C++ in Indigo Presses FW

Ronen Friedman

Dec, 2018

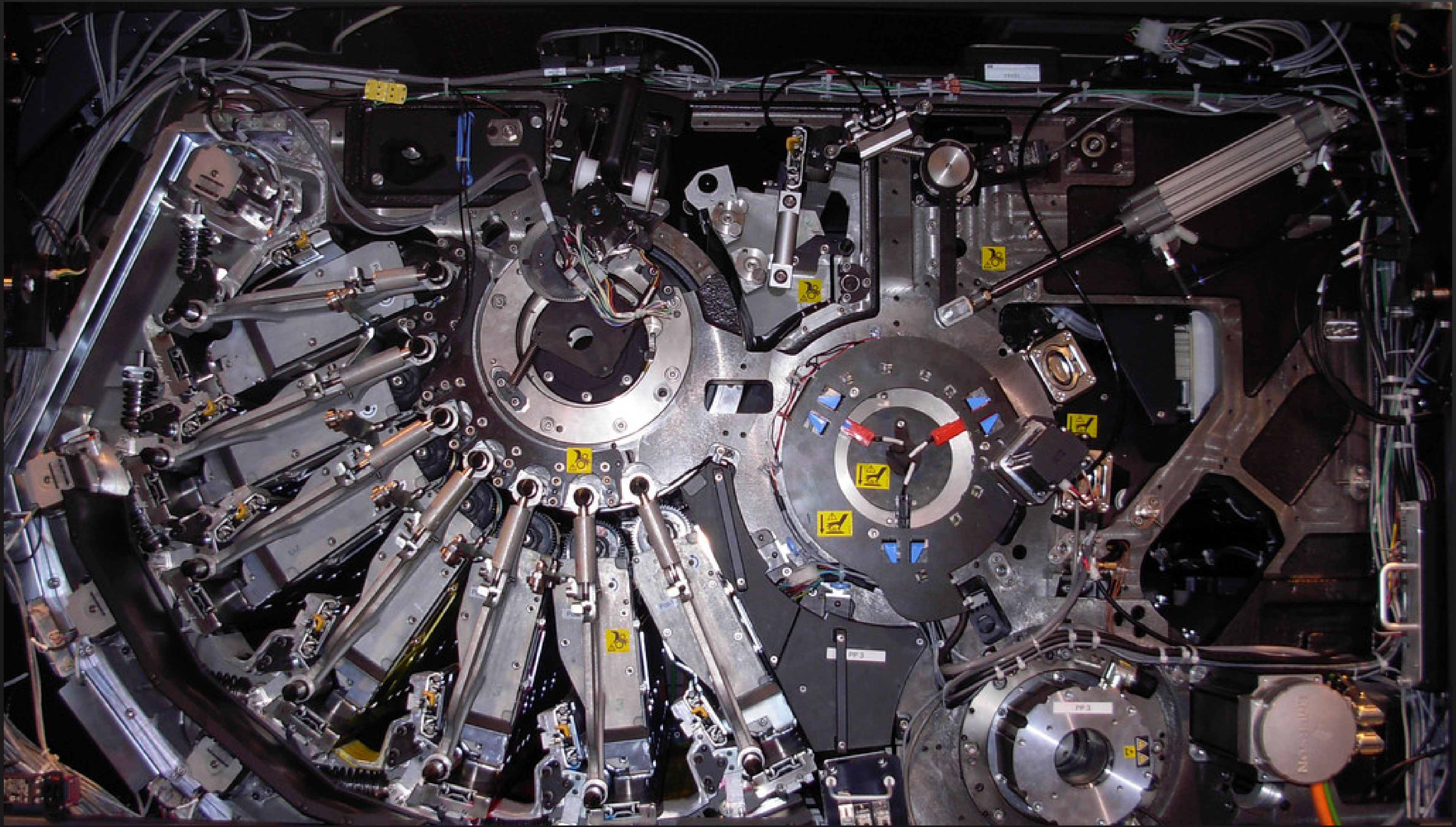
i'm french!
why do you think i have
this outrageous accent?

