

Static Calls in a Dynamically-typed Language: From Implementation Challenges to Opportunities

IWST'26

Faouzi Rayane Mokhefi, Stéphane Ducasse, Luc Fabresse, Pablo Tesone, Guillermo Polito - 18/06/2026

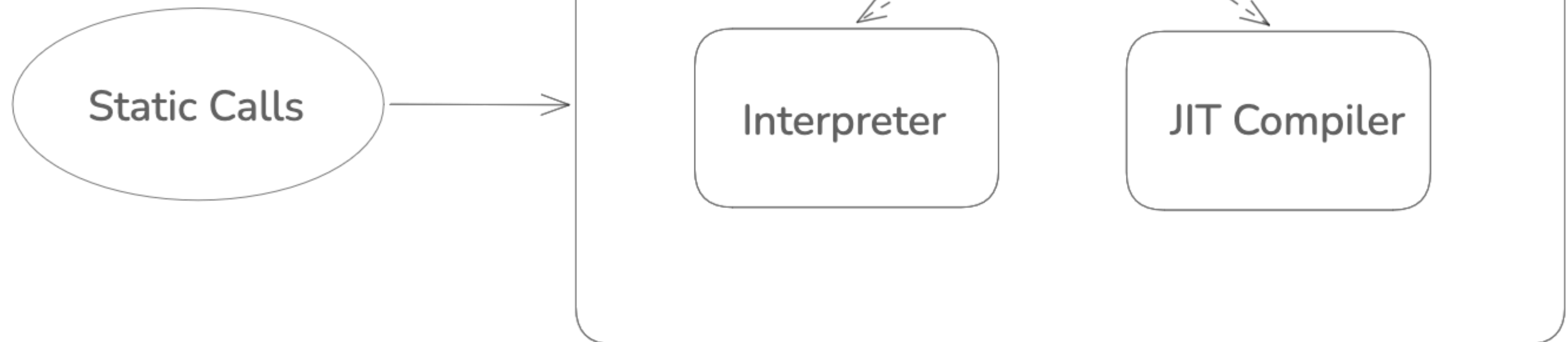


ESUG'26

International Workshop for Smalltalk Technologies 2026

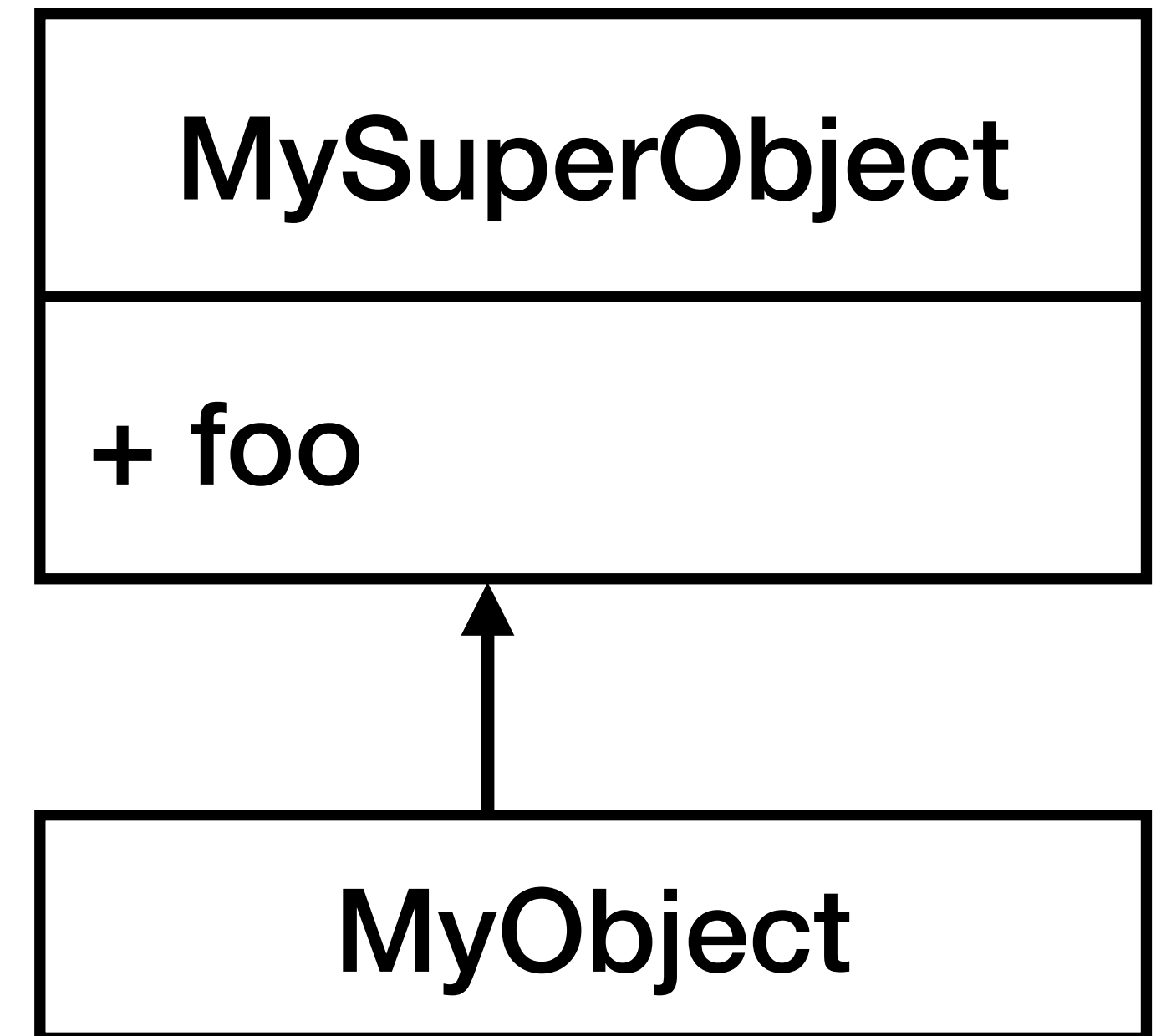
Contributions and Agenda

- **Two** static calls implementations in Pharo
 - Image-side support (syntax)
 - Interpreter-based (new bytecode)
 - JIT-based
- Evaluations



Context: Dynamically-typed Language

- Dynamic dispatch is **slow**
- Possible optimizations:
 - Caches (Global, inline)
 - Devirtualization

