* 변수

var (global) ← 페이지 전체에서 사용 가능

let (block) ← block 안에서만 사용

const (상수)

* type binding

Java int a;

int 자료형 변수명

} static type binding

JavaScript var a;

변수명 앞에 자료형 명시 안함

} dynamic

a = "Hello";

string

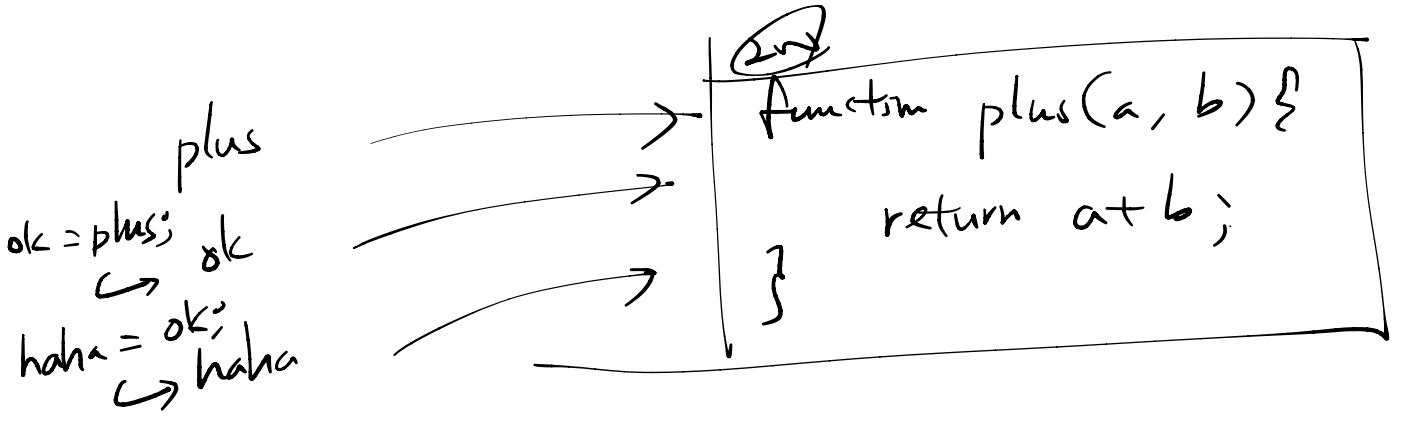
a = 100;

number

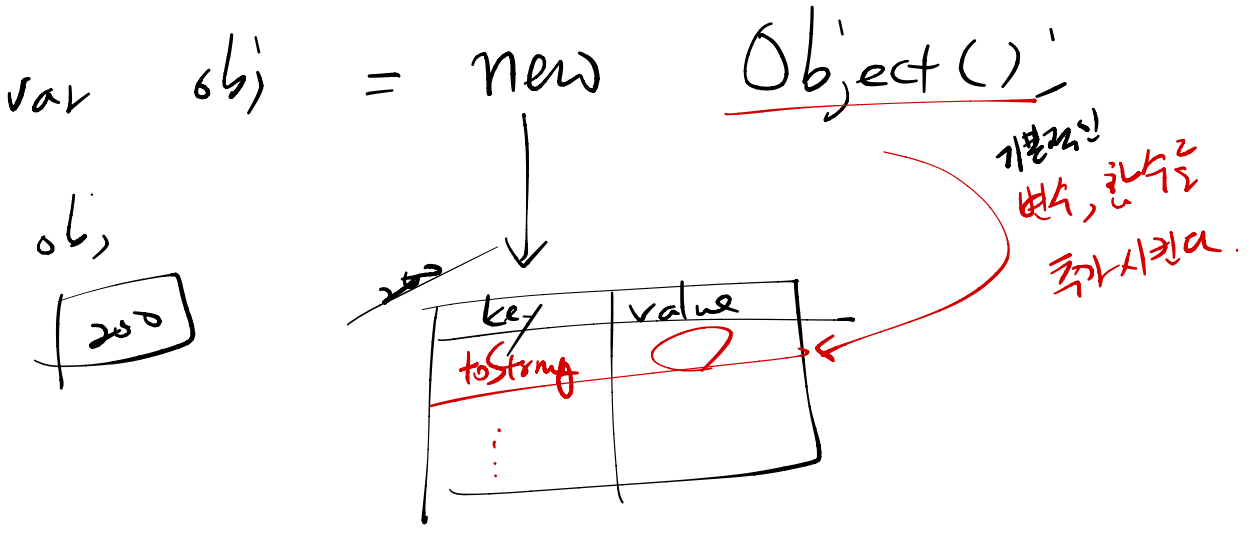
a = true;

boolean

* function



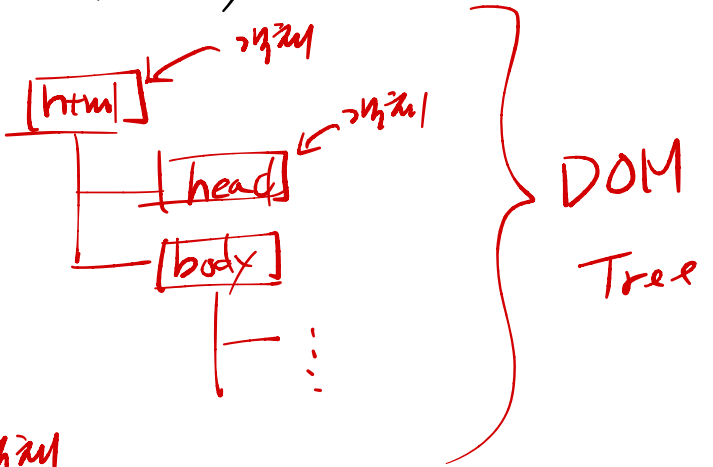
* object

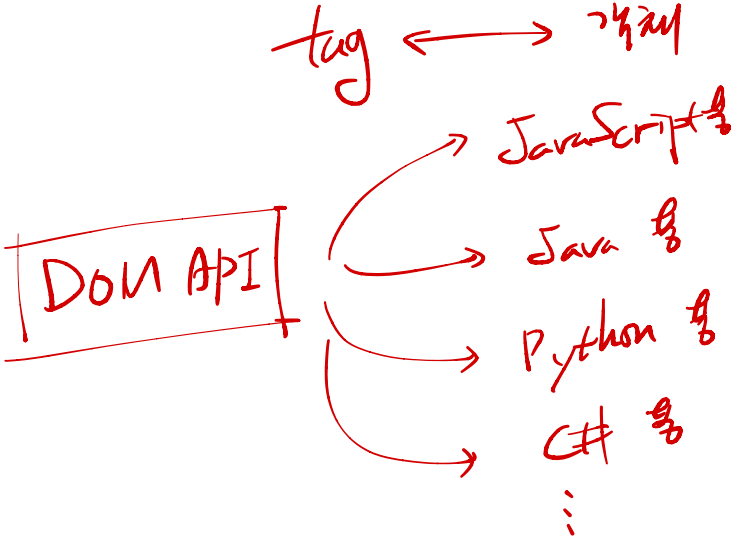


* DOM API (Document Object Model)

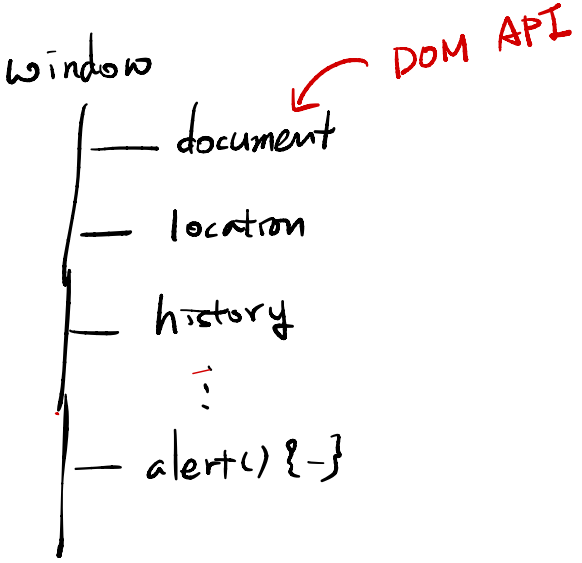
↳ tag들을 다루기 위한 API

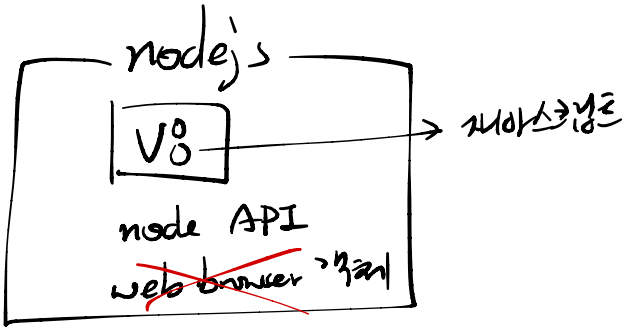
```
<html>
  <head> </head>
  <body> </body>
</html>
```

