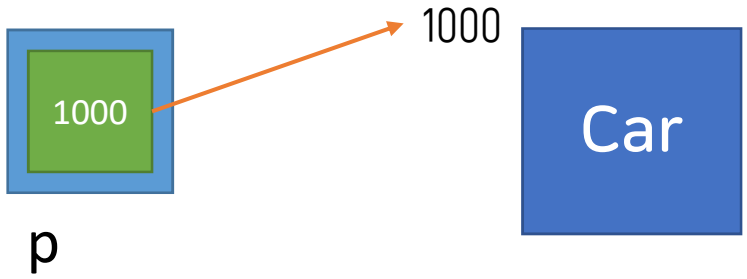
"포인터와 유사하게 동작하는 추상화된 타입 으로 포인터의 기능 외에 자동화된 자원관리 등의 기능을 추가로 제공한다."

- Raw Pointer — 생성자, 소멸자 등을 가질 수 없다.
- Smart Point — 생성자, 소멸자 등을 가진다.
  생성/복사/대입/소멸의 과정에 추가적인 기능을 수행 할 수 있다. 일반적으로 소멸자에서 자동 화된 자원관리를 수행한다.

스마트 포인터의 원리



- 내부적으로 객체의 주소를 보관하기 위해 포인터 멤버를 가지고 있다.
- -> 와 * 연산자를 재정의 되어 있기 때문에 포인터 처럼 사용할 수 있다.
- 생성/복사/대입/소멸의 과정을 제어 할 수 있다. 소멸자에서 자원을 삭제 한다.

