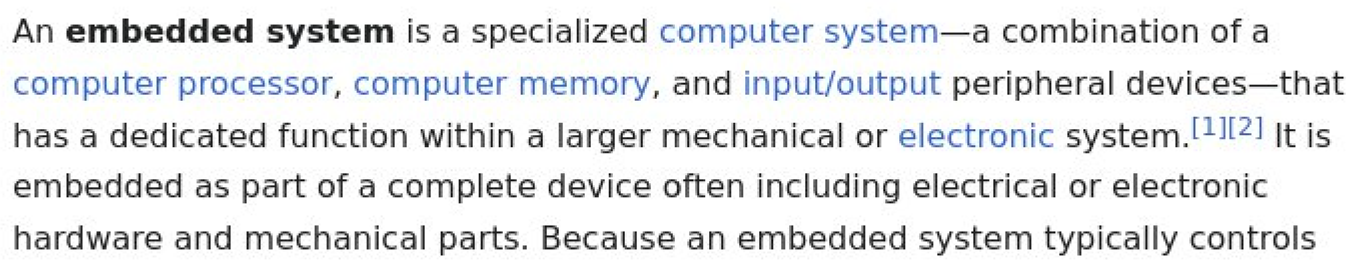
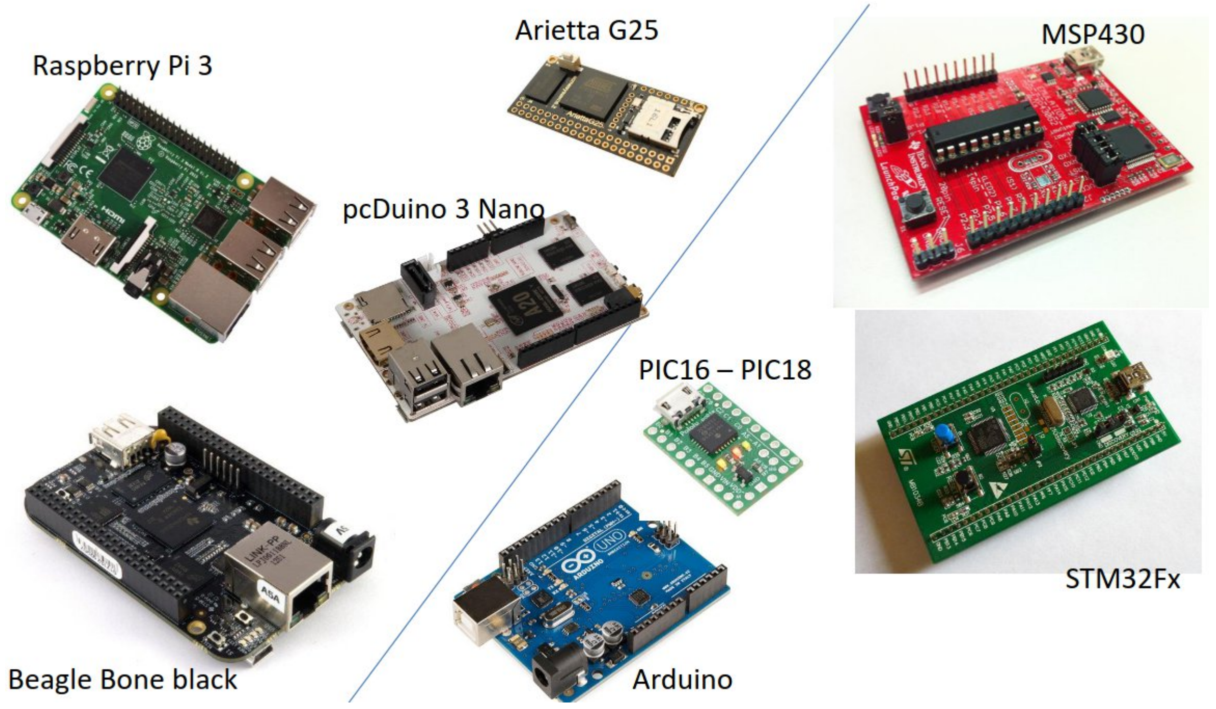
Evolution du C++ (pour l'embarqué)

damien.marchal@univ-lille.fr

The sub-culture of embeded developpement



An **embedded system** is a specialized computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electronic system.[1][2] It is embedded as part of a complete device often including electrical or electronic hardware and mechanical parts. Because an embedded system typically controls



Extrait : Programming Embedded Systems with C/C++ (Thomas Grenier)

WHO AM I

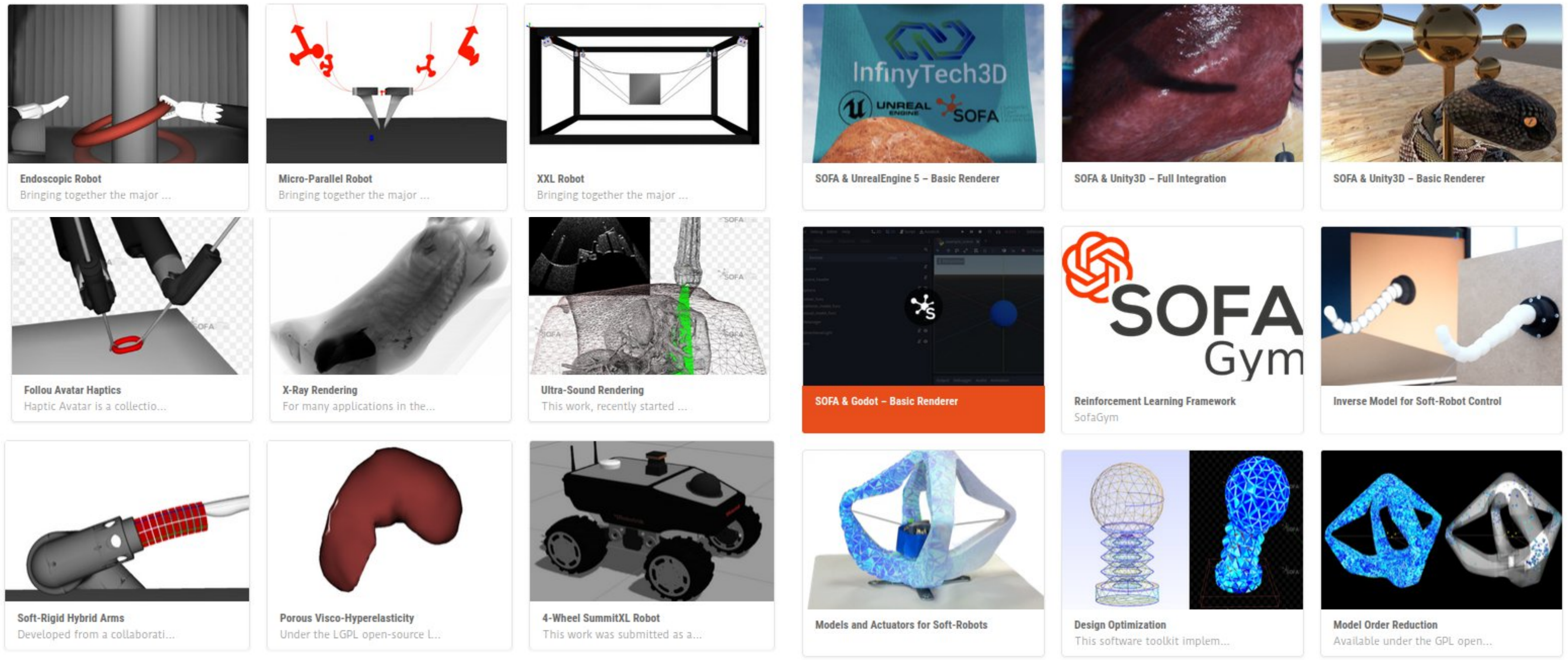
- Damien Marchal,
- Software developper
- Research engineer at PADR/CRIStAL
- Developping and maintaining c++ and python softwares

  Sofa and its Soft-Robots plugin eco-system

SOFA

A framework for interactive multi-physics simulation

https://www.sofa-framework.org/



Endoscopic Robot
Bringing together the major ...

Micro-Parallel Robot
Bringing together the major ...

XXL Robot
Bringing together the major ...

SOFA & UnrealEngine 5 – Basic Renderer

SOFA & Unity3D – Full Integration

SOFA & Unity3D – Basic Renderer

Follou Avatar Haptics
Haptic Avatar is a collectio...

X-Ray Rendering
For many applications in the...

Ultra-Sound Rendering
This work, recently started ...

SOFA & Godot – Basic Renderer

Reinforcement Learning Framework
SofaGym

Inverse Model for Soft-Robot Control

Soft-Rigid Hybrid Arms
Developed from a collaborati...

Porous Visco-Hyperelasticity
Under the LGPL open-source L...

4-Wheel SummitXL Robot
This work was submitted as a...

Models and Actuators for Soft-Robots

Design Optimization
This software toolkit implem...

Model Order Reduction
Available under the GPL open...

SOFA

A very big and old code base (a least 2005)

- influenced by language evolution (c++98, 03,11,14,17,20,...)

c++14

c++11

c++03

c++98

c

Archeology

SOFA

A very big and old code base (a least 2005)

- influenced by language evolution (c++98, 03,11,14,17,20,...)
- different developpement teams and technical direction,
- different target (eg : m. thesis or phd. thesis work, publication,company)
- having different coding style

c++14

c++11

c++03

c++98

c

Archeology

Anthropology

SOFA

A very big and old code base (a least 2005)

- influenced by language evolution (c++98, 03,11,14,17,20,...)
- different developpement teams and technical direction,
- different target (eg : m. thesis or phd. thesis work, publication,company)
- having different coding style

c++14

c++11

c++03

c++98

c

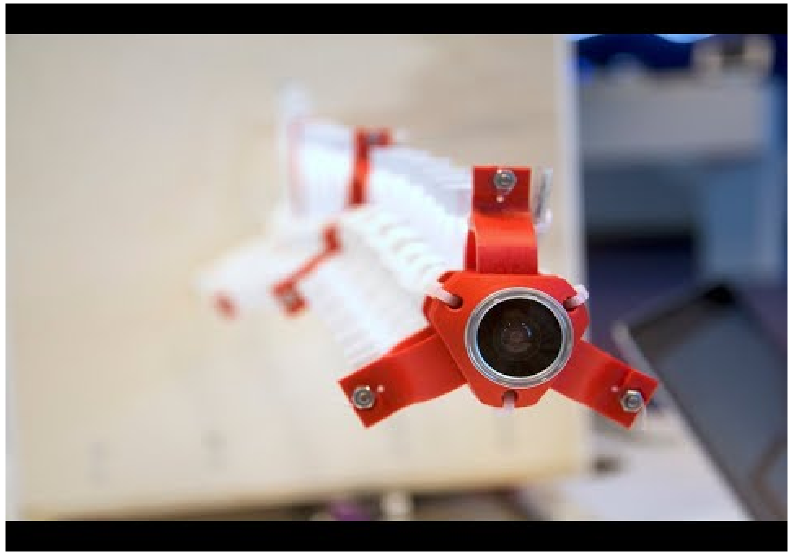
Archeology

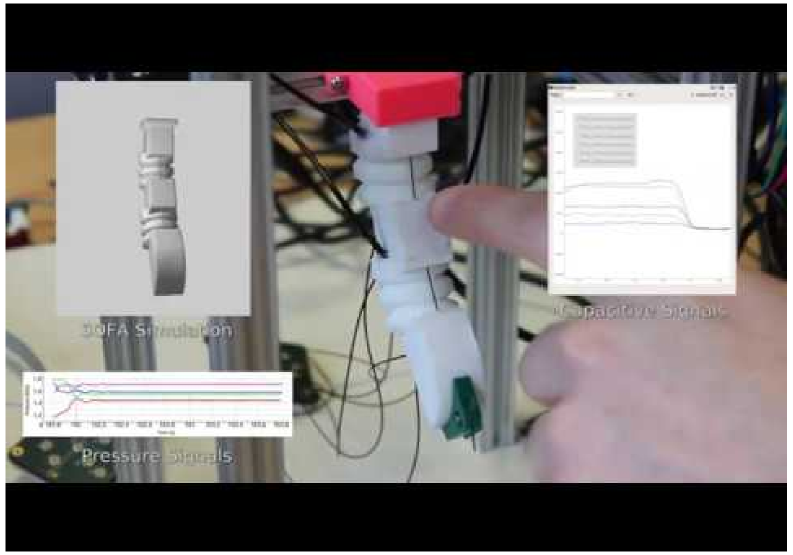
Anthropology

Maintaining & upgrading

At DEFROST (https://www.defrost.inria.fr/)

We creates soft robots and controls them by SOFA.





