The One Ranges Proposal

Three Proposal for `Views` under the Sky,
Seven for LEWG in their halls of stone,
Nine for the Ranges TS doomed to die,
One for the LWG on its dark throne
In the Land of Geneva where the Standard lie.

One Proposal to `ranges::merge` them all, One Proposal to `ranges::find` them,
One Proposal to bring them all and in `namespace ranges bind` them,
In the Land of Geneva where the Standard lie.

C++ Ranges

吴咏炜
wuyongwei@gmail.com

A Bit of History

- Boost.Range, 2003
- D range, ~2009
- Range-v3, 2013
- Range TS, 2014
- Merged into C++20, 2018
- Formally standardized, ~2020

About the Code

- All code can be compiled with
  - GCC with concept support (`-fconcept`)
  - C++17 support (`-std=c++17`)
  - **Cmcstl2** (https://github.com/CaseyCarter/cmcstl2)
  - Some need range-v3 (https://github.com/ericniebler/range-v3)
  - Some need nvwa (https://github.com/adah1972/nvwa)
- Not in final C++20 form
- Code is available at:
  - https://github.com/adah1972/cpp_conf_china_2018

Pre-Ranges Code

```
int a[] = {1, 7, 3, 6, 5, 2, 4, 8};
std::sort(std::begin(a), std::end(a));
std::copy(std::begin(a), std::end(a),
          std::ostream_iterator<int>(
              std::cout, " "));
```

Ranges Example—Sort

```
namespace ranges = std::experimental::ranges;
int a[] = {1, 7, 3, 6, 5, 2, 4, 8};
ranges::sort(a);
ranges::copy(a, ranges::ostream_iterator<int>(
                std::cout, " "));
```

Ranges Example—Reverse View

```
namespace ranges = std::experimental::ranges;
int a[] = {1, 7, 3, 6, 5, 2, 4, 8};
auto r = ranges::reverse_view(a);
ranges::copy(a, ranges::ostream_iterator<int>(
                std::cout, " "));
```

Ranges Example—Filter View

```
namespace ranges = std::experimental::ranges;
int a[] = {1, 7, 3, 6, 5, 2, 4, 8};
auto r = ranges::filter_view(a,
          [](int i) { return i % 2 == 0; });
ranges::copy(a, ranges::ostream_iterator<int>(
                  std::cout, " "));
```

Ranges Example—Nested Views

```
namespace ranges = std::experimental::ranges;
int a[] = {1, 7, 3, 6, 5, 2, 4, 8};
auto r = ranges::reverse_view(
           ranges::filter_view(a,
               [](int i) { return i % 2 == 0; }));
ranges::copy(a, ranges::ostream_iterator<int>(
                 std::cout, " "));
```

Ranges Example—Pipe

```
namespace ranges = std::experimental::ranges;
int a[] = {1, 7, 3, 6, 5, 2, 4, 8};
auto r = a
     | ranges::view::filter(
            [](int i) { return i % 2 == 0; })
     | ranges::view::reverse;
ranges::copy(a, ranges::ostream_iterator<int>(
                std::cout, " "));
```

