2023 MS AI School
강사: 이영록

객체지향 프로그래밍

객체 지향 프로그래밍

- 잘 설계된 클래스를 이용하여 객체(오브젝트)를 만들고
- 클래스는 속성(인스턴스 변수)과 행위(메소드)를 가지도록 설계하고
  - 객체는 클래스에서 정의한 속성(state)과 행위(behavior)를 수행한다
  - 소프트웨어상에서 객체의 상태 또는 속성은 인스턴스 변수로 표현
  - 행위 혹은 동작은 메소드로 표현함

객체지향의 용어

- 객체 : 데이터, 행위, 아이덴티티를 가지고 있는 것.
- 클래스 : 클래스란 객체를 생성하는 틀이다.
- 캡슐화 : 행위와 상태를 포장하고 외부에 노출할 것과 감출 것을 결정하는 것
- 상속 : 코드의 재사용성과 계층구조의 표현
- 다형성 : 캡슐화, 상속과 함께 동작 함으로써 객체-지향 프로그램의 흐름 제어를 단순화 하는 것 입니다.

인스턴스(instance)

- 클래스로부터 만들어지는 각각의 객체를 그 클래스의 인스턴스라고 한다.
- 서로 다른 인스턴스는 서로 다른 속성 값을 가진다.

클래스의 정의

```
class Person:
    def hello(self):
        print('Hello')

person = Person()
person.hello()
person1 = Person()
person1.hello()
```

속성 정의

```
class Person:
    def __init__(self):
        self.hi = 'Hello'

    def hello(self):
        print(self.hi)

person = Person()
person.hello()
print(person.hi)
```

속성 정의와 초기화

```
class Person:
    def __init__(self, n, a):
        self.name = n
        self.age = a

    def hello(self):
        print('Hello {}’.format(self.name))
        print(‘당신은 {}살입니다.'.format(self.age))

person = Person('홍길동', 20)
person.hello()
```

클래스 정의 실습

- 학생 클래스를 정의하라.
- 학년, 반, 이름의 세 가지 속성을 가진다.
- 속성은 생성자를 통해서 설정된다.
- '몇학년 몇반 누구입니다.' 라고 출력하는 introduce 라는 메소드를 정의하라.

비공개 속성 정의

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.__age = age

    def hello(self):
        print('Hello {}’.format(self.name))
        print('당신은 {}살입니다.'.format(self.__age))

person = Person('홍길동', 20)
person.hello()
print(person.__age) # 에러
person.__age = 100  # 에러
```

비공개 속성 사용

```
class Person:
    def __init__(self, name, age):
        self.name = name
        if 0 <= age <= 20: self.__age = age
        else: age = 0

    def inc_age(self):
        self.__age += 1

    def info(self):
        print(self.__age)

person = Person('홍길동', 20)
person.inc_age()
person.info()
```

클래스 속성 사용

```
class Person:
    count = 0

    def __init__(self, name):
        self.name = name
        Person.count += 1

person1 = Person('홍길동')
print(person1.count)
person2 = Person('허균')
print(person2.count)
```

클래스(class) 메서드 정의

- 클래스메서드는 클래스에서 바로 호출할 수 있는 메서드
- 클래스명.메서드명()으로 호출
- 메서드 위에 @classmethod를 붙임
- 매개변수에 self를 사용하지 않음

```
class 클래스이름:
    @classmethod
    def 메서드명(cls, 매개변수1, 매개변수2):
        코드
```

정적(static) 메서드 정의

- 정적메서드는 클래스에서 바로 호출할 수 있는 메서드
- 클래스명.메서드명()으로 호출
- 메서드 위에 @staticmethod를 붙임
- 매개변수에 self를 사용하지 않음

```
class 클래스이름:
    @staticmethod
    def 메서드명(매개변수1, 매개변수2):
        코드
```

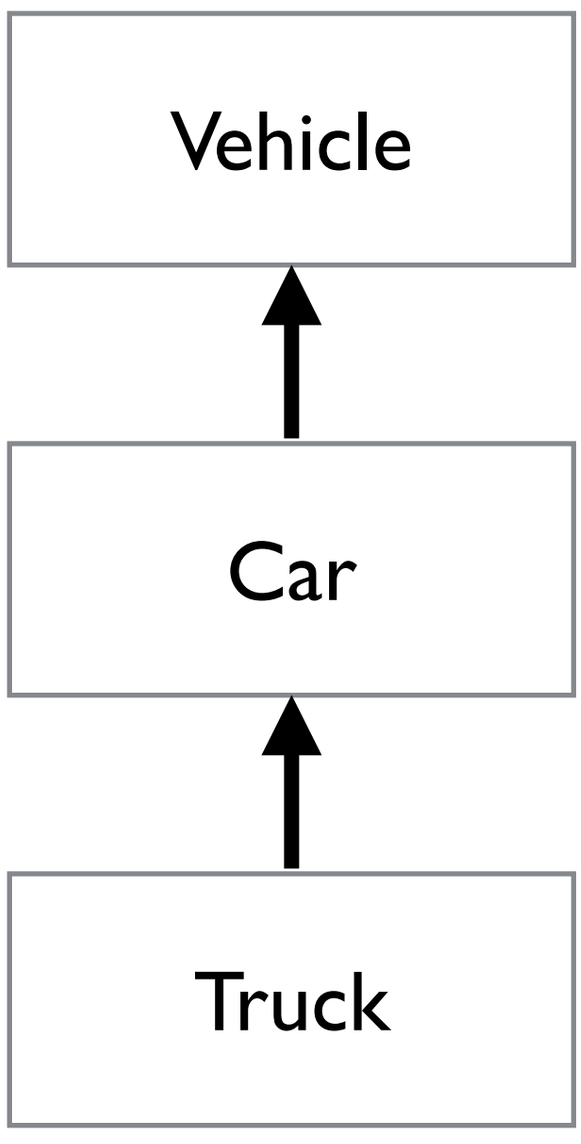
정적(static) 메서드사용

```
class Math:
    @staticmethod
    def add(a, b):
        return a + b

    @staticmethod
    def sub(a, b):
        return a - b

print(Math.add(4,5))
print(Math.sub(9,5))
```

클래스 정의 실습

- 사각형을 관리하는 클래스를 정의하라.
- 가로, 세로를 저장하는 속성을 정의하라.
- 가로, 세로로 면적을 반환하는 메소드를 정의하라
- 가로, 세로를 각각 두배로 만드는 메서드를 정의하라

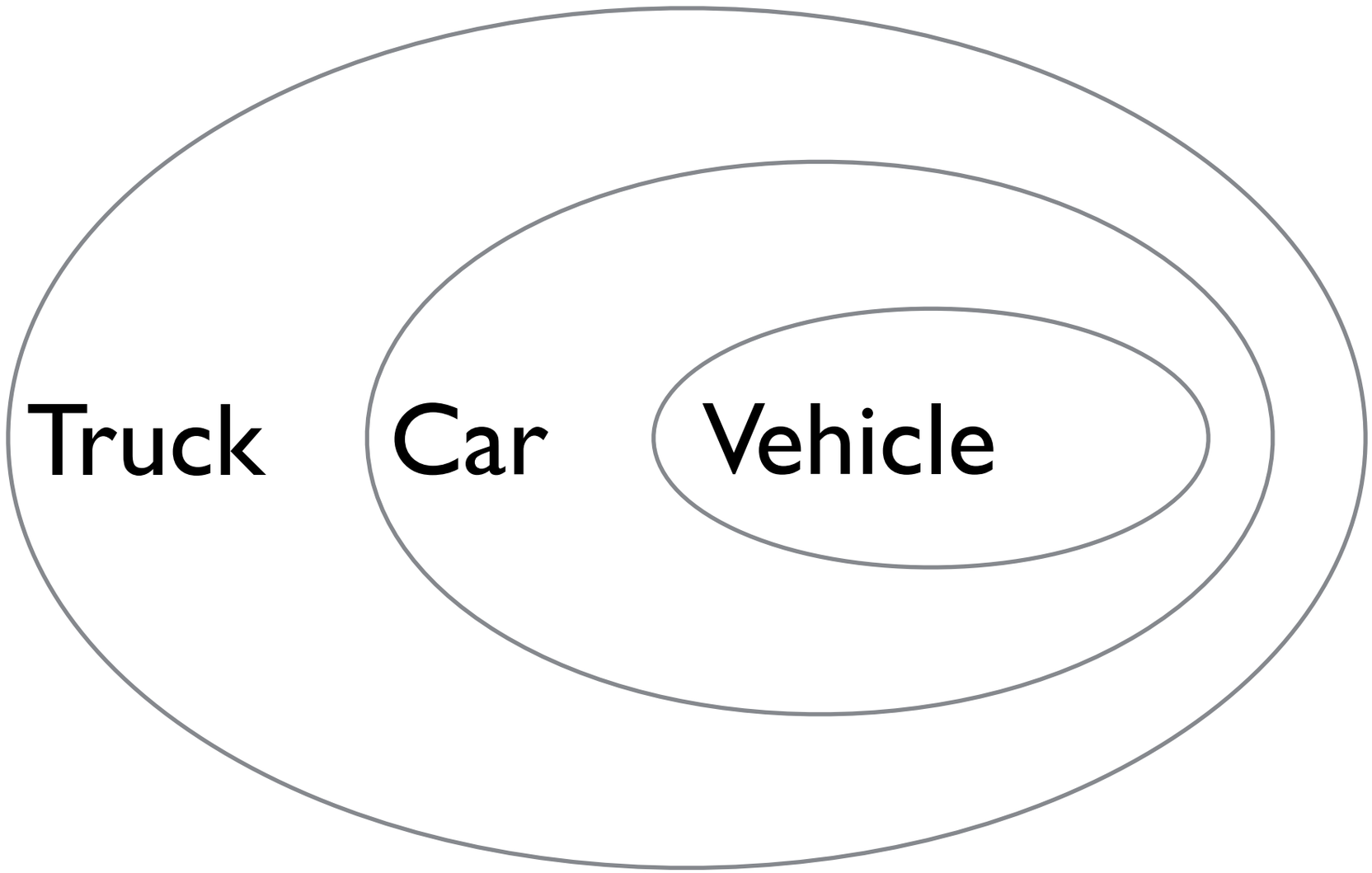
클래스의 상속



Car의 Super class

Car의 Sub class

클래스의 상속



클래스의 상속

```
class Vehicle:
    def __init__(self, speed):
        self.speed = speed

    def speed_up(self):
        self.speed += 10

    def speed_dn(self):
        self.speed -= 10
```

클래스의 상속

```
class Car(Vehicle):
    def __init__(self, speed, wheels, seats):
        Vehicle.__init__(self, speed)
        self.wheels = wheels
        self.seats = seats

    def info(self):
        print(self.speed, self.wheels, self.seats)

car = Car(100, 4, 4)
car.speed_up()
car.info()
```

클래스 상속 실습

- Car class를 상속받아서 Truck class를 정의하라.
- 적재량을 관리하는 loadage 변수를 정의하라.
- ‘load’를 화면에 인쇄하는 load 메쏘드를 정의하라.
- ‘unload’를 화면에 인쇄하는 unload 메쏘드를 정의하라.

메서드 재정의

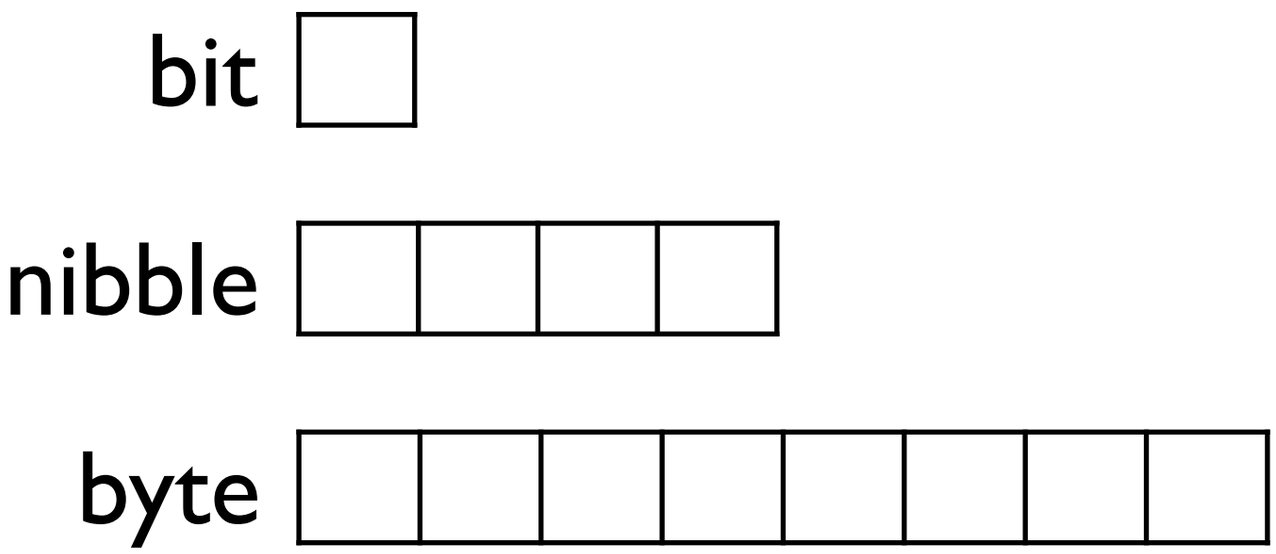
```
class Truck(Car):
    def __init__(self, speed, wheels, seats, loadage):
        Car.__init__(self, speed, wheels, seats)
        self.loadage = loadage

    def load(self):
        print('load')

    def unload(self):
        print('unload')

    def info(self):
        print(self.speed, self.wheels, self.seats, self.loadage)

truck = Truck(100, 6, 2, 30)
truck.load()
truck.unload()
truck.info()
```

객체지향의 동작방식

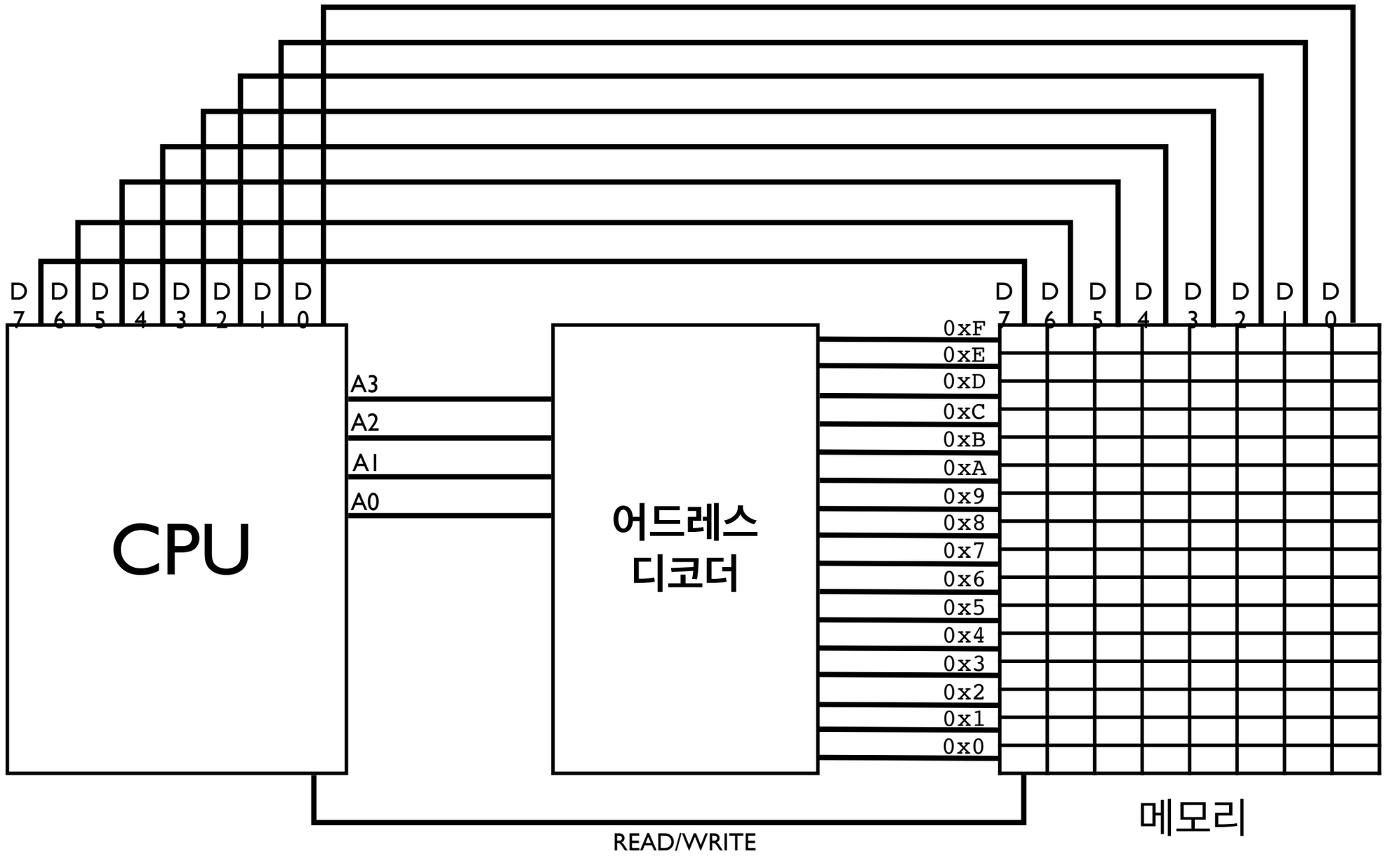
메모리란?

플립플롭(Flip-Flop)

- 출력값을 유지(저장)하는 회로
- 기본적으로 1비트의 정보를 저장하는 회로
- 0V의 경우 Low라고 (논리적으로 FALSE, 이진수로 0으로 표기)
- 5V(3.3V, 1.8V)의 경우 High (논리적으로 TRUE, 이진수로 1로 표기)

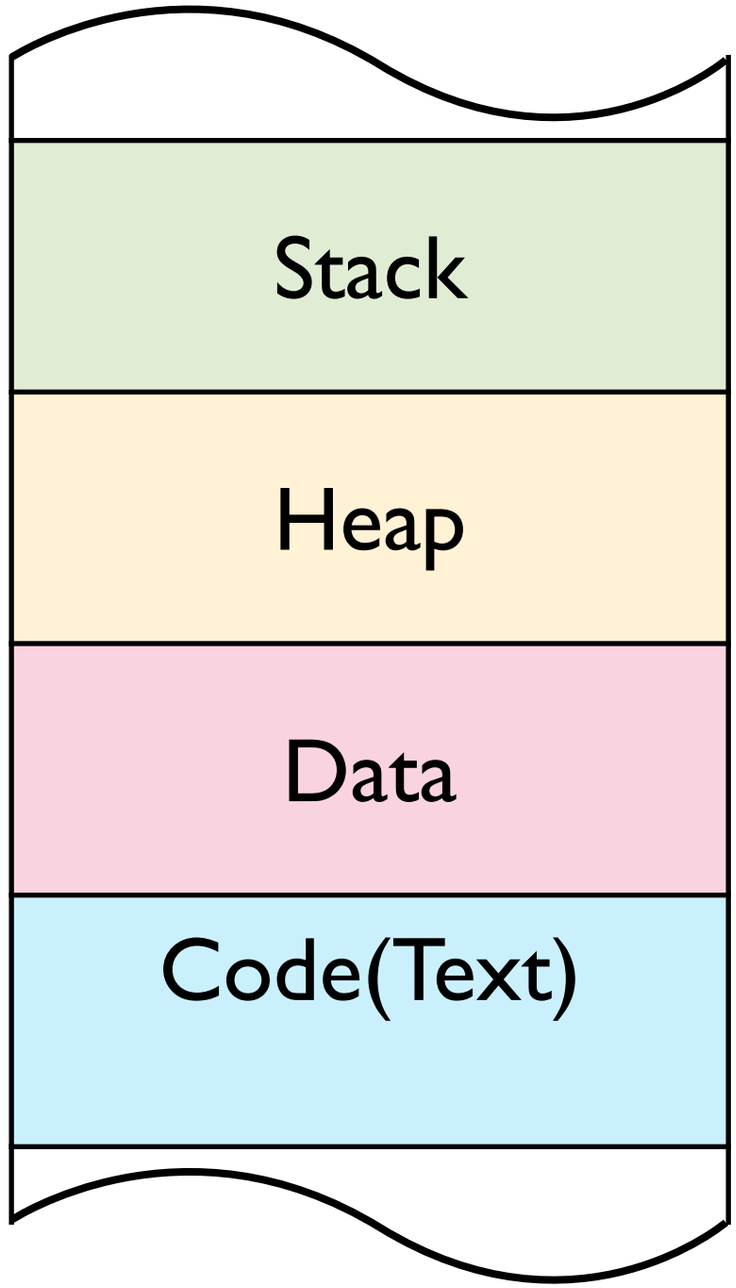


메모리주소란?

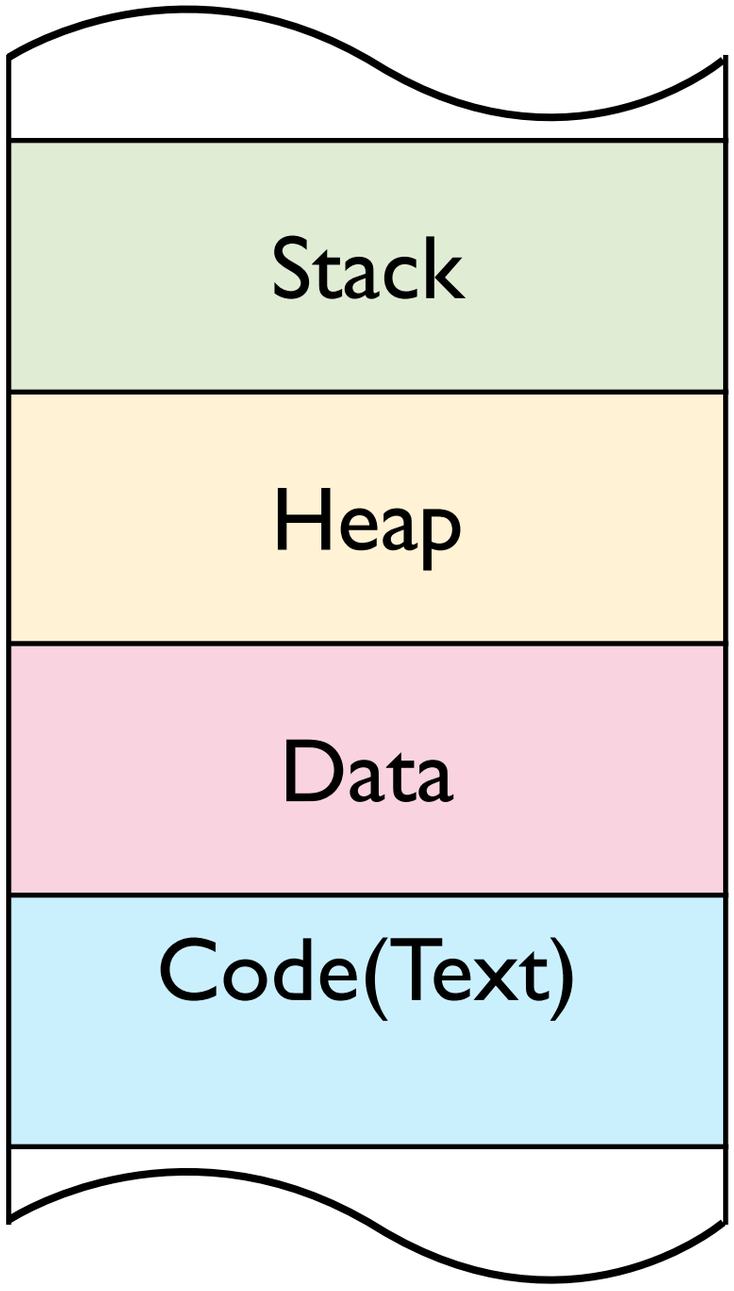

D7
D6
D5
D4
D3
D2
D1
D0
CPU
A3
A2
A1
A0
어드레스
디코더
0xF
0xE
0xD
0xC
0xB
0xA
0x9
0x8
0x7
0x6
0x5
0x4
0x3
0x2
0x1
0x0
메모리
READ/WRITE

OS에 의한 메모리 구분

OS에 의한 메모리 구분

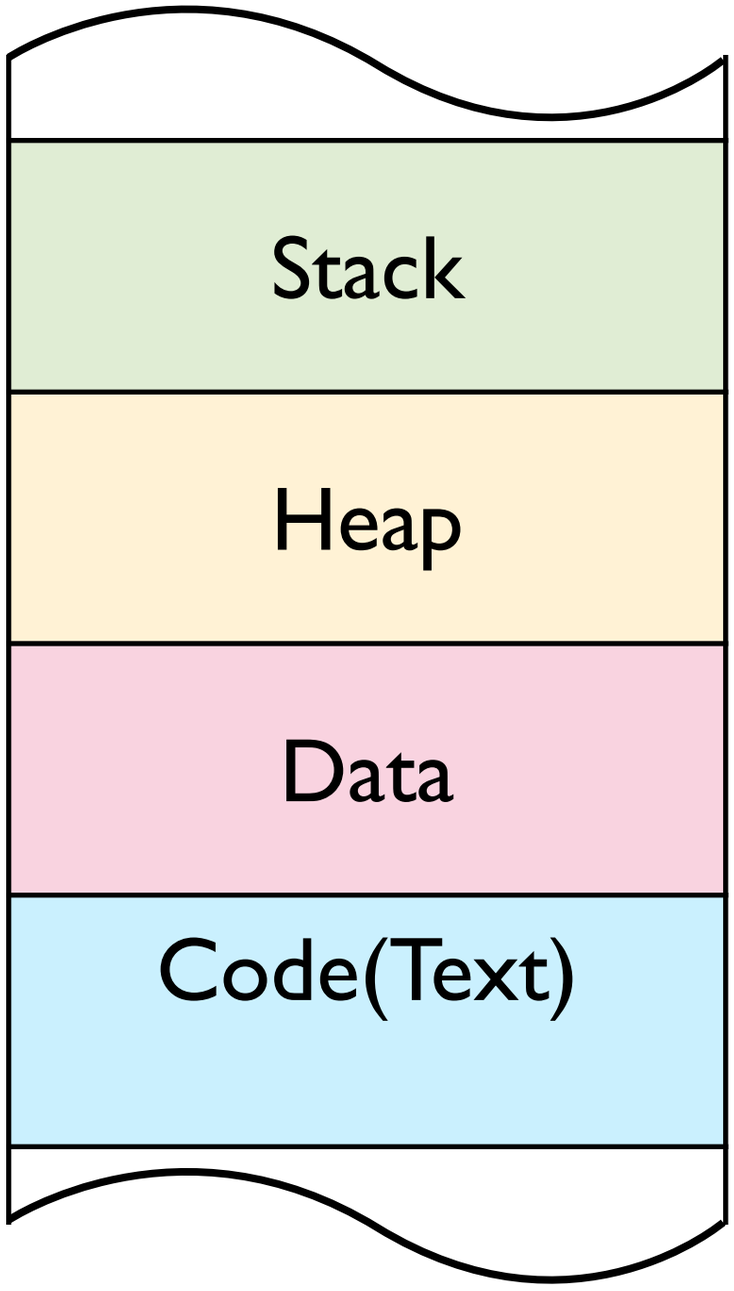


OS에 의한 메모리 구분



프로그램코드가 저장

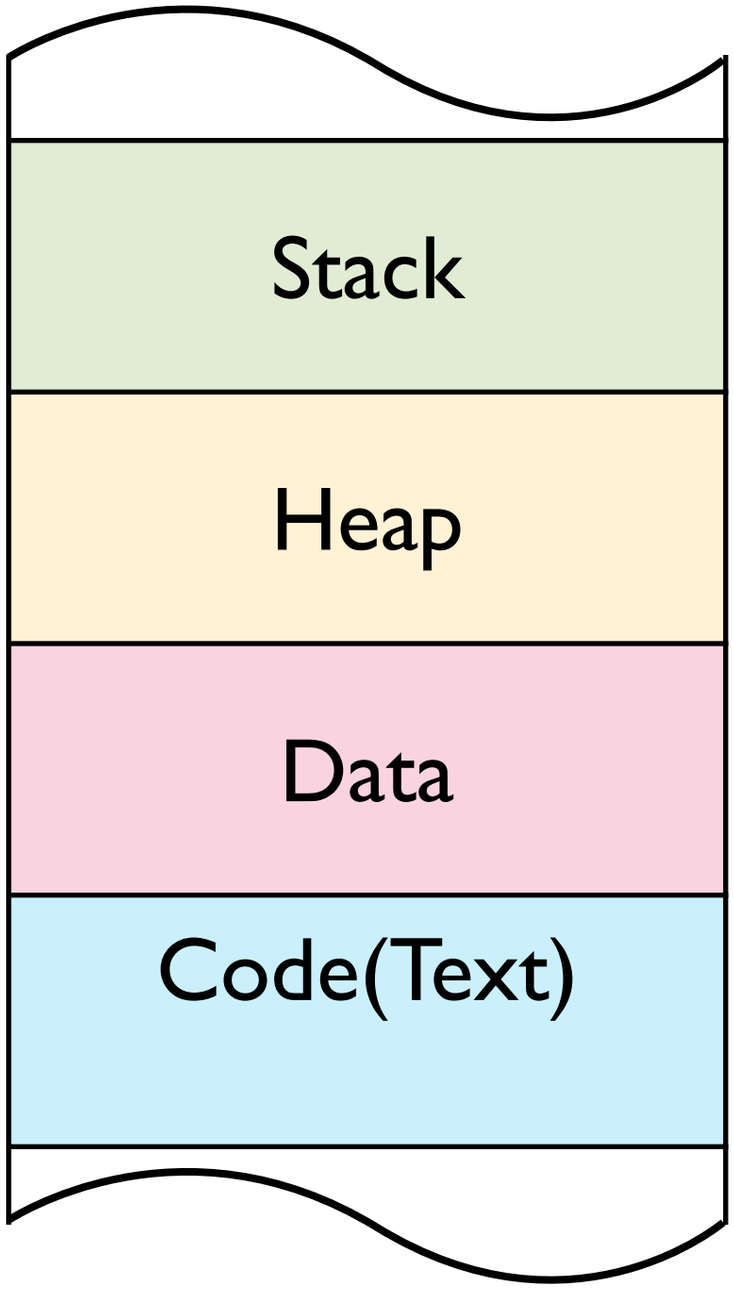
OS에 의한 메모리 구분



전역변수와 정적변수가 저장

프로그램코드가 저장

OS에 의한 메모리 구분

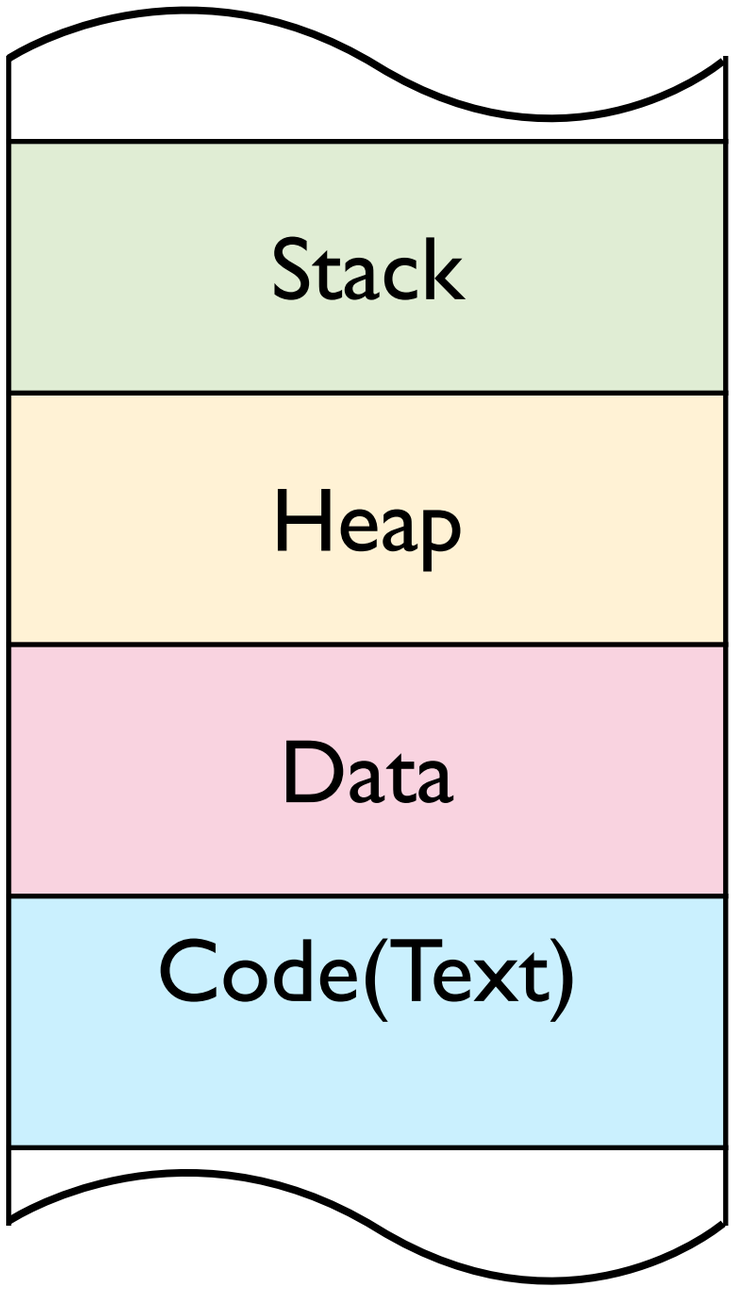


동적할당을 위한 공간

전역변수와 정적변수가 저장

프로그램코드가 저장

OS에 의한 메모리 구분



지역변수와 매개변수가 저장

동적할당을 위한 공간

전역변수와 정적변수가 저장

프로그램코드가 저장

클래스의 선언

```
class Vehicle:

    def __init__(self, wheels, seats, engine):
        self.wheels = wheels
        self.seats = seats
        self.engine = engine

def drive(self):
    print('Run')
```

인스턴스 생성 및 사용

```
bike = Vehicle(2, 1, 2)
print('bike.wheels = ', bike.wheels)
print('bike.seats = ', bike.seats)
bike.wheels = 3
print('bike.wheels = ', bike.wheels)
bike.drive()
```

```
car = Vehicle(4, 4, 4)
print('car.wheels = ', car.wheels)
print('car.seats = ', bike.seats)
car.drive()

copy_bike = bike
copy_bike.wheels = 4
print('bike.wheels = ', bike.wheels)
```

Stack

Heap

Code

```
bike = Vehicle(2, 1, 2)
```

Stack

Heap

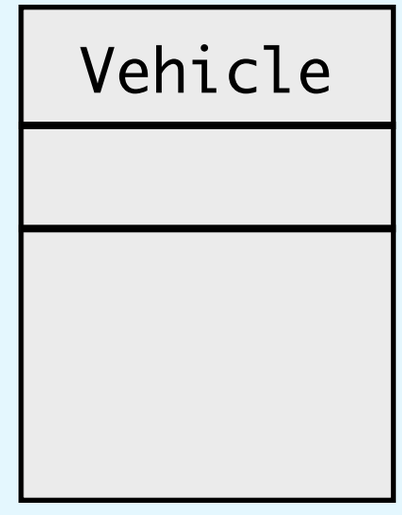
Code

