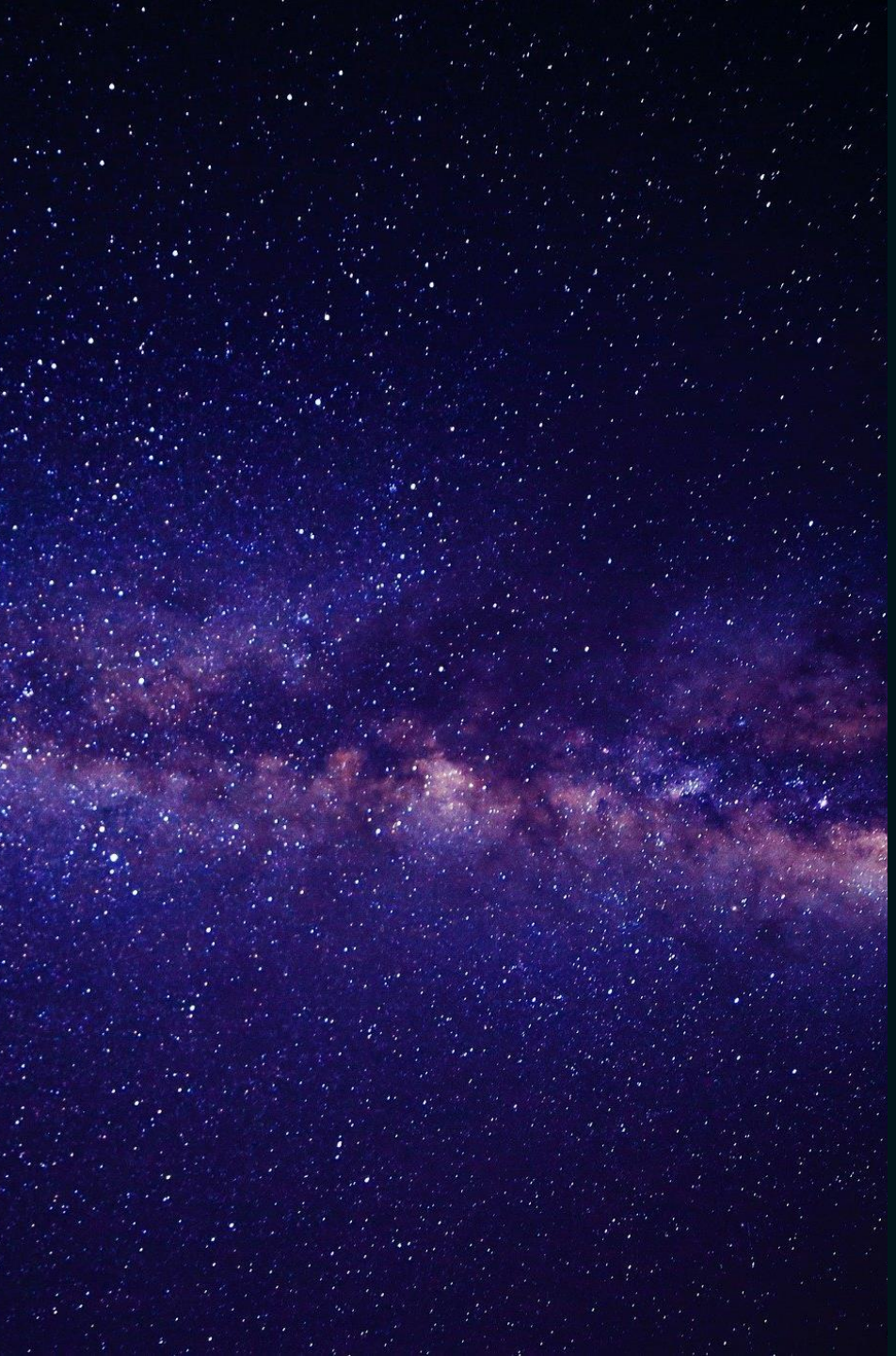
accu 2022

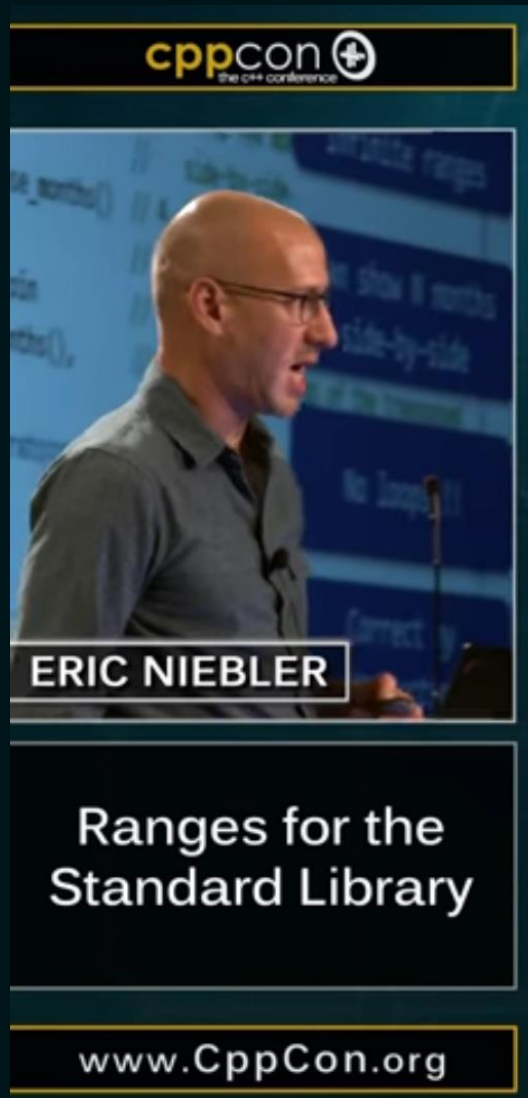
HOW TO RANGIFY YOUR CODE

TINA ULBRICH



Talk Recommendations

Talk Recommendations



Calendar Solution

```
int main() {
    copy(
        dates_in_year(2015)      // 0. Make a range
          | by_month()           // 1. Group the dates by month.
          | layout_months()      // 2. Format the mo
                                 //    strings.
          | chunk(3)             // 3. Group the mon
                                 //    side-by-side.
          | transpose_months()   // 4. Transpose the
                                 //    of the size-b
          | view::join           // 6. Ungroup the s
          | join_months(),       // 7. Join the strings of the transposed
                                 //    months.
        ostream_iterator<>(std::cout, "\n")
    );
}
```

- Composable
- Reusable
- Works with infinite ranges
- Can show *N* months side-by-side
- No loops!!!
- Correct *by construction*.

Talk Recommendations

Talk Recommendations

Talk Recommendations

Talk Recommendations – Sy Brand

Register Allocation Explained With My Cats

343 Aufrufe • vor 3 Wochen

Untertitel

Livecoding C++ Ranges: standardisation,...

