Pharo Smalltalk as Universal Development Platform

Dave Mason
Ryerson Universityi

RYERSON UNIVERSITY

One IDE to rule them all,
One IDE to find them,
One IDE to bring them all
and in the syntax bind them.
– with apologies to J.R.R.Tolkien

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference:

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference:

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference:

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference:

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference: GlamourousToolit

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference: Roassal

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference: DrTest

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference: Seaside

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference: Scientific Workbench

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference: and so much more....

Why Pharo? (or other modern Smalltalks)

- want to program in Smalltalk
- best-in-class IDE
- live object debugging
- All the things at this conference: and so much more....
- including what Dave West said yesterday

What Pharo?

- desktop
- headless
- configurable images
- command line scripting
- ???

What Pharo?

- desktop
- headless
- configurable images
- command line scripting
- web with PharoJS

What Pharo?

- desktop
- headless
- configurable images
- command line scripting
- web with PharoJS
- standalone with Illicium???

What Pharo?

- desktop
- headless
- configurable images
- command line scripting
- web with PharoJS
- standalone with Illicium???

What Pharo?

- desktop
- headless
- configurable images
- command line scripting
- web with PharoJS
- standalone with Illicium???

What Pharo?

- desktop
- headless
- configurable images
- command line scripting
- web with PharoJS
- standalone with Illicium???

How Pharo?

- primitives/plug-ins
- Foreign Function Interface - FFI
- communication
- transpilation: PharoJS, Illicium, PharoJVM, Woden
- embedded VM

How Pharo?

- primitives/plug-ins Slang/Illicium
- Foreign Function Interface - FFI
- communication
- transpilation: PharoJS, Illicium, PharoJVM, Woden
- embedded VM

How Pharo?

- primitives/plug-ins Slang/Illicium
- Foreign Function Interface - FFI
- communication
- transpilation: PharoJS, Illicium, PharoJVM, Woden
- embedded VM

How Pharo?

- primitives/plug-ins Slang/Illicium
- Foreign Function Interface - FFI
- communication
- transpilation: PharoJS, Illicium, PharoJVM, Woden
- embedded VM

How Pharo?

- primitives/plug-ins Slang/Illicium
- Foreign Function Interface - FFI
- communication Python-Bridge
- transpilation: PharoJS, Illicium, PharoJVM, Woden
- embedded VM

How Pharo?

- primitives/plug-ins Slang/Illicium
- Foreign Function Interface - FFI
- communication Python-Bridge
- transpilation: PharoJS, Illicium, PharoJVM, Woden
- embedded VM

How Pharo?

- primitives/plug-ins Slang/Illicium
- Foreign Function Interface - FFI
- communication Python-Bridge
- transpilation: PharoJS, Illicium, PharoJVM, Woden
- embedded VM

Why Transpile?

- convert from a friendly (Smalltalk) language to ugly one
- deployment target limitations (GPU/Browser/cloud provider/footprint)

Why Transpile?

- convert from a friendly (Smalltalk) language to ugly one
- deployment target limitations (GPU/Browser/cloud provider/footprint)

How Transpile?

- walk the compiler AST
- typically translate to target AST
- possible type-checking/type-inference - helped by target types
- optional optimization of target AST
- walk the target AST generating target code

How Transpile?

- walk the compiler AST
- typically translate to target AST
- possible type-checking/type-inference - helped by target types
- optional optimization of target AST
- walk the target AST generating target code

How Transpile?

- walk the compiler AST
- typically translate to target AST
- possible type-checking/type-inference - helped by target types
- optional optimization of target AST
- walk the target AST generating target code

How Transpile?

- walk the compiler AST
- typically translate to target AST
- possible type-checking/type-inference - helped by target types
- optional optimization of target AST
- walk the target AST generating target code

How Transpile?

- walk the compiler AST
- typically translate to target AST
- possible type-checking/type-inference - helped by target types
- optional optimization of target AST
- walk the target AST generating target code

Example: Why PharoJVM?

