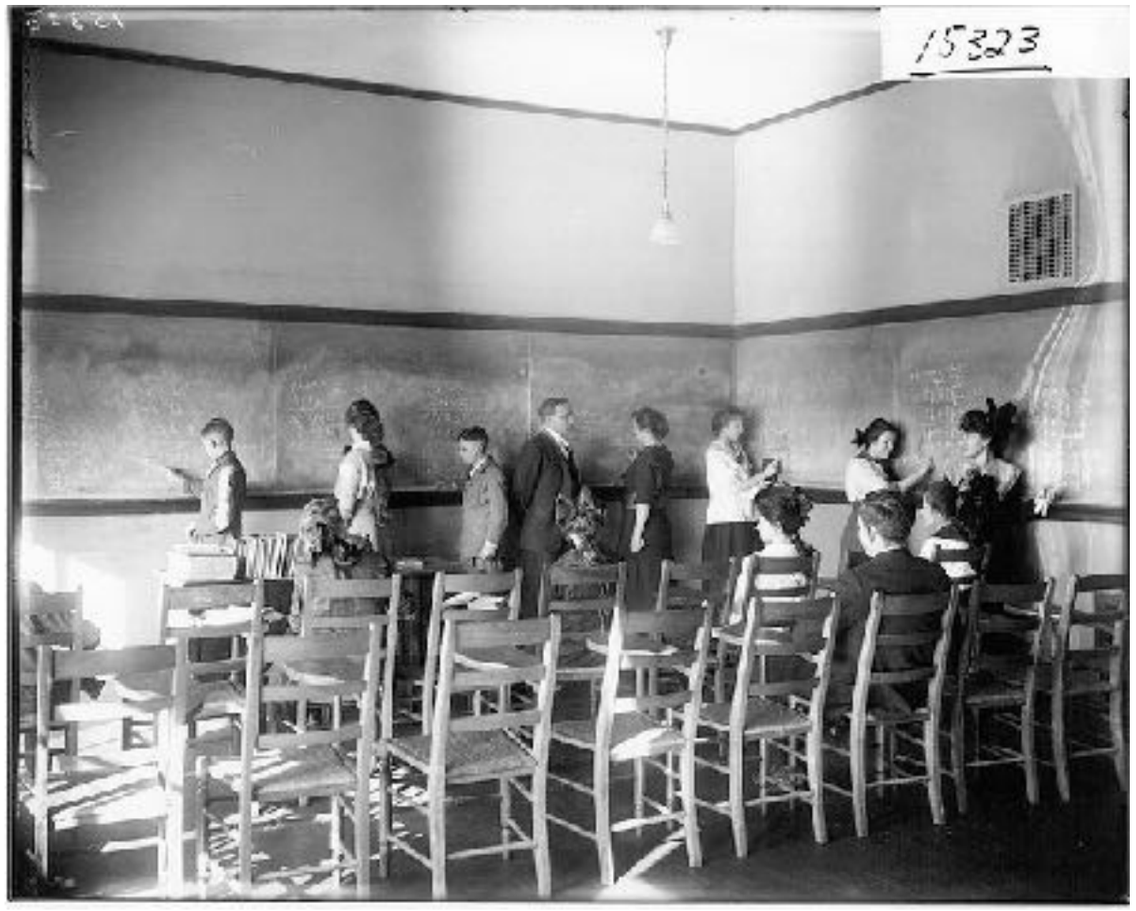
Consultations



Julian García

Tuesday: 10AM to 11AM

Office 230, 25 Exhibition Walk, Clayton

Prepared by Julian García, based on material by
David Albrecht and María García de la Banda

Lecture 6
Arrays in MIPS

FIT 1008
Introduction to Computer Science

Objectives for this lecture

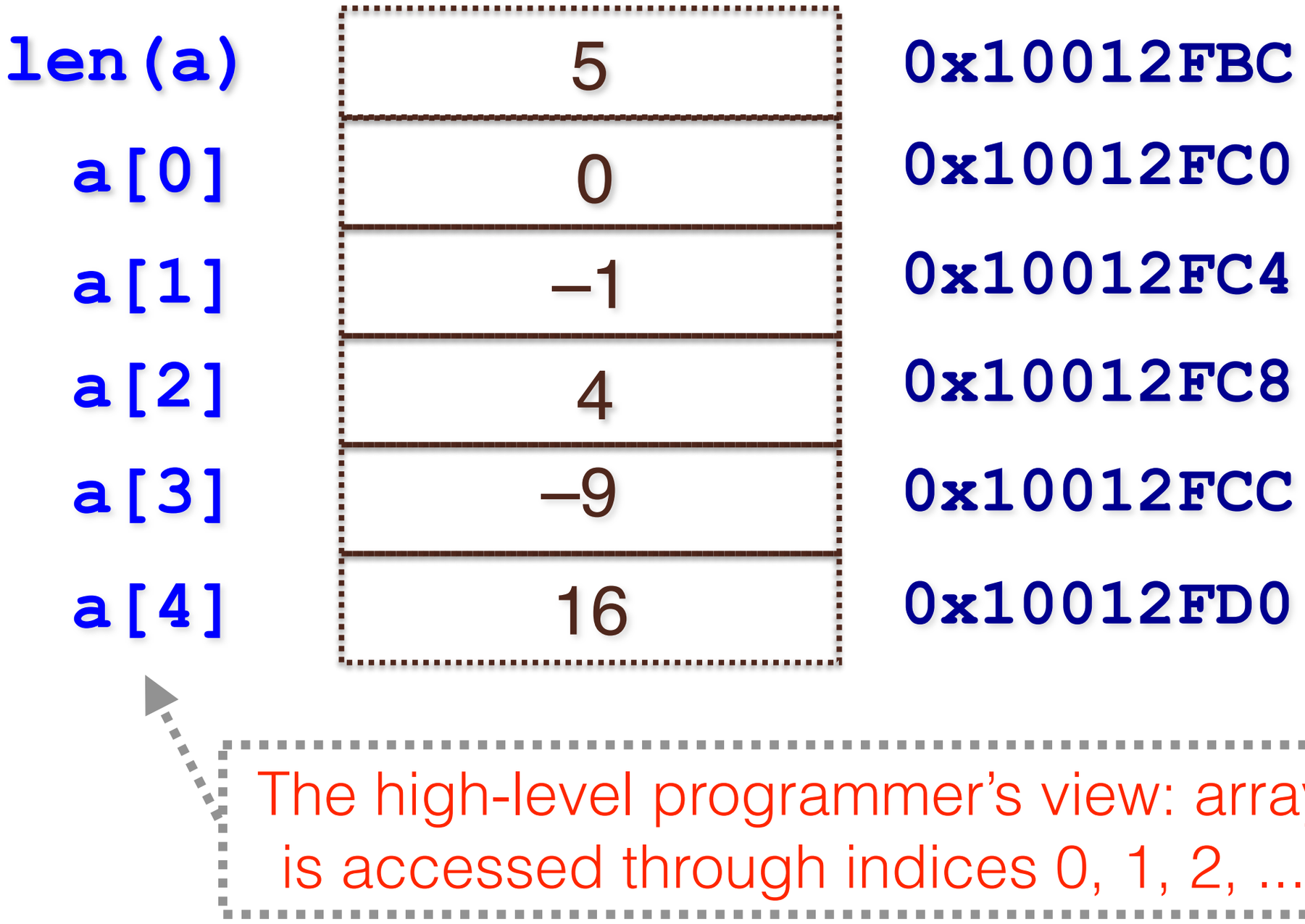
- How to write MIPS programs that involve **arrays**
- The need for **memory diagrams** and how to draw them
- Understand what are **pointers** and how do we use them.
- How to use **addressing modes** to access variables
- How to allocate memory on the **Heap**

