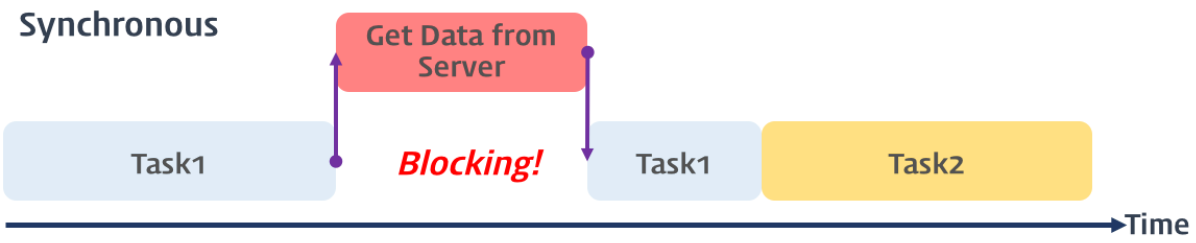
동기식 처리 모델 vs 비동기식 처리 모델

동기식 처리 모델(Synchronous processing model)은 직렬적으로 태스크(task)를 수행한다. 즉, 태스크는 순차적으로 실행되며 어떤 작업이 수행 중이면 다음 작업은 대기하게 된다.

예를 들어 서버에서 데이터를 가져와서 화면에 표시하는 작업을 수행할 때, 서버에 데이터를 요청하고 데이터가 응답될 때까지 이후 태스크들은 블로킹(blocking, 작업 중단)된다.

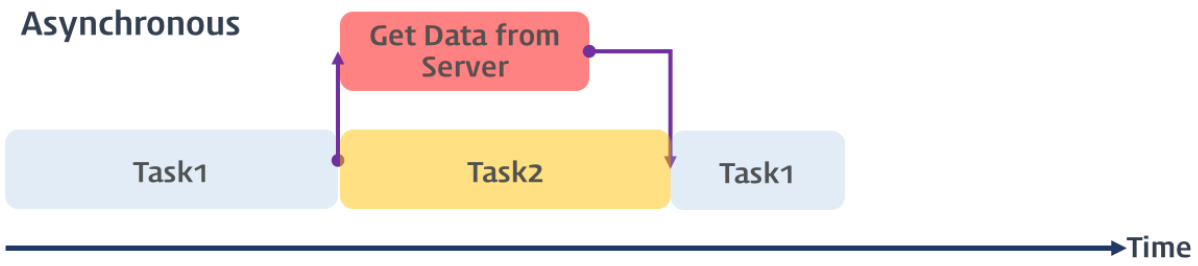


아래는 동기식으로 동작하는 코드이다. 순차적으로 실행된다.

```
function func1() {
  console.log('func1');
  func2();
}

function func2() {
  console.log('func2');
  func3();
}

function func3() {
  console.log('func3');
}

func1();
```

비동기식 처리 모델(Asynchronous processing model 또는 Non-Blocking processing model)은 병렬적으로 태스크를 수행한다. 즉, 태스크가 종료되지 않은 상태라 하더라도 대기하지 않고 다음 태스크를 실행한다. 예를 들어 서버에서 데이터를 가져와서 화면에 표시하는 태스크를 수행할 때, 서버에 데이터를 요청한 이후 서버로부터 데이터가 응답될 때까지 대기하지 않고(Non-Blocking) 즉시 다음 태스크를 수행한다. 이후 서버로부터 데이터가 응답되면 이벤트가 발생하고 이벤트 핸들러가 데이터를 가지고 수행할 태스크를 계속해 수행한다.

자바스크립트의 대부분의 DOM 이벤트 핸들러와 Timer 함수(setTimeout, setInterval), Ajax 요청은 비동기식 처리 모델로 동작한다.

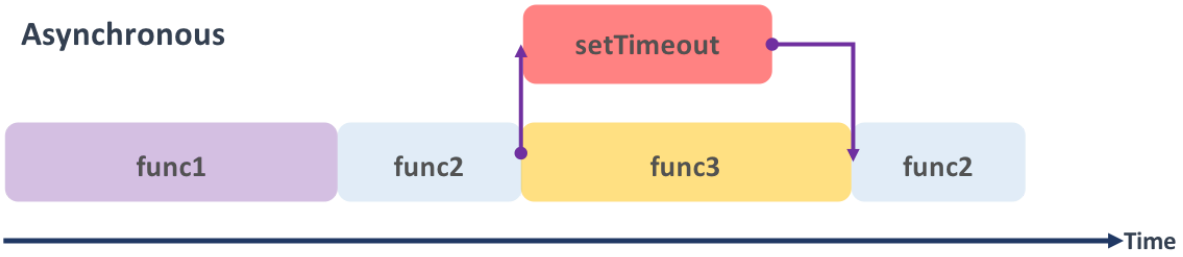


아래는 비동기식으로 동작하는 코드이다. 순차적으로 실행되지 않는다.

