C++23 Tools That You Will Actually Use

About Me:

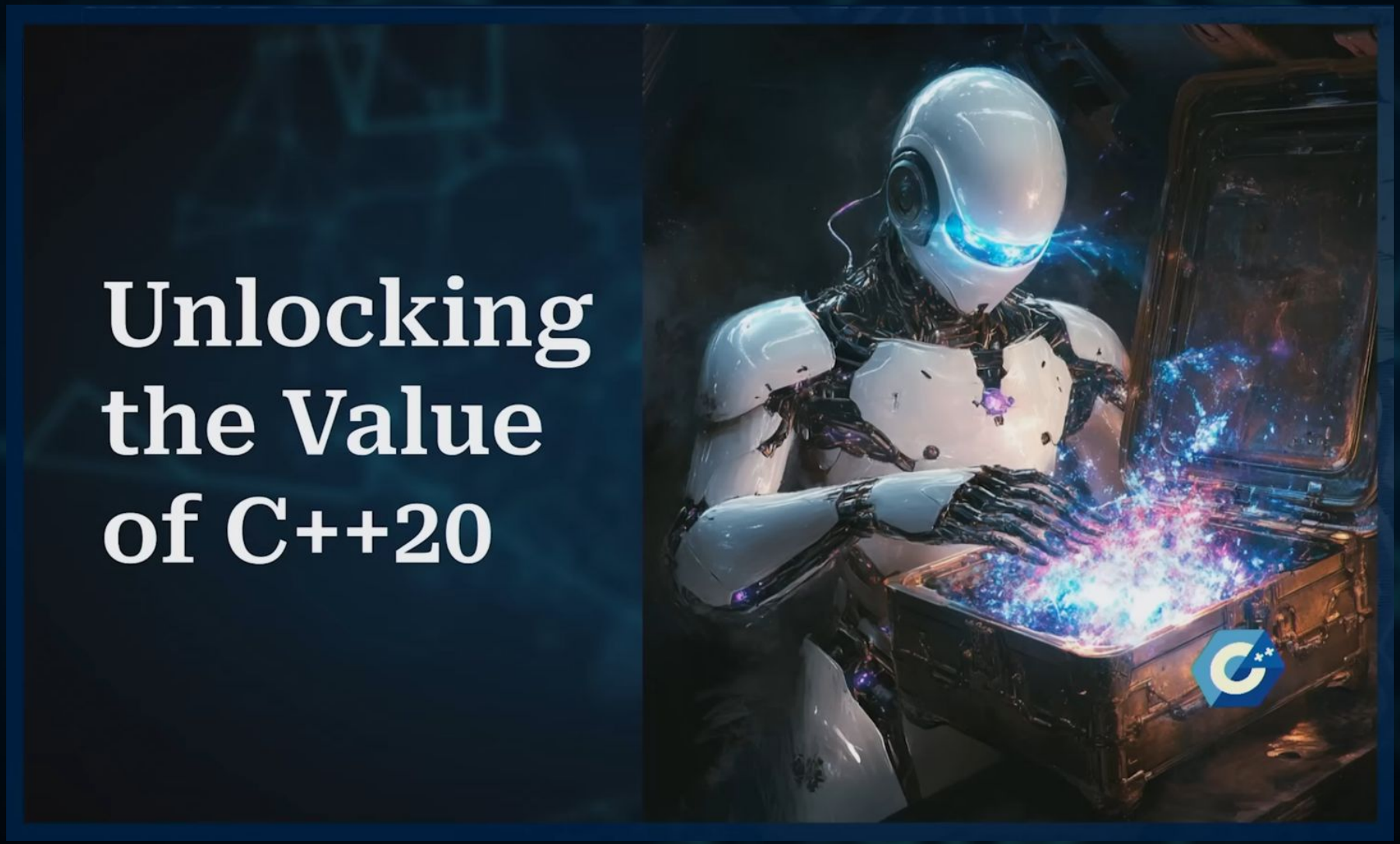
alex.datskovsky@scylladb.com

www.linkedin.com/in/alexdathskovsky

www.cppnext.com

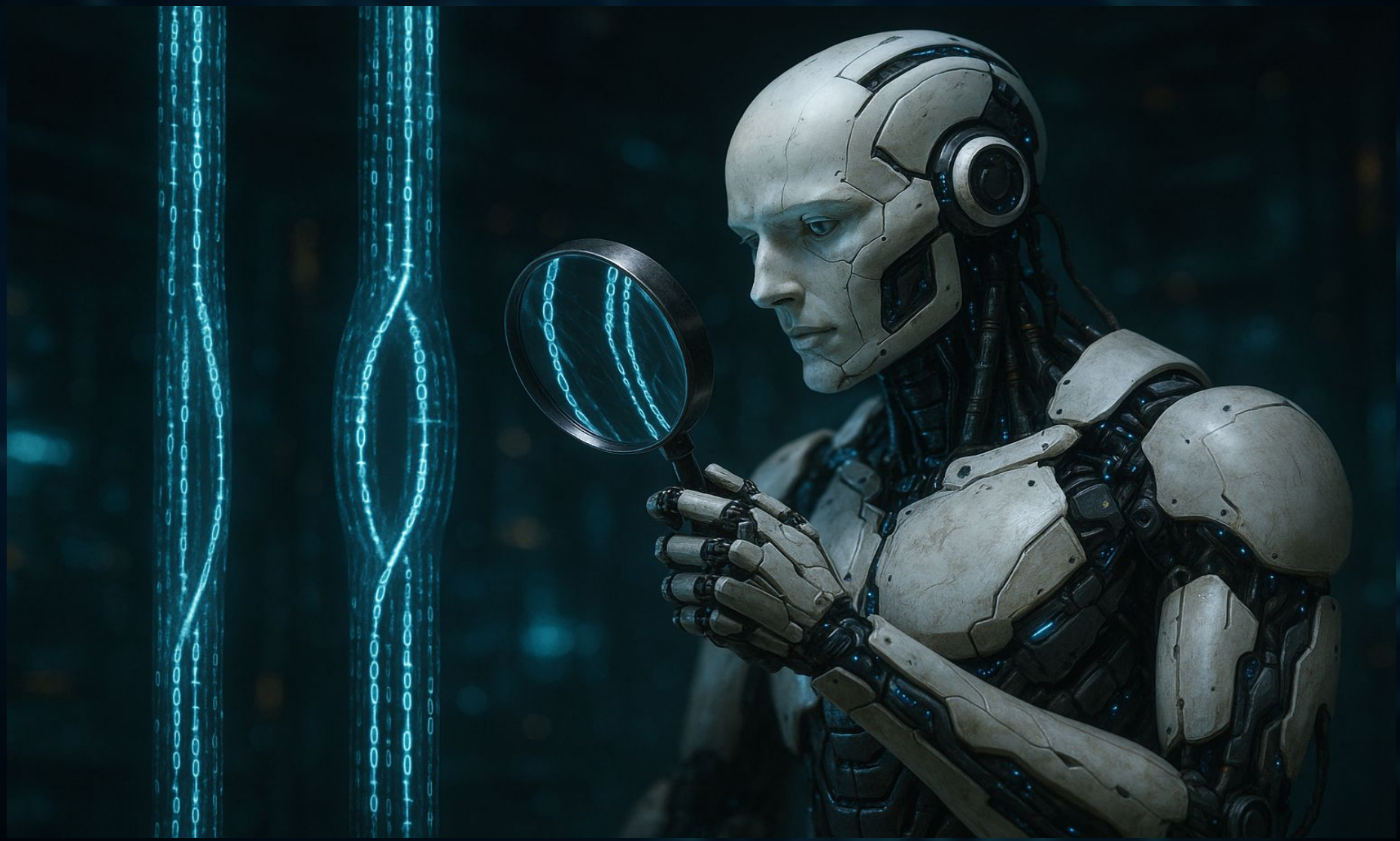
https://www.youtube.com/@cppnext-alexd

This is Funny



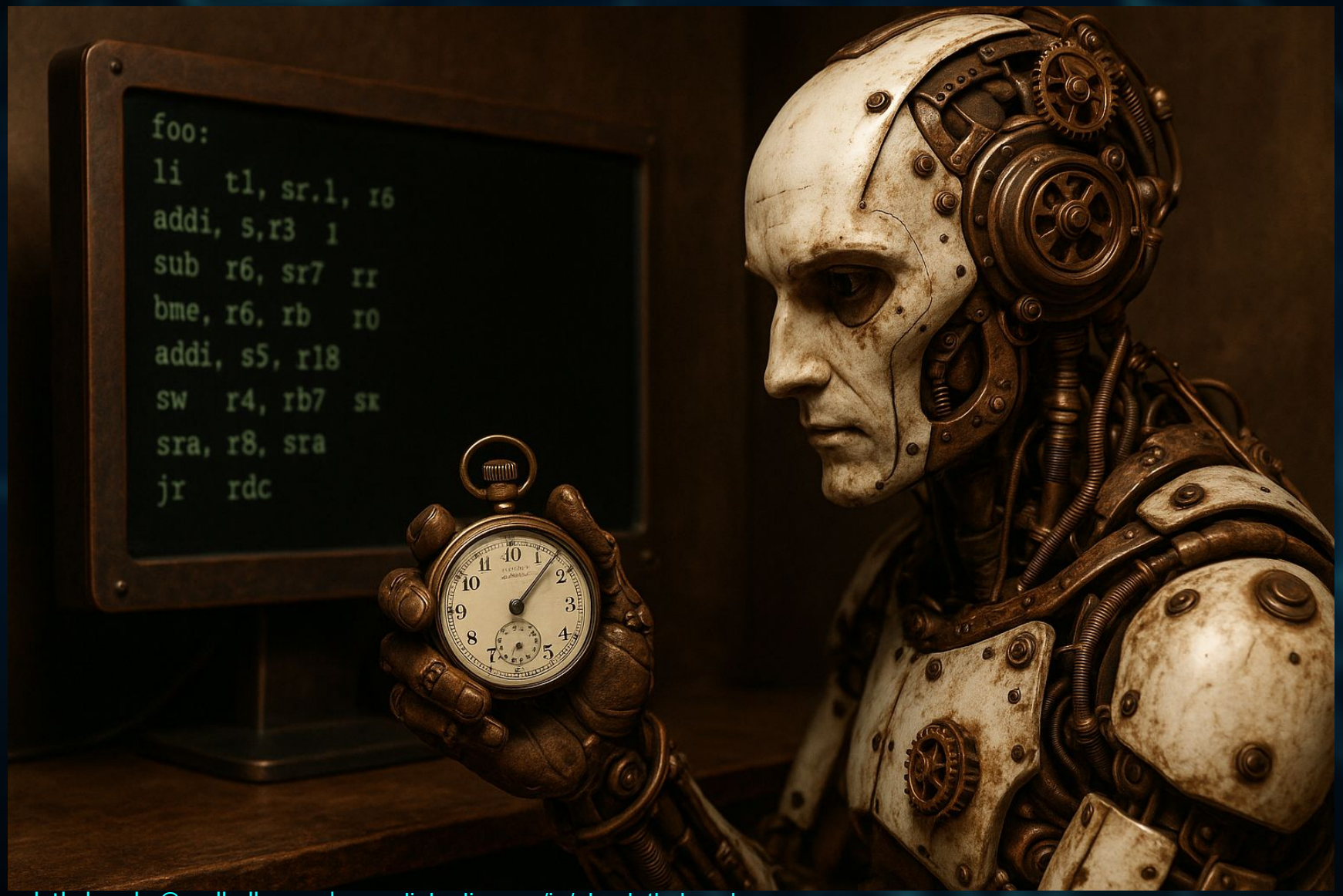
Alex Dathskovsky | alex.dathskovsky@scylladb.com| www.linkedin.com/in/alexdathskovsky

What to Expect



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

What to Expect



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Language Features

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Feature Testing

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Feature Testing

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Feature Testing

```
#if __cplusplus < 202000
#  error "We only allow C++20 and higher here!"
#endif
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Feature Testing

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Feature Testing

```
#ifdef __cpp_constexpr
#   if __cpp_constexpr < 201907
        #error "we need a version that suports constexpr virtual"
#   endif
#endif
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Feature Testing

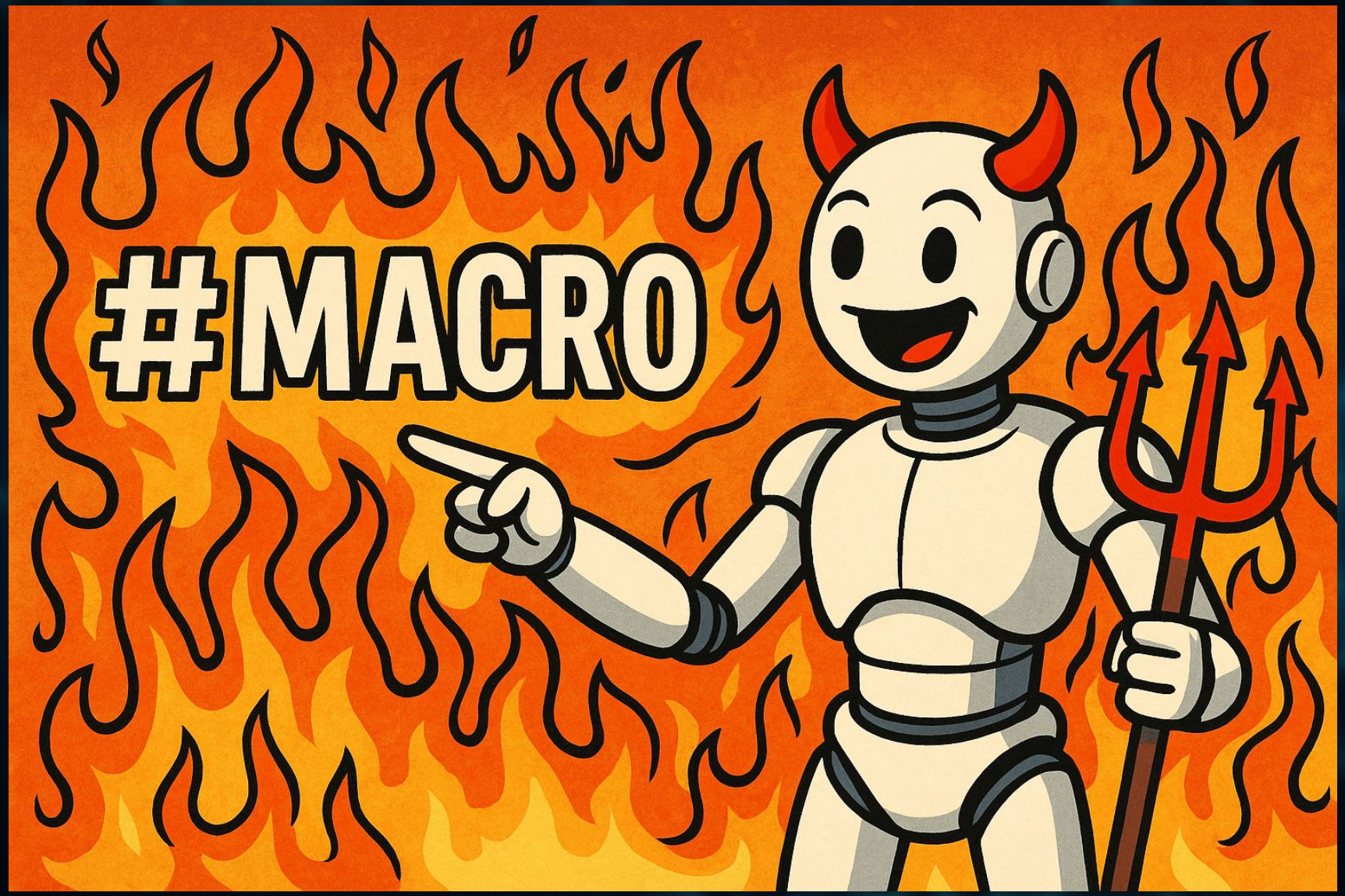
```
#ifdef __cpp_lib_constexpr_string
#   if __cpp_lib_constexpr_string < 201907
|   |     #error "we need string to have constexpr constructor"
#   endif
#endif
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Feature Testing

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

More Evil But Useful Macros



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

