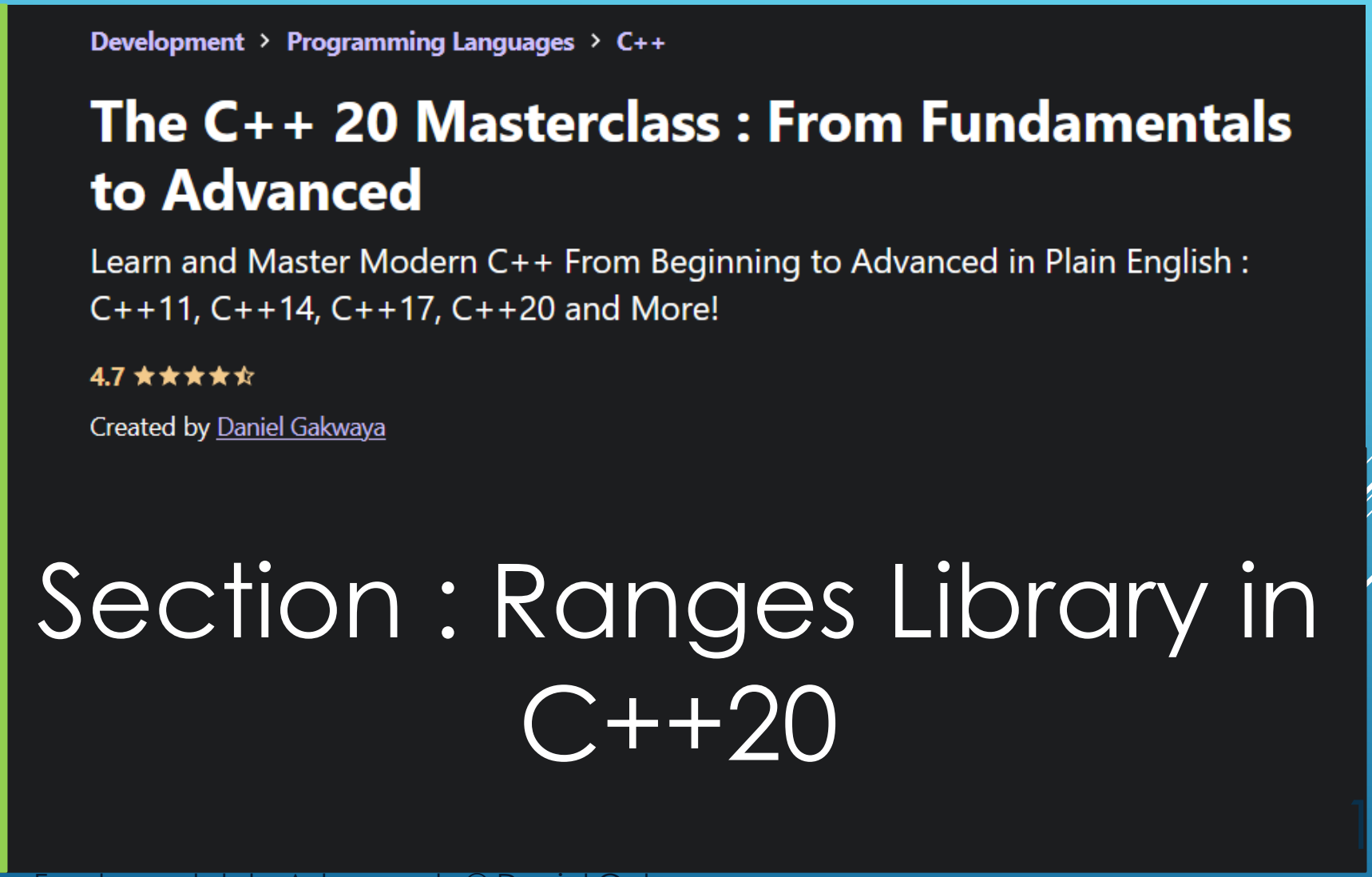
Slides

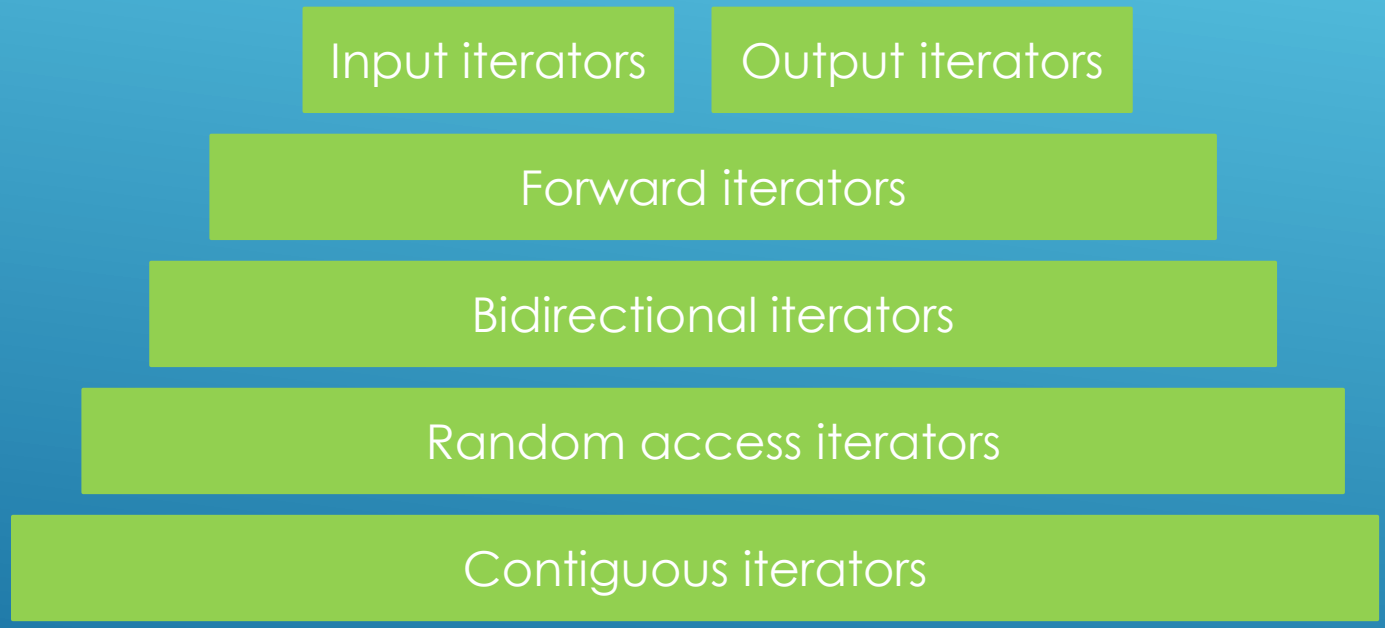


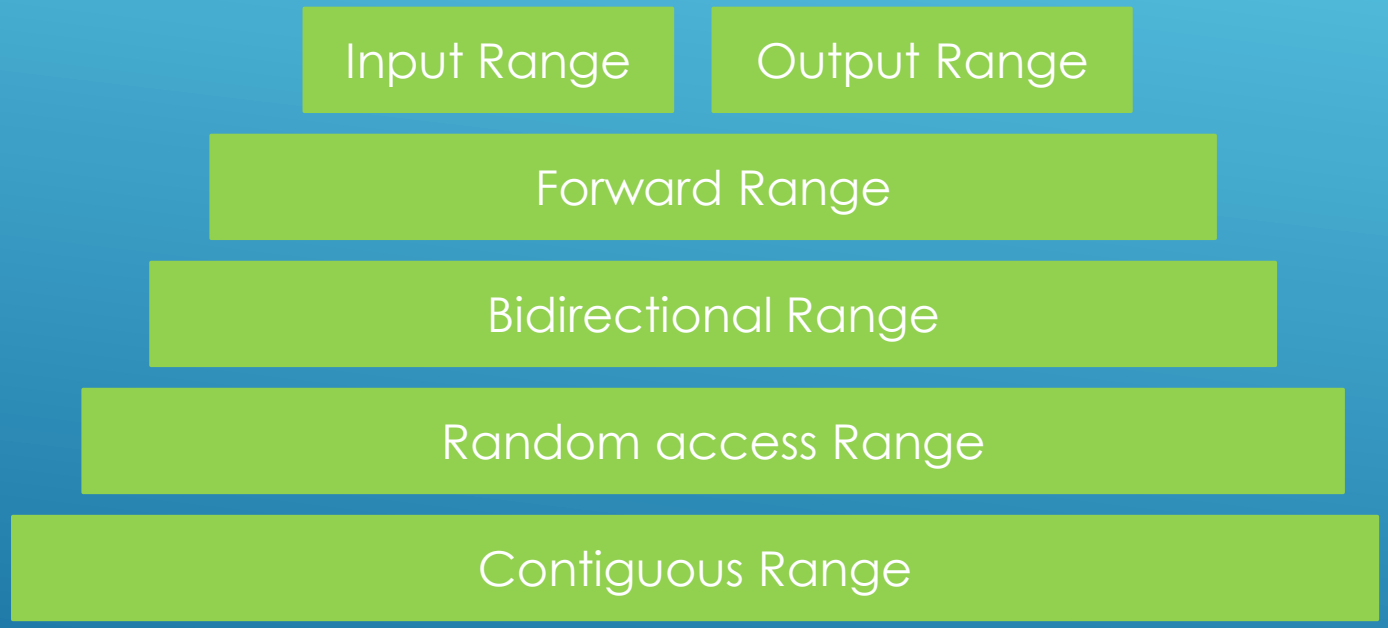
Section : Ranges Library in C++20

Slide intentionally left empty

Ranges Library in C++20

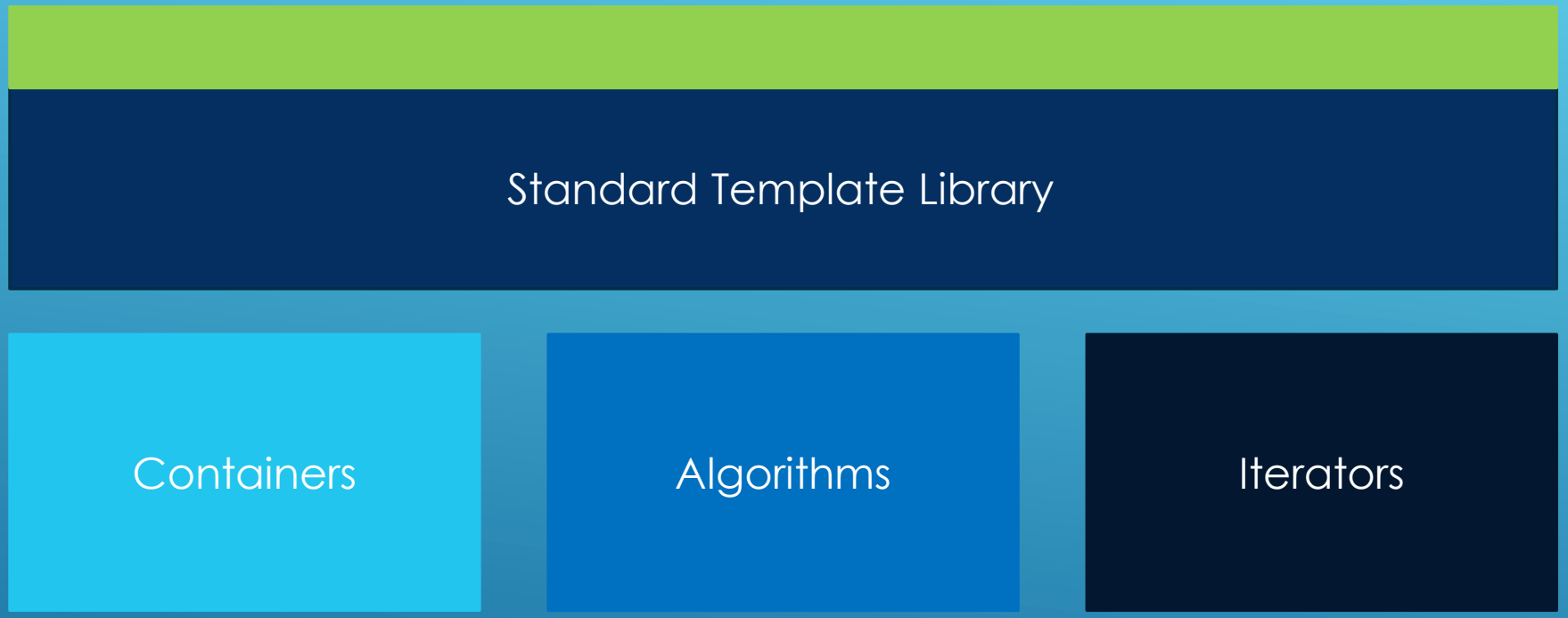
- Range algorithms
- Projections
- Views and view adaptors
- Function composition
- Range factories


Input iterators
Output iterators
Forward iterators
Bidirectional iterators
Random access iterators
Contiguous iterators


Input Range
Output Range
Forward Range
Bidirectional Range
Random access Range
Contiguous Range

Slide intentionally left empty

Range Algorithms


Standard Template Library
Containers
Algorithms
Iterators


Legacy algorithms
Work on iterator pairs
Range algorithms
Work on containers directly

std::all_of [Iterator pair]

```
//std::vector<int> collection{2,6,8,40,64,70};
