C++ now

2025

A View for Any Occasion

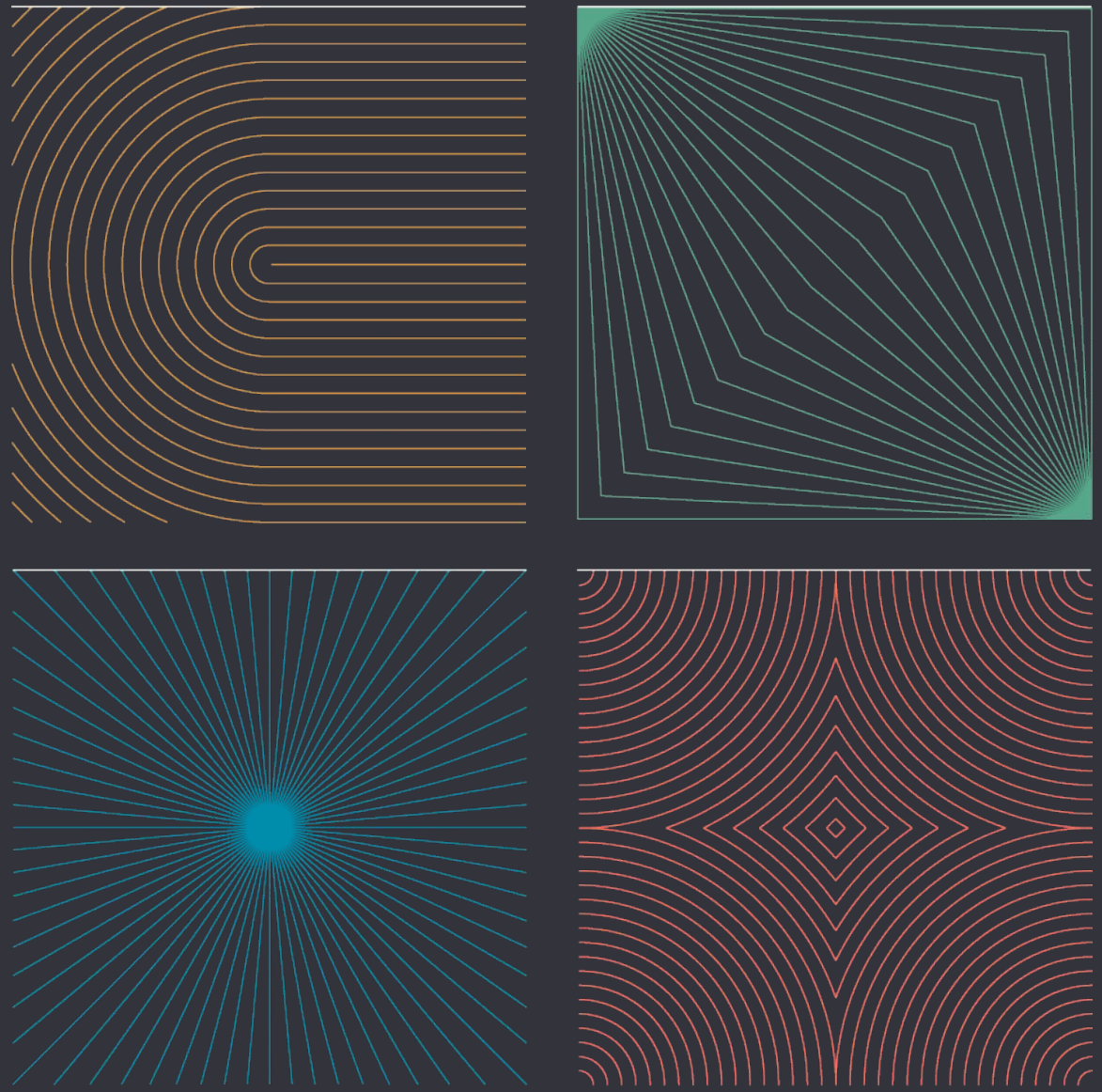
Implementing the C++ Standard Library Proposal for any_view

Patrick Roberts

A View for Any Occasion

Implementing the C++ standard library proposal for any_view

Patrick Roberts, P.E.



