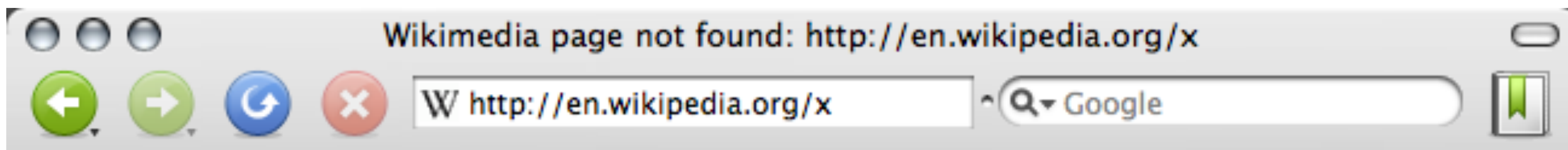


세번째 스터디

-오주년

HTTP



404 error: File not found

The [URL](#) you requested was not found.

Did you mean to type <http://en.wikipedia.org/wiki/x>? You will be automatically redirected there in five seconds.

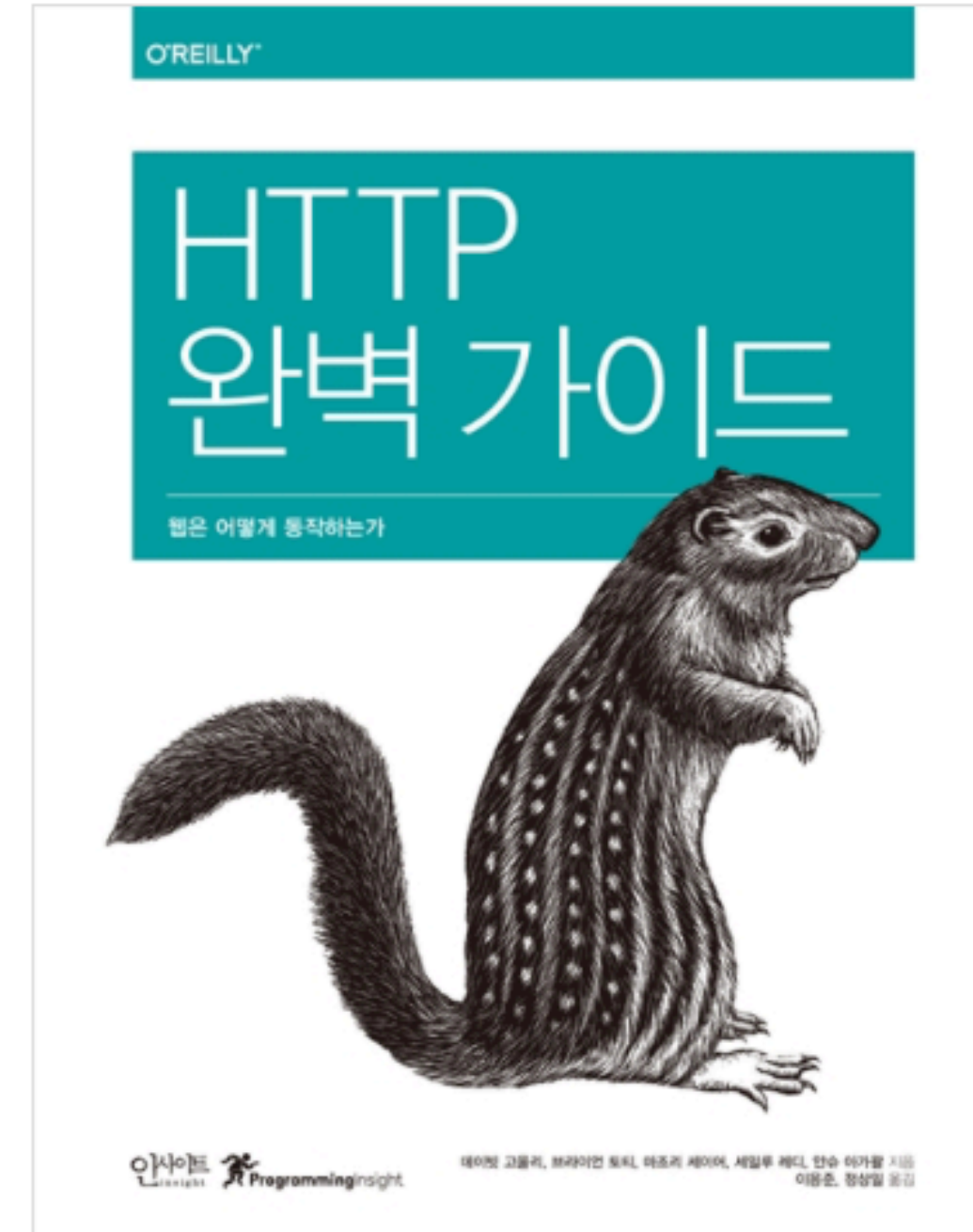
Maybe you would like to look at:

- [The main page](#)
- [The list of Wikimedia downloads](#)

A project of the [Wikimedia foundation](#).



손 더 게스트
드라마 <손 the guest>
원작 소설



정가	39,000원
판매가	35,100원 (10%, 3,900원 할인)
마일리지	1,950원(5%) + 멤버십(3~1%) + 5만원이상 구매시 2,000원 ?
세액절감액	1,580원 (도서구입비 소득공제 대상 및 조건 충족 시) ?
배송료	무료 ?
수령예상일	지금 택배로 주문하면 오늘(17~21시) 수령 ∨ 최근 1주 88.1% (중구 중립등 기준) 지역변경

프로그래밍 개발/방법론 주간 18위, 컴퓨터/모바일 top100 9주 1

Sales Point : 2,846 ?

★★★★★ 8.0 100자평(0) 리뷰(2) [이 책 어때요?](#)

[카드간편결제 할인](#) > [무이자 할부](#) >

스프링 분철

스프링 분철 서비스 대상도서입니다. 자세히 보기 >

수량

[장바구니 담기](#) [바로구매](#) [선물하기](#) [보관함 +](#)

[전자책 출간알림 신청](#) > [중고 등록알림 신청](#) > [중고로 팔기](#) ∨



김미경의 리부트

바뀌는 판에 투자해야 돈이 모인다!

AD

찬호께이 <13.67> 리커버

특별판 단독 판매!



알라디너TV,
구독하면
적립금 1천원

기본정보

756쪽 183*240mm 1436g ISBN : 9788966261208

주제 분류 [신간알림 신청](#)

◦ 국내도서 > 컴퓨터/모바일 > 프로그래밍 개발/방법론 > 웹 서비스/웹 프로그래밍

어떻게 해야 이해하기 쉬울까...

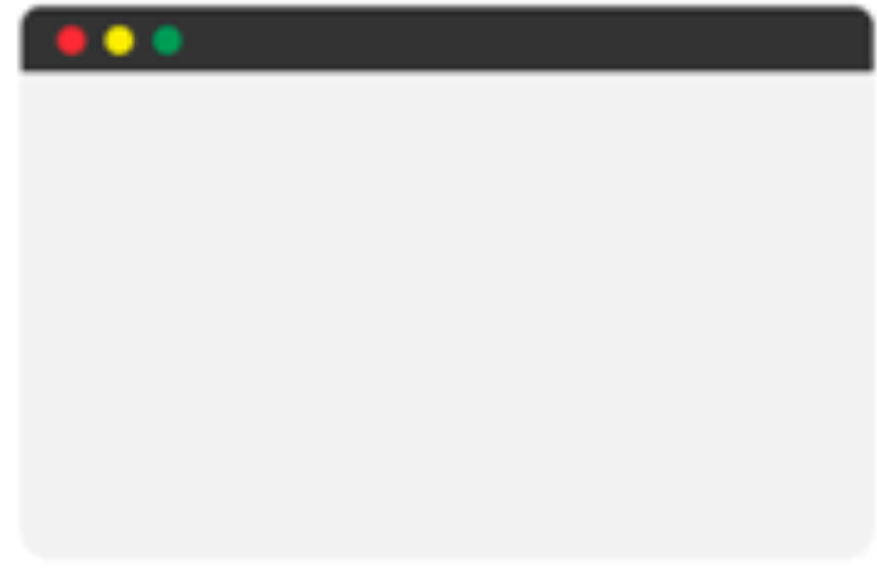
우선 HTTP는 하나의 약속이라는걸 알려주세요🙏

HTTP - HyperText Transfer Protocol

서버와 클라이언트 사이에서 데이터를 주고 받기 위한 하나의 약속 !

좀 더 알아보자면

- ISO 모형의 응용계층(L7), TCP/IP의 응용계층(L4)이다 !
- 무상태 프로토콜이다 ! (서버가 통신이 끝나면 통신에 대한 정보를 지워버림) - 이때문에 헤더에 쿠키, 세션, 토큰을 올려 상태를 판단 !



CLIENT

REQUEST →

← RESPONSE



SERVER

HTTP REQUEST

Request Line

GET /docs/index.html HTTP/1.1

Request
Header

Host: www.test.com

Accept: image/gif, image/jpeg, */*

Accept-Language: en-us

Accept-Encoding: gzip, deflate

User-Agent: Mozilla/4.0

Blank

Content-Length: 35

Request
Message
Header

request body

...

...

Request
Message
Body

HTTP REQUEST

Request Line

METHOD URL VERSION CR LF

Request
Header

HEADER FIELD NAME : VALUE CR LF

•
•
•

Request
Message
Header

Blank

HEADER FIELD NAME : VALUE CR LF

CR LF

Request
Message
Body

ENTITY BODY

HTTP RESPONSE

Status Line

HTTP/1.1 200 OK

Response
Header

Date: Wed, 29 Apr 2020 04:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Wed, 29 Apr 2020 07:16:26 GMT
ETag: "10000000565a5-2c-3e94b66c2e680"
Accept-Ranges: bytes
Content-Length: 44
Connection: close
Content-Type: text/html
X-Pad: avoid browser bug

Blank

<html>
<body>
 <h1> It works! </h1>
</body>
</html>

Response
Message
Header

Response
Message
Body

HTTP RESPONSE

Status Line

VERSION STATUS CODE PHRASE CR LF

Response
Header

HEADER FIELD NAME : VALUE CR LF

•
•
•

Blank

HEADER FIELD NAME : VALUE CR LF

CR LF

Response
Message
Header

Response
Message
Body

ENTITY BODY

HTTP METHOD with CRUD

CRUD	ACTION	HTTP METHOD	SQL	요청에 Body가 있음	응답에 Body가 있음
CREATE	생성	POST	INSERT	O	O
READ	조회	GET	SELECT	X	O
UPDATE	수정	PUT	UPDATE	O	O
DELETE	삭제	DELETE	DELETE	X	O

HTTP status code

정보 응답

100 Continue

이 임시적인 응답은 지금까지의 상태가 괜찮으며 클라이언트가 계속해서 요청을 하거나 이미 요청을 완료한 경우에는 무시해도 되는 것을 알려줍니다.

