

# 소유권

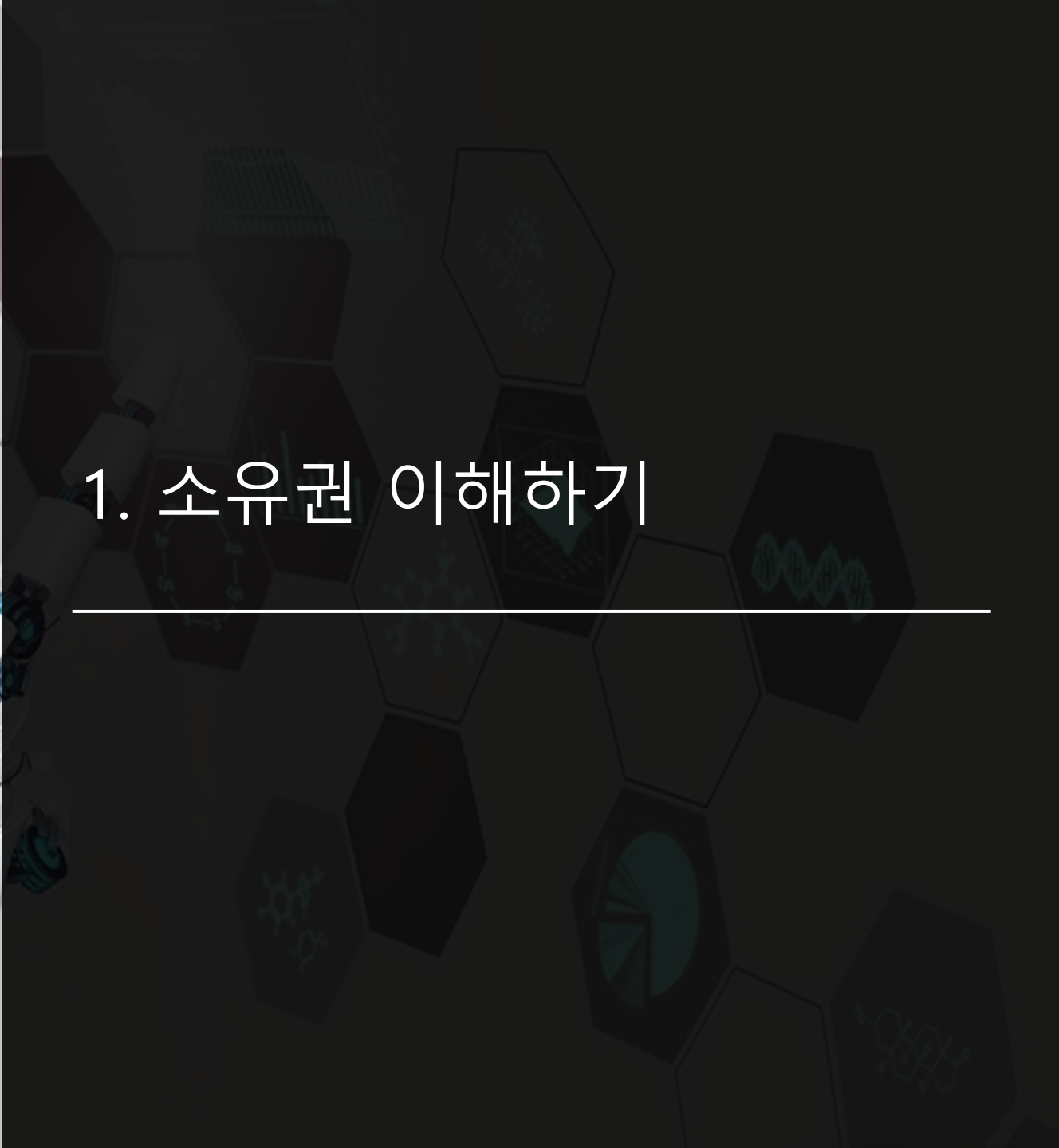
---

문용준



# 1. 소유권 이해하기

---



## 부동산의 소유자와 대여 관계

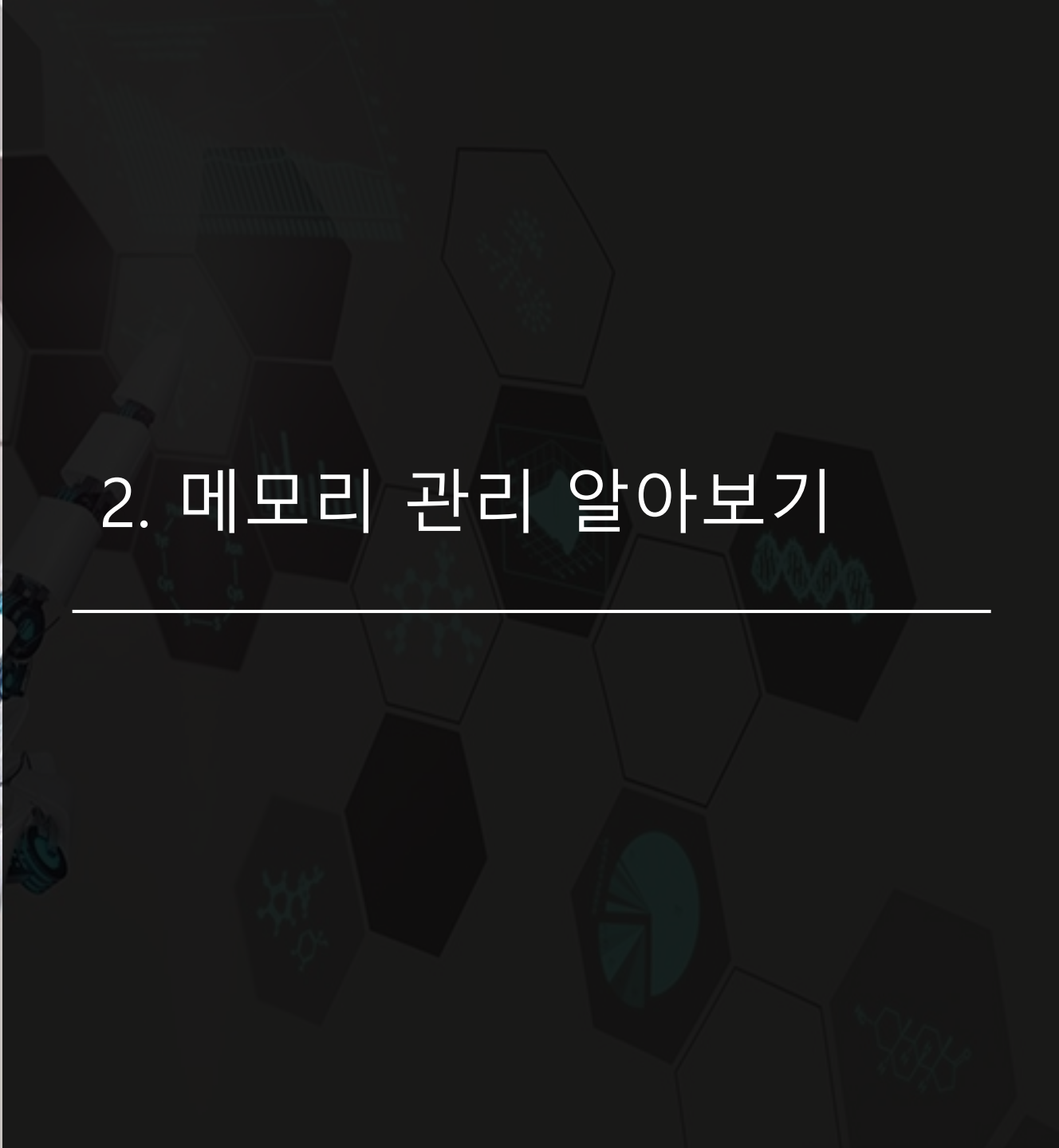
부동산등은 실제 소유자를 인정하고 소유자가 다른 사람에게 대여할 수 있다. 이때도 소유권에는 아무런 영향을 미치지 않는다. 실제 부동산은 소유권이 변동되어도 임차권은 변동되지 않을 수 있다.



