CHUNGNAM NATIONAL UNIVERSITY

시스템 프로그래밍

강의 12. 메모리 II

교재 9 장

http://eslab.cnu.ac.kr

주요 할당 정책 요약

- 블럭 할당 정책:
  - 처음할당, 다음할당, 최적할당
  - 단편화를 줄이는 노력과 처리량을 늘리는 노력 간의 절충이 필요
    - segregated free lists 방식을 사용하면 free 리스트 전체를 검색하지 않고도 best fit 과 유사한 결과를 얻을 수 있다.
- 블럭 나누기 정책:
  - 언제 블록나누기를 하는 것이 좋은가 ?
  - 얼마만큼의 내부 단편화를 허용할 수 있는가?
- 블럭 통합 정책:
  - 즉시 통합 : free 가 호출 될 때 인접블럭과 즉시 통합
  - 지연 통합 : 통합의 효과를 극대화 할 수 있을 때까지 free 할 때의 통합을 지연 e.g.,
    - malloc 실행을 위해서 가용리스트를 검색할 때 통합하는 방안
    - 외부단편화의 양이 일정수준을 넘으면 통합하는 방안

간접 리스트 방식 : 요약

- 구현난이도 : 매우 간단(?)
- 할당 : linear time worst case
- Free: constant time worst case – 통합을 구현하더라도
- 메모리 사용량 : 할당정책에 따라 다르다
  - **First fit, next fit or best fit**

- 실제 `malloc/free` 의 구현에서는 이용하지 않고 있다. 그 이유는 할당시의 선형적인 성능 때문이다. 특수한 경우에만 이용된다

- 그렇지만 블럭 나누기와 경계 태그를 이용하여 가용 블럭을 통합하는 방법은 모든 할당 라이브러리에서 이용하는 방법이다.

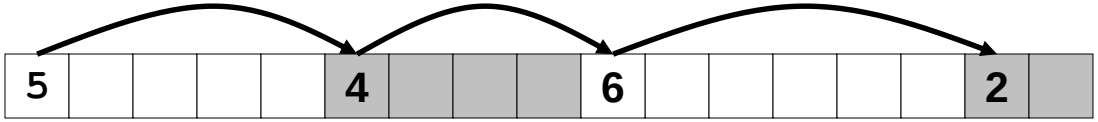
연습문제 1. 메모리 블록의 설계

- 다음과 같은 alignment 조건과 블록설계를 하는 경우에 최소블럭 크기를 결정하시오. 간접리스트 방식으로 구현하며, 단, 할당된 블록의 payload의크기는 0보다 커야하고, free블록에서는 payload가 0이어도 됨. header와 footer는 4바이트 워드 크기로 저장된다.

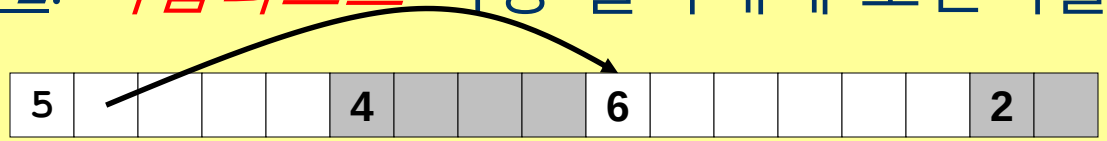
Alignment	할당된 블록	**Free** 블록	최소블럭크기 **(bytes)**
Single-word	**Header, footer**	**Header, footer**	
Double-word	**Header**	**Header, footer**	

