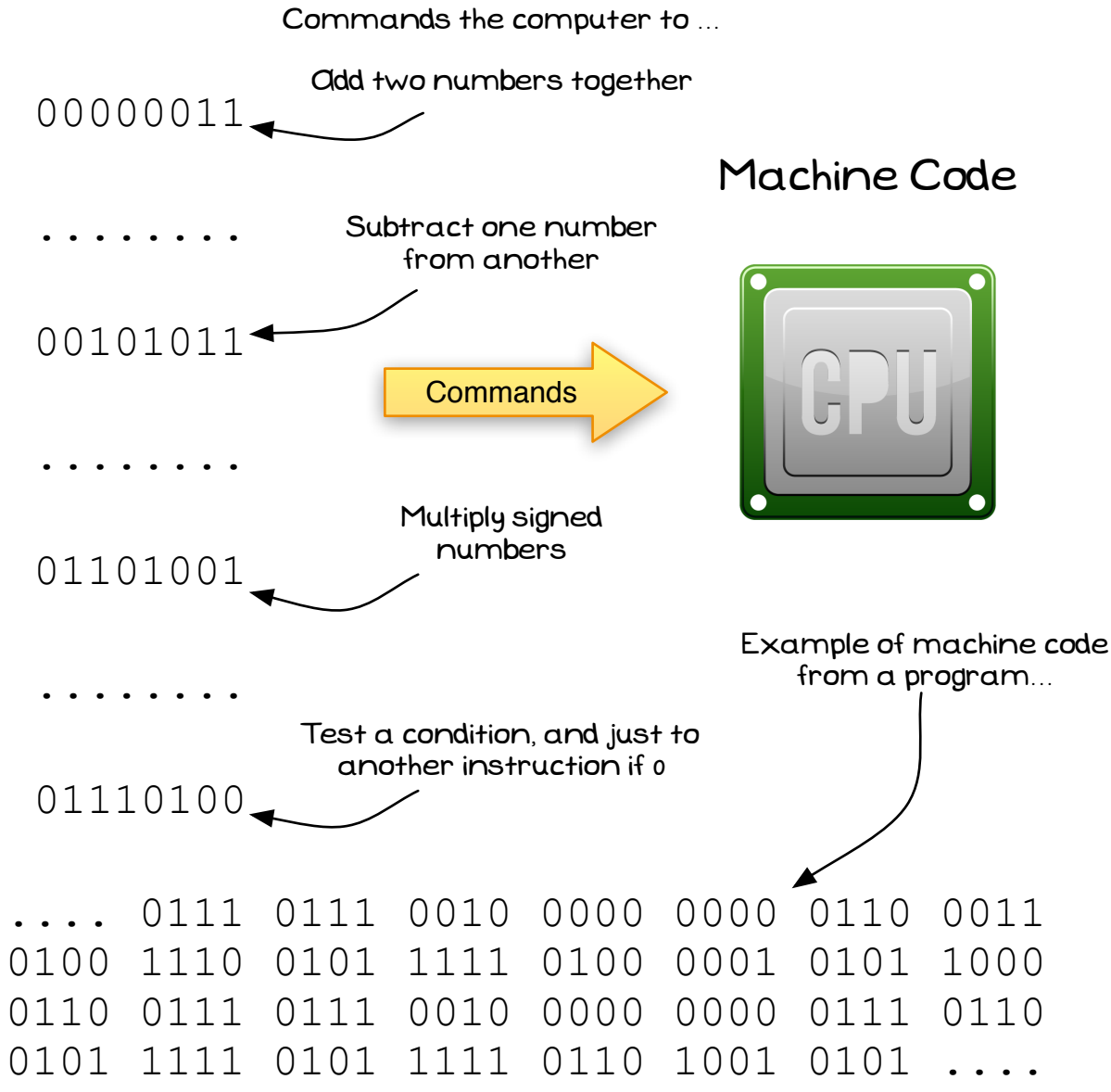
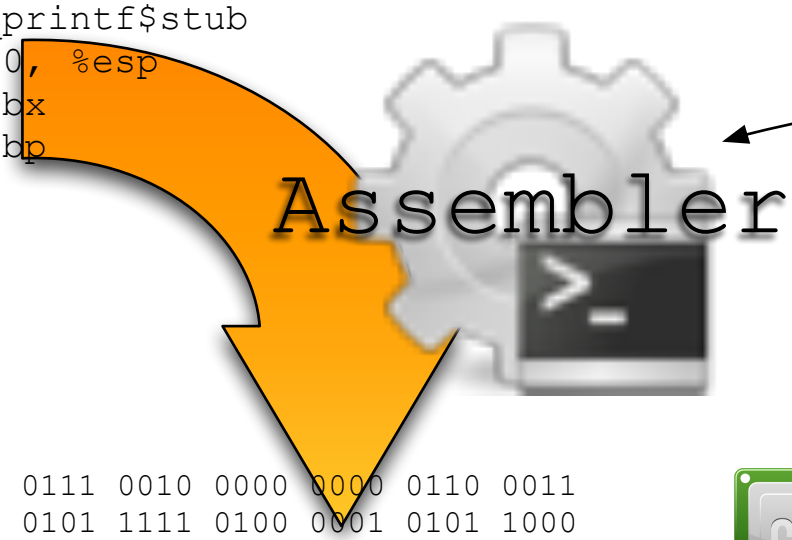

Commands the computer to ...
Add two numbers together
00000011
Machine Code
.....
Subtract one number from another
00101011
Commands
CPU
.....
Multiply signed numbers
01101001
.....
Example of machine code from a program...
Test a condition, and just to another instruction if 0
01110100
.... 0111 0111 0010 0000 0000 0110 0011
0100 1110 0101 1111 0100 0001 0101 1000
0110 0111 0111 0010 0000 0000 0111 0110
0101 1111 0101 1111 0110 1001 0101 ....

Assembly Code

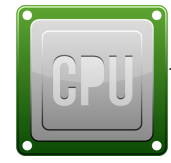
```
  .cstring
LC0:
  .ascii "Hello\0"
  .text
.globl _main
_main:
  pushl   %ebp
  movl %esp, %ebp
  pushl   %ebx
  subl $20, %esp
  call ___i686.get_pc_thunk.bx
"L000001$pb":
  leal LC0-"L000001$pb"(%ebx), %eax
  movl %eax, (%esp)
  call L_printf$stub
  addl $20, %esp
  popl %ebx
  popl %ebp
  ret
```

Symbolic names for the machine code instructions

Assembler converts Assembly Code to Machine Code

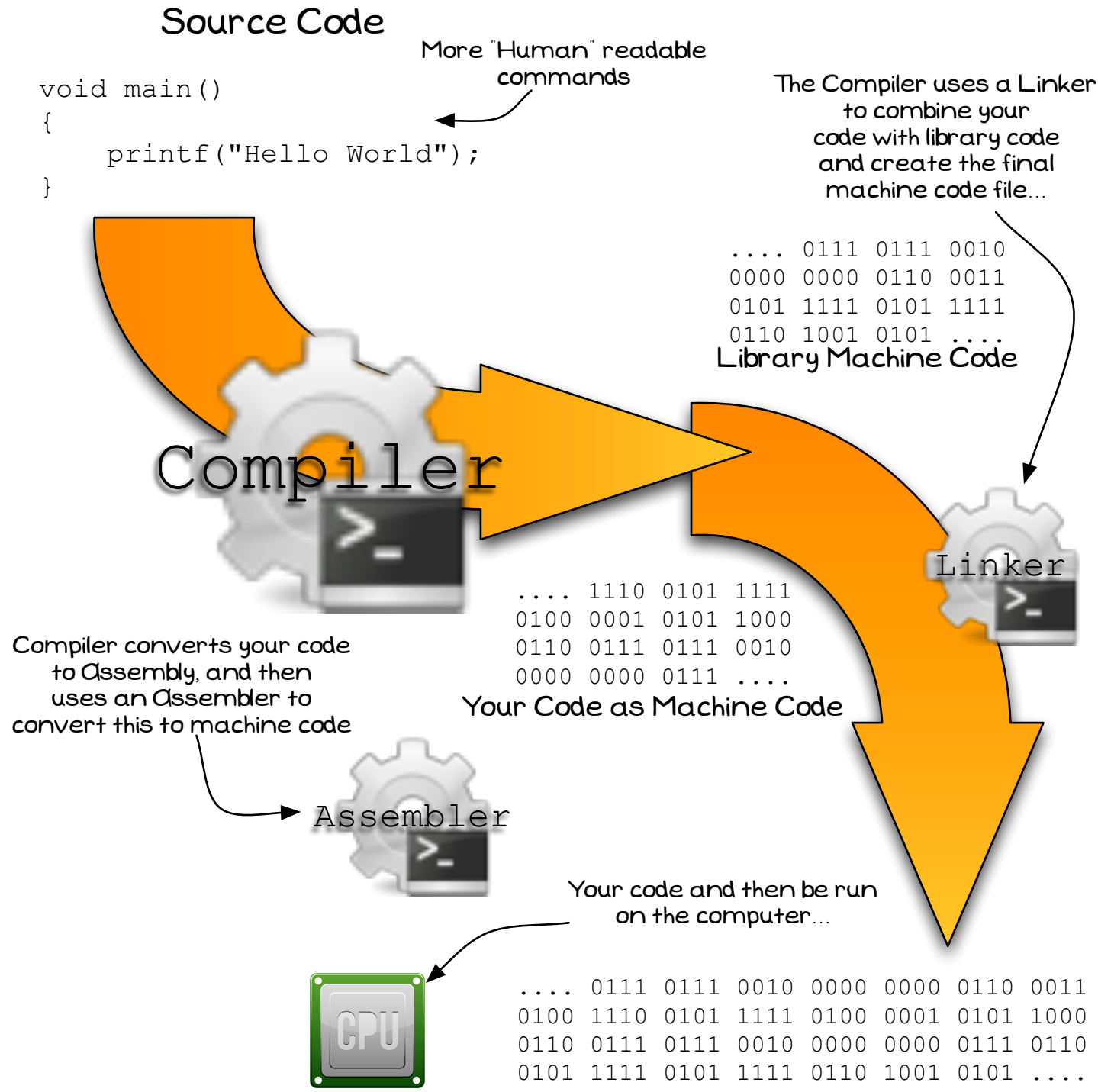


Only the machine code can be executed by the computer

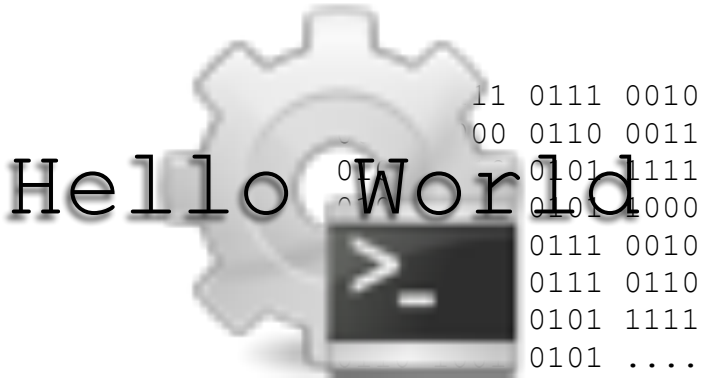



```
.... 0111 0111 0010 0000 0000 0110 0011
0100 1110 0101 1111 0100 0001 0101 1000
0110 0111 0111 0010 0000 0000 0111 0110
0101 1111 0101 1111 0110 1001 0101 ....
```

Machine Code

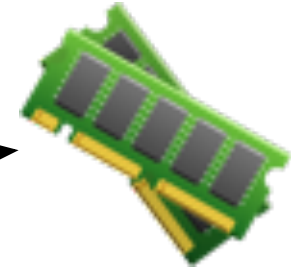

Source Code
More "Human" readable commands
void main()
{
printf("Hello World");
}
The Compiler uses a Linker to combine your code with library code and create the final machine code file...
.... 0111 0111 0010
0000 0000 0110 0011
0101 1111 0101 1111
0110 1001 0101 ....
Library Machine Code
Compiler
Linker
.... 1110 0101 1111
0100 0001 0101 1000
0110 0111 0111 0010
0000 0000 0111 ....
Your Code as Machine Code
Compiler converts your code to Assembly, and then uses an Assembler to convert this to machine code
Assembler
Your code and then be run on the computer...
CPU
.... 0111 0111 0010 0000 0000 0110 0011
0100 1110 0101 1111 0100 0001 0101 1000
0110 0111 0111 0010 0000 0000 0111 0110
0101 1111 0101 1111 0110 1001 0101 ....

A program is an executable file



It contains machine code instructions to tell the computer what to do when it is run

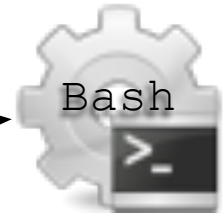
The program's file will exist on your computer's disk



The file is loaded into memory before it can start executing

The Terminal is a program.

It creates an Window into which programs can output text, and read input.



Bash is a shell - a program that interprets your commands and commands the computer to perform actions...

