CHUNGNAM NATIONAL UNIVERSITY

시스템 프로그래밍

강의 13. 메모리 II

교재 9 장

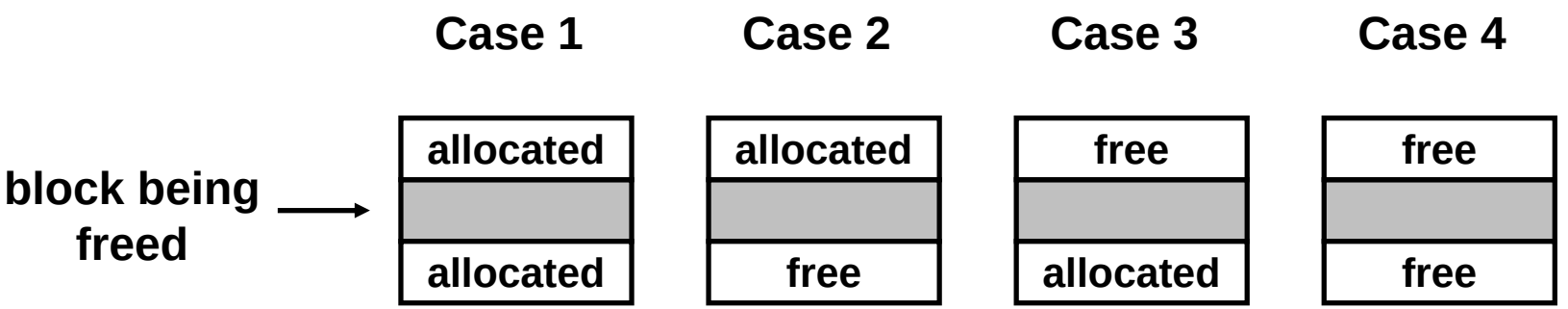
Nov. 19, 2010

http://eslab.cnu.ac.kr

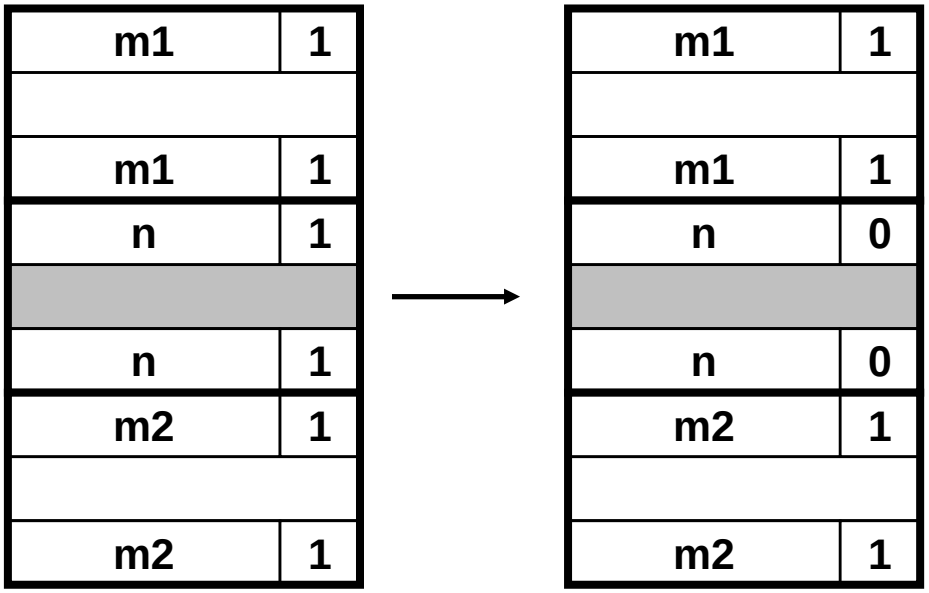
전달 사항

- 다음주 화요일 수업 휴강 => 오늘 보강했으니

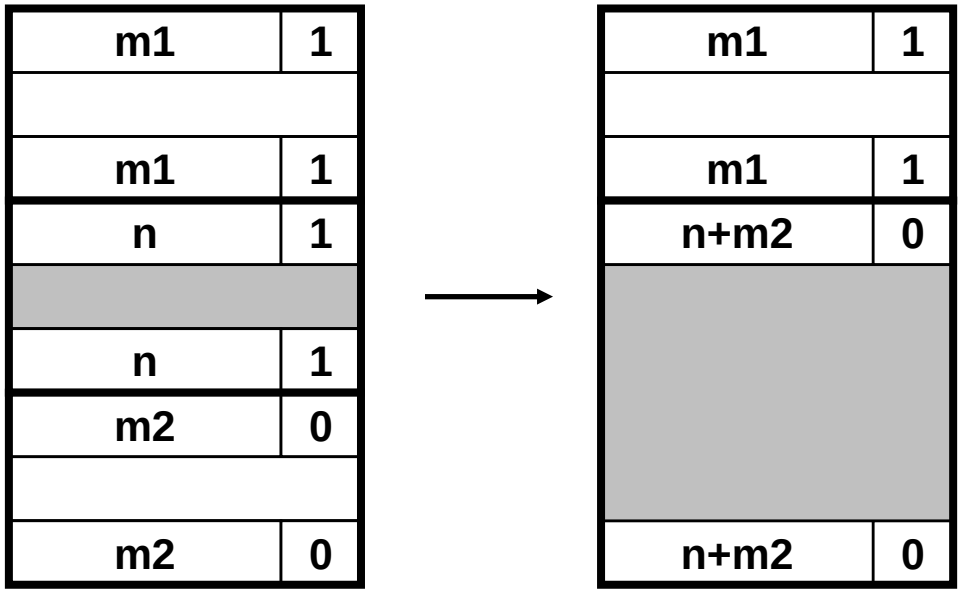
상수 소요시간 연결법


Case 1
Case 2
Case 3
Case 4
block being freed
allocated
allocated
allocated
free
free
allocated
free
free