```
bike = Vehicle(2, 1, 2)
```

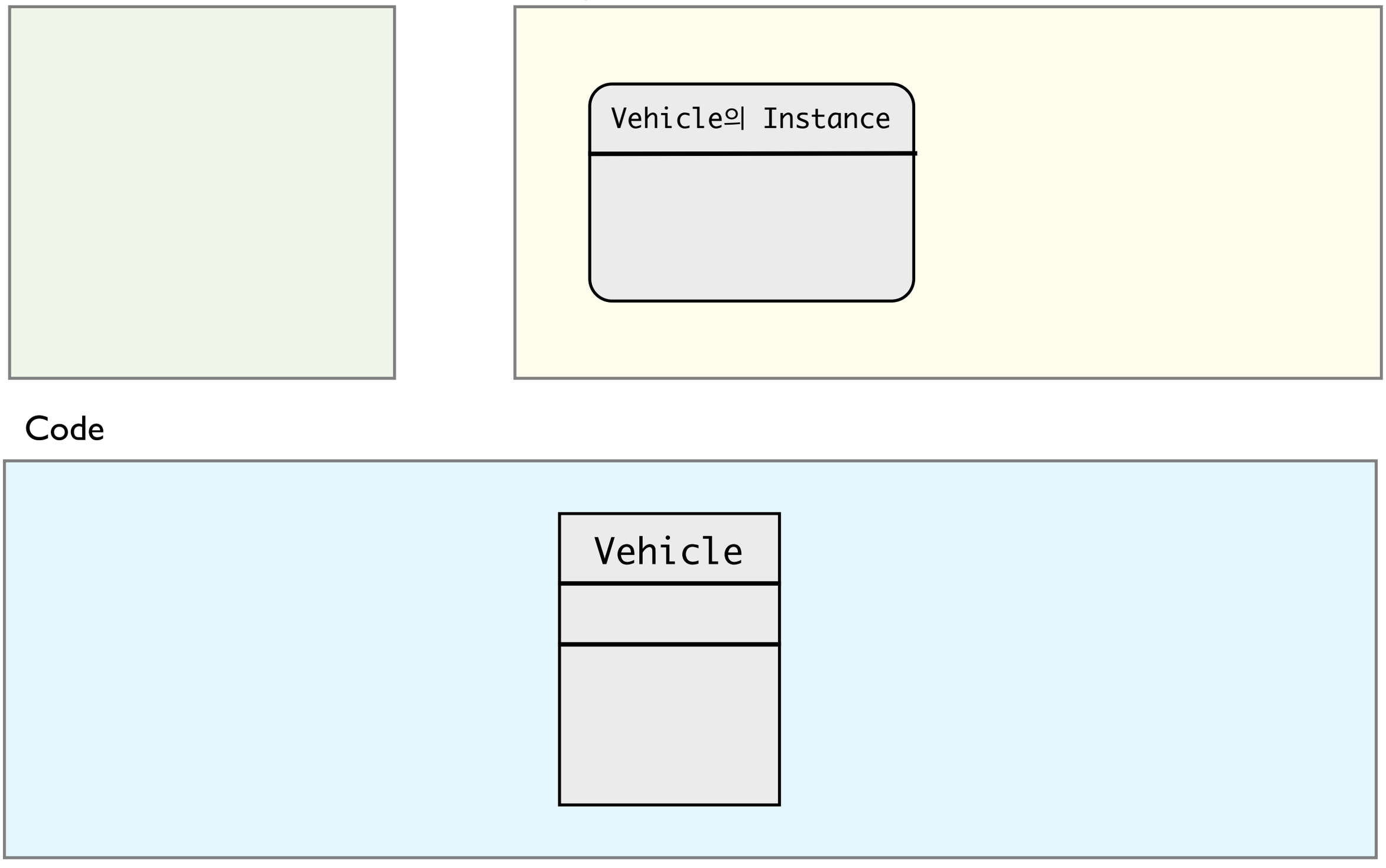
Stack

Heap

Code



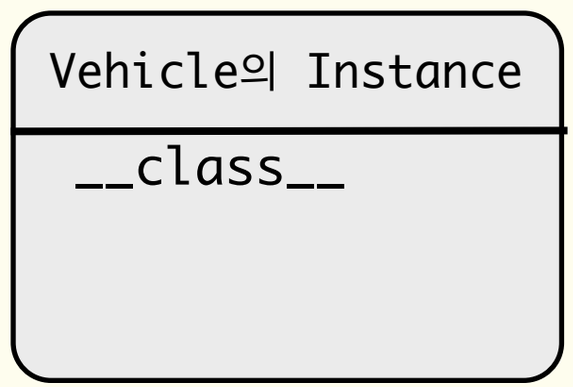
```
bike = Vehicle(2, 1, 2)
```



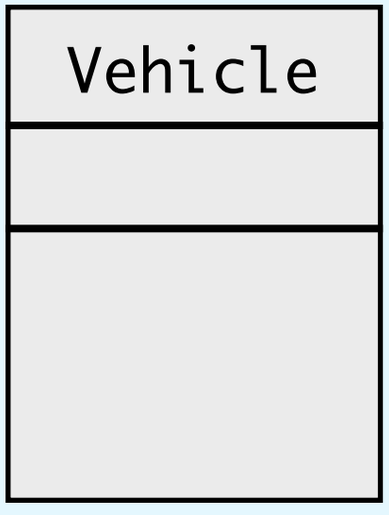
```
bike = Vehicle(2, 1, 2)
```

Stack

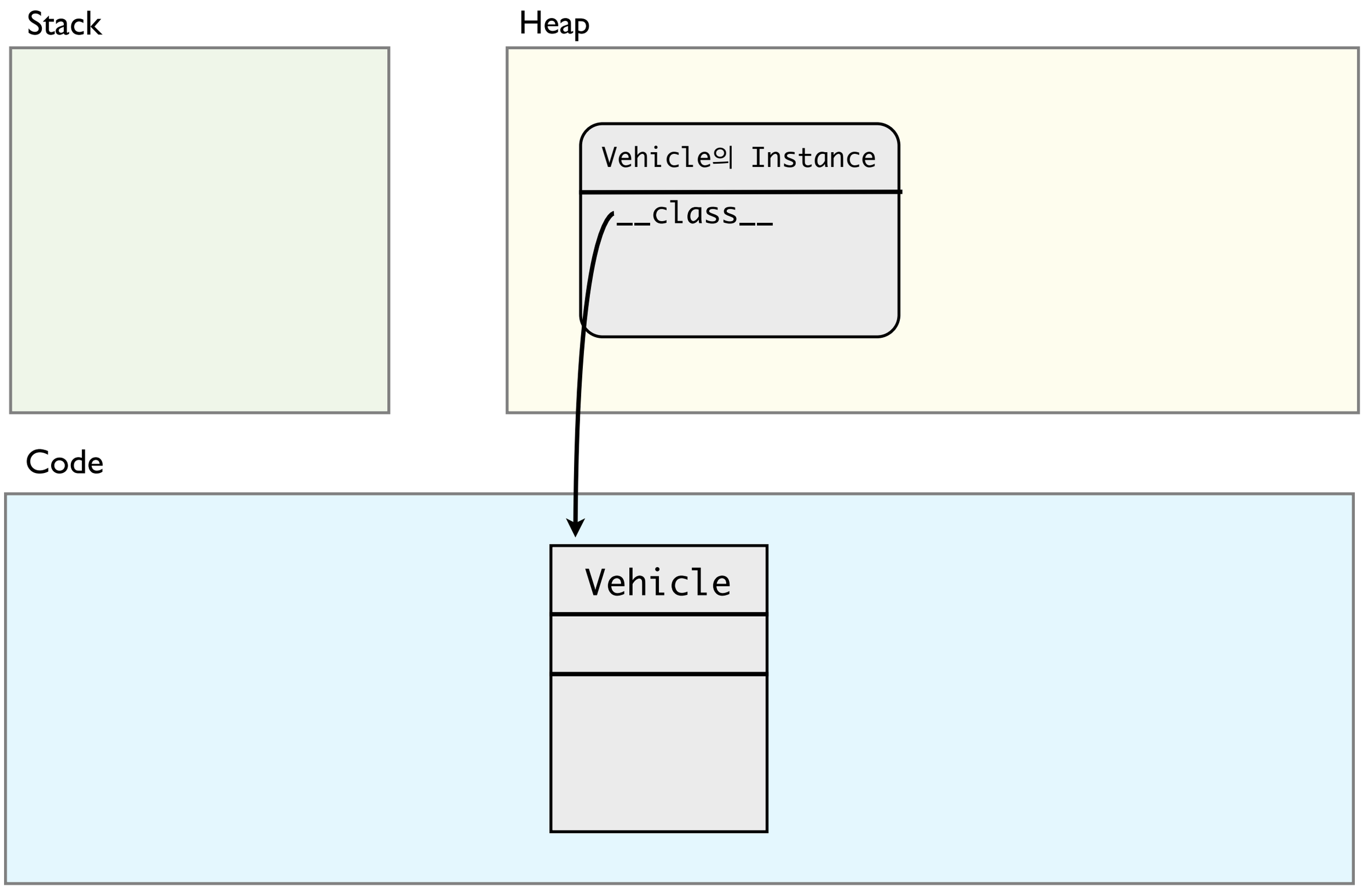
Heap



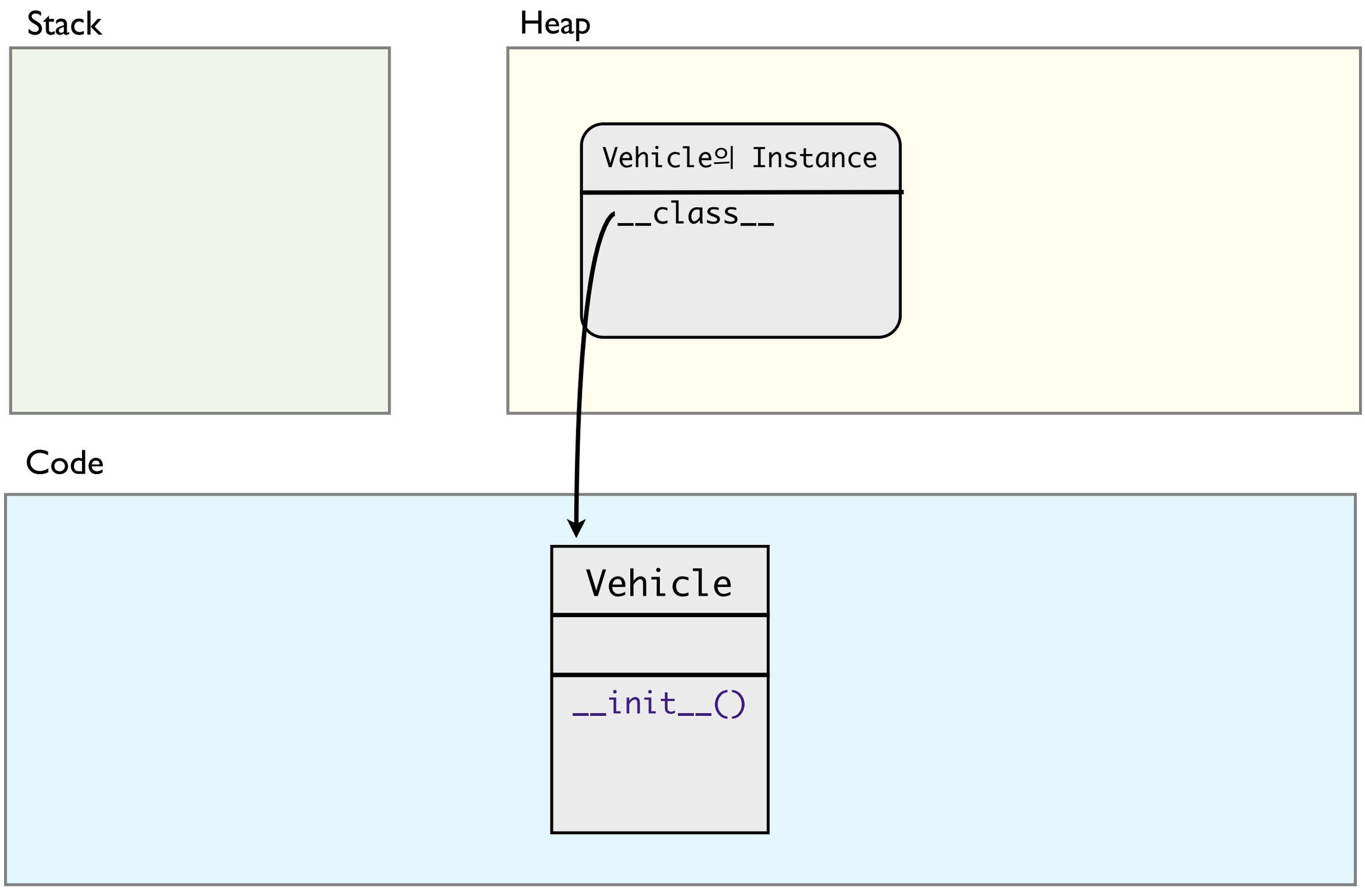
Code



```
bike = Vehicle(2, 1, 2)
```



```
bike = Vehicle(2, 1, 2)
```



```
bike = Vehicle(2, 1, 2)
```

Stack

Heap

Vehicle의 Instance

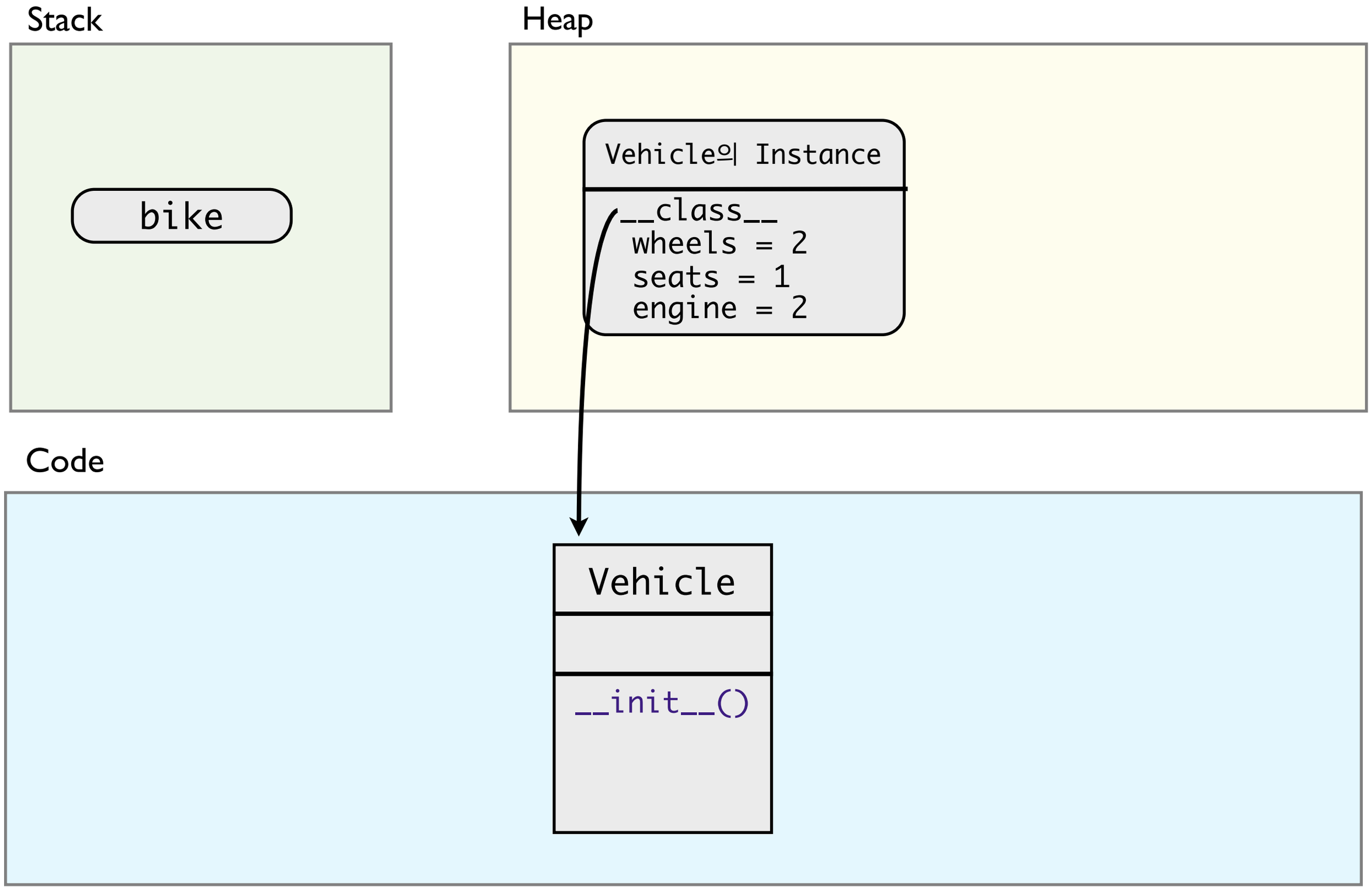
__class__
wheels = 2
seats = 1
engine = 2

Code

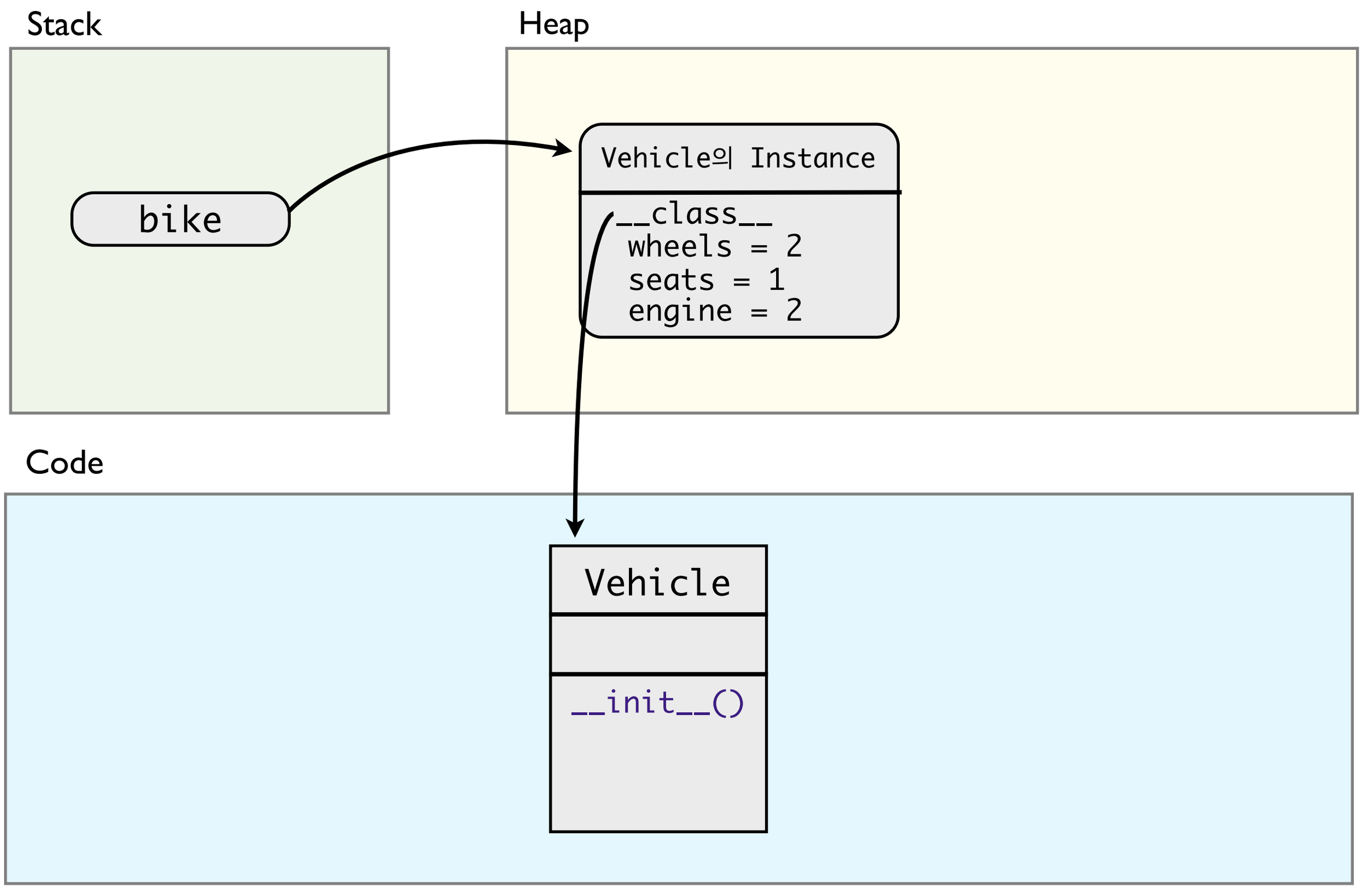
Vehicle

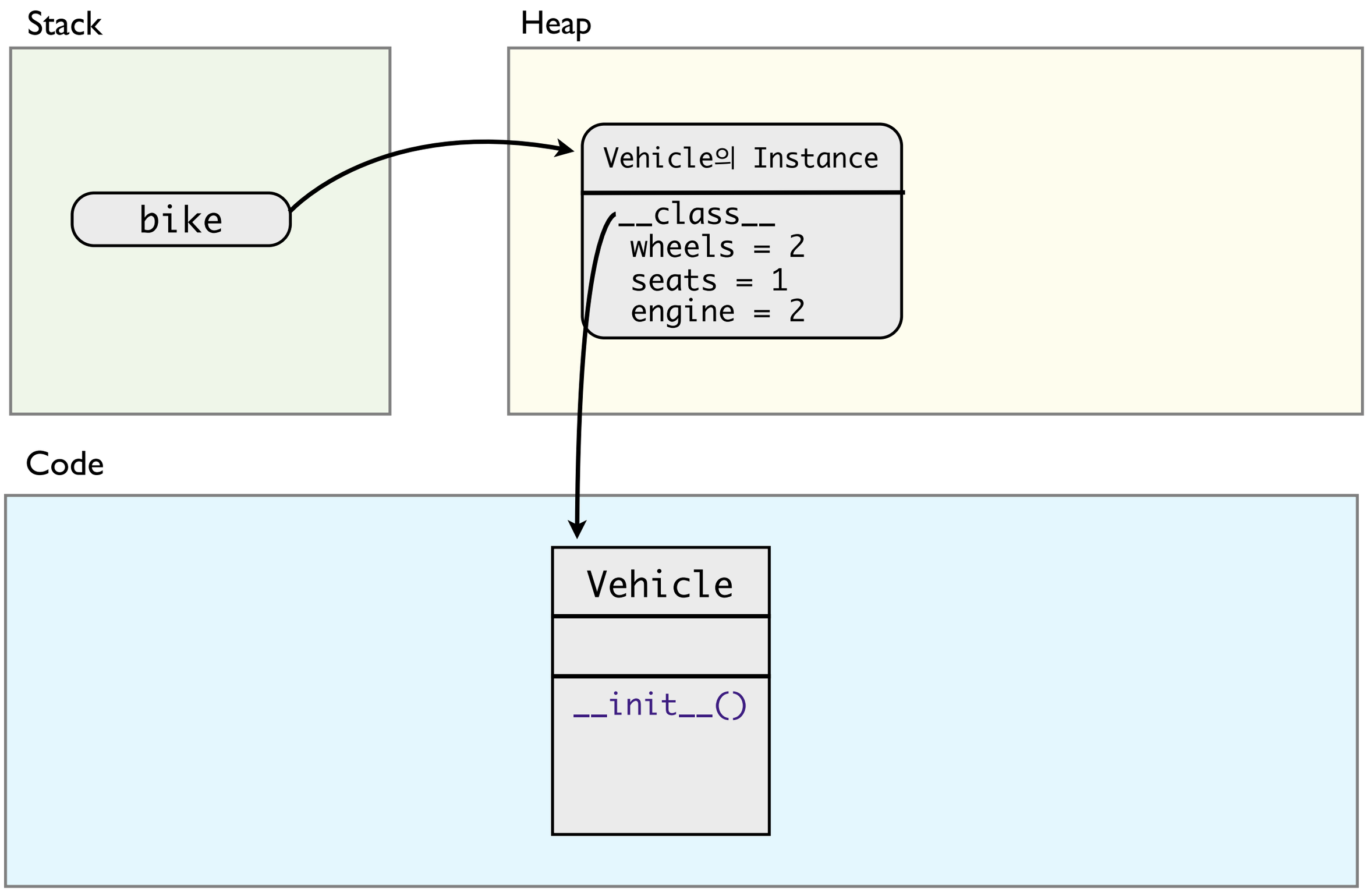
__init__()

```
bike = Vehicle(2, 1, 2)
```

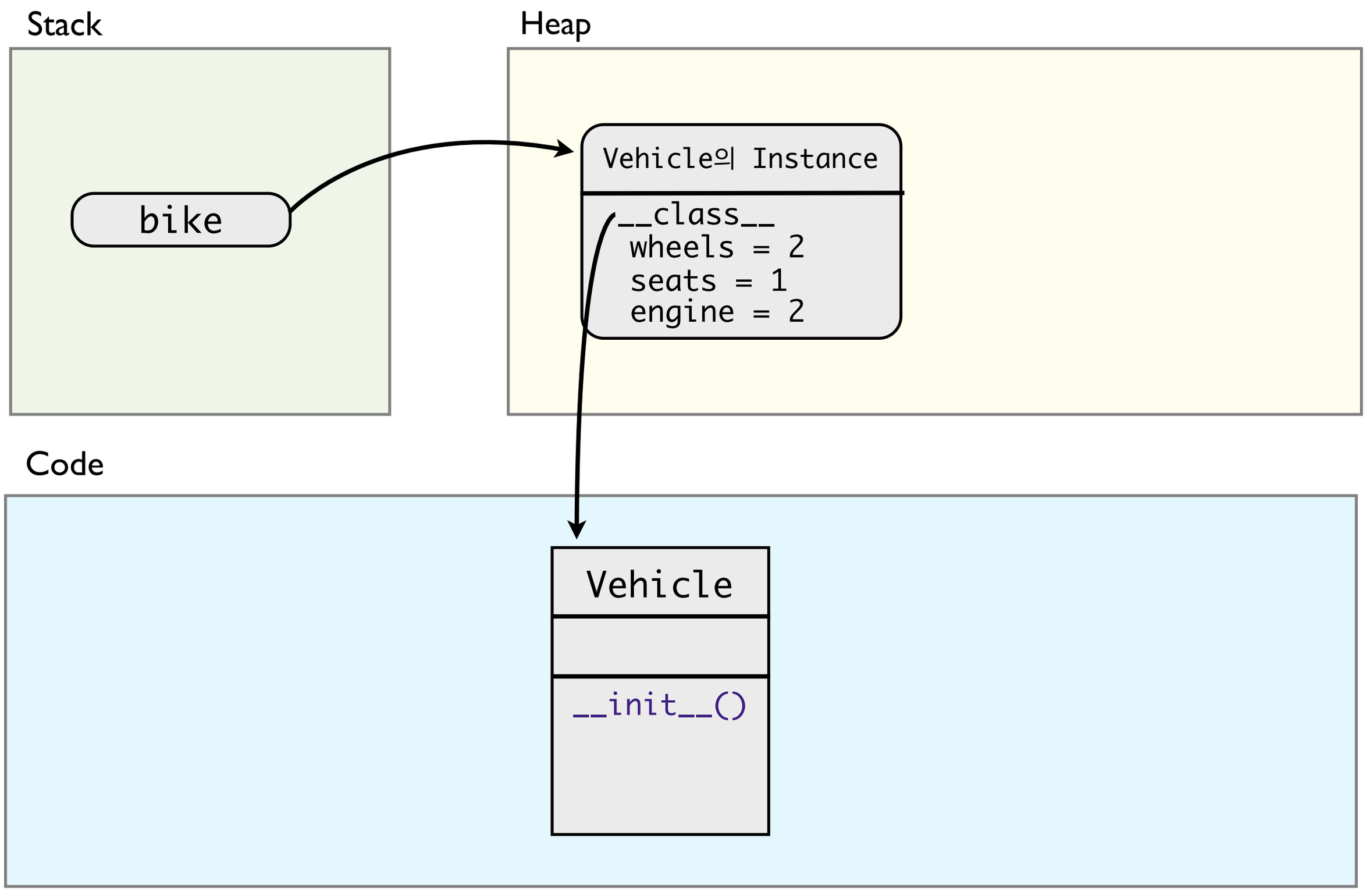


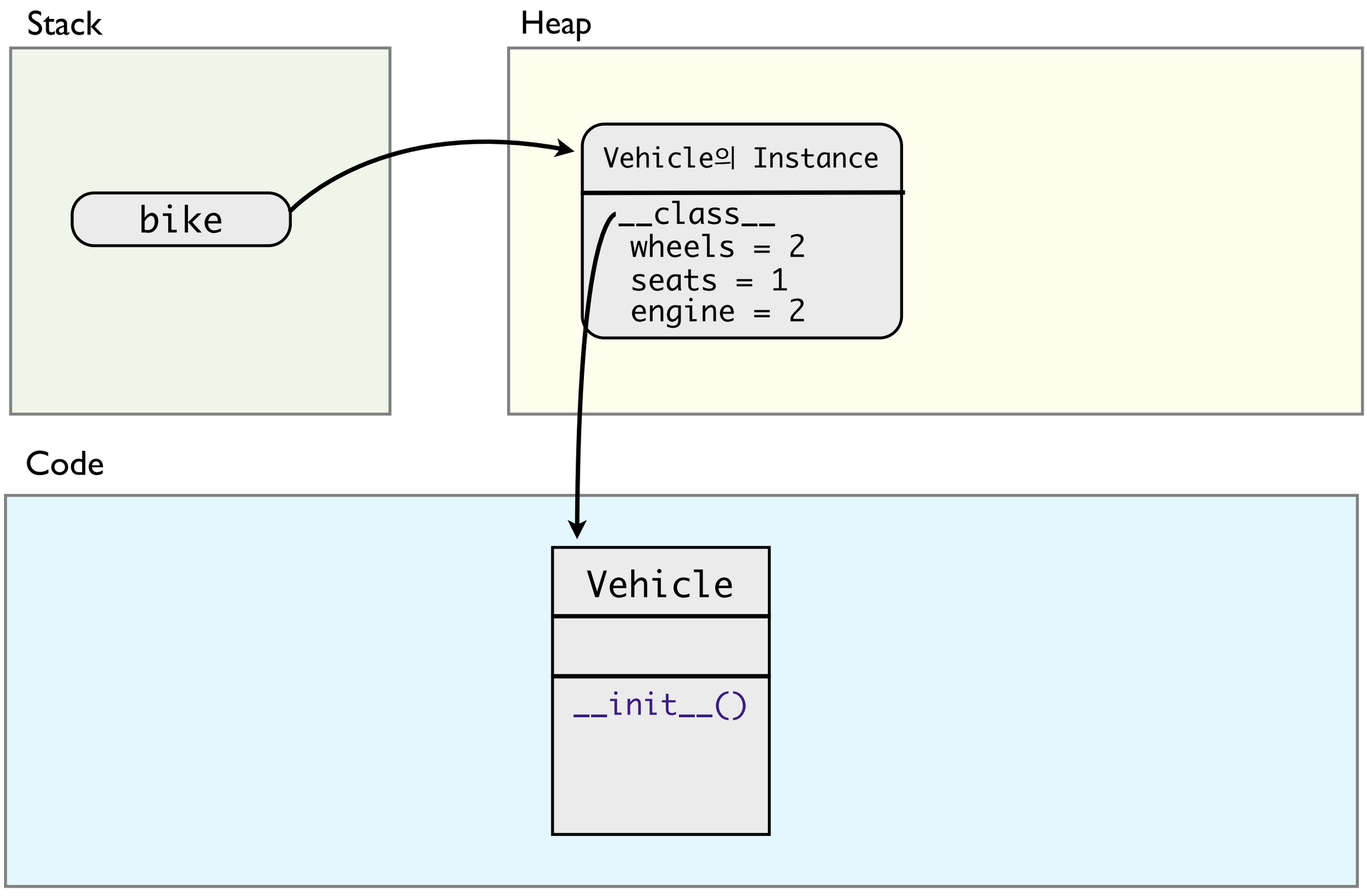
```
bike = Vehicle(2, 1, 2)
```



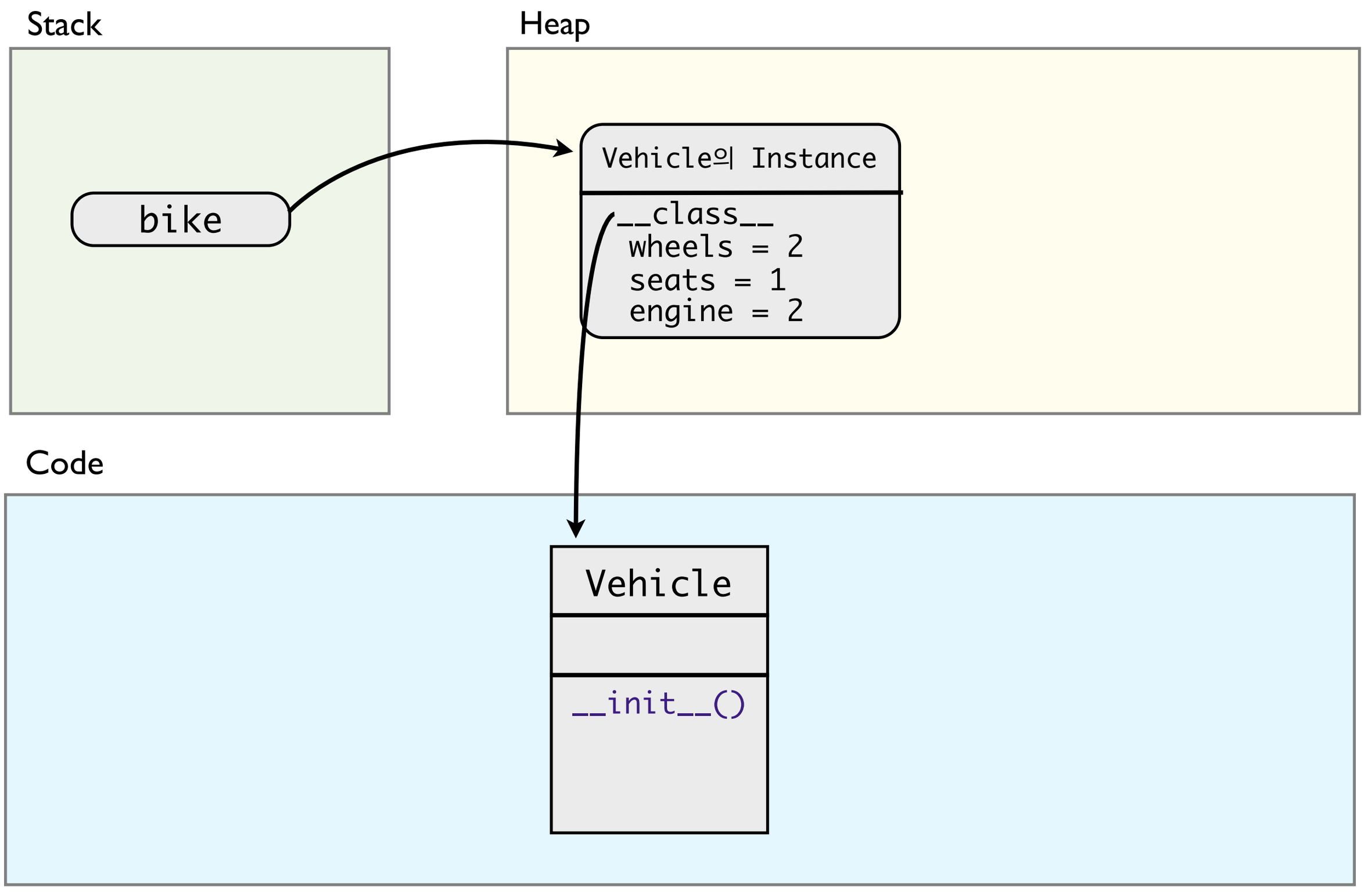

Stack
Heap
bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Vehicle
__init__()