What we mean by "real-time"

(this is a “printer”…)

(and this is a Press…)

What we mean by “real-time”



What we mean by “real-time”

- Most(*) real-time aspects are controlled by our firmware, which is:
  - Running on multiple electronic boards;
  - Mostly ARM-based processors;
  - CMX, ThreadX, VxWorks, Linux

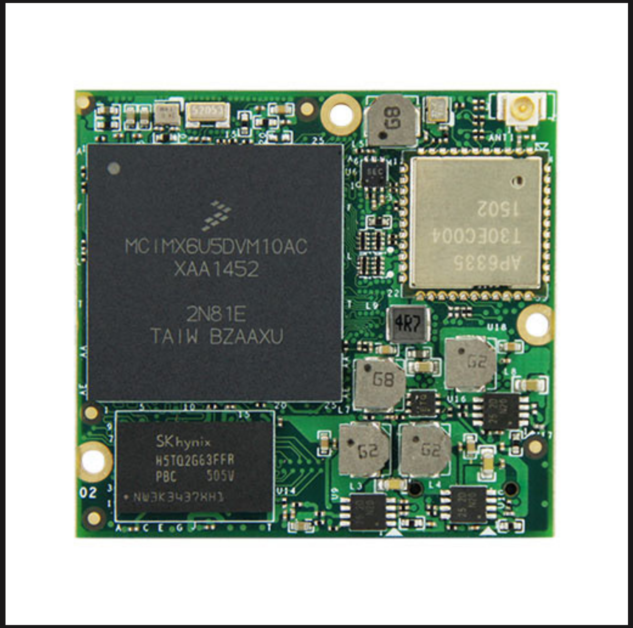
- It used to be ‘C’ only….

First – there are those buzzwords…

- OOD, OOP
- The types system
- Metaprogramming

Limitations

- 512KB FLASH + 64KB RAM
- 1MB FLASH + 192KB RAM
- …
- 1GB RAM



FUD

- performance:
  - the “virtual tables menace”
  - the “wasteful arrays-with-checks”

- losing control over memory allocation:
  - “new() is called, but you do not see it in the code”

- difficult language

- losing determinism

- code bloat (AKA “do not use the STL”)

First – there are those buzzwords…

- OOD, OOP
- The types system
- Metaprogramming

and it comes down to:

High-level abstraction with limited and known cost

Enabling:

- faster coding
- clearer code thru improved expressiveness
- improved quality, robustness
- smaller / faster binary code (*)

The C++ we use…

- Dialect: C++03 to C++17, governed by compilers availability and legacy code.
- No exceptions.
- Limited RTTI.
- STL – but carefully.

CppCon 2018 Embedded C++ Panel

Godbolt.org – the best weapon

std::array

```
#include <array>

std::array<uint32_t, 256> array_1;

uint32_t array_2[256];

int get_1(int index) { return array_1[index]; }

int get_2(int index) { return array_2[index]; }
```

```
get_1(int):
        movsx   rdi, edi
        mov     eax, DWORD PTR array_1[0+rdi*4]
        ret

get_2(int):
        movsx   rdi, edi
        mov     eax, DWORD PTR array_2[0+rdi*4]
        ret
```

“I think I’m gonna like it here” – 1 of n – compile-time

example 1: dispatch tables

```
// common .h file
#define COMMAND_A    1
#define COMMAND_B    3
#define COMMAND_C    5
#define COMMAND_D    2
```

```
using Handler = void (*)(Param);

array<Handler, 10> dispatch;
```

```

void handle_incoming_msg(int opcode, Param p)
{
   (*dispatch[opcode])(p);
}
```

“I think I’m gonna like it here” – 1 of n – compile-time

example 1: dispatch tables

```
// common .h file
#define COMMAND_A    1
#define COMMAND_B    3
#define COMMAND_C    7
#define COMMAND_D    2
```

```
using Handler = void (*)(Param);
```

```
void handle_msg(int opcode, Param p)
{
   (*dispatch[opcode])(p);
}
```

```
const array<Handler, 10> dispatch{

 /*  */ reject_cmd,
 /*  */ do_command_a,
 /*  */ do_command_d,
 /*  */ do_command_b,
 /*  */ reject_cmd,
 /*  */ reject_cmd,
 /*  */ reject_cmd,
 /*  */ reject_cmd,
 /*  */ do_command_c,
 /*  */ reject_cmd,
 /*  */ reject_cmd
};
```

“I think I’m gonna like it here” – 1 of n – compile-time

example 1: dispatch tables

```
// common .h file
#define COMMAND_A    1
#define COMMAND_B    3
#define COMMAND_C    7
#define COMMAND_D    2
```

```
using Handler = void (*)(Param);
```

```
void handle_msg(int opcode, Param p)
{
   (*dispatch[opcode])(p);
}
```

```
array<Handler, 10> dispatch;

const DispInit known_disp[] = {
 {COMMAND_A, do_command_a},
 {COMMAND_B, do_command_b},
 {COMMAND_C, do_command_c},
 {COMMAND_D, do_command_d},
};

void init_dispatch()
{
  for (auto& e : dispatch)
    e = reject_cmd;

  for (i=0; i<4 ;++i)
    dispatch[known_disp[i].cmd_] = known_disp[i].func_;
}
```