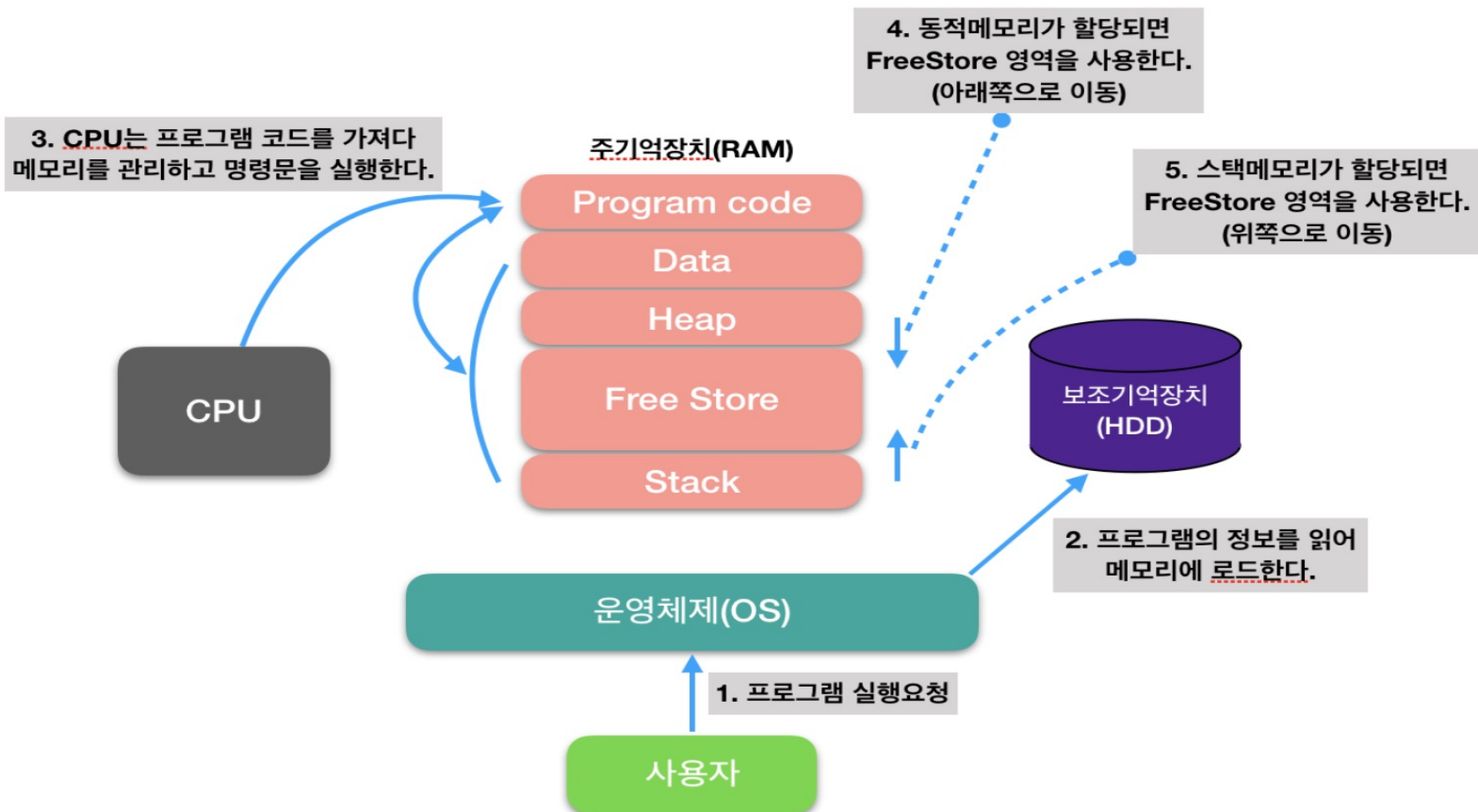


## 2. 메모리 관리 알아보기

---

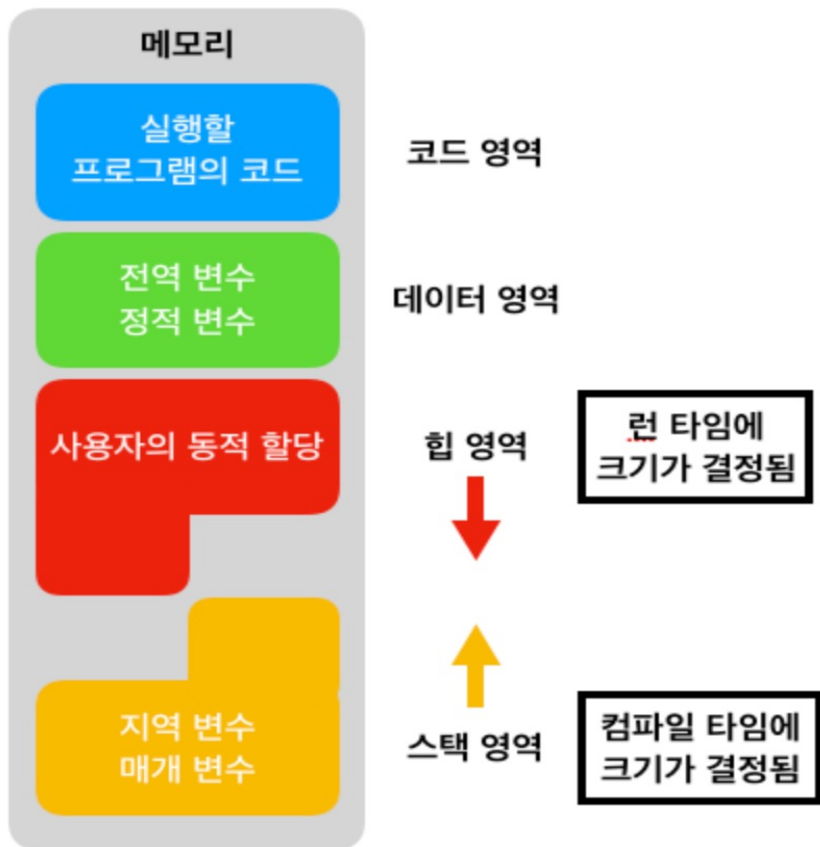


## 컴퓨터에서 메모리 처리



## ■ 프로그램에서 메모리 처리

프로그램이 실행될 때 가상 메모리를 기준으로 처리 처리



### • 데이터 영역

데이터 영역은 전역변수, static 변수를 위한 메모리 공간입니다. 이 둘의 공통점은 시작과 동시에 메모리에 올라가서 프로그램이 종료될 때 까지 남아있다는 것입니다.

### • 힙 영역

힙 영역은 필요할 때 동적으로 메모리를 할당하기 위한 영역으로 malloc 등으로 동적으로 선언된 변수들이 메모리 공간의 힙 영역에 저장됩니다. 즉, 할당되어야 할 메모리의 크기를 실행 시간 도중에 결정해야 할 경우가 이에 속하게 됩니다.

