Renjin's Loop JIT
RIOT 2016
Standford University

renjin

Agenda

- What is Renjin?
- Quick project update
- JIT Loop Compiler

renjin

What is Renjin?

- New interpreter for R
- Core is written in Java
- Runs on JVM 7, no native library requirements
- Tool chain to build/convert existing CRAN and BioC packages

Project Update

Research Funding

- Collaboration CWI (2014-2015)
- Horizon 2020: “SOUND” (2015-2018)

renjin

GNU R Compatibility

	Package Count	Cumulative Percentage
A: All tests pass	1617	17%
B: Most tests pass	1011	28%
D: At least one test pass	1660	45%
F: Blocked/No tests passing	5071	
Total	9359	

	Package Count	Cumulative Percentage
A: All tests pass	22	2%
B: Most tests pass	47	6%
D: At least one test pass	112	16%
F: Blocked/no tests passing	915	

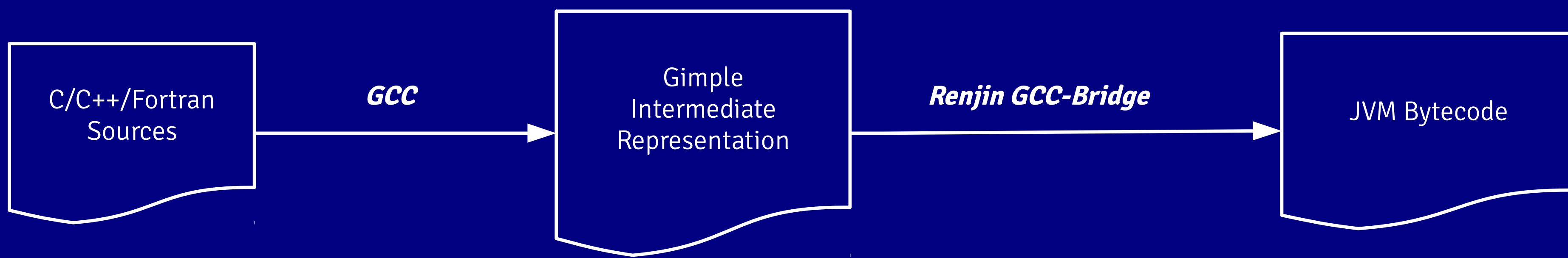
renjin

Compatibility Challenges

- S4: dispatching from primitives (ick)
- C/C++ unions (yuck)
- Details detail details…
  - Need more fine-precision unit tests

renjin

Native code with GCC-Bridge




```
SEXP *s;
double * p = REAL(s);
for(i=0;i<LENGTH(s);++i)
  p[i] = p[i]*2
```

```
    SEXP *s;
    double * p = call(REAL, s);
    int i = 0;

L1: int t1 = call(LENGTH, s);
    if(i < t1) goto L2
    goto L3

L2: double *t2 = pointer_plus(p, i)

    double  t3 = mem_ref(t2)
    double  t4 = t3 * 2.0
           *t2 = t4;
    goto L1

L3: return
```

```
    org.renjin.sexp.SEXP s;
    double[] p = GnuRApi.REAL(s);
    int i = 0;

L1: int t1 = GnuRApi.LENGTH(s);
    if(i < t1) goto L2
    goto L3

L2: double t2 = p;
    int    t2$offset = i;
    double t3 = t2[t2$offset]
    double t4 = t3 * 2.0
    t2[t2$offset] = t4;
    goto L1

L3: return
```

