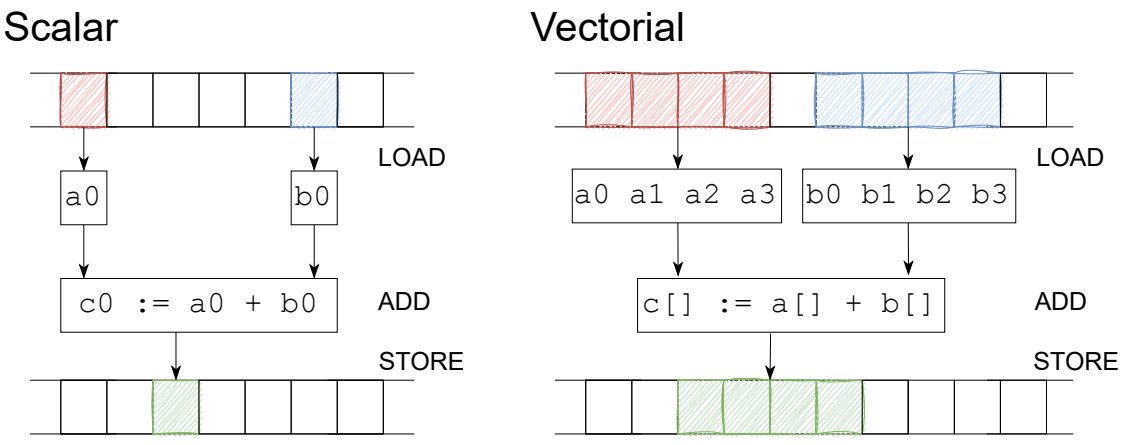
Analyzing the cost of safety for Vectorized bytecode in dynamically-typed languages

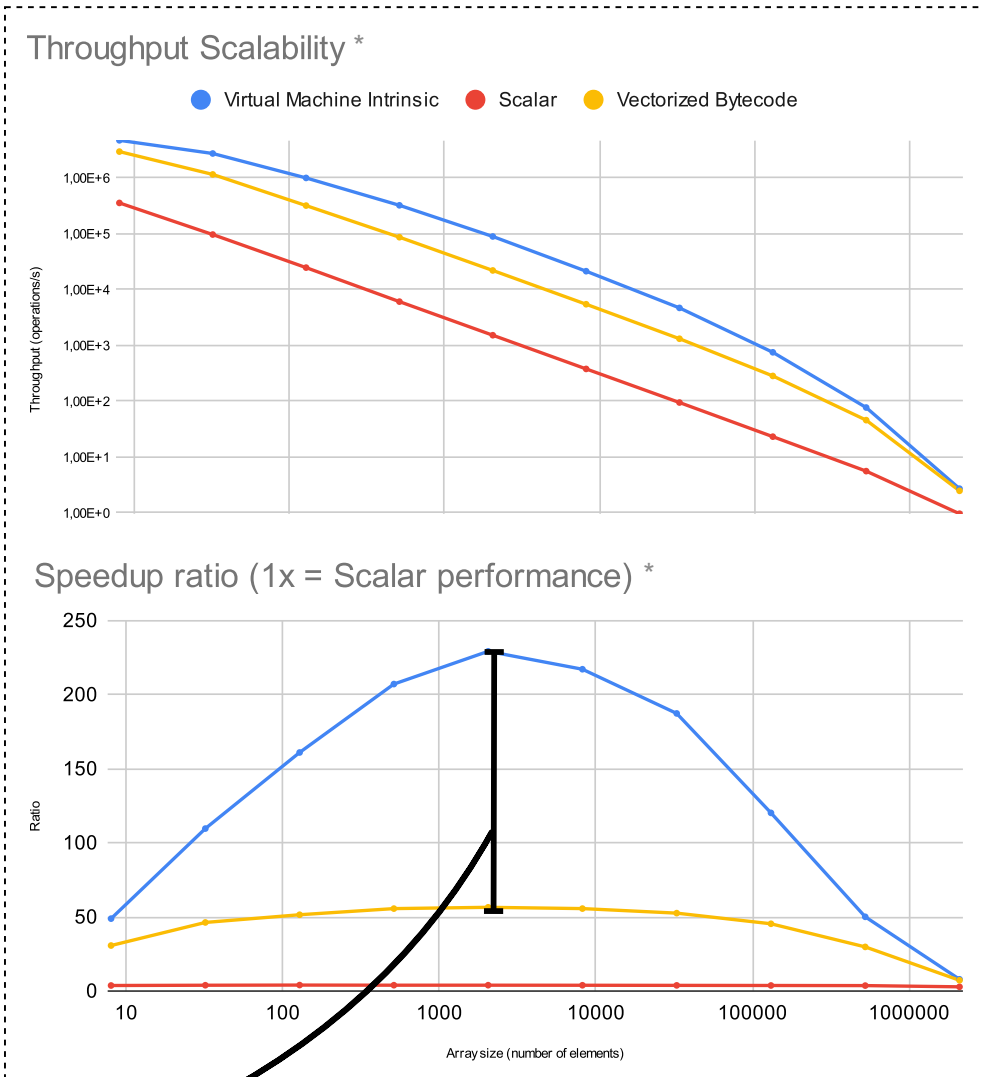
Nicolás Mauricio Rainhart, Guillermo Polito, Pablo Tesone, Stéphane Ducasse

Analyzing the cost of safety for Vectorized bytecode in dynamically-typed languages

What are vector instructions?