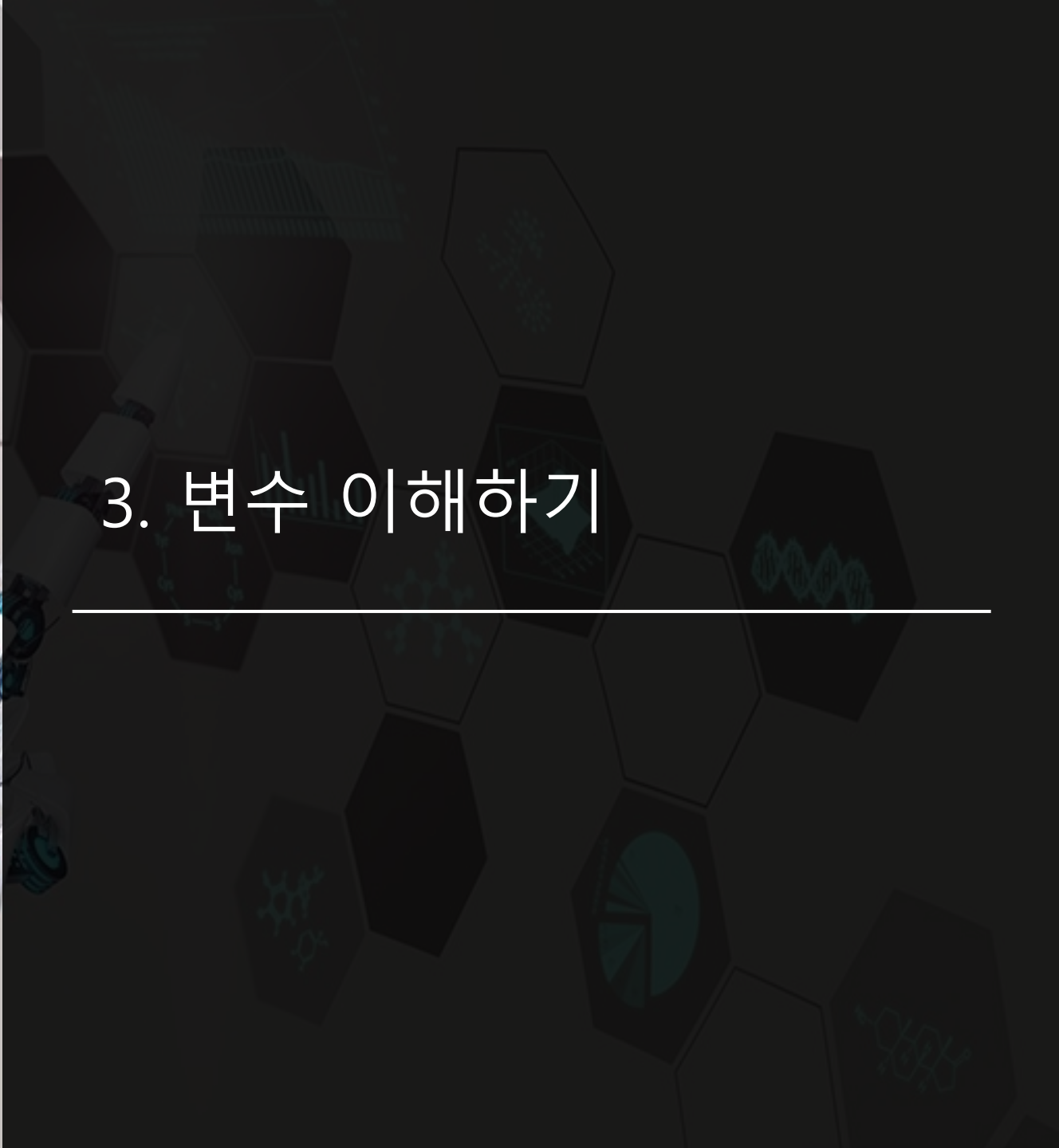
### • 스택 영역

스택 영역은 함수 호출 시 생성되는 지역 변수와 매개 변수가 저장되는 메모리 공간입니다. 이 스택 영역은 함수 호출이 시작되는 시점에 사용되는 지역 변수, 매개 변수들이 메모리의 스택 영역에 지정되고 함수 호출이 끝나는 시점에 소멸됩니다.



### 3. 변수 이해하기

---





## ■ 변수란

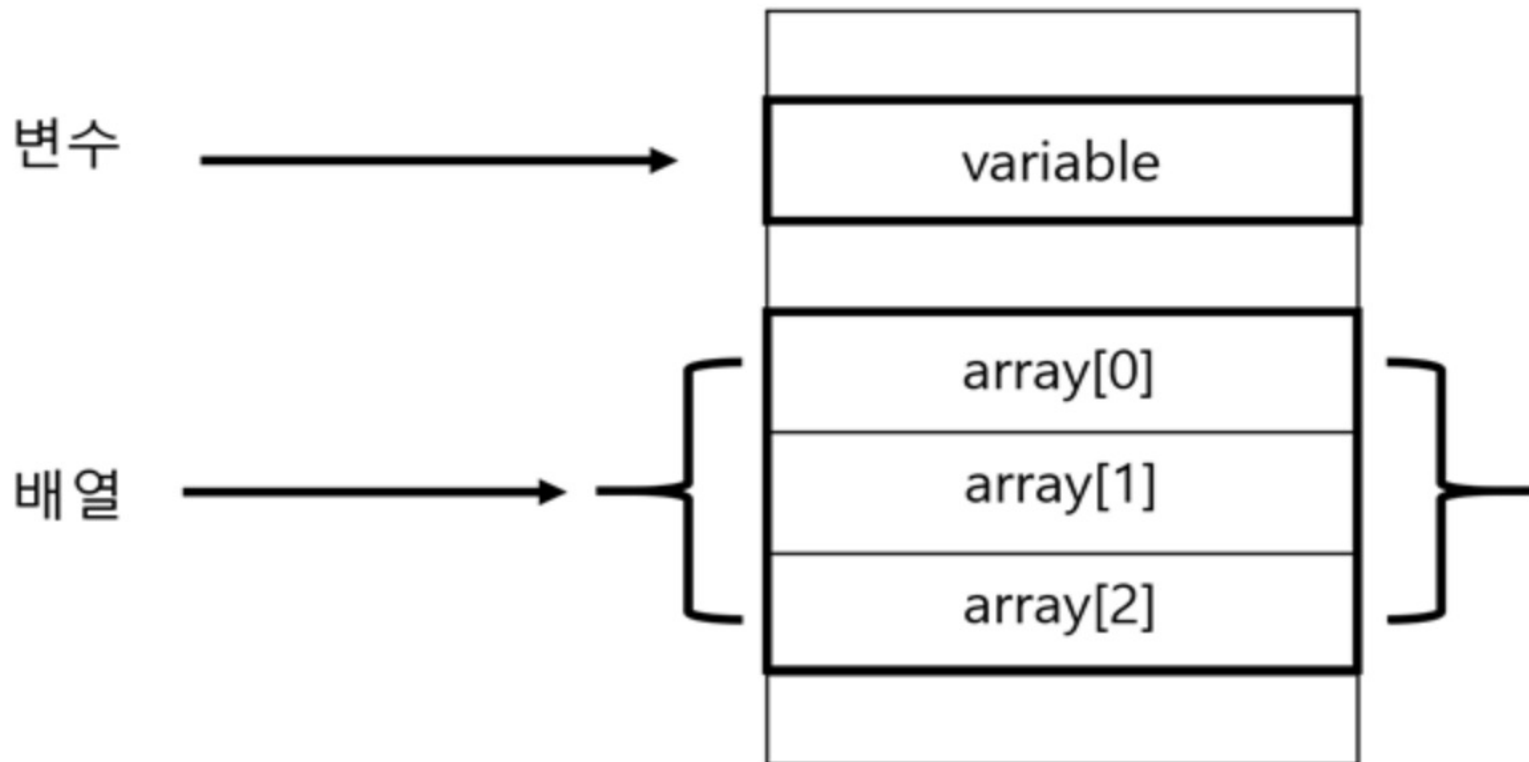
메모리 공간에 저장된 값을 식별할 수 있는 고유한 이름을 변수 이름(변수명)이라 한다





