renjin

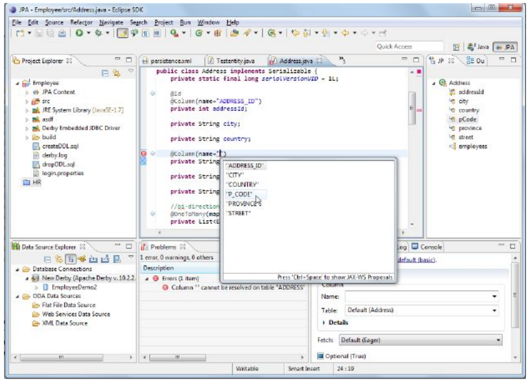
Parham Solaimani, Ph.D.
BeDataDriven BV
The Hague, The Netherlands

What is Renjin

- R interpreter in Java running in JVM

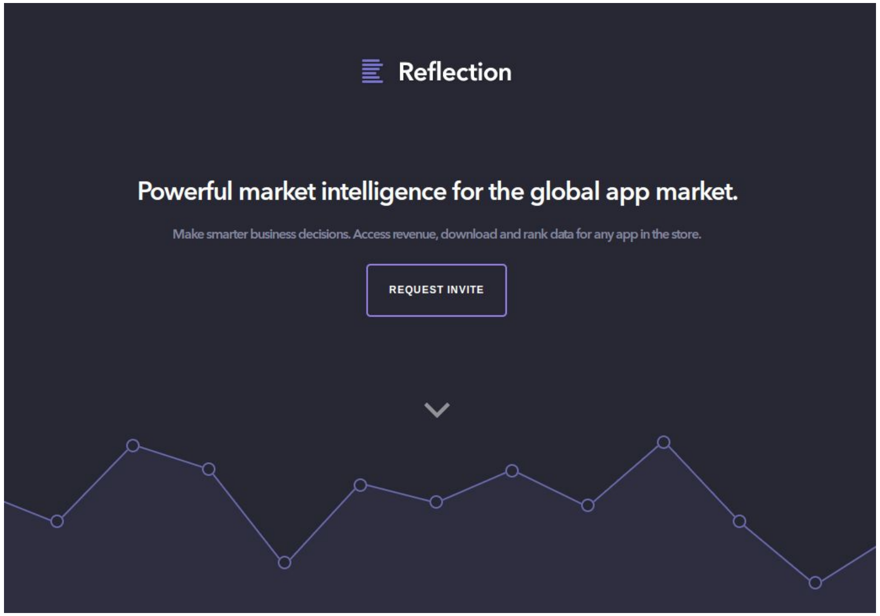
Run and scale with
could Platform-as-a-Service

