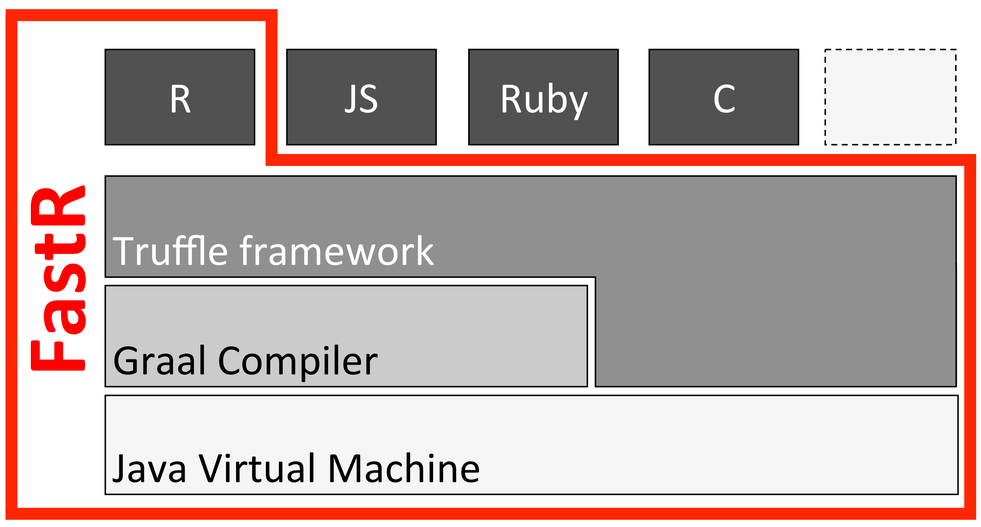
High-performance R with FastR

Adam Welc, *Oracle Labs*
adam.welc@oracle.com

ORACLE®

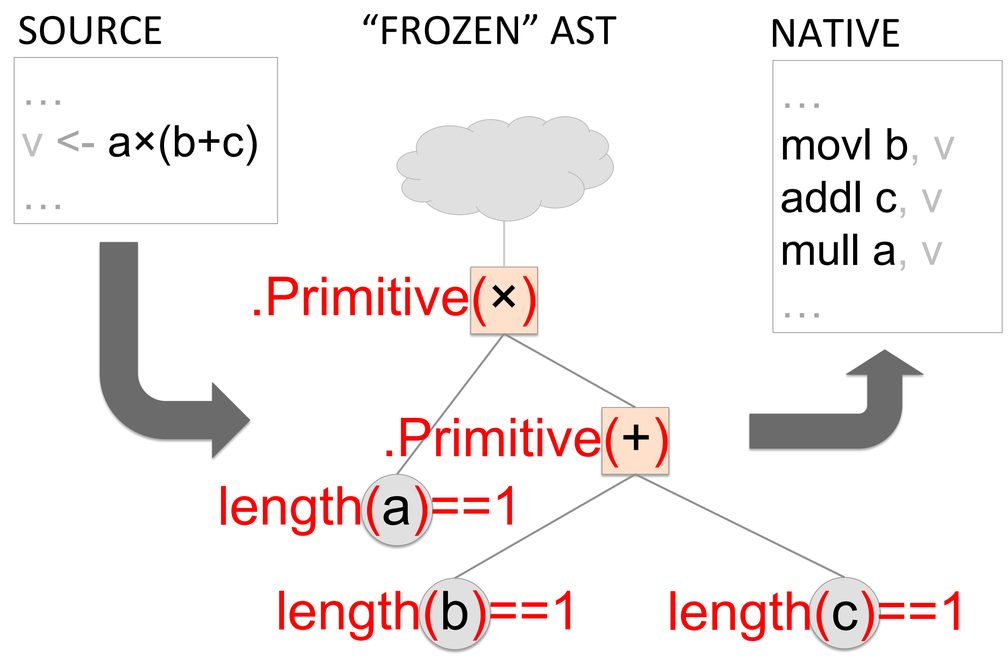
FastR project

- FastR is an alternative R execution engine, developed under GPL v2
- Drop-in, fully compatible replacement for R's reference implementation GNU R
- Focused on improving performance of long-running R code
- Open-source: https://github.com/graalvm/fastr
- Implemented on top of Truffle framework, utilizing Graal native compiler
- FastR team at Oracle Labs:
  - Mick Jordan, Štěpán Šindelář, Zbyňek Šlajchtr, Lukas Stadler, Adam Welc
- Status
  - Implemented all important language features, including lazy evaluation, calls to C/Fortran, as well as S3 and S4 object models
  - FastR can load over 2000 unmodified CRAN packages and run selected production applications in parallel
  - Missing features include portions of the native interface and selected builtins