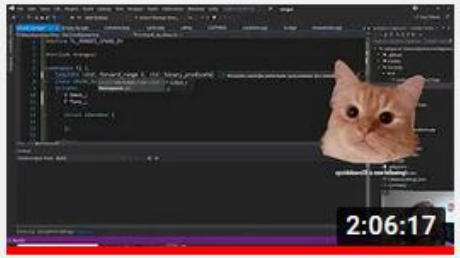
254 Aufrufe • vor 3 Monaten

Untertitel

Concurrency vs Parallelism Explained With My Cats

651 Aufrufe • vor 3 Monaten

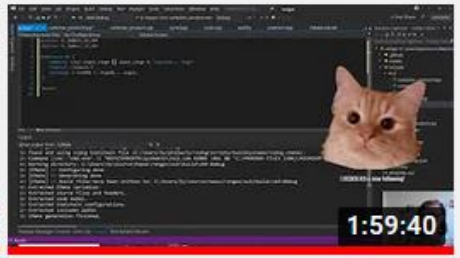
Untertitel



Livecoding C++ Ranges: chunk_by and chunk_by_key

304 Aufrufe • vor 4 Monaten

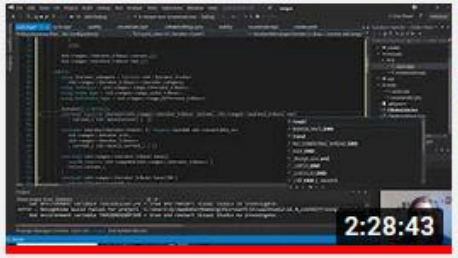
Untertitel



Livecoding C++ Ranges: std::ranges::to

346 Aufrufe • vor 4 Monaten

Untertitel



Livecoding C++ Ranges: cycle and cartesian_product

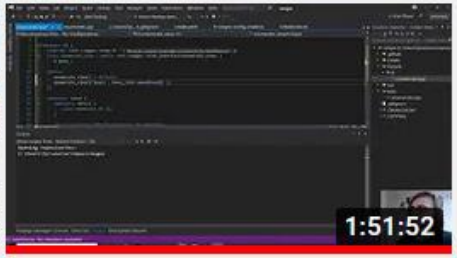
290 Aufrufe • vor 5 Monaten

Untertitel

Apex Legends: 30 Seconds Per Frame

256 Aufrufe • vor 5 Monaten

Untertitel



Livecoding C++20 Ranges: enumerate

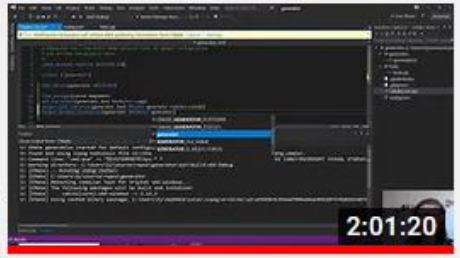
294 Aufrufe • vor 5 Monaten

Untertitel

C++ GUIs with coroutines, WinUI3, C++/WinRT and...

