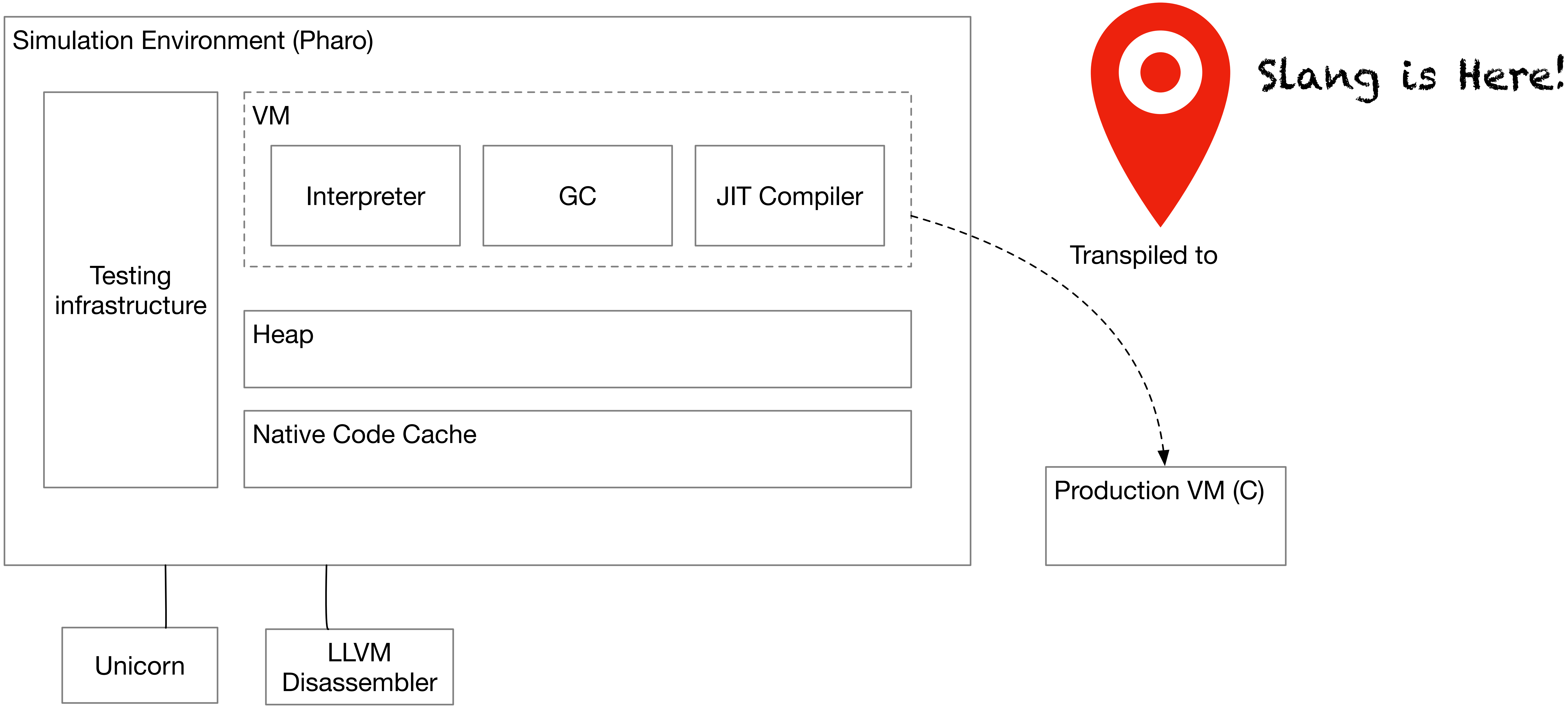
Interpreter Register Autolocalisation

Improving the performance of efficient interpreters

Guille Polito - **Nahuel Palumbo** - Soufyane Labsari - Pablo Tesone - Stéphane Ducasse
@guillep

Inria - RMod - cnrs - CRIStAL Centre de Recherche en Informatique, Signal et Automatique de Lille - Université de Lille

The Slang VM Generator



Simulation Environment (Pharo)