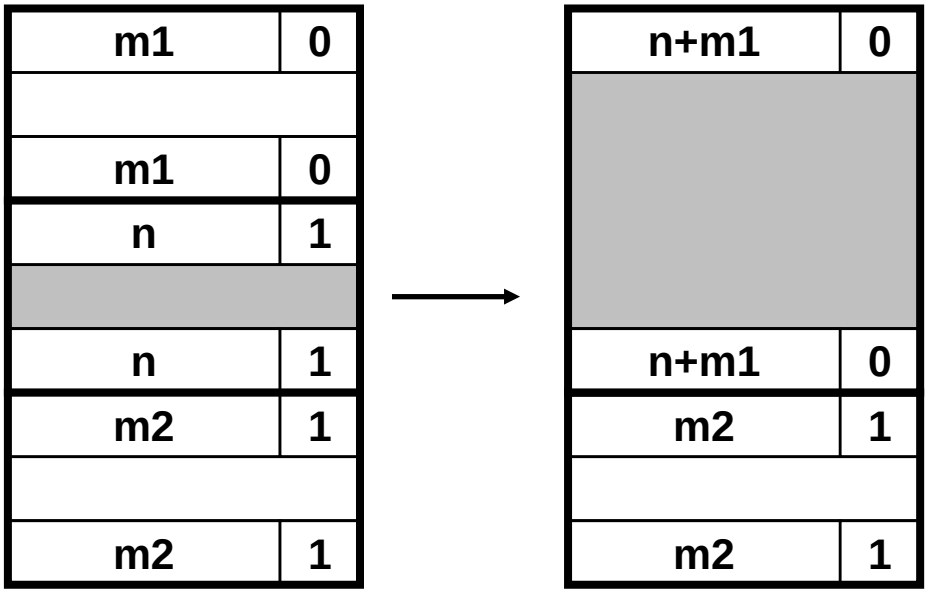
상수시간 연결법 (Case 1)



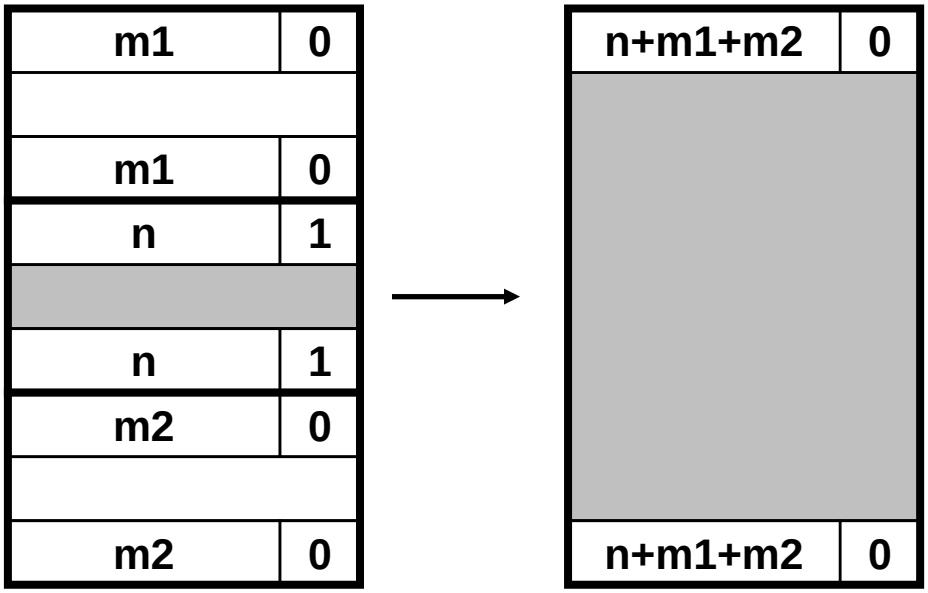
상수시간 연결법 (Case 2)



상수시간 연결법 (Case 3)



상수시간 연결법 (Case 4)



주요 할당 정책 요약

- 블럭 할당 정책:
  - **First fit, next fit, best fit, etc.**
  - **Trades off lower throughput for less fragmentation**
    - Interesting observation: segregated free lists 방식을 사용하면 free 리스트 전체를 검색하지 않고도 best fit 과 유사한 결과를 얻을 수 있다.
- 블럭 나누기 정책:
  - **When do we go ahead and split free blocks?**
  - **How much internal fragmentation are we willing to tolerate?**
- 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.,**
    - Coalesce as you scan the free list for malloc.
    - Coalesce when the amount of external fragmentation reaches some threshold.

간접 리스트 방식 : 요약

- Implementation: very simple
- Allocate: linear time worst case
- Free: constant time worst case -- even with coalescing
- Memory usage: will depend on placement policy
  - **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 바운더리 태그를 이용하여 free 블럭을 통합하는 방법은 모든 할당 라이브러리에서 이용하는 방법이다.

Knowledge Check 1

- 다음과 같은 순서로 malloc 요청을 하는 경우에 블록의 크기와 header에 저장되는 값을 쓰시오. 헤더는 4바이트를 사용한다.
  - 단, 할당은 double-word alignment를 유지해야 하고, header만 사용하는 간접리스트 방식으로 메모리를 관리한다고 가정한다.
  - 할당되는 블록의 크기는 8의 배수로 해야 한다.