1760 Aufrufe • vor 5 Monaten

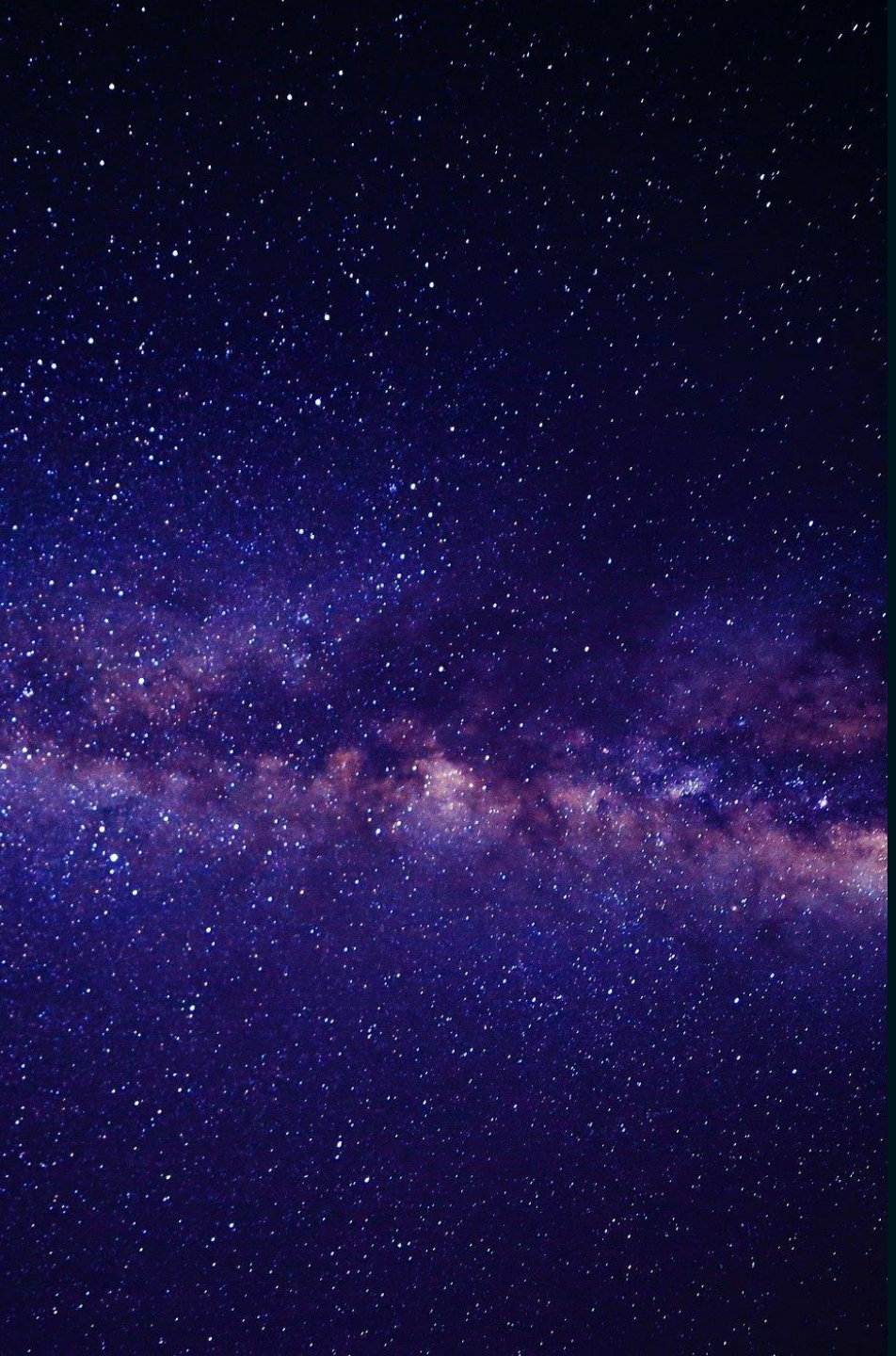
Untertitel



Implementing Generators with C++20 Coroutines

2002 Aufrufe • vor 6 Monaten

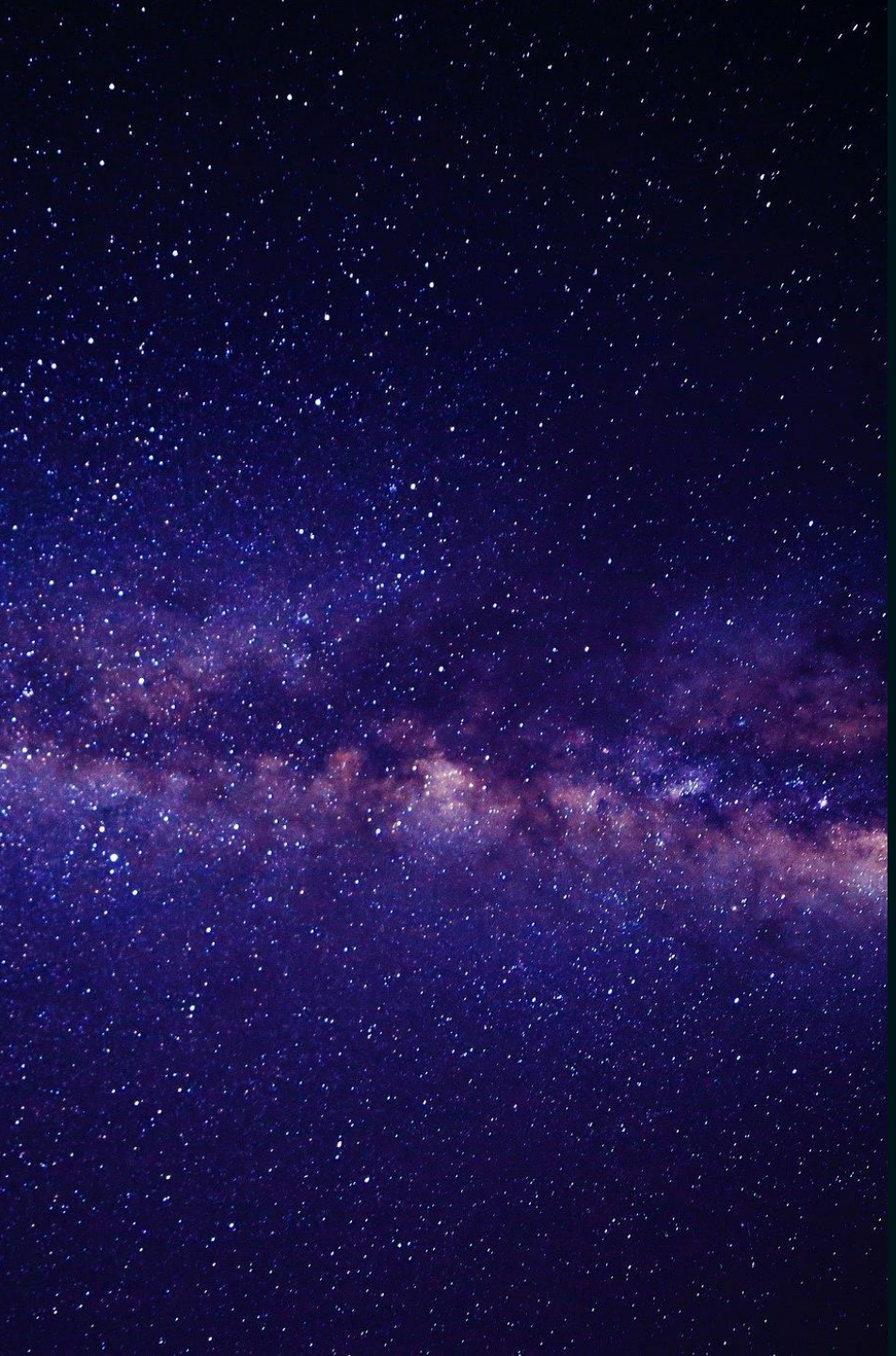
Untertitel



Libraries

Libraries

- range-v3
- nanorange
- TartanLlama/ranges
- rangesnext
- boost/range

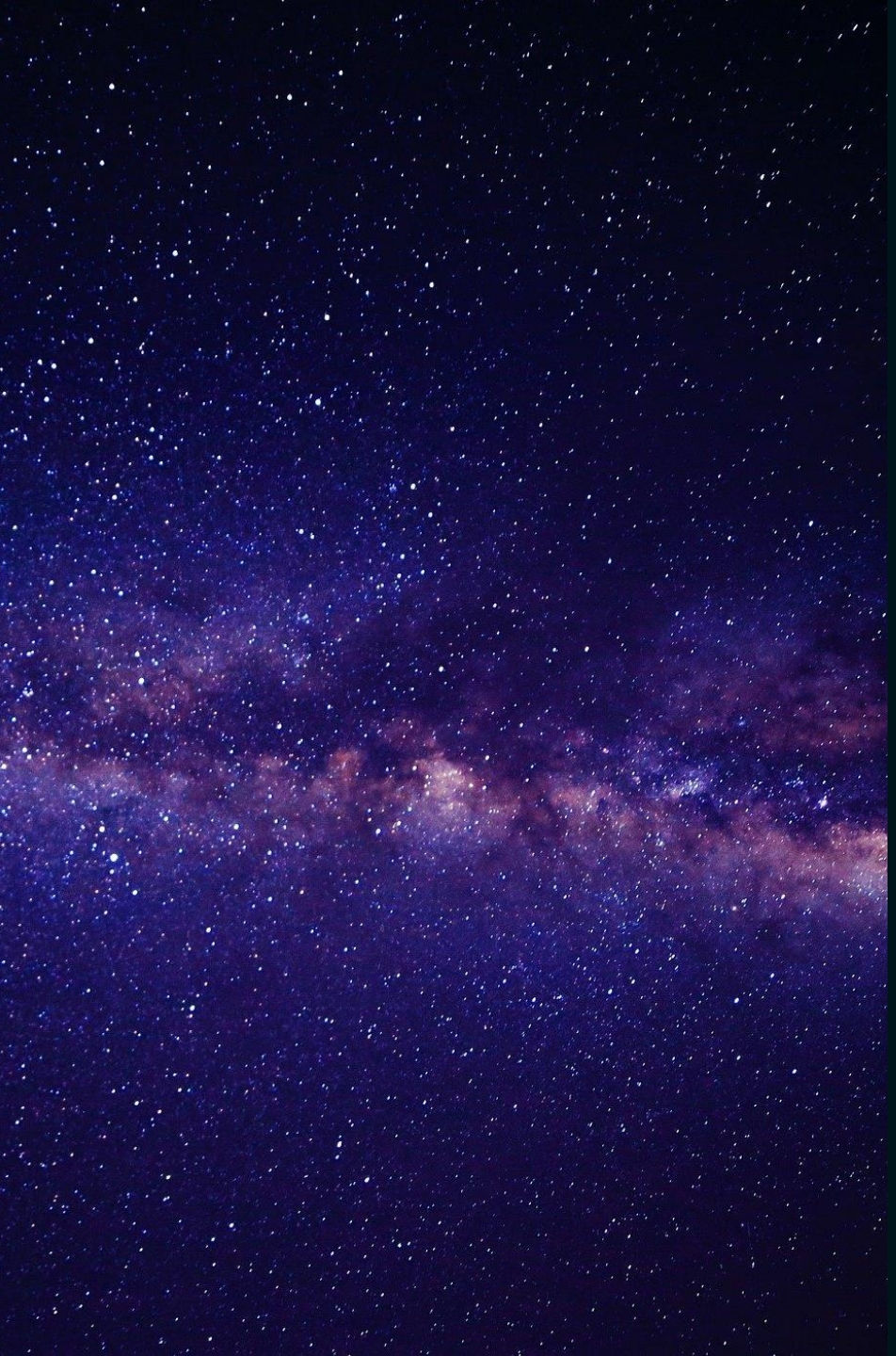


Rangify?

ranges are algorithms are loops

Rangify?

```cpp
const auto modify = [](const double elem) { return (elem * 2.0) / 10.0; };

const auto vec = std::vector{ 1.0, 2.0, 3.0 };
auto out = std::vector<double>(vec.size());

std::transform(vec.begin(), vec.end(), out.begin(), modify);

std::ranges::transform(vec, out.begin(), modify);

std::ranges::copy(vec | std::views::transform(modify), out.begin());

const auto out2 = vec | ranges::views::transform(modify) | ranges::to<std::vector>();
```



Examples

Max number

```cpp
auto max_number(const std::vector<std::string>& numbers)
{
    auto max = std::numeric_limits<int>::min();
    for (const auto& number : numbers)
    {
        max = std::max(std::stoi(number), max);
    }

    return max;
}

auto max_number(const std::vector<std::string>& numbers)
{
    return std::ranges::max(numbers
        | std::views::transform([](const auto& number) { return std::stoi(number); }));
}
```

Sliding mean

```cpp
auto sliding_mean(const std::span<const double> rng)
{
    auto out = std::vector<double>(rng.size() - 4);
    for (size_t i = 2; i < rng.size() - 2; ++i)
    {
        out[i - 2] = mean(
            std::array<double, 5>{ rng[i - 2], rng[i - 1], rng[i], rng[i + 1], rng[i + 2] });
    }

    return out;
}

auto sliding_mean(const std::span<const double> rng)
{
    return rng
        | ranges::views::sliding(5)
        | ranges::views::transform(mean)
        | ranges::to<std::vector>();
}
```

Subtract mean

```cpp
void subtract_mean(
    const std::vector<double>& column_mean,
    boost::multi_array_ref<double, 2> matrix)
{
    for (auto row : matrix)
        for (size_t i = 0; i < column_mean.size(); ++i)
            row[i] -= column_mean[i];
}

void subtract_mean(
    const std::vector<double>& column_mean,
    boost::multi_array_ref<double, 2> matrix)
{
    std::ranges::transform(std::span(matrix.origin(), matrix.num_elements()),
        // matrix | ranges::views::join()
        column_mean | ranges::views::cycle,
        matrix.origin(),
        [](const auto elem, const auto mean) { return elem - mean; });
}
```

Arange

```cpp
auto arange(size_t len, double start, double step)
{
    auto out = std::vector<double>(len, start);
    for (size_t i = 0; i < len; ++i)
    {
        out[i] += static_cast<double>(i) * step;
    }

    return out;
}

auto arange(size_t len, double start, double step)
{
    return ranges::views::ints(0, static_cast<int>(len))
        | ranges::views::transform([start, step](const auto x) { return x * step + start; })
        | ranges::to<std::vector>();
}
```

Sum non defective values

```
struct data
{
    bool is_defective;
    int value;
};

auto sum_non_defective(const std::vector<data>& range)
{
    auto sum = 0;
    for (size_t i = 0; i < range.size(); ++i)
    {
        if (!range[i].is_defective)
        {
            sum += range[i].value;
        }
    }

    return sum;
}
```

