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

IWST'26

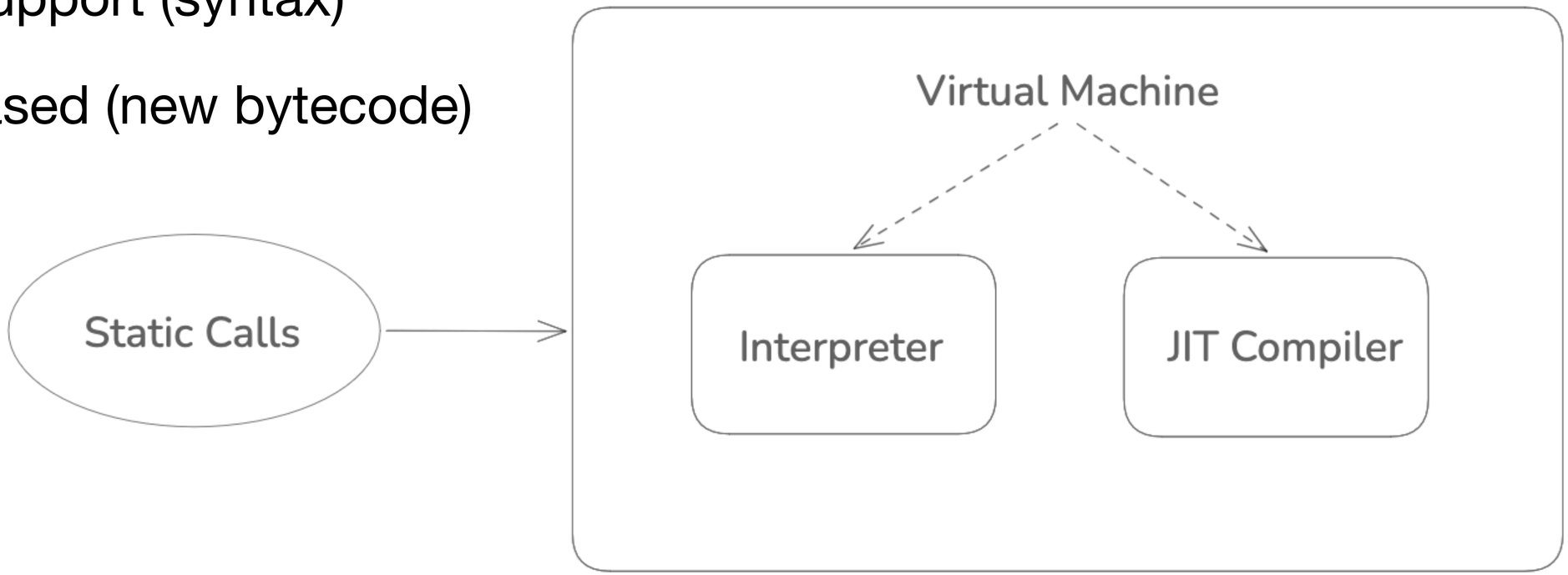
Faouzi Rayane Mokhefi, Stéphane Ducasse, Luc Fabresse, Pablo Tesone, Guillermo Polito - 18/06/2026

ESUG'26

International Workshop for Smalltalk Technologies 2026

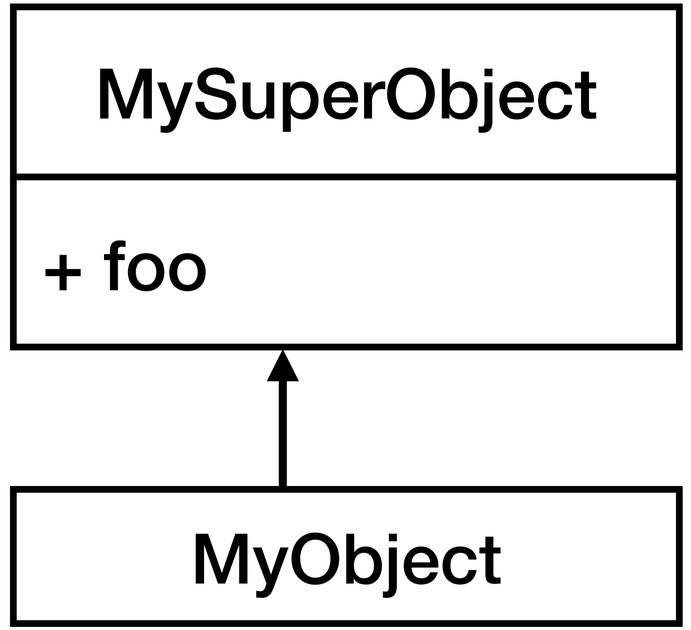
Contributions and Agenda

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



Context: Dynamically-typed Language

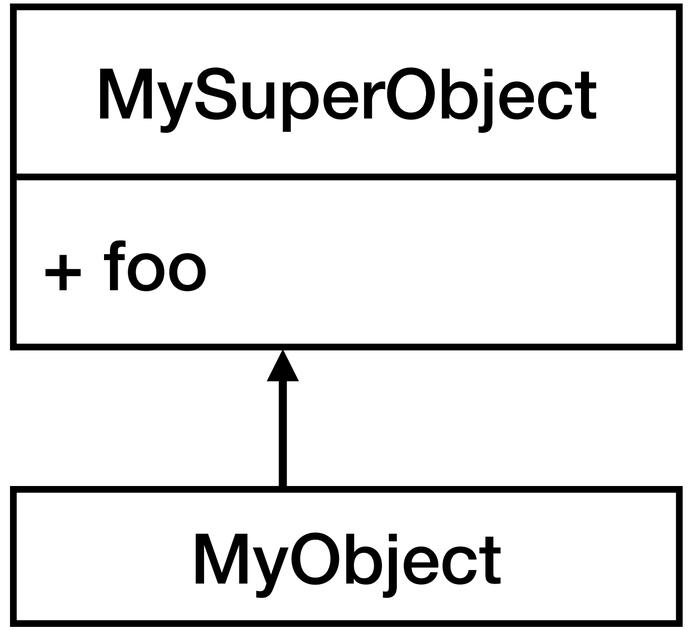
- Dynamic dispatch is **slow**
- Possible optimizations:
  - Caches (Global, inline)
  - Devirtualization




```
o := MyObject new

o foo
```