요청	블록크기(바이트)	블록 **header(hex)**
Malloc(1)		
Malloc(5)		
Malloc(12)		

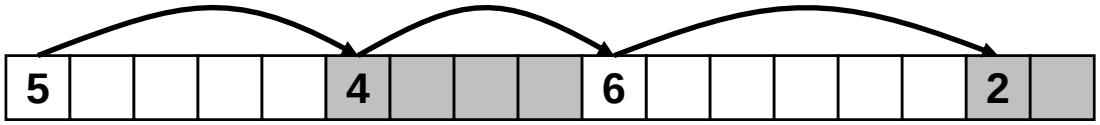
Knowledge Check 2

- 다음과 같은 alignment 조건과 블록설계를 하는 경우에 최소블럭 크기를 결정하시오. 간접리스트 방식으로 구현하며, 단, payload의 크기는 0보다 커야하고, header와 footer는 4바이트 워드 크기로 저장된다.

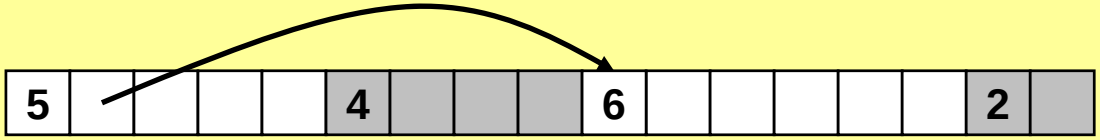
Alignment	할당된 블록	Free 블록	최소블럭크기 (bytes)
Single-word	Header, footer	Header, footer	
Double-word	Header	Header, footer	

Free 블록 관리하기(10.9.13)

- *Method 1*: Implicit list using lengths -- links all blocks



- *Method 2*: Explicit list among the free blocks using pointers within the free blocks

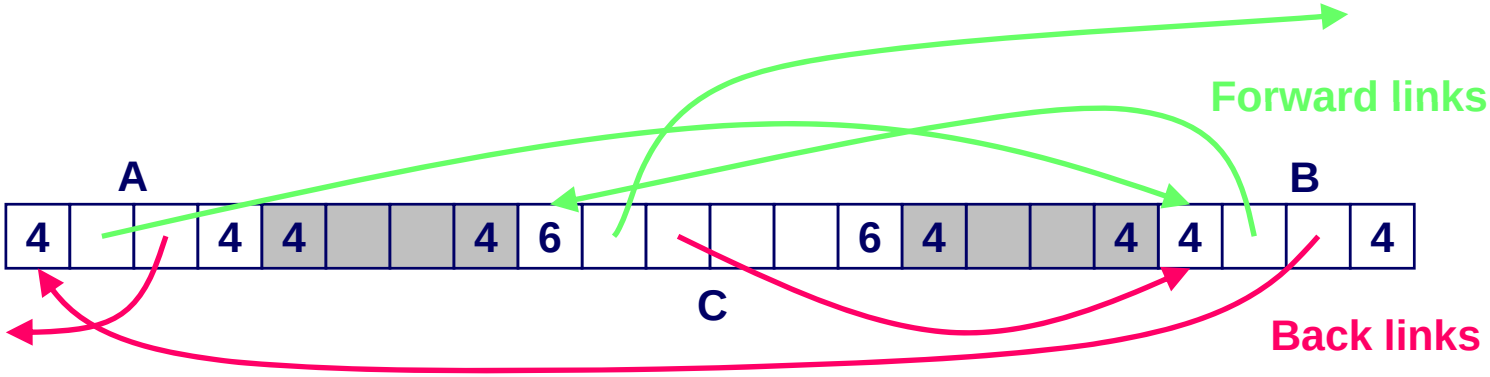


- *Method 3*: Segregated free lists
  - **Different free lists for different size classes**
- *Method 4*: Blocks sorted by size (not discussed)
  - **Can use a balanced tree (e.g. Red-Black tree) with pointers within each free block, and the length used as a key**

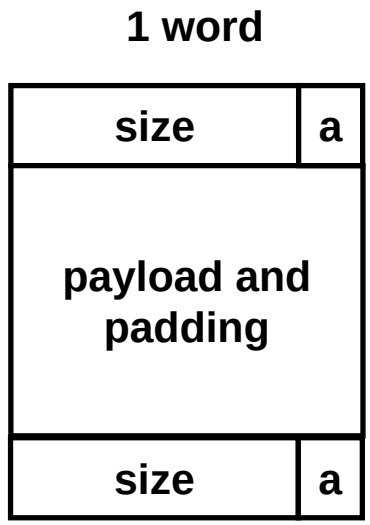
직접 Free 리스트

- Maintain list(s) of *free* blocks, not *all* blocks
  - **The "next" free block could be anywhere**
    - So we need to store pointers, not just sizes
  - **Still need boundary tags for coalescing**
  - **Luckily we track only free blocks, so we can use payload area**

