C++20 Coroutines

What's next?

Dawid Pilarski

dawid.pilarski@panicsoftware.com
blog.panicsoftware.com
dawid.pilarski@tomtom.com

Introduction

Agenda

Introduction

Quick refresh about the coroutines.

Missing coroutines parts

Questions...

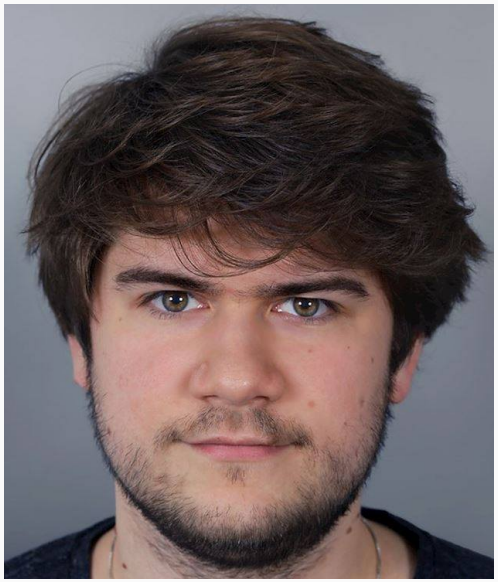
Time is rather tight.

Please hold your questions till the end.

Who am I?

Dawid Pilarski

- Senior Software Developer in TomTom
- Member of the ISO/JTC1/SC22/WG21
- Member of the PKN KT **(programming languages)**
- C++ blog writer



Quick refresh about the coroutines.

What are the coroutines?

Subroutine Is a sequence of program instructions that performs a specific task, packaged as a unit.

Function Is a subroutine

Coroutine Is generalization of the function.

What are the coroutines?

Function can be:

- called
- returned from

What are the coroutines?

Coroutine can be:

- called
- returned from
- suspended

What are the coroutines?

Coroutine can be:

- called
- returned from
- suspended
- resumed from

What are the coroutines?

Coroutine can be:

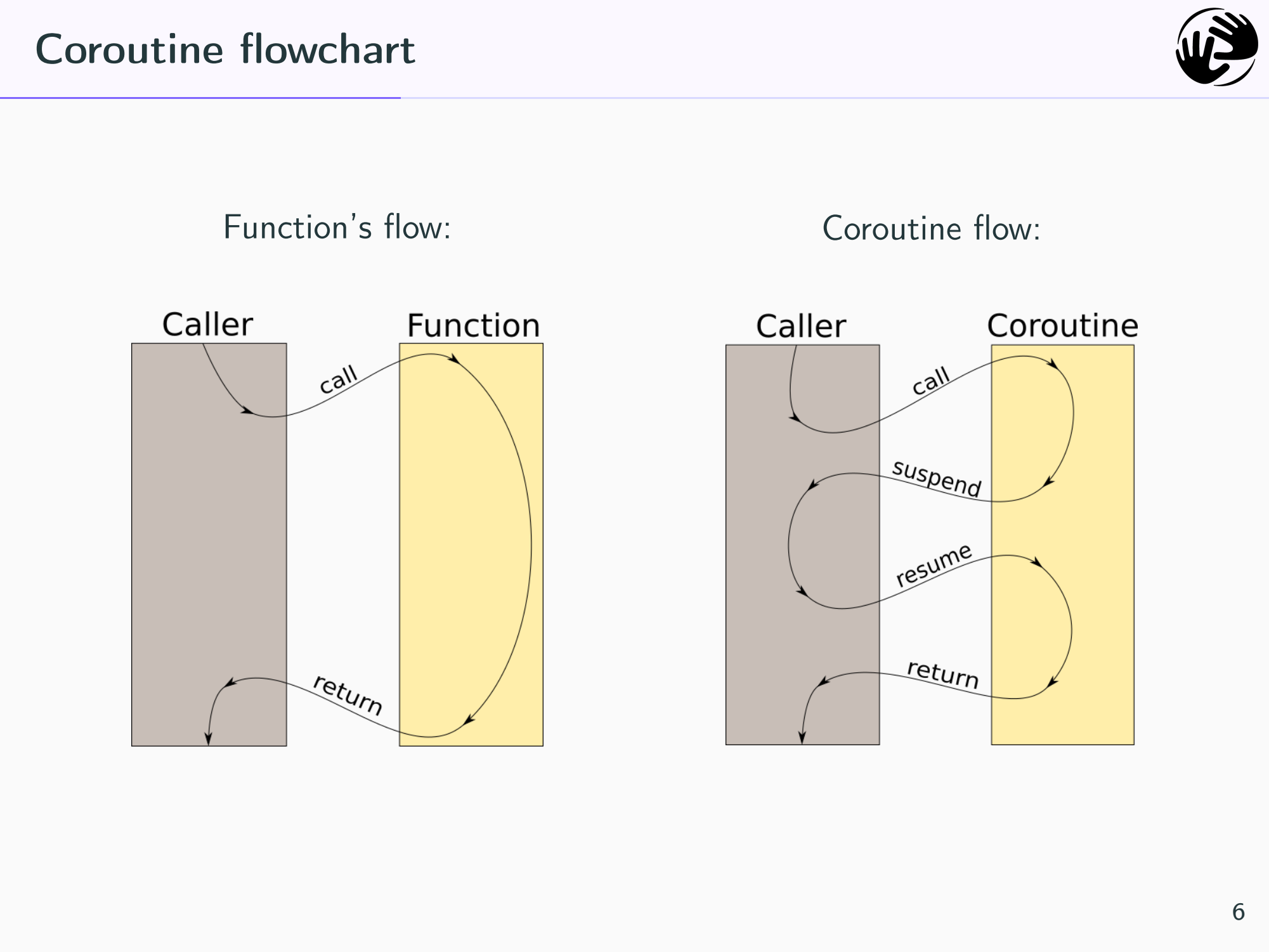
- called
- returned from
- suspended
- resumed from
- created

