GildaVM:
a Non-Blocking I/O Architecture for the Cog VM

Pablo Tesone
Pharo Consortium

Guille Polito
CNRS UMR9189
CRIStAL, Inria
RMoD

Eliot Miranda
Stellect Systems Inc

David Simmons
The Light Phone, USA

Université de Lille

Inria — inventors for the digital world

cnrs

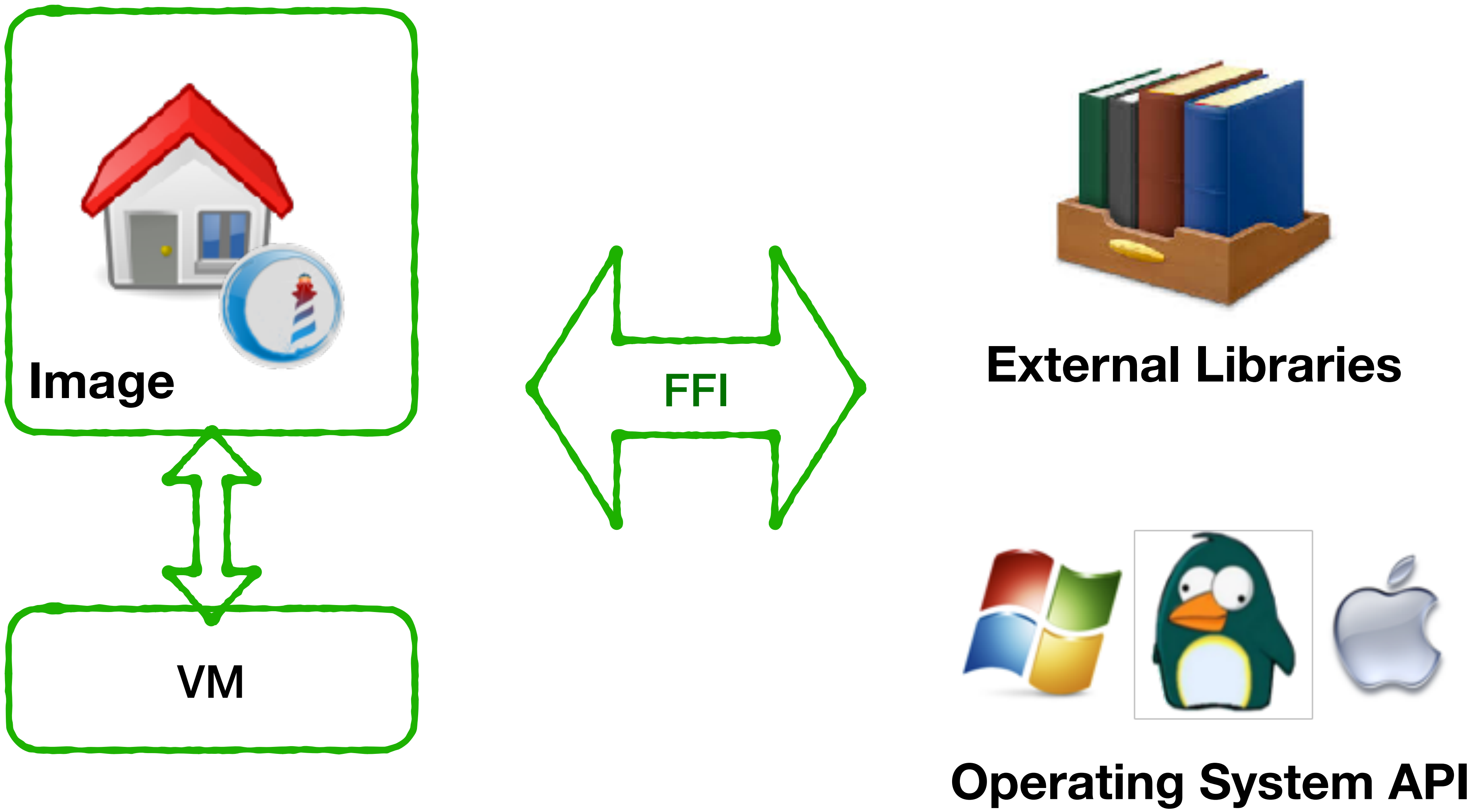
CRIStAL — Centre de Recherche en Informatique, Signal et Automatique de Lille

RMod

Blocking I/O

- I/O execution blocks the interpreter
- While in a I/O call the interpreter is blocked
- E.g., System-calls, FFI

FFI? Foreign Function Interface



We can communicate with anything that has a C API

Unified FFI in a nutshell

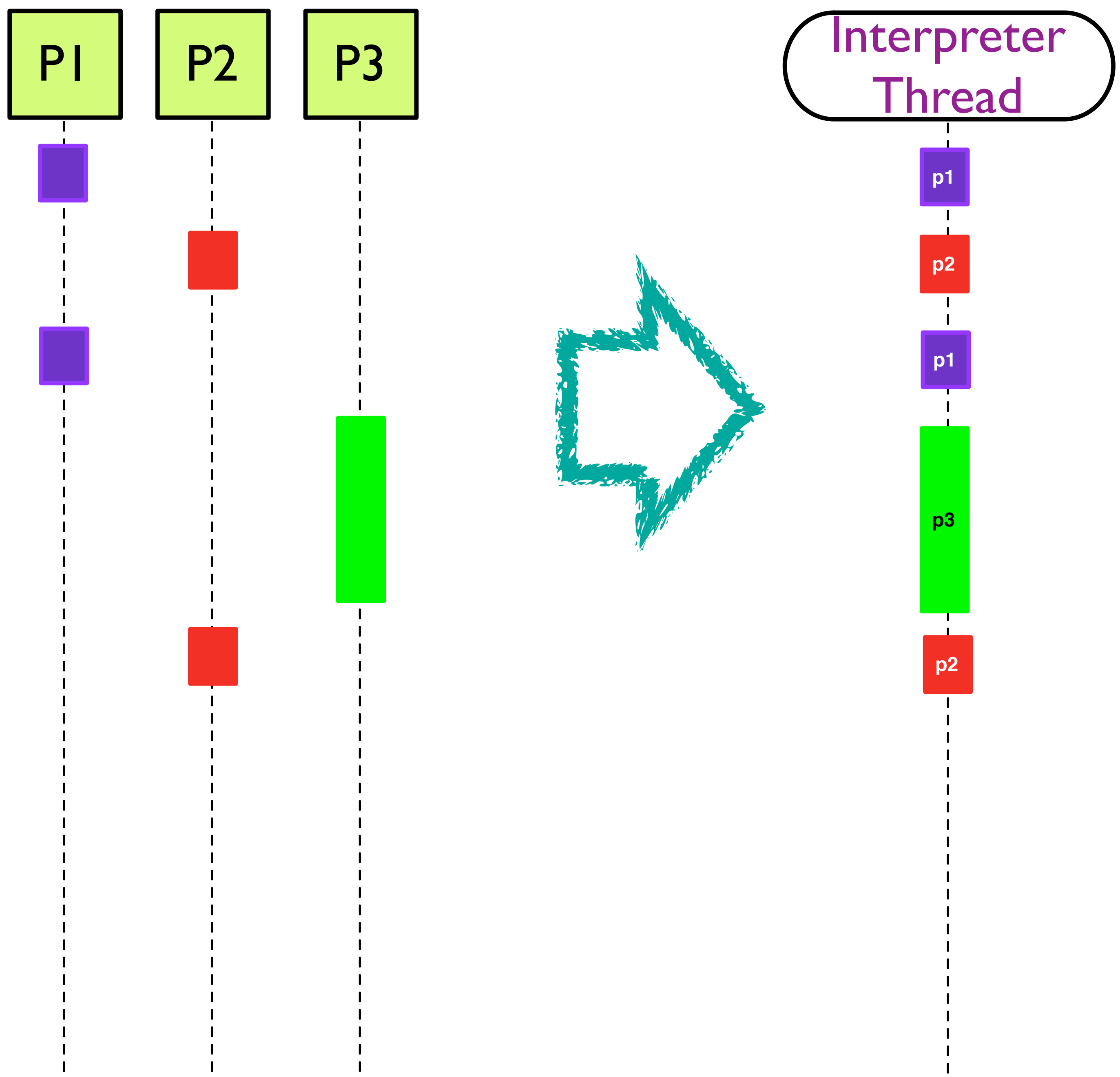
```
#include <string.h>
void *memcpy(void *dest, const void *src, size_t n);
```