Testing infrastructure

VM

Interpreter

GC

JIT Compiler

Heap

Native Code Cache

Unicorn

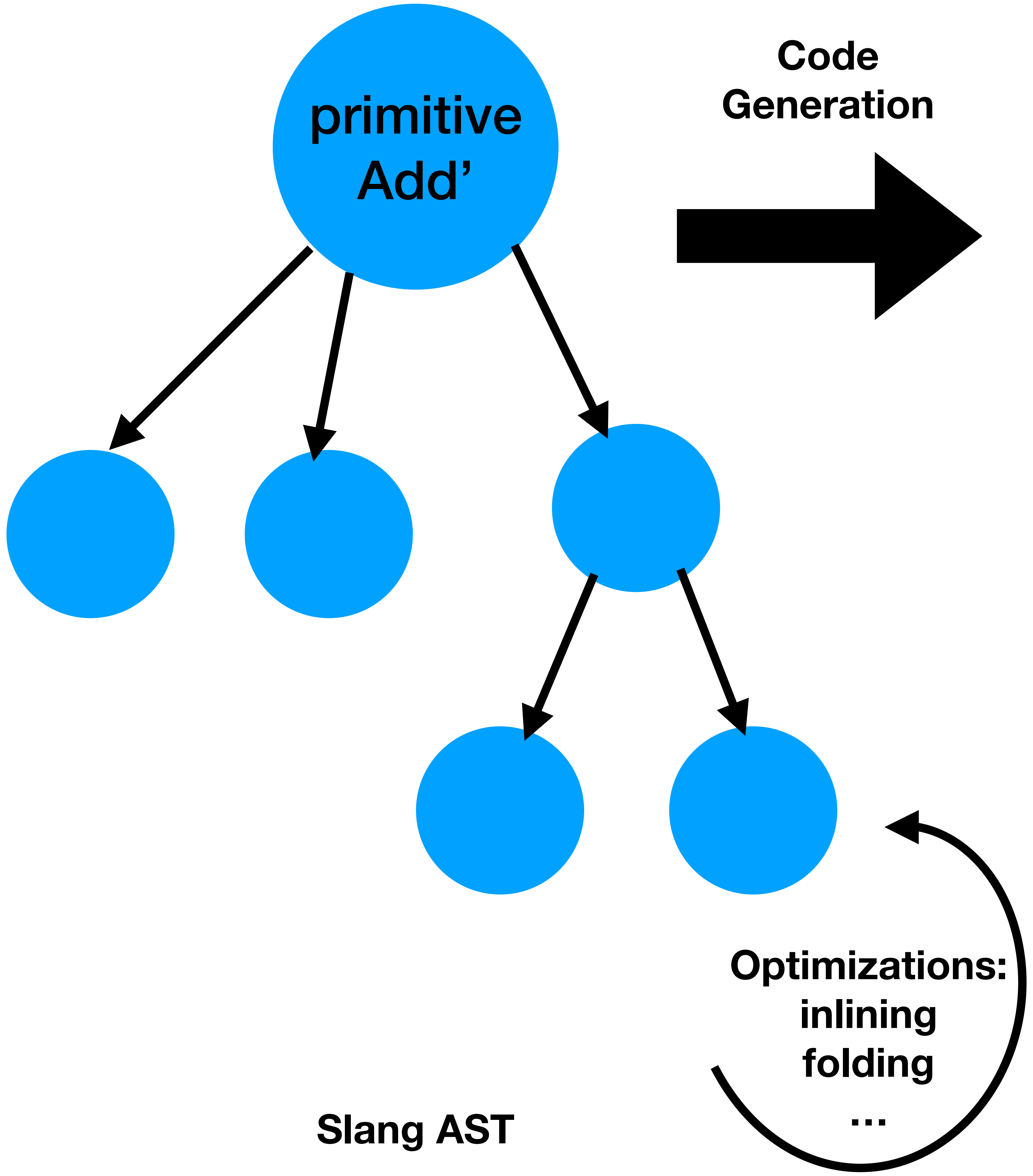
LLVM Disassembler

Slang is Here!

