

23장 실행 컨텍스트

소스 코드의 타입

전역 코드

함수 코드

eval 코드

모듈 코드

소스코드의 평가와 실행

소스 코드의 평가

평가 결과

실행 컨텍스트

정보

실행 결과

1. 변수, 함수 선언문 실행 컨텍스트가 관리하는 스코프에 등록

2. 런타임

소스 코드의 실행

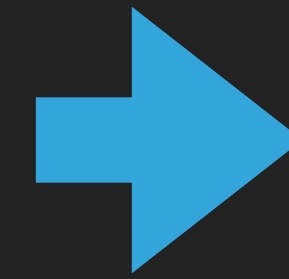
실행 컨텍스트, CALL 스택

코드 실행하려면?

실행중인 코드 순서 변경, 되돌아가기

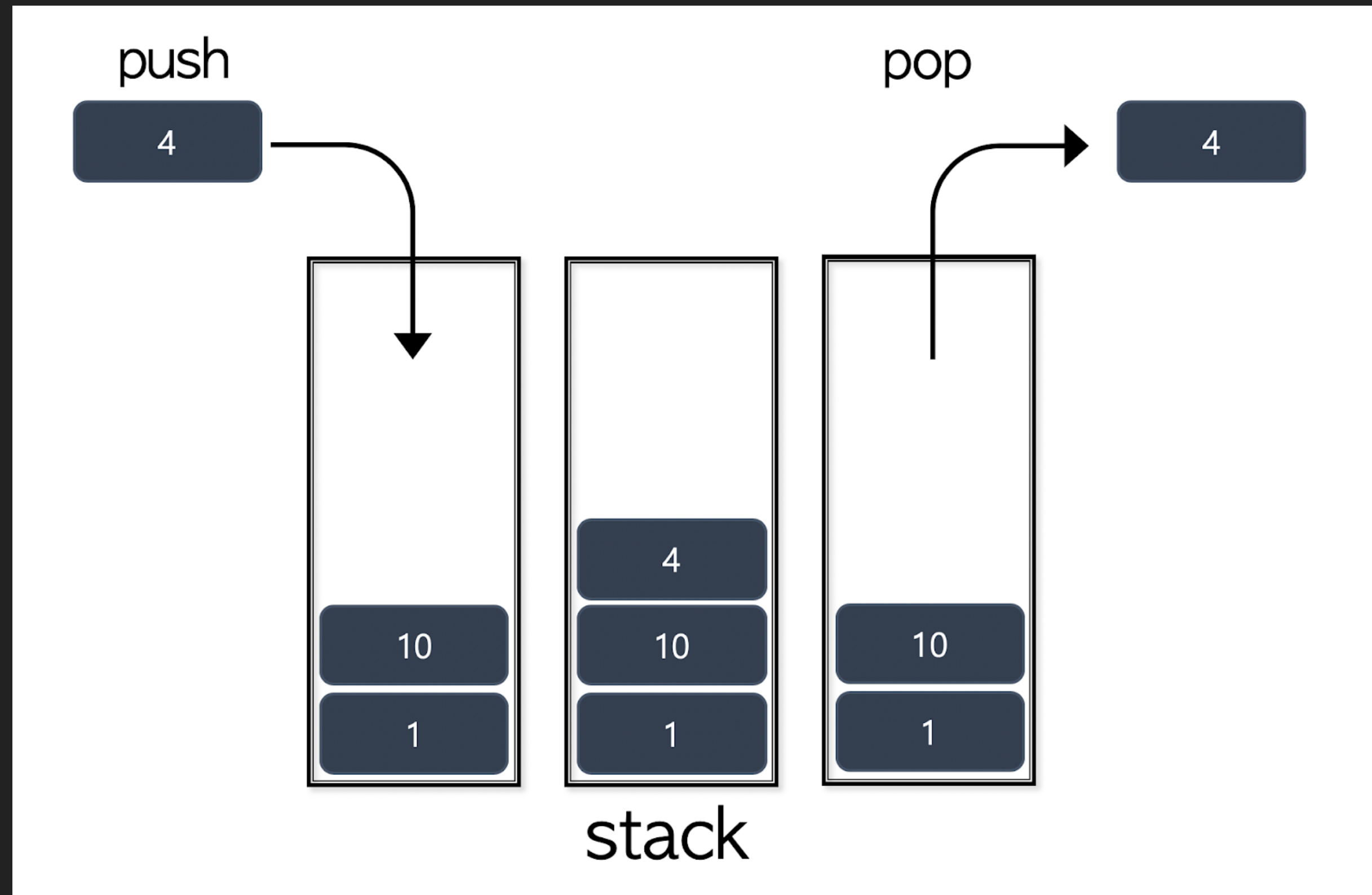
식별자 검색이 가능하도록 스코프 체인 형성

식별자를 스코프별로 구분, 상태변화 관리



실행 컨텍스트

실행 컨텍스트, CALL 스택



Call stack

BAR() 실행 컨텍스트

FOO() 실행 컨텍스트

전역 실행 컨텍스트

소스코드의 평가와 실행

1. 전역 코드 평가

```
const x = 1;    ← 선언
const y = 2;    ← 선언

function foo(a) { ← 선언
  const x = 10;
  const y = 20;

  console.log(a + x + y);
}

foo(100);
console.log(x + y);
```

Call stack

전역 실행 컨텍스트

소스코드의 평가와 실행

1. 전역 코드 실행

```
const x = 1;  ← 1. 값 저장  
const y = 2;  ← 2. 값 저장
```

```
function foo(a) {  
  const x = 10;  
  const y = 20;  
  
  console.log(a + x + y);  
}
```

```
foo(100);  ← 3. 함수!?  
console.log(x + y);
```

Call stack

전역 실행 컨텍스트

소스코드의 평가와 실행


1. 함수 코드 평가

```
const x = 1;
const y = 2;

function foo(a) {
  const x = 10;
  const y = 20;

  console.log(a + x + y);
}

foo(100);
console.log(x + y);
```



Call stack

함수 실행 컨텍스트

전역 실행 컨텍스트

소스코드의 평가와 실행

1. 함수 코드 실행

```
const x = 1;
const y = 2;

function foo(a) {
  const x = 10;
  const y = 20;
  console.log(a + x + y);
}

foo(100);
console.log(x + y);
```