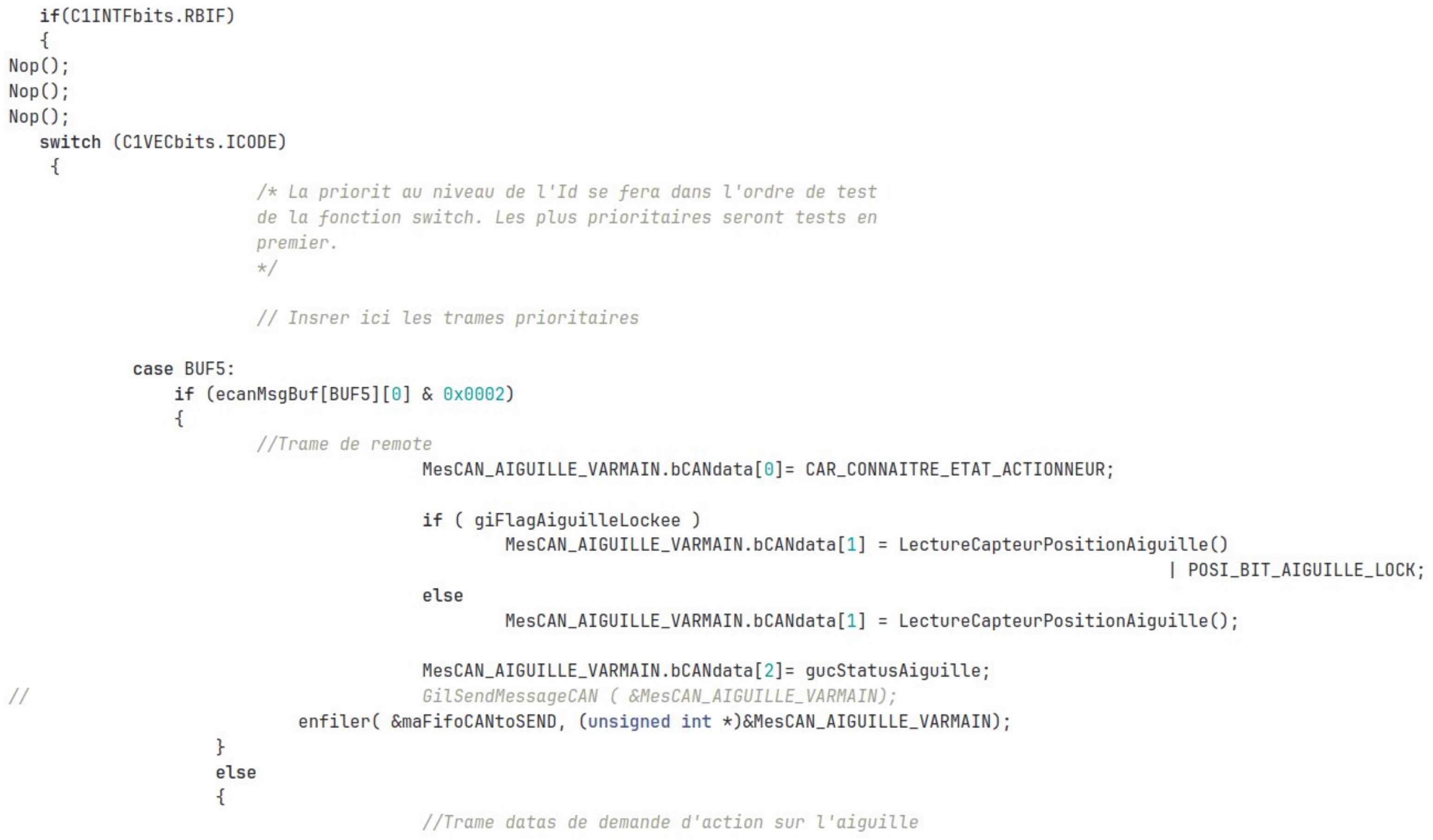
We strongly rely on small computing plateforms for doing the interface to the hardware (motors/sensors...),
Compared to SW, the embedded prototypes are often developped by hw engineers

Typical embeded code we have

« Good » old C

C++ with C style

full c++ style



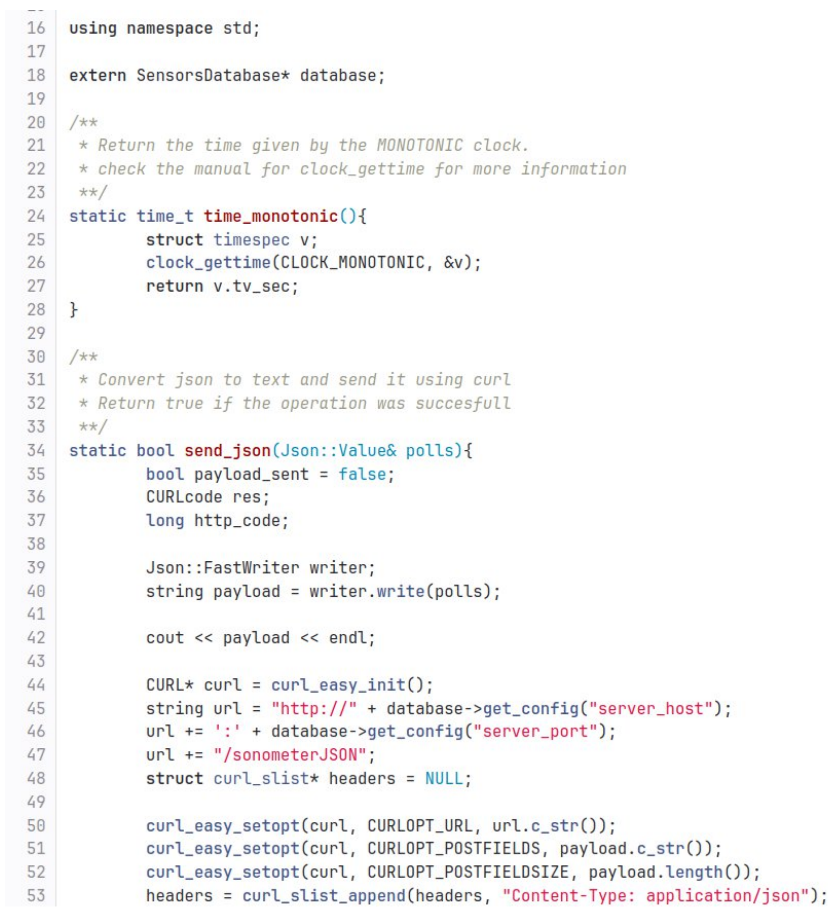
```
   if(C1INTFbits.RBIF)
   {
Nop();
Nop();
Nop();
   switch (C1VECbits.ICODE)
    {
                    /* La priorit au niveau de l'Id se fera dans l'ordre de test
                    de la fonction switch. Les plus prioritaires seront tests en
                    premier.
                    */

                    // Insrer ici les trames prioritaires

           case BUF5:
               if (ecanMsgBuf[BUF5][0] & 0x0002)
               {
                    //Trame de remote
                                   MesCAN_AIGUILLE_VARMAIN.bCANdata[0]= CAR_CONNAITRE_ETAT_ACTIONNEUR;

                                   if ( giFlagAiguilleLockee )
                                           MesCAN_AIGUILLE_VARMAIN.bCANdata[1] = LectureCapteurPositionAiguille()
                                                                                                  | POSI_BIT_AIGUILLE_LOCK;
                                   else
                                           MesCAN_AIGUILLE_VARMAIN.bCANdata[1] = LectureCapteurPositionAiguille();

                                   MesCAN_AIGUILLE_VARMAIN.bCANdata[2]= gucStatusAiguille;
//                                 GilSendMessageCAN ( &MesCAN_AIGUILLE_VARMAIN);
                            enfiler( &maFifoCANtoSEND, (unsigned int *)&MesCAN_AIGUILLE_VARMAIN);
                   }
                   else
                   {
                                   //Trame datas de demande d'action sur l'aiguille
```

Typical embeded code we have

« Good » old C

C++ with C style

full c++ style



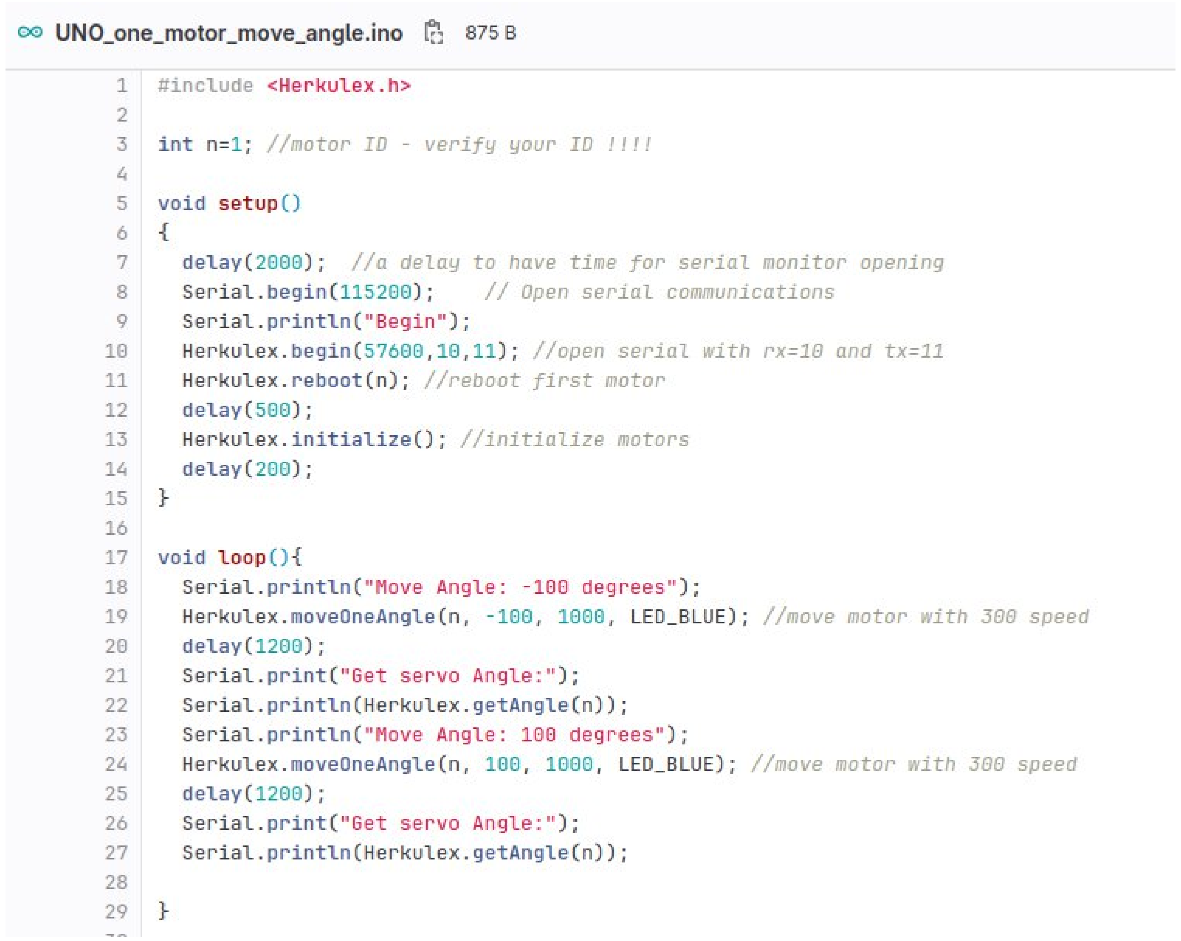
```
using namespace std;

extern SensorsDatabase* database;

/**
 * Return the time given by the MONOTONIC clock.
 * check the manual for clock_gettime for more information
 **/
static time_t time_monotonic(){
        struct timespec v;
        clock_gettime(CLOCK_MONOTONIC, &v);
        return v.tv_sec;
}

/**
 * Convert json to text and send it using curl
 * Return true if the operation was succesfull
 **/
static bool send_json(Json::Value& polls){
        bool payload_sent = false;
        CURLcode res;
        long http_code;

        Json::FastWriter writer;
        string payload = writer.write(polls);

        cout << payload << endl;

        CURL* curl = curl_easy_init();
        string url = "http://" + database->get_config("server_host");
        url += ':' + database->get_config("server_port");
        url += "/sonometerJSON";
        struct curl_slist* headers = NULL;

        curl_easy_setopt(curl, CURLOPT_URL, url.c_str());
        curl_easy_setopt(curl, CURLOPT_POSTFIELDS, payload.c_str());
        curl_easy_setopt(curl, CURLOPT_POSTFIELDSIZE, payload.length());
        headers = curl_slist_append(headers, "Content-Type: application/json");
```

Typical embeded code we have

« Good » old C

C++ with C style

full c++ style


UNO_one_motor_move_angle.ino 875 B

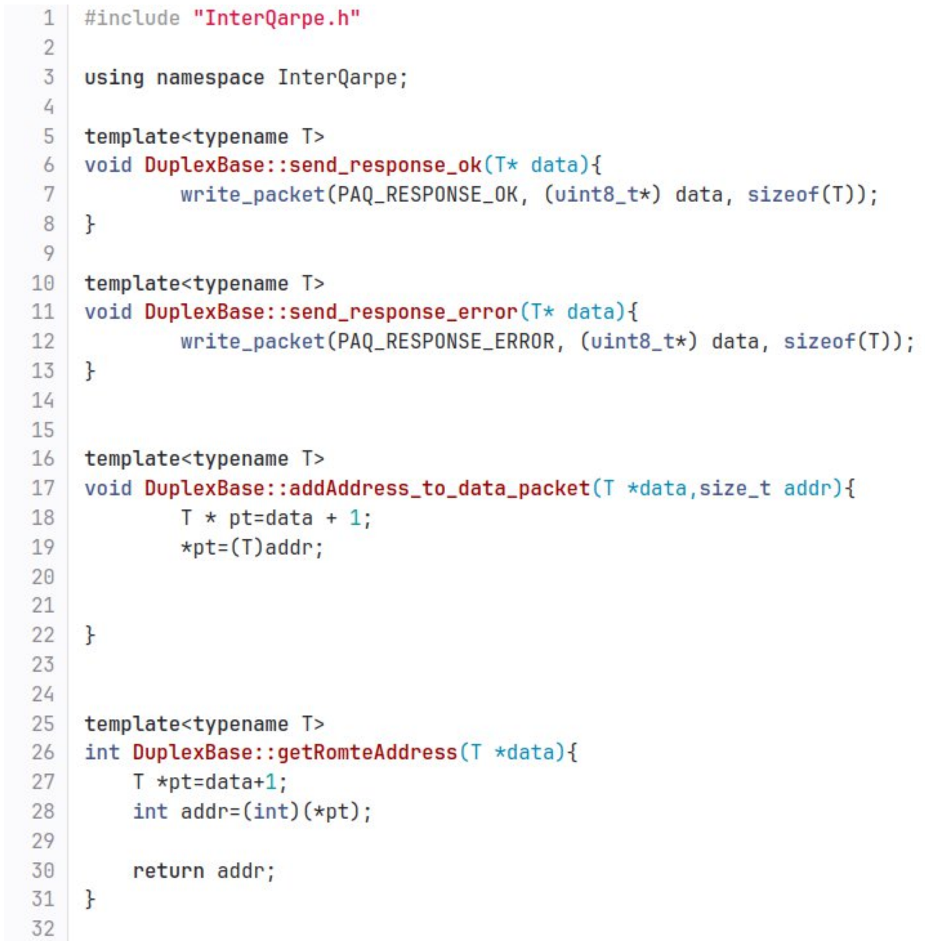
```
#include <Herkulex.h>

int n=1; //motor ID - verify your ID !!!!

void setup()
{
  delay(2000);  //a delay to have time for serial monitor opening
  Serial.begin(115200);    // Open serial communications
  Serial.println("Begin");
  Herkulex.begin(57600,10,11); //open serial with rx=10 and tx=11
  Herkulex.reboot(n); //reboot first motor
  delay(500);
  Herkulex.initialize(); //initialize motors
  delay(200);
}

void loop(){
  Serial.println("Move Angle: -100 degrees");
  Herkulex.moveOneAngle(n, -100, 1000, LED_BLUE); //move motor with 300 speed
  delay(1200);
  Serial.print("Get servo Angle:");
  Serial.println(Herkulex.getAngle(n));
  Serial.println("Move Angle: 100 degrees");
  Herkulex.moveOneAngle(n, 100, 1000, LED_BLUE); //move motor with 300 speed
  delay(1200);
  Serial.print("Get servo Angle:");
  Serial.println(Herkulex.getAngle(n));

}
```

