Our Adventures With REST API in C++ : Making it Easy

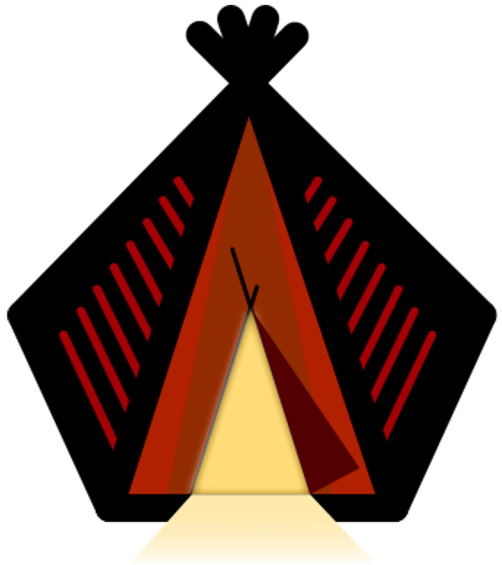
DAMIEN BUHL

Our Adventures With REST API in C++ : Making it Easy

CppCon 2021 – 29th October 2021

Cppcon
The C++ Conference

www.tipi.build



Tipi

Damien Buhl
damien@tipi.build
+41 (0) 78 984 08 13

Our Founders

Damien Buhl
CEO & CTO

Yannic Staudt
COO & Business Development

cesi

EH

