COS10004 Computer Systems

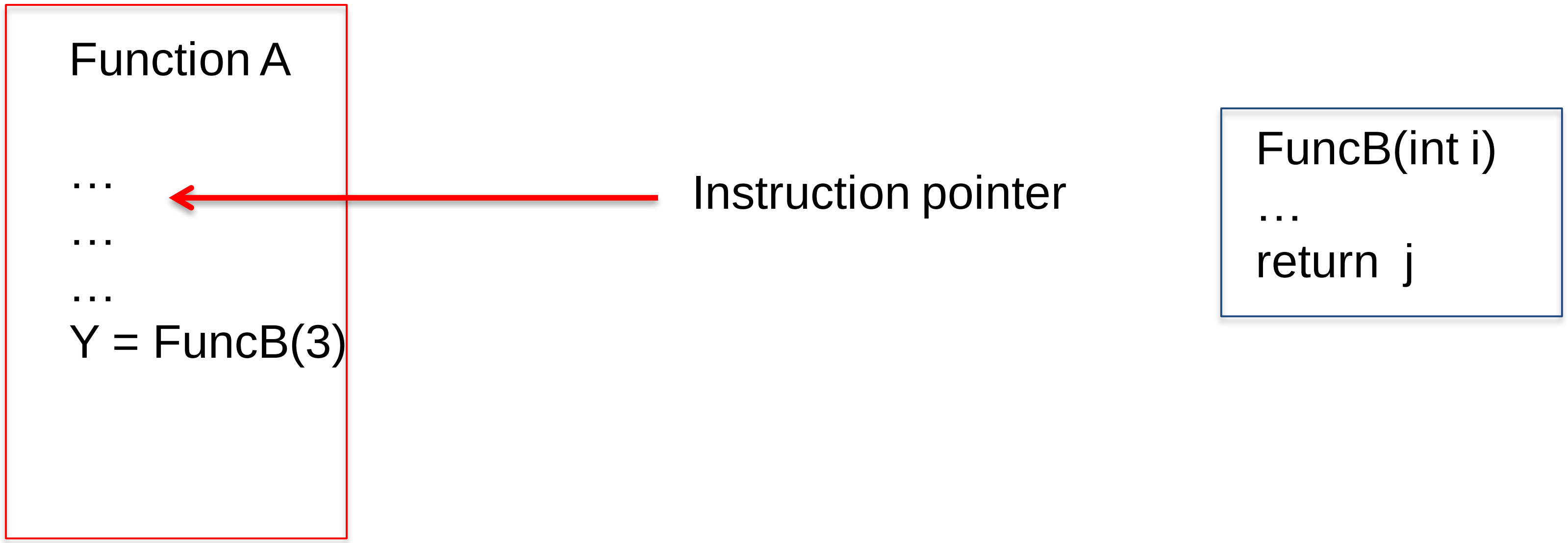
Lecture 10.1 – Functions in ARM Assembly - Function basics

