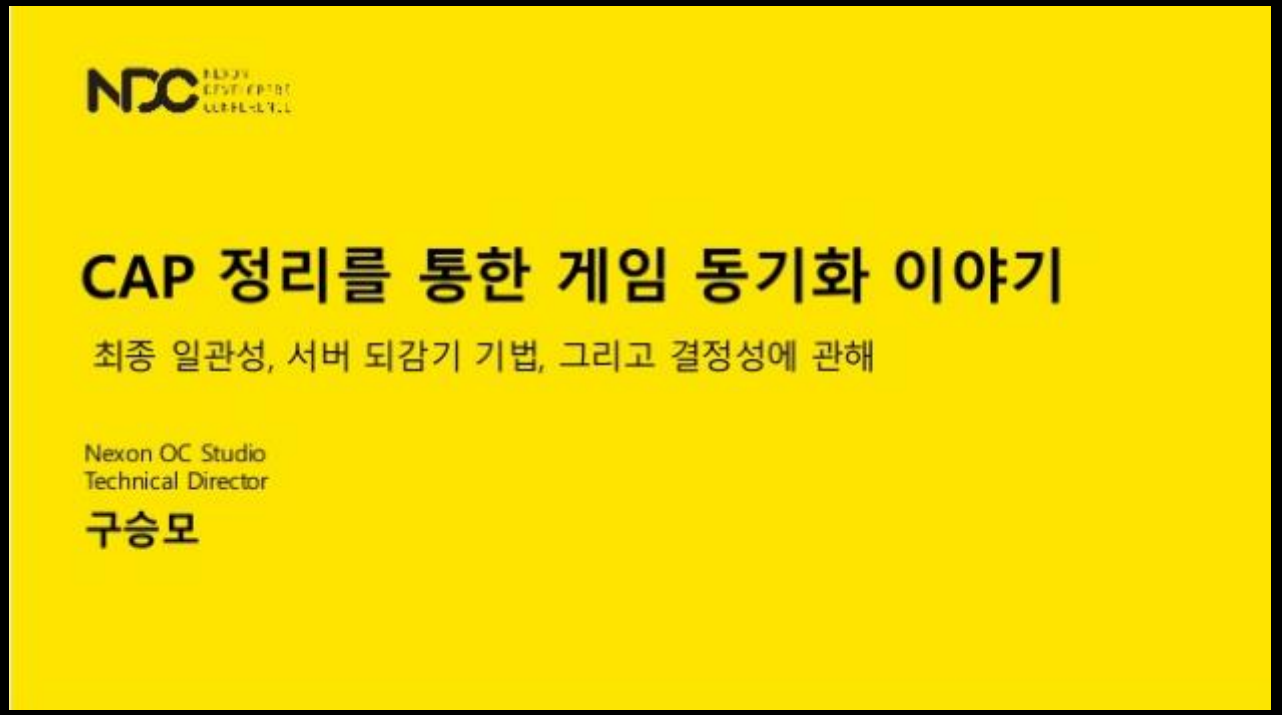
코루틴
(Coroutine)

쿠재아이
김재경

주제 선정 이유

원래 발표 하려고 했던 주제



재미있습니다. 한 번 보시길…

http://ndcreplay.nexon.com/NDC2018/sessions/NDC2018_0089.html#k%5B%5D=cap

왜 포기했나?

- 방대한 분량
- 이미 발표한 내용을 똑같이 발표 → 스터디 목적에 적절하지 못하다 생각
- 3번정도 반복해서 들어도 이해 못하는 내용이 있음 (이건 코루틴도…)
- 흥미도 코루틴 〉 CAP 이론

그럼 코루틴을 선정한 계기는?

201X년 X월 X일 발표에서 질문이 하나 나옴

그럼 코루틴을 선정한 계기는?

Q. 실행 도중 멈추고 멈춘 시점부터 다시 시작하는 함수가 있는데 뭘까요?

A. 코루틴이요

그럼 코루틴을 선정한 계기는?

실행 도중 멈추고 멈춘 시점부터 다시 시작하는 함수

???

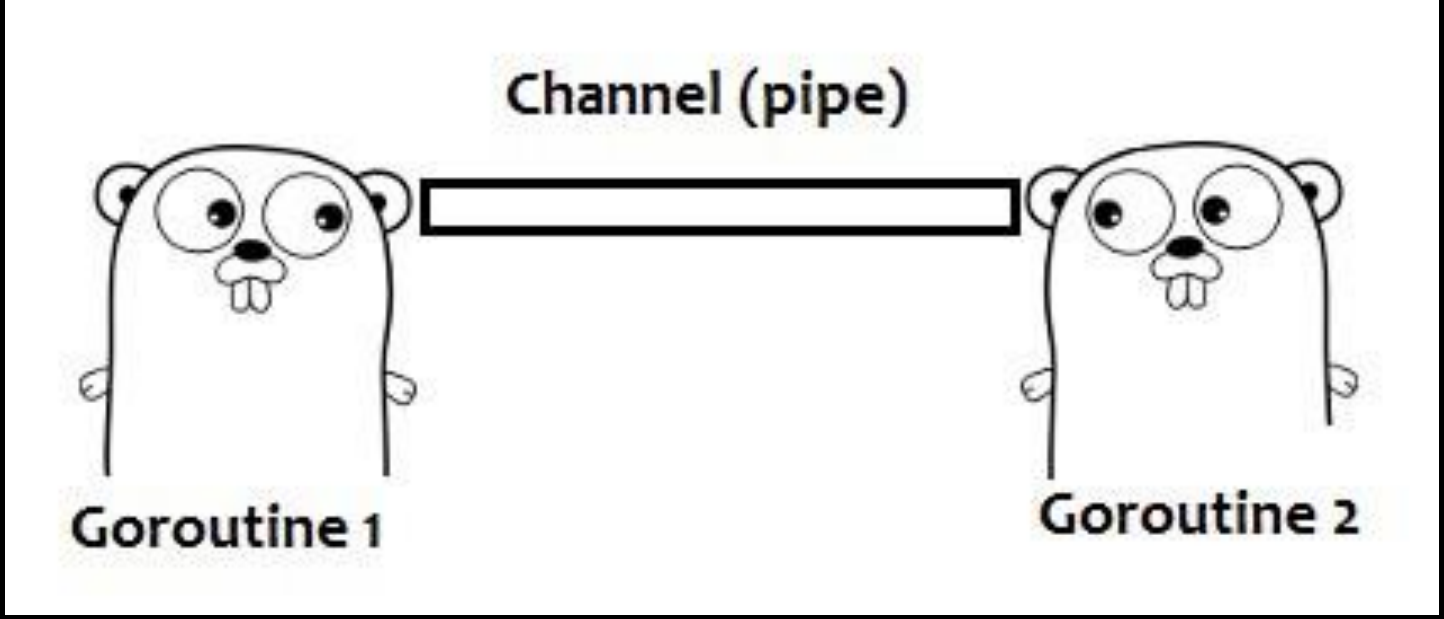
그럼 코루틴을 선정한 계기는?

질문을 들은 내 표정

그럼 코루틴을 선정한 계기는?

과거에 이해 못한 질문과 더불어

저번 발표(Go 언어)에서 나온 Goroutine이 기폭제가 되어 발표 주제로 선정



코루틴의 정의

코루틴이란 무엇인가?

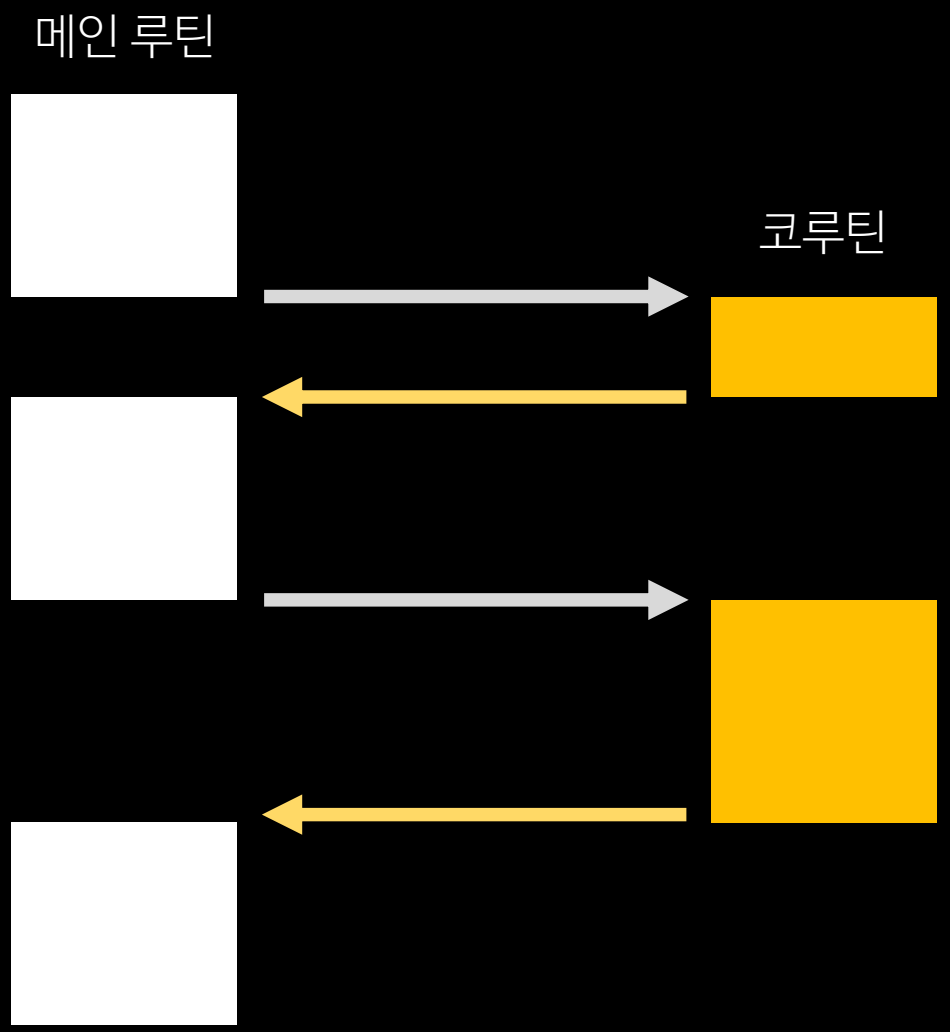
- Coroutines are computer-program components that generalize subroutines for non-preemptive multitasking, by allowing multiple entry points for suspending and resuming execution at certain locations.

