Stop war
Stop Putin

Belle Views on Ranges with Details and the ~~Devil~~ Shaman

Nicolai M. Josuttis
josuttis.com
@NicoJosuttis

11/22

C++

josuttis | eckstein
IT communication

Stop war
Stop Putin

Nicolai M. Josuttis

- **Independent consultant**
  - Continuously learning since 1962
- **C++:**
  - since 1990
  - ISO Standard Committee since 1997
- **Other Topics:**
  - Systems Architect
  - Technical Manager
  - SOA
  - X and OSF/Motif

@NicoJosuttis

Stop war
Stop Putin

C++20

Stop war
Stop Putin

Init a Range with Random Numbers

C++11

```
template <typename T>
void randomAdd(T& vals)
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution<int> random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);
print(coll);
```

Possible output:

```
84 15 85 76 90 29 99 19
```

Stop war
Stop Putin

Init a Range with Random Numbers

C++17

```
template <typename T>
void randomAdd(T& vals)
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);
print(coll);
```

Possible output:

```
84 15 85 76 90 29 99 19
```


Stop war
Stop Putin

Abbreviated Function Templates

C++20

```
void randomAdd(auto& vals)
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);
print(coll);
```

Possible output:

```
84 15 85 76 90 29 99 19
```

Stop war
Stop Putin

Abbreviated Function Templates

C++20

```
void randomAdd(auto& vals)
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

Some error message in the wild

Possible output:

```
84 15 85 76 90 29 99 19
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);
print(coll);
```

```
std::queue<int> q;
randomAdd(q);      // ERROR
```


Stop war
Stop Putin

Concepts as Type Constraints

C++20

```
void randomAdd(std::ranges::forward_range auto& vals)
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

Possible output:

```
84 15 85 76 90 29 99 19
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);
print(coll);
```

```
std::queue<int> q;
randomAdd(q);      // ERROR: constraint forward_range not satisfied
```

Stop war
Stop Putin

Concepts as Type Constraints

C++20

```
void randomAdd(std::ranges::forward_range auto& vals)
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

Possible output:

```
84 15 85 76 90 29 99 19
tic? tac? toe?
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);
print(coll);
```

```
std::vector<std::string> words{"tic", "tac", "toe"};
randomAdd(words);
print(words);
```


Stop war
Stop Putin

Constraints with `requires`

C++20

```
void randomAdd(std::ranges::forward_range auto& vals)
requires std::integral<typename decltype(vals)::value_type>>
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

Possible output:

```
84 15 85 76 90 29 99 19
tic? tac? toe?
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);    // ERROR
print(coll);
```

`std::vector<int>&::value_type` is not valid

```
std::vector<std::string> words{"tic", "tac", "toe"};
randomAdd(words);    // ERROR
print(words);
```

Stop war
Stop Putin

Constraints with `requires`

C++20

```
void randomAdd(std::ranges::forward_range auto& vals)
requires std::integral<typename std::remove_cvref_t<decltype(vals)>::value_type>
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

Possible output:

```
84 15 85 76 90 29 99 19
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);     // OK
print(coll);

std::vector<std::string> words{"tic", "tac", "toe"};
randomAdd(words);    // ERROR: constraint integral not satisfied
print(words);
```


Stop war
Stop Putin

Multiple Constraints

C++20

```
template <typename RgT>
void randomAdd(RgT& vals)
requires std::ranges::forward_range<RgT> &&
         std::integral<typename RgT::value_type>
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

Possible output:

```
84 15 85 76 90 29 99 19
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);     // OK
print(coll);

std::vector<std::string> words{"tic", "tac", "toe"};
randomAdd(words);    // ERROR: constraint integral not satisfied
print(words);
```

Stop war
Stop Putin

Type Constraints for Template Parameters

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<typename RgT::value_type>
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

Possible output:

```
84 15 85 76 90 29 99 19
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);     // OK
print(coll);

std::vector<std::string> words{"tic", "tac", "toe"};
randomAdd(words);    // ERROR: constraint integral not satisfied
print(words);
```


Stop war
Stop Putin

Raw Arrays as Ranges

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<typename RgT::value_type>
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
     v += random(dre);
  }
}
```

Possible output:

```
84 15 85 76 90 29 99 19
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);     // OK
print(coll);

int data[] = {1, 2, 3};
randomAdd(data);     // ERROR: no member value_type for raw arrays
```

Stop war
Stop Putin

Using Ranges Utilities

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  std::default_random_engine dre{std::random_device{}()};
  std::uniform_int_distribution random{1,99};
  for (auto& v : vals) {   // add random value to each elem
    v += random(dre);
  }
}
```

```
std::vector<int> coll(8);   // 8 elems with value 0
randomAdd(coll);    // OK
print(coll);
```

```
int data[] = {1, 2, 3};
randomAdd(data);    // OK
```

Possible output:

```
84 15 85 76 90 29 99 19
30 86 65
```


Stop war
Stop Putin

Threads

C++11/C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  auto init = [&] {
    std::default_random_engine dre{std::random_device{}()};
    std::uniform_int_distribution random{1,99};
    for (auto& v : vals) {   // add random value to each elem
      v += random(dre);
    }
  };
  std::vector<std::thread> threads;
  for (int i = 0; i < 5; ++i) {
    threads.push_back(std::thread{init});
  }
  ...
  for (auto& t : threads) {
    t.join();
  }
}
```

Several Issues:

- Concurrent writes
- Core dump without `join()`
  - Needs exceptions handling
  - How to stop the threads?

Stop war
Stop Putin

Joining Threads

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  auto init = [&] {
                std::default_random_engine dre{std::random_device{}()};
                std::uniform_int_distribution random{1,99};
                for (auto& v : vals) {   // add random value to each elem
                  v += random(dre);
                }
              };
  std::vector<std::jthread> threads;
  for (int i = 0; i < 5; ++i) {
    threads.push_back(std::jthread{init});
  }
  ...
  for (auto& t : threads) {   // implicit by destructor
    t.join();
  }
}
```

