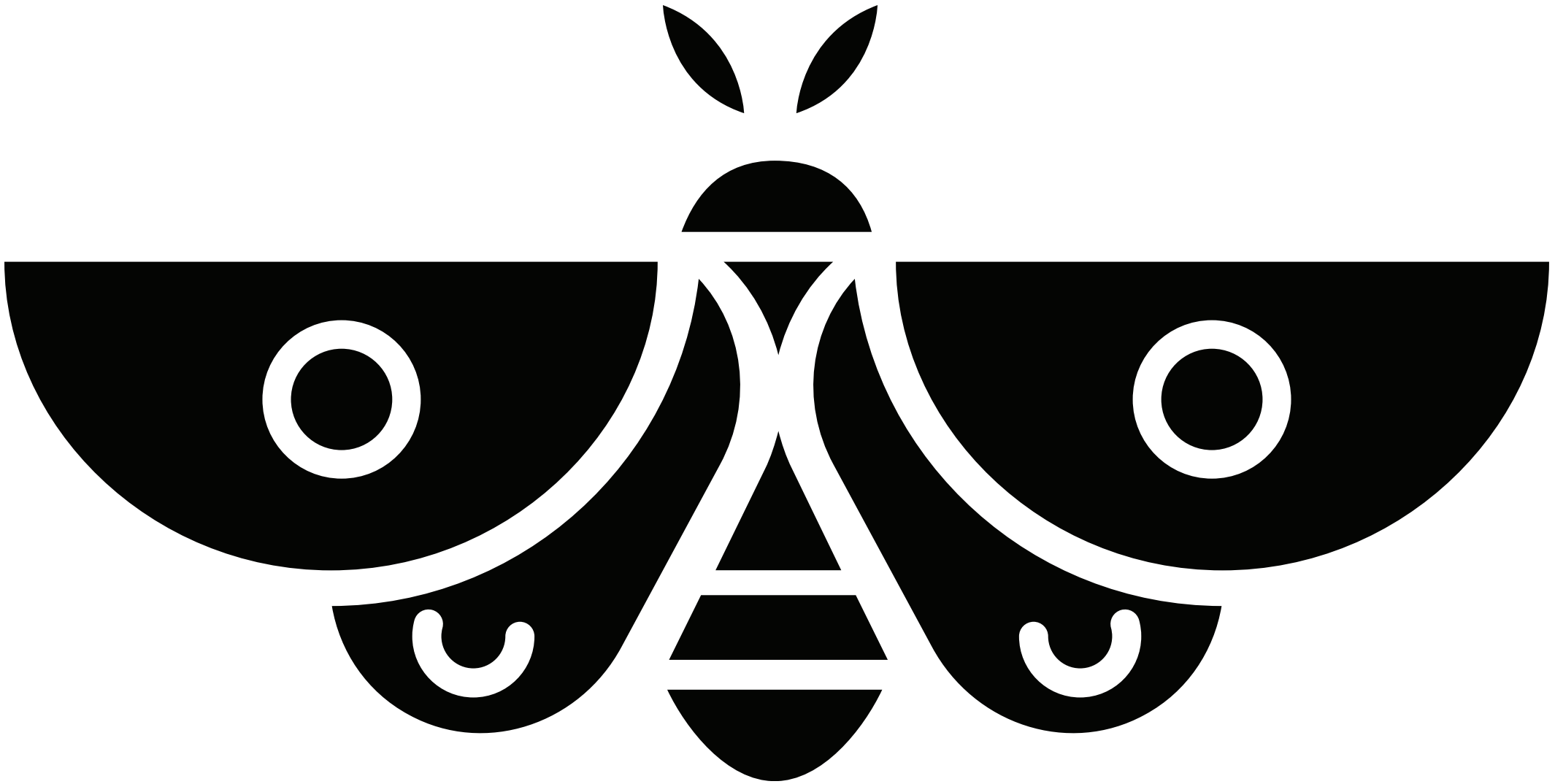


Transient Typechecks are (almost) Free

Richard Roberts, Stefan Marr
Michael Homer, James Noble

Transient Performance

"*The transient approach checks types at uses, so the act of adding types to a program introduces more casts and may slow the program down (even in fully typed code).*" … "*transient semantics…is a* ***worst case scenario****…, there is a cast at almost every call*"

Chung, Li, Nardelli and Vitek, ECOOP 2018

"*imposes a run-time checking overhead that is* ***directly proportional*** *to the number of [type annotations] in the program*"

Greenman and Felleisen, ICFP 2018

"*clear trend that adding type annotations adds performance overhead. The increase is typically* ***linear****.*"

Greenman and Migeed, PEPM 2018

Microbenchmarks

```
method foo9(xa : A, xb : B, xc : C, xd : D, xe : E) {
  count := count + 1
  foo8(a,b,c,d,e)
}

method foo8(xa : A, xb : B, xc : C, xd : D, xe : E) {
  count := count + 1
  foo7(a,b,c,d,e)
}

method foo7(xa : A, xb : B, xc : C, xd : D, xe : E) {
  count := count + 1
  foo6(a,b,c,d,e)
}
```

Microbenchmarks

```
method foo9(xa    , xb    , xc    , xd    , xe    ) {
  count := count + 1
  foo8(a,b,c,d,e)
}

method foo8(xa    , xb    , xc    , xd    , xe    ) {
  count := count + 1
  foo7(a,b,c,d,e)
}

method foo7(xa    , xb    , xc    , xd    , xe    ) {
  count := count + 1
  foo6(a,b,c,d,e)
}
```

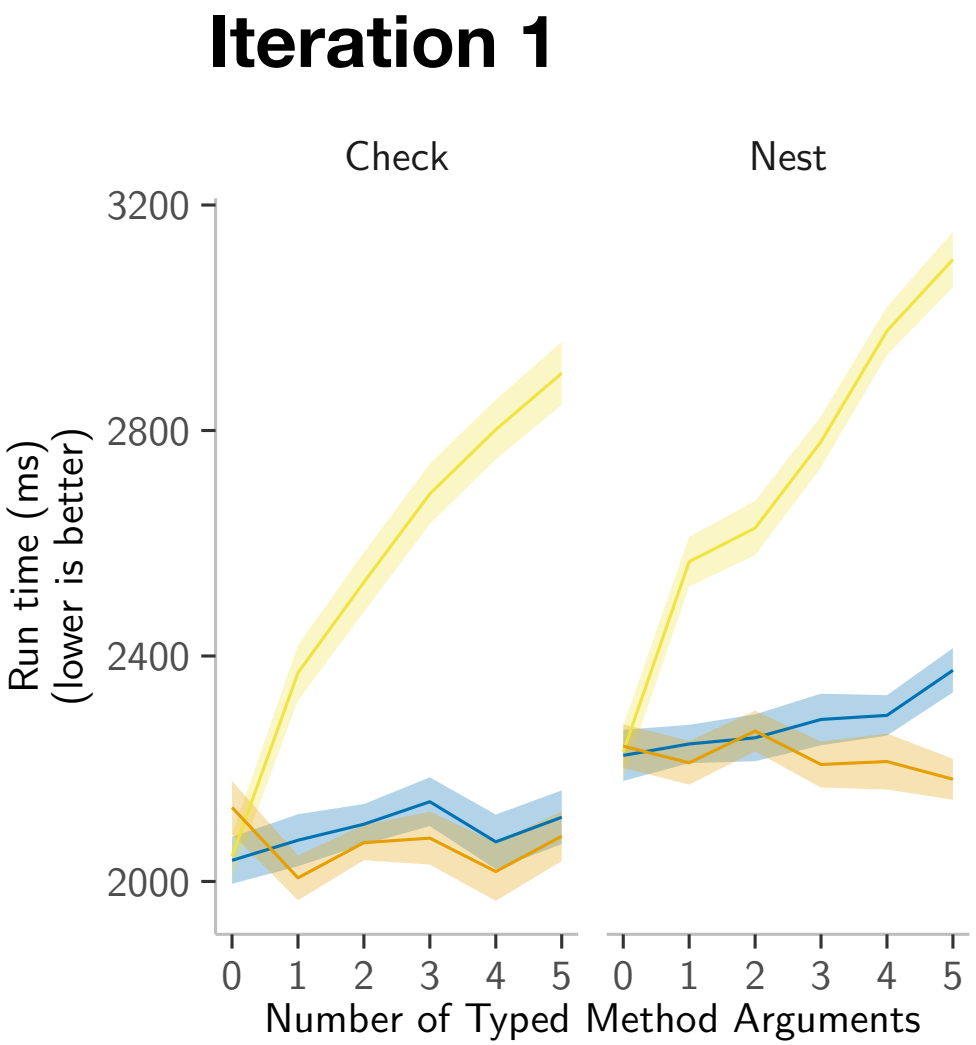
Microbenchmarks

```
method foo9(xa : A, xb : B, xc : C, xd    , xe    ) {
  count := count + 1
  foo8(a,b,c,d,e)
}

method foo8(xa : A, xb : B, xc : C, xd    , xe    ) {
  count := count + 1
  foo7(a,b,c,d,e)
}

method foo7(xa : A, xb : B, xc : C, xd    , xe    ) {
  count := count + 1
  foo6(a,b,c,d,e)
}
```