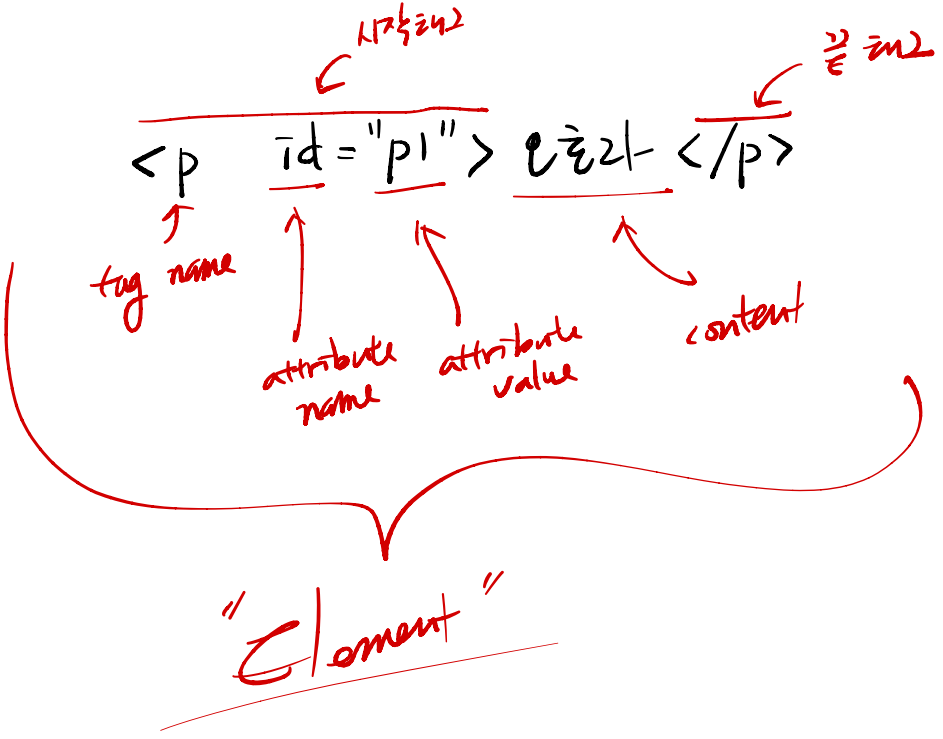




* Web Browser에서 제공하는 객체

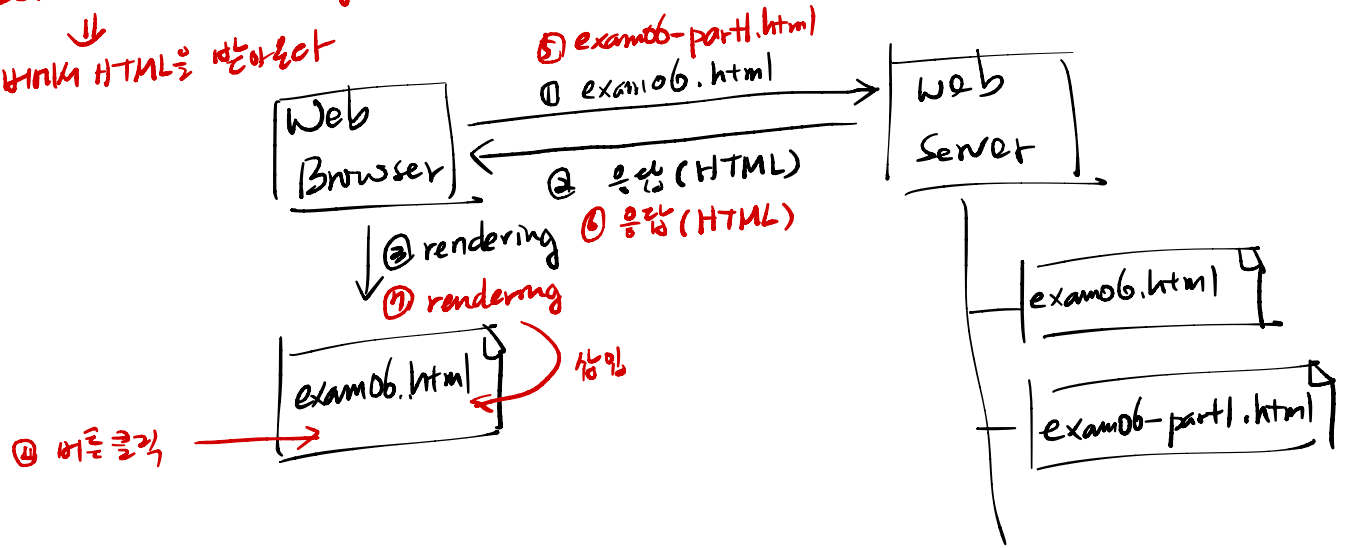





시작태그
끝 태그
안녕
tag name
attribute name
attribute value
content
"Element"

* AJAX (Asynchronous JavaScript And XML)

① server-side rendering
⇓
서버에서 HTML을 받아온다



* AJAX (Asynchronous JavaScript And XML)

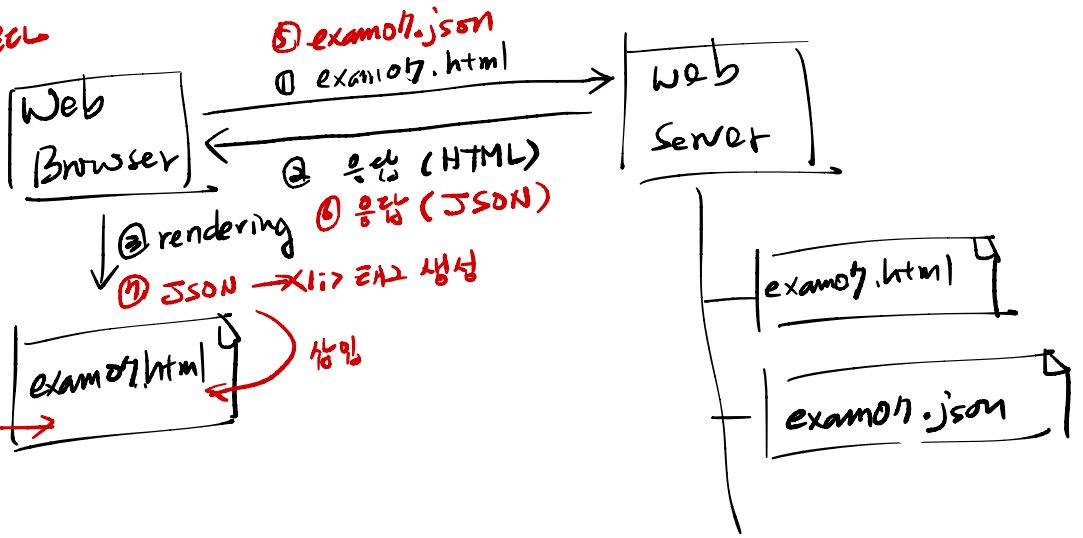
② client-side rendering

⇓

서버에서 Data 받아온다

⇓

클라이언트에서 HTML을 만든다



* Data Type

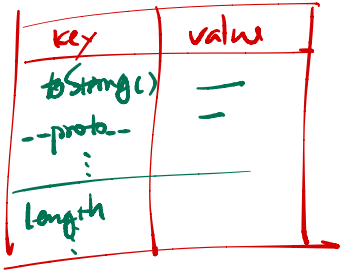
- ✓ string → "aaa" , 'aaa'
- ✓ number → 300 , 3.14 , 314E-2 (314×10^{-2}), NaN , Infinty , -Infinity
- ✓ boolean → true , false
- ✓ undefined → 변수에 값을 지정하지 않은 상태 = 변수의 type이 definite하지 않은 상태
- ✓ object → 객체 , null ← 객체형은
- ✓ function → 함수
- ✓ symbol → 유니크 값

* 배열

```
var arr = new Array();
```

new → 빈객체

Array() → 추가



new
↓
Object()
↓
Array()