Free 블럭 관리하기

- *방법 1*: *간접리스트* 크기 정보를 이용하여 모든 블록을 연결



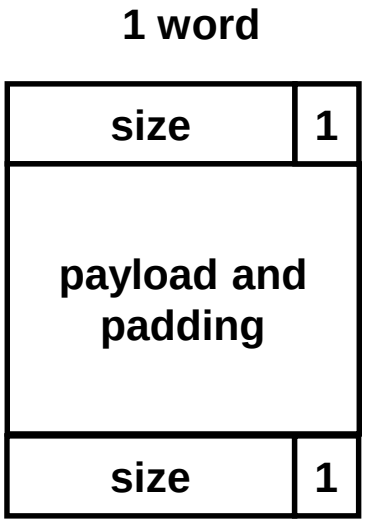
- *방법 2*: *직접리스트* 가용 블록내에 포인터를 이용



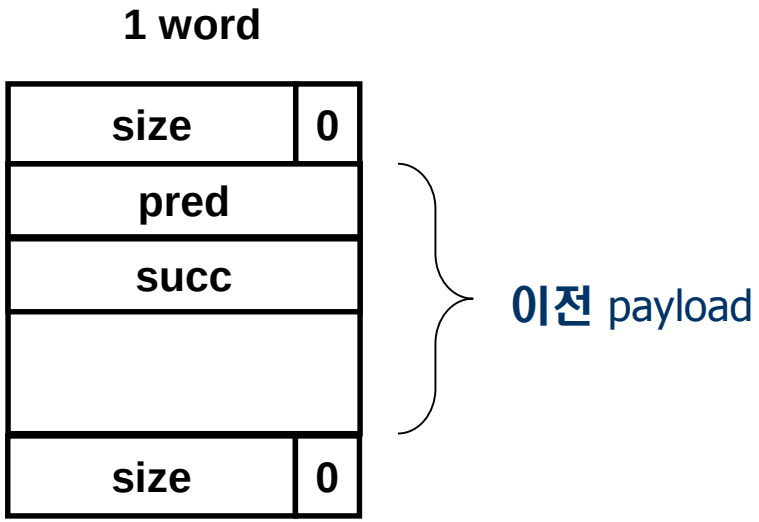
- *방법 3*: *구분 가용 리스트 segregated free list*
  - 크기 클래스마다 각각 별도의 가용 리스트를 유지
- *방법 4*: 크기로 정렬된 블록
  - 가용 블록내에 포인터를 이용하고, 크기를 키로 사용하여 균형 트리를 사용할 수 있다

직접 가용리스트

- 가용 블록들의 리스트만을 관리하며, 모든 블록을 관리하지는 않는다
  - "다음"가용 블록의 위치는 정해지지 않는다
    - 따라서, 크기로는 접근이 안되므로, 포인터를 저장한다
  - 블록연결을 위해서는 여전히 경계태그를 사용한다
  - 이 경우에 가용블록들만 관리하면 되므로, 데이터 영역을 활용할 수 있게 된다



할당된 블록의 구조
(이전과 동일)

