**Assignment 1&2 Report**

**Zihan Zhang**

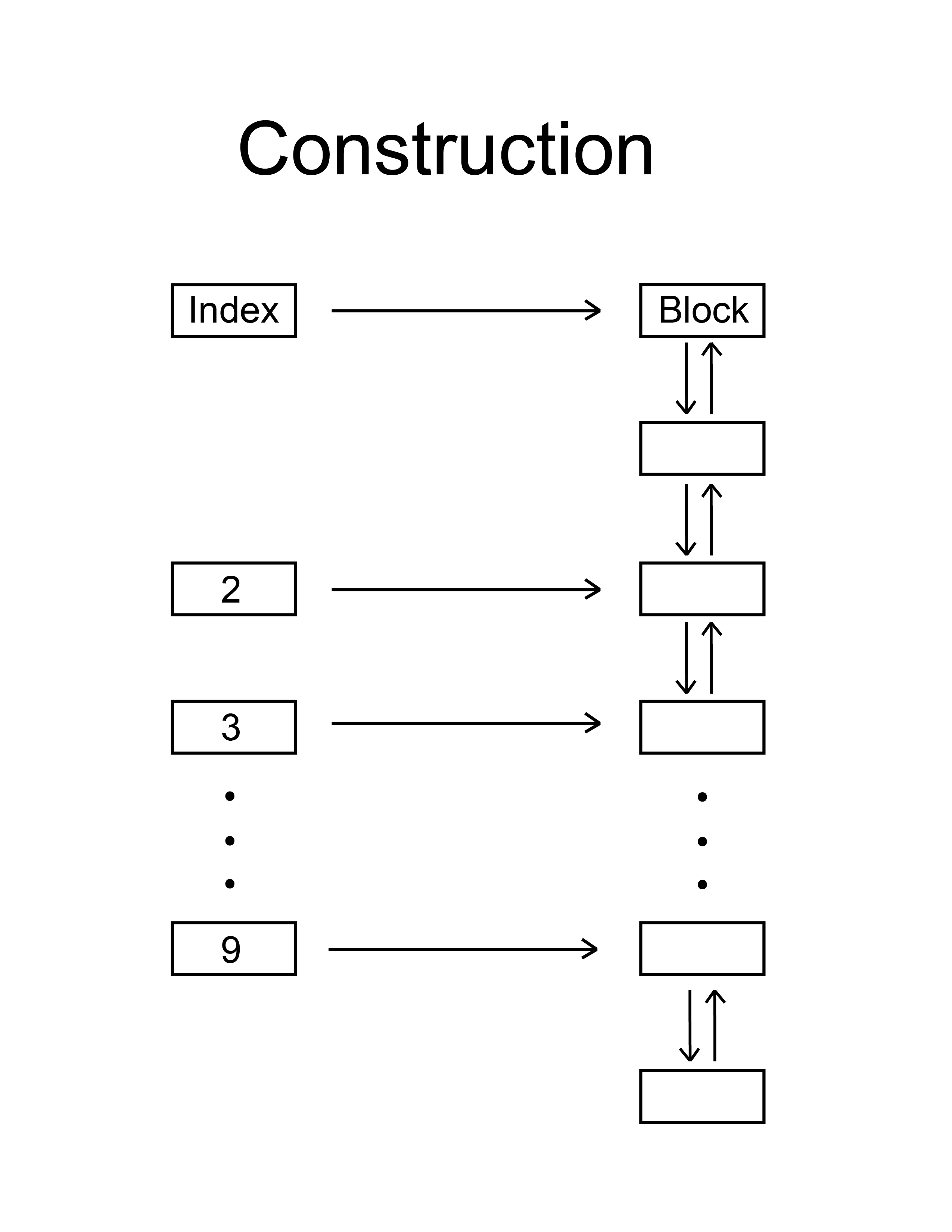
**NUID: 001280965**

1. **Goal:**

design a memory simulator

1. **Design:**
2. **General design:**

There are two elements that construct the whole system. Index array and linked list. Linked list presents all the blocks available to use. From small blocks to large blocks. Index array means the first block of the same size on the linked list. In my code, all the operations(fetch block, return block, merge block, split block) are all been executed on the head of the blocks in same size which is the index array.