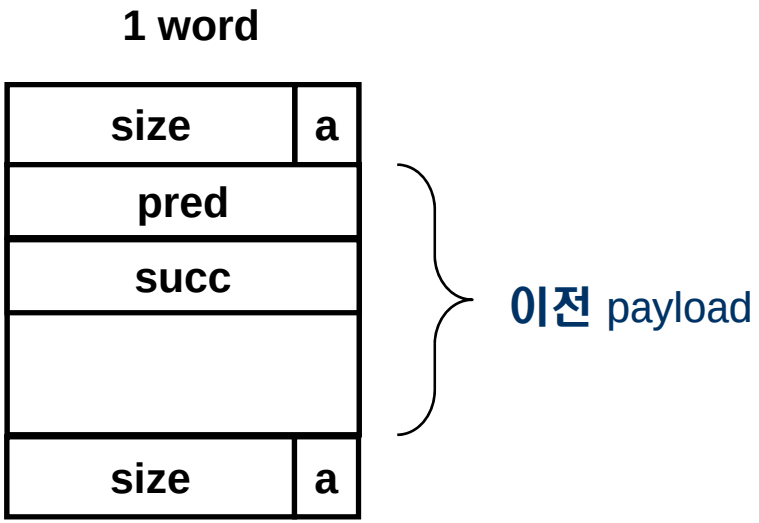
주의 : 연결 링크는 메모리 블럭의 순서와 무관하다



직접 리스트에서의 메모리 블럭 정의

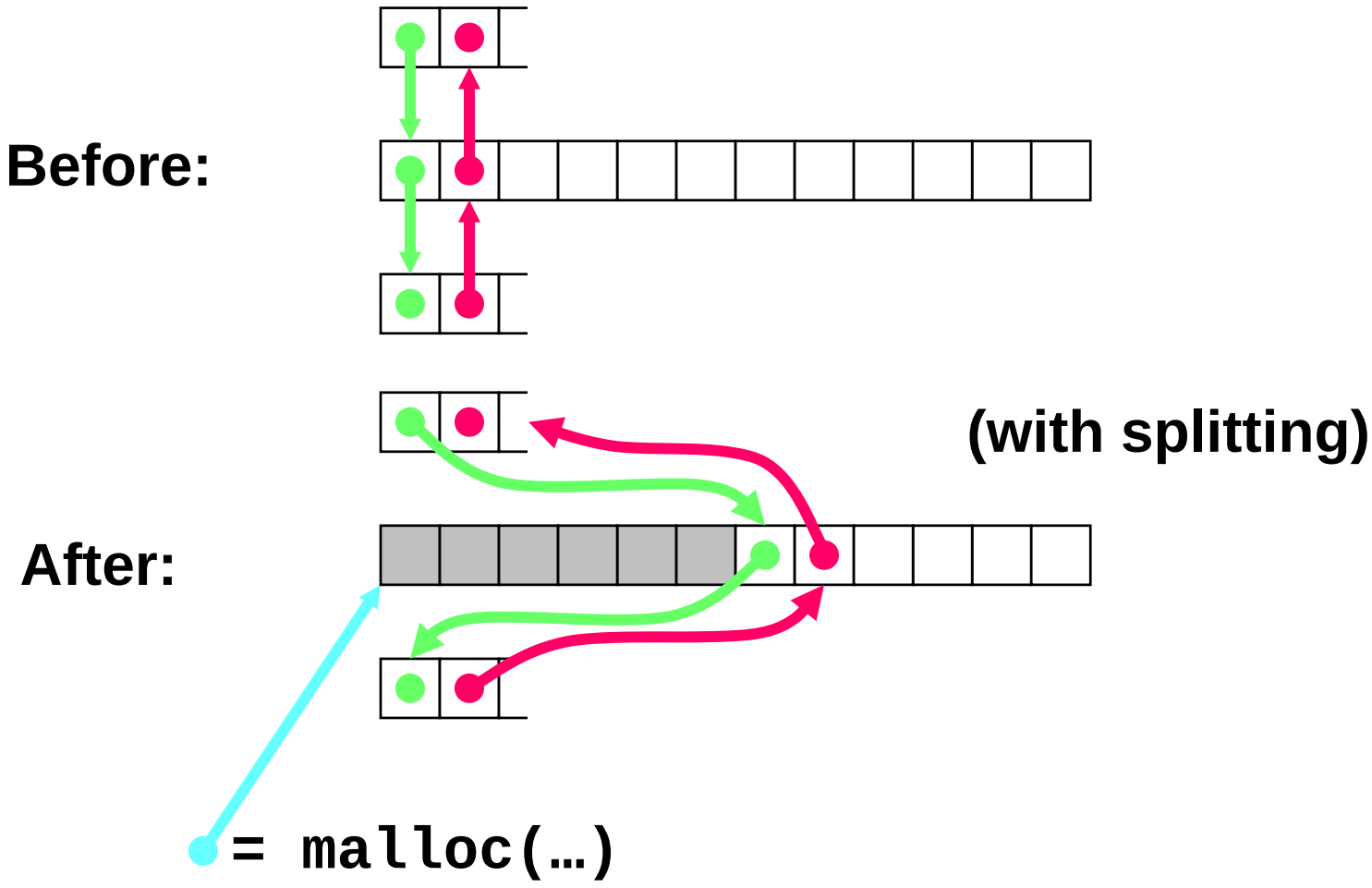


Format of
allocated blocks



Format of
free blocks

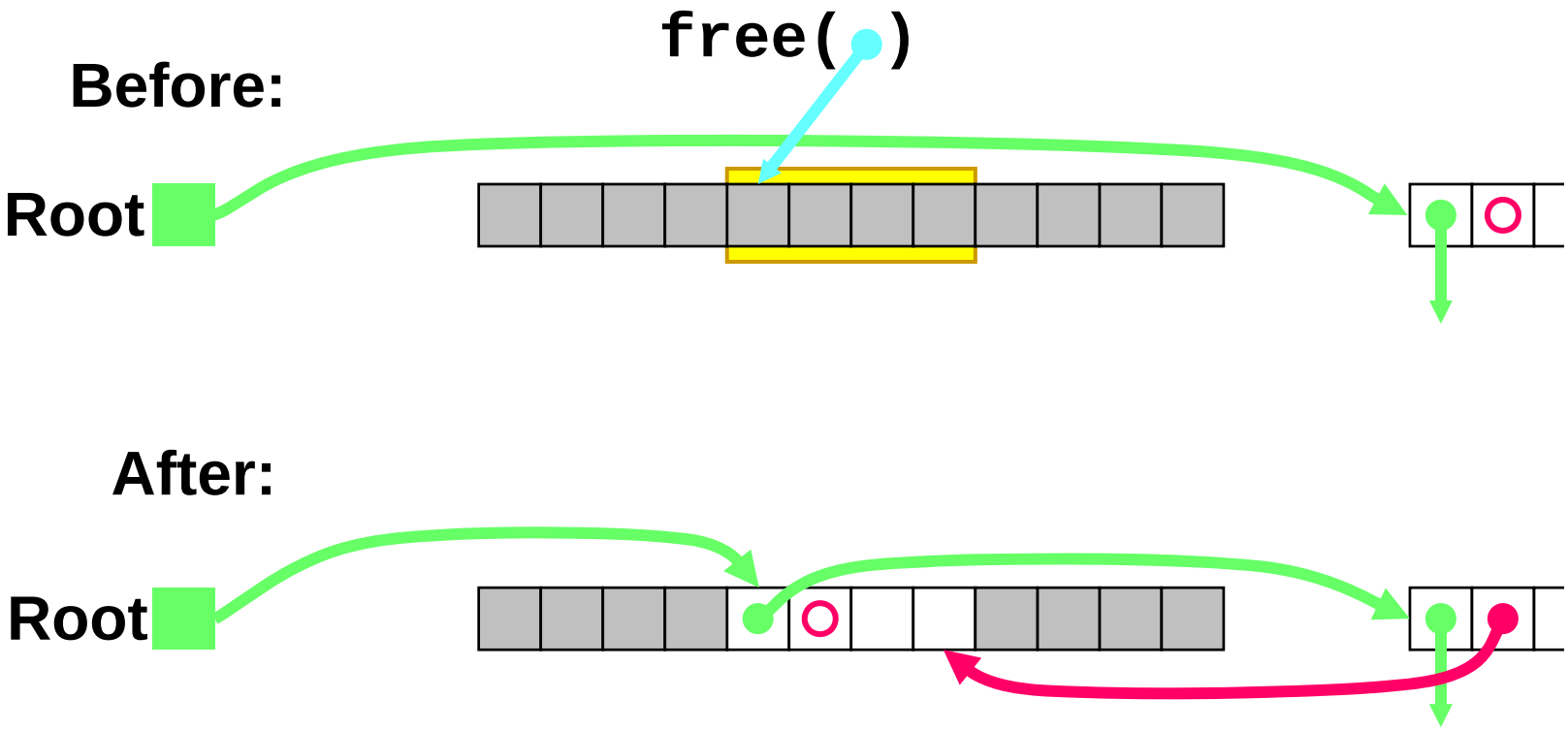
직접 Free 리스트 할당하기(이중 연결리스트)



직접 Free 리스트에서의 Free 작업

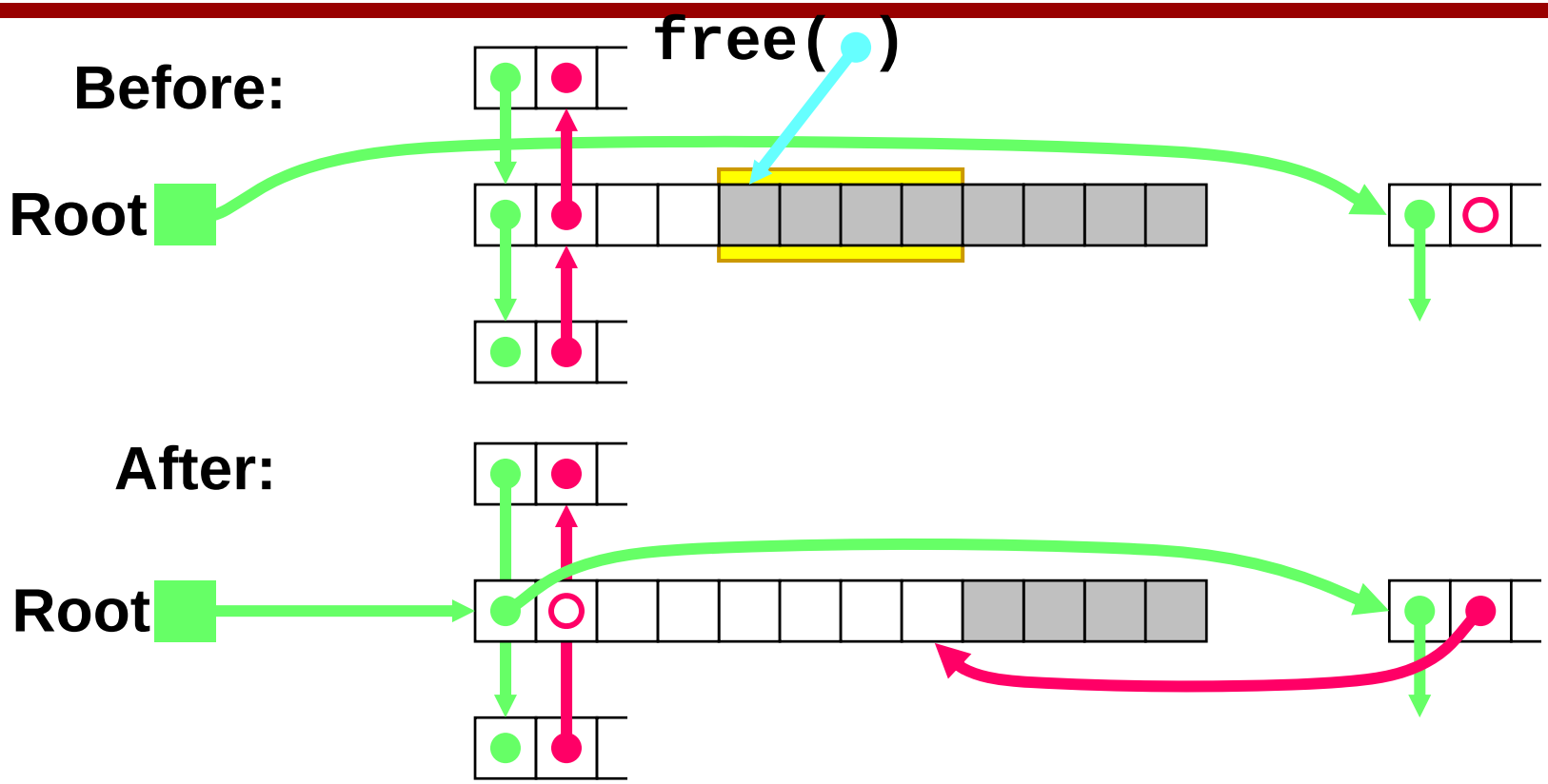
- 삽입 방법 : Where in the free list do you put a newly freed block?
  - **LIFO (last-in-first-out) policy**
    - Insert freed block at the beginning of the free list
    - Pro: simple and constant time
    - Con: studies suggest fragmentation is worse than address ordered.
  - **Address-ordered policy**
    - Insert freed blocks so that free list blocks are always in address order
      - i.e. addr(pred) < addr(curr) < addr(succ)
    - Con: requires search
    - Pro: studies suggest fragmentation is lower than LIFO

