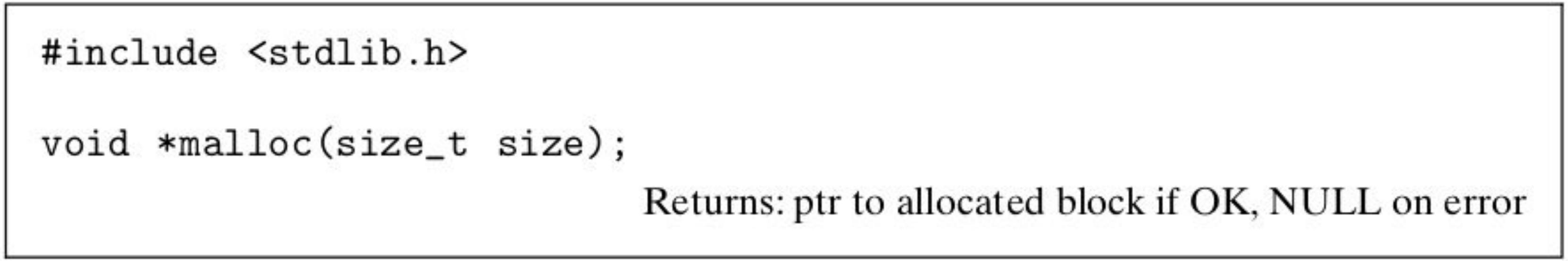
malloc

malloc()

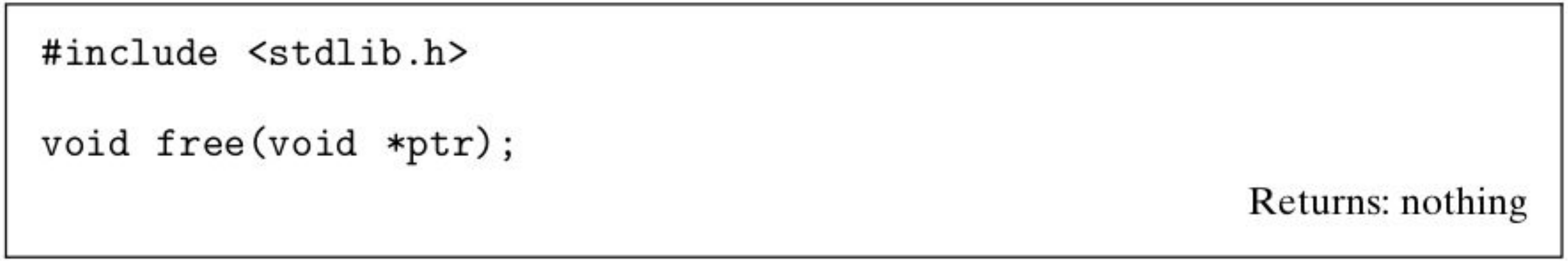


```
#include <stdlib.h>

void *malloc(size_t size);
```

Returns: ptr to allocated block if OK, NULL on error

- En unix malloc() devuelve un bloque de size bytes alineado a 8-bytes (double word).
- No inicializa la memoria devuelta.
- Utiliza la system call sbrk o mmap.

free()



```
#include <stdlib.h>

void free(void *ptr);
```

Returns: nothing

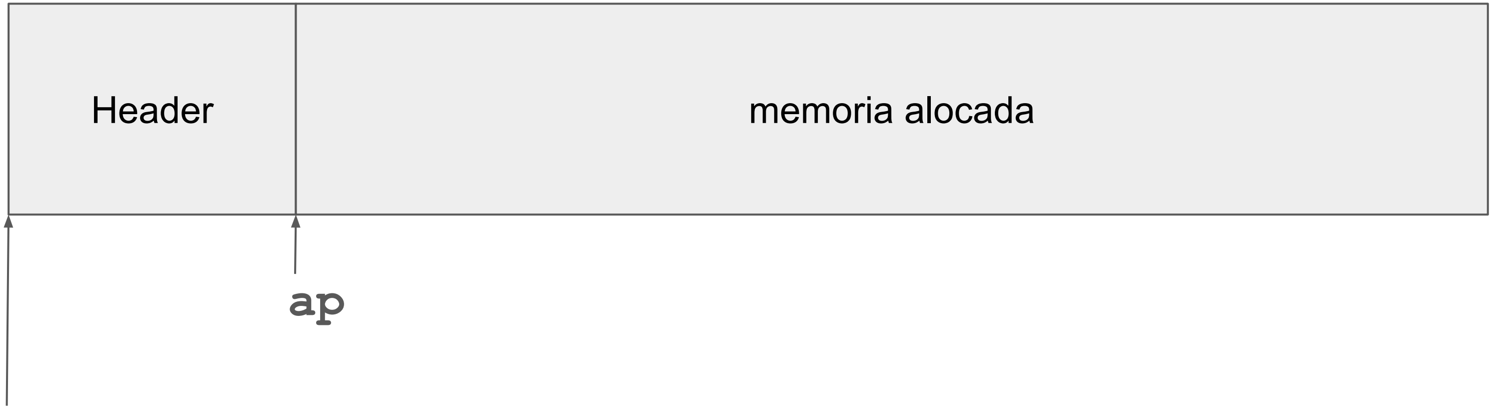
Libera bloques reservados en el heap.

El ptr debe haber sido reservado previamente con malloc(), calloc() o realloc(). Si esto no sucede el comportamiento de free es INDEFINIDO

Ejemplo

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int *p1 = malloc(4*sizeof(int));  // allocates enough for an array of 4 int
    int *p2 = malloc(sizeof(int[4])); // same, naming the type directly
    int *p3 = malloc(4*sizeof *p3);   // same, without repeating the type name

    if(p1) {
        for(int n=0; n<4; ++n) // populate the array
            p1[n] = n*n;
        for(int n=0; n<4; ++n) // print it back out
            printf("p1[%d] == %d\n", n, p1[n]);
    }

    free(p1);
    free(p2);
    free(p3);
}
```

`ap = malloc(n)`



Header

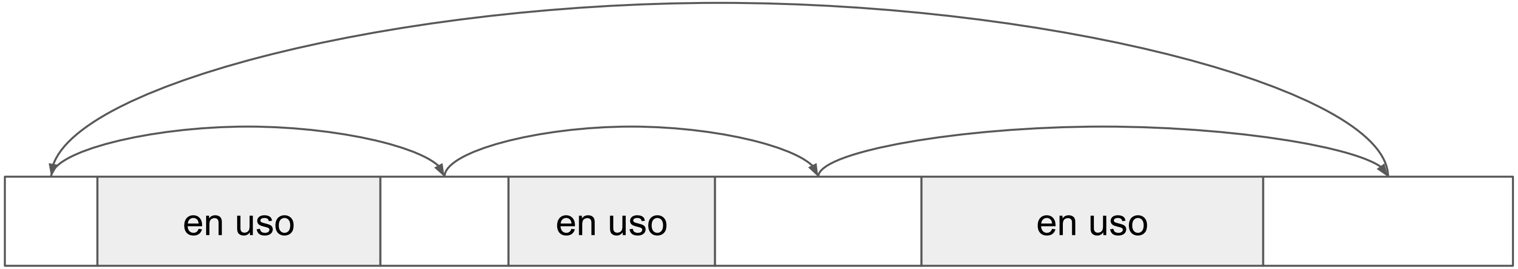
memoria alocada

`ap`

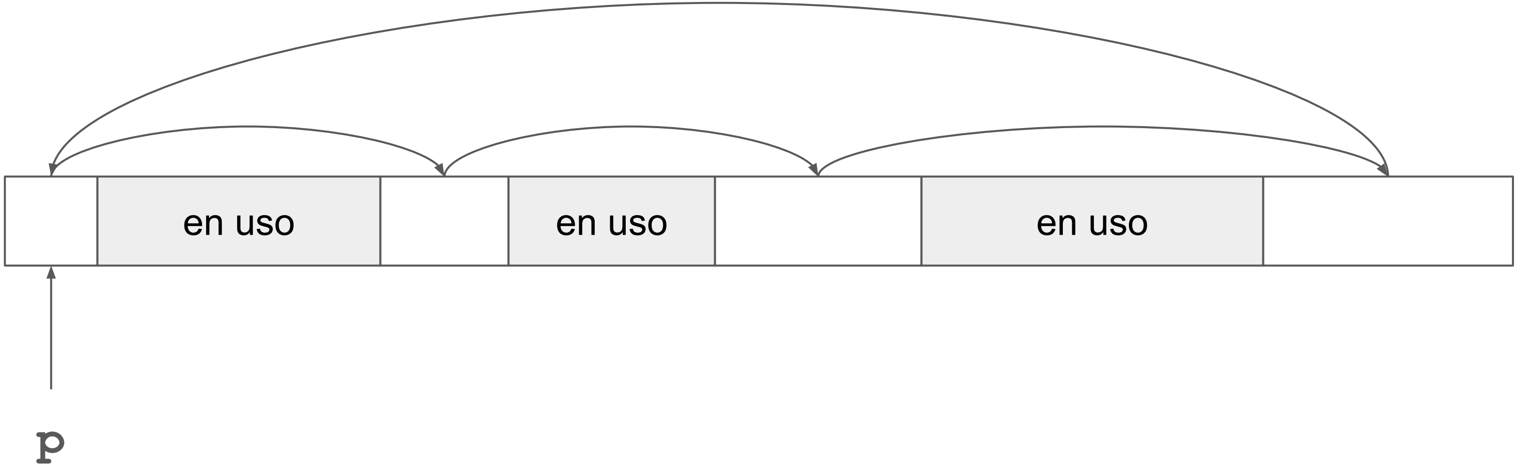
`bp =(Header *)addr - 1`

Estructura Header

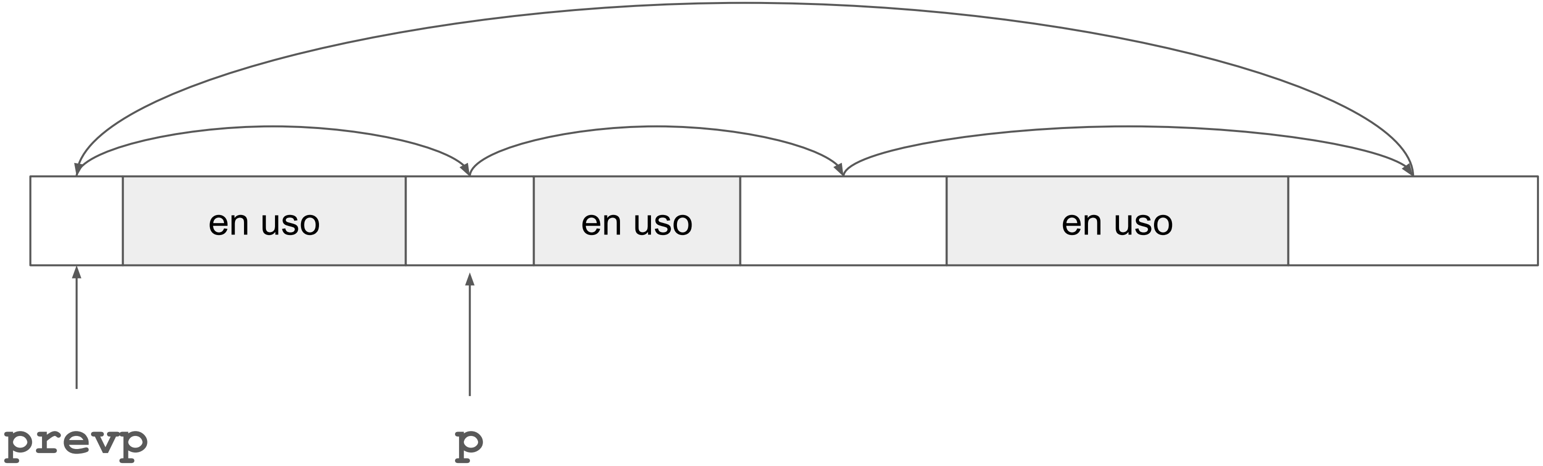
```
struct header {
  struct header *ptr;
  unsigned int size;
};

