Mitigating lifetime issues in C++20 coroutines

Utkarsh Saxena
Google

Coroutines in C++20

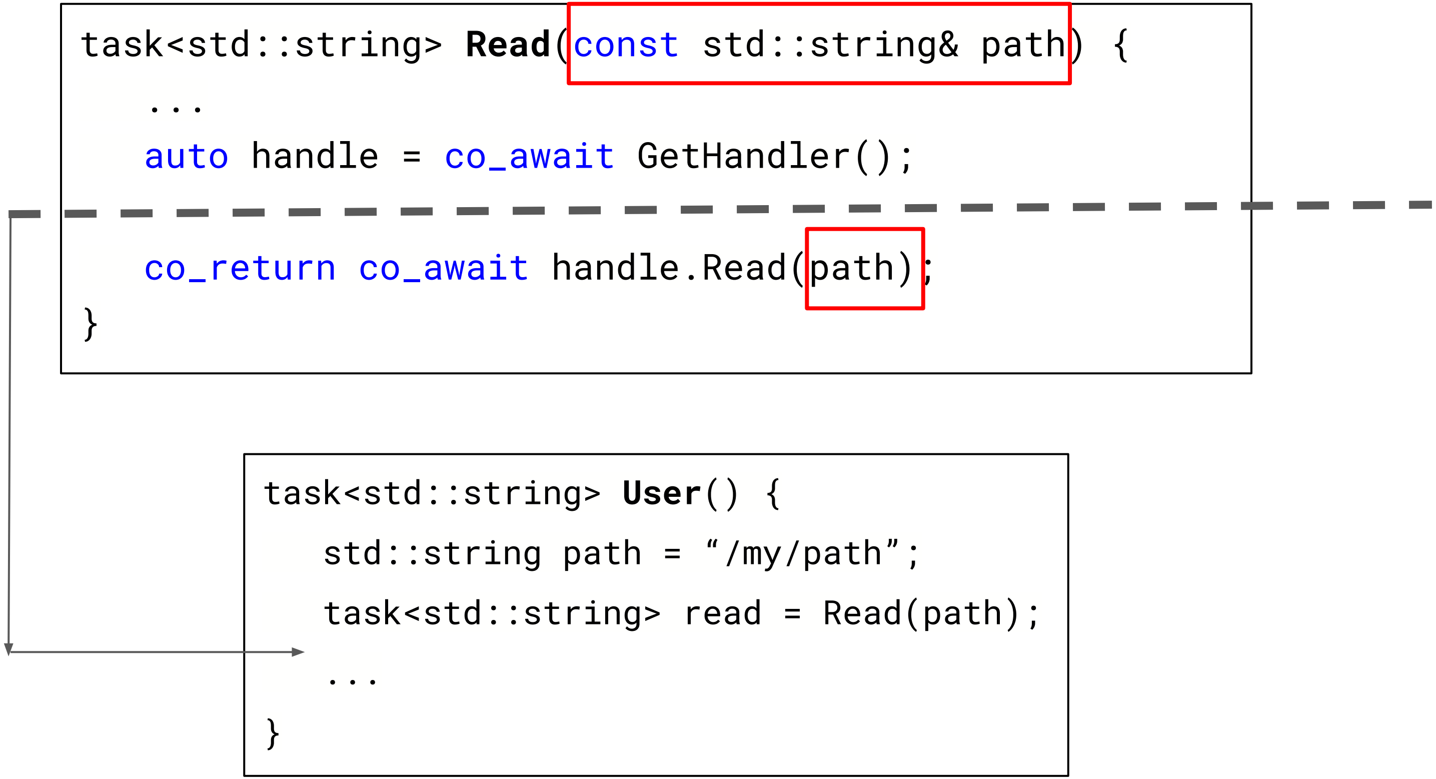
- Suspendable functions
  - Can suspend themselves.
  - Other entities can resume them.
- Stateful
  - Stores the state (local variable, resume points)
- Stackless

```
task<std::string> Read(const std::string& path) {
   auto handle = co_await GetFileHandler();
   co_return co_await handle.Read(path);
}
```

```
task<std::string> User() {
   std::string path = “/path/to/file”;
   std::string content = co_await Read(path);
}
```

Lifetime issues:
What can go wrong ?