Microbenchmarks

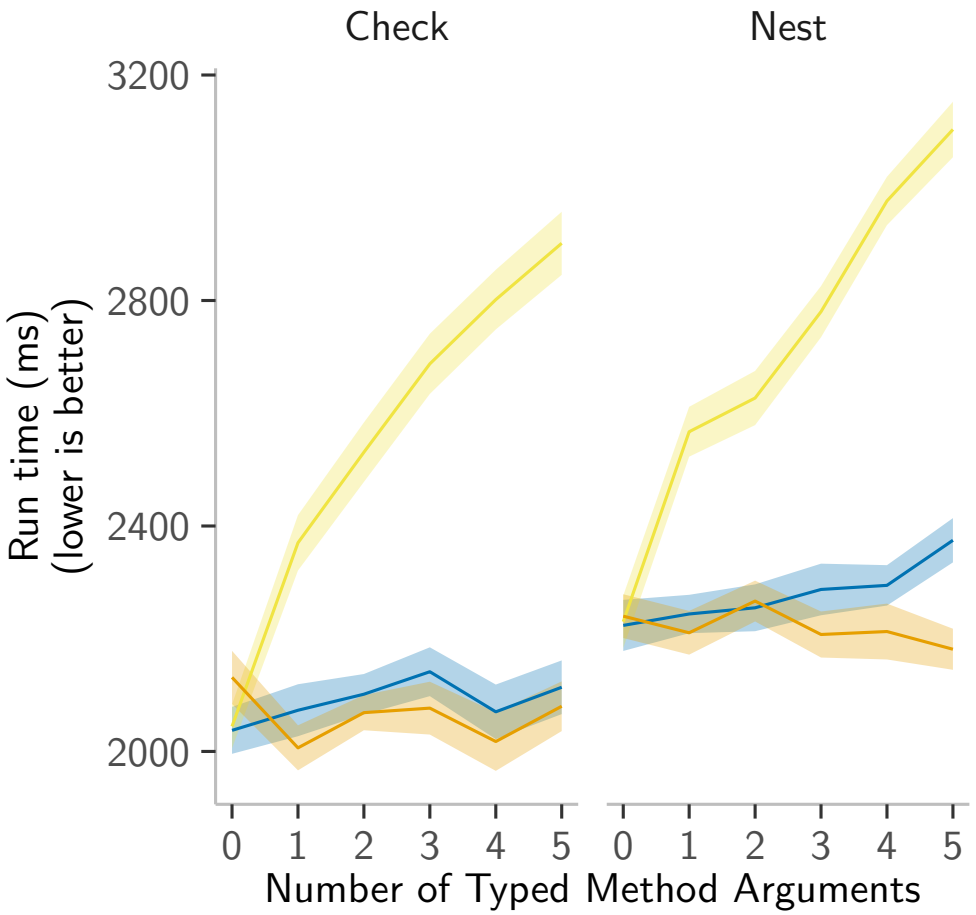
```
method foo9(xa : A, xb : B, xc : C, xd : D, xe : E) {
  count := count + 1
  foo8(a,b,c,d,e)
}

method foo8(xa : A, xb : B, xc : C, xd : D, xe : E) {
  count := count + 1
  foo7(a,b,c,d,e)
}

method foo7(xa    , xb    , xc    , xd    , xe    ) {
  count := count + 1
  foo6(a,b,c,d,e)
}
```

Are We Fast Yet?

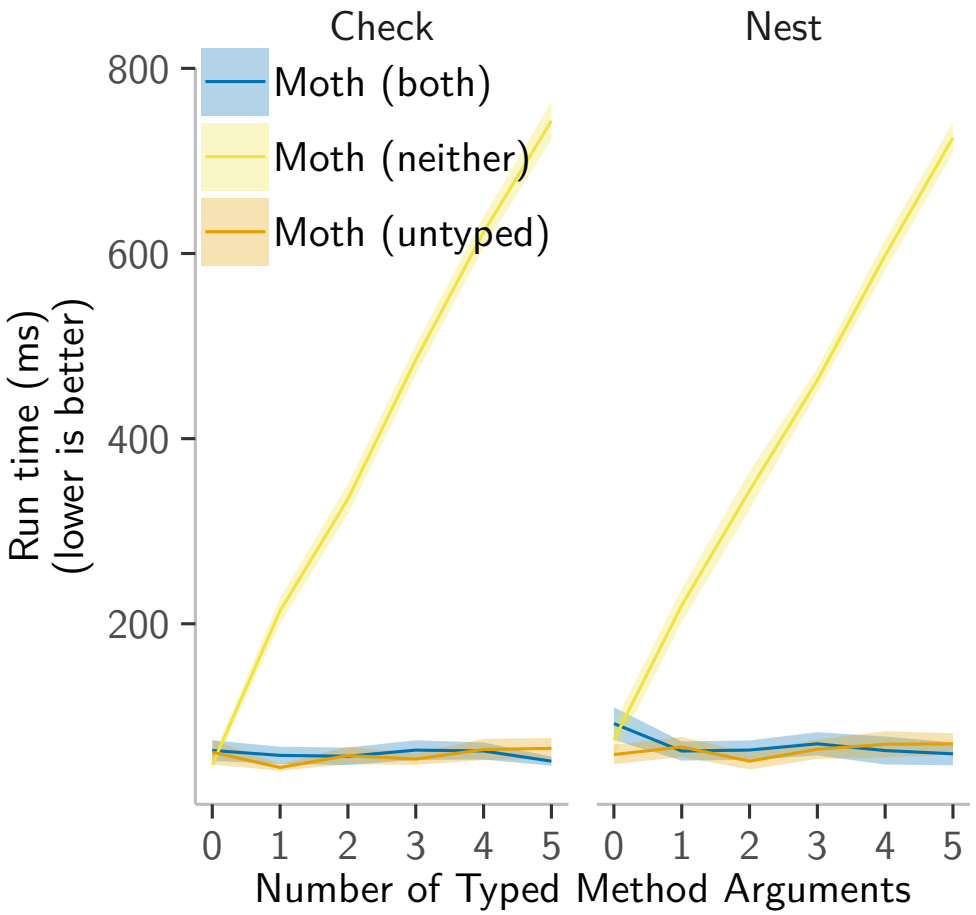


Are We Fast Yet?

