Linear types

Secret Meeting C++ 2018, Berlin

Dr Ivan Čukić

ivan@cukic.co
http://cukic.co

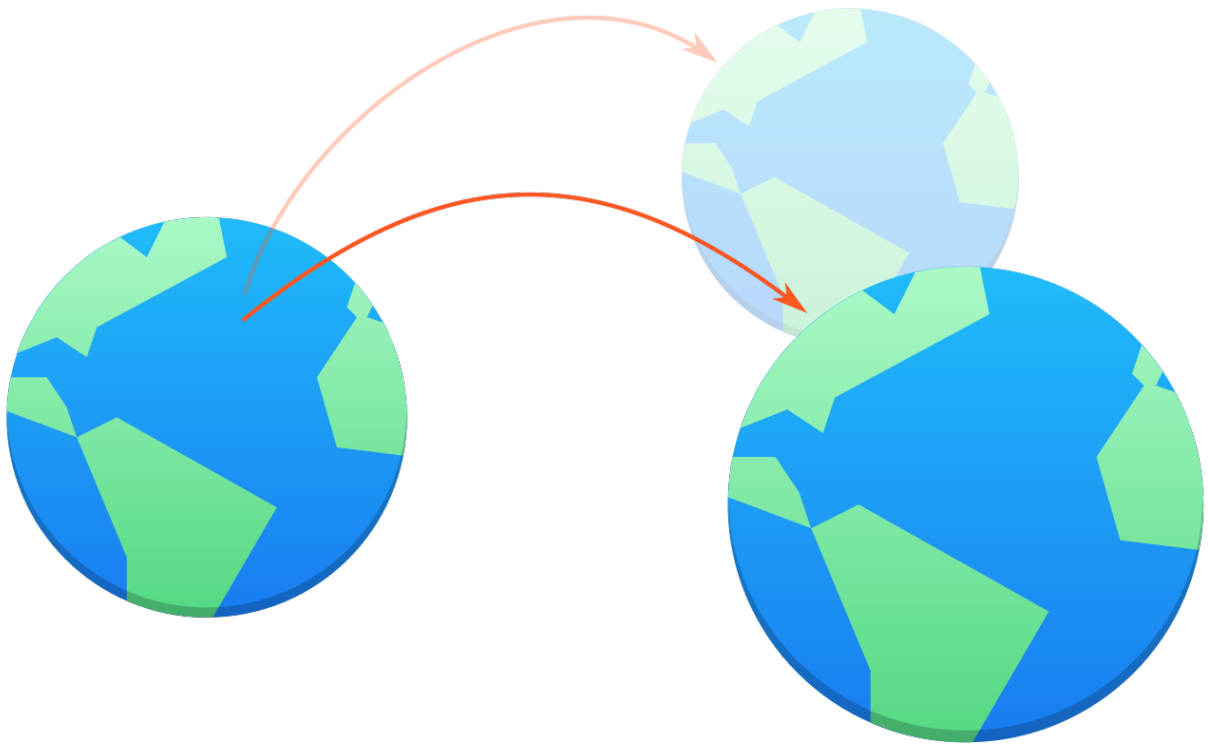
kde.org

Disclaimer

> Make your code readable. Pretend the next person who looks at your code is a psychopath and they know where you live.
>
> Philip Wadler

FAR AWAY WORLDS

Far away worlds



Far away worlds

Values belonging to a linear type must be **used exactly once**: like the world, they can not be duplicated or destroyed. Such values require no reference counting or garbage collection...

Linear types can change the world!, Philip Wadler

CLONES

Attack of the clones

```
istream_sequence<std::string> in{std::cin};

std::string result;
for (const auto& token: in) {
    result.append(token);
}
```

Attack of the clones

```
istream_sequence<std::string> in{std::cin};

std::string result;
for (const auto& token: in) {
    result.append(token);
}
```

Remember what Sean said?