Wikipedia

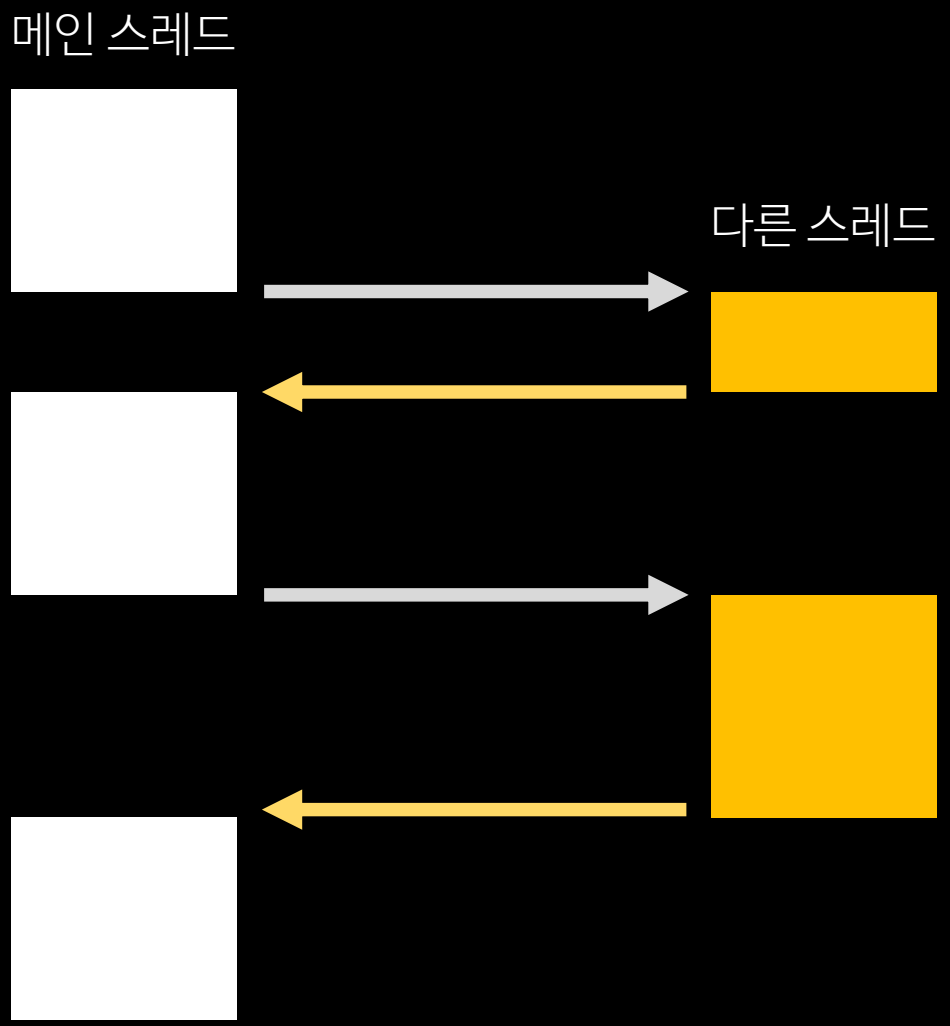
- 코루틴은 특정 위치에서 실행을 일시 중단하고 다시 시작할 수 있는 여러 진입점을 허용하여 비선점 멀티 태스킹을 위해 서브루틴을 일반화하는 컴퓨터 프로그램 구성 요소입니다.

위키피디아

코루틴이란 무엇인가?



이거 완전 스레드인데?

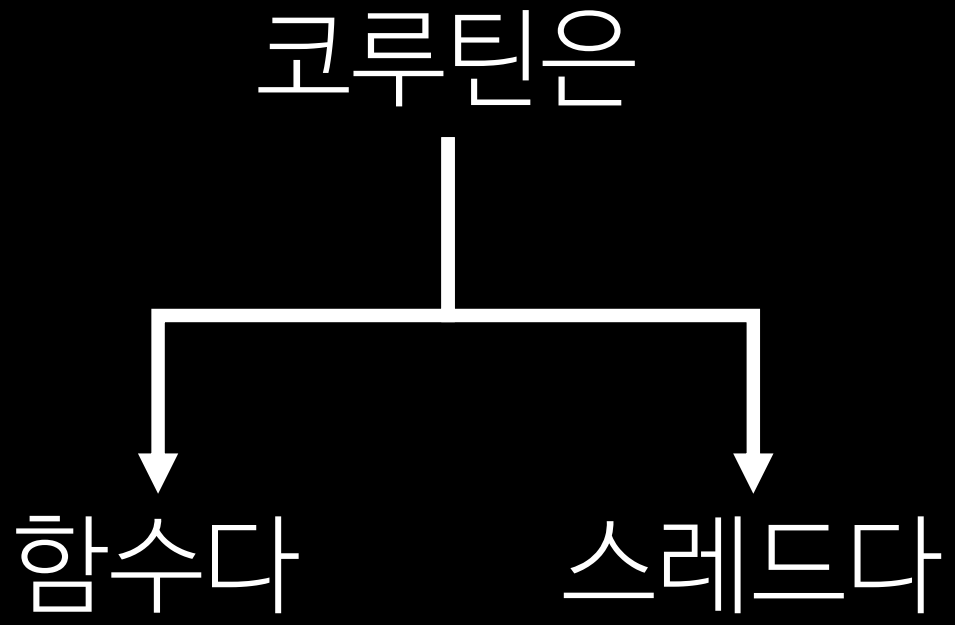


뭐가 다른거야?

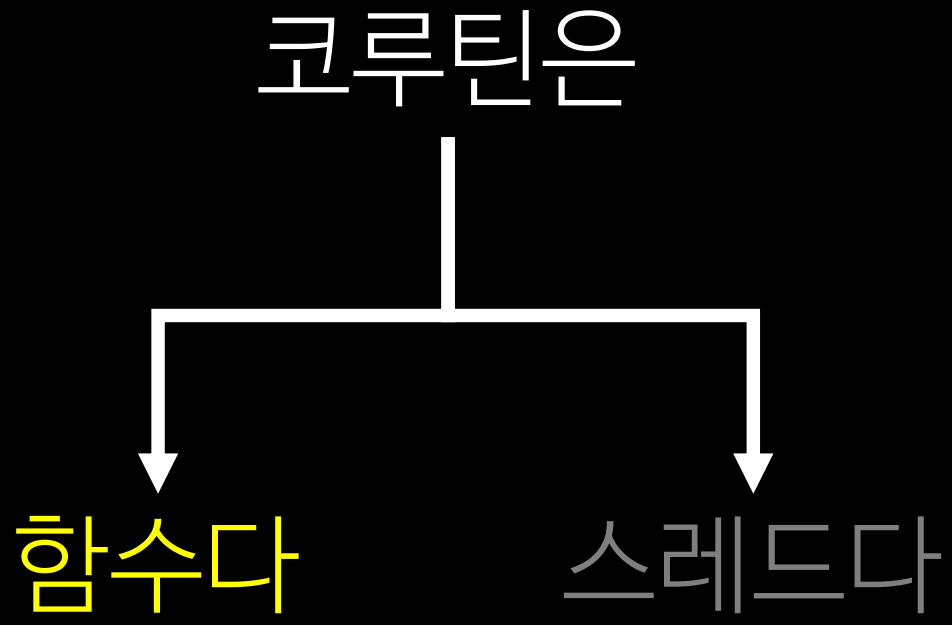
이거 완전 스레드인데?

코루틴을 잘 모른체 스레드와 비교하면 큰 착각에 빠진다

코루틴이란 무엇인가?



코루틴이란 무엇인가?



코루틴이란 무엇인가?

- Coroutines are computer-program components that generalize subroutines for non-preemptive multitasking, by allowing multiple entry points for suspending and resuming execution at certain locations.