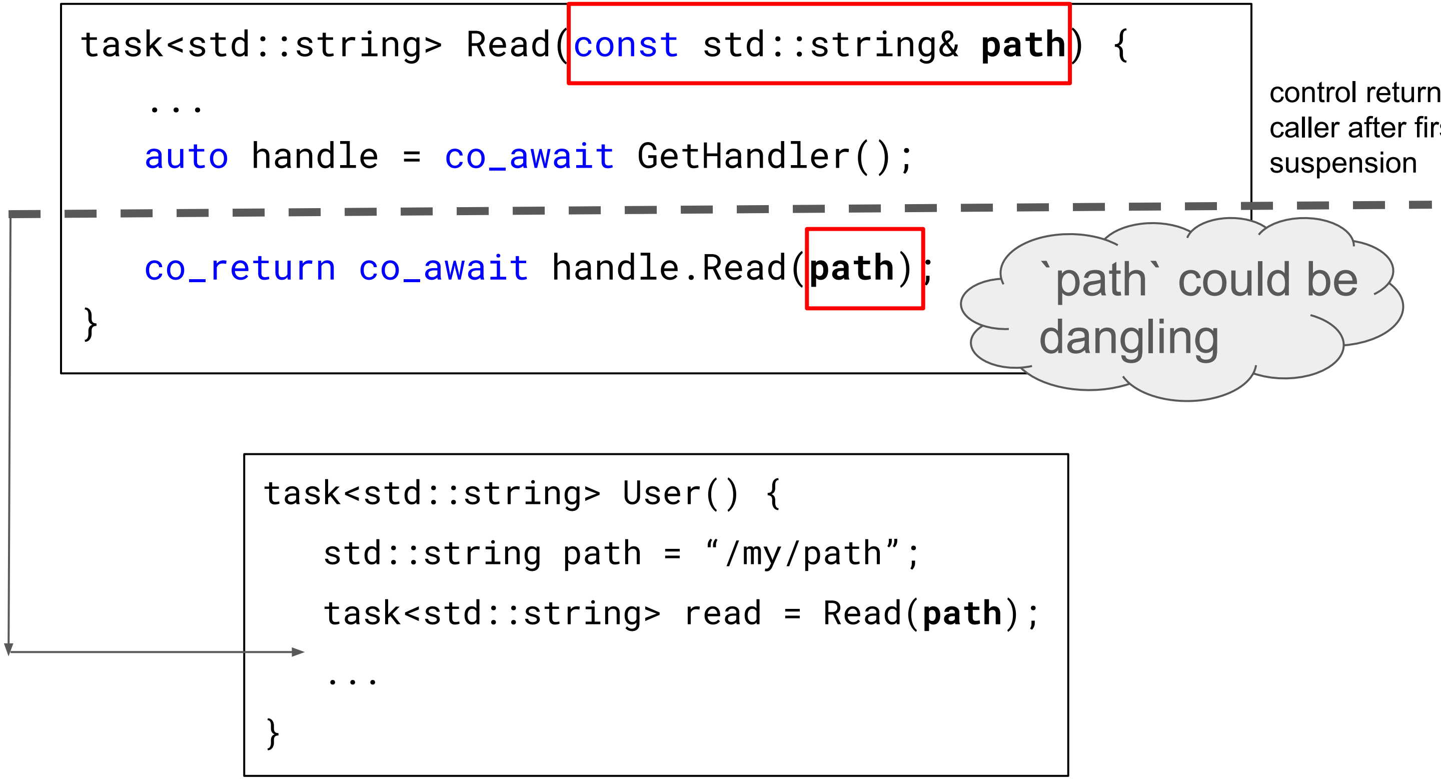
Control flow



```
task<std::string> Read(const std::string& path) {
    ...
    auto handle = co_await GetHandler();

    co_return co_await handle.Read(path);
}
```

```
task<std::string> User() {
    std::string path = “/my/path”;
    task<std::string> read = Read(path);
    ...
}
```

