Deciphering C++ Coroutines A Diagrammatic Coroutine Cheat Sheet

Saliha Derin

Ömer Faruk Bitikçioğlu

Overview

Essential use cases for coroutines

Coroutines from the caller's perspective

Steps involved in starting a coroutine

Suspend and resume

Drawing a map of coroutine land

Interacting with coroutines

Coroutine Basics

- Function that can be paused
- Can be resumed later with surrounding state still intact
- Coroutine is always stateful - we need to remember where we left off
- Think of coroutines like factory functions returning a function object
- Stackless

Coroutine Basics

Asynchronous Computation

```
auto [ec, bytes_read] = read(socket, buffer);
// ...

auto [ec, bytes_read] =
        co_await async_read(socket, buffer);
// ...
```

```
auto [ec, bytes_read] = read(socket, buffer);
// ...

async_read(socket, buffer,
  [](std::error_code ec, std::size_t bytes_read) {
    // ...
  });
```

The smallest coroutine

```
struct ReturnType {
};
```

```
ReturnType hello_coroutine() {
  co_return;
}
```

Compiler Error: Compiler is looking for a nested type ReturnType::promise type.

Implementing the return object

```
struct ReturnType {
  struct promise_type {};
};
```

What is promise_type?

```
struct promise_type {

  ReturnType get_return_object() { return {}; }
  std::suspend_always initial_suspend() { return {}; }

  void return_void() { }
  void unhandled_exception() { };
  std::suspend_always final_suspend() noexcept { return {}; }

};
```

Awaitable

```
AsyncRead awaitable = async_read(socket, buffer);
auto [ec, bytes_read] = co_await awaitable;
```

```
struct Awaitable {
  bool await_ready();
  void await_suspend(std::coroutine_handle promise_type>);
  void await_resume();
};
```

```
struct SuspendAlways {
  bool await_ready() { return false; }
  void await_suspend(std::coroutine_handle<>) {}
  void await_resume() {}
};
```

The key components

- Return type
- Promise type
- Awaitable type
- std::coroutine handle<>

```
struct std::coroutine_handle<promise_type> {
  // ...
  void resume() const;
  void destroy() const;
  promise_type& promise() const;
  static coroutine_handle from_promise(promise_type&);
};
```

Cheatsheets

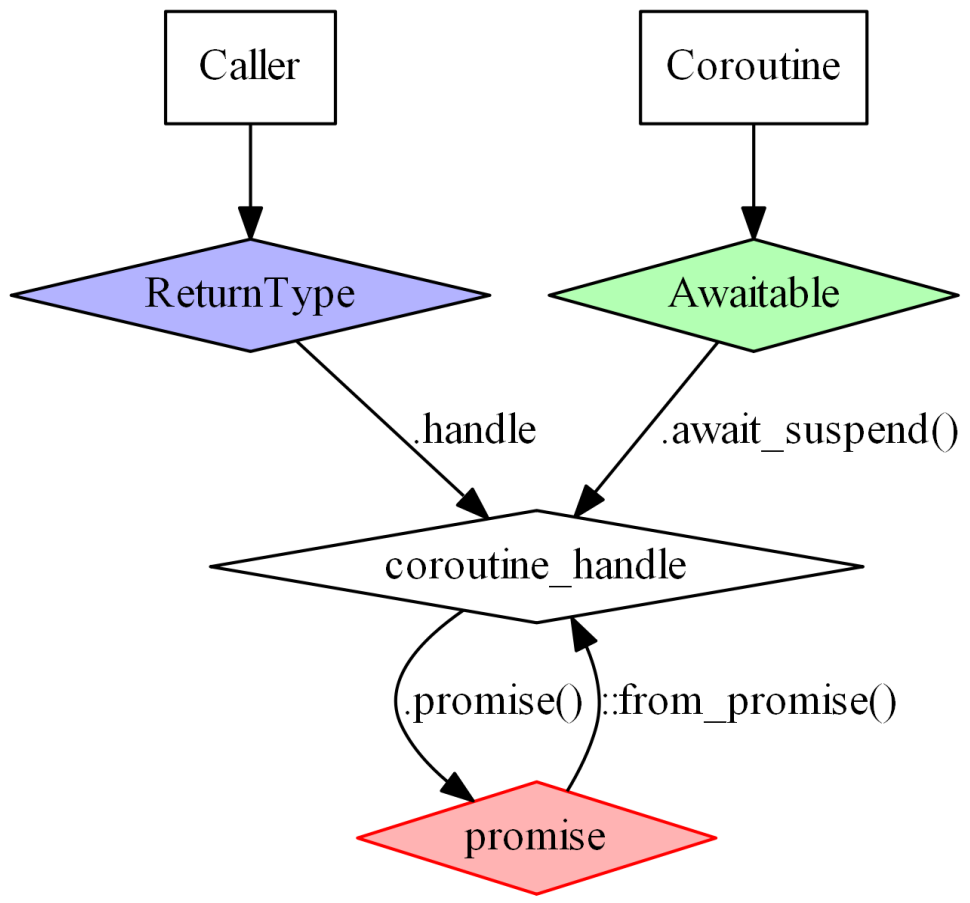
```
1  struct ReturnType / std::coroutine_traits<ReturnType, ...> {
2    struct promise_type {
3      promise_type(T...);   // opt.
4      ReturnType get_return_object();
5      std::suspend_always initial_suspend();
6      // ---- ⇑ Start / ⇓ Shutdown ----
7      void return_value(T); / void return_void();
8      void unhandled_exception();
9      std::suspend_always final_suspend() noexcept;

     };
   };
```

```
1  struct Awaitable {
2    bool await_ready();
3    void await_suspend(std::coroutine_handle <promise_type>);
4    void await_resume();
5  };
```

