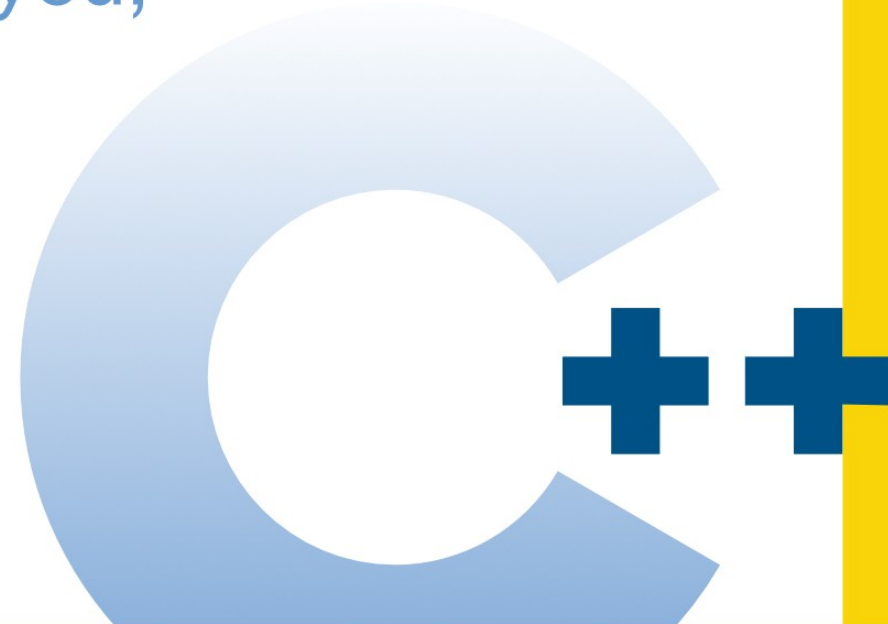
BeC++ SYMPOSIUM

Type Punning: the joke is on you, pun intended



Type punning, the joke is on you, pun intended.

Lieven de Cock
www.codeblocks.org
lieven.de.cock@telenet.be

Type Punning

- Wikipedia: In computer science, type punning is a common term for any programming technique that **subverts** or **circumvents** the **type system** of a programming language in order to achieve an effect that would be difficult or impossible to achieve within the bounds of the formal language.
- ===> an interpretation: Circumventing the type system to access the same memory location through different types.

Why this talk ?

- It all started when reading this article from Andreas Fertig back in early March 2025, together with a static checker suggesting to reduce on the use of reinterpret_cast.
- https://andreasfertig.com/blog/2025/03/the-correct-way-to-do-type-punning-in-cpp/ :
  - What ?
  - I was not aware of that (since I hardly or never use reinterpret_cast, but after quick look, this was present in our codebase)
  - Most colleagues/friends were also not aware of that
  - Even some C++ experts learned about this late in their career
  - Even Jason Turner is having discussions to have people realize the problems of this
- Tools are becoming available

Why this talk ?

The correct way to do type punning in C++

Mar 04, 2025

Today's post is relevant for everybody who does type punning in C++. Something I did for years while working in the embedded software domain. Something others have done for a long time before I entered the game. Something that is 100% illegal according to the standard. And yet I know that many embedded devices are built using type punning despite it being illegal and with that UB.

Let's start at the beginning

First, by using the code below, let's establish what I mean by type punning and why it is undefined behavior in C++.

```
1  float pi = 3.14f;  (A) The base value and type
2
3  (B) Does not do what we want
4  uint32_t first = static_cast<uint32_t>(pi);
5
6  (C) Would do what you want, but is UB
7  uint32_t second = *reinterpret_cast<uint32_t*>(&pi);
```

Reinterpret_cast: Trust me ...

- https://www.youtube.com/watch?v=V9qTsDg-RPo&list=RDV9qTsDg-RPo&start_radio=1

Trust me,
I know
what I'm doing
SLEDGE
a HAMMER! SUPERCUT

Memoirs

- We are a published author, writing the memoirs of 2 variables called ‘foo’ and ‘bar’

```
4   void myFunction()
5   {
6       /// code here
7
8       int foo;
9
10      /// code here
11
12      int bar{};
13
14      /// code here
15  }
```

Memoirs

- Foo
  - Was born in 20**08**
  - Had a most glamorous life
  - Died in 20**15**
- Bar
  - Was born in 20**12**
  - Had a regular life
  - Died in 20**15**

Lifetime

- So our variables/objects had a lifetime
  - They came into existence (started their lifetime)
  - They died (stopped their lifetime)

- No object, can have a life without these 2 steps
- During their lifetime an object occupies some memory
- At a given memory location only 1 object can live

Naive Random Allocator

- Let's create a silly allocator, which puts an object at a random location (ensuring nothing lives there already)
- int i1, i2
- bool b1, b2
- char c1, c2
- std::array<char, 5> ca
- float f1, f2

i1	i1	i1	i1		i2
i2	i2	i2		c2	
f1	f1	f1	f1		
	b1		c1		
				b2	ca
ca	ca	ca	ca		
		f2	f2	f2	f2

Alignment

```
13  template <typename T>
14  void infoOn(const std::string& typeName)
15  {
16      std::println("{}: size: {} , alignment: {}", typeName, sizeof(T), std::alignment_of_v<T>);
17  }
```

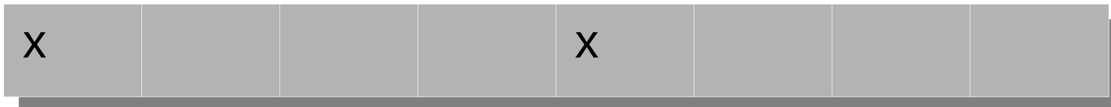
Alignment

```
47        infoOn<double>("double");
48        infoOn<float>("float");
49        infoOn<int>("int");
50        infoOn<uint64_t>("uint64_t");
51        infoOn<int64_t>("int64_t");
52        infoOn<uint32_t>("uint32_t");
53        infoOn<int32_t>("int64_t");
54        infoOn<uint16_t>("uint16_t");
55        infoOn<int16_t>("int16_t");
56        infoOn<uint8_t>("uint8_t");
57        infoOn<int8_t>("int8_t");
58        infoOn<bool>("bool");
59        infoOn<char>("char");
60        infoOn<std::byte>("byte");
61        infoOn<std::array<char, 1>>("std::array<char, 1>");
62        infoOn<std::array<char, 4>>("std::array<char, 4>");
63        infoOn<std::array<char, 5>>("std::array<char, 5>");
64        infoOn<std::array<char, 100>>("std::array<char, 100>");
```

```
double: size: 8 , alignment: 8
float: size: 4 , alignment: 4
int: size: 4 , alignment: 4
uint64_t: size: 8 , alignment: 8
int64_t: size: 8 , alignment: 8
uint32_t: size: 4 , alignment: 4
int64_t: size: 4 , alignment: 4
uint16_t: size: 2 , alignment: 2
int16_t: size: 2 , alignment: 2
uint8_t: size: 1 , alignment: 1
int8_t: size: 1 , alignment: 1
bool: size: 1 , alignment: 1
char: size: 1 , alignment: 1
byte: size: 1 , alignment: 1
std::array<char, 1>: size: 1 , alignment: 1
std::array<char, 4>: size: 4 , alignment: 1
std::array<char, 5>: size: 5 , alignment: 1
std::array<char, 100>: size: 100 , alignment: 1
```