More Evil But Useful Macros

```
#ifdef X
    fmt::print("X\n");
#else
    #ifdef Y
        fmt::print("Y\n");
    #else
        #ifndef Z
            fmt::print("not Z\n");
        #else
            fmt::print("GO AWAY!\n");
        #endif
    #endif
#endif
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

More Evil But Useful Macros

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

#elifdef and #elifndef

```
#ifdef X
    fmt::print("X\n");
#elifdef Y
    fmt::print("Y\n");
#elifndef Z
    fmt::print("not Z\n");
#else
    fmt::print("GO AWAY!\n");
#endif
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

#warning

```
101     #if __cplusplus >= 202302L
102     #   warning "Warning about warnings"
103     #endif
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Multidimensional Array Access

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Multidimensional Array Access

```
47    int a[10][10][10] = {};
48    a[1][2][3] = 3;
49
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Multidimensional Array Access

a[1][2][3] = 3

a[1] -> b[10][10]

b[2] -> c[10]

c[3] -> &a[1][2][3]

Multidimensional Array Access

```
22  template <size_t SX, size_t SY, size_t SZ>
23  struct Array3D{
24      int& operator()(size_t x, size_t y, size_t z){
25          return array[x * (SY * SZ) + y * SZ + z];
26      }
27
28      public:
29      std::array<int, SX*SY*SZ> array = {};
30  };
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Multidimensional Array Access

```
61    Array3D<2, 3, 3> aa;
62    aa(1, 2, 1) = 3;
63    fmt::print("{}", aa.array);
```