Typical embeded code we have

« Good » old C

C++ with C style

full c++ style

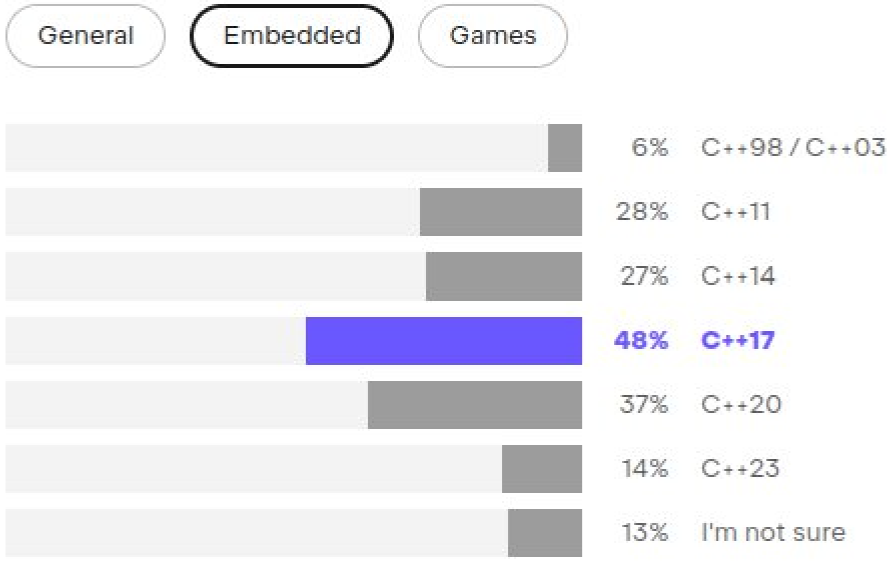


```cpp
#include "InterQarpe.h"

using namespace InterQarpe;

template<typename T>
void DuplexBase::send_response_ok(T* data){
        write_packet(PAQ_RESPONSE_OK, (uint8_t*) data, sizeof(T));
}

template<typename T>
void DuplexBase::send_response_error(T* data){
        write_packet(PAQ_RESPONSE_ERROR, (uint8_t*) data, sizeof(T));
}


template<typename T>
void DuplexBase::addAddress_to_data_packet(T *data,size_t addr){
        T * pt=data + 1;
        *pt=(T)addr;


}


template<typename T>
int DuplexBase::getRomteAddress(T *data){
    T *pt=data+1;
    int addr=(int)(*pt);

    return addr;
}

```

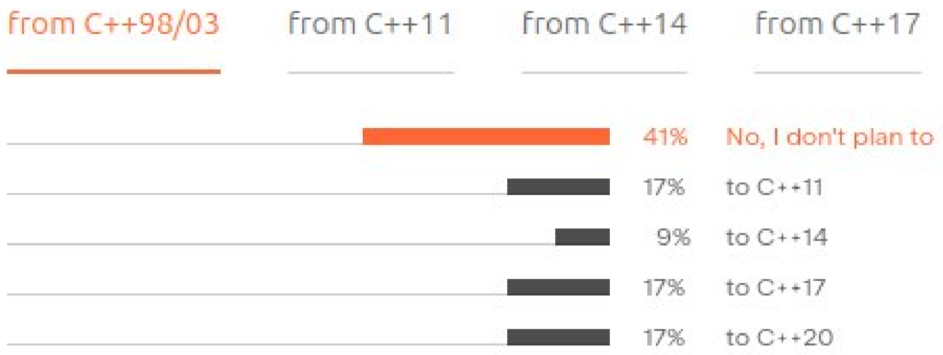
Small poll

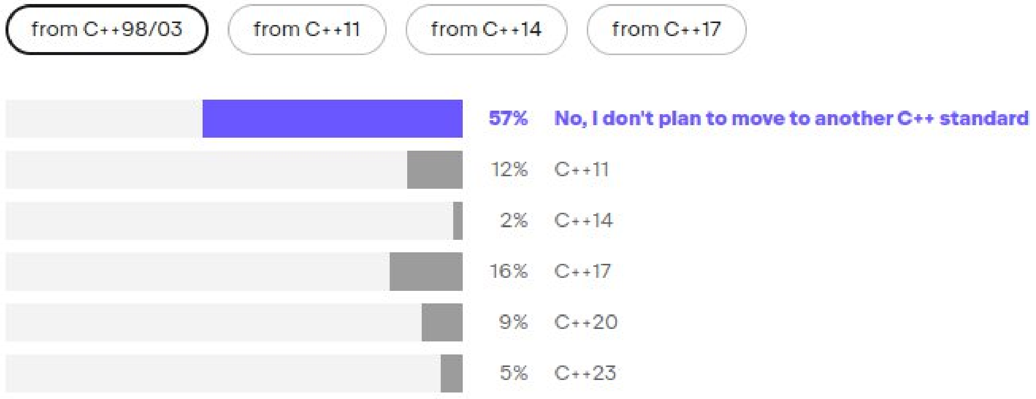
Current trends in C++ for embedded

https://www.jetbrains.com/lp/devecosystem-2023/



C++ standards migration

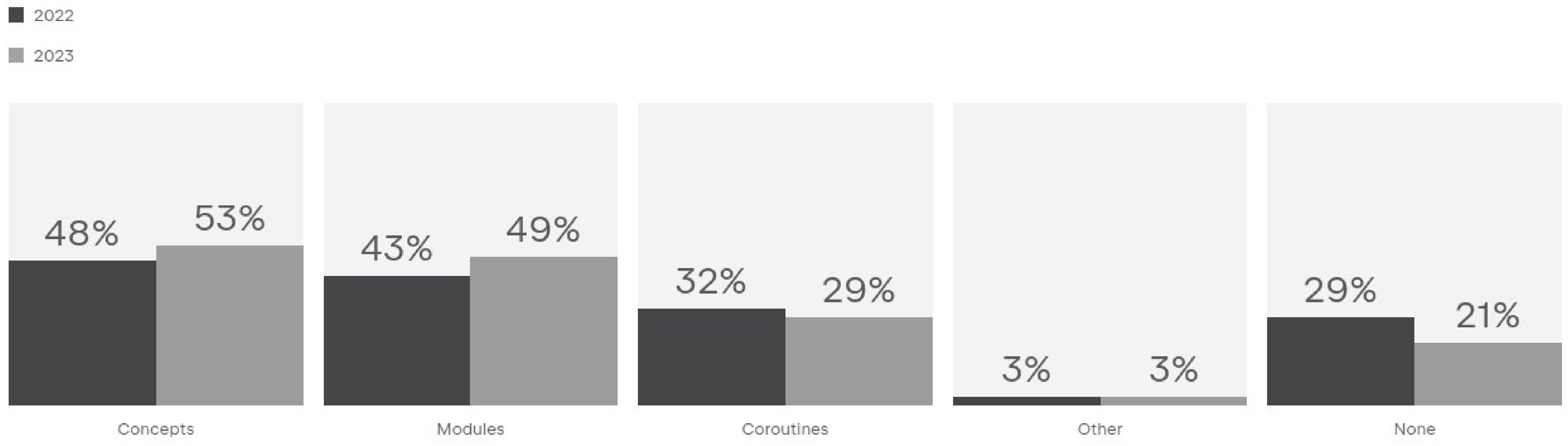




Current trends in C++ for embedded

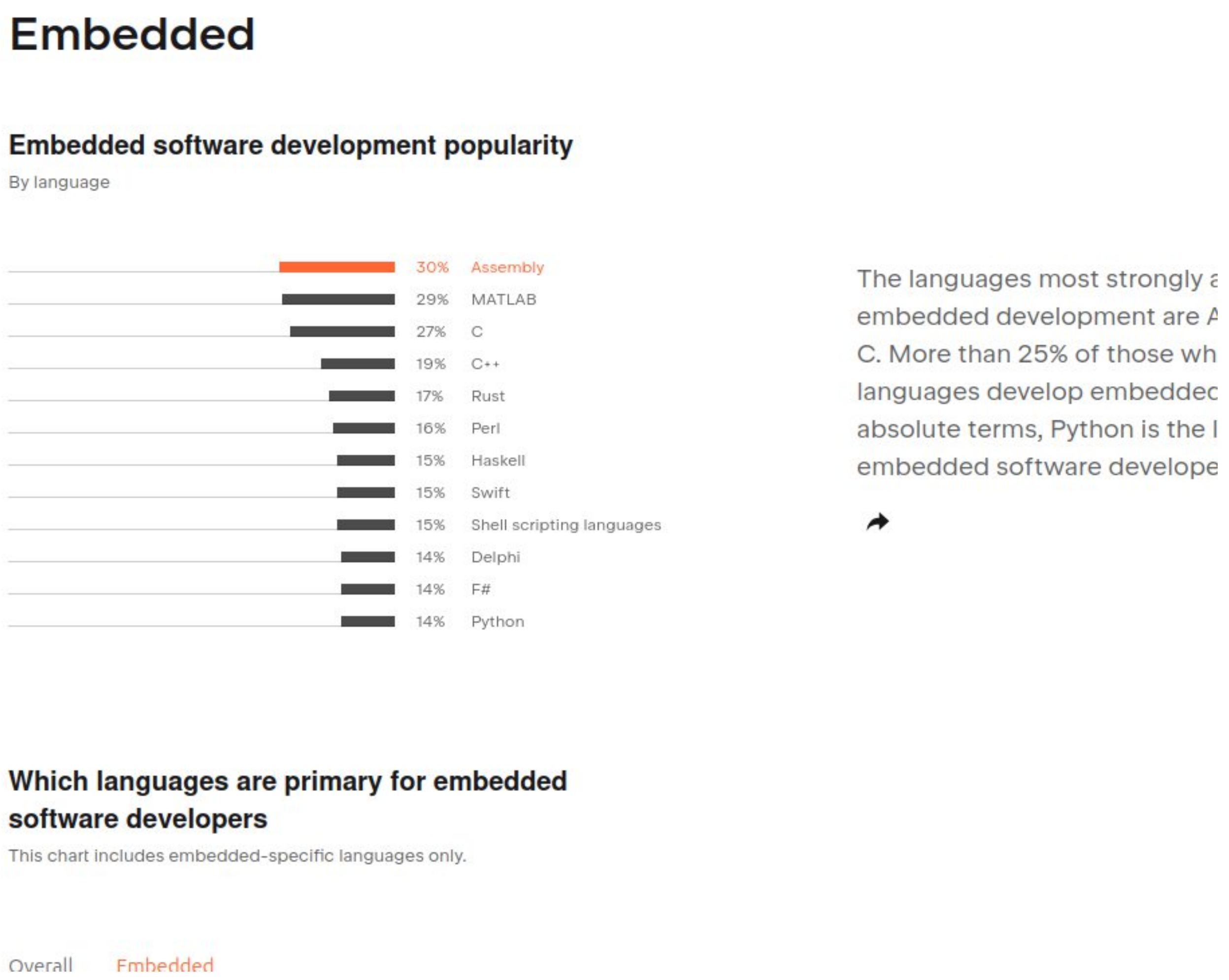
https://www.jetbrains.com/lp/devecosystem-2023/

Is your current project planning to use any of the following C++20 features in the next 12 months?



