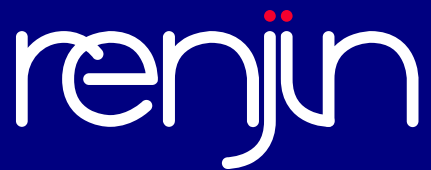


Renjin's Loop JIT  
RIOT 2016  
Stanford University

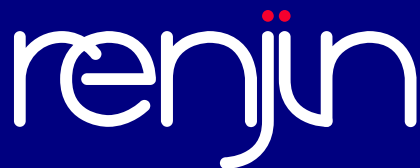


# Agenda

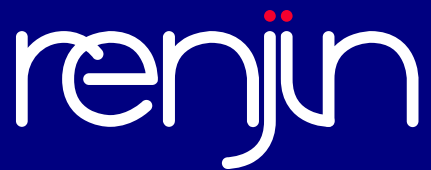
- What is Renjin?
- Quick project update
- JIT Loop Compiler

# 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)

**CWI**

Centrum Wiskunde & Informatica

**SOUND**

STATISTICAL MULTI-ORBIT UNDERSTANDING



# GNU R Compatibility

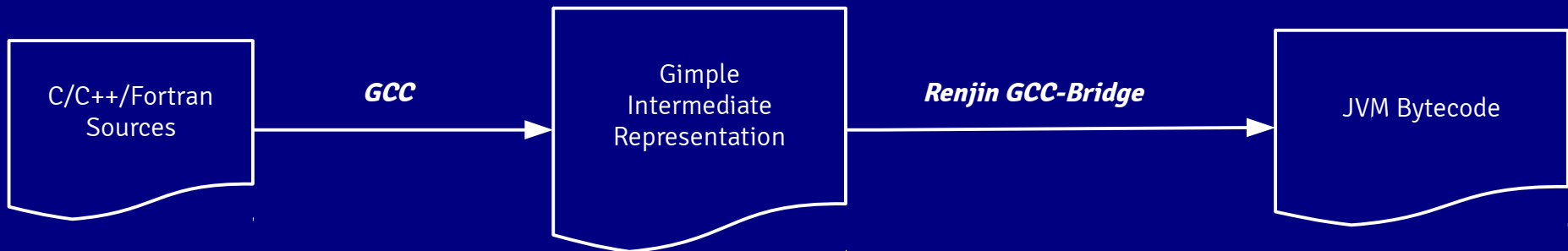
	Package Count	Cumulative Percentage
A: All tests pass	1617	17%
B: Most tests pass	1011	28%
D: At least one test pass	1660	45%
F: Blocked/No tests passing	5071	
Total	9359	

	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	

# Compatibility Challenges

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

# 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 Terminals Tabs Help
alex@alex-laptop76: ~ x alex@alex-laptop76: /tmp x alex@alex-laptop76: ~/dev/cran/i... x alex@alex-laptop76: ~/dev/R-3.2... x
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

The image shows two overlapping browser windows. The background window is a GitHub repository page for 'bedatadriven/renjin', specifically the pull request 'Fixes #90 (ClassC...)' which has been merged. The foreground window is a Stack Overflow question titled 'Renjin is Not Working'. The question text reads: 'I am attaching a screen shot of the problem that I am facing with Renjin Engine. Please see the image below. I'm not able to load the Renjin Engine'. Below the text is a screenshot of a Java IDE showing a code snippet and its output. The code defines a main method that attempts to load the Renjin engine. The output shows a 'java.lang.NoClassDefFoundError' for 'com.google.common.collect.Lists'. The Stack Overflow page also shows a 'HOT META POSTS' section with three items and a 'Jobs near you' section with two job listings: 'Core Java Developer' and 'Open challenge in Technology', both from 'Flow Traders' in Amsterdam, Netherlands. The bottom right of the Stack Overflow page shows 'Java Developer' as a tag.

Fixes #90 (ClassC...)

bedatadriven / renjin

Fixes #90 (ClassC...)