101 Switching Protocol

이 코드는 클라이언트가 보낸 Upgrade 요청 헤더에 대한 응답에 들어가며 서버에서 프로토콜을 변경할 것임을 알려줍니다.

102 Processing (WebDAV)

이 코드는 서버가 요청을 수신하였으며 이를 처리하고 있지만, 아직 제대로 된 응답을 알려줄 수 없음을 알려줍니다.

103 Early Hints

이 상태 코드는 주로 Link 헤더와 함께 사용되어 서버가 응답을 준비하는 동안 사용자 에이전트가(user agent) 사전 로딩(preloading)을 시작할 수 있도록 한다.

성공 응답

200 OK

요청이 성공적으로 되었습니다. 성공의 의미는 HTTP 메소드에 따라 달라집니다:

GET: 리소스를 불러와서 메시지 바디에 전송되었습니다.

HEAD: 개체 헤더가 메시지 바디에 있습니다.

PUT 또는 POST: 수행 결과에 대한 리소스가 메시지 바디에 전송되었습니다.

TRACE: 메시지 바디는 서버에서 수신한 요청 메시지를 포함하고 있습니다.

201 Created

요청이 성공적이었으며 그 결과로 새로운 리소스가 생성되었습니다. 이 응답은 일반적으로 POST 요청 또는 일부 PUT 요청 이후에 따라옵니다.

202 Accepted

요청을 수신하였지만 그에 응하여 행동할 수 없습니다. 이 응답은 요청 처리에 대한 결과를 이후에 HTTP로 비동기 응답을 보내는 것에 대해서 명확하게 명시하지 않습니다. 이것은 다른 프로세스에서 처리 또는 서버가 요청을 다루고 있거나 배치 프로세스를 하고 있는 경우를 위해 만들어졌습니다.

203 Non-Authoritative Information

이 응답 코드는 돌려받은 메타 정보 세트가 오리진 서버의 것과 일치하지 않지만 로컬이나 서드 파티 복사본에서 모아졌음을 의미합니다. 이러한 조건에서는 이 응답이 아니라 200 OK 응답을 반드시 우선됩니다.

204 No Content

요청에 대해서 보내줄 수 있는 콘텐츠가 없지만, 헤더는 의미있을 수 있습니다. 사용자-에이전트는 리소스가 캐시된 헤더를 새로운 것으로 업데이트 할 수 있습니다.

205 Reset Content

이 응답 코드는 요청을 완수한 이후에 사용자 에이전트에게 이 요청을 보낸 문서 뷰를 리셋하라고 알려줍니다.

206 Partial Content

이 응답 코드는 클라이언트에서 복수의 스트림을 분할 다운로드를 하고자 범위 헤더를 전송했기 때문에 사용됩니다.

207 Multi-Status (WebDAV)

멀티-상태 응답은 여러 리소스가 여러 상태 코드인 상황이 적절한 경우에 해당되는 정보를 전달합니다.

208 Multi-Status (WebDAV)

DAV에서 사용됩니다: propstat(property와 status의 합성어) 응답 속성으로 동일 컬렉션으로 바인드된 복수의 내부 멤버를 반복적으로 열거하는 것을 피하기 위해 사용됩니다.

226 IM Used (HTTP Delta encoding)

서버가 GET 요청에 대한 리소스의 의무를 다 했고, 그리고 응답이 하나 또는 그 이상의 인스턴스 조작이 현재 인스턴스에 적용이 되었음을 알려줍니다.

HTTP status code

리다이렉션 메시지

300 Multiple Choice

요청에 대해서 하나 이상의 응답이 가능합니다. 사용자 에이전트 또는 사용자는 그중에 하나를 반드시 선택해야 합니다. 응답 중 하나를 선택하는 방법에 대한 표준화 된 방법은 존재하지 않습니다.

301 Moved Permanently

이 응답 코드는 요청한 리소스의 URI가 변경되었음을 의미합니다. 새로운 URI가 응답에서 아마도 주어질 수 있습니다.

302 Found

이 응답 코드는 요청한 리소스의 URI가 일시적으로 변경되었음을 의미합니다. 새롭게 변경된 URI는 나중에 만들어질 수 있습니다. 그러므로, 클라이언트는 향후의 요청도 반드시 동일한 URI로 해야합니다.

303 See Other

클라이언트가 요청한 리소스를 다른 URI에서 GET 요청을 통해 얻어야 할 때, 서버가 클라이언트로 직접 보내는 응답입니다.

304 Not Modified

이것은 캐시를 목적으로 사용됩니다. 이것은 클라이언트에게 응답이 수정되지 않았음을 알려주며, 그러므로 클라이언트는 계속해서 응답의 캐시된 버전을 사용할 수 있습니다.

305 Use Proxy

이전 버전의 HTTP 기술 사양에서 정의되었으며, 요청한 응답은 반드시 프록시를 통해서 접속해야 하는 것을 알려줍니다. 이것은 프록시의 in-band 설정에 대한 보안상의 걱정으로 인하여 사라져가고 있습니다.

306 unused

이 응답 코드는 더이상 사용되지 않으며, 현재는 추후 사용을 위해 예약되어 있습니다. 이것은 HTTP 1.1 기술사양 이전 버전에서 사용되었습니다.

307 Temporary Redirect

클라리언트가 요청한 리소스가 다른 URI에 있으며, 이전 요청과 동일한 메소드를 사용하여 요청해야할 때, 서버가 클라이언트에 이 응답을 직접 보냅니다. 이것은 302 Found HTTP 응답 코드와 동일한 의미를 가지고 있으며, 사용자 에이전트가 반드시 사용된 HTTP 메소드를 변경하지 말아야 하는 점만 다릅니다: 만약 첫 요청에 post가 사용되었다면, 두번째 요청도 반드시 post를 사용해야 합니다.

308 Permanent Redirect

이것은 리소스가 이제 HTTP 응답 헤더의 Location: 에 명시된 영구히 다른 URI에 위치하고 있음을 의미합니다. 이것은 301 Moved Permanently HTTP 응답 코드와 동일한 의미를 가지고 있으며, 사용자 에이전트가 반드시 HTTP 메소드를 변경하지 말아야 하는 점만 다릅니다: 만약 첫 요청에 post가 사용되었다면, 두번째 요청도 반드시 post를 사용해야 합니다.

HTTP status code

400 Bad Request

이 응답은 잘못된 문법으로 인하여 서버가 요청을 이해할 수 없음을 의미합니다.

401 Unauthorized

비록 HTTP 표준에서는 "미승인(unauthorized)"를 명확히 하고 있지만, 의미상 이 응답은 "비인증(unauthenticated)"을 의미합니다. 클라이언트는 요청한 응답을 받기 위해서는 반드시 스스로를 인증해야 합니다.

402 Payment Required

이 응답 코드는 나중에 사용될 것을 대비해 예약되었습니다. 첫 목표로는 디지털 결제 시스템에 사용하기 위하여 만들어졌지만 지금 사용되고 있지는 않습니다.

403 Forbidden

클라이언트는 콘텐츠에 접근할 권리를 가지고 있지 않습니다. 예를들어 그들은 미승인이어서 서버는 거절을 위한 적절한 응답을 보냅니다. 401과 다른 점은 서버가 클라이언트가 누구인지 알고 있습니다.

404 Not Found

서버는 요청받은 리소스를 찾을 수 없습니다. 브라우저에서는 알려지지 않은 URL을 의미합니다. 이것은 API에서 종점은 적절하지만 리소스 자체는 존재하지 않음을 의미할 수도 있습니다. 서버들은 인증받지 않은 클라이언트로부터 리소스를 숨기기 위하여 이 응답을 403 대신에 전송할 수도 있습니다. 이 응답 코드는 웹에서 반복적으로 발생하기 때문에 가장 유명할지도 모릅니다.

405 Method Not Allowed

요청한 메소드는 서버에서 알고 있지만, 제거되었고 사용할 수 없습니다. 예를 들어, 어떤 API에서 리소스를 삭제하는 것을 금지할 수 있습니다. 필수적인 메소드인 `GET`과 `HEAD`는 제거될 수 없으며 이 에러 코드를 리턴할 수 없습니다.

406 Not Acceptable

이 응답은 서버가 **서버 주도 콘텐츠 협상** 을 수행한 이후, 사용자 에이전트에서 정해진 규격에 따른 어떠한 콘텐츠도 찾지 않았을 때, 웹서버가 보냅니다.

407 Proxy Authentication Required

이것은 401과 비슷하지만 프록시에 의해 완료된 인증이 필요합니다.

408 Request Timeout

이 응답은 요청을 한지 시간이 오래된 연결에 일부 서버가 전송하며, 어떨 때에는 이전에 클라이언트로부터 어떠한 요청이 없었다고 하더라도 보내지기도 합니다. 이것은 서버가 사용되지 않는 연결을 끊고 싶어한다는 것을 의미합니다. 이 응답은 특정 몇몇 브라우저에서 빈번하게 보이는데, Chrome, Firefox 27+, 또는 IE9와 같은 웹서핑 속도를 올리기 위해 HTTP 사전 연결 메카니즘을 사용하는 브라우저들이 해당됩니다. 또한 일부 서버는 이 메시지를 보내지 않고 연결을 끊어버리기도 합니다.

