Unlocking the Value of C++20

Why is it Still Relevant?

	Yes: Pretty much all	**Partial**: Limited features/usage	**No**: Not allowed
C++98/03 (e.g., exceptions, templates, RTTI)	84.7%	8.6%	6.7%
C++11 (e.g., auto, move semantics, =delete/=default,…)	91.3%	5.8%	2.9%
C++14 (e.g., generic lambdas, auto return types, general…	87.1%	9.2%	3.7%
C++17 (e.g., if constexpr, if/switch scoped variables,…	81.2%	13.3%	5.5%
C++20 (e.g., concepts, coroutines, modules)	43.2%	32.2%	24.6%
C++23 (e.g., expected, mdspan)	26.9%	22.5%	50.6%

About Me:

alex.datskovsky@scylladb.com

www.linkedin.com/in/alexdathskovsky

www.cppnext.com

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

What to Expect

```
11      ([](const auto&... args){
12          auto count = sizeof...(args);
13          ([&count](const auto& arg){
14              std::cout << arg <<(--count ? ", " : "\n");
15              return std::ignore;
16          }(args) = ...);
17      })(1, 2, 'a', "Hello");
18
```

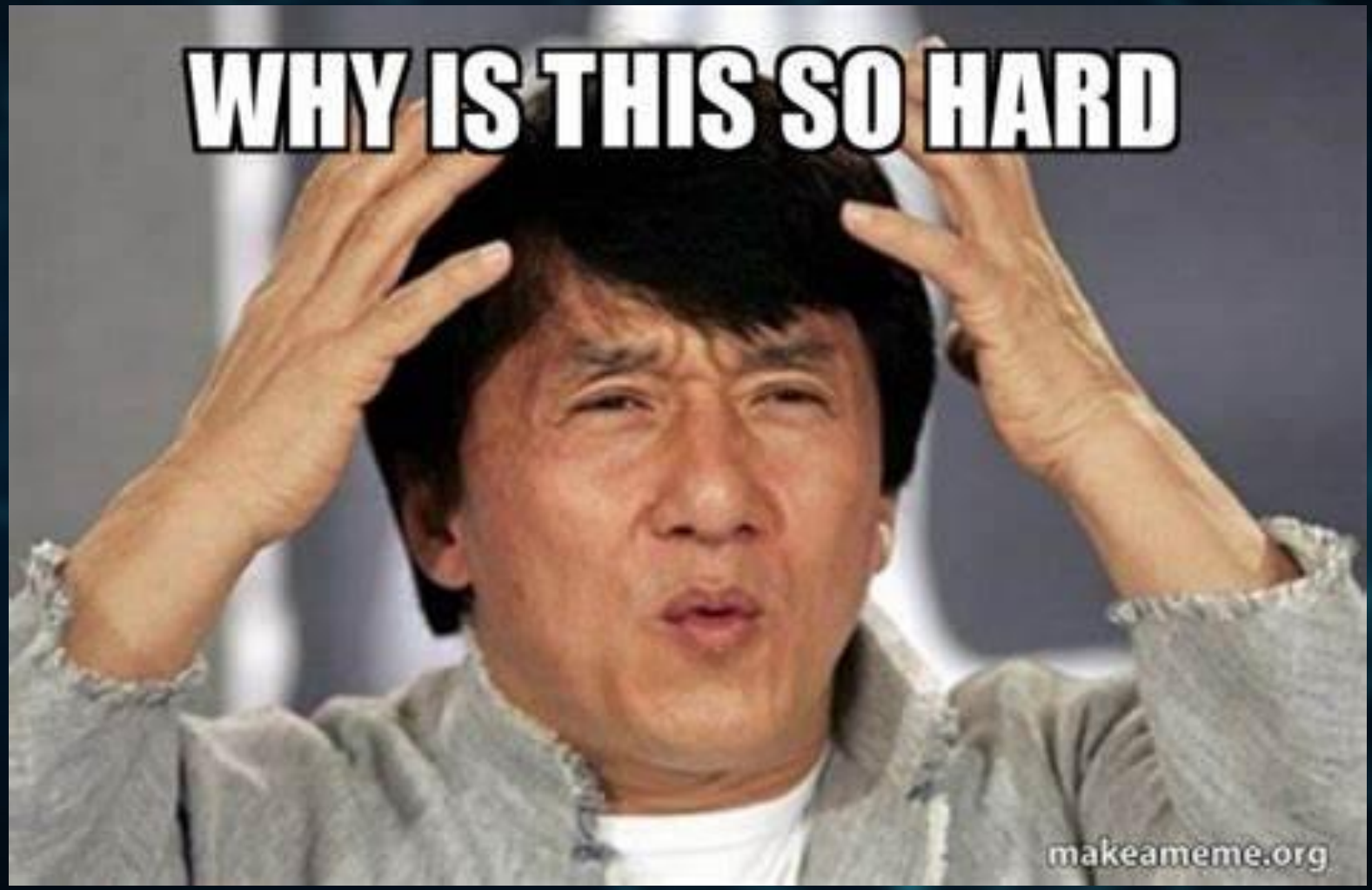
Alex Dathskovsky | alex.dathskovsky@scylladb.com | www.linkedin.com/in/alexdathskovsky

What to Expect

```
arc_unsigned(unsigned long):
        test    rdi, rdi
        je      .LBB1_1
        inc     rdi
        cmp     rdi, 3
        mov     ecx, 2
        cmovae  rcx, rdi
        lea     rax, [rcx - 2]
        lea     rdx, [rcx - 3]
        mul     rdx
        shld    rdx, rax, 63
        lea     rax, [rdx + 2*rcx]
        add     rax, -3
        ret
.LBB1_1:
        xor     eax, eax
        ret
```

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

Shoutout

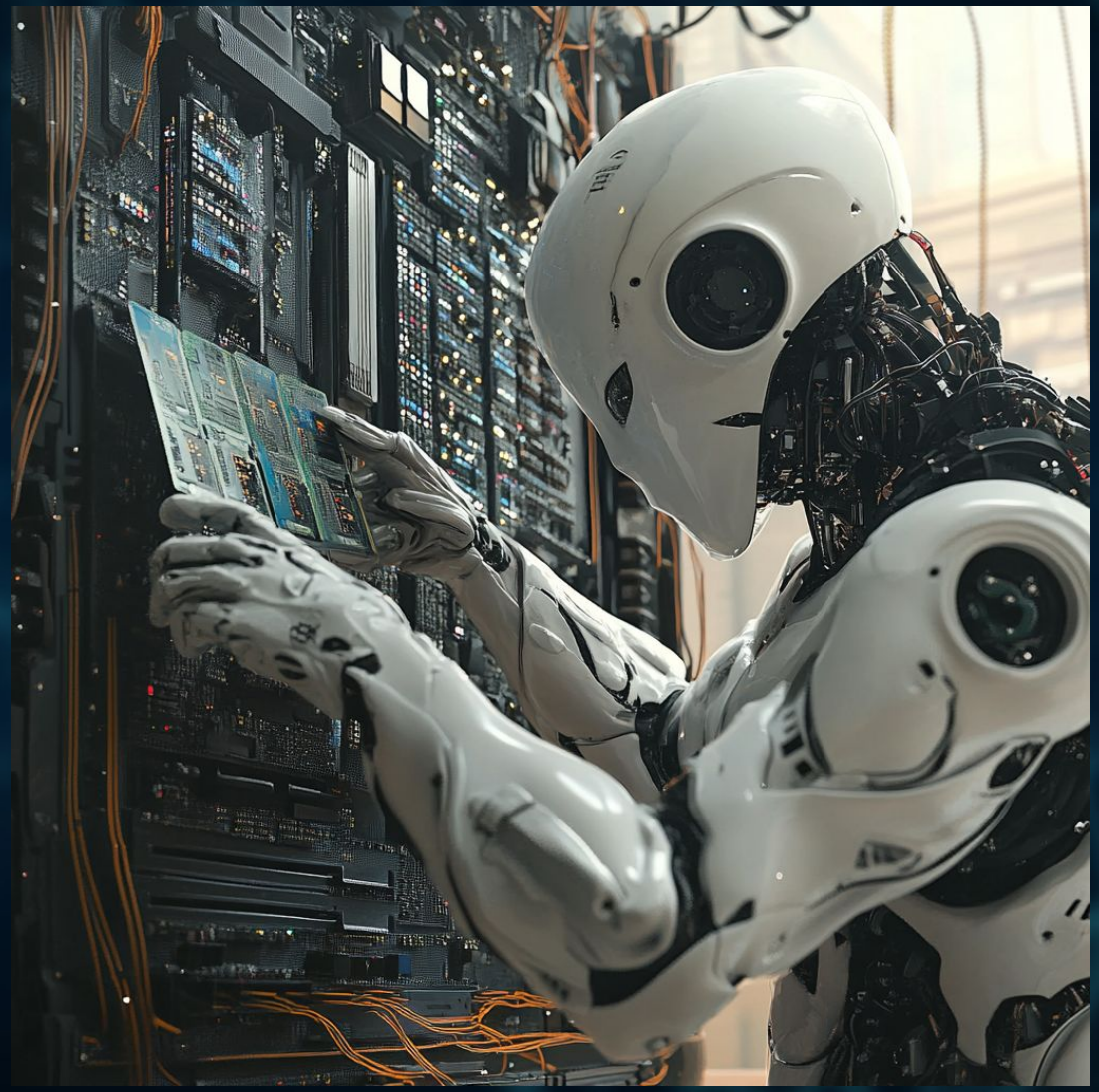
How to understand modern C++ features in practice? Let's create a compiler!

