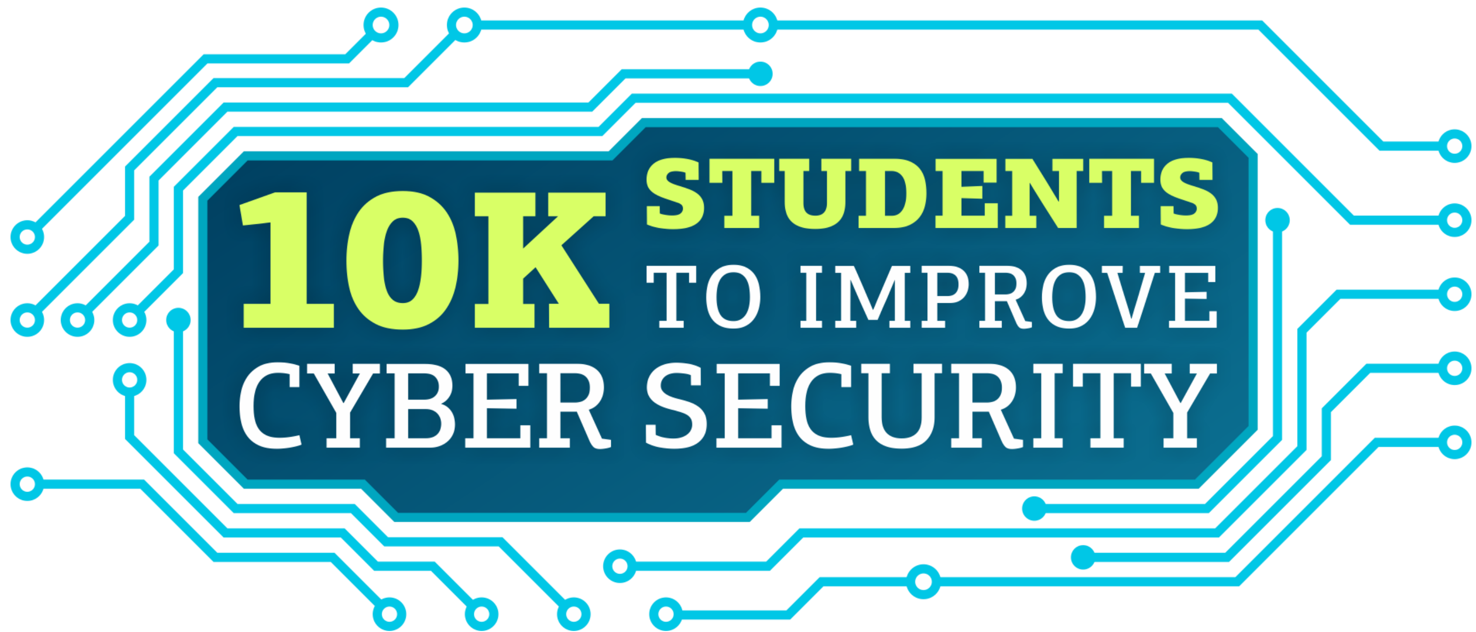

10K STUDENTS
TO IMPROVE
CYBER SECURITY

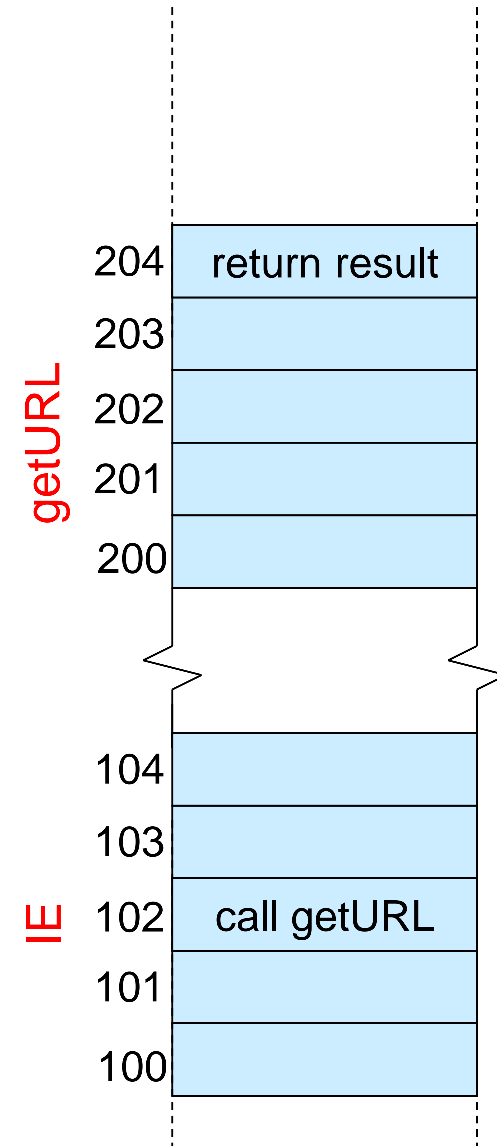
syssec

# Part I
# The Basic Idea

VU UNIVERSITY AMSTERDAM

# software

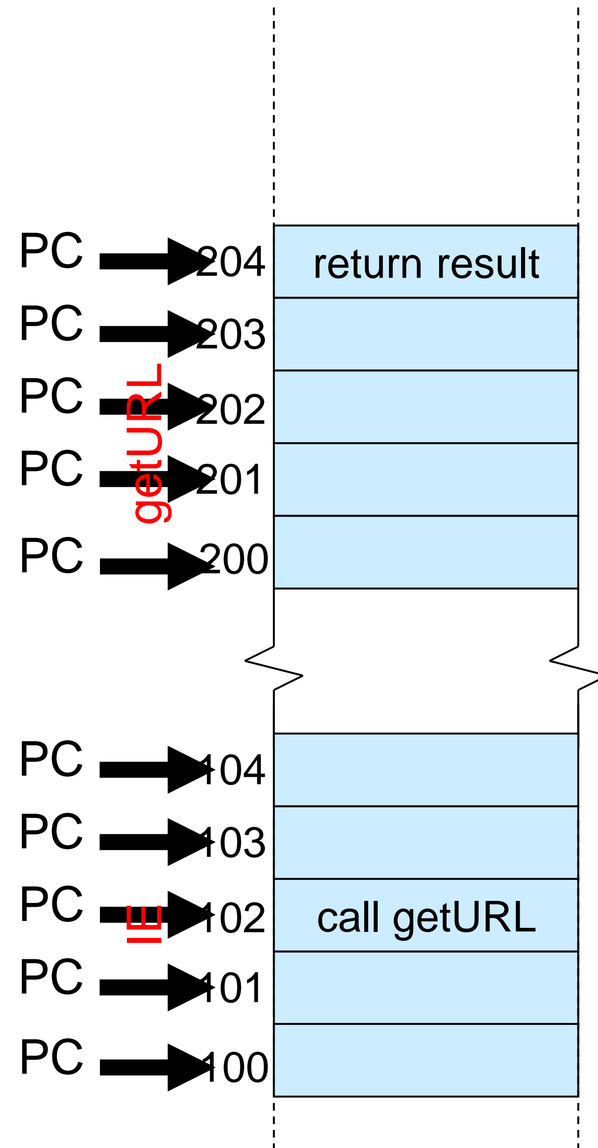
- sequence of instructions in memory
- logically divided in functions that call each other
  - function 'IE' calls function 'getURL' to read the corresponding page
- in CPU, the program counter contains the address in memory of the next instruction to execute
  - normally this is the next address (instruction 100 is followed by instruction 101, etc)
  - ***not so with function call***




