

* destructuring

```
var {body} = document;
```

K	V
:	
<u>body</u>	<u>200</u>
:	

~~200~~
<body> — </body>

document.getElementsByTagName("body") [0]

body == document.body ==

* function



function prototype (C/C++)
= method signature (Java)

function 함수명 (파라미터, 파라미터, ...)

Function body {
 `문장1;`
 :
 return 표현식;
}

리턴값 : "aaa", 20, true, {}-[], []
변수 : a, score, sum ...
식 : a + "hello", a * 2, 함수호출 ...

함수호출
(함수 선언시켜놓은 것)
⇒ 함수호출 (인자, 인자, ...);
 ↑ ↓
 아주먼드 (argument)

* 아규먼트와 파라미터

$f(\underline{10});$

function $f(a, b)$ {

}

\equiv

arguments = [10]

- 모든 함수에 두 가지는 Built-in 매개변수
- 아규먼트 변수를 사용하기 때문에

* 아규먼트와 파라미터

$f(10, 20);$

function $f(a, b) \{$

\equiv

}

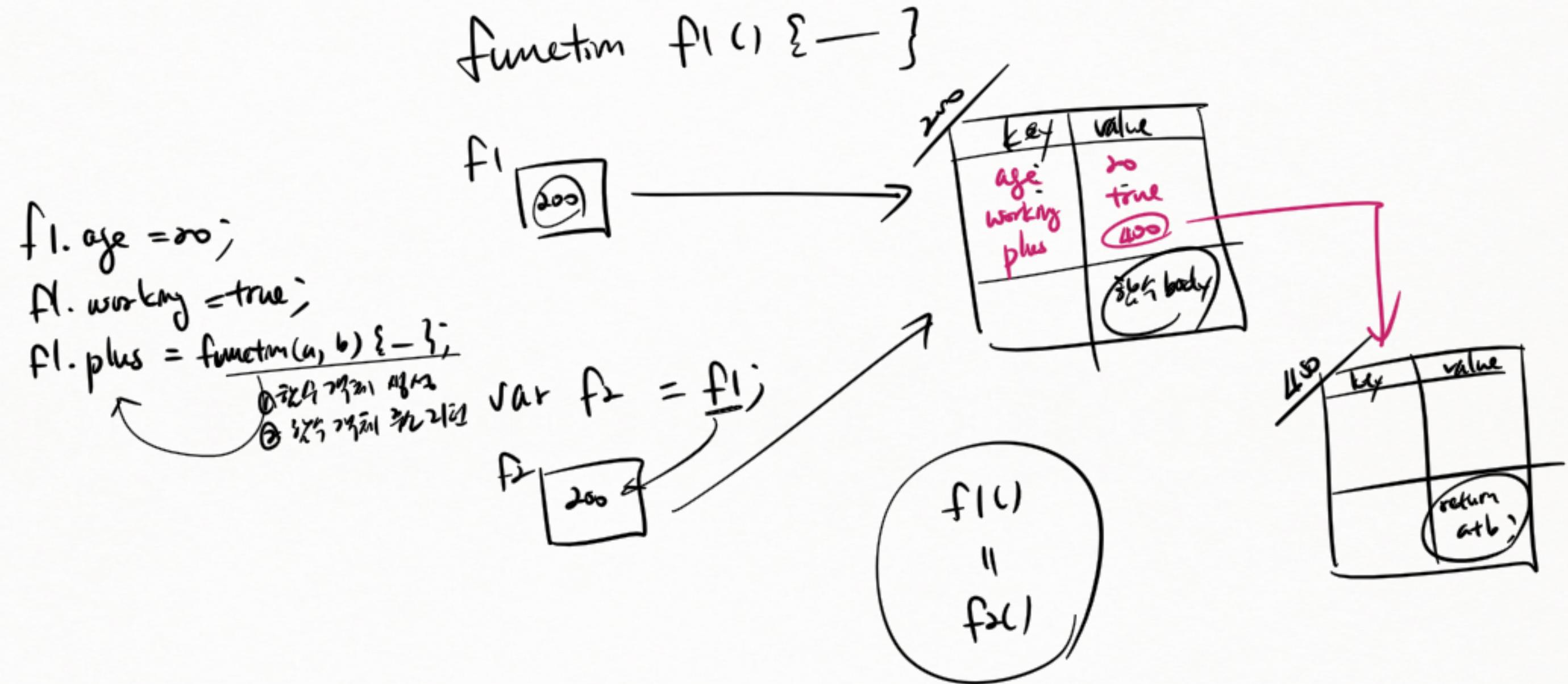
arguments = [10, 20]

* 아규먼트와 파라미터

$f(10, 20, 30, 40);$

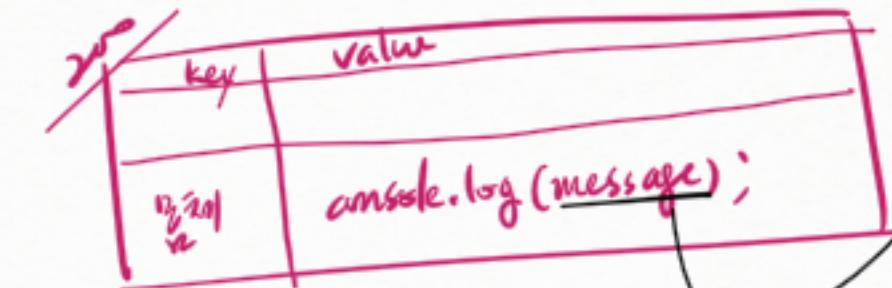
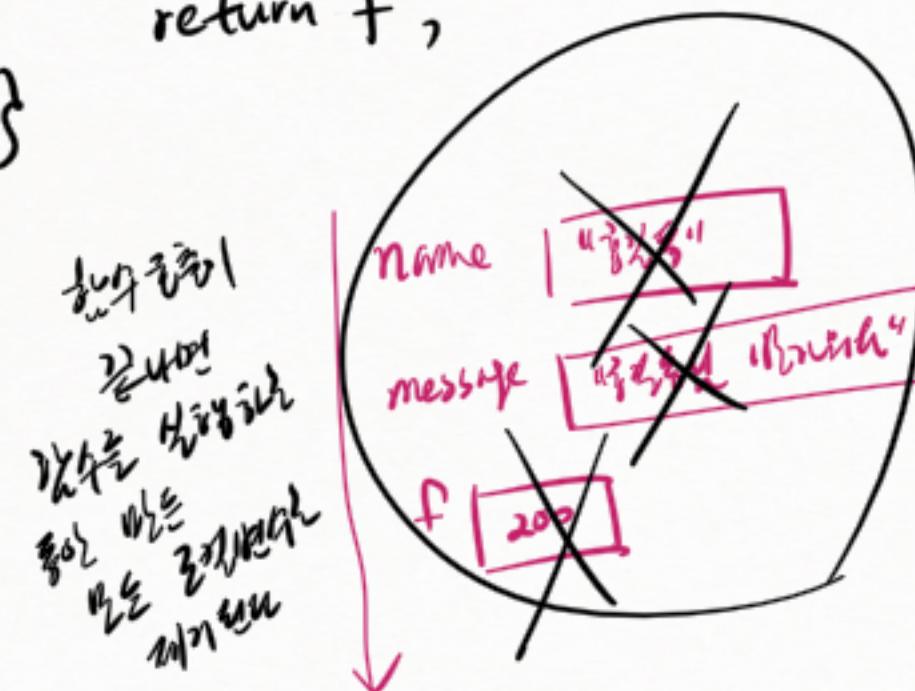
function $f(a, b)$ {
 \equiv arguments = [10, 20, 30, 40]
}

* 흡수와 레퍼런스



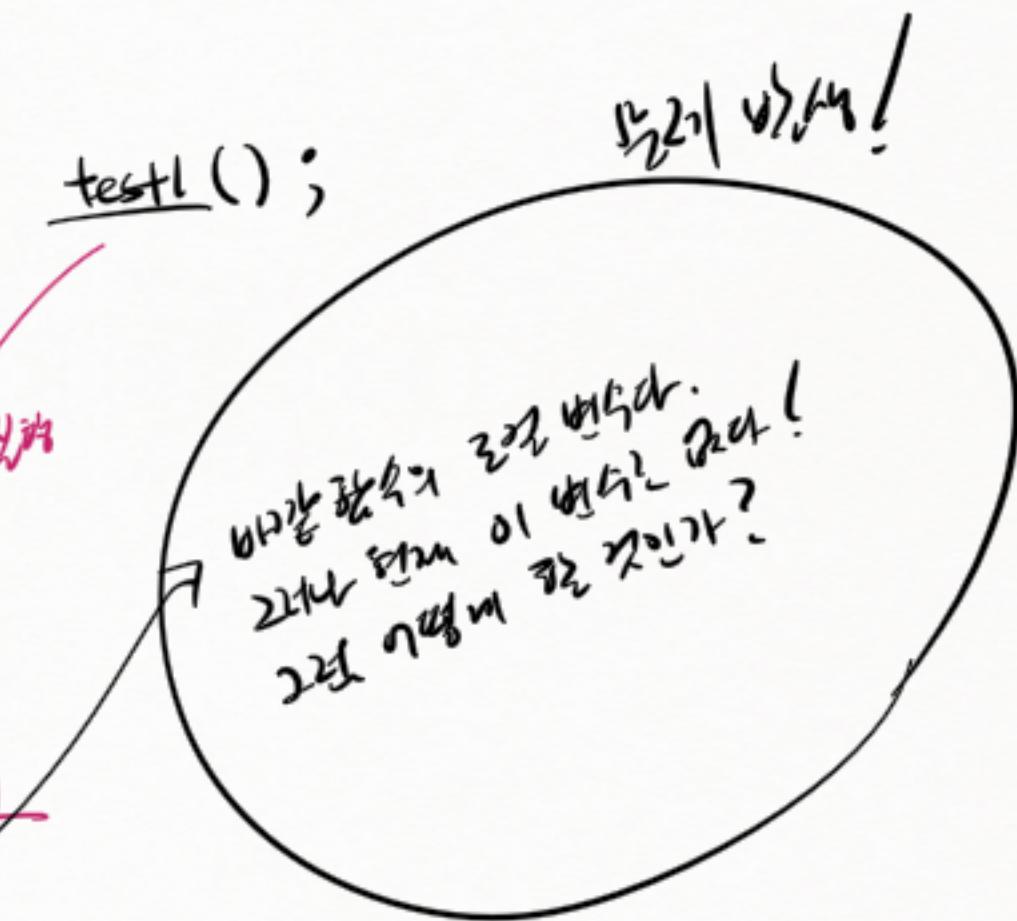
* closure

```
function createGreeting(name) {  
    var message = name + "님 반갑습니다";  
  
    var f = function() { console.log(message); };  
  
    return f;  
}
```



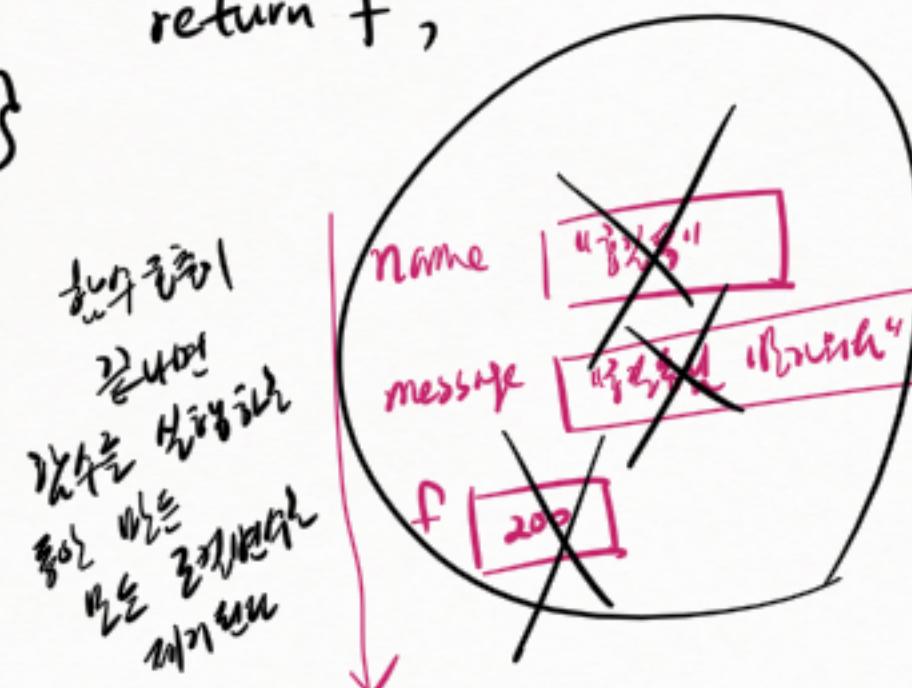
```
var test1 = createGreeting("김민석");  
test1  
200
```

- name
- message
- *



* closure : 100% 깨끗이 청결 상태를
갖고 있어야 한다!

```
function createGreeting(name) {  
    var message = name + "님 반갑습니다";  
    var f = function() { console.log(message); }  
}
```



closure 를 만들 때
클로저가 사용하는
내부 함수의 로컬변수가
아니면,
클로저의 내부를
복제해온다.

key	value
log	console.log(<u>message</u>);

key	value
message	" "

복지제도 평가를 하라거나

```
var test1 = createGreeting("John");
```

test1 → 200

200

- name
- message
- *

test()

1802 1803
message right
done
right!

* closure 例

test(1)
test2()

var test1 = createGreeting ("Hello");

test1

200

key	value
	console.log(message);

closure 例 2
var test1 = createGreeting ("Hello");
var test2 = createGreeting ("World");

key	value
message	"Hello World"

var test2 = createGreeting ("World");

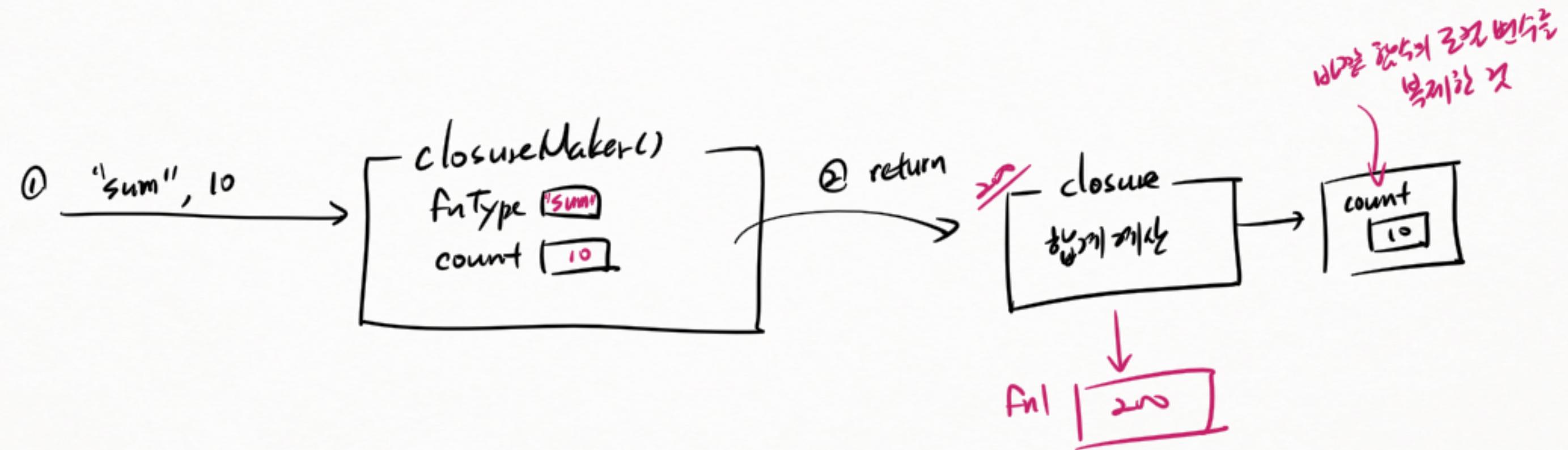
test2

300

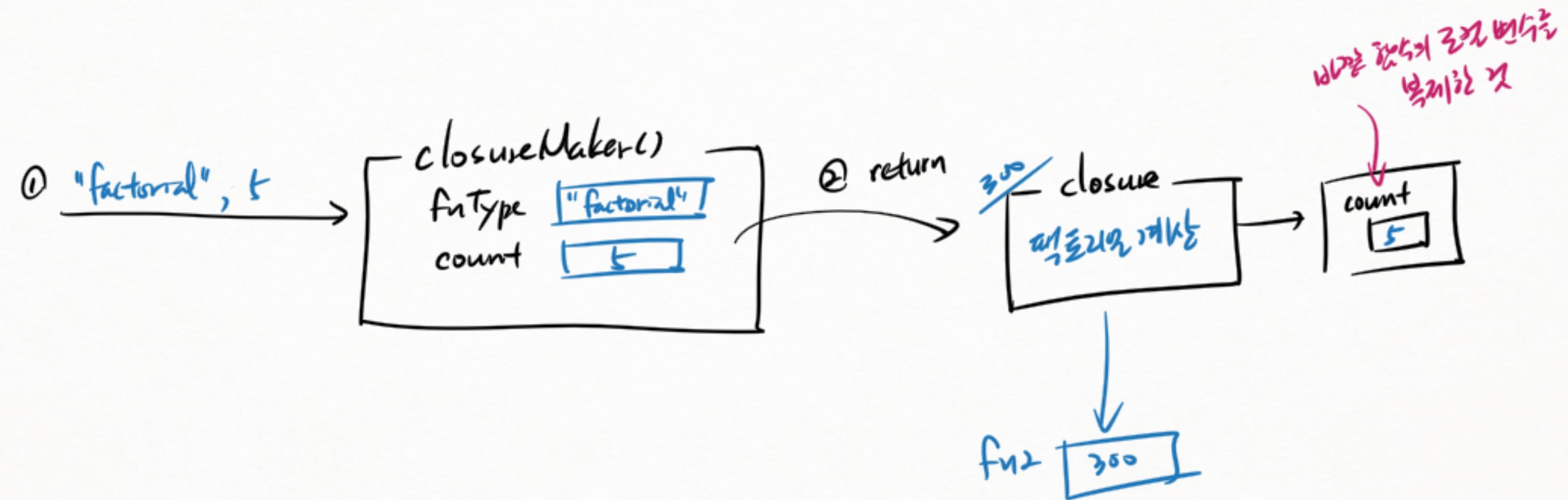
key	value
	console.log(message);

key	value
message	"Hello World"