Speaker: Boguslaw Cyganek

Audience level: [Beginner | Intermediate | Advanced]

Recent C++ standards have introduced many new features. We often hear about them, but it would be interesting to test them in practice. Nothing could be simpler! We need to make a relatively simple project in which we will use many of them. And so the implementation of the FORTH compiler-interpreter was created. You may ask – what is that? I've never heard of it. Don't worry, after this talk you will know more about this language and about few cool features of modern C++.

Let's Start by Getting This Out of the System



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

Not About the Big 4 (But Let's Mention Them)

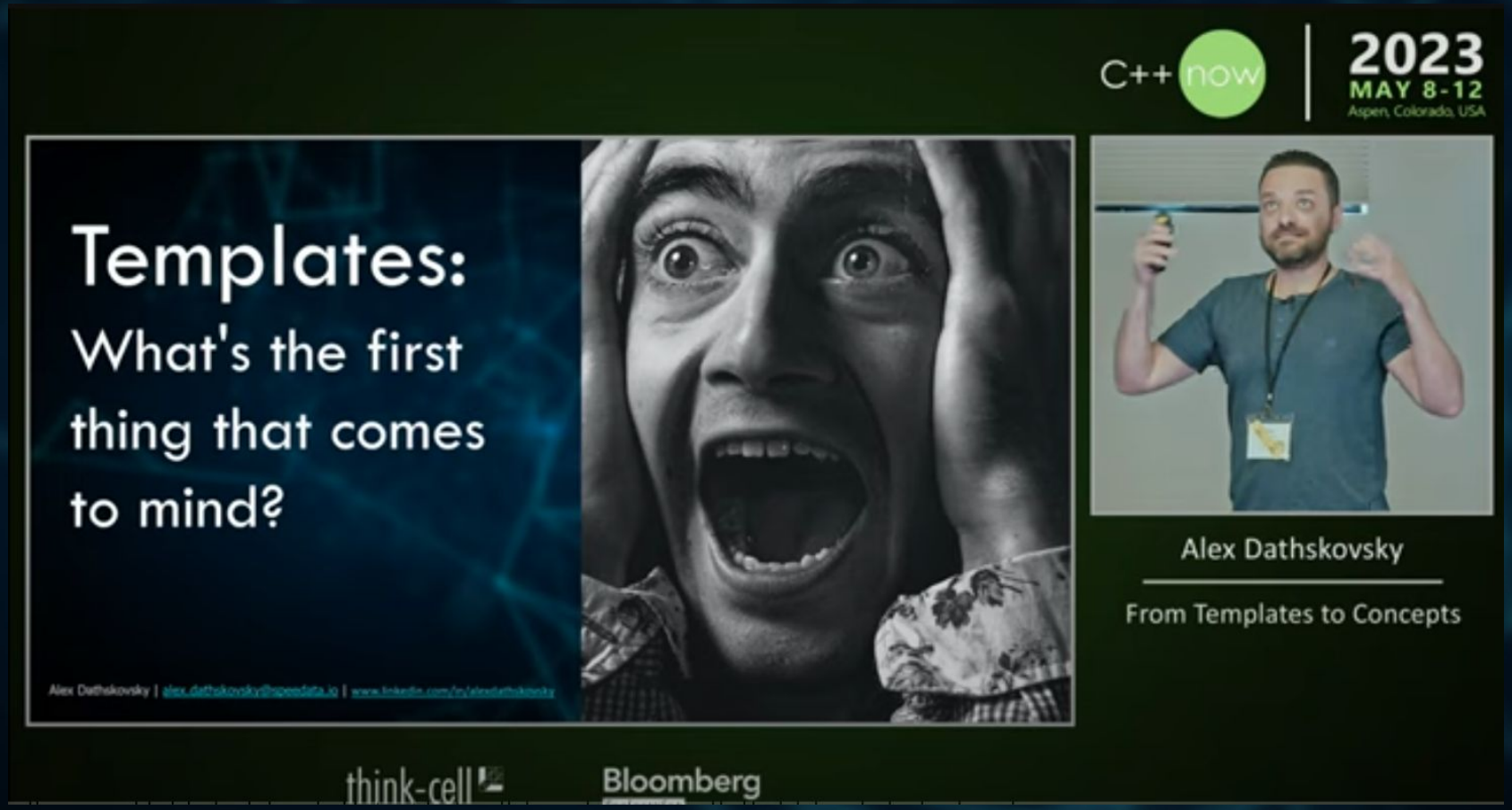
Concepts:

```
170   template <typename T>
171   requires requires { requires std::is_pointer_v<T>;
172   requires requires { requires std::convertible_to<T, int>;
173   requires requires { requires sizeof(T) > sizeof(int)
174   ;} ;} ;}
175   struct Smile{};
```

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

Not About the Big 4 (But Let's Mention Them)

Concepts:



https://youtu.be/x6_o-jz_Q-8?si=8fA47fFRy2e4WL-e

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

Not About the Big 4 (But Let's Mention Them)

Ranges:

```
 9    std::vector<uint64_t> iota(30);
10
11    auto odd = [](uint64_t x){return x%2 != 0;};
12
13    std::iota(iota.begin(), iota.end(), 1);
14
```

```
[19, 17, 15, 13, 11, 9, 7, 5, 3, 1]
```

```
17
18    std::copy_if(iota.begin(), iota.end(), temp.begin(), odd);
19
20    std::vector<uint64_t> temp2(temp.begin(), temp.begin()+10);
21
22    std::reverse(temp2.begin(), temp2.end());
23
24    fmt::print("{}", temp2);
```

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

Not About the Big 4 (But Let's Mention Them)