17

@NicoJosuttis

Stop war
Stop Putin

Joining Threads and Stop Tokens

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  auto init = [&] (std::stop_token st) {
                std::default_random_engine dre{std::random_device{}()};
                std::uniform_int_distribution random{1,99};
                for (auto& v : vals) {   // add random value to each elem
                  if (st.stop_requested()) return;
                  v += random(dre);
                }
              };
  std::vector<std::jthread> threads;
  for (int i = 0; i < 5; ++i) {
    threads.push_back(std::jthread{init});
  }
  ...
  for (auto& t : threads) {   // implicit by destructor
    t.request_stop();
    t.join();
  }
}
```

18

@NicoJosuttis

Stop war
Stop Putin

Joining Threads and Stop Tokens

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  auto init = [&] (std::stop_token st) {
                std::default_random_engine dre{std::random_device{}()};
                std::uniform_int_distribution random{1,99};
                for (auto& v : vals) {   // add random value to each elem
                  if (st.stop_requested()) return;
                  v += random(dre);
                }
              };
  std::vector<std::jthread> threads;
  for (int i = 0; i < 5; ++i) {
    threads.push_back(std::jthread{init});
  }
}
```


Stop war
Stop Putin

Atomic References

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  auto init = [&] (std::stop_token st) {
                std::default_random_engine dre{std::random_device{}()};
                std::uniform_int_distribution random{1,99};
                for (auto& v : vals) {   // add random value to each elem
                  if (st.stop_requested()) return;
                  std::atomic_ref{v} += random(dre);
                }
              };
  std::vector<std::jthread> threads;
  for (int i = 0; i < 5; ++i) {
    threads.push_back(std::jthread{init});
  }
}
```

v is atomic
only in this context

Stop war
Stop Putin

Init a Range with Random Numbers

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  auto init = [&] (std::stop_token st) {
                std::default_random_engine dre{std::random_device{}()};
                std::uniform_int_distribution random{1,99};
                for (auto& v : vals) {   // add random value to each elem
                  if (st.stop_requested()) return;
                  std::atomic_ref{v} += random(dre);
                }
              };
  std::vector<std::jthread> threads;
  for (int i = 0; i < 5; ++i) {
    threads.push_back(std::jthread{init});
  }
}

std::vector<int> coll coll(8);   // 8 elems
randomAdd(coll);
print(coll);
```

Possible output:

```
285 219 219 314 222 155 304 207
```


Stop war
Stop Putin

C++20 Views

Stop war
Stop Putin

Init a Range with Random Numbers

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  ...
}
```

```
std::vector<int> vec1(8);   // 8 elems
randomAdd(vec1);
print(vec1);
```

Possible output:

```
285 219 219 314 222 155 304 207
```

23 @NicoJosuttis

Stop war
Stop Putin

Init a Range with Random Numbers

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  ...
}
```

```
std::vector<int> vec1(8);   // 8 elems
randomAdd(vec1);
print(vec1);
```

Possible output:

```
285 219 219 314 222 155 304 207
```

```
std::vector<int> vec2(8);   // 8 elems
auto v2 = vec2 | std::views::take(5);
randomAdd(v2);
print(vec2);
```

```
251 267 358 159 266 0 0 0
```

```
std::vector<int> vec3(8);   // 8 elems
int skip = 2;
auto v3 = vec3 | std::views::drop(skip) | std::views::take(5);
randomAdd(v3);
print(vec3);
```

```
0 0 145 161 240 108 123 0
```

24 @NicoJosuttis

Stop war
Stop Putin

Init a Range with Random Numbers

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  ...
}
```

```
std::vector<int> vec1(8);   // 8 elems
randomAdd(vec1);
print(vec1);

std::vector<int> vec2(8);   // 8 elems
auto v2 = vec2 | std::views::take(5);
randomAdd(v2);
print(lst2);

std::list<int> lst3(8);   // 8 elems
int skip = 2;
auto v3 = lst3 | std::views::drop(skip) | std::views::take(5);
randomAdd(v3);
print(lst3);
```

Possible output:

```
285 219 219 314 222 155 304 207
251 267 358 159 266 0 0 0
0 0 145 161 240 108 123 0
```

✓ how cool

25

@NicoJosuttis

Stop war
Stop Putin

Feel the Views

- Implement `print()`
- Call `randomAdd()` with filter
- Print number of elems inside `randomAdd()`

C++

josuttis | eckstein
IT communication

Stop war
Stop Putin

Member Functions of Views

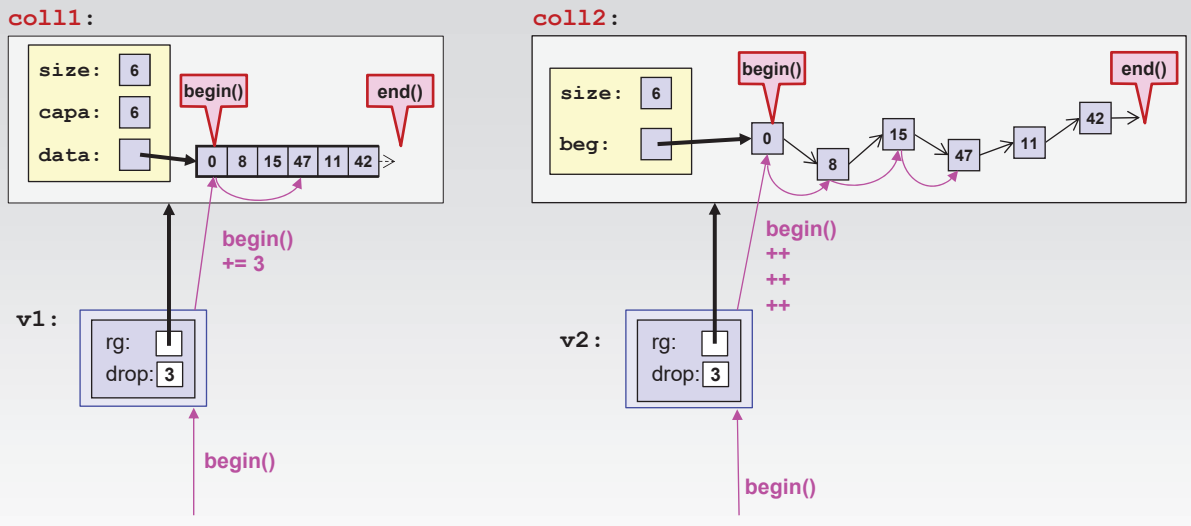
C++

josuttis | eckstein
IT communication

Stop war
Stop Putin

How Expensive is `begin()` ?

C++20



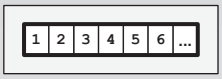

