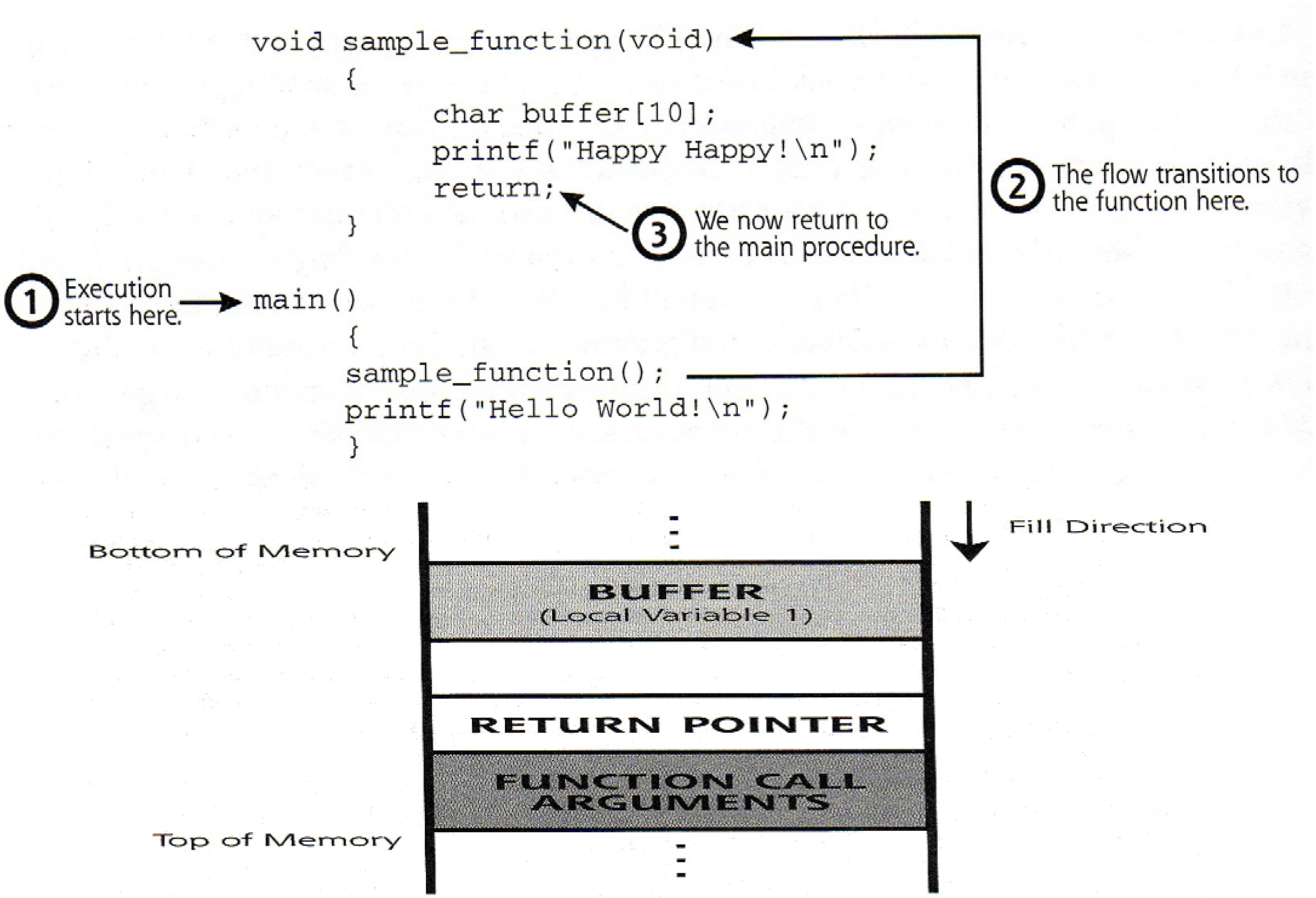
Stack-Based Buffer Overflow Attacks

Stack: A data structure that is used to store information associated with function calls on the computer.