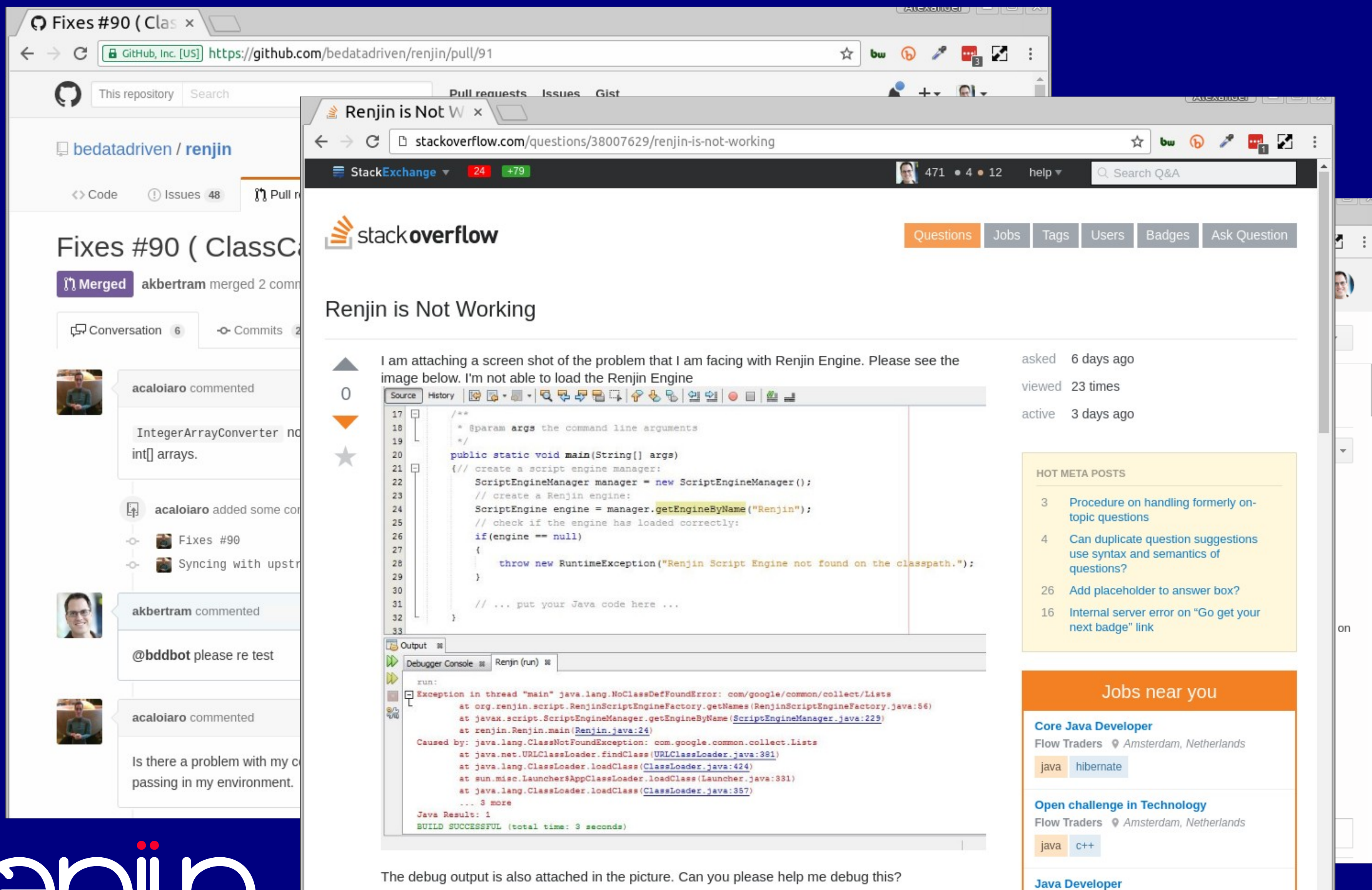
Improving support for C++

```
Terminal File Edit View Search Terminal Tabs Help
alex@alex-laptop76: ~   alex@alex-laptop76: /tmp   alex@alex-laptop76: ~/dev/cran/i...   alex@alex-laptop76: ~/dev/R-3.2....
        at org.renjin.repl.JlineRepl.run(JlineRepl.java:107)
        at org.renjin.maven.test.TestExecutor.executeTestFile(TestExecutor.java:176)
        at org.renjin.maven.test.TestExecutor.executeTest(TestExecutor.java:110)
        at org.renjin.maven.test.TestExecutor.execute(TestExecutor.java:86)
        at org.renjin.maven.test.TestExecutor.main(TestExecutor.java:51)
java.lang.NullPointerException
        at org.renjin.cran.intervals.reduce__._ZN8Endpoint15set_state_arrayEPA2_A2_Ki(reduce.cpp:48)
        at org.renjin.cran.intervals.reduce__._reduce(reduce.cpp:35)
        at org.renjin.cran.intervals.intervals._reduce(Unknown Source)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:497)
        at org.renjin.invoke.reflection.FunctionBinding$Overload.invoke(FunctionBinding.java:86)
        at org.renjin.invoke.reflection.FunctionBinding.invoke(FunctionBinding.java:141)
        at org.renjin.invoke.reflection.FunctionBinding.invoke(FunctionBinding.java:134)
        at org.renjin.primitives.Native.delegateToJavaMethod(Native.java:452)
        at org.renjin.primitives.Native.dotCall(Native.java:366)
        at org.renjin.primitives.R$primitive$$Call.apply(R$primitive$$Call.java:60)
        at org.renjin.eval.Context.evaluateCall(Context.java:282)
        at org.renjin.eval.Context.evaluate(Context.java:206)
        at org.renjin.primitives.special.AssignLeftFunction.assignLeft(AssignLeftFunction.java:60)
        at org.renjin.primitives.special.AssignLeftFunction.apply(AssignLeftFunction.java:44)
        at org.renjin.eval.Context.evaluateCall(Context.java:282)
```

GCC-Bridge vs JNI

- Avoid memory management complexities
- Pure JVM:
  - Safe to run in-process
  - Platform independence
  - Simplified deployment
- Opportunities for transformation
  - Global variable elimination (TODO)