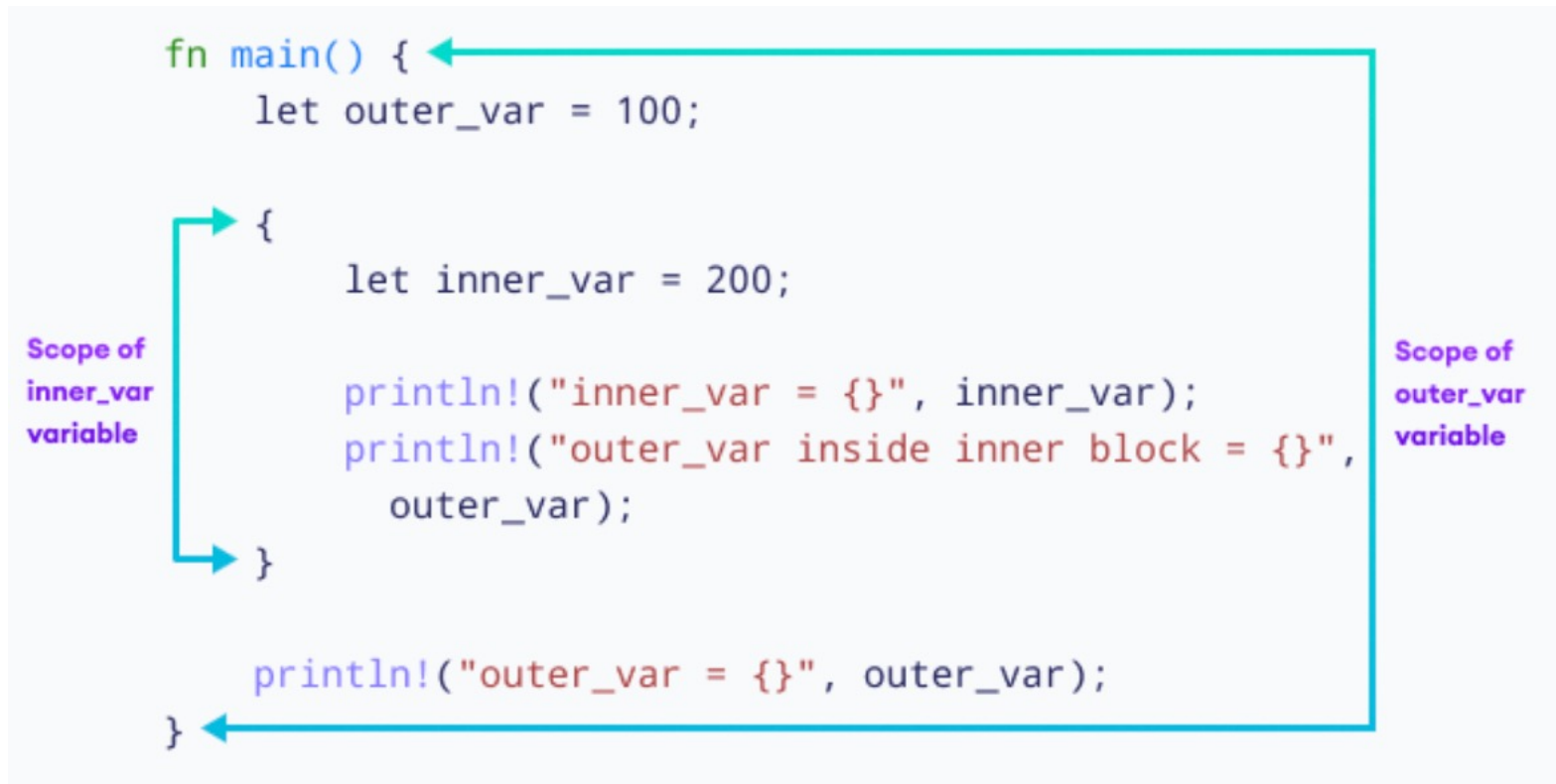
## 이름없는 변수란

배열 등은 배열의 이름을 가지지 않으면 실제 저장하는 곳의 별도의 이름이 없지만 값을 저장할 수 있는 구조이다.



## ■ 변수 스코프

변수는 특정 영역에서만 사용이 가능하면 그 범위를 벗어나면 사용할 수 없다. 이를 변수 스프코로 관리한다.

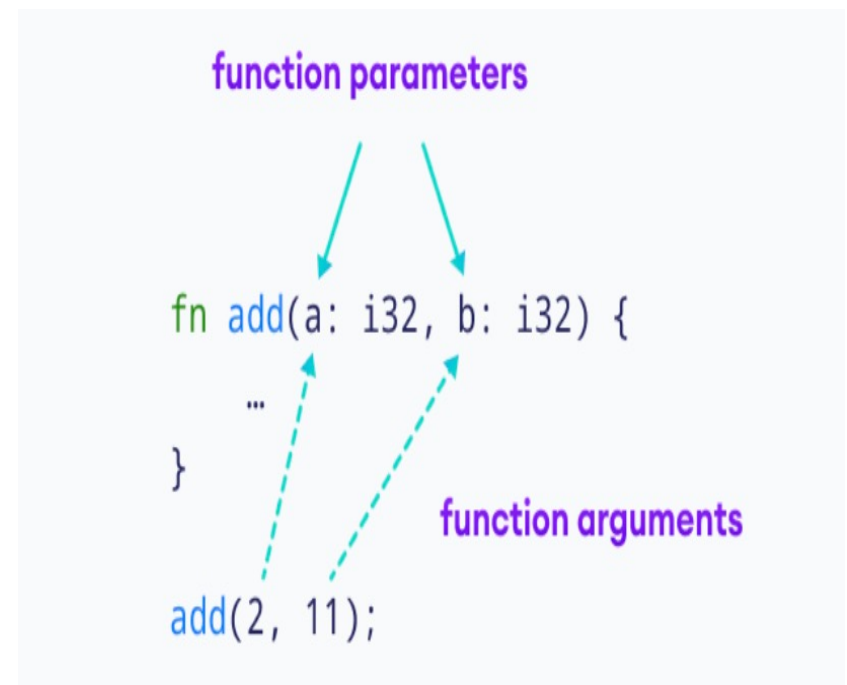



## 함수 인자 전달 방식

변수는 특정 영역에서만 사용이 가능하면 그 범위를 벗어나면 사용할 수 없다. 이를 변수 스킵코로 관리한다.

함수에 인자를 전달하는 방법

- call by value는 함수에 인자의 값을 복사하여 전달하는 방법입니다. 함수 내부에서 인자의 값을 변경해도 원본 값은 변경되지 않습니다.
- call by reference는 함수에 인자의 참조를 전달하는 방법입니다. 함수 내부에서 인자의 값을 변경하면 원본 값도 변경됩니다.





## 4. 소유권 알아보기

---

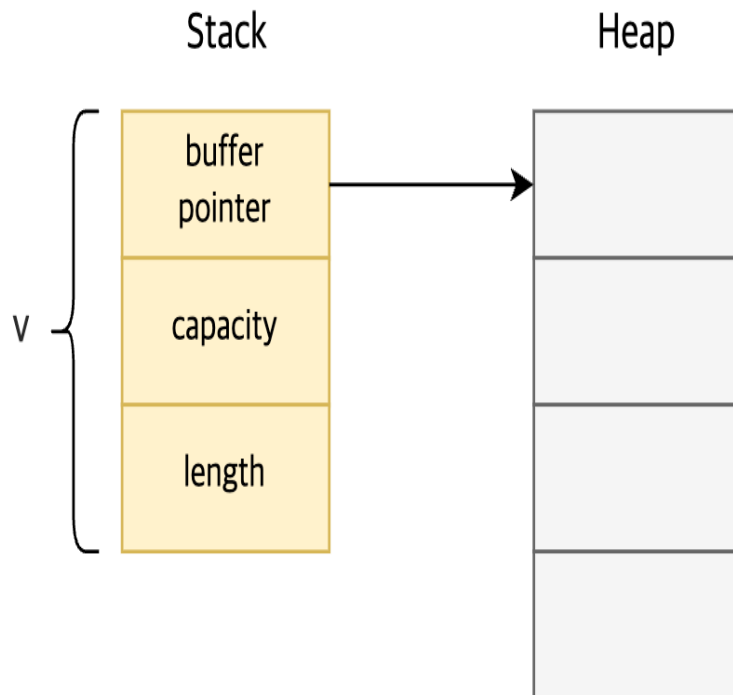
## ■ 소유권의 규칙

소유권은 항상 아래의 규칙에 따라 발생하고 소멸된다.