Current trends in C++ for embedded

https://www.jetbrains.com/lp/devecosystem-2021/embedded/



Embedded

Embedded software development popularity

By language

30%	Assembly
29%	MATLAB
27%	C
19%	C++
17%	Rust
16%	Perl
15%	Haskell
15%	Swift
15%	Shell scripting languages
14%	Delphi
14%	F#
14%	Python

The languages most strongly a
embedded development are A
C. More than 25% of those wh
languages develop embeddec
absolute terms, Python is the l
embedded software develope

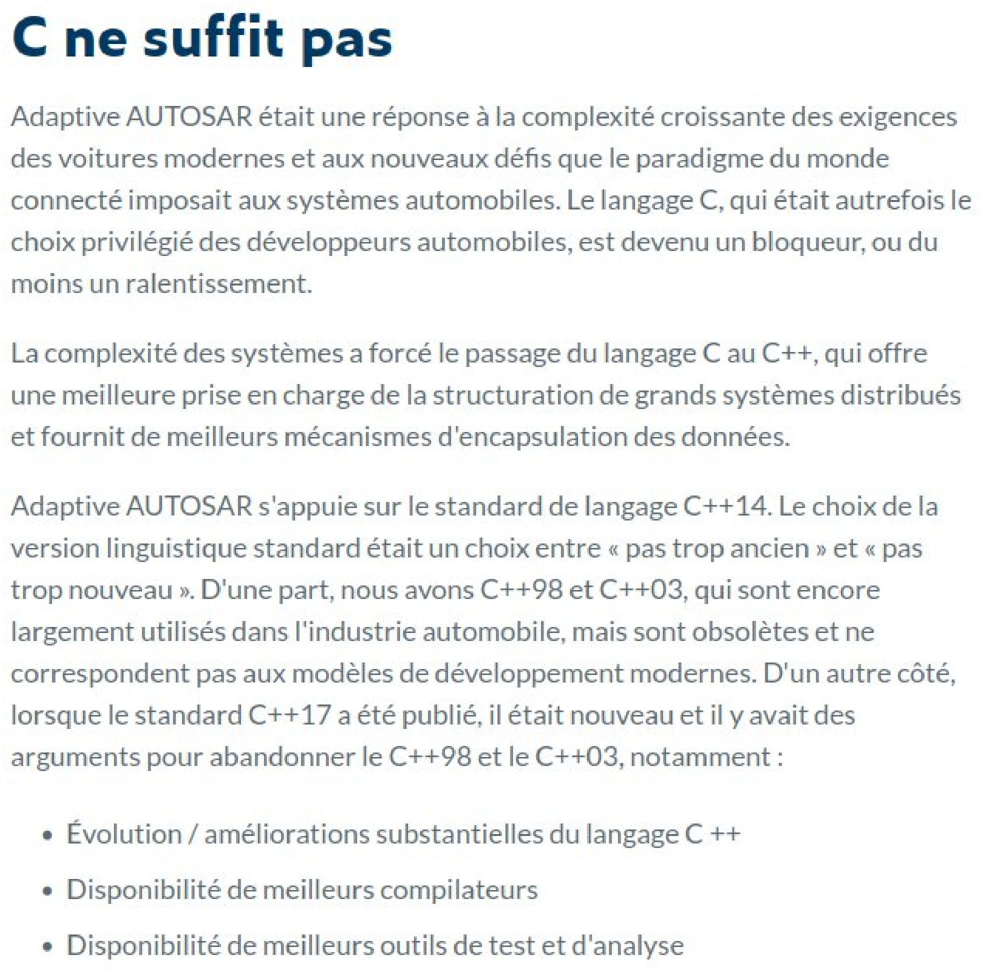
Which languages are primary for embedded software developers

This chart includes embedded-specific languages only.

Overall Embedded

Current trends in C++ for embedded

https://www.jetbrains.com/lp/devecosystem-2021/embedded/



C ne suffit pas

Adaptive AUTOSAR était une réponse à la complexité croissante des exigences des voitures modernes et aux nouveaux défis que le paradigme du monde connecté imposait aux systèmes automobiles. Le langage C, qui était autrefois le choix privilégié des développeurs automobiles, est devenu un bloqueur, ou du moins un ralentissement.

La complexité des systèmes a forcé le passage du langage C au C++, qui offre une meilleure prise en charge de la structuration de grands systèmes distribués et fournit de meilleurs mécanismes d'encapsulation des données.

Adaptive AUTOSAR s'appuie sur le standard de langage C++14. Le choix de la version linguistique standard était un choix entre « pas trop ancien » et « pas trop nouveau ». D'une part, nous avons C++98 et C++03, qui sont encore largement utilisés dans l'industrie automobile, mais sont obsolètes et ne correspondent pas aux modèles de développement modernes. D'un autre côté, lorsque le standard C++17 a été publié, il était nouveau et il y avait des arguments pour abandonner le C++98 et le C++03, notamment :

- Évolution / améliorations substantielles du langage C ++
- Disponibilité de meilleurs compilateurs
- Disponibilité de meilleurs outils de test et d'analyse

Summary on trends

Modern c++ with at least c++20/23 features
(and there is good reason for that : safety by opt-in, abstraction with zero overhead, expressivity, backward compatibility, !complexity)

Python as side language,
(and there is also good reason for that)

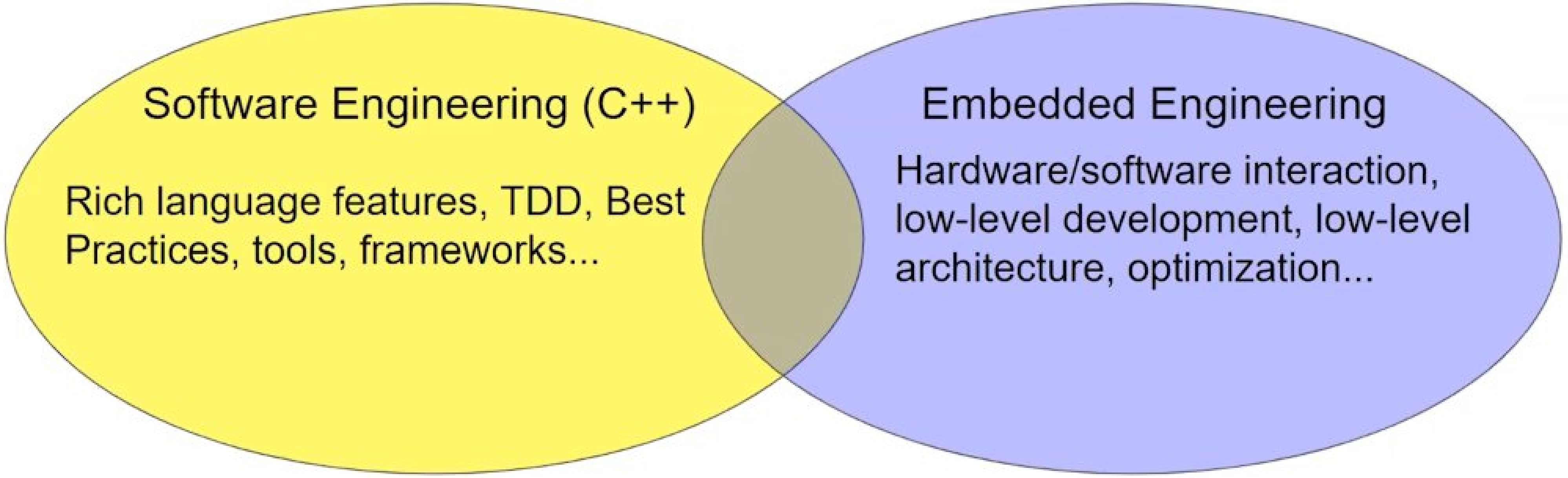
Rust as a challenger
(safety by default, abstraction with zero overhead, !learning curve)

Close to the SOFA code base,

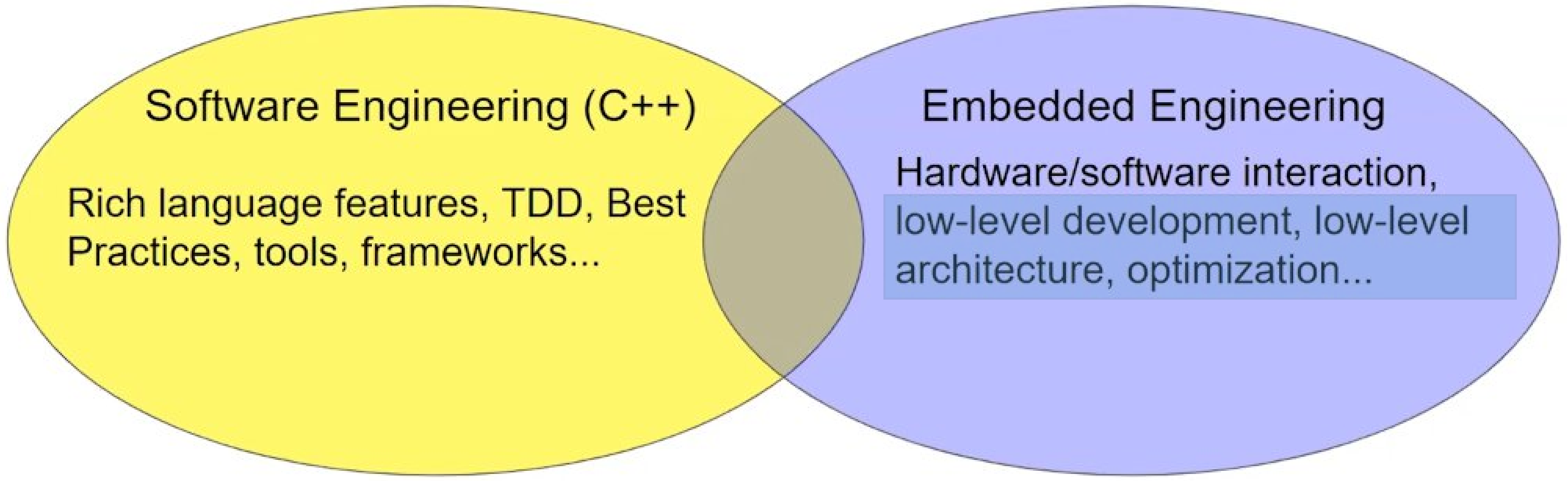
But far away from our « embeded » code base and practices,

Additional thinking :
- selection bias on people who answer' jetbrain polls (need to check the methodology)
- modern c++ may be pulled by the inscreased complexity of system to buid

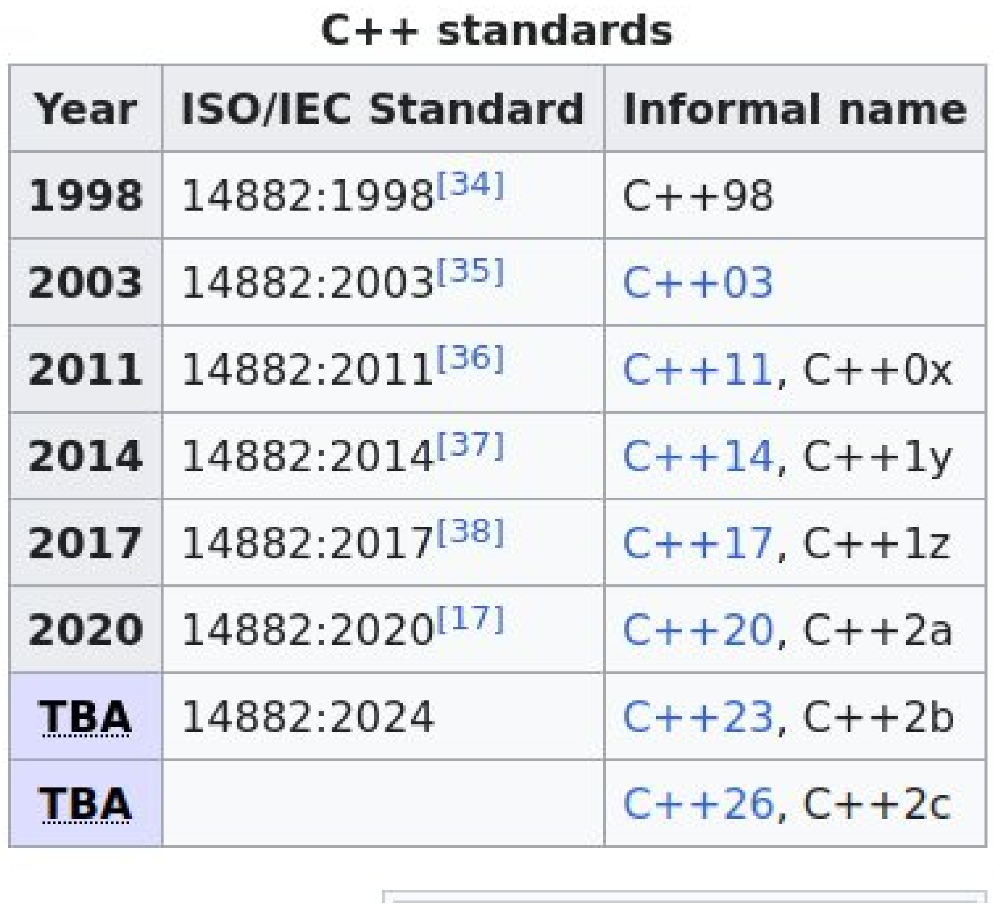
The remaining of the presentation


Software Engineering (C++)
Rich language features, TDD, Best Practices, tools, frameworks...
Embedded Engineering
Hardware/software interaction, low-level development, low-level architecture, optimization...

The remaining of the presentation


Software Engineering (C++)
Rich language features, TDD, Best Practices, tools, frameworks...
Embedded Engineering
Hardware/software interaction, low-level development, low-level architecture, optimization...