Chris McCarthy

FUNCTIONS

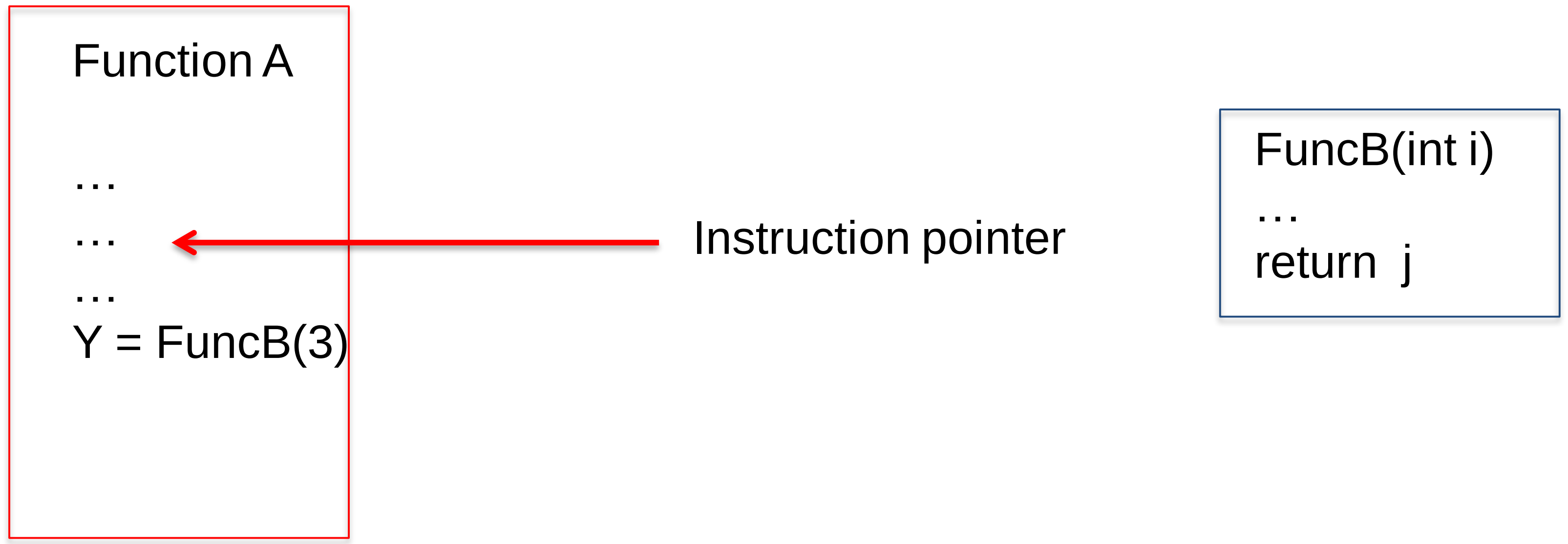
- Functions/methods/procedures/sub-routines:
  - A callable block of organised, re-usable code
  - Typically single action
  - accepts arguments (ie parameters)
- Eg in C:

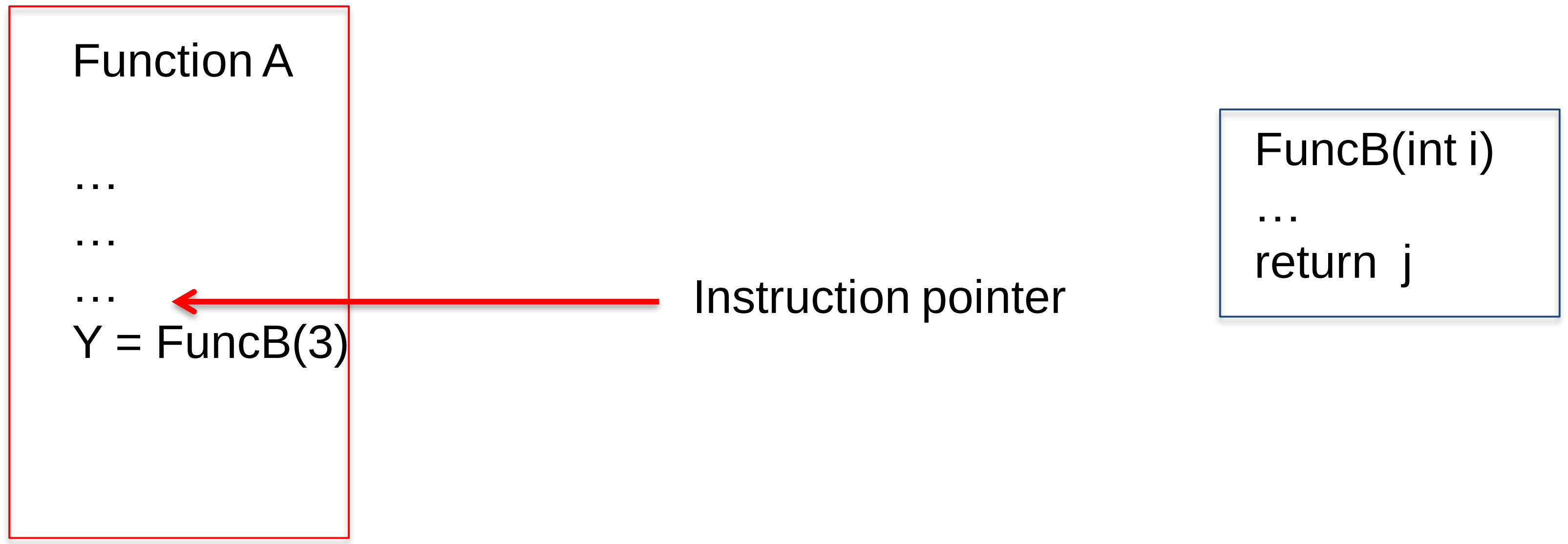
```
int add(int x, inty)
{
   int sum = x + y;
   return(sum);
}
```