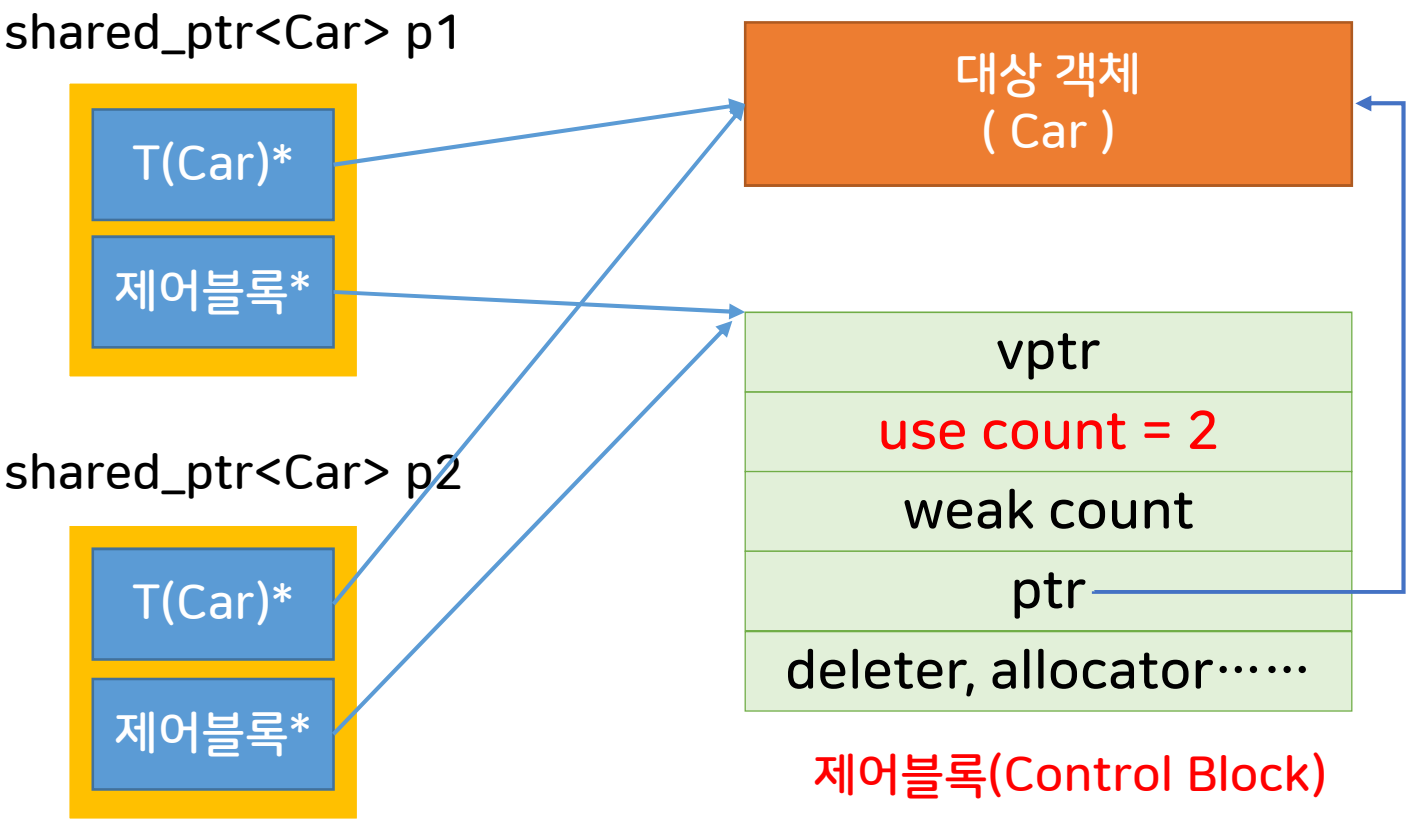
C++ 표준 스마트 포인터

- shared_ptr<>, weak_ptr<>, unique_ptr<>

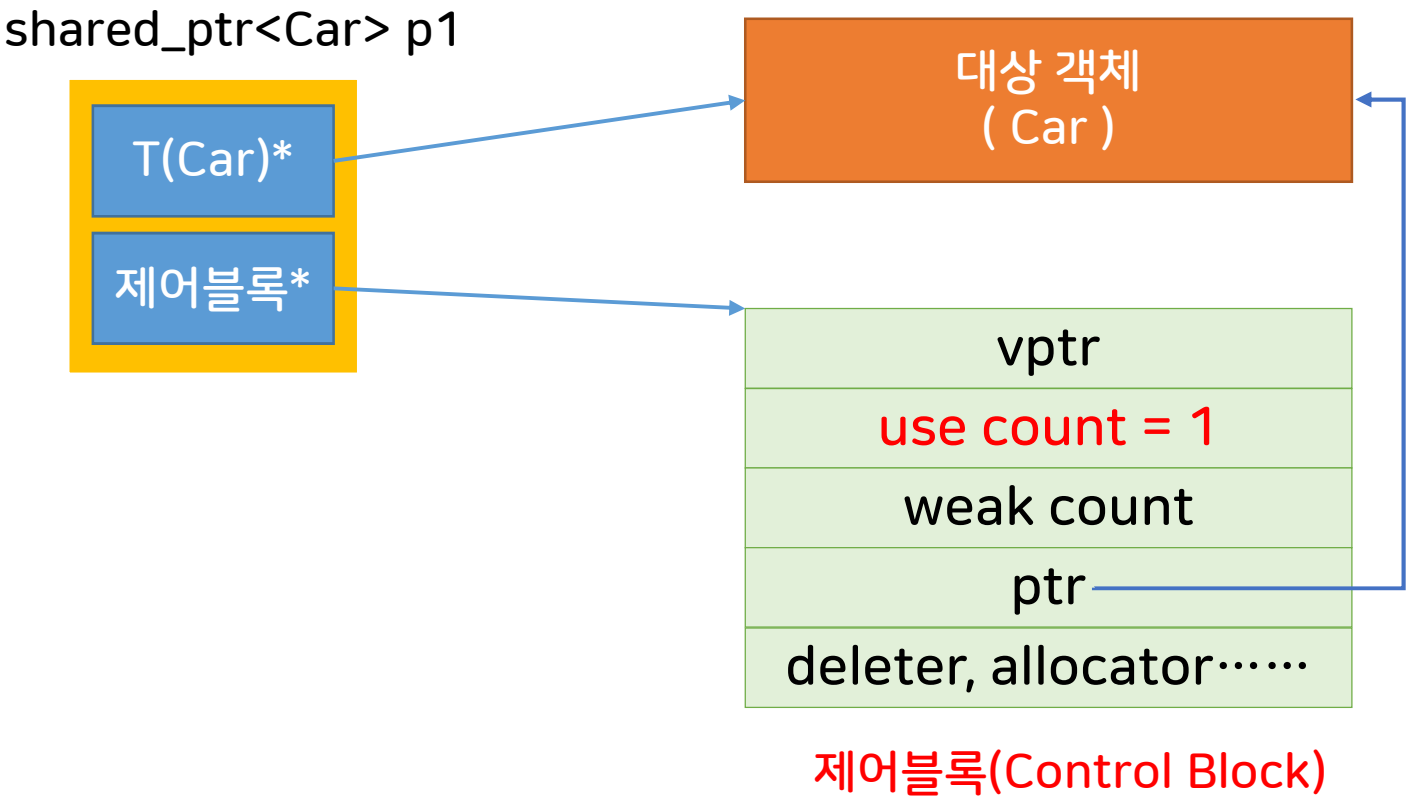
shared_ptr 기본

- <memory> 헤더
- copy 초기화 될 수 없고, direct 초기화 만 가능하다. – explicit 생성자

shared_ptr 의 메모리 구조



- shared_ptr 생성시, 참조 계수 등을 관리하는 제어 블럭 이 생성된다.
- 자원을 공유(share)하는 스마트 포인터.