← 1. 할당

← 2. 할당

← 3. 할당

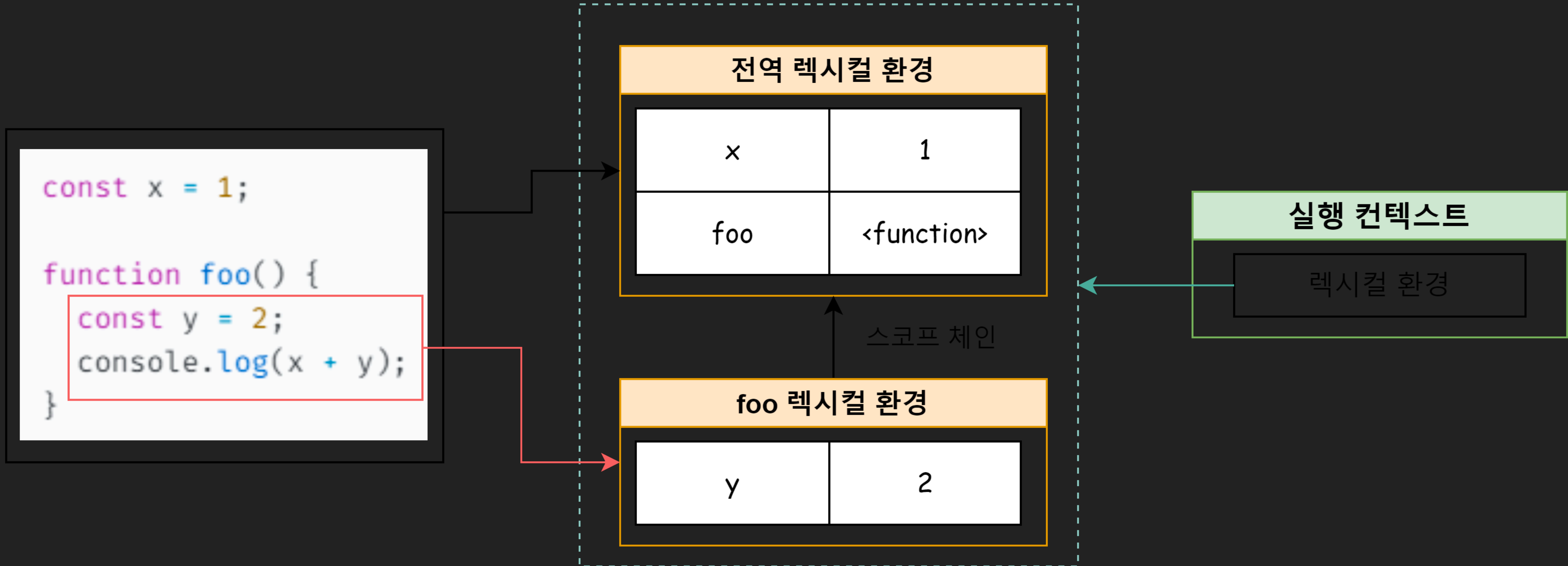
← 4. 실행

Call stack

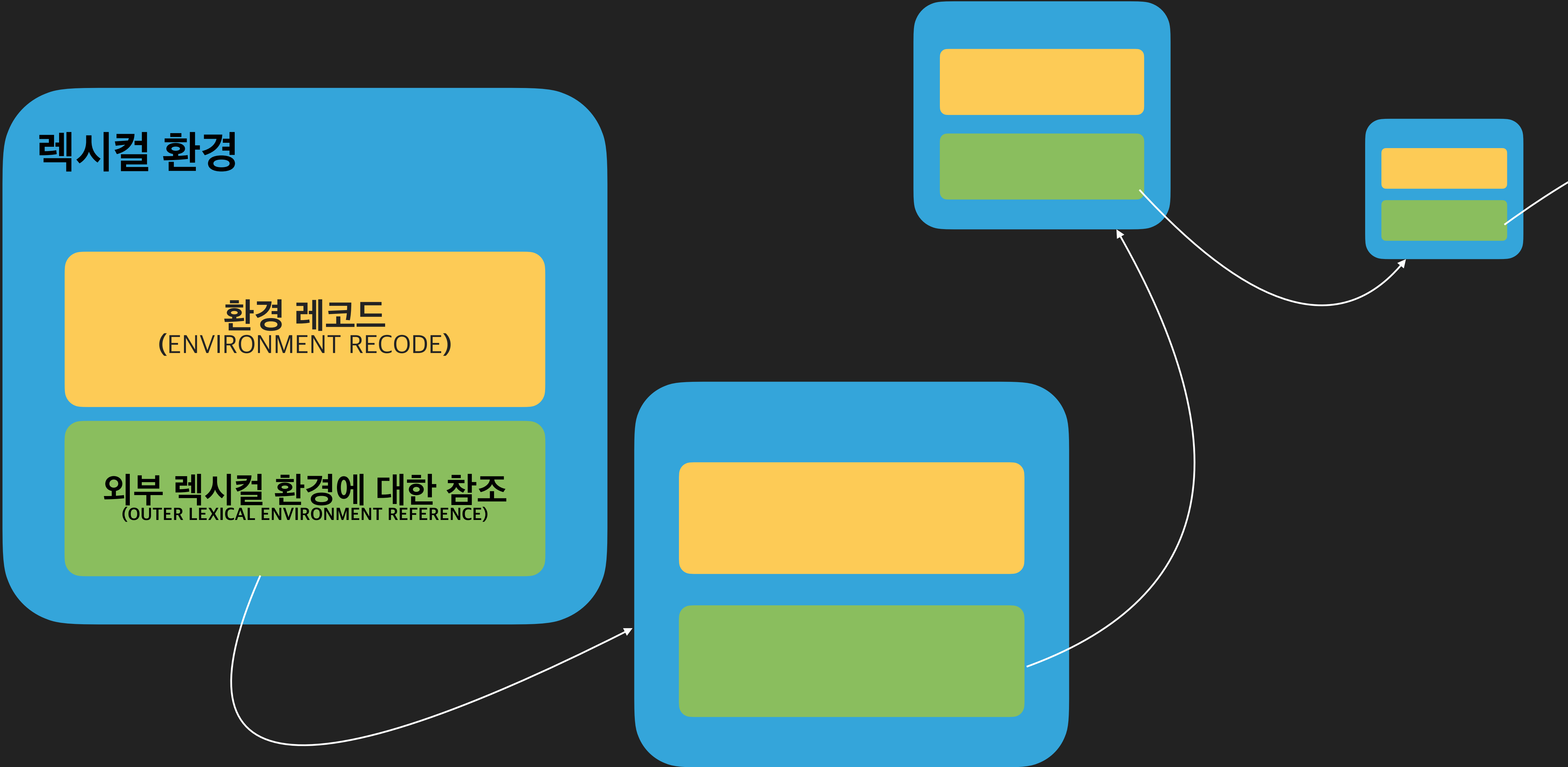
함수 실행 컨텍스트

전역 실행 컨텍스트

렉시컬 환경



렉시컬 환경



실행 컨텍스트의 생성과 식별자 검색 과정

```
var x = 1;
const y = 2;

function foo(a) {
  var x = 3;
  const y = 4;

  function bar(b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }

  bar(10);
}

foo(20);
```

1. 전역 객체 생성
2. 전역 코드 평가
3. 전역 코드 실행
4. 함수 코드 평가
5. 함수 코드 실행
6. 함수 코드 종료
7. 전역 코드 종료

1. 전역 객체 생성

```
var x = 1;
const y = 2;

function foo(a) {
  var x = 3;
  const y = 4;

  function bar(b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }

  bar(10);
}

foo(20);
```

window

fetch

console

DOM

2. 전역 코드 평가

```
var x = 1;
const y = 2;

function foo(a) {
  var x = 3;
  const y = 4;

  function bar(b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }

  bar(10);
}

foo(20);
```

📌 전역 코드 평가 순서

1. 전역 실행 컨텍스트 생성

2. 전역 렉시컬 환경 생성

전역 환경 레코드 생성

- 객체 환경 레코드 생성
- 선언적 환경 레코드 생성

this바인딩

+ :: 외부 렉시컬 환경에 대한 참조 결정

2. 전역 코드 평가 - 실행 컨텍스트, 렉시컬 환경 생성

Call stack

전역 실행 컨텍스트

전역 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)