LIFO 방식 Free (Case 1)



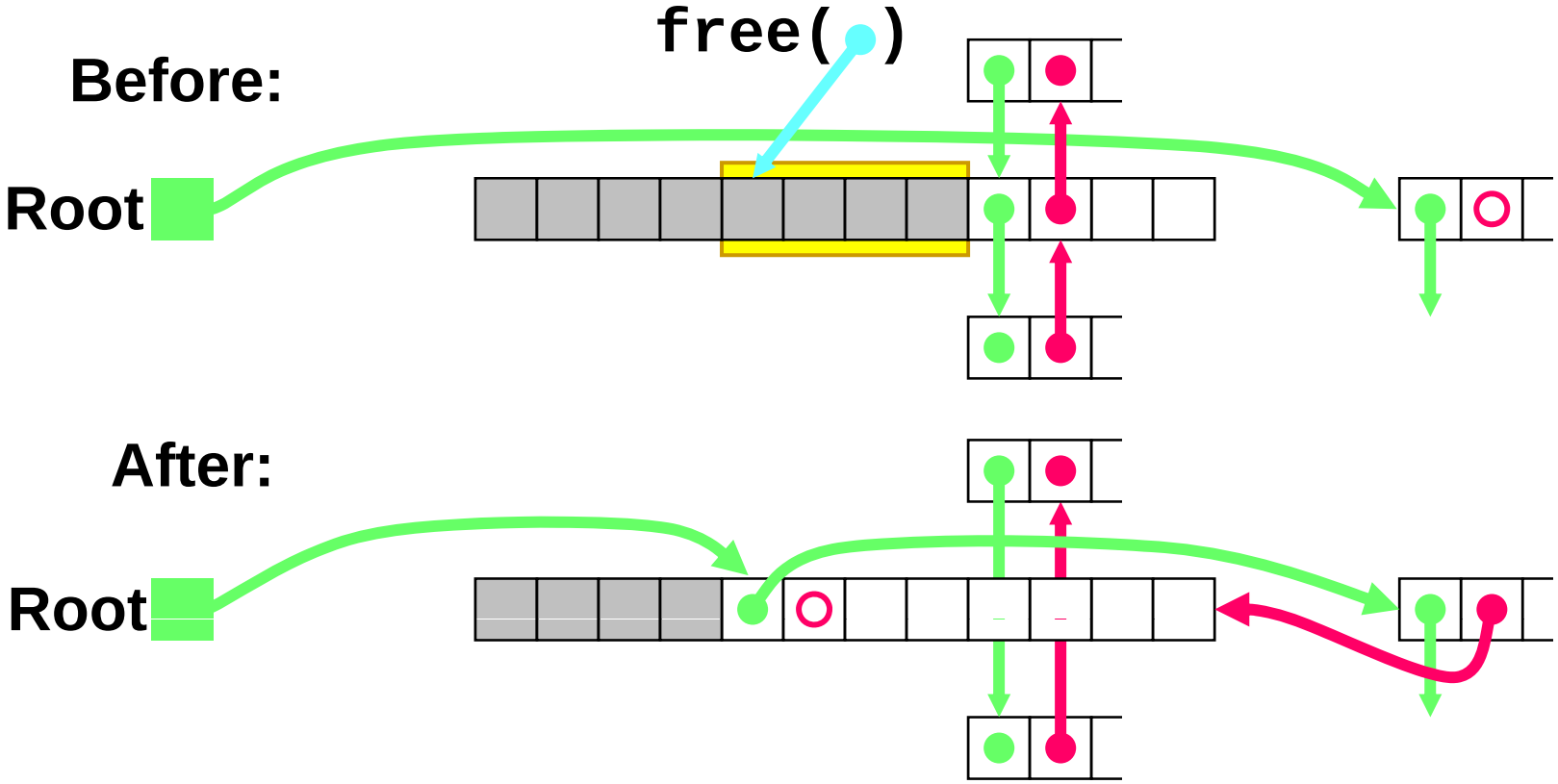
- Insert the freed block at the root of the list

