Most of the code is about this size...

... and goes roughly over here,

And there will be some with this size

But nothing is smaller than this

... over here,

... and about here

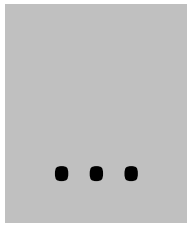
Angry Nerd Threatens
ISO Committee

How To Poison The
Front Row Unnoticed

Cpp North

… + CTAD + NTTP = 

Cpp North

 + CTAD + NTTP = 🤩

Variadic Templates
Parameter Packs

Variadic Templates C++11

```
template<typename... Ts>
struct type { };

  type<>{};
  type<int>{};
  type<int, double>{};
  type<int, double, std::string>{};
```

Variadic Templates C++11

```
template<typename... Args>
auto f(Args... args) { }

  f();
  f(1);
  f(1, 2.0);
  f(1, 2.0, "3");

template<typename... Args>
auto f(Args const&... args) { }

template<typename... Args>
auto f(Args const*... args) { }
```

Variadic Templates C++11

```
template<typename... Args>
auto f(Args... args)
{
  auto x = type<Args...>{};
  return g(x, args...);
}

template<typename... Args>
auto f(Args... args)
{
  auto x = type<std::remove_cvref_t<Args>...>{};
  return g(x, h(args)...);
}
```


… + CTAD + NTTP = 😻

Variadic Templates C++11

```
template<typename... Args>
auto f(Args... args)
{
  auto x = type<Args...>{};
  return g(x, args...);
}

template<typename... Args>
auto f(Args... args)
{
  auto x = type<std::remove_cvref_t<Args>...>{};
  return g(x, h(args...));
}
```

Variadic Templates C++11

```cpp
template<typename... Args>
auto f(Args... args)
{
  // g(h(args_0 + args_0), ..., h(args_n + args_n))
  return g(h(args + args)...);
}
```


… + CTAD + NTTP = 😻

Variadic Templates C++11

```
template<typename... Args>
auto f(Args... args)
{
  // g(h(args_1, args_1, args_2, ..., args_n),
  //   h(args_2, args_1, args_2, ..., args_n),
  //       ⋮
  //   h(args_n, args_1, args_2, ..., args_n))
  return g(h(args, args...)...);
}
```


… + CTAD + NTTP = 😻

Variadic Templates C++11

```
template<typename... Ts>
struct type { };

template<typename... Args>
auto f(Args... args) { }
auto f(auto... args) { }

[](auto... args) { }
[]<typename... Args>(Args... args) { }

template<typename... Ts>
using pointers = type<std::add_pointer_t<Ts>...>;

template<typename... Ts>
auto var = type<Ts...>{};
```


… + CTAD + NTTP = 😻

Fold Expressions C++17

```
template<typename... Args>
auto sum(Args... args)
{
  return (... + args);
}
```

$$(\ldots \otimes E)$$

$$((((E_1 \otimes E_2) \otimes E_3) \otimes \ldots) \otimes E_n)$$

$$(E \otimes \ldots)$$

$$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes E_n))))$$

Fold Expressions C++17

```
template<typename... Args>
auto sum(Args... args)
{
  return (0 + ... + args);
}
```

$$(\ldots \otimes E)$$

$$((((E_1 \otimes E_2) \otimes E_3) \otimes \ldots) \otimes E_n)$$

$$(E \otimes \ldots)$$

$$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes E_n))))$$

$$(\underline{I \otimes} \ldots \otimes E)$$

$$(((((\underline{I \otimes} E_1) \otimes E_2) \otimes \ldots) \otimes E_n)$$

$$(E \otimes \ldots \underline{\otimes I})$$

$$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes (E_n \underline{\otimes I})))))$$

$$\otimes$$

$$+ \; - \; * \; / \; \% \; \hat{} \; \& \; | \; = \; < \; > \; << \; >> \; += \; -= \; *= \; /= \; \%=$$

$$\hat{}= \; \&= \; |= \; <<= \; >>= \; == \; != \; <= \; >= \; \&\& \; || \; , \; .* \; ->*$$

Fold Expressions C++17

```
template<typename... Args>
auto sum(Args... args)
{
  return (0 + ... + args);
}
```

$$(\ldots \otimes E)$$

$$((((E_1 \otimes E_2) \otimes E_3) \otimes \ldots) \otimes E_n)$$

$$(E \otimes \ldots)$$

$$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes E_n))))$$

$$(\underline{I \otimes} \ldots \otimes E)$$

$$(((((\underline{I \otimes} E_1) \otimes E_2) \otimes \ldots) \otimes E_n)$$

$$(E \otimes \ldots \underline{\otimes I})$$

$$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes (E_n \underline{\otimes I})))))$$

$$\otimes$$

$$+ \ - \ * \ / \ \% \ \hat{} \ \& \ | \ = \ < \ > \ << \ >> \ += \ -= \ *= \ /= \ \%=$$

$$\hat{}= \ \&= \ |= \ <<= \ >>= \ == \ != \ <= \ >= \ \&\& \ || \ , \ .* \ ->*$$