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

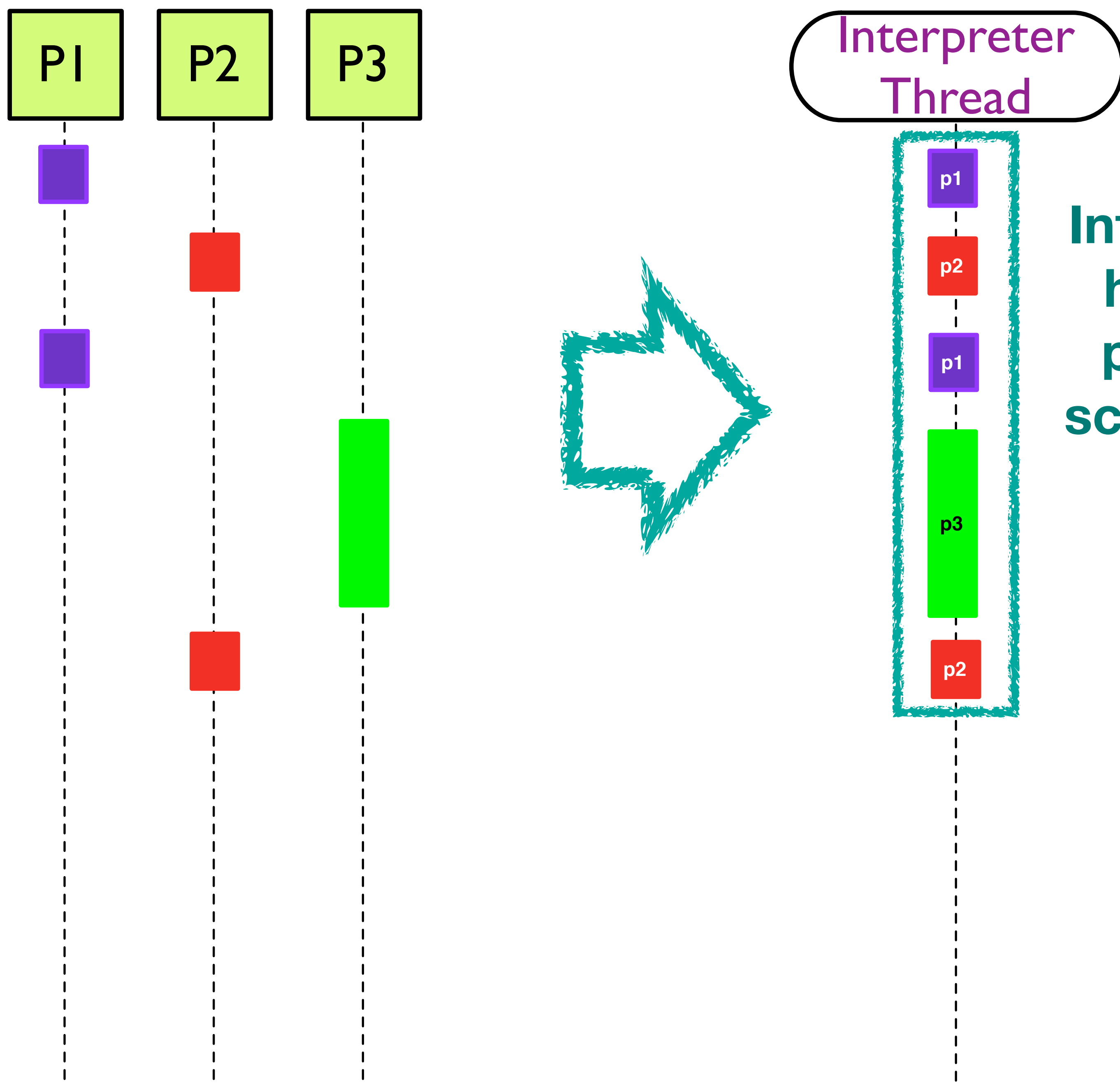
UFFI handles:

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

Concurrency in Pharo

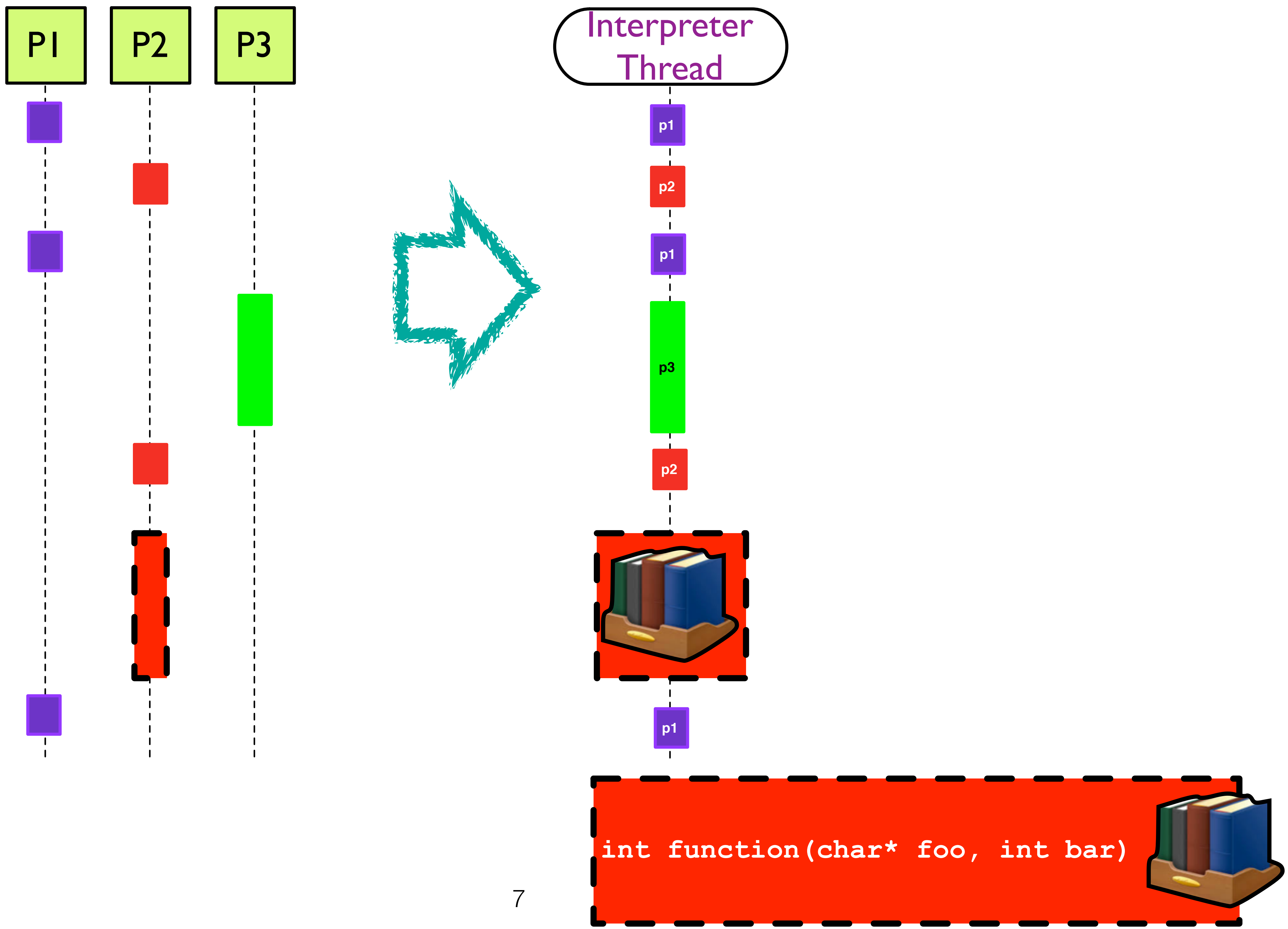


Concurrency in Pharo

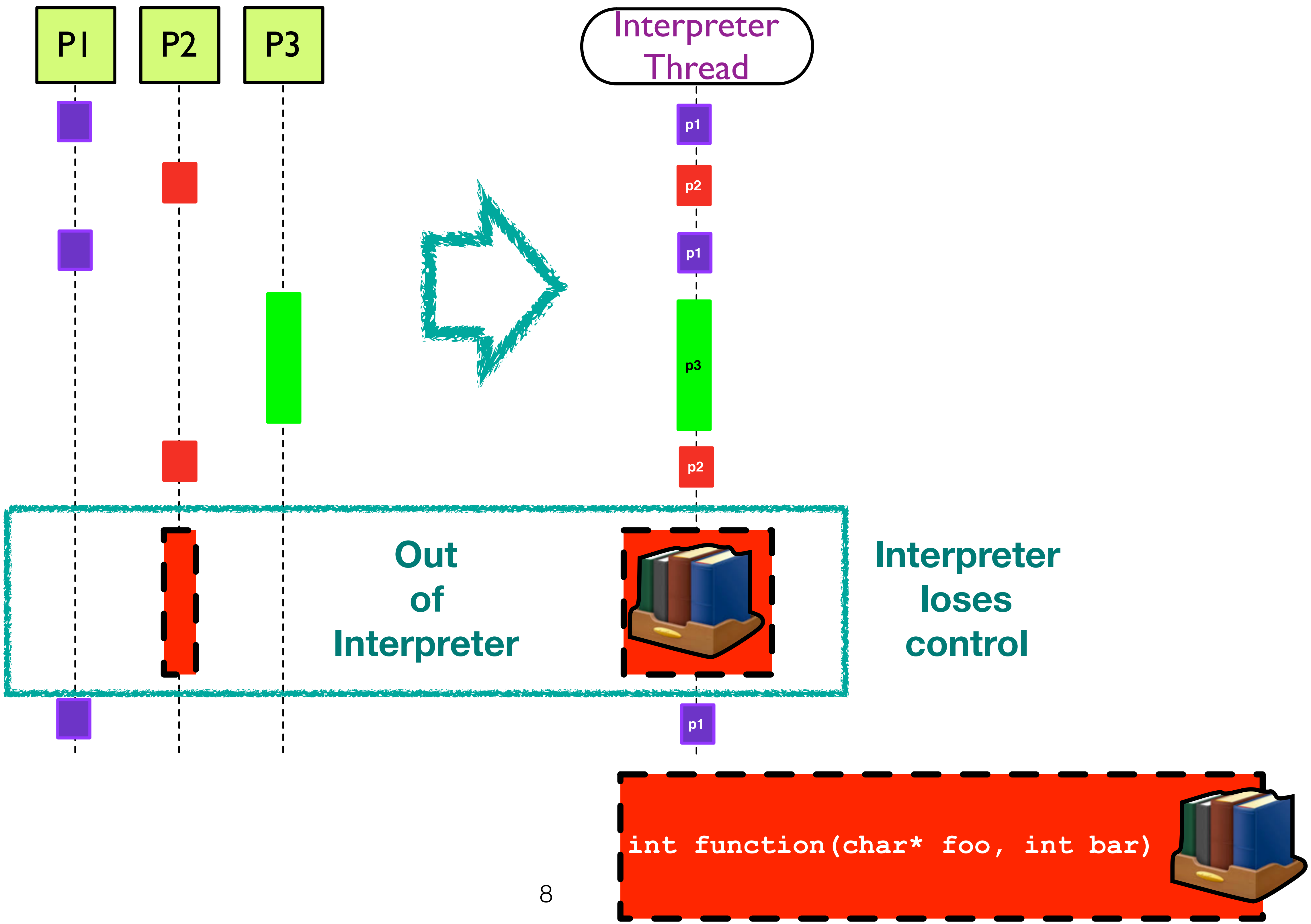


Interpreter handles process scheduling

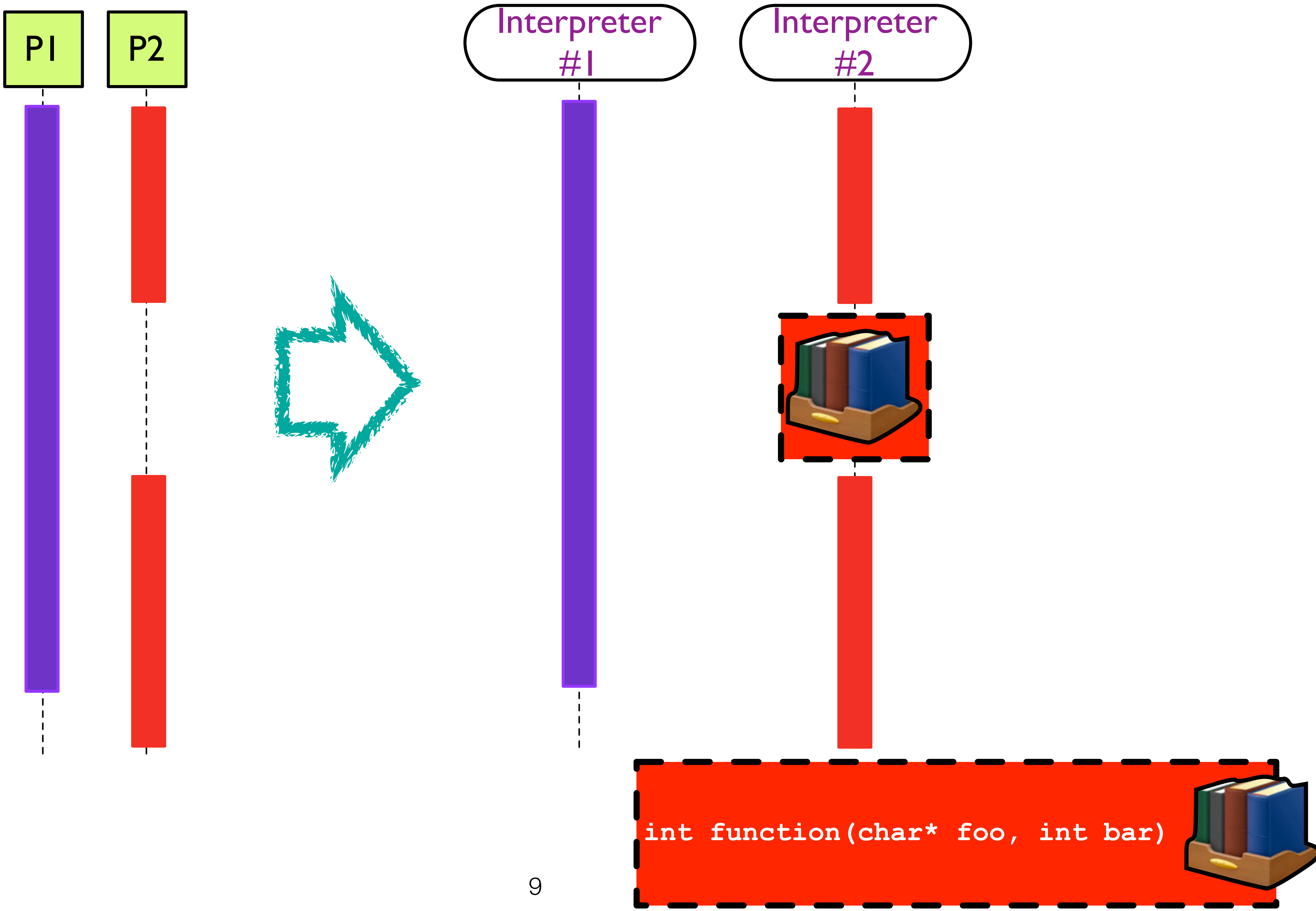
Concurrency in Pharo



Concurrency in Pharo



What we want!