Lists vs **Arrays**

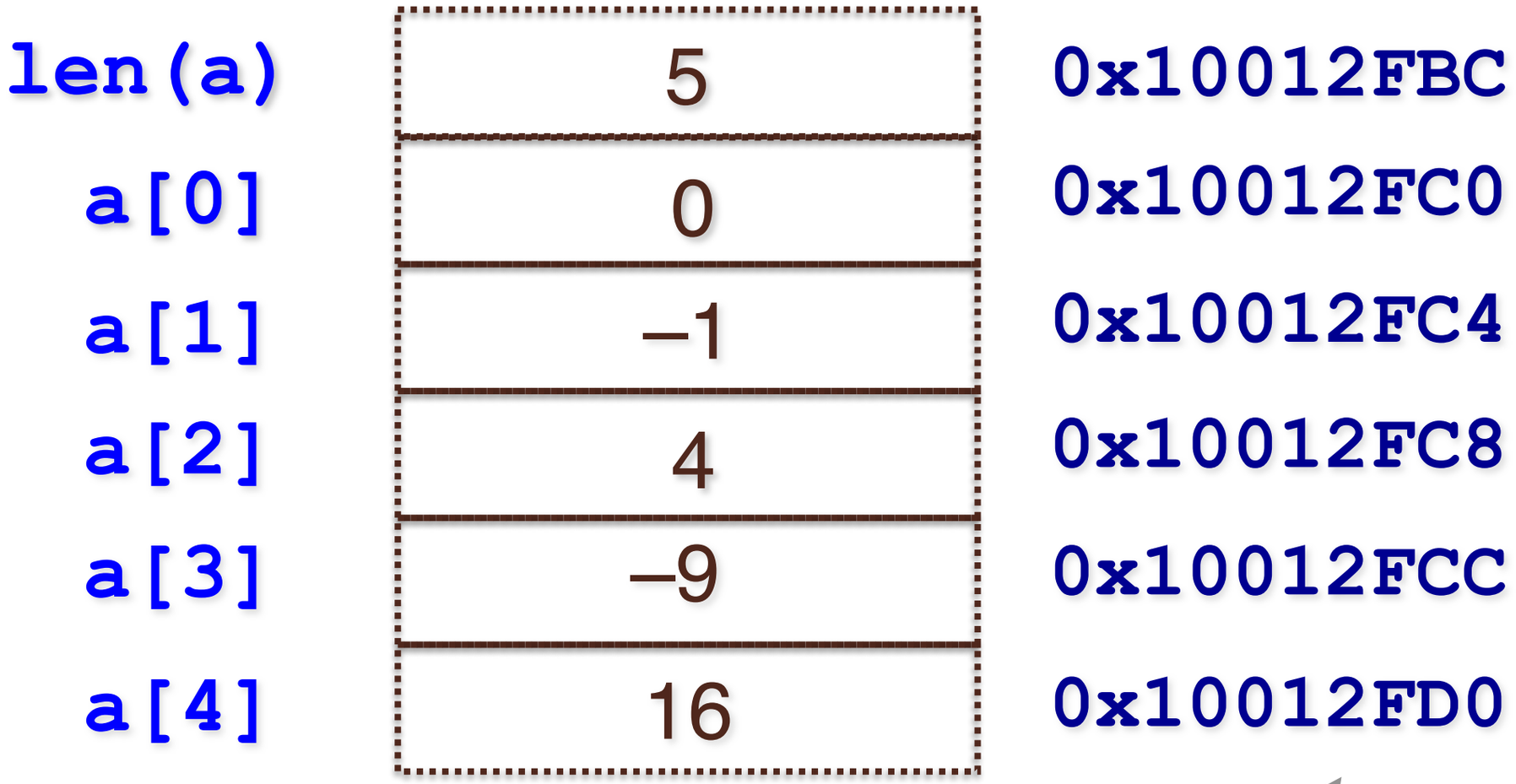
Simple Implementation

- Arrays have a **fixed length**.
- All the items will have the same size.
- All the items in the list are initialised to 0.
- We will **store length** before the items.