409 Conflict

이 응답은 요청이 현재 서버의 상태와 충돌될 때 보냅니다.

410 Gone

이 응답은 요청한 콘텐츠가 서버에서 영구적으로 삭제되었으며, 전달해 줄 수 있는 주소 역시 존재하지 않을 때 보냅니다. 클라이언트가 그들의 캐쉬와 리소스에 대한 링크를 지우기를 기대합니다. HTTP 기술 사양은 이 상태 코드가 "일시적인, 홍보용 서비스"에 사용되기를 기대합니다. API는 알려진 리소스가 이 상태 코드와 함께 삭제되었다고 강요해서는 안된다.

411 Length Required

서버에서 필요로 하는 `Content-Length` 헤더 필드가 정의되지 않은 요청이 들어왔기 때문에 서버가 요청을 거절합니다.

412 Precondition Failed

클라이언트의 헤더에 있는 전제조건은 서버의 전제조건에 적절하지 않습니다.

413 Payload Too Large

요청 엔티티는 서버에서 정의한 한계보다 큼니다; 서버는 연결을 끊거나 혹은 `Retry-After` 헤더 필드로 돌려보낼 것이다.

414 URI Too Long

클라이언트가 요청한 URI는 서버에서 처리하지 않기로 한 길이보다 깁니다.

415 Unsupported Media Type

요청한 미디어 포맷은 서버에서 지원하지 않습니다, 서버는 해당 요청을 거절할 것입니다.

416 Requested Range Not Satisfiable

Range 헤더 필드에 요청한 지정 범위를 만족시킬 수 없습니다; 범위가 타겟 URI 데이터의 크기를 벗어났을 가능성이 있습니다.

417 Expectation Failed

이 응답 코드는 `Expect` 요청 헤더 필드로 요청한 예상이 서버에서는 적당하지 않음을 알려줍니다.

418 I'm a teapot

서버는 커피를 찌 주전자에 끓이는 것을 거절합니다.

421 Misdirected Request

서버로 유도된 요청은 응답을 생성할 수 없습니다. 이것은 서버에서 요청 URI와 연결된 스킴과 권한을 구성하여 응답을 생성할 수 없을 때 보내집니다.

422 Unprocessable Entity (WebDAV)

요청은 잘 만들어졌지만, 문법 오류로 인하여 따를 수 없습니다.

423 Locked (WebDAV)

리소스는 접근하는 것이 잠겨있습니다.

424 Failed Dependency (WebDAV)

HTTP status code

서버 에러 응답

500 Internal Server Error

서버가 처리 방법을 모르는 상황이 발생했습니다. 서버는 아직 처리 방법을 알 수 없습니다.

501 Not Implemented

요청 방법은 서버에서 지원되지 않으므로 처리할 수 없습니다. 서버가 지원해야 하는 유일한 방법은 `GET`와 `HEAD`이다. 이 코드는 반환하면 안됩니다.

502 Bad Gateway

이 오류 응답은 서버가 요청을 처리하는 데 필요한 응답을 얻기 위해 게이트웨이로 작업하는 동안 잘못된 응답을 수신했음을 의미합니다.

503 Service Unavailable

서버가 요청을 처리할 준비가 되지 않았습니다. 일반적인 원인은 유지보수를 위해 작동이 중단되거나 과부하가 걸렸을 때 입니다. 이 응답과 함께 문제를 설명하는 사용자 친화적인 페이지가 전송되어야 한다는 점에 유의하십시오. 이 응답은 임시 조건에 사용되어야 하며, `Retry-After:` HTTP 헤더는 가능하면 서비스를 복구하기 전 예상 시간을 포함해야 합니다. 웹마스터는 또한 이러한 일시적인 조건 응답을 캐시하지 않아야 하므로 이 응답과 함께 전송되는 캐싱 관련 헤더에 대해서도 주의해야 합니다.

504 Gateway Timeout

이 오류 응답은 서버가 게이트웨이 역할을 하고 있으며 적시에 응답을 받을 수 없을 때 주어집니다.

505 HTTP Version Not Supported

요청에 사용된 HTTP 버전은 서버에서 지원되지 않습니다.

506 Variant Also Negotiates

서버에 내부 구성 오류가 있다. 즉, 요청을 위한 투명한 콘텐츠 협상이 순환 참조로 이어진다.

507 Insufficient Storage

서버에 내부 구성 오류가 있다. 즉, 선택한 가변 리소스는 투명한 콘텐츠 협상에 참여하도록 구성되므로 협상 프로세스의 적절한 종료 지점이 아닙니다.

508 Loop Detected (WebDAV)


서버가 요청을 처리하는 동안 무한 루프를 감지했습니다.

510 Not Extended

서버가 요청을 이행하려면 요청에 대한 추가 확장이 필요합니다.

511 Network Authentication Required

511 상태 코드는 클라이언트가 네트워크 액세스를 얻기 위해 인증을 받아야 할 필요가 있음을 나타냅니다.

이렇게 많은데,,, 어떻게 외워,,, 

HTTP status code

상태코드	표시 상태
1xx	조건부 응답
2xx	성공
3xx	리다이렉션 완료
4xx	요청 오류
5xx	서버 오류

HTTP status code

상태코드		표시 상태	상태코드		표시 상태
200	OK	성공	400	Bad Request	서버가 요청을 이해하지 못함
201	Create	성공, 서버가 새로운 리소스를 생성	401	Unauthorized	인증이 필요
204	No Content	성공, 전달해줄 응답 데이터 없음	403	Forbidden	요청을 거부함
301	Moved permanently	우회한 페이지로 영구적 이동	404	Not Found	페이지, 리소스를 찾을 수 없음
302	Found	우회한 페이지로 일시적 이동	500	Internal Server error	서버 내부 오류
304	Not modified	캐시 목적, 요청 이후 수정된 것이 없음	503	Service unvailble	일시적으로 서버를 이용할 수 없음

Node.js

<https://nodejs.org/en/about/>

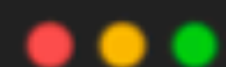
공식문서를 본다면...

Node.js는 비동기 이벤트 중심의 JavaScript 런타임환경이에요!

먼저 간단하게 정리하자면... 이벤트가 생성될 때만 실행이 되고, 모든 이벤트를 비동기적으로 실행시키는것이 Node.js입니다.

이제 천천히 알아보도록 합시다!!

EVENT-Driven



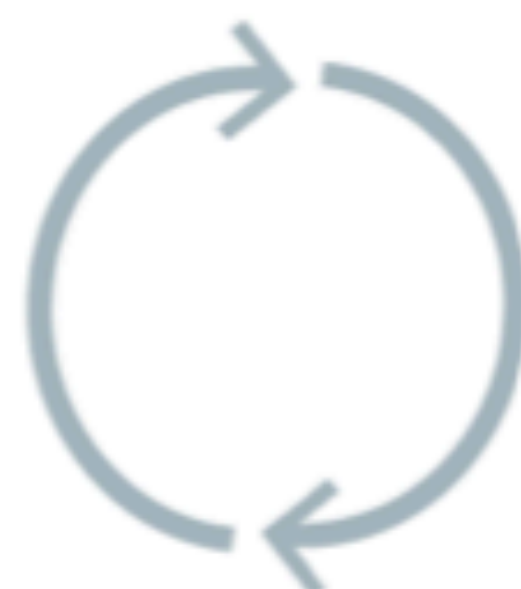
```
function greet() {  
  return "Hello!";  
}  
  
function timer() {  
  return setTimeout(() => {  
    return "End!";  
  }, 3000);  
}  
  
greet();  
timer();
```