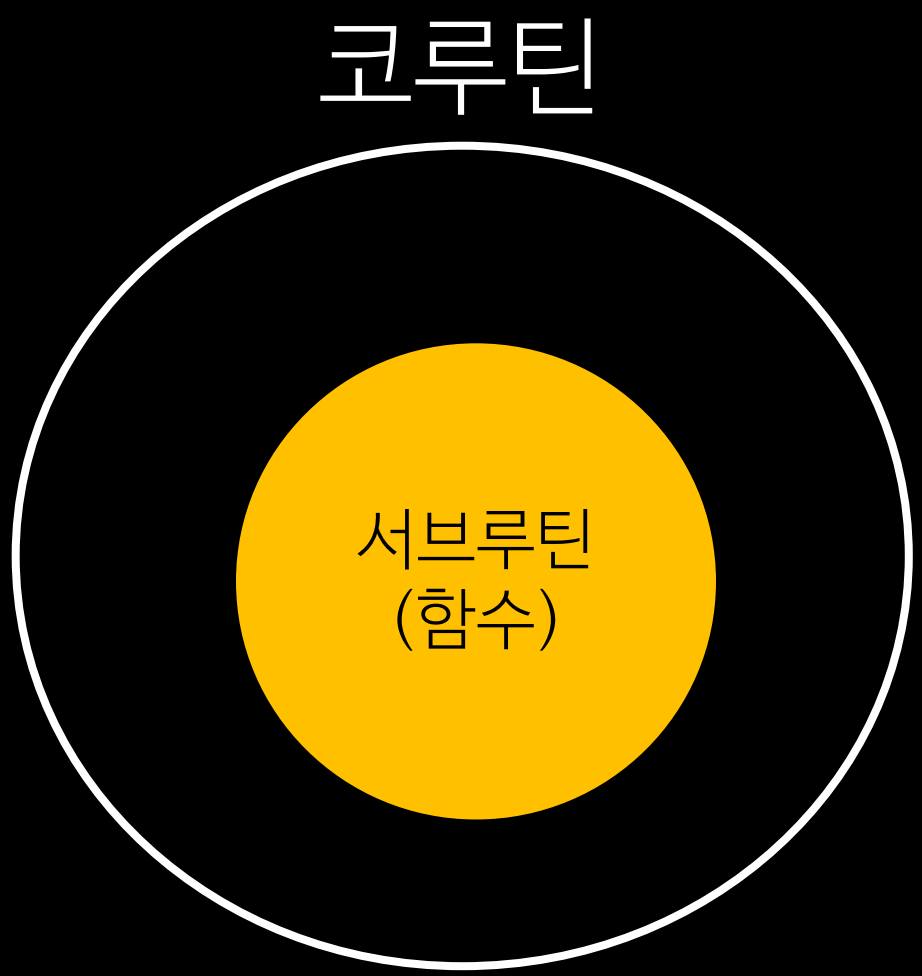
Wikipedia

- 코루틴은 특정 위치에서 실행을 일시 중단하고 다시 시작할 수 있는 여러 진입점을 허용하여 비선점 멀티 태스킹을 위해 서브루틴을 일반화하는 컴퓨터 프로그램 구성 요소입니다.

위키피디아

코루틴이란 무엇인가?

- 서브루틴은 진입점과 중단점이 1개인 코루틴이라고 정의할 수 있다



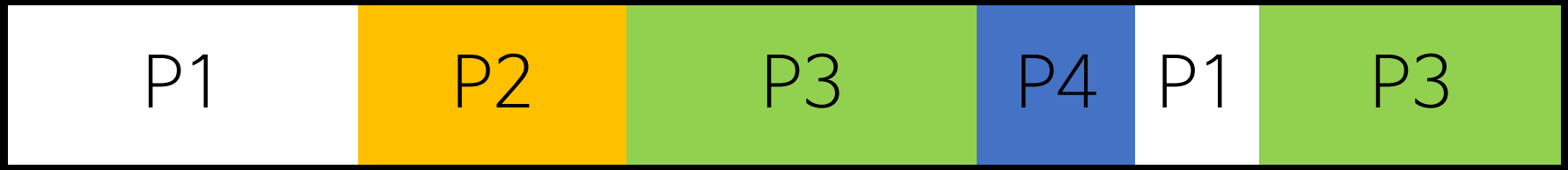
코루틴과 스레드의 차이점

코루틴과 스레드의 차이점은?