Transpiled to

Production VM (C)

Slang Code Generation

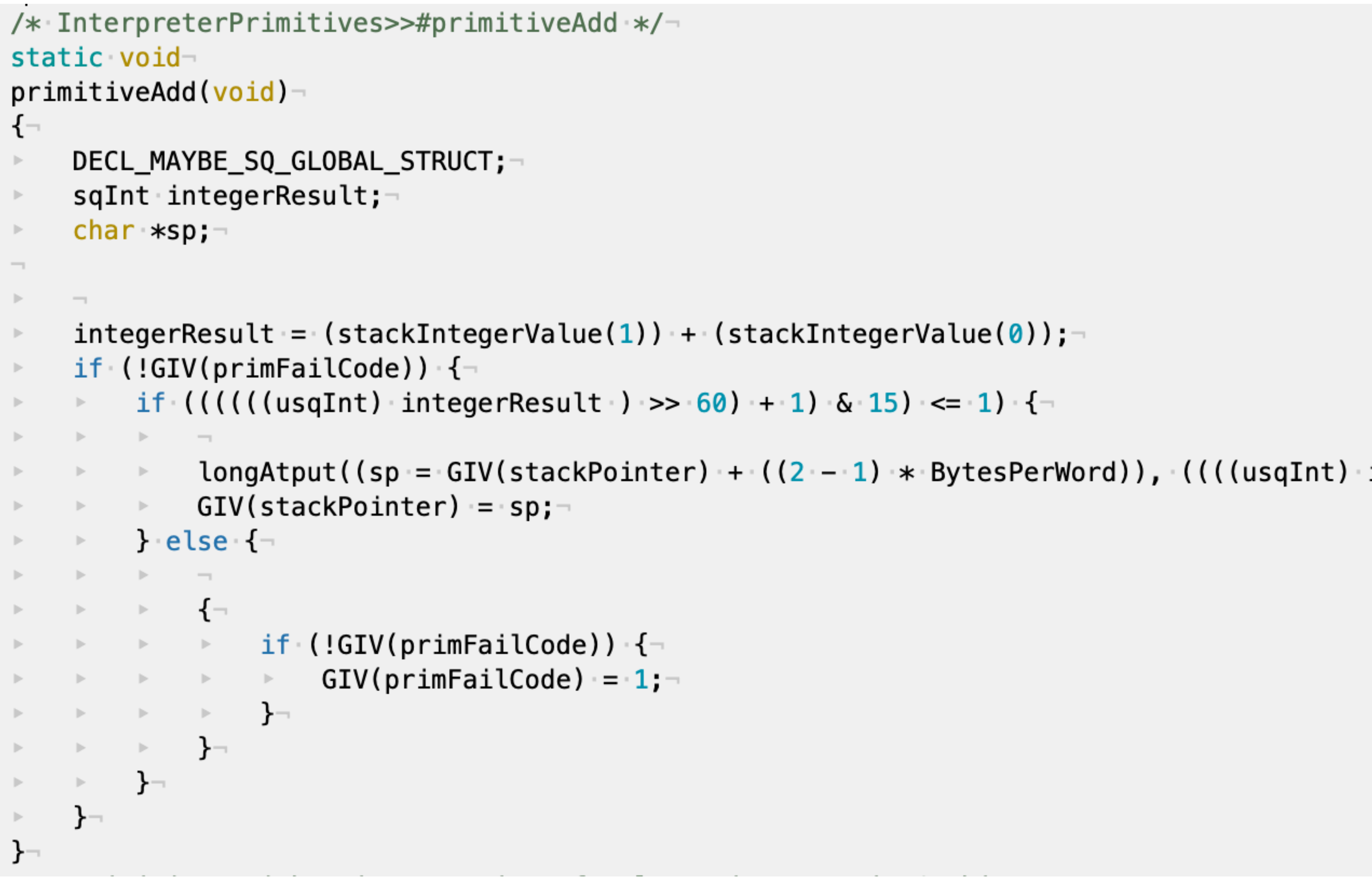


primitive Add’

Code Generation

Optimizations: inlining folding …

Slang AST



```
/* InterpreterPrimitives>>#primitiveAdd */
static void
primitiveAdd(void)
{
    DECL_MAYBE_SQ_GLOBAL_STRUCT;
    sqInt integerResult;
    char *sp;


    integerResult = (stackIntegerValue(1)) + (stackIntegerValue(0));
    if (!GIV(primFailCode)) {
        if ((((((usqInt) integerResult ) >> 60) + 1) & 15) <= 1) {

            longAtput((sp = GIV(stackPointer) + ((2 - 1) * BytesPerWord)), ((((usqInt)
            GIV(stackPointer) = sp;
        } else {

            {
                if (!GIV(primFailCode)) {
                    GIV(primFailCode) = 1;
                }
            }
        }
    }
}
```