CALL STACK

실행되는 함수들이
스택의 구조로
쌓이는 곳

BACKGROUND

로직이 실행되는 공간



EVENT LOOP

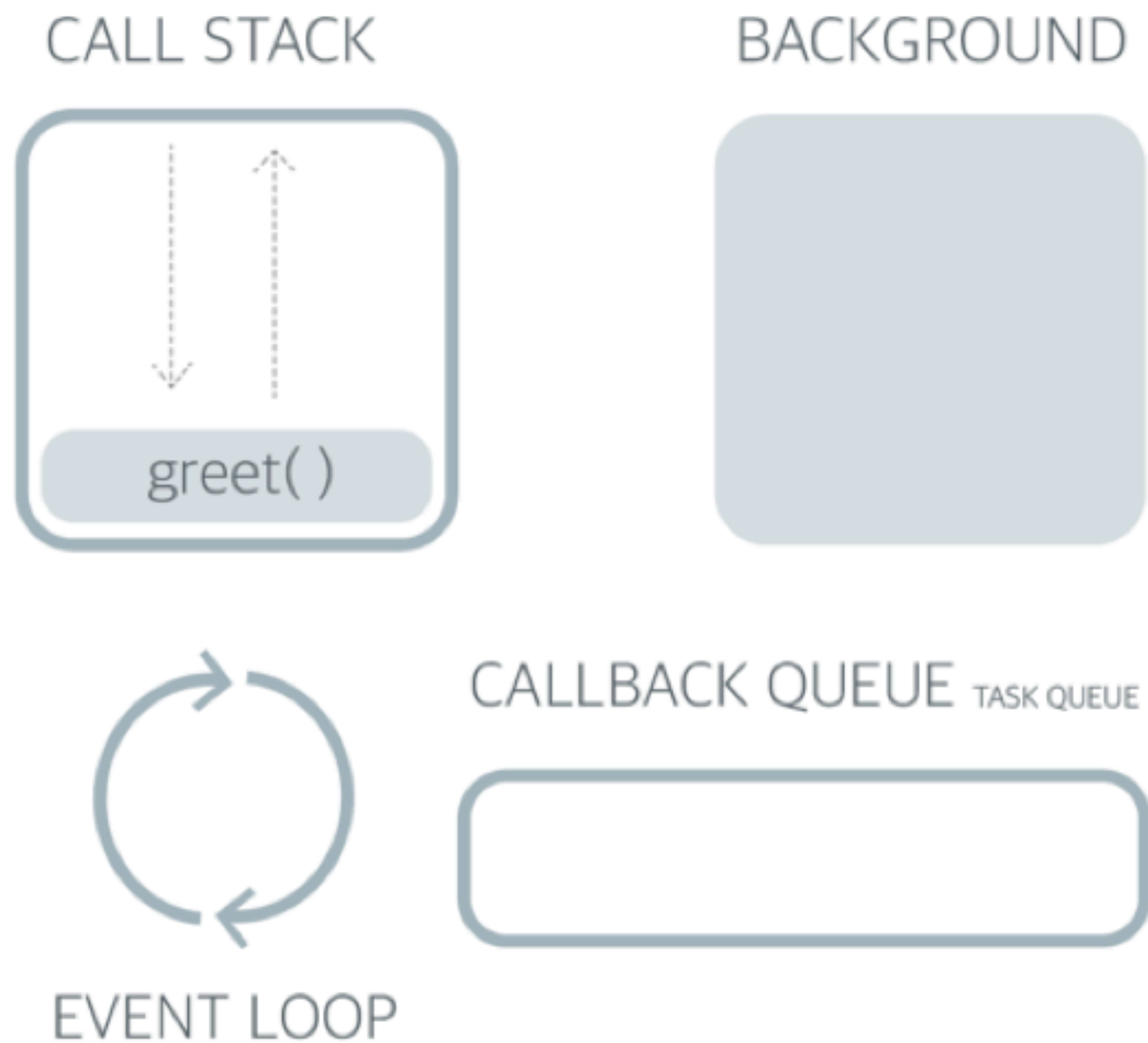
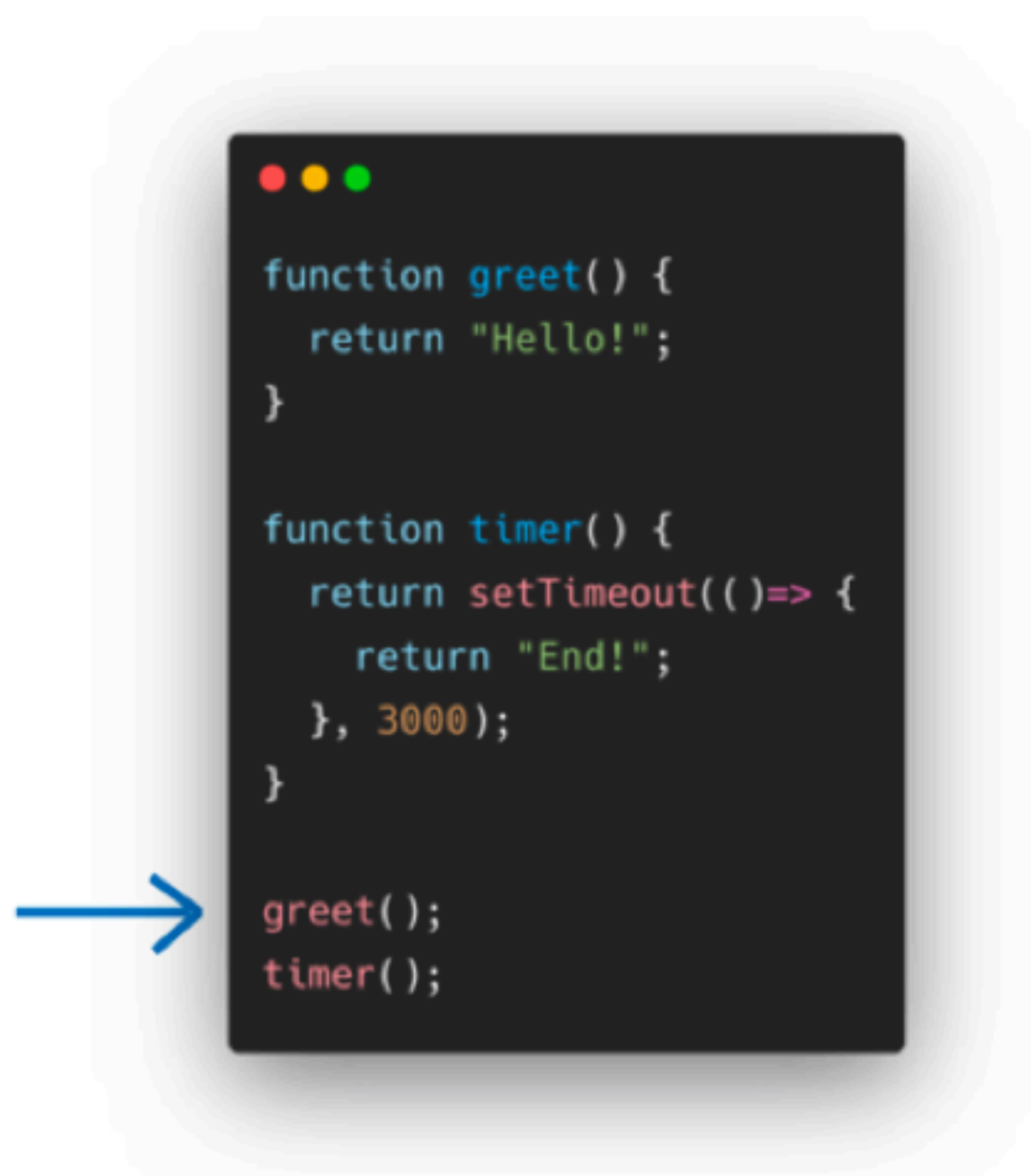
콜백 큐의 콜백함수를
콜 스택에 전달
단, 콜 스택이 비어있을 때만 가져옴

CALLBACK QUEUE TASK QUEUE

이벤트 발생 후
콜백 함수들이 기다리는 공간

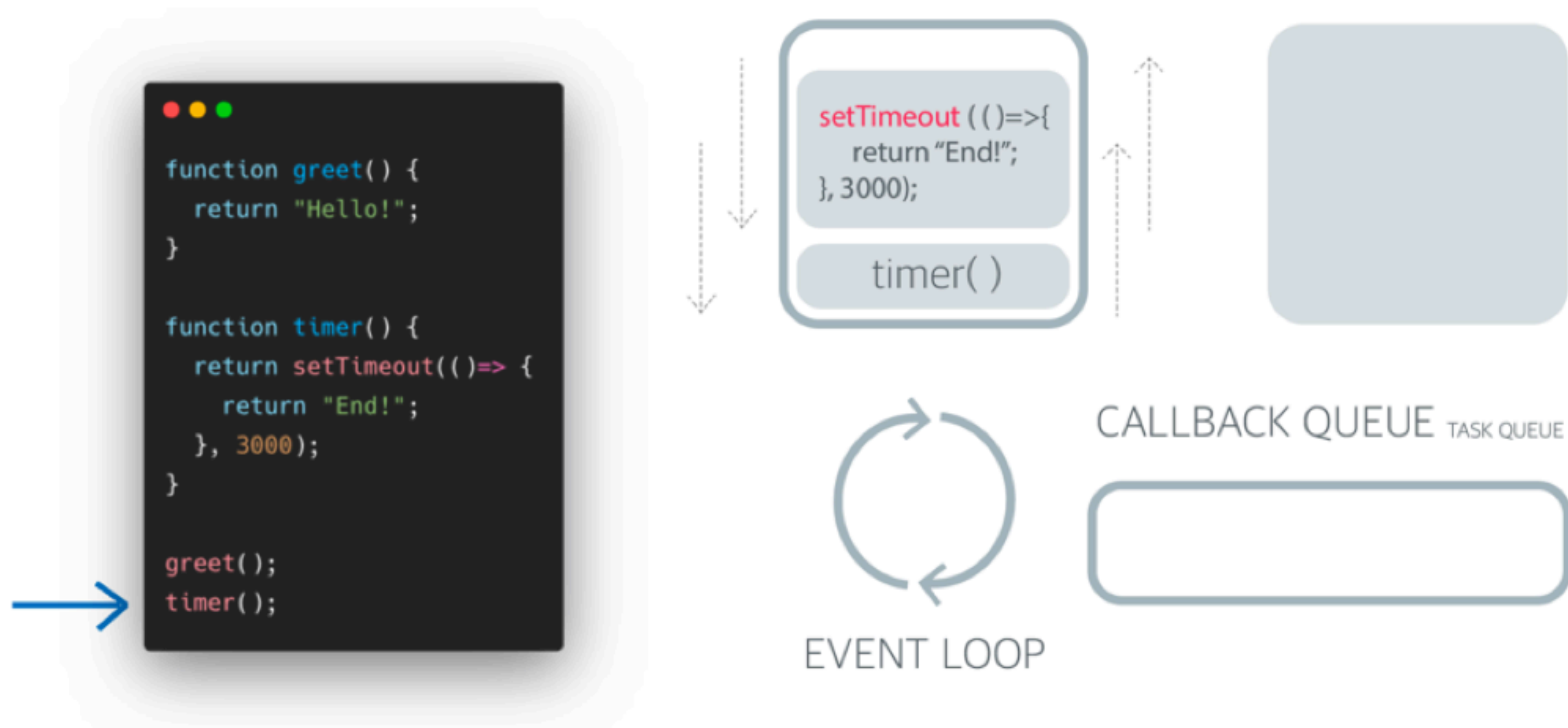
CALLBACK FUNCTION

어떤 이벤트가 발생했거나,
특정 시점에 도달했을 때 시스템에서 호출하는 함수.
다른 함수의 인자로써 이용



1. `greet()` 함수가 콜 스택호출 스택으로 Push되었다가 Pop되면서 “Hello” 을 출력합니다.

