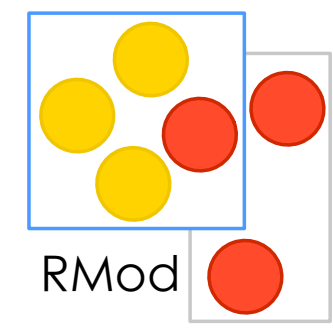




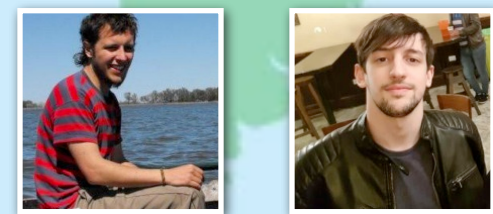
Pharo Virtual Machine

News from the Front

P. Tesone - G. Polito - 03/03/2022



2022 VM+ Team

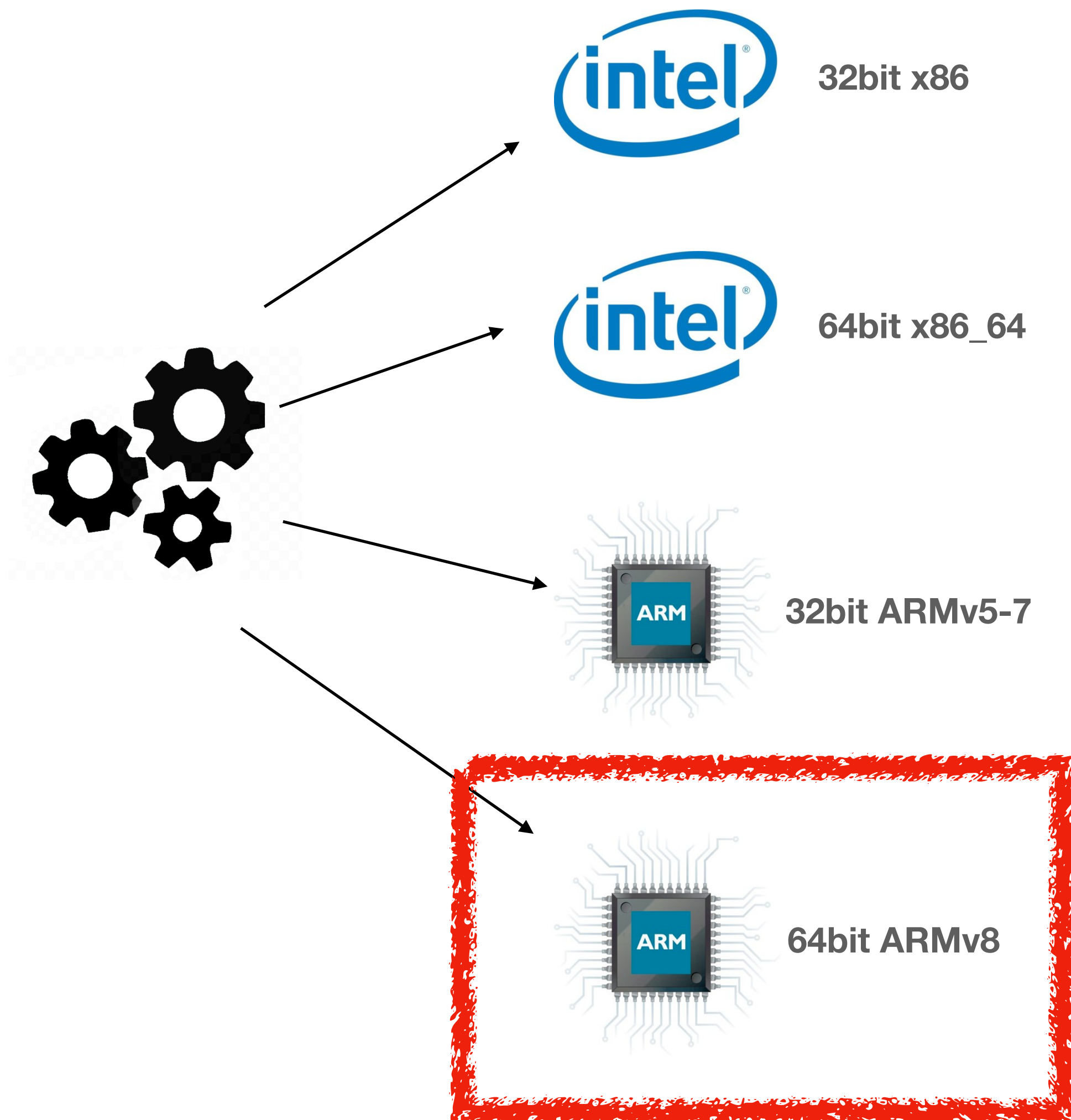


ARM64 Backend

- ARM64 is now pervasive:
 - New Apple M1
 - Raspberry Pi 4
 - Microsoft Surface Pro X
 - PineBook Pro
 - ...

```
move r1 #1
move r2 #17
checkSmallInt r1
checkSmallInt r2
add r3 r1 r2
checkSmallInt r3
move r1 r3
ret
```

JIT compiler IR



Testing & TDDing the VM

- No useful unit tests by ~06/2020
- Large manual testing effort during 2020 while porting to ARM64bits
 - Extended VM simulation with a (TDD compatible) unit testing infrastructure
 - **450+** written tests on the interpreter and the garbage collector*
 - **580+** written tests on the JIT compiler*
- Parametrisable for 32 and 64bits, ARM32, ARM64, x86, x86-64

* Numbers by 05/2021





VM Component	Operation	Independent Tests	Variations	Total Executions
<i>Test Infrastructure</i>	Method Builder	10		20
	Stack Builder	18	32 bits / 64 bits	36
	<i>Total</i>	28		56
<i>Object Memory</i>	Stack Reification	7		14
	Context / Stack Mapping	13		26
	GC Data Structures	13		26
	Unmovable Objects	9		18
	Old Object Garbage Collection	63		126
	Young Objects Garbage Collection	38	32 bits / 64 bits	76
	Weak Object Garbage Collection	9		18
	Ephemeron Object Garbage Collection	19		38
	Old Objects FreeSpace Management	85		170
Memory Structure Preconditions	30		60	
<i>Total</i>		286		572
<i>Interpreter</i>	Bytecode Tests	43		86
	Method Lookup	15		30
	Object Representation	17	32 bits / 64 bits	34
	Primitives	62		124
	<i>Total</i>		137	
Total		451		902

Testing & TDDing the VM

1040+ tests, are they enough?

- No useful unit tests by ~06/2020
- Large manual testing effort during 2020 while porting to ARM64bits
 - Extended VM simulation with a (TDD compatible) unit testing infrastructure
 - **450+** written tests on the interpreter and the garbage collector*
 - **580+** written tests on the JIT compiler*
- Parametrisable for 32 and 64bits, ARM32, ARM64, x86, x86-64

