Windows Heap Exploitation

Angelboy

Windows Heap Exploitation

- Windows Memory Allocator
  - NT heap
    - BackEnd
      - Exploitation
    - FrontEnd
      - LowFragmentationHeap
      - Exploitation

Windows memory allocator

- Win 10 的 memory allocator 非常複雜，基本上主要有下列兩種
- Nt Heap
  - Default memory allocator
- SegmentHeap
  - Win10 中全新的 memory allocator 機制
  - 部分系統程式及 UWP 會使用

Windows memory allocator

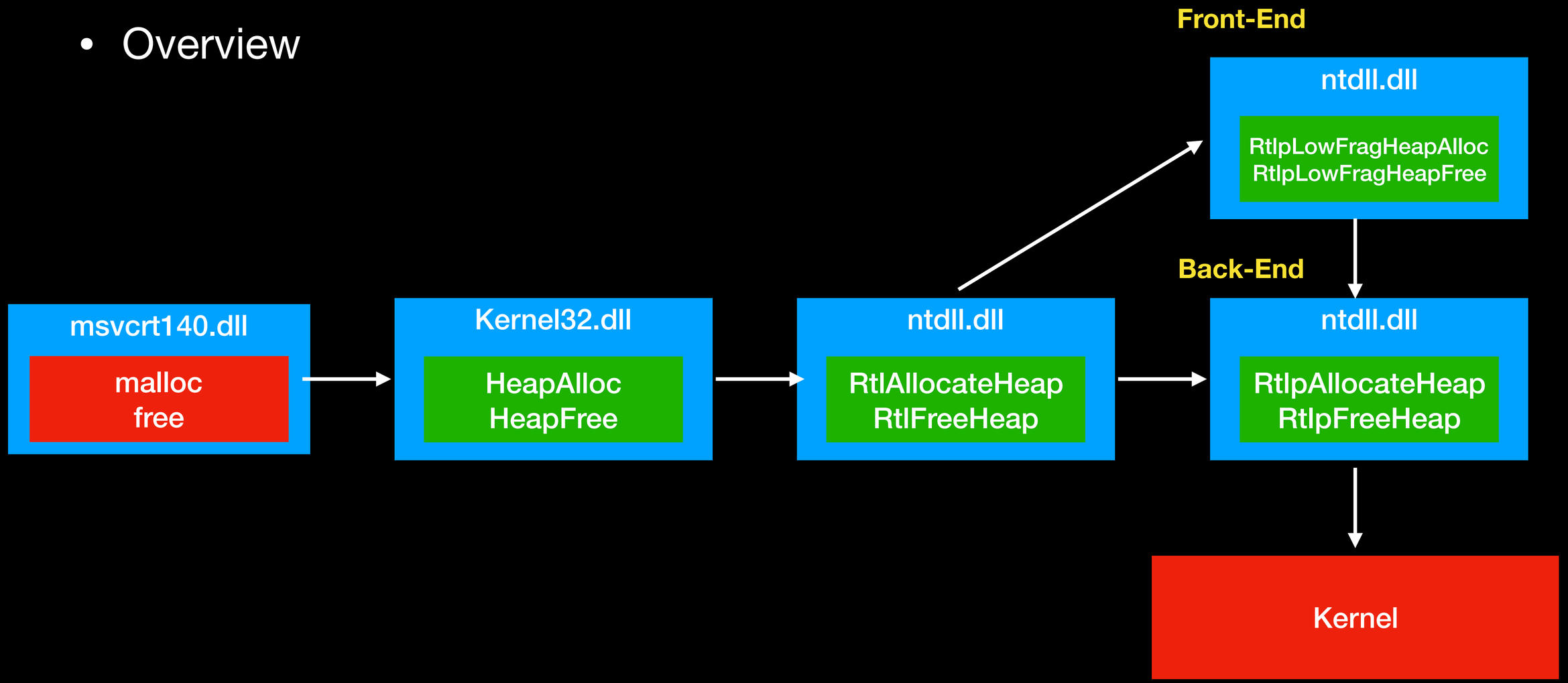
- Nt Heap 中又可分
  - 後端管理器 (Back-End)
  - 前端管理器 (Front-End)
    - LowFragmentationHeap
    - 基本上是為了不讓記憶體過於破碎，相同大小的記憶體區塊分配到一定數量後就會使用 (連續分配相同記憶體區塊 18 次)
    - Size <= 0x4000

Windows memory allocator

- 目前會以 win 10 (1809) 為主
  - OS build - 17763.379
  - 之後版本可能不一定完全正確
  - 結構常常改動很大，有個概念後，到新的發行版本，需要自己去追一下結構

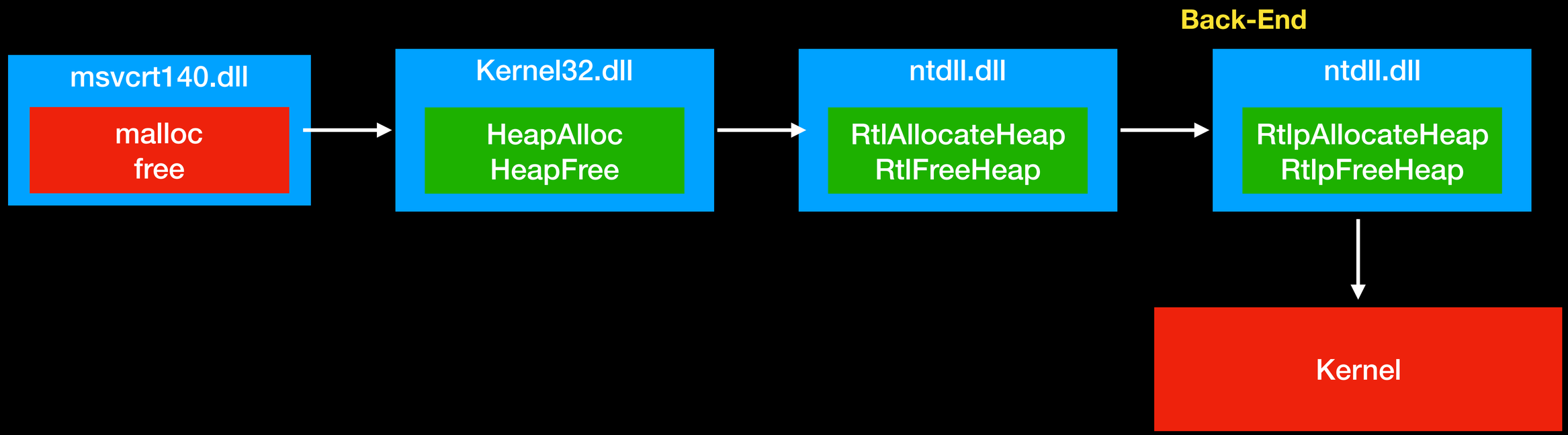
Windows memory allocator

- Overview



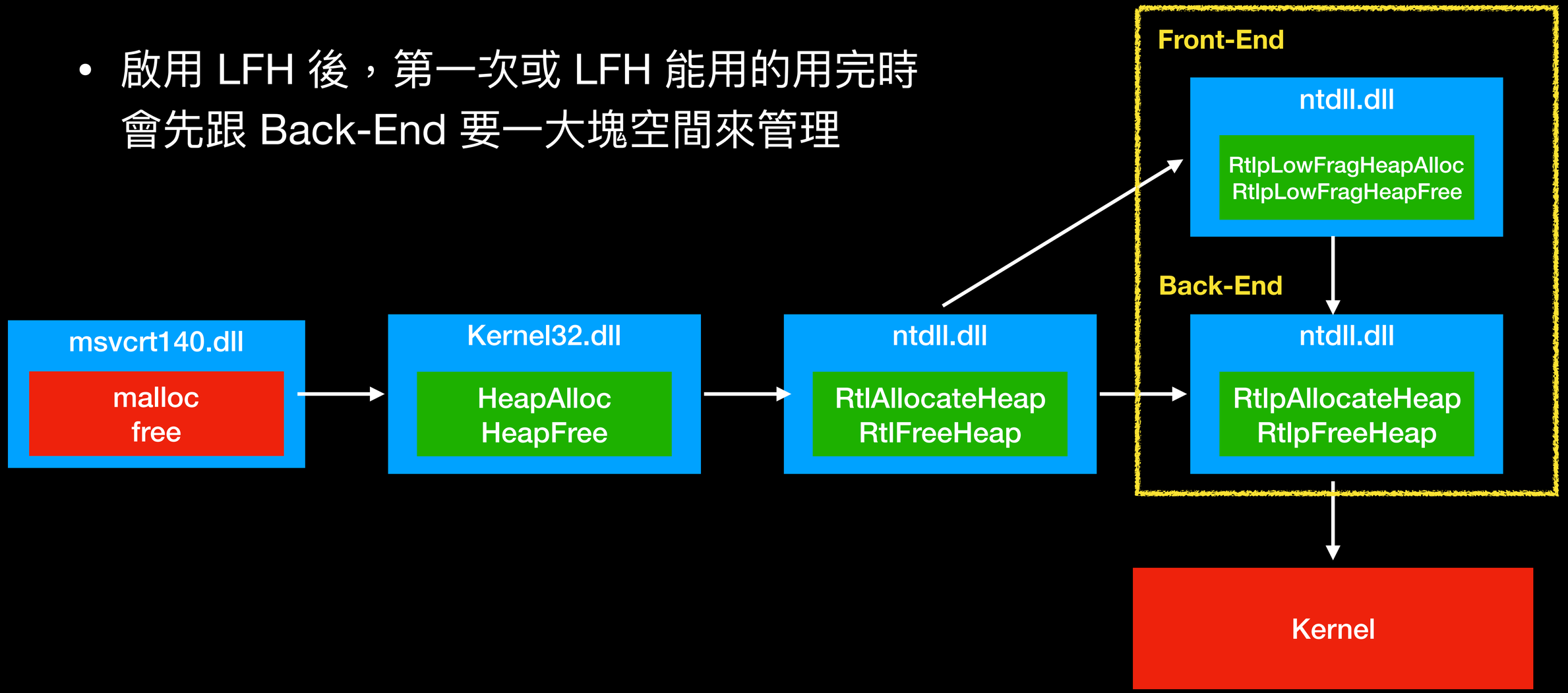
Windows memory allocator

- 在 LFH 沒啟用下，我們呼叫 malloc



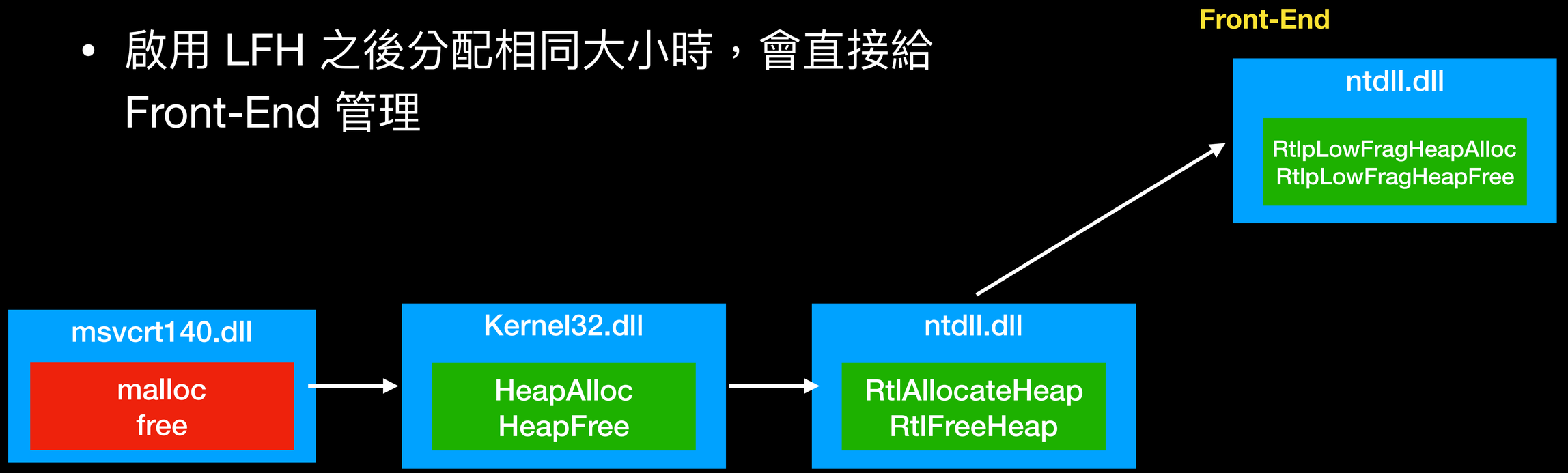
Windows memory allocator

- 啟用 LFH 後，第一次或 LFH 能用的用完時會先跟 Back-End 要一大塊空間來管理



Windows memory allocator

- 啟用 LFH 之後分配相同大小時，會直接給 Front-End 管理



Windows memory allocator

- HEAP 可分為
  - Process Heap
    - Default heap
      - 整個 process 共享，呼叫 api 時會用到
        - 會存在 _PEB 結構中
      - CRT 中的函式時也會用到
        - 存在 crt_heap

Windows memory allocator

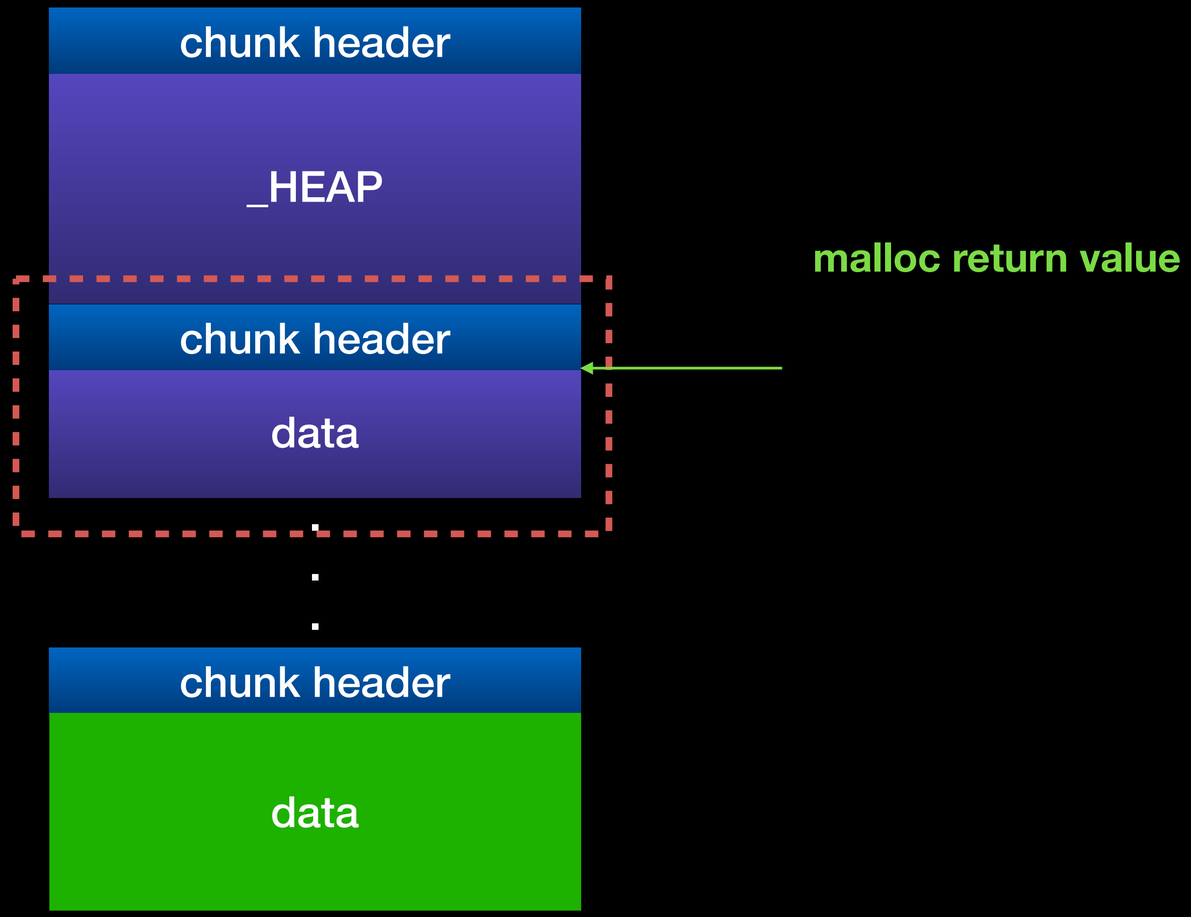
- HEAP 可分為
  - Private heap
    - 另外創建的 heap
      - HeapCreate

Windows memory allocator

- Core data structure
  - _HEAP_ENTRY (chunk)
    - Malloc 出來的基本結構
    - 在前後端分配器中，長相會不太一樣，但都是同一個名字

Windows memory allocator

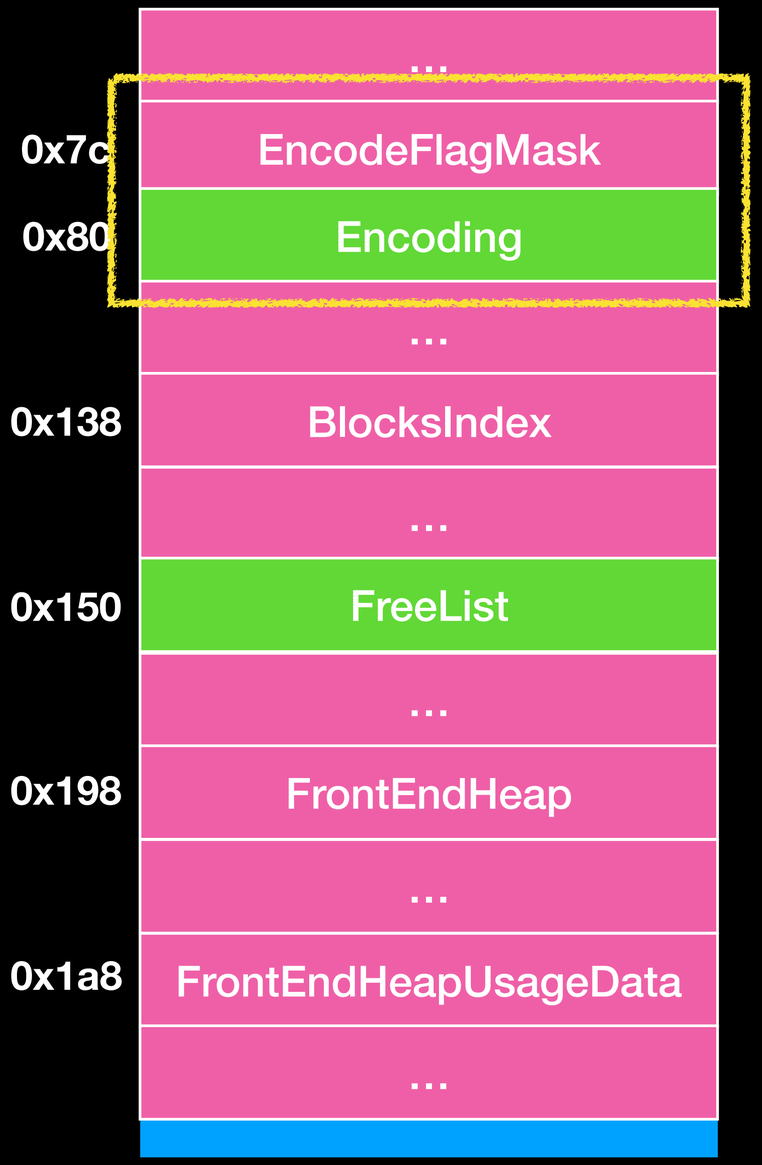
- _HEAP_ENTRY (chunk)



Windows memory allocator

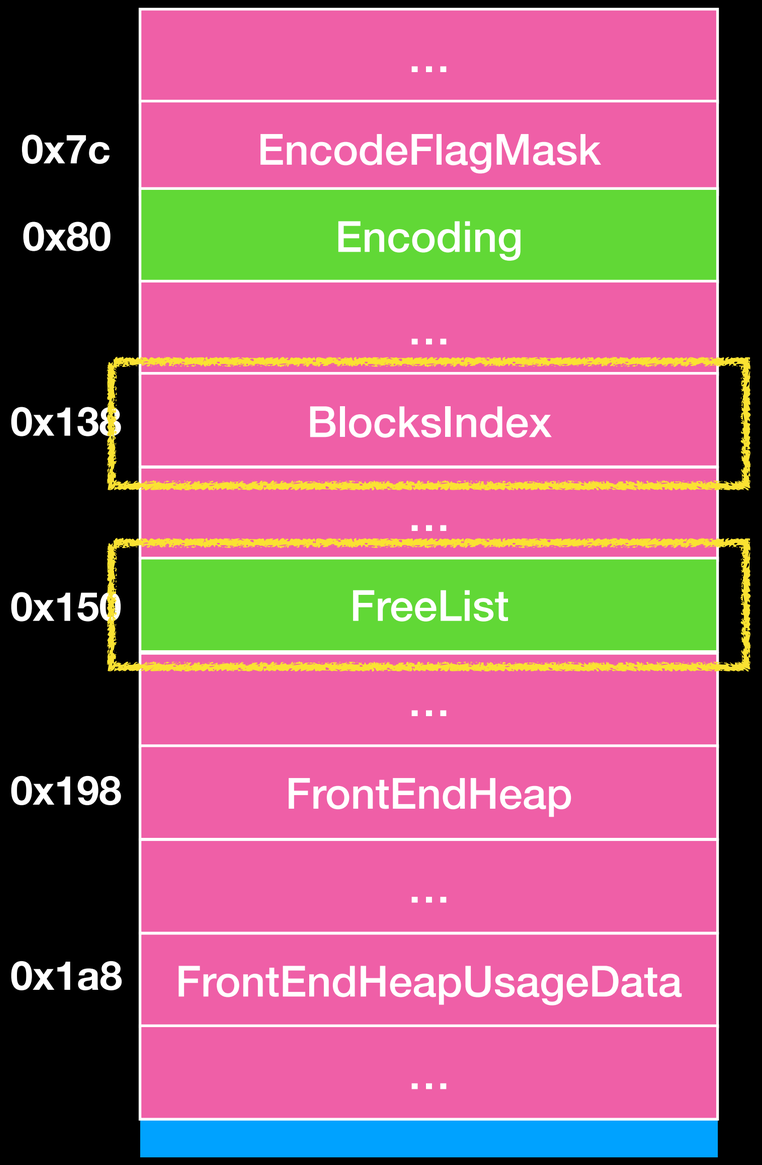
- Core data structure
  - _HEAP
    - 整個 memory allocator 中最核心的結構，用來管理該 heap
    - 每個 Heap 都會有一個 _HEAP 通常在 heap 開頭

Windows memory allocator



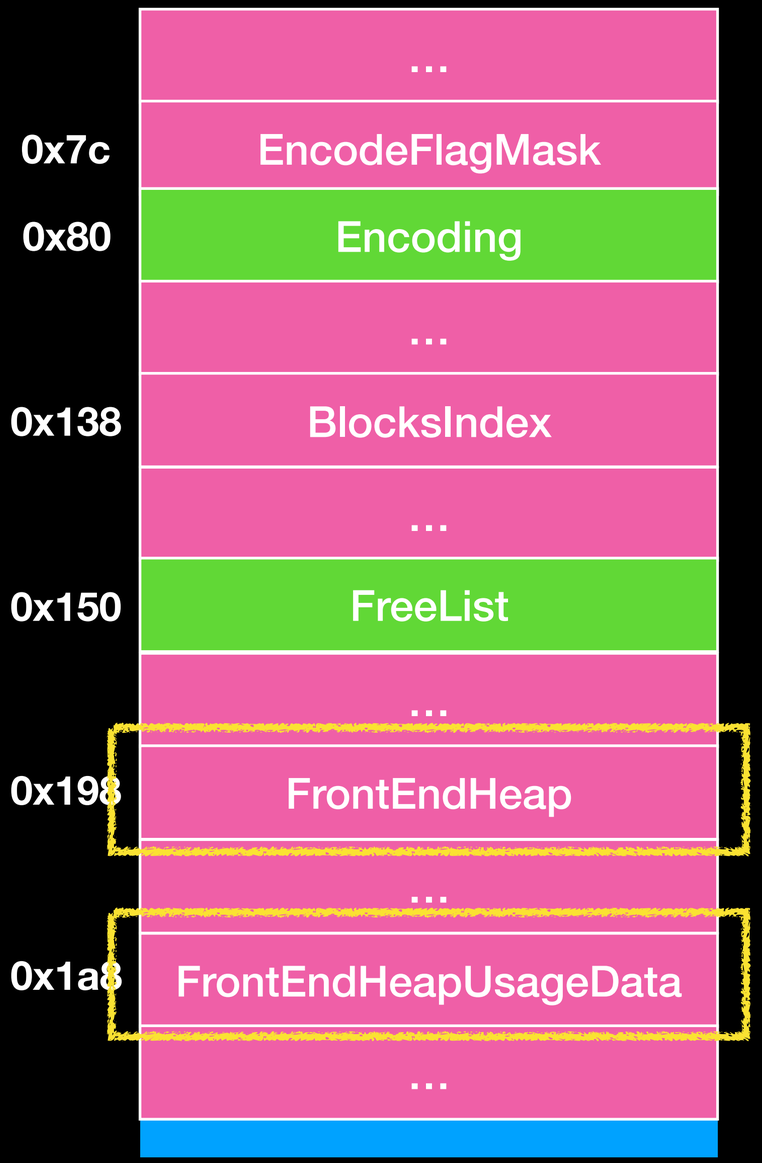
- _HEAP
  - EncodeFlagMask
    - Heap 初始化後會設置為 0x100000 在判斷是否要 encode 該 heap 中 chunk 的 header
  - Encoding (_HEAP_ENTRY)
    - 用來與 chunk header 做 xor 的 cookies

Windows memory allocator



- _HEAP
  - BlocksIndex (_HEAP_LIST_LOOKUP_)
    - Back-End 中，重要的結構之一，用來管理 Back-End 中的 chunk ，後面會詳細敘述
  - FreeList (_HEAP_ENTRY)
    - 串接 Back-End 中的所有 free chunk ，類似 unsorted bin
    - Sorted list

Windows memory allocator



- _HEAP
  - FrontEndHeap
    - 指向管理 FrontEnd 的 Heap 的結構
  - FrontEndHeapUsageData
    - 指向一個對應各大小 chunk 的陣列
    - 紀錄各種大小 chunk 使用次數，到達某個程度時會去 enable 該對應大小 chunk 的 Front-End allocator

Windows memory allocator

- Nt Heap
  - 後端管理器 (Back-End)
  - 前端管理器 (Front-End)

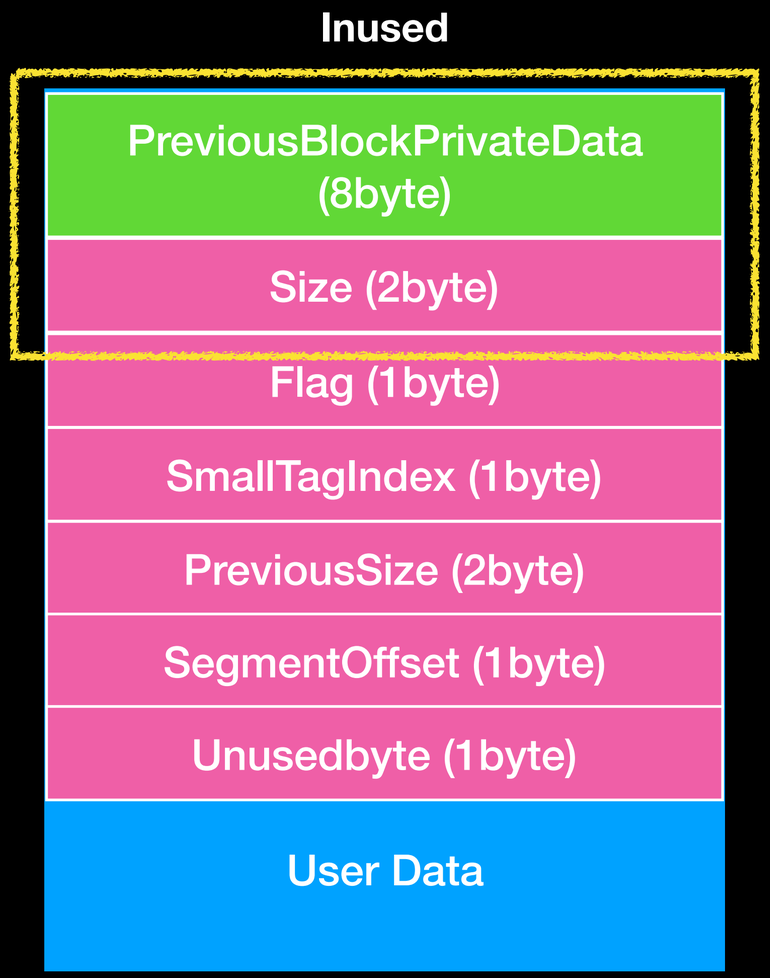
Nt heap

- Nt Heap
  - 後端管理器 (Back-End)
    - Data structure
    - 分配機制

Windows memory allocator

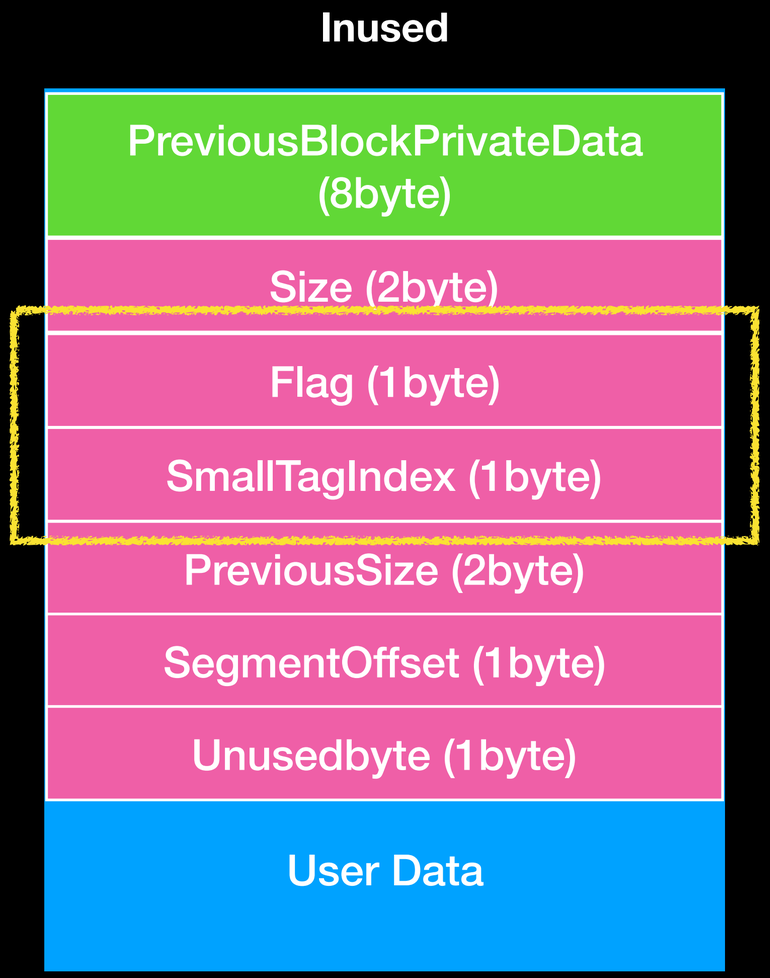
- _HEAP_ENTRY (chunk)
  - 分為三種狀況
    - Allocated chunk
    - Freed chunk
    - VirtualAlloc chunk

Nt heap



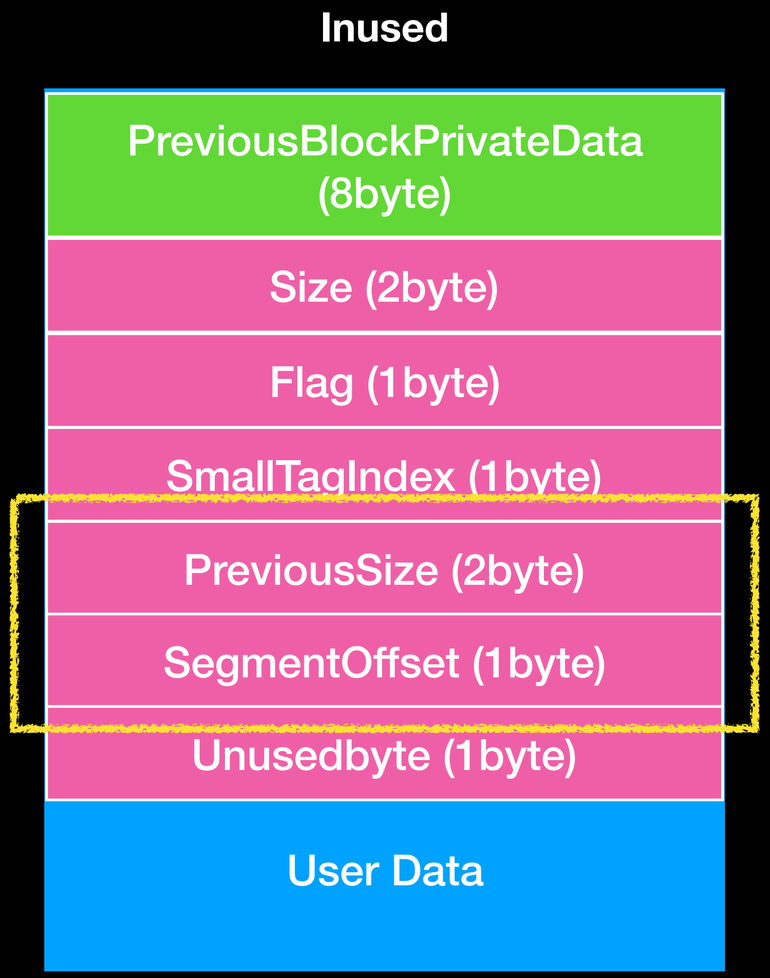
- _HEAP_ENTRY (chunk)
  - PreviousBlockPrivateData
    - 基本上可為前一塊 chunk 的 data，因為 chunk 必須對齊 0x10
  - Size
    - Chunk 的 size 但存入方式並不是直接存 size 而是存 (size >>4) 後的值

Nt heap



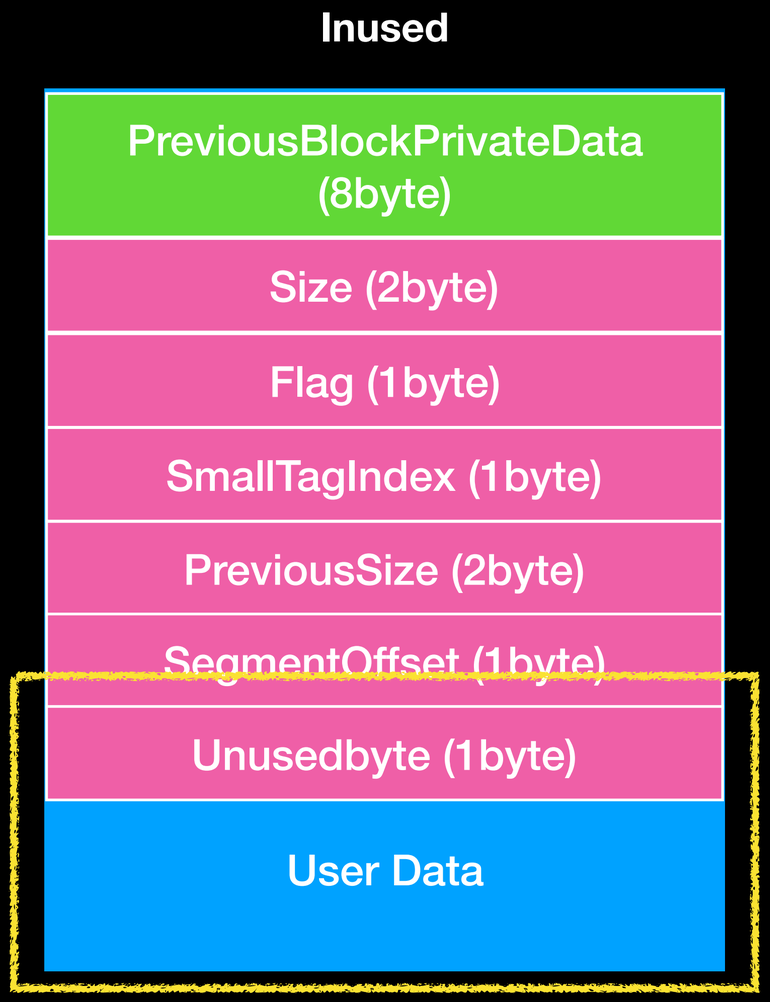
- _HEAP_ENTRY (chunk)
  - Flag
    - 表示該 chunk 是否 inused
  - SmallTagIndex
    - 前面三個 byte 的 Checksum(三個 byte 做 xor)，之後拿 header 出來時會驗證

Nt heap



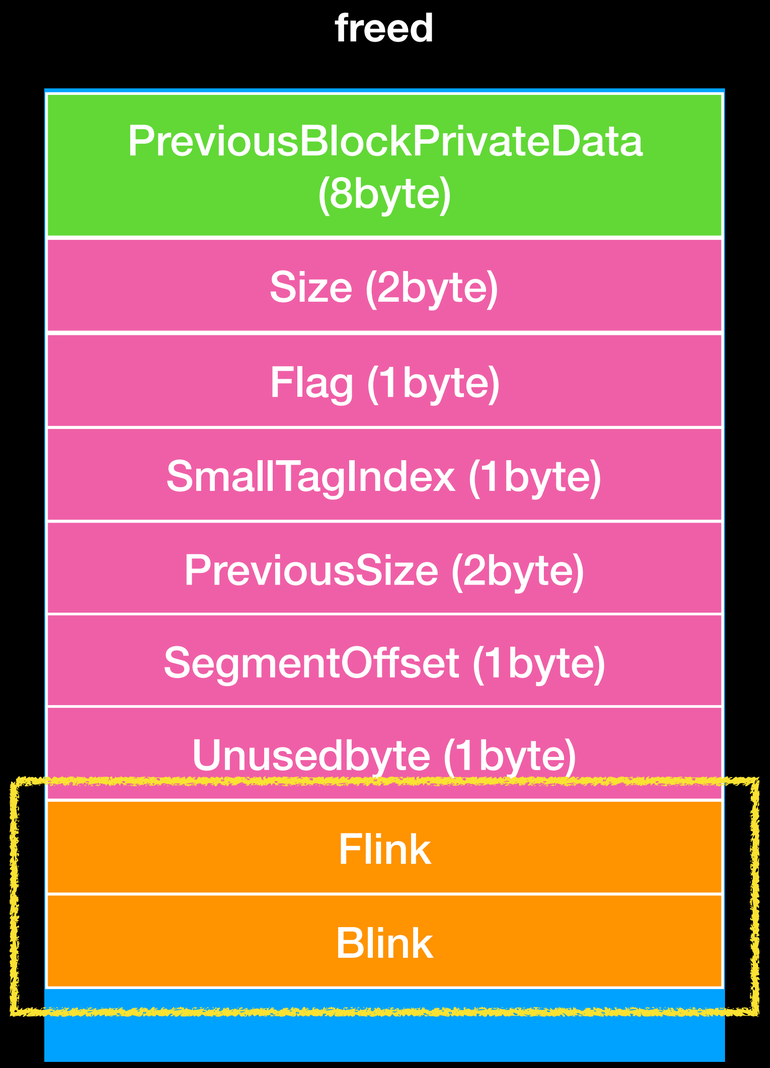
- _HEAP_ENTRY (chunk)
  - PreviousSize
    - 相鄰的前一塊 chunk 的 Size ，這邊一樣是 (Size >> 4) 過後的數值
  - SegmentOffset
    - 某些情況下用來找 segment 的

Nt heap



- _HEAP_ENTRY (chunk)
  - Unusedbyte
    - 紀錄 user malloc 後所剩 chunk 的空間
    - 可以用來判斷 chunk 的狀態判斷 chunk 是 FrontEnd or BackEnd
  - User Data
    - User 所使用的區塊

Nt heap

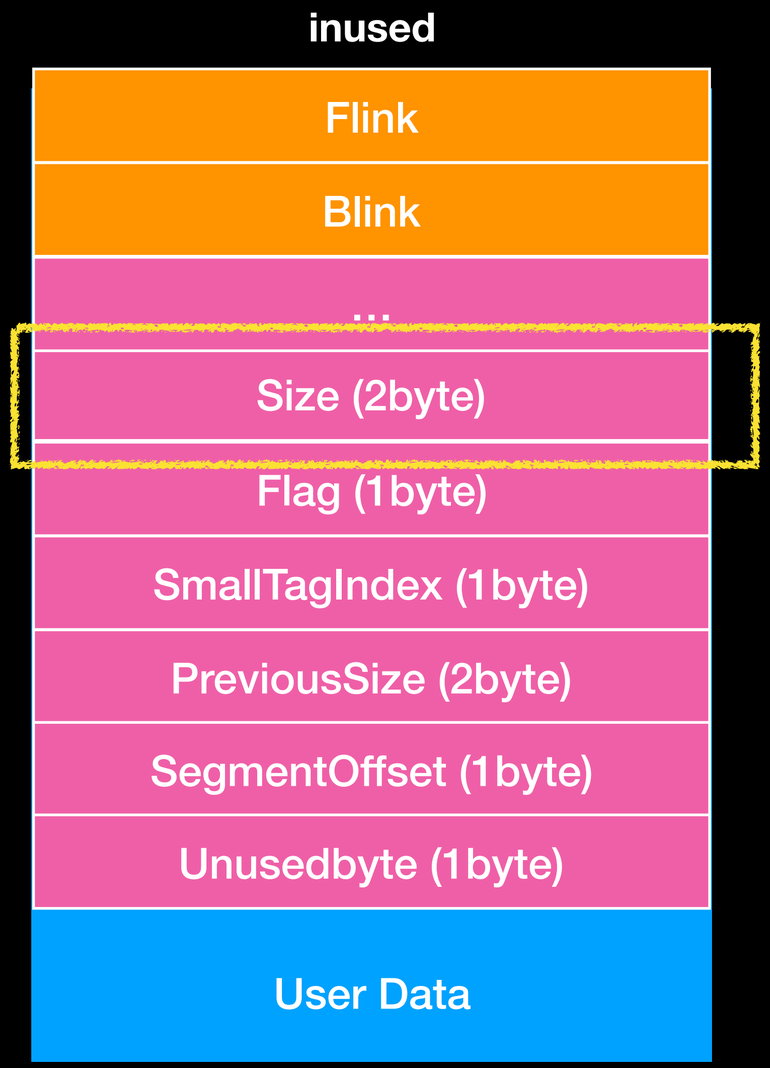


- _HEAP_ENTRY (chunk)
  - Flink
    - 指向 linked list 中下一塊 chunk
  - Blink
    - 指向 linked list 中上一塊 chunk
  - Unusedbyte
    - 恆為 0

Nt heap

- _HEAP_VIRTUAL_ALLOC_ENTRY (mmap chunk)
  - Flink
    - 指向下一塊 VirtualAlloc 出來的 chunk
  - Blink
    - 指向上一塊 VirtualAlloc 出來的 chunk

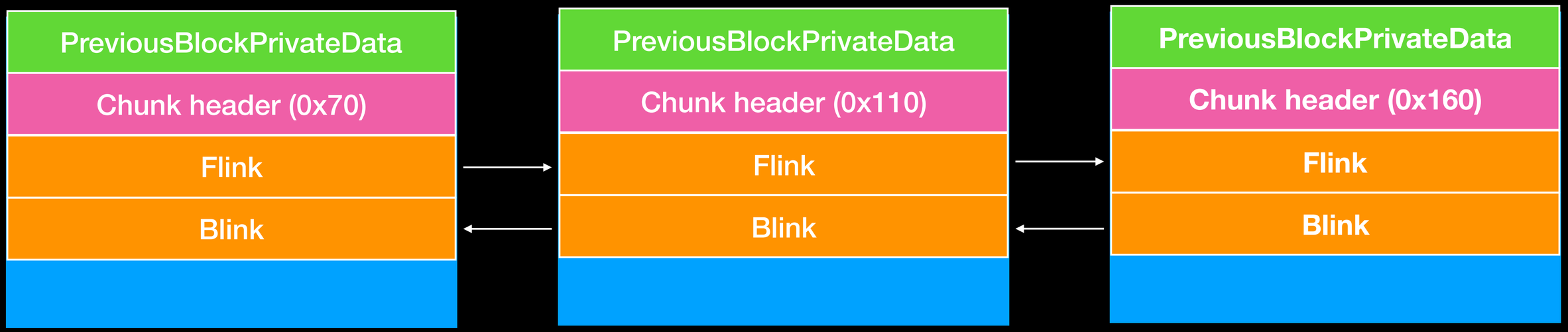
Nt heap



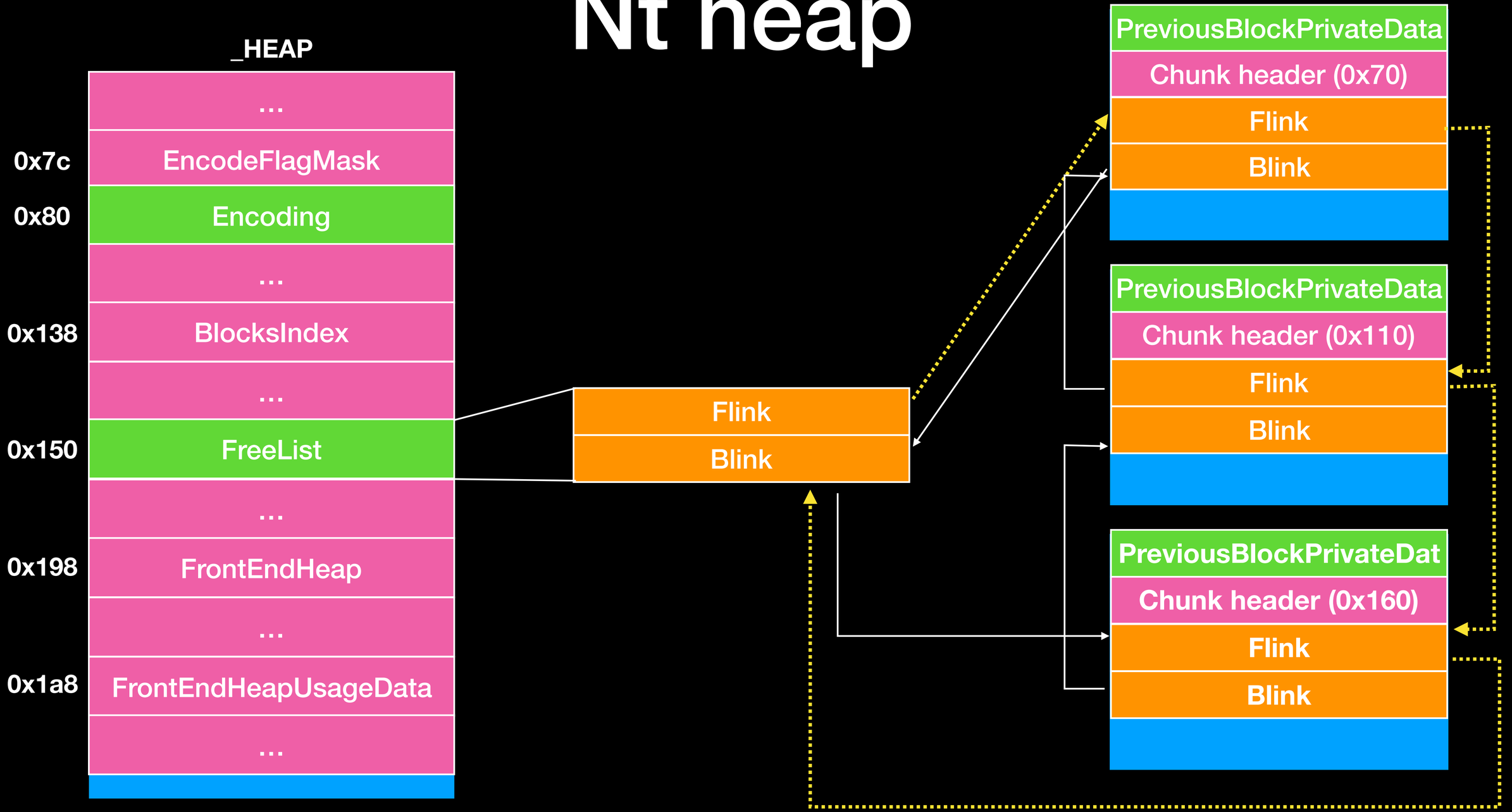
- _HEAP_VIRTUAL_ALLOC_ENTRY (mmap chunk)
  - Size
    - 這邊的 size 指的是 unused size，且沒有 shift 過
  - Unusedbyte
    - 恆為 4

Nt heap

- FreeLists (_HEAP_ENTRY)
  - 在 Free 完一塊 chunk 後，會將該 chunk 放到 FreeLists 中，並會依照大小決定插在 FreeLists 中的位置



Nt heap


_HEAP
...
0x7c
EncodeFlagMask
0x80
Encoding
...
0x138
BlocksIndex
...
0x150
FreeList
...
0x198
FrontEndHeap
...
0x1a8
FrontEndHeapUsageData
...
Flink
Blink
PreviousBlockPrivateData
Chunk header (0x70)
Flink
Blink
PreviousBlockPrivateData
Chunk header (0x110)
Flink
Blink
PreviousBlockPrivateDat
Chunk header (0x160)
Flink
Blink

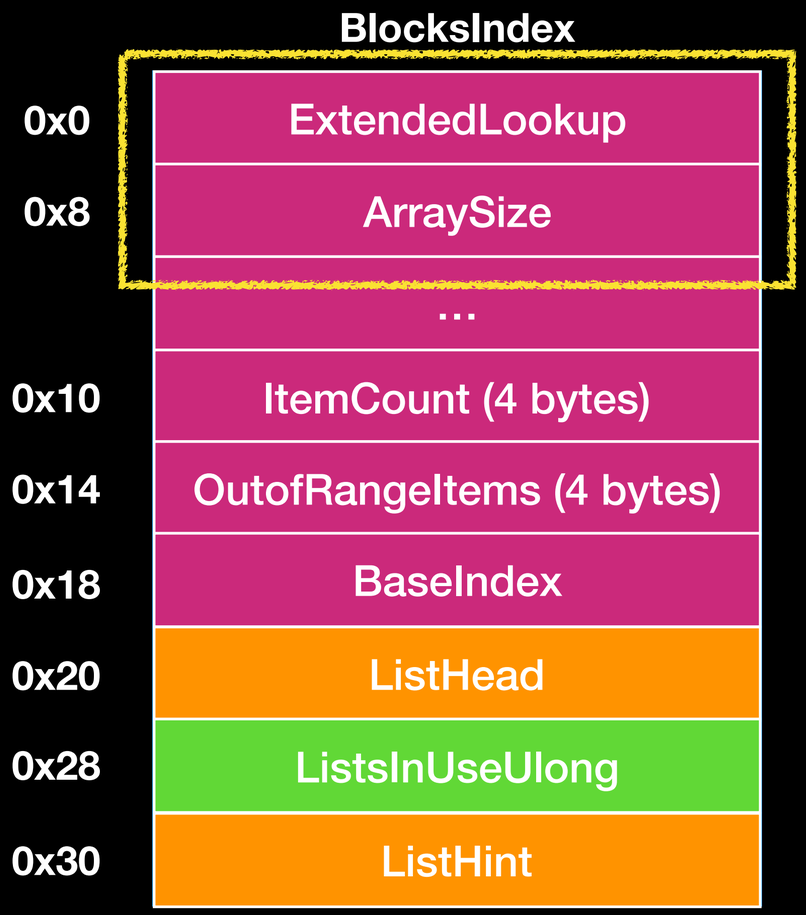
Nt heap

- Remark
  - 關於 header encoding
    - 所有的 chunk 都會經過 xor 過，在存 chunk header 時，會將整個 header^(_HEAP->Encoding) 在存入
    - decode 時會驗證 check sum 確保沒被改掉
      - 驗證方式為 encode_header^(_HEAP->Encoding) 後的值，前 3 byte xor 後和第 4 byte 比對

Nt heap

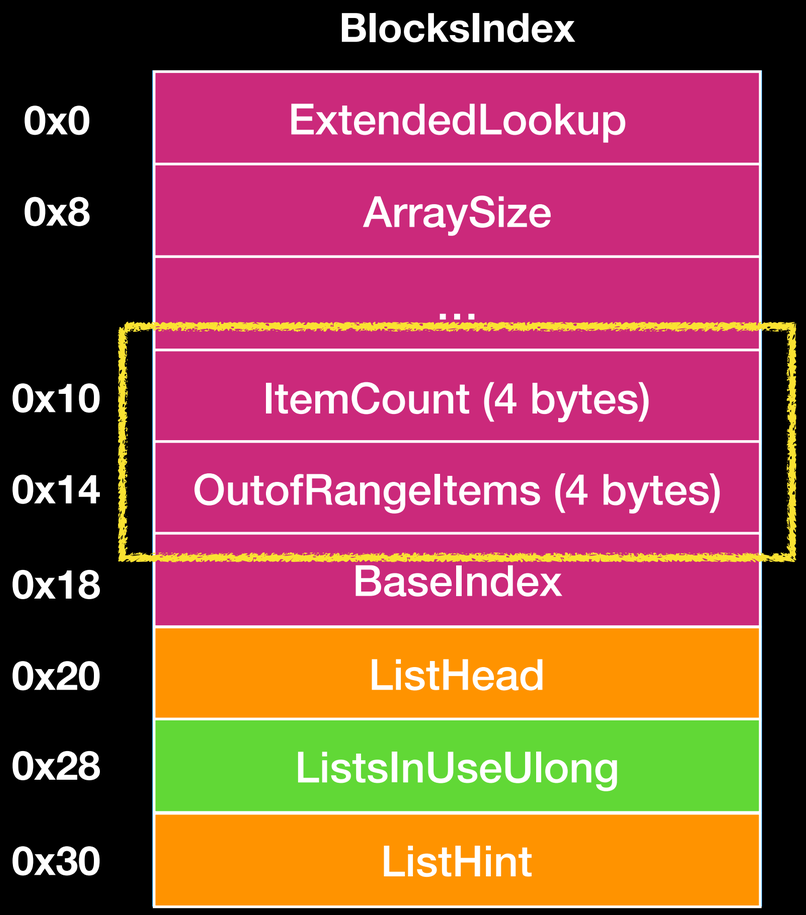
- BlocksIndex (_HEAP_LIST_LOOKUP)
  - 主要用來管理各種不同大小的 freed chunk 的結構，方便在配置記憶體時，能快速的找到適合的 chunk

Nt heap



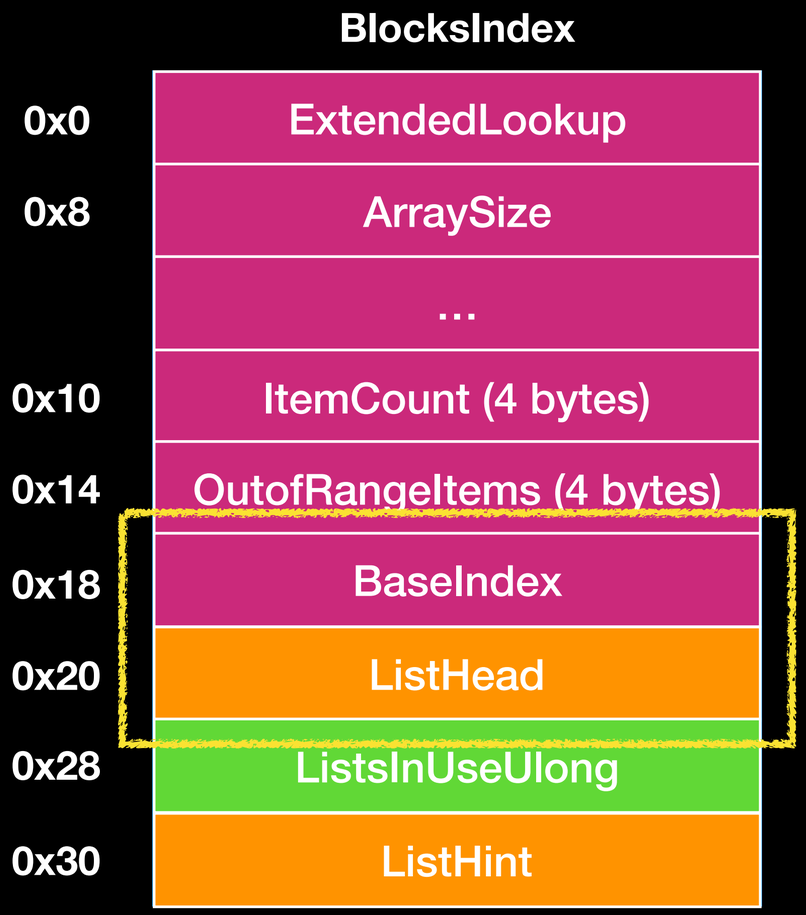
- BlocksIndex (_HEAP_LIST_LOOKUP)
  - ExtendedLookup (_HEAP_LIST_LOOKUP)
    - 指向下一個 ExtendedLookup
    - 通常下一個會管理更大塊 chunk 所用的結構
  - ArraySize
    - 該結構會管理的，最大 chunk 的大小，通常為 0x80 (實際上是 0x800)