Iteration 1



Iteration 100



Transient Typechecks are (almost) Free

Richard Roberts, Stefan Marr
Michael Homer, James Noble


Ennce

Goals

Not *require* type annotations

- Dynamic types must be checked
- Checking must be cheap
- Run statically incorrect code

Lightweight Implementation


INGSOC

Every object has a class
Methods, fields, constants
Multipart names
Blocks for control
Non-local returns
Optionally typed
Modules as classes
Classes inside classes

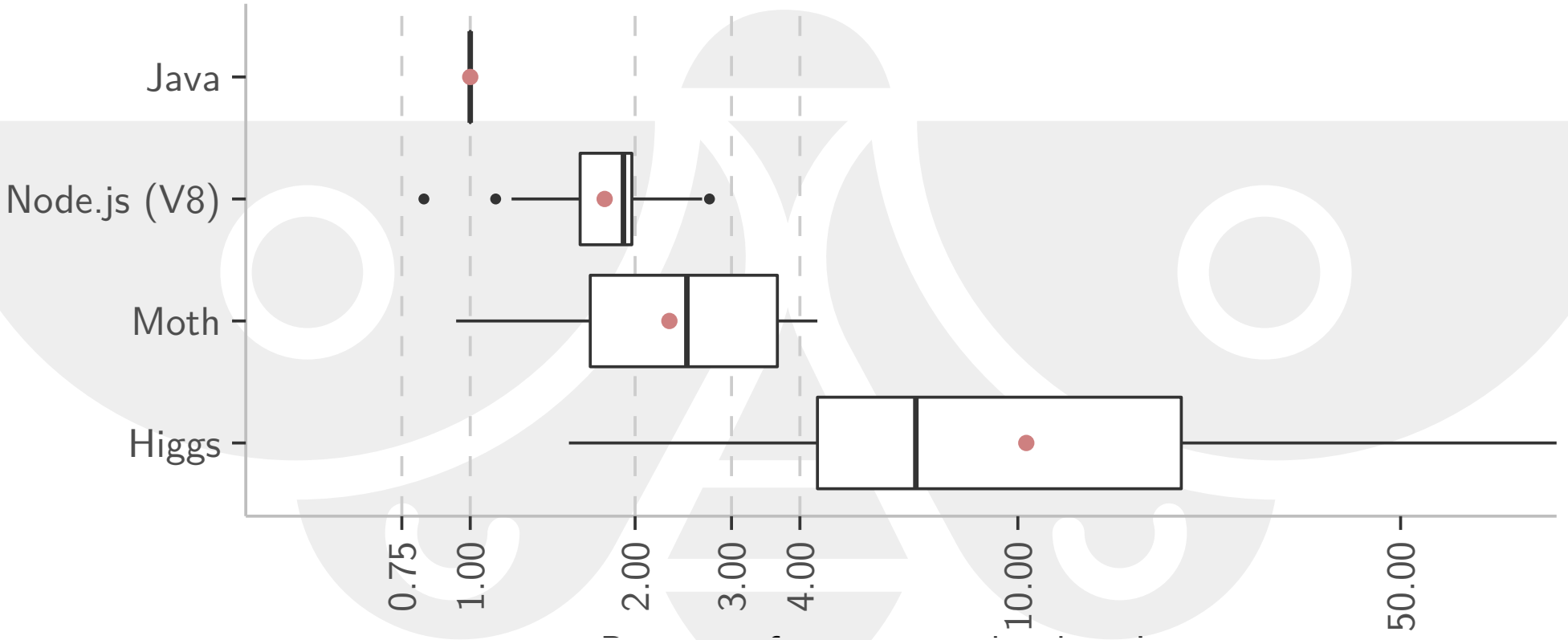
Everything is an object
Methods, fields, constants
Multi-part & arity names
Blocks for control
Non-local returns
Optionally & gradually typed
Modules as objects
Classes inside classes
Objects inside methods


Moth

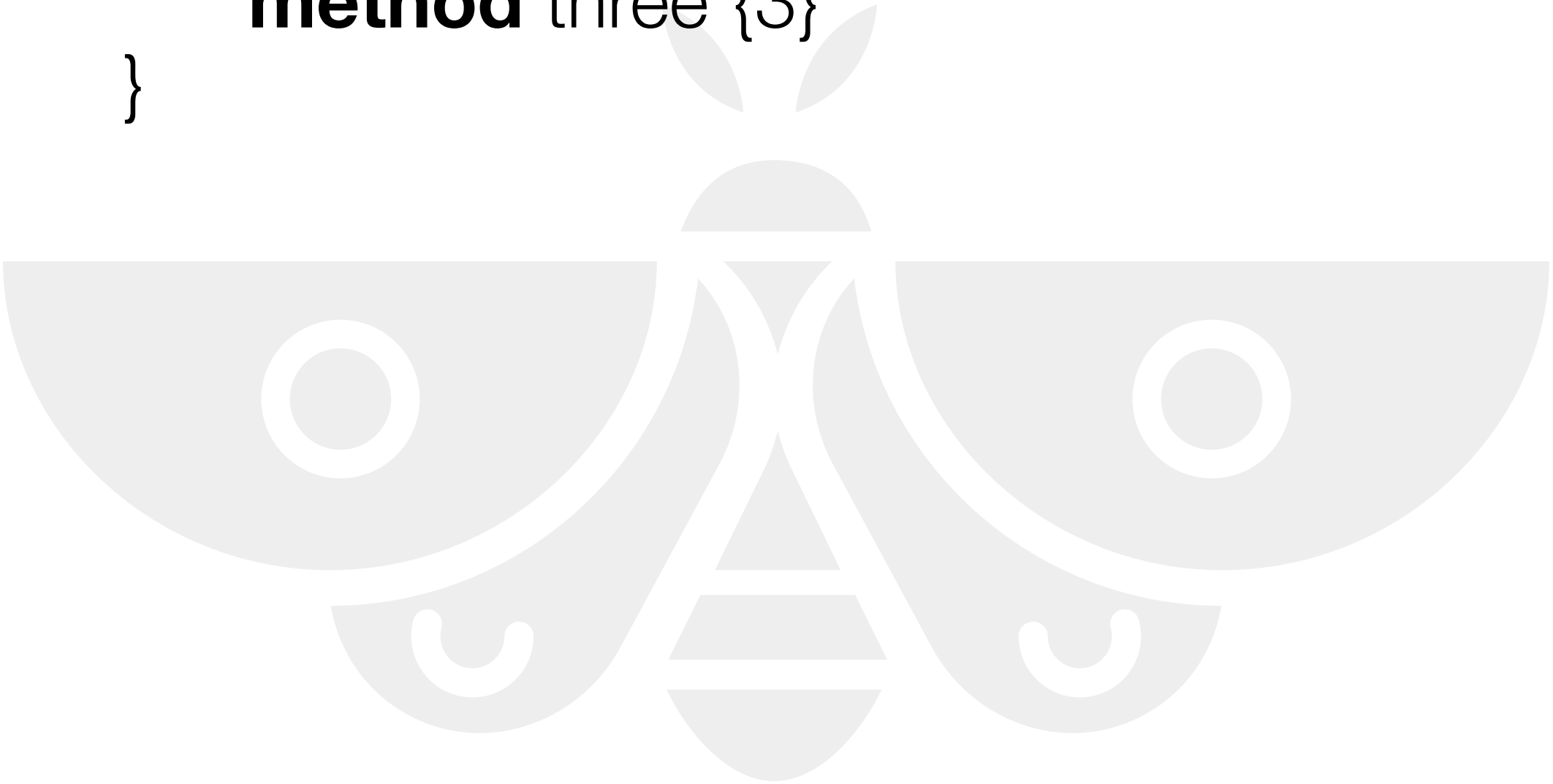

SOMNS
SimpleObjectMachine


GraalVM™

Are We Fast Yet?



```
def o = object {
    method three {3}
}
```



```
def o = object {
    method three {3}
}

o.three
```

```
def o = object {
    method three {3}
}

type Three = interface {
    three
}
```

```
def o = object {
    method three {3}
}

type Three = interface {
    three
}

def p : Three = o

p.three
```

