# Προγραμματισμός II (ECE116)

#1

## δέσμευση/αποδέσμευση δυναμικής μνήμης

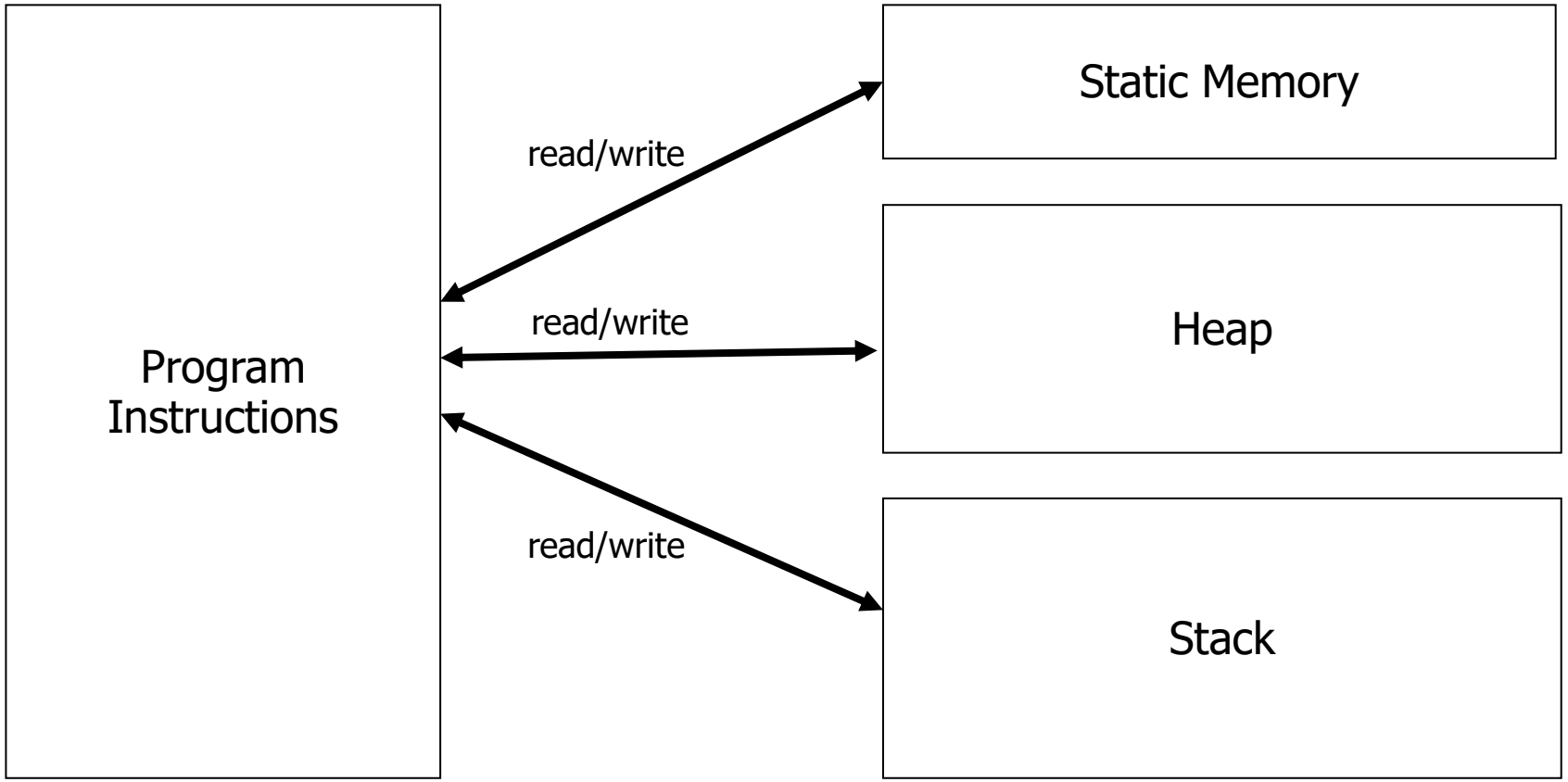

Σπύρος Λάλης
lalis@uth.gr

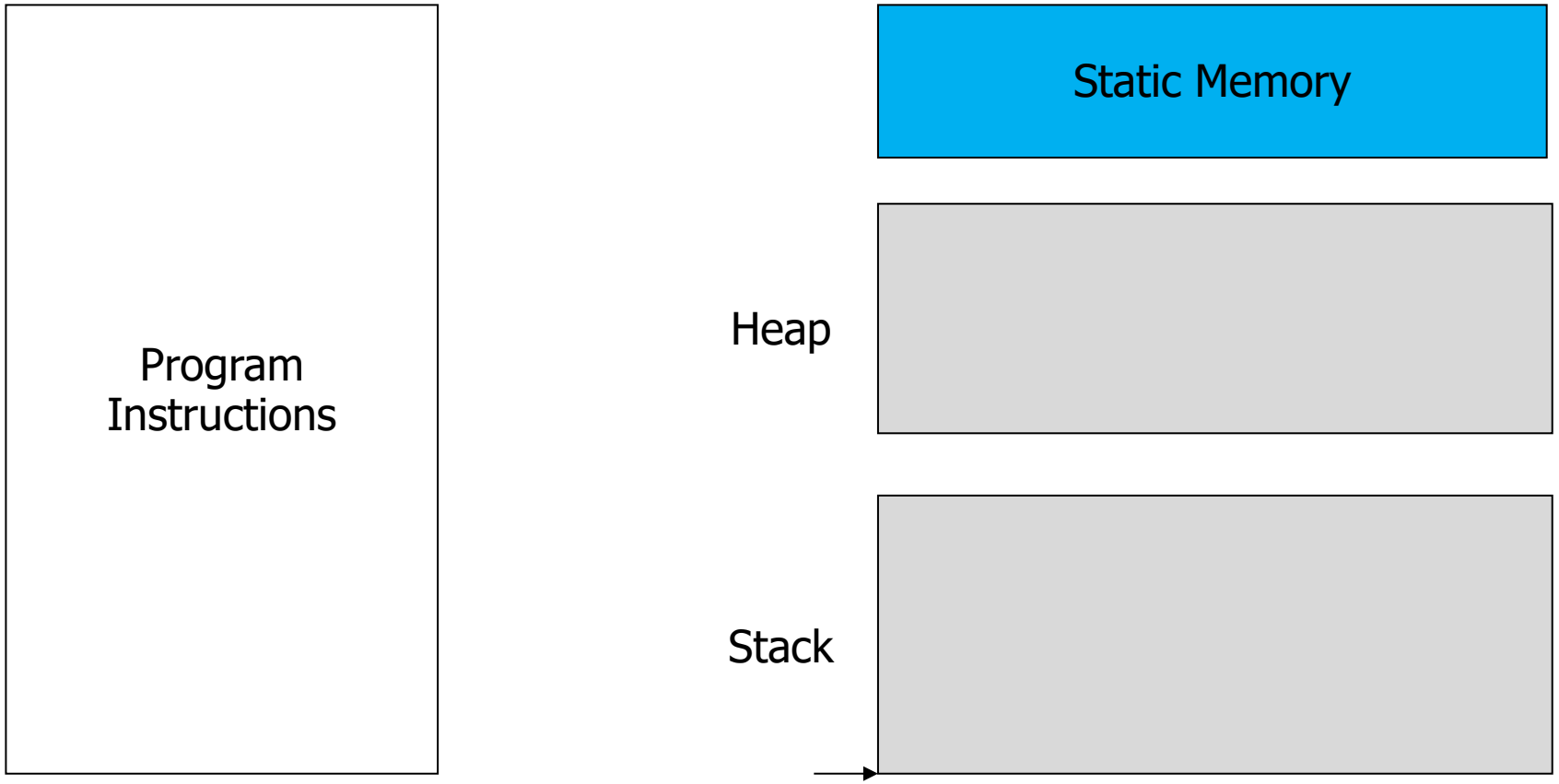
# Μνήμη προγράμματος

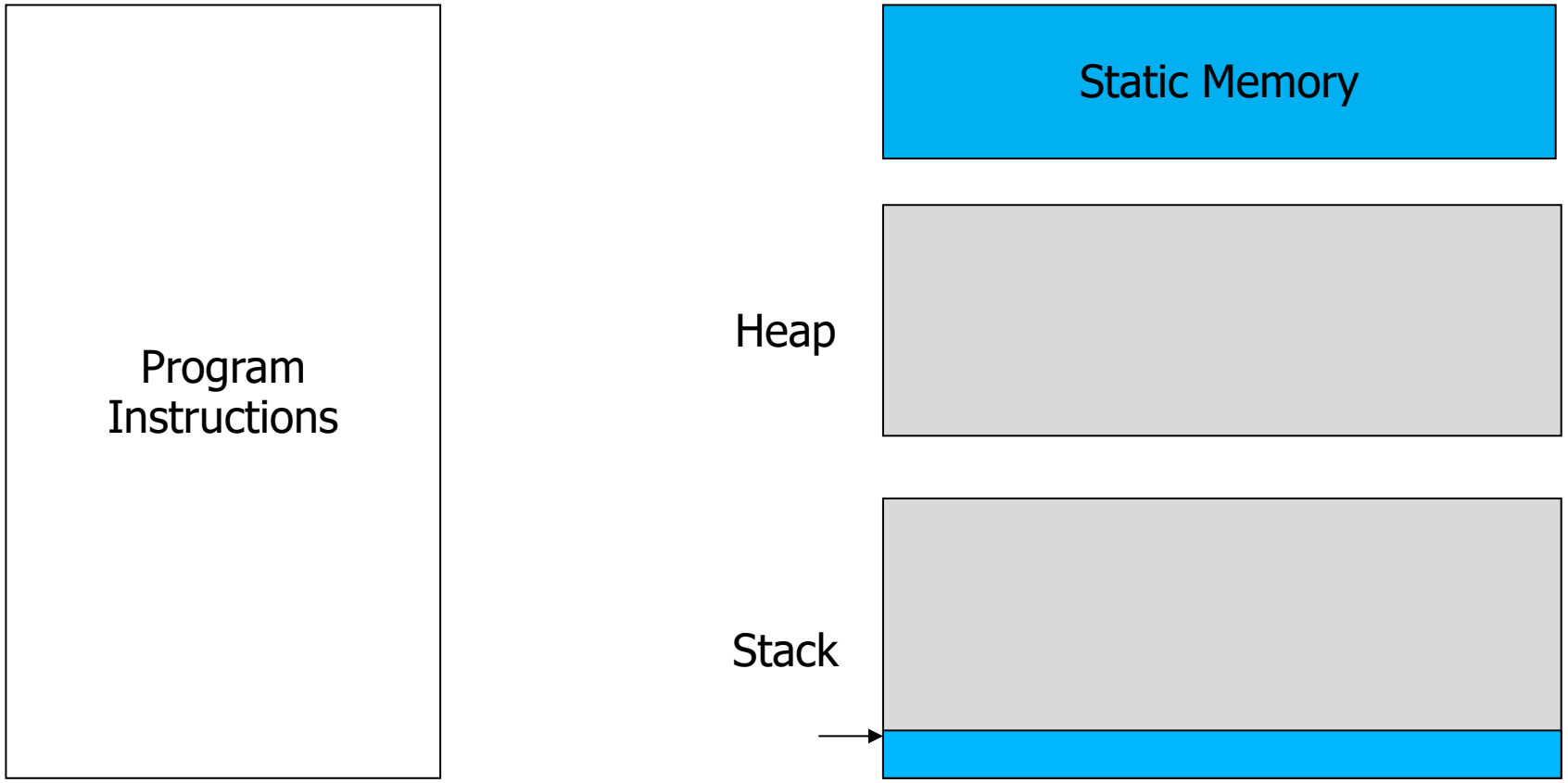
- Στατική μνήμη (static memory)
  - καθολικές & στατικές μεταβλητές (globals, static)
  - δεσμεύεται **αυτόματα/στατικά**, κατά την έναρξη της εκτέλεσης
- Στοίβα (stack)
  - μνήμη για παραμέτρους & τοπικές μεταβλητές συναρτήσεων
  - δεσμεύεται/αποδεσμεύεται **αυτόματα**
  - η μνήμη που δεσμεύεται για μια κλήση συνάρτησης έχει ισχύ **μόνο** κατά την διάρκεια της συγκεκριμένης κλήσης
- **Δυναμική μνήμη / σωρός (heap)**
  - δεσμεύεται/αποδεσμεύεται **ρητά** από τον προγραμματιστή
  - έχει ισχύ **καθ' όλη** την διάρκεια εκτέλεσης του προγράμματος
  - **ανεξάρτητα** από τις κλήσεις συνάρτησης

# Γιατί χρειάζεται η δυναμική μνήμη;

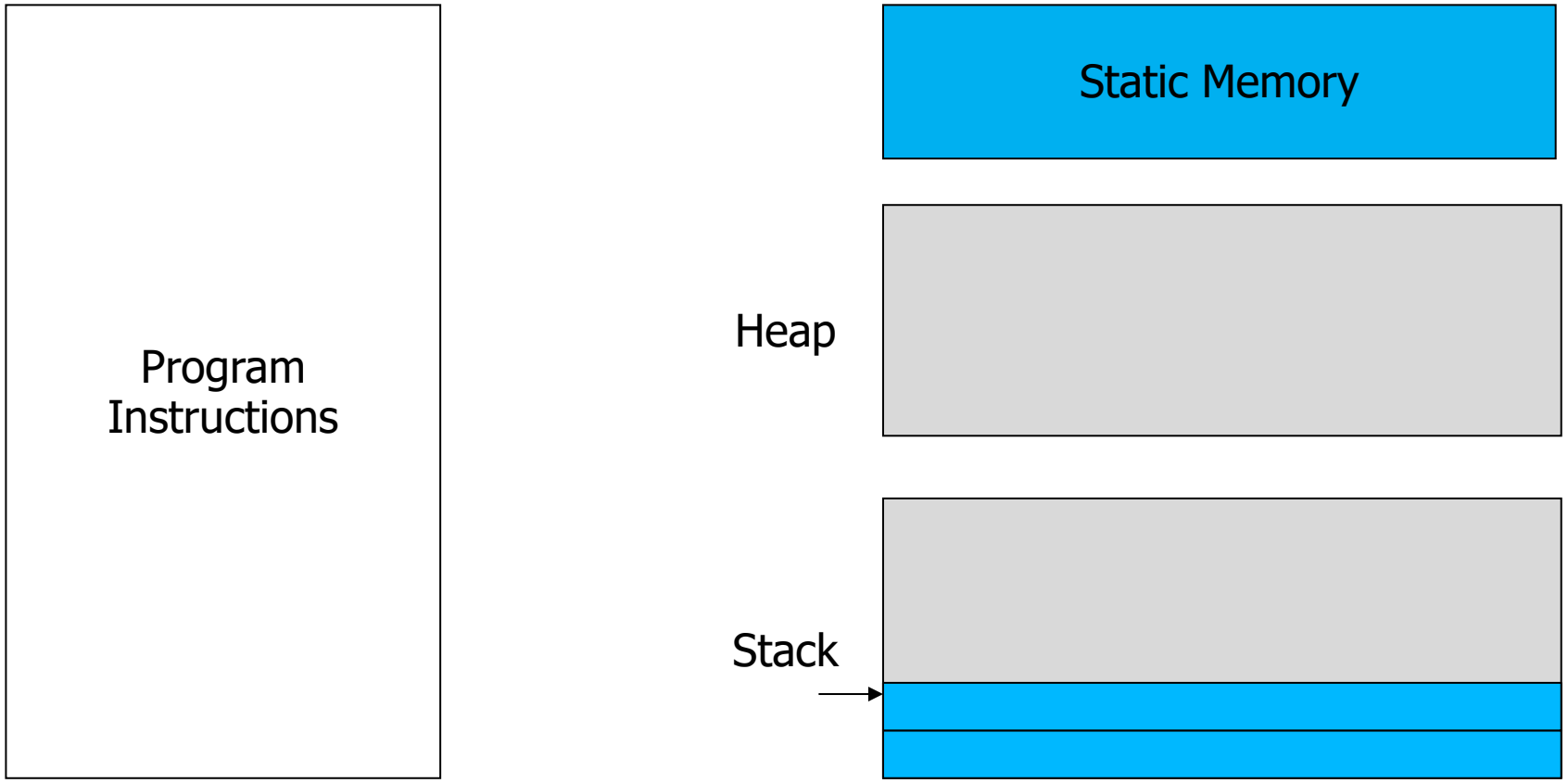
- Οι απαιτήσεις του προγράμματος σε μνήμη μπορεί να είναι **άγνωστες** την ώρα της συγγραφής του κώδικα
- Για «σιγουριά» μπορούμε να χρησιμοποιήσουμε έναν πολύ μεγάλο αριθμό μεταβλητών / πίνακα
  - όμως, αυτή η μνήμη ίσως να μην χρησιμοποιηθεί ποτέ
  - η αχρησιμοποίητη μνήμη πρακτικά **χάνεται** – δεν μπορεί να διατεθεί σε άλλα προγράμματα που εκτελούνται στον Η/Υ
- Για «οικονομία» μπορούμε να χρησιμοποιήσουμε έναν μικρό αριθμό μεταβλητών / πίνακα
  - όμως, αυτή η μνήμη ίσως να μην είναι αρκετή
  - το πρόγραμμα θα αναγκαστεί να **πετάξει/χάσει** δεδομένα ή απλά να τερματίσει
- **Λύση:** χρησιμοποιούμε δυναμική μνήμη!