Context: Dynamically-typed Language

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

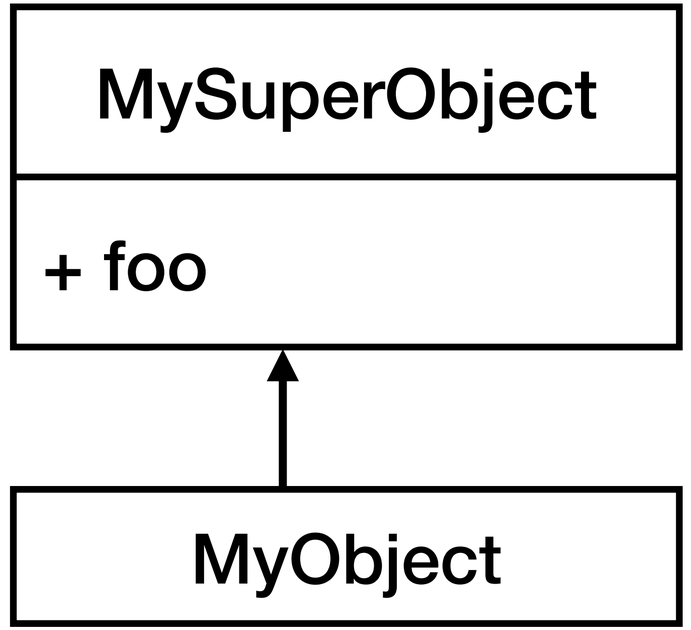



```
o := MyObject new
o foo
```

Call site

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

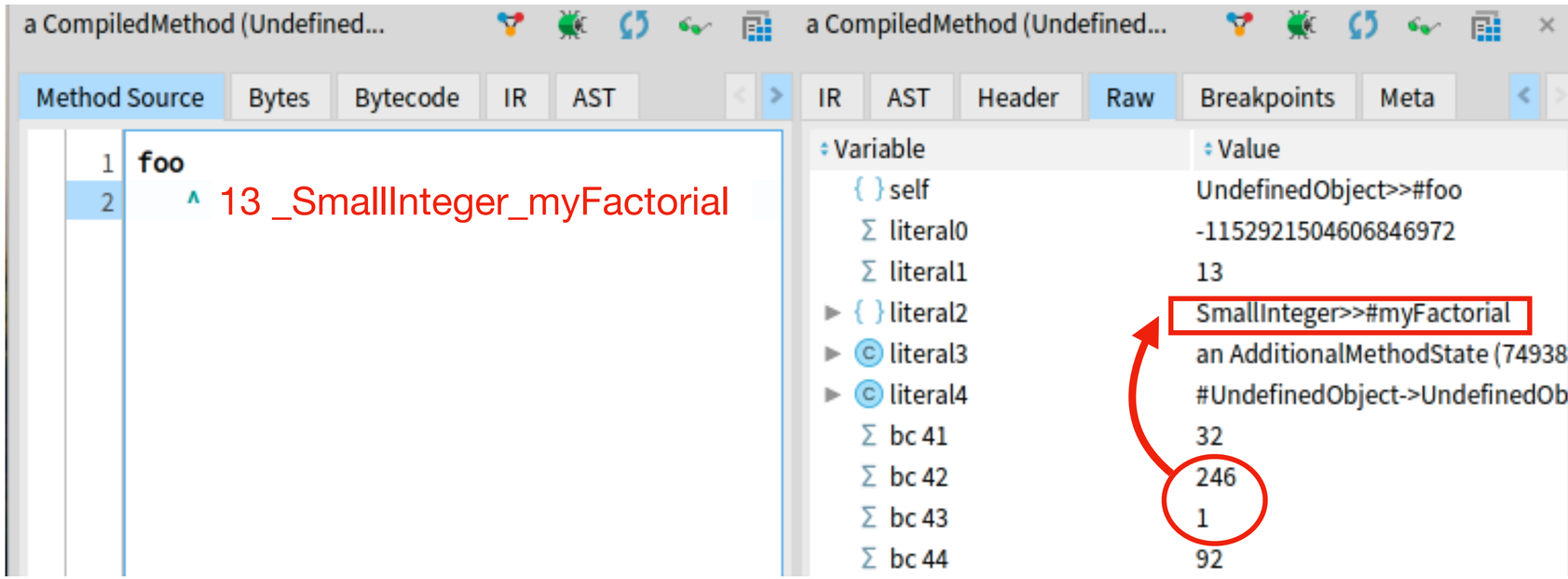
  ★ No Guard => errors may occur

Static Calls

- New syntax (IR and AST node with type information)
- Two-byte instruction:

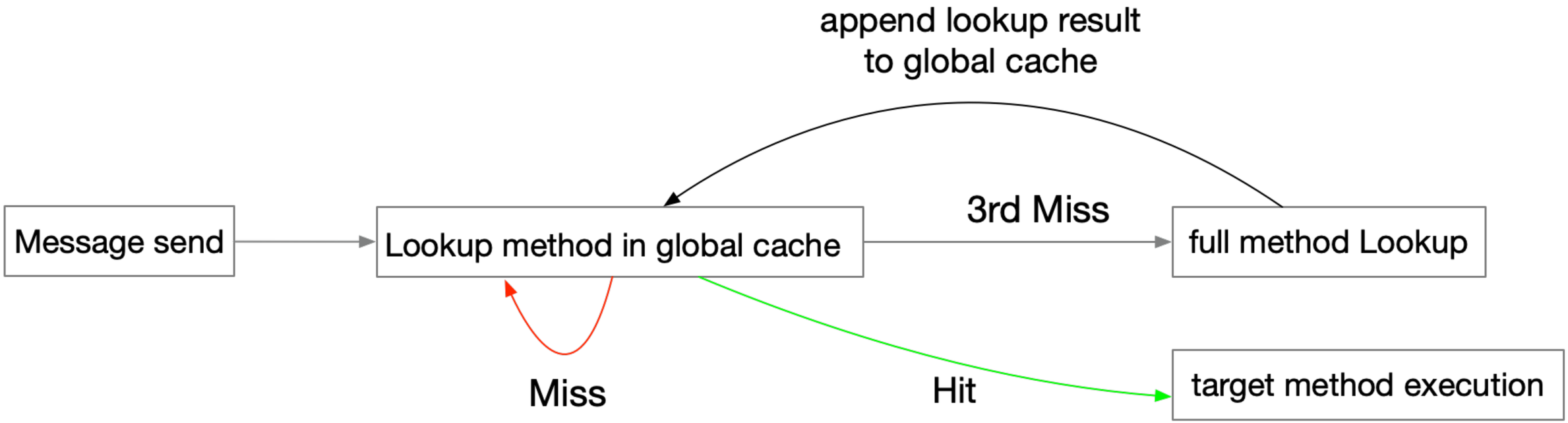
  1. Static Call opcode (246)