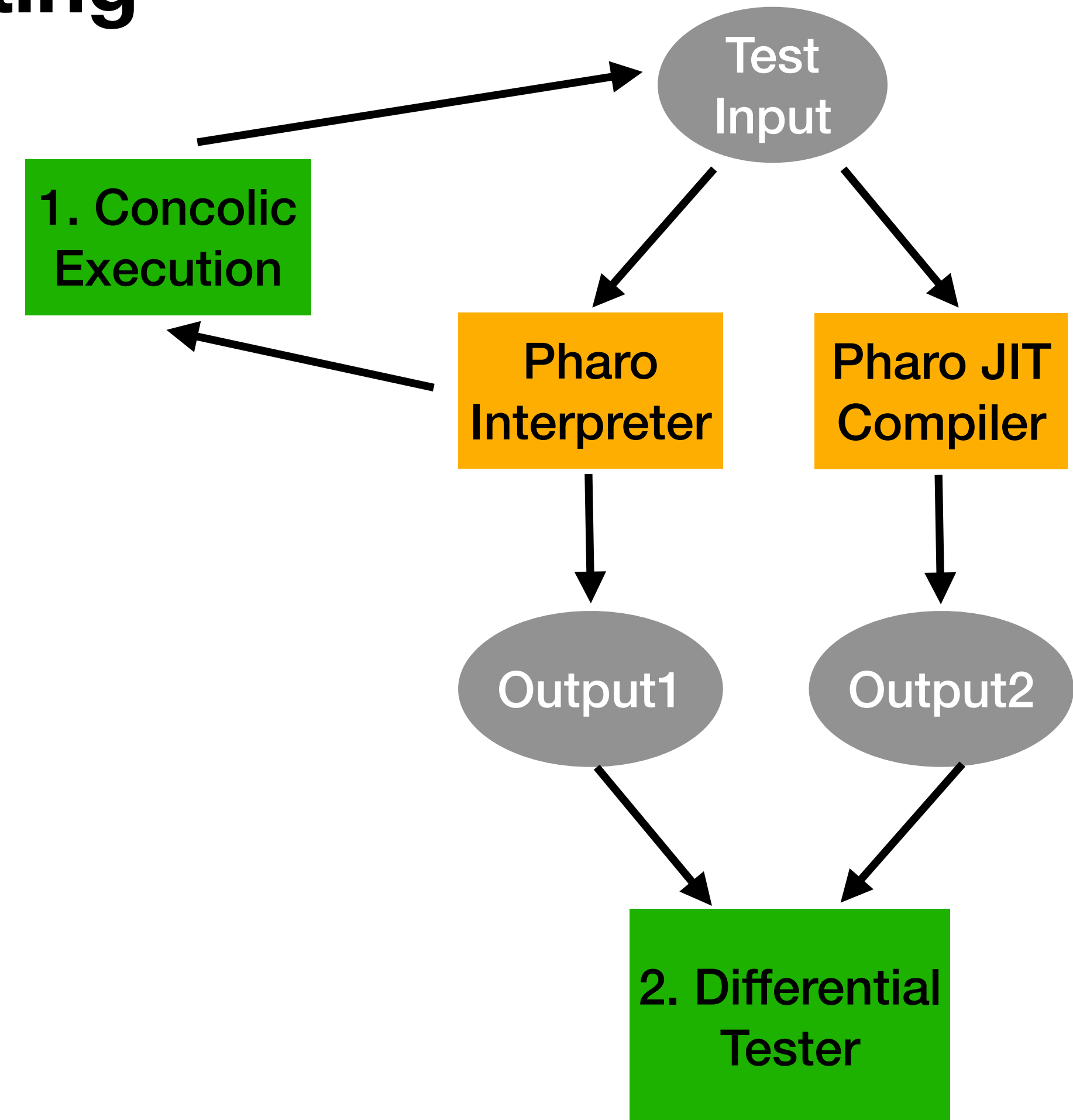
* Numbers by 05/2021



Automatic Test Generator

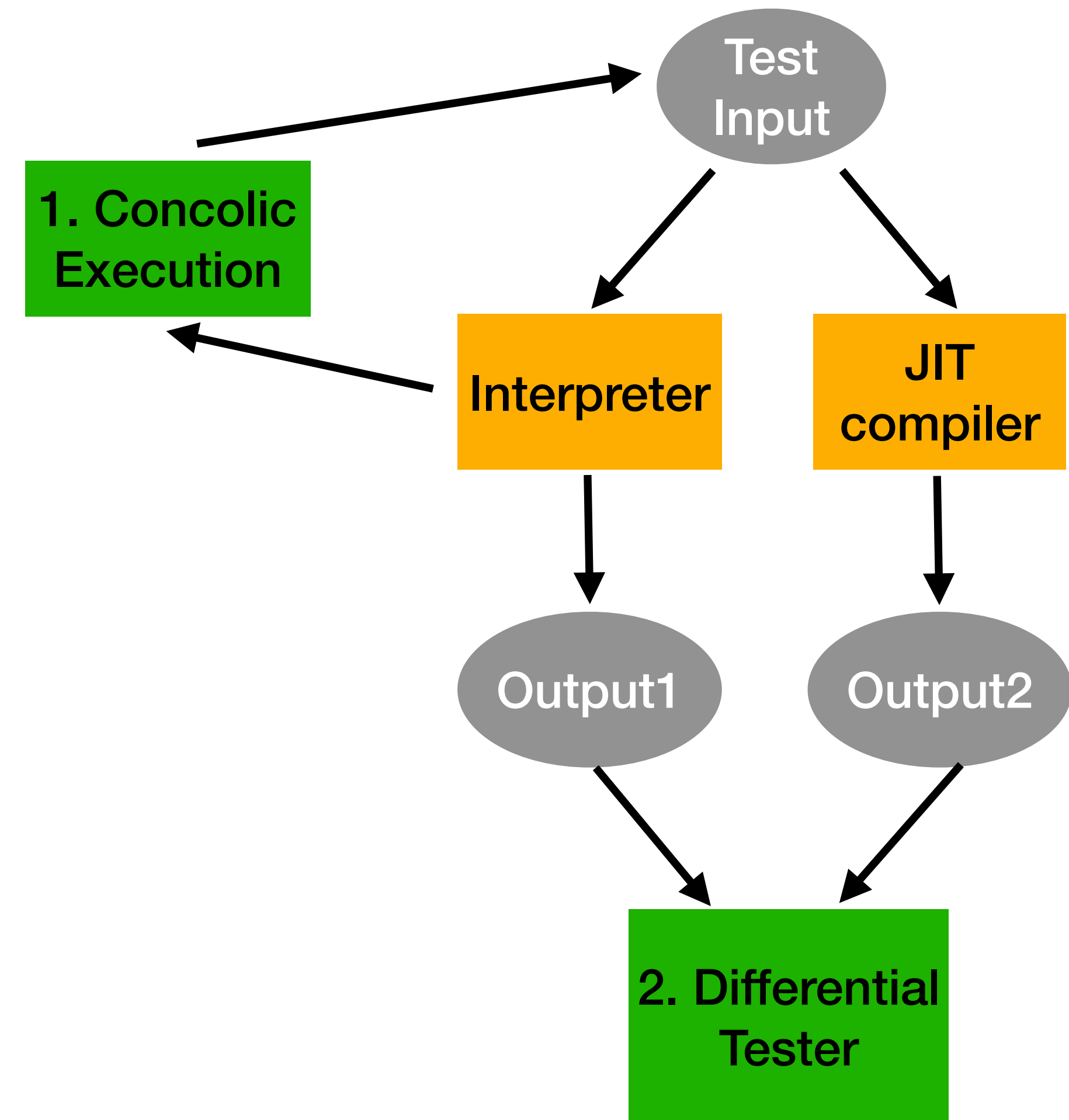
Interpreter-Guided JIT compiler Testing

1. Automatic test generation through **interpreter concolic execution**
2. Bug detection through differential testing



Automatic Test Generator

- Automatic test generation through **interpreter concolic execution**
- Bug detection through differential testing
- 468 differences found, 91 causes, 6 categories
- Practical:
 - 4582 tests generated in ~8 minutes
 - 4582 tests run in ~40 seconds