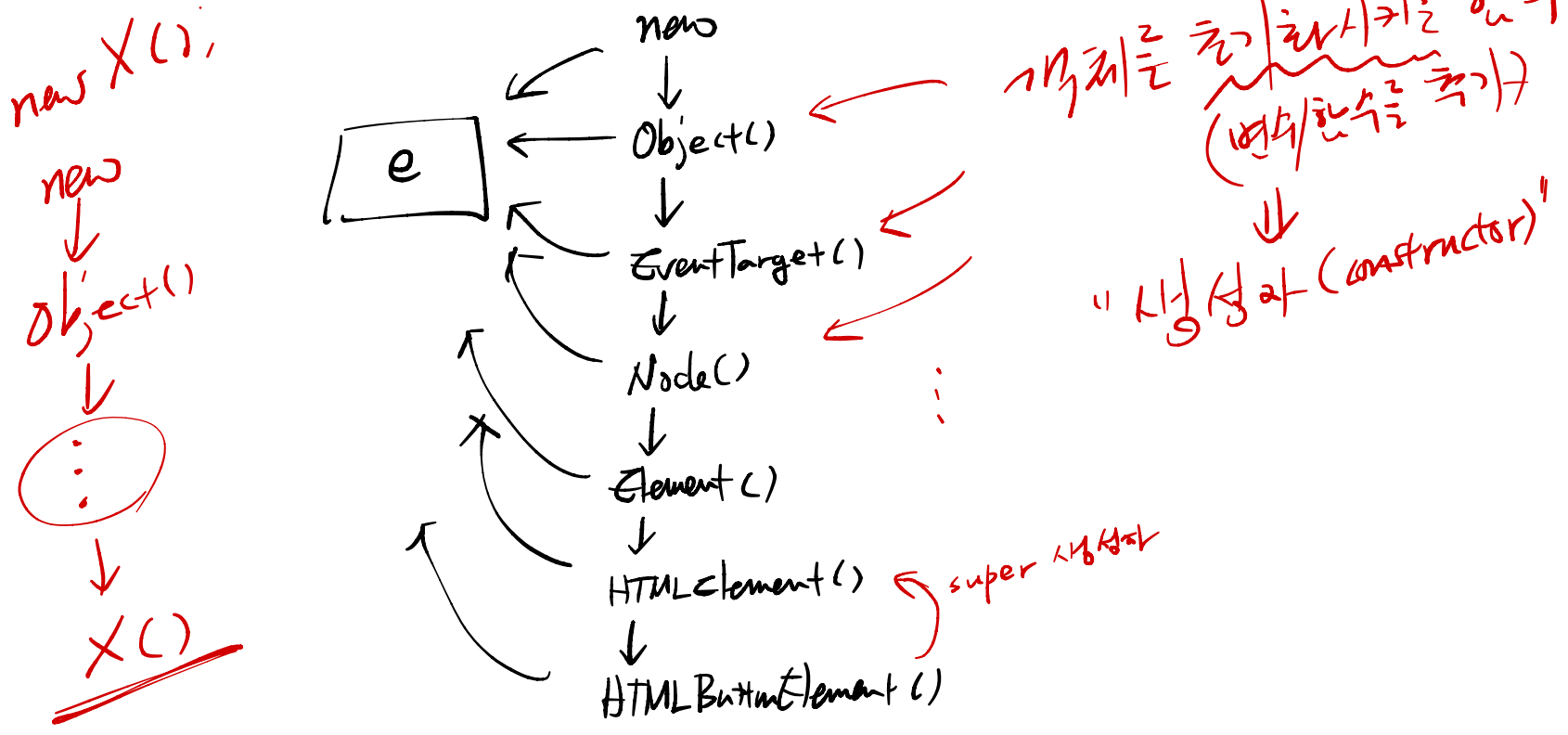
추가
⇓
Object() 호출 → 모든 객체가 반드시 갖춰야 할 프로퍼티 (변수, 함수) 객체 추가
⇓
Array() 호출
↓
배열만의 프로퍼티 추가

```
var arr = new Array()

arr[0] = "aaa";
arr[1] = "bbb";
arr[5] = "ccc";

console.log(arr.length);
```

key	value
0	"aaa"
1	"bbb"
5	"ccc"

* 생성자

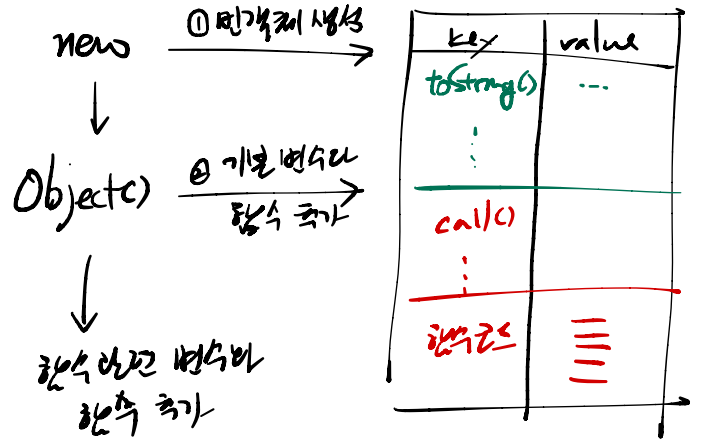
```
var e = document.getElementById("btn1"); // <button> ... </button>
```



* function 과 객체

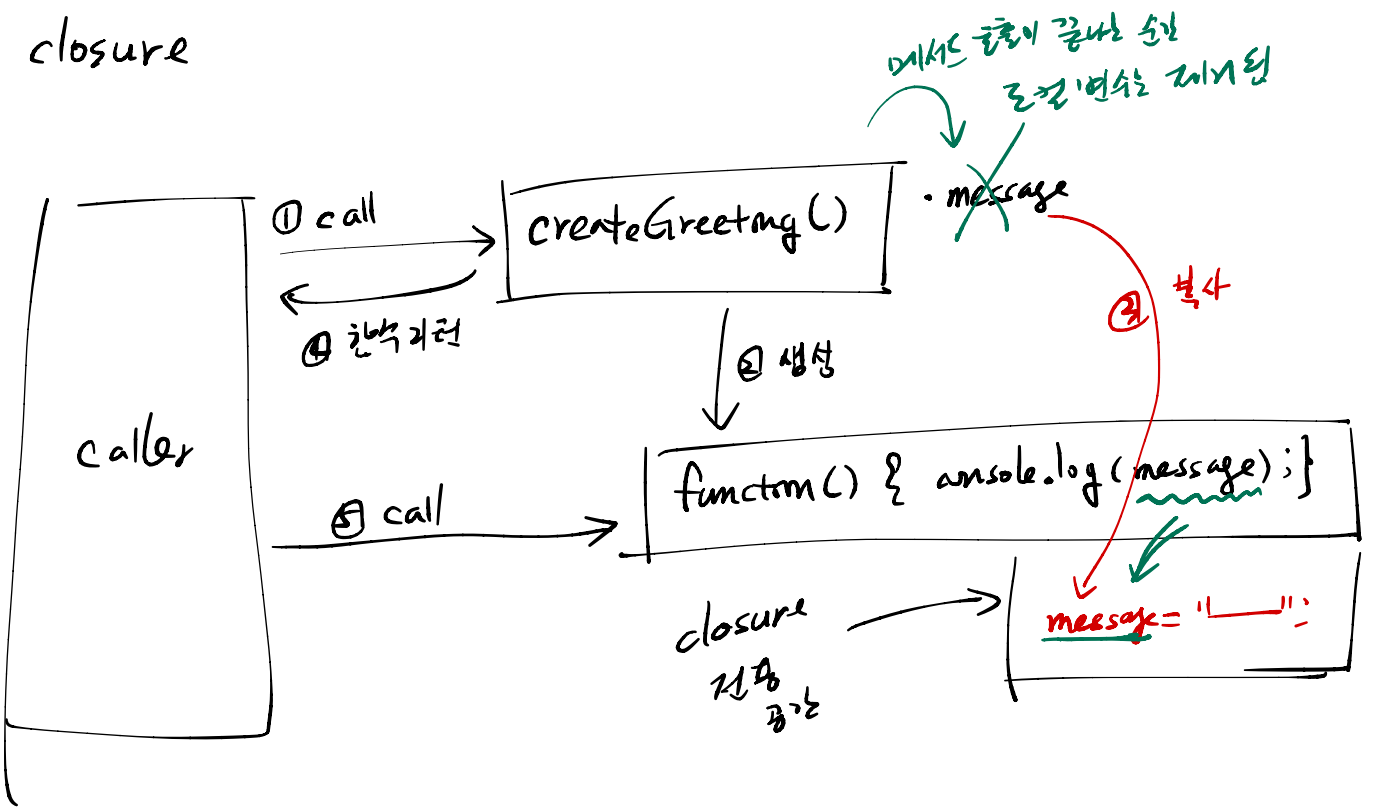
```
function f1() {
    ≡ ← 함수코드
        ||
        function body
}
```