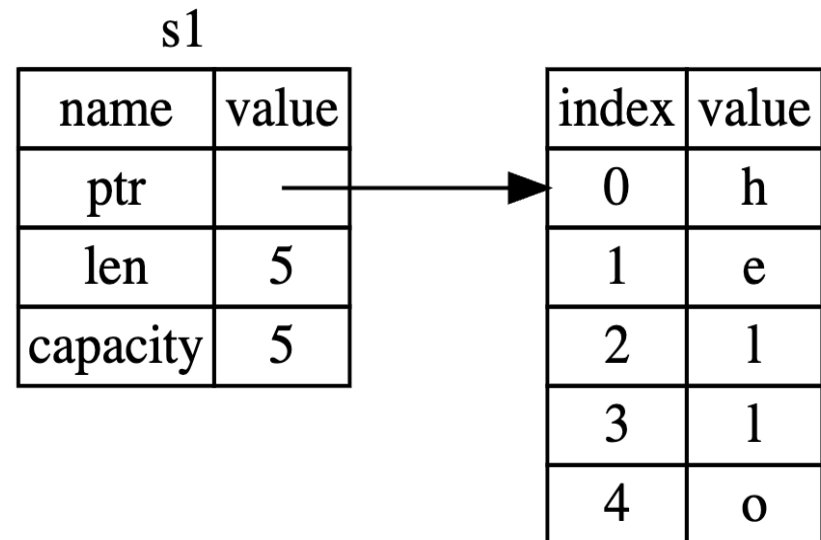
1. 리스트의 각각의 값은 해당값의 *오너(owner)*라고 불리는 변수를 갖고 있다.
2. 한번에 딱 하나의 오너만 존재할 수 있다.
3. 오너가 스코프 밖으로 벗어나는 때, 값은 버려진다(dropped).

## ■ 변수 선언과 초기화

변수를 선언하고 값을 할당하면, 해당 변수는 그 값의 소유권을 획득합니다. 소유권을 가진 변수는 해당 값을 사용하고 변경할 수 있습니다.



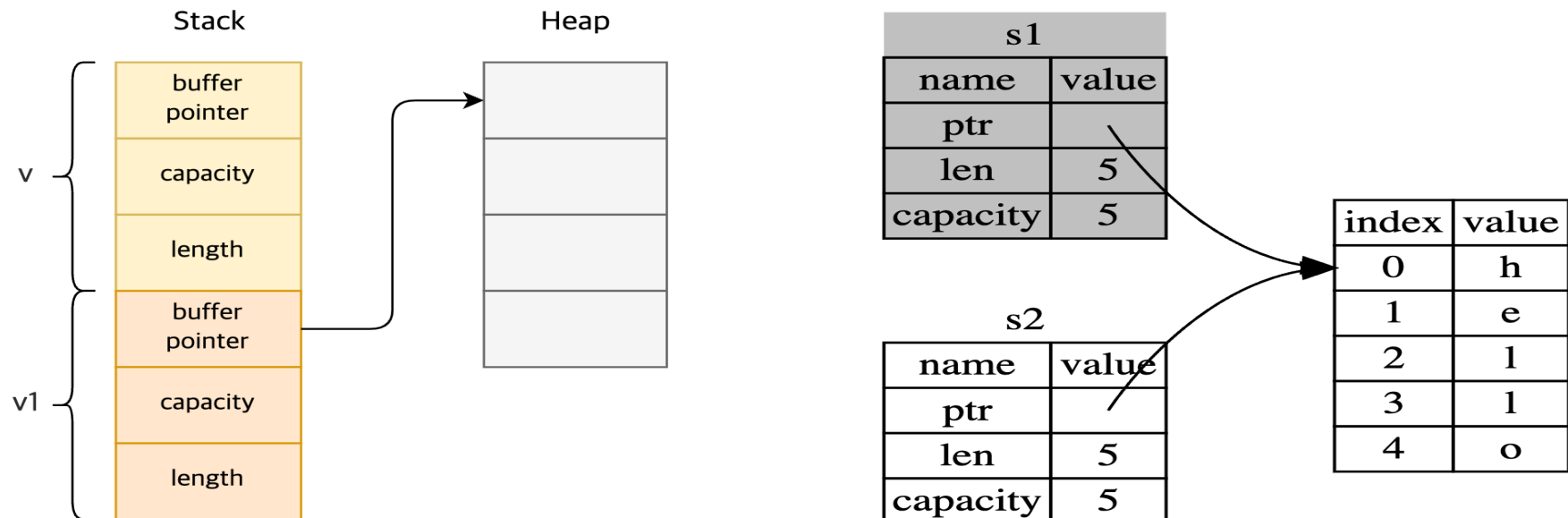
```
let s = String::from("hello");
```



## ■ 소유권 이전(Ownership Transfer) : 변수 재할당

소유권을 가진 변수의 값을 다른 변수로 이동시킬 수 있습니다. 이렇게 하면 이전 변수는 해당 값을 사용할 수 없게 됩니다.

```
let s1 = String::from("hello");  
let s2 = s1;
```

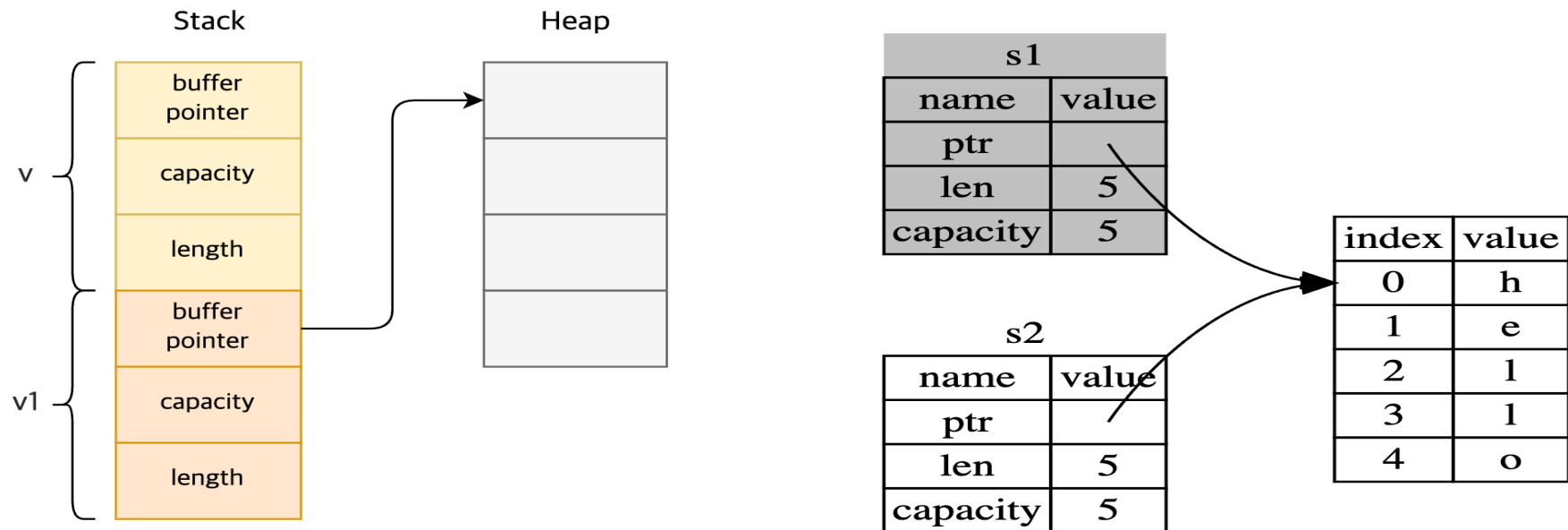




## ■ 소유권 이전(Ownership Transfer) : 함수 호출

함수에 인자로 값을 전달하거나, 함수에서 값을 반환할 때도 소유권 이전이 발생합니다. 인자로 전달된 값은 함수 내에서 소유권을 가지며, 반환된 값은 호출한 쪽으로 소유권이 이전됩니다.

