COMPILE-TIME AND RUN-TIME DEPENDENCY INJECTION

C++ on Sea

Dr. Marius Feilhauer

13th July 2022

MARIUS FEILHAUER

M.Sc. Automotive Engineering
University of Stuttgart

Working Experience
@ Bosch, Aston Martin & Porsche

Ph.D. Simulation-based validation of advanced driver assistance systems
HLRS University of Stuttgart & ETAS GmbH

Software Developer
C++-based Simulationmodels @ETAS GmbH

Software Developer iMOW
@ STIHL AG

Lectures on Software Development
University Stuttgart

AGENDA

STIHL

1. **Alarm Clock Example** ⏲️
2. **Run-Time Dependency Injection**
   Splitting the Alarm Clock Example
3. **Compile-Time Dependency Injection**
   Splitting the Alarm Clock Example
4. **Summary**

ALARM CLOCK EXAMPLE

STIHL

Basic Idea

We want to have an alarm clock class
which enables us to **set the current time**,
set an alarm and **(de-)activate the alarm**.
The alarm shall be outputted on the console.

ALARM CLOCK EXAMPLE

STIHL

Get the code

You can find the sample code for this talk on
Compiler Explorer

https://godbolt.org/z/q5cTEMKPa

ALARM CLOCK EXAMPLE

STIHL

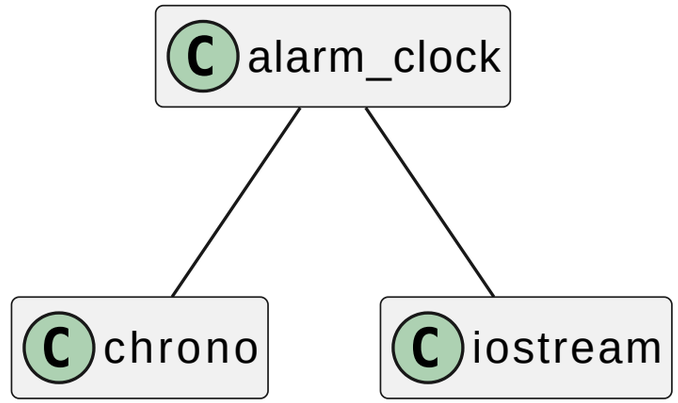
API Overview

```cpp
#include <chrono>
class alarm_clock {
public:
    using time_point = std::chrono::time_point<std::chrono::system_clock>;

    void set_alarm(time_point);
    void activate();
    void deactivate();

private:
    void check_alarm();
    // ...
};
```

ALARM CLOCK EXAMPLE

Naïve Implementation

```
1  void alarm_clock::check_alarm() {
2    if (!is_active_) return;
3
4    if (now_is_approx_alarm_time()) {
5      ring_alarm();
6    }
7  }
8
9  bool alarm_clock::now_is_approx_alarm_time() const {
10   auto diff = abs(now() - alarm_time_);
11   return (diff < 1s);
12 }
13
14 void alarm_clock::ring_alarm() {
15   cout << "Alarm\n";
16 }
```

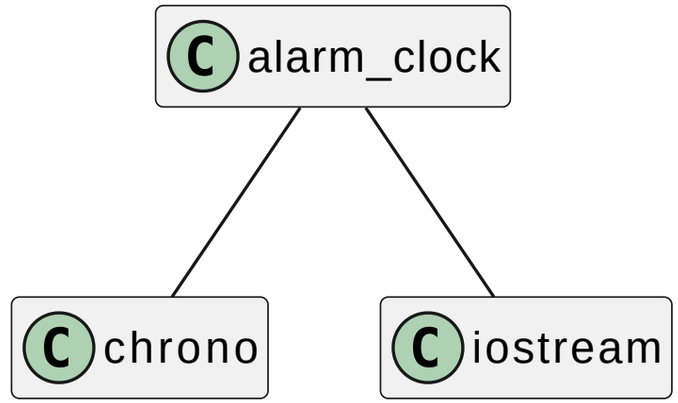


ALARM CLOCK EXAMPLE

STIHL

Naïve Implementation

```cpp
void alarm_clock::check_alarm() {
  if (!is_active_) return;

  if (now_is_approx_alarm_time()) {
    ring_alarm();
  }
}

bool alarm_clock::now_is_approx_alarm_time() const {
  auto diff = abs(now() - alarm_time_);
  return (diff < 1s);
}

void alarm_clock::ring_alarm() {
  cout << "Alarm\n";
}
```