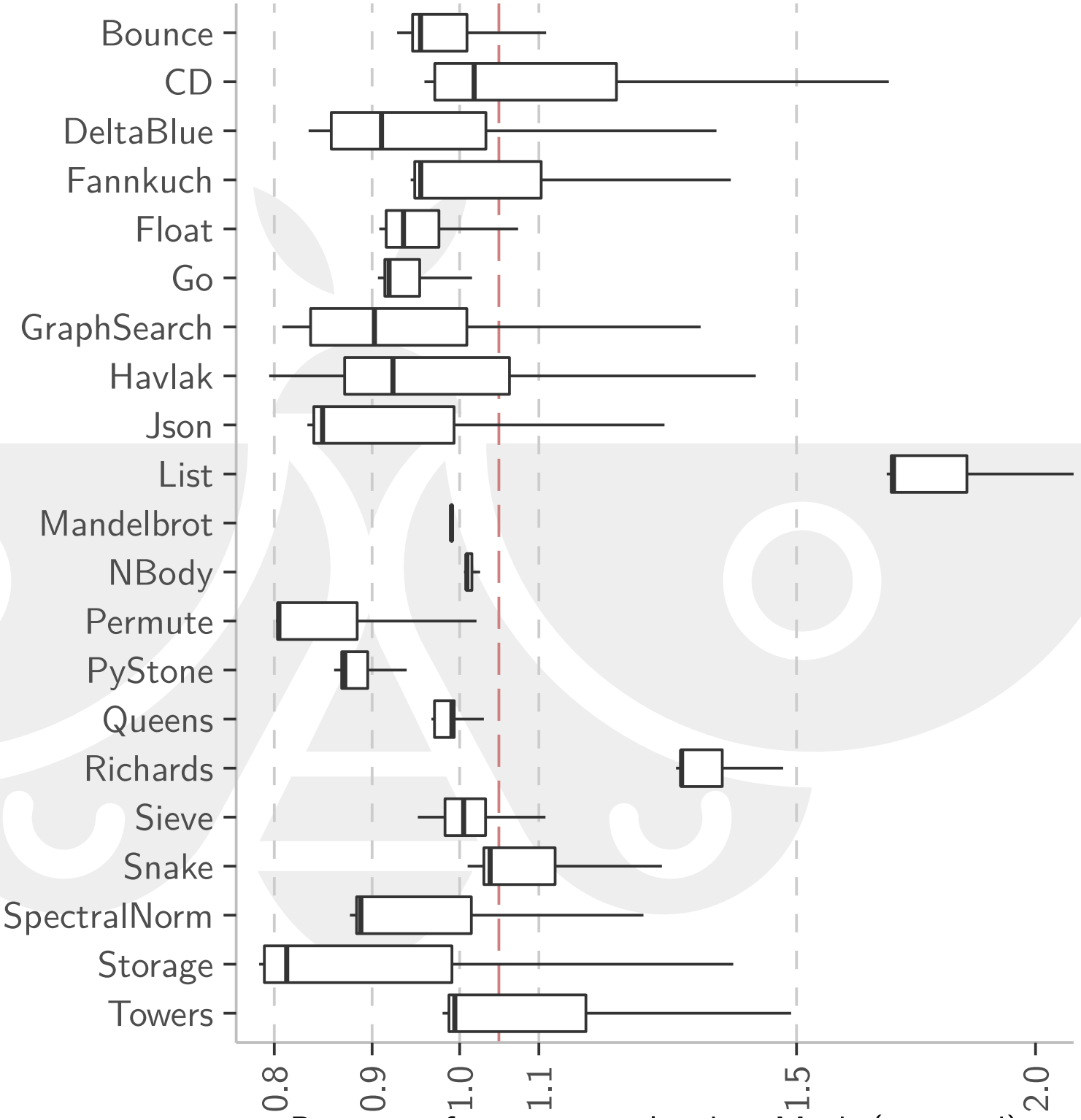
```
def o = object {
    method three {3}
}

type Three = interface {
    three
}

method wantsThree( trois : Three ) { }

wantsThree( o )
```

```
def o = object {
    method four {3}
}

type Three = interface {
    three
}

method wantsThree( trois : Three ) { }

wantsThree( o ) //  should crash!
```

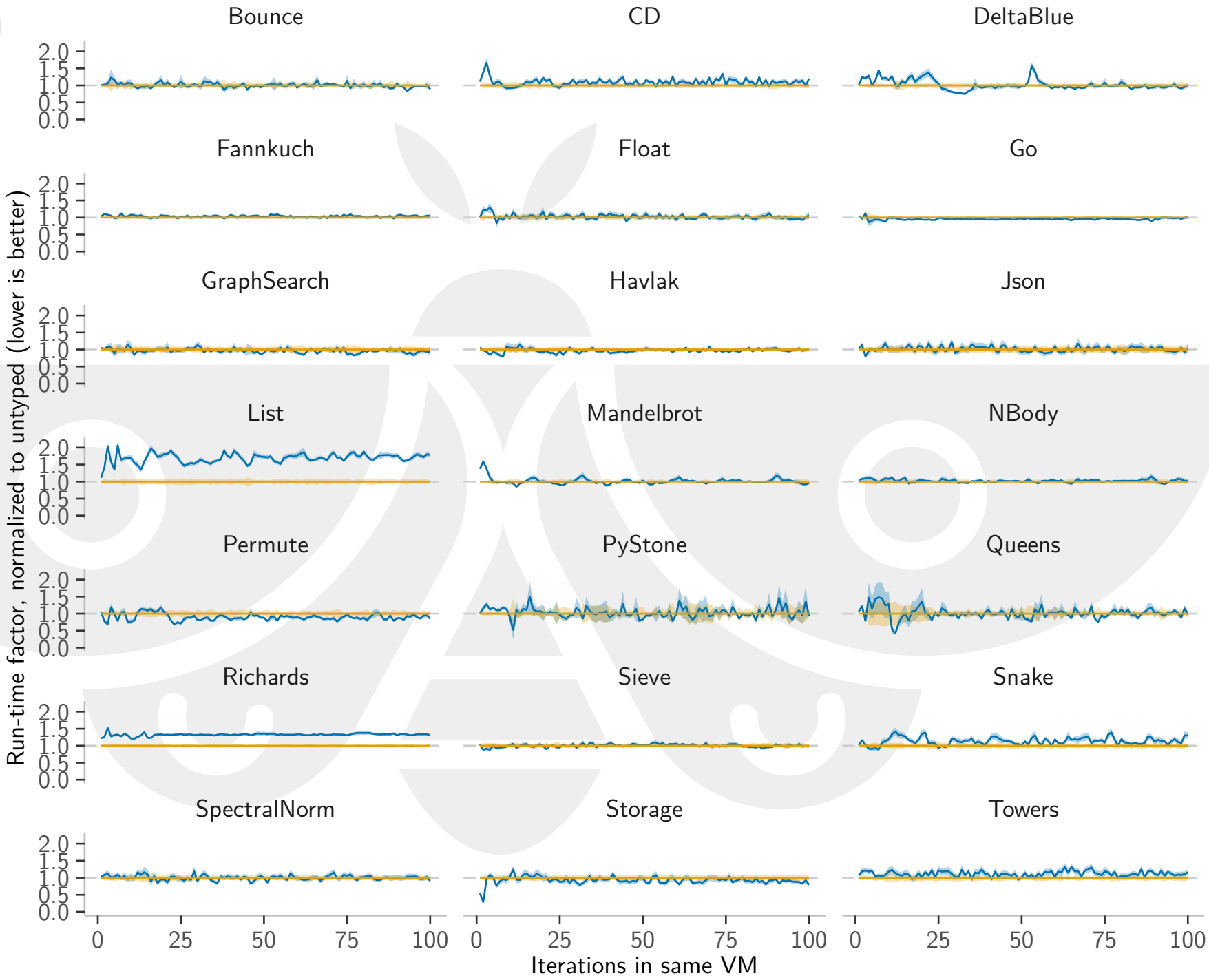
Transient Typechecks

```
method wantsThree( trois : Three ) { }

method wantsThree( trois ) {
   assert { Three.match(trois) }
}
```