```
print('bike.wheels = ', bike.wheels)
```

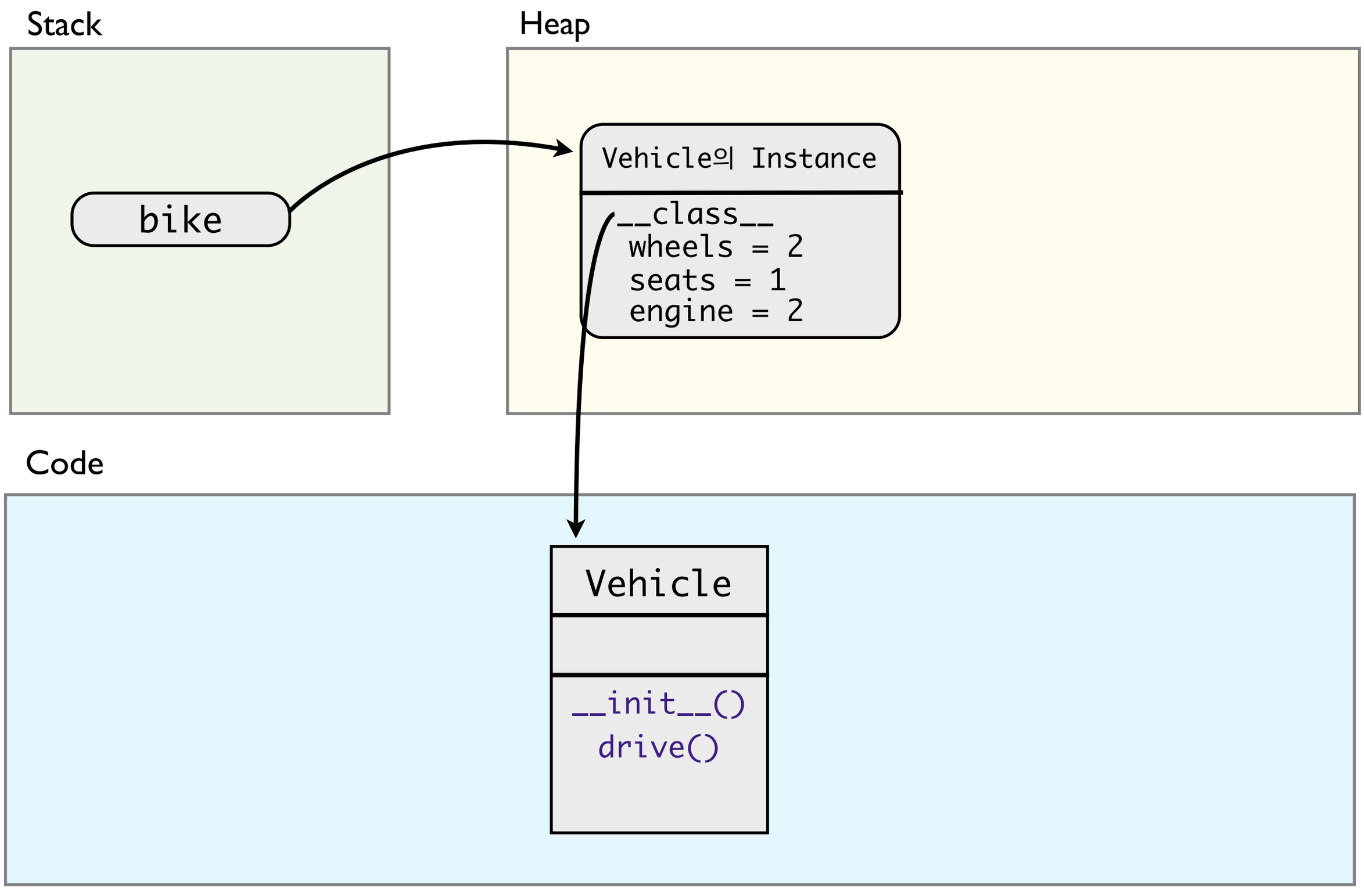


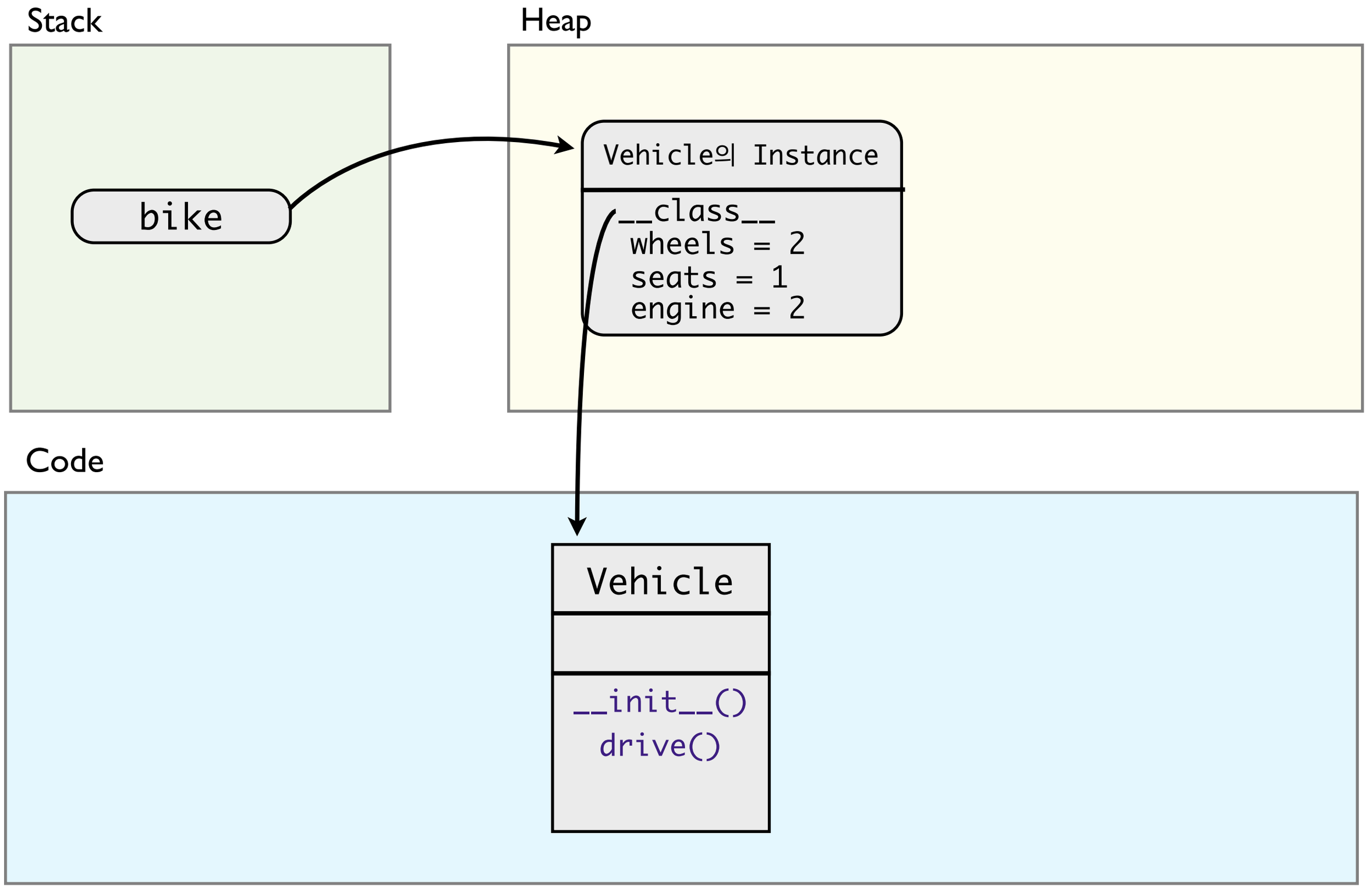

Stack
Heap
bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Vehicle
__init__()

```
bike.drive()
```

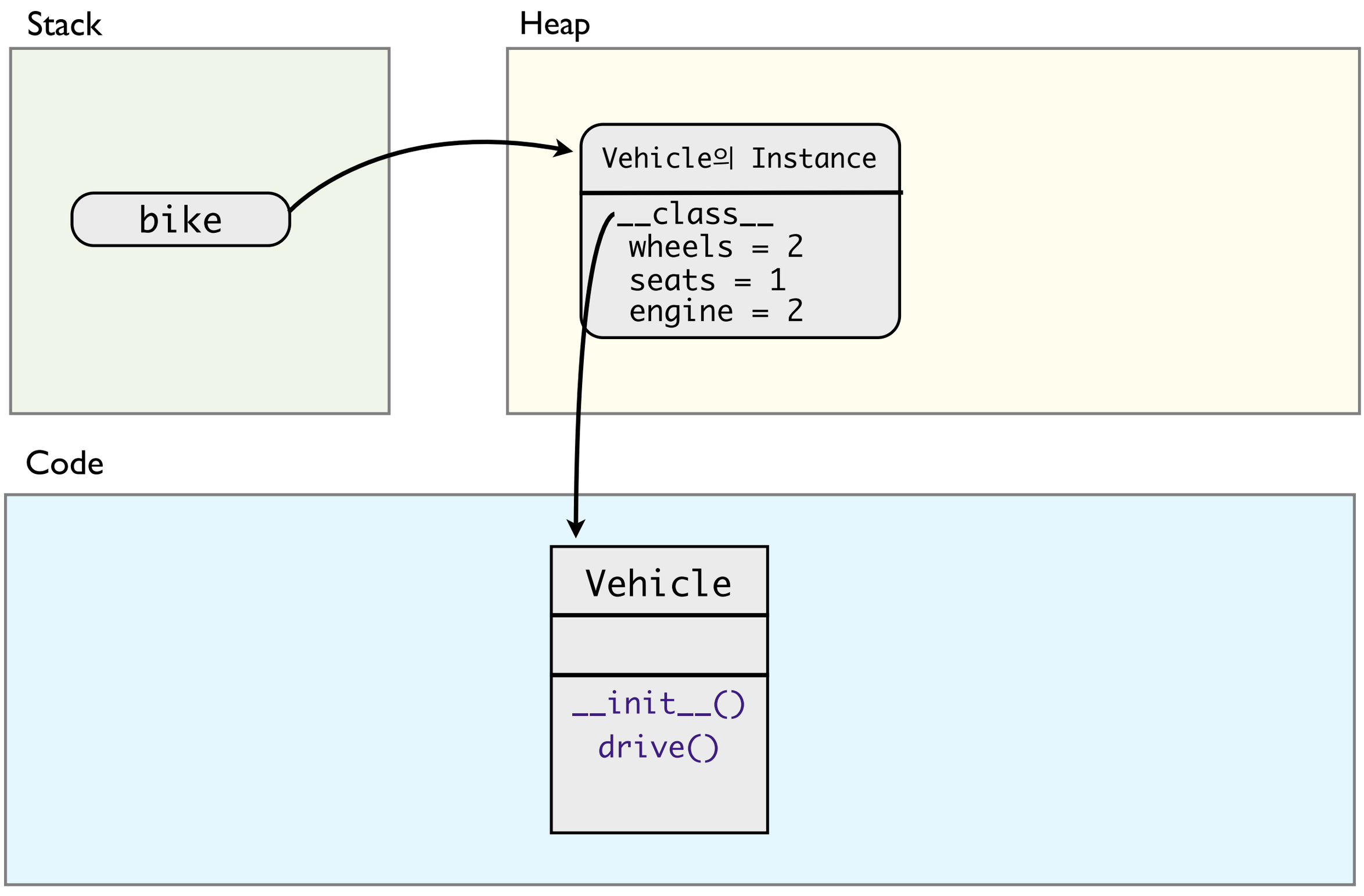

Stack
Heap
bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Vehicle
__init__()

```
bike.drive()
```

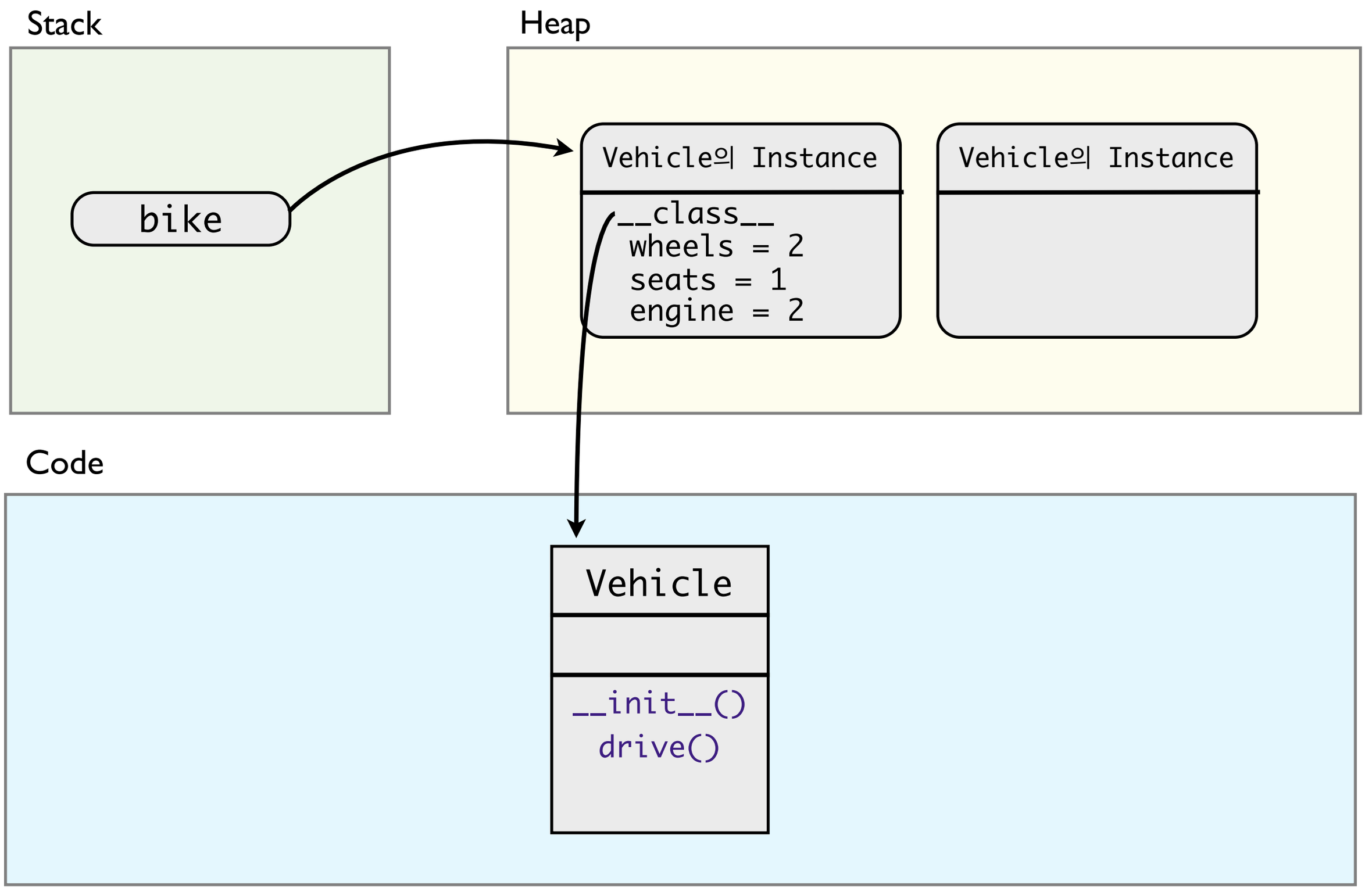

Stack
Heap
bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Vehicle
__init__()
drive()


Stack
Heap
bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Vehicle
__init__()
drive()

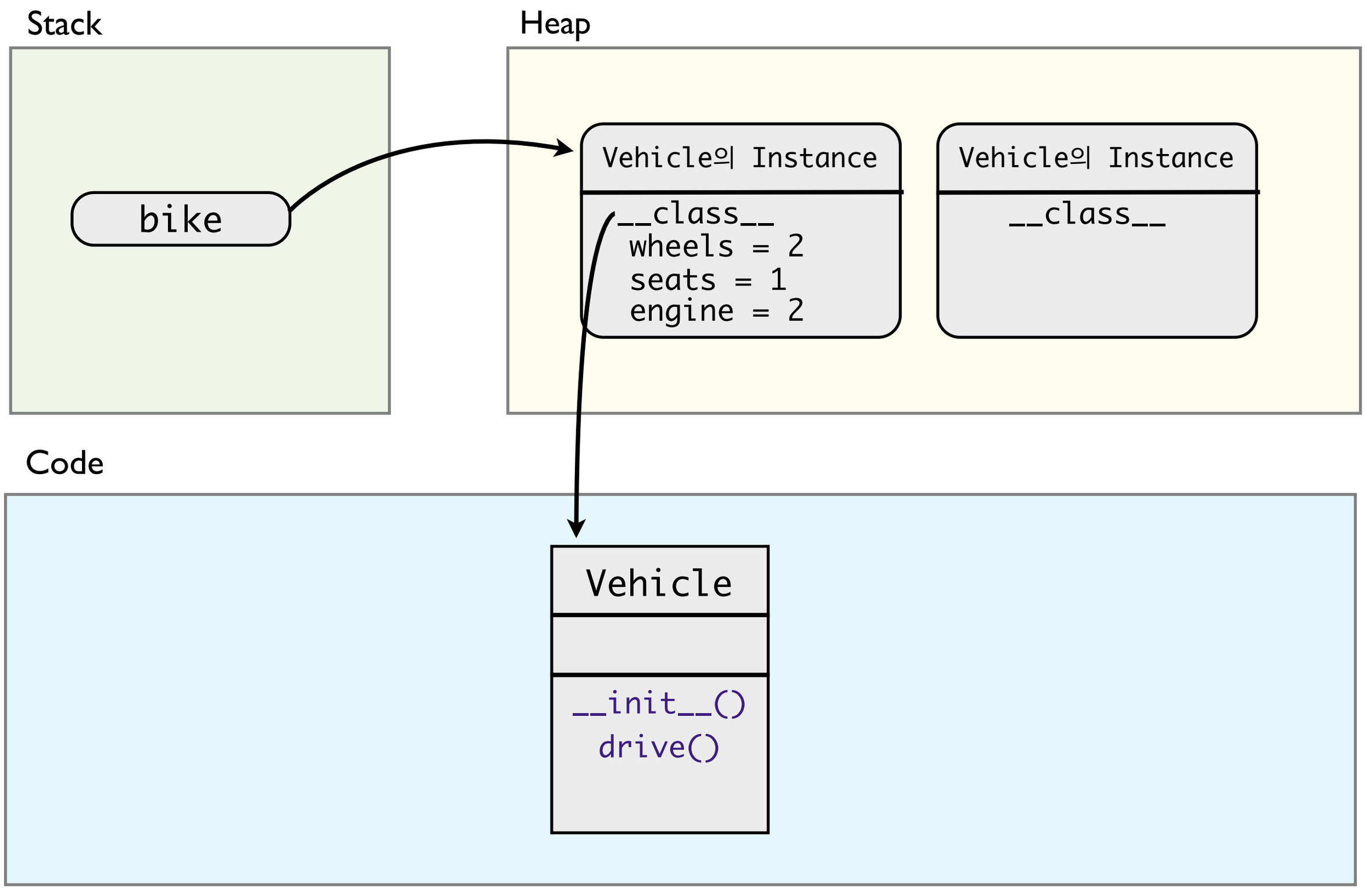
```
car = Vehicle(4, 4, 4)
```



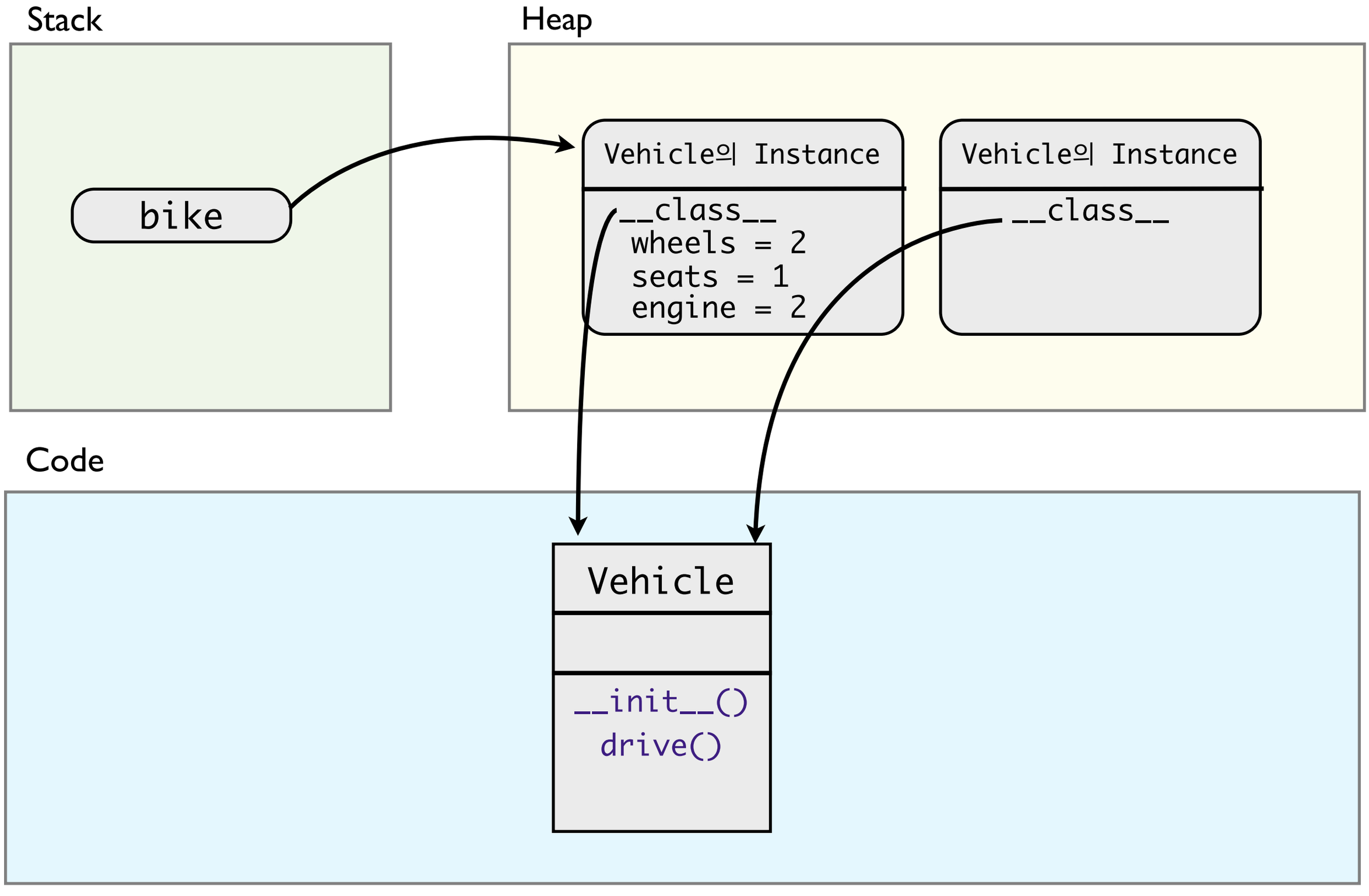
```
car = Vehicle(4, 4, 4)
```



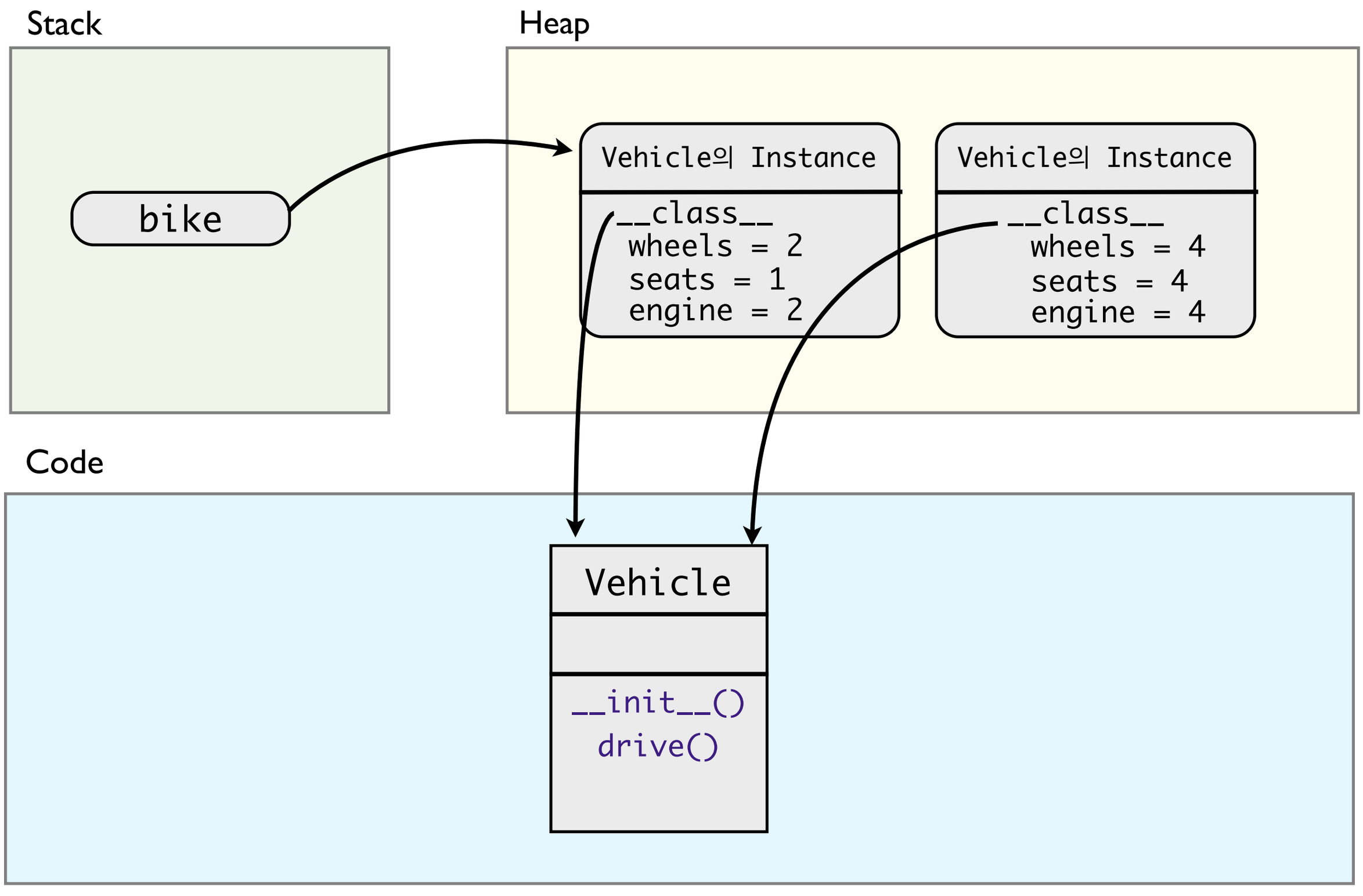
```
car = Vehicle(4, 4, 4)
```



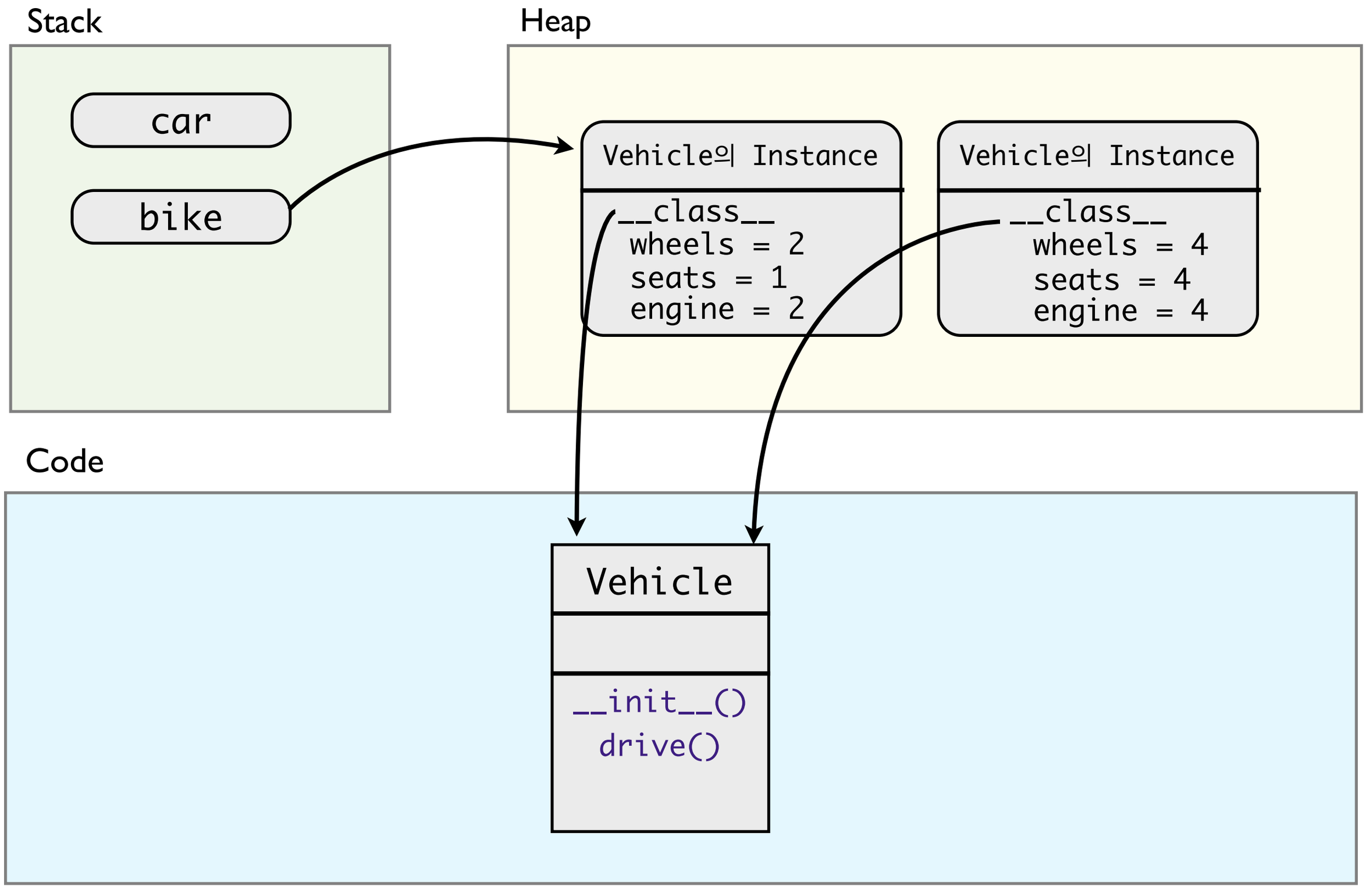
```
car = Vehicle(4, 4, 4)
```



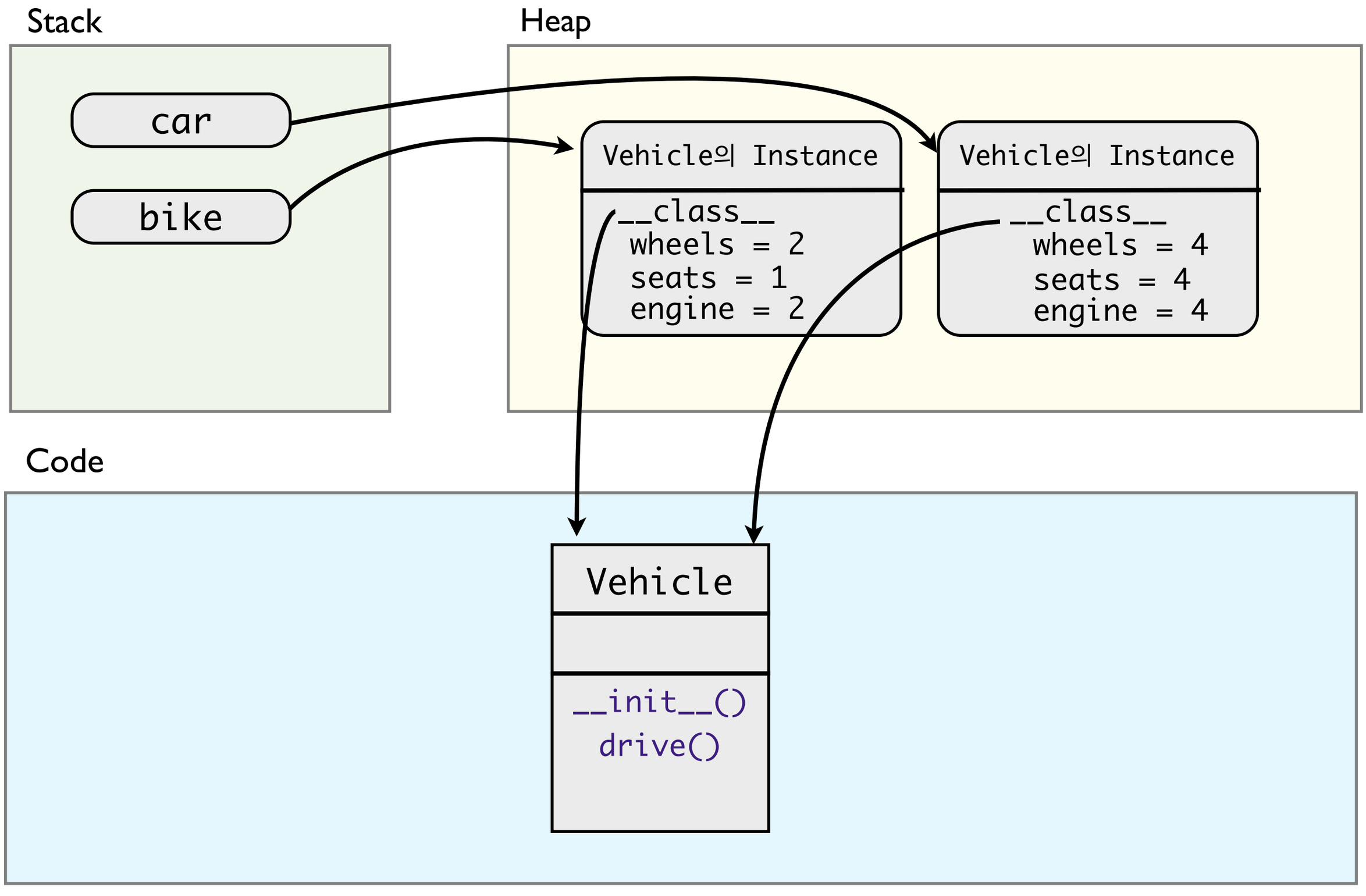
```
car = Vehicle(4, 4, 4)
```

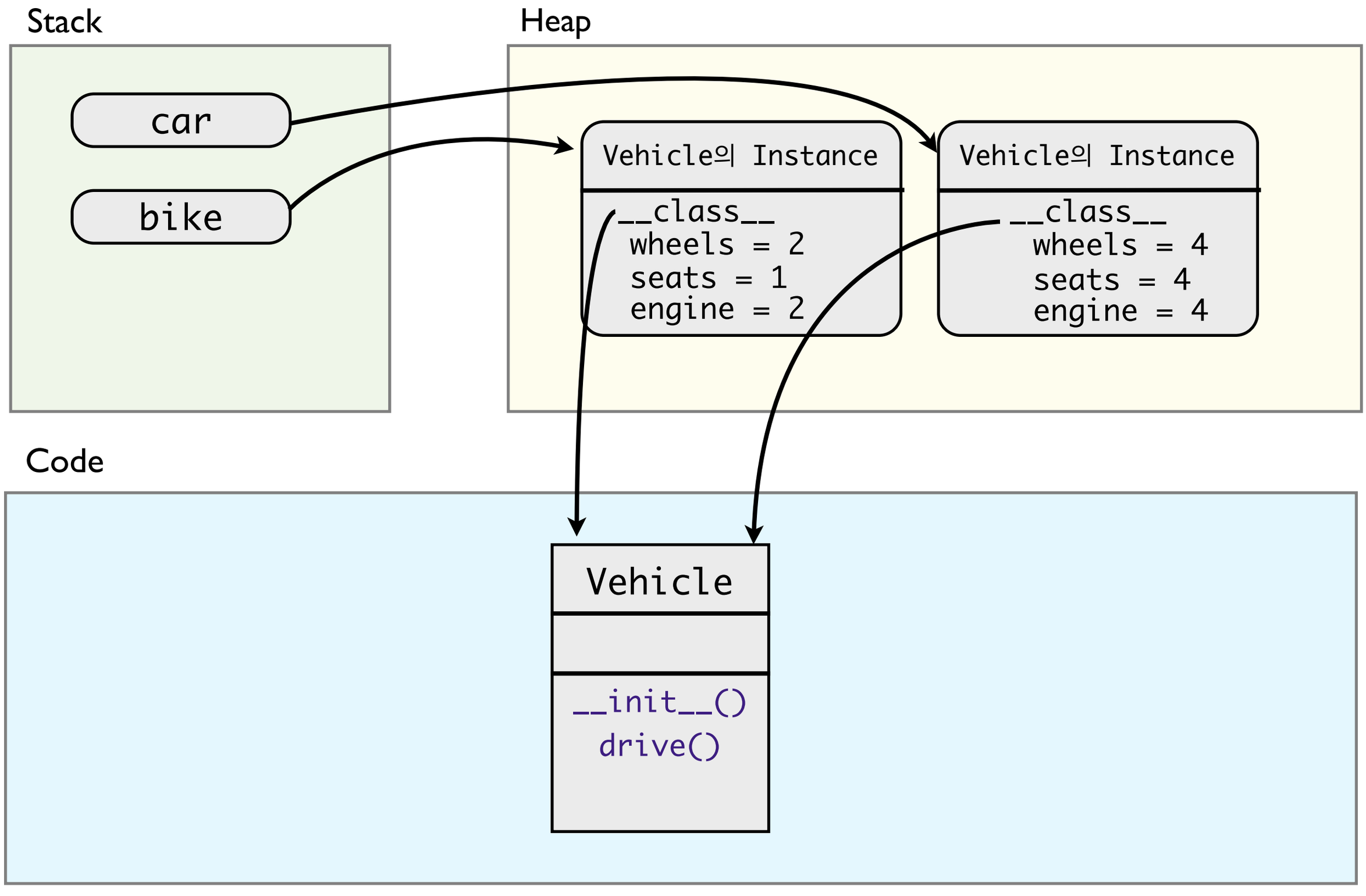