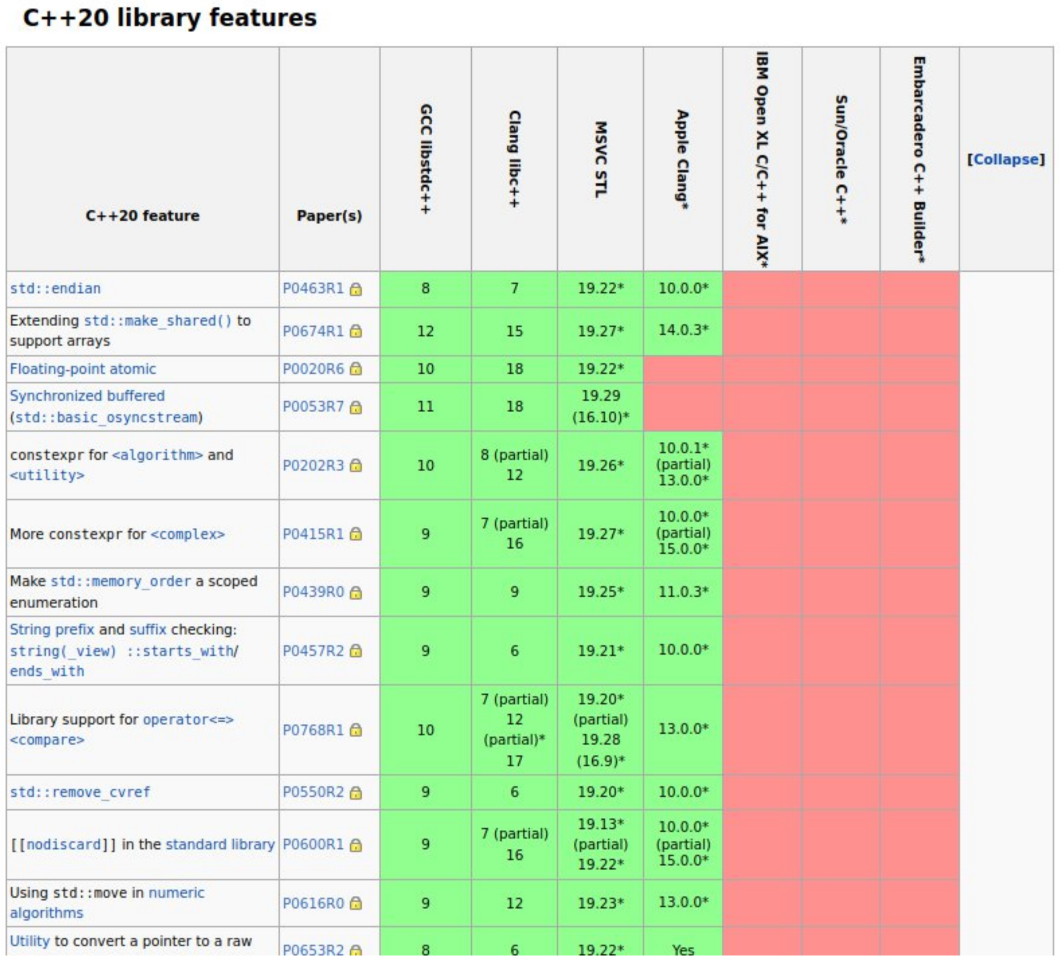
C++ evolutions



C++ standards

Year	ISO/IEC Standard	Informal name
1998	14882:1998[34]	C++98
2003	14882:2003[35]	C++03
2011	14882:2011[36]	C++11, C++0x
2014	14882:2014[37]	C++14, C++1y
2017	14882:2017[38]	C++17, C++1z
2020	14882:2020[17]	C++20, C++2a
TBA	14882:2024	C++23, C++2b
TBA		C++26, C++2c

C++ evolutions



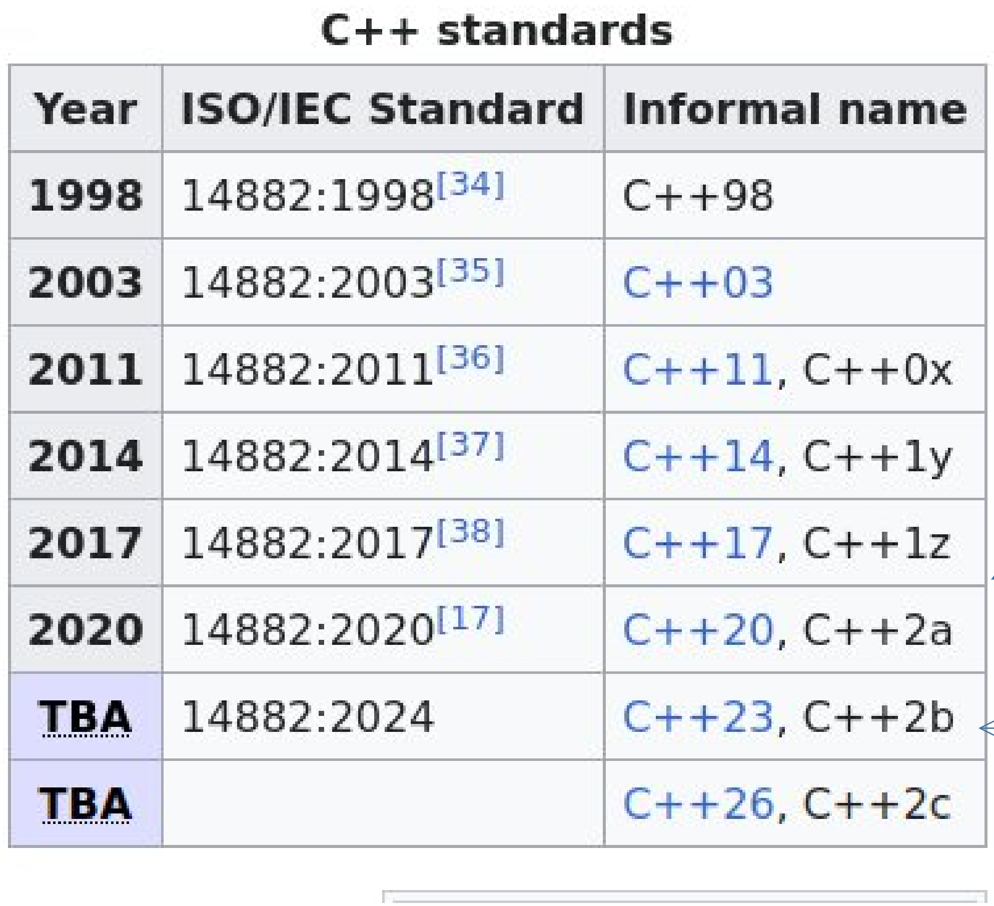
C++20 library features

C++20 feature	Paper(s)	GCC libstdc++	Clang libc++	MSVC STL	Apple Clang*	IBM Open XL C/C++ for AIX*	Sun/Oracle C++*	Embarcadero C++ Builder*	[Collapse]
std::endian	P0463R1	8	7	19.22*	10.0.0*				
Extending std::make_shared() to support arrays	P0674R1	12	15	19.27*	14.0.3*				
Floating-point atomic	P0020R6	10	18	19.22*					
Synchronized buffered (std::basic_osyncstream)	P0053R7	11	18	19.29 (16.10)*					
constexpr for <algorithm> and <utility>	P0202R3	10	8 (partial) 12	19.26*	10.0.1* (partial) 13.0.0*				
More constexpr for <complex>	P0415R1	9	7 (partial) 16	19.27*	10.0.0* (partial) 15.0.0*				
Make std::memory_order a scoped enumeration	P0439R0	9	9	19.25*	11.0.3*				
String prefix and suffix checking: string(_view) ::starts_with/ ends_with	P0457R2	9	6	19.21*	10.0.0*				
Library support for operator<=> <compare>	P0768R1	10	7 (partial) 12 (partial)* 17	19.20* (partial) 19.28 (16.9)*	13.0.0*				
std::remove_cvref	P0550R2	9	6	19.20*	10.0.0*				
[[nodiscard]] in the standard library	P0600R1	9	7 (partial) 16	19.13* (partial) 19.22*	10.0.0* (partial) 15.0.0*				
Using std::move in numeric algorithms	P0616R0	9	12	19.23*	13.0.0*				
Utility to convert a pointer to a raw	P0653R2	8	6	19.22*	Yes				

c++ language
c++ standard libraries
(everything you get with #include or *import*)

https://en.cppreference.com/w/cpp/compiler_support

C++ evolutions



C++ standards

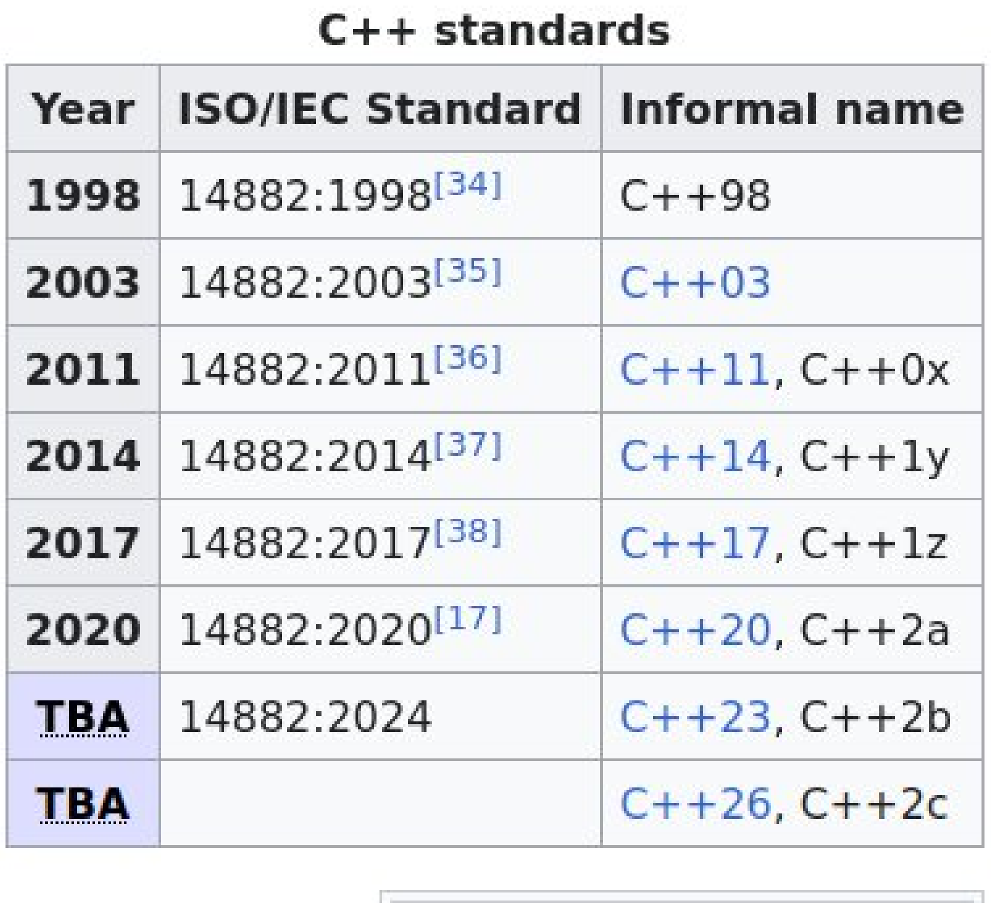
Year	ISO/IEC Standard	Informal name
1998	14882:1998[34]	C++98
2003	14882:2003[35]	C++03
2011	14882:2011[36]	C++11, C++0x
2014	14882:2014[37]	C++14, C++1y
2017	14882:2017[38]	C++17, C++1z
2020	14882:2020[17]	C++20, C++2a
TBA	14882:2024	C++23, C++2b
TBA		C++26, C++2c

For product

For prototyping in a research lab

https://en.cppreference.com/w/cpp/compiler_support

C++ evolutions



C++ standards

Year	ISO/IEC Standard	Informal name
1998	14882:1998[34]	C++98
2003	14882:2003[35]	C++03
2011	14882:2011[36]	C++11, C++0x
2014	14882:2014[37]	C++14, C++1y
2017	14882:2017[38]	C++17, C++1z
2020	14882:2020[17]	C++20, C++2a
TBA	14882:2024	C++23, C++2b
TBA		C++26, C++2c

C with class

Dark age of template metaprogramming

for & auto & lambda & constexpr

Minor changes & fixes

Threading, Async

Concepts, Modules, Co-routines, std ::expect

https://en.cppreference.com/w/cpp/compiler_support

Let’s go for live examples...

C->C++ : OOP
C++->c++03 : templates
C++03->C++11 : for, auto, ...constexpr

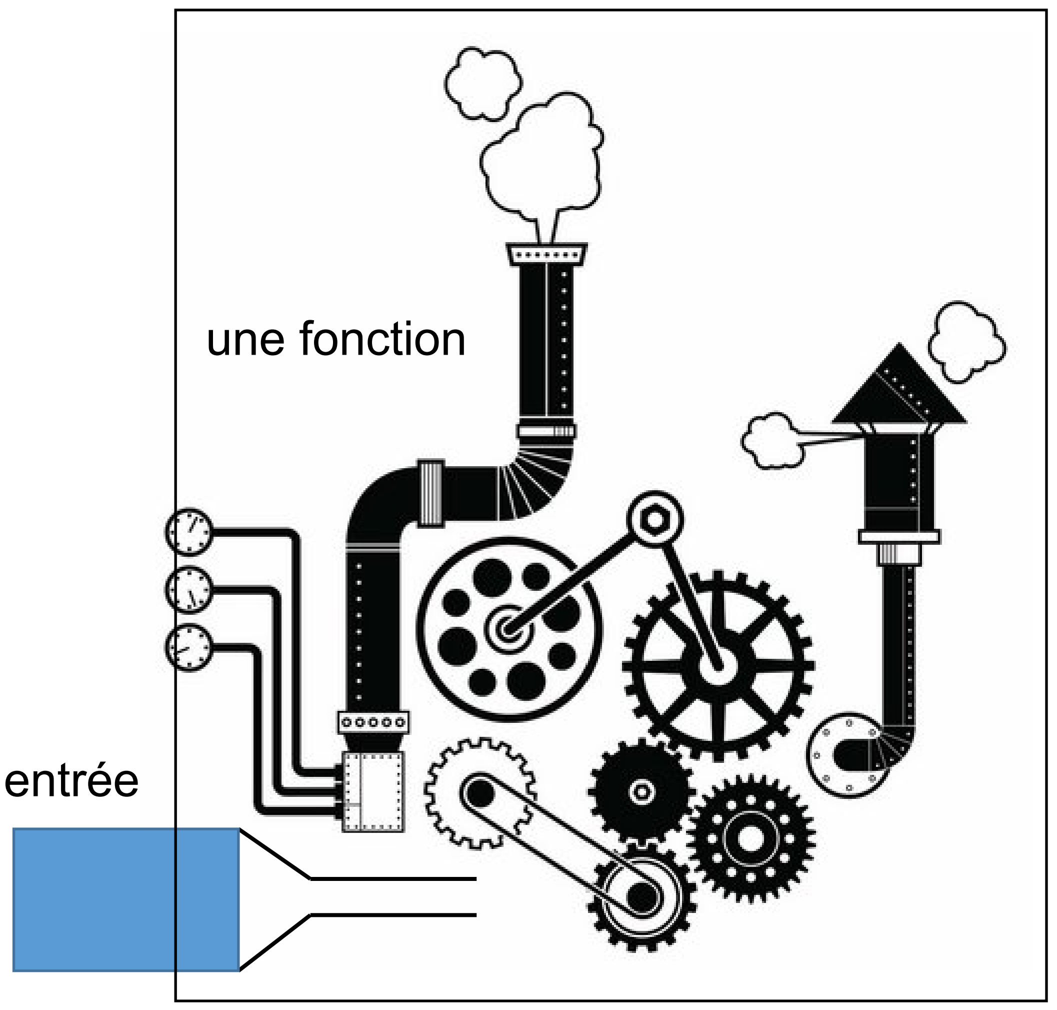
Concepts

Defining restriction on « type » or ....