- want to program in Smalltalk
- sometimes need to deploy in enterprise (e.g. WebSphere)
- accessing Java frameworks - Minecraft, Yarn
- possible performance advantage
- may combine with PharoJS to target WebAssembly/native
- might allow writing Android apps directly

Example: Why PharoJVM?

- want to program in Smalltalk
- sometimes need to deploy in enterprise (e.g. WebSphere)
- accessing Java frameworks - Minecraft, Yarn
- possible performance advantage
- may combine with PharoJS to target WebAssembly/native
- might allow writing Android apps directly

Example: Why PharoJVM?

- want to program in Smalltalk
- sometimes need to deploy in enterprise (e.g. WebSphere)
- accessing Java frameworks - Minecraft, Yarn
- possible performance advantage
- may combine with PharoJS to target WebAssembly/native
- might allow writing Android apps directly

Example: Why PharoJVM?

- want to program in Smalltalk
- sometimes need to deploy in enterprise (e.g. WebSphere)
- accessing Java frameworks - Minecraft, Yarn
- possible performance advantage
- may combine with PharoJS to target WebAssembly/native
- might allow writing Android apps directly

Example: Why PharoJVM?

- want to program in Smalltalk
- sometimes need to deploy in enterprise (e.g. WebSphere)
- accessing Java frameworks - Minecraft, Yarn
- possible performance advantage
- may combine with PharoJS to target WebAssembly/native
- might allow writing Android apps directly

Example: Why PharoJVM?

- want to program in Smalltalk
- sometimes need to deploy in enterprise (e.g. WebSphere)
- accessing Java frameworks - Minecraft, Yarn
- possible performance advantage
- may combine with PharoJS to target WebAssembly/native
- might allow writing Android apps directly

Minecraft example

in Java

```
package lavavision;

import java.util.logging.Logger;
import org.bukkit.command.Command;
import org.bukkit.command.CommandSender;
import org.bukkit.entity.Player;
import org.bukkit.plugin.Plugin;
import org.bukkit.plugin.java.JavaPlugin;
import org.bukkit.block.Block;
import org.bukkit.util.BlockIterator;
import org.bukkit.Material;
import org.bukkit.Sound;
import org.bukkit.Effect;

public class LavaVision extends JavaPlugin {

  public boolean onCommand(CommandSender sender, Command command,
                           String commandLabel, String[] args) {
    if (commandLabel.equalsIgnoreCase("lavavision")) {
      if (sender instanceof Player) {
        Player me = (Player)sender;
        BlockIterator sightItr = new BlockIterator (me, 100);
        while (sightItr.hasNext()) {
          Block b = sightItr.next();
          me.playEffect(b.getLocation(), Effect.MOBSPAWNER_FLAMES, null);
          if (b.getType() != Material.AIR) {
            b.setType(Material.LAVA);
            me.playSound(b.getLocation(), Sound.ENTITY_ENDERDRAGON_FIREBALL_EXPLODE, 1.0f, 0.5f);
            break;
          }
        }
        return true;
      }
    }
    return false;
  }
```

in Smalltalk

```
onCommand2: me command: command label: commandLabel args: args
  {org bukkit entity. org bukkit util. org bukkit Effect. org bukkit Material. org bukkit Sound.} scope: [: s |
    (commandLabel equalsIgnoreCase: 'lavavision' ) ifTrue: [
      (me isKindOf: s Player) ifTrue: [
        (s BlockIterator new: me with: 100) do: [: b |
          me playEffect: b getLocation effect: s MOBSPAWNER_FLAMES ignore: nil.
          b getType = s AIR ifFalse: [
            b setType: s LAVA.
            me playSound: b getLocation sound: s ENTITY_ENDERDRAGON_FIREBALL_EXPLODE
                volume: 1.0 pitch: 0.5.
            ^ true
  ]]]]].
  ^ false
```

Challenges - Block Closures

```
1  eg1: param
2      | x y z |
3      x := 0.
4      z := y := 42.
5      #(2 5 7) asOrderedCollection
6          do: [: each | | w |
7          w := y + param;
8          x := x + w.
9          x > 10 ifTrue: [↑ y].
10     ].
11     ↑ x + z
```

