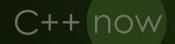
# So You Think You Know How to Work With Concepts?

Andrzej Krzemieński





## Definitions

Two meanings of "concept" in C++:

- Interfaces in Generic Programming
- A C++20 language feature

## Generic Libraries

#### **Boost.Optional library**

Uses templates, but no concepts.

```
boost::optional<std::mutex> om; // only requires Destructible
auto om2 = om; // won't compile (conditional interface)
```

## Generic Libraries

Markable library https://github.com/akrzemi1/markable

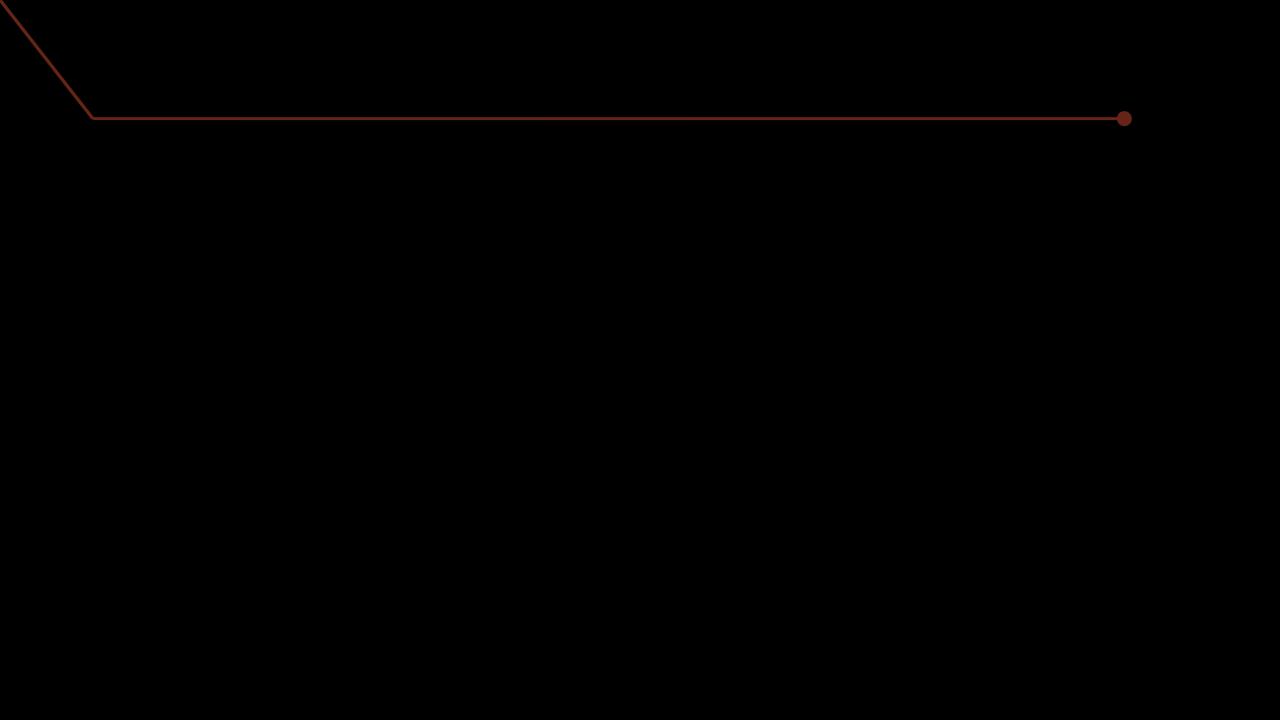
Uses concept for the mark policy

```
markable<mark_int<int, -1>> oi;
```

```
template <typename MP>
concept mark_policy = requires {
   // set marked value
   // check for marked value
};
```

### Generic Libraries

```
template <typename MP>
concept mark_policy =
 requires
    typename MP::value type;
   typename MP::storage_type;
   typename MP::reference type;
   typename MP::representation type;
  } &&
 requires(const typename MP::representation_type & r,
           const typename MP::storage_type &
                                                    s,
           const typename MP::value_type &
                                                    cv,
                 typename MP::value_type &&
                                                    rv)
    { MP::marked value() }
                                       -> std::convertible to<typename MP::representation type>;
    { MP::is marked value(r) }
                                       -> std::convertible to<bool>;
    { MP::access_value(s) }
                                       -> std::same_as<typename MP::reference_type>;
    { MP::representation(s) }
                                       -> std::same as<const typename MP::representation type &>;
    { MP::store_value(cv) }
                                       -> std::convertible_to<typename MP::storage_type>;
    { MP::store value(std::move(rv)) } -> std::convertible to<typename MP::storage_type>;
  };
```