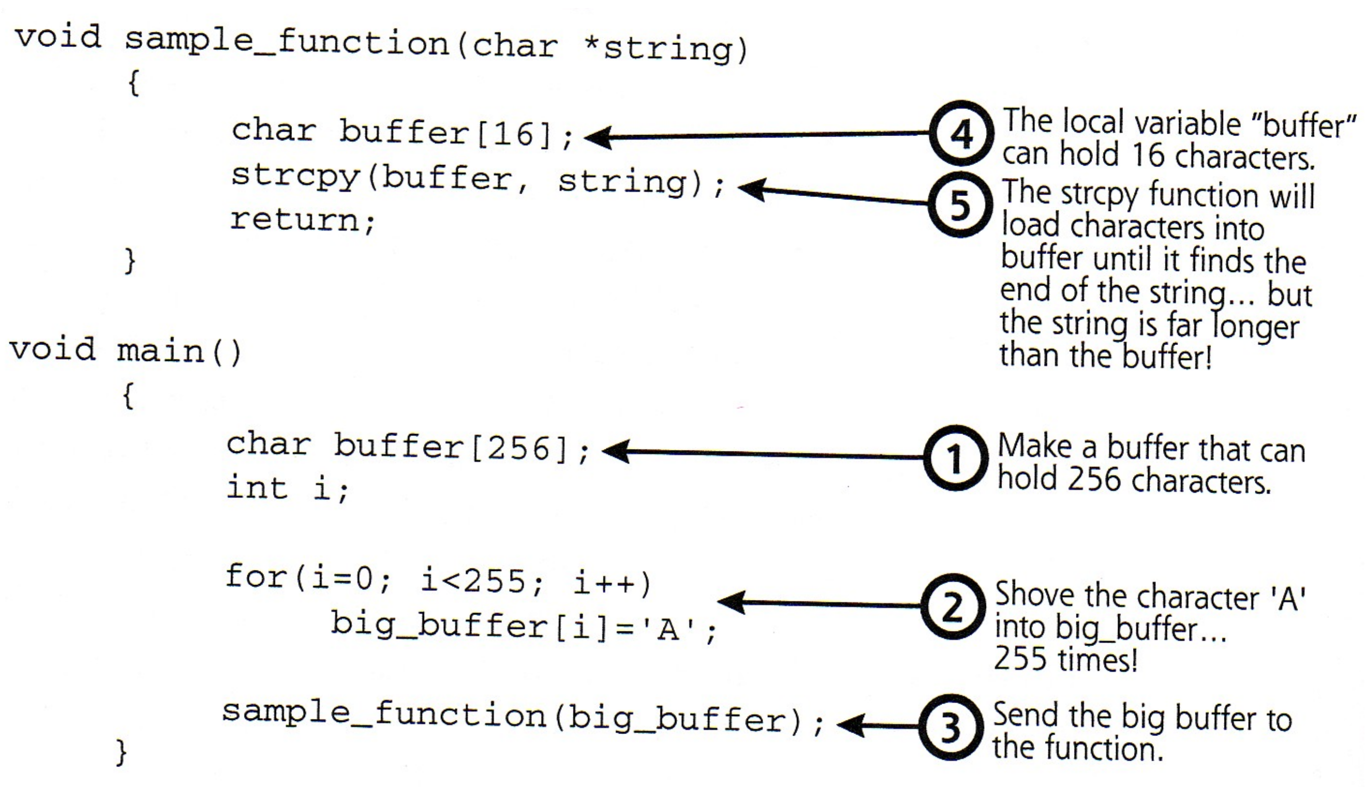


```
void sample_function(void)
    {
        char buffer[10];
        printf("Happy Happy!\n");
        return;
    }

main()
    {
    sample_function();
    printf("Hello World!\n");
    }
```

Stack-based Buffer Overflow?

Example: Putting 10 litres of stuff into a bag that will only hold 5 litter.



```
void sample_function(char *string)
     {
          char buffer[16];
          strcpy(buffer, string);
          return;
     }

void main()
     {
          char buffer[256];
          int i;

          for(i=0; i<255; i++)
               big_buffer[i]='A';

          sample_function(big_buffer);
     }
```

4 The local variable "buffer" can hold 16 characters.

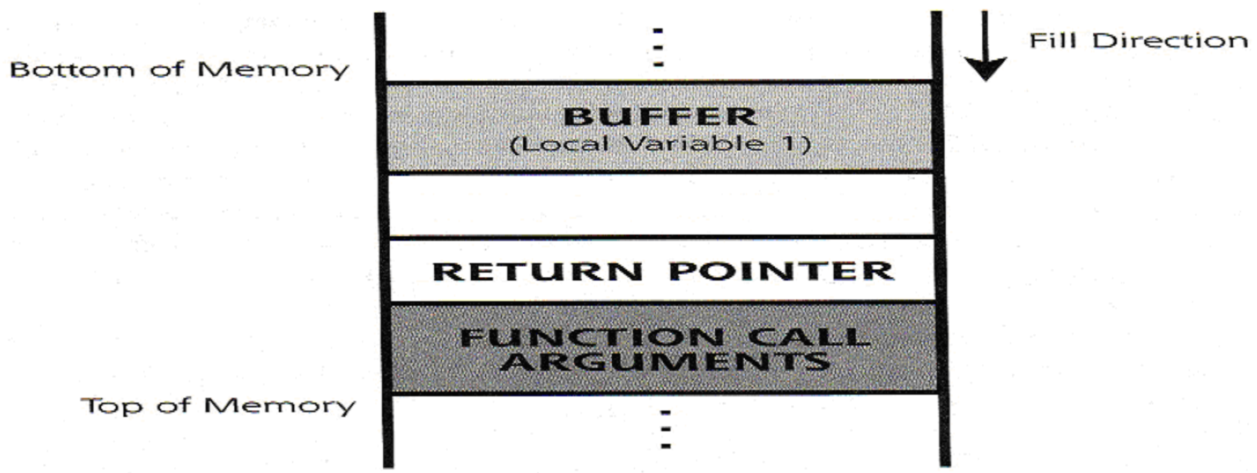
5 The strcpy function will load characters into buffer until it finds the end of the string... but the string is far longer than the buffer!

1 Make a buffer that can hold 256 characters.

2 Shove the character 'A' into big_buffer... 255 times!

3 Send the big buffer to the function.

- Strcpy doesn’t check the the size of string
- System allow strcpy to write far beyond where it is supposed to.



What happens to the stack when we do this?

It gets messed up.

- When the function finished executed the function, the return pointer is popped off. The address of next instruction will be "AAAA…..".
- Most likely, this is a bogus memory location, and the program will crash.