2. 전역 코드 평가 - 객체 환경 레코드 생성

Call stack

전역 실행 컨텍스트

전역 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

객체 환경 레코드

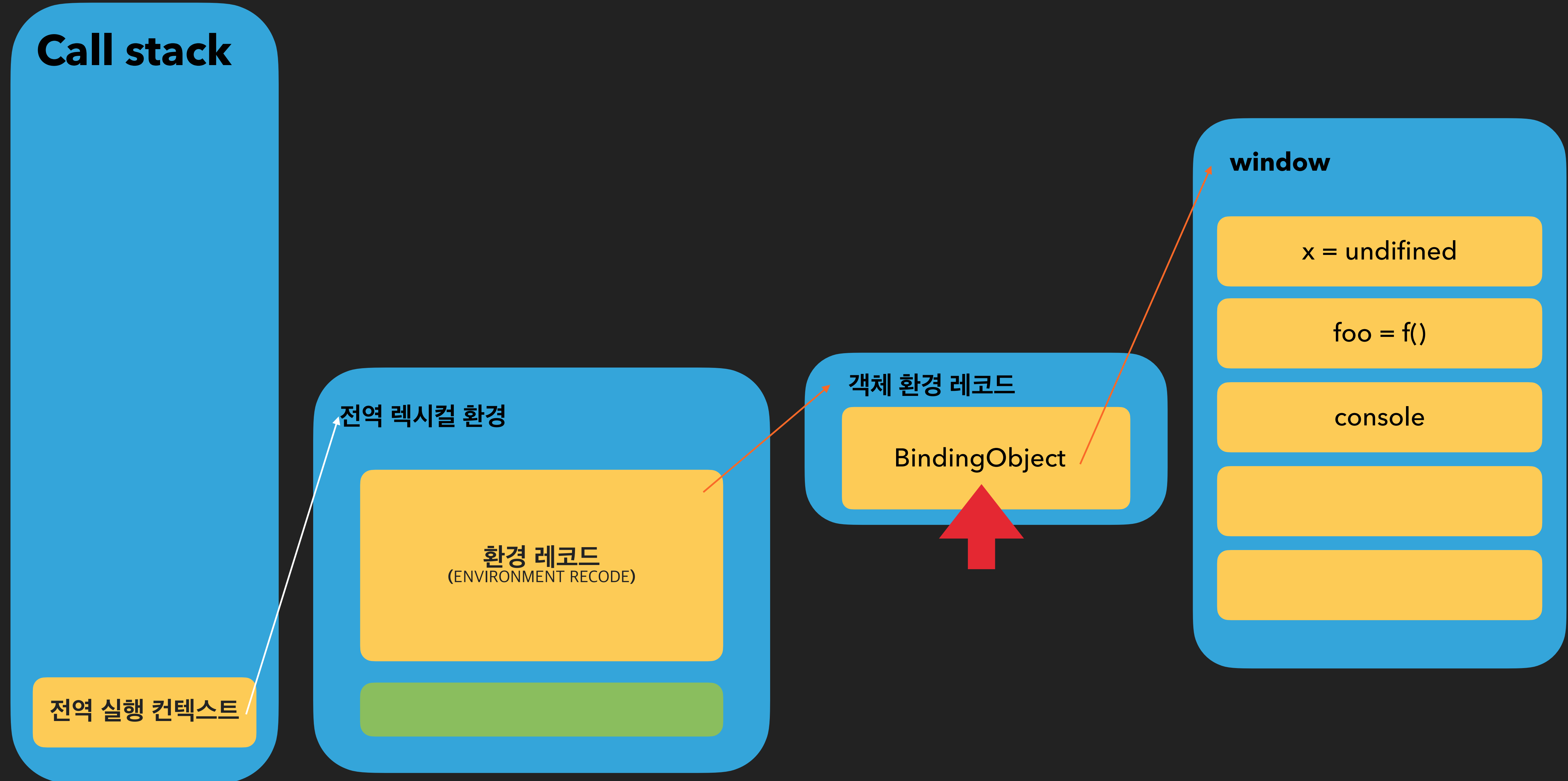
BindingObject

window

x = undefined

foo = f()

console



2. 전역 코드 평가 - 선언적 환경 레코드 생성, THIS 바인딩

Call stack

전역 실행 컨텍스트

전역 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

객체 환경 레코드

BindingObject

선언적 환경 레코드

y

window

x = undefined

foo = f()

console

2. 전역 코드 평가 - 외부 렉시컬 환경에 대한 참조

Call stack

전역 실행 컨텍스트

전역 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

NULL

객체 환경 레코드

BindingObject

선언적 환경 레코드

y

window

x = undefined

foo = f()

console

3. 전역 코드 실행

Call stack

```
var x = 1;
const y = 2;

function foo(a) {
  var x = 3;
  const y = 4;

  function bar(b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }

  bar(10);
}

foo(20);
```

전역 실행 컨텍스트

전역 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

NULL

객체 환경 레코드

BindingObject

선언적 환경 레코드

y = 2

window

x = 1

foo = f()

console



4. 함수 코드 평가

Call stack

전역 실행 컨텍스트



함수코드 평가 순서

1. 함수 실행 컨텍스트 생성
2. 함수 렉시컬 환경 생성

함수 환경 레코드 생성

this바인딩

외부 렉시컬 환경에 대한 참조 결정

4. 함수 코드 평가 - 실행 컨텍스트, 렉시컬 환경 생성

Call stack

FOO() 컨텍스트

전역 실행 컨텍스트

foo 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)



