CSE 340 Spring 2018 Homework 4 Solution

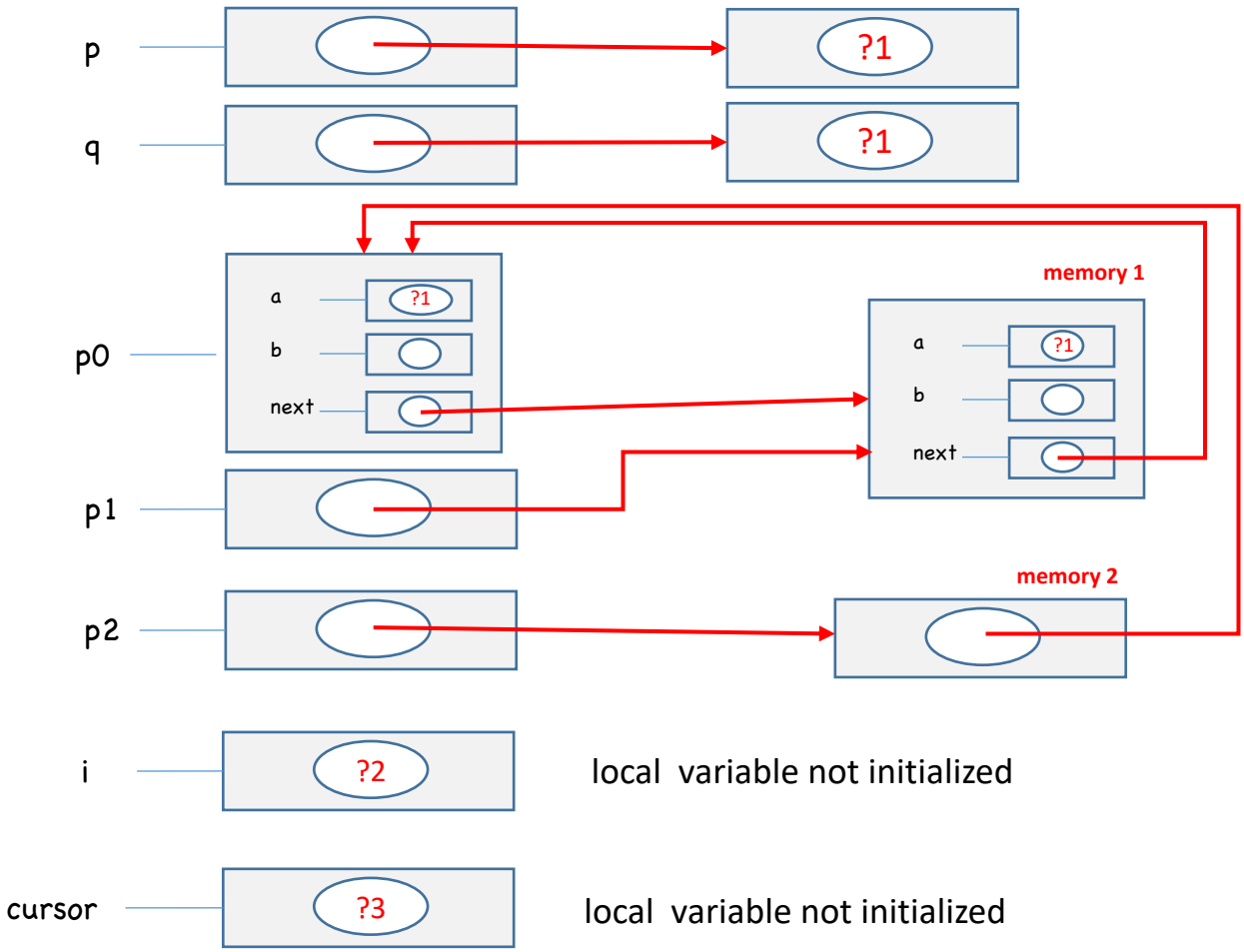
II Pointer semantics.

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

struct T {
              int a;
              int *b;
              struct T *next;
} ;

struct T p0;
struct T *p1;
struct T **p2;

int *p;
int *q;
```

... continued on next page ...

II Pointer semantics.