tenstorrent

A little bit about me

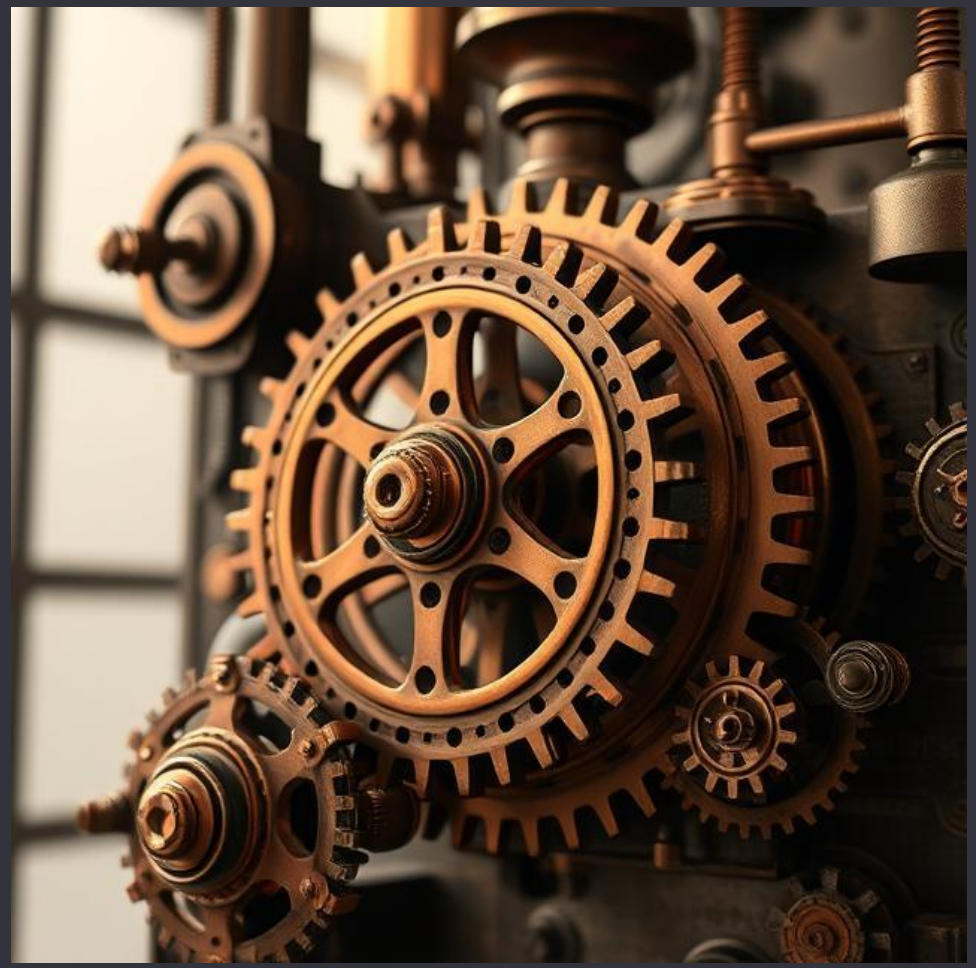
- Licensed professional electrical engineer since 2022
- Currently implementing RISC-V device kernels for an AI hardware company
- Worked for a high frequency trading firm between 2020 and 2024
- Began career writing firmware for embedded systems
- Enjoy metaprogramming and fractal art
  - https://patrickroberts.dev/fractal
- This is my first conference talk (excluding “lightning talks”)

Topics Covered

1. Motivation
2. Performance
3. Design
4. Implementation

Motivation

- Enabling range-based library APIs
- Standardizing established practices
- Generalization with acceptable cost



C++20 Ranges library

- Provides composable range adaptors
- Views are expression templates
  - Great for declarative programming
  - Not ideal for API boundaries

Enabling range-based library APIs

- Function return types
  - Type-erases a library-created range
  - Enables library to modify implementation freely without affecting users
- Function parameter types
  - Type-erases a user-provided range
  - Enables users to avoid explicitly constructing a tailored range for the library to consume

```
auto get_view() -> ???;
auto use_view(???);
```

Enabling range-based library APIs

- std::vector
  - Allocates memory to store data
  - Not intended for type-erasure, but often used that way for its simplicity
  - Unacceptable performance
- std::span
  - Specializes to enable type-erasure and performance
  - Constrained to borrowed, contiguous, sized ranges
  - Unacceptable for generality

```
auto get_view() -> ???;
auto use_view(???);
```

Standardizing established practices

- ranges::v3 any_view
  - Customizes:
    - Reference type
    - Traversal category
  - Does not customize:
    - Rvalue reference type
    - Difference type
- boost::ranges any_range
  - Customizes:
    - Reference type
    - Traversal category
    - Difference type
  - Does not customize:
    - Rvalue reference type
  - Only supports borrowed ranges

Generalization with acceptable cost

- Type-erasure of any viewable_range
- Configurable options to widen interface
  - Wide constraints by default
- Ideally usage should be faster than std::vector
  - When iterating the range once

Performance

- Design space
- Existing practices (std::vector)
- Proposal (std::ranges::any_view)

Performance

- Compare
- Increment
- Dereference

Syntax

attr (optional) **for** (*init-statement* (optional) *item-declaration* : *range-initializer*) *statement*

attr - any number of attributes

init-statement - (since C++20) one of

- an expression statement (which may be a null statement `;`)
- a simple declaration (typically a declaration of a variable with initializer), it may declare arbitrarily many variables or be a structured binding declaration
- an alias declaration (since C++23)