Control flow: Dangling references



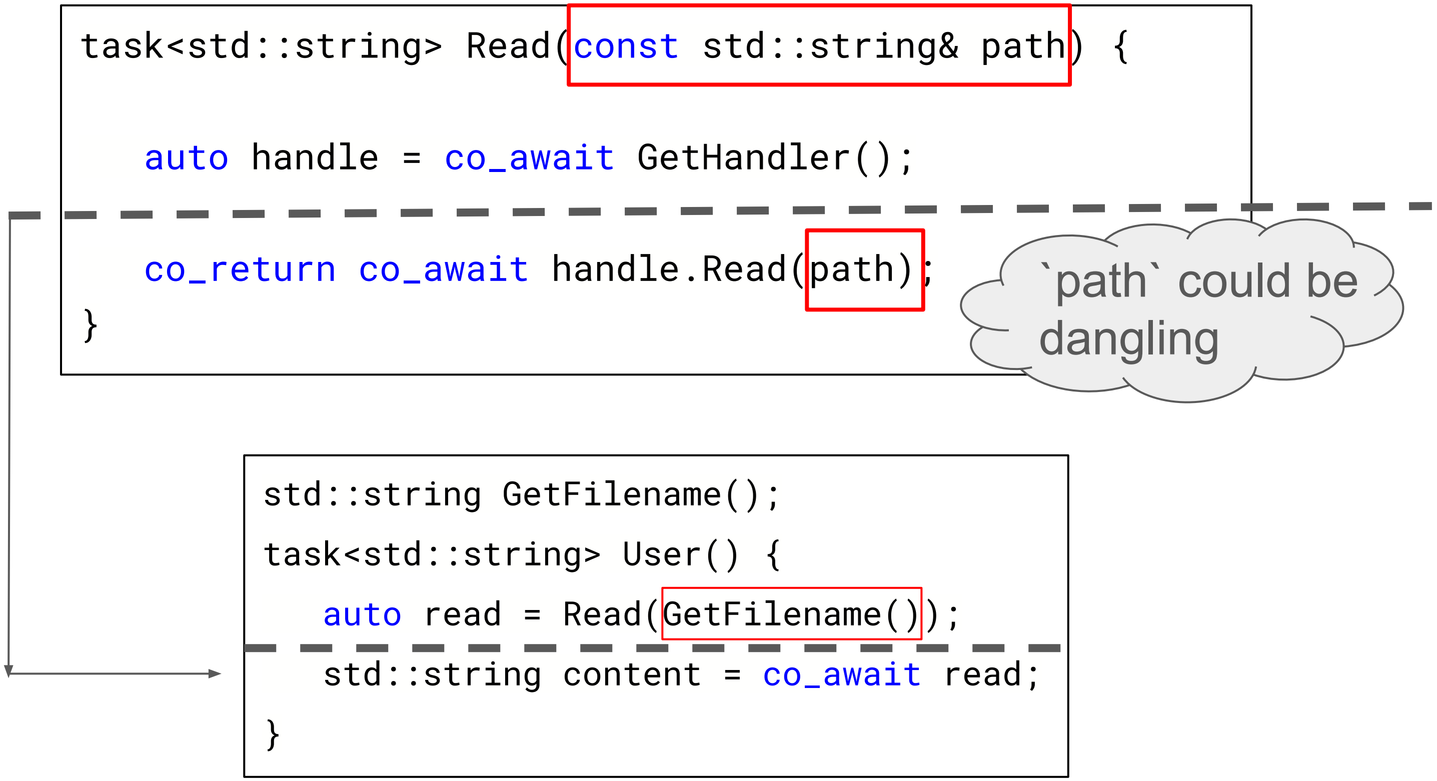
```
task<std::string> Read(const std::string& path) {
   ...
   auto handle = co_await GetHandler();

   co_return co_await handle.Read(path);
}
```