Alignment

- So an object of a certain type can not just live anywhere
  - char → any byte in the memory is ok
  - int → needs to live at an address which is a multiple of 4
- So if we would have an array of chars, they can live anywhere (alignment of array → alignment of an element)
  - But if we would then look at this array and say, those 4 bytes in there, well let's consider those to be an int (reinterpret_cast)
    - ¾ are wrong => **UB**



UB due to incorrect alignment

- We already have UB on line 12, even before we would try to use the pointer
- UBSAN spots this, but somehow not in the case our array would be of size 4, both cases are UB

```
 8      std::array<char , 5> buf{'b', 'c', 'a', 'd', 'e'};
 9
10      {
11          /// here we have unaligned memory access ==> UB
12          auto* asUi16 = reinterpret_cast<uint16_t*>(buf.data());
13
14          std:: cout << *asUi16 << '\n';
15      }
```

UB due to incorrect alignment

```
/home/ldco/Projects/Teaching/TypePunning/Reinterpret2/src/main.cpp:14:23: runtime error: load of misaligned address 0x7f
fe0b93a103 for type 'short unsigned int', which requires 2 byte alignment
0x7ffe0b93a103: note: pointer points here
 00  6e d9 43 62 63 61 64 65  03 a1 93 0b fe 7f 00 00  10 a2 93 0b fe 7f 00 00  00 00 00 00 00 00 00
              ^
```

- We will come back to this example, presenting a solution

Current problems of reinterpret_cast

- alignment

Alignment OK

- In the next examples, the alignment will be ok, we will be working with a **float** and an **int**.
- We have an int, and want to reinterpret the bits of it as a float, or the other way around
- Same size, same alignment

- => that part of UB is already avoided … but ...

Strict Aliasing

- Aliasing: more than 1 name (pointer / reference) refers to the same address
  - Remember only 1 object can life at a give location
- **Strict** Aliasing: only pointers of the **same type** can point to the same object
  - Same type: and some minor variations (cv, (un)signed, ...)

Strict Aliasing

- Line 8 : store 242 at location pointed to by i
- Line 9: store 0.0 at location pointed to by f
- Line 10: read the value at i ==> 242
  - And return it → so we return 242

```
6   int foo(float* f, int* i)
7   {
8       *i = 242;
9       *f = 0.0f;
10      return *i;
11  }
```

Strict Aliasing: Please optimize

- Line 8 : store 242 at location pointed to by i
- Strict aliasing
  - Float != int
- So the value at i can not change after line 8
- So no need to read it out again, we know we can just return 242

```
6   int foo(float* f, int* i)
7   {
8       *i = 242;
9       *f = 0.0f;
10      return *i;
11  }
```

Strict Aliasing: But …… bad ...

```
15  int main()
16  {
17      int x = 0;
18      std::println("{}", foo(reinterpret_cast<float*>(&x), &x));
19  }
```

```
6   int foo(float* f, int* i)
7   {
8       *i = 242;
9       *f = 0.0f;
10      return *i;
11  }
```

- What will be printed out, we violated strict aliasing ?
  - 0
  - 242

	Debug	Release
gcc	0	242
clang	0	242

Strict Aliasing

- Violating Strict Aliasing is UB
- Compiler assumes UB can not occur
- Compiler uses this to optimize

Current problems of reinterpret_cast

- Alignment
- Strict Aliasing

Lifetime

- There are more issues with reinterpret_cast

reinterpret_cast: UB

- Line 10 is still ok
- UB is at line 11: *f

```
7   float notOk()
8   {
9       int i{242};
10      float* f = reinterpret_cast<float*>(&i);
11      return *f;
12  }
```

- Remember, when using an object, it needs to have started it's lifetime
- Nowhere a float started it's lifetime, there is no float object alive in this code
- But the program seems to work correctly …
- UBSAN does not complain
- With UB anything can happen (even acceptable results …)
- Trying to verify with consteval, does not work since reinterpret_cast is not constexpr
- Strict aliasing also does not agree ...

UB detector : constexpr

- C++26
  - static_cast<void*> constexpr

```
1
2   consteval void notOk()
3   {
4       int* i = new int(242);
5       float* f = static_cast<float *>(static_cast<void *>(i));
6
7   //      *f = 3.0;
8       delete i;
9   }
10
11  int main()
12  {
13      notOk();
14      return 0;
15  }
```

- The static_casts are still ok, no UB yet
- But the moment we dereference and think we can interact with a float object (which does not exist) we hit UB

UB detector : constexpr

C++ source #1

```cpp

consteval void notOk()
{
    int* i = new int(242);
    float* f = static_cast<float *>(static_cast<void *>(i));

//    *f = 3.0;
    delete i;
}

int main()
{
    notOk();
    return 0;
}
```

x86-64 gcc (trunk) -std=c++26 -O2

```
"main":
        xor     eax, eax
        ret
```

Output (0/0) x86-64 gcc (trunk) - 759ms (26638) ~155 lines filtered

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
```

UB detector : constexpr

COMPILER EXPLORER Add... More Templates Get cool branded gear in the Compiler Explorer shop Sponsors intel. Solid Sands JETBRAINS Share Policies

C++ source #1

C++

```cpp
consteval void notOk()
{
    int* i = new int(242);
    float* f = static_cast<float *>(static_cast<void *>(i));

    *f = 3.0;
    delete i;
}

int main()
{
    notOk();
    return 0;
}
```

x86-64 gcc (trunk) (Editor #1)

x86-64 gcc (trunk) -std=c++26 -O2

Output... Filter... Libraries Overrides Add new... Add tool...

```
<Compilation failed>

# For more information see the output window
```

Output (0/11) x86-64 gcc (trunk) - 254ms Compiler License

Output of x86-64 gcc (trunk) (Compiler #1)

Wrap lines Select all

```
<source>: In function 'int main()':
<source>:13:10: error: call to consteval function 'notOk()' is not a constant expression
   13 |     notOk();
      |     ~~~~~^~
<source>:13:10:   in 'constexpr' expansion of 'notOk()'
<source>:7:5: error: accessing value of 'int [1]' object through a 'float' glvalue in a constant expression
    7 |     *f = 3.0;
      |     ^~
<source>:4:25: note: 'int [1]' object created here
    4 |     int* i = new int(242);
      |                         ^
Compiler returned: 1
```

Wait a minute, I am smart

- The union trick
- C ↔ C++
- UB: access to non active member of union

```
11  union U
12  {
13      float f;
14      int i;
15  };
16
17  /// constexpr so when in compiler context we can prove it is UB
18  constexpr int BadUBwayFloat2Int(float f)
19  {
20      U u{f};
21      return u.i;    /// UB since i is NOT the active member
22  }
23
```

- **constexpr float fl = BadUBwayFloat2Int(1.5f);**
- Again the outcome seems correct (in case not constexpr, so it compiles and runs)
- Lets have a look at the compiler complaints
  - Gcc14
  - Clang21

Compiler says no

