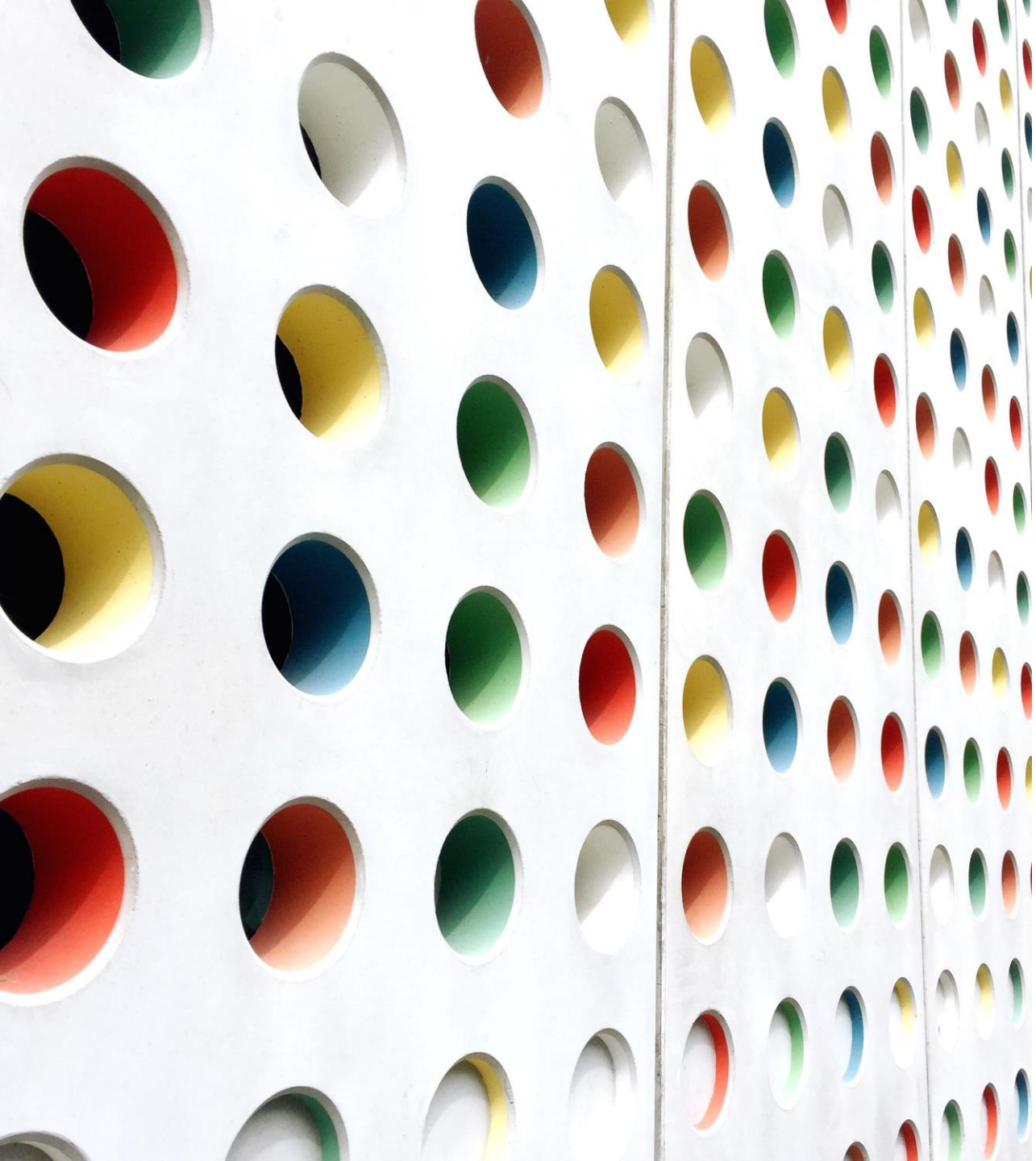


Experiments with AOT Compilation

ROSS TATE

main.mn

```
function main() {
    twice(fooey());
}
```

twice.mn

A

```
function twice(f : dyn) : dyn {
    return f.foo(f.foo(0));
}
```

⇓

B

```
function twice(f : dyn) : dyn {
    Foo F = f;
    return F.foo(F.foo(0));
}
```

⇓

C

```
function twice(f : Foo) : int {
    return f.foo(f.foo(0));
}
```

foo.mn

A ∅

⇓

B

```
interface Foo {
    foo(i : int) : int;
}
```

fooey.mn

A

```
function fooey() : dyn {
    dyn f = new {};
    f.foo = λ(i : dyn) : dyn { return i+1; };
    return f;
}
```