함수와 스레드의 차이점과 같다

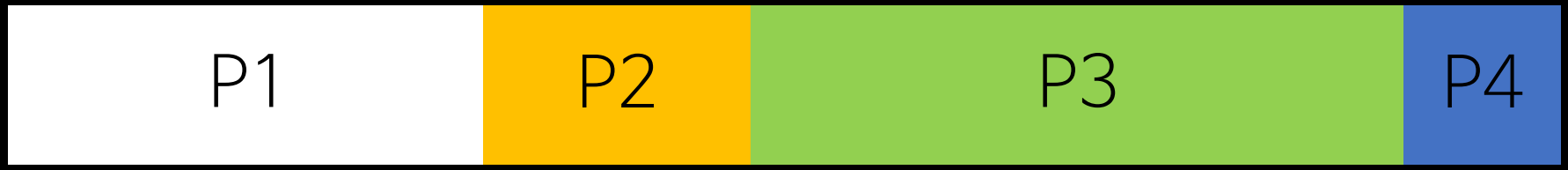
스레드는 선점형(Preemptive)이다

- 다른 스레드가 실행 도중 끼어들 수 있다

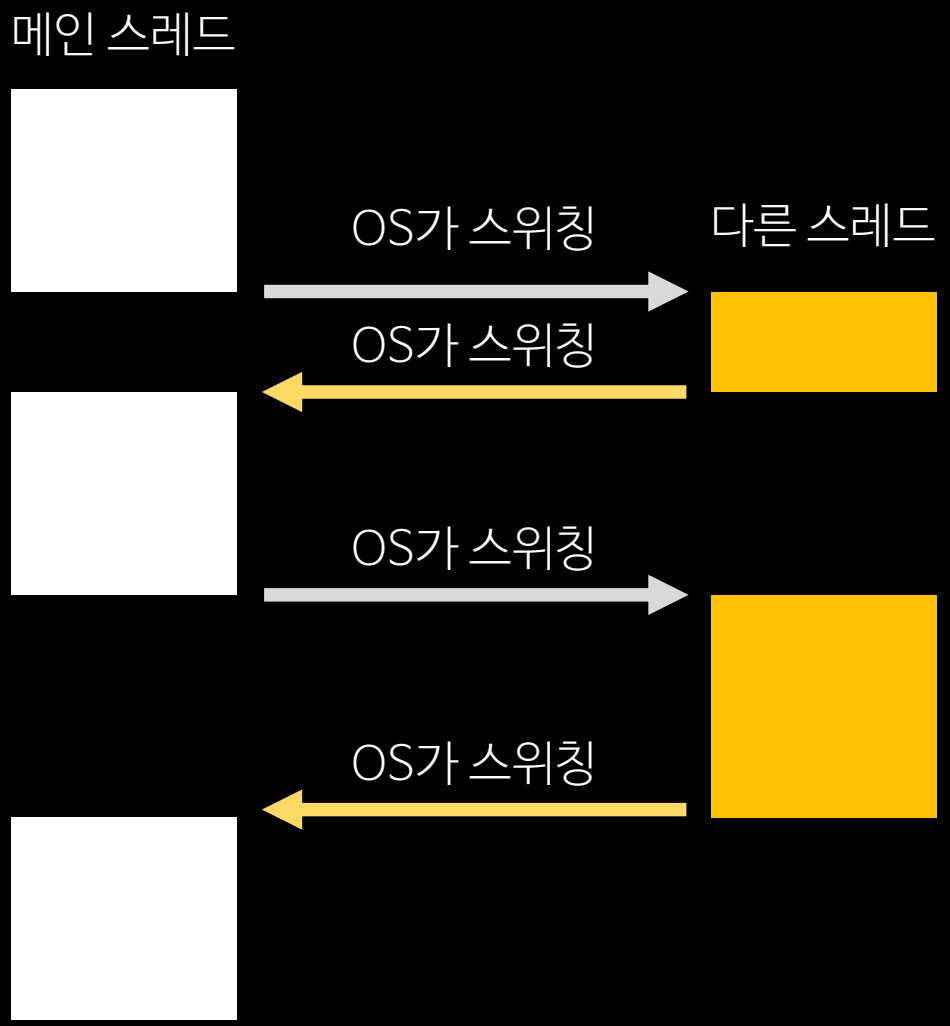


코루틴은 비선점형(Non-Preemptive)이다

- 실행 중인 코루틴이 반드시 중단 or 종료되어야 다른 코루틴을 실행할 수 있다

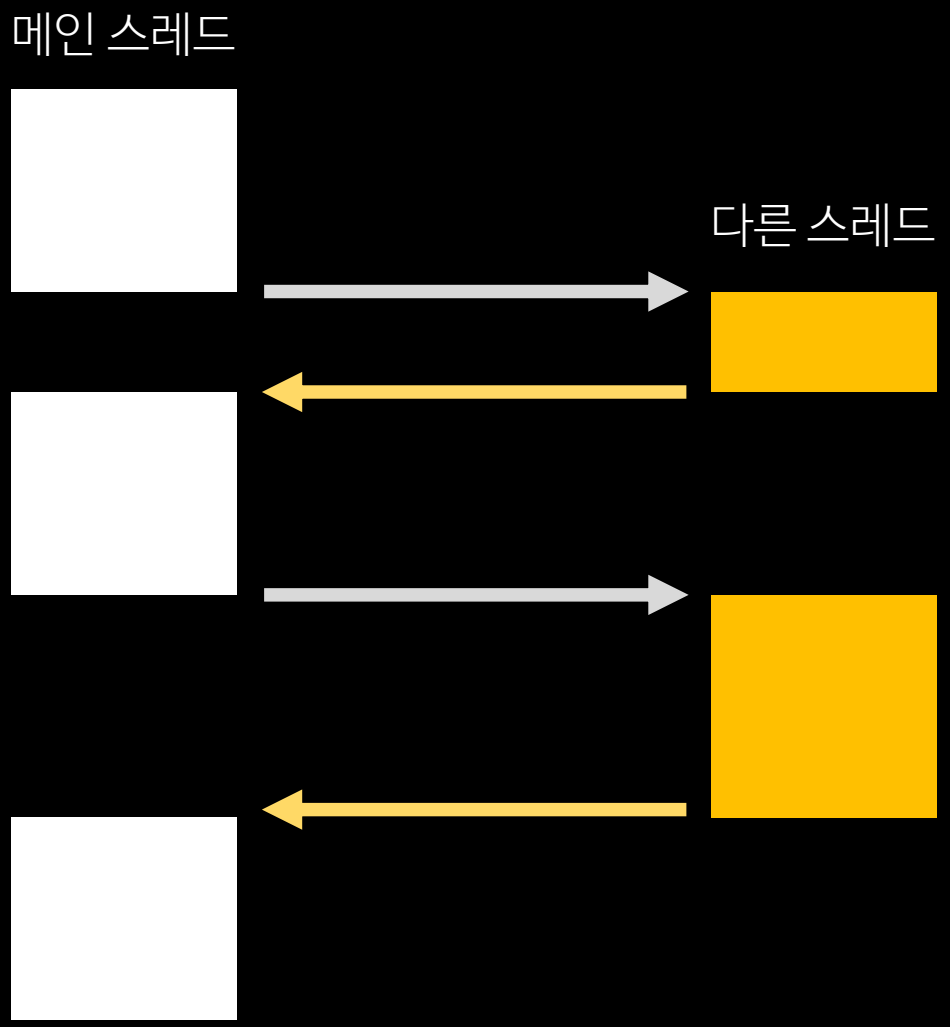


스레드는 OS가 스케줄링한다



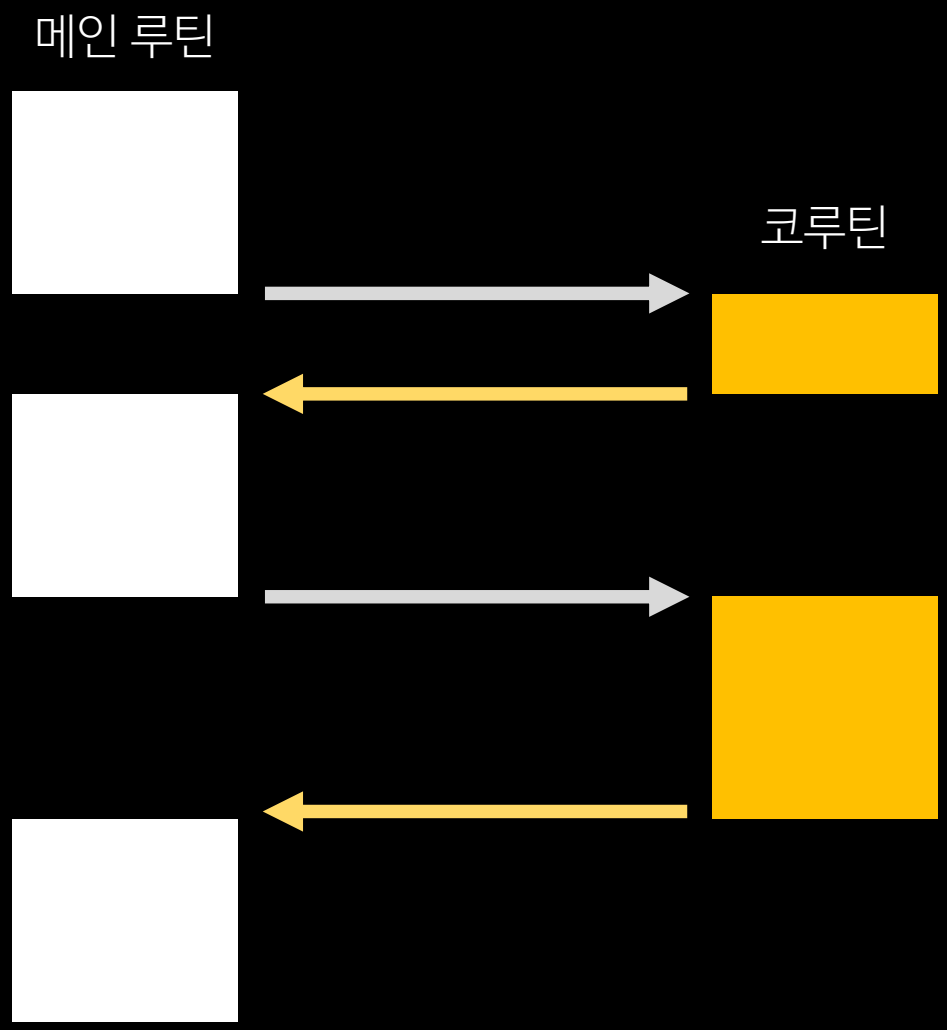
코루틴은 유저가 스케줄링한다

(멀티)스레드는 스레드가 최소 2개다



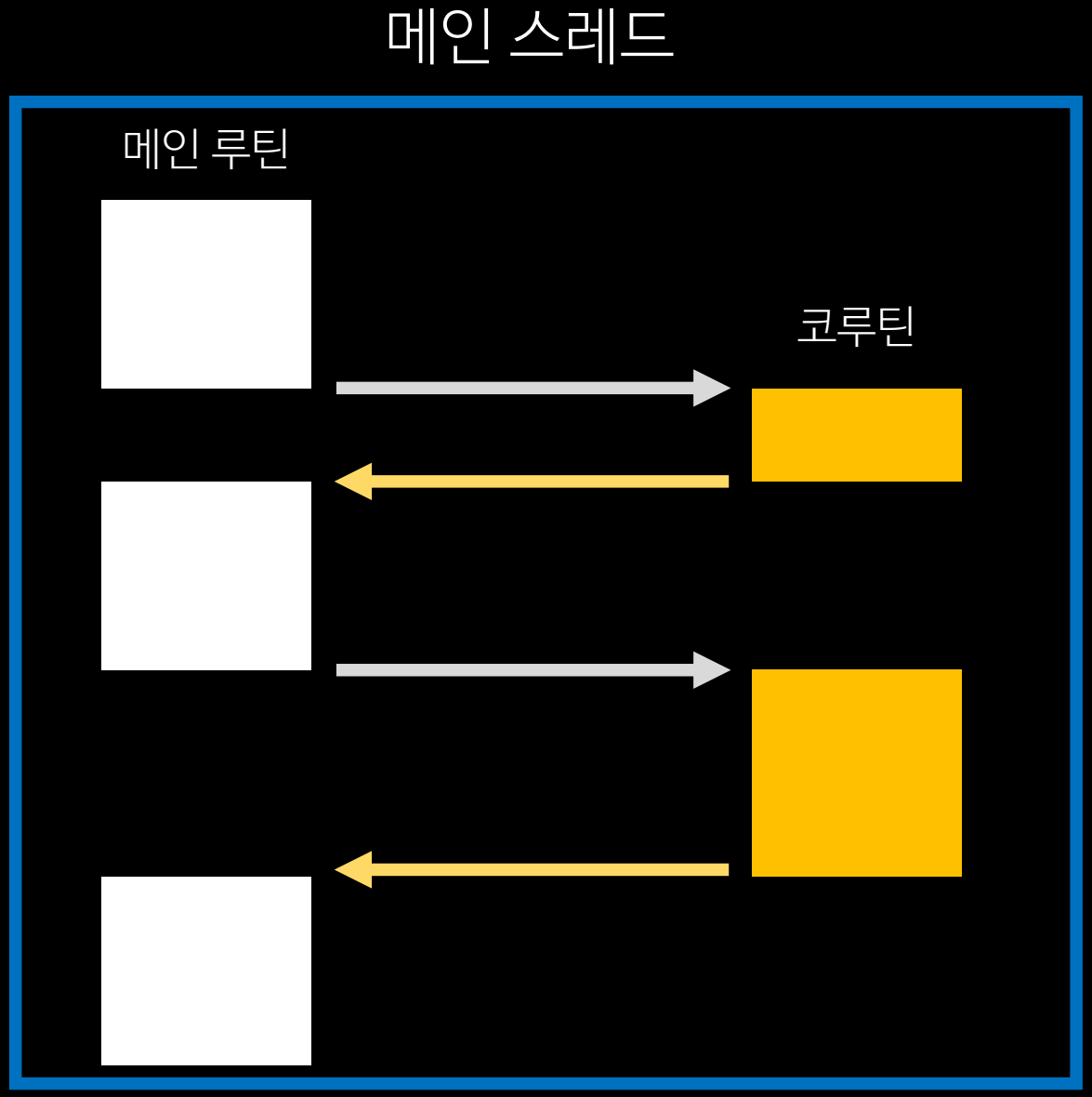
코루틴은 스레드가 1개다

- 코루틴은 함수라는 것을 명심하자



코루틴은 스레드가 1개다

- 코루틴은 함수라는 것을 명심하자
- 함수를 사용한다고 스레드가 새로 생성되는가?