//std::set<int> collection{2,6,8,40,64,70};
int collection[] {2,6,8,40,64,70};

//std::all_of , lambda function predicate
if (std::all_of(std::begin(collection), std::end(collection), [](int i){ return i % 2 == 0; })) {
    std::cout << "(std::all_of) :  All numbers in collection are even" << std::endl;
}else{
    std::cout << "(std::all_of) : Not all numbers in collection are even" << std::endl;
}
```

std::ranges::all_of

```
std::vector<int> numbers {11,2,6,4,8,3,17,9};
print_collection(numbers);

//std::ranges::all_of()
std::cout << std::endl;
std::cout << "std::ranges::all_of() : " << std::endl;

auto odd = [](int n){
    return n%2 !=0;
};

auto result = std::ranges::all_of(numbers,odd);

if(result){
    std::cout << "All elements in numbers are odd" << std::endl;
}else{
    std::cout << "Not all elements in numbers are odd" << std::endl;
}
```

std::ranges::for_each

```
//For each
std::cout << std::endl;
std::cout << "std::ranges::for_each() : " << std::endl;
print_collection(numbers);
std::ranges::for_each(numbers,[](int& n){n*=2;});
print_collection(numbers);
```

std::ranges::sort

```
//Sort
std::cout << std::endl;
std::cout << "std::ranges::sort() : " << std::endl;
print_collection(numbers);
std::ranges::sort(numbers);
print_collection(numbers);
```

std::ranges::find

```
//Find
std::cout << std::endl;
std::cout << "std::ranges::find() : " << std::endl;
auto odd_n_position = std::ranges::find_if(numbers,odd);

if (odd_n_position != std::end(numbers)) {
    std::cout << "numbers contains at least one odd number : " << *odd_n_position  << std::endl;
} else {
    std::cout << "numbers does not contain any odd number" << std::endl;
}
```