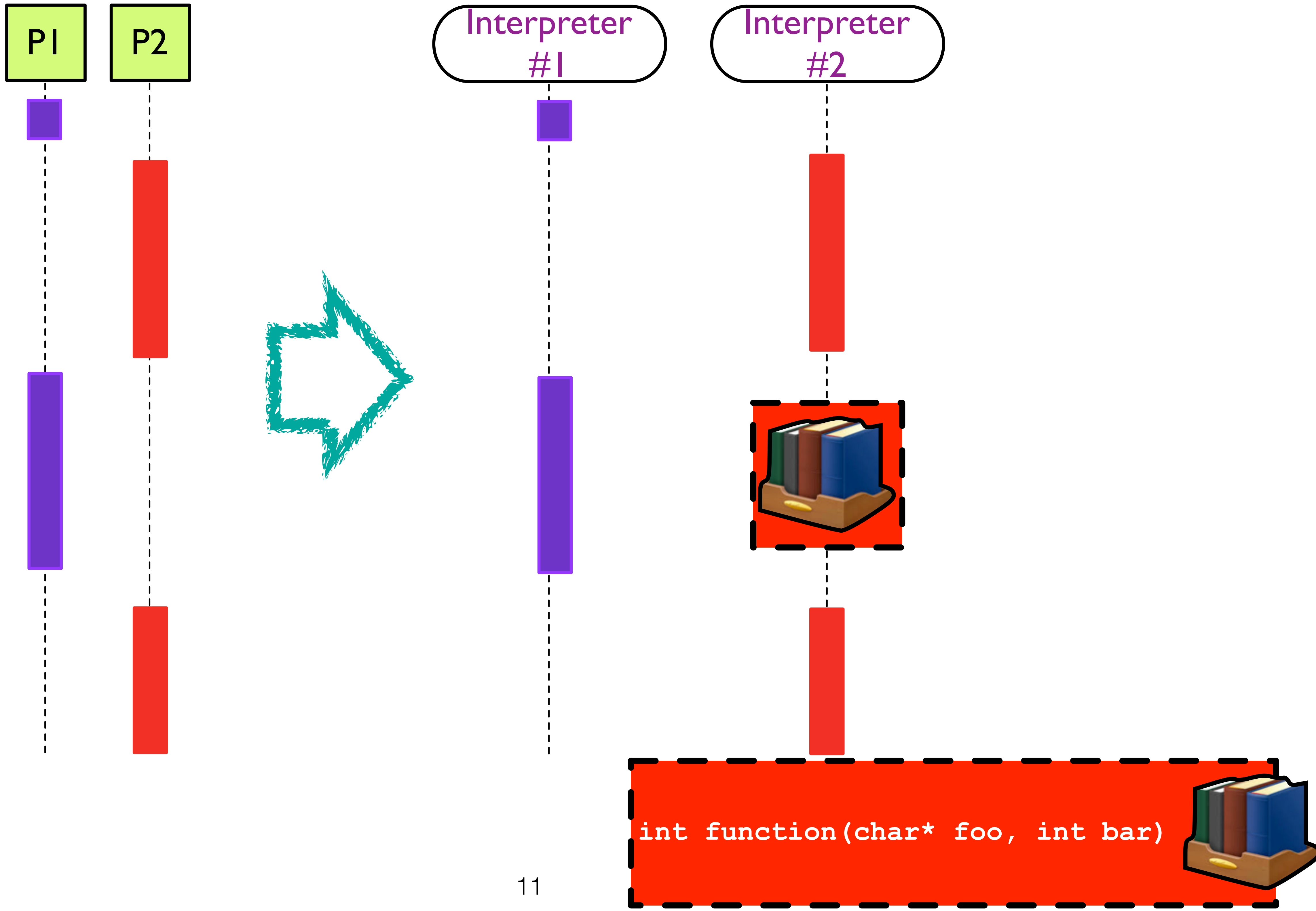

P1
P2
Interpreter #1
Interpreter #2
int function(char* foo, int bar)

What we want!

- Real multithreading not only for FFI

- Requires extensive modification of VM, Plugins and Image core libraries
- Applications should be written with threading in mind