Thinking of Unix Pipe

do_something … | grep … | sort | uniq

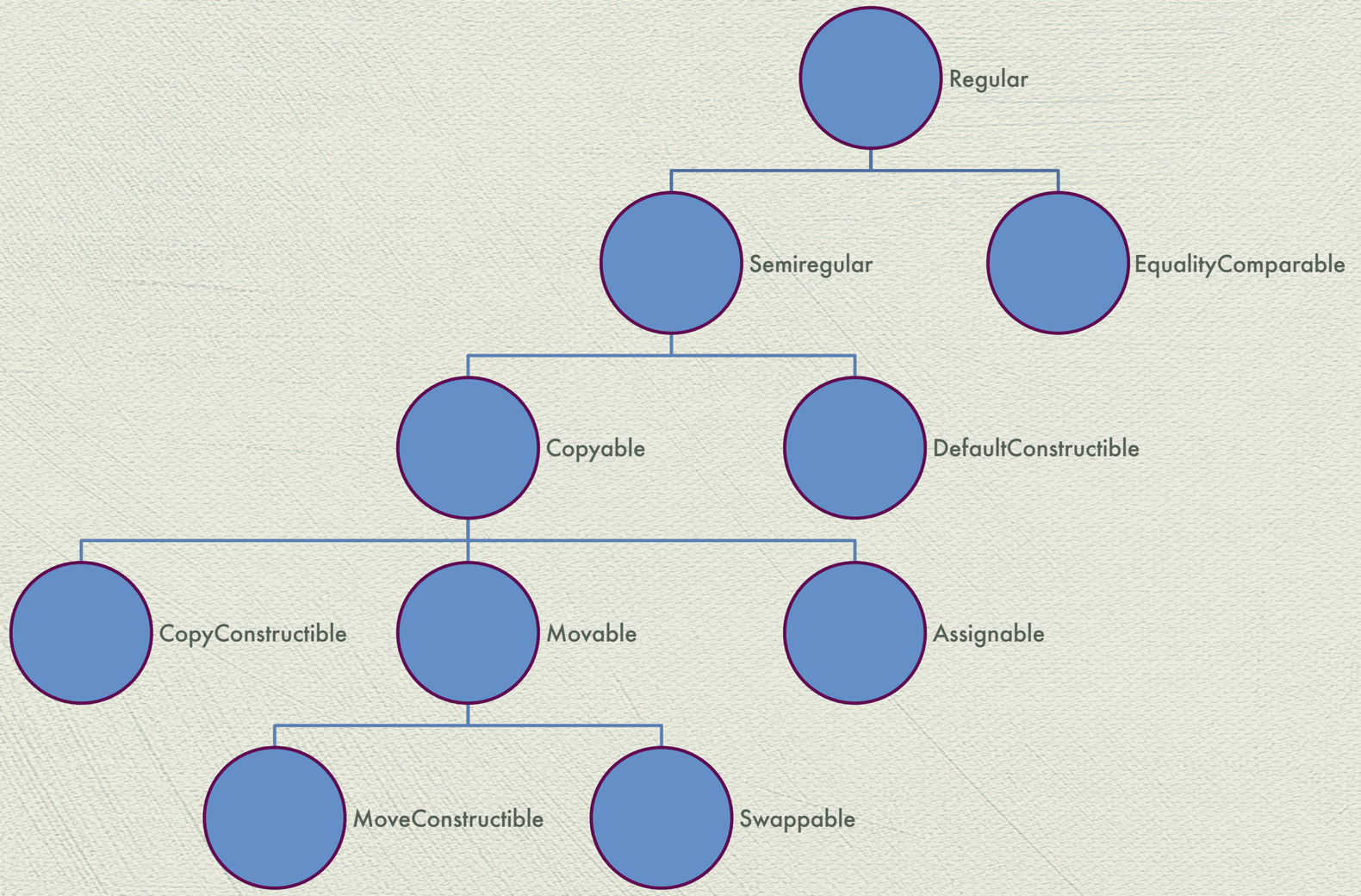
What is a range, exactly?