Integrate into existing
Java applications



Use Enterprise
Development Environment

R on cloud Platform-as-a-Service



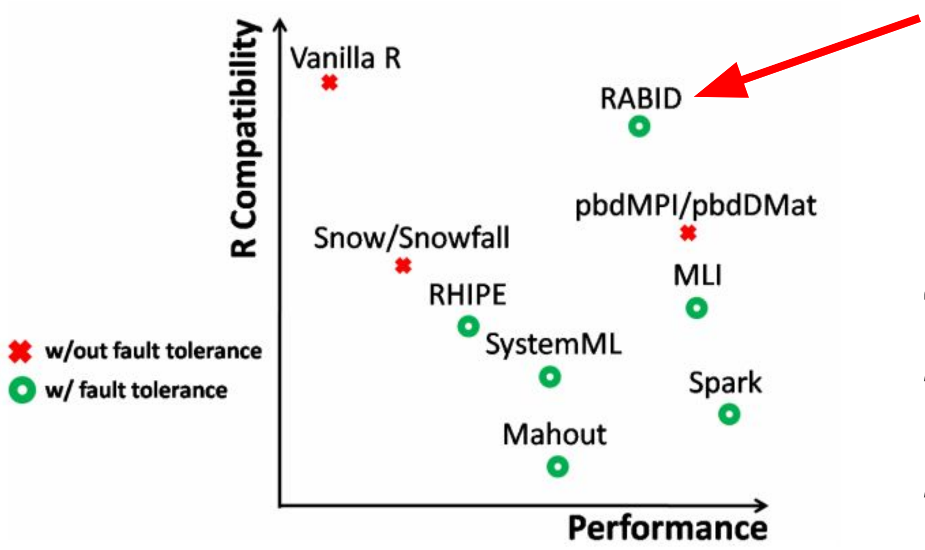
reflection.io

- R model predicting app revenue (statistician)
- Java-based platform on Google AppEngine (developers)

Other examples

- **Yodle**: Deploy R based statistical models directly into production without having to rewrite into Java
- **Renjin AppEngine Demo**: renjindemo.appspot.com

Renjin on Spark cluster



RABID: Spark + Renjin / GNU R

- Fault tolerance, efficiency, low overhead and minimized network transfers

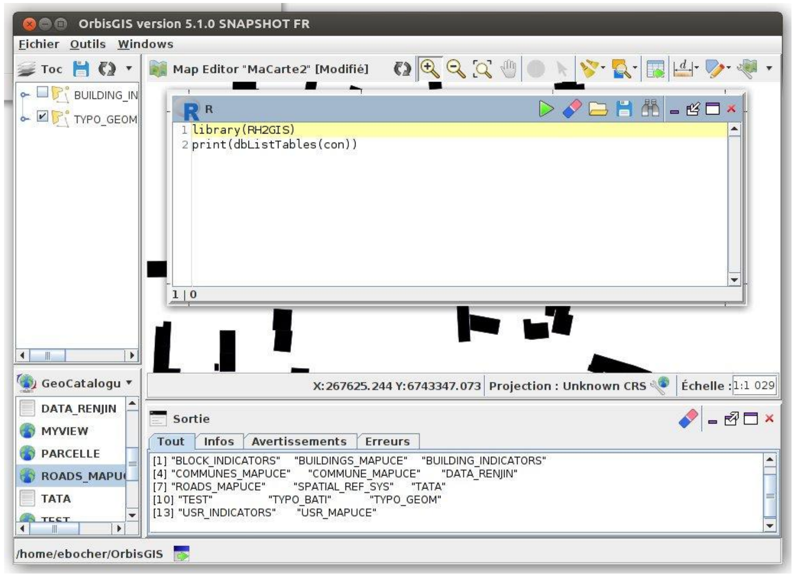
"*it [Renjin], like Spark, is implemented in Java, and consequently can be better integrated with Spark*"

Lin H., *et al*. 2014, *IEEE Int. Congress on Big Data*.

Others

- Spark+Renjin used by Apple in production cluster (of 1000 nodes)
- REX: Apache Spark Renjin Executer (on github)