Arrays in MIPS

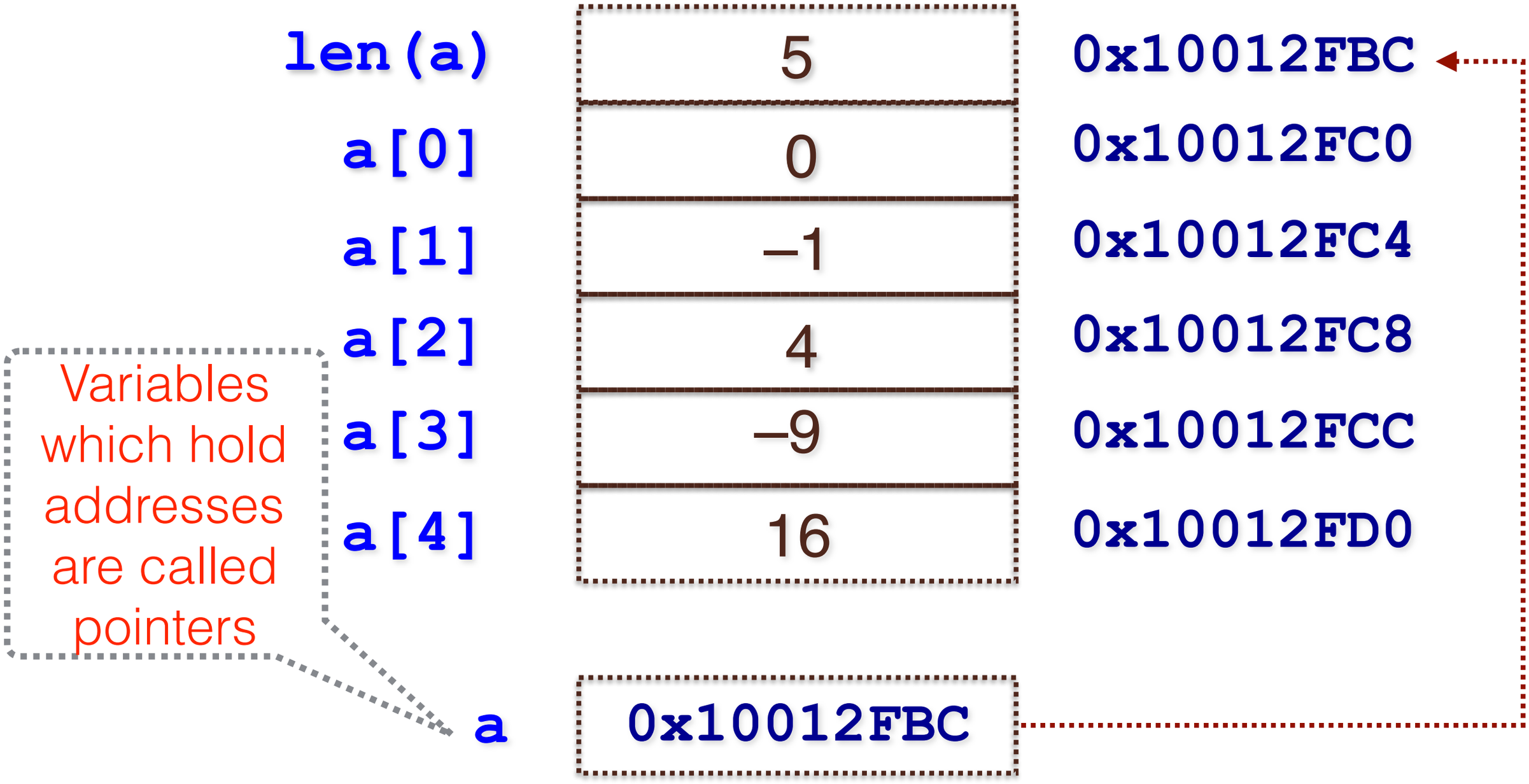


Arrays in MIPS

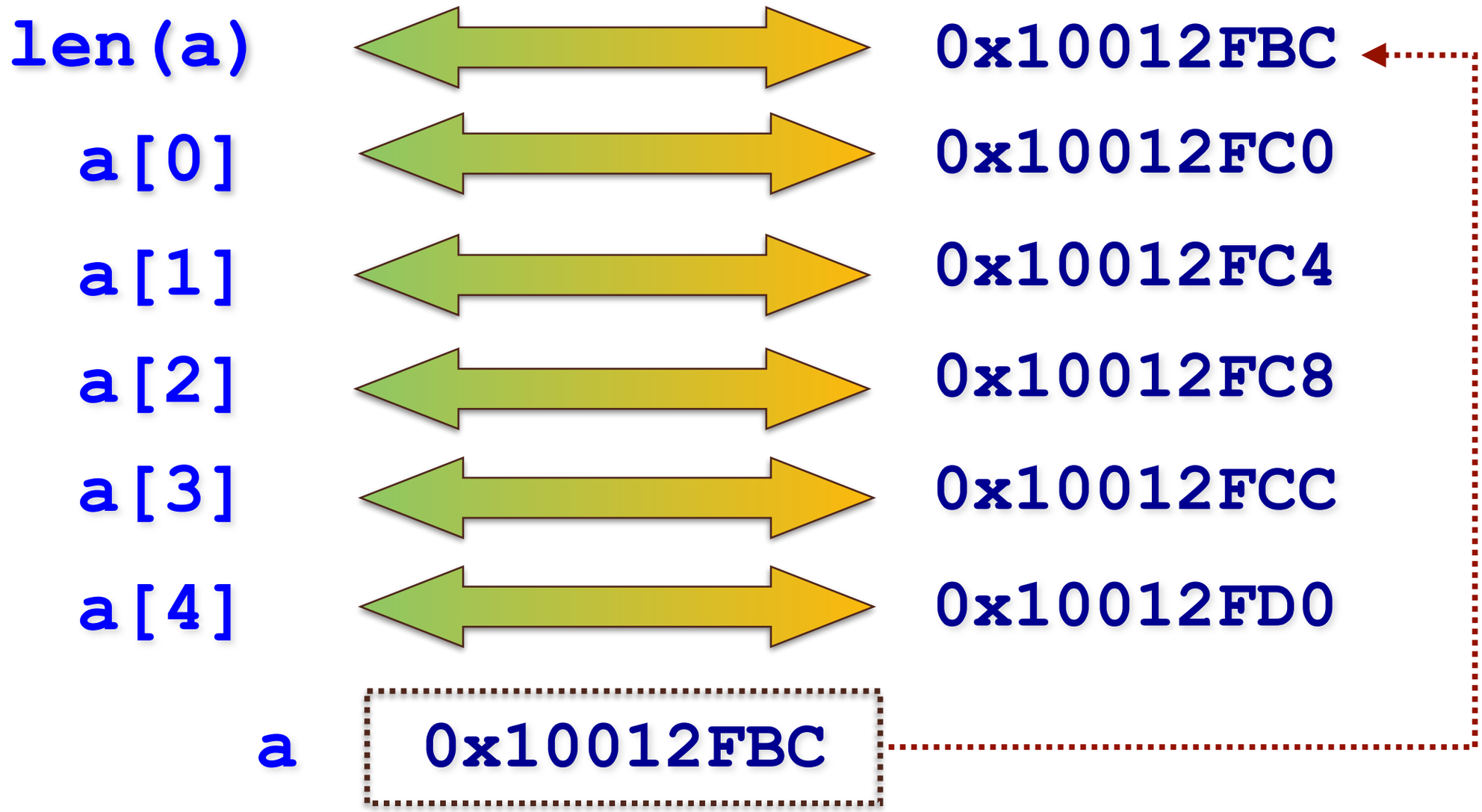


The computer's view: array is part of memory, accessed through addresses 0x10012FC0, 0x10012FC4, ...