with

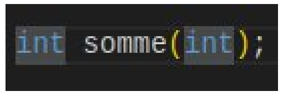
Concepts

Let’s try for an analogy

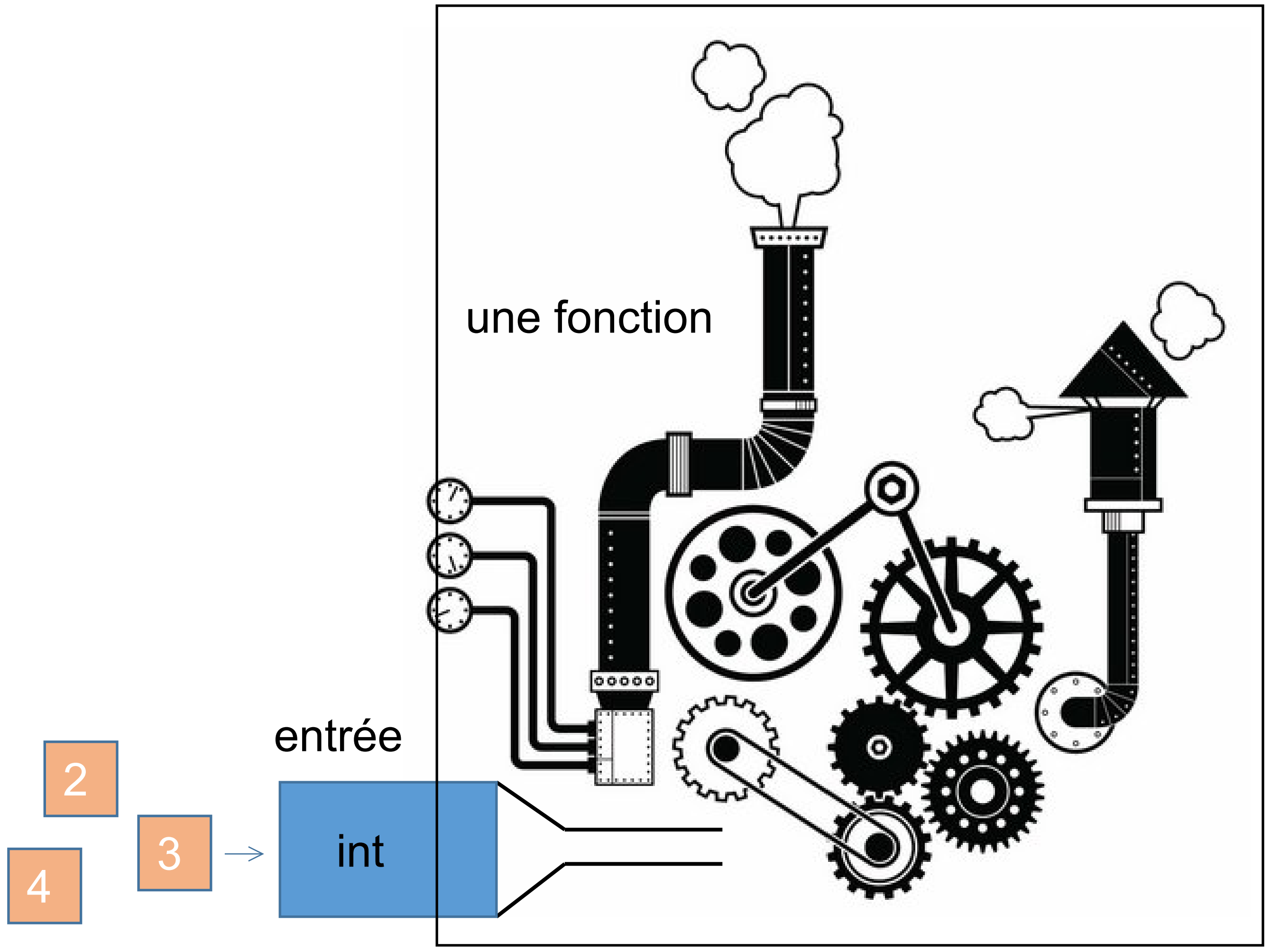


Concepts

une fonction sur un type (à la C)

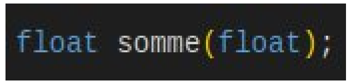


```
int somme(int);
```

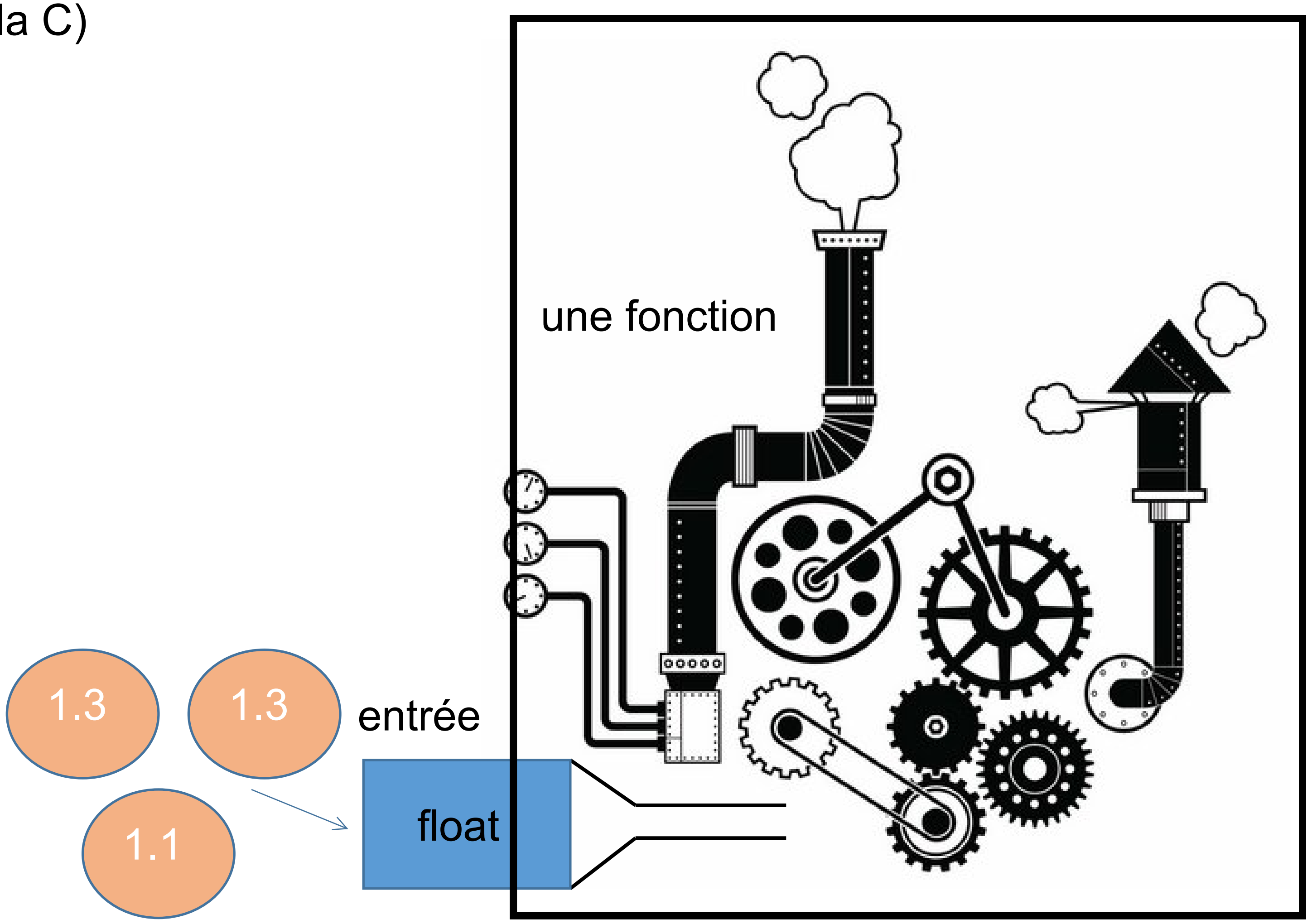


Concepts

une autre fonction sur un type (à la C)

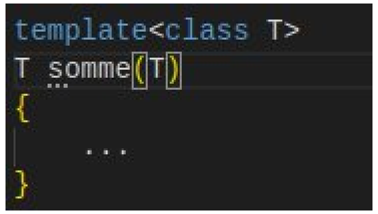


```
float somme(float);
```

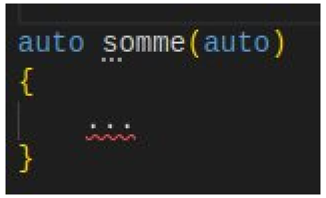


Concepts

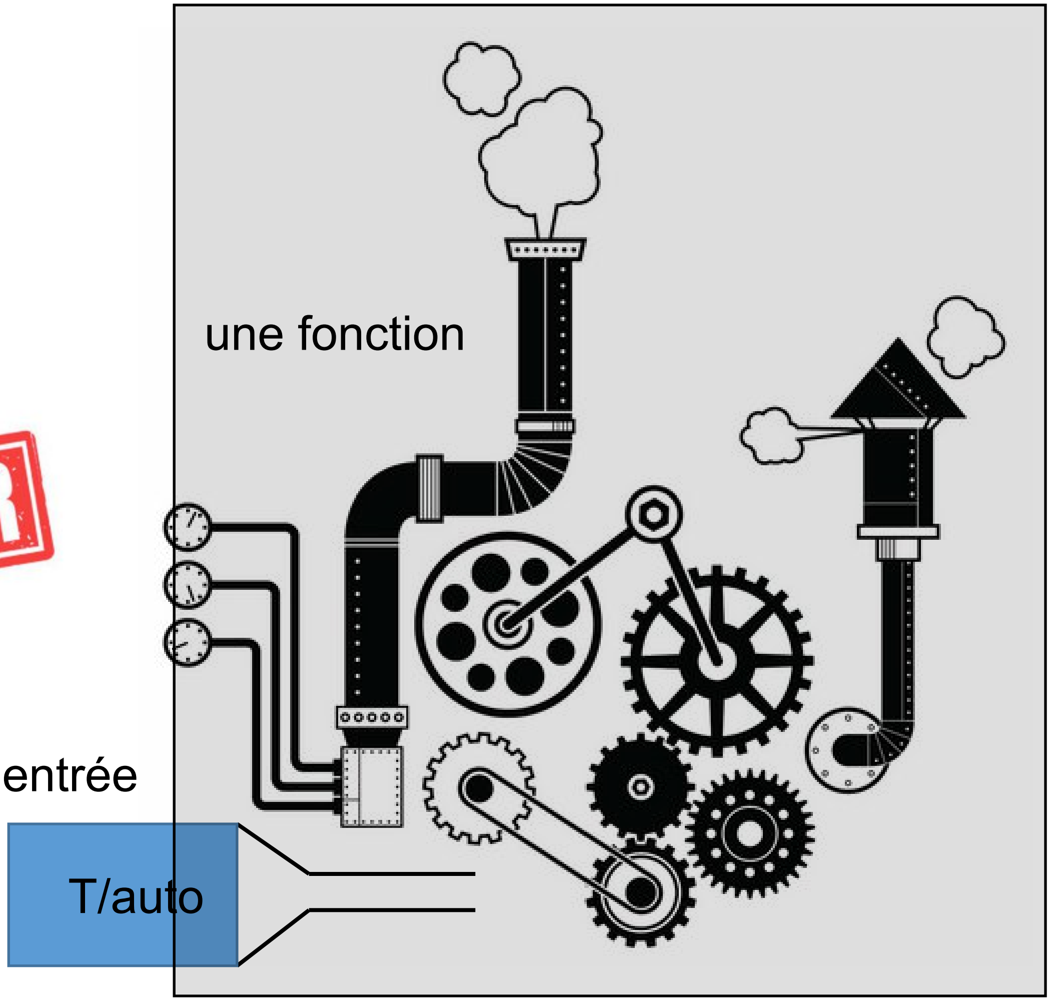
template/auto (c++ 03)



```
template<class T>
T somme(T)
{
    ...
}
```