⇒

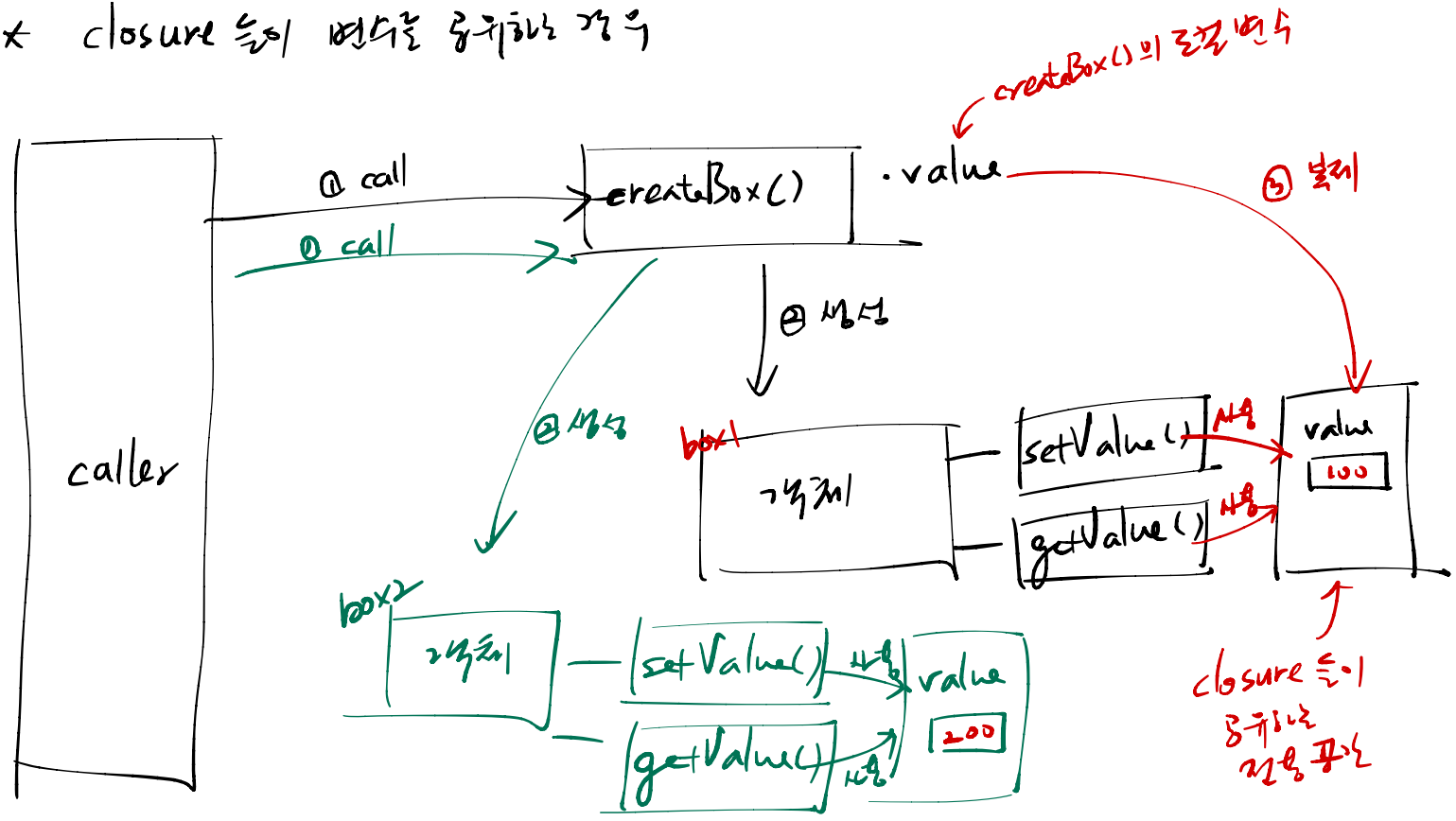


function = 객체 + 함수코드

* closure

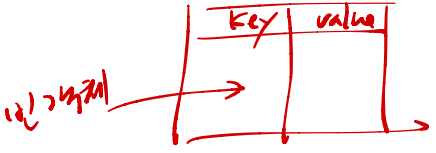

메서드 호출이 끝나도 로컬 변수를 제거됨
caller
① call
② 반환 리턴
createGreeting()
· message
③ 생성
④ 복사
⑤ call
function() { console.log(message); }
closure 전용 공간
message = '!!!'

★ closure 들의 변수를 공유하는 경우



* 객체 생성

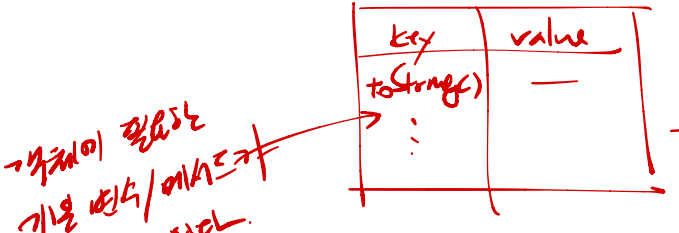
① Object.create(null) → 생성 → 빈 객체



② Object.create(Object.prototype) = new Object() ③

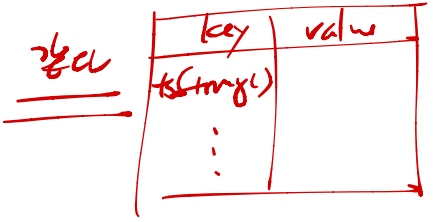
원형 객체(prototype)

원형을 참조하여 객체 생성



객체의 원형은 기본 변수/메서드가 들어 있다.

같다



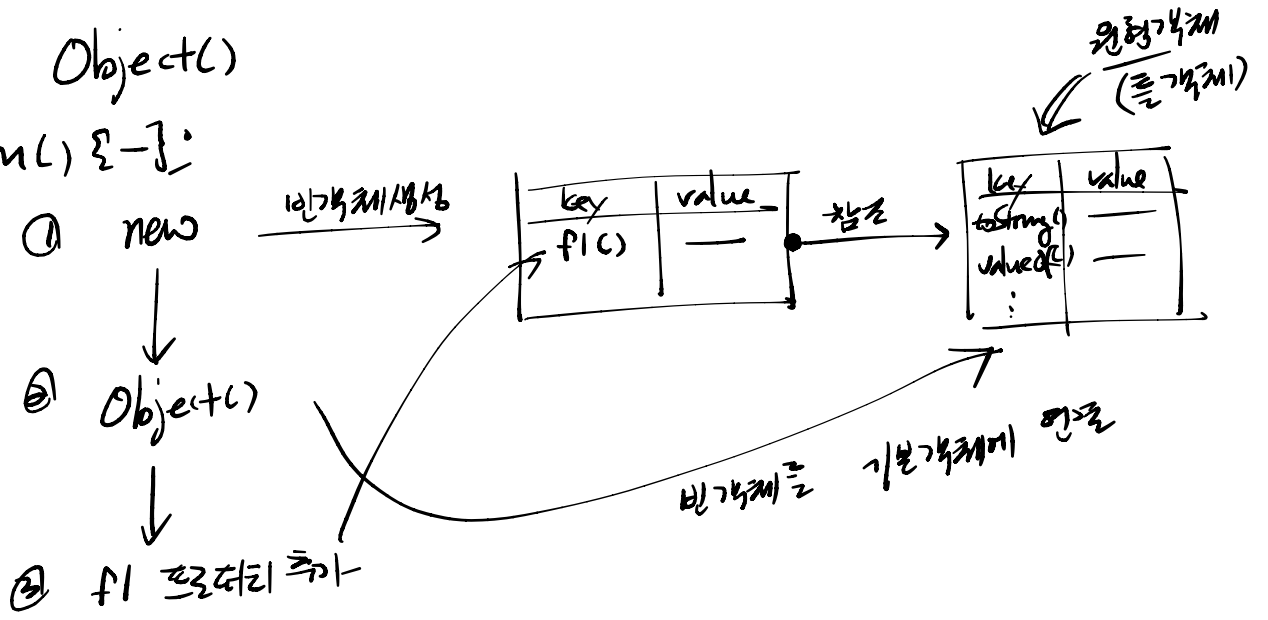
원형을 참조하여 객체생성

생성

④ { }

* 객체와 원형객체

```
var obj = new Object()
obj.f1 = function() {-};
```

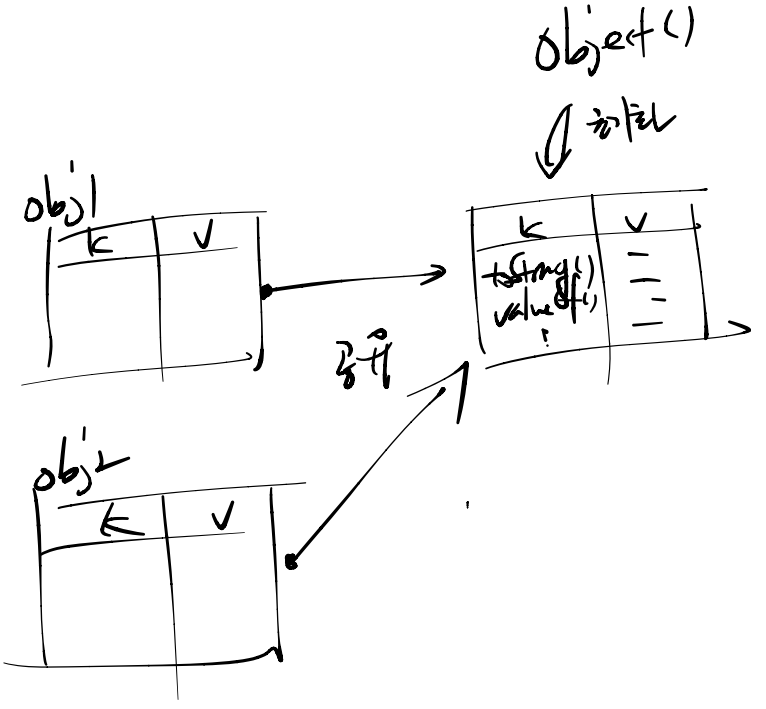


obj.f1(); ⟶ obj 객체에서 f1()을 찾아 호출

obj.toString(); ⟶ obj 객체에서 f1()을 찾으면(X) ⟶ 원형객체에서 찾아 호출!