Bash is run for you when the Terminal starts, it will show you a command prompt where you can type your commands

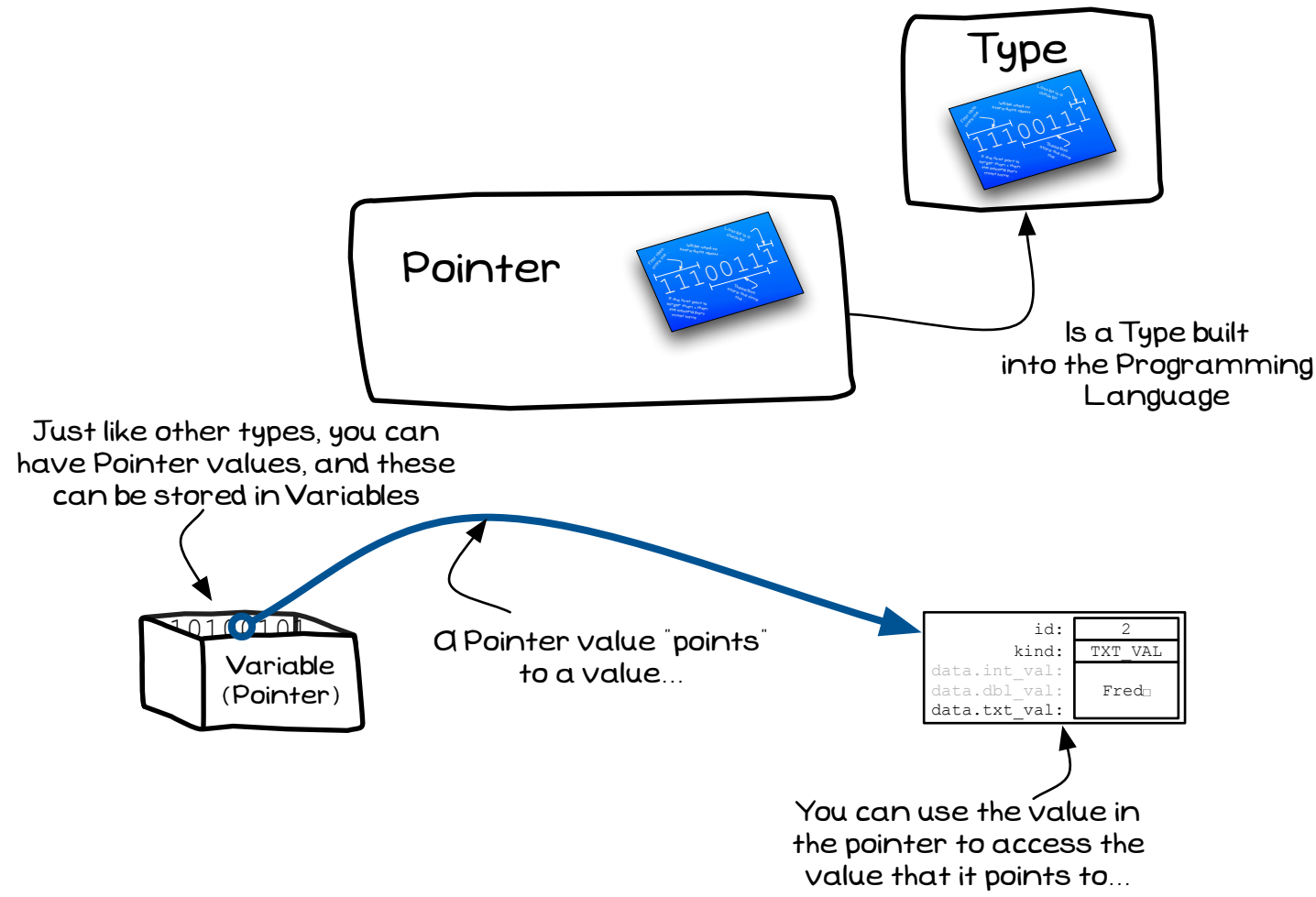
4 Heap

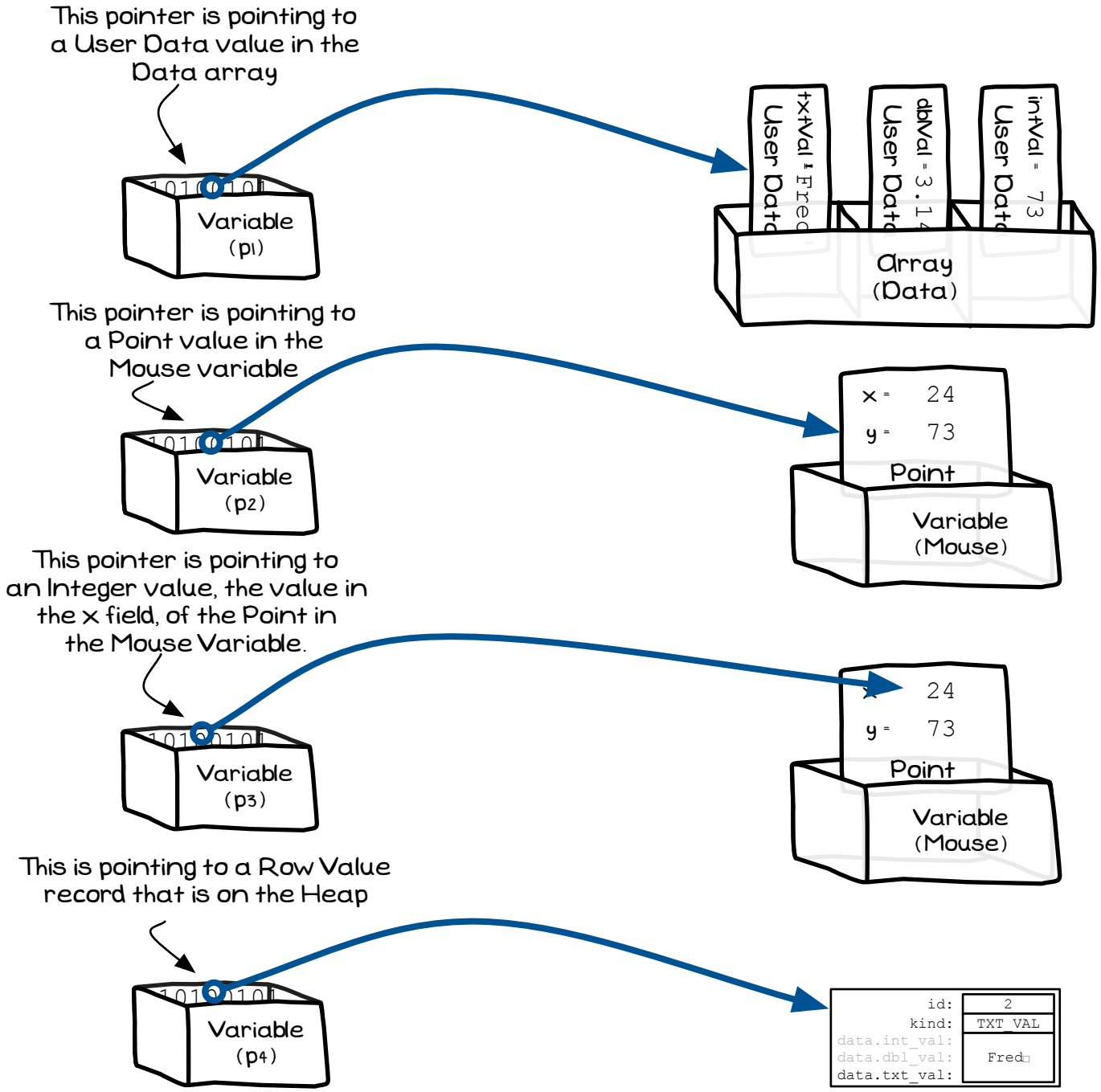
Stack
2

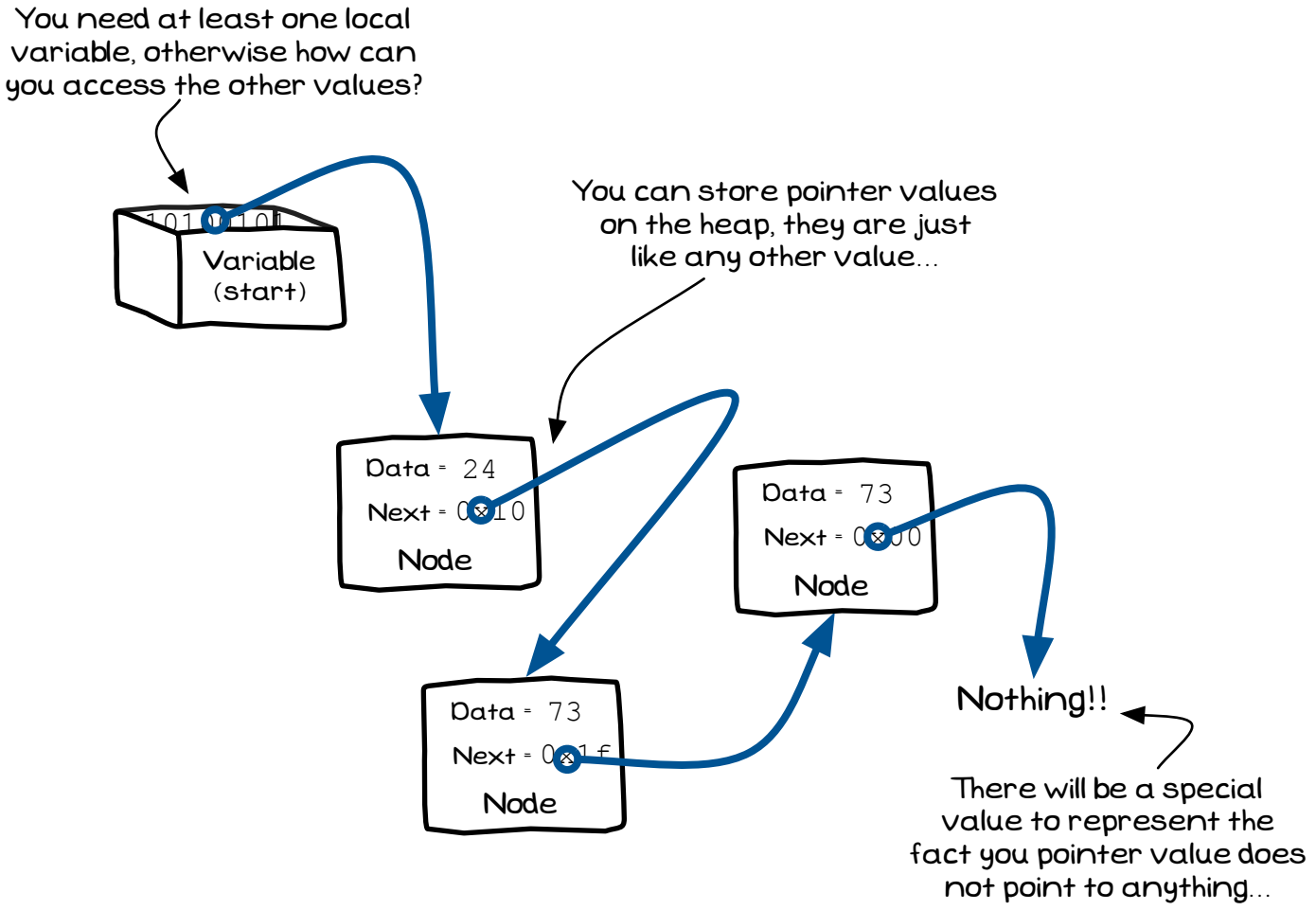
Code Area
1

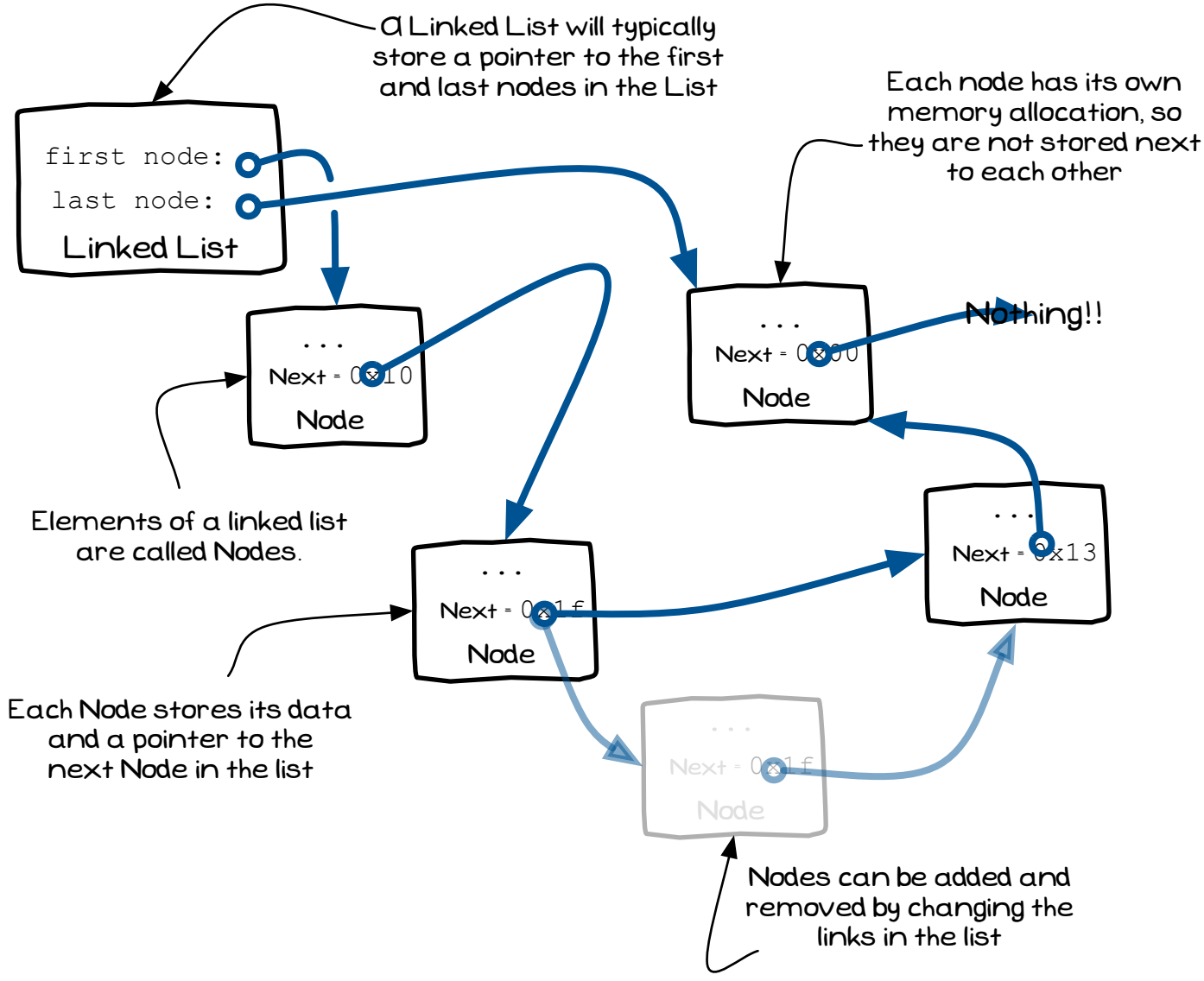
3 Global Variables

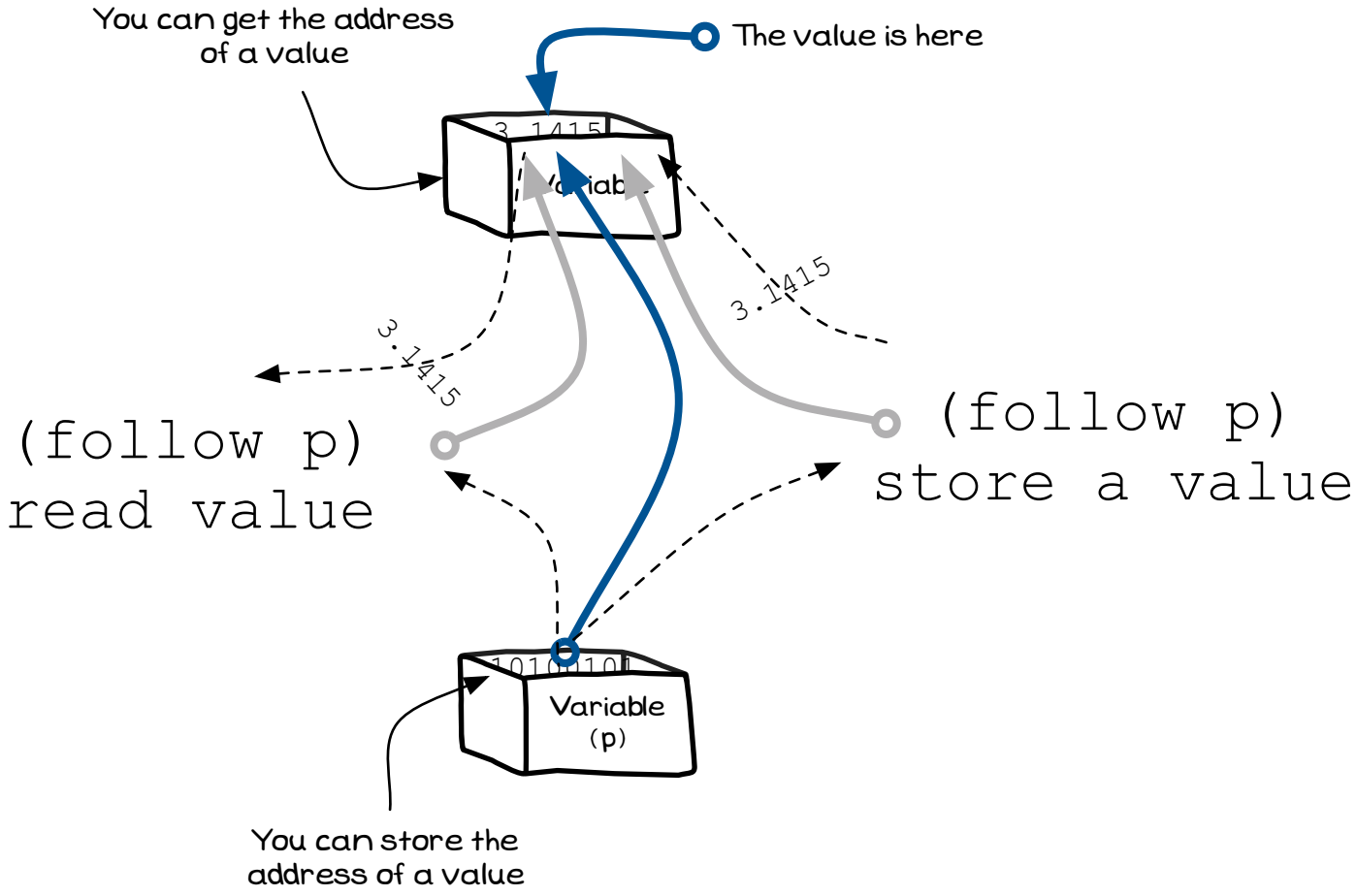
The Program's Memory

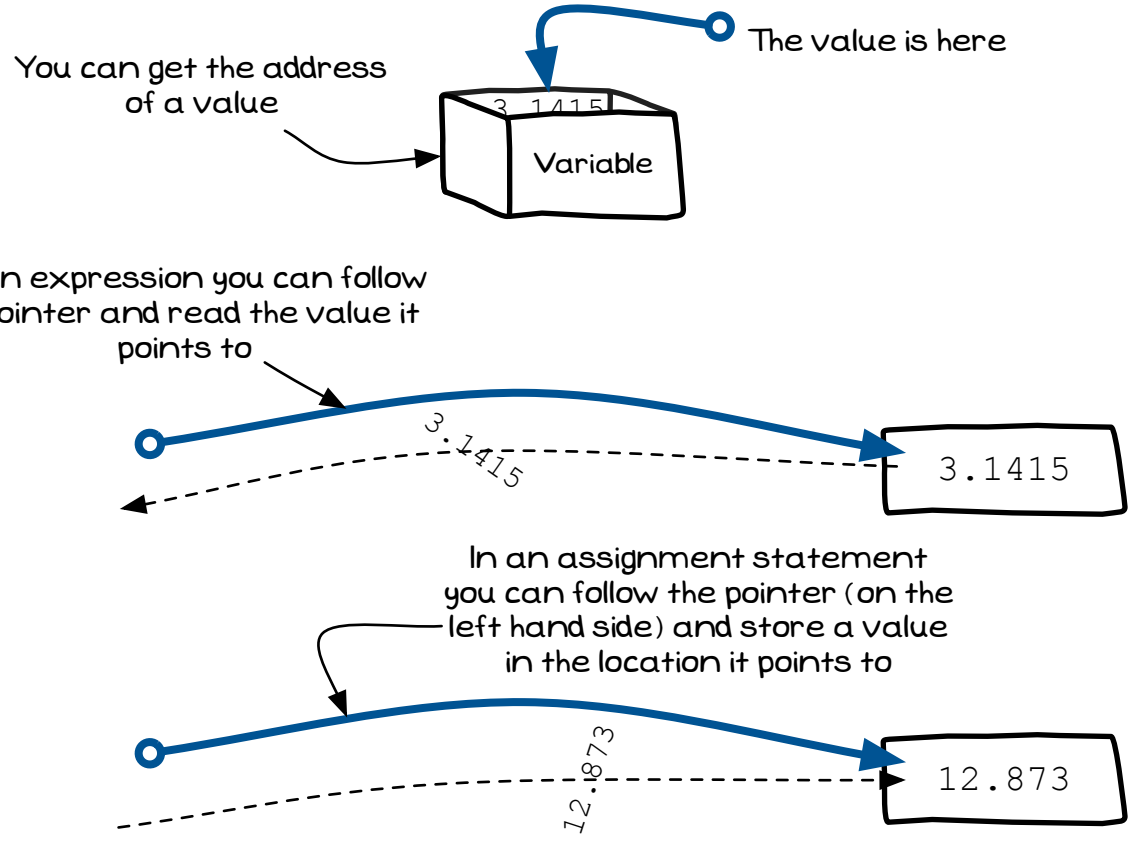

Type
Pointer
Is a Type built into the Programming Language
Just like other types, you can have Pointer values, and these can be stored in Variables
Variable (Pointer)
A Pointer value "points" to a value...
id: 2
kind: TXT_VAL
data.int_val:
data.dbl_val:
data.txt_val: Fred
You can use the value in the pointer to access the value that it points to...

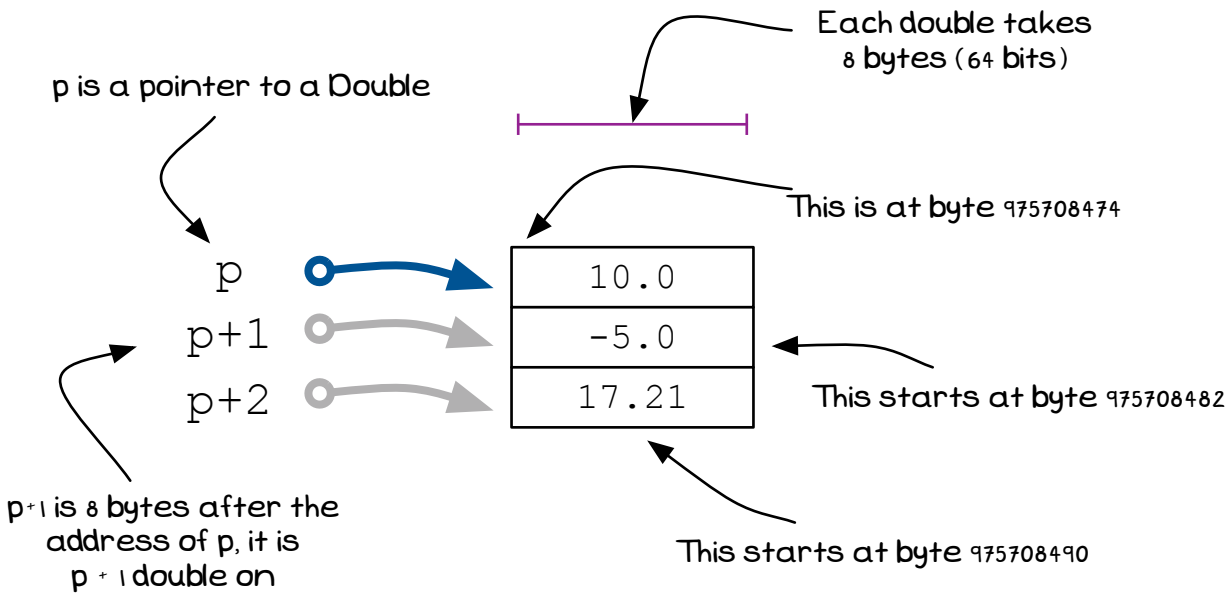

This pointer is pointing to a User Data value in the Data array
Variable (p1)
txtVal = Fred
dblVal = 3.14
intVal = 73
User Data
Array (Data)
This pointer is pointing to a Point value in the Mouse variable
Variable (p2)
x = 24
y = 73
Point
Variable (Mouse)
This pointer is pointing to an Integer value, the value in the x field, of the Point in the Mouse Variable.
Variable (p3)
x = 24
y = 73
Point
Variable (Mouse)
This is pointing to a Row Value record that is on the Heap
Variable (p4)
id: 2
kind: TXT_VAL
data.int_val:
data.dbl_val:
data.txt_val: Fred


You need at least one local variable, otherwise how can you access the other values?
Variable (start)
You can store pointer values on the heap, they are just like any other value...