Program Instructions
read/write
read/write
read/write
Static Memory
Heap
Stack

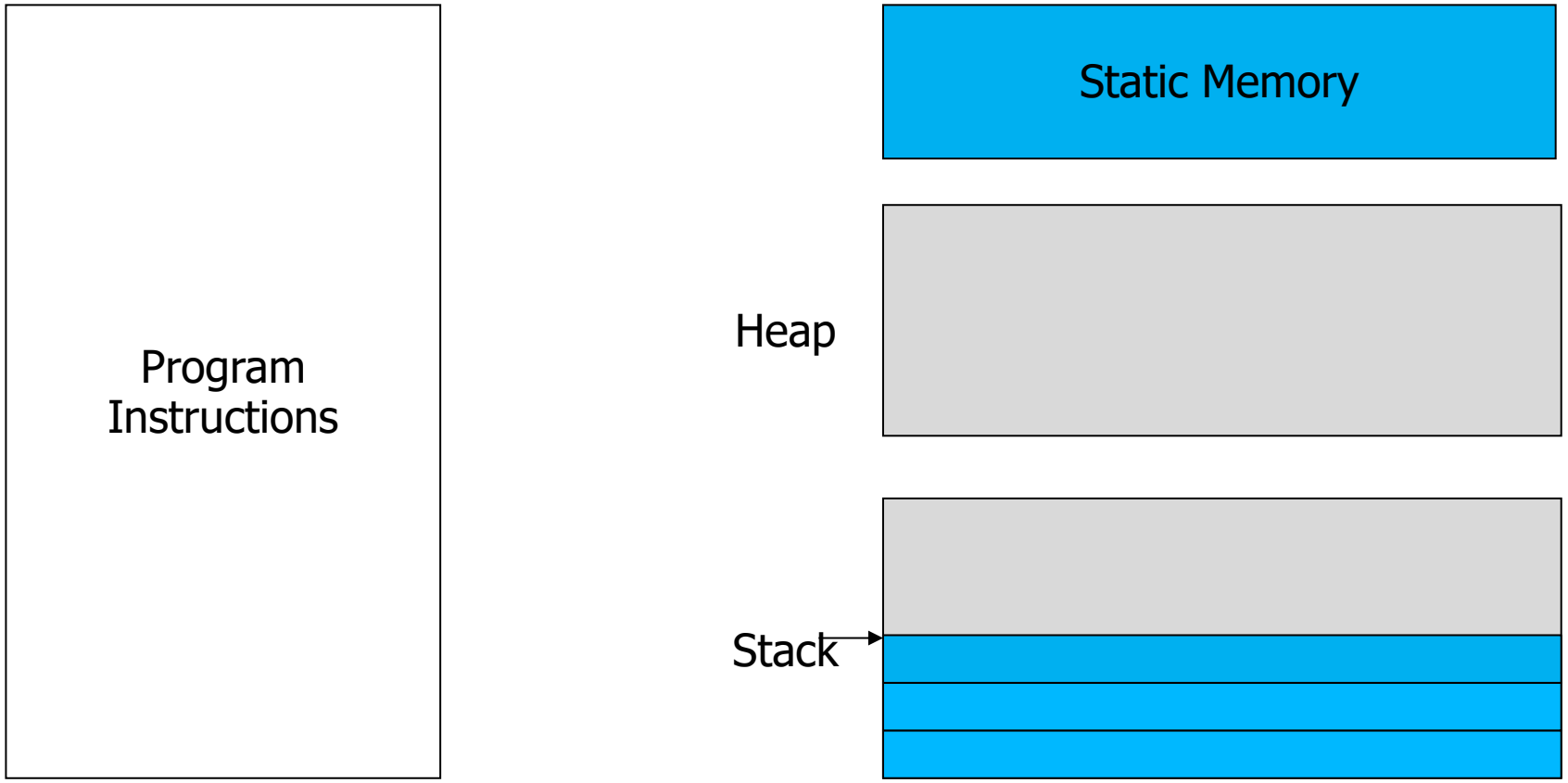

Program
Instructions
Static Memory
Heap
Stack



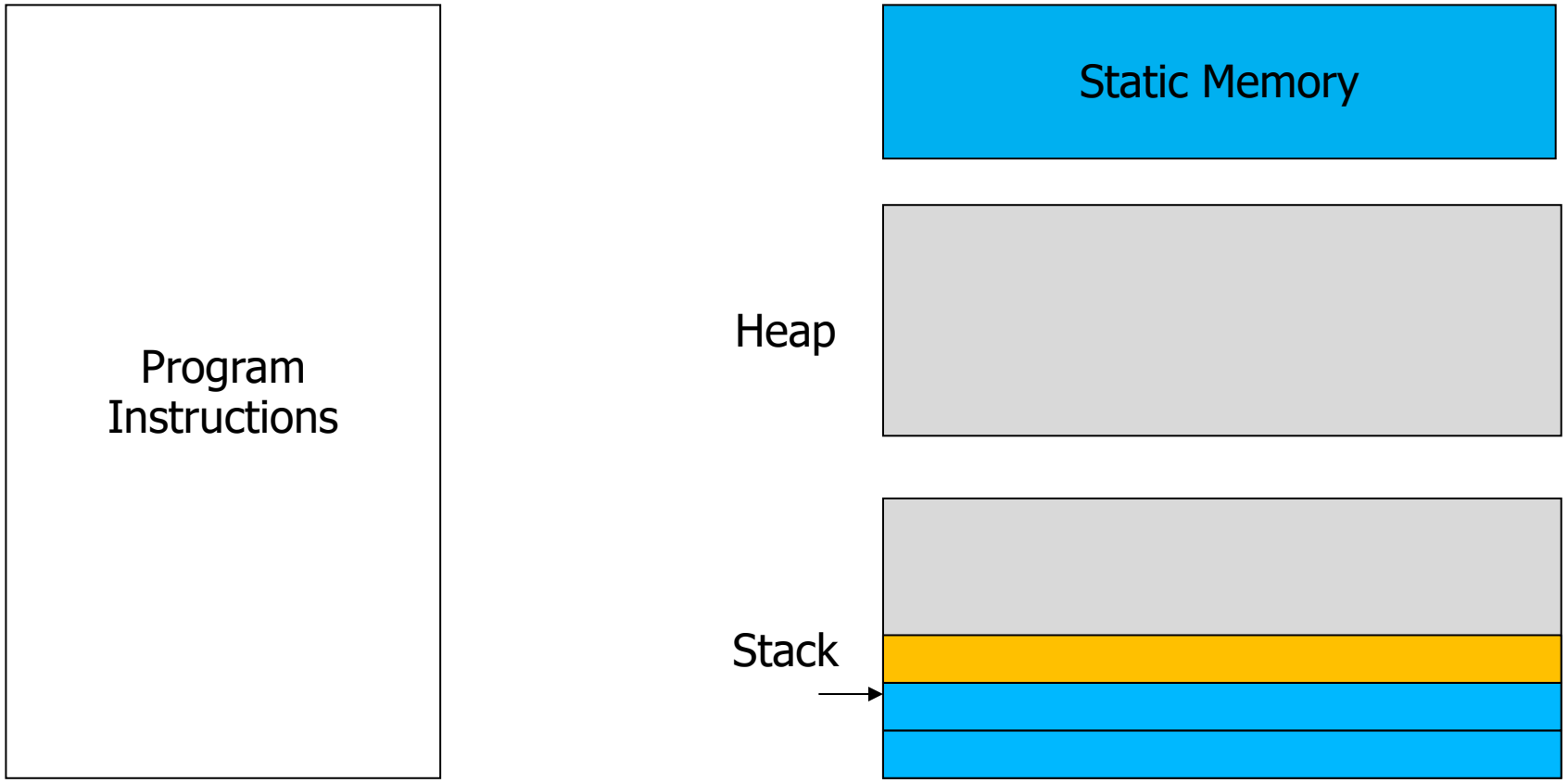

-> κλήση main



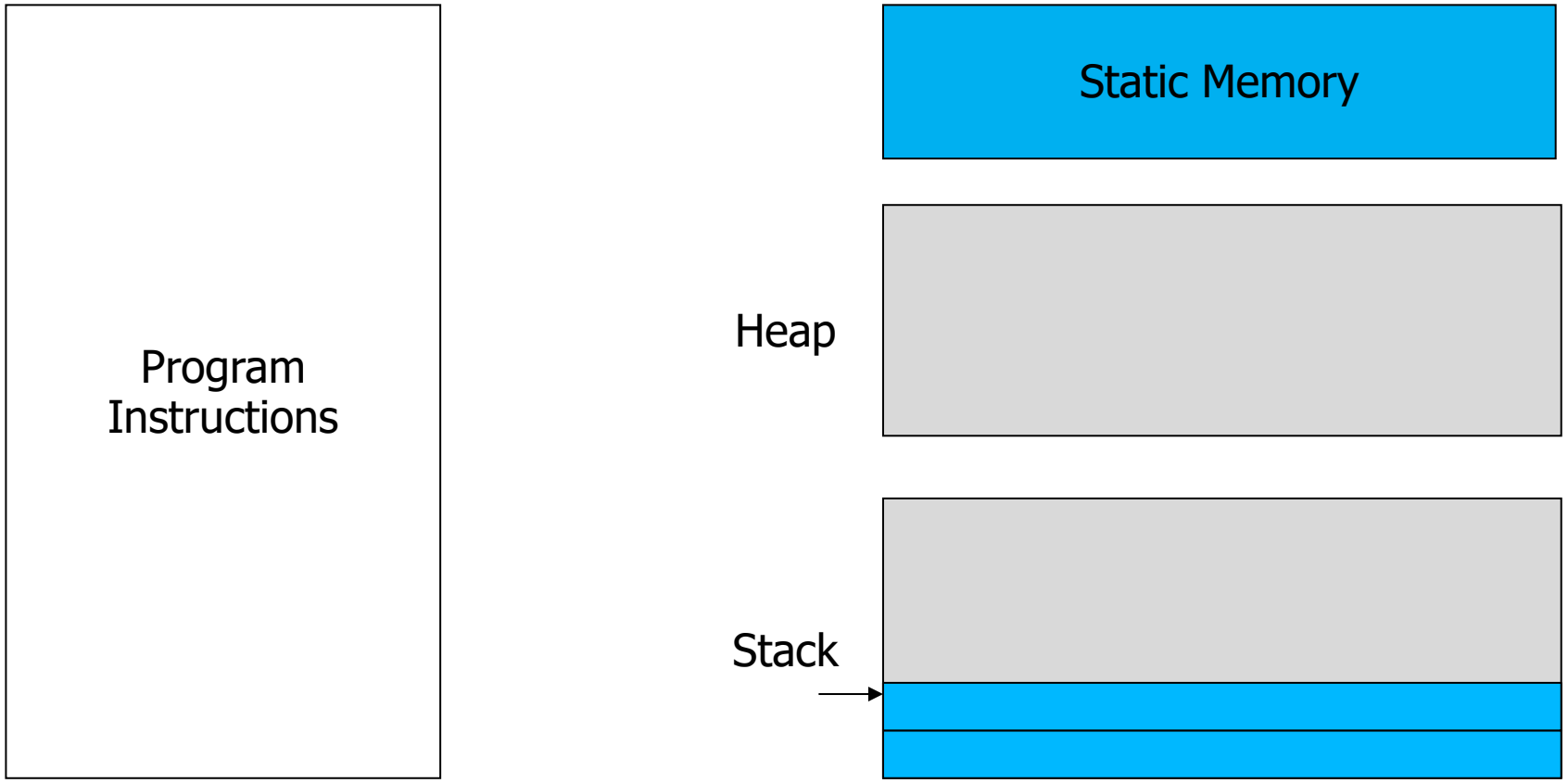

-> κλήση main -> κλήση f1



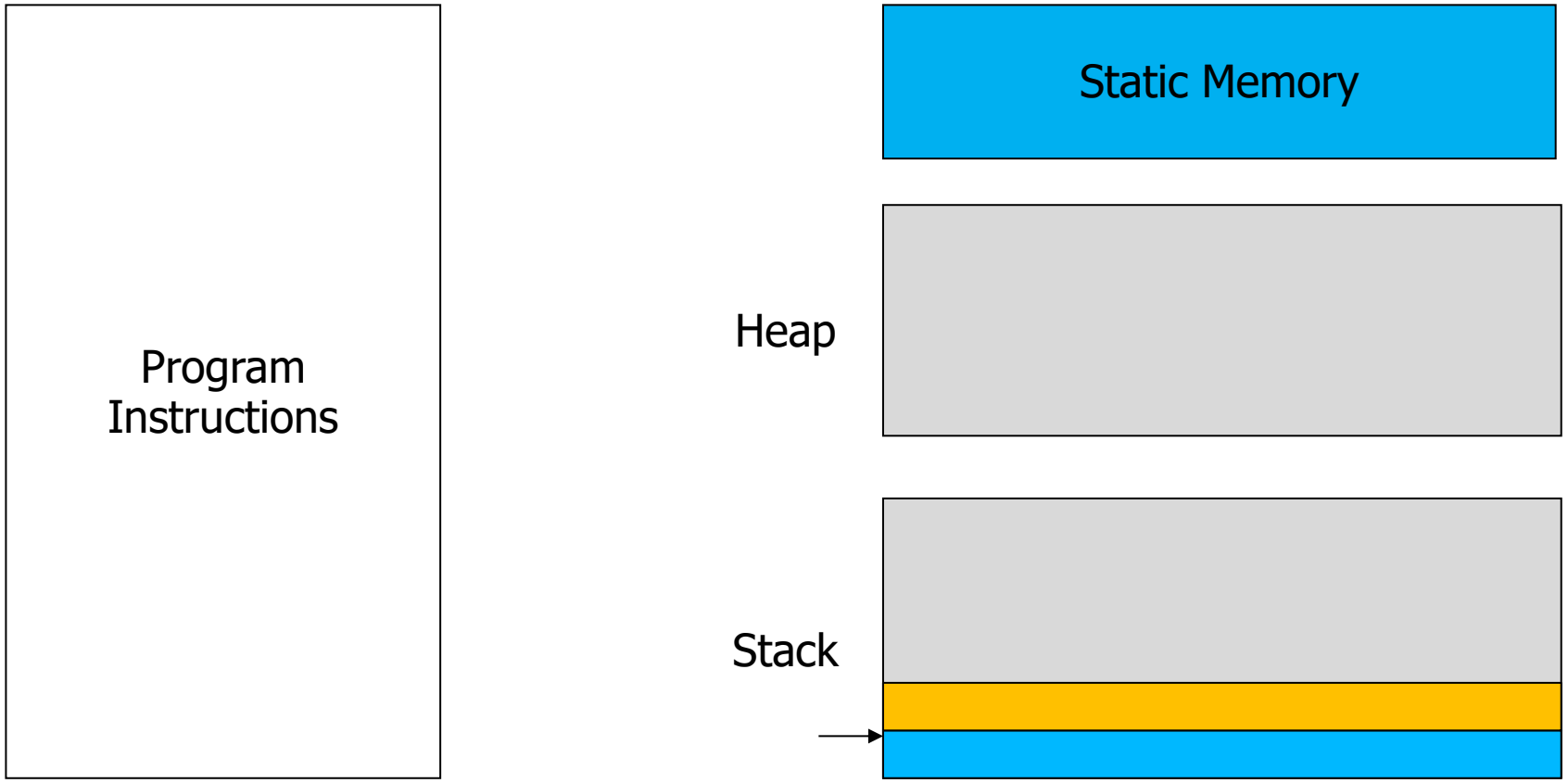

-> κλήση main -> κλήση f1 -> κλήση f2



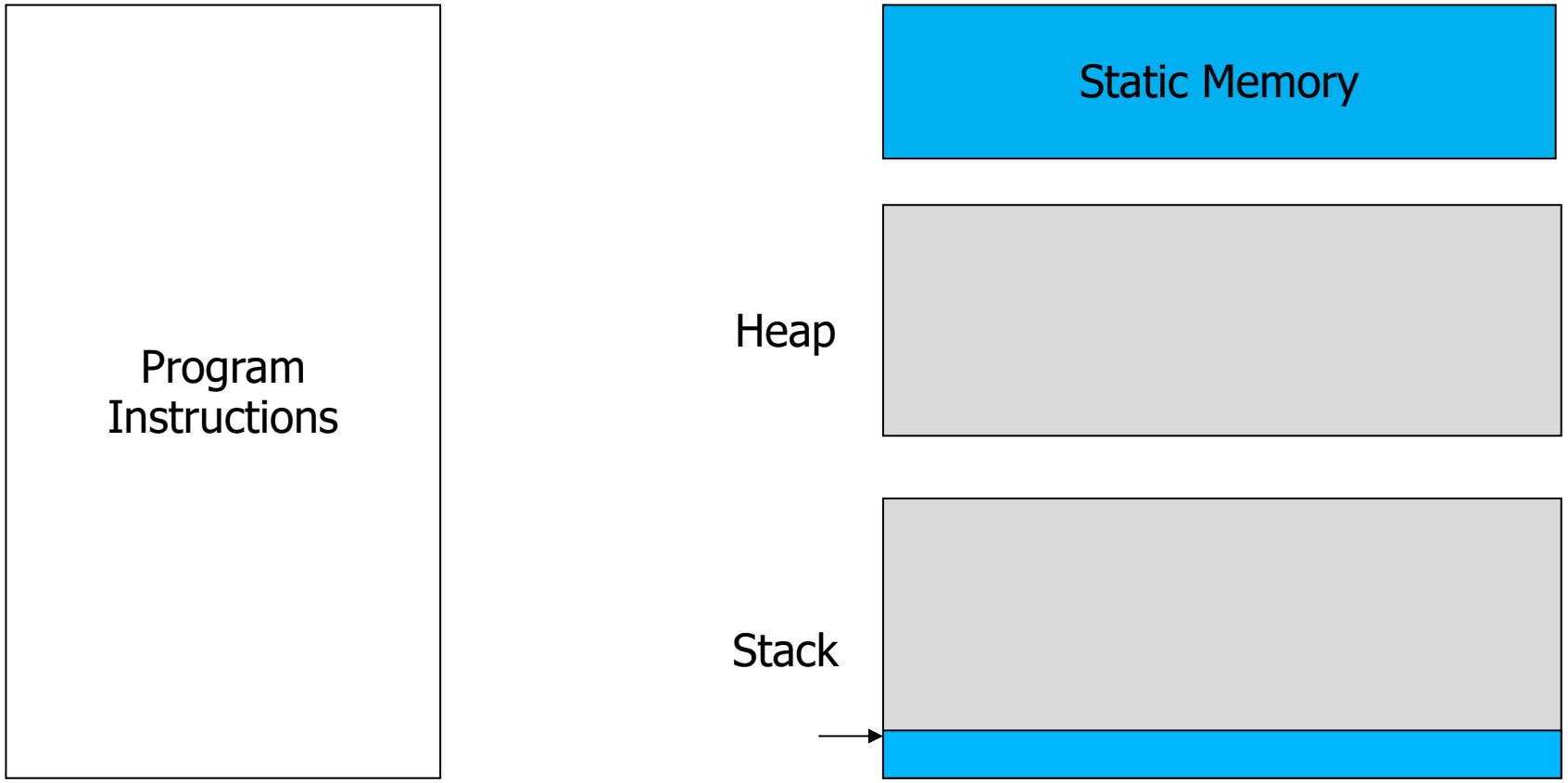

-> κλήση main -> κλήση f1 <- επιστροφή f2



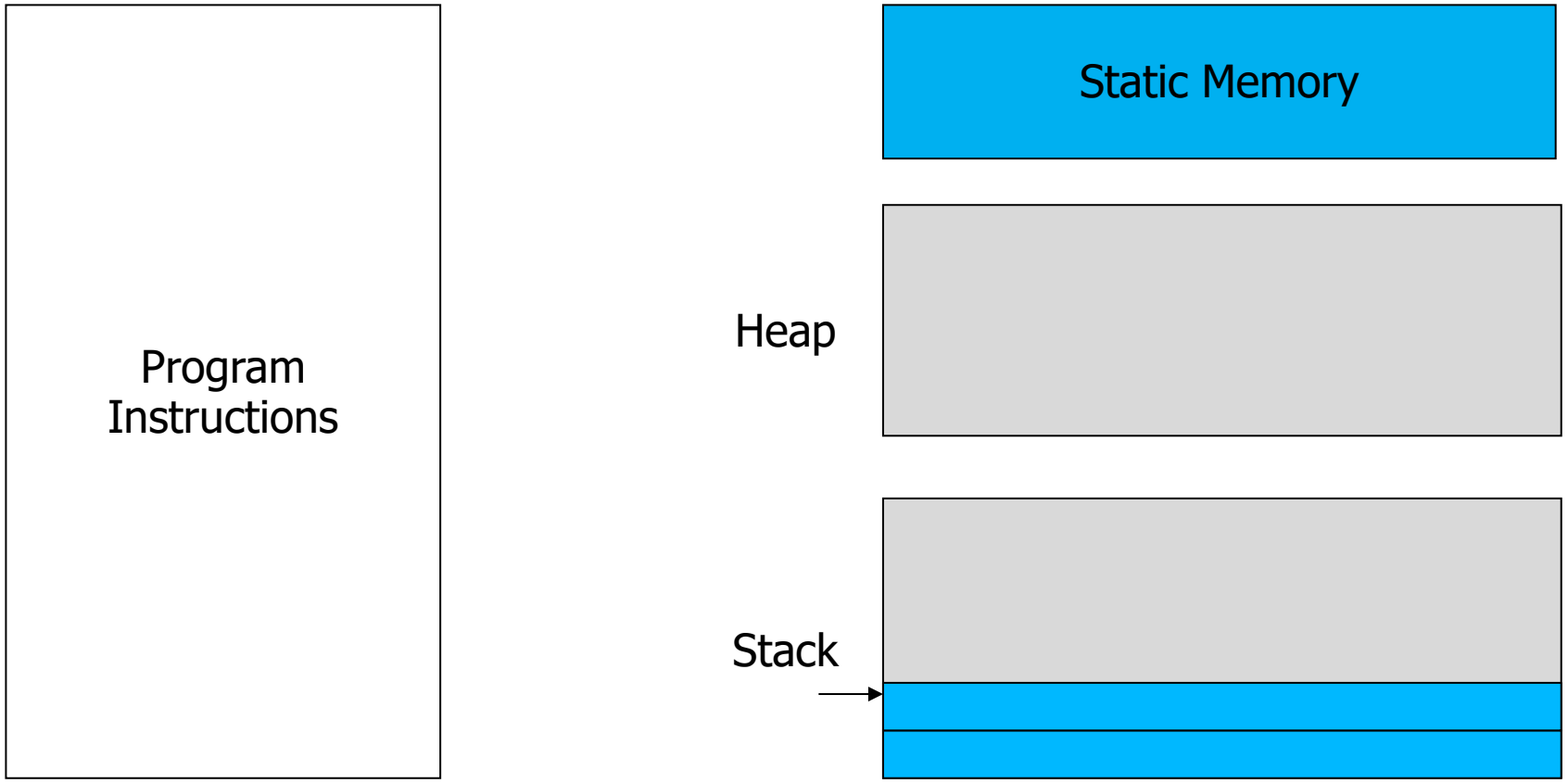

-> κλήση main -> κλήση f1



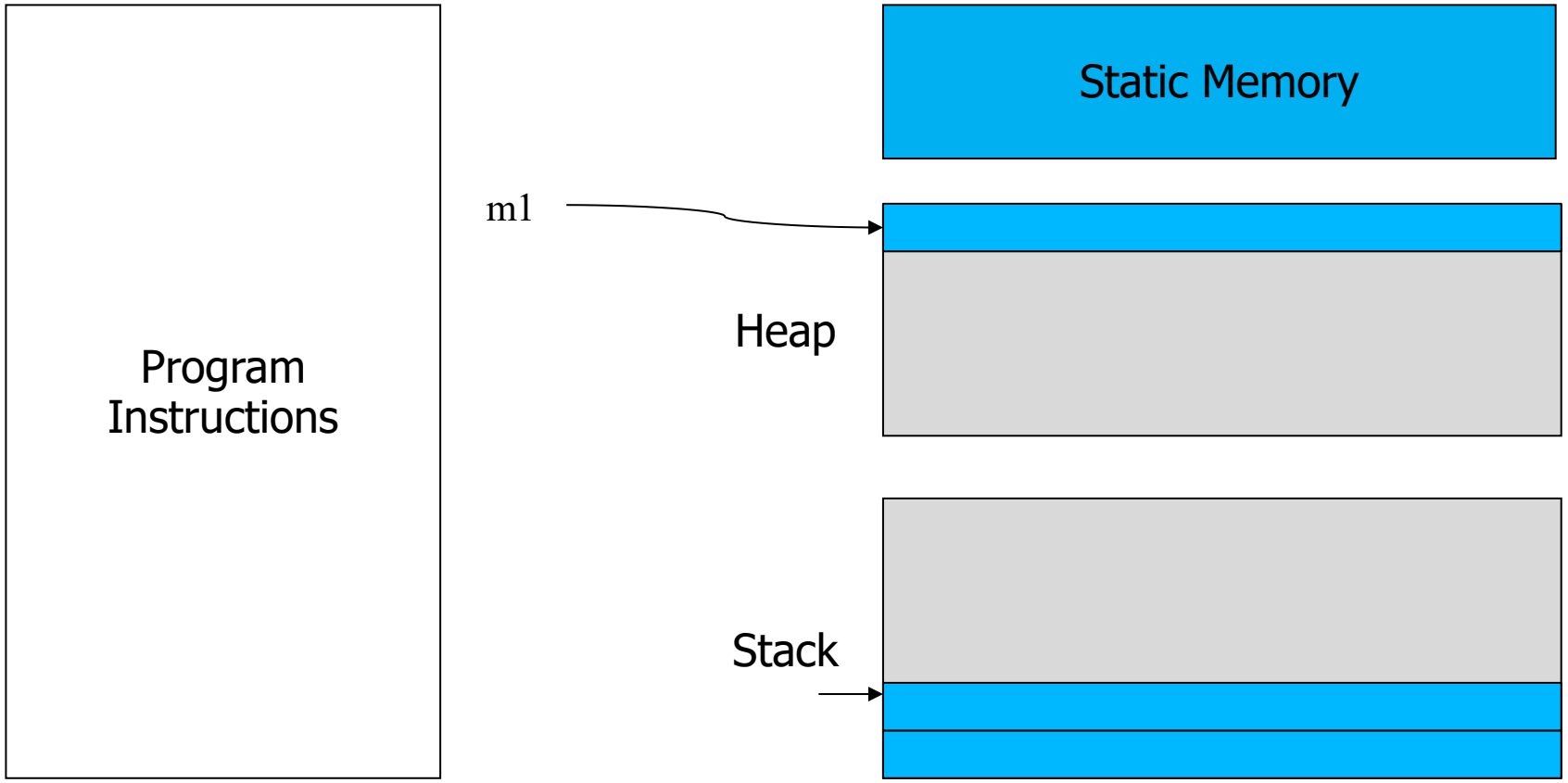

-> κλήση main <- επιστροφή f1




-> κλήση main