“I think I’m gonna like it here” – 1 of n – compile-time

example 1: dispatch tables

```
template <class T> struct Fn_n_index { int idx_; T t_; };

template <class T, size_t N, size_t AN, size_t... I>
constexpr array<remove_cv_t<T>, N>
to_tbl_int(const array<Fn_n_index<T>, AN> a, const T& def, index_sequence<I...>)
{
  return { {[&](int idx) constexpr -> T {
    for (auto e : a)
      if (e.idx_ == idx)
        return e.t_;
    return def;
  }(I)...} };
}

template <class T, size_t N, size_t AN>
constexpr array<remove_cv_t<T>, N> to_tbl(const array<Fn_n_index<T>, AN> &a, const T& def)
{
  return to_tbl_int<T,N>(a, def, make_index_sequence<N>{});
}
```

“I think I’m gonna like it here” – 1 of n – compile-time

example 1: dispatch tables

```
template <class T> struct Fn_n_index { int idx_; T t_; };

template <class T, size_t N, size_t AN, size_t... I>
constexpr array<remove_cv_t<T>, N>
to_tbl_int(const array<Fn_n_index<T>, AN> a, const T& def, index_sequence<I...>)
{
  return { {[&](int idx) constexpr -> T {
    for (auto e : a)
      if (e.idx_ == idx)
        return e.t_;
    return def;
  }(I)...} };
}

template <class T, size_t N, size_t AN>
constexpr array<remove_cv_t<T>, N> to_tbl(const array<Fn_n_index<T>, AN> &a, const T& def)
{
  return to_tbl_int<T,N>(a, def, make_index_sequence<N>{});
}
```

“I think I’m gonna like it here” – 1 of n – compile-time

example 1: dispatch tables

```
using Fp = int (*)(float x);

static int reject_cmd(float x) { return 0; }

static constexpr const std::array< Fn_n_index<Fp>, 3 > fp_initar {
     Fn_n_index<Fp>
     { COMMAND_D, f1}
    ,{ COMMAND_B, f3}
    ,{ COMMAND_C, f5}
};

constexpr const std::array<Fp,10> dispatch_tbl{to_tbl<Fp,10>(fp_initar,
                                                            reject_cmd)};
```

https://godbolt.org/z/NFPylp

```
dispatch_tbl:
        .quad   f_def(float)
        .quad   f_def(float)
        .quad   f1(float)
        .quad   f3(float)
        .quad   f_def(float)
        .quad   f5(float)
        .quad   f_def(float)
        .quad   f_def(float)
        .quad   f_def(float)
        .quad   f_def(float)
```

“I think I’m gonna like it here” – 2 of n

<chrono>

```
extern  Retv sdo_read(uint8_t chan_number, uint16_t index, uint8_t subix, uint8_t* data, int data_sz, uint32_t timeout);
```

“I think I’m gonna like it here” – 2 of n

<chrono>

```
extern  Retv sdo_read(uint8_t chan_number, uint16_t index, uint8_t subix, uint8_t* data, int data_sz, uint32_t timeout);


extern  Retv sdo_read(uint8_t chan_number, uint16_t index, uint8_t subix, uint8_t* data, int data_sz, milliseconds
timeout);

using ChannelNumber =
    fluent::NamedType< uint16_t, struct ChannelNumber_tag, fluent::ImplicitlyConvertibleTo< uint16_t >::templ>;

extern  Retv sdo_read( ChannelNumber cnl, uint16_t index, uint8_t subix, uint8_t* data, int data_sz, milliseconds timeout);
```

“I think I’m gonna like it here” – 3 of n

<outcome>, <optional> et. al.

```
RetCode read_sensor(int sensor_id, int* d);
```