```
int main()
{
            int i;
            struct T* cursor;

            p1 = (struct T *) malloc(sizeof(struct T));
            p2 = (struct T **) malloc(sizeof(struct T *));

            p = (int *) malloc(sizeof(int));
            q = (int *) malloc(sizeof(int));

            *p = *q;

            p0.next = p1;
            (*p1).a = *p;
            (*p1).next = &p0;
            *p2 = (*p1).next;

   //-------------------------------------------------------------------
   // Question1. Draw a box-circle diagram for p,q, p0, p1, p2 and the
   //            memory allocated above at this point
   //-------------------------------------------------------------------
```

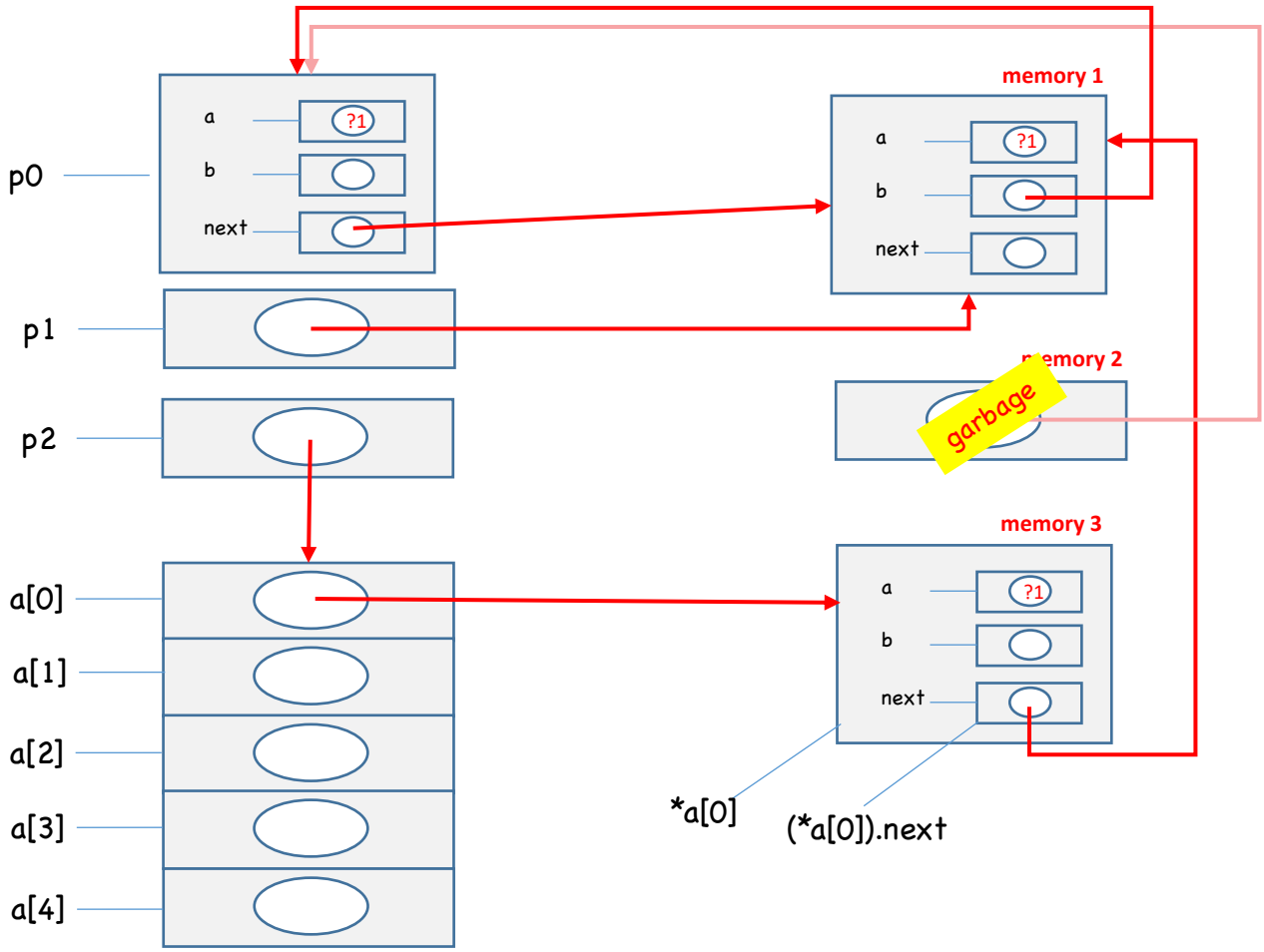


Note: I use ?x to indicate unknown values. ?1 and ?2 can be different or the same, but all ?1 are the same.

Note: You do not need to show all variables in your solution. Only the memory allocated with malloc() and the specified variables

II Pointer semantics.

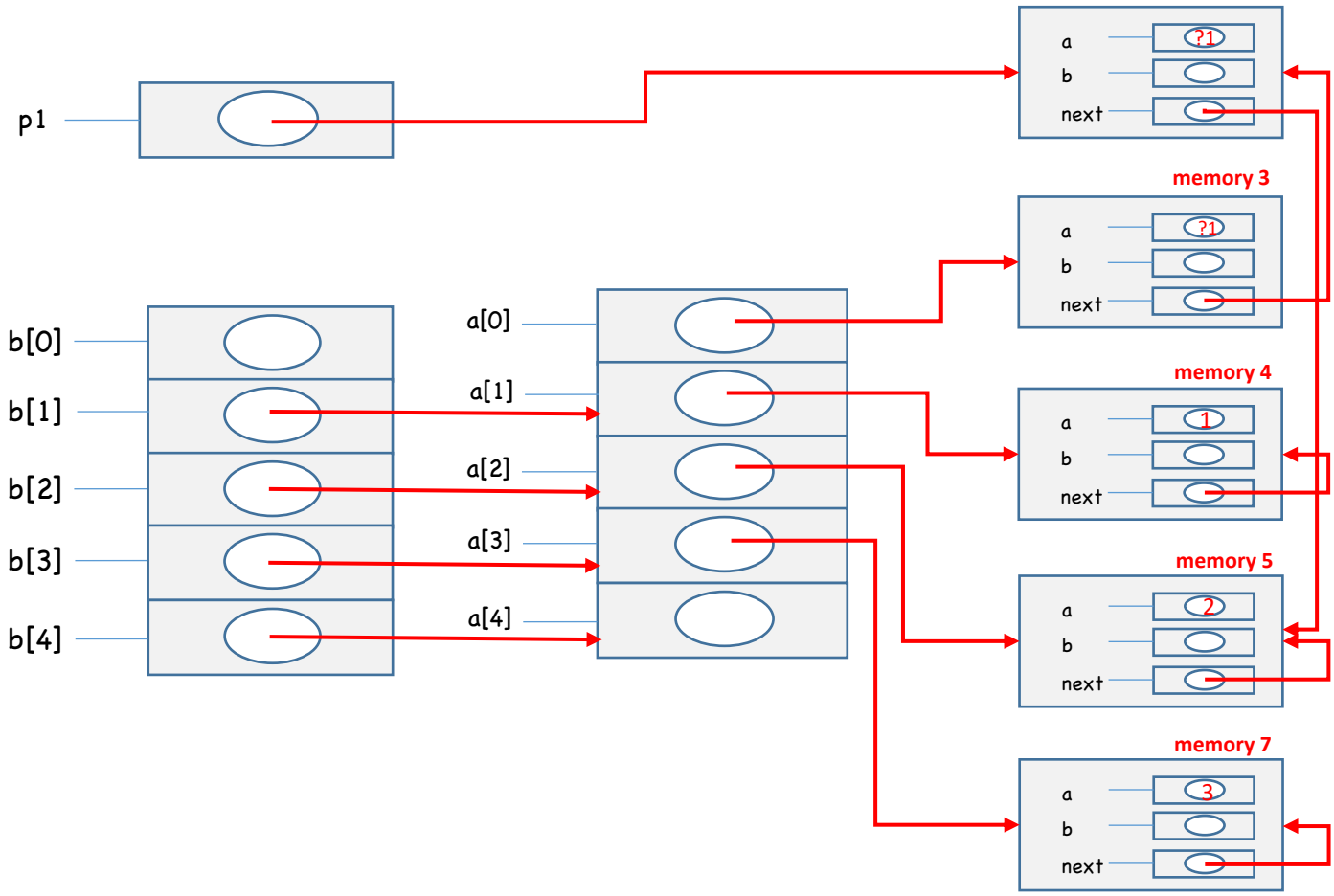
```
{
            struct T *a[5];
            struct T **b[5];
            int i,j;