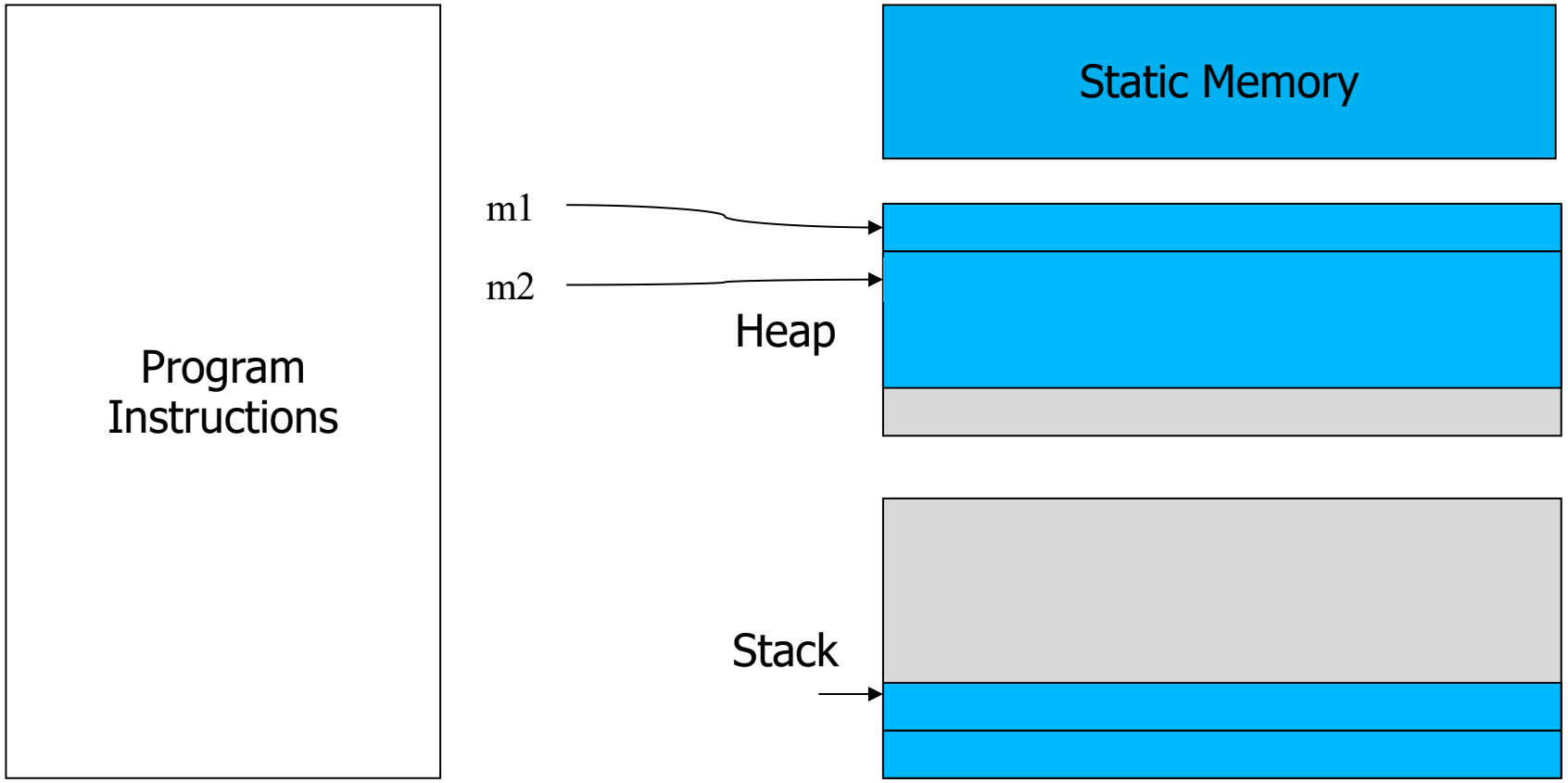



-> κλήση main -> κλήση f1



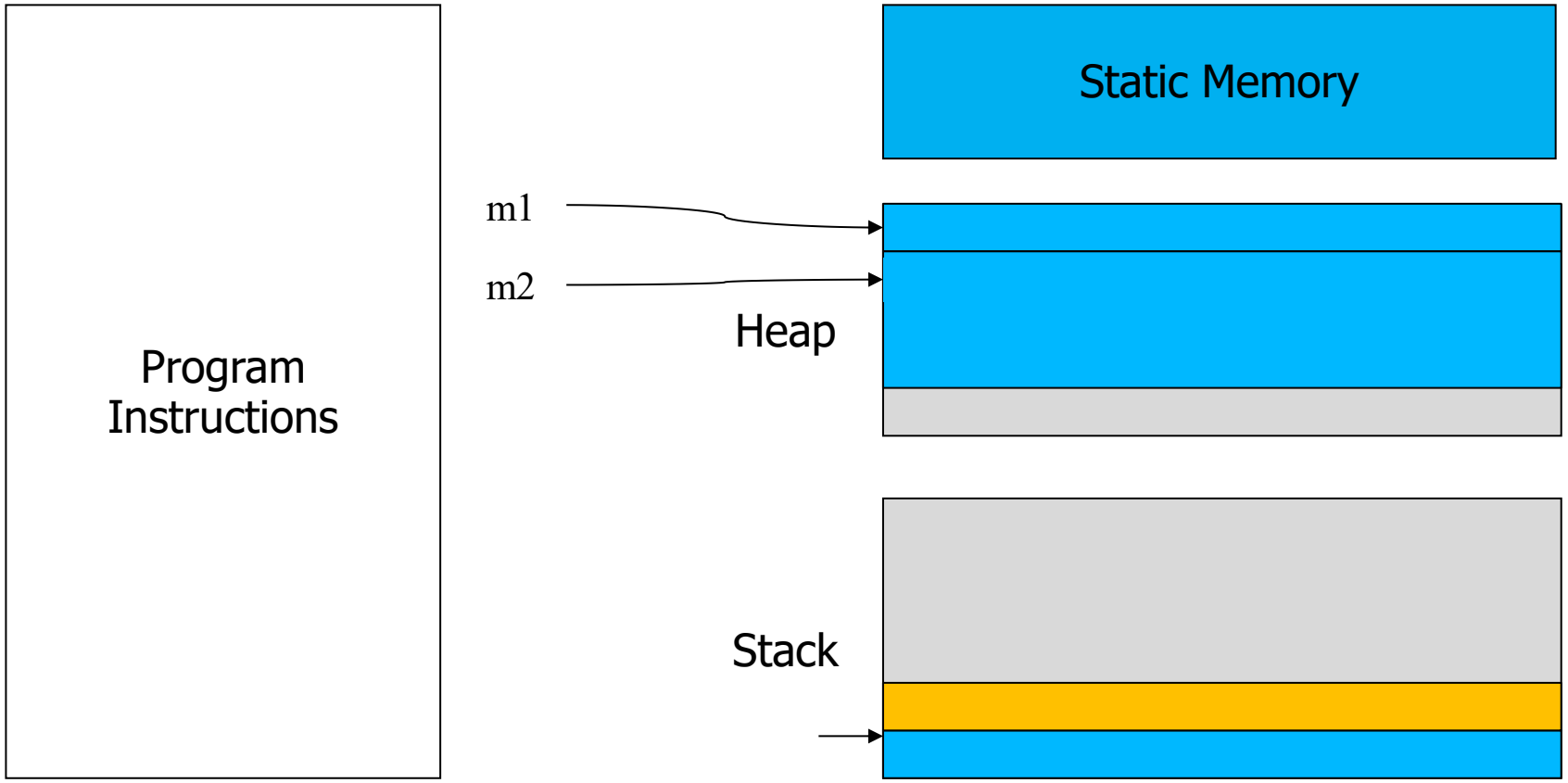

-> κλήση main -> κλήση f1

δέσμευση δυναμικής μνήμης m1

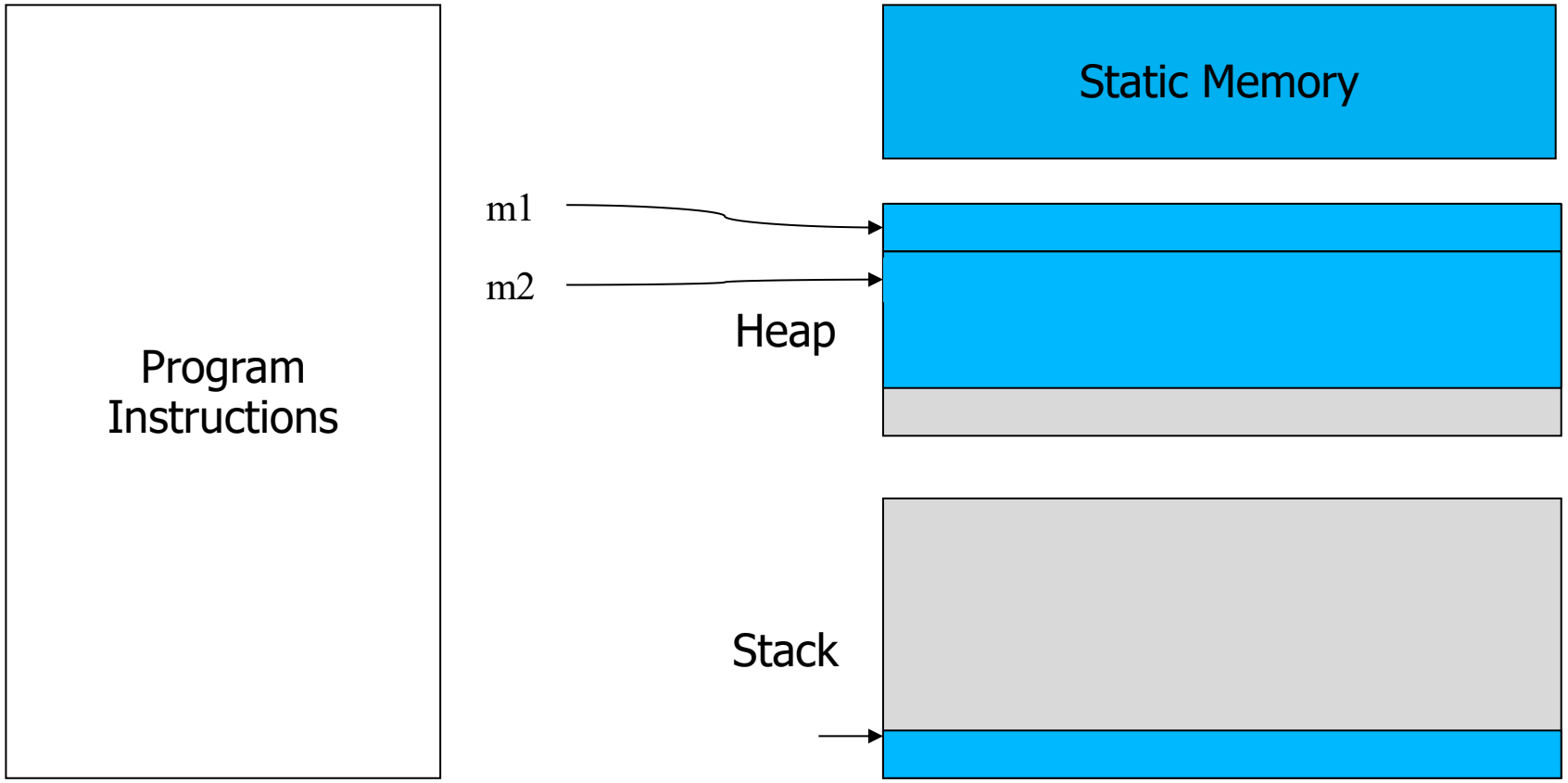



-> κλήση main -> κλήση f1

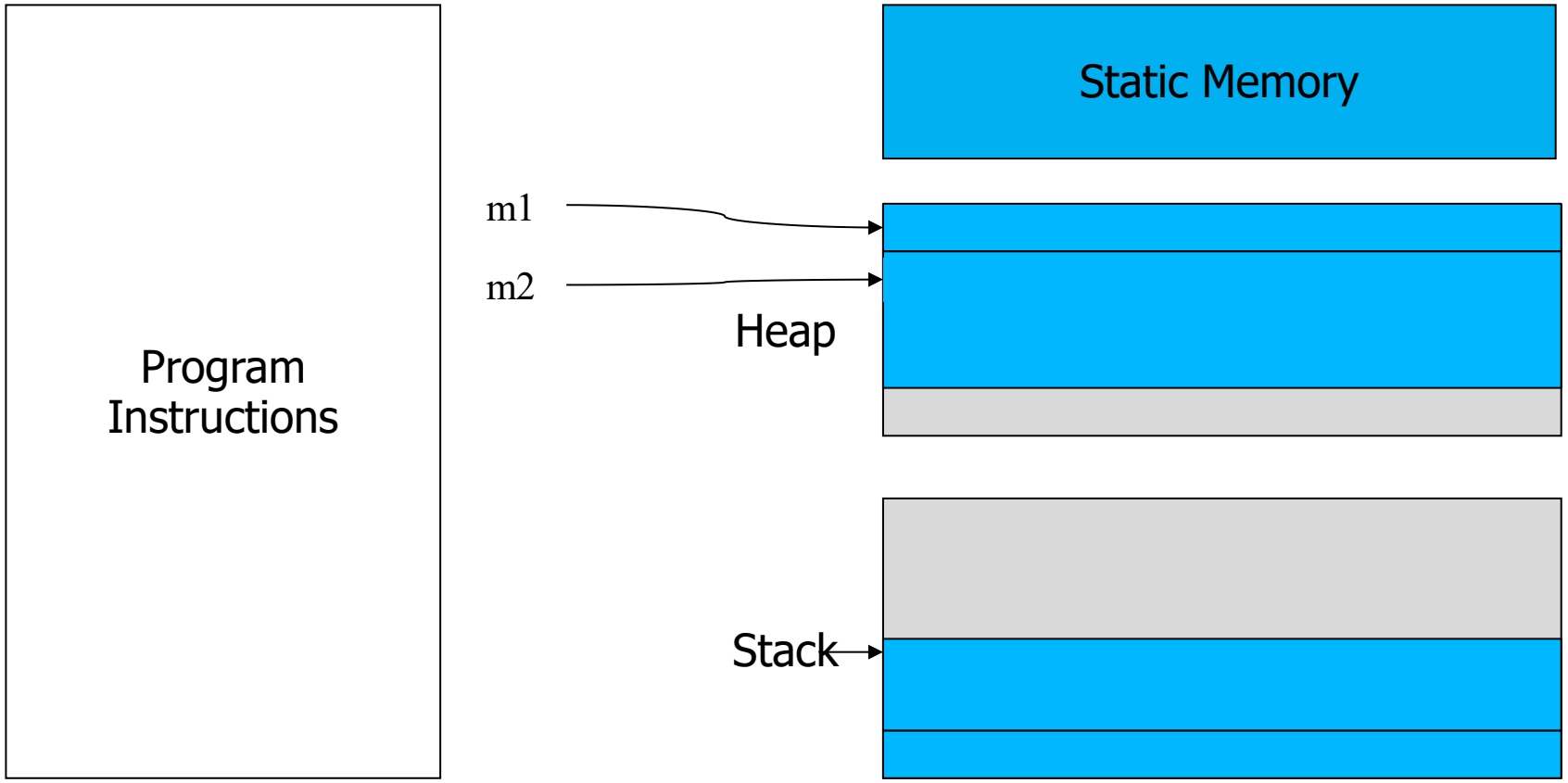
δέσμευση δυναμικής μνήμης m2



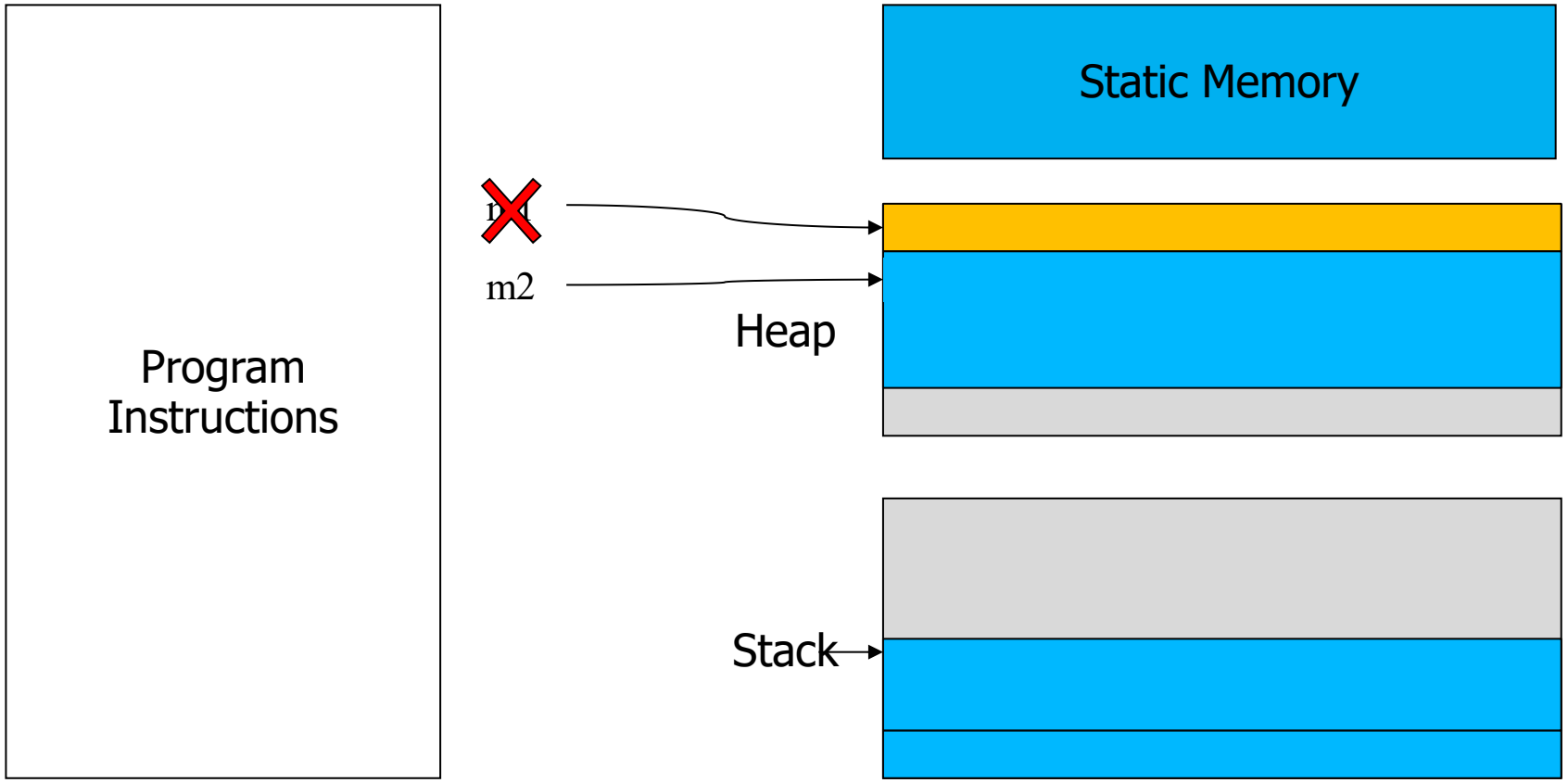

-> κλήση main <- επιστροφή f1




-> κλήση main

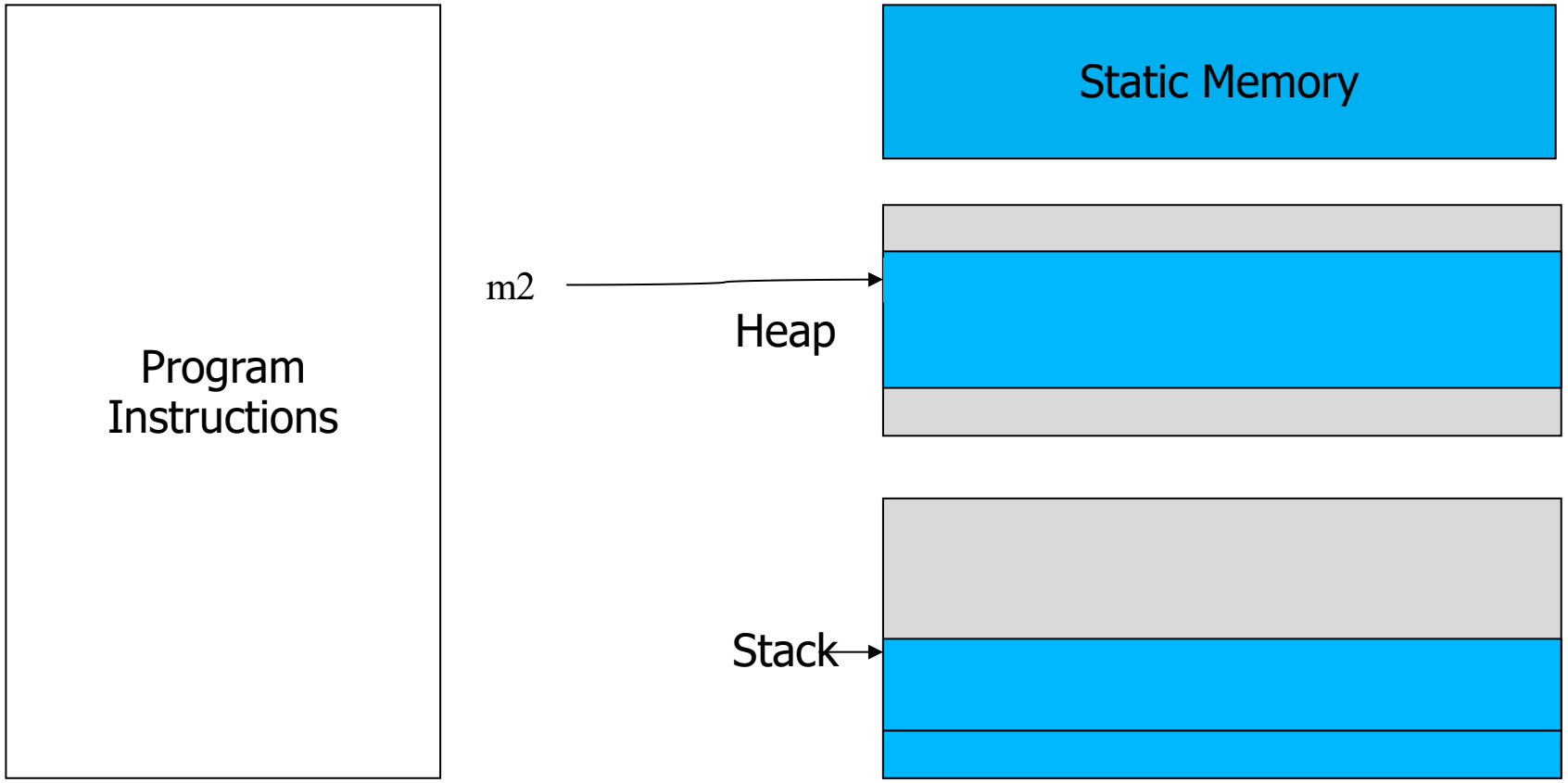



-> κλήση main -> κλήση f3

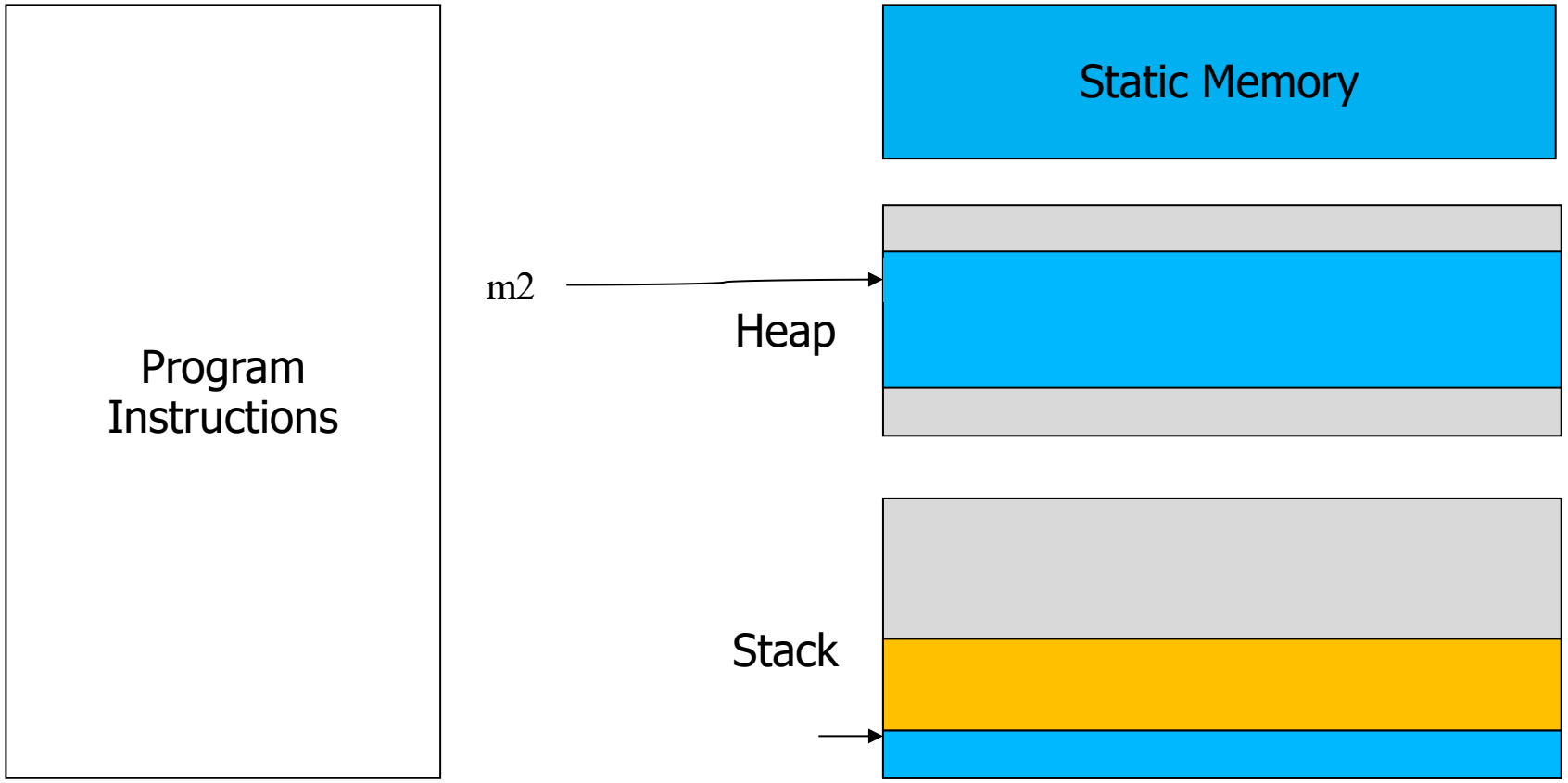



-> κλήση main -> κλήση f3

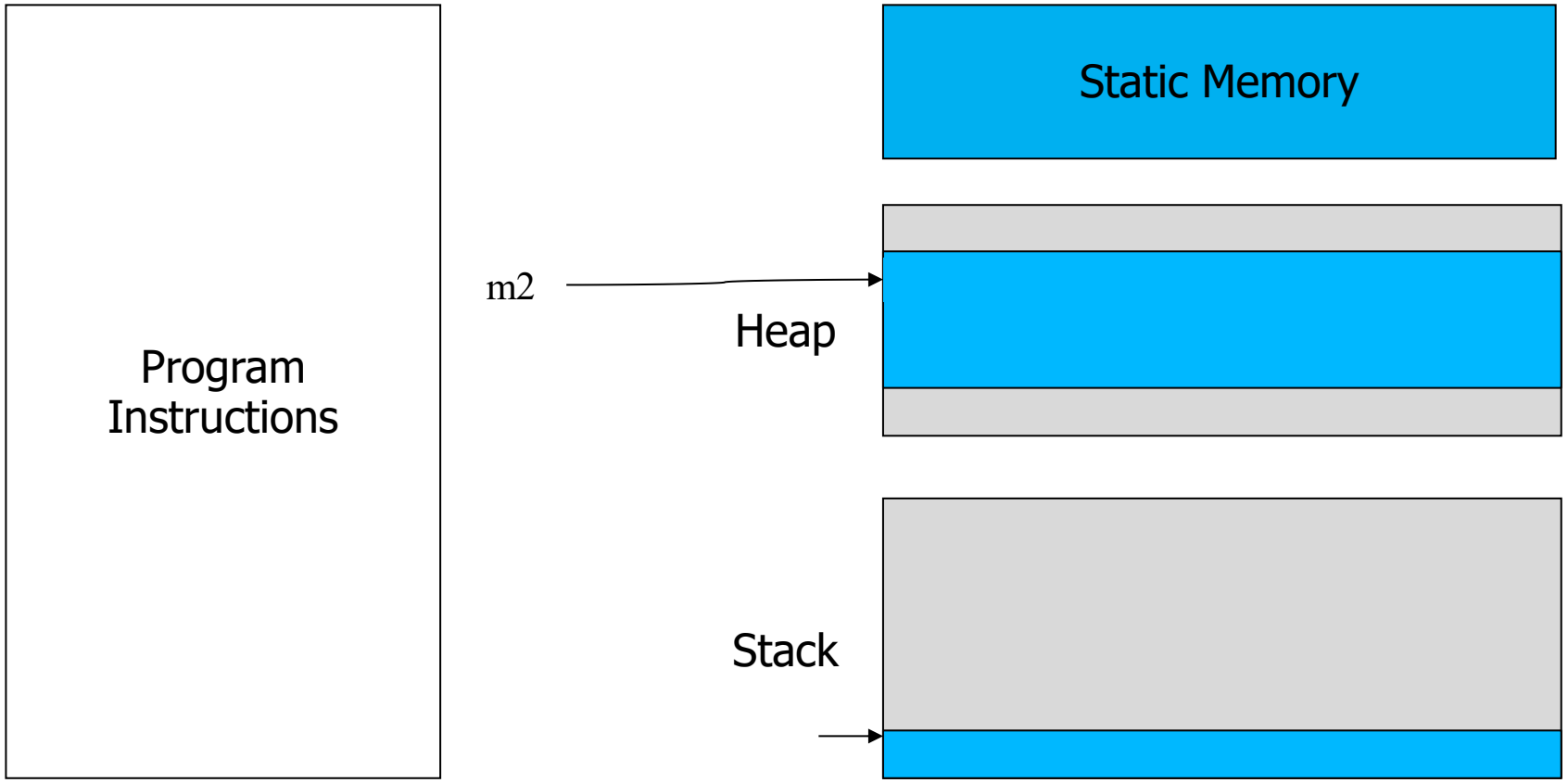
αποδέσμευση μνήμης m1



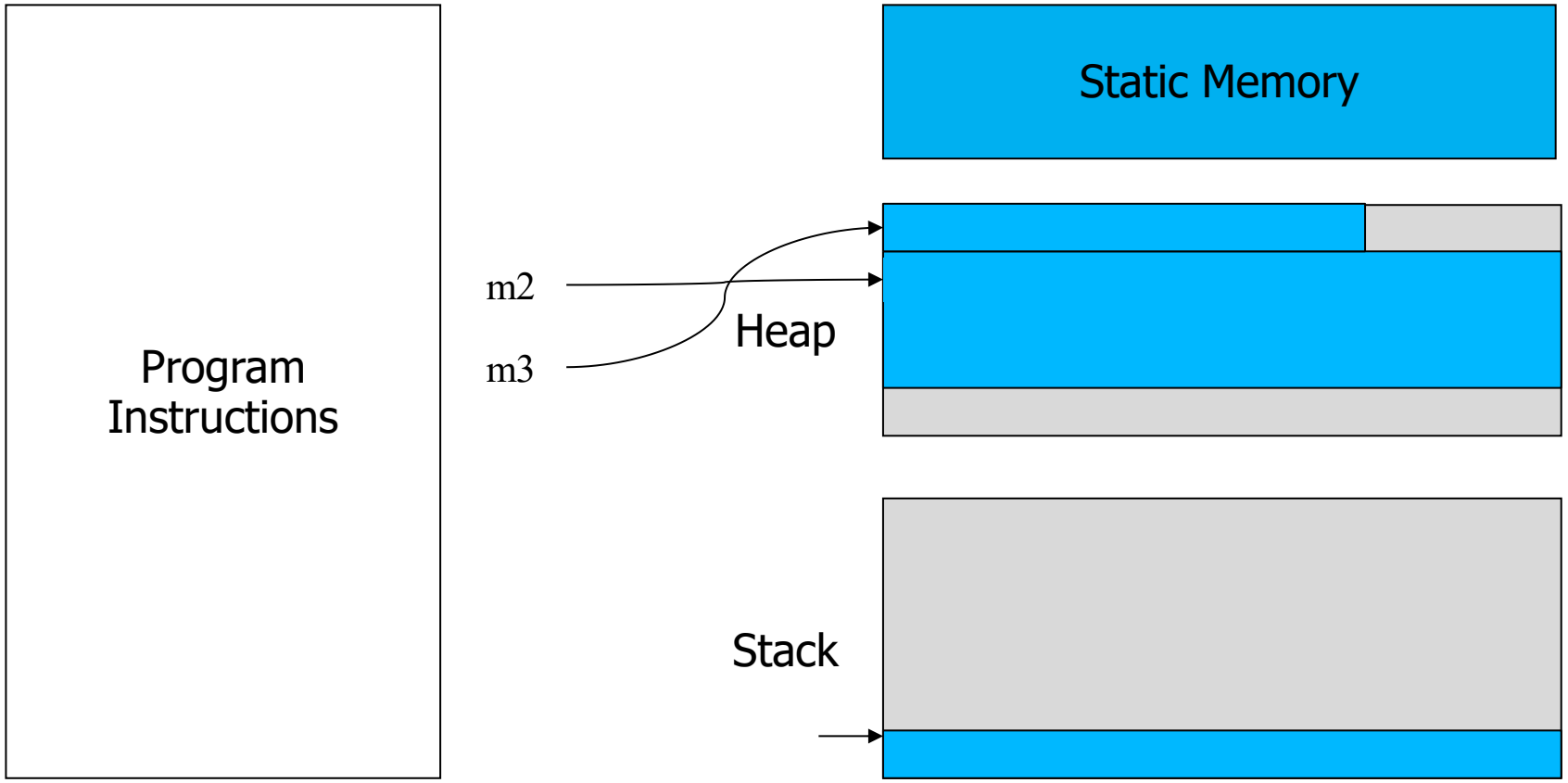

