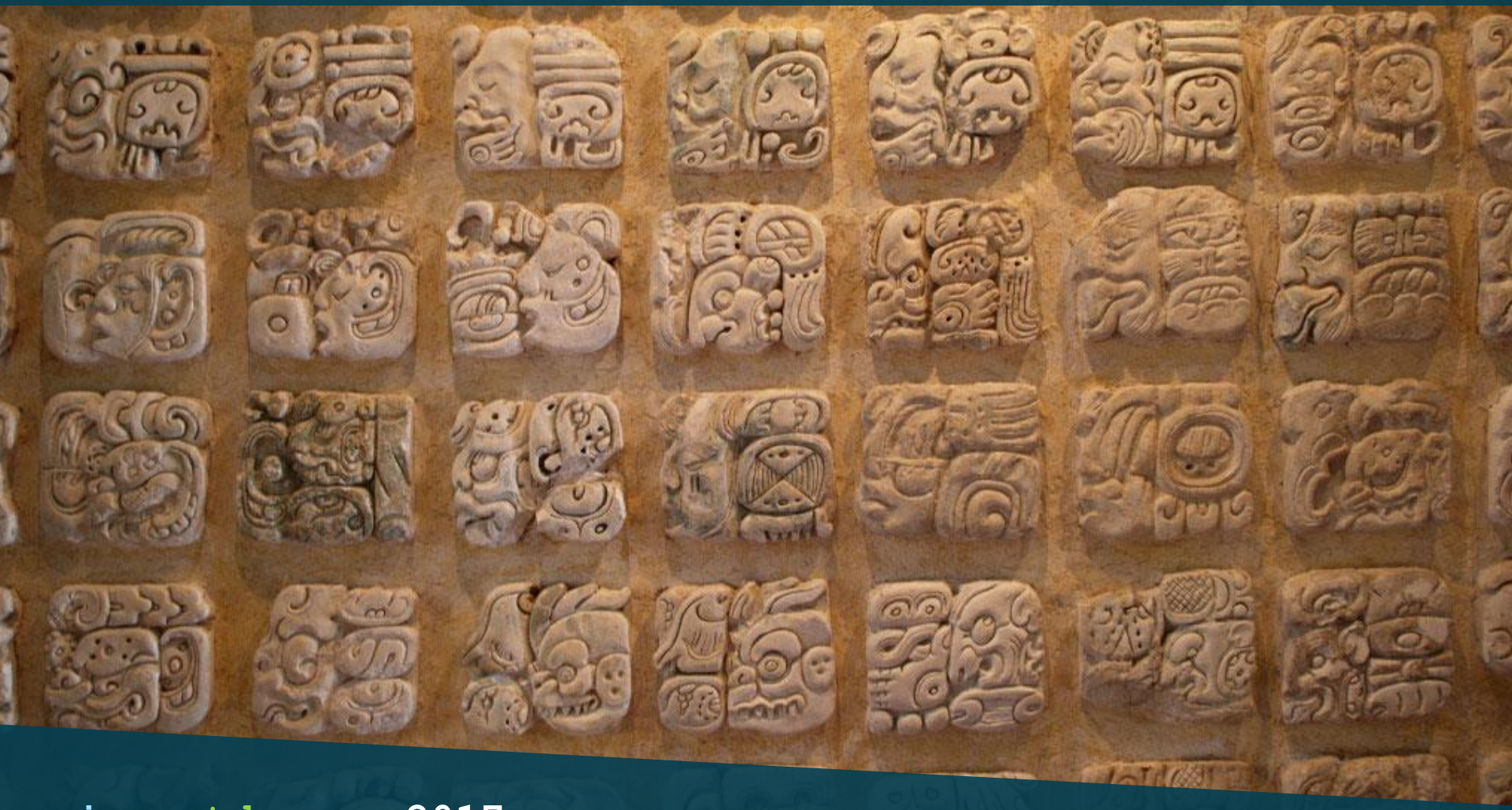
C++17 vocabulary types



`using std::cpp 2017`

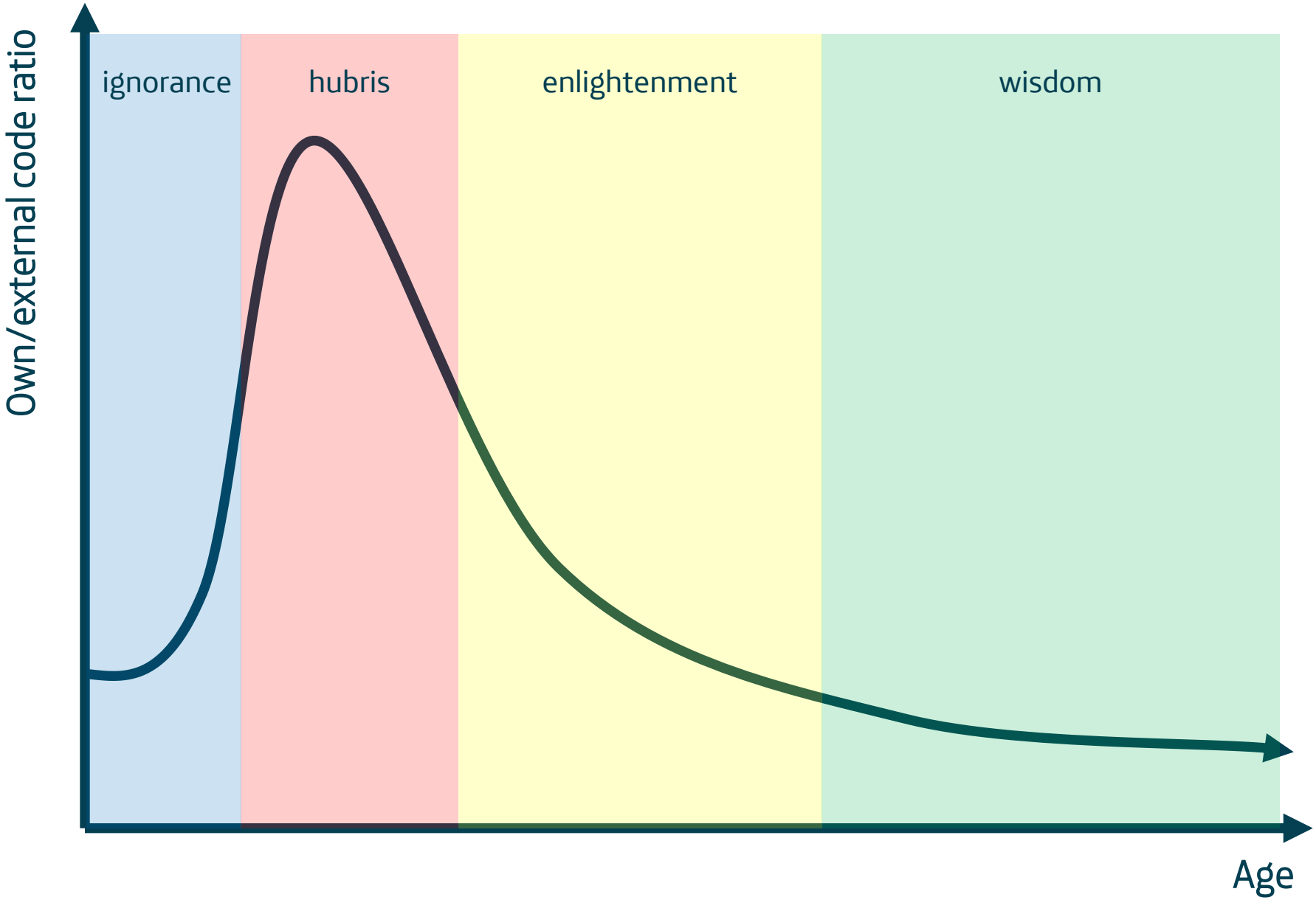
Joaquín M López Muñoz <joaquin.lopezmunoz@gmail.com>

Madrid, November 2017

Is programming an art or a craft?

“Not invented here” syndrome

Stages through the life of a programmer



Vocabulary is for combination

Nunca ſereys de alguno reprochado,
Por home de obras viles, y ſoezes.
Seran vueſſas fazañas los joezes,
Pues tuertos desfaziendo aueys andado,
Siendo vegadas mil apaleado,
Por follones cautiuos, y rahezes.
Y ſi la vueſſa linda Dulzinea,
Deſſaguiſado contra vos comete,
Ni a vueſſas cuytas mueſtra buen talante.
En tal deſman vueſſo conorte ſea,
Que Sancho Pança fue mal alcaguete,
Necio el, dura ella, y vos no amante.

DIALAGO ENTRE BABIECA, y Rozinante.

SONETO.