Copy into output stream on the fly

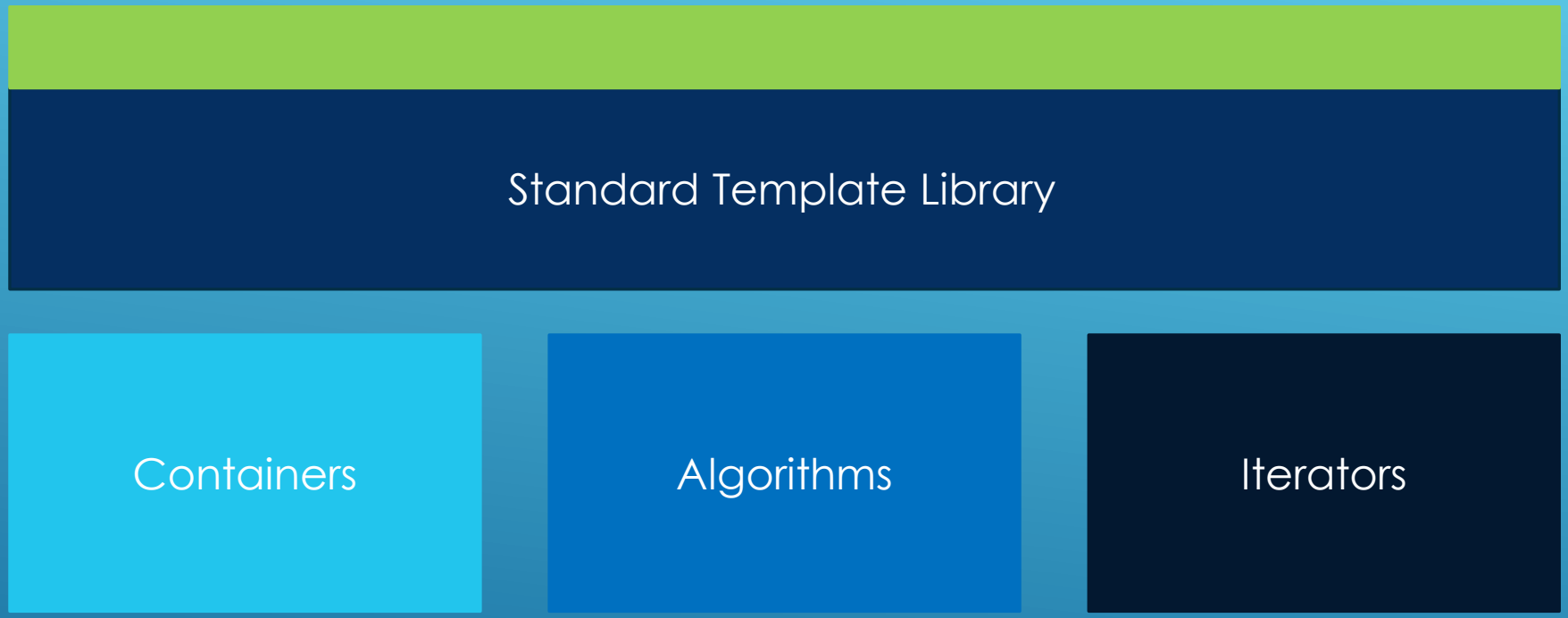
```
//Important, copying into outputstream on the fly
std::cout << std::endl;
std::cout << "numbers : " ;
std::ranges::copy(numbers,std::ostream_iterator<int>(std::cout, " "));
```

More

https://en.cppreference.com/w/cpp/algorithm

Slide intentionally left empty

Constrained Iterator Pair Algorithms


Standard Template Library
Containers
Algorithms
Iterators


Legacy algorithms
Work on iterator pairs
Range algorithms
Work on containers directly

std::all_of [Iterator pair]

```
//std::vector<int> collection{2,6,8,40,64,70};
//std::set<int> collection{2,6,8,40,64,70};
int collection[] {2,6,8,40,64,70};

//std::all_of , lambda function predicate
if (std::all_of(std::begin(collection), std::end(collection), [](int i){ return i % 2 == 0; })) {
    std::cout << "(std::all_of) :  All numbers in collection are even" << std::endl;
}else{
    std::cout << "(std::all_of) : Not all numbers in collection are even" << std::endl;
}
```

std::ranges::all_of

```
std::vector<int> numbers {11,2,6,4,8,3,17,9};
print_collection(numbers);

//std::ranges::all_of()
std::cout << std::endl;
std::cout << "std::ranges::all_of() : " << std::endl;

auto odd = [](int n){
    return n%2 !=0;
};

auto result = std::ranges::all_of(numbers,odd);

if(result){
    std::cout << "All elements in numbers are odd" << std::endl;
}else{
    std::cout << "Not all elements in numbers are odd" << std::endl;
}
```

std::ranges::for_each

```
//For each
std::cout << std::endl;
std::cout << "std::ranges::for_each() : " << std::endl;
print_collection(numbers);
std::ranges::for_each(numbers,[](int& n){n*=2;});
print_collection(numbers);
```

std::ranges::sort

```
//Sort
std::cout << std::endl;
std::cout << "std::ranges::sort() : " << std::endl;
print_collection(numbers);
std::ranges::sort(numbers);
print_collection(numbers);
```

std::ranges::find

```
//Find
std::cout << std::endl;
std::cout << "std::ranges::find() : " << std::endl;
auto odd_n_position = std::ranges::find_if(numbers,odd);

if (odd_n_position != std::end(numbers)) {
    std::cout << "numbers contains at least one odd number : " << *odd_n_position  << std::endl;
} else {
    std::cout << "numbers does not contain any odd number" << std::endl;
}
```

Copy into output stream on the fly

```
//Important, copying into outputstream on the fly
std::cout << std::endl;
std::cout << "numbers : " ;
std::ranges::copy(numbers,std::ostream_iterator<int>(std::cout, " "));
```