… + CTAD + NTTP = 😻

Fold Expressions C++17

```
template<typename... Args>
auto sum(Args... args)
{
  return (0 == ... == args);
}
```

$$(\ldots \otimes E)$$

$$((((E_1 \otimes E_2) \otimes E_3) \otimes \ldots) \otimes E_n)$$

$$(E \otimes \ldots)$$

$$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes E_n))))$$

$$(I \otimes \ldots \otimes E)$$

$$(((((I \otimes E_1) \otimes E_2) \otimes \ldots) \otimes E_n)$$

$$(E \otimes \ldots \otimes I)$$

$$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes (E_n \otimes I)))))$$

$$\otimes$$

$$+ \ - \ * \ / \ \% \ \hat{} \ \& \ | \ = \ < \ > \ << \ >> \ += \ -= \ *= \ /= \ \%=$$

$$\hat{}= \ \&= \ |= \ <<= \ >>= \ == \ != \ <= \ >= \ \&\& \ || \ , \ .* \ ->*$$

… + CTAD + NTTP = 😻

Fold Expressions C++17

```
template<typename... Args>
auto sum(Args... args)
{
  return (... && (args == 0));
}
```

$$(\ldots \otimes E)$$

$$((((E_1 \otimes E_2) \otimes E_3) \otimes \ldots) \otimes E_n)$$

$$(E \otimes \ldots)$$

$$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes E_n))))$$

$$(\underline{I \otimes} \ldots \otimes E)$$

$$(((((\underline{I \otimes} E_1) \otimes E_2) \otimes \ldots) \otimes E_n)$$

$$(E \otimes \ldots \underline{\otimes I})$$

$$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes (E_n \underline{\otimes I})))))$$

$$\otimes$$

$$+ \; - \; * \; / \; \% \; \hat{} \; \& \; | \; = \; < \; > \; << \; >> \; += \; -= \; *= \; /= \; \%=$$

$$\hat{}= \; \&= \; |= \; <<= \; >>= \; == \; != \; <= \; >= \; \&\& \; || \; , \; .* \; ->*$$

Fold Expressions C++17

Fold	⊗	n = 0	n = 1
(E ⊗ …) (… ⊗ E)	&& || , other	false true void() ❌	E_1
(E ⊗ … ⊗ I) (I ⊗ … ⊗ E)	all	I	$E_1 \otimes I$ $I \otimes E_1$

(… ⊗ E)

$((((E_1 \otimes E_2) \otimes E_3) \otimes \ldots) \otimes E_n)$

(E ⊗ …)

$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes E_n))))$

(I ⊗ … ⊗ E)

$(((((I \otimes E_1) \otimes E_2) \otimes \ldots) \otimes E_n)$

(E ⊗ … ⊗ I)

$(E_1 \otimes (E_2 \otimes (\ldots \otimes (E_{n-1} \otimes (E_n \otimes I)))))$

⊗

+ - * / % ^ & | = < > << >> += -= *= /= %=

^= &= |= <<= >>= == != <= >= && || , .* ->*

… + CTAD + NTTP = 😻

Fold Expressions

```
template<typename... Args>
auto sum(Args... args)
{
  return (0 + ... + [&]{ return args; }());
}
```

Fold Expressions

```
template<typename... Args>
auto sum(Args... args)
{
  return (0 + ... + [&]{ return args; }());
}
```

Fold Expressions

```
template<typename... Args>
auto sum(Args... args)
{
  (... , [&]{ /* do something with args */ }());
}
```

Fold Expressions

```cpp
template <size_t... Is, size_t... Js>
auto print_pairwise(
  std::index_sequence<Is...>,
  std::index_sequence<Js...>)
{
  (...,
    [&, i = Is]
    {
      (..., (std::cout << I << ' ' << Js << '\n'));
    }()
  );
}
print_pairwise(
  std::make_index_sequence<3>{},
  std::make_index_sequence<4>{});
```

```
I J
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
```

Fold Expressions

```
template <size_t... Is, size_t... Js>
auto print_pairwise(
  std::index_sequence<Is...>,
  std::index_sequence<Js...>)
{
  (...,
    [&, i = Is]
    {
      (..., (std::cout << I << ' ' << Js << '\n'));
    }()
  );
}
print_pairwise(
  std::make_index_sequence<3>{},
  std::make_index_sequence<4>{});
```