typedef struct header Header;
```


en uso
en uso
en uso

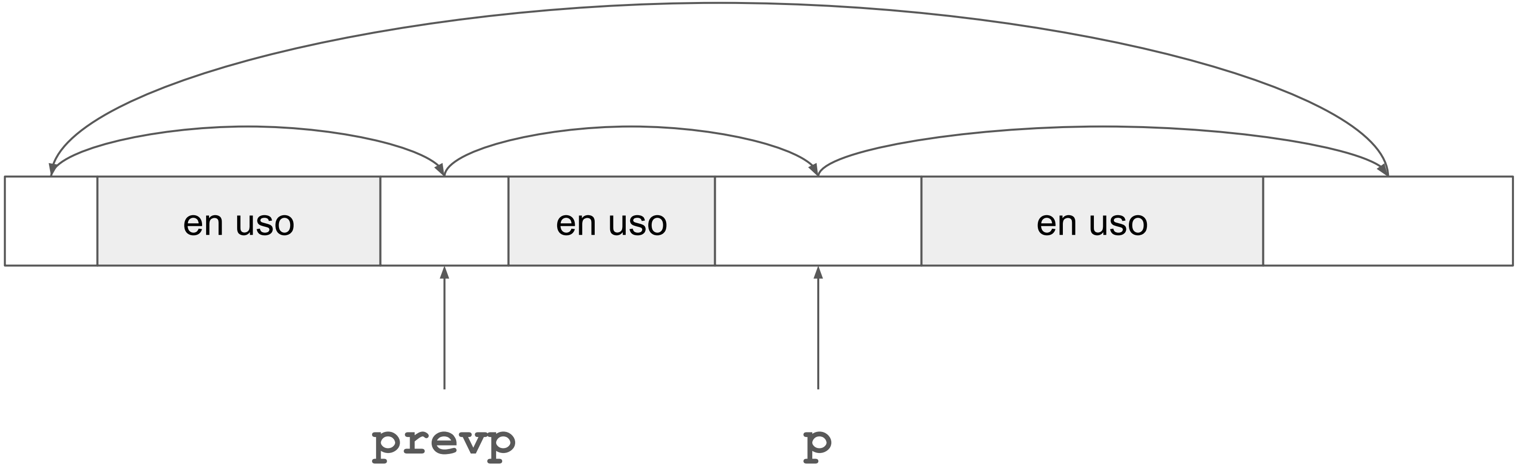
Buscar un bloque libre



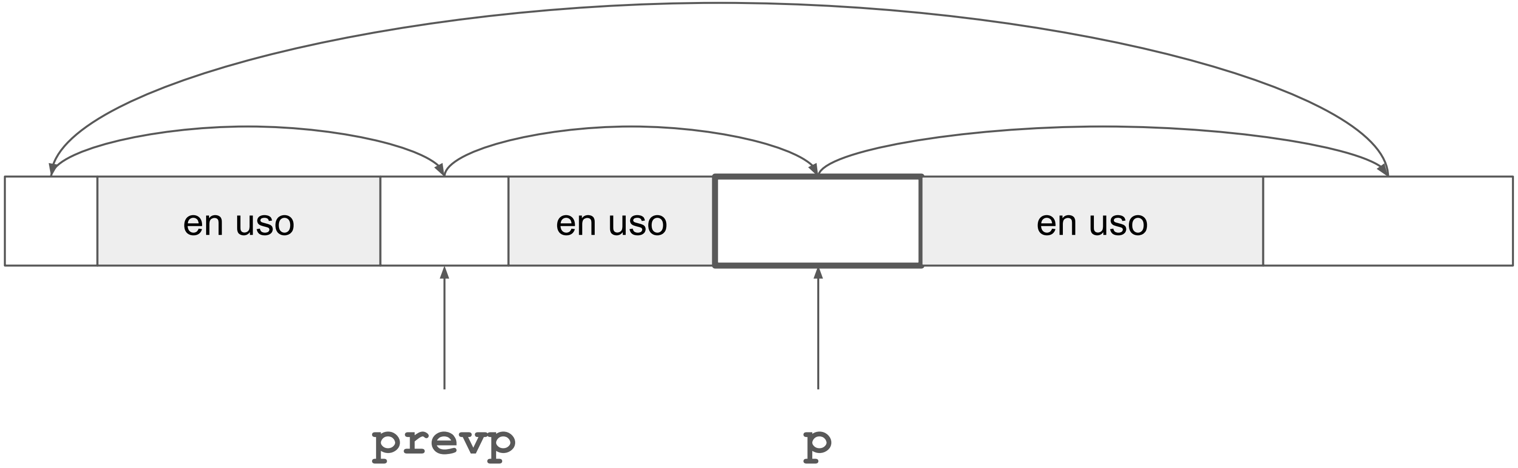
Buscar un bloque libre



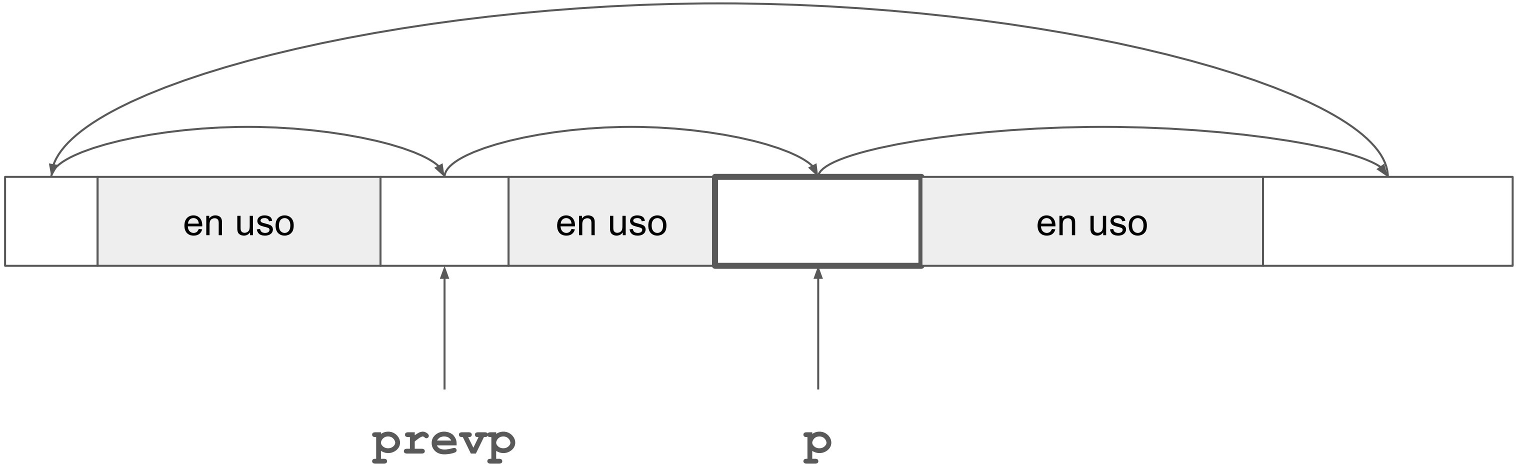
Buscar un bloque libre



Buscar un bloque libre

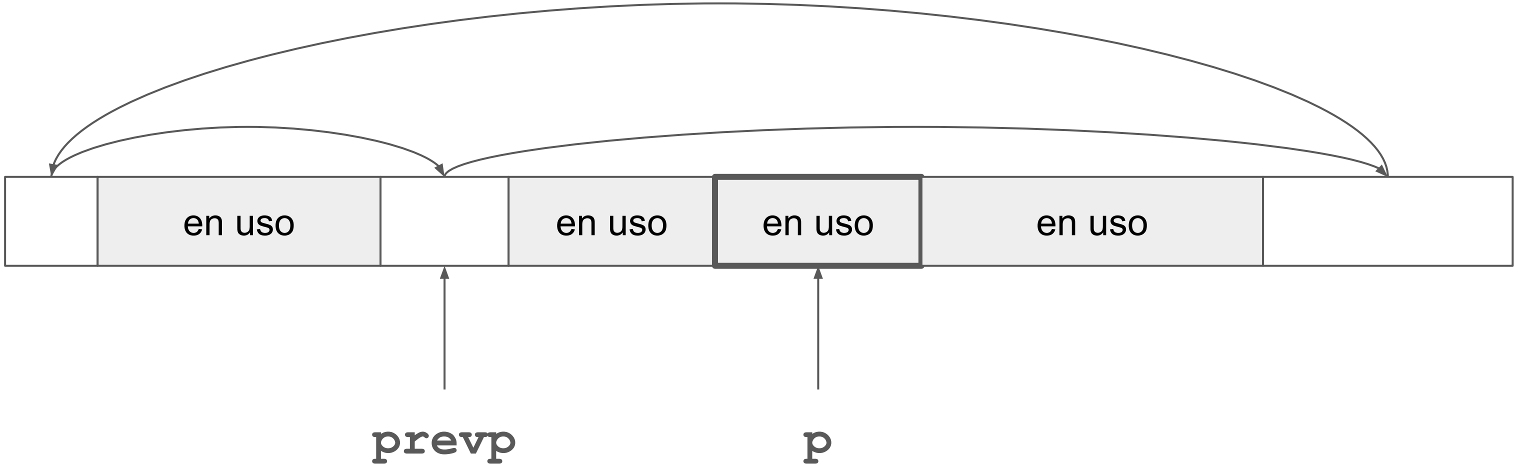


Opcion 1: Reservar el bloque completo



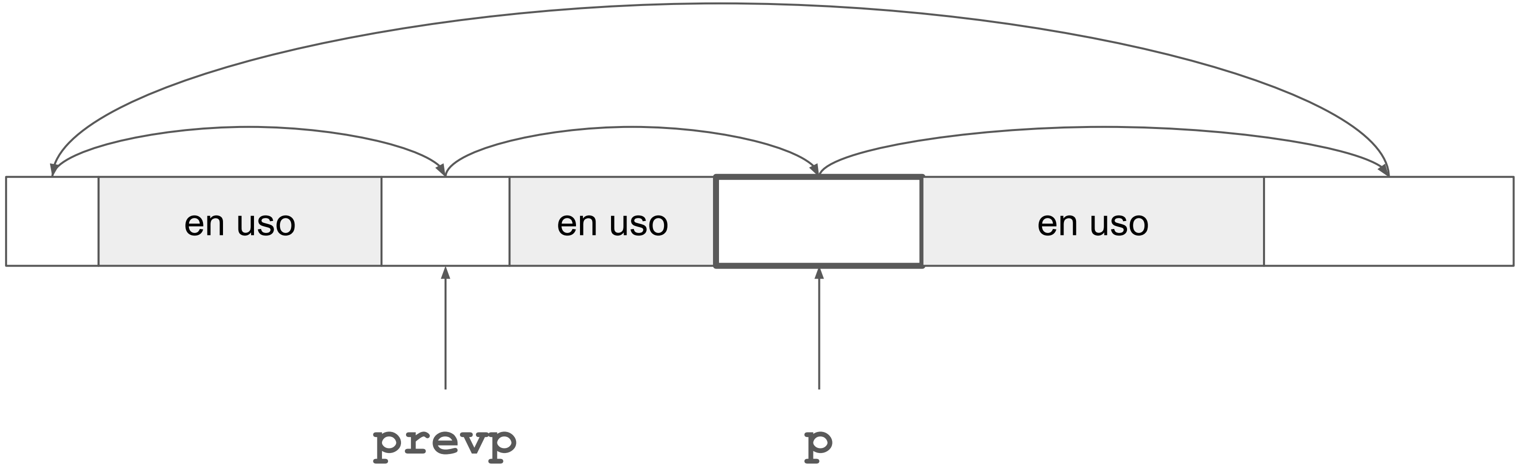
```
prevp->ptr = p-›ptr;
return (p+1); // p es de tipo Header
```

Opcion 1: Reservar el bloque completo

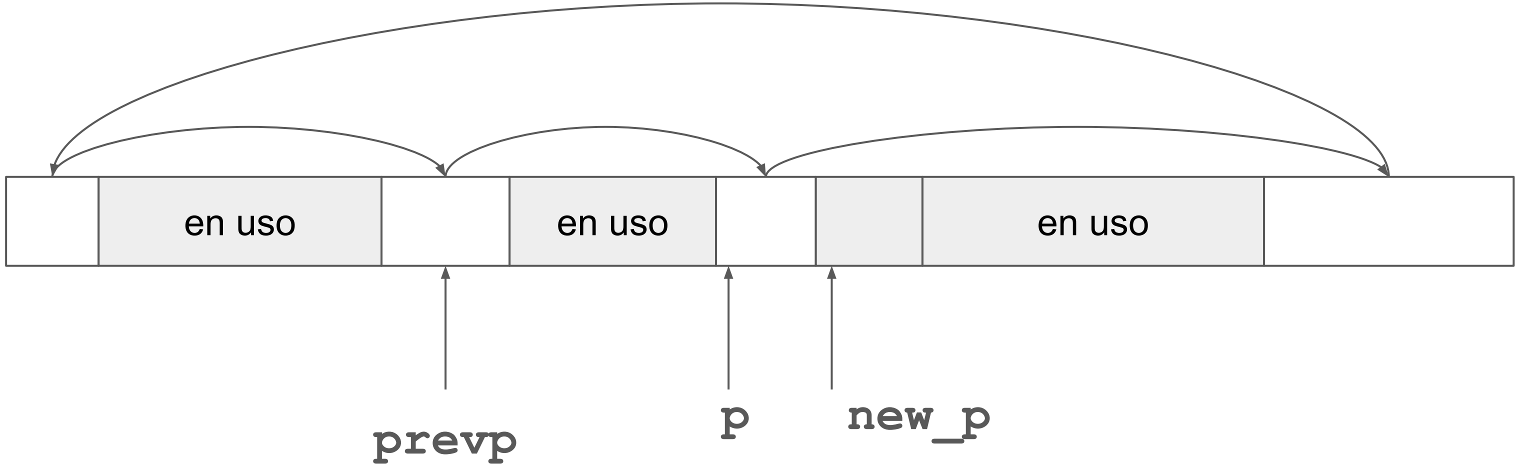