n|w Fachhochschule Nordwestschweiz

boost C++ libraries

Qt

staudt engineering

MARQUARDT

Cppcon The C++ Conference

CppCast

KICKSTARTER

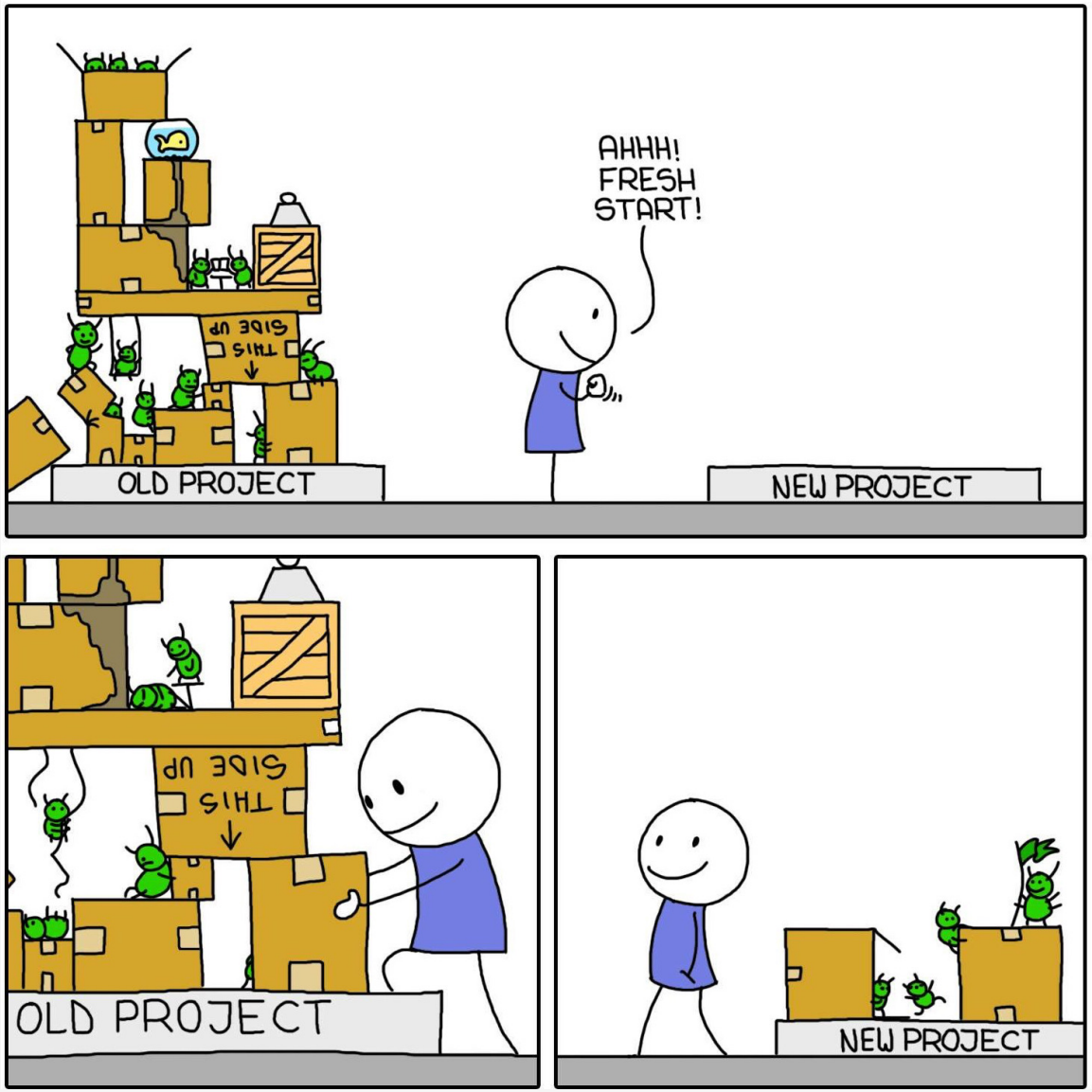
Agenda

Our Adventures With REST API in C++ : Making it Easy

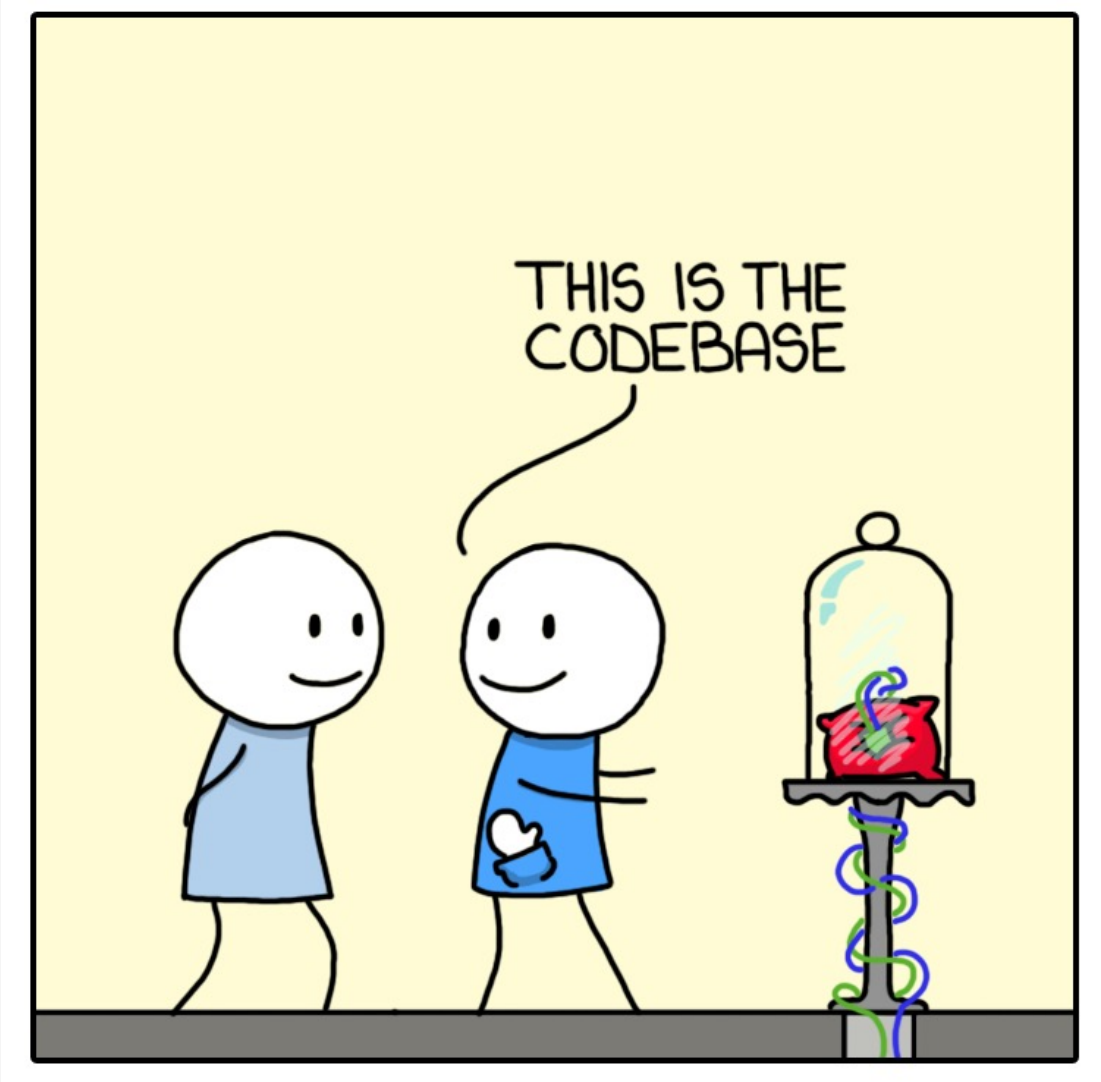
- REST what it is
- REST & SOAP
- Writing C++ REST Client
  - Traditional solutions
  - Code generation
  - Modern C++20 REST solution