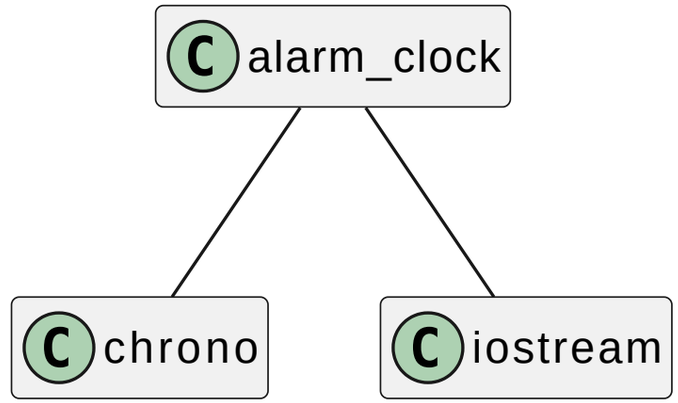


ALARM CLOCK EXAMPLE

STIHL

Naïve Implementation

```cpp
void alarm_clock::check_alarm() {
  if (!is_active_) return;

  if (now_is_approx_alarm_time()) {
    ring_alarm();
  }
}

bool alarm_clock::now_is_approx_alarm_time() const {
  auto diff = abs(now() - alarm_time_);
  return (diff < 1s);
}

void alarm_clock::ring_alarm() {
  cout << "Alarm\n";
}
```



ALARM CLOCK EXAMPLE

STIHL

Naïve Implementation Testing

```
1 TEST(alarm, checkAlarmAfter10Minutes) {
2   alarm_clock clock;
3
4   auto ring_time = now() + 10min;
5   clock.set_alarm(ring_time);
6
7 }
```

ALARM CLOCK EXAMPLE

STIHL

Naïve Implementation Testing

```
1 TEST(alarm, checkAlarmAfter10Minutes) {
2   alarm_clock clock;
3
4   auto ring_time = now() + 10min;
5   clock.set_alarm(ring_time);
6
7   // Now we have to wait a while ...
8 }
```

ALARM CLOCK EXAMPLE

STIHL

Naïve Implementation Testing

```
1 TEST(alarm, checkAlarmAfter10Minutes) {
2   alarm_clock clock;
3
4   auto ring_time = now() + 10min;
5   clock.set_alarm(ring_time);
6
7   // Now we have to wait a while ...
8   // And how do we check the output?
9 }
```

