of the proces d) Social and contract and second strain strain strain and a contract and a contract and a strain a) Series (a) Series (b) Series of the real particular consideration to the statement of the See togi toruk tokaravatany taravatan tarabitati ta tarabitati patenti tarabitati tarabitati tarabitati tarabitati tarabitati patenti see togi toruk tarabiyat tarabitati tarabitati patenti see togi toruk tarabiyat tarabiyatan tarabitati tarabitati patenti see togi toruk tarabiyat tarabiyatan tarabitati tarabitati patenti see togi toruk tarabiyatati tarabiyatan tarabitati patenti see togi toruk tarabiyatati tarabiyatati tarabitati tarabitati patenti see togi toruk tarabiyatati tarabiyatati tarabitati tarabitati patenti see togi toruk tarabiyatati tarabiyatati tarabitati tarabitati tarabitati patenti see togi toruk tarabiyatati tarabitati tarabitati tarabitati tarabitati patenti see togi toruk tarabiyatati tarabitati tarabitati tarabitati tarabitati tarabitati see togi toruk tarabiyatati tarabitati tarabitati tarabitati tarabitati tarabitati see togi toruk tarabiyatati tarabitati tarabitati tarabitati tarabitati see togi toruk tarabiyatati tarabitati tarabitati tarabitati tarabitati tarabitati see togi toruk tarabiyatati tarabitati tarabitati tarabitati tarabitati tarabitati see togi toruk tarabiyatati tarabitati tarabitati tarabitati tarabitati tarabitati see togi toruk tarabiyatati tarabitati tarabitati tarabitati tarabitati see togi tarabitati tarabitati tarabitati tarabitati tarabitati tarabitati tarabitati see togi tarabitati tarabitati tarabitati tarabitati tarabitati tarabitati tarabitati tarabitati tarabitati see togi tarabitati tarabi a) proc. Log. Long. Long. Conference on the second seco A Serie Concerning Contemporation and a contemporal processing of the contemporation of Design and the interpretation of the interpretation o 2 June Long, Longe Landerscharter Strendens Linder, MALE, Landerscharter, 2 June Long, Longe Landerscharter Strenders, Landerscharter, M. Landerscharter, 1999. d) proc. Long. These is an experimental intervention of the second se A personal control to an and the second COLUMN TWO IS NOT THE OWNER. a) proc. Long. Links. Landor VI. Schulp of Marco Landor VI. Spin Link, Source Links, Landor VI. Schulp of Marco Landor VI. Spin Link, Schult J. Schult, Landor VI. Schulp of Marco Landor VI. Schultz Schult Schult Schultz, Schultz Schultz, Schultz Schultz, Schult a) See Log. Lengt, Long And March Margaretta in Line and Langt Spin City Landauter, Spin Spin Log. Length, Landauter, Spin Spin Log. Length Spin City, Landauter, Spin City, a) pers lang latente tantateren langeren in termenten passed e tantatere persitere o pers lang latente tantateren interpret des passed e tantateres persitere et pers lang latente tantateren interpret della passed passed e tantatere passe lang latente tantateren interpret della passed passed passed en pers lang latente tantateren persiteren interpret passed passed passe passed en persitere tantateren persiteren interpret passed passed passed passed passed en persitere tantateren passed Jose Lag, Jose Lander-Contractive Contraction (Contractive Contractive)
 Jose Lag, Lander-Contractive Contractive Contractive