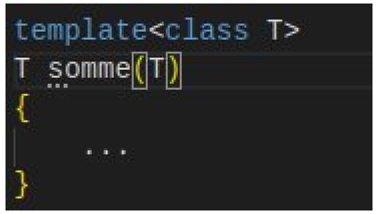


```
auto somme(auto)
{
    ...
}
```

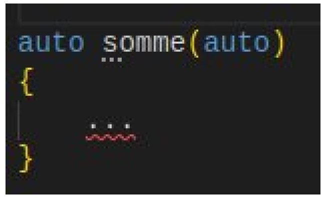


Concepts

template/auto (c++ 03)



```
template<class T>
T somme(T)
{
    ...
}
```

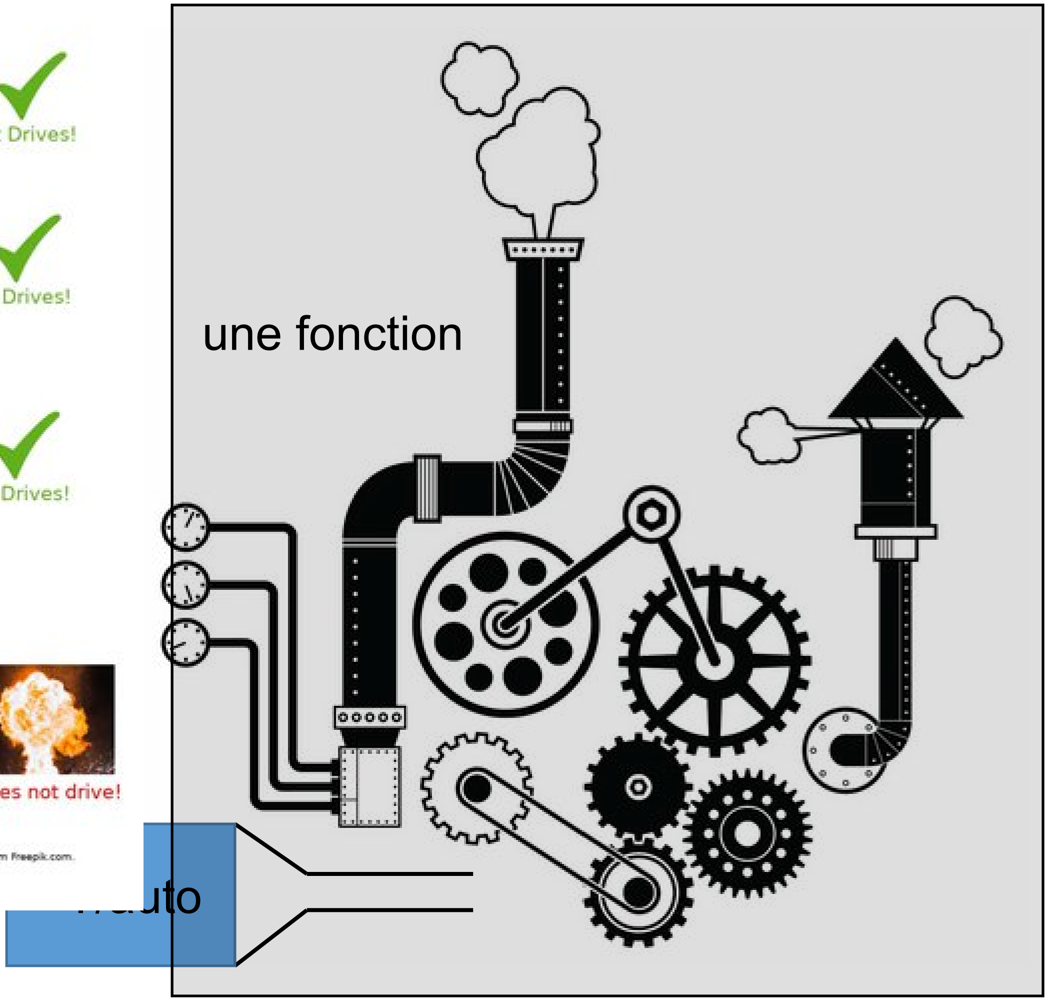


```
auto somme(auto)
{
    ...
}
```

Think of as python

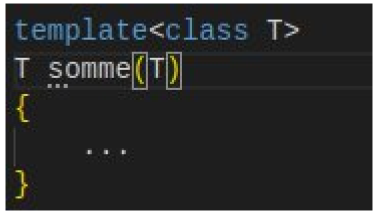
Duck Typing
(by Ben Koshy)
"Drive!"
It Drives!
"Drive!"
It Drives!
"Drive!"
It Drives!
"Drive!"
Does not drive!
Image Attribution:
(1) Magic Wand: image: Freepik.com". This cover has been designed using resources from Freepik.com.
License terms require me to source attribution for the magic wand.
(2) Photos: taken from Unsplash.com, and other images do not require attribution.
My hearty thanks to all those who provide these images for use on the internet.

but at compile time !

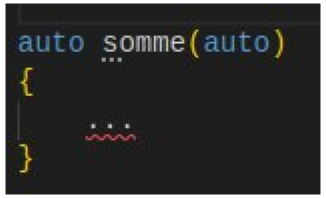


Concepts

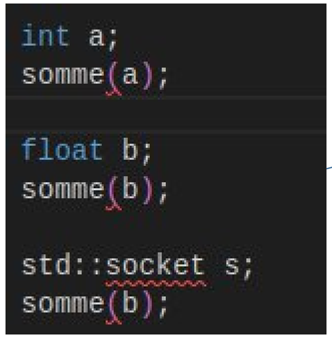
template/auto (c++ 03)



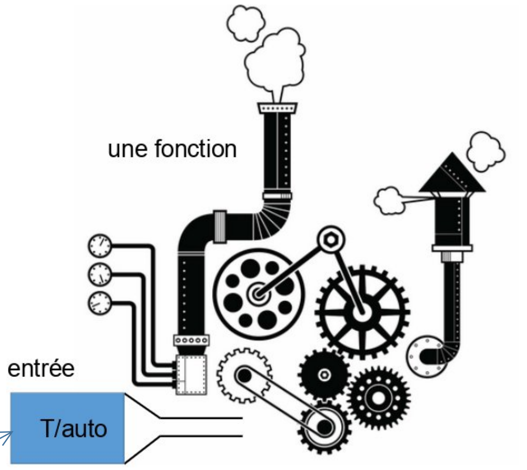
```
template<class T>
T somme(T)
{
    ...
}
```

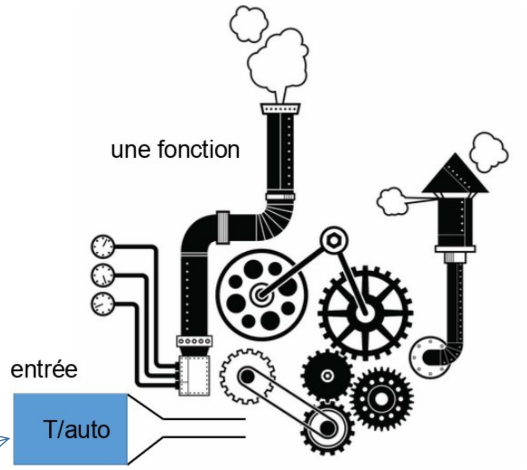


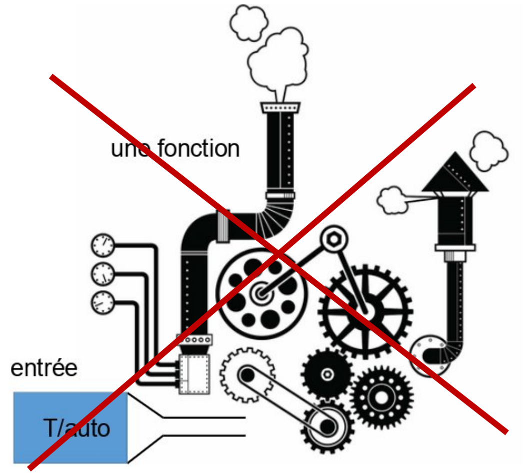
```
auto somme(auto)
{
    ...
}
```



```
int a;
somme(a);

float b;
somme(b);

std::socket s;
somme(b);
```

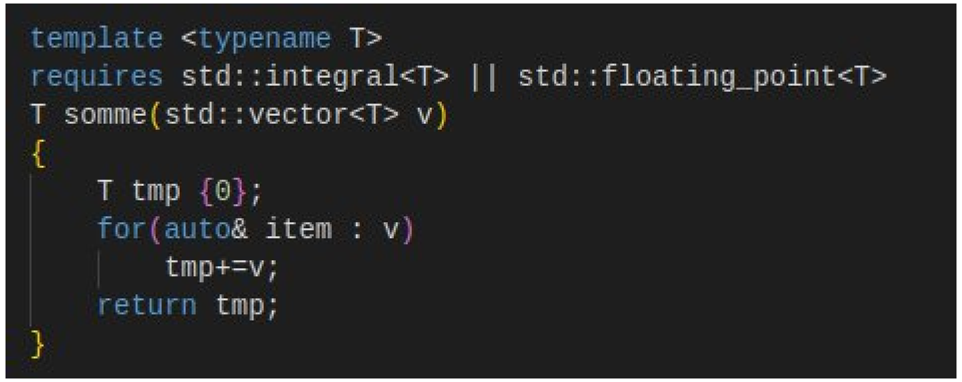




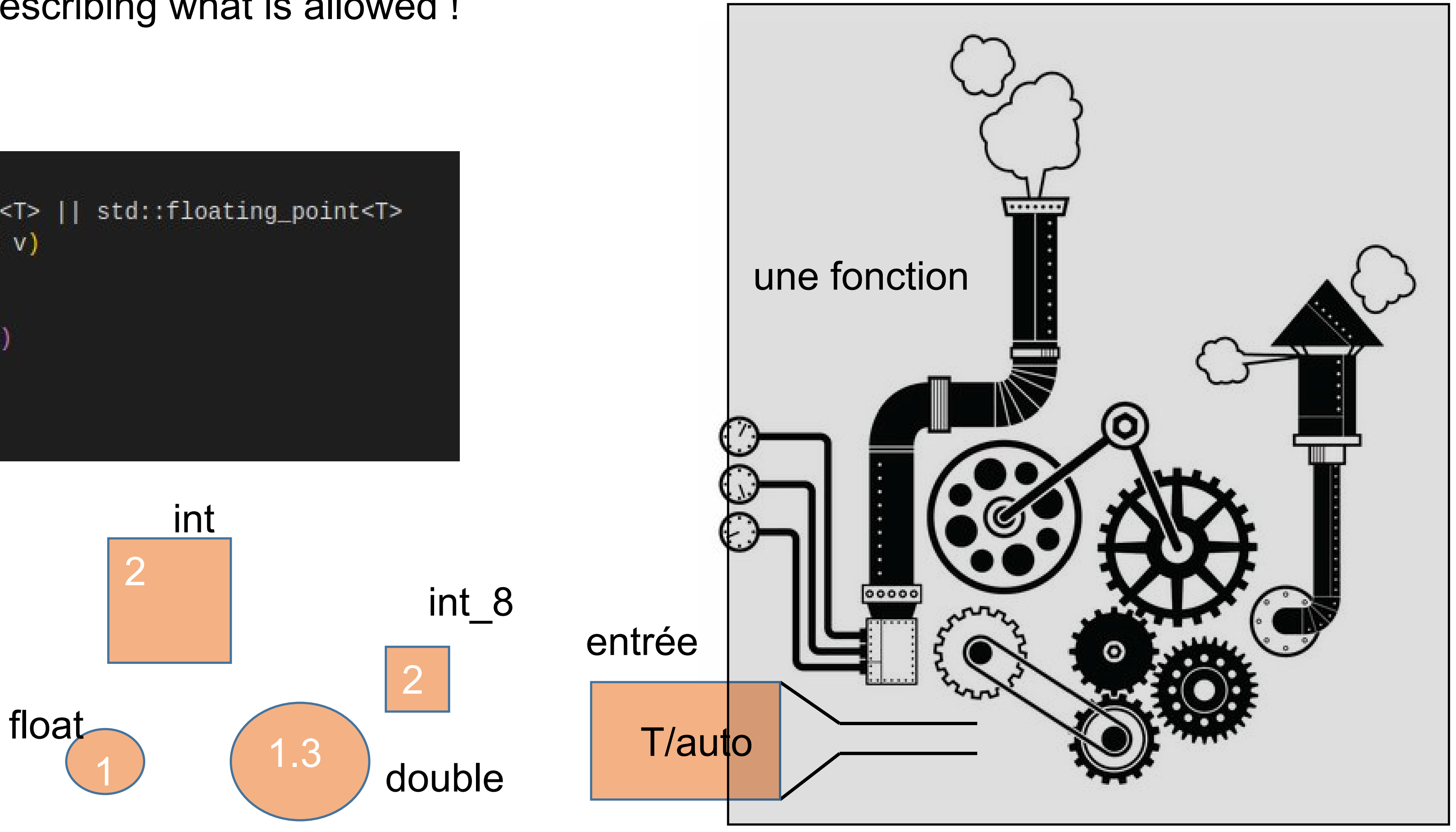


Concepts

create a concept describing what is allowed !

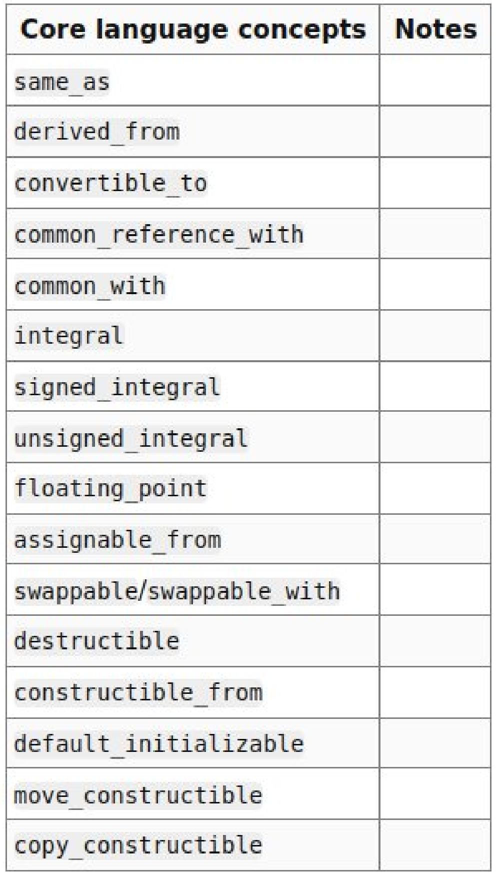


```
template <typename T>
requires std::integral<T> || std::floating_point<T>
T somme(std::vector<T> v)
{
    T tmp {0};
    for(auto& item : v)
        tmp+=v;
    return tmp;
}
```



Concepts

create a concept describing what is allowed !