Data = 24
Next = 0x10
Node
Data = 73
Next = 0x00
Node
Data = 73
Next = 0x1f
Node
Nothing!!
There will be a special value to represent the fact you pointer value does not point to anything...

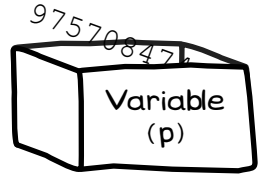

A Linked List will typically store a pointer to the first and last nodes in the List
Each node has its own memory allocation, so they are not stored next to each other
first node:
last node:
Linked List
. . .
Next = 0x10
Node
. . .
Next = 0x00
Node
Nothing!!
Elements of a linked list are called Nodes.
. . .
Next = 0x1f
Node
. . .
Next = 0x13
Node
Each Node stores its data and a pointer to the next Node in the list
. . .
Next = 0x1f
Node
Nodes can be added and removed by changing the links in the list

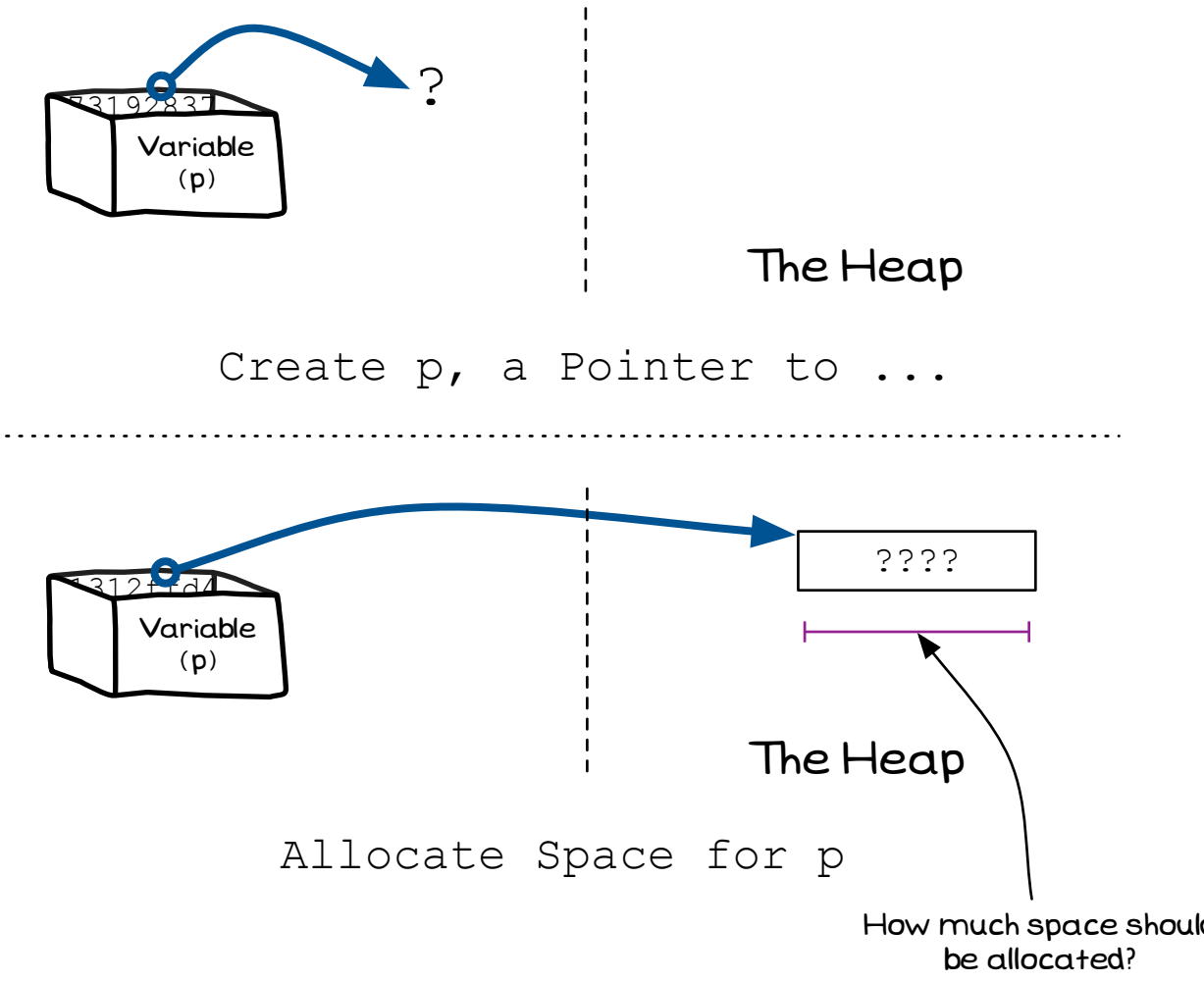

You can get the address of a value
The value is here
3.1415
3.1415
3.1415
(follow p)
read value
(follow p)
store a value
Variable
(p)
You can store the address of a value

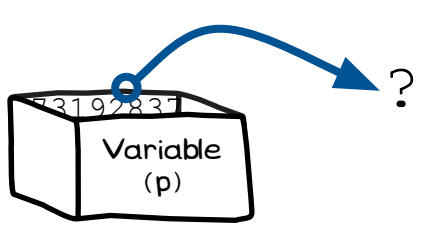

You can get the address of a value
The value is here
3.1415
Variable
In an expression you can follow a pointer and read the value it points to
3.1415
3.1415
In an assignment statement you can follow the pointer (on the left hand side) and store a value in the location it points to
12.873
12.873



This value stored in p is 975708474,
the address of where the value is stored

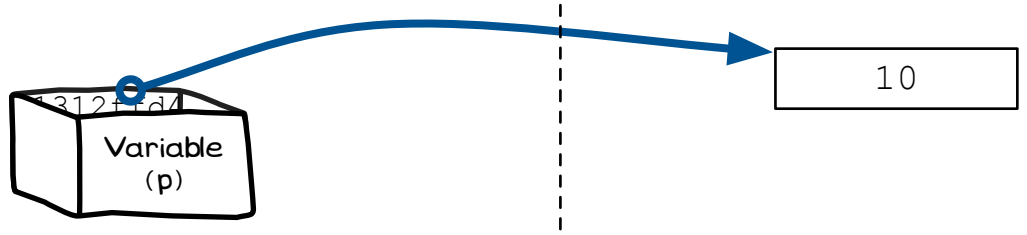



?
Variable
(p)
The Heap
Create p, a Pointer to ...
????
Variable
(p)
The Heap
Allocate Space for p
How much space should
be allocated?