4. 함수 코드 평가 - 함수 환경 레코드 생성

Call stack

FOO() 컨텍스트

전역 실행 컨텍스트

foo 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

함수 환경 레코드

a = undefined

x = undefined

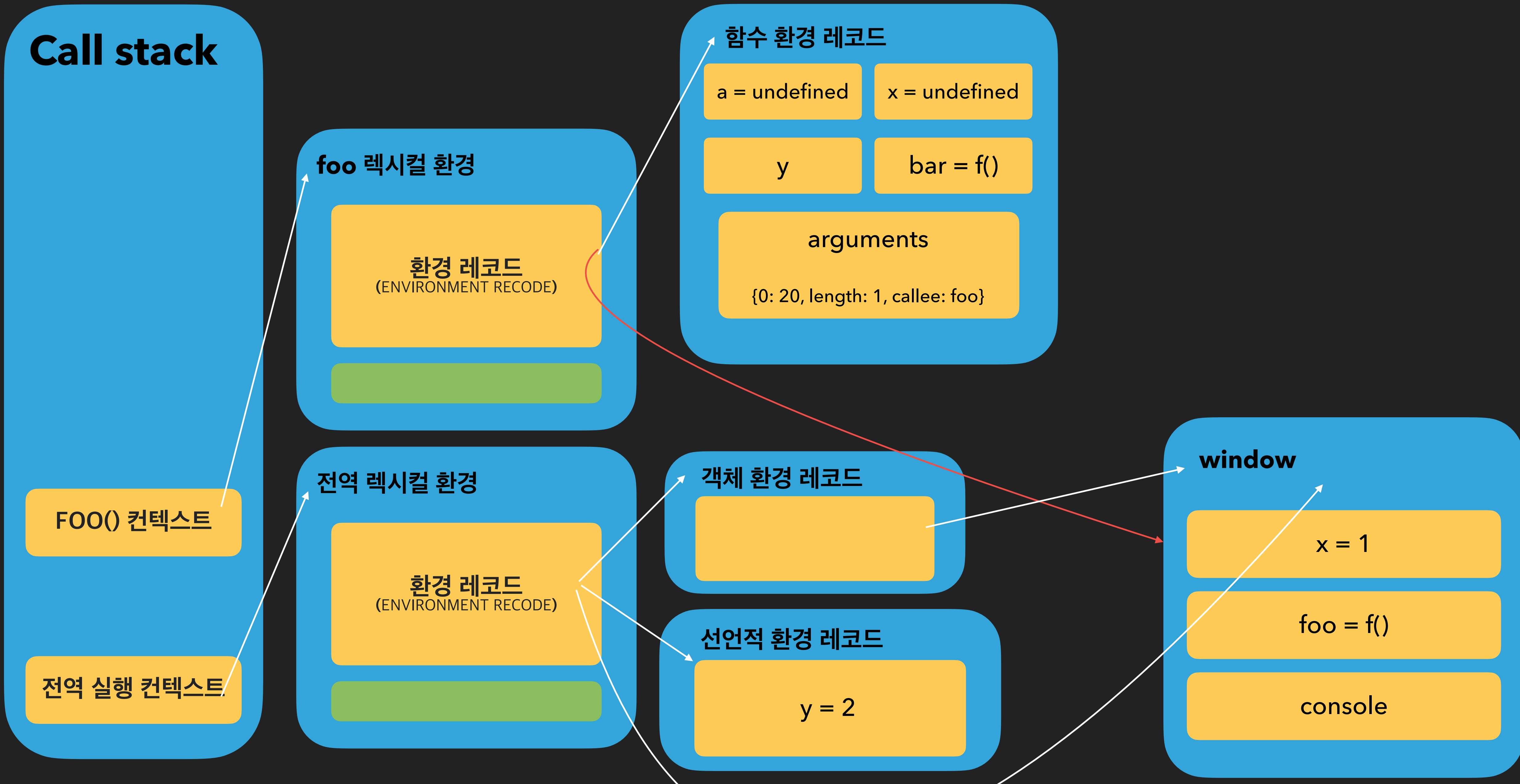
y

bar = f()

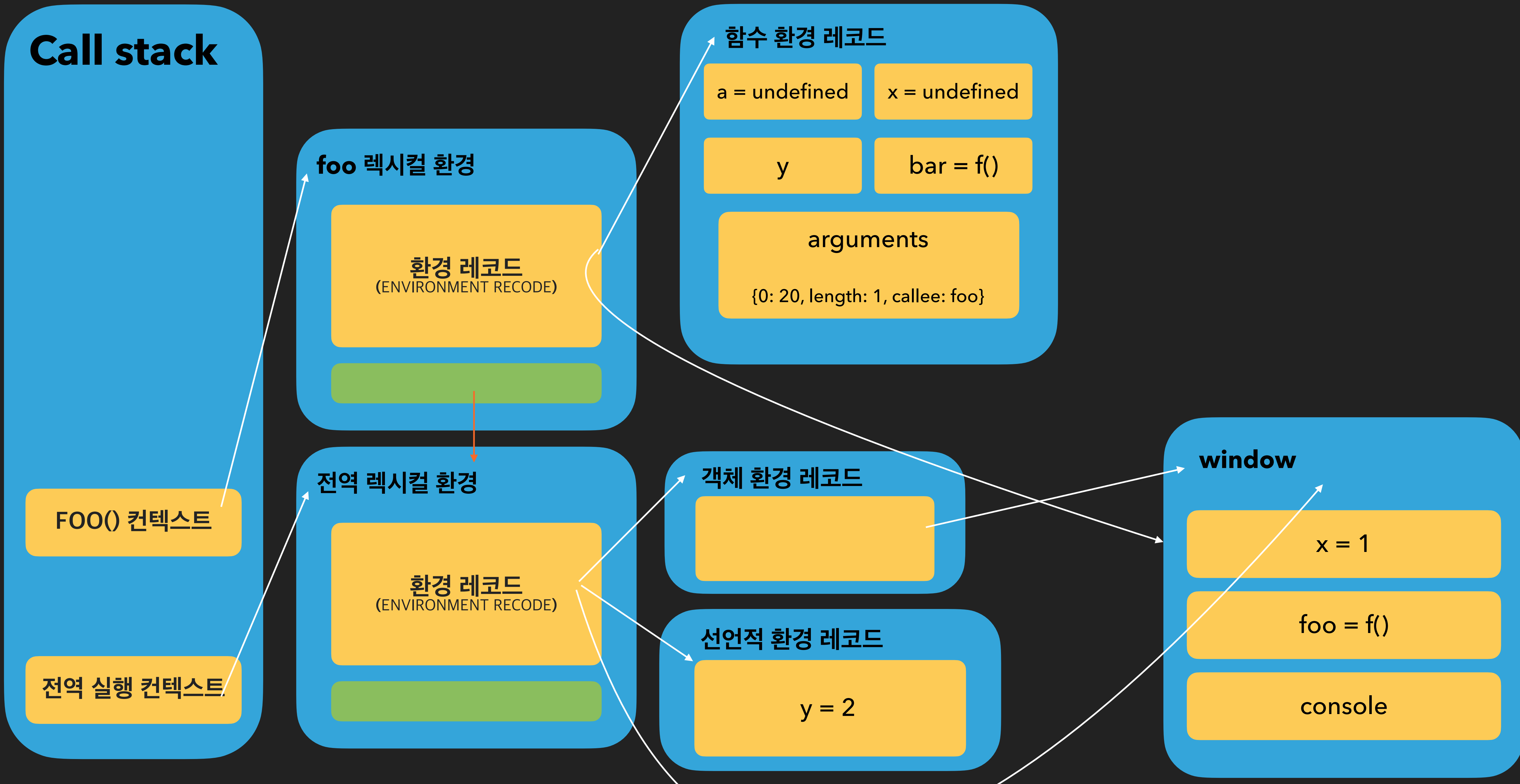
arguments

{0: 20, length: 1, callee: foo}

4. 함수 코드 평가 - THIS 바인딩



4. 함수 코드 평가 - 외부 렉시컬 환경에 한 참조 결정



5. 함수 코드 실행

Call stack

foo 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

함수 환경 레코드

a = 20

x = 3

y = 4

bar = f()

arguments

{0: 20, length: 1, callee: foo}

전역 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

객체 환경 레코드

선언적 환경 레코드

y = 2

window

x = 1

foo = f()