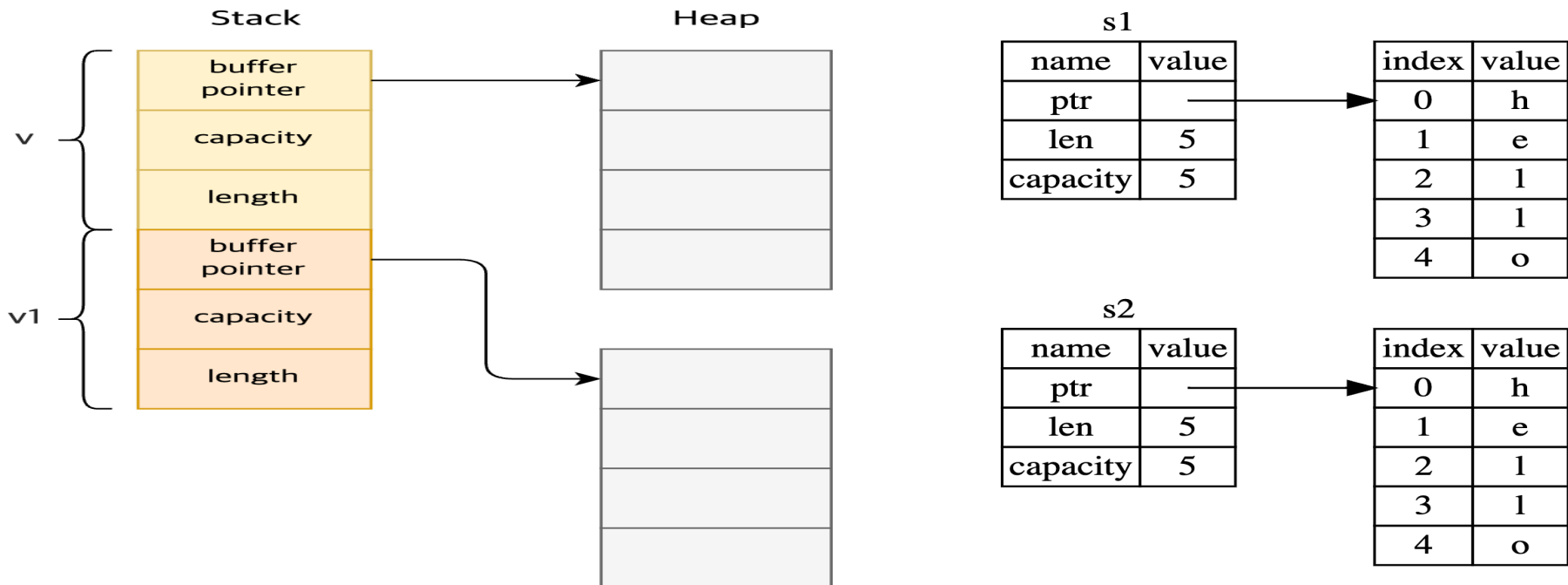
```
let s = String::from("hello");  
  
takes_ownership(s);
```



## 소유권 복제

처리하는 값에 대한 소유권을 유지하려면 두 개의 값이 소유자를 2개 만들어야 한다. 그래서 복제한다.

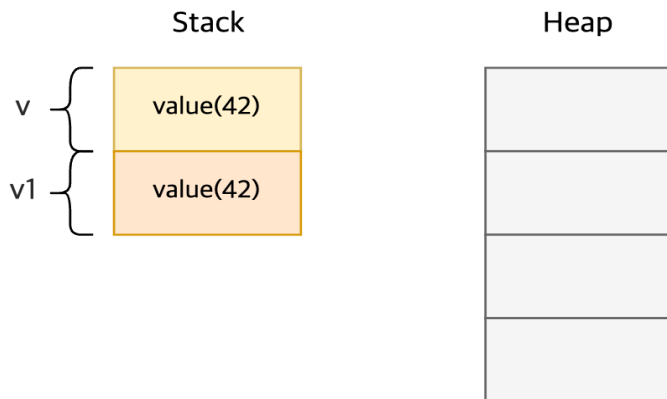
```
let s1 = String::from("hello");  
let s2 = s1.clone();
```



## ■ 원시 값에 대한 소유권 복사

원시 자료형의 값은 이동을 하지 않고 소유권을 다시 만든다. 즉 값을 복사해서 새로운 값을 만든다.

```
let x = 5;  
let y = x;
```



- u32와 같은 모든 정수형 타입들
- true와 false 값을 갖는 부울린 타입 bool
- f64와 같은 모든 부동 소수점 타입들
- Copy가 가능한 타입만으로 구성된 튜플들.  
(i32, i32)는 Copy가 되지만, (i32, String) 안됨

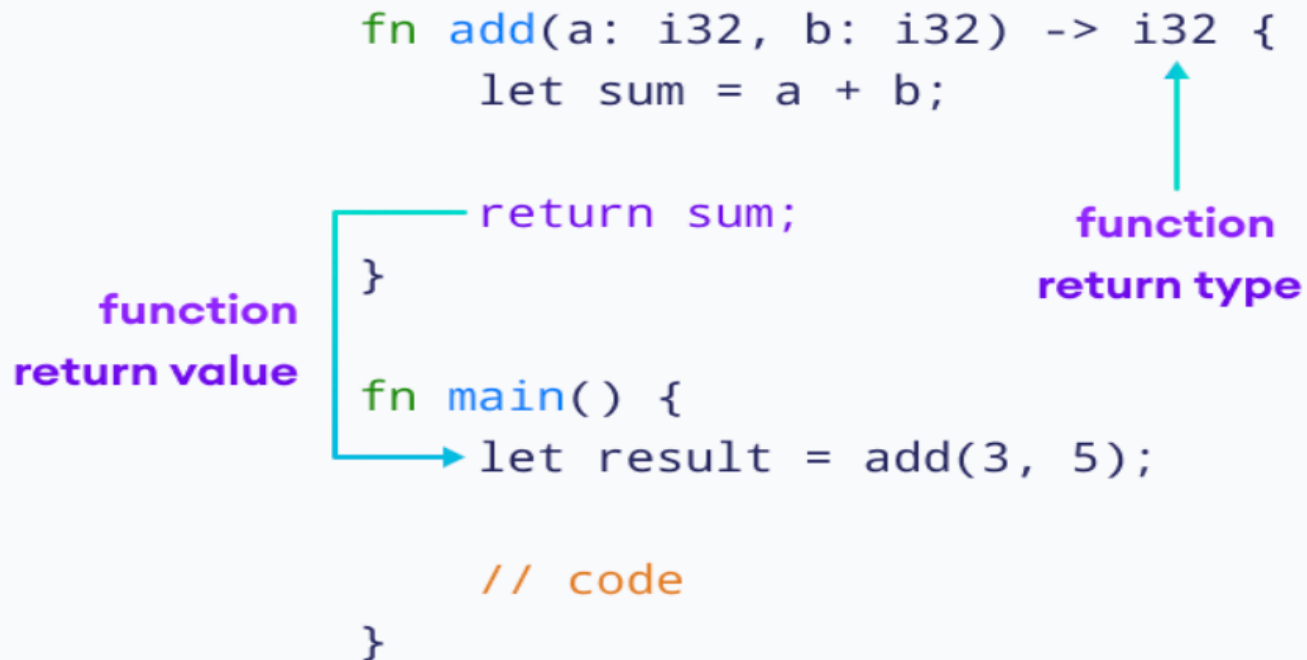
## ■ 원시 값에 대한 소유권 복사 : 함수

원시 자료형의 값은 이동을 하지 않고 소유권을 다시 만든다. 즉 값을 복사해서 새로운 값을 만든다.

```
fn add(a: i32, b: i32) -> i32 {  
    let sum = a + b;  
    return sum;  
}  
fn main() {  
    let result = add(3, 5);  
    // code  
}
```