I	J
0	0
0	1
0	2
0	3
1	0
1	1
1	2
1	3
2	0
2	1
2	2
2	3

Fold Expressions

```
template <size_t... Is, size_t... Js>
auto print_pairwise(
  std::index_sequence<Is...>,
  std::index_sequence<Js...>)
{
  (...,
    [&, i = Is]
    {
      (..., (std::cout << I << ' ' << Js << '\n'));
    }()
  );
}
print_pairwise(
  std::make_index_sequence<3>{},
  std::make_index_sequence<4>{});
```

I	J
0	0
0	1
0	2
0	3
1	0
1	1
1	2
1	3
2	0
2	1
2	2
2	3

Fold Expressions

```
template <size_t... Is, size_t... Js>
auto print_pairwise(
  std::index_sequence<Is...>,
  std::index_sequence<Js...>)
{
  (...,
    [&, i = Is]
    {
      (..., (std::cout << i << ' ' << Js << '\n'));
    }()
  );
}
print_pairwise(
  std::make_index_sequence<3>{},
  std::make_index_sequence<4>{});
```

```
I J
0 0
0 1
0 2
0 3
1 0
1 1
1 2
1 3
2 0
2 1
2 2
2 3
```

Fold Expressions

```
template <size_t... Is, size_t... Js>
auto print_pairwise(
  std::index_sequence<Is...>,
  std::index_sequence<Js...>)
{
  (...,
    [&]
    {
      (..., (std::cout << Is << ' ' << Js << '\n'));
    }()
  );
}
print_pairwise(
  std::make_index_sequence<3>{},
  std::make_index_sequence<4>{});
```

I	J
0	0
0	1
0	2
0	3
1	0
1	1
1	2
1	3
2	0
2	1
2	2
2	3

… + CTAD + NTTP = 😻

The “C++ packs hate you”-Pattern

```cpp
template<typename... Ts>
auto f(Ts... args)
{
  [&]<size_t... Is>(std::index_sequence<Is...>)
  {
    // expand Ts/args and Is in tandem
  }
  (std::make_index_sequence<sizeof...(args)>{});
}
```

Class Template
Argument Deduction

... + CTAD + NTTP = 🤩

✅

Variadic Templates
Parameter Packs

Class Template Argument Deduction C++17

```cpp
auto v = std::vector{1, 2, 3};
auto t = std::tuple{1, 2.0, '3'};

template<typename A, typename B>
struct foo
{
  A a;
  B b;
  foo(A a, B b) : a(a), b(b) {}
};
foo{1, 2.0};
```


… + CTAD + NTTP = 😻

Class Template Argument Deduction C++17

```cpp
auto v = std::vector{1, 2, 3};
auto t = std::tuple{1, 2.0, '3'};

template<typename A, typename B>
struct foo
{
  A a;
  B b;
  foo(A a, B b) : a(a), b(b) {} // C++20
};
foo{1, 2.0};
foo{.a = 1, .b = 2.0}; // C++20
```


… + CTAD + NTTP = 😻

Class Template Argument Deduction C++17

```cpp
auto v = std::vector{1, 2, 3};
auto t = std::tuple{1, 2.0, '3'};

template<typename A, typename B>
struct foo
{
  A a;
  B b;
  foo(A a, B b) : a(a), b(b) {}
};
foo{1, 2.0};
```

```cpp
template<typename A, typename B>
foo(A, B) -> foo<std::optional<A>,
                 std::optional<B>>;

foo(int, int) -> foo<double, double>;

foo<int>{1, 2.0};

template<typename B>
using foo_int = foo<int, B>;

foo_int{1, 2.0};
```

¯_(ツ)_/¯

… + CTAD + NTTP = 😻

Class Template
Argument Deduction

... + CTAD + NTTP = 🤩

✅ Variadic Templates
Parameter Packs

✅ Non-Type
Template Parameters

Non-Type Template Parameters

```
template<int N>
struct foo {};

foo<1>{};
foo<1u>{};
foo<'2'>{};
```

Non-Type Template Parameters

```
template<auto N> // C++17
struct foo {};

foo<1>{};
foo<1u>{};
foo<'2'>{};
```

Non-Type Template Parameters C++20

```
template<auto N>
struct foo {};

foo<1>{};
foo<1u>{};
foo<'2'>{};

foo<1.f>{};
foo<2.0>{};
foo<[](int x) { return x * x;}>{};
```

```
struct point
{
  int x;
  int y;
};

foo<point{10, 20}>{};

foo<std::array{1, 2, 3}>{};

static_assert( // does NOT use operator==
  std::same_as<foo<point{10, 20}>,
               foo<point{10, 20}>>);
```