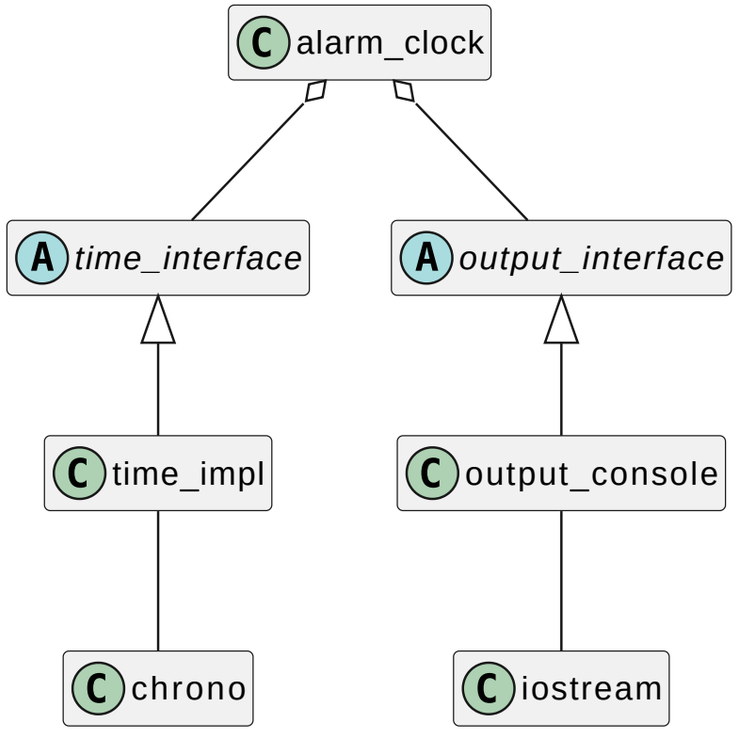
AGENDA

STIHL

1. ~~Alarm Clock Example~~ ⏲️
2. **Run-Time Dependency Injection**
   Splitting the Alarm Clock Example
3. **Compile-Time Dependency Injection**
   Splitting the Alarm Clock Example
4. **Summary**

RUN-TIME DEPENDENCY INJECTION

Class Hierarchy



RUN-TIME DEPENDENCY INJECTION

STIHL

time_interface

```cpp
#include <chrono> // Required for the using declaration

class time_interface {
public:
  using time_point = std::chrono::time_point<std::chrono::system_clock>;

  virtual ~time_interface() = default;

  virtual time_point now() const = 0;
};
```

RUN-TIME DEPENDENCY INJECTION

STIHL

time_interface

```cpp
#include <chrono> // Required for the using declaration

class time_interface {
public:
  using time_point = std::chrono::time_point<std::chrono::system_clock>;

  virtual ~time_interface() = default;

  virtual time_point now() const = 0;
};
```

RUN-TIME DEPENDENCY INJECTION

STIHL

output_interface

```
1 class output_interface {
2 public:
3
4   virtual ~output_interface() = default;
5
6   virtual output_interface& operator<<(const string&) = 0;
7 };
```

RUN-TIME DEPENDENCY INJECTION

STIHL

output_interface

```
1 class output_interface {
2 public:
3
4   virtual ~output_interface() = default;
5
6   virtual output_interface& operator<<(const string&) = 0;
7 };
```

RUN-TIME DEPENDENCY INJECTION

STIHL

Constructor

```
 1 #include "time_interface.h"
 2 #include "output_interface.h"
 3
 4 class alarm_clock {
 5 public:
 6   alarm_clock(const time_interface&, output_interface&)
 7   // ...
 8
 9 private:
10   // ...
11   const time_interface& time_;
12   output_interface& output_;
13 };
```

RUN-TIME DEPENDENCY INJECTION

STIHL

Constructor

```
 1 #include "time_interface.h"
 2 #include "output_interface.h"
 3
 4 class alarm_clock {
 5 public:
 6   alarm_clock(const time_interface&, output_interface&)
 7   // ...
 8
 9 private:
10   // ...
11   const time_interface& time_;
12   output_interface& output_;
13 };
```

RUN-TIME DEPENDENCY INJECTION

STIHL

Construction

```
1 int main() {
2   time_impl t;         // implements the time_interface
3   output_console o;    // implements the output_interface
4   alarm_clock alarm(t, o);
5
6   // ...
7
8   return 0;
9 }
```

RUN-TIME DEPENDENCY INJECTION

STIHL

Alarm Clock Implementation

```
 1 void alarm_clock::check_alarm() {
 2   if (!is_active_) return;
 3
 4   if (now_is_approx_alarm_time()) {
 5     ring_alarm();
 6   }
 7 }
 8
 9 bool alarm_clock::now_is_approx_alarm_time() const {
10   auto diff = abs(time_.now() - alarm_time_);
11   return (diff < 1s);
12 }
13
14 void alarm_clock::ring_alarm() {
15   output_ << "Alarm\n";
16 }
```