1. **Operations:**

**Fundamental operations**

1. Fetch block:

Fetching blocks can be divided into 5 situations

1. The index is null

Go split the large block(clarified in page), if there is no block to be split, merge the small blocks(clarified in page)

1. The index has only 1 block

Delete it from linked list and make it in index array null

1. The index has more than 1 block

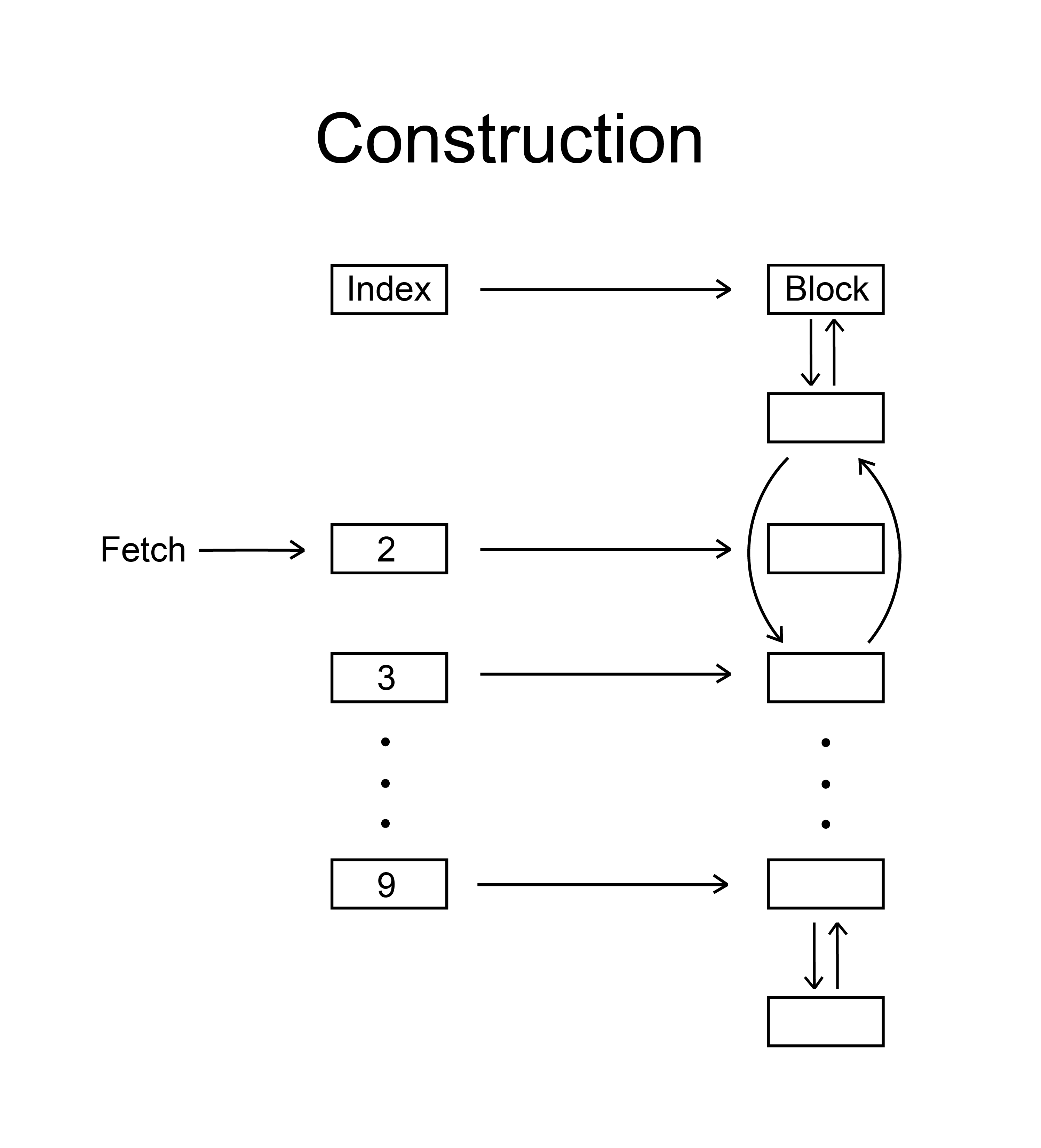
Same as before but differentiate in way to deal with the pointers