```
car = Vehicle(4, 4, 4)
```

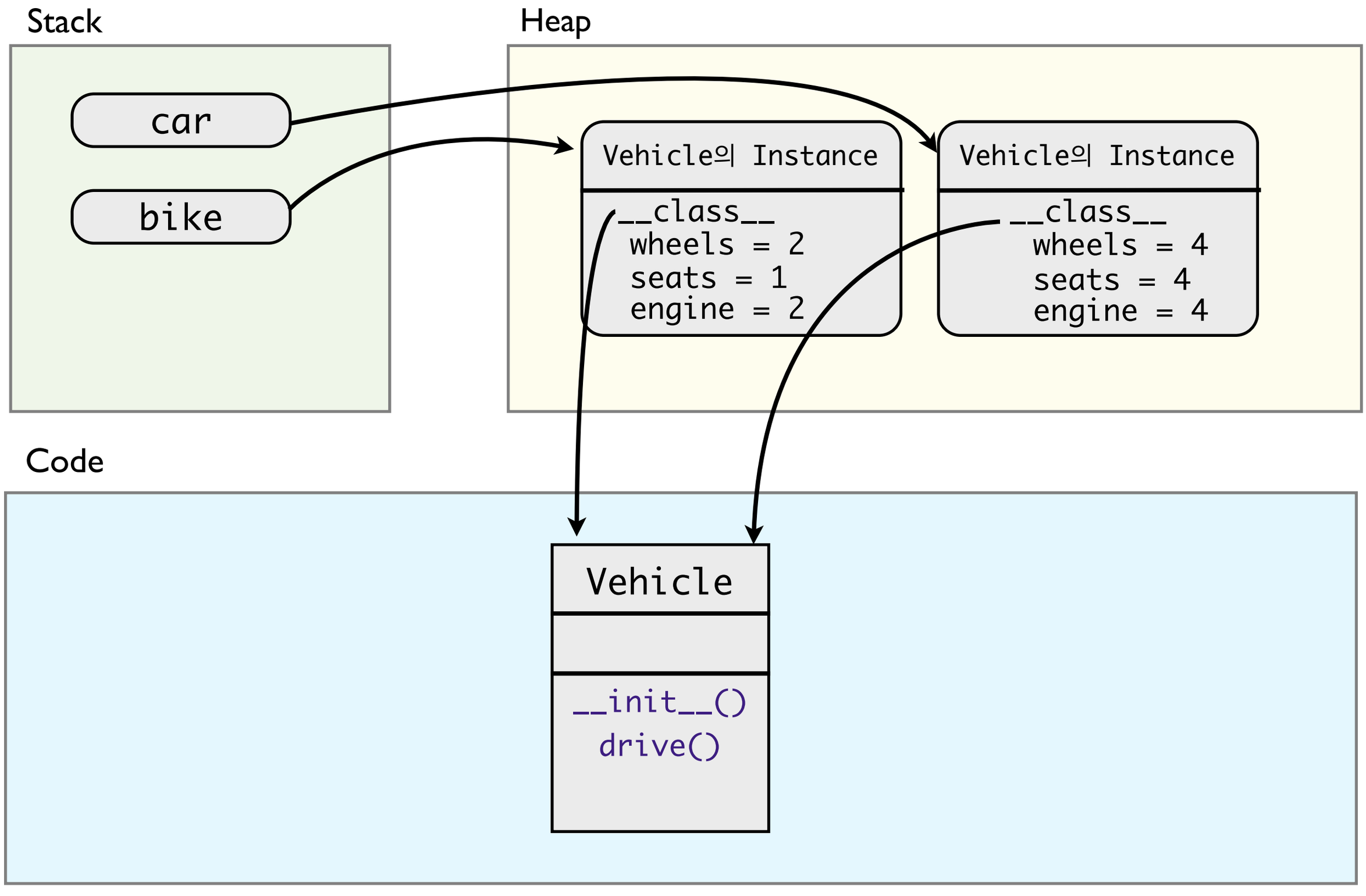


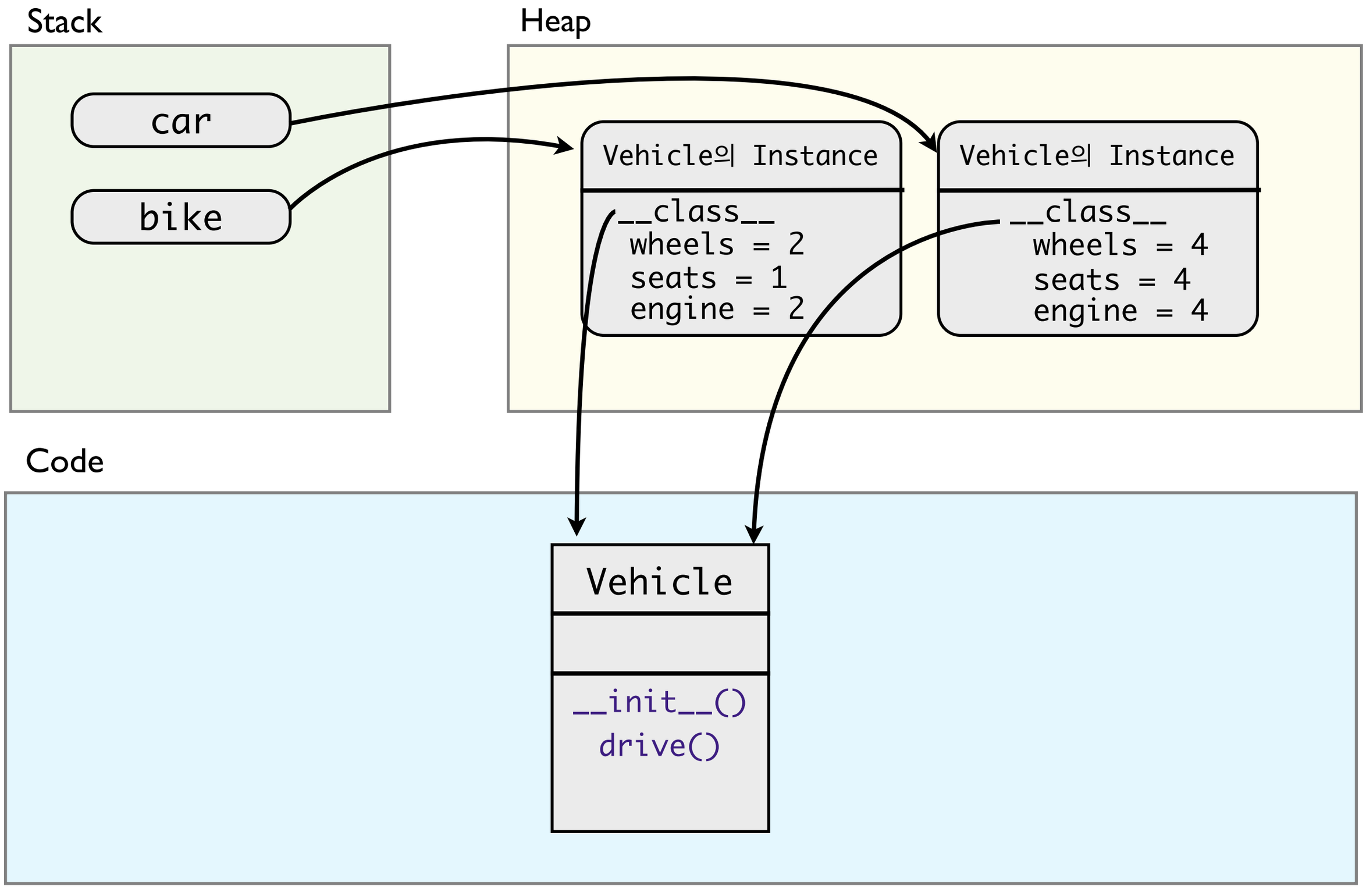
```
car = Vehicle(4, 4, 4)
```



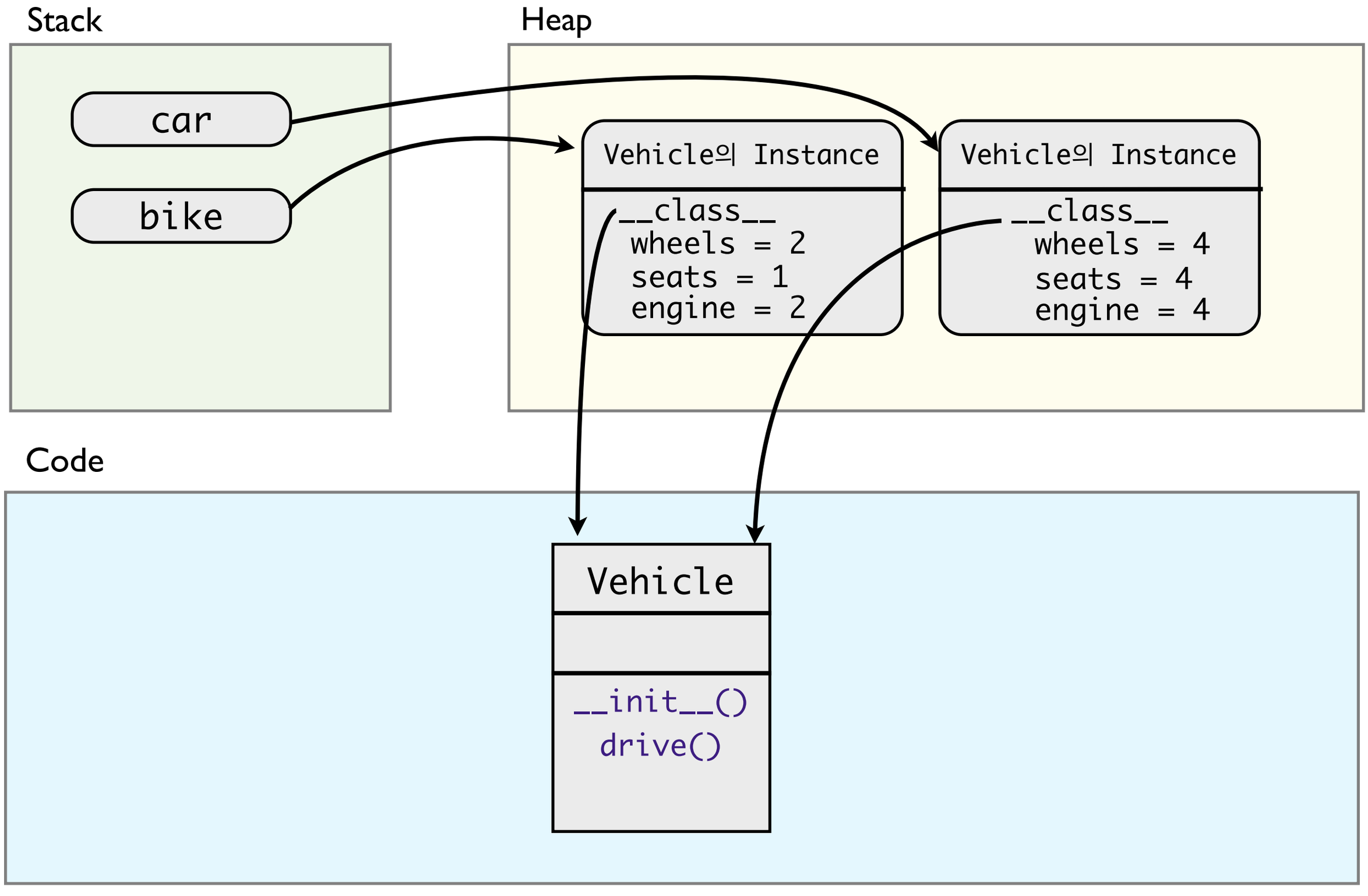

Stack
Heap
car
bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Vehicle의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Vehicle
__init__()
drive()

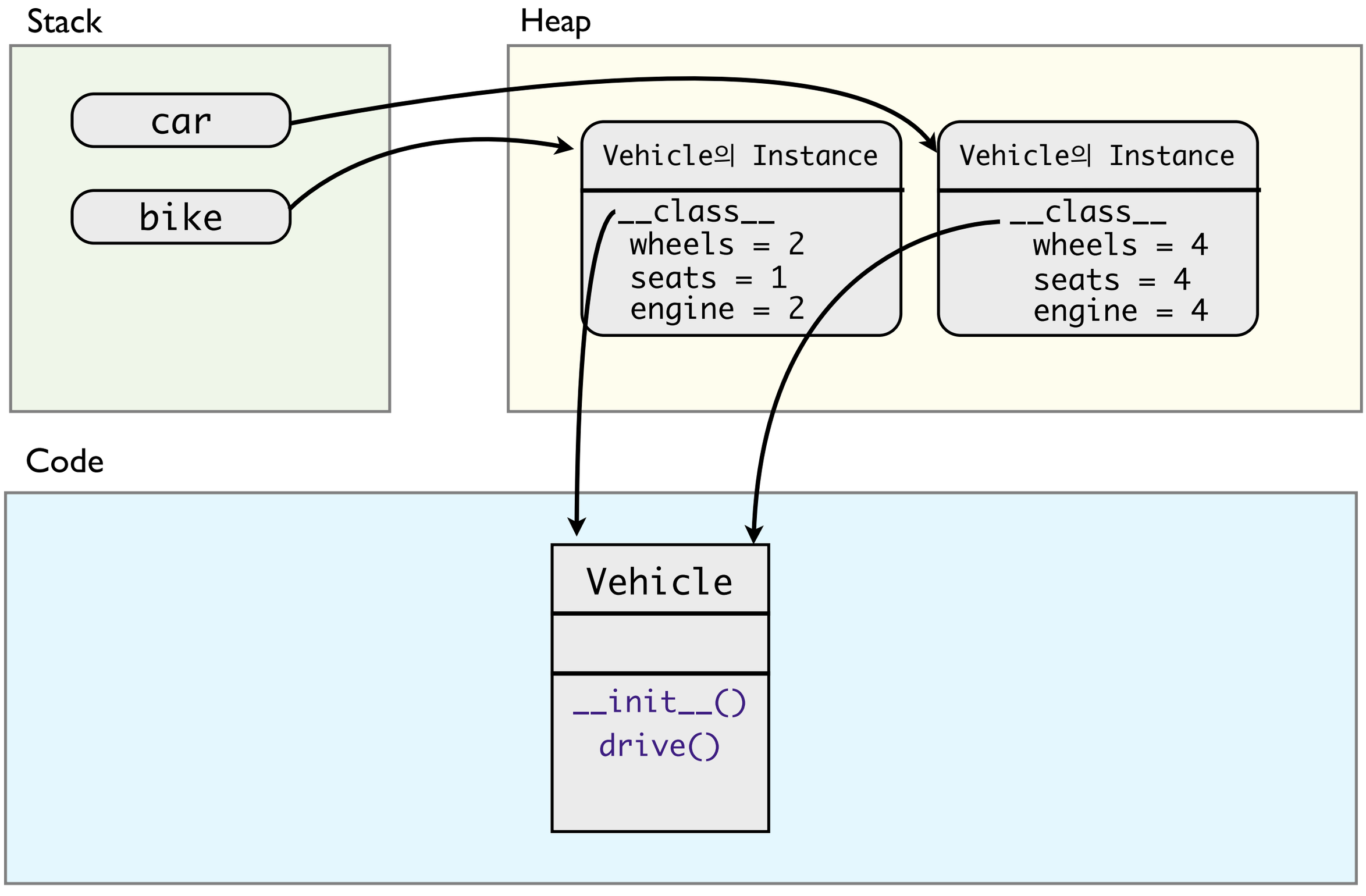
```
print('car.wheels = ', car.wheels)
```



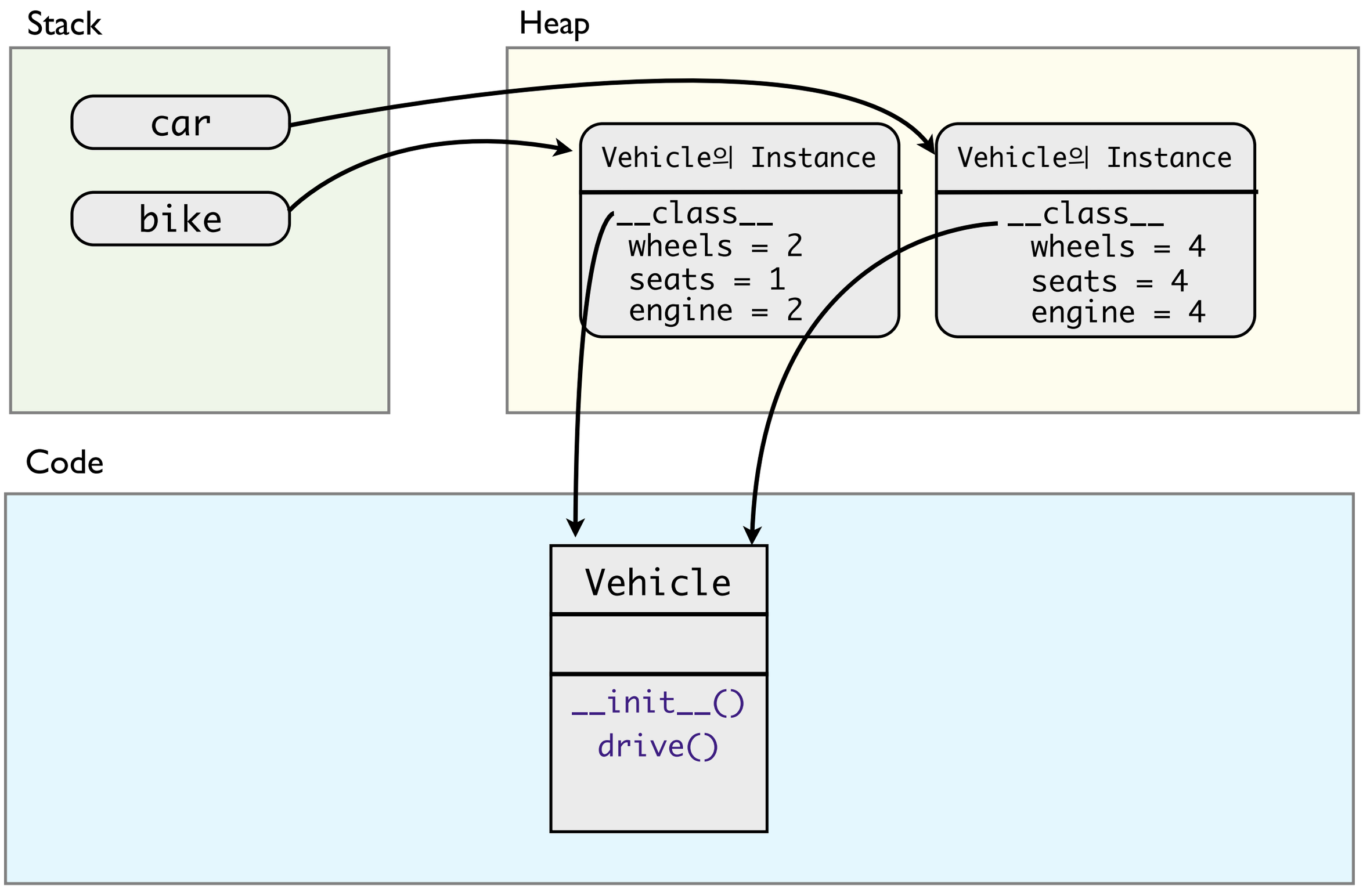

Stack
Heap
car
bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Vehicle의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Vehicle
__init__()
drive()

```
car.drive()
```

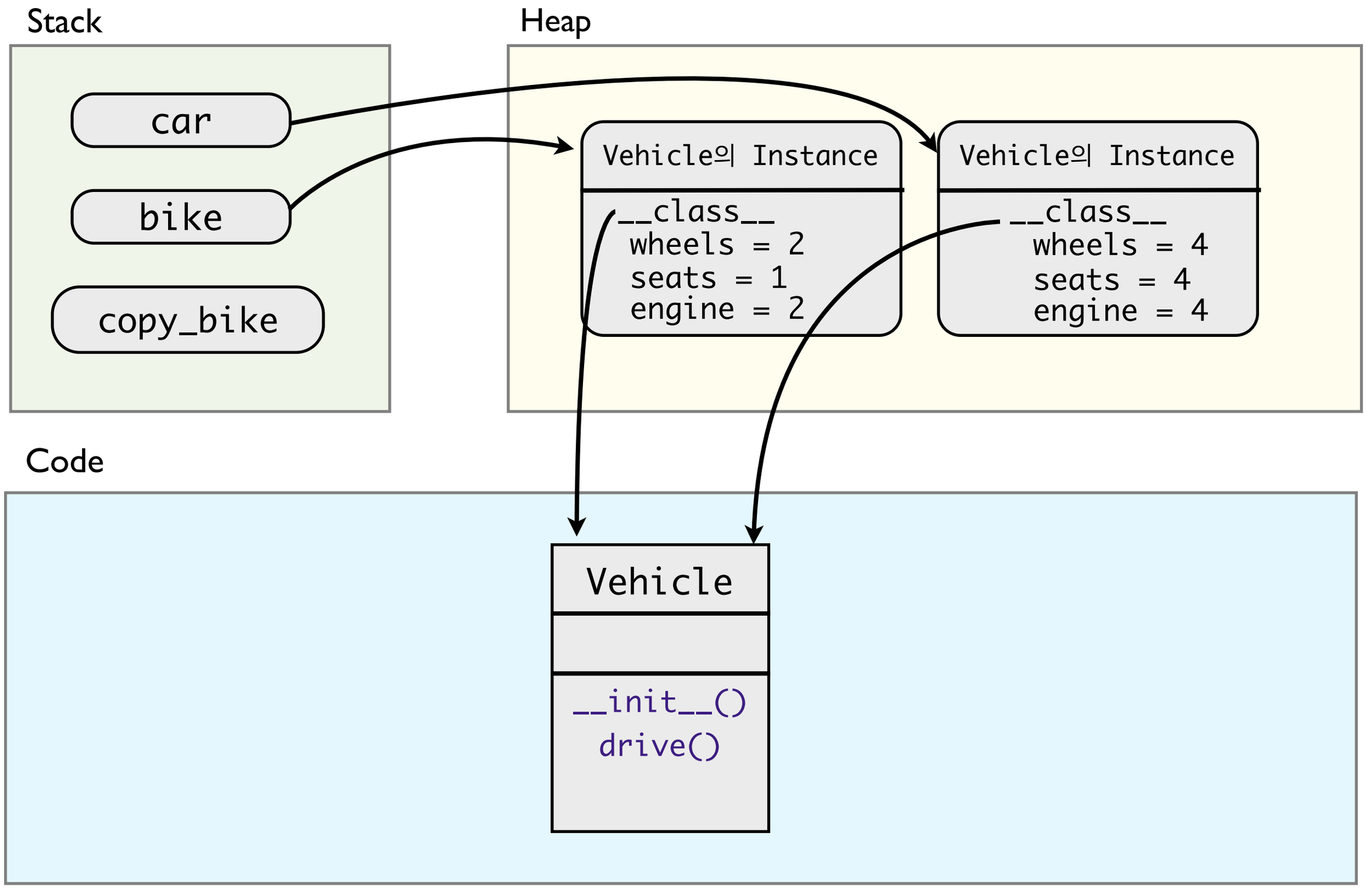

Stack
Heap
car
bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Vehicle의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Vehicle
__init__()
drive()


Stack
Heap
car
bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Vehicle의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Vehicle
__init__()
drive()

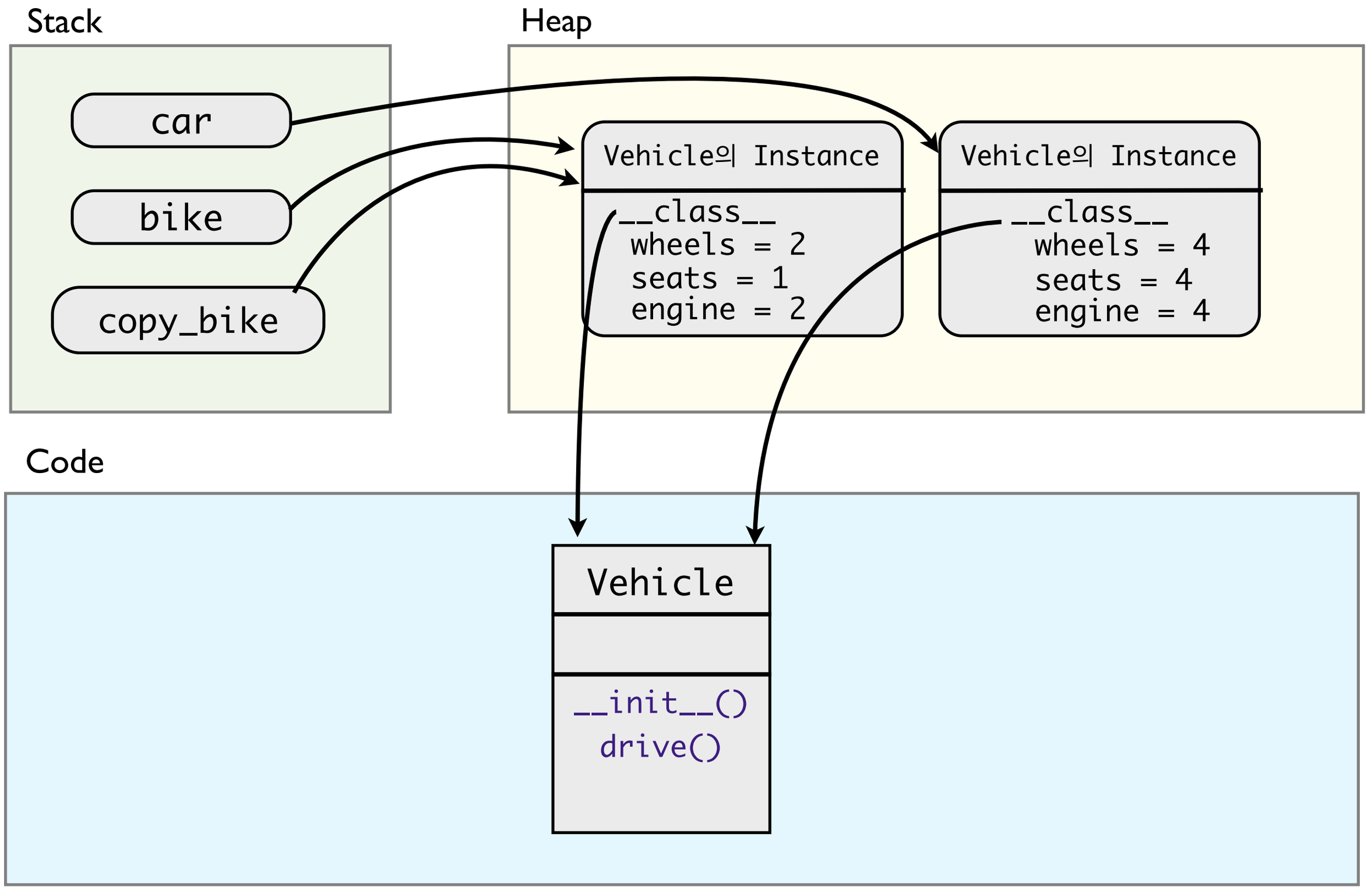
copy_bike = bike

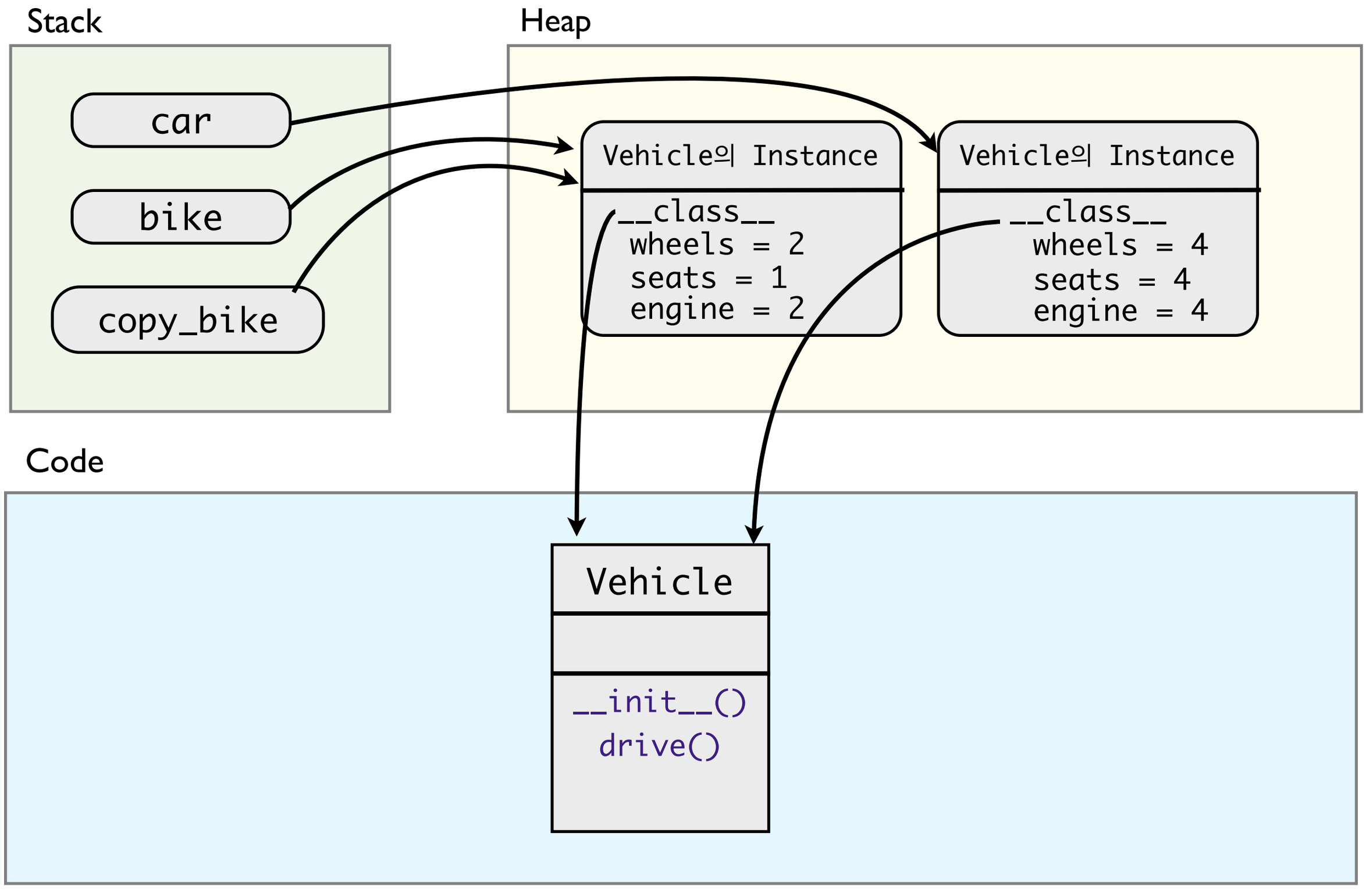


copy_bike = bike

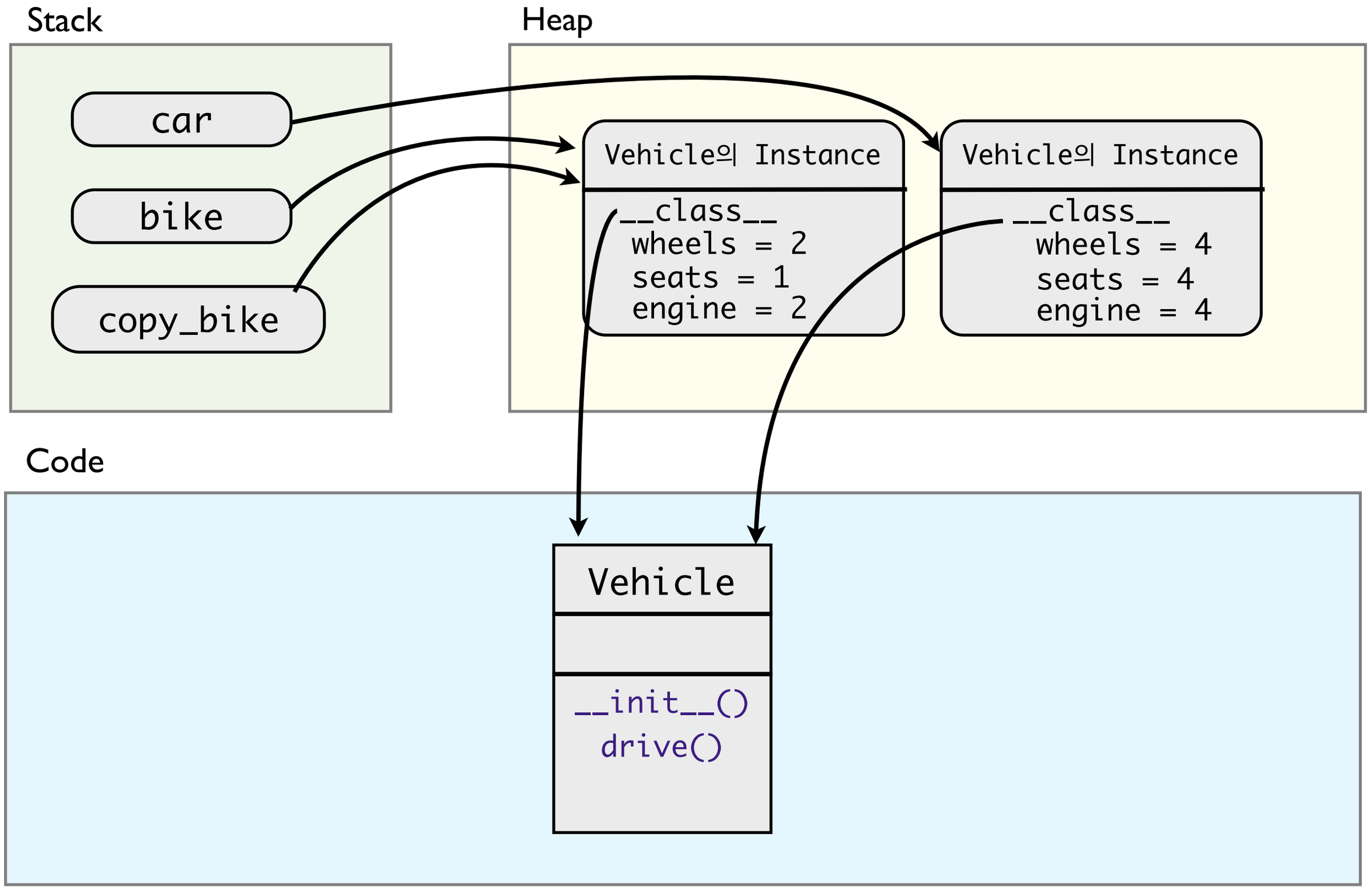


copy_bike = bike

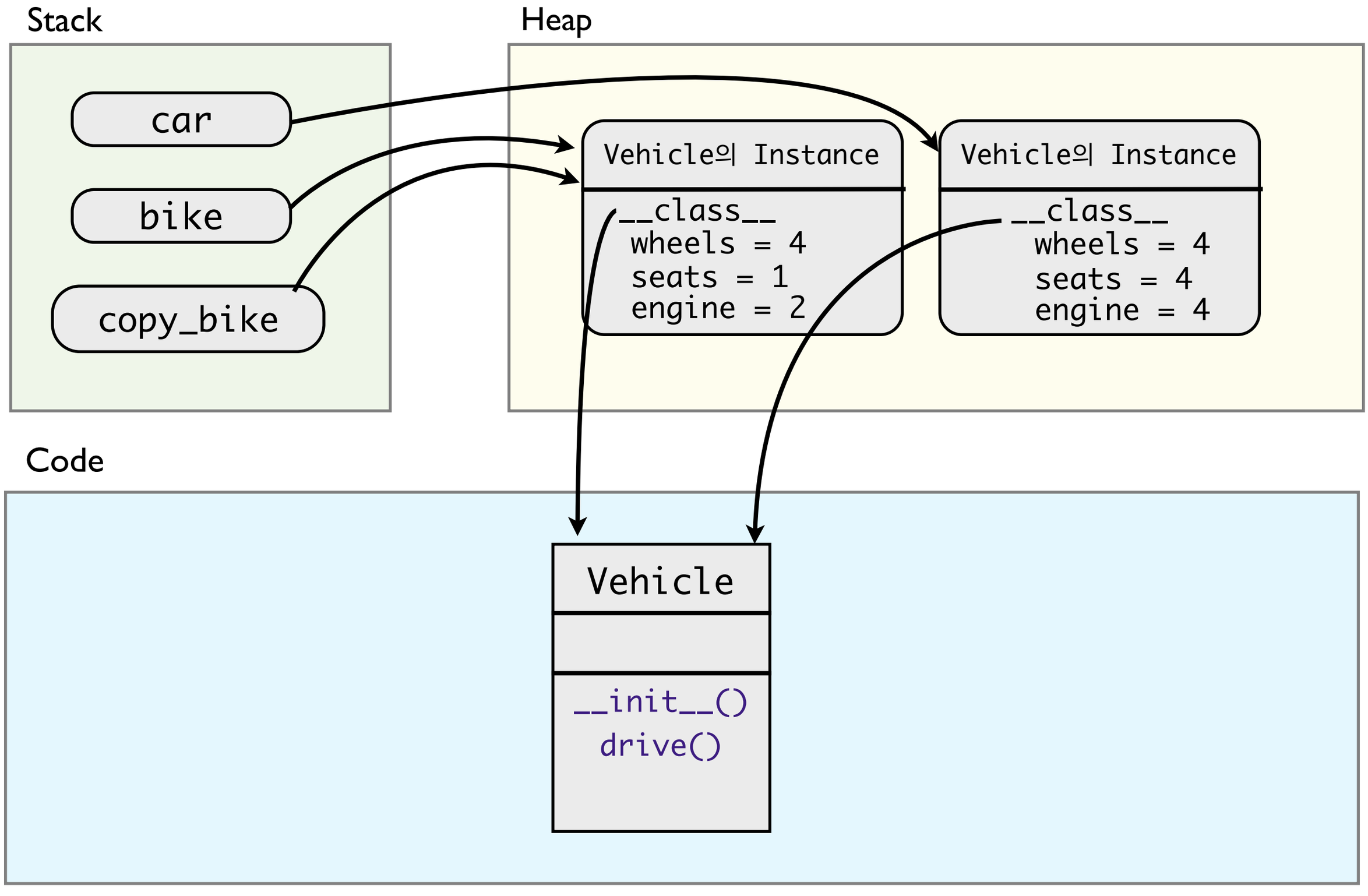


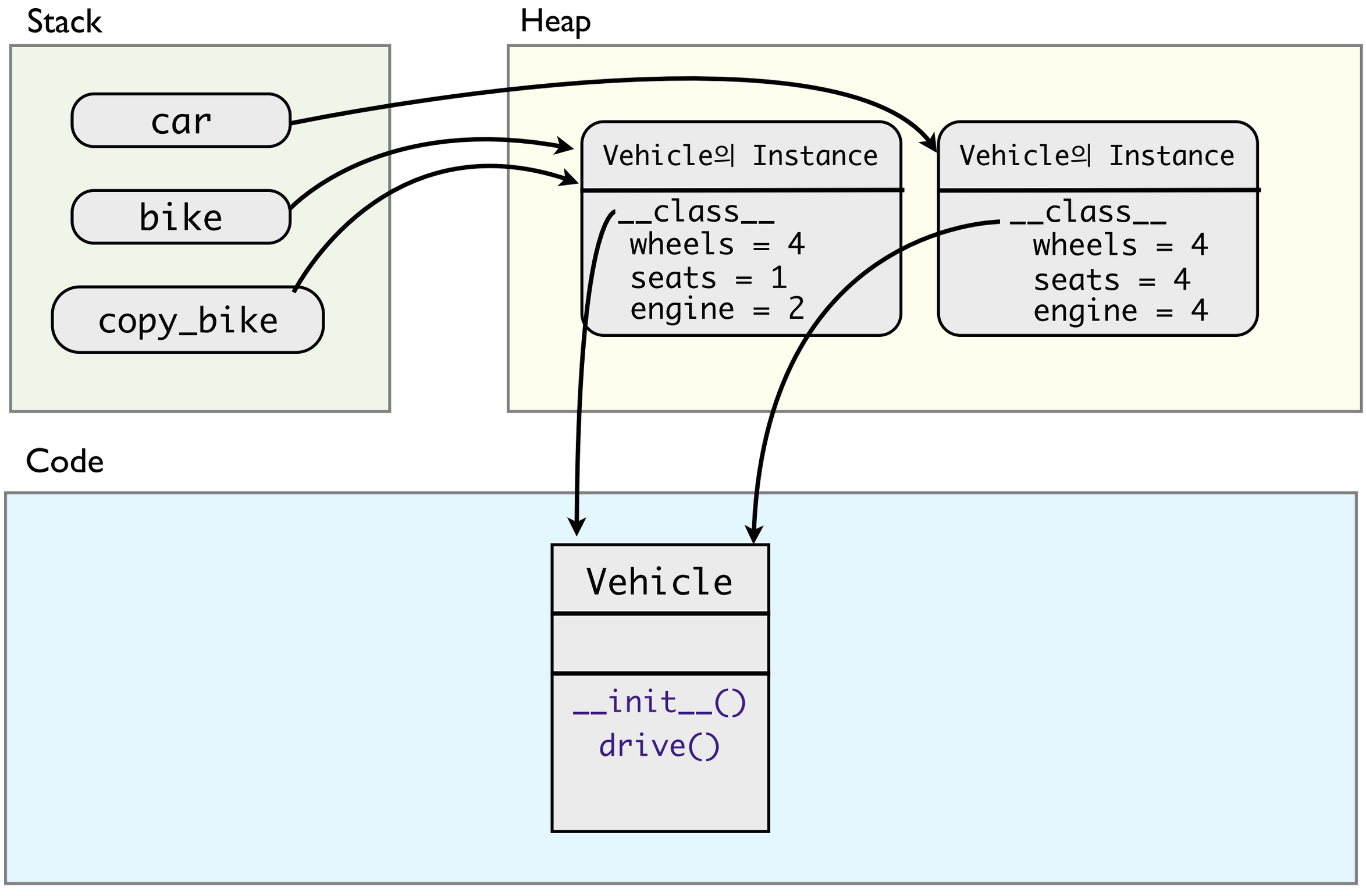

Stack
Heap
car
bike
copy_bike
Vehicle의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Vehicle의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Vehicle
__init__()
drive()