control returns back to caller after first suspension

`path` could be dangling

```
task<std::string> User() {
   std::string path = “/my/path”;
   task<std::string> read = Read(path);
   ...
}
```

Dangling reference to temporaries

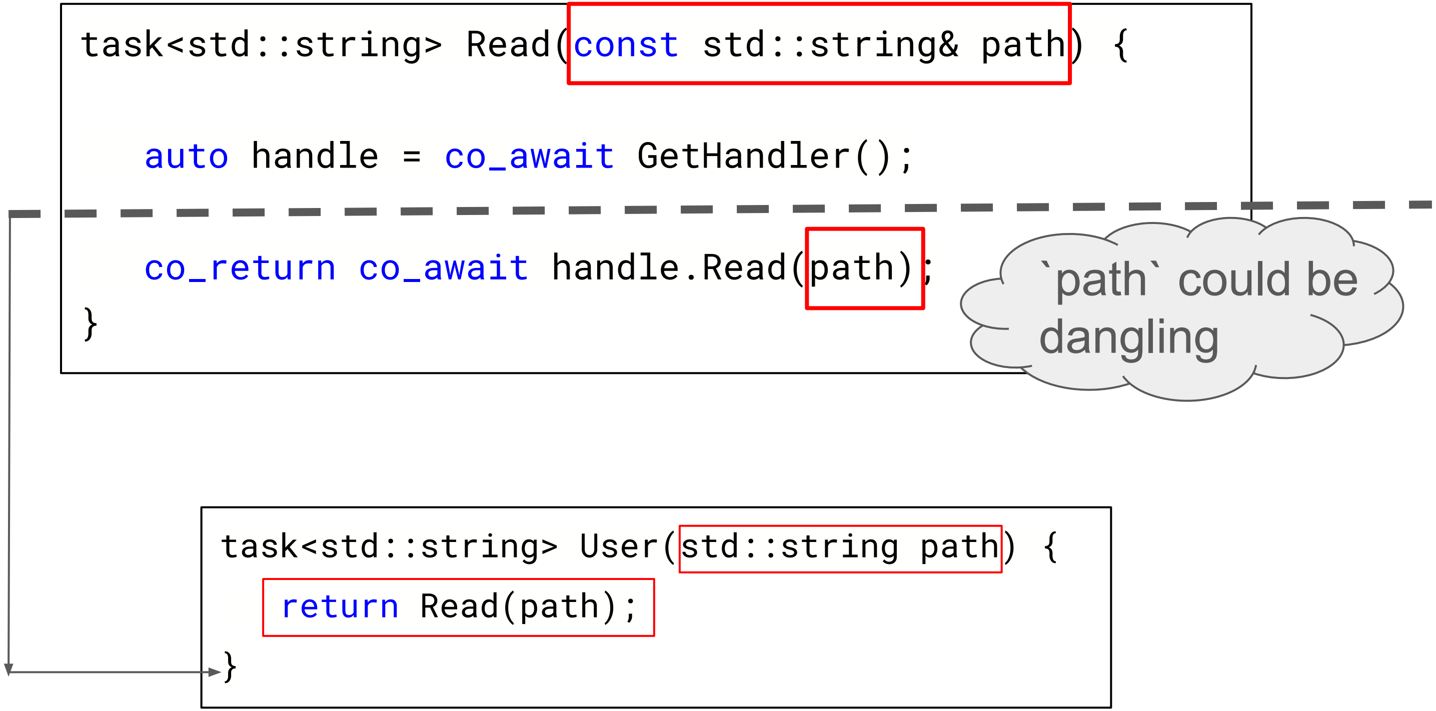


```
task<std::string> Read(const std::string& path) {

   auto handle = co_await GetHandler();

   co_return co_await handle.Read(path);
}
```