1. The index is 0

Same as before but differentiate in way to deal with the trailer pointer

1. The index is 9

Same as before but differentiate in way to deal with the next pointer



1. Return block

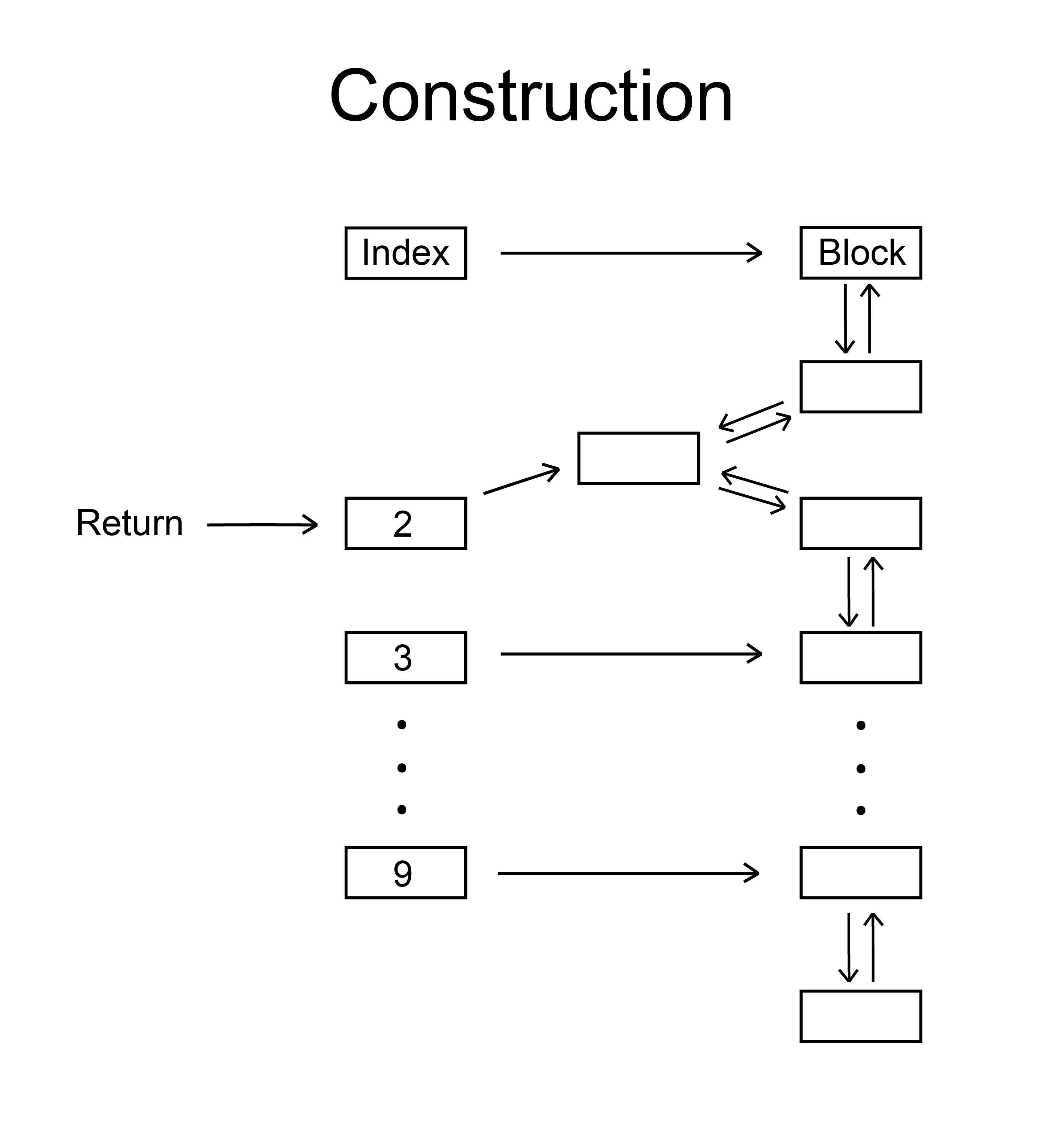
Returning blocks are divided into 2 classes