Note that any *init-statement* must end with a semicolon. This is why it is often described informally as an expression or a declaration followed by a semicolon.

item-declaration - a declaration for each range item

range-initializer - an expression or brace-enclosed initializer list

statement - any statement (typically a compound statement)

```
{
    init-stateme
    auto&& /* ra  */ = range-initializer;
    auto /* begin  * begin-expr */;
    auto /* end */ =  end-expr */;
    for ( ; /* begin */ != /* end */; ++/* begin */)
    {
        item-declaration = */* begin */;
        statement
    }
}
```

Performance

- Database of products
- Filter by quantity
- Get name

```
struct product {
    std::string name;
    std::size_t quantity;
};
```

```
struct database {
    std::vector<product> products;

    auto get_products(std::size_t min_available) const -> names;
};
```

“eager”

```
using names = std::vector<std::string_view>;
```

“lazy”

```
using names = beman::any_view::any_view<const std::string>;
```

“fused”

```
using names = beman::any_view::any_view<const std::string, beman::any_view::any_view_options::forward>;
```

Performance

```cpp
constexpr auto max_size = 1 << 18;

const auto global_products = generate_random_products(max_size);

static void BM_all_eager(benchmark::State& state) {
    eager::database db{global_products
        | std::views::take(state.range(0))
        | std::ranges::to<std::vector>()};

    for (auto _ : state) {
        for (std::string_view name : db.get_products(10)) {
            benchmark::DoNotOptimize(name);
        }
    }
}

BENCHMARK(BM_all_eager)->RangeMultiplier(2)->Range(1 << 10, max_size);
```

```cpp
auto generate_random_products(std::size_t count) -> products {
    products results;
    results.reserve(count);

    constexpr char alphanum[] = "0123456789"
                                "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                                "abcdefghijklmnopqrstuvwxyz";

    std::mt19937                                          char_rng;
    std::uniform_int_distribution<std::mt19937::result_type> char_dist(0, sizeof(alphanum) - 1);

    std::mt19937                                          len_rng;
    std::uniform_int_distribution<std::mt19937::result_type> len_dist(1, 30);

    const auto gen_next_str = [&]() {
        std::string str;
        str.reserve(len_dist(len_rng));

        for (std::size_t i = 0; i < str.capacity(); ++i) {
            str.push_back(alphanum[char_dist(char_rng)]);
        }

        return str;
    };

    std::mt19937                      w_rng;
    std::uniform_int_distribution<int> w_dist(0, 100);

    const auto gen_size = [&] { return w_dist(w_rng); };

    for (std::size_t i = 0; i < results.capacity(); ++i) {
        results.emplace_back(gen_next_str(), gen_size());
    }

    return results;
}
```

Performance

- Eager
- Lazy

```
auto eager::database::get_products(std::size_t min_quantity) const -> names {
    names results;

    for (const auto& product : products) {
        if (product.quantity >= min_quantity) {
            results.push_back(product.name);
        }
    }

    return results;
}
```

```
auto lazy::database::get_products(std::size_t min_quantity) const -> names {
    return products
        | std::views::filter([=](const auto& product) {
            return product.quantity >= min_quantity;
        })
        | std::views::transform(&product::name);
}
```

Performance

- Eager
- Lazy

Benchmark	Time	CPU	Time Old	Time New	CPU Old	CPU New
BM_all_[eager vs. lazy]/1024	+0.6241	+0.6241	1971	3202	1971	3202
BM_all_[eager vs. lazy]/2048	+0.5822	+0.5822	3844	6081	3843	6081
BM_all_[eager vs. lazy]/4096	+0.6798	+0.6798	7403	12435	7403	12435
BM_all_[eager vs. lazy]/8192	-0.2333	-0.2333	33649	25798	33649	25798
BM_all_[eager vs. lazy]/16384	+0.5452	+0.5452	32073	49560	32072	49559
BM_all_[eager vs. lazy]/32768	+0.2315	+0.2315	84985	104655	84985	104655
BM_all_[eager vs. lazy]/65536	+0.0991	+0.0991	194991	214322	194990	214320
BM_all_[eager vs. lazy]/131072	+0.1106	+0.1106	414870	460755	414868	460751
BM_all_[eager vs. lazy]/262144	-0.1106	-0.1105	1036610	922002	1036591	921996
OVERALL_GEOMEAN	+0.2381	+0.2381	0	0	0	0