```
copy_bike.wheels = 4
```

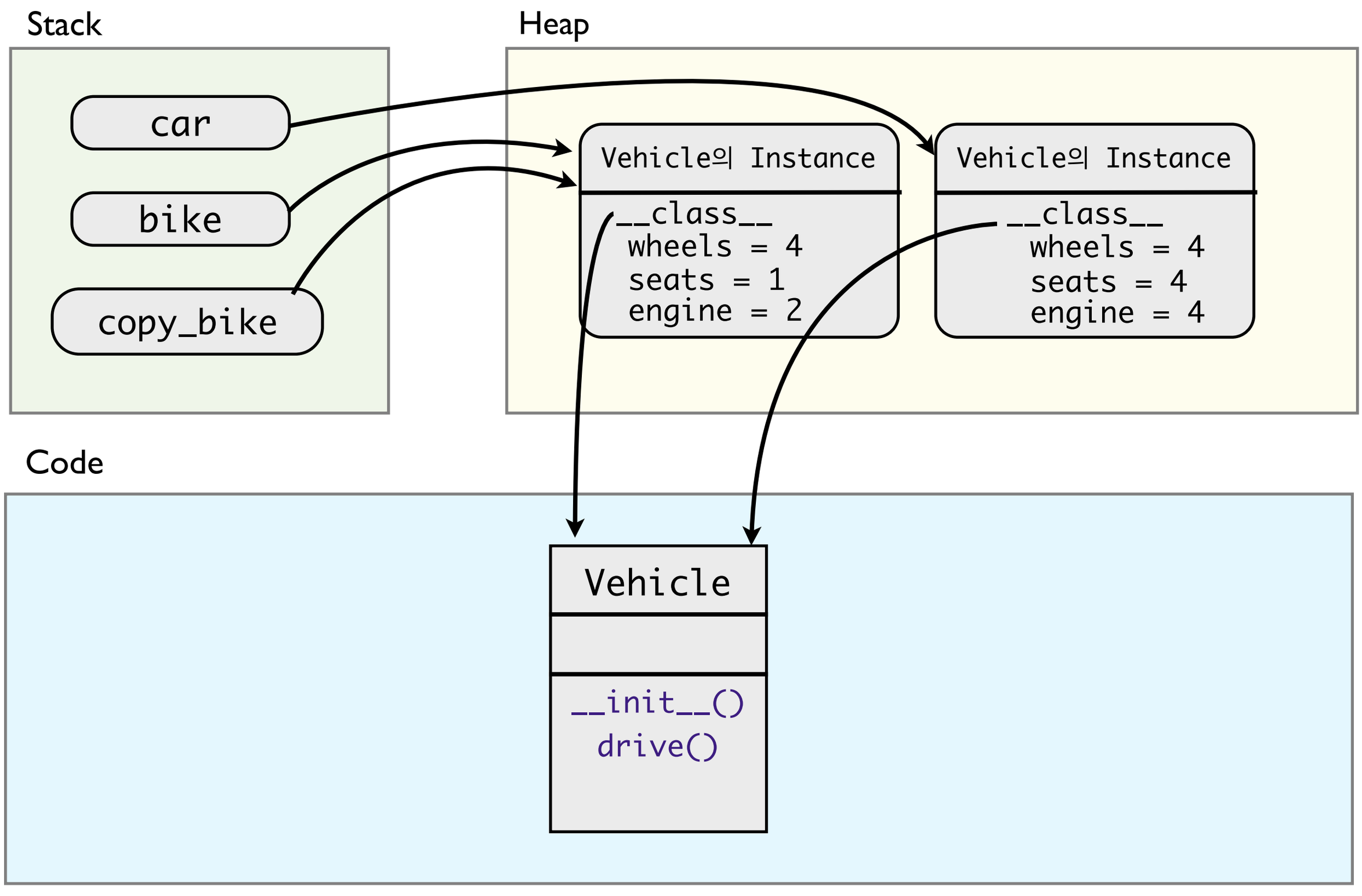


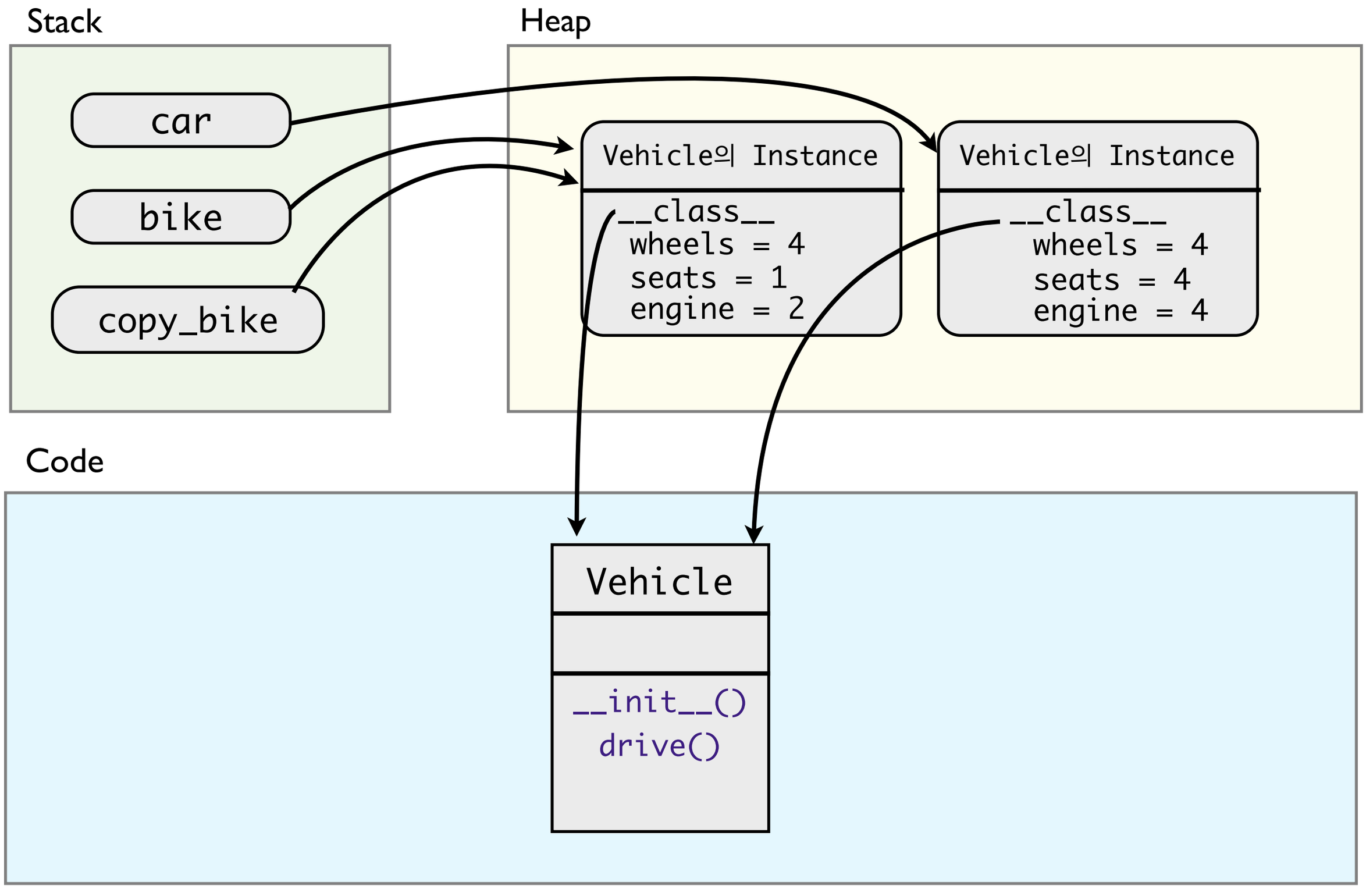
copy_bike.wheels = 4




Stack
Heap
car
bike
copy_bike
Vehicle의 Instance
__class__
wheels = 4
seats = 1
engine = 2
Vehicle의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Vehicle
__init__()
drive()

```
print('bike.wheels = ', bike.wheels)
```




Stack
Heap
car
bike
copy_bike
Vehicle의 Instance
__class__
wheels = 4
seats = 1
engine = 2
Vehicle의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Vehicle
__init__()
drive()

상속

```
class Car(Vehicle):
    def __init__(self):
        super().__init__(4, 5, 4)

    def shift_gear():
        print('변속')


car1 = Car()
car1.wheels = 6
print('car1.wheels = ', car1.wheels)
print('car1.seats = ', car1.seats)
car1.drive()
```

Stack

Heap

Code

```
car1 = Car()
```

Stack

Heap

Code

```
car1 = Car()
```

Stack

Heap

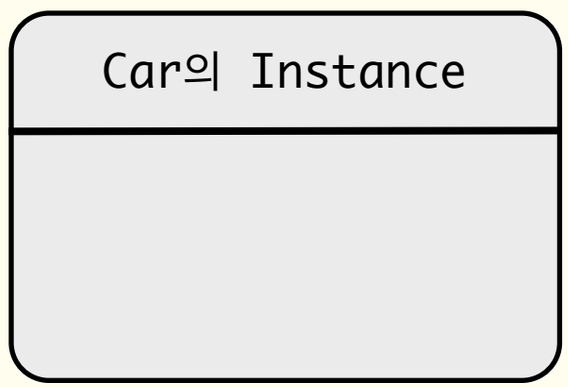
Code

Car

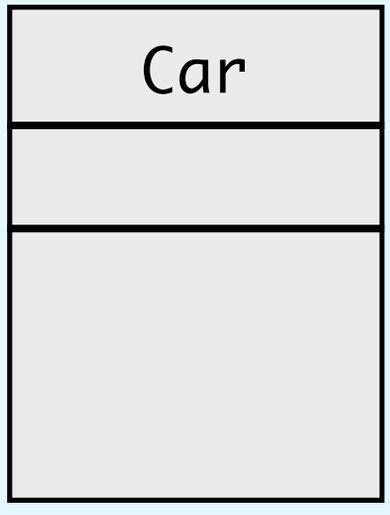
```
car1 = Car()
```

Stack

Heap



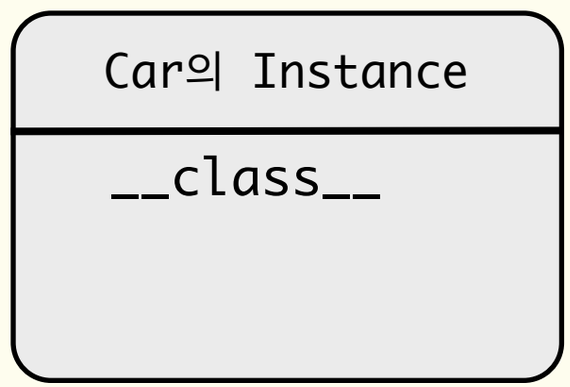
Code



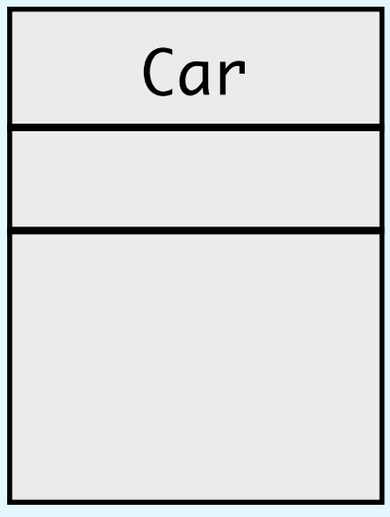
```
car1 = Car()
```

Stack

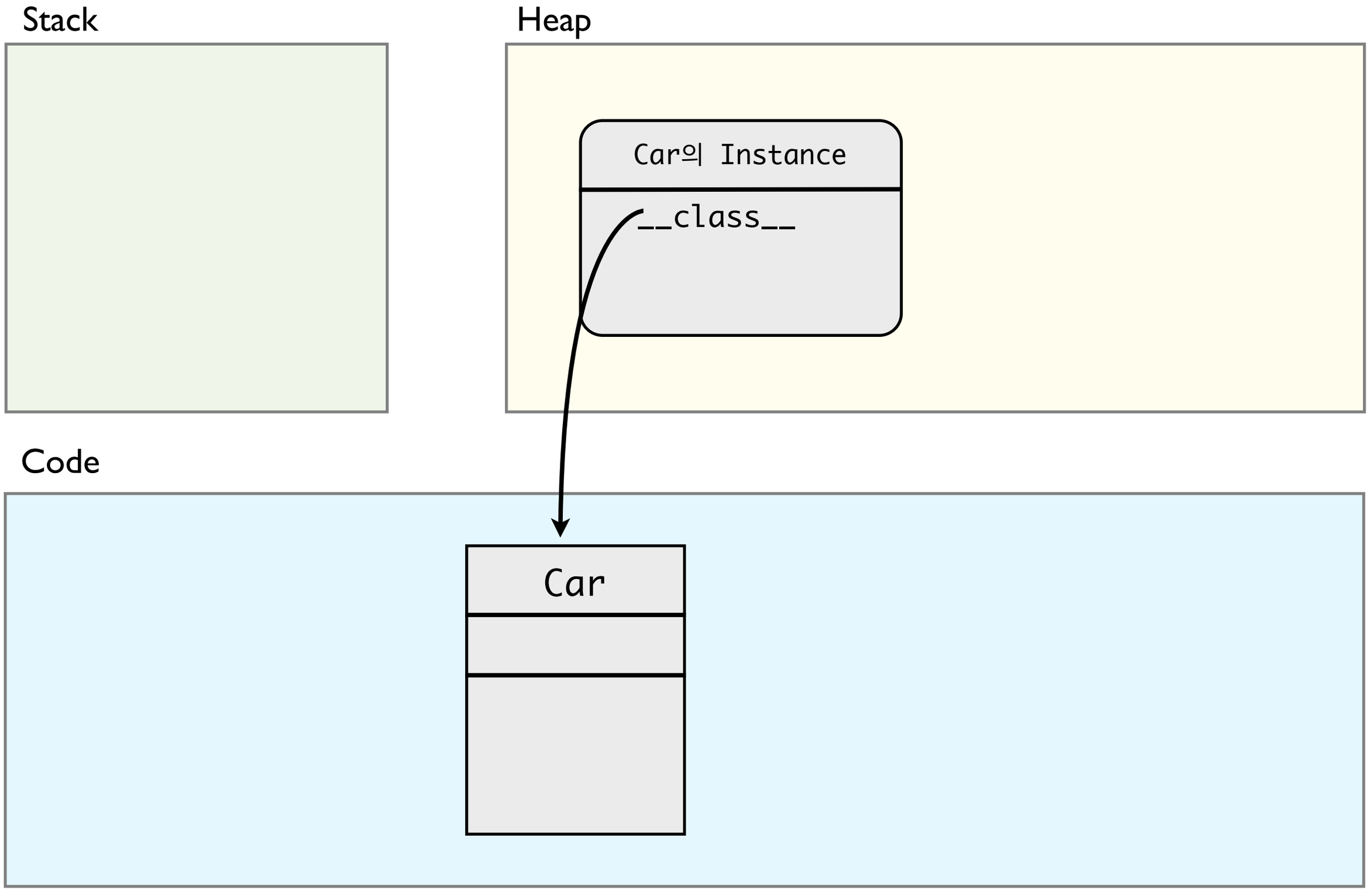
Heap



Code



```
car1 = Car()
```



```
car1 = Car()
```

Stack

Heap

Car의 Instance

__class__

Code

Car

__init__

```
car1 = Car()
```

Stack

Heap

Car의 Instance

__class__

Code

Car

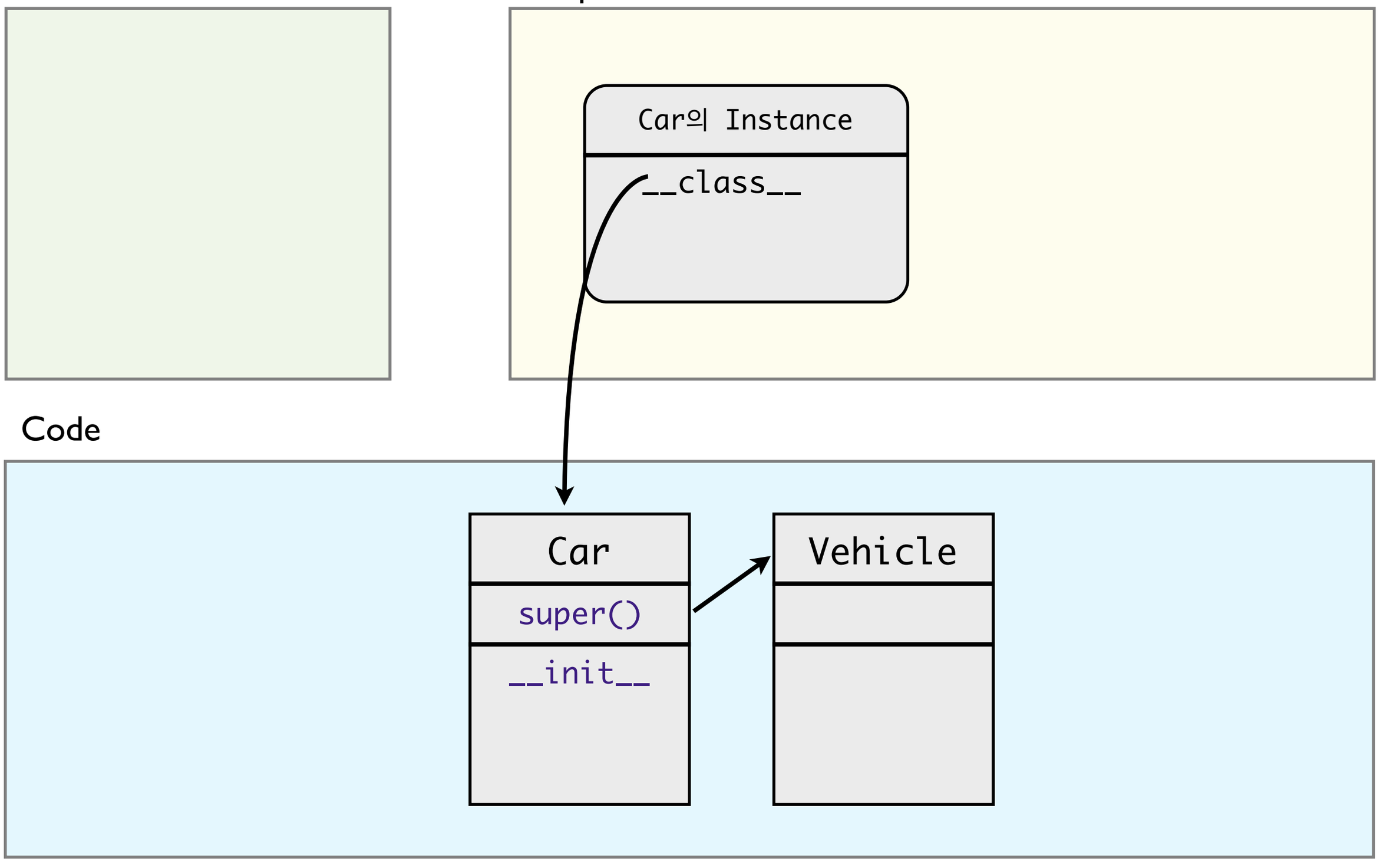
super()

__init__