(*) Array addition benchmark
Higher is better



Two ways of integrating them into the VM:

- VM intrinsics / primitives

Lower-level, specialized

More performant

```
ldr x4, [x28, #8]       adds x4, x4, #8
ldr x3, [x28, #16]      adds x22, x22, #8
mov x1, #0              cmp x1, #0
ldurb w1, [x22, #7]     b.le #28
cmp x1, #255            ld1.2d {v0}, [x3], #16
b.lt #16                ld1.2d {v1}, [x4], #16
ldur x1, [x22, #-8]     fadd.2d v2, v0, v1
lsl x1, x1, #8          st1.2d {v2}, [x22], #16
lsr x1, x1, #8          subs x1, x1, #1
mov x23, x22            b.al #-28
lsr x1, x1, #1          add x28, x28, #8
                        ret
```

- Vectorized bytecode

Higher-level, composable

Slower

```
pushTemp: #index
pushTemp: #firstArray
pushFloat32ArrayToRegister

pushTemp: #index
pushTemp: #secondArray
pushFloat32ArrayToRegister

addFloat32Vector

pushTemp: #index
pushTemp: #resultArray
storeFloat32RegisterIntoArray
```

What makes bytecode slower?

- Overflow tests (for array iterator)
- **Type tests**
- **Bounds checks**

On many bytecodes!!

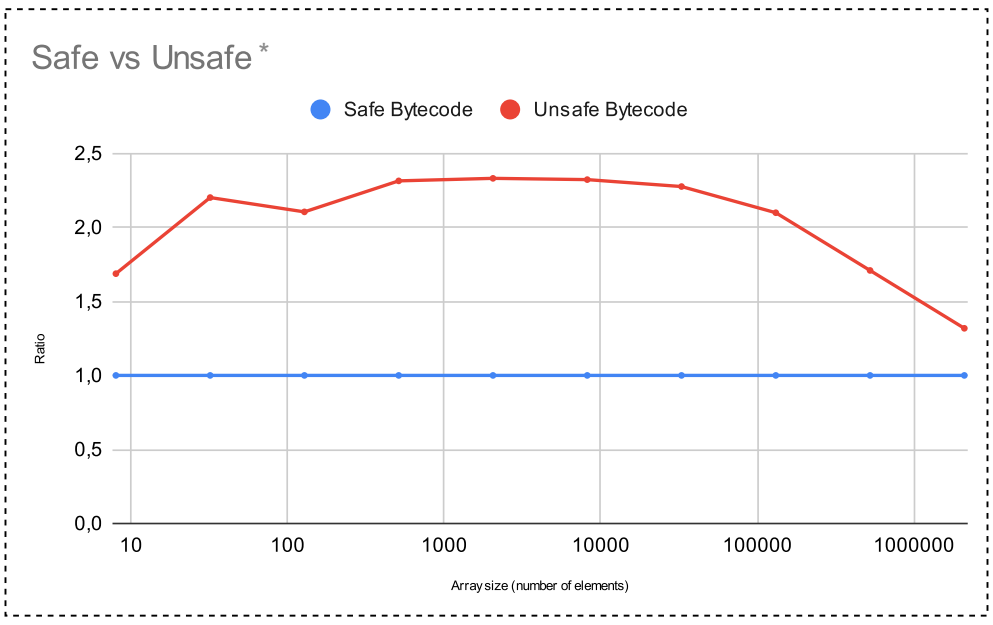
Example:

pushFloat32ArrayToRegisterBytecode

```
array := self popStackValue.
index := self popStackValue.

self assertIsIntegerValue: index      ┐ Type checks
self assertIs32BitArray: array        ┘
self assertIsIndex: index inArray: array   ┐ Bounds checks
self assertIsindex: index + 3 inArray: array ┘

<load array chunk into vector register>
```



Can we have the best of both worlds?

Performant vectorized bytecode

Future work:

- Optimizing Compiler
  - Overflow test elimination
  - Type test elimination
  - Bounds-checking elimination
- Error recovery / Deoptimization
  - Vector register spilling
  - Interpreter <-> JIT mapping

Other lines of research:

- Alternative bytecode designs
  - More granular instructions → Separate checks from actual operations
- Autovectorization
  - How to keep information about high-level patterns at the vectorizer level?

Nicolás Rainhart† Pablo Tesone†
Guillermo Polito† Stephane Ducasse†
† firstname.lastname@inria.ft