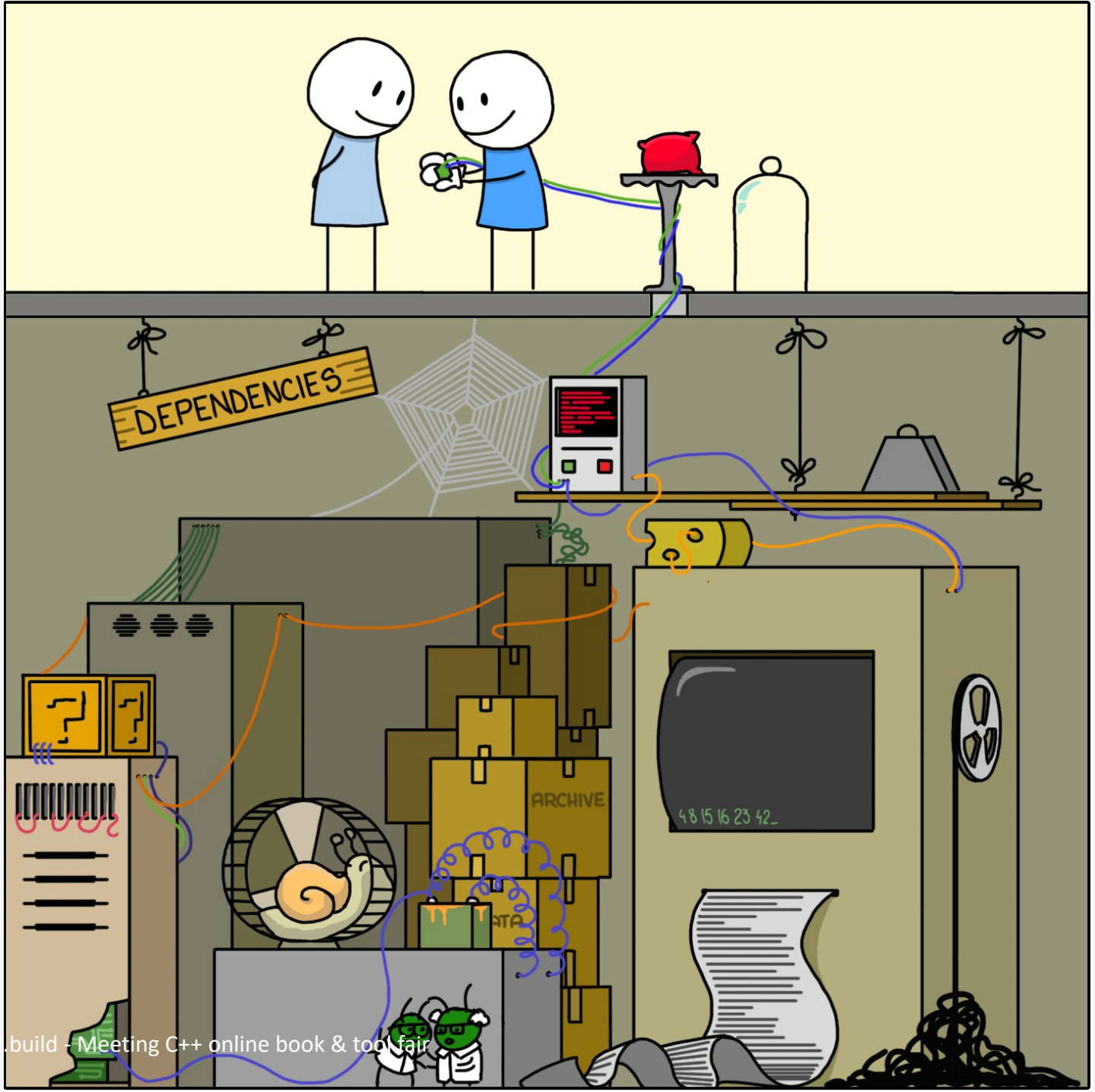
3 problems with C & C++

Code Reuse


AHHH!
FRESH
START!
THIS
SIDE UP
OLD PROJECT
NEW PROJECT
THIS
SIDE UP
OLD PROJECT
NEW PROJECT

Dependencies & Environments




DEPENDENCIES
ARCHIVE
4 8 15 16 23 42_

Build & Deploy Time

Join the open beta

Tipi.build

Build

Superfast cloud powered C & C++ builds

Join the Beta !

Tipi.run

Run & CI/CD

Machines sized right for C & C++

Join the Beta !

Join now :

https://tipi.build/

Activation Code :

tipi::launch(cppcon2021)

The fastest native edit-build-test cycle

"It could definitely change the whole ecosystem!"
Arthur Sonzogni – C++ Software Developer

Join the open beta

Tipi.build

Build

Superfast cloud powered C & C++ builds

Join the Beta !

Tipi.run

Run & CI/CD

Machines sized right for C & C++

Join the Beta !

Join now :

https://tipi.build/

Activation Code :

tipi::launch(cppcon2021)

The fastest native edit-build-test cycle

"It could definitely change the whole ecosystem!"
Arthur Sonzogni – C++ Software Developer

Join the open beta

Tipi.build

Build

Superfast cloud powered C & C++ builds

Join the Beta !

Tipi.run

Run & CI/CD

Machines sized right for C & C++

Join the Beta !

Join now :

https://tipi.build/

Activation Code :

tipi::launch(cppcon2021)

The fastest native edit-build-test cycle

"It could definitely change the whole ecosystem!"
Arthur Sonzogni – C++ Software Developer

Another problem : REST APIs

*Representational state transfer (REST) is **a software architectural style** that was created […] for the World Wide Web.*

Wikipedia

REST : Intro

REST

Representational state transfer

- Accessing resources using URIs
- Resources encapsulate entities
- Resources are manipulated
  - CRUD
  - through hypertext representations
  - transferred in messages
- Strong decoupling of client and server

  “anarchic scalability” & “low-entry barrier”