`path` could be dangling

```
std::string GetFilename();
task<std::string> User() {
   auto read = Read(GetFilename());
   std::string content = co_await read;
}
```

Dangling reference to stack variable



```
task<std::string> Read(const std::string& path) {

   auto handle = co_await GetHandler();

   co_return co_await handle.Read(path);
}
```

`path` could be dangling

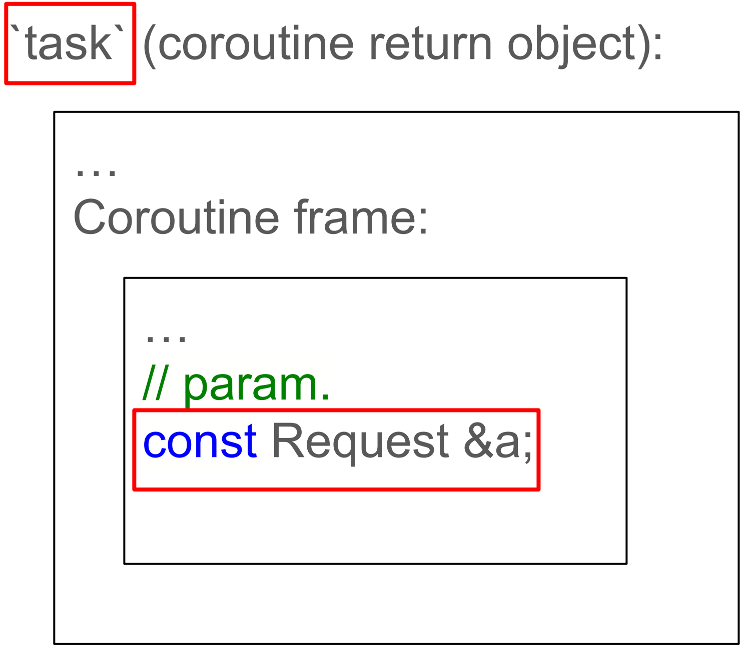
```
task<std::string> User(std::string path) {
   return Read(path);
}
```

Statically detecting lifetime issues

Condition to check

```
struct Request { int num; };


task<int> Add(const Request& a) {
   co_return a.num + 1;
}
```

```
// Ref to temporary.
task<int> foo = Add(Request{0});
```



`task` (coroutine return object):

...

Coroutine frame:

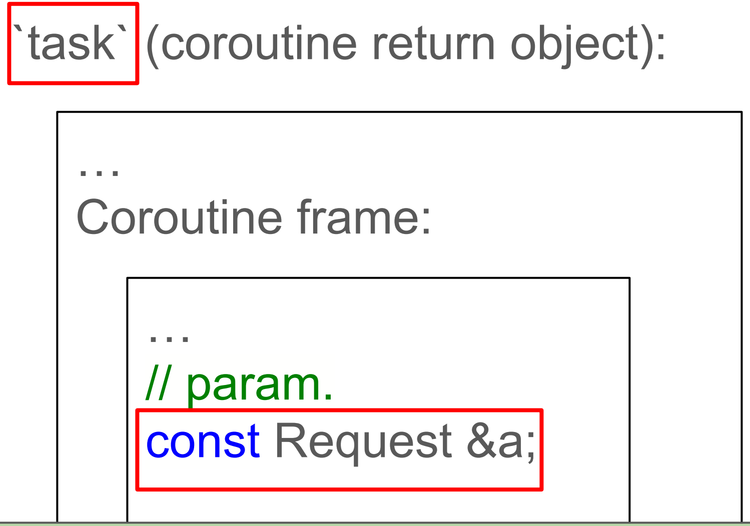
...