Non-Type Template Parameters

```
constexpr auto data = std::array{
  #embed "data.bin" // C23
};

template<auto Stuff>
struct foo {};

foo<data>{}; // makes a copy of data!
```

Non-Type Template Parameters

```
constexpr auto data = std::array{
  #embed "data.bin" // C23
};

template<auto& Stuff>
struct foo {};

foo<data>{}; // makes a copy of data!
```

Non-Type Template Parameters

```
constexpr auto data = std::array{
  #embed "data.bin" // C23
};

template<decltype(auto) Stuff>
struct foo {};

foo<(data)>{}; // makes a copy of data!
```

Non-Type Template Parameters

```
template<auto Stuff>
struct foo {};

template<auto Stuff>
struct bar {};

template<auto Stuff>
auto foo_to_bar(foo<Stuff>)
{
  return bar<Stuff>{}; // copy!
}
```

Non-Type Template Parameters

```
template<auto Stuff>
consteval auto ref() -> auto const&
{
  return Stuff;
}

ref<std::array{1, 2, 3}>(); // OK: returns std::array<int, 3> const&

ref<1>(); // ERROR: returning reference to local temporary object
```

Non-Type Template Parameters

```
template<auto Stuff>
consteval auto ref() -> auto const&
{
  return Stuff; <- built-in types and C arrays are always prvalues
}

ref<std::array{1, 2, 3}>(); // OK: returns std::array<int, 3> const&

ref<1>(); // ERROR: returning reference to local temporary object
```

Non-Type Template Parameters

```
template<auto Stuff>
consteval auto ref() -> decltype(auto)
{
  return (Stuff);
}

ref<std::array{1, 2, 3}>(); // OK: returns std::array<int, 3> const&

ref<1>(); // OK: returns int (by value)
```

Class Template
Argument Deduction

... + CTAD + NTTP = 🤩

✅ Variadic Templates
Parameter Packs

✅ Non-Type
Template Parameters

Thermodynamic Simulation

Ideal Gas Volume

$$V = \frac{nRT}{P}$$

V	volume
T	temperature
n	amount of substance
P	pressure
R	universal gas constant

DIPPR 105

$$\rho = \frac{a}{b^{1+\left(1-\frac{T}{c}\right)^{d}}}$$

ρ	density
T	temperature
a	parameter
b	parameter
c	parameter
d	parameter

Aly & Lee

$$C_p = a + b\left(\frac{\frac{c}{T}}{\sinh\frac{c}{T}}\right)^2 + \mathrm{d}\left(\frac{\frac{e}{T}}{\cosh\frac{e}{T}}\right)^2$$

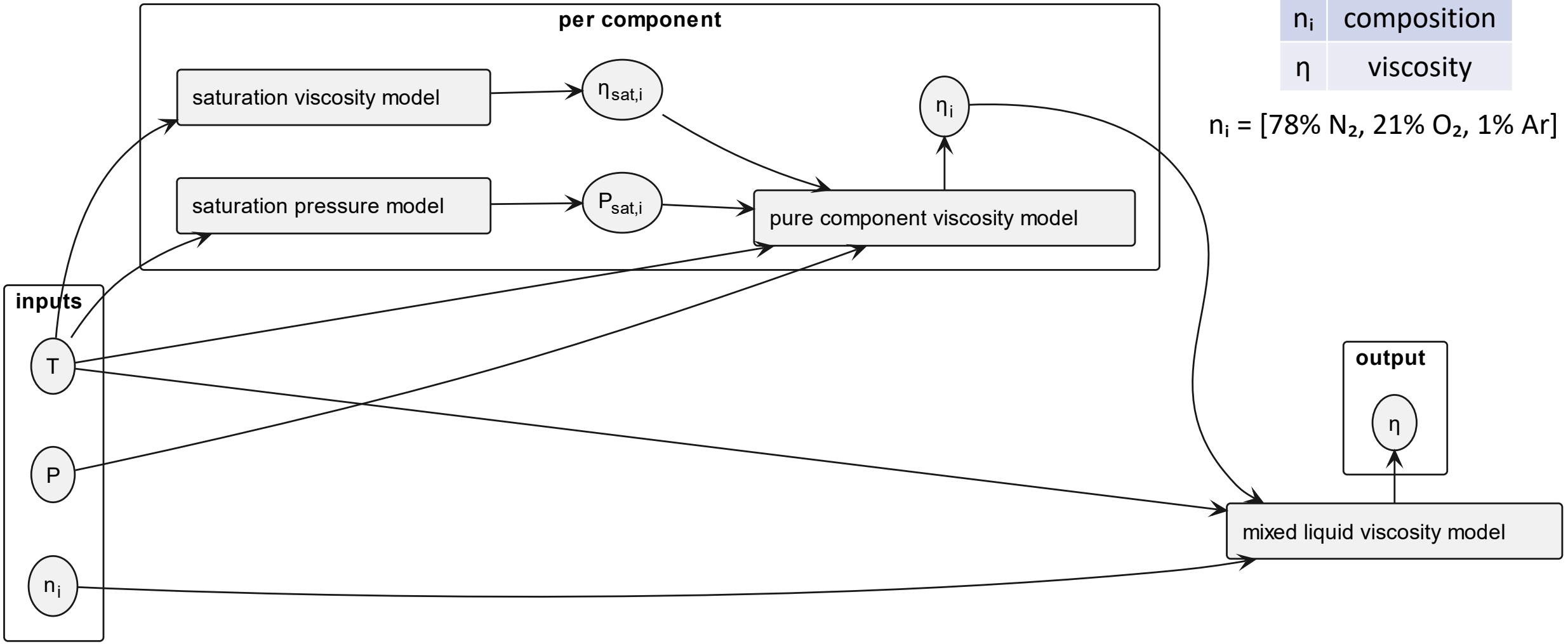
C_p	heat capacity
T	temperature
a	parameter
b	parameter
c	parameter
d	parameter
e	parameter

