Safely Encapsulating Fixed-size Arrays with std::array

Giovanni Dicanio
AUTHOR, SOFTWARE ENGINEER

https://blogs.msmvps.com/gdicanio

Overview

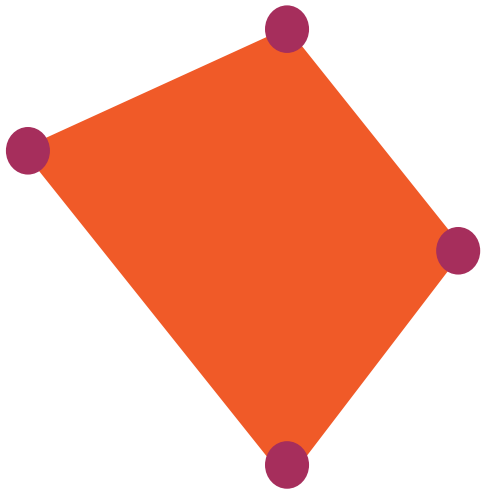
Introduction to std::array

- std::array vs. std::vector

Common operations

Reuse Standard Library's algorithms

Concrete Example: X/Y Coordinates for a Quad



```
// X/Y coordinates for a quad:
//   2 floats (X/Y) per vertex * 4 vertices = 8 floats
std::vector<float> quad(8);
```

X/Y Coordinates for a Quad

Using std::vector

```
// X/Y coordinates for a quad:
//   2 floats (X/Y) per vertex * 4 vertices = 8 floats
std::vector<float> quad(8);
```

X/Y Coordinates for a Quad

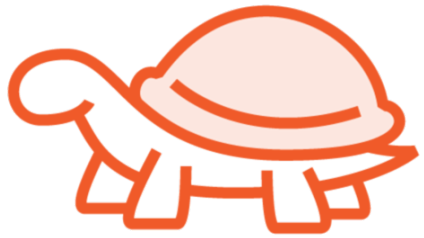
Using std::vector

```
// X/Y coordinates for a quad:
//   2 floats (X/Y) per vertex * 4 vertices = 8 floats
std::vector<float> quad(8);
```

std::vector (default):
dynamic (*heap*) memory allocation
new[]

X/Y Coordinates for a Quad

Using std::vector



```
// X/Y coordinates for a quad:
//   2 floats (X/Y) per vertex * 4 vertices = 8 floats
float quad[8];
```

X/Y Coordinates for a Quad

Optimization: stack-allocated C-style array

```
// X/Y coordinates for a quad:
//   2 floats (X/Y) per vertex * 4 vertices = 8 floats
float quad[8];
```