Performance

- Eager
- Fused

Benchmark	Time	CPU	Time Old	Time New	CPU Old	CPU New
BM_all_[eager vs. fused]/1024	-0.0689	-0.0682	2022	1882	2020	1882
BM_all_[eager vs. fused]/2048	-0.0380	-0.0379	3952	3802	3952	3802
BM_all_[eager vs. fused]/4096	+0.0395	+0.0398	7763	8070	7760	8069
BM_all_[eager vs. fused]/8192	-0.5182	-0.5164	35432	17071	35296	17071
BM_all_[eager vs. fused]/16384	+0.0907	+0.0910	33589	36635	33578	36635
BM_all_[eager vs. fused]/32768	-0.1143	-0.1142	86879	76953	86873	76953
BM_all_[eager vs. fused]/65536	-0.2311	-0.2319	202251	155506	202149	155277
BM_all_[eager vs. fused]/131072	-0.2307	-0.2293	439950	338467	439185	338465
BM_all_[eager vs. fused]/262144	-0.2845	-0.2845	938975	671801	938889	671802
OVERALL_GEOMEAN	-0.1718	-0.1712	0	0	0	0

Performance

- Implementation uses virtual polymorphism
- Potential optimizations remaining
  - Inline vtable
    - May reduce cache misses
  - Storage-aware vtable
    - Eliminates branch in dispatch
    - Reduces size of class layout

Performance

```cpp
constexpr auto max_size = 1 << 18;

const auto global_products = generate_random_products(max_size);

static void BM_all_eager(benchmark::State& state) {
    eager::database db{global_products
        | std::views::take(state.range(0))
        | std::ranges::to<std::vector>()};

    for (auto _ : state) {
        for (std::string_view name : db.get_products(10)) {
            benchmark::DoNotOptimize(name);
        }
    }
}

BENCHMARK(BM_all_eager)->RangeMultiplier(2)->Range(1 << 10, max_size);
```

```cpp
constexpr auto max_size = 1 << 18;

const auto global_products = generate_random_products(max_size);

static void BM_take_eager(benchmark::State& state) {
    eager::database db{global_products
        | std::views::take(state.range(0))
        | std::ranges::to<std::vector>()};

    for (auto _ : state) {
        for (std::string_view name : db.get_products(10) | std::views::take(100) {
            benchmark::DoNotOptimize(name);
        }
    }
}

BENCHMARK(BM_take_eager)->RangeMultiplier(2)->Range(1 << 10, max_size);
```