The Slang VM Generator

And the Pharo VM

```
interpret
 self fetchNextBytecode.
  [ true ] whileTrue: [
    self
     dispatchOn: currentBytecode
     in: BytecodeTable ].
  ^ nil
```

- Stack based VM
- Bytecode Dispatch table
  - 1 bytecode = 1 method
- Transformed in a C token threaded interpreter

  + aggressive inlining

The Slang VM Generator

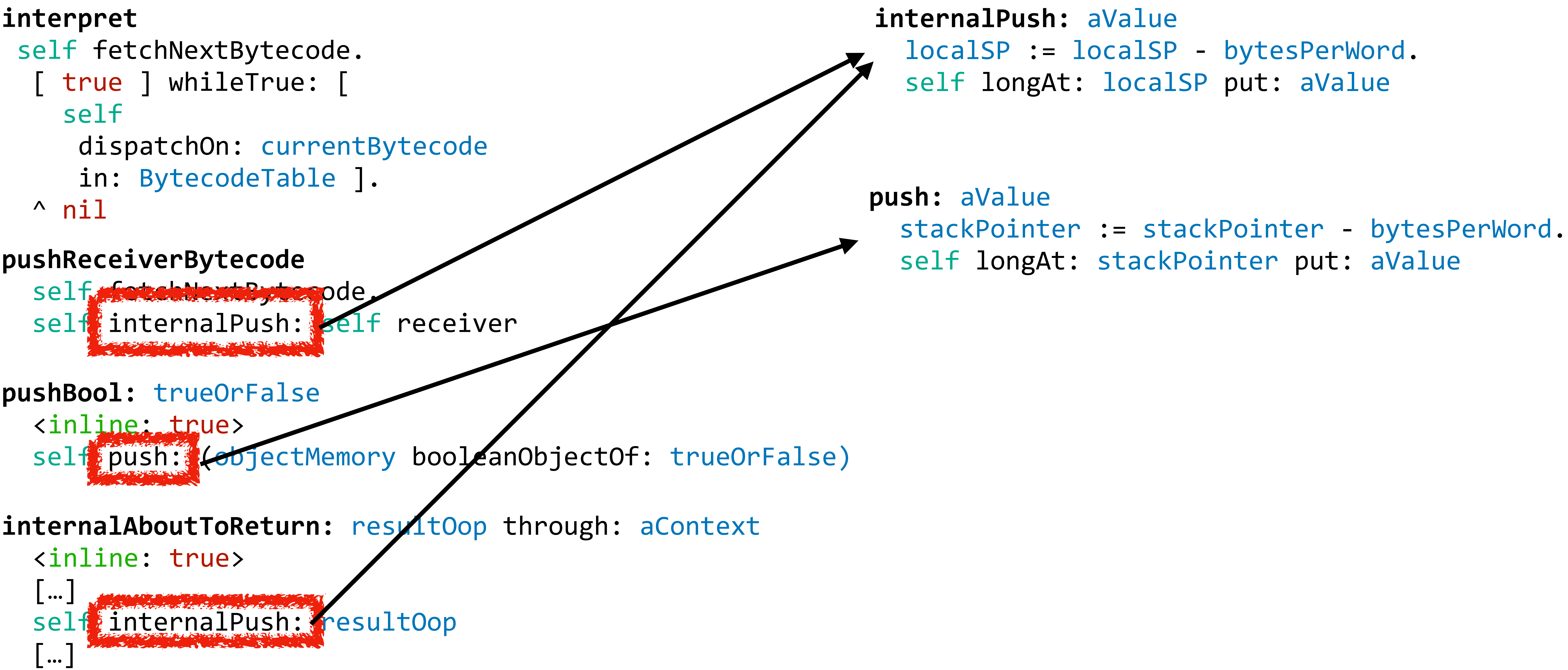
And the Pharo VM

```
interpret
 self fetchNextBytecode.
  [ true ] whileTrue: [
    self
     dispatchOn: currentBytecode
     in: BytecodeTable ].
  ^ nil

pushReceiverBytecode
  self fetchNextBytecode.
  self internalPush: self receiver

pushBool: trueOrFalse
  <inline: true>
  self push: (objectMemory booleanObjectOf: trueOrFalse)

internalAboutToReturn: resultOop through: aContext
  <inline: true>
  […]
  self internalPush: resultOop
  […]
```