```
car1 = Car()
```

Stack

Heap



```
car1 = Car()
```

Stack

Heap

Car의 Instance

__class__

Code

Car

super()

__init__

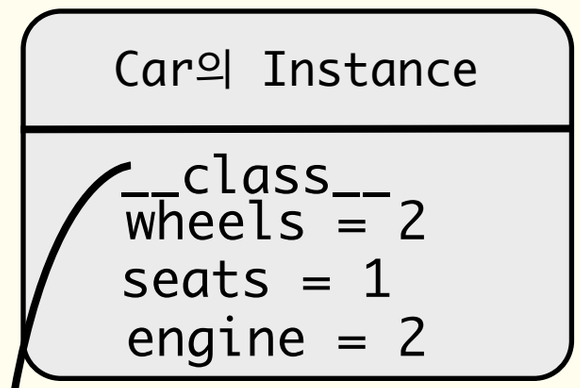
Vehicle

__init__

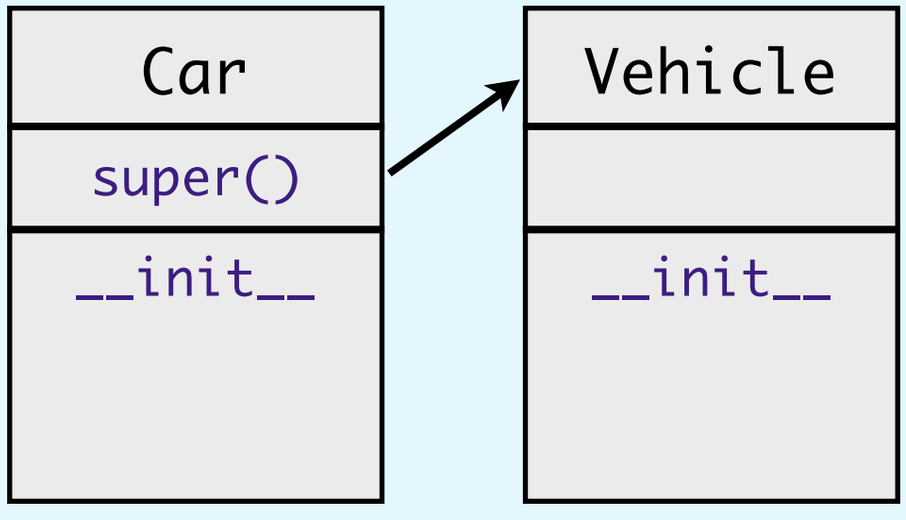
```
car1 = Car()
```

Stack

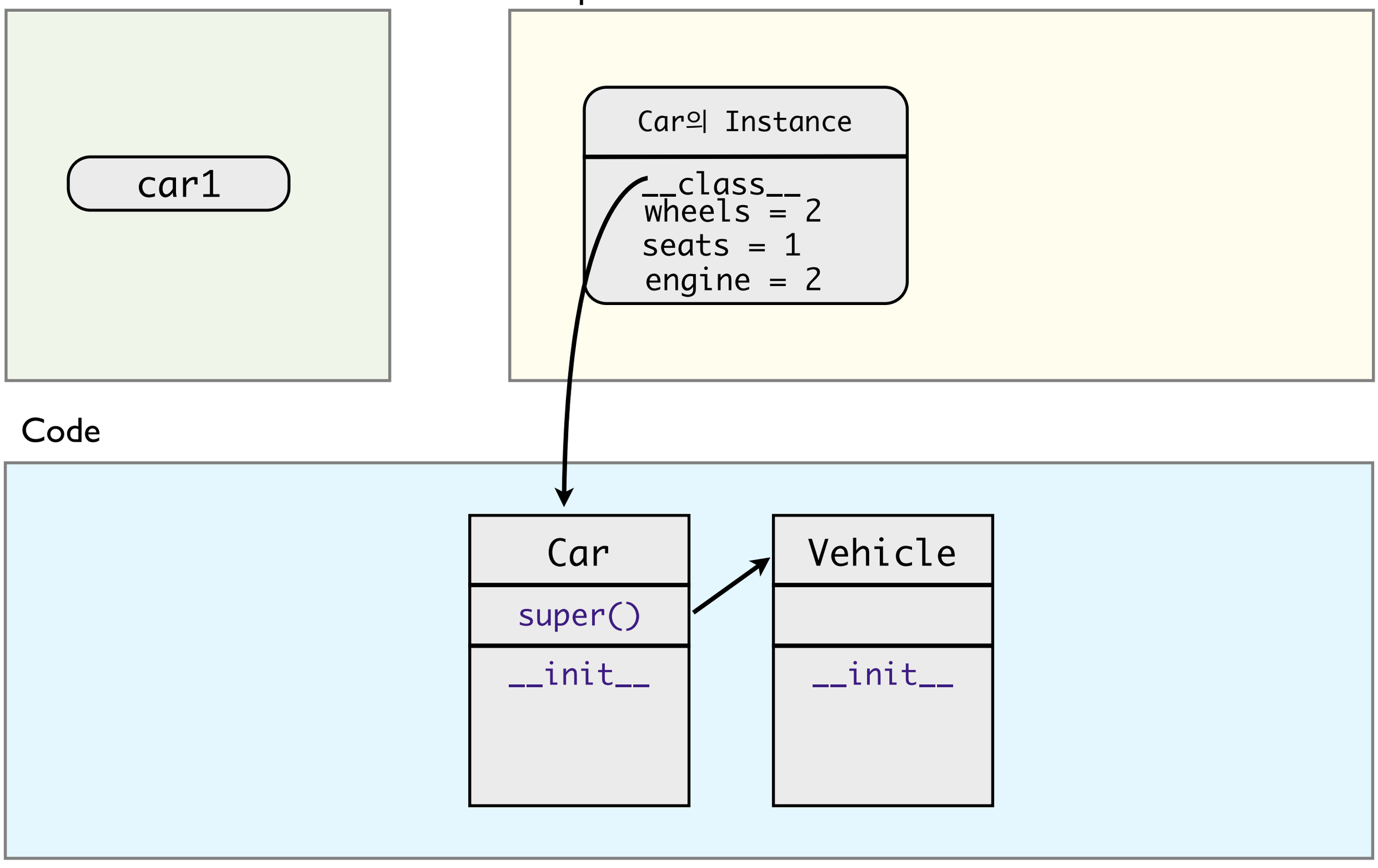
Heap



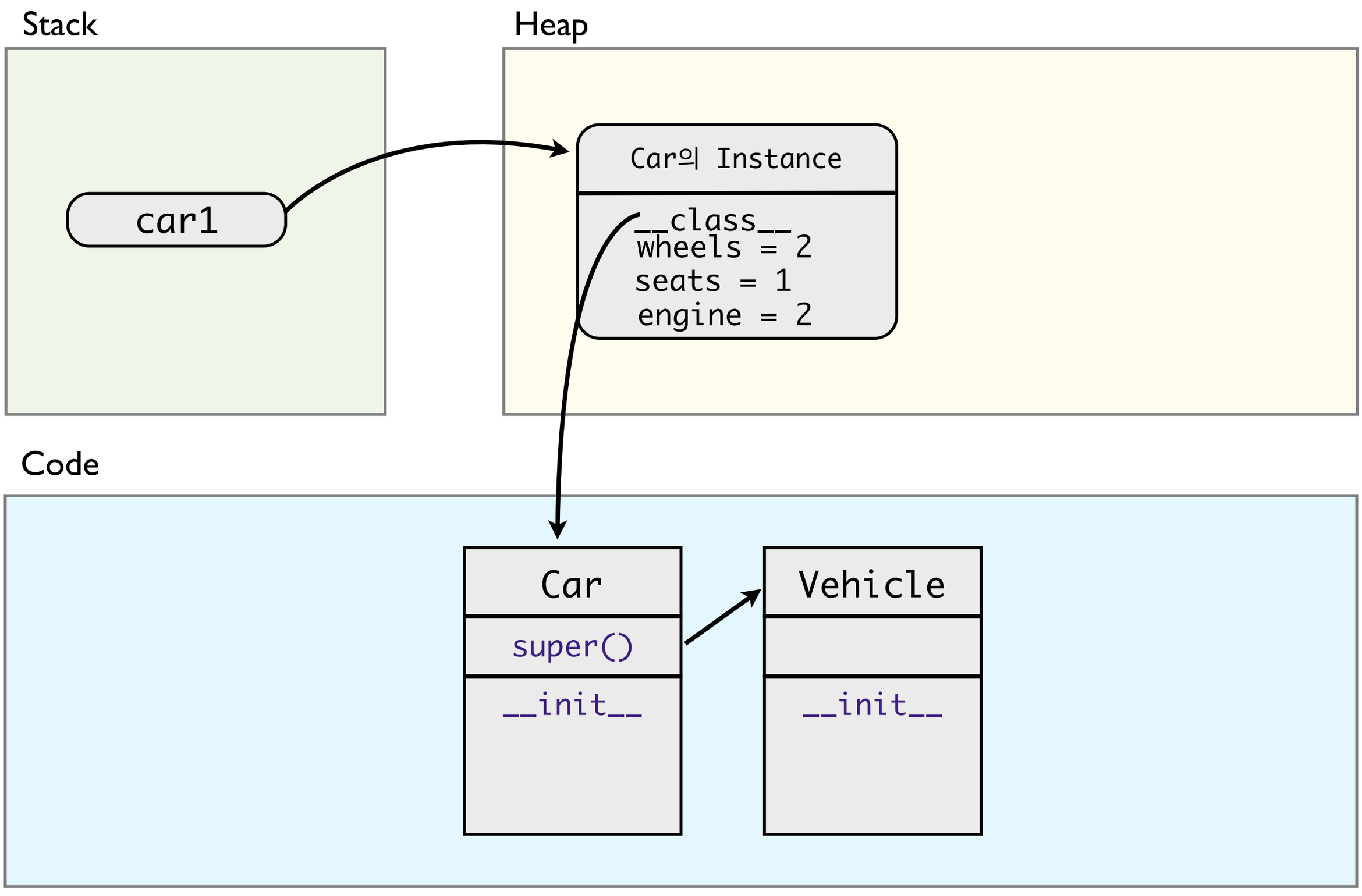
Code

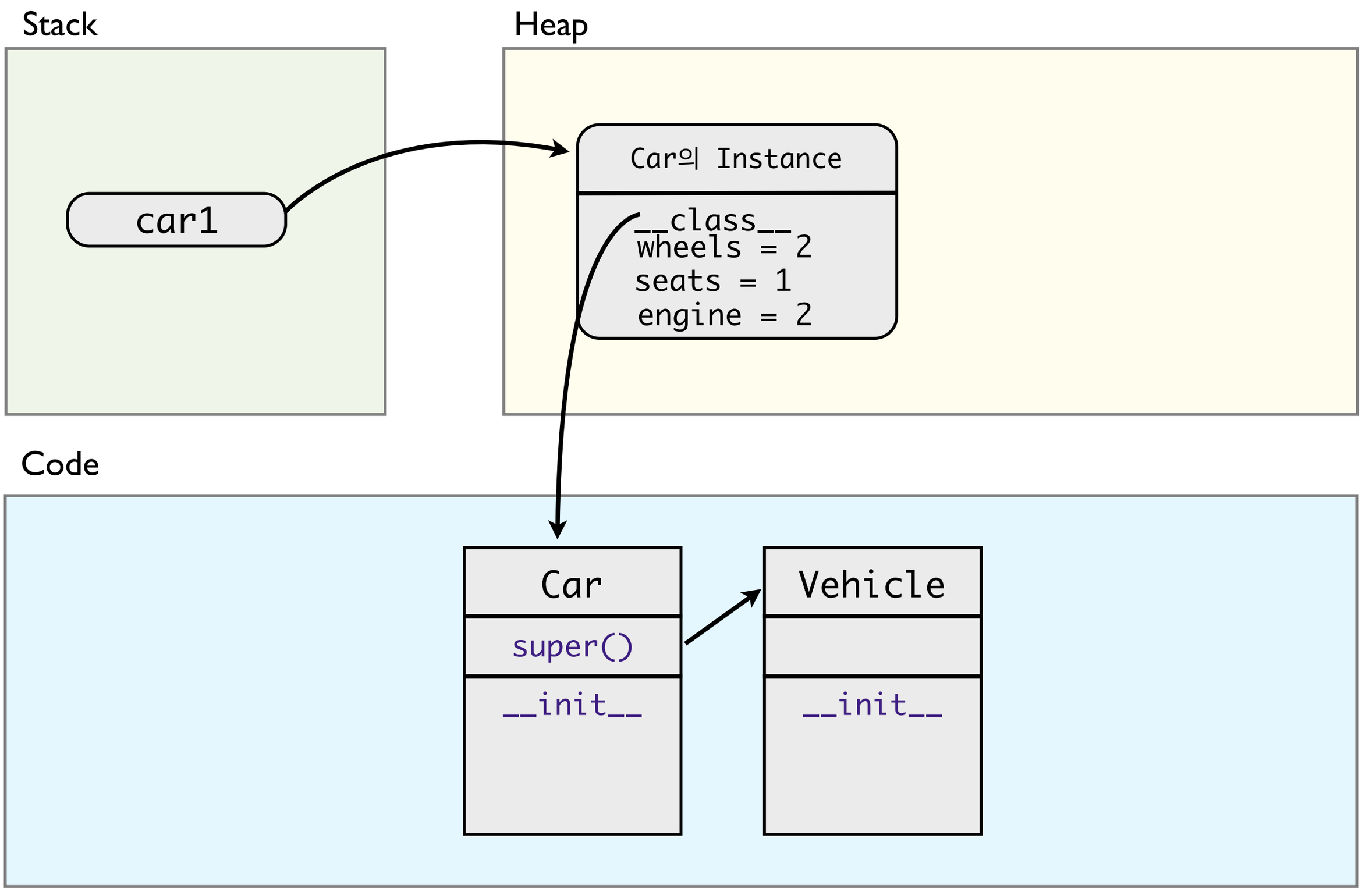


```
car1 = Car()
```

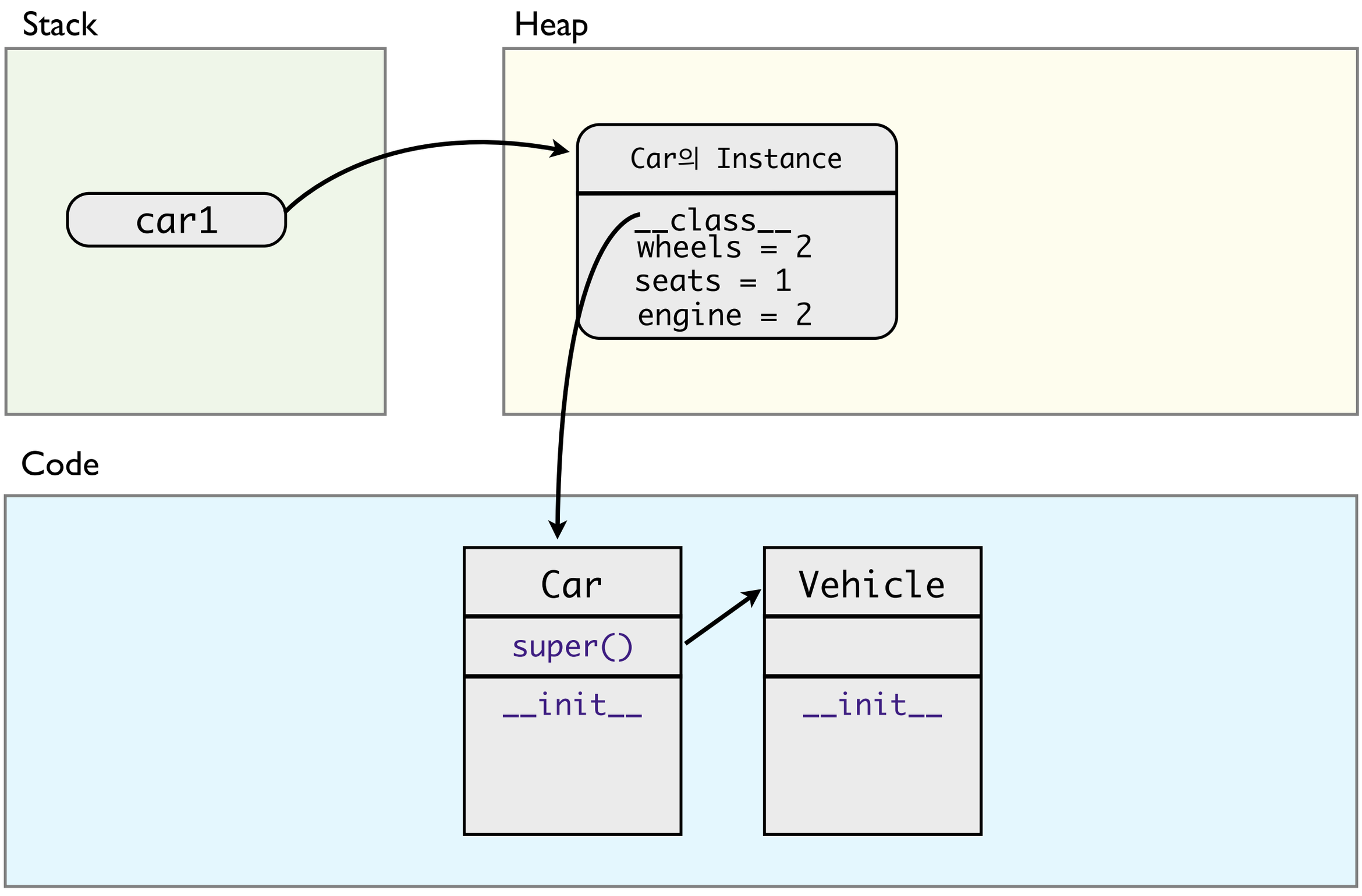

Stack
car1
Heap
Car의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Car
super()
__init__
Vehicle
__init__

```
car1 = Car()
```

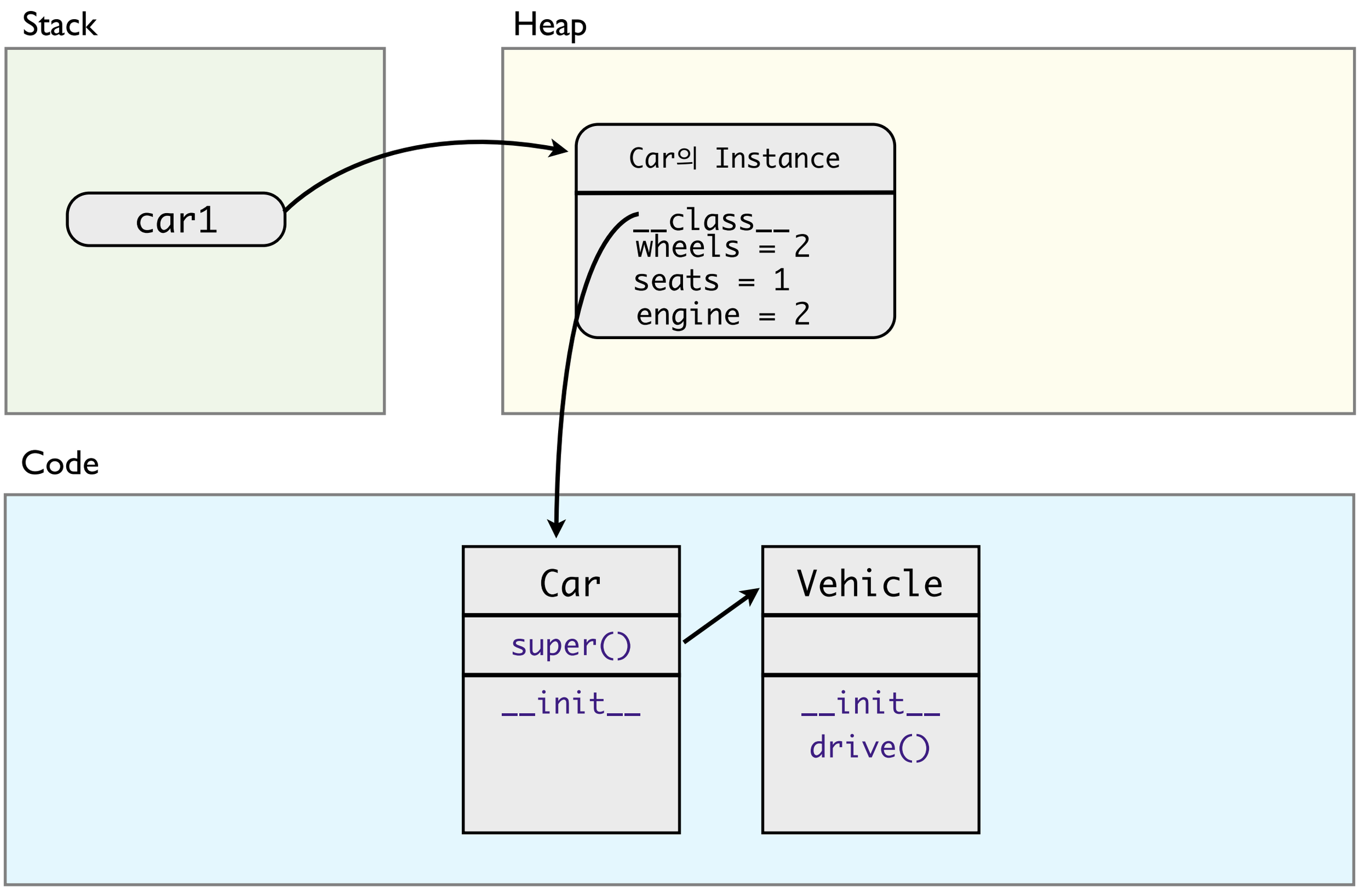

Stack
Heap
car1
Car의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Car
super()
__init__
Vehicle
__init__

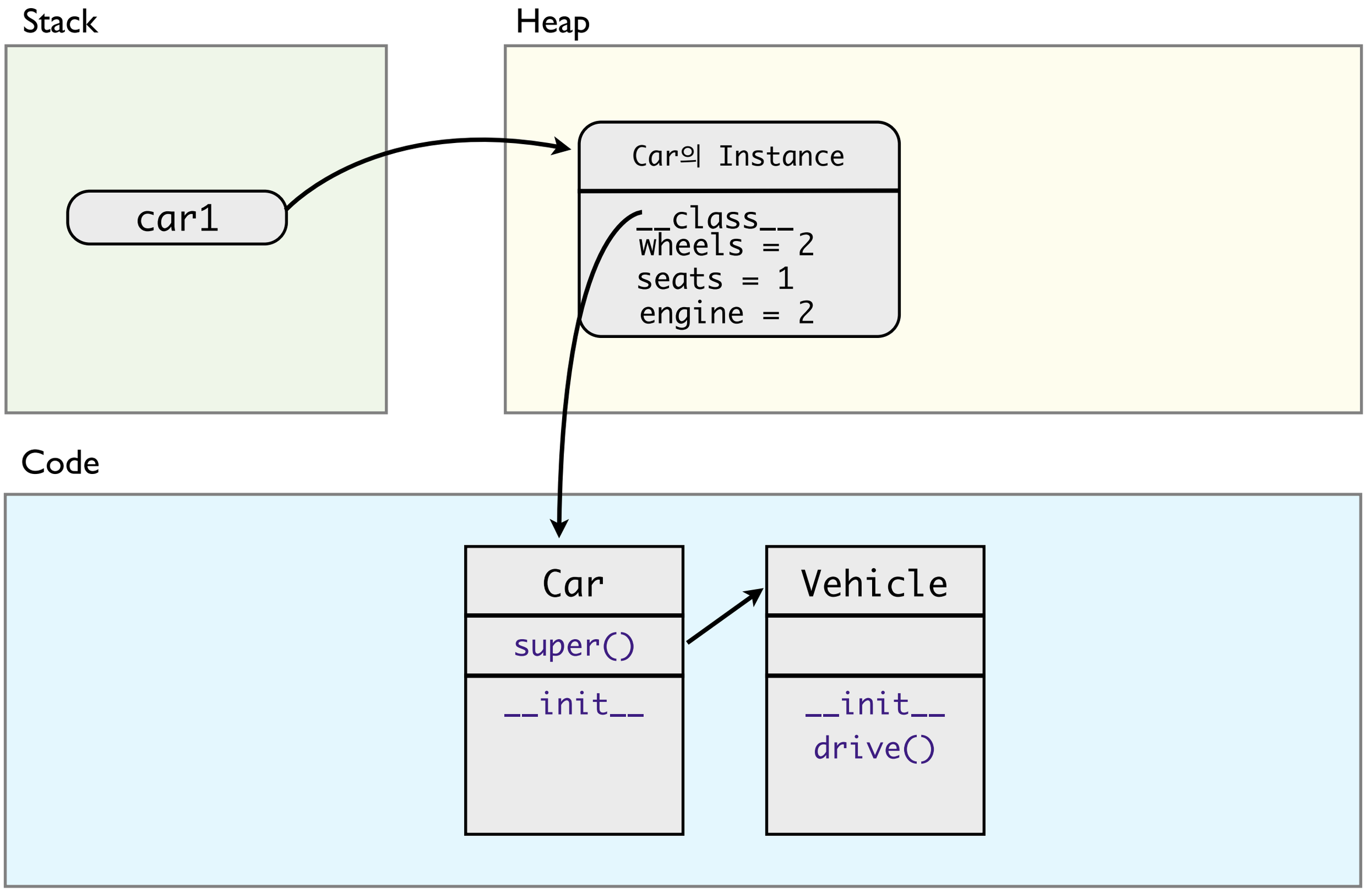

Stack
Heap
car1
Car의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Car
super()
__init__
Vehicle
__init__

```
car1.drive()
```


Stack
Heap
car1
Car의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Car
super()
__init__
Vehicle
__init__

```
car1.drive()
```


Stack
Heap
car1
Car의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Car
super()
__init__
Vehicle
__init__
drive()


Stack
Heap
car1
Car의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Code
Car
super()
__init__
Vehicle
__init__
drive()

상속

```
class Sedan(Car):
    def load_passenger(self):
        print('Load Passenger')


class Truck(Car):
    def load_cargo(self):
        print('Load Cargo')
```

```
sedan = Sedan()
sedan.load_passenger()
sedan.shift_gear()
sedan.drive()


truck = Truck()
truck.load_cargo()
truck.shift_gear()
```

Stack

Heap

Code

sedan = Sedan()

Stack

Heap

Code

sedan = Sedan()

Stack

Heap

Code

Sedan

sedan = Sedan()

Stack

Heap

Sedan의 Instance

Code

Sedan

sedan = Sedan()

Stack

Heap

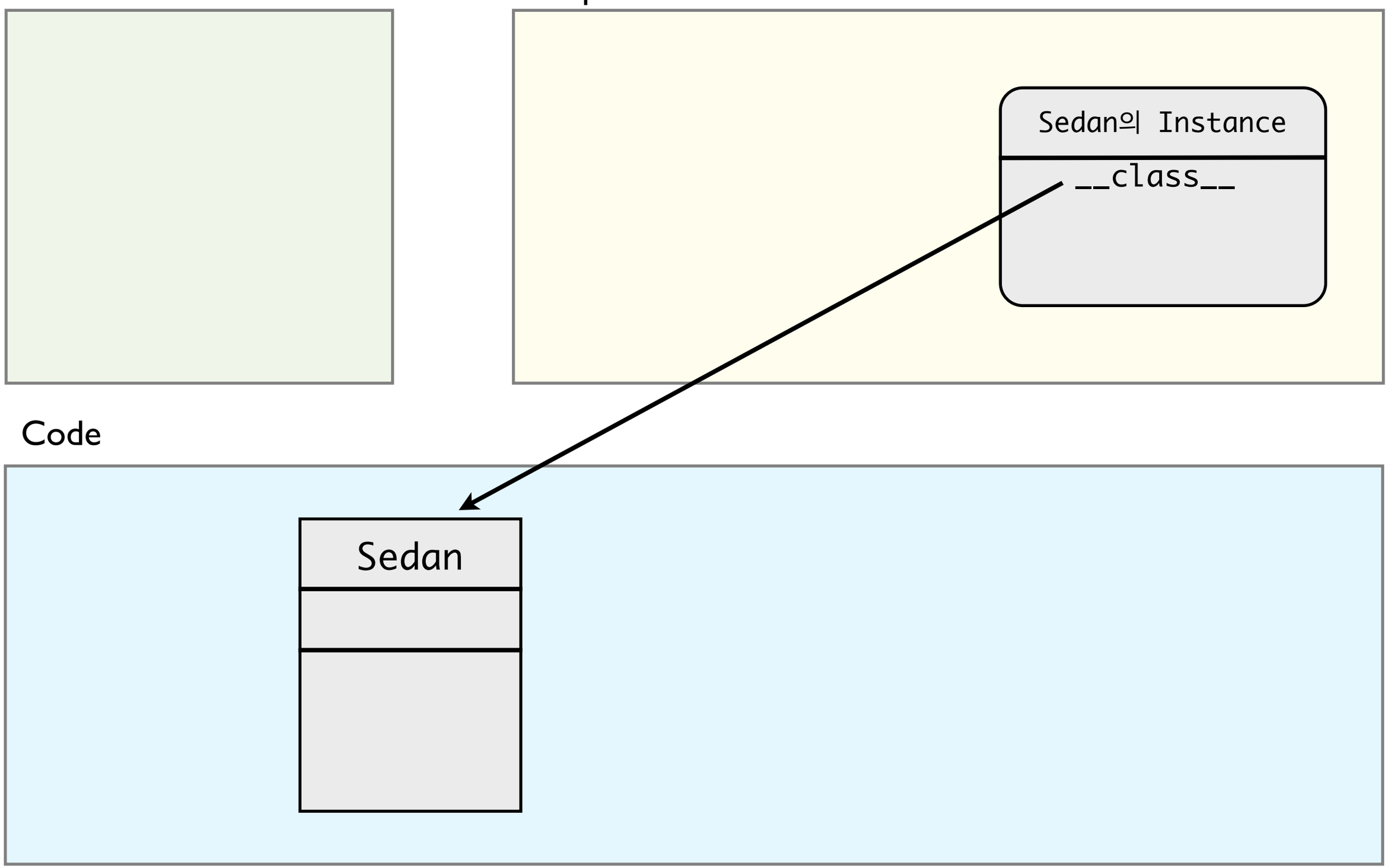
Sedan의 Instance

__class__

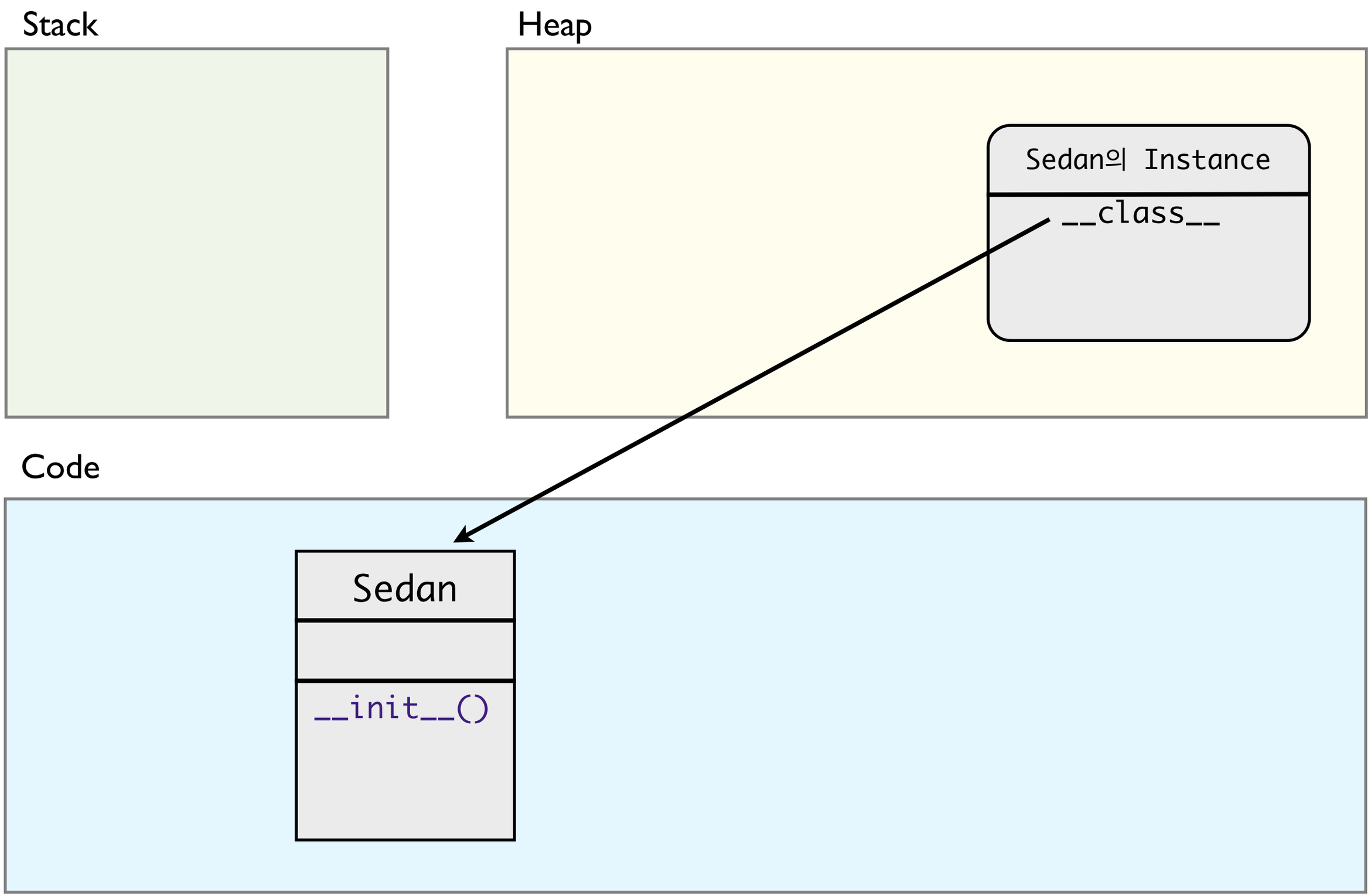
Code

Sedan

```
sedan = Sedan()
```



```
sedan = Sedan()
```



sedan = Sedan()

Stack

Heap

Sedan의 Instance

__class__

Code

Sedan

super()

__init__()

sedan = Sedan()

Stack

Heap

Sedan의 Instance

__class__

Code

Sedan

super()

__init__()

sedan = Sedan()

Stack

Heap

Sedan의 Instance

__class__

Code

Sedan

super()

__init__()

Car

```
sedan = Sedan()
```

Stack

Heap

Sedan의 Instance

__class__

Code

Sedan

super()

__init__()

Car

__init__()

sedan = Sedan()

Stack

Heap

Sedan의 Instance

__class__

Code

Sedan

super()

__init__()

Car

super()

__init__()

```
sedan = Sedan()
```

Stack

Heap

Sedan의 Instance

__class__

Code

Sedan

super()

__init__()

Car

super()

__init__()

sedan = Sedan()

Stack

Heap

Sedan의 Instance

__class__

Code

Sedan

super()

__init__()

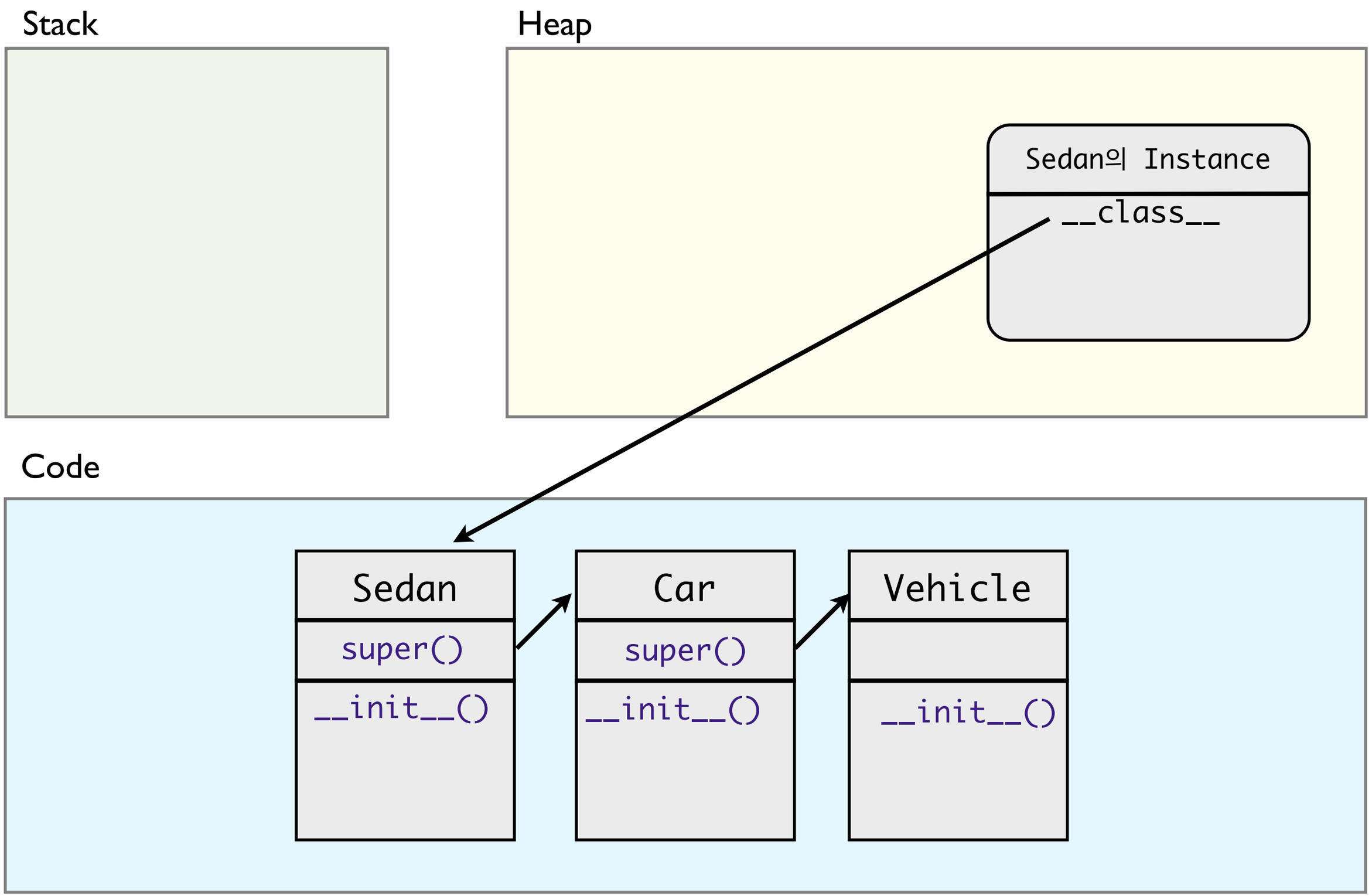
Car

super()

__init__()

Vehicle

sedan = Sedan()


Stack
Heap
Sedan의 Instance
__class__
Code
Sedan
super()
__init__()
Car
super()
__init__()
Vehicle
__init__()

sedan = Sedan()

Stack

Heap

Sedan의 Instance

__class__
wheels = 4
seats = 4
engine = 4

Code

Sedan

super()

__init__()

Car

super()

__init__()

Vehicle

__init__()

sedan = Sedan()

Stack

Heap

Sedan의 Instance

__class__
wheels = 4
seats = 4
engine = 4

Code

Sedan
super()
__init__()

Car
super()
__init__()

Vehicle
super()
__init__()

sedan = Sedan()

Stack

Heap

Sedan의 Instance

__class__
wheels = 4
seats = 4
engine = 4

Code

Sedan

super()

__init__()

Car

super()

__init__()

Vehicle

super()

__init__()

sedan = Sedan()

Stack

Heap

Sedan의 Instance

__class__
wheels = 4
seats = 4
engine = 4

Code

Sedan
super()
__init__()

Car
super()
__init__()

Vehicle
super()
__init__()

object

```
sedan = Sedan()
```

Stack

Heap

Sedan의 Instance

__class__
wheels = 4
seats = 4
engine = 4

Code

Sedan
super()
__init__()

Car
super()
__init__()

Vehicle
super()
__init__()

object
__init__()