VU UNIVERSITY AMSTERDAM

# software

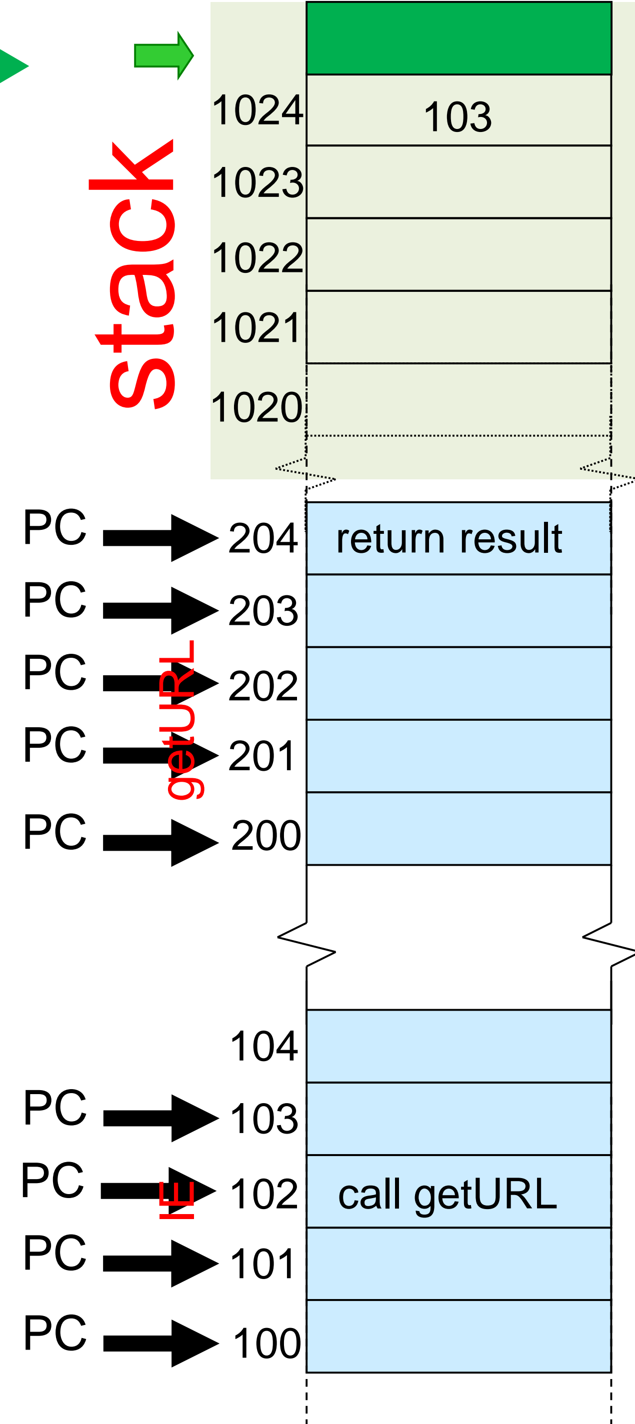
- sequence of instructions in memory
- logically divided in functions that call each other
  - function ‘IE’ calls function ‘getURL’ to read the corresponding page
- in CPU, the program counter contains the address in memory of the next instruction to execute
  - normally this is the next address (instruction 100 is followed by instruction 101, etc)
  - ***not so with function call***




VU UNIVERSITY AMSTERDAM

# software

- so how does our CPU know where to return?
  - it keeps administration
  - on a 'stack'




VU UNIVERSITY AMSTERDAM

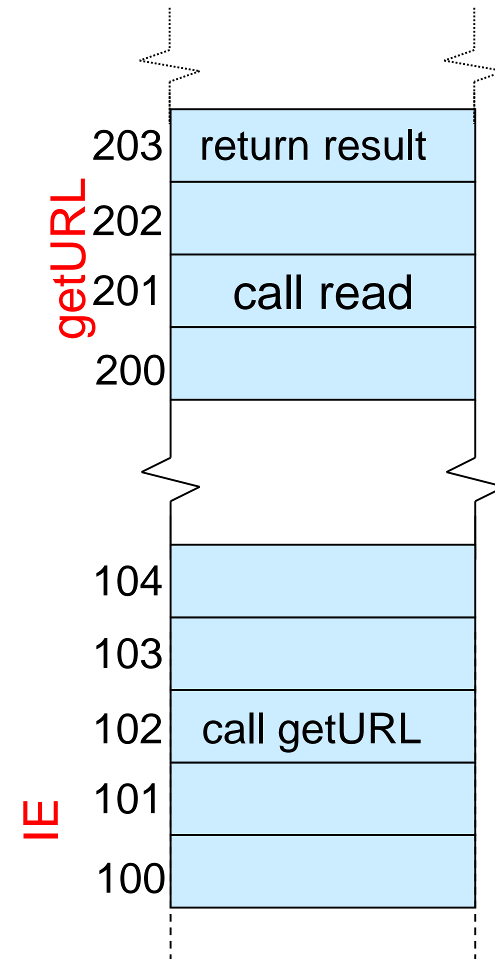
# real functions
# ➔ variables

```
getURL ()
{
  char buf[10];
  read(keyboard,buf,64);
  get_webpage (buf);
}
IE ()
{
  getURL ();
}
```

VU UNIVERSITY AMSTERDAM

# real functions
# ➔ variables

```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```

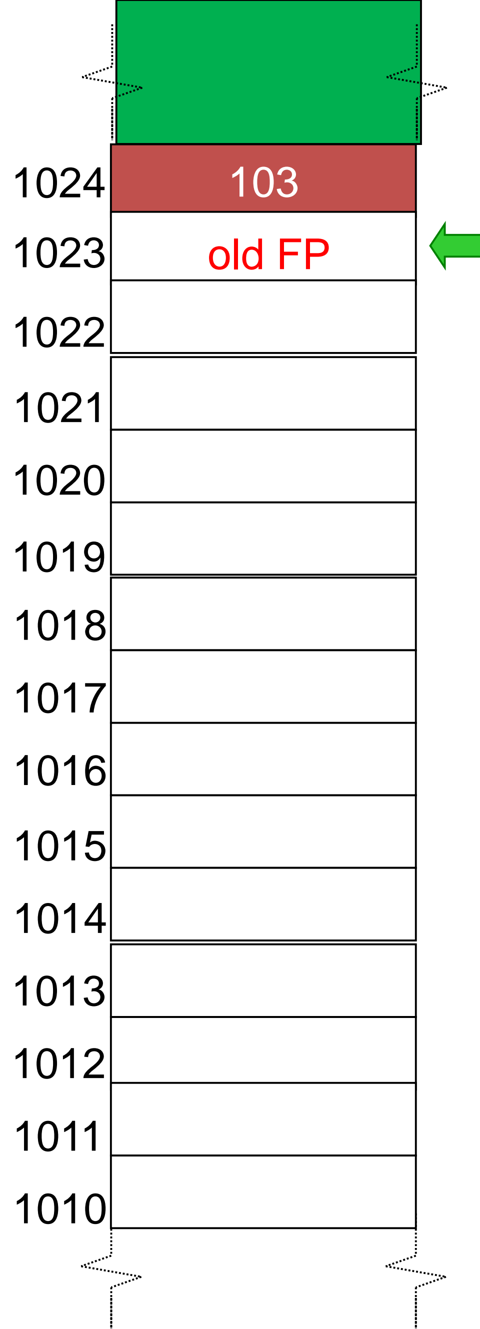


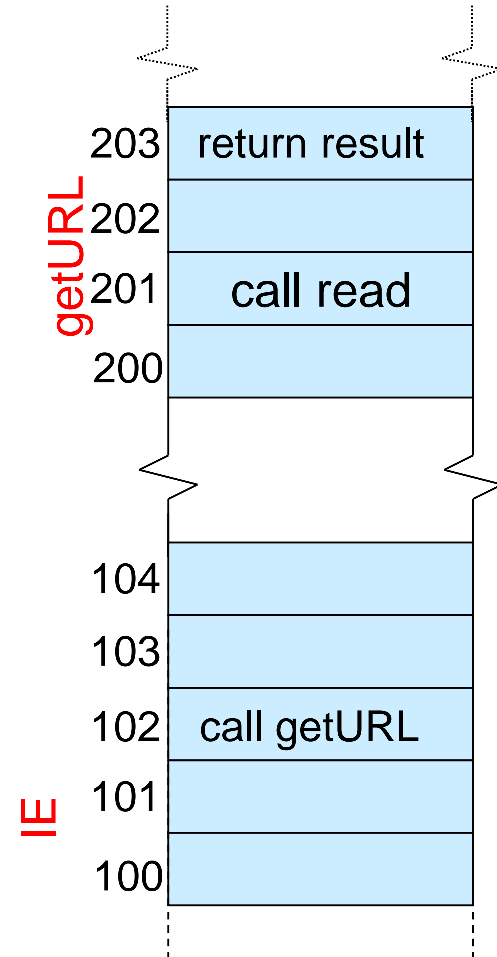

VU UNIVERSITY AMSTERDAM

real functions
→ variables

frame pointer (FP)
→ Points to start of this function's stack frame