RUN-TIME DEPENDENCY INJECTION

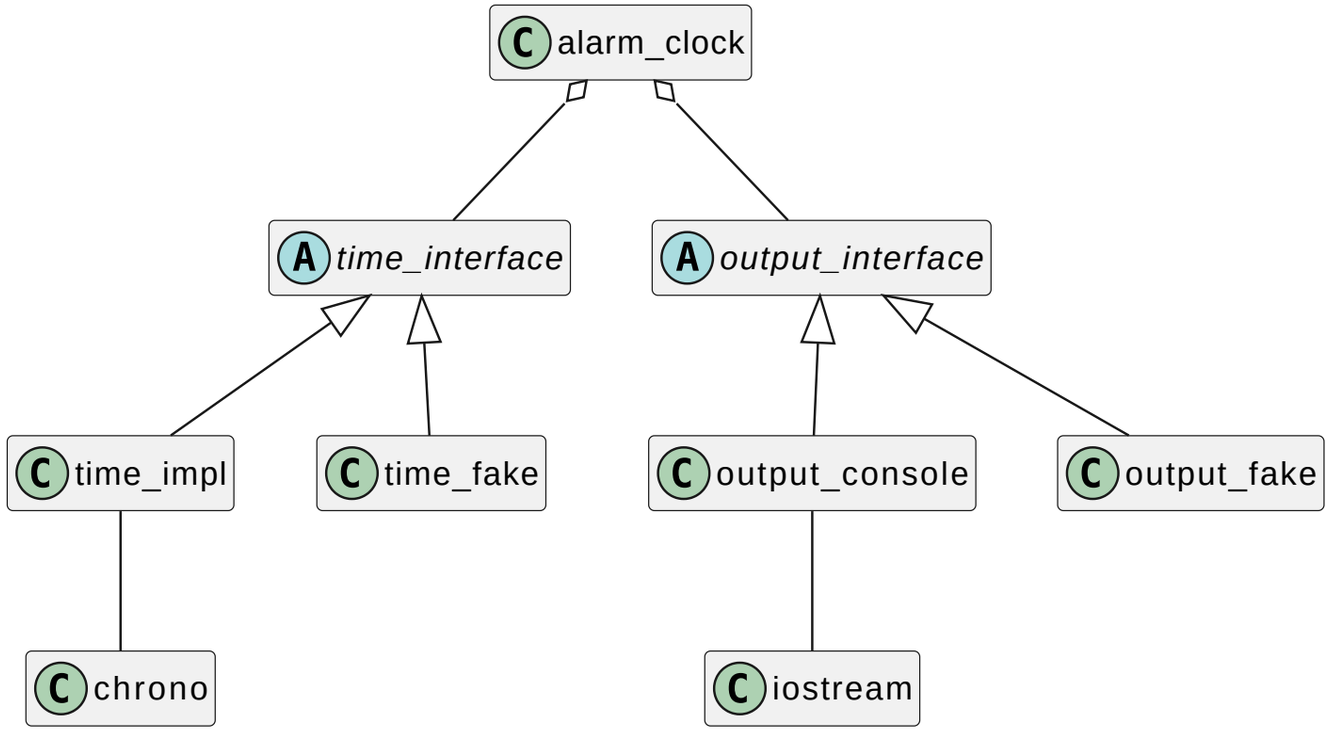
STIHL

Alarm Clock Implementation

```
 1 void alarm_clock::check_alarm() {
 2   if (!is_active_) return;
 3
 4   if (now_is_approx_alarm_time()) {
 5     ring_alarm();
 6   }
 7 }
 8
 9 bool alarm_clock::now_is_approx_alarm_time() const {
10   auto diff = abs(time_.now() - alarm_time_);
11   return (diff < 1s);
12 }
13
14 void alarm_clock::ring_alarm() {
15   output_ << "Alarm\n";
16 }
```

RUN-TIME DEPENDENCY INJECTION

Class Hierarchy



RUN-TIME DEPENDENCY INJECTION

STIHL

Testing

```
1 TEST(alarm, checkAlarmAfter10Minutes) {
2   time_fake t;   // implements the time_interface
3   output_fake o; // implements the output interface
4   alarm_clock alarm(t, o);
5
6   auto ring_time = t.now() + 10min;
7   clock.set_alarm(ring_time);
8 }
```