```
sedan = Sedan()
```

Stack

sedan

Heap

Sedan의 Instance

__class__
wheels = 4
seats = 4
engine = 4

Code

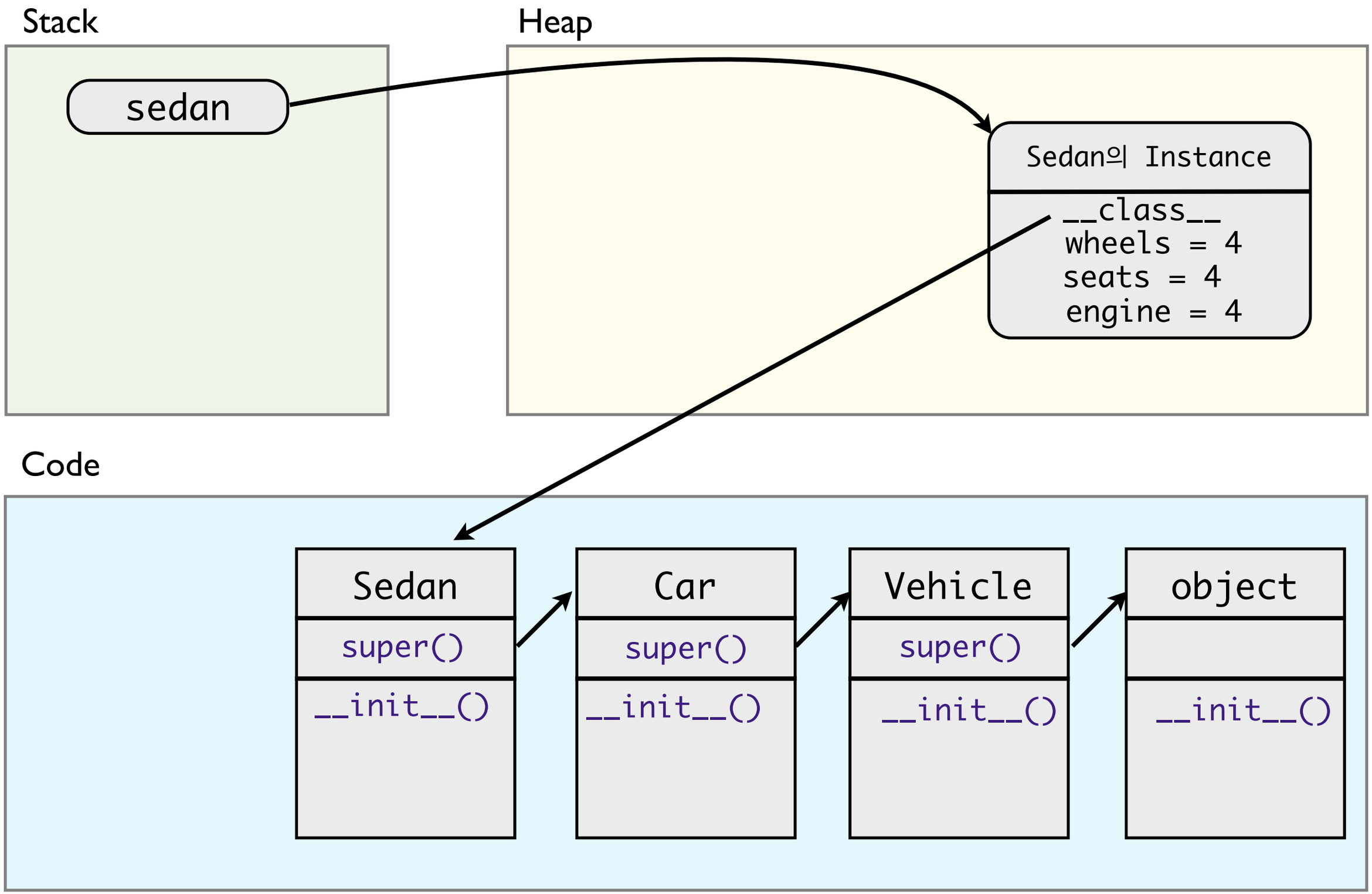
Sedan
super()
__init__()

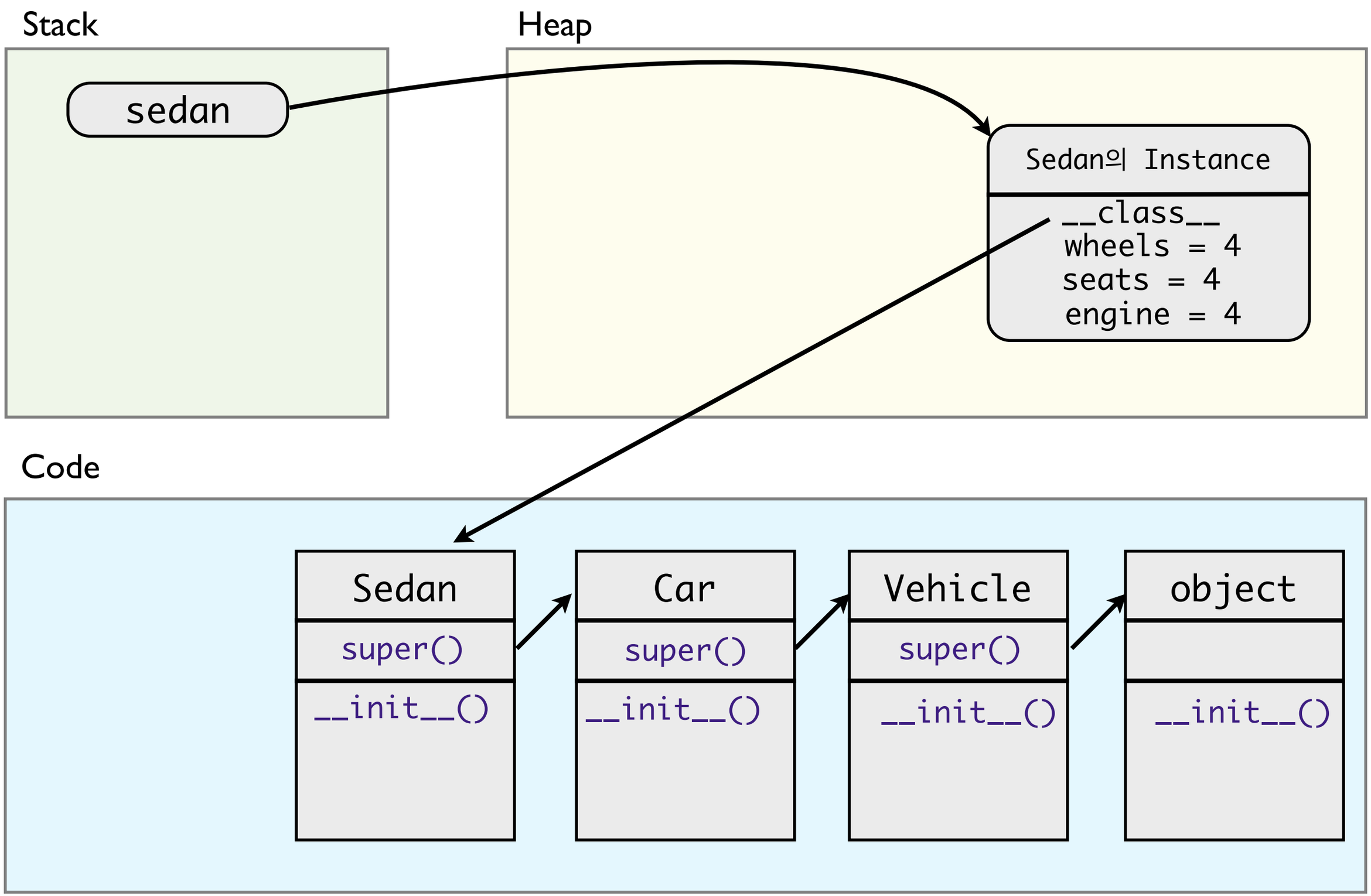
Car
super()
__init__()

Vehicle
super()
__init__()

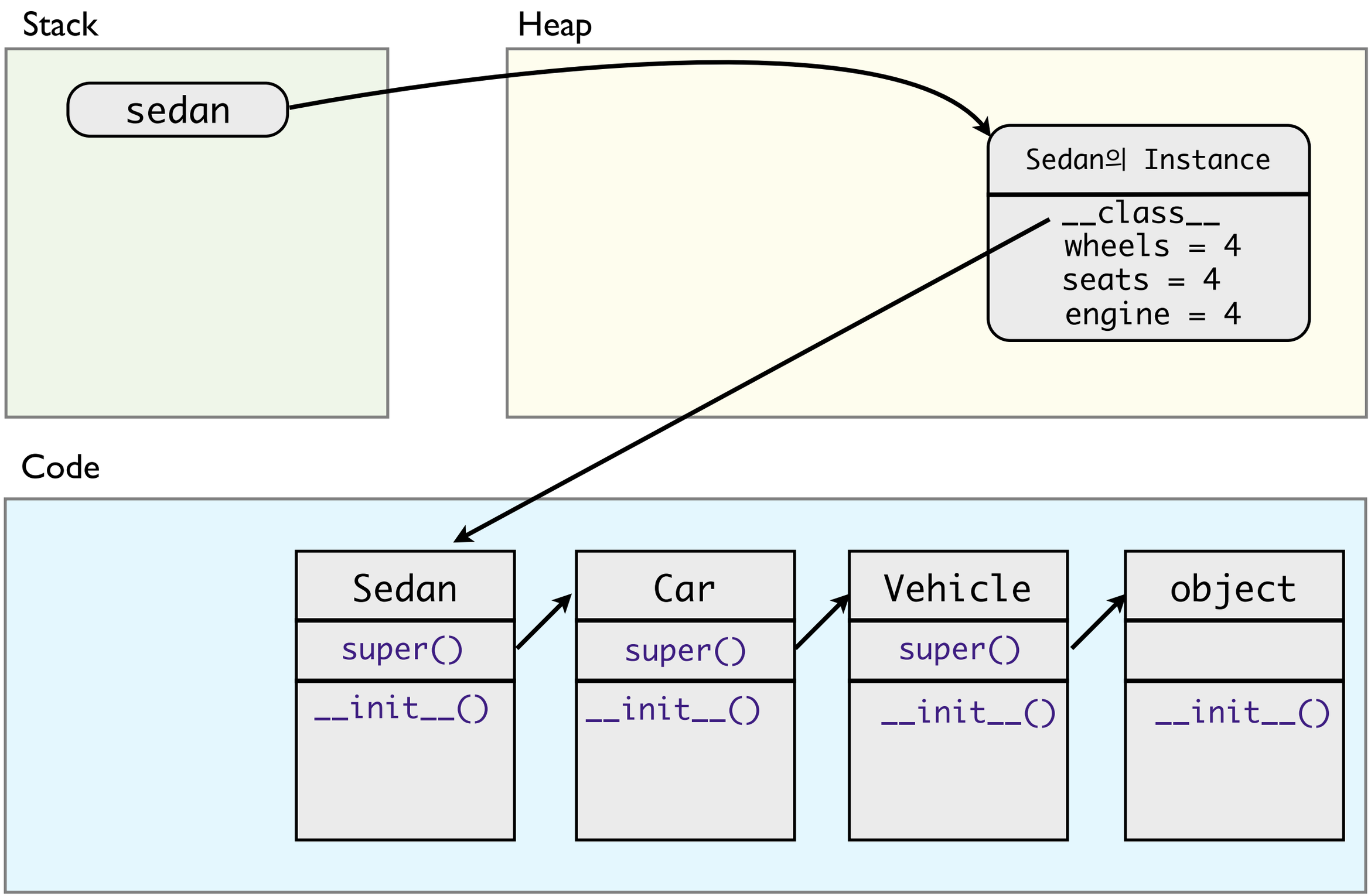
object
__init__()

sedan = Sedan()

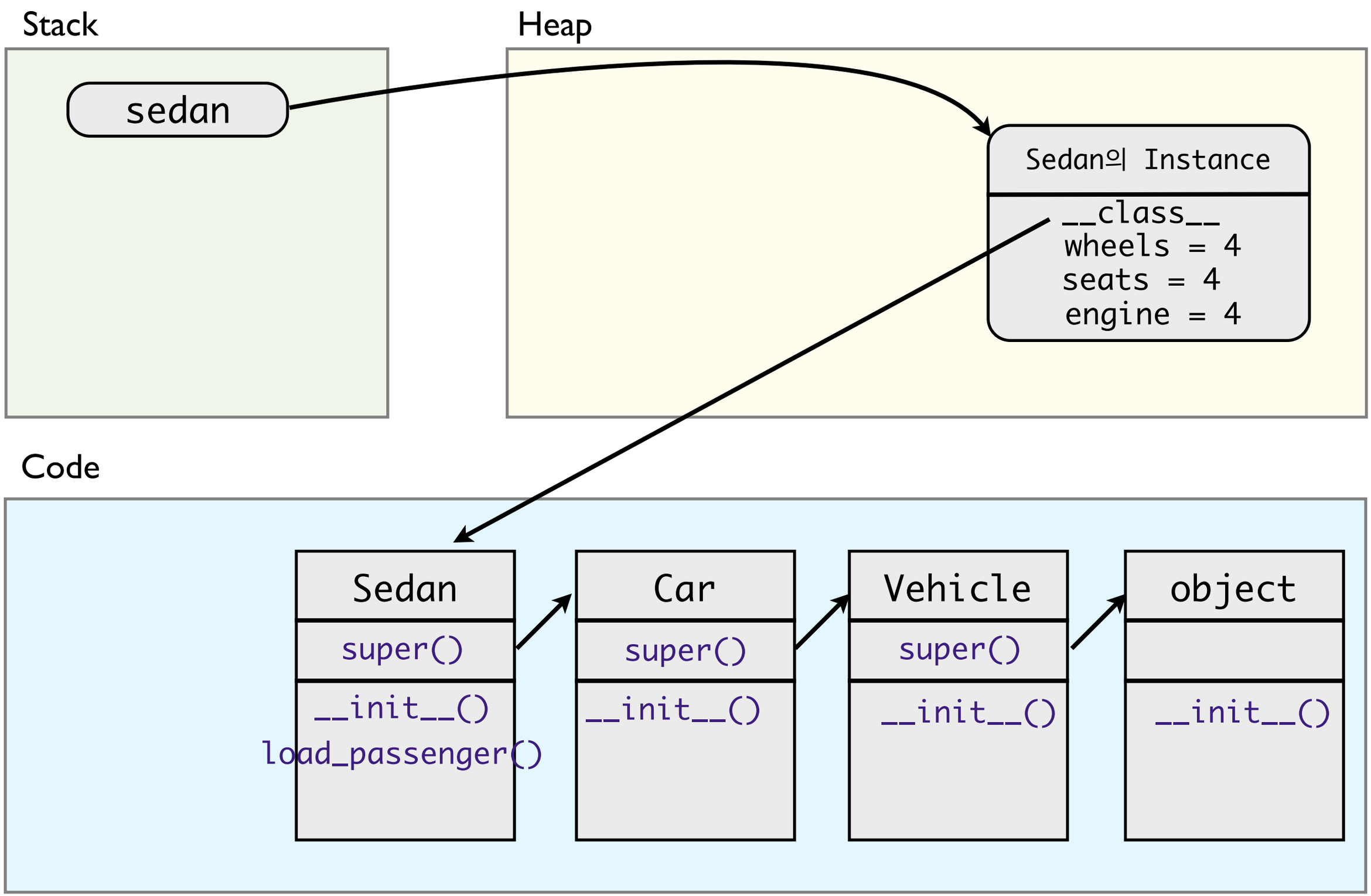

Stack
Heap
sedan
Sedan의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Sedan
super()
__init__()
Car
super()
__init__()
Vehicle
super()
__init__()
object
__init__()

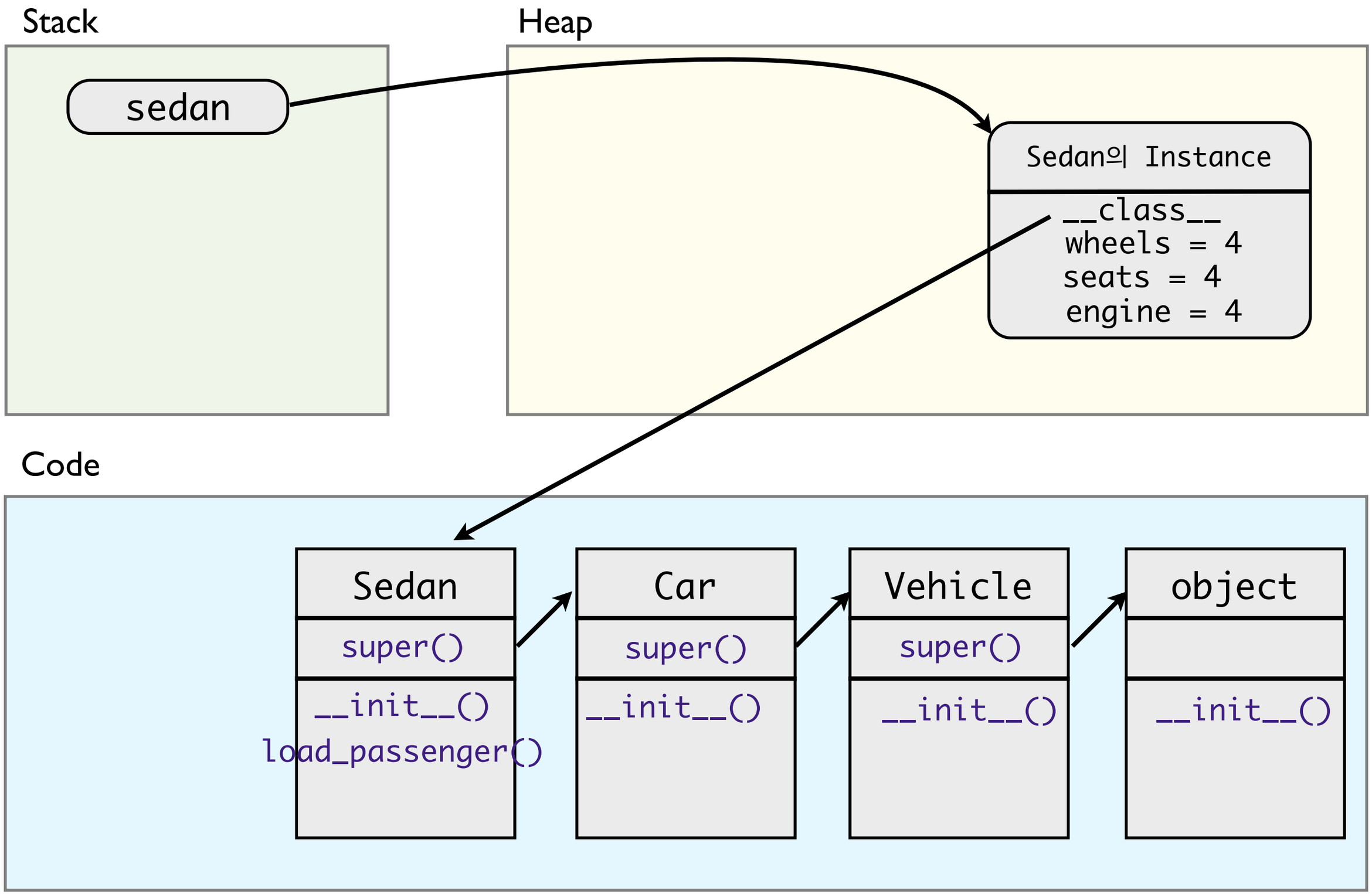

Stack
Heap
sedan
Sedan의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Sedan
super()
__init__()
Car
super()
__init__()
Vehicle
super()
__init__()
object
__init__()

```
sedan.load_passenger()
```

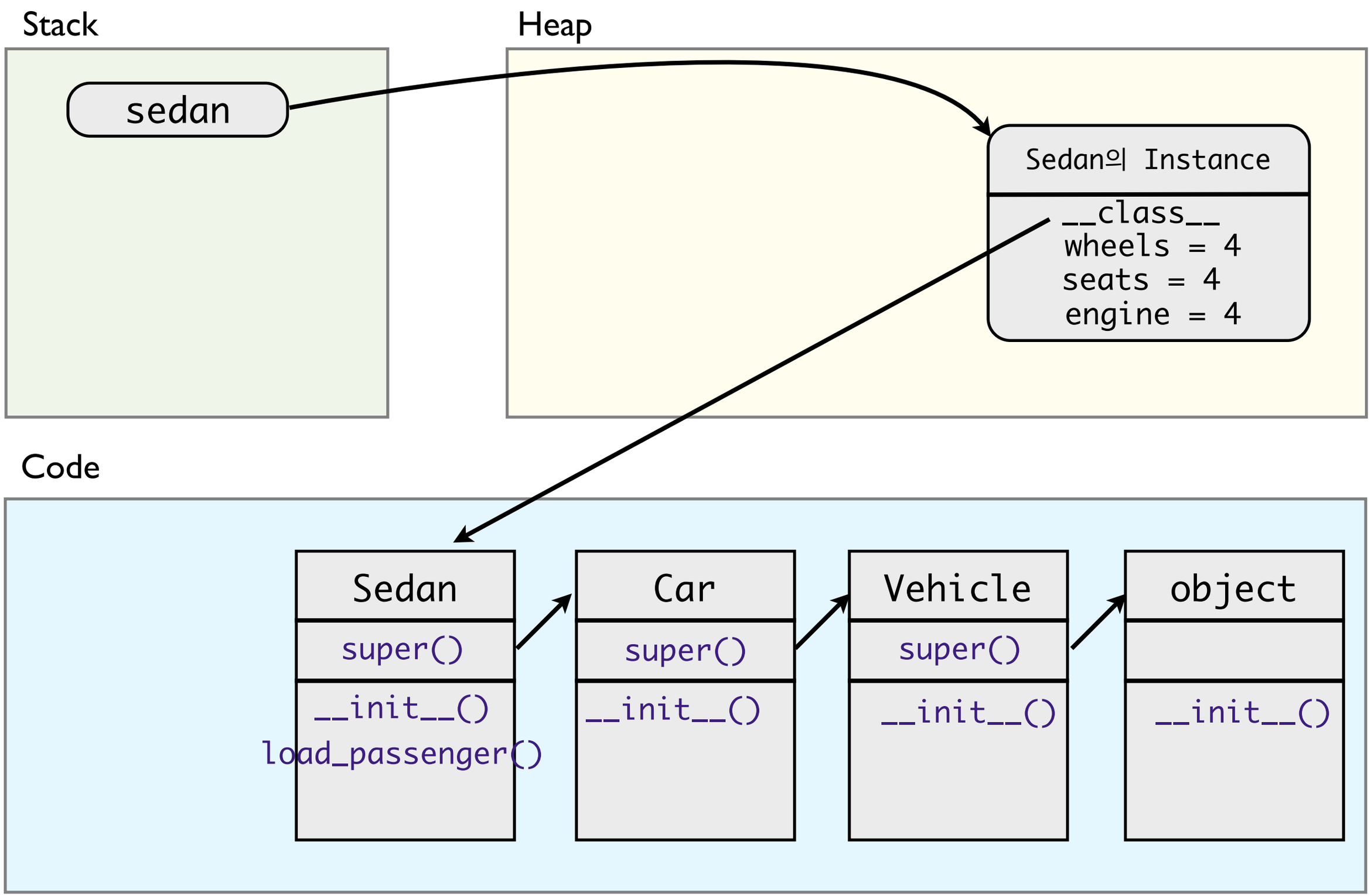


```
sedan.load_passenger()
```

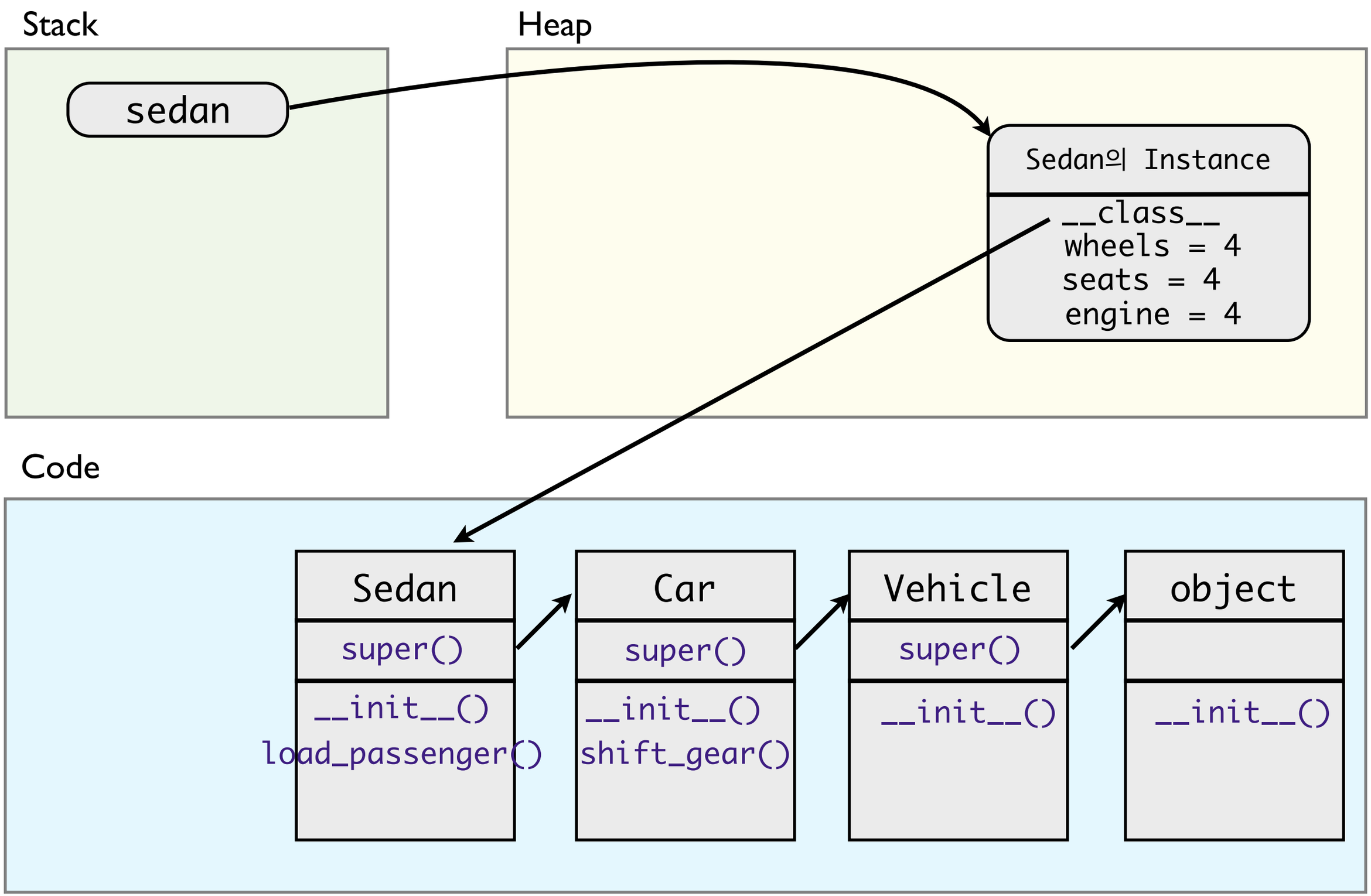


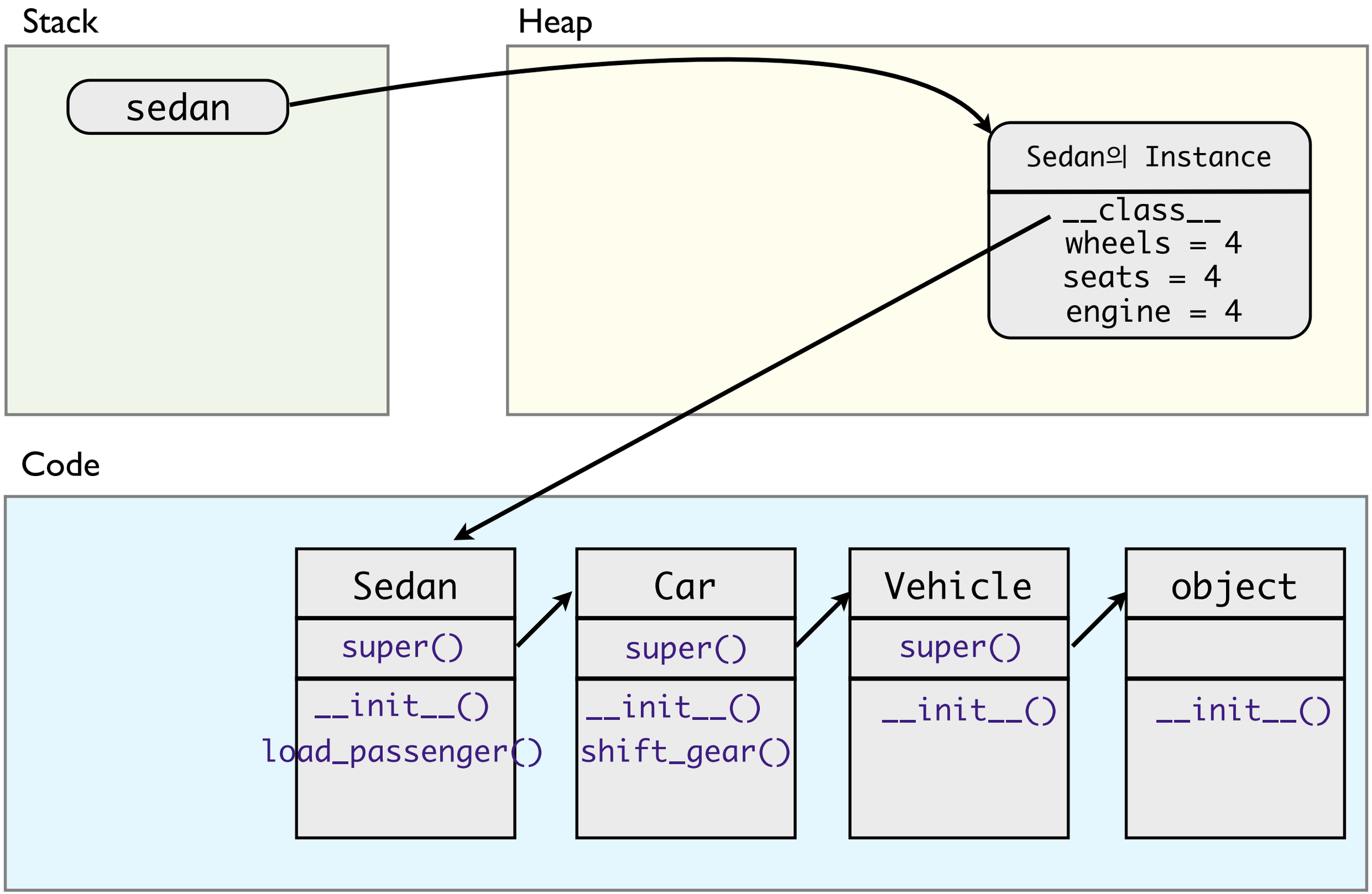

Stack
Heap
sedan
Sedan의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Sedan
super()
__init__()
load_passenger()
Car
super()
__init__()
Vehicle
super()
__init__()
object
__init__()