“I think I’m gonna like it here” – 3 of n

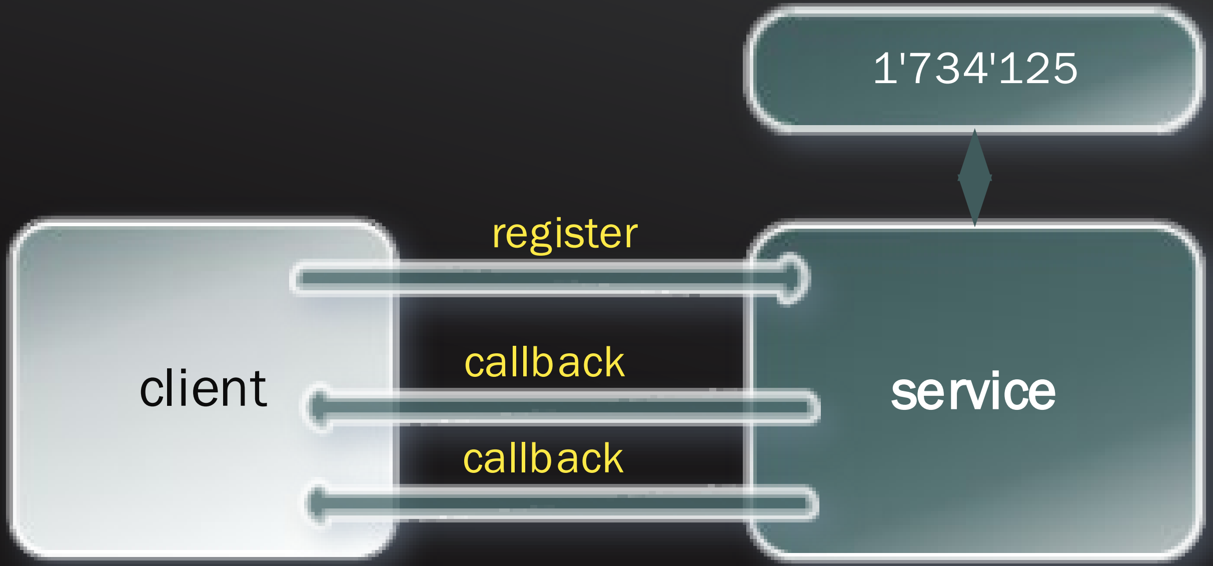
<outcome>, <optional> et. al.

```
RetCode read_sensor(int sensor_id, int* d);
```

```
using MaybeSensorData = outcome::result< SensorVal, MyErrors >;

MaybeSensorData   read_sensor(  Sensor  sensor_id  );
```

“I think I’m gonna like it here” – 4 of n - callbacks



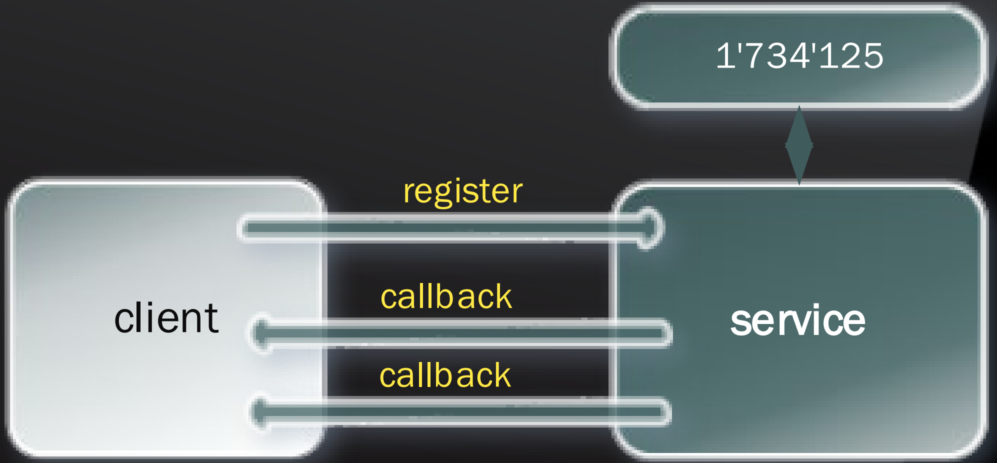
“I think I’m gonna like it here” – 4 of n - callbacks

```
typedef void (*Func)(Token, Results );

Token register_client(RequestParams* p);


void client()
{
   global_token1 = register_client(&request_1);
   // ...
   global_token2 = register_client(&request_2);
}

void my_callback(Token tk, Results* data)
{
   switch (tk) {
      case global_token1: …
      case global_token2: …
   }
}
```