Nt heap



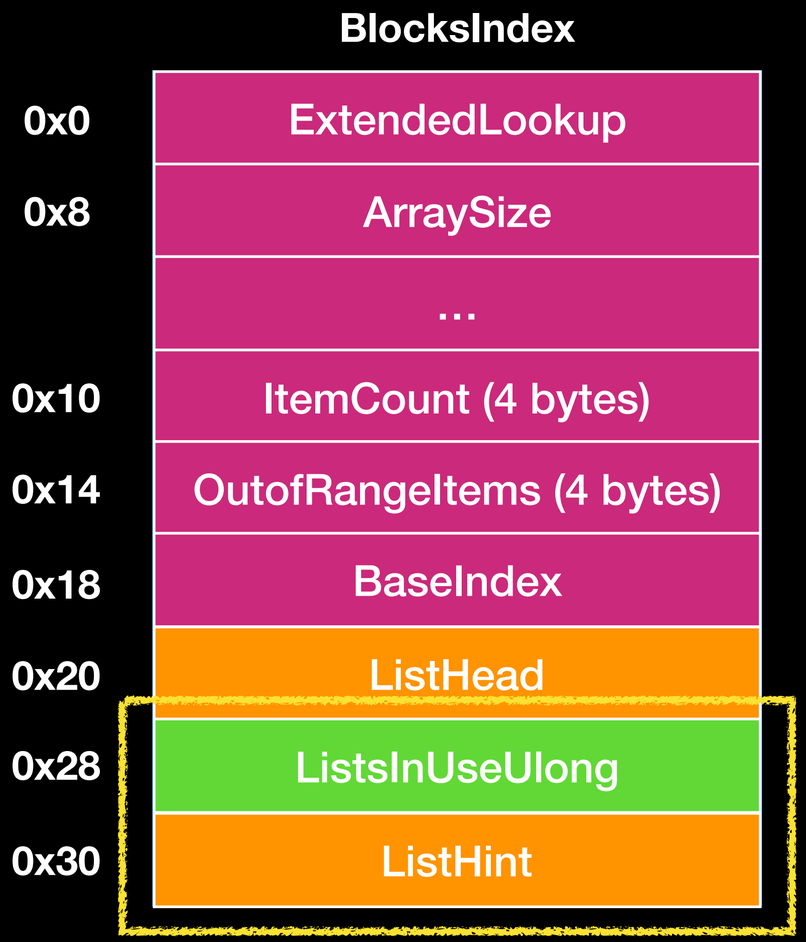
- BlocksIndex (_HEAP_LIST_LOOKUP)
  - ItemCount
    - 目前該結構所管理的 chunk 數
  - OutofRangeItems
    - 超出該結構所管理大小的 chunk 的數量

Nt heap

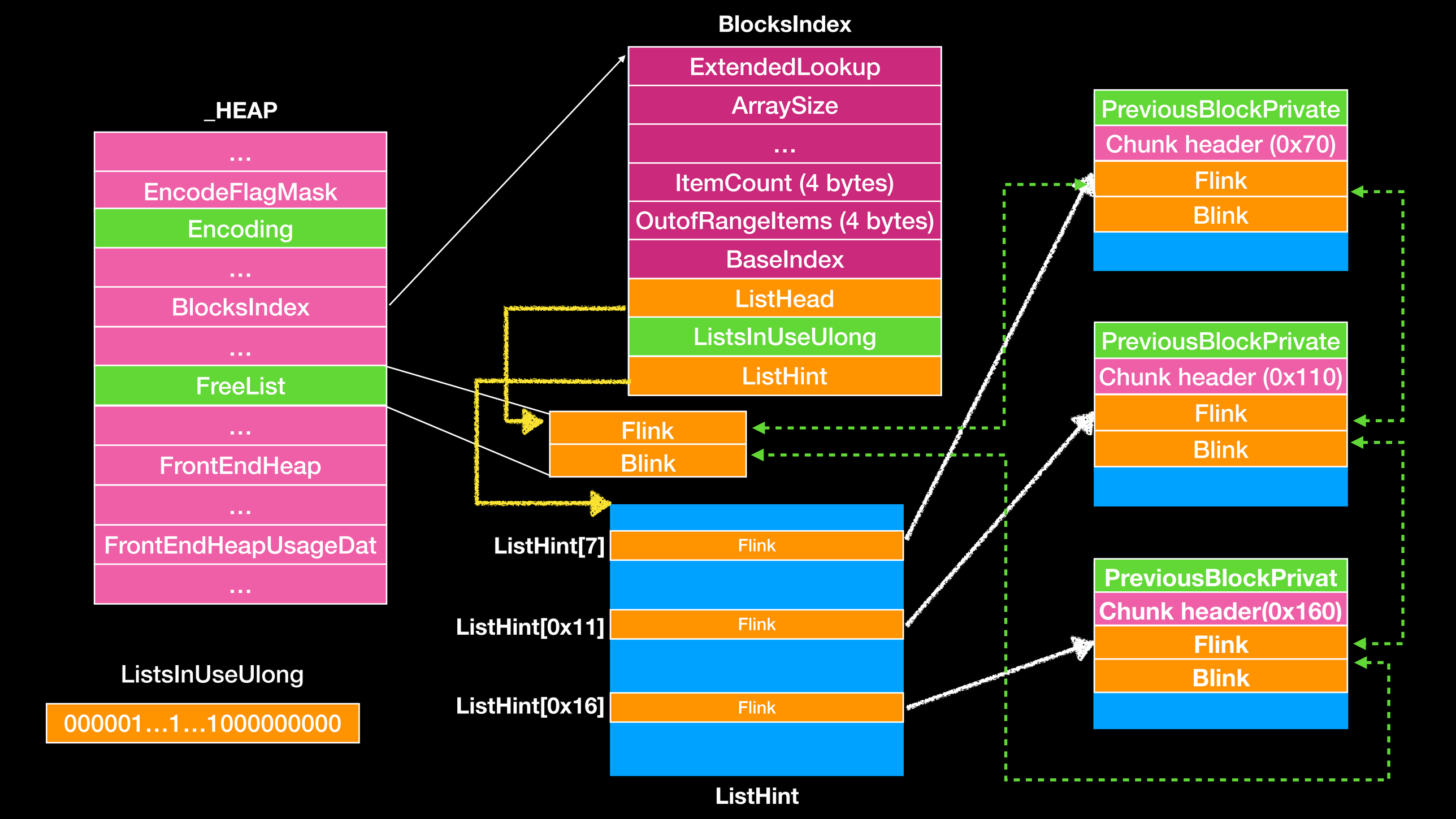


- BlocksIndex (_HEAP_LIST_LOOKUP)
  - BaseIndex
    - 該結構所管理的 chunk 的起始 index，用來從 ListHint 找適合大小的 free chunk 用的，下一個 BlocksIndex 將從這個 index 的最大值作為 BaseIndex
  - ListHead (_HEAP_ENTRY)
    - FreeList 的 Head

Nt heap



- BlocksIndex (_HEAP_LIST_LOOKUP)
  - ListsInUseUlong
    - 用在判斷 ListHint 中是否有適合大小的 chunk，是一個 bitmap
  - ListHint
    - 重要結構，用來指向相對應大小的 chunk array
    - 其目的就在於更快速找到適合大小的 chunk 大小為 0x10 為一個間隔


BlocksIndex
ExtendedLookup
ArraySize
…
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
_HEAP
…
EncodeFlagMask
Encoding
…
BlocksIndex
…
FreeList
…
FrontEndHeap
…
FrontEndHeapUsageDat
…
Flink
Blink
ListHint[7]
Flink
ListHint[0x11]
Flink
ListHint[0x16]
Flink
ListHint
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
PreviousBlockPrivate
Chunk header (0x110)
Flink
Blink
PreviousBlockPrivat
Chunk header(0x160)
Flink
Blink
ListsInUseUlong
000001…1…1000000000

Nt heap

- Nt Heap
  - 後端管理器 (Back-End)
    - Data structure
    - 分配機制

Nt heap

- Allocate (RtlpAllocateHeap)
  - 基本上分為三種
    - Size <= 0x4000
    - 0x4000 < size <= 0xff000
    - Size > 0xff000

Nt heap

- Allocate (RtlpAllocateHeap)
  - Size <= 0x4000
    - 基本上主要分配都會在 RtlpAllocateHeap
      - 接著會去看該 size 對應到的 FrontEndHeapStatusBitmap 使否有啟用 LFH
      - 沒有的話會對對應到的 FrontEndHeapUsageData 加上 0x21
        - 並且檢查值是否超過 0xff00 或者 & 0x1f 後超過 0x10
          通過條件就會啟用 LFH

Nt heap

- Allocate (RtlpAllocateHeap)
  - Size <= 0x4000
    - 接下來會先看對應到的 ListHint 是否有值，會以 ListHint 中的 chunk 為優先
    - 如果有適合剛好的 chunk 在 ListHint 上則移除 ListHint，並且會看該 chunk 的 Flink 大小是否剛好也是同樣 size
      - 是的話就將 LintHint 填上 Flink
      - 不是則清空
      - 最後則 unlink 該 chunk 把這塊 chunk 從 linked list 中移除返還給使用者，並將 header xor 回去

Nt heap

- Allocate (RtlpAllocateHeap)
  - Size <= 0x4000
    - 如果沒有剛好適合的
      - 從比較大的 ListHint 中找，有找到比較大的後如果該 chunk 將會從 ListHint 則移除之，並看下一塊 chunk size 有沒有剛好大小，有則補上，並且 unlink 將該 chunk 移除 freelist
      - 最後將該 chunk 做切割，剩下的大小重新加入 Freelist ，如果可以放進 ListHint 就會放進去
      - 然後回傳切好的 chunk 給使用者，這邊也會對切割後跟回傳使用者的 chunk header xor

Nt heap

- Allocate (RtlpAllocateHeap)
  - Size <= 0x4000
    - 如果 FreeList 中都沒有
      - 嘗試 ExtendHeap 加大 heap 空間
      - 再從 extend 出來的 chunk 拿
      - 接著後面一樣切割，放回 ListHint，xor header

Nt heap

- Allocate (RtlpAllocateHeap)
  - 0x4000 < size <= 0xff000
    - 除了沒有對 LFH 相關的操作外，其餘都跟 0x4000 一樣

Nt heap

- Allocate (RtlpAllocateHeap)
  - Size > 0xff000 (VirtualMemoryThreshold << 4)
    - 直接使用 ZwAllocateVirtualMemory
    - 類似 mmap 直接給一大塊，並且會插入 _HEAP->VirtualAllocdBlocks 這個 linked list 中
      - 這個 linked list 是串接該 Heap VirtualAllocate 出來的區段用的

Nt heap

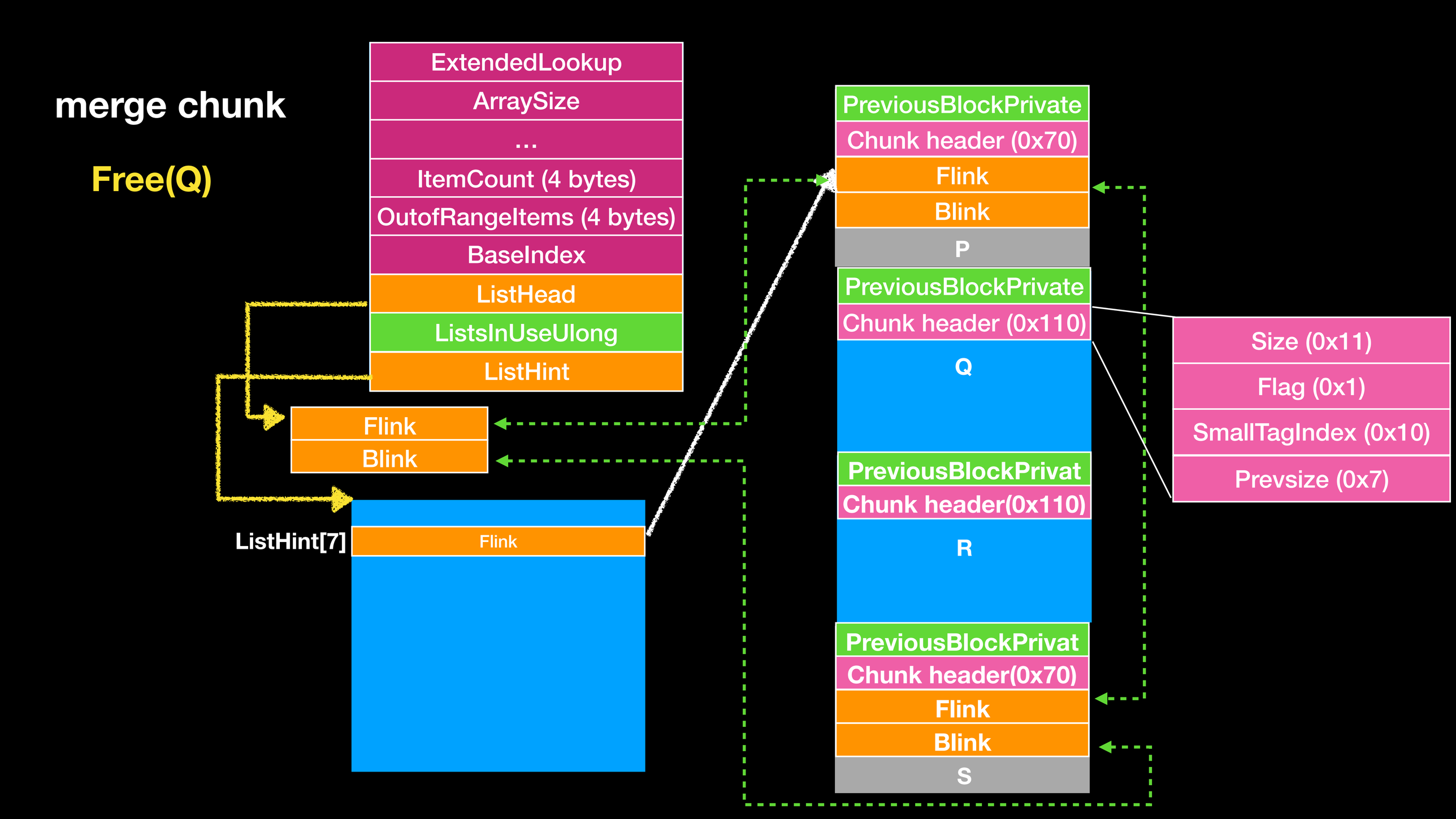
- Free (RtlpFreeHeap)
  - 可分為
    - Size <= 0xff000
    - Size > 0xff000

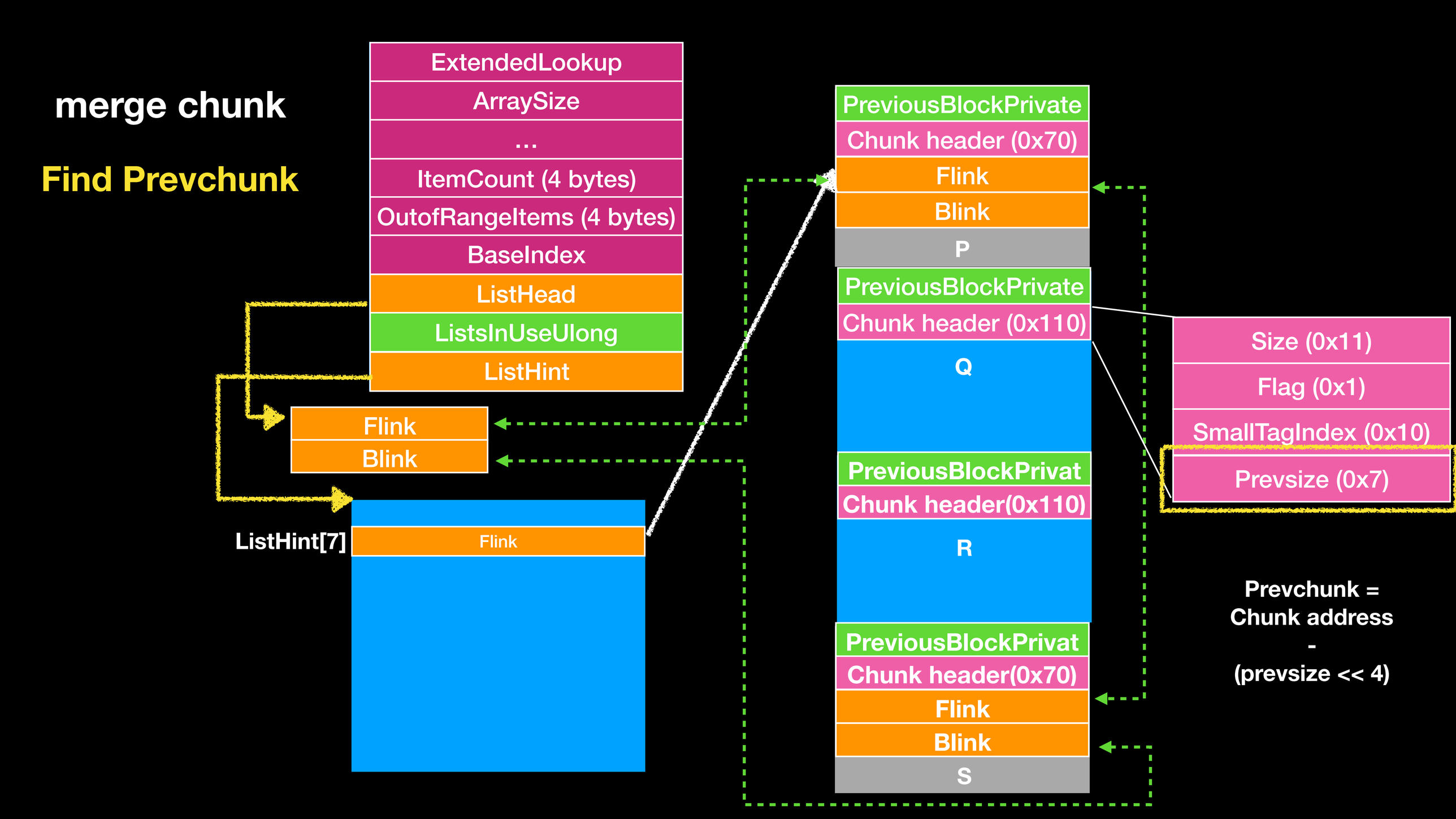
Nt heap

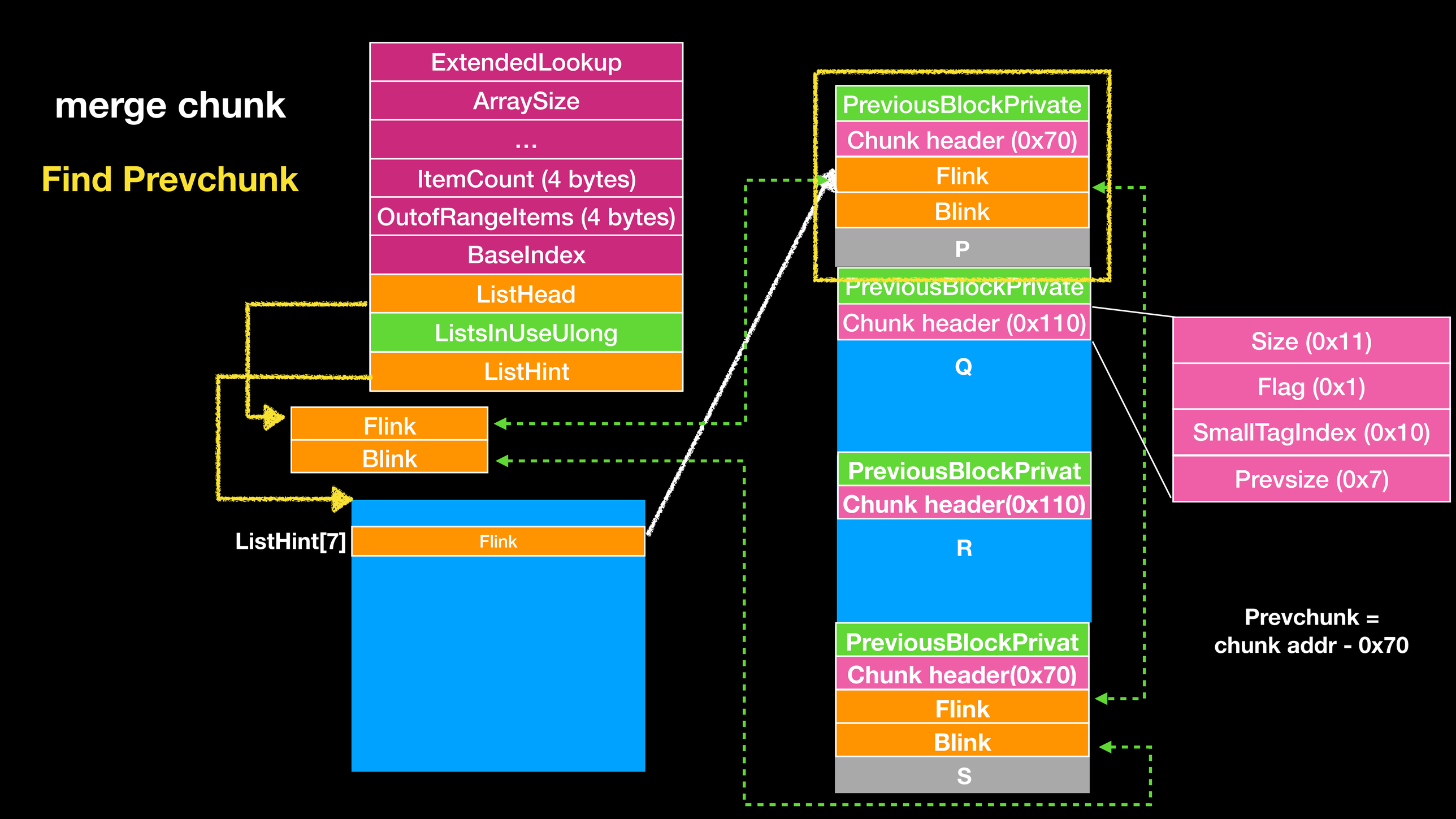
- Free (RtlpFreeHeap)
  - Size <= 0xff000
    - 會先檢查 alignment、利用 unused byte 判斷該 chunk 狀態
      - 如果是非 LFH 下，會對對應到的 FrontEndHeapUsageData 減 1
      - 接著會判斷前後的 chunk 是否為 freed ，是的話就合併
        - 此時會把可以合併的 chunk 做 unlink ，並從 ListHint 移除
        - 移除方式與前面相同，看看下一塊是不是同樣大小，是的話補上 ListHint

Nt heap

- Free (RtlpFreeHeap)
  - Size <= 0xff000
    - 合併完後， update size & prevsize ，然後會看看是不是最前跟最後，是就插入，不行插入就從 ListHint 中插入，並且 update ListHint，插入時也會對 linked list 做檢查
      - 但這檢查不會 abort
        - 其原因主要因為不做 unlink 寫入

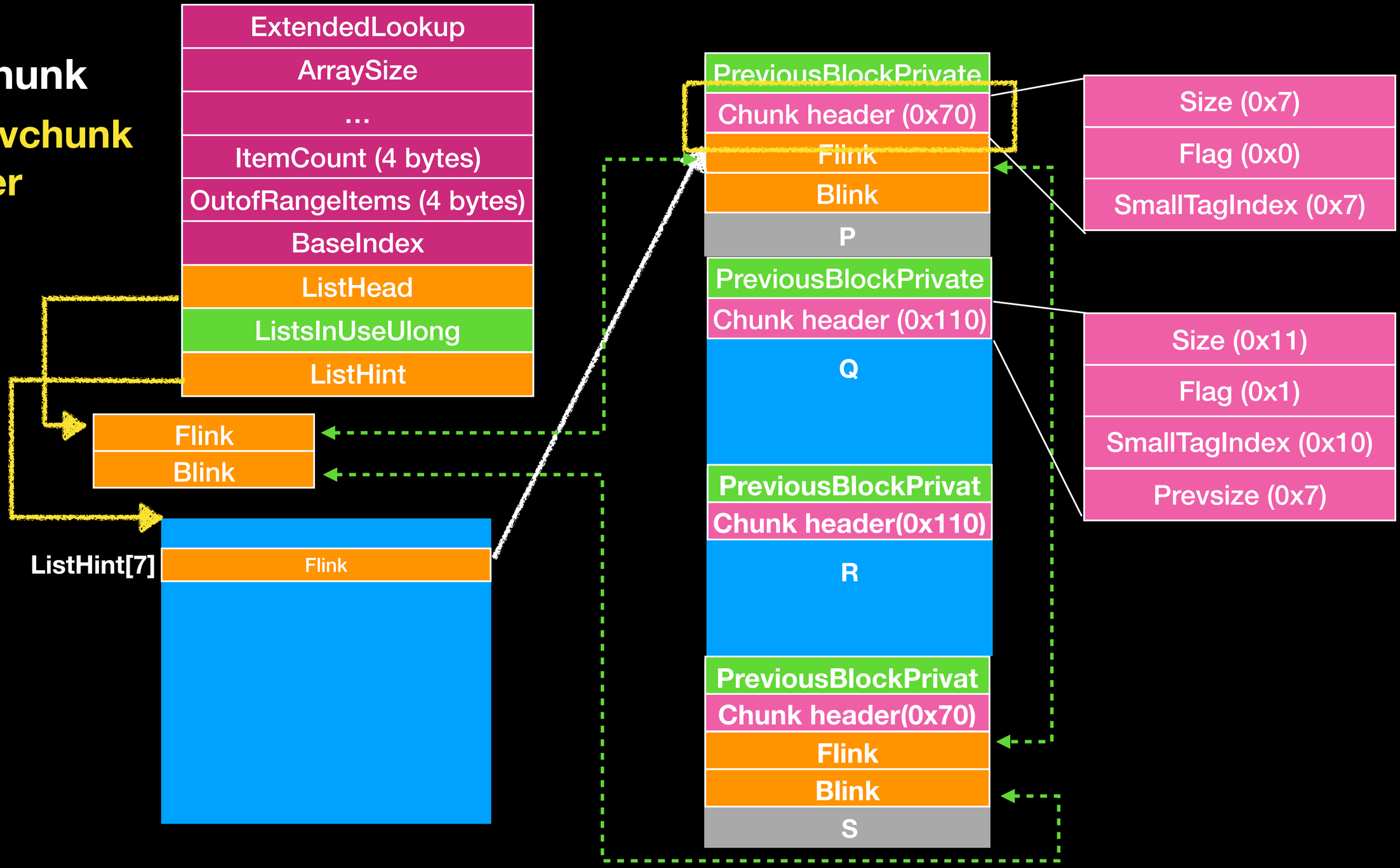

merge chunk
Free(Q)
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
PreviousBlockPrivate
Chunk header (0x110)
Q
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)

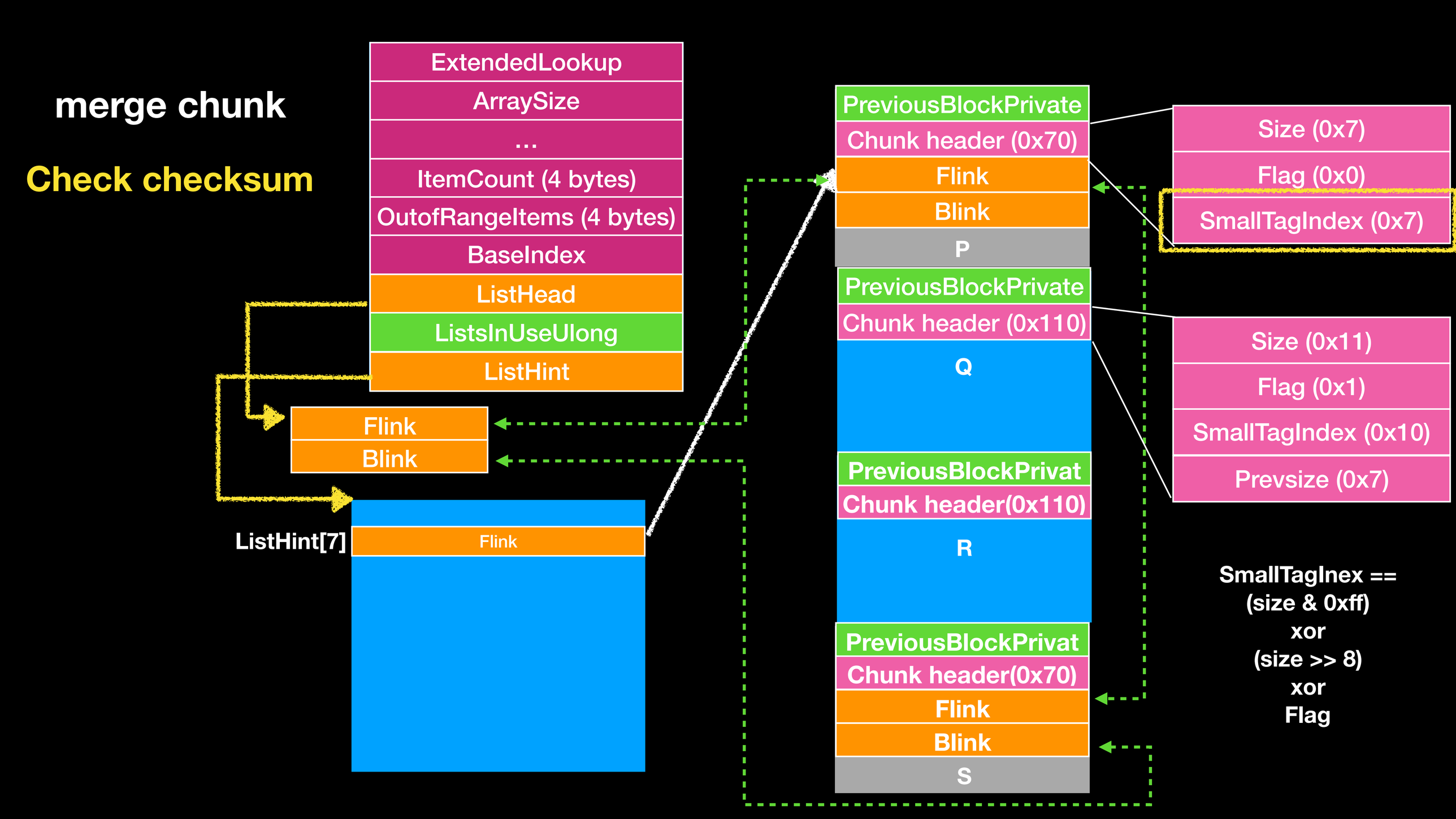

merge chunk
Find Prevchunk
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
PreviousBlockPrivate
Chunk header (0x110)
Q
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
Prevchunk =
Chunk address
-
(prevsize << 4)


merge chunk
Find Prevchunk
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
PreviousBlockPrivate
Chunk header (0x110)
Q
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
Prevchunk =
chunk addr - 0x70

merge chunk

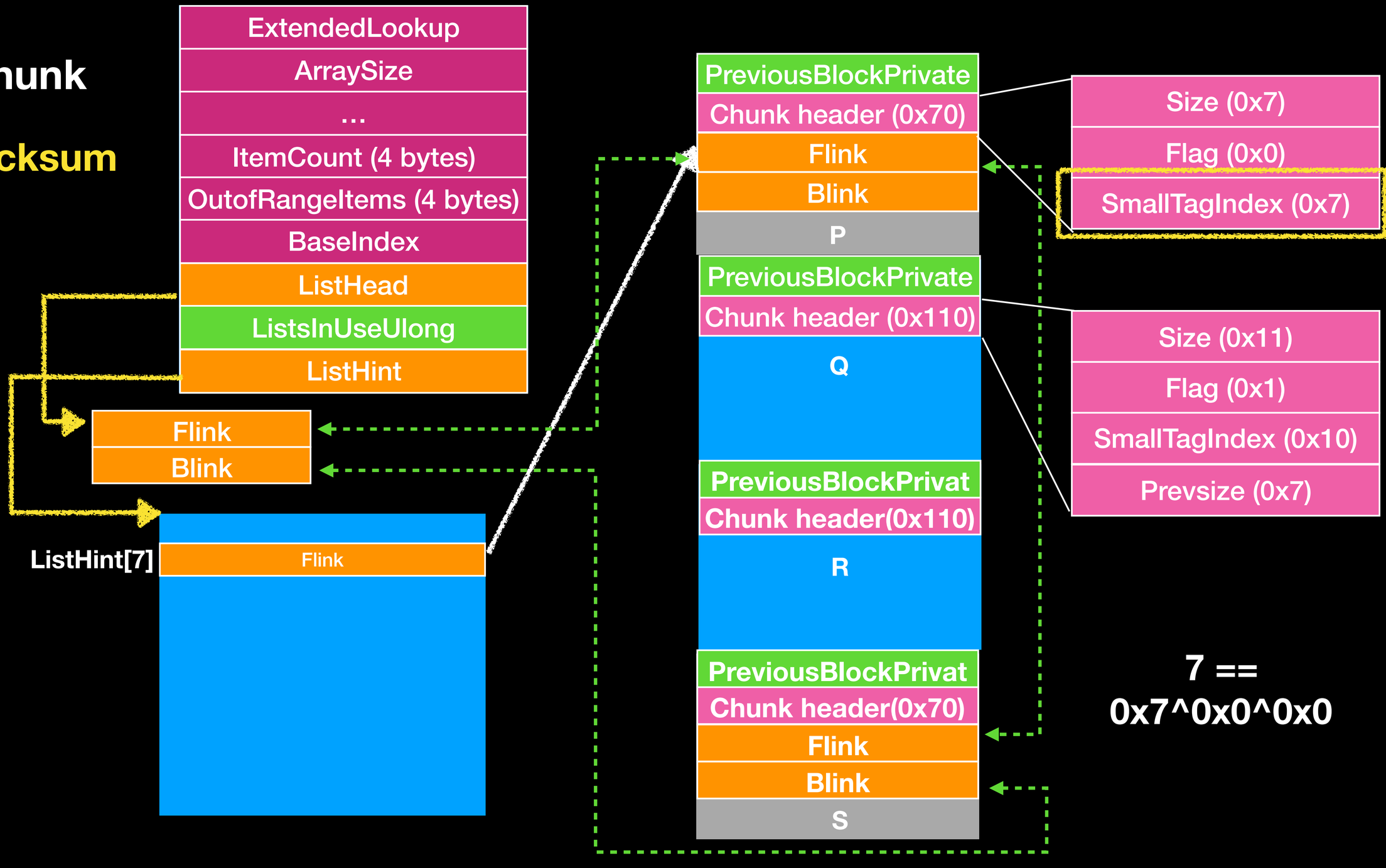
Decode prevchunk header

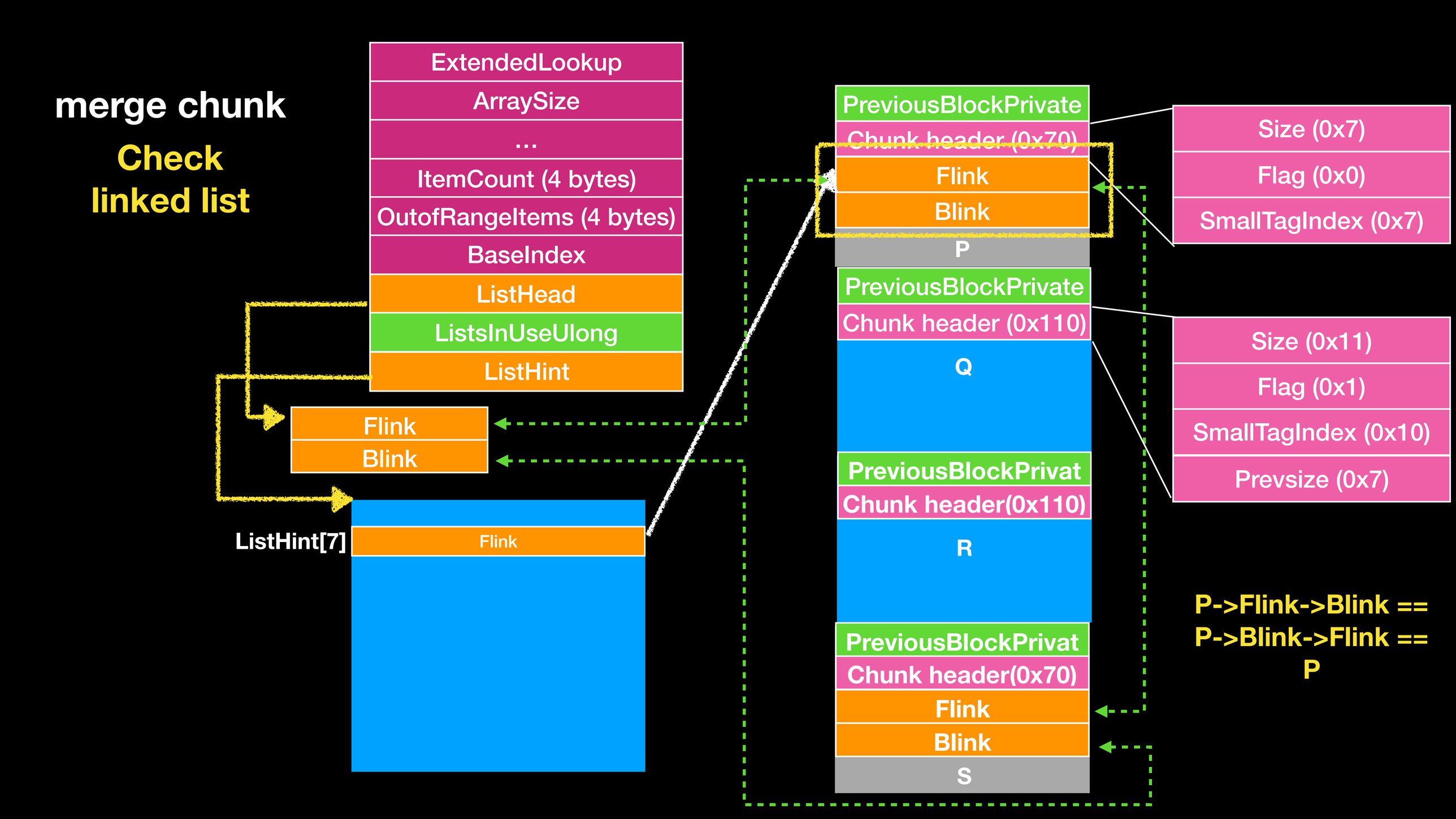


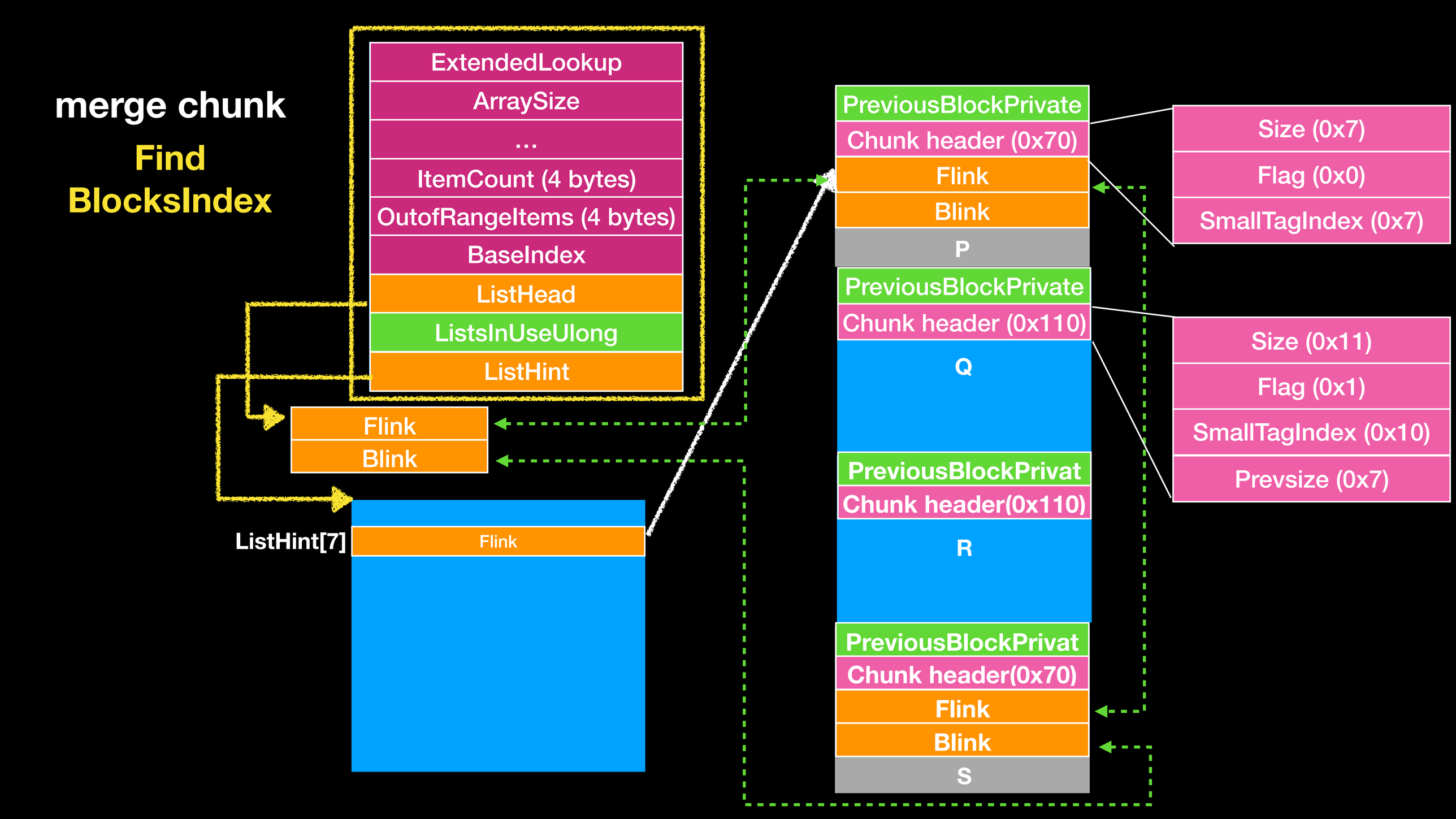

merge chunk
Check checksum
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
PreviousBlockPrivate
Chunk header (0x110)
Q
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
SmallTagInex ==
(size & 0xff)
xor
(size >> 8)
xor
Flag

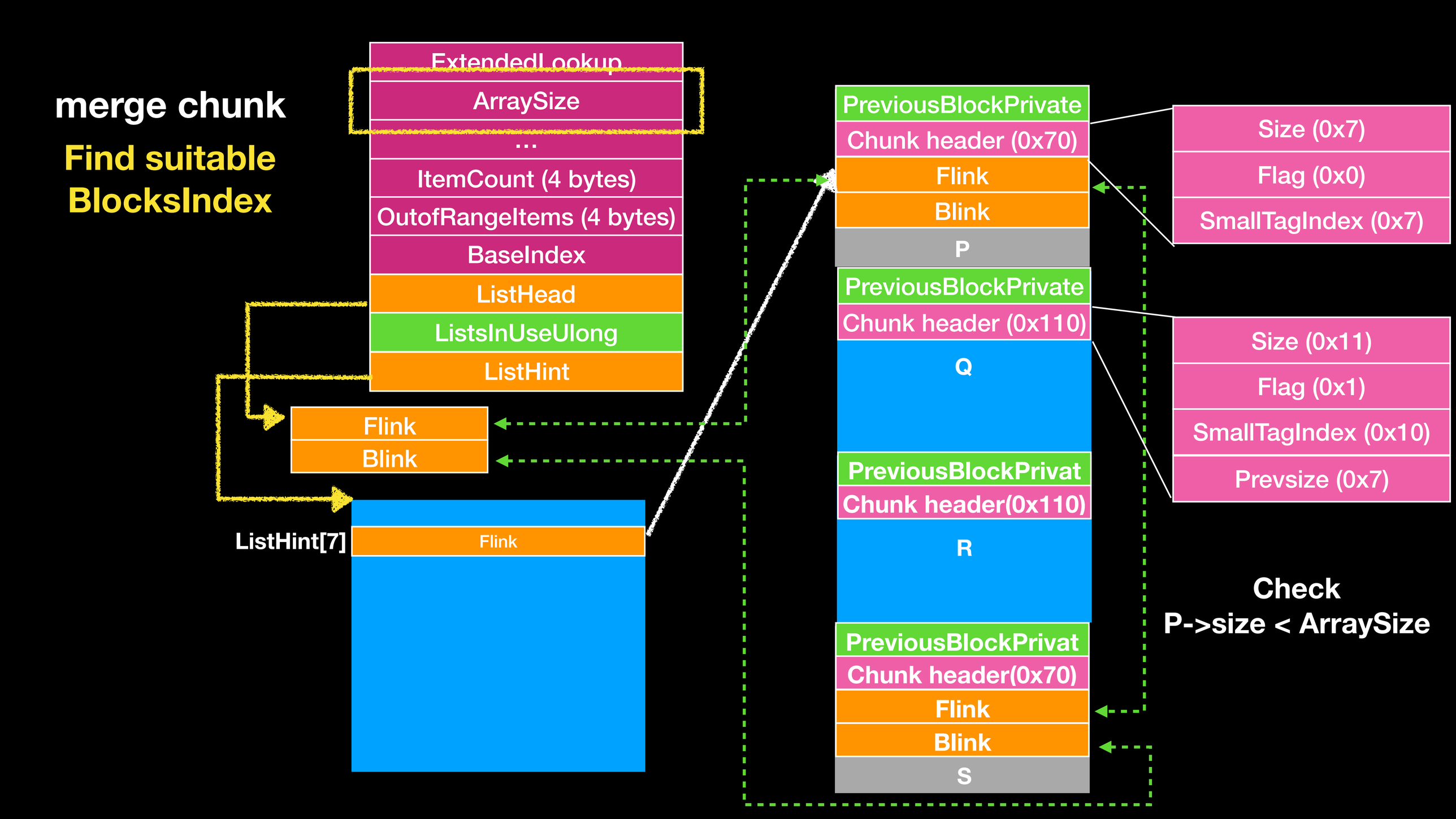
merge chunk

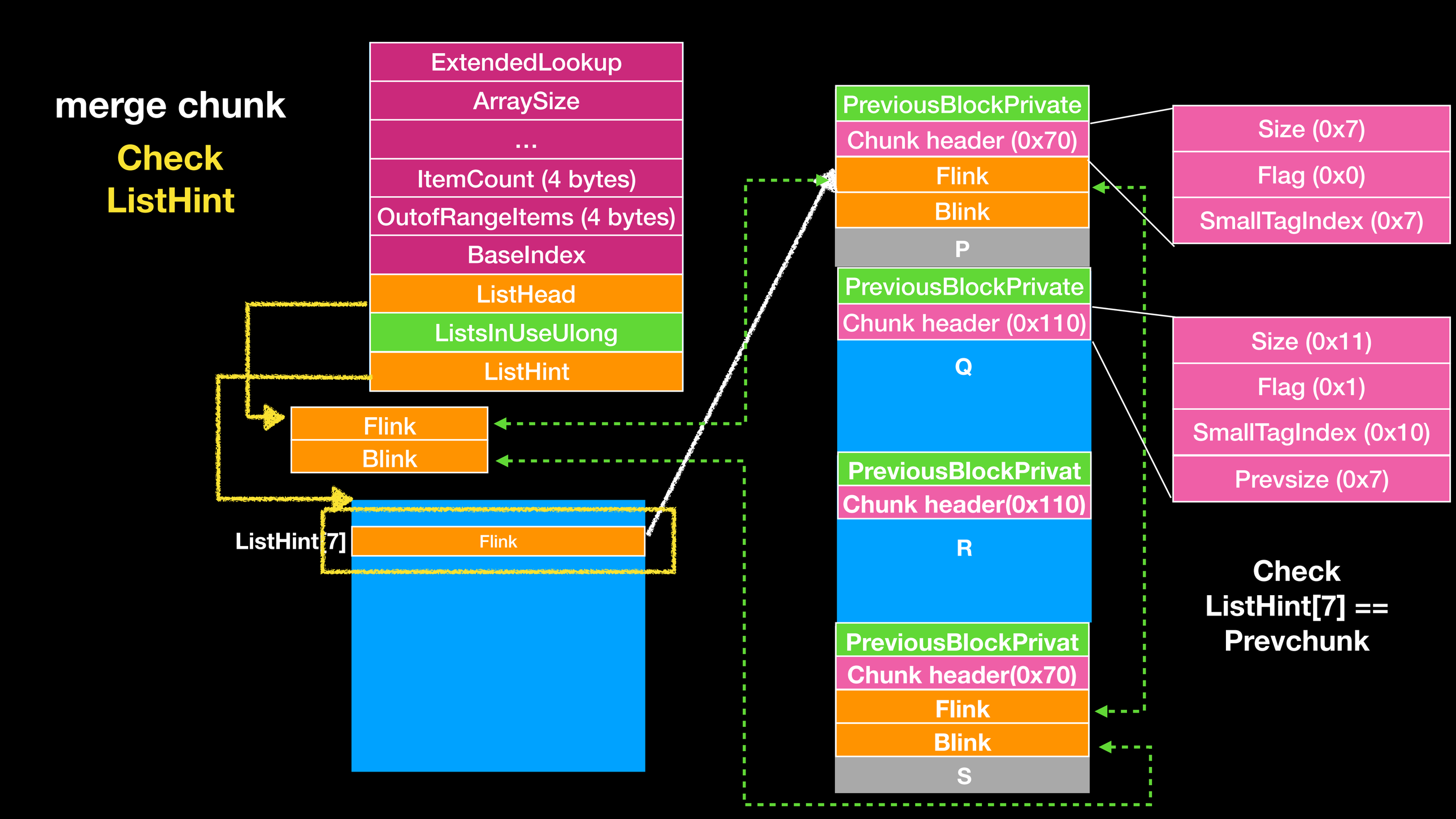
Check checksum

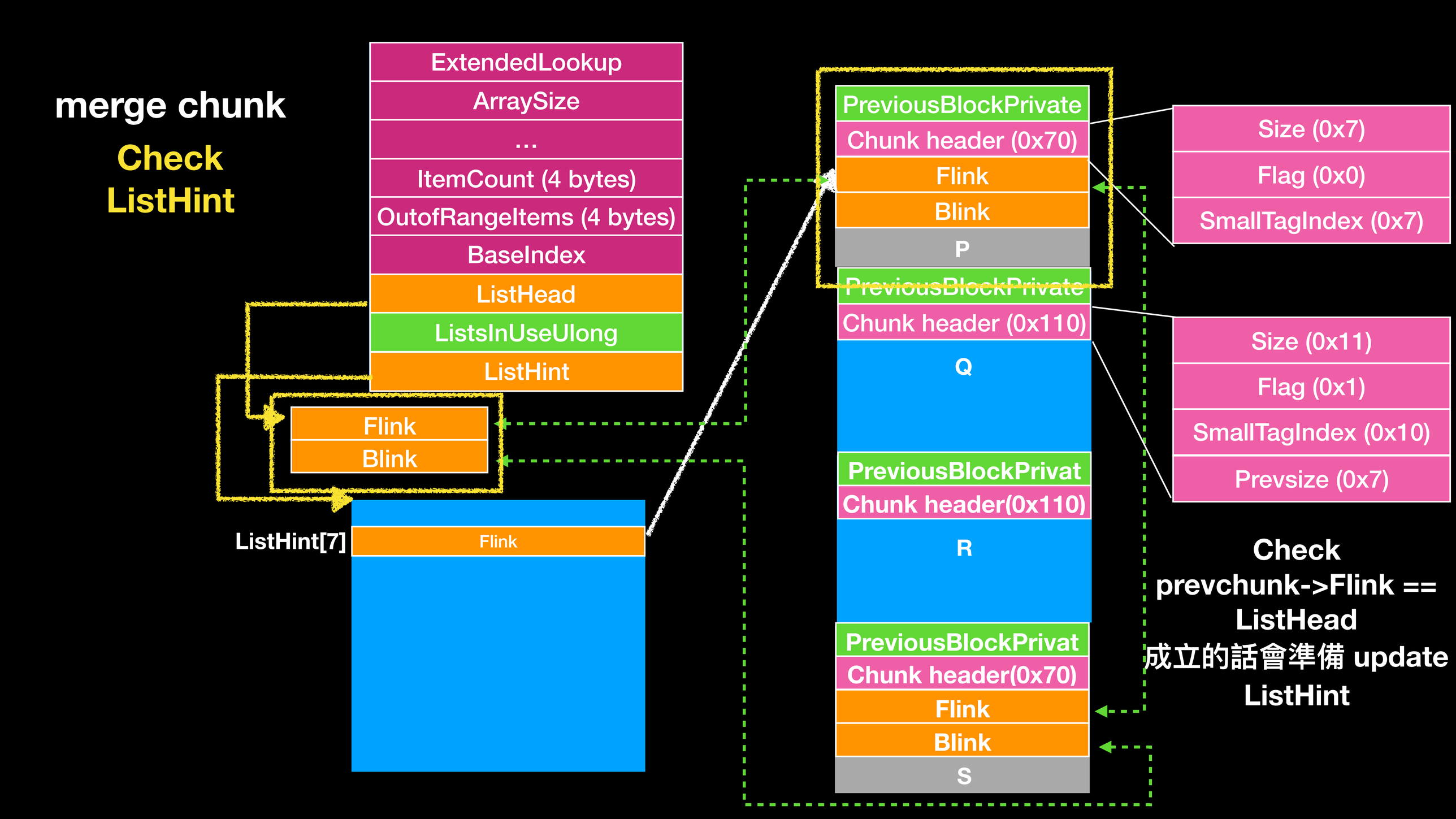


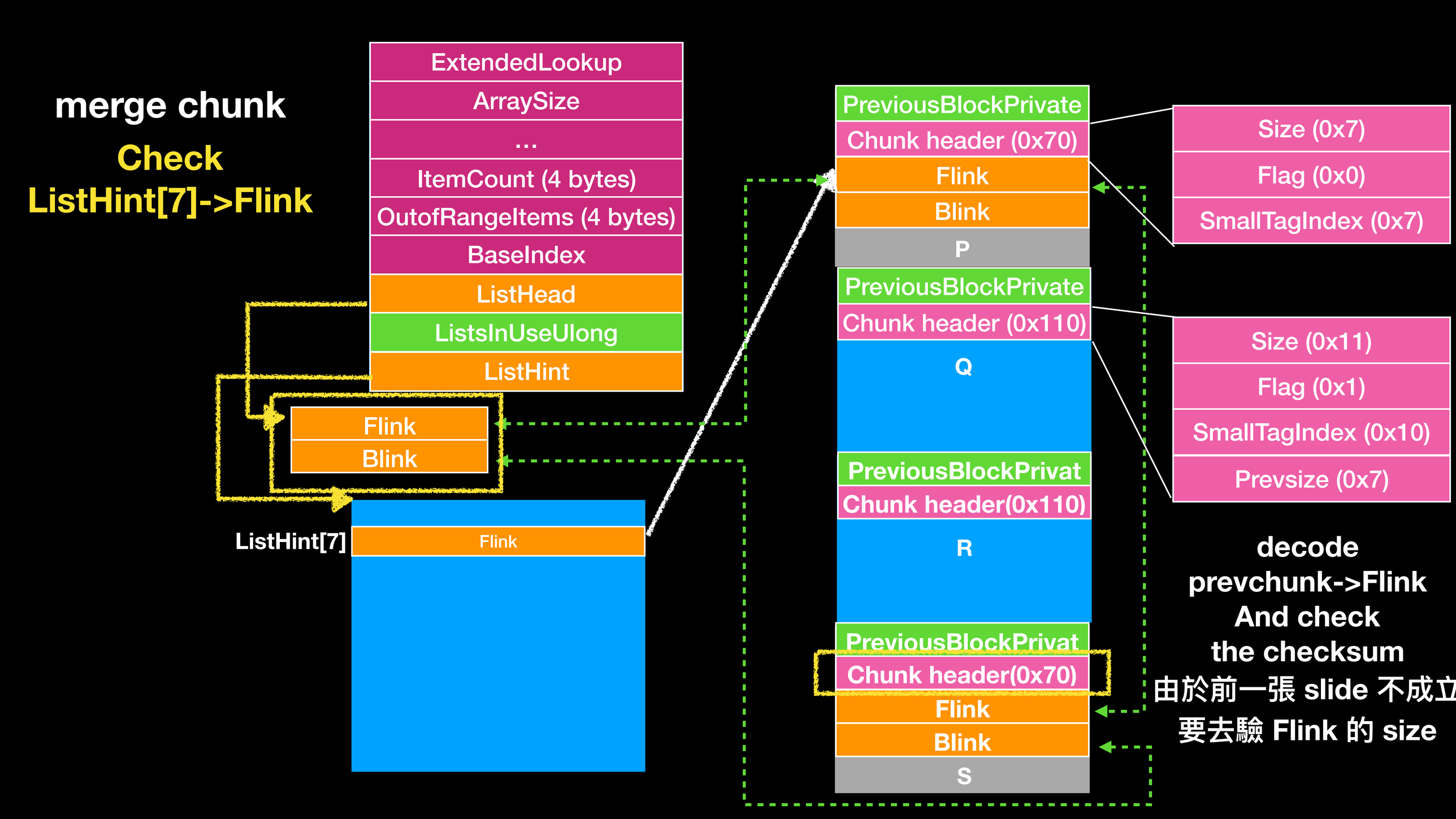

merge chunk
Check
linked list
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
PreviousBlockPrivate
Chunk header (0x110)
Q
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
P->Flink->Blink ==
P->Blink->Flink ==
P