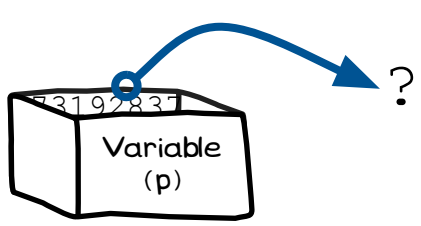
The Heap

Create p, a Pointer to an Integer



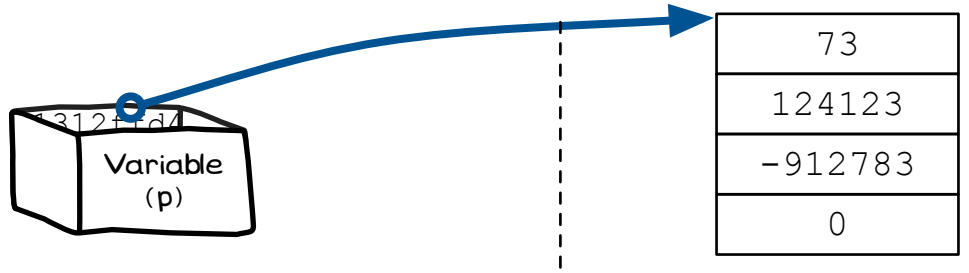
The Heap

Allocate Space for what p points to (an Integer)



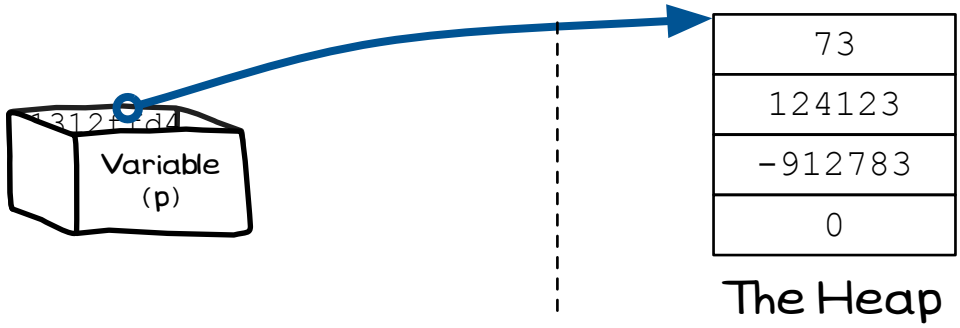
The Heap

Create p, a Pointer to an Integer

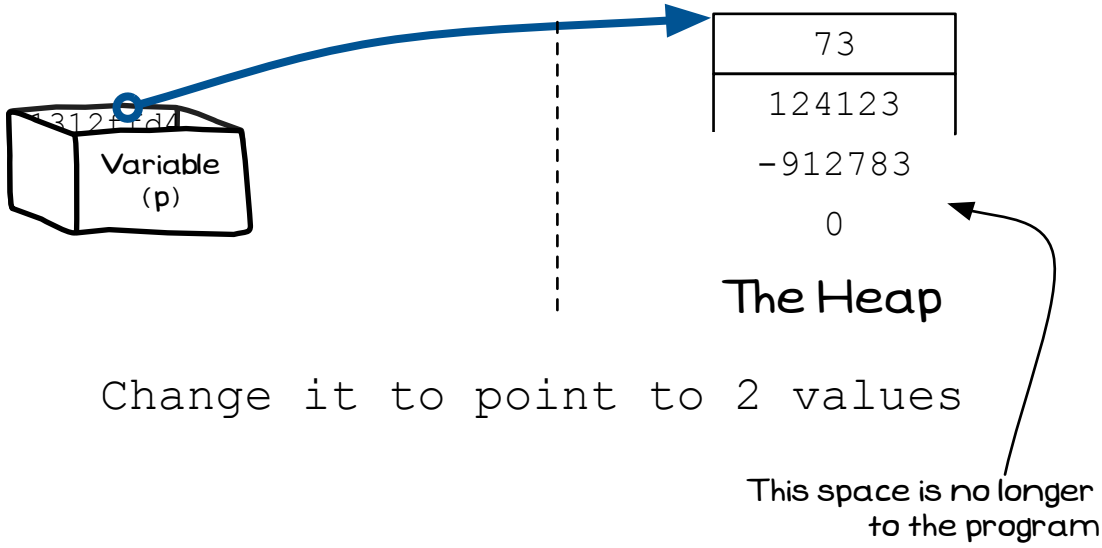


The Heap

Allocate Space for 4 Integer values

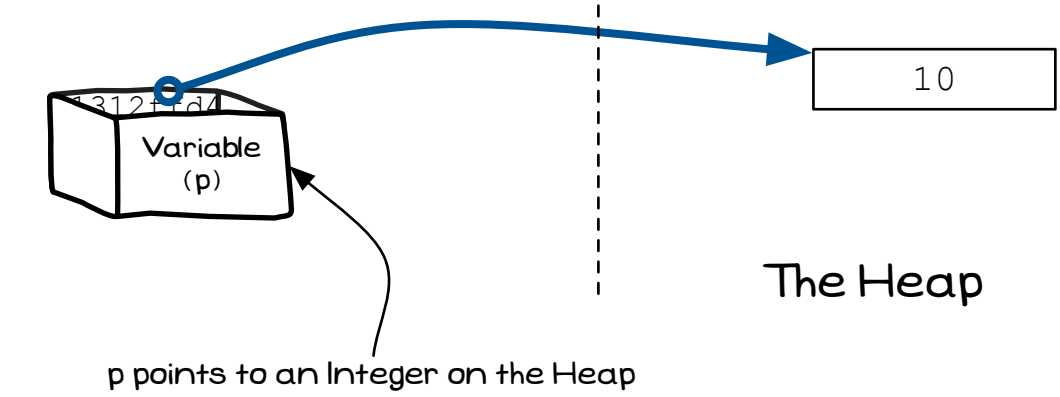


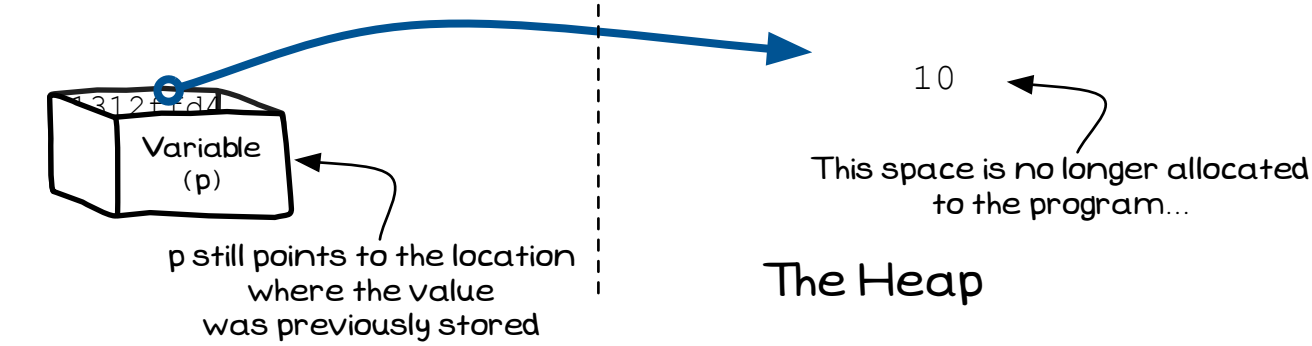
p points to 4 Integer value



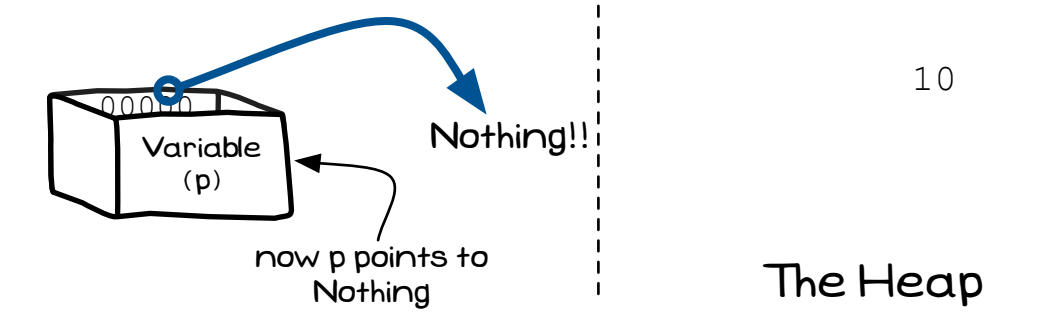
Change it to point to 2 values

This space is no longer allocated to the program...

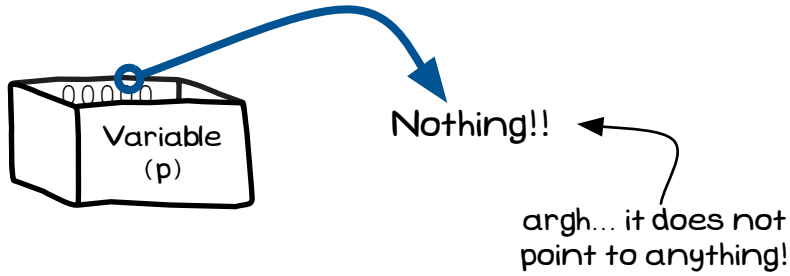




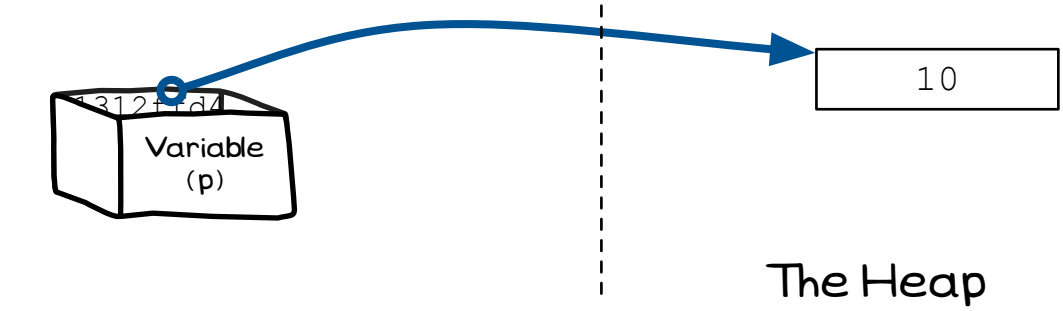
Free the memory referred to by p

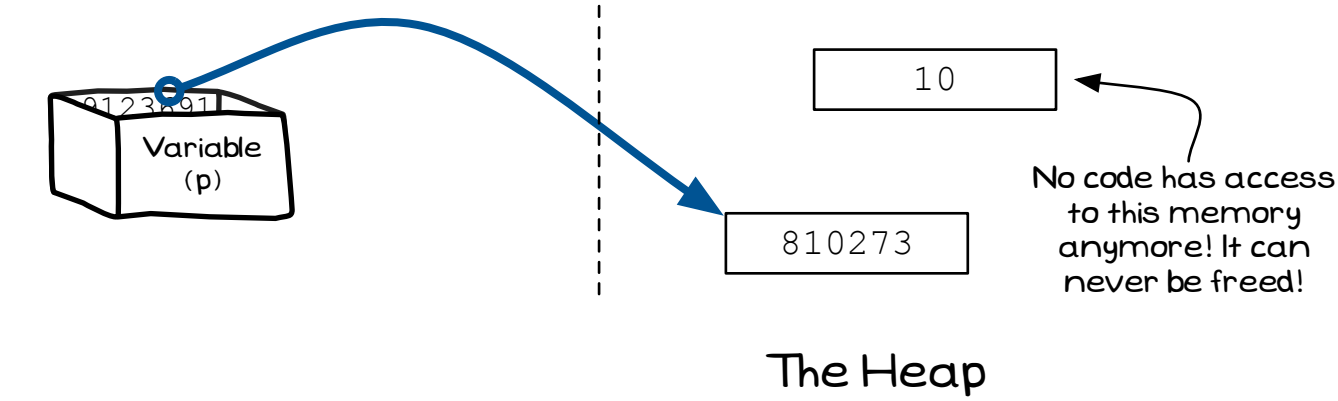


Set p to point to Nothing!