RUN-TIME DEPENDENCY INJECTION

STIHL

Testing

```
1  TEST(alarm, checkAlarmAfter10Minutes) {
2    time_fake t;   // implements the time_interface
3    output_fake o; // implements the output interface
4    alarm_clock alarm(t, o);
5
6    auto ring_time = t.now() + 10min;
7    clock.set_alarm(ring_time);
8
9    t = t.now() + 10min;              // We can increase our time
10 }
```

RUN-TIME DEPENDENCY INJECTION

STIHL

Testing

```
 1 TEST(alarm, checkAlarmAfter10Minutes) {
 2   time_fake t;   // implements the time_interface
 3   output_fake o; // implements the output interface
 4   alarm_clock alarm(t, o);
 5
 6   auto ring_time = t.now() + 10min;
 7   clock.set_alarm(ring_time);
 8
 9   t = t.now() + 10min;              // We can increase our time
10   EXPECT_EQ(o.value(), "Alarm\n"); // We can check the output
11 }
```

AGENDA

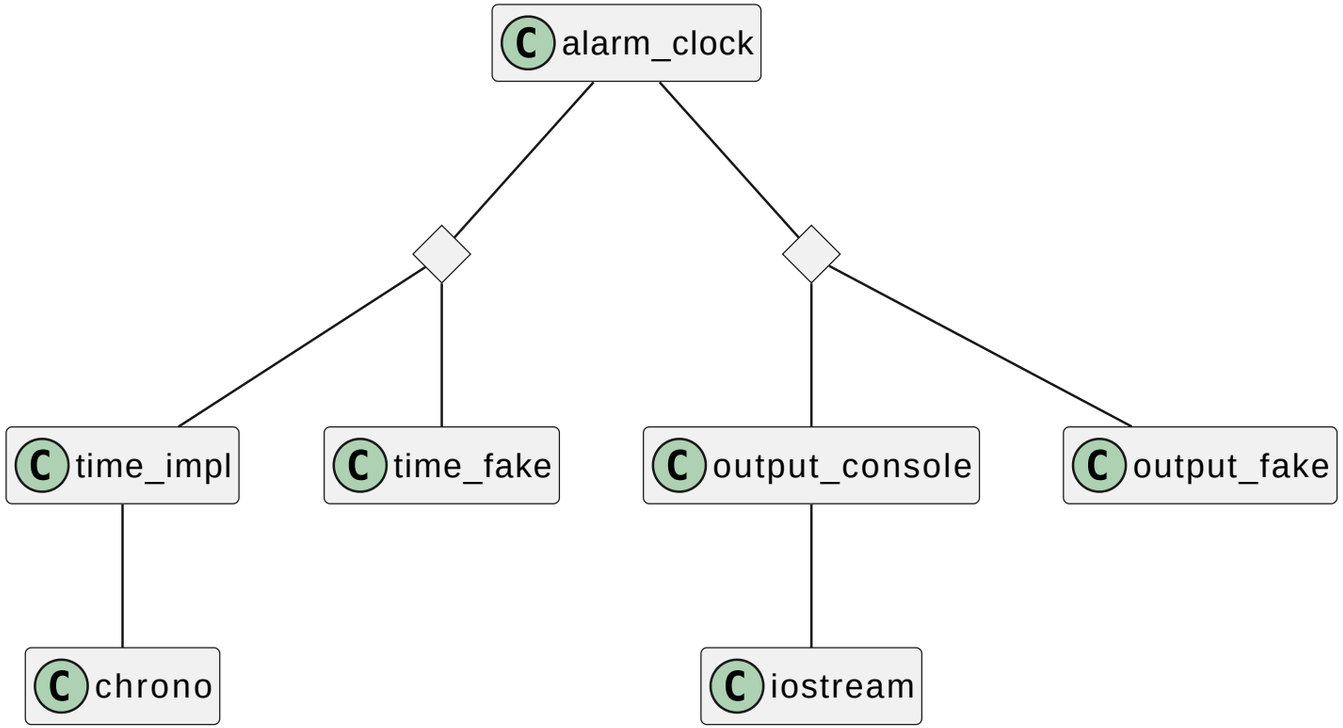
STIHL

1. ~~**Alarm Clock Example** ⏲️~~
2. ~~**Run-Time Dependency Injection**~~
   ~~Splitting the Alarm Clock Example~~
3. **Compile-Time Dependency Injection**
   Splitting the Alarm Clock Example
4. **Summary**

COMPILE-TIME DEPENDENCY INJECTION

STIHL

Class Hierarchy



COMPILE-TIME DEPENDENCY INJECTION

STIHL

Constructor

```
 1 template <class TIME, class OUTPUT>
 2 class alarm_clock {
 3 public:
 4   alarm_clock(const TIME&, OUTPUT&);
 5   // ...
 6
 7 private:
 8   // ...
 9   const TIME& time_;
10   OUTPUT& output_;
11 };
```

COMPILE-TIME DEPENDENCY INJECTION

STIHL

Constructor

```
 1 template <class TIME, class OUTPUT>
 2 class alarm_clock {
 3 public:
 4   alarm_clock(const TIME&, OUTPUT&);
 5   // ...
 6
 7 private:
 8   // ...