console

FOO() 컨텍스트

전역 실행 컨텍스트

```
var x = 1;
const y = 2;

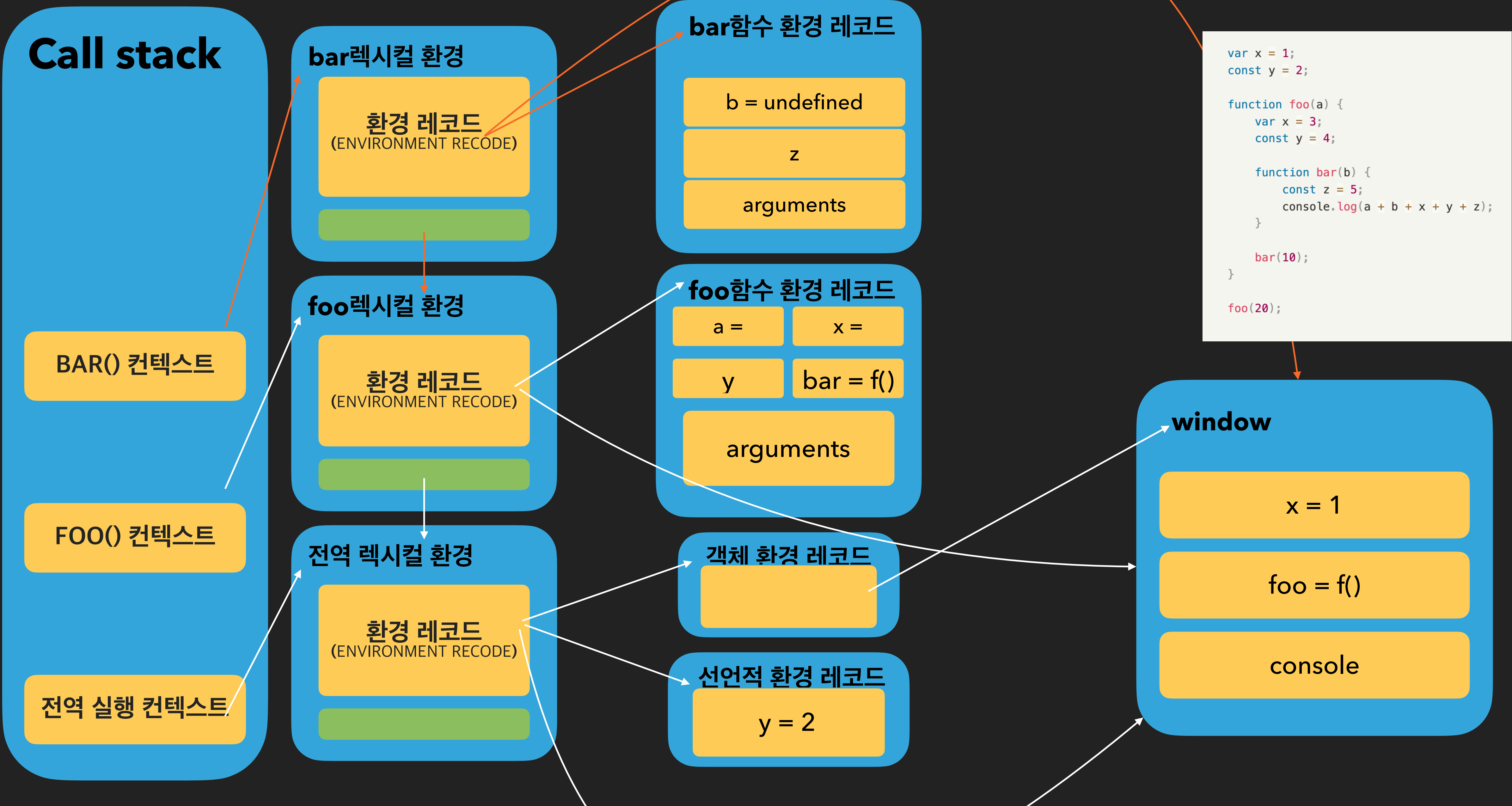
function foo(a) {
  var x = 3;
  const y = 4;

  function bar(b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }

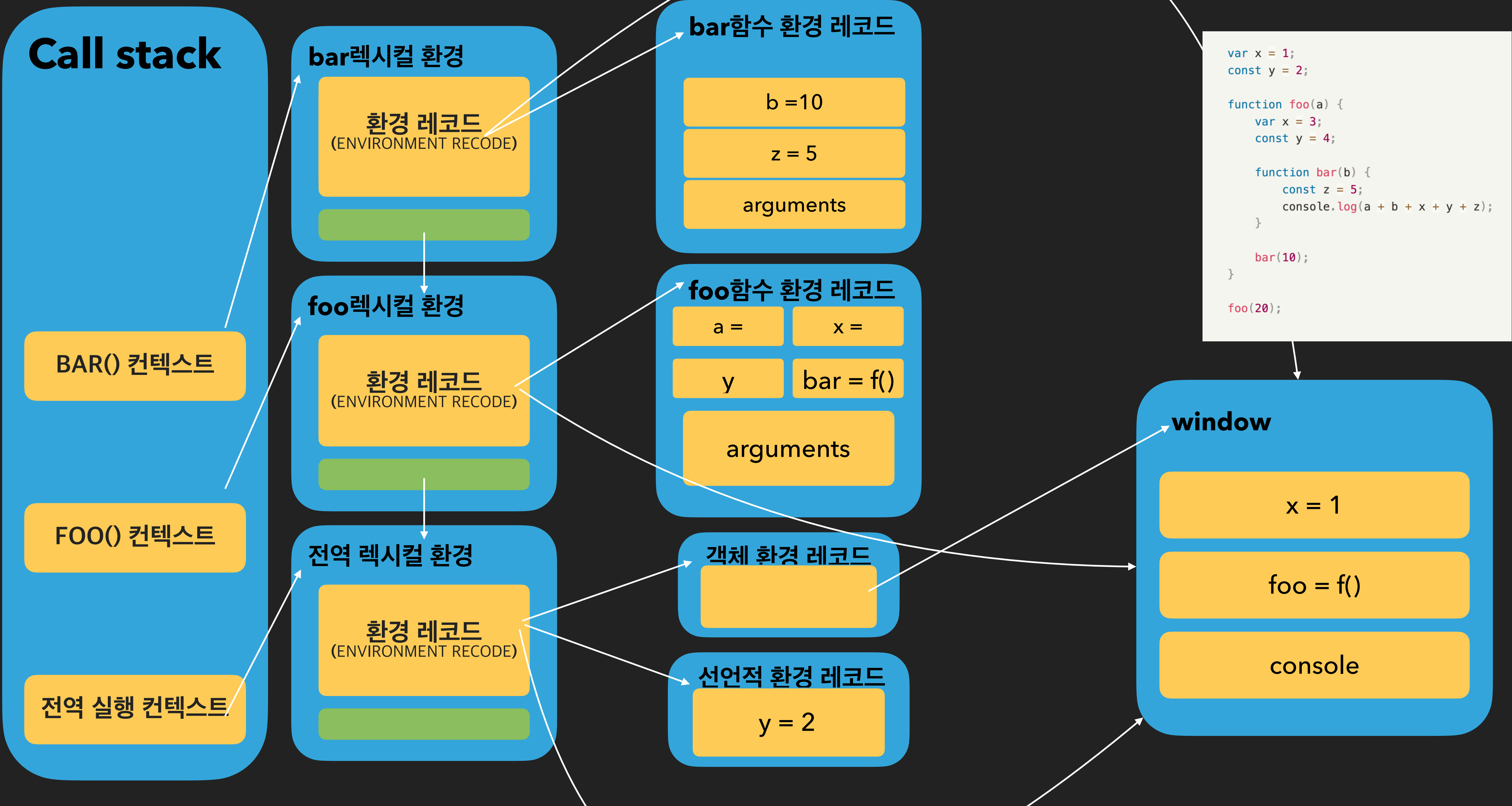
  bar(10);
}

foo(20);
```