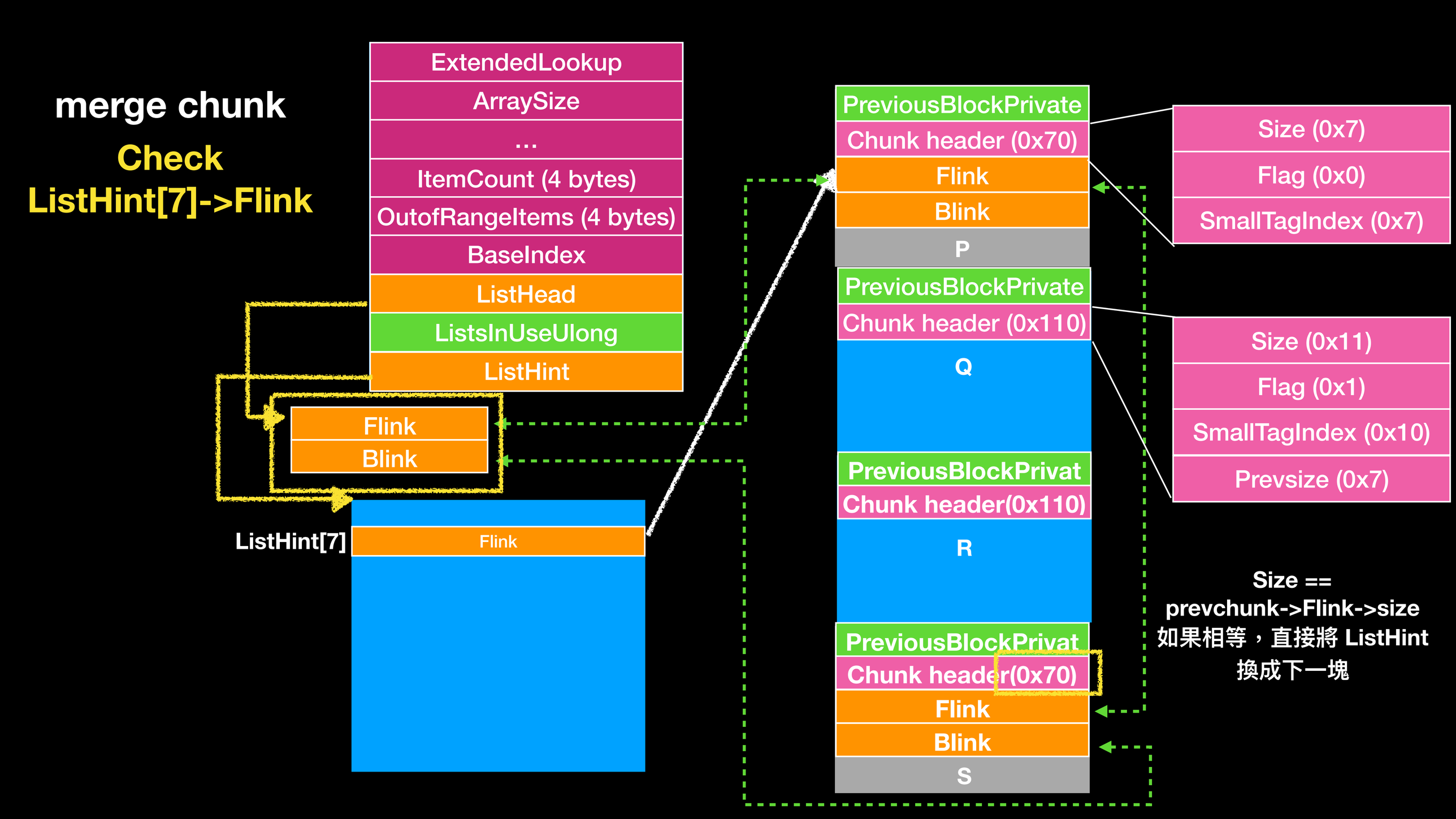

merge chunk
Find
BlocksIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
PreviousBlockPrivate
Chunk header (0x110)
Q
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S

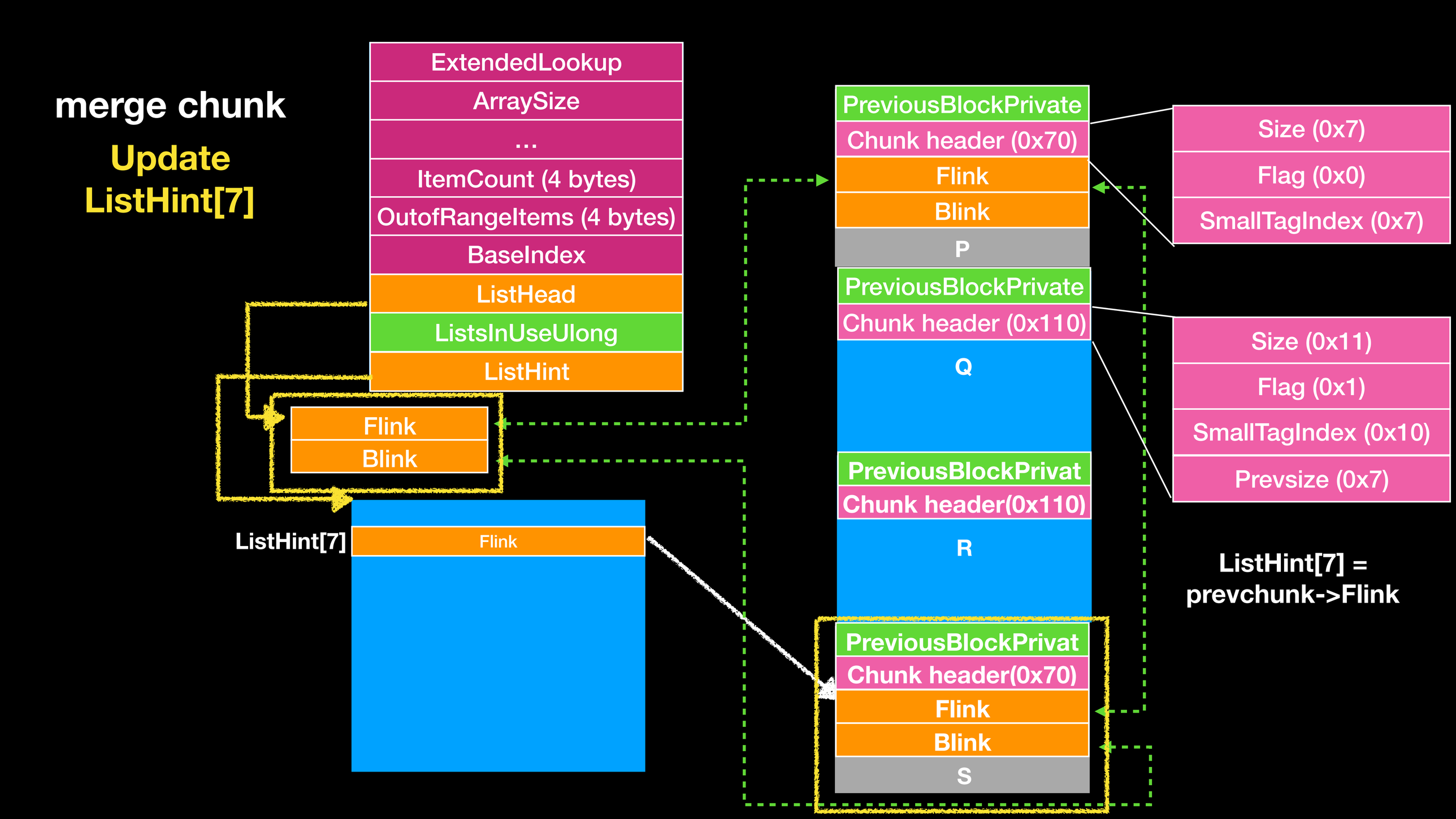

merge chunk
Find suitable BlocksIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
PreviousBlockPrivate
Chunk header (0x110)
Q
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Check
P->size < ArraySize

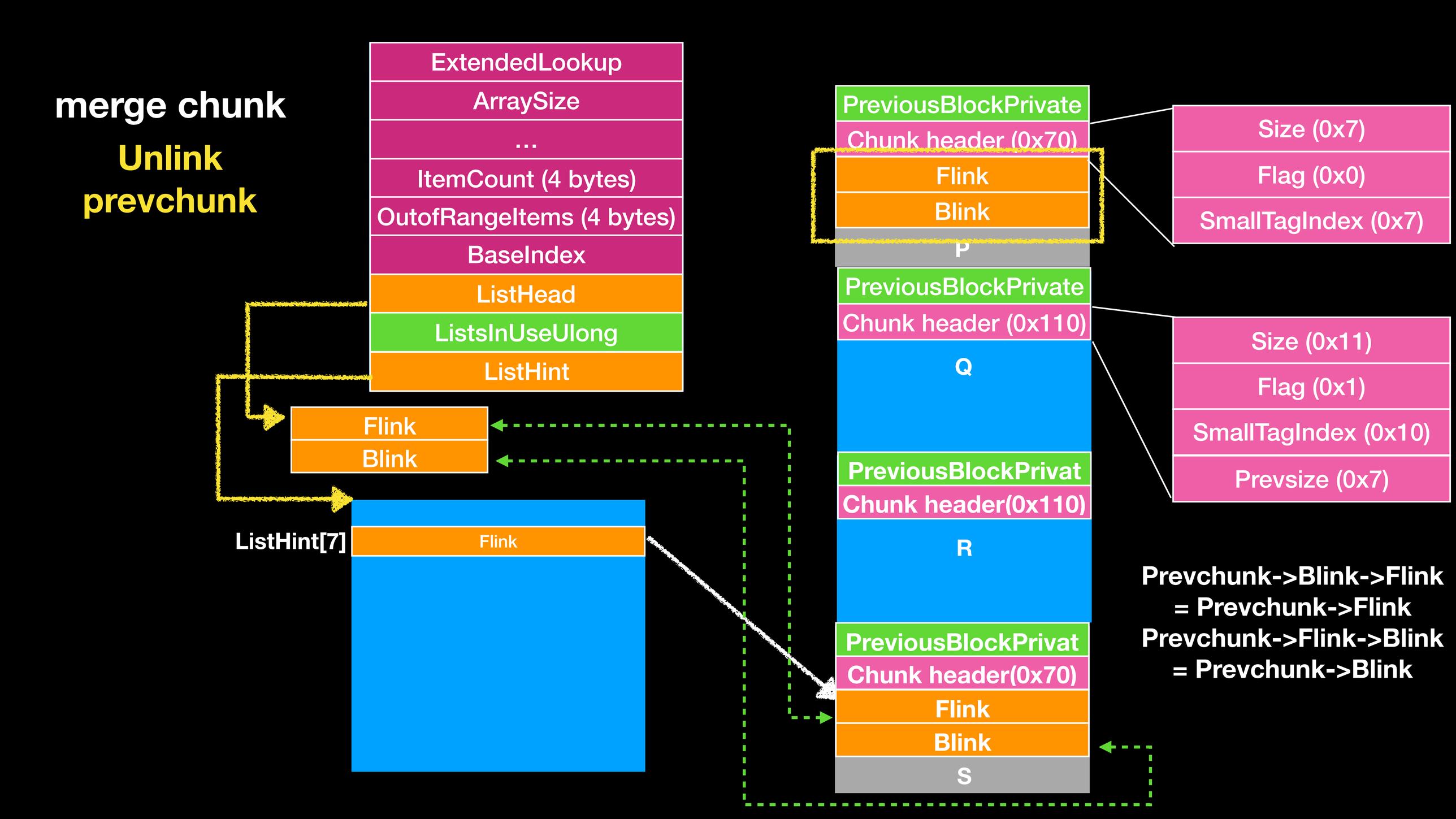

merge chunk
Check
ListHint
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
PreviousBlockPrivate
Chunk header (0x110)
Q
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Check
ListHint[7] ==
Prevchunk

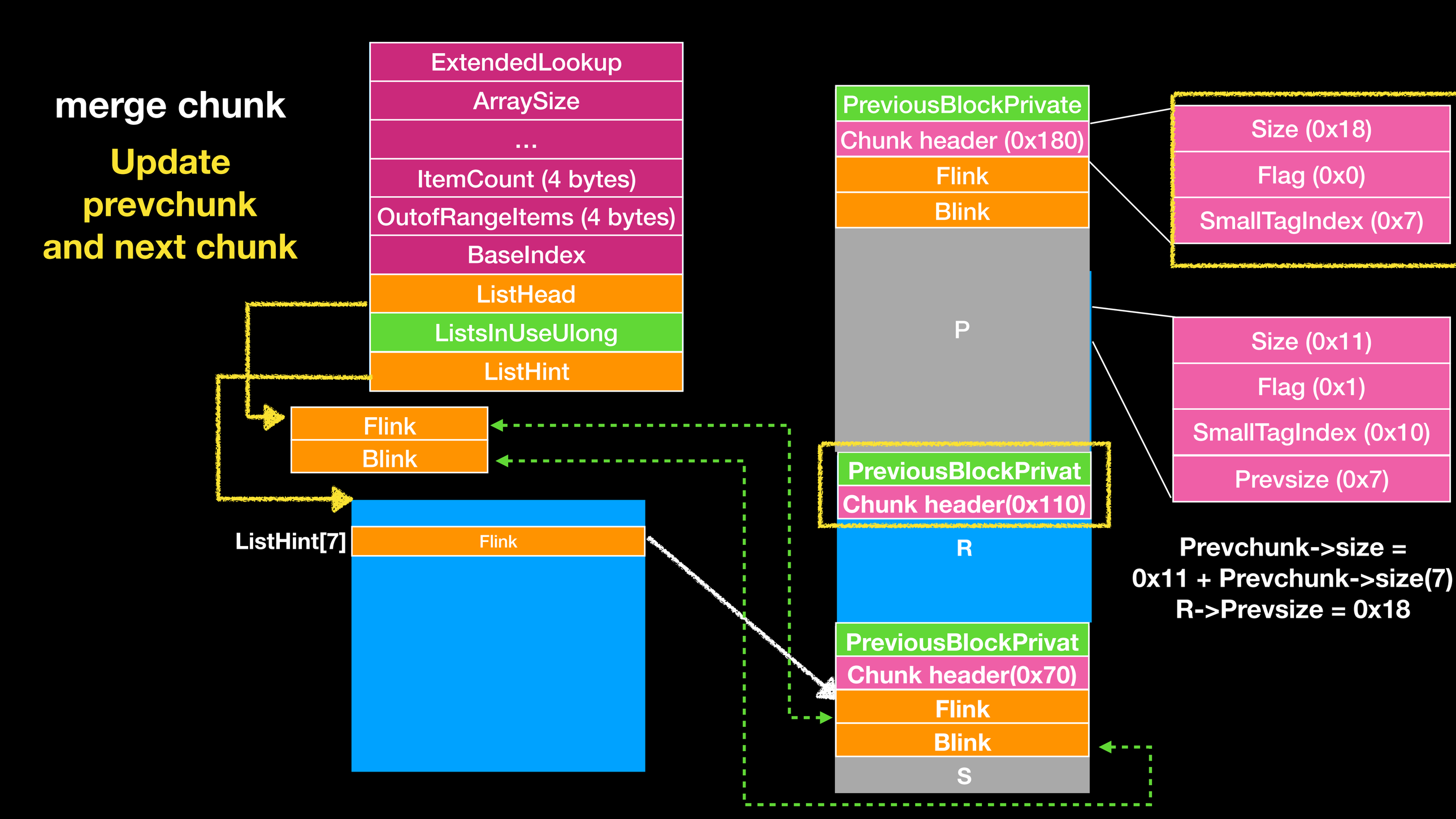

merge chunk
Check
ListHint
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
PreviousBlockPrivate
Chunk header (0x110)
Q
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
Check
prevchunk->Flink ==
ListHead
成立的話會準備 update
ListHint

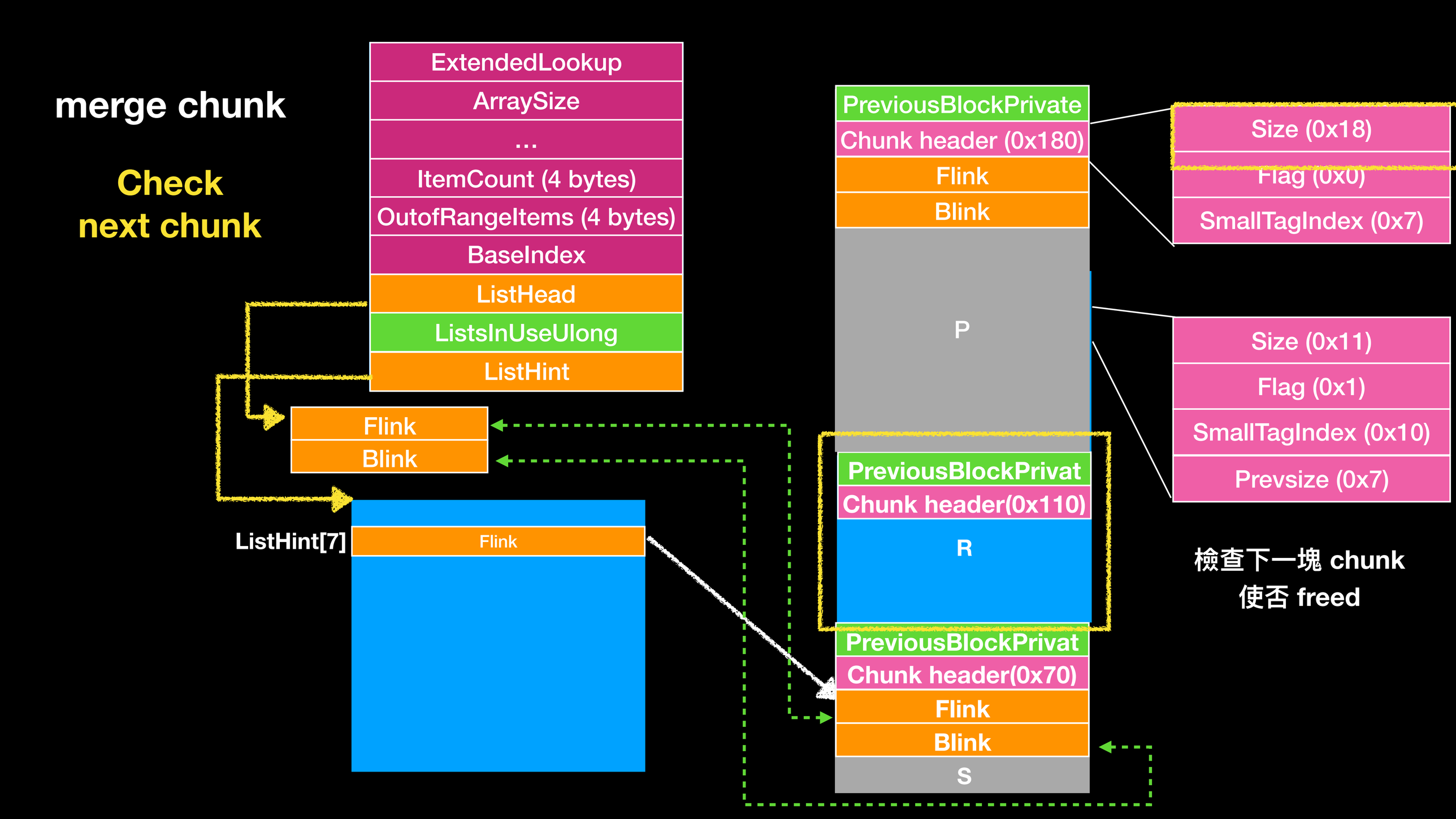

merge chunk
Check
ListHint[7]->Flink
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
PreviousBlockPrivate
Chunk header (0x110)
Q
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
decode
prevchunk->Flink
And check
the checksum
由於前一張 slide 不成立
要去驗 Flink 的 size

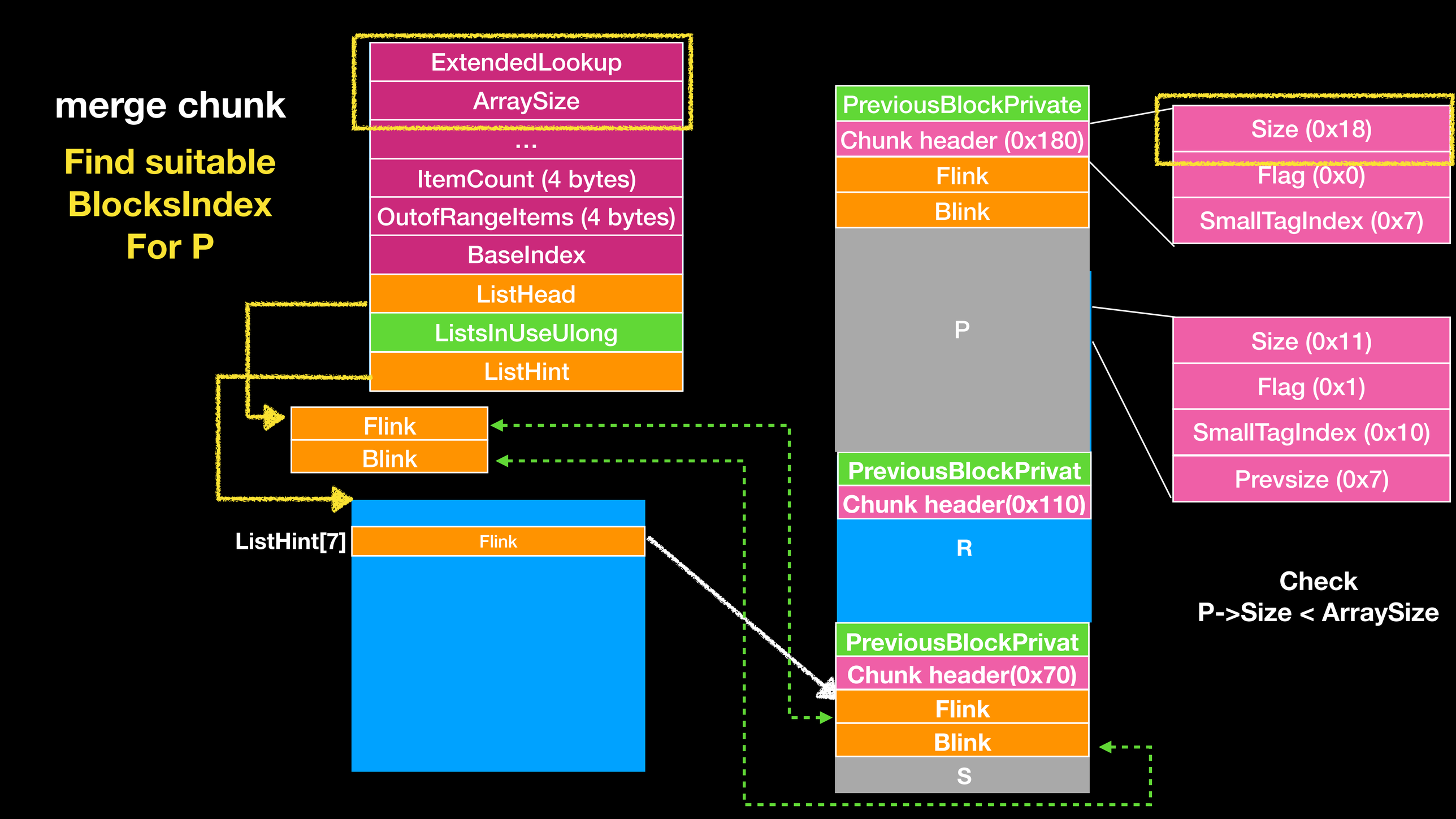

merge chunk
Check
ListHint[7]->Flink
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
PreviousBlockPrivate
Chunk header (0x110)
Q
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size ==
prevchunk->Flink->size
如果相等，直接將 ListHint
換成下一塊


merge chunk
Update
ListHint[7]
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
PreviousBlockPrivate
Chunk header (0x110)
Q
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
ListHint[7] =
prevchunk->Flink


merge chunk
Unlink
prevchunk
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x70)
Flink
Blink
P
Size (0x7)
Flag (0x0)
SmallTagIndex (0x7)
PreviousBlockPrivate
Chunk header (0x110)
Q
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Prevchunk->Blink->Flink
= Prevchunk->Flink
Prevchunk->Flink->Blink
= Prevchunk->Blink

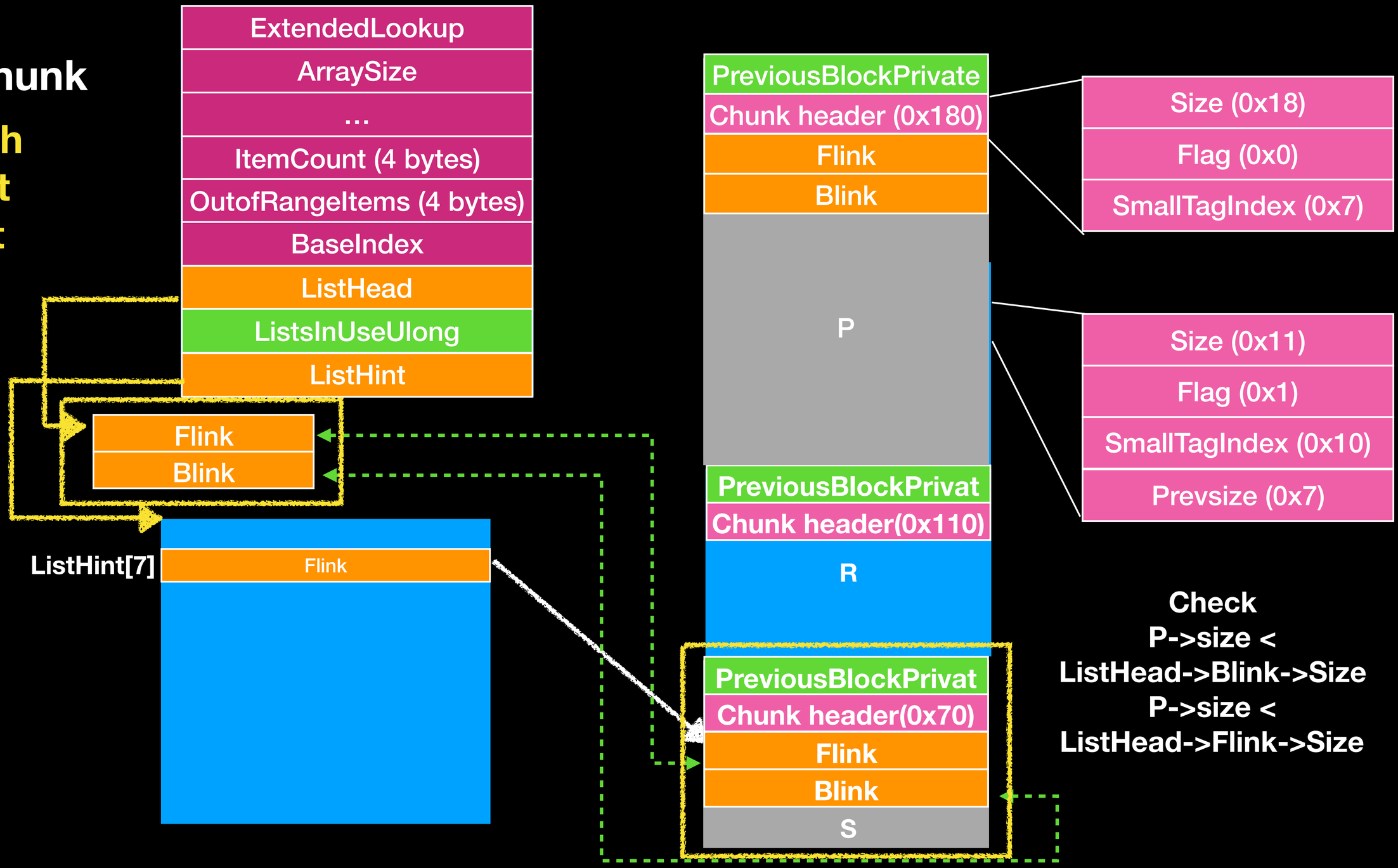

merge chunk
Update prevchunk and next chunk
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x180)
Flink
Blink
P
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x18)
Flag (0x0)
SmallTagIndex (0x7)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
Prevchunk->size =
0x11 + Prevchunk->size(7)
R->Prevsize = 0x18


merge chunk
Check
next chunk
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x180)
Flink
Blink
P
Size (0x18)
Flag (0x0)
SmallTagIndex (0x7)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
檢查下一塊 chunk
使否 freed


merge chunk
Find suitable BlocksIndex For P
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
PreviousBlockPrivate
Chunk header (0x180)
Flink
Blink
P
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x18)
Flag (0x0)
SmallTagIndex (0x7)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
Check
P->Size < ArraySize

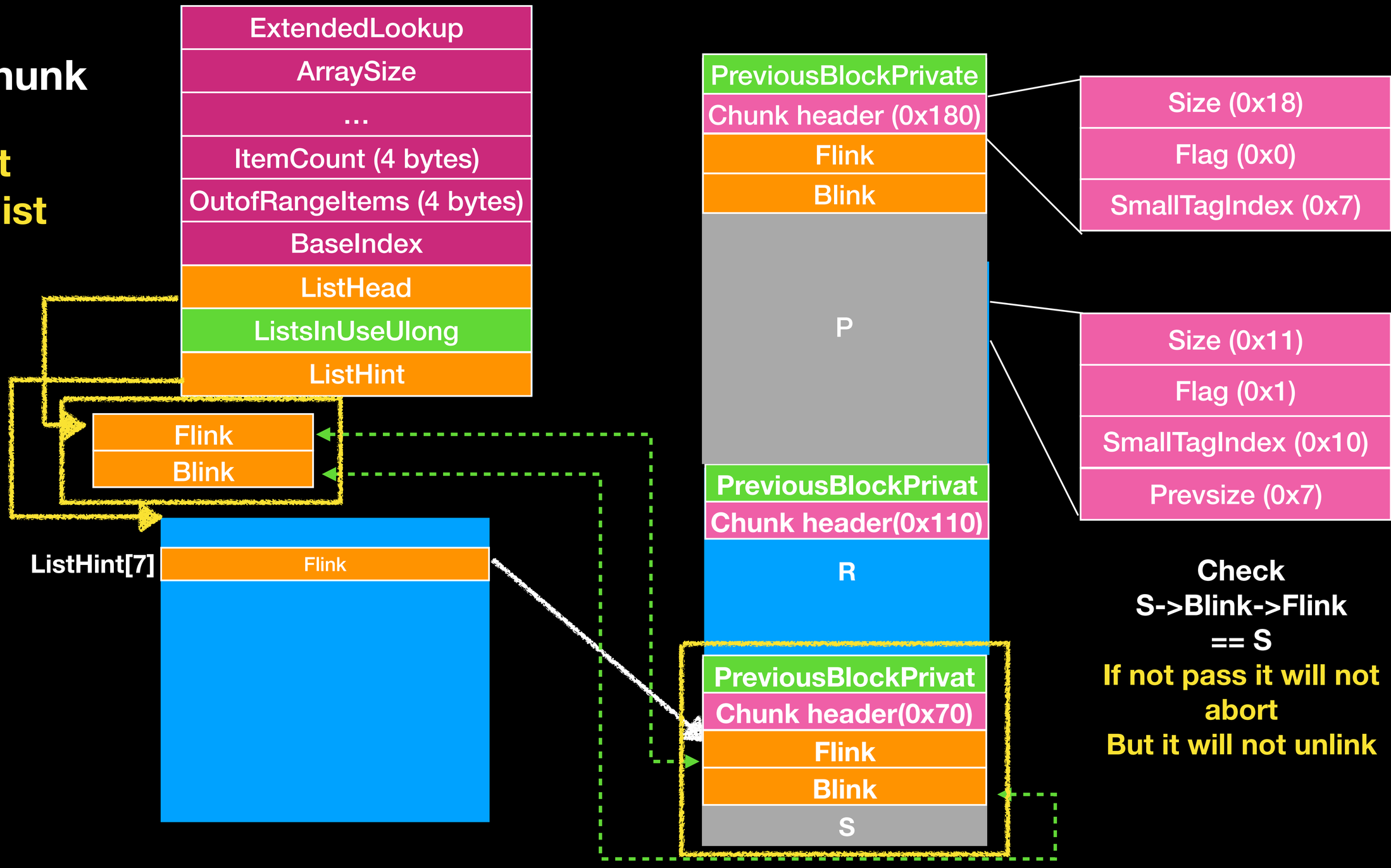
merge chunk

Search insert point



merge chunk

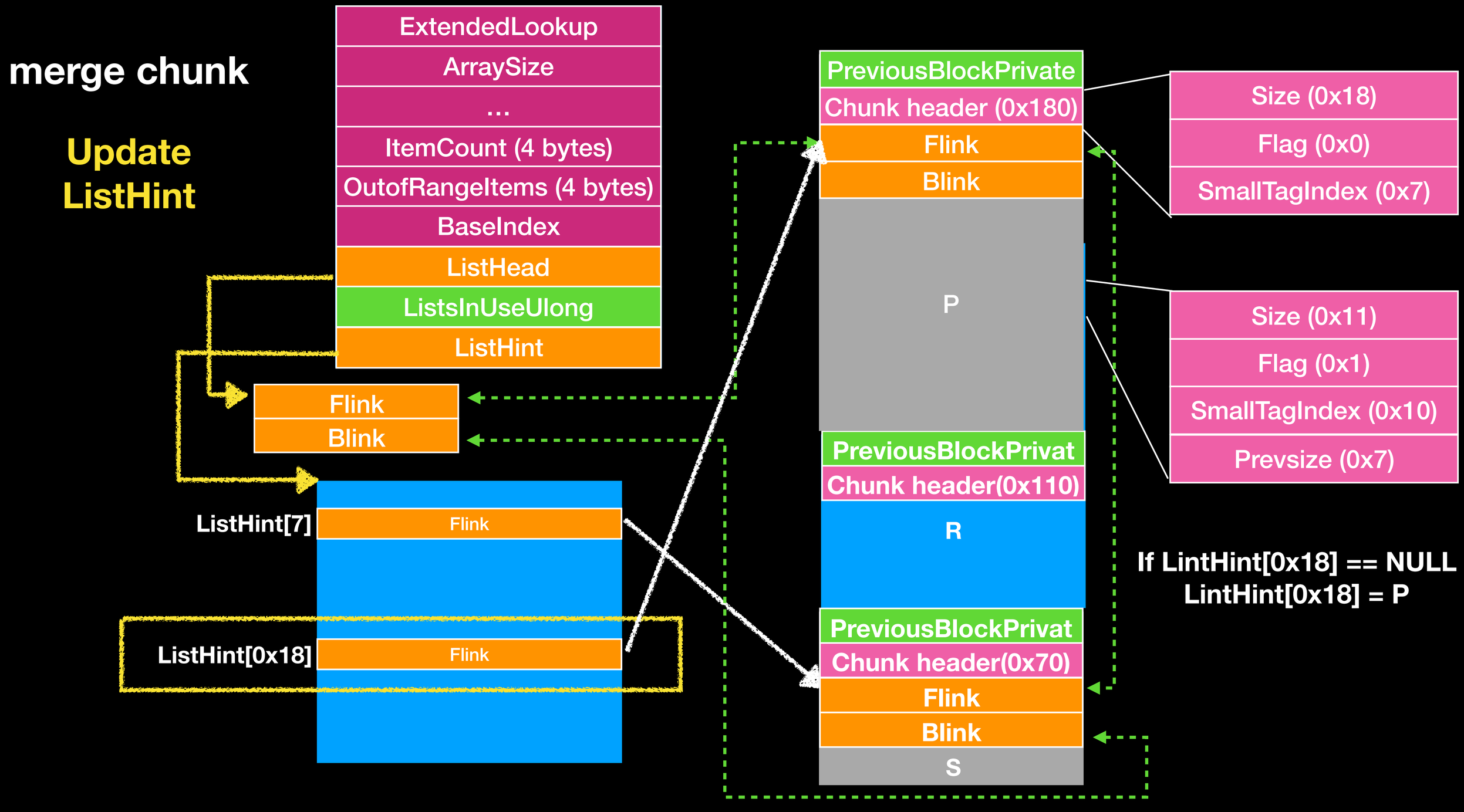
Insert linked list

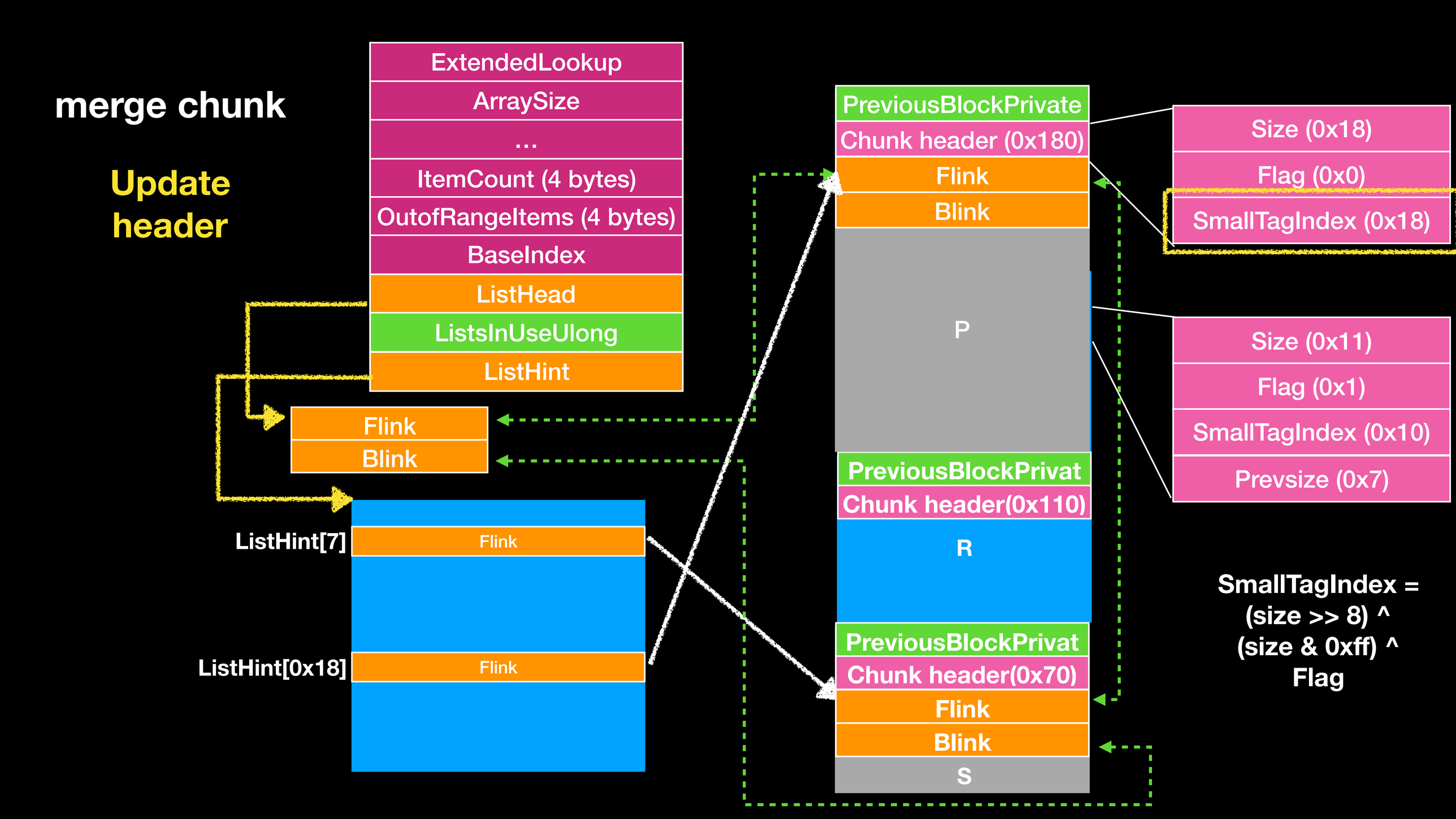


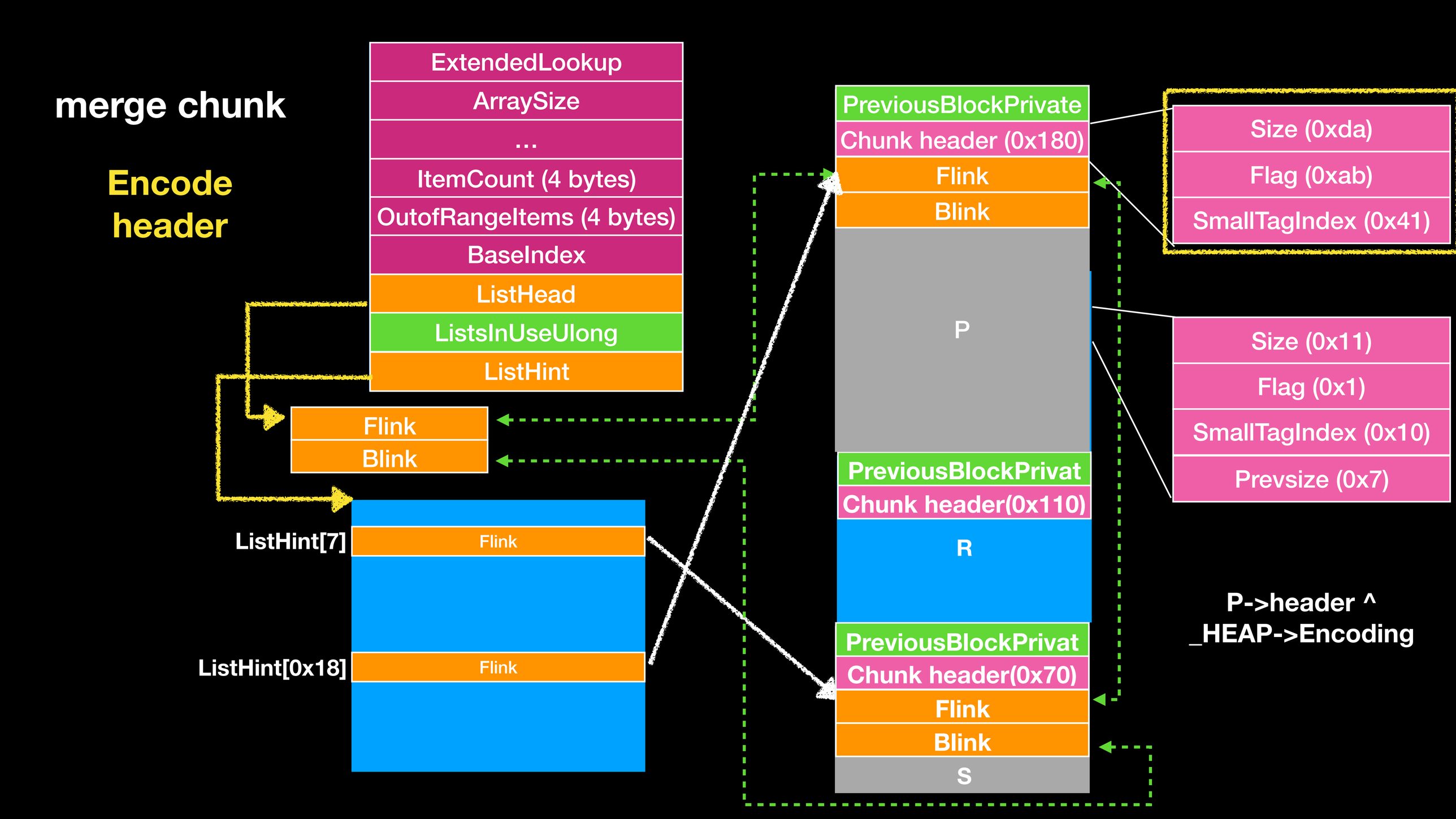
Check S->Blink->Flink == S

If not pass it will not abort

But it will not unlink


merge chunk
Update
ListHint
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
ListHint[0x18]
Flink
PreviousBlockPrivate
Chunk header (0x180)
Flink
Blink
P
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x18)
Flag (0x0)
SmallTagIndex (0x7)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
If LintHint[0x18] == NULL
LintHint[0x18] = P


merge chunk
Update header
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
ListHint[0x18]
Flink
PreviousBlockPrivate
Chunk header (0x180)
Flink
Blink
P
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0x18)
Flag (0x0)
SmallTagIndex (0x18)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
SmallTagIndex = (size >> 8) ^ (size & 0xff) ^ Flag


merge chunk
Encode header
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
ListHint[7]
Flink
ListHint[0x18]
Flink
PreviousBlockPrivate
Chunk header (0x180)
Flink
Blink
P
PreviousBlockPrivat
Chunk header(0x110)
R
PreviousBlockPrivat
Chunk header(0x70)
Flink
Blink
S
Size (0xda)
Flag (0xab)
SmallTagIndex (0x41)
Size (0x11)
Flag (0x1)
SmallTagIndex (0x10)
Prevsize (0x7)
P->header ^
_HEAP->Encoding

Nt heap

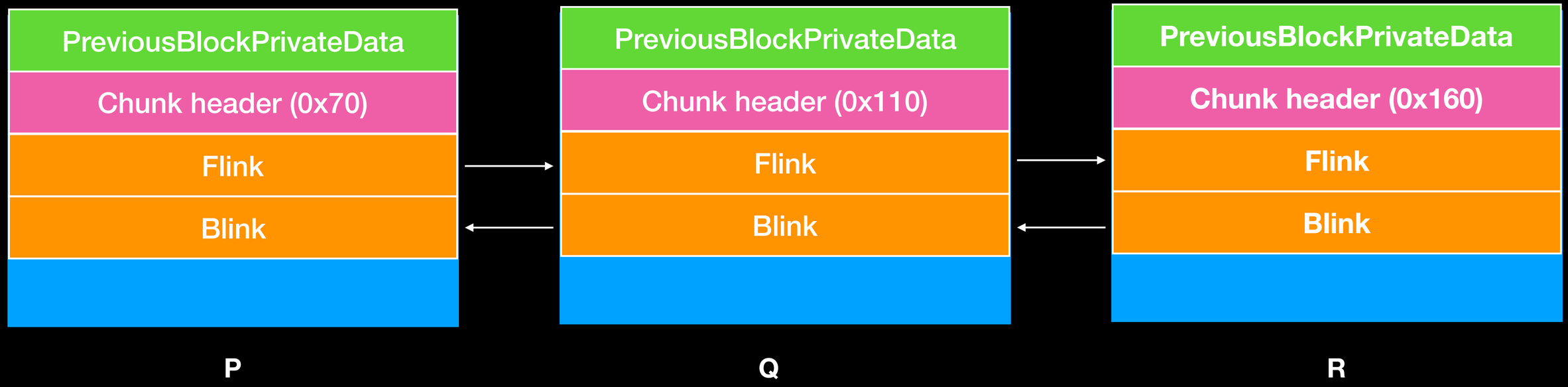
- Free (RtlpFreeHeap)
  - Size > 0xff000
    - 檢查該 chunk 的 linked list 並從 _HEAP->VirtualAllocdBlocks 移除
    - 接著使用 RtlpSecMemFreeVirtualMemory 將 chunk 整個 munmap 掉

Back-End Exploitation

- Unlink
  - 基本上與 Linux 中的 unlink 很類似，繞過方法也差不多
    - 簡而言之就是利用從 linked listed 移除 node 的行為來做有限制的寫入
  - 要注意的是在會 decode 的地方，都要讓他正常 decode
    - 也就是 check sum 要過
  - 另外一點是 Flink 及 Blink 並不是指向 chunk 開頭，而是直接指向 User data 部分，也就是不太需要做偏移偽造 chunk
    - 所以找到一個指向該 userdata 的 pointer 讓他繞過 double linked list 的驗證就好了

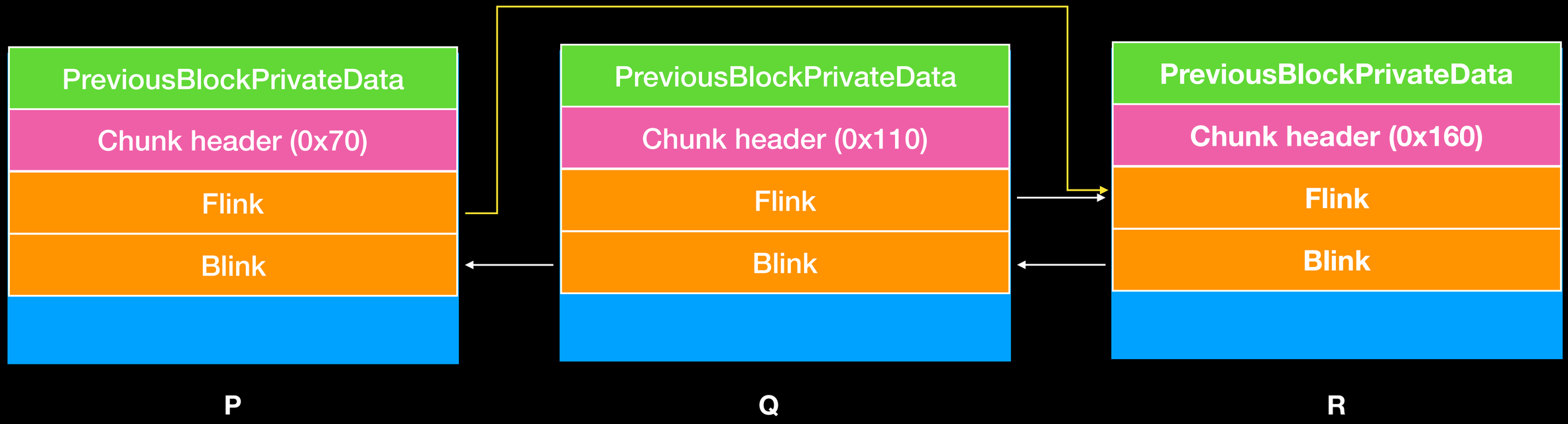
Back-End Exploitation

- Unlink
  - Q->Blink->Flink = Q->Flink
  - Q->Flink->Blink = Q->Blink



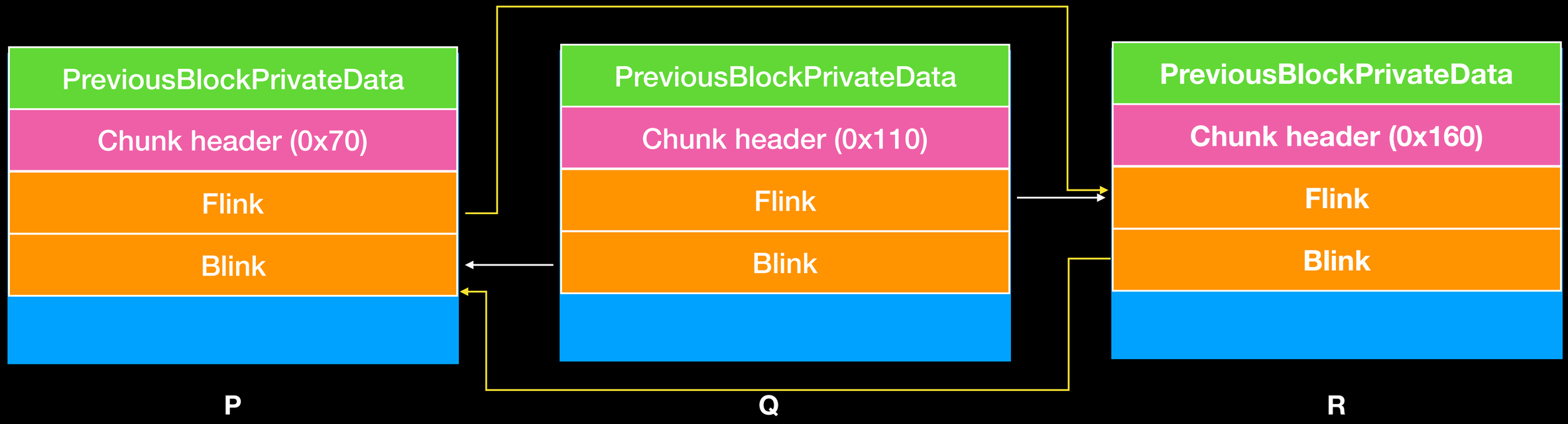
Back-End Exploitation

- Unlink
  - Q->Blink->Flink = Q->Flink
  - Q->Flink->Blink = Q->Blink



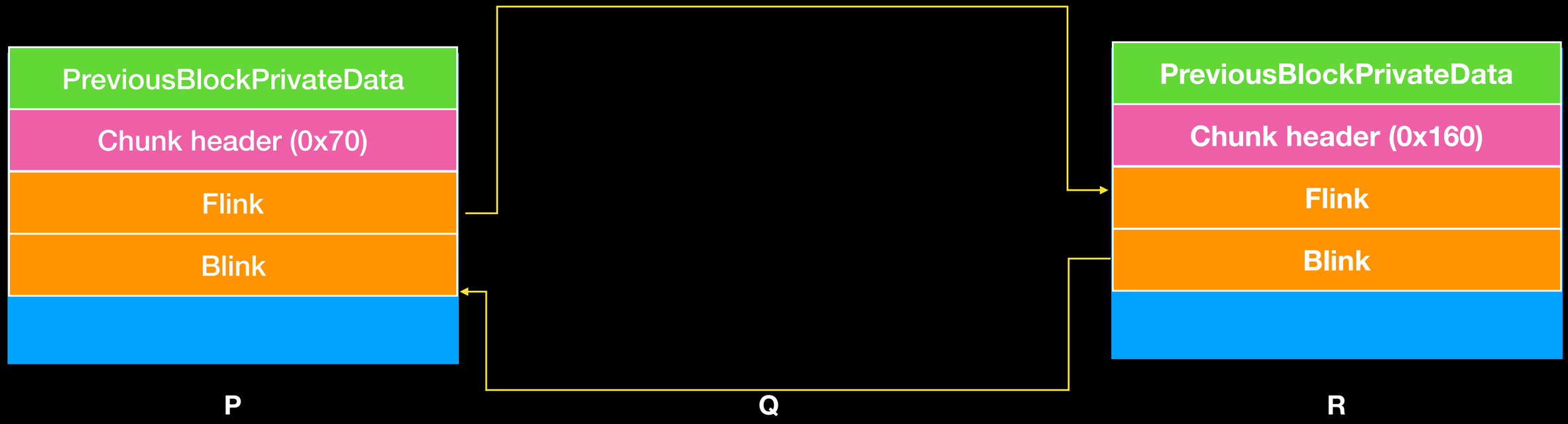
Back-End Exploitation

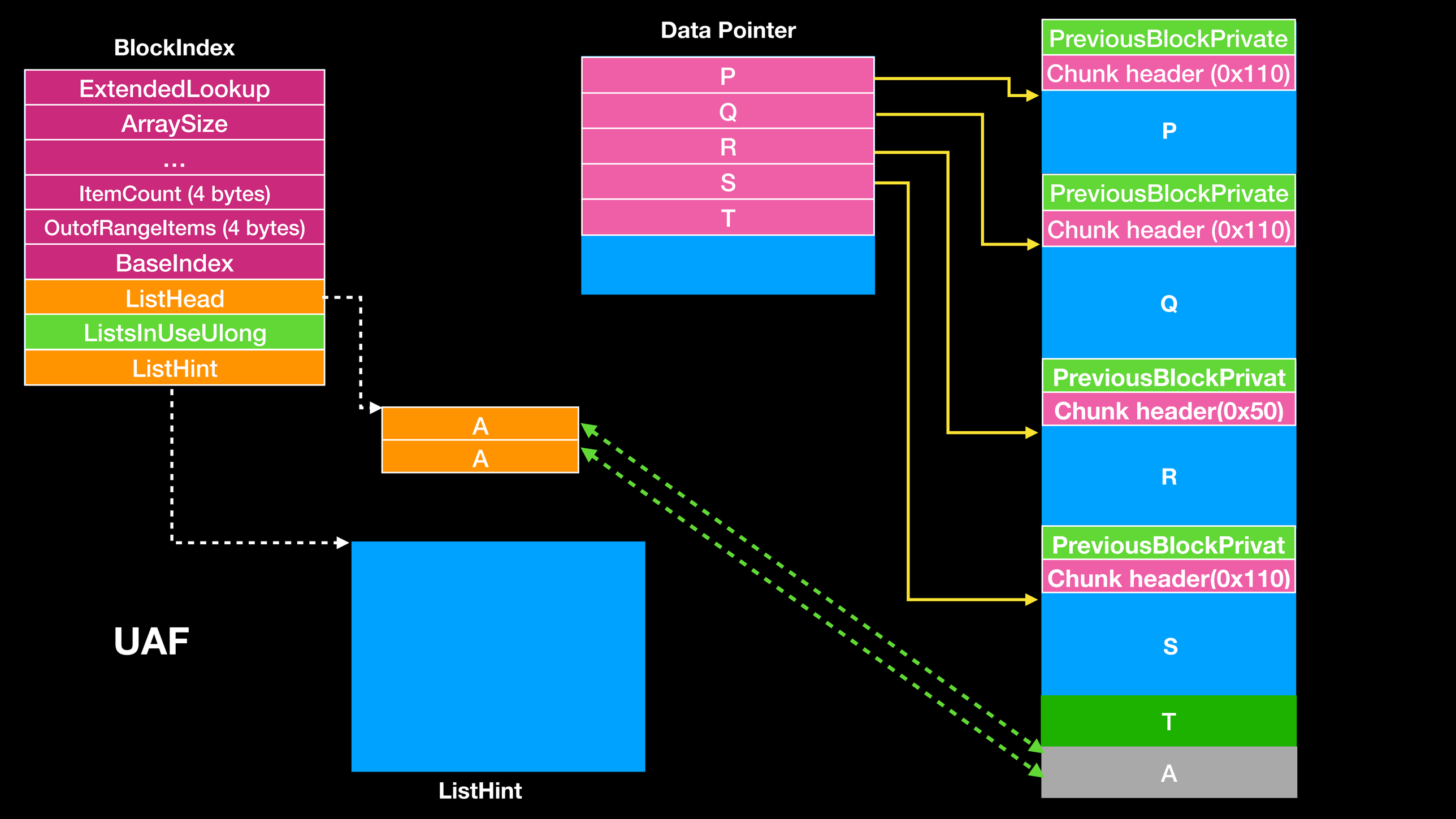
- Unlink
  - Q->Blink->Flink = Q->Flink
  - Q->Flink->Blink = Q->Blink