REST APIs

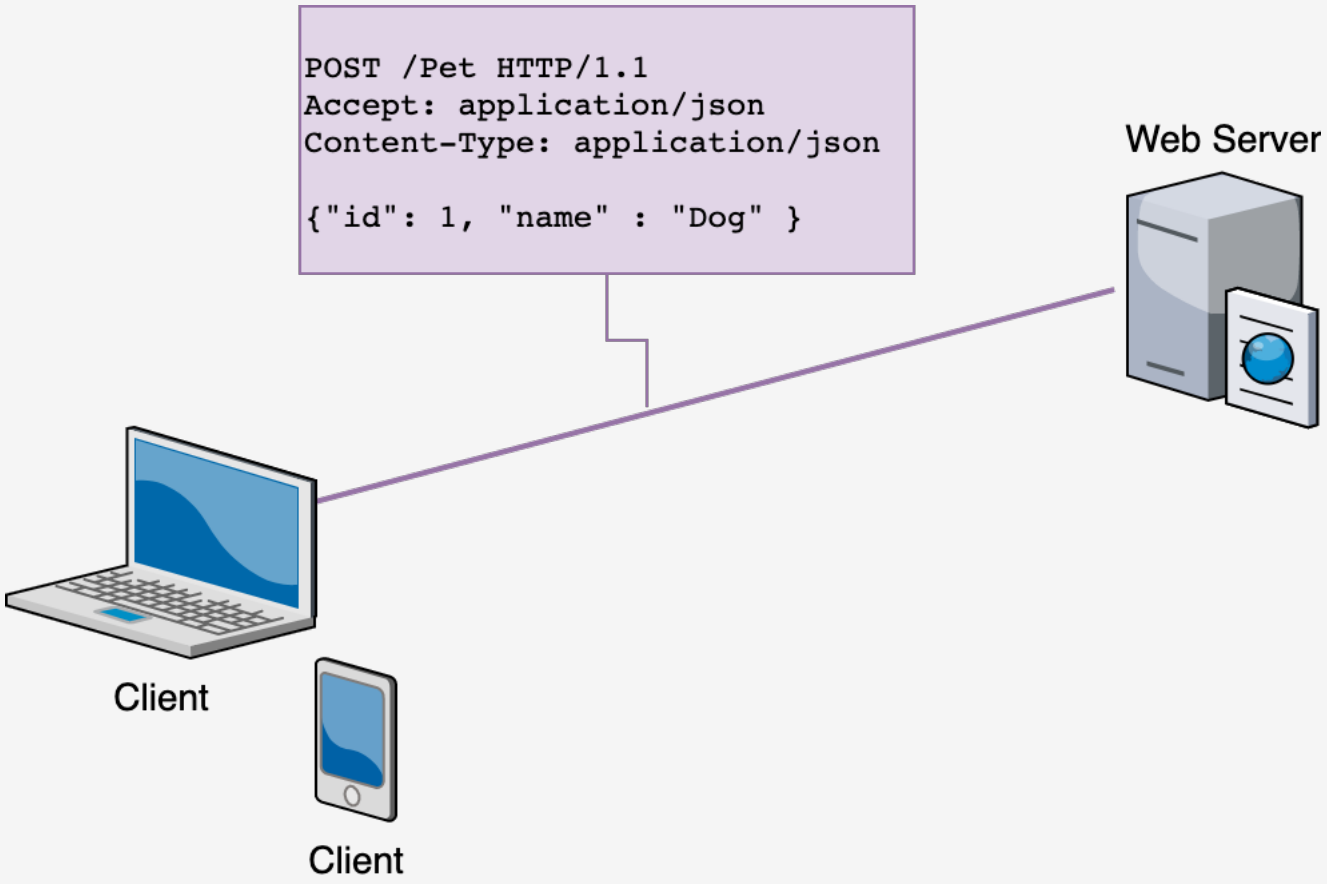
RESTful webservices

- A base URI
  - https://petstore3.swagger.io/
- HTTP methods

  e.g. GET, POST, PUT, DELETE
- media type that defines state transition data elements

  e.g. application/json; charset=utf-8
- URI segments to identify resources

RESTful webservice Petstore



Writing a traditional REST Client

Writing a traditional REST Client for tipi's official cat

tipi.build official cat : Haskell

REST

REST

Writing a traditional REST Client

- Taking a simple example petstore.io
- [de]serialization

[de]serialization

```cpp
namespace ns {
  void to_json(json& j, const person& p) {
    j = json{{"name", p.name}, {"address", p.address}, {"age", p.age}};
  }

  void from_json(const json& j, person& p) {
    j.at("name").get_to(p.name);
    j.at("address").get_to(p.name);
    j.at("age").get_to(p.age);
  }
}
```