Sum non defective values

```
auto sum_non_defective(const std::vector<data>& range)
{
    auto out = 0;
    for (size_t i = 0; i < range.size(); ++i)
    {
        if (!range[i].is_defective)
        {
            out += range[i].value;
        }
    }

    return out;
}

auto sum_non_defective(const std::vector<data>& range)
{
    return ranges::accumulate(range | std::views::filter(std::not_fn(&data::is_defective))
                                    | std::views::transform(&data::value), 0);
}
```

Sum non defective values

```
auto sum_non_defective(const std::vector<data>& range)
{
    auto out = 0;
    for (size_t i = 0; i < range.size(); ++i)
    {
        if (!range[i].is_defective)
        {
            out += range[i].value;
        }
    }

    return out;
}

auto sum_non_defective(const std::vector<data>& range)
{
    return ranges::accumulate(range | std::views::filter(std::not_fn(&data::is_defective)),
                              0, std::plus{}, &data::value);
}
```

Sum non defective values

```
auto sum_non_defective(const std::vector<data>& range)
{
    return ranges::accumulate(range | std::views::filter(std::not_fn(&data::is_defective))
                                    | std::views::transform(&data::value), 0);
}

auto sum_non_defective(const std::vector<data>& range)
{
    return ranges::accumulate(range | std::views::filter(std::not_fn(&data::is_defective)),
                              0, std::plus{}, &data::value);
}
```

Length calculation

```
auto calc_length(const std::vector<double>& x,
                 const std::vector<double>& y,
                 const std::vector<double>& z)
{
    auto length = std::vector<double>(x.size());
    for (size_t i = 0; i < x.size(); ++i)
        length[i] = std::sqrt(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);

    return length;
}

auto calc_length(const std::vector<double>& x,
                 const std::vector<double>& y,
                 const std::vector<double>& z)
{
    auto length = std::vector<double>(x.size());
    for (auto&& [x_, y_, z_, l] : ranges::view::zip(x, y, z, length))
        l = std::sqrt(x_ * x_ + y_ * y_ + z_ * z_);

    return length;
}
```

Length calculation

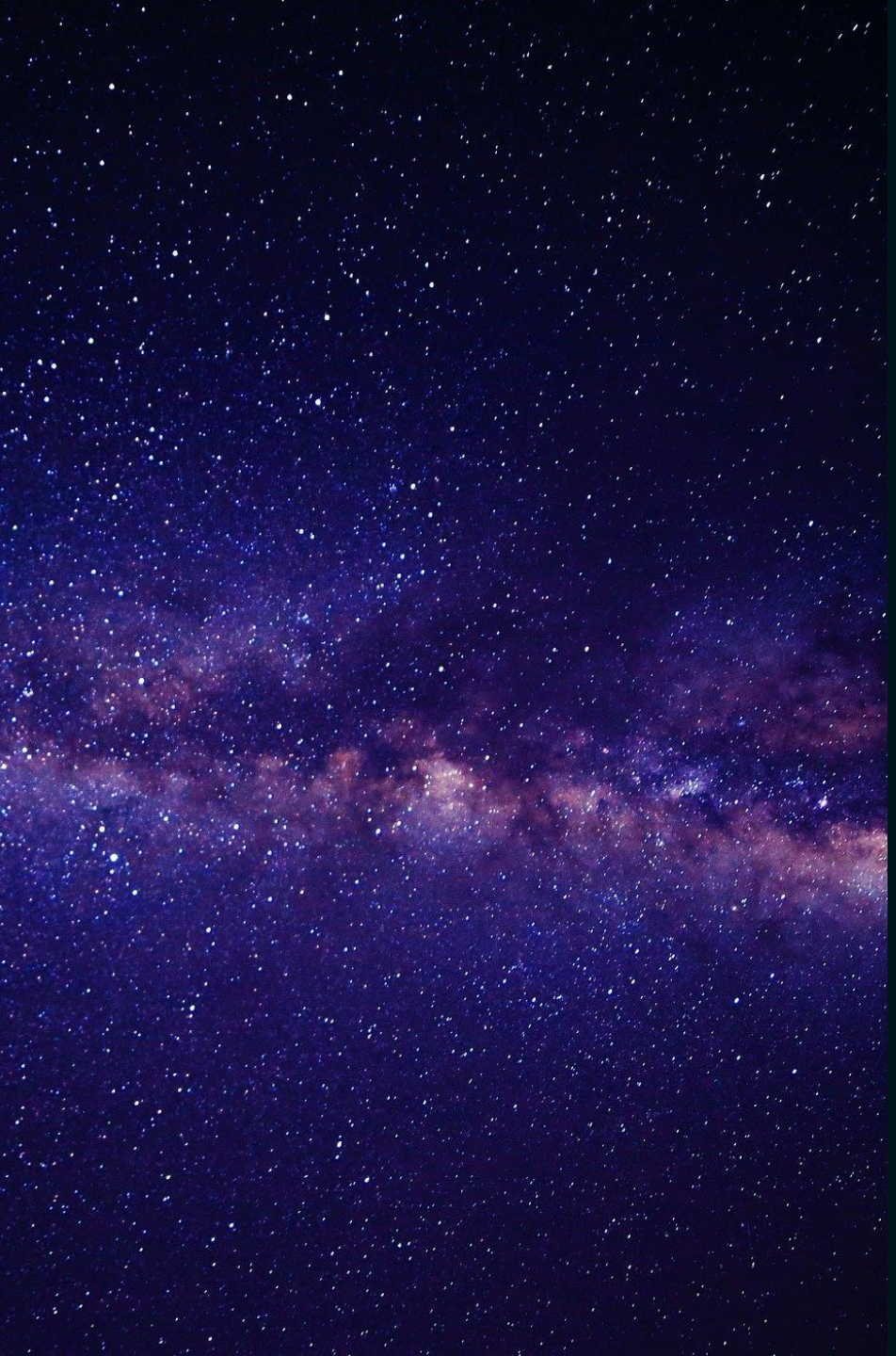
```cpp
auto calc_length(const std::vector<double>& x,
                 const std::vector<double>& y,
                 const std::vector<double>& z)
{
    auto length = std::vector<double>(x.size());
    for (size_t i = 0; i < x.size(); ++i)
        length[i] = std::sqrt(x[i] * x[i] + y[i] * y[i] + z[i] * z[i]);

    return length;
}

auto calc_length(const std::vector<double>& x,
                 const std::vector<double>& y,
                 const std::vector<double>& z)
{
    return ranges::view::zip(x, y, z) | ranges::view::transform([](const auto coordinates)
        {
            const auto& [x_, y_, z_] = coordinates;
            return std::sqrt(x_ * x_ + y_ * y_ + z_ * z_);
        }) | ranges::to<std::vector>;
}
```

Length calculation

```
auto calc_length(const std::vector<double>& x,
                 const std::vector<double>& y,
                 const std::vector<double>& z)
{
    auto length = std::vector<double>(x.size());
    for (auto&& [x_, y_, z_, l] : ranges::view::zip(x, y, z, length))
        l = std::sqrt(x_ * x_ + y_ * y_ + z_ * z_);

    return length;
}

auto calc_length(const std::vector<double>& x,
                 const std::vector<double>& y,
                 const std::vector<double>& z)
{
    return ranges::view::zip(x, y, z) | ranges::view::transform([](const auto coordinates)
        {
            const auto& [x_, y_, z_] = coordinates;
            return std::sqrt(x_ * x_ + y_ * y_ + z_ * z_);
        }) | ranges::to<std::vector>;
}
```

Index handling

```cpp
auto index_handling(const std::vector<std::vector<size_t>>& index)
{
    std::vector<size_t> out;
    for (size_t i = 0; i < index.size(); ++i)
        for (const auto idx : index[i])
            if (idx != i)
                out.push_back(idx);

    return out;
}

auto index_handling(const std::vector<std::vector<size_t>>& index)
{
    return ranges::view::enumerate(index)
        | ranges::view::transform([](const auto& indices)
            {
                const auto& [i, idx_rng] = indices;
                return idx_rng | ranges::view::filter([=](const auto idx) { return i != idx; });
            })
        | ranges::view::join
        | ranges::to<std::vector>;
}
```