5. 함수 코드 평가 X 2

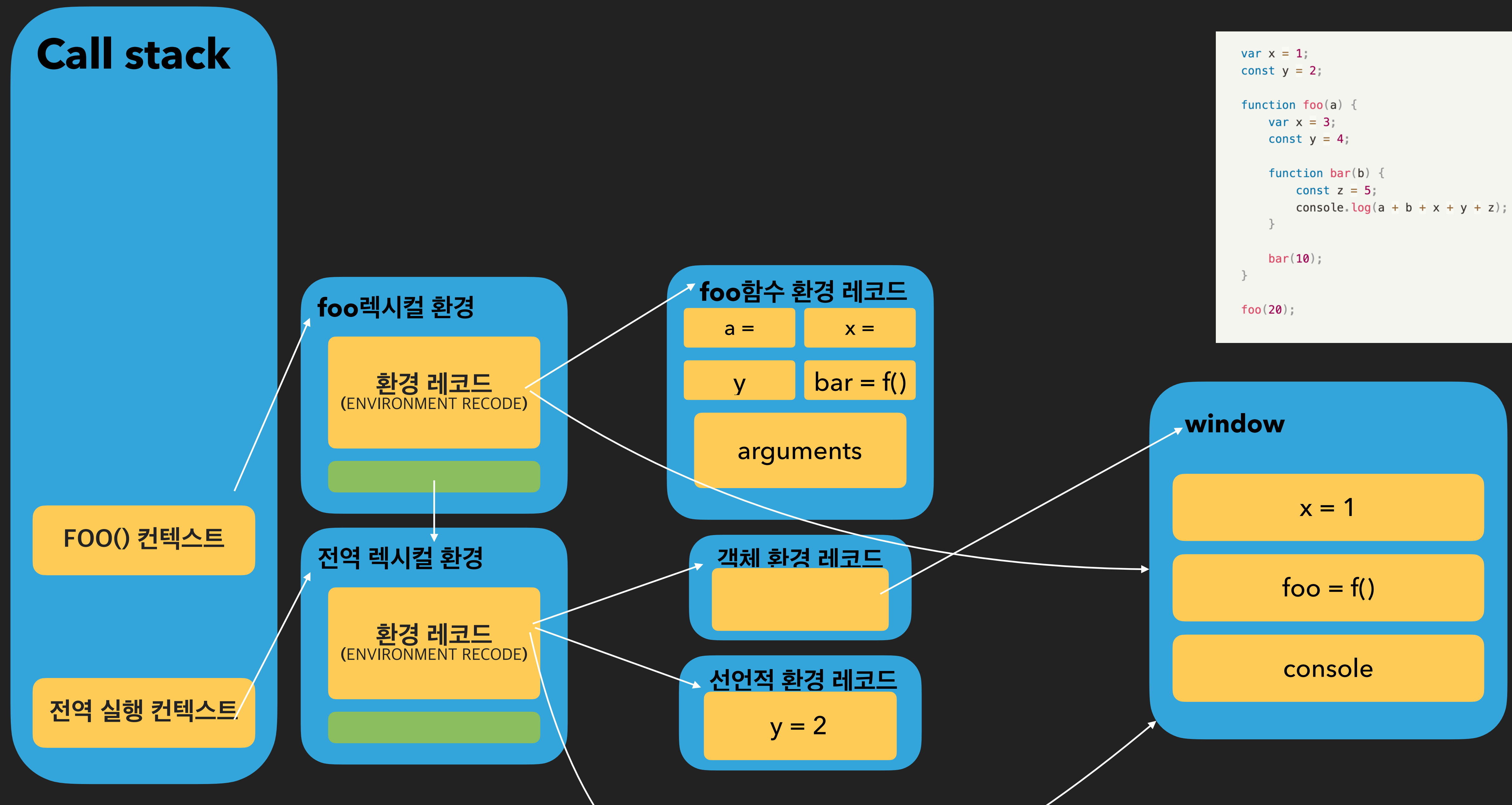


5. 함수 코드 실행 X 2



6. 함수 코드 종료

Call stack



6. 함수 코드 종료 X 2

Call stack

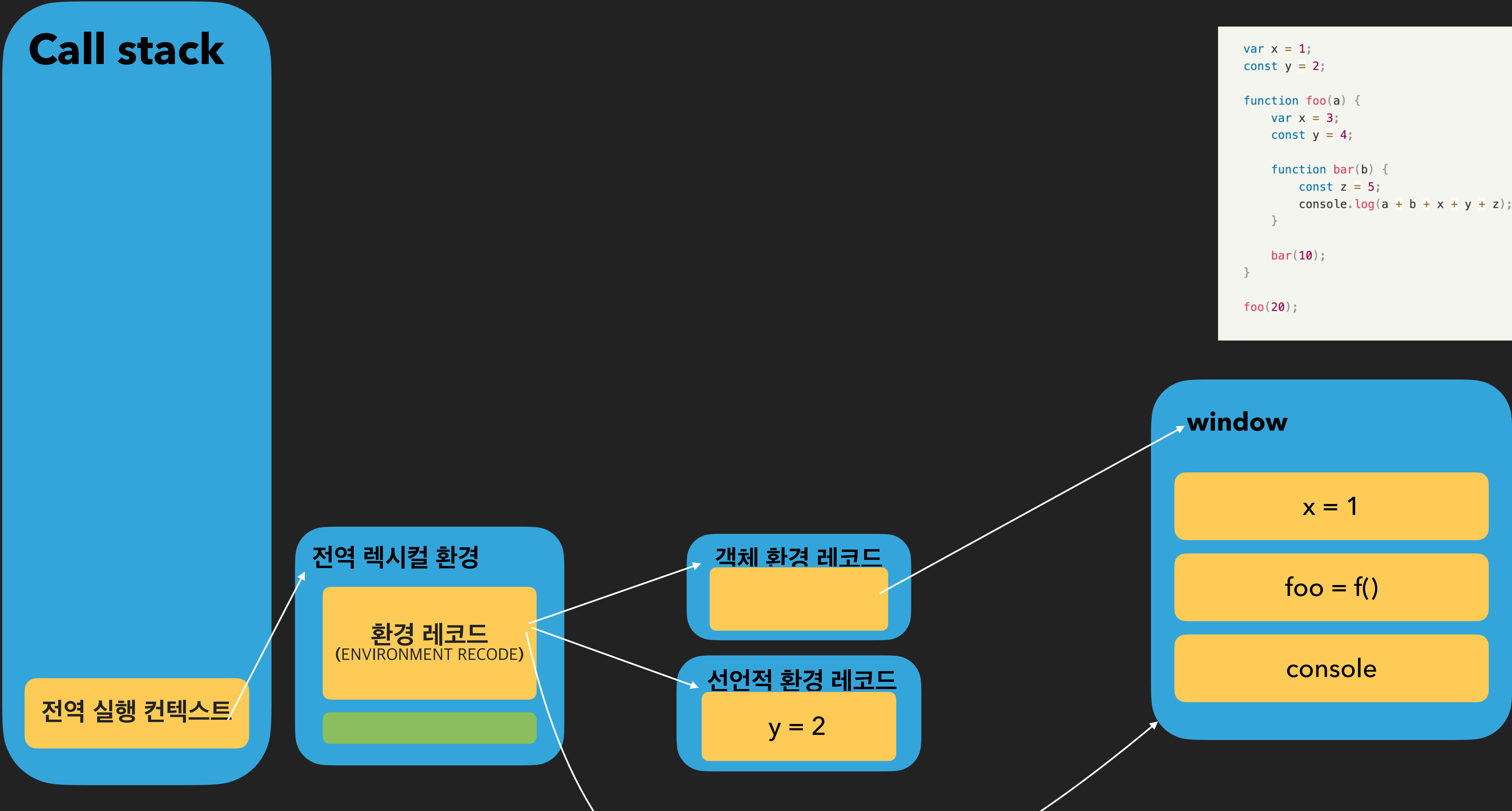
```
var x = 1;
const y = 2;

function foo(a) {
  var x = 3;
  const y = 4;

  function bar(b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }

  bar(10);
}

foo(20);
```



7. 전역 코드 종료

Call stack

```
var x = 1;
const y = 2;

function foo(a) {
  var x = 3;
  const y = 4;

  function bar(b) {
    const z = 5;
    console.log(a + b + x + y + z);
  }

  bar(10);
}

foo(20);
```

실행 컨텍스트와 블록 레벨 스코프

Call stack

```
let x = 1;

if (true) {
  let x = 10;
  console.log(x); // 10
}

console.log(x); // 1
```