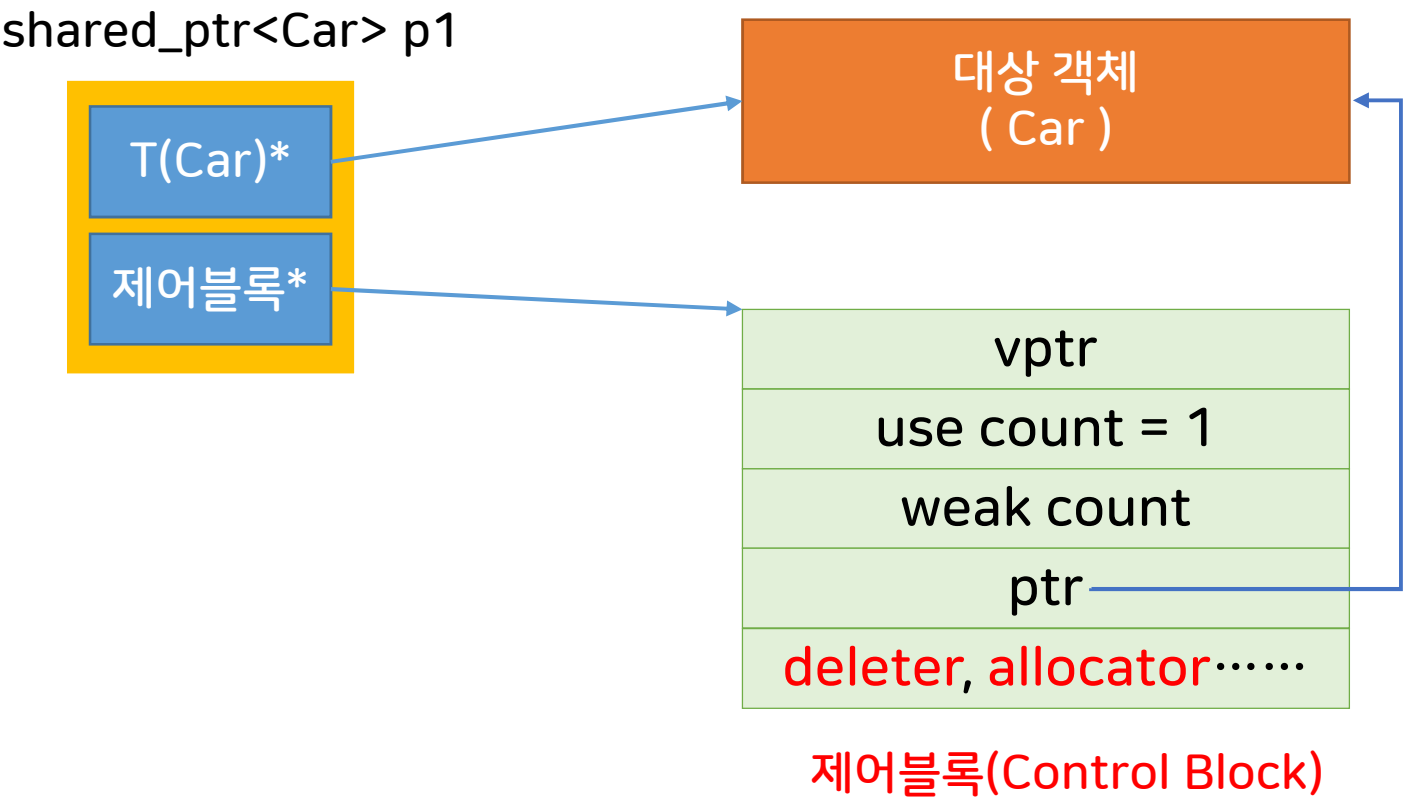


shared_ptr 특징

- raw pointer의 2배의 크기를 가지게 된다.
- shared_ptr 생성시 참조계수등을 관리하는 제어블록이 생성된다.
- 참조 계수의 증가/감소는 원자적 연산 (Atomic Operation) 으로 수행된다

삭제자 변경

- shared_ptr<> 생성자의 2번째 인자로 삭제자 전달.
- 함수, 함수 객체, 람다표현식등을 사용



할당자(allocator) 변경

- shared_ptr<> 생성자의 3번째 인자로 할당자 전달.
- 제어블럭을 만들고 파괴할 때 할당자 사용

shared_ptr 과 배열 – until C++17

- 삭제자를 변경해야 한다.
- [] 연산자가 제공 되지 않는다.
- shared_ptr을 사용해서 배열을 관리하는 것은 권장 되지 않았다. – vector 또는 array 를 사용 권장

C++17 이후

- shared_ptr<T[]> 를 사용한다.
- shared_ptr<T> vs shared_ptr<T[]>

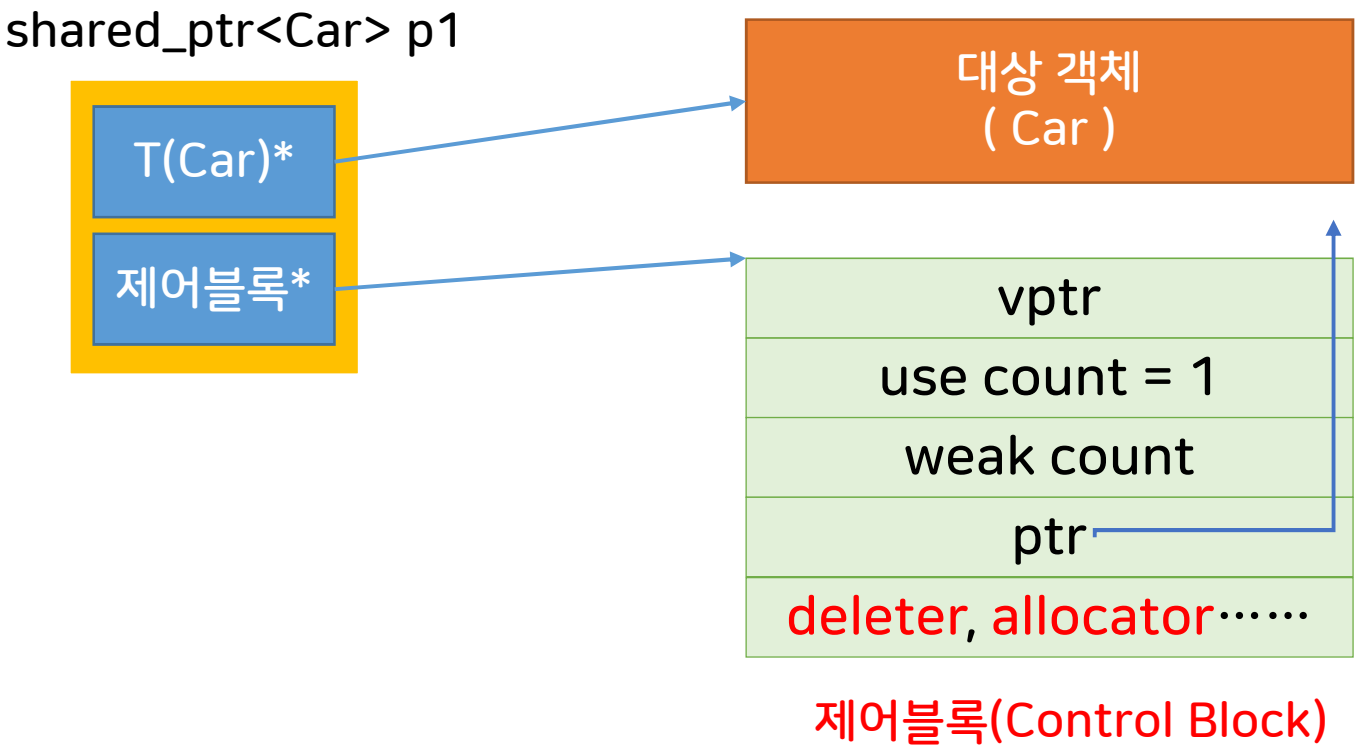
shared_ptr<T>	[] 연산 제공 안됨. delete 를 사용한 자원 해지
shared_ptr<T[]>	[] 연산 제공 됨 delete[] 를 사용한 자원 해지