            a[0] = (struct T*) malloc(sizeof(struct T));
            p2 = &(a[0]);
            (*a[0]).next = p1;

   //-------------------------------------------------------------
   // Question 2. Draw a box-circle diagram for p1, p2, a[]
   //             and the memory allocated above at this point
   //-------------------------------------------------------------
```



Note a has no location associated with it. We cannot say a = ... but in C the expressions &a is the same as &a[0] and *a the same as a[0]

```
for (i = 1; i < 4; i++)
{
            a[i] = (struct T*) malloc(sizeof(struct T));
            (*a[i]).next = a[i-1];
            (*a[i]).a = i;
            b[i] = &a[i];              // b[i] to point to a[i]
            (**b[i]).next = *b[i];     // *b[i] is an alias of a[i];
                                       // (**b[i]).next = a[i] which points to the
                                       // node containing (**b[i]).next ;
}
(*p1).next = a[2]; // (*p1).next will have the address of memory 5
```

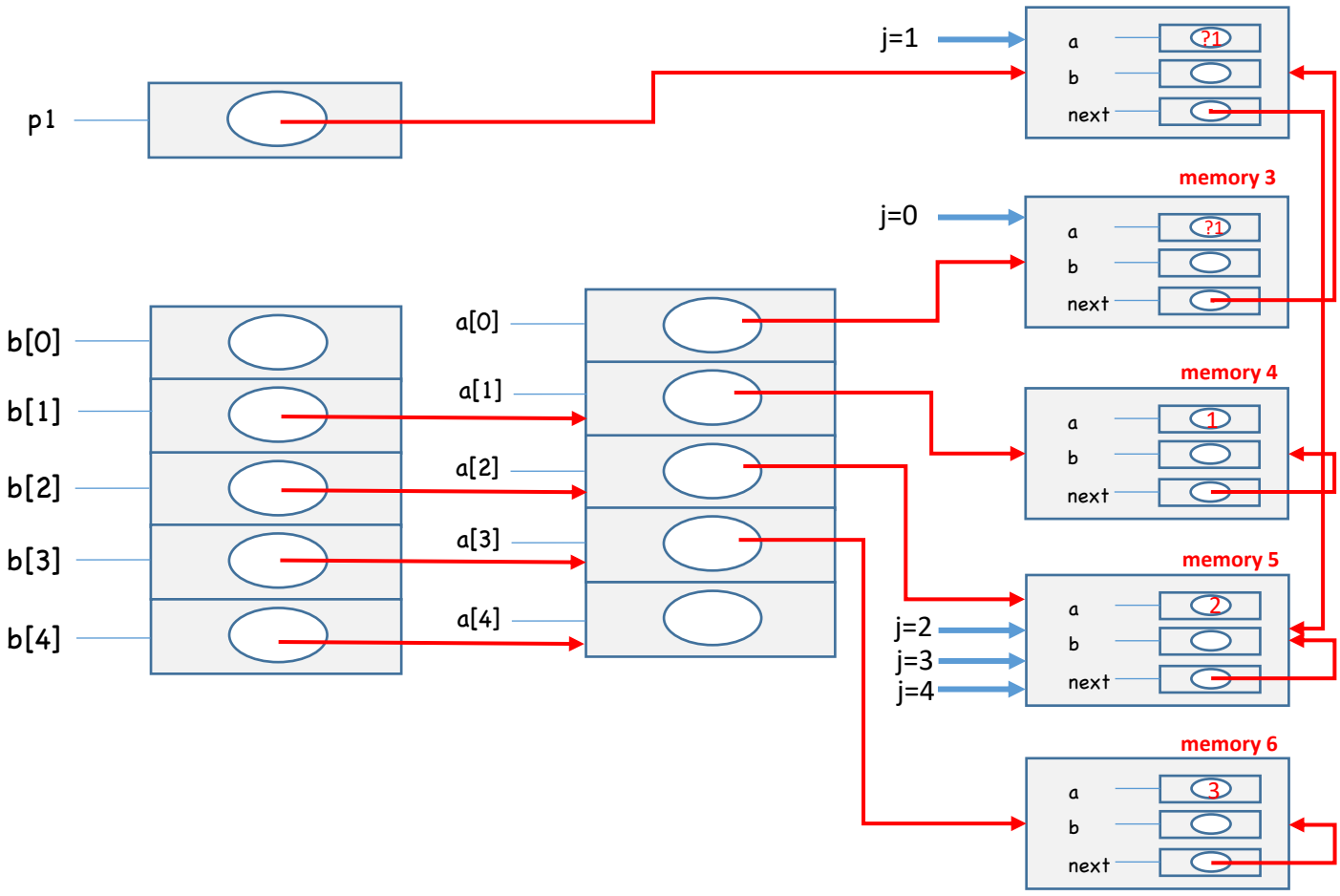
After executing the code above, we get the following



II Pointer semantics.

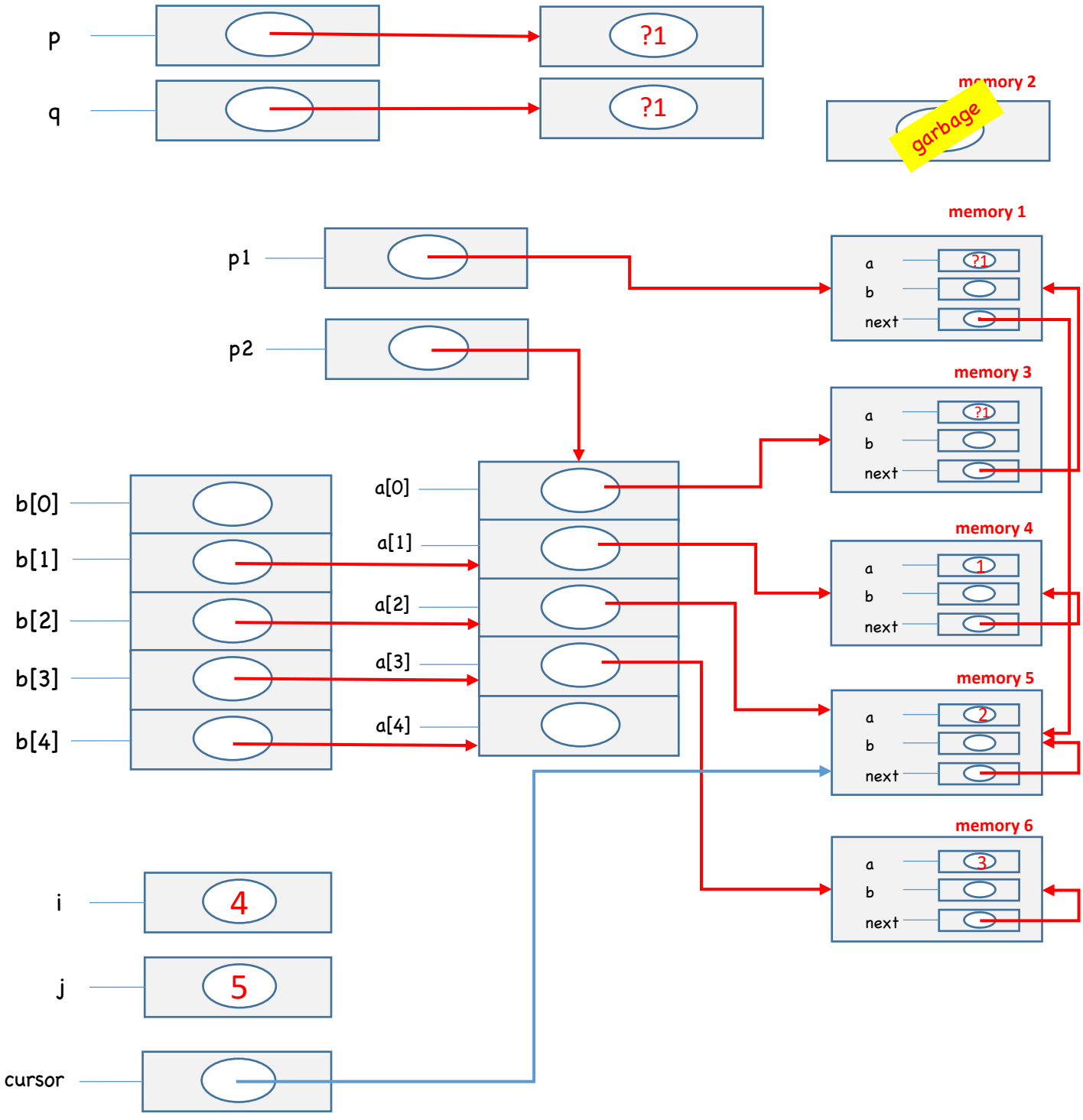
```
//-------------------------------------------------------------------
// Question 3. Explain the output produced by the
//             following loop
//-------------------------------------------------------------------
cursor = a[0];
for (j = 0; j < 5; j++)
{
             printf("%d  ",(*cursor).a);
            cursor = (*cursor).next;
}
printf("\n");
```

The following illustrate the value of cursor (blue arrow) for the successive values of j.
It also explains the values being printed which are ?1 ?1 2 2 2
In a particular execution ?1 might be 0, but in general it need not be 0



II Pointer semantics.