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0]

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Multidimensional Array Access

```
22  template <size_t SX, size_t SY, size_t SZ>
23  struct Array3D{
24      int& operator[](size_t x, size_t y, size_t z){
25          return array[x * (SY * SZ) + y * SZ + z];
26      }
27
28      public:
29      std::array<int, SX*SY*SZ> array = {};
30  };
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Multidimensional Array Access

```
61    Array3D<2, 3, 3> aa;
62    aa[1, 2, 1] = 3;
63    fmt::print("{}", aa.array);
```

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 0]

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

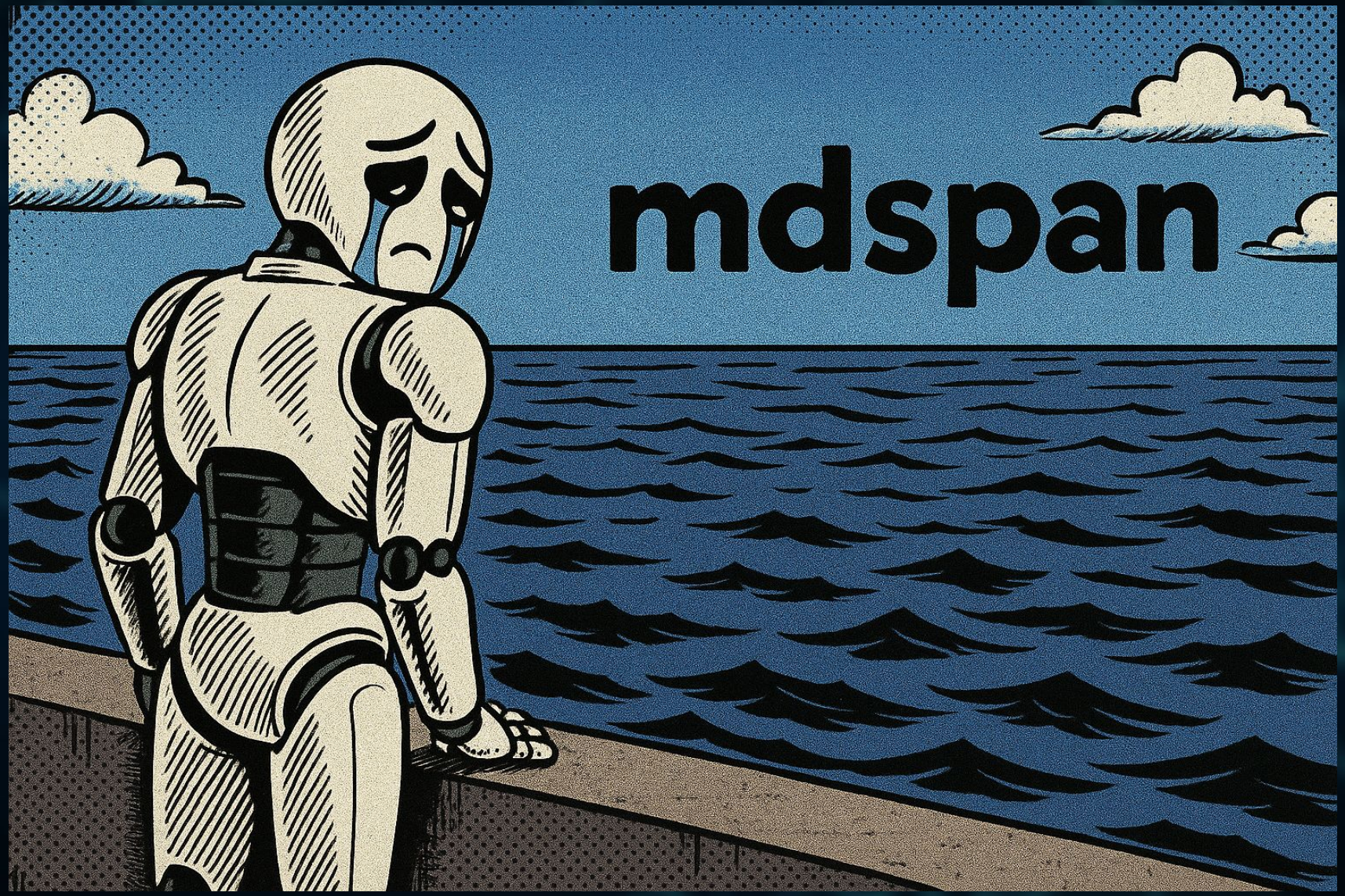
Pascal Does it Better

```
 9  program Hello;
10  var threedim: array[1..5, 1..5, 1..5] of integer;
11  begin
12    writeln ('Hello this is threedim[1,2,3] ', threedim[1,2,3]);
13  end.
```

Hello this is threedim[1,2,3] 0

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Pascal Does it Better



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Static Operators

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Static Operators

```
42  struct Test1{
43      bool operator()(int i) const;
44  };
45
46  struct Test2{
47      static bool f(int i);
48  };
49
50  static inline constexpr Test1 t1;
51
52  int count_1(const std::vector<int>& v){
53      return std::count_if(v.begin(), v.end(), t1);
54  }
55
56  int count_2(const std::vector<int>& v){
57      return std::count_if(v.begin(), v.end(), Test2::f);
58  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Static Operators

```
1   pushq %r15
2   pushq %r14
3   pushq %r12
4   pushq %rbx
5   pushq %rax
6   movq (%rdi), %r15
7   movq 8(%rdi), %r12
8   xorl %ebx, %ebx
9   cmpq %r12, %r15
10  je .LBB0_3
11  leaq 7(%rsp), %r14
12 .LBB0_2:
13  movl (%r15), %esi
14  movq %r14, %rdi
15  callq Test1::operator()(int) const@PLT
```

```
1   pushq %r15
2   pushq %r14
3   pushq %rbx
4   movq (%rdi), %r14
5   movq 8(%rdi), %r15
6   xorl %ebx, %ebx
7   cmpq %r15, %r14
8   je .LBB1_3
9  .LBB1_1:
10  movl (%r14), %edi
11  callq Test2::f(int)@PLT
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Static Operators

```
42  struct Test1{
43      static bool operator()(int i);
44  };
45
46  static inline constexpr Test1 t1;
47
48  int count_1(const std::vector<int>& v){
49      return std::count_if(v.begin(), v.end(), t1);
50  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Static Operators

```
2     pushq %r15
3     pushq %r14
4     pushq %rbx
5     movq (%rdi), %r14
6     movq 8(%rdi), %r15
7     xorl %ebx, %ebx
8     cmpq %r15, %r14
9     je .LBB0_3
10  .LBB0_1:
11    movl (%r14), %edi
12    callq Test1::operator()(int)@PLT
13    movzbl %al, %eax
14    addl %eax, %ebx
15    addq $4, %r14
16    cmpq %r15, %r14
17    jne .LBB0_1
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Power Lambdas

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Lambdas Can be Static

```
53  int count_2(const std::vector<int>& v){
54      return std::count_if(v.begin(), v.end(),
55          [](int i) static {return i > 5;});
56  }
```

```
 9  class __lambda_54_9
10  {
11    public:
12    static inline /*constexpr */ bool operator()(int i)
13    {
14      return i > 5;
15    }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Attributes on Lambdas

```
93      auto l = [] [[nodiscard]] (int x) static noexcept [[]]
94      { return x+1; };
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Decaying Copy: auto(x) and auto{x}

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

auto(x) or auto{x}

```
65  void pop_back1(std::vector<int>& x) {
66      auto a = x.back();
67      std::erase(x, a);
68  }
69
70  void pop_back2(std::vector<int>& x) {
71      std::erase(x, x.back());
72  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

auto(x) or auto{x}

pop_back1

```
36 .LBB0_16:
37   addi a4, a4, 4
38   beq a4, a1, .LBB0_14
39 .LBB0_17:
40   lw a5, 0(a4)
41   beq a5, a2, .LBB0_16
42   sw a5, 0(a3)
```

pop_back2

```
36 .LBB1_16:
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

auto(x) or auto{x}

```
74    void pop_back(std::vector<int>& x) {
75        std::erase(x, auto(x.back()));
76    }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

auto(x) or auto{x}

```
78  void pop_back_T(std::vector<int>& x) {
79      using T = std::remove_cvref_t<decltype(x.back())>;
80      std::erase(x, T(x.back()));
81  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

auto(x) or auto{x}

```
83  struct M;
84
85  struct S
86  {
87      S* operator()();
88      int N;
89      int M;
90
91      void getm(S s)
92      {
93          // expression (S::M hides ::M), invalid before C++23
94          auto(s)()->M;
95      }
96  };
97
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Literal Types

