
2022
C++ Puzzlers
Yuri Minaev

About me

PVS-Studio C++ analyser's architect.

I love C++, cats, and whining about legacy code

What is this talk about?

- Exploring dusty corners of C++
- Interactivity
- Hopefully, learning something new
- Having fun

C+++++

C+++++

```
int main()
{
  int a = 5;
  int b = 2;
  std::cout << a+++++b;
}
```

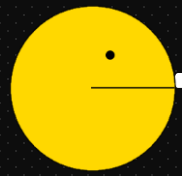
C+++++

- Does it compile?
- Why?

```
int main()
{
  int a = 5;
  int b = 2;
  std::cout << a+++++b;
}
```

C+++++

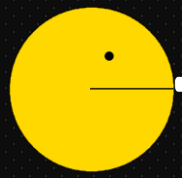

a+++++b


++++++b

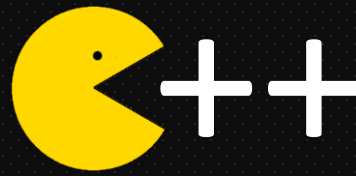
a :*identifier*

C+++++

a :*identifier*



\+ :*addition*



\++ :*increment*

C+++++

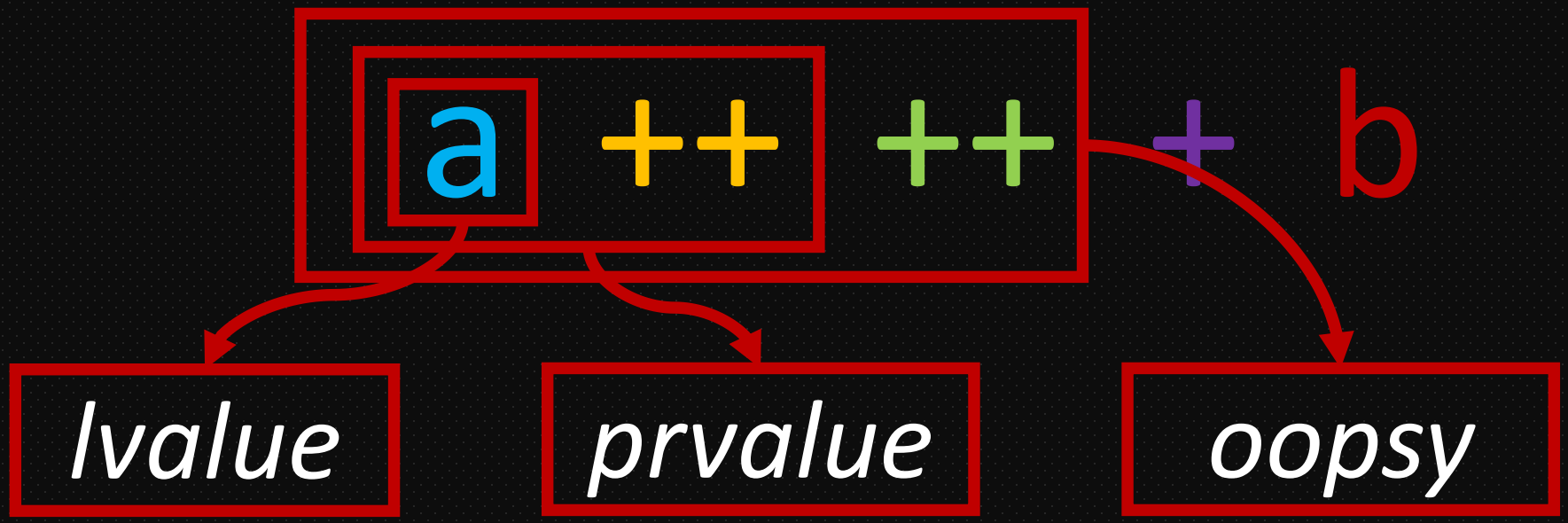
C+++++

a+++++b

a :*identifier* ++ :*increment* ++ :*increment*

\+ :*unary* b :*identifier*

C+++++



The operand expr of a built-in postfix increment or decrement operator must be a modifiable (non-const) lvalue of non-boolean (since C++17) arithmetic type or pointer to completely-defined object type. © cppreference.com

Templates everywhere

Templates everywhere

```
template <typename T>
T* gimme_addr(T &ref) { return &ref; }

struct thing;

template
const thing* gimme_addr(const thing &ref);

struct thing
{
  const thing* operator&() const;
};
```

Templates everywhere

- Does it compile?
- What happens if it does?