-> 연산과 . 연산

-> 연산	대상 객체 (Car) 의 멤버에 접근
. 연산	shared_ptr 자체 멤버에 접근

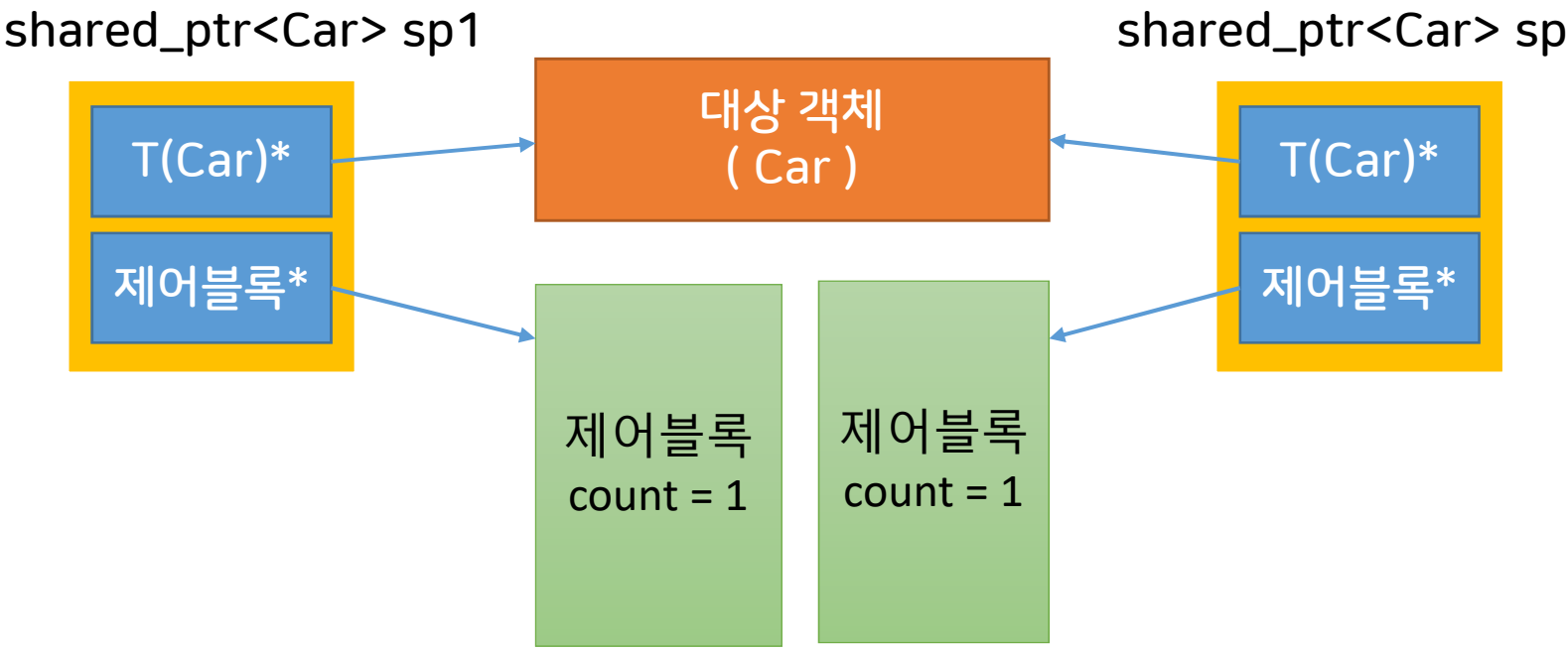
shared_ptr 주요 멤버 함수

get	대상체의 포인터 반환
use_count	참조계수 반환
reset	대상체 변경
swap	대상체 교환

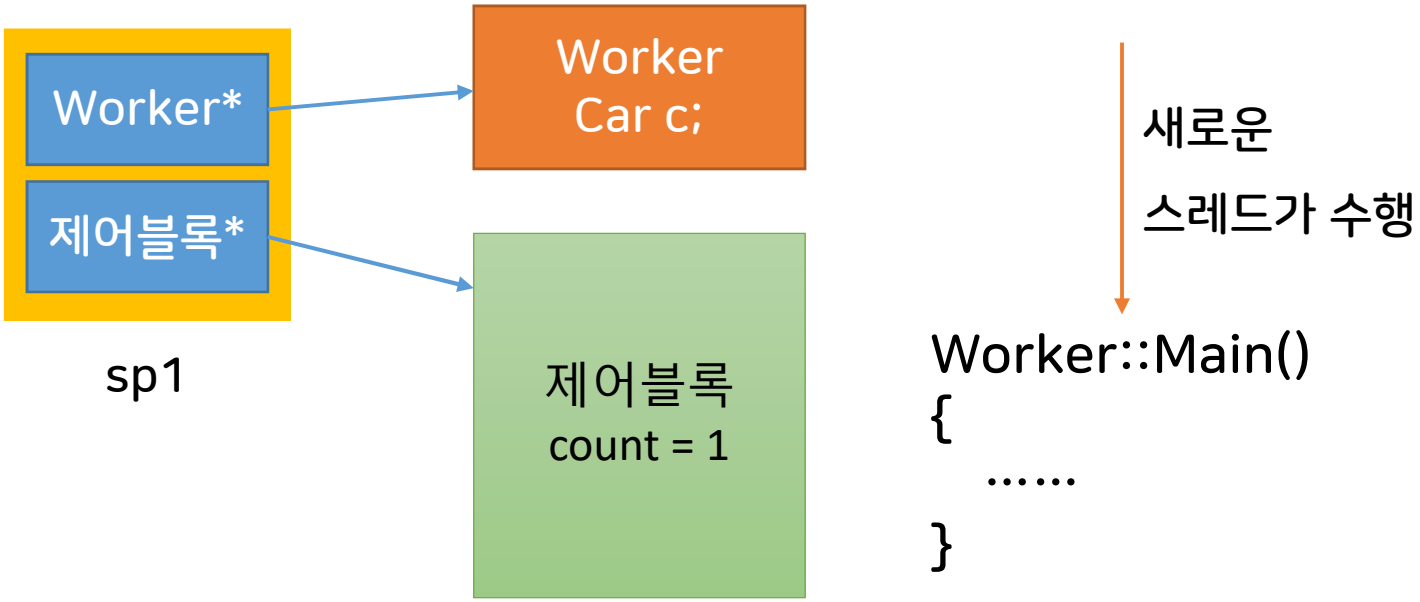


make_shared

- 대상 객체와 제어 블럭을 동시에 메모리 할당 하므로 보다 효율적이다.