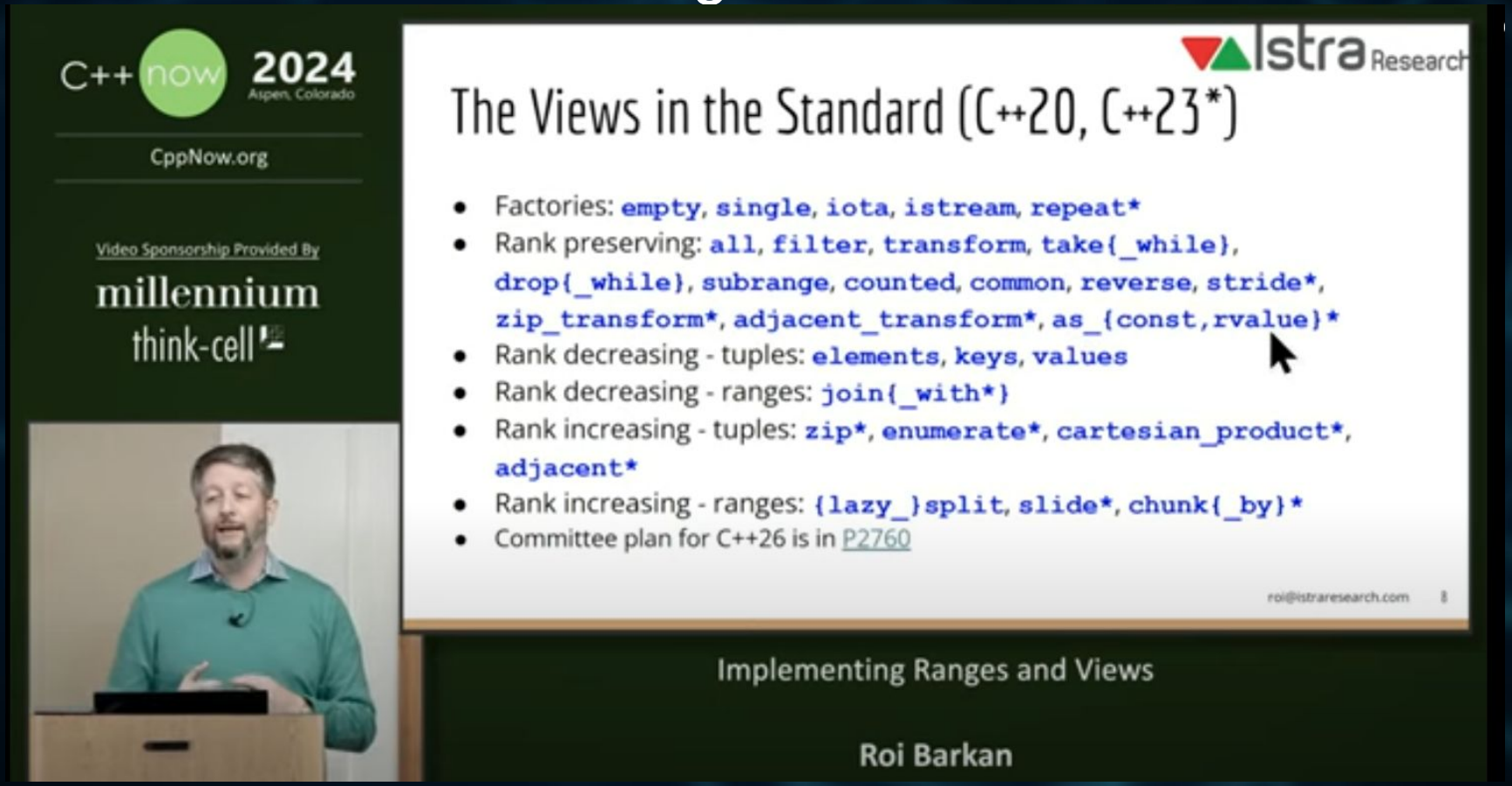
Ranges:

```
26      auto odd = [](uint64_t x){return x%2 != 0;};
27      auto v = std::views::iota(1, 30) | std::views::filter(odd)
28 [19, 17, 15, 13, 11, 9, 7, 5, 3, 1]
29
30      fmt::print("{}", v);
```

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

Not About the Big 4 (But Let's Mention Them)

Ranges:



https://youtu.be/ngaty13aE9M?si=T3OxpnoqFjDHR_mg

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

Not About the Big 4 (But Let's Mention Them)

Modules:

```
export module mymodule; // module declaration

import <iostream>;      // import declaration

export void hello()     // export declaration
{
    fmt::print("Hello my name is Alex!");
}
```

Not About the Big 4 (But Let's Mention Them)

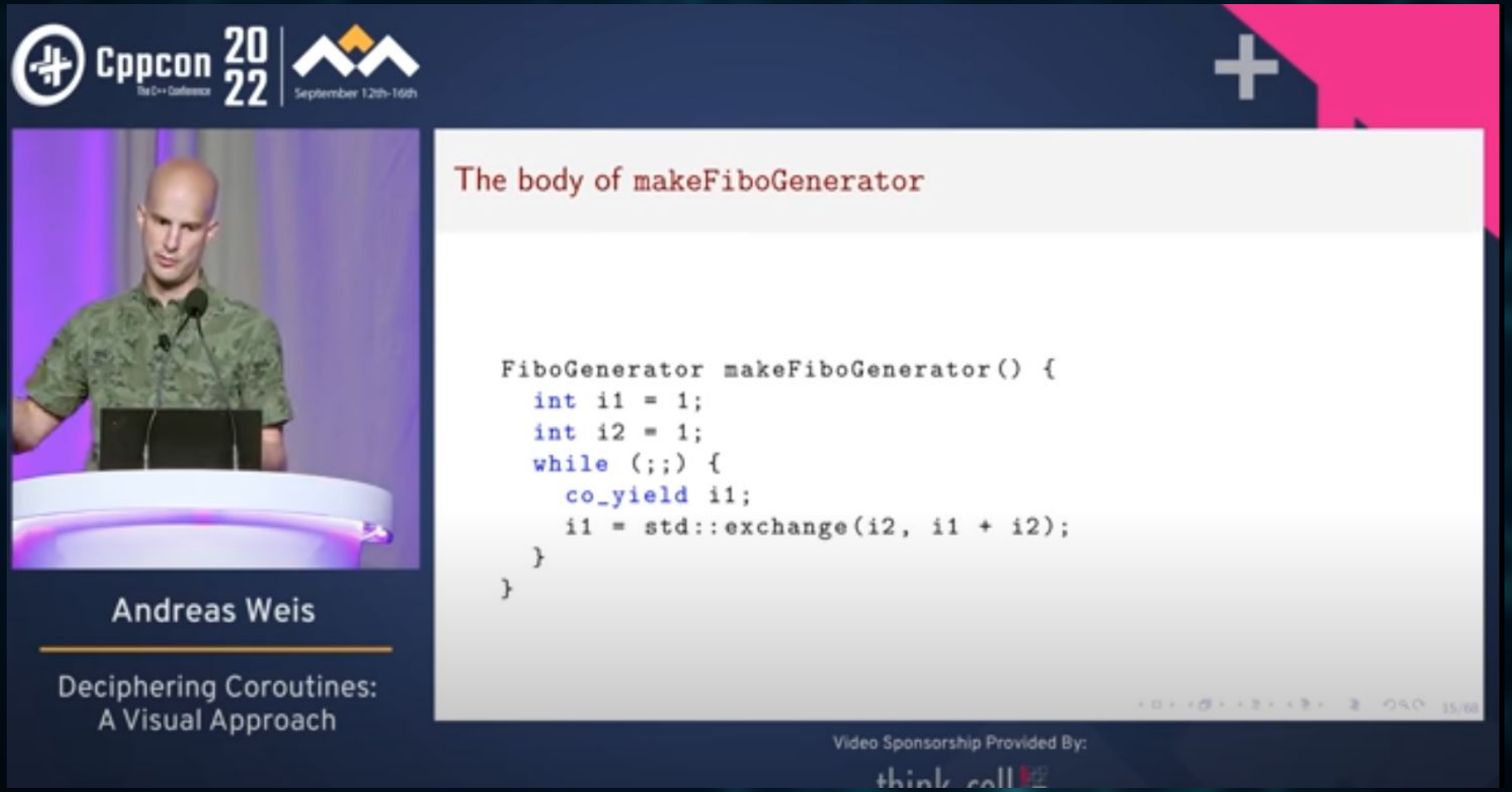
Modules:

https://youtu.be/IwuBZpLUq8Q?si=LLaAlfPQsS26f5pD

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

Not About the Big 4 (But Let's Mention Them)