-> κλήση main -> κλήση f3




-> κλήση main <- επιστροφή f3

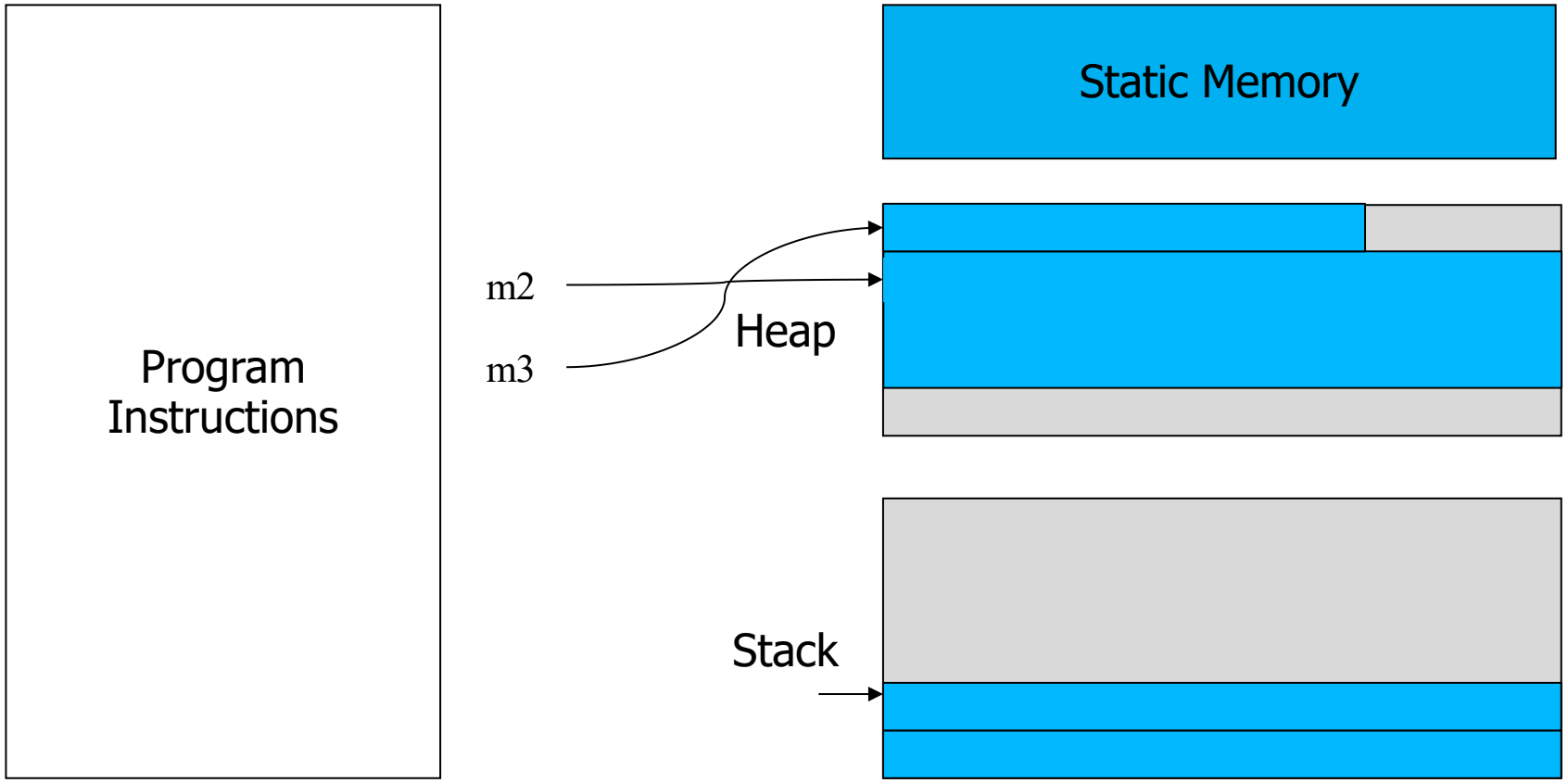



-> κλήση main

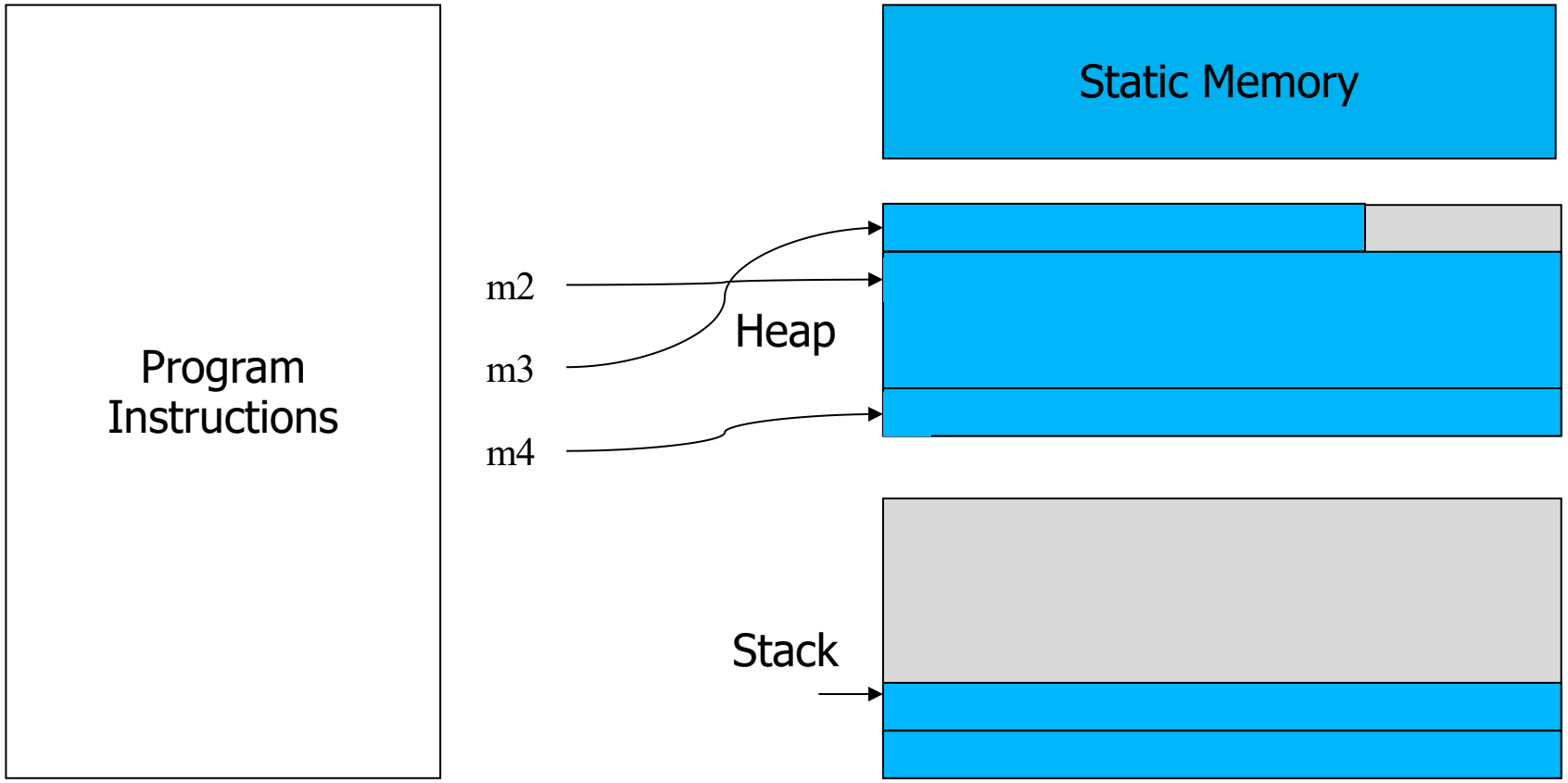



-> κλήση main

δέσμευση δυναμικής μνήμης m3

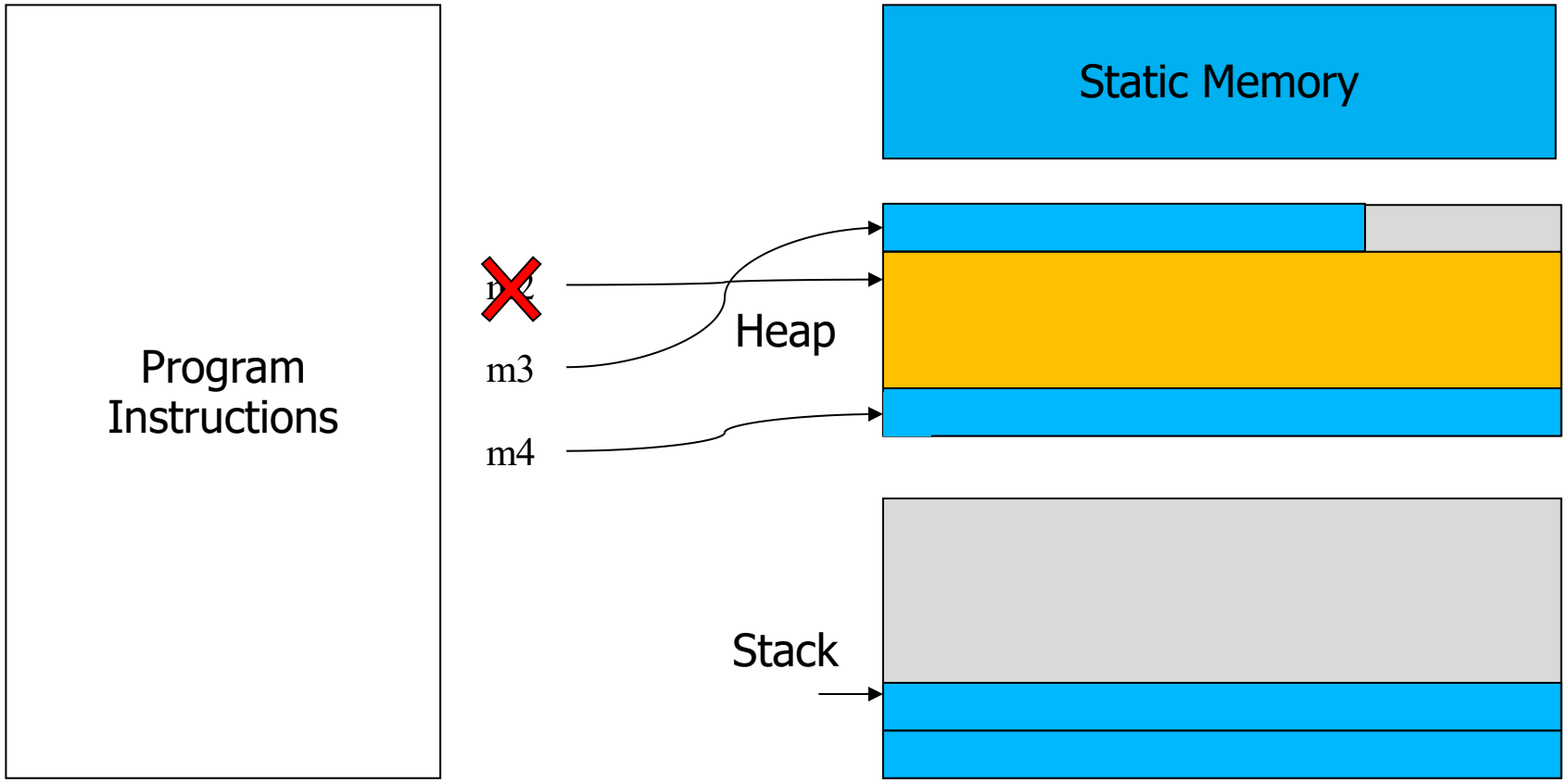



-> κλήση main -> κλήση f1



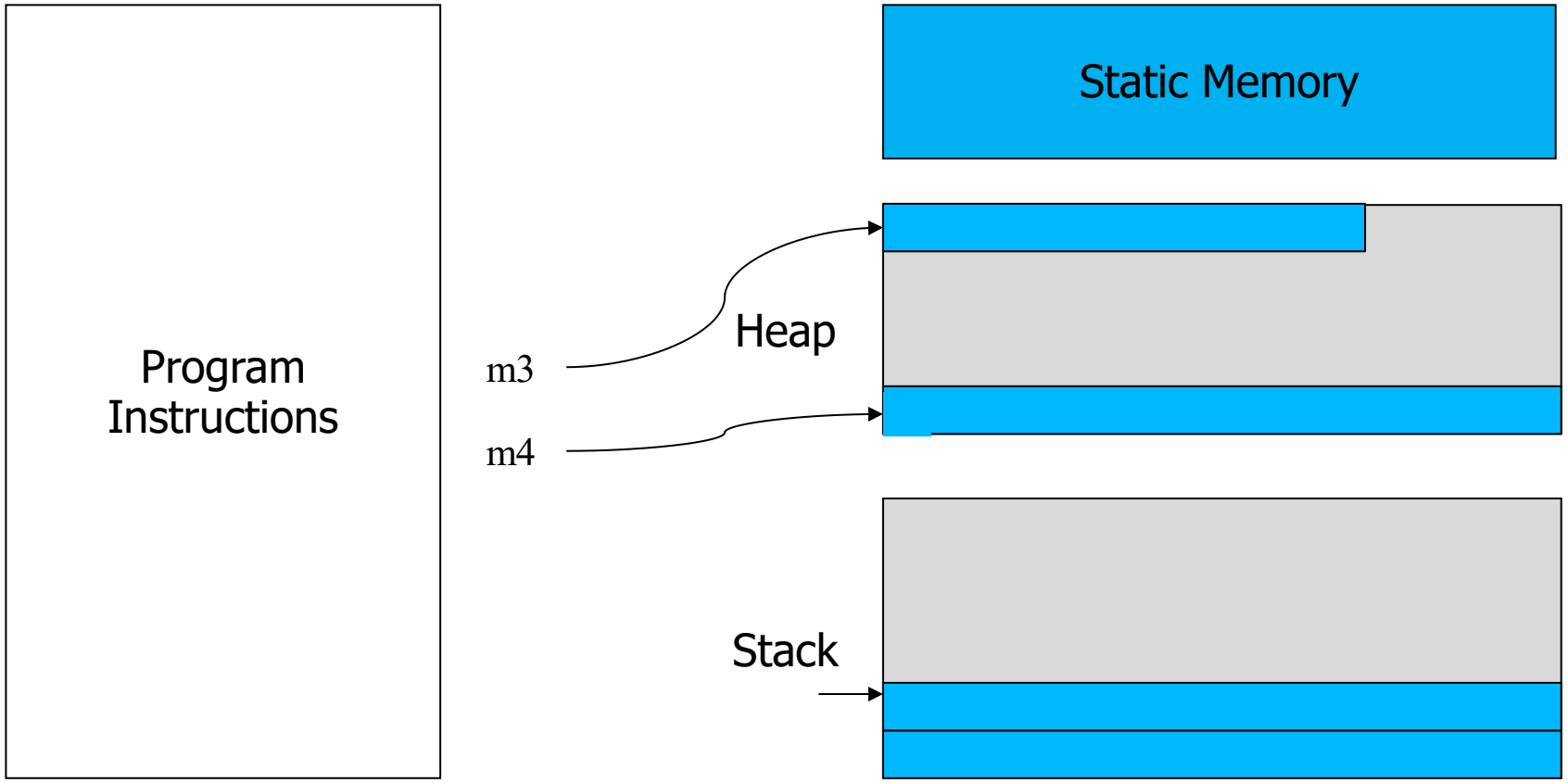

-> κλήση main -> κλήση f1

δέσμευση δυναμικής μνήμης m4

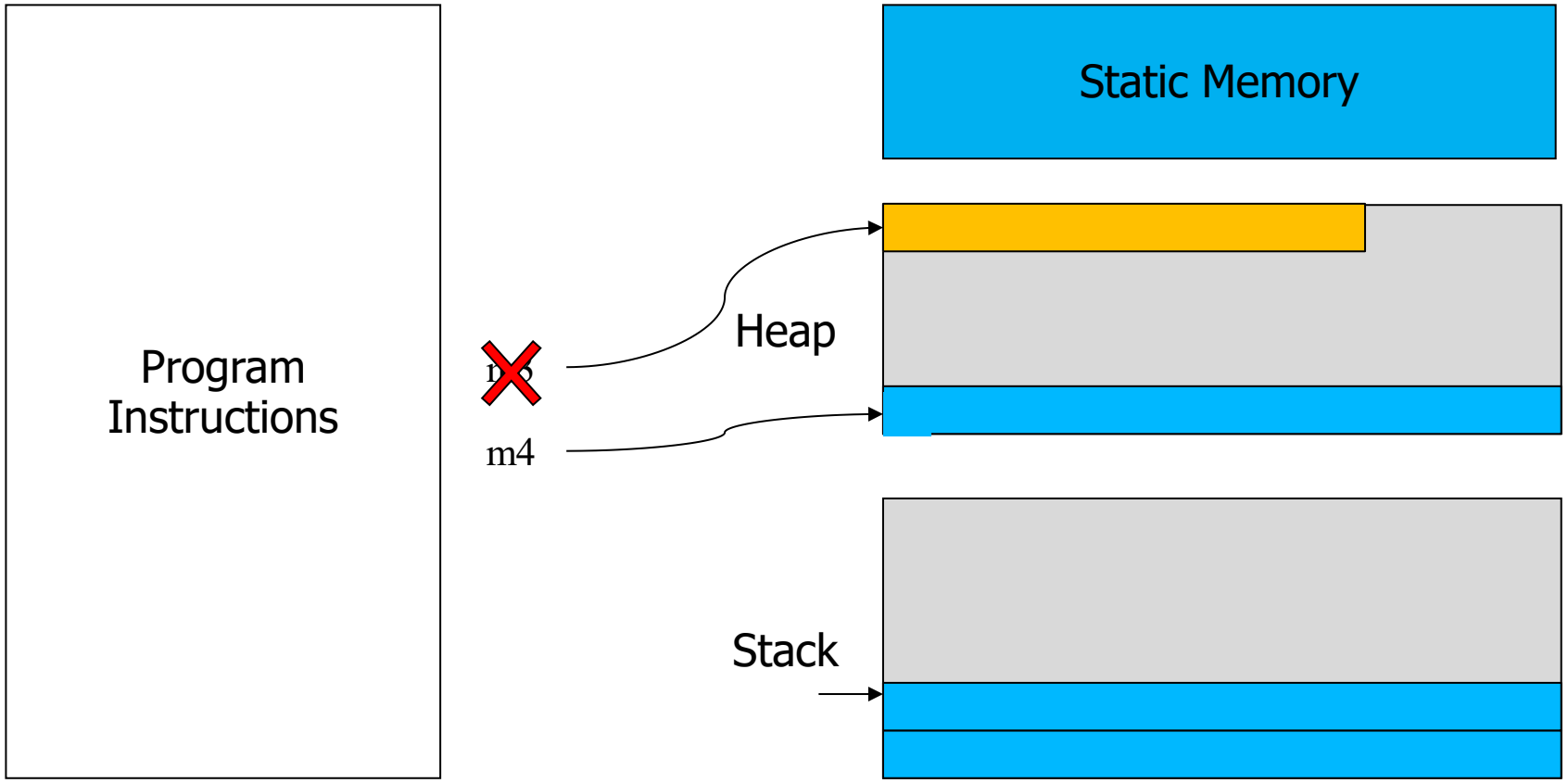



-> κλήση main -> κλήση f1

αποδέσμευση δυναμικής μνήμης m2

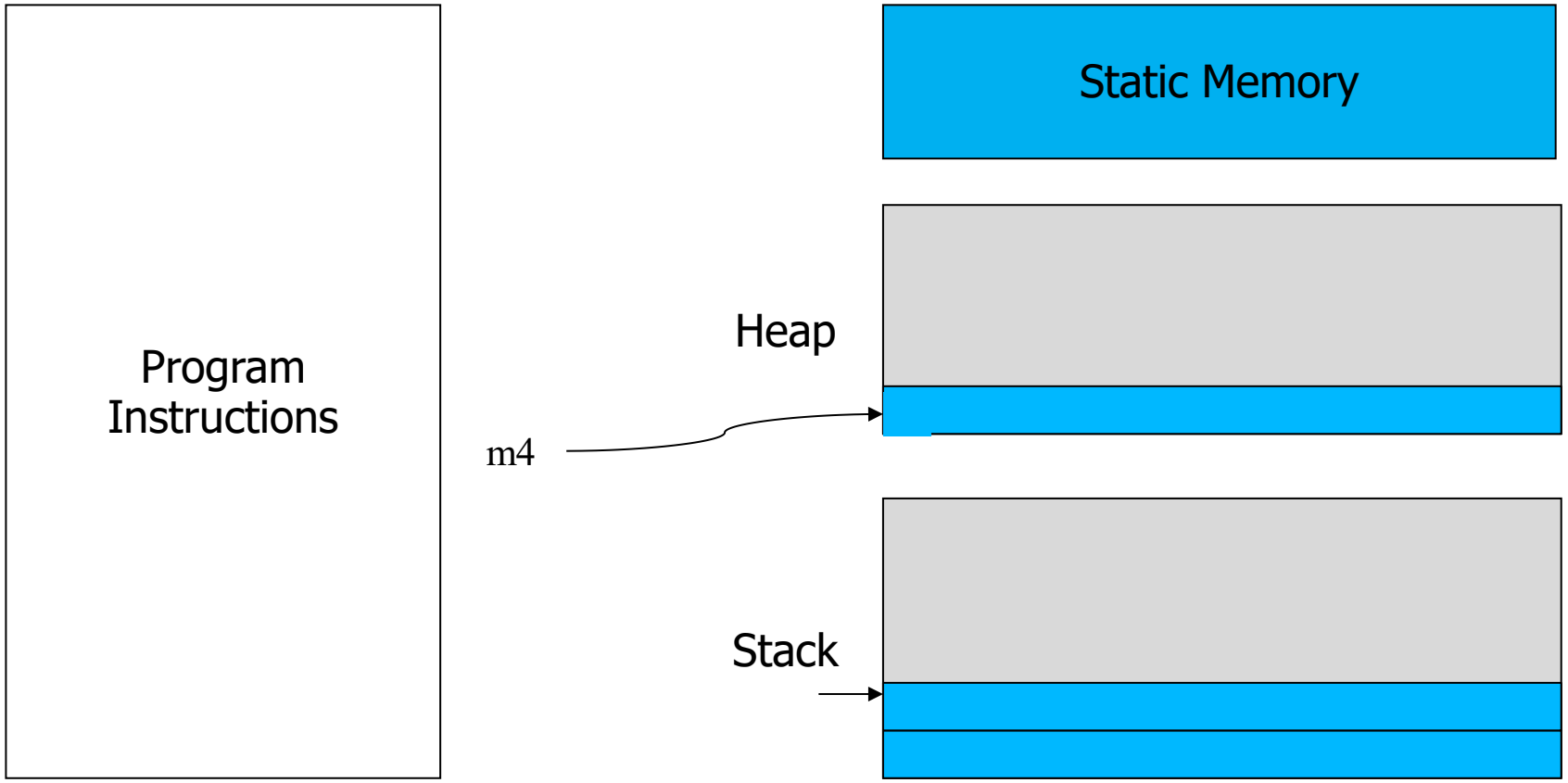



-> κλήση main -> κλήση f1

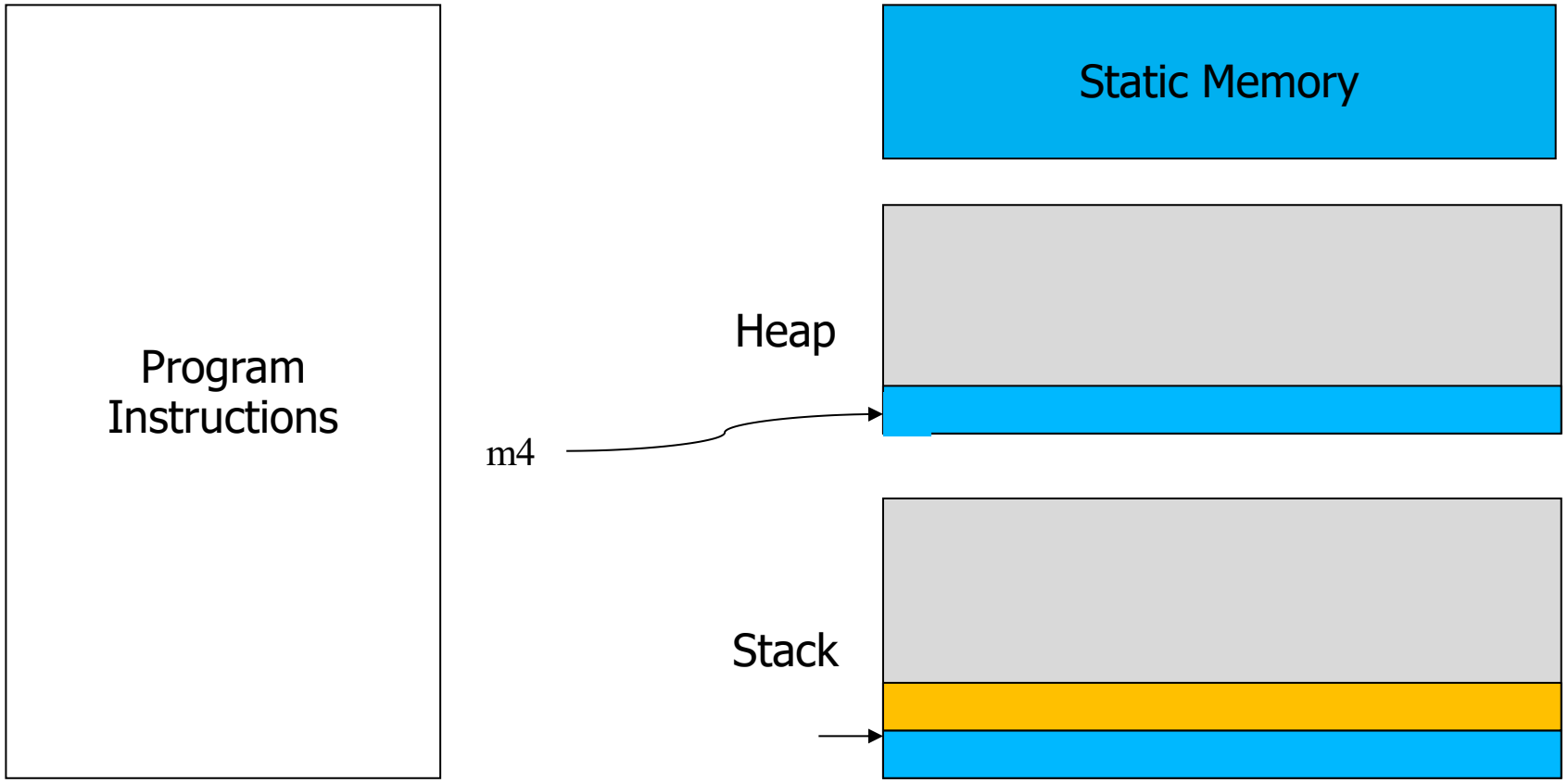



-> κλήση main -> κλήση f1