Complex Example

Hondt Method

	party 1 votes: 110		**party 2 votes: 85**		**party 3 votes: 35**	
1	(1)	110 / 1 = 110	(2)	85 / 1 = 85	(6)	35 / 1 = 35
2	(3)	110 / 2 = 55	(4)	85 / 2 = 42.5		35 / 2 = 17.5
3	(5)	110 / 3 = 36.66	(7)	85 / 3 = 28.33		35 / 3 = 11.66
4		110 / 4 = 27.5		85 / 4 = 21.25		35 / 4 = 8.75
5		110 / 5 = 22		85 / 5 = 17		35 / 5 = 7
6		110 / 6 = 18.33		85 / 6 = 14.16		35 / 6 = 5.83
7		110 / 7 = 15.71		85 / 7 = 12.14		35 / 7 = 5
	seats: 3		**seats: 3**		**seats: 1**	

Hondt Method – pre ranges / C++17

C++17

```
auto hondt_method(
    const std::map<std::string, int>& votes_per_party,
    const int total_number_of_seats)
{
}
```

Hondt Method – pre ranges / C++17

	party 1 votes: 110		party 2 votes: 85		party 3 votes: 35	
1		110 / 1 = 110		85 / 1 = 85		35 / 1 = 35
2		110 / 2 = 55		85 / 2 = 42.5		35 / 2 = 17.5
3		110 / 3 = 36.66		85 / 3 = 28.33		35 / 3 = 11.66
4		110 / 4 = 27.5		85 / 4 = 21.25		35 / 4 = 8.75
5		110 / 5 = 22		85 / 5 = 17		35 / 5 = 7
6		110 / 6 = 18.33		85 / 6 = 14.16		35 / 6 = 5.83
7		110 / 7 = 15.71		85 / 7 = 12.14		35 / 7 = 5

Hondt Method – pre ranges / C++17

C++17

```
auto hondt_method(
    const std::map<std::string, int>& votes_per_party,
    const int total_number_of_seats)
{
    auto proportional_votes = std::vector<std::pair<std::string, double>>();
    for (int i = 1; i < total_number_of_seats + 1; ++i)
    {
        for (const auto& [party, number_of_votes] : votes_per_party)
        {
            proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });
        }
    }
}
```

Hondt Method – pre ranges / C++17

	party 1 votes: 110		**party 2 votes: 85**		**party 3 votes: 35**	
1	(1)	110 / 1 = 110	(2)	85 / 1 = 85	(6)	35 / 1 = 35
2	(3)	110 / 2 = 55	(4)	85 / 2 = 42.5		35 / 2 = 17.5
3	(5)	110 / 3 = 36.66	(7)	85 / 3 = 28.33		35 / 3 = 11.66
4		110 / 4 = 27.5		85 / 4 = 21.25		35 / 4 = 8.75
5		110 / 5 = 22		85 / 5 = 17		35 / 5 = 7
6		110 / 6 = 18.33		85 / 6 = 14.16		35 / 6 = 5.83
7		110 / 7 = 15.71		85 / 7 = 12.14		35 / 7 = 5

Hondt Method – pre ranges / C++17

C++17

```
auto hondt_method(
    const std::map<std::string, int>& votes_per_party,
    const int  total_number_of_seats)
{
    auto proportional_votes = std::vector<std::pair<std::string, double>>();
    for (int i = 1; i < total_number_of_seats + 1; ++i)
    {
        for (const auto& [party, number_of_votes] : votes_per_party)
        {
            proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });
        }
    }

    std::sort(proportional_votes.begin(),
              proportional_votes.end(),
              [](const auto& rhs, const auto& lhs)
    {
        return rhs.second > lhs.second;
    });
    proportional_votes.resize(total_number_of_seats);
}
```

Hondt Method – pre ranges / C++17

	party 1 votes: 110		**party 2 votes: 85**		**party 3 votes: 35**	
1	(1)	110 / 1 = 110	(2)	85 / 1 = 85	(6)	35 / 1 = 35
2	(3)	110 / 2 = 55	(4)	85 / 2 = 42.5		35 / 2 = 17.5
3	(5)	110 / 3 = 36.66	(7)	85 / 3 = 28.33		35 / 3 = 11.66
4		110 / 4 = 27.5		85 / 4 = 21.25		35 / 4 = 8.75
5		110 / 5 = 22		85 / 5 = 17		35 / 5 = 7
6		110 / 6 = 18.33		85 / 6 = 14.16		35 / 6 = 5.83
7		110 / 7 = 15.71		85 / 7 = 12.14		35 / 7 = 5
	seats: 3		**seats: 3**		**seats: 1**	

Hondt Method – pre ranges / C++17

C++17

```
auto hondt_method(
    const std::map<std::string, int>& votes_per_party,
    const int total_number_of_seats)
{
    ...

    proportional_votes.resize(number_of_seats);

    auto distribution = std::map<std::string, int>();
    for (const auto& [party, number_of_votes] : votes_per_party)
    {
        const auto count = std::count_if(proportional_votes.begin(),
                                         proportional_votes.end(),
                                         [&](const auto& votes)
        {
            return votes.first == party;
        });
        distribution.insert({ party, count });
    }

    return distribution;
}
```

C++17

```cpp
auto hondt_method(const std::map<std::string, int>& votes_per_party, const int total_number_of_seats)
{
    auto proportional_votes = std::vector<std::pair<std::string, double>>();
    for (int i = 1; i < total_number_of_seats + 1; ++i)
    {
        for (const auto& [party, number_of_votes] : votes_per_party)
        {
            proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });
        }
    }
```

calculate proportional votes

```cpp
    std::sort(proportional_votes.begin(), proportional_votes.end(), [](const auto& rhs, const auto& lhs)
    {
        return rhs.second > lhs.second;
    });
    proportional_votes.resize(total_number_of_seats);
```

sort and cut

```cpp
    auto distribution = std::map<std::string, int>();
    for (const auto& [party, number_of_votes] : votes_per_party)
    {
        const auto count = std::count_if(proportional_votes.begin(), proportional_votes.end(),
                                         [&](const auto& votes)
        {
            return votes.first == party;
        });
        distribution.insert({ party, count });
    }
```

count seats per party

```cpp
    return distribution;
}
```

Hondt Method – rangified

C++17

```cpp
auto proportional_votes = std::vector<std::pair<std::string, double>>();
for (int i = 1; i < total_number_of_seats + 1; ++i)
{
    for (const auto& [party, number_of_votes] : votes_per_party)
    {
        proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });
    }
}
```

C++20

```cpp
auto seat_divisors = ranges::views::ints(1, total_number_of_seats + 1);
auto proportional_votes = ranges::views::cartesian_product(votes_per_party, seat_divisors)
```

Hondt Method – rangified

```
{ { “party_1“, 110 }, 1 },
{ { “party_1“, 110 }, 2 }, { { “party_2“, 85 }, 2 }, { { “party_3“, 35 }, 2 },
{ { “party_1“, 110 }, 3 }, { { “party_2“, 85 }, 3 }, { { “party_3“, 35 }, 3 },
...
```

Hondt Method – rangified

C++17

```
auto proportional_votes = std::vector<std::pair<std::string, double>>();
for (int i = 1; i < total_number_of_seats + 1; ++i)
{
    for (const auto& [party, number_of_votes] : votes_per_party)
    {
        proportional_votes.push_back({ party, static_cast<double>(number_of_votes) / i });
    }
}
```

C++20

```
const auto divide_votes_by_seat_divisors = [](const auto& votes_and_divisor)
{
    const auto& [party_and_vote, divisor] = votes_and_divisor;
    return party_and_proportion{ party_and_vote.first,
                                 party_and_vote.second / static_cast<double>(divisor) };
};

auto seat_divisors = ranges::views::ints(1, total_number_of_seats + 1);
auto proportional_votes = ranges::views::cartesian_product(votes_per_party, seat_divisors)
    | ranges::views::transform(divide_votes_by_seat_divisors)
    | ranges::to<std::vector>();
```

Hondt Method – rangified

C++17

```
std::sort(proportional_votes.begin(),
          proportional_votes.end(),
          [](const auto& rhs, const auto& lhs)
{
    return rhs.second > lhs.second;
});
proportional_votes.resize(total_number_of_seats);
```

C++20

```
std::ranges::sort(proportional_votes, [](const auto& rhs, const auto& lhs)
{
    return rhs.second > lhs.second;
});
proportional_votes.resize(total_number_of_seats);
```