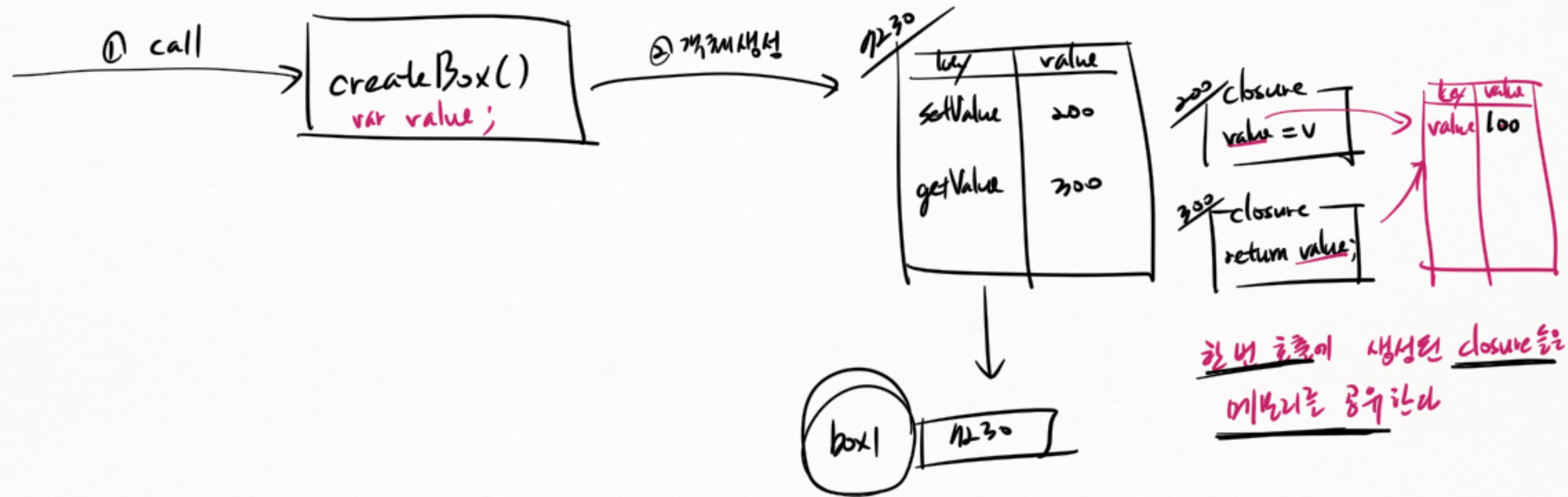
* closure № II



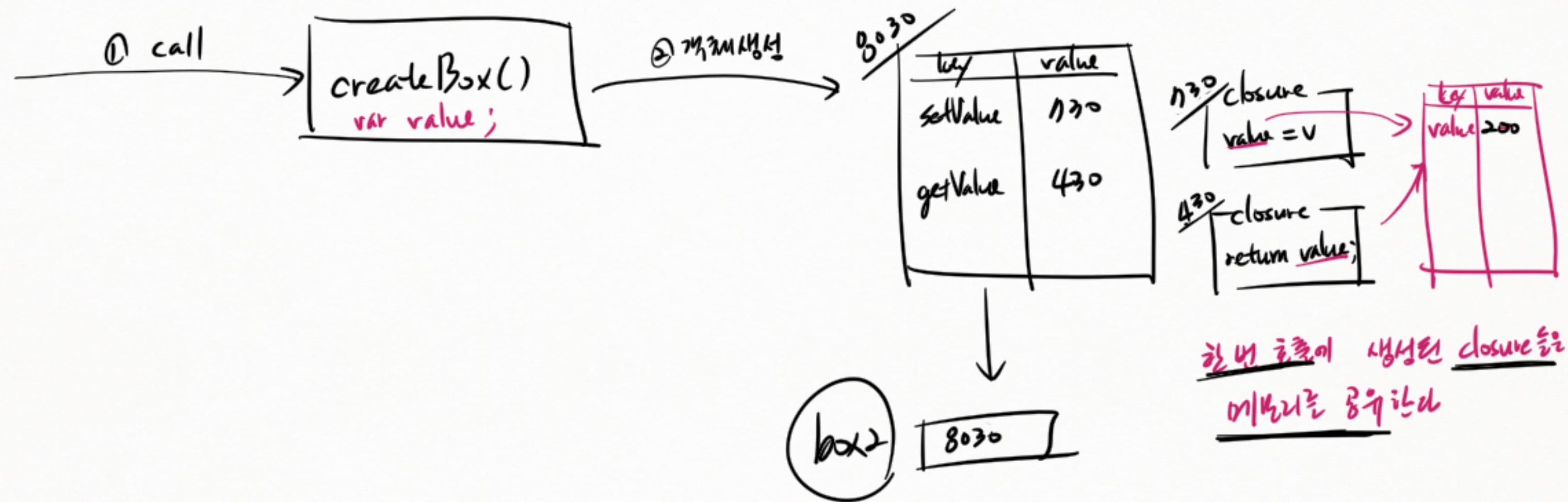
* closure № II



* 4th Ⅲ : box1

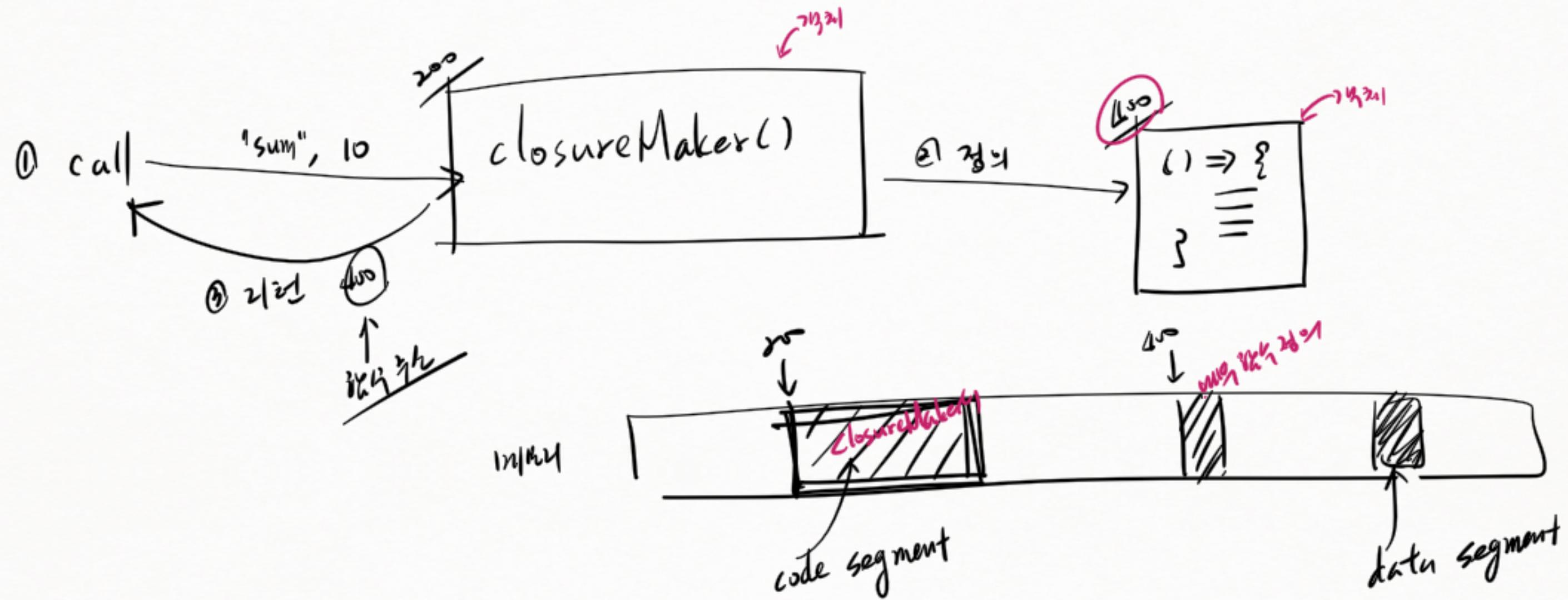


* 4th Ⅲ : box 2

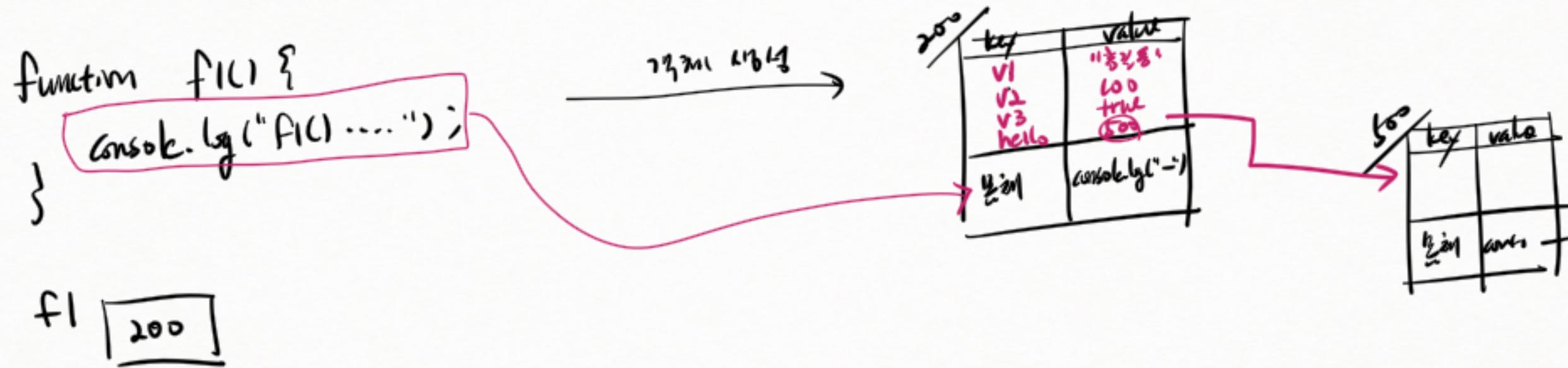


```
function() { return "안녕" }  
          ↓  
console.log( (값주기)() )  
          ↑  
값주기를 가지는 함수 호출  
  
          ( ) => "안녕"  
          ↓  
console.log( (값주기)() )
```

* closure 2번



* 值 (值)



`f1();` $\xrightarrow{\text{语句}} \text{语句!} \Rightarrow \text{"值" 语句(function call)"}$

`f1.v1 = "Hello";`

`f1.v2 = 100;`

`f1.v3 = true;`

`f1.hello = function() {
 console.log("Hello!");
};`

`f1.hello();` $\xrightarrow{\text{语句}} \text{语句!} \xrightarrow{\text{语句}} \text{语句!} \Rightarrow \text{语句!}$

* 동기화 와 비동기화
Synchronize Asynchronize

Synchronize(동기화)

기본적인 문법
변수 선언
함수 정의
변수 초기화
결과 출력
var a = 100;
var b = 200;
var result = plus(a, b);
console.log(result);

Asynchronize(비동기화)

var a = 100;
var b = 200;
var result = 0;
function calculate() {
 result = a + b;
}
window.setTimeout(plus, 5000);
console.log(result);

호이스팅 예제
비동기화
선행 초기화

* eval()

"JavaScript is"
↓
eval() → this

* onclick 속성

HTML의 onclick = 함수명;

↑
호출할 때마다
이벤트 핸들러
 onclick 이벤트
 메시지 처리
 응답

event property
↑
(callback)
" " event handler
" " event listener

Web Browser

* JSON.parse()

JSON 풀기

① JavaScript 객체 리터럴 풀기로 가능

이를 뭘을 Object Literal 풀기로
대?

② 문자열은 " "으로 풀기

③ 프로퍼티명은 문자열로 풀기

④ 허수 풀기 어렵

- 값만 풀기가 가능!
- 다른 객체 풀기가 가능
- 함수 풀기가 가능

가져온 JSON 형식으로 정의한 문자열

JSON.parse()

↑
JavaScript의
Built-in 객체

가져온
가져온 객체

JavaScript
Object
Notation



JavaScript
Object Literal
풀기

* Data 포맷 : binary vs text

Data

{ 이름: ABC
나이: 20
재직: true

→
바이트 단위로 구조화 따라
저장

① binary

이름길이	이름	나이	재직여부
00	03	41	42

↳ 이를 때 바이트 규칙에 따라 읽는다

* 파일 크기가 가장 작은 편이다

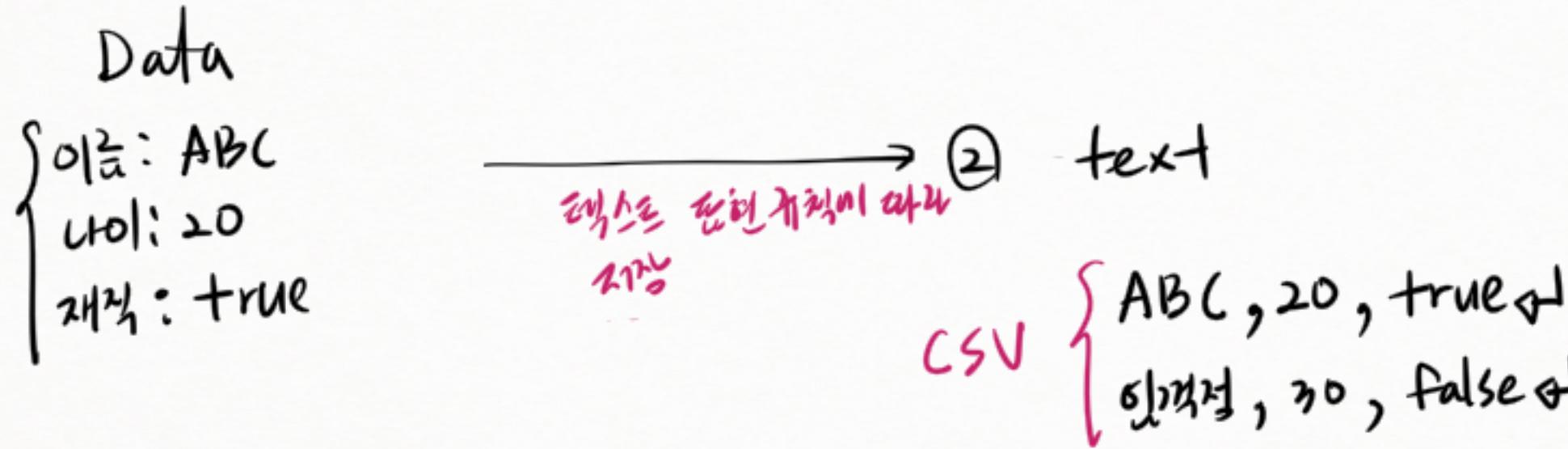
↳ 예? 데이터에 대한 목록이나 설명이 있다
(meta data)



바이트 저장 규칙을 몰라도
데이터를 차대로 읽을 수 있다

* 맥스로 편집기로 데이터를 차대로 볼 수 있다.

* Data 포맷 : binary vs text



- * 텍스트로 데이터를 쉽게 접근할 수 있다
- * binary 형식에 비해 파일 크기는 커진다.

* Text 파일 표기

① CSV

Comma-Separated Value

홍길동, 20, true ↪

임꺽정, 30, false ↪

- 간접적.

- 한 칸에 한 데이터

- 각 항목의 정보가 없다

↓
직접적으로 데이터가 흘러온다

- 제공 기관 데이터를 다루기 어렵다

② XML

extensible Markup Language

```

<members>
  <member>
    <name>홍길동</name>          ← data
    <age>20</age>                ← meta data: 태그 자체를 설명하는 태그다.
    <working>true</working>
    <schools>
      <school>
        <name>부드천고</name>
        <state>경기</state>
      </school>
      <school>
        <name>부드천고</name>
        <state>경기</state>
      </school>
    </schools>
  </member>
  :
</members>
```

- 제공 구조의 데이터 표현 가능

- 각 항목의 의미를 표현 가능
metadata

특정 항목의 항목을 찾기 쉬워

- data или metadata가
다른 수 있다.

↓
파일 크기가 커다.

* Text 파일 형식

① JSON

JavaScript Object Notation

```
[  
  { "name": "홍길동",  
    "age": 20,  
    "working": true,  
    "schools": [  
      { "name": "마르코폴로", "state": "경기" },  
      { "name": "비즈쿱", "state": "경기" }  
    ]  
  },  
  :  
]
```

- XML 보다 더 간결한
meta data
- JavaScript 와 의사
언어로의 암호화

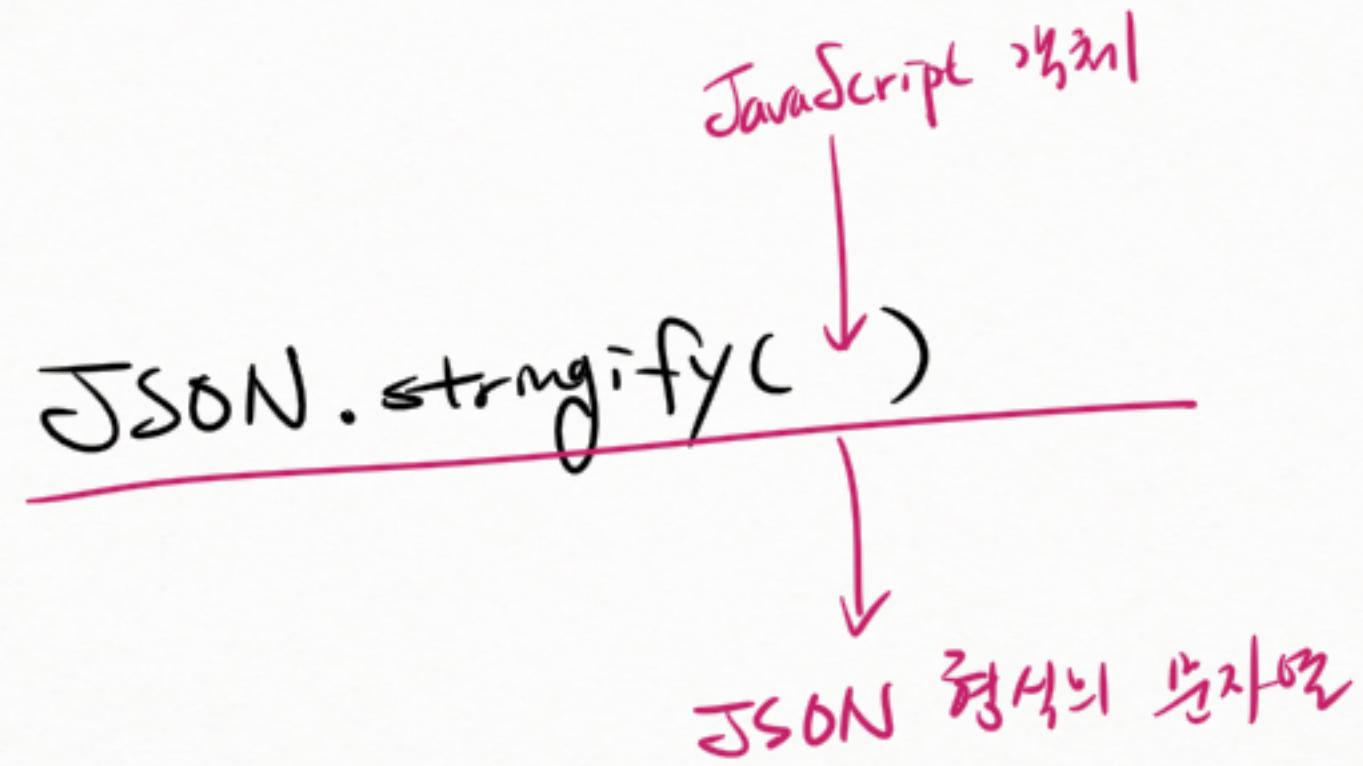
④ YAML

Yet Another Markup Language
↳ YAML ain't markup Language

```
name: 홍길동  
age: 20  
working: true  
schools:  
  - school:  
    name: 마르코폴로  
    state: 경기  
  - school:  
    name: 비즈쿱  
    state: 경기
```

- JSON 보다 더 간결
- 들여쓰기 (indent)로
세이프 구조를 표현

* JSON.stringify()



211 211

* 12월 105회 ~ 드로잉 106회

① Képalkotásokhoz gyűjtemények (pl: Java, C++, ...)

```
class Student {  
    String name;  
    int age;  
    boolean working;  
}
```

Diagram illustrating static type binding:

```

graph TD
    A["(all: Java, C++, ...)"] -- "Static type binding" --> B["Student obj = new Student();"]
    B -- "Object creation" --> C["obj"]
    C -- "Value" --> D["200"]
    B -- "Object creation" --> E["name: '홍길동'"]
    B -- "Object creation" --> F["age: 20"]
    B -- "Object creation" --> G["working: true"]

```

The diagram shows the creation of a `Student` object named `obj`. The variable `obj` is assigned the value `200`, which is highlighted in red. Below the variable, there is a box containing the value `200`, also highlighted in red. To the right of the assignment, there is a pink bracket grouping `name`, `age`, and `working`, with the label `Object creation` above it. Below this bracket, there are three separate boxes: one for `name` containing `"홍길동"`, one for `age` containing `20`, and one for `working` containing `true`. Each of these three boxes is also highlighted in red.

```
obj.name = "3218"
```

ob). age = 20;

6b). Working = true

obj. tel ~~= "02-111-2222"~~

Comprile $\ell_{\eta}^{\pm} = m_{\ell_{\eta}^{\pm}}$

* JavaScript 와 프로토타이핑 비교

② 프로토타이핑 (prototyping) 방법으로 객체 만들기

함수로 접근!

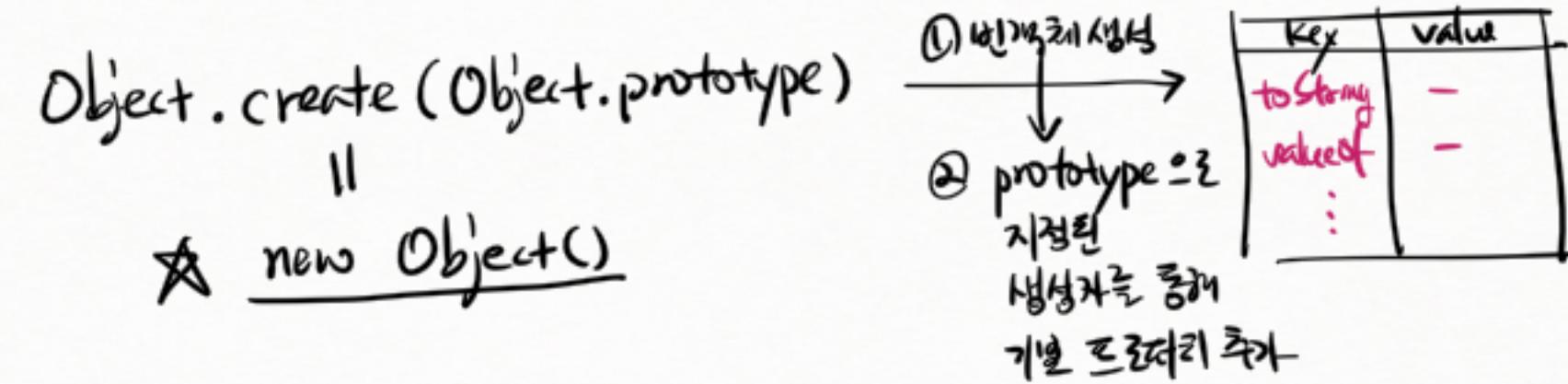
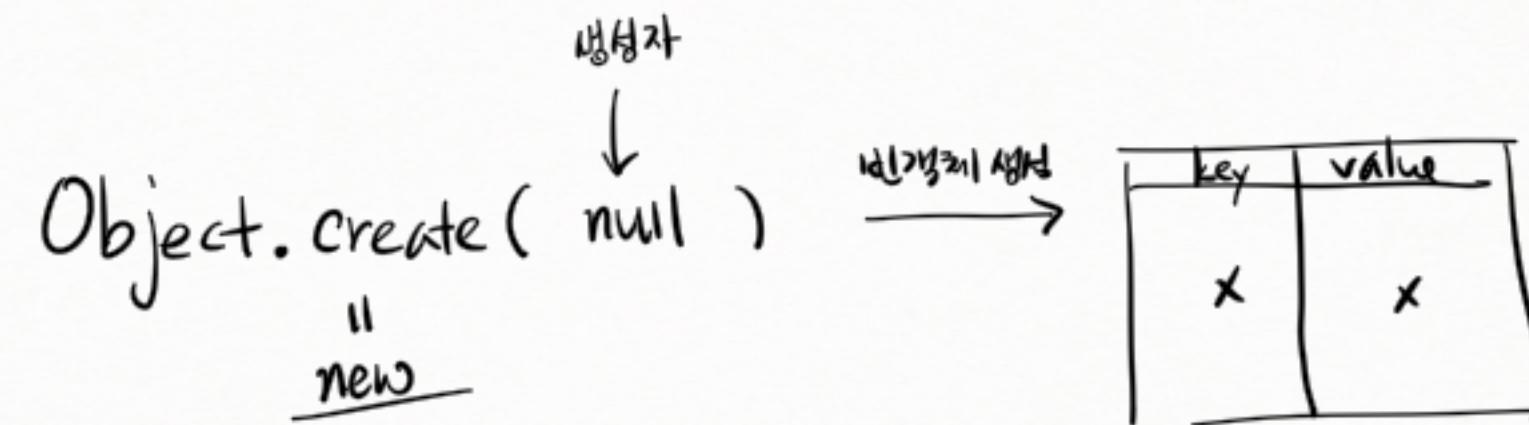


```
var obj = new Object();
```