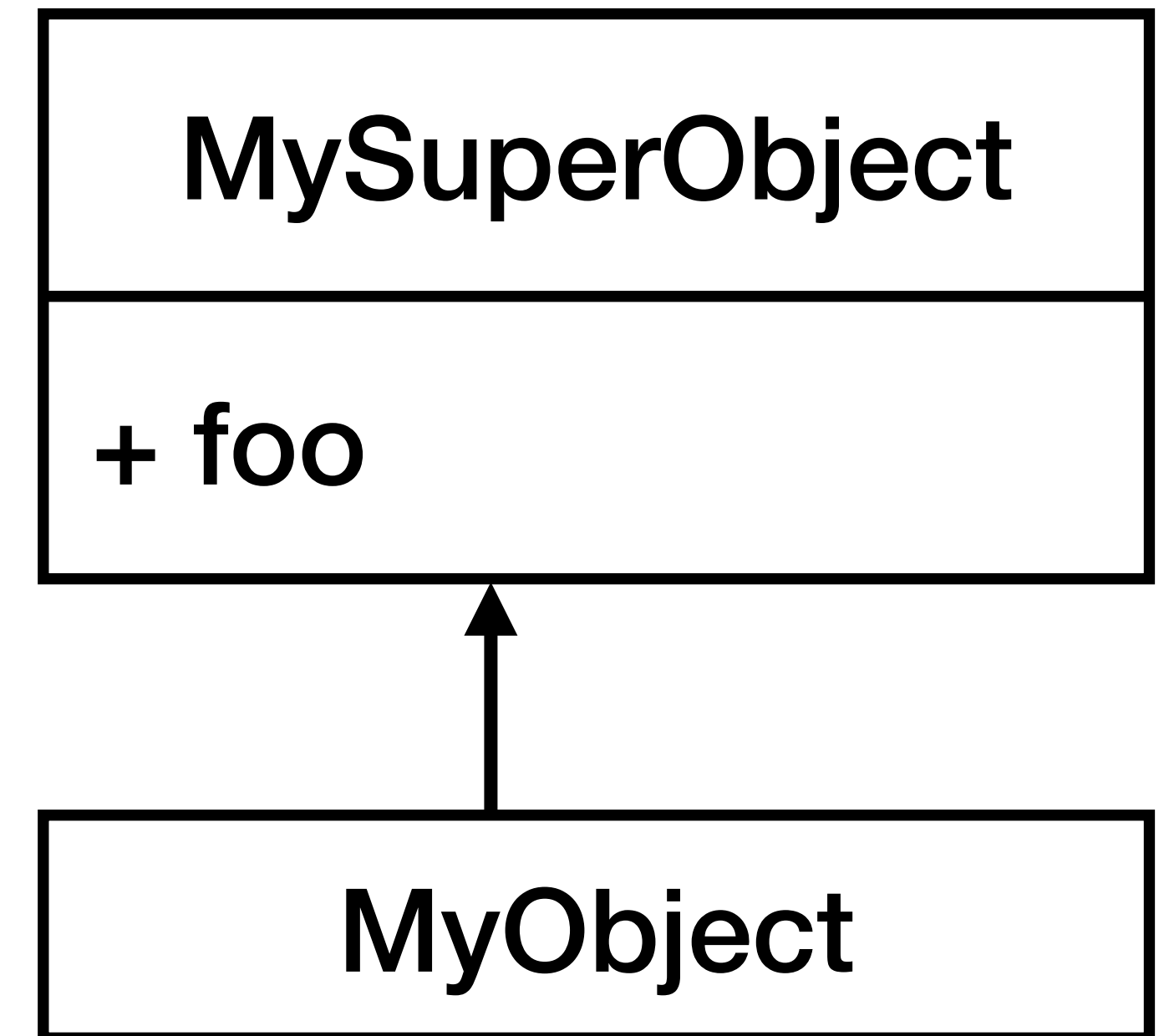


`o := MyObject new`

`o foo`

Context: Dynamically-typed Language

- Dynamic dispatch is **slow**
- Possible optimizations:
 - Caches (Global, inline)
 - Devirtualization