Merged akbertram merged 2 comm

Conversation 6 Commits 2

acaloiaro commented

IntegerArrayConverter no  
int[] arrays.

acaloiaro added some co

Fixes #90

Syncing with upstr

akbertram commented

@bddbot please re test

acaloiaro commented

Is there a problem with my c  
passing in my environment.

Renjin is Not Working

StackExchange 24 +79

471 4 12 help Search Q&A

Questions Jobs Tags Users Badges Ask Question

asked 6 days ago  
viewed 23 times  
active 3 days ago

HOT META POSTS

- 3 Procedure on handling formerly on-topic questions
- 4 Can duplicate question suggestions use syntax and semantics of questions?
- 26 Add placeholder to answer box?
- 16 Internal server error on "Go get your next badge" link

```
Source History  
17 /**  
18  * @param args the command line arguments  
19  */  
20 public static void main(String[] args)  
21 {  
22     // create a script engine manager:  
23     ScriptEngineManager manager = new ScriptEngineManager();  
24     // create a Renjin engine:  
25     ScriptEngine engine = manager.getEngineByName("Renjin");  
26     // check if the engine has loaded correctly:  
27     if(engine == null)  
28     {  
29         throw new RuntimeException("Renjin Script Engine not found on the classpath.");  
30     }  
31     // ... put your Java code here ...  
32 }  
33
```

Output  
Debugger Console Renjin (run)

```
run:  
Exception in thread "main" java.lang.NoClassDefFoundError: com/google/common/collect/Lists  
    at org.renjin.script.RenjinScriptEngineFactory.getNames(RenjinScriptEngineFactory.java:56)  
    at javax.script.ScriptEngineManager.getEngineByName(ScriptEngineManager.java:229)  
    at renjin.Renjin.main(Renjin.java:24)  
Caused by: java.lang.ClassNotFoundException: com.google.common.collect.Lists  
    at java.net.URLClassLoader.findClass(URLClassLoader.java:381)  
    at java.lang.ClassLoader.loadClass(ClassLoader.java:424)  
    at sun.misc.Launcher$AppClassLoader.loadClass(Launcher.java:381)  
    at java.lang.ClassLoader.loadClass(ClassLoader.java:357)  
    ... 3 more  
Java Result: 1  
BUILD SUCCESSFUL (total time: 3 seconds)
```

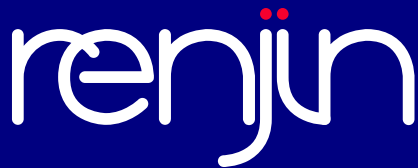
The debug output is also attached in the picture. Can you please help me debug this?

Jobs near you

Core Java Developer  
Flow Traders Amsterdam, Netherlands  
java hibernate

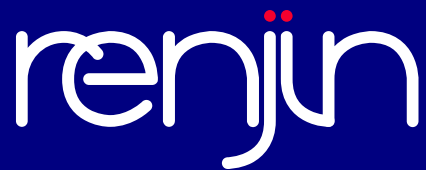
Open challenge in Technology  
Flow Traders Amsterdam, Netherlands  
java c++

Java Developer

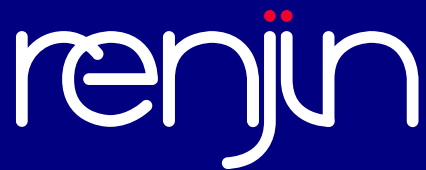


Take a moment to download Renjin...