1. The index returned used to has block

Add a block on linked list and make it new in index array

1. The index used to be null

Same as before but differentiate in way to deal with the pointers



Split block

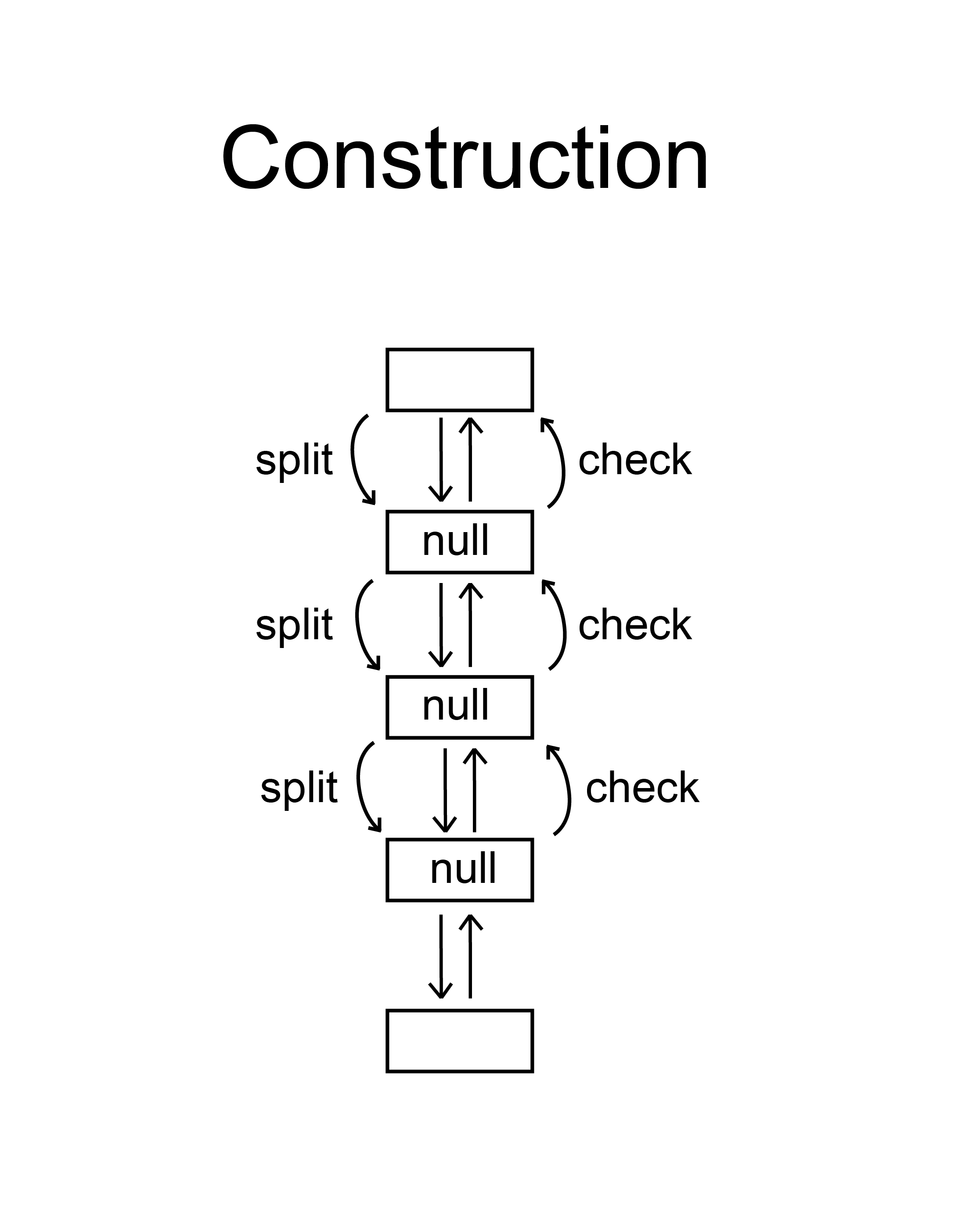
Splitting blocks consist of 3 procedures