of the last state and state and state the second state and the second state THE LOOP BOUND AND ADDRESS OF ADDRESS OF ADDRESS ADDRE or the two series and relationship the control of a transmission was they be (4) 201-101 E. State And Annual State State Control on System 17, 201-201, 201-20 See tany array and second s second se (a) Section State Statements (prover) interaction & Distancements of Section Statements (Section Statements) (Section Statements) (Section). The last bride balance interpreting timester, protect at her tale mean antique interpreter require talespera processo at her tale mean antiquery descent interpret transmissioner; M. See Long. Brittle An M. See Long. Brittle An M. Halp. __Author. Add. M. See Long. Brittle M. See Long. Brittle the construction and to be been a second second A TANK LAND, MICHAEL The constraint parties, and an interest of the particular of the p Tark Land, Britshi ten tere arters anteres interpretantianters prater. at the task many antiques interpret interpret, interpret, protection of the task state and an antiput descent of the task of t a) Sen and annual and a sense of the set 2. Sale tang dening databarak interpreter dependent tantaken sena tenya sele tang dening databaraken seleken se seleken sel seleken se (a) Son tang distang dan pertambahan di kanangkan pertambahan pertam pertambahan pertam at the last mode setting a comparison of the particular terms the last of the last set in the setting and the See tany more conservation, reprint conservation, a conservation, A. D. Lang, M. Lang, M. Lang, and M. Lang, M 2. Sak tang ketak kabapatèn di Ketaran Internetati K. K. Kakapatèn M. Sak tang ketak kabapatèn kabapatèn di Ketak Manyakatèn Kabapatèn Juli Juli tang ketak kabapatèn kabapatèn kabapatèn kabapatèn Juli Juli tang ketak kabapatèn kabapatèn kabapatèn kabapatèn kabapatèn Juli Juli tang ketak kabapatèn kabapat Kabapatèn k Kabapatèn kabap A. Long article, and antibility of an end of the strength of the 2 See 1610 Brites Relative Completeneous Completeneous Sectors) 5 See 1610 Brites Relative Completence Sectors (Sectors) Section States and even an A THE LOSS BOARD AND AND AND A LOSS PARTY AND ADDRESS AND ADDRESS TO A LOSS AND THE TARK BOARD AND POSTATION TO DESCRIPTION AND ADDRESS AND ADDRES M. Seit and a state with the state of the st A. Details and a series attraction of a structure part of performance. Jour 2019, edited and a structure performance of the last story assesses interpretation to the second statements of (c) Set they have an entry in the part of the part of the set o at you have access and around the topology

top-level of script





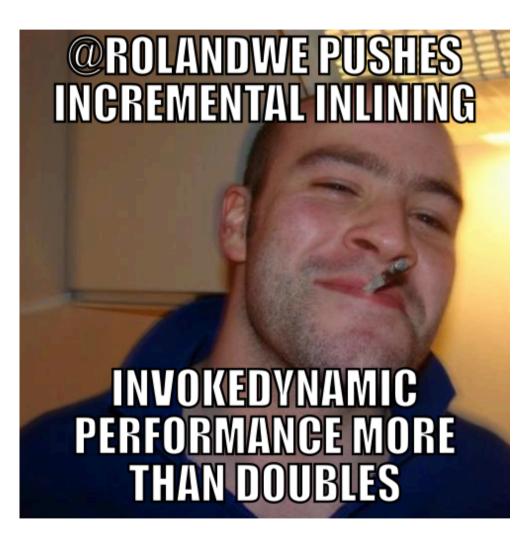


JVM issues

- Many inlining problems
 - Even, traditionally, for normal Java code add a code line, 50% of performance disappears
 - Seen that from time to time with HotSpot
 - Relevant in our quick paths in Nashorn too
- LambdaForms & MethodHandles
 - Tremendous pressure on inlining, lambda form classes also on metaspace
- Discovered a few very old bugs in C2 inliner
 - E.g: dead nodes counted as size.

















- LambdaForms compile a lot of code, generate a lot of metaspace stress
- If we have to have LambdaForms, they might not be able to remain in bytecode land?
- Inlining, despite tweaking has a lot of problems that remain to be solved
- Boxing removal boxing removal boxing removal
 - (probably enabled by local escape analysis)





MethodHandle.invoke (not exact) is slow

```
public class Test {
    private final static MethodHandle CALC =
        MethodHandles.publicLookup().findStatic(
            Test.class, "calc", int.class, int.class, Object.class);
    static int test() throws Throwable {
        MethodHandle mh = CALC;
        Object aString = "A";
        int a = mh.invoke(1, aString);
        int b = mh.invoke(2, "B");
        Integer c = mh.invoke((Integer)3, 3);
        return a+b+c;
    }
    static int calc(int x, Object o) {
        return x + o.hashCode();
    }
}
```