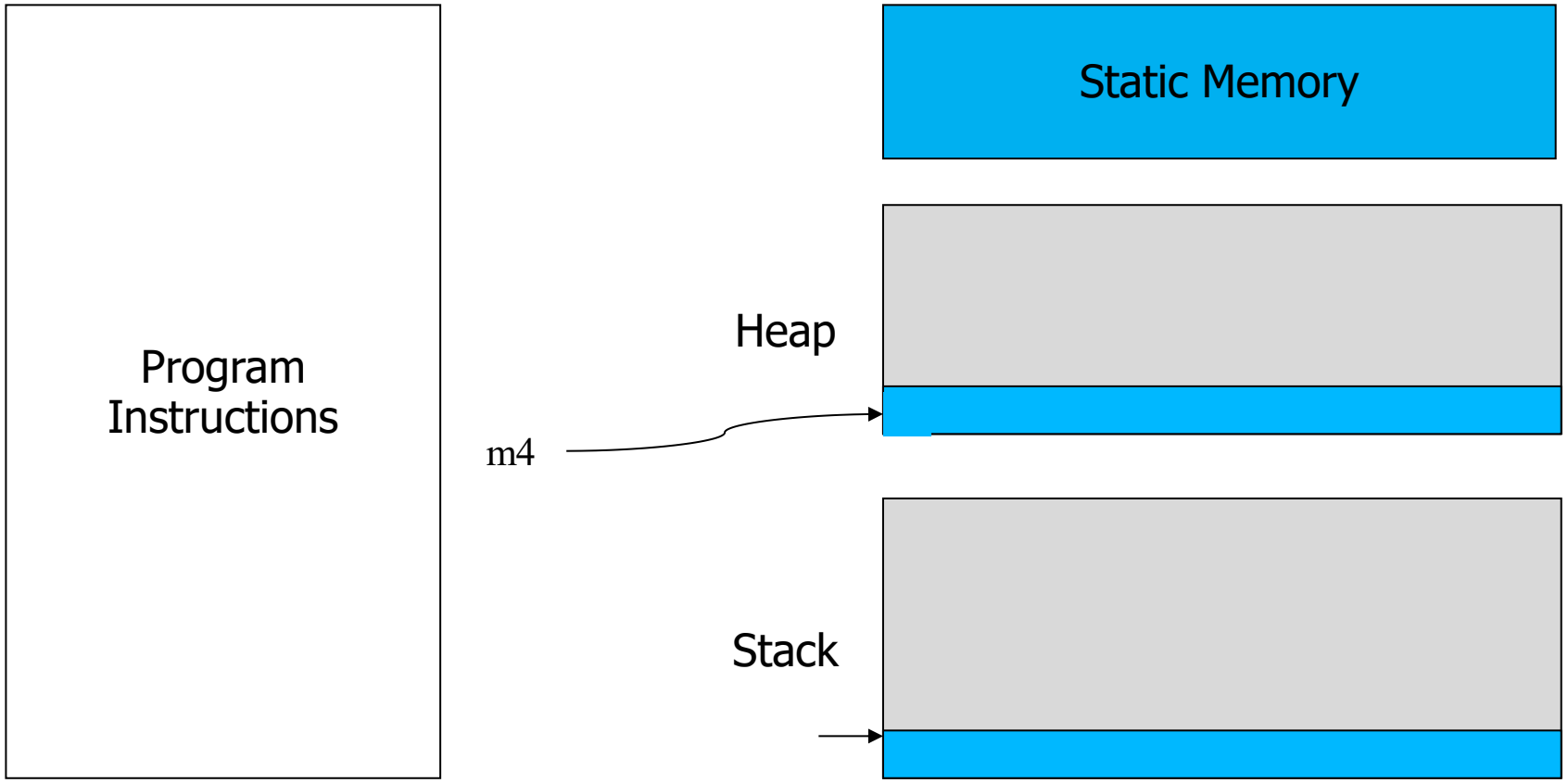
αποδέσμευση δυναμικής μνήμης m3



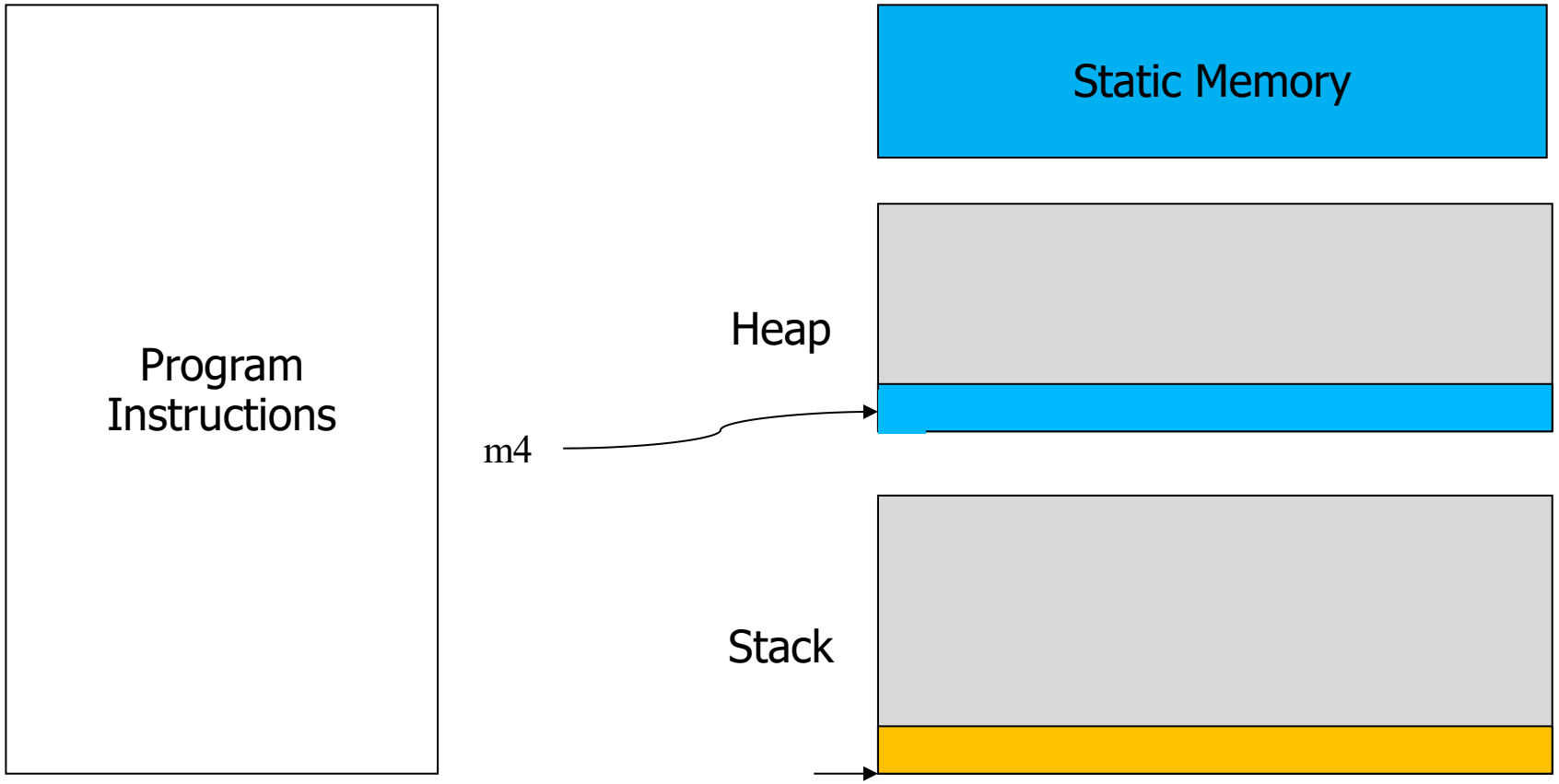

-> κλήση main -> κλήση f1



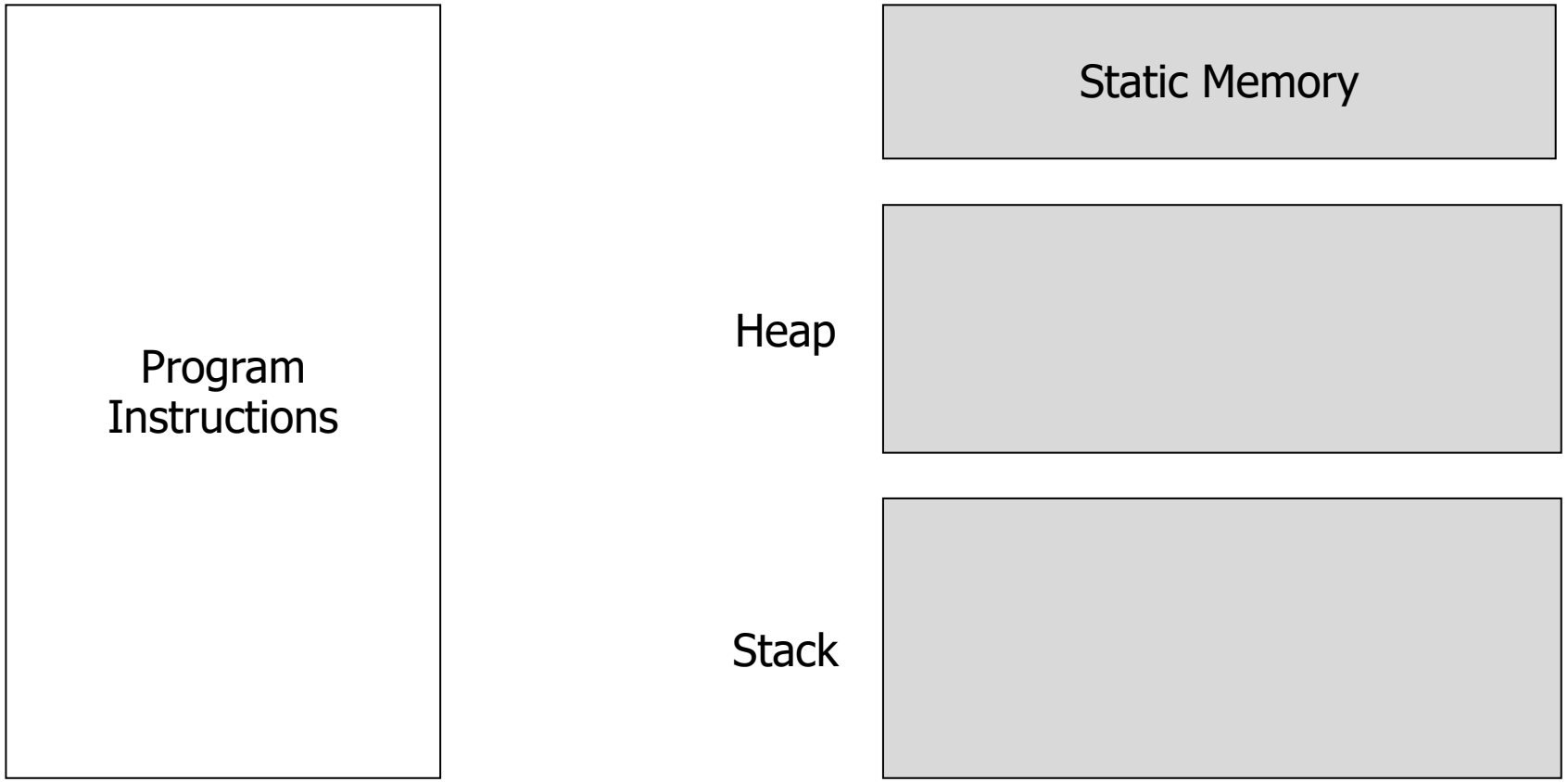

-> κλήση main <- επιστροφή f1




-> κλήση main




<- επιστροφή main


Program
Instructions
Static Memory
Heap
Stack

# Βασικές συναρτήσεις δυναμικής μνήμης

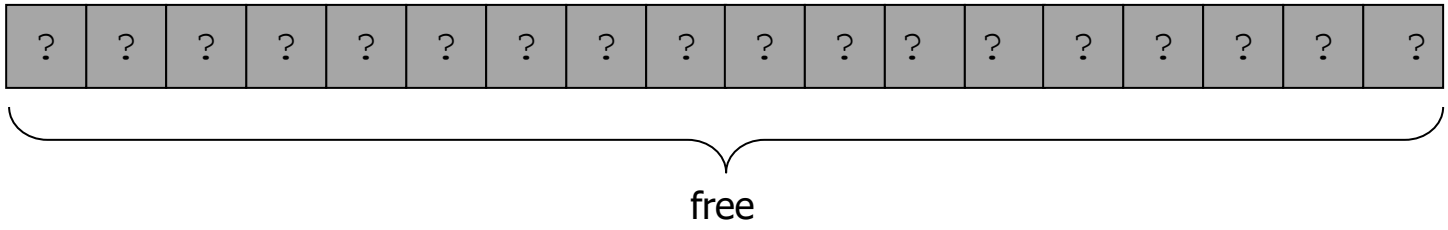
```
#include <stdlib.h>

void *malloc(size_t size);
void *calloc(size_t n, size_t size);
void *realloc(void *p, size_t size);
void free(void *p);
```

# Βασικές λειτουργίες (1)

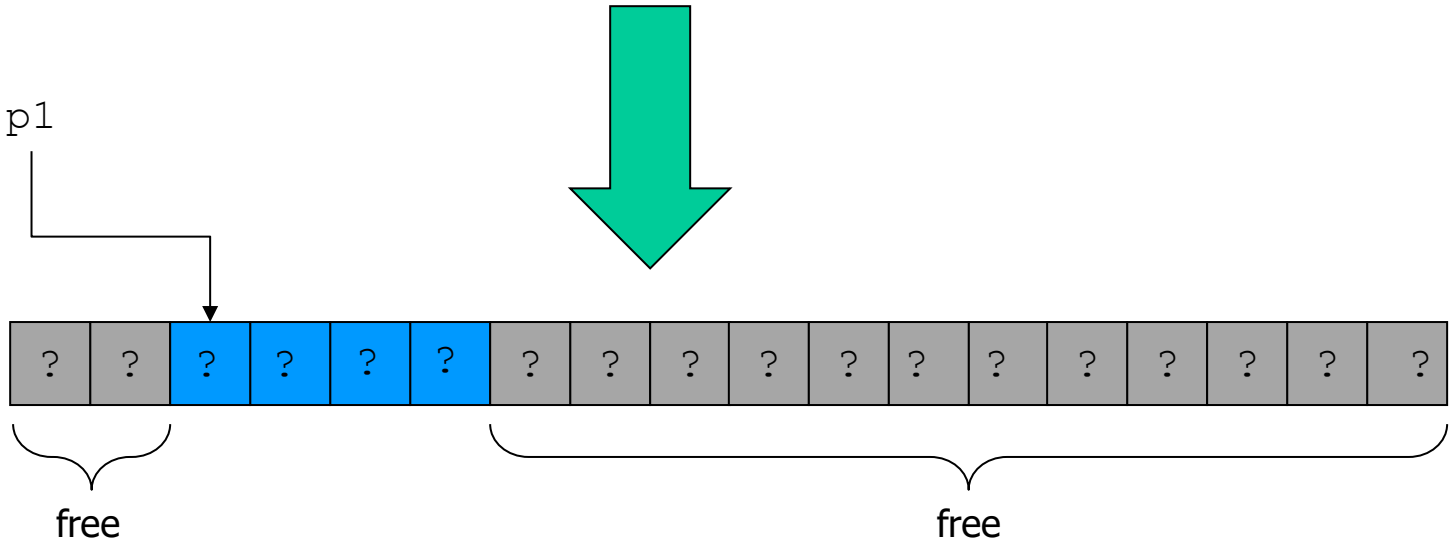
- `void *malloc(size_t size);`