Coroutines:



https://youtu.be/J7fYddslH0Q?si=diBr43nbXdRlOvkH

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

Language Features

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

Abbreviated Function Templates

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

Abbreviated Function Templates

```
 9 template <typename T, typename U>
10 void dothings(const T& t, U u){
11     ...
12 }
13
```

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

Abbreviated Function Templates

```
11 void dothings(const auto& t, auto u){
12     ...
13 }
```

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

Abbreviated Function Templates

```
10    template <typename T>
11    concept I = std::integral<T>;
12
13    auto dothings(I auto t, I auto u)->
14        std::common_type_t<decltype(t), decltype(u)>{
15        return t + u;
16    }
```

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

Abbreviated Function Templates: Homogeneity

```
18    template <typename T>
19    T dothings(T a, T b){
20        return a + b;
21    }
```

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

Abbreviated Function Templates: Homogeneity

```
23  auto dothings(auto a, decltype(a) b){
24      return a + b;
25  }
```

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

Abbreviated Function Templates: Homogeneity

```
34  auto dothings(auto a, std::decay_t<decltype(a)> b){
35      return a + b;
36  }
```

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

What is the Problem With Decay

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

std::decay pitfaults

1. Decay will decay arrays to pointers.
2. Decay will decay any function like object to a function pointers

std::decay pitfaults

```
42    int a[] = {1, 2, 3};
43    static_assert(std::same_as<decltype(a),
44    std::decay_t<decltype(a)>>);
```

```
because 'std::is_same_v<int[3], int *>' evaluated to false
```

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

std::decay pitfaults

```
39    using f = int(int);
40    static_assert(std::same_as<f, std::decay_t<f>>);
```

```
because 'std::is_same_v<int (int), int (*)(int)>' evaluated to false
```

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

std::decay Solution

```
46    int a[] = {1, 2, 3};
47    using A = decltype(a);
48    static_assert(std::same_as<A,
49    std::remove_cv_t<std::remove_reference_t<A>>>);
```

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

std::decay Solution

```
51    int a[] = {1, 2, 3};
52    using A = decltype(a);
53    static_assert(std::same_as<A, std::remove_cvref_t<A>>);
```

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

Back to language Features

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

Designated Initializers

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

Designated Initializers

```
56  struct S{
57      int x;
58      float y;
59      std::vector<int> z;
60  };
61
62  S s1{.x=10, .y=1.1};
63  //S s2{.y=1.1, .x=10}; #error
64  S s3{.x=10, .z{1,2,3}};
```

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

Designated Initializers: class type rules

- class types that has
  - no user-declared constructors (until C++11)
  - no user-provided, inherited, or explicit constructors (since C++11) (until C++20)
  - no user-declared or inherited constructors (since C++20)
- no private or protected direct non-static data members
  - no base classes (until C++17)
  - no virtual base classes
  - no private or protected direct base classes (since C++17)
- no virtual member functions
  - no default member initializers (since C++11) (until C++14)

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

Ranged For Initializers

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

Initializer in Ranged-For

```
68      std::array say_it = {"hello", "world"};
69      uint64_t i=0;
70      for (const auto& v : say_it){
71          fmt::print("word index:{} word:{}", i++, v);
72      }
```

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

Initializer in Ranged-For

```
74      for (std::size_t i = 0; i < say_it.size(); ++i) {
75          fmt::print("word index:{} word:{}", i, say_it[i]);
76      }
```

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

Initializer in Ranged-For

```
78      for (uint64_t i=0; const auto& v : say_it){
79          fmt::print("word index:{} word:{}", i++, v);
80      }
```

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

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

Default Comparison

```
65  struct S{
66      int x,y;
67      float z;
68
69      bool operator==(const S& o) const{
70          return o.x == x and o.y == y and o.z == z;
71      }
72      bool operator<(const S& o) const{
73          return o.x > x or o.y > y or o.z > z;
74      };
75
76      bool operator>(const S& o) const{
77          return o.x < x or o.y < y or o.z < z;
78      };
79
80      //more operators
81  };
```

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

Default Comparison

```
S s1{1, 2, 3.12};
S s2{1, 2, 3.13};
fmt::print("s1==s2: {}\n", s1 == s2);
fmt::print("s1>s2: {}\n", s1 > s2);
fmt::print("s1<s2: {}\n", s1 < s2);
```

```
s1==s2: false
s1>s2: false
s1<s2: true
```

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

Default Comparison

```
83  struct S{
84      int x,y;
85      float z;
86
87      bool operator==(const S& o) const = default;
88      bool operator<(const S& o) {
89          return o.x > x or o.y > y or o.z > z;
90      };
91      bool operator>(const S& o) const{
92          return o.x < x or o.y < y or o.z < z;
93      }
94      //more operators
95  };
```

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

Operator < = >

1. 3way comparison operator
2. may return
   a. std::strong_ordering
   b. std::weak_ordering
   c. std::partial_ordering
3. Generates all operator in simple cases

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

Operator < = >

```
97    struct S{
98        int x,y;
99        float z;
100
101       auto operator<=>(const S& o) const = default;
102   };
```

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

Operator < = >

```
106     S s1{1
107     S s2{1
108     fmt::p                                    n", s1 == s2);
109     fmt::p                                    ", s1 > s2);
110     fmt::p                                    ", s1 < s2);
111     fmt::p                                    n", s1 != s2);
112     fmt::p                                    n", s1 >= s2);
113     fmt::p                                    n", s1 <= s2);
```

```
s1==s2: false
s1>s2: false
s1<s2: true
s1!=s2: true
s1>=s2: false
s1<=s2: true
```

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

Strong Ordering

1. Denotes if a==b then f(a) == f(b)
2. exactly one of a < b, a == b, or a > b must be true

Weak Ordering

1. if a==b there may be some functions where f(a) != f(b)
2. equivalent values may be distinguishable
3. exactly one of a < b, a == b, or a > b must be true

Partial ordering

1. if a==b there may be some functions where f(a) != f(b)
2. equivalent values may be distinguishable
3. a < b, a == b, and a > b may all be false
4. <compare> header must be included to use special partial_ordering functionality

Operator < = >

```
520
521     auto operator<=>(const S& other) const noexcept {
522         auto mag = [](const S& v) -> float {
523             return std::sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
524         };
525         float lhs_len = mag(*this);
526         float rhs_len = mag(other);
527         if (lhs_len < rhs_len) return std::strong_ordering::less;
528         if (lhs_len > rhs_len) return std::strong_ordering::greater;
529         return std::strong_ordering::equal;
530     }
```

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

Operator < = >

```
532     bool operator==(const S& other) const  {
533         return (*this <=> other) == std::strong_ordering::equal;
534     }
```

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

Operator < = >

```
S s1{1, 3, 3.2};
S s2{3, 1, 3.2};
fmt::print("s1==s2: {}\n", s1 == s2);
fmt::print("s1>s2: {}\n", s1 > s2);
fmt::print("s1<s2: {}\n", s1 < s2);
fmt::print("s1!=s2: {}\n", s1 != s2);
fmt::print("s1>=s2: {}\n", s1 >= s2);
fmt::print("s1<=s2: {}\n", s1 <= s2);
```