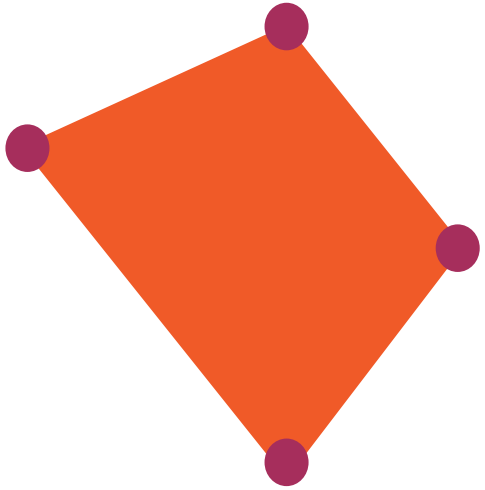
X/Y Coordinates for a Quad

Optimization: stack-allocated C-style array

In C++: prefer *std::array* to C-style arrays

std::array

Zero-overhead



Very small containers

Fixed-size

std::array vs. std::vector

std::array

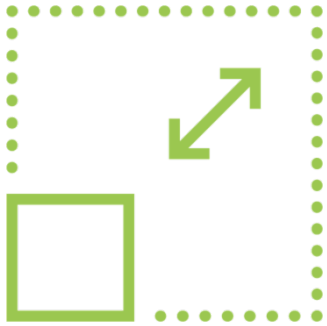
Zero-overhead

Fixed-size

Stack space is limited

std::vector

Dynamic allocation overhead



Resizable

Heap is fine for large allocations

std::array vs. std::vector

std::array

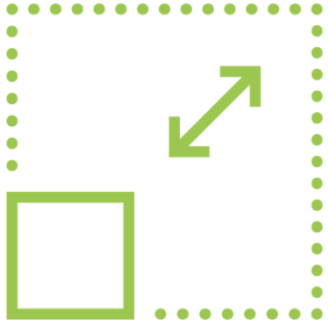
Zero-overhead

Fixed-size

Stack space is limited

std::vector

Dynamic allocation overhead



Resizable

Heap is fine for large allocations

std::array vs. std::vector

std::array

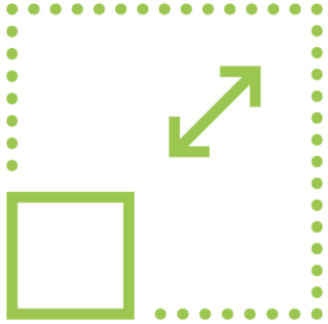
Zero-overhead

Fixed-size

Stack space is limited

std::vector

Dynamic allocation overhead



Resizable

Heap is fine for large allocations

```
std::array<int, 3> a = {11, 22, 33};
```

std::array Construction

ELEMENT TYPE

```
std::array<int, 3> a = {11, 22, 33};
```

std::array Construction

NUMBER OF ELEMENTS

```
std::array<int, 3> a = {11, 22, 33};
```

std::array Construction

INITIAL VALUES

```
std::array<int, 3> a = {11, 22, 33};
```

std::array Construction

OMITTING THE EQUALS SIGN

```
std::array<int, 3> a{11, 22, 33};
```

std::array Construction

Element *type* and element *count* automatically deduced

```
std::array a{11, 22, 33};
```

std::array Construction

Requires at least C++17

```cpp
std::array<double, 4> a = {10.0, 20.0, 30.0, 40.0};

std::cout << "Number of elements: " << a.size();
```

std::array::size Method

Returns the number of elements in the container

```cpp
std::array<double, 4> a = {10.0, 20.0, 30.0, 40.0};

std::cout << "Number of elements: " << a.size();
```

std::array::size Method

Returns the number of elements in the container

array::size returns the number of *elements*

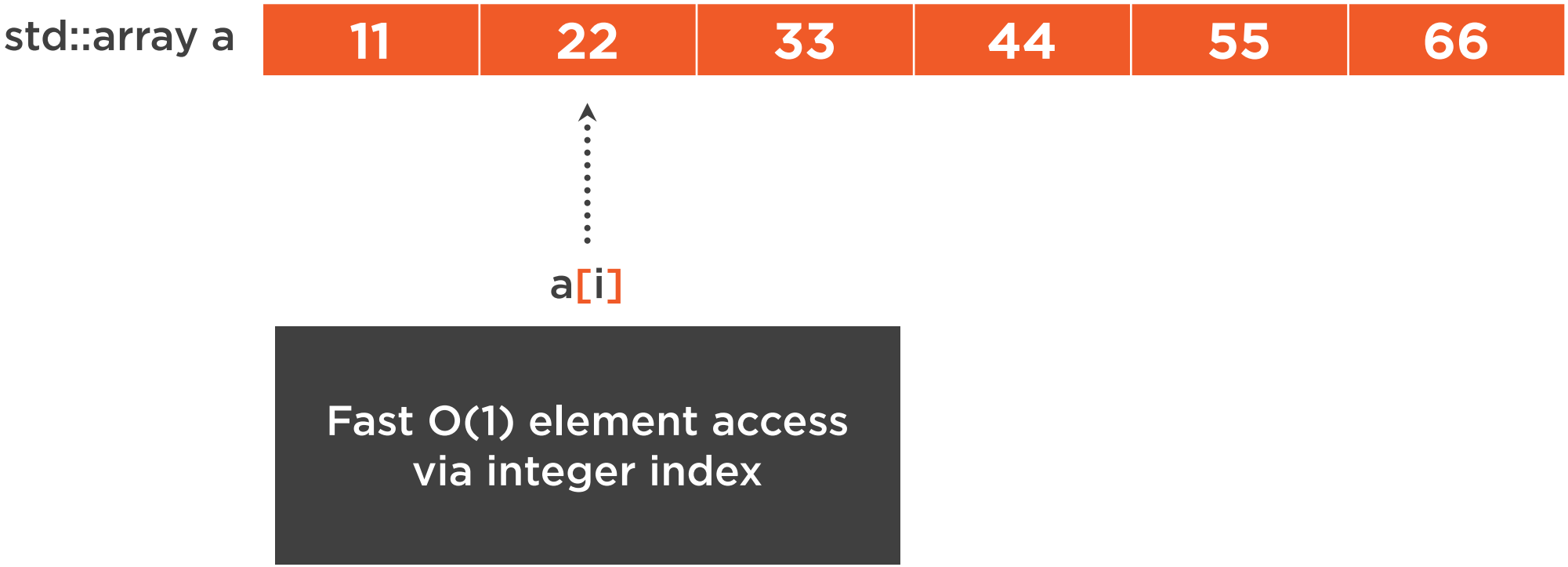
sizeof returns the number of *bytes*

```
std::array  ::size()
std::vector::size()
std::list   ::size()
std:: ...   ::size()
```