Challenges - PharoJS

- both have dynamic/manifest types
- Javascript is prototype, Smalltalk is class-based
- non-local return
- *almost* everything is an object
- deficit numeric stack, strings immutable
- infix arithmetic faster than method calls - need to leverage

Challenges - PharoJS

- both have dynamic/manifest types
- Javascript is prototype, Smalltalk is class-based
- non-local return
- *almost* everything is an object
- deficit numeric stack, strings immutable
- infix artithmetic faster than method calls - need to leverage

Challenges - PharoJS

- both have dynamic/manifest types
- Javascript is prototype, Smalltalk is class-based
- non-local return
- *almost* everything is an object
- deficit numeric stack, strings immutable
- infix artithmetic faster than method calls - need to leverage

Challenges - PharoJS

- both have dynamic/manifest types
- Javascript is prototype, Smalltalk is class-based
- non-local return
- *almost* everything is an object
- deficit numeric stack, strings immutable
- infix artithmetic faster than method calls - need to leverage

Challenges - PharoJS

- both have dynamic/manifest types
- Javascript is prototype, Smalltalk is class-based
- non-local return
- *almost* everything is an object
- deficit numeric stack, strings immutable
- infix artithmetic faster than method calls - need to leverage

Challenges - PharoJS

- both have dynamic/manifest types
- Javascript is prototype, Smalltalk is class-based
- non-local return
- *almost* everything is an object
- deficit numeric stack, strings immutable
- infix artithmetic faster than method calls - need to leverage

Challenges - PharoJVM

- Smalltalk has dynamic/manifest types
- the JVM is fundamentally staticly typed - even though many new Java language changes hide this
- could create everything as `Object` - like Scala, Jython, Redline, jRuby
- e.g. `new Integer(42)` - will heap allocate, `final`, no methods for add, etc.
- includes 8 primitive types (not Objects - although)
- for performance, need to leverage this
- non-local return, BlockClosure, Symbols, immutable String, deficient numeric stack

Challenges - PharoJVM

- Smalltalk has dynamic/manifest types
- the JVM is fundamentally staticly typed - even though many new Java language changes hide this
- could create everything as `Object` - like Scala, Jython, Redline, jRuby
- e.g. `new Integer(42)` - will heap allocate, `final`, no methods for add, etc.
- includes 8 primitive types (not Objects - although)
- for performance, need to leverage this
- non-local return, BlockClosure, Symbols, immutable String, deficient numeric stack

Challenges - PharoJVM

- Smalltalk has dynamic/manifest types
- the JVM is fundamentally staticly typed - even though many new Java language changes hide this
- could create everything as `Object` - like Scala, Jython, Redline, jRuby
- e.g. `new Integer(42)` - will heap allocate, `final`, no methods for add, etc.
- includes 8 primitive types (not Objects - although)
- for performance, need to leverage this
- non-local return, BlockClosure, Symbols, immutable String, deficient numeric stack

Challenges - PharoJVM

- Smalltalk has dynamic/manifest types
- the JVM is fundamentally staticly typed - even though many new Java language changes hide this
- could create everything as `Object` - like Scala, Jython, Redline, jRuby
- e.g. `new Integer(42)` - will heap allocate, `final`, no methods for add, etc.
- includes 8 primitive types (not Objects - although)
- for performance, need to leverage this
- non-local return, BlockClosure, Symbols, immutable String, deficient numeric stack

Challenges - PharoJVM

- Smalltalk has dynamic/manifest types
- the JVM is fundamentally staticly typed - even though many new Java language changes hide this
- could create everything as `Object` - like Scala, Jython, Redline, jRuby
- e.g. `new Integer(42)` - will heap allocate, `final`, no methods for add, etc.
- includes 8 primitive types (not Objects - although)
- for performance, need to leverage this
- non-local return, BlockClosure, Symbols, immutable String, deficient numeric stack

Challenges - PharoJVM

- Smalltalk has dynamic/manifest types
- the JVM is fundamentally staticly typed - even though many new Java language changes hide this
- could create everything as `Object` - like Scala, Jython, Redline, jRuby
- e.g. `new Integer(42)` - will heap allocate, `final`, no methods for add, etc.
- includes 8 primitive types (not Objects - although)
- for performance, need to leverage this
- non-local return, BlockClosure, Symbols, immutable String, deficient numeric stack