R in existing Java applications

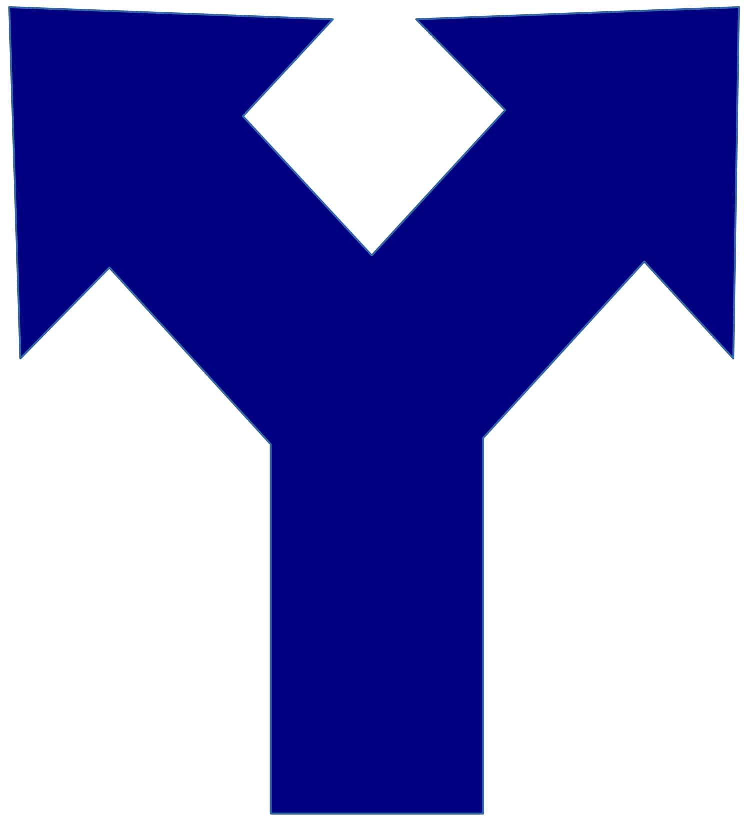


OrbisGIS
An Open Source Geographic Information System
Lab-STICC – CNRS

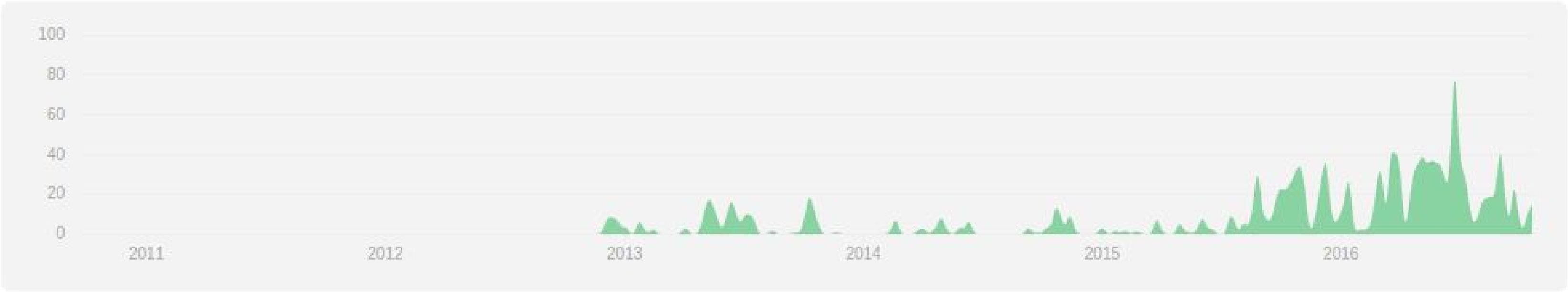
Renjin as R console to allow statistical analysis of GIS information

SciJava Renjin module: Provides a scripting plugin for Renjin interpreter to tools such as ImageJ, KNIME, CellProfiler, OMERO and others.
SciCom: SciCom is a JRuby gem that allows very tight integration between Ruby and R languages.
icCube: Business Intelligence tool with R integration provided by Renjin