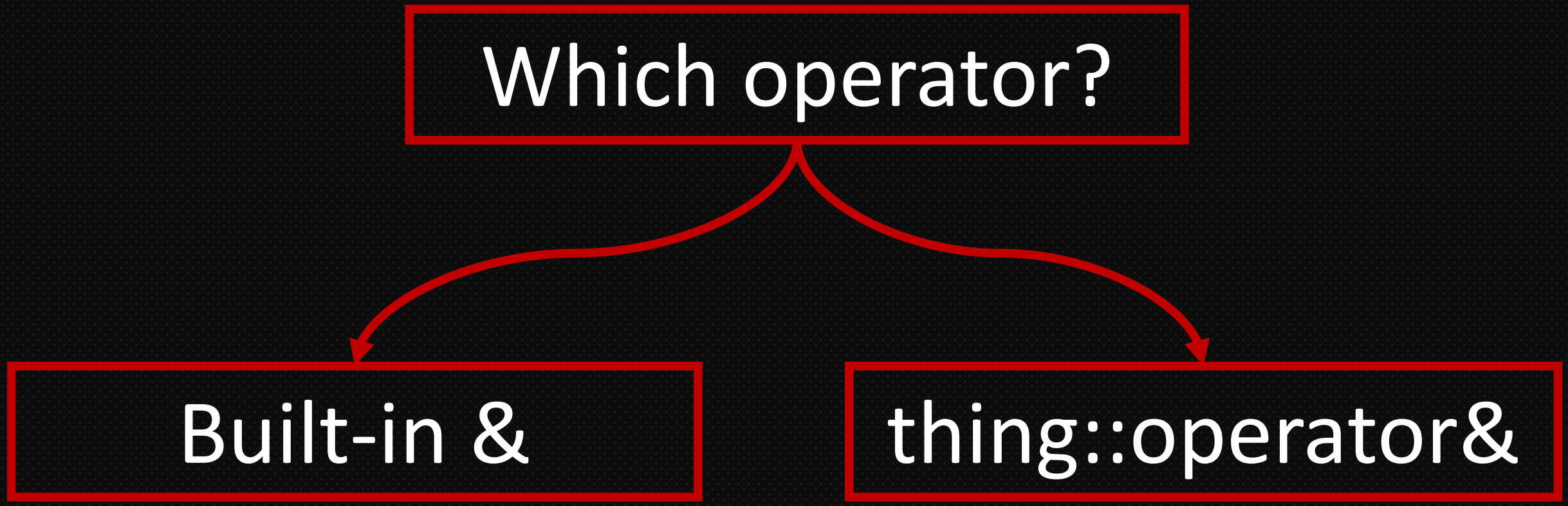
```
template <typename T>
T* gimme_addr(T &ref) { return &ref; }

struct thing;

template
const thing* gimme_addr(const thing &ref);

struct thing
{
  const thing* operator&() const;
};
```

Templates everywhere



Templates everywhere

```
template <typename T>
T* gimme_addr(T &ref) { return &ref; }

struct thing;

template
const thing* gimme_addr(const thing &ref);

struct thing
{
  const thing* operator&() const;
};
```

Templates everywhere

Which operator?

x86-64 gcc 12.1 -O2

Output... Filter... Libraries Add new...

```
1  thing const* gimme_addr<thing const>(thing const&):
2          jmp     thing::operator&() const
```

x86-64 clang 14.0.0 -O2

Output... Filter... Libraries Add new...

```
1  thing const* gimme_addr<thing const>(thing const&):
2          mov     rax, rdi
3          ret
```

x64 msvc v19.latest /O2

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

```
1  ref$ = 8
2  thing const * gimme_addr<thing const >(thing const &) PROC
3          jmp     thing const * thing::operator&(void)const
4  thing const * gimme_addr<thing const >(thing const &) ENDP
```

Templates everywhere

If & is applied to an lvalue of incomplete class type and the complete type declares operator&(), it is unspecified whether the operator has the built-in meaning or the operator function is called.

The operand of & shall not be a bit-field.