Call site

$o := \text{MyObject new}$

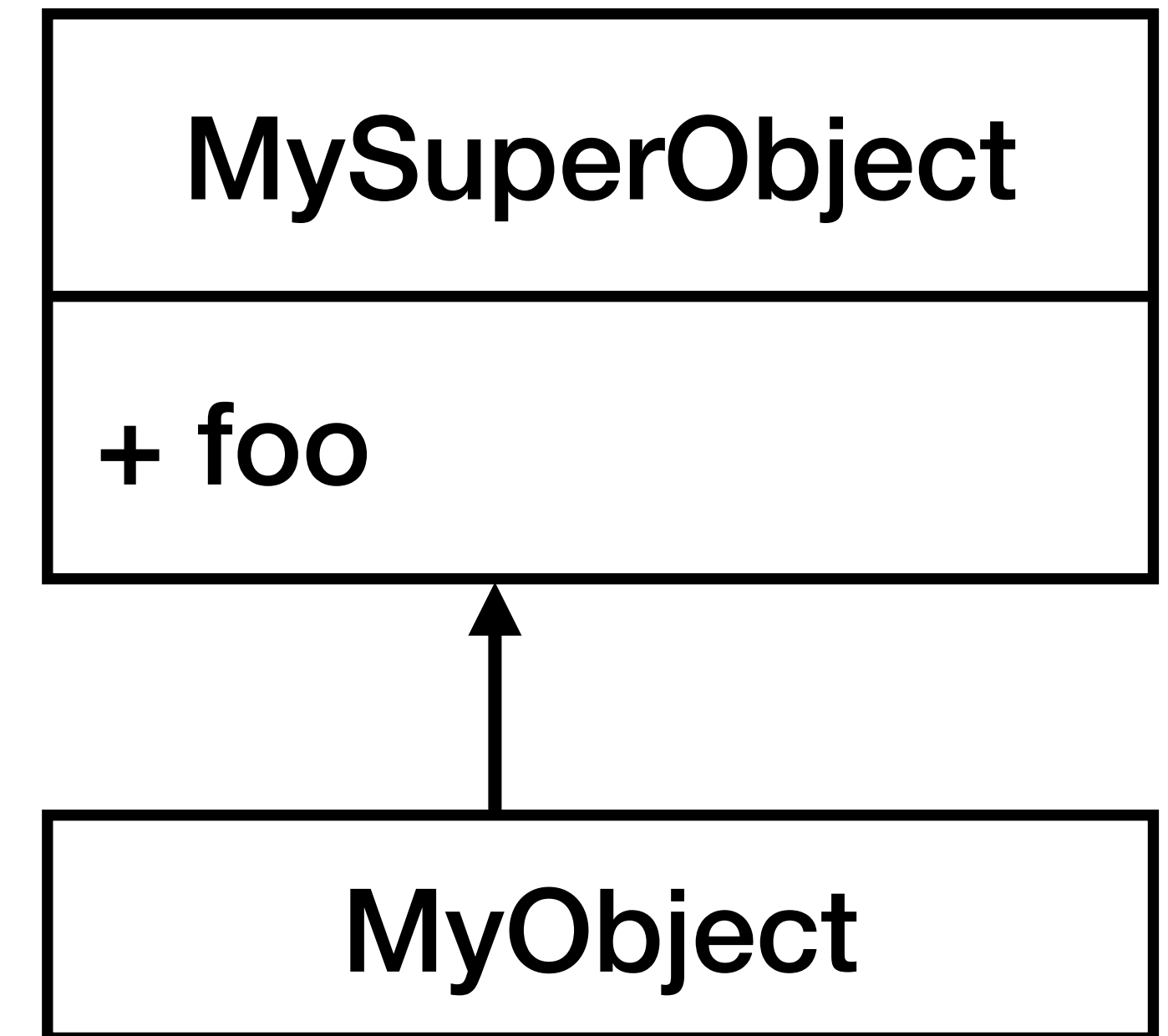
$o \text{ foo}$

Static Calls

- Specify the exact method to execute directly in the call site at compile time
- No late-binding/lookup

New syntax or static message:

Receiver _classname_selector



`o := MyObject new`

`o _MySuperObject_foo`

Why Static Calls ?

Empower developers:

- Basic Building block for advanced mechanisms
 - ➡ Image and VM side communication (primitives)
- Direct method call without late binding
 - ✓ More efficient
 - ★ No Guard => errors may occur

Static Calls

- New syntax (IR and AST node with type information)
- Two-byte instruction:
 1. Static Call opcode (246)
 2. Literal offset of the target method

The screenshot displays a debugger interface with two windows. The left window, titled 'a CompiledMethod (Undefined...', shows the 'Method Source' tab with the following code:

```
1 foo
2 ^ 13 _SmallInteger_myFactorial
```

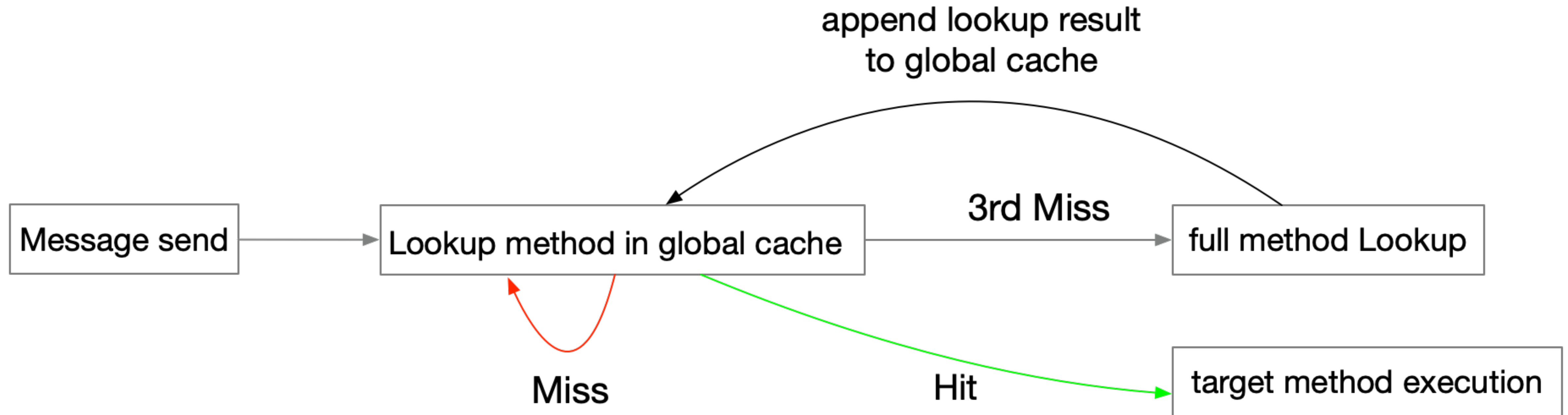
The right window, also titled 'a CompiledMethod (Undefined...', shows the 'Raw' tab with a table of variables and their values:

Variable	Value
{ } self	UndefinedObject>>#foo
Σ literal0	-1152921504606846972
Σ literal1	13
▶ { } literal2	SmallInteger>>#myFactorial
▶ © literal3	an AdditionalMethodState (74938
▶ © literal4	#UndefinedObject->UndefinedOb
Σ bc 41	32
Σ bc 42	246
Σ bc 43	1
Σ bc 44	92

A red arrow points from the '246' value in the 'bc 42' row to the '13' in the source code, indicating the offset of the target method. The 'SmallInteger>>#myFactorial' value in the 'literal2' row is also highlighted with a red box.

Static Calls: Interpreter implementation

Overview of message sends at runtime



Message Sending Implementation (Interpreter)

```
{1 . 1.0 . 10000000000000000000000000. 1/4}
```

```
do: [ :num | num negated ]
```

Message Sending Implementation (Interpreter)

```
{1 . 1.0 . 10000000000000000000000000. 1/4}
```

```
do: [ :num | num negated ]
```



Global cache

Message Sending Implementation (Interpreter)

{1 . 1.0 . 10000000000000000000000000. 1/4}

do: [:num | num negated]