B. COmo eſtays Rozinante tan delgado?
R. Porque nunca ſe come, y ſe trabaja,
B. Pues que es de la ceuada, y de la paja?
R. No me dexa mi amo ni vn bocado.
B. Andà ſeñor que eſtays muy malcriado.
Pues vueſtra lengua de aſno al amo vltraja,
R. Aſno ſe es de la cuna a la mortaja,
Quereyſlo ver, miraldo enamorado.
B. Es necedad amar? R. Nó es gran prudencia.
B. Metafiſico eſtays. R. Es que no como.
B. Quexaos del eſcudero. R. No es baſtante.
Como me he de quexar en mi dolencia,
Si el amo y eſcudero o mayordomo.

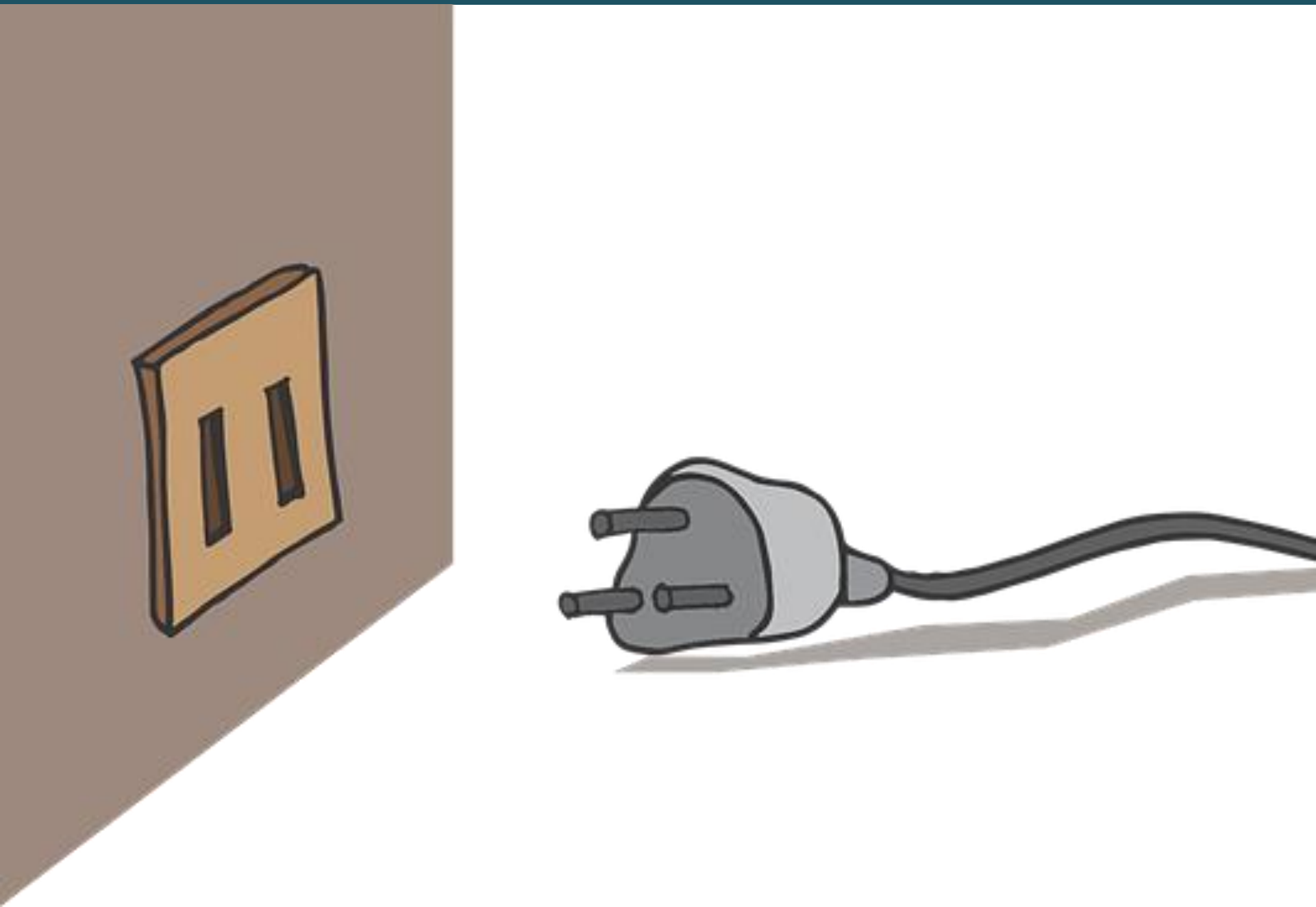
PRIMERA PARTE DEL INGENIOSO hidalgo don Quixote de la Mancha.