Start/Stop

Awaitable

Getting data out of a coroutine

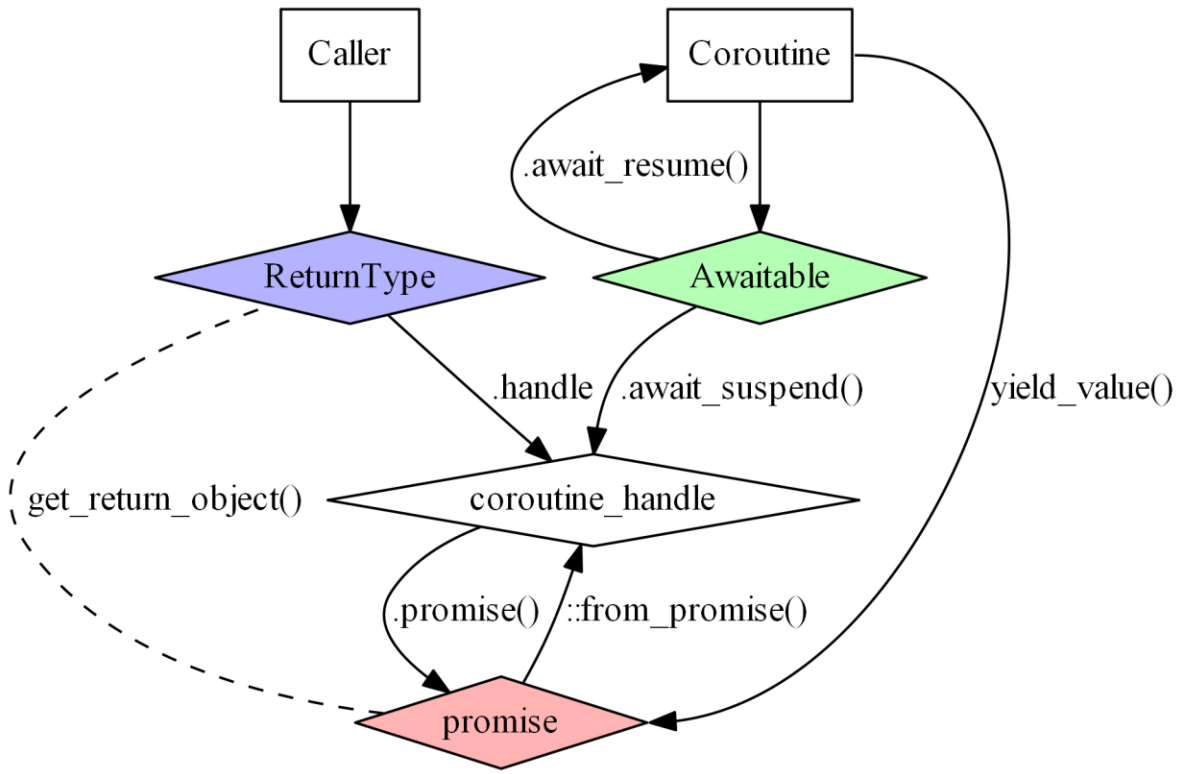



```
struct promise {
  // ...
  int value;
};

void TheAnswer::await_suspend(
       std::coroutine_handle<promise> h)
{
  h.promise().value = value_;
}
```

```
Coroutine f1() {
  // ...
  co_await TheAnswer{42};
}

TheAnswer::TheAnswer(int v)
:value_(v) {}
```

```
struct Coroutine {
  // ...
  std::coroutine_handle<promise> handle;
  int getAnswer() {
    return handle.promise().value;
  }
};
int main() {
  Coroutine c1 = f1();
  std::println("The answer is {}", c1.getAnswer());
}
```

Getting data into a coroutine




```
struct OutsideAnswer {
  bool await_ready() { return false; }
  void await_suspend(std::coroutine_handle<promise> h) {
    handle = h;
  }
  int await_resume() {
    return handle.promise().value;
  }

  std::coroutine_handle<promise> handle;
};
```

```
Coroutine f1() {
  int the_answer = co_await OutsideAnswer{};
}

int main() {
  Coroutine c1 = f1();
  c1.provide(42);
}
```

```
void Coroutine::provide(int the_answer) {
  handle.promise().value = the_answer;
  handle.resume();
}
```

Yielding values

```
struct promise_type {
  // ...
  NewNumberAwaitable yield_value(int i) {
    return NewNumberAwaitable{ i };
  }
};
```

```
struct promise_type {
  // ...
  int value;
  std::suspend_always yield_value(int i) {
    value = i;
    return {};
  }
};
```

```
FiboGenerator makeFiboGenerator() {
  int i1 = 1;
  int i2 = 1;
  while (;;) {
    co_await NewNumberAwaitable{ i1 };
    i1 = std::exchange(i2, i1 + i2);
  }
}
```

```
FiboGenerator makeFiboGenerator() {
  int i1 = 1;
  int i2 = 1;
  while (;;) {
    co_yield i1;
    i1 = std::exchange(i2, i1 + i2);
  }
}
```

Symmetric transfer

```
co_await Transfer{};

struct promise {
  // ...
  std::coroutine_handle<promise> other;
};

std::coroutine_handle<> Transfer::await_suspend(
                std::coroutine_handle<promise> me)
{
  return me.promise().other ? me.promise().other : me;
}
```

THANK YOU

Saliha Derin

Ömer Faruk Bitikçioğlu