  - δεσμεύει ένα **συνεχές** τμήμα μνήμης `size` bytes
  - επιστρέφει έναν δείκτη **στην αρχή** του τμήματος
  - τα περιεχόμενα της μνήμης **δεν αρχικοποιούνται**
  - πρέπει να **ελέγχεται** η τιμή επιστροφής
    - μπορεί να είναι `NULL` αν δεν υπάρχει διαθέσιμη μνήμη

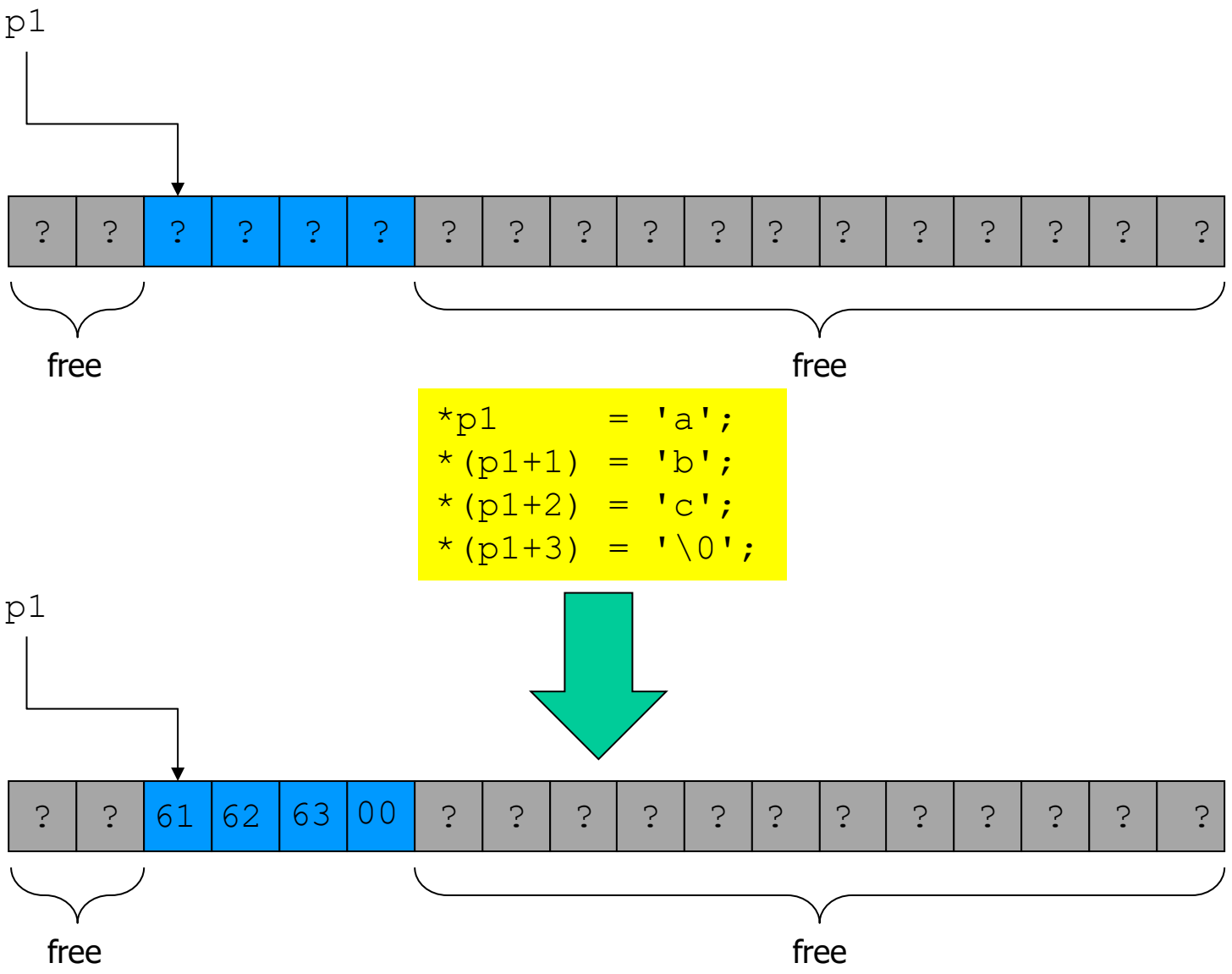
- `void free(void *ptr);`
  - αποδεσμεύει την μνήμη στην οποία δείχνει ο δείκτης `ptr`
  - αποδεσμεύεται **ολόκληρο** το αντίστοιχο τμήμα μνήμης
  - η τιμή του δείκτη `ptr` πρέπει να έχει επιστραφεί από κάποια προηγούμενη λειτουργία δέσμευσης μνήμης
    - **δεν** μπορεί να είναι τυχαία
  - η τιμή του δείκτη `ptr` **δεν** αλλάζει (**δεν** γίνεται `NULL`)