```
-------------- Build: GccDebug in TypePunning1 (compiler: GNU GCC Compiler)---------------
[ 50.0%] g++ -Wshadow -Winit-self -Wredundant-decls -Wcast-align -Wunreachable-code -Wmissing-declarations -Wmissing-include-dirs -Wswitch-default -Wmain -pedantic -Wextra -Wall -std=c++23 -fexceptions -g -Wno-unuse
local-typedefs -Wno-deprecated-declarations  -c /home/ldco/Projects/Teaching/TypePunning/TypePunning1/src/main.cpp -o ../Deliv/GccDebug/src/main.o
[100.0%] g++  -o ../Deliv/GccDebug/TypePunning1 ../Deliv/GccDebug/src/main.o
/home/ldco/Projects/Teaching/TypePunning/TypePunning1/src/main.cpp: In function ‘int main()’:
/home/ldco/Projects/Teaching/TypePunning/TypePunning1/src/main.cpp:45:43:   in ‘constexpr’ expansion of ‘{anonymous}::BadUBwayFloat2Int(1.5e+0f)’
/home/ldco/Projects/Teaching/TypePunning/TypePunning1/src/main.cpp:21:14: error: accessing ‘{anonymous}::U::i’ member instead of initialized ‘{anonymous}::U::f’ member in constant expression
   21 |     return u.i;   /// UB since i is NOT the active member
      |              ~~^
```

```
-------------- Build: ClangDebug in TypePunning1 (compiler: LLVM Clang Compiler)---------------
[ 50.0%] clang++ -Wshadow -Winit-self -Wredundant-decls -Wcast-align -Wunreachable-code -Wmissing-declarations -Wmissing-include-dirs -Wswitch-default -Wmain -pedantic -Wextra -Wall -std=c++23 -fexceptions -g -
stdlib=libc++ -Wno-unknown-warning-option -fexperimental-library  -c /home/ldco/Projects/Teaching/TypePunning/TypePunning1/src/main.cpp -o ../Deliv/ClangDebug/src/main.o
[100.0%] clang++  -o ../Deliv/ClangDebug/TypePunning1 ../Deliv/ClangDebug/src/main.o   -lc++ -lc++abi
/home/ldco/Projects/Teaching/TypePunning/TypePunning1/src/main.cpp:45:21: error: constexpr variable 'fl' must be initialized by a constant expression
   45 |     constexpr float fl = BadUBwayFloat2Int(1.5f);  /// this will not compile :-)
      |                     ^    ~~~~~~~~~~~~~~~~~~~~~~~~
/home/ldco/Projects/Teaching/TypePunning/TypePunning1/src/main.cpp:21:12: note: read of member 'i' of union with active member 'f' is not allowed in a constant expression
   21 |     return u.i;   /// UB since i is NOT the active member
      |            ^
```

```
Result : 1069547520
```

Now what ?

- So a lot of code in production is actually UB
- So how should it be done ?
  - Pre c++20
    - memcpy
  - C++20 and later
    - bit_cast

Pre c++20: memcpy

```
14  float okPre20()
15  {
16      int i{242};
17      float f{};
18      std::memcpy(&f, &i , sizeof(i));
19      return f;
20  }
```

```
24  /// memcpy is not constexpr so we can not try to prove it in a similar way
25  constexpr int GoodOldWayFloat2Int(float f)
26  {
27      int i{};
28      std::memcpy(&i, &f, sizeof(f));
29      return i;
30  }
```

Oh no … memcpy …

C++ source #1

A ▾ Save/Load + Add new... ▾ V Vim CppInsights Quick-bench C++

```cpp
#include <cstring>

float notOk()
{
    int i{242};
    float* f = reinterpret_cast<float*>(&i);
    return *f;
}

float okPre20()
{
    int i{242};
    float f{};
    std::memcpy(&f, &i , sizeof(i));
    return f;
}
```

x86-64 gcc (trunk) (Editor #1)

x86-64 gcc (trunk) -std=c++23 -g -O1

A ▾ Output... ▾ Filter... ▾ Libraries Overrides +

```asm
"notOk()":
        movss   xmm0, DWORD PTR .LC0[rip]
        ret
"okPre20()":
        movss   xmm0, DWORD PTR .LC0[rip]
        ret
.LC0:
        .long   242
```

- **Compilers** know about this, and **optimize**, they are **smart**

Oh no … memcpy …

C++ source #1

A Save/Load + Add new... V Vim CppInsights Quick-bench C++

```cpp
#include <cstring>

float notOk()
{
    int i{242};
    float* f = reinterpret_cast<float*>(&i);
    return *f;
}

float okPre20()
{
    int i{242};
    float f{};
    std::memcpy(&f, &i , sizeof(i));
    return f;
}
```

x86-64 clang (trunk) (Editor #1)

x86-64 clang (trunk) -std=c++23 -g -O1

A Output... Filter... Libraries Overrides

```asm
.LCPI0_0:
        .long   0x000000f2
notOk():
        movss   xmm0, dword ptr [rip + .LCPI0_0]
        ret

.LCPI1_0:
        .long   0x000000f2
okPre20():
        movss   xmm0, dword ptr [rip + .LCPI1_0]
        ret
```

- **Compilers** know about this, and **optimize**, they are **smart**

c++20: bit_cast

```
22  float ok()
23  {
24      int i{242};
25      return std::bit_cast<float>(i);
26  }
```

```
34  constexpr int ModernWayFloat2Int(float f)
35  {
36      return std::bit_cast<int>(f);    /// bit_cast starts the lifetime of a new object, the integer
37  }
```

- Both types have the same size
- Both types are trivially copyable

Let's have a look

C++ source #1

A Save/Load + Add new... V Vim CppInsights Quick-bench C++

```cpp
#include <cstring>
#include <bit>

float notOk()
{
    int i{242};
    float* f = reinterpret_cast<float*>(&i);
    return *f;
}

float okPre20()
{
    int i{242};
    float f{};
    std::memcpy(&f, &i , sizeof(i));
    return f;
}

float ok()
{
    int i{242};
    return std::bit_cast<float>(i);
}
```

x86-64 gcc (trunk) (Editor #1)

x86-64 gcc (trunk) -std=c++23 -g -O1

A Output... Filter... Libraries Overrides

```asm
"notOk()":
        movss   xmm0, DWORD PTR .LC0[rip]
        ret
"okPre20()":
        movss   xmm0, DWORD PTR .LC0[rip]
        ret
"ok()":
        movss   xmm0, DWORD PTR .LC0[rip]
        ret
.LC0:
        .long   242
```

- **Compilers** know about this, and **optimize**, they are **smart**

Let's have a look

C++ source #1

A ▾ Save/Load + Add new... ▾ V Vim CppInsights Quick-bench C++