-DRIVEN



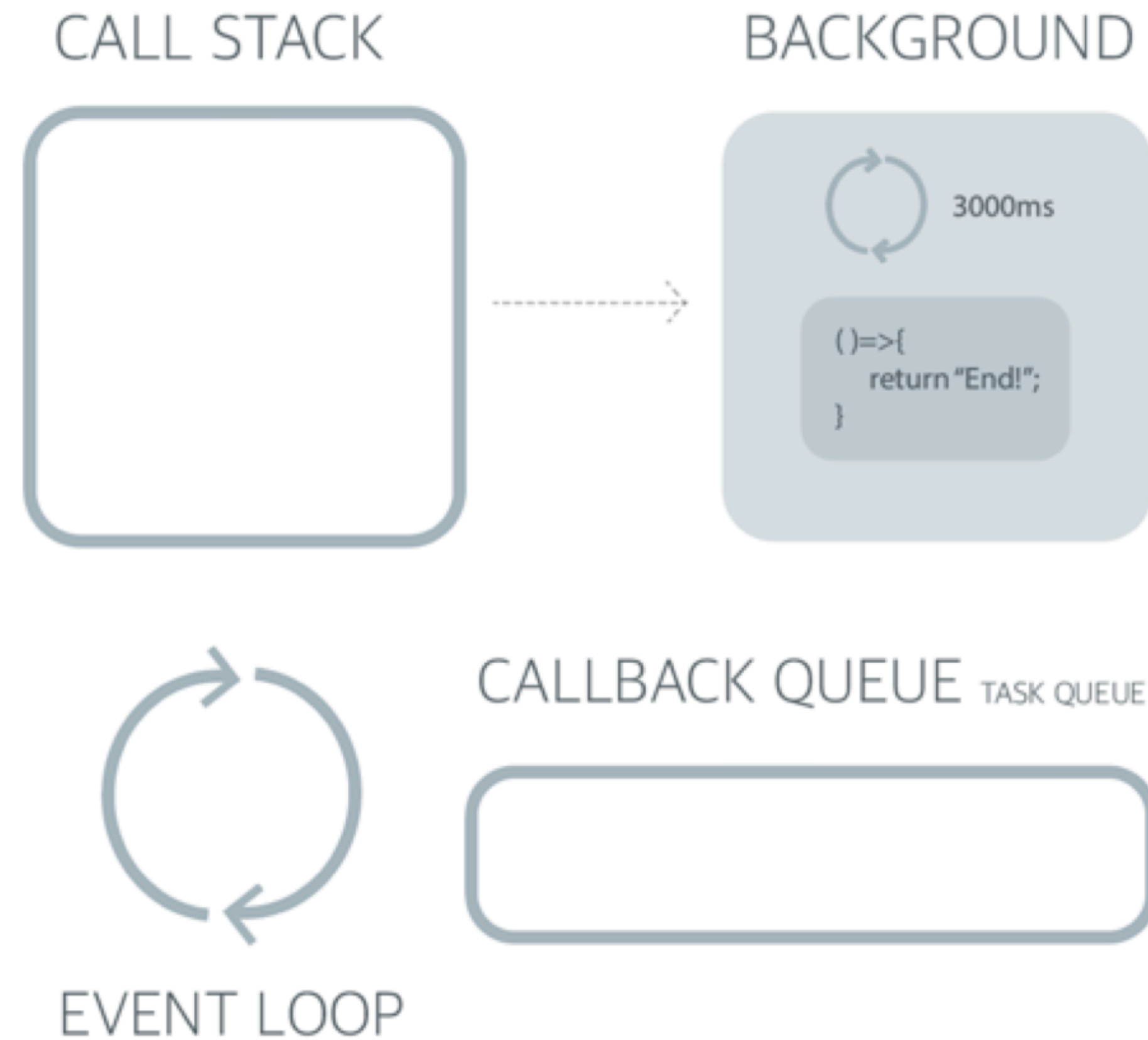
2. `timer()` 함수가 콜 스택 호출 스택으로 Push되고,

`setTimeout()` 함수가 push 되었다가 백그라운드로 pop됩니다.

또, 바로 `timer()`가 pop됩니다.

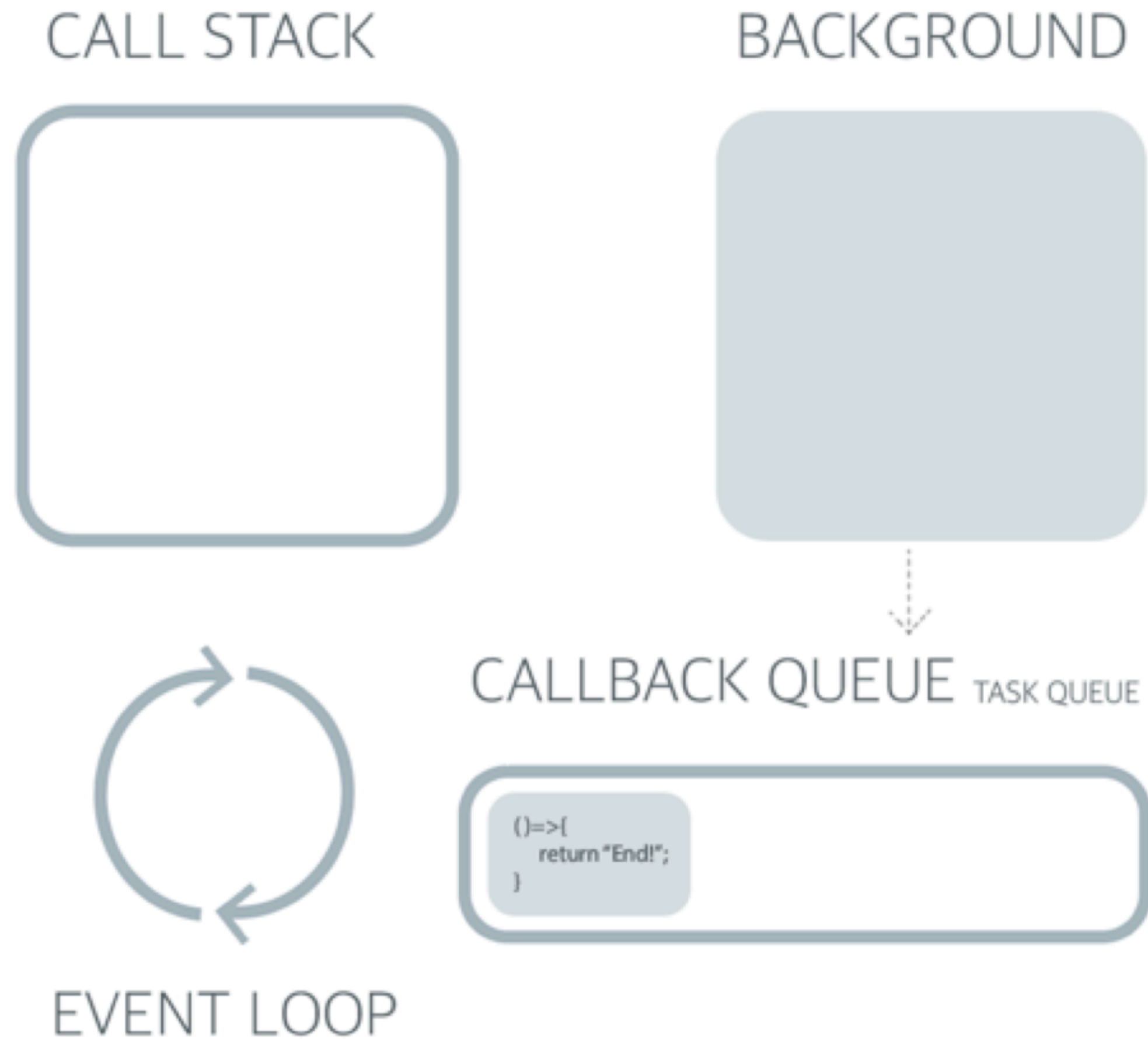
`setTimeout()` 함수에 의해 콜백함수가 백그라운드에서 실행됩니다.

```
function greet() {  
  return "Hello!";  
}  
  
function timer() {  
  return setTimeout(()=> {  
    return "End!";  
  }, 3000);  
}  
  
greet();  
timer();
```



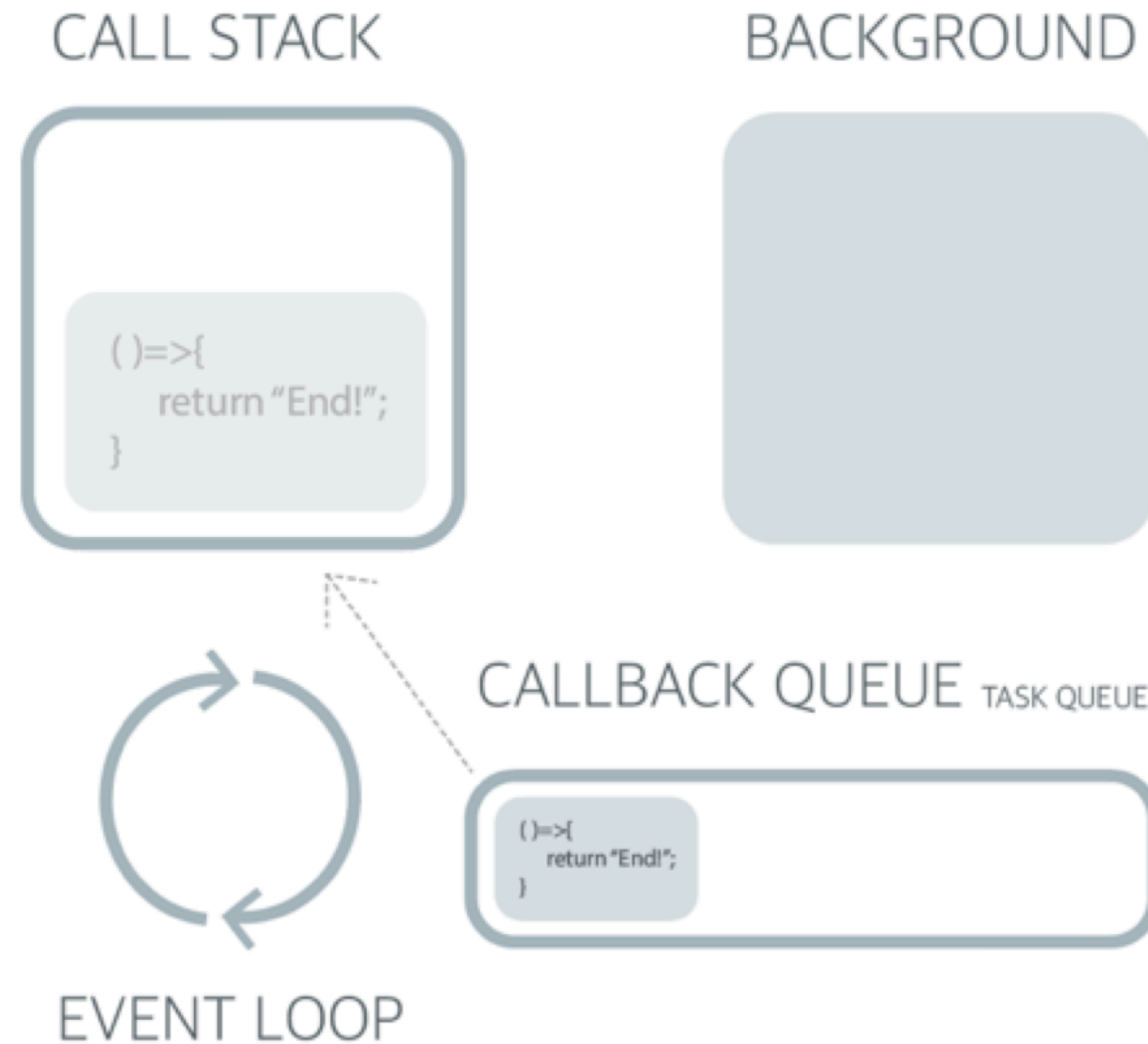
3. 콜 스택에서 pop되는 동시에 setTimeout()에 설정한 함수가 백그라운드로 추가됩니다.

```
function greet() {  
  return "Hello!";  
}  
  
function timer() {  
  return setTimeout(() => {  
    return "End!";  
  }, 3000);  
}  
  
greet();  
timer();
```



