Coroutines in the C++

Why and How?

Dawid Pilarski

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

Introduction

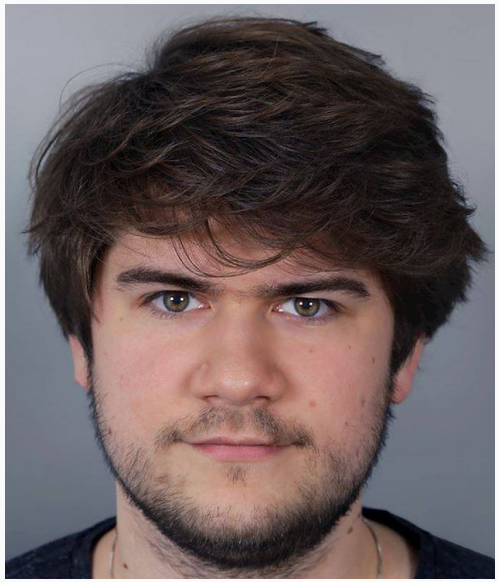
Agenda

Questions...

Ask questions any time.
Don't be afraid to interrupt me :)

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

Why do we need 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
- suspended

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

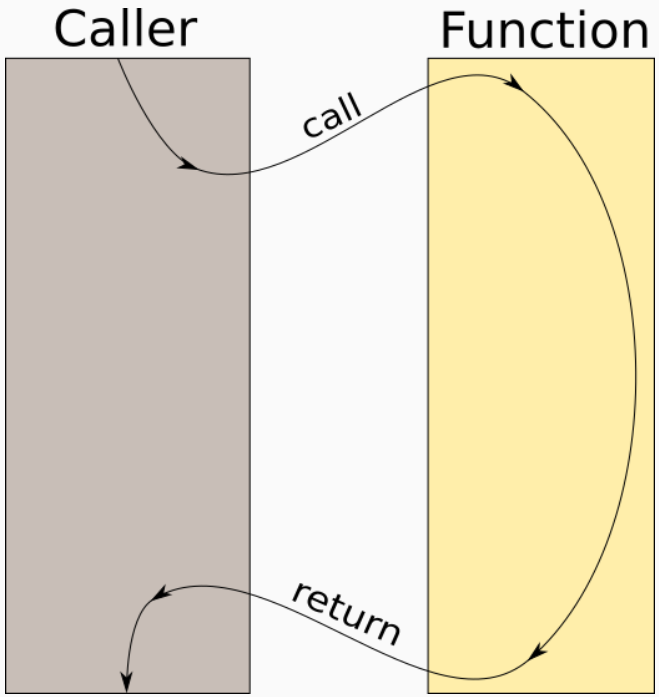
What are the coroutines?

Coroutine can be:

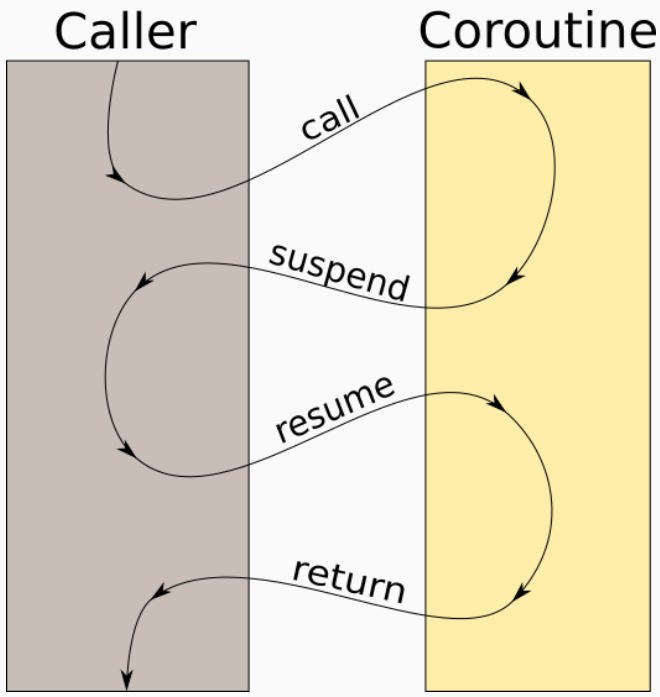
- called
- returned from
- suspended
- resumed from
- created
- destroyed

Coroutine flowchart

Function's flow:



Coroutine flow:

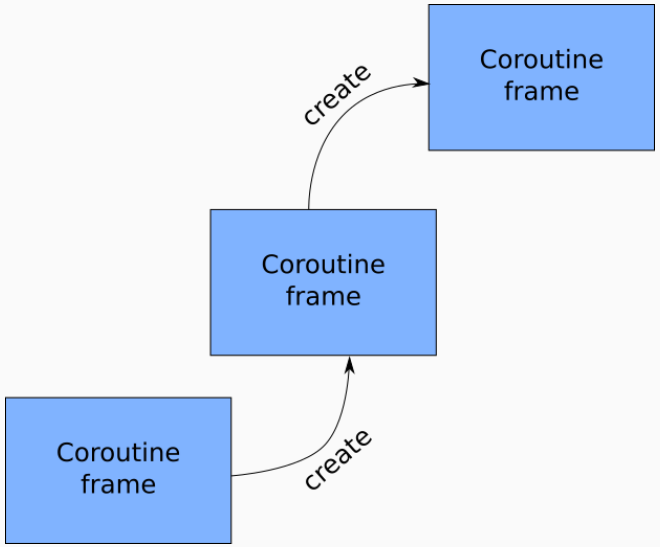


Coroutines use cases

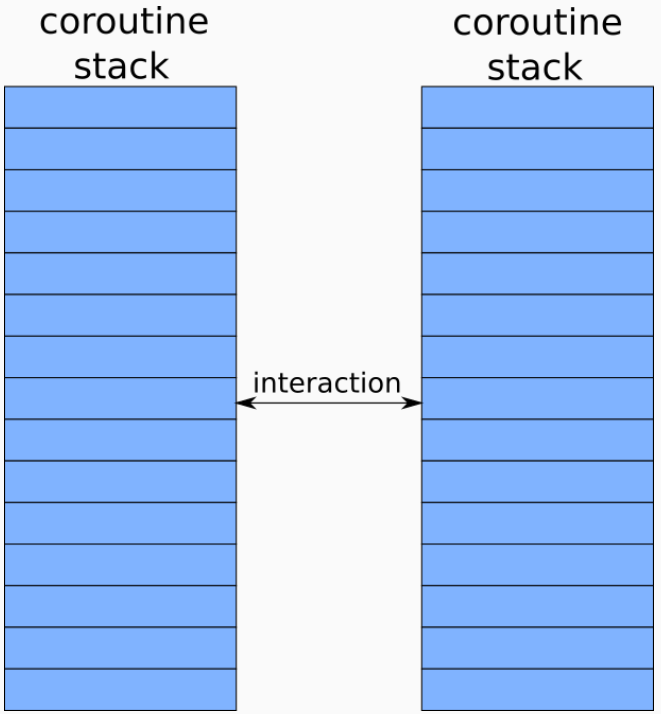
Why do we need language support for the coroutines

C++ coroutines vs library coroutines

Language based



Library based



C++ vs library - summary

- Need to allocate the stack for the Fiber/Coroutine

C++ vs library - summary

- Need to allocate the stack for the Fiber/Coroutine
- Can be suspended from the top level functions and below

C++ vs library - summary

- Need to allocate the stack for the Fiber/Coroutine
- Can be suspended from the top level functions and below
- Allocation of the memory in advance

C++ vs library - summary

- Need to allocate the stack for the Fiber/Coroutine
- Can be suspended from the top level functions and below
- Allocation of the memory in advance
- One allocation per whole stack.

C++ vs library - summary

Boost.Fiber

- Need to allocate the stack for the Fiber/Coroutine
- Can be suspended from the top level functions and below
- Allocation of the memory in advance
- One allocation per whole stack.

built-in coroutines

- Need to allocate the frame for the Coroutine

C++ vs library - summary

Boost.Fiber

- Need to allocate the stack for the Fiber/Coroutine
- Can be suspended from the top level functions and below
- Allocation of the memory in advance
- One allocation per whole stack.

built-in coroutines

- Need to allocate the frame for the Coroutine
- Can be suspended only from the top level function

C++ vs library - summary

Boost.Fiber

- Need to allocate the stack for the Fiber/Coroutine
- Can be suspended from the top level functions and below
- Allocation of the memory in advance
- One allocation per whole stack.

built-in coroutines

- Need to allocate the frame for the Coroutine
- Can be suspended only from the top level function
- Minimal memory allocations

C++ vs library - summary

Boost.Fiber

- Need to allocate the stack for the Fiber/Coroutine
- Can be suspended from the top level functions and below
- Allocation of the memory in advance
- One allocation per whole stack.

built-in coroutines

- Need to allocate the frame for the Coroutine
- Can be suspended only from the top level function
- Minimal memory allocations
- Multiple allocations

How to implement your own coroutine types?

What's the issue?

Creating custom coroutine type is not easy:

- C++ provides keywords only.

What's the issue?

Creating custom coroutine type is not easy:

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

What's the issue?

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)

What's the issue?

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)

What's the issue?

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);
```

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
- specialization of the coroutine_traits<returned_type>

Keywords

co_await

Thank you for attention

Recommended further readings

Questions?

Questions?