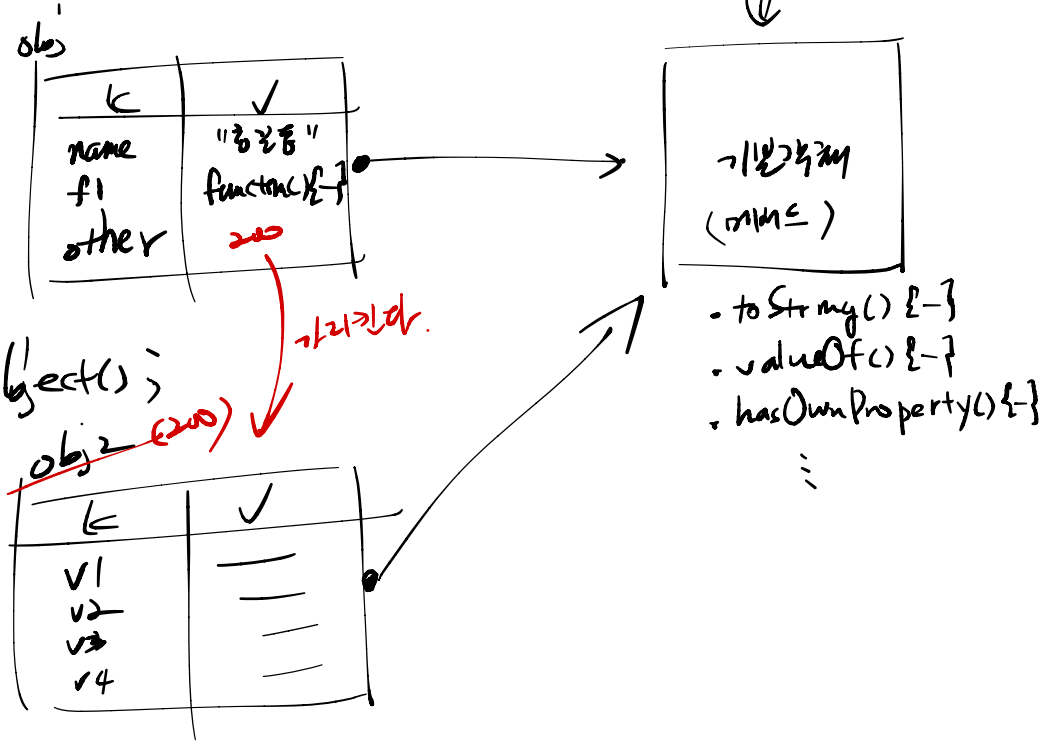
* 원형객체 상속

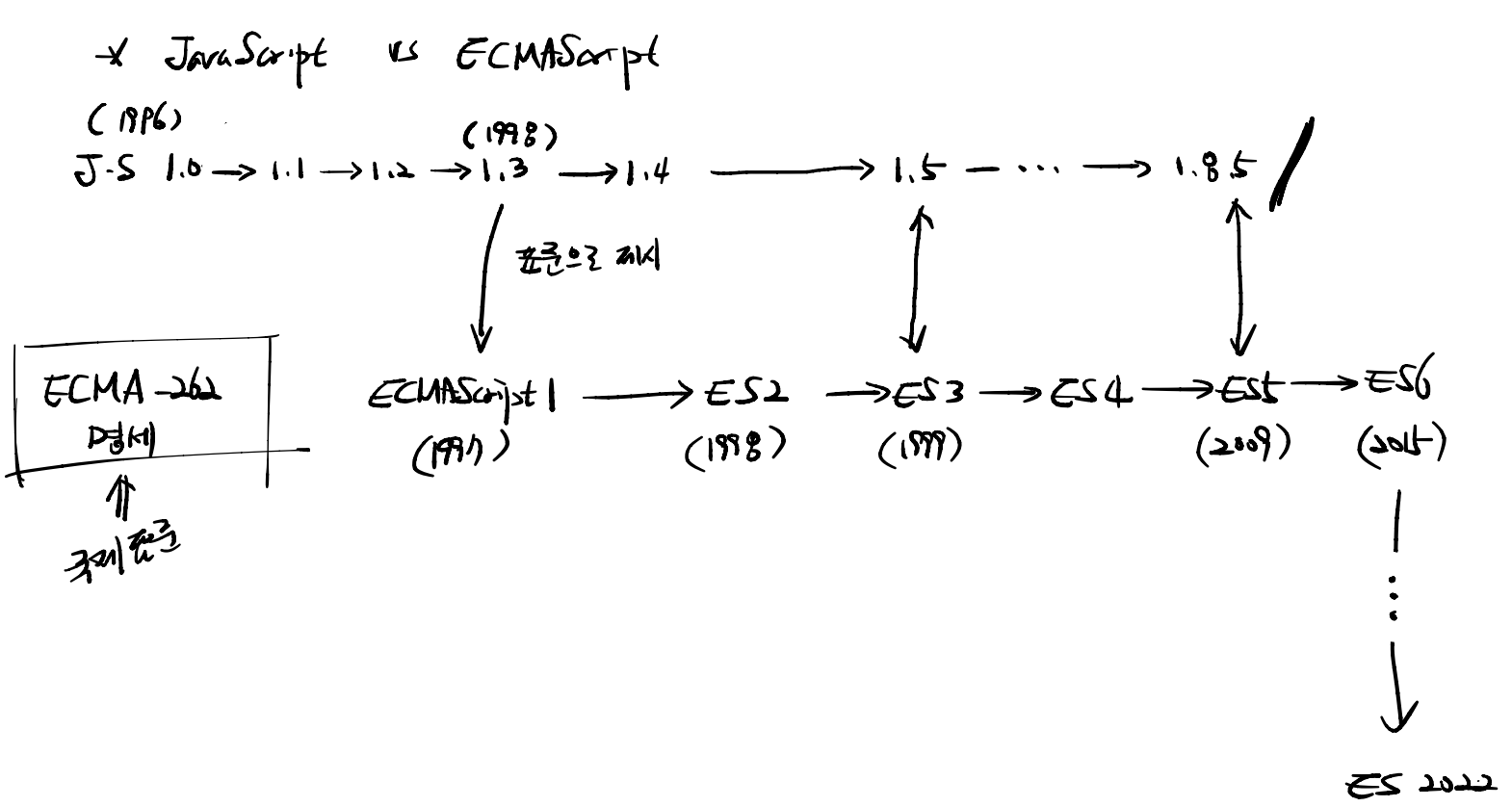
```
var obj1 = new Object();

var obj2 = new Object();
```



* 객체 생성 → 다른 객체는 프로퍼티로 참조 (참조)

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

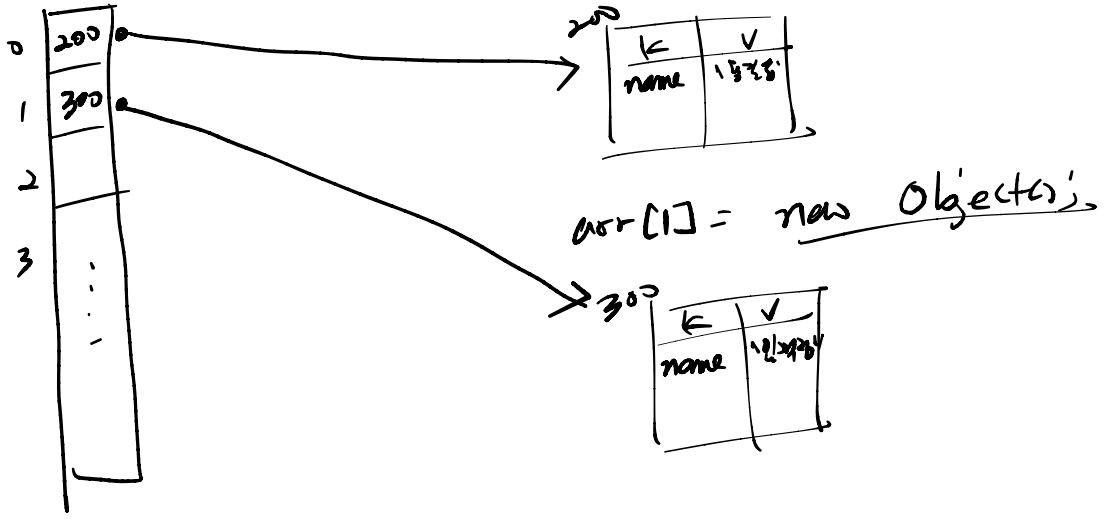



JavaScript vs ECMAScript
(1996)
J-S 1.0 → 1.1 → 1.2 → 1.3 → 1.4 → 1.5 → ··· → 1.8.5
(1998)
표준으로 채택
ECMA-262
명세
국제표준
ECMAScript 1 → ES2 → ES3 → ES4 → ES5 → ES6
(1997) (1998) (1999) (2009) (2015)
ES 2022

* 배열의 객체

var arr = [];

arr[0] = new Object();



arr[1] = new Object();