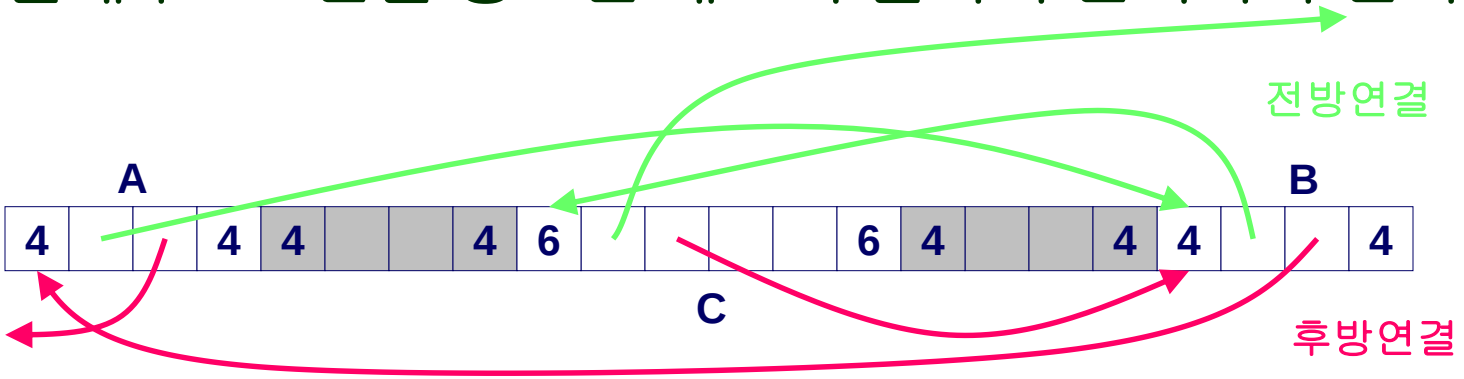


가용블록의 구조

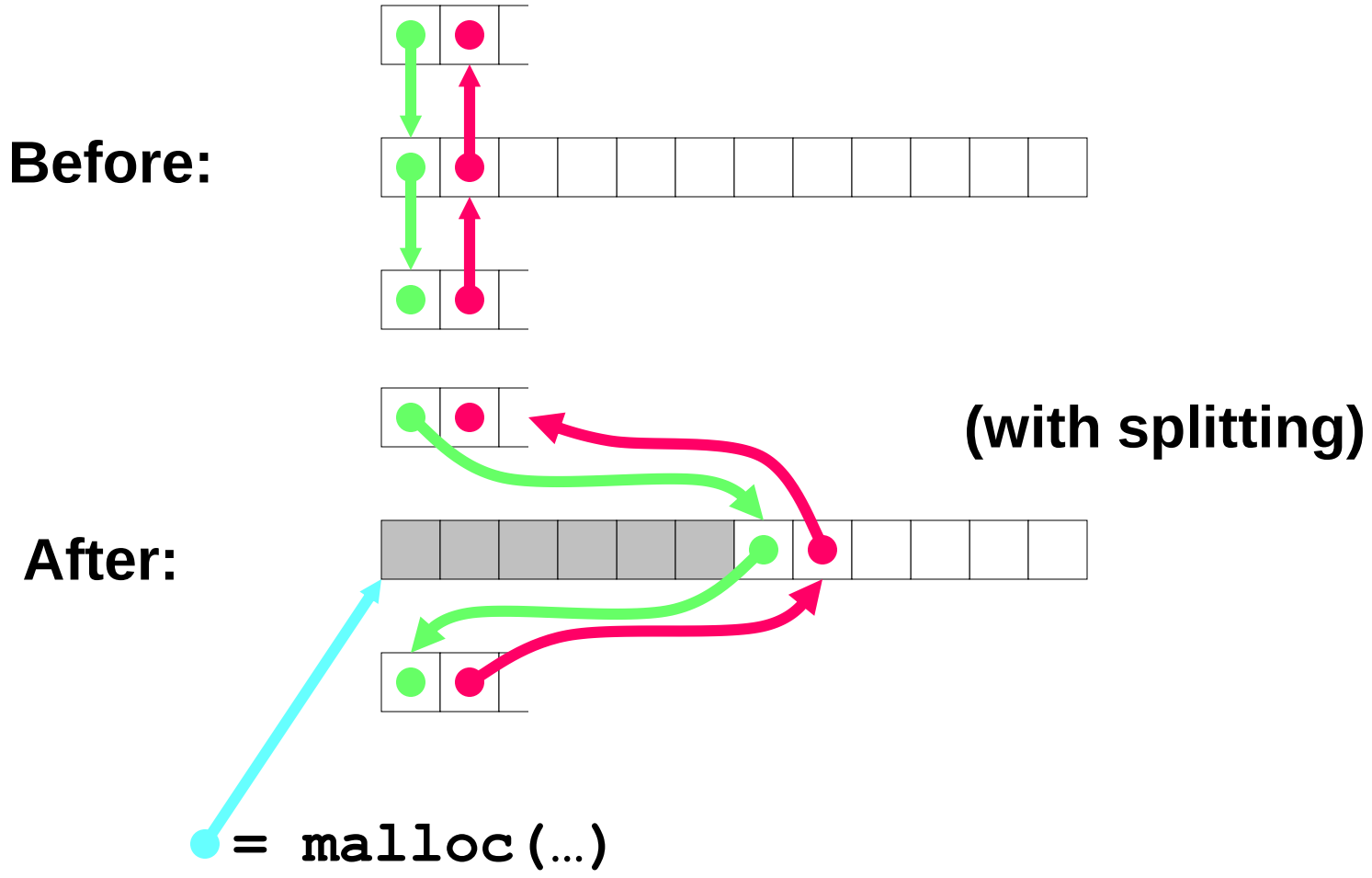
직접 가용 리스트