Definition of Range

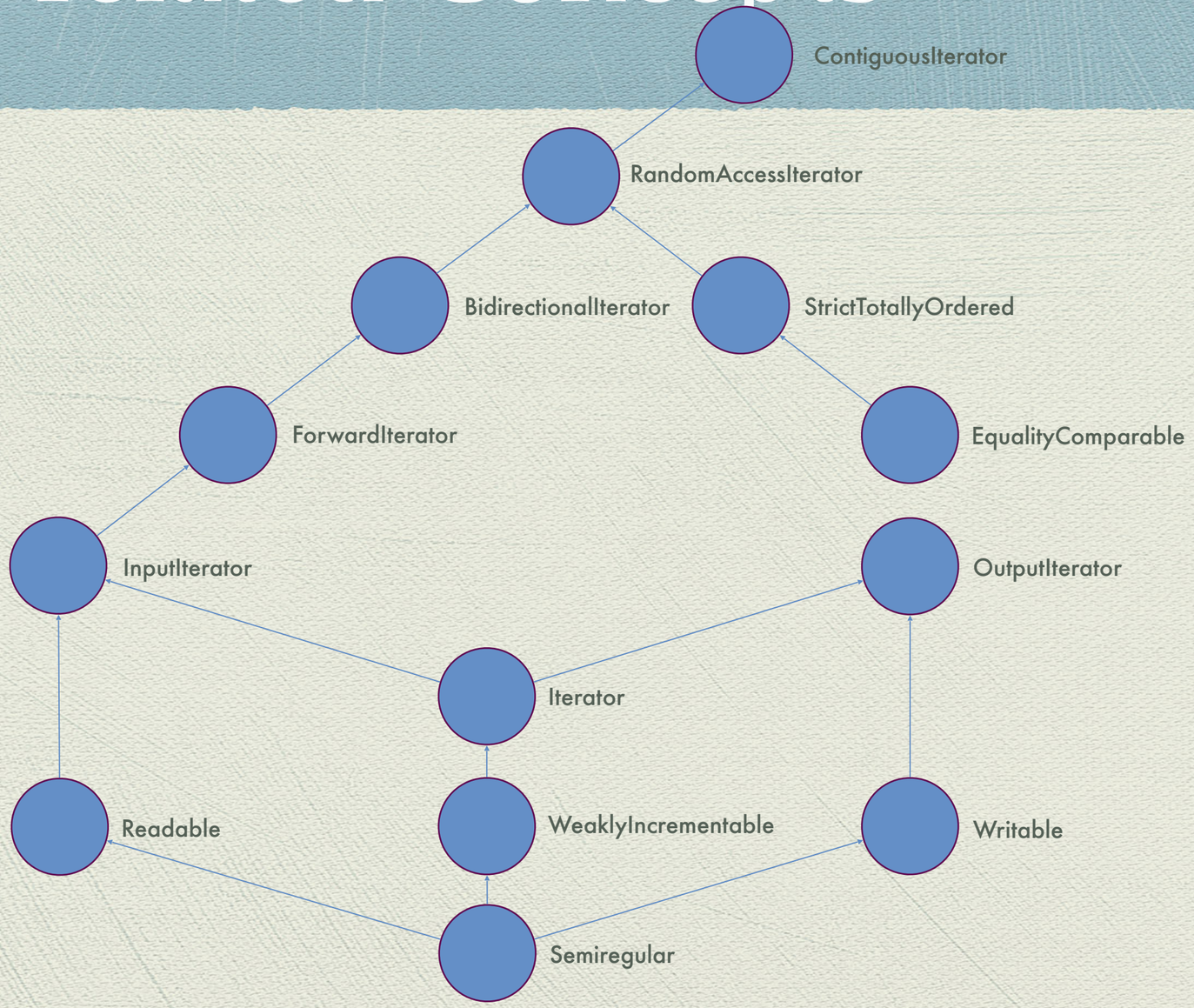
```
template<class T>
concept bool _RangeImpl =
    requires(T&& t) {
        ranges::begin(static_cast<T&&>(t));
        ranges::end(static_cast<T&&>(t));
    };

template<class T>
concept bool Range = _RangeImpl<T&>;
```