```
std::vector<int> vec{0, 8, 15, 47, 11, 42};
auto v1 = vec | std::views::drop(3);
auto pos = v1.begin();

std::list<int> lst, 8, 15, 47, 11, 42};
auto v2 = lst | std::views::drop(3);
auto pos = v2.begin();
```

28
@NicoJosuttis

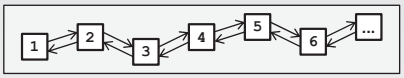
Stop war
Stop Putin

Member Functions of Views

C++20




```
std::vector vec{1, 2, 3, 4, 5, 6, ...};
auto vVec = vec | std::views::drop(n);
vVec.begin()          // fast: vec.begin() + n
vVec.empty()          // fast: vec.size() <= n
vVec.size()           // fast: n >= vec.size() ? 0 : vec.size() - n
vVec[idx]             // fast: vec[idx - n]
```



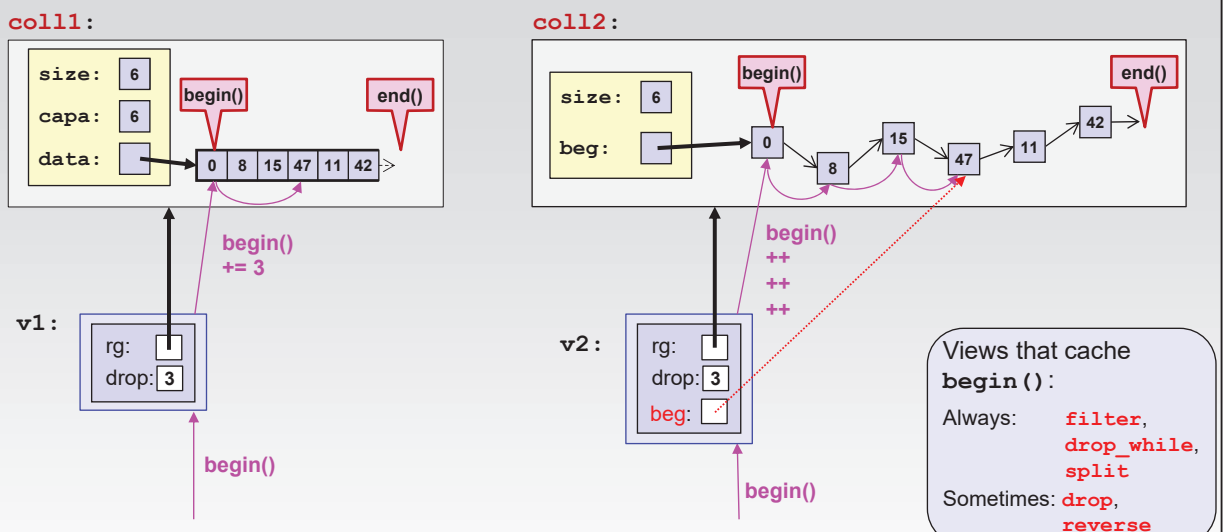

```
std::list lst{1, 2, 3, 4, 5, 6, ...};
auto vLst = lst | std::views::drop(n);
vLst.begin()          // slow: lst.begin() and n times ++
vLst.empty()          // fast:  lst.size() <= n
vLst.size()           // fast:  n >= lst.size() ? 0 : lst.size() - n
~~vLst[idx]~~         // slow: lst.begin() and idx times ++

auto vFlt = coll | std::views::filter(pred);
vFlt.begin()          // slow: pred for all elements until first true
vFlt.empty()          // slow: pred for all elements until first true
~~vFlt.size()~~       // slow: pred for all elements
~~vFlt[idx]~~         // slow: pred for all elements until idx times true
```


@NicoJosuttis

Stop war
Stop Putin

Views that Cache `begin()`

C++20




```
std::vector<int> vec{0, 8, 15, 47, 11, 42};
auto v1 = vec | std::views::drop(3);
auto pos = v1.begin();

std::list<int> lst{0, 8, 15, 47, 11, 42};
auto v2 = lst | std::views::drop(3);
auto pos = v2.begin();
```


@NicoJosuttis

Stop war
Stop Putin

Member Functions of Views

C++20

	1st `begin()`	2nd `begin()`	`size()`	1st `empty()`	2nd `empty()`
`std::vector vec`	constant	constant	constant	constant	constant
`std::list lst`	constant	constant	constant	constant	constant
`vec | drop(n)`	constant	constant	constant	constant	constant
`lst | drop(n)`	linear	constant	constant	constant[1]	constant
`vec | filter(...)`	linear	constant	--	linear	constant
`lst | filter(...)`	linear	constant	--	linear	constant
`vec | filter(...) | drop(n)`	linear	constant	--	linear	constant
`lst | filter(...) | drop(n)`	linear	constant	--	linear	constant

[1]: linear with g++

"First linear, then constant" is called "*amortized constant*" in the C++ standard

[range.range]:
Given an expression `t` such that `decltype((t))` is `T&`,
`T` models range only if
...
(3.2) — both `ranges::begin(t)` and `ranges::end(t)` are **amortized constant time** and non-modifying,
...

31

@NicoJosuttis

Stop war
Stop Putin

Feel the Complexity

- **Measure with `drop()`**

C++

josuttis | eckstein
IT communication

Stop war
Stop Putin

Using Views in Practice

C++

josuttis | eckstein
IT communication

Stop war
Stop Putin

Processing Containers and Views

C++20

```
void print(const auto& coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}
```

```
std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::list<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);
print(coll2);
print(coll1 | std::views::take(3));          // print first three elements
print(coll2 | std::views::take(3));          // print first three elements
print(coll1 | std::views::drop(3));          // print fourth to last element
print(coll2 | std::views::drop(3));          // Compile-time ERROR
for (int v : coll2 | std::views::drop(3)) {  // OK: print fourth to last element
   std::cout << v << ' ';
}
```

Output:

```
0 8 15 47 11 42
0 8 15 47 11 42
0 8 15
0 8 15
47 11 42
ERROR
47 11 42
```


@NicoJosuttis

Stop war
Stop Putin

Processing Containers and Views

C++20

```
void print(const auto& coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}
```

```
std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::list<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);
print(coll2);
print(coll1 | std::views::take(3));          // print first three elements
print(coll2 | std::views::take(3));          // print first three elements
print(coll1 | std::views::drop(3));          // print fourth to last element
print(coll2 | std::views::drop(3));          // Compile-time ERROR
auto v = coll2 | std::views::drop(3);
print(std::ranges::subrange{v.begin(), v.end()});   // OK
```

Output:

```
0 8 15 47 11 42
0 8 15 47 11 42
0 8 15
0 8 15
47 11 42
ERROR
47 11 42
```

35

@NicoJosuttis

Stop war
Stop Putin

Passing Containers and Views by Universal Reference

C++20

```
void print(auto&& coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}
```

Universal (or *forwarding*) ***reference***
- Can universally refer to every expression (even temporaries/rvalues) without making it `const`

```
std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::list<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);
print(coll2);
print(coll1 | std::views::take(3));          // print first three elements
print(coll2 | std::views::take(3));          // print first three elements
print(coll1 | std::views::drop(3));          // print fourth to last element
print(coll2 | std::views::drop(3));          // OK: print fourth to last element
for (int v : coll2 | std::views::drop(3)) {  // OK: print fourth to last element
   std::cout << v << ' ';
}
```

Output:

```
0 8 15 47 11 42
0 8 15 47 11 42
0 8 15
0 8 15
47 11 42
47 11 42
47 11 42
```

36

@NicoJosuttis

Stop war
Stop Putin

Concurrent Iterations over Views

C++20

```
auto printAndSum(auto&& rg) {
  // one thread prints the elements:
  std::jthread printThread{[&] {
    for (const auto& elem : rg) {
      std::cout << elem << ' ';
    }
    std::cout << '\n';
  }};
  // this thread computes the sum of the element values:
  return std::accumulate(rg.begin(), rg.end(),
                         0L);
}
```

Use `const auto&` in this case

Concurrent read iterations
cause undefined behavior
- Concurrent `begin()` and `end()` only safe for containers

```
std::list<int> coll{0, 8, 15, 47, 11, 42};

auto sum1 = printAndSum(coll);                          // OK

auto sum2 = printAndSum(coll | std::views::drop(2));    // undefined behavior
```


Stop war
Stop Putin

Passing Containers and Views by Value

C++20

```
void print(auto coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}
```

```
std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::list<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);                                   // expensive
print(coll2);                                   // expensive
print(coll1 | std::views::take(3));             // print first three elements
print(coll2 | std::views::take(3));             // print first three elements
print(coll1 | std::views::drop(3));             // print fourth to last element
print(coll2 | std::views::drop(3));             // OK: print fourth to last element
for (int v : coll2 | std::views::drop(3)) {     // OK: print fourth to last element
   std::cout << v << ' ';
}
```

Output:

```
0 8 15 47 11 42
0 8 15 47 11 42
0 8 15
0 8 15
47 11 42
47 11 42
47 11 42
```

Stop war
Stop Putin

Accepting Only Views by Value

C++20

```
void print(std::ranges::view auto coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}

std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::list<int> coll2{0, 8, 15, 47, 11, 42};

print(coll1);                                  // ERROR
print(coll2);                                  // ERROR
print(coll1 | std::views::take(3));            // print first three elements
print(coll2 | std::views::take(3));            // print first three elements
print(coll1 | std::views::drop(3));            // print fourth to last element
print(coll2 | std::views::drop(3));            // OK: print fourth to last element
for (int v : coll2 | std::views::drop(3)) {    // OK: print fourth to last element
   std::cout << v << ' ';
}
```

Output:

```
0 8 15 47 11 42
0 8 15 47 11 42
0 8 15
0 8 15
47 11 42
47 11 42
47 11 42
```


Stop war
Stop Putin

Accepting Only Views by Value

C++20

```
void print(std::ranges::view auto coll) {
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}

std::vector<int> coll1{0, 8, 15, 47, 11, 42};
std::list<int> coll2{0, 8, 15, 47, 11, 42};

print(std::views::all(coll1));                 // cheap
print(std::views::all(coll2));                 // cheap
print(coll1 | std::views::take(3));            // print first three elements
print(coll2 | std::views::take(3));            // print first three elements
print(coll1 | std::views::drop(3));            // print fourth to last element
print(coll2 | std::views::drop(3));            // OK: print fourth to last element
for (int v : coll2 | std::views::drop(3)) {    // OK: print fourth to last element
   std::cout << v << ' ';
}
```

Output:

```
0 8 15 47 11 42
0 8 15 47 11 42
0 8 15
0 8 15
47 11 42
47 11 42
47 11 42
```


@NicoJosuttis

Stop war
Stop Putin

Overloading for Containers and Views

C++20

```
void print(std::ranges::view auto coll) { // for views only
  for (const auto& elem : coll) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}

template<typename T>
void print(const T& coll)                 // not for views
requires (!std::ranges::view<T>)
{
  print(std::views::all(coll));
}
```

Necessary to avoid ambiguities when views are passed

```
std::vector<int> vec{1, 2, 3, 4, 5, 6, 7, 8, 9};
print(vec);                               // OK, calls 2nd print()
print(vec | std::views::take(3));         // OK, calls 1st print()

print(getColl());                         // OK, calls 2nd print()
print(getColl() | std::views::take(3));   // OK, calls 1st print()
```

41

@NicoJosuttis

Stop war
Stop Putin

Bring Back Constness

C++

josuttis | eckstein
IT communication

Stop war
Stop Putin

cbegin()

C++20

C++20

```
template<typename T>
void print(T&& coll) {
  auto pos = coll.cbegin();              // OOPS: not available for views
  std::cout << *pos;

}
```

```
std::vector vec{1, 2, 3, 4, 5};
print(vec);                              // OK
print(vec | std::views::drop(3));        // ERROR
print(vec | std::views::filter(...));    // ERROR

std::list lst{1, 2, 3, 4, 5};
print(lst | std::views::drop(3));        // ERROR
```


Stop war
Stop Putin

cbegin()

C++20

C++20

```
template<typename T>
void print(T&& coll) {
  auto pos = coll.cbegin();
  *std::cbegin(coll) = 0;                // OOPS: modifies first elem of views
  *std::ranges::cbegin(coll) = 0;        // OOPS: modifies first elem of views
}
```