… + CTAD + NTTP = 😻

Thermodynamic Simulation

T	temperature
P	pressure
n_i	composition
η	viscosity

n_i = [78% N_2, 21% O_2, 1% Ar]



… + CTAD + NTTP = 😻

A Little Helper

```
template <auto... Xs>
struct tuple
{
  static consteval auto size()
    -> size_t;
  static consteval auto index_sequence()
    -> std::make_index_sequence<size()>;

  template <size_t I>
  friend consteval auto get(tuple)
    -> decltype(auto);

  template <auto... Ys>
  consteval bool operator==(tuple<Ys...>) const noexcept;
};
```

```
constexpr auto ideal_gas_volume_params = model_parameters
<
  {
    .name        = "T",
    .description = "temperature",
    .flow        = input,
    .dimension   = isq::thermodynamic_temperature[si::kelvin],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "P",
    .description = "pressure",
    .flow        = input,
    .dimension   = isq::pressure[si::pascal],
    .domain      = domain_positive,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "n",
    .description = "molar composition",
    .flow        = input,
    .dimension   = isq::amount_of_substance[si::mole],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<ncomp>,
  },
  {
    .name        = "V",
    .description = "ideal gas volume",
    .flow        = output,
    .dimension   = isq::volume[cubic(si::metre)],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  }
>;
```

```
constexpr auto ideal_gas_volume_params = model_parameters
<
  {
    .name        = "T",
    .description = "temperature",
    .flow        = input,
    .dimension   = isq::thermodynamic_temperature[si::kelvin],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "P",
    .description = "pressure",
    .flow        = input,
    .dimension   = isq::pressure[si::pascal],
    .domain      = domain_positive,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "n",
    .description = "molar composition",
    .flow        = input,
    .dimension   = isq::amount_of_substance[si::mole],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<ncomp>,
  },
  {
    .name        = "V",
    .description = "ideal gas volume",
    .flow        = output,
    .dimension   = isq::volume[cubic(si::metre)],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  }
>;
```

```
constexpr auto ideal_gas_volume_params = model_parameters
<
  {
    .name        = "T",
    .description = "temperature",
    .flow        = input,
    .dimension   = isq::thermodynamic_temperature[si::kelvin],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "P",
    .description = "pressure",
    .flow        = input,
    .dimension   = isq::pressure[si::pascal],
    .domain      = domain_positive,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "n",
    .description = "molar composition",
    .flow        = input,
    .dimension   = isq::amount_of_substance[si::mole],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<ncomp>,
  },
  {
    .name        = "V",
    .description = "ideal gas volume",
    .flow        = output,
    .dimension   = isq::volume[cubic(si::metre)],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  }
>;
```

```
constexpr auto ideal_gas_volume_params = model_parameters
<
  {
    .name        = "T",
    .description = "temperature",
    .flow        = input,
    .dimension   = isq::thermodynamic_temperature[si::kelvin],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "P",
    .description = "pressure",
    .flow        = input,
    .dimension   = isq::pressure[si::pascal],
    .domain      = domain_positive,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "n",
    .description = "molar composition",
    .flow        = input,
    .dimension   = isq::amount_of_substance[si::mole],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<ncomp>,
  },
  {
    .name        = "V",
    .description = "ideal gas volume",
    .flow        = output,
    .dimension   = isq::volume[cubic(si::metre)],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  }
>;
```

```
constexpr auto ideal_gas_volume_params = model_parameters
<
  {
    .name        = "T",
    .description = "temperature",
    .flow        = input,
    .dimension   = isq::thermodynamic_temperature[si::kelvin],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "P",
    .description = "pressure",
    .flow        = input,
    .dimension   = isq::pressure[si::pascal],
    .domain      = domain_positive,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "n",
    .description = "molar composition",
    .flow        = input,
    .dimension   = isq::amount_of_substance[si::mole],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<ncomp>,
  },
  {
    .name        = "V",
    .description = "ideal gas volume",
    .flow        = output,
    .dimension   = isq::volume[cubic(si::metre)],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  }
>;
```

```
constexpr auto ideal_gas_volume_params = model_parameters
<
  {
    .name        = "T",
    .description = "temperature",
    .flow        = input,
    .dimension   = isq::thermodynamic_temperature[si::kelvin],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "P",
    .description = "pressure",
    .flow        = input,
    .dimension   = isq::pressure[si::pascal],
    .domain      = domain_positive,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "n",
    .description = "molar composition",
    .flow        = input,
    .dimension   = isq::amount_of_substance[si::mole],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<ncomp>,
  },
  {
    .name        = "V",
    .description = "ideal gas volume",
    .flow        = output,
    .dimension   = isq::volume[cubic(si::metre)],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  }
>;
```

