* <https://medium.com/@gaurav.pandvia/understanding-javascript-function-executions-tasks-event-loop-call-stack-more-part-1-5683dea1f5ec>

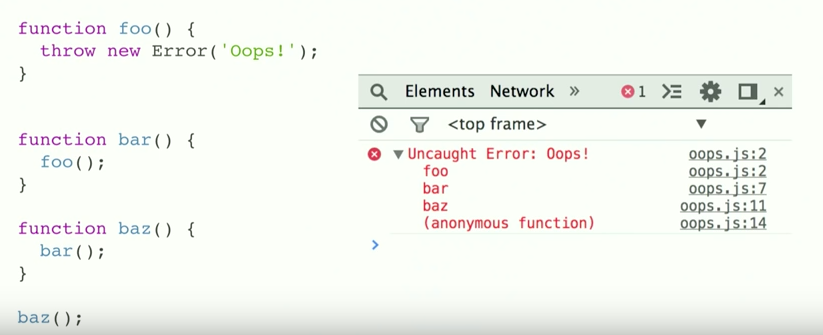
**Understanding Javascript Function Executions – Call Stack, Event Loop, Tasks & more**

Javascript is a single threaded single concurrent language 🡪 한 번에 한 task를 다룰 수 있음

* **용어 정리**

1. Call Stack

A data structure(실행되는 순서를 기억하고 있음), function을 call하면 stack에 push, 모두 push 되면 top부터 pop



Foo()가 call stack 최상단에 위치하고 pop되기 때문에 가장 먼저 실행되는 모습

1. Heap

Variables, objects에 모든 메모리 할당은 여기서 일어남

1. Queue

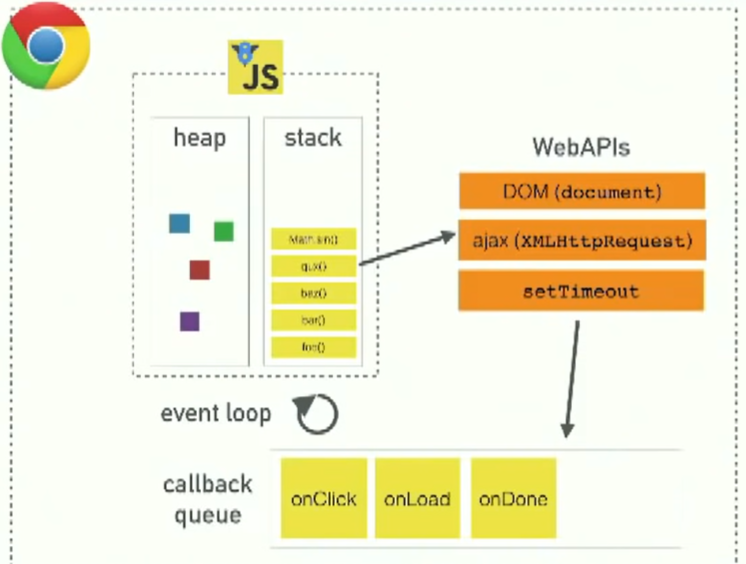
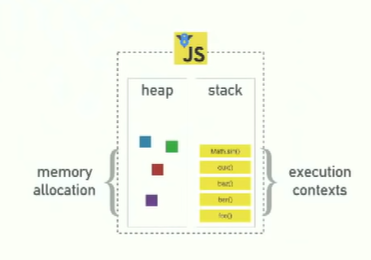
Js runtime은 message queue를 가짐. 이는 실행할 callback function과 관련된 것으로 수행될 messages들의 list를 관리함. Stack(call stack?)에 용량이 충분할 때 queue에서 message를 꺼내고 관련 function을 호출. Message processing은 stack이 빌 때 끝나며 이는 반복됨.

이 message들은 외부 async events에 응답으로 queue에 추가됨. 그래서 외부 동작에 callback function이 제공되지 않는다면 message는 없겠지 = enqueued가 일어나지 않겠지

* Event Loop

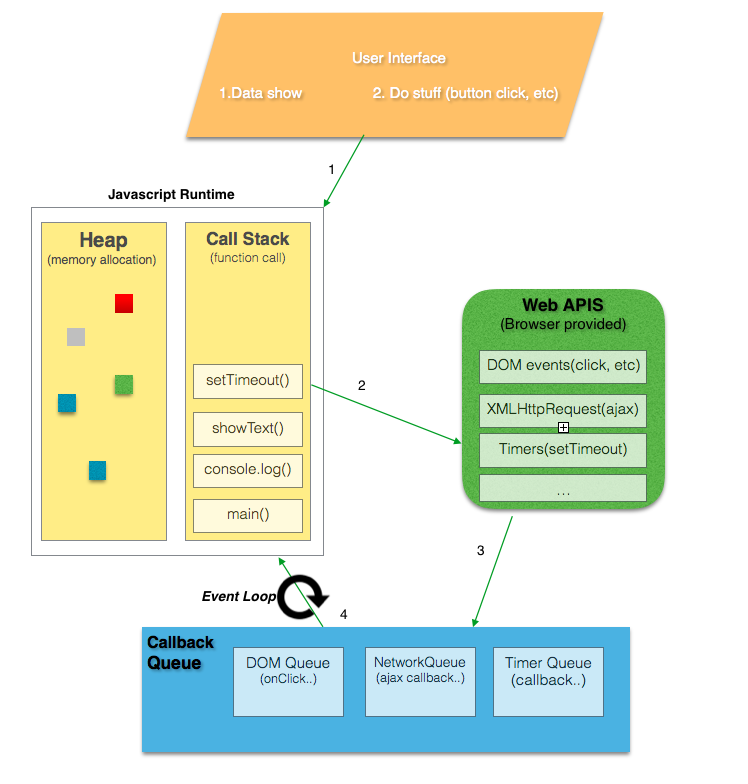
Call stack과 task queue를 주시하는 역할로 스택이 모두 비워진 후 queue에 있는 콜백을 stack에 push할 수 있다.

Ex) Console.log는 빠르지만 이게 반복적으로 실행된다면 많은 stack을 차지하게 될 것임 -> blocking script(Blocking의 명확한 정의는 없지만 대충 느린 동작이 stack에 남아있는 것을 일컫는다)

setTimeout을 0으로 설정하고 콜백함수를 돌리는 경우가 있음 🡪 일반적으로 스택이 비어 있을 때까지 기다린 후 실행하기 위해서 임. 스택에 코드가 있다면 렌더링을 못함. 렌더도 하나의 callback처럼 행동하기 때문. But 우리가 사용하던 콜백에 비해서는 더 높은 우선순위를 가짐.

결론: 오래 걸리는 코드는 비동기식으로 처리하는 것이 효율적



Web browsers에서는 event가 일어날 때마다 messages가 추가되며 이것에 event listener가 붙여짐. Listener가 없다면 event는 손실됨

Callback queue에 callback이 여러 개 쌓여 있고 이것들이 실행되는 중간에 call stack에 새로운 동기 task가 추가된다면 어떻게 될까?

[V8, chrome’s Runtime, Webpack, Babel, ESLint, Mocha, Karma, Grunt…]