- 예외 상황에 좀 더 안전하다.
- 메모리 할당/해지 방식을 변경하려면 allocate_shared 를 사용한다.



- raw pointer 를 사용해서 2개이상의 shared_ptr 를 생성하면 안된다.



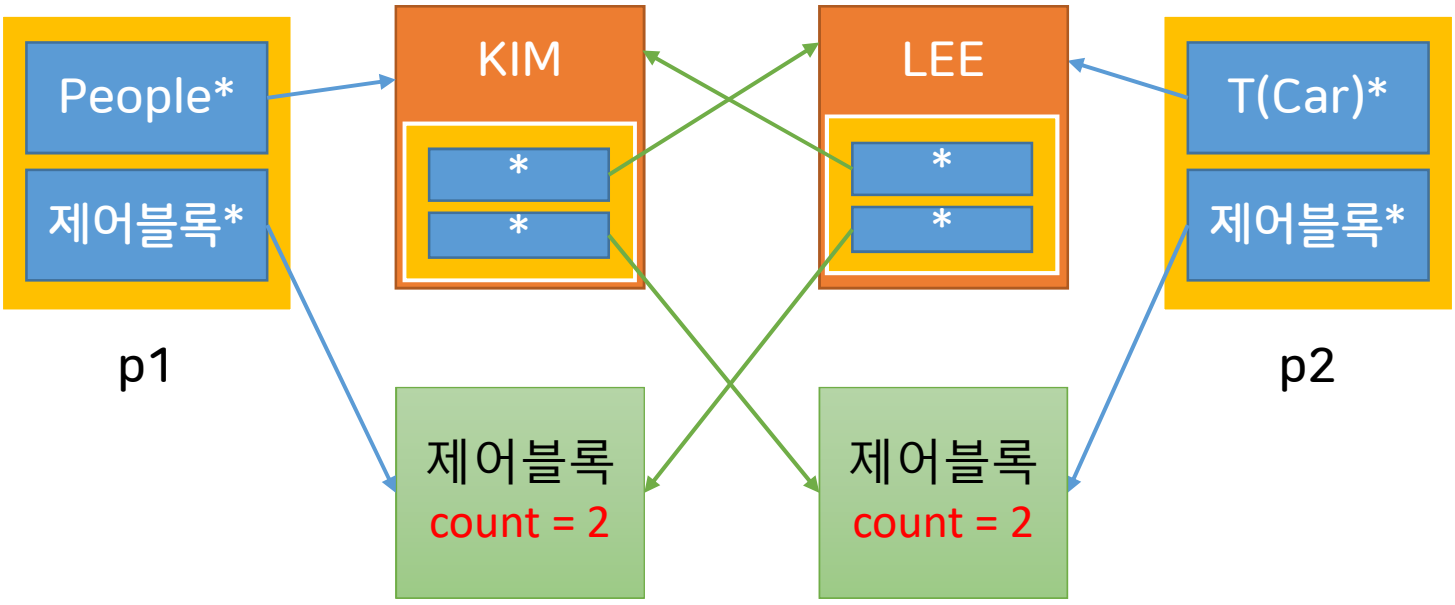
Worker 객체의 수명

- 주 스레드의 sp1도 사용하지 않고,새로운 스레드도 종료 되었을 때 파괴 되어야 한다.
- Worker 객체가 자기 자신의 참조계수를 증가 해야 한다.