[de]serialization

```cpp
namespace ns {
  void to_json(json& j, const person& p) {
    j = json{{"name", p.name}, {"address", p.address}, {"age", p.age}};
  }

  void from_json(const json& j, person& p) {
    j.at("name").get_to(p.name);
    j.at("address").get_to(p.name); 🐞
    j.at("age").get_to(p.age);
  }
}
```

C++03 Better [de]serialization

```cpp
#include <pre/json/from_json.hpp>

struct person {
  std::string name;
  std::size_t age;
  std::string address;
};

BOOST_FUSION_ADAPT_STRUCT(person, name, age, address)

//...
auto person = pre::json::from_json<person>(R"(
    { "name" : "Edouard",
      "age" : 31,
      "address" : "hello@tipi.build"
    }
)");
```

C++03 Better [de]serialization

```cpp
#include <pre/json/from_json.hpp>

struct person {
  std::string name;
  std::size_t age;
  std::string address;
  bool public;
};

BOOST_FUSION_ADAPT_STRUCT(person, name, age, address, public)

//...
auto person = pre::json::from_json<person>(R"(
    { "name" : "Edouard",
      "age" : 31,
      "address" : "hello@tipi.build",
      "public" : false
    }
)");

```

C++03 Better [de]serialization

```cpp
#include <pre/json/from_json.hpp>

struct person {
  std::string name;
  std::size_t age;
  std::string address;
  bool public;
};

BOOST_FUSION_ADAPT_STRUCT(person, name, age, address, public)

//...
auto person = pre::json::from_json<person>(R"(
    { "name" : "Edouard",
      "age" : 31,
      "address" : "hello@tipi.build",
      "public" : false
    }
)");

```

C++20 Better [de]serialization

Do we really need to repeat ourselves ?

Given :

- C++20 enables CTAD
  Class Template Argument Deduction
- constexpr NTTP
  Non Type Template Parameter
- C++17 Structured binding

C++20 Better [de]serialization

No Reflection in C++ :

Structured Bindings to the rescue

```cpp
struct some_type {
  int count = 12;
  double factor = 4.3;
};

auto [x, y] = some_type{};

assert(x == 12);
assert(y == 4.3);
```

Looking into Boost.PFR by Antony Polukhin :

aggregate initialization to detect fields count in user-provided structure

structured bindings to decompose a type T to known amount of fields

C++20 Better [de]serialization

No Reflection in C++ :
Structured Bindings to the rescue

Looking into Boost.PFR by Antony Polukhin :

- aggregate initialization to detect fields count in user-provided structure
- structured bindings to decompose a type T to known amount of fields

C++20 better [de]serialization

```cpp
#include <pre/json/from_json.hpp>

struct person {
  pre_json_key(std::string, name);
  pre_json_key(std::size_t, age);
  pre_json_key(std::string, address);
};

//...
auto person = pre::json::from_json<person>(R"(
    { "name" : "Edouard",
      "age" : 31,
      "address" : "hello@tipi.build"
    }
)");
```