Capitulo Primero. Que trata de la condicion, y exercicio del famoſo hidalgo don Quixote de la Mancha.

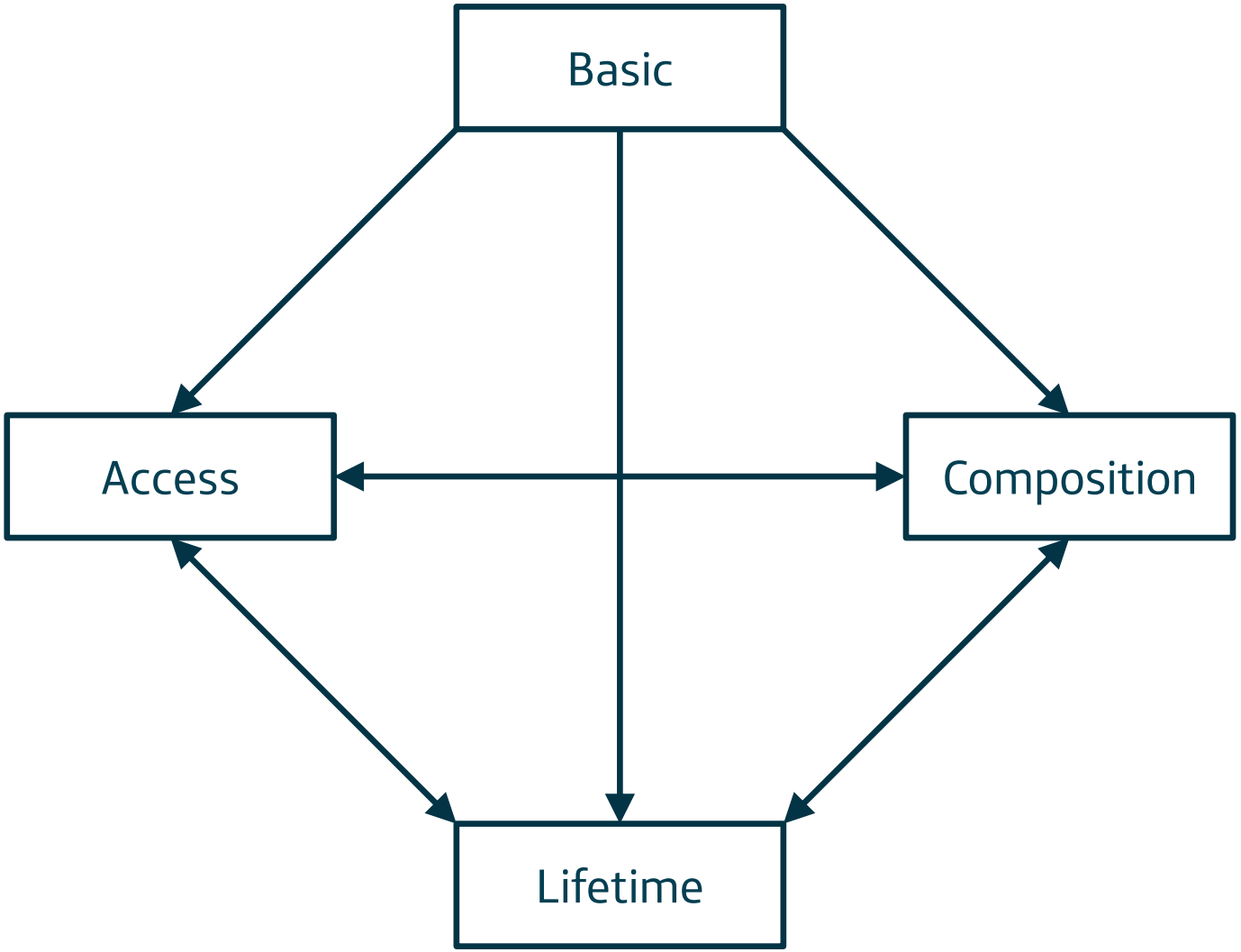
EN Vn lugar de la Mancha, de cuyo nombre no quiero acordarme, no ha mucho tiempo que viuia vn hidalgo de los de lança en aſtillero, adarga antigua, rozin flaco, y galgo corredor. Vna olla de algo mas vaca que carnero, ſalpicon las mas noches, duelos y quebrãtos los Sabados, lantejas los Viernes, algun palomino de añadidura los Domingos: conſumian las tres partes de ſu