Truffle and Graal

- Truffle is an open-source framework for implementing programming language runtimes
  - Based on Java Virtual Machine (JVM) technology
  - Reuses highly optimized JVM services (e.g. memory management)
  - Graal compiler eventually compiles "hot" paths (e.g. loops) to machine code
- A program in a Truffle-based language is represented as Abstract Syntax Tree (AST)
  - AST encodes both program behavior (e.g. two numbers are being added), and language semantics (e.g. how the addition operator works)
  - Execution of a program in Truffle-based language == AST traversal
- Truffle transforms an R program:
  - Analyzes running program to gather additional information (e.g. argument vectors have certain length, or + and × operators are primitive functions)
  - Speculates that observed conditions will hold in the future
  - "Freezes" the AST and with the help of Graal compiler generates (guarded) native code



High performance through speculation

- FastR optimizes R language execution via three different facets of speculation: assumptions, caching, and specialization
- These three techniques permeate the entire FastR implementation, with the following being selected examples of FastR optimizations

ASSUMPTIONS

- Assumptions are natively supported by Truffle
  - Cheap to check if condition holds
  - Expensive to handle invalid assumptions
- Example use – eager evaluation of promises

```
var <- 42
fun(var)
```

associate Truffle assumption with var (no change to var until needed in fun)

evaluate var eagerly on call to fun

inside fun, truffle can speculate that assumption holds and have the compiler eliminate both promise and related code

CACHING

- Implementation of caches simplified via Truffle DSL
- Example use – caching of argument signatures

```
f <- function(a, b, carg, …, d=6)
    list(a, b, carg, ..1, ..2, d)
```

first invocation: run argument matching algorithm and cache argument signagure <'b', NULL, 'c', NULL, NULL>

```
f(b=2, 1, c=3, 4, 5)
```

on second invocation Truffle can speculate that the new argument signature matches the cache and use the same argument permutation for the function call

SPECIALIZATION

- Truffle DSL helps building specialized nodes
- Example use – vector length

```
vec <- c(1, 2, 3, 4)
for (i in 2:length(vec)
    vec[i-1] <- vec[i] + vec[i-1]
```

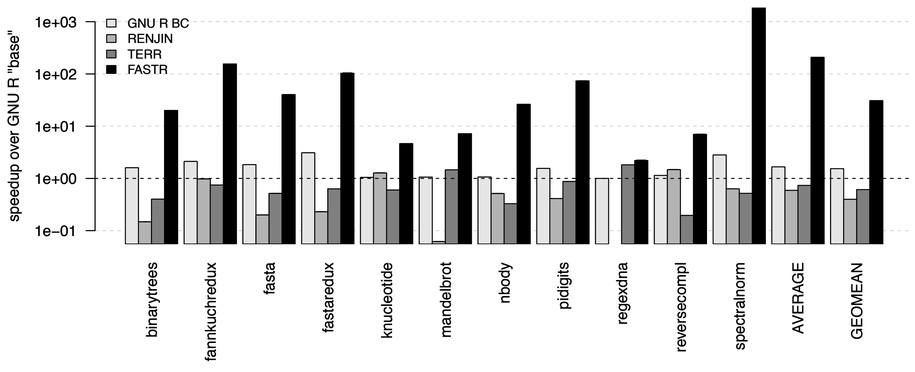
Truffle can speculate on vector length being constant and create specialized inlined code

```
vec[1] <- vec[2] + vec[1]
vec[2] <- vec[3] + vec[2]
vec[3] <- vec[4] + vec[3]
```

Results

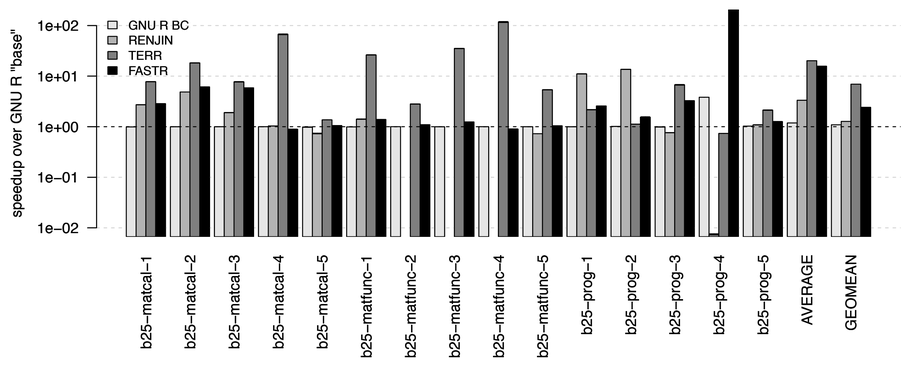
- Two benchmark suites
  - Shootout
  - B25
- Five runtime configurations
  - GNU R "base"
  - GNU R "BC" (b-code "compiler")
  - Renjin and TERR (alternate R runtimes)
  - FastR
- Plotted peak performance on a logarithmic scale

SHOOTOUT (http://benchmarksgame.alioth.debian.org)



- Small applications consisting mostly of R code
- FastR's avg. speedup over GNU R "base": ~208.7 (geomean: ~30.8)

B25 (http://r.research.att.com/benchmarks)



- Matrix calculations + simple R computation tasks
- FastR's avg. speedup over GNU R "base": ~15.7 (geomean: ~ 2.4)