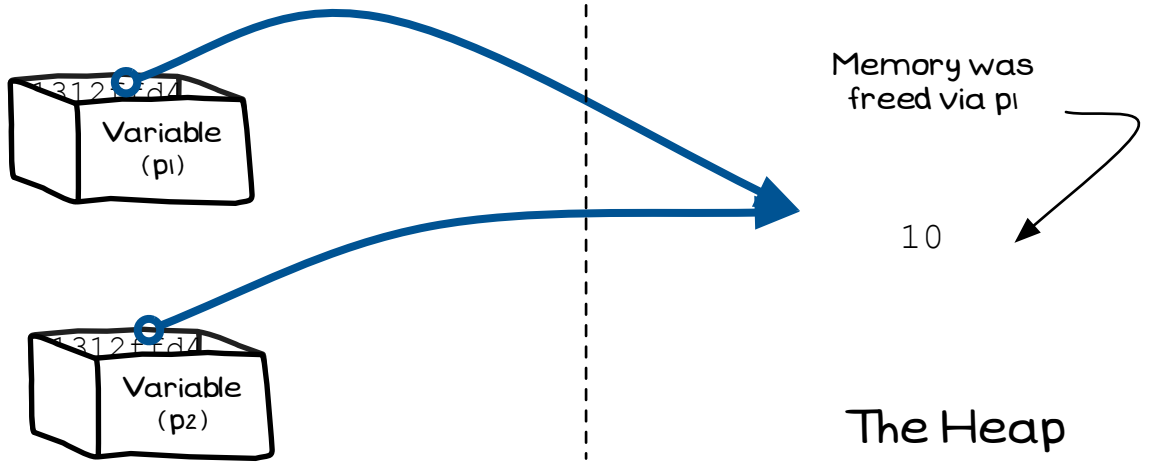


Follow the Pointer p, and ...