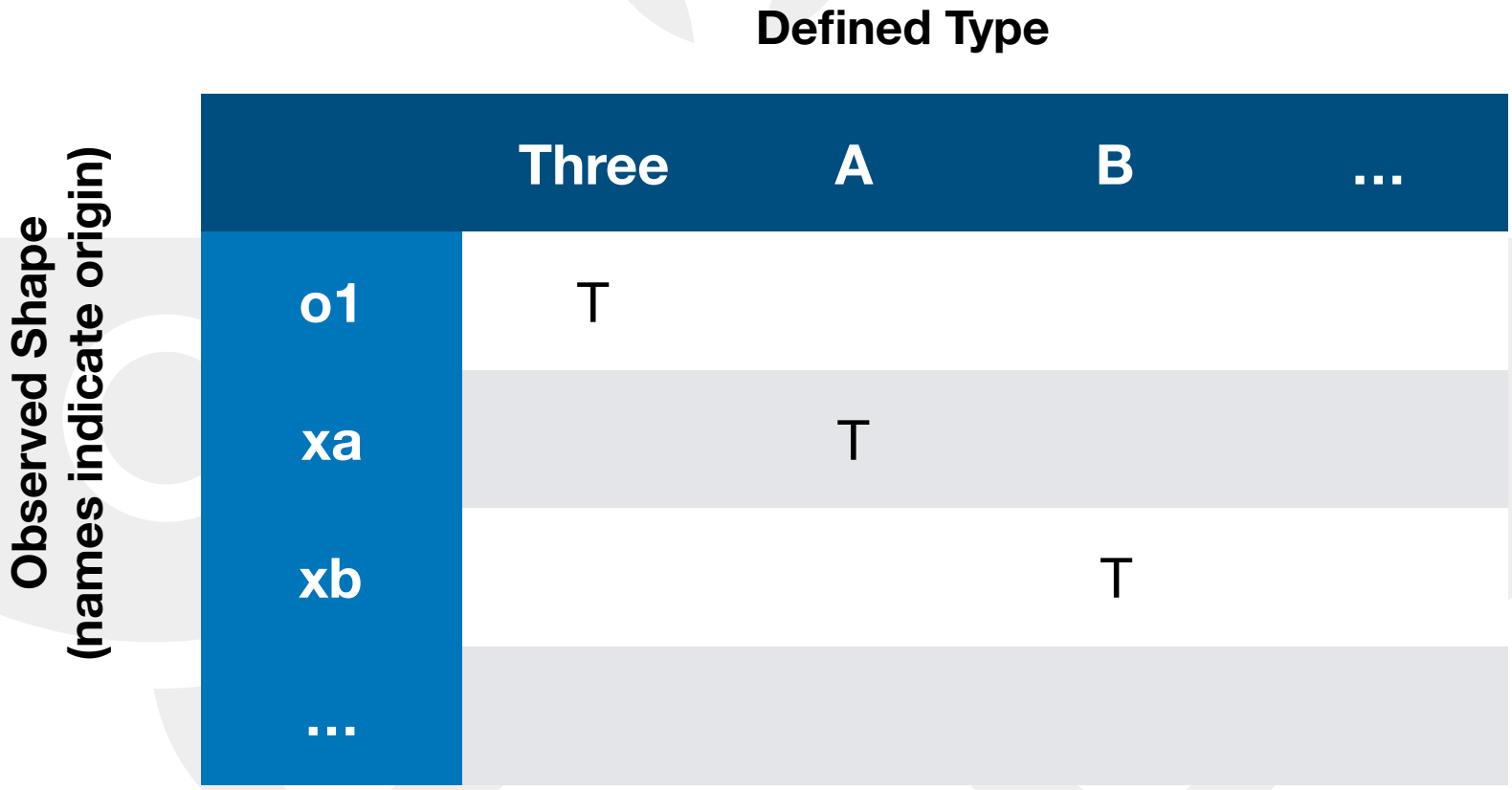
Transient Overhead



Warmup


Bounce
CD
DeltaBlue
Fannkuch
Float
Go
GraphSearch
Havlak
Json
List
Mandelbrot
NBody
Permute
PyStone
Queens
Richards
Sieve
Snake
SpectralNorm
Storage
Towers
Run-time factor, normalized to untyped (lower is better)
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
Iterations in same VM