```
std::vector vec{1, 2, 3, 4, 5};
print(vec);                              // Comile-time ERROR
print(vec | std::views::drop(3));        // OOPS: compiles and modifies coll
print(vec | std::views::filter(...));    // OOPS: compiles and modifies coll

std::list lst{1, 2, 3, 4, 5};
print(lst | std::views::drop(3));        // OOPS: compiles and modifies coll
```

Stop war
Stop Putin

cbegin()

C++23

Don't use `std::cbegin()` since C++20
- **Prefer `std::ranges::` over `std::`**

C++23

```
template<typename T>
void print(T&& coll) {
  auto pos = coll.cbegin();              // OK: available since C++23
  *std::cbegin(coll) = 0;                // OOPS: still broken
  *std::ranges::cbegin(coll) = 0;        // ERROR
}

std::vector vec{1, 2, 3, 4, 5};
print(vec);                              // ERROR
print(vec | std::views::drop(3));        // ERROR(unless only std::begin() used)
print(vec | std::views::filter(...));    // ERROR (unless only std::begin() used)
std::list lst{1, 2, 3, 4, 5};
print(lst | std::views::drop(3));        // ERROR (unless only std::begin() used)
```


Stop war
Stop Putin

Declare Elements `const`

C++20

```
void print(auto&& coll)
{
  for (const auto& elem : coll) {
    std::cout << elem << ' ';    // can't modify elem here
  }
  std::cout << '\n';
}
```

Stop war
Stop Putin

zip_view

C++23

```
void printPairs(const auto& coll)
{
  for (const auto& elem : coll) {
    std::cout << elem.first << ':' << elem.second << '\n';
  }
}
```

```
std::vector v1{1, 2, 3};
std::vector v2{10, 20, 30};
printPairs(std::views::zip(v1, v2));
```

Output:

```
1:10
2:20
3:30
```


Stop war
Stop Putin

zip_view

C++23

```
void printPairs(const auto& coll)
{
  for (const auto& elem : coll) {
    if (elem.first == 2) {
      std::cout << "* ";
    }
    std::cout << elem.first << ':' << elem.second << '\n';
  }
}
```

```
std::vector v1{1, 2, 3};
std::vector v2{10, 20, 30};
printPairs(std::views::zip(v1, v2));
```

Output:

```
1:10
* 2:20
3:30
```

Stop war
Stop Putin

zip_view

C++23

```
void printPairs(const auto& coll)
{
  for (const auto& elem : coll) {
    if (elem.first = 2) {
      std::cout << "* ";
    }
    std::cout << elem.first << ':' << elem.second << '\n';
  }
}
```

Some views ignore this **const**

Some views ignore this **const**

```
std::vector v1{1, 2, 3};
std::vector v2{10, 20, 30};
printPairs(std::views::zip(v1, v2));
```

Output:

```
* 2:10
* 2:20
* 2:30
```


Stop war
Stop Putin

const Propagation

Stop war
Stop Putin

const Propagation is Broken for Views

C++20

- **Views do not propagate const**
  - *If they **refer*** to ranges (and not own them)

```
template<typename T>
void print(const T& coll) {
  *coll.begin() = 0;      // OOPS: OK if referencing view passed
  coll.front() = 0;       // OOPS: OK if referencing view passed
  if (coll[0] = 0) {      // OOPS: OK if referencing view passed
    …
  }
}

std::vector vec{1, 2, 3, 4, 5};
print(vec);                                  // ERROR (good)
print(vec | std::views::drop(3));            // OOPS: compiles and modifies vec
print(getColl() | std::views::drop(3));      // ERROR (good)

print(std::as_const(vec) | std::views::take(5));          // ERROR (good)
print(vec | std::views::take(5) | std::views::as_const); // ERROR (C++23)
```


Stop war
Stop Putin

Exact Type of Views

C++20

- **ref_view when referring to named objects (lvalues)**
- **owning_view when referring to temporaries (rvalues)**
  - Added with fix against C++20 with wg21.link/**P2415**

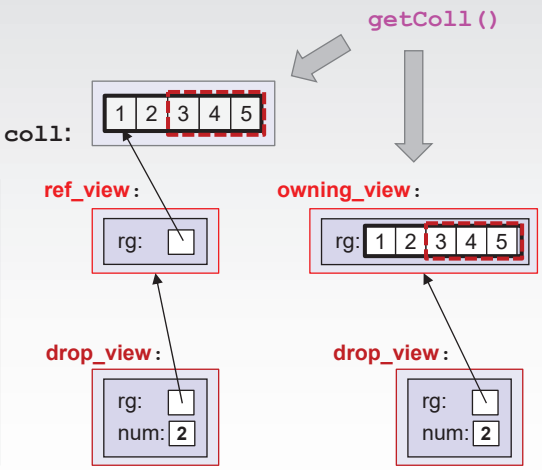
```
std::vector<int> getColl()
{
  return {1, 2, 3, 4, 5};
}
```

```
auto coll = getColl();
auto v1 = coll | std::views::drop(2);
```

drop_view< ref_view< vector<int> > >

```
auto v2 = getColl() | std::views::drop(2);
```

drop_view< owning_view< vector<int> > >



Stop war
Stop Putin

Views Kill `const`

- **Before C++ views:**

```
template <typename T>
void print(const T& coll)
{
  for (const auto& elem : coll) {
    ...
  }
}
```

- **Since C++ views:**

```
template <typename T>
void print(T&& coll)
{
  for (const auto& elem : std::views::as_const(coll)) {
    ...
  }
}
```

Don't use `std::as_const(coll)`
It will compile, but have no effect

Stop war
Stop Putin

Dealing with Containers and Views with C++23

C++23

```
template<typename T>
void print(T&& coll) {
{
  if constexpr (std::ranges::const_range<T>) {
    // the passed range and its elements are constant
    for (const auto& elem : coll) {
      std::cout << elem << ' ';
    }
    std::cout << '\n';
  }
  else {
    // call this function again after making elements const:
    print(std::views::as_const(std::forward<T>(coll)));
  }
}
```

```
std::vector<int> vec{1, 2, 3, 4, 5, 6, 7, 8, 9};
print(vec);                              // OK, calls 2nd print()
print(vec | std::views::take(3));        // OK, calls 1st print()
```

Stop war
Stop Putin

Modifications

C++

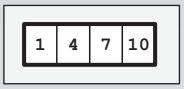
josuttis | eckstein
IT communication

Stop war
Stop Putin

Using the Filter View

C++20

```
std::vector<int> coll{1, 4, 7, 10};
print(coll);

auto isEven = [] (auto&& i) { return i % 2 == 0; };
auto collEven = coll | std::views::filter(isEven);