Arrays in MIPS

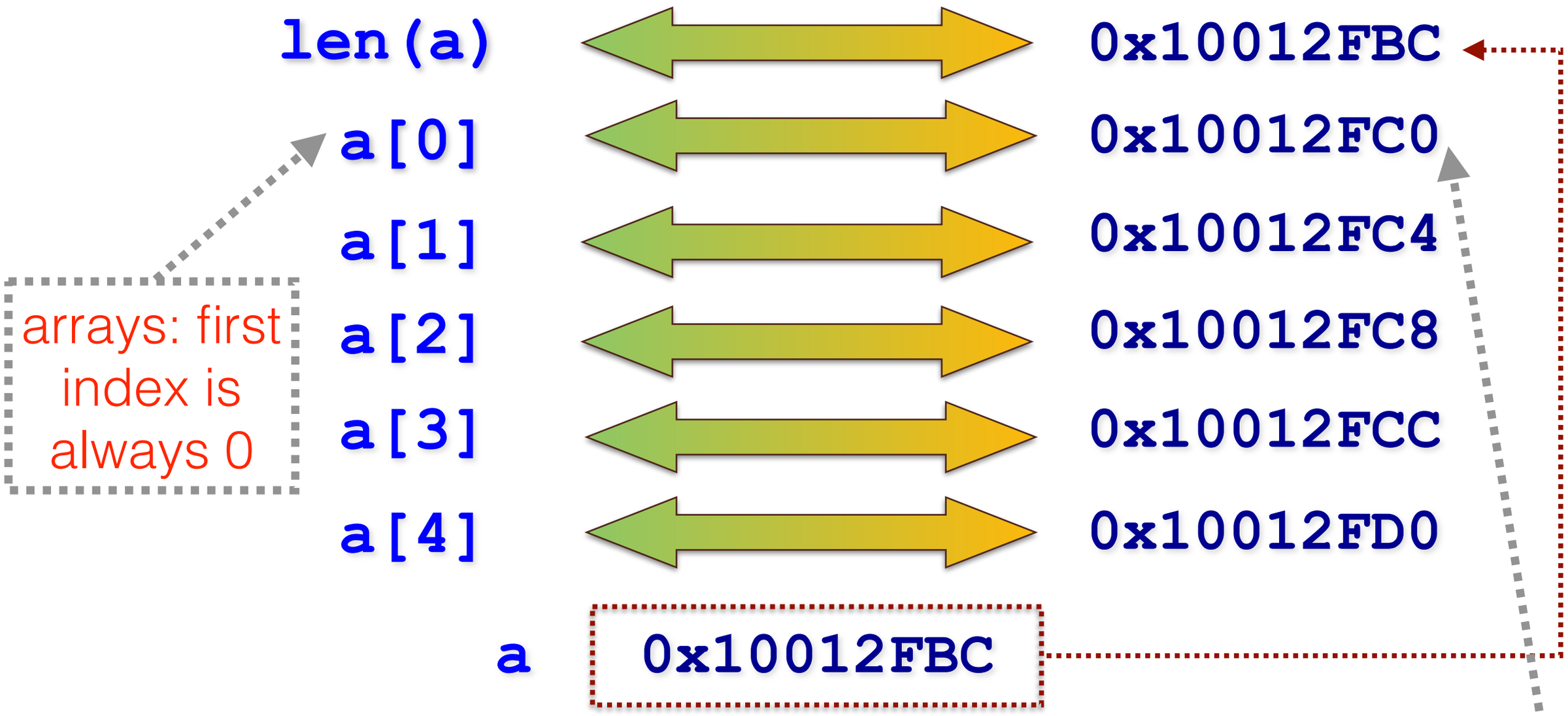


Arrays in MIPS



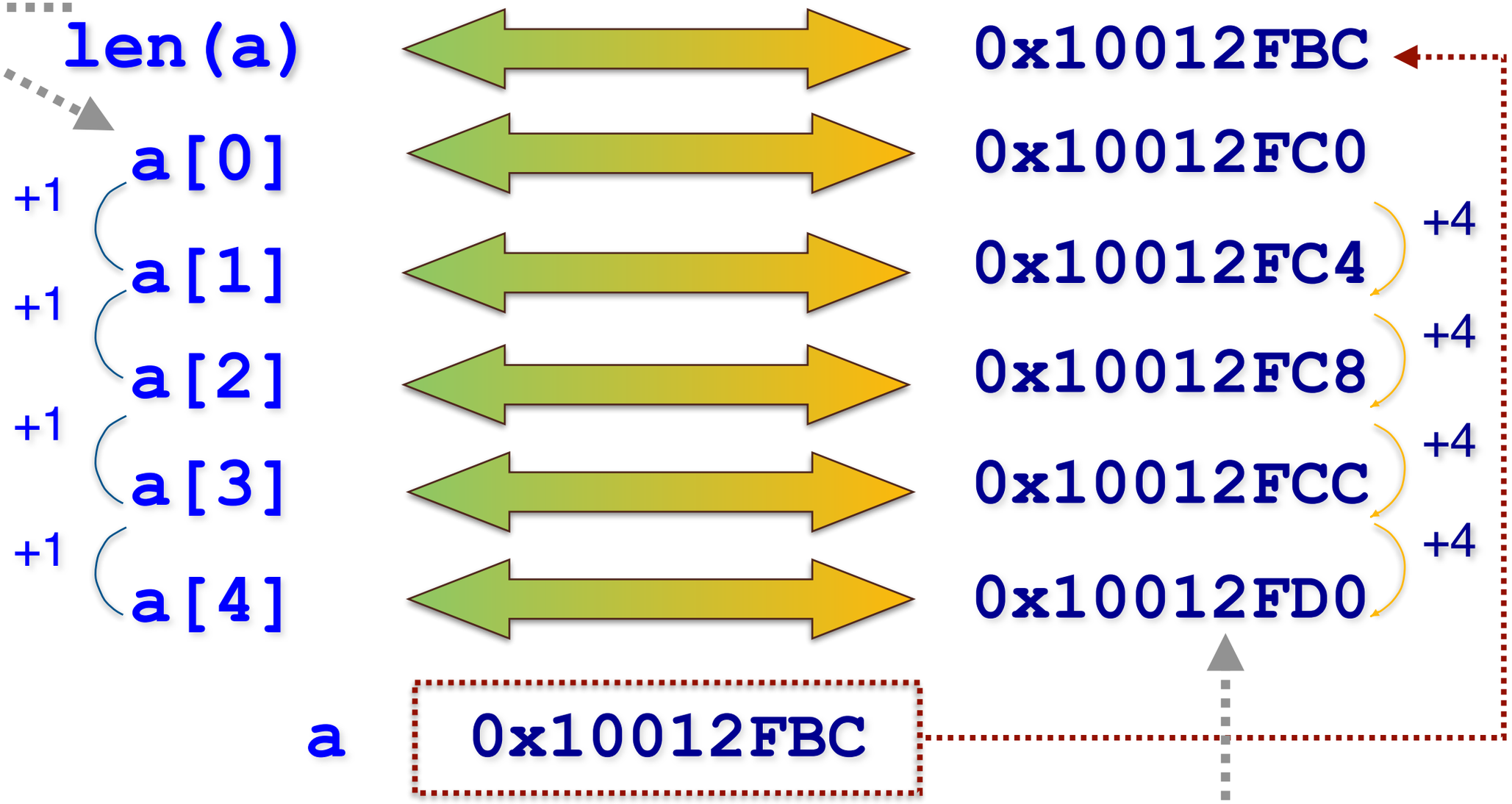
To program arrays in assembly language, need to understand their relationship, and how to convert from one to another.

Arrays in MIPS



Arrays in MIPS

arrays: adjacent indices differ by 1



addresses: addresses differ by size of array element type (here, 4 bytes for **integers**)