논리적인 구조

실제구조 : 연결 링크는 메모리 블럭의 순서와 무관하다



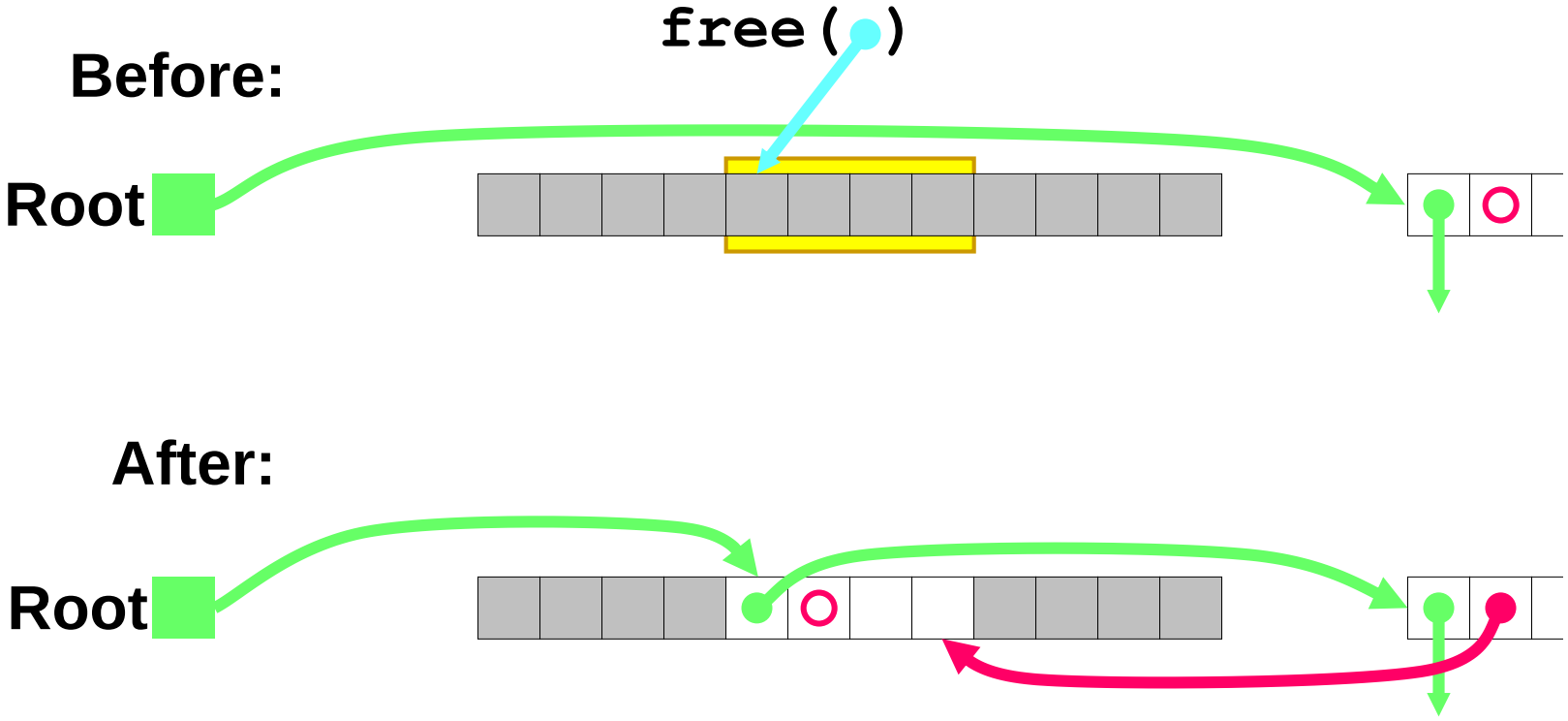
직접 Free 리스트 할당하기(이중 연결리스트)