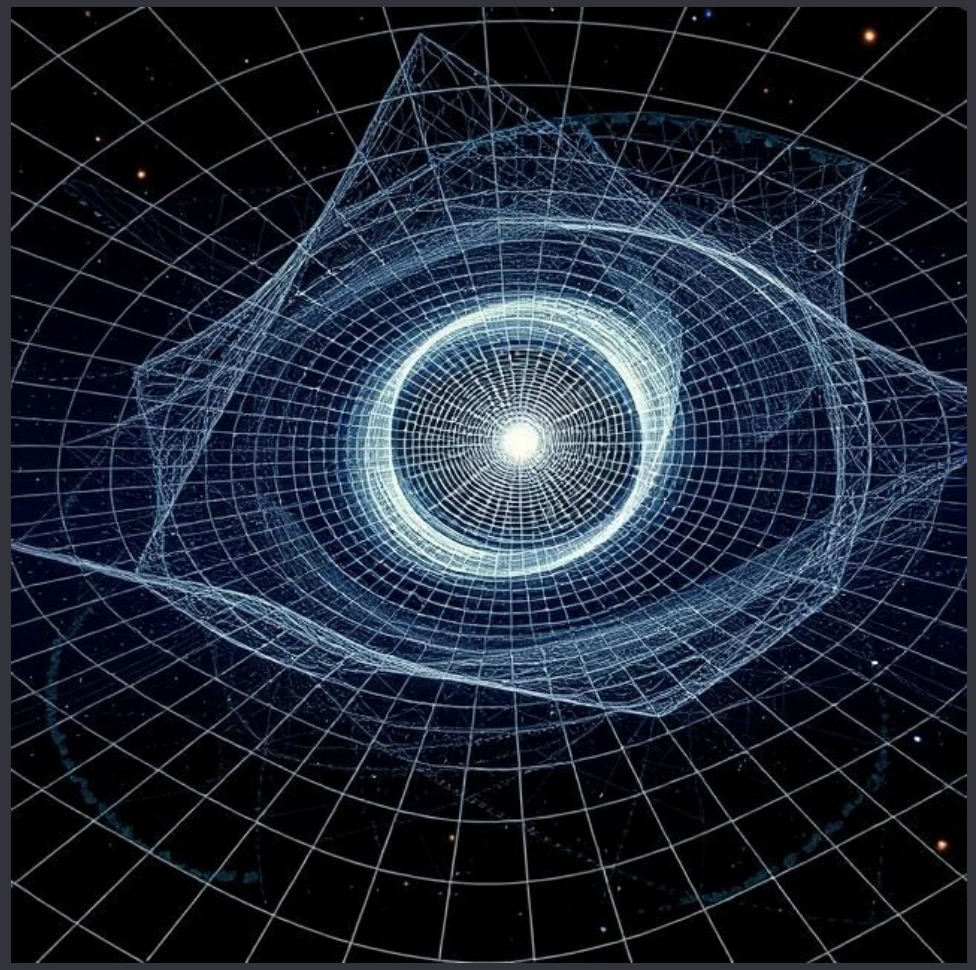
Performance

Benchmark	Time	CPU	Iterations
BM_take_eager/1024	1468 ns	1468 ns	563485
BM_take_eager/2048	3221 ns	3221 ns	245144
BM_take_eager/4096	5023 ns	5023 ns	149442
BM_take_eager/8192	9482 ns	9482 ns	56011
BM_take_eager/16384	20539 ns	20539 ns	36914
BM_take_eager/32768	49319 ns	49319 ns	15582
BM_take_eager/65536	100452 ns	100452 ns	6840
BM_take_eager/131072	233676 ns	233676 ns	3029
BM_take_eager/262144	500944 ns	500942 ns	1320
BM_take_fused/1024	188 ns	187 ns	3730095
BM_take_fused/2048	188 ns	188 ns	3610550
BM_take_fused/4096	191 ns	191 ns	3223302
BM_take_fused/8192	197 ns	197 ns	3730675
BM_take_fused/16384	199 ns	199 ns	3691553
BM_take_fused/32768	188 ns	188 ns	3686154
BM_take_fused/65536	188 ns	188 ns	3738689
BM_take_fused/131072	194 ns	194 ns	3593208
BM_take_fused/262144	189 ns	189 ns	3376302
BM_take_lazy/1024	310 ns	310 ns	2275033
BM_take_lazy/2048	310 ns	310 ns	2041406
BM_take_lazy/4096	316 ns	316 ns	2246033
BM_take_lazy/8192	316 ns	316 ns	2171179
BM_take_lazy/16384	313 ns	313 ns	2250327
BM_take_lazy/32768	312 ns	312 ns	2105924
BM_take_lazy/65536	313 ns	313 ns	2242918
BM_take_lazy/131072	314 ns	314 ns	2209593
BM_take_lazy/262144	316 ns	316 ns	2244185

```cpp
constexpr auto max_size = 1 << 18;

const auto global_products = generate_random_products(max_size);

static void BM_take_eager(benchmark::State& state) {
    eager::database db{global_products
        | std::views::take(state.range(0))
        | std::ranges::to<std::vector>()};

    for (auto _ : state) {
        for (std::string_view name : db.get_products(10) | std::views::take(100)) {
            benchmark::DoNotOptimize(name);
        }
    }
}

BENCHMARK(BM_take_eager)->RangeMultiplier(2)->Range(1 << 10, max_size);
```

Design

- View vs. range
- Iterator concept
- Range concepts
- Reference type
- Move-iteration
- Difference type



View vs. range

- any_view *is* a view
- any_view *type-erases* a range

- A range must have
  - begin() that returns an iterator
  - end() that returns a sentinel

- A view is a range that must additionally be
  - O(1) movable
  - Non-copyable, or O(1) copyable

Customizing iterator concept

- input_range<R>
- forward_range<R>
- bidirectional_range<R>
- random_access_range<R>
- contiguous_range<R>

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

input

- Default iterator concept
- Almost any range is viewable
- Iterator behavior:
  - Not copyable
  - Not default constructible
  - i++ returns void
- Example:
  - std::views::istream

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

forward

- Fused increment/compare/dereference
- Input range is no longer viewable
- Iterator behavior:
  - Copyable
  - Default constructible
  - Weakly comparable (i == j, i != j)
  - i++ returns value
- Examples:
  - std::forward_list
  - std::views::lazy_split

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

bidirectional

- Iterator is decrementable
- std::ranges::rbegin(r) is still ill-formed
  - Sentinel is not dereferenceable
  - Not a common range
  - Could provide if
    - Random access
    - Sized
    - Workaround: std::views::common
- Examples:
  - std::list
  - std::views::filter

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

random_access

- Random access does not imply sized
- Iterator behavior:
  - Comparable (i < j, i > j, i <= j, i >= j)
  - Subtractable (i - j)
  - Advanceable (i += n, i -= n)
  - Indexable (i[n])
- Examples:
  - std::deque
  - std::views::iota

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

contiguous

- Contiguous also does not imply sized
- Iterator behaves exactly like address
- Examples:
  - std::vector
  - std::span
  - std::optional (C++26)

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

Customizing range concepts

- sized_range<R>
- borrowed_range<R>
- common_range<R>
- range<const R>
- copyable<R>