이런 차이점이 있군…

이 차이점을 안다면 몇 가지를 더 유추할 수 있다

스레드를 왜 사용하는가?

- 여러 이유가 있겠지만 대표적으로 I/O Blocking 때문에

스레드를 왜 사용하는가?

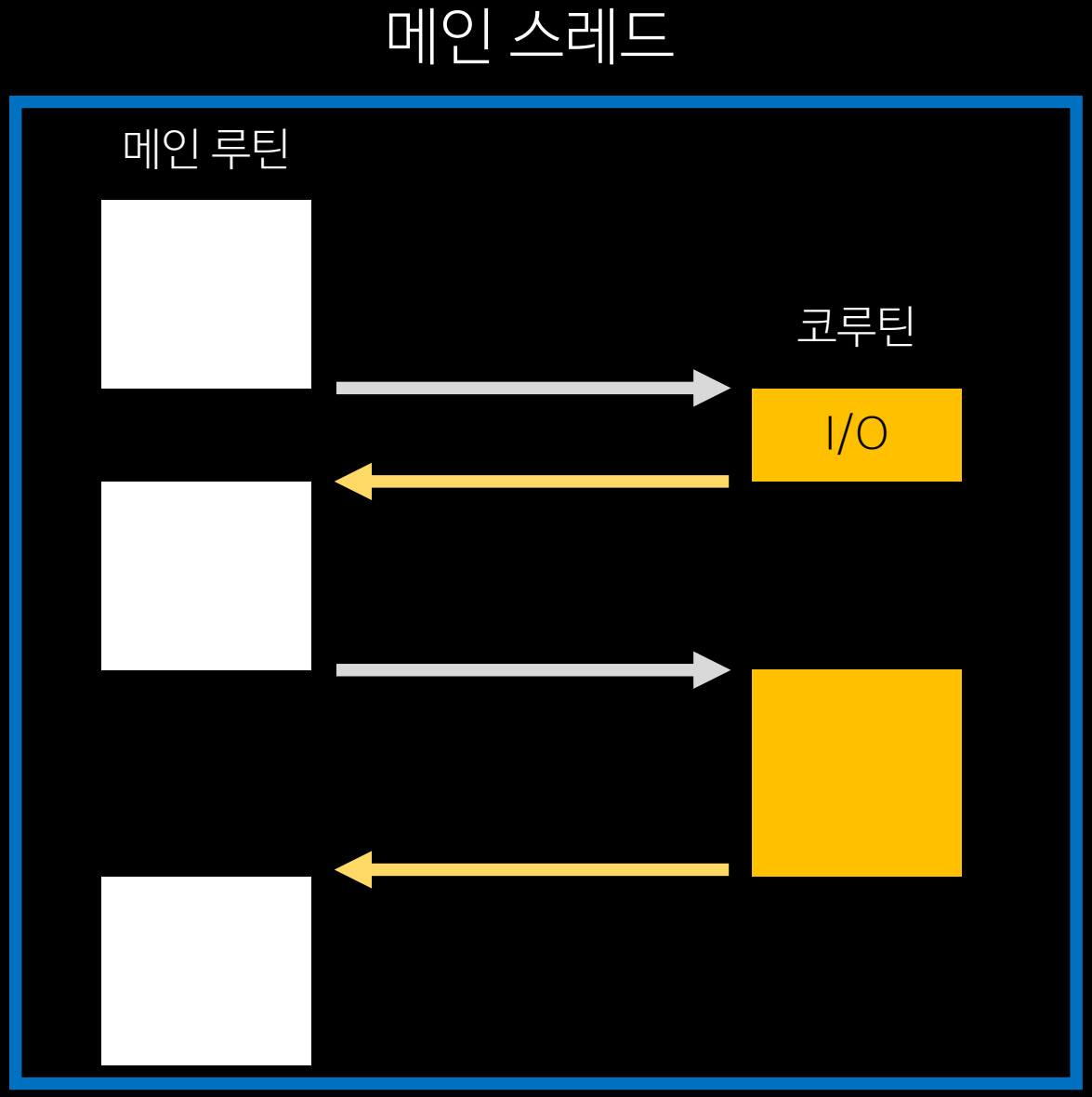
- 여러 이유가 있겠지만 대표적으로 I/O Blocking 때문에
- 그럼 문제점은?

스레드를 왜 사용하는가?

- 여러 이유가 있겠지만 대표적으로 I/O Blocking 때문에
- 그럼 문제점은?
- 데드락, 기아현상, 공유변수 기타 등등

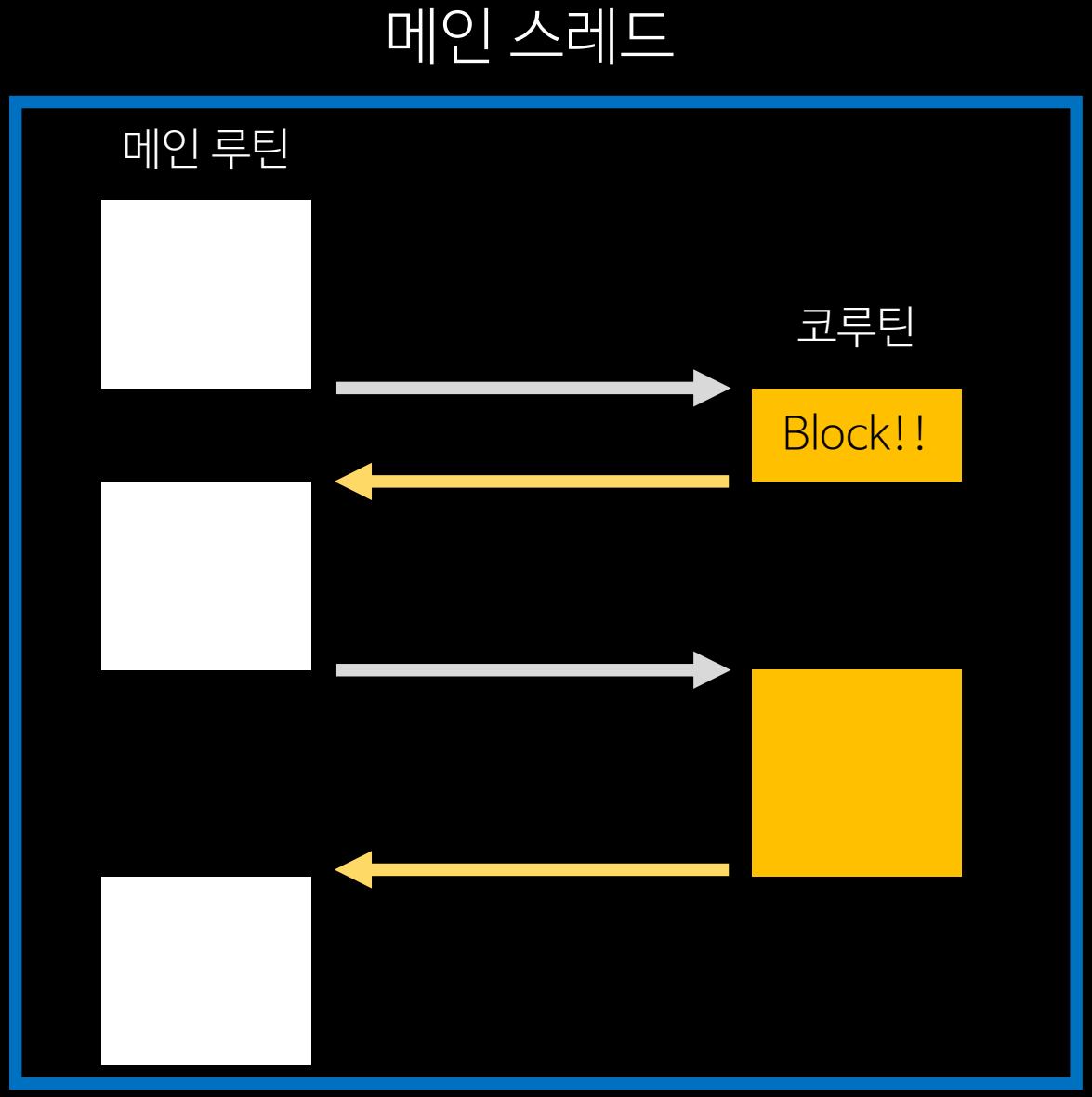
스레드 대신 코루틴을 사용하면?

- 코루틴에서 I/O 작업을 하면 어떻게 될까?