Challenges - PharoJVM

- Smalltalk has dynamic/manifest types
- the JVM is fundamentally staticly typed - even though many new Java language changes hide this
- could create everything as `Object` - like Scala, Jython, Redline, jRuby
- e.g. `new Integer(42)` - will heap allocate, `final`, no methods for add, etc.
- includes 8 primitive types (not Objects - although)
- for performance, need to leverage this
- non-local return, BlockClosure, Symbols, immutable String, deficient numeric stack

Challenges - Python/Ruby

- why bother?
- can provide immediate access to libraries
- similar enough that long-term should be to leverage FFI
- non-local return, BlockClosure, missing fractions

Challenges - Python/Ruby

- why bother?
- can provide immediate access to libraries
- similar enough that long-term should be to leverage FFI
- non-local return, BlockClosure, missing fractions

Challenges - Python/Ruby

- why bother?
- can provide immediate access to libraries
- similar enough that long-term should be to leverage FFI
- non-local return, BlockClosure, missing fractions

Challenges - Python/Ruby

- why bother?
- can provide immediate access to libraries
- similar enough that long-term should be to leverage FFI
- non-local return, BlockClosure, missing fractions

Challenges - GPU/SQL/NoSQL

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- probably useful with limited semantics
- gain common environment, expressive syntax

Challenges - GPU/SQL/NoSQL

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- probably useful with limited semantics
- gain common environment, expressive syntax

Challenges - GPU/SQL/NoSQL

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- probably useful with limited semantics
- gain common environment, expressive syntax

Challenges - GPU/SQL/NoSQL

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- probably useful with limited semantics
- gain common environment, expressive syntax

Challenges - C/Wasm/native

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- garbage collection
- non-local return
- could create everything as `Object`
- for performance, need to avoid this
- BlockClosure, deficient numeric stack

Challenges - C/Wasm/native

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- garbage collection
- non-local return
- could create everything as `Object`
- for performance, need to avoid this
- BlockClosure, deficient numeric stack

Challenges - C/Wasm/native

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- garbage collection
- non-local return
- could create everything as `Object`
- for performance, need to avoid this
- BlockClosure, deficient numeric stack

Challenges - C/Wasm/native

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- garbage collection
- non-local return
- could create everything as `Object`
- for performance, need to avoid this
- BlockClosure, deficient numeric stack

Challenges - C/Wasm/native

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- garbage collection
- non-local return
- could create everything as `Object`
- for performance, need to avoid this
- BlockClosure, deficient numeric stack

Challenges - C/Wasm/native

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- garbage collection
- non-local return
- could create everything as `Object`
- for performance, need to avoid this
- BlockClosure, deficient numeric stack

Challenges - C/Wasm/native

- Smalltalk has dynamic/manifest types
- targets are fundamentally staticly typed
- garbage collection
- non-local return
- could create everything as `Object`
- for performance, need to avoid this
- BlockClosure, deficient numeric stack

Conclusions

- can leverage FFI
- can leverage transpilation
- for some applications don't need full semantics
- transpilation is quite popular - would be nice to create a common infrastructure
- contact me if you have ideas about what this would look like

Conclusions

- can leverage FFI
- can leverage transpilation
- for some applications don't need full semantics
- transpilation is quite popular - would be nice to create a common infrastructure
- contact me if you have ideas about what this would look like

Conclusions

- can leverage FFI
- can leverage transpilation
- for some applications don’t need full semantics
- transpilation is quite popular - would be nice to create a common infrastructure
- contact me if you have ideas about what this would look like

Conclusions

- can leverage FFI
- can leverage transpilation
- for some applications don’t need full semantics
- transpilation is quite popular - would be nice to create a common infrastructure
- contact me if you have ideas about what this would look like

Conclusions

- can leverage FFI
- can leverage transpilation
- for some applications don’t need full semantics
- transpilation is quite popular - would be nice to create a common infrastructure
- contact me if you have ideas about what this would look like

Questions?

@dmasonrose @pharojs