직접 가용 리스트에서의 Free 작업

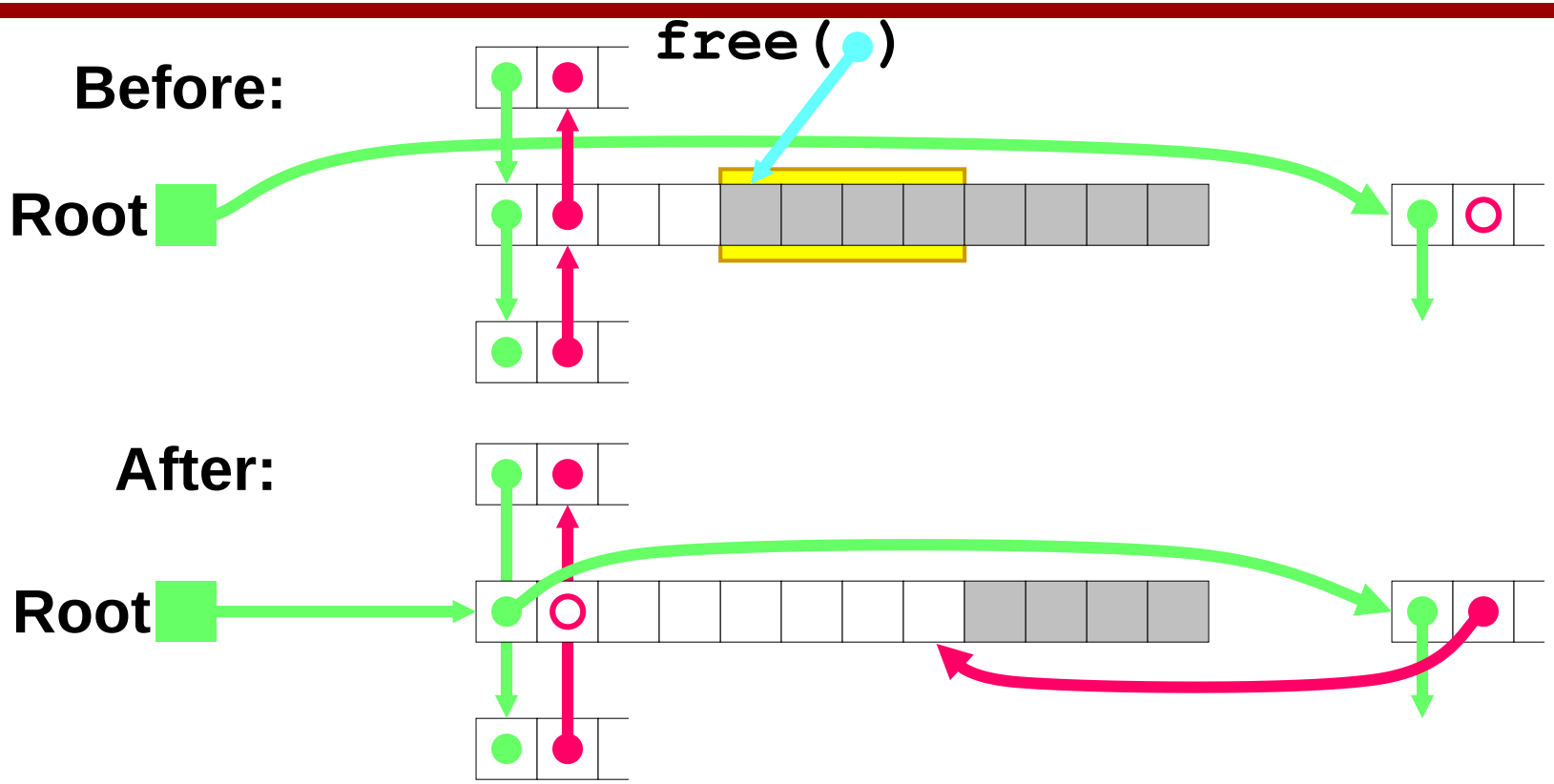
- 삽입 방법 :새롭게 반환한 블록은 가용리스트의 어느 위치에 끼워넣어야 하는가?
  - **LIFO (last-in-first-out) 정책**
    - 반환블록을 리스트 맨 앞에 끼워넣는 방법
    - 장점 : 간단하고 상수시간소요
    - 단점: 연구에 의하면 단편화가 주소정렬방식보다 나빠진다
  - **주소정렬 정책**
    - 가용블록 리스트가 블록들의 주소가 순서를 유지하도록 삽입
      - i.e. addr(pred) < addr(curr) < addr(succ)
    - 단점: 리스트를 탐색해야 한다
    - 장점: 연구에 의하면 LIFO 보다 단편화 성능이 우수하다

LIFO 방식에서의 Free (Case 1)