Five ways to specify an address

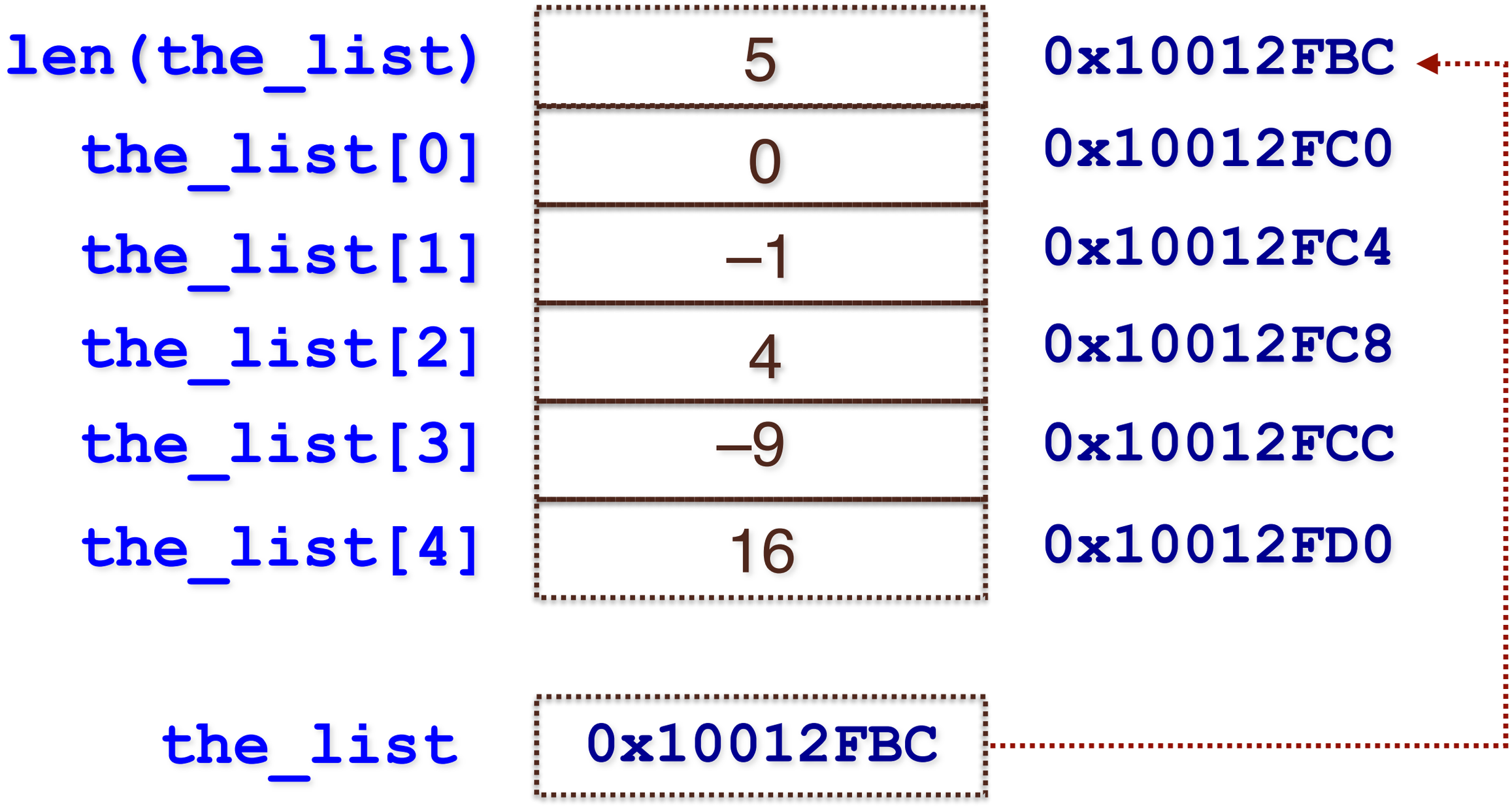
- Directly (or using a label), e.g.
  ```
  lw $t1, N          # loads from label N
  ```
- Label plus offset, e.g.
  ```
  lw $t1, N+4        # loads from (label N + 4)
  ```
- Using a GPR to store the address, e.g.
  ```
  lw $t1, ($s0)  # loads from address stored in $s0
  ```
- GPR + offset, e.g.
  ```
  lw $t1, 4($s0)
  # loads from (address stored in $s0)+4
  ```
- Label, offset, and GPR, e.g.
  ```
  lw $t1, N+4($s0)
  # loads from (label N+4)+contents of $s0
  ```