Slide intentionally left empty

Constrained Iterator Pair Algorithms

```cpp
//Ranges, iterator pair
auto result = std::ranges::all_of(numbers.begin(),numbers.end(),odd);

if(result){
    std::cout << "All elements in numbers are odd" << std::endl;
}else{
    std::cout << "Not all elements in numbers are odd" << std::endl;
}
```

```cpp
//For each
std::cout << std::endl;
std::cout << "std::ranges::for_each() : " << std::endl;
print_collection(numbers);
std::ranges::for_each(numbers.begin(),numbers.end(),[](int& n){n*=2;});
print_collection(numbers);


//Sort
std::cout << std::endl;
std::cout << "std::ranges::sort() : " << std::endl;
print_collection(numbers);
std::ranges::sort(numbers.begin(),numbers.end());
print_collection(numbers);
```

```cpp
//Find
std::cout << std::endl;
std::cout << "std::ranges::find() : " << std::endl;
auto odd_n_position = std::ranges::find_if(numbers.begin(),numbers.end(),odd);

if (odd_n_position != std::end(numbers)) {
    std::cout << "numbers contains at least one odd number : " << *odd_n_position  << std::endl;
} else {
    std::cout << "numbers does not contain any odd number" << std::endl;
}

//Copying into outputstream on the fly
std::cout << std::endl;
std::cout << "numbers : " ;
std::ranges::copy(numbers.begin(),numbers.end(),std::ostream_iterator<int>(std::cout, " "));
```

std::ranges algorithms should be PREFERED

```
//Why you should prefer std::ranges algorithms from now on
std::list<int> numbers {11,2,6,4,8,3,17,9};
print_collection(numbers);

std::ranges::sort(numbers.begin(),numbers.end());// BAD!!!
```

Slide intentionally left empty

Projections

Some algorithms support projections

Points are ordered by distance by default

```
class Point
{
    friend std::ostream& operator<< (std::ostream& out , const Point& p);
public:
    Point() = default;
    Point(double x, double y) :
        m_x(x), m_y(y){
    }
    //Operators
    bool operator==(const Point& other) const;
    std::partial_ordering operator<=>(const Point& right) const;

private:
    double length() const;    // Function to calculate distance from the point(0,0)
public :
    double m_x{};
    double m_y{};
};

inline std::ostream& operator<< (std::ostream& out , const Point& p){
    out << "Point [ x : " << p.m_x << ", y : " << p.m_y <<
        " , length : " << p.length()    << " ]" ;
    return out;
}
```

Points are ordered by distance by default

```
double Point::length() const{
    return sqrt(pow(m_x - 0, 2) +  pow(m_y - 0, 2) * 1.0);
}

bool Point::operator==(const Point& other) const{
    return (this->length() == other.length());
}

std::partial_ordering Point::operator<=>(const Point& right) const{
    if(length() > right.length())
        return std::partial_ordering::greater;
    else if(length() == right.length())
        return std::partial_ordering::equivalent;
    else if(length() < right.length())
        return std::partial_ordering::less;
    else
        return std::partial_ordering::unordered;
}
```

Sorting without projection

```
//Sorting without projection : uses operator< as is
std::cout << std::endl;
std::cout << "Sorting points (default : based on length) : " << std::endl;
std::vector<Point> points { {10,90} ,{30,70}, {20,80} };

print_collection(points);

//Sorting with the default comparator
std::ranges::sort(points,std::less<>{}); // Default sort based on distance
print_collection(points);
```

Using lambda as a projection

```
//Sorting with a projection : The data is passed into the projection before
//it's passed into the comparator. std::less<> is going to compare two doubles
//instead of comparing two Points.
std::cout << std::endl;
std::cout << "projection on Point::m_x : " << std::endl;
print_collection(points);
std::ranges::sort(points,std::less<>{},[](auto const & p){
    return p.m_x;
});
print_collection(points);
```

Using public member as a projection

```
//Projecting with direct member variable
std::cout << std::endl;
std::cout << "projection on Point::m_y with direct member variables : " << std::endl;
print_collection(points);
std::ranges::sort(points,std::less<>{},&Point::m_y);
print_collection(points);
```