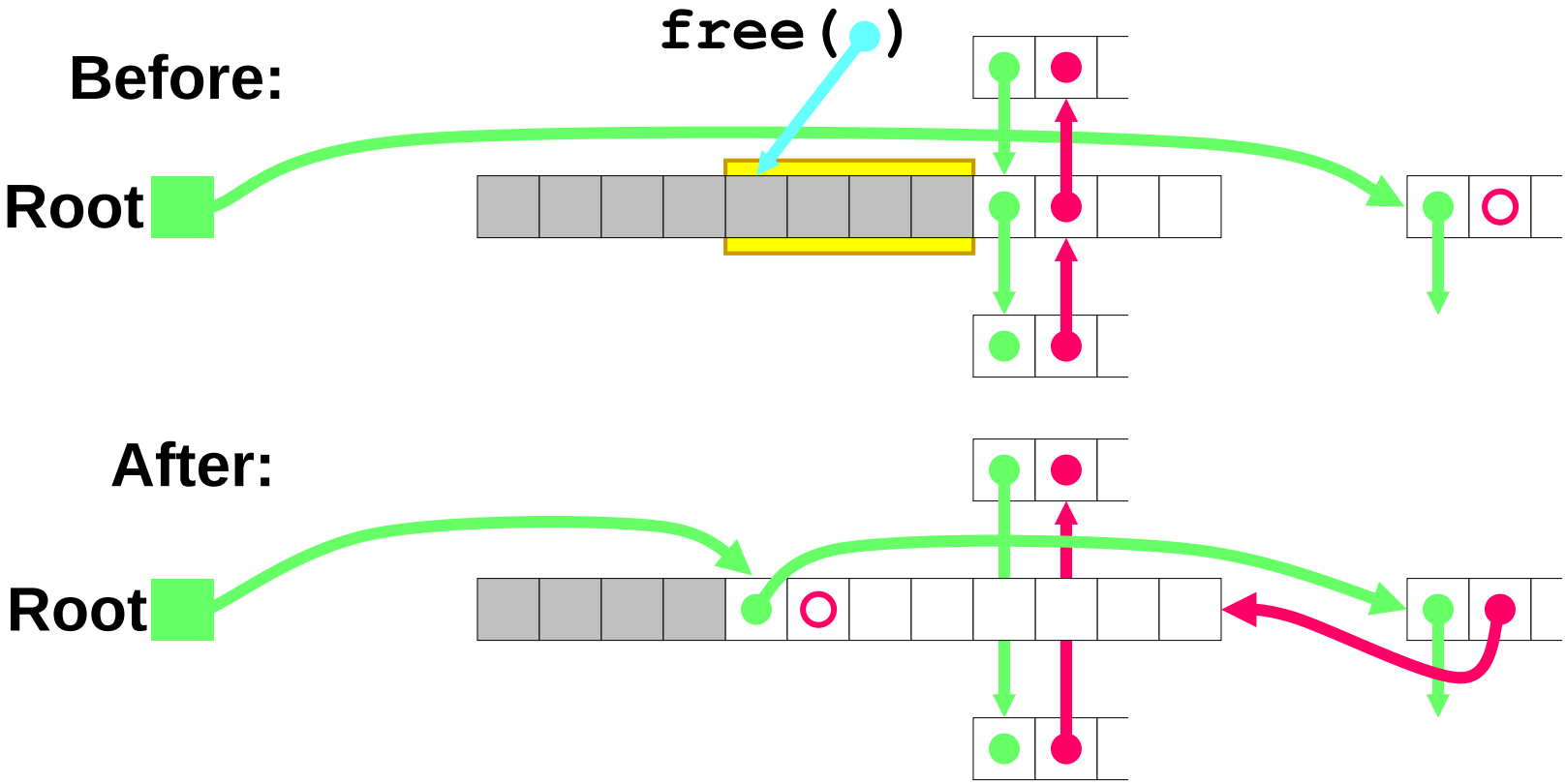
- 반환하는 블록을 리스트 맨 앞에 추가

LIFO 방식에서의 Free (Case 2)



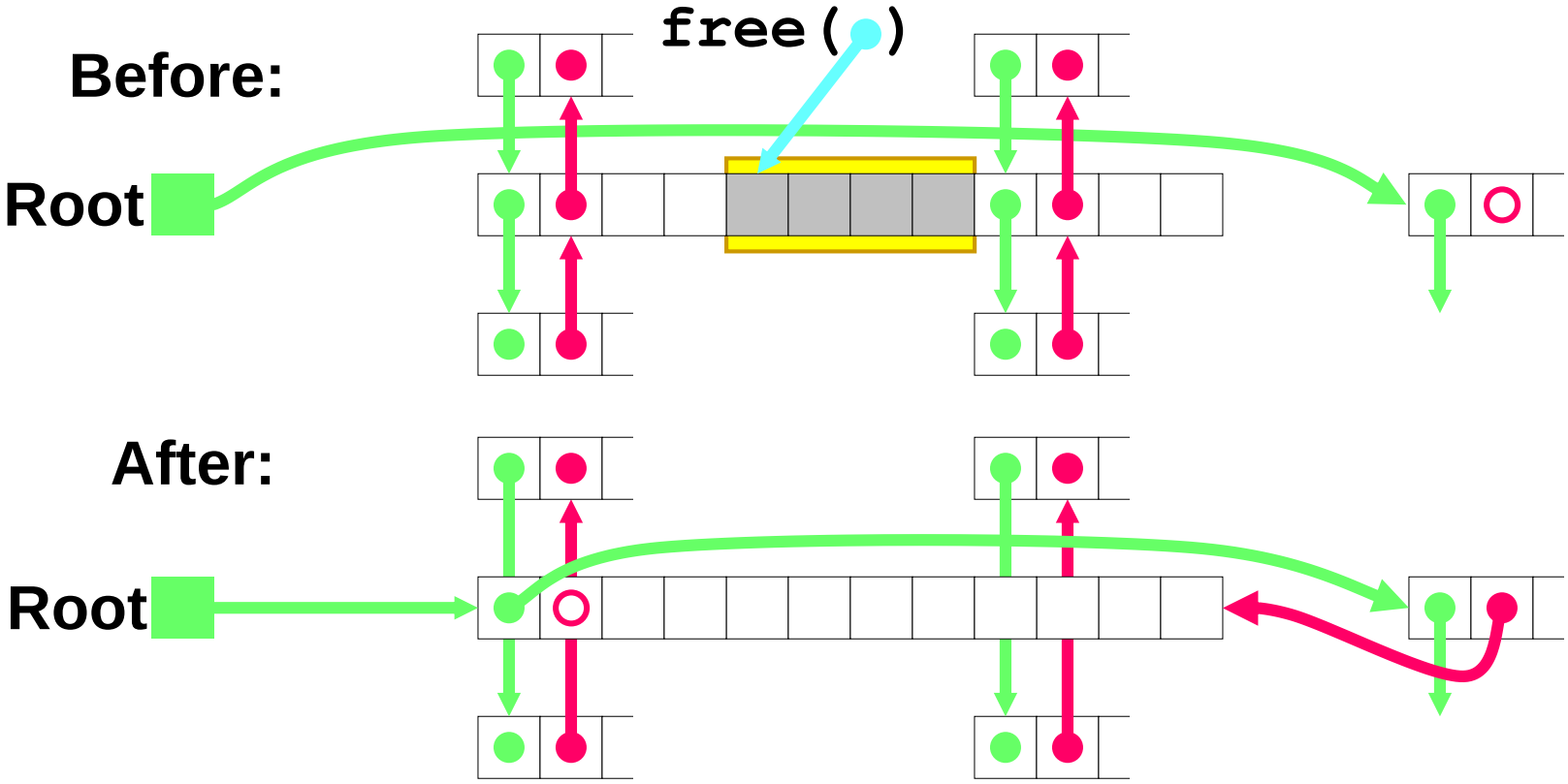
- 이전블록을 빼내고, 두 블록을 연결하고, 새 블록을 리스트 root 노드로 만든다