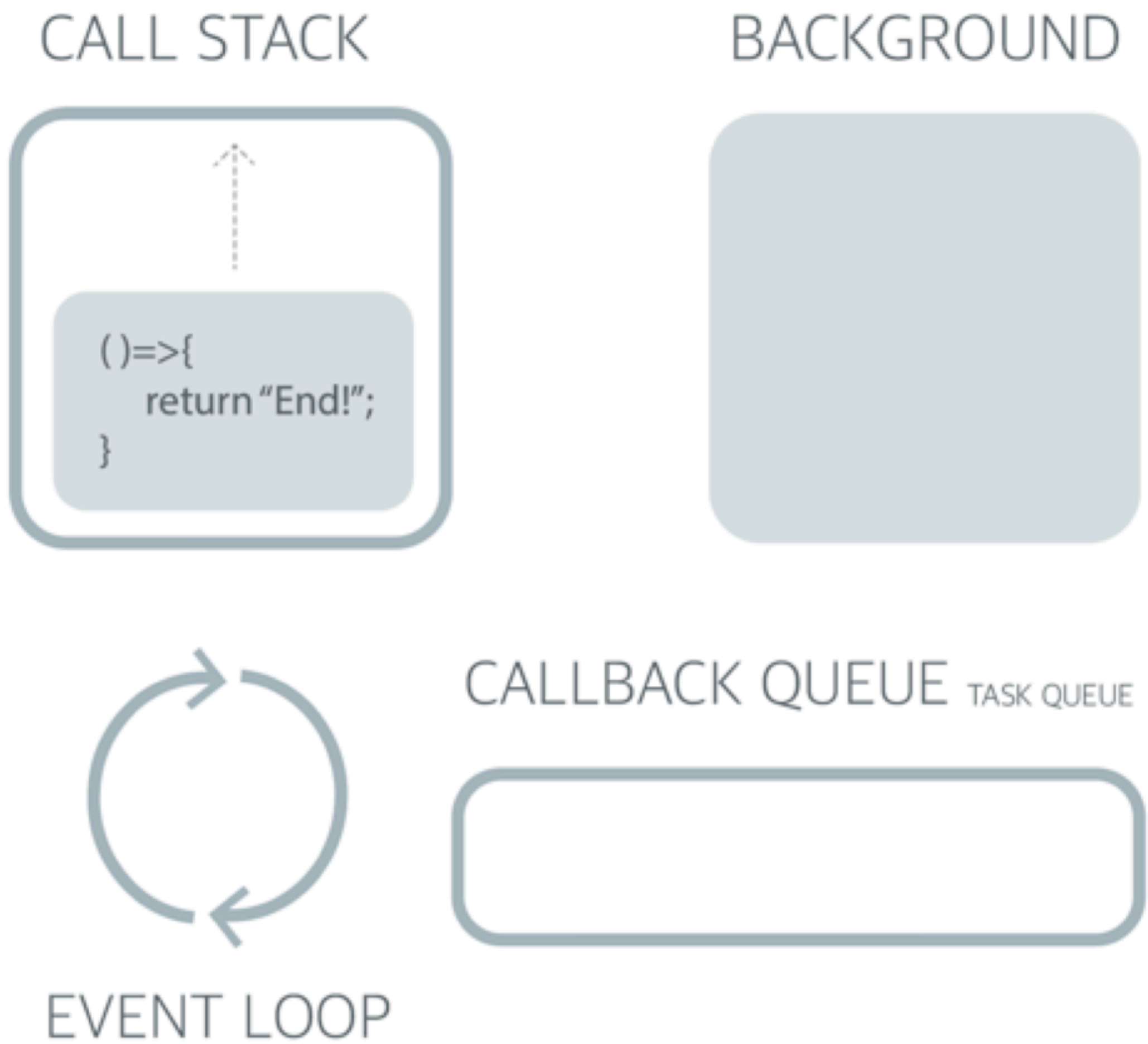
4. 설정된 시간이 끝나면 콜 스택이 아니라 콜백 큐 (대기열)로 이동합니다.

```
function greet() {  
  return "Hello!";  
}  
  
function timer() {  
  return setTimeout(() => {  
    return "End!";  
  }, 3000);  
}  
  
greet();  
timer();
```



5. 콜 스택이 비어있다면 이벤트루프가 콜백 큐에서 콜백함수를 콜 스택으로 옮깁니다.


```
function greet() {  
  return "Hello!";  
}  
  
function timer() {  
  return setTimeout(()=> {  
    return "End!";  
  }, 3000);  
}  
  
greet();  
timer();
```



6. 콜 스택에서 콜백함수를 pop합니다.

실제로 프로그램을 만들게 되면...???

```
console.log('Hello');
```

```
setTimeout(function() {  
  console.log('Bye');  
}, 3000);
```