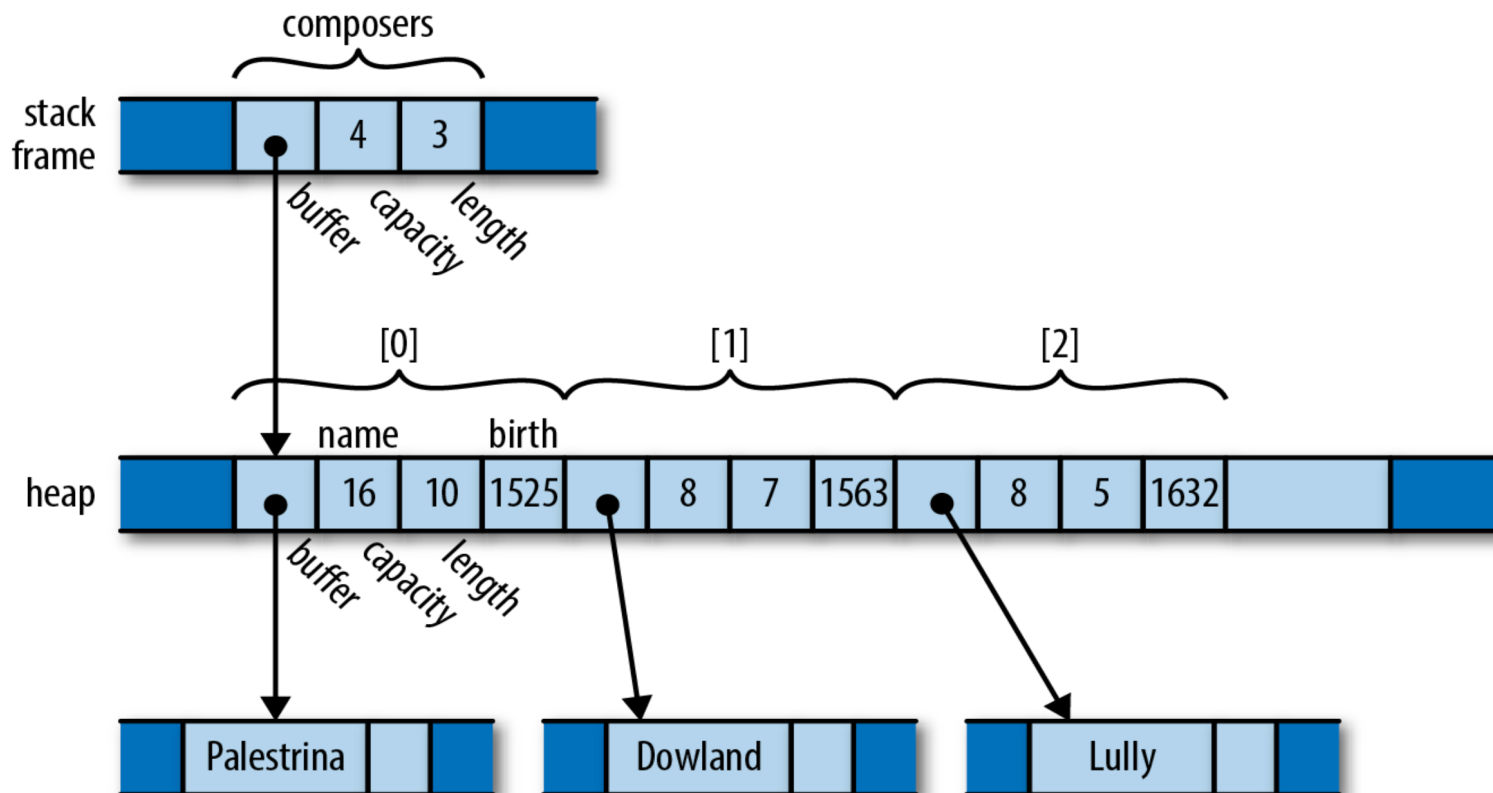
function return value

function return type



## 소유권 트리

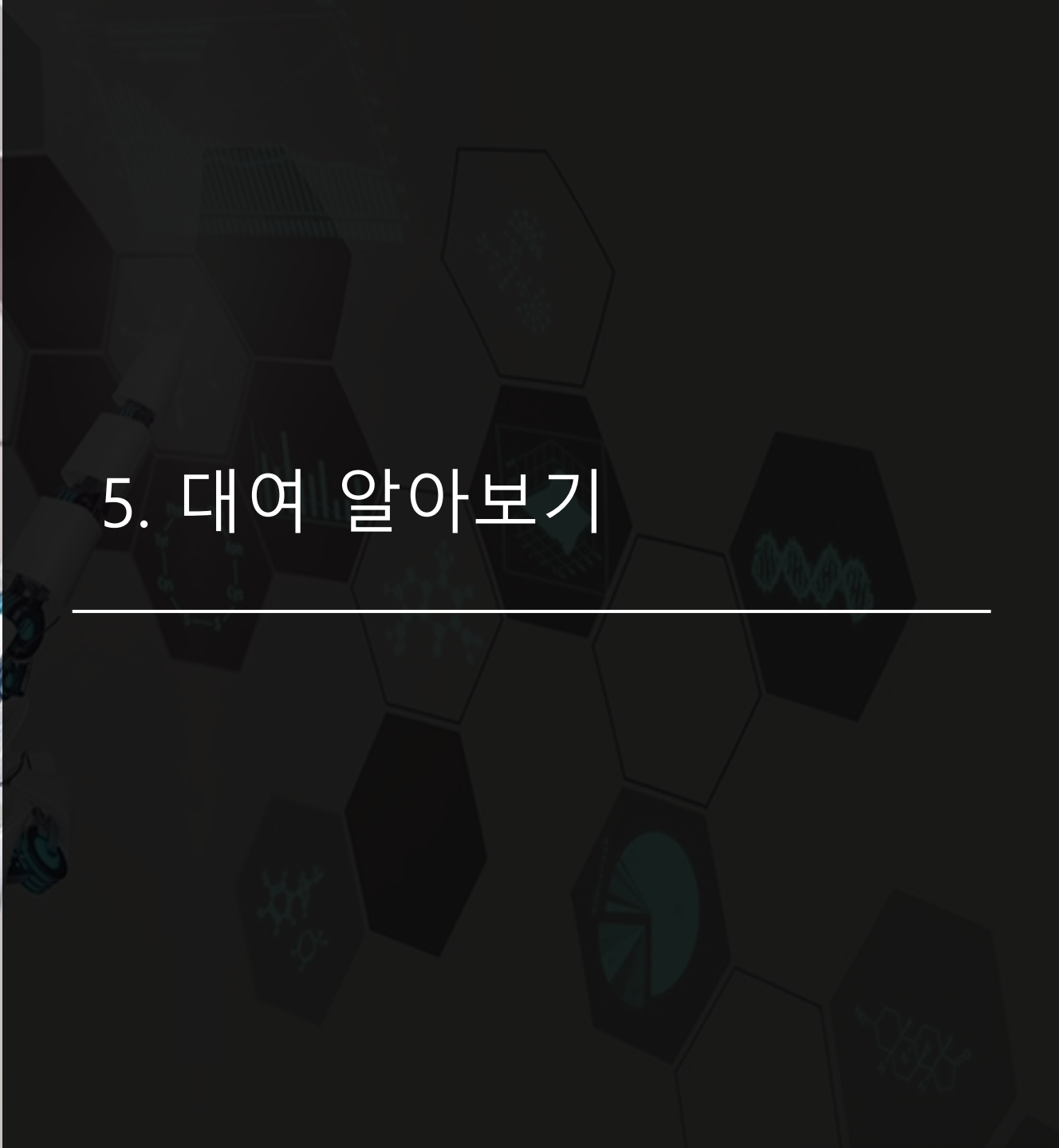
복잡한 자료구조를 가지면 소유권에 대한 트리가 루트는 스택에 나머지는 힙에 구성한다.