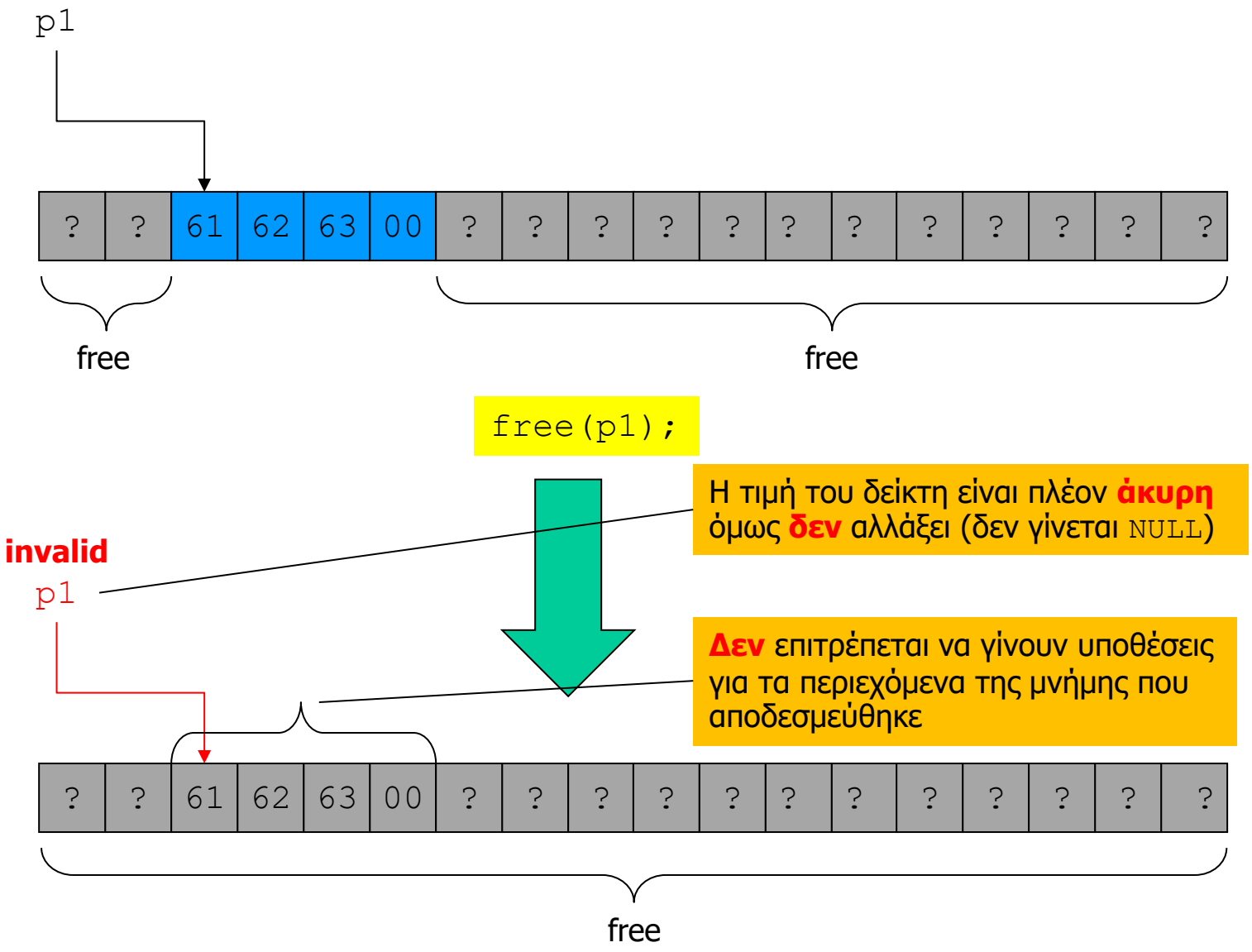
```
p1 = NULL
```




```
p1 = malloc(4);
```

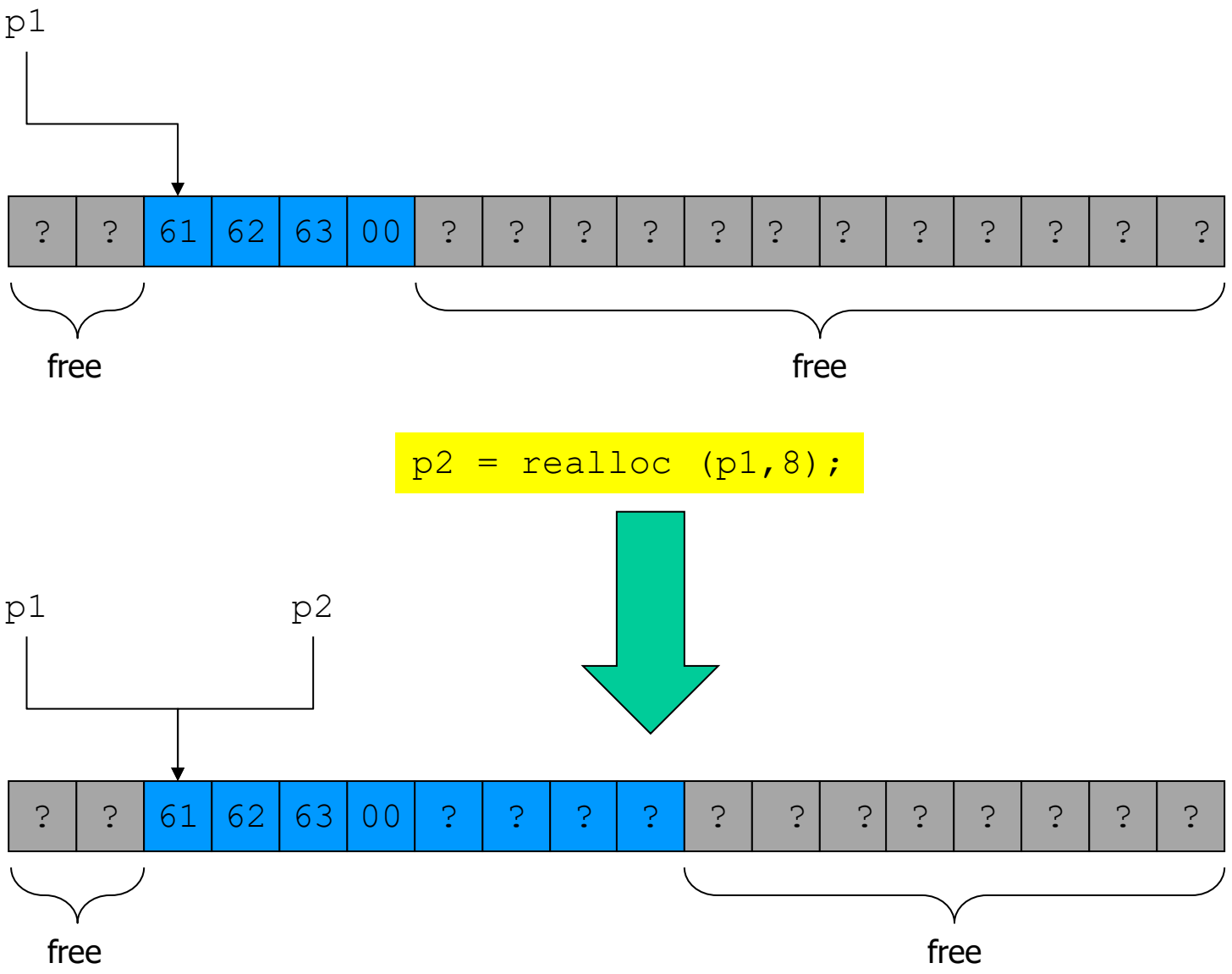


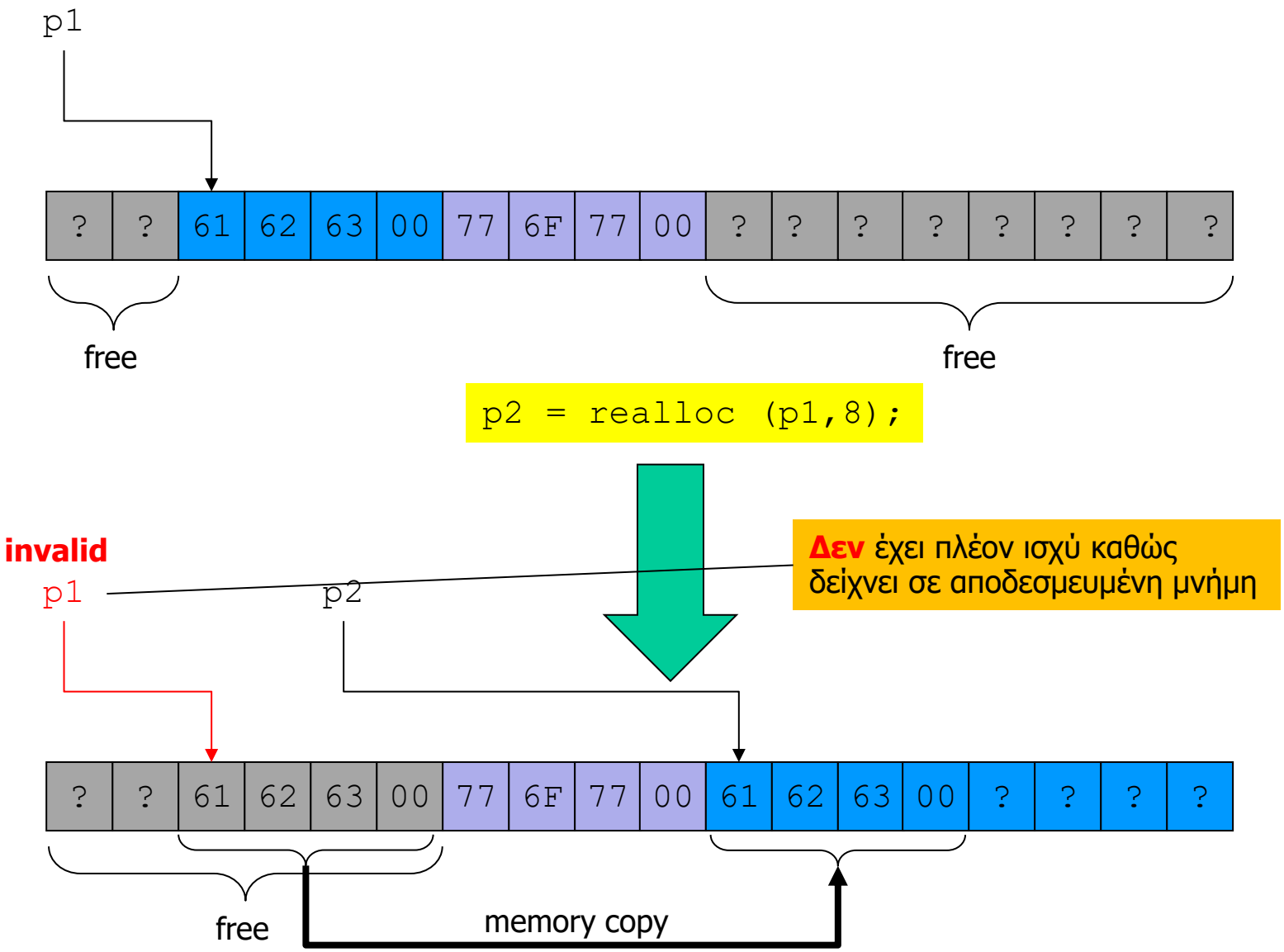

p1
? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?
free
free
*p1 = 'a';
*(p1+1) = 'b';
*(p1+2) = 'c';
*(p1+3) = '\0';
p1
? ? 61 62 63 00 ? ? ? ? ? ? ? ? ? ? ? ?
free
free

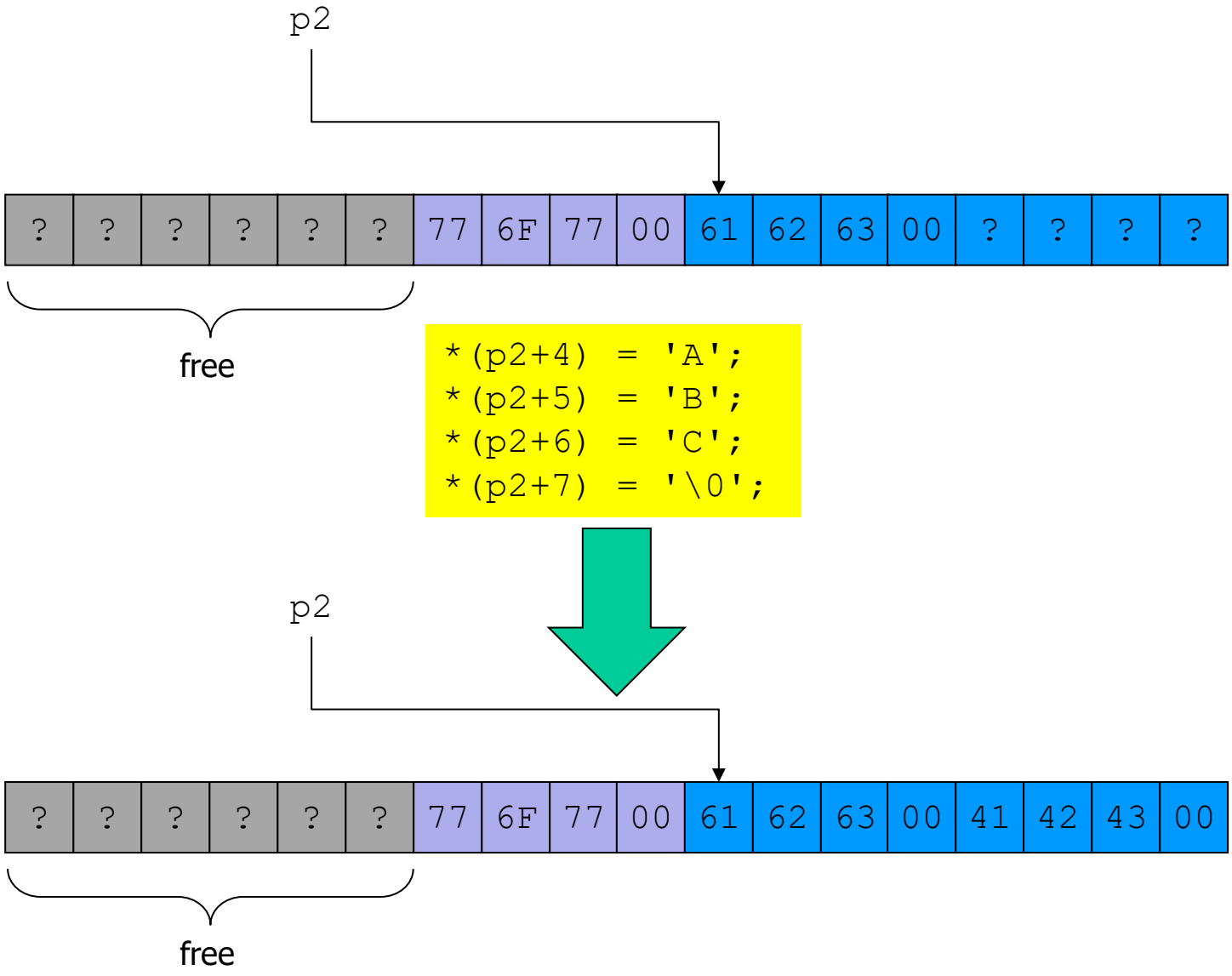

p1
? ? 61 62 63 00 ? ? ? ? ? ? ? ? ? ? ? ?
free
free
free(p1);
Η τιμή του δείκτη είναι πλέον άκυρη όμως δεν αλλάξει (δεν γίνεται NULL)
invalid
p1
Δεν επιτρέπεται να γίνουν υποθέσεις για τα περιεχόμενα της μνήμης που αποδεσμεύθηκε
? ? 61 62 63 00 ? ? ? ? ? ? ? ? ? ? ? ?
free

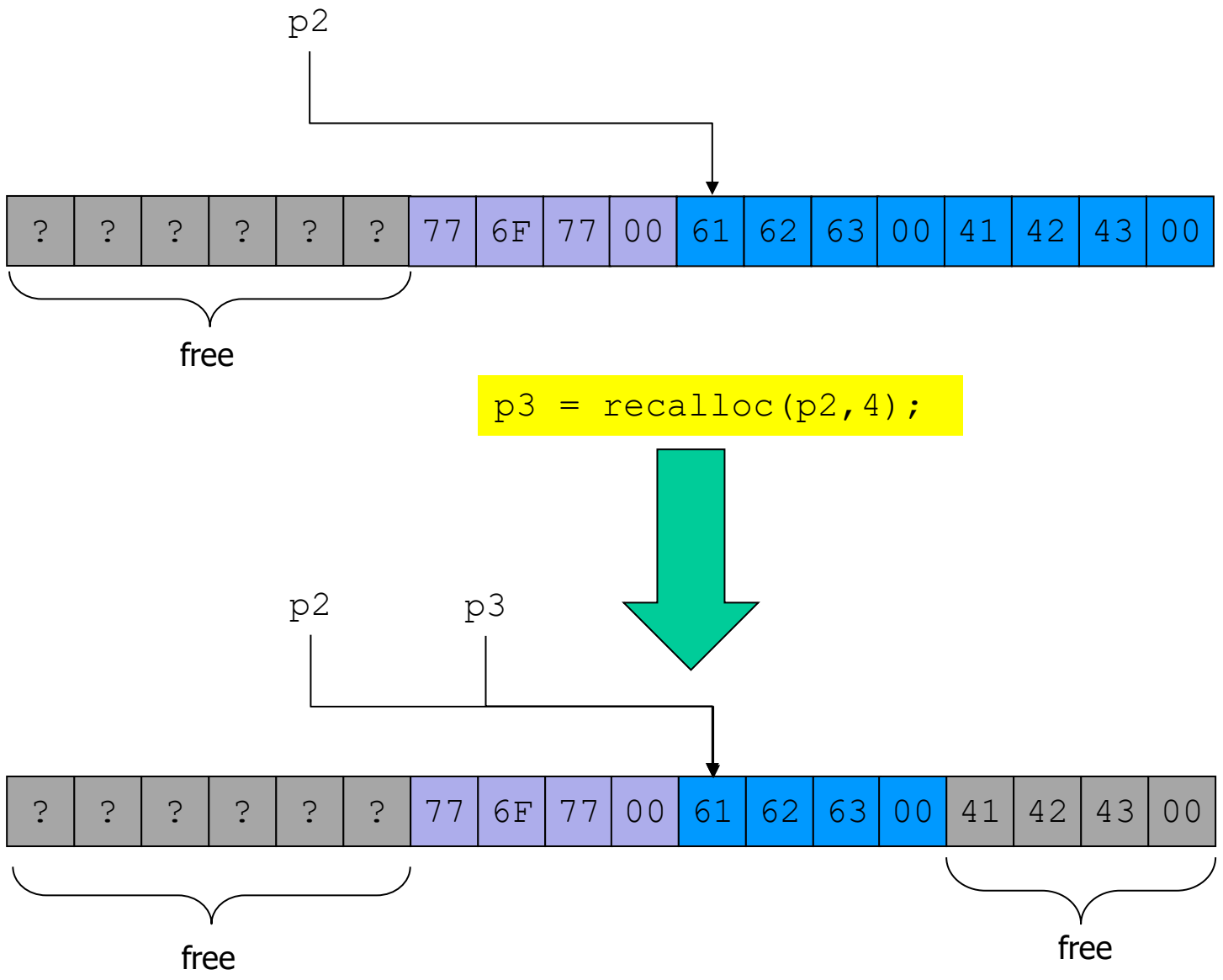
# Βασικές λειτουργίες (2)