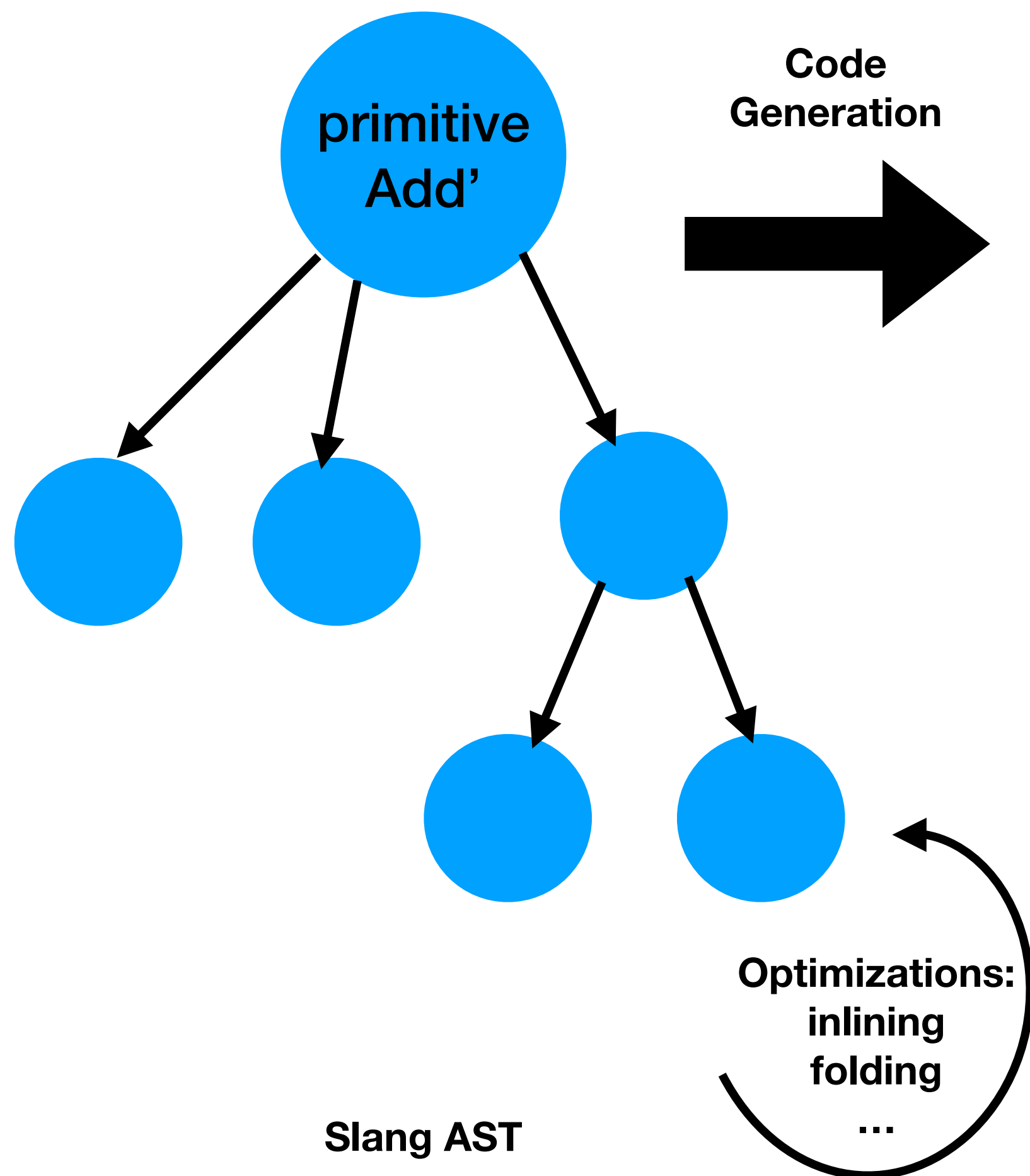


Accepted in PLDI'22 - Top Programming Language Conference



Improving Slang

Automatic transformations, simplifications, C AST nodes



```
/* 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) int  
            GIV(stackPointer) = sp;  
        } else {  
            if (!GIV(primFailCode)) {  
                GIV(primFailCode) = 1;  
            }  
        }  
    }  
}
```


Tools for Debugging

Insights: build your own tools, based on needs, not desires

The screenshot displays a VM Debugger window with the following components:

- IR Instructions:** A list of instructions with their IR representations, such as '(PopR 10 13503 810113)', '(Label 1)', '(TstCqR 7 10 757D93)', etc.
- Address:** Hexadecimal addresses for each instruction, ranging from 16r1000000 to 16r100005C.
- ASM:** Assembly code for each instruction, such as 'ld a0, 0(sp) #[3 53 1 0]', 'addi sp, sp, 8 #[19 1 129 0]', etc.
- Bytes:** Hexadecimal byte sequences for each instruction, such as '#[3 53 1 0]', '#[19 1 129 0]', etc.
- Registers:** A list of registers (lr, pc, sp, fp, x0-x22) and their current values, such as '16r1001000', '16r1000', etc.
- Stack/Frame Pointers:** SP (Stack Pointer) and FP (Frame Pointer) values, such as '16r1002FE8', '16r1002FF0', etc.
- Control Panel:** Buttons for 'Jump to', 'Step', and 'Disassemble at PC' at the bottom.