Growing Community

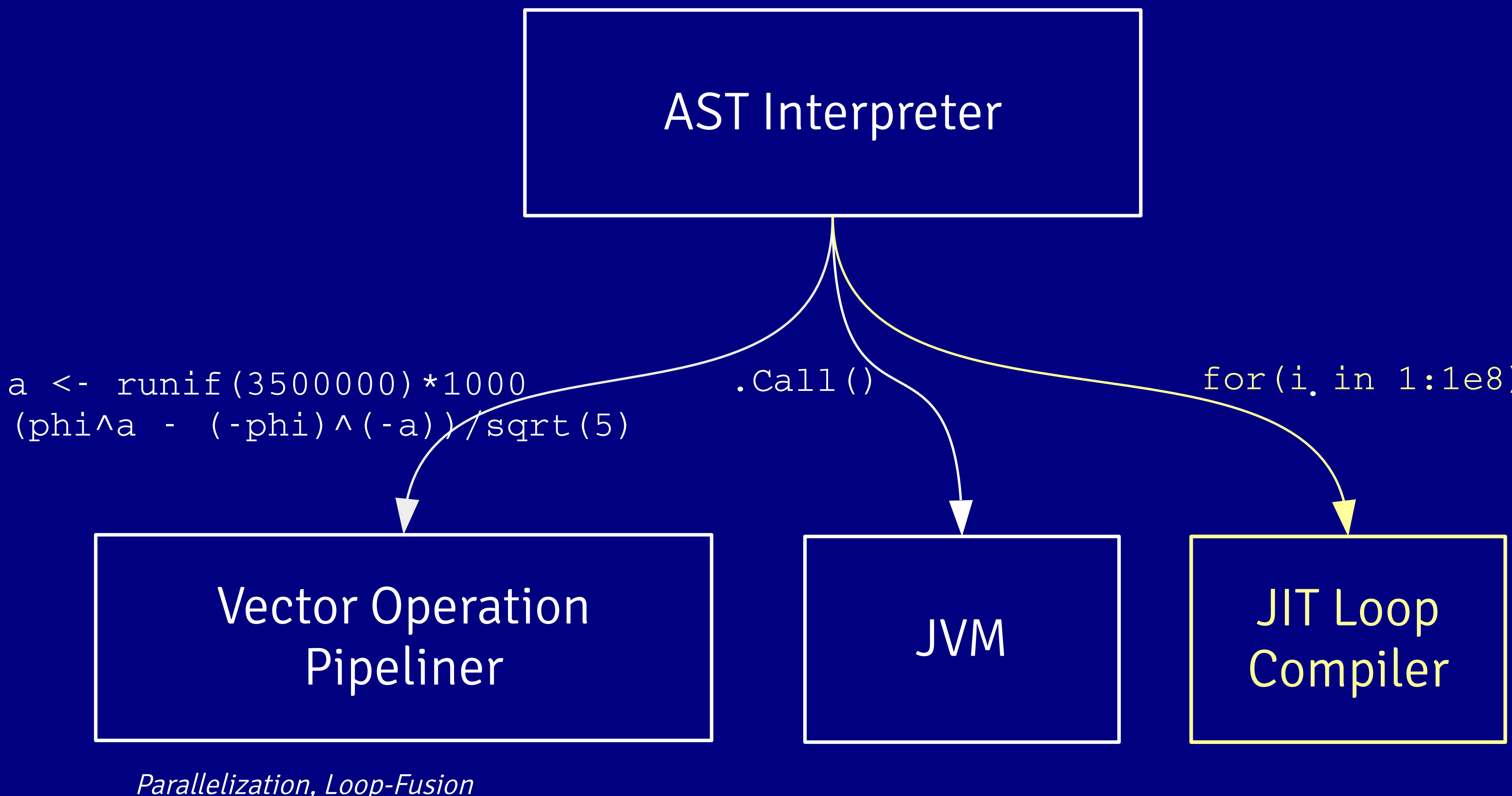


Take a moment to download Renjin. . . !

http://www.renjin.org/downloads.html

Introducing Renjin's JIT Compiler

Execution Modes



Compare:

Vector Operations	Loops

```
x <- 1:1e8
s <- sum(sqrt(x))
```

~ 10 R expressions evaluated

```
x <- 1:1e8
S <- 0
for(i in x)
  s <- s + sqrt(i)
```