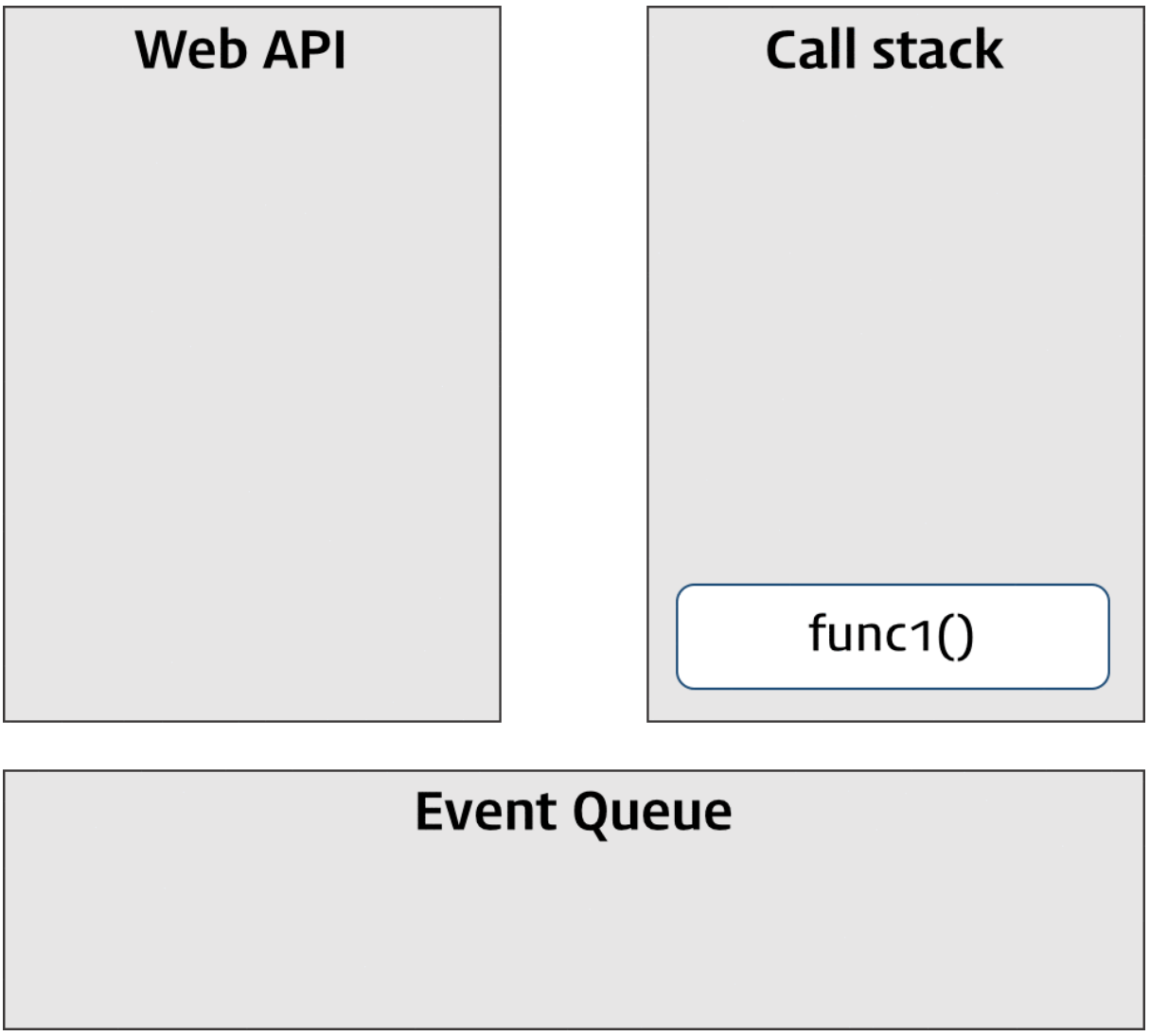
```
function func1() {
  console.log('func1');
  func2();
}

function func2() {
  setTimeout(function() {
    console.log('func2');
  }, 0);

  func3();
}

function func3() {
  console.log('func3');
}

func1();
```

위 예제를 실행하면 setTimeout 메소드에 두번째 인수 인터벌을 0초로 설정하여도 콘솔에 “func1 func2 func3”의 순서로 로그가 출력되지 않는다. 이는 setTimeout 메소드가 비동

기 함수이기 때문이다.



함수 func1이 호출되면 함수 func1은 Call Stack에 쌓인다. 그리고 함수 func1은 함수 func2을 호출하므로 함수 func2가 Call Stack에 쌓이고 setTimeout가 호출된다. **setTimeout의 콜백함수는 즉시 실행되지 않고 지정 대기 시간만큼 기다리다가 "tick" 이벤트가 발생하면 태스크 큐로 이동한 후 Call Stack이 비어졌을 때 Call Stack으로 이동되어 실행된다.**


Web API
Call stack
func1()
Event Queue