Attack of the clones

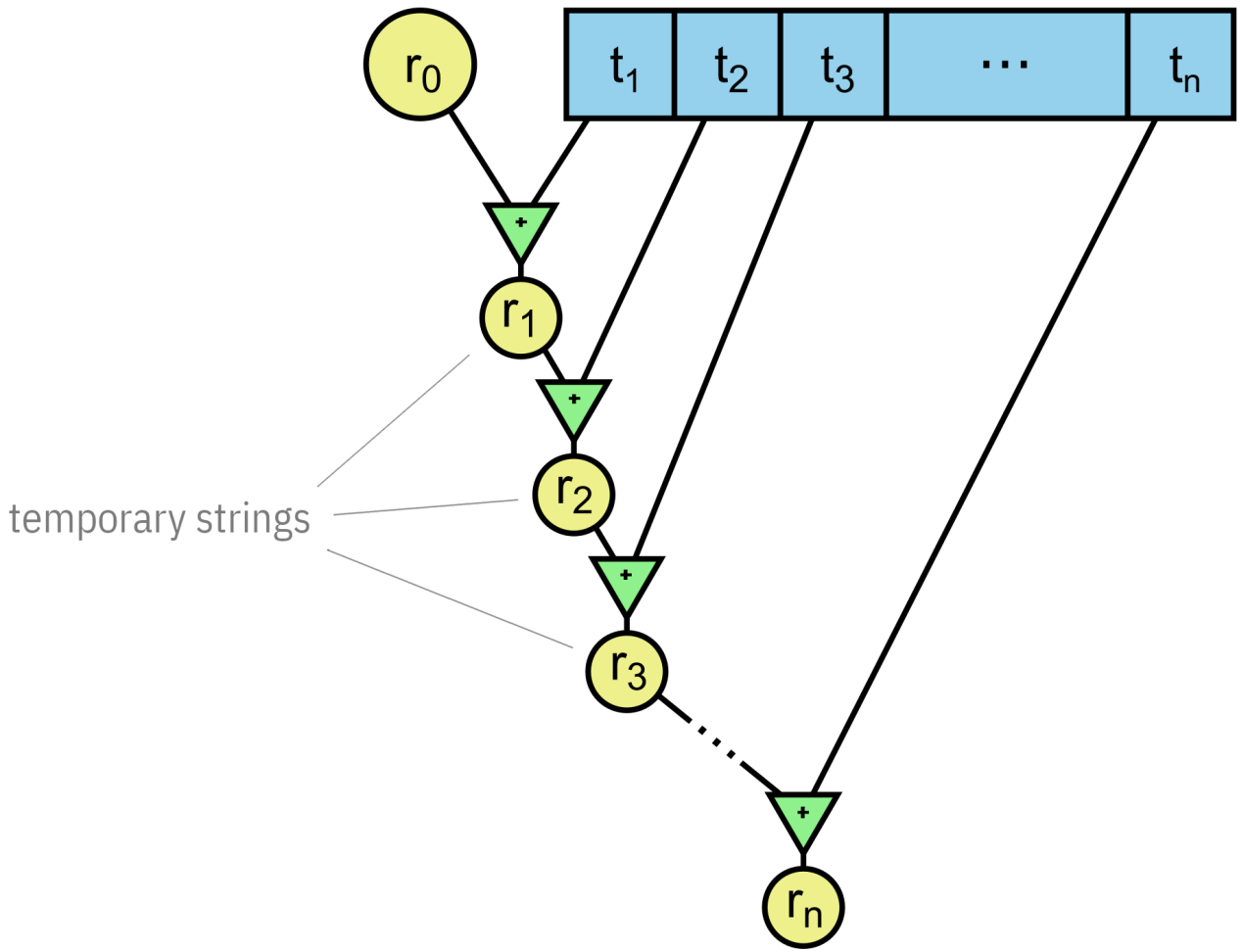
```
istream_sequence<std::string> in{std::cin};

const auto result =
          accumulate(in, string{});
```

Remember what Sean said?

Attack of the clones

```
template <typename InputIt, typename T>
T accumulate(InputIt first, InputIt last, T init)
{
    while (first != last) {
        init = init + *first;
        ++first;
    }
    return init;
}
```

Attack of the clones



Attack of the clones

```
template <typename InputIt, typename T>
T accumulate(InputIt first, InputIt last, T init)
{
    while (first != last) {
        init = std::move(init) + *first;
        ++first;
    }
    return init;
}
```

Attack of the clones

Copying is the silent (performance) killer

LINEAR IN C++

Linear in C++

```
namespace detail {

template <typename T, typename U>
constexpr bool linear_usable_as_v =

    std::is_nothrow_constructible_v<T, U> and
    std::is_nothrow_assignable_v<T&, U> and
    std::is_nothrow_convertible_v<U, T>;

}
```