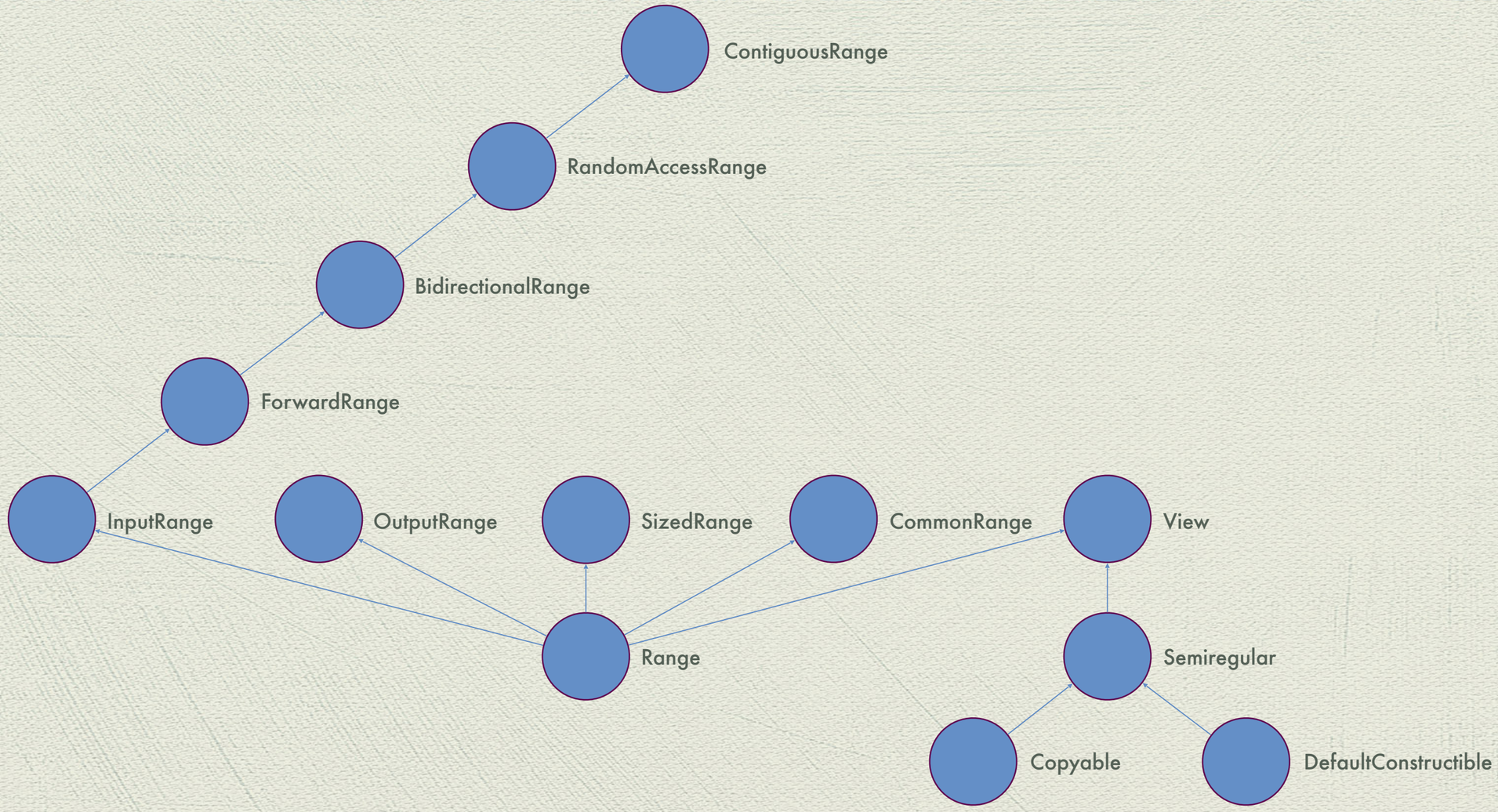
Basic Concepts



Iterator-related Concepts



Range Concepts



Definition of Range

```
template<class T>
concept bool _RangeImpl =
    requires(T&& t) {
        ranges::begin(static_cast<T&&>(t));
        ranges::end(static_cast<T&&>(t));
    };

template<class T>
concept bool Range = _RangeImpl<T&>;
```

View

```
template <class T>
concept bool View =
    Range<T> &&
    Semiregular<T> &&
    enable_view<remove_cvref_t<T>>;
```

CommonRange

```
template <class T>
concept bool CommonRange =
     Range<T> &&
     Same<iterator_t<T>, sentinel_t<T>>;
```

SizedRange

```
template <class T>
concept bool SizedRange =
     Range<T> &&
     !disable_sized_range<remove_cvref_t<T>> &&
     requires(T& r) { size(r); };
```

OutputRange

```
template <class R, class T>
concept bool OutputRange =
    Range<R> &&
    OutputIterator<iterator_t<R>, T>;
```

InputRange

```
template <class T>
concept bool InputRange =
     Range<T> &&
     InputIterator<iterator_t<T>>;
```

A Sentinel Example

```
struct null_sentinel {};
template <Iterator I>
bool operator==(I i, null_sentinel) { return *i == 0; }
//template <Iterator I>
//operator==(null_sentinel, I i), operator≠, …

ranges::for_each(argv[1], null_sentinel(),
                 [](char ch) { std::cout << ch; });
```

`Views`

- Do not own elements (but shared ownership is OK)
- Take constant time to copy, move, or assign
  - Most containers are `Ranges`, but not `Views`
  - C++17 `string_view` satisfies `View`, but `string` does not
- `Views` are `Semiregular`, i.e. `Copyable` and `DefaultConstructible`