 9   const TIME& time_;
10   OUTPUT& output_;
11 };
```

COMPILE-TIME DEPENDENCY INJECTION

STIHL

Construction

```
1 int main() {
2   time_impl t;
3   output_console o;
4   alarm_clock<time_impl, output_console> alarm(t, o);
5
6   // ...
7
8   return 0;
9 }
```

COMPILE-TIME DEPENDENCY INJECTION

STIHL

Construction

```
1 int main() {
2   time_impl t;
3   output_console o;
4   alarm_clock alarm(t, o); // Class template argument deduction
5
6   // ...
7
8   return 0;
9 }
```

COMPILE-TIME DEPENDENCY INJECTION

STIHL

Testing

```
 1 TEST(alarm, checkAlarmAfter10Minutes) {
 2   time_fake t;
 3   output_fake o;
 4   alarm_clock alarm(t, o);
 5
 6   auto ring_time = t.now() + 10min;
 7   clock.set_alarm(ring_time);
 8
 9   t = t.now() + 10min;
10   EXPECT_EQ(o.value(), "Alarm\n");
11 }
```

COMPILE-TIME DEPENDENCY INJECTION

STIHL

Testing

```
1  TEST(alarm, checkAlarmAfter10Minutes) {
2    time_fake t;
3    output_fake o;
4    alarm_clock alarm(t, o);
5
6    auto ring_time = t.now() + 10min;
7    clock.set_alarm(ring_time);
8
9    t = t.now() + 10min;
10   EXPECT_EQ(o.value(), "Alarm\n");
11 }
```

COMPILE-TIME DEPENDENCY INJECTION

STIHL

Testing

```
 1 TEST(alarm, checkAlarmAfter10Minutes) {
 2   time_fake t;
 3   output_fake o;
 4   alarm_clock alarm(t, o);
 5
 6   auto ring_time = t.now() + 10min;
 7   clock.set_alarm(ring_time);
 8
 9   t = t.now() + 10min;
10   EXPECT_EQ(o.value(), "Alarm\n");
11 }
```

AGENDA

STIHL

1. ~~Alarm Clock Example~~ ⏲️
2. ~~Run-Time Dependency Injection~~
   ~~Splitting the Alarm Clock Example~~
3. ~~Compile-Time Dependency Injection~~
   ~~Splitting the Alarm Clock Example~~
4. Summary