# Concepts checks

#### Guarantees for the author

```
template <LibConcept T>
void libFun(T val) {
    /* ... */
}
```

No guarantee that the function uses only the concept interface

## Concepts checks

Guarantees for the user

```
static_assert(LibConcept<UserType>);
```

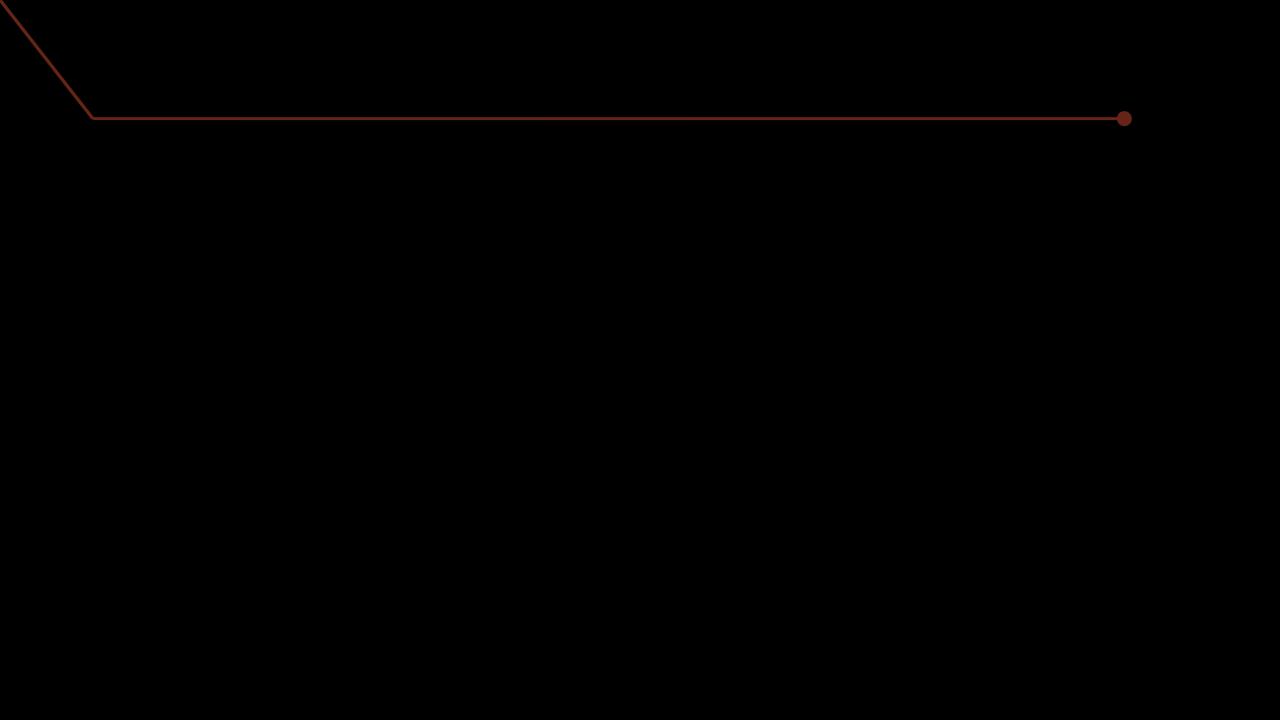
- On failure, a guarantee that the type will not work.
- On pass, no guarantee that the type will work.

## Concepts checks

#### The negative guarantee

- User type doesn't work work by accident
- Library gets implementation flexibility

```
template <typename T>
bool differ(T const& a, T const& b) {
  return !(a == b);
  // return a != b;
};
```



## Multi-type concepts

```
template <class Iter, class Sentinel, class Pred>
concept PredicatedIteration = requires(Iter i, Sentinel s, Pred p) {
    { i != s } -> convertible_to<bool>;
    { p(*i) } -> convrtible_to<bool>;
    ++i;
};
```