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

sized

- Not implied by random_access or contiguous
- std::ranges::size(r) well-formed
- Iterator subtractable from sentinel (s - i)

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

borrowed

- Iterators may outlive the range
- Function can take the range by value and return iterators obtained from it
- Examples:
  - Lvalues (std::vector<int> &)
  - std::string_view

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

common ❌

- Constrained algorithms don't need common ranges
- Adds implementation complexity
- Workaround: std::views::common

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

constant ❌

- wg21.link/p3431

Deprecate `const`-qualifier on begin/end of views

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

~~move_only~~ copyable ✅

- Default to wide constraint
  - Should “just work”
- Default to narrow interface
  - Only pay for what you use
- Often coincides with borrowed
- Examples:
  - Lvalues
  - std::ranges::subrange

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

Customizing reference type

- A reference might not be an lvalue
- Examples:
  - std::vector<bool>
  - std::views::transform

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

Customizing move-iteration

- A value might be an aggregate
- Examples:
  - std::views::zip
  - std::views::enumerate

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

Customizing difference type

- An iterator might not have the same domain as an address
- Example:
  - std::ranges::iota_view

```
enum class any_view_options {
    input          = 0b00000001,
    forward        = 0b00000011,
    bidirectional  = 0b00000111,
    random_access  = 0b00001111,
    contiguous     = 0b00011111,
    sized          = 0b00100000,
    borrowed       = 0b01000000,
    copyable       = 0b10000000,
};

template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view;
```

Implementation

- Small buffer optimization
- Type-erasing constructors
- Constraining virtual functions
- Fused iteration

Small buffer optimization

- Strong exception safety guarantee
- Inplace storage requires
  - Non-throwing move construction
  - Non-constant evaluation
  - Sufficiently small size and alignment

Small buffer optimization

```
template <class InterfaceT>
concept interface_movable =
    std::has_virtual_destructor_v<InterfaceT> and requires(InterfaceT& instance, void* destination) {
        { instance.move_to(destination) } noexcept -> std::same_as<void>;
    };

template <class InterfaceT>
concept interface_copyable =
    interface_movable<InterfaceT> and requires(const InterfaceT& instance, void* destination) {
        { instance.copy_to(destination) } -> std::same_as<void>;
        { instance.copy() } -> std::same_as<InterfaceT*>;
    };

enum class index_type : bool {
    is_inplace,
    is_pointer,
};

template <interface_movable InterfaceT,
          std::size_t SizeV = sizeof(void*),
          std::size_t AlignV = alignof(void*)>
class intrusive_small_ptr {
    struct inplace_type {
        alignas(AlignV) std::byte data[SizeV];
    };

    using pointer_type = InterfaceT*;

    union {
        inplace_type inplace;
        pointer_type pointer;
    };

    index_type index;
```

Small buffer optimization

```
template <class ElementT,
          any_view_options OptsV = any_view_options::input,
          class RefT             = ElementT&,
          class RValueRefT       = detail::as_rvalue_t<RefT>,
          class DiffT            = std::ptrdiff_t>
class any_view : public std::ranges::view_interface<any_view<ElementT, OptsV, RefT, RValueRefT, DiffT>> {
    using iterator_concept = decltype(detail::get_iter_concept<OptsV>());
    using interface_type = detail::view_interface<iterator_concept, ElementT, RefT, RValueRefT, DiffT>;
    // inplace storage sufficient for a vtable pointer and a std::vector<T>
    detail::intrusive_small_ptr<interface_type, 4 * sizeof(void*)> view_ptr;
```

```
template <class IterConceptT, class ElementT, class RefT, class RValueRefT, class DiffT>
class any_iterator {
    using interface_type = iterator_interface<ElementT, RefT, RValueRefT, DiffT>;

    // inplace storage sufficient for a vtable pointer and two pointers
    intrusive_small_ptr<interface_type, 3 * sizeof(void*)> iterator_ptr;
```

Type-erasing construction

- Virtual polymorphism is not directly compatible with value semantics
- There is no such thing as a virtual constructor
  - Will slice object without placement-new
- Derived classes have unknown size

```
Constructor cannot be declared 'virtual' clang(constructor_cannot_be)

constructor foo

Parameters:

• const foo &

// In foo
public: foo(const foo&)

View Problem (Alt+F8)    No quick fixes available

struct foo {
    virtual foo(const foo &) = 0;
};
```

Type-erasing construction

```
template <class InterfaceT>
concept interface_movable =
    std::has_virtual_destructor_v<InterfaceT> and requires(InterfaceT& instance, void* destination) {
        { instance.move_to(destination) } noexcept -> std::same_as<void>;
    };

template <class InterfaceT>
concept interface_copyable =
    interface_movable<InterfaceT> and requires(const InterfaceT& instance, void* destination) {
        { instance.copy_to(destination) } -> std::same_as<void>;
        { instance.copy() } -> std::same_as<InterfaceT*>;
    };
```