- Stack based VM
- Bytecode Dispatch table
  - 1 bytecode = 1 method
- Transformed in a C token threaded interpreter

  + aggressive inlining

Pharo VM Manual Variable Localisation



```
interpret
 self fetchNextBytecode.
  [ true ] whileTrue: [
    self
     dispatchOn: currentBytecode
     in: BytecodeTable ].
  ^ nil

pushReceiverBytecode
  self fetchNextBytecode.
  self internalPush: self receiver

pushBool: trueOrFalse
  <inline: true>
  self push: (objectMemory booleanObjectOf: trueOrFalse)

internalAboutToReturn: resultOop through: aContext
  <inline: true>
  […]
  self internalPush: resultOop
  […]
```

```
internalPush: aValue
  localSP := localSP - bytesPerWord.
  self longAt: localSP put: aValue
```

```
push: aValue
  stackPointer := stackPointer - bytesPerWord.
  self longAt: stackPointer put: aValue
```

Interpreter Register Localisation

- Variables critical to the interpreter efficiency (e.g., IP, FP, SP)
- Variables are duplicated and synchronized
    - a local version accessible to the interpreter loop
      => meant to be optimised as registers
    - a global version accessible to the entire runtime
      => meant to be used by slower routines

```
internalPush: aValue
  localSP := localSP - bytesPerWord.
  self longAt: localSP put: aValue
```

```
push: aValue
  stackPointer := stackPointer - bytesPerWord.
  self longAt: stackPointer put: aValue
```

Is Interpreter Register Localisation Critical?

- Interpreter registers: interpreter variables with ***frequent*** usage
  e.g., IP, SP, FP, ??
    - **Intuition**: they are critical for performance
    - **Requirement**: need to be globally accessible for e.g.,
        - stack unwinding (exceptions, frame reification)
        - collection of GC roots
        - …
- But! Manually copying values of interpreter registers:
    - **is *error prone***
    - ***does not allow to systematically verify our intuition***

Automatic Localisation

Making Interpreter Registers Local at Translation Time

- An interpreter register (and using code) is defined only ***once***
- Automatic duplication on need

- Objectives:
    - Remove burden from VM developers
    - Allow systematic measure + specialisation of interpreter registers

Automatic Localisation

Making Interpreter Registers Local at Translation Time

```
var register1; // global
function interpret() {
  ...
  while(1) { switch(bytecode){
    // global reads and writes
    ... register1 ...
  } }
  return;
}
```

```
function interpret() {
  // localisation: copy from global
  var local_register1 := register1;
  ...
  while(1) { switch(bytecode){
    // local reads and writes
    ... local_register1 ...
  } }
  // globalisation: copy back to global
  register1 := local_register1;
  return;
}
```