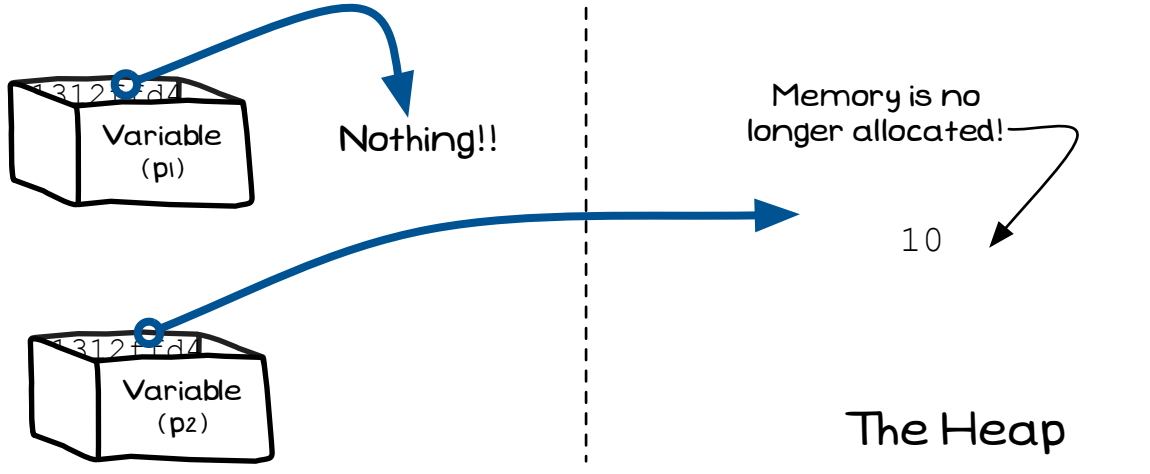




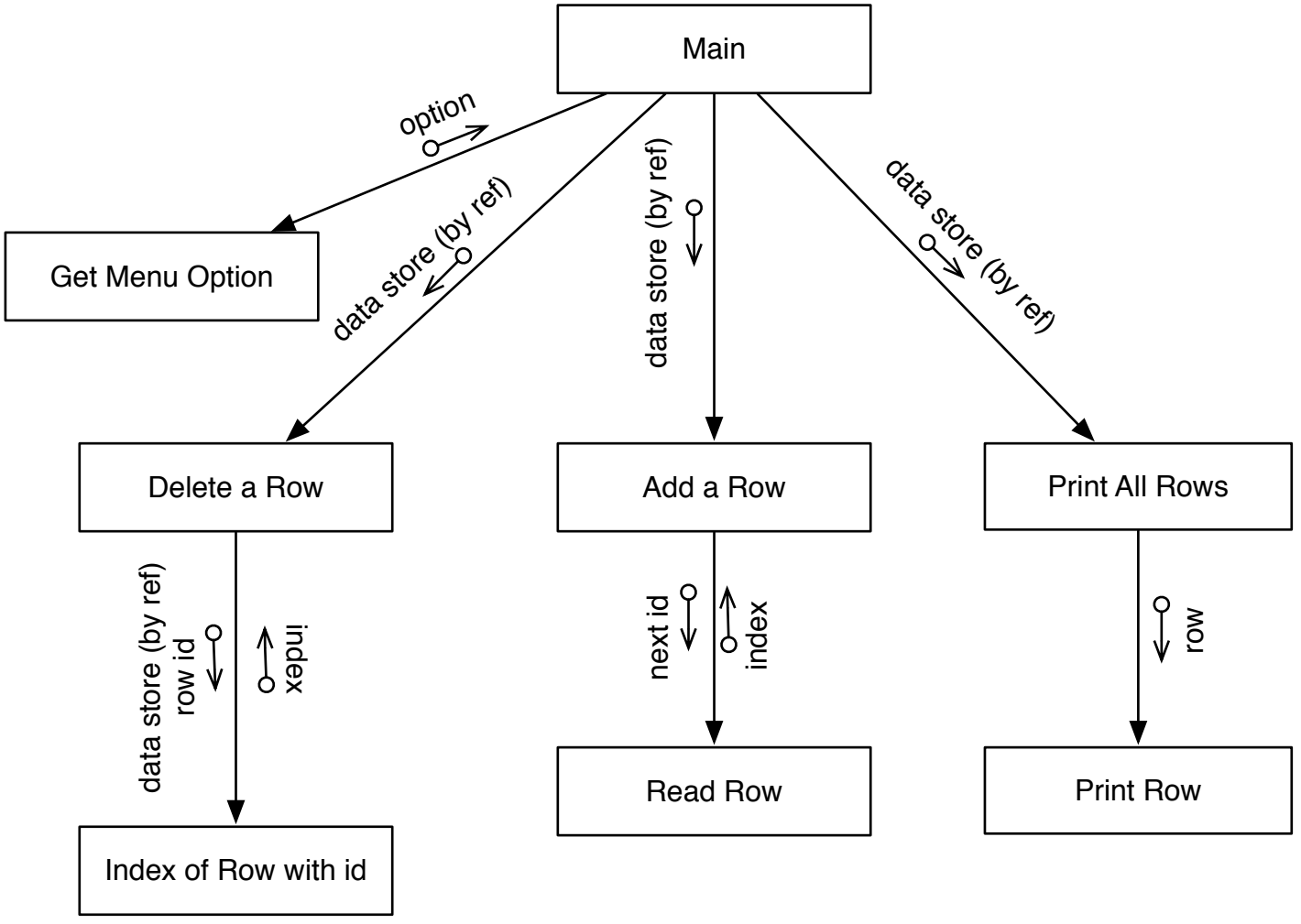
Allocate some memory, and point p at it

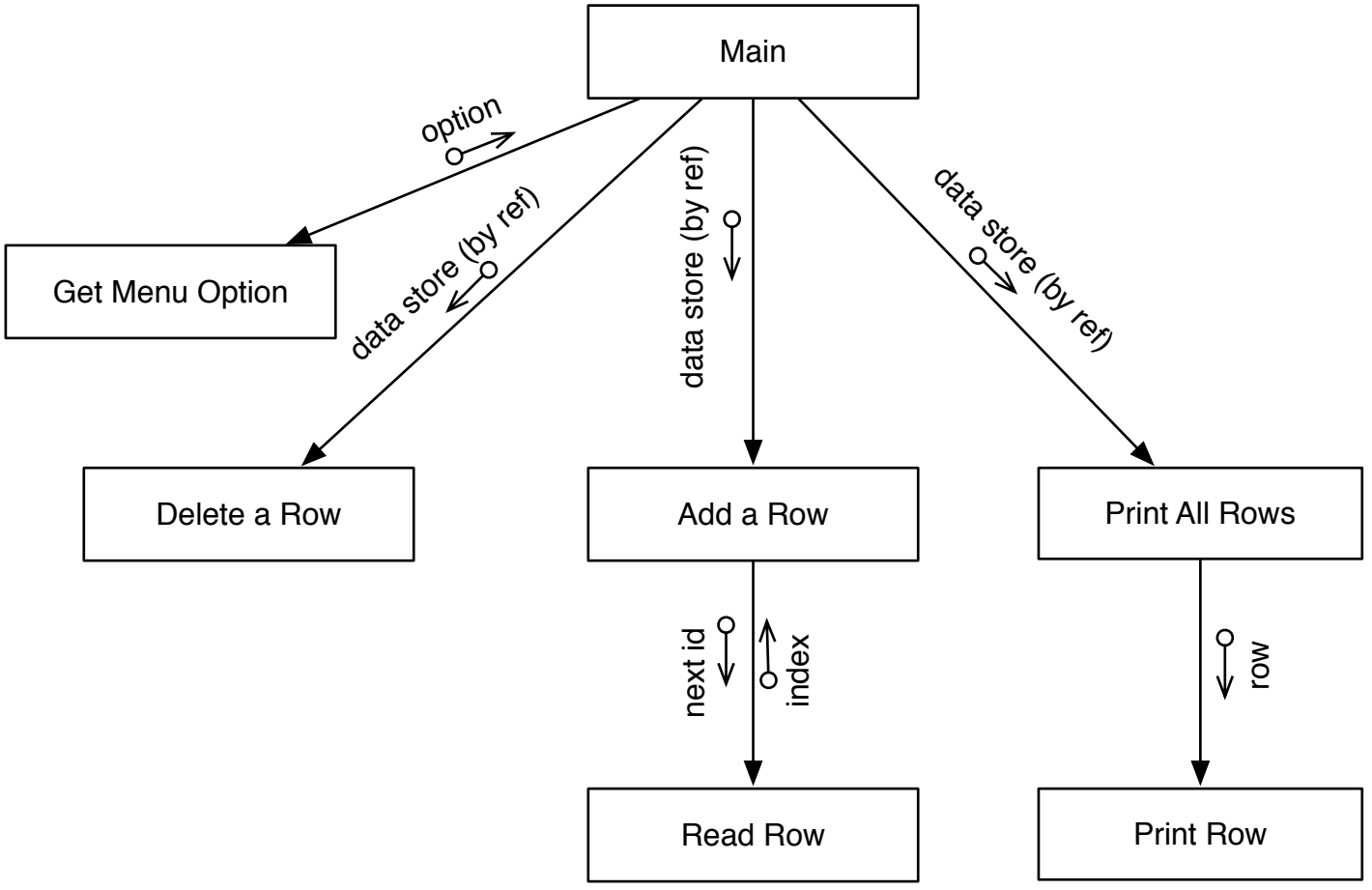


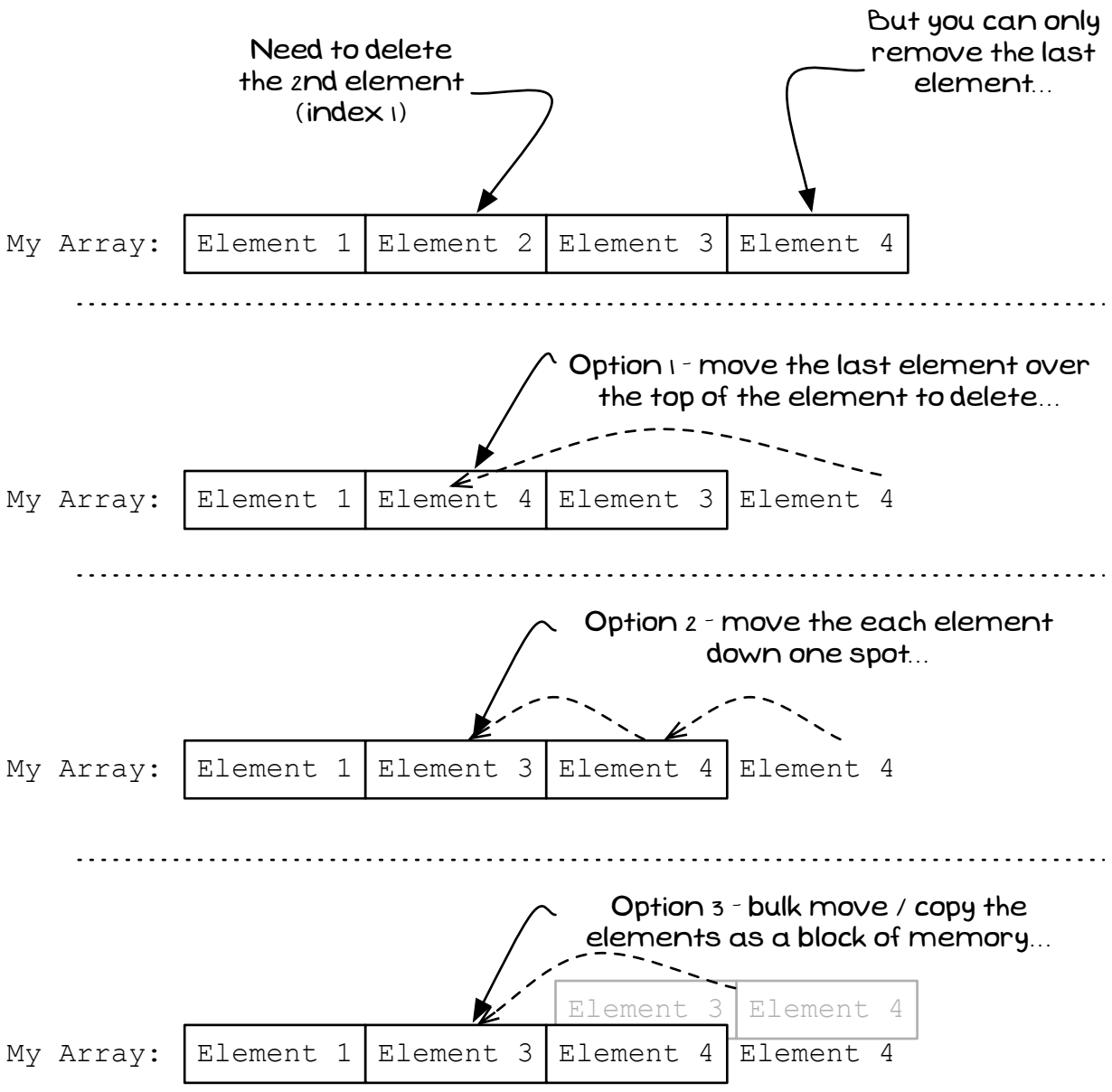
Free memory at p1

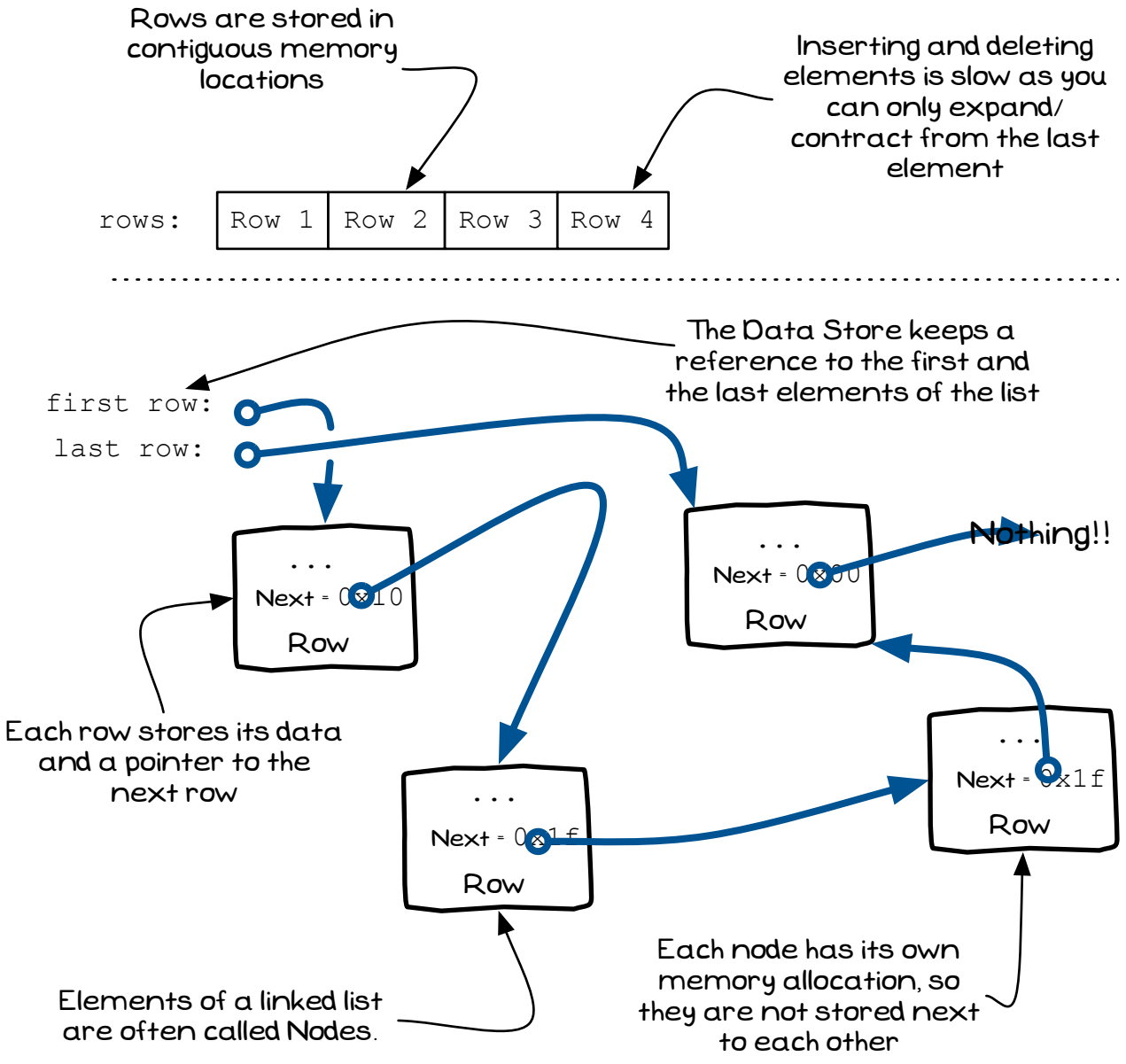


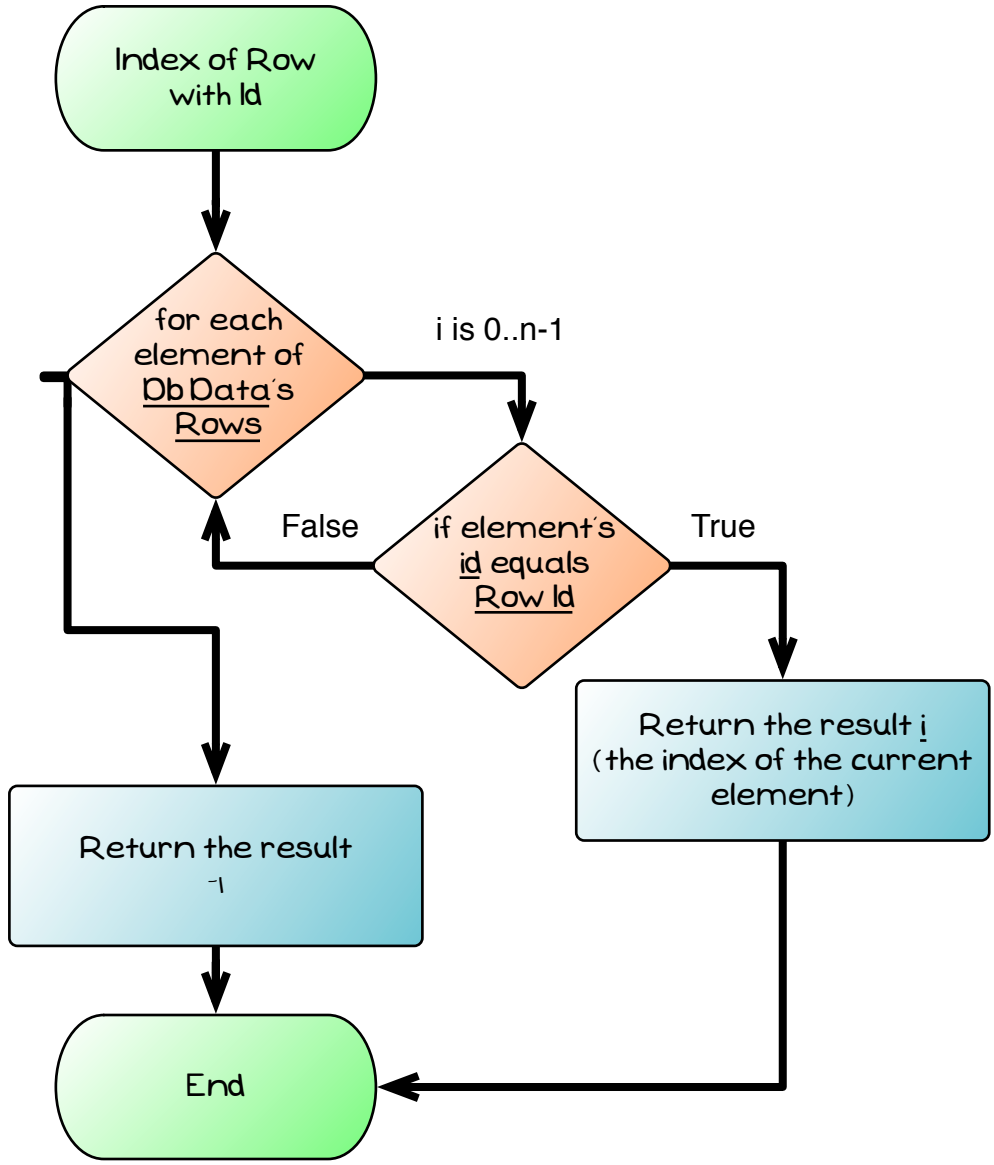
Set p1 to point to Nothing
Read memory at p2 ...

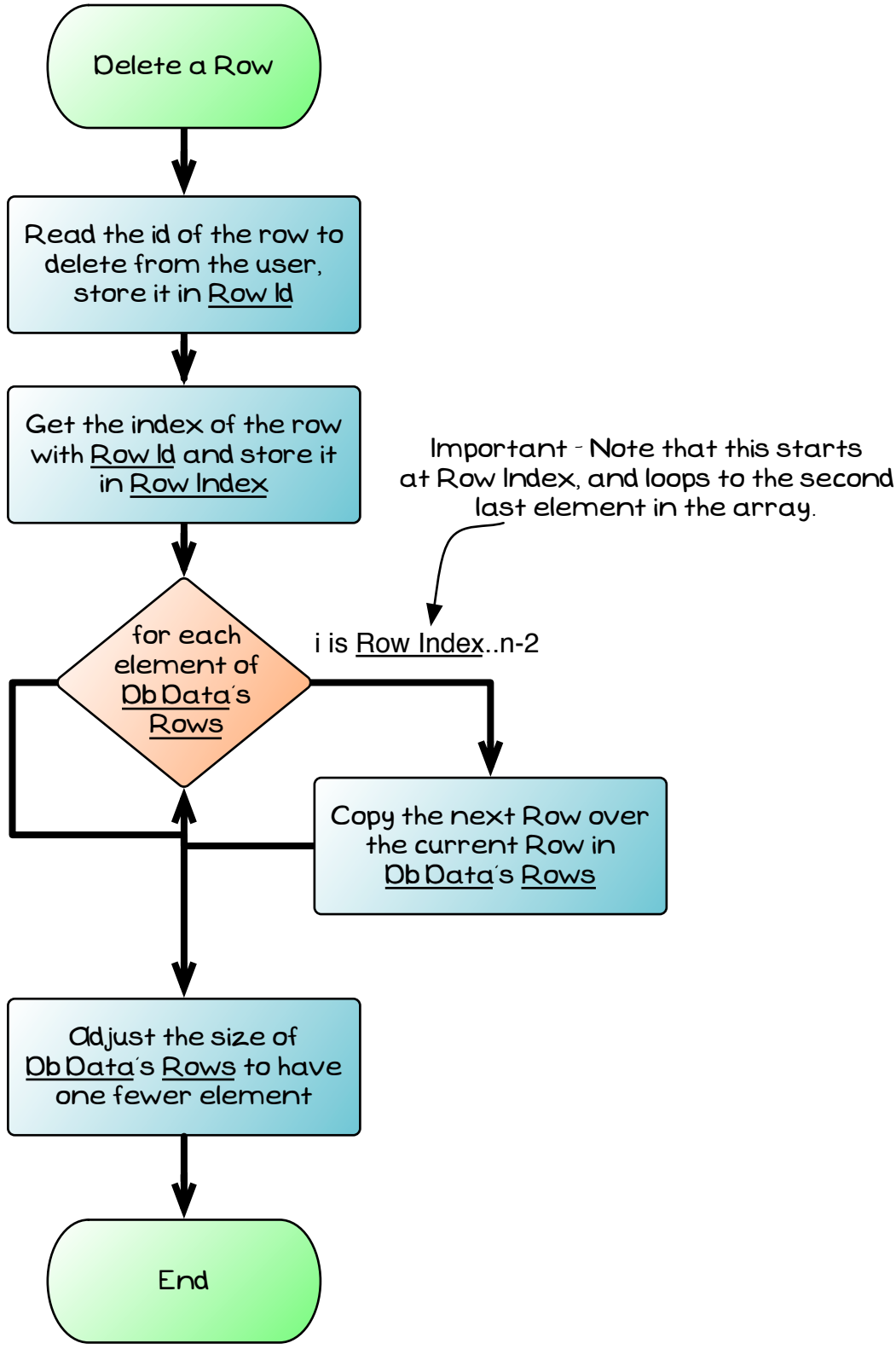

Main
option
Get Menu Option
data store (by ref)
data store (by ref)
data store (by ref)
Delete a Row
Add a Row
Print All Rows
data store (by ref)
row id
index
next id
index
row
Index of Row with id
Read Row
Print Row

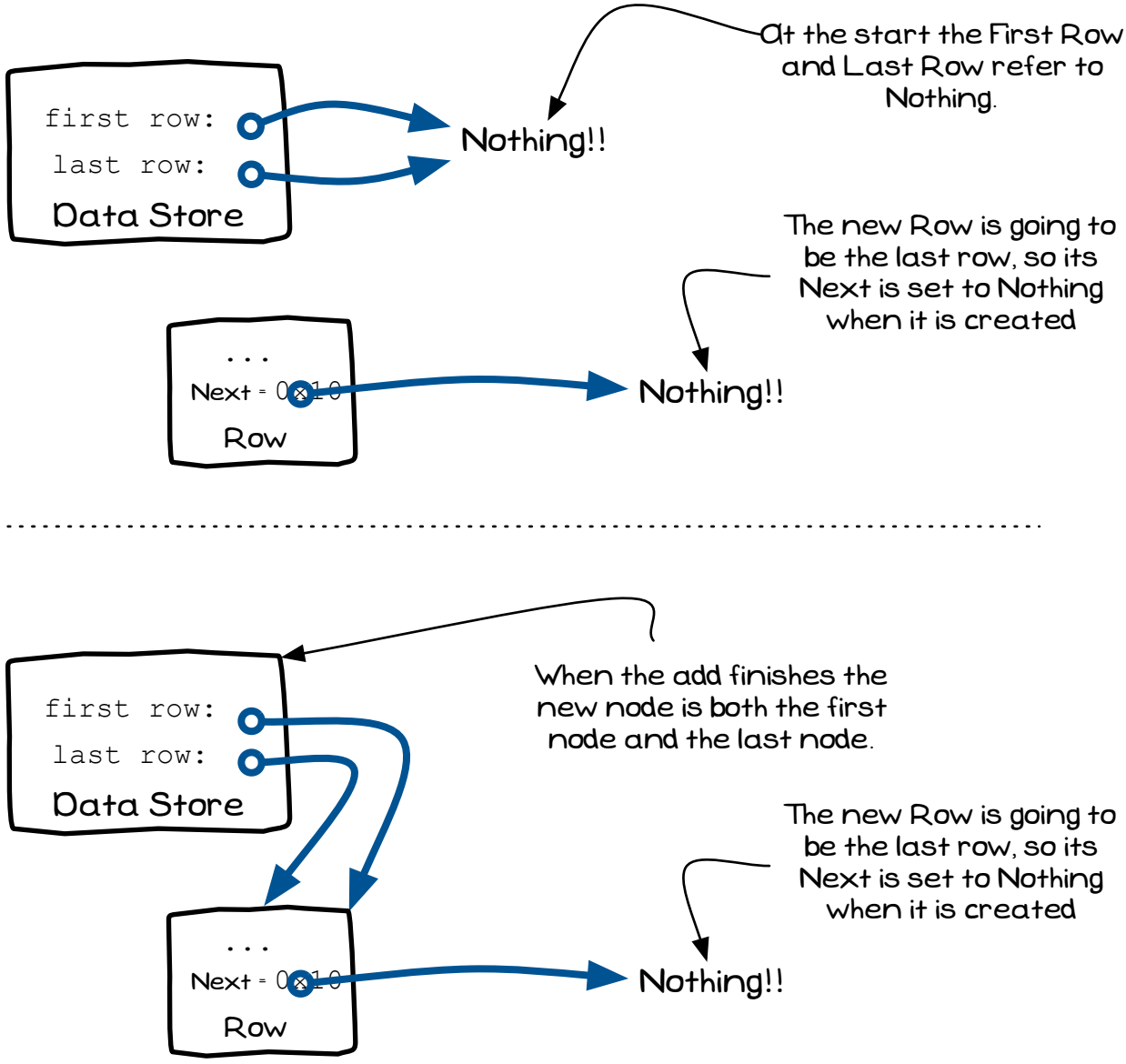

Main
option
Get Menu Option
data store (by ref)
data store (by ref)
data store (by ref)
Delete a Row
Add a Row
Print All Rows
next id
index
row
Read Row
Print Row

