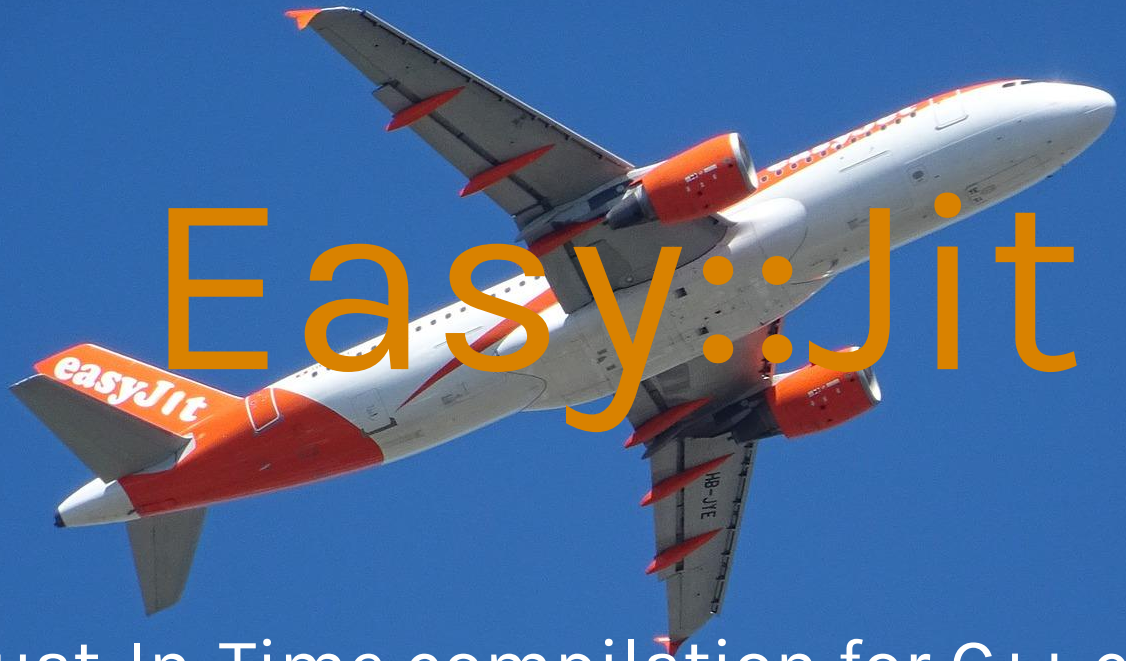



# Easy::Jit

Just-In-Time compilation for C++ codes

Serge Guelton
**Juan Manuel Martinez Caamaño** (me)

from **Quarkslab**

# Introduction

- ✔ Compiler-assisted library for runtime code generation
  - Easy to understand C++ wrapper around the LLVM
- ✘ An omniscient virtual machine
- ✘ Read-Eval-Print Loop
- ✘ Building blocks for a Just-in-Time compiler

# Easy::Jit: by example

```
static void apply_filter(const char *mask, unsigned mask_size, unsigned mask_area,
                         cv::Mat &image, cv::Mat *&out) {

    kernel(mask, mask_size, mask_area,
           image.ptr(0,0), out->ptr(0,0),
           image.rows, image.cols, image.channels());
}
```

# Easy::Jit: by example

```
static void apply_filter(const char *mask, unsigned mask_size, unsigned mask_area,
                         cv::Mat &image, cv::Mat *&out) {

    kernel(mask, mask_size, mask_area,
           image.ptr(0,0), out->ptr(0,0),
           image.rows, image.cols, image.channels());
}
```

# Easy::Jit: by example

```
static void apply_filter(const char *mask, unsigned mask_size, unsigned mask_area,
                         cv::Mat &image, cv::Mat *&out) {

    kernel(mask, mask_size, mask_area,
           image.ptr(0,0), out->ptr(0,0),
           image.rows, image.cols, image.channels());
}
```