```cpp
#include <cstring>
#include <bit>

float notOk()
{
    int i{242};
    float* f = reinterpret_cast<float*>(&i);
    return *f;
}

float okPre20()
{
    int i{242};
    float f{};
    std::memcpy(&f, &i , sizeof(i));
    return f;
}

float ok()
{
    int i{242};
    return std::bit_cast<float>(i);
}
```

x86-64 clang (trunk) (Editor #1)

x86-64 clang (trunk) -std=c++23 -g -O1

A ▾ Output... ▾ Filter... ▾ Libraries Overrides + Ad

```
.LCPI0_0:
        .long   0x000000f2
notOk():
        movss   xmm0, dword ptr [rip + .LCPI0_0]
        ret

.LCPI1_0:
        .long   0x000000f2
okPre20():
        movss   xmm0, dword ptr [rip + .LCPI1_0]
        ret

.LCPI2_0:
        .long   0x000000f2
ok():
        movss   xmm0, dword ptr [rip + .LCPI2_0]
        ret
```

- **Compilers** know about this, and **optimize**, they are **smart**

Size

- Size matters
  - Size of int
  - Size of my struct/class
  - …
- What is the size of ‘my struct’ ?

Sizes and padding

- What are the sizes and alignment of the following structs ?

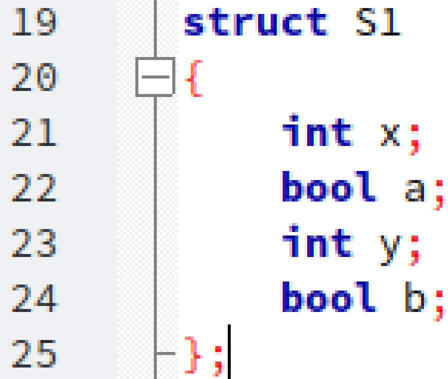
```
19  struct S1
20  {
21      int x;
22      bool a;
23      int y;
24      bool b;
25  };
26
27  struct S2
28  {
29      int x;
30      bool a;
31      bool b;
32      int y;
33  };
34
35  struct S3
36  {
37      int x;
38      int y;
39      bool a;
40      bool b;
41  };
```

```
S1: size: 16 , alignment: 4
S2: size: 12 , alignment: 4
S3: size: 12 , alignment: 4
std::array<S3, 100>: size: 1200 , alignment: 4
```

- Alignment struct = = max(alignment members)

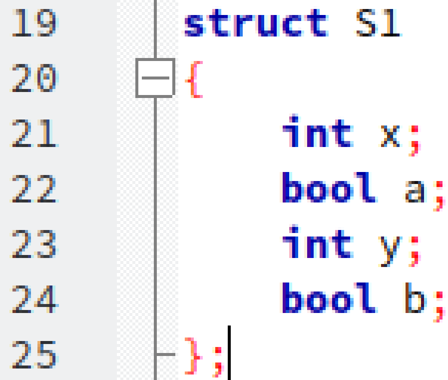
- 10 != 16
- 10 != 12
- 1000 != 1200

Sizes and padding



- **4** bytes for x
- **1** byte for a
- … y has alignment of 4, so it can only start 3 bytes after a => gap of 3, aka padding of **3** bytes
- **4** bytes for y
- **1** byte for b
- ==> that would total to 13, but ...

Sizes and padding


```
19  struct S1
20  {
21      int x;
22      bool a;
23      int y;
24      bool b;
25  };
```

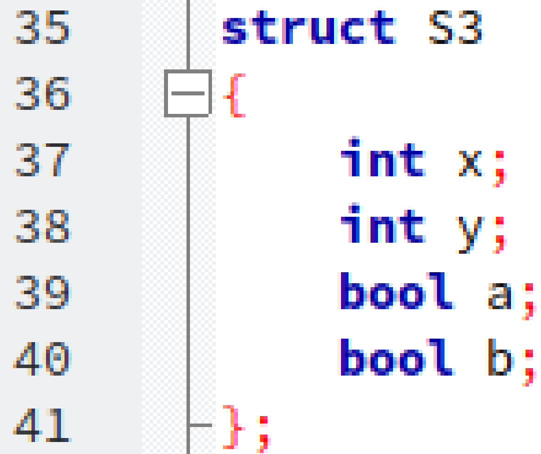
- Next to internal padding there is also padding at the end
- Why ?
  - Arrays : S1 foo[2];
  - The second element can only start at an alignment of 4, meaning 3 bytes after the ‘core 13’
  - This padding at the end is part of the size of the type => size of **16**

Sizes and padding

```
27  struct S2
28  {
29          int x;
30          bool a;
31          bool b;
32          int y;
33  };
```

- **4** bytes for x
- **1** byte for a
- **1** byte for b
- … y needs has alignment of 4, so it can only start 2 bytes after b => gap of 2, aka padding of **2** bytes
- **4** bytes for y
- ==> total **12**

Sizes and padding



- **4** bytes for x
- **4** bytes for y
- **1** byte for a
- **1** byte for b
- **2** bytes padding at the end
- ==> total **12**

What are the values of the padding bytes ?

- Unknown
- You should not ask that question !
- Compare with care
  - <=> is your friend

Is there padding in my struct ?

- std::has_unique_object_representations_v<T>
  - If there is padding it will say true
  - But can say true even when no padding
    - It's purpose: This trait was introduced to make it possible to determine **whether a type can be correctly hashed by hashing its object representation as a byte array**.
    - eg.: float (multiple representations of NaN)

```
48     static_assert( not std::has_unique_object_representations_v<S1>);
49     static_assert( not std::has_unique_object_representations_v<S2>);
50     static_assert( not std::has_unique_object_representations_v<S3>);
51     static_assert( std::has_unique_object_representations_v<S4>);
```

```
9   struct S1
10  {
11      int x;
12      bool a;
13      int y;
14      bool b;
15  };
16
17  struct S2
18  {
19      int x;
20      bool a;
21      bool b;
22      int y;
23  };
24
25  struct S3
26  {
27      int x;
28      int y;
29      bool a;
30      bool b;
31  };
32
33  struct S4
34  {
35      int x;
36      bool a;
37      bool b;
38      bool c;
39      bool d;
40      int y;
41  };
```

Sizes and padding: reflection

```
24
25  int main()
26  {
27      std::array<int, sizeof(S1)> arr{};
28      arr.fill(-1);
29
30      constexpr auto ctx = std::meta::access_context::current();
31      size_t memberIdx{};
32      template for (constexpr auto m :
33          std::define_static_array(nonstatic_data_members_of(^^S1, ctx)))
34      {
35          std::println("{} at offset {} and has a size {}", identifier_of(m), offset_of(m).bytes, size_of(m)); /// there is also alignment_of we can use
36
37          auto start = offset_of(m).bytes;
38          auto stop = start + size_of(m);
39          do
40          {
41              arr[start] = memberIdx;
42          } while (++start < stop);
43          ++ memberIdx;
44      }
45      std::println("see the members and their padding, padding byte has the value -1: {}", arr);
46  }
```

Sizes and padding: reflection

```
19  struct S1
20  {
21      int x;
22      bool a;
23      int y;
24      bool b;
25  };
```

- https://godbolt.org/z/9nosan7fs
- Padding byte is a -1
- Index of member

```
x at offset 0 and has a size 4
a at offset 4 and has a size 1
y at offset 8 and has a size 4
b at offset 12 and has a size 1
see the members and their padding, padding byte has the value -1: [0, 0, 0, 0, 1, -1, -1, -1, 2, 2, 2, 2, 3, -1, -1, -1]
```

Sizes and padding: reflection

- Let’s turn this into a padding detector : first step, only suited for our S1-S4 examples (not yet ready for inheritance, public/private/protected, …) -- will come in the future ….