Communication

And collaboration



A classification of vocabulary types



Basic

- `std::chrono::duration` / `std::chrono::time_point`
- `std::complex`
- `std::string`

Access

- `std::function`
- `std::reference_wrapper`
- `std::string_view` C++17

Composition

- `std::array`
- `std::any` C++17
- `std::optional` C++17
- `std::pair` / `std::tuple`
- `std::variant` C++17

Lifetime

- `std::shared_ptr` / `std::weak_ptr`
- `std::unique_ptr`
- `std::cloned_ptr` / `std::polymorphic_value` C++20?

std::string_view

string_view.cpp

- string-like object over a non-owned sequence of characters

```
std::string_view find(const char* what,const std::string& where)
{
  auto s=std::strlen(what);
  auto n=where.find(what,s);
  if(n!=std::string::npos)return {where.data()+n,s};
  else return {};
}

std::string str{"John feeds the beautiful cat in the kitchen"};
auto strv=find("feeds",str);
std::cout<<strv<<"\n"; // print "feeds"
str[8]='l';
std::cout<<strv<<"\n"; // print "feels"
```

std::any

any.cpp

- Keep me anything

```
void print(const std::any& x)
{
  if(x.type()==typeid(int))
    std::cout<<std::any_cast<int>(x)<<"\n";
  else if(x.type()==typeid(std::string))
    std::cout<<std::any_cast<const std::string>(x)<<"\n";
  else
    std::cout<<"[unknown]"<<"\n";
}

print(std::make_any<int>(10)); // prints "10"
print(std::make_any<std::string>("Hello")); // prints "Hello"
print(std::make_any<double>(3.14159265));  // prints "[unknown]"
```

- `std::any` is probably a code smell
  - There's little one can do with *anything*
- Most of the time you'll want `std::variant`

std::variant

variant.cpp

- Keep me a value from a compile-time set of types

```
using namespace std::string_literals; // for the s literal suffix
using my_type=std::variant<int,std::string,double>;

auto printer=[](const auto& x){std::cout<<x<<"\n";};

my_type v{10};
std::visit(printer,v);
v="Hello"s;
std::visit(printer,v);
v=3.14159265;
std::visit(printer,v);
```

std::optional

optional.cpp

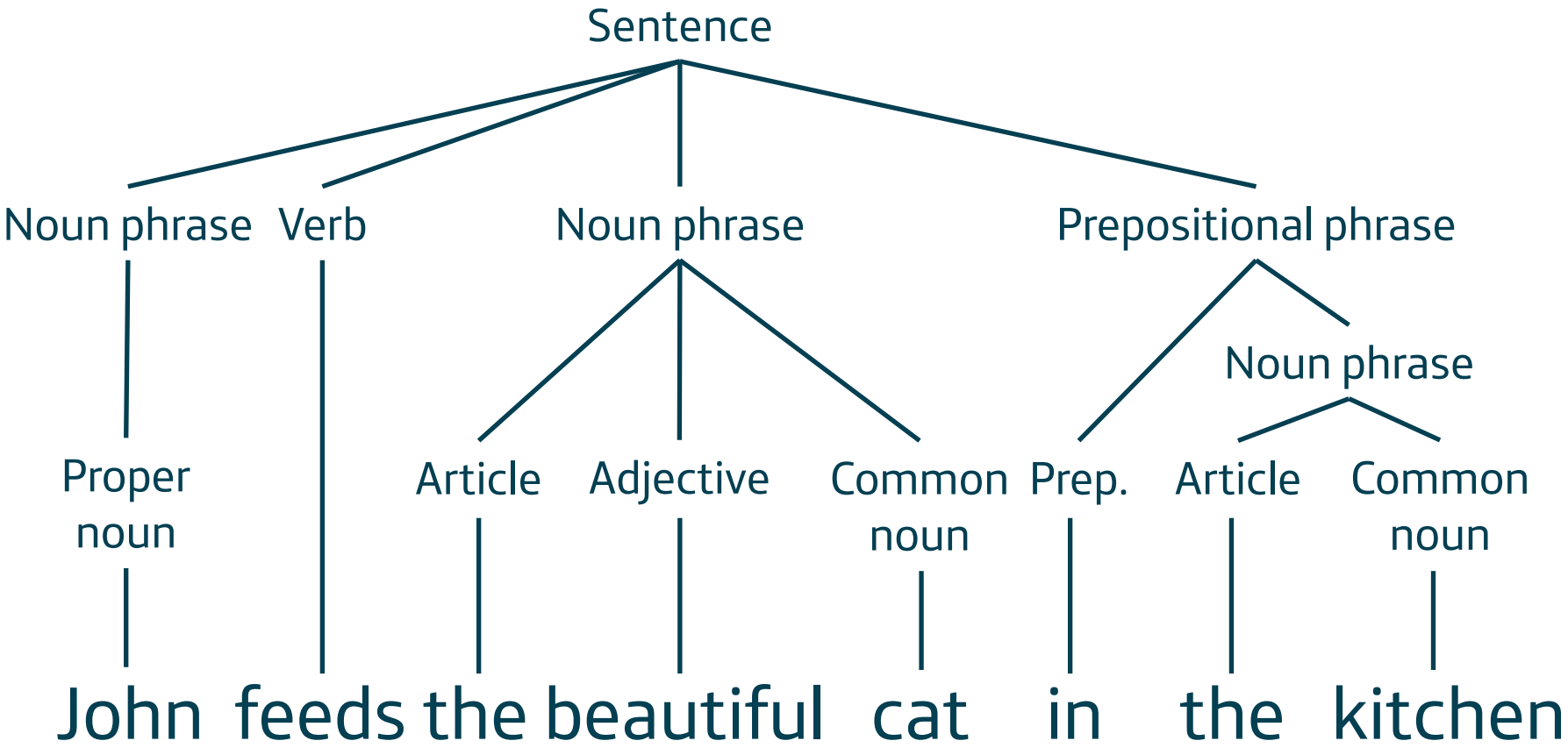
- Possibly keep me a value

```
struct name
{
  std::string                 given;
  std::optional<std::string> middle;
  std::string                 family;
};

void print(const name& n)
{
  std::cout<<n.given<<" ";
  if(n.middle)std::cout<<n.middle.value()<<" ";
  std::cout<<n.family<<"\n";
}

print(name{"John","Fitzgerald","Kennedy"});
print(name{"Bob",{},"Laszlo"});
```