Examples:

- Machine code debugger
- Bytecode-IR visualization
- Disassembler DSL





Improvements: Clean Up

- V3 Support
- Old Memory Format
- Old Block Closures
- Dead Code
- ~ 65KLOC



Improvements: Sockets

- Unified Implementation in all Platforms
- Better Async Support
- Unix Sockets (Under Work)
- IPv6 Addresses (Under Work)



Improvements: Serial Port FFI

- Pure FFI implementation
- Working in all Platforms (Unix / Windows / OSX)
- Migrating Plugins to FFI

Improvements: RISCv64

Ongoing Port

- Currently under development: Real HW testing stage
- Taking advantage of our harness test suite.
- Improving tests and scenarios

- Collaboration with Q. Ducasse, P. Cortret, L. Lagadec from ENSTA Bretagne
- Future work on: Hardware-based security enforcement





Improvements: Open Build Service

Better Support for Linux Distributions

	Arch	Debian_10	Debian_9.0	Debian_Testing	Fedora_31	Fedora_32	Fedora_33	Raspbian_10	Raspbian_9.0		
	↑↓ x86_64↓	x86_64↓	x86_64↓	x86_64 ↑↓	x86_64↓	x86_64↓	x86_64↓	aarch64↓	x86_64↓	aarch64↓	x86_64↓
libffi7		succeeded	succeeded		succeeded	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded
libgit2-1		succeeded		failed							
pharo9	failed	succeeded	failed	failed	failed	failed	failed	succeeded	succeeded	failed	failed
pharo9-ui	succeeded	succeeded	succeeded	failed	succeeded	succeeded	succeeded		succeeded		succeeded