```
54  template <typename T>
55  constexpr bool hasPadding()
56  {
57      std::array<int, sizeof(T)> arr{};
58      arr.fill(-1);
59
60      constexpr auto ctx = std::meta::access_context::current();
61      size_t memberIdx{};
62      template for (constexpr auto m :
63          std::define_static_array(nonstatic_data_members_of(^^T, ctx)))
64      {
65          auto start = offset_of(m).bytes;
66          auto stop = start + size_of(m);
67          do
68          {
69              arr[start] = memberIdx;
70          } while (++start < stop);
71          ++ memberIdx;
72      }
73      return std::ranges::find(arr, -1) != arr.end();
74  }
75
76  template <typename T>
77  struct has_padding
78  {
79      static constexpr bool value = hasPadding<T>();
80  };
```

```
88          static_assert( hasPadding<S1>());
89          static_assert( hasPadding<S2>());
90          static_assert( hasPadding<S3>());
91          static_assert( not hasPadding<S4>());
92          static_assert( not hasPadding<S5>());
93
94          static_assert( has_padding<S1>::value);
95          static_assert( has_padding<S2>::value);
96          static_assert( has_padding<S3>::value);
97          static_assert( not has_padding<S4>::value);
98          static_assert( not has_padding<S5>::value);
```

Question

- If I want to serialize my struct, do I include the padding bytes ?
  - Yes:
    - Simple
    - Waste of bandwidth
  - No
    - now what ?

Recap

- Alignment
- Size
- Padding
- Lifetime
- reinterpret_cast quickly results in UB
  - Solutions:
    - std::memcpy
    - std::bit_cast

Questions

- We saw the fix for “reinterpreting” a scalar, built-in type
- What with an array of things ?
- What with an array of bytes (coming from a file, or the network), where we know, that those bytes are:
  - A certain struct (eg our S1)
  - An array of things:
    - Array of ints
    - Array of S1s
  - std::bit_cast can not help
  - std::memcpy, this will probably be actual copies this time, the power of compilers have limits
- Aka deserializing
- We will ignore endianness for simplicity

Working example – UB … works …

```
4   namespace
5   {
6
7   struct Coordinate
8   {
9       int x{};
10      int y{};
11  };
12
13  static_assert( std::has_unique_object_representations_v<Coordinate>);
14  static_assert( std::is_implicit_lifetime_v<Coordinate>); /// clang has it, coming in gcc16
15
16  } // namespace
17
18  int main()
19  {
20      /// let's hand craft an array of bytes, assuming it could have come from the network
21      alignas(Coordinate) char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0, 0};
22      auto* cos = reinterpret_cast<Coordinate*>(buffer);
23
24      std::println("{},{} {},{}", cos[0].x, cos[0].y, cos[1].x, cos[1].y);
25  }
```

```
10,10 20,20
```

Working example – memcpy

```
5    namespace
6    {
7
8    struct Coordinate
9    {
10       int x{};
11       int y{};
12   };
13
14   static_assert( std::has_unique_object_representations_v<Coordinate>);
15   static_assert( std::is_implicit_lifetime_v<Coordinate>); /// clang has it, coming in gcc16
16
17   } // namespace
18
19   int main()
20   {
21       /// let's hand craft an array of bytes, assuming it could have come from the network
22       char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0, 0};
23       Coordinate cos[2]{};
24       std::memcpy(cos, buffer, sizeof(cos));
25
26       std::println("{},{} {},{}", cos[0].x, cos[0].y, cos[1].x, cos[1].y);
27   }
```

```
10,10 20,20
```

No copy please

- Avoid the copy
  - Compilers again may notice what we are doing and no copy happens
- Efficient solution please
- std::start_lifetime_as, std::start_lifetime_as_array
  - Still not available (end of 2025) in neither of the 3 major compilers (gcc, clang, MSVC)
  - But … is in gcc trunk (November 2025) → should be in gcc16
  - Clang trunk (March 2026) → should be in clang 23
- But first the std::memmove trick

No copy please: memmove trick

- Since c++20 some methods are blessed and start lifetime (malloc, memcpy, **memmove**, …)
- Let’s memmove to ourselves
  - Use return value
    - So no std::launder needed

```
20    /// let's hand craft an array of bytes, assuming it could have come from the network
21    alignas(Coordinate) char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0};
22    auto* ptr = static_cast<Coordinate*>(std::memmove(buffer, buffer, sizeof(Coordinate)));
23
24    std::println("{},{}", ptr->x, ptr->y);
```

C++23

- std::**start_lifetime_as**
- std::**start_lifetime_as_array**

start_lifetime_as

- Will start the lifetime of something (eg. array of bytes), as something else (eg a Coordinate): std::start_lifetime_as<Coordinate>(buffer)
- Destructive, the old thing is no longer there, it is now the new thing
  - Remember strict aliasing
- Compilers again optimize, so no copying occurs
- Still requires that alignment requirements are fulfilled!

No copy please

```
18  int main()
19  {
20      {
21          /// let's hand craft an array of bytes, assuming it could have come from the network
22          char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0, 0};
23          auto* cos = std::start_lifetime_as_array<Coordinate>(buffer, 2);
24
25          std::println("{},{} {},{}", cos[0].x, cos[0].y, cos[1].x, cos[1].y);
26      }
27      {
28          /// let's hand craft an array of bytes, assuming it could have come from the network
29          char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0, 0};
30          auto* cos0 = std::start_lifetime_as<Coordinate>(buffer);
31          auto cos1 = *std::start_lifetime_as<Coordinate>(buffer + 8);
32
33          std::println("{},{} {},{}", cos0->x, cos0->y, cos1.x, cos1.y);
34      }
35  }
```

No copy please

COMPILER EXPLORER Add... More Templates

C++ source #1

```cpp
#include <print>
#include <type_traits>
#include <memory>

namespace
{

struct Coordinate
{
    int x{};
    int y{};
};

static_assert( std::has_unique_object_representations_v<Coordinate>);

} // namespace

int main()
{
    {
        /// let's hand craft an array of bytes, assuming it could have come from the network
        char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0, 0};
        auto* cos = std::start_lifetime_as_array<Coordinate>(buffer, 2);

        std::println("{},{} {},{}", cos[0].x, cos[0].y, cos[1].x, cos[1].y);
    }
    {
        /// let's hand craft an array of bytes, assuming it could have come from the network
        char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0, 0};
        auto* cos0 = std::start_lifetime_as<Coordinate>(buffer);
        auto cos1 = *std::start_lifetime_as<Coordinate>(buffer + 8);

        std::println("{},{} {},{}", cos0->x, cos0->y, cos1.x, cos1.y);
    }
}
```

x86-64 gcc (trunk) -std=c++23

Output of x86-64 gcc (trunk) (Compiler #1)

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
  10,10 20,20
  10,10 20,20
```

Previous 2 slides ...