⇓

B

```
function fooey() : dyn {
    return new {
        foo(i : dyn) : dyn { return i+1; };
    };
}
```

⇓

C

```
function fooey() : dyn {
    return new {
        foo(i : int) : int { return i+1; };
    };
}
```

⇓

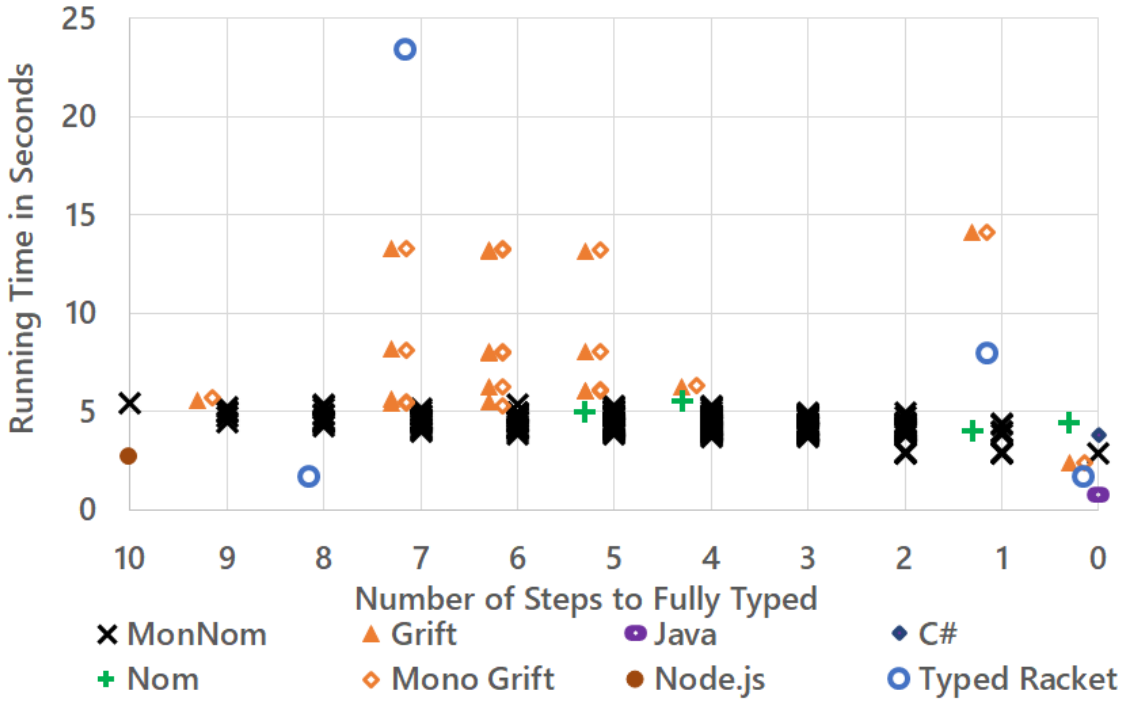
D

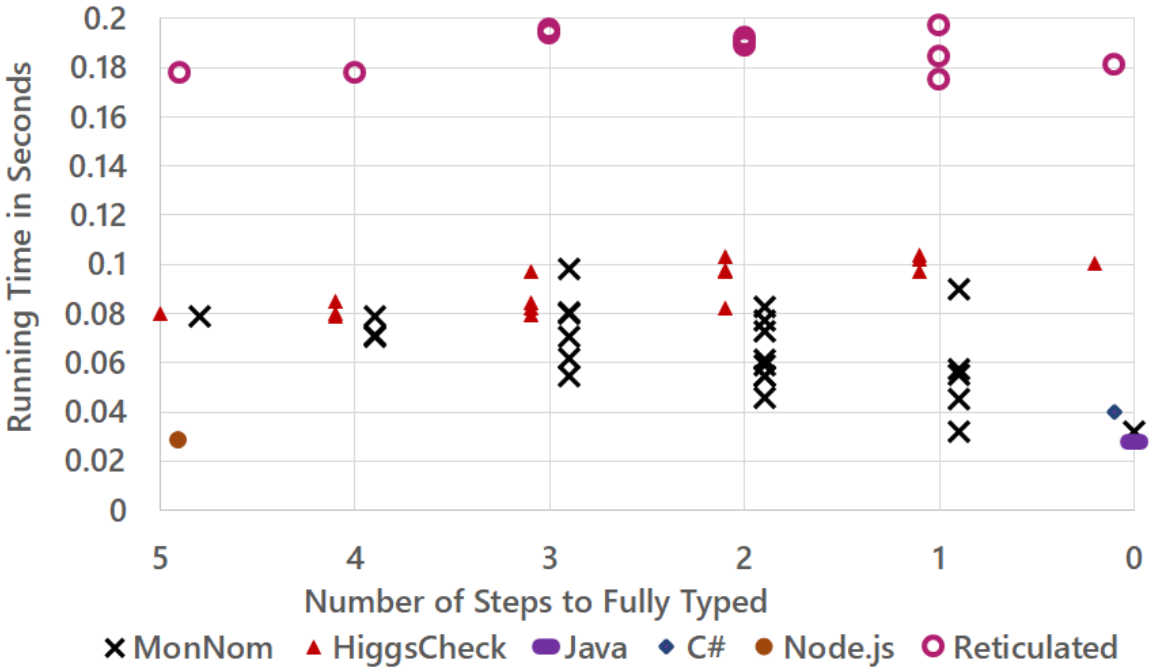
```
class Fooey() implements Foo {
    foo(i : int) : int { return i+1; };
}
function fooey() : dyn { return new Fooey(); }
```