```
sedan.shift_gear()
```

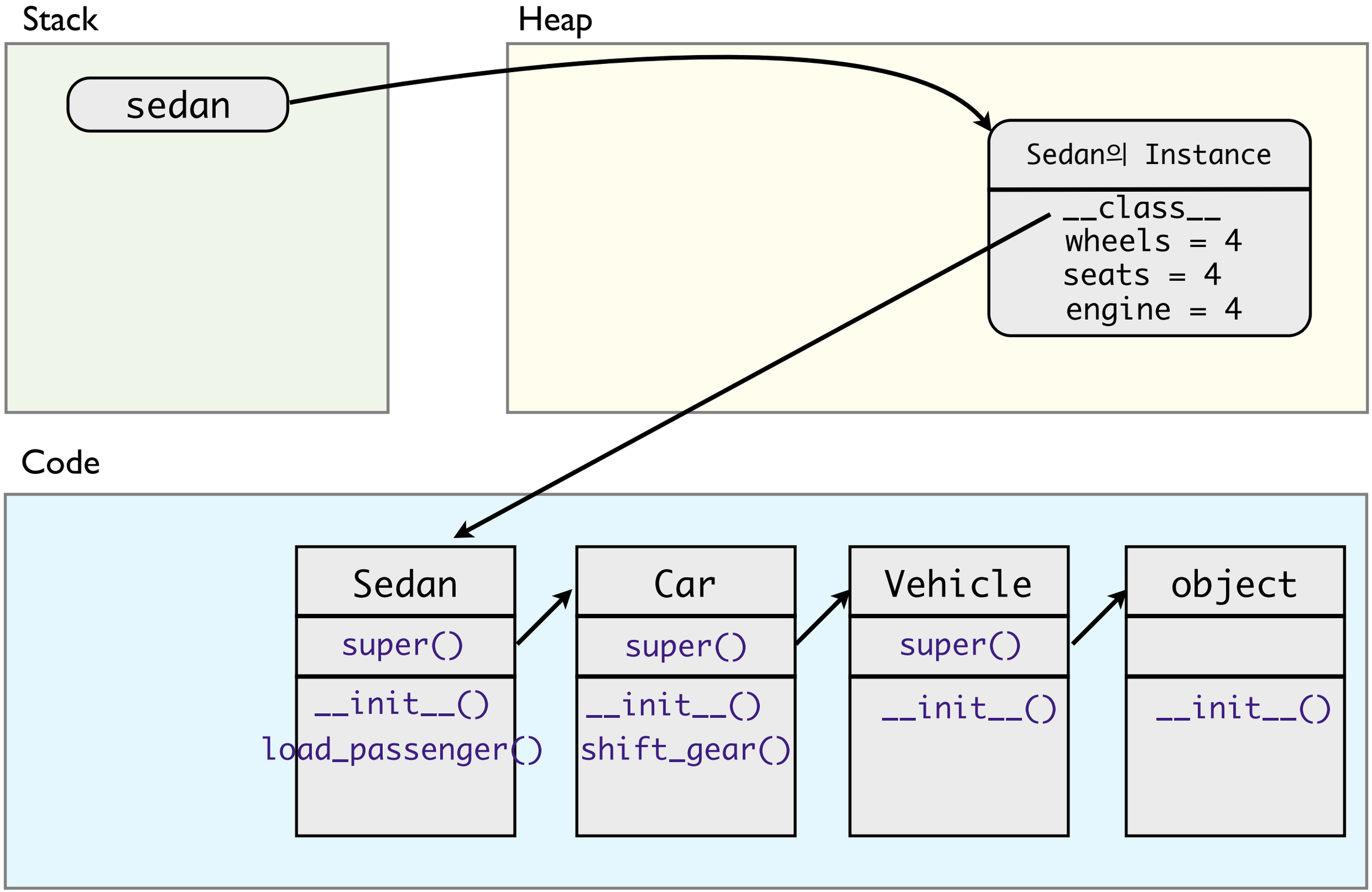


```
sedan.shift_gear()
```

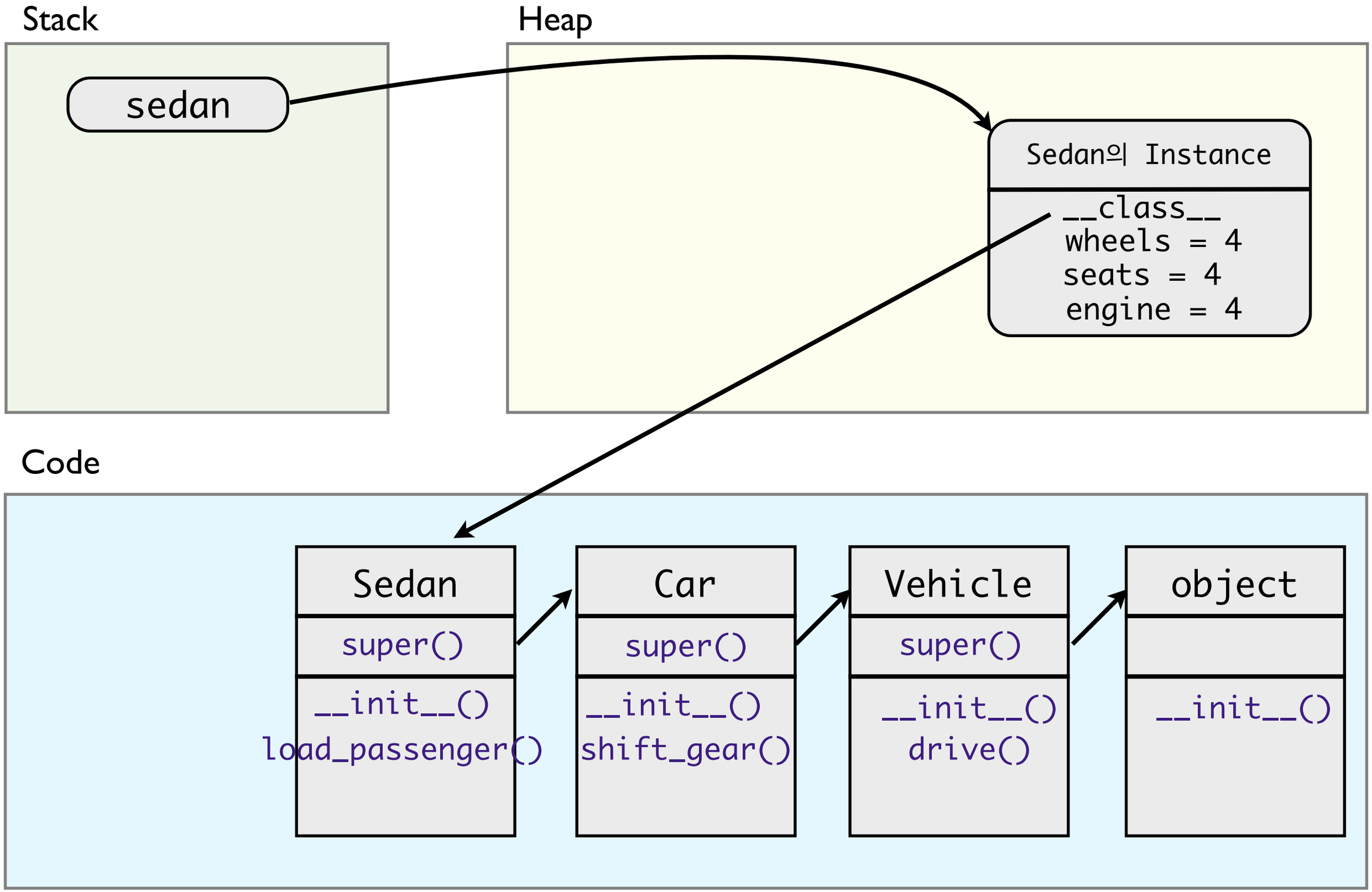


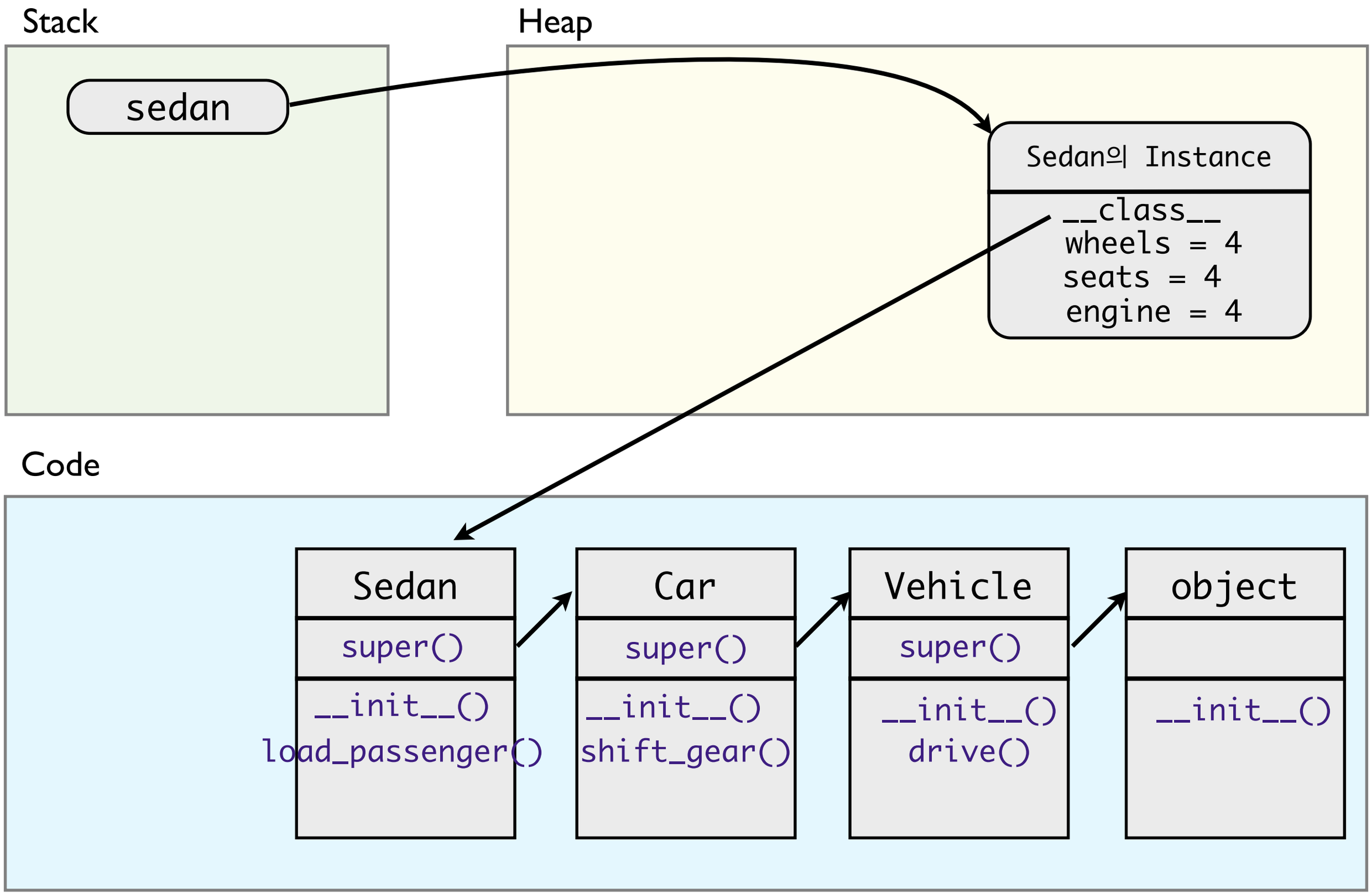

Stack
Heap
sedan
Sedan의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Sedan
super()
__init__()
load_passenger()
Car
super()
__init__()
shift_gear()
Vehicle
super()
__init__()
object
__init__()

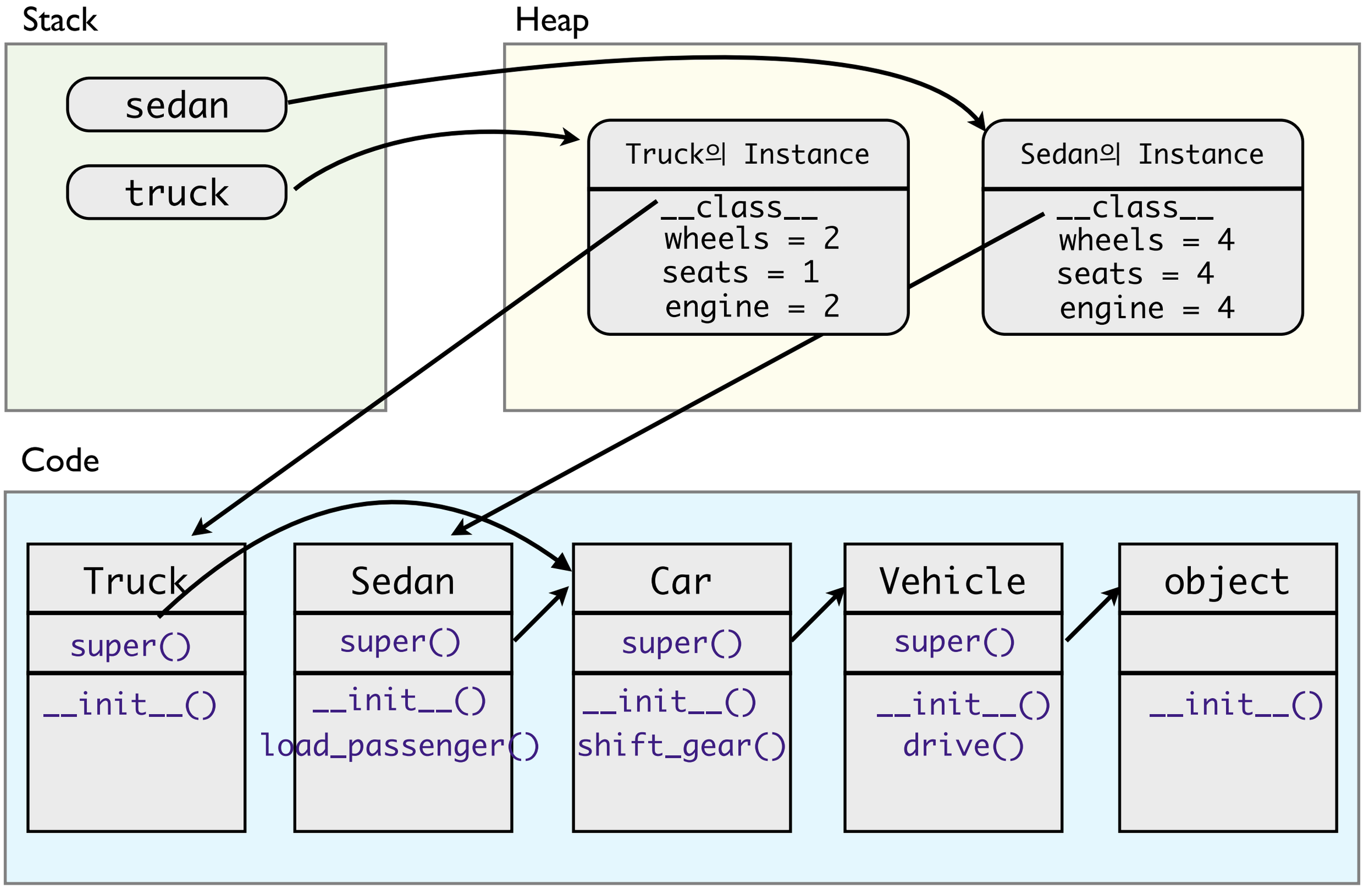
```
sedan.drive()
```


Stack
Heap
sedan
Sedan의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Sedan
super()
__init__()
load_passenger()
Car
super()
__init__()
shift_gear()
Vehicle
super()
__init__()
object
__init__()

```
sedan.drive()
```


Stack
Heap
sedan
Sedan의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Sedan
super()
__init__()
load_passenger()
Car
super()
__init__()
shift_gear()
Vehicle
super()
__init__()
drive()
object
__init__()


Stack
Heap
sedan
Sedan의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Sedan
super()
__init__()
load_passenger()
Car
super()
__init__()
shift_gear()
Vehicle
super()
__init__()
drive()
object
__init__()


Stack
Heap
sedan
truck
Truck의 Instance
__class__
wheels = 2
seats = 1
engine = 2
Sedan의 Instance
__class__
wheels = 4
seats = 4
engine = 4
Code
Truck
super()
__init__()
Sedan
super()
__init__()
load_passenger()
Car
super()
__init__()
shift_gear()
Vehicle
super()
__init__()
drive()
object
__init__()

다형성

```
class Vehicle():
    def drive(self):
        print('Run')

class Sedan(Vehicle):
    def drive(self):
        print('Run Sedan')

class Truck(Vehicle):
    def drive(self):
        print('Run Truck')

class Boat():
    def drive(self):
        print('Fly')
```

```
def drive(vehicle):
    if isinstance(vehicle, Vehicle):
        vehicle.drive()

sedan = Sedan()
drive(sedan)

truck = Truck()
drive(truck)

boat = Boat()
drive(boat)
```

람다(lambda)

일반 함수와 람다 함수

- 일반 함수를 이용하면 필요한 기능의 재사용이 가능하다
  - 함수를 정의한 후 필요할 때마다 호출하여 사용
- 가끔씩은 함수를 만들지 않고 함수화된 기능만을 불러 사용하고자 할 경우가 있다
  - 프로그램을 단순하게 만들 수 있다
  - 1회용으로 함수를 만들고 싶은데 def xxx(): 과 같은 이름을 짓는 것이 번거롭게 느껴질 수 있다

람다 함수

- 람다, 람다 표현식라고도 불리우는 람다 함수는 이름이 없는 함수
  - 익명함수(anonymous function)로 부르기도 함
  - 이름이 필요할 경우 할당하여 사용할 수 있다
- 주의 : 표현식 안에서 새 변수를 선언할 수 없다
- 반환값은 변수없이 식 한줄로 표현할 수 있어야 하며 복잡한 함수가 필요하면 def 로 함수를 정의하여야 한다.
  - return 문이 필요없다

람다함수 정의

```
def add(x, y):
    return x + y


f = lambda x, y: x + y


print(add(1, 2))
print(f(1,2))
```

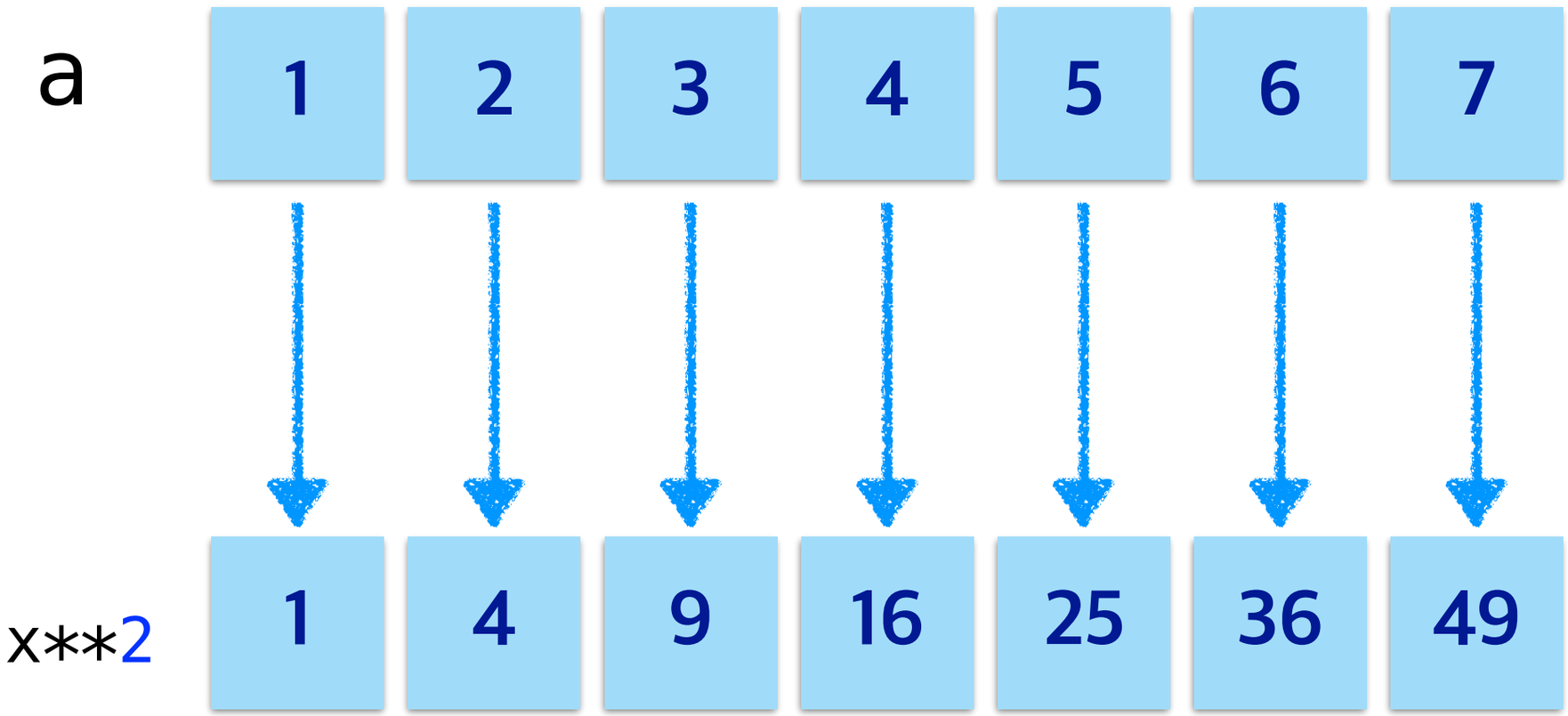
람다함수 정의

```
def add(x, y):
    return x + y

f = lambda x, y: x + y

print(add(1, 2))
print(f(1,2))

f = lambda x: x if x % 3 == 0 else 0
print(f(3))
print(f(4))
```

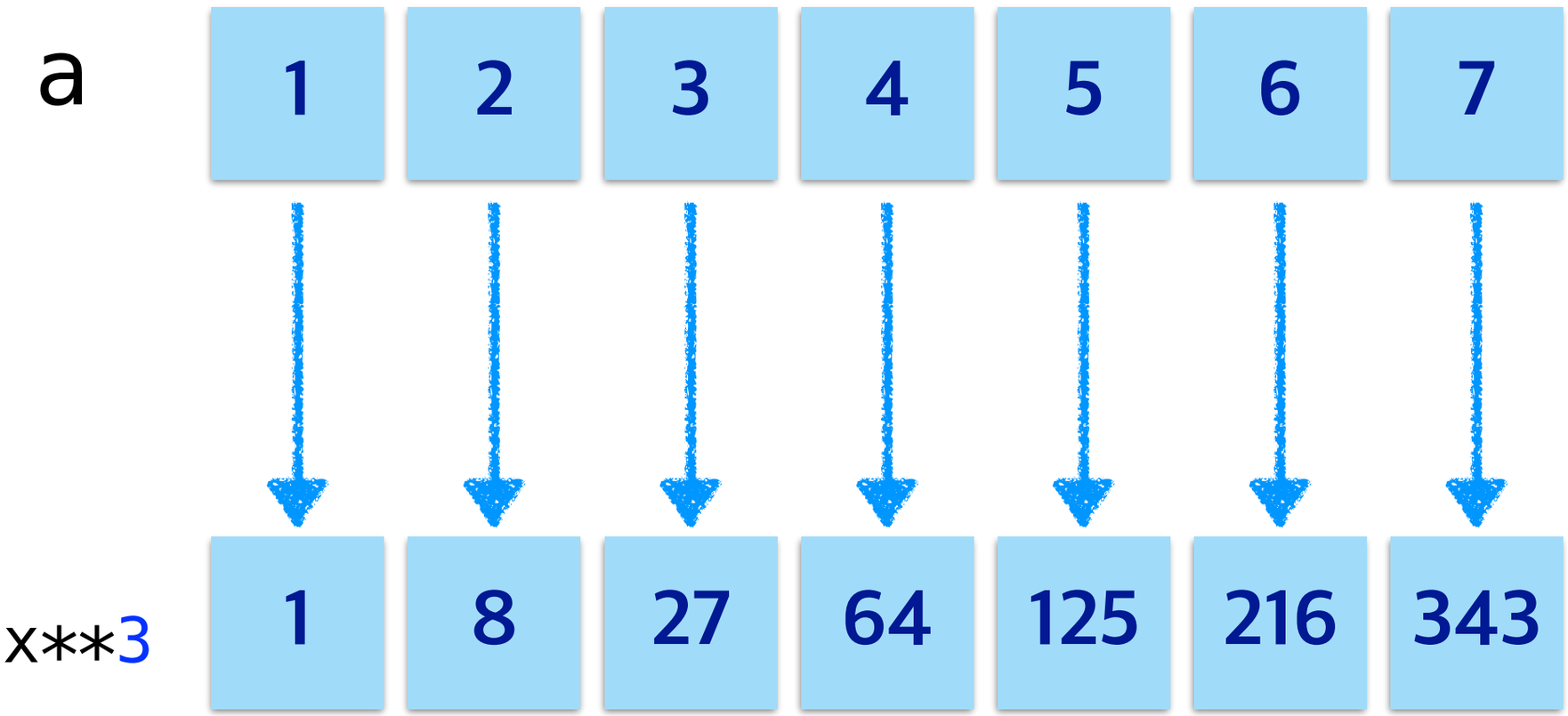
맵(map) 함수와 람다

- 파이썬은 map() 이라는 내장함수를 제공하는데 열거가능한 자료형의 각 요소들에 대해서 매핑 함수를 적용한다

```
a = [1, 2, 3, 4, 5, 6, 7]
map(lambda x: x**2, a)
```



```
a = [1, 2, 3, 4, 5, 6, 7]
map(lambda x: x**3, a)
```



맵(map) 함수 사용

```
nums = [1, 2, 3, 4, 5]
def double(x):
    return x ** 2
f = lambda x: x**2
print(list(map(double, nums)))
print(list(map(f, nums)))
print(list(map(lambda x: x**2, nums)))
```

맵(map) 함수 사용

```
nums1 = [1, 2, 3, 4, 5]
nums2 = [6, 7, 8, 9, 10]

print(list(map(lambda x, y: x*y, nums1, num2)))
```

필터(filter) 함수와 람다

- 리스트에서 다룬 filter() 함수는 순환가능한 요소들을 함수에 넣어 그 리턴값이 참인 것만 묶어서 반환한다.

필터(filter) 함수 사용

```
ages = [18, 19, 39, 12, 43, 13, 21, 25]

def adult_func(age):
    if age >= 19:
        return True
    else:
        return False

for age in filter(adult_func, ages):
    print(age)
print()
for age in filter(lambda age: age >= 19, ages):
    print(age)
```

필터(filter) 함수 사용

```
ages = [18, 19, 39, 12, 43, 13, 21, 25]

adult_ages = list(filter(lambda  age: age >= 19, ages))

print(adult_ages)
```

실습

1부터 10 사이의 정수를 가진 리스트에서 짝수만 나오게 필터링한 결과 리스트의 모든 원소에 대해 제곱을 수행해서 리스트로 반환하는 코드를 필터와 맵과 람다를 이용해서 작성하시요.

실습

```
nums = list(range(1, 10))

result = list(map(lambda x: x**2,
                  filter(lambda  num: num %2 == 0, nums)))
print(result)
```

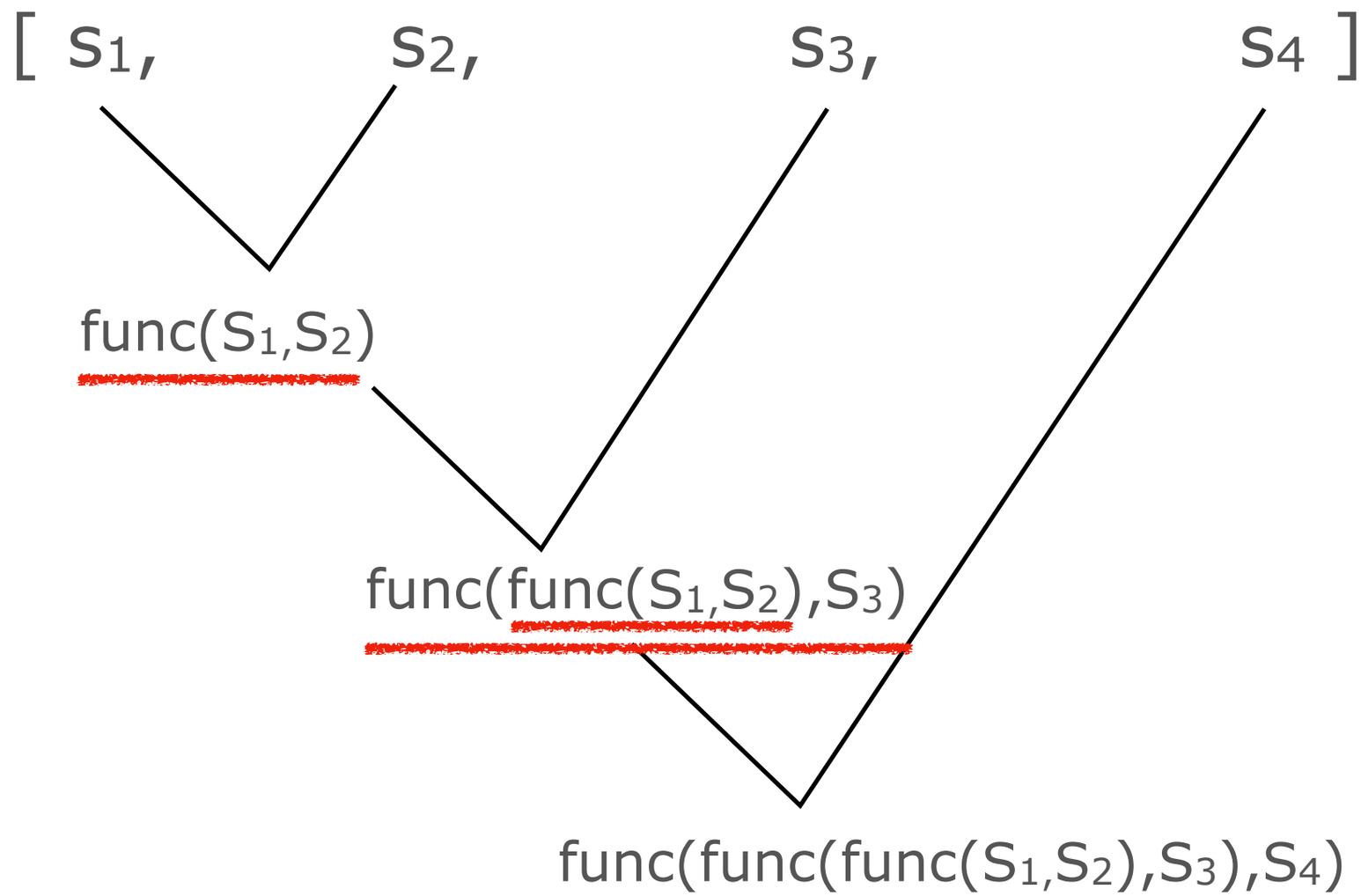
```
[4, 16, 36, 64]
```

reduce()

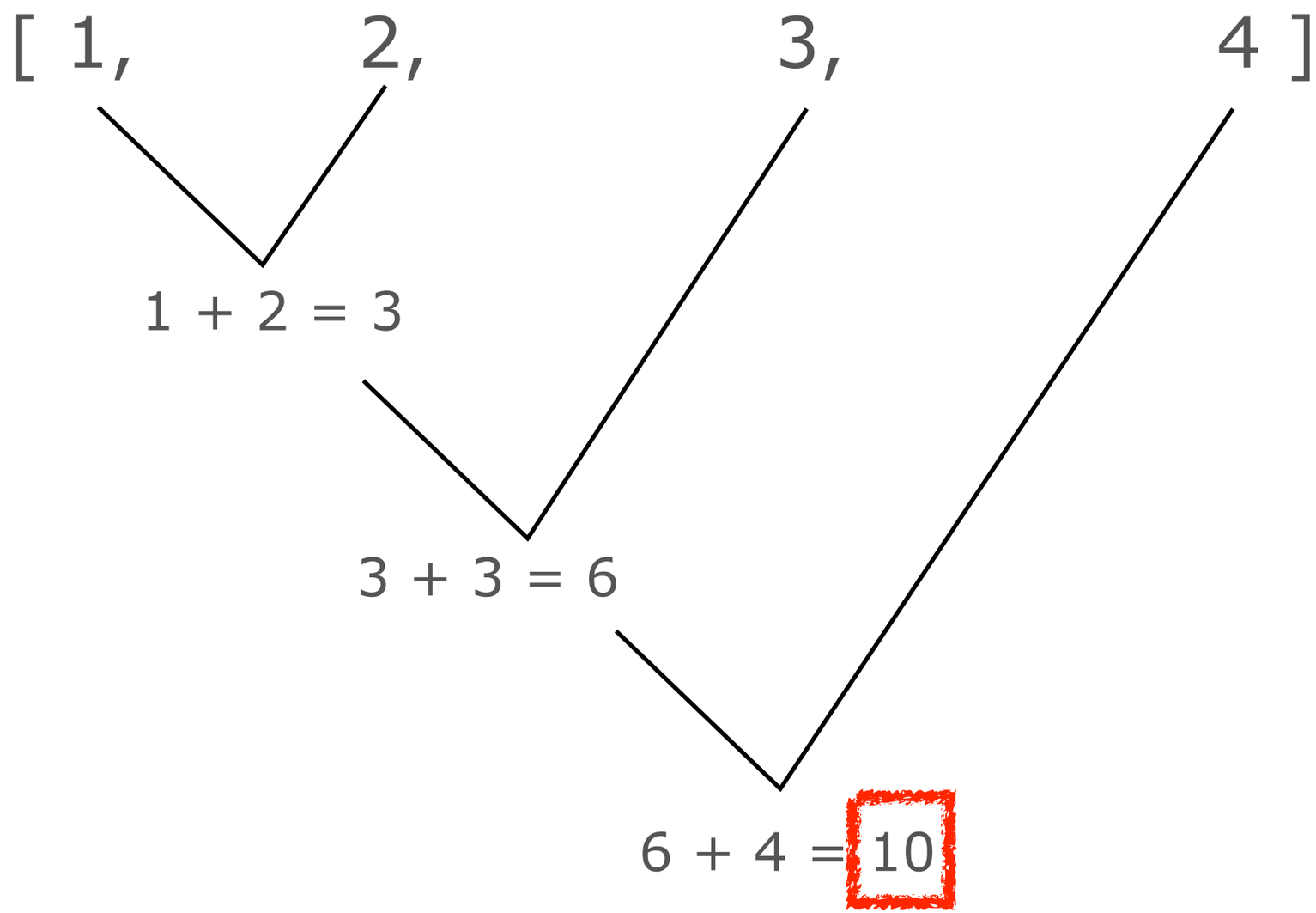
- 리듀스 함수는 functools 모듈에 포함되어 있는데 순환 가능 요소들을 같은 방법으로 연산하여 하나의 값을 얻는데 사용할 수 있다

- 문제는 이 결론을 얻는 것이 쉽게 이해하기 어렵다는 점이다

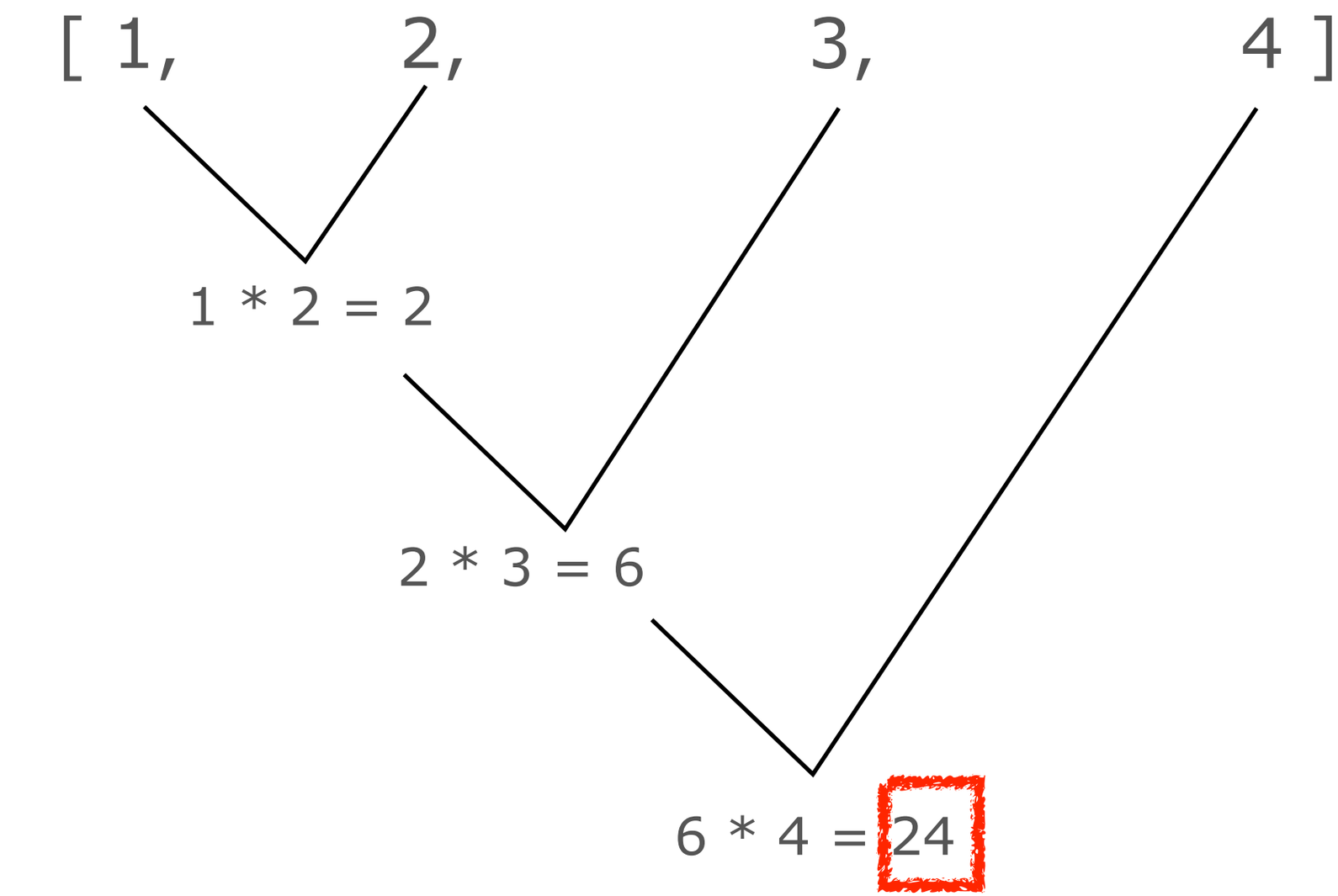
seq = [s_1, s_2, s_3, s_4] 라는 리스트가 있고 reduce(func, seq)가 호출되면 다음과 같은 작업이 이루어진다



```
a = [1, 2, 3, 4]
n = reduce(lambda x, y: x + y, a)
```



```
a = [1, 2, 3, 4]
n = reduce(lambda x, y: x * y, a)
```



reduce 함수 사용

```
from functools import reduce

nums = [1, 2, 3, 4]

sum = reduce(lambda x, y: x+y, nums)

print(sum)


mul = reduce(lambda x, y: x*y, nums)

print(mul)
```

실습

1부터 10 사이의 정수를 가진 리스트에서 짝수만 나오게 필터링한 결과 리스트의 모든 원소의 곱을 구하는 코드를 lambda, filter, reduce 함수를 사용해서 작성하시요.

실습

```
from functools import reduce



nums = list(range(1, 10))

result = reduce(lambda x, y: x*y,
                    filter(lambda  num: num %2 == 0, nums))
print(result)
```

```
 384
```