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

Signed Integer Rules

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

Signed integer representation

- Stored using:
  - Sign and magnitude
  - One’s complement
  - Two’s complement

no_unique_address

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

no_unique_address attribute

```
110   struct PlaceHolder{};
111
112   struct S1{
113       uint32_t x;
114       PlaceHolder p;
115   };
116
117   struct S2{
118       uint32_t x;
119       [[no_unique_address]] PlaceHolder p;
120   };
```

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

no_unique_address attribute

```
124     fmt::print("Size of PlaceHolder: {}\n", sizeof(PlaceHolder));
125
126     Size of PlaceHolder: 1
127
128     fmt::print("Size of S1:{}\n", sizeof(S1));
129
130     Size of S1:8
131     fmt::print("Size of S2:{}\n", sizeof(S2));
132
        Size of S2:4
```

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

no_unique_address attribute

```
122    struct S3{
123        uint32_t x;
124        [[no_unique_address]] PlaceHolder p1, p2;
125    };
```

```
Size of S3:8
```

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

no_unique_address attribute

```
127   struct Empty{};
128
129   struct S4{
130       uint32_t x;
131       [[no_unique_address]] PlaceHolder p;
132       [[no_unique_address]] Empty e;
133   };
```

```
Size of S4:4
```

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

no_unique_address attribute

```
537   template <int i>
538   struct PlaceHolder{};
539
540   struct S5{
541       uint32_t x;
542       [[no_unique_address]] PlaceHolder<0> p0;
543       [[no_unique_address]] PlaceHolder<1> p1;
544   };
```

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

Lambdas Got better

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

Lambda parameter expansion

```
135   template <typename... T>
136   void func(T&&... a){
137       auto s = [&...x = a](){ return (x + ...); }();
138       fmt::print("sum={}\n", s);
139   }
```

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

Generic Lambdas

```
145     auto glambda = []<typename T>(T a, T b, auto c){
146         return a+b-c;
147     };
```

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

Generic Lambdas

```
141  template <size_t... IS, typename... T>
142  auto sum_N_first(std::index_sequence<IS...>, T&&... data){
143      return (std::get<IS>(std::tuple(data...)) + ...);
144  }
145
146  template <typename... T>
147  auto sum_first_5(T&&... a){
148      constexpr uint64_t num_args = sizeof...(T);
149      return sum_N_first(std::make_index_sequence<
150          num_args >= 5ul ? 5ul : num_args>(), a...);
151  }
```

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

Generic Lambdas

```
264     fmt::print("sum of the first 5 elements is:\n{}\n",
265     sum_first_5(1, 2, 2.1, 3.6f, 3.3, "hello", 'a', 22));
```

```
sum of the first 5 elements is:
11.999999904632569
```

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

Generic Lambdas

```
153  template <typename... T>
154  auto sum_first_5(T&&... a){
155      constexpr uint64_t num_args = sizeof...(T);
156      return [t = std::tuple(a...)]<size_t... IS>(
157          std::index_sequence<IS...>){
158          return (std::get<IS>(t) + ...);
159      }(std::make_index_sequence<
160      num_args >= 5ul ? 5ul : num_args>());
161  }
```

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

Generic Lambdas

```
264     fmt::print("sum of the first 5 elements is:\n{}\n",
265     sum_first_5(1, 2, 2.1, 3.6f, 3.3, "hello", 'a', 22));
```

```
sum of the first 5 elements is:
11.999999904632569
```

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

Constant Evaluation Got Better

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

Consteval

```
160   consteval int mult1(int x, int y){
161       return x*y;
162   }
163
164   constexpr int mult2(int x, int y){
165       return mult1(x, y);
166   }
```

```
error: call to consteval function 'mult1' is not a constant expression
```

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

Consteval

```
160  consteval int mult1(int x, int y){
161      return x*y;
162  }
163
164  consteval int mult2(int x, int y){
165      return mult1(x, y);
166  }
```

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

ConstInit

```
172     static constinit auto mult = mult1(10, 20);
173     mult += 20;
174     fmt::print("mult=={}\n", mult);
```

```
mult==220
```

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

Memory Allocation During Compile Time

```
168  consteval auto foo(){
169      auto x = new int[10];
170      delete[] x;
171      return true;
172  }
```

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

Memory Allocation During Compile Time

```
168  consteval auto foo(){
169      auto x = new int[10];
170      //delete[] x;
171      return true;
172  }
```

```
note: allocation performed here was not deallocated
```

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

NTTP is More Relaxed

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

NTTP Relaxation

```
174    template <float x>
175    void doit(){}
```

```
error: a non-type template parameter cannot have type 'float' before C++20
```

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

NTTP Relaxation

```cpp
template <std::size_t Size>
struct fixed_string {
    char data_[Size + 1]{0};
    static constexpr std::size_t size_ = Size;

    constexpr fixed_string(char const* str) {
        std::copy_n(str, Size + 1, data_);
    }

    constexpr fixed_string() = default;

    constexpr const char* data() const {
        return data_;
    }

    constexpr auto operator<=>(const fixed_string&) const = default;
};
```

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

NTTP Relaxation

```
195   template <unsigned int Size>
196   fixed_string(char const (&)[Size]) -> fixed_string<Size - 1>;
197
198   template<fixed_string Name>
199   constexpr auto operator""_fs() { return Name; };
200
201   template <fixed_string fs>
202   char get_last_letter(){
203       return fs.data()[decltype(fs)::size_-1];
204   };
```

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

NTTP Relaxation

```
208     fmt::print("Last Letter is: {}",
209     get_last_letter<"Hello"_fs>());
```

```
Last Letter is: o
```

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

Explicit or is it ?

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

Conditional Explicit

```
206    template <size_t N>
207    struct S{
208        int y_;
209        explicit(N % 2 == 0) S(int y): y_{y} {};
210    };
```

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

Conditional Explicit

```
212    template <size_t N>
213    void foo(S<N> s){};
```

```
217    foo<3>(2);
218
219    foo<2>(2);
```

```
note: candidate function template not viable: no known conversion from 'int' to 'S<2UL>'
```

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

Constexpr Polymorphism

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

Constexpr Polymorphism

```
3   struct Animal {
4       virtual constexpr ~Animal() = default;
5       virtual constexpr std::size_t num_legs() const = 0;
6   };
7
8   struct Duck : public Animal {
9       constexpr std::size_t num_legs() const override { return 2; }
10  };
11
12  struct Cat : public Animal {
13      constexpr std::size_t num_legs() const override { return 4; }
14  };
```

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

Constexpr Polymorphism

```
24  int main(){
25      constexpr Duck d;
26      constexpr Cat c;
27      static_assert(count_legs(std::array<const Animal*, 2>{{&d, &c}}) == 6);
28  }
```

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

Constexpr Polymorphism

```
16  constexpr std::size_t count_legs(auto animals){
17      std::size_t legs = 0;
18      for (const auto* a : animals) {
19          legs += a->num_legs();
20      }
21      return legs;
22  }
```

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

Library Features

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

Safer Integers

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

Safe Compare

```
4   int64_t func(auto x, auto y){
5       if (x < y) return y;
6       return x;
7   }
```

func(-10, 20ul) —> -10

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

Safe Compare

- `std::cmp_equal: ==`
- `std::cmp_not_equal: !=`
- `std::cmp_less: <`
- `std::cmp_less_equal: <=`
- `std::cmp_greater: >`
- `std::cmp_greater_equal: >=`