Automatic Localisation

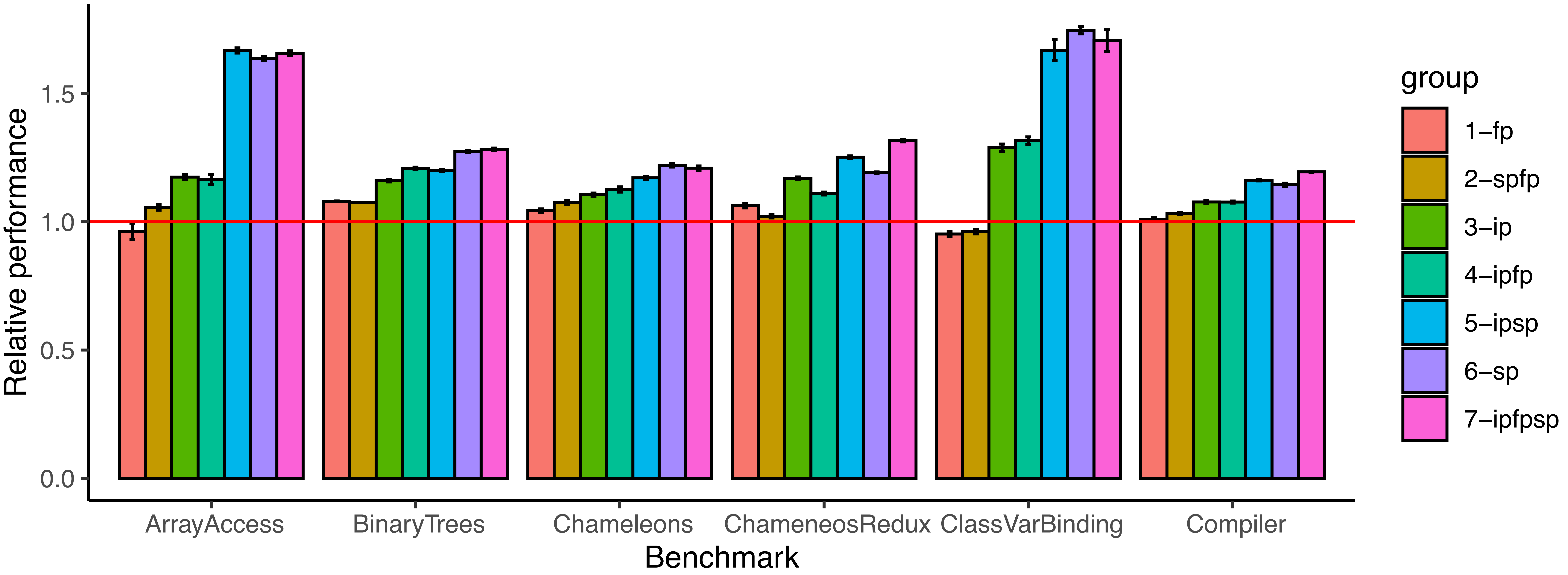
Synchronisation at Interpreter Exit Points

```
// inside the interpreter loop
…
register1 := local_register1; // globalisation
exit_point();
local_register1 := register1; // localisation
…

// outside the interpreter loop
function exit_point() {
  … register1 … // global reads and writes
}
```

+ **Callgraph Optimisation:** Only synchronise variables used by the called function