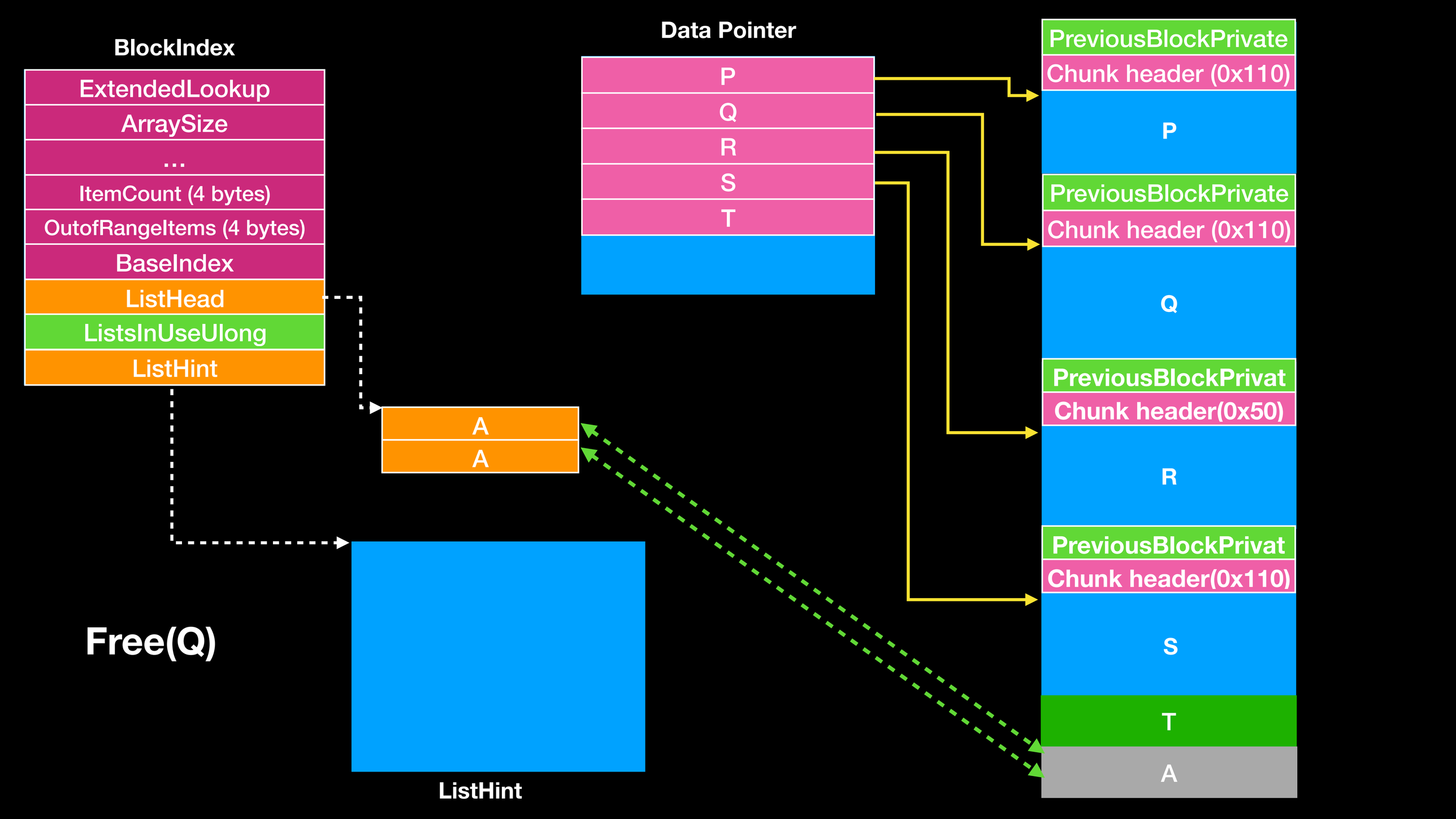


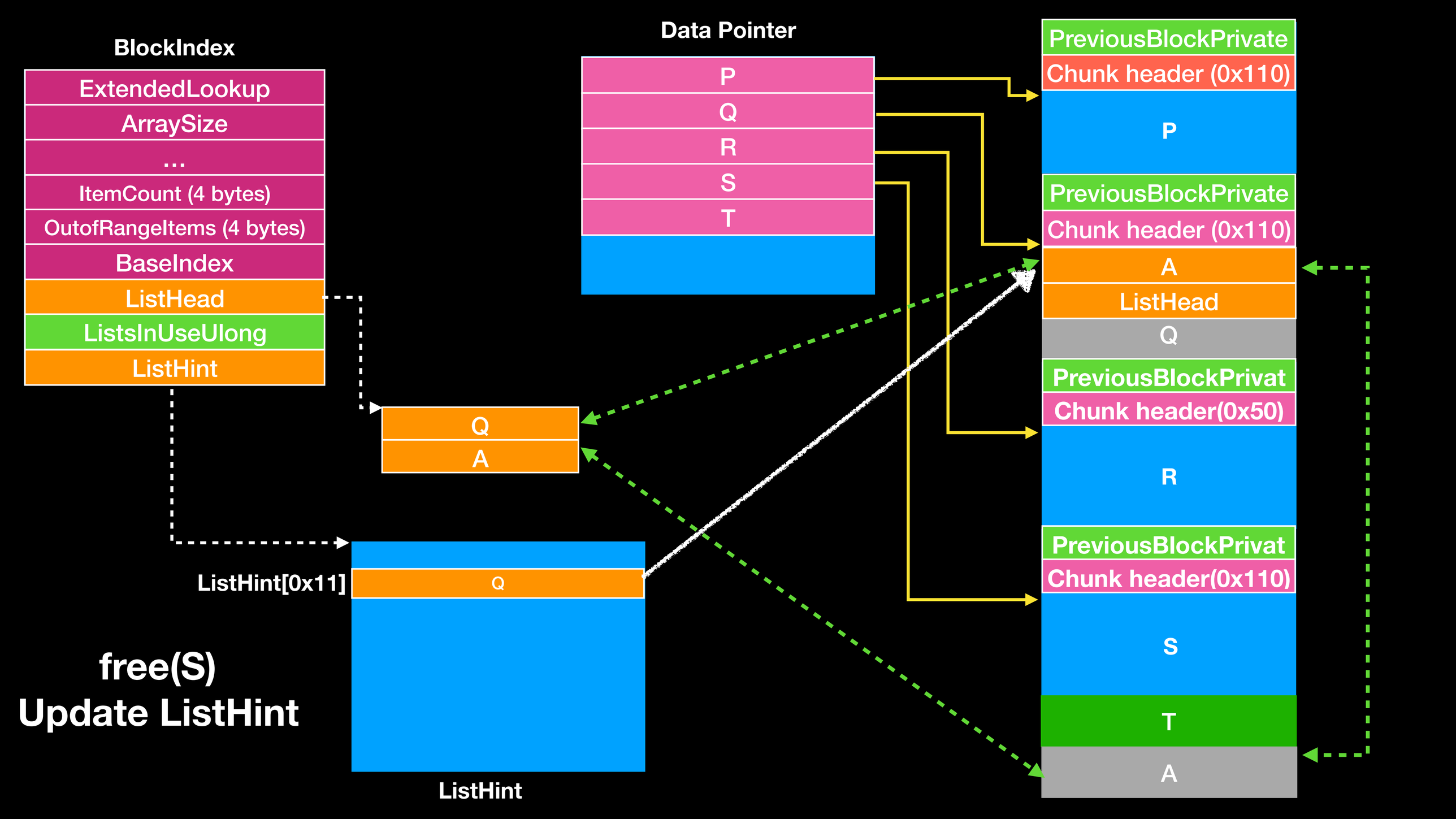
Back-End Exploitation

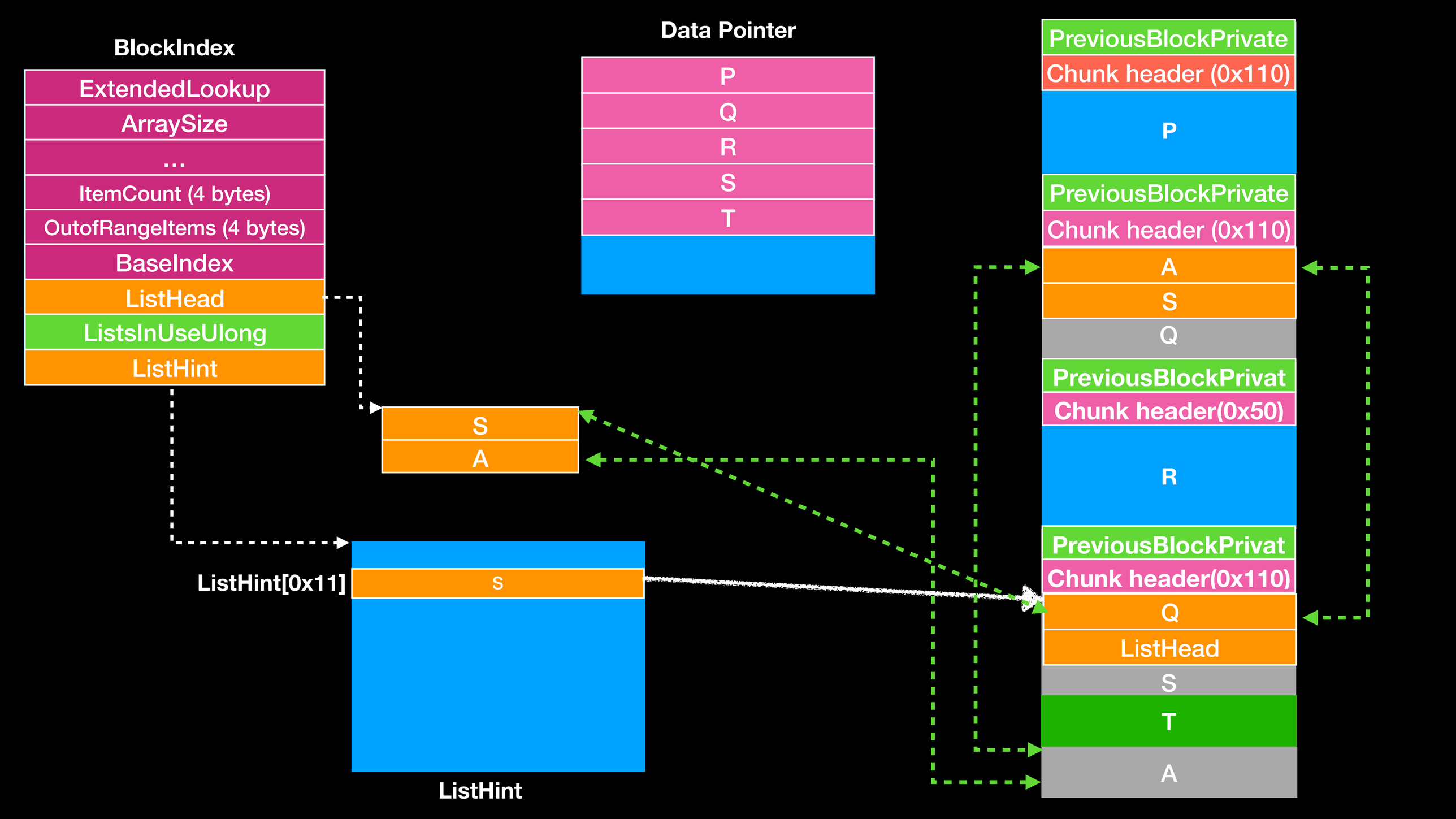
- Unlink
  - Q->Blink->Flink = Q->Flink
  - Q->Flink->Blink = Q->Blink

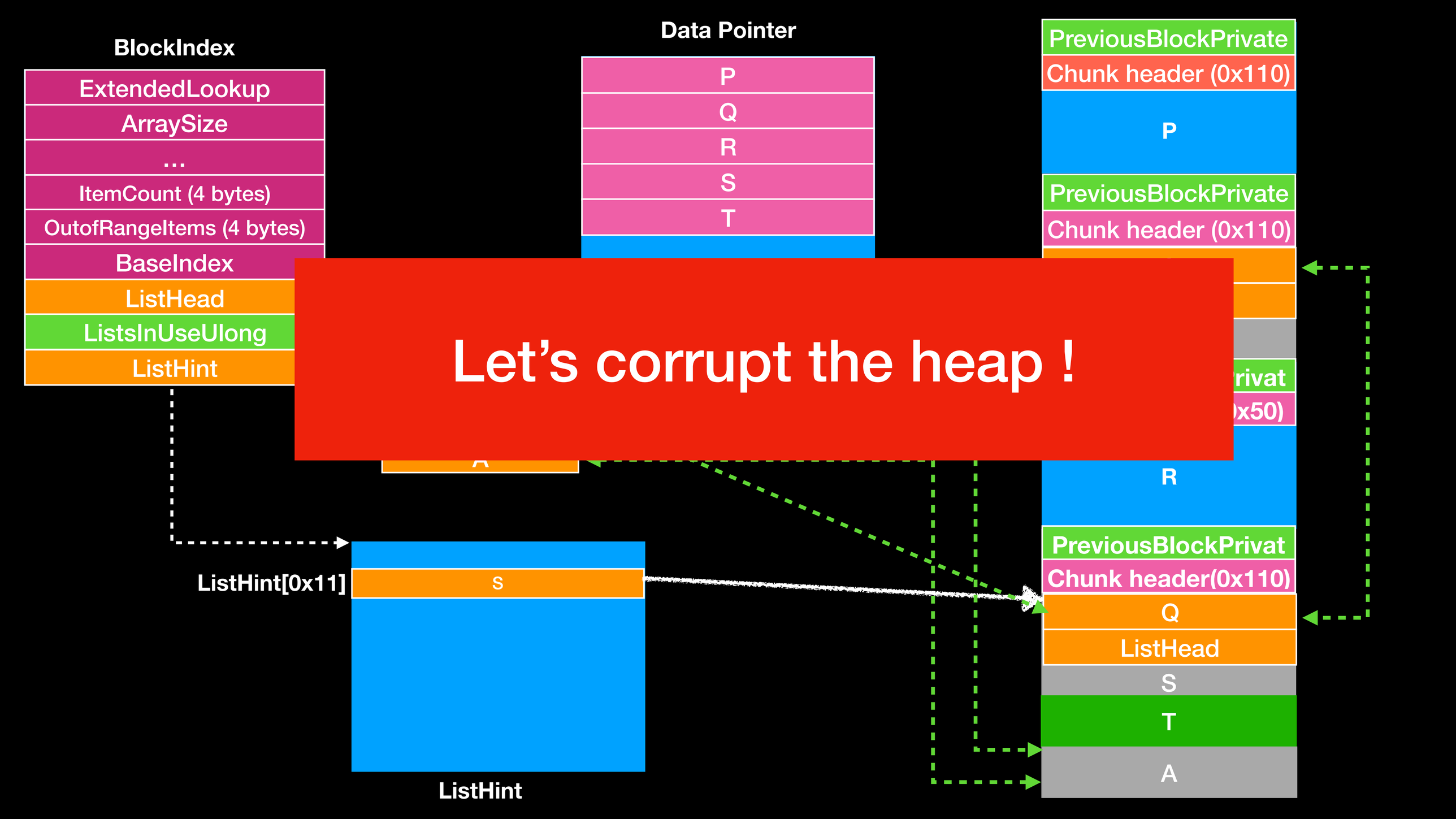


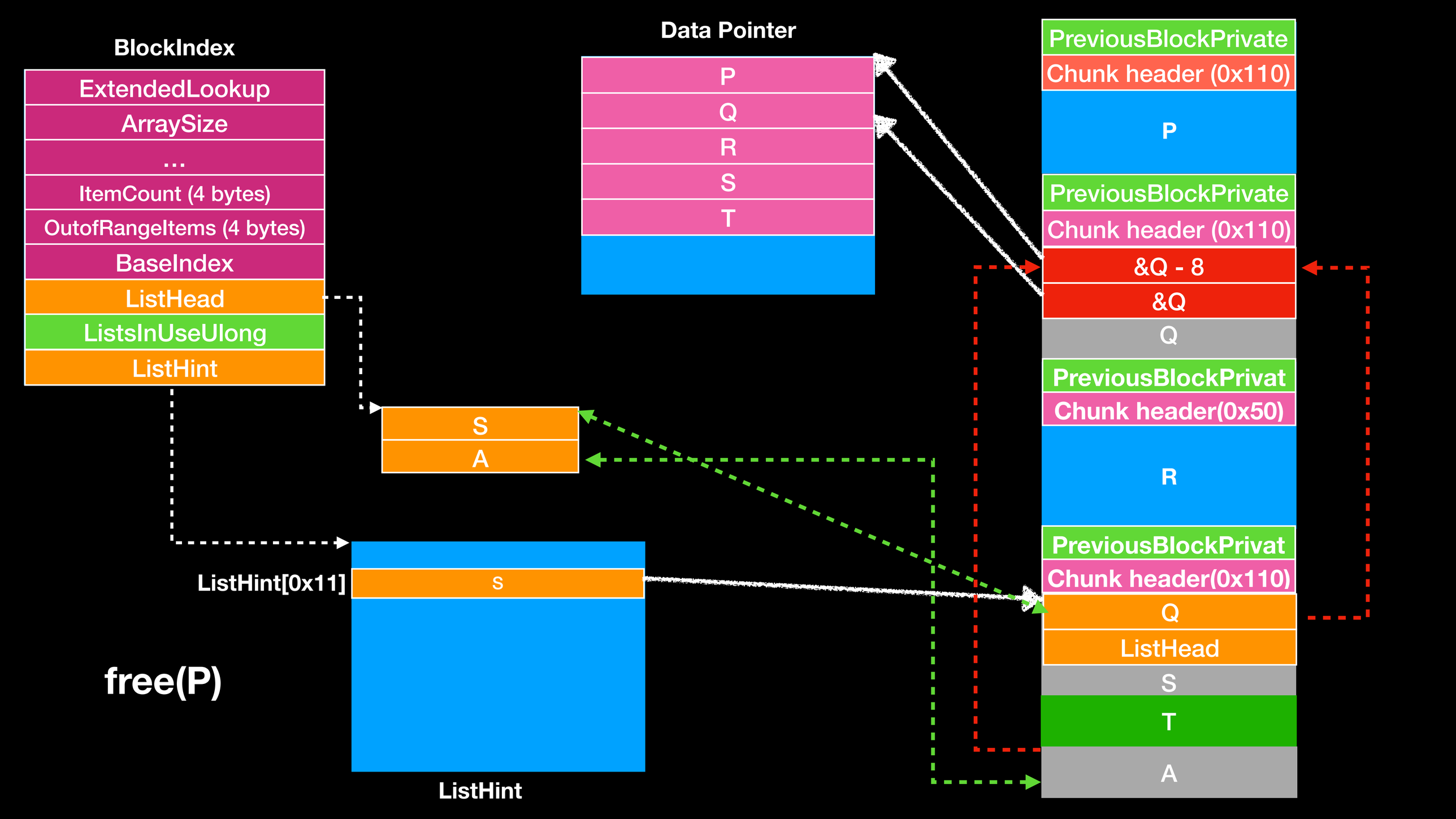

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
A
A
UAF
ListHint
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
S
T
A

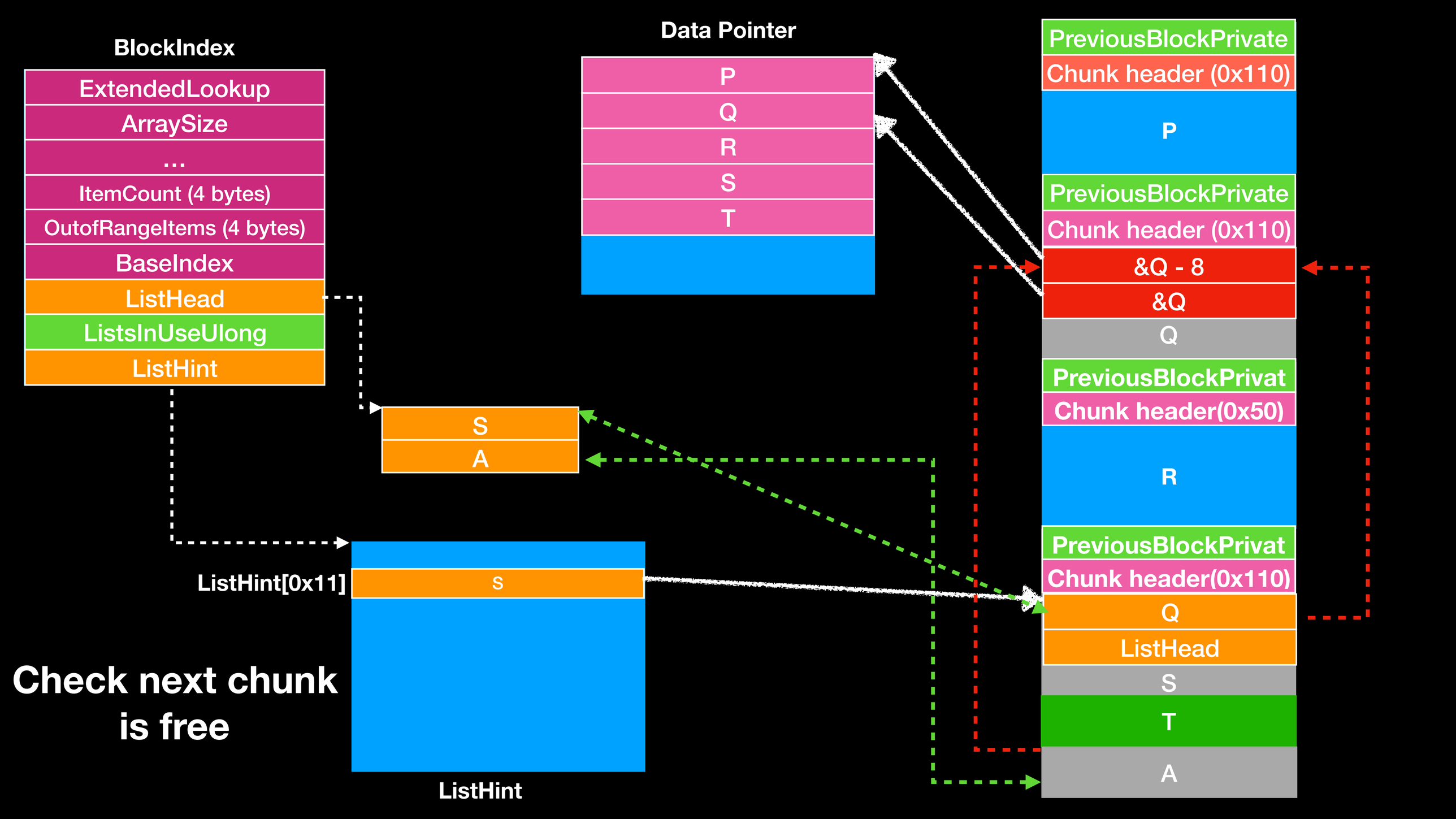

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
A
A
ListHint
Free(Q)
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
S
T
A

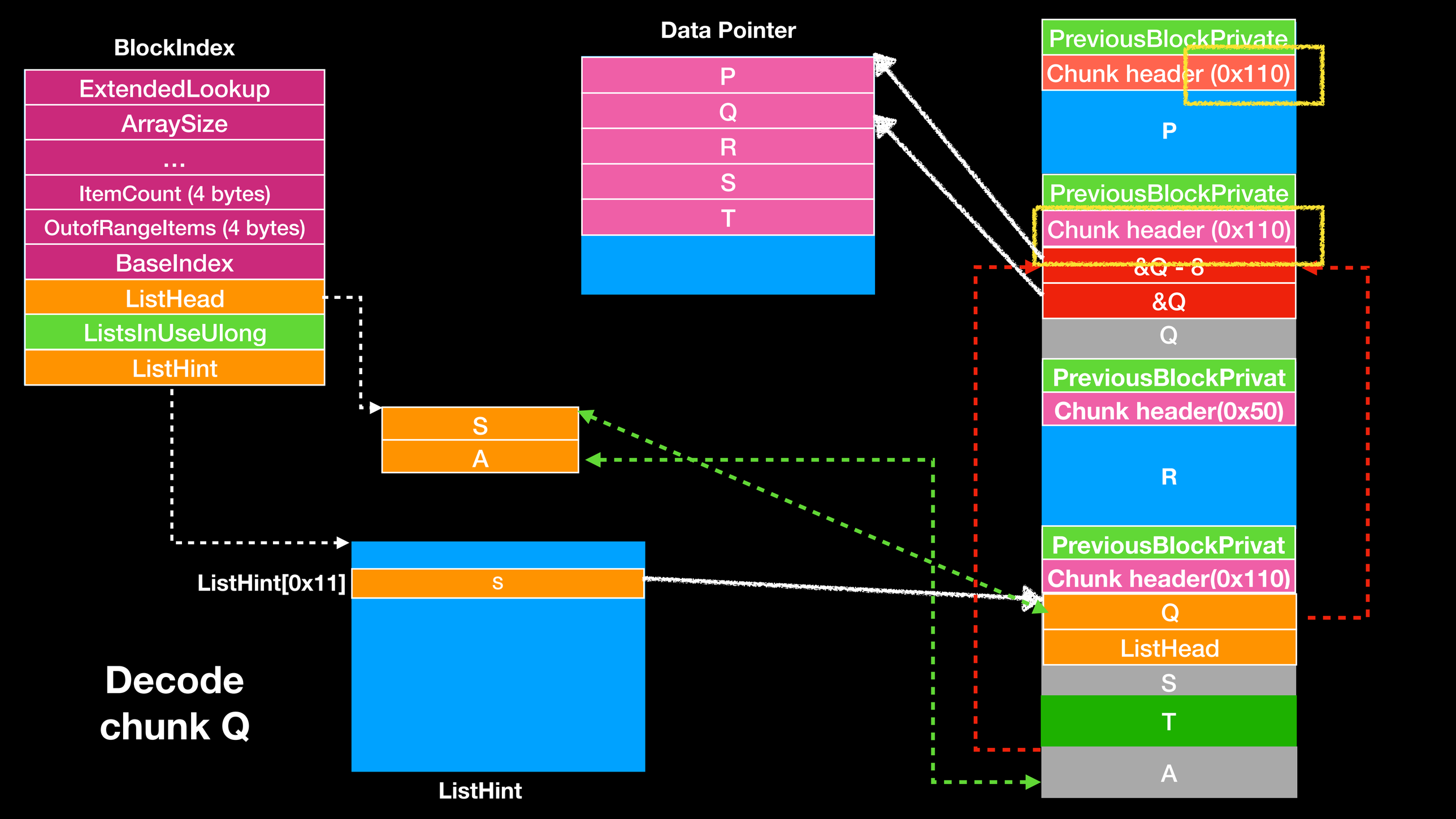

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
Q
A
ListHint[0x11]
Q
ListHint
free(S)
Update ListHint
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
A
ListHead
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
S
T
A

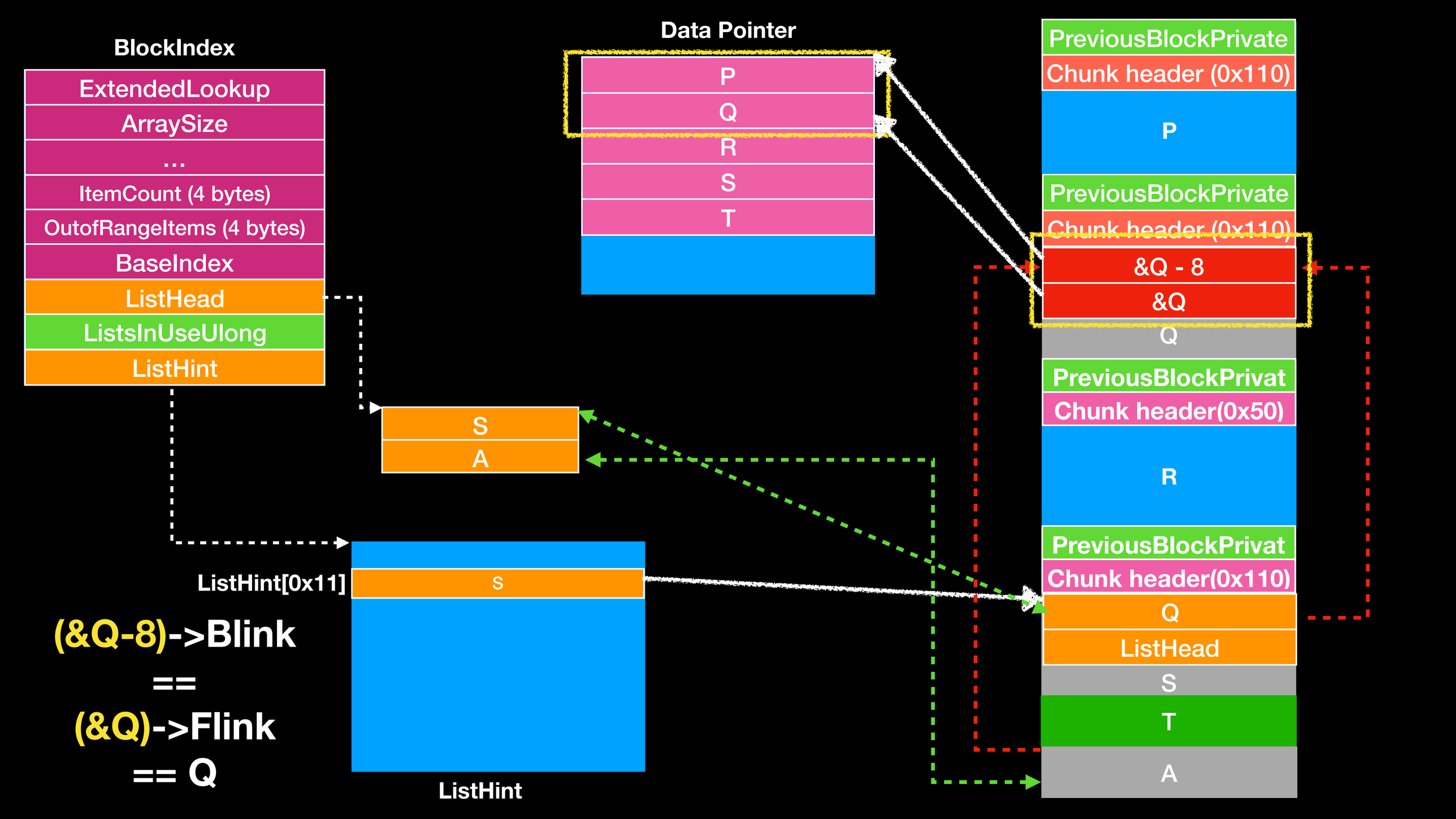

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
A
S
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

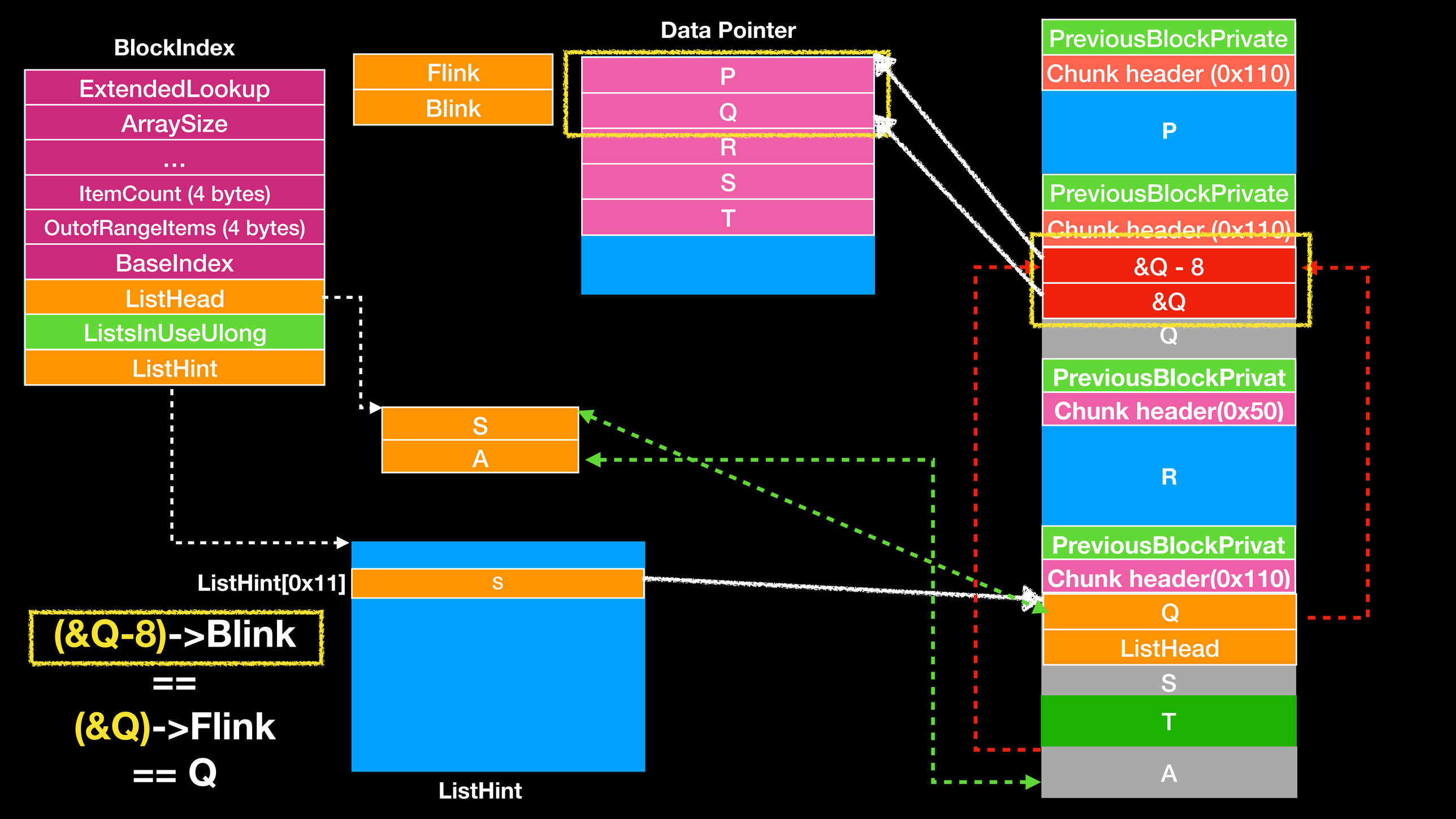

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
Let’s corrupt the heap !
ListHint[0x11]
s
ListHint
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
rivat
0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

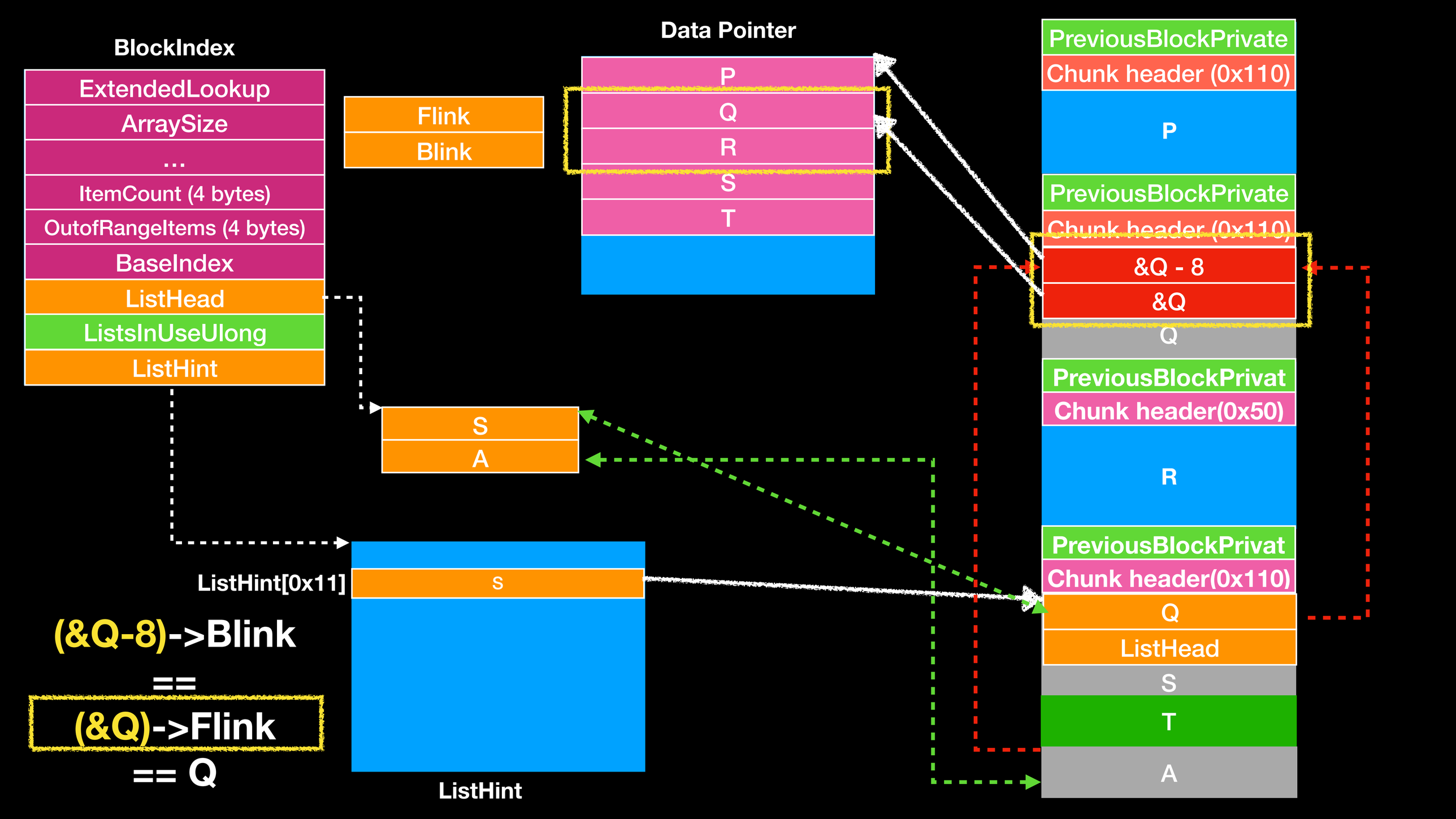

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
free(P)
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

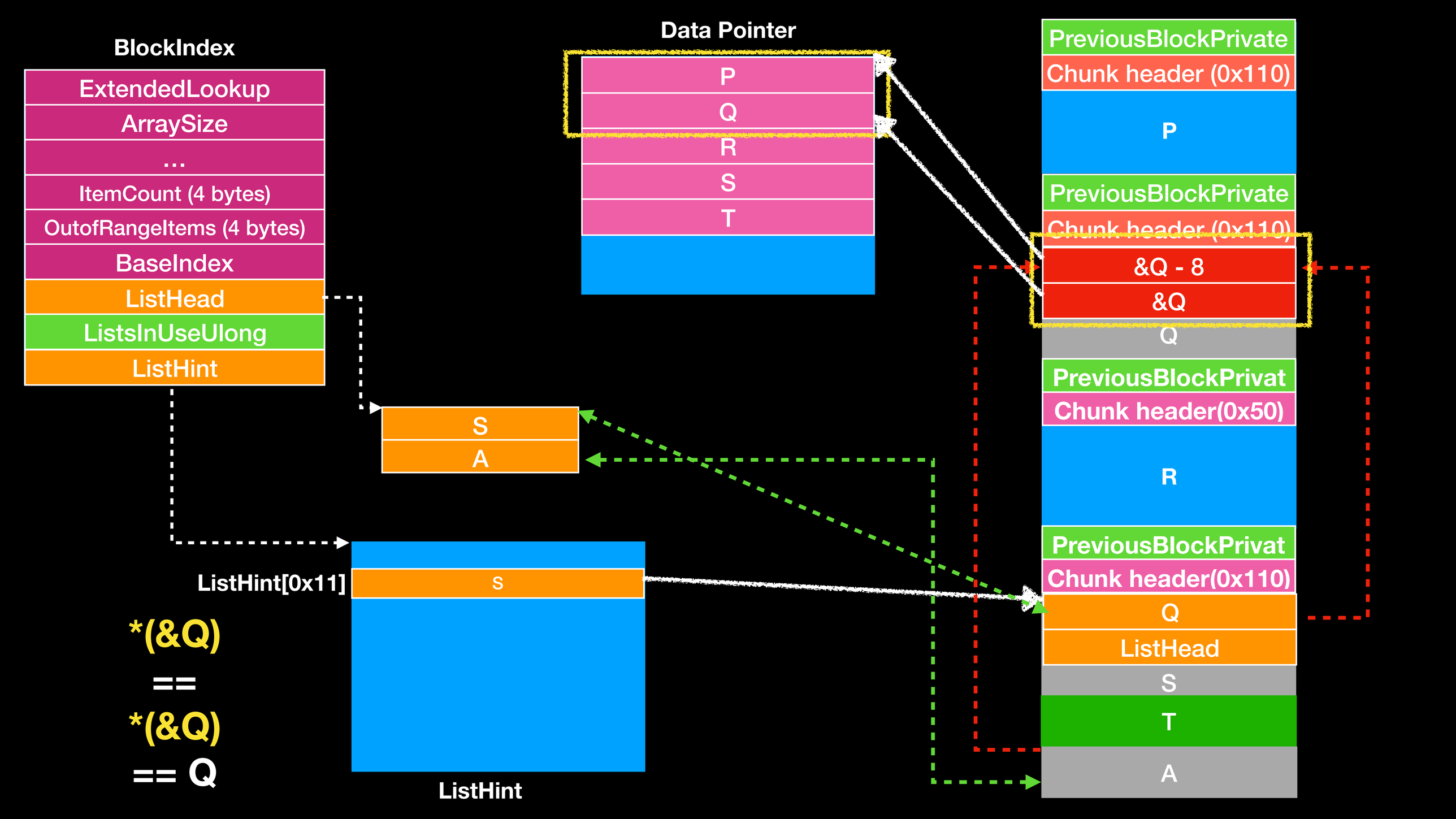

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Check next chunk is free
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

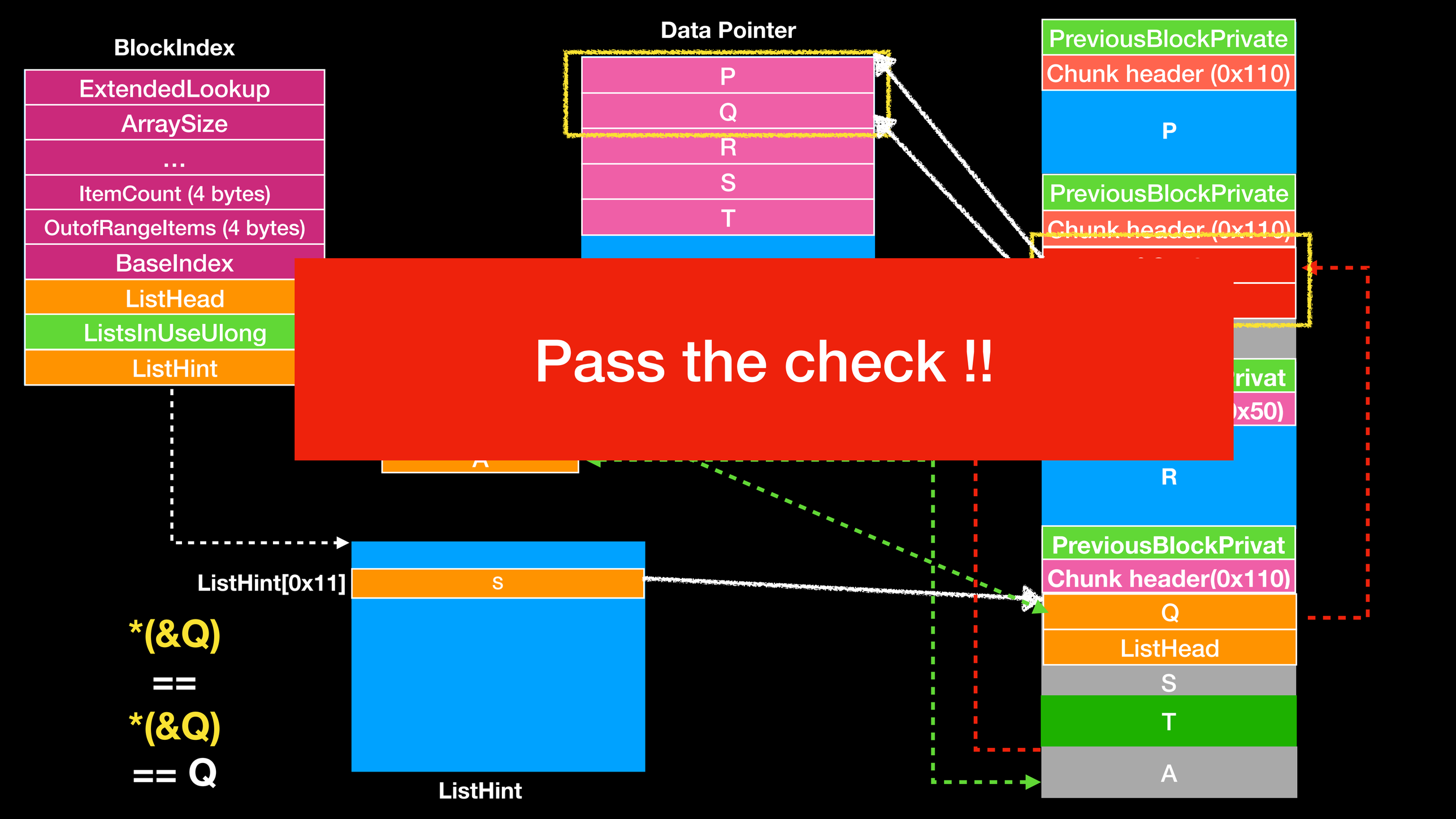

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Decode
chunk Q
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

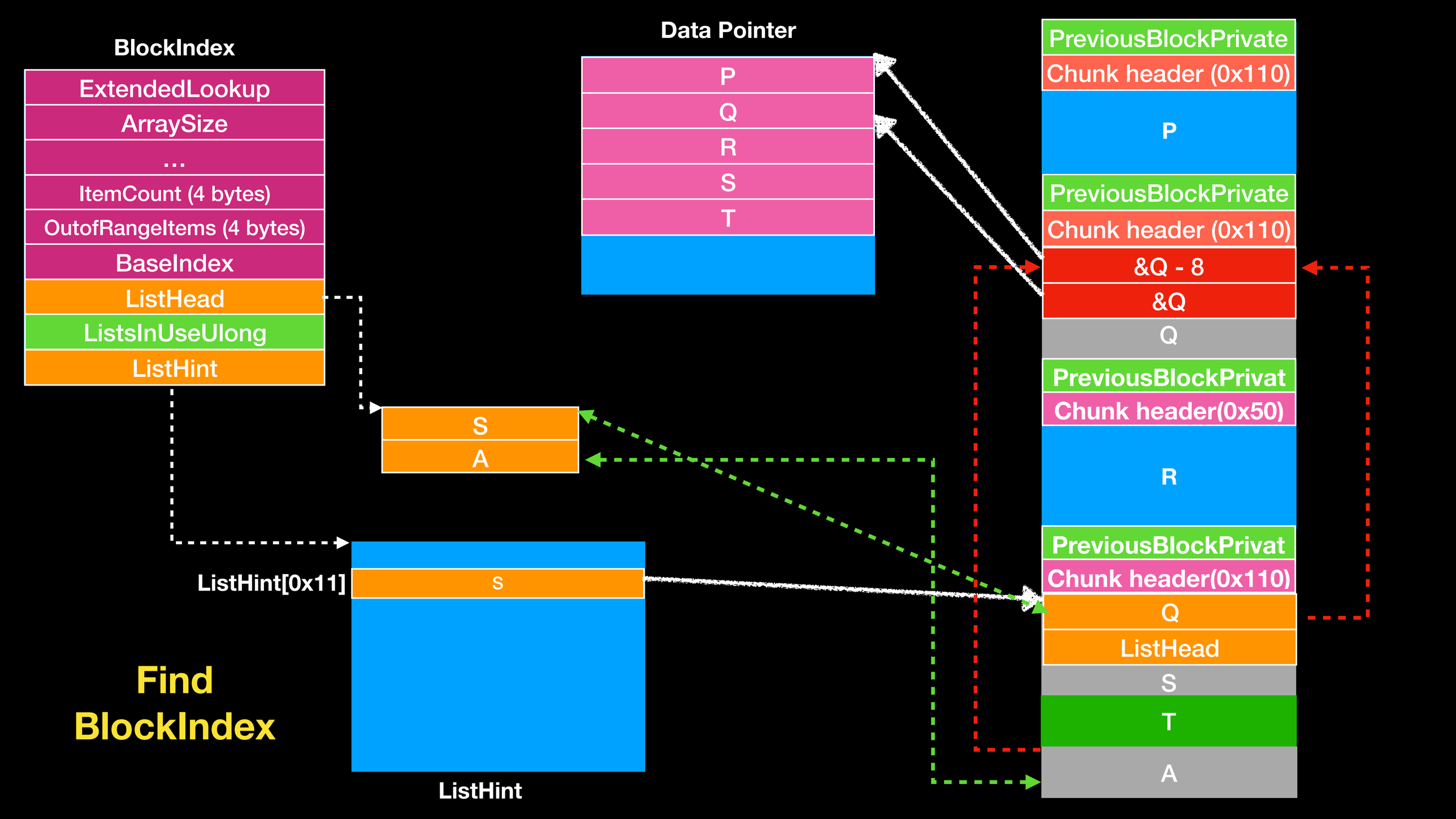

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
(&Q-8)->Blink
==
(&Q)->Flink
== Q
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

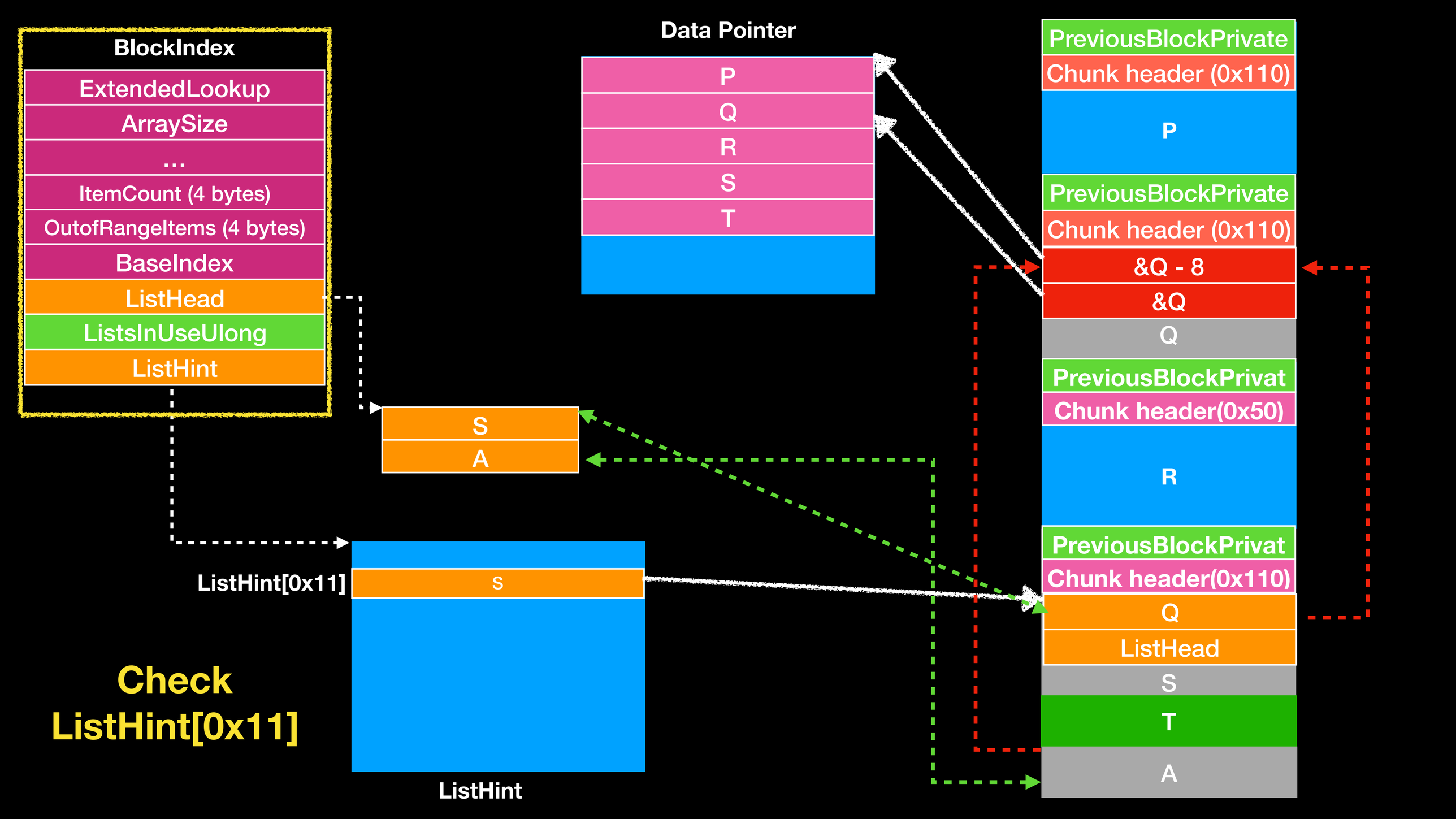

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
(&Q-8)->Blink
==
(&Q)->Flink
== Q
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

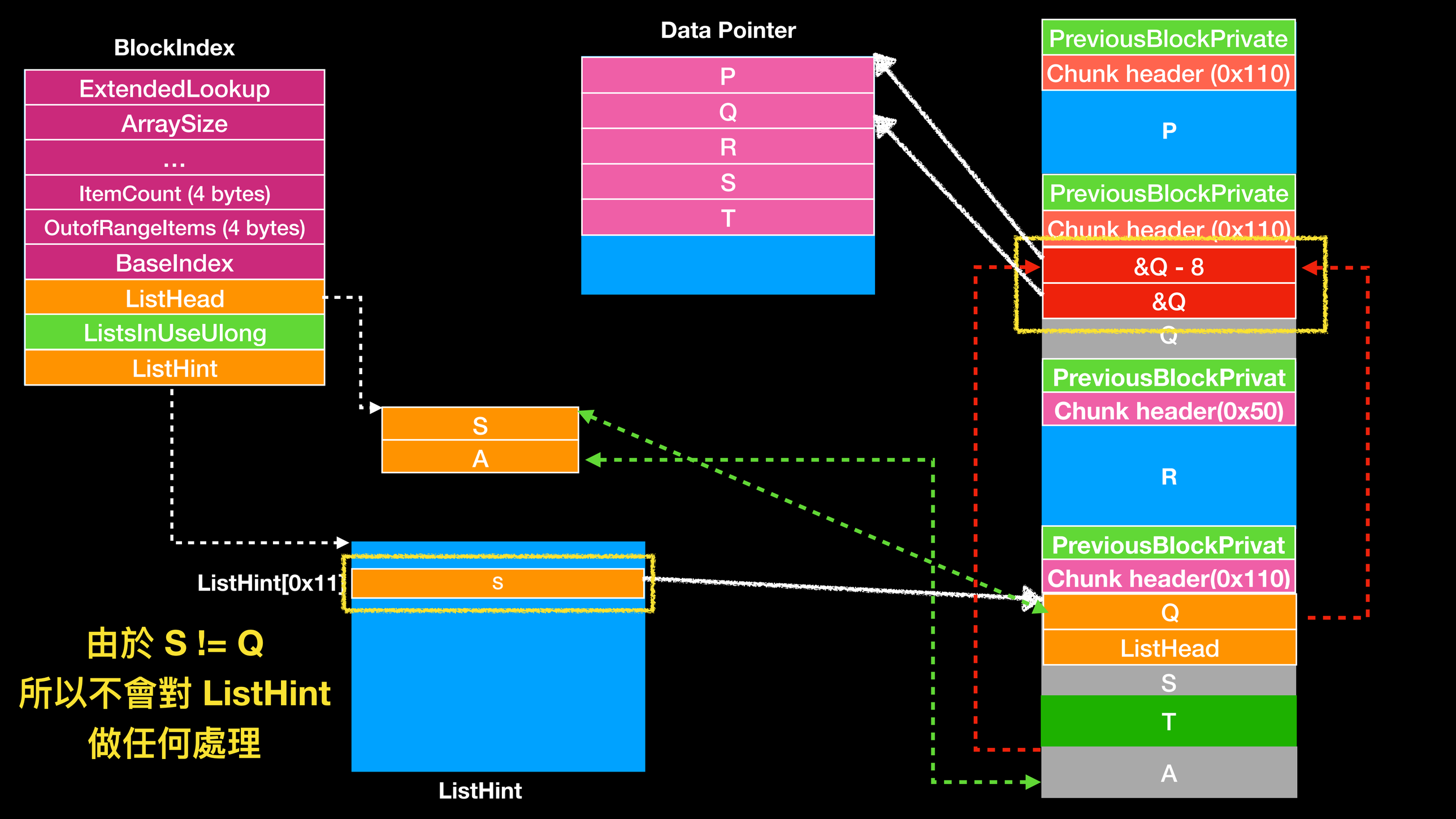

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
(&Q-8)->Blink
==
(&Q)->Flink
== Q
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