```
//-------------------------------------------------------------------
// Question 4. Draw a box-circle diagram of ALL the variables
//             and the memory allocated up to this point?
//                        what locations are garbage at this point?
//-------------------------------------------------------------------
```



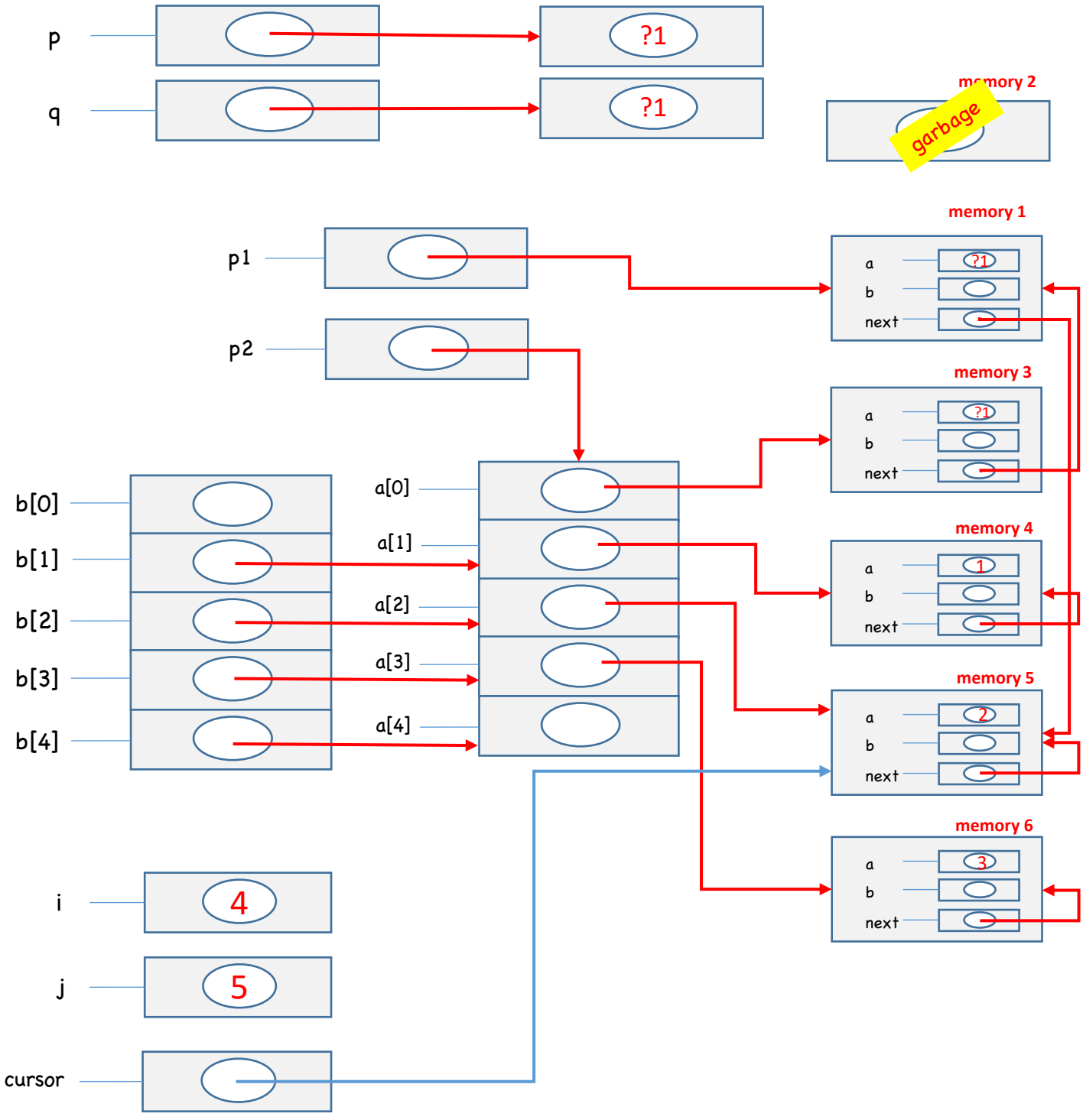
II Pointer semantics.

