Non-Blocking Strategies for FFI

Don’t Block me Now!

Pablo Tesone
Pharo Consortium Engineer

Guille Polito
CNRS UMR9189
CRIStAL, Inria
RMoD

Guille Polito

CNRS Engineer
RMod Team

- Experience industrial on service-oriented and mobile applications.
- PhD in Computer Science
- Main research interests are modularity and development tools.
- In the Pharo community since 2010
- More noticeable contributions: Pharo Bootstrap process and Iceberg.

Pablo Tesone

Pharo Consortium
Engineer

- 10 years of experience in industrial applications
- PhD in Dynamic Software Update
- Interested in improving development tools and the daily development process.
- Enthusiast of the object oriented programming and their tools.

FFI? Foreign Function Interface

Image

VM

FFI

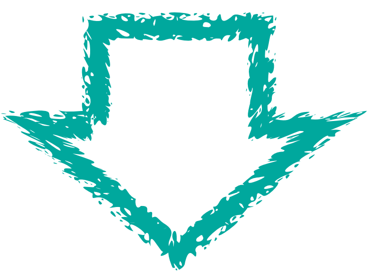
External Libraries

Operating System API

We can communicate with anything that has a C API

Unified FFI in a nutshell

```
#include <string.h>
void *memcpy(void *dest, const void *src, size_t n);
```

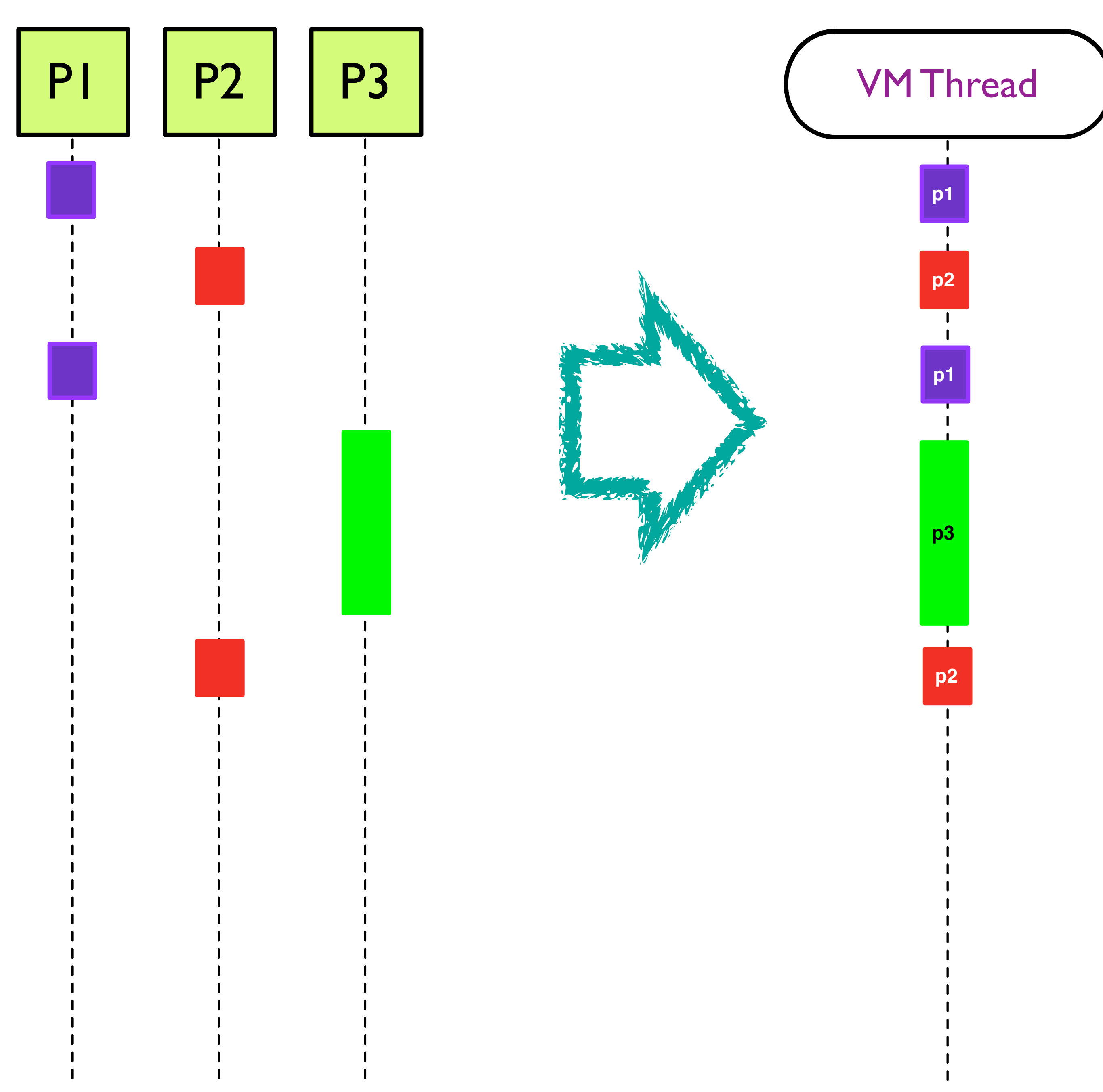


```
memCopy: src to: dest size: n
    ^ self ffiCall: #(void *memcpy(void *dest, const void *src, size_t n))
```