Compatibility Performance



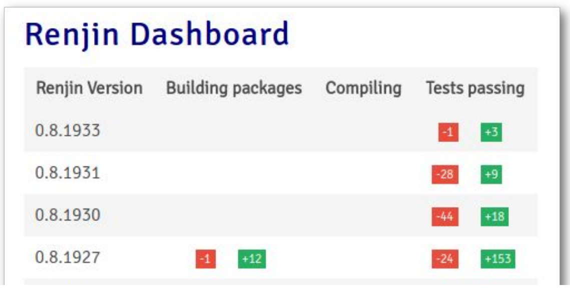
Approach to Compatibility

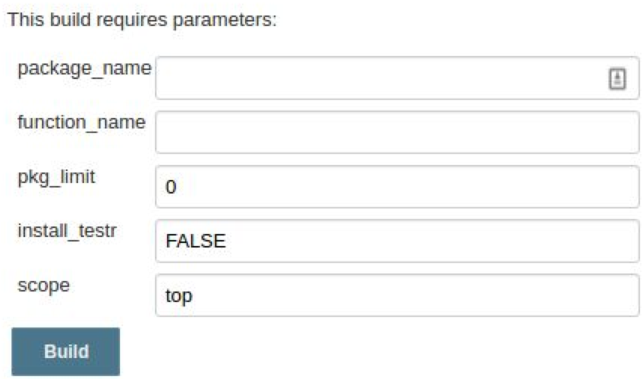


- Support major dependencies
  - S4 object system, Rcpp, MASS, etc.
- Improvement of Renjin development and testing environment
- Measurement and tracking of compatibility over time

Development environment

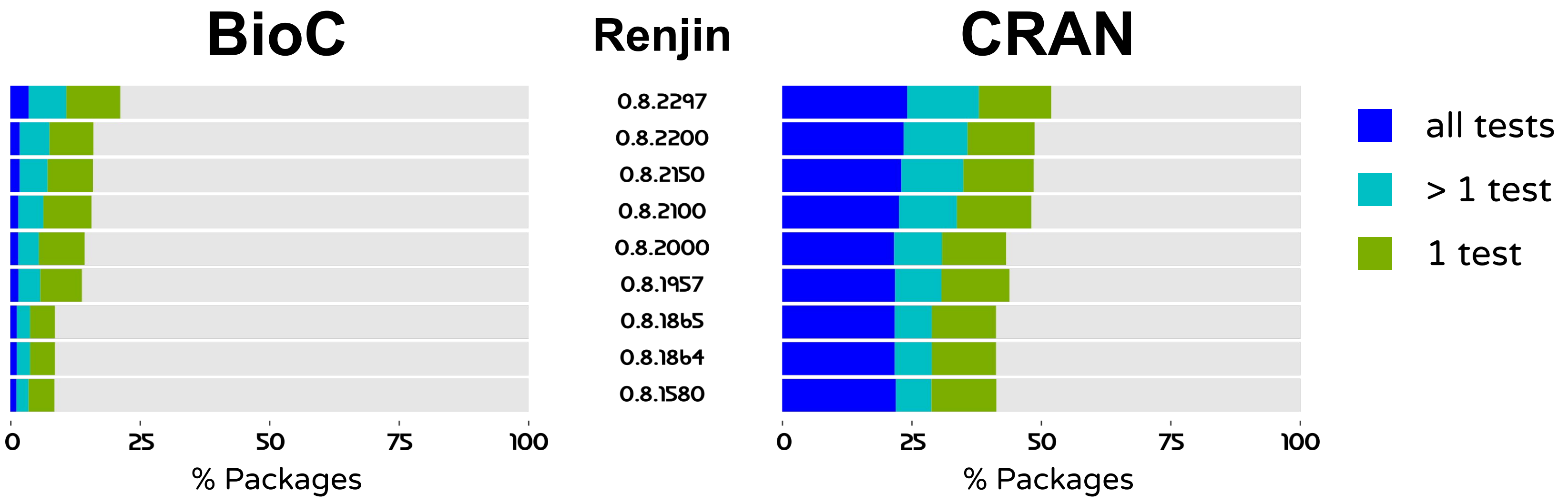
- Real-world with real data bioInformatics workflow (renjin-benchmarks)
- Automated test-case generation (based on testr)
- Renjin dashboard





- Goals:
  - Reduce time-to-answer for workflows
  - Reduce developer time required for performant solutions.