  2. Literal offset of the target method



Static Calls: Interpreter implementation

Overview of message sends at runtime



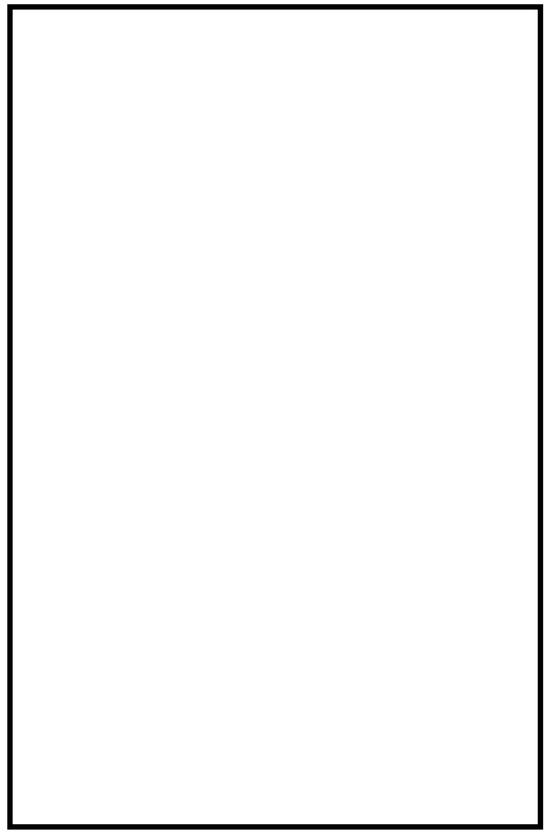
Message Sending Implementation (Interpreter)

```
{1 . 1.0 . 100000000000000000. 1/4}

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

Message Sending Implementation (Interpreter)

{1 . 1.0 . 100000000000000000. 1/4}

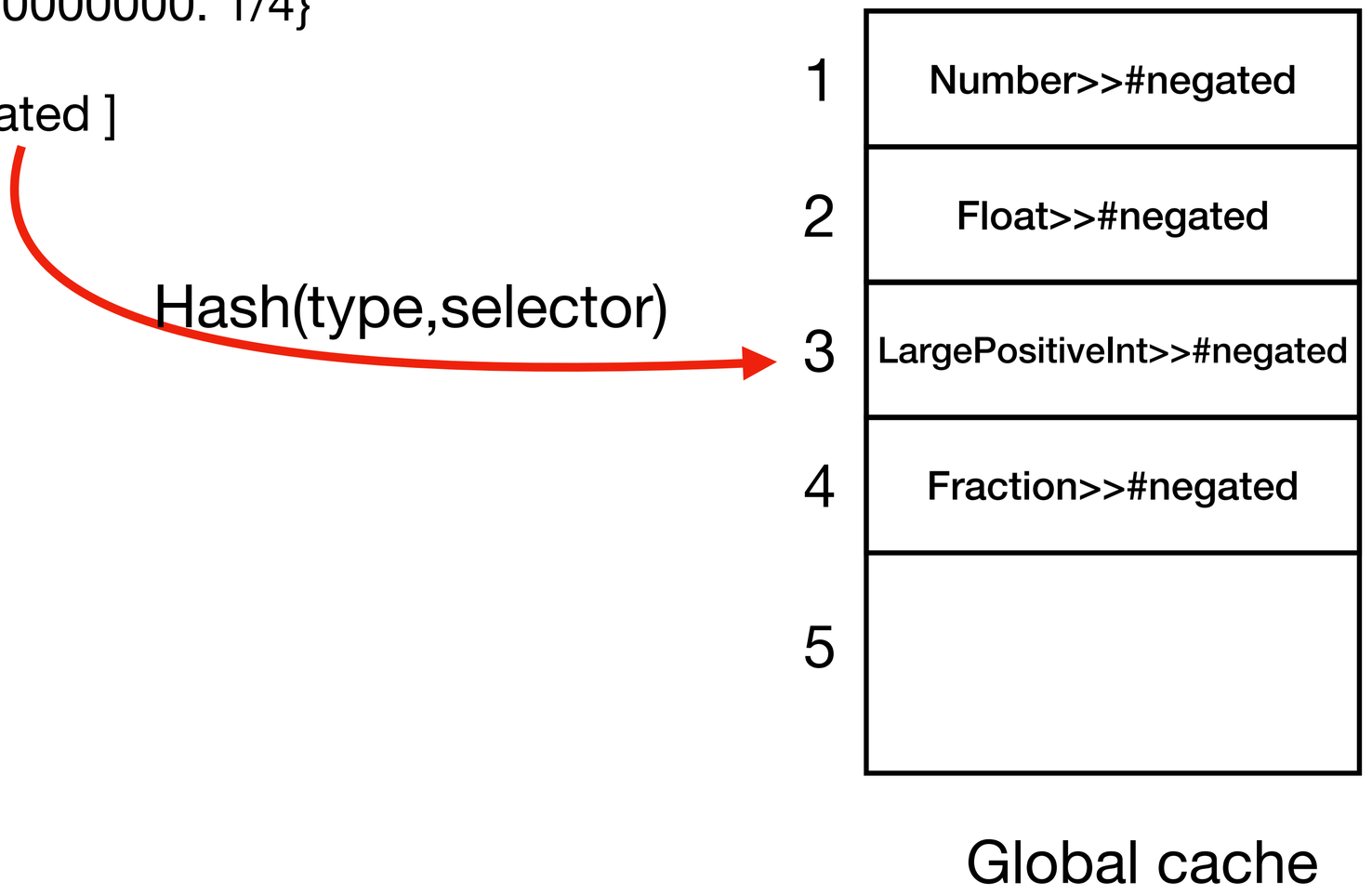
do: [:num | num negated]



Global cache

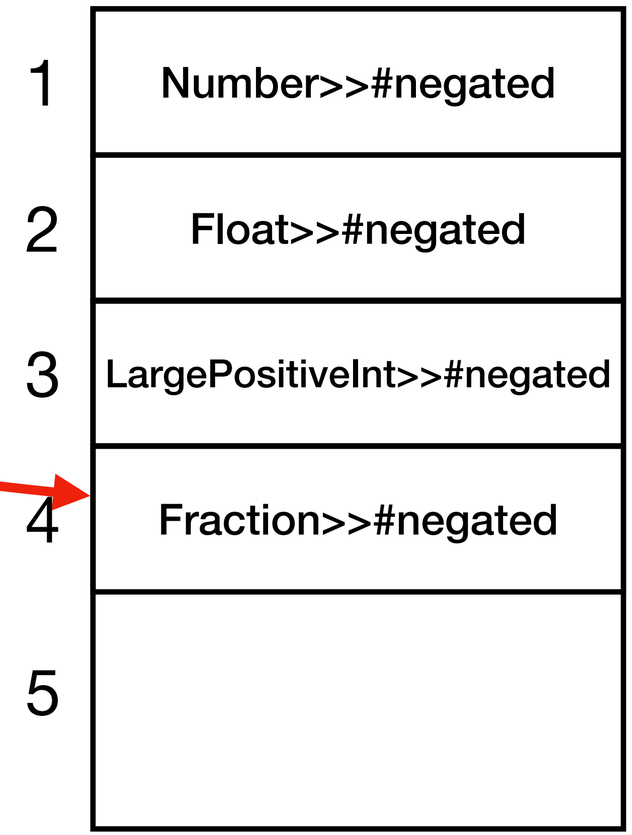
Message Sending Implementation (Interpreter)