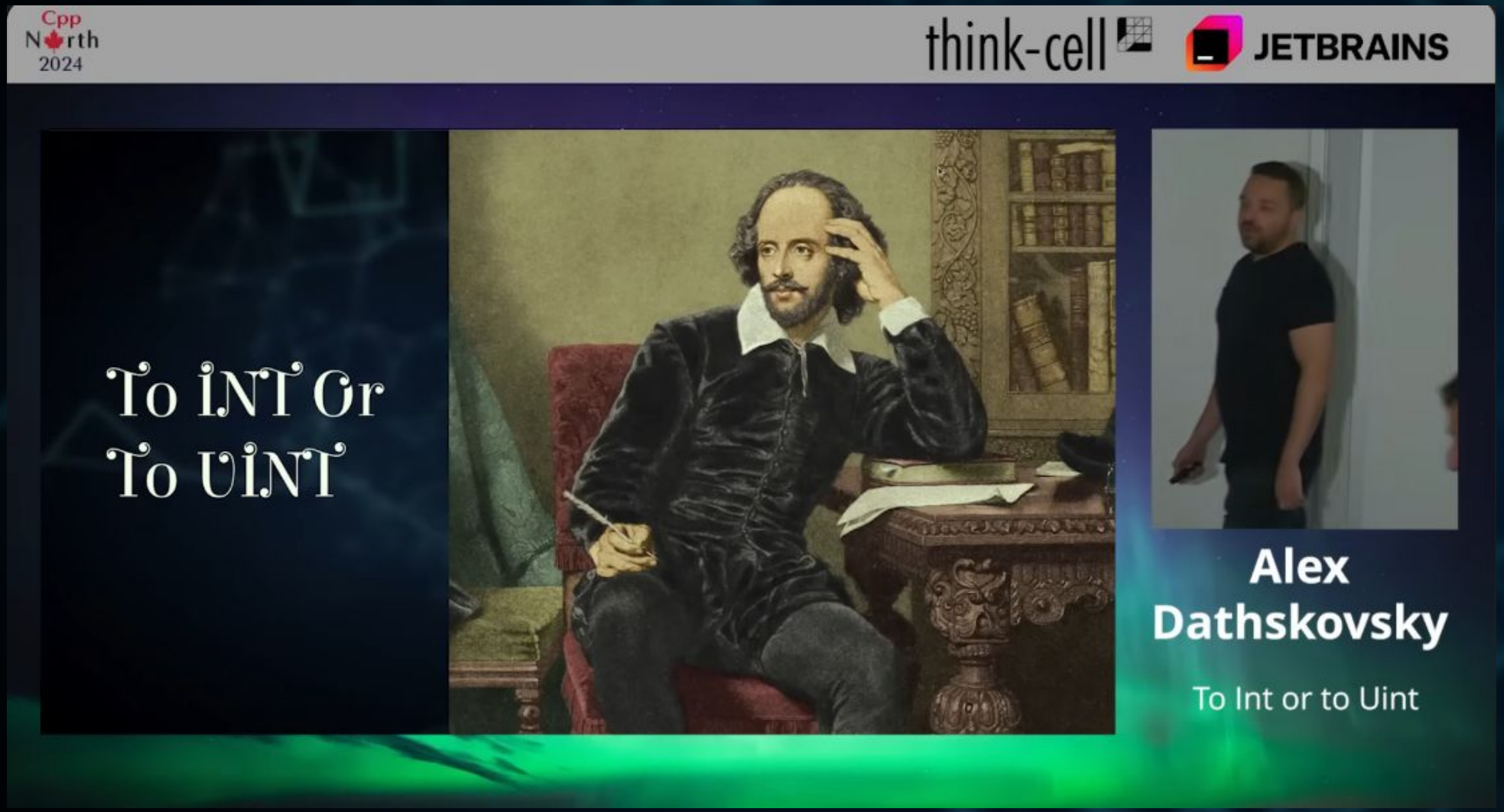
Safe Compare

```
4  int64_t func(auto x, auto y){
5      if (std::cmp_less(x, y)) return y;
6      return x;
7  }
```

func(-10, 20ul) —> 20

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

Signed integer representation



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

Heterogeneous Lookup

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

Lookup in Ordered Containers

```
498     std::map<std::string, int> bigMap{
499         {"/api/v1/users/profile/get", 1},
500         {"/api/v1/users/profile/update", 2},
501         {"/api/v1/users/orders/history", 3},
502         {"/api/v1/admin/config/reload", 4},
503         {"/api/v1/system/health/check", 5},
504         {"/api/v1/system/metrics/collect", 6}
505     };
```

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

Lookup in Ordered Containers

```
508     if (bigMap.find("/api/v1/users/profile/get") !=
509         bigMap.end()){ /*...*/ }
```

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

Lookup in Ordered Containers

```
499     std::map<std::string, int, std::less<>>
500     bigMap{
501         {"/api/v1/users/profile/get", 1},
502         {"/api/v1/users/profile/update", 2},
503         {"/api/v1/users/orders/history", 3},
504         {"/api/v1/admin/config/reload", 4},
505         {"/api/v1/system/health/check", 5},
506         {"/api/v1/system/metrics/collect", 6}
507     };
```

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

Lookup in Ordered Containers

```
500  struct hetro_str_hash {
501    using is_transparent = void;
502    [[nodiscard]] size_t operator()(const char* s) const {
503      return std::hash<std::string_view>{}(s);
504    }
505    [[nodiscard]] size_t operator()(std::string_view s) const {
506      return std::hash<std::string_view>{}(s);
507    }
508    [[nodiscard]] size_t operator()(const std::string& s) const {
509      return std::hash<std::string>{}(s);
510    }
511  };
```

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

Lookup in Ordered Containers

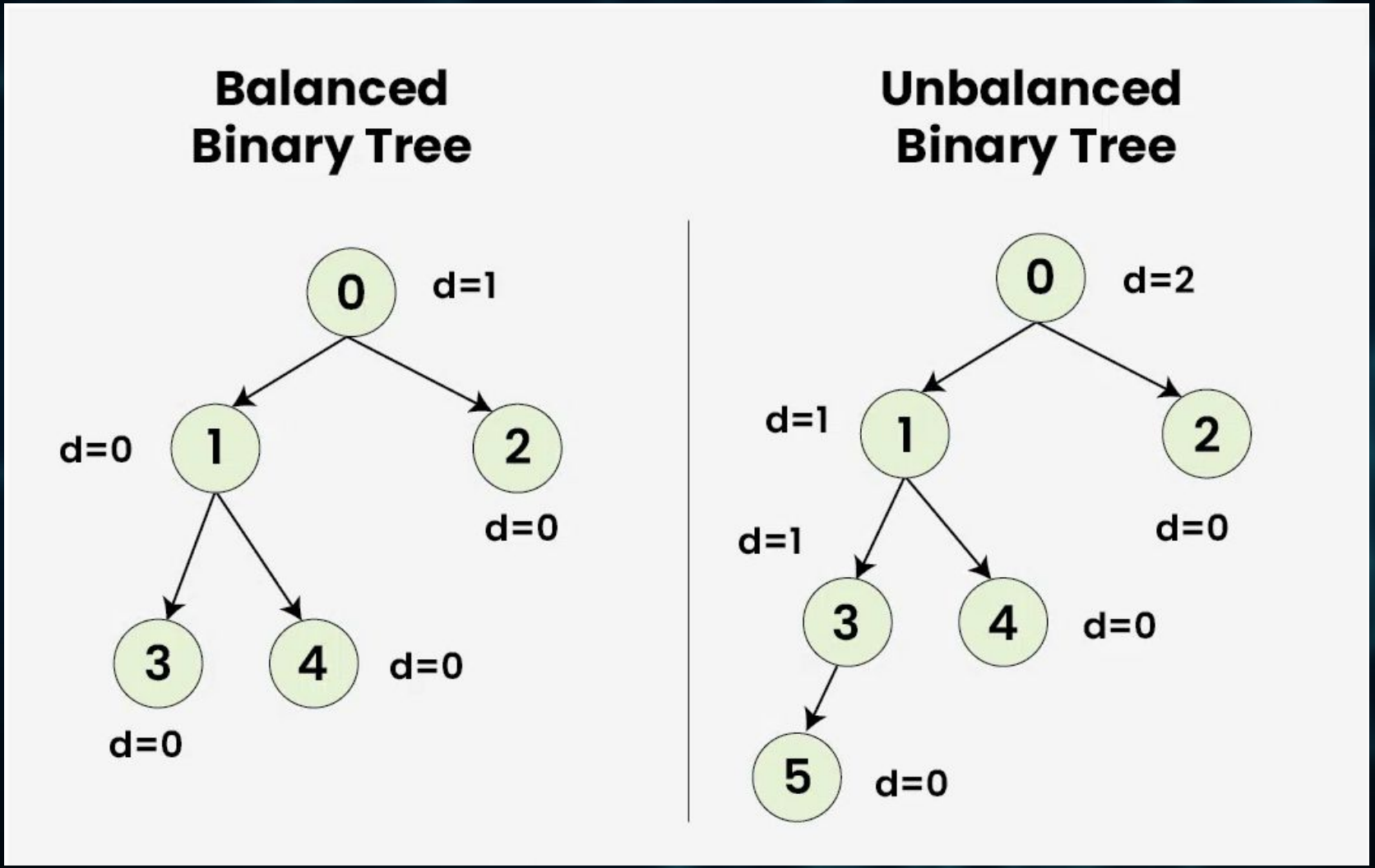
```
514     std::unordered_map<std::string, int,
515     hetro_str_hash, std::equal_to<>>
516     bigMap{
517         {"/api/v1/users/profile/get", 1},
518         {"/api/v1/users/profile/update", 2},
519         {"/api/v1/users/orders/history", 3},
520         {"/api/v1/admin/config/reload", 4},
521         {"/api/v1/system/health/check", 5},
522         {"/api/v1/system/metrics/collect", 6}
523     };
```

