*Coroutine NOSIL Suspend Edilir: • Promise nesnesinin fonksiyonu çağrılarak, implicitly (örtülü olarak) co_await
• Explicit olarak, co_await / co_yield çağırılır!

```
co_yield expr;
co_avait prom.yield_value(expr);
```

yield arkada değer olan bir await

*co_await: • Amacı coroutine'i suspend etmek
• Derleyici awaiter tipinden nesne elde etmek zorunda
• Derleyici co_await kelimesi awaiter tipinden nesnenin fonksiyonlarından birini çağırı yapar!

```
awaiter
    await_ready
    await_suspend
    await_resume
```

→ Bir sınıf bu fonksiyonlara sahipse Bir awaiter sınıftır!

*Cpp'de: - 2 adet default awaiter var!

```
std::suspend_always
std::suspend_never
```

→ co_await operatörünün operandı olurlar!

• susp-always: programın akışı, coroutine'i çağıran yerden / suspend kabul edilir.
susp-never: // , coroutine yerden / suspend kabul edilmez!

```
struct suspend_always {
    constexpr bool await_ready() const noexcept { return false; }
    constexpr void await_suspend(std::coroutine_handle<>) const noexcept {}
    constexpr void await_resume() const noexcept {}
}

struct suspend_never {
    constexpr bool await_ready() const noexcept { return true; }
    constexpr void await_suspend(std::coroutine_handle<>) const noexcept {}
    constexpr void await_resume() const noexcept {}
}
```

* Derleyicinin co_await kurallığı yönden kod:

```
template<typename P, typename T>
decltype(auto) get_awaitable(P& promise, T&& expr)
{
  if constexpr (has_any_await_transform_member_v<P>)
    return promise.await_transform(static_cast<T&&>(expr));
  else
    return static_cast<T&&>(expr);
}

template<typename Awaitable>
decltype(auto) get_awaiter(Awaitable&& awaitable)
{
  if constexpr (has_member_operator_co_await_v<Awaitable>)
    return static_cast<Awaitable&&>(awaitable).operator co_await();
  else if constexpr (has_non_member_operator_co_await_v<Awaitable&&>)
    return operator co_await(static_cast<Awaitable&&>(awaitable));
  else
    return static_cast<Awaitable&&>(awaitable);
}
```

awaitable elde etmeye çalışıyoruz!

↳ eğer expr'in await transform func'ı varsa, onu çağırır | yoksa expr awaitable olarak alınır!

```
co_await expr;
```

1. ilgili promise sınıfının, await_transform fonksiyonu var mı
   ↳ eğer varsa: expr, o fonksiyona gönderilir. Return değeri awaitable olarak kullanılır!
   ↳ eğer yoksa; expr'in kendisine awaitable olarak kullanılır!

↓ awaitable elde edildikten sonra, awaiter elde edilir!

2. Awaitable'ın, co_await'i var mı diye bakılır.
   ↳ eğer yoksa; global co_await
   ↳ eğer bu da yoksa; expr'in kendisi awaiter olur!

• Derleyici sırasıyla → calls await-ready:

```
bool awaiter::await_ready()
{
    return false;
}
```

→ coroutine'i durdurmadan hemen önce çağırır!

→ false olduğu için, suspension kabul edilir.

→ true ise coroutine hiç suspend edilmez!

→ Eğer suspension kabul edilmişse: await-suspend çağırılır:

```
void Awaiter::await_suspend(std::coroutine_handle<>)
{
}
```

→ Çağıran kodda geri dönmek için, return değeri void olmalı!

```
bool Awaiter::await_suspend(std::coroutine_handle<>)
{
}
```

→ eğer bool return ederse suspend edilsin mi / edilmesin mi?

- true: suspension kabul edilir
- false " kabul edilmez!

→ Suspend edilmiş coroutine'de: await-resume

• Derleyici coroutine çalışmadan önce çağırır.

```
struct Awaiter {

    bool await_ready()const
    {
        std::cout << "Awaiter::await_ready()\n";
        return false;
    }

    void await_suspend(std::coroutine_handle<>)const
    {
        std::cout << "Awaiter::await_suspend()\n";
    }

    void await_resume()const
    {
        std::cout << "Awaiter::await_resume()\n";
    }
};
```

Burada asıncıda istediğiniz kodu çalıştırabilirsiniz!

Çünkü suspend edip ne yapacağınızı burada yazabilirsiniz!

```
CoroFace foo()
{
    std::cout << "coroutine started! [1]\n";
    co_await Awaiter{};
    std::cout << "coroutine resumed! [2]\n";
}
```

```cpp
int main()
{

    std::cout << "main [1]\n";
    auto crface = foo();
    std::cout << "main [2]\n";
    crface.resume();
    std::cout << "main [3]\n";

}
```