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Literal Types

```
47    int a1 = 0;
48    unsigned int a2 = 0U;
49    long a3 = 0L;
50    unsigned long a4 = 0UL;
51    long long a5 = 0LL;
52    unsigned long long a6 = 0ULL;
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Literal Types C++23 Addition

```
35        auto a7 = 0z;
36        auto a8 = 0uz;
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Literal Types C++23 Addition

```
53    long a7 = 0L;
54    unsigned long a8 = 0UL;
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Better Implicit Move Returns

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Better Implicit Move Returns

```
 99    int& bad(bool cond, int x) {
100        int y = 42;
101        if (cond)
102            return y;
103        else
104            return x;
105    }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Better Implicit Move Returns

```
<source>:102:16: error: cannot bind non-const lvalue reference of type 'int&' to an rvalue of type 'int'
  102 |         return y;
      |                ^
<source>:104:16: error: cannot bind non-const lvalue reference of type 'int&' to an rvalue of type 'int'
  104 |         return x;
      |                ^
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Better Implicit Move Returns

```
 99    auto f() {
100        std::string s = "Alex";
101        return (s);
102    }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Deduction by Inheritance

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Deduction by Inheritance

```
105  template <typename T> struct B {
106      B(T) {};
107  };
108
109  template <typename T> struct C : public B<T> {
110      using B<T>::B;
111  };
112
113  template <typename T> struct D : public B<T> {};
114
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Deduction by Inheritance

```
119    D d(42);
```

Error: Deduction failed no inherited deduction guide

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Deduction by Inheritance

```
116     C c(42);
```

auto deduced to be C<int>
with internal deduction guide

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Alias in for Statements

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Alias in for Statements

```
123  void do_on_map(std::map<std::string, int> m){
124      for (using Ret = std::pair<const std::string, int>; const Ret& p: m){
125          //do stuff with p and use Ret type
126      }
127  }
```

Alex Dathskovsky | alex.dathskovsky@speedata.io | www.linkedin.com/in/alexdathskovsky

Down With ()!

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Down With ()!

```
131     std::map<std::string, int> map{{"1", 1}, {"2", 2}};
132     auto l1 = [&map](){};
133     auto l2 = [&map]{};
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Down With ()!

```
131     std::map<std::string, int> map1{{"1", 1}, {"2", 2}};
132     std::map<std::string, int> map2{{"3", 3}, {"4", 4}};
133     auto l1 = [m1 = std::move(map1)] () mutable {};
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Down With ()!

```
135 |     auto l2 = [m2 = std::move(map2)] mutable {};
```

```
<source>:135:38: error: expected '{' before 'mutable'
  135 |     auto l2 = [m2 = std::move(map2)] mutable {};
      |                                      ^~~~~~~
<source>: In function 'int main()':
<source>:135:38: error: expected ',' or ';' before 'mutable'
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Down With ()!

```
135     auto l2 = [m2 = std::move(map2)] mutable {};
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

if consteval

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

if consteval

```
129  constexpr char get_char(char const* s, size_t i) {
130      if constexpr (std::is_constant_evaluated()) {
131          for (const char *p = s; ; ++p, --i) {
132              if (not i) {
133                  return *p;
134              }
135          }
136      } else {
137          return s[i];
138      }
139  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

if consteval

```
129  constexpr char get_char(char const* s, size_t i) {
130      if consteval {
131          for (const char *p = s; ; ++p, --i) {
132              if (not i) {
133                  return *p;
134              }
135          }
136      } else {
137          return s[i];
138      }
139  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

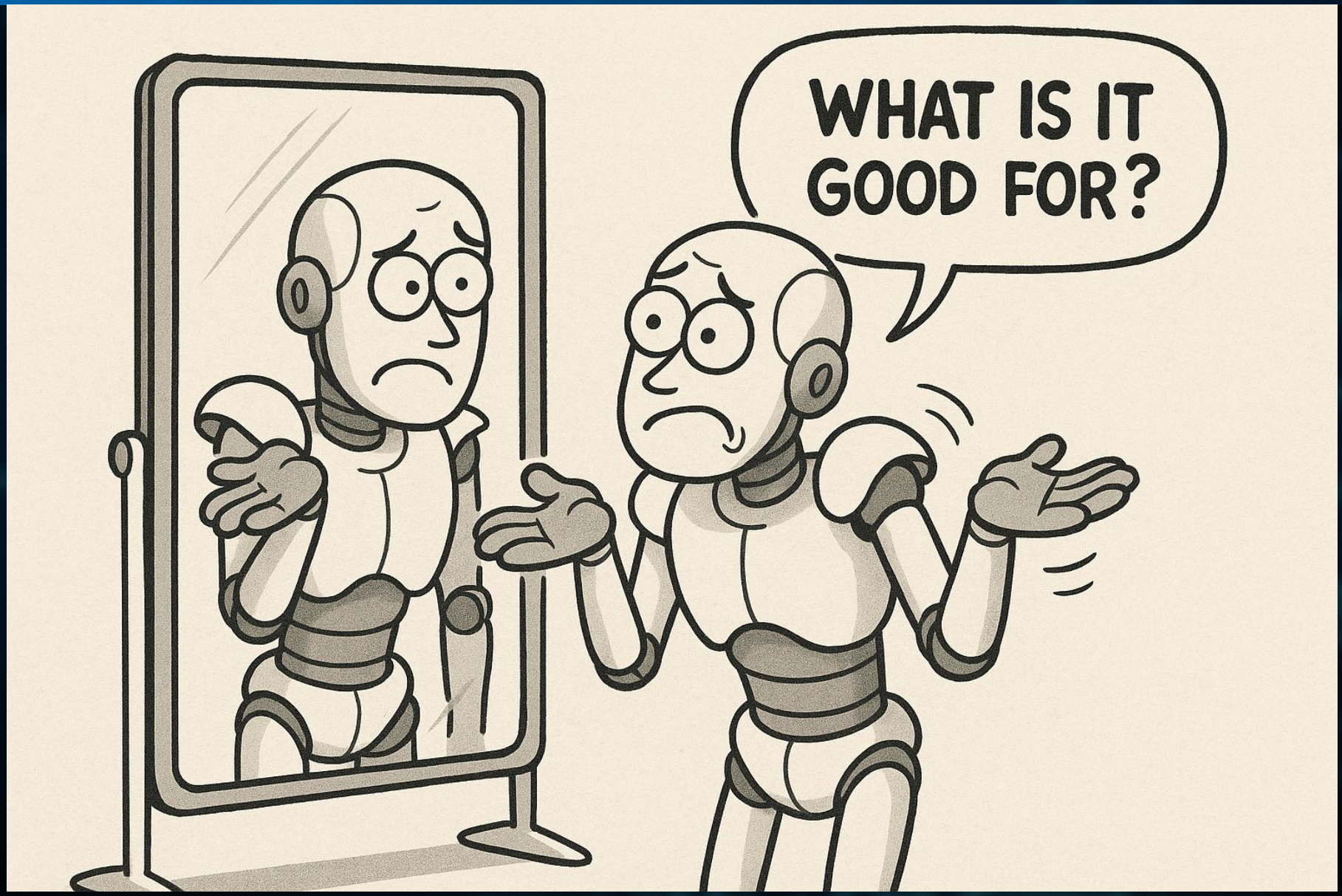
Self Deducing `this`

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`

```
142  struct self_deduce_example1{
143      void f(this self_deduce_example1& self, int i) {
144          self.i_ = i;
145      }
146
147      int i_;
148  };
```

Self Deducing `this`



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: Boilerplate Code Reduction

```
150  template <typename T>
151  struct value{
152      private:
153          bool has_value{false};
154          T val;
155      public:
156          T& get() & { return has_value ? val :
157          throw std::runtime_error{"no val"}; }
158          const T& get() const& { return has_value ? val :
159          throw std::runtime_error{"no val"}; }
160          T&& get() && { return has_value ? std::move(val) :
161          throw std::runtime_error{"no val"}; }
162          const T&& get() const&& { return has_value ?
163          std::move(val) :
164          throw std::runtime_error{"no val"}; }
165  };
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: Boilerplate Code Reduction

```
150   template <typename T>
151   struct value{
152       private:
153           bool has_value{false};
154           T val;
155       public:
156           template <typename Self>
157           auto&& get(this Self&& self) {
158               return self.has_value ?
159               std::forward<Self>(self).val :
160               throw std::runtime_error{"no value"};
161           }
162   };
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: CRTP

```
164  template <typename Thing>
165  struct NotSame{
166      bool operator!=(const Thing& o) const {
167          auto const& self = *static_cast<Thing const*>(this);
168          return (self < o) or (o < self);
169      }
170  };
171
172  struct Dumbell : public NotSame<Dumbell> {
173      bool operator<(const Dumbell& o) const{
174          return this->w < o.w;
175      }
176
177      int w;
178  };
```

Self Deducing `this`: CRTP

```
180  int main(){
181      Dumbell x{.w=10};
182      Dumbell y{.w=10};
183
184      return x != y ? 0 : -1;
185  }
```

Self Deducing `this`: CRTP