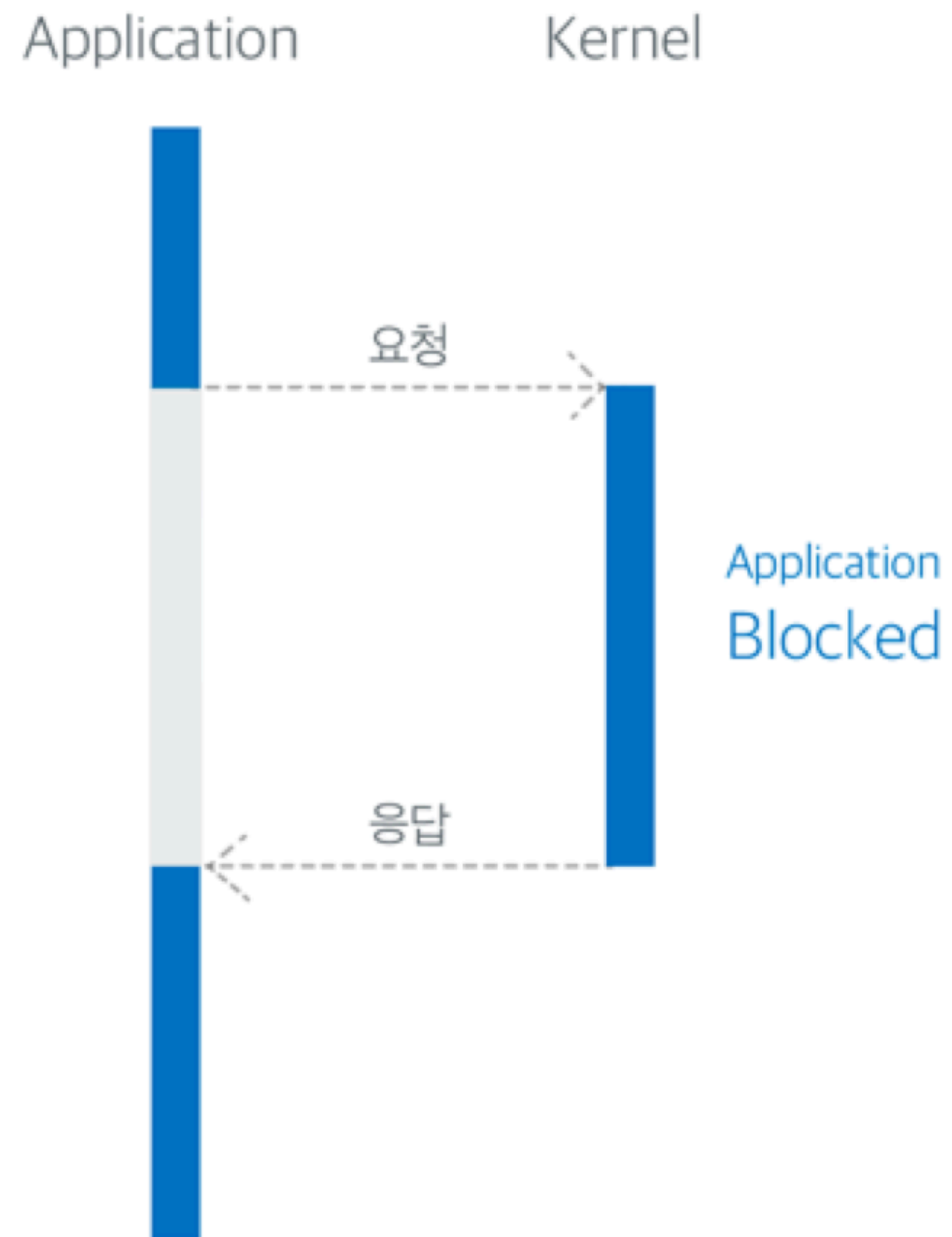
```
console.log('Hello Again');
```


Non-blocking I/O

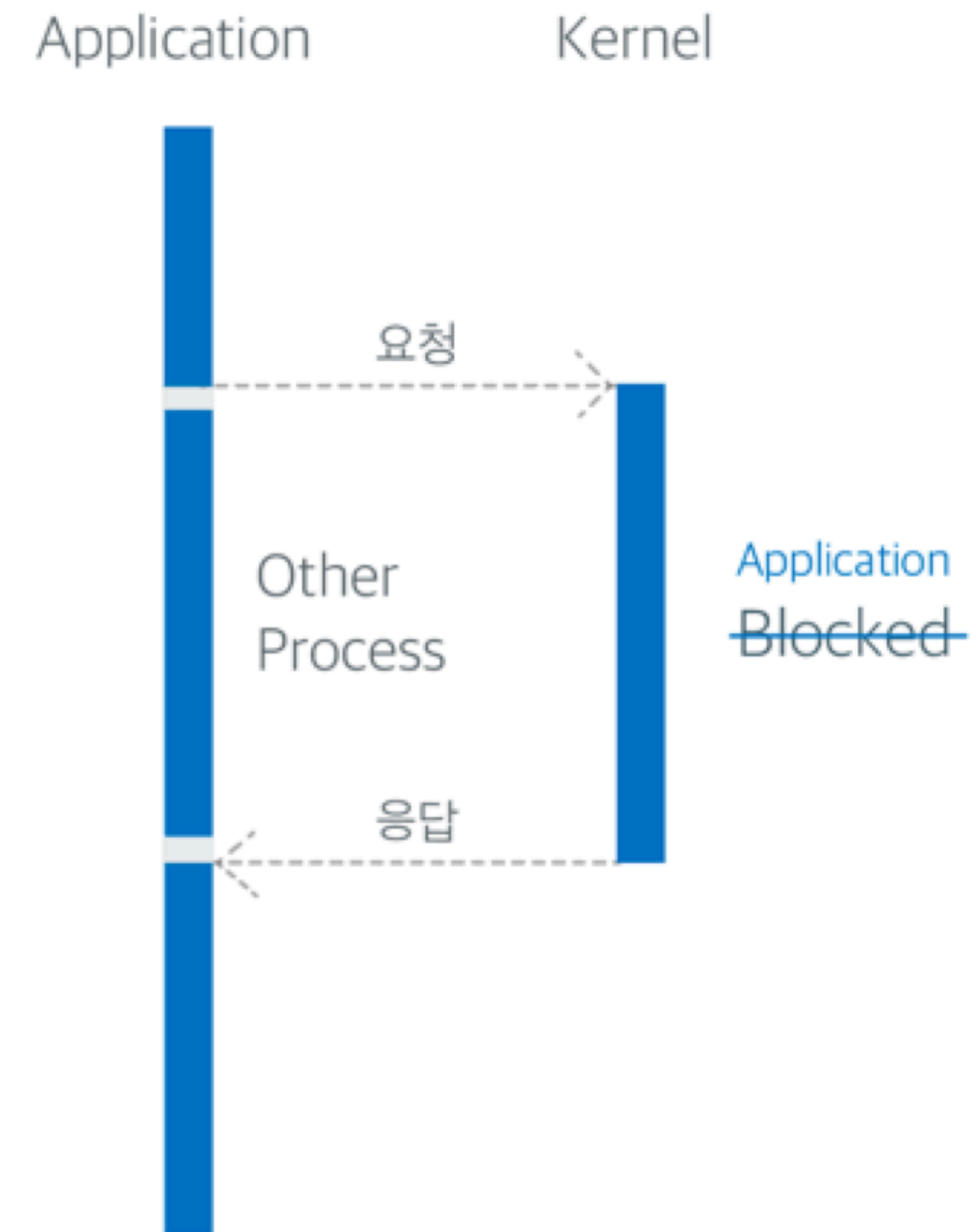
동기/비동기

Single Thread

BLOCKING



NON-BLOCKING



Blocking	Input / Output 작업이 진행중에는 중단된 채 대기
Non-Blocking	Input / Output 작업이 진행중에는 중단하지 않고 작업 진행

동기	요청을 하고 완료를 할 때 까지 기다리는 방식 백그라운드 작업 완료 여부를 계속 관찰함
비동기	요청을 하고 다시 다른 프로그램을 처리할수있음 완료 이벤트가 발생하면 미리 지정한 처리를 진행함

그럼... 프로그램 동기처리는 어떻게 하지...??

그냥 단순하게 작성해보자!!

```

function firshCallBack() {
  console.log("월요일")
}

function secondCallBack() {
  console.log("화요일")
}

function thirdCallBack() {
  setTimeout (function third() {
    console.log("수요일")
  }, 4000)
}

function fourthCallBack() {
  console.log("목요일")
}

function fifthCallBack() {
  console.log("금요일")
}

function sixthCallBack() {
  console.log("토요일")
}

function seventhCallBack() {
  console.log("일요일")
}

```

```

firshCallBack()
secondCallBack()
thirdCallBack()
fourthCallBack()
fifthCallBack()
sixthCallBack()
seventhCallBack()

```

```

function callBack(cb) {
  cb()
}

function week(day) {
  day = "월요일"
  callBack(function () {
    console.log(day)
    day = "화요일"
    callBack(function () {
      console.log(day)
      day = "수요일"
      callBack(function () {
        console.log(day)
        day = "목요일"
        callBack(function () {
          console.log(day)
          day = "금요일"
          callBack(function () {
            console.log(day)
            day = "토요일"
            callBack(function () {
              console.log(day)
              day = "일요일"
              callBack(function () {
                console.log(day)
              })
            })
          })
        })
      })
    })
  })
}

week()

```

아... 너무 복잡해...

Promise / Async / Await

Promise

Promise - ES6부터 공식적으로 포함된 흐름 제어 패턴

Promise를 쓰는 이유는???

비동기 이벤트기반인 Javascript에서 동기처럼 사용하기 위해서!!

Promise는 3가지 상태가 있어요!!

Pending : 비동기 처리 로직이 아직 완료되지 않은 상태

Fulfilled : 비동기 처리가 완료되어 프로미스가 결과 값을 반환해준 상태

Rejected : 비동기 처리가 실패하거나 오류가 발생한 상태

Pending - 비동기 처리 로직이 아직 완료되지 않은 상태

```
new Promise ( function(resolve, reject) {  
  
    ...code  
  
});
```

이렇게 작성을 해주면 Pending 상태인 Promise를 생성하는것이예요

function 안에 파라미터들은 무엇이냐???

동작에 대한 결과를 올바르게 건네줄수있으면 resolve

동작이 실패한다면 reject 를 사용합니다.

Fulfilled - 비동기 처리가 완료되어 프로미스가 결과 값을 반환해준 상태

```
new Promise ( function(resolve, reject) {  
  
    resolve(...code....);  
  
});
```

콜백 함수의 인자 resolve를 아래와 같이 실행하는 것이 fulfilled상태예요~~

Rejected - 비동기 처리가 실패하거나 오류가 발생한 상태

```
new Promise ( function(resolve, reject) {  
  
    reject(...code...);  
  
});
```

promise안에서 작업이 실패했다는 것을 넘겨주는 인자예요~~

Chaining - 여러개의 프로미스를 연결해 사용

then

- `promise.then(...function...)`

비동기 작업 완료 시 결과에 따라 함수 호출

catch

- `promise.catch(...function...)`

체이닝 형태로 연결된 상태에서 비동기 작업이
중간에 에러가 났을 때 호출

example.

Async / Await

Async / Await

Async

Promise를 사용하지 않고도 효과적으로 CallBack Hell을 해결

Async는 암묵적으로 promise를 반환

Await

Promise를 기다림 (성공 or 실패)

Async로 정의된 내부에서만 사용 가능

example.

서버 구성해보기

```

  ✓ project
    > bin
    > config
    > controllers
    > models
    > modules
    > node_modules
    > public
    > routes
    > views
  JS app.js
  {} package.json
  {} package-lock.json
  ⓘ README.md
```

bin : 서버를 실행하는 www 스크립트가 저장된 폴더

config : 서버에 필요한 환경설정을 하는 폴더

controllers : HTTP Response 전달하는 파일 관리 폴더

models : DB 모델을 관리하는 폴더

modules : 모듈을 관리하는 폴더

node_modules : Node.js 관리 폴더

public : 각종 리소스를 포함하는 폴더

routes : 페이지 라우팅과 관련된 파일을 저장하는 폴더

views : 웹 서버 사용시 사용자에게 보여주는 뷰 관리 폴더

app.js : 프로젝트를 관리하는 파일

package.json : npm 의존성 관리 파일 현재 프로젝트에 사용된
모듈을 설치하는데 필요한 코드를 관리하는 폴더

Study 파일 구성

```
6
7  var indexRouter = require('./routes/index');
8
9  var app = express();
10
11 // view engine setup
12 app.set('views', path.join(__dirname, 'views'));
13 app.set('view engine', 'jade');
14
15 app.use(logger('dev'));
16 app.use(express.json());
17 app.use(express.urlencoded({ extended: false }));
18 app.use(cookieParser());
19 app.use(express.static(path.join(__dirname, 'public')));
20 |
21 app.use('/', indexRouter);
22 |
```

App.js를 보시면

index라는 라우터를 기본적으로 가지고 있습니다.

index.router에서 다른 라우터를 연결하기 때문에

/routes/index.js로 넘어가죠!!

```
1  const express = require('express');
2  const router = express.Router();
3  const studyController = require('../controllers/studyController');
4
5  // GET
6  router.get('/', studyController.findPerson);
7  router.post('', studyController.findPerson);
8
9  // POST
10
11
12  module.exports = router;
13
```

study.js의 코드예요~!

study와 관련된 라우터를 관리해주는 파일이에요

그리고 controller에서 사용자에게 받는 데이터와 건네줄 데이터를 관리해주고 있어요 controller를 등록해준 다음 router로 http 메소드를 지정한 이후

컨트롤러에서 만들어준 함수를 적어주세요!!