Core language concepts	Notes
`same_as`	
`derived_from`	
`convertible_to`	
`common_reference_with`	
`common_with`	
`integral`	
`signed_integral`	
`unsigned_integral`	
`floating_point`	
`assignable_from`	
`swappable/swappable_with`	
`destructible`	
`constructible_from`	
`default_initializable`	
`move_constructible`	
`copy_constructible`	

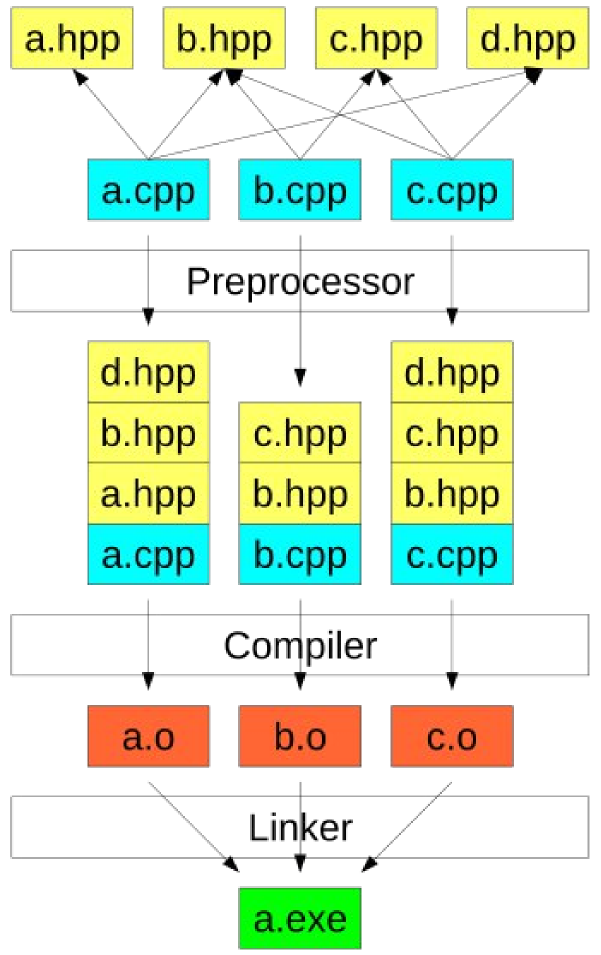
custom concepts....
est iterable
a les méthode XX et YY mais pas ZZ
a un attribut toto
etc...

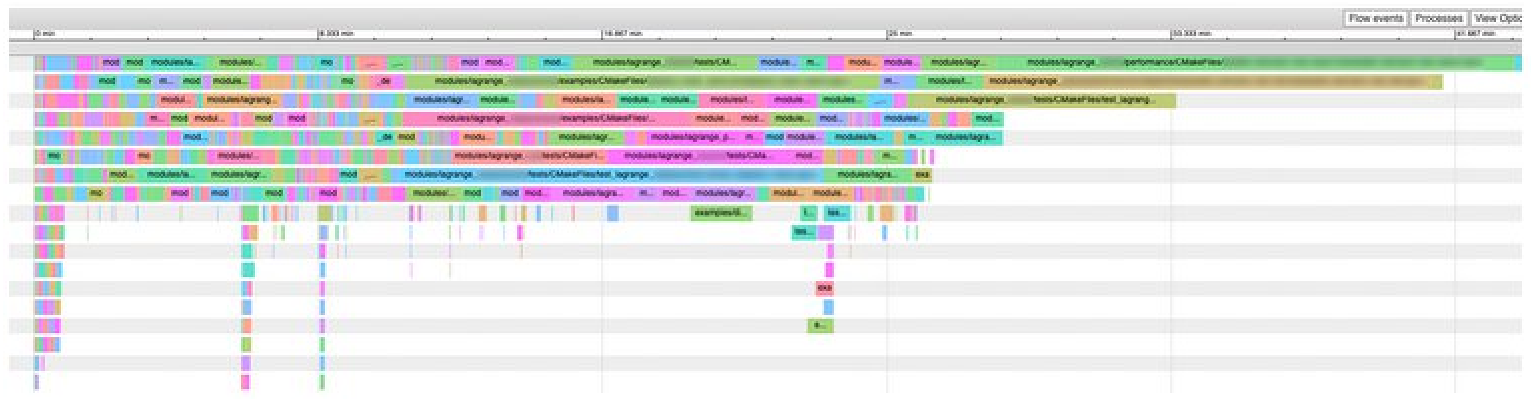
Co-routines & async

Fonctions qui peuvent suspendre leur exécution...

Modules

What happens when you include





```
/////// A.cpp (primary module interface unit of 'A')
export module A;

export char const* hello() { return "hello"; }

/////// B.cpp (primary module interface unit of 'B')
export module B;

export import A;

export char const* world() { return "world"; }

/////// main.cpp (not a module unit)
#include <iostream>
import B;

int main()
{
    std::cout << hello() << ' ' << world() << '\n';
}
```

Bye bye macros.
But interopartion with legacy code using #include and import

Conclusion...

Modern c++ a very good match for highly abstracted code
(reusability, composabilité) with zero overhead

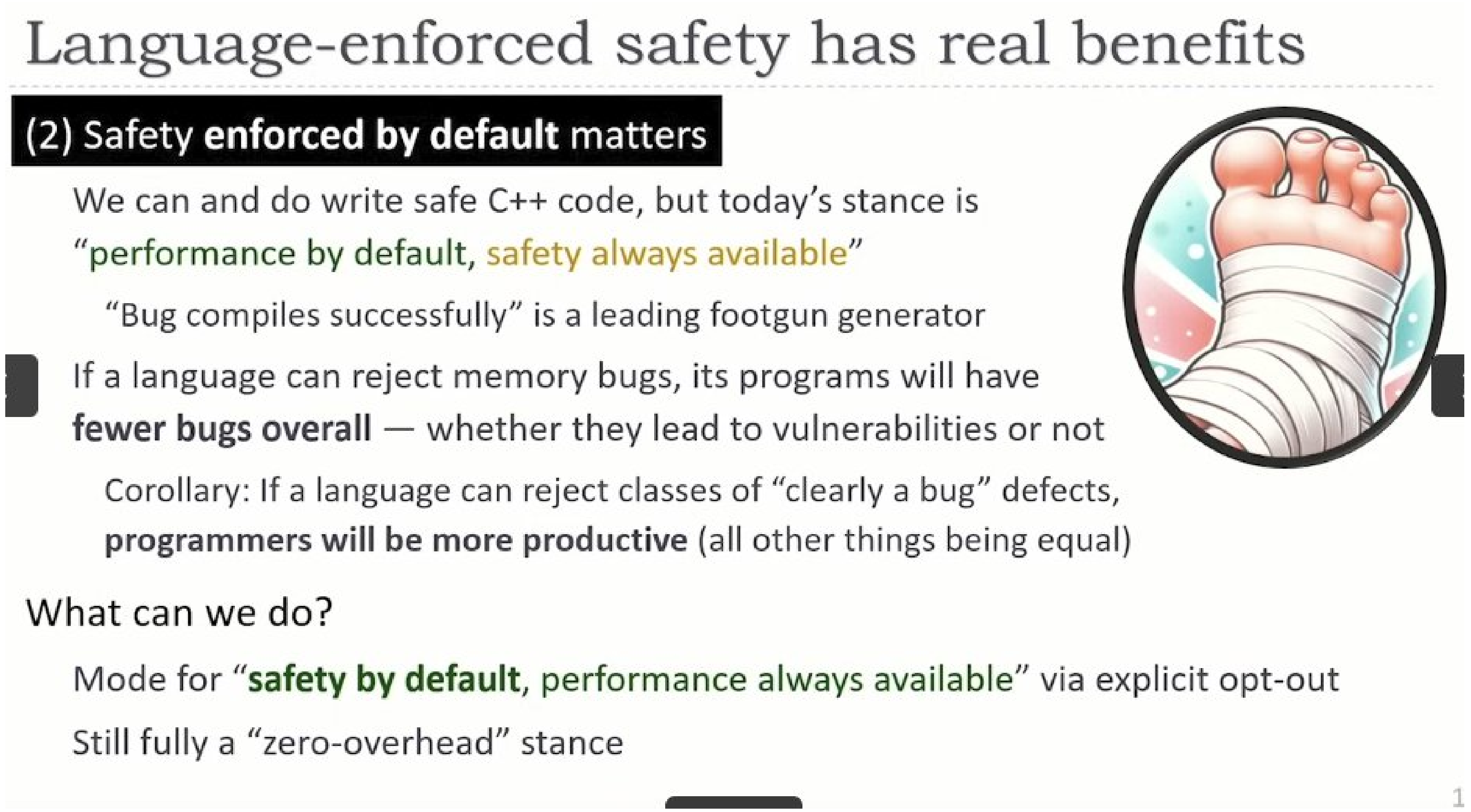
Type system allows it build safe & secure code with error detected at compiled time

Smart pointer are important to catch memory error

Modern c++ improves a lot the syntax (auto, packing) so it looks nearly like python

Conclusion...

Modern c++ vs RUST



Language-enforced safety has real benefits

(2) Safety enforced by default matters

- We can and do write safe C++ code, but today's stance is "performance by default, safety always available"
  - "Bug compiles successfully" is a leading footgun generator
- If a language can reject memory bugs, its programs will have **fewer bugs overall** — whether they lead to vulnerabilities or not
  - Corollary: If a language can reject classes of "clearly a bug" defects, **programmers will be more productive** (all other things being equal)

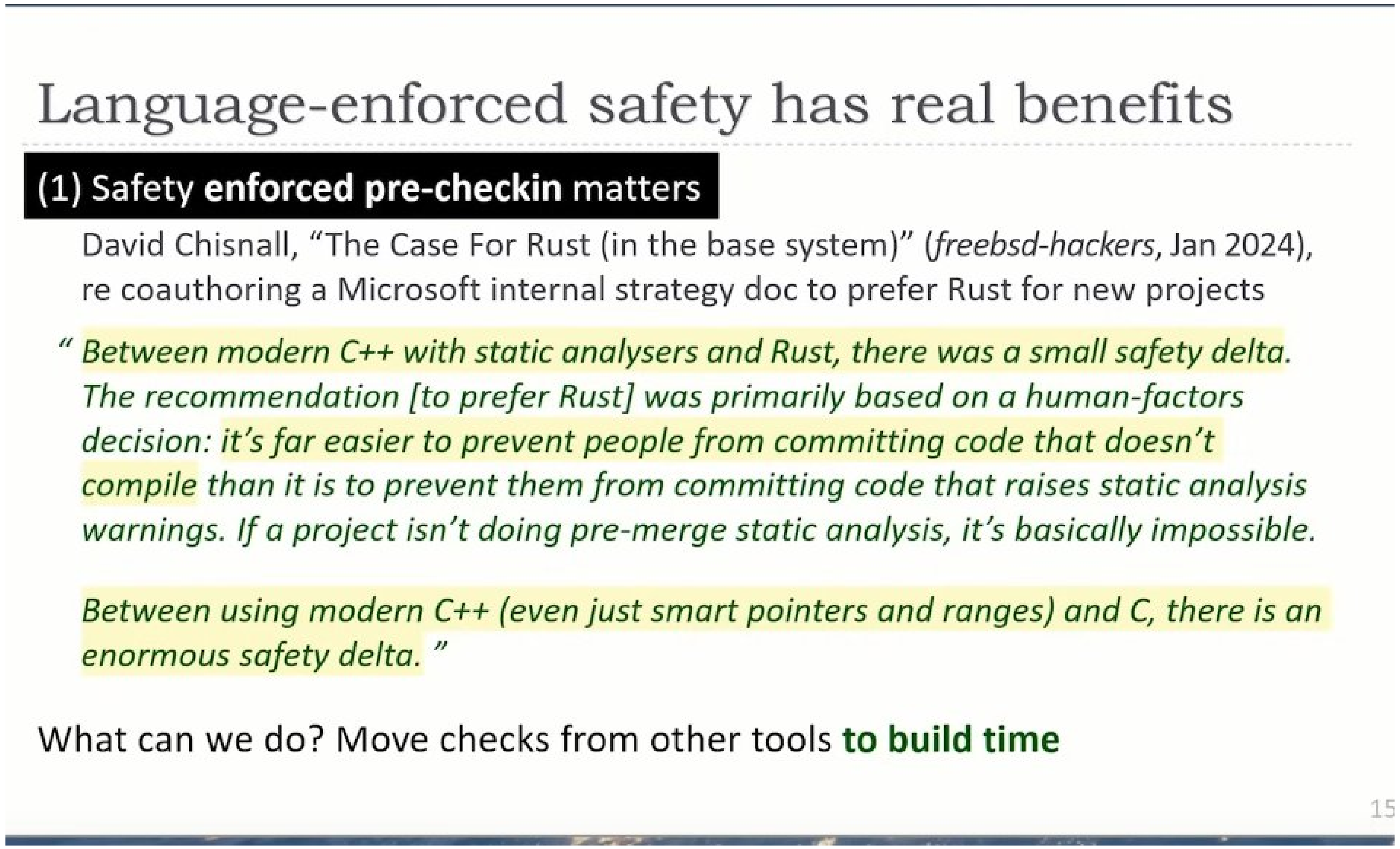
What can we do?

- Mode for "**safety by default**, performance always available" via explicit opt-out
- Still fully a "zero-overhead" stance

1

Conclusion...

Modern c++ vs RUST



Language-enforced safety has real benefits

(1) Safety enforced pre-checkin matters

David Chisnall, "The Case For Rust (in the base system)" (*freebsd-hackers*, Jan 2024), re coauthoring a Microsoft internal strategy doc to prefer Rust for new projects

" *Between modern C++ with static analysers and Rust, there was a small safety delta. The recommendation [to prefer Rust] was primarily based on a human-factors decision: it's far easier to prevent people from committing code that doesn't compile than it is to prevent them from committing code that raises static analysis warnings. If a project isn't doing pre-merge static analysis, it's basically impossible.*

Between using modern C++ (even just smart pointers and ranges) and C, there is an enormous safety delta. "

What can we do? Move checks from other tools **to build time**

15

My preferred « best practice »

DRY programmer's

WET programmer's

My preferred « best practice »

My preferred « best practice »

DRY programmer's

WET programmer's

Thank you.

WET programmer’s