```
187  struct NotSame{
188      template <typename Self>
189      bool operator!=(this Self&& self, Self&& o){
190          return (self < o) or (o < self);
191      }
192  };
193
194  struct Dumbell : public NotSame {
195      bool operator<(const Dumbell& o) const{
196          return this->w < o.w;
197      }
198
199      int w;
200  };
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: Recursive Lambdas

```
204     auto fib = [](this auto&& self, int n) {
205         return (n <= 1) ? n : self(n - 1) + self(n - 2);
206     };
207
208     auto fact = [](this auto&& self, int n){
209         return (n <= 1) ? 1 : n * self(n - 1);
210     };
211
```

Self Deducing `this`: Recursive Lambdas

```
219     auto inorder = [](this auto&& self, Node* n){
220         if (not n){
221             return;
222         }
223         self(n->left);
224         fmt::print("{}", n->v);
225         self(n->right);
226     };
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: This by Value

```
struct Small1{
    int x,y;
    int mult(int z);
};

int main(int c, char** v){
    Small1 s1{1, 2};
    return s1.mult(c);
}
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: This by Value

```
2       sub sp, sp, #32
3       stp x29, x30, [sp, #16]
4       add x29, sp, #16
5       mov x8, #1
6       mov w1, w0
7       add x0, sp, #8
8       movk x8, #2, lsl #32
9       str x8, [sp, #8]
10      bl Small1::mult(int)
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: This by Value

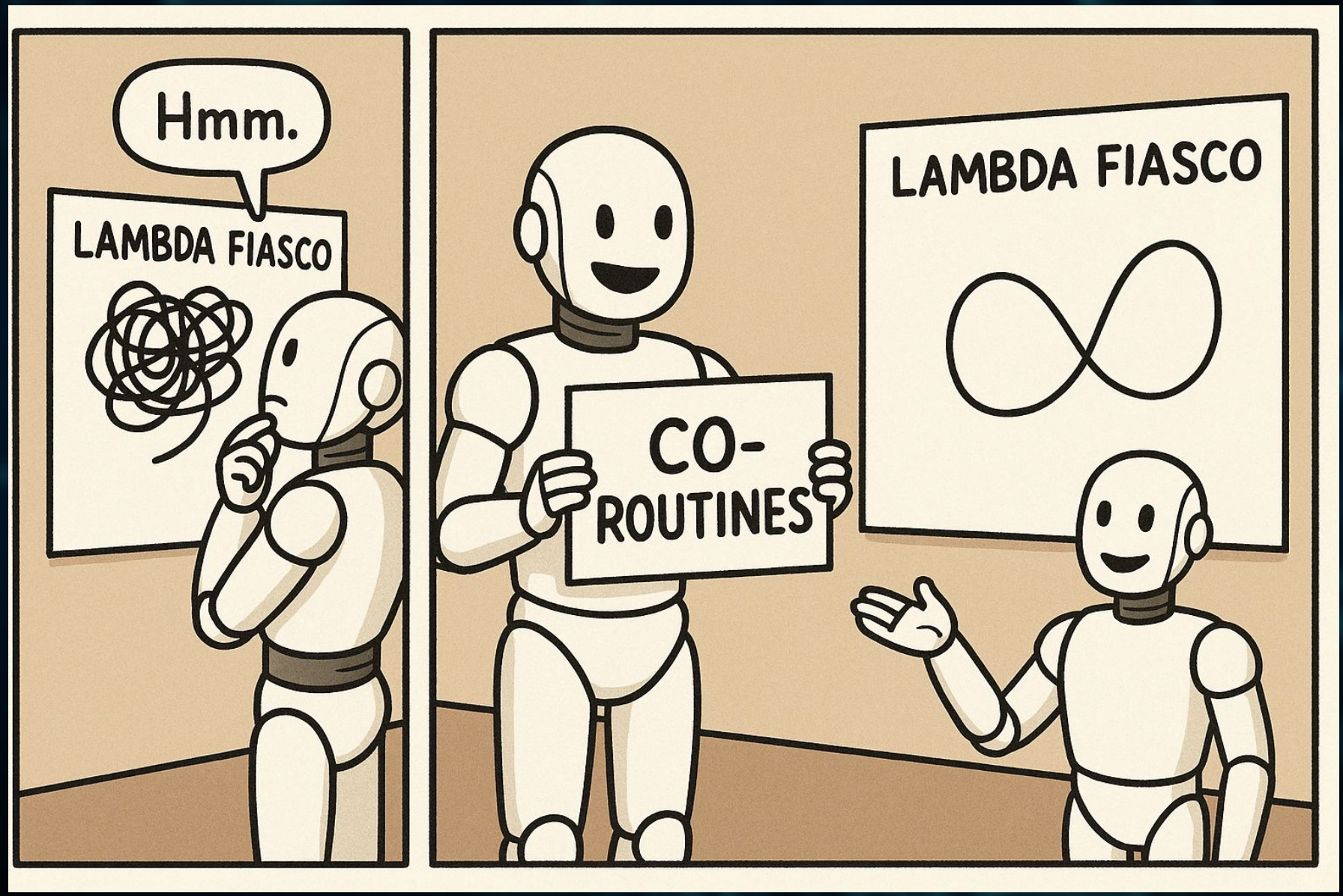
```
struct Small2{
    int x,y;
    int mult(this Small2 self, int z);
};

int main(int c, char** v){
    Small2 s2{1, 2};
    return s2.mult(c);
}
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: This by Value

```
2       mov w1, w0
3       mov x0, #1
4       movk x0, #2, lsl #32
5       b Small2::mult(this Small2, int)
```

Self Deducing `this`: Fixing a Coroutine Fiasco



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: Fixing a Coroutine Fiasco

```
26 corutine lmbda::operator()() const;
27 //          | |
28 //          | |
29 //          \ /
30 //           v
31 corutine lambda_call_operator(const lambda& lmbda);
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: Fixing a Coroutine Fiasco

```
55  coroutine do_things() {
56      return [x = std::make_unique<int>(42)] () -> coroutine {
57          co_await do_things_with_me();
58          co_return *x + 1;
59      }();
60  }
```

Program terminated with signal: SIGSEGV

Self Deducing `this`: Fixing a Coroutine Fiasco

```
55  coroutine do_things() {
56      return [x = std::make_unique<int>(42)] (this auto self) -> coroutine {
57          co_await do_things_with_me();
58          co_return *x + 1;
59      }();
60  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: Fixing a Coroutine Fiasco

```
26 corutine lmbda::operator(this auto self)() const;
27 //           | |
28 //           | |
29 //           \ /
30 //            v
31 corutine lambda_call_operator(this auto self);
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Self Deducing `this`: Amazing

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Library Features

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::forward_like

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::forward_like

```
34 auto callback = [m=get_message(), &scheduler](this auto &&self) -> bool {
35 return scheduler.submit(std::forward_like<decltype(self)>(m));
36 };
37 callback(); // retry(callback)
38 std::move(callback)(); // try-or-fail(rvalue)
```

https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2022/p2445r1.pdf

std::generator

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::generator

```
15  int main(){
16      for (auto [i, v] : enumarate(std::array{1, 2, 3, 4})){
17          std::println("the value in idx {} is {}", i, v);
18      }
19  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::generator Pre C++23

```
247  template <class T>
248  class generator {
249  public:
250      struct promise_type {
251          std::optional<T> current;
252
253          generator get_return_object() noexcept {
254              return generator{std::coroutine_handle<promise_type>::from_promise(*this)};
255          }
256          std::suspend_always initial_suspend() const noexcept { return {}; }
257          std::suspend_always final_suspend() const noexcept { return {}; }
258          std::suspend_always yield_value(T value) noexcept {
259              current = std::move(value);
260              return {};
261          }
262          void unhandled_exception() { throw; }
263          void return_void() noexcept {}
264      };
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::generator