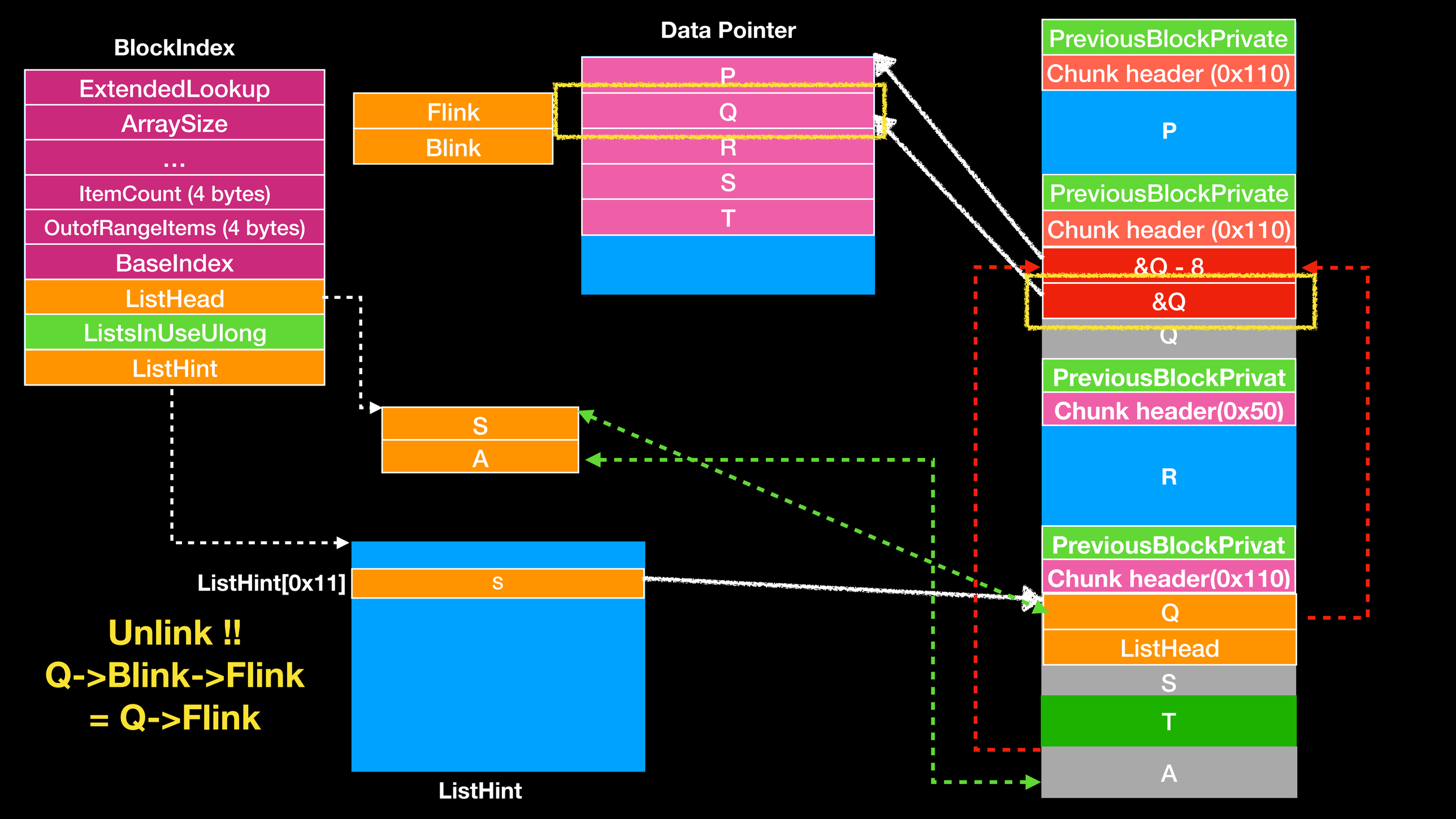

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
*(&Q)
==
*(&Q)
== Q
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

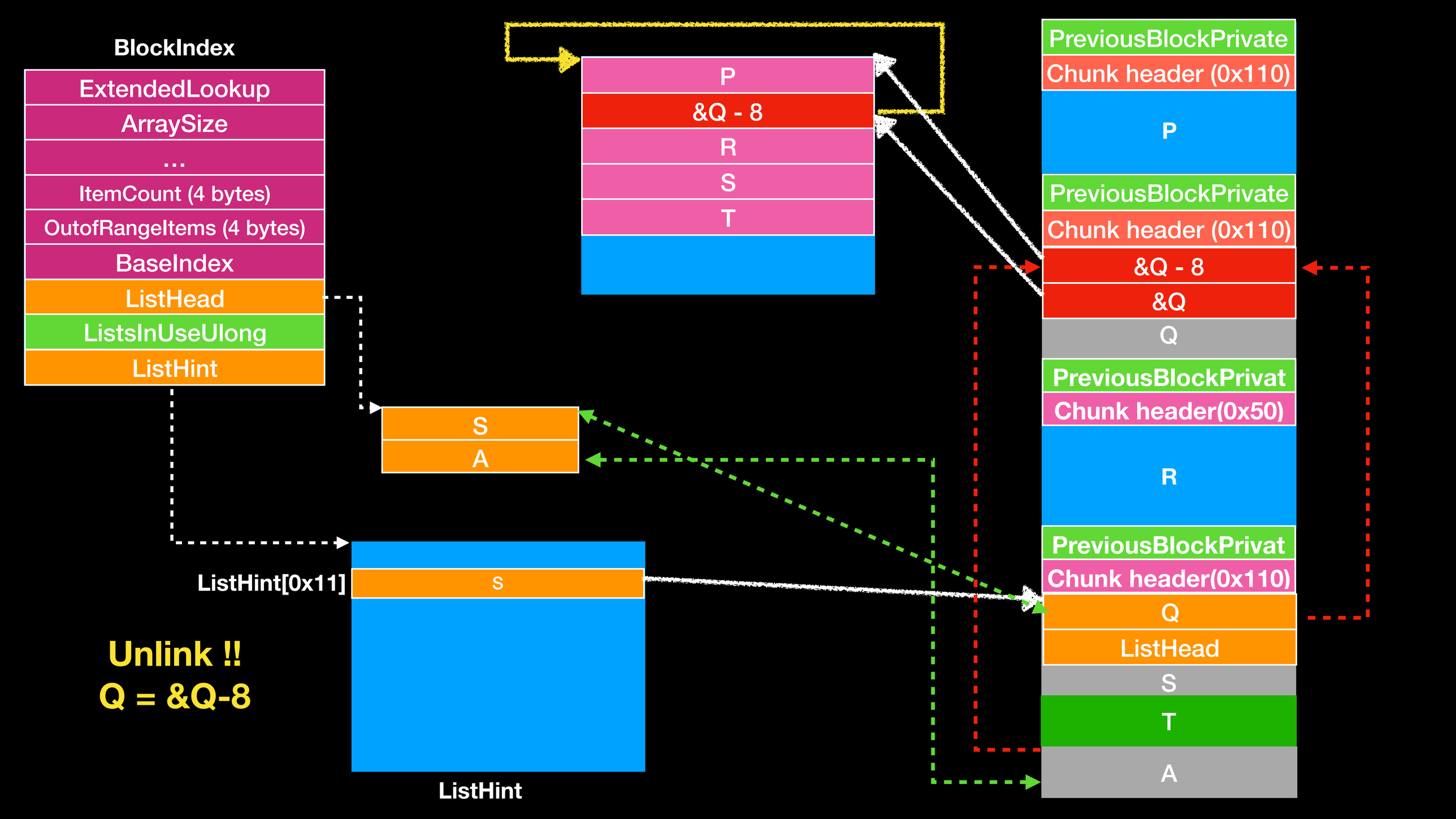

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
rivat
0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A
Pass the check !!
ListHint[0x11]
S
ListHint
*(&Q)
==
*(&Q)
== Q

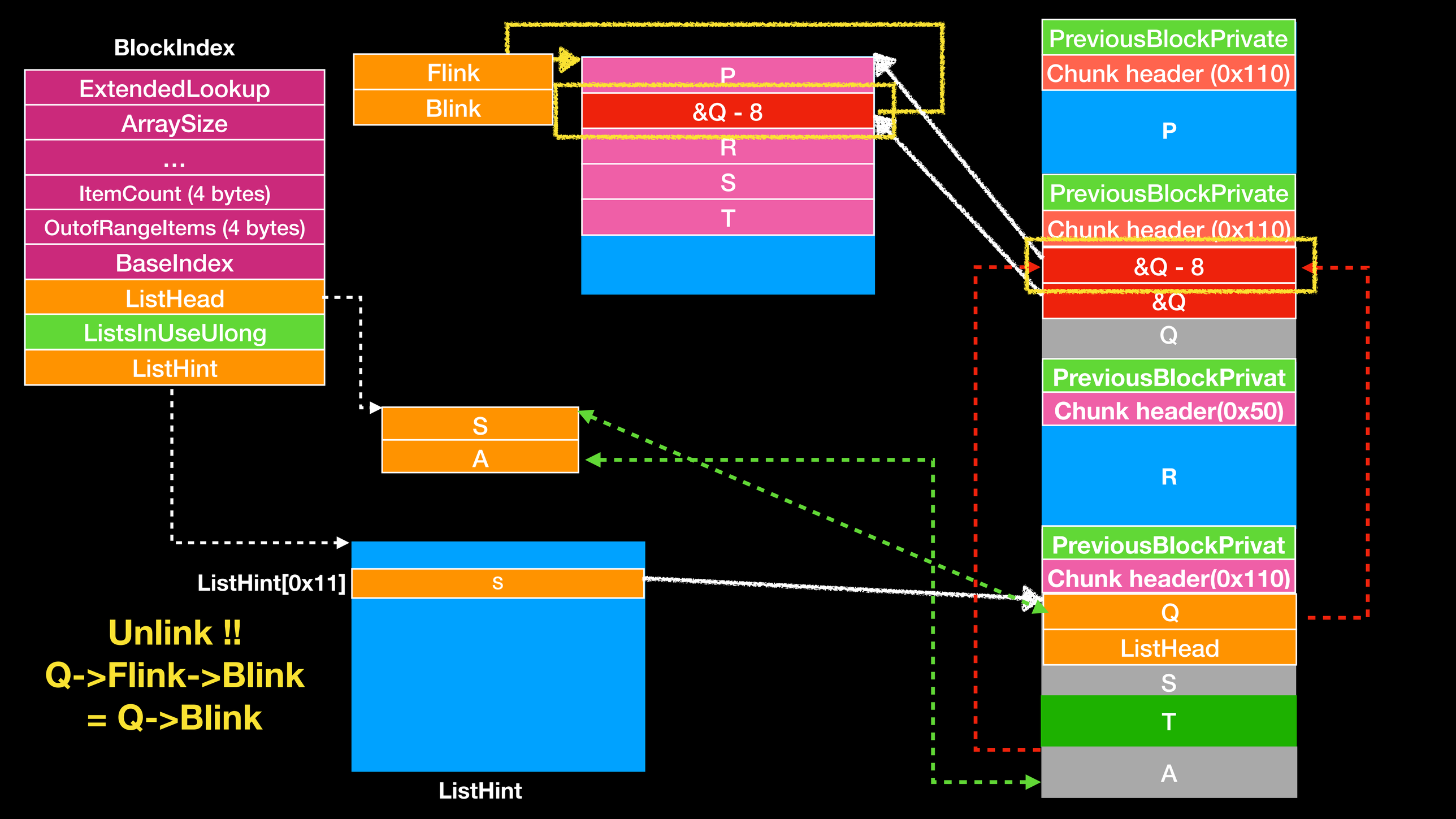

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Find
BlockIndex
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

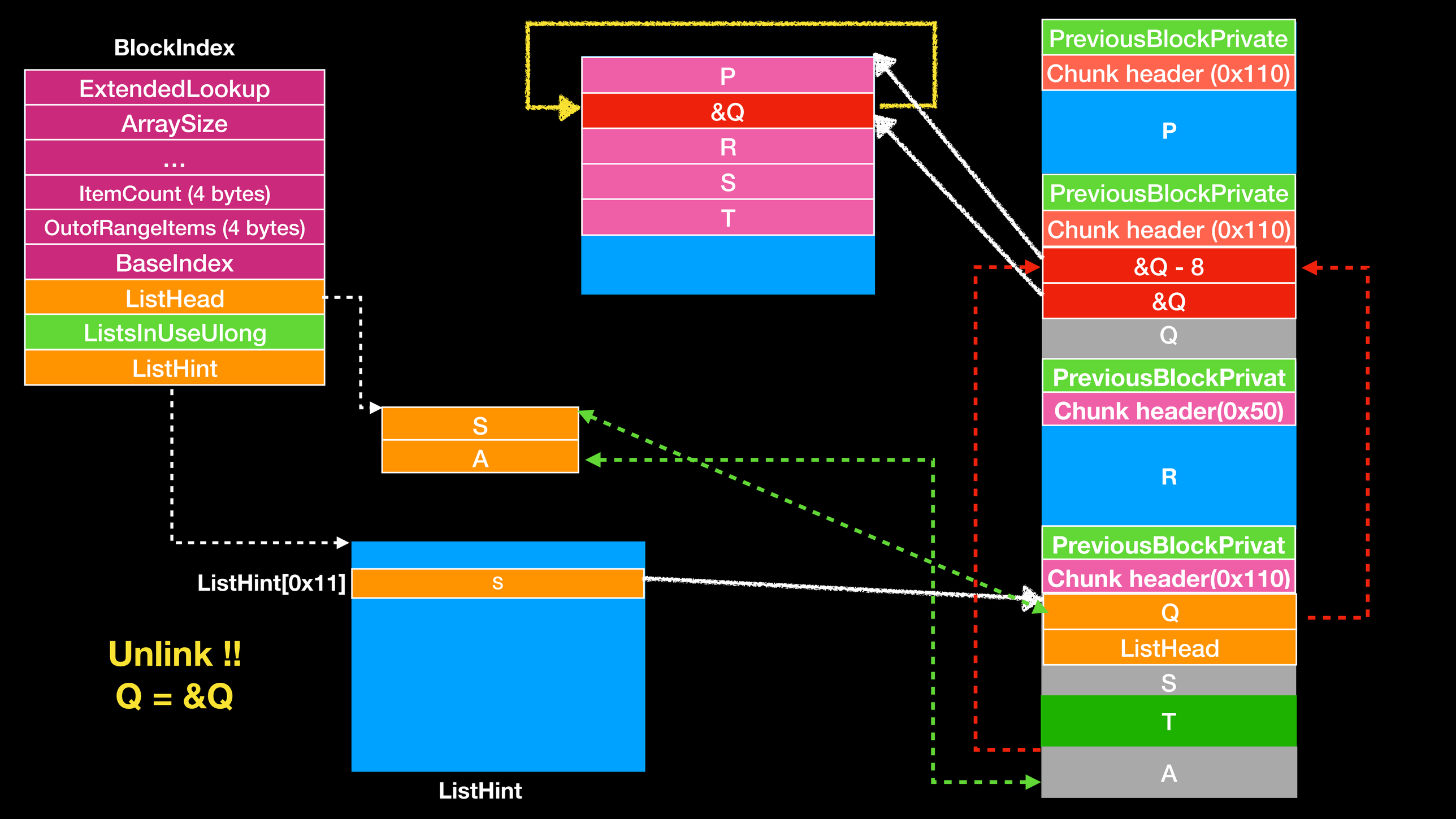

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Check
ListHint[0x11]
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

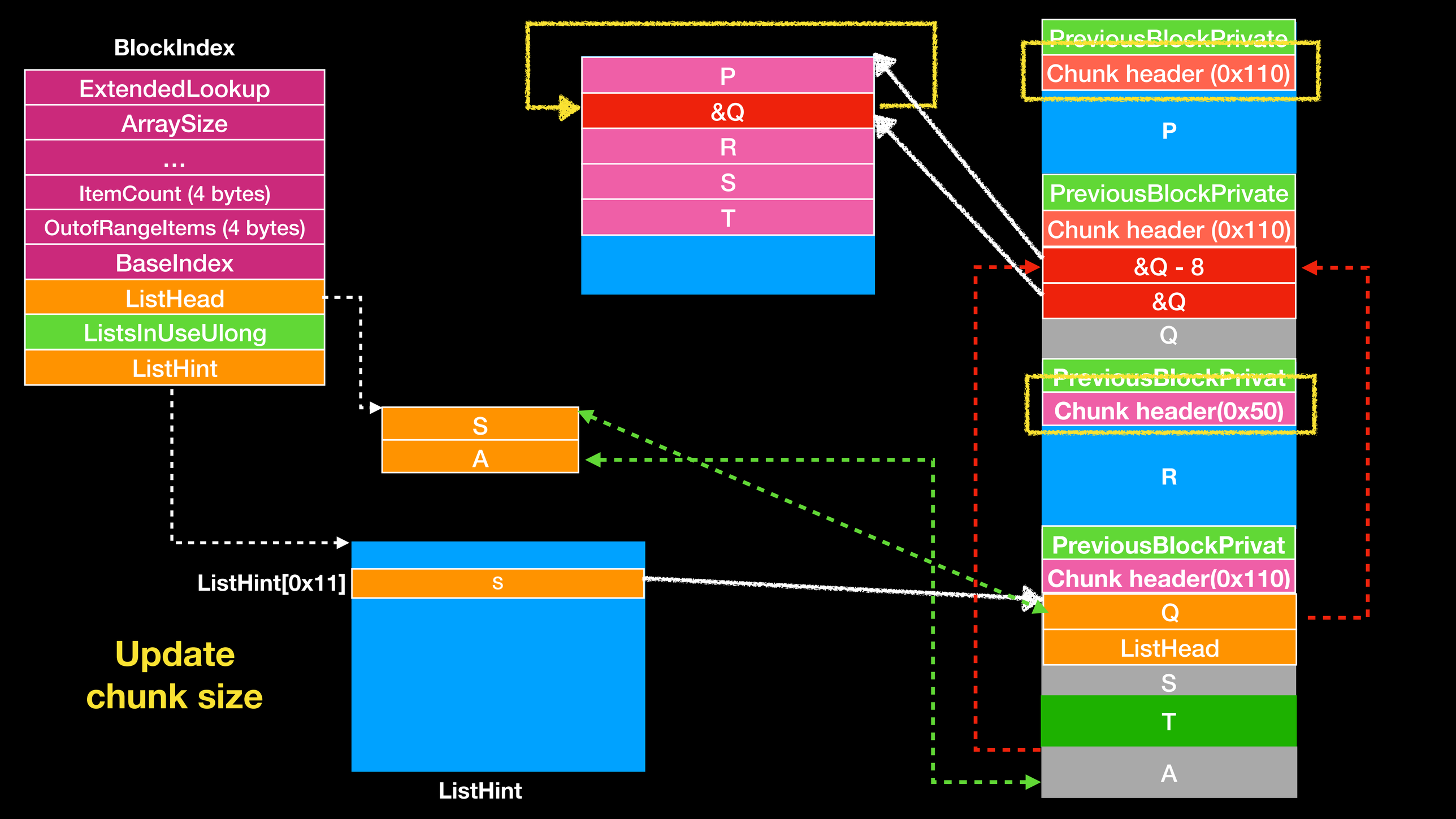

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A
由於 S != Q
所以不會對 ListHint
做任何處理

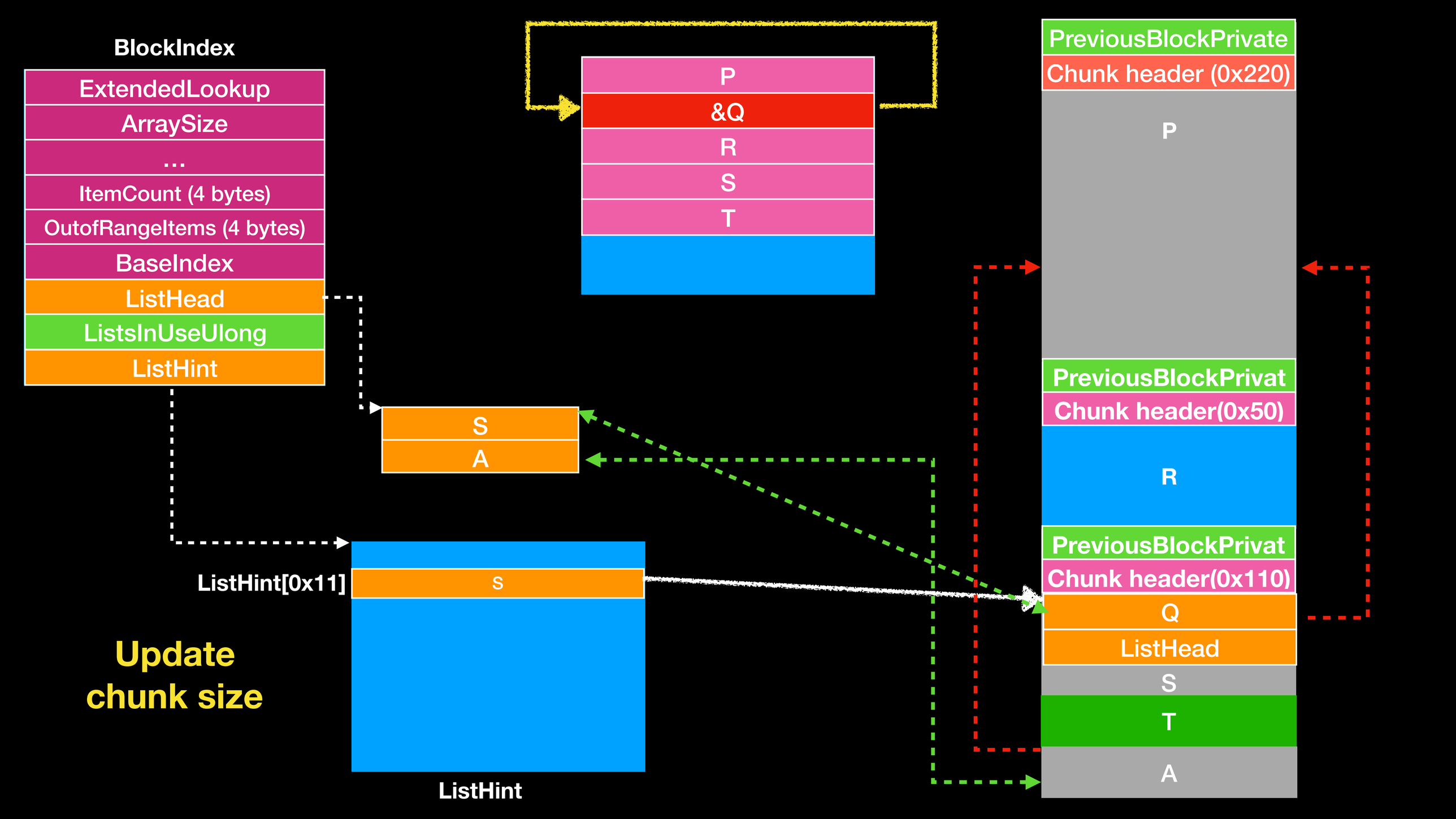

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Data Pointer
Flink
Blink
P
Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Unlink !!
Q->Blink->Flink
= Q->Flink
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

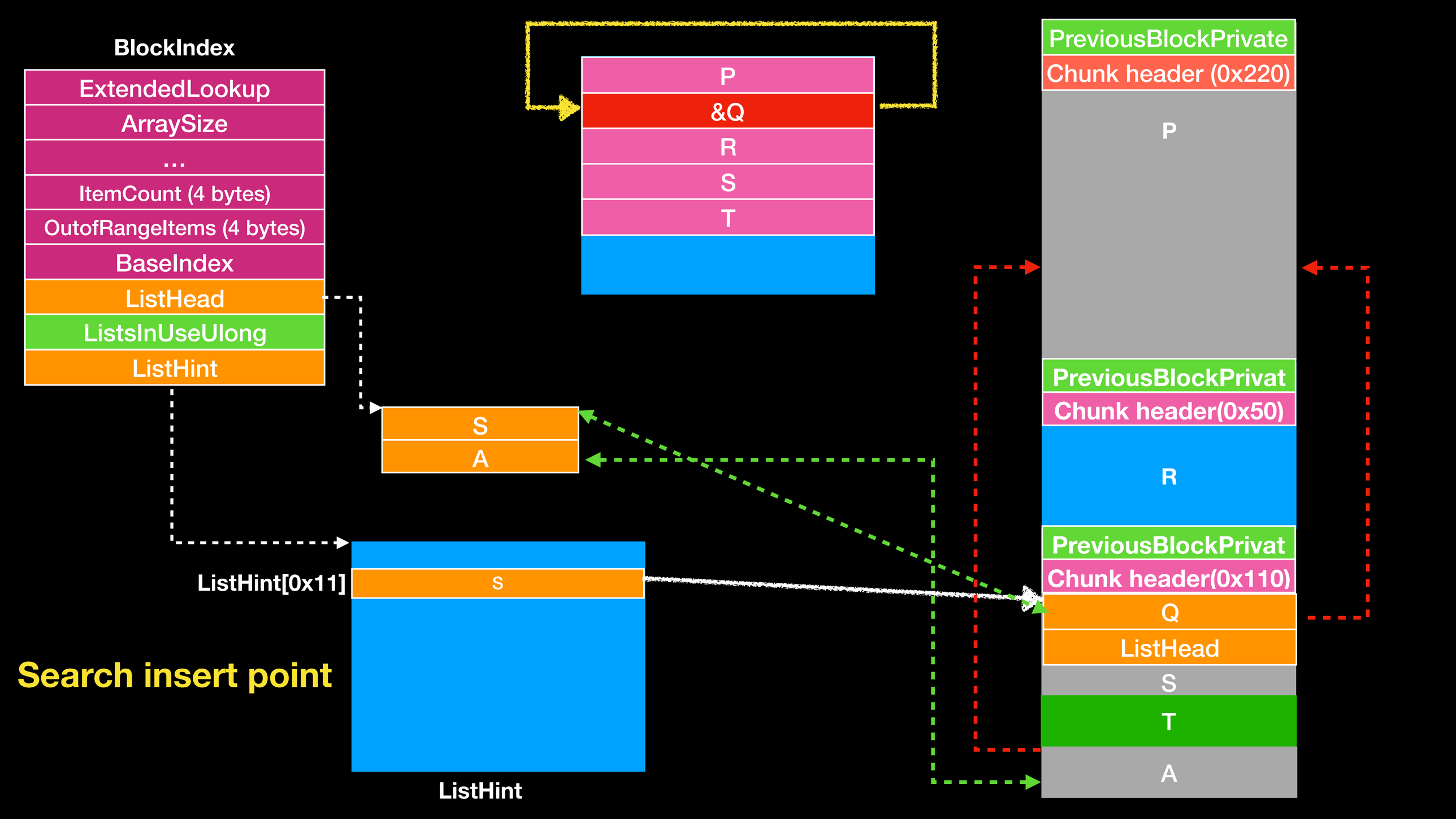

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q - 8
R
S
T
S
A
ListHint[0x11]
s
ListHint
Unlink !!
Q = &Q-8
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

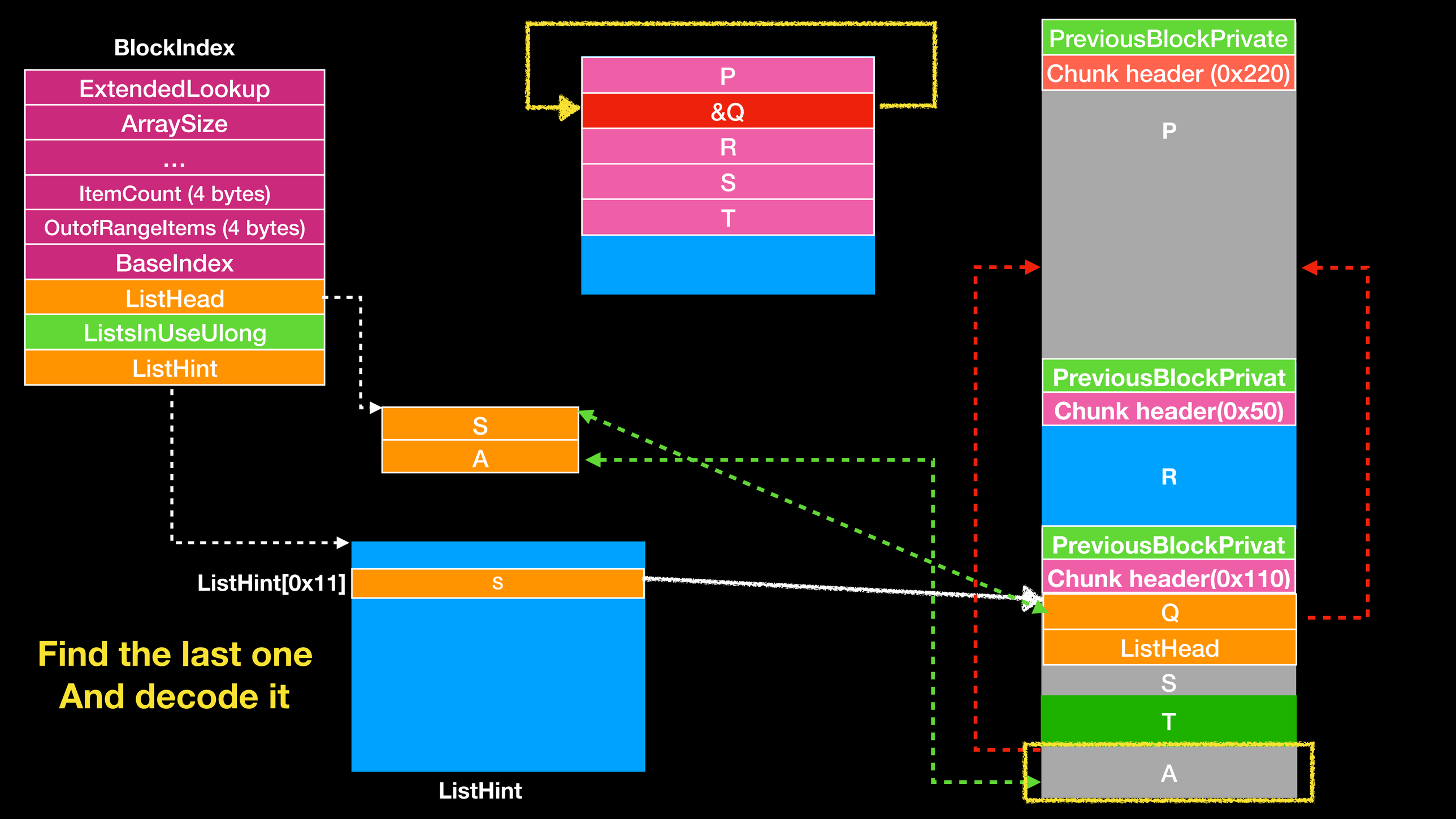

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
Flink
Blink
P
&Q - 8
R
S
T
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A
S
A
ListHint[0x11]
s
ListHint
Unlink !!
Q->Flink->Blink
= Q->Blink

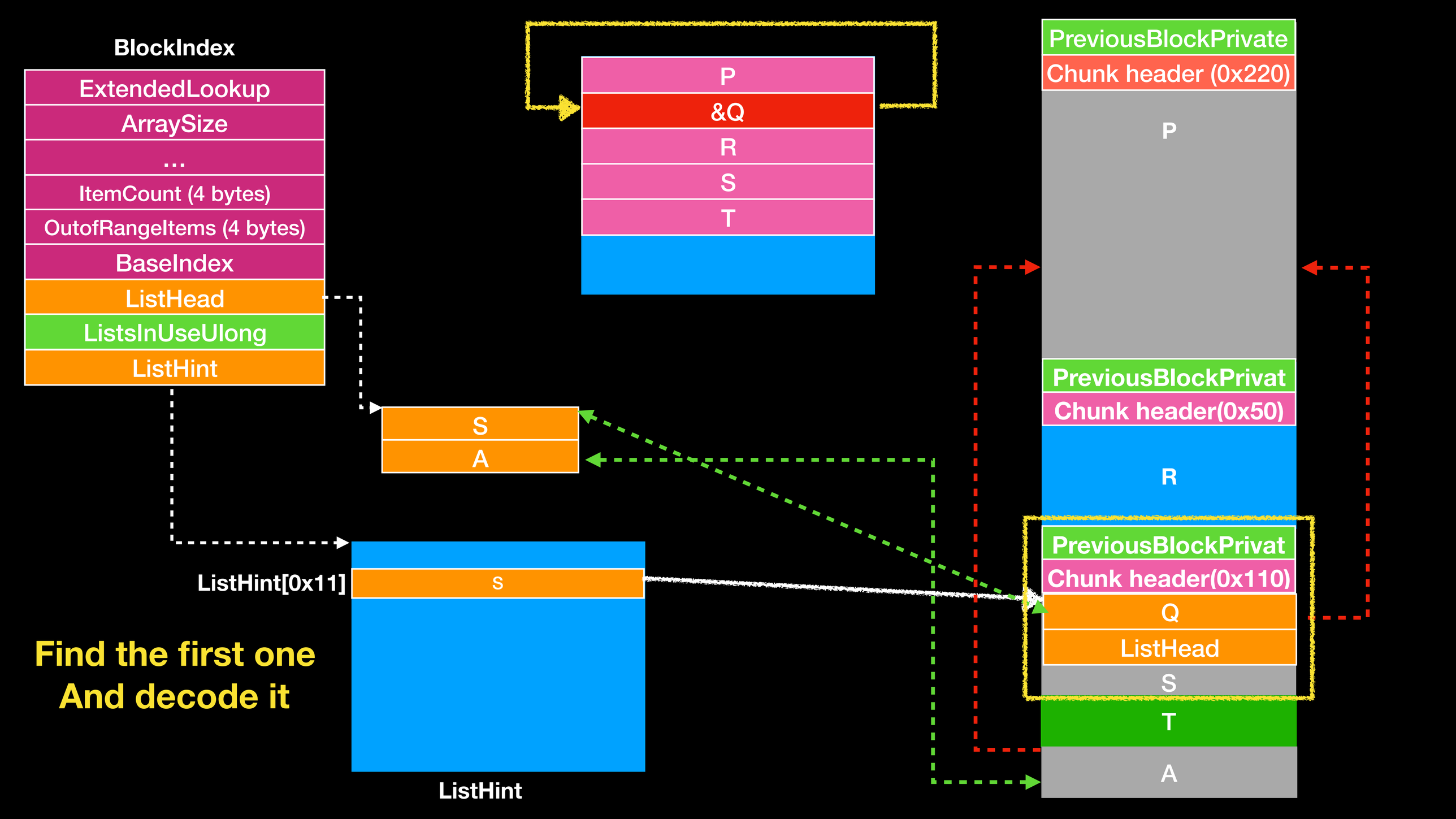

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Unlink !!
Q = &Q
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

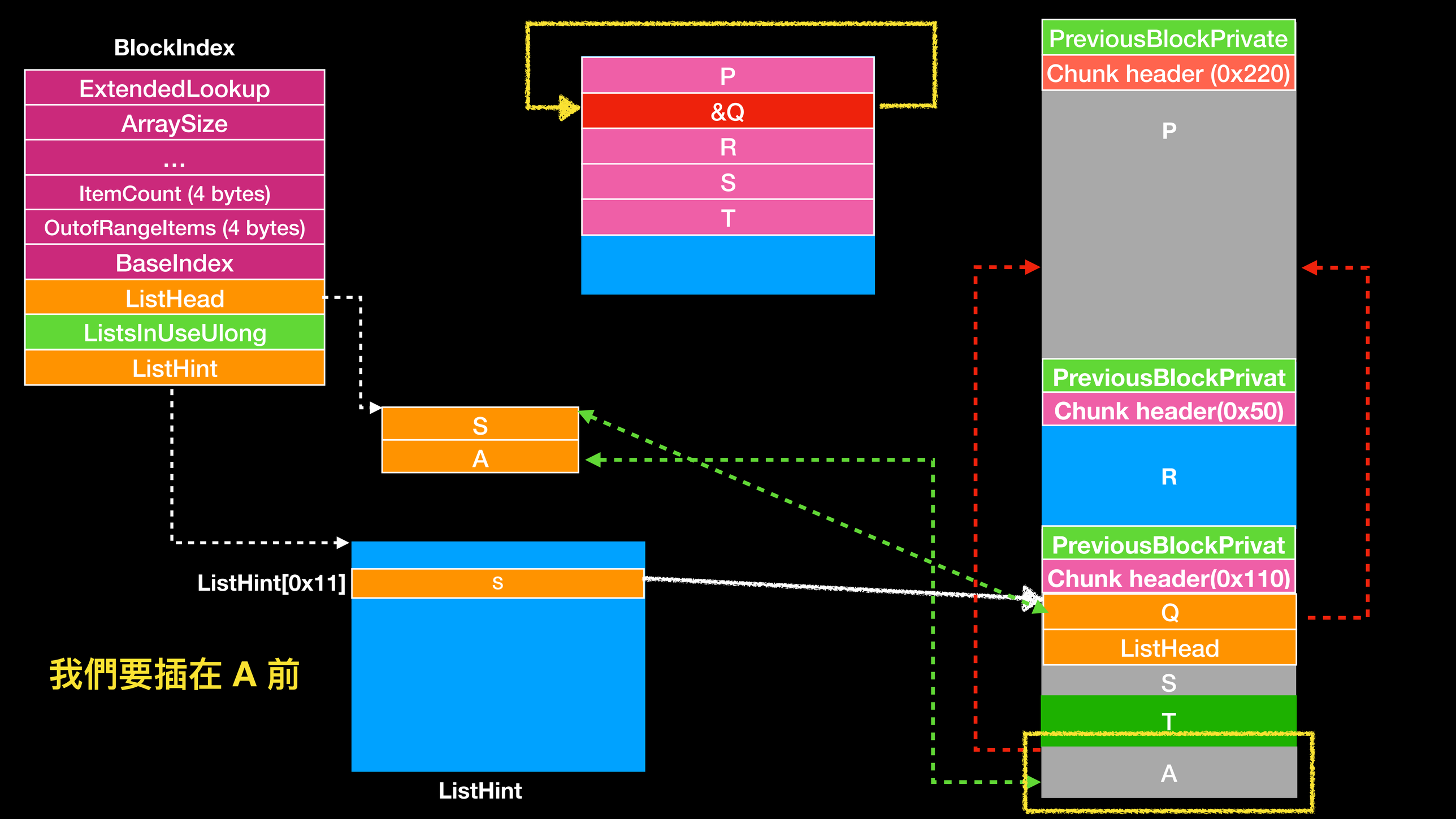

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Update
chunk size
PreviousBlockPrivate
Chunk header (0x110)
P
PreviousBlockPrivate
Chunk header (0x110)
&Q - 8
&Q
Q
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

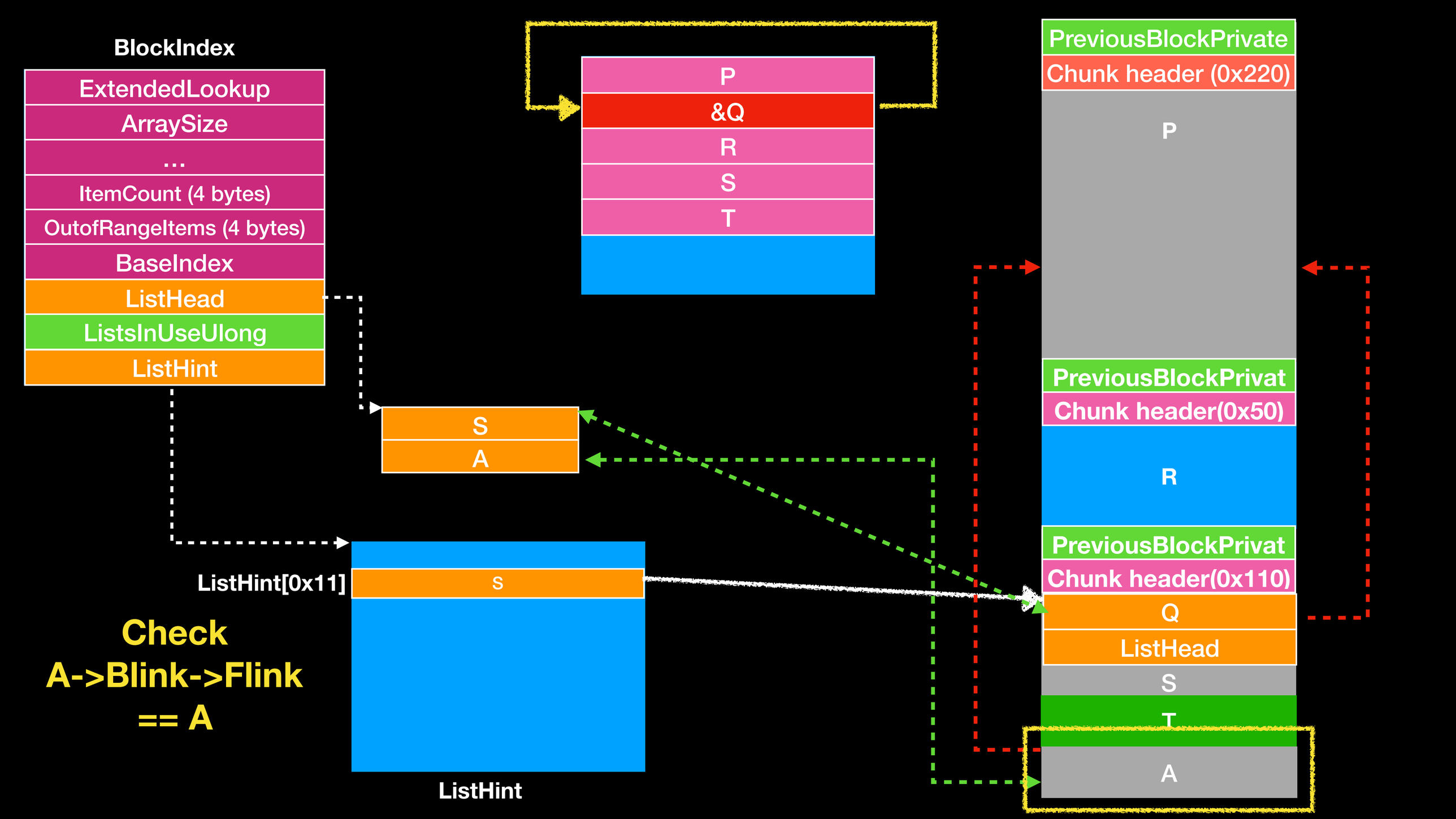

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Update
chunk size
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

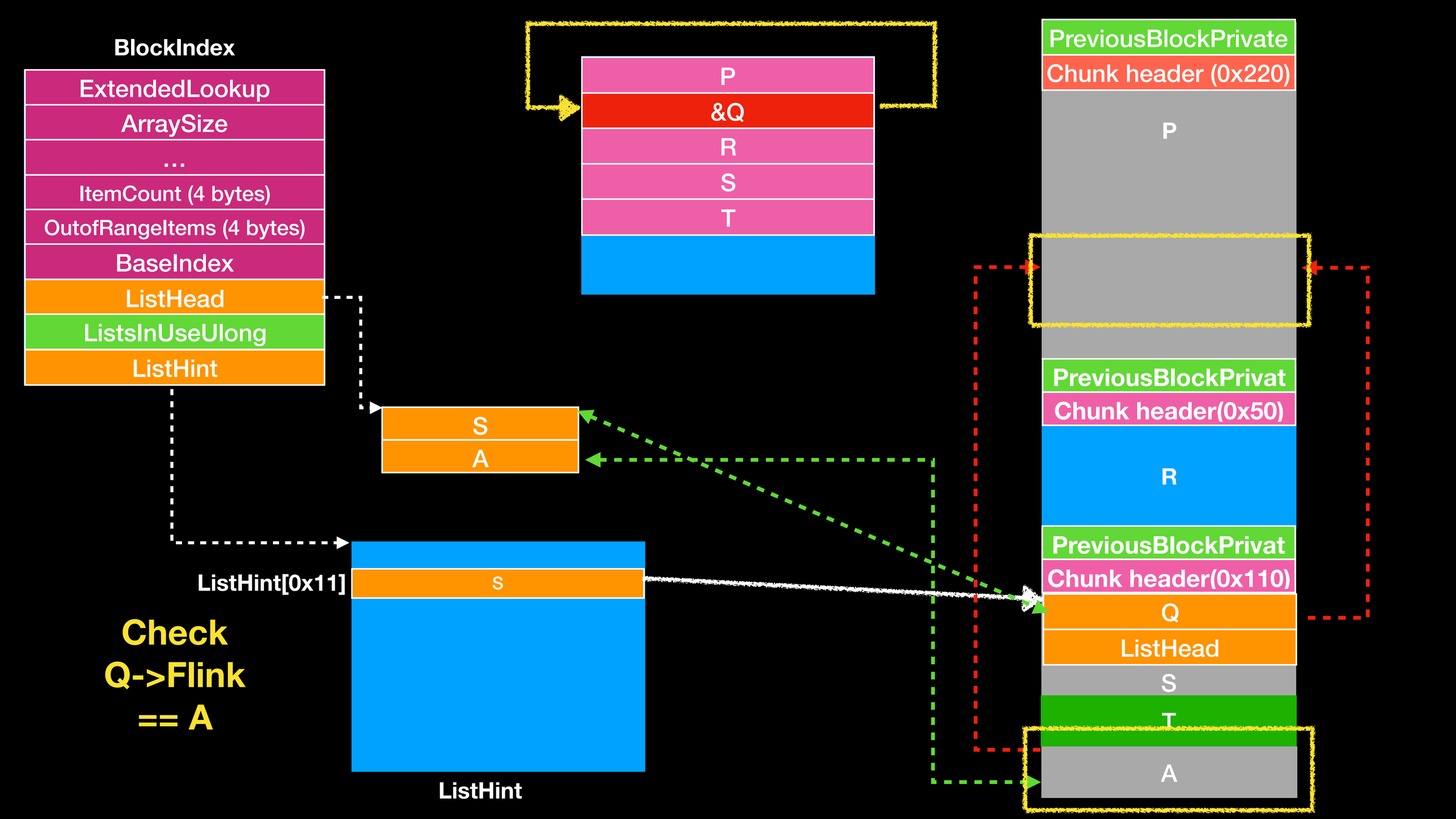

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Search insert point
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

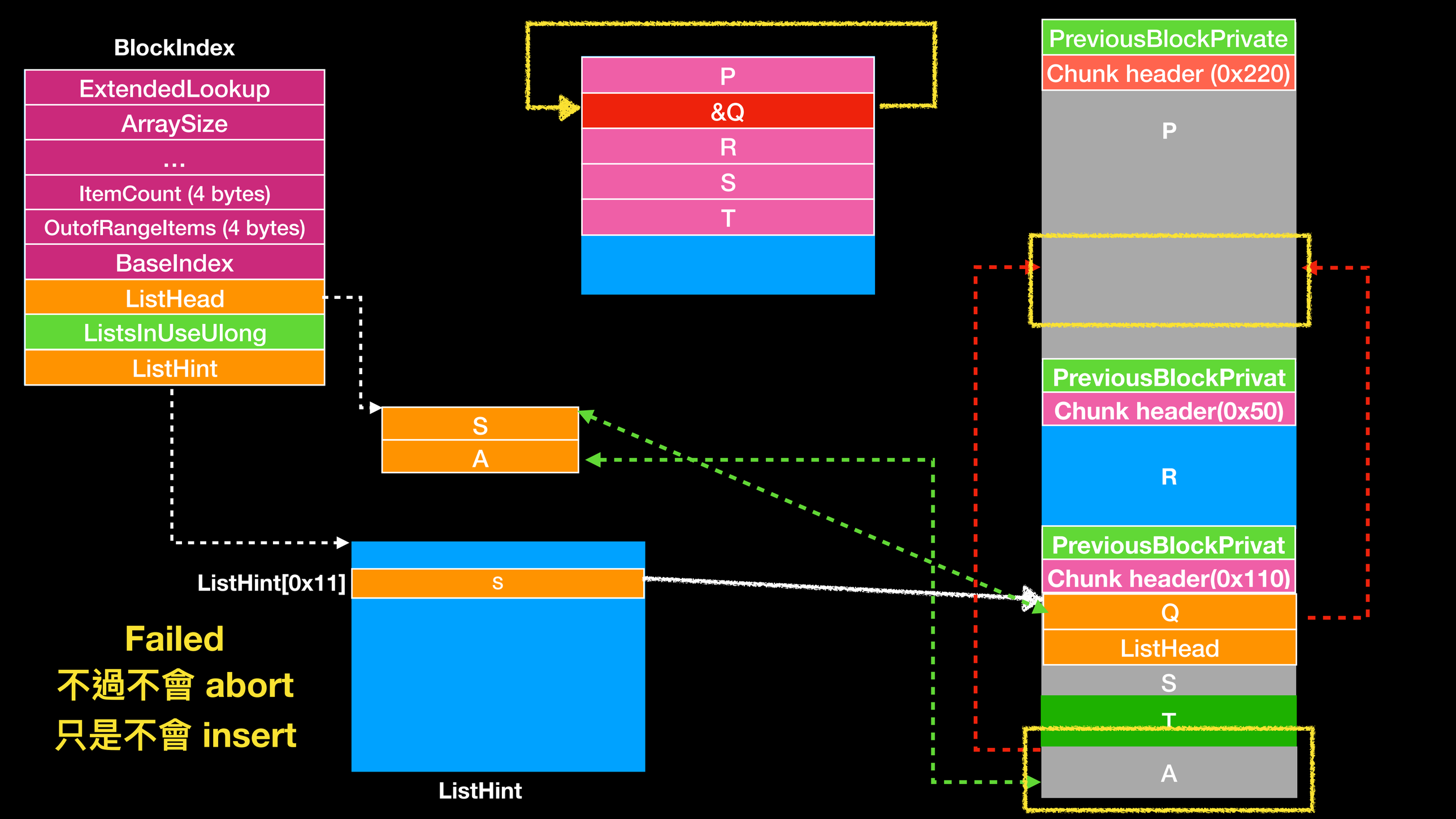

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Find the last one
And decode it
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

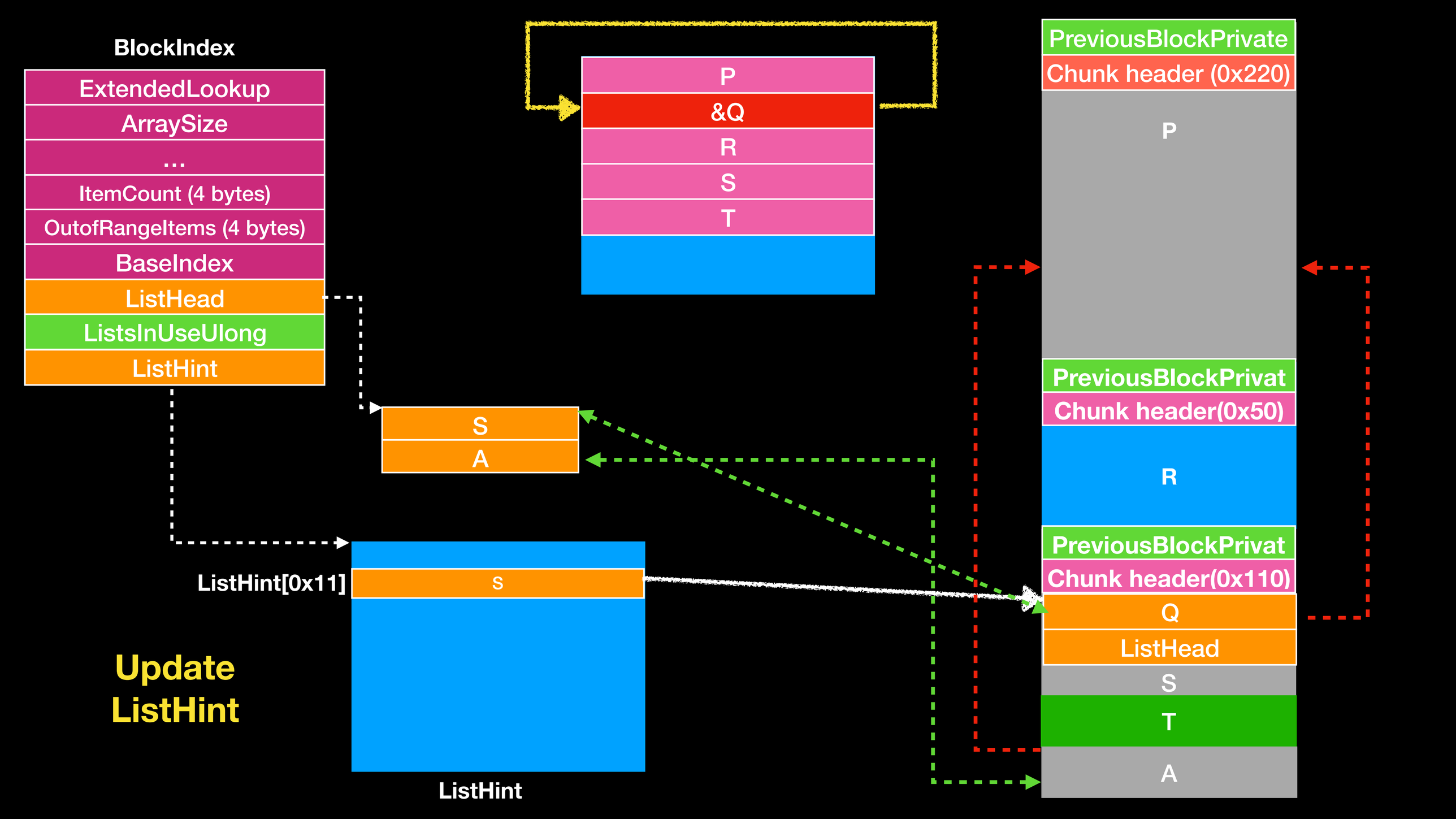

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A
Find the first one
And decode it

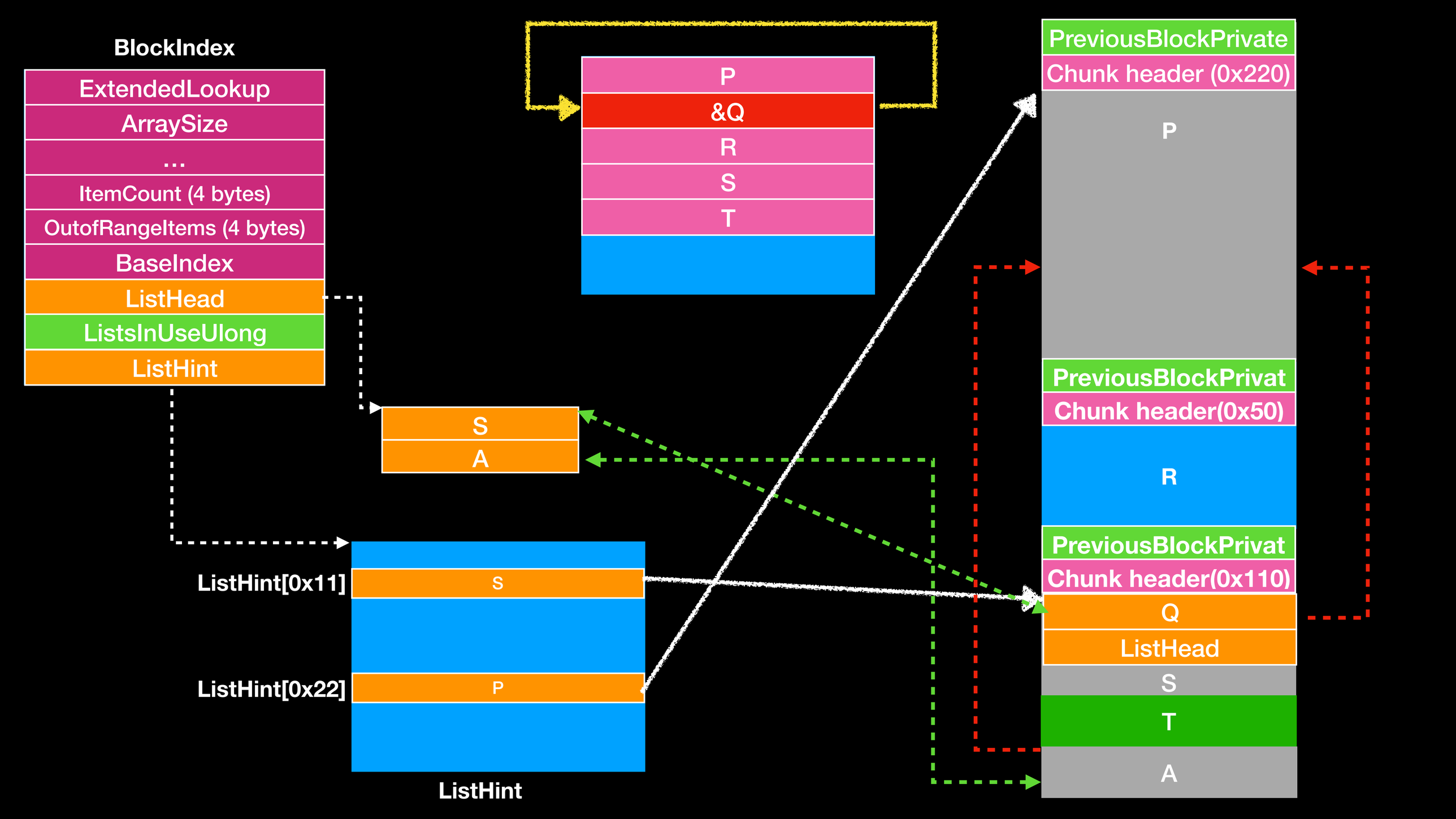

BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
我們要插在 A 前
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A


BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Check
A->Blink->Flink
== A
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A


BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Check
Q->Flink
== A
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A


BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Failed
不過不會 abort
只是不會 insert
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A


BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
s
ListHint
Update
ListHint
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A


BlockIndex
ExtendedLookup
ArraySize
...
ItemCount (4 bytes)
OutofRangeItems (4 bytes)
BaseIndex
ListHead
ListsInUseUlong
ListHint
P
&Q
R
S
T
S
A
ListHint[0x11]
ListHint[0x22]
S
P
ListHint
PreviousBlockPrivate
Chunk header (0x220)
P
PreviousBlockPrivat
Chunk header(0x50)
R
PreviousBlockPrivat
Chunk header(0x110)
Q
ListHead
S
T
A

Back-End Exploitation

- Edit Q
  - 我們可以把 Q 周圍的 pointer 寫掉
    - 最後可達成任意記憶體讀寫

Back-End Exploitation

- Edit Q
  - 我們可以把 Q 周圍的 pointer 寫掉
    - 最後可達成任意記憶體讀寫
    - 可對 Target 1 - 4 做寫入

Back-End Exploitation

- After arbitrary memory reading and writing
  - Leak
    - ntdll.dll
    - PebLdr
    - binary address
    - Kernel32.dll

Back-End Exploitation

- After arbitrary memory reading and writing
  - Leak
    - kernelbase
    - KERNELBASE!BasepFilterInfo
    - Stack address

Back-End Exploitation

- After arbitrary memory reading and writing
  - Leak
    - ntdll.dll
      - _HEAP_LOCK
        - _HEAP->LockVariable.Lock
        - CriticalSection->DebugInfo
        - 指向 ntdll

Back-End Exploitation

- After arbitrary memory reading and writing
  - Leak
    - ntdll!PebLdr
      - _PEB_LDR_DATA
      - 可找出所有 DLL 的位置
      - 不過缺點是最後一兩個 byte 可能是 0
      - 可先找出 binary base 從 IAT 找 kernel32

Back-End Exploitation

- After arbitrary memory reading and writing
  - Leak
    - BinaryBase
      - 從 IAT 找出 Kernel32 位置

Back-End Exploitation

- After arbitrary memory reading and writing
  - Leak
    - Kernel32
      - 重要 dll 很多常用 function 都在這 (CreateFile, ReadFile,WriteFile)
      - IAT
        - 獲得 kernelbase.dll 位置

Back-End Exploitation

- After arbitrary memory reading and writing
  - Leak
    - kernelbase
      - KERNELBASE!BasepFilterInfo
        - 指向 Heap 上的結構
        - 裡面有高機率會有 stack pointer
      - https://j00ru.vexillium.org/2016/07/disclosing-stack-data-from-the-default-heap-on-windows/

Back-End Exploitation

- After arbitrary memory reading and writing
  - Leak
    - 如果 BasepFilterInfo 中沒有 stack address
      - 因 BasepFilterInfo 內容會有 stack address 是因為未初始化因素，所以有機會沒有 stack address
      - 可以從 PEB 往前或往後算 1 個 page ，通常會是 TEB 位置，上面繪有 stack 尾段的位置，可以藉由該位置找出 retrun address

Back-End Exploitation

- After arbitrary memory reading and writing
  - Write
    - Return address
  - Control RIP
    - ROP to VirtualProtect/VirtualAlloc
    - Jmp to shellcode

Windows memory allocator

- Nt Heap
  - 後端管理器 (Back-End)
  - 前端管理器 (Front-End)

Windows memory allocator

- Nt Heap
  - 前端管理器 (Front-End)
    - 目前 win10 主要使用
      - LowFragmentationHeap
    - 在非 Debug 下才會 enable
    - Size < 0x4000

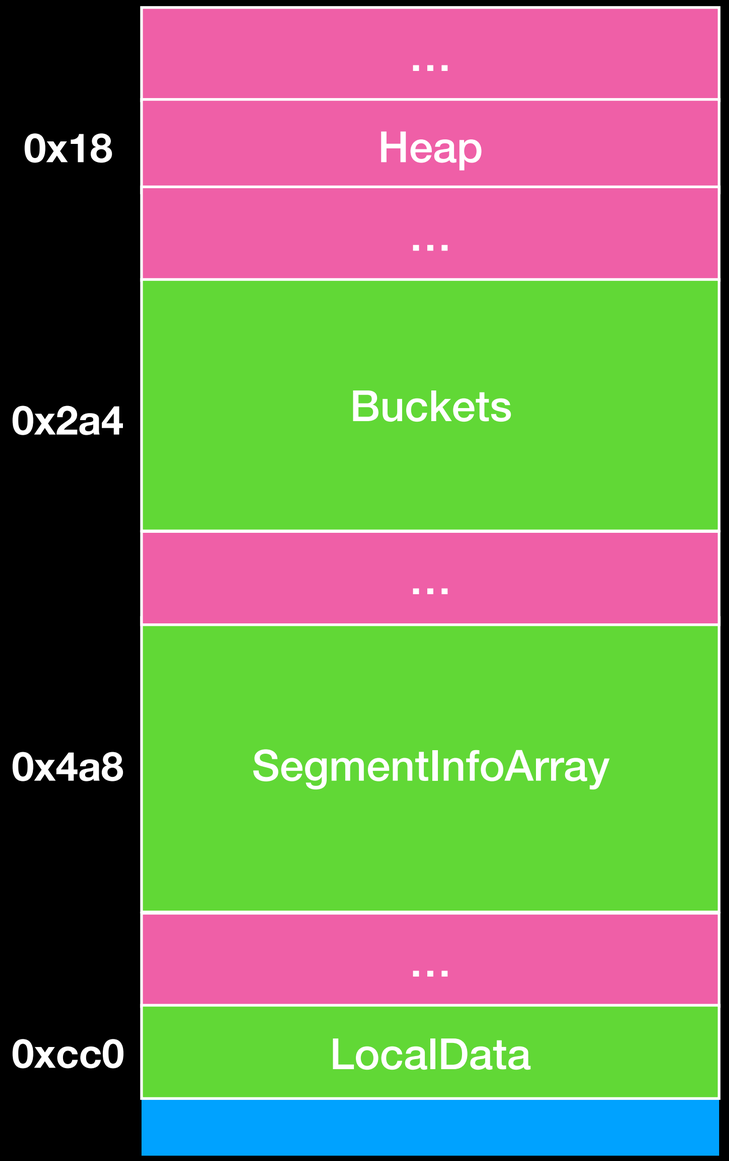
Windows memory allocator

- Nt Heap
  - 前端管理器 (Front-End)
    - Data Structure
    - 分配機制

Windows memory allocator

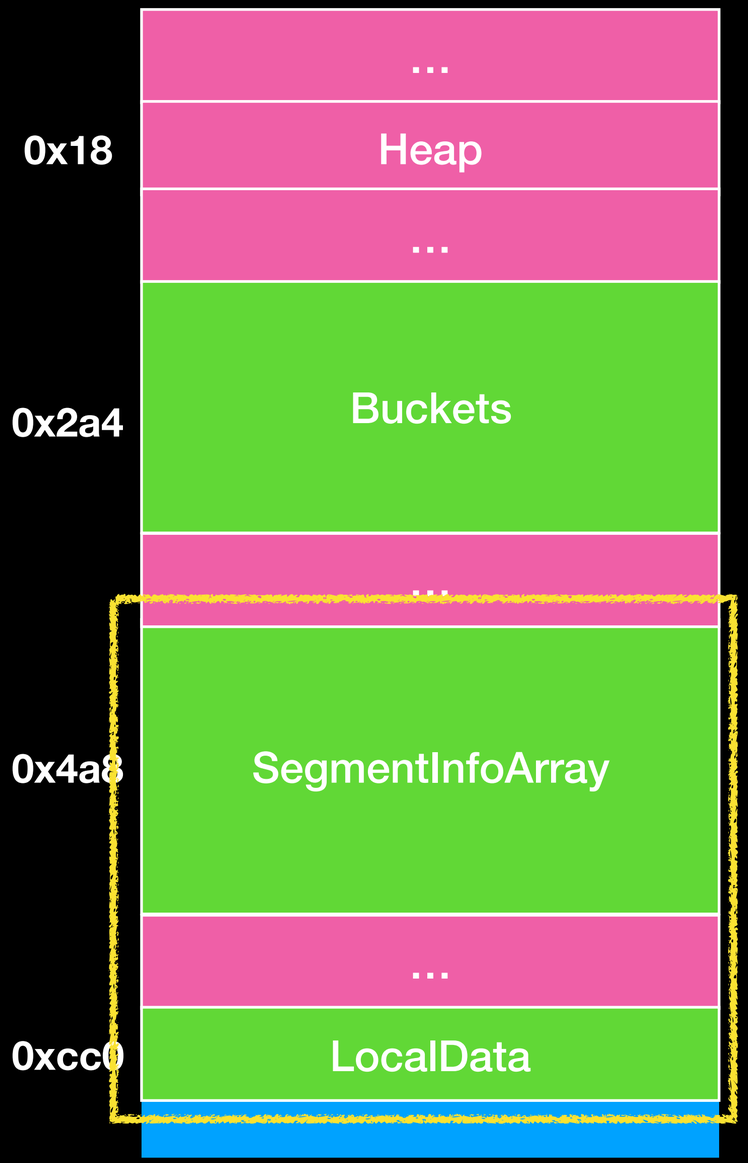
- Nt Heap
  - 前端管理器 (Front-End)
    - Data Structure
    - 分配機制

LFH



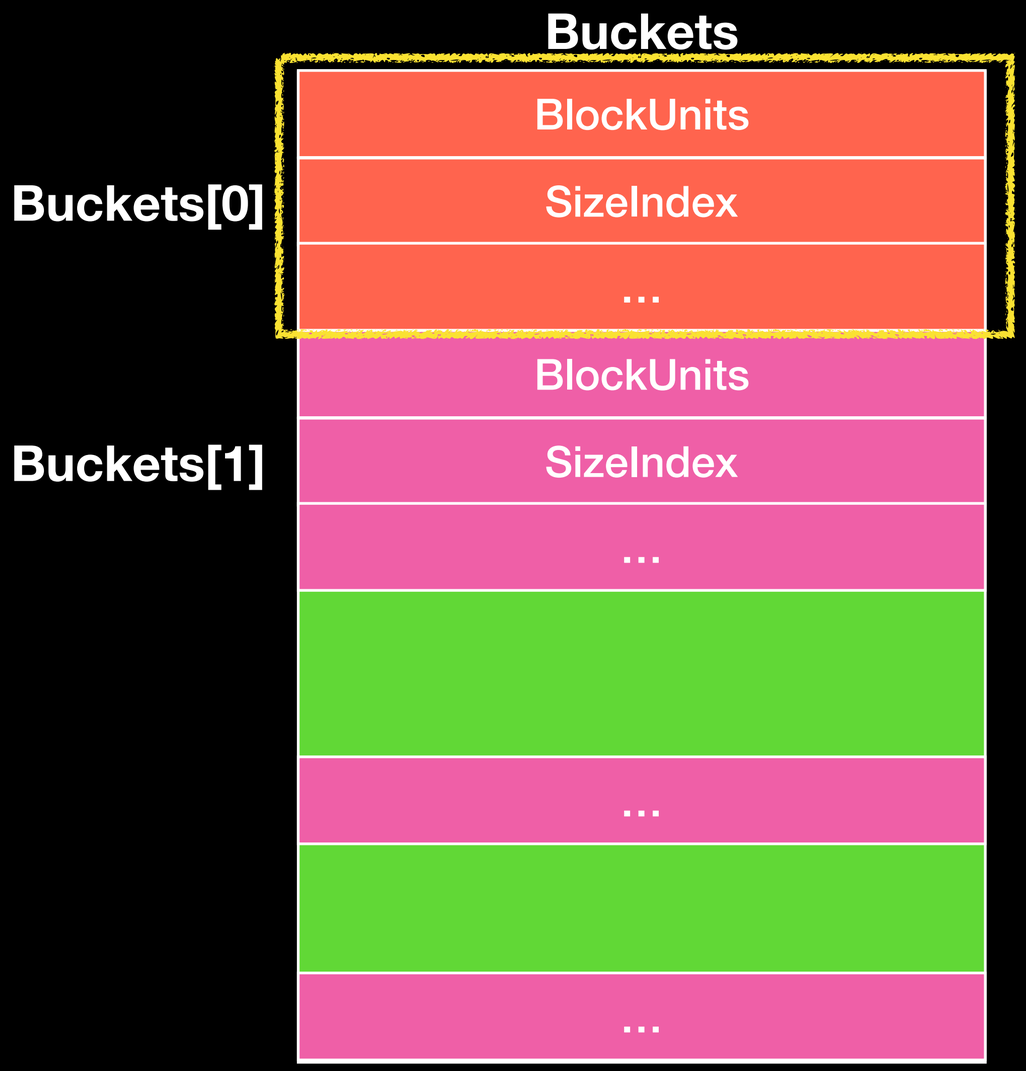
- FrontEndHeap (_LFH_HEAP)
  - HEAP (_HEAP)
    - 指向對應到的 _HEAP
  - Buckets (_HEAP_BUCKET)
    - 用來尋找配置大小對應到 Block 大小的陣列結構

LFH



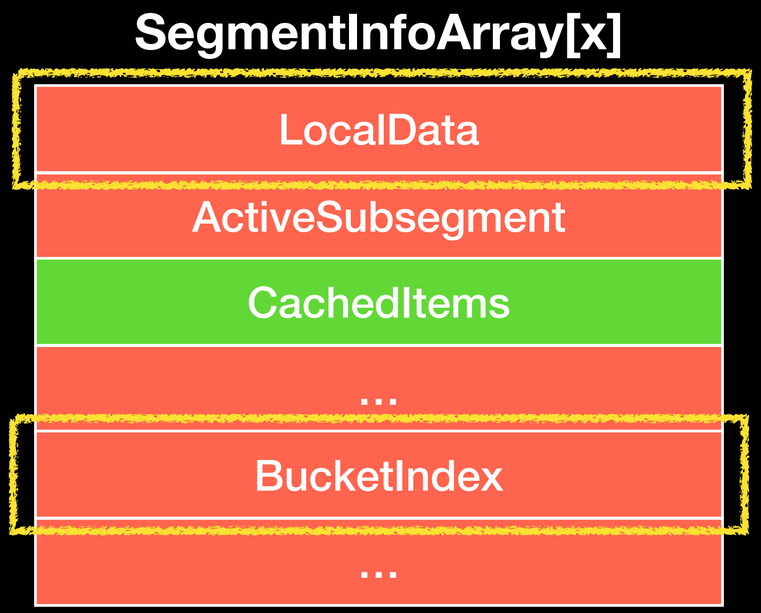
- FrontEndHeap (_LFH_HEAP)
  - SegmentInfoArray (_HEAP_LOCAL_SEGMENT_INFO)
    - _HEAP_LOCAL_SEGMENT_INFO array，不同大小對應到不同的 Segment_info 結構，主要管理對應到的 Subsegment 的資訊
  - LocalData (_HEAP_LOCAL_DATA)
    - 其中有個欄位指向 LFH 本身，通常用來找回 LFH

LFH



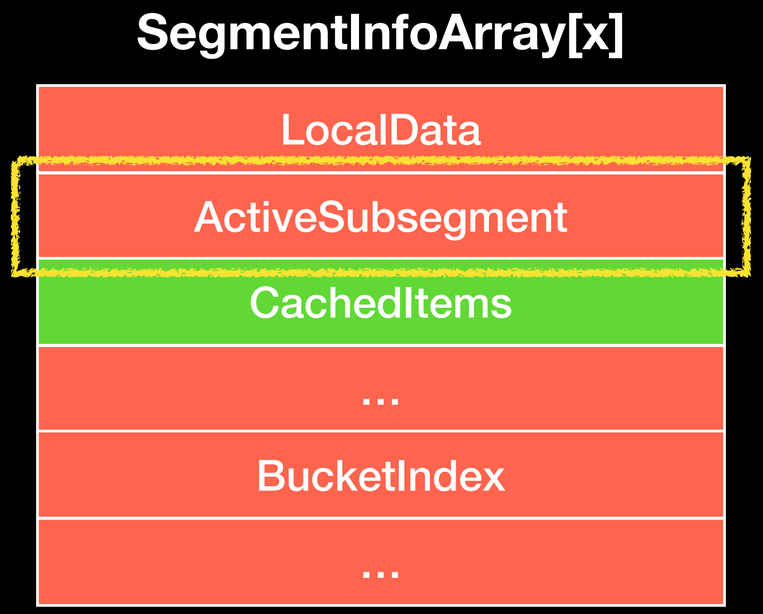
- Buckets (_HEAP_BUCKET)
  - BlockUnits
    - 要分配出去的一個 block 大小 >> 4
  - SizeIndex
    - 使用者需要的大小 >> 4

LFH



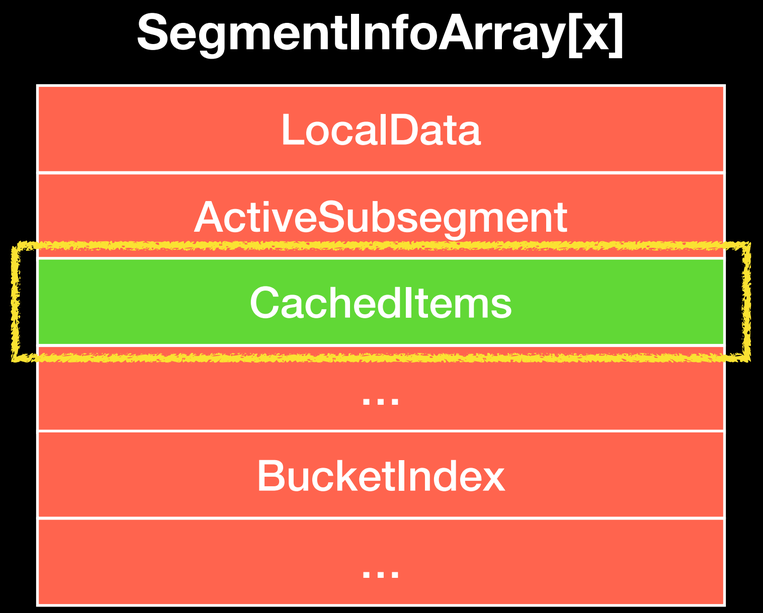
- SegmentInfoArray (_HEAP_LOCAL_SEGMENT_INFO)
  - LocalData (_HEAP_LOCAL_DATA)
    - 對應到 _LFH_HEAP->LocalData 方便從 SegmentInfo 找回 _LFH_HEAP
  - BucketIndex
    - 對應到的 BucketIndex

LFH



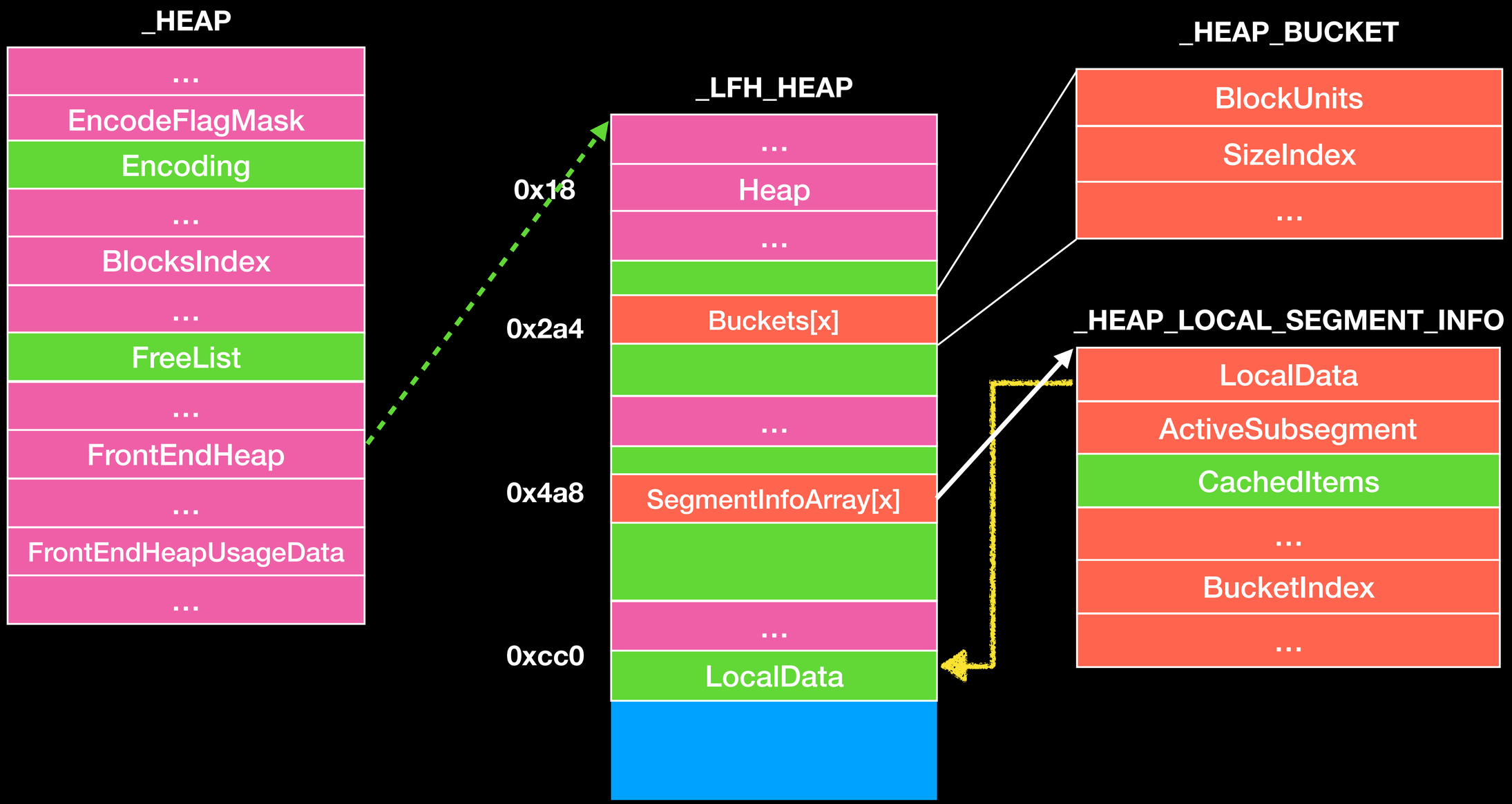
- SegmentInfoArray (_HEAP_LOCAL_SEGMENT_INFO)
  - ActiveSubsegment (_HEAP_SUBSEGMENT)
    - 極為重要結構
    - 對應到分配出去的 Subsegment
    - 其目的在於管理 Userblock
      - 記錄了剩餘多少 chunk
      - 該 Userblock 最大分配數等等

LFH

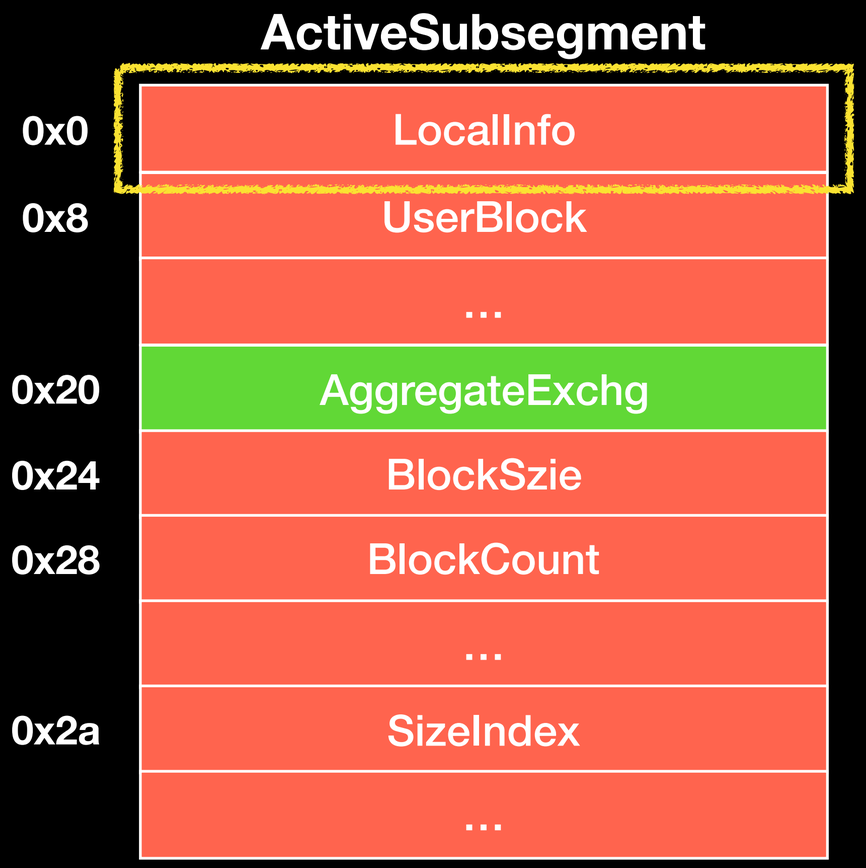


- SegmentInfoArray (_HEAP_LOCAL_SEGMENT_INFO)
  - CachedItems (_HEAP_SUBSEGMENT)
    - _HEAP_SUBSEGMENT array
      - 存放對映到該 SegmentInfo 且還有可以分配 chunk 給 user 的Subsegment
      - 當 ActiveSubsegment 用完時，會從這邊填充，置換掉 ActiveSubsegment

LFH

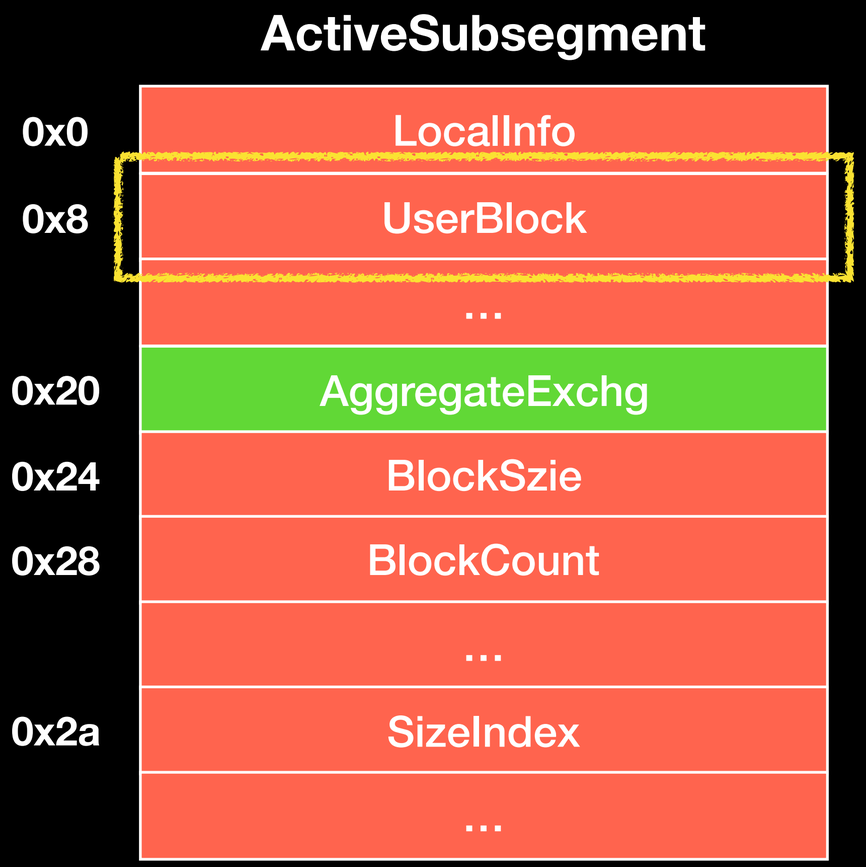

_HEAP
...
EncodeFlagMask
Encoding
...
BlocksIndex
...
FreeList
...
FrontEndHeap
...
FrontEndHeapUsageData
...
_LFH_HEAP
0x18
0x2a4
0x4a8
0xcc0
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...

LFH



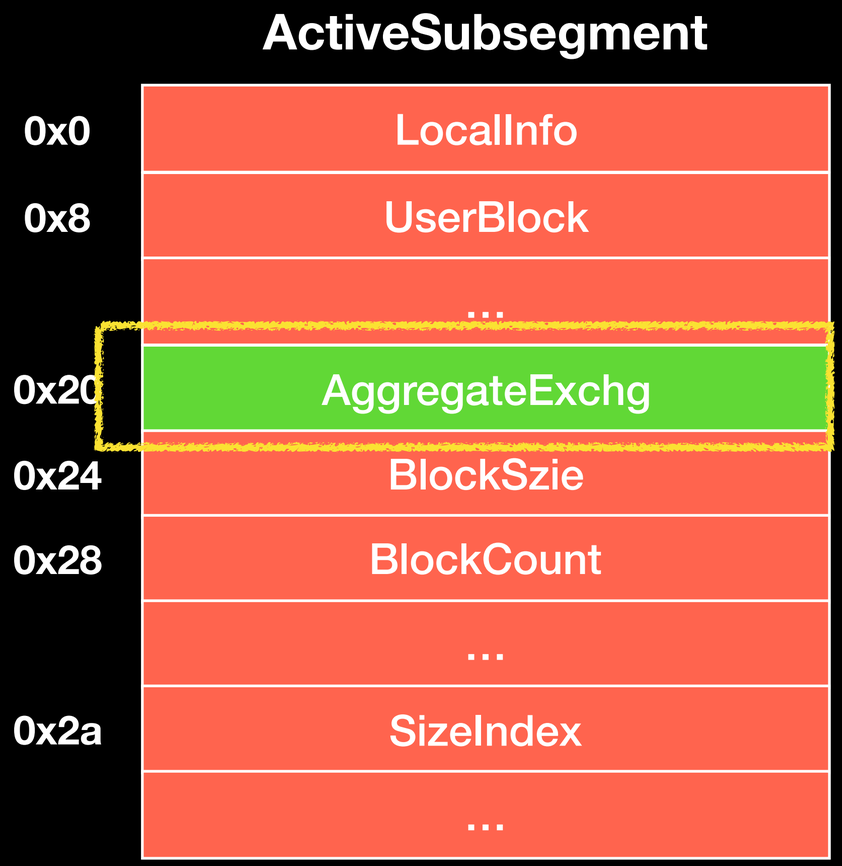
- ActiveSubsegment (_HEAP_SUBSEGMENT)
  - LocalInfo (_HEAP_LOCAL_SEGMENT_INFO)
    - 指回對應到的 _HEAP_LOCAL_SEGMENT_INFO

LFH



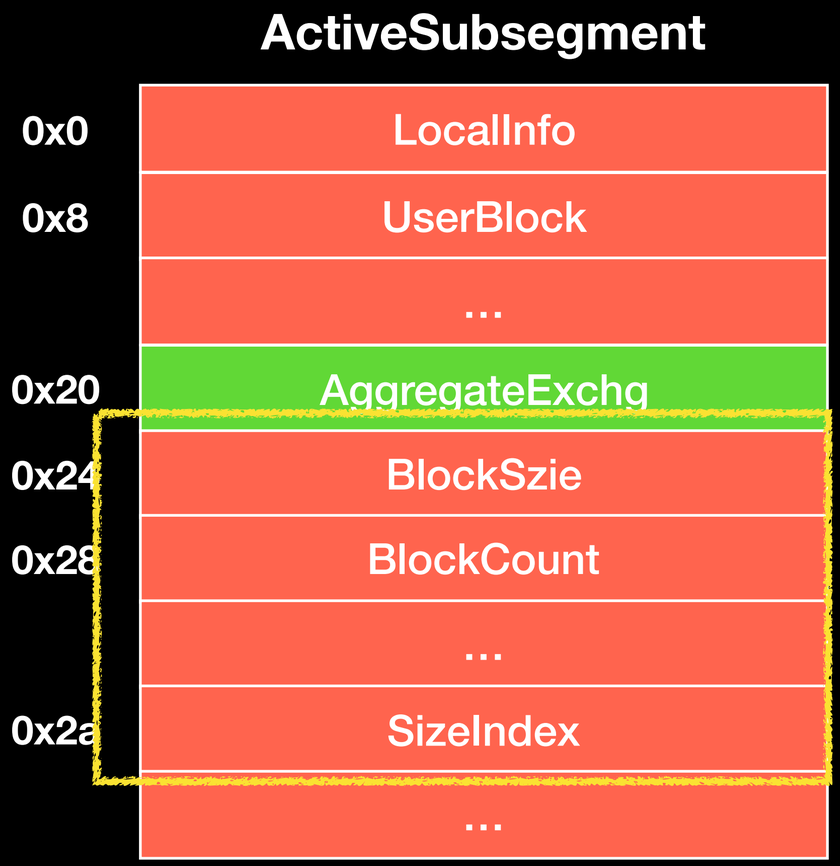
- ActiveSubsegment (_HEAP_SUBSEGMENT)
  - UserBlock (_HEAP_USERDATA_HEADER)
    - LFH 的記憶體分配池
      - 也就是要分配出去的 Chunk (Block) 所在位置
    - 開頭會有些 metadata 管理這些 chunk
    - 重要結構

LFH



- ActiveSubsegment (_HEAP_SUBSEGMENT)
  - AggregateExchg (_INTERLOCK_SEQ)
    - 用來管理對應到的 UserBlock 中還有多少 freed chunk
    - LFH 用這個判斷是否還從該 UserBlock 分配
    - 同時也有 Lock 的作用

LFH



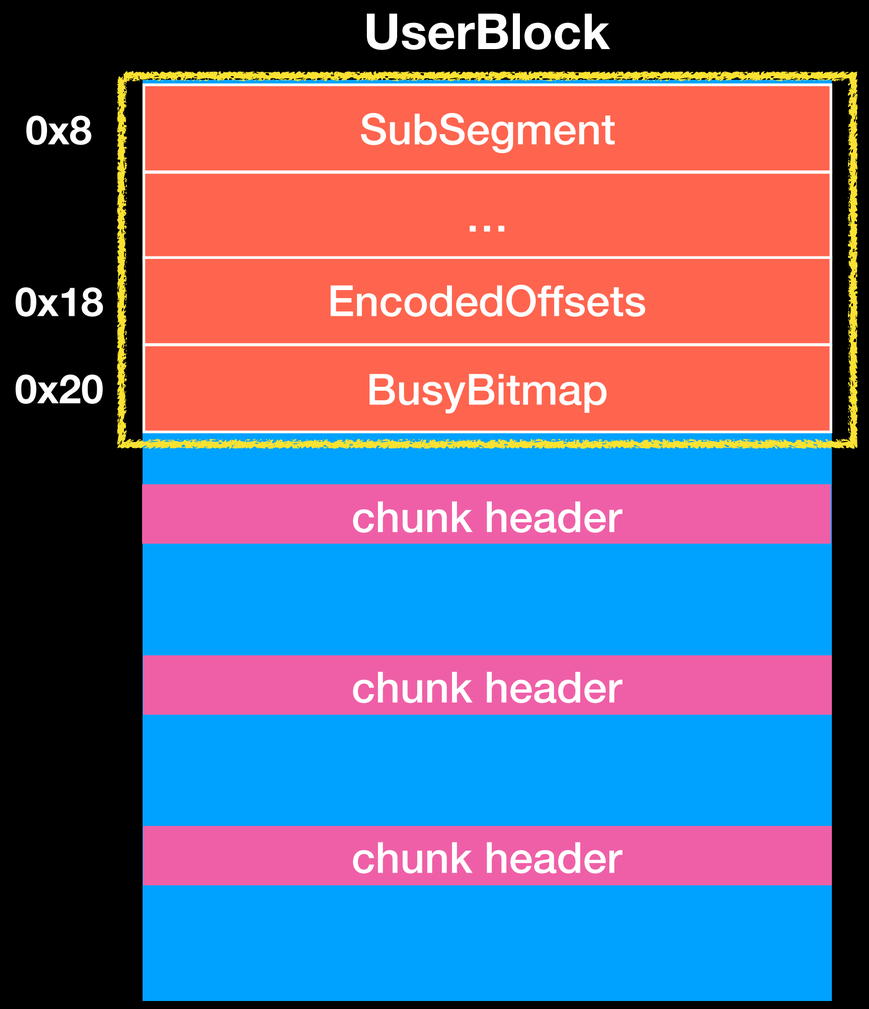
- ActiveSubsegment (_HEAP_SUBSEGMENT)
  - BlockSize
    - 在該 UserBlock 中每個 Block (chunk)的大小
  - BlockCount
    - 在該 UserBlock 中 Block 的總數
  - SizeIndex
    - 該 UserBlock 對應到的 SizeIndex

LFH

AggregateExchg

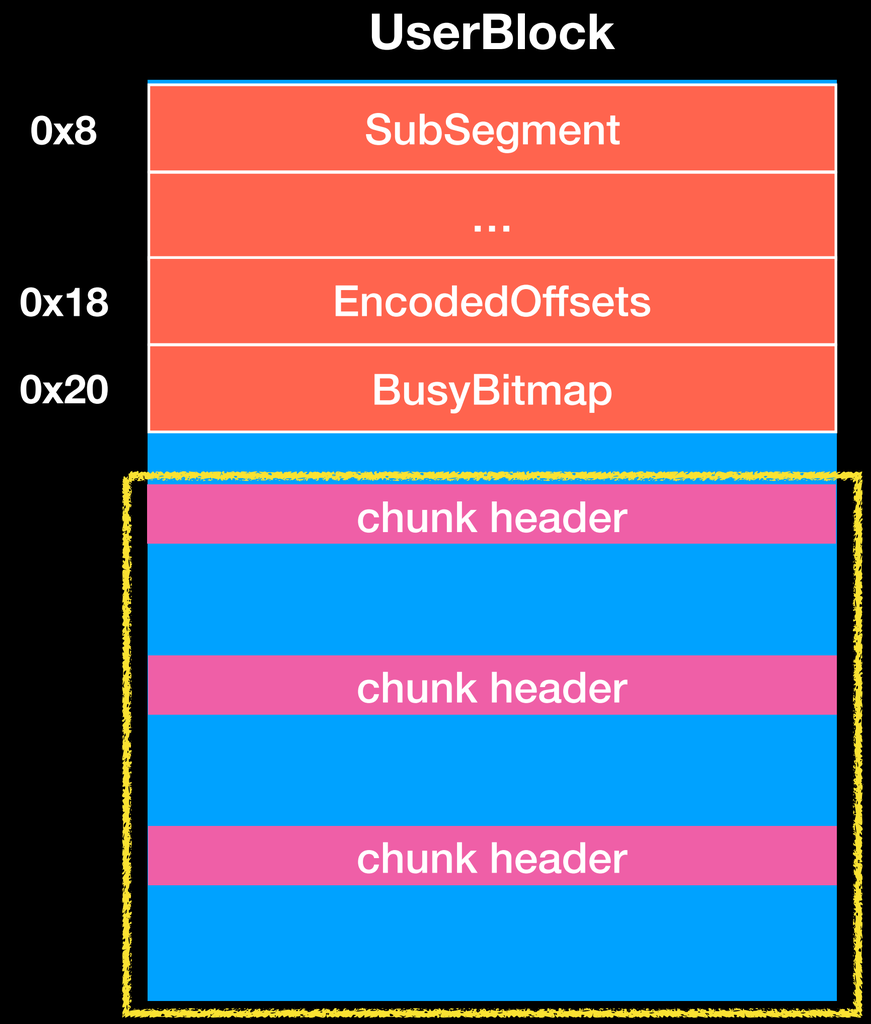
- AggregateExchg (_INTERLOCK_SEQ)
- Depth
  - 該 UserBlock 所剩下的 Freed chunk 數量
- Lock
  - 就是 Lock

LFH



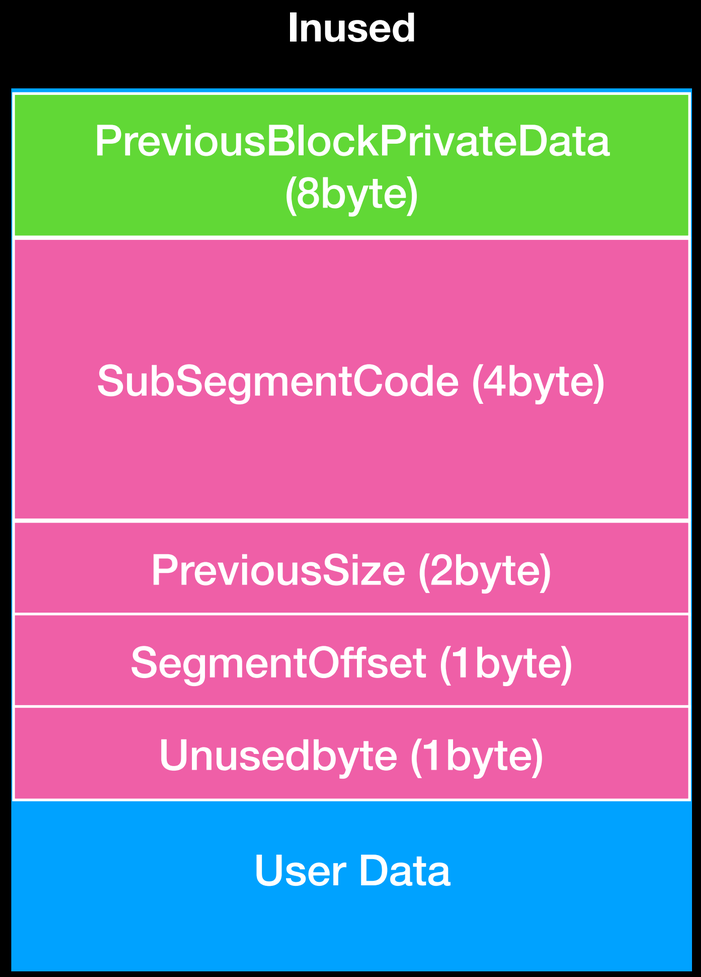
- UserBlock (_HEAP_USERDATA_HEADER)
  - SubSegment
    - 指回對應的 Subsegment
  - EncodeOffsets
    - 用來驗證 chunk header 是否有被改過
  - BusyBitmap
    - 記錄該 UserBlock 哪些 chunk 有在用的 bitmap

LFH



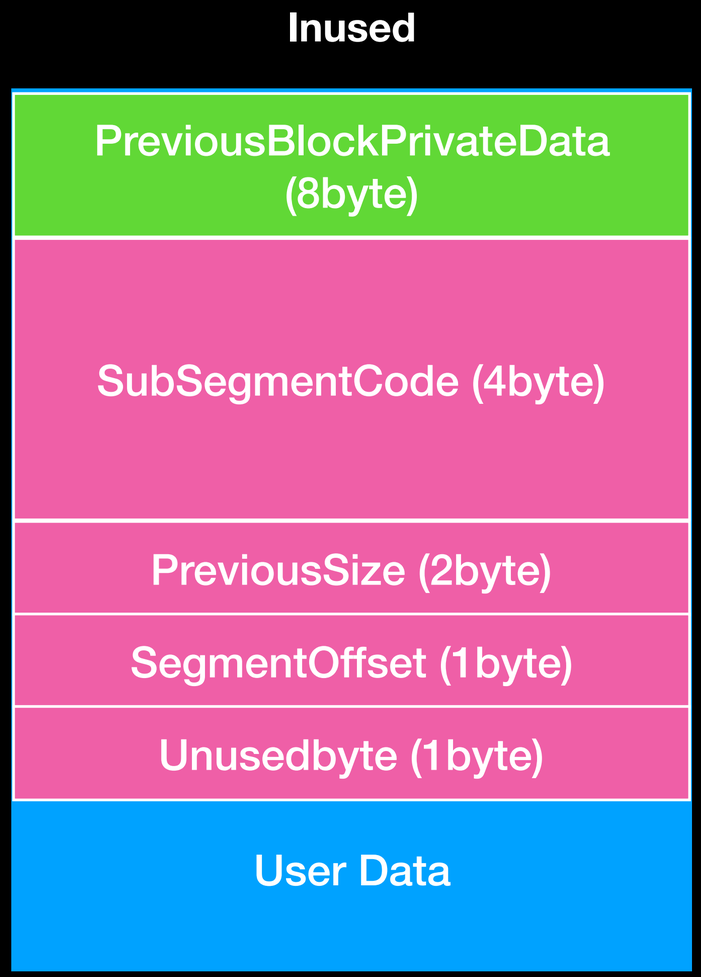
- UserBlock (_HEAP_USERDATA_HEADER)
  - Block (chunk)
    - LFH 回傳給使用者的 chunk

LFH



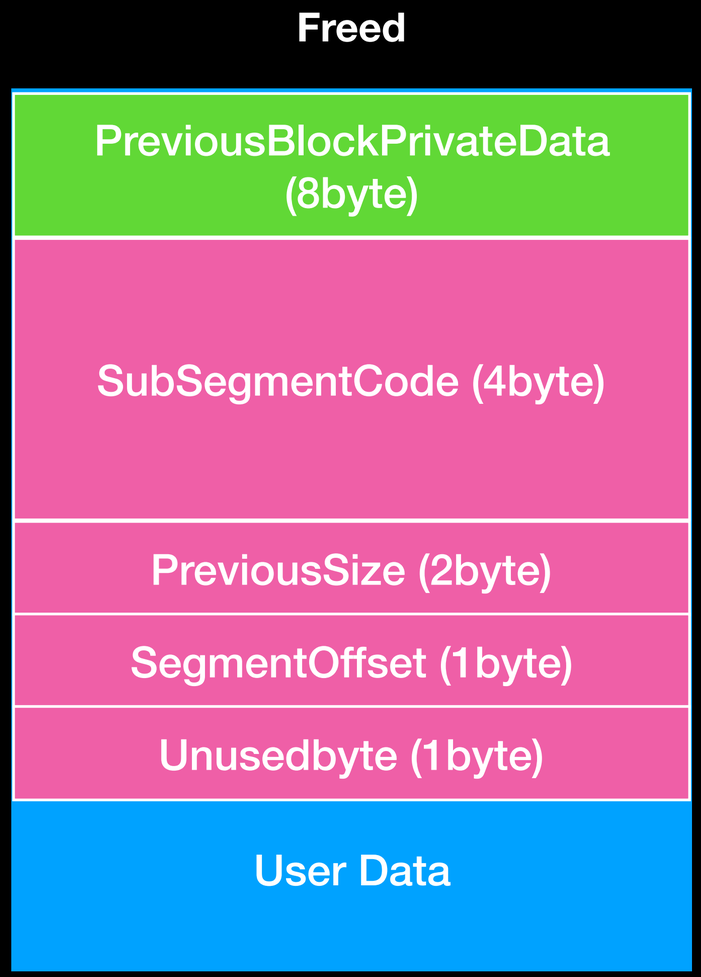
- _HEAP_ENTRY (chunk)
  - SubSegmentCode
    - Encode 過的 metadata 用來推回 userblock 的位置
  - PreviousSize
    - 該 chunk 在 UserBlock 中的 index

LFH



- _HEAP_ENTRY (chunk)
  - Unusedbyte
    - Unusedbyte & 0x80 is true

LFH



- _HEAP_ENTRY (chunk)
  - Unusedbytes
    - 恆為 0x80
    - 用來判斷是否為 LFH 的 Free chunk

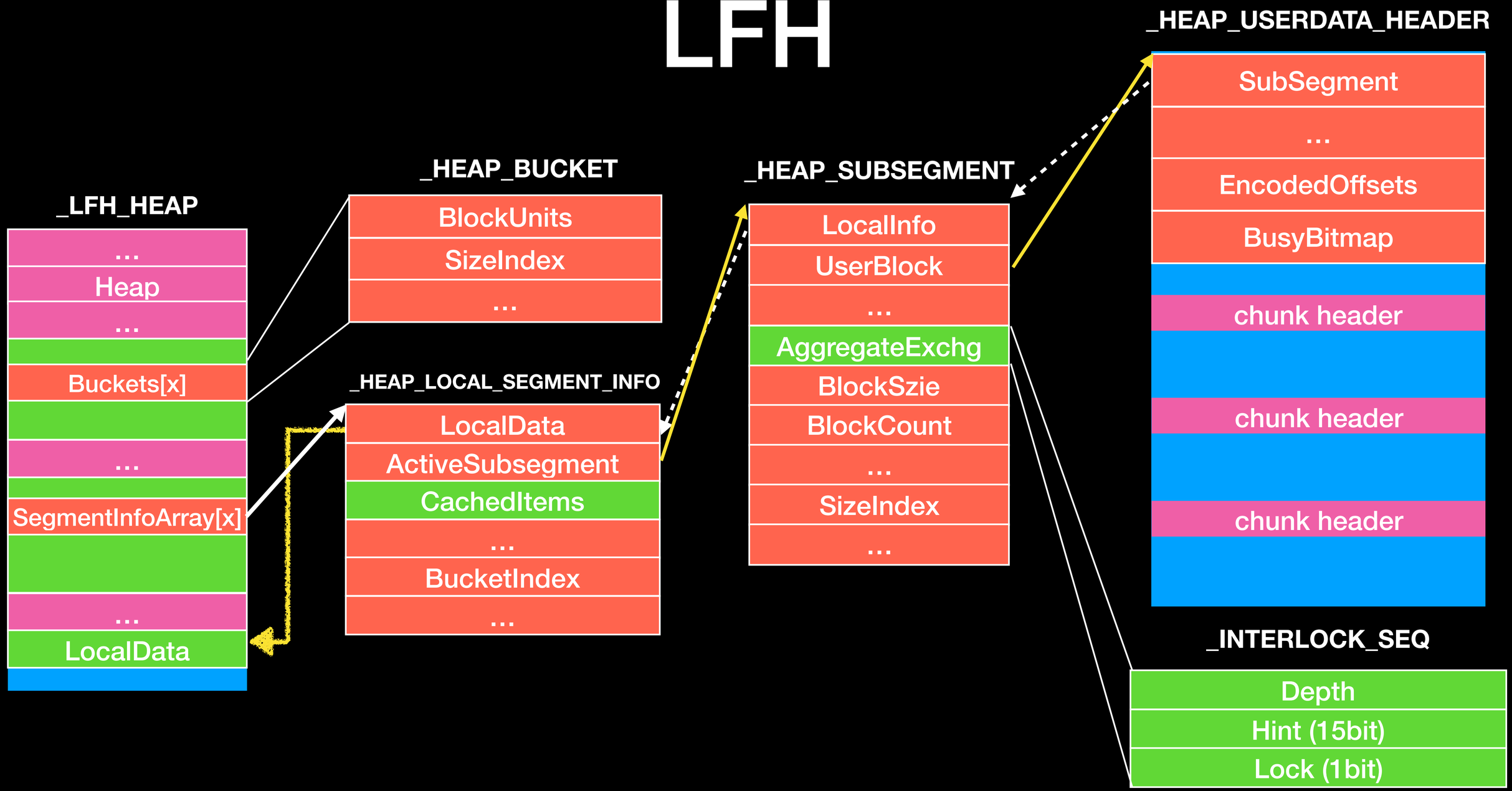
LFH

- Remark
  - 關於 EncodedOffsets
    - 在 UserBlock 初始化時設置，其算法為下面四個值的 xor
      - (sizeof(userblock header) | (BlockUnit*0x10 << 16))
      - LFHkey
      - Userblock address
      - _LFH_HEAP address

LFH

- Remark
  - 關於 LFH header encoding
    - 所有的 chunk header 在初始化時都會經過 xor 過，算法為下面個值的 xor
      - _HEAP address
      - LFHkey
      - Chunk address >> 4
      - ((chunk address) - (UserBlock address)) << 12

LFH


_LFH_HEAP
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_INTERLOCK_SEQ
Depth
Hint (15bit)
Lock (1bit)

Windows memory allocator

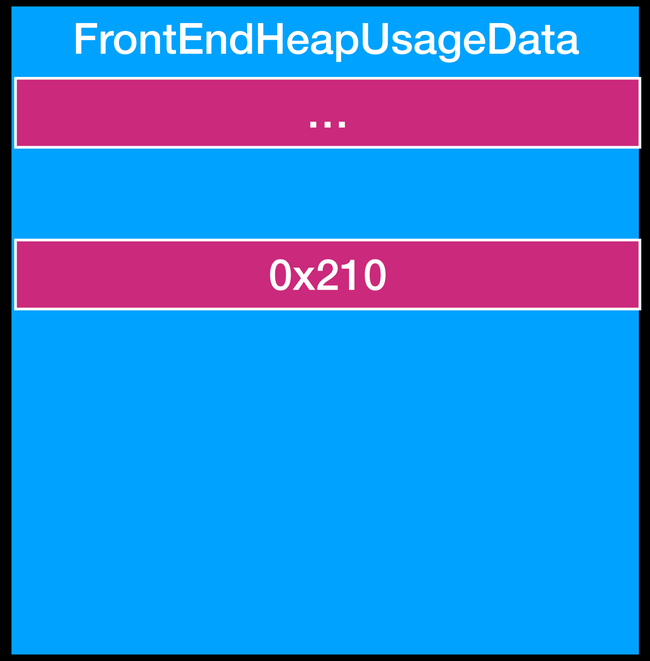
- Nt Heap
  - 前端管理器 (Front-End)
    - Data Structure
    - 分配機制

Windows memory allocator

- Nt Heap
  - 前端管理器 (Front-End)
    - 初始化
      - 在 FrontEndHeapUsageData[x] & 0x1f > 0x10 時，下一次的 allocate 會對 LFH 做出初始化
        - 會先 ExtendFrontEndUsageData 及增加更大的 BlocksIndex (0x80-0x400)
        - 建立 FrontEndHeap
        - 初始化 SegmentInfoArrays[idx]
      - 接下來在 allocate 相同大小的 chunk 就會開始使用 LFH

Windows memory allocator

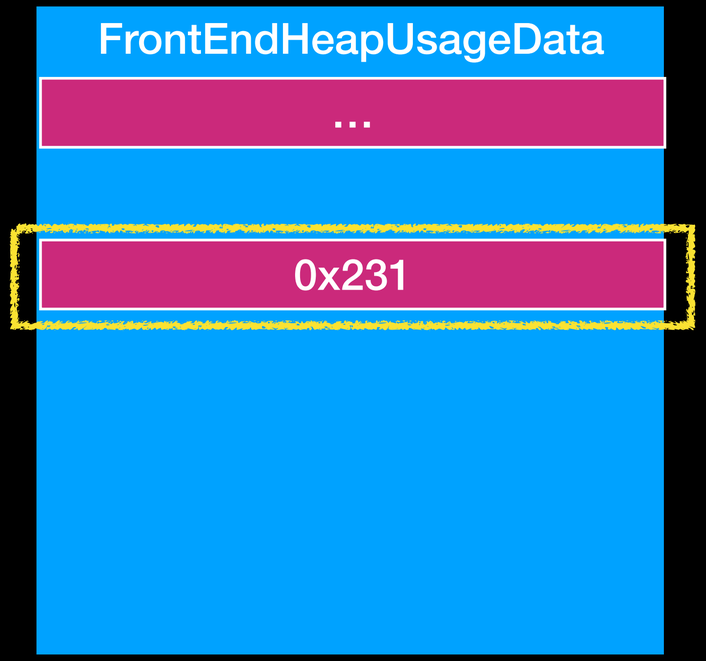
- LFH(初始化)
  - malloc(0x40) * 16



此時可 enable LFH 範圍為
index 0x0 - 0x80

Windows memory allocator

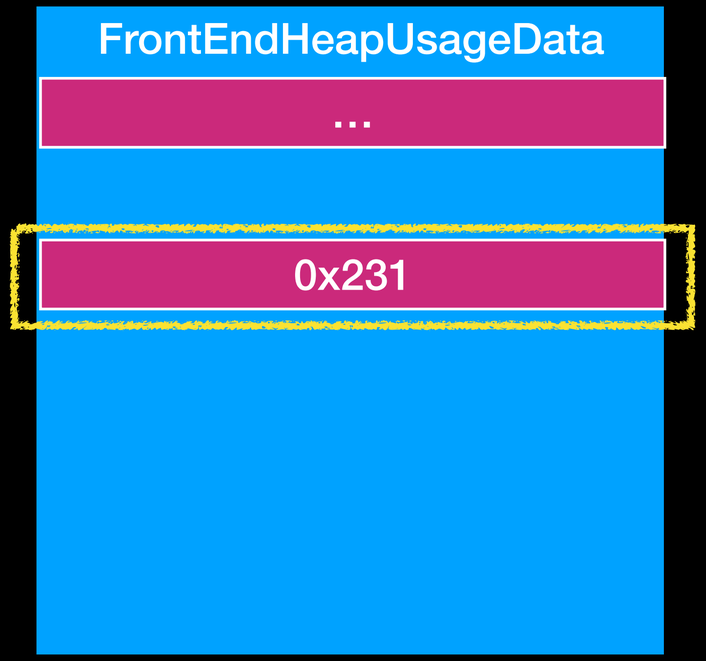
- LFH(初始化)
  - malloc(0x40) (第 17 次)