```
prevp->ptr = p-›ptr;
return (p+1); // p es de tipo Header
```

Opcion 2: Split

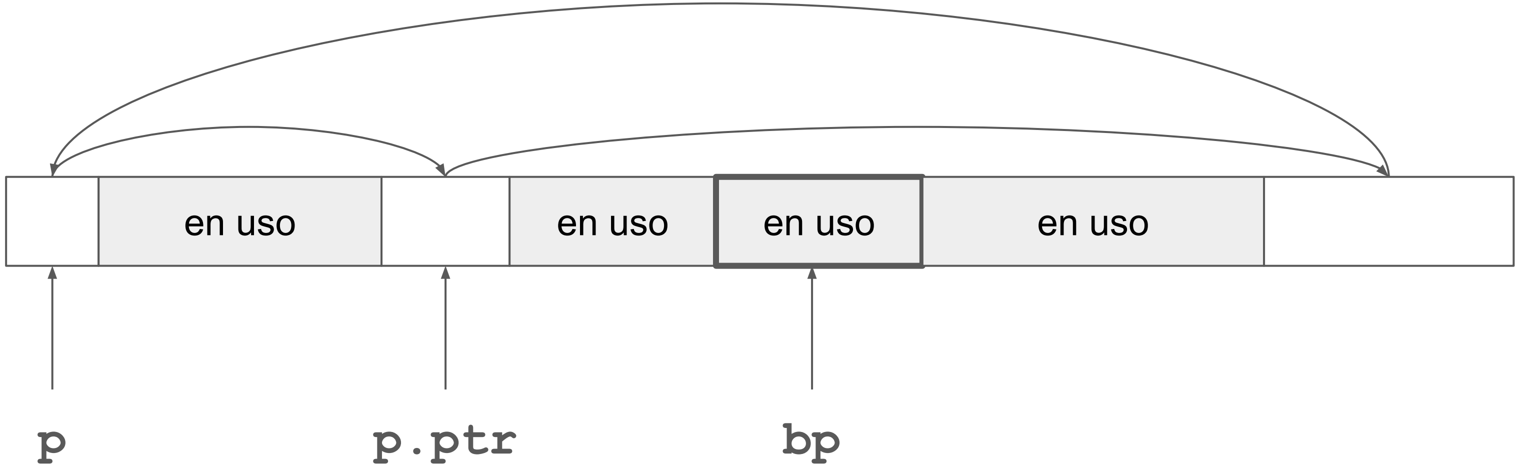


Opcion 2: Reservar parte del bloque

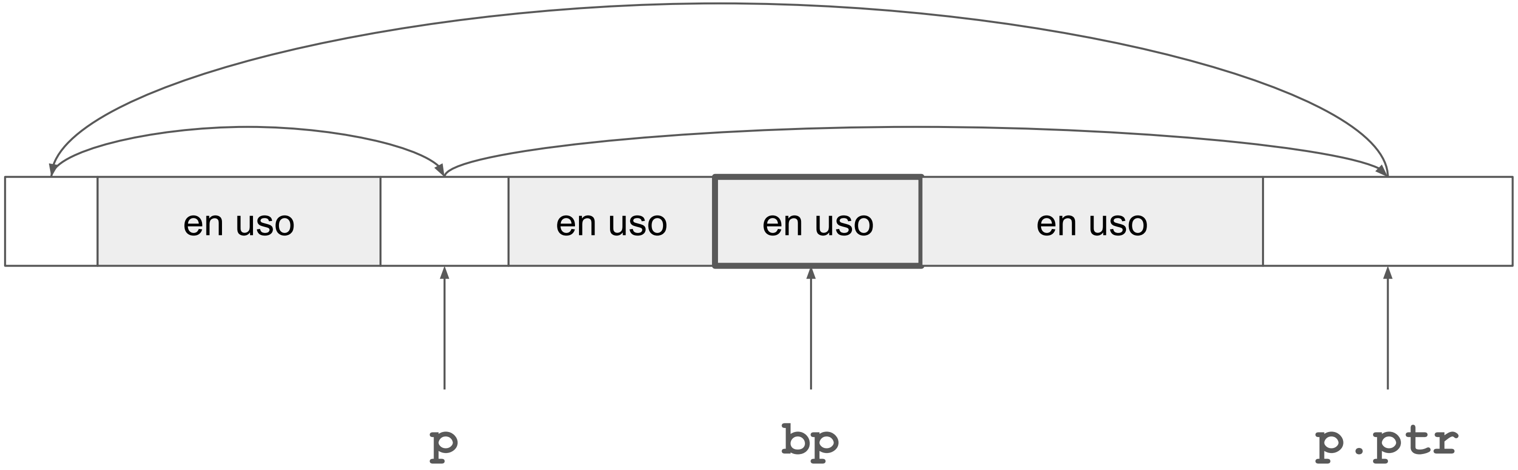