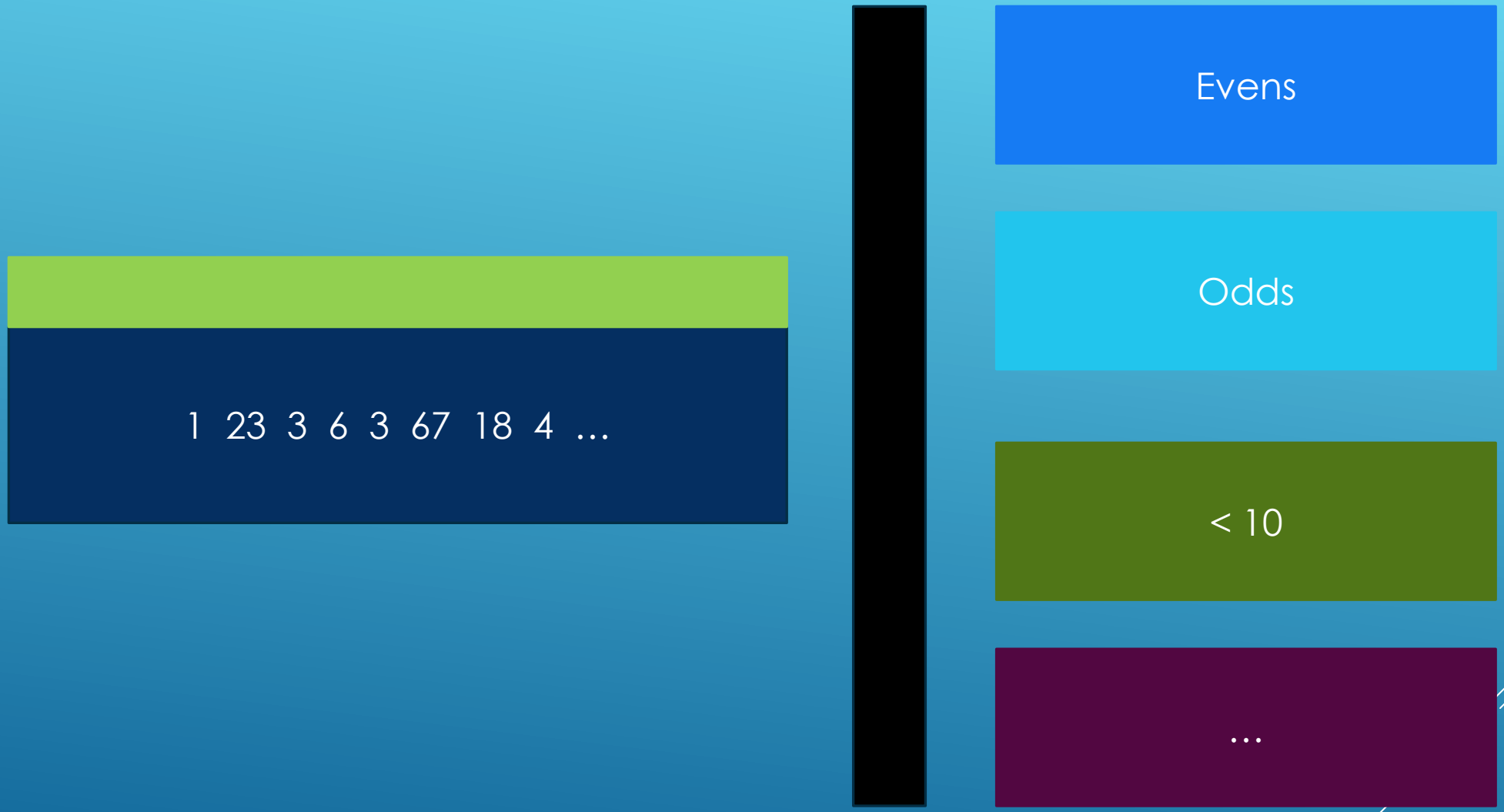
Projections with the for_each algorithm

```
//Projections with for_each
std::cout << std::endl;
std::cout << "Projections with for_each : " << std::endl;

auto print = [](const auto& n) { std::cout << " " << n; };
using pair = std::pair<int, std::string>;
std::vector<pair> pairs{{1,"one"}, {2,"two"}, {3,"tree"}};

std::cout << "project the pair::first: ";
std::ranges::for_each(pairs, print, [](const pair& p) { return p.first; });
std::cout << std::endl;

std::cout << "project the pair::first: ";
std::ranges::for_each(pairs, print, &pair::first);
std::cout << std::endl;

std::cout << "project the pair::second: ";
std::ranges::for_each(pairs, print, [](const pair& p) { return p.second; });
std::cout << std::endl;
```

Slide intentionally left empty

Views and range adaptors

- A view is a non owning range
- It's like a window we can set up to view some real data without setting up the infrastructure to store data
- Views are cheap to copy and pass around as function parameters by design.


1 23 3 6 3 67 18 4 ...
Evens
Odds
< 10
...

Filtering view

```
std::vector<int> vi {1,2,3,4,5,6,7,8,9};

//std::ranges::filter_view
std::cout <<std::endl;
std::cout << "std::ranges::filter_view : " << std::endl;
auto evens = [](int i){
    return (i %2) == 0;
};
std::cout << "vi : " ;
print(vi);
std::ranges::filter_view v_evens = std::ranges::filter_view(vi,evens); //No computation
std::cout << "vi evens : ";
print(v_evens); //Computation happens in the print function
//Print evens on the fly
std::cout << "vi evens : " ;
print(std::ranges::filter_view(vi,evens));
//Print odds on the fly
std::cout << "vi odds : " ;
print(std::ranges::filter_view(vi,[](int i){
    return (i%2)!=0;
}));
```

Filtering view

```
void print(auto  view){
    for(auto i : view){ // Computation happens here.
        std::cout << i << " ";
    }
    std::cout << std::endl;
}
```