```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```

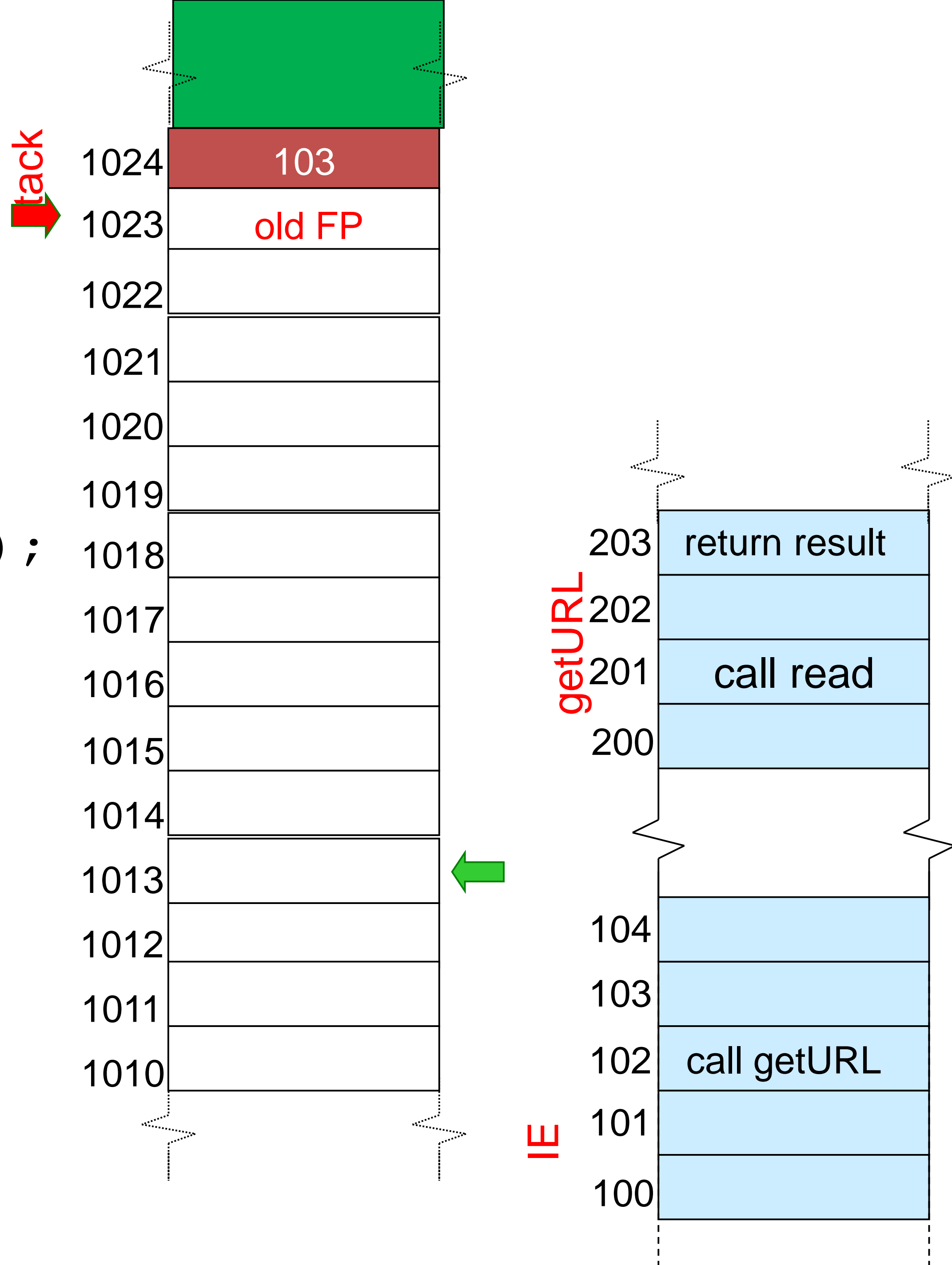



VU UNIVERSITY AMSTERDAM

# real functions
# ➔ variables

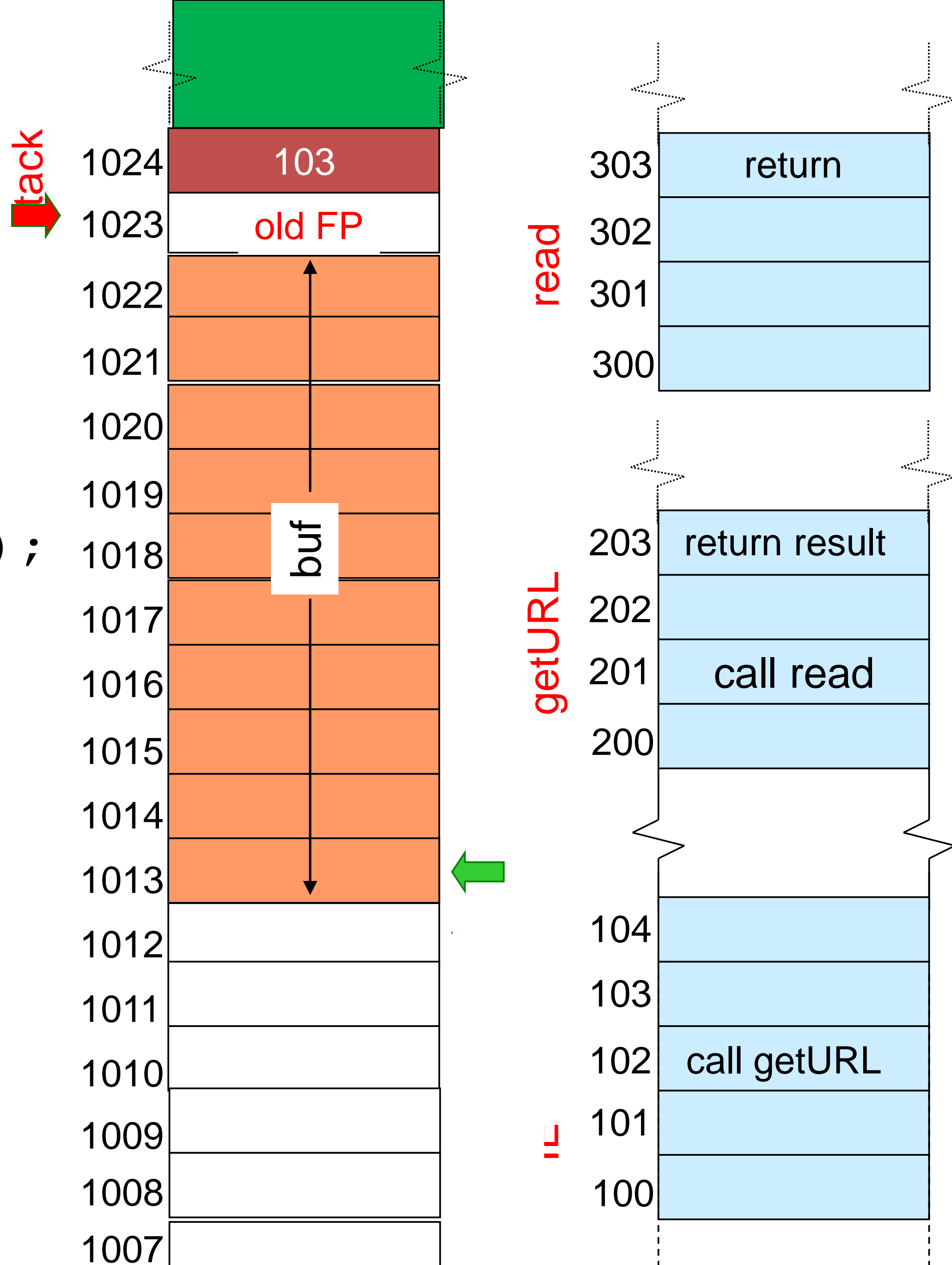