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

Span

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

Span: Creating a Balanced Tree From a Vector



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

Span: Creating a Balanced Tree From a Vector

```
1   #include <fmt/format.h>
2   #include <vector>
3   #include <span>
4   #include <memory>
5
6   struct Node{
7       explicit Node(int v): value{v} {};
8       int value;
9       std::unique_ptr<Node> left = nullptr;
10      std::unique_ptr<Node> right = nullptr;
11  };
12
```

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

Span: Creating a Balanced Tree From a Vector

```
13  std::unique_ptr<Node> create_tree(std::span<int> s){
14      auto get_span = [](std::size_t size, std::span<int> ret){
15          return (size <= 0) ? std::span<int>() : ret;};
16      if (s.empty()){
17          return nullptr;
18      }
19      std::size_t middle = s.size()/2;
20      auto root = std::make_unique<Node>(s[middle]);
21      root->left = create_tree(get_span(middle, {s.begin(), middle}));
22      root->right = create_tree(get_span(s.size()-middle-1, {s.begin()+middle+1, s.size()-middle-1}));
23      return root;
24  }
```

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

Span: Creating a Balanced Tree From a Vector

```
27  int main(){
28      std::vector<int> v{1,2,3,4,5,6};
29      auto tree = create_tree(v);
30      return 0;
31  }
```

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

<bit> header

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

<bit> header

Types

`endian` (C++20)	indicates the endianness of scalar types (enum)

Functions

`bit_cast` (C++20)	reinterpret the object representation of one type as that of another (function template)
`byteswap` (C++23)	reverses the bytes in the given integer value (function template)
`has_single_bit` (C++20)	checks if a number is an integral power of `2` (function template)
`bit_ceil` (C++20)	finds the smallest integral power of two not less than the given value (function template)
`bit_floor` (C++20)	finds the largest integral power of two not greater than the given value (function template)
`bit_width` (C++20)	finds the smallest number of bits needed to represent the given value (function template)
`rotl` (C++20)	computes the result of bitwise left-rotation (function template)
`rotr` (C++20)	computes the result of bitwise right-rotation (function template)
`countl_zero` (C++20)	counts the number of consecutive `0` bits, starting from the most significant bit (function template)
`countl_one` (C++20)	counts the number of consecutive `1` bits, starting from the most significant bit (function template)
`countr_zero` (C++20)	counts the number of consecutive `0` bits, starting from the least significant bit (function template)
`countr_one` (C++20)	counts the number of consecutive `1` bits, starting from the least significant bit (function template)
`popcount` (C++20)	counts the number of `1` bits in an unsigned integer (function template)

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

<bit> header: Endian

```cpp
enum class Endian {
    Little,
    Big,
    Unknown
};

Endian check_endian() {
    uint16_t number = 0x1; // 16-bit number
    uint8_t* ptr = reinterpret_cast<uint8_t*>(&number);

    if (*ptr == 0x1) {
        return Endian::Little;
    } else if (*ptr == 0x0) {
        return Endian::Big;
    } else {
        return Endian::Unknown;
    }
}
```

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

<bit> header: Endian

```
237     Endian endian = check_endian();
238
239     switch (endian) {
240         case Endian::Little:
241             fmt::print("System is Little Endian\n");
242             break;
243         case Endian::Big:
244             fmt::print("System is Big Endian\n");
245             break;
246         default:
247             fmt::print("System Endianness is Unknown\n");
248     }
```

```
System is Little Endian
```

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

<bit> header: Endian

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

<bit> header: Endian

```
238     if (std::endian::native == std::endian::little){
239         fmt::print("System is Little Endian\n");
240     } else if (std::endian::native == std::endian::big){
241         fmt::print("System is Big Endian\n");
242     }
```

```
System is Little Endian
```

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

<bit> header: Endian

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

<bit> header: popcount

```
239  struct S{
240      uint32_t a: 4;
241      uint32_t b: 28;
242  };
243
244  constexpr uint32_t popcount(uint32_t val){
245      uint32_t res{};
246      while (val){
247          res += val & 0x1;
248          val >>= 1;
249      }
250      return res;
251  }
```

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

<bit> header: popcount

```
256     static constexpr S s{~0u, ~0u};
257     static constexpr auto ca = popcount(s.a);
258     static constexpr auto cb = popcount(s.b);
259     static_assert(ca == std::popcount(s.a));
260     static_assert(cb == std::popcount(s.b));
261
```

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

<bit> header: popcount

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

<bit> header: bit_cast

```
238     float y = 0.256;
239     auto z = std::bit_cast<uint32_t>(y);
240
241     fmt::print("y={}, z={}", y, z);
```

```
y=0.256, z=1048777327
```

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

<bit> header: bit_cast

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

<bit> header: bit_cast

Alex Dathskovsky - Misusing reinterpret_cast?! You Probably Are :) (Keynote)

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

<bit> header: bit_cast is Not Perfect

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

Source Location

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

source_location

```cpp
template <>
struct fmt::formatter<std::source_location> :
fmt::formatter<std::string_view>{
    constexpr auto format(std::source_location sl, fmt::format_context& ctx){
        auto formatted = fmt::format("[{}]-[{}]:{}:{}", sl.file_name(),
        sl.function_name(), sl.line(), sl.column());
        return formatter<std::string_view>::format(formatted, ctx);
    }
};


void log_error(std::string_view sf,
               std::source_location loc=std::source_location::current()){
    fmt::print("location: {} message: {}\n", loc, sf);
}

int main(){
    //location: [/app/example.cpp]-[int main()]:69:5 message: This is error1
    log_error("This is error1");
    //location: [/app/example.cpp]-[int main()]:71:5 message: This is error2
    log_error(fmt::format("This is error{}", 2));
};
```