```
p->size =- nunits;
new_p = p + p->size;
new_p->size = nunits;

return (new_p + 1); // new_p es de tipo Header
```

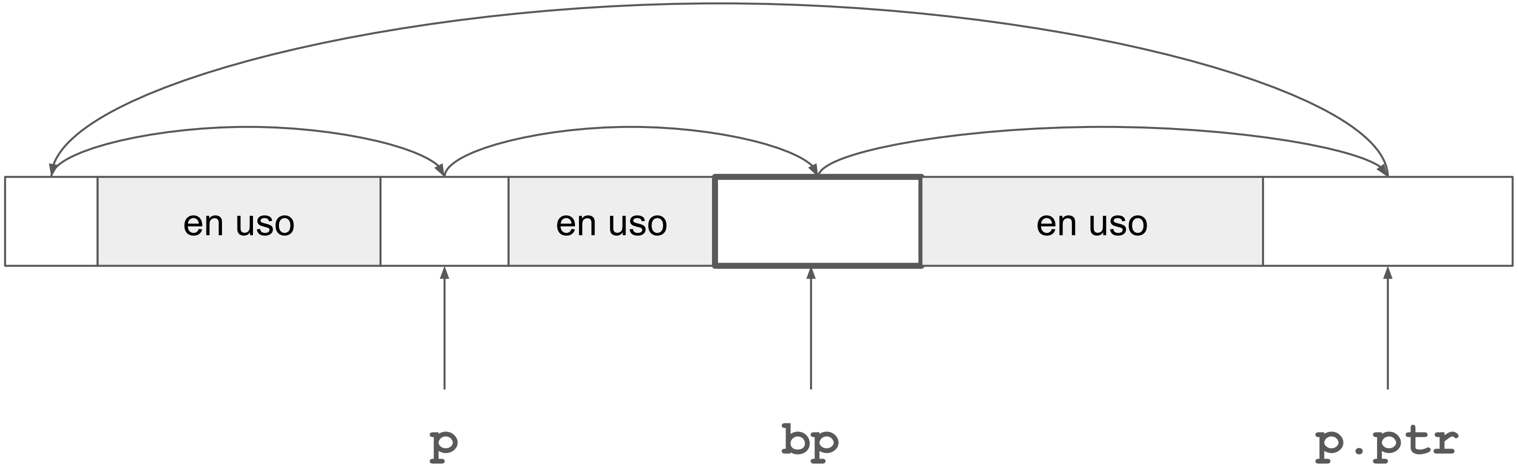
Liberar el bloque



Liberar el bloque

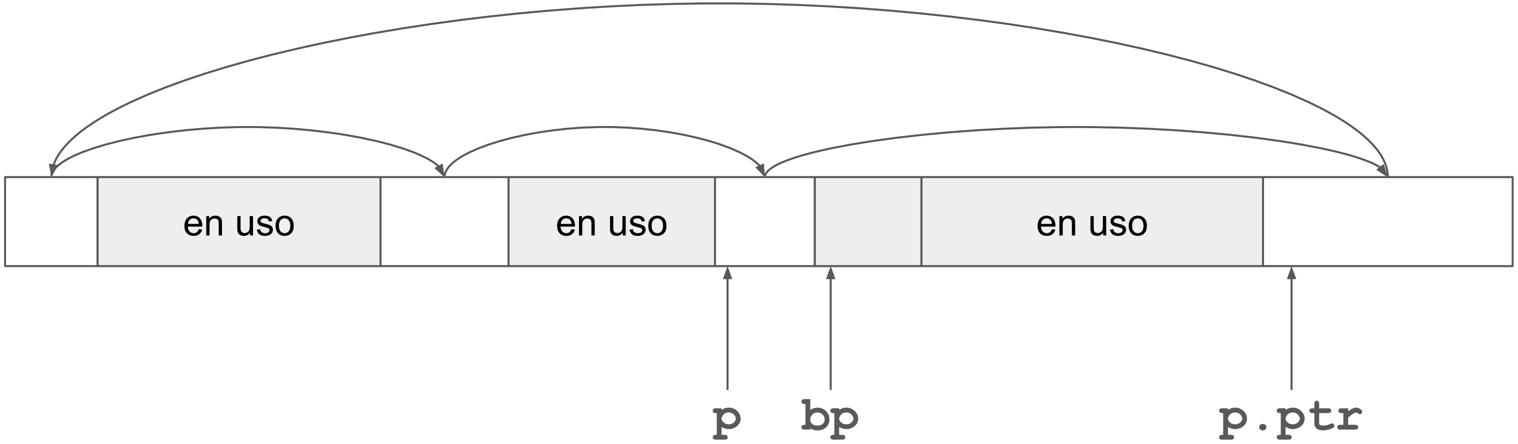


Liberar el bloque

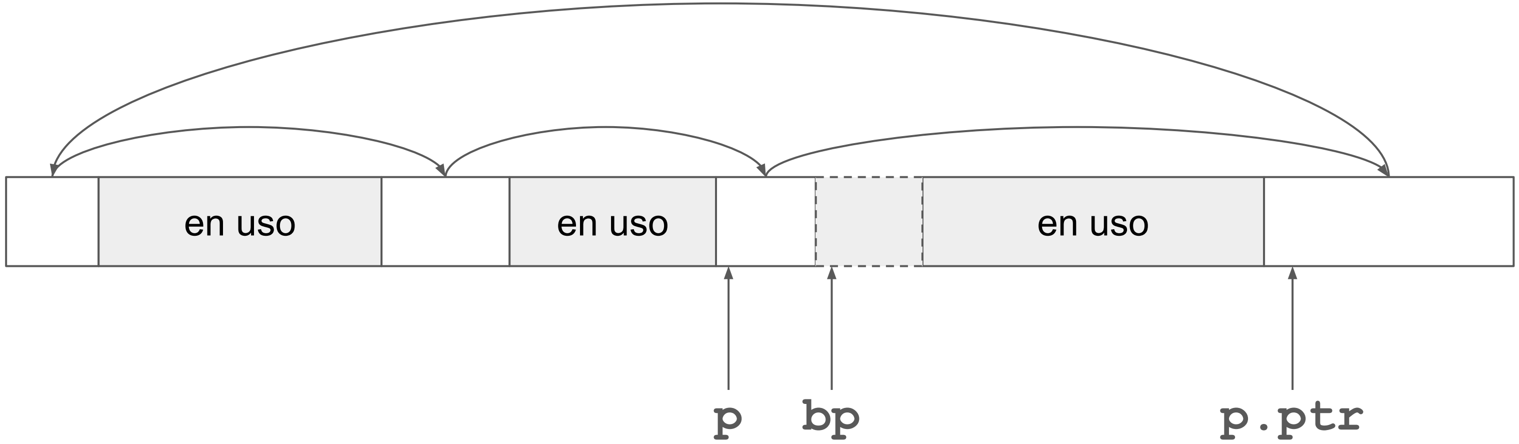