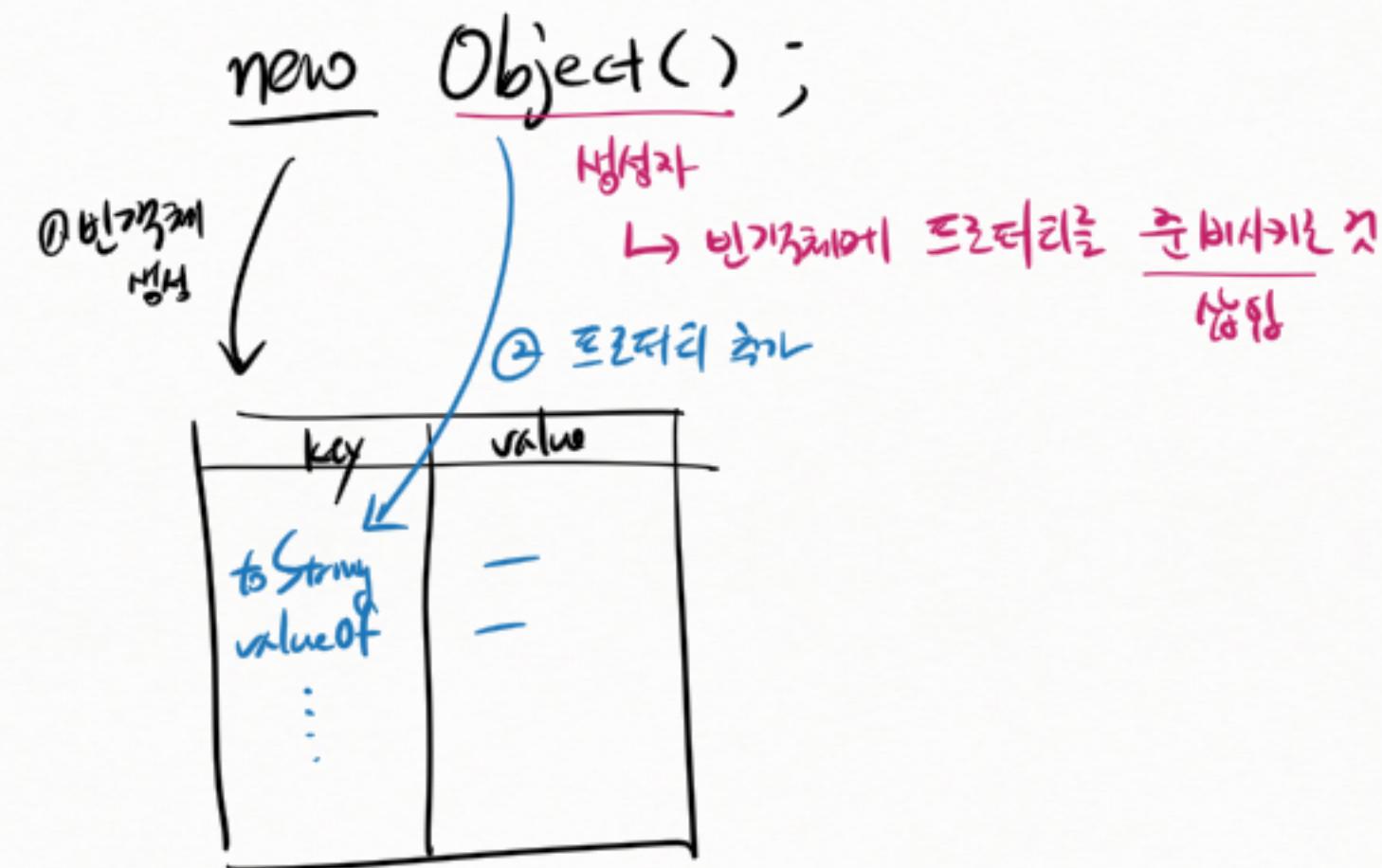
① 비주얼 형태	② 기본 프로퍼리 추가								
<pre>obj.name = "홍길동"; obj.age = 20; obj.working = true;</pre>	<p>기본 프로퍼리 추가 할 때마다 빌드 할 때마다 업데이트 된다.</p> <table border="1"><thead><tr><th>key</th><th>value</th></tr></thead><tbody><tr><td>name</td><td>"홍길동"</td></tr><tr><td>age</td><td>20</td></tr><tr><td>working</td><td>true</td></tr></tbody></table>	key	value	name	"홍길동"	age	20	working	true
key	value								
name	"홍길동"								
age	20								
working	true								

```
obj.name = "홍길동";  
obj.age = 20;  
obj.working = true;
```

* 객체 생성



* 생성자 (constructor)



* hasOwnProperty ("프로퍼티인가")

↓
기존에 추가시킨 프로퍼티인가 검사

var obj = new Object()

obj.hasOwnProperty("toString") → false
" " ("valueOf") → false

obj.plus3 = () => {};
↑
값이 아니

obj.title = " — ";
obj["content"] = " — ";
obj['viewCount'] = " — ";
obj.plus1 = f1;
obj.plus2 = function() { — }
↑
값이 아니

key	value
toString	-
valueOf	-
:	-
title	-
content	-
viewCount	-
plus1	-
plus2	-
plus3	-

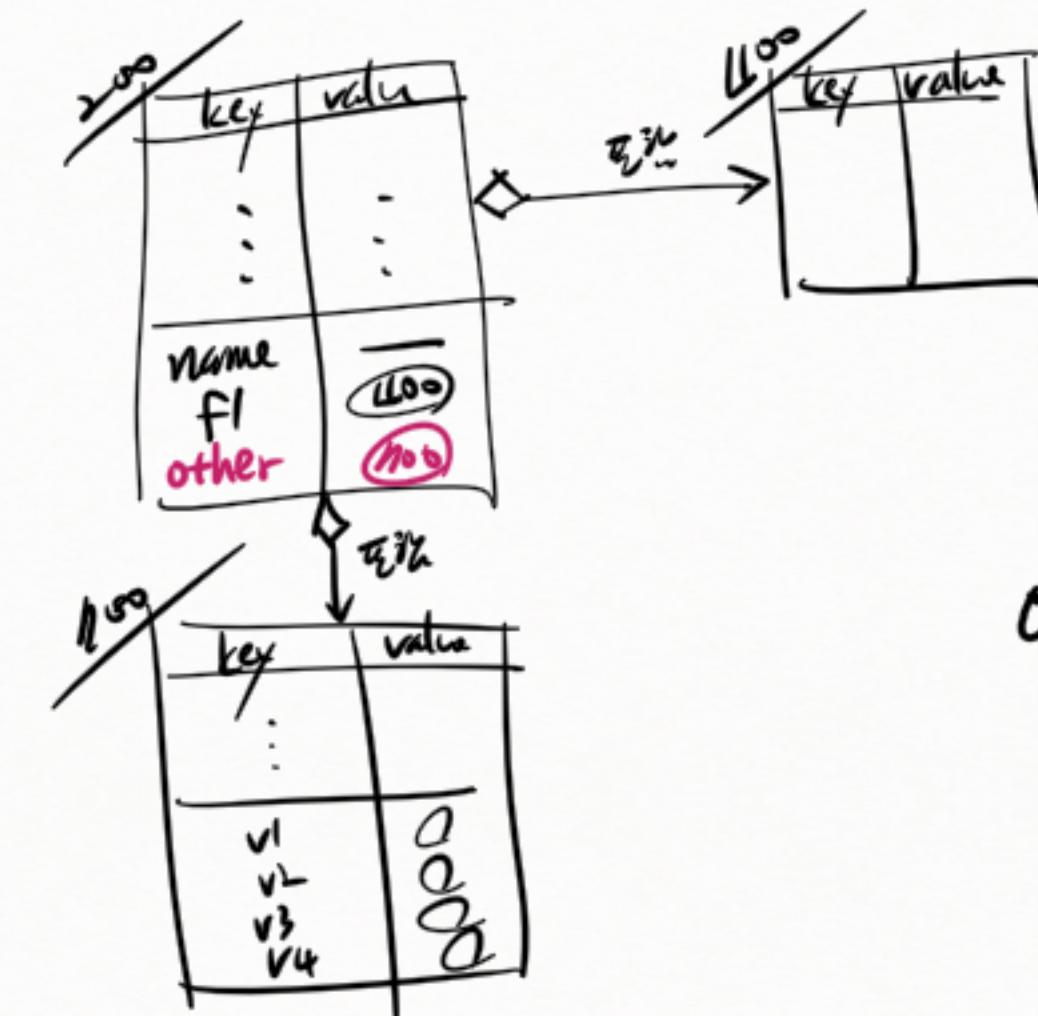
* گیگانی اتے چھپاں کیوں نہیں

let obj = new Object();
100

obj.name = "—";
obj.fl = () => {};

let obj2 = new Object();

obj2.v1 = 0;
obj2.v2 = 0;
obj2.v3 = 0;
obj2.v4 = 0;



obj.other = obj2;
100
ئىچىكى

* think of this

100

K	V
name	-
kor	-
eng	-
math	-
toString	200

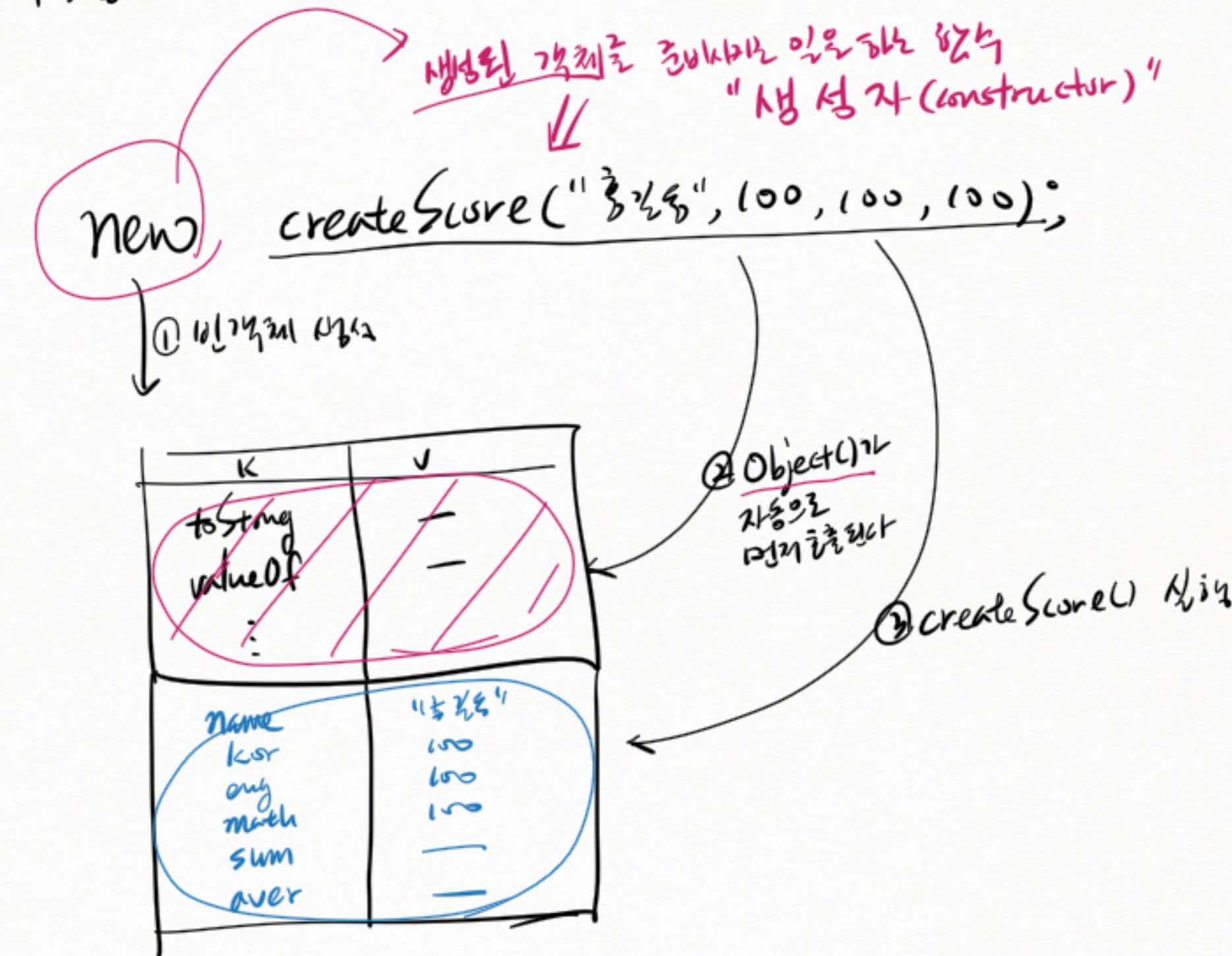
100

K	V
100	100

obj.toString = function() {
 return this.name + ...;
}

100 → return this.name + ...;

* new 484



* Higher prototype

`<생성자>`
Score()

prototype

K	V
sum	<code>f(){};</code>
aver	<code>f(){};</code>

Score()가 더 높은 prototype!
즉, 더 높은 prototype
보다 더 높은 prototype!

`new`

K	V
name	-
kor	-
eng	-
math	-

K	V
name	-
kor	-
eng	-
math	-

K	V
name	-
kor	-
eng	-
math	-

`var scores = [200, 300, 400]`

`scores[0].sum();`

`call`

`Score.prototype.sum()`

`scores[0].aver();`

`call`

`Score.prototype.aver()`

`scores[2].sum();`

`call`

`Score.prototype.sum()`

* 생성자와 일상기능

function f() {}

var obj = new f();
 ↑
 new가 생성자임을 알기

① new ~~250~~ $\xrightarrow{\text{Object}}$ K V
 ↓
② Object $\xrightarrow{\text{toString}}$ toString value
 ↓
③ f() $\xrightarrow{\cdot}$. .

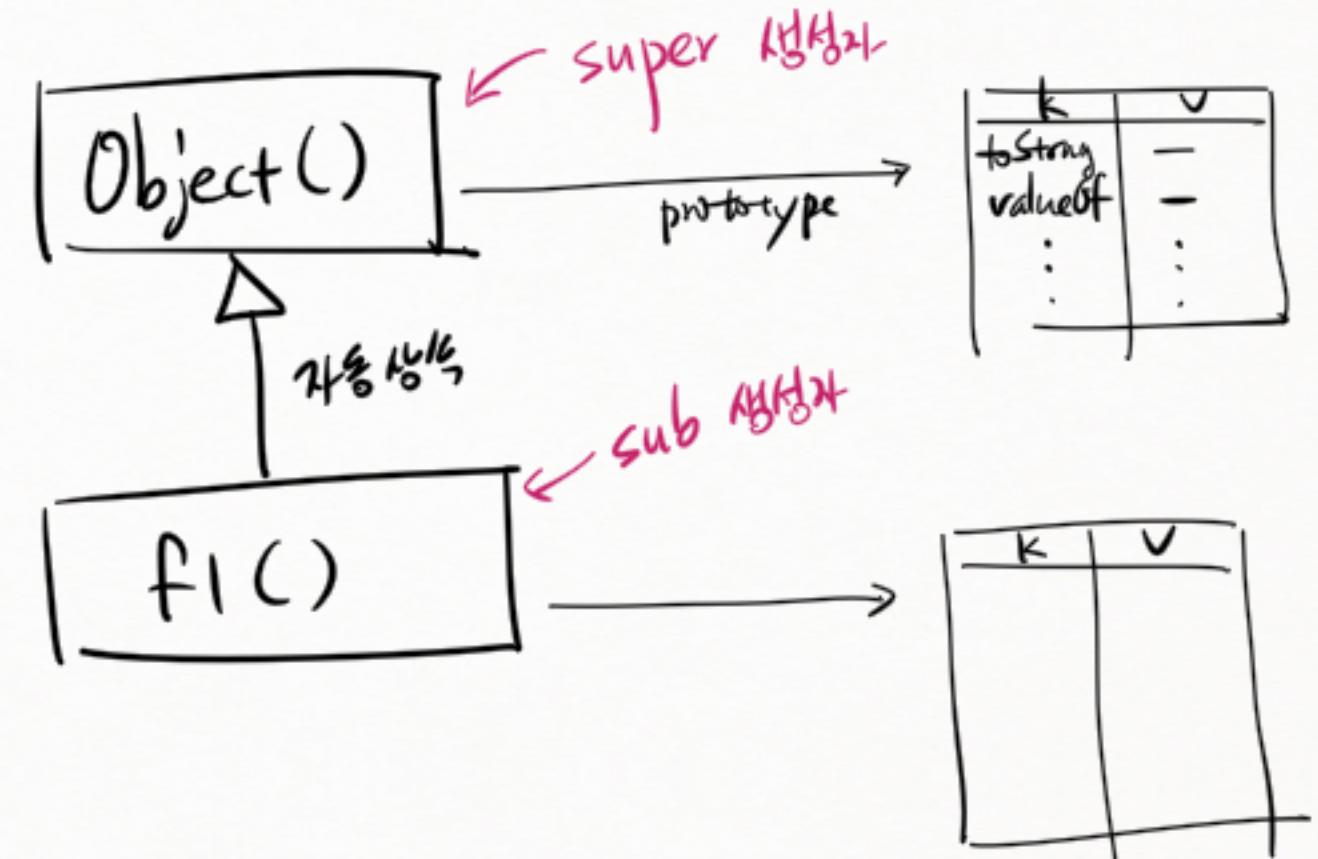
K	V
toString	—
value	—
:	:
.	.

var obj = f();
 ↑
 undefined

* 생성자와 Object()

`var obj = new f1();`

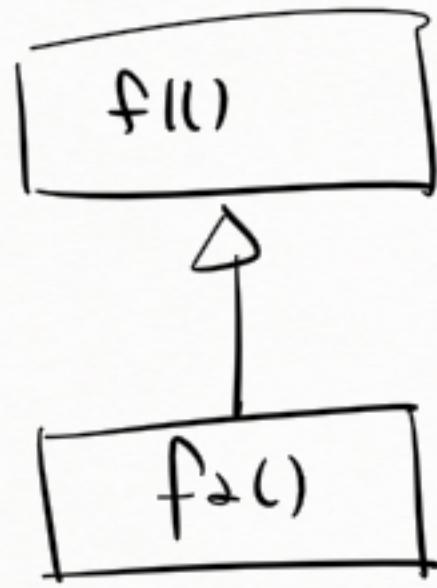
- ① new: 빈 객체 생성
- ② f1()의 super 생성자 호출
↳ Object()
- ③ f1() 호출



`obj.toString()`

- ① 기본적인 출력.
- ② 생성자 `f1.prototype`에서 출력.
- ③ `super` 생성자 `Object.prototype`에서 출력.

* 생성자를 상속하기



```

f1(n) {
    this.name = n;
}
  
```

f2(n, k, e, m) ?

f1(n); ← 일반 함수 호출방법으로는 f2()가 this 라는

f1.call(this, n);
this.kor = k;
this.eng = e;
this.math = m;

}

변수에 받았을 때까지 주소를
f1()에 전달할 때마다

var obj = 제작자 주소 전달

new f2("홍길동", 100, 90, 80);