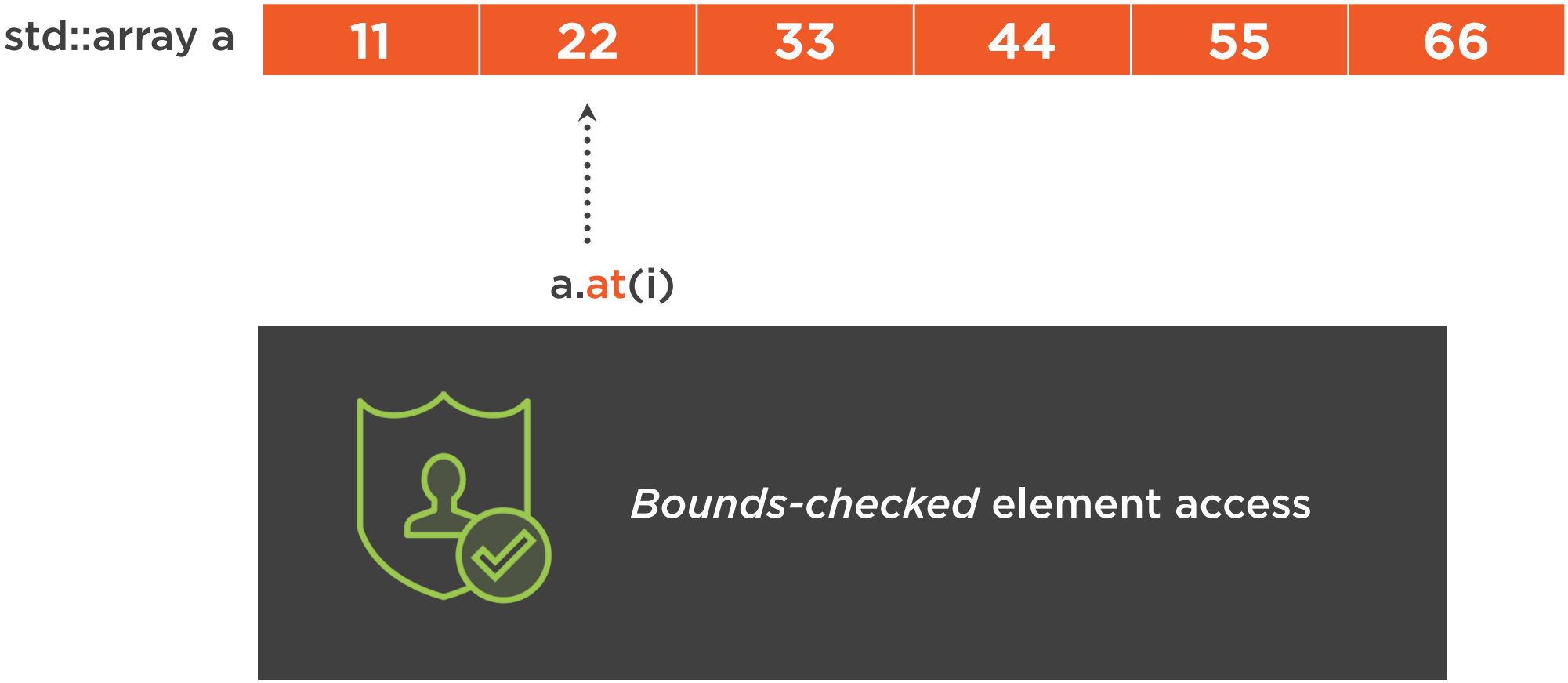
container::size Method

Returns the number of elements in the container

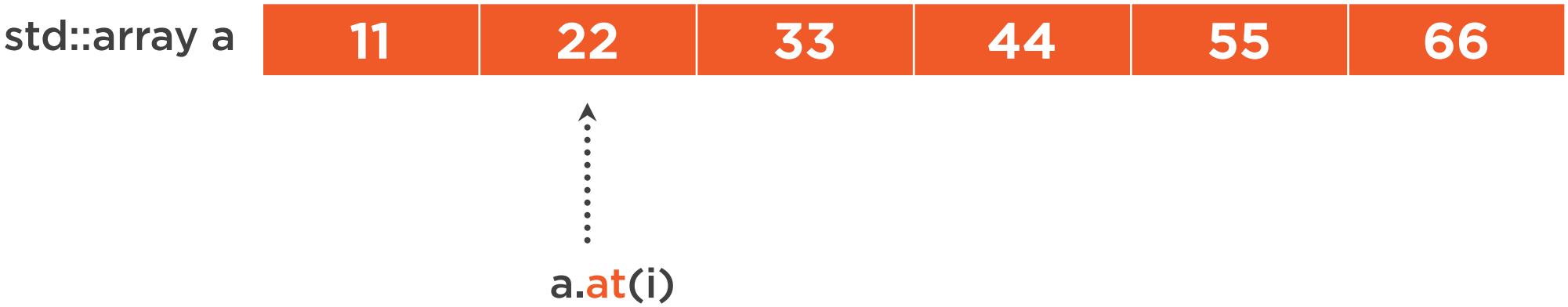
Accessing std::array's Elements



Accessing std::array's Elements



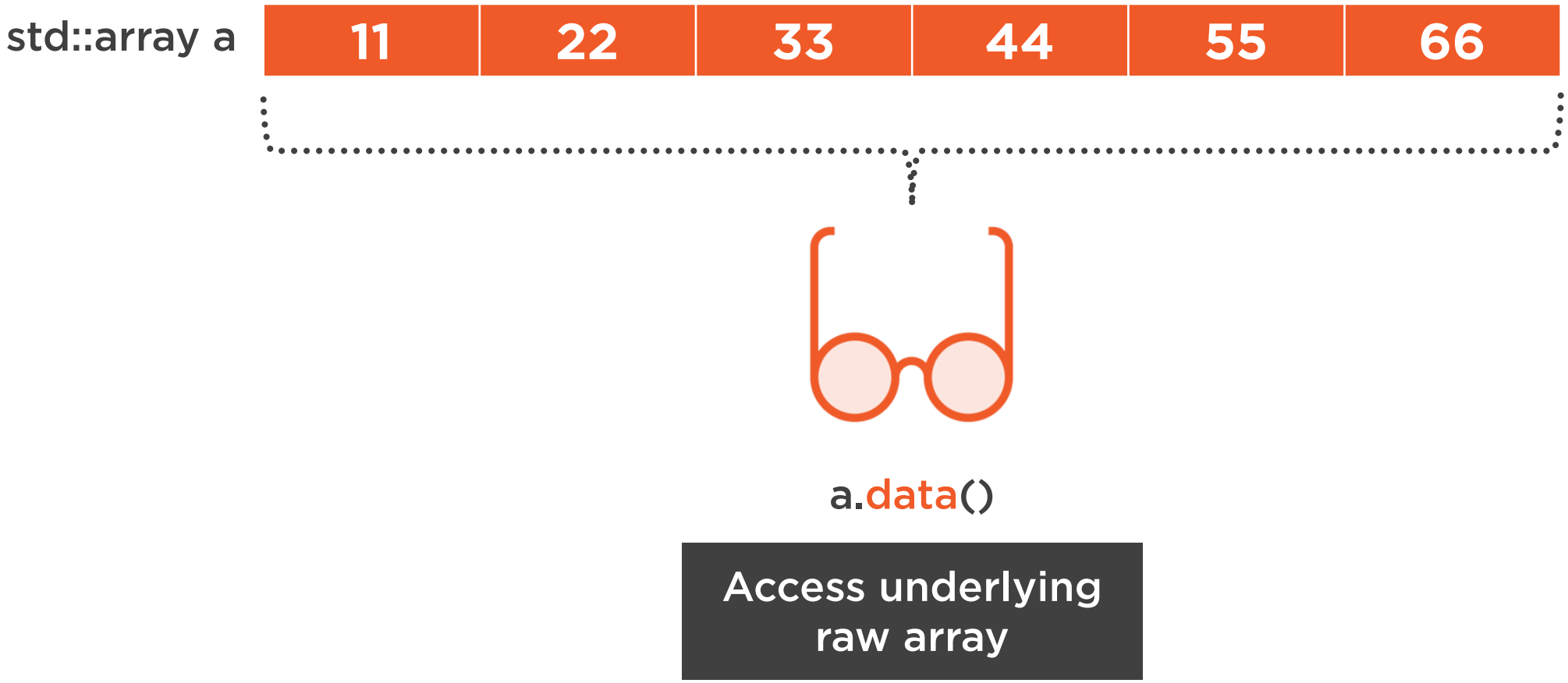
Accessing std::array's Elements



Safely Accessing std::vector Elements

Module: «Storing Sequences of Elements with the Standard std::vector Container»

Accessing std::array's Elements



```
int a1[] = {10, 20, 30};
int a2[] = {11, 22, 33};

a2 = a1; // Compiler error
```