UFFI handles:

- Look-up of functions
- Marshalling of arguments
- Execution
- Marshalling of the return values

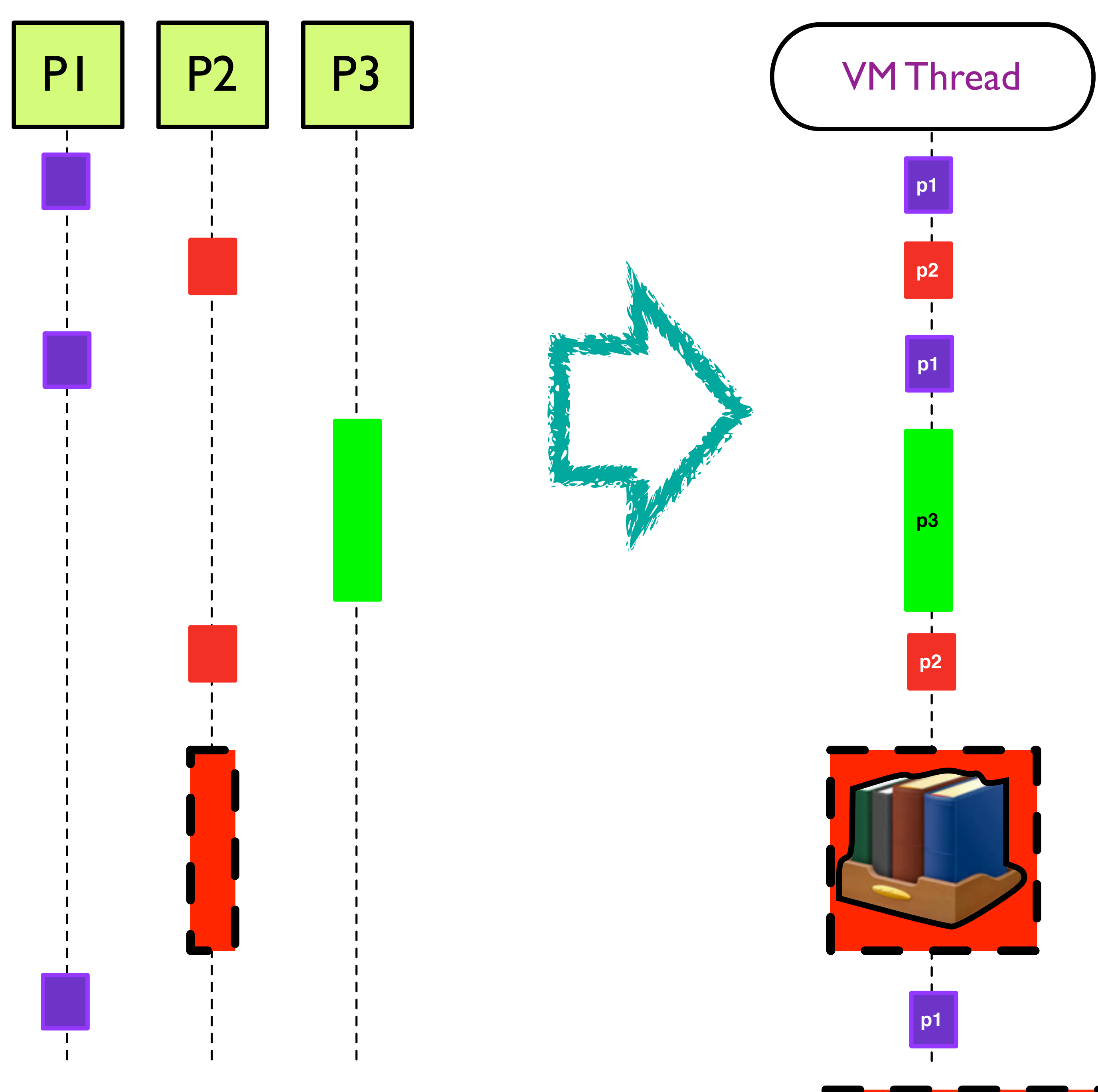
Concurrency in Pharo

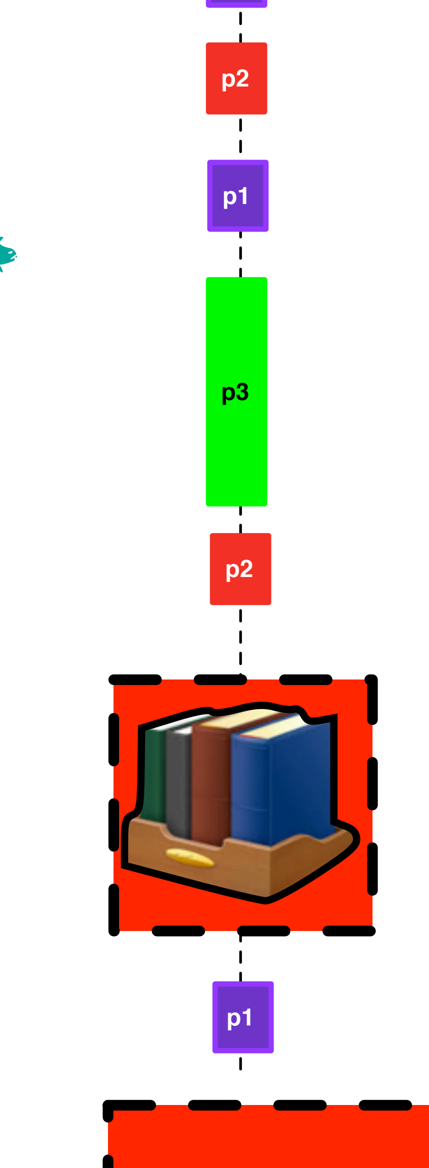


Concurrency in Pharo

VM handles process scheduling

Concurrency in Pharo





int function(char* foo, int bar)

Concurrency in Pharo

P1
P2
P3

VM Thread

p1
p2
p1
p3
p2

Out of VM

VM loses control

p1

```
int function(char* foo, int bar)
```

Conceptual Non-Blocking FFI

P1

P2

P3

VM Thread

p1

p2

p1

p3

p2

p1

p3

p2

?

`int function(char* foo, int bar)`

Strategy #1: Thread per Call-out

P1

P2

VM Thread

Thread 1

p1

p2

<<Spawn>>

p1

<<Signal Semaphore>>

p2

```
int function(char* foo, int bar)
```

Strategy #1: Thread per Call-out

- Simple

- Expensive to spawn threads
- Calls are not in the same thread
- Cannot reuse existing threads (e.g., UI threads)

Not all libraries are designed equally

- Different requirements
  - Must run in the main thread (Cocoa)
  - Must run in a single thread (Gtk+3)

  - Runs on any thread but not concurrent (libgit, sqlite)
  - Is a Thread-safe Library
  - ….