Creating Arrays in MIPS

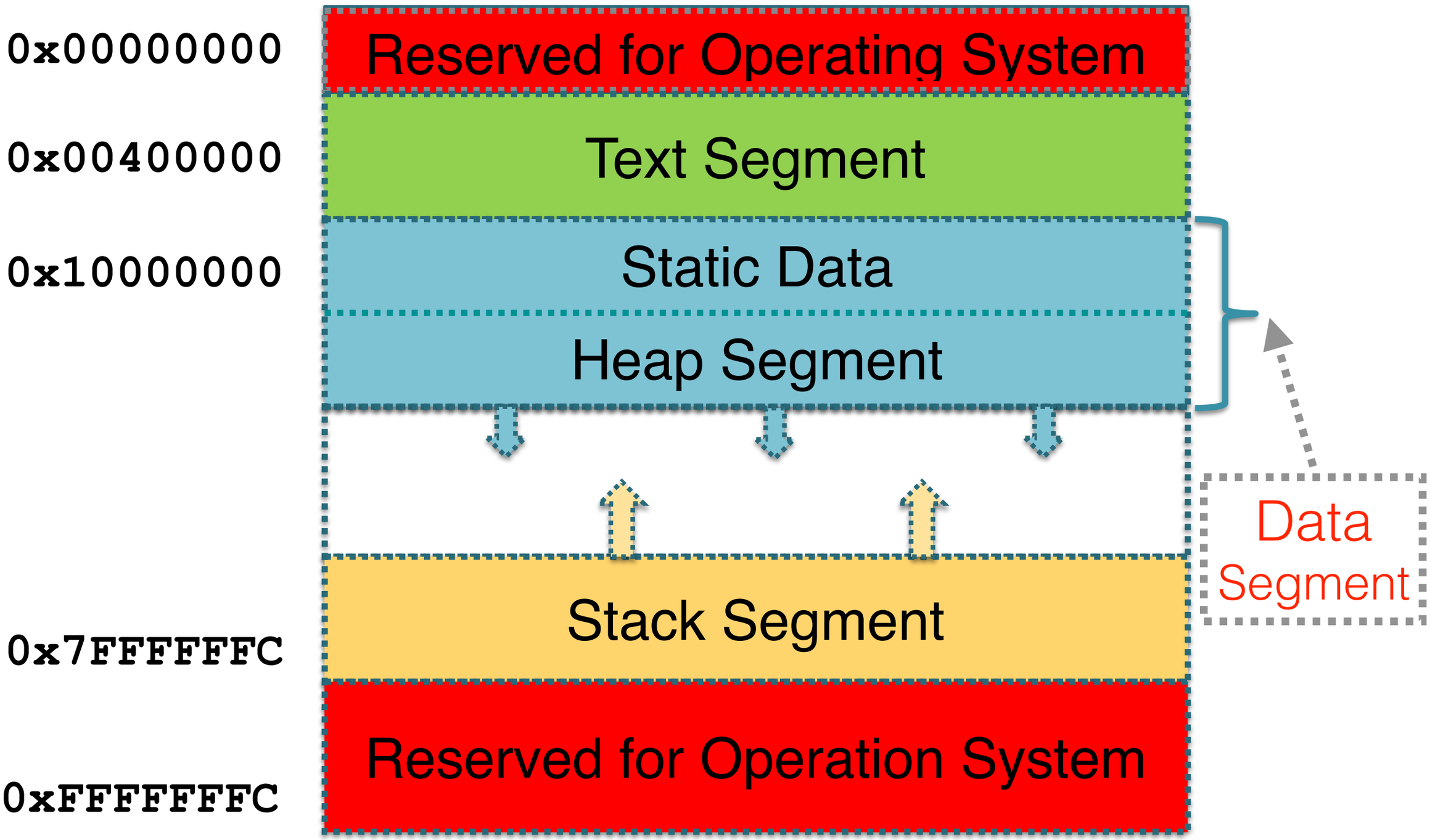
```
the_list = [0]*size
```

- Allocate memory on the Heap for the list together with the length of list
- For integers, space required: `4*size + 4`
- Store the address of the first byte of memory allocated in `the_list`

Arrays in MIPS



MIPS Architecture: Memory



construct_list.py

```
size = int(input("Enter number of values: "))
the_list = [0] * size
for i in range(size):
    the_list[i] = int(input("Value: "))
print(the_list)
```

Creating the list

```
lw      $t0, size
addi $t1, $0, 4
mult      $t1, $t0
mflo      $t2