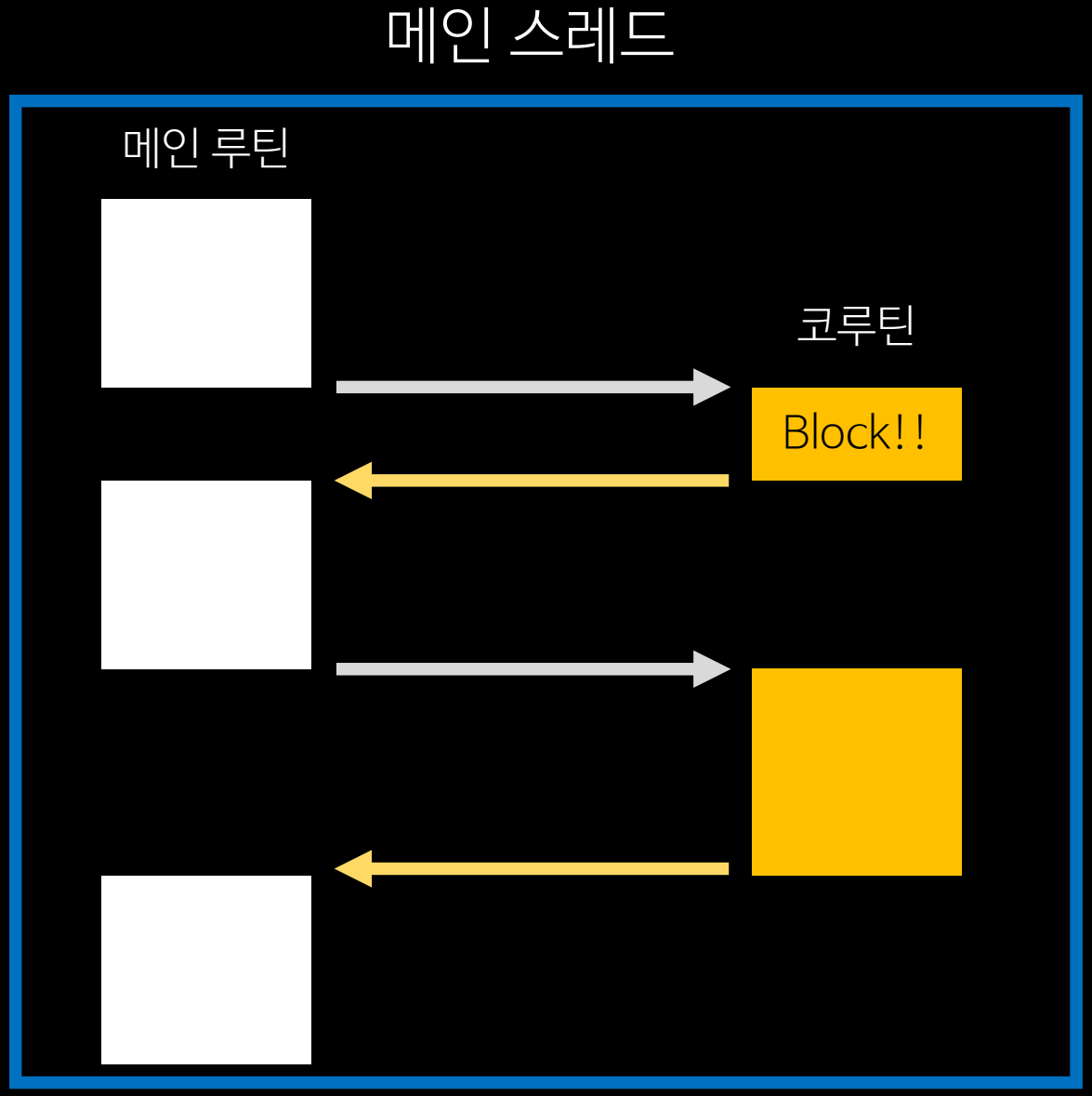
스레드 대신 코루틴을 사용하면?

- 코루틴에서 I/O 작업을 하면 어떻게 될까?
- 멀티 스레드 효과를 얻을 수 없다
- 어떻게 해야되나?



스레드 대신 코루틴을 사용하면?

- 코루틴에서 I/O 작업을 하면 어떻게 될까?
- 멀티 스레드 효과를 얻을 수 없다
- 어떻게 해야되나?
- 비동기 I/O를 사용해야 한다

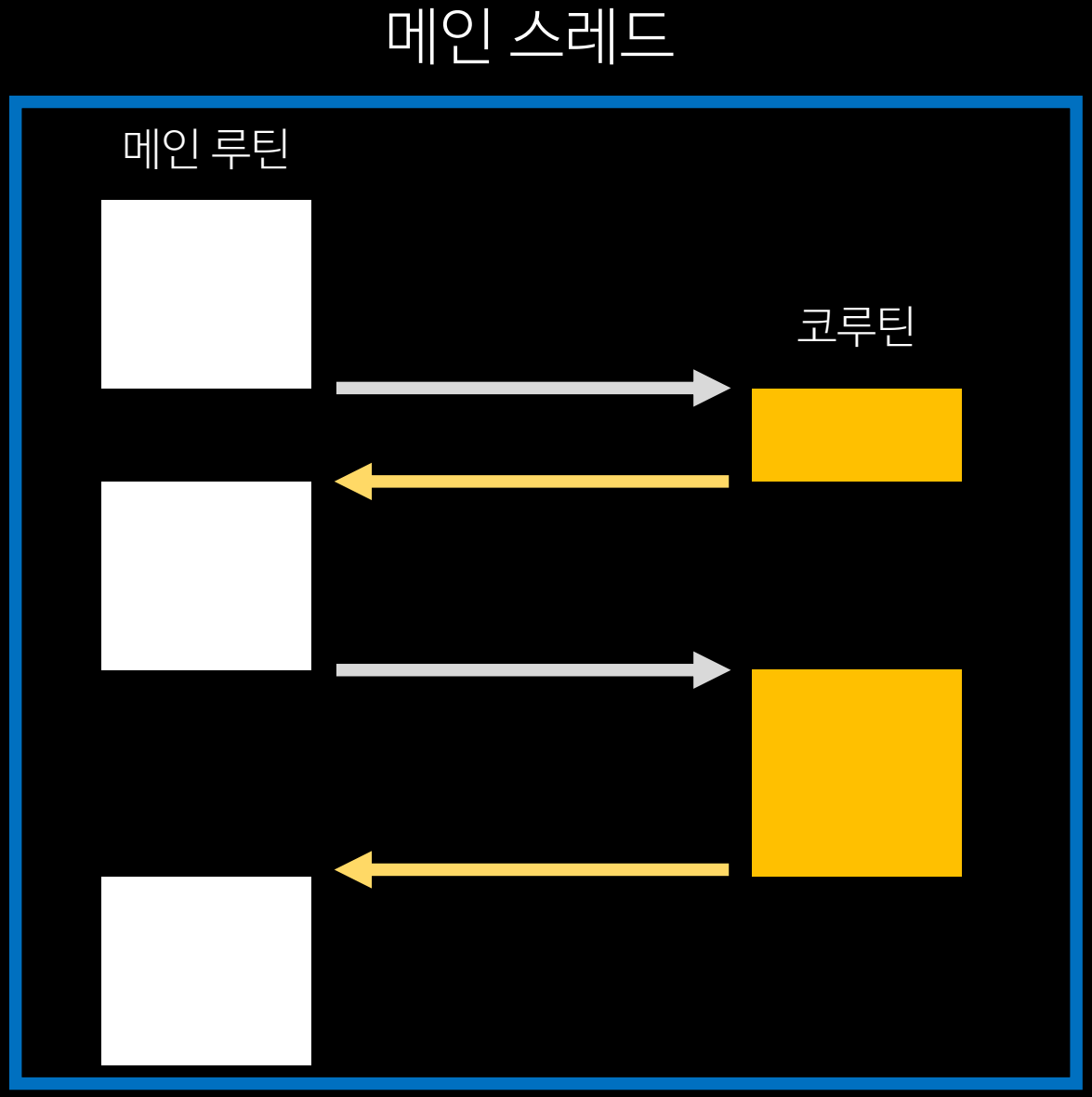


스레드 대신 코루틴을 사용하면?

- 스레드는 공유변수 문제로 동기화를 해야 한다
- 코루틴은?