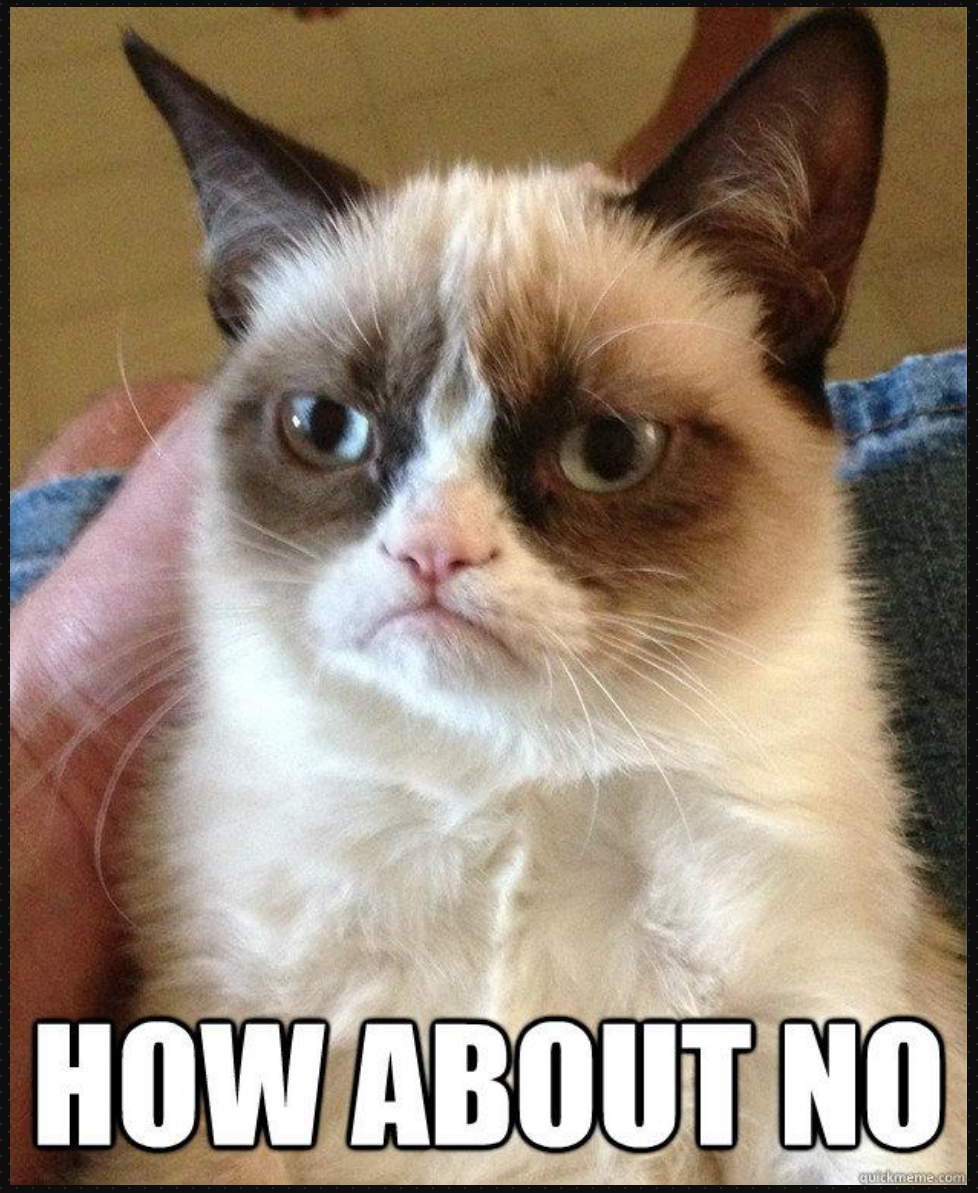
© C++ Standard 7.6.2.1.5

Know thy traits

Know thy traits

- We need Data Flow analysis for C++
- To store and track a value, you need a larger container size
- I.e. 128 bits for your average 64-bit variable
- int128 anyone?

Know thy traits



Know thy traits

```
class Int128 { /*Spooky things*/ };

template <>
struct std::is_signed<Int128>
      : std::true_type {};

static_assert(std::is_signed_v<Int128>);
```

Know thy traits

- Why are my tests failing like crazy all of a sudden?

```
class Int128 { /*Spooky things*/ };

template <>
struct std::is_signed<Int128>
      : std::true_type {};

static_assert(std::is_signed_v<Int128>);
```