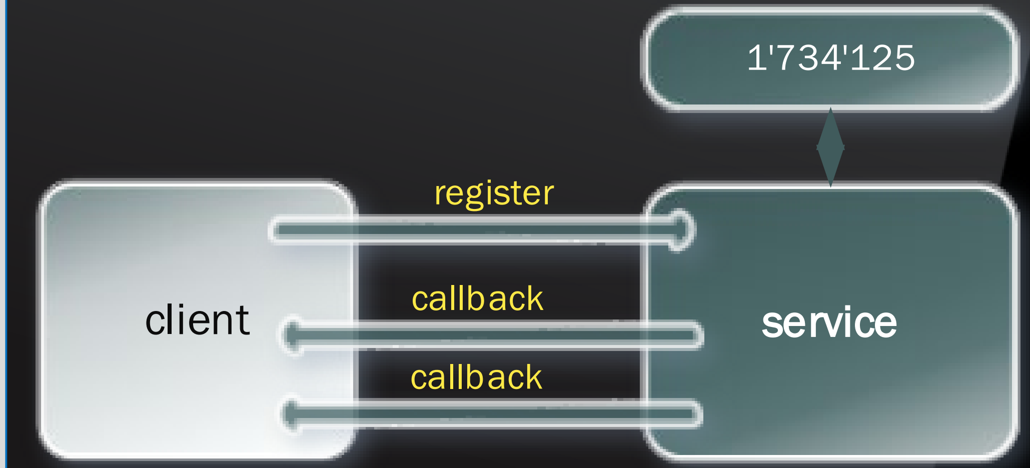
“I think I’m gonna like it here” – 4 of n - callbacks

```
template <typename P, class C, typename R>
class Callback
{
public:
  using F = R (C::*)(P);
  R operator()(P p){ (c_->*f_)(p); }

  Callback(C* c, F f): c_{c}, f_{f} {}
private:
  C* c_;
  F f_;
};

struct Client {
  bool my_cb1(float t) ;
  Callback<float, Client, bool> funct{this, &Client::my_cb1};
};

*
```



“I think I’m gonna like it here” – 4 of n - callbacks

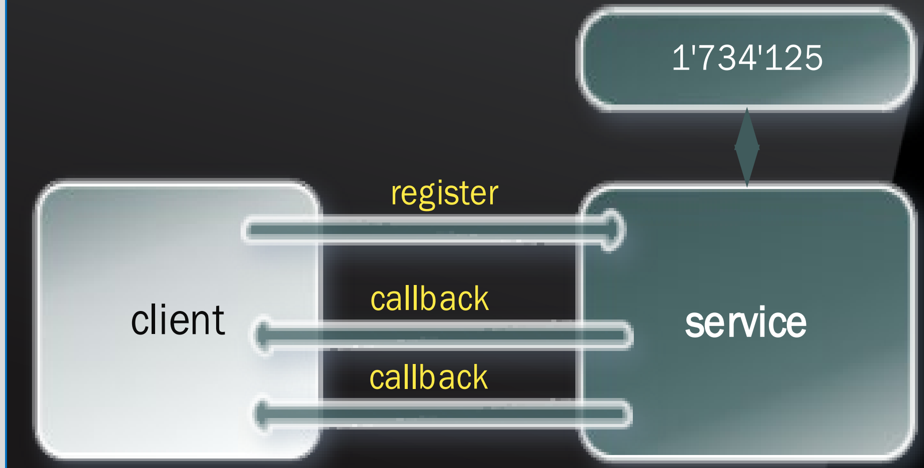
```
// std::function (which allocates), or
using Fct = stdext::inplace_function< bool( float ), 16 >;

struct Client {
  // …
  bool my_cb1(float t) { return (t > 1.0f); }
};


void x()
{
  Client client;

  Fct funct{[&client](float t) {
                    return client.my_cb1(t);}};

  volatile bool res = (funct)(22.0);
}
*
```



Hardship and Suffering – 1 of n

std::thread

- No interface to control stack size, priority, thread name
- … and the stack size must be set at creation.

- The cost:
  - OS-specific code to set all thread parameters;

  - Avoiding std::thread and all library features built using threads.

- P0320, P0484: CppCon 2017: Patrice Roy “Designing A Feature That Doesn't Fit”

Hardship and Suffering – 2 of n

PI synchronization primitives

- Priority inversion – a big no-no in real-time systems;
- The common solution: Priority Inheritance – supported by *every* RTOS for mutexes

Hardship and Suffering – 3 of n

no exceptions

- We are willing to live in the world of “die if something throws”
- But not wish to pay for what we do not eat
- Missing documentation

Hardship and Suffering – 4 of n

Allocators