Linear in C++

```
namespace detail {

template <typename T, typename U>
constexpr bool linear_unusable_as_v =

    not std::is_constructible_v<T, U> and
    not std::is_assignable_v<T&, U> and
    not std::is_convertible_v<U, T>;

}
```

Linear in C++

```
template <typename T>
concept Linear =
    std::is_nothrow_destructible_v<T> and

    detail::linear_usable_as<T, T> and
    detail::linear_usable_as<T, T&&> and

    detail::linear_unusable_as<T, T&> and
    detail::linear_unusable_as<T, const T&> and
    detail::linear_unusable_as<T, const T>;
```

Linear in C++

```
Linear ptr = std::make_unique<person>(); // OK

Linear str = ”Hello world”s; // ERROR
```

Wrapper

```
class linear_wrapper {
public:
    linear_wrapper(T&& value)
        : m_value{std::move(value)}
    {
    }

private:
    T m_value;
};
```

Linear wrapper

```
class linear_wrapper {
public:
    linear(const linear&) = delete;
    linear(linear&&) = default; // noexcept

    linear& operator=(const linear&) = delete;
    linear& operator=(linear&&) = default; // noexcept

private:
    T m_value;
};
```

Linear wrapper

```
class linear_wrapper {
public:
    [[nodiscard]] T&& get() && noexcept
    {
        return std::move(value);
    }

private:
    T m_value;
};
```

Linear wrapper

```
class linear_wrapper {
public:
    [[nodiscard]] T&& operator*() && noexcept
    {
        return std::move(value);
    }

private:
    T m_value;
};
```

Linear wrapper

```
linear<std::string> operator"" _ls(const char* data,
                                   std::size_t len)
{
    return linear<std::string>{data};
}

accumulate(in, "Concatenated:"_ls); // ERROR before C++20
```

Pipelines

```
auto pipeline =
    voy::system_cmd("ping"s, "localhost"s)
    | voy::transform([] (lstring value) {
        std::transform(value.begin(), value.end(), value.begin(), toupper);
        return value;
      })
    | append_pid
    | voy_bridge(frontend_to_backend_1)

    | voy::transform([] (lstring value) {
        const auto pos = value.find_last_of('=');
        return std::make_pair(std::move(value), pos);
      })
    | voy::transform([] (std::pair<lstring, size_t>&& pair) {
        auto [ value, pos ] = pair;
        return pos == std::string::npos
                    ? std::move(value)
                    : std::string(value.cbegin() + pos + 1, value.cend());
      })
    | append_pid
    | voy_bridge(backend_1_to_backend_2)

    | voy::filter([] (lstring value) {
        return value < "0.145"s;
      })
    | append_pid
    | voy_bridge(backend_2_to_frontend)

    | voy::sink{cout};
```

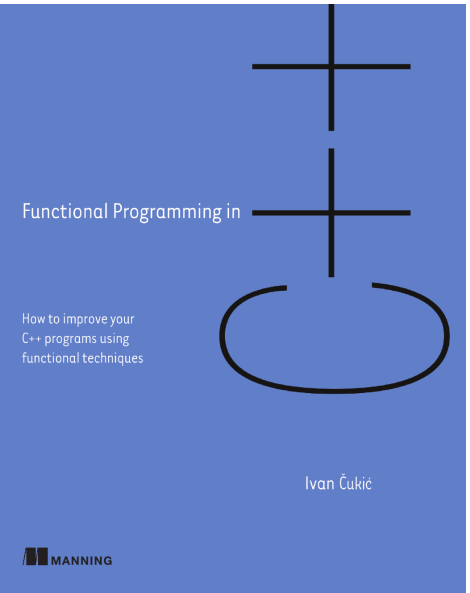
Additional

- Use after move (`clang-tidy:bugprone-use-after-move`)
- Unused variable error (`-Werror=unused-variable`)
- Error handling (`optional<T>, expected<T,E>`)

Questions

Kudos (in chronological order):

Friends at **KDE**
Saša Malkov and **Zoltan Porkolab**



MANNING PUBLICATIONS
Functional Programming in C++
cukic.co/to/fp-in-cpp

Discount code **ctwmeetingcpp18**
(40% off all books and videos)