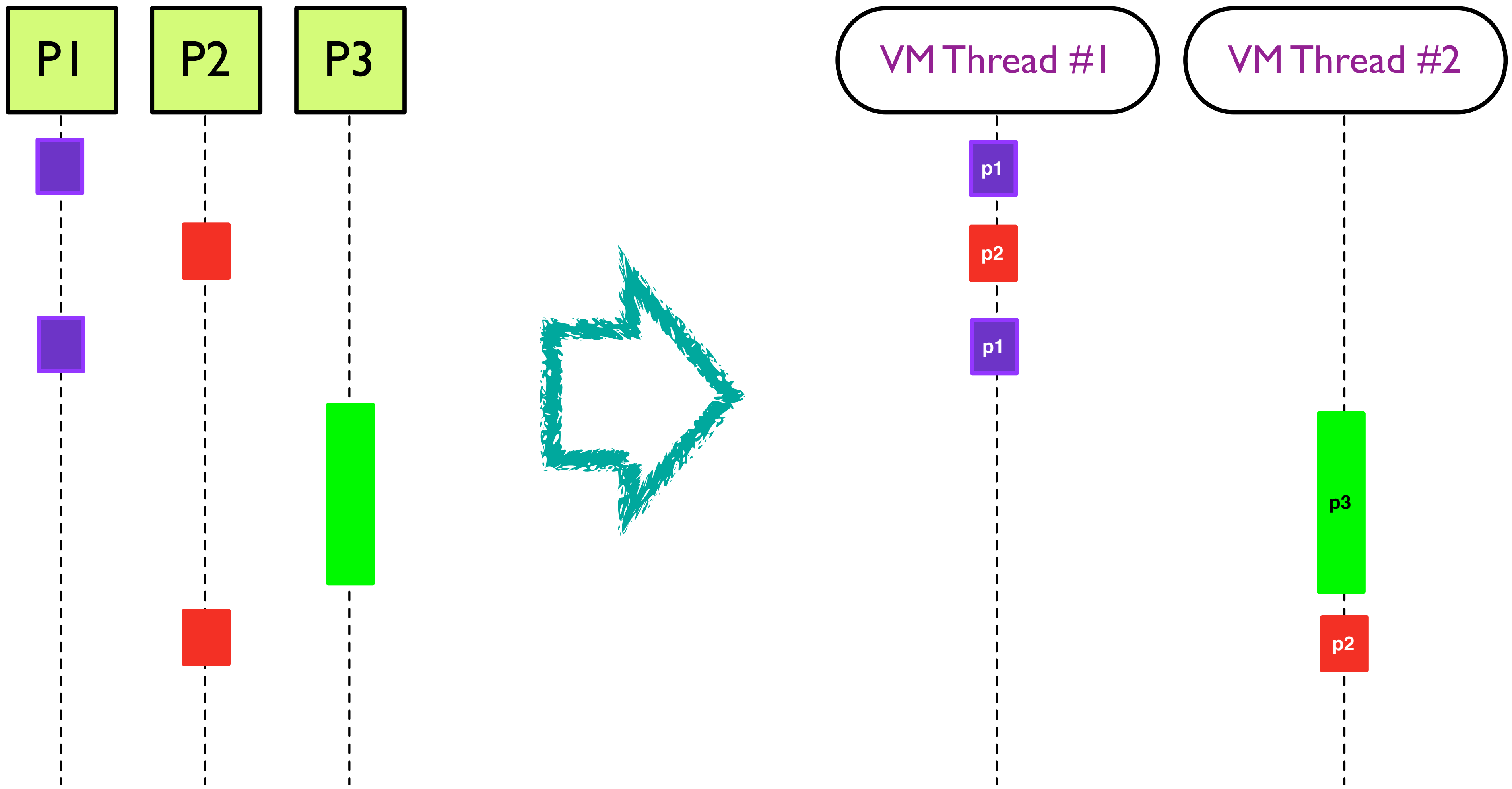
Proposal: Global Interpreter Lock VM



Research Questions

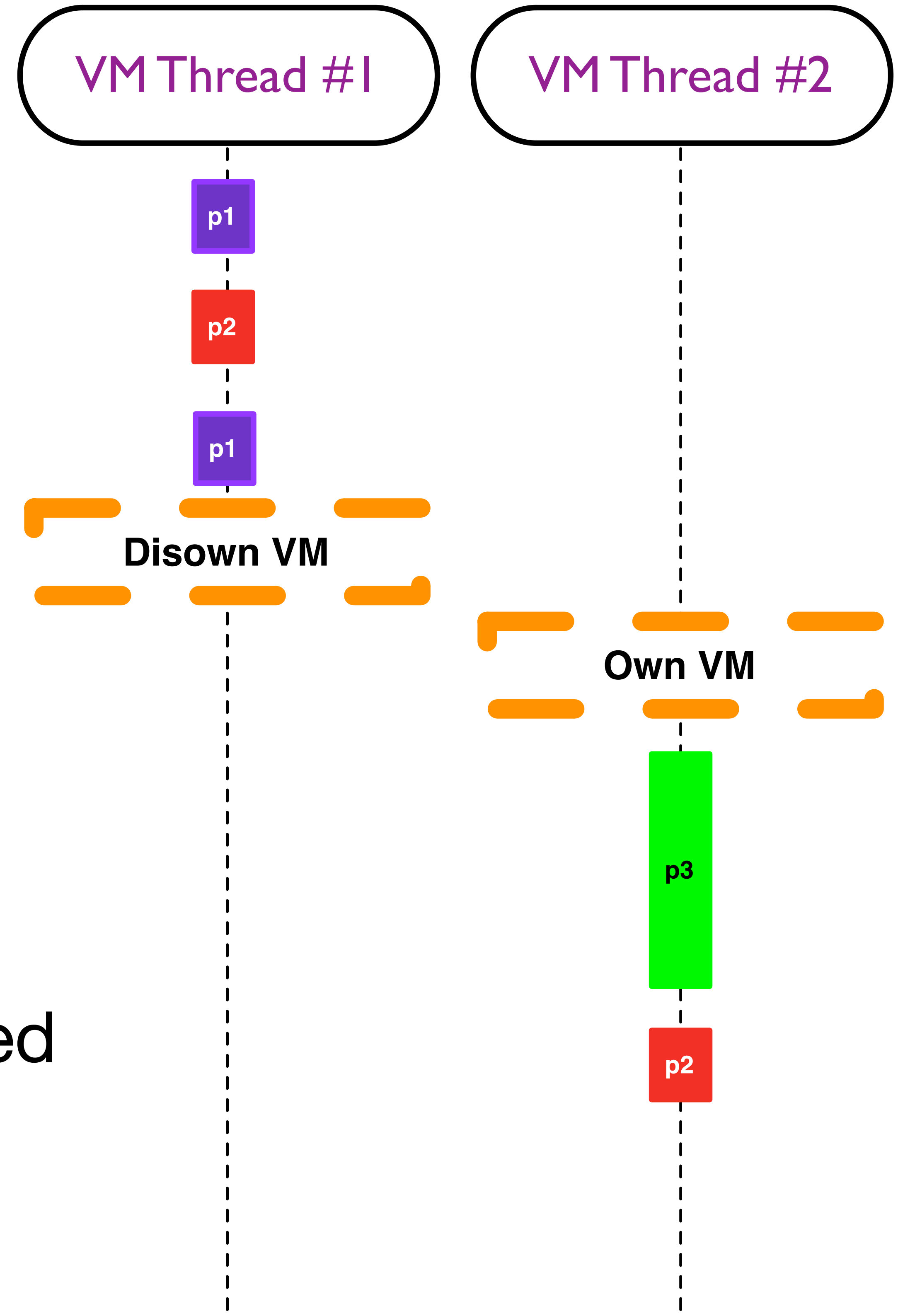
- RQ1: How does scheduling work in presence of processes and native threads?
- RQ2: What is the overhead of thread switching?

Process scheduling with many VM threads



One VM Thread owns the VM at each time

Process Affinity

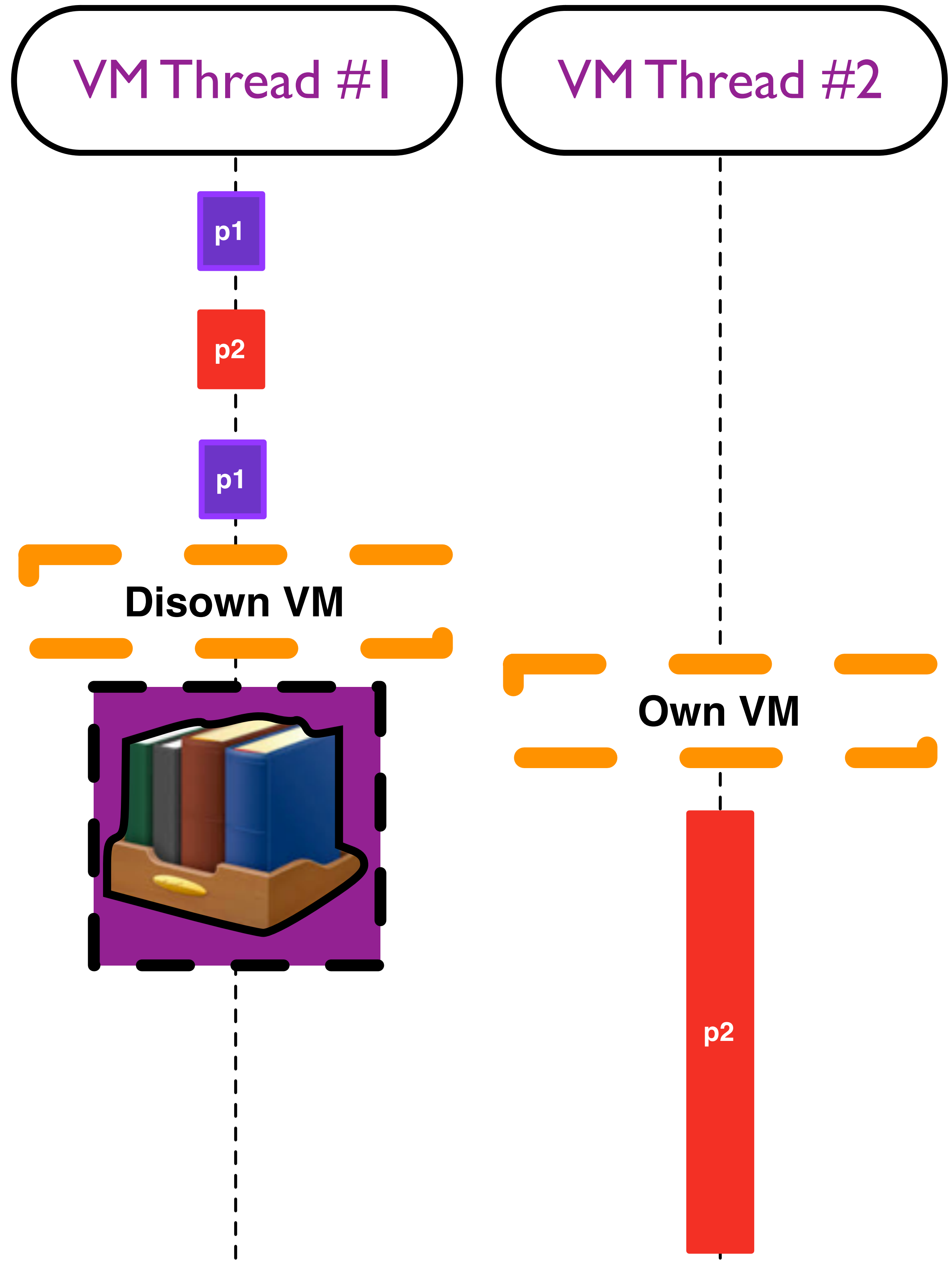


`p3 bindToThreadId: 2`

- Explicit binding
- When a process is activated, it is run in the affined thread
- Or in the same thread if not affined