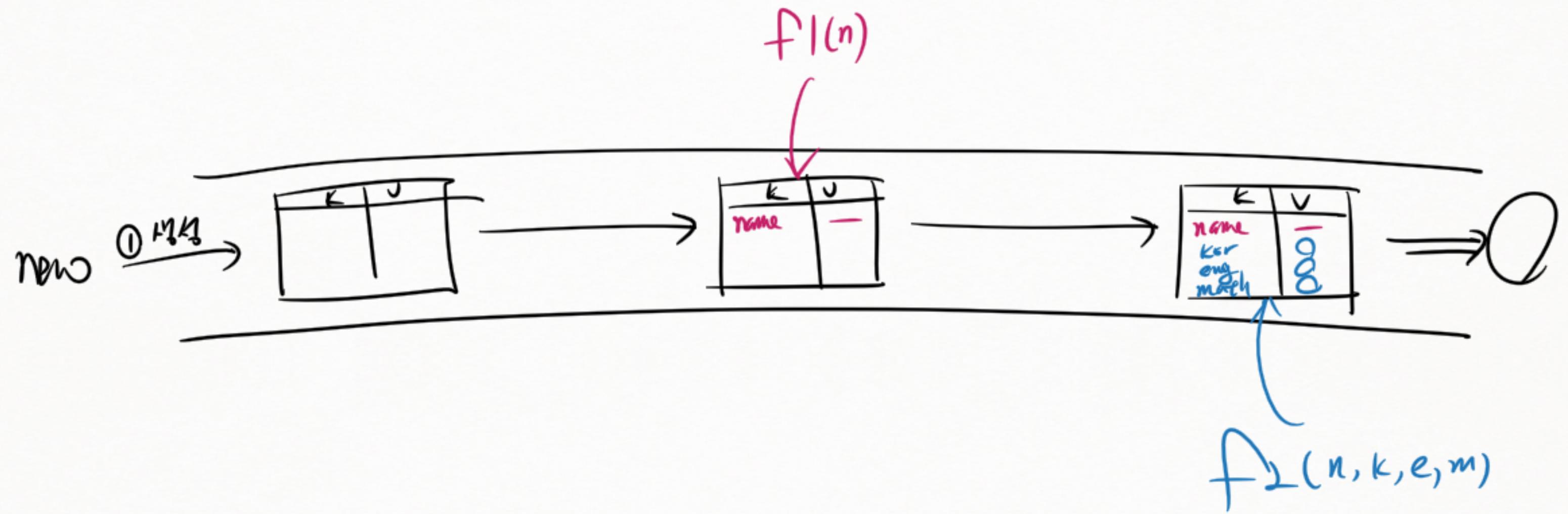
<u>200</u>	K	V

* 함수를 호출하는 방법

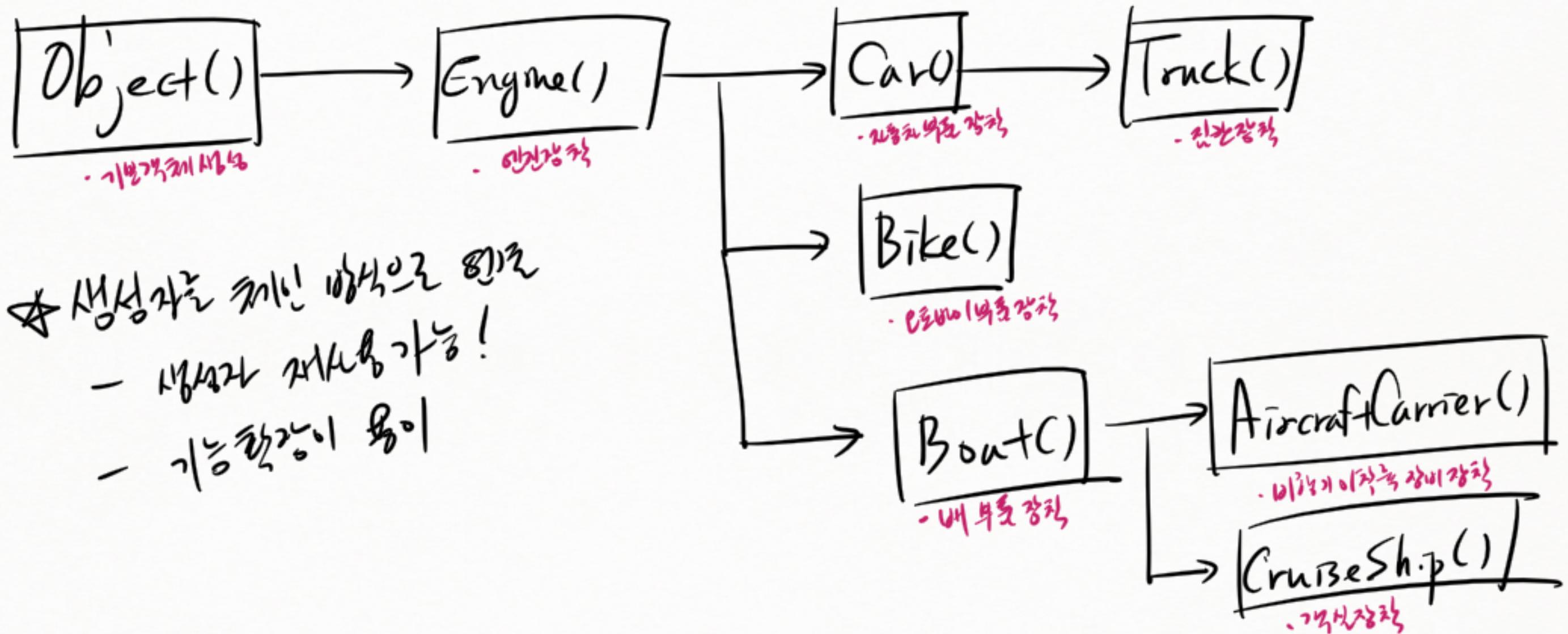
① 함수명(아구먼트, ...);
예) f1();

② 함수명.call(개체주소, 아구먼트, ...);
예) f1.call();

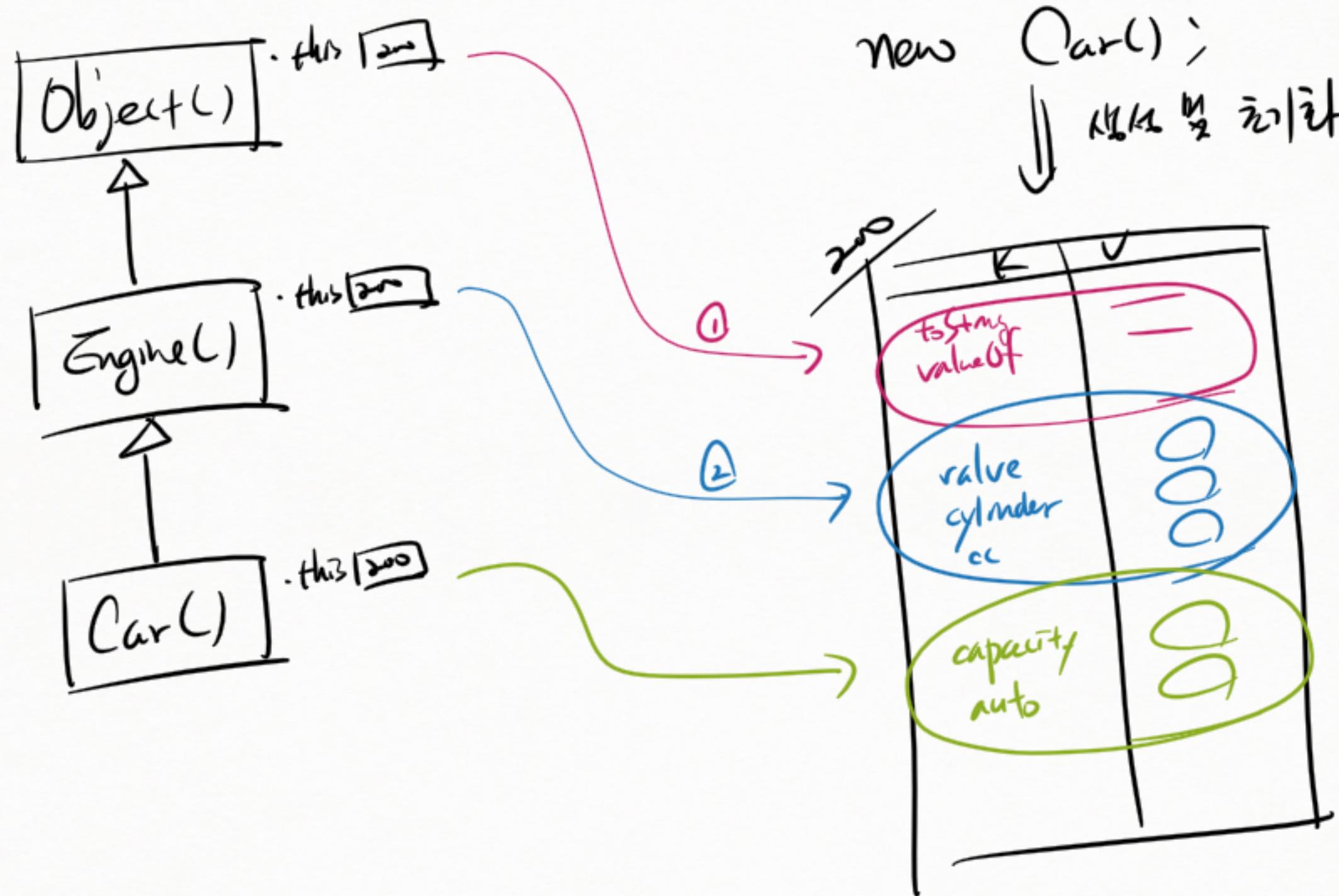
* 생성자와 전파하여 배운



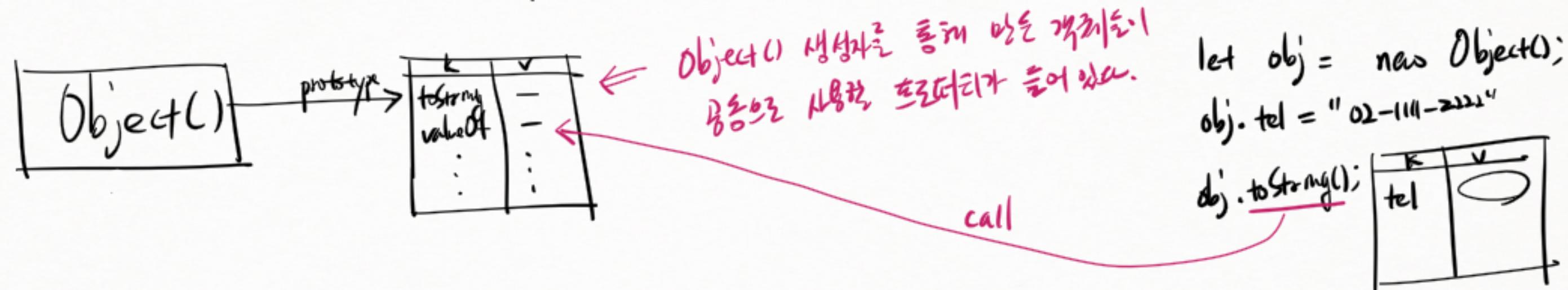
* 생성자를 차인으로 초기화되어 super-sub로 만드는 이유?



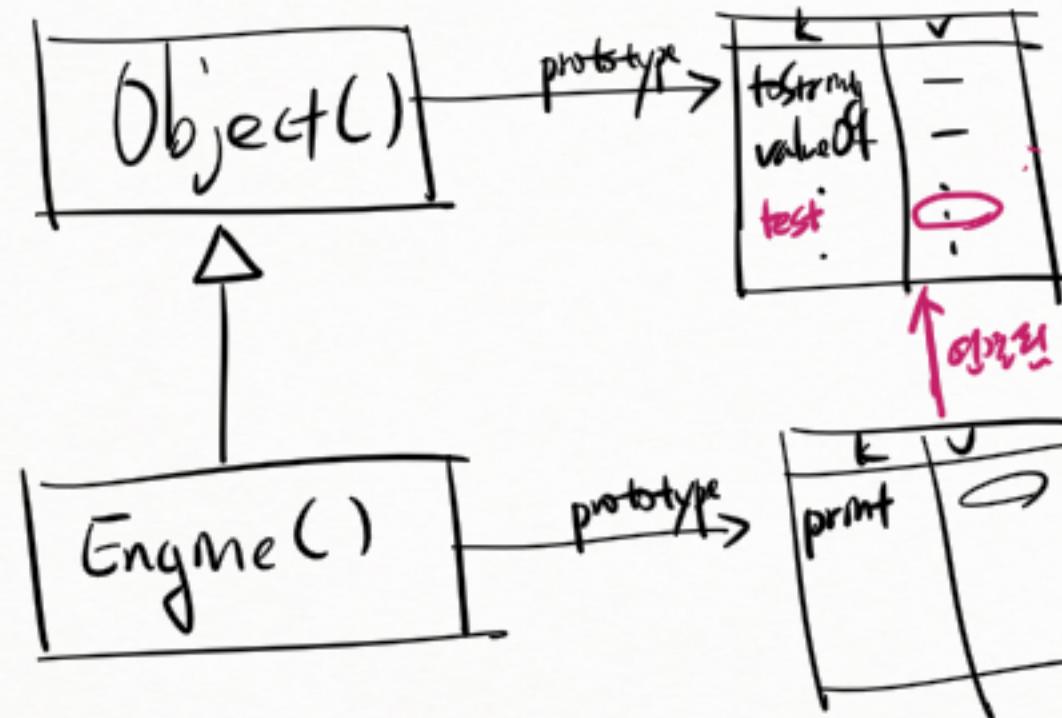
* 생성자 체인 예:



* 자식 생성자의 prototype 사용하기



* 자식 생성자의 prototype 사용하기



Engine() 생성자는 Object의
기반으로 만들어져 있으므로
Object.prototype에 있는
함수를 사용할 수 있다.

`let el = new Engine(...)`

K	V
value	0
cylinder	0
cc	0

`el. test();`

↳ `Object.prototype.test()`

`el. toString();`

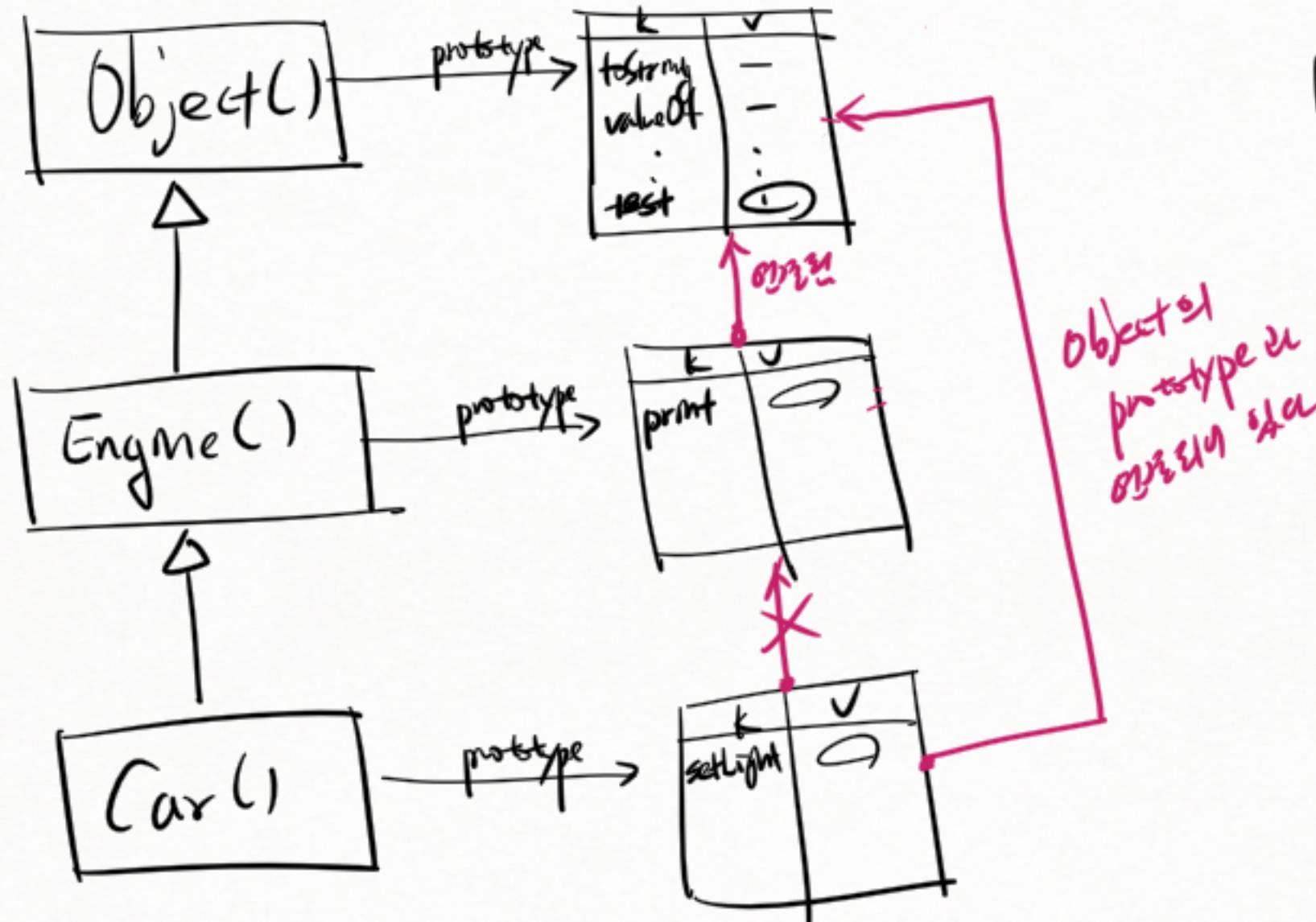
↳ `Object.prototype.toString()`

- ① Object.prototype.toString()는 `el`의
- ② `Object.prototype.toString()`은 `el`의
- ③ `Object.prototype.toString()`은 `el`의

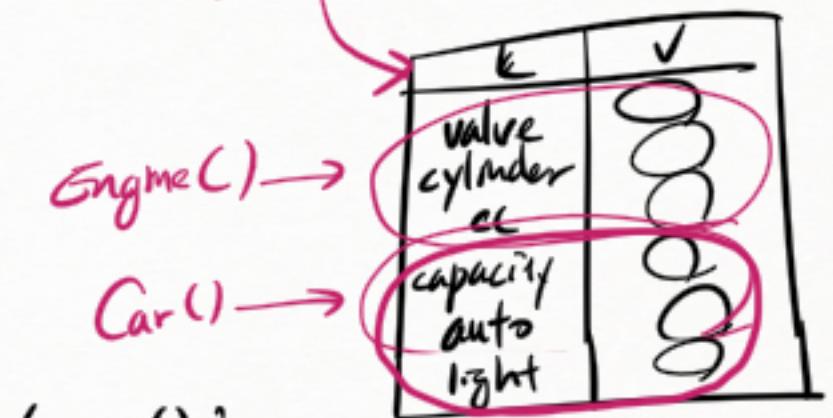
`el. print();`

↳ `Engine.prototype.print()`

* 자식 생성자의 prototype 활용



`let c1 = new Car(...);`



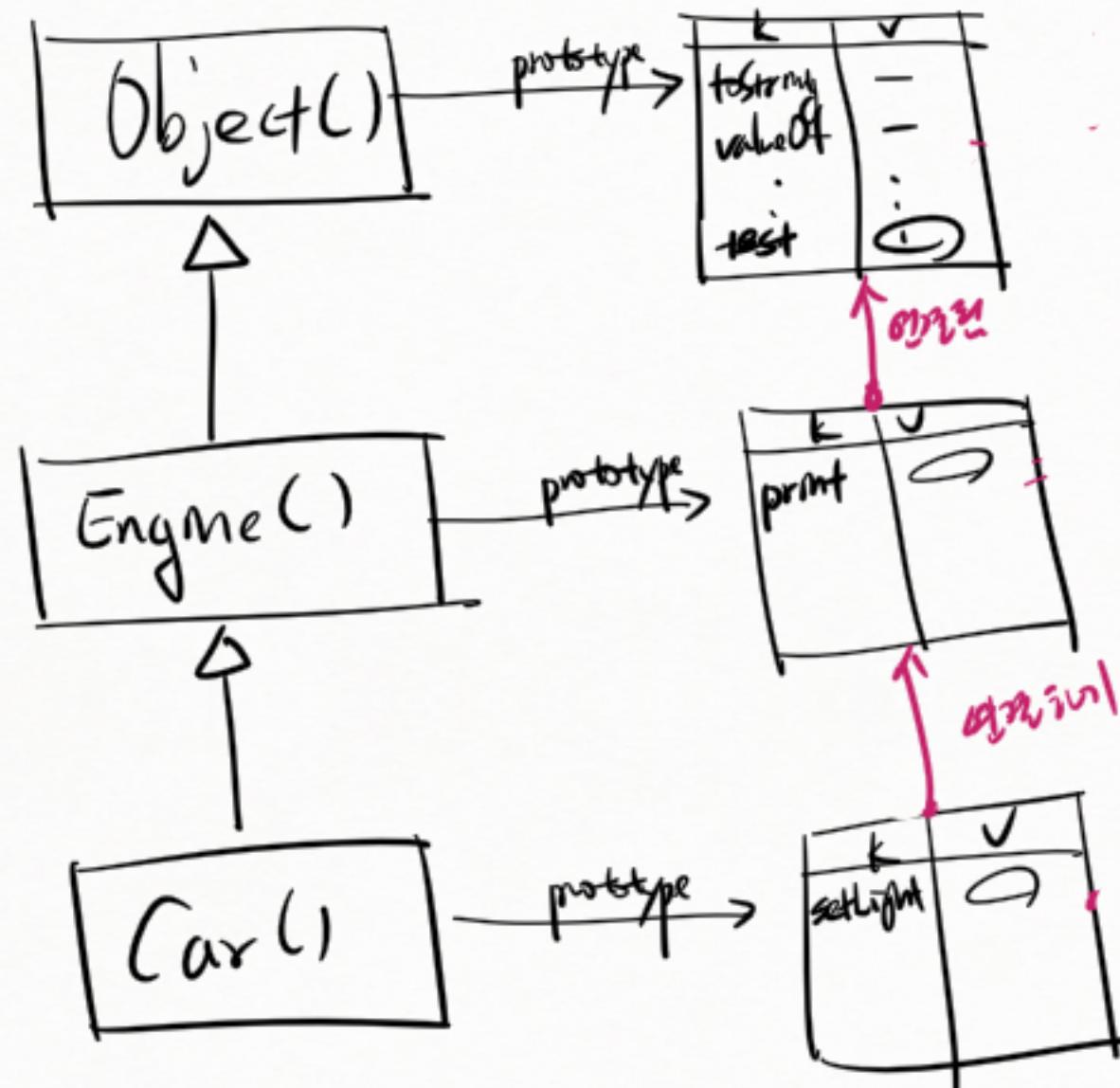
`c1.toString();`
↳ Object.prototype.toString()

`c1.setLight();`
↳ Car.prototype.setLight()

`c1.print();`
↳ ~~Engine.prototype.print()~~

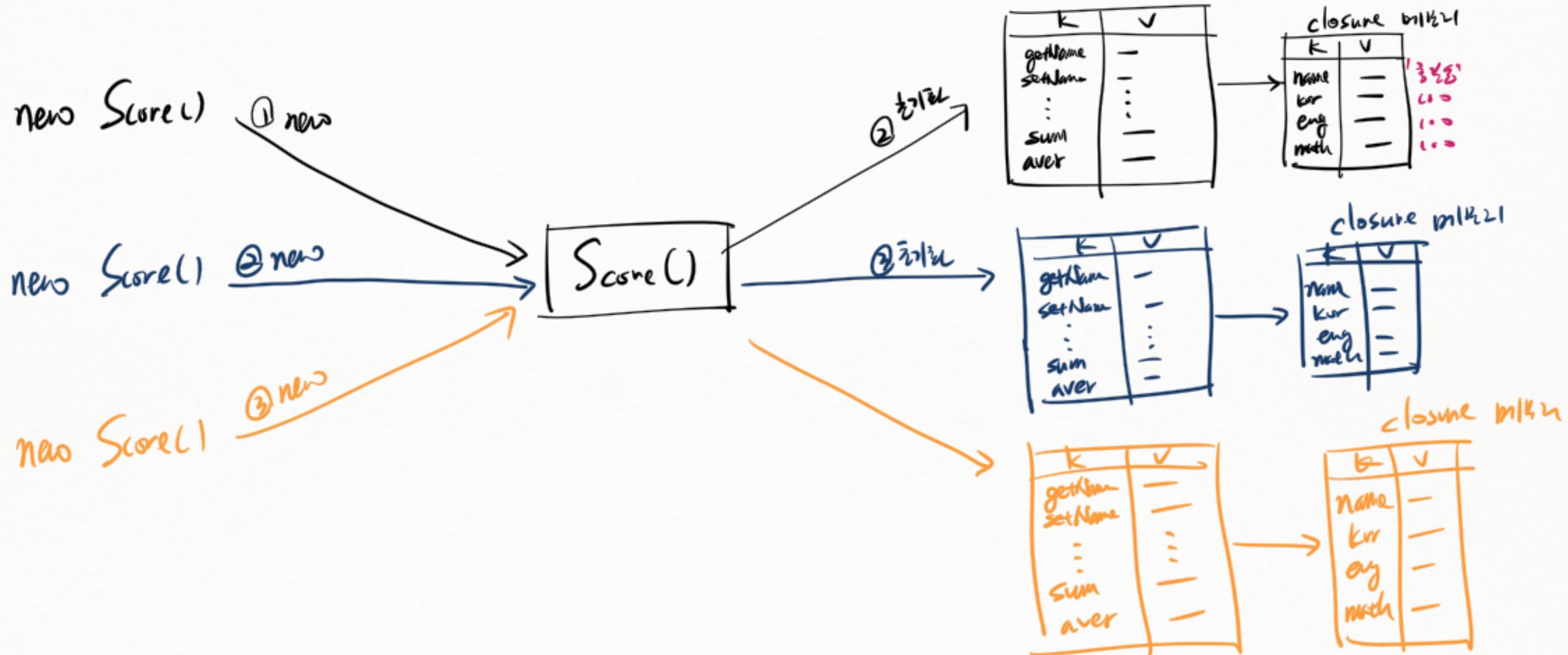
Car() 생성자 허무|무기
Engine.prototype이 연결되어 있지 않음

* 자바스크립트의 prototype 개념



Object.setPrototypeOf(
Car.prototype,
Engine.prototype);
Object.setPrototypeOf(
Engine.prototype,
Object.prototype);

* closure et local var obj



* 함수 프로퍼티와 prototype 프로퍼티

★ 특정 객체에 대해 작업하지 않고 함수를 끌어올 때는

객체에 바로 저장한다.