전역 실행 컨텍스트

전역 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

NULL

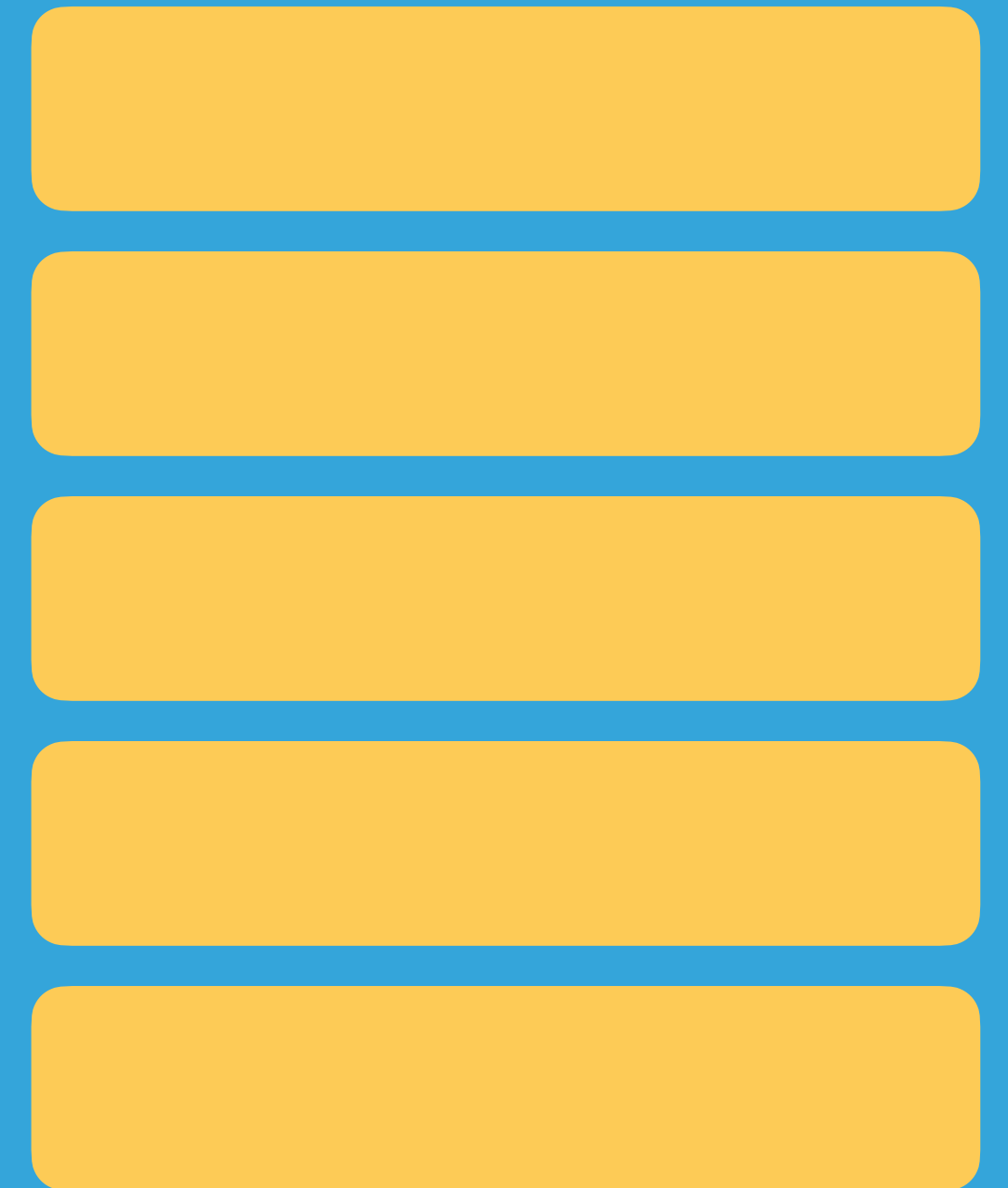
객체 환경 레코드

BindingObject

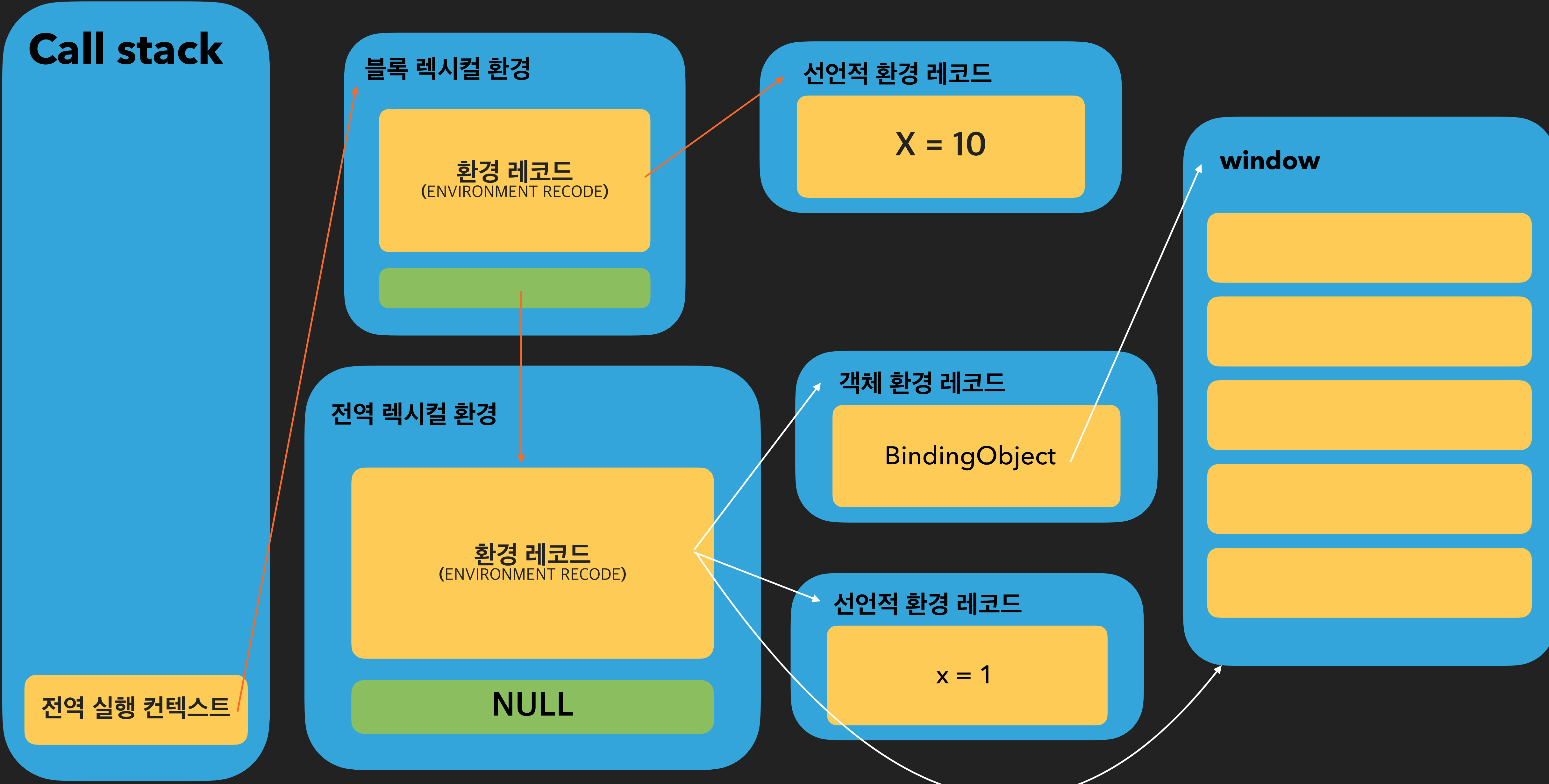
선언적 환경 레코드

x = 1

window



실행 컨텍스트와 블록 레벨 스코프



실행 컨텍스트와 블록 레벨 스코프

Call stack

블록 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

선언적 환경 레코드

X = 10

전역 렉시컬 환경

환경 레코드
(ENVIRONMENT RECODE)