Tensor Constraints

```
constexpr auto matrix_mul_params = model_parameters
<
  {
    .name        = "A",
    .description = "left operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<A_rows, inner>,
  },
  {
    .name        = "B",
    .description = "right operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<inner, B_cols>,
  },
  {
    .name        = "C",
    .description = "product A * B",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<A_rows, B_cols>,
  }
>;
```

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

Tensor Constraints

```
constexpr auto matrix_mul_params = model_parameters
<
  {
    .name        = "A",
    .description = "left operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<A_rows, inner>,
  },
  {
    .name        = "B",
    .description = "right operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<inner, B_cols>,
  },
  {
    .name        = "C",
    .description = "product A * B",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<A_rows, B_cols>,
  }
>;
```

$$C = A \times B$$
$$[i,k] = [i,\boldsymbol{j}] \times [\boldsymbol{j},k]$$

… + CTAD + NTTP = 😻

Tensor Constraints

```
constexpr auto matrix_mul_params = model_parameters
<
  {
    .name        = "A",
    .description = "left operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<A_rows, inner>,
  },
  {
    .name        = "B",
    .description = "right operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<inner, B_cols>,
  },
  {
    .name        = "C",
    .description = "product A * B",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<A_rows, B_cols>,
  }
>;
```

$$C = A \times B$$
$$[\boldsymbol{i}, \boldsymbol{k}] = [\boldsymbol{i}, j] \times [j, \boldsymbol{k}]$$

… + CTAD + NTTP = 😻

Tensor Constraints

```
constexpr auto matrix_mul_params = model_parameters
<
  {
    .name        = "A",
    .description = "left operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<A_rows, inner>,
  },
  {
    .name        = "B",
    .description = "right operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<inner, B_cols>,
  },
  {
    .name        = "C",
    .description = "product A * B",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<A_rows, B_cols>,
  }
>;
```

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

Tensor Constraints

```
constexpr auto matrix_mul_params = model_parameters
<
  {
    .name        = "A",
    .description = "left operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<2, inner>,
  },
  {
    .name        = "B",
    .description = "right operand",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<inner, 5>,
  },
  {
    .name        = "C",
    .description = "product A * B",
    .flow        = input,
    .dimension   = any,
    .domain      = domain_infinite,
    .extents     = extent_constraints<2, 5>,
  }
>;
```

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

… + CTAD + NTTP = 😻

Descriptors

```
template <size_t N1, size_t N2, typename Flow,
          typename Dimension, typename Domain,
          typename Extents>
struct param_descriptor
{
  fixed_string<N1> name;
  fixed_string<N2> description;
  Flow             flow;
  Dimension        dimension;
  Domain           domain;
  Extents          extents;
};
```

Descriptors

```
template <typename Flow, typename Dimension,
          typename Domain, typename Extents>
struct param_descriptor
{
  Flow              flow;
  Dimension         dimension;
  Domain            domain;
  Extents           extents;
};
```

```
inline constexpr struct input_t { } input{};
inline constexpr struct output_t { } output{};
```

```
{
  .flow        = input, // <- input_t
  .dimension   = isq::pressure[si::pascal],
  .domain      = domain_non_zero,
  .extents     = extent_constraints<2, inner>,
}
```