SUMMARY

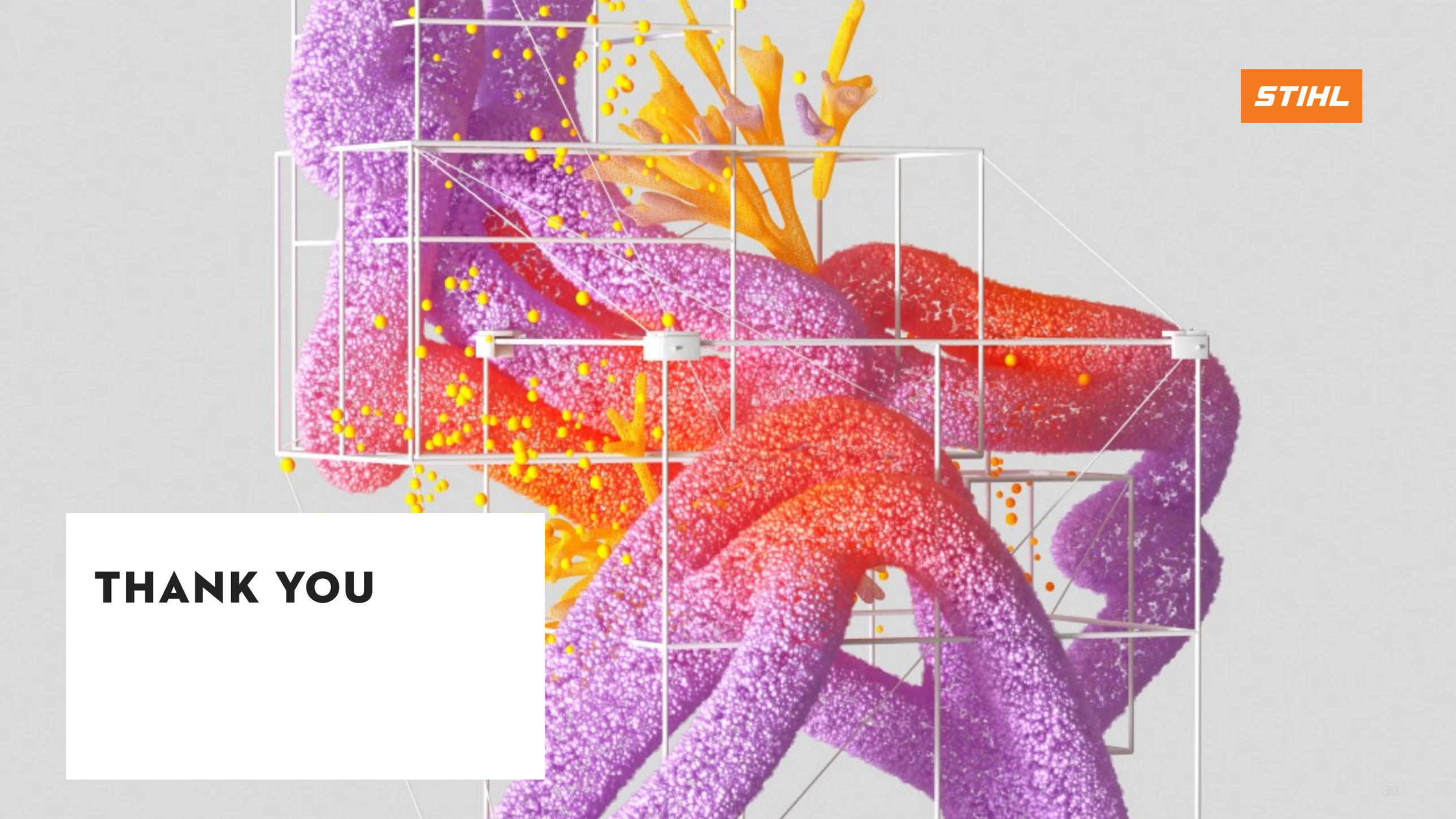
STIHL

Run-Time

- Implementation can be changed during run-time
- Clear interface definitions
- Virtual function calls

Compile-Time

- Concrete Implementations must be known at compile-time
- No direct interface definition
- No virtual function calls



THANK YOU