Hondt Method – rangified

C++17

```
std::sort(proportional_votes.begin(),
          proportional_votes.end(),
          [](const auto& rhs, const auto& lhs)
{
    return rhs.second > lhs.second;
});
proportional_votes.resize(total_number_of_seats);
```

C++20

```
std::ranges::sort(proportional_votes, [](const auto& rhs, const auto& lhs)
{
    return rhs.proportion > lhs.proportion;
});
proportional_votes.resize(total_number_of_seats);
```

Hondt Method – rangified

C++17

```
std::sort(proportional_votes.begin(),
          proportional_votes.end(),
          [](const auto& rhs, const auto& lhs)
{
    return rhs.second > lhs.second;
});
proportional_votes.resize(total_number_of_seats);
```

C++20

```
std::ranges::sort(proportional_votes, std::greater(), &party_and_proportion::proportion);
proportional_votes.resize(total_number_of_seats);
```

Hondt Method – rangified

C++17

```
std::sort(proportional_votes.begin(),
          proportional_votes.end(),
          [](const auto& rhs, const auto& lhs)
{
    return rhs.second > lhs.second;
});
proportional_votes.resize(total_number_of_seats);
```

C++20

```
proportional_votes |= ranges::actions::sort(std::greater(),
                        &party_and_proportion::proportion);
proportional_votes.resize(total_number_of_seats);
```

Hondt Method – rangified

C++17

```
std::sort(proportional_votes.begin(),
          proportional_votes.end(),
          [](const auto& rhs, const auto& lhs)
{
    return rhs.second > lhs.second;
});
proportional_votes.resize(total_number_of_seats);
```

C++20

```
proportional_votes |= ranges::actions::sort(std::greater(),
                        &party_and_proportion::proportion);
```

Hondt Method – rangified

C++17

```
auto distribution = std::map<std::string, int>();
for (const auto& [party, number_of_votes] : votes_per_party)
{
    const auto count = std::count_if(proportional_votes.begin(), proportional_votes.end(),
                                     [&](const auto& votes)
    {
        return votes.first == party;
    });
    distribution.insert({ party, count });
}
```

C++20

```
return votes_per_party
    | ranges::views::keys
    | ranges::views::transform(count_seats_per_party(proportional_votes, total_number_of_seats))
    | ranges::to<std::map>();
```

Hondt Method – rangified

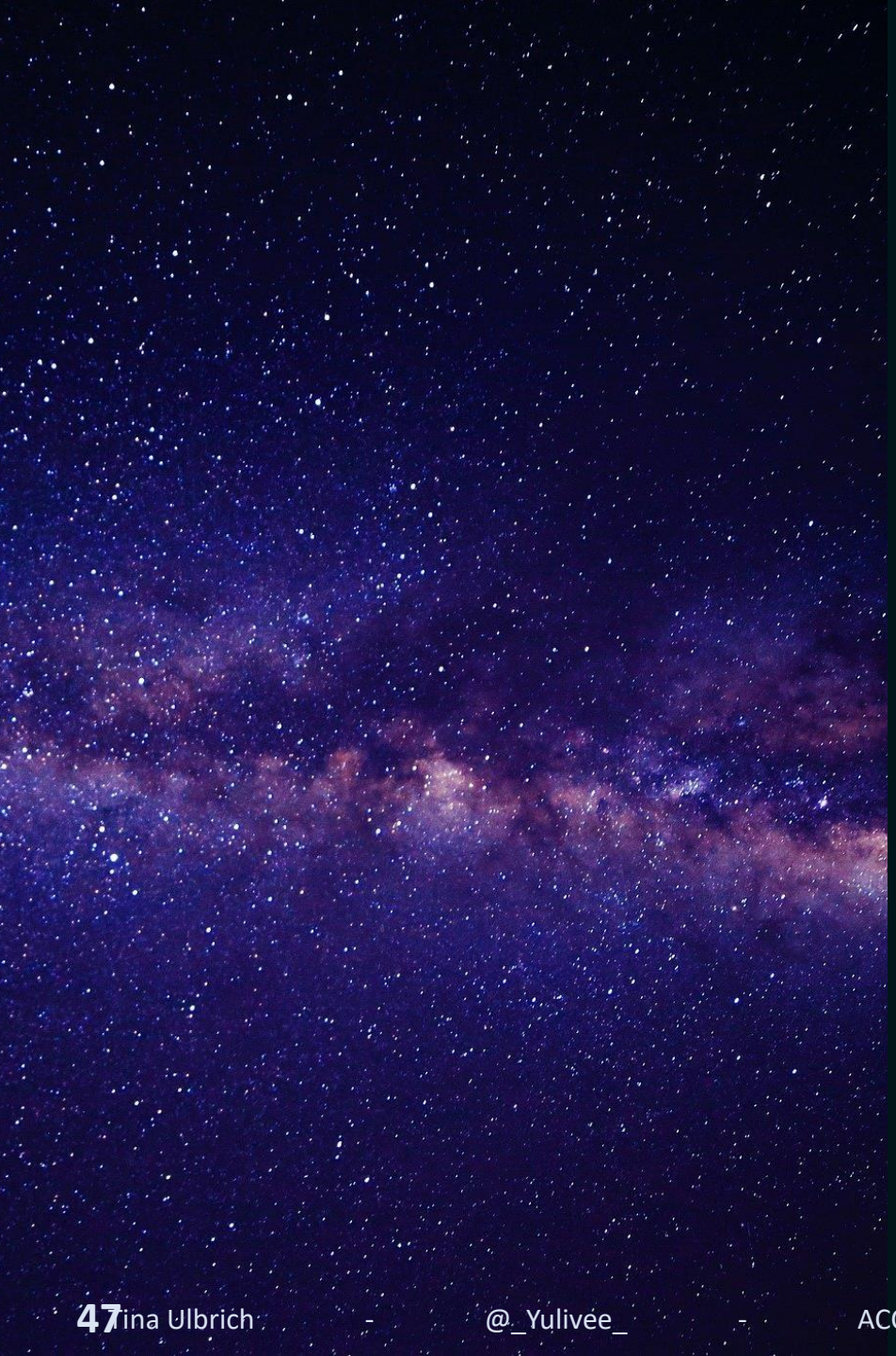
C++20

```
auto calculate_number_of_seats(
    const std::vector<party_and_proportion>& proportional_votes,
    const int total_number_of_seats,
    const std::string_view party)
{
    return ranges::count_if(proportional_votes | ranges::views::take(total_number_of_seats),
                            [&](const auto& party_and_votes)
    {
        return party_and_votes.party == party;
    });
}

auto count_seats_per_party(
    const std::vector<party_and_proportion>& proportional_votes,
    const int total_number_of_seats)
{
    return [&, number_of_seats](const auto& party)
    {
        const auto seats =
              calculate_number_of_seats(proportional_votes, total_number_of_seats, party);
        return std::pair{ party, seats };
    };
}
```

Hondt Method – rangified

C++20

```
auto hondt_method(
    const std::map<std::string, int>& votes_per_party,
    const int total_number_of_seats)
{
    auto seat_divisors = ranges::views::ints(1, total_number_of_seats + 1);
    auto proportional_votes = ranges::views::cartesian_product(votes_per_party, seat_divisors)
        | ranges::views::transform(divide_votes_by_seat_divisors)
        | ranges::to<std::vector>();

    proportional_votes |= ranges::actions::sort(std::greater(),
                                                &party_and_proportion::proportion);
    return votes_per_party
        | ranges::views::keys
        | ranges::views::transform(count_seats_per_party(proportional_votes,
                                                         total_number_of_seats))
        | ranges::to<std::map>();
}
```