LIFO 방식 Free (Case 2)



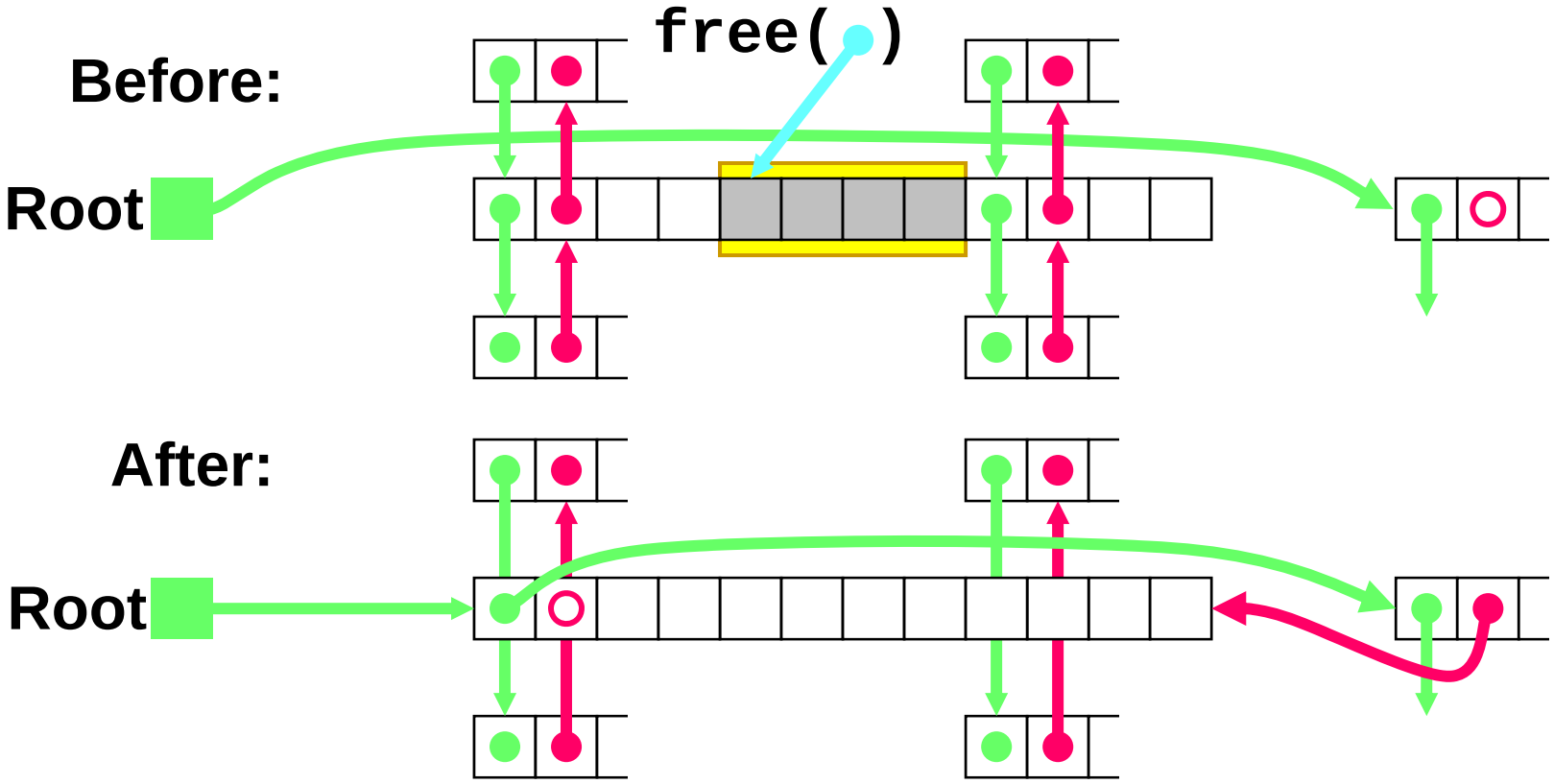
- Splice out predecessor block, coalesce both memory blocks and insert the new block at the root of the list

LIFO 방식 Free (Case 3)



- Splice out successor block, coalesce both memory blocks and insert the new block at the root of the list

LIFO 방식 Free (Case 4)



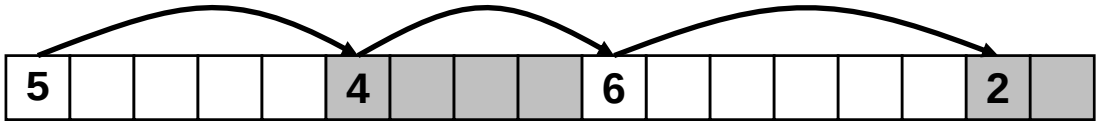
- Splice out predecessor and successor blocks, coalesce all 3 memory blocks and insert the new block at the root of the list