enable_shared_from_this

- this를 가지고 제어 블럭을 공유하는 shared_ptr 을 만들수 있게 한다. – CRTP 기술
- shared_from_this() 함수 사용
- shared_from_this() 를 호출하기 전에 만드시 제어블럭이 생성되어 있어야 한다.

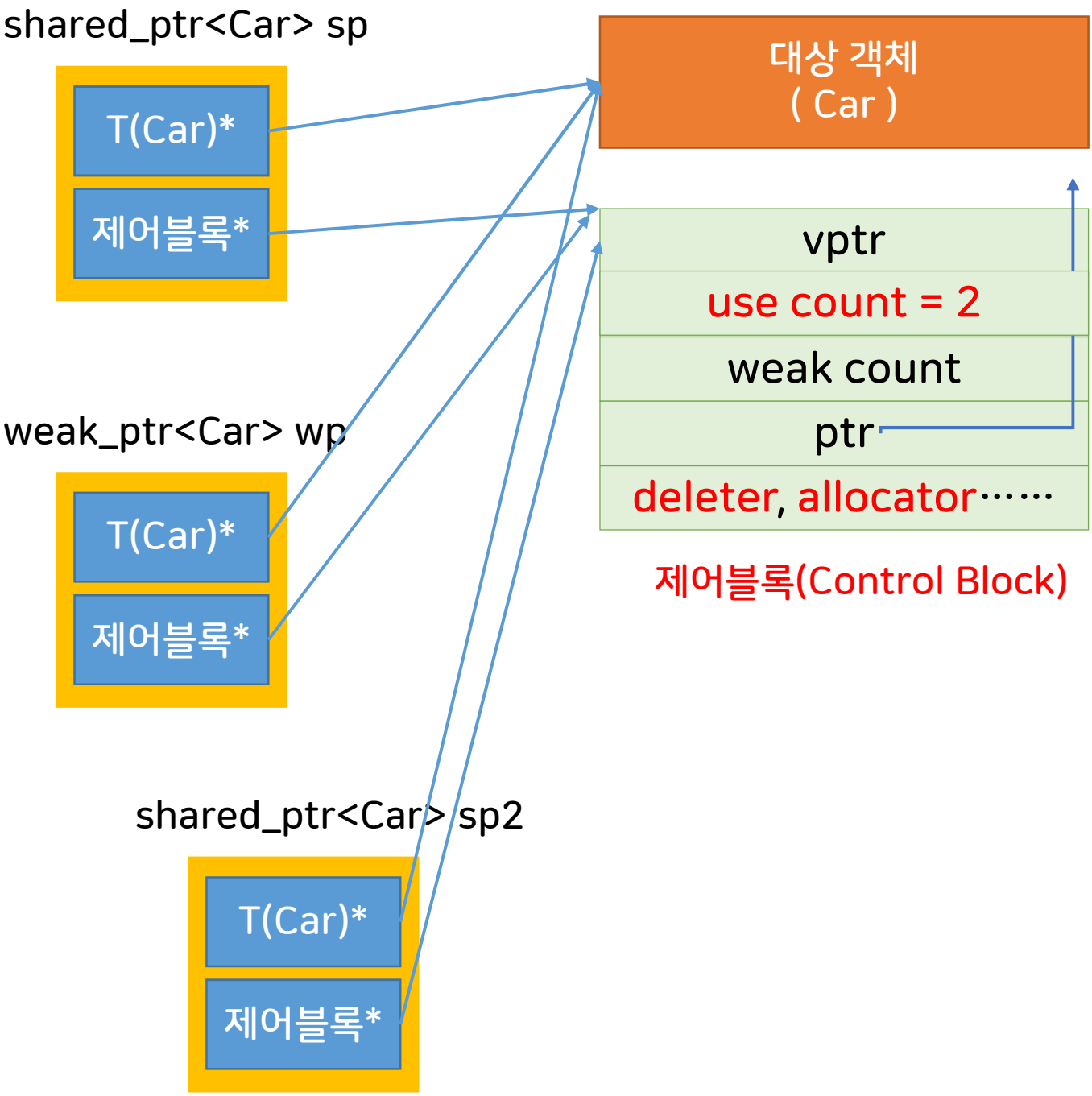
- shared_ptr 과 상호 참조



- shared_ptr 사용시 상호 참조가 발생하면 자원 해지가 되지 않는다.