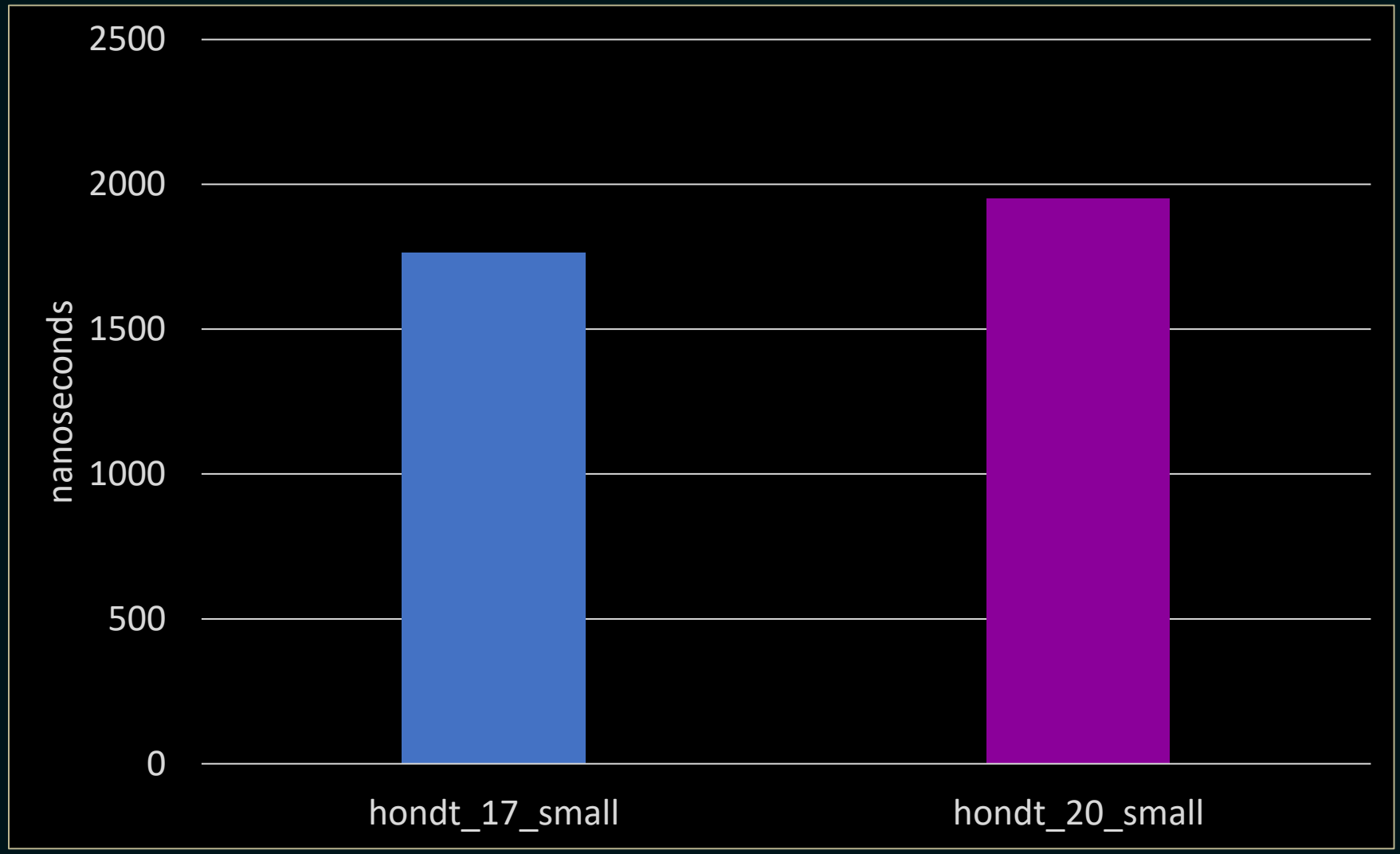


Performance

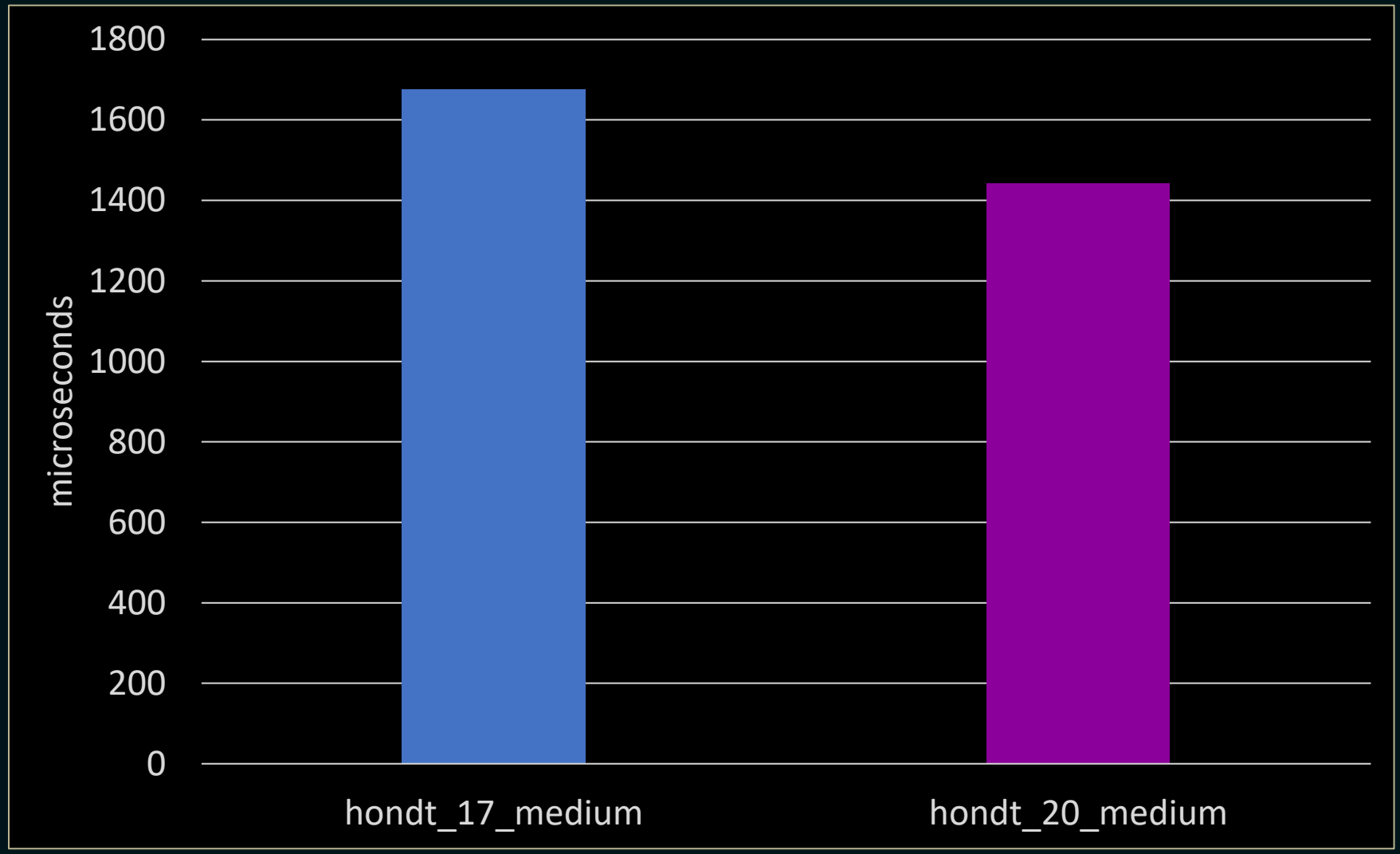
Performance Hondt Method

```
Run on (4 X 2400 MHz CPU s)
CPU Caches:
  L1 Data 32 KiB (x2)
  L1 Instruction 32 KiB (x2)
  L2 Unified 256 KiB (x2)
  L3 Unified 3072 KiB (x1)
-----------------------------------------------------------
Benchmark                 Time             CPU   Iterations
-----------------------------------------------------------
hondt_17_small         1747 ns         1765 ns       407273
hondt_20_small         1965 ns         1950 ns       344615
hondt_17_medium     1684946 ns      1675603 ns          373
hondt_20_medium     1434124 ns      1443273 ns          498
hondt_17_large    869263800 ns    875000000 ns            1
hondt_20_large    836679300 ns    843750000 ns            1
```