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



# Introducing Renjin's JIT Compiler



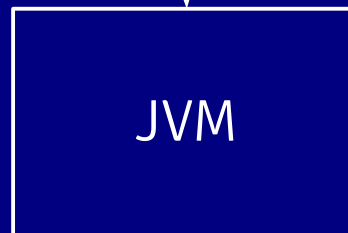
# Execution Modes



```
a <- runif(3500000)*1000  
(phi^a - (-phi)^(-a))/sqrt(5)
```

```
.Call()
```

```
for(i in 1:1e8)
```



*Parallelization, Loop-Fusion*

# Compare:

Vector  
Operations

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

~ 10 R expressions  
evaluated

Loops

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

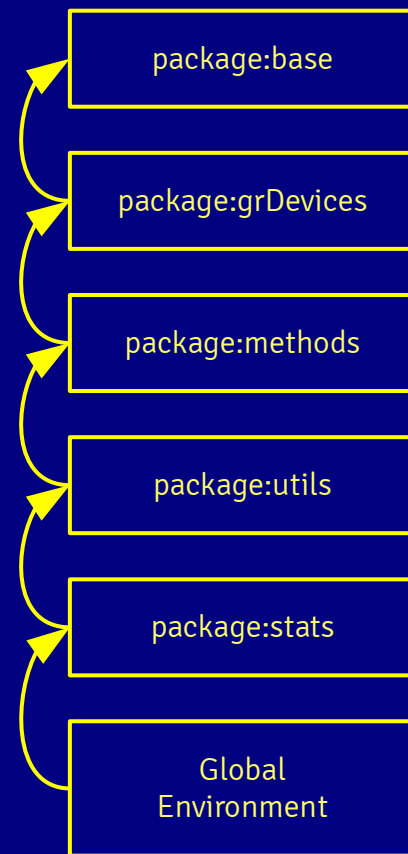
~ 300m R expressions  
evaluated

**38 x slower**

# Function Loop

**Function Lookup**

```
s <- 0  
for (i in 1:1e3) {  
  s <- s + sqrt(i)  
}  
print(s)
```



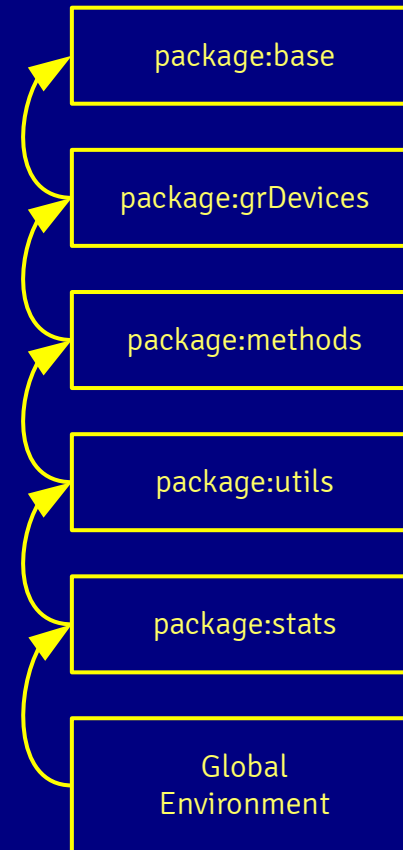
`+` = `.Primitive("+")`  
`sqrt` = `.Prim("sqrt")`