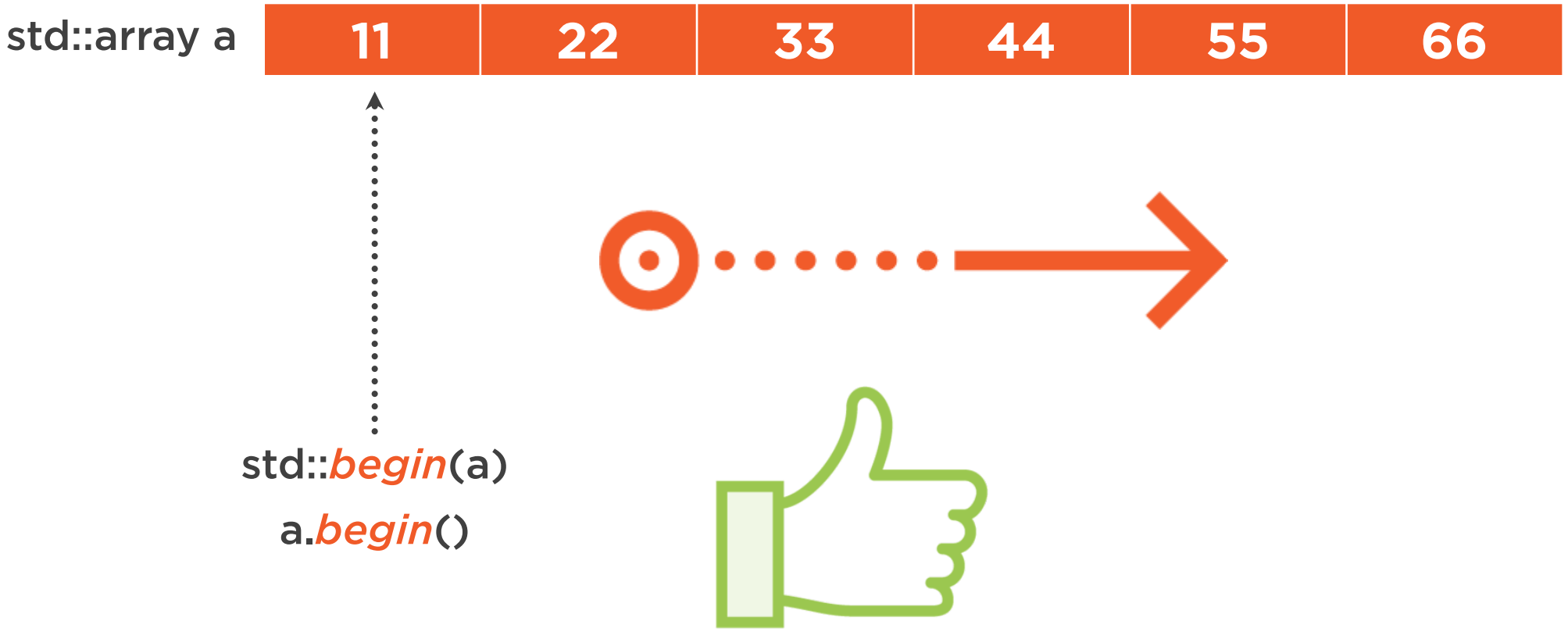
- You can’t simply assign built-in arrays

```cpp
std::array a1 = {10, 20, 30};
std::array a2 = {11, 22, 33};

a2 = a1; // Works fine
```