```

$t0 = size
$t1 = 4
$t2 = 4*size

```
add $a0, $t2, $t1     # $a0 = 4*size + 4
addi      $v0, $0, 9     # $v0 = 9
syscall              # allocate memory
```

allocate 4*size + 4 bytes
(result in $v0)

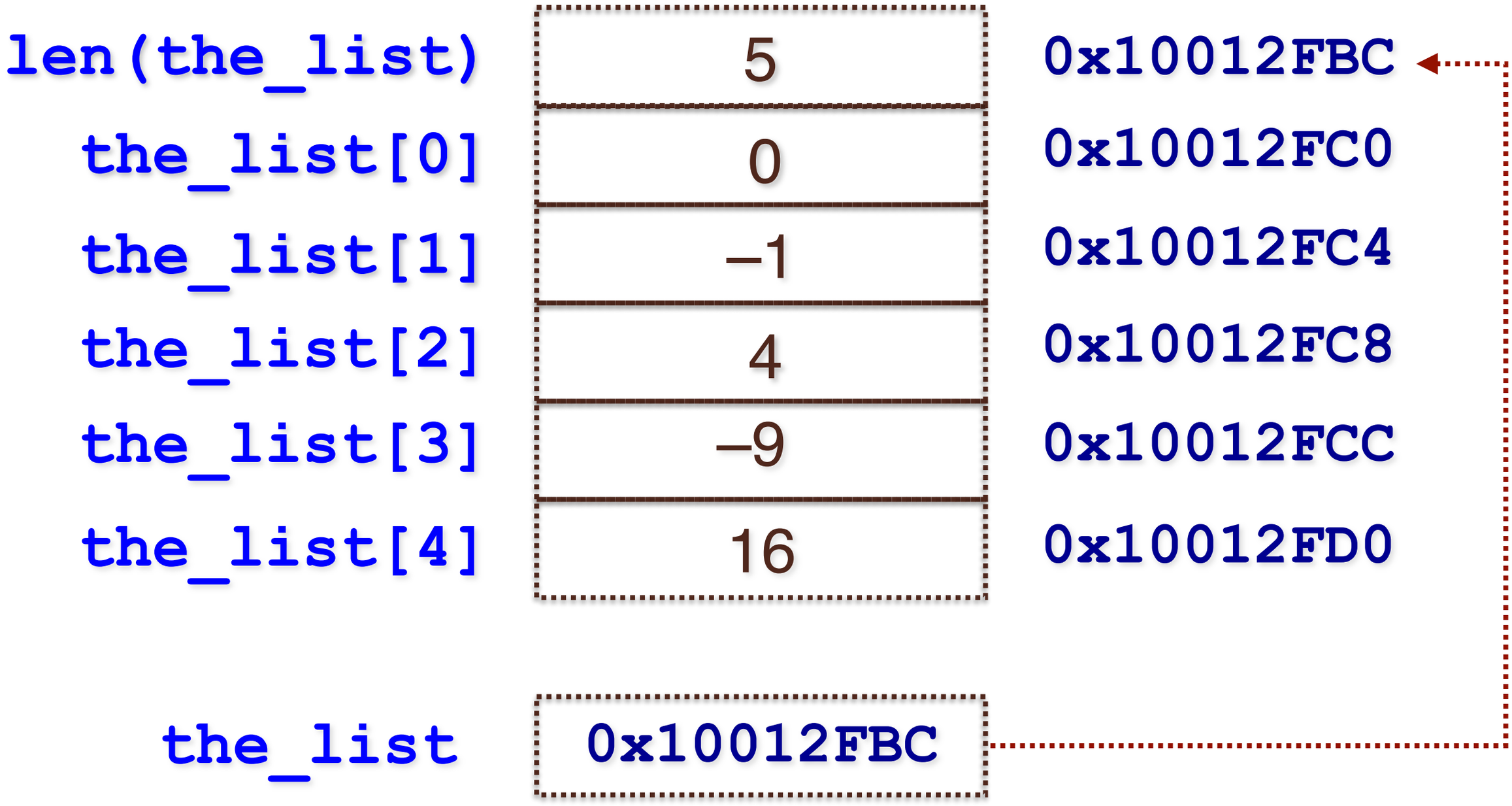
```
sw    $v0, the_list # the_list now points to the returned address
sw  $t0, ($v0)      # store length of list
```

label the_list
references the memory
that will store the size

store the size in the
address referenced by
$t0

Call code ($v0)	Service	Arguments	Returns	Notes
9	Allocate memory	$a0 = number of bytes	$v0 = address of first byte	-