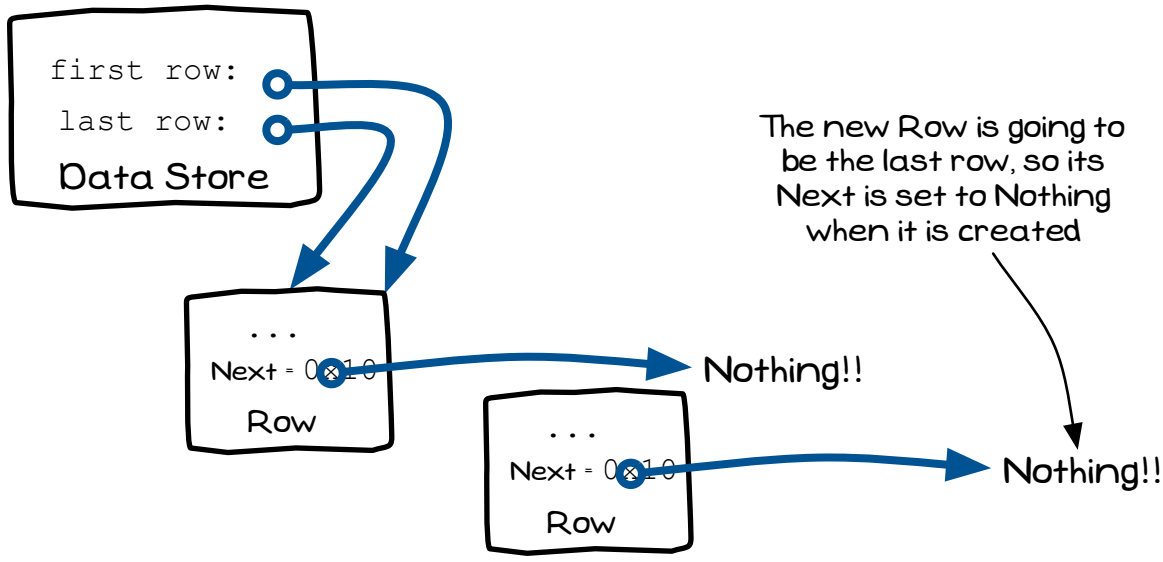

Need to delete the 2nd element (index 1)
But you can only remove the last element...
My Array:
Element 1
Element 2
Element 3
Element 4
Option 1 - move the last element over the top of the element to delete...
My Array:
Element 1
Element 4
Element 3
Element 4
Option 2 - move the each element down one spot...
My Array:
Element 1
Element 3
Element 4
Element 4
Option 3 - bulk move / copy the elements as a block of memory...
Element 3
Element 4
My Array:
Element 1
Element 3
Element 4
Element 4

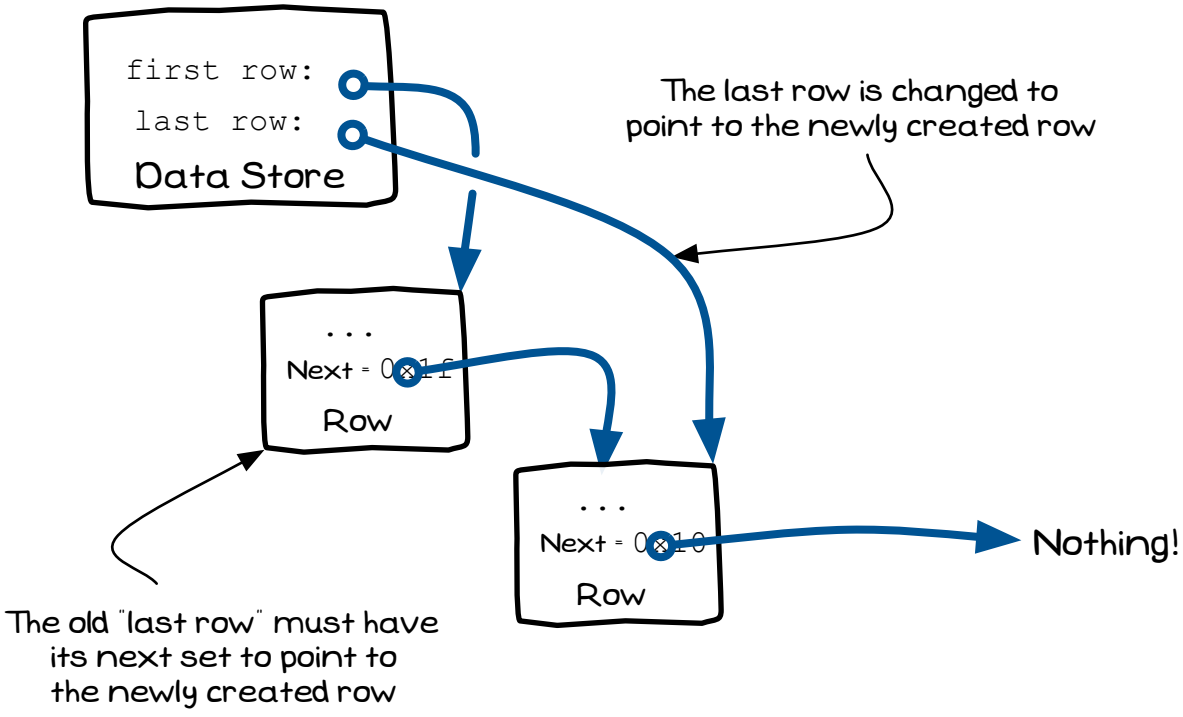

Rows are stored in contiguous memory locations
Inserting and deleting elements is slow as you can only expand/ contract from the last element
rows:
Row 1
Row 2
Row 3
Row 4
The Data Store keeps a reference to the first and the last elements of the list
first row:
last row:
. . .
Next = 0x10
Row
. . .
Next = 0x00
Row
Nothing!!
Each row stores its data and a pointer to the next row
. . .
Next = 0x1f
Row
. . .
Next = 0x1f
Row
Elements of a linked list are often called Nodes.
Each node has its own memory allocation, so they are not stored next to each other

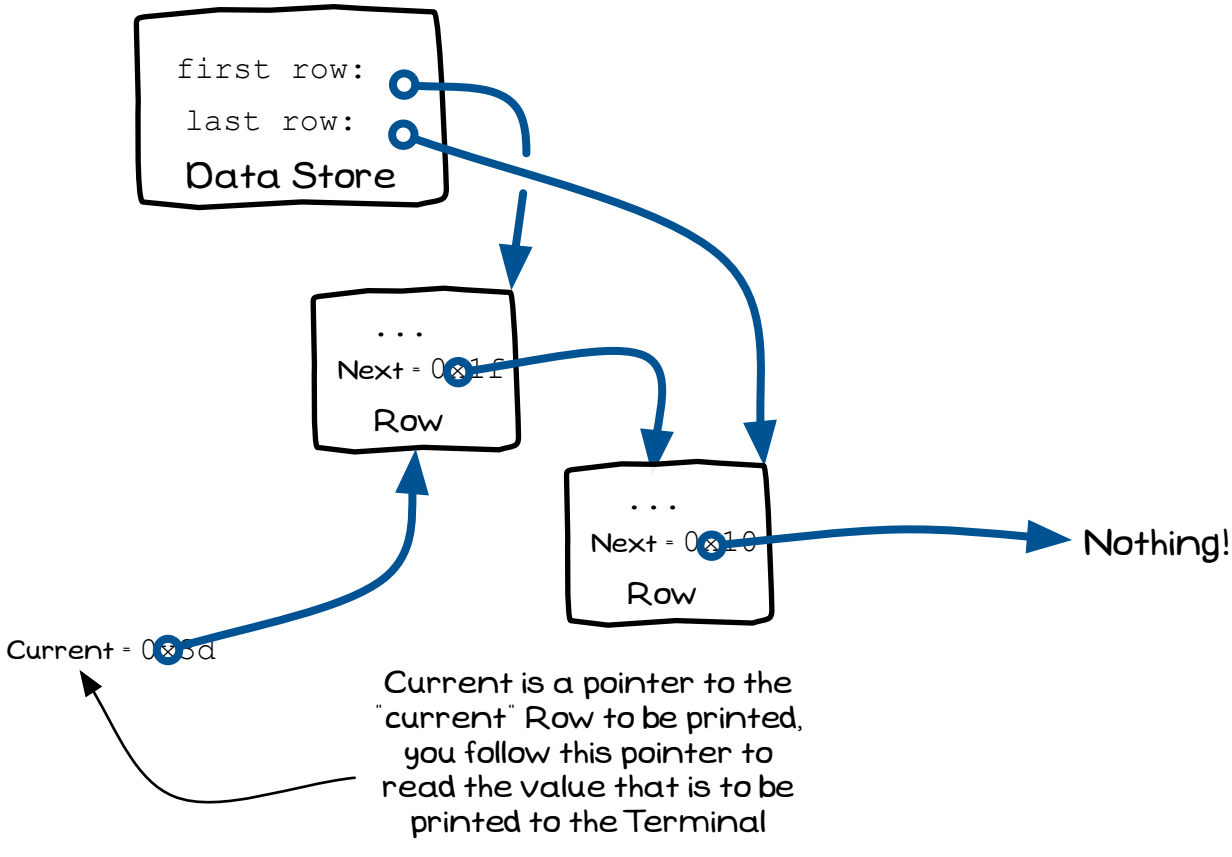

Index of Row with Id
for each element of DbData's Rows
i is 0..n-1
if element's id equals Row Id
False
True
Return the result i (the index of the current element)
Return the result -1
End

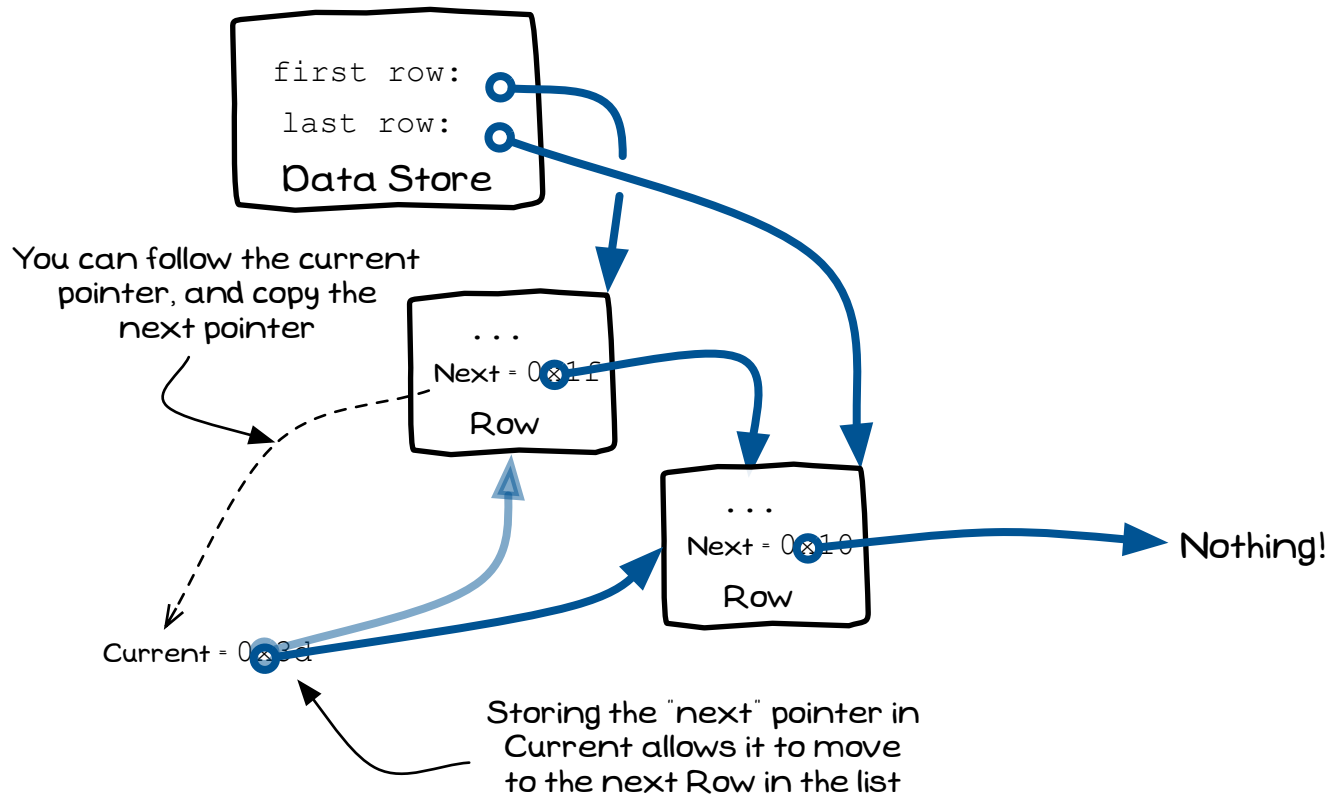

Delete a Row
Read the id of the row to delete from the user, store it in Row Id
Get the index of the row with Row Id and store it in Row Index
Important - Note that this starts at Row Index, and loops to the second last element in the array.
for each element of Db Data's Rows
i is Row Index..n-2
Copy the next Row over the current Row in Db Data's Rows
Adjust the size of Db Data's Rows to have one fewer element
End