(값수객체)

random()

min()

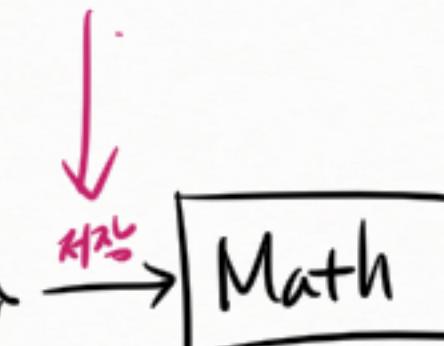
max()

sin()

round()

tan()

:



↓ 사용법

Math.random()

Math.min()

:

↑
값수객체

값수객체

★ 특정 객체의 값을 사용해서

작업을 수행하는 함수를 끌어올 때는

생성자의 prototype에 저장한다.



↓ 사용법

S1. sum()

S2. sum()

:

↑
값수객체

값수객체
기
대상값객체에 접근
하지 않는다

* 함수의 속성

① 기본적인 접근 허용

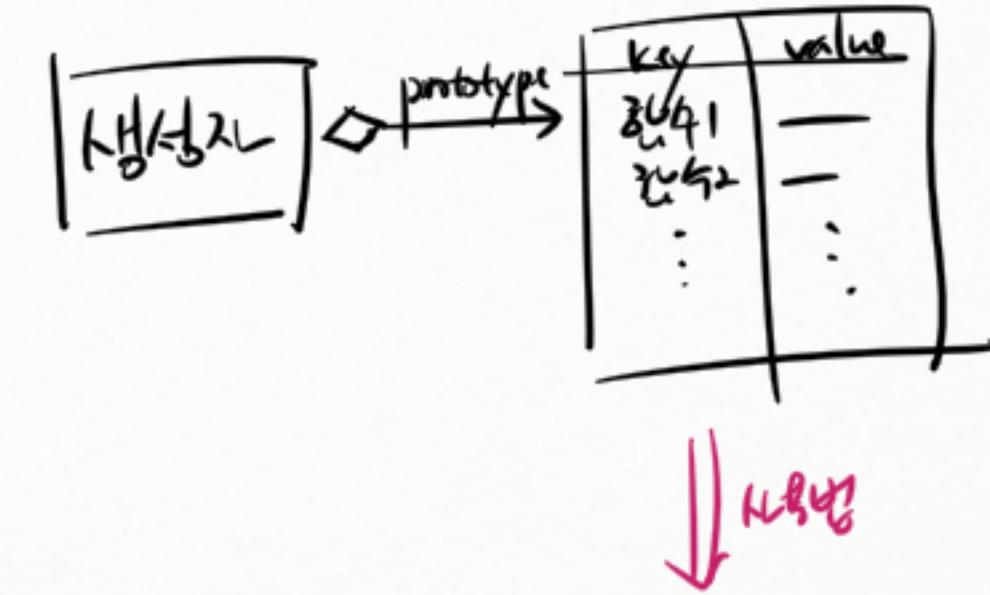
~~var m1 = new Math();
m1.random();~~

key	value
값1	-
값2	-
:	:

기본 접근

값1
값2
①) Math.random();
JSON.parse();

② 생성자의 prototype 속성



var obj = new 생성자();
obj.random();

↑
값1과 같은 객체를 갖도록 초기화

①) s1.sum() str1.split(",")
s2.sum() str2.split(",")
s3.sum() arr1.forEach()

* 생성자와 인스턴스 (instance)

↳ 실제적인 예

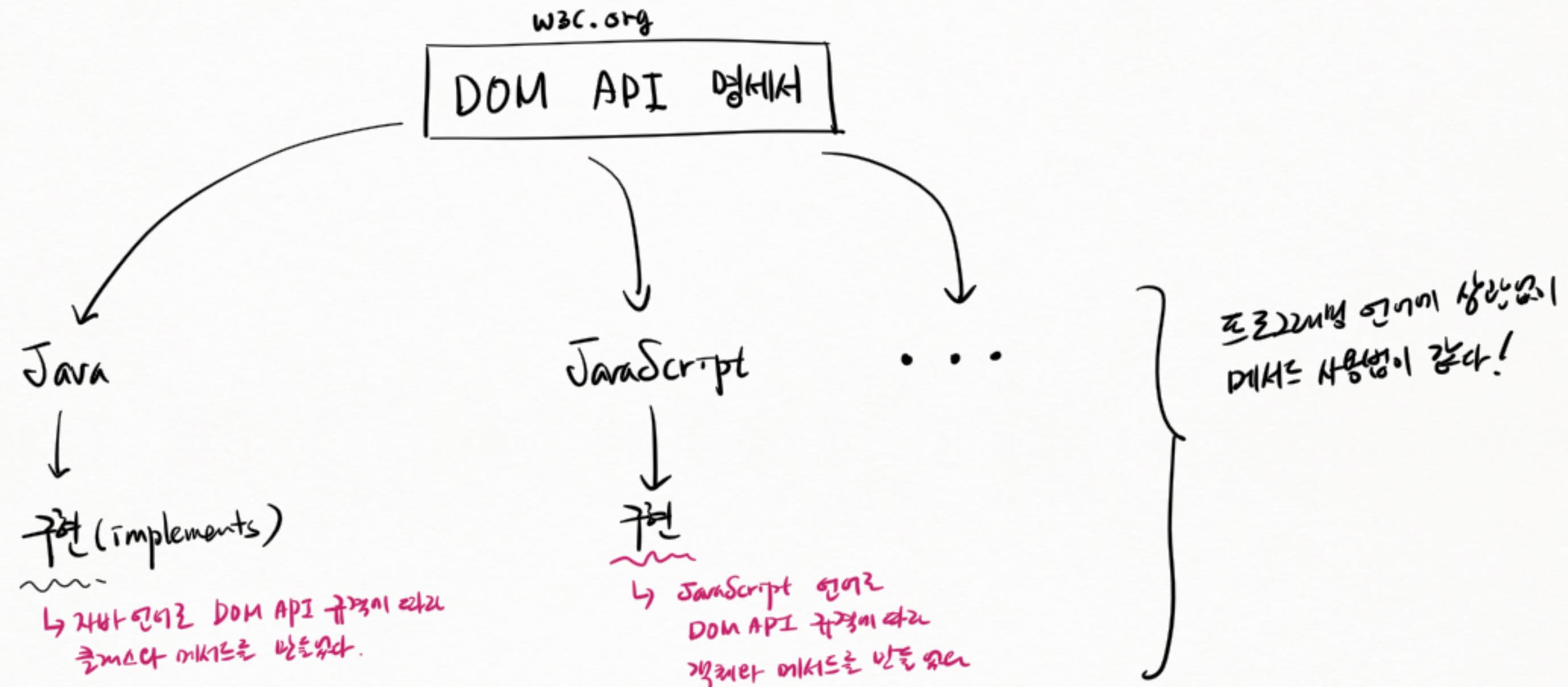


"Score의 인스턴스"
↓
Score() 생성자를 통한
기본인 초기화

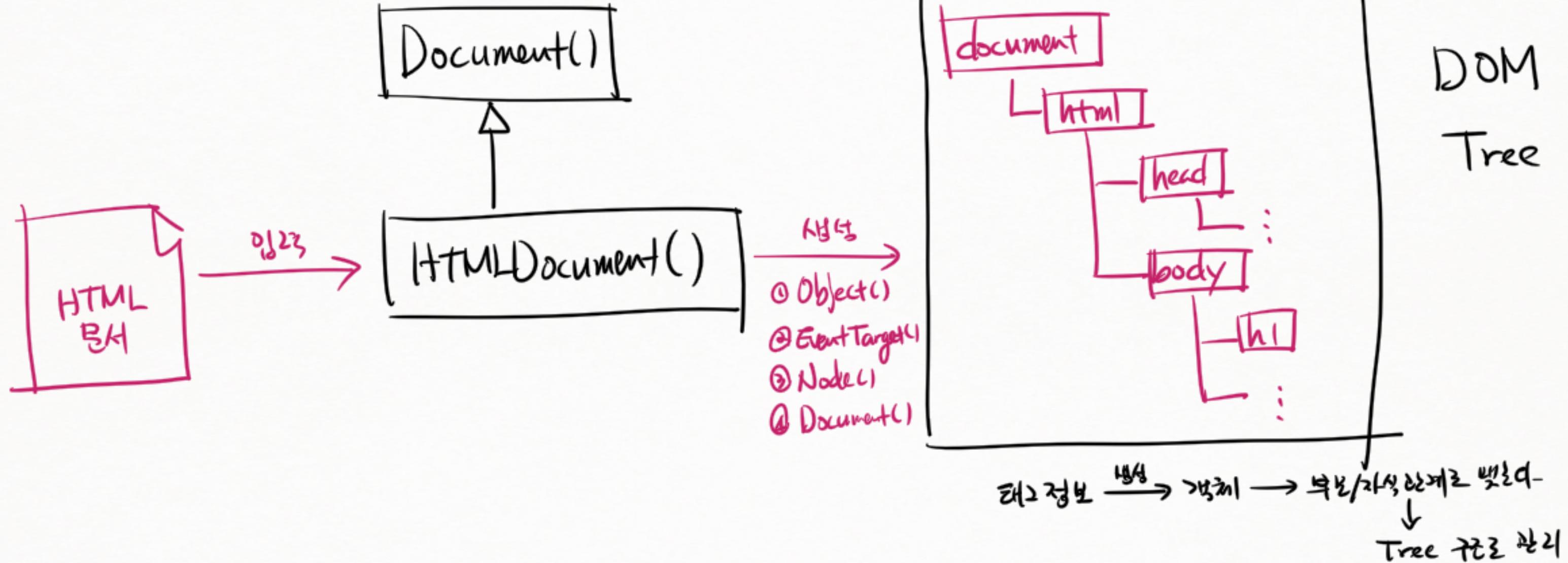
* 자바
"Score 클래스(클래스)의 기본
생성인 초기화"

DOM API

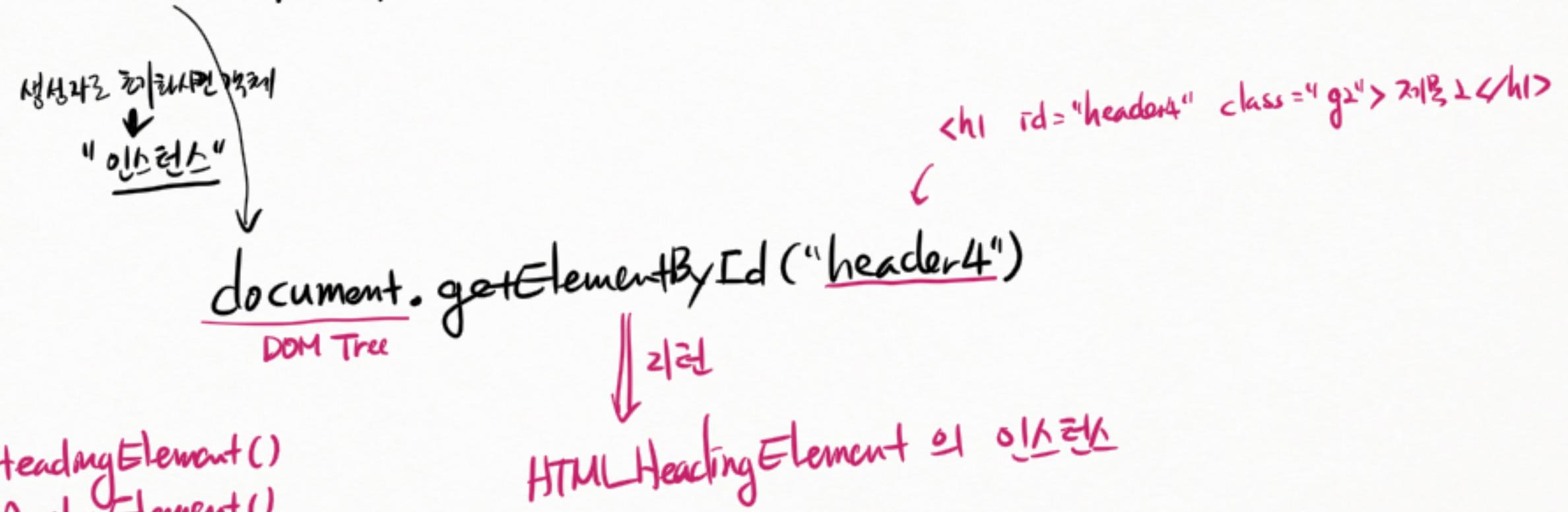
* DOM API



* HTMLDocument (Document()) et document



* Document.prototype.getElementById()



`<h1>` → `HTMLHeadingElement()`

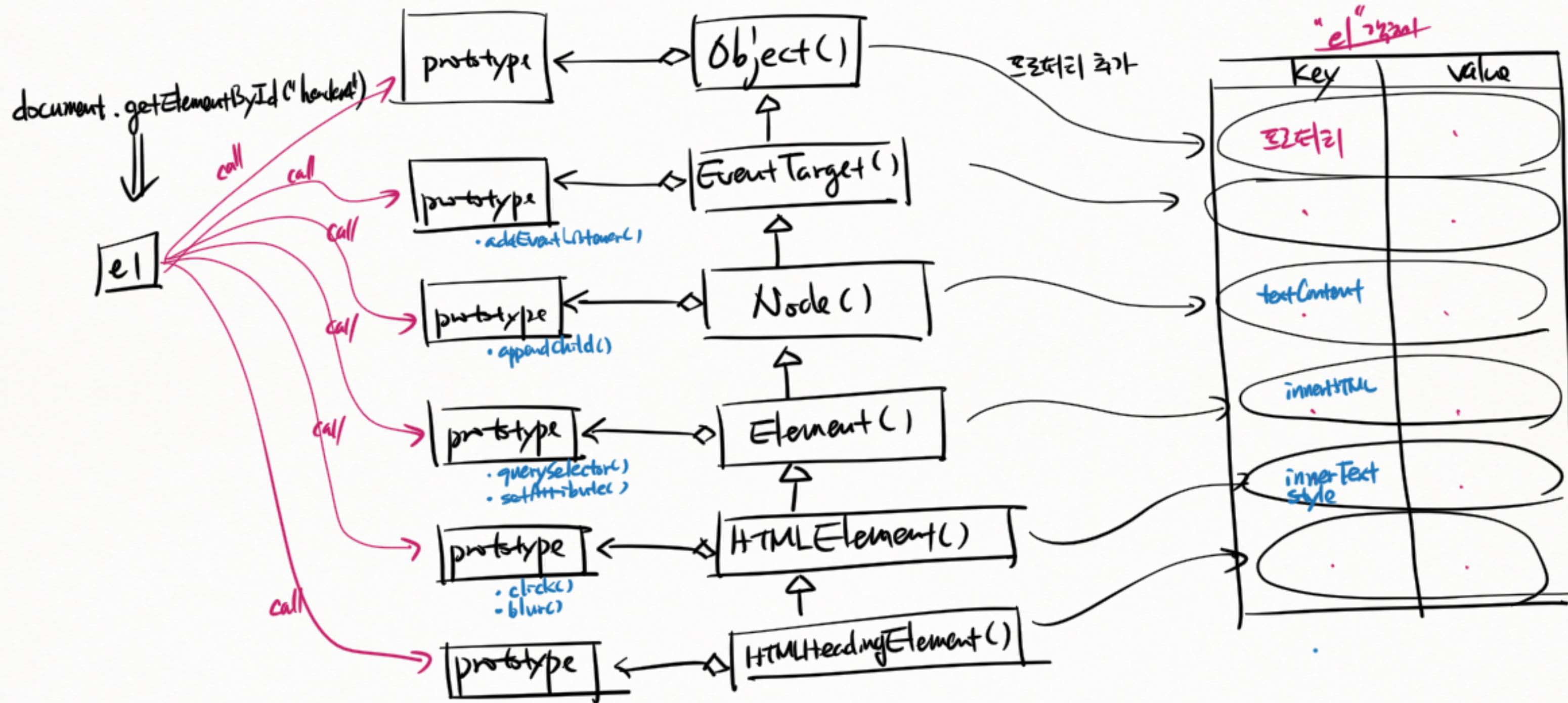
`<a>` → `HTMLAnchorElement()`

`<p>` → `HTMLParagraphElement()`

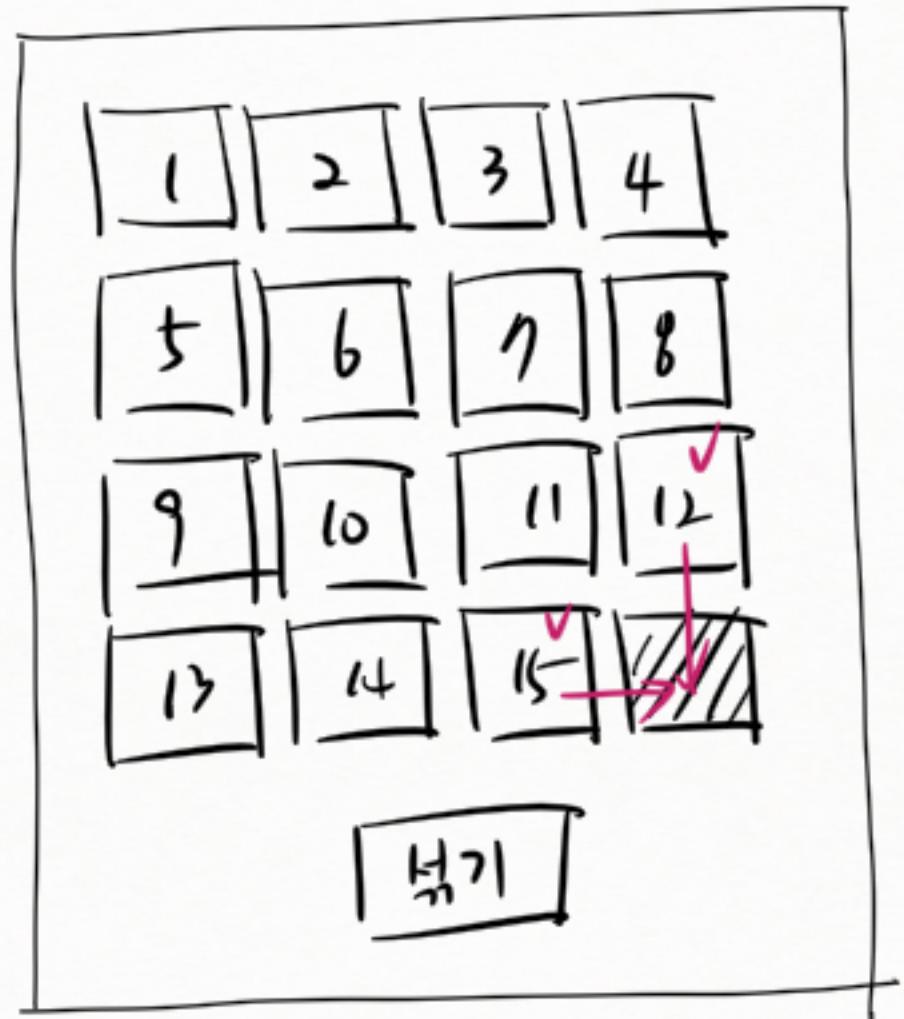
:

:

* HTMLHeadingElement() 생성자 초기화의 과정



* 15 퍼즐 만들기

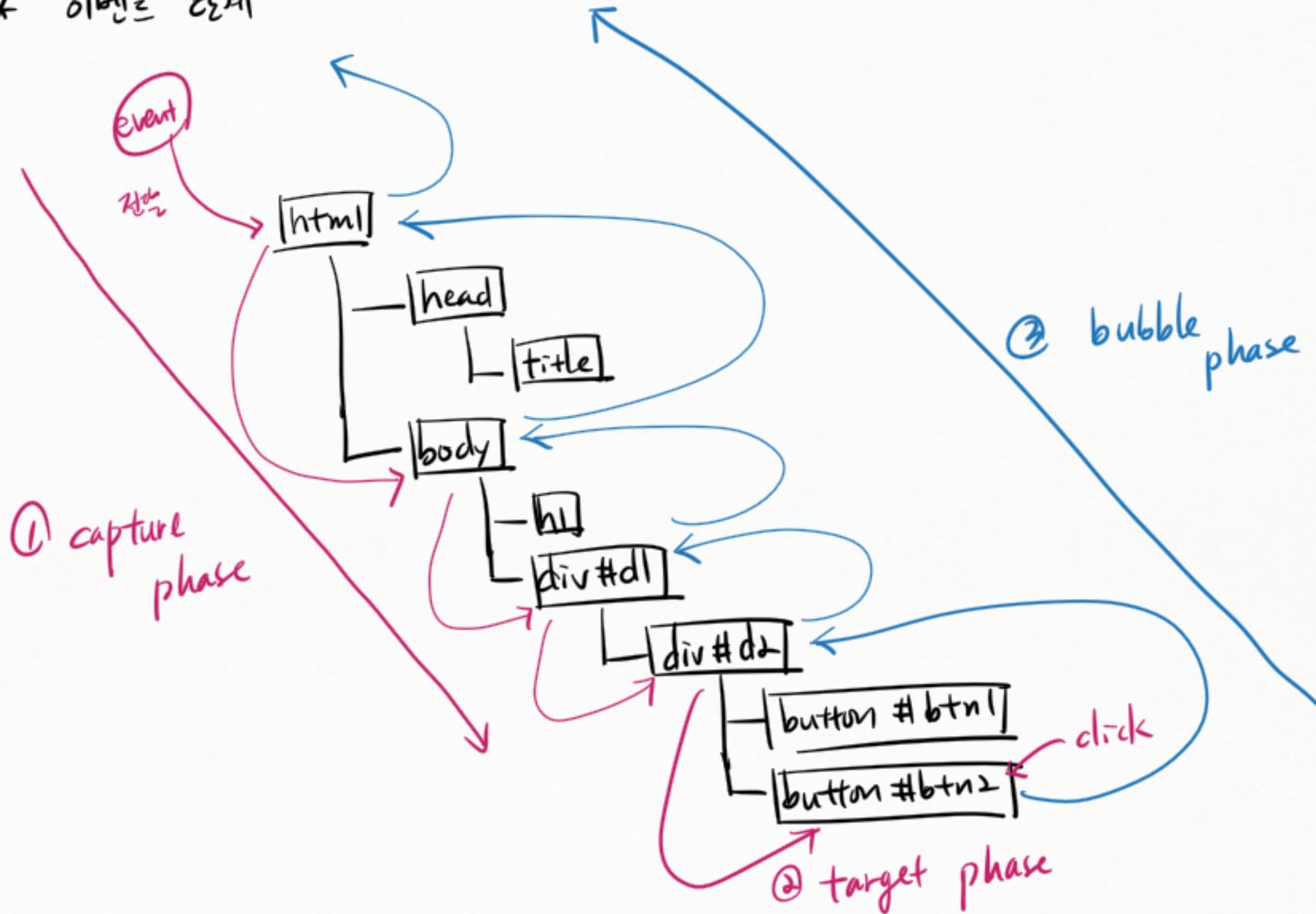


제출일 : 2022-12-15 09:00
(한국)

Event $c|\frac{2}{7}>|$

* 이벤트 단계

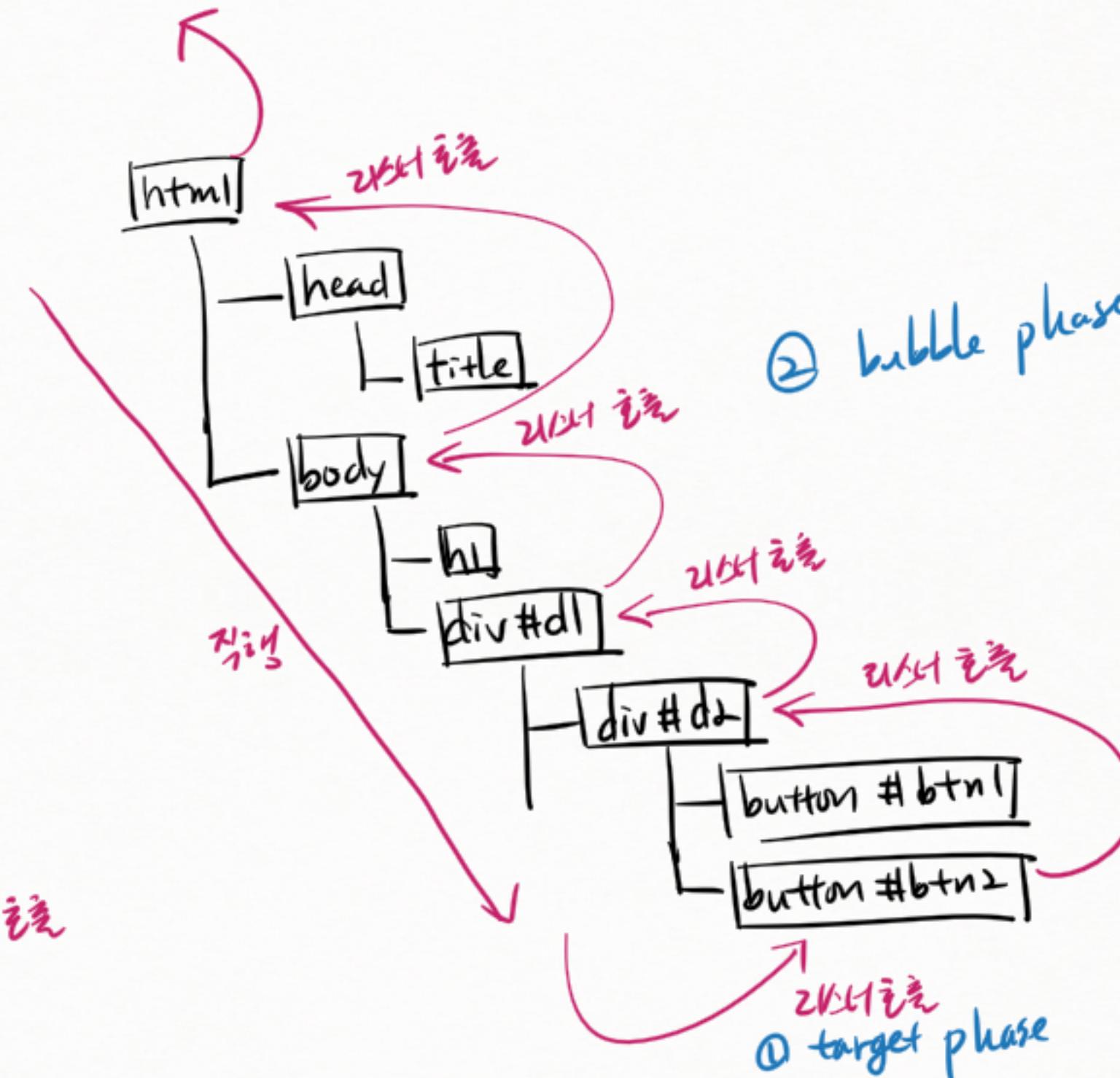
- event 1단계
↓
① capture phase 2단계
↓
② target phase 3단계
↓
③ bubble phase 4단계



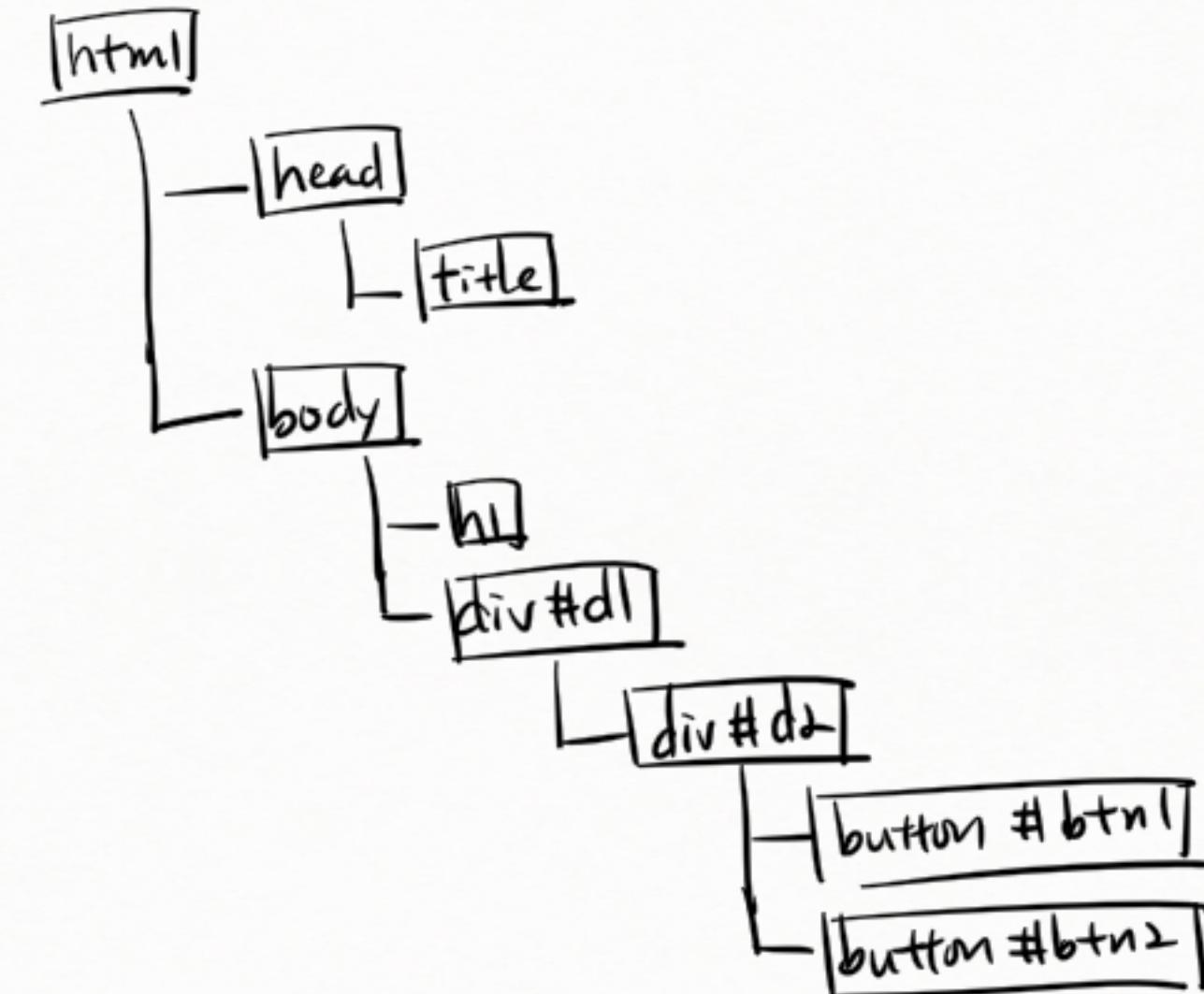
* 이벤트 단계와 리스너 흐름

```
{  
    el.onclick = function() {};  
    el.addEventListener(  
        "click",  
        function() {});  
  
    el.addEventListener(  
        "click",  
        function() {},  
        false);
```

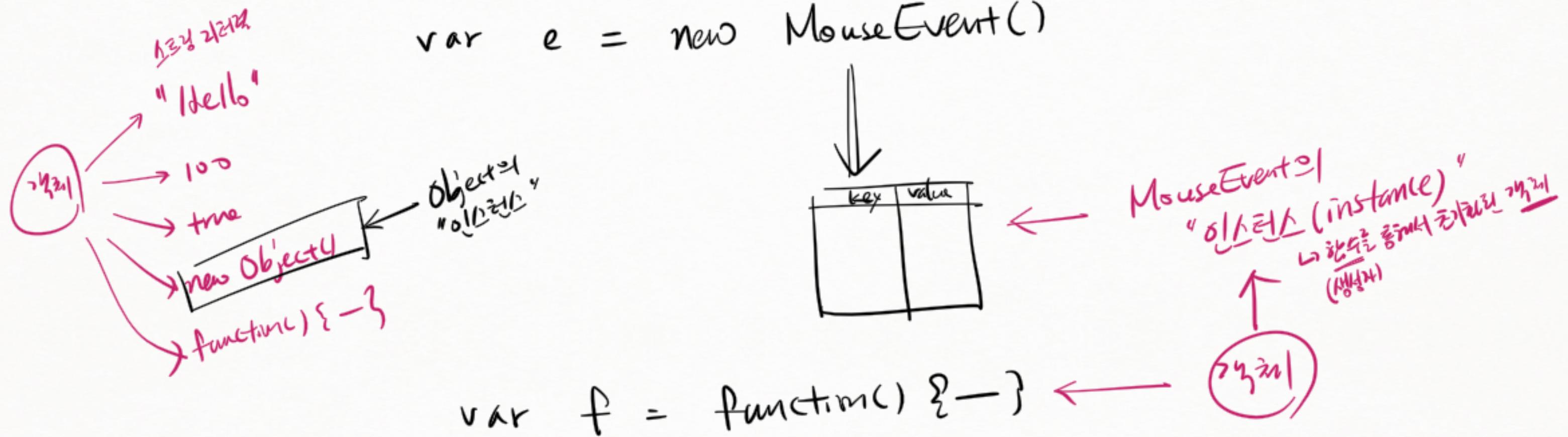
↳ event 흐름
→ ~~구조화하기~~
→ ~~리스너에 따른 흐름~~
→ 버블링하기 "



* 이번주 단계



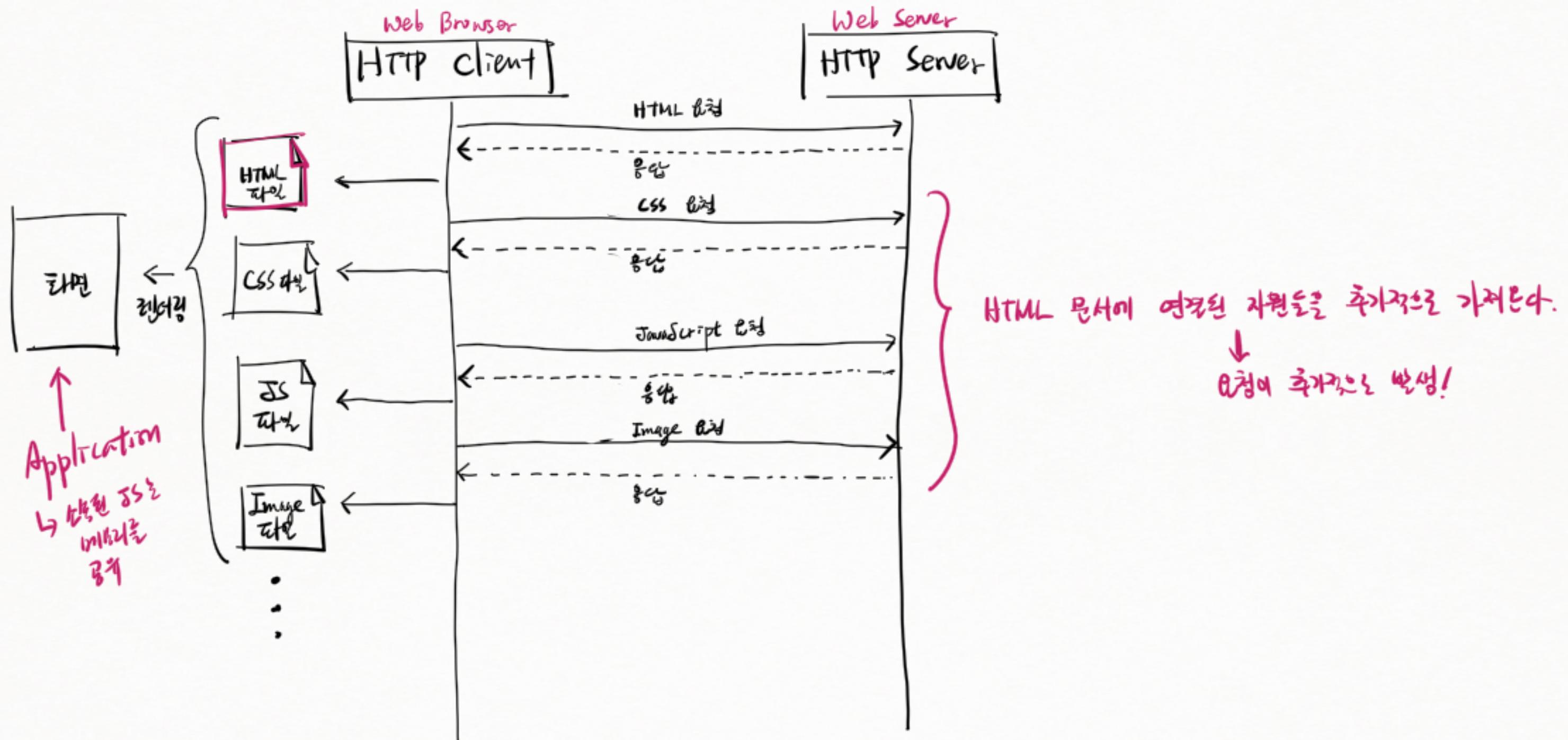
* 개체와 인스턴스



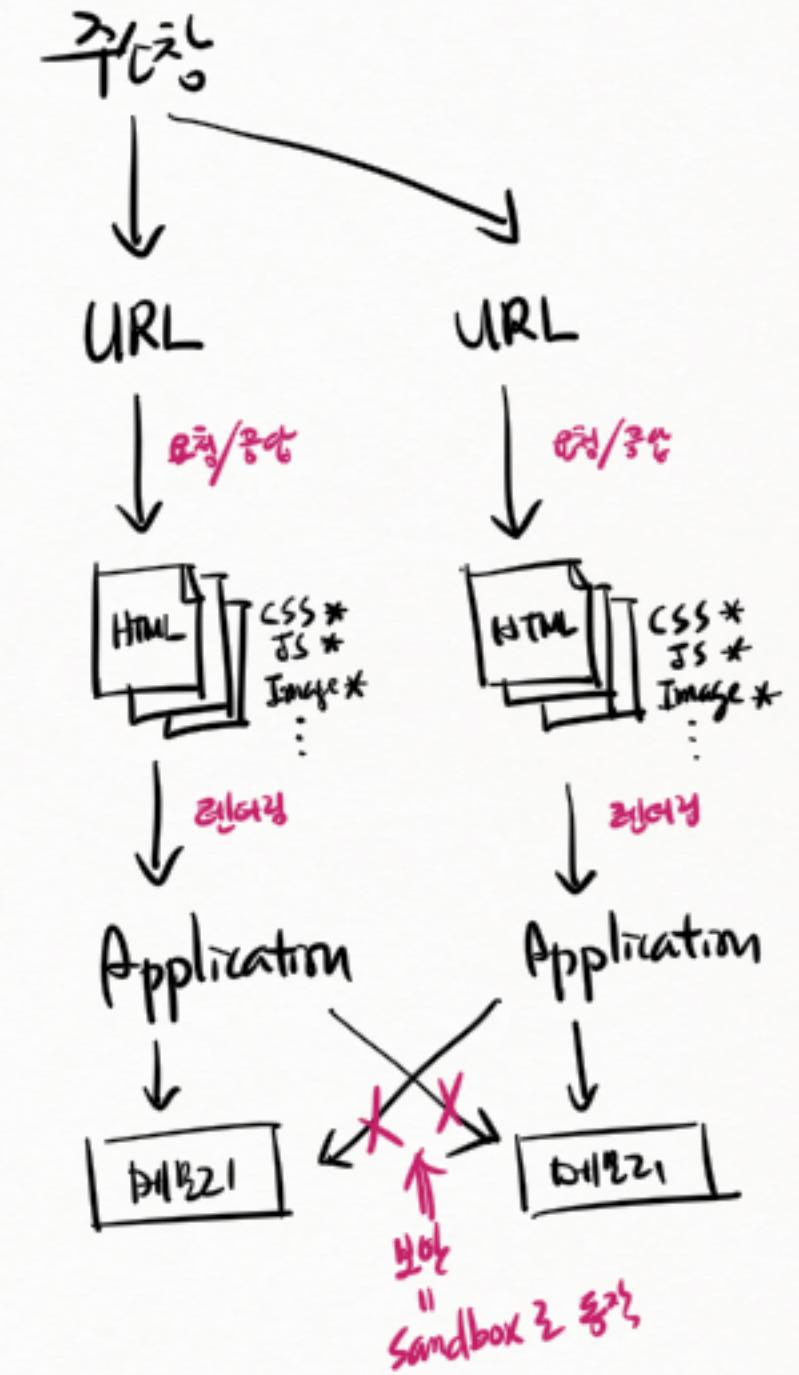
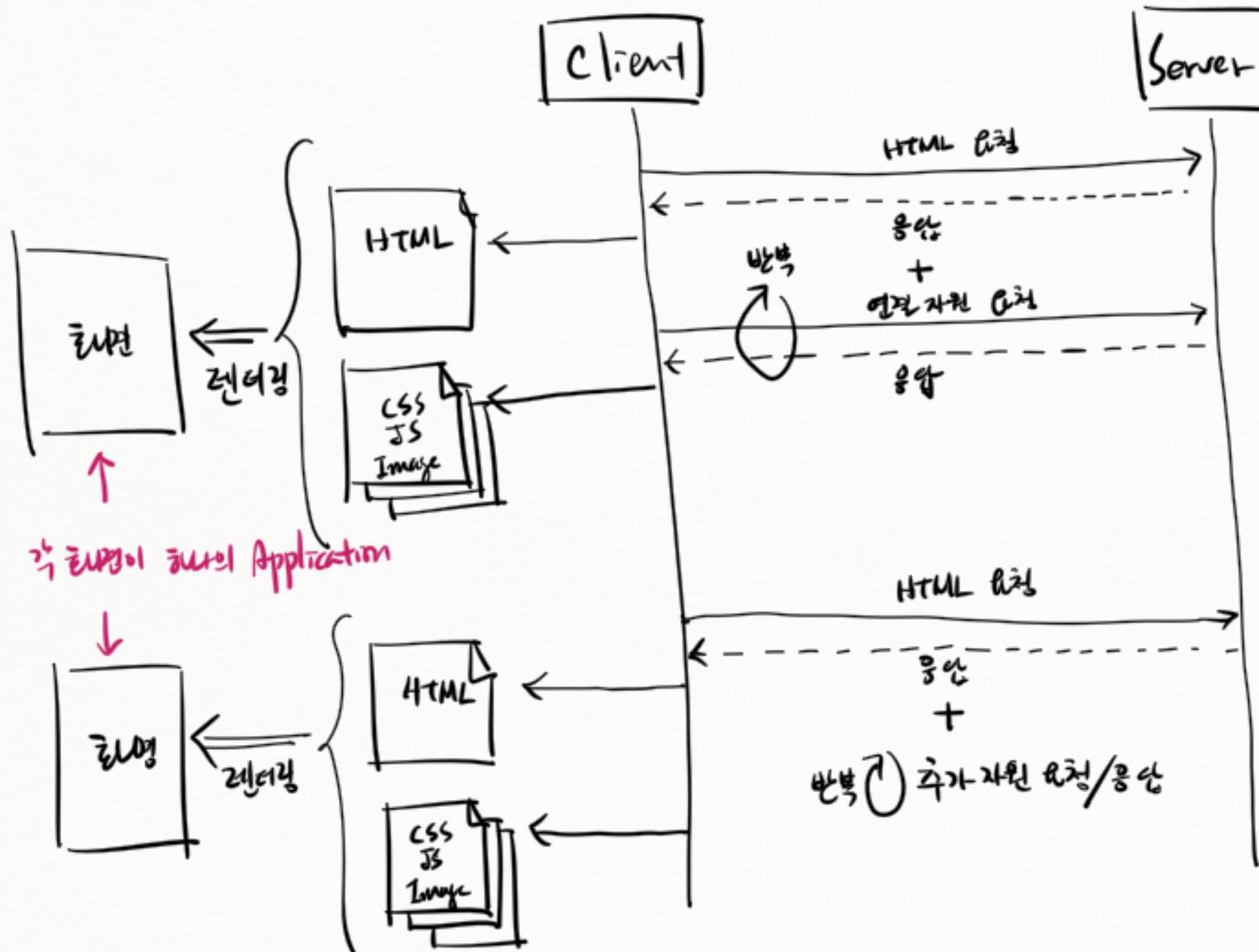
AJAX