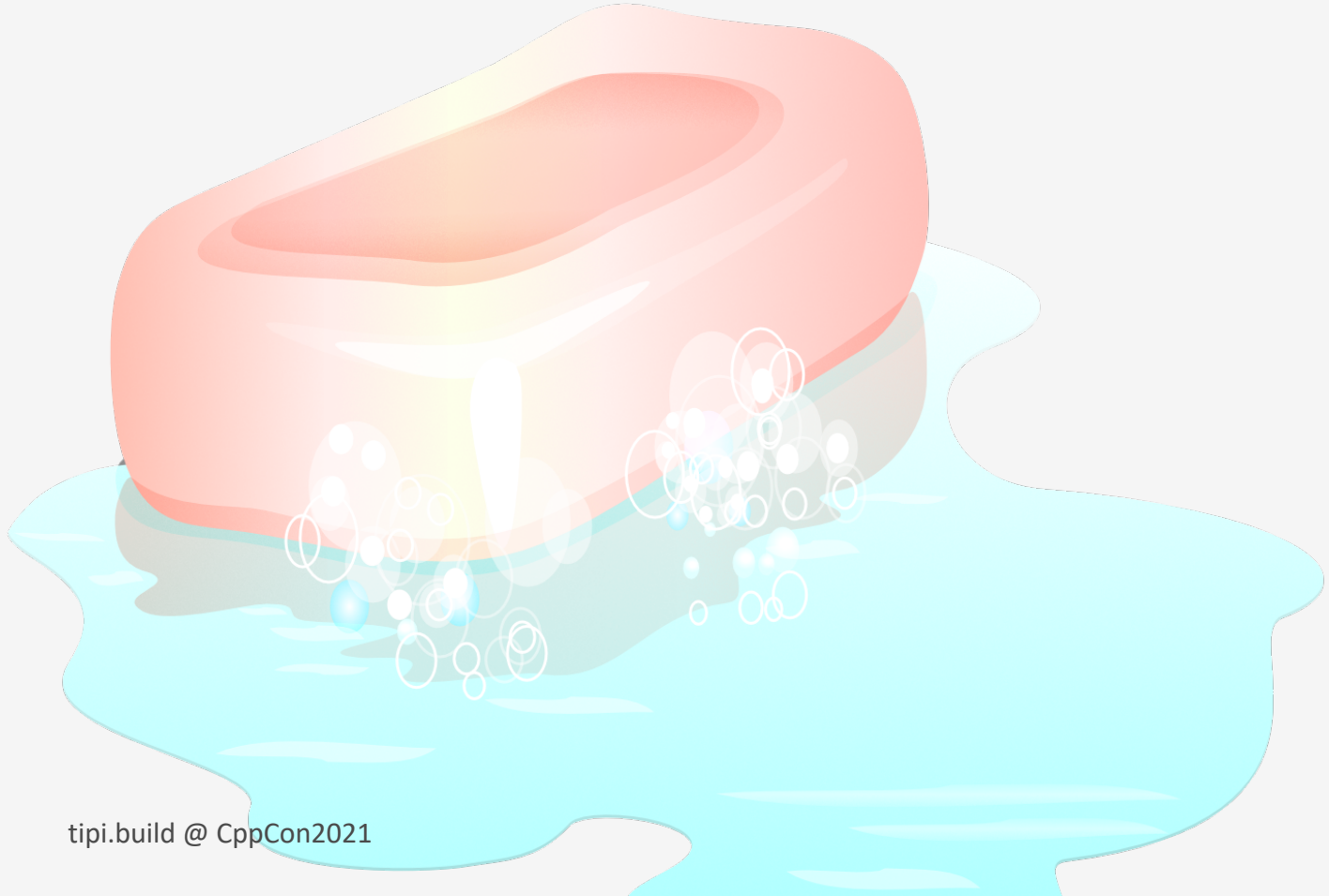
C++20 better [de]serialization

```cpp
template <cx::static_string str>
struct tstring {
  static constexpr cx::static_string value = str;
};

template <cx::static_string key_, class T>
struct key_value_pair {
  using key = tstring<key_>;
  T value;
};

#define pre_json_key(type, name)
  key_value_pair<"" # name, type> name
```

REST vs SOAP



REST vs SOAP

REST vs SOAP

Representational state transfer

- REST: Software Architectural Style
  - Freedom
  - KISS
- SOAP : Standardized protocol
  - Automated Tooling
  - Code auto generation
  - Heavyweight
  - High barrier to entry

Traditional REST Client

Writing a REST Client

- Repetitive :
  - [de]serialization
  - network calls
- Needs an extensive test suite
  - Ensuring API Compatibility
- What if the API changes ?