객체 환경 레코드

BindingObject

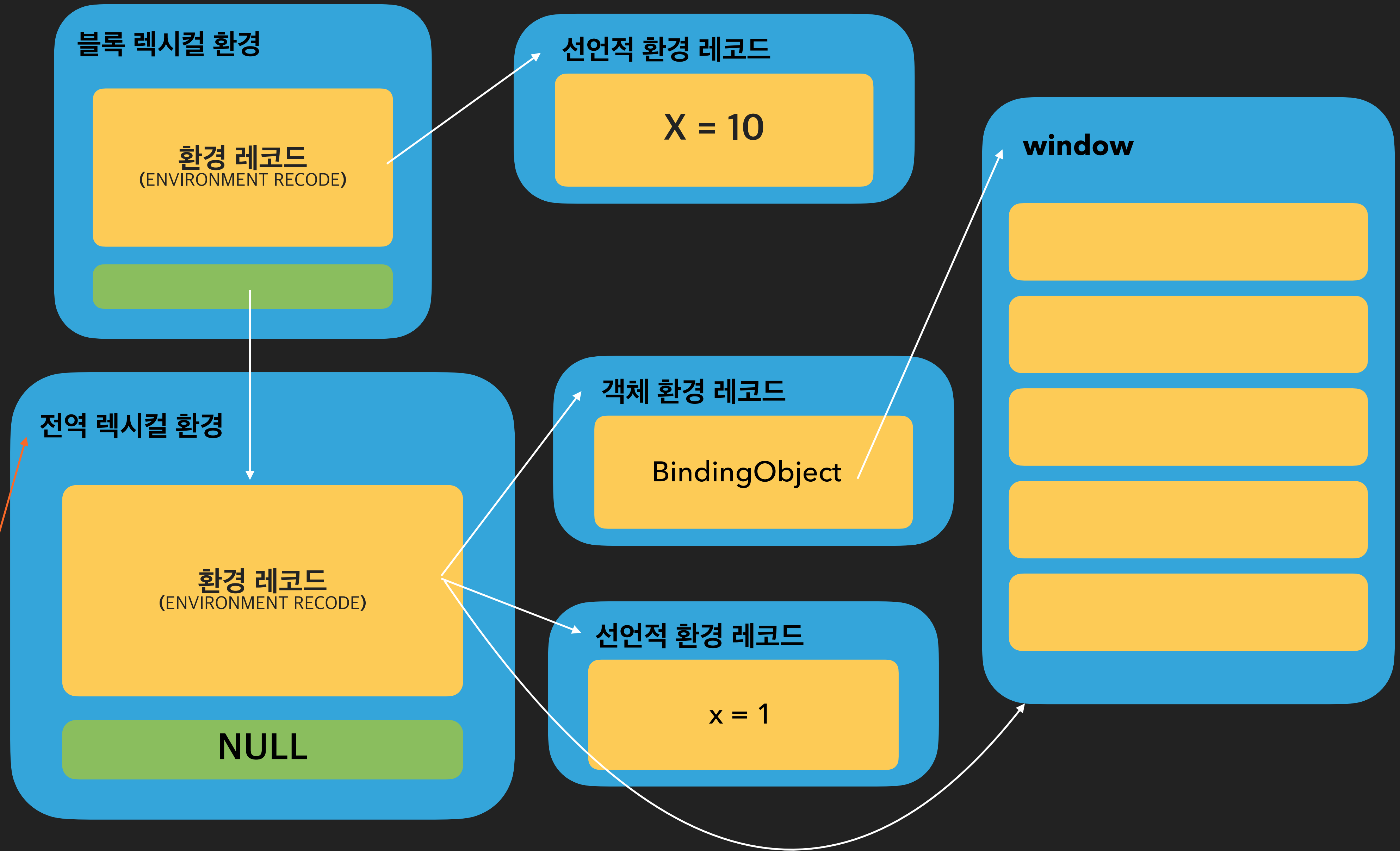
선언적 환경 레코드

x = 1

window

전역 실행 컨텍스트

NULL

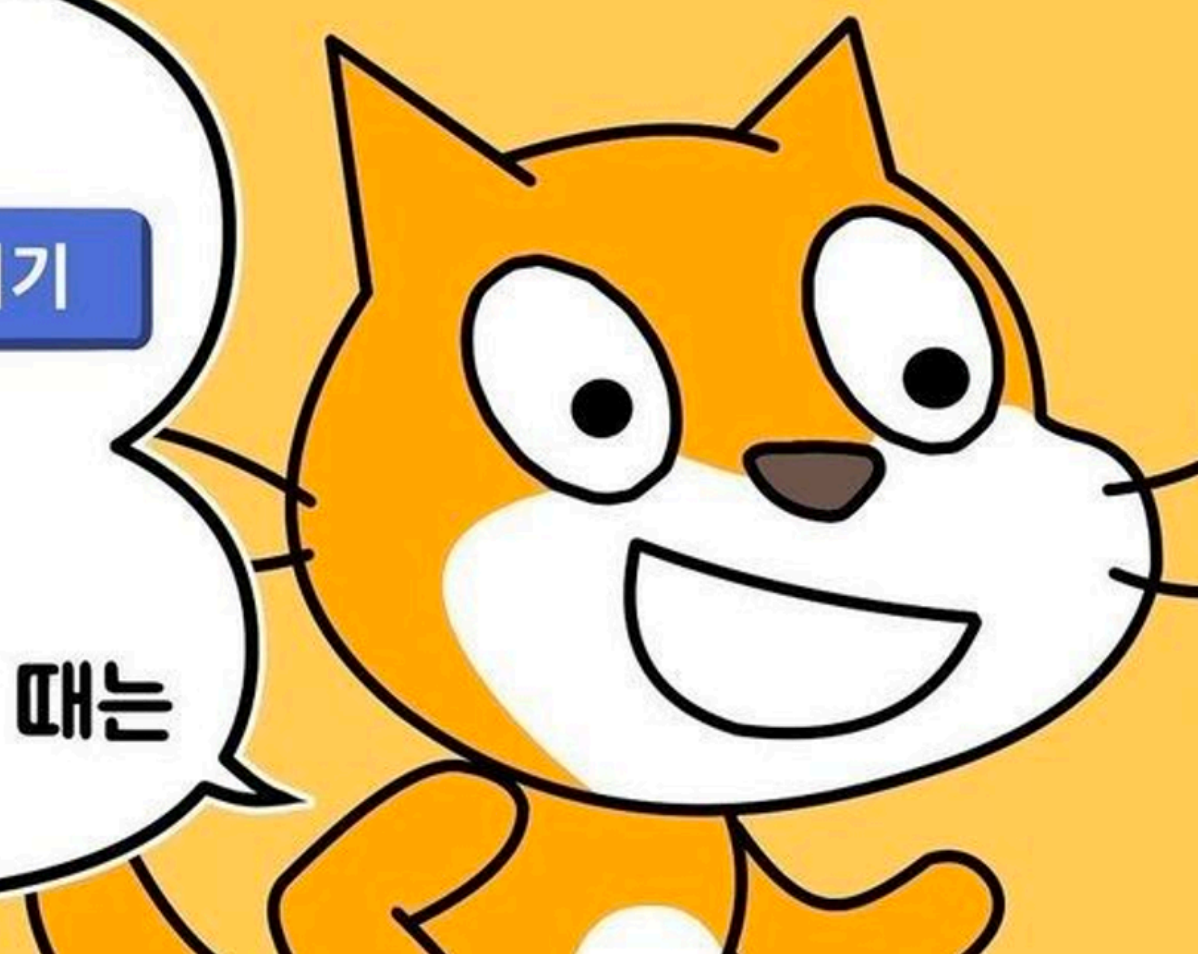


Event Loop

내가

10 만큼 움직이기

하면서
재롱 떨고 춤출 때는



개발이
재밌는 줄
알았지?

Branch: master v8 / test / mjsunit / closure.js
function f(n) {
 return function () {
 }
}



Q & A