Some Benchmarks

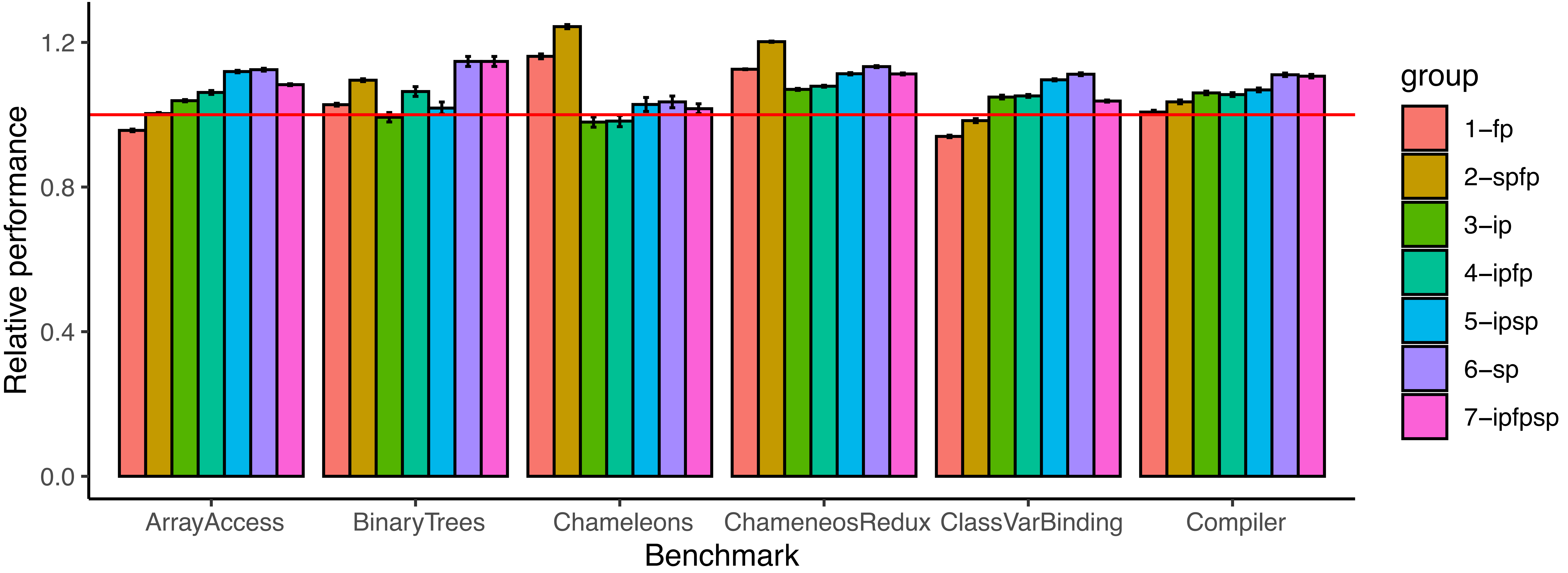
Intel x86-64



Averages of 100 iterations + stdev. Relative to baseline (no optimisation). Higher is better.

Some Benchmarks

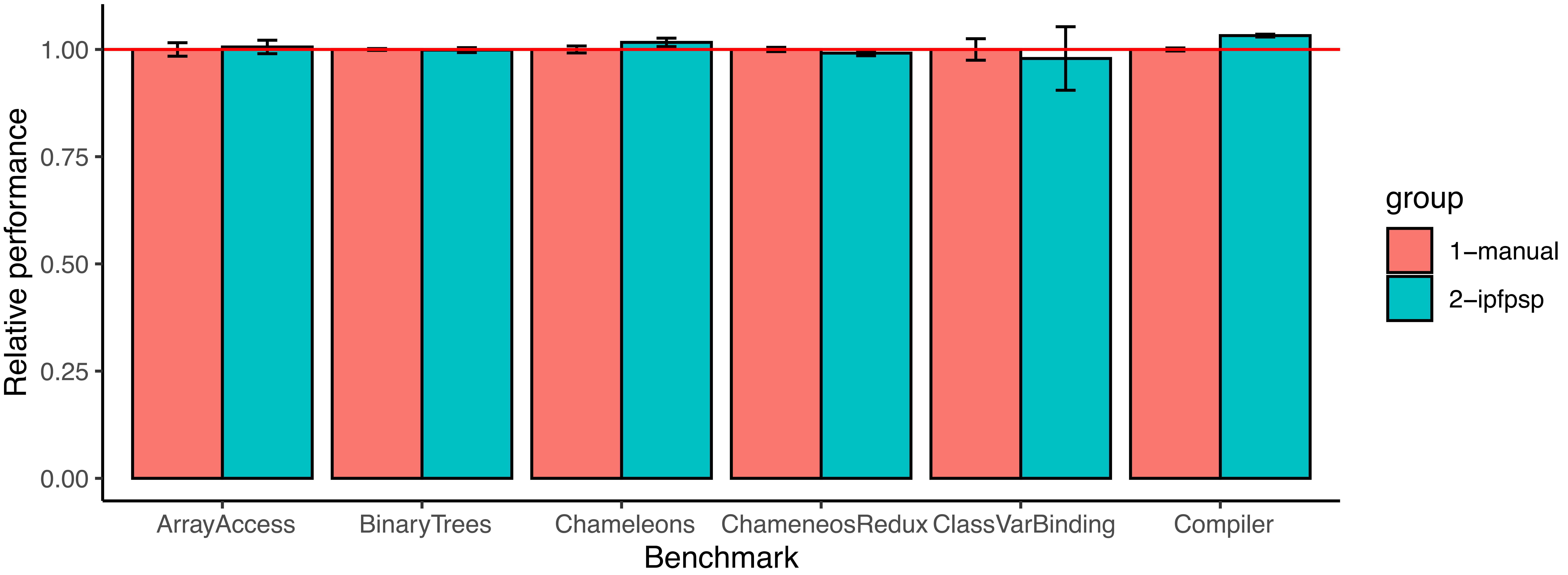
ARM64 - Raspberry Pi



Averages of 100 iterations + stdev. Relative to baseline (no optimisation). Higher is better.

Manual vs Automatic

Intel x86-64



Averages of 100 iterations. Relative to baseline (manual). Higher is better.

Conclusion

- Interpreter register localisation yields **improvements of up to 1.92x**
- Not all interpreter registers impact performance in the same way (FP?)
- It can be done automatically without loss of performance!

- Future:
  - Study the impact in different architectures
  - Study the CPU and cache impact of these optimizations
  - Is there an optional combination for different setups?
  - Are there variables other than IP, FP, SP that should be considered interpreter registers?