# 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**

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

# Function Calls

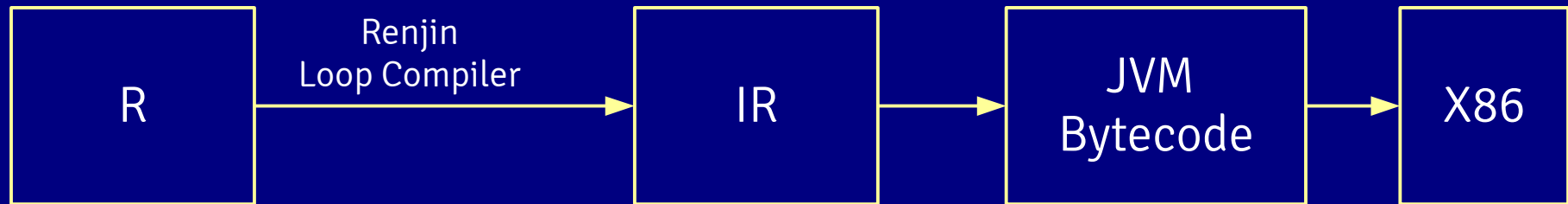
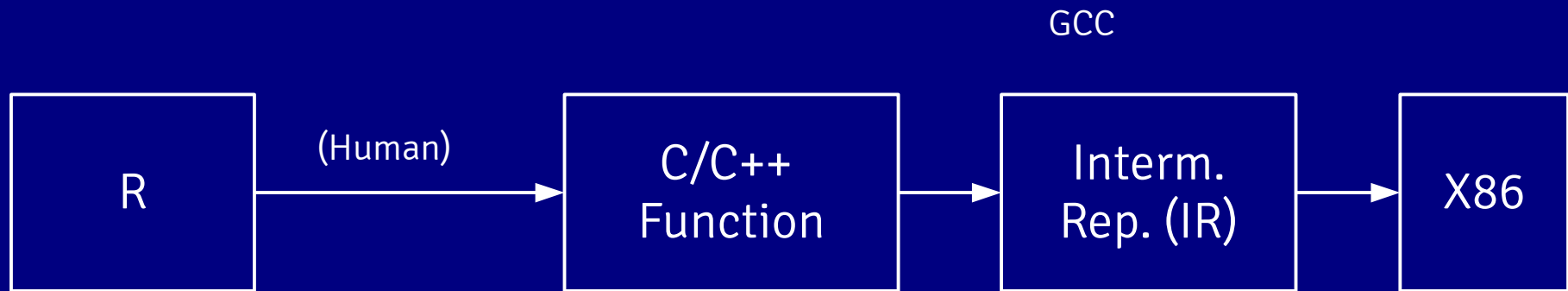
Function Calls are Expensive

```