Conway’s law

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

— Melvin E. Conway

Adding a little of SOAP into REST

Shared Data Model

- Different approaches :
  - Share classes model + deserialization code
  - Code Generation

Generating REST Clients
Open API

Generating REST Client

Generating code

- One generator : many problem
- Regenerating may switch classes silently
- Files generated might not be up-to-date

Metaprogramming REST Clients

Metaprograms & REST Clients

Requirements

- Compile Error if API not respected
  - Change in datamodel
  - Resource URI inexistant
- Auto-completion

Metaprograms & REST Clients

Metaprogramming RESTful clients

- Combining OpenAPI
- Constexpr JSON parsing
- Modern C++ template metaprogramming

Metaprogramming REST Clients

```
static constexpr auto openapi_json = R"x(
{
    "openapi": "3.0.2",
    "info": {
        "title": "Swagger Petstore - OpenAPI 3.0",
        "description": "This is a sample Pet Store Server based on the OpenAPI 3.0 specification.",
        "termsOfService": "http://swagger.io/terms/",
        "contact": {
            "email": "apiteam@swagger.io"
        },
        "license": {
            "name": "Apache 2.0",
            "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
        },
        "version": "1.0.6"
    }
})x"_cx_json

...
auto reply = client.call_operation<"addPet">(std::make_tuple(
  key_value_pair<"id" , int>  {43},
  key_value_pair<"name" , std::string>  {"Haskell"s},
  key_value_pair<"photoUrls" , std::vector<std::string>>  { {"little_cat.png"s} }
));
```

Metaprogramming REST Clients

```
static constexpr auto openapi_json = R"x(
{
    "openapi": "3.0.2",
    "info": {
        "title": "Swagger Petstore - OpenAPI 3.0",
        "description": "This is a sample Pet Store Server based on the OpenAPI 3.0 specification.",
        "termsOfService": "http://swagger.io/terms/",
        "contact": {
            "email": "apiteam@swagger.io"
        },
        "license": {
            "name": "Apache 2.0",
            "url": "http://www.apache.org/licenses/LICENSE-2.0.html"
        },
        "version": "1.0.6"
    }
})x"_cx_json


auto reply =  addPet(Pet{{43}, {"Haskell junior"s}, {{"some_cat.png"}} });
```

tipi.build

Rethinking the C & C++ development experience

Join the open beta

Build

Superfast cloud powered C & C++ builds

Join the Beta !

Run & CI/CD

Machines sized right for C & C++

Join the Beta !

Join now :

https://tipi.build/

Activation Code :

tipi::launch(cppcon2021)

The fastest native edit-build-test cycle

"It could definitely change the whole ecosystem!"
Arthur Sonzogni – C++ Software Developer

Join the open beta

Build

Superfast cloud powered C & C++ builds

Join the Beta !

Run & CI/CD

Machines sized right for C & C++

Join the Beta !

Join now :

https://tipi.build/

Activation Code :

tipi::launch(cppcon2021)

The fastest native edit-build-test cycle

"It could definitely change the whole ecosystem!"
Arthur Sonzogni – C++ Software Developer

Join the open beta

Build

Superfast cloud powered C & C++ builds

Join the Beta !

Run & CI/CD

Machines sized right for C & C++

Join the Beta !

Join now :

https://tipi.build/

Activation Code :

tipi::launch(cppcon2021)

The fastest native edit-build-test cycle

"It could definitely change the whole ecosystem!"
Arthur Sonzogni – C++ Software Developer

Thanks

Tipi

www.tipi.build

Damien Buhl

damien@tipi.build

+41 (0) 78 984 08 13