- What was wrong ?

No copy please

```cpp
#include <print>
#include <type_traits>
#include <memory>

namespace
{

struct Coordinate
{
    int x{};
    int y{};
};

static_assert( std::has_unique_object_representations_v<Coordinate>);

} // namespace

int main()
{
    {
        /// let's hand craft an array of bytes, assuming it could have come from the net
        alignas(Coordinate) char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20,
        auto* cos = std::start_lifetime_as_array<Coordinate>(buffer, 2);

        std::println("{},{} {},{}", cos[0].x, cos[0].y, cos[1].x, cos[1].y);
    }
    {
        /// let's hand craft an array of bytes, assuming it could have come from the net
        alignas(Coordinate) char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20,
        auto* cos0 = std::start_lifetime_as<Coordinate>(buffer);
        auto cos1 = *std::start_lifetime_as<Coordinate>(buffer + 8);

        std::println("{},{} {},{}", cos0->x, cos0->y, cos1.x, cos1.y);
    }
}
```

x86-64 gcc (trunk) -std=c++23

```
5385    .long   2147418111
```

```
ASM generation compiler returned: 0
Execution build compiler returned: 0
Program returned: 0
  10,10 20,20
  10,10 20,20
```

Let's check if alignment ok

```
16  int processCoordinates(std::span<const Coordinate> cos)
17  {
18      if(cos.empty())
19      {
20          return 0;
21      }
22      /// let's calculate the average x-coordinate explicitly
23      int res{};
24      for(const auto&co : cos)
25      {
26          res += co.x;
27      }
28      return res / cos.size();
29  }
30
31  template <typename To>
32  auto asArray(std::span<const char> from)
33  {
34      assert(reinterpret_cast<uintptr_t>(from.data()) % alignof(To) == 0);
35      assert(from.size() % sizeof(To) == 0);
36
37      const auto elements = from.size() / sizeof(To);
38      return std::span<const To>{
39          std::start_lifetime_as_array<To>(from.data(), elements), elements};
40  }
41
42  int processData(const char* theBytes, std::size_t size)
43  {
44      return processCoordinates(asArray<Coordinate>(std::span<const char>(theBytes, size)));
45  }
```

- https://godbolt.org/z/3oz3ExPGG

Let’s check if alignment ok

```
50     /// so let's assume we got the bytes from the network (or some file ...)
51     {
52         alignas(Coordinate) char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0, 0};
53
54         const auto result = processData(buffer, sizeof(buffer));
55         std::println("{}", result);
56     }
57
58     {   /// seems correctly aligned .. just luck
59         char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0, 0};
60
61         const auto result = processData(buffer, sizeof(buffer));
62         std::println("{}", result);
63     }
64
65     {   /// let's force an unalignment --> first assert will trigger
66         alignas(Coordinate) char buffer[] = {10, 10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0, 0};
67
68         const auto result = processData(buffer + 1, sizeof(buffer) - 1);
69         std::println("{}", result);
70     }
71
72     {   /// 1 byte missing will trigger -> the second assert
73         alignas(Coordinate) char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0, 20, 0, 0, 0, 20, 0, 0};
74
75         const auto result = processData(buffer, sizeof(buffer));
76         std::println("{}", result);
77     }
```

- https://godbolt.org/z/3oz3ExPGG

Coming back on bit_cast

- Does not require alignment

main.cpp

```cpp
namespace
{

struct Coordinate
{
    int x{};
    int y{};
};

consteval int foo()
{
    /*alignas(Coordinate)*/ char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0};  /// bit cast does not require alignment
    auto coord = std::bit_cast<Coordinate>(buffer);
    return coord.x;
}

} // namespace

int main()
{
    constexpr int cox = foo();
    static_assert(cox == 10);

    {
        /// let's hand craft an array of bytes, assuming it could have come from the network
        alignas(Coordinate) char buffer[] = {10, 0, 0, 0, 10, 0, 0, 0};
        auto coord = std::bit_cast<Coordinate>(buffer);

        std::println("{},{}", coord.x, coord.y);
    }
}
```

No std::start_lifetime_as yet ?

- Implement it yourself

```
 9   namespace My
10   {
11   template<typename T>
12   T* start_lifetime_as(void* ptr)
13   {
14       return static_cast<T*>(std::memmove(ptr, ptr, sizeof(T)));
15   }
16   } // namespace My
```

My Hex editor

- I want to inspect the bytes

My Hex editor : old style: UB

- There just is no array of bytes, just an int
- The for loop gives the UB: dereference, and pointer arithmetic

```
5   namespace
6   {
7
8   constexpr int value{0x24AB50CD};
9
10  } // namespace
11
12  int main()
13  {
14      std::println("{}", value);
15      const auto bytes = reinterpret_cast<const unsigned char*>(&value);
16      for(int i = 0; i < 4; ++i)
17      {
18          std::println("{:x} aka {}", bytes[i], static_cast<int>(bytes[i]));  /// UB
19      }
20  }
```

```
615207117
cd aka 205
50 aka 80
ab aka 171
24 aka 36
```

My Hex editor : correct way

- let's also add a consteval → UB detector

```
8   constexpr int value{0x24AB50CD};
9
10  consteval int foo()
11  {
12      const auto bytes = std::bit_cast<std::array<unsigned char, sizeof(int)>>(value);
13      int sum{};
14      for(const auto& abyte: bytes)
15      {
16          sum += abyte;
17      }
18      return sum;
19  }
20
21  } // namespace
22
23  int main()
24  {
25      std::println("{}", value);
26      const auto bytes = std::bit_cast<std::array<unsigned char, sizeof(int)>>(value);
27      for(const auto& abyte: bytes)
28      {
29          std::println("{:x} aka {}", abyte, static_cast<int>(abyte));
30      }
31
32      constexpr auto sum = foo();
33      static_assert(sum == 492);
34      std::println("sum: {}", sum);
35  }
```

```
615207117
cd aka 205
50 aka 80
ab aka 171
24 aka 36
sum: 492
```

My Hex editor : Unfortunately

- Padding bytes :-(

```cpp
 8  struct S1
 9  {
10      int x;
11      bool a;
12      int y;
13      bool b;
14  };
```

```cpp
16  struct Coordinate
17  {
18      int x;
19      int y;
20  };
```

```cpp
const auto bytes = std::bit_cast<std::array<unsigned char, sizeof(S1)>>(value);
for(const auto& abyte: bytes)
{
    std::println("{:x} aka {}", abyte, static_cast<int>(abyte)); /// erroneous for the padding bytes ?
}
```

```cpp
22  constexpr S1 value{0x24AB50CD, true, 0x24AB50CD, false};
23  constexpr Coordinate value2{0x24AB50CD, 0x24AB50CD};
24
25  consteval int foo()
26  {
27  //    const auto bytes = std::bit_cast<std::array<unsigned char, sizeof(S1)>>(value);
28      const auto bytes = std::bit_cast<std::array<unsigned char, sizeof(Coordinate)>>(value2);
29      int sum{};
30      for(const auto& abyte: bytes)
31      {
32          sum += abyte;    /// will not work for S1, the padding bytes --> uninitialized object
33      }
34      return sum;
35  }
```