Subtype Cache

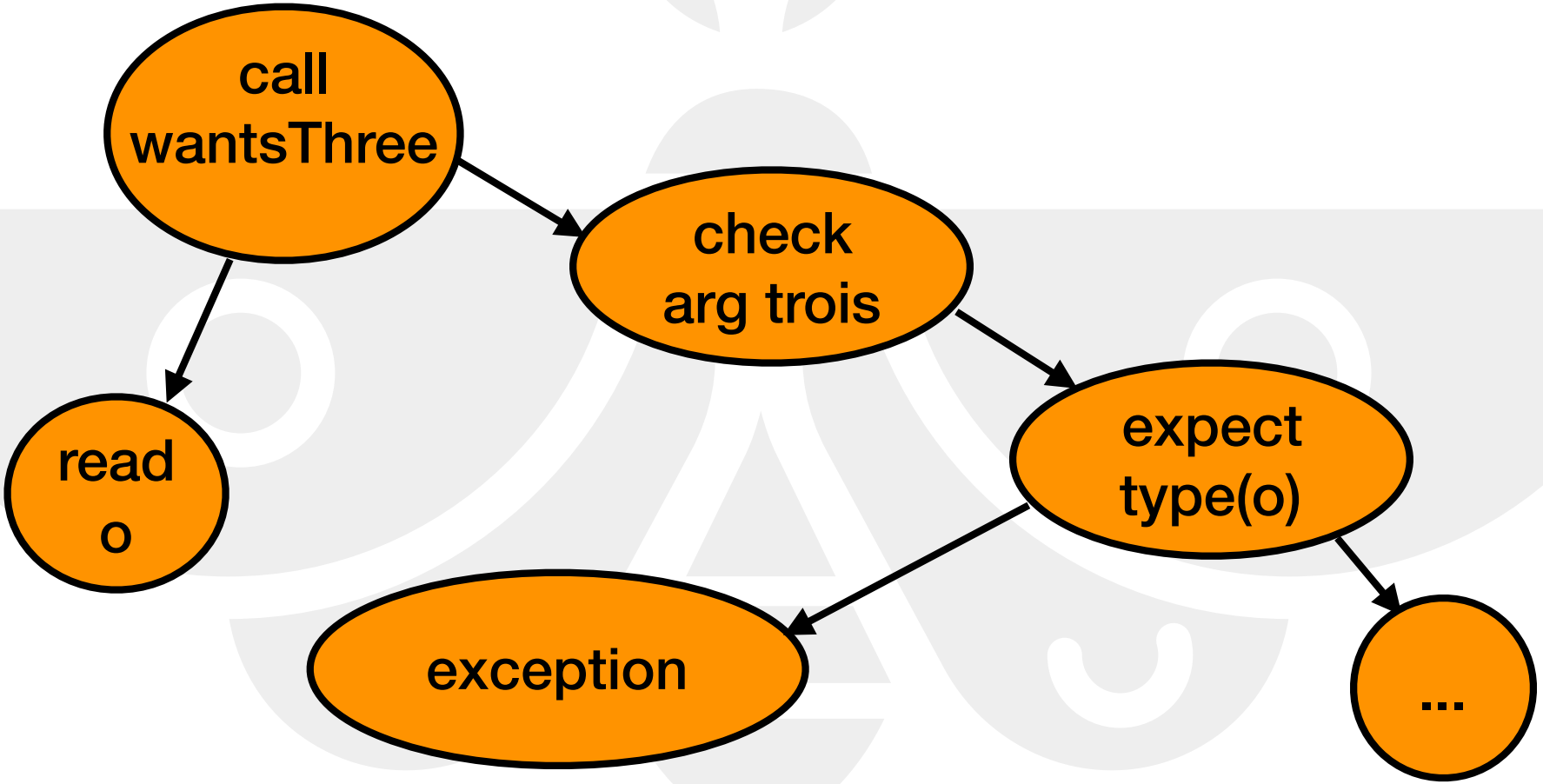


method wantsThree(trois : Three) { }
wantsThree(o)

Subtype Cache

```
1  global record: Matrix
2
3  class TypeCheckNode(Node):
4
5    expected: Type
6
21   @Fallback
22   def check(obj: Any):
23     T = obj.get_type()
24
25     if record[T, expected] is unknown:
26       record[T, expected] = T.is_subtype_of(expected)
27
28     if not record[T, expected]:
29       raise TypeError(f"{obj} doesn't implement {expected}")
```

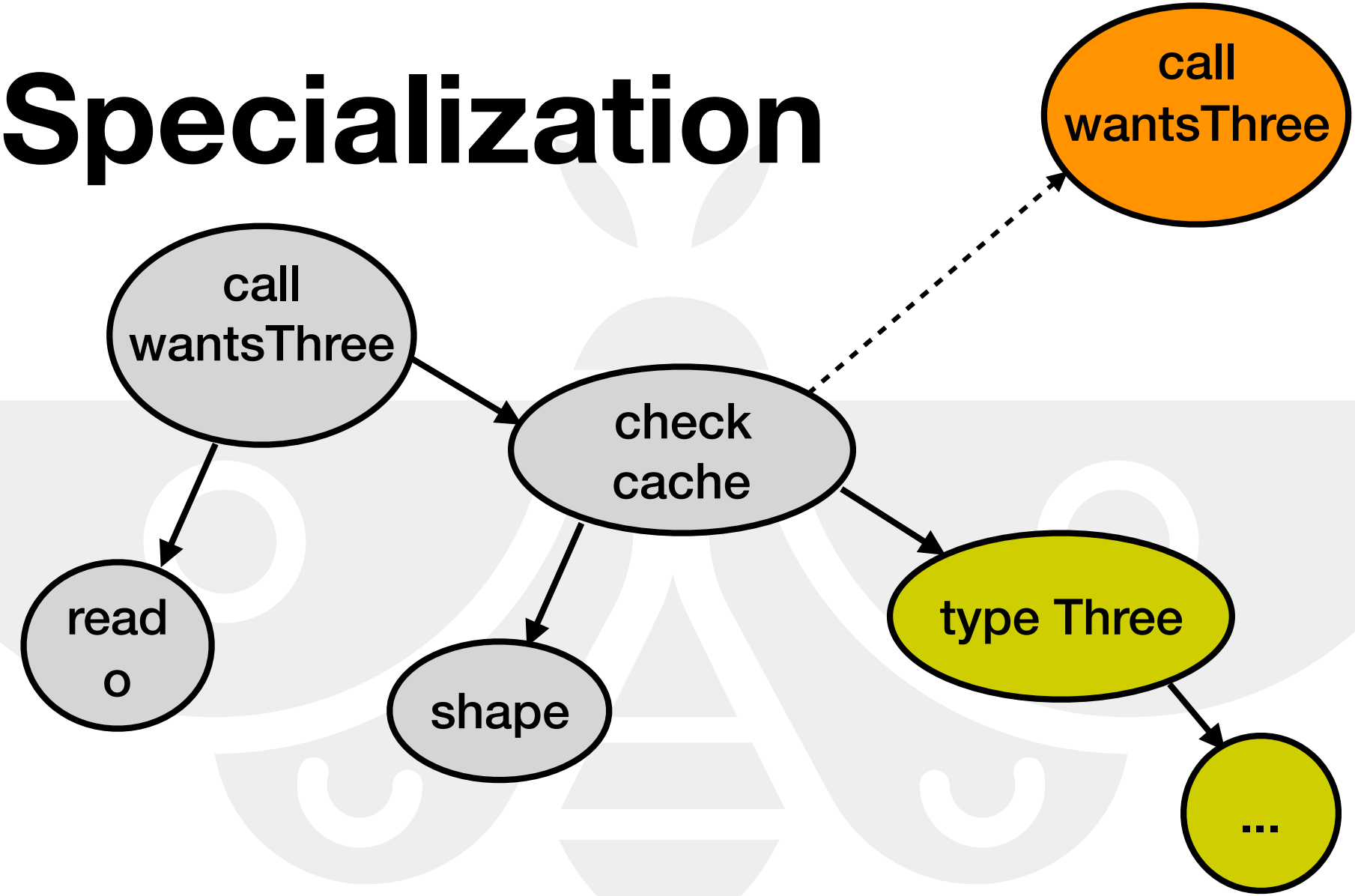
Specialization



method wantsThree(trois : Three) { }
wantsThree(o)