GNU R Compatibility



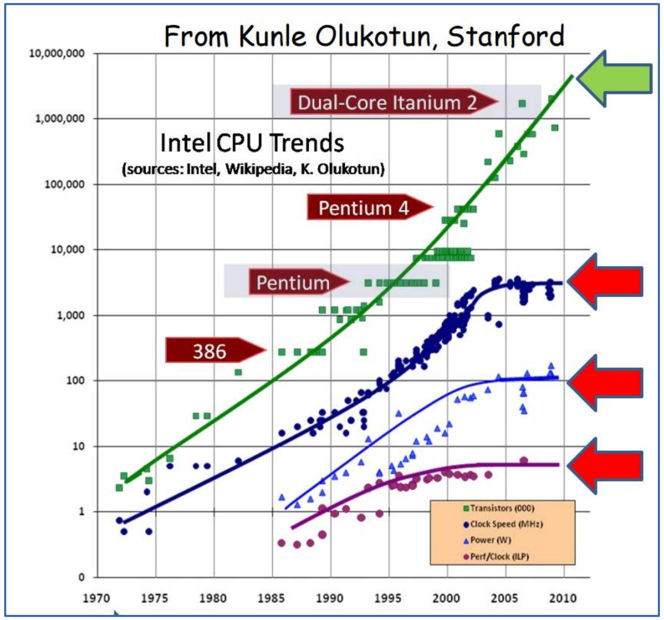
Sinds 1st January 2016	
Builds	**~ 250**
Compiles	**~ 800**
Passing tests	**> 9000**

Performance.

Trends

Package Sources
Overall Statistics

	R	C	C++	Fortran
CRAN	17.16	8.84	5.24	1.84
BioConductor	2.50	1.86	1.71	0.02



Compare:

Vector Operations	Loops

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

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

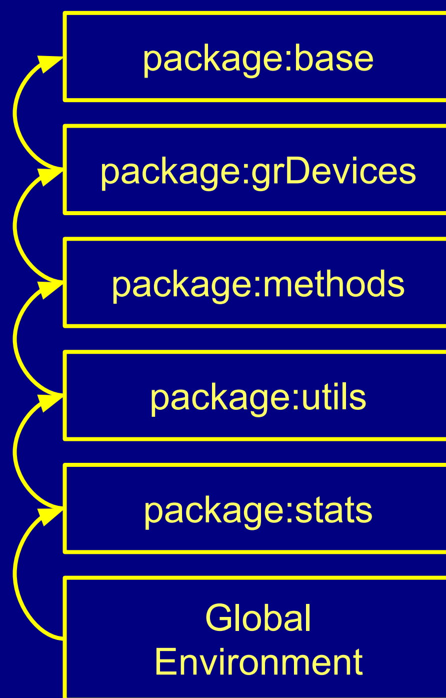
~ 10 R expressions evaluated

~ 300m R expressions evaluated

Function Lookup → Function selection → Boxing → Function Call

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

Function Lookup

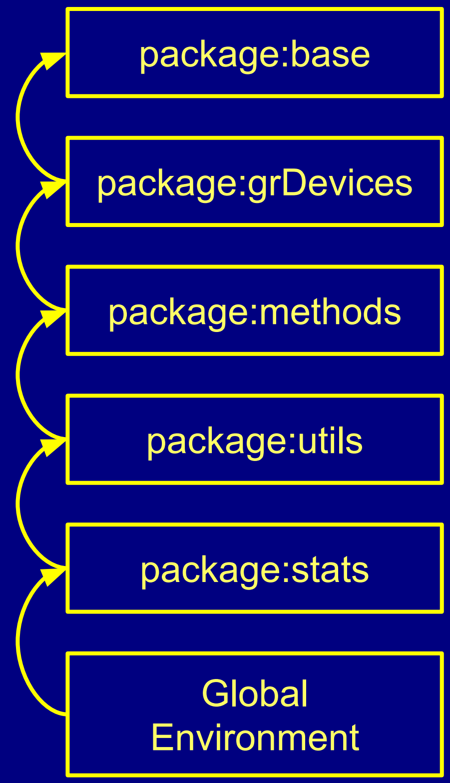


+ = .Primitive(“+”)
sqrt = .Prim(“sqrt”)

Function Lookup → **Function selection** → Boxing → Function Call

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

Function Lookup



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

Function Lookup → Function selection → **Boxing** → Function Call

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.