Transform_view

```
//std::ranges::transform_view
std::cout <<std::endl;
std::cout << "std::ranges::transform_view : " << std::endl;
std::ranges::transform_view v_transformed = std::ranges::transform_view(vi,[](int i){
    return i * 10;
});
std::cout << "vi : " << std::endl;
print(vi);
std::cout << "vi transformed : " ;
print(v_transformed);
```

```
//std::ranges::take_view
std::cout <<std::endl;
std::cout << "std::ranges::take_view : " << std::endl;
std::ranges::take_view v_taken = std::ranges::take_view(vi,5);
std::cout << "vi : " ;
print(vi);
std::cout << "vi taken : ";
print(v_taken);


//std::ranges::take_while_view : takes elements as long as the predicate condition
//is met
std::cout <<std::endl;
std::cout << "std::views::take_while : " << std::endl;
vi = {1,11,23,131,2,3,4,5,6,7,8,9};
std::ranges::take_while_view v_taken_while = std::ranges::take_while_view(vi,[](int i){
    return (i%2)!=0;
});
std::cout << "vi : ";
print(vi);
std::cout << "vi taken_while : ";
print(v_taken_while);
```

```
//std::ranges::drop_view : drop n first elements
std::cout <<std::endl;
std::cout << "std::ranges::drop_view : " << std::endl;
vi = {1,11,23,131,2,3,4,5,6,7,8,9};
std::ranges::drop_view v_drop = std::ranges::drop_view(vi,5);
std::cout << "vi : ";
print(vi);
std::cout << "vi_drop : ";
print(v_drop);

//std::views::drop_while_view : drops elements as long as the predicate is met
std::cout <<std::endl;
std::cout << "std::ranges::drop_while_view : " << std::endl;
vi = {1,11,23,4,2,3,4,5,6,7,8,9};
std::ranges::drop_while_view v_drop_while = std::ranges::drop_while_view(vi,[](int i){
    return (i%2)!=0;
});
std::cout << "vi : ";
print(vi);
std::cout << "v_drop_while : ";
print(v_drop_while);
```

Compiler errors while constructing some views directly

```
std::cout << std::endl;
using pair = std::pair<int, std::string>;
std::vector<pair> numbers{{1,"one"}, {2,"two"}, {3,"tree"}};

//Compiler error when you build views explicitly. Don't understand why yet
//auto k_view = std::ranges::keys_view(numbers);
//auto v_view = std::ranges::values_view(numbers);

/* ...

auto k_view = std::views::keys(numbers);
auto v_view = std::views::values(numbers);
print(k_view);
print(v_view);
```

Filter range adaptor

```
//Filter range adaptor example
 vi = {1,2,3,4,5,6,7,8,9};

//std::ranges::filter_view
std::cout <<std::endl;
std::cout << "std::views::filter : " << std::endl;
std::cout << "vi : " ;
print(vi);
auto v_evens_1 = std::views::filter(vi,evens); //No computation
std::cout << "vi evens : ";
print(v_evens); //Computation happens in the print function
//Print evens on the fly
std::cout << "vi evens : " ;
print(std::views::filter(vi,evens));
//Print odds on the fly
std::cout << "vi odds : " ;
print(std::views::filter(vi,[](int i){
    return (i%2)!=0;
}));
```

Student type

```
struct Student{
    friend std::ostream& operator<<(std::ostream& out, const Student& s){
        out << "Student [ name : " << s.m_name << ", age : " << s.m_age << "]";
        return out;
    }
    auto operator <=>(const Student& s) const= default;
    std::string m_name;
    unsigned int m_age;
};
```

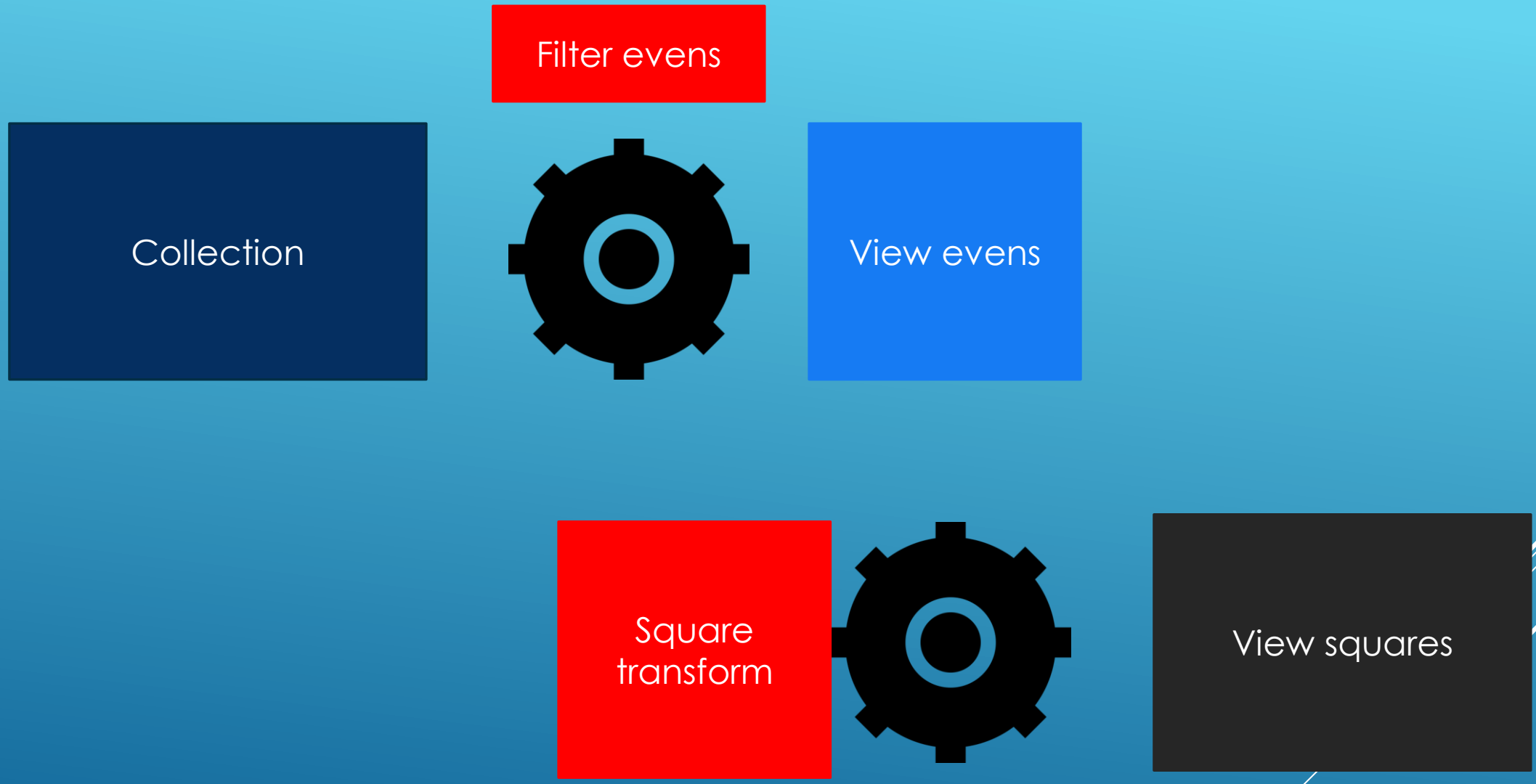
Students in a classroom

```
//Students example
std::cout << std::endl;
std::cout << "students example : " << std::endl;

std::vector<Student> class_room {{"Mike",12},{"John",17},{"Drake",14},{"Mary",16}};
std::cout << std::endl;
std::cout << "classroom : " << std::endl;
for( auto& s : class_room){
    std::cout << "    " <<  s << std::endl;
}

std::ranges::sort(class_room,std::less<>{},&Student::m_age);
std::cout << std::endl;
std::cout << "classroom (after sort) : " << std::endl;
for( auto& s : class_room){
    std::cout << "    " <<  s << std::endl;
}

std::cout << "students under 15 : " ;
print(std::views::take_while(class_room,[](const Student& s){return (s.m_age <15);}));
```