Descriptors

```
template <typename Flow, typename Dimension,
          typename Domain, typename Extents>
struct param_descriptor
{
  Flow              flow;
  Dimension         dimension;
  Domain            domain;
  Extents           extents;
};
```

```
{
  .flow        = input,
  .dimension   = isq::pressure[si::pascal],
  .domain      = domain_non_zero,
  .extents     = extent_constraints<2, inner>,
}
```

```
constexpr auto domain_non_zero = domain<
  {.lower = inclusive(-inf), .upper = exclusive(0)},
  {.lower = exclusive(0), .upper = inclusive(inf)}>;

template <interval... Is>
inline constexpr auto domain = domain_t<Is...>{};

template <interval... Is>
struct domain_t : tuple<Is...> { };

template <typename Lower, typename Upper>
struct interval { Lower lower; Upper upper; };

struct inclusive { double limit; };
struct exclusive { double limit; };
```

Descriptors

```
template <typename Flow, typename Dimension,
          typename Domain, typename Extents>
struct param_descriptor
{
  Flow              flow;
  Dimension         dimension;
  Domain            domain;
  Extents           extents;
};
```

```
{
  .flow        = input,
  .dimension   = isq::pressure[si::pascal],
  .domain      = domain_non_zero,
  .extents     = extent_constraints<2, inner>,
}
```

```
constexpr auto domain_non_zero = domain_t<
  {.lower = inclusive(-inf), .upper = exclusive(0)},
  {.lower = exclusive(0), .upper = inclusive(inf)}>{};

template <interval... Is>
inline constexpr auto domain = domain_t<Is...>{};

template <interval... Is>
struct domain_t : tuple<Is...> { };

template <typename Lower, typename Upper>
struct interval { Lower lower; Upper upper; };

struct inclusive { double limit; };
struct exclusive { double limit; };
```

Descriptors

```
template <typename Flow, typename Dimension,
          typename Domain, typename Extents>
struct param_descriptor
{
  Flow              flow;
  Dimension         dimension;
  Domain            domain;
  Extents           extents;
};
```

```
{
  .flow        = input,
  .dimension   = isq::pressure[si::pascal],
  .domain      = domain_non_zero,
  .extents     = extent_constraints<2, inner>,
}
```

```
constexpr auto domain_non_zero = domain<
  {.lower = inclusive(-inf), .upper = exclusive(0)},
  {.lower = exclusive(0), .upper = inclusive(inf)}>;

template <interval... Is>
inline constexpr auto domain = domain_t<Is...>{};

template <interval... Is>
struct domain_t : tuple<Is...> { };

template <typename Lower, typename Upper>
struct interval { Lower lower; Upper upper; };

struct inclusive { double limit; };
struct exclusive { double limit; };
```

Descriptors

```
template <typename Flow, typename Dimension,
          typename Domain, typename Extents>
struct param_descriptor
{
  Flow              flow;
  Dimension         dimension;
  Domain            domain;
  Extents           extents;
};
```

```
{
  .flow         = input,
  .dimension    = isq::pressure[si::pascal],
  .domain       = domain_non_zero,
  .extents      = extent_constraints<2, inner>,
}
```

```
template <auto... Extents>
struct extent_constraints_t : tuple<Extents...> { };

template <auto... Extents>
inline constexpr auto extent_constraints =
                        extent_constraints_t<Extents...>{};

inline constexpr struct inner_t { } inner{};
```

Descriptors

```
template <typename Flow, typename Dimension,
          typename Domain, typename Extents>
struct param_descriptor
{
  Flow              flow;
  Dimension         dimension;
  Domain            domain;
  Extents           extents;
};
```

```
template <param_descriptor... Params>
inline constexpr auto model_parameters = tuple<Params...>{};
```

```
{
  .flow        = input,
  .dimension   = isq::pressure[si::pascal],
  .domain      = domain_non_zero,
  .extents     = extent_constraints<2, inner>,
}
```