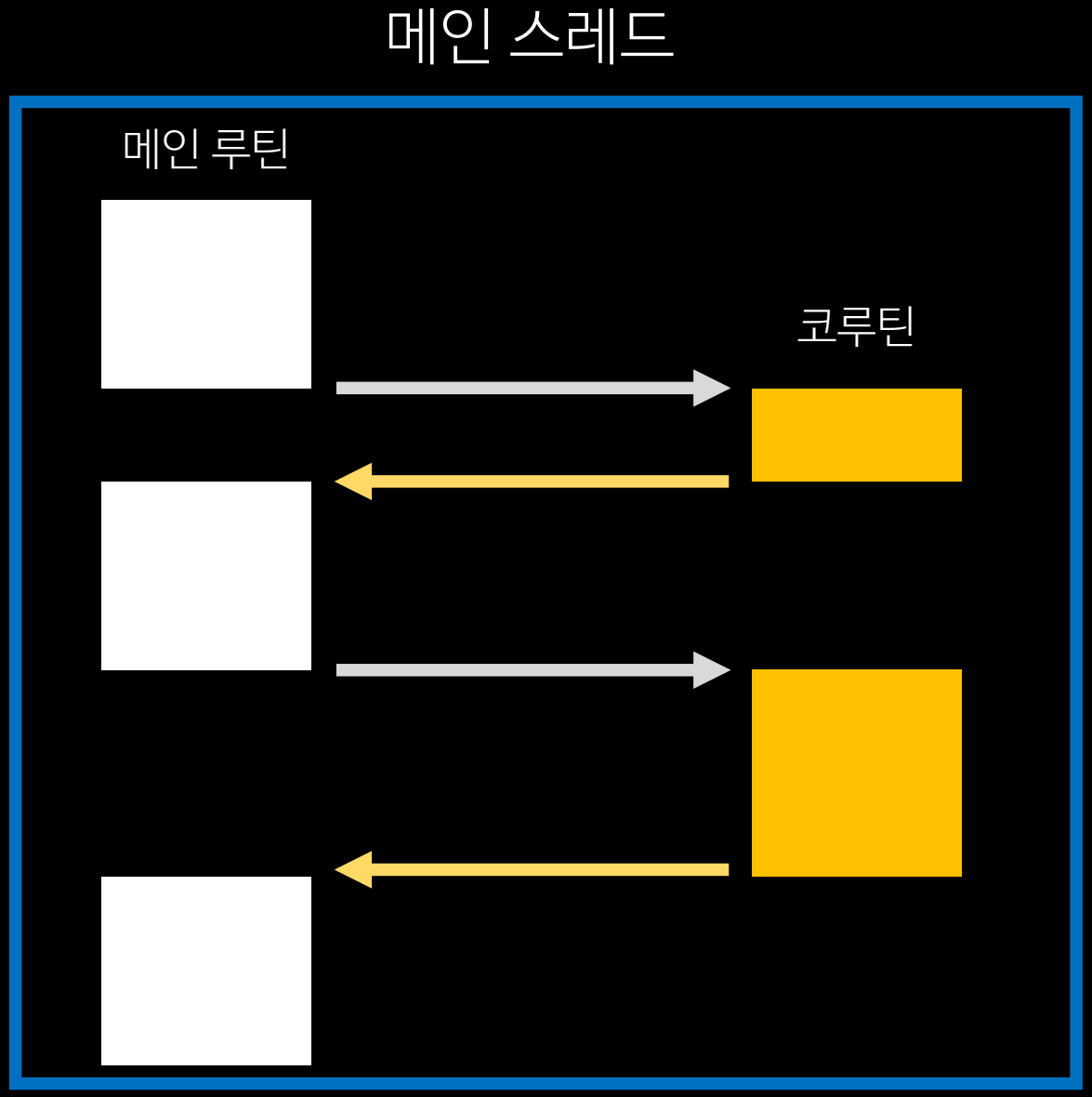
스레드 대신 코루틴을 사용하면?

- 코루틴은 동기화가 필요 없다



스레드 대신 코루틴을 사용하면?

- 코루틴은 동기화가 필요 없다
- 여러 함수에서 전역변수에 접근한다고 동기화를 하는가?



(유니티에서)코루틴을 왜 사용하는가?

- 코루틴이 구현된 언어나 라이브러리가 많지만 게임 분야에서는 유니티에서만 사용되는 것 같다 (구글 검색하니 유니티 글밖에 없음)

(유니티에서)코루틴을 왜 사용하는가?

- 코루틴이 구현된 언어나 라이브러리가 많지만 게임 분야에서는 유니티에서만 사용되는 것 같다 (구글 검색하니 유니티 글밖에 없음)
- 안써봐서 모름

(유니티에서)코루틴을 왜 사용하는가?

- 코루틴이 구현된 언어나 라이브러리가 많지만 게임 분야에서는 유니티에서만 사용되는 것 같다 (구글 검색하니 유니티 글밖에 없음)
- 안써봐서 모름
- 몇 가지 차이점이 있지만 코루틴의 호출 방식과 로직 분리는 멀티 스레드를 흉내낼 수 있다

(유니티에서)코루틴을 왜 사용하는가?

- 코루틴이 구현된 언어나 라이브러리가 많지만 게임 분야에서는 유니티에서만 사용되는 것 같다 (구글 검색하니 유니티 글밖에 없음)
- 안써봐서 모름
- 몇 가지 차이점이 있지만 코루틴의 호출 방식과 로직 분리는 멀티 스레드를 흉내낼 수 있다
- 어려운 멀티 스레드를 사용하느니 코루틴을 쓰라는게 아닐까? (뇌피셜)

코루틴의 구현

어떻게 구현하는가?

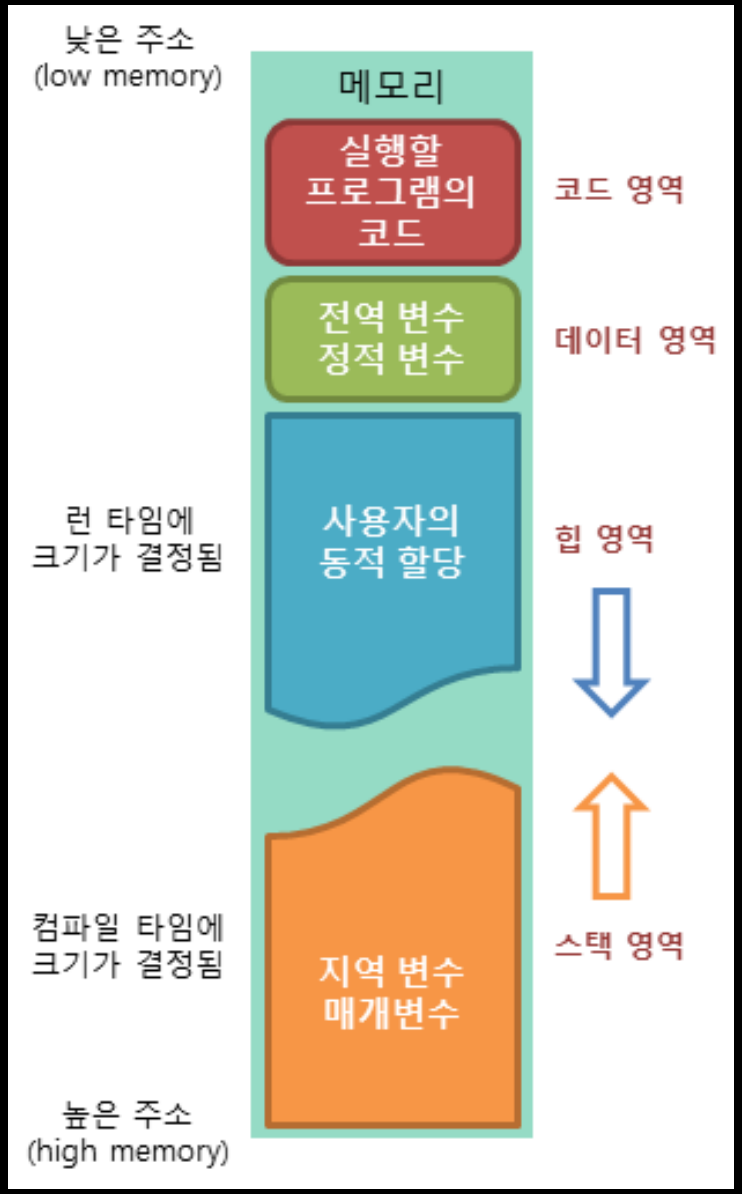
- 프로그래밍에서 중간에 그만두고 다시 시작하는 것은 어딘가에 정보를 저장했다가 불러오는 것을 의미한다
- 스레드를 생각하면 쉽다 (Context Switch)

어떤 정보를 저장하는가?

- 함수가 실행되는데 필요한 정보
- 지역변수, 레지스터 값 등

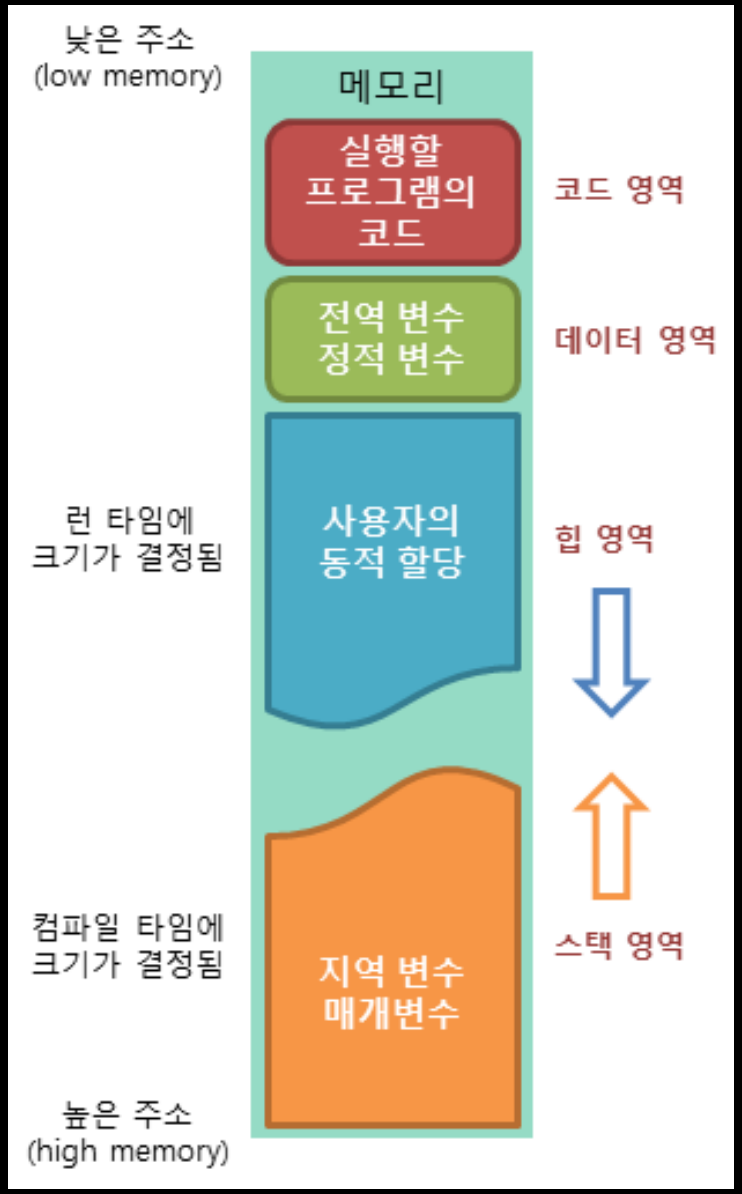
어디에 저장하는가?