# Easy::Jit: by example

```
#include <easy/jit.h>

static void apply_filter(const char *mask, unsigned mask_size, unsigned mask_area,
                         cv::Mat &image, cv::Mat *&out) {
   using namespace std::placeholder;

   auto callme = easy::jit(kernel, mask, mask_size, mask_area,
                           _1, _2, image.rows, image.cols, image.channels());
    callme(image.ptr(0,0), out->ptr(0,0));
}
```

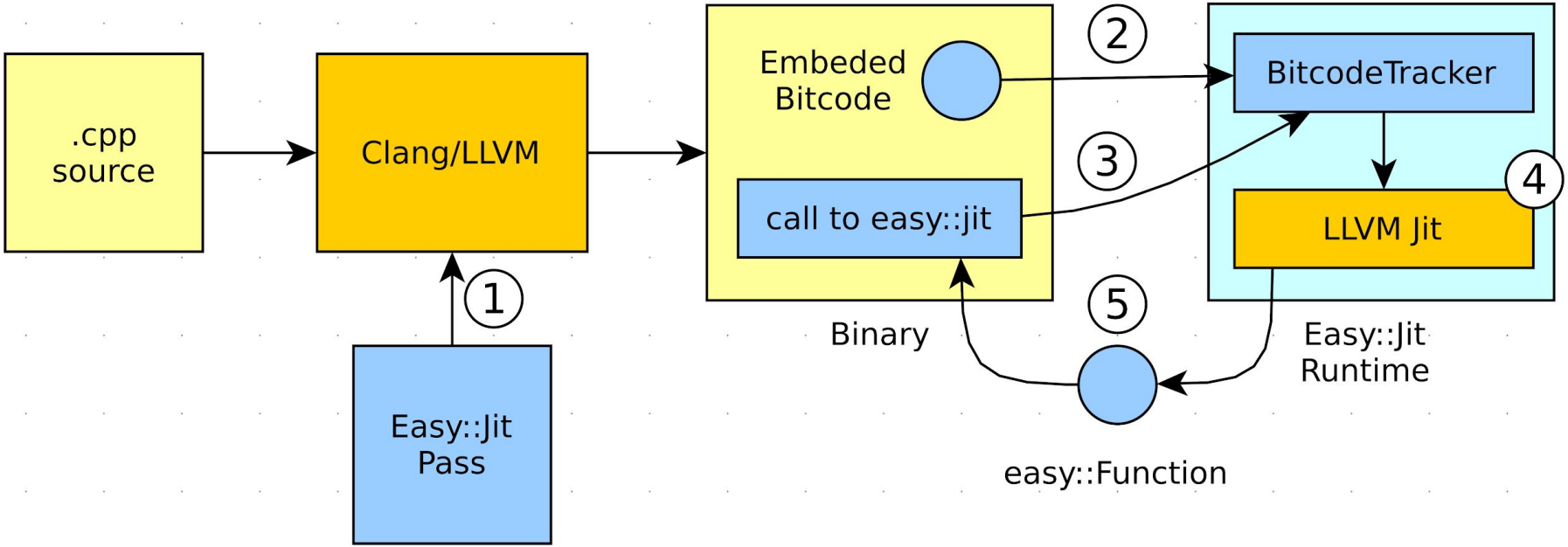
# Easy::Jit: by example

```
#include <easy/code_cache.h>

static void apply_filter(const char *mask, unsigned mask_size, unsigned mask_area,
                         cv::Mat &image, cv::Mat *&out) {
   using namespace std::placeholder;
   static easy::Cache<> cache;
   auto const& callme = cache.jit(kernel, mask, mask_size, mask_area,
                                  _1, _2, image.rows, image.cols, image.channels());
    callme(image.ptr(0,0), out->ptr(0,0));
}
```