We need different Strategies to choose from

We need to choose different strategies for each library

Strategy #2: Worker Threads

P1

P2

VM Thread

Worker #1

p1

p2

p1

p2

<<Enqueue>>

Call out Queue
of Worker #1

<<Signal Semaphore>>

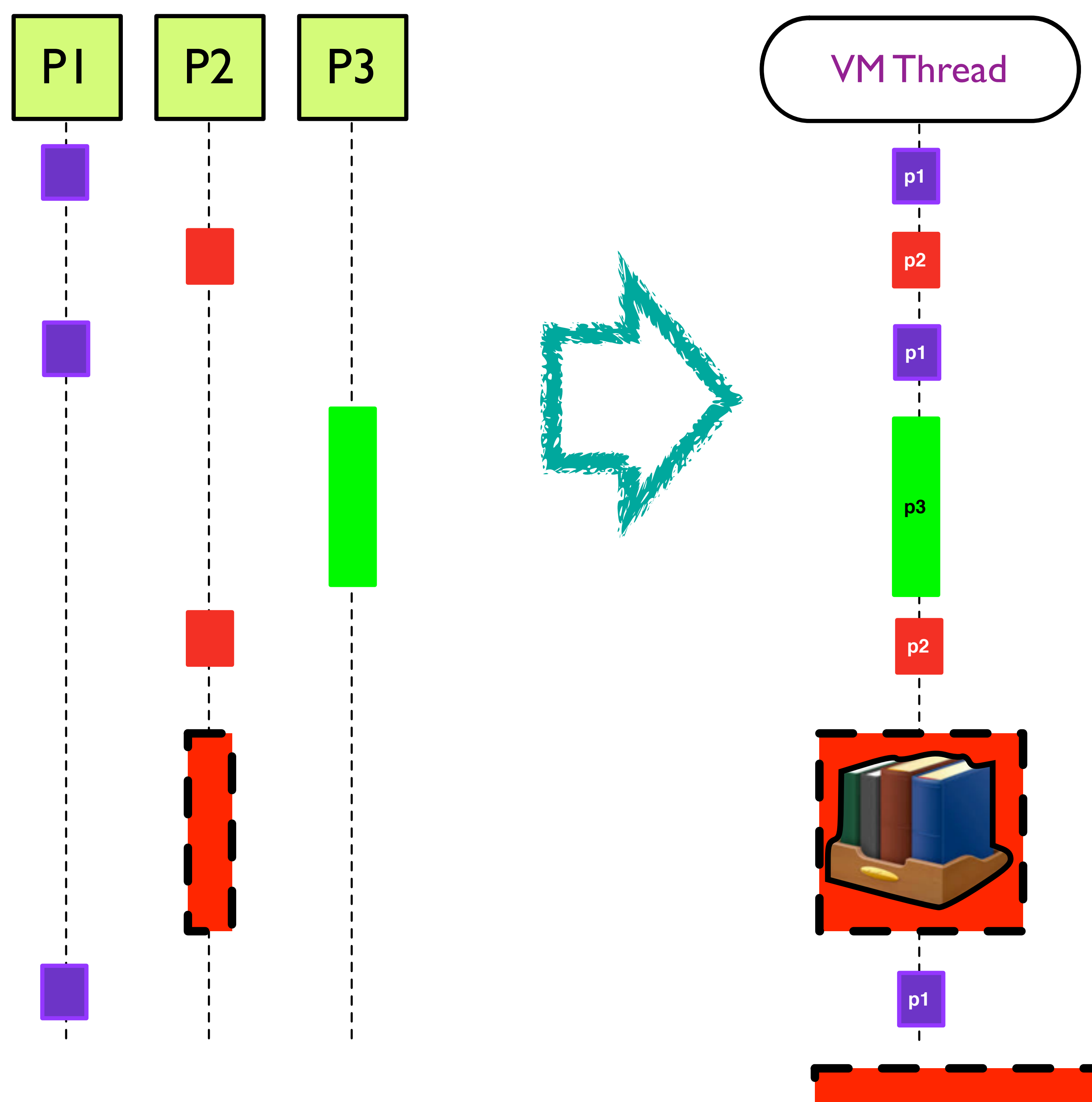
```
int function(char* foo, int bar)
```

Strategy #2: Worker Threads

- Simple
- Group related calls
- No thread spawn overhead

- Expensive Callouts (synchronising queue)
- Do not support main thread!

Strategy #3: VM Thread Runner




```
int function(char* foo, int bar)
```

Strategy #3: VM Thread Runner

- Simpler
- Group related calls
- No thread spawn overhead
- Backward compatibility

- Blocking

Strategy #4: Main Thread Runner

Strategy #4: Main Thread Runner

- Simple
- Group related calls
- No thread spawn overhead
- Supports main thread

- Expensive Callouts (synchronising queue)
- VM should be run in separate thread

Strategy #5: Global Interpreter Lock

P1

P2

VM Thread #1

VM Thread #2

```
int function(char* foo, int bar)
```

Strategy #5: Global Interpreter Lock

- Group related calls
- No thread spawn overhead

- No Backward compatibility
- Application should be written with threading in mind
- Requires VM modification