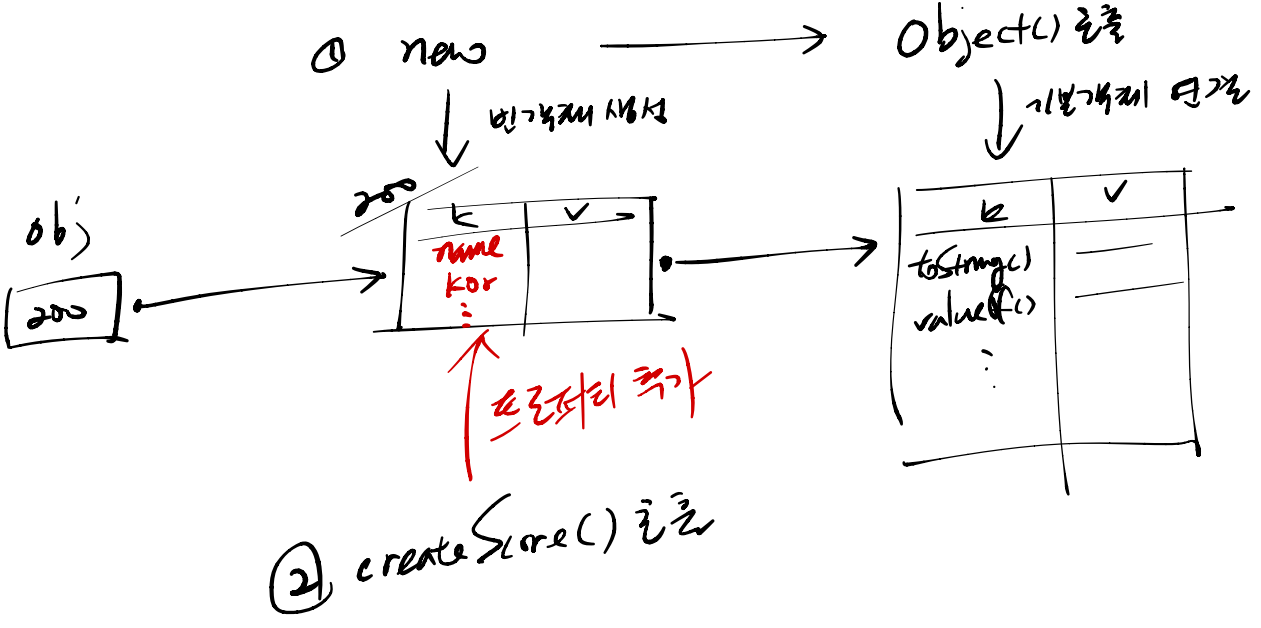
arr[0].name = "홍길동";

arr[1].name = "박찬호";

* new 객체 생성

생성자(constructor) ↳ new 다음에 호출되는 함수

```
var ob1 = new createScore();
```



* 생성자와 prototype

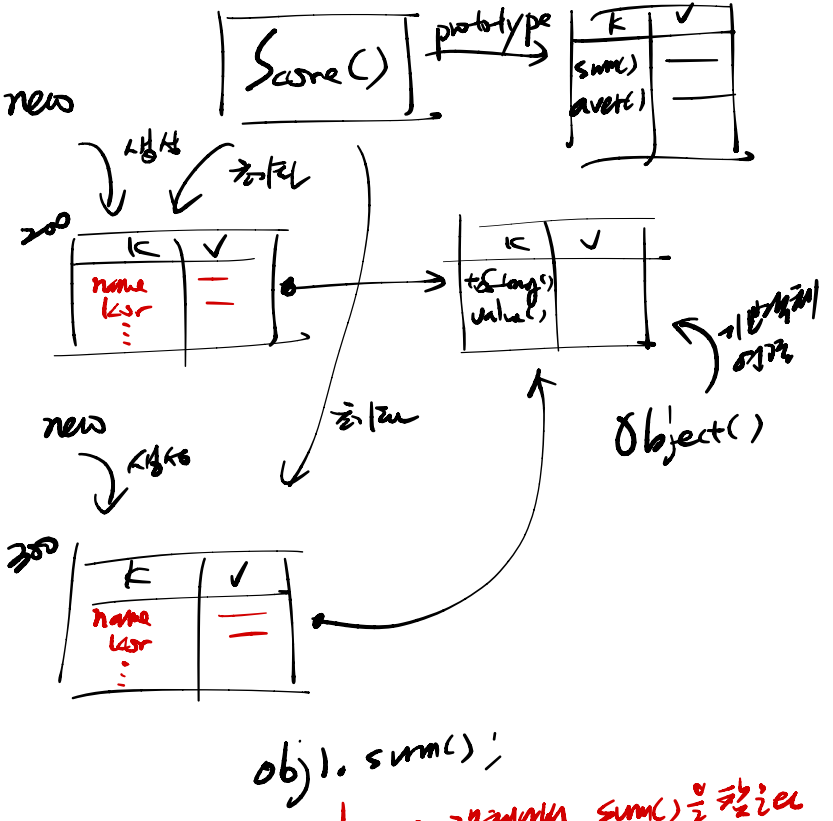
```
var obj1 = new Score();
```

obj1 [200]

```
var obj2 = new Score();
```

obj2 [300]

```
Score.prototype.sum = function() {};
Score.prototype.aver = function() {};
```



obj1.sum();

↳ ① 객체에서 sum()을 찾는다

↓ 없다면

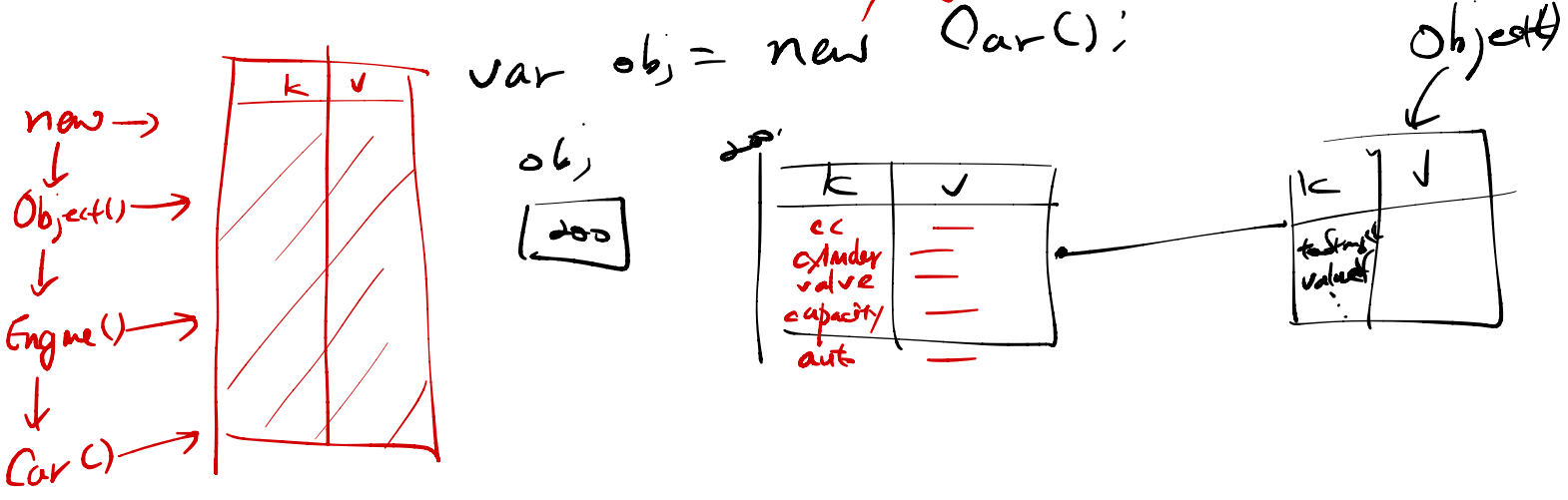
② 생성자의 prototype에서 sum()을 찾는다

* 생성자를 이용한 상속

```
Engine(va, cy, cc) {
  this.cc = cc;
  this.cylinder = cy;
  this.valve = va;
}
```

```
Car(va, cy, cc, ca, au) {
  Engine.call(this, va, cy, cc);
  this.capacity = ca;
  this.auto = au;
}
```

```
var ob = new Car();
```



* 생성자 호출

```
var e = new Engine();
```

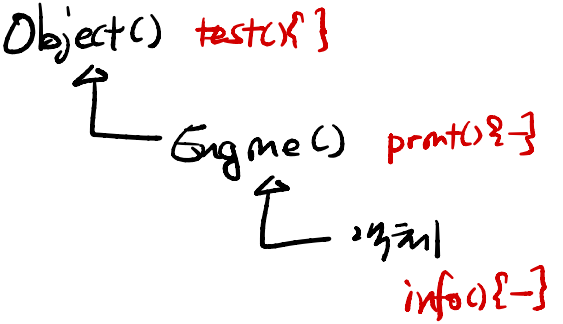
e.info(); → 객체.info()

e.prnt(); → Engine.prototype.prnt()

e.test(); → Object.prototype.test()

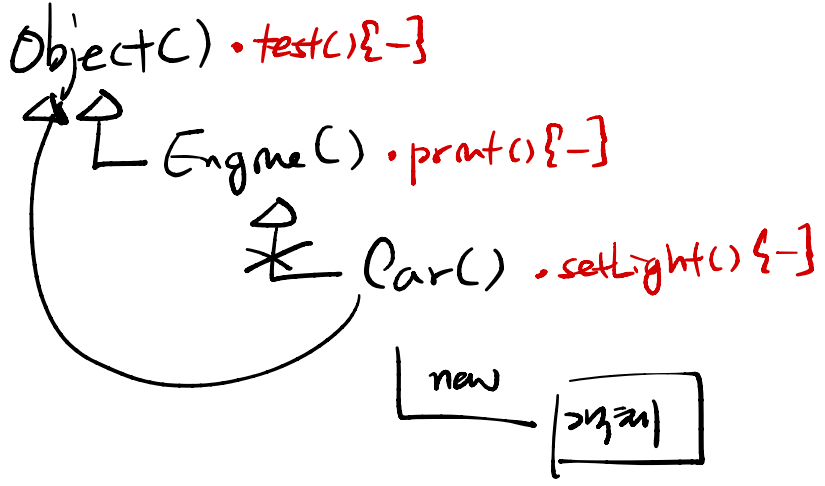
```
Engine() {
  Object.call();
  this.info = function(){-}
}
```