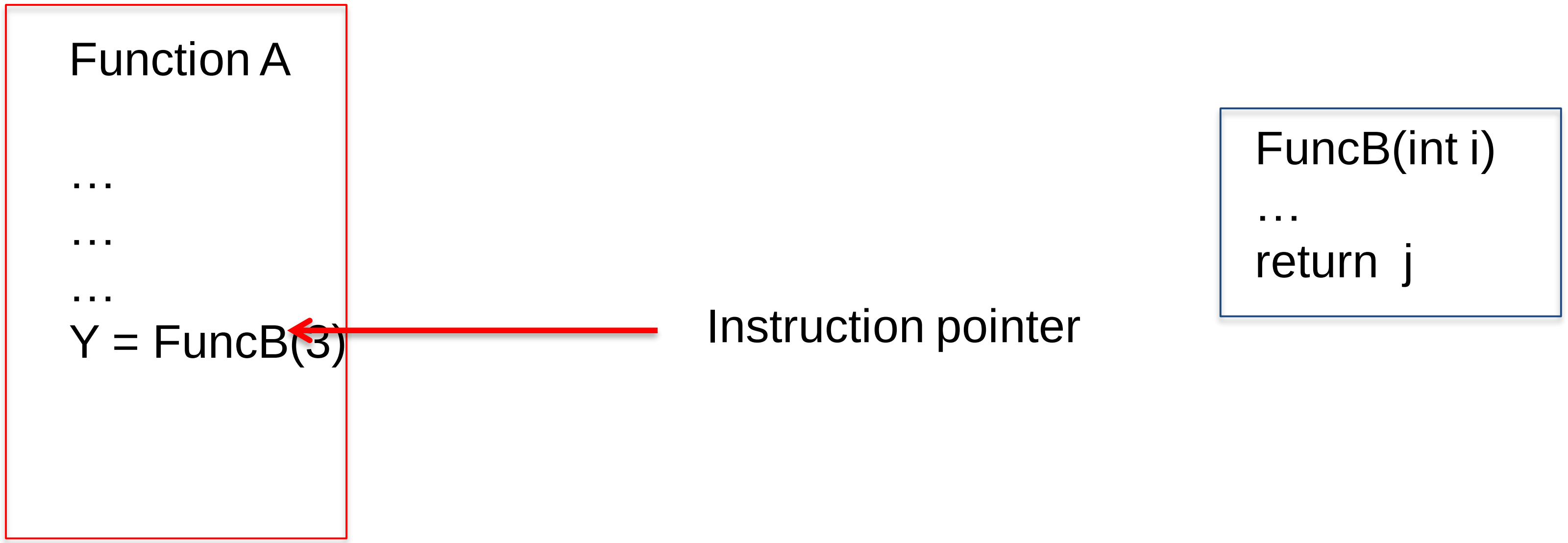


```
Function A

…
…
…
Y = FuncB(3)
```