	Raspbian_9.0		openSUSE_Leap_15.1	openSUSE_Leap_15.2	openSUSE_Tumbleweed	xUbuntu_18.04	xUbuntu_19.04	xUbuntu_20.04	
	↑↓ arch64↓	x86_64↓	x86_64 ↑↓	x86_64 ↑↓	x86_64 ↑↓	x86_64 ↑↓	x86_64 ↑↓	aarch64↓	x86_64↓
libffi7	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded	succeeded
libgit2-1			succeeded	succeeded		succeeded	succeeded		succeeded
pharo9	failed	failed	failed	failed	failed	failed	succeeded	succeeded	succeeded
pharo9-ui		succeeded	succeeded	succeeded	succeeded	succeeded	succeeded		succeeded

Initial targets:

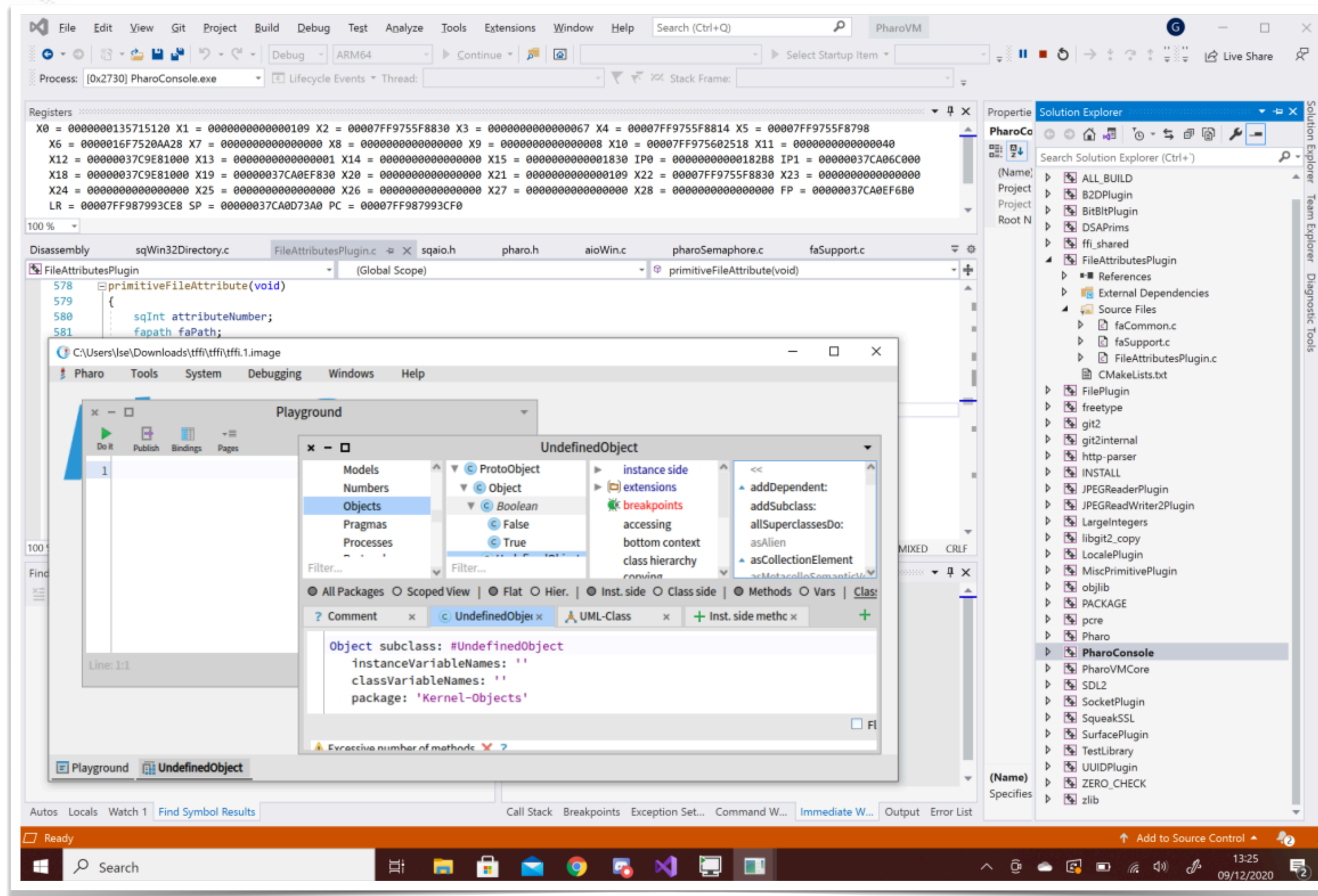
- Arch / Manjaro
- Debian
- Fedora
- Raspbian
- Ubuntu
- openSuse