- 메모리 어디?



어디에 저장하는가?

- 메모리 어디?
- Heap



흠… 생각을 해보니

- 레지스터까지 접근한다는 건 어셈블리 레벨까지 내려간다는 건데… 그렇게까지 할까?

흠… 생각을 해보니

- 레지스터까지 접근한다는 건 어셈블리 레벨까지 내려간다는 건데… 그렇게까지 할까?
- 찾아보니 진짜 그렇게 함;;

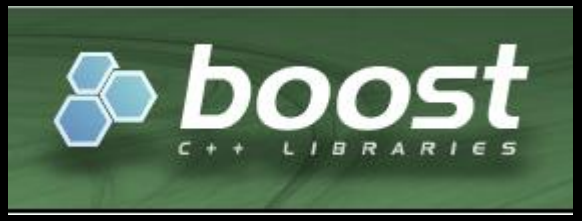
C++ 표준에서는 구현되어 있을까?

- 아직 미구현
- C++20에서 추가 예정

C++20에서 추가된다고?

그럼 이 라이브러리에는 구현되어 있지 않을까?

Boost에는 구현되어 있다

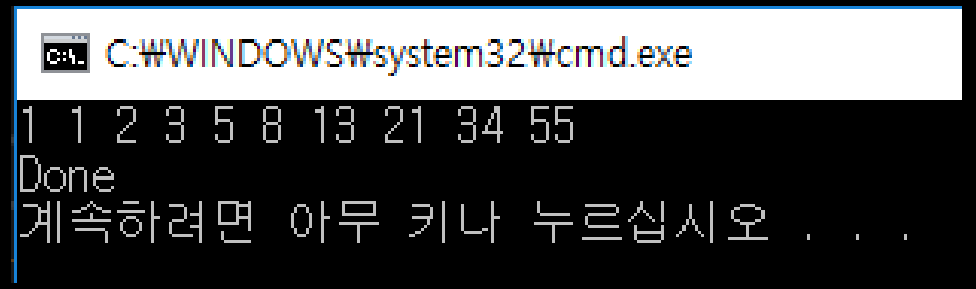


- 최신 버전(1.68.0) 기준
- Coroutine과 Coroutine2가 있다
- Coroutine은 deprecated 상태

Boost Coroutine을 이용한 피보나치 수열

```cpp
int main() {
    boost::coroutines2::coroutine< int >::pull_type source(
        [](boost::coroutines2::coroutine< int >::push_type & sink) {
        int first = 1, second = 1;
        sink(first);
        sink(second);
        for (int i = 0; i < 8; ++i) {
            int third = first + second;
            first = second;
            second = third;
            sink(third);
        }
    });
    for (auto i : source) {
        std::cout << i << " ";
    }

    std::cout << "\nDone" << std::endl;
    return EXIT_SUCCESS;
}
```

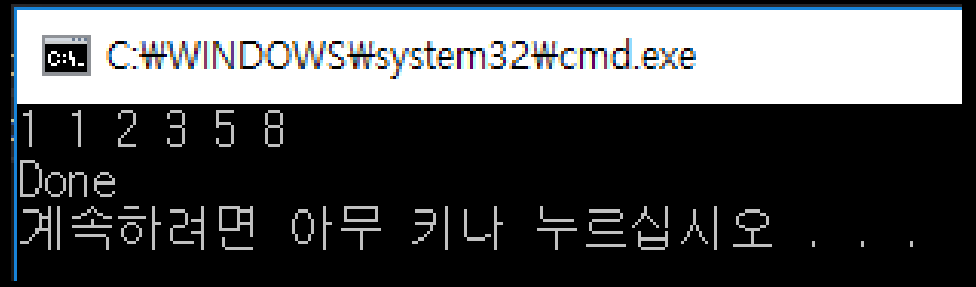


Boost Coroutine을 이용한 피보나치 수열

```cpp
int main() {
    boost::coroutines2::coroutine< int >::pull_type source(
        [](boost::coroutines2::coroutine< int >::push_type & sink) {
        int first = 1, second = 1;
        sink(first);
        sink(second);
        for (int i = 0; i < 8; ++i) {
            int third = first + second;
            first = second;
            second = third;
            sink(third);
        }
    });

    std::cout << source.get() << " ";
    source();
    std::cout << source.get() << " ";
    source();
    std::cout << source.get() << " ";
    source();
    std::cout << source.get() << " ";
    source();
    std::cout << source.get() << " ";
    source();
    std::cout << source.get() << " ";

    std::cout << "\nDone" << std::endl;
    return EXIT_SUCCESS;
}
```



coroutine -> pull_coroutine

```cpp
template< typename T >
struct coroutine {
    using pull_type = detail::pull_coroutine< T >;
    using push_type = detail::push_coroutine< T >;
};
```

```cpp
class pull_coroutine {
private:
    template< typename X >
    friend class push_coroutine;

    struct control_block;

    control_block   *   cb_;

    // ...
};
```

pull_coroutine -> control_block

```
class pull_coroutine {
private:
    template< typename X >
    friend class push_coroutine;

    struct control_block;

    control_block   *   cb_;

    // ...
};
```

```
template< typename T >
struct pull_coroutine< T >::control_block {
    boost::context::fiber                                   c;
    typename push_coroutine< T >::control_block         *   other;
    state_t                                                 state;
    std::exception_ptr                                      except;
    bool                                                    bvalid;
    typename std::aligned_storage< sizeof(T), alignof(T) >::type  storage;

    // ...
};
```

control_block -> fiber

```
template< typename T >
struct pull_coroutine< T >::control_block {
    boost::context::fiber                                   c;
    typename push_coroutine< T >::control_block         *   other;
    state_t                                                 state;
    std::exception_ptr                                      except;
    bool                                                    bvalid;
    typename std::aligned_storage< sizeof(T), alignof(T) >::type  storage;

    // ...
};
```

```
class fiber {
private:
    // ...

    detail::fcontext_t  fctx_{ nullptr };

    // ...
};
```

Fiber -> fcontext_t

```
class fiber {
private:
    // ...

    detail::fcontext_t  fctx_{ nullptr };

    // ...
};
```

```
namespace detail {

typedef void*   fcontext_t;
```

유니티의 코루틴은?

IEnumerator를 이용한 변종에 가깝다

자세한건 구글 검색

Q/A

참고 자료

- Wikipedia - Coroutine
  https://en.wikipedia.org/wiki/Coroutine

- Kotlin의 코루틴은 어떻게 동작하는가
  https://www.slideshare.net/cwdoh/hey-kotlin-how-coroutine-works

- Blocking-NonBlocking-Synchronous-Asynchronous
  https://homoefficio.github.io/2017/02/19/Blocking-NonBlocking-Synchronous-Asynchronous/

- 코루틴(Coroutine) 이해하기
  https://kwangyulseo.com/2015/05/15/%EC%BD%94%EB%A3%A8%ED%8B%B4coroutine-%EC%9D%B4%ED%95%B4%ED%95%98%EA%B8%B0/

참고 자료

- 코루틴 소개
  https://medium.com/@jooyunghan/%EC%BD%94%EB%A3%A8%ED%8B%B4-%EC%86%8C%EA%B0%9C-504cecc89407

- 코루틴을 구분해보자
  https://medium.com/@jooyunghan/%EC%BD%94%EB%A3%A8%ED%8B%B4%EC%9D%84-%EA%B5%AC%EB%B6%84%ED%95%B4%EB%B3%B4%EC%9E%90-98428c491ace

- 코루틴과 파이버
  https://medium.com/@jooyunghan/%EC%BD%94%EB%A3%A8%ED%8B%B4%EA%B3%BC-%ED%8C%8C%EC%9D%B4%EB%B2%84-9e93c12bce30

- Knock! Knock! 코루틴
  https://medium.com/@jooyunghan/knock-knock-%EC%BD%94%EB%A3%A8%ED%8B%B4-c4ccc17a5d66

참고 자료

- Stackful/Stackless 코루틴
  https://medium.com/@jooyunghan/stackful-stackless-%EC%BD%94%EB%A3%A8%ED%8B%B4-4da83b8dd03e

- Knock! Knock! 코루틴 #2
  https://medium.com/@jooyunghan/knock-knock-%EC%BD%94%EB%A3%A8%ED%8B%B4-2-e5d26678e021

- Fibers (Microsoft Docs)
  https://docs.microsoft.com/en-us/windows/desktop/procthread/fibers

- Using Fibers
  https://docs.microsoft.com/en-us/windows/desktop/procthread/using-fibers