```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```




VU UNIVERSITY AMSTERDAM

# real functions
# ➔ variables

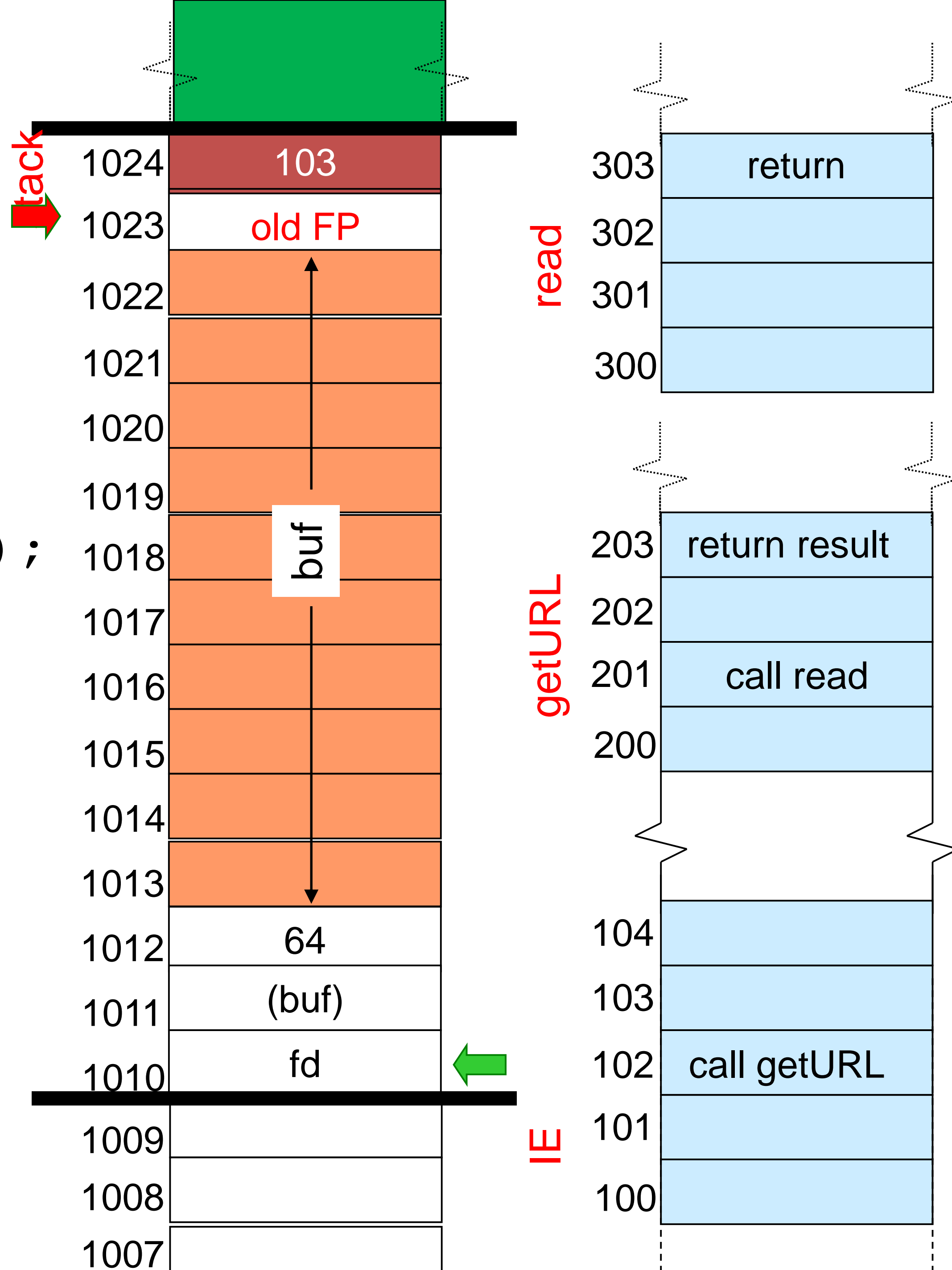
```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```