What are the coroutines?

Coroutine can be:

- called
- returned from
- suspended
- resumed from
- created
- destroyed

Coroutine flowchart



Function's flow:

Coroutine flow:

How to implement custom coroutine types.

Creating custom coroutine type is not easy:

- C++ provides keywords only.

How to implement custom coroutine types.

Creating custom coroutine type is not easy:

- C++ provides keywords only.
- Developer must implement what keywords do.

How to implement custom coroutine types.

Creating custom coroutine type is not easy:

- C++ provides keywords only.
- Developer must implement what keywords do.

This means:

- Implementation of promise_type (~6 functions)

How to implement custom coroutine types.

Creating custom coroutine type is not easy:

- C++ provides keywords only.
- Developer must implement what keywords do.

This means:

- Implementation of promise_type (~6 functions)
- Implementation of the co_await keyword (~3 functions)

How to implement custom coroutine types.

Creating custom coroutine type is not easy:

- C++ provides keywords only.
- Developer must implement what keywords do.

This means:

- Implementation of promise_type (~6 functions)
- Implementation of the co_await keyword (~3 functions)

You need to remember to implement on average 9 functions.

Coroutine declaration

```
// returned-type   name       arguments
//|-------------| |-------| |--------------|
   generator<int> fibonacci (int from_value);
```

- Whether the function is a coroutine depends on it's definition.

Coroutine declaration

```
// returned-type   name       arguments
//|-------------| |-------| |--------------|
   generator<int> fibonacci (int from_value);
```

- Whether the function is a coroutine depends on it's definition.
- If function is a coroutine it's return type must support coroutines.

Promise_type

Type supports coroutines if it has promise_type.

promise_type can be:

- member of the class
- member of the specialization of the coroutine_traits<returned_type>

Promise_type

Promise_type controls coroutine's behavior.

- `awaitable initial_suspend();`
- `suspension at the beginning`

Promise_type

Promise_type controls coroutine's behavior.