◀ std::array supports assignment

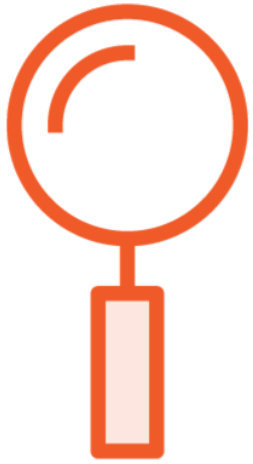
Iterators for std::array



Searching for Elements in std::array

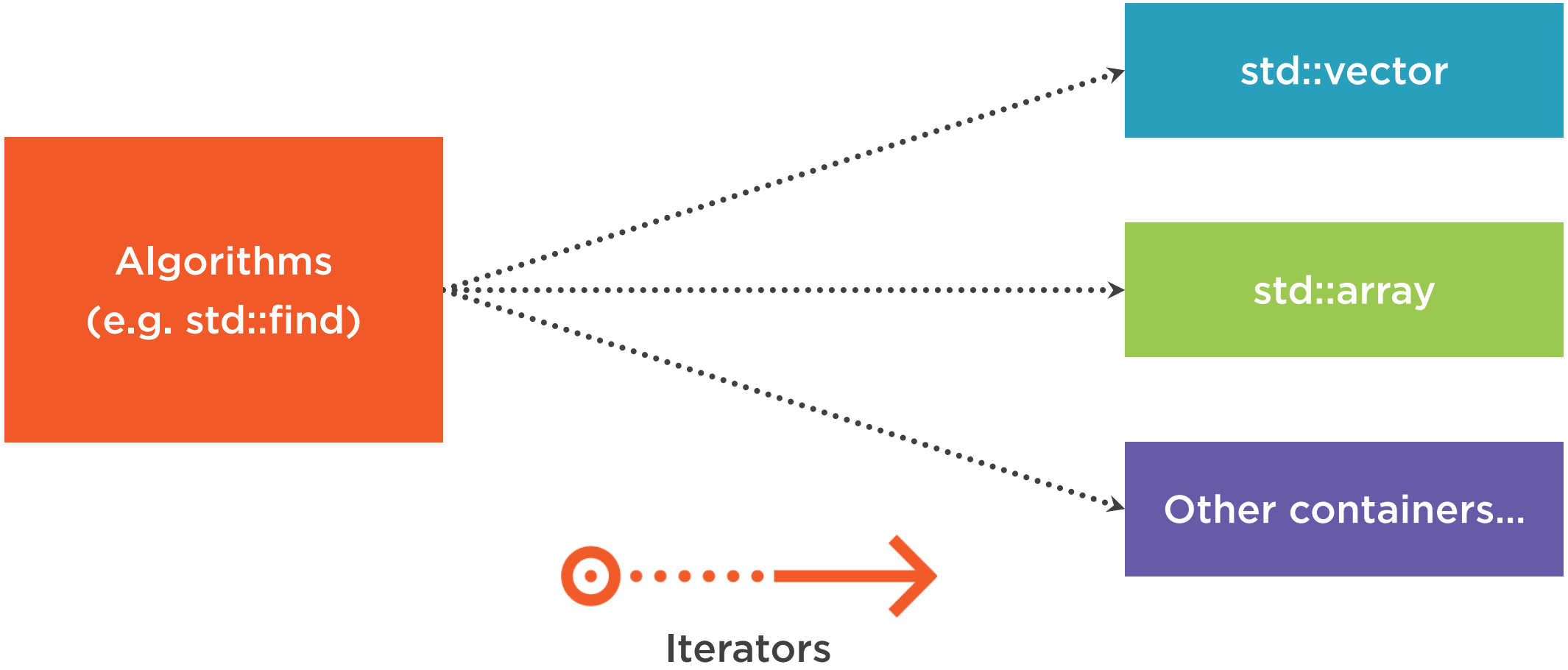
67	79	78	78	73	69

std::array::*find* method?

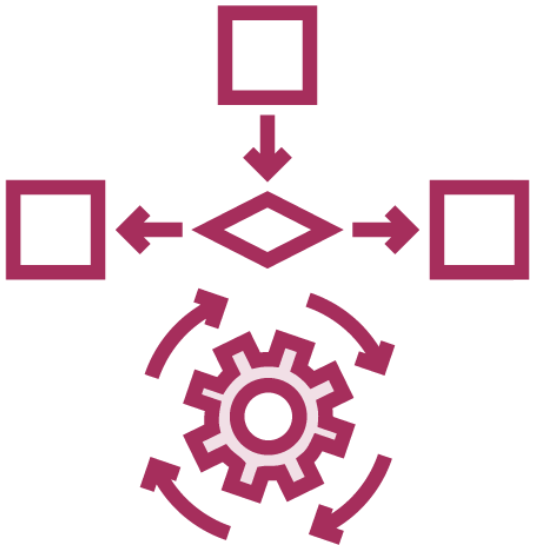


std::*find* algorithm

Reusing Algorithms with Different Containers

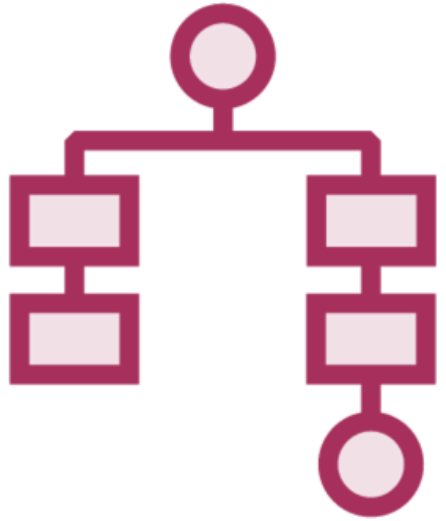


Algorithms, Iterators, and Containers



Algorithms

Iterators



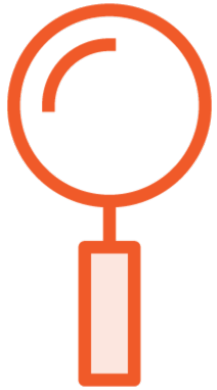
Containers

From module:
"Breaking the Ice with Useful Standard Algorithms: Sorting std::vector"

```
// std::array a
auto result = std::find(begin(a), end(a), value);
```

