The Promise Workflow

The compiler transforms a coroutine into the following workflow.

```
{
    Promise prom;
    co_await prom.initial_suspend();
    try {
        <function body having co_return, co_yield, or co_await>
    }
    catch (...) {
        prom.unhandled_exception();
    }
FinalSuspend:
    co_await prom.final_suspend();
}
```

The Awaiter Workflow

The compiler creates the following workflow based on the Awaiter.

```
awaiter.await_ready() returns false:
    suspend coroutine
    awaiter.await_suspend(coroutineHandle) returns:
        void:
        bool:
        another coroutine handle:
resumptionPoint:
return awaiter.await_resume();
```

Return value of `awaitable.await_suspend()`	Description
`void`	The coroutine keeps suspended and returns to the caller.
`bool == true`	The coroutine keeps suspended and returns to the caller.
`bool == false`	The coroutine is resumed and does not return to the caller.
`anotherCoroutineHandle`	The other coroutine is resumed and returns to the caller.

`eagerFutureWithComments.cpp`
`lazyFuture.cpp`
`lazyFutureOnOtherThread.cpp`