// add 2 to even elements:
for (int& i : collEven) {
  i += 2;
}
print(coll);

// add 2 to even elements:
for (int& i : collEven) {
  i += 2;
}
print(coll);
```



Output:

```
1 4 7 10
1 6 7 12
1 8 7 14
```

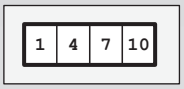

@NicoJosuttis

Stop war
Stop Putin

Using the Filter View

C++20

```
std::vector<int> coll{1, 4, 7, 10};
print(coll);

auto isEven = [] (auto&& i) { return i % 2 == 0; };
auto collEven = coll | std::views::filter(isEven);

// increment even elements:
for (int& i : collEven) {
  i += 1;          // Runtime Error: UB: predicate broken
}
print(coll);

// increment even elements:
for (int& i : collEven) {
  i += 1;          // Runtime Error: UB: predicate broken
}
print(coll);
```



Output:

```
1 4 7 10
1 5 7 11
1 6 7 11
```


Stop war
Stop Putin

Using the Filter View

- **Main use case of a filter:**
  - Fix an attribute that some elements might have

has undefined behavior:

[range.filter.iterator]:
Modification of the element a filter_view::*iterator* denotes is permitted, but results in undefined behavior if the resulting value does not satisfy the filter predicate.

```
// as a shaman:
for (auto& m : monsters | std::views::filter(isDead)) {
  m.resurrect();   // undefined behavior: because no longer dead
  m.burn();        // OK (because it is still dead)
}
```

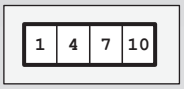
Thanks to Patrice Roy for this example

Stop war
Stop Putin

Using the Filter View

C++20

```
std::vector<int> coll{1, 4, 7, 10};
print(coll);

auto isEven = [] (auto&& i) { return i % 2 == 0; };
```




```
// increment even elements:
for (int& i : coll | std::views::filter(isEven)) {
  i += 1;          // UB: but works
}
print(coll);

// increment even elements:
for (int& i : coll | std::views::filter(isEven)) {
  i += 1;          // UB: but works
}
print(coll);
```

Output:

```
1 4 7 10
1 5 7 11
1 5 7 11
```

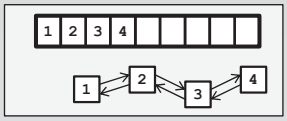
Use (and reuse) views ad hoc

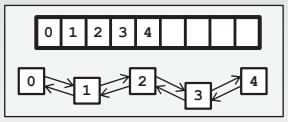
Stop war
Stop Putin

Modifications Considered Harmful with Caching

C++20

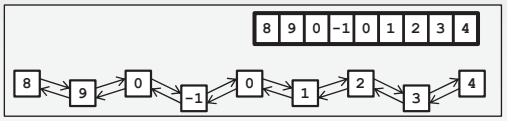
```
std::vector vec{1, 2, 3, 4};
vec.reserve(9);
std::list lst{1, 2, 3, 4};
auto vVec = vec | std::views::drop(2);
auto vLst = lst | std::views::drop(2);
```






```
2 3 4
2 3 4
```

```
// insert new elements at the front:
vec.insert(vec.begin(), 0);
lst.insert(lst.begin(), 0);
print(vVec, vLst);
```




```
vec.insert(vec.begin(), {8, 9, 0, -1});
lst.insert(lst.begin(), {8, 9, 0, -1});
print(vVec, vLst);
```

```
0 -1 0 1 2 3 4
2 3 4
```

```
auto vVecB = vVec;
auto vLstB = vLst;
print(vVecB, vLstB);
```

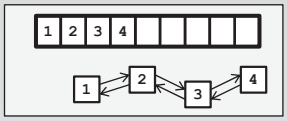
```
0 -1 0 1 2 3 4
0 -1 0 1 2 3 4
```

Stop war
Stop Putin

Modifications Considered Harmful with Caching

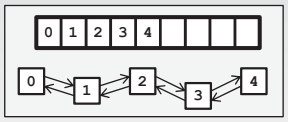
C++20

```
std::vector vec{1, 2, 3, 4};
vec.reserve(9);
std::list lst{1, 2, 3, 4};
auto vVec = vec | std::views::drop(2);
auto vLst = lst | std::views::drop(2);
print(vVec, vLst);
```



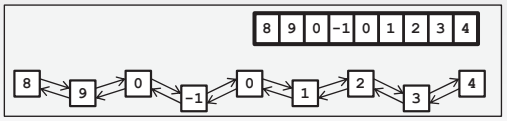

```
3 4
3 4
```

```
// insert new elements at the front:
vec.insert(vec.begin(), 0);
lst.insert(lst.begin(), 0);
print(vVec, vLst);
```




```
2 3 4
3 4
```

```
vec.insert(vec.begin(), {8, 9, 0, -1});
lst.insert(lst.begin(), {8, 9, 0, -1});
print(vVec, vLst);
```




```
0 -1 0 1 2 3 4
3 4
```

```
auto vVecB = vVec;
auto vLstB = vLst;
print(vVecB, vLstB);
```

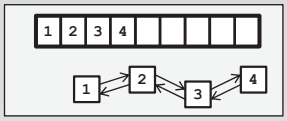
```
0 -1 0 1 2 3 4
0 -1 0 1 2 3 4
```


Stop war
Stop Putin

Modifications Considered Harmful with Caching

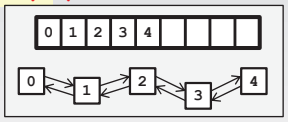
C++20

```
std::vector vec{1, 2, 3, 4};
vec.reserve(9);
std::list lst{1, 2, 3, 4};
auto gt0 = [](const auto& v){return v > 0;};
auto vVec = vec | std::views::filter(gt0) | std::views::drop(2);
auto vLst = lst | std::views::filter(gt0) | std::views::drop(2);
print(vVec, vLst);
```



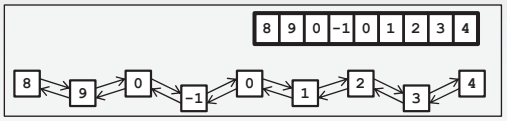

```
3 4
3 4
```

```
// insert new elements at the front:
vec.insert(vec.begin(), 0);
lst.insert(lst.begin(), 0);
print(vVec, vLst);
```




