The (dark) Art of Type Functions

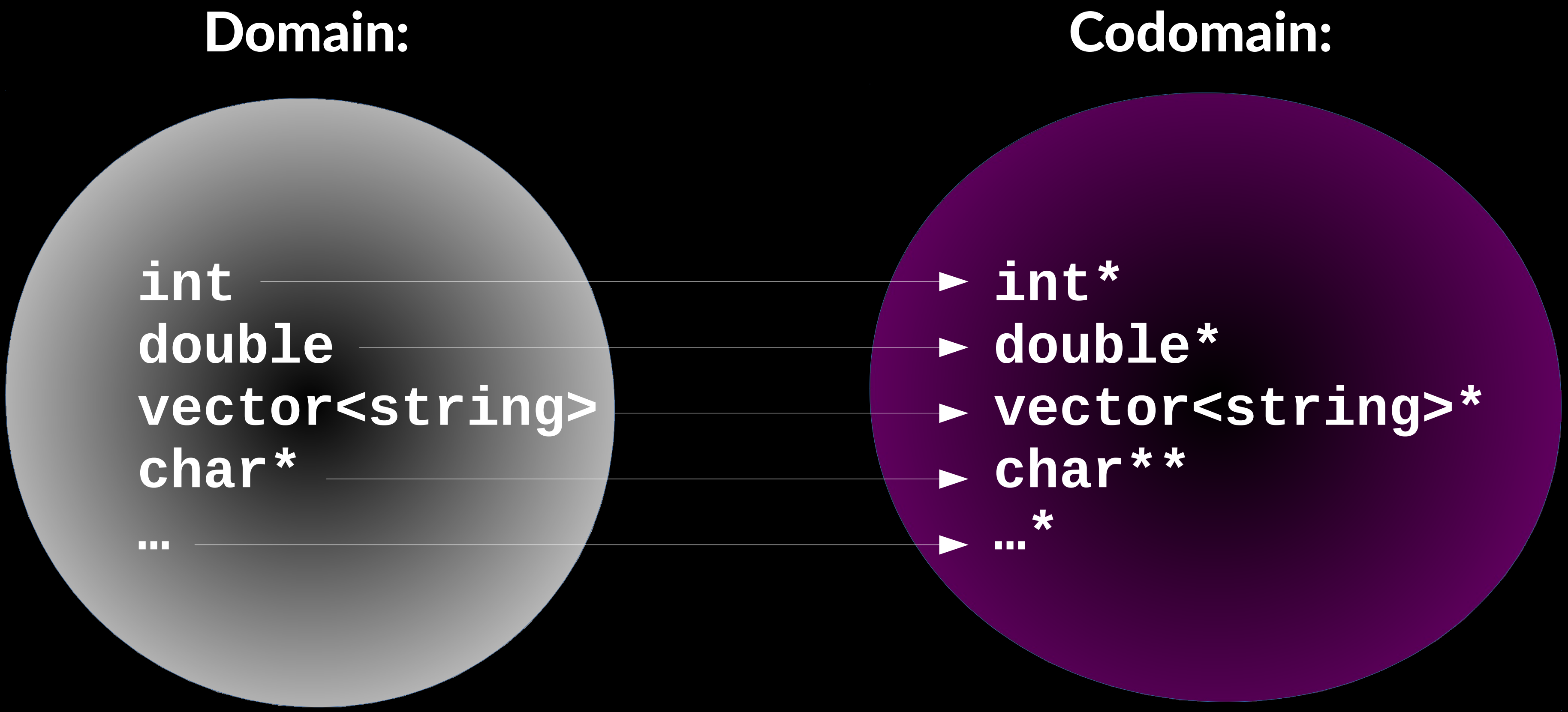
Petter Holmberg – C++ Stockholm 0x11 – November 2018

```
int* x;
```

```
#define pointer_type(T) T*

pointer_type(int) x;
```

`pointer_type: type → type`



Domain:

Codomain:

int ► int*

double ► double*

vector<string> ► vector<string>*

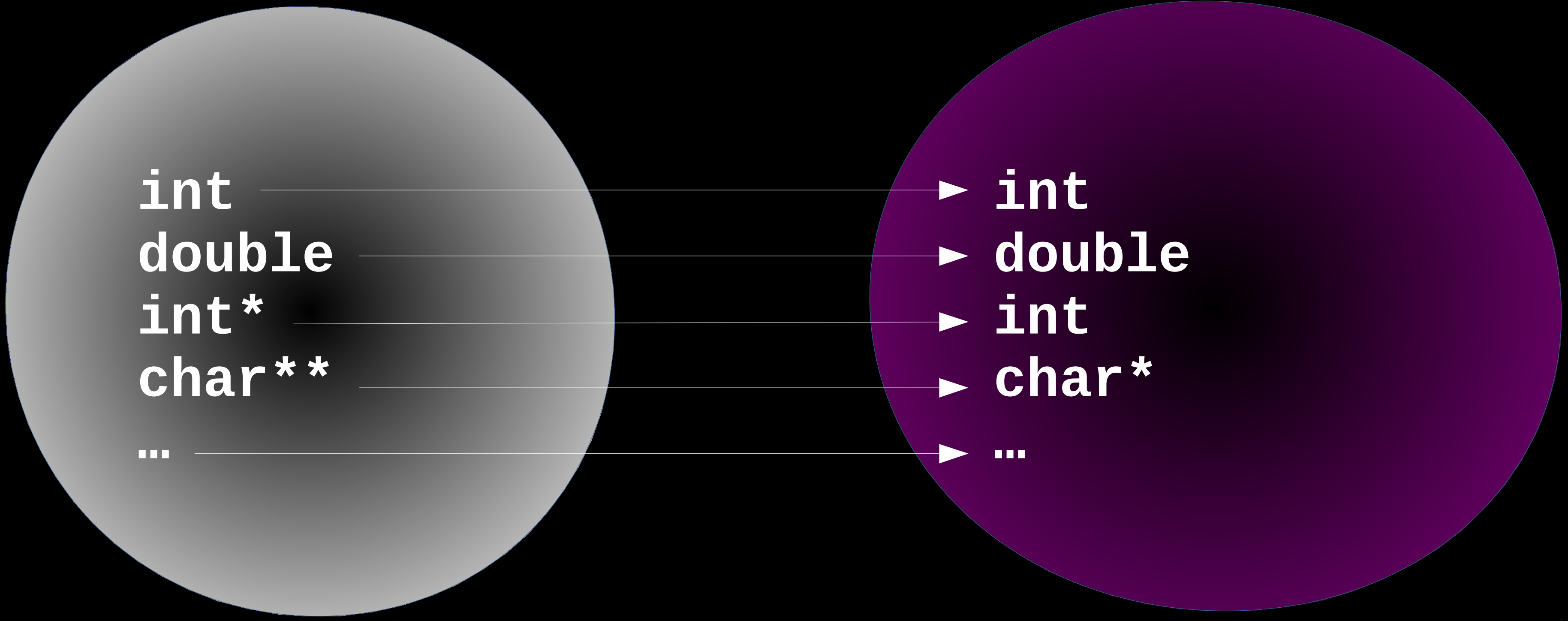
char* ► char**

… ► …*

```
template <typename T>
using pointer_type<T> = T*;

pointer_type<int> x;
```

`value_type: type → type`



Domain:	Codomain:
`int`	`int`
`double`	`double`
`int*`	`int`
`char**`	`char*`
…	…

```
template <typename T>
struct value_type_traits
{
    using type = T;
};
```

```
int x;
```

```
typename value_type_traits<int>::type x;
```

```
// Alias template for simpler syntax
template <typename T>
using value_type =
    typename value_type_traits<T>::type;
```

```
value_type<int> x;
```

```
// Default version
template <typename T>
struct value_type_traits
{
    using type = T;
};

// Specialization for pointers
template <typename T>
struct value_type_traits<pointer_type<T>>
{
    using type = T;
};
```

```
template <typename InputIt, typename T>
constexpr auto
find(InputIt first, InputIt last, T const& value)
-> InputIt
{
    while (first != last)
    {
        if (*first == value) break;
        ++first;
    }
    return first;
}
```

```
template <typename InputIt>
constexpr auto
find(InputIt first, InputIt last,
     value_type<InputIt> const& value)
-> InputIt
{
    while (first != last)
    {
        if (*first == value) break;
        ++first;
    }
    return first;
}
```

```
// Default version
template <typename T>
constexpr auto
load(T const& value) -> T const&
{
    return value;
}

// Specialization for pointers
template <typename T>
constexpr auto
load(pointer_type<T> value) -> T const&
{
    return *value;
}
```

```
template <typename InputIt>
constexpr auto
find(InputIt first, InputIt last,
     value_type<InputIt> const& value)
-> InputIt
{
    while (first != last)
    {
        if (*first == value) break;
        ++first;
    }
    return first;
}
```

```
template <typename InputIt>
constexpr auto
find(InputIt first, InputIt last,
     value_type<InputIt> const& value)
-> InputIt
{
    while (first != last)
    {
        if (load(first) == value) break;
        ++first;
    }
    return first;
}
```

```
static_assert(find(1, 11, 5) == 5);

static_assert(find(1, 11, 15) == 11);
```

```
// Find first number matching my_predicate
find_if(1, 11, my_predicate);

// Fill my_array with the numbers 1 to 10
copy(1, 11, my_array);

// Print the numbers 1 to 10
for_each(1, 11, [](int x){ cout << x << ‘\n’; });
```

Type functions are ***just normal functions***!

Built into the core language syntax, provided by the standard library (see `<type_traits>` header for more examples)

Hack your own using traits classes and alias templates

JTC1/SC22/WG21 paper: *“Type functions and beyond”* [P0844R0] - *J. Monnon*

With C++20 concepts and constraints you’ll need them all the time