VU UNIVERSITY AMSTERDAM

# real functions
# → variables

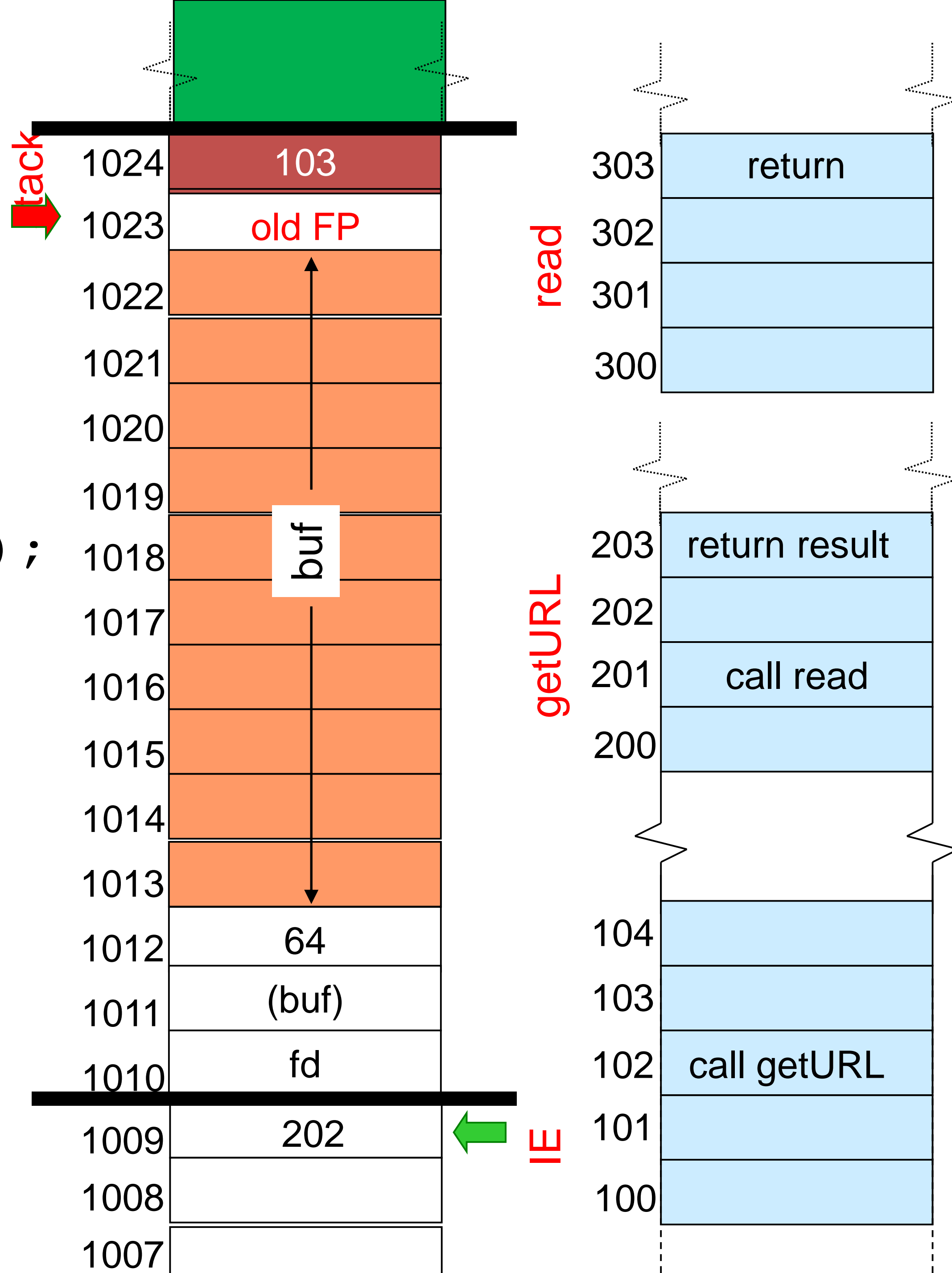
```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```




VU UNIVERSITY AMSTERDAM

# real functions
# ➔ variables

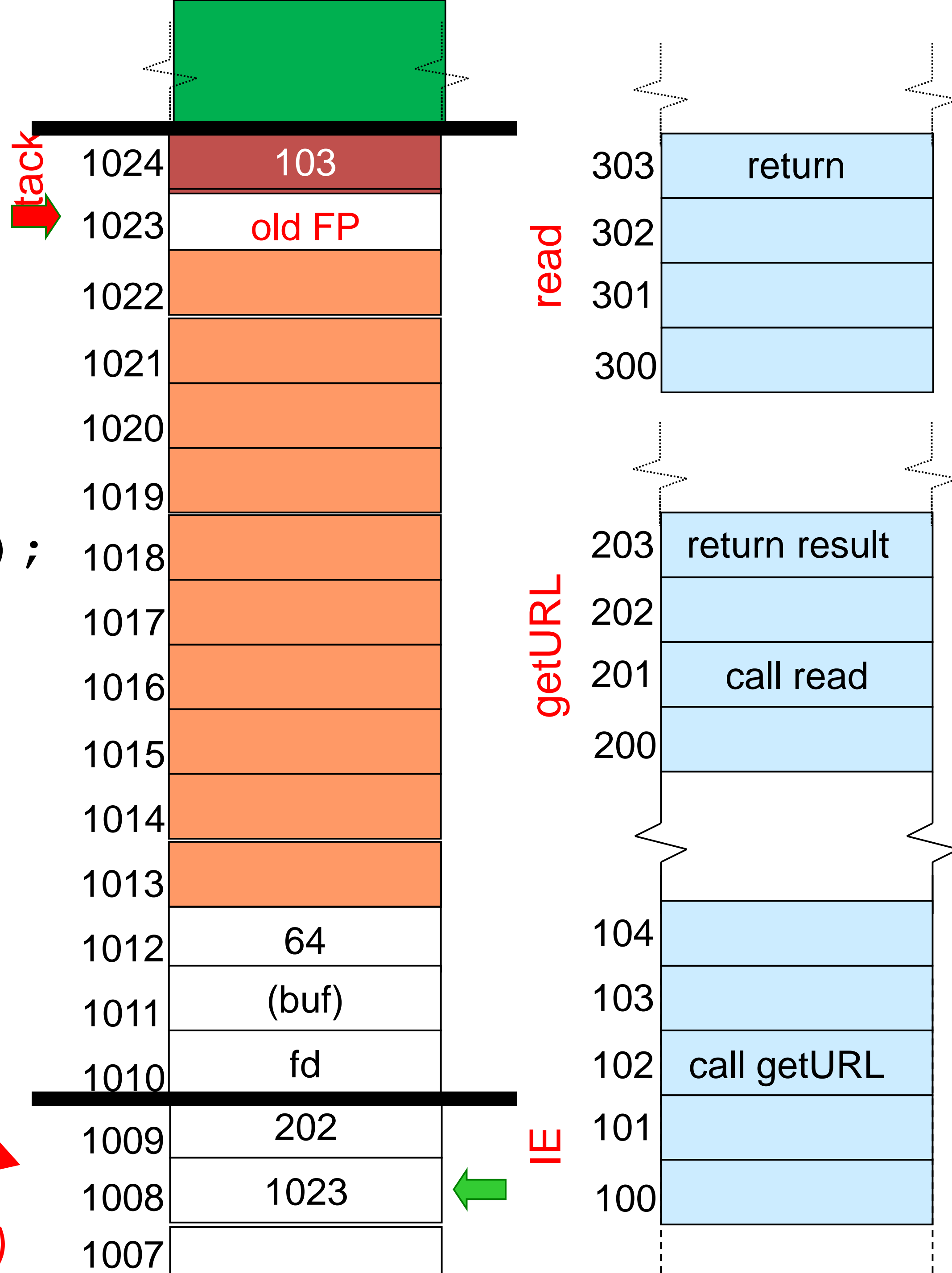
```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```