Multiple Architectures

Supporting system packagings

Building using existing system libraries

Improvements: Visual Studio Support Building & Debugging



MSVC - No cygwin

Improvements: Windows ARM



C:\Users\Use\Downloads\tffi\tffi\tffi.1.image

Pharo Tools System Debugging Windows Help

Playground

```
1 canvas := RSCanvas new.  
2 shapes := #(20 10 5 30 24 32) collect: [ :e | RSEllipse model: e ]  
3 canvas addAll: shapes.  
4 RSNormalizer size shapes: shapes; normalize: #yourself.  
5 RSFlowLayout new alignCenter; on: shapes.  
6 canvas @ RSCanvasController
```

Test Runner

274 ran, 270 passed, 0 skipped, 3 expected failures, 0 failures, 1 error, 0 passed unexpected

Inspector on a RSCanvas

a RSCanvas

Canvas Raw Shapes Items Meta

CmdMenu>>asSpMenuPresenter

UML-Class

System Reporter

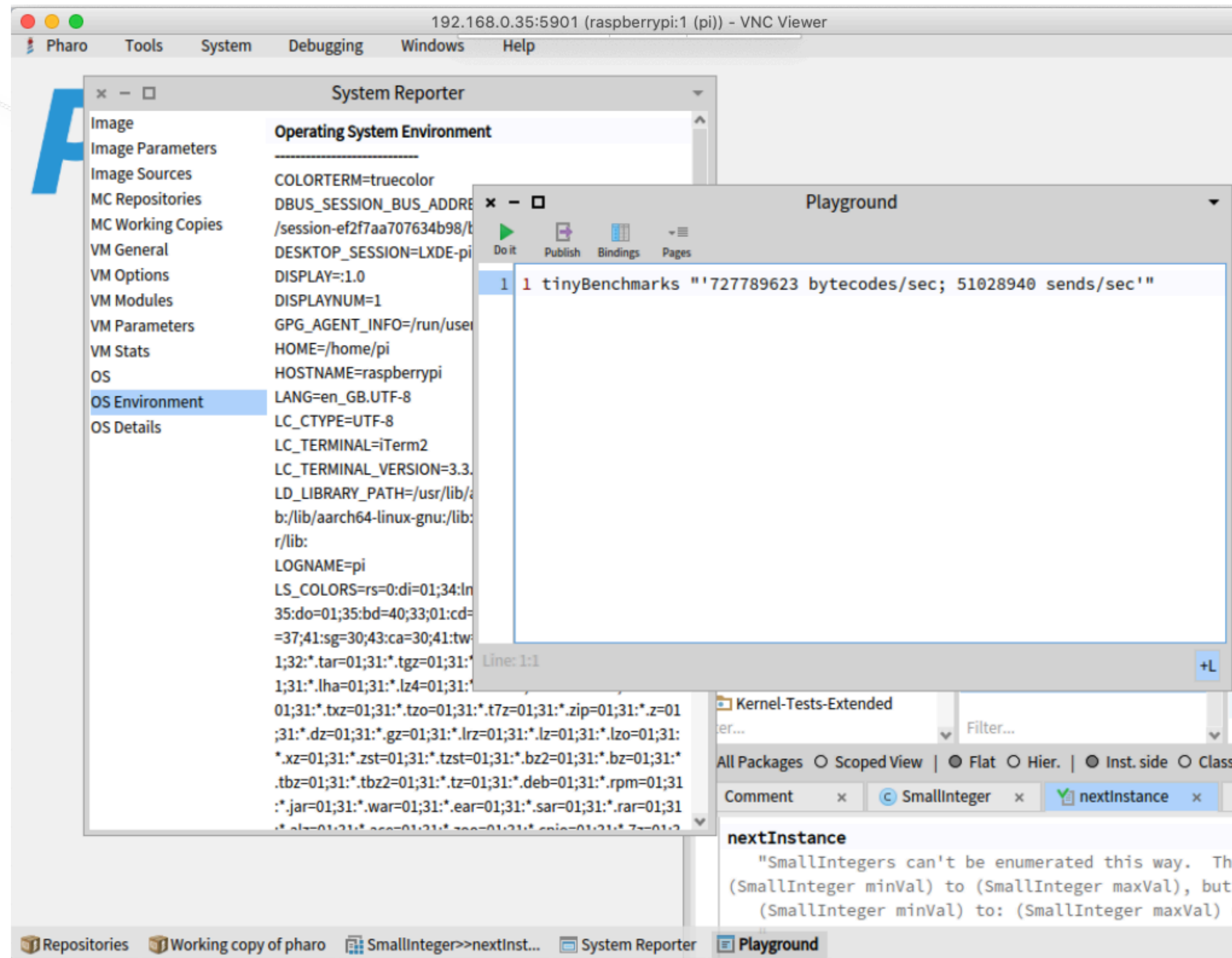
Operating System/Hardware

Win32 Windows-ARM64 ARM64

Windows taskbar: Type here to search, 15:08, 10/12/2020

MSVC - No cygwin

Improvements: Raspbian 32/64 bits



The screenshot displays the Pharo IDE interface. The main window is titled "System Reporter" and shows the "Operating System Environment" section. The environment variables listed include:

- COLORTERM=truecolor
- DBUS_SESSION_BUS_ADDRESS=/session-ef2f7aa707634b98/t...
- DESKTOP_SESSION=LXDE-pi
- DISPLAY=:1.0
- DISPLAYNUM=1
- GPG_AGENT_INFO=/run/user...
- HOME=/home/pi
- HOSTNAME=raspberrypi
- LANG=en_GB.UTF-8
- LC_CTYPE=UTF-8
- LC_TERMINAL=iTerm2
- LC_TERMINAL_VERSION=3.3
- LD_LIBRARY_PATH=/usr/lib/...
- LOGNAME=pi
- LS_COLORS=rs=0:di=01;34:ln...
- ...

Overlaid on the System Reporter is a "Playground" window. It contains a single line of code:

```
1 tinyBenchmarks "'727789623 bytecodes/sec; 51028940 sends/sec'"
```

Below the playground, a "nextInstance" window is visible, displaying an error message:

```
"SmallIntegers can't be enumerated this way. They are not ordered (SmallInteger minVal) to (SmallInteger maxVal), but you can iterate (SmallInteger minVal) to: (SmallInteger maxVal) de"