⇓ 생성자 반환

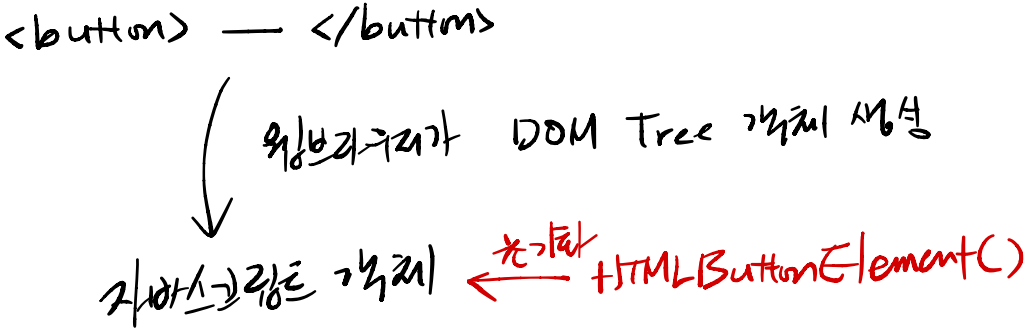


* 생성자 응용 II

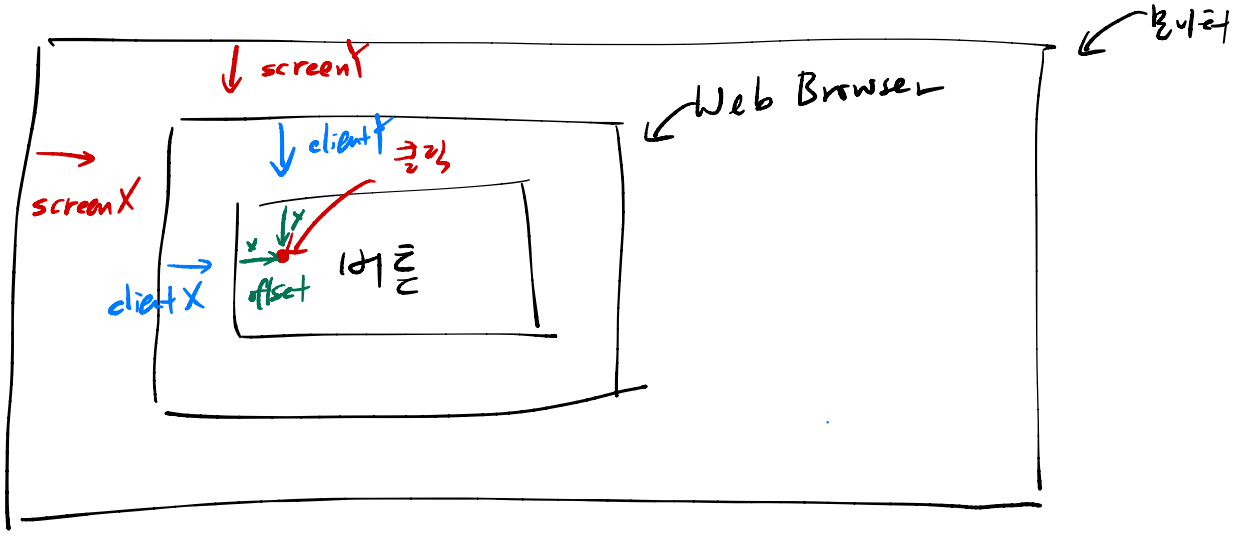
Car() —call→ Engine() —call→ Object()