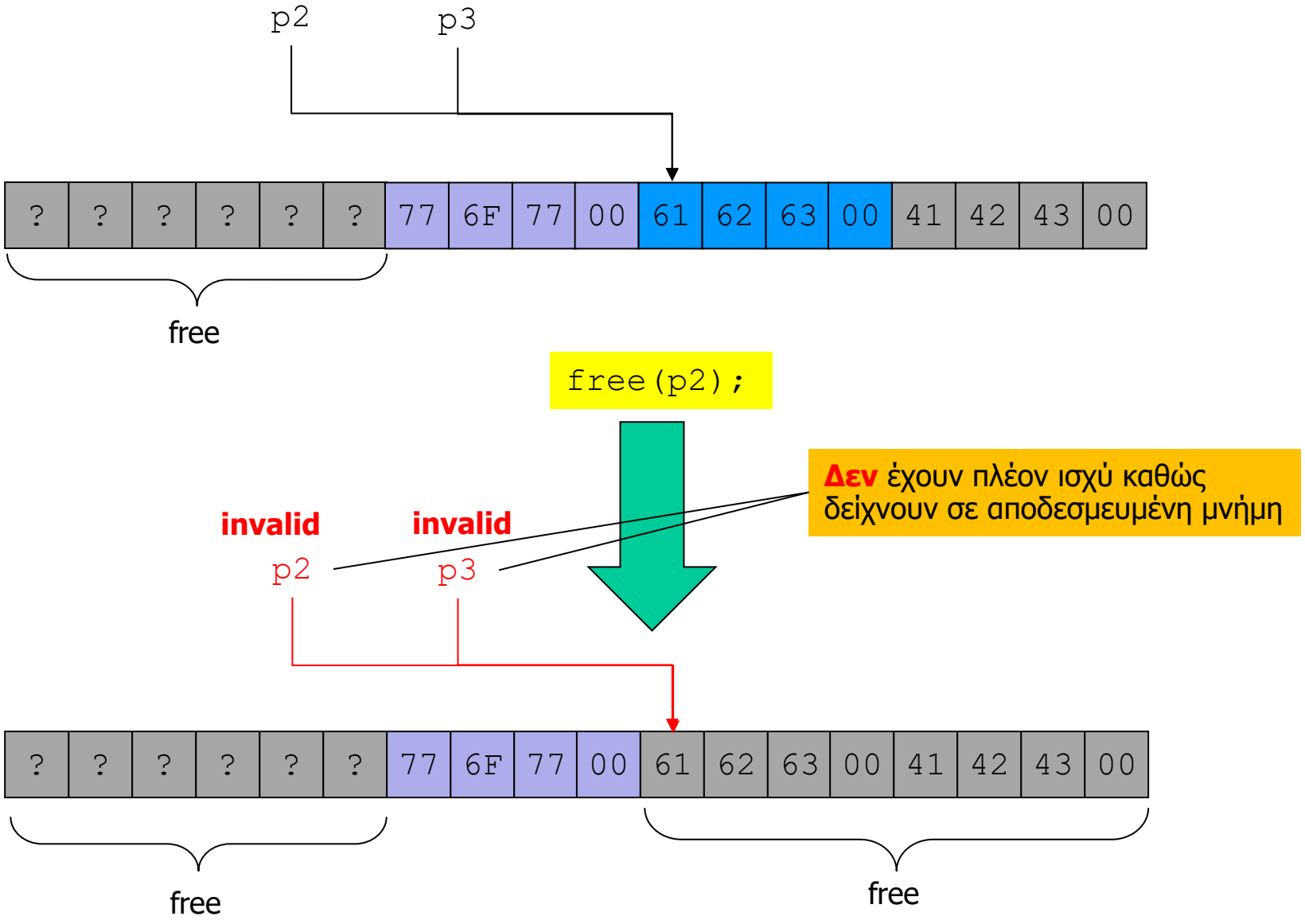
- `void *calloc(size_t nmemb, size_t size);`
  - δεσμεύει ένα **συνεχές** κομμάτι μνήμης `nmemb * size` bytes
  - επιστρέφει έναν δείκτη **στην αρχή** του κομματιού
  - τα περιεχόμενα της μνήμης **αρχικοποιούνται** σε `0`
- `void *realloc(void *ptr, size_t size);`
  - **προσαρμόζει** το κομμάτι μνήμης που δείχνει ο δείκτης `ptr` ώστε να γίνει `size` bytes
  - η τιμή του δείκτη `ptr` **δεν** αλλάζει
  - τα προηγούμενα περιεχόμενα της μνήμης **δεν αλλάζουν**
  - τυχόν επιπλέον bytes **δεν αρχικοποιούνται**
  - **προσοχή:** μπορεί να γίνει **αντιγραφή** των δεδομένων σε άλλη περιοχή της μνήμης (που δεσμεύεται εκ νέου, με απελευθέρωση της μνήμης που δείχνει ο δείκτης `ptr`), οπότε η τιμή επιστροφής **δεν** είναι ίση με τη τιμή `ptr`
  - αν η τιμή `ptr` είναι `NULL` γίνεται ότι και στην `malloc()`

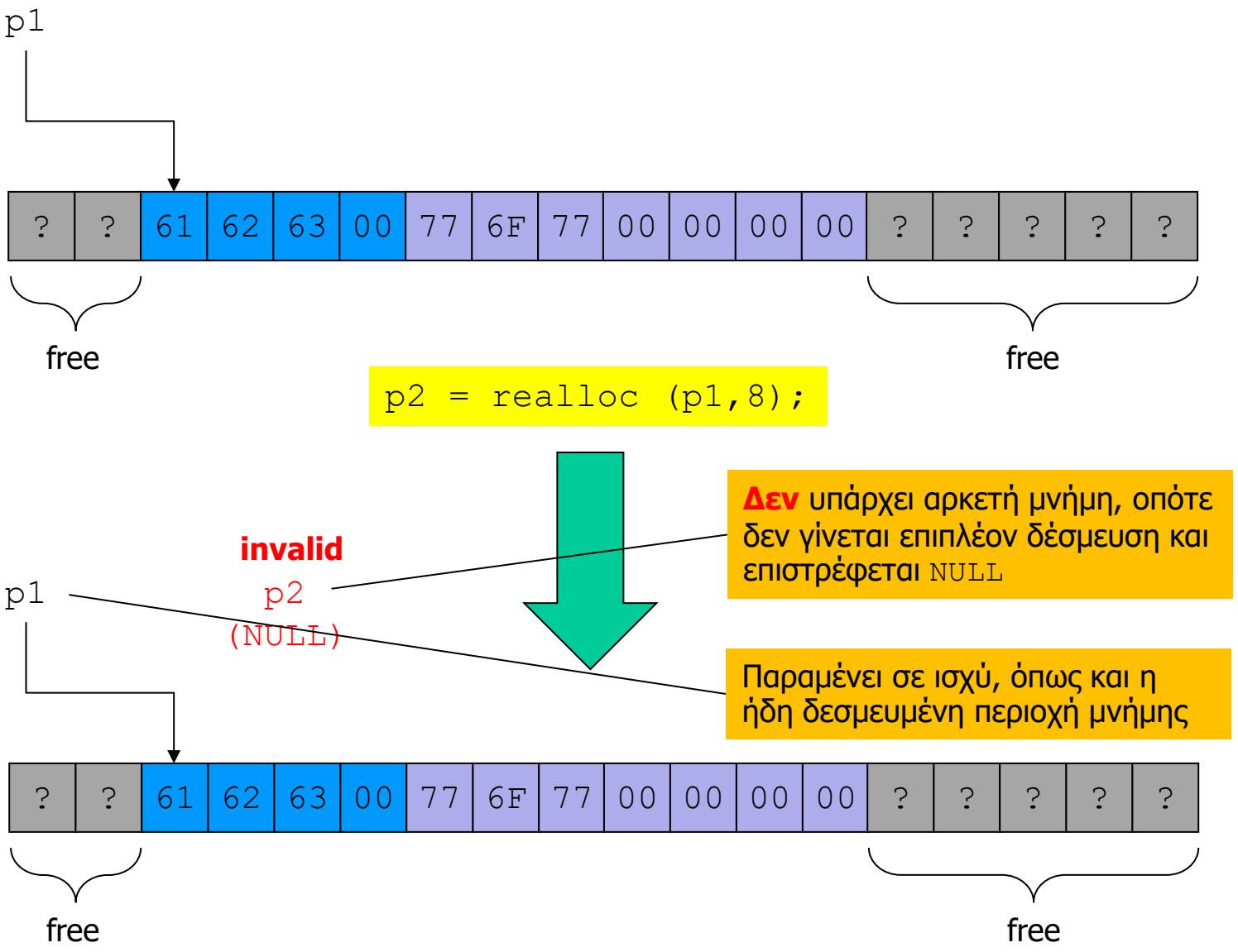

p1
? ? 61 62 63 00 ? ? ? ? ? ? ? ? ? ? ? ?
free
free
p2 = realloc (p1,8);
p1
p2
? ? 61 62 63 00 ? ? ? ? ? ? ? ? ? ? ? ?
free
free


p1
? ? 61 62 63 00 77 6F 77 00 ? ? ? ? ? ? ? ?
free
free
p2 = realloc (p1,8);
invalid
p1
p2
Δεν έχει πλέον ισχύ καθώς δείχνει σε αποδεσμευμένη μνήμη
? ? 61 62 63 00 77 6F 77 00 61 62 63 00 ? ? ? ?
free
memory copy


p2
? ? ? ? ? ? 77 6F 77 00 61 62 63 00 ? ? ? ?
free
*(p2+4) = 'A';
*(p2+5) = 'B';
*(p2+6) = 'C';
*(p2+7) = '\0';
p2
? ? ? ? ? ? 77 6F 77 00 61 62 63 00 41 42 43 00
free


p2
? ? ? ? ? ? 77 6F 77 00 61 62 63 00 41 42 43 00
free
p3 = recalloc(p2,4);
p2
p3
? ? ? ? ? ? 77 6F 77 00 61 62 63 00 41 42 43 00
free
free


p2
p3
? ? ? ? ? ? 77 6F 77 00 61 62 63 00 41 42 43 00
free
free(p2);
Δεν έχουν πλέον ισχύ καθώς δείχνουν σε αποδεσμευμένη μνήμη
invalid
p2
invalid
p3
? ? ? ? ? ? 77 6F 77 00 61 62 63 00 41 42 43 00
free
free


p1
? ? 61 62 63 00 77 6F 77 00 00 00 00 ? ? ? ? ?
free
free
p2 = realloc (p1,8);
invalid
p2
(NULL)
Δεν υπάρχει αρκετή μνήμη, οπότε δεν γίνεται επιπλέον δέσμευση και επιστρέφεται NULL
Παραμένει σε ισχύ, όπως και η ήδη δεσμευμένη περιοχή μνήμης
p1
? ? 61 62 63 00 77 6F 77 00 00 00 00 ? ? ? ? ?
free
free

# Έλεγχος τιμής που επιστρέφεται

- Οι `malloc`, `calloc` και `realloc` επιστρέφουν `NULL` αν δεν μπορεί να δεσμευτεί όση μνήμη ζητήθηκε
- Η τιμή επιστροφής **πρέπει να ελέγχεται**
- Ώστε το πρόγραμμα να γνωρίζει ότι **δεν** έλαβε την μνήμη που ζήτησε
  - να λαμβάνει τις απαραίτητες αποφάσεις / ενέργειες
  - να αποφεύγονται στην συνέχεια άκυρες αναφορές σε δείκτη με τιμή `NULL` (προκαλούν τον τερματισμό)

# Χρήση δυναμικής μνήμης

- Η διεύθυνση που επιστρέφεται χρησιμοποιείται με **αποκλειστική ευθύνη** του προγραμματιστή
  - οι λειτουργίες δυναμικής μνήμης **δεν γνωρίζουν** το πως θα χρησιμοποιηθεί η μνήμη (μέσα από τι είδους τύπους δεδομένων θα γραφτούν/διαβαστούν τα περιεχόμενα της)
- Συνήθως ένα πρόγραμμα δεσμεύει μνήμη στο (πολλαπλάσιο) μέγεθος ενός αντικειμένου τύπου `T`
- Και η διεύθυνση που επιστρέφεται, ανατίθεται (με type casting) σε μια μεταβλητή τύπου `T*`
  - για την σωστή αποθήκευση/ερμηνεία των δεδομένων
- Ο προγραμματιστής πρέπει να **αποδεσμεύει** την δυναμική μνήμη που δεν χρησιμοποιεί το πρόγραμμα
  - αποφυγή σπατάλης της διαθέσιμης μνήμης

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

int main(int argc, char *argv[]) {
  int a, b, *sumPtr;

  scanf("%d %d", &a, &b);

  sumPtr = (int *) malloc( sizeof(int) );

  if (sumPtr == NULL) {
    printf("no memory\n");
    return(1);
  }

  *sumPtr = a + b;
  printf("%d\n", *sumPtr);

  free(sumPtr);
  return(0);
}
```

δέσμευση μνήμης από `sizeof(int)` bytes

type cast σε δείκτη-σε-ακέραιο

αποδέσμευση μνήμης

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

int main(int argc, char *argv[]) {
  int *table, size, i;

  size = 1;
  table = (int *) malloc(sizeof(int)*size);
  /* Should check malloc return value */

  while (scanf("%d", &table[size-1]) == 1) {
    size++;
    table = (int *) realloc(table, sizeof(int)*size);
  }

  for (i=0; i<size-1; i++) {
    printf("%d ", table[i]);
  }
  printf("\n");

  free(table);
  return(0);
}
```

γιατί αυτό είναι (πιθανό) **bug**;

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

int main(int argc, char *argv[]) {
  int *table, *new_table, size, i;

  size = 1;
  table = (int *) malloc(sizeof(int));
  /* Should check malloc return value */

  while (scanf("%d", &table[size-1]) == 1) {
    size++;
    new_table = (int *) realloc(table, sizeof(int)*size);
    if (new_table == NULL) {
      /* handle the case realloc fails */
    }
    else {
      table = new_table;
    }
  }

  for (i=0; i<size-1; i++) {
    printf("%d ", table[i]);
  }
  printf("\n");

  free(table);
  return(0);
}
```

# Ισχύς δυναμικής μνήμης

- Η δυναμική μνήμη υφίσταται **καθ' όλη** τη διάρκεια της εκτέλεσης του προγράμματος
- Έχει ισχύ **ανεξάρτητα** από το πλαίσιο συναρτήσεων
- Δυναμική μνήμη που δεσμεύεται μέσα από συνάρτηση **παραμένει εν ισχύ** ακόμα και μετά την κλήση της
- Άλλος ένας τρόπος επιστροφής αποτελεσμάτων μέσα από το (προσωρινό) περιβάλλον κλήσης συνάρτησης
  - η συνάρτηση δεσμεύει δυναμική μνήμη όπου αποθηκεύει τα δεδομένα, και επιστρέφει σαν αποτέλεσμα την διεύθυνση
- Το περιβάλλον κλήσης («πελάτης» της συνάρτησης) πρέπει να **αποδέσμευσει** την μνήμη μετά την χρήση

```c
#include <stdio.h>

int *add(int a, int b) {
  int sum;
  sum = a + b;
  return(&sum);
}

int main(int argc, char *argv[]) {
  int a, b, *sumPtr;

  scanf("%d %d", &a, &b);

  sumPtr = add(a, b);
  printf("%d\n", *sumPtr);

  return(0);
}
```

δείκτης στην **στοίβα**:
αυτό είναι **λάθος!**

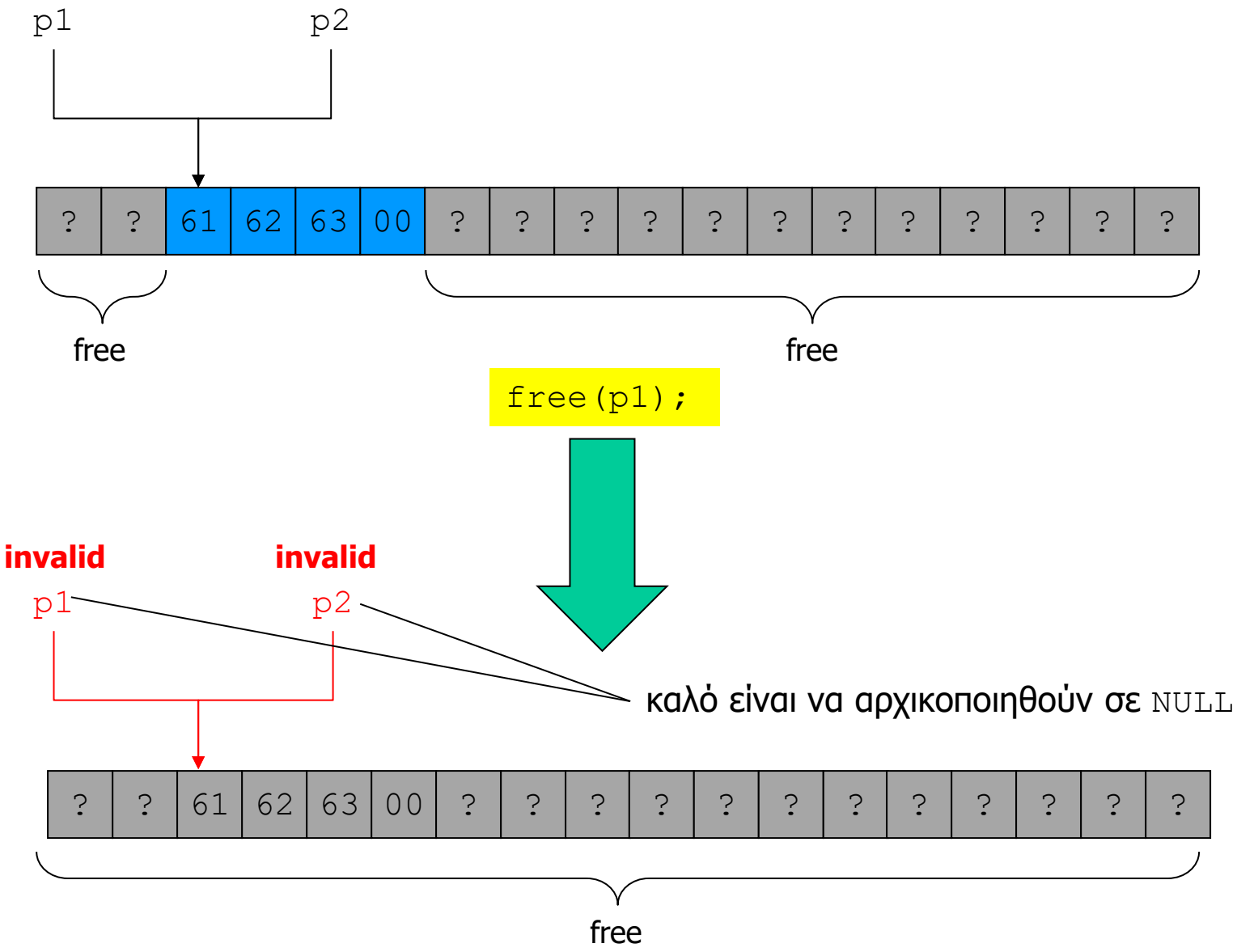
```c
#include <stdio.h>
#include <stdlib.h>

int *add(int a, int b) {
  int *sumPtr = (int *)malloc(sizeof(int));
  *sumPtr = a + b;
  return(sumPtr);
}

int main(int argc, char *argv[]) {
  int a, b, *sumPtr;

  scanf("%d %d", &a, &b);

  sumPtr = add(a, b);
  printf("%d\n", *sumPtr);

  free(sumPtr);

  return(0);
}
```

δείκτης σε **δυναμική μνήμη**: αυτό είναι **σωστό!**

αυτός που κάλεσε την συνάρτηση πρέπει να **απελευθερώσει** την δυναμική μνήμη που δεσμεύτηκε μέσα από την συνάρτηση

# Ισχύς δεικτών

- Ένας δείκτης με τιμή `NULL` προφανώς αντιστοιχεί σε άκυρη διεύθυνση που δεν μπορεί να προσπελαστεί
- **Δεν** ισχύει το αντίστροφο!
- Ένας δείκτης που **δεν** είναι `NULL` **μπορεί** να αντιστοιχεί σε **άκυρη** διεύθυνση
  - μνήμη που **δεν έχει δεσμευθεί** από το πρόγραμμα
  - μνήμη που έχει **αποδεσμευθεί** από το πρόγραμμα
- Μετά την `free()` καλό είναι ο δείκτης που περάστηκε ως παράμετρος να αρχικοποιείται ρητά σε `NULL`
  - καθώς και άλλοι δείκτες που δείχνουν στην ίδια μνήμη
- Εκτός αν πρόκειται να λάβει άμεσα νέα τιμή ή δεν θα χρησιμοποιηθεί ποτέ ξανά – **famous last words!**


p1
p2
? ? 61 62 63 00 ? ? ? ? ? ? ? ? ? ? ? ?
free
free
free(p1);
invalid
invalid
p1
p2
καλό είναι να αρχικοποιηθούν σε NULL
? ? 61 62 63 00 ? ? ? ? ? ? ? ? ? ? ? ?
free

# Κλασικά bugs

1. Προσπέλαση μνήμης **πέρα** από τα όρια του τμήματος που έχει δεσμευθεί
   - αναλογία: προσπέλαση μνήμης πέρα από τα όρια ενός πίνακα
2. Προσπέλαση μνήμης **μετά** την αποδέσμευση
   - αναλογία: προσπέλαση τμήματος της στοίβας μετά την επιστροφή της αντίστοιχης συνάρτησης
3. Παράλειψη **αποδέσμευσης** μνήμης (όταν πλέον δεν χρειάζεται) – memory leaks

- Δυστυχώς, **δεν** προκαλούν (πάντα) άμεσο σφάλμα
  - το πρόγραμμα μπορεί να «σκάσει» αργότερα, σε ανύποπτο (και φαινομενικά εντελώς άσχετο) σημείο στον κώδικα
- Μάθετε να προγραμματίζετε με **αυστηρό** τρόπο
  - χωρίς να «ποντάρετε» σε sanitization και debugging

# Εξάσκηση (για το σπίτι)

- Υλοποιήστε την συνάρτηση

  ```
  char *strdup2(const char *str);
  ```

  στο πνεύμα της «κανονικής» `strdup`

- Δημιουργεί ένα **αντίγραφο** του string που δίνεται ως παράμετρος, χρησιμοποιώντας **δυναμική** μνήμη, και επιστρέφει ένα δείκτη στο τμήμα της μνήμης που δεσμεύθηκε – βλέπε manual

- Ο «πελάτης» της συνάρτησης έχει την υποχρέωση να **απελευθερώσει** την μνήμη που δεσμεύθηκε
  - όταν δεν χρειάζεται άλλο το αντίγραφο

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

char *strdup2(const char *str) {
 /* add your implementation here */
}

/* test the above function */

int main(int argc, char *argv[]) {
  char *str; int i;

  for (i = 0; i < argc; i++) {
    str = strdup2(argv[i]);
    printf("%s\n", str);
    free(str);
  }

  return(0);
}
```