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

sorce_location

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

Better Threading

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

Jthread

- Same as std::thread but is joinable by default
- Jthread is stoppable with std::stop_source
- Provides easier implementation where user don’t have to think about joins

semaphore

```
 8  std::binary_semaphore start_speaking{0}, person1_spoke{0},
 9                        person2_spoke{0};
10
11
12  void person_speak(std::string_view text, std::binary_semaphore& w, std::binary_semaphore& r){
13      w.acquire();
14      fmt::print("{}", text);
15      std::this_thread::sleep_for(
16      r.release();
17  }
18
19
20  int main(){
21      std::jthread p1(person_speak, "Hello Person2\n", std::ref(start_speaking), std::ref(person1_spoke));
22      std::jthread p2(person_speak, "Hello person1\n", std::ref(person1_spoke), std::ref(person2_spoke));
23      start_speaking.release();
24      person2_spoke.acquire();
25
26      fmt::print("GoodBye!");
27
28      return 0;
29  }
```

```
Hello Person2
Hello person1
GoodBye!
```

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

stop_source

```cpp
std::binary_semaphore start_speaking{0}, p1_done{0}, p2_done{0};

void person_speak(std::stop_token s, std::string_view text, std::binary_semaphore& w, std::binary_semaphore& r){
    w.acquire();
    while (not s.stop_re
        fmt::print("{}",
        std::this_thread
    }
    r.release();
}

void stop_speaking(std::
    for (auto i = 0u; i
        std::this_thread
        ssrc[i].request_
    }
}


int main() {
    std::jthread p1(pers
                    std:
    std::jthread p2(pers
                    std:
    start_speaking.relea
    std::jthread sched(stop_speaking, std::array{p1.get_stop_source(), p2.get_stop_source()});
    p2_done.acquire();
    fmt::print("Goodby!");
}
```

```
Hello Peson2 im Person1
Hello Peson2 im Person1
Hello Peson2 im Person1
Hello Peson2 im Person1
Hello Peson1 im Person2
Hello Peson1 im Person2
Hello Peson1 im Person2
Hello Peson1 im Person2
Goodby!
```

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

Latch

```cpp
int main(){
    std::vector<std::string_view> ppl{"Alex", "Dani", "Benny", "Guy"};
    std::vector<std::jthread> workers;
    std::latch jobs{ppl.size()}, go_home{1};
    auto job = [&jobs, &go_home](std::string_view name){
        fmt::print("{} is doing a job\n", name);
        jobs.count_down();
        go_home.wait();
        fmt::print("{} is going home\n", name);
    };

    for (const auto& p : ppl) {
        workers.push_back(std::jthread(job, p));
    }
    jobs.wait();
    fmt::print("Go home!\n");
    go_home.count_down();
}
```

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

Latch

```
Alex is doing a job
Dani is doing a job
Guy is doing a job
Benny is doing a job
Go home!
Alex is going home
Dani is going home
Guy is going home
Benny is going home
```

Barrier

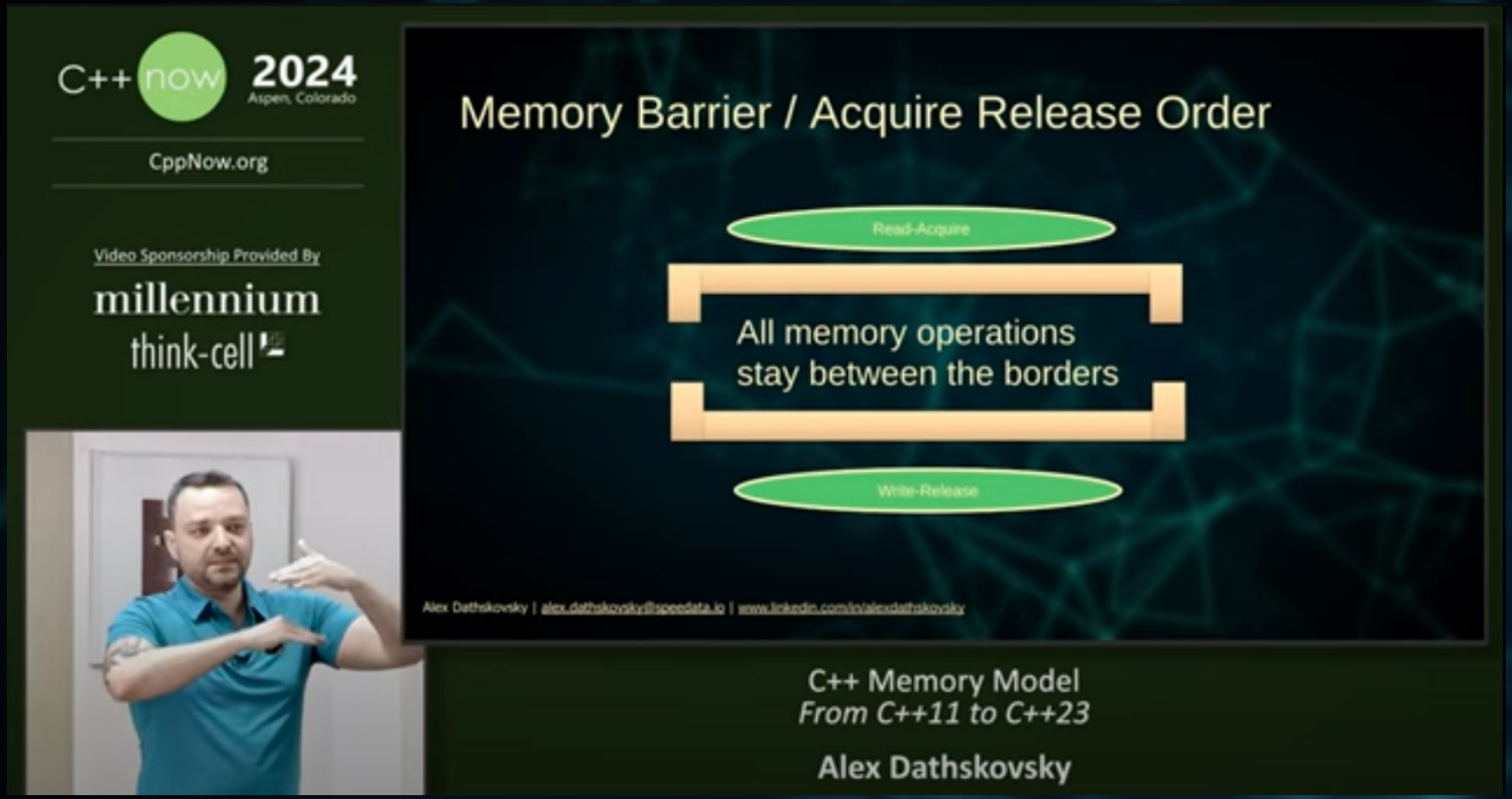
```
162  int main(){
163      std::vector<std::string_view> ppl{"Alex", "Dani", "Benny", "Guy"};
164      std::vector<std::jthread> workers;
165      auto on_complete = [](){
166          static int x = 0;
167          not x++ ? fmt::print("Go Home!\n") : fmt::print("Done!\n"); };
168      std::barrier sync{ppl.size(), on_complete};
169      auto job = [&sync](std::string_view name){
170          fmt::print("{} is doing a job\n", name);
171          sync.arrive_and_wait();
172          fmt::print("{} is going home\n", name);
173          sync.arrive_and_wait();
174      };
175
176      for (const auto& p : ppl) {
177          workers.push_back(std::jthread(job, p));
178      }
179  }
```

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

Barrier

```
Dani is doing a job
Guy is doing a job
Benny is doing a job
Alex is doing a job
Go Home!
Benny is going home
Alex is going home
Dani is going home
Guy is going home
Done!
```

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



https://youtu.be/VWiUYbtSWRI?si=kfhuZK-ZgorpKTDi

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

Barrier

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

Pop Quiz

```
11      ([](const auto&... args){
12          auto count = sizeof...(args);
13          ([&count](const auto& arg){
14              std::cout << arg <<(--count ? ", " : "\n");
15              return std::ignore;
16          }(args) = ...);
17      })(1, 2, 'a', "Hello");
18
```

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