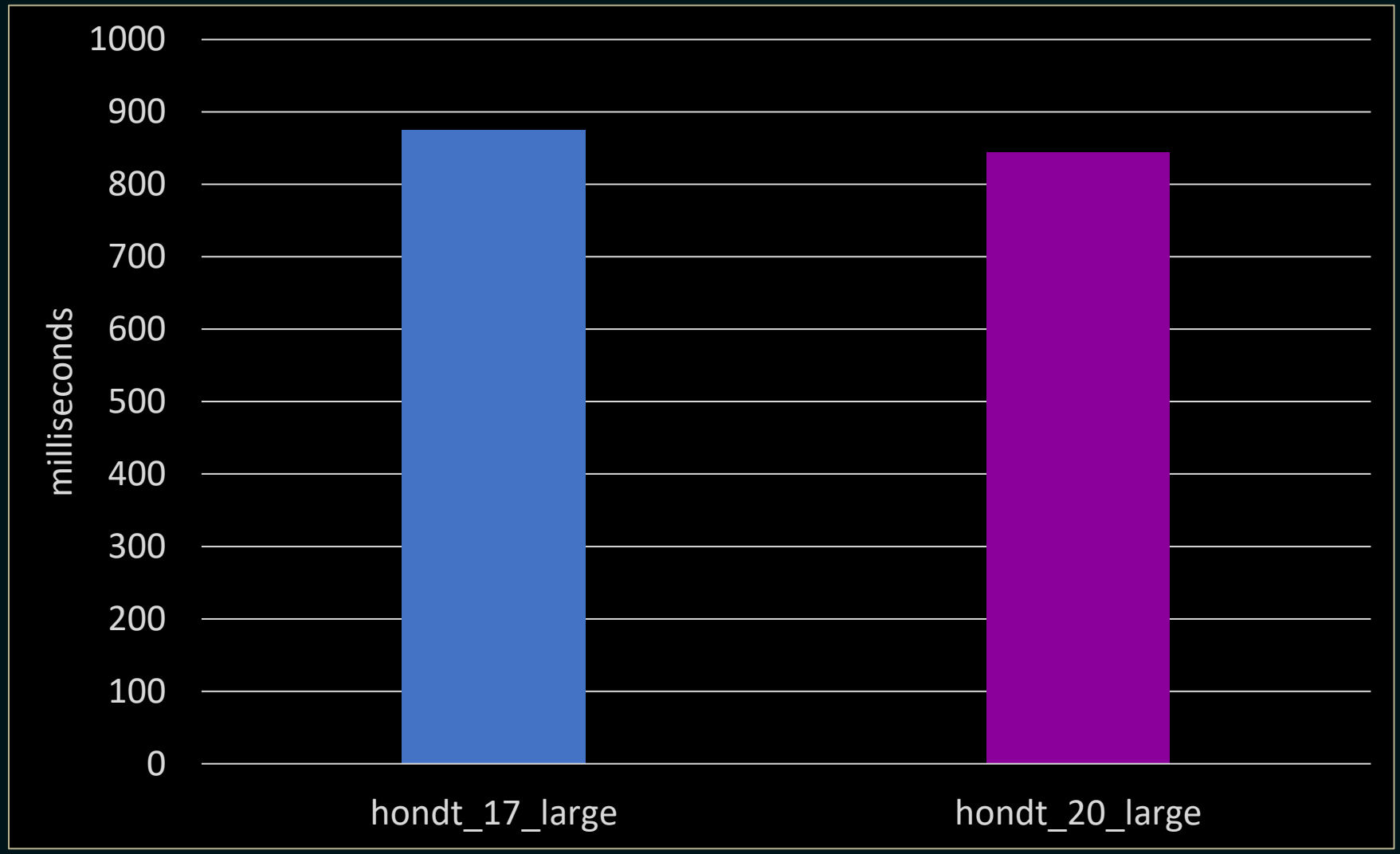
Performance Hondt Method

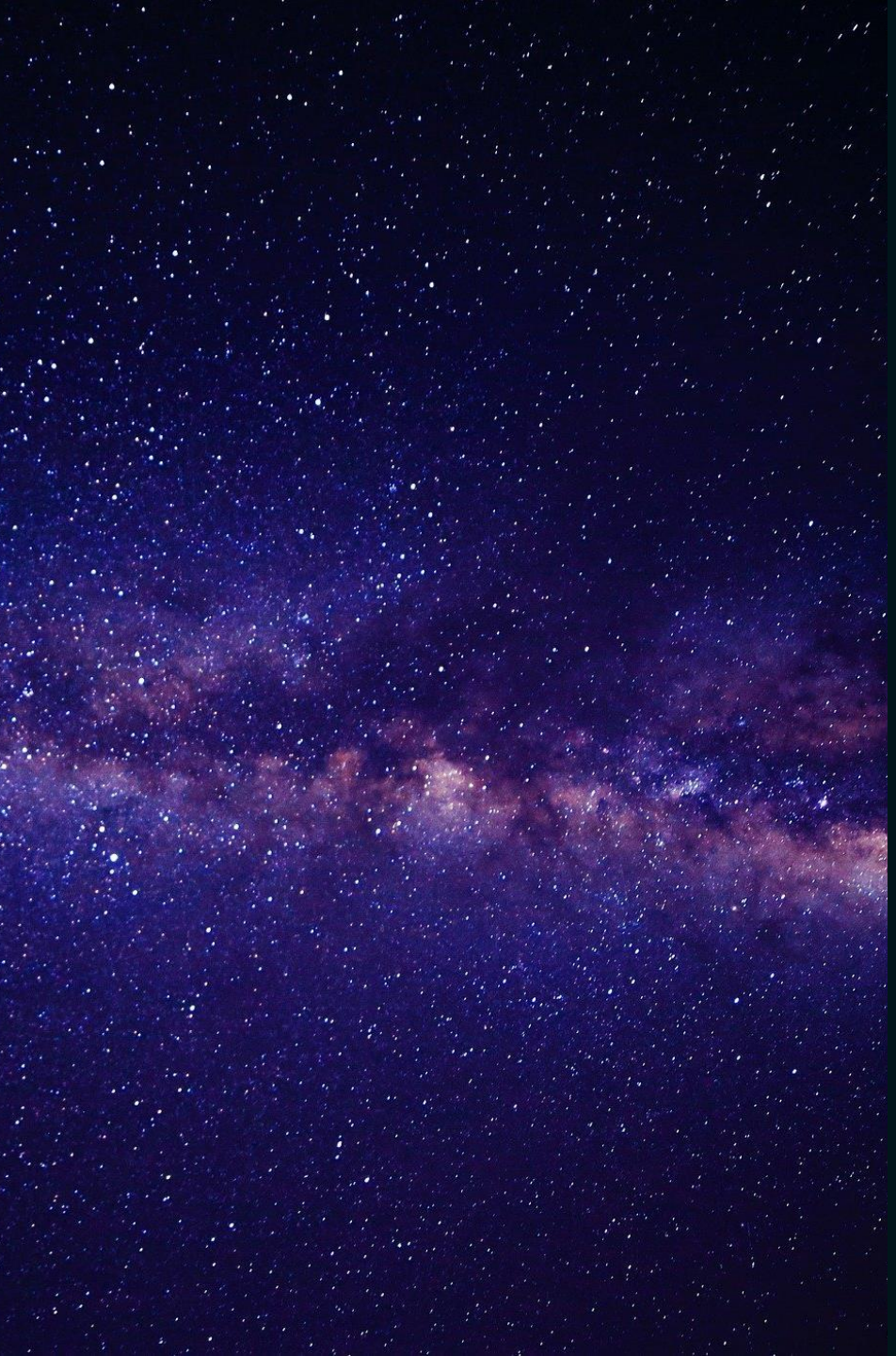


Performance Hondt Method



Performance Hondt Method





Summary

Questions?

Tina Ulbrich
@_Yulivee_
ROSEN Technology and Research Center GmbH

Talks

- https://www.youtube.com/c/SyBrandPlusCats/videos
- https://youtu.be/YWayW5ePpkY
- https://youtu.be/d_E-VLyUnzc
- https://youtu.be/d9qDEEJFwNc
- https://youtu.be/mFUXNMfaciE

Libraries

- https://en.cppreference.com/w/cpp/ranges
- https://github.com/cor3ntin/rangesnext
- https://github.com/tcbrindle/NanoRange
- https://github.com/TartanLlama/ranges
- https://github.com/ericniebler/range-v3
- https://www.boost.org/doc/libs/1_75_0/libs/range/doc/html/index.html

Resources

- https://en.wikipedia.org/wiki/D%27Hondt_method
- https://en.cppreference.com/w/cpp/ranges
- https://ericniebler.github.io/range-v3/
- https://github.com/cor3ntin/rangesnext