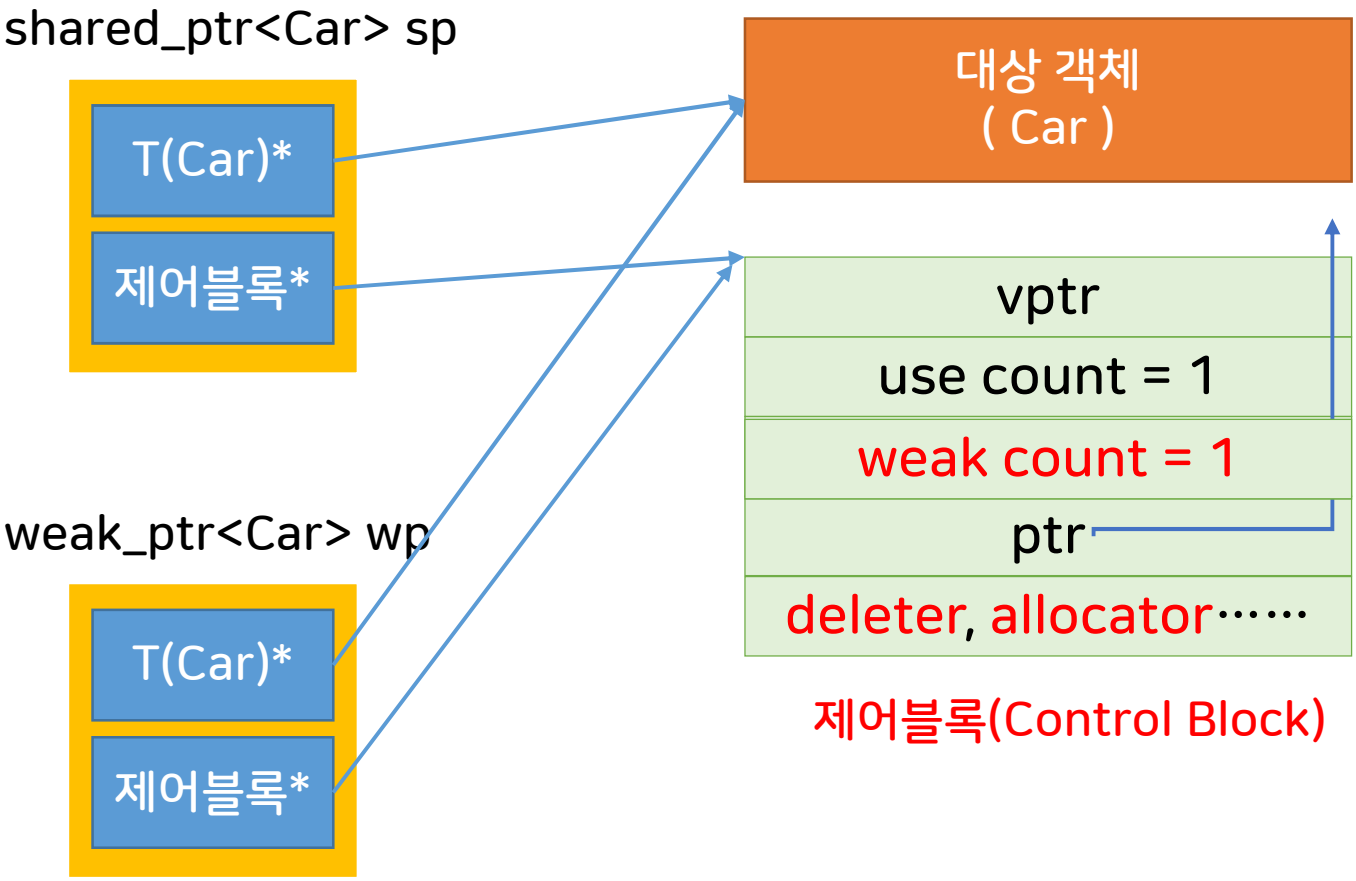
weak_ptr

- use count 가 증가하지 않는 스마트 포인터
- expired() 멤버 함수로 대상 객체의 유효성을 확인 할 수 있다.