```
//--------------------------------------------------------------------
// Question 5. Give an alias for a[2] at this point.
//             This alias should not include the names "a" !!
//             This might be tricky and you might want to skip
//             it and do other parts first.
//--------------------------------------------------------------------
```

Answer: *(p2+2) or p2[2]

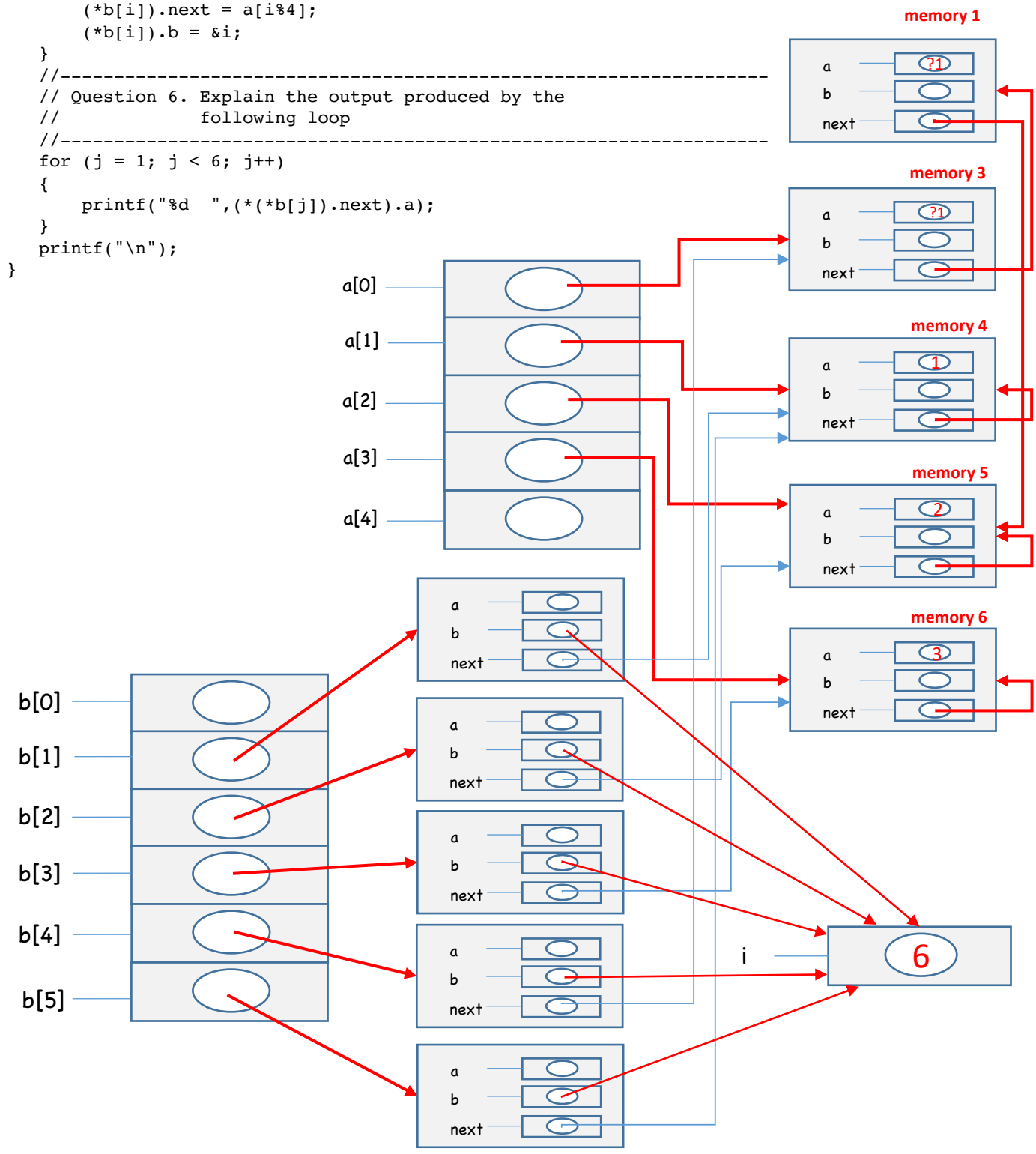
II Pointer semantics.