1. Fetch the block
2. Return 2 blocks to the (index – 1)

**Advanced operations**

1. Split block

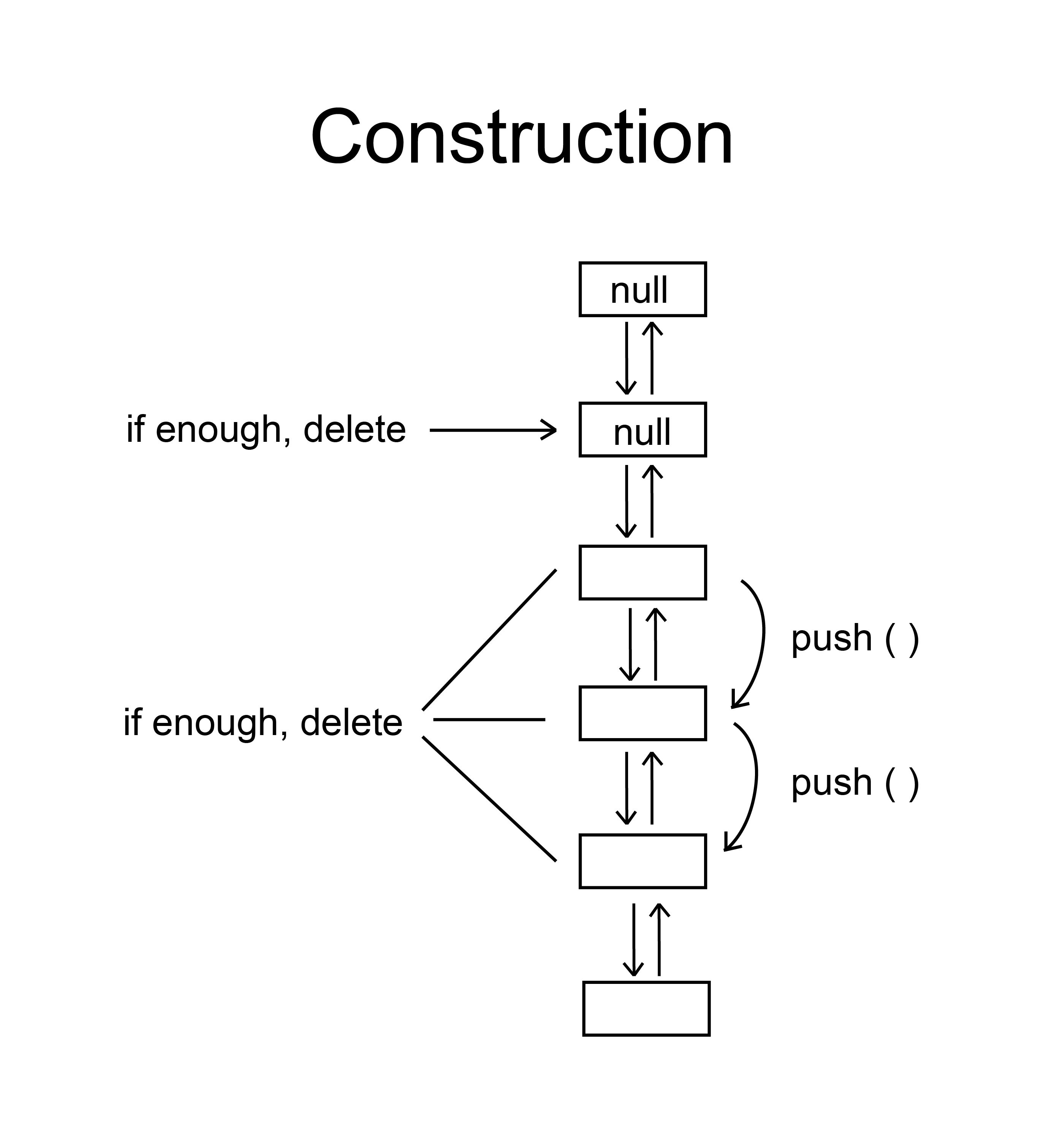
When we want to find a block, but the index is null, we have to split the large block to form small block. What I did is to search the index from the block we want(index, index + 1, index + 2…) and keep recording how many paces we walked. When we finally find an index that is not null, then we split it to form 2 small blocks, then split one of the small block to form 2 smaller block… until we form the block we want.



1. Merge block

Merging block is the opposite operation of the splitting, both the function and the direction. The only time we need to merge block is when we want a block but all the block equal and larger than this block is used. So we need to merge the small blocks into large block to be used. So this operation consists of 4 steps:

1. Find the last index that is not null
2. Go backwards to find the largest block that is smaller than the index we want to merge.
3. Go backwards to collect the blocks and push them into a stack(instead of delete them, because we are not sure whether the total size is enough to form the size we want). Add the size of these blocks and compare it to the size required.
4. When the size is enough, pop all the blocks in the stack and delete(same as fetch) them. Then add(same as return) the block we want originally.



1. **Test method:**

1. To simulate the real operation in memory pool, I design the test class to give out 5 requests which have random sizes a time. After that the test class will return all the blocks fetched, and keep going.

2. To present how fragmental the memory pool is, I used an integer:fragment to present. The fragment of each block is (10 – index) which means the smaller the block is, the more fragment it has. And the total fragment present the total fragment of the memory pool.

1. **Result:**