 After Würthinger et al, *One VM to Rule them all* Onward! 2013

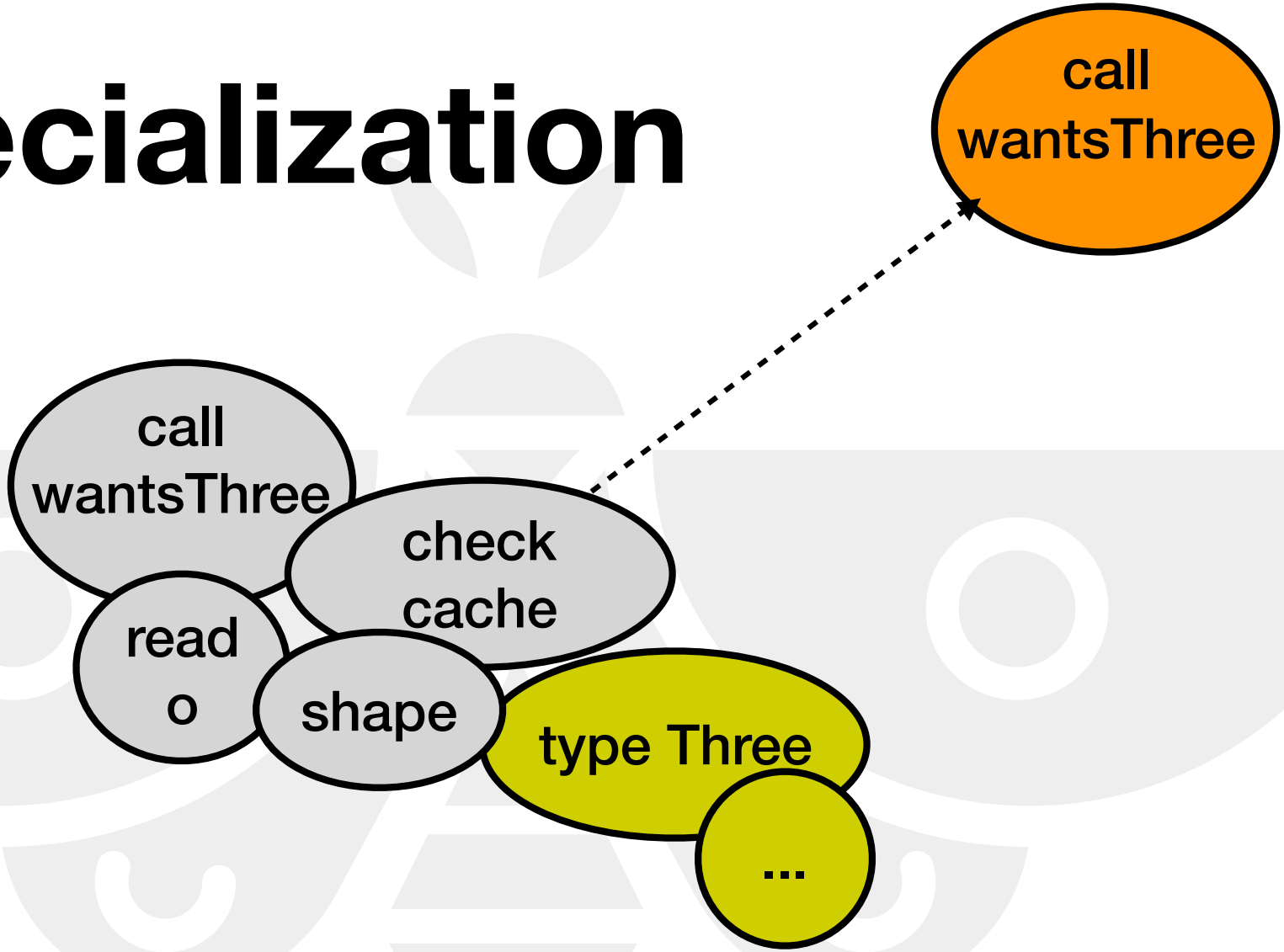
Specialization



method wantsThree(trois : Three) { }
wantsThree(o)

 After Würthinger et al, *One VM to Rule them all* Onward! 2013

Specialization

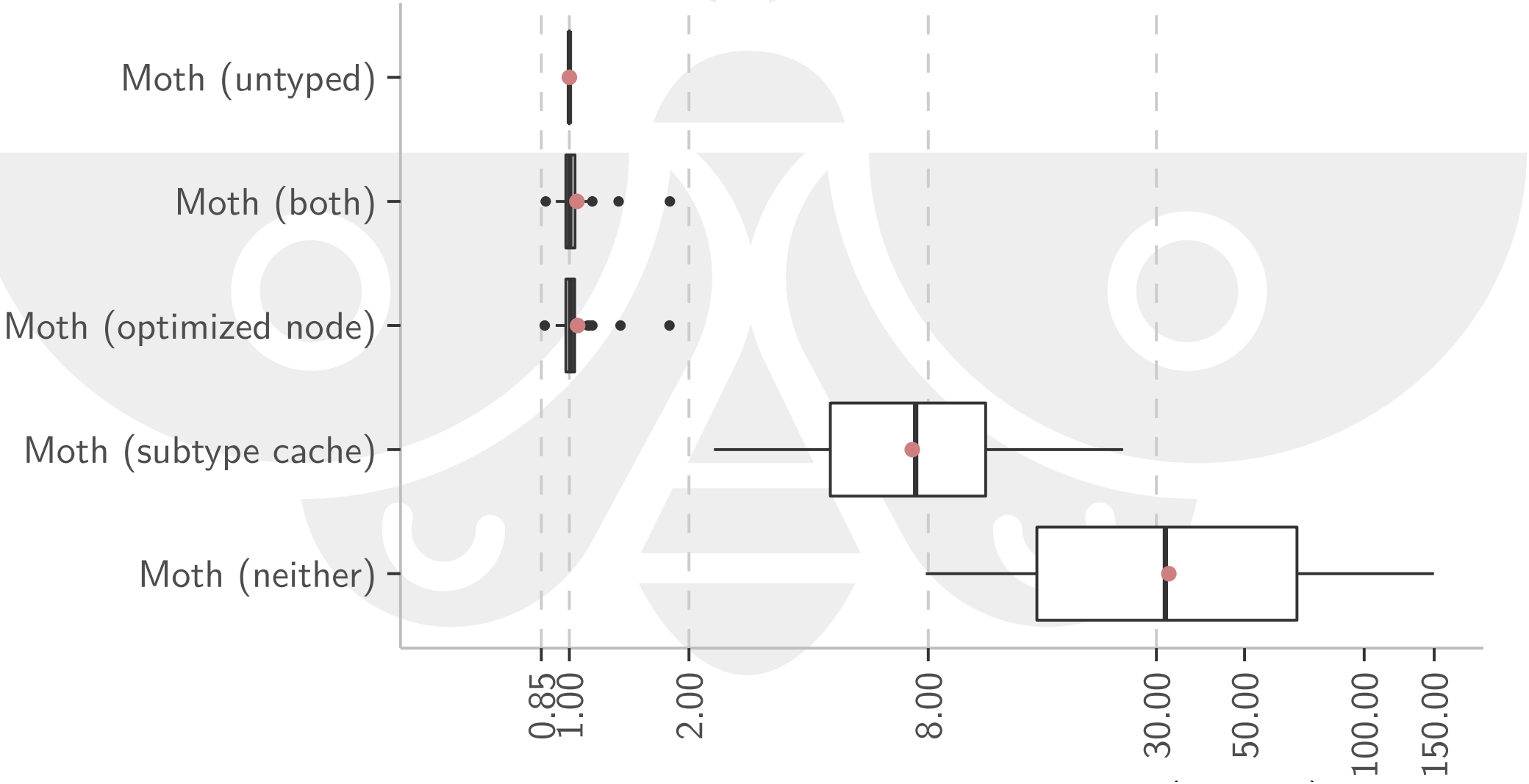


method wantsThree(trois : Three) { }
wantsThree(o)

Specialization

```
7   @Spec(static_guard=`expected.check(obj)`)
8   def check(obj: Number):
9     pass
10
11  @Spec(static_guard=`expected.check(obj)`)
12  def check(obj: String):
13    pass
14
15  ...
16
17  @Spec(guard=`obj.shape==cached_shape`, static_guard=`expected.check(obj)`)
18  def check(obj: Object, @Cached(obj.shape) cached_shape: Shape):
19    pass
20
21  @Fallback
22  def check(obj: Any):
23    T = obj.get_type()
24
```