METHOD

.get(...), .post(...)

response

response로 전달할 데이터를 담음



```
router.METHOD( 'path', (req, res) => { ... } );
```

path

접근할 path 설정

request

request로 전달받은 데이터들이 담겨있음

```
const StudyModel = require('../models/study');

module.exports = {
  get: async ( req, res ) => {
    const {
      id
    } = req.query;

    const person = await StudyModel.studyPerson(id);

    res.status(200).send(person)
  },
  post: async ( req, res ) => {
    const {
      id
    } = req.query

    const {
      body
    } = req.body

    console.log(id)
    console.log(body)
  }
}
```

StudyController파일입니다.

각각의 controller에서 작동할 code를 작성해주면 됩니다

req와 res으로 관리를 해주게 되는데



req.query

{{url}}/?member=server

req.query.member === server

req.params

{{url}}/post/:idx

req.params.idx === 23

req.headers

req.body

{"name": "gngsn", "part": "server"}

req.body.name === 'gngsn'

req.file



```
router.METHOD('path', (req, res) => { ... });
```

`res.status(xxx).send(json)`

`.status()` : status code를 정수로 입력

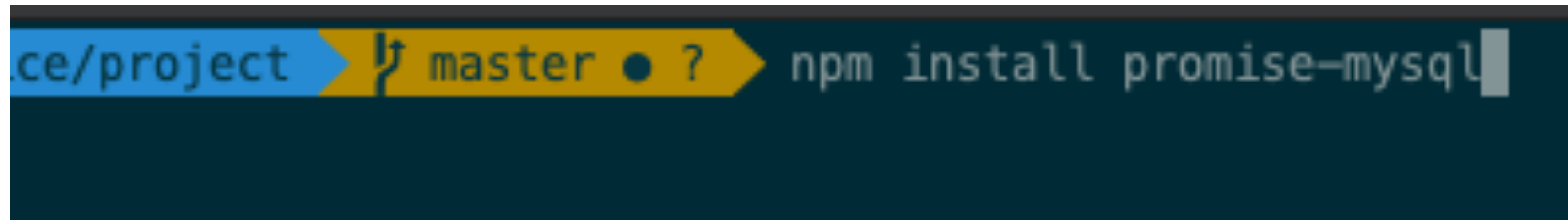
`.send()` : json형식으로 response body에 입력

규칙적인 데이터로 보내줘야 프론트와의 통신이 원활

DB 연결하기

Terminal에서 프로젝트 폴더에 들어갑니다~ npm install promise-mysql

config 폴더에서 database.js 파일을 만들어주세요~!

A screenshot of a terminal window with a dark blue background. The prompt shows the current directory as 'ce/project' in a blue box, followed by a yellow box indicating the current branch is 'master' with a question mark. The command 'npm install promise-mysql' is entered and the cursor is at the end of the line.

```
ce/project > master ● ? > npm install promise-mysql
```

database.js 파일에서 아래와 같이 코드를 작성해줍니다.

DB 연결을 하는 환경설정입니다

```
const mysql = require('promise-mysql');

const connection = {
  host: '127.0.0.1', // 로컬 호스트 주소
  port: 3306,        // 포트번호
  user: 'admin',      // 유저
  password: '1234',   // 비밀번호
  database: '0unce'   // 접근할 데이터베이스 (스키마)
}

module.exports = mysql.createPool(connection)
```

database.js 파일에서 아래와 같이 코드를 작성해줍니다.

DB를 사용하기 위한 모듈을 하나 추가해줄건데요!!

<https://github.com/5anniversary/node/blob/master/project/modules/pool.js>

pool 생성

```
pool = createPool(config)
```

connection pool을 생성

pool 사용 종료

```
releaseConnection(connection)
```

반납하지 않으면 connection이 쌓여서

connection leak 현상 발생

pool 사용

```
connection = pool.getConnection()
```

connection pool에서 connection을 가져옴

```
connection.query(query, [value])
```

query

String

실행할 쿼리 입력. 변수 자리에 ? 로 표시하면 런타임 시 배정.
혹은 백틱 문자열로 변수를 바로 넣어주어도 됨

values

[String]

쿼리문 안에 ? 순서에 맞게 들어갈 변수들이 배열 형식으로 들어감

Transaction

데이터베이스의 상태를 변화시키기 위해 수행하는 작업의 단위

→ 여러 단계의 처리를 하나처럼 다룸

commit : Transaction의 실행 결과를 데이터베이스에 반영하는 것

roll-back : Transaction의 실행 결과를 반영하지 않고 원상태로 되돌리는 것

Transaction

`connection.beginTransaction(config)`

Transaction 적용 시작

`connection.commit()`

Transaction 내 모든 쿼리 완료 후 결과 반영

`connection.rollback()`

모든 쿼리가 정상적으로 마치지 못했다면 원상태로 돌려놓음

DB 모델링을 해보자!!

1. models 폴더에 study.js라는 DB관리 파일을 만들어주세요
2. id를 가지고 해당 id에 해당하는 사람의 name과 part를 알아보는 API를 만들어볼거예요!!

```
const pool = require('../modules/pool');
const table = 'study';

const study = {
  studyPerson: async (id) => {
    const query = `SELECT name, part FROM ${table} WHERE id = ${id}`;
    console.log(query)
    try {
      const result = await pool.queryParamArr(query);
      console.log(result)
      return result;
    } catch (err) {
      console.log('error: ', err);
      throw err;
    }
  }
}

module.exports = study;
```

3. controller 파일로 돌아가보죠!!

4. 사용자 request의 parameter에서의 id값을 가지고 모델에서 사용자를 검색하는 코드를 작성했어요

```
module.exports = {  
  get: async ( req, res ) => {  
    const id = req.params.id;  
  
    const person = await StudyModel.studyPerson(id);  
  
    res.status(200).send(person)  
  },  
}
```


5. 마지막으로 router파일로 돌아가보죠!!

6. request의 parameter를 받아오기 위해서 아래의 get 메소드와 같이 작성해주세요

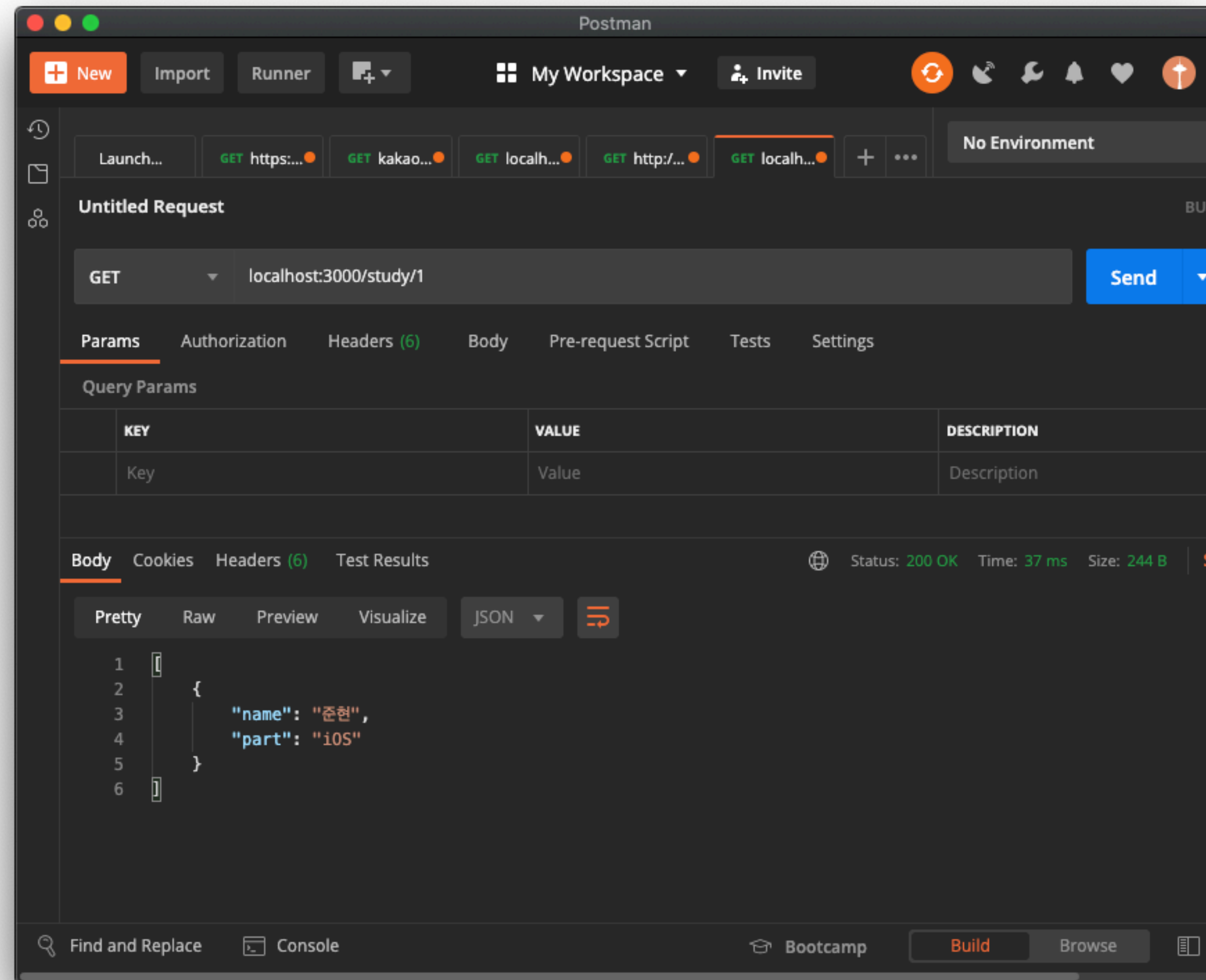
```
const express = require('express');
const router = express.Router();
const studyController = require('../controllers/studyController');

// GET
router.get('/:id', studyController.get);

// POST
router.post('/', studyController.post);

module.exports = router;
```

7. POSTMAN에서 localhost:3000/study/1 을 호출해주시면!!





과제

- 1.Study와 동일한 Ounce라는 테이블을 다시 만들어주세요!
- 2.사람을 추가할수있는 API
- 3.현재 Ounce에 속해있는 모든 사람들을 볼수있는 API
- 4.Ounce에서 이름으로 검색해서 사람을 볼수있는 API를 만들어주세요