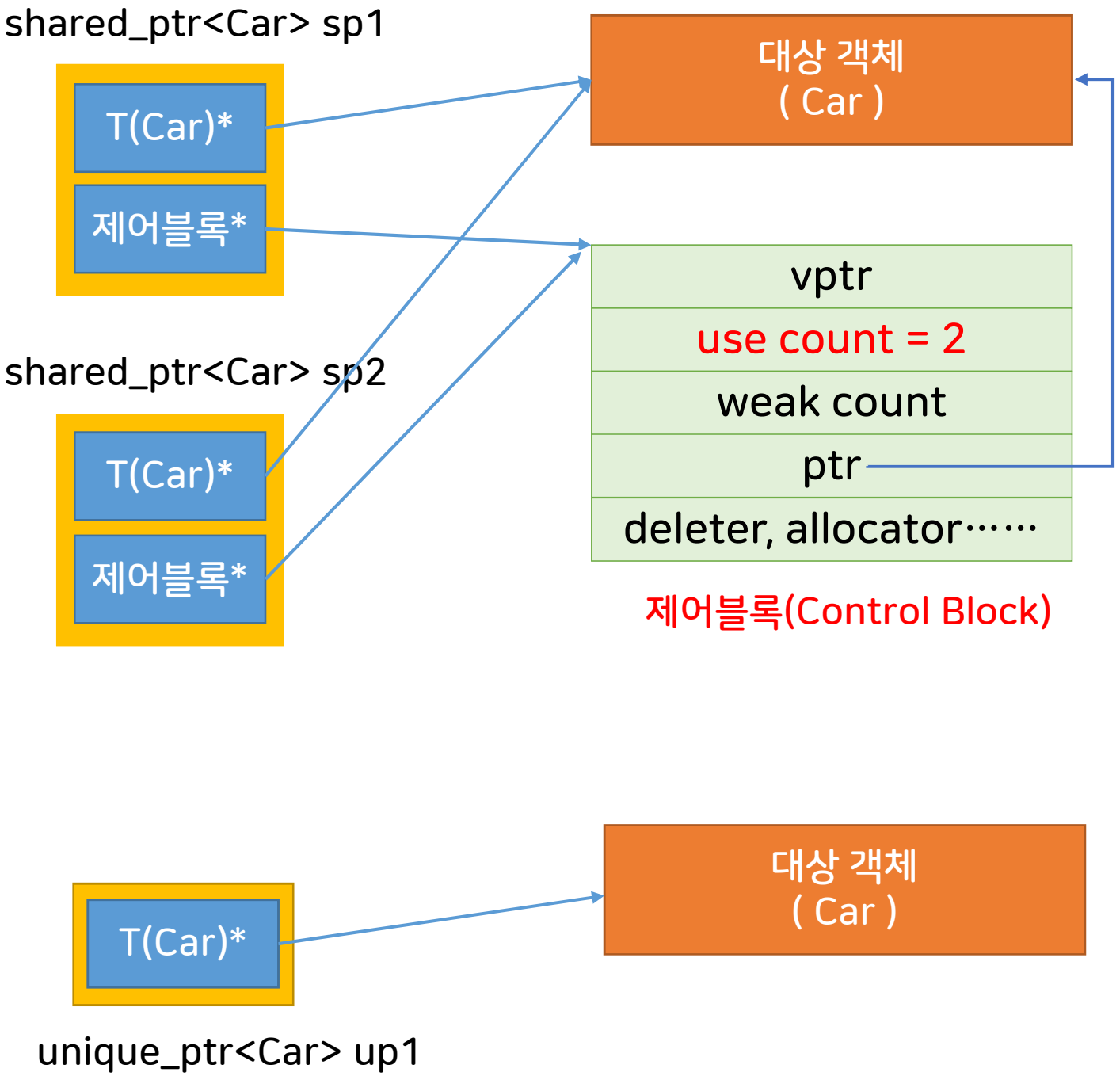
weak_ptr

- use count 가 증가하지 않는 스마트 포인터
- expired() 멤버 함수로 대상 객체의 유효성을 확인 할 수 있다.



use count == 0 일때 대상 객체는 파괴 되지만 제어 블록은 use count, weak count 가 모두 0일때 파괴 된다.

shared_ptr vs unique_ptr



- 기본적으로는 Raw Pointer 와 동일한 크기를 가진다. - 단, 삭제자 변경시 크기가 커질 수 있다.

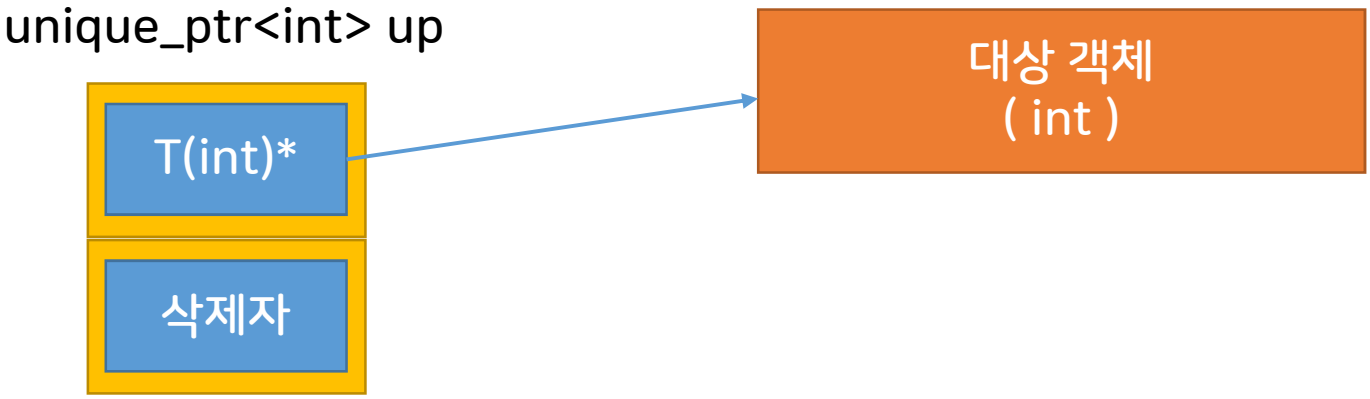
복사 될 수 없지만, 이동 될 수 는 있다.

삭제자 변경

shared_ptr	생성자를 사용해서 전달
unique_ptr	템플릿 인자와 생성자를 사용해서 전달

- 람다표현식, 함수 등을 사용하려면 템플릿 인자로 타입을 전달하고 생성자의 2번째 인자로 람다표현식(또는 함수)를 전달한다.
- 배열의 경우는 unique_ptr<T[]> 를 사용한다.

- 삭제자 변경시 unique_ptr의 크기가 커질 수 있다.



shared_ptr 과 unique_ptr

sp = up	Error
sp = move(up)	Ok
up = sp	Error
up = move(sp)	Error