0x231 & 0x1f > 0x10

Windows memory allocator

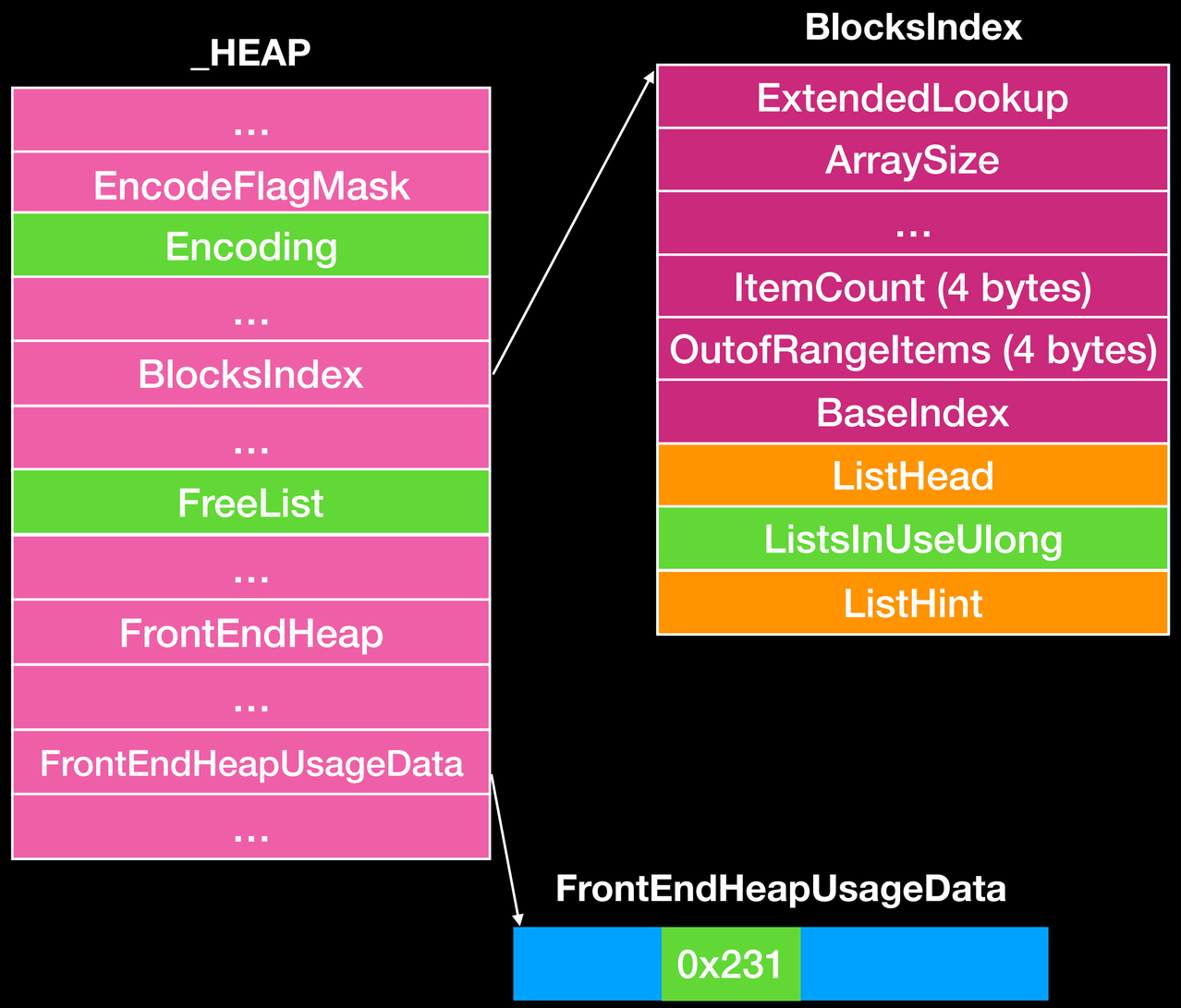
- LFH(初始化)



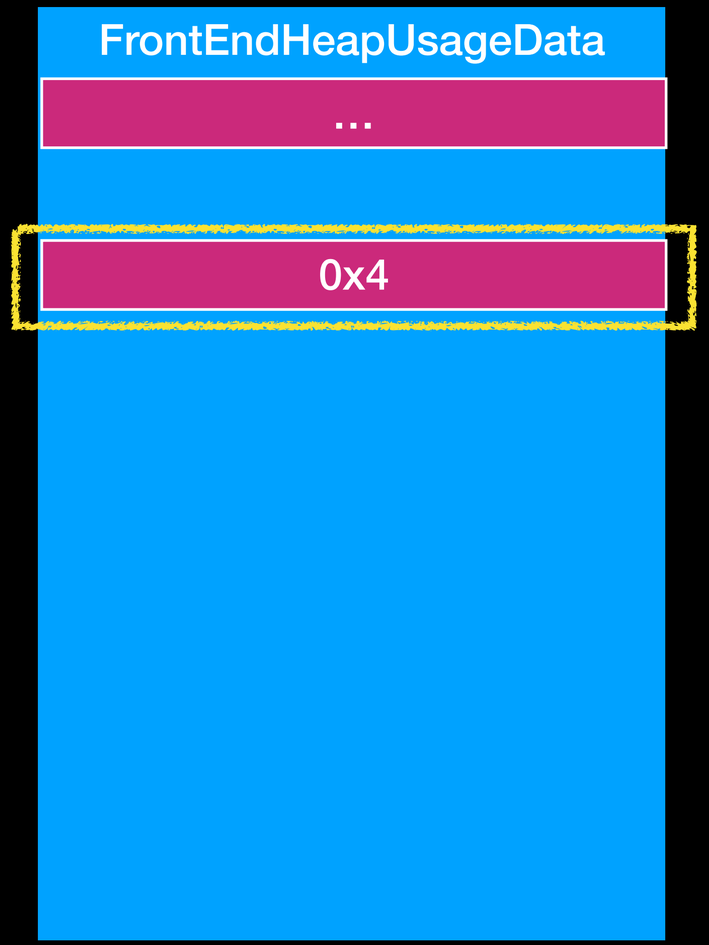
heap->CompatibilityFlag |= 0x20000000

設置上這個 flag 後
下次 allocate 時會去初始化LFH

Windows memory allocator

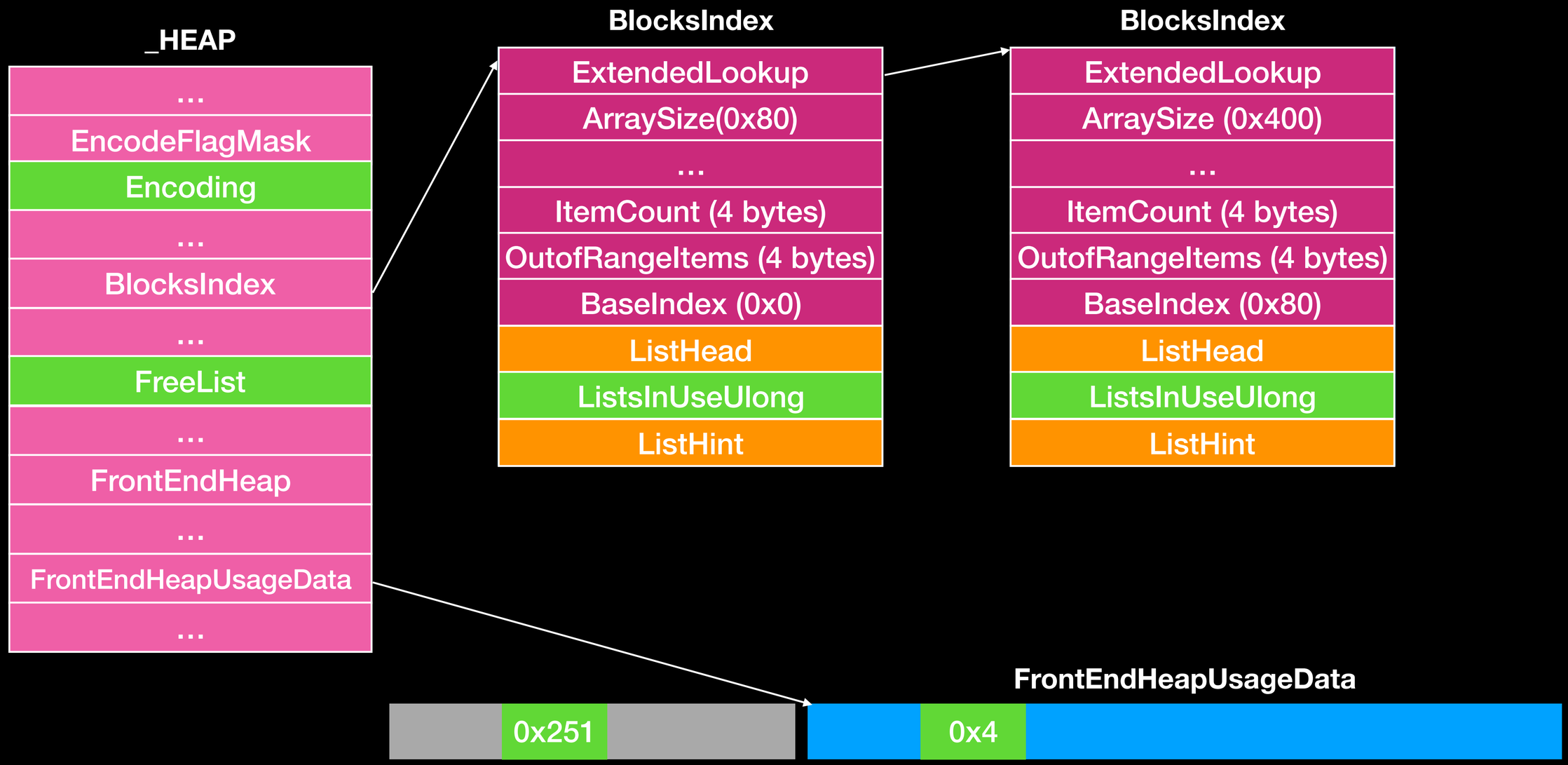


Windows memory allocator

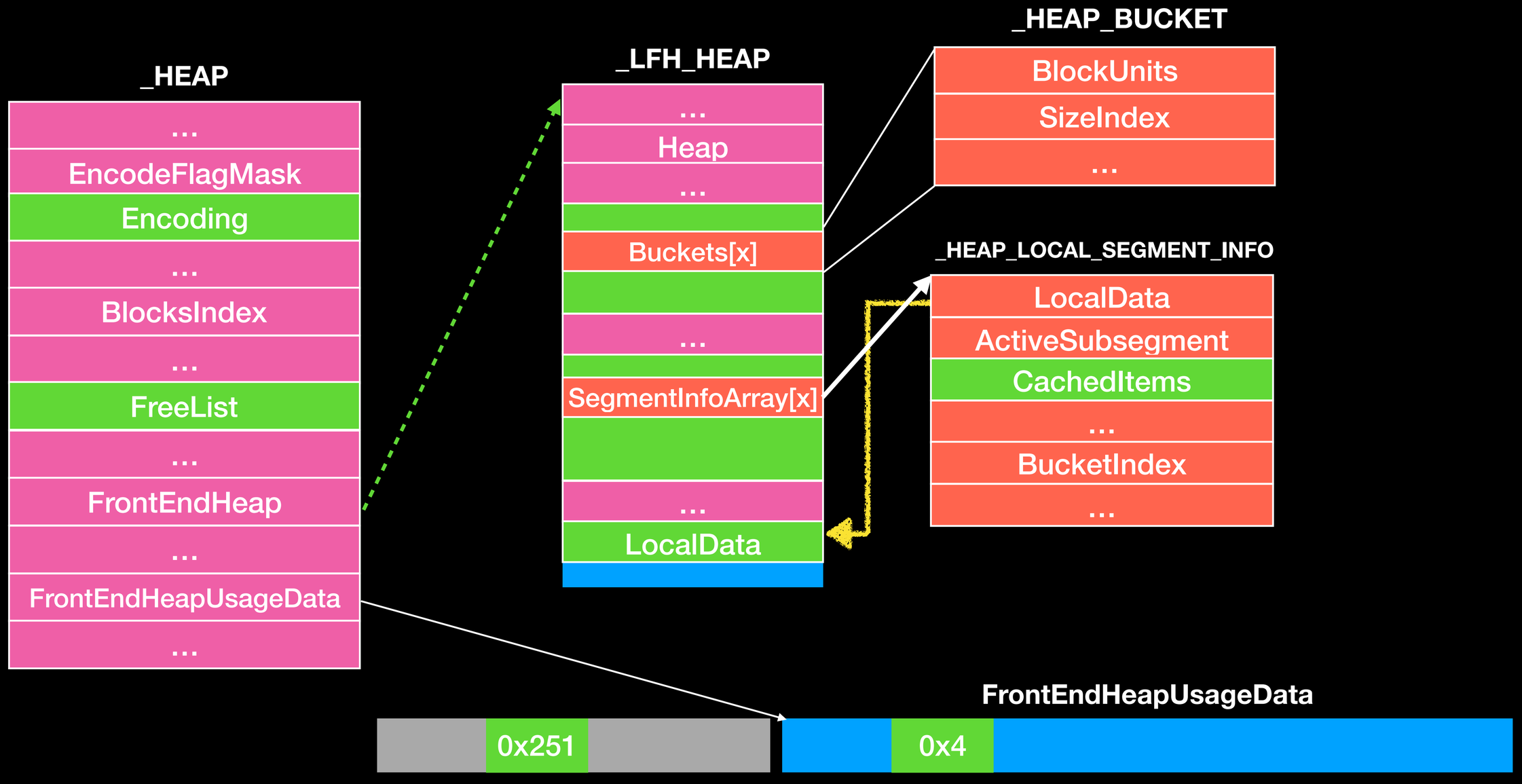


- LFH(初始化)
  - malloc(0x40) (第 18 次)
    - ExtendFrontEndUsageData 及增加更大的 BlocksIndex (0x80-0x400) ，並設置對應的 bitmap
      - 並在該對應的 FrontEndHeapUsageData 寫上對應的 index，此時可 enable LFH 範圍變為 (idx: 0-0x400)
    - 建立並初始化 FrontEndHeap (mmap)
    - 初始化 SegmentInfoArrays[idx]
      - 在 SegmentInfoArrays[BucketIndex] 填上 segmentInfo

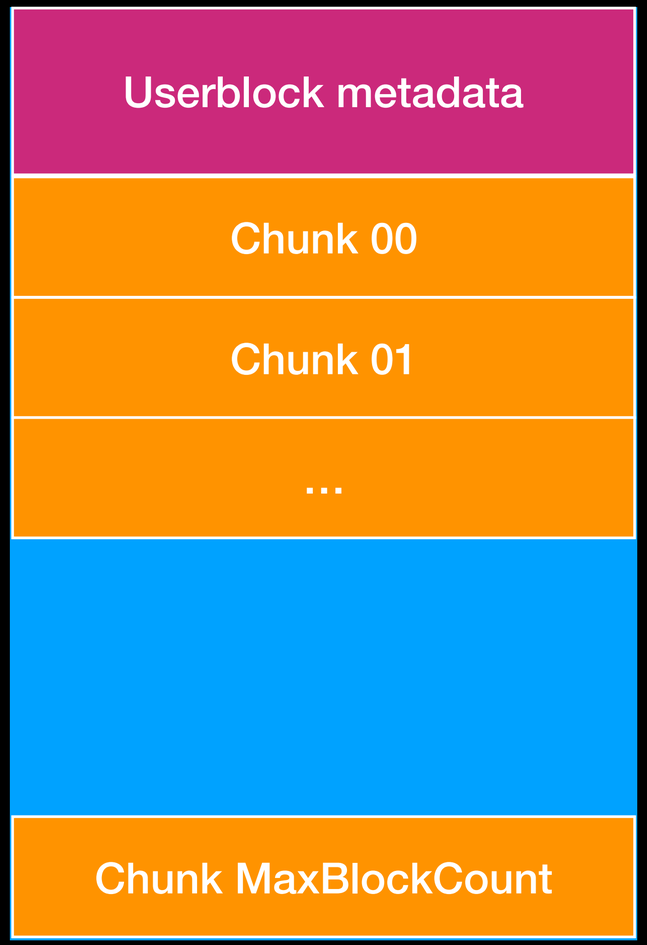
Windows memory allocator



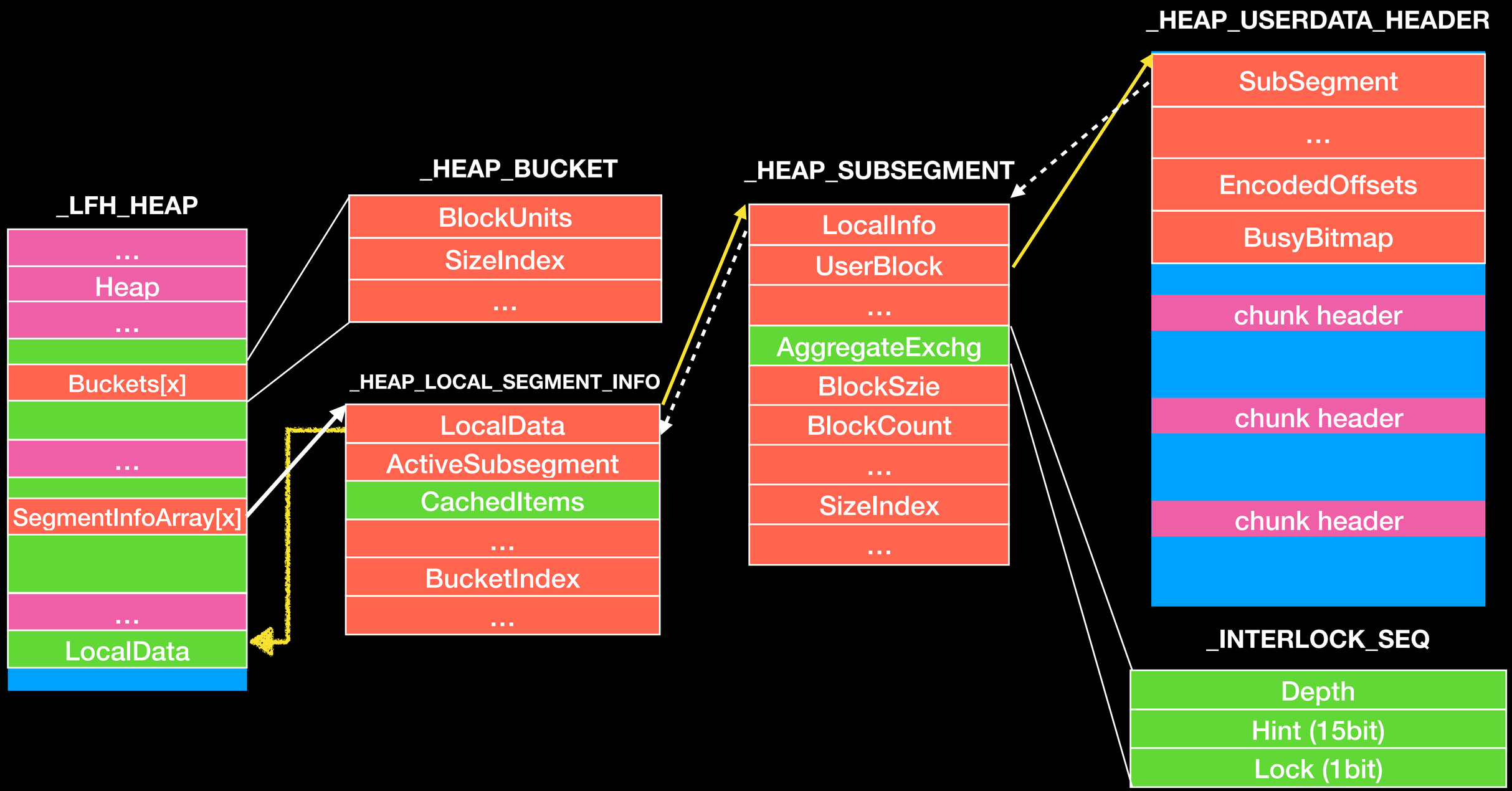
Windows memory allocator



Windows memory allocator

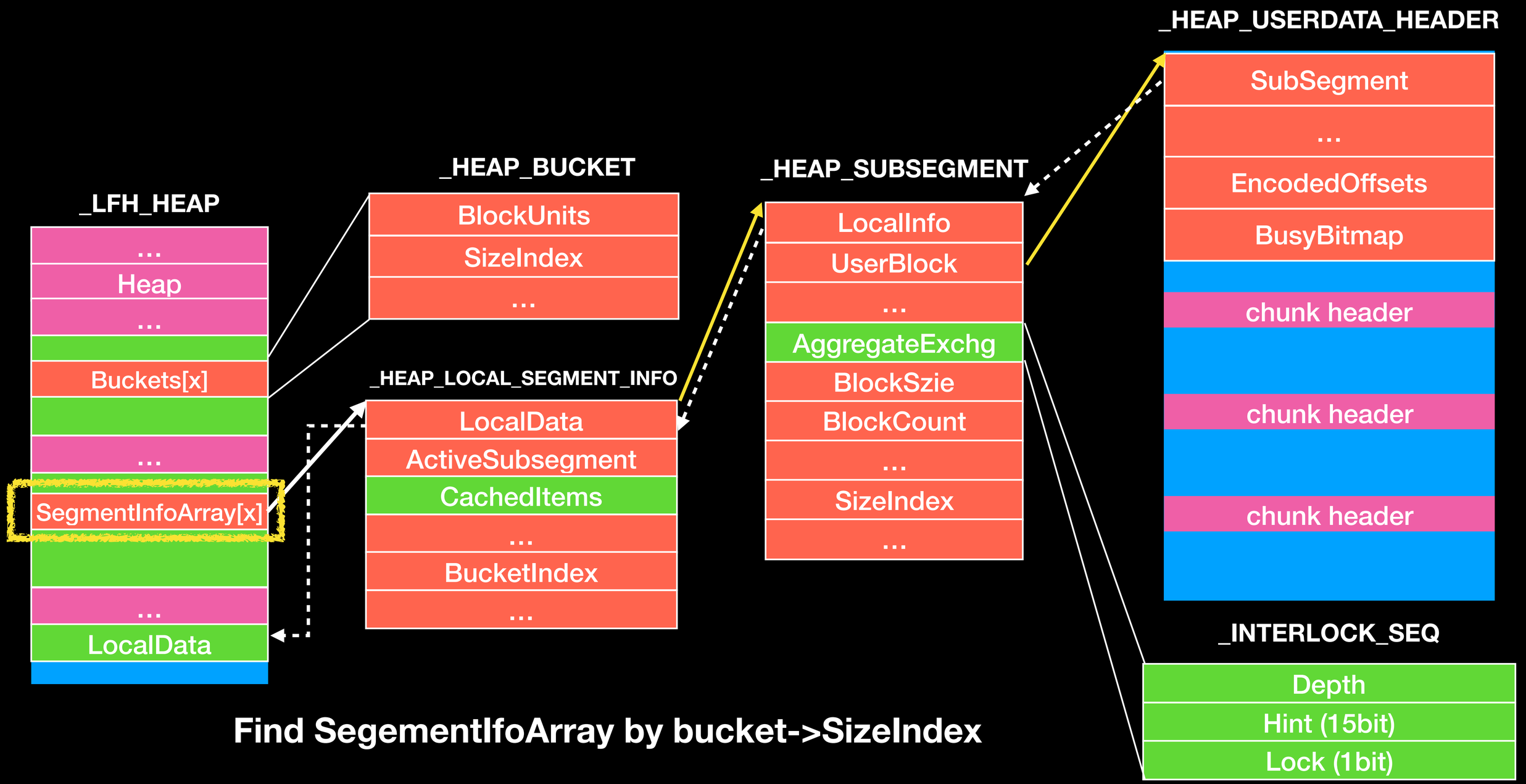


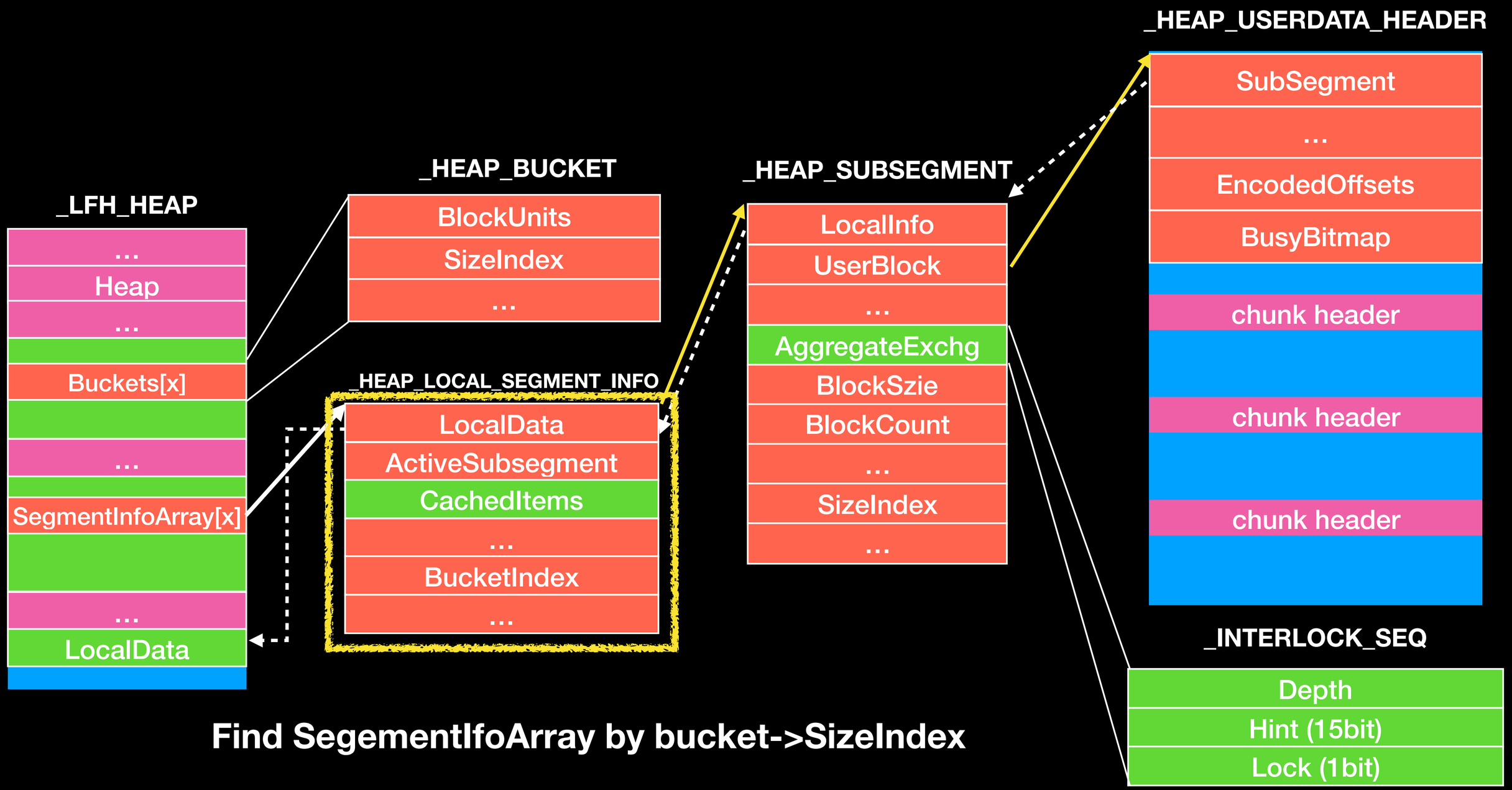
- LFH(初始化)
  - malloc(0x40) (第 19 次)
    - Allocate Userblock 並初始化
      - 設置對應的 chunk
    - 設置對應的 ActiviteSubsegment
    - 隨機返回其中的一個 chunk

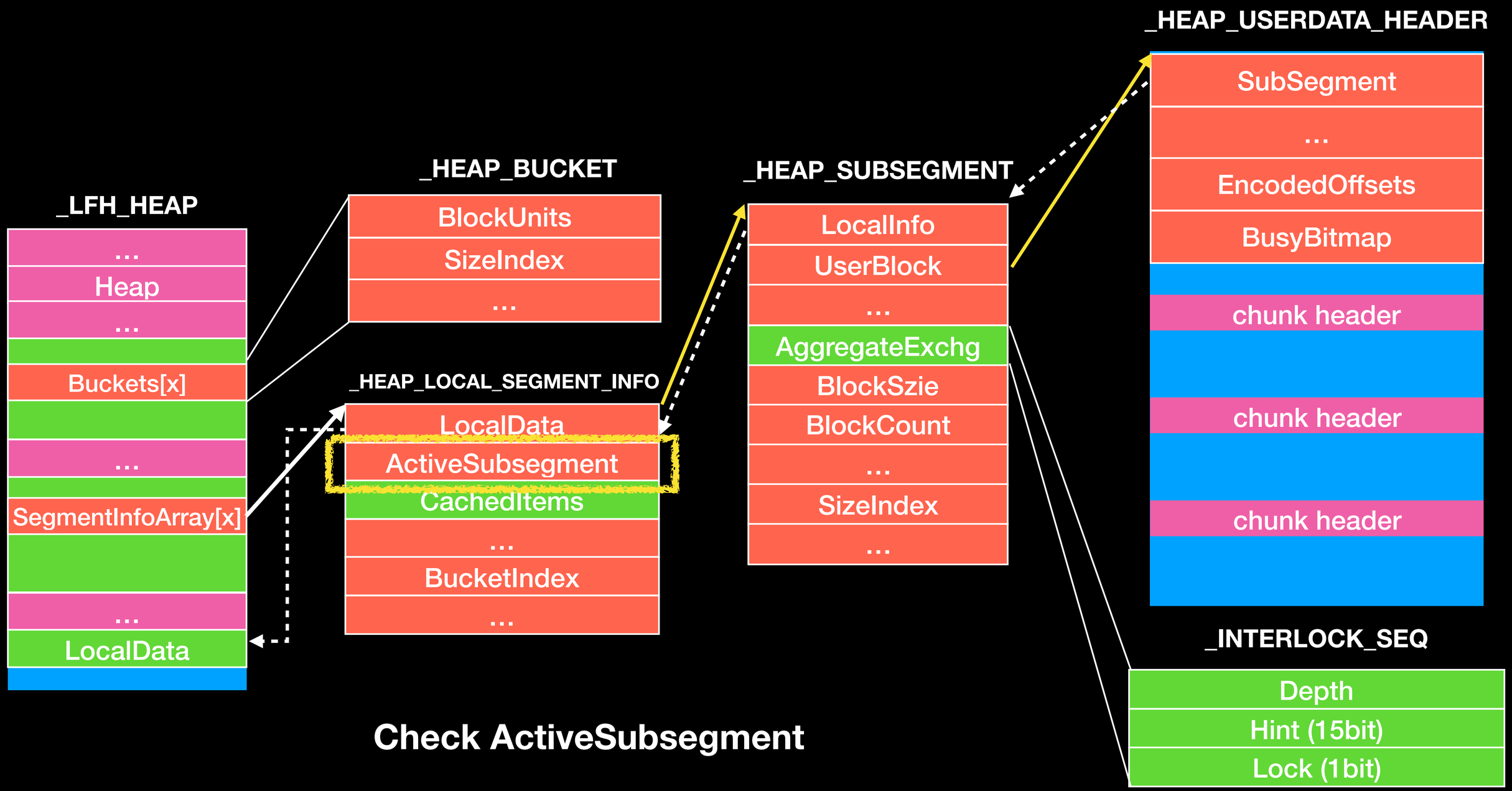

_LFH_HEAP
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_INTERLOCK_SEQ
Depth
Hint (15bit)
Lock (1bit)

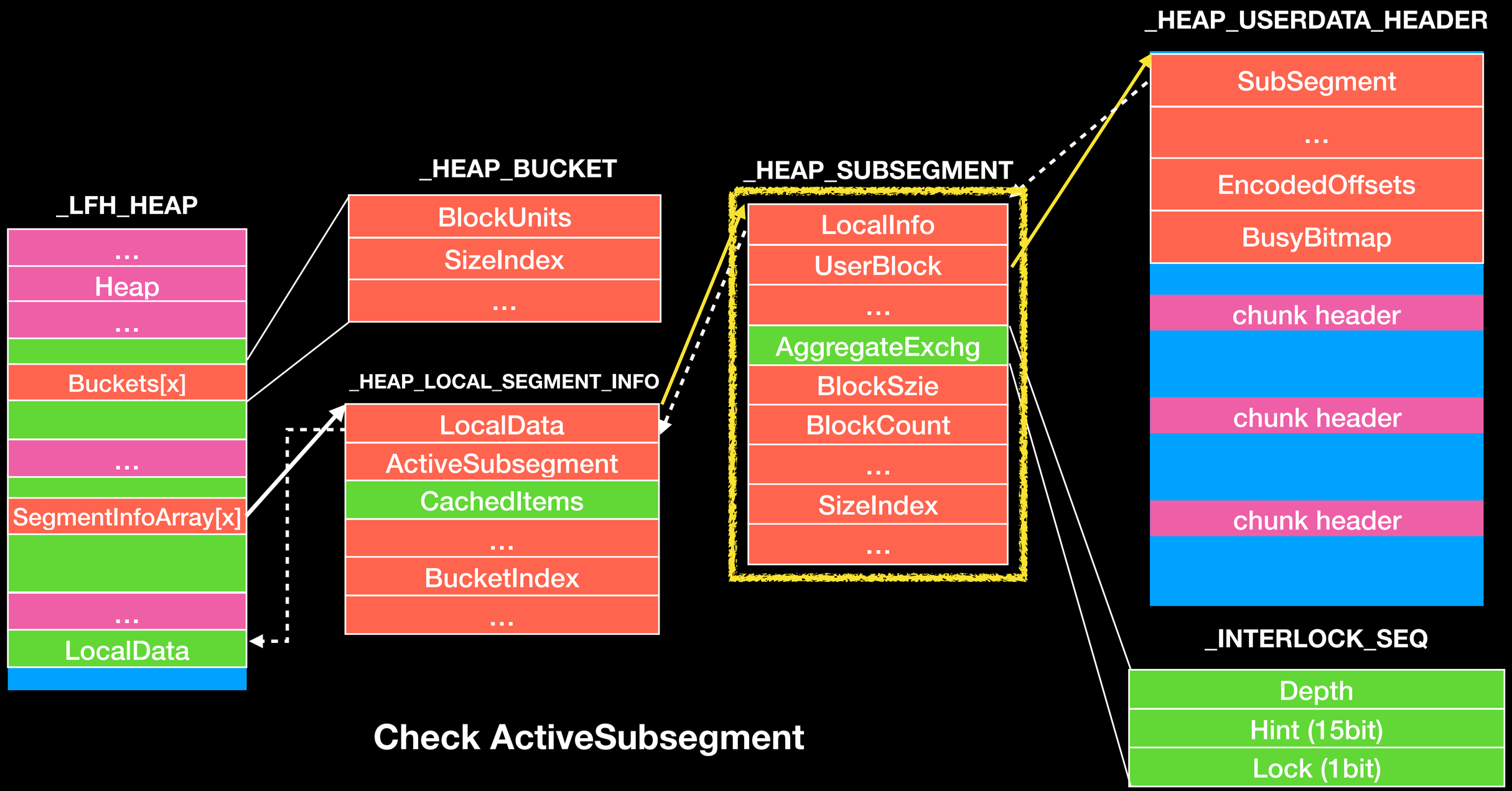
Windows memory allocator

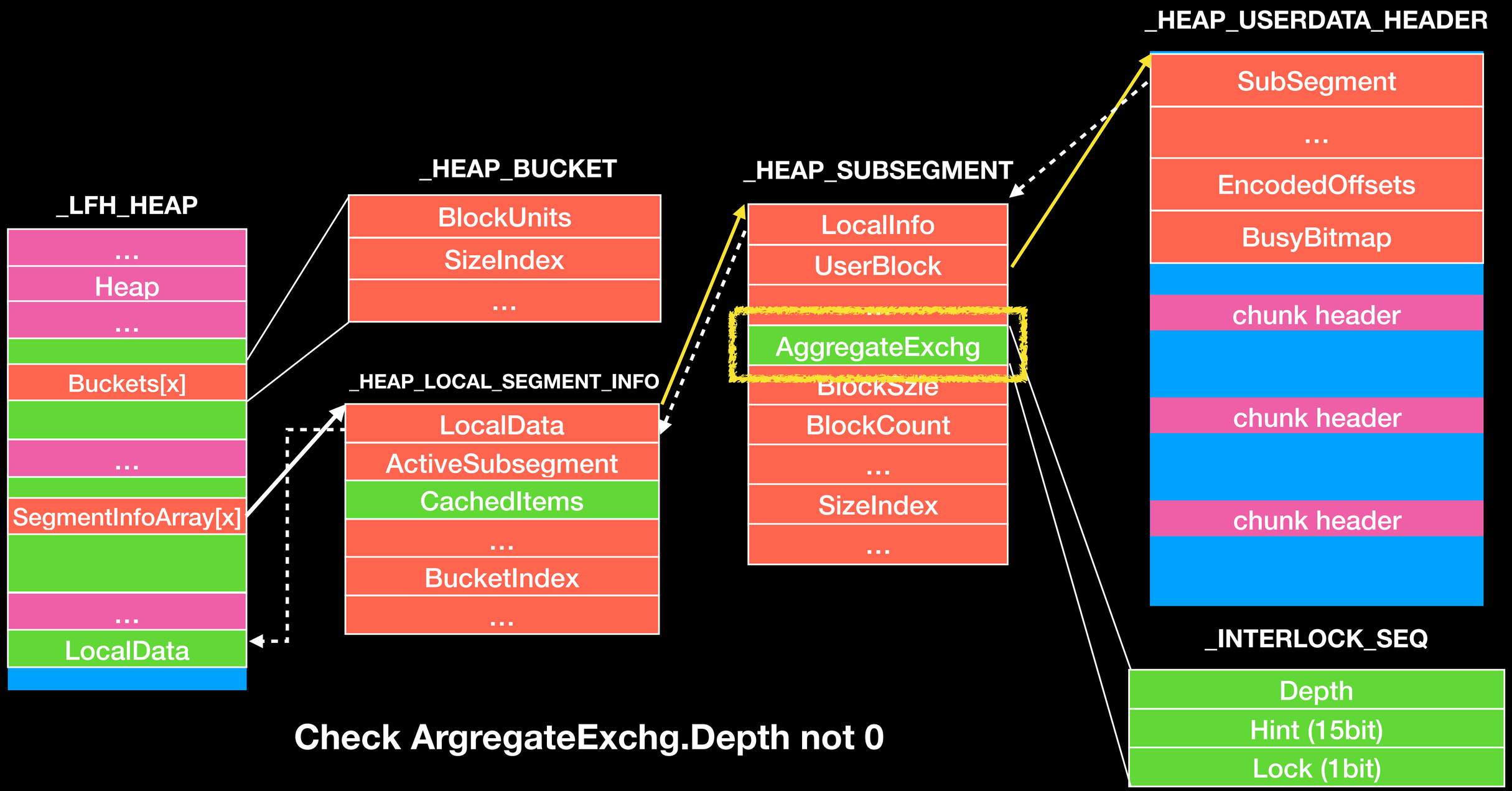
- LFH
  - Allocate (RtlpLowFragHeapAllocFromContext)
    - 先看看 ActiveSubsegment 中是否有可分配的 chunk
      - 從 ActiveSubsegment->depth 判斷
    - 如果沒有則會從 CachedItem 找，有找到的話會把 ActiveSubsegment 換成 CachedItem 中的 SubSegment

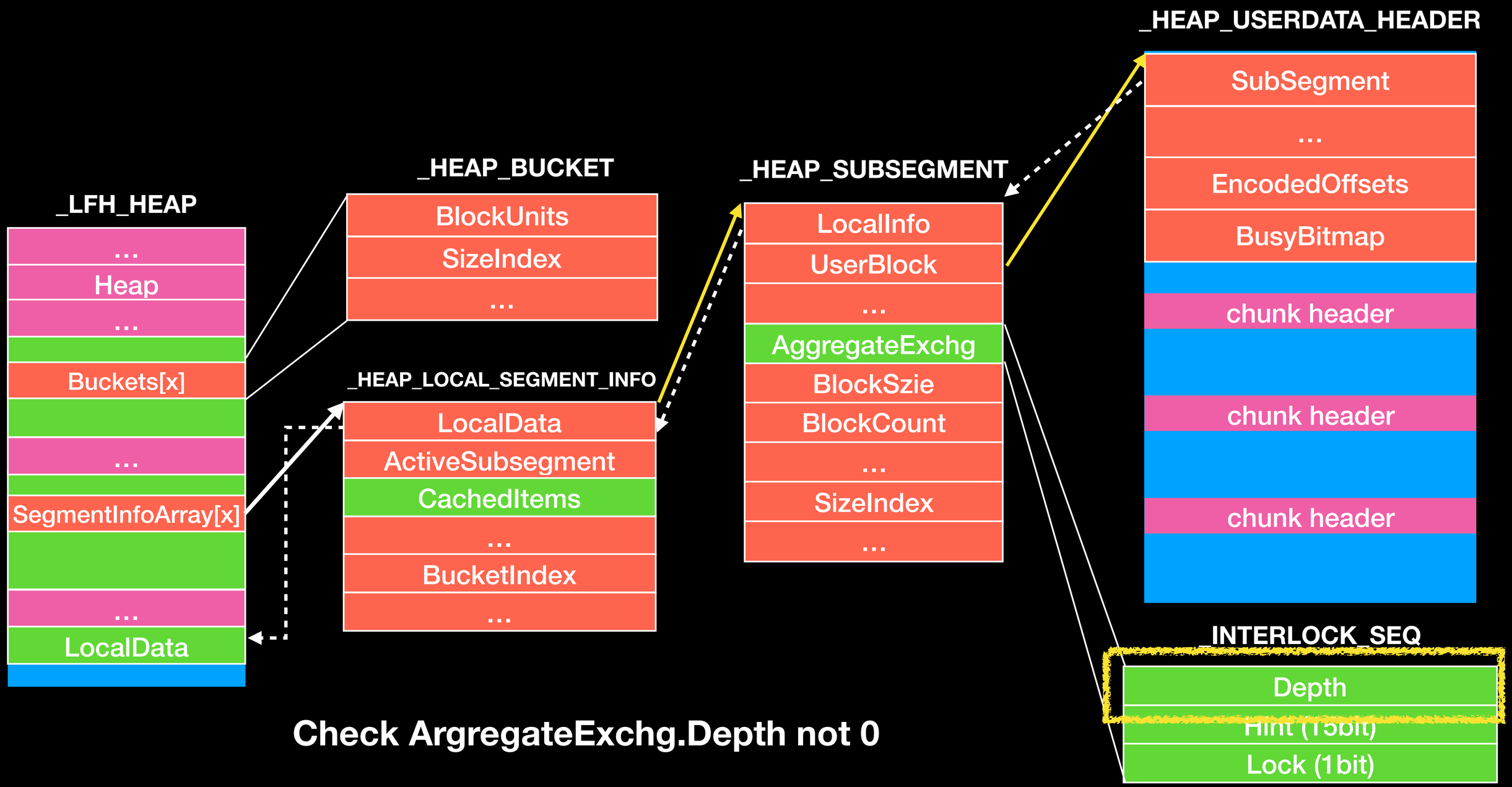

_LFH_HEAP
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_INTERLOCK_SEQ
Depth
Hint (15bit)
Lock (1bit)
Find SegementIfoArray by bucket->SizeIndex

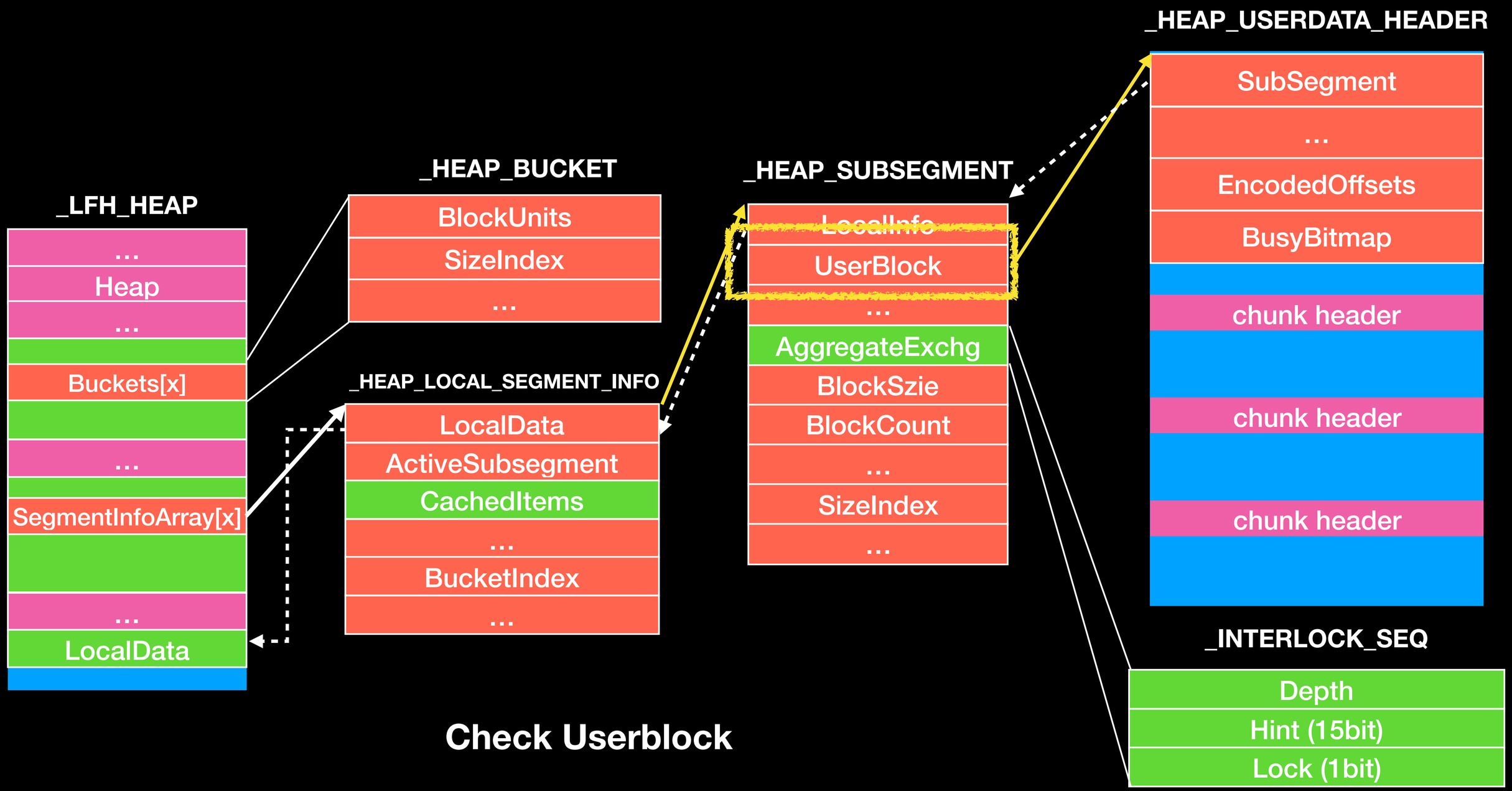

_LFH_HEAP
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_INTERLOCK_SEQ
Depth
Hint (15bit)
Lock (1bit)
Find SegementIfoArray by bucket->SizeIndex


_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_LFH_HEAP
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
_INTERLOCK_SEQ
Depth
Hint (15bit)
Lock (1bit)
Check ActiveSubsegment


_LFH_HEAP
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_INTERLOCK_SEQ
Depth
Hint (15bit)
Lock (1bit)
Check ActiveSubsegment


_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_LFH_HEAP
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
_INTERLOCK_SEQ
Depth
Hint (15bit)
Lock (1bit)
Check ArgregateExchg.Depth not 0


_LFH_HEAP
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_INTERLOCK_SEQ
Depth
Hint (15bit)
Lock (1bit)
Check ArgregateExchg.Depth not 0

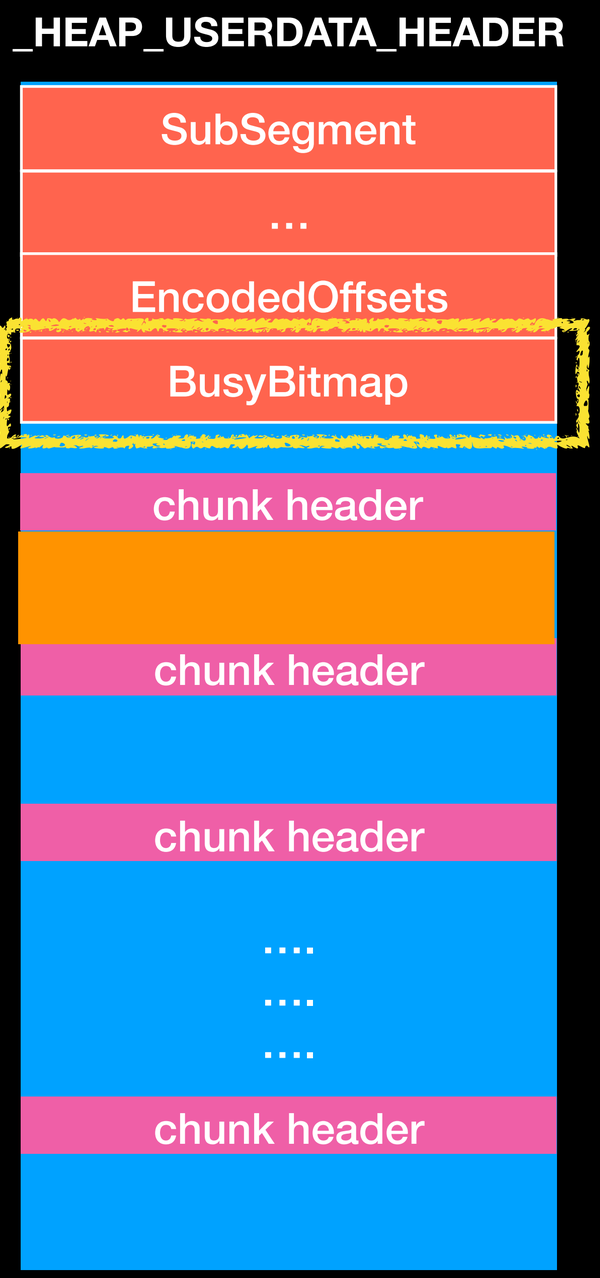

_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_HEAP_BUCKET
BlockUnits
SizeIndex
...
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_LFH_HEAP
...
Heap
...
Buckets[x]
...
SegmentInfoArray[x]
...
LocalData
_HEAP_LOCAL_SEGMENT_INFO
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
_INTERLOCK_SEQ
Depth
Hint (15bit)
Lock (1bit)
Check Userblock

Windows memory allocator

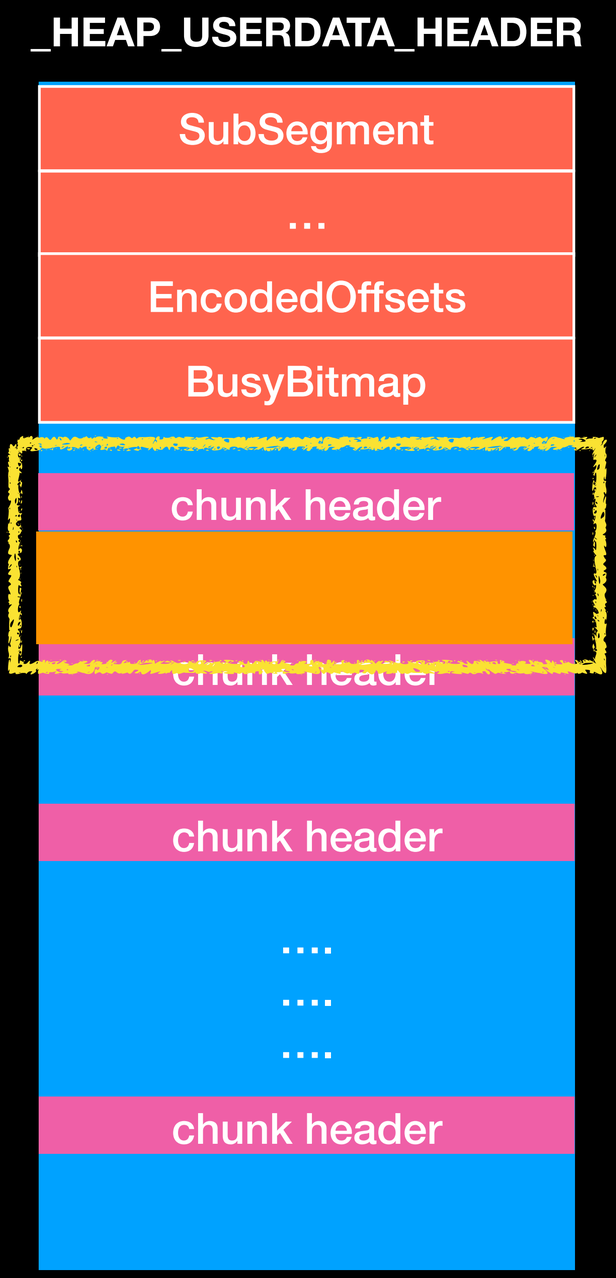
- LFH
  - Allocate (RtlpLowFragHeapAllocFromContext)
    - 取得 RtlpLowFragHeapRandomData[x] 上的直
      - 下一次會從 RtlpLowFragHeapRandomData[x+1] 取
      - x 為 1 byte ，當下次輪回 x 則會 x = rand() %256 繼續取
      - RtlpLowFragHeapRandomData 為一個 random 256 byte 的陣列
        - 範圍為 0x0 - 0x7f

Windows memory allocator

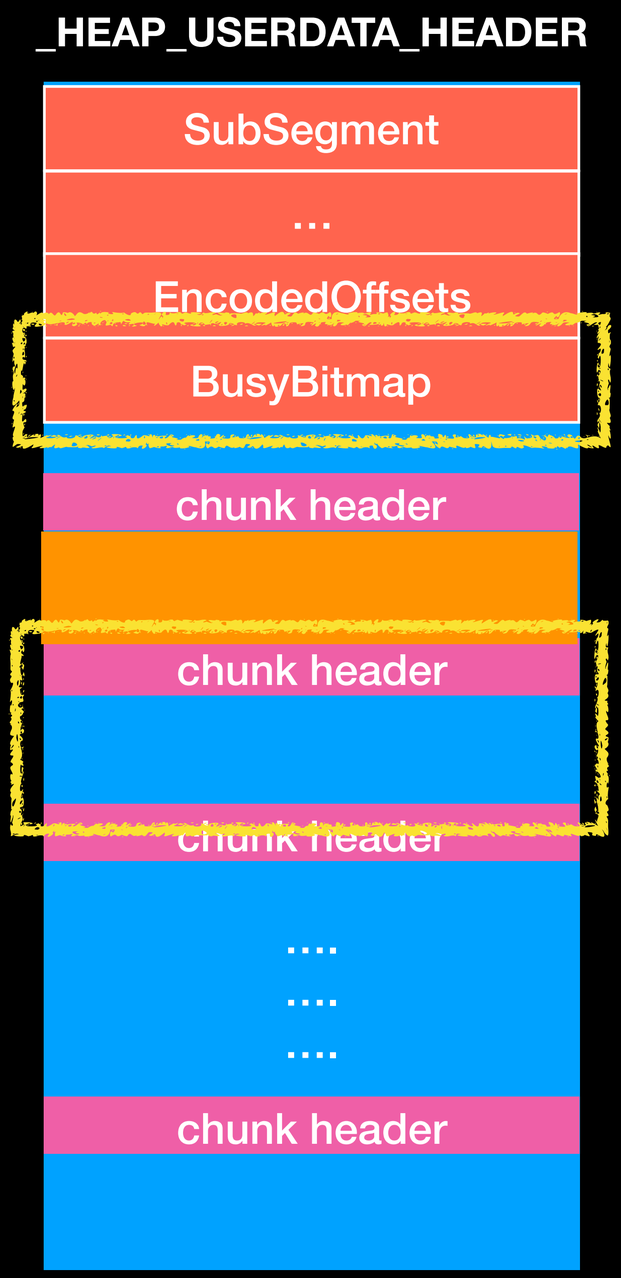
- LFH
  - Allocate (RtlpLowFragHeapAllocFromContext)
    - 最獲得的 index 為
      - RtlpLowFragHeapRandomData[x]*maxidx >> 7
      - 如果 collision 則往後取最近的
    - 檢查 (unused byte & 0x3f) !=0 (表示 chunk 是 freed)
    - 最後設置 index 及 unused byte 就返回給使用者



- Get an index
  - RtlpLowFragHeapRandomData[x]*maxidx >> 7
- 檢查該 index 對映到的 bitmap 是否為 0
  - 是 0 則返回對映的 bitmap
  - 非 0 擇取最近的下一個