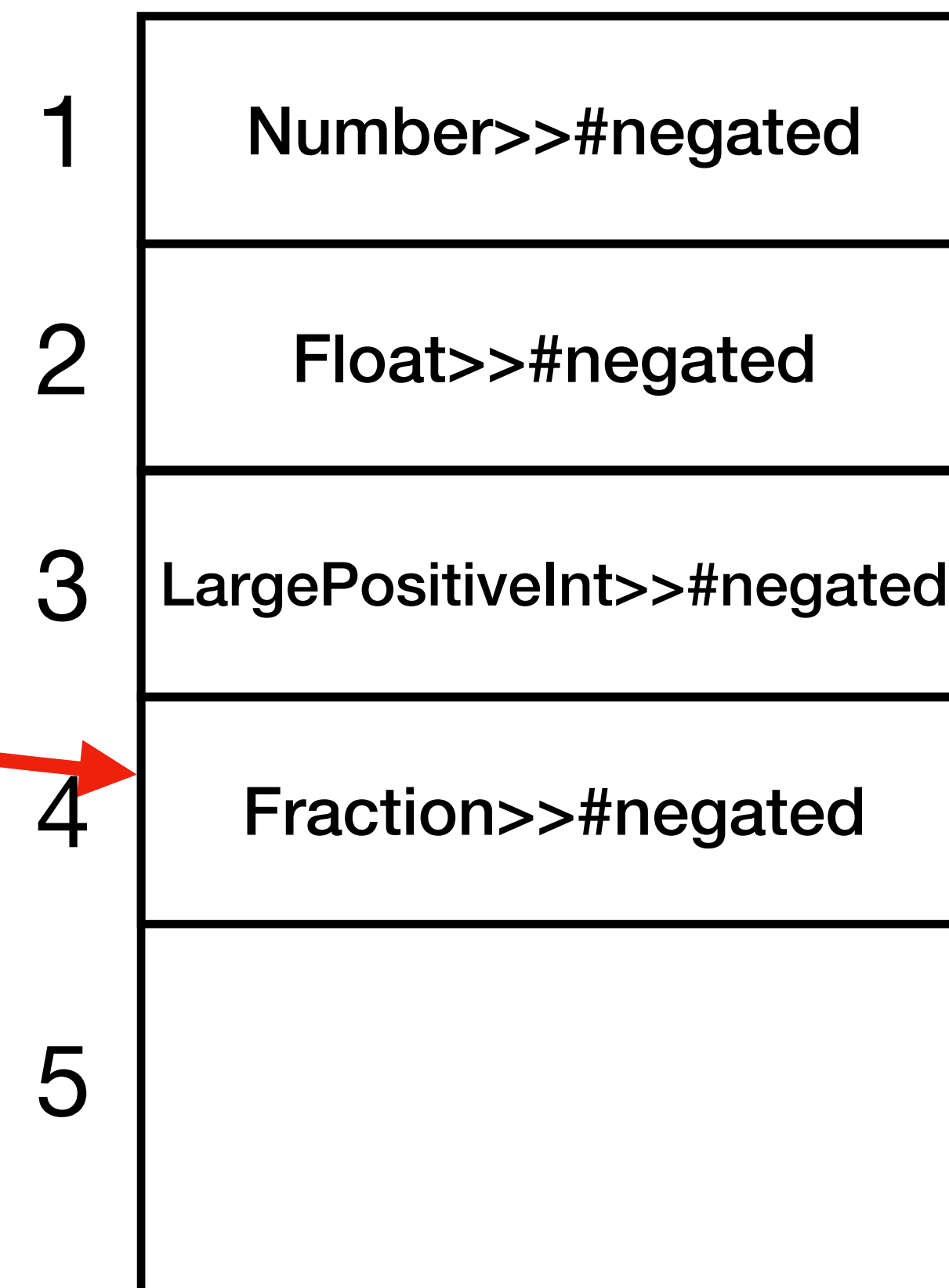
1	Number>>#negated
2	Float>>#negated
3	LargePositiveInt>>#negated
4	Fraction>>#negated
5	

Global cache

Message Sending Implementation (Interpreter)

[illegible]

```
do: [ :num | num negated ]
```



Global cache



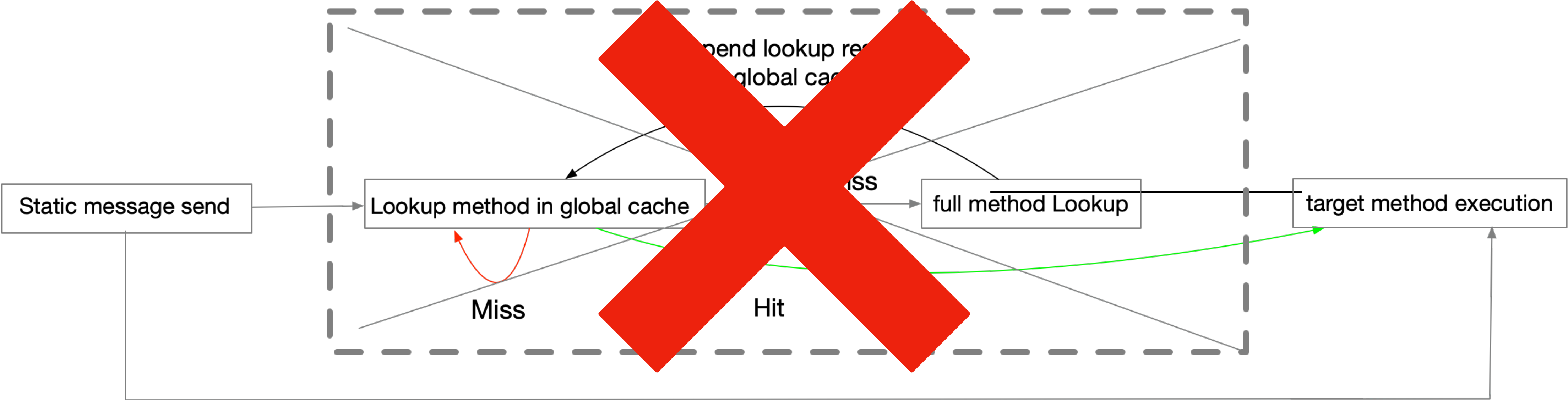
Hash collisions are possible

Static Calls Interpreter Implementation

In a Nutshell

- Doesn't pass through the global cache nor the lookup
- Fetches the method from the literal frame and activates it directly

Extending Interpreter to support Static Calls



Static Calls: JIT-based implementation

Current Message Sending Jit

$\{1 \cdot 1.0 \cdot 10000000000000000000000. 1/4\}$

```
do: [ :num | num negated ]
```

```
num = 1
```

```
num = 1.0
```

```
num = 10000000000000000 ...
```