Slide intentionally left empty

View composition and pipe operator

Composing multiple views


Filter evens
Collection
View evens
Square transform
View squares

Raw function composition

```
std::vector<int> vi {1,2,3,4,5,6,7,8,9};

auto even = [](int n){return n%2==0;};
auto my_view =  std::views::transform(std::views::filter(vi,even) ,[](auto  n){return n*=n;});
std::cout << "vi transformed : ";
print(my_view);
```

Pipe operator

```
std::vector<int> vi {1,2,3,4,5,6,7,8,9};

auto even = [](int n){return n%2==0;};
auto my_view =  vi | std::views::filter(even)
                   | std::views::transform([](auto  n){return n*=n;});
std::cout << "vi transformed : ";
print(my_view);
```

Pipe operator

```
//classroom done as map : Keys are sorted by default
//std::unordered_map<std::string,unsigned int> classroom    {
std::map<std::string,unsigned int> classroom    {
                                                    {"John",11},{"Mary",17},
                                                    {"Steve",15},{"Lucy",14},{"Ariel",12}
                                                };

 //Print out the names
 //auto names_view = std::views::keys(classroom);
 auto names_view = classroom | std::views::keys;
 std::cout << "names : ";
 std::ranges::copy(names_view, std::ostream_iterator<std::string>(std::cout, " "));

 //Print out the ages :
 std::cout << std::endl;
 auto ages_view = std::views::values(classroom);
 std::cout << "ages : " ;
 std::ranges::copy(ages_view,std::ostream_iterator<unsigned int>(std::cout," "));
```

```
//Print names in reverse : this doesn't work if you store the data in an
//unordered_map. The reason is the unoredered_map doesn't have reverse iterators,
//that are needed to set up a reverse view.

std::cout << std::endl;
std::cout << "names in reverse : ";
std::ranges::copy(std::views::keys(classroom) | std::views::reverse ,
                  std::ostream_iterator<std::string>(std::cout, " "));
```

```cpp
//Pick names that come before the letter "M" in the alphabet
std::cout << std::endl;
auto before_M = [](const std::string& name){
    return  (static_cast<unsigned char>(name[0]) < static_cast<unsigned char>('M'));
};

std::cout << "names before M : ";
std::ranges::copy(classroom | std::views::keys | std::views::filter(before_M),
          std::ostream_iterator<std::string>(std::cout, " "));
```

Slide intentionally left empty

Range factories

Producing views out of the blue

iota

```
//Generate an infinite sequence of numbers
auto infinite_view = std::views::iota(1); // Stores the compuation
//Numbers are generated lazily, on the fly, as we need them in each iteration
for(auto i : infinite_view){
    std::cout << i << std::endl;
}

//Loop through the view on the fly
for(auto i : std::views::iota(1)){
    std::cout << i << std::endl;
}

//Limit the range : provide an upper limit, upper limit not included.
for(auto i : std::views::iota(1,20)){
    std::cout << i << std::endl;
}

//Limit the range : Use view composition with | operator
for(auto i : std::views::iota(1) | std::views::take(20)){
    std::cout << i << std::endl;
}
```

iota

```
//Raw function composition
for(auto i : std::views::take(std::views::iota(1) , 20)){
    std::cout << i << std::endl;
}
```

Slide intentionally left empty

Ranges Library in C++20 : Summary

- Range algorithms
- Projections
- Views and view adaptors
- Function composition
- Range factories