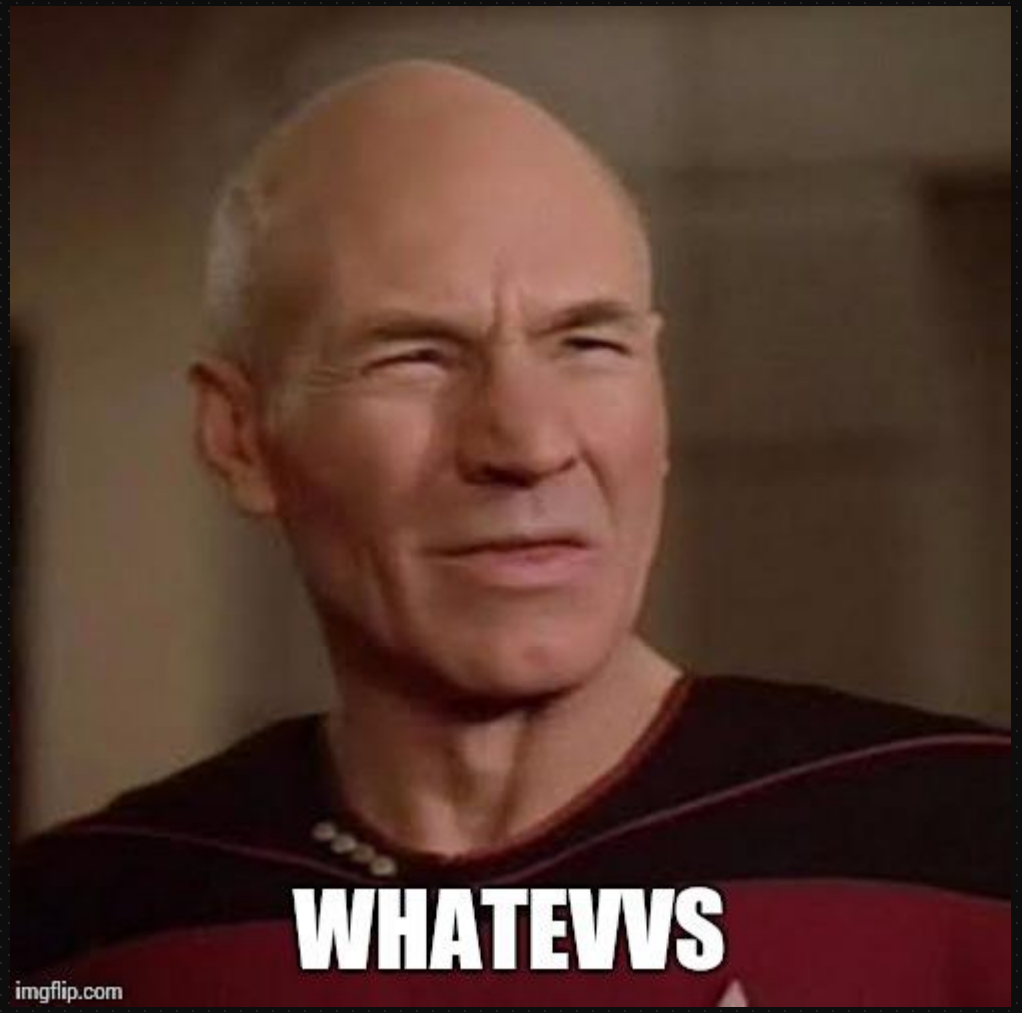
Know thy traits

x86-64 gcc 11.2

x86-64 clang 14.0.0

x64 msvc v19.20

Know thy traits



x64 msvc v19.21

```
example.cpp
<source>(13): error C2607: static assertion failed
Compiler returned: 2
```

Know thy traits

None of the templates defined in <type_traits> may be specialized for a program-defined type, except for std::common_type and std::basic_common_reference (since C++20). This includes the type traits and the class template std::integral_constant.

© cppreference.com

Schrödinger's variable

Schrödinger's variable

```
volatile int a;


int main()
{
  std::cout << a + a;
}
```

Schrödinger's variable

- Does it compile?
- Is this even legal?
- What is going to happen?

```
volatile int a;


int main()
{
   std::cout << a + a;
}
```

Schrödinger's variable

Reading an object designated by a volatile glvalue, modifying an object, calling a library I/O function, or calling a function that does any of those operations are all *side effects*, which are changes in the state of the exe

© C++ Standard 6.9.1.7

Schrödinger's variable

Except where noted, evaluations of operands of individual operators and of subexpressions of individual expressions are unsequenced.[...]

If a side effect on a memory location is unsequenced relative to either another side effect on the same memory location or a value computation using the value of any object in the same memory location, and they are not potentially concurrent, the behavior is undefined.