My Hex editor : Unfortunately

```cpp
const auto bytes = std::bit_cast<std::array<unsigned char, sizeof(S1)>>(value);
for(const auto& abyte: bytes)
{
    std::println("{:x} aka {}", abyte, static_cast<int>(abyte)); /// erroneous for the padding bytes ?
}
```

```
cd aka 205
50 aka 80
ab aka 171
24 aka 36
1 aka 1
0 aka 0
0 aka 0
0 aka 0
cd aka 205
50 aka 80
ab aka 171
24 aka 36
0 aka 0
0 aka 0
0 aka 0
0 aka 0
```

```
g++  -o ../Deliv/GccDebug/InspectTheBytesS1 ../Deliv/GccDebug/src/main.o
/home/lieven/Teaching/TypePunning/InspectTheBytesS1/src/main.cpp: In function ‘int main()’:
/home/lieven/Teaching/TypePunning/InspectTheBytesS1/src/main.cpp:41:29:   in ‘constexpr’ expansion of ‘{anonymous}::foo()’
/home/lieven/Teaching/TypePunning/InspectTheBytesS1/src/main.cpp:32:16: error: accessing uninitialized array element
   32 |         sum += abyte;   /// will not work for S1, the padding bytes --> uninitialized object
      |                ^~~~~
```

__attribute__((packed))

- Non standard
- To avoid/remove padding
- Alignment will be broken ==> UB
- Can work but performance hit
- can crash

```
15
16  struct S
17  {
18      uint64_t x;
19      uint8_t y;
20      uint16_t z;
21  };
22
23  static_assert(alignof(S) == 8);
24  static_assert(sizeof(S) == 16);
25
26  struct SPack
27  {
28      uint64_t x;
29      uint8_t y;
30      uint16_t z;
31  } __attribute__((packed));
32
33  static_assert(alignof(SPack) == 1);
34  static_assert(sizeof(SPack) == 11);
35
```

```
43      SPack arr[5];
44          /// with some luck arr[0].x is aligned, but for sure arr[1].x is not
45      std::println("{} {}", (void*)&arr[0].x, (void*)&arr[1].x);
```

```
S: size: 16 , alignment: 8
SPack: size: 11 , alignment: 1
0x7ffd0ac6a210 0x7ffd0ac6a21b
```

__attribute__((packed))

```
 7  struct S
 8  {
 9      uint64_t x;
10      uint8_t y;
11      uint16_t z;
12  } __attribute__((packed));
13
14  static S s[10];
15
16  uint64_t access0()
17  {
18      s[0].x = 242;
19      return s[0].x;
20  }
21
22  uint64_t access1()
23  {
24      s[1].x = 242;
25      return s[1].x;
26  }
27  } // namespace
28
29  int main()
30  {
31      std::println("{} {}", access0(), access1());
32  }
```

- Let’s inspect the assembly at compiler explorer
- https://godbolt.org/z/66Yxrnvov
- arm 32 bit
- access1() way longer than access0()

No copies

```cpp
struct Coordinate
{
    int x{};
    int y{};
};

void process(char* buffer)
{
    decltype(auto) coor = std::launder(reinterpret_cast<Coordinate*>(buffer)); /// note launder will not help, since the
    coor->x = 1;
    coor->y = 2;
}

void processMemCpy(char* buffer)    /// compilers will recognize and optimize, but they might miss it
{
    Coordinate coor;
    std::memcpy(&coor, buffer, sizeof(Coordinate));

    coor.x = 3;
    coor.y = 4;

    std::memcpy(buffer, &coor, sizeof(Coordinate));
}

void process20(char* buffer)
{
    decltype(auto) coor = static_cast<Coordinate*>(std::memmove(buffer, buffer, sizeof(Coordinate)));

    coor->x = 5;
    coor->y = 6;
}

void process23(char* buffer)
{
    decltype(auto) coor = std::start_lifetime_as<Coordinate>(buffer);

    coor->x = 7;
    coor->y = 8;
}
```

x86-64 gcc (trunk) -std=c++23 -O1

```asm
"process(char*)":
        mov     DWORD PTR [rdi], 1
        mov     DWORD PTR [rdi+4], 2
        ret
"processMemCpy(char*)":
        movabs  rax, 17179869187
        mov     QWORD PTR [rdi], rax
        ret
"process20(char*)":
        mov     DWORD PTR [rdi], 5
        mov     DWORD PTR [rdi+4], 6
        ret
"process23(char*)":
        mov     DWORD PTR [rdi], 7
        mov     DWORD PTR [rdi+4], 8
        ret
```

Output (0/0) x86-64 gcc (trunk) - 5454ms (174743B) ~11828 lines filtered

Output of x86-64 gcc (trunk) (Compiler #1)

```
ASM generation compiler returned: 0
/cefs/e0/e0dd74b26364d606eb9392a4_gcc-trunk-20260223/bin/../lib/gcc/x86_6
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
Execution build compiler returned: 1
 Build failed
```

No copies

```cpp
25  static_assert( std::is_implicit_lifetime_v<Coordinate>); /// clang has it, coming in gcc16
26
27  void halfCoordinate(Coordinate& co)
28  {
29      co.x /= 2;
30      co.y /= 2;
31  }
32
33  void processDataUB(char* theBytes)
34  {
35      auto* co = reinterpret_cast<Coordinate*>(theBytes);
36      halfCoordinate(*co);
37
38  //    std::println("{},{}", co->x, co->y);
39  }
40
41  void processDataMemCpy(char* theBytes)
42  {
43      Coordinate co;
44      std::memcpy(&co, theBytes, sizeof(Coordinate));
45      halfCoordinate(co);
46
47
48      std::memcpy(theBytes, &co, sizeof(Coordinate));
49
50  //    std::println("{},{}", co.x, co.y);
51  }
52
53  void processDataMemMove(char* theBytes)
54  {
55      auto* co = static_cast<Coordinate*>(std::memmove(theBytes, theBytes, sizeof(Coordinate)));
56      halfCoordinate(*co);
57
58  //    std::println("{},{}", co->x, co->y);
59  }
60
61  void processData(char* theBytes)
62  {
63      auto* co = std::start_lifetime_as<Coordinate>(theBytes);
64      halfCoordinate(*co);
65
66  //    std::println("{},{}", co->x, co->y);
67  }
68
69  void processData2(char* theBytes)
70  {
71      auto* co = My::start_lifetime_as<Coordinate>(theBytes);
72      halfCoordinate(*co);
73
74  //    std::println("{},{}", co->x, co->y);
75  }
```