Asynchronous
JavaScript
And
XML

* 웹페이지 가져오기



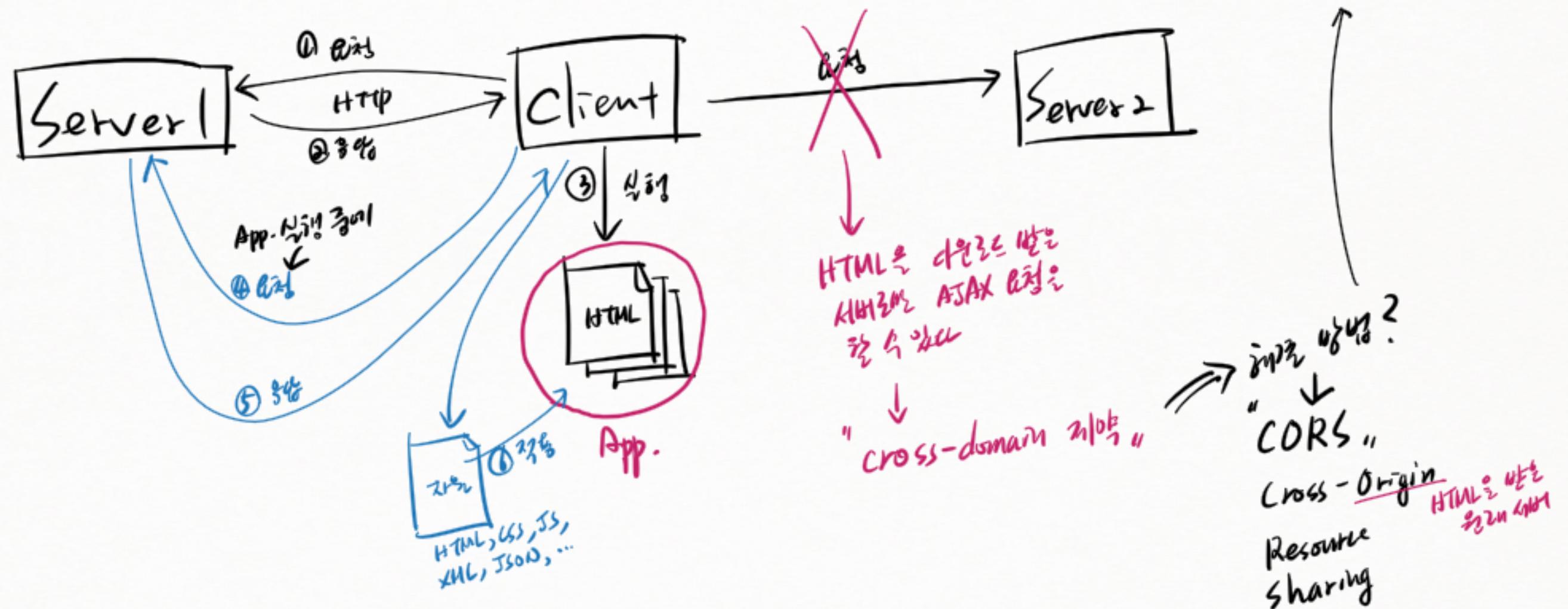
* 웹 레이저와 애플리케이션



* AJAX (Asynchronous JavaScript And XML)

↳ 현재 App.의 실행을 유지한 채 서버에 자원을 요청하는 기술
HTML, CSS, JS, XML, JSON, ...

Origin이 다른 경우는
Access-Control-Allow-Origin: *



* AJAX 를 이용한 API

```
let xhr = new XMLHttpRequest();
```

AJAX 툴이 웹kit 도구를 준비하기로 생겼다

```
xhr.open("GET", "a.html", false);
```

HTTP 요청 메소드

비동기 여부

HTTP 요청 URL

비동기 여부

```
xhr.send();
```

HTTP 요청이 최종 정복 되었다

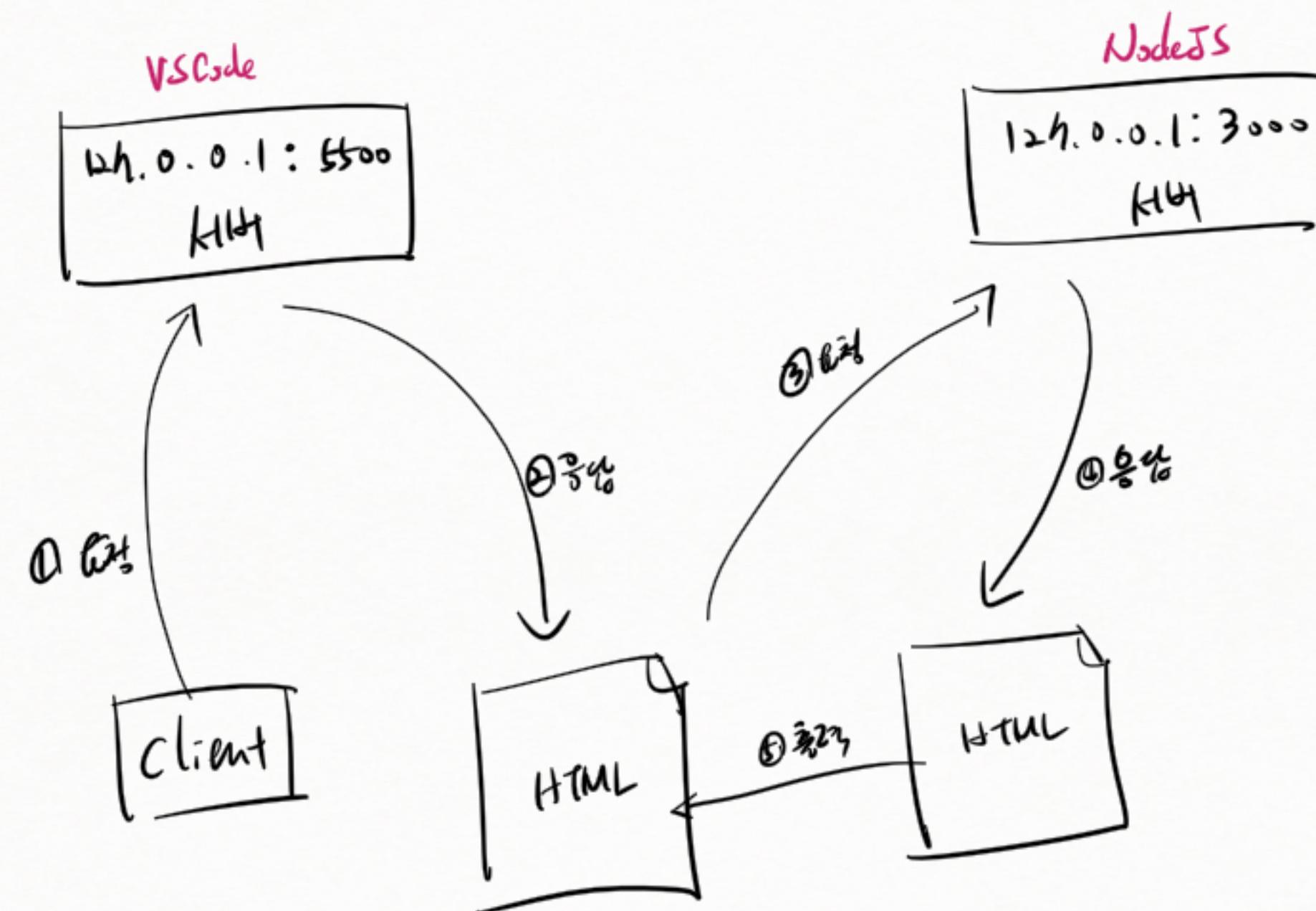
→ 키보드 응답되면 리턴된다 (synchronous 리턴)

→ 키보드 응답과 상관없이 즉시 리턴된다 (Asynchronous 리턴)

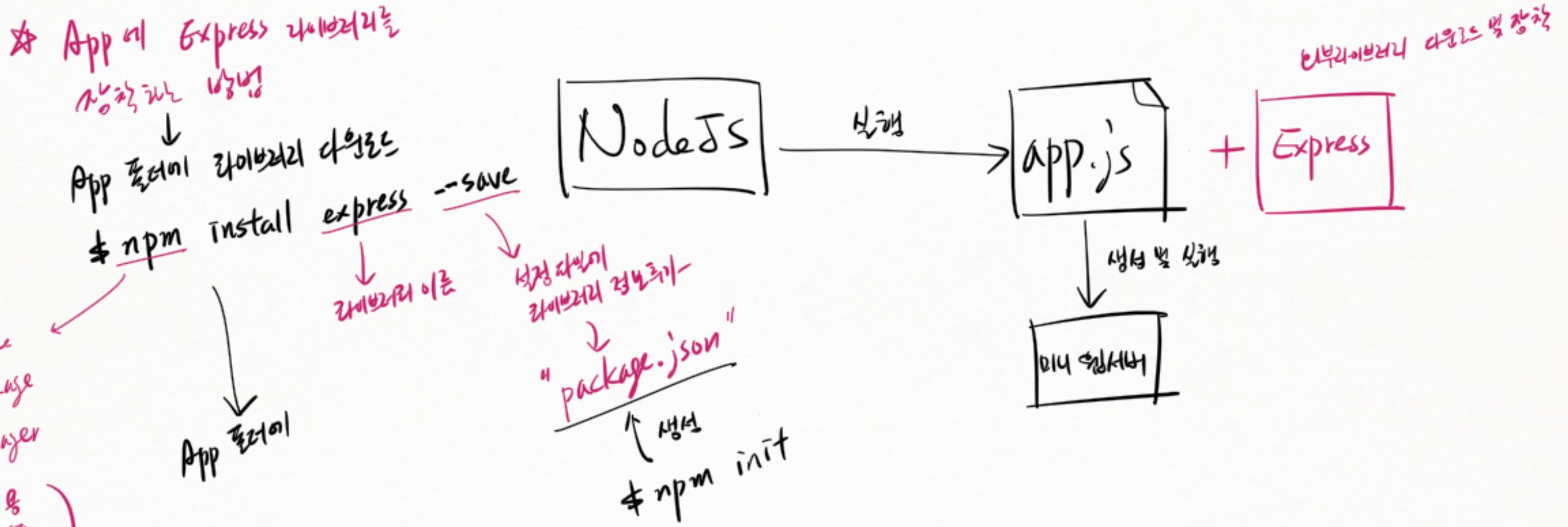
```
xhr.responseText
```

응답 데이터를 갖고 드로퍼다

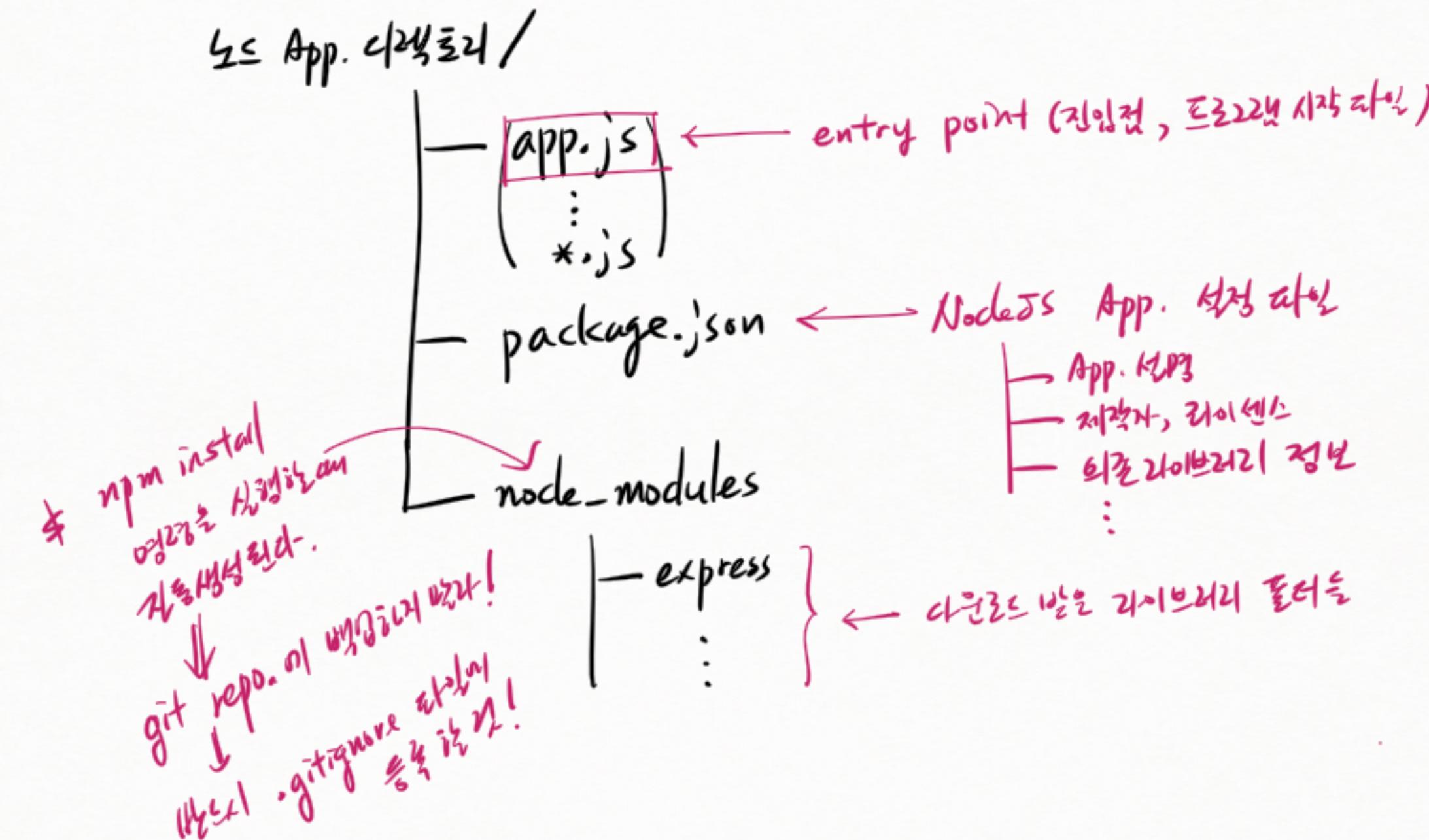
* AJAX 구조 히스토리



* NodeJS 와 Express 라이브러리



* NodeJS Application 구조



* npm install | 라이브러리 설치

\$ npm install | 라이브러리 설치

체증 이유를 가지 라이브러리는 다음은 같다.

① node_modules 폴더가 있다면 생성된다.

② 라이브러리가 없으면 다음으로 가다.

↳ (설정된 경로에 따라
버전을 갖나하나
설정한 버전을 다음으로 한다.)

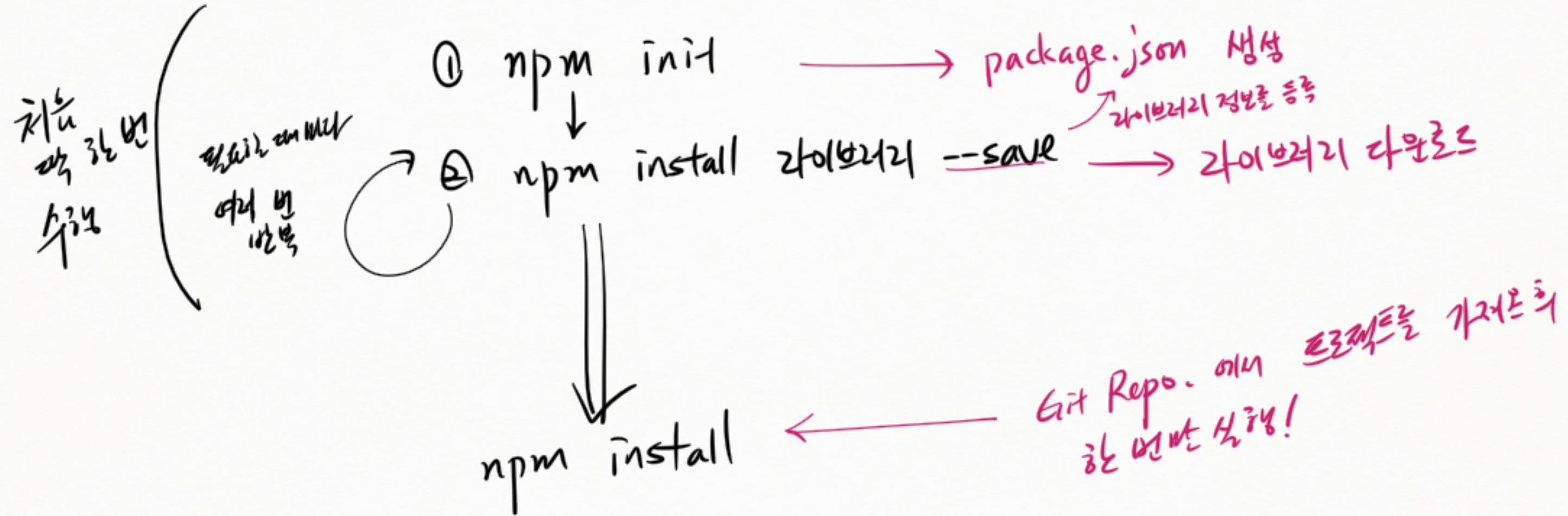
* npm install

\$ npm install

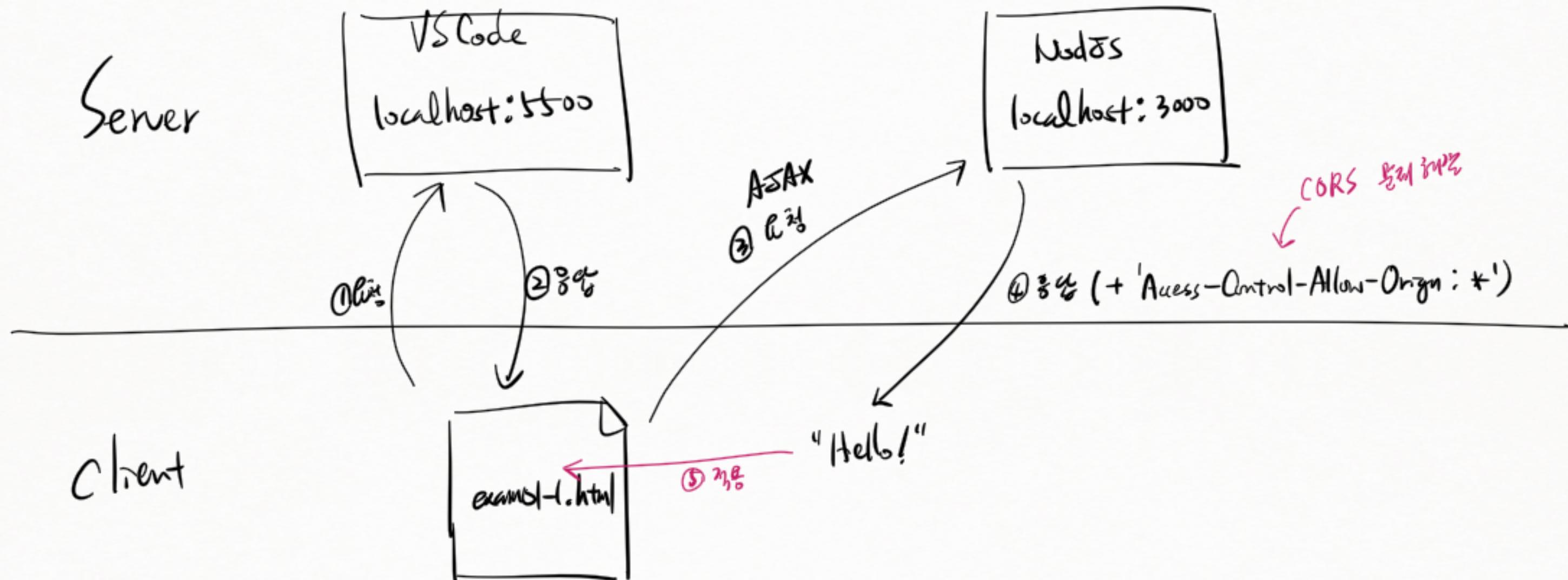
지나친 ID

- ① 작업 폴더에서 package.json 파일을 찾는다
명령을 실행하는 폴더
- ② package.json 파일이 등록한 라이브러리를 다운로드 한다
- |
 | node_modules 폴더가 없으면 생성한다
 | 라이브러리가 없으면 다운로드 한다.
 |
 | ↳ (설정된 경로에 따라
 | 버전을 갖나타내
 | 설정한 버전을 다운로드 한다.)