■ ■ ■

Unlinked site

Monomorphic site

Polymorphic site

Megamorphic site

rewritten as

rewritten as

rewritten as

A >>

arg

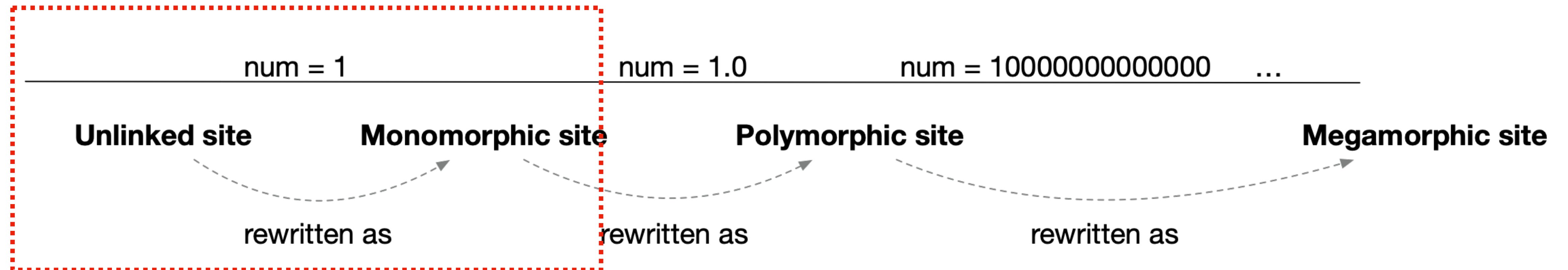
com

Current Message Sending

Jit

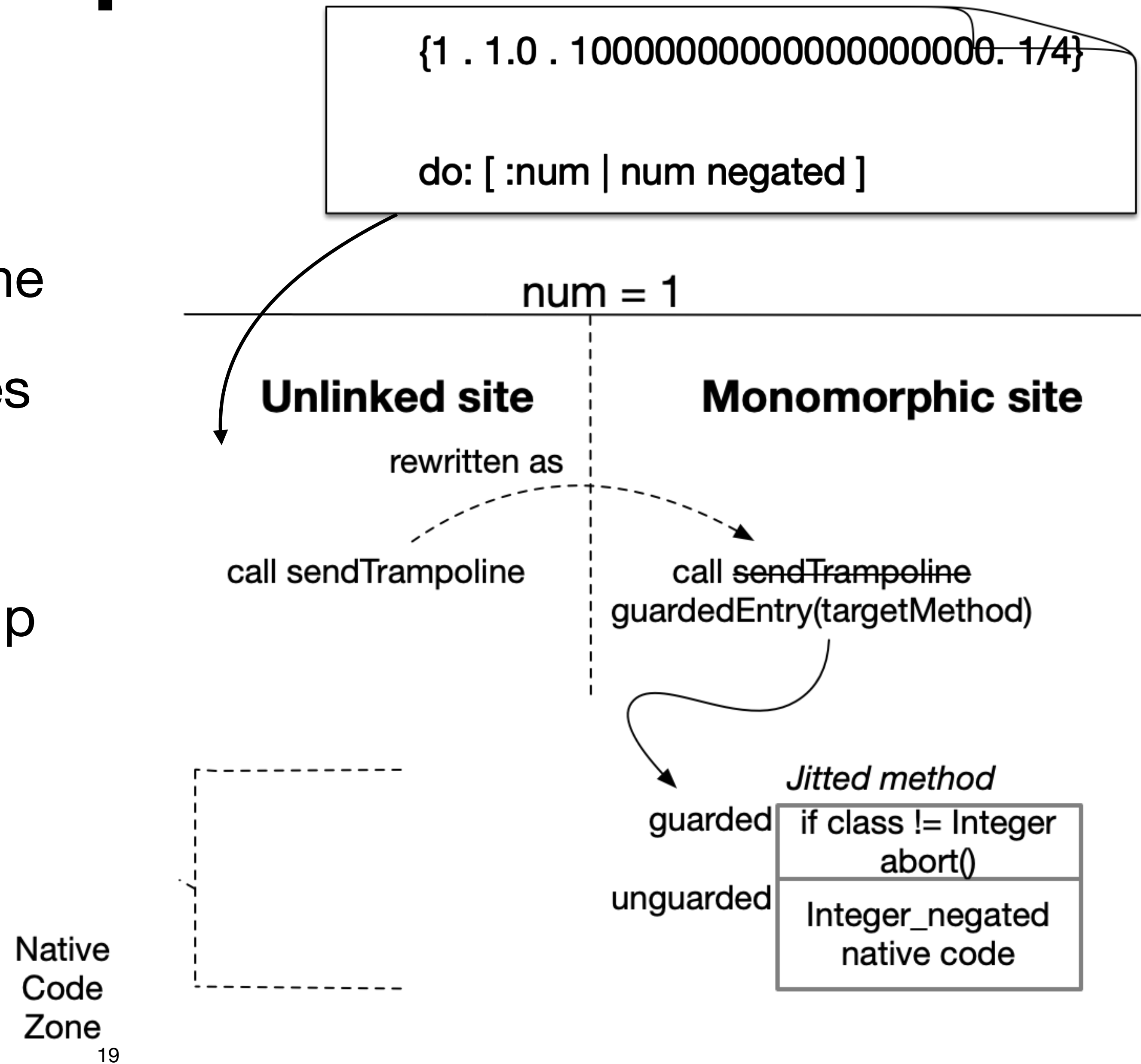
$$\{1 \cdot 1.0 \cdot 100000000000000000000. 1/4\}$$

```
do: [ :num | num negated ]
```



Unlinked to Monomorphic

- First call is managed by Trampoline
- Runtime performs lookup, Patches the site to the right target
- Trampolines are used for different functions, like patching and lookup