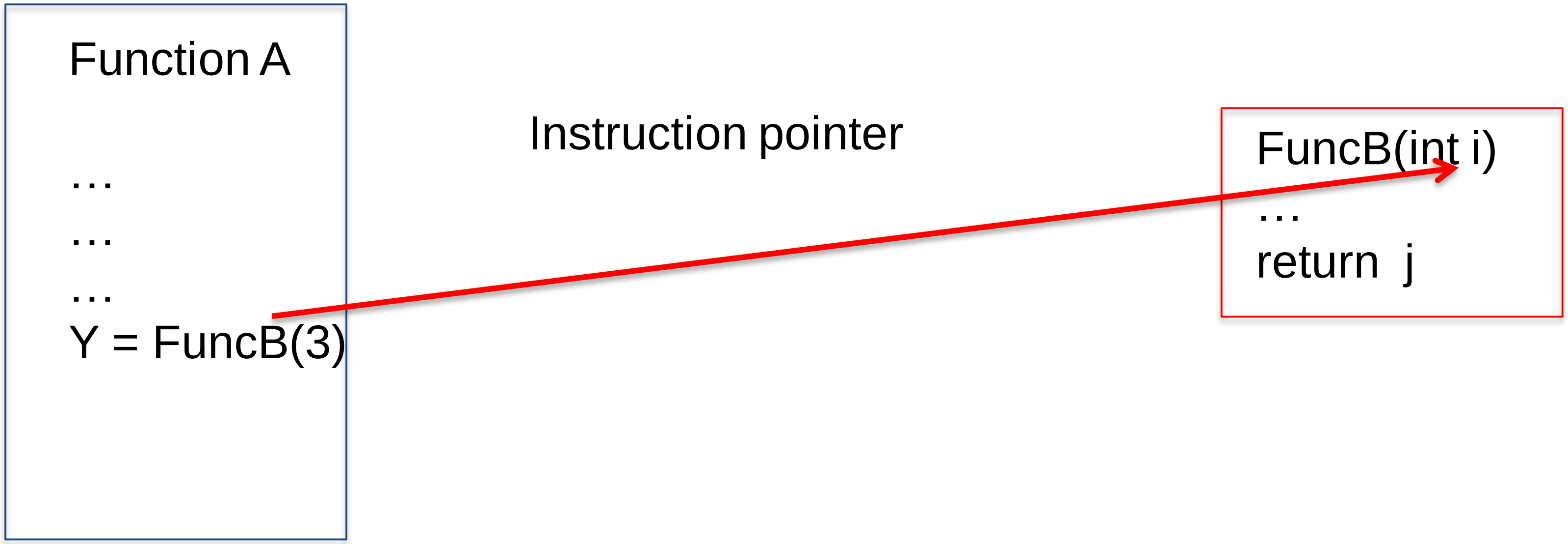
```
FuncB(int i)
…
return  j
```

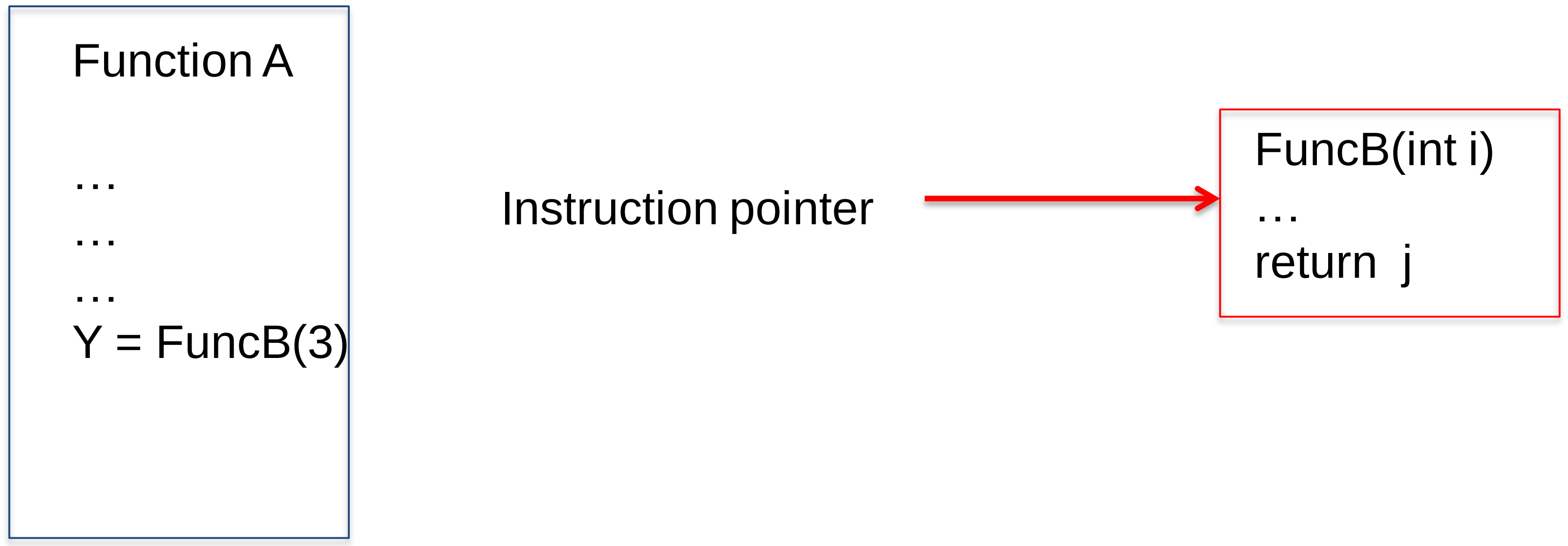

Function A
…
…
…
Y = FuncB(3)
Instruction pointer
FuncB(int i)
…
return j

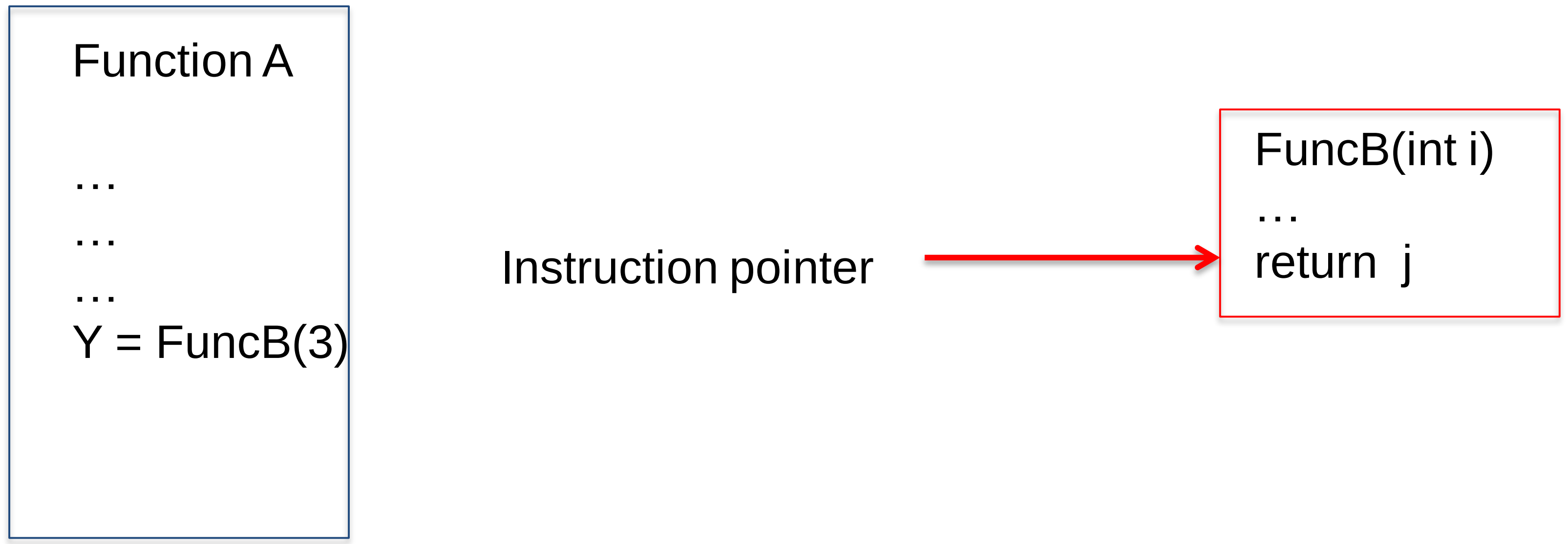

Function A
…
…
…
Y = FuncB(3)
Instruction pointer
FuncB(int i)
…
return j

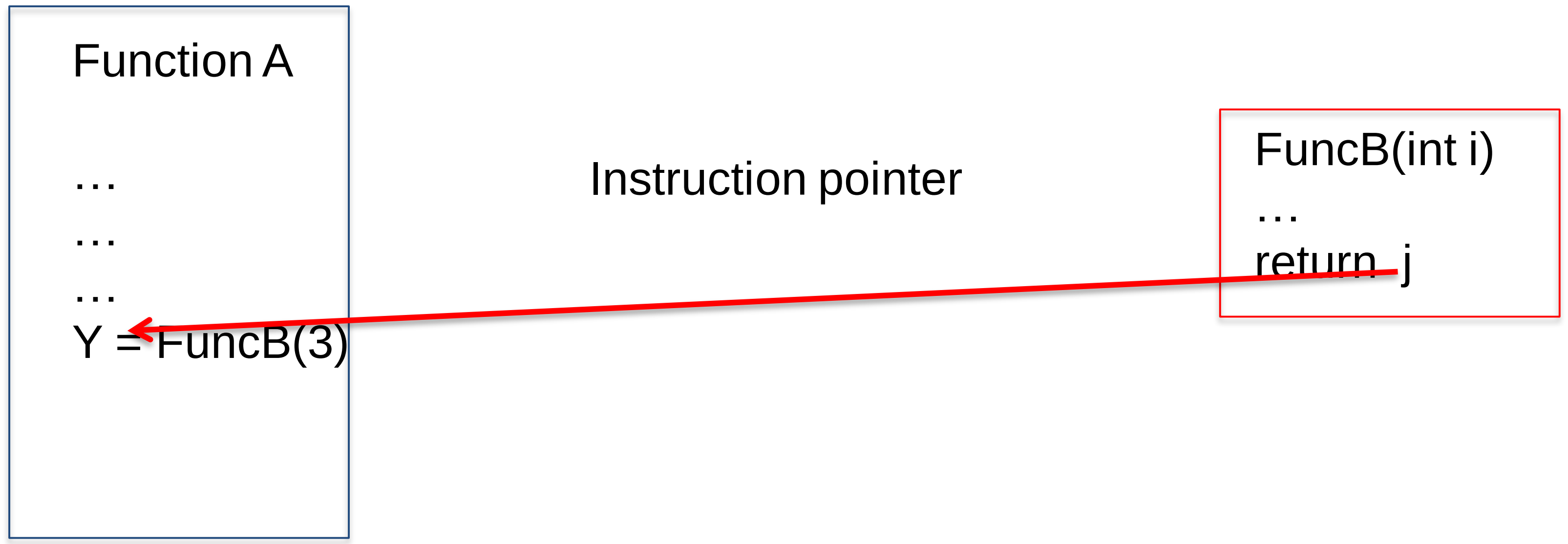

```
Function A

…
…
…
Y = FuncB(3)
```

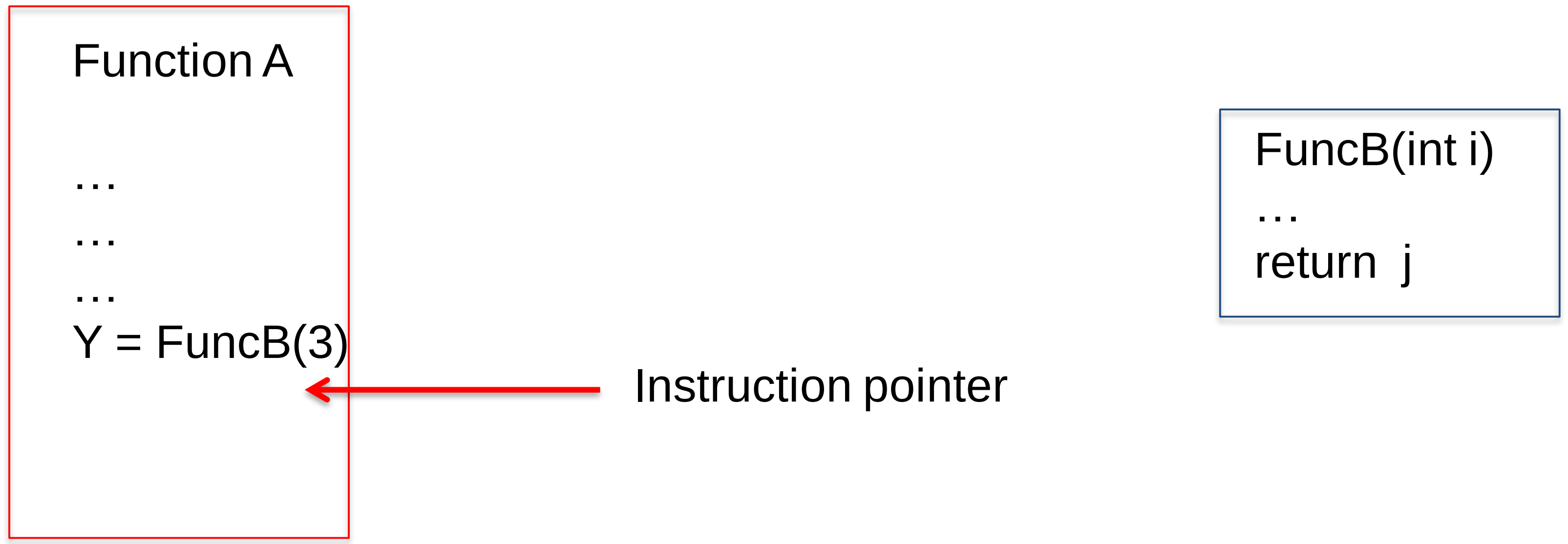
```
FuncB(int i)
…
return  j
```


Function A
…
…
…
Y = FuncB(3)
Instruction pointer
FuncB(int i)
…
return j


Function A
…
…
…
Y = FuncB(3)
Instruction pointer
FuncB(int i)
…
return j


Function A
…
…
…
Y = FuncB(3)
Instruction pointer
FuncB(int i)
…
return j


Function A
…
…
…
Y = FuncB(3)
Instruction pointer
FuncB(int i)
…
return j



```
Function A

…
…
…
Y = FuncB(3)
```

```
FuncB(int i)
…
return  j
```

Function Basics

- When a function is called:
  - Arguments need to be placed somewhere the function can access