VU UNIVERSITY AMSTERDAM

# real functions
# ➔ variables

```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```



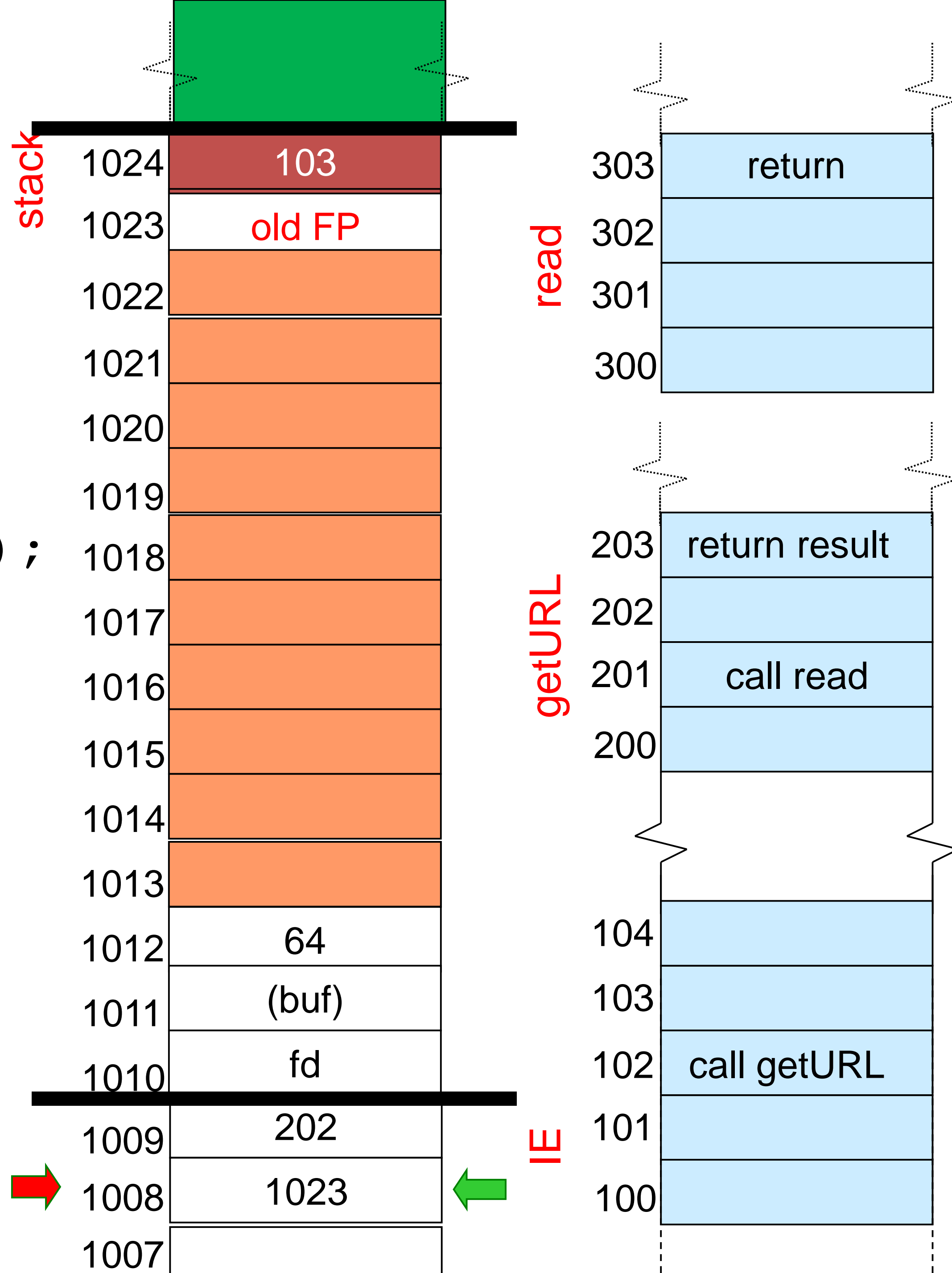

frame pointer (FP)

VU UNIVERSITY AMSTERDAM

# real functions
# ➔ variables

```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```

on "return from read"