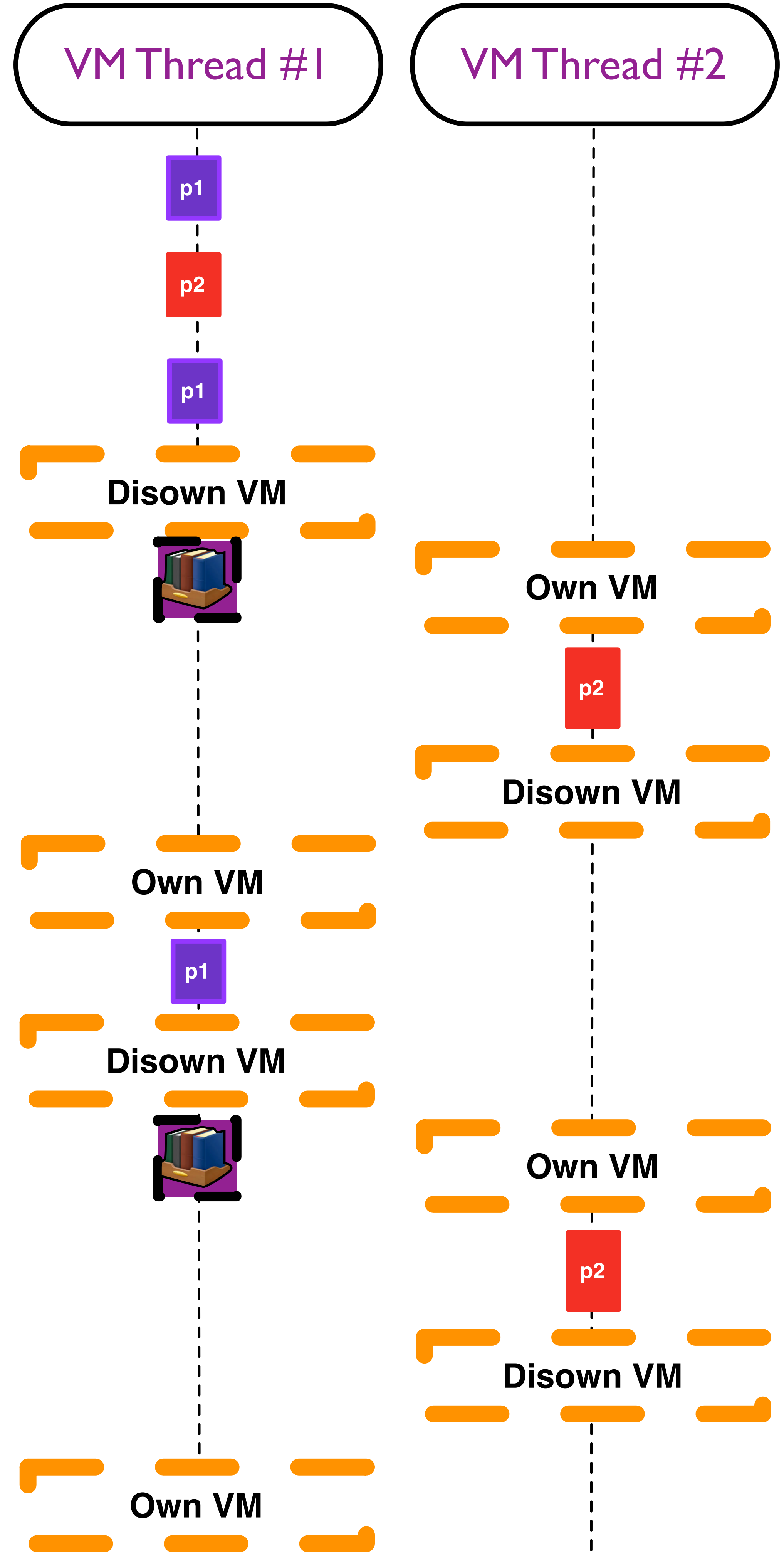
Non-blocking FFI

- Before FFI calls the current thread disowns the VM
- Another thread owns the VM
- Non-blocked processes are scheduled



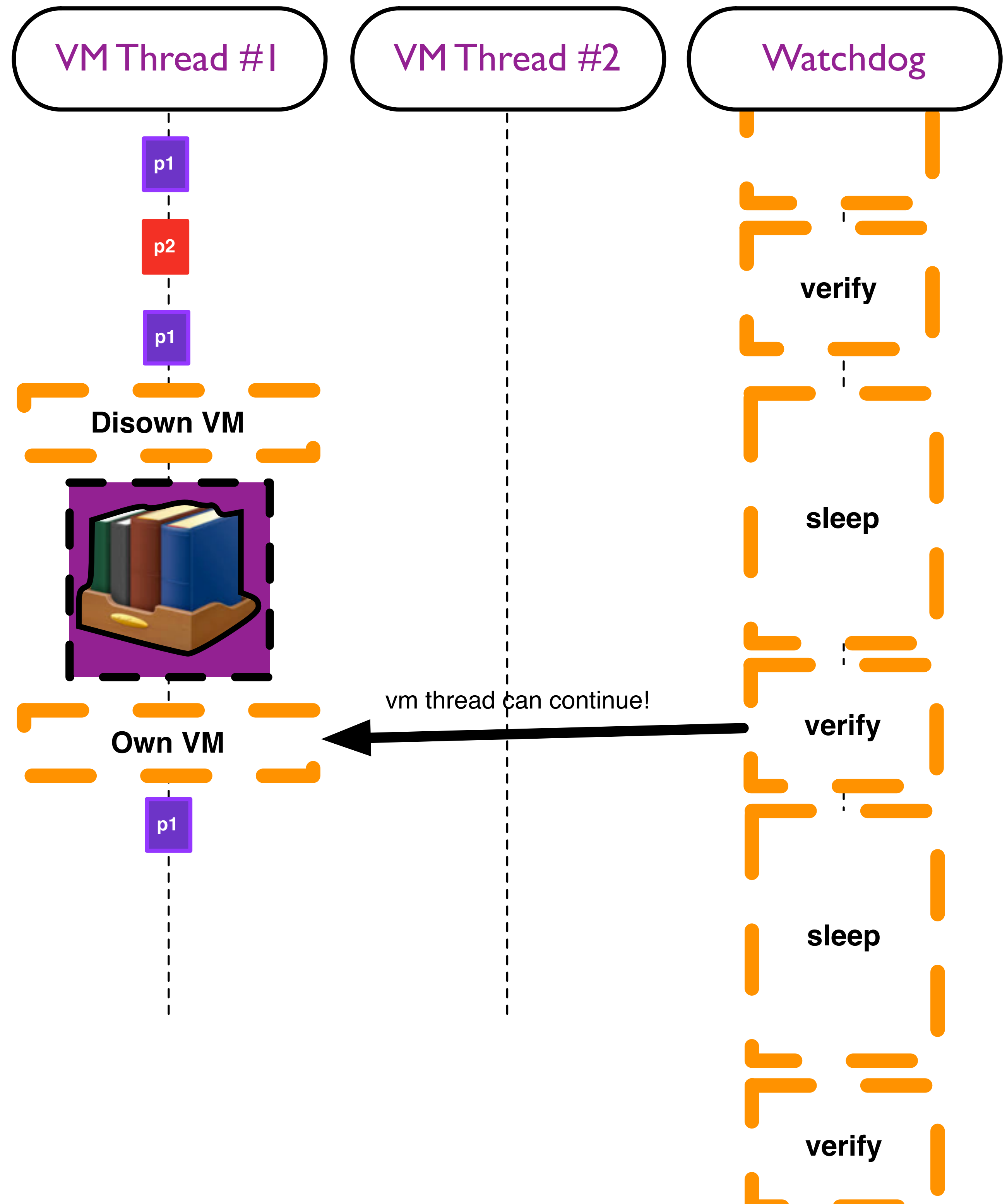
Short callouts?

- If naive, each disown creates a lot of overhead!