```

The bottom of the IDE shows a workspace with several objects, including "SmallInteger" and "nextInstance".



Back to the Future

Objectives for 2022



Permanent Space Problem

- Many permanent objects
- They have references from/to other objects
- We are traversing them to GC
- E.g., Classes, Methods, Literals, Resources



**Generates Long
Pauses (GC/
Saves/Loads)**



Permanent Space

Our Solution



- New Object Space for permanent Objects
- Minimise or Eliminate GC passes
- Persisting them through executions



Permanent Space

Our Solution



- New Object Space for permanent Objects
- Minimise or Eliminate GC passes
- Persisting them through executions

**We need to put
them in a
separate place**



New Image Format

Problem



- Current Image format only support a single object space
- No extensible: not new metadata nor new data
- Cannot be Memory Mapped (it is modified before save/load)
- Requires to discard all state of the running VM (slow saves)



New Image Format

Problem



- Current Image format only support a single object space
- No extensible: not new metadata nor new data
- Cannot be Memory Mapped (it is modified before save/load)
- Requires to discard all state of the running VM (slow saves)

**Slow and
Restricting
PermSpace**



New Image Format

Our Solution



- New Image format based in directories / bundles
- Many Elements of data and metadata
- Metadata en User & Machine readable format (STON?)
- Extensible format



Fast Snapshots / Loading

Based on PermSpace & Image Format

- Memory Mapped Image
- Shareable State
- Saving / Loading Warm State of the VM

Next Objectives

Permanent Space

New Image Format

Faster Startup / Saving

Ephemeron

Speculative
Compilation

- ARM64, RISCV64, Slang...
- Lots of Tests!
- Integration: Sockets, serial
- Visual Studio, Open Build Service

@pharoproject



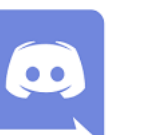
pharo.org



consortium-adm@pharo.org



discord.gg/QewZMZa



thepharo.dev