```
bp->ptr = p->ptr
p->ptr = bp;
```

Liberar bloque: Coalesce

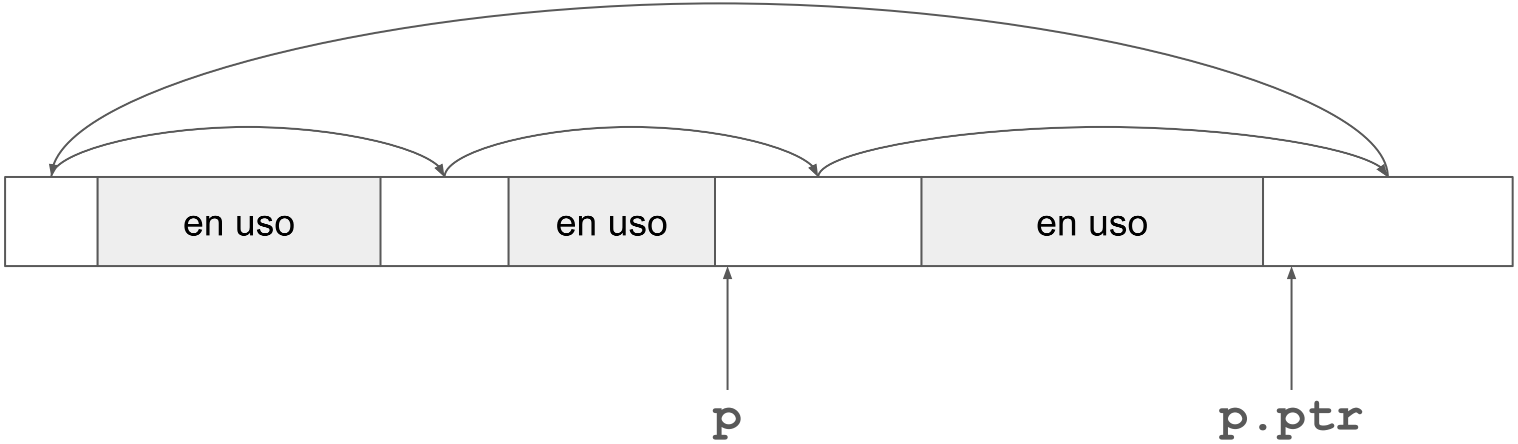


Liberar bloque: Coalesce



```
if (p + p->size == bp) {
   p->size += bp->size
}
```

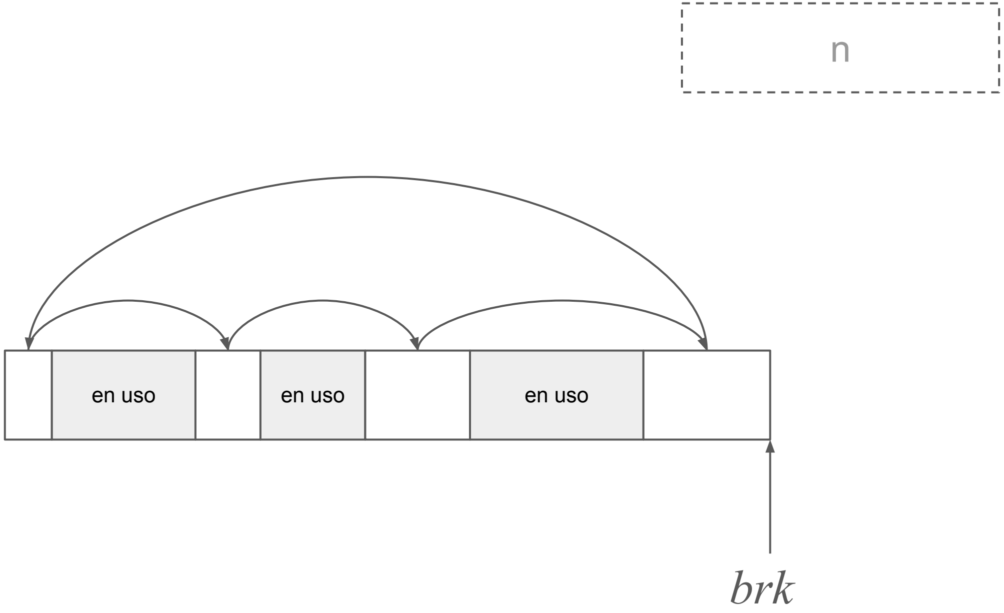
Liberar bloque: Coalesce



```
if (p + p->size == bp) {
   p->size += bp->size
}
```

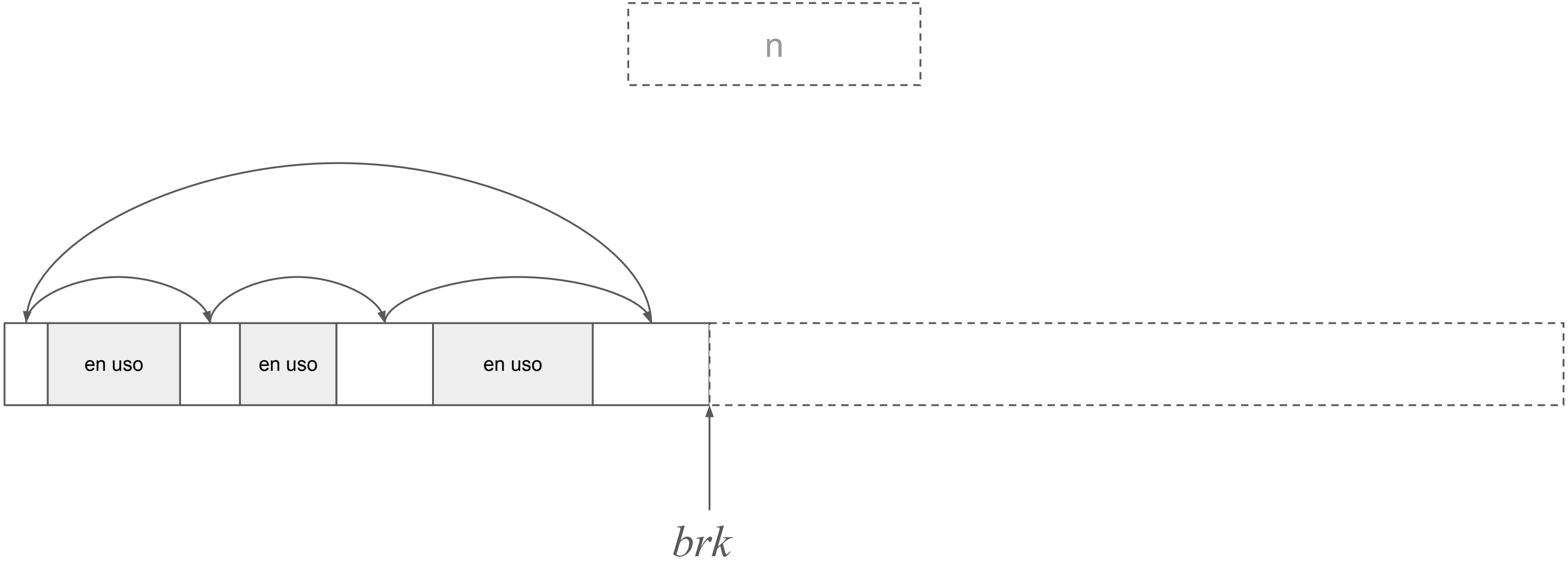
Reservar memoria pidiendo mas al OS