```
{1 . 1.0 . 10000000000000000000. 1/4}

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



Message Sending Implementation (Interpreter)

```
{1 . 1.0 . 10000000000000000000. 1/4}

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



Global cache

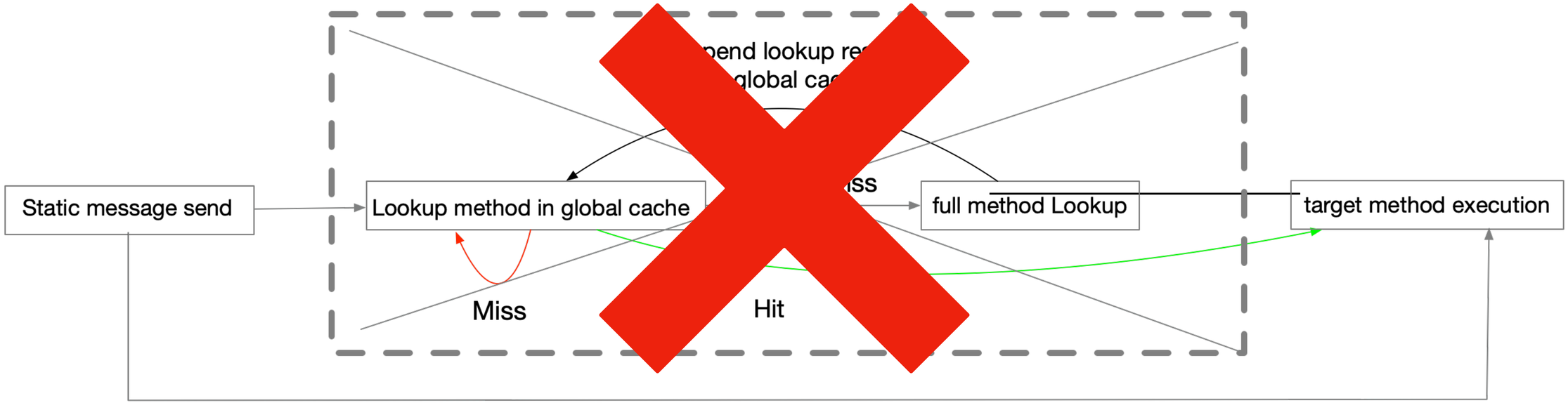
Hash collisions are possible