MethodHandle.invoke (not exact) is slow

```
public class Test {
    private final static MethodHandle CALC =
        MethodHandles.publicLookup().findStatic(
            Test.class, "calc", int.class, int.class, Object.class);
    static int test() throws Throwable {
        return 140;
    }
    static int calc(int x, Object o) {
        return x + o.hashCode();
    }
}
```





```
@Override
public long getLong(final long key) {
   final int index = ArrayIndex.getArrayIndex(key);
   final ArrayData array = getArray();
   if (array.has(index)) {
      return array.getLong(index);
   }
   return getLong(index, convertKey(key));
}
```





```
@Override
public long getLong(final double key) {
   final int index = ArrayIndex.getArrayIndex(key);
   final ArrayData array = getArray();
   if (array.has(index)) {
      return array.getLong(index);
   }
   return getLong(index, convertKey(key));
}
```





```
@Override
public long getLong(final Object key) {
   final int index = ArrayIndex.getArrayIndex(key);
   final ArrayData array = getArray();
   if (array.has(index)) {
      return array.getLong(index);
   }
   return getLong(index, convertKey(key));
}
```





```
@Override
public long getLong(final int key) {
    final ArrayData array = getArray();
    if (array.has(key)) {
        return array.getLong(key);
    }
    return getLong(key, convertKey(key));
}
```



War story: warmup

- Indy intrinsically needs bootstrapping
- Every call site contributes to warmup
- LambdaForms contribute to warmup
- Tiered compilation has gone back and forth.
 - Peak performance is reached sooner, even without C2 compiling all the methods
 - Added deviation has been very large
 - C2 is slow



Another war story: Metaspace

- Runtime didn't know about anonymous classes
- Build b58-b74 were broken ⊗
- Compressed klass pointers gave us a fixed size 100 MB default klass pointer chunk ⁽²⁾
- Metaspace allocated from metaspace pool subject to fragmentation. Chunks went 5% full to different classloaders
- HotSpot did not hand back dealloced Metaspace memory to the OS



- Optimistic code everywhere
- Static analysis/IR
- Field representations
 - Objects only, dual fields, sun.misc.TaggedArray (TaggedObject?)
- · Parallelism



Future work - JVM

- Boxing removal (probably requires Local EA)
- sun.misc.TaggedArray?
- Intrinsify Math.addExact and friends
 - Done!
- MethodHandle.invoke must be fast
- LambdaForms
 - Caching for footprint?
 - Replacing LambdaForms with something else?
 - . Get them out of class/bytecode land



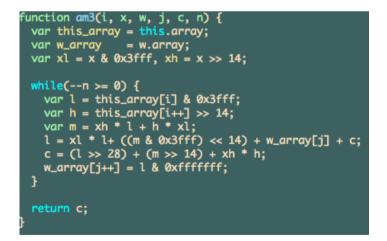
Future work - JVM

- Is bytecode even the correct format to do this entire in
 - Pluggable frontends?
 - More magic: I probably really need to talk to my compiler
 - Or have my compiler talk to me



Nashorn current performance status

- (Very) initial POC after 2.5 weeks of work:
 - Broke out octane.crypto.am3 the hotspot in the Crypto benchmark in octane.
 - Turned it into microbenchmark





Nashorn current performance status

• Runtime

- Rhino (with –opt 9): 34.6 s
 Nashorn tip: 10.8 s
- · V8 1.3 s



Nashorn with optimistic types

• Runtime

- Rhino (with –opt 9): 34.6 s
- Nashorn tip: 5.8 s
- · V8 1.3 s



Add JVM math intrinsics...

. Runtime

- Rhino (with –opt 9): 34.6 s
- Nashorn tip: 4.4 s
- · V8 1.3 s

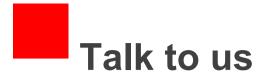


Patch JVM to keep more type info while inlining...

· Runtime

- Rhino (with –opt 9): 34.6 s
- Nashorn tip: 2.5 s
- · V8 1.3 s





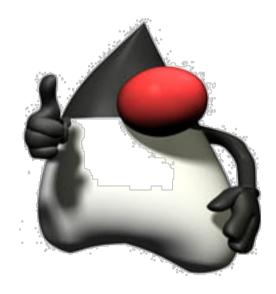
- Tweet us: @lagergren, @wickund, @asz,
 @hannesw, @sundararajan_a
- http://blogs.oracle.com/nashorn
- nashorn-dev@openjdk.java.net
- mlvm-dev@openjdk.java.net











@lagergren