```
ap = malloc(n)
```



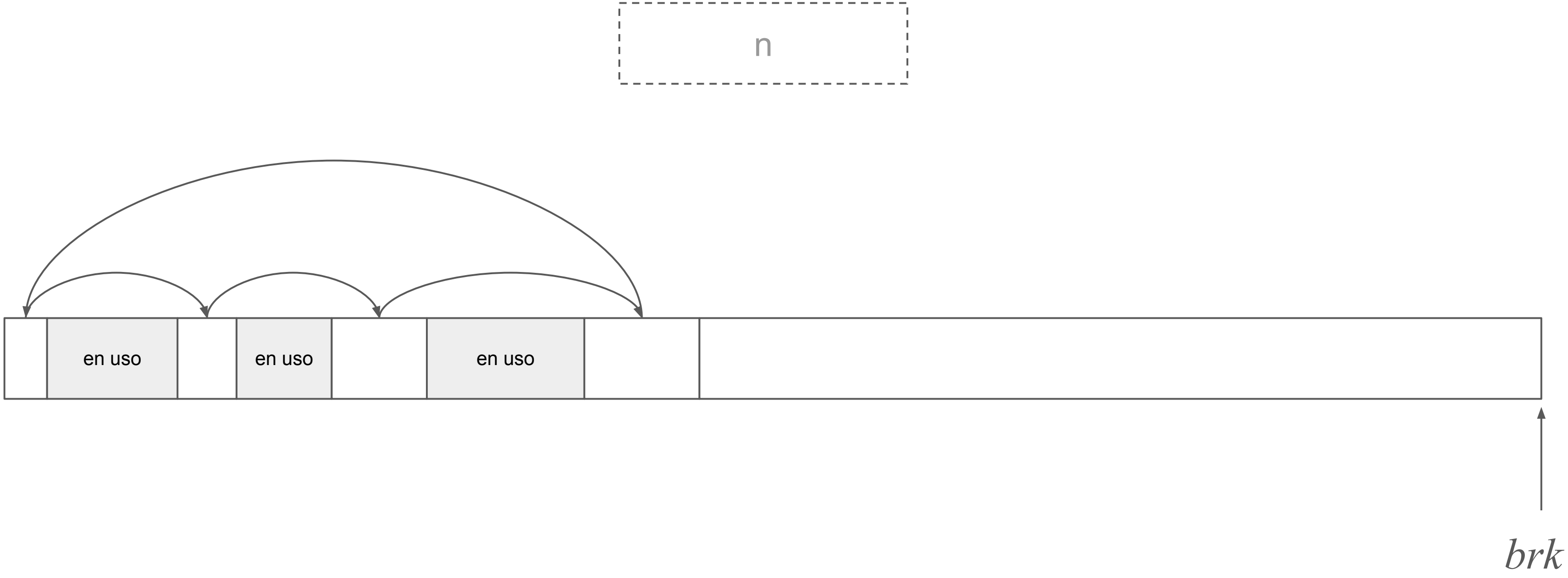
Reservar memoria pidiendo mas al OS

```
ap = malloc(n)
```



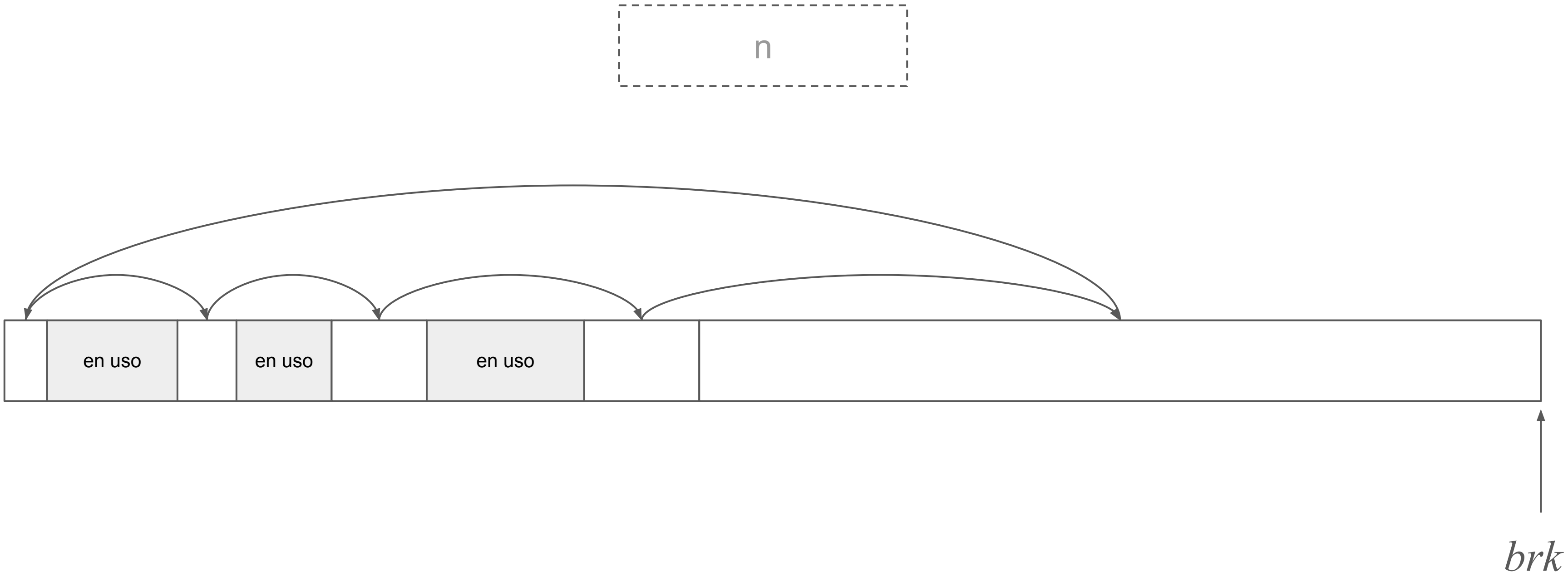
Reservar memoria pidiendo mas al OS

```
ap = malloc(n)
```



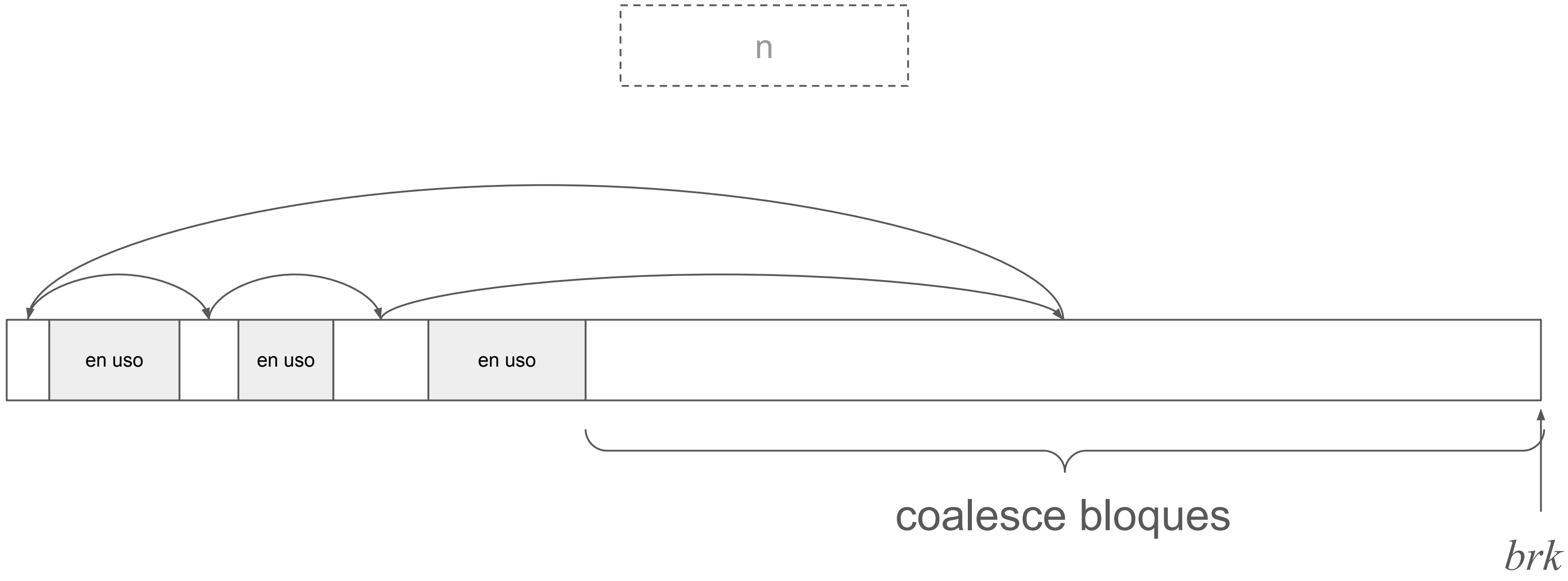
Reservar memoria pidiendo mas al OS