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)
```

# Handling Promises

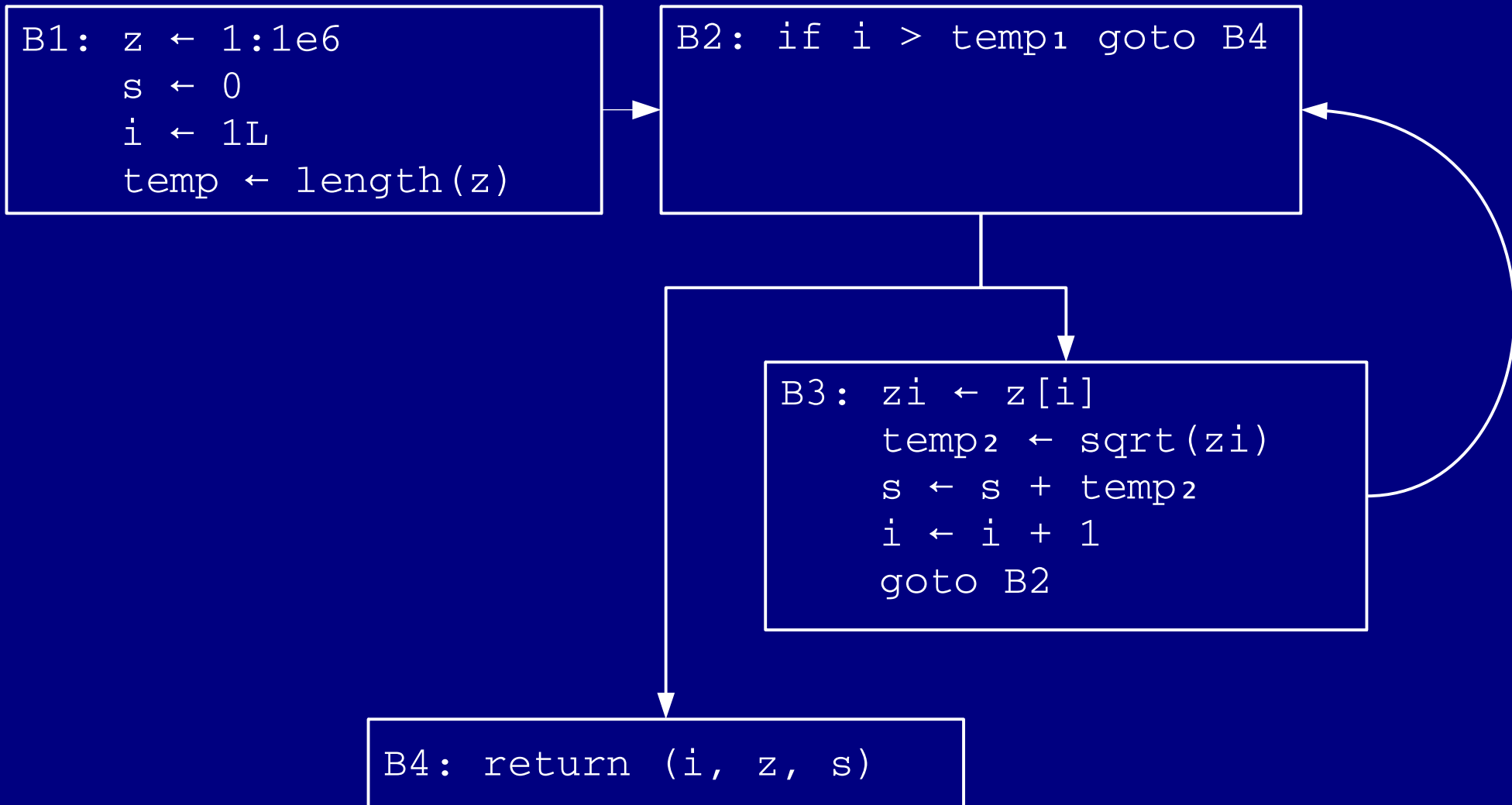
```
f <- function(x, y) {  
  s <- 0  
  for(i in x) {  
    s <- s +  
    if(i < 0)  
      y  
    else  
      log(i)  
  }  
  s  
}
```

```
f(1:10000)
```