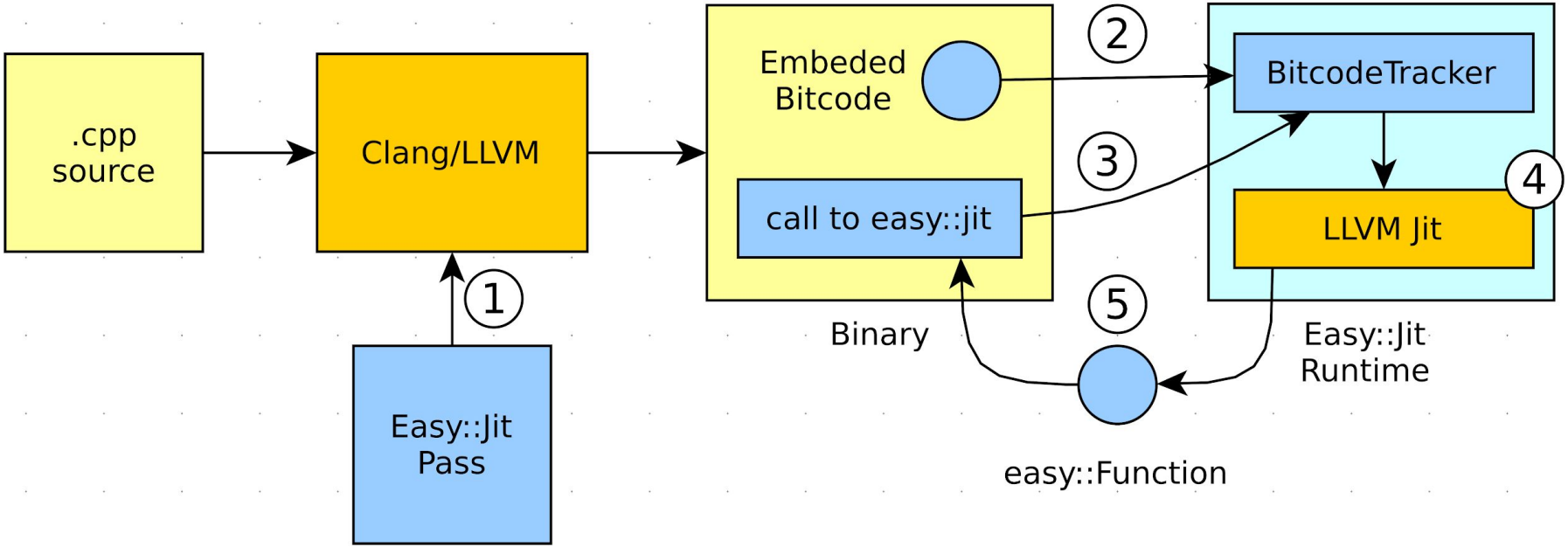
# How?