Searching for Elements in std::array

Use the std::find algorithm

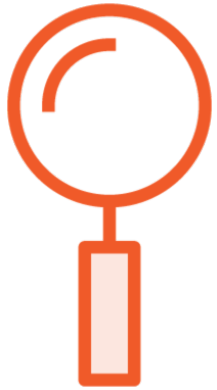


```
// std::array a
auto result = std::find(begin(a), end(a), value);

// std::vector v
auto result = std::find(begin(v), end(v), value);
```

Searching for Elements in std::array

Use the std::find algorithm

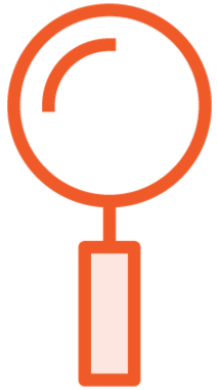


```
// std::array a
auto result = std::find(begin(a), end(a), value);

// std::vector v
auto result = std::find(begin(v), end(v), value);
```

Searching for Elements in std::array

Use the std::find algorithm



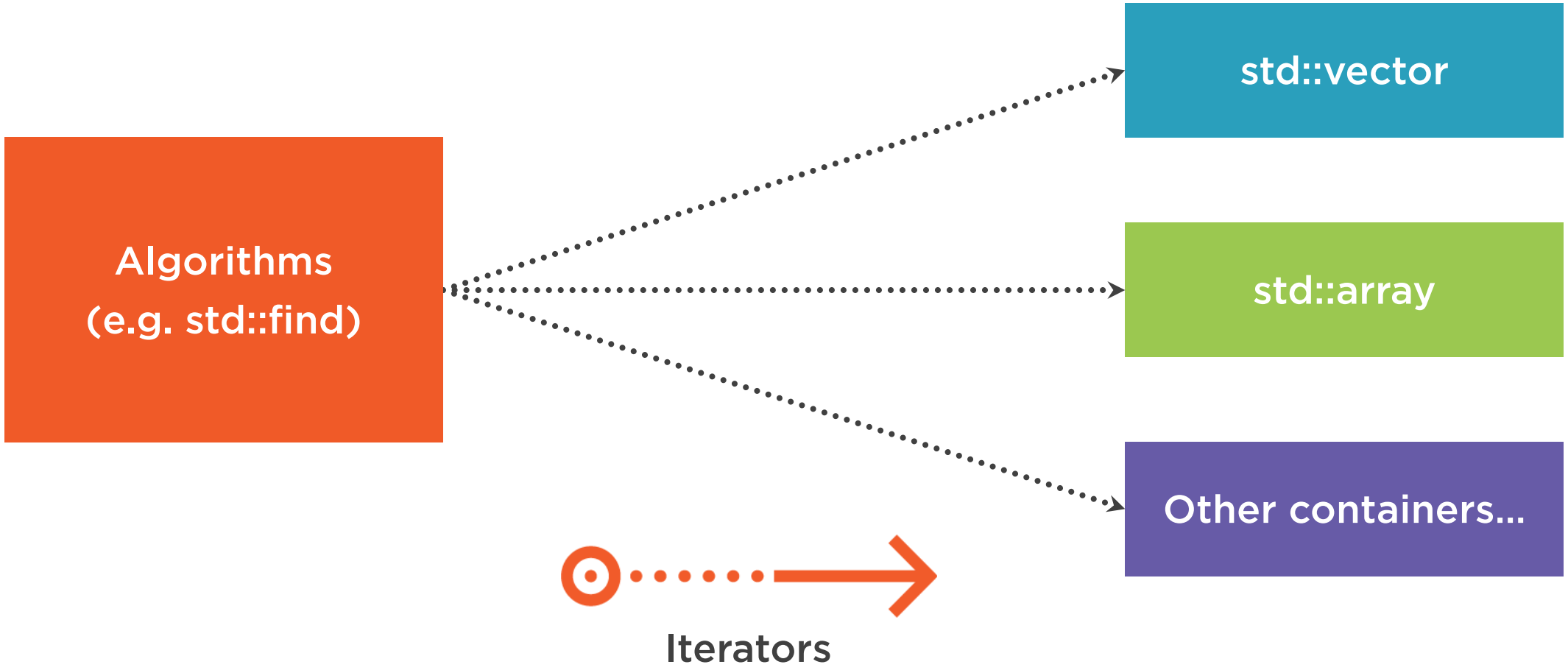
```
// std::array a
std::sort(begin(a), end(a));
```