⇓

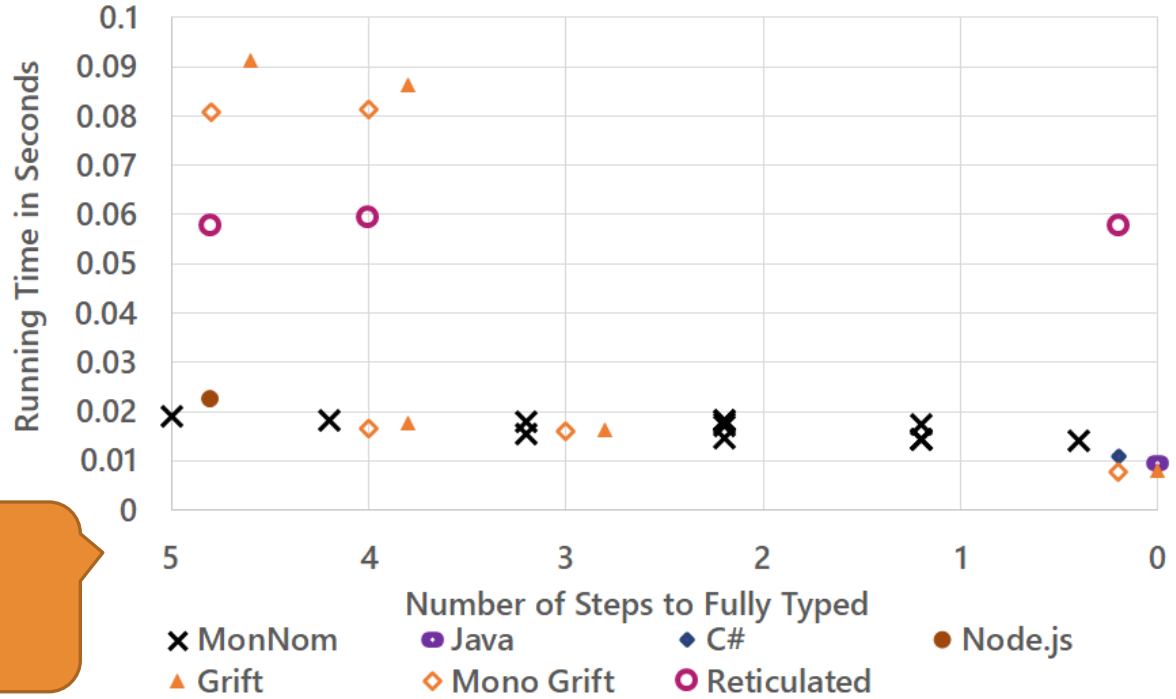
E

```
class Fooey() implements Foo {
    foo(i : int) : int { return i+1; };
}
function fooey() : Foo { return new Fooey(); }
```





Lambdas and Short-Lived Objects



Classes and Interfaces

64-Bit Floats

Hypothesis: WebAssembly should be able to match JavaScript *without* advanced optimizations

Caching

Observation: The L1 cache is amazing

- Typically no performance benefit gained from eliminating a load from the L1 cache
  - Even chained loads

Hypothesis: Module-instance data should be quickly accessible if in L1

Call Tags

We implemented interface methods and lambdas using call tags

- Used call-tag switching to handle conflicts between interface methods
- Used defaults to support typed-untyped interop (e.g. casting and representation conversion)

Class methods implemented using standard typed code pointers

Experiment:

- Replace all interfaces with classes in benchmark where each interface is implemented by only one class
- Effect on implementation: replaces uses of call tags with uses of typed code pointers
- Impact on performance: **None**

Hypothesis: Typed-function references add complexity without adding performance or expressiveness benefits

Hypothesis: Call tags add both performance and expressiveness benefits