```
266     using handle_type = std::coroutine_handle<promise_type>;
267
268     generator() noexcept : h_(nullptr) {}
269     explicit generator(handle_type h) : h_(h) {}
270     generator(const generator&) = delete;
271     generator(generator&& other) noexcept : h_(std::exchange(other.h_, {})) {}
272     generator& operator=(generator&& other) noexcept {
273         if (this != &other) {
274             if (h_) h_.destroy();
275             h_ = std::exchange(other.h_, {});
276         }
277         return *this;
278     }
279     ~generator() {
280         if (h_) h_.destroy();
281     }
```

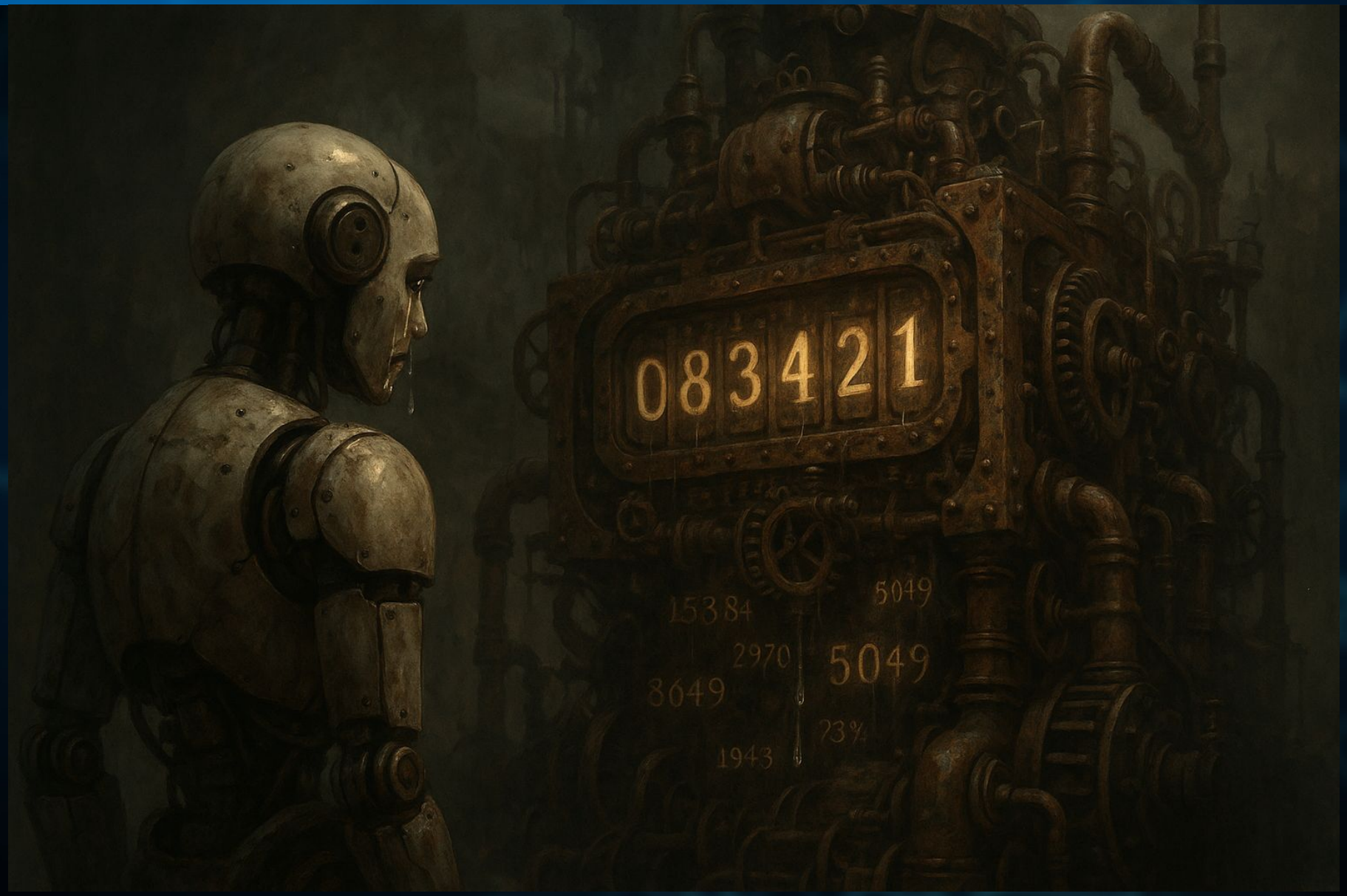
std::generator

```
283     class iterator {
284     public:
285         using value_type = T;
286         using difference_type = std::ptrdiff_t;
287
288         iterator() noexcept : h_(nullptr) {}
289         explicit iterator(handle_type h) : h_(h) { advance(); }
290
291         iterator& operator++() { advance(); return *this; }
292         const T& operator*() const noexcept { return *h_.promise().current; }
293         const T* operator->() const noexcept { return &*h_.promise().current; }
294
295         friend bool operator==(const iterator& it, std::default_sentinel_t) noexcept {
296             return !it.h_ || it.h_.done();
297         }
298         friend bool operator!=(const iterator& it, std::default_sentinel_t s) noexcept {
299             return !(it == s);
300         }
301
```

std::generator

```
302         private:
303             void advance() {
304                 if (!h_ || h_.done()) return;
305                 h_.resume();
306                 if (h_.done()) h_ = nullptr;
307             }
308             handle_type h_;
309         };
310
311         iterator begin() { return iterator{h_}; }
312         std::default_sentinel_t end() const noexcept { return {}; }
313
314     private:
315         handle_type h_;
316     };
```

std::generator



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::generator C++23

```
 7   std::generator<const std::tuple<int, int>>
 8   enumarate(auto container){
 9       int idx = 0;
10       for (auto v : container){
11           co_yield std::tuple{idx++, v};
12       }
13   }
14
15   int main(){
16       for (auto [i, v] : enumarate(std::array{1, 2, 3, 4})){
17           std::println("the value in idx {} is {}", i, v);
18       }
19   }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::print

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::print

```
std::print("Hello World! {}\n", 1);
std::println("Hello World! {}", 2);
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Better Error Handling

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::expected

```
329  enum class error_val{
330      function_not_invoked = 0,
331      other_error
332  };
333
334  std::expected<int, error_val>
335  process_error(error_val e){
336      std::println("Error: {}",
337      e == error_val::function_not_invoked ?
338      "Function Not Invoked" : "Other");
339      return std::unexpected(e);
340  }
341
342  std::expected<int, error_val>
343  process_value(int val){
344      std::println("got value {}", val);
345      return 0;
346  }
```

std::expected

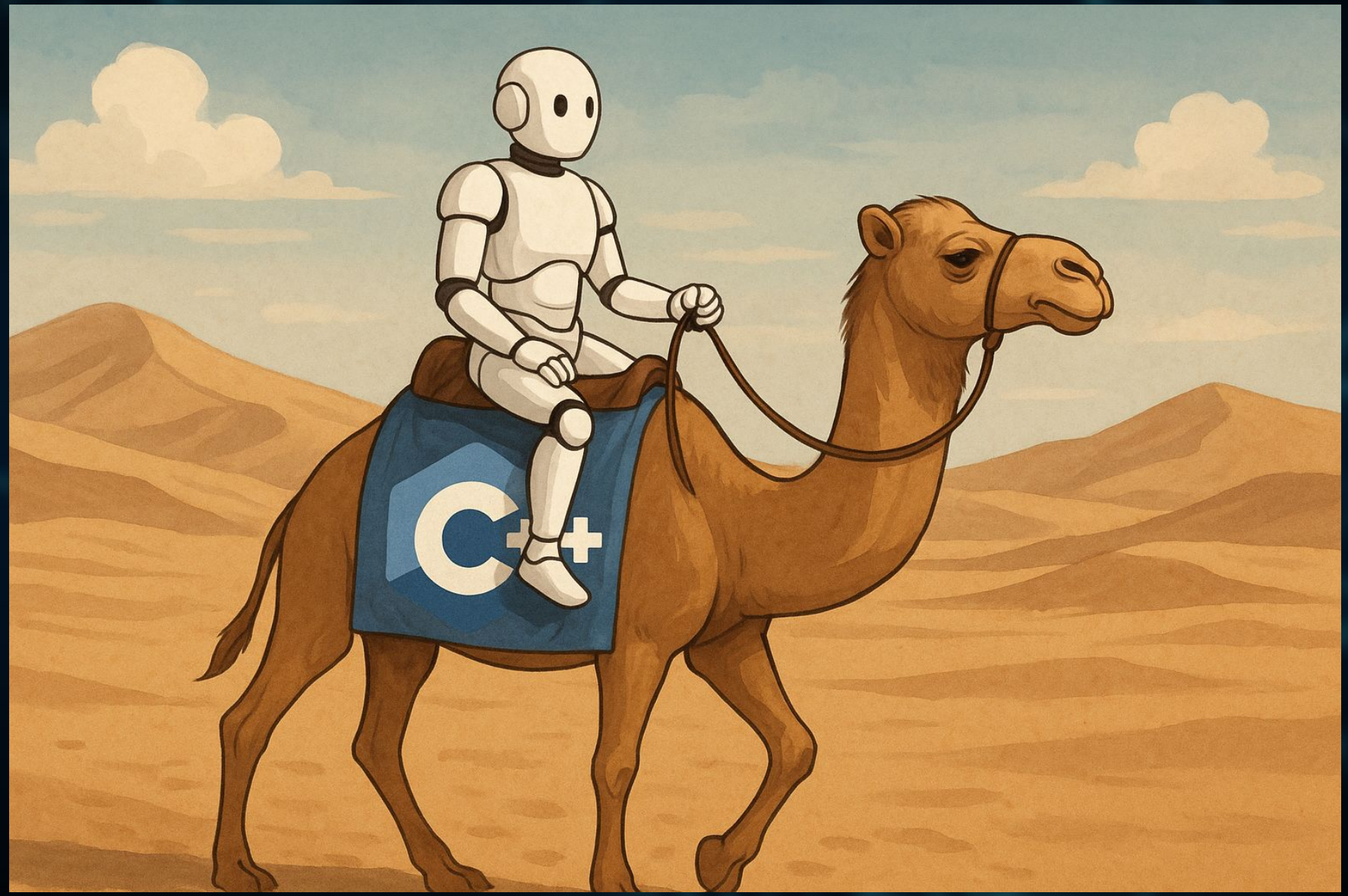
```
348  constexpr std::expected<int, error_val>
349  invoke_return(auto&& f, auto&&... params){
350      if constexpr (std::invocable<decltype(f),
351      decltype(params)...>) {
352          return f(std::forward<decltype(params)>(params)...);
353      } else {
354          return std::unexpected(
355              error_val::function_not_invoked);
356      }
357  }
```

std::expected

