# Who You Gonna Call: Analyzing the Run-time Call-Site Behavior of Ruby Applications

Sophie Kaleba
S.Kaleba@kent.ac.uk
University of Kent
United Kingdom

Octave Larose
O.Larose@kent.ac.uk
University of Kent
United Kingdom

Richard Jones
R.E.Jones@kent.ac.uk
University of Kent
United Kingdom

Stefan Marr
s.marr@kent.ac.uk
University of Kent
United Kingdom

## Abstract

Applications written in dynamic languages are becoming larger and larger and companies increasingly use multi-million line codebases in production. At the same time, dynamic languages rely heavily on dynamic optimizations, particularly those that reduce the overhead of method calls.

In this work, we study the call-site behavior of Ruby benchmarks that are being used to guide the development of upcoming Ruby implementations such as TruffleRuby and YJIT. We study the interaction of call-site lookup caches, method splitting, and elimination of duplicate call-targets.

We find that these optimizations are indeed highly effective on both smaller and large benchmarks, methods and closures alike, and help to open up opportunities for further optimizations such as inlining. However, we show that

**ACM Reference Format:**
Sophie Kaleba, Octave Larose, Richard Jones, and Stefan Marr. 2022. Who You Gonna Call: Analyzing the Run-time Call-Site Behavior of Ruby Applications. In *Proceedings of the 18th ACM SIGPLAN International Symposium on Dynamic Languages (DLS '22), December 07, 2022, Auckland, New Zealand*. ACM, New York, NY, USA, 16 pages.
https://doi.org/10.1145/3563834.3567538

## 1 Introduction

Dynamic languages such as JavaScript, PHP, Python, and Ruby are used in industry to build a wide range of systems including application backends. Their dynamic language features support rapid application development, but require run-time compilation and optimization to achieve good per-

This is research by

**Sophie Kaleba**

Doctoral Student
University of Kent, UK

I’m just advertising it with her permission

Call-sites are everywhere in Ruby:

```
user.send_welcome_email

point.x

left + right

[a, b, c] + not_an_array
```

Most Ruby code is a call-site!

Call-sites can be:

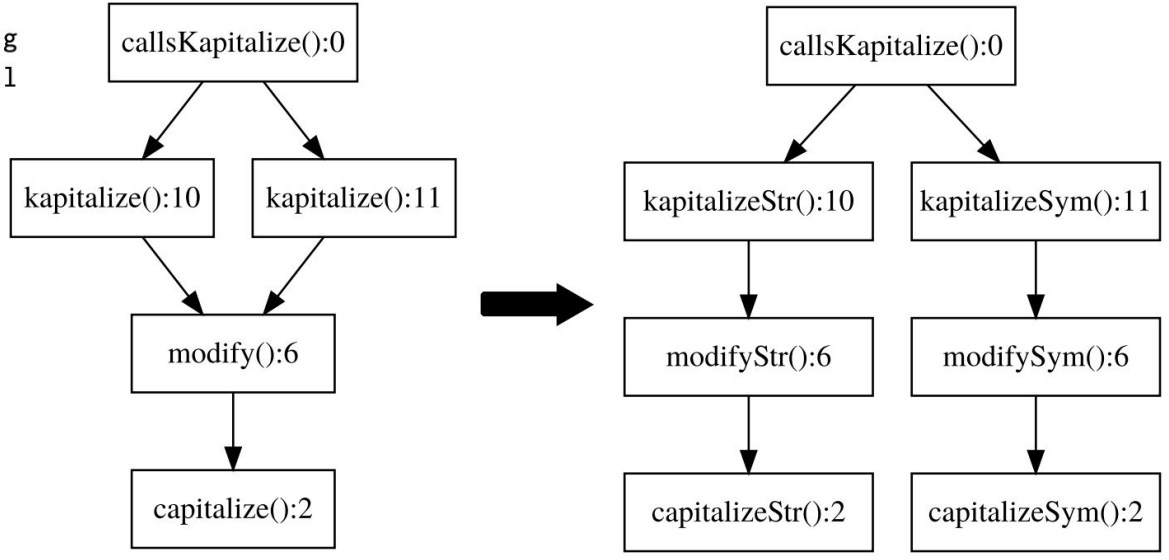
- Monomorphic - only ever calls one method
- Polymorphic - calls one of a small number of methods
- Megamorphic - calls one of a large number of methods, or literally any method

Monomorphic is the best because monomorphic calls are like simple C calls - a simple machine call instruction.

```ruby
def modify(arg1)
    arg1.capitalize()
end

def kapitalize(arg1)
    modify(arg1)
end

def callsKapitalize()
    kapitalize("foo") # a String
    kapitalize(:bar)  # a Symbol
end
```