Descriptors

```
template <typename Flow, typename Dimension,
          typename Domain, typename Extents>
struct param_descriptor
{
  Flow              flow;
  Dimension         dimension;
  Domain            domain;
  Extents           extents;
};
constexpr auto params = model_parameters
<
  {
    .flow         = input,
    .dimension    = isq::pressure[si::pascal],
    .domain       = domain_non_zero,
    .extents      = extent_constraints<2, inner>,
  }
>;
```

```
template <param_descriptor... Params>
inline constexpr auto model_parameters = tuple<Params...>{};
```

```
constexpr auto ideal_gas_volume_params = model_parameters
<
  {
    .name        = "T",
    .description = "temperature",
    .flow        = input,
    .dimension   = isq::thermodynamic_temperature[si::kelvin],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "P",
    .description = "pressure",
    .flow        = input,
    .dimension   = isq::pressure[si::pascal],
    .domain      = domain_positive,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "n",
    .description = "molar composition",
    .flow        = input,
    .dimension   = isq::amount_of_substance[si::mole],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<ncomp>,
  },
  {
    .name        = "V",
    .description = "ideal gas volume",
    .flow        = output,
    .dimension   = isq::volume[cubic(si::metre)],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  }
>;
```

$$C = A \times B$$
$$[i, k] = [i, j] \times [j, k]$$

Extent Constraints

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

$$C = A \times B$$
$$[i, k] = [i, j] \times [j, k]$$

Extent Constraints

```
A_rows  inner
        B_cols
```

$$C = A \times B$$
$$[i, k] = [i, j] \times [j, k]$$

Extent Constraints

```
A_rows
inner
B_cols
```

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };
```

```
   constraint_size<A_rows>
   constraint_size<inner >
   constraint_size<B_cols>
```

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	-1
inner	-1
B_cols	-1

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

A: [3,5]
B: [5,2]
C: [3,2]

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	-1
inner	-1
B_cols	-1

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

$A: [\mathbf{3}, 5]$
$B: [5,2]$
$C: [3,2]$

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	**3**
inner	-1
B_cols	-1

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

$A: [\mathbf{3}, 5]$
$B: [5,2]$
$C: [3,2]$

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	**-1**
B_cols	-1

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

A: $[3, \mathbf{5}]$
B: $[5,2]$
C: $[3,2]$

… + CTAD + NTTP = 😻

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	**5**
B_cols	-1

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

$A: [3, \mathbf{5}]$
$B: [5,2]$
$C: [3,2]$

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	**5**
B_cols	-1

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

A: [3,5]
B: [**5**, 2]
C: [3,2]

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	5
B_cols	**-1**

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

$A: [3,5]$
$B: [5, \mathbf{2}]$
$C: [3,2]$

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	5
B_cols	**2**

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

A: [3,5]
B: [5, **2**]
C: [3,2]

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	5
B_cols	2

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

$A: [3,5]$
$B: [5,2]$
$C: [\mathbf{3}, 2]$

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	5
B_cols	**2**

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

A: [3,5]
B: [5,2]
C: [3, **2**]

… + CTAD + NTTP = 😻

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	5
B_cols	2

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

A: [3,5]
B: [5,2] ✅
C: [3,2]

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	**5**
B_cols	2

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

$A: [3, \mathbf{5}]$
$B: [\mathbf{6}, 2]$
$C: [3,2]$

… + CTAD + NTTP = 😻

Extent Constraints

$$C = A \times B$$
$$[i,k] = [i,j] \times [j,k]$$

```
template <auto>
class constraint_size { size_t _size = -1; };

auto constraints = std::make_tuple(
    constraint_size<A_rows>{},
    constraint_size<inner >{},
    constraint_size<B_cols>{});
```

A_rows	3
inner	**5**
B_cols	2

```
extent_constraints<A_rows, inner>
extent_constraints<inner,  B_cols>
extent_constraints<A_rows, B_cols>
```

$A: [3, \mathbf{5}]$
$B: [\mathbf{6}, 2]$ ❌
$C: [3,2]$

… + CTAD + NTTP = 😻

```
constexpr auto ideal_gas_volume_params = model_parameters
<
  {
    .name        = "T",
    .description = "temperature",
    .flow        = input,
    .dimension   = isq::thermodynamic_temperature[si::kelvin],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "P",
    .description = "pressure",
    .flow        = input,
    .dimension   = isq::pressure[si::pascal],
    .domain      = domain_positive,
    .extents     = extent_constraints<>,
  },
  {
    .name        = "n",
    .description = "molar composition",
    .flow        = input,
    .dimension   = isq::amount_of_substance[si::mole],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<ncomp>,
  },
  {
    .name        = "V",
    .description = "ideal gas volume",
    .flow        = output,
    .dimension   = isq::volume[cubic(si::metre)],
    .domain      = domain_non_negative,
    .extents     = extent_constraints<>,
  }
>;
```

… + CTAD + NTTP = 😍

Non-Conforming C++
The Secrets the Committee Is
Hiding From You

godbolt.org/z/qnh6qqE3T

@mknejp

Teaching Containers And Allocators
How To Sanitize Addresses