~ 300m R expressions evaluated

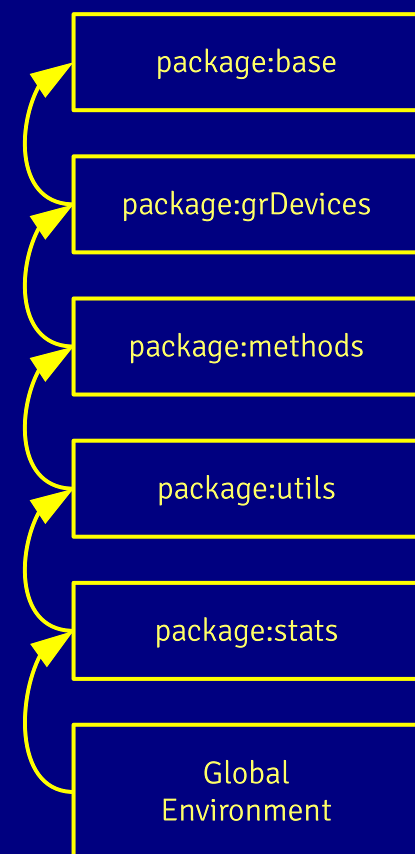
38 x slower

renjin

Function Loop

Function Lookup

```
s <- 0
for(i in 1:1e8) {
  s <- s + sqrt(i)
}
print(s)
```

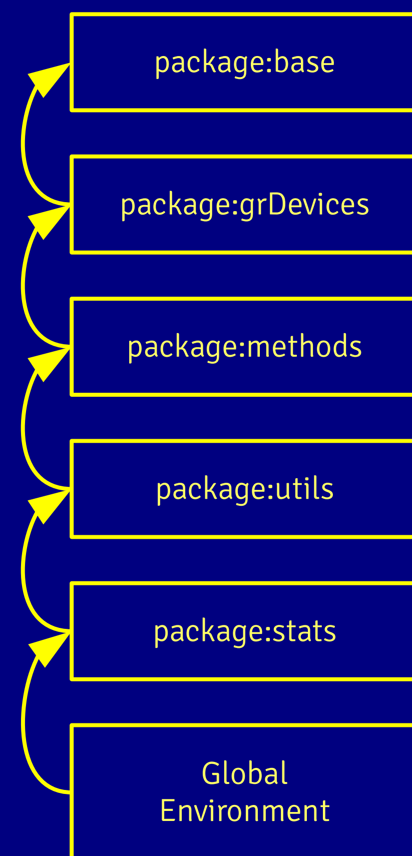


+ = .Primitive("+")
sqrt = .Prim("sqrt")

Function selection

Function Lookup

```
s <- 0
class(s) <- "foo"
for(i in 1:1e8) {
    s <- s + sqrt(i)
}
print(s)
```



+ = .Primitive("+")
sqrt = .Prim("sqrt")

"Boxing"

Boxing/Unboxing of Scalars

```
s <- 0
for(i in 1:1e8) {
   s <- s + sqrt(i)
}
print(s)
```

1

Two double-precision values stored in a register can be added with one processor instruction

1000s

SEXPs live in memory and must be copied back and forth, attributes need to be computed, etc. requiring 100s-1000s of cycles.

renjin

Function Calls

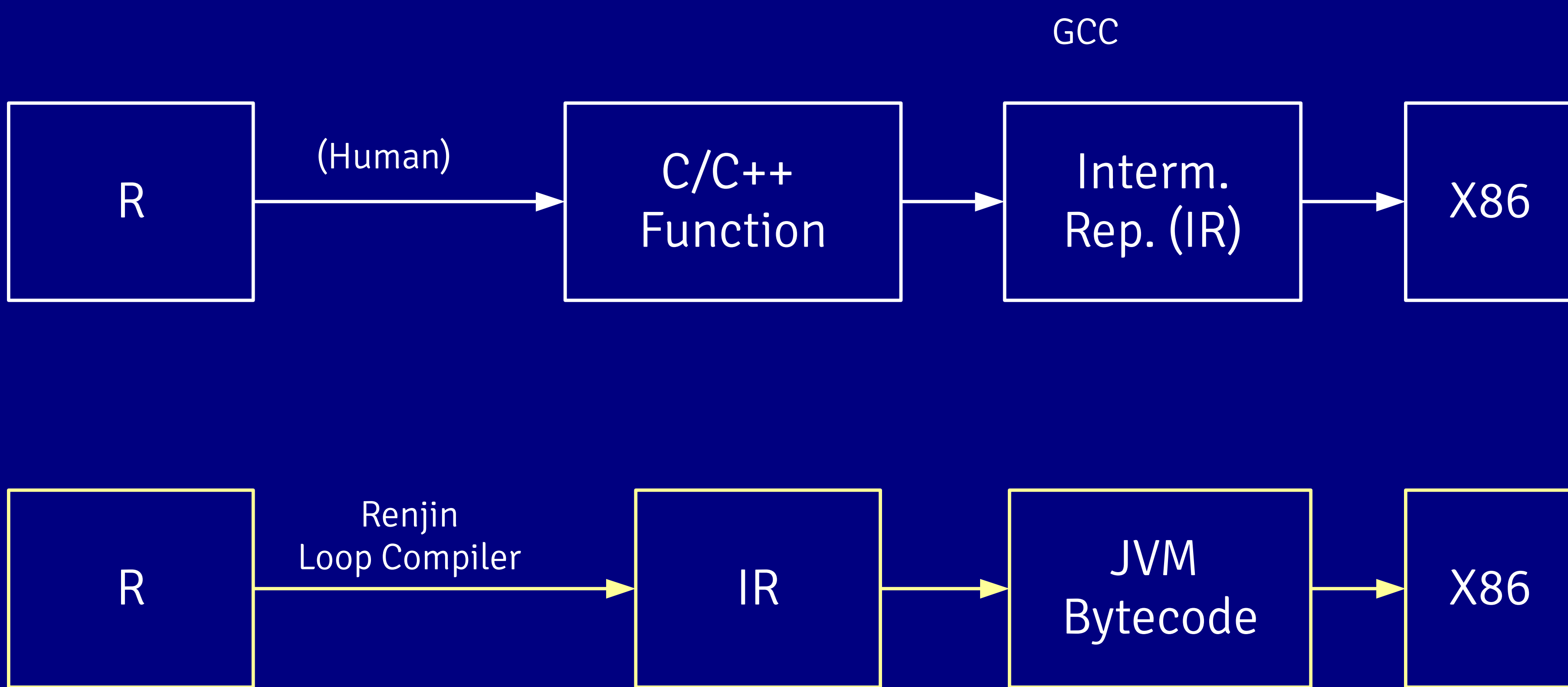
Function Calls are Expensive

```
s <- 0
cube <- function(x) x^3
for(i in 1:1e8) {
   s <- s + cube(i)
}
print(s)
```

