Call-target-specific Method Arguments

ICOOOLPS 2015 (Short Paper)

Fabio Niephaus, Matthias Springer, Tim Felgentreff, Tobias Pape, Robert Hirschfeld

Hasso Plattner Institute, Software Architecture Group

July 6, 2015

Introduction

- **Goal:** Make argument handling faster $\rightarrow$ make method calls faster
- **How to:** Prepare arguments at call site.
- **Running example:** Keyword arguments in JRuby $\rightarrow$ twice as fast

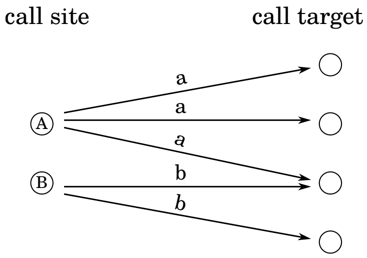
Argument Mismatch

Method Signature Parameters $\neq$ Call Arguments

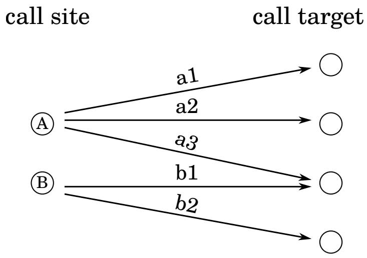
```
def method(a: 0, b: 0, c: 0)
  ...
end

method(a: 1, b: 2, c: 3)

method(b: 1, a: 2)
method(c: 4)
method()
```

When to Convert Arguments?

Convert after invoke:

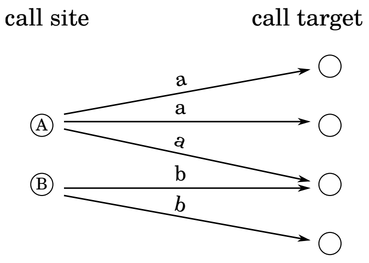


Convert before invoke:



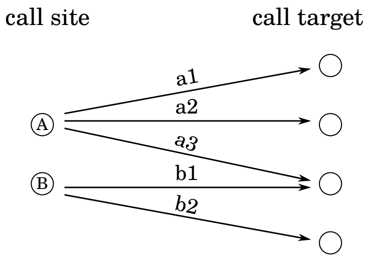
When to Convert Arguments?

Convert after invoke:



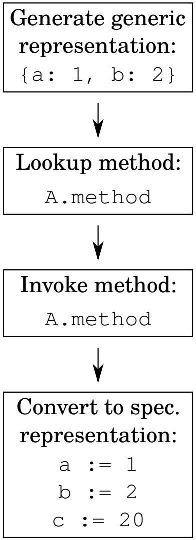
1. Convert args to generic repres.
2. Lookup receiver
3. Invoke target method
4. Convert args to specific repres.

Convert before invoke:



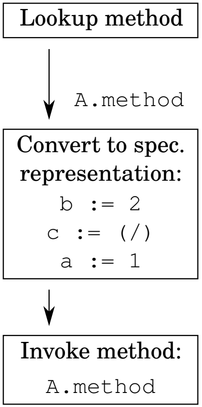
1. Lookup receiver
2. Convert args to specific repres.
3. Invoke target method

Convert After Invocation: Call-site-specific Arguments



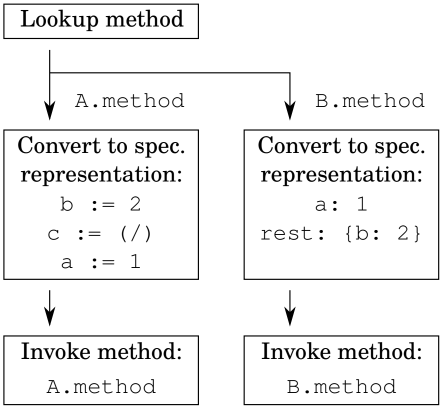
```
def A.method(b: 10, c: 20, a: 30)
  ...
end

obj.method(a: 1, b: 2)
```

Convert Before Invocation: Call-target-specific Arguments



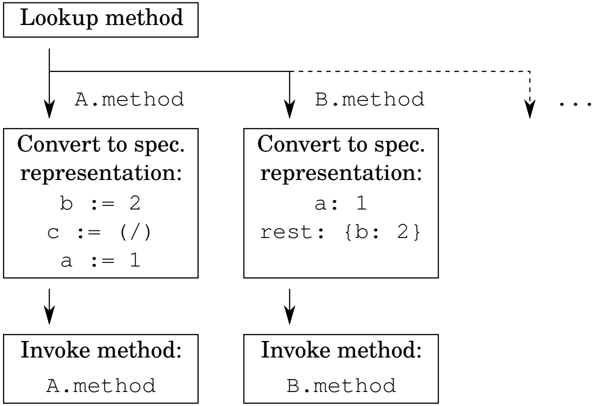
```
def A.method(b: 10,
    c: 20, a: 30)
  ...
end

obj.method(a: 1, b: 2)
```

Convert Before Invocation: Call-target-specific Arguments



```
def A.method(b: 10,
    c: 20, a: 30)
  ...
end

def B.method(a:,
    **rest)
  ...
end

obj.method(a: 1,
    b: 2)
```