직접 리스트 요약

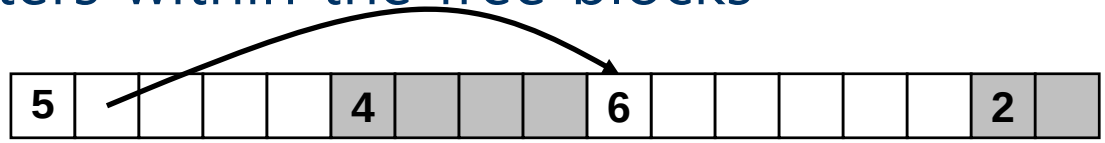
- Comparison to implicit list:
  - **Allocate is linear time in number of free blocks instead of total blocks -- much faster allocates when most of the memory is full**
  - **Slightly more complicated allocate and free since needs to splice blocks in and out of the list**
  - **Some extra space for the links (2 extra words needed for each block)**
- Main use of linked lists is in conjunction with segregated free lists
  - **Keep multiple linked lists of different size classes, or possibly for different types of objects**

Free 블록 관리(ch.10.9.14)

- *Method 1*: *Implicit list* using lengths -- links all blocks



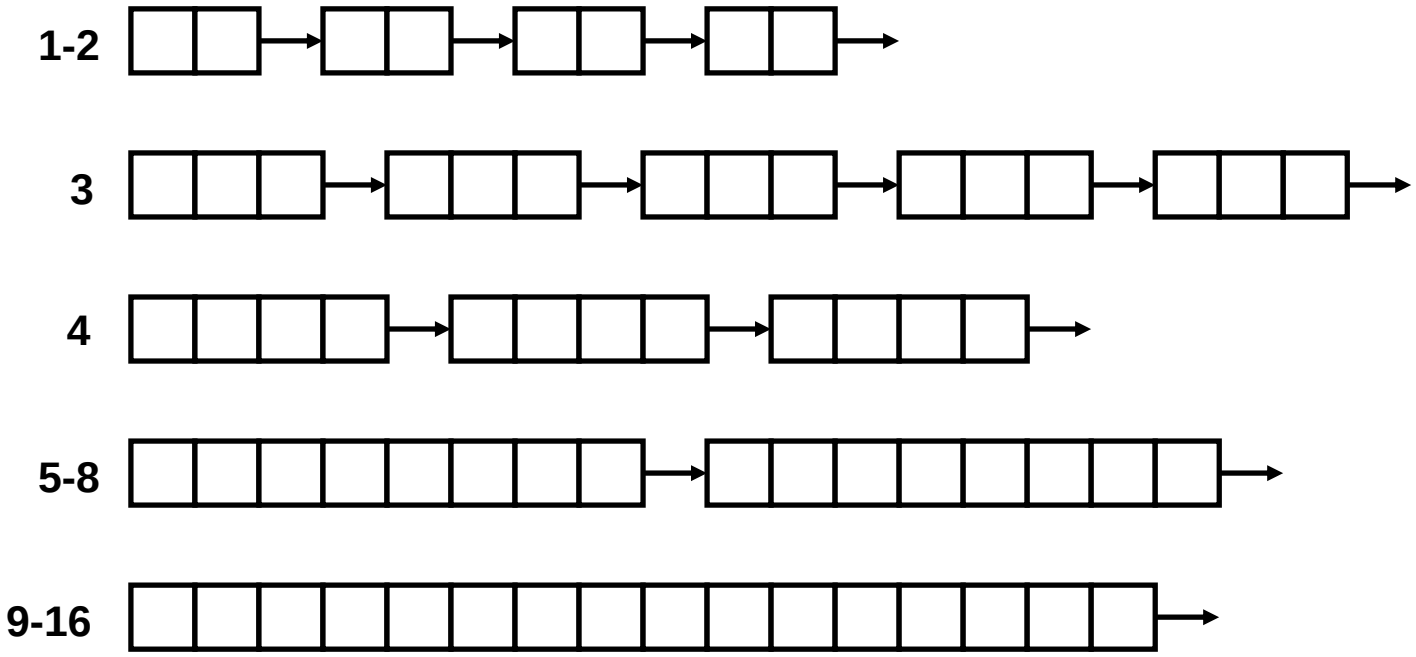
- *Method 2*: *Explicit list* among the free blocks using pointers within the free blocks



- *Method 3*: 종류별 free 리스트*Segregated free list*
  - Different free lists for different size classes
- *Method 4*: Blocks sorted by size
  - Can use a balanced tree (e.g. Red-Black tree) with pointers within each free block, and the length used as a key

크기별 관리

- Each *size class* has its own collection of blocks



- **Often have separate size class for every small size (2,3,4,...)**
- **For larger sizes typically have a size class for each power of 2**

간단한 크기별 관리

- Separate heap and free list for each size class
- No splitting
- To allocate a block of size n:
  - **If free list for size n is not empty,**
    - allocate first block on list (note, list can be implicit or explicit)
  - **If free list is empty,**
    - get a new page
    - create new free list from all blocks in page
    - allocate first block on list
  - **Constant time**
- To free a block:
  - **Add to free list**
  - **If page is empty, return the page for use by another size (optional)**
- Tradeoffs:
  - **Fast, but can fragment badly (왜?)**

크기별 리스트에서의 최적할당

- Array of free lists, each one for some size class
- To allocate a block of size n:
  - **Search appropriate free list for block of size m > n**
  - **If an appropriate block is found:**
    - Split block and place fragment on appropriate list (optional)
  - **If no block is found, try next larger class**
  - **Repeat until block is found**
- To free a block:
  - **Coalesce and place on appropriate list (optional)**
- Tradeoffs
  - **Faster search than sequential fits (i.e., log time for power of two size classes)**
  - **Controls fragmentation of simple segregated storage**
  - **Coalescing can increase search times**
    - Deferred coalescing can help