TODO

1. Lookup cube symbol
2. Create pair.list of promised arguments
3. Match arguments to closure's formals pair.list (exact, partial, and then positional)
4. Create a new context for the call
5. Create a new environment for the function call
6. Assign promised arguments into environment
7. Evaluate the closure's body in the newly created environment.

Current Workaround



Step 1: Transform to 3AC

```
s <- 0
z <- 1:1e6

for(zi in z) {

        s <- s + sqrt(zi)

}
```

Assumptions recorded:

- “for” symbol = Primitive(“for”)
- “{“ symbol = .Primitive(“{“)
- “+” symbol = Primitive(“+”)
- “sqrt” symbol = Primitive(“sqrt”)

```
B1: z ← 1:1e6
    s ← 0
    i ← 0L
    temp1 ← length(z)

B2: if i ? temp1 goto B4

B3: zi ← z[i]
    temp2 ← (sqrt zi)
    s ← s + temp2
    index ← index + 1
    goto B2

B4: return (i, z, s)
```

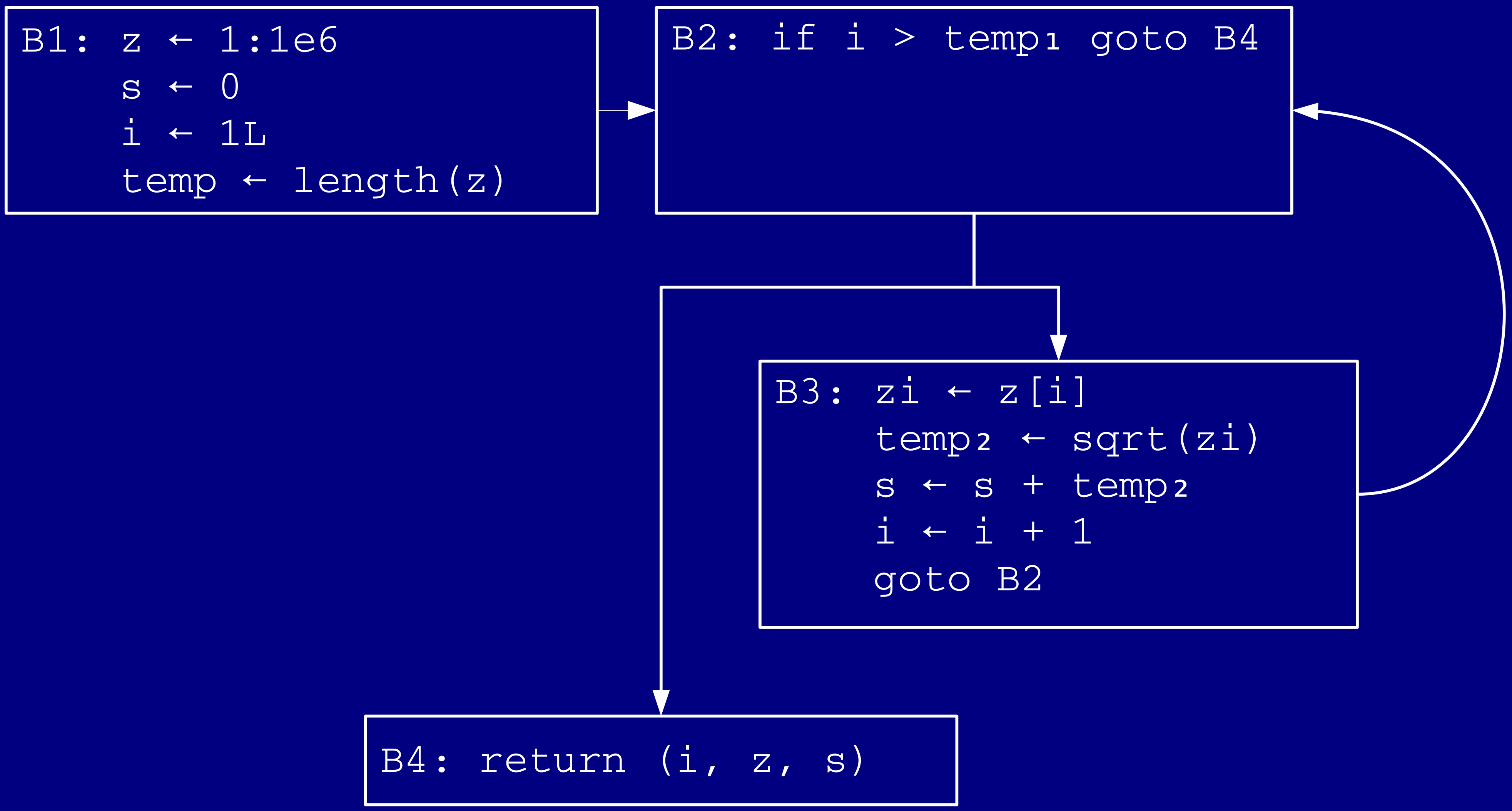
renjin

Handling Promises

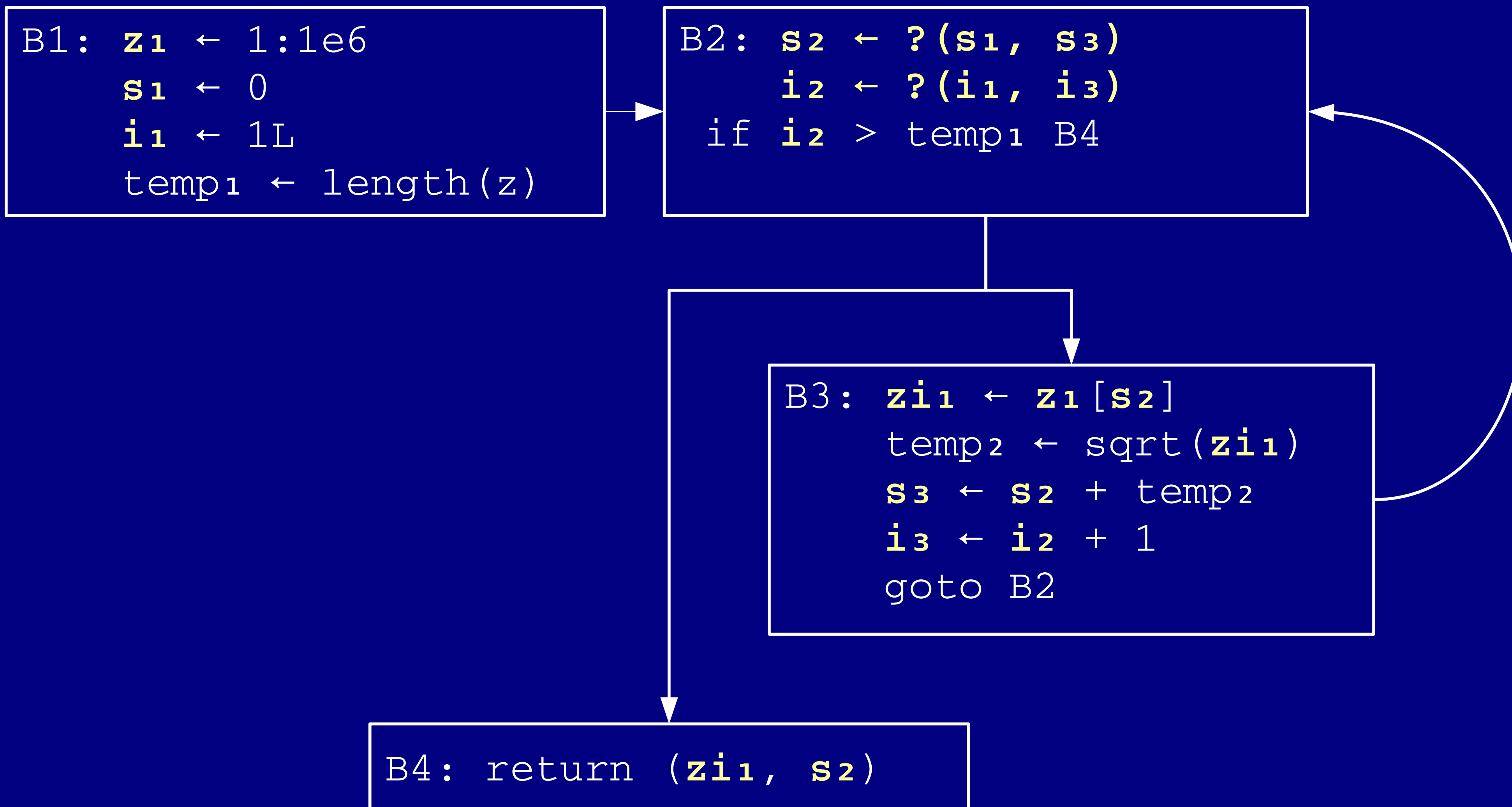
```
f <- function(x, y) {
  s <- 0
  for(i in x) {
    s <- s +
      if(i < 0)
        y
      else
        log(i)
   }
   s
}

f(1:10000)
```

- Run loop 5 times in interpreter to force as many promises as possible
- Currently bails if unevaluated promise is encountered.