Strategy #6: Thread-safe interpreters

P1

P2

VM Thread #1

VM Thread #2

```
int function(char* foo, int bar)
```

Strategy #6: Thread-safe interpreters

- Real multithreading not only for FFI

- Does not exist for Pharo
- Requires extensive modification of VM, Plugins and Image core libraries
- Application should be written with threading in mind

NOT YET AVAILABLE

Implementations

Queue Based FFI
(Pharo Threaded FFI Plugin)

Strategy #1: Thread per Call-out

Strategy #2: Worker Threads

Strategy #3: VM Thread Runner

Strategy #4: Main Thread Runner

GILda VM
(Global Interpreter Lock VM)

Strategy #5: Global Interpreter Lock

Future???

Strategy #6: Thread-safe interpreters

Implementations

Queue Based FFI
(Pharo Threaded FFI Plugin)

Strategy #1: Thread per Call-out

Strategy #2: Worker Threads

Strategy #3: VM Thread Runner

Strategy #4: Main Thread Runner

GILda VM
(Global Interpreter Lock VM)

Strategy #5: Global Interpreter Lock

Future???

Strategy #6: Thread-safe interpreters

Transparency through UFFI

Your Program

Unified FFI

Squeak FFI | Pharo Threaded FFI

Squeak FFI Plugin | Pharo Threaded FFI Plugin

S#1 S#2 S#3 S#4

Library

Decision Table

	Same Thread	Worker Threads
Long Calls	Blocking	Parallel
Short Calls	Little Overhead	Lots of Overhead

WORK IN PROGRESS

OPTIMIZATIONS

Implementations

Queue Based FFI
(Pharo Threaded FFI Plugin)

Strategy #1: Thread per Call-out

Strate

Strategy

Strategy

GILda VM
(Global Interpreter Lock VM)

Strategy #5: Global Interpreter Lock

More Details

IWST - Virtual Machines Session

Thursday 11:00 am

Future???

Strategy #6: Thread-safe interpreters

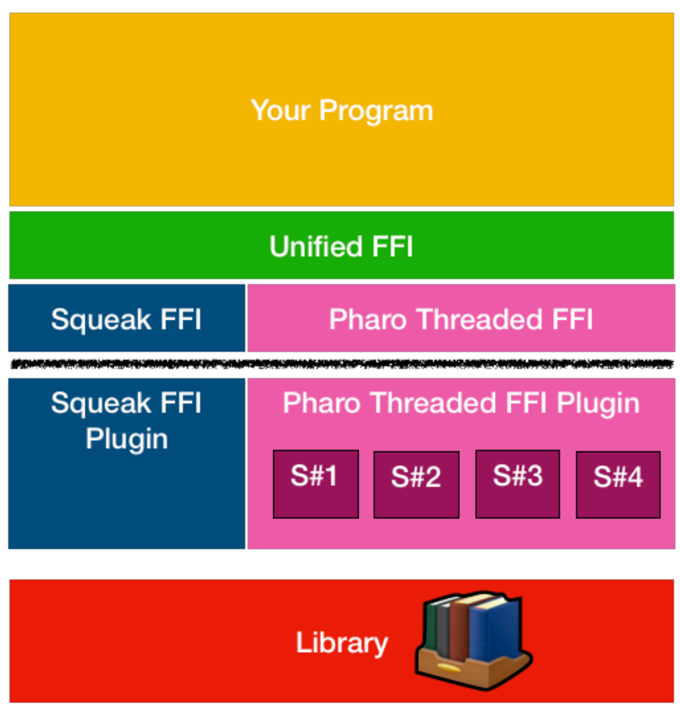
Start Using It!

```
Metacello new
   baseline: ‘ThreadedFFI'
   repository:
       ‘github://pharo-project/threadedFFI-Plugin/src';
   load
```

Only in Pharo 8 + Headless VM

Conclusion

- Beta Version (In usage for Gtk+)
- Transparent for the user
- Strategies selected in the Image
- Uses LibFFI
- Re-entrant Callback support
- Tests!



Preliminar results

- All marshalling image side + Lib FFI
- Short call
  - Same thread 27 us
  - Single worker thread 6791 us
- 2 Parallel long Calls (1 second per call)
  - Same Thread 2001.9 ms
  - Different working threads 1006.4ms