# Easy::Jit: Internals



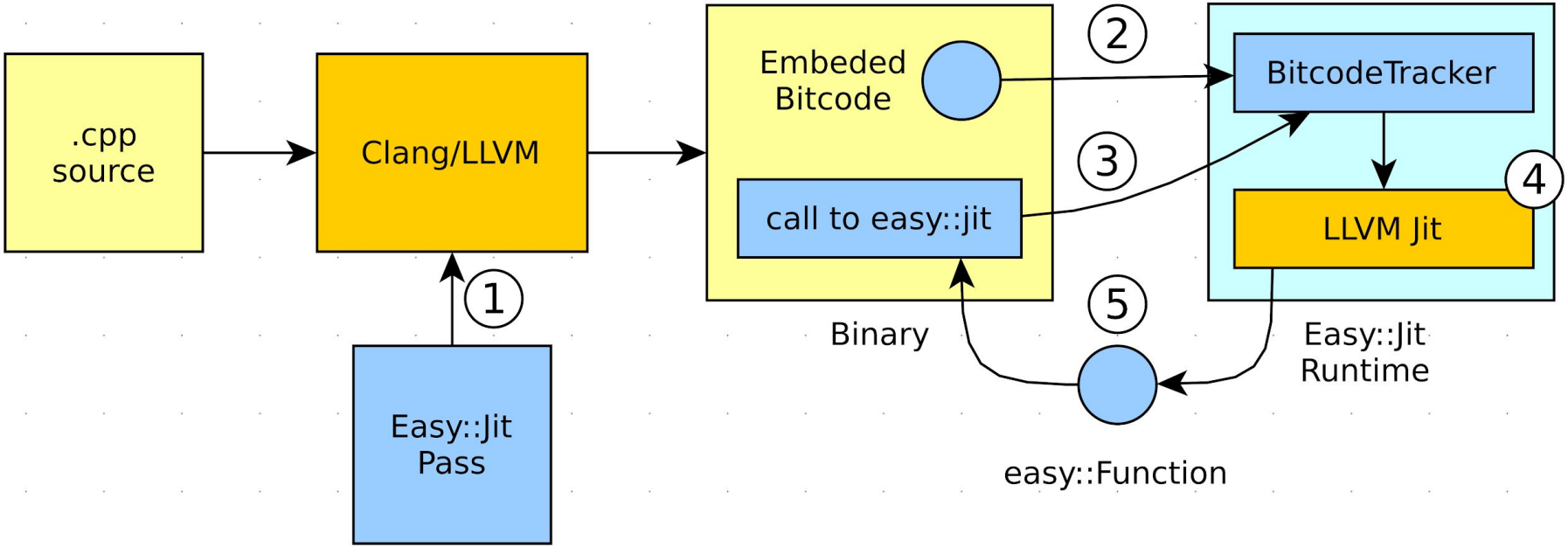

1. Parse calls to easy::jit and embed bitcode

# Easy::Jit: Internals



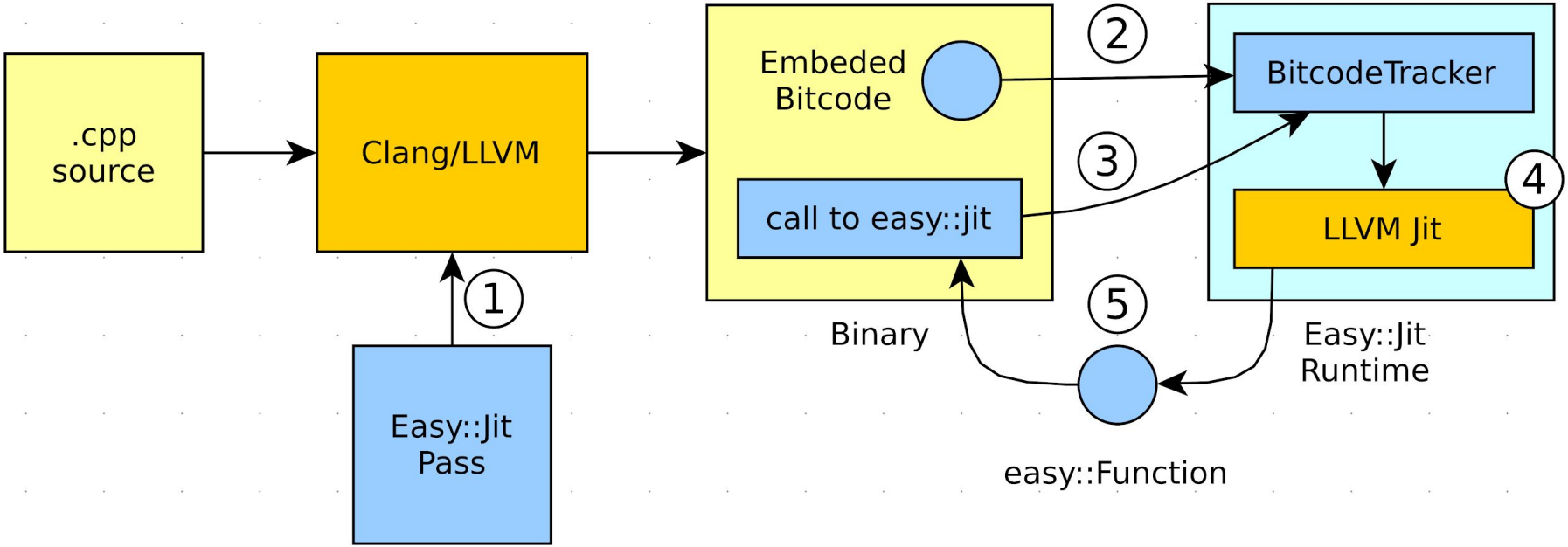

2. Associate function pointers with bitcode

# Easy::Jit: Internals



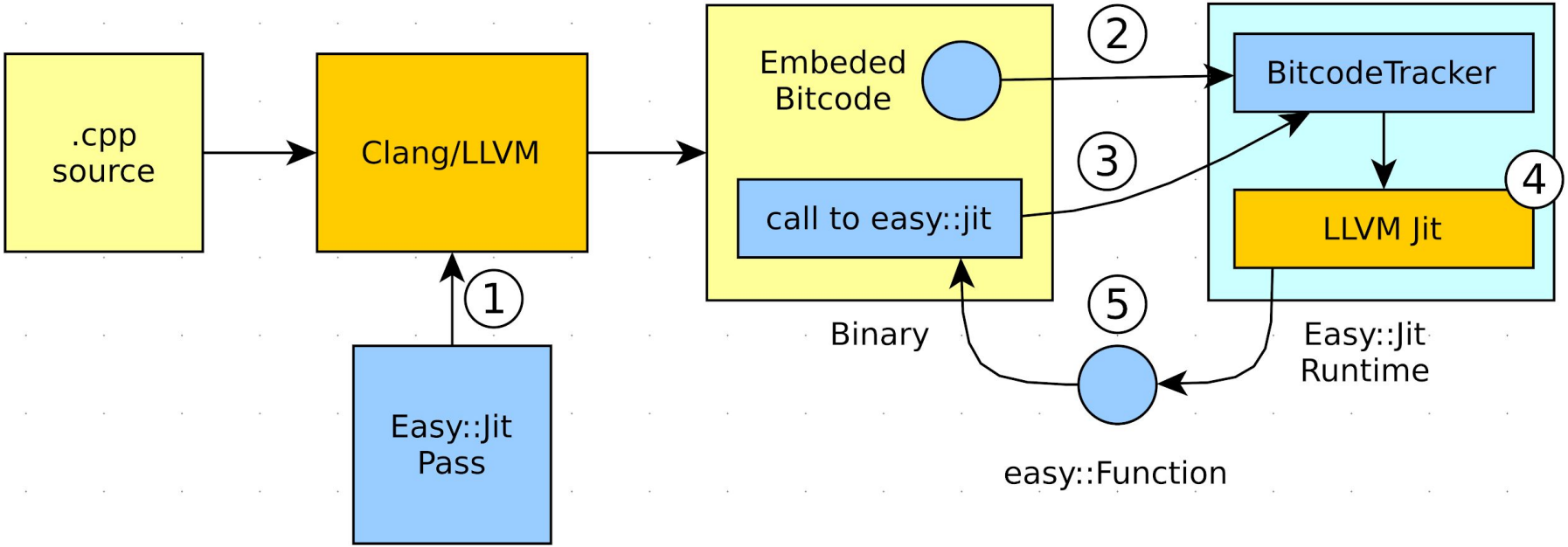

3. Recover the bitcode using the function pointer, specialize, apply classical optimizations