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

Function Calls are Expensive

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.

Transform to SSA

```
s <- 0
z <- 1:1e6
for(zi in z) {
        s <- s + sqrt(zi)
}
```

B1: $z_1 \leftarrow$ 1:1e6
$s_1 \leftarrow 0$
$i_1 \leftarrow$ 1L
$temp_1 \leftarrow$ length(z)

B2: $s_2 \leftarrow \Phi(s_1, s_3)$
$i_2 \leftarrow \Phi(i_1, i_3)$
if $i_2 > temp_1$ B4

B3: $zi_1 \leftarrow z_1[s_2]$
$temp_2 \leftarrow$ sqrt(zi_1)
$s_3 \leftarrow s_2 + temp_2$
$i_3 \leftarrow i_2 + 1$
goto B2

B4: return (zi_1, s_2)

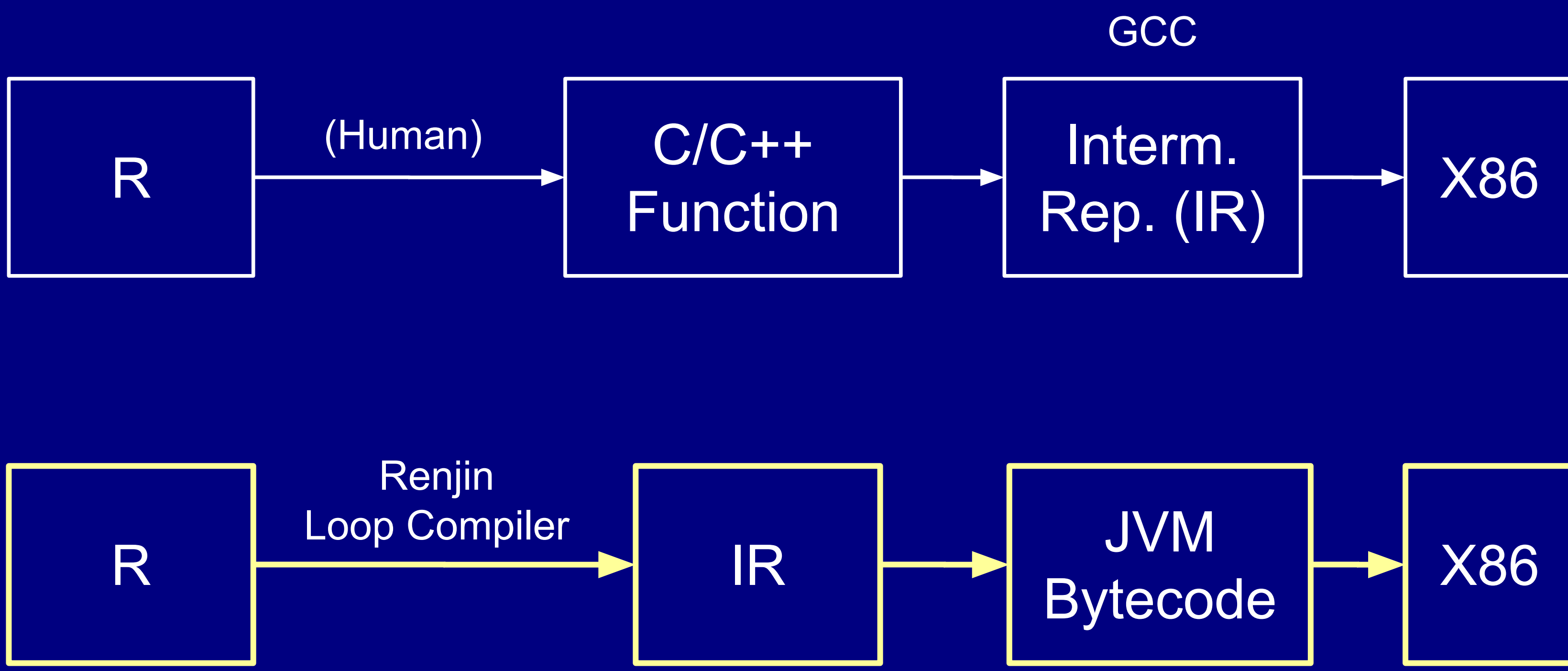
Assumptions recorded:

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

renjin

renjin.org

Comparing Workarounds



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”

Timings

```
f <- 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	0.355

renjin

Timings

```
f <- 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	0.107	0.367

Timings

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

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

	`f(1:1e6)`	`f(1:1e8)`
GNU R 3.2.0	28.284	278.757
+ BC	26.179	-
Renjin+JIT	0.117	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

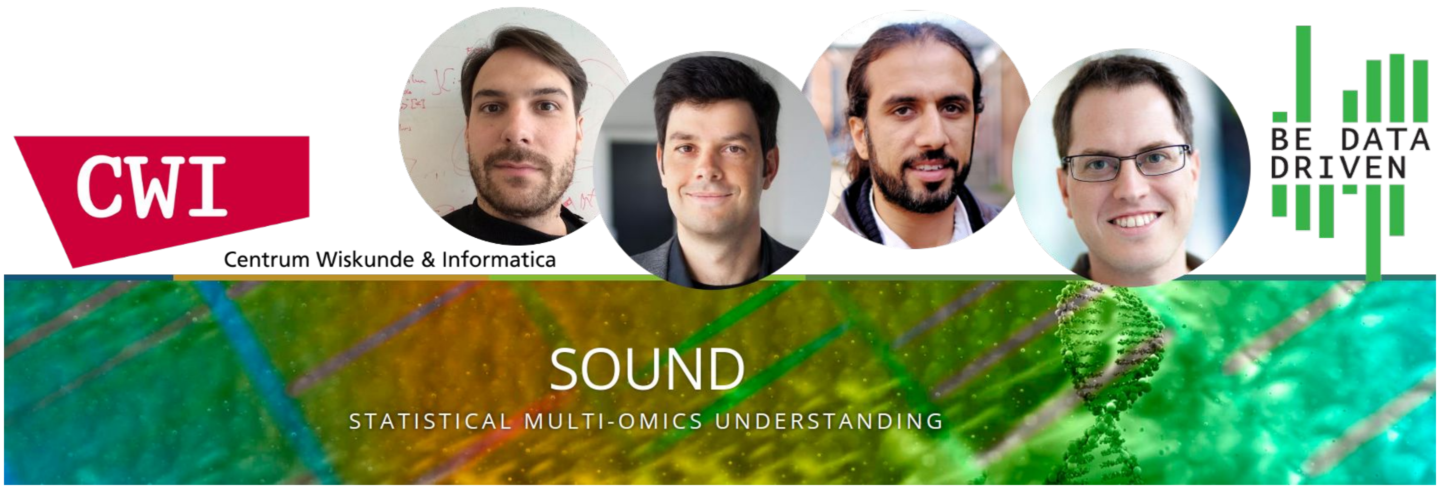
```
f <- function(x) x * 2
g <- compiler::cmpfun(f)
`*` <- function(...) "FOO"
f(1) # "FOO"
g(1) # 2
```

renjin

Next Steps

- Continue work on compatibility with GNU R / BioConductor
- Expand and continue profiling benchmark library
- More in depth analysis of CPU, (cache) memory, disk usage by benchmarks
- Extend impliciet optimizations

Questions?


CWI
Centrum Wiskunde & Informatica
BE DATA DRIVEN
SOUND
STATISTICAL MULTI-OMICS UNDERSTANDING