Convert Before Invocation: Call-target-specific Arguments



Call-target-specific Method Arguments

- Code/AST for generating arguments representation depends on call target
- Caching one AST subtree generating the arguments array per PIC entry
- Call-target-specific argument handling is part of the PIC

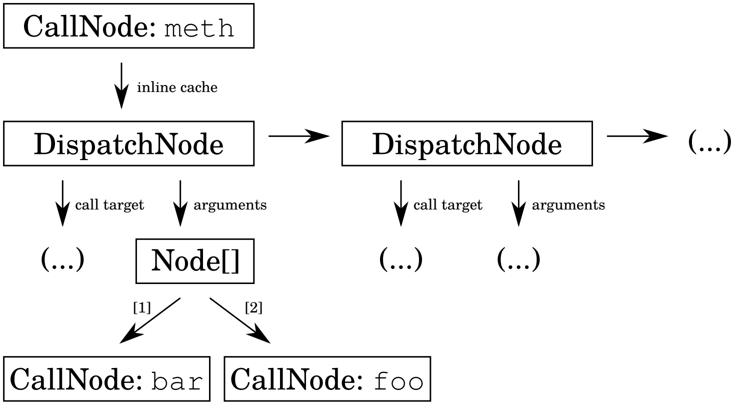
PIC Argument Cache

- Truffle: AST Interpreter Framework
- PIC implemented as linked list of AST nodes
- Caching one AST subtree generating the array of arguments per PIC entry;
  No bytecode manipulations neccessary

Execution Order of Argument Nodes

```
def meth(a: 0, b: 0)
  ...
end
```

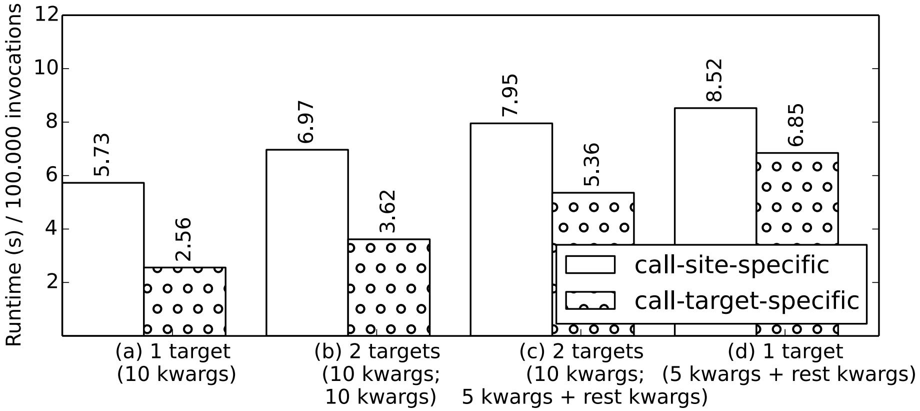
```
meth(b: foo(), a: bar())
```



Megamorphic Call Sites

- Call site switches to *megamorphic* once the PIC treshold is reached
- Megamorphic call sites use call-site-specific method argument (old behavior)
- Call target is able to detect whether call is *optimized* (call-target-specific args) or *unoptimized* (call-site-specific args)

Micro-Benchmarks

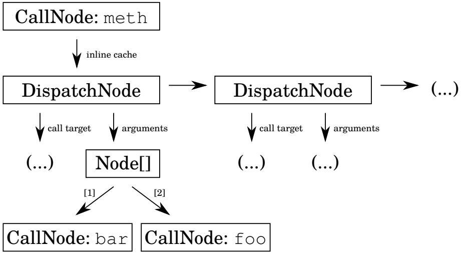


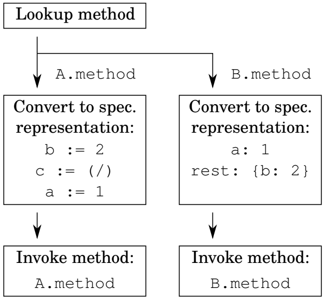
Micro-Benchmarks (b)

```
class A
  def foo_b(a:1, b:2, c:3, d:4, e:5, f:6, g:7, h
     :8, i:9, j:10)
    a + b + c + d + e + f + g + h + i + j
  end
end

class B
  def foo_b(j:11, i:12, h:13, g:14, f:15, e:16, d
     :17, c:18, b:19, a:20)
    a + b + c + d + e + f + g + h + i + j
  end
end

obj.foo_b(a:1, b:2, c:3, d:4, e:5, f:6, g:7, h:8,
    i:9, j:10)
```

Summary

- Call-site-specific method arguments: an **optimization for method argument handling** in dynamically-typed languages
- Call sites can have multiple polymorphic call targets
- **Prepare arguments** for call target at call site
- Only efficient if **call target analysis is cached** at the call site (as part of the PIC)





MagLev

- MagLev: a Ruby implementation in Smalltalk (GemStone/S).
- Compiled to byte code for a Smalltalk virtual machine
- Generates a number of wrapper (*bridge*) methods for different method arguments.

```
def method(a, b = 1, *args)
...
end

def method#1(a)
# call method(a, 1)
end

def method#3(a, b, c, d, e)
# call method(a, b, [c, d, e])
end
```