Type-erasing construction

```
constexpr intrusive_small_ptr(const intrusive_small_ptr& other)
    requires interface_copyable<InterfaceT>
    : index(other.index) {
    if (index == index_type::is_inplace) {
        other.get_inplace()->copy_to(&inplace);
    } else {
        pointer = other ? other.pointer->copy() : nullptr;
    }
}

constexpr intrusive_small_ptr(intrusive_small_ptr&& other) noexcept : index(other.index) {
    if (index == index_type::is_inplace) {
        other.get_inplace()->move_to(&inplace);
    } else {
        pointer = std::exchange(other.pointer, nullptr);
    }
}
```

```
constexpr auto operator=(const intrusive_small_ptr& other) -> intrusive_small_ptr&
    requires interface_copyable<InterfaceT>
{
    // prevent self-assignment
    if (this == std::addressof(other)) {
        return *this;
    }

    this->~intrusive_small_ptr();
    std::construct_at(this, other);
    return *this;
}

constexpr auto operator=(intrusive_small_ptr&& other) noexcept -> intrusive_small_ptr& {
    this->~intrusive_small_ptr();
    std::construct_at(this, std::move(other));
    return *this;
}
```

Constraining virtual functions

- C++20 introduces constrained non-template member functions
- But virtual functions cannot be constrained

```
template <class IterConceptT>
struct iterator_interface {
    virtual auto operator+=(std::ptrdiff_t n) -> iterator_interface&
                Virtual function cannot have a requires clause clang(constrained_virtual_method)
                namespace std
                namespace std {}
                View Problem (Alt+F8)   No quick fixes available
        requires std::derived_from<IterConceptT, std::random_access_iterator_tag>
    = 0;
};
```

Constraining virtual functions

- std::unreachable (C++23)
  - Allows us to discard ill-formed statements
  - No restrictions on return type
  - Implementation detail only, user-facing methods are conditionally enabled

```
[[nodiscard]] constexpr auto operator-(const iterator_interface& other) const -> DiffT override {
    if constexpr (std::random_access_iterator<IteratorT>) {
        if (const auto adaptor = down_cast(other)) {
            return iterator - adaptor->iterator;
        }
    }

    unreachable();
}
```

Fused iteration

- Virtual dispatch has significant overhead
- Multi-pass guarantee allows optimization
- Reduces virtual dispatches per iteration from three to one

```
[[nodiscard]] constexpr auto next() -> iter_cache_t<RefT> override {
    if constexpr (std::forward_iterator<IteratorT>) {
        return ++iterator != sentinel ? iter_cache_t<RefT>{*iterator} : std::nullopt;
    }

    unreachable();
}
```