Sorting Elements in std::array

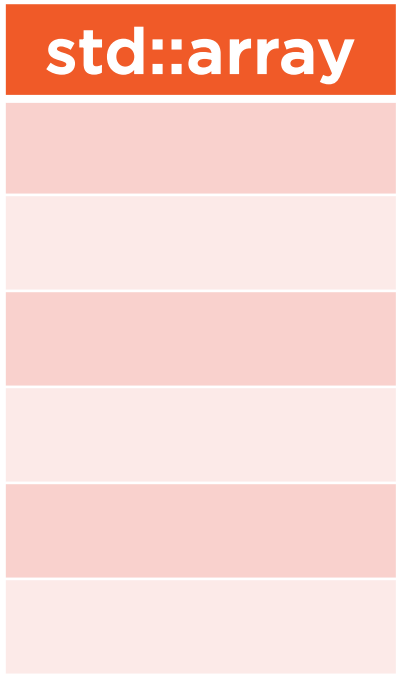
```
// std::array a
std::sort(begin(a), end(a));
```

Sorting Elements in std::array

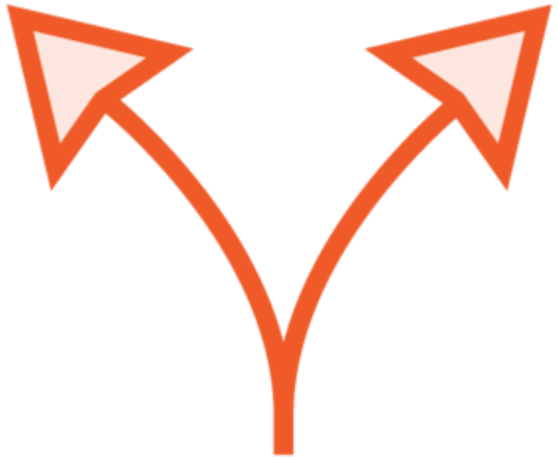
Reusing Algorithms with Different Containers



Implementing Look-up Tables with std::array



Array content must be *sorted*

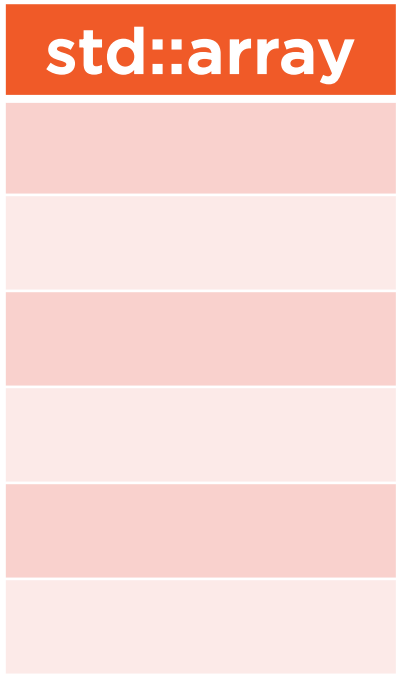


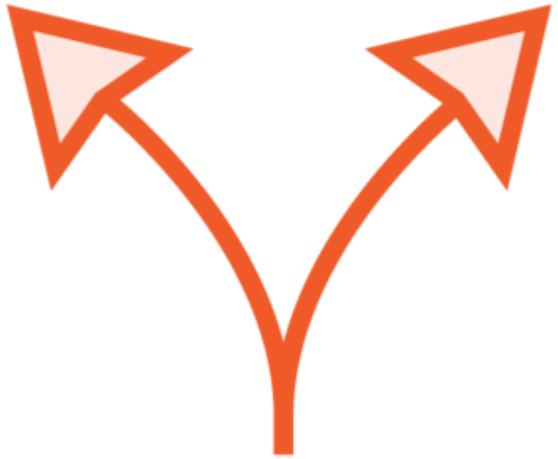
To Learn More on Binary Search...

Module: "*Efficiently Searching*"

Course: "Introduction to Data Structures and Algorithms in C++"

Implementing Look-up Tables with std::array





Binary search algorithm:
std::*lower_bound*

```
// Function scope

static constexpr std::array table = {
    …
};
```

No machine code generated to push values on the stack

Implementing Look-up Tables with std::array

Tip: Use *static constexpr*

Summary

Introduction to the std::array container

- Performance and zero-overhead high-level standard C++ container

When to use std::array

Reuse standard algorithms with std::array