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

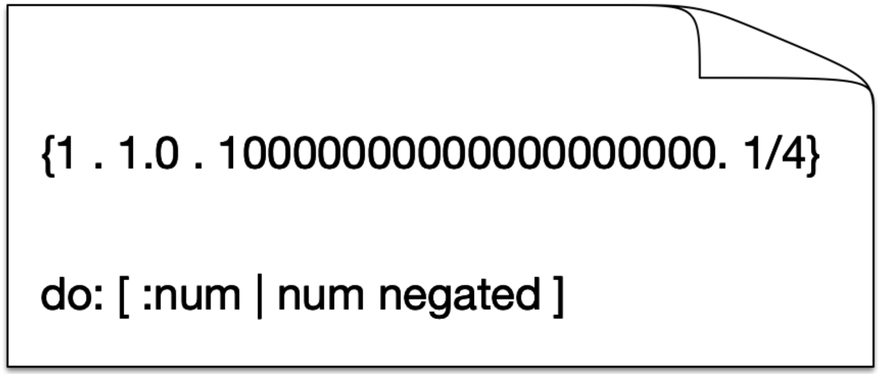
Extending Interpreter to support Static Calls

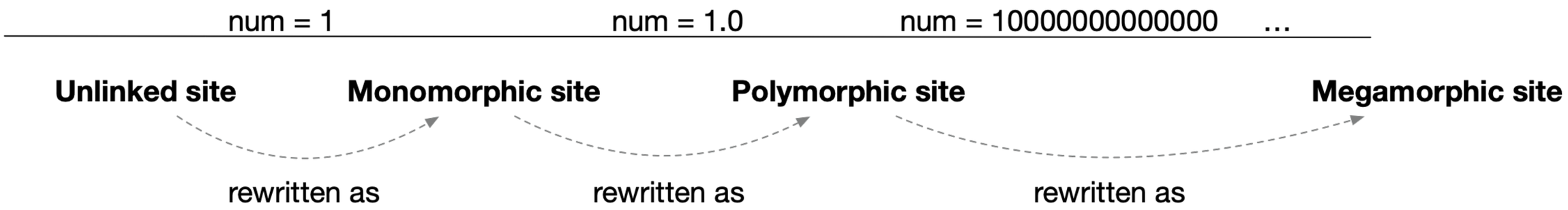


Static Calls: JIT-based implementation

Current Message Sending

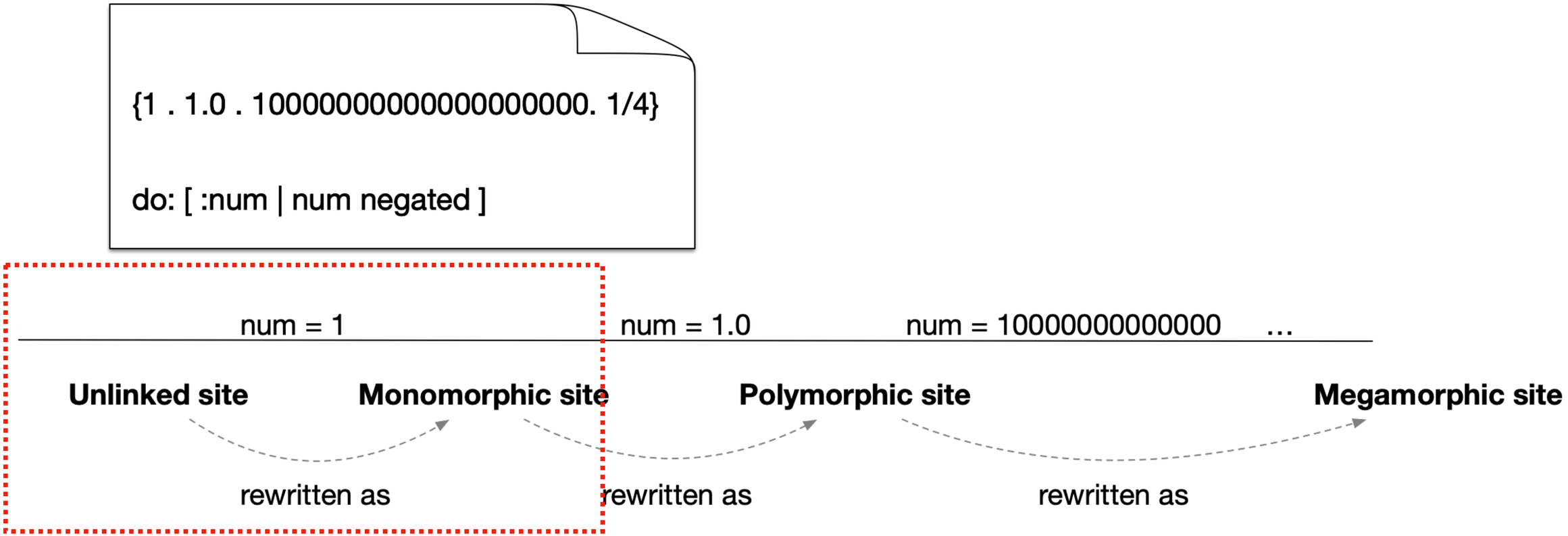
Jit





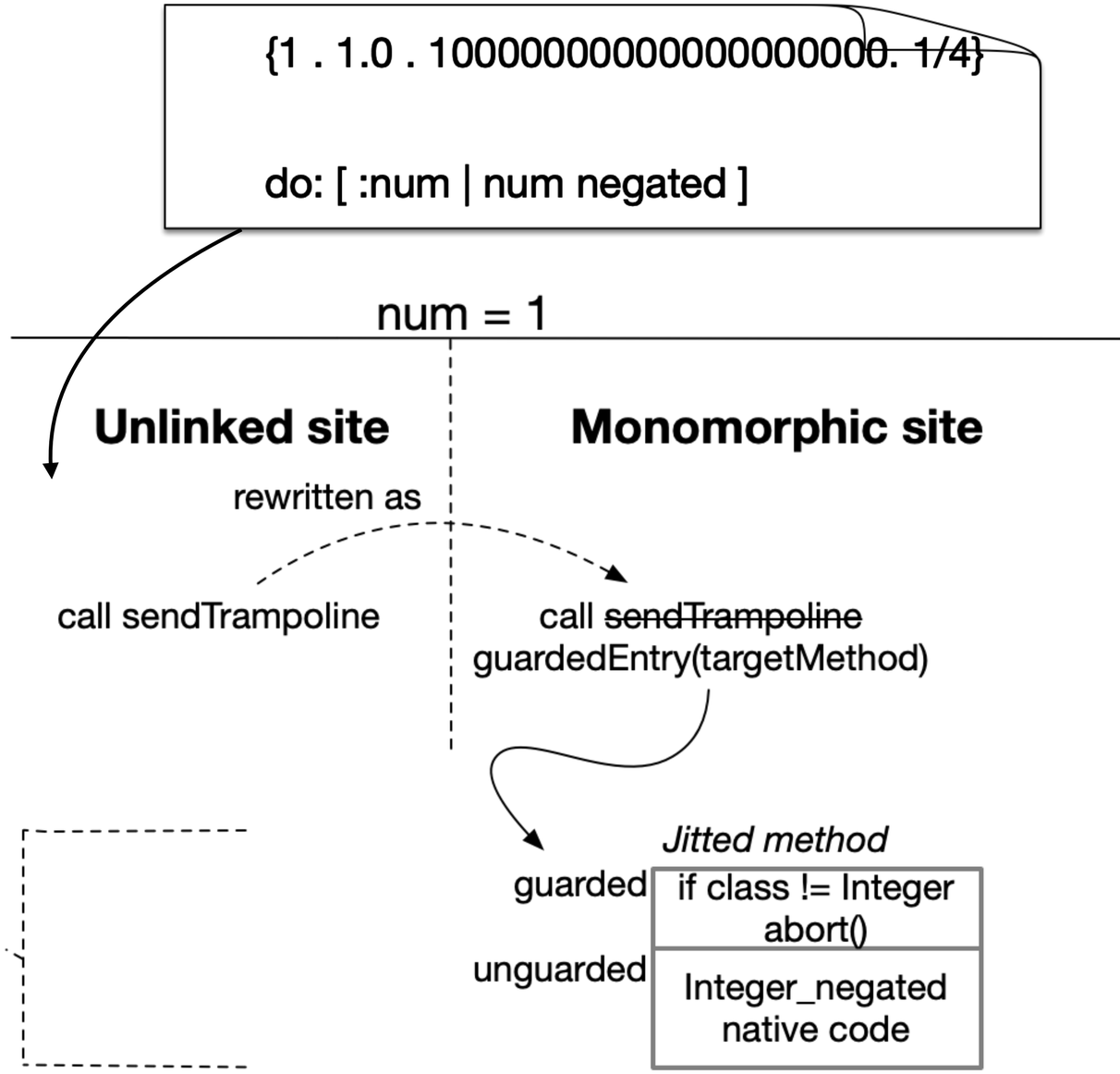
Current Message Sending

Jit