LIFO 방식에서의 Free (Case 3)



- 다음블록을 빼내고, 두 블록을 연결하고, 리스트의 root 에 삽입

LIFO 방식에서의 Free (Case 4)



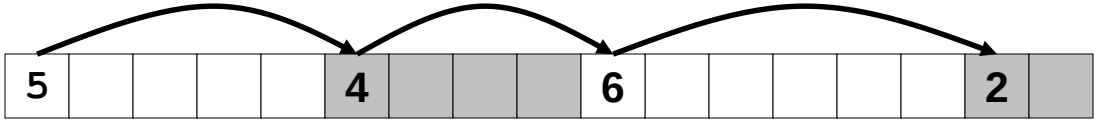
- 앞, 뒤의 블록 모두를 떼어내고, 세 블록을 모두 연결하고 , 이 새블록을 root 노드로 만든다

직접 리스트 요약

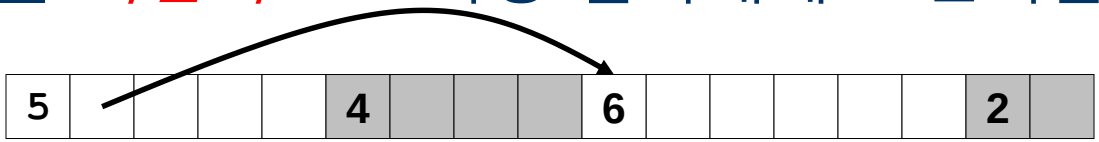
- 간접리스트 방식과의 비교
  - 할당시간이 전체 블록의 수가 아니라 가용블록 수에 비례한다 – 메모리가 거의 차 있는 경우에 매우 빠른 할당성능을 갖는다
  - 할당과 반환과정이 약간 더 복잡한데, 그 이유는 블록들을 리스트에 추가했다가 떼어냈다가 하는 작업이 필요하기 때문이다
  - 링크포인터 저장을 위해 추가적인 공간이 필요하다(블록마다 2워드 추가필요)
- 연결리스트 방식은 구분가용리스트와 함께 주로 사용된다
  - 여러개의 크기 클래스들의 연결리스트를 사용
  - 객체별로 별도의 연결리스트를 사용