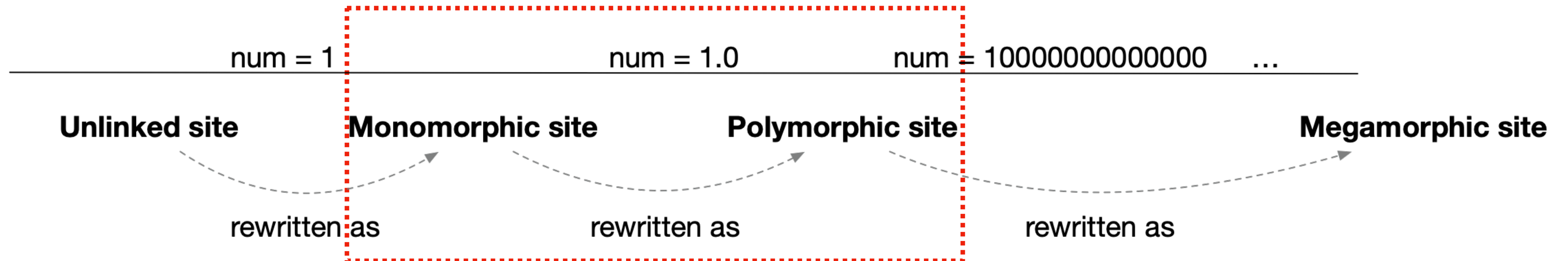


Current Message Sending

Jit

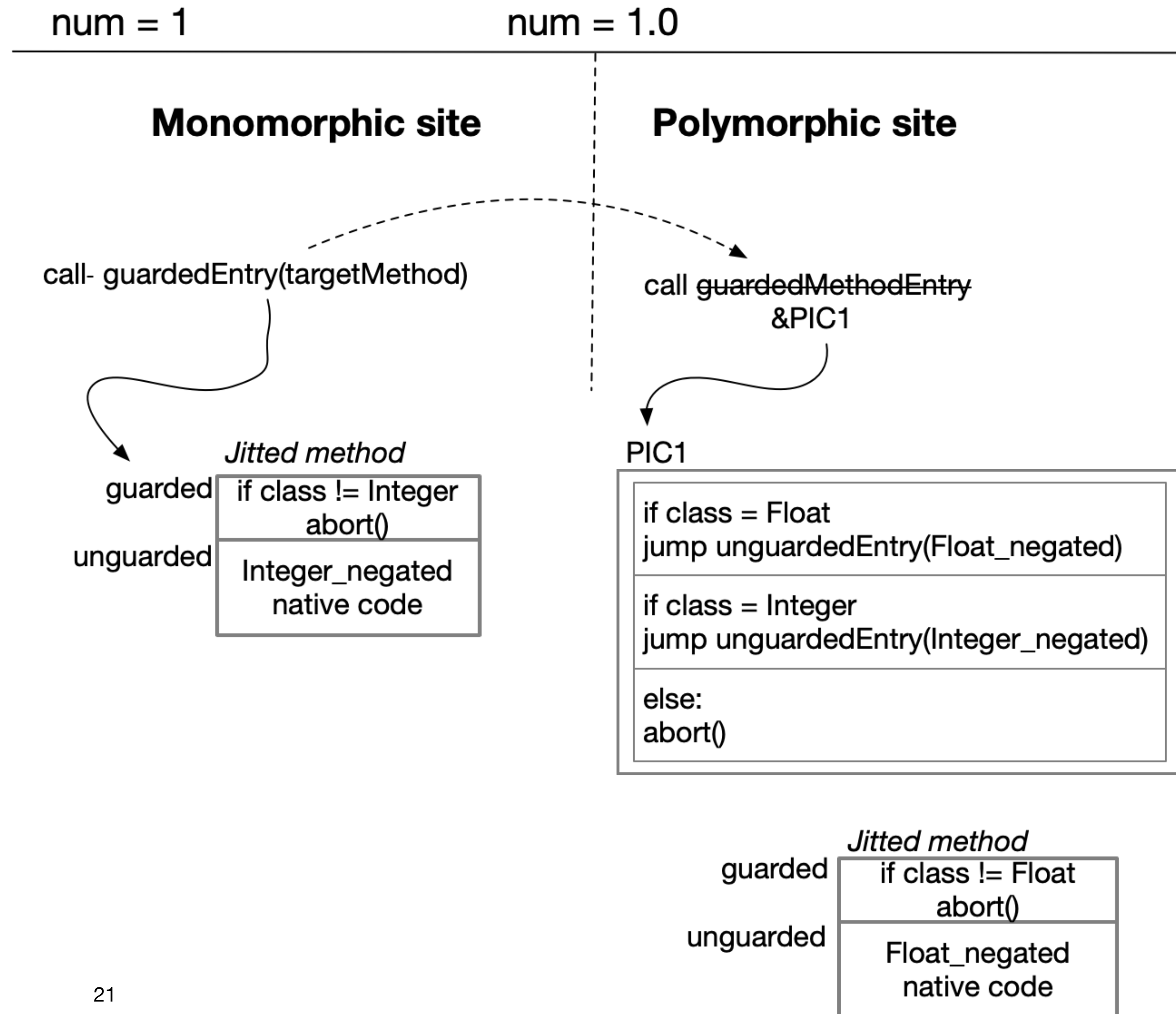
$$\{1 \cdot 1.0 \cdot 100000000000000000000. 1/4\}$$

```
do: [ :num | num negated ]
```



Monomorphic to Polymorphic

- Monomorphic call jumps into method check entry offset
- Type fail calls Trampoline
- Creates a PIC with 2 entries
- Patch call site with new PIC



Current Message Sending

Jit

A >> method1: arg

arg foo

compiled into

A new method1: 1

A new method1: 1.0

A new method1: Object new

Unlinked site

Monomorphic site

Polymorphic site

Megamorphic site

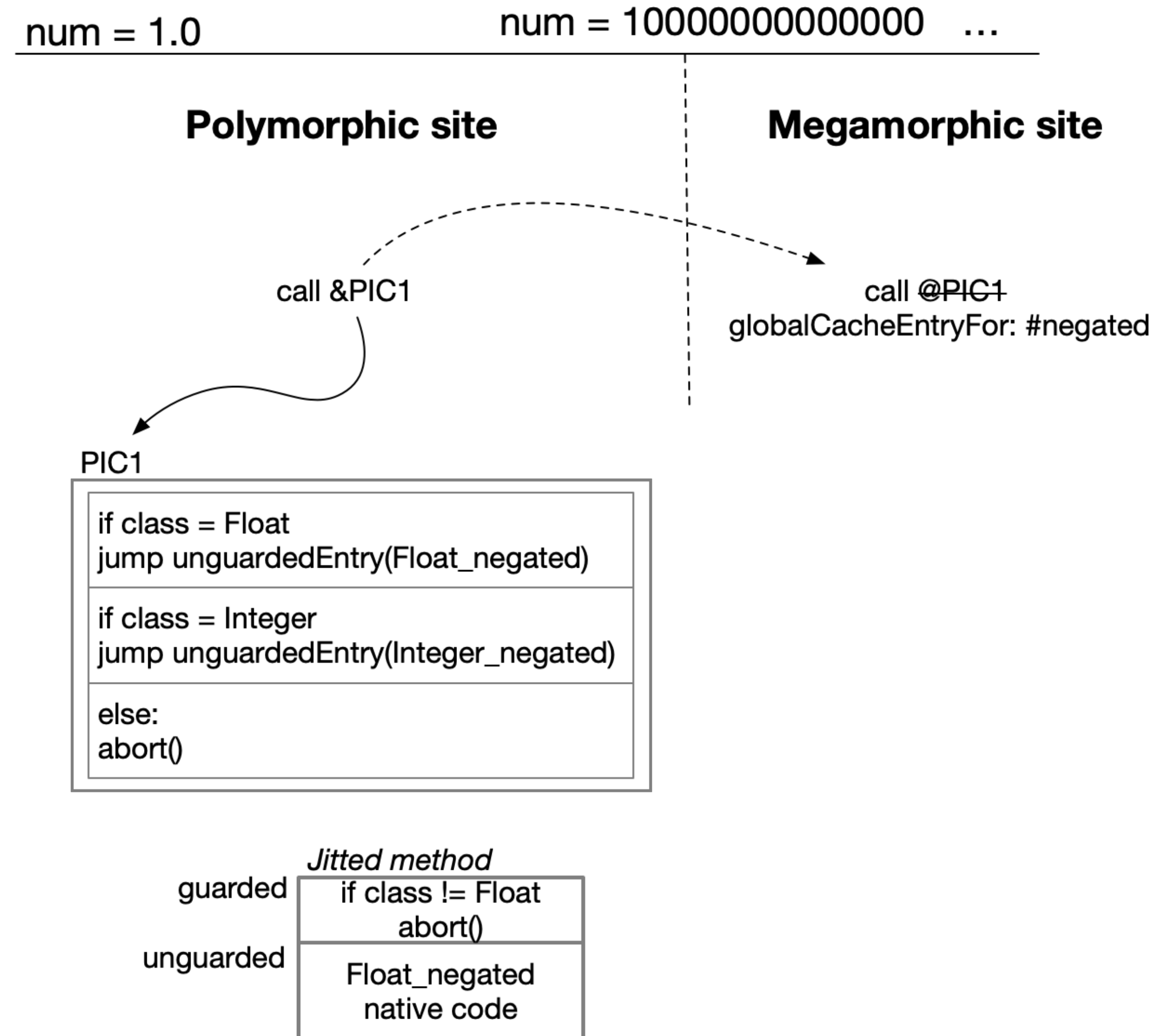
rewritten as

rewritten as

rewritten as

Polymorphic to Megamorphic

- PIC reaches 6 entries
- Trampoline patches the site to call a global cache lookup function
- Shared by all sites with the same selector