© C++ Standard 6.9.1.10

Schrödinger's variable

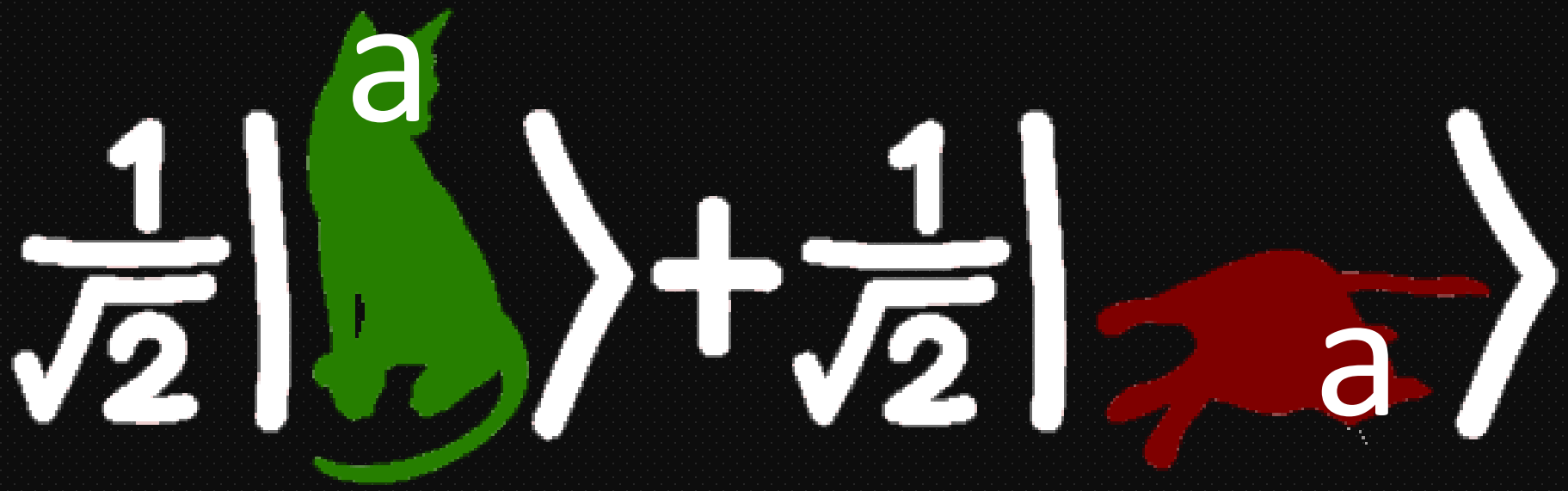
```
std::cout << a + a;
```

== UB*

* In case a is a volatile entity**

** The big 3 don't seem to care

Schrödinger's variable



The future is now

The future is now

```
auto thing = "fluffy"sv;

std::async(std::launch::async,
           [&thing] { thing = "spooky"sv; });

std::async(std::launch::async,
           [&thing] { thing = "jabberwock"sv; });

std::cout << thing << std::endl;
```

The future is now

- What will it cout?
- Is there, by chance, a data race here?

```
auto thing = "fluffy"sv;

std::async(std::launch::async,
           [&thing] { thing = "spooky"sv; });

std::async(std::launch::async,
           [&thing] { thing = "jabberwock"sv; });

std::cout << thing << std::endl;
```

The future is now

Output?