```
//-------------------------------------------------------------------
// Question 4. Draw a box-circle diagram of ALL the variables
//             and the memory allocated up to this point?
//                        what locations are garbage at this point?
//-------------------------------------------------------------------
```



II Pointer semantics.

```
{  struct T *b[6];
   int j;
   for (i = 1; i < 6; i++)
   {
       b[i] = (struct T*) malloc(sizeof(struct T));
       (*b[i]).next = a[i%4];
       (*b[i]).b = &i;
   }
   //-------------------------------------------------------------------
   // Question 6. Explain the output produced by the
   //             following loop
   //-------------------------------------------------------------------
   for (j = 1; j < 6; j++)
   {
       printf("%d  ",(*(*b[j]).next).a);
   }
   printf("\n");
}
```



The output is 1 2 3 0 1 which is *a[1].a *a[2].a *a[3].a *a[0].a , and *a[1].a
The value 0 just happens to be in the field *a[0].a. The other values have been set by the program

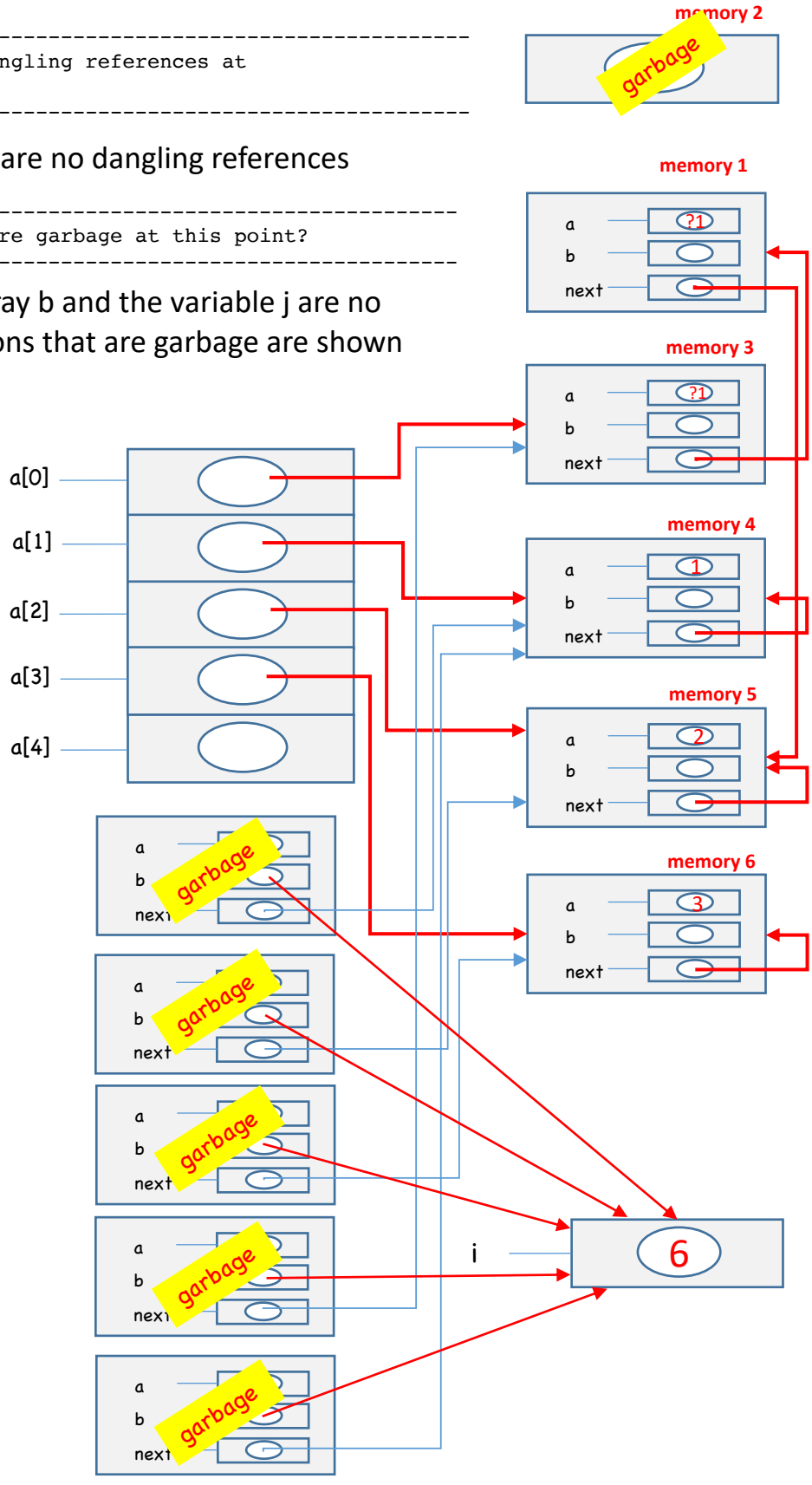
II Pointer semantics.

```
//-------------------------------------------------------------------------
// Question 7. What are the dangling references at
//             this point?
//-------------------------------------------------------------------------
```

Answer: at this point, there are no dangling references

```
//-------------------------------------------------------------------------
// Question 8. What locations are garbage at this point?
//-------------------------------------------------------------------------
```

Answer: at this point, the array b and the variable j are no longer accessible. The locations that are garbage are shown below



II Pointer semantics.

```
//-------------------------------------------------------------------
// Question . What are the dangling references at this point.
 //-------------------------------------------------------------------
```

Answer: at this point, the locations for the array a are deallocated, but the locations are still accessible. p2 is a dangling reference. Even if the location is not deallocated, we cannot make an assumption about how the locations associated with the array a are used after the scope of a is exited, so p2 should be counted as a dangling reference.