Optimizations



Optimizations

Type Test	Enabled Optimization	mean #invocations	min	max
check_generic	Neither	137,525,845	11,628,068	896,604,537
	Subtype Cache	137,525,845	11,628,068	896,604,537
	Optimized Node	292	68	1,012
	Both	292	68	1,012
is_subtype_of	Neither	134,125,215	11,628,067	896,604,534
	Subtype Cache	16	10	29
	Optimized Node	292	68	1,012
	Both	16	10	29

Pathology

```
1 var elem: ListElement := headOfList
2 while (...) do {
3    elem := elem.next
4 }
```

79%

Local Semantics

```
def o = object {
      method three -> Unknown {3}
}
type ThreeString = interface {
      three -> String
}
def t : ThreeString = o
printString (t.three)
```

Lexical Semantics

```
def o = object {
      method three -> Unknown {3}
}
type ThreeString = interface {
      three -> String
}
def t : ThreeString = o
printString (t.three)
```

Shallow Semantics

```
def o = object {
        method three -> Number {3}
}
type ThreeString = interface {
        three -> String
}
method wantsThree( trois : ThreeString ) {}
wantsThree( o )
```

Deep Semantics

```
def o = object {
        method three -> Number {3}
}
type ThreeString = interface {
        three -> String
}
method wantsThree( trois : ThreeString ) {}
wantsThree( o )
```

Deep emulates Shallow

```
def o = object {
      method three -> Number {3}
}
type Three = interface {
      three -> Unknown
}
method wantsThree( trois : Three ) {}
wantsThree( o )
```

Concrete Semantics

```
def o = object {
        method three -> Unknown {3}
}
type ThreeString = interface {
        three -> String
}
method wantsThree( trois : ThreeString ) {}
wantsThree( o )
```

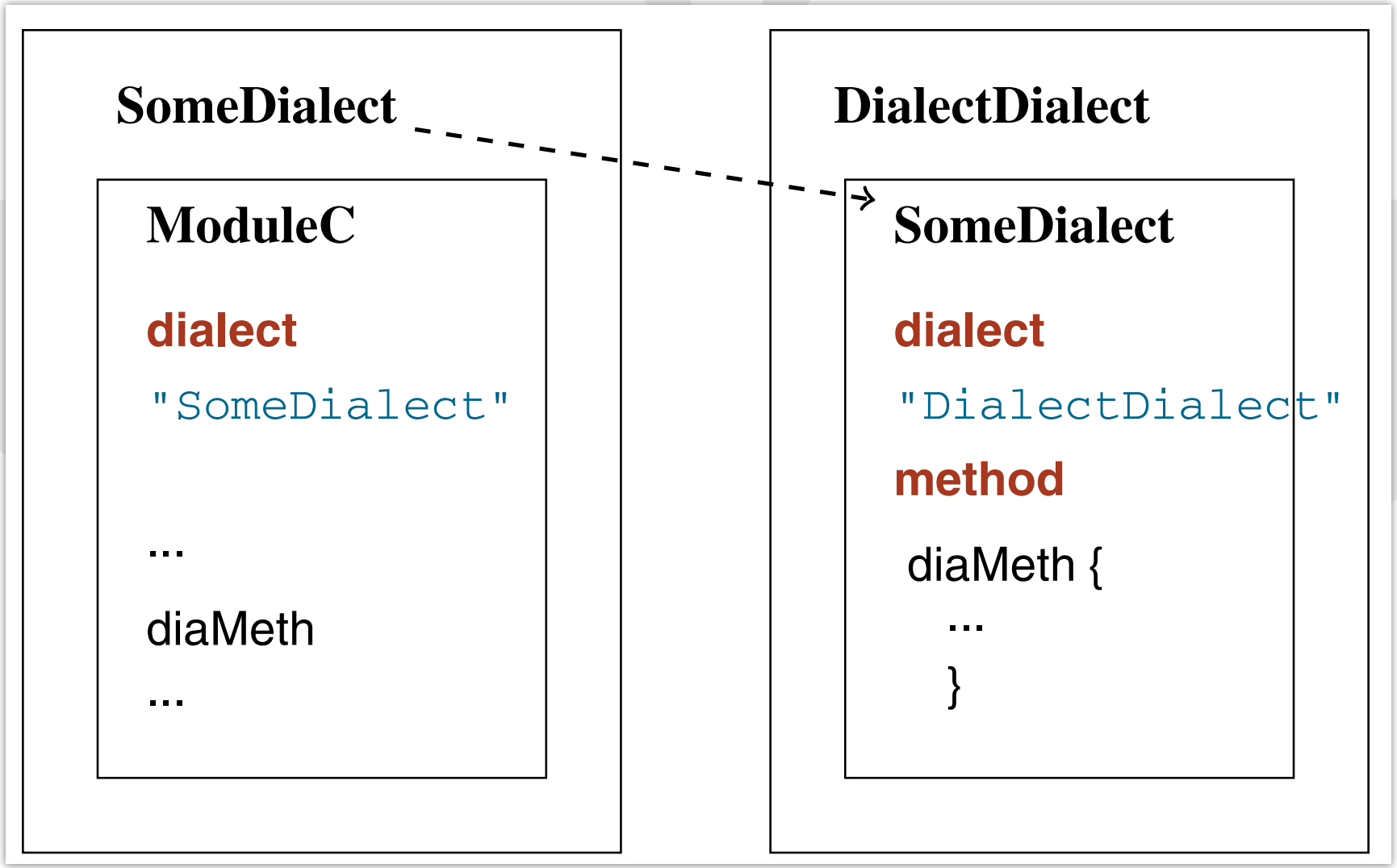
Graceful Semantics?

```
def o = object {
    method three -> Unknown {3}
}
type ThreeString = interface {
    three -> String
}
method wantsThree( trois : ThreeString ) {}
wantsThree( o )
```

Pathology

```
for (1.. innerIterations) do
 { i: Number ->
   system.advance(0.01)
 }

1.asInteger.to(innerIterations) do
 { i: Number ->
   system.advance(0.01)
 }
```

Dialects


SomeDialect
ModuleC
dialect
"SomeDialect"
...
diaMeth
...
DialectDialect
SomeDialect
dialect
"DialectDialect"
method
diaMeth {
...
}

Into the Gracer-verse?

```
method wantsThree( trois : ThreeString ) {

method wantsThree( trois ) {
   assert { ThreeString.match(trois) }
}

method wantsThree( trois” ) {
   def trois’ = ThreeString.match(trois”)
   assert { trois’ }
   def trois = trois’.result }
```

Even more semantics

Optional vs Mandatory
Structural vs Nominal
Erasure vs Shallow vs Deep
Symmetric vs Asymmetric
Local vs Lexical vs Reference vs Global
Identity vs Chaperones vs Coercions
Pure vs Impure
Crash vs Exceptions vs Warnings

Related Work

Computer Science > Programming Languages

Optimizing and Evaluating Transient Gradual Typing

Michael M. Vitousek, Jeremy G. Siek, Avik Chaudhuri

(Submitted on 20 Feb 2019)

Gradual typing enables programmers to combine static and dynamic typing in the same language. However, ensuring a sound interaction between the static and dynamic parts can incur significant runtime cost. In this paper, we perform a detailed performance analysis of the transient gradual typing approach implemented in Reticulated Python, a gradually typed variant of Python. The transient approach inserts lightweight checks throughout a program rather than installing proxies on higher order values. We show that, when running Reticulated Python and the transient approach on CPython, performance decreases as programs evolve from dynamic to static types, up to a 6x slowdown compared to equivalent Python

Conclusions?

Transient checks (almost) for free

Use a “real” VM

Steal one if you can

Dynamic vs Static optimisation

Many more gradual semantics…

github.com/gracelang/moth-SOMns