```
359   int f1(int a, int b) {return a+b;}
360
361   int main(int c, char** v){
362       invoke_return(f1, 1, 2).and_then(process_value);
363       invoke_return(f1, "1", 2).and_then(process_value).or_else(process_error);
364   }
```

```
got value 3
Error: Function Not Invoked
```

More Monads Please



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::optional is Monadic as Well

```
331  std::optional<int>
332  process_error(){
333      std::println("Error: Function Not Invoked");
334      return {};
335  }
336
337  std::optional<int>
338  process_value(int val){
339      std::println("got value {}", val);
340      return val;
341  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::optional is Monadic as Well

```
343  constexpr std::optional<int>
344  invoke_return(auto&& f, auto&&... params){
345      if constexpr (std::invocable<decltype(f),
346      decltype(params)...>) {
347          return f(std::forward<decltype(params)>(params)...);
348      } else {
349          return {};
350      }
351  }
352
353  int f1(int a, int b) {return a+b;}
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::optional is Monadic as Well

```
343  constexpr std::optional<int>
344  invoke_return(auto&& f, auto&&... params){
345      if constexpr (std::invocable<decltype(f),
346      decltype(params)...>) {
347          return f(std::forward<decltype(params)>(params)...);
348      } else {
349          return {};
350      }
351  }
352
353  int f1(int a, int b) {return a+b;}
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::optional is Monadic as Well

```
359   int f1(int a, int b) {return a+b;}
360
361   int main(int c, char** v){
362       invoke_return(f1, 1, 2).and_then(process_value);
363       invoke_return(f1, "1", 2).and_then(process_value).or_else(process_error);
364   }
```

```
got value 3
Error: Function Not Invoked
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Detective Mode

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::stacktrace

```
365  void inner_function() {
366      auto trace = std::stacktrace::current();
367
368      //print rrace automatically
369      std::println("=== Stacktrace (inside inner_function) ===\n{}", trace)
370
371      // Optionally, inspect frames manually:
372      for (std::size_t i = 0; i < trace.size(); ++i) {
373          const auto& frame = trace[i];
374          std::println("#{}:{} @ {}:{}",i, frame.description(),
375                                  frame.source_file(), frame.source_line());
376      }
377  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::stacktrace

```
=== Stacktrace (inside inner_function) ===
   0# inner_function() at /app/example.cpp:366
   1# outer_function() at /app/example.cpp:380
   2#       at :0
   3# __libc_start_main at :0
   4# _start at :0
   5#

#0:inner_function() @ /app/example.cpp:366
#1:outer_function() @ /app/example.cpp:380
#2: @ :0
#3:__libc_start_main @ :0
#4:_start @ :0
#5: @ :0
```

std::stacktrace

```
387   struct traced_error : std::runtime_error {
```

CppCon 2025

Erez Strauss

Eisler Captal

Strat - Sr Software Engineer

erezstrauss.com

your phone or calendar.