```
2 3 4
3 4
```

```
vec.insert(vec.begin(), {8, 9, 0, -1});
lst.insert(lst.begin(), {8, 9, 0, -1});
print(vVec, vLst);
```




```
0 1 2 3 4
3 4
```

```
auto vVecB = vVec;
auto vLstB = vLst;
print(vVecB, vLstB);
```

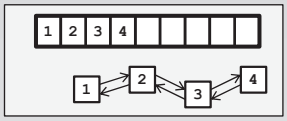
```
1 2 3 4
1 2 3 4
```

Stop war
Stop Putin

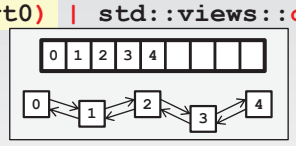
Modifications Considered Harmful with Caching

C++20

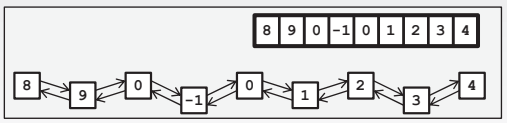
```
std::vector vec{1, 2, 3, 4};
vec.reserve(9);
std::list lst{1, 2, 3, 4};
auto gt0 = [](const auto& v){return v > 0;};
auto vVec = vec | std::views::filter(gt0) | std::views::drop(2);
auto vLst = lst | std::views::filter(gt0) | std::views::drop(2);
print(vVec, vLst);

// insert new elements at the front:
vec.insert(vec.begin(), 0);
lst.insert(lst.begin(), 0);
print(vVec, vLst);

vec.insert(vec.begin(), {8, 9, 0, -1});
lst.insert(lst.begin(), {8, 9, 0, -1});
print(vVec, vLst);

auto vVecB = vVec;
auto vLstB = vLst;
print(vVecB, vLstB);
```



3 4
3 4



undefined behavior
3 4



undefined behavior
3 4

Usually: 1 2 3 4

undefined behavior
1 2 3 4

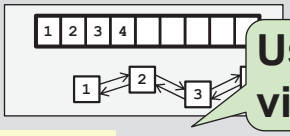
@NicoJosuttis

Stop war
Stop Putin

Modifications Considered Harmful with Caching

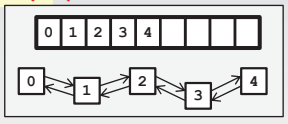
C++20

```
std::vector vec{1, 2, 3, 4};
vec.reserve(9);
std::list lst{1, 2, 3, 4};
auto gt0 = [](const auto& v){return v > 0;};
auto vVec = vec | std::views::filter(gt0) | std::views::drop(2);
auto vLst = lst | std::views::filter(gt0) | std::views::drop(2);
print(vVec, vLst);

// insert new elements at the front:
vec.insert(vec.begin(), 0);
lst.insert(lst.begin(), 0);
print(vVec, vLst);

vec.insert(vec.begin(), {8, 9, 0, -1});
lst.insert(lst.begin(), {8, 9, 0, -1});
print(vVec, vLst);

auto vVecB = vVec;
auto vLstB = vLst;
print(vVecB, vLstB);
```

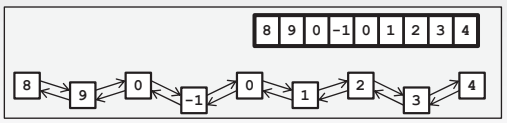


Use (and reuse) views ad hoc

3 4



undefined behavior
3 4



undefined behavior
3 4

Usually: 1 2 3 4

undefined behavior
1 2 3 4

@NicoJosuttis

Stop war
Stop Putin

Standard Views

C++

josuttis | eckstein
IT communication

Stop war
Stop Putin

Basic Idioms for Containers and Arrays

- You can iterate if the range is `const` Broken
- A read does not change behavior Broken
- `const` makes elements immutable Broken
- `cbegin()` makes elements immutable Broken
- Concurrent iterations are safe Broken
- A copy of a range has the same state Broken

@NicoJosuttis

Stop war
Stop Putin

Init a Range with Random Numbers

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range_value_t<RgT>>
{
  ...
}
```

```
std::vector<int> vec1(8);   // 8 elems
randomAdd(vec1);
print(vec1);

std::vector<int> vec2(8);   // 8 elems
auto v2 = vec2 | std::views::take(5);
randomAdd(v2);
print(lst2);

std::list<int> lst3(8);   // 8 elems
int skip = 2;
auto v3 = lst3 | std::views::drop(skip) | std::views::take(5);
randomAdd(v3);
print(lst3);
```

Possible output:

```
285 219 219 314 222 155 304 207
251 267 358 159 266 0 0 0
0 0 145 161 240 108 123 0
```

✓ how cool

67

@NicoJosuttis

Stop war
Stop Putin

Init a Range with Random Numbers

C++20

```
template <std::ranges::forward_range RgT>
void randomAdd(RgT& vals)
requires std::integral<std::ranges::range
{
  ...
}
```

Undefined Behavior for v2 due to concurrent iterations

```
std::vector<int> vec1(8);   // 8 elems
randomAdd(vec1);
print(vec1);

std::vector<int> vec2(8);   // 8 elems
auto v2 = vec2 | std::views::take(5);
randomAdd(v2);
print(lst2);

std::list<int> lst3(8);   // 8 elems
int skip = 2;
auto v3 = lst3 | std::views::drop(skip) | std::views::take(5);
randomAdd(v3);
print(lst3);
```

Possible output:

```
285 219 219 314 222 155 304 207
251 267 358 159 266 0 0 0
0 0 145 161 240 108 123 0
```

✓ how cool

68

@NicoJosuttis

Stop war
Stop Putin

Design of Standard Views

- **C++ views break several fundamental idioms**
  - Iterations are not stateless
  - `const` is *sometimes* not propagated
  - Only to benefit from caching
    - Even though the benefit is gone with the consequences of caching
- **C++ views have a technical instead of semantical API**
  - Yes, technically a pointer, but semantically a collection
- **That creates**
  - Incredible confusion
  - Unnecessary undefined behavior
  - Compromises of `const`
  - Really bad and ugly workarounds
  - Frustration