- `awaitable initial_suspend();`
- `awaitable final_suspend();`

- `suspension at the beginning`
- `suspension at the end`

Promise_type

Promise_type controls coroutine's behavior.

- `awaitable initial_suspend();`
- `awaitable final_suspend();`
- `return_type get_return_object();`

- `suspension at the beginning`
- `suspension at the end`
- `how to create return_type`

Promise_type

Promise_type controls coroutine's behavior.

- `awaitable initial_suspend();`
- `awaitable final_suspend();`
- `return_type get_return_object();`
- `void unhandled_exception();`

- `suspension at the beginning`
- `suspension at the end`
- `how to create return_type`
- `handling unhandled exception`

Keywords and promise_type

Promise_type is also responsible for keyword's actions:

- `co_return V;`
- `p.return_value(V);`

Keywords and promise_type

Promise_type is also responsible for keyword's actions:

- `co_return V;`
- `co_return;`

- `p.return_value(V);`
- `p.return_void();`

Keywords and promise_type

Promise_type is also responsible for keyword's actions:

- `co_return V;`
- `co_return;`
- `co_yield V;`

- `p.return_value(V);`
- `p.return_void();`
- `co_await p.yield_value();`

co_await

In order to support co_await expressions, the argument (awaitable) must:

- have `awaiter operator co_await` defined, or

co_await

In order to support co_await expressions, the argument (awaitable) must:

- have `awaiter operator co_await` defined, or
- have global `awaiter operator co_await(A)` support, or

co_await

In order to support co_await expressions, the argument (awaitable) must:

- have `awaiter operator co_await` defined, or
- have global `awaiter operator co_await(A)` support, or
- implement 3 functions:

co_await

In order to support co_await expressions, the argument (awaitable) must:

- have `awaiter operator co_await` defined, or
- have global `awaiter operator co_await(A)` support, or
- implement 3 functions:
  - `bool await_ready()`

co_await

In order to support co_await expressions, the argument (awaitable) must:

- have `awaiter operator co_await` defined, or
- have global `awaiter operator co_await(A)` support, or
- implement 3 functions:
  - `bool await_ready()`
  - `await_suspend(coroutine_handle<P>)` returning

co_await

In order to support co_await expressions, the argument (awaitable) must:

- have `awaiter operator co_await` defined, or
- have global `awaiter operator co_await(A)` support, or
- implement 3 functions:
  - `bool await_ready()`
  - `await_suspend(coroutine_handle<P>)` returning
    - void

co_await

In order to support co_await expressions, the argument (awaitable) must:

- have `awaiter operator co_await` defined, or
- have global `awaiter operator co_await(A)` support, or
- implement 3 functions:
  - `bool await_ready()`
  - `await_suspend(coroutine_handle<P>)` returning
    - void
    - bool

co_await

In order to support co_await expressions, the argument (awaitable) must:

- have `awaiter operator co_await` defined, or
- have global `awaiter operator co_await(A)` support, or
- implement 3 functions:
  - `bool await_ready()`
  - `await_suspend(coroutine_handle<P>)` returning
    - void
    - bool
    - another `coroutine_handle`

co_await

In order to support co_await expressions, the argument (awaitable) must:

- have `awaiter operator co_await` defined, or
- have global `awaiter operator co_await(A)` support, or
- implement 3 functions:
  - `bool await_ready()`
  - `await_suspend(coroutine_handle<P>)` returning
    - void
    - bool
    - another `coroutine_handle`
  - `T await_resume()`

Missing coroutines parts

Type erasure

asynchronous RAII

RVO on coroutines

return value [and|or] return void

Thank you for attention

Special thank you! goes to:

- Gor Nishanov
- Lewiss Baker

Bibliography and further reading

- Lewiss Baker's Assymetric transfer blog
- newest C++ draft
- My blog - blog.panicsoftware.com
- James McNellis - "Introduction to the C++ Coroutines"
- Gor Nishanov - any video about the coroutines
- Toby Allsopp - "Coroutines: what can't they do?"

Questions?

Questions?