Range Concept Check I

	`vector<int>`	`const vector<int>`
Range	✓	✓
View	✗	✗
SizedRange	✓	✓
CommonRange	✓	✓
OutputRange	✓	✗
InputRange	✓	✓
ForwardRange	✓	✓
BidirectionalRange	✓	✓
RandomAccessRange	✓	✓
ContiguousRange	✓	✓

Range Concept Check II

	`list<int>`	`const list<int>`
Range	✓	✓
View	✗	✗
SizedRange	✓	✓
CommonRange	✓	✓
OutputRange	✓	✗
InputRange	✓	✓
ForwardRange	✓	✓
BidirectionalRange	✓	✓
RandomAccessRange	✗	✗
ContiguousRange	✗	✗

Range Concept Check III

	`int [8]`	`int const [8]`
Range	✓	✓
View	✗	✗
SizedRange	✓	✓
CommonRange	✓	✓
OutputRange	✓	✗
InputRange	✓	✓
ForwardRange	✓	✓
BidirectionalRange	✓	✓
RandomAccessRange	✓	✓
ContiguousRange	✓	✓

Range Concept Check IV

	reverse_view<int [8]>	*reverse_view*<int const [8]>
Range	✓	✓
View	✓	✓
SizedRange	✓	✓
CommonRange	✓	✓
OutputRange	✓	✗
InputRange	✓	✓
ForwardRange	✓	✓
BidirectionalRange	✓	✓
RandomAccessRange	✓	✓
ContiguousRange	✗	✗

Range Concept Check V

	filter_view<int [8]>	*filter_view*<int const [8]>
Range	✓	✓
View	✓	✓
SizedRange	✗	✗
CommonRange	✓	✓
OutputRange	✓	✗
InputRange	✓	✓
ForwardRange	✓	✓
BidirectionalRange	✓	✓
RandomAccessRange	✗	✗
ContiguousRange	✗	✗

Range Concept Check VI

	`istream_line_reader`	`take_view<istream_line_reader>`
Range	✓	✓
View	✓	✓
SizedRange	✗	✗
CommonRange	✓	✗
OutputRange	✗	✗
InputRange	✓	✓
ForwardRange	✗	✗
BidirectionalRange	✗	✗
RandomAccessRange	✗	✗
ContiguousRange	✗	✗

Range Concept Check VII

	`iota_view(0)`	`iota_view(0, 5)`	`iota_view(0) | take(5)`
Range	✓	✓	✓
View	✓	✓	✓
SizedRange	✗	✓	✗
CommonRange	✗	✓	✗
OutputRange	✗	✗	✗
InputRange	✓	✓	✓
ForwardRange	✓	✓	✓
BidirectionalRange	✓	✓	✓
RandomAccessRange	✓	✓	✓
ContiguousRange	✗	✗	✗

Range as Merged into C++20

- Based on D4128, Range-v3, and Range TS
- Separate namespace to minimize impact
  - `std::ranges` (formerly `std::experimental::ranges`)
  - Except basic concepts, which will be in `std`
- Defined many *concepts*
- Added *views* and *view adapters*
- Modified standard *algorithms* and *utilities*

Standard `Views`

- `all`
- `empty`
- `take`
- `counted`
- `common`
- `reverse`
- `filter`
- `iota`
- `join`
- `single`
- `split`
- `transform`

Why should I use them?

A Simple Case of Map–Reduce

```
reduce(lambda x, y: x + y,
       map(lambda x: x * x, range(1, 101)))
```

Manual Loop?

```
auto square = [](int x) { return x * x; };

int sum = 0;
for (int i = 1; i < 101; ++i) {
    sum += square(i);
}
```

transform/accumulate?

```
auto rng = view::iota(1, 101);
std::vector<int> result;
std::transform(rng.begin(), rng.end(),
               std::back_inserter(result),
               square);
int sum = std::accumulate(
    result.begin(), result.end(), 0);
```

Slightly Improved

```
int sum = nvwa::reduce(
    std::plus<int>(),
    nvwa::fmap(square, view::iota(1, 101)));
```

Using `view::transform`

```
int sum = nvwa::reduce(
    std::plus<int>(),
    view::iota(1, 101) | view::transform(square));
```

Optimized Assembly Output

```
movl $338350, %esi
```

However, `Views` do not own elements.