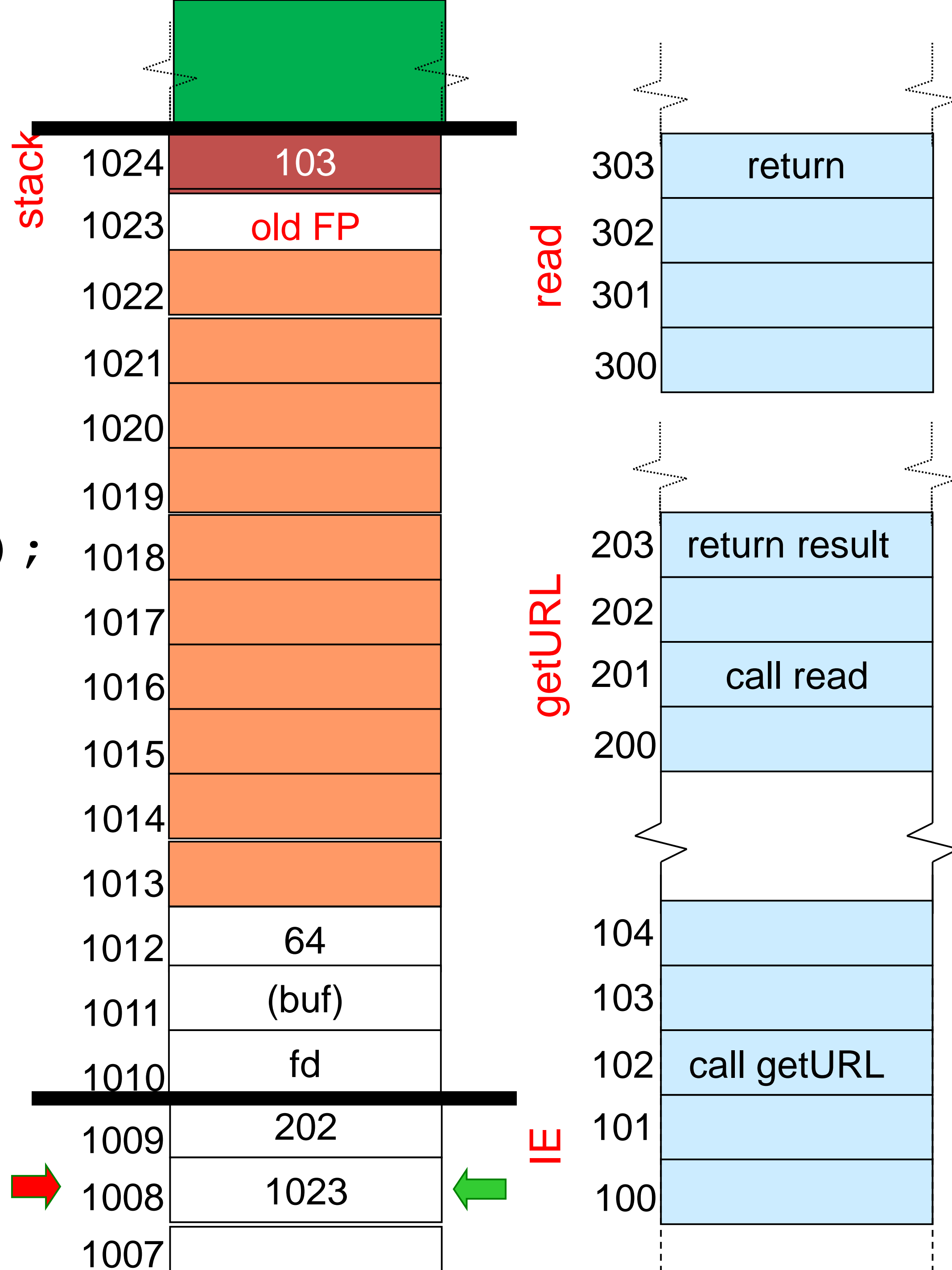

VU UNIVERSITY AMSTERDAM

real functions

➔ variables

```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```

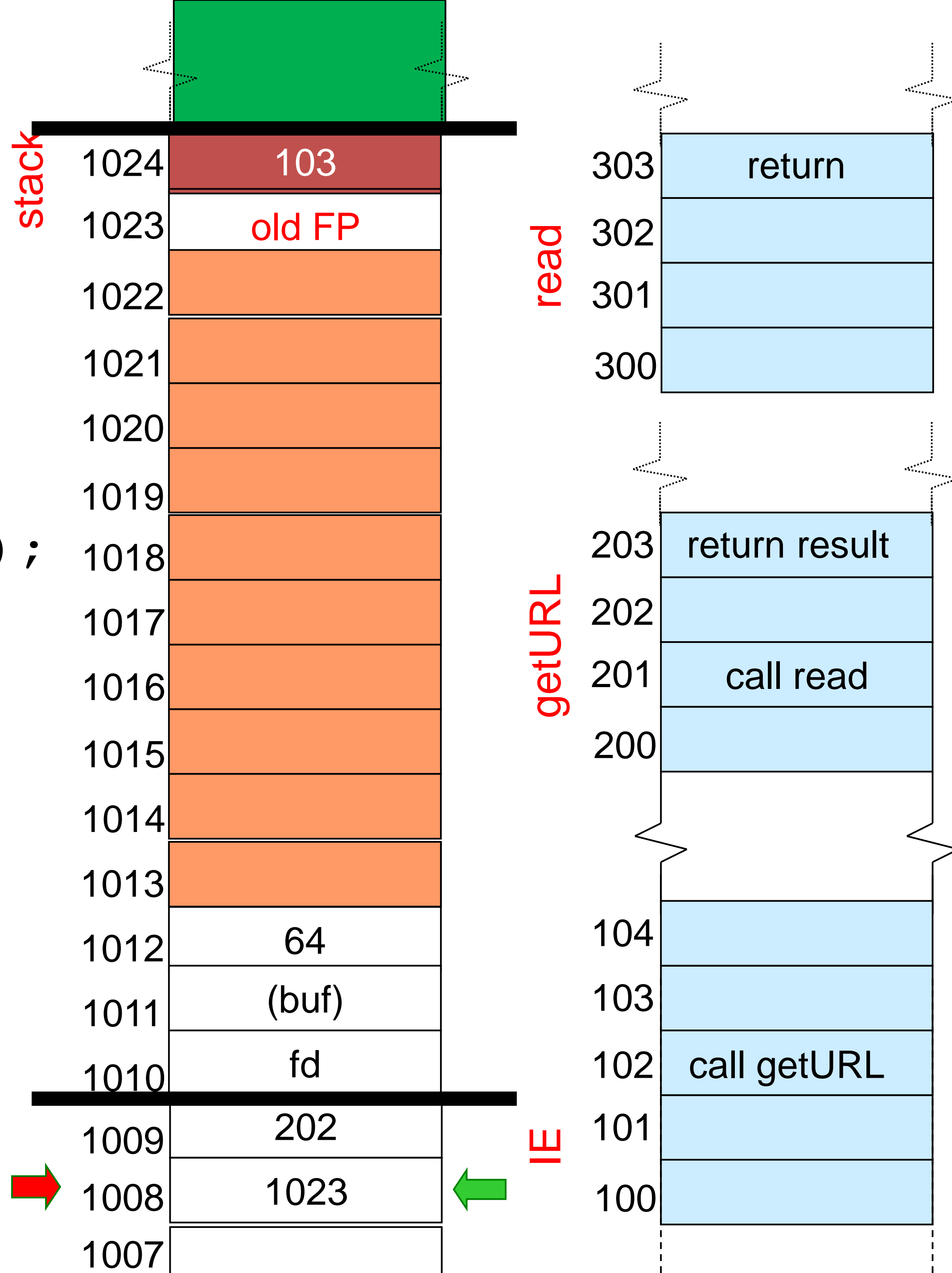
POP




VU UNIVERSITY AMSTERDAM

# real functions
# → variables

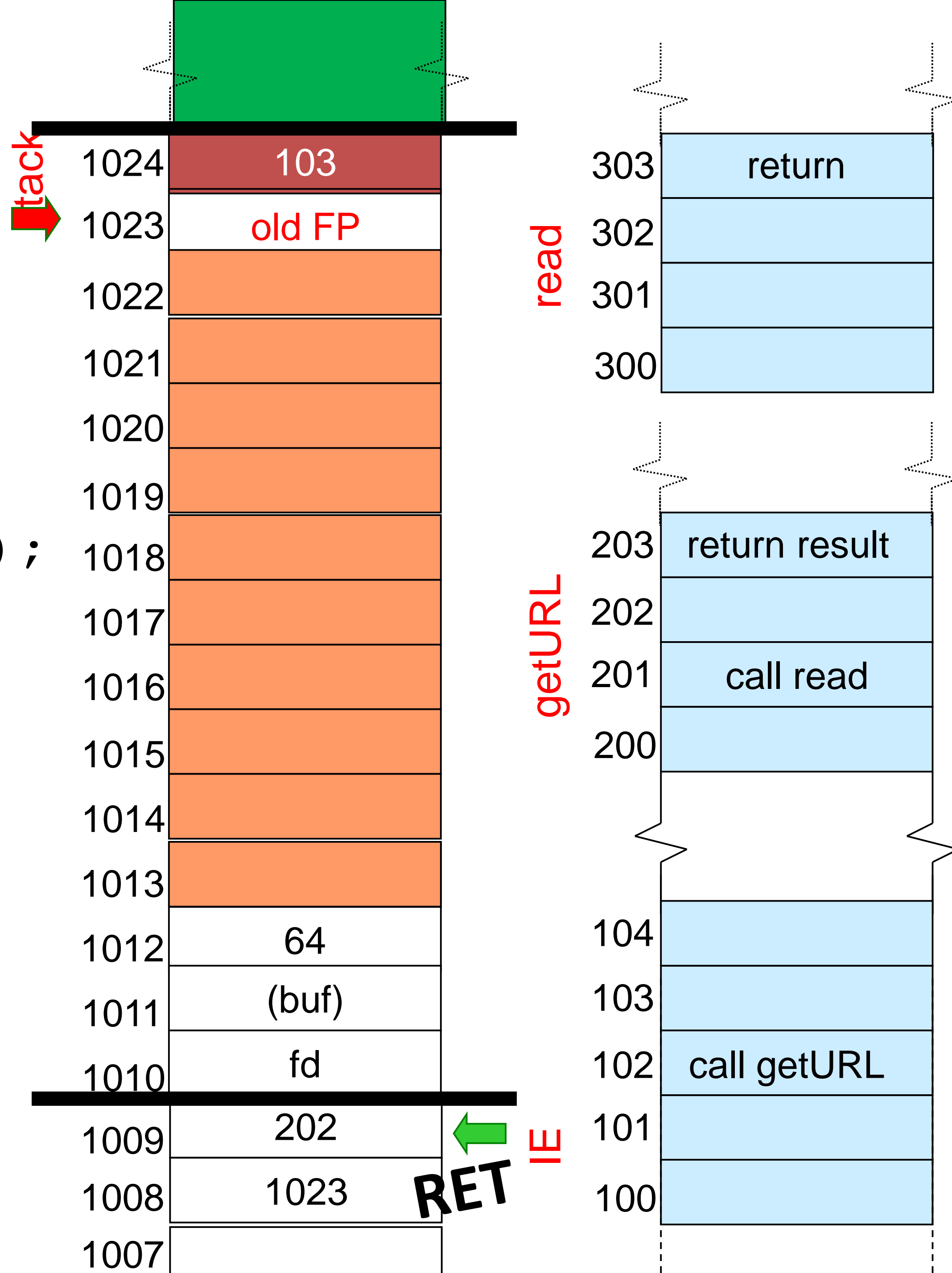
```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```