One concept – three constrained types

Requires three archetypes to test the concept

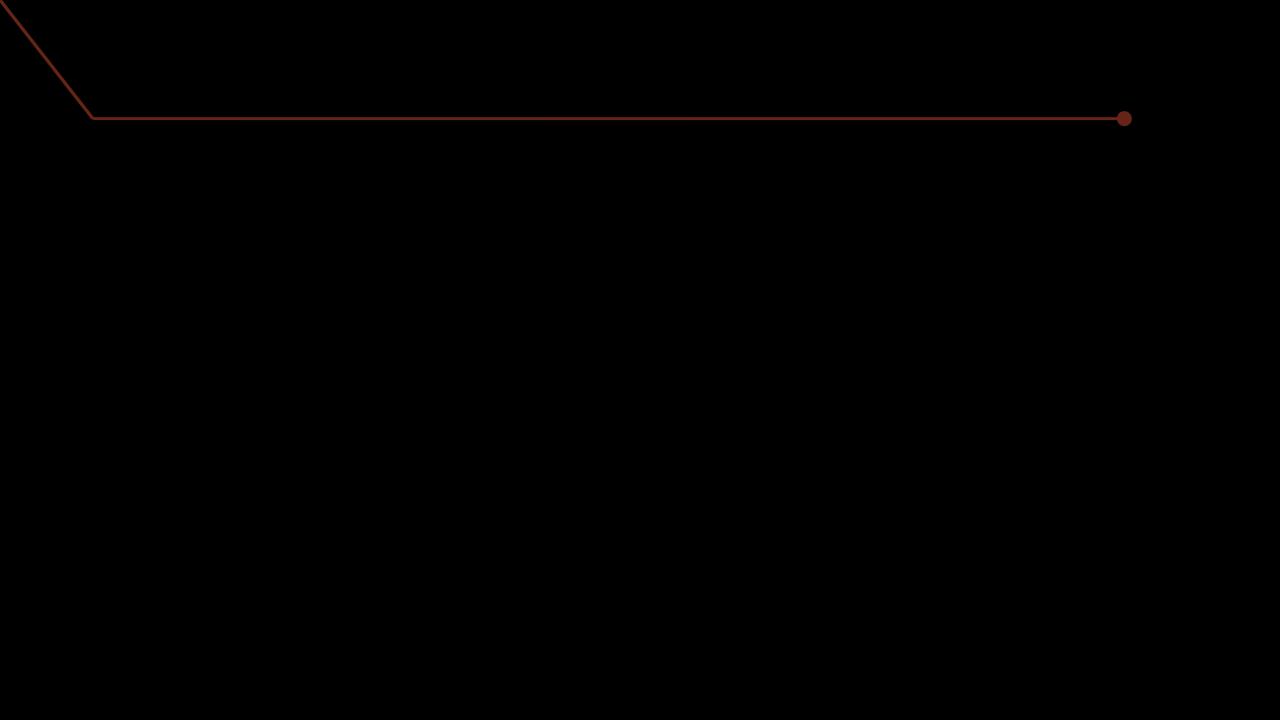
## Archetypes

```
template <class X, class Y> requires C<X, Y> void fun(X, Y);
template <class X, class Y> requires C<X, Y> && D<X, Y> void fun(X, Y);

// requires up to 4 archetypes:
// X_for_C, Y_for_C
// X_for_C_and_D, Y_for_C_and_D
```

One archetype set per one set of constraints

```
template <std::regular T>
void properties(T a) {
  assert(T{a} == a);
template <Addable T>
void properties(T a, T b) {
  T x = a;
  assert((a + b) == (x += b));
```



```
struct BigInt : AddableIface {
   // ...
};
```

- Author commits to complying with semantic requirements of the interface.
- This cannot happen by accident.

AddableConcept<std::string>

Accidental compliance with syntactic concept requirements.

Violating *syntactic* requiements:

Compiler error

Violating *semantic* requiements:

- Bug
- UB for STD concepts

# Enforced Semantic Requirements in STD

std::ranges::view

std::ranges::enable\_view

Copying and destruction is O(1)

std::ranges::sized\_range

std::ranges::disable\_sized\_range

ranges::size() is amortized O(1)

```
template <Range R, Pred P>
void algo_1(R r, P p);
```

Probably a bug

```
template <Range R, Pred P>
  requires SomeProperty<R, P>
void algo_2(R r, P p);
```

SomeProperty has additional semantic requirements

## Contact

akrzemi1@gmail.com akrzemi1.wordpress.com