## 5. 대여 알아보기

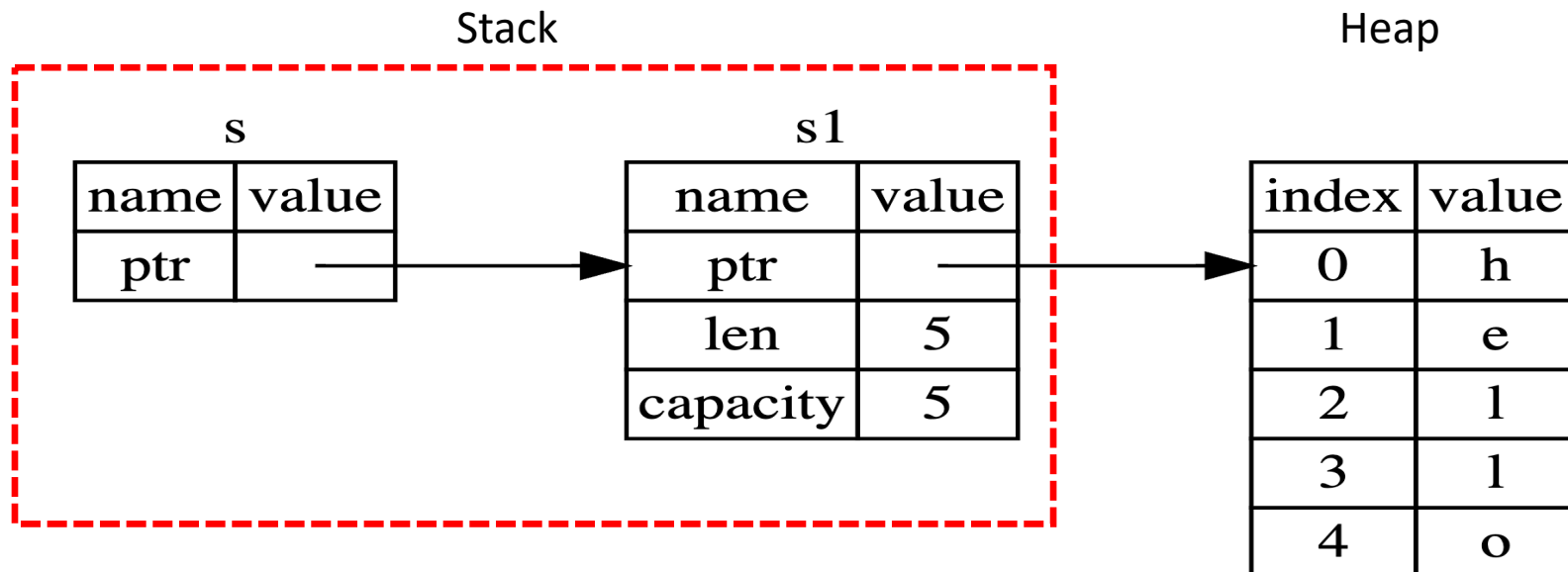
---



## 대여/빌림(Borrow)

빌림은 참조를 사용한다. 소유권이 이동되는 것이 아니라 사용할 수 있는 방법만 얻는 것이다.

```
let s1 = String::from("hello");  
let len = calculate_length(&s1);
```





## ■ 대여/빌림(Borrow) 규칙

### 참조자의 규칙

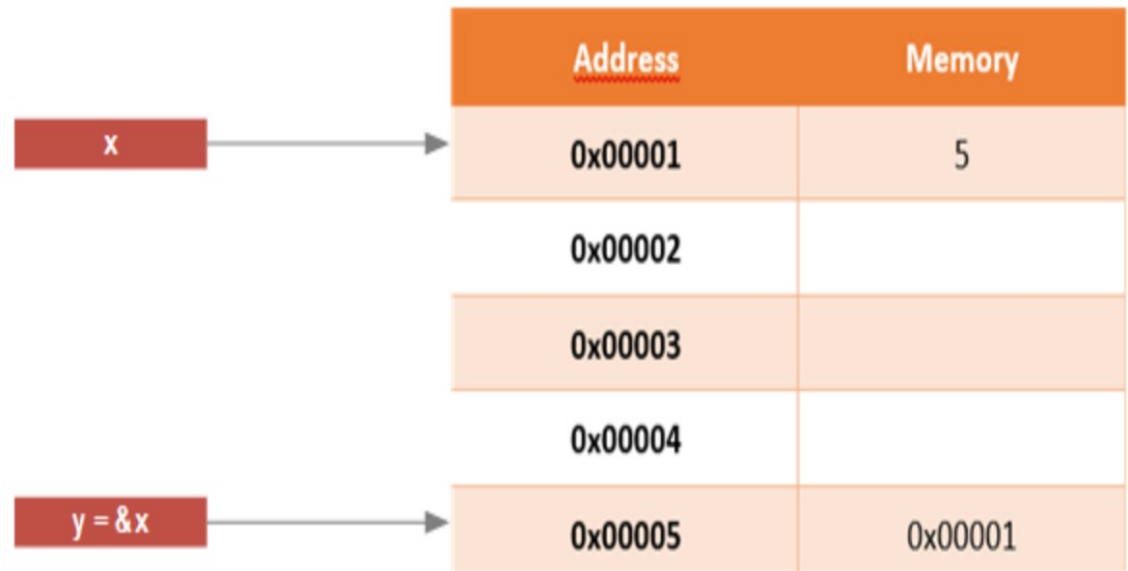
우리가 참조자에 대해 논의한 것들을 정리해 봅시다:

1. 어떠한 경우이든 간에, 여러분은 아래 둘 다는 아니고 *둘 중 하나만* 가질 수 있습니다:
  - 하나의 가변 참조자
  - 임의 개수의 불변 참조자들
2. 참조자는 항상 유효해야만 한다.

## 참조 값의 메모리 저장 방식

실제 변수에는 메모리에 값이 저장된다. 참조를 변수에 저장하면 메모리에 주소가 저장된다.

```
fn main() {  
    let x = 5;  
    let y = &x;  
  
    assert_eq!(5, x);  
    assert_eq!(5, *y);  
}
```



## 스마트 포인터와 참조의 차이점

스마트 포인터는 소유권을 가지는 포인터이다. 그래서 참조와 달리 새롭게 값을 소유하므로 원시 자료형의 값이라도 힙에 저장한다.

```
fn main() {  
    let x = 5;  
    let y = Box::new(x);  
  
    assert_eq!(5, x);  
    assert_eq!(5, *y);  
}
```

### Stack

	ADDR	VALUE
x =	0x0001	5
y =	0x0002	0xFF01
	0x0003	
	0x0004	
	0x0005	

### Heap

	ADDR	VALUE
	0xFF01	5
	0xFF02	
	0xFF03	
	0xFF04	
	0xFF05	



Q & A

---