Step 2: Build CFG



Step 3: Transform to SSA



renjin

Sparse Conditional Constant Propagation

- Propagate “ValueBounds”
  - ConstantValue: { SEXP | varies }
  - TypeSet: { bitmask of possible types }
  - Length: { 0, 1, … 2321 | varies }
  - Class: { constant | none | varies }
  - Attributes: { constant | none | varies }

Statically Computing Bounds

```
B1: z₁ ← 1:1e6                z₁    = 1:1e6
    s₁ ← 0                    S₁    = 0.0
    i₁ ← 1L                   I₁    = 1L
    temp₁ ← length(z₁)        temp₁ = 1e6L

B2: s₂ ← ?(s₁, s₃)
    i₂ ← ?(i₁, i₃)
 if i₂ > temp₁ B4

B3: zi₁ ← z₁[i₂]
    temp₂ ← sqrt(zi₁)
    s₃ ← s₂ + temp₂
    i₃ ← i₂ + 1
    goto B2

B4: return (zi₁, s₂)
```

renjin

Statically Computing Bounds

```
B1: z1 ← 1:1e6                 z1    = 1:1e6
    s1 ← 0                     s1    = 0.0
    i1 ← 1L                    I1    = 1L
    temp1 ← length(z1)         temp1 = 1e6L

B2: s2 ← ?(s1, s3)             s2    = s1 = 0.0
    i2 ← ?(i1, i3)             i2    = i1 = 1L
 if i2 > temp1 B4                    = 1L > 1e6L = false

B3: zi1 ← z1[i2]
    temp2 ← sqrt(zi1)
    s3 ← s2 + temp2
    i3 ← i2 + 1
    goto B2

B4: return (zi1, s2)
```

Statically Computing Bounds

```
B1: z1 ← 1:1e6                 z1     = 1:1e6
    s1 ← 0                     s1     = 0.0
    i1 ← 1L                    I1     = 1L
    temp1 ← length(z1)         temp1  = 1e6L

B2: s2 ← ?(s1, s3)             s2     = s1 = 0.0
    i2 ← ?(i1, i3)             i2     = i1 = 1L
 if i2 > temp1 B4                     = 1L > 1e6L = false

B3: zi1 ← z1[i2]               Zi1    = 1L
    temp2 ← sqrt(zi1)          temp2  = 1.0
    s3 ← s2 + temp2            s3     = 1.0
    i3 ← i2 + 1                i3     = 2L
    goto B2

B4: return (zi1, s2)
```

renjin

Statically Computing Bounds

```
B1: z₁ ← 1:1e6                       z₁     = 1:1e6
    s₁ ← 0                           s₁     = 0.0
    i₁ ← 1L                          I₁     = 1L
    temp₁ ← length(z₁)               temp₁  = 1e6L

B2: s₂ ← ?(s₁, s₃)                   S₂     = 0.0 ? 1.0 = num[1]
    i₂ ← ?(i₁, i₃)                   i₂     = 1L ? 2L = int[1]
 if i₂ > temp₁ B4                           = int[1] > 1e6 = T|F

B3: zi₁ ← z₁[i₂]                     Zi₁    = 1L
    temp₂ ← sqrt(zi₁)                temp₂  = 1.0
    s₃ ← s₂ + temp₂                  s₃     = 1.0
    i₃ ← i₂ + 1                      i₃     = 2L
    goto B2

B4: return (zi₁, s₂)
```

renjin

renjin.org

Statically Computing Bounds

```
B1: z1 ← 1:1e6                 z1     = 1:1e6
    s1 ← 0                     s1     = 0.0
    i1 ← 1L                    I1     = 1L
    temp1 ← length(z1)         temp1  = 1e6L

B2: s2 ← ?(s1, s3)             S2     = 0.0 ? 1.0 = num[1]
    i2 ← ?(i1, i3)             i2     = 1L ? 2L = int[1]
 if i2 > temp1 B4                     = int[1] > 1e6 = T|F

B3: zi1 ← z1[i2]               Zi1    = 1e6[ int[1] ] = num[1]
    temp2 ← sqrt(zi1)          temp2  = sqrt(num[1]) = num[1]
    s3 ← s2 + temp2            s3     = num[1] + num[1] = num[1]
    i3 ← i2 + 1                i3     = int[1] + int[1] = int[1]
    goto B2

B4: return (zi1, s2)
```