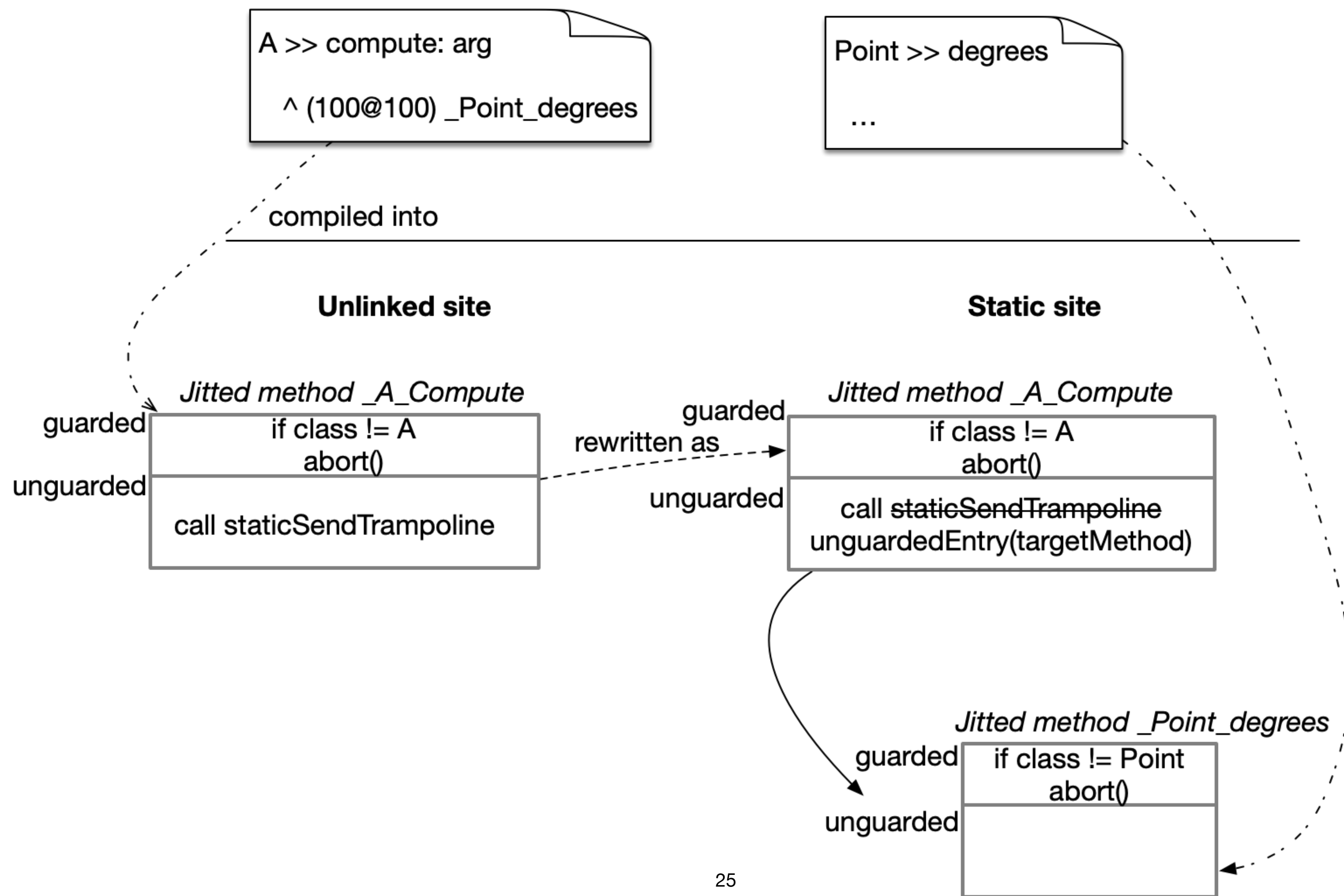


Static Calls JIT-based Implementation

In a Nutshell

- 4 new trampolines
 - Jump to jitted code (ceStaticMessageSend:)
 - Precomputes the handling of arguments.
- A new instruction to generate assembly for: fetching the target method to be executed from literals
- Call inverse trampoline when unable to locate/generate jitted target method