x86-64 gcc 11.2

```
Program returned: 0
Program st
jabberwock
```

x86-64 clang 14.0.0

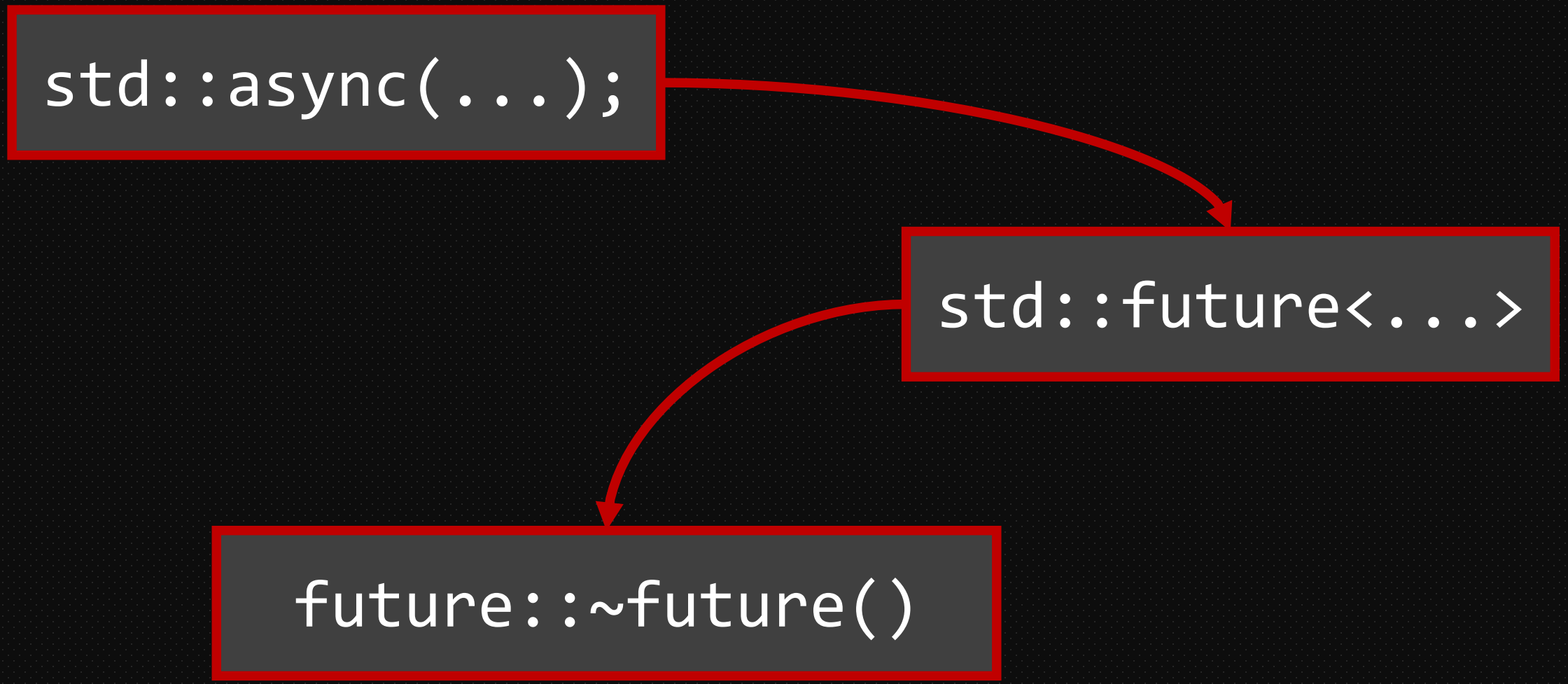
```
Program returned: 0
Program stdout
jabberwock
```

x64 msvc v19.latest /O2 /std:c++17

```
Compiler stdout
example.cpp
<source>(10): warning C4834: discarding return value of function with 'nodiscard' attribute
<source>(12): warning C4834: discarding return value of function with 'nodiscard' attribute

Program returned: 0
Program stdout
jabberwock
```

The future is now



The future is now

If the implementation chooses the launch::async policy
[...]
the associated thread completion synchronizes with the return from the first function that successfully detects the ready status of the shared state or with the return from the last function that releases the shared state, whichever happens first.

© C++ Standard 32.9.9.6.4

The future is now

```
auto thing = "fluffy"sv;

std::async(std::launch::deferred,
           [&thing] { thing = "spooky"sv; });

std::async(std::launch::deferred,
           [&thing] { thing = "jabberwock"sv; });

std::cout << thing << std::endl;
```

The future is now

```
auto thing = "fluffy"sv;

std::async(std::launch::deferred,
           [&thing]
           {
             thing = "spooky"sv;
           })
     .wait_for(5s);

std::cout << thing << std::endl;
```

The future is now

Output?

x86-64 gcc 11.2

```
Program returned: 0
Program st
fluffy
```

x86-64 clang 14.0.0

```
Program returned: 0
Program stdout
fluffy
```

x64 msvc v19.latest | /O2 /std:c++17

```
Compiler stdout
example.cpp
<source>(10): warning C4834: discarding return value of function with 'nodiscard' attribute
<source>(12): warning C4834: discarding return value of function with 'nodiscard' attribute

Program returned: 0
Program stdout
fluffy
```

The future is now

If launch::deferred is set in policy[...]

The shared state is not made ready until the function has completed.
The first call to a non-timed waiting function on an asynchronous return object referring to this shared state invokes the deferred function in the thread that called the waiting function.

© C++ Standard 32.9.9.4.2

QUESTIONS