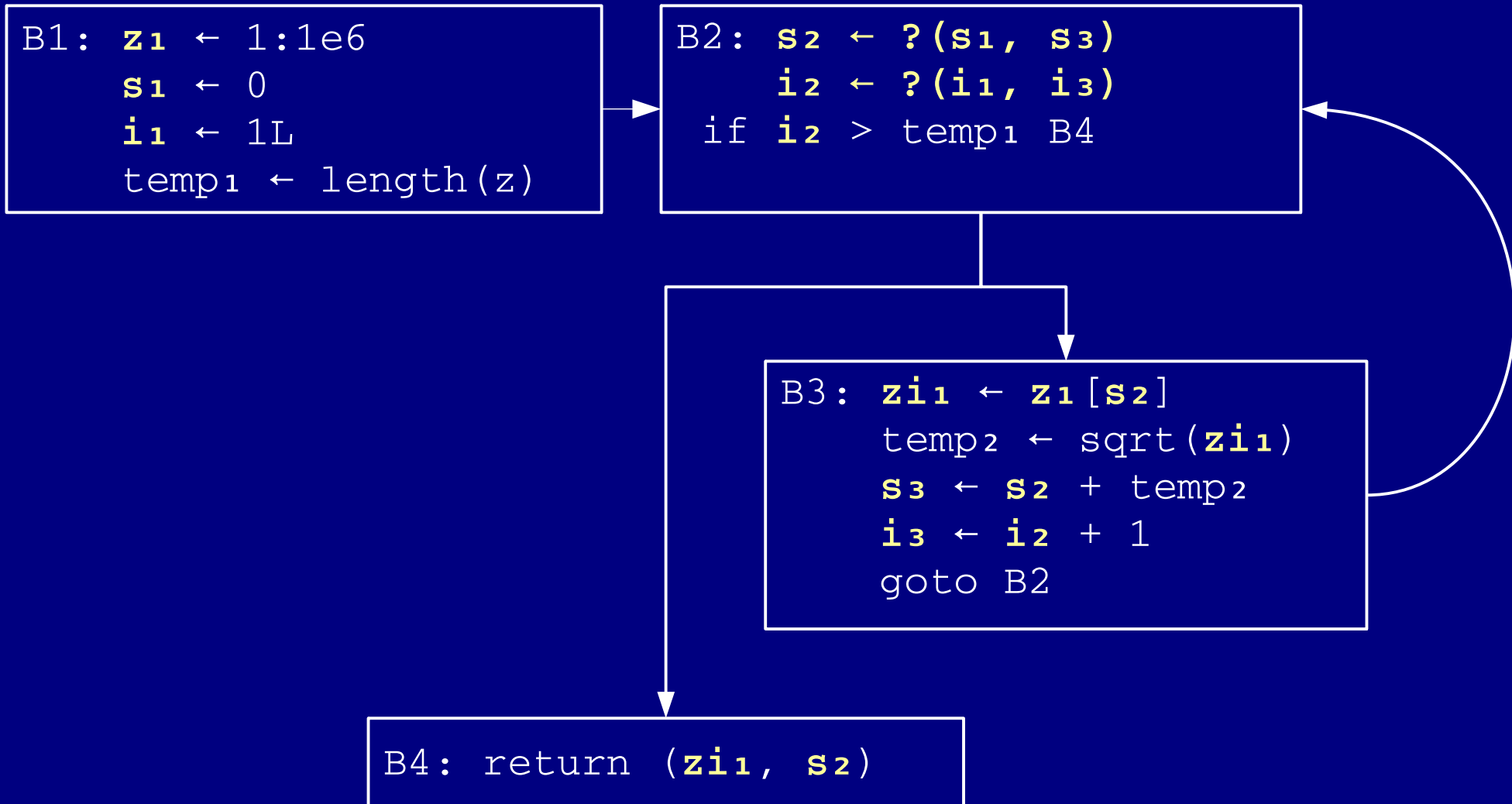


- Run loop 5 times in interpreter to force as many promises as possible
- Currently bails if unevaluated promise is encountered.

# Step 2: Build CFG



# Step 3: Transform to SSA





# 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: <b>z1</b> ← 1:1e6	z1	=	1:1e6
<b>s1</b> ← 0	S1	=	0.0
<b>i1</b> ← 1L	I1	=	1L
<b>temp1</b> ← length( <b>z1</b> )	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)
```

# 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)
```

# 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)
```

# Statically Computing Bounds

B1: <b>z1</b> ← 1:1e6	z1	=	1:1e6
<b>s1</b> ← 0	s1	=	0.0
<b>i1</b> ← 1L	I1	=	1L
<b>temp1</b> ← length( <b>z1</b> )	temp1	=	1e6L
B2: <b>s2</b> ← ?( <b>s1</b> , <b>s3</b> )	S2	=	0.0 ? 1.0 = num[1]
<b>i2</b> ← ?( <b>i1</b> , <b>i3</b> )	i2	=	1L ? 2L = int[1]
if <b>i2</b> > <b>temp1</b> B4		=	int[1] > 1e6 = T F

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

B4: return (**zi1**, **s2**)

# Statically Computing Bounds

B1: <b>z1</b> ← 1:1e6	z1	=	1:1e6
<b>s1</b> ← 0	s1	=	0.0
<b>i1</b> ← 1L	I1	=	1L
<b>temp1</b> ← length( <b>z1</b> )	temp1	=	1e6L

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

B3: <b>zi1</b> ← <b>z1</b> [ <b>i2</b> ]	Zi1	=	1e6[ int[1] ] = num[1]
<b>temp2</b> ← sqrt( <b>zi1</b> )	temp2	=	sqrt(num[1]) = num[1]
<b>s3</b> ← <b>s2</b> + <b>temp2</b>	s3	=	num[1] + num[1] = num[1]
<b>i3</b> ← <b>i2</b> + 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”



# 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
```

# 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

# 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

# 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

# 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
```



# Next Steps

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