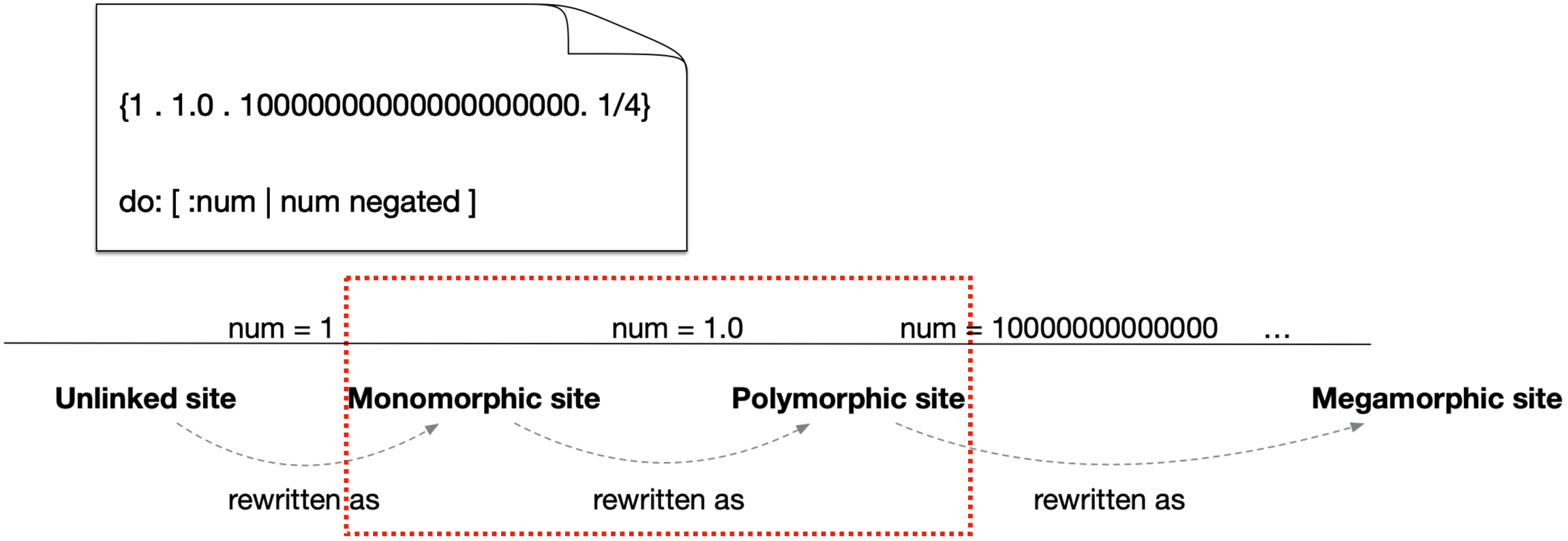
Unlinked to Monomorphic

- 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



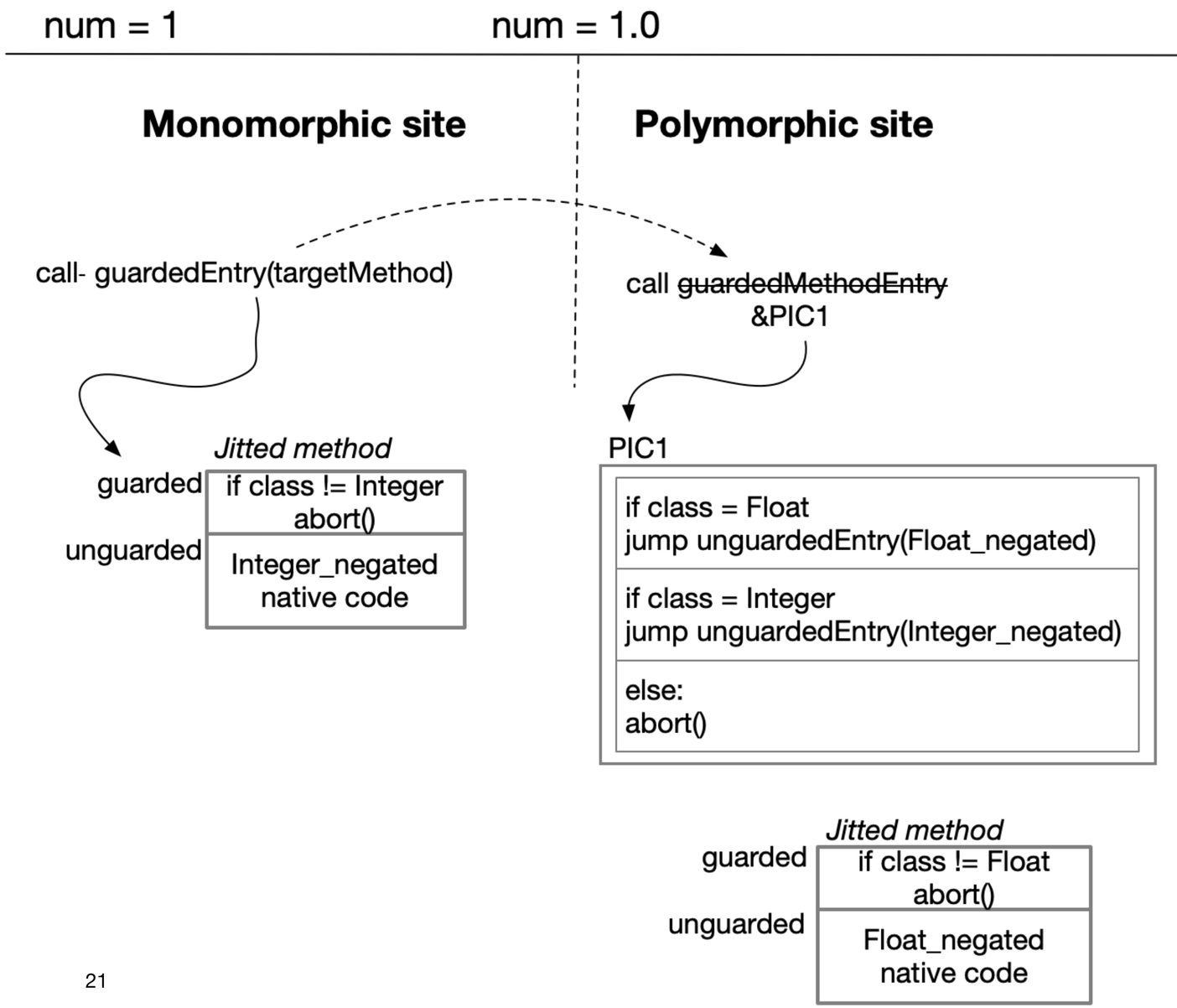
Current Message Sending

Jit



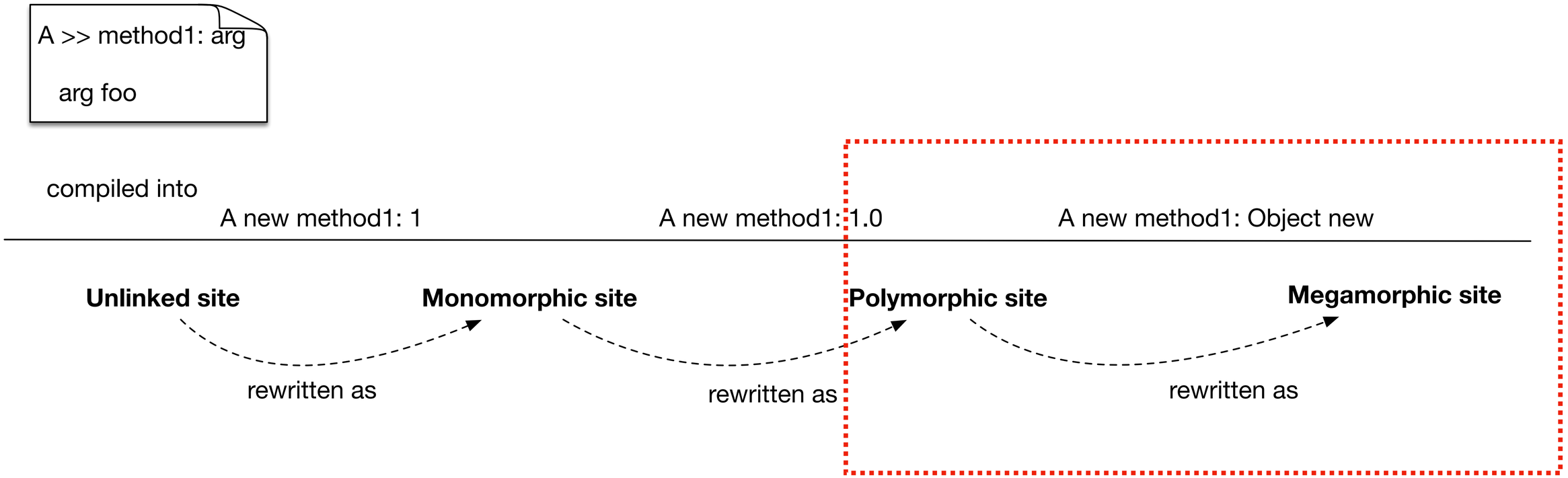
Monomorphic to Polymorphic

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



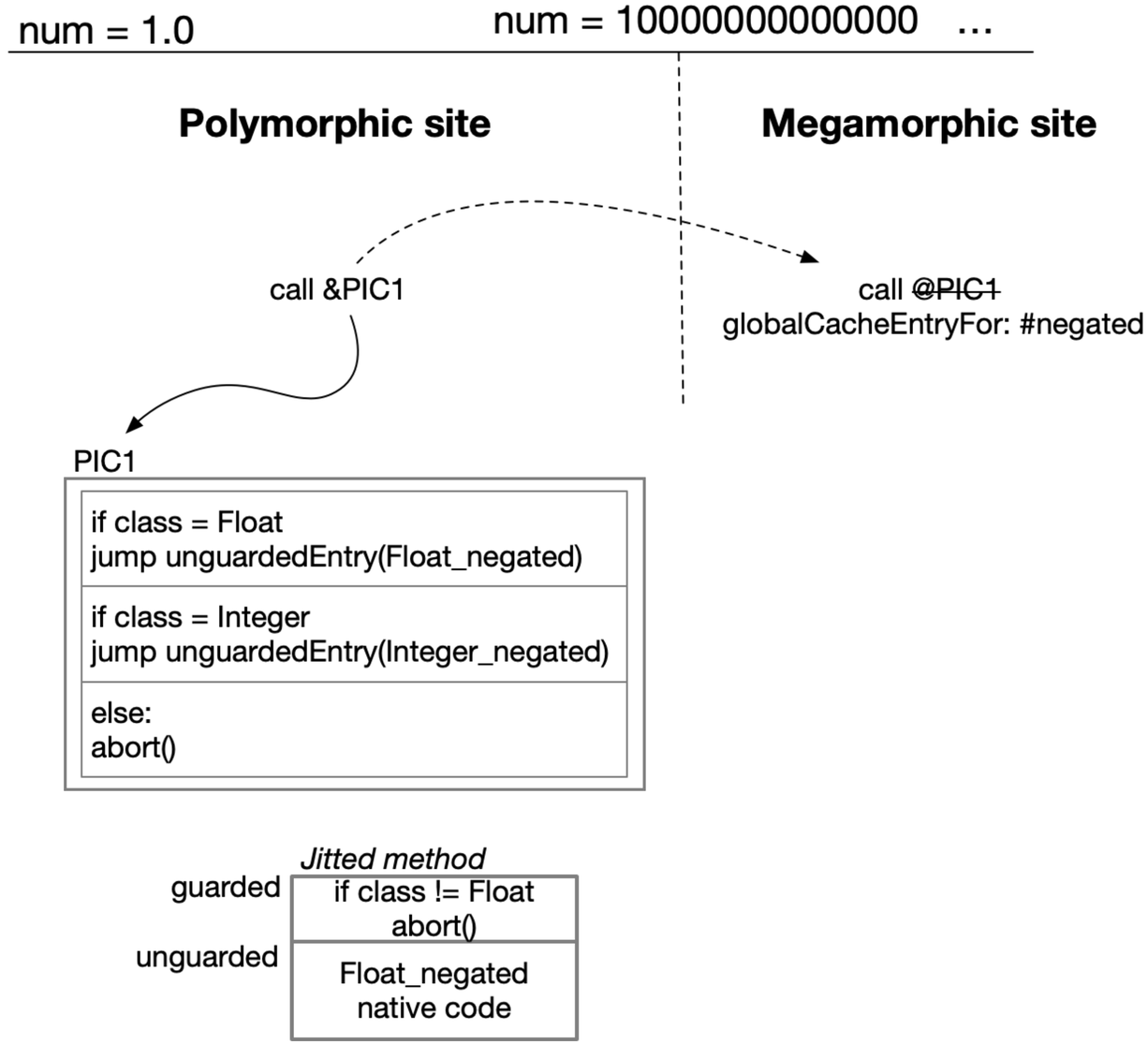