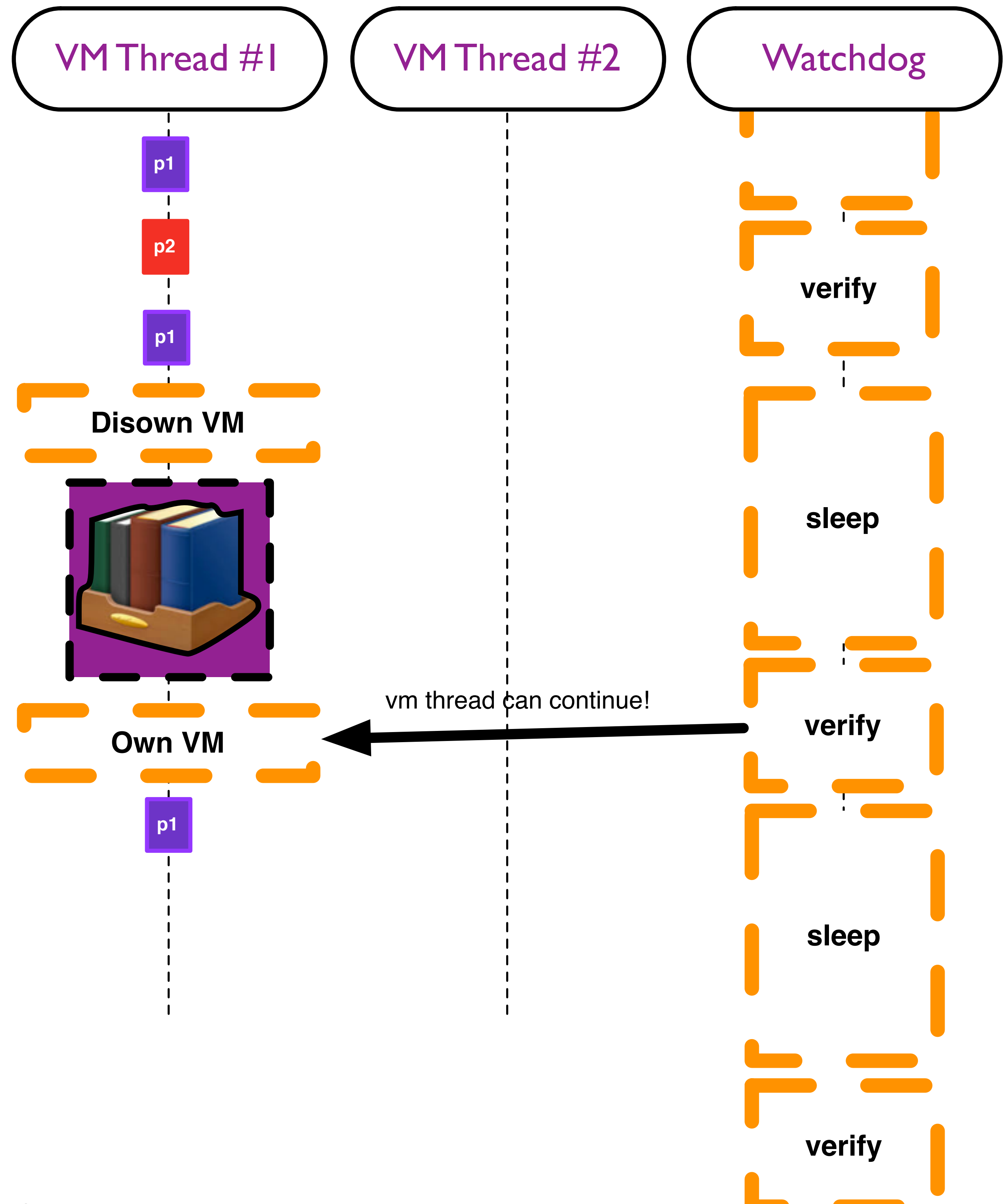
Watchdog Native Thread

- A watchdog periodically verifies if the VM is busy
- If idle, selects a thread with work to do and activate it



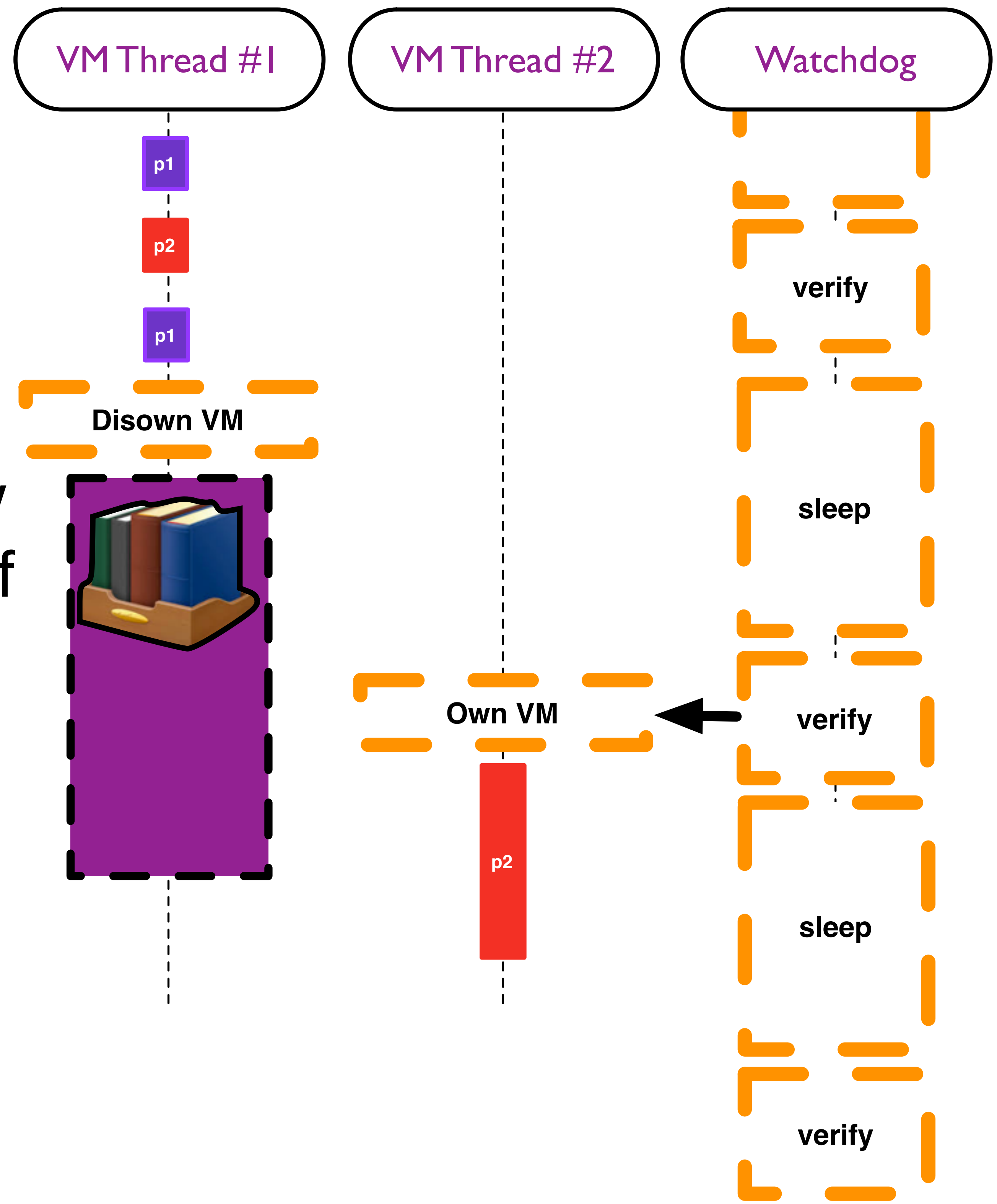
Short calls

- The watchdog sleeping window defines the “length” of the short call



Long call preemption

- The watchdog sleeping window also defines the max “length” of idle-ness



Process switch without affinity

Benchmark	Stock VM (Avg.)	Modified VM (Avg.)
Same Priority without yielding	1784 ms	1784 ms
Same Priority with yielding	1786 ms	1800 ms
Different priorities	1783 ms	1784 ms

Table 2. Comparison of the Execution of Smalltalk code

50 iterations, mean showed

Long I/Os

2 one-sec callouts

Benchmark	Stock VM (Avg.)	Modified VM (Avg.)
sequencial	2001 ms	2003 ms
concurrent processes	2006 ms	1216 ms