**(b)** Impact of splitting on the application's structure

**Table 3.** The polymorphic and megamorphic calls remaining after having eliminated target duplicates are almost completely monomorphized by splitting.

| | Number of calls | | After splitting | | Number |
|---|---|---|---|---|---|
| Benchmark | Poly. | Mega. | Poly. | Mega. | of splits |
| BlogRails | 490,072 | 557 | -100% | -100% | 2163 |
| ChunkyCanvas* | 66 | 0 | -100% | 0% | 43 |
| ChunkyColor* | 66 | 0 | -100% | 0% | 42 |
| ChunkyDec | 66 | 0 | -100% | 0% | 42 |
| ERubiRails | 391,997 | 553 | -100% | -100% | 1851 |
| HexaPdfSmall | 1,443,211 | 2,066 | -100% | -100% | 498 |
| LiquidCartParse | 219 | 0 | -100% | 0% | 107 |
| LiquidCartRender | 2,000 | 0 | -100% | 0% | 207 |
| LiquidMiddleware | 233 | 0 | -100% | 0% | 114 |
| LiquidParseAll | 679 | 0 | -100% | 0% | 136 |
| LiquidRenderBibs | 23,633 | 0 | -100% | 0% | 191 |
| MailBench | 18,322 | 0 | -100% | 0% | 343 |
| PsdColor | 6,586 | 0 | -100% | 0% | 300 |
| PsdCompose* | 6,586 | 0 | -100% | 0% | 300 |
| PsdImage* | 6,588 | 0 | -100% | 0% | 300 |
| PsdUtil* | 6,584 | 0 | -100% | 0% | 300 |
| Sinatra | 1,362 | 220 | -100% | -100% | 297 |
| ADConvert | 12,226 | 0 | -100% | 0% | 236 |
| ADLoadFile | 10,525 | 0 | -100% | 0% | 175 |
| DeltaBlue | 561 | 0 | -100% | 0% | 78 |
| PsychLoad | 103,506 | 0 | -100% | 0% | 78 |
| RedBlack | 8,043,472 | 0 | -100% | 0% | 50 |

“Ruby is slow because any call-site could call any method!”

No - we have the technology to fully monomorphise a Ruby application! *

* may come with a cost to memory and start-up and warm-up time!

Made possible by:

TruffleRuby - super-powerful Ruby interpreter, developed by Oracle and Shopify, by me and friends like Maple Ong and Kevin Menard here today at RubyConf Mini

Dispatch chains - ‘multi-dimensional’ inline caches, new research idea invented by me and Stefan Marr in order to optimise Ruby’s trickiest call-sites, being explored further by Matthew Alp at Shopify

Splitting - old idea, turned up to 11 in TruffleRuby *

* Sophie has found it’s possibly turned up too far

Could this idea go into MRI and work there?

Maybe! We should get someone to try that!

# The Ruby Bibliography

Academic writing on the Ruby programming language

The Ruby programming language hasn't historically been the subject of much research, either in industry or academia. A lot of recent systems research has used languages like C, C++ and Java. Contemporary programming language research often uses languages like Java, Scala, Racket and Haskell. Modern research into VMs, compilers and garbage collectors is often based on Java or recently Python.

However there are now a growing number of research projects using Ruby. On this page we list theses and peer-reviewed papers and articles that cover Ruby implementation or use Ruby, including alternative implementations such as JRuby.

Also see the Ruby Compiler Survey.

## Virtual Machines and Compilers

# S. Kaleba, O. Larose, R. Jones, S. Marr. Who You Gonna Call: Analyzing the Run-time Call-Site Behavior of Ruby Applications. In Proceedings of the 18th Symposium on Dynamic Languages (DLS), 2022. TruffleRuby

# M. Chevalier-Boisvert, N. Gibbs, J. Boussier, S. Wu, A. Patterson, K. Newton, J. Hawthorn. YJIT: a basic block versioning JIT compiler for CRuby. In Proceedings of the 13th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages (VMIL), 2021. MRI

# B. Daloze. Thread-Safe and Efficient Data Representations in Dynamically-Typed Languages. PhD thesis, Johannes Kepler University Linz, 2019. TruffleRuby

# R. Mosaner, D. Leopoldseder, M. Rigger, R. Schatz, H. Mössenböck. Supporting On-Stack Replacement in Unstructured Languages by Loop Reconstruction and Extraction. In Proceedings of the 16th International Conference on Managed Programming Languages and Runtimes (MPLR), 2019. TruffleRuby

# K. Sasada. Gradual Write-Barrier Insertion into a Ruby interpreter. In Proceedings of the International Symposium on Memory Management (ISMM), 2019. MRI

# K. Sugiyama, K. Sasada, M. J. Dürst. Dynamic Extension of the Ruby Virtual Machine Stack. In the IPSJ Journal of Programming (PRO), 2018. *In Japanese.* MRI