Extending JIT to support Static Calls

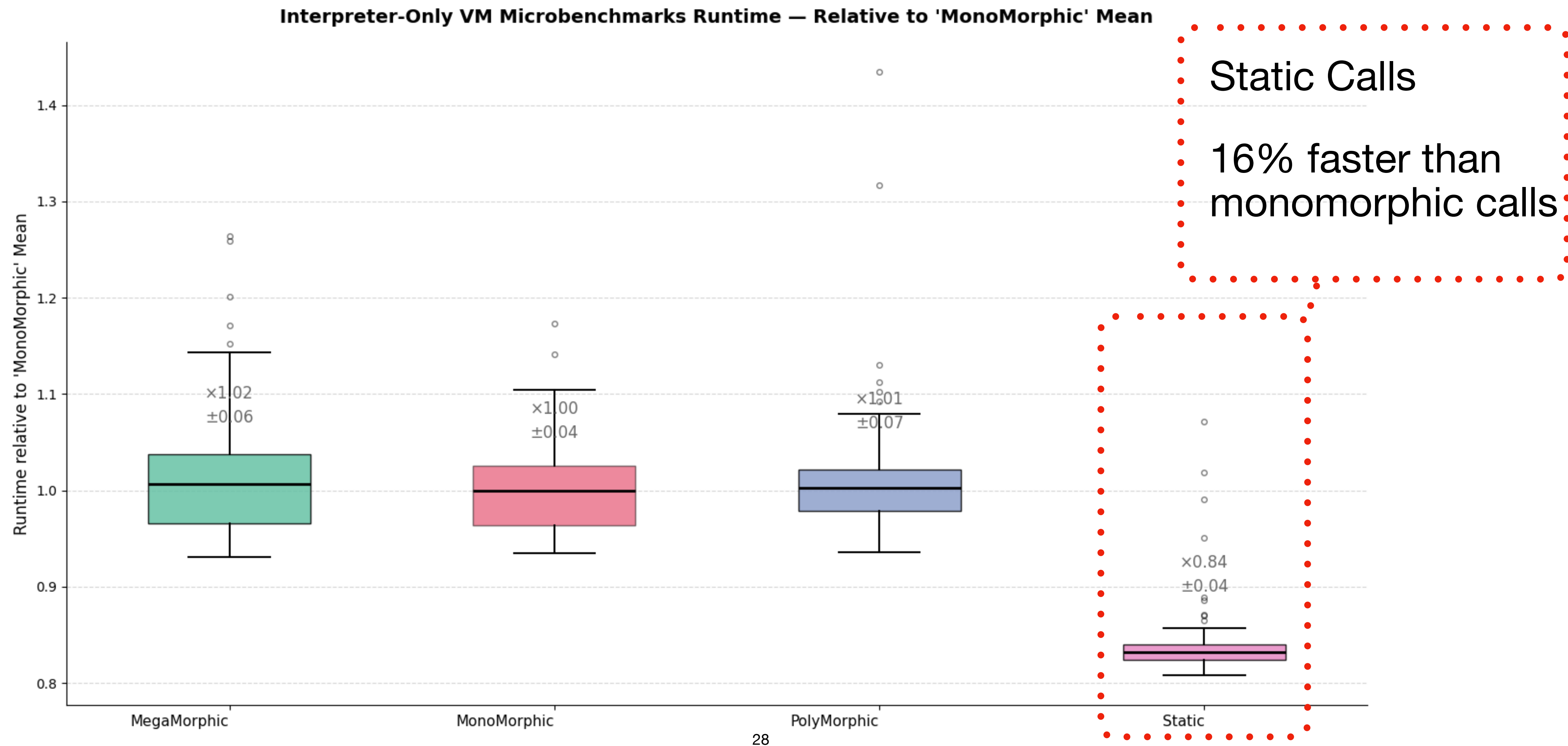


Static Calls: Evaluation

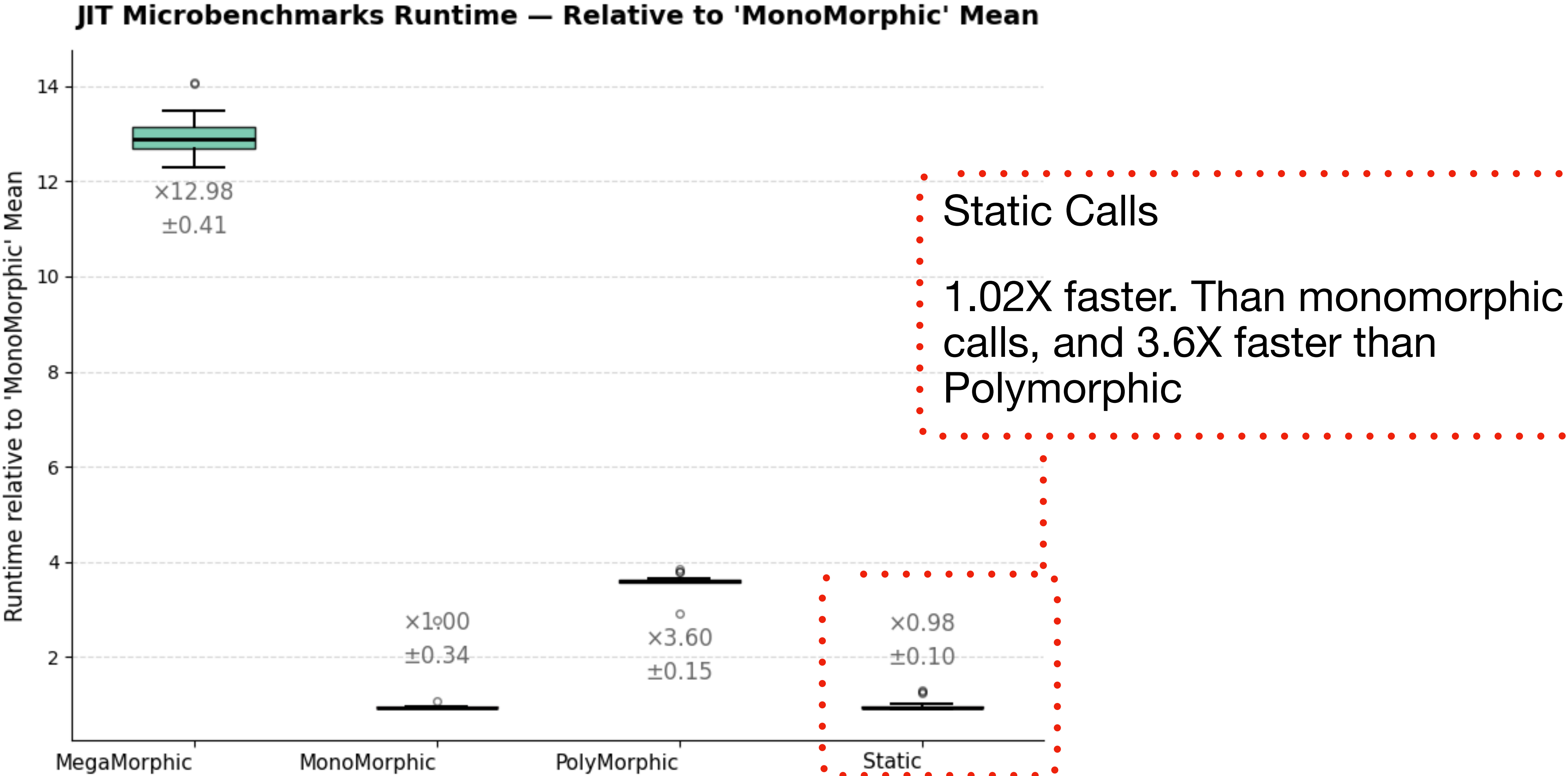
Evaluation: Setup

```
1 initializeCaches
2   3 timesRepeat: [
3     self
4       sending64TimeMessageTo: Set;
5       sending64TimeMessageTo: 1;
6       sending64TimeMessageTo: Object new;
7       sending64TimeMessageTo: 1.0;
8       sending64TimeMessageTo: 'toto';
9       sending64TimeMessageTo: Object ]
10
11 1 to: iterations do: [ :e | self sending64TimeMessageTo: Set ].
```

Evaluation: Interpreter



Evaluation: JIT



Conclusion

- New static calls implementation at the Interpreter and JIT compiler levels
- Static calls speed up the interpreter. But not as much in JIT due to already fast monomorphic inline caches
- A building block for future experiments
 - Breach image to VM access restriction, without generating new VMs

Static Calls in a Dynamically-typed Language: From Implementation Challenges to Opportunities

- Questions ?

IWST'26