Lists in MIPS

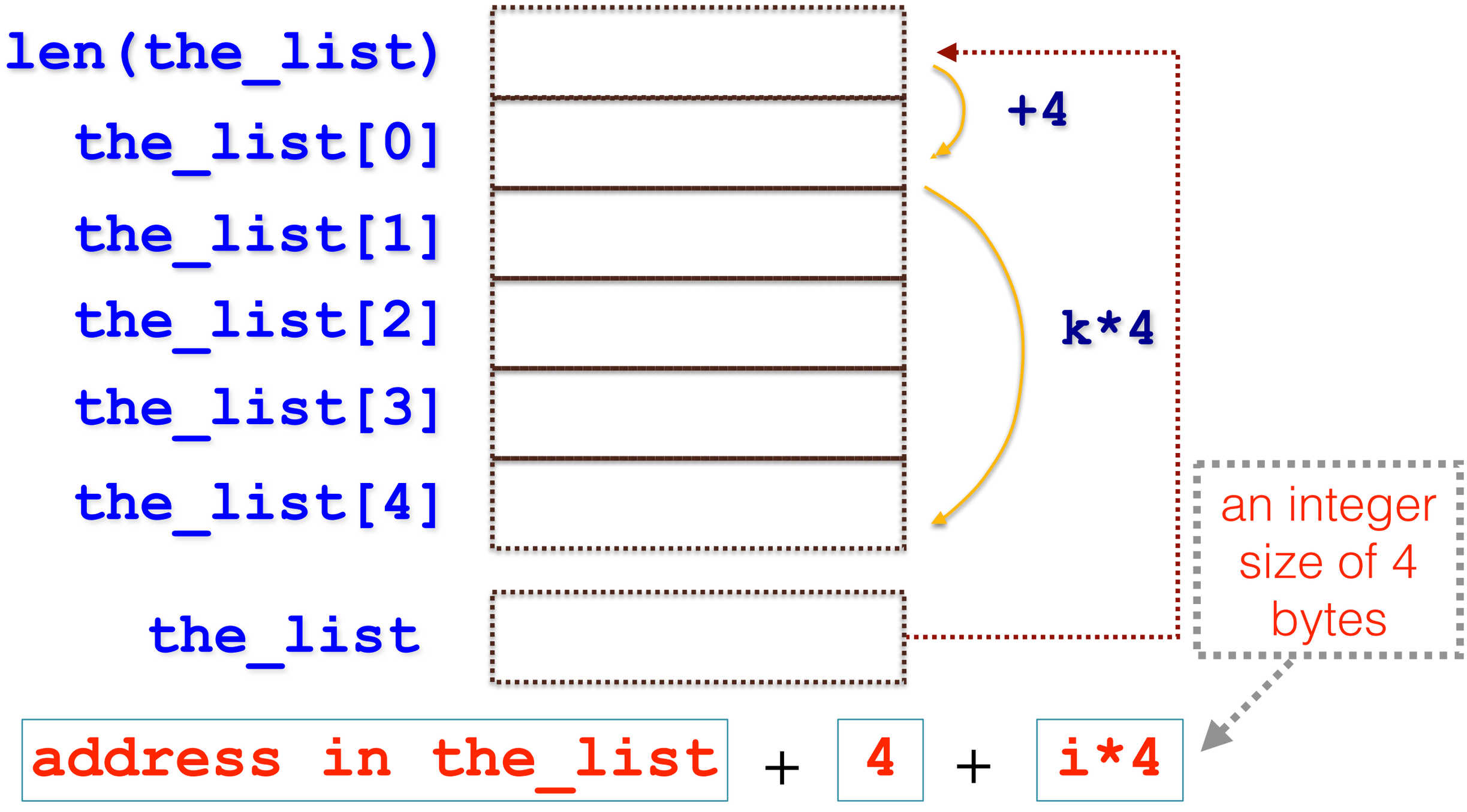


Read in values into the list

```
for i in range(size):

    the_list[i] = int(input("Value: "))
```

Address of **the_list[k]**



Read in values into the list

```
      sw      $0, i             # i = 0
loop: lw      $t0, i
      lw      $t1, size
      # if i >= size goto endloop (Details omitted)
      # print prompt2 (Details omitted)
      # read next item into $v0 (Details omitted)
      lw      $t2, the_list
      addi  $t3, $0, 4
      mult  $t3, $t0
      mflo  $t4
      add    $t4, $t4, $t3 # t4 = i * 4 + 4
      add     $t4, $t4, $t2 # $t4 points to next location in the list
      sw     $v0, ($t4) # store the next value
      addi   $t0, $t0, 1 # i = i + 1
      sw     $t0, i
      j    loop
endloop:
```

Print list

```
print(the_list)
```

Print list

lw: This is correct, because *the_list* is a pointer, its value is an address!

```
            addi $t0, $0, 0   # t0 = 0
loop2:      lw   $t1, the_list # $t1 = address of the_list
      lw    $t2, ($t1)  # $t2 = size of list
      # if $t0 >= size goto endloop2 (Details omitted)
      addi $t3, $0, 4
      mult $t3, $t0
      mflo $t4
      add  $t4, $t4, $t3 # $t4 = $t0 * 4 + 4
      add  $t4, $t4, $t1
      lw   $a0, ($t4) # load current item value into $a0
      addi $v0, $0, 1
      syscall # print current item
      addi $a0, $0, 32 # print a space - ascii code 32
      addi $v0, $0, 11
      syscall
      addi $t0, $t0, 1  # $t0 = $t0 + 1
      j    loop2
endloop2:
```

(using directly $t0 instead of i)

Summary

- How we could represent lists in MIPS
- How to create lists
- How to access items in lists
- How to write MIPS programs involving lists