A ▾ ⚙ Output... ▾ Filter... ▾ Libraries Overrides + Add new

```asm
1   "halfCoordinate(Coordinate&)":
2           mov     edx, DWORD PTR [rdi]
3           mov     eax, edx
4           shr     eax, 31
5           add     eax, edx
6           sar     eax
7           mov     DWORD PTR [rdi], eax
8           mov     edx, DWORD PTR [rdi+4]
9           mov     eax, edx
10          shr     eax, 31
11          add     eax, edx
12          sar     eax
13          mov     DWORD PTR [rdi+4], eax
14          ret
15  "processDataUB(char*)":
16          call    "halfCoordinate(Coordinate&)"
17          ret
18  "processDataMemCpy(char*)":
19          push    rbx
20          sub     rsp, 16
21          mov     rbx, rdi
22          mov     rax, QWORD PTR [rdi]
23          mov     QWORD PTR [rsp+8], rax
24          lea     rdi, [rsp+8]
25          call    "halfCoordinate(Coordinate&)"
26          mov     rax, QWORD PTR [rsp+8]
27          mov     QWORD PTR [rbx], rax
28          add     rsp, 16
29          pop     rbx
30          ret
31  "processDataMemMove(char*)":
32          call    "halfCoordinate(Coordinate&)"
33          ret
34  "processData(char*)":
35          call    "halfCoordinate(Coordinate&)"
36          ret
37  "processData2(char*)":
38          call    "halfCoordinate(Coordinate&)"
39          ret
```

Output (0/0) x86-64 gcc (trunk) - 2372ms (183461B) ~12357 lines filtered

What about those horrible void* APIs

- Would this ancient relic finally have died ?
  - No :-(
  - Let’s look at a more or less modern library: opentelemetry

Opentelemetry: poor API

```
// Copyright The OpenTelemetry Authors
// SPDX-License-Identifier: Apache-2.0

#pragma once

#include "opentelemetry/metrics/observer_result.h"
#include "opentelemetry/version.h"

OPENTELEMETRY_BEGIN_NAMESPACE
namespace metrics
{

using ObservableCallbackPtr = void (*)(ObserverResult, void *);

class ObservableInstrument
{
public:
  ObservableInstrument()          = default;
  virtual ~ObservableInstrument() = default;

  /**
   * Sets up a function that will be called whenever a metric collection is initiated.
   */
  virtual void AddCallback(ObservableCallbackPtr, void *state) noexcept = 0;

  /**
   * Remove a function that was configured to be called whenever a metric collection is initiated.
   */
  virtual void RemoveCallback(ObservableCallbackPtr, void *state) noexcept = 0;
};

}  // namespace metrics
OPENTELEMETRY_END_NAMESPACE
```

- Register callback, which takes 2 arguments, the second being void*
- So no type safety, it can be anything
- Rules of the game:
  - Stuff in what you want
  - Cast the void* to the thing you stuffed in (and pray)

Opentelemetry: poor API

- Let’s assume we have a class (MyClass) which owns such an observable instrument (mInstr) and wants to register a static member method as the callback (foo)

```
106  MyClass::MyClass()
107      /// some init list --> mInstr
108  {
109      mInstr->AddCallback(foo, (void*)this);
110  }
111
112  void MyClass::foo(opentelemetry::metrics::ObserverResult result, void* sourcePtr)
113  {
114      const auto& self = *(static_cast<MyClass*>(sourcePtr));
115
116      /// let's get to work with 'self'
117  }
```

The good news: works

- Why does it work ?
  - There lives an object of type MyClass there
    - Assuming we keep it correctly alive
  - So the void* can be casted back to a pointer to the type of that object, since it lives there, we are not casting it to another type (so not int to float, but int* – void* - int*)

Alternative type safe solution

- And one can bind whatever one wants into the std::function

```
121 using ObservableCallback = std::function<void (ObserverResult)>;
122
123   virtual void AddCallback(ObservableCallback) noexcept = 0;
124
125 MyClass::MyClass()
126     /// some init list --> mInstr
127 {
128     mInstr->AddCallback(std::bind_front(&MyClass::foo, this));
129 }
130
131 void MyClass::foo(opentelemetry::metrics::ObserverResult result)
132 {
133     /// regular member method now, and we can use the members as is
134     /// no casting needed
135 }
```

Conclusions

- There are many reasons why reinterpret_cast results in UB
- Stay up to date
  - C++20: std::bit_cast, lifetime starting functions
  - C++23: std::start_lifetime_as(_array)
- Start addressing your reinterpret_casts
- In case of old C-style casts, first get rid of those and replace them by the C++ dedicated casts
  - How can we find those, hard to grep → see next slide

Find C-style casts

- Turn them into a compiler warning
- GCC/Clang: **-Wold-style-cast**

```
2   int main()
3   {
4   
5       const float f{242.0f};
6   
7       const int i = (int)f;
8   
9       return i;
10  }
```

```
/home/lieven/Teaching/TypePunning/WarnOldStyleCast/src/main.cpp: In function ‘int main()’:
/home/lieven/Teaching/TypePunning/WarnOldStyleCast/src/main.cpp:7:24: warning: use of old-style cast to ‘int’ [-Wold-style-cast]
    7 |     const int i = (int)f;
      |                        ^
      |                   ------
      |                   static_cast<int> (f)
```

```
/home/lieven/Teaching/TypePunning/WarnOldStyleCast/src/main.cpp:7:19: warning: use of old-style cast [-Wold-style-cast]
    7 |     const int i = (int)f;
      |                   ^    ~
1 warning generated.
```

References

- https://andreasfertig.com/blog/2025/03/the-correct-way-to-do-type-punning-in-cpp
- https://andreasfertig.com/blog/2025/04/the-correct-way-to-do-type-punning-in-cpp-the-second-act
- Timur Doumler : Type punning in modern c++ : Cppcon 2019
  - https://www.youtube.com/watch?v=_qzMpk-22cc&t=2770s
- Javier Lopez Gomez: Type punning done right: Using std::cpp 2025
  - https://www.youtube.com/watch?v=NRV_bgN92DI
- Alex Dathskosvky: Misusing reinterpret_cast ? Your probably are :) : Cpp North 2025
  - https://www.youtube.com/watch?v=xxCtaAiEIcQ
- Robert Leahy: Taking a Byte Out of C++ - Avoiding Punning by Starting Lifetimes: Cppcon 2022
  - https://www.youtube.com/watch?v=pbkQG09grFw
- Jason Turner: C++ Weekly 185 : Stop Using reinterpret_cast
  - https://www.youtube.com/watch?v=L06nbZXD2D0
- Jason Turner: C++ Weekly 512 : reinterpret_cast is Finally Fixed!
  - https://www.youtube.com/watch?v=JtFVyXQ00PQ
- Jonathan Muller: An (In-)Complete Guide to C++ Object Lifetimes: C++ Now 2024
  - https://www.youtube.com/watch?v=oZyhq4D-QL4
- Patrice Roy: C++ Memory Management: Write leaner and safer C++ code using proven memory-management techniques
  - https://www.amazon.com/Memory-Management-leaner-memory-management-techniques/dp/1805129805

QUESTIONS

There’s more ...

- std::launder
- Pointer provisioning
- Pointer-interconvertible
- Common initial sequence