A Python cat

```
def cat(files):
    for fn in files:
        with open(fn) as f:
            for line in f:
                yield line.rstrip('\n')

for line in cat(sys.argv[1:]):
    print(line)
```

The `join` View

1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

A Naïve Implementation of `cat`

```
auto make_line_reader = [](const char* filename) {
    std::ifstream ifs(filename);
    return nvwa::istream_line_reader(ifs);
};
for (auto&& line : view::counted(argv + 1, argc - 1) |
                   view::transform(make_line_reader) |
                   view::join) {
    std::cout << line << std::endl;
}
```

Nice!

Nice?

It would crash. . . .

Resources Need to Kept Alive

```
std::ifstream tmp;
auto make_line_reader = [&tmp](const char* filename) {
    tmp = std::ifstream(filename);
    return nvwa::istream_line_reader(tmp);
};
for (auto&& line : view::counted(argv + 1, argc - 1) |
                   view::transform(make_line_reader) |
                   view::join) {
    std::cout << line << std::endl;
}
```

More Goodies in Range-v3

- More `Views` (e.g. `drop`, `repeat`, `take_while`, & `zip`)
- Actions
- Range-ified versions of the numeric algorithms (e.g. `accumulate`)
- Being able to compile range-v3 is newsworthy for Microsoft!

Special thanks to *Eric Niebler* and *Casey Carter* for proposing Ranges, writing Cmcstl2, and providing help for the code presented here!

Main References

- Eric Niebler, ‘D4128: Ranges for the Standard Library: Revision 1’, https://ericniebler.github.io/std/wg21/D4128.html
- Eric Niebler, ‘Range-v3’, https://ericniebler.github.io/range-v3/
- Eric Niebler and Casey Carter, ‘Working Draft, C++ Extensions for Ranges’, http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2017/n4685.pdf
- Eric Niebler, Casey Carter, and Christopher Di Bella, ‘The One Ranges Proposal’, http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0896r2.pdf

Other References

- Andrei Alexandrescu, 'Iterators Must Go', http://accu.org/content/conf2009/AndreiAlexandrescu_iterators-must-go.pdf
- Casey Carter, 'Use the official range-v3 with MSVC 2017 version 15.9', https://blogs.msdn.microsoft.com/vcblog/2018/11/07/use-the-official-range-v3-with-msvc-2017-version-15-9/
- '2018 San Diego ISO C++ Committee Trip Report', https://www.reddit.com/r/cpp/comments/9vwvbz/2018_san_diego_iso_c_committee_trip_report_ranges/
- Ivan Čukić, *Functional Programming in C++*, Manning, 2019, https://www.manning.com/books/functional-programming-in-c-plus-plus

Backup

Iterators Must Go?

- A range is a pair of `begin`/`end` iterators packed together
- Iterators are replaced by ranges
- Such ranges are implemented in D

D Range: The Good

- Simple model, simple code
- Good composability

D Range: The Bad

- Ranges can only be shrunken
- Worse performance in some cases

D Range: The Ugly

- Too aggressive and not backward compatible
- Several functions for one algorithm
  - E.g. `find`, `findSkip`, `findSplit`, `findSplitBefore`, `findSplitAfter`

Modified Algorithms

- all_of
- any_of
- none_of
- for_each
- find*
- mismatch
- equal
- is_permutation
- search*
- copy*
- move*
- swap*
- transform
- replace*
- fill
- remove*
- reverse*
- rotate*
- shuffle
- sort*
- nth_element
- lower_bound
- upper_bound
- equal_range
- binary_search
- partition*
- merge*
- includes
- set_union
- set_intersection
- set_difference
- heap*
- min*
- max*
- *compare
- *permutation

Changes to Algorithms

- Copied under `std::ranges`
- Support `Range` as input
- Return value may be changed
  - Typically it is a `struct` that has two members: `in` and `out`

Range Utilities

- `begin`, `cbegin`, `rbegin`, `crbegin`
- `end`, `cend`, `rend`, `crend`
- `size`
- `empty`
- `data`, `cdata`