High-school grammar



Model with vocabulary types

generative_grammar.cpp

```
struct proper_noun:std::string;
struct article:std::string;
struct adjective:std::string;
struct common_noun:std::string;
using  common_noun_phrase=std::tuple<
  article,
  std::optional<adjective>,
  common_noun
>;
using  noun_phrase=std::variant<proper_noun,common_noun_phrase>;
struct verb:std::string;
struct preposition:std::string;
using  prepositional_phrase=std::tuple<preposition,noun_phrase>;
using  sentence=std::tuple<
  noun_phrase,
  verb,
  noun_phrase,
  std::optional<prepositional_phrase>
>;
```

Dump

generative_grammar.cpp

```
void print(const std::string& str){std::cout<<str<<" ";}

template<typename T>
void print(const std::optional<T>& o){if(o)print(o.value());}

template<typename... Ts>
void print(const std::variant<Ts...>& v)
{std::visit([](const auto& x){print(x);},v);}

template<typename... Ts>
void print(const std::tuple<Ts...>& t)
{std::apply([](const auto&... x){int l[]={0,(print(x),0)...};(void)l;},t);}
```

Voilà

generative_grammar.cpp

```
print(sentence{
  proper_noun{"John"},
  verb{"feeds"},
  common_noun_phrase{
    article{"the"},adjective{"beautiful"},common_noun{"cat"}
  },
  prepositional_phrase{
    preposition{"in"},
    common_noun_phrase{
      article{"the"},{},common_noun{"kitchen"}
    }
  }
});
```

John feeds the beautiful cat in the kitchen

Some random fun

`generative_grammar.cpp`

Mary kisses the dog outside a red pool
John kisses Bob
John kisses Mary in the pool
a red postman kisses a postman
a big cat feeds John in a red kitchen
the dog meets a big postman outside the kitchen
a beautiful cat kisses the beautiful postman
Bob meets the big cat
a postman kisses a dog in the big pool
Bob meets a red dog outside the big pool

- Exercise for the reader (hard)
  - Change terminal nodes to use `std::string_view`
  - Parse `std::string` into a `sentence`

Time to wrap up

Time to wrap up

- Grow wise: don't reinvent the wheel
- Vocabulary as a communication and collaboration tool
- New in C++17
  - `string_view`, `any`, `optional`, `variant`
- `std::any` is likely a code smell
  - Do you come from dynamically typed languages?
- Composition types form an algebra
- I think I ran out of time

Thank you

github.com/joaquintides/usingstdcpp2017

Joaquín M López Muñoz <joaquin.lopezmunoz@gmail.com>