Statically Computing Bounds

```
B1: z₁ ← 1:1e6                    z₁     = 1:1e6
    s₁ ← 0                        s₁     = 0.0
    i₁ ← 1L                       I₁     = 1L
    temp₁ ← length(z₁)            temp₁  = 1e6L

B2: s₂ ← ?(s₁, s₃)                S₂     = 0.0 ? num[1] = num[1]
    i₂ ← ?(i₁, i₃)                i₂     = 1L ? int[1] = int[1]
 if i₂ > temp₁ B4                        = int[1] > 1e6 = T|F

B3: zi₁ ← z₁[i₂]                  Zi₁    = 1e6[ int[1] ] = num[1]
    temp₂ ← sqrt(zi₁)             temp₂  = sqrt(num[1]) = num[1]
    s₃ ← s₂ + temp₂               s₃     = num[1] + num[1] = num[1]
    i₃ ← i₂ + 1                   i₃     = int[1] + int[1] = int[1]
    goto B2

B4: return (zi₁, s₂)
```

Statically Computing Bounds

- We've computed types for all our variables
- Identified scalars that can be stored in registers
- Propagated constants to eliminate work
- Selected specialized methods for “+”, “sqrt”

renjin

Specialization via Metadata

```
@Pure
@Builtin("+")
@GroupGeneric("Ops")
@Vectorized(PreserveAttributeStyle.ALL)
public static double plus(double x, double y) {
 return x + y;

@Pure
@Builtin("+")
@GroupGeneric("Ops")
@Vectorized(PreserveAttributeStyle.ALL)
public static int plus(int a, int b) {
 // check for integer overflow
 int r = a + b;
 boolean bLTr = b < r;
 if (a > 0) {
  if (bLTr) {
   return r;
  }
 } else {
  if (!bLTr) {
   return r;
  }
 }
 return IntVector.NA;
```

Check...

- Bail if method selection results in unpredictable behavior:

```
s <- 0
z <- 1:1e6

for(zi in z) {
 s <- s + eval(readLines(stdin()))
}
```

...One last check

- Check for function assignments that would invalidate our initial compilation

Bail...

```
s <- 0
z <- 1:1e6

for(zi in z) {
 s <- s + sqrt(zi)
 if(zi < 0) `+` <- `-`
}
```

OK...

```
s <- 0
z <- 1:1e6

for(zi in z) {
 s <- s + sqrt(zi)
 `+` <- 42
}
```

Finally, Compile

- Remove phi statements
- Compile to JVM bytecode and run
- JVM compiles to machine code

```
double s1 = 0
int index1 = c(0L)
int ?1 = length(z1) 1e6
double s2 = s1
index2 = index1

BB2: if index2 >= ?1 goto BB4

BB1
double i3 = z1[index2]   (double)index2+1
double ?2 ← (sqrt i3)
double ?3 ← (length z1) 1e6
double ?4 ← (/ 1.0 ?3)
double ?5 ← (* ?2 ?4)
double s3 ← (+ s2 ?5)
int index3 ← index2 + 1
double s2 ← s3
int index2 ← index3
goto BB2

BB4:
rho.setVariable("i",  IntVector.of(i2))
rho.setVariable("s",  DoubleVector.of(s2))
return
```

renjin

Timings

```
f1 <- function(x) {
  s <- 0
  for(i in x) {
    s <- s + sqrt(i)
  }
  return(s)
}
```

	`f(1:1e6)`	`f(1:1e8)`
GNU R 3.2.0	0.255	25.637
+ BC	0.130	12.503
Renjin+JIT	0.107	1.114

renjin

Timings

```
f2 <- function(x) {
  s <- 0
  class(x) <- "foo"
  for(i in x) {
    s <- s + sqrt(i)
  }
  return(s)
}
```

	f(1:1e6)	f(1:1e8)
GNU R 3.2.0	0.675	69.046
+ BC		57.466
Renjin+JIT	0.02	1.157

renjin

Timings

```
halfsquare <- function(n) (n*n)/2

f3 <- function(x) {
  s <- 0
  for(i in x) {
    s <- s + halfsquare(i)
  }
  return(s)
}
```

	f(1:1e6)	f(1:1e8)
GNU R 3.2.0	28.284	278.757
+ BC	26.179	-
Renjin+JIT	0.02	1.069

renjin

Comparison with GNU R Bytecode Compiler

- **Compilation occurs at runtime, not AOT:**
  - **More information available**
  - **(Hopefully) can compile without making breaking assumptions**

```
f <- function(x) x * 2
g <- compiler::cmpfun(f)
`*` <- function(...) "FOO"
f(1) # "FOO"
g(1) # 2
```

renjin

Next Steps

- S4 dispatch
- sapply()
- Specializations for [] and []←
- Minimal copy optimizations

renjin