Current Message Sending

Jit



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

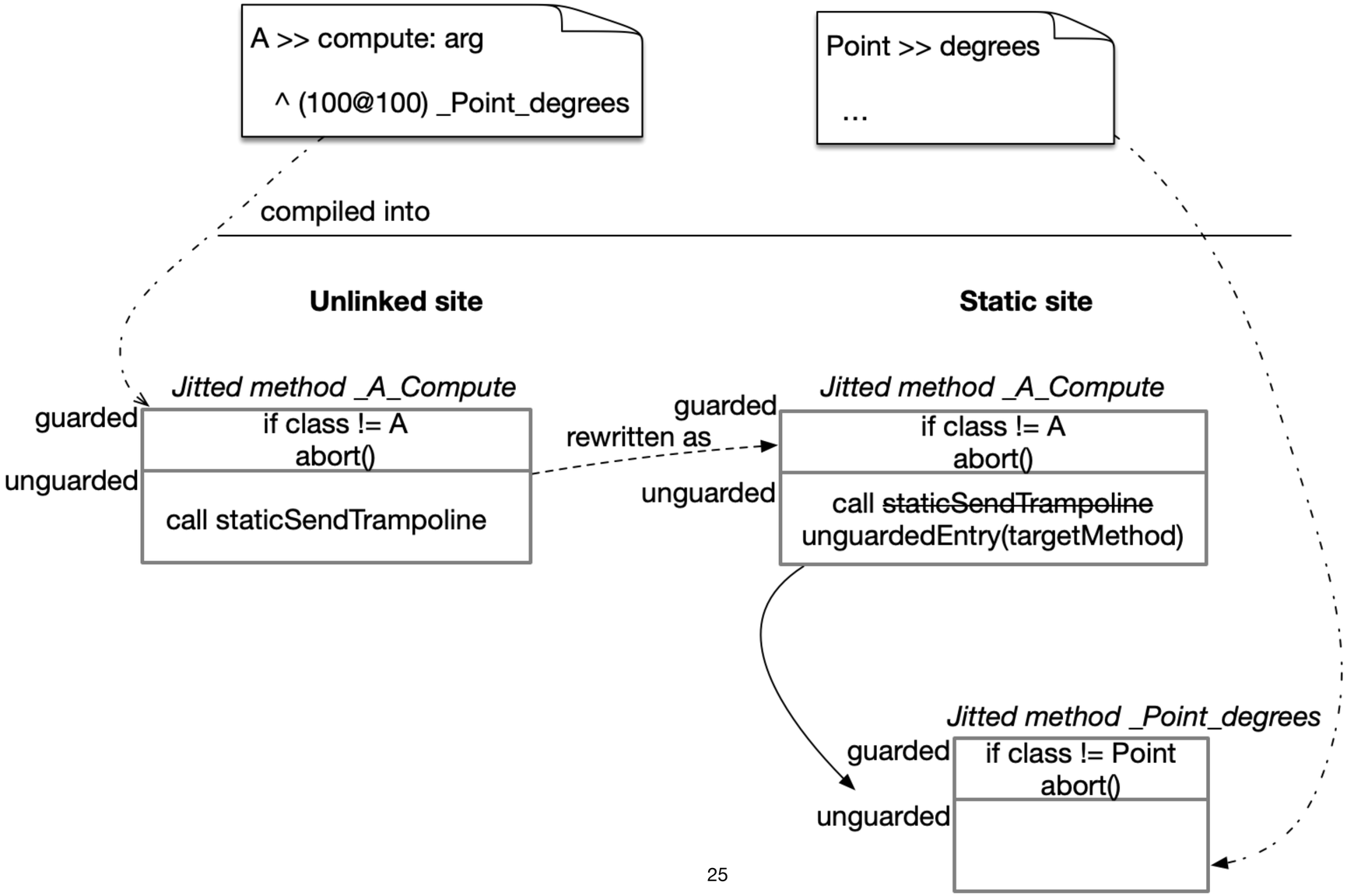


Static Calls JIT-based Implementation

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

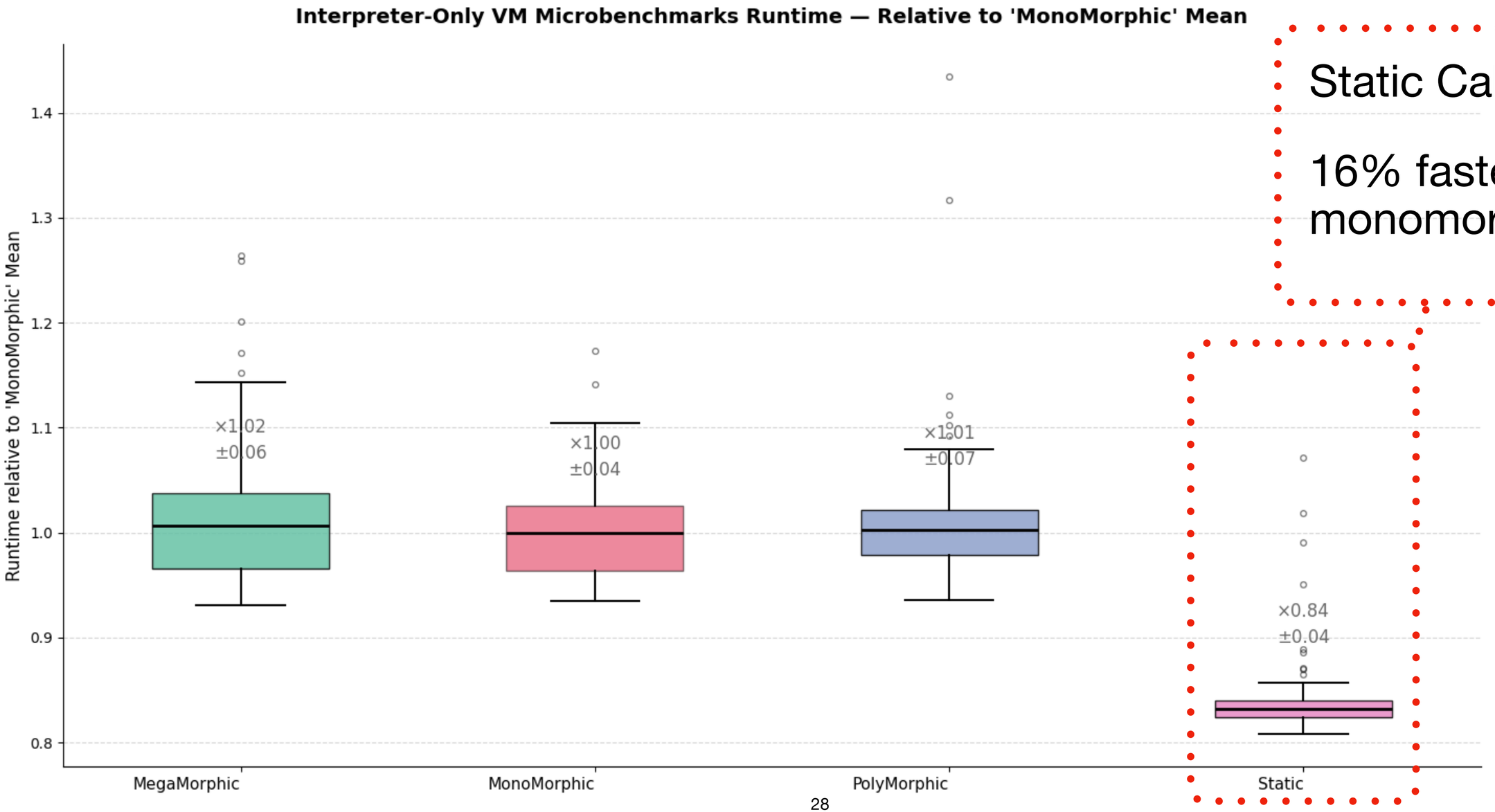


Static Calls: Evaluation

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 ].
```

