# FFI

The good, the bad and the ugly

Esteban Lorenzano

*(The Pharo firefighter)*

## Current status of FFI:

*A mess :(*

Several options, none of them very clear.

Three options:
FFI, AlienFFI, NB-FFI

## FFI

- Basic types and not much more
  - you can declare your own types (but nobody knows how, anyway)
- No callbacks
  - There is a version of Eliot with callback support, but still not integrated into Pharo.

# FFI Characteristics

- Pros
  - Simple
- Cons
  - No callbacks (yet)
  - cdecl/apicall, module lookup

## AlienFFI

- Object per method
- GC of external memory allocations
- Callbacks
- Uses same primitives as FFI

# AlienFFI Characteristics

- Pros
  - Object-oriented approach
  - Powerful
  - Nice callback implementation
- Cons
  - More complex to use than plain FFI
  - Abandoned to a enhanced version of FFI (It should be considered legacy)
  - A bit slower

## NativeBoost FFI

- Primitive call + binary code generation magic
- Nice function call and type definition
- Callbacks (though slower than Alien)
- Uses assembly

# NB-FFI Characteristics

- Pros
  - all-in-image approach
  - nice syntax declaration
  - fast
- Cons
  - no platform independence (no ARM, no x86_64, etc.)
  - each new platform needs a new ASMJIT
    - different assembly knowledge (and well… assembly knowledge in general)
    - VirtualCPU can help here, but you still need to know the platform architecture

## How to fix the mess?

- We need ONE solution that works in all cases.
- Sadly, NativeBoost requires a lot of effort that we cannot spend on it.
- Happily, there is an existing FFI implementation, maintained by Eliot, that we can use.
- And we can take parts from NativeBoost too! (like the syntax declaration)

## WIP

- NB-FFI to FFI
- ThreadFFI
- uFFI

## WIP: NB-FFI to FFI

- Replaces ASM generated part with plain FFI primitive calls.
- Portable (to ARM, x86_64, etc.)
- No executable memory (can be used in scenaries like iPhone or with security constraints)
- No need to know assembly to maintain it (yes, that’s a pro for me ;)
- ASMJIT will be pluggable and still usable

## WIP: ThreadFFI

- It will allow us to execute expensive foreign calls (i.e. SQL queries) in a separate native thread, and callback the system when finished.

## WIP: uFFI

- Specific bytecodes (pointer allocation, primitive types, function calling)
    - Fast due direct memory manipulation and Cog JIT
- Common interface for different backends (it will use NB syntax, too)

## Summary

- One FFI that will work on all platforms is arriving (it will be in Pharo 4 or early Pharo 5)
- Threaded FFI will come soon after
- uFFI, with important enhancements will arrive some time later