```
ap = malloc(n)
```



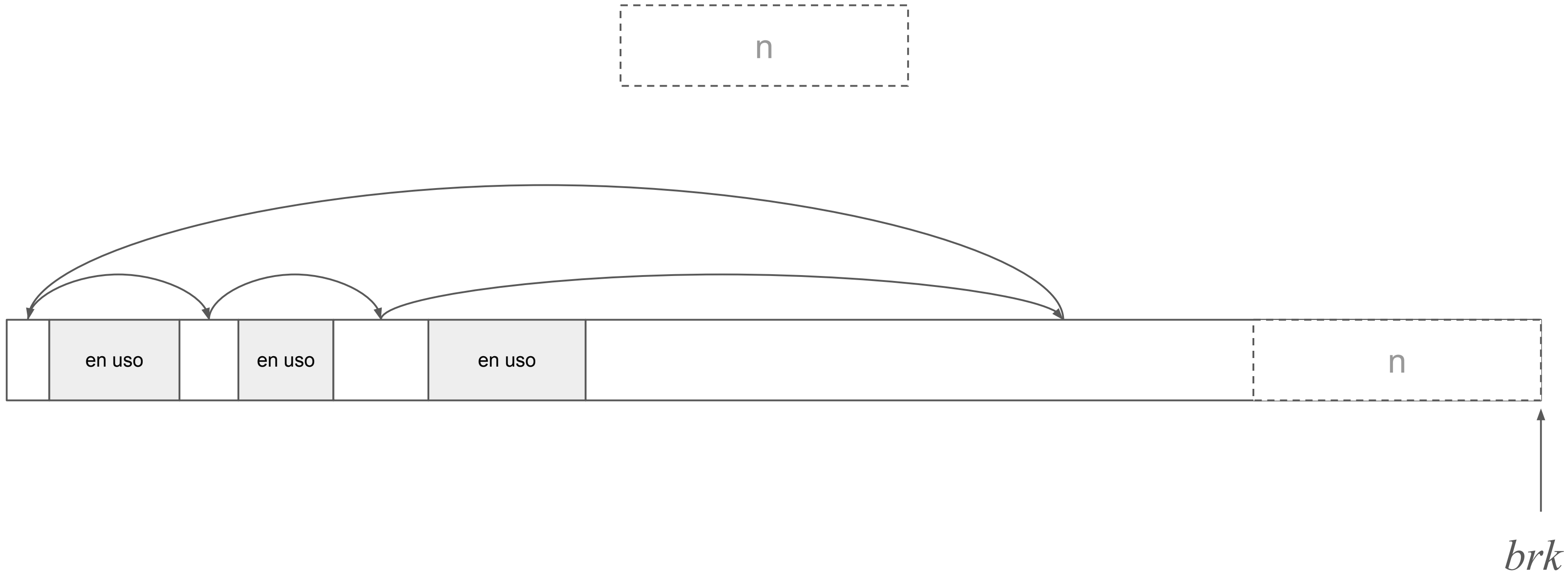
Reservar memoria pidiendo mas al OS

```
ap = malloc(n)
```



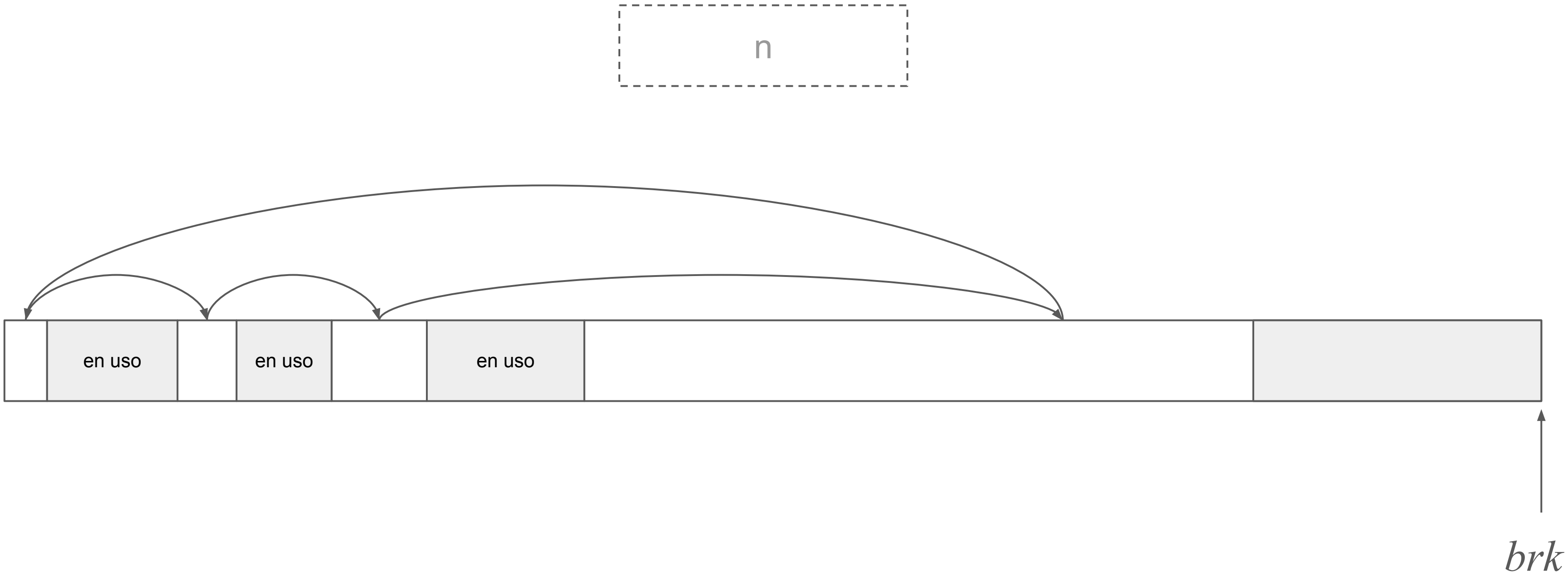
Reservar memoria pidiendo mas al OS

```
ap = malloc(n)
```



Reservar memoria pidiendo mas al OS

```
ap = malloc(n)
```



mmap

mmap

```
#include <sys/mman.h>

void *mmap(void *addr, size_t length, int prot, int flags,
           int fd, off_t offset);
int munmap(void *addr, size_t length);
```

https://dashdash.io/2/mmap

Reservar un bloque con mmap

```
// mmap() returns a pointer to a chunk of free space

node_t *head = mmap(NULL, 4096, PROT_READ|PROT_WRITE,
MAP_ANONYMOUS|MAP_PRIVATE, -1, 0);

head->size = 4096 - sizeof(node_t);