  - program control shifts to the function's instructions
- When a function completes:
  - Return value needs to placed somewhere for the calling function to retrieve
  - program control shifts back to the instruction immediately after where it was called from
- Managing this requires a some *house keeping* needed *!*
  - High level programming languages hide most of this !
  - Not ASM!

FUNCTIONS IN ASM

- Not 'native' to assembly
  - We need to do a lot of the management ourselves
- Argument passing:
  - How do we pass arguments from one function to another
- Storing and recalling register values
  - each function we call will want to use the same registers (only 13 general purpose registers !)
  - How do we manage this ?
- Managing the program control
  - Jumping from one function to another, and then returning back !

FUNCTIONS IN ASM

- Not 'native' to assembly
  - We need to do a lot of the management ourselves
- Argument passing:
  - How do we pass arguments from one function to another
- Storing and recalling register values
  - each function we call will want to use the same registers (only 13 general purpose registers !)
  - How do we manage this ?
- Managing the program control
  - Jumping from one function to another, and then returning back !

Register Management

- **Application Binary Interface** (ABI) sets standard way of using ARM registers.
  - r0-r3 used for function arguments and return values
  - r4-r12 promised not to be altered by functions
  - **lr** and **sp** used for stack management
  - **pc** is the next instruction – we can use it to exit a function call

ABI

Register	Brief	Preserved	Rules
r0	Argument and result	No	r0 and r1 are used for passing the first two arguments to functions, and returning the results of functions. If a function does not use them for a return value, they can take any value after a function.
r1	Argument and result	No	
r2	Argument	No	r2 and r3 are used for passing the second two arguments to functions. There values after a function is called can be anything.
r3	Argument	No	

Calling functions

- By convention, the first two function arguments are loaded into r0 and r1.
- The next two are put into r2 and r3.
- The return value of the function is written into r0 and r1 (lowest word in r0).
- The function promises not to alter r4-r12.
- ... but suppose the function needs to use many registers to do calculations...

SUMMARY

- Functions are the building blocks of programs:
  - Organised, re-usable blocks of code
- Higher level programming languages have built in support for functions:
  - Not ASM!
- One thing we need to manage is register use
- Application Binary Interface (ABI) defines conventions for the use of registers
- Next lecture:
  - How do we store and recall register values ? With a stack of course !