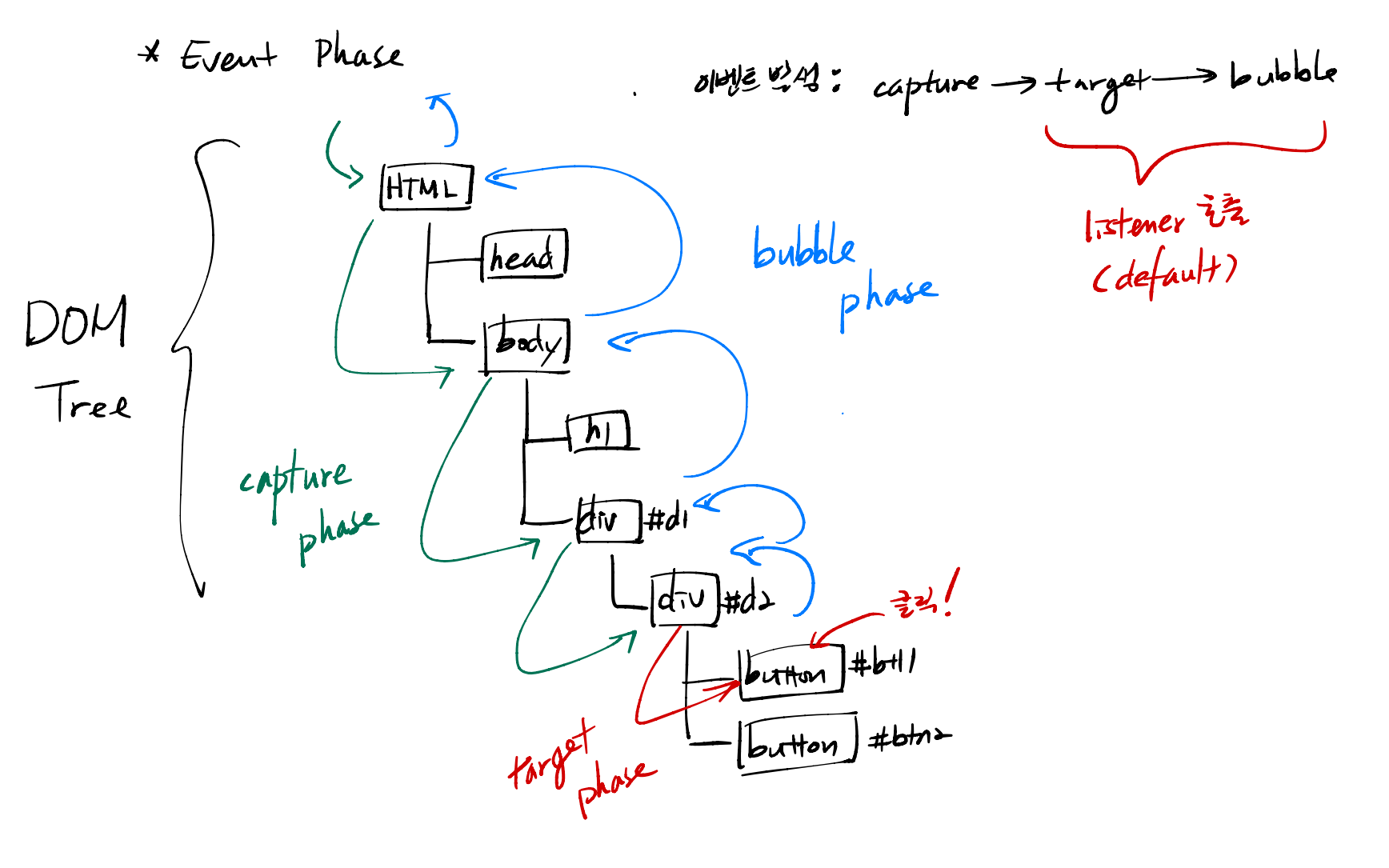


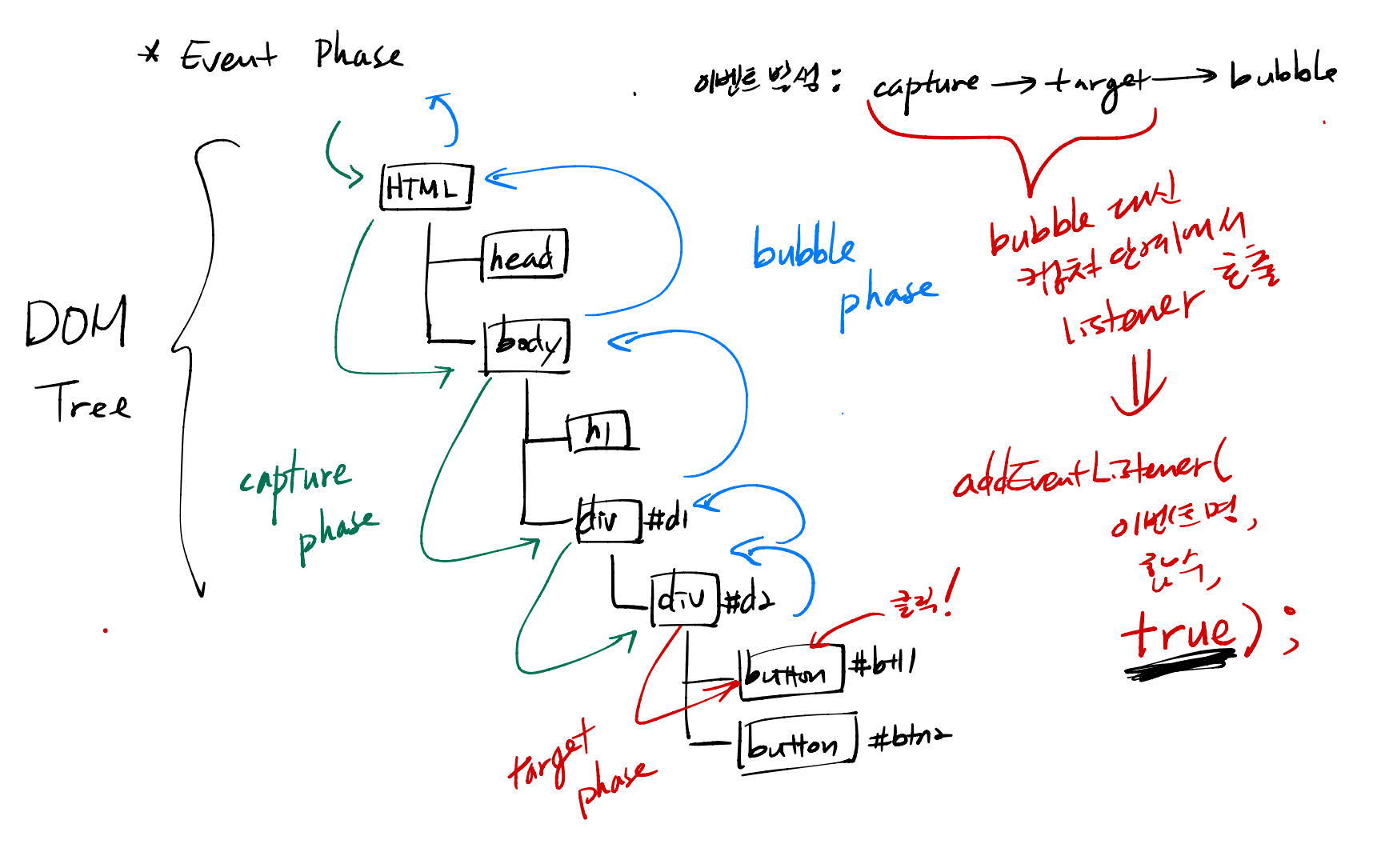
✗ 태그와 DOM API


<button> — </button>
웹브라우저가 DOM Tree 객체 생성
자바스크립트 객체
※ 참고
+HTMLButtonElement()

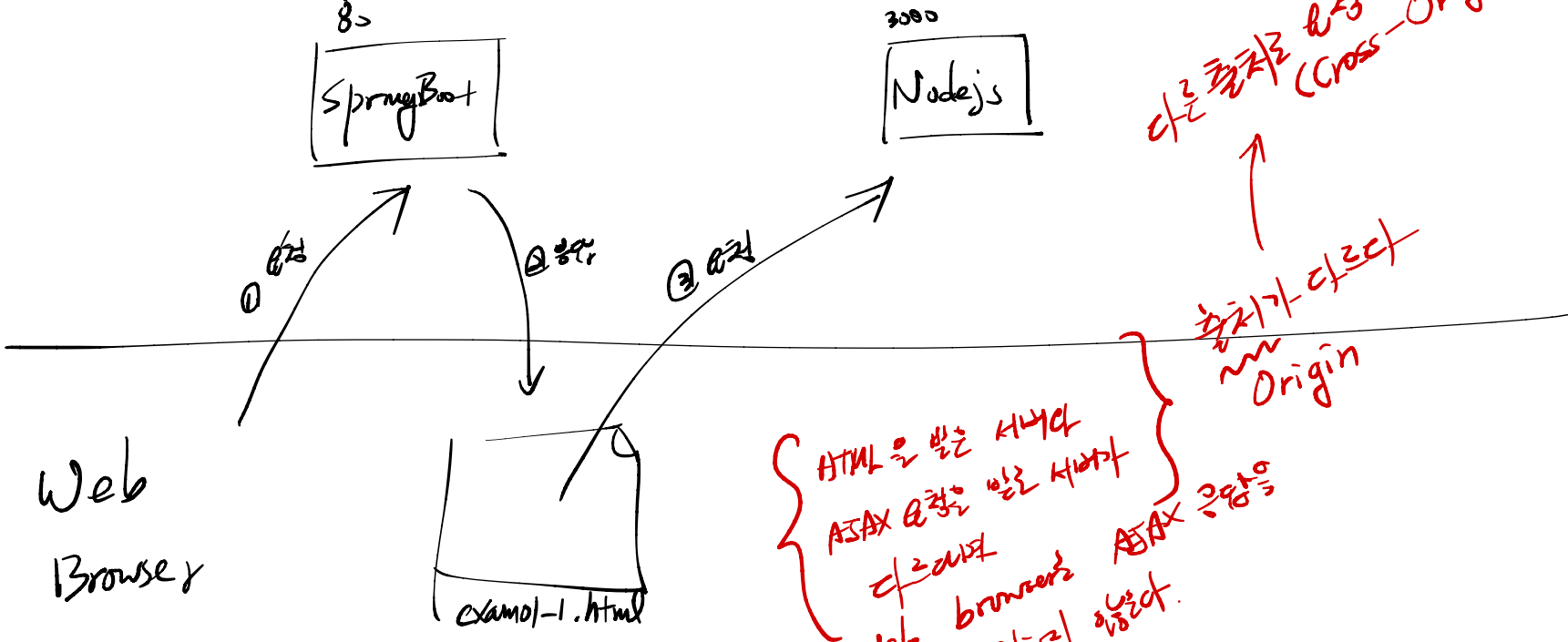
* Event 객체 - click 이벤트 활용



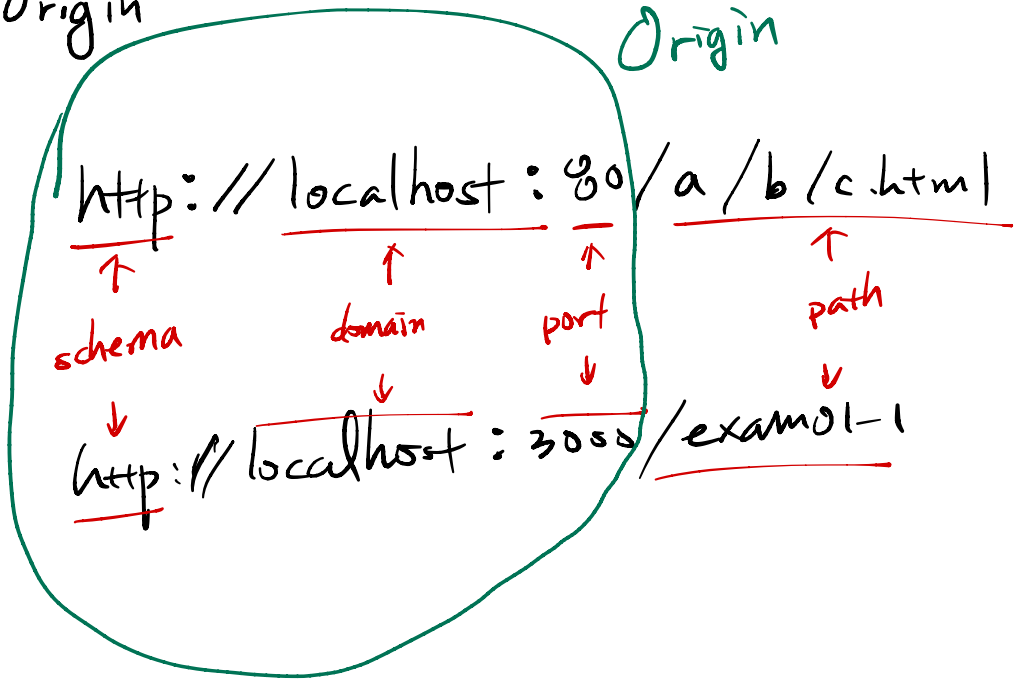

* Event Phase
이벤트 발생: capture → target → bubble
listener 등록 (default)
DOM Tree
HTML
head
body
h1
div #d1
div #d2
button #btn1
button #btn2
bubble phase
capture phase
target phase
클릭!


* Event Phase
이벤트 발생: capture → target → bubble
DOM Tree
HTML
head
body
h1
div #d1
div #d2
button #btn1
button #btn2
capture phase
target phase
bubble phase
클릭!
bubble 단계에 반응하고 싶을 때 listener 등록
addEventListener(이벤트명, 함수, true);

* AJAX 요청


8>
SpringBoot
3000
Nodejs
① 요청
② 응답
③ 요청
Web
Browser
example-1.html
다른 출처로 요청 (Cross-Origin)
출처가 다르다
Origin
HTML을 받은 서버와
AJAX 요청을 받은 서버가
다르므로
Web browser AJAX 금지
차단되어 실행된다.

* Origin


Origin
http://localhost:80/a/b/c.html
schema
domain
port
path
http://localhost:3000/examo1-1

* Proxy 서버를 이용한 CORS 문제 해결