69 @NicoJosuttis

Stop war
Stop Putin

C++ Standard Views are Broken

for Ordinary Programmers

70 @NicoJosuttis

Belleviews

C++ Views that Just Work

71 @NicoJosuttis

Stop war
Stop Putin

Belleviews Goals

- **Usability**
  - Views should just work
  - Simplicity
  - Consistency
- **Safety**
  - No non-intuitive undefined behavior
- **Predictability**
  - No breakage of standard idioms
  - Common use cases should work as expected
- **Performance**
  - Caching can explicitly requested (e.g. by the **eager_begin** view)

72 @NicoJosuttis

Stop war
Stop Putin

Belleviews Principles

- **Iterating over a view is stateless**
  - You can always iterate when the view is const
  - You can iterate concurrently
  - Read iterations do not have any impact on later behavior
- **Respect the idioms of containers and arrays**
- **Honor constness**
  - Always propagate constness
  - Avoid using references that break const for elements
- **A copy of a view always behaves the same as its source**
- **All view types have a name ending with view**
  - Use sub_view instead of subrange
- **For all view types there is an adaptor/factory**
  - New factory bel::views::sub(beg,end)
- **Fix all knows flaws of specific views**

Stop war
Stop Putin

Iterate Over `const` Views

C++20 / Bel

```
void print(const auto& coll) {
  ...
}

std::vector<int> vec{0, 8, 15, 47, 11, 42};
std::list<int> lst{0, 8, 15, 47, 11, 42};
```

Std Views
✓
✓
✓
CT Error
✓
CT Error

```
print(vec | std::views::take(3));
print(lst | std::views::take(3));
print(vec | std::views::drop(3));
print(lst | std::views::drop(3));
for (int v : lst | std::views::drop(3)) {
   std::cout << v << ' ';
}
print(vec | std::views::filter(...));
```

Stop war
Stop Putin

Iterate Over const Views

C++20 / Bel

```
void print(const auto& coll) {
  ...
}
```

```
std::vector<int> vec{0, 8, 15, 47, 11, 42};
std::list<int> lst{0, 8, 15, 47, 11, 42};
```

	Std Views	Belleviews
`print(vec | bel::views::take(3));`	✓	✓
`print(lst | bel::views::take(3));`	✓	✓
`print(vec | bel::views::drop(3));`	✓	✓
`print(lst | bel::views::drop(3));`	CT Error	✓
`for (int v : lst | bel::views::drop(3)) { std::cout << v << ' '; }`	✓	✓
`print(vec | bel::views::filter(...));`	CT Error	✓

Stop war
Stop Putin

Iterate Over const Views

C++20 / Bel

void print(const auto& coll) {
  ...
}

std::vector<int> vec{0, 8, 15, 47, 11, 42};
std::list<int> lst{0, 8, 15, 47, 11, 42};

	Std Views	Belleviews
`print(vec | std/bel::views::take(3));`	✓	✓
`print(lst | std/bel::views::take(3));`	✓	✓
`print(vec | std/bel::views::drop(3));`	✓	✓
`print(lst | std/bel::views::drop(3));`	CT Error	✓
`for (int v : lst | std/bel::views::drop(3)) { std::cout << v << ' '; }`	✓	✓
`print(vec | std/bel::views::filter(...));`	CT Error	✓

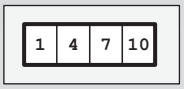
@NicoJosuttis

Stop war
Stop Putin

Using the Filter View

C++20

```
std::vector<int> coll{1, 4, 7, 10};
print(coll);

auto isEven = [] (auto&& i) { return i % 2 == 0; };
auto collEven = coll | std/bel::views::filter(isEven);

// increment even elements:
for (int& i : collEven) {
  i += 1;
}
print(coll);

// increment even elements:
for (int& i : collEven) {
  i += 1;
}
print(coll);
```



Std Views	Belleviews
RT Error	✓
RT Error	✓

Stop war
Stop Putin

Concurrent Iterations

C++20

```
std::list<int> coll{0, 8, 15, 47, 11, 42};

auto v = coll | std/bel::views::drop(2);

// while another thread prints the elements:
std::jthread printThread{[&] {
  for (const auto& elem : v) {
    std::cout << elem << ' ';
  }
  std::cout << '\n';
}};

// this thread computes the sum of the elements:
auto sum = std::accumulate(v.begin(), v.end(),
                           0L);
```

Std Views	Belleviews
RT Error	✓

Stop war
Stop Putin

const Propagation

C++20

```
std::vector vec{1, 2, 3, 4, 5, 6, 7, 8};

const auto& v = vec | std/bel::views::drop(2);

v[0] += 42;

if (v.front() = 2) {
  ...
}

auto v2 = v;      // NOTE: removes constness
v2[0] += 42;
```

Std Views	Belleviews
OOPS: OK	ERROR
OOPS: OK	ERROR
OK	OK

Stop war
Stop Putin

Modifications

C++20

```
std::vector vec{1, 2, 3, 4};
vec.reserve(9);
std::list lst{1, 2, 3, 4};
auto vVec = vec | std::views::drop(2);
auto vLst = lst | std::views::drop(2);
print(vVec, vLst);
```

Std Views	Belleviews
3 4 3 4	3 4 3 4

```
// insert new elements at the front:
vec.insert(vec.begin(), 0);
lst.insert(lst.begin(), 0);
print(vVec, vLst);
```

Std Views	Belleviews
2 3 4 3 4	2 3 4 2 3 4

```
vec.insert(vec.begin(), {8, 9, 0, -1});
lst.insert(lst.begin(), {8, 9, 0, -1});
print(vVec, vLst);
```

Std Views	Belleviews
0 -1 0 1 2 3 4 3 4	0 -1 0 1 2 3 4 0 -1 0 1 2 3 4

```
auto vVecB = vVec;
auto vLstB = vLst;
print(vVecB, vLstB);
```

Std Views	Belleviews
0 -1 0 1 2 3 4 0 -1 0 1 2 3 4	0 -1 0 1 2 3 4 0 -1 0 1 2 3 4

Stop war
Stop Putin

Belleviews Status

- https://github.com/josuttis/belleviews

Please help
to make C++ better

- **Not even beta**
  – I need your help (code, test, reviews)
- **But we already see how simply and safe views could have been**
- **I don't promise to solve all problems**
  – Ideally some time adopted by the C++ standard
- **Code based on C++20 ranges and const fixes of C++23**

81 @NicoJosuttis

Stop war
Stop Putin

Thank You and Take Care

Nicolai M. Josuttis

www.josuttis.com
nico@josuttis.com
@NicoJosuttis

82 @NicoJosuttis