// param.

const Request &a;

Condition to check

```
struct Request { int num; };


task<int> Add(const Request& a) {
   co_return a.num + 1;
}
```



`task` (coroutine return object):

...
Coroutine frame:

...
// param.
const Request &a;

```
// Ref to temporary.
task<int> foo = Add(Request{0});
```

The lifetime of **argument** to parameter **`a`** must outlive the return object **`task`**.

This is not new to C++

```
struct Result { const int& x; };


Result Foo(const int& x) {
   return Result{x};
}


int Bar() {
   Result R = Foo(0);
   return R.x;
}
```

This is not new to C++

```
struct Result { const int& x; };

Result Foo(const int& x) {
    return Result{x};
}

int Bar() {
    Result R = Foo(0);
    return R.x;
}
```

AddressSanitizer: stack-use-after-scope

This is not new to C++ : [[clang::lifetimebound]]

```
struct Result { const int& x; };


Result Foo([[clang::lifetimebound]]const int& x) {
   return Result{x};
}


int Bar() {
   Result R = Foo(0);
   return R.x;
}
```

```
warning: temporary whose address is used as value of
local variable R will be destroyed at the end of the
full-expression [-Wdangling]
  16 |     Response R = Foo(0);
     |                       ^~~
```

Introducing [[clang::coro_lifetimebound]]

```
co_task<int> Add(const Request& a) {
   co_return a.num + 1;
}
```

Implicitly lifetime bound

Introducing [[clang::coro_lifetimebound]]

```
co_task<int> Add(const Request& a) {
   co_return a.num + 1;
}
```

Implicitly lifetime bound

```
template <typename T = void>
struct [[clang::coro_return_type, clang::coro_lifetimebound]]
co_task { /**/ };
```

“Coroutine return type”

Lifetime bound coroutines: Plain returns

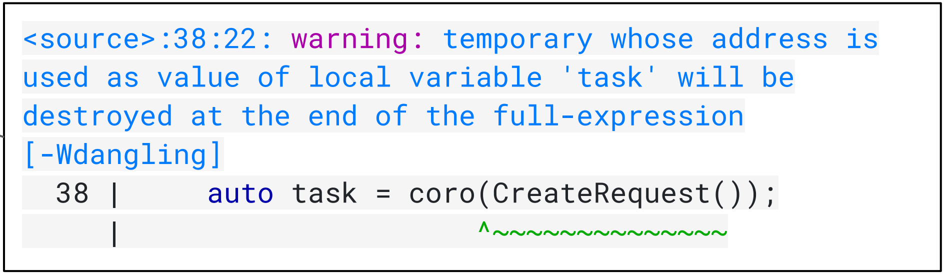
```cpp
co_task<int> coro(const int& n) {
    co_return n+1;
}

co_task<int> user(int n) {
    return coro(n);
}
```

```
<source>:31:17: warning: address of stack memory
associated with parameter 'n' returned
[-Wreturn-stack-address]
   31 |     return coro(n);
```

Lifetime bound coroutines: Temporaries

```
co_task<int> coro(const Request& r) {
   co_return r.num + 1;
}
```

```
Request CreateRequest();


co_task<int> user() {
   auto task = coro(CreateRequest());
   co_return co_await task;
}
```



```
<source>:38:22: warning: temporary whose address is
used as value of local variable 'task' will be
destroyed at the end of the full-expression
[-Wdangling]
  38 |     auto task = coro(CreateRequest());
     |                      ^~~~~~~~~~~~~~~
```

Future work: control flow

```
co_task<int> coro(const Request& r) {
   co_return r.n;
}

co_task<int> user(Request r) {
   auto task = coro(r);
   return task;
}
```

Not detected

Thank you

Utkarsh Saxena
Google