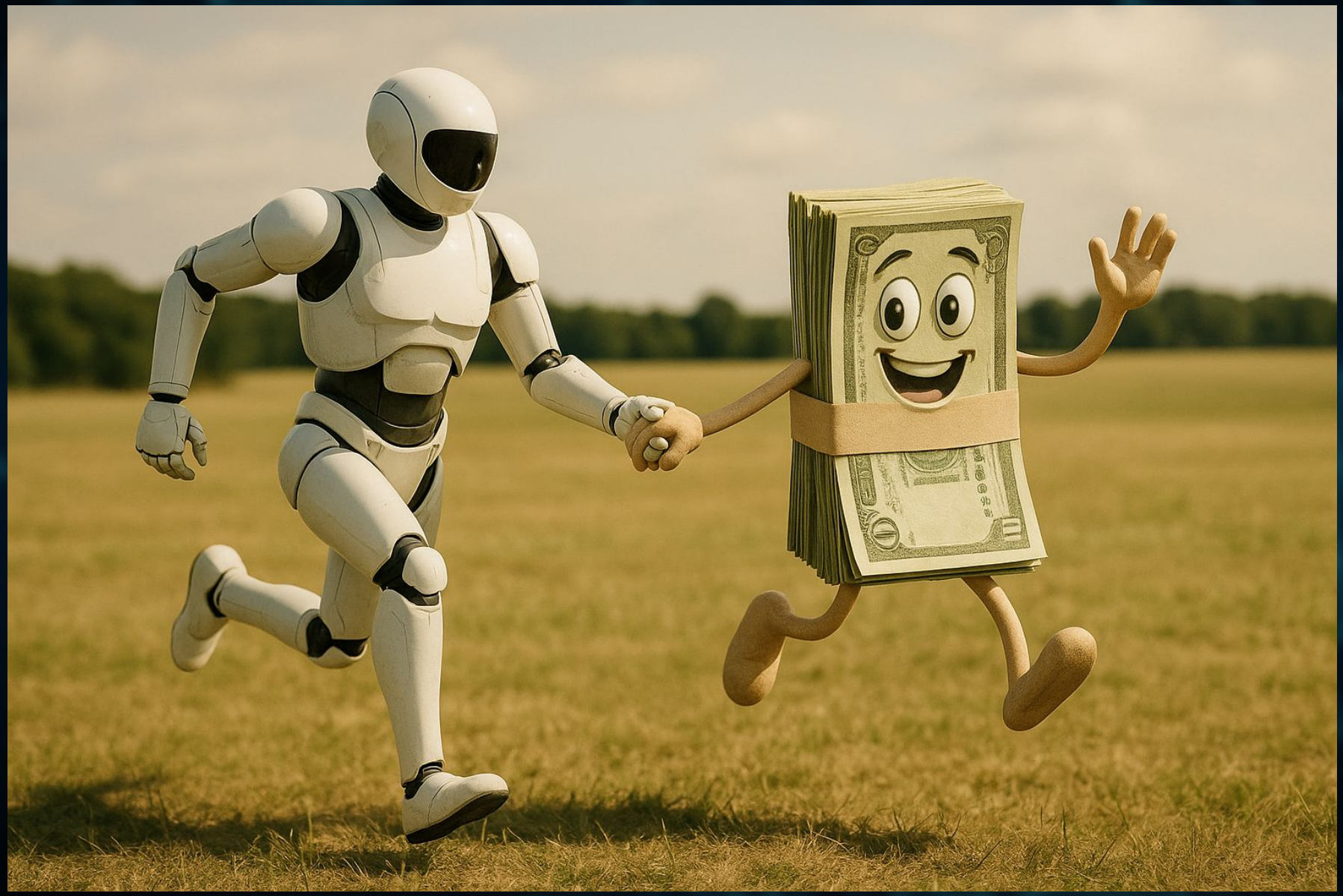
dees Map Search

Enhancing Exception Handling and Debugging using C++23 std::stacktrace

```
400           f();
401       } catch (const traced_error& e) {
402           std::println("Exception:\n{} Stacktrace:\n{}",
403           e.what(), e.trace);
404       }
405   }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Cache Friendliness

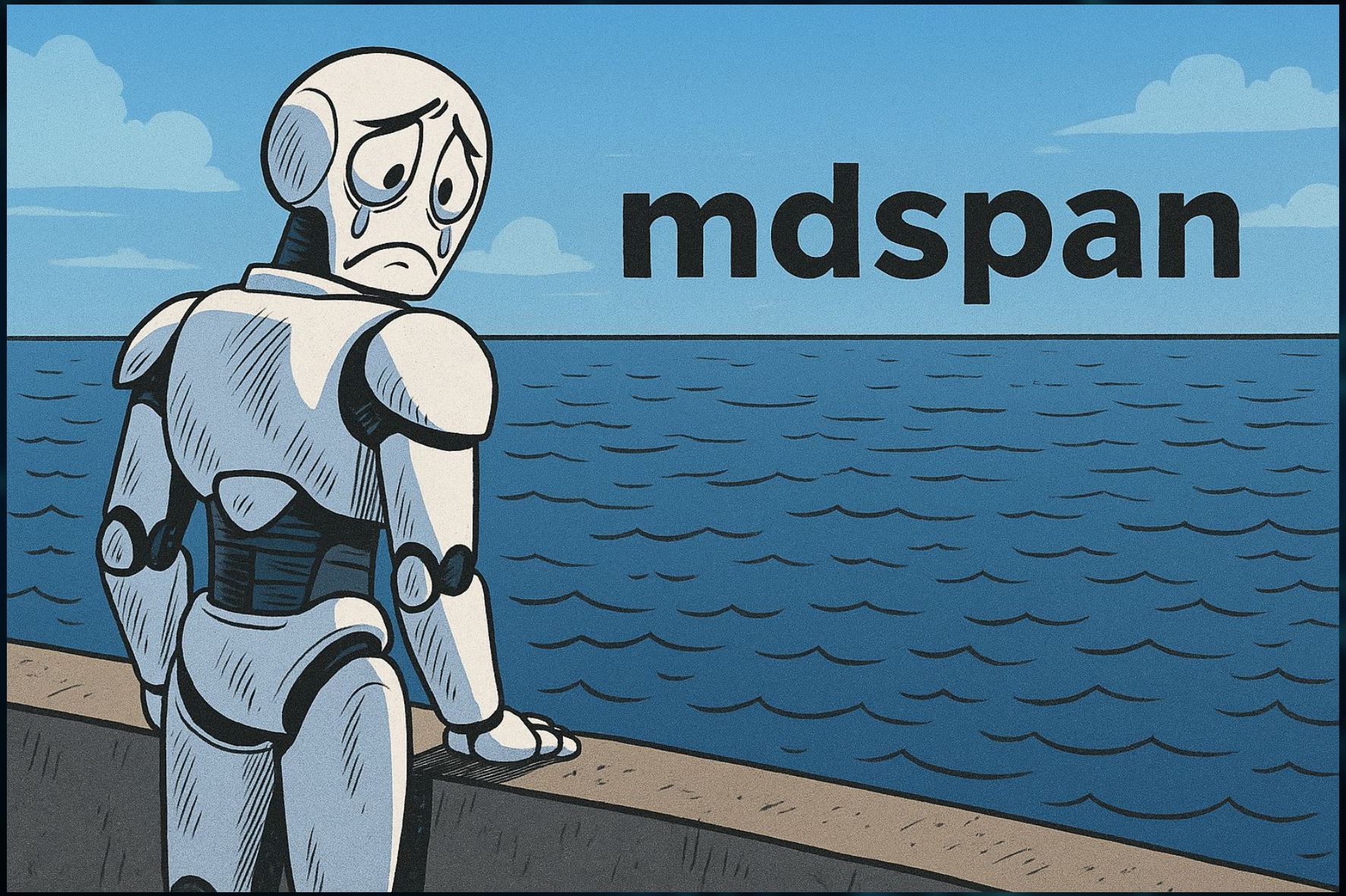


Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

std::flat_set std::flat_map

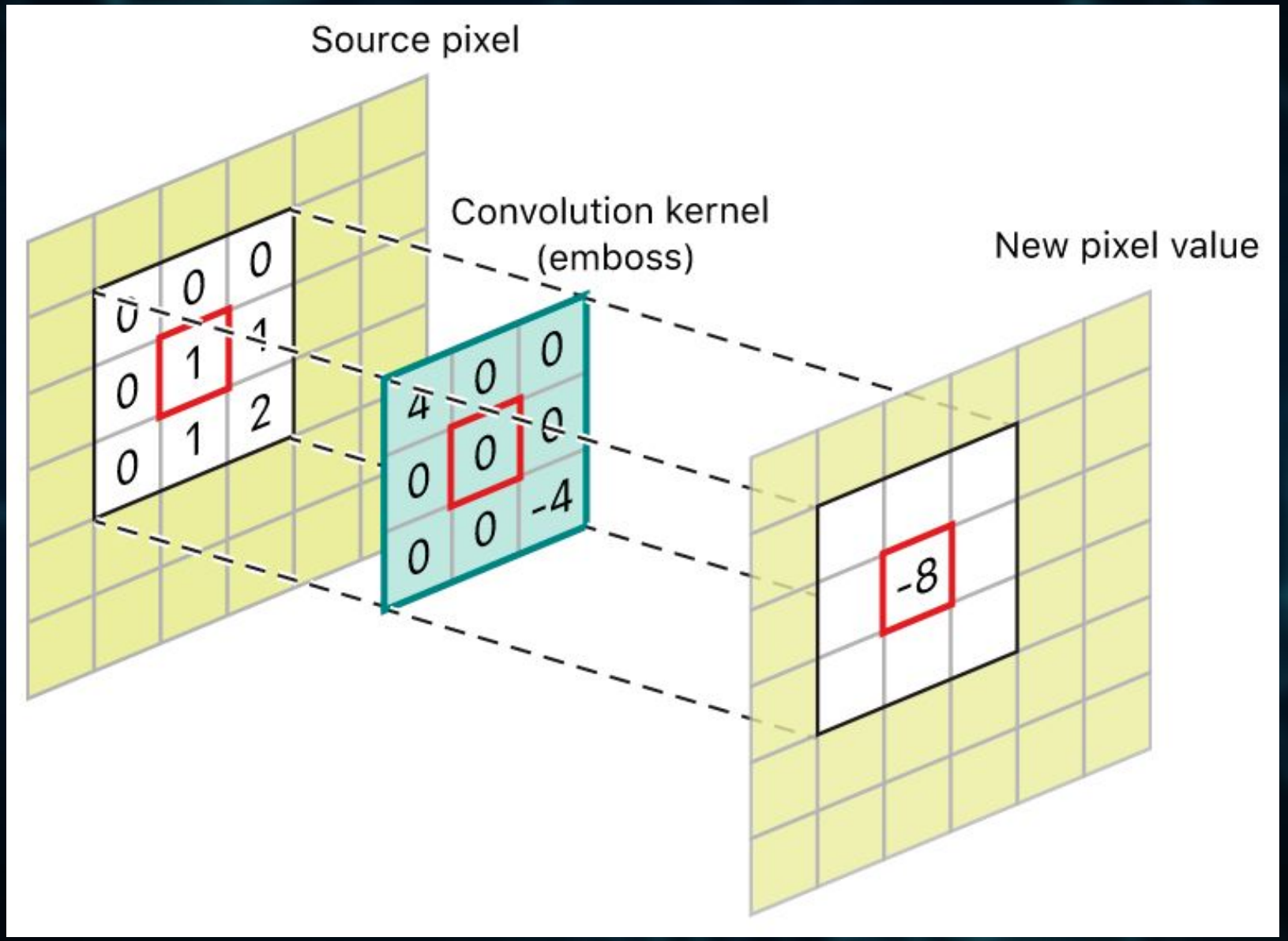
```
410  int main() {
411      std::vector<int> values = {5, 3, 1, 3, 7, 5, 9, 1};
412      std::flat_set<int> unique(values.begin(), values.end());
413      std::println("{}", unique);
414      return 0;
415  }
```

mdspan



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Convolution



Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Convolution

```
2   #include <https://raw.githubusercontent.com/kokkos/mdspan/single-header/mdspan.hpp>
3
4   namespace stdex = std::experimental;
5   static constexpr std::size_t image_w = 6, image_h = 6, padded_w = 7, padded_h = 7;
6   static constexpr std::size_t filter_w = 3, filter_h = 3;
7   using Image_Span_Padded = stdex::mdspan<float, stdex::extents<size_t, padded_w, padded_h>>;
8   using Image_Span = stdex::mdspan<float, stdex::extents<size_t, image_w, image_h>>;
9   using Kernel_Span = stdex::mdspan<float, stdex::extents<size_t, filter_w, filter_h>>;
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Convolution

```
11    void do_on_span(auto span, auto func)
12    requires (span.rank() == 2) {
13        for (std::size_t i = 0; i < span.extent(0); ++i){
14            for (std::size_t j = 0; j < span.extent(1); ++j){
15                func(span[i, j], i, j);
16            }
17        }
18    }
```

Convolution

```
39  void convolution(Image_Span_Padded im, Kernel_Span k, Image_Span o){
40      do_on_span(o, [&im, &k](auto& d, auto i, auto j){
41          d = conv_impl(stdex::submdspan(im, std::pair{i, i+3}, std::pair{j, j+3}), k);});
42  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Convolution

```
29  float conv_impl(auto im_ker, auto ker){
30      auto ret = 0;
31      for (std::size_t i = 0; i < im_ker.extent(0); ++i){
32          for (std::size_t j = 0; j < im_ker.extent(1); ++j){
33              ret += im_ker[i, j]*ker[i, j];
34          }
35      }
36      return ret;
37  }
```

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

Conclusion

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

QUESTIONS

Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

THANK YOU FOR LISTENING