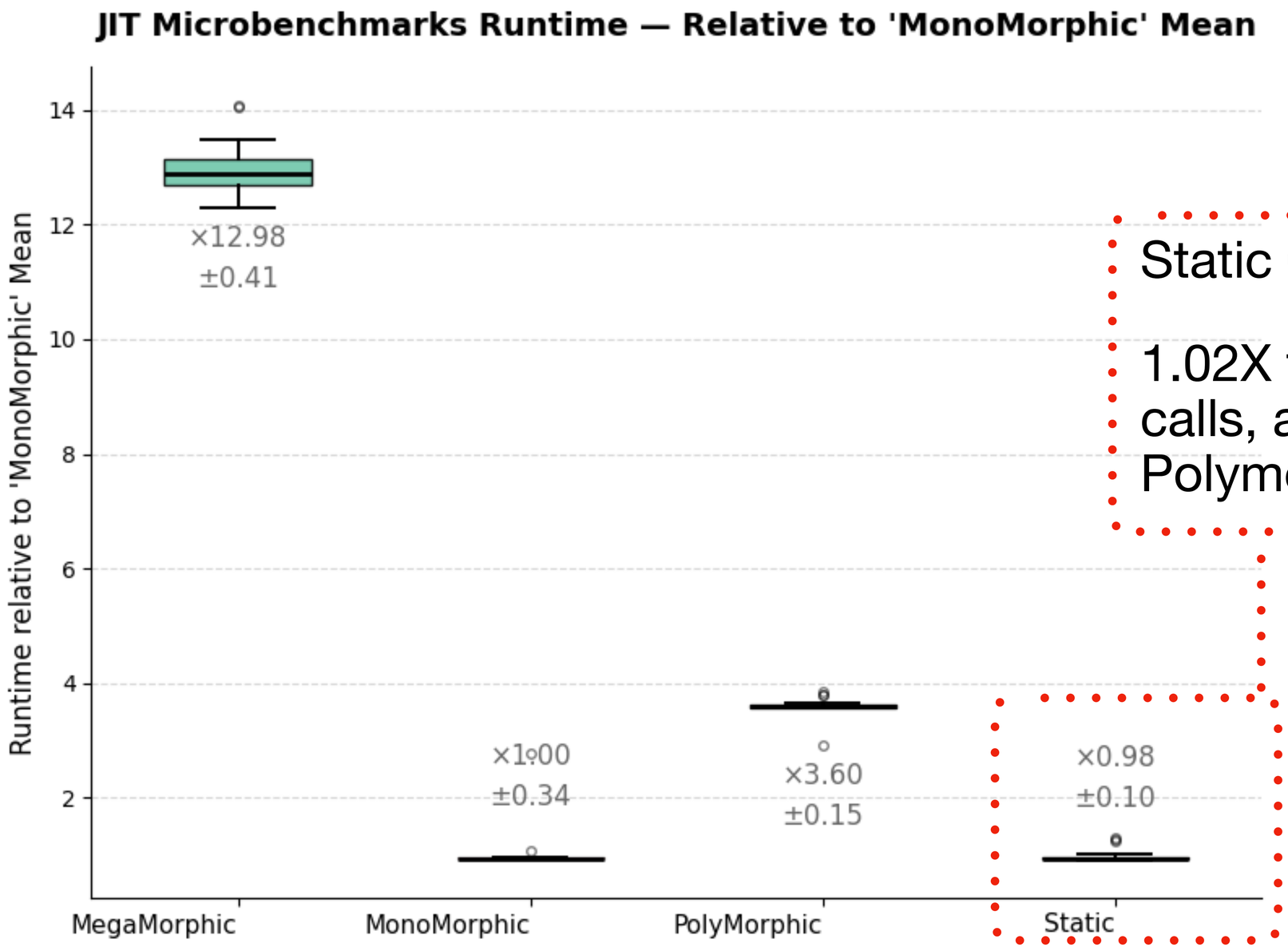
Evaluation: Interpreter



Static Calls

16% faster than monomorphic calls

Evaluation: JIT



Static Calls

1.02X faster. Than monomorphic calls, and 3.6X faster than Polymorphic

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