Fused iteration

- Prevents RVO of dereference
- Increases size of class layout

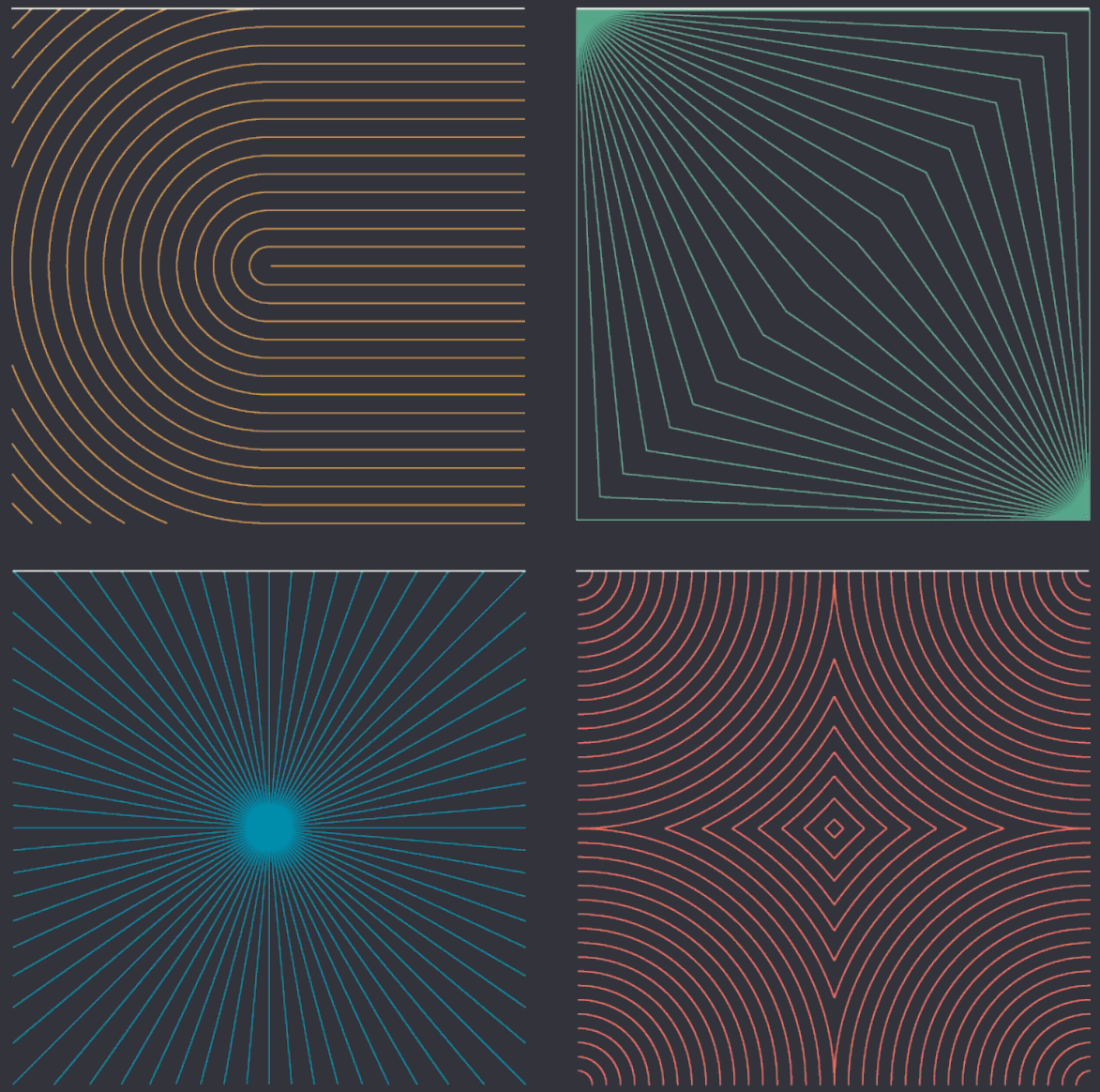
```
using cache_type =
    std::conditional_t<std::derived_from<IterConceptT, std::forward_iterator_tag>, iter_cache_t<RefT>, no_cache>;

static constexpr bool cached = not std::same_as<cache_type, no_cache>;

[[no_unique_address]] cache_type cache;

constexpr any_iterator& operator++() {
    if constexpr (cached) {
        cache = iterator_ptr->next();
    } else {
        ++*iterator_ptr;
    }
    return *this;
}

[[nodiscard]] constexpr bool operator==(std::default_sentinel_t sentinel) const {
    if constexpr (cached) {
        return not cache.has_value();
    } else {
        return *iterator_ptr == sentinel;
    }
}

[[nodiscard]] constexpr reference operator*() const {
    if constexpr (cached) {
        return *cache;
    } else {
        return **iterator_ptr;
    }
}
```

Questions?

Thank you

https://patrickroberts.dev
https://github.com/bemanproject/any_view
https://wg21.link/P3411

tenstorrent



Bonus Slides

Pre-allocated vector vs. any_view with fused iteration

Benchmark	Time	CPU	Time Old	Time New	CPU Old	CPU New
BM_all_[reserved vs. fused]/1024	-0.0702	-0.0702	1785	1659	1785	1659
BM_all_[reserved vs. fused]/2048	-0.1221	-0.1221	3697	3245	3697	3245
BM_all_[reserved vs. fused]/4096	-0.0486	-0.0486	7514	7149	7514	7149
BM_all_[reserved vs. fused]/8192	+0.0005	+0.0005	15377	15384	15377	15384
BM_all_[reserved vs. fused]/16384	+0.0781	+0.0781	30856	33267	30856	33267
BM_all_[reserved vs. fused]/32768	+0.0071	+0.0071	70153	70649	70153	70649
BM_all_[reserved vs. fused]/65536	-0.1004	-0.1004	165312	148711	165313	148709
BM_all_[reserved vs. fused]/131072	-0.0892	-0.0893	345413	314586	345414	314585
BM_all_[reserved vs. fused]/262144	-0.1537	-0.1537	765841	648129	765834	648118
OVERALL_GEOMEAN	-0.0578	-0.0579	0	0	0	0

```cpp
#include "reserved.hpp"

#include <algorithm>

auto reserved::database::get_products(query_t query) const -> names_t {
    const auto capacity = std::ranges::count_if(
        products, [=](const product_t& product) -> bool { return product.quantity >= query.min_quantity; });

    names_t results;
    results.reserve(capacity);

    for (const auto& product : products) {
        if (product.quantity >= query.min_quantity) {
            results.emplace_back(product.name);
        }
    }

    return results;
}
```

tenstorrent