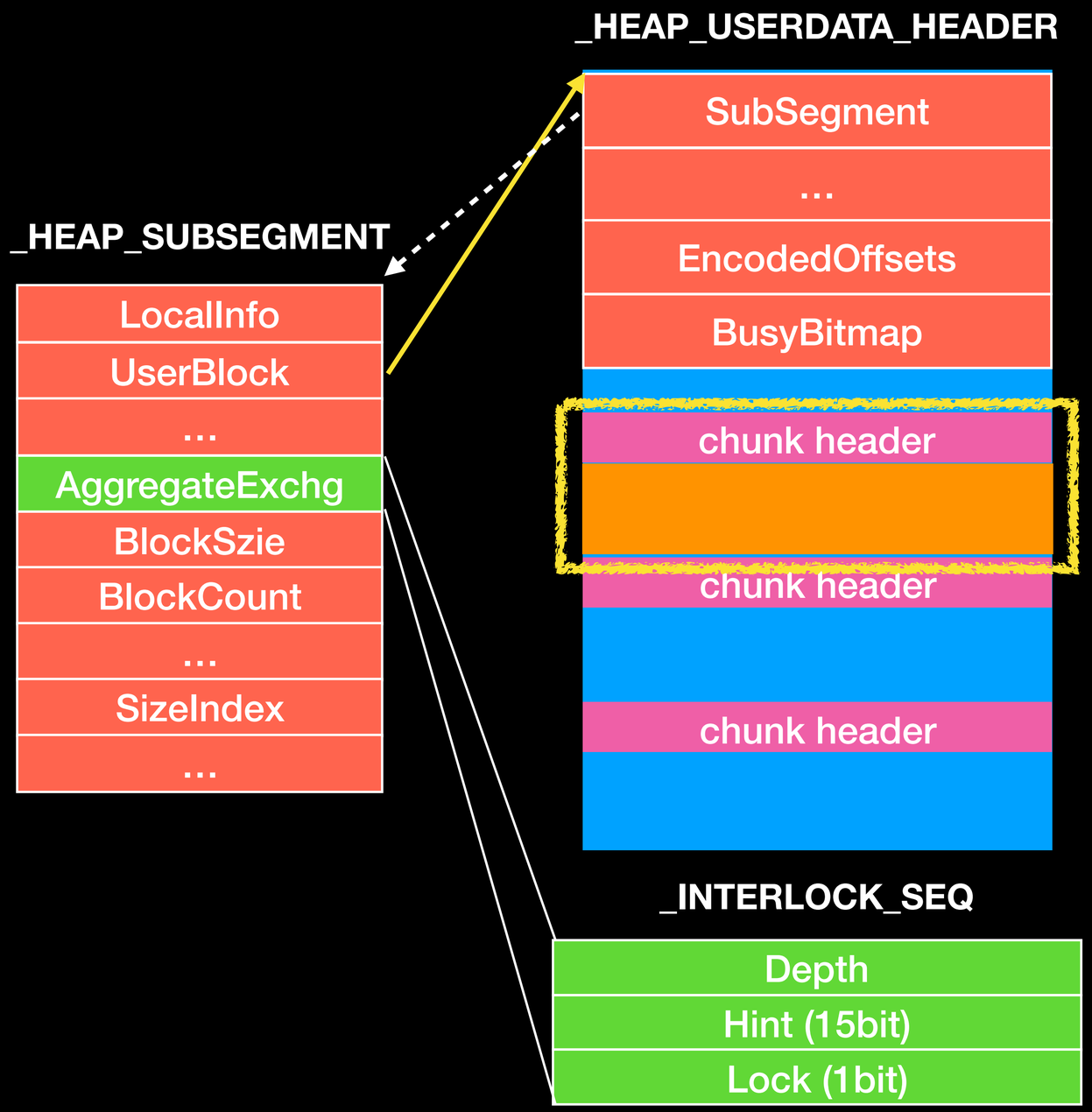
- 算出來的 index 發現已用過
  - 會取下一個
  - 並設置對映的 bitmap



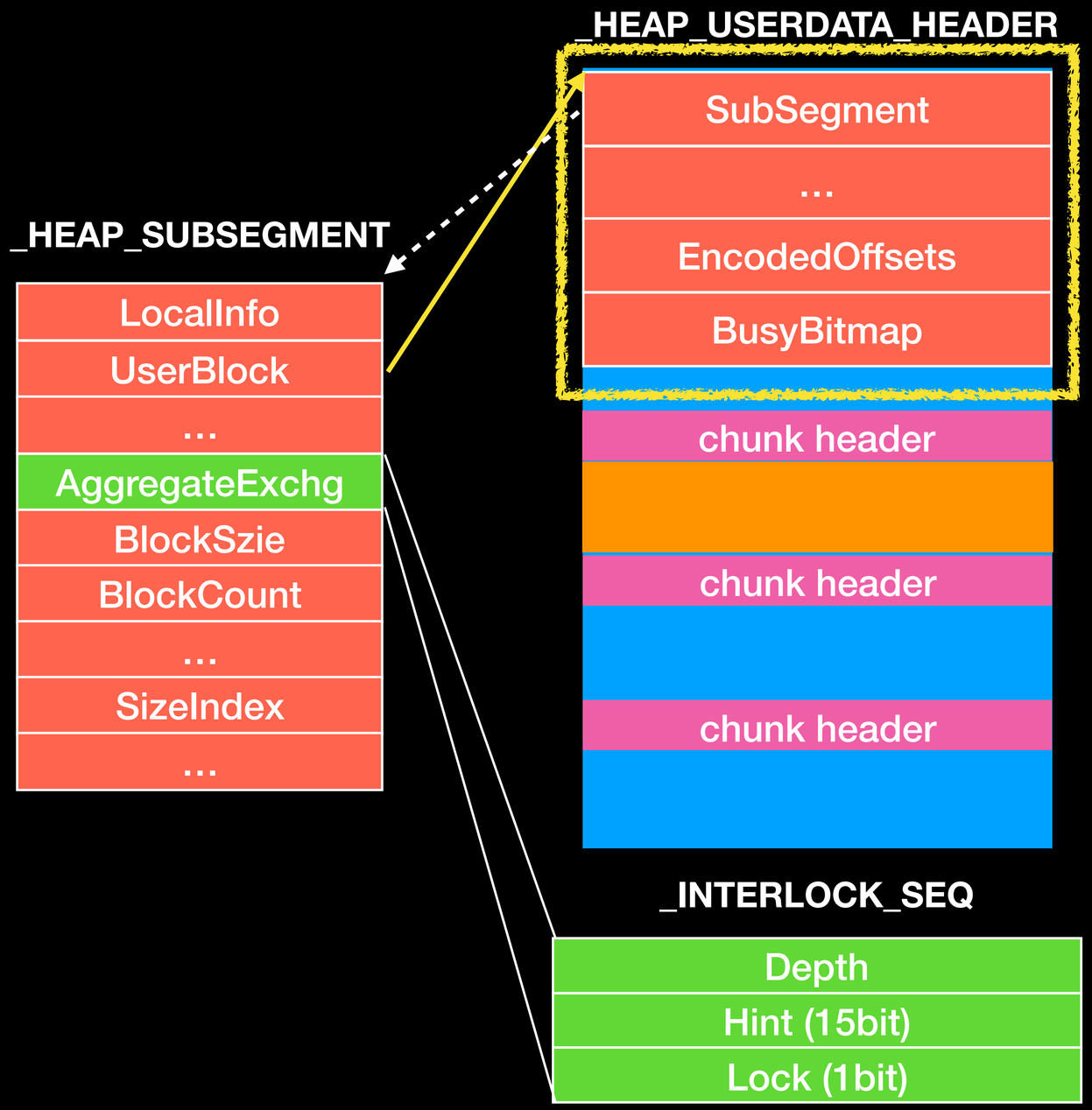
- 算出來的 index 發現已用過
  - 會取下一個
  - 並設置對映的 bitmap

Windows memory allocator

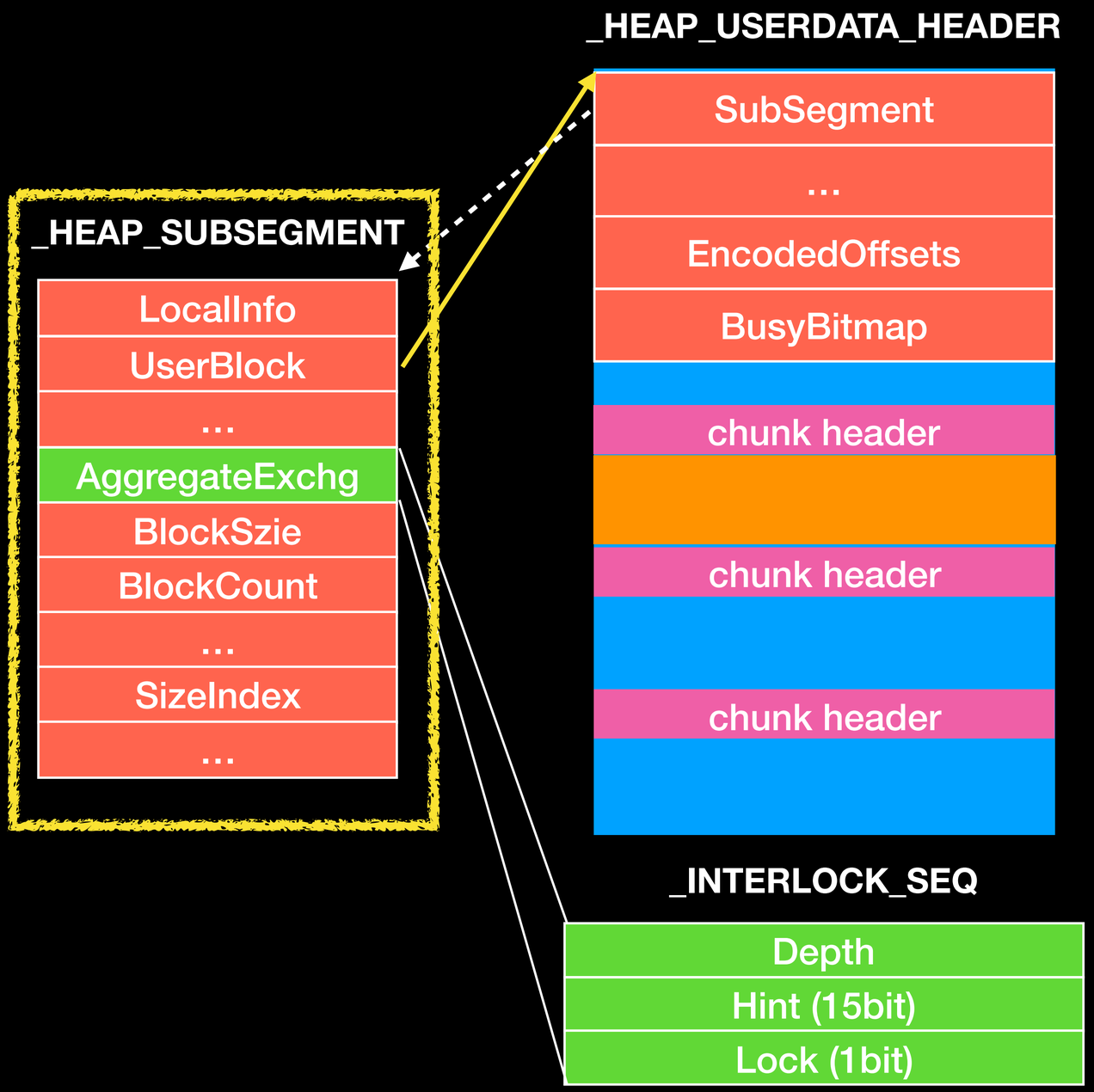
- LFH
  - Free (RtlFreeHeap)
    - Update unused byte in chunk header
    - Find the index of the Chunk and reset Userblock->BusyBitmap
    - Update ActiveSubsegment->AggregateExchg
    - 如果 Free 的 chunk 不屬於當前 ActiveSubsegment 則會看看能不能放進 cachedItems ，可以放就放進去



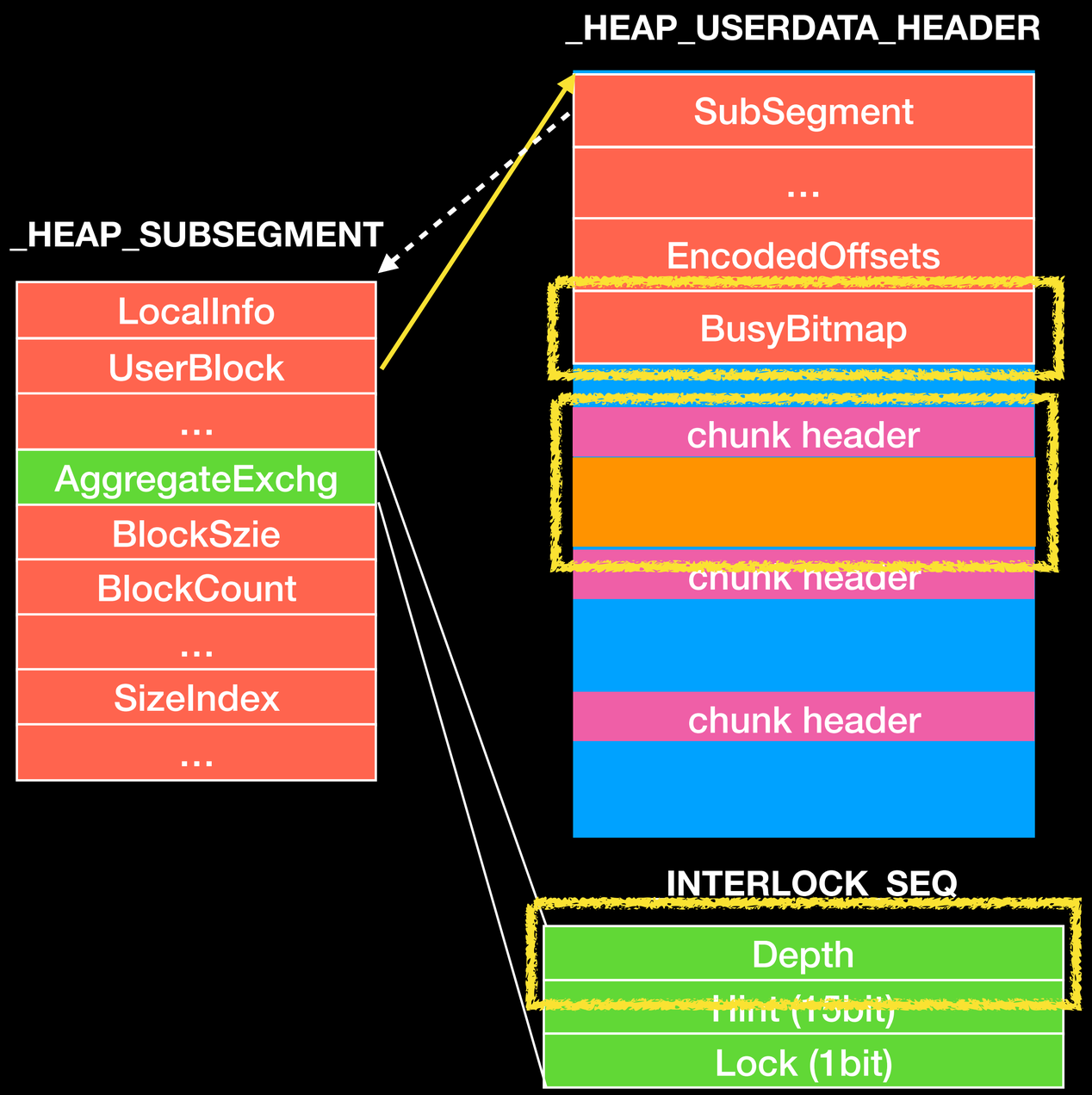
- Free
  - 用 chunk header 尋找 UserBlock



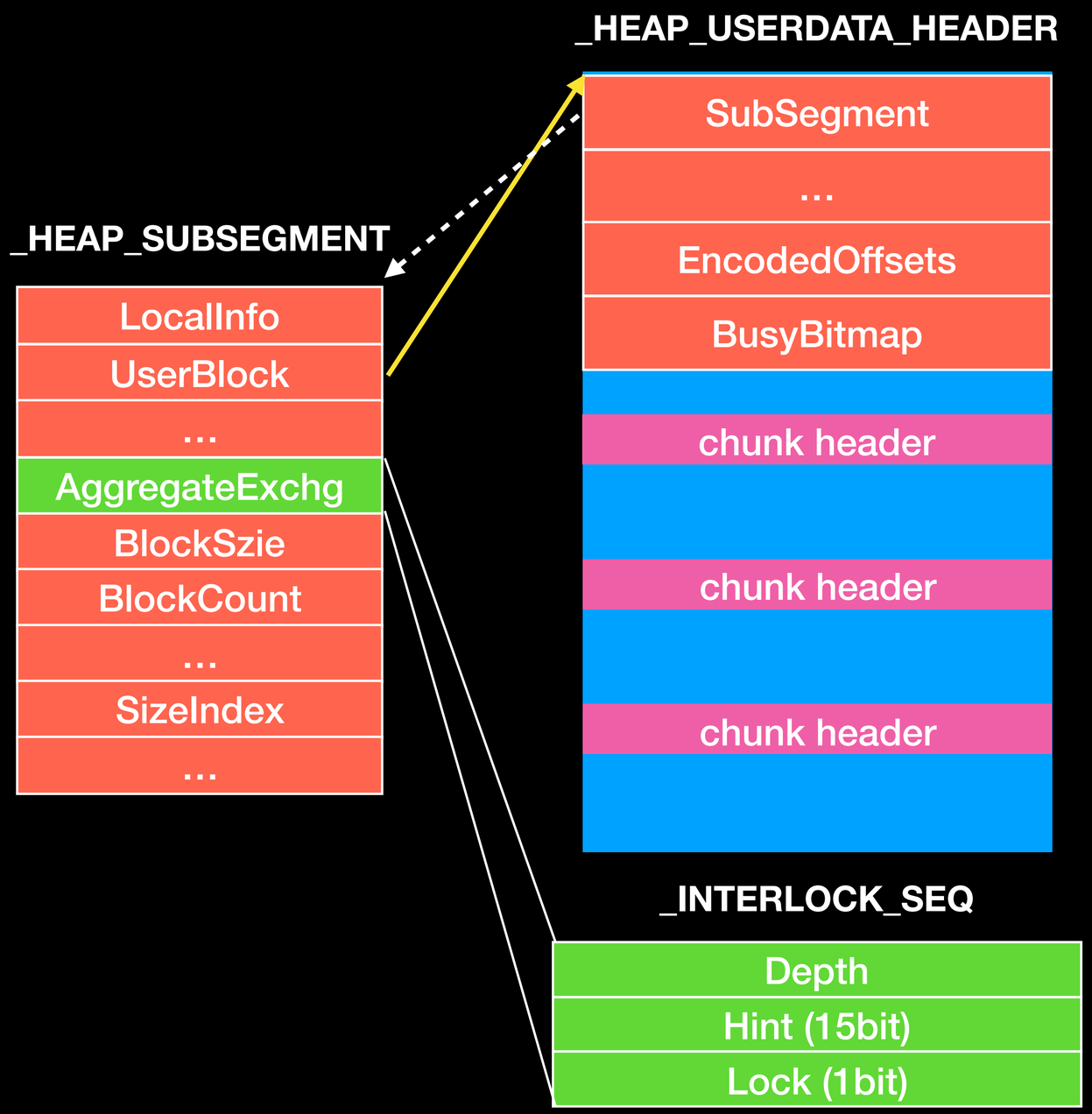
- Free
  - 用 chunk header 尋找 UserBlock
  - 找回對映的 SubSegment



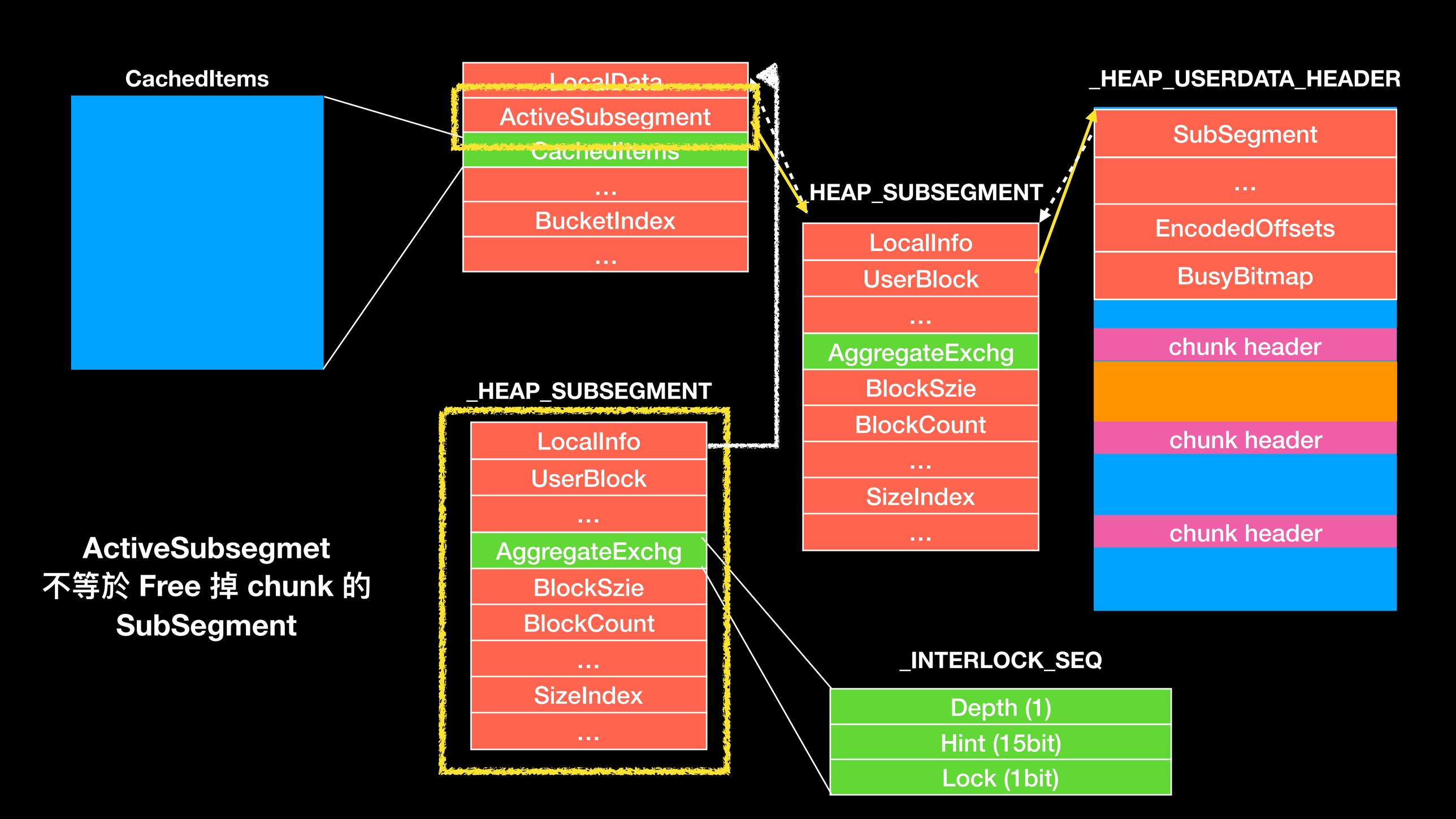
- Free
  - 用 chunk header 尋找 UserBlock
  - 找回對映的 SubSegment
  - 設置 unused byte = 0x80
  - 清除對映的 bitmap
  - Update AggregateExchg

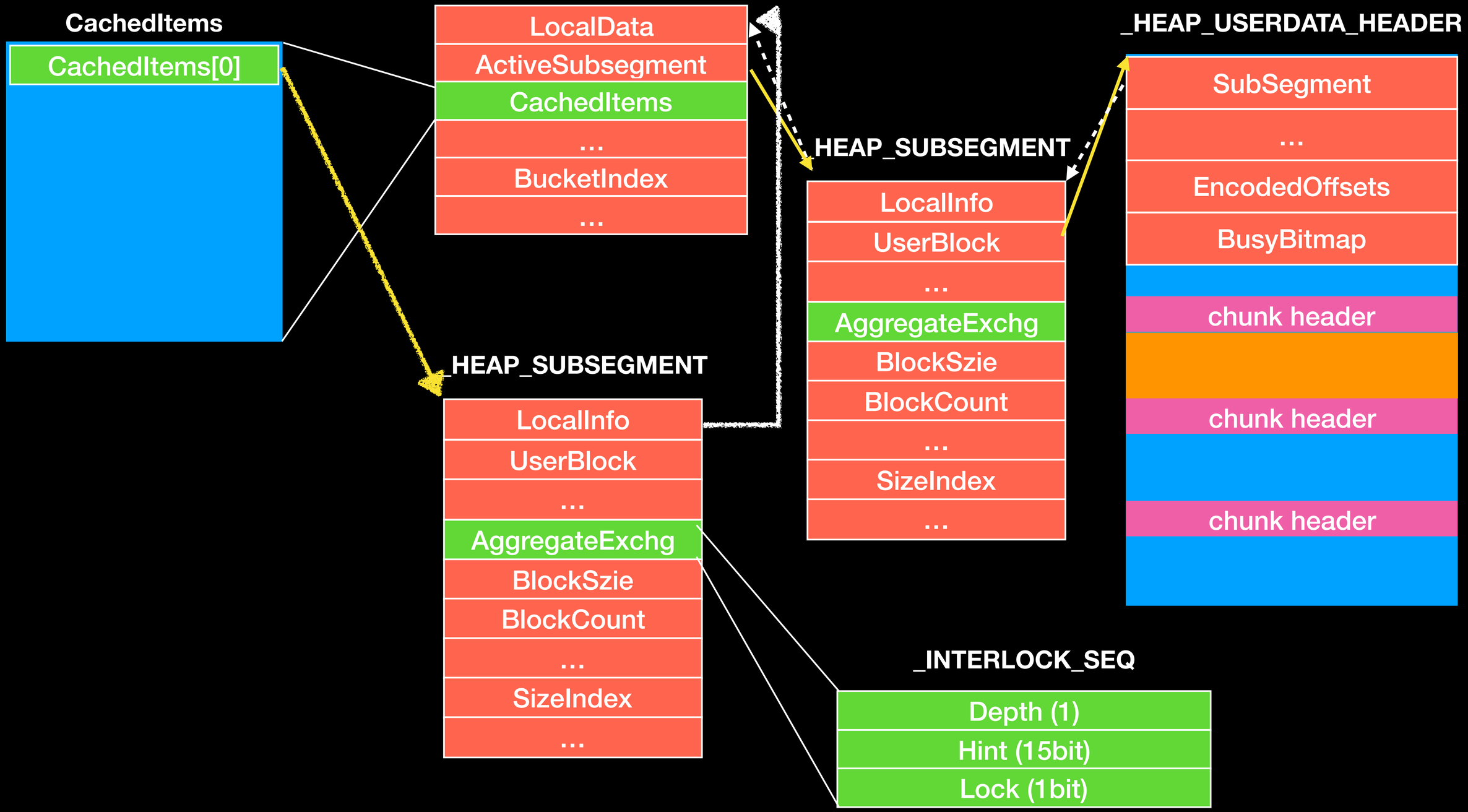


- Free
  - 用 chunk header 尋找 UserBlock
  - 找回對映的 SubSegment
  - 設置 unused byte = 0x80
  - 清除對映的 bitmap
  - Update AggregateExchg



- Free
  - 用 chunk header 尋找 UserBlock
  - 找回對映的 SubSegment
  - 設置 unused byte = 0x80
  - 清除對映的 bitmap
  - Update AggregateExchg


CachedItems
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_INTERLOCK_SEQ
Depth (1)
Hint (15bit)
Lock (1bit)
ActiveSubsegmet
不等於 Free 掉 chunk 的
SubSegment

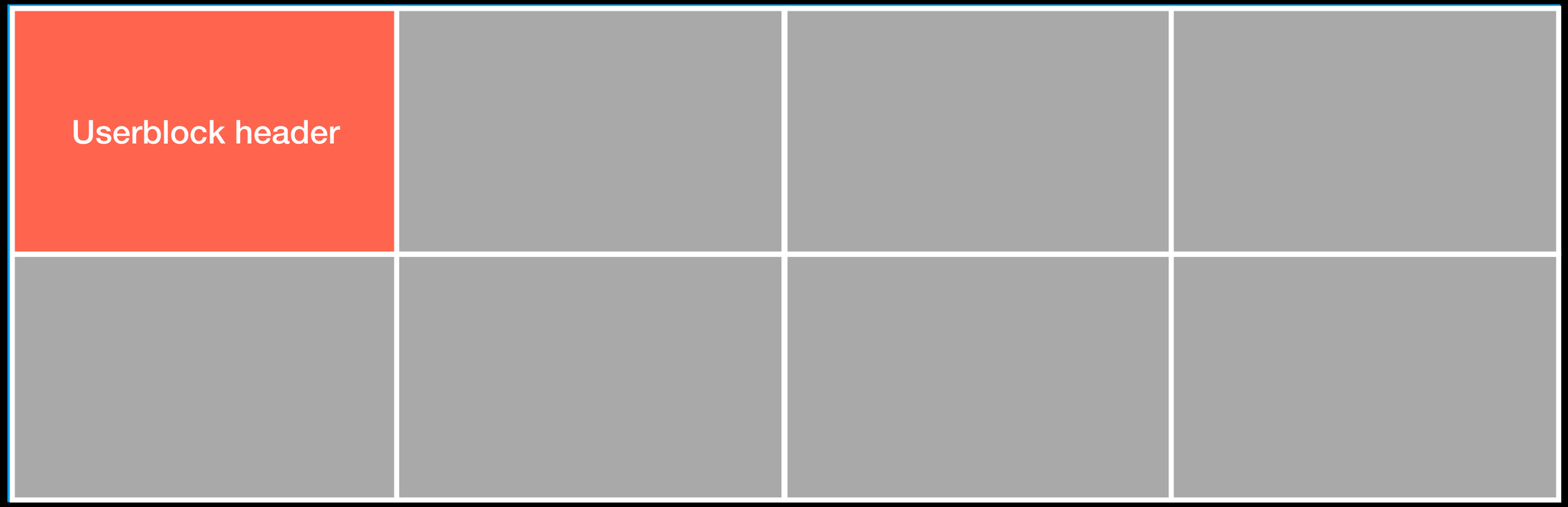

CachedItems
CachedItems[0]
LocalData
ActiveSubsegment
CachedItems
...
BucketIndex
...
HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
HEAP_SUBSEGMENT
LocalInfo
UserBlock
...
AggregateExchg
BlockSzie
BlockCount
...
SizeIndex
...
_HEAP_USERDATA_HEADER
SubSegment
...
EncodedOffsets
BusyBitmap
chunk header
chunk header
chunk header
_INTERLOCK_SEQ
Depth (1)
Hint (15bit)
Lock (1bit)

LFH Exploitation

- Reuse attack
  - 假設我們有 Use after free 的漏洞
    - 但因為 LFH 的隨機性，導致無法預側下一塊 chunk 會在哪，使得我們很難利用
    - 這時可以填滿 Userblock 的方式，在 free 掉其中一塊，那麼下次該 chunk 必定會拿到同一塊，我們可以利用這個特性，拿到 overlap chunk 做進一步利用

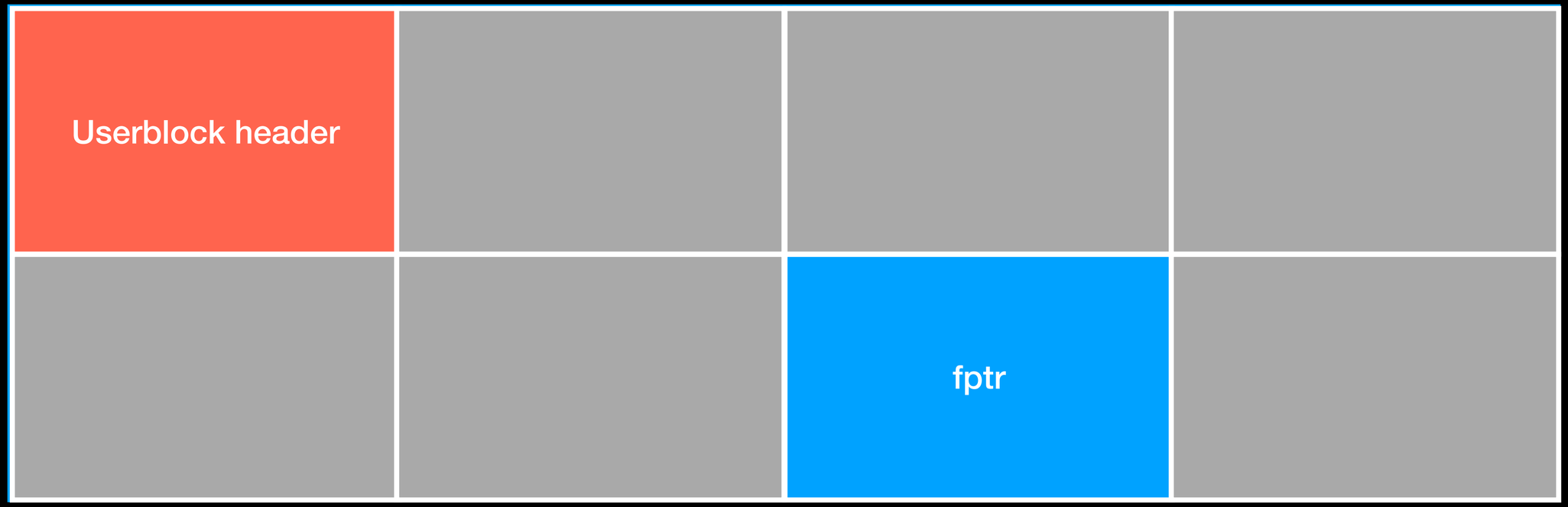
LFH Exploitation

- normal case
  - malloc(sizeof(A))



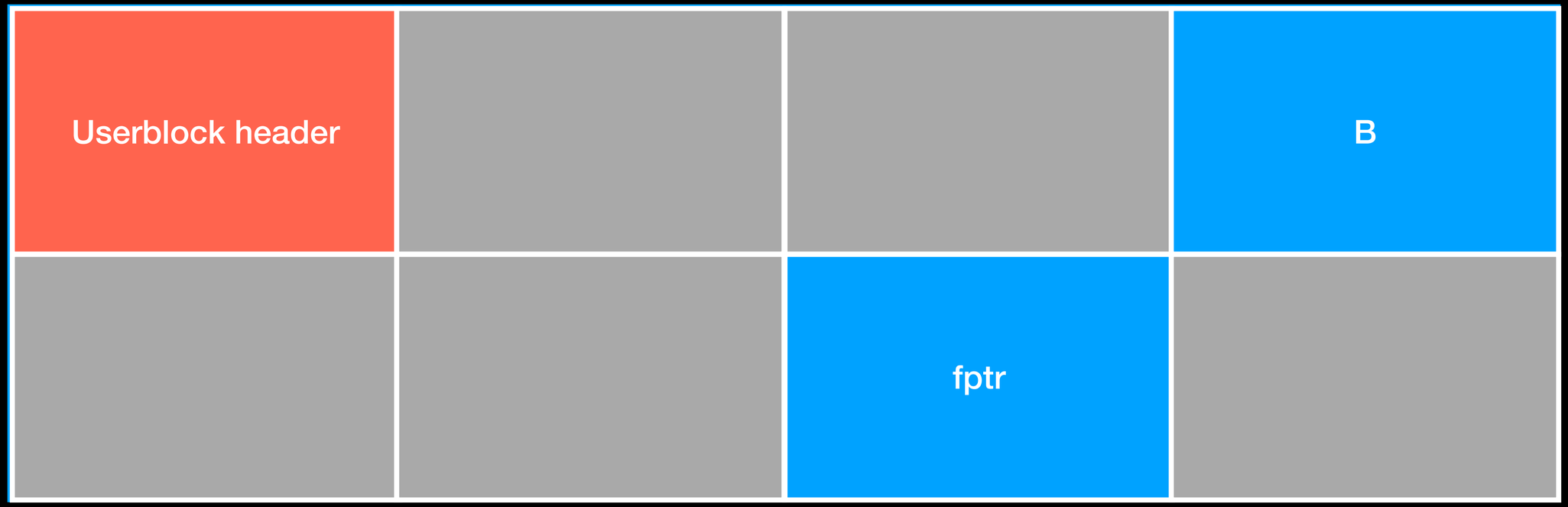
LFH Exploitation

- normal case
  - malloc(sizeof(B))



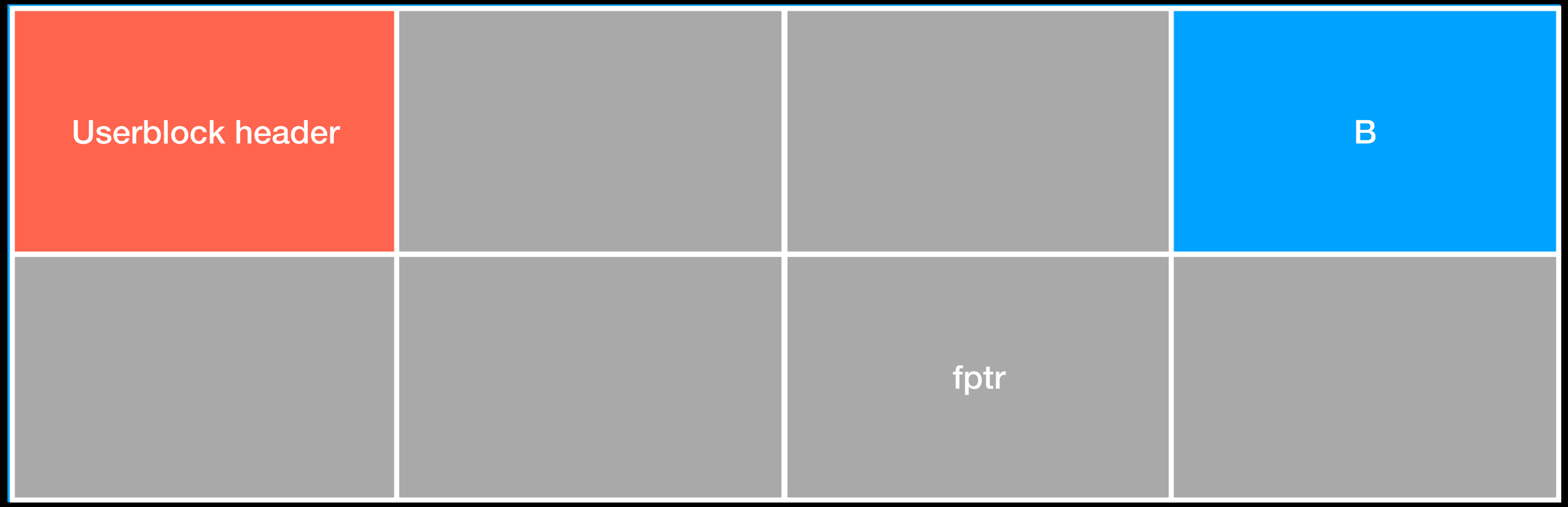
LFH Exploitation

- normal case
  - free(A)



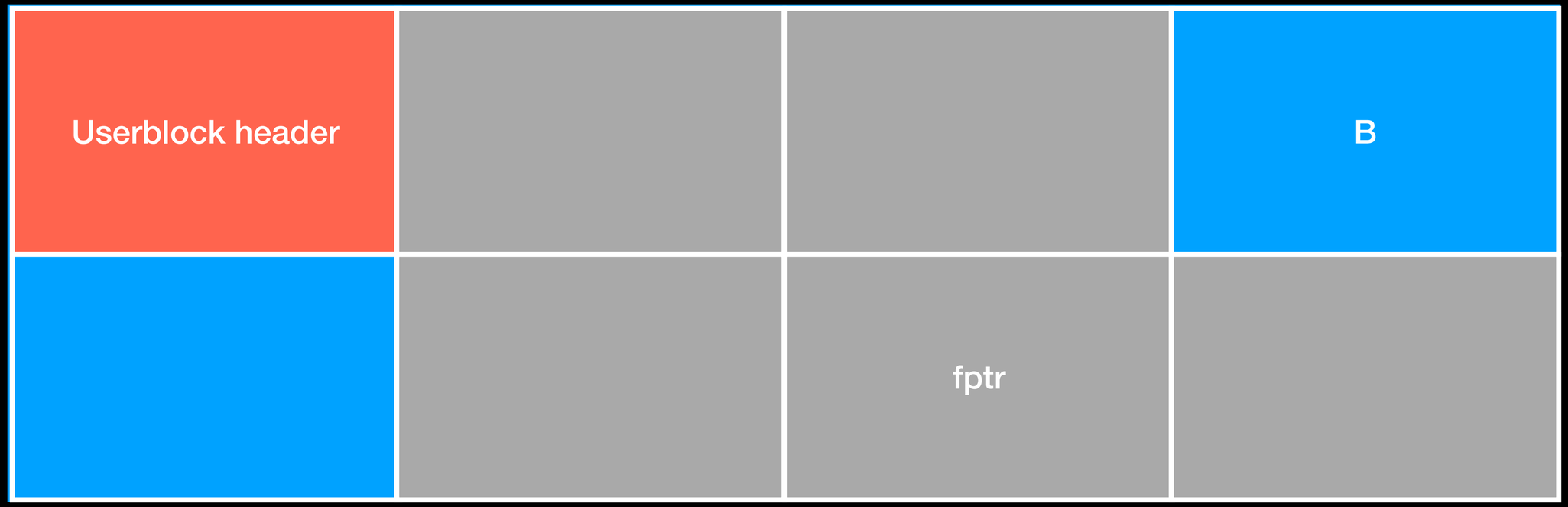
LFH Exploitation

- normal case
  - malloc(sizeof(A))



LFH Exploitation

- normal case
  - malloc(sizeof(B))

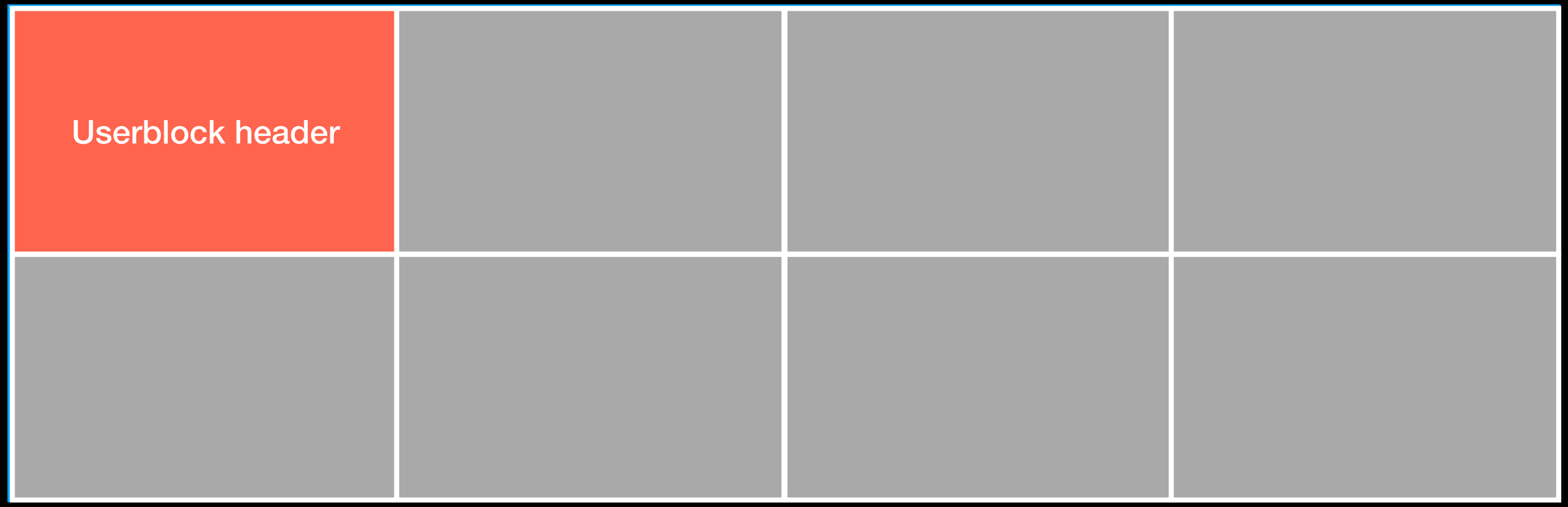


LFH Exploitation

- normal case
  - malloc(sizeof(B))

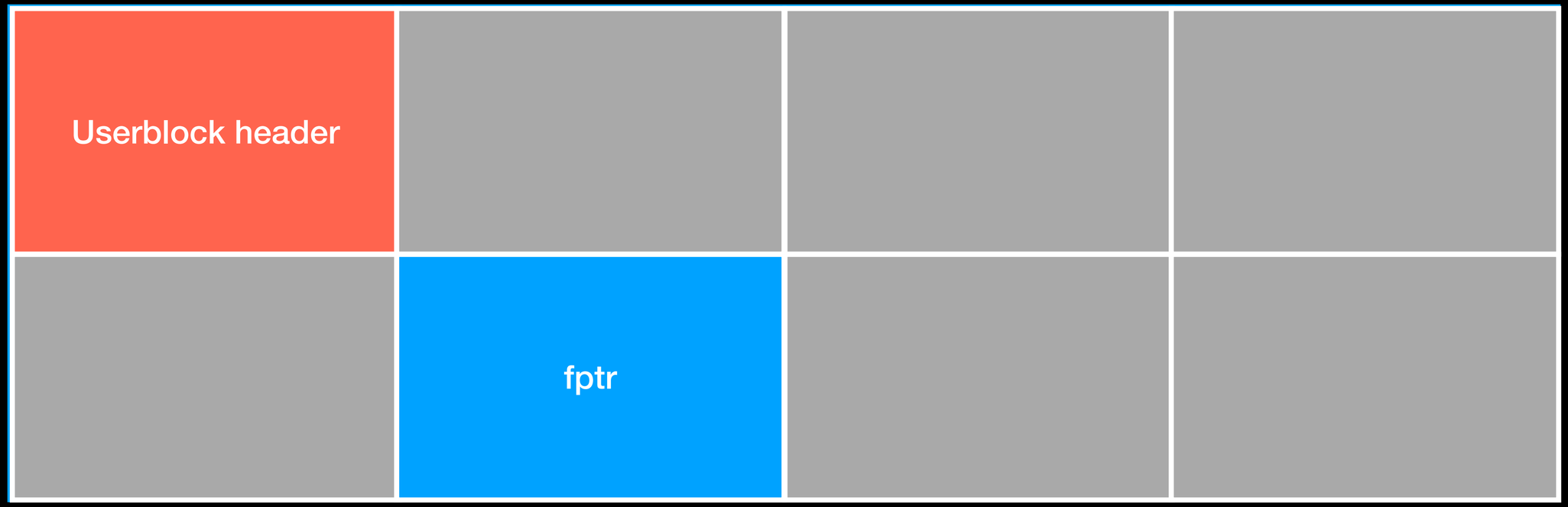
LFH Exploitation

- Reuse attack
  - malloc(sizeof(A))



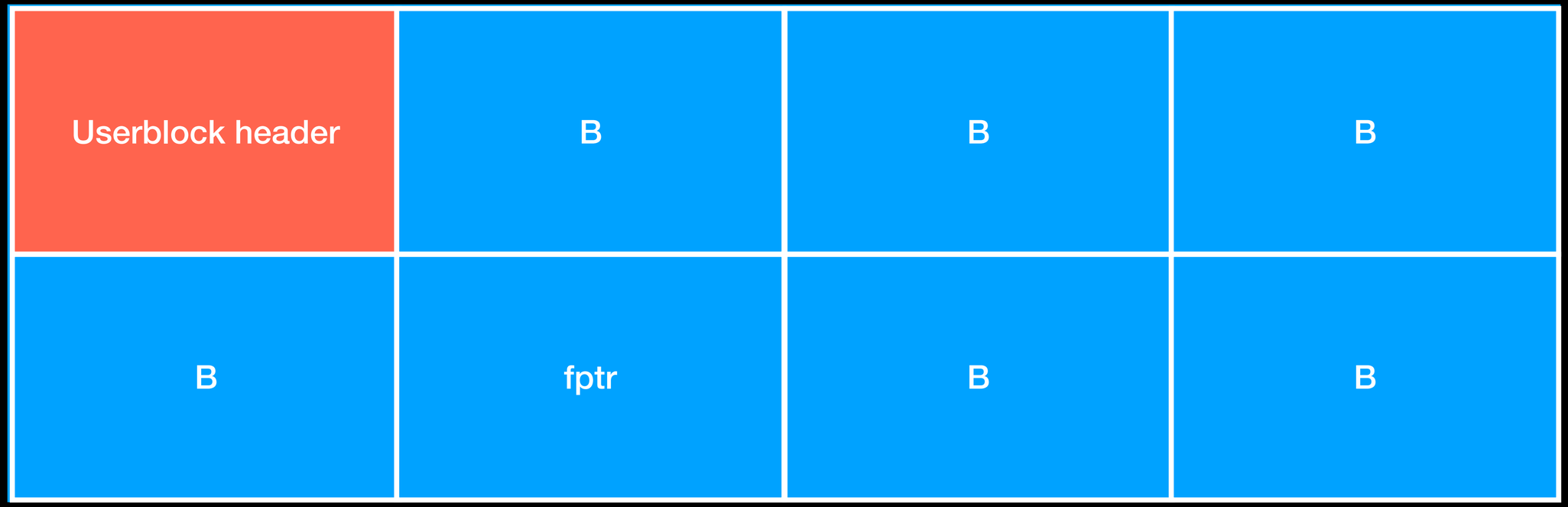
LFH Exploitation

- Reuse attack
  - malloc(sizeof(B)) x 6



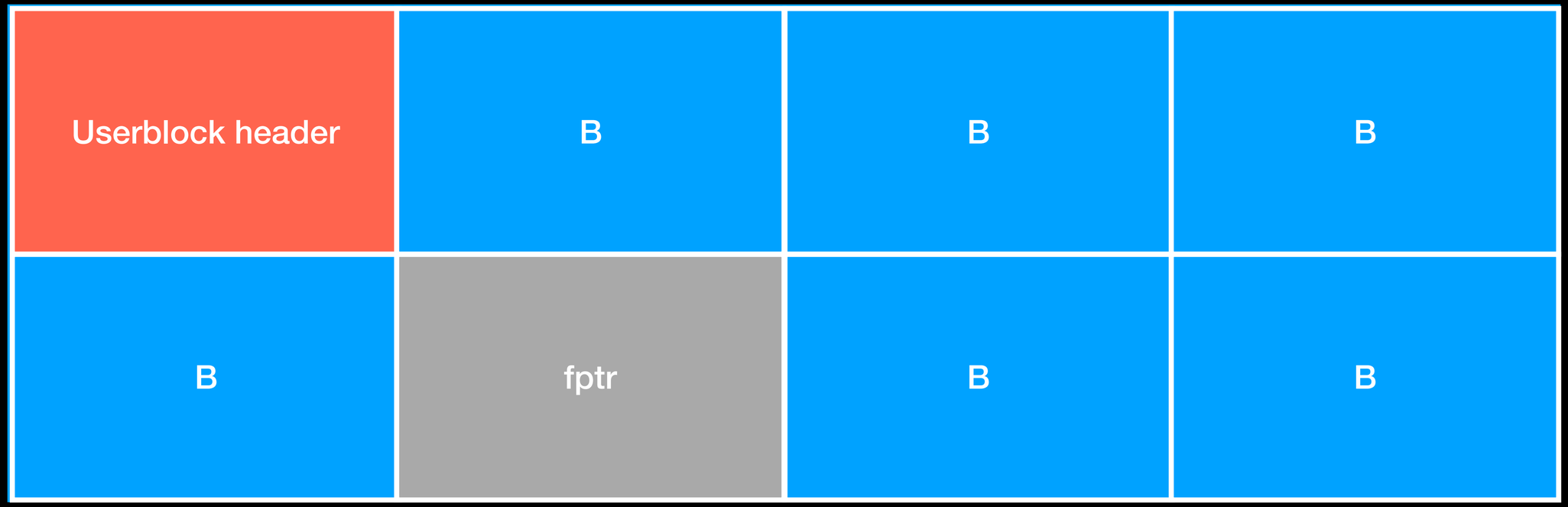
LFH Exploitation

- Reuse attack
  - free(A)



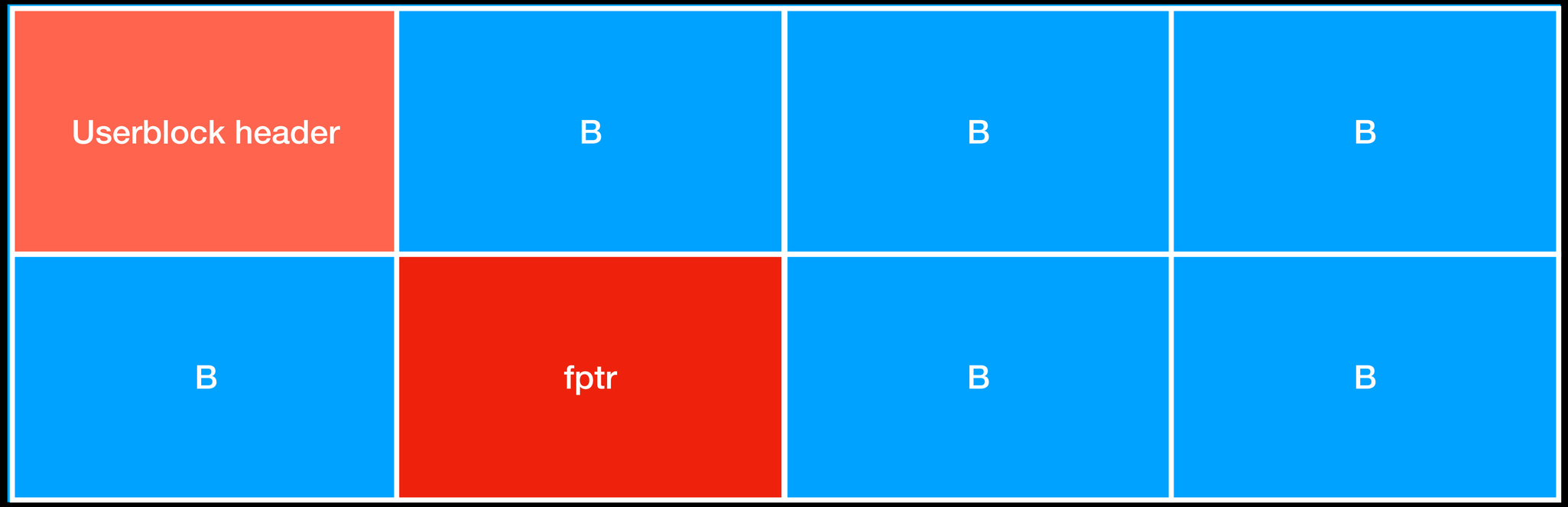
LFH Exploitation

- Reuse attack
  - malloc(sizeof(B))



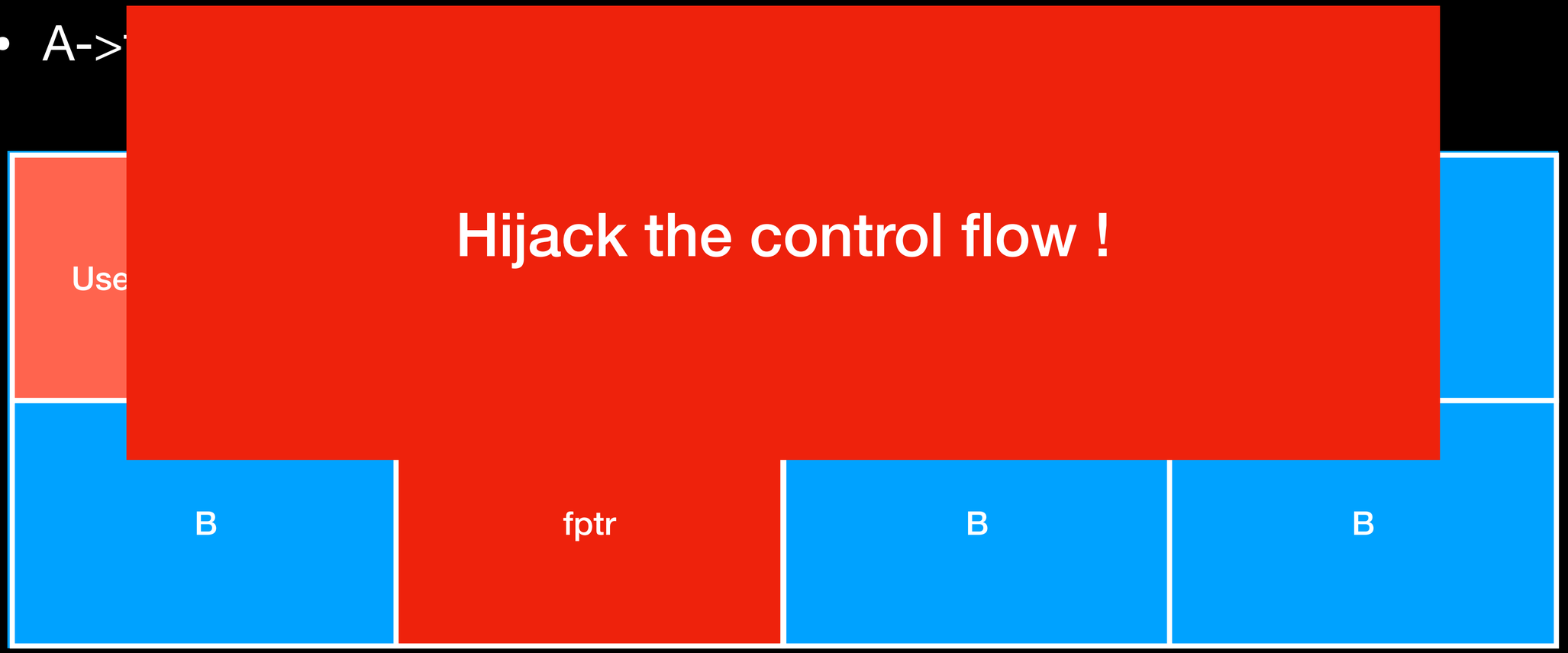
LFH Exploitation

- Reuse attack
  - A->fptr



LFH Exploitation

- Reuse attack
  - A->



Reference

- https://github.com/saaramar/35C3_Modern_Windows_Userspace_Exploitation
- http://illmatics.com/Understanding_the_LFH.pdf
- https://github.com/saaramar/Deterministic_LFH

Thank you for listening

 angelboy@chroot.org

 @scwuaptx