Table 3. Comparison of the Execution of Long callouts

50 iterations, mean showed

Short calls

100,000 short callouts

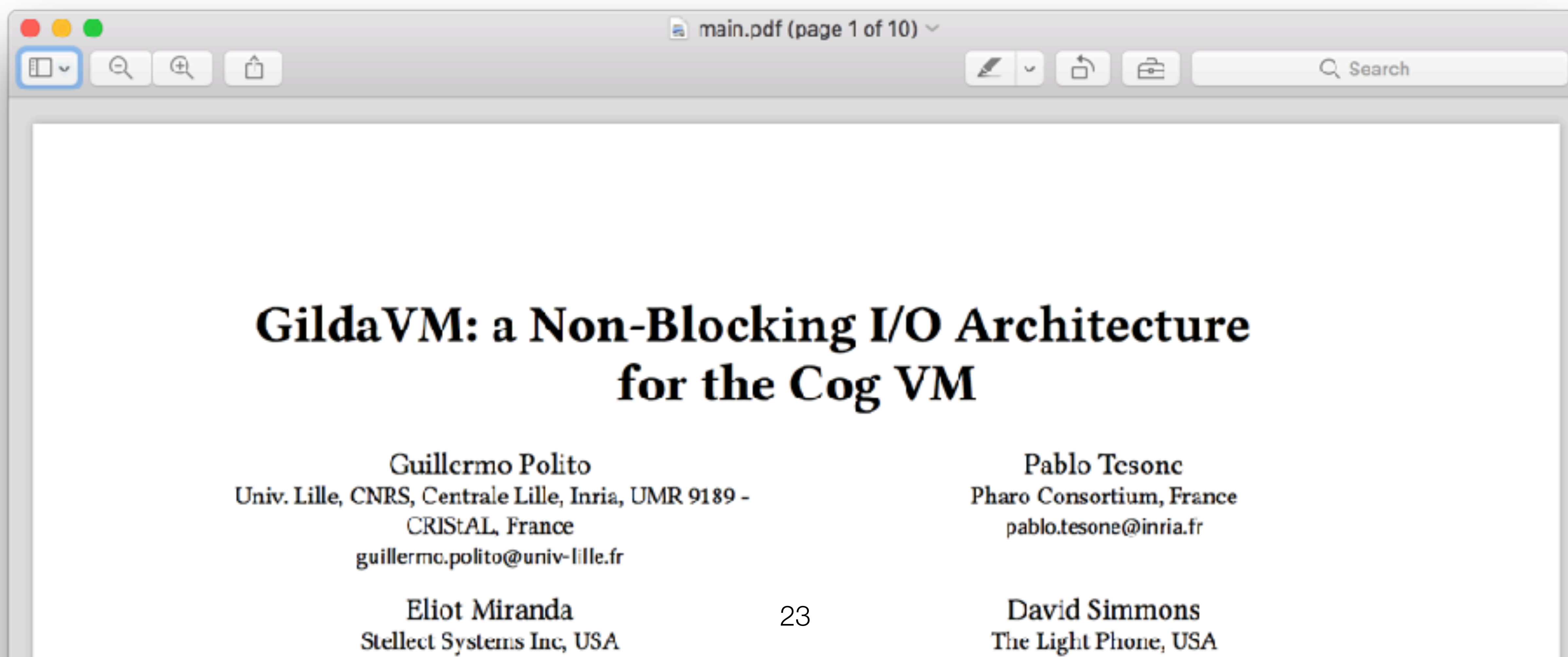
Benchmark	Stock VM (Avg.)	Modified VM (Avg.)
sequencial	63	88 ms
concurrent processes	60	995 ms

Table 4. Comparison of the Execution of Short callouts

50 iterations, mean showed

Also in the paper…

- Callbacks
- Reentrant callbacks
- More on preemption
- Implementation details



GildaVM: a Non-Blocking I/O Architecture for the Cog VM

Guillermo Polito
Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL, France
guillermo.polito@univ-lille.fr

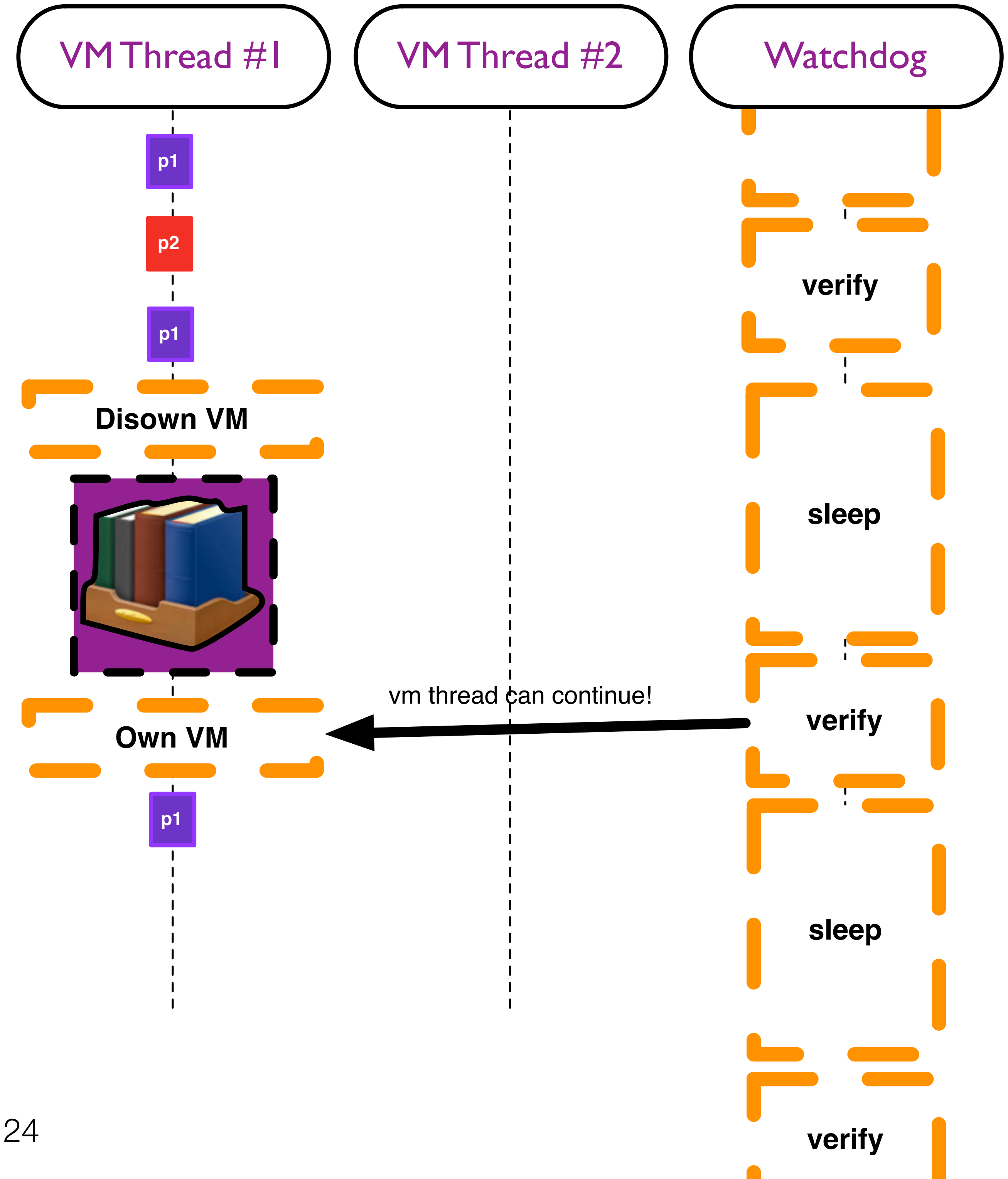
Pablo Tesone
Pharo Consortium, France
pablo.tesone@inria.fr

Eliot Miranda
Stellect Systems Inc, USA

David Simmons
The Light Phone, USA

Future Work #1: watchdog impact

- If the watchdog window is not aligned with the FFI calls, short callouts are recognised as long ones (false positives)
- Long watchdog window will recognise long calls as short calls and be blocking (false negatives)



Future Work #2: thread management

- Should VM threads be created implicitly or explicitly?
- Should the thread pool be size-bound? Analyse strategies for particular applications.

Conclusion

- A Global interpreter lock architecture for green-threaded smalltalk implementations
- Good for parallelising long blocking I/O
- Some strategies to reduce the overhead of thread switch

Pablo Tesone
Pharo Consortium

Guille Polito
CNRS UMR9189
CRIStAL, Inria
RMoD

Eliot Miranda
Stellect Systems Inc

David Simmons
The Light Phone, USA