Free 블럭 관리하기

- *방법 1*: *간접리스트* 크기 정보를 이용하여 모든 블록을 연결



- *방법 2*: *직접리스트* 가용 블록내에 포인터를 이용

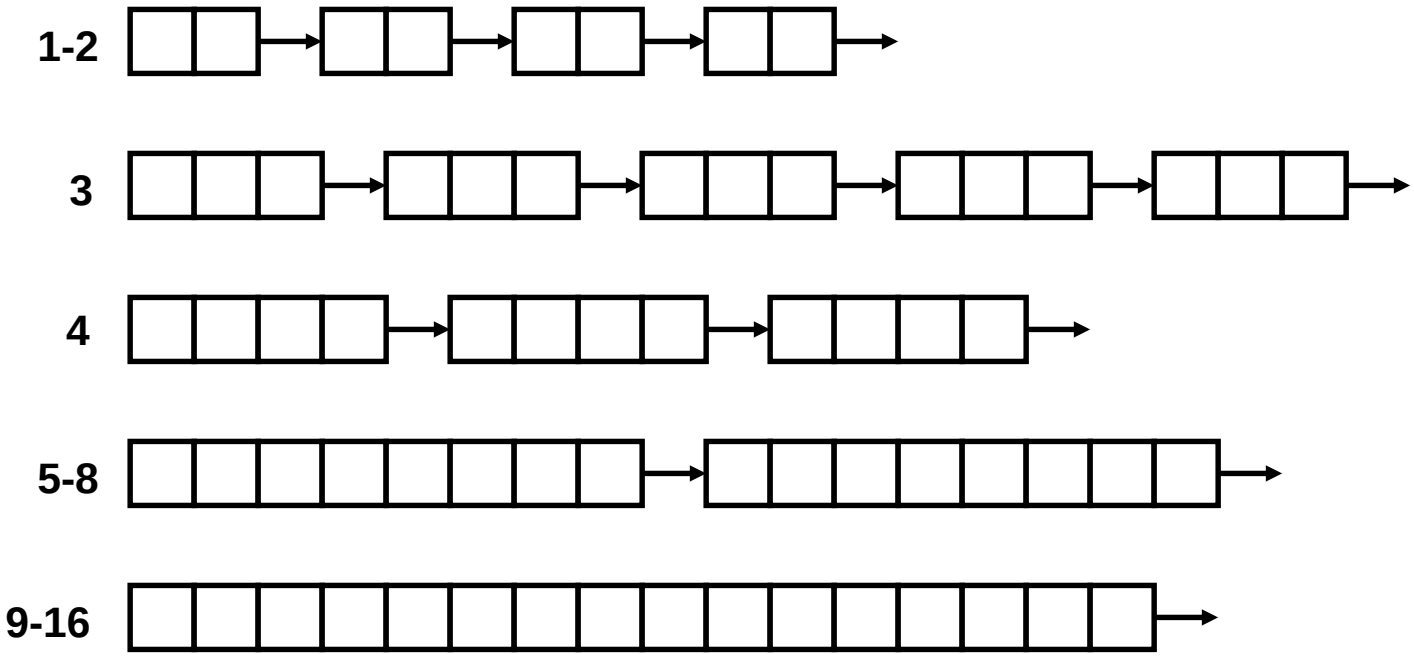


- *방법 3*: *구분 가용 리스트 segregated free list*
  - 크기 클래스마다 각각 별도의 가용 리스트를 유지

- *방법 4*: 크기로 정렬된 블록
  - 가용 블록내에 포인터를 이용하고, 크기를 키로 사용하여 균형 트리를 사용할 수 있다

구분 가용리스트 : 가용블록 관리

- 각 크기 클래스들은 클래스마다의 블록들을 관리한다



  - 대개 적은 크기의 블록들은 매 크기마다 클래스를 갖도록 한다**(2,3,4,…)**
  - 대개 크기가 큰 경우는 **2**의 제곱 크기마다 클래스를 정의한다

간단한 구분 가용리스트 할당기

- 힙을 분리하고, 각 크기 클래스별로 가용리스트를 운영

- 크기 n의 블록할당 방법 :
  - **크기 n의 블록 리스트가 비어 있지 않다면,**
    - 리스트의 첫 블록을 할당한다(리스트는 간접 또는 직접 모두 가능)
  - **가용리스트가 비어 있다면,**
    - 새 페이지를 할당받는다 (OS로부터 sbrk( ) 를 사용하여)
    - 이 페이지의 모든 블록들로부터 새로운 가용리스트를 생성한다
    - 리스트의 첫 블록을 할당한다
  - **상수시간**
- 블록 반환방법:
  - **가용 리스트에 추가**
  - **블록연결 후 해당 클래스 가용리스트에 추가**

구분 리스트 할당기

- 장점
  - 높은 처리량
    - 매우 빠른 상수 처리시간
  - 우수한 메모리 이용율
    - 최초할당 방법을 이용하면 전체 힙을 최적할당시와 유사한 성능
    - 각 할당 요청 블록에 대해 그 크기를 클래스로 각각 생성한다면, 최적할당과 동일해짐