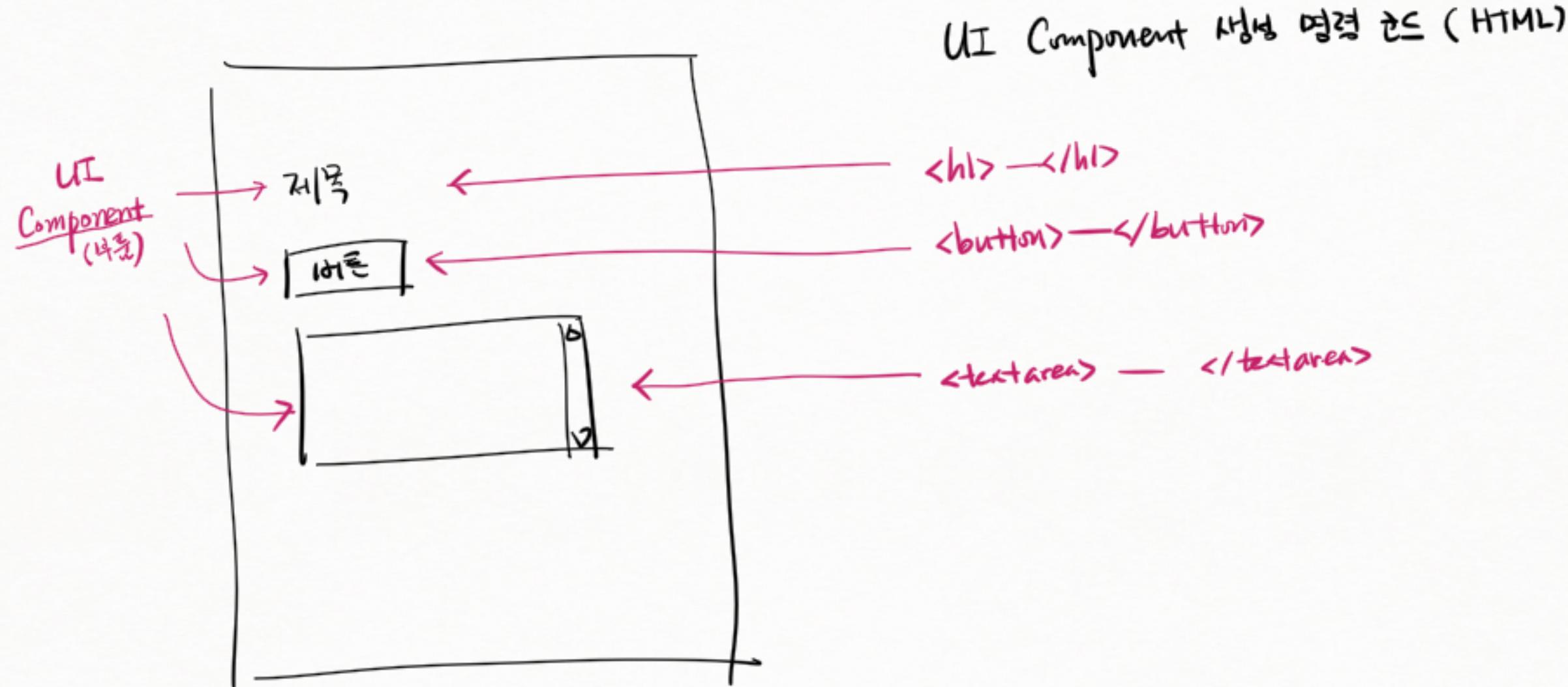
* npm 사용



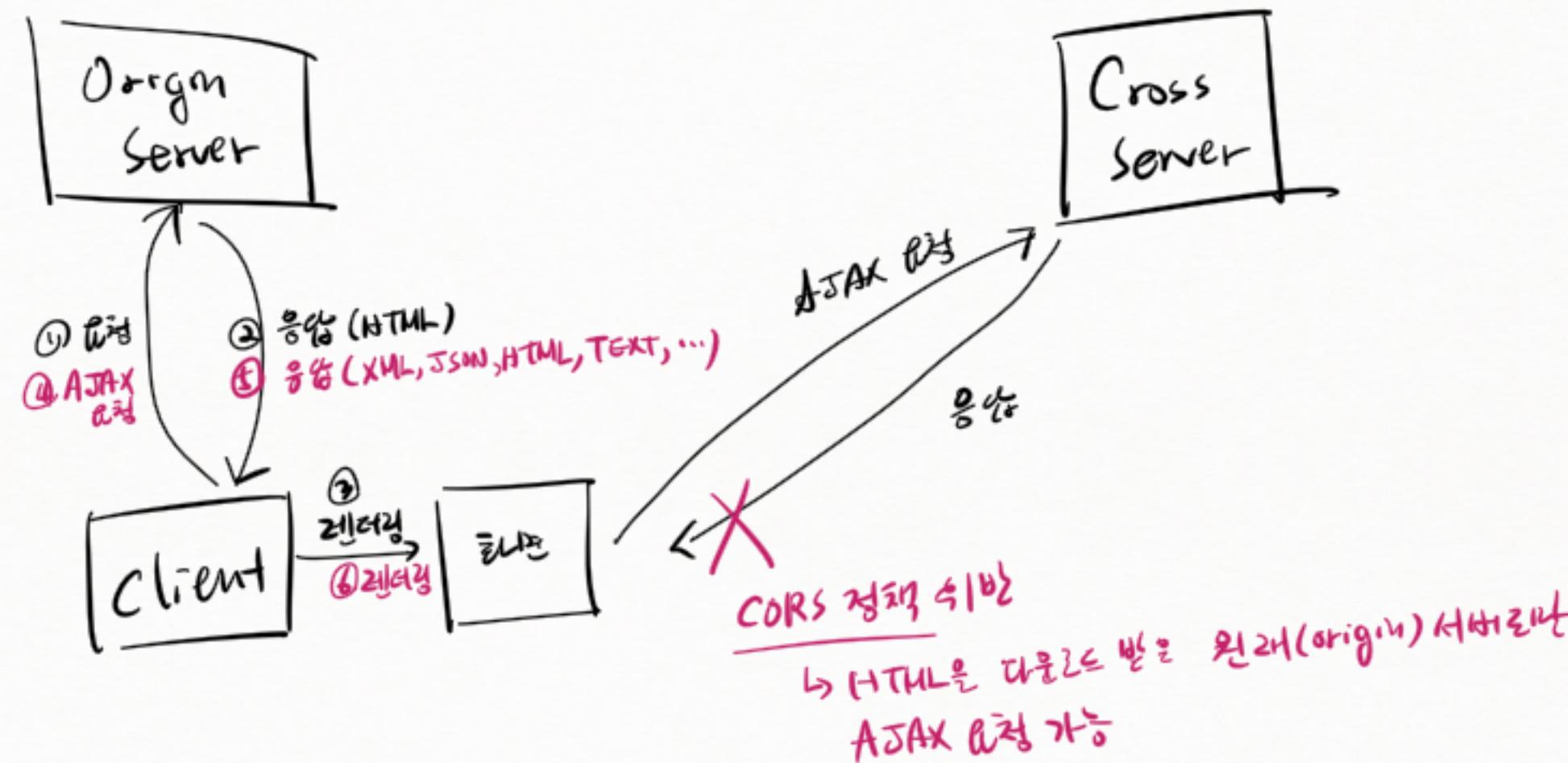
* AJAX : "Hello!"



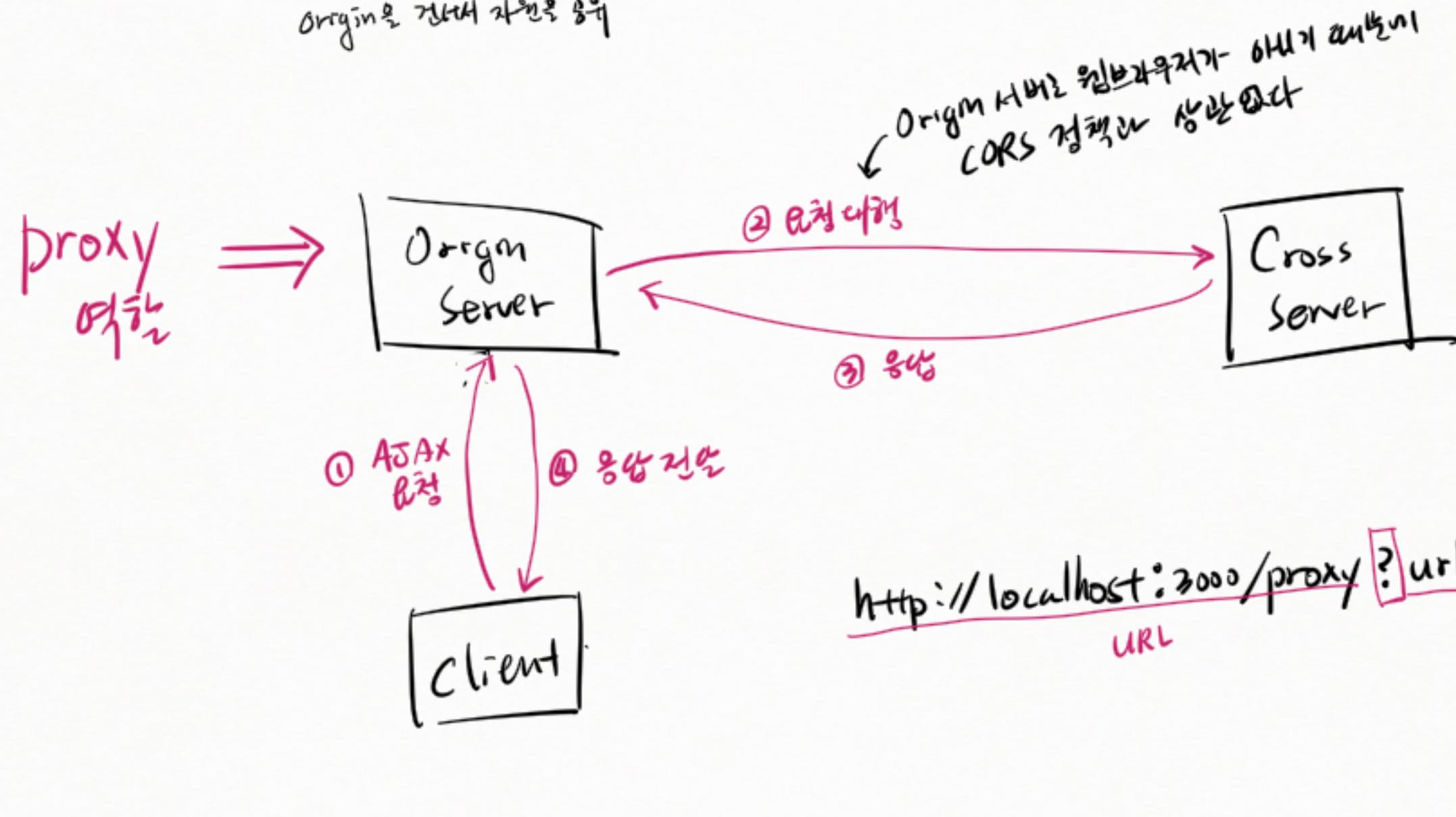
* 래그와 UI Component



* CORS Policy (정책) 을 우회하는 방법 - 현황
origin을 전역 차원을 풀기



* CORS Policy(정책) 을 우회하는 방법 - 해제 방식
origin을 전역 차원으로 풀기



* Query String چیزی

Client

http://localhost:3000/proxy?url=http://www.naver.com

Server

```
app.get('/proxy', (req, res) => {  
  req.query.url  
});
```

↓ return

http://www.naver.com

:

```
});
```

* serialization / deserialization

객체

```
let obj = {  
    name: "홍길동",  
    age: 20  
};
```

→ JSON.stringify(obj)

JSON 문자열

serialization 정의
(encoding)

JSON.parse()

deserialization 정의
(decoding)

Data
데이터 타입이나
(구조)

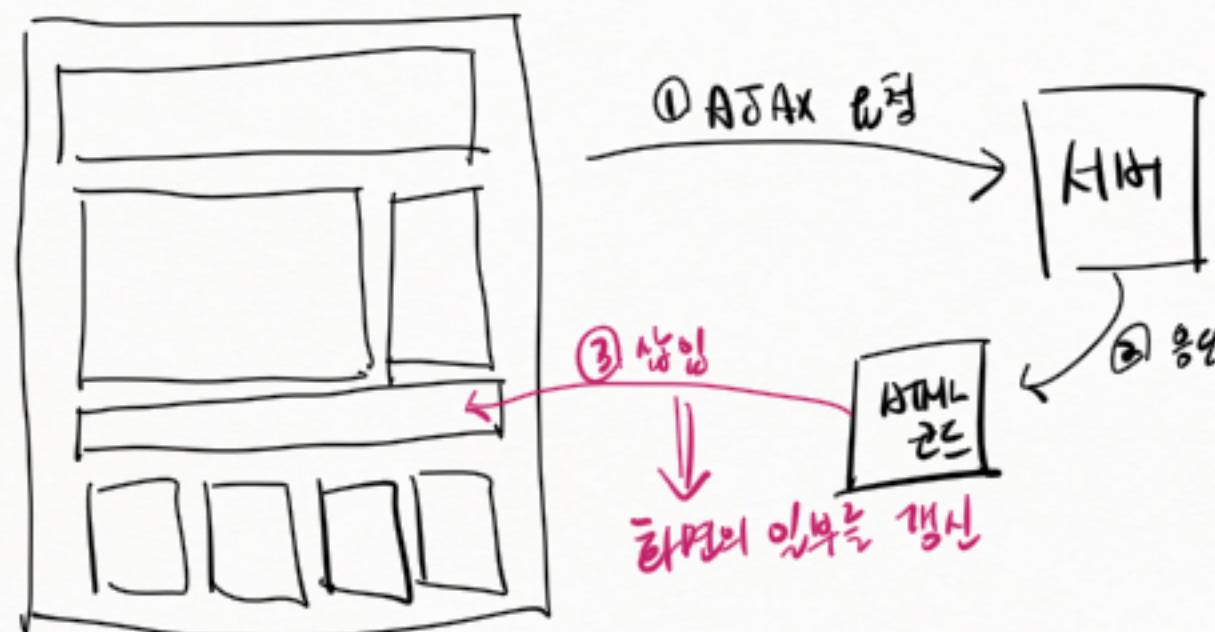
serialize

deserialize

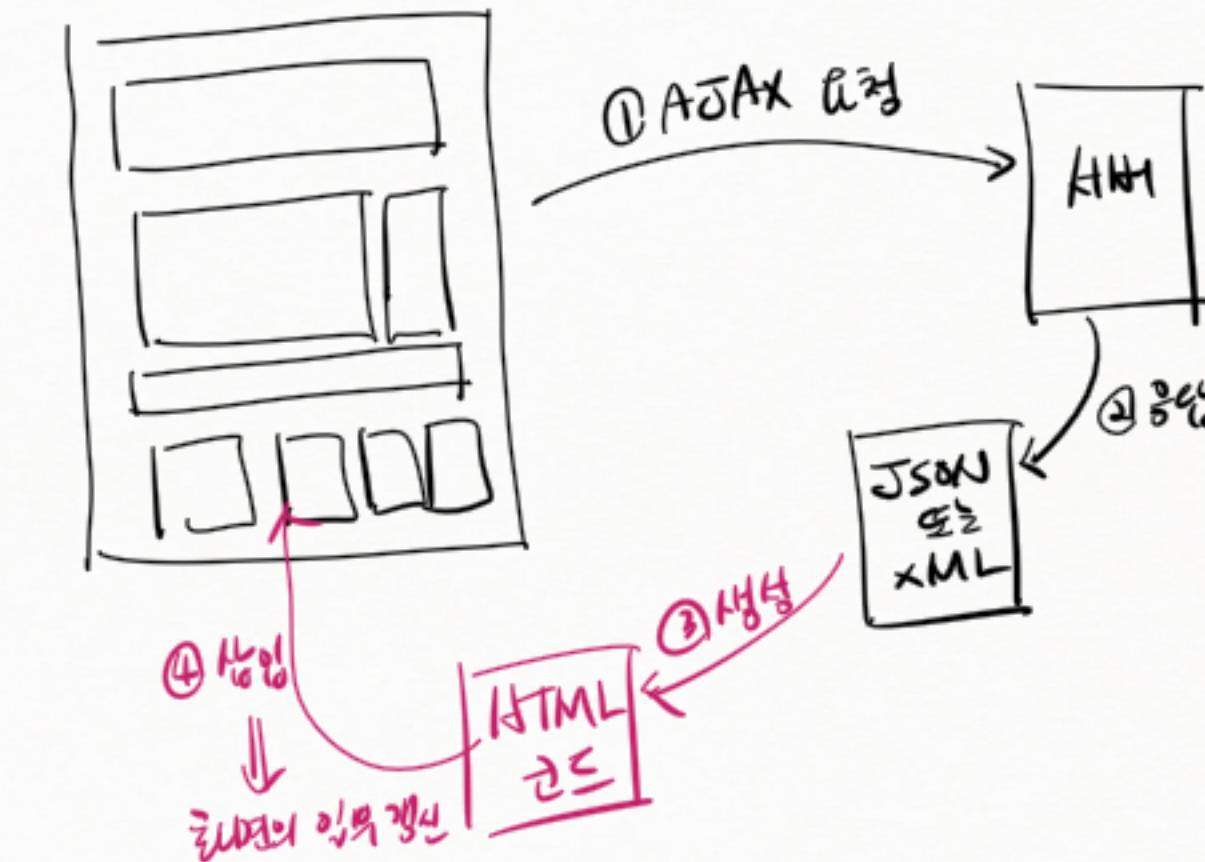
문자 / 라이브
알고리즘
(serial)

* AJAX 활용

① 화면의 일부를 가져오기

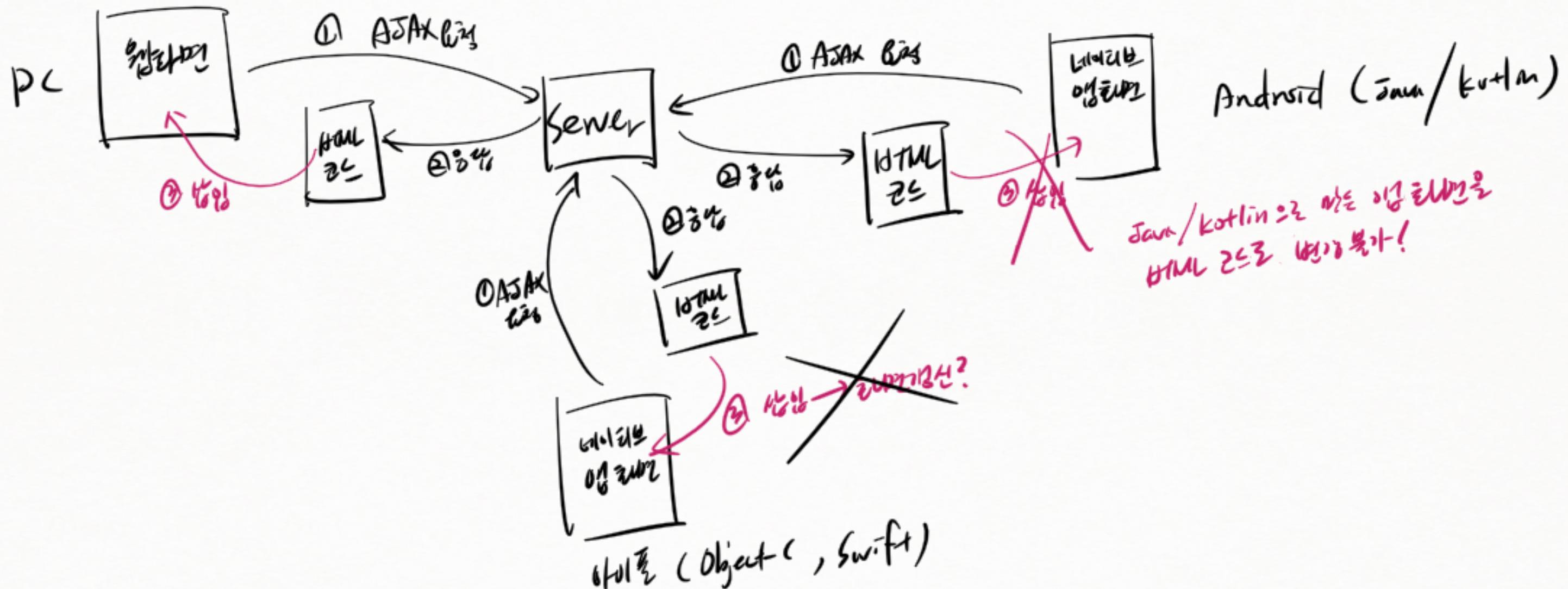


② 데이터를 가져오기



* AJAX의 응답 결과가 JSON 또는 XML 데이터인 이유?

① HTML 코드를 응답으로 쓰면 문제점



* AJAX의 응답 결과가 JSON 또는 XML 데이터인 이유?

② 서버에서 JSON 또는 XML 형식으로 데이터를 응답

↓
먼저 디비에 내용이 있다!