head->next = NULL;
```

MAP_ANONYMOUS

The mapping is not backed by any file; its contents are initialized to zero. The *fd* argument is ignored [...] The *offset* argument should be zero.

Administración de memoria en xv6

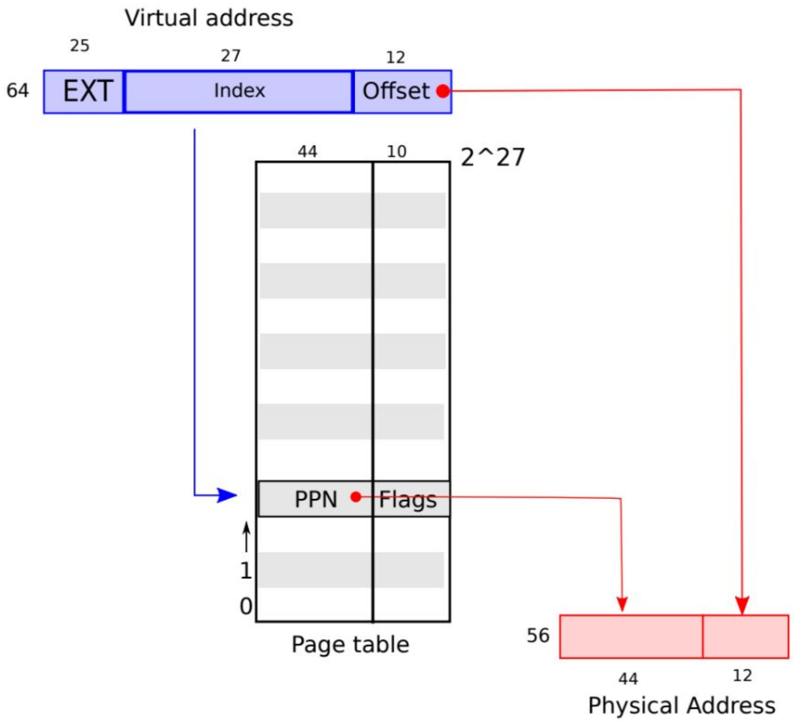


Figure 3.1: RISC-V virtual and physical addresses, with a simplified logical page table.

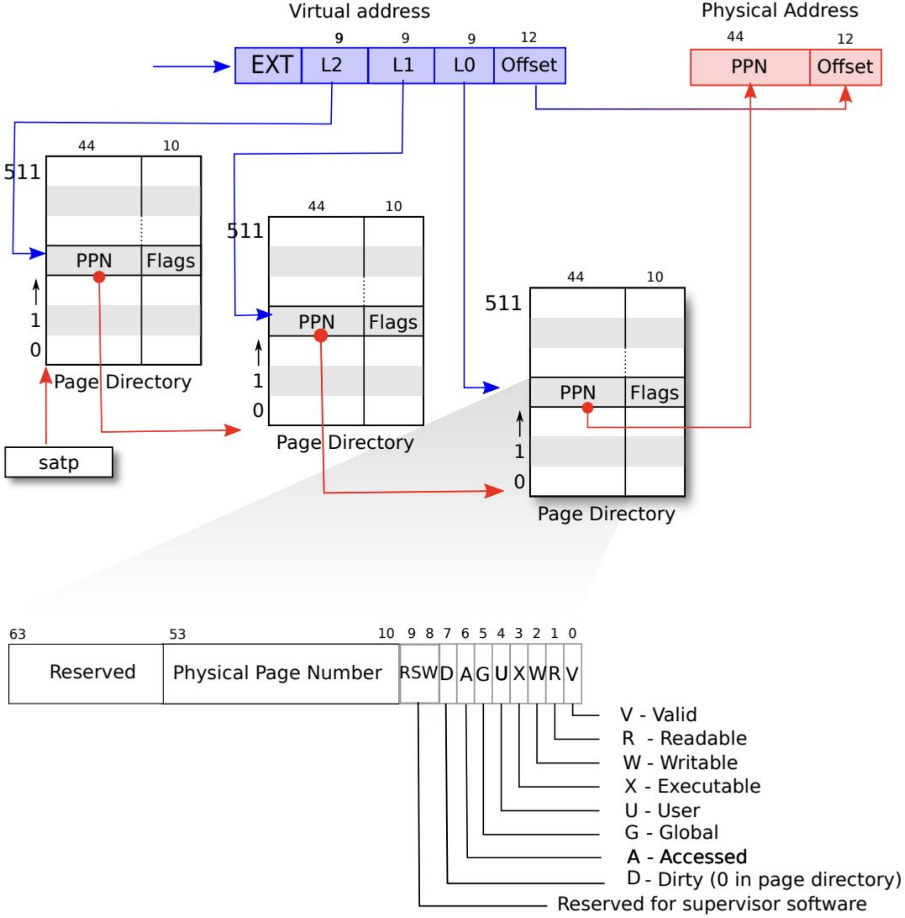


Figure 3.2: RISC-V address translation details.

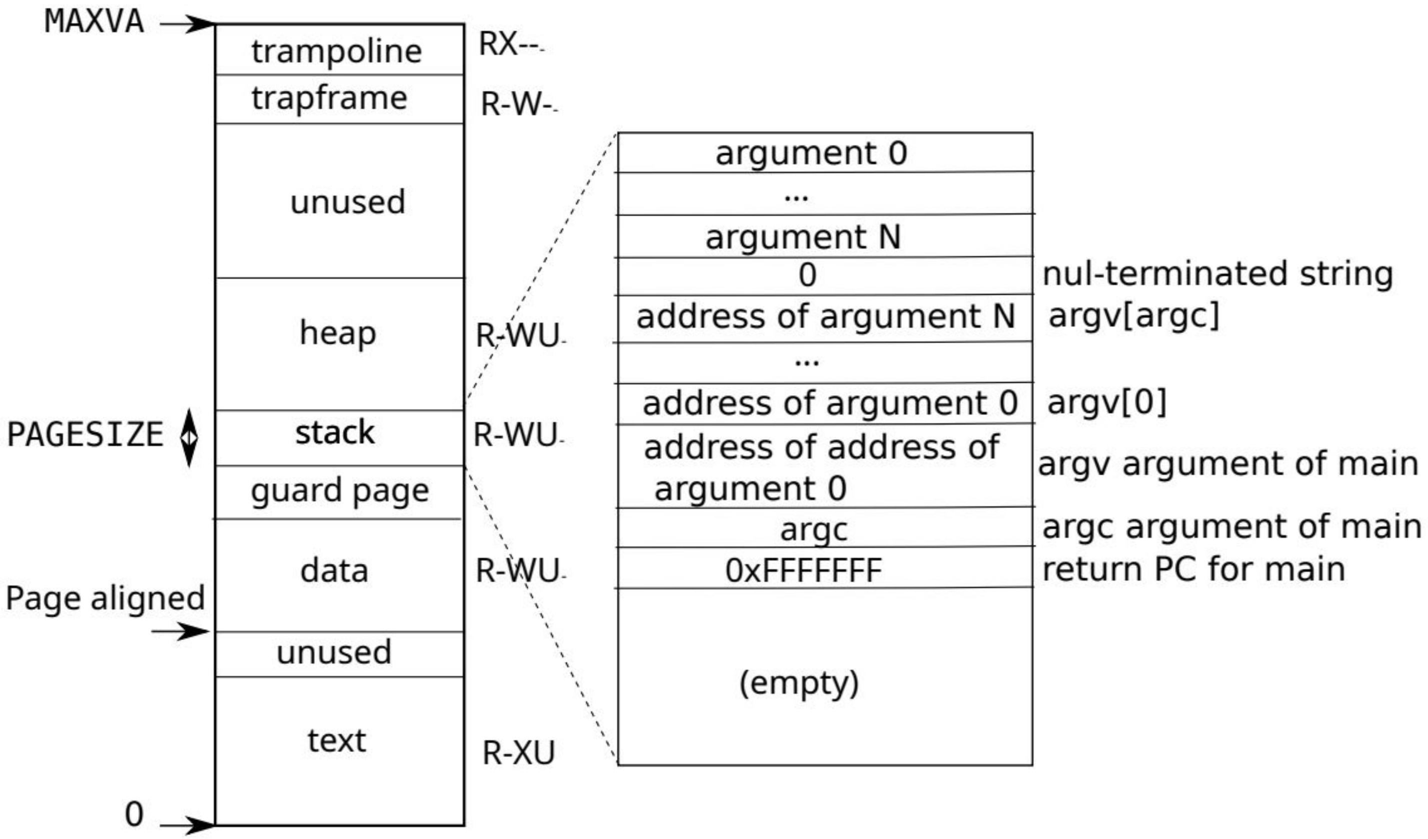


Figure 3.4: A process's user address space, with its initial stack.

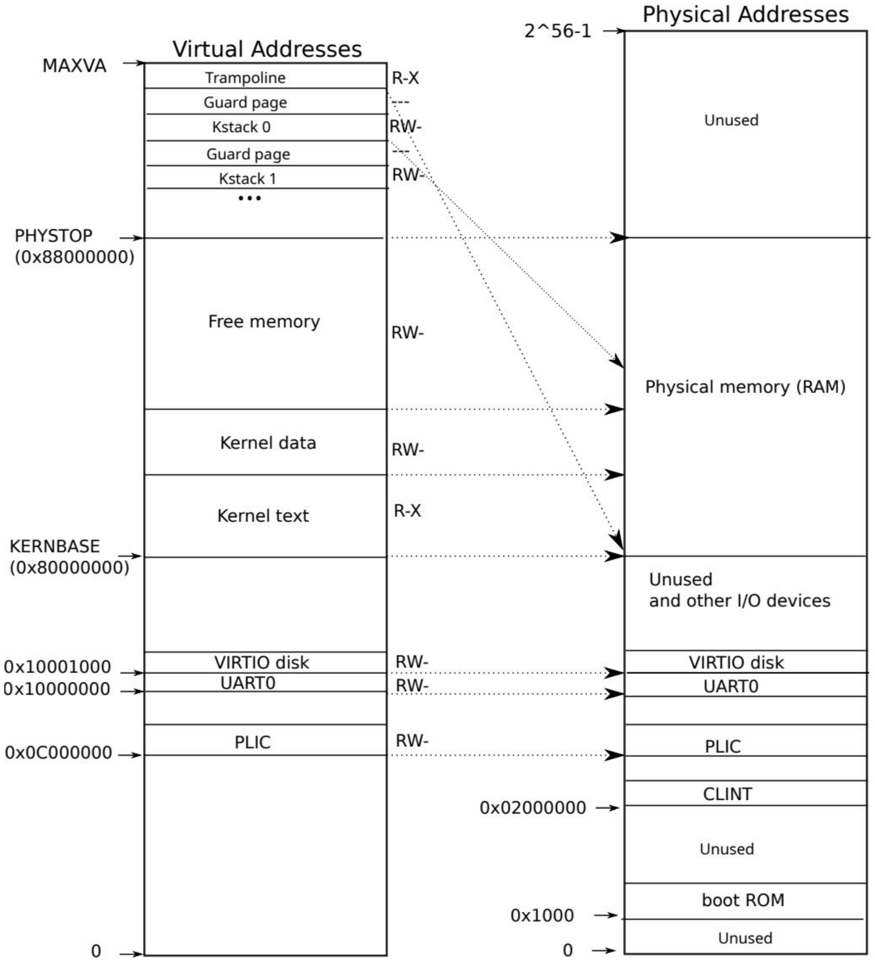


Figure 3.3: On the left, xv6's kernel address space. RWX refer to PTE read, write, and execute permissions. On the right, the RISC-V physical address space that xv6 expects to see.