# Easy::Jit: Internals



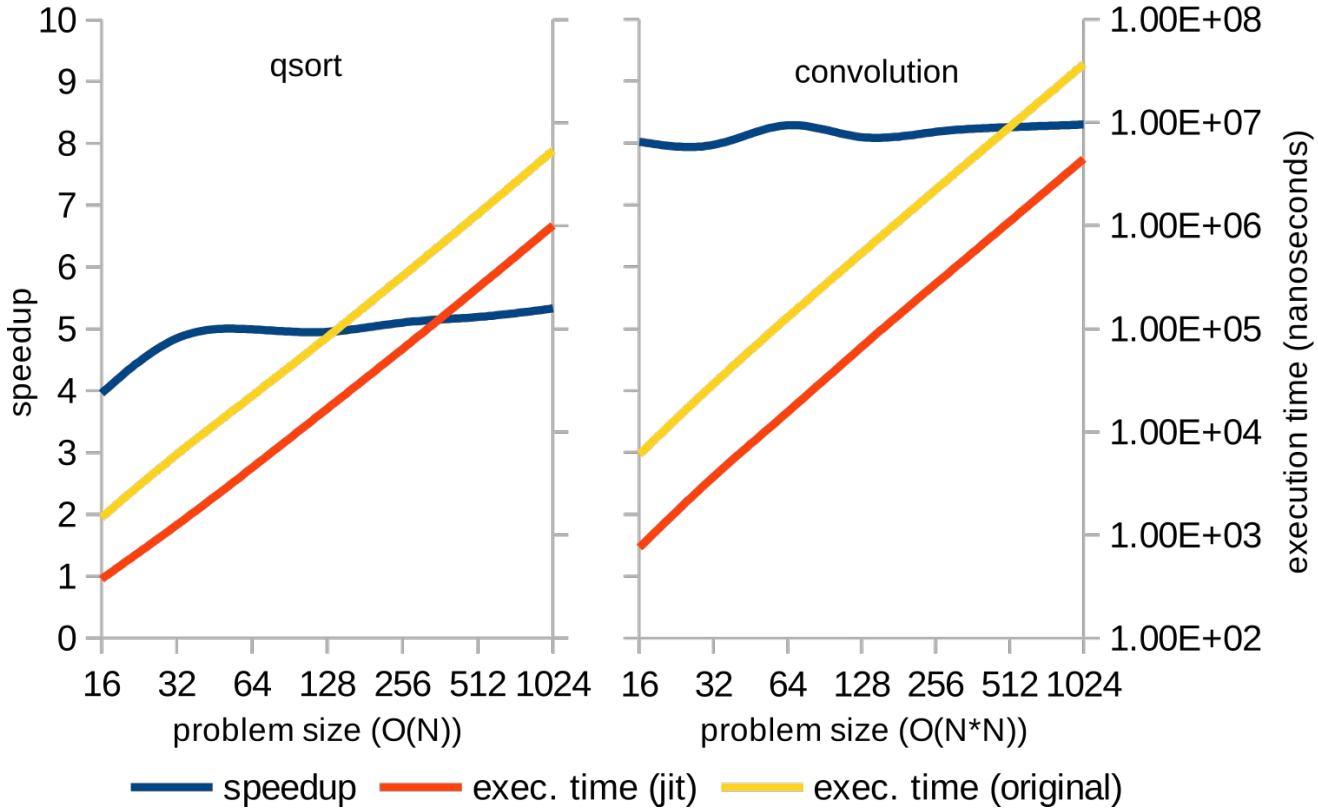

4. Generate code

# Easy::Jit: Internals




5. Wrap in an opaque object

# Easy::Jit: The numbers



# Final words

# Easy::Jit: Stuff not mentioned

- Serialization / Deserialization on standard streams
- Inlining of function
- Composition of generated code
- Devirtualization of virtual method calls

# Easy::Jit: Contribute!

- C API (work started)
- Cache: Threading + Persistency
- Member functions and function objects
- Partial Evaluation

```
void eval(AST* ast, int variables[]);
...
auto program = easy::jit(eval, my_ast, _1);
program(var_values)
```

# Contribute!

github.com/jmmartinez/easy-just-in-time

Merci **Quarkslab** :)