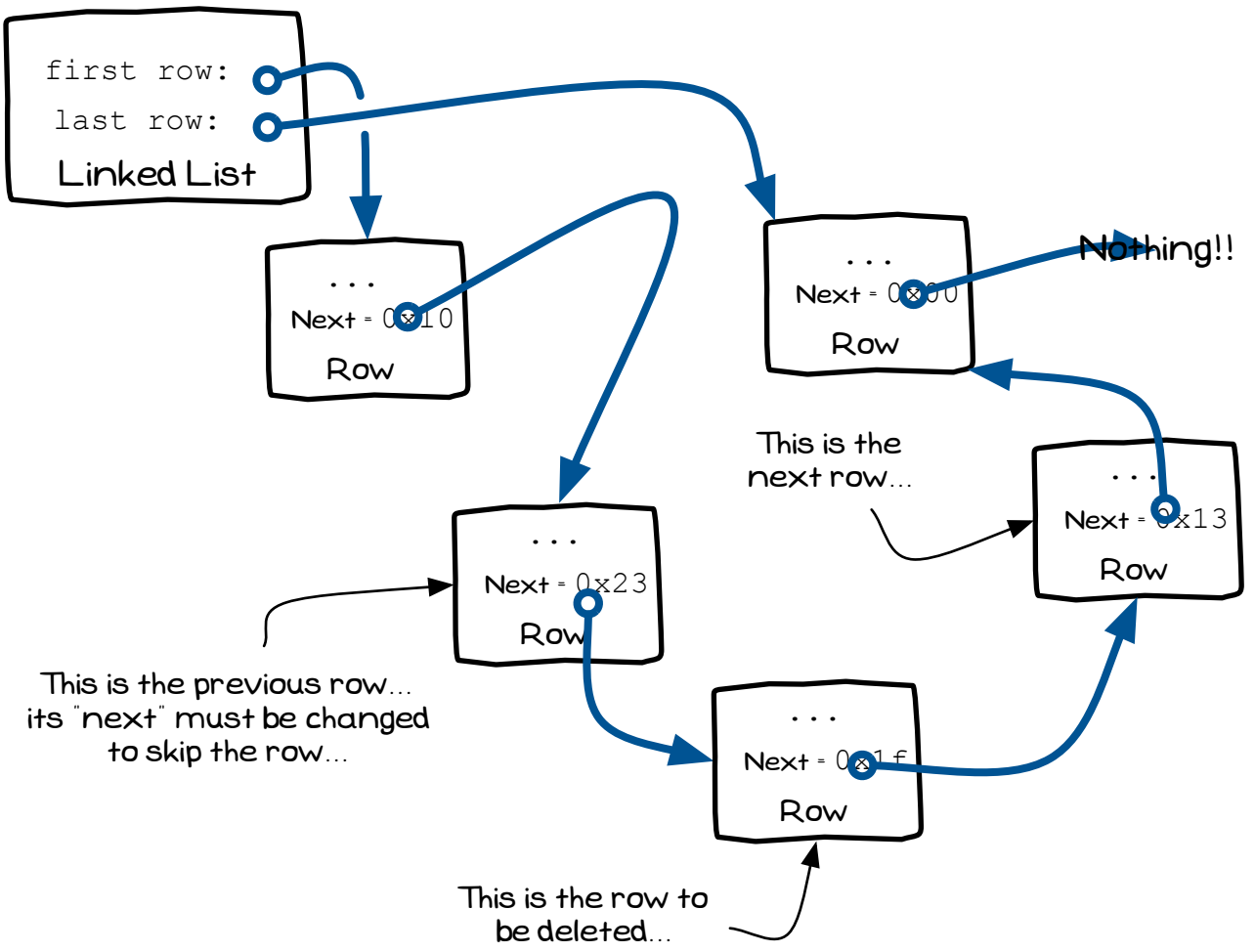

At the start the First Row and Last Row refer to Nothing.
first row:
last row:
Data Store
Nothing!!
The new Row is going to be the last row, so its Next is set to Nothing when it is created
. . .
Next = 0x10
Row
Nothing!!
When the add finishes the new node is both the first node and the last node.
first row:
last row:
Data Store
The new Row is going to be the last row, so its Next is set to Nothing when it is created
. . .
Next = 0x10
Row
Nothing!!

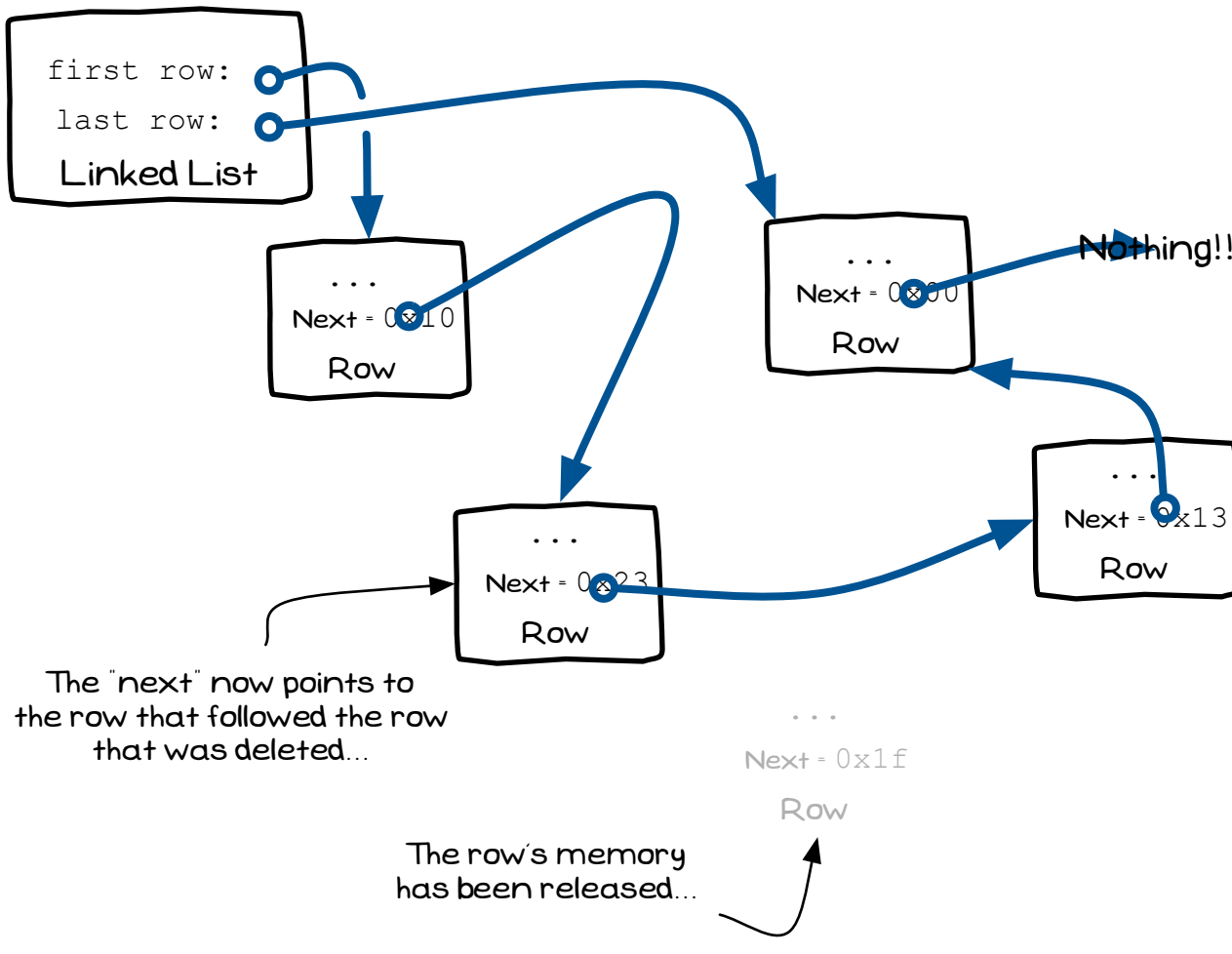

first row:
last row:
Data Store
. . .
Next = 0x10
Row
Nothing!!
. . .
Next = 0x10
Row
Nothing!!
The new Row is going to be the last row, so its Next is set to Nothing when it is created

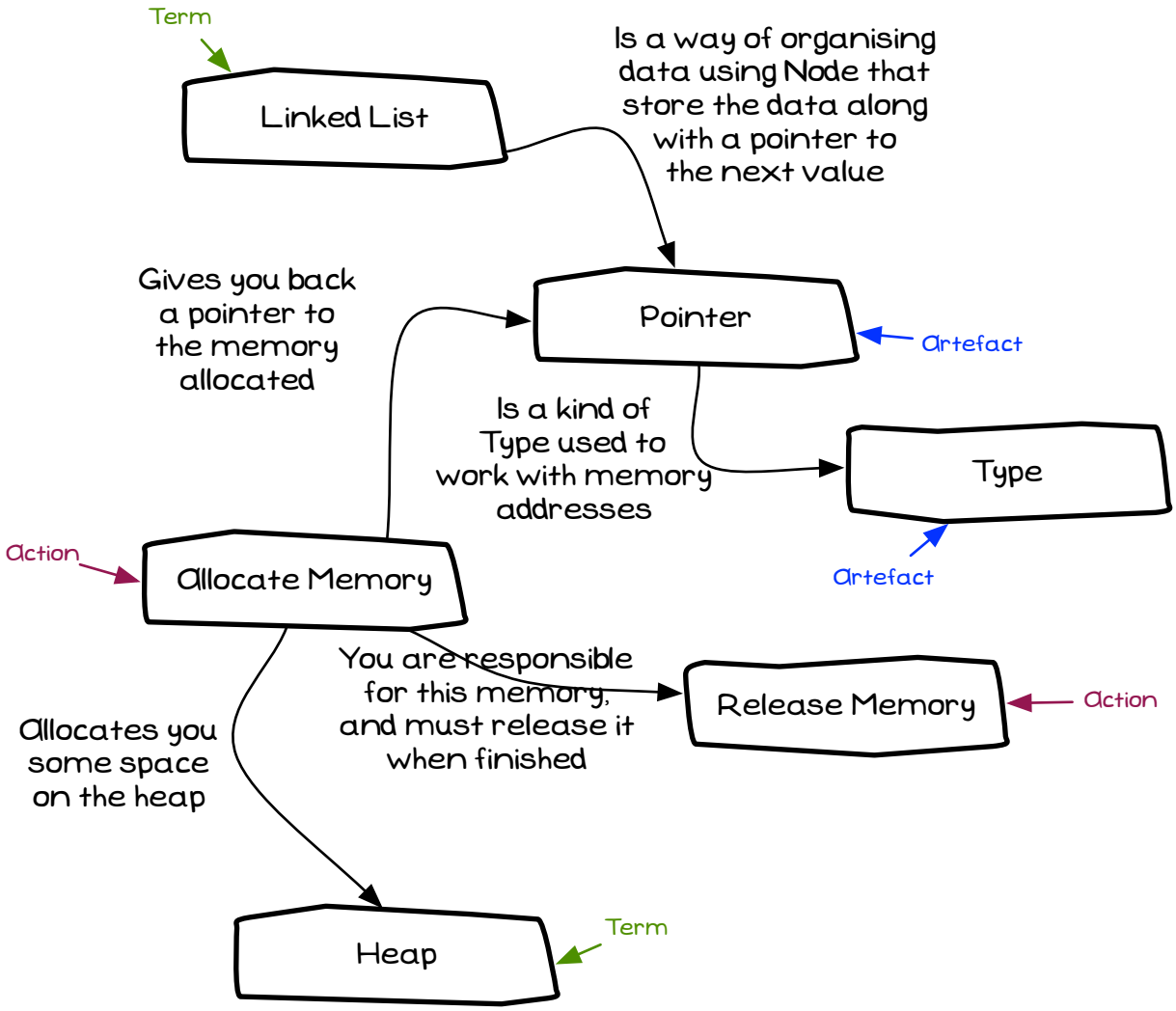

first row:
last row:
Data Store
The last row is changed to point to the newly created row
. . .
Next = 0x1F
Row
. . .
Next = 0x10
Row
Nothing!!
The old "last row" must have its next set to point to the newly created row


first row:
last row:
Data Store
. . .
Next = 0x11
Row
. . .
Next = 0x10
Row
Nothing!!
Current = 0x3d
Current is a pointer to the "current" Row to be printed, you follow this pointer to read the value that is to be printed to the Terminal


first row:
last row:
Data Store
You can follow the current pointer, and copy the next pointer
. . .
Next = 0x11
Row
. . .
Next = 0x10
Row
Nothing!!
Current = 0x3d
Storing the "next" pointer in Current allows it to move to the next Row in the list


first row:
last row:
Linked List
. . .
Next = 0x10
Row
. . .
Next = 0x00
Row
Nothing!!
This is the
next row...
. . .
Next = 0x13
Row
. . .
Next = 0x23
Row
This is the previous row...
its "next" must be changed
to skip the row...
. . .
Next = 0x1f
Row
This is the row to
be deleted...


first row:
last row:
Linked List
. . .
Next = 0x10
Row
. . .
Next = 0x00
Row
Nothing!!
. . .
Next = 0x13
Row
. . .
Next = 0x23
Row
The "next" now points to
the row that followed the row
that was deleted...
. . .
Next = 0x1f
Row
The row's memory
has been released...


Term
Linked List
Is a way of organising data using Node that store the data along with a pointer to the next value
Gives you back a pointer to the memory allocated
Pointer
Artefact
Is a kind of Type used to work with memory addresses
Type
Artefact
Action
Allocate Memory
You are responsible for this memory, and must release it when finished
Release Memory
Action
Allocates you some space on the heap
Heap
Term