VU UNIVERSITY AMSTERDAM

# real functions
# ➔ variables

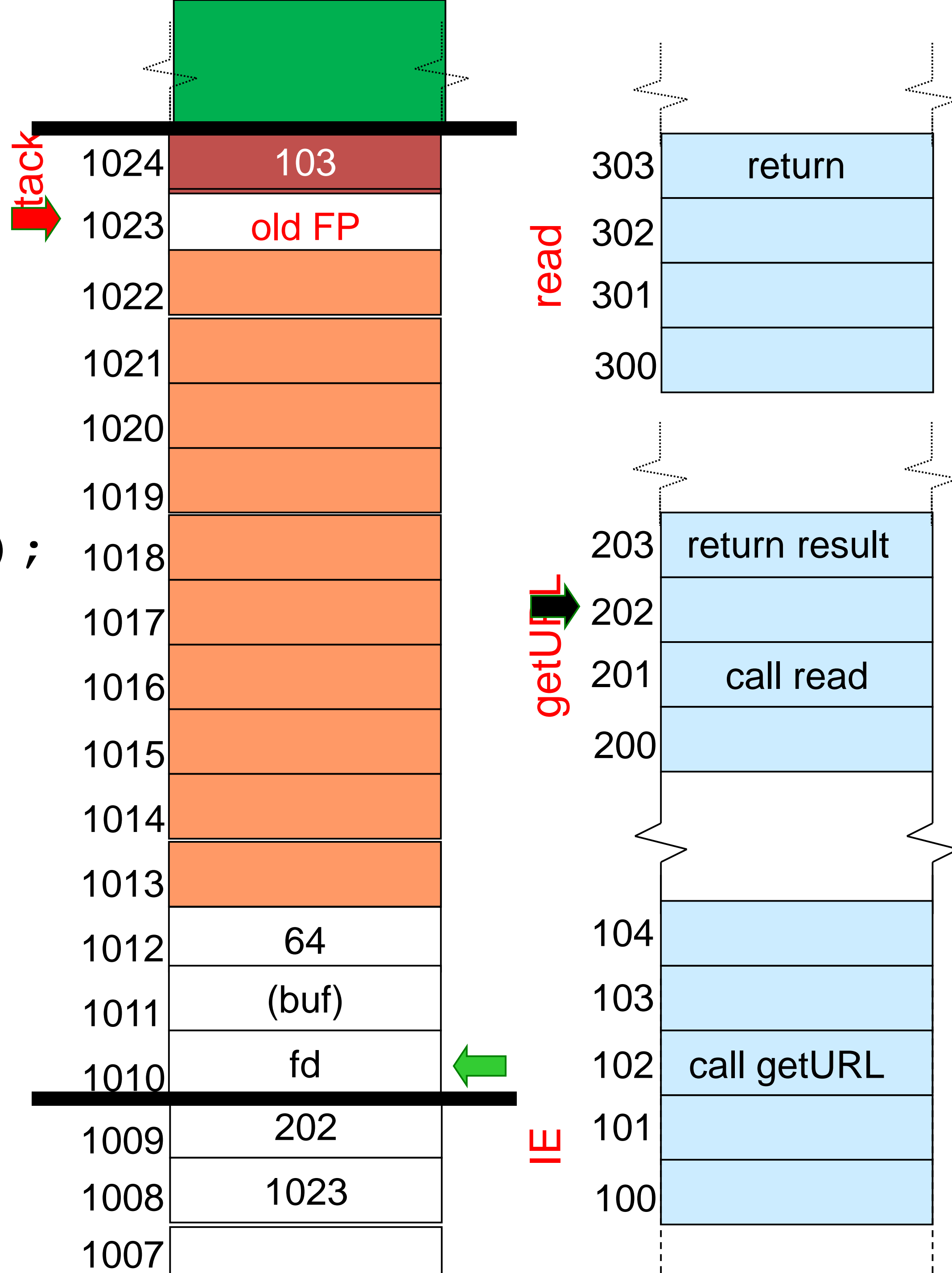
```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```




VU UNIVERSITY AMSTERDAM

# real functions
# ➔ variables

```
getURL ()
{
   char buf[10];
   read(keyboard,buf,64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```

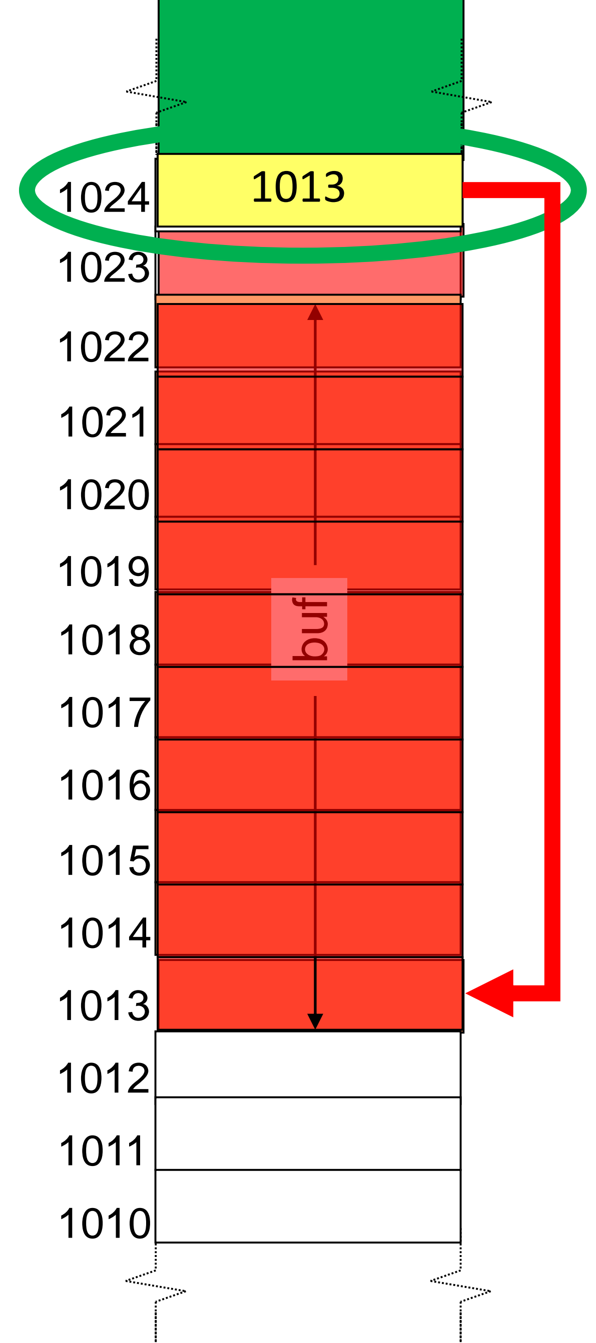



UNIVERSITY AMSTERDAM

# Where is the vulnerability?

# Exploit

```
getURL ()
{
   char buf[10];
   read(keyboard, buf, 64);
   get_webpage (buf);
}
IE ()
{
   getURL ();
}
```




VU UNIVERSITY AMSTERDAM

# You may also overwrite other things

- For instance:
  - Other variables that are also on the stack
  - Other addresses
  - Etc.

# Memory Corruption

## Final words Part I

- We have sketched only the most common memory corruption attack
  - many variations, e.g.:
    - heap ← → stack
    - more complex overflows
    - off-by-one
- But there are others also
  - integer overflows
  - format string attacks
  - double free
  - etc.
- Not now, perhaps later…



*different kinds