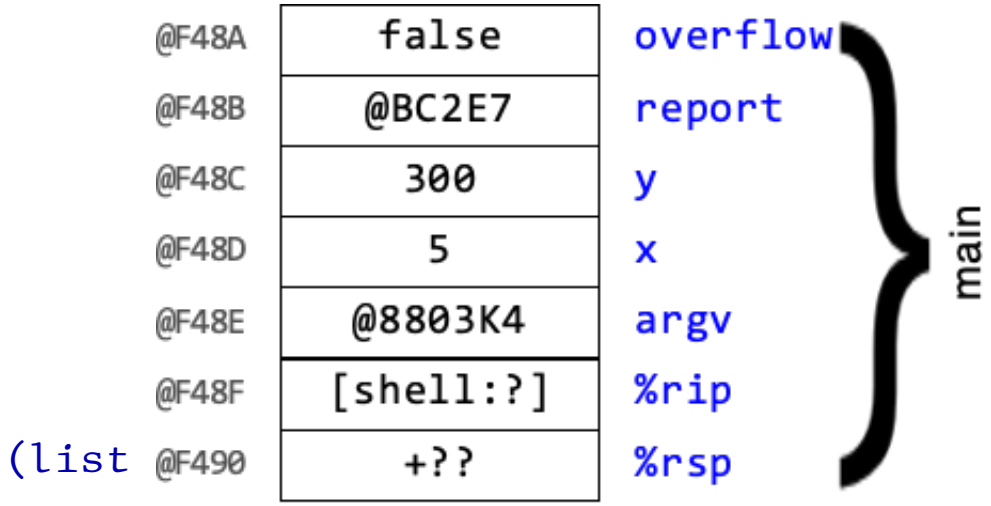
Welcome to DrRacket, version 8.15 [cs].
Language: racket, with debugging; memory limit: 128 MB.

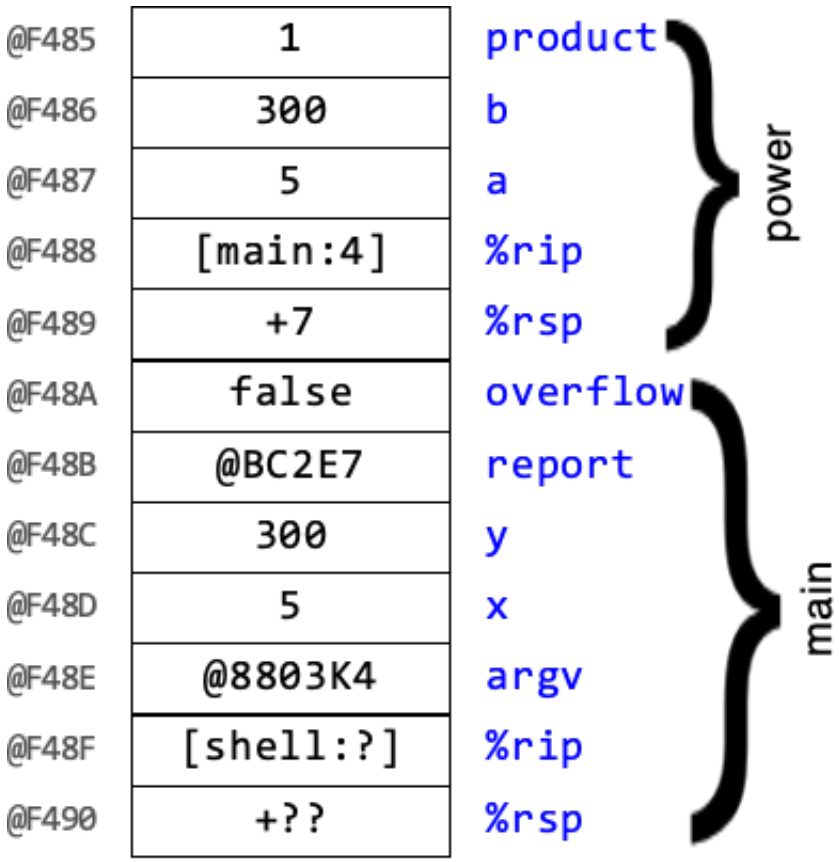
```
 A Java program, and how information is placed on the
run-time stack at three moments in time.

The purpose is to help understand how the run-time
stack holds the current function's local variables
(incl. its parameters), as well as a vital piece of
book-keeping: what program-instruction to resume at,
when the current helper function finishes.

This is important cybersecurity, since it explains how
an attacker, if they can get a stack overflow to both
(a) place malicious code onto the stack, *and* (b)
overwrite the return-instruction-pointer so that the
program 'returns' to that malicious code rather than
the real return-site, then the attacker has achieved
"running of arbitrary code".
It's important to programmers, since knowing which data
are on the heap vs the stack explains why local
variables are local, and what overhead function-calls
might incur.

Notes:
- "%rip" is a local system variable for "return
instruction pointer" -- where to resume the program at,
  when finishing the current helper function.
- "%rsp" is a local system variable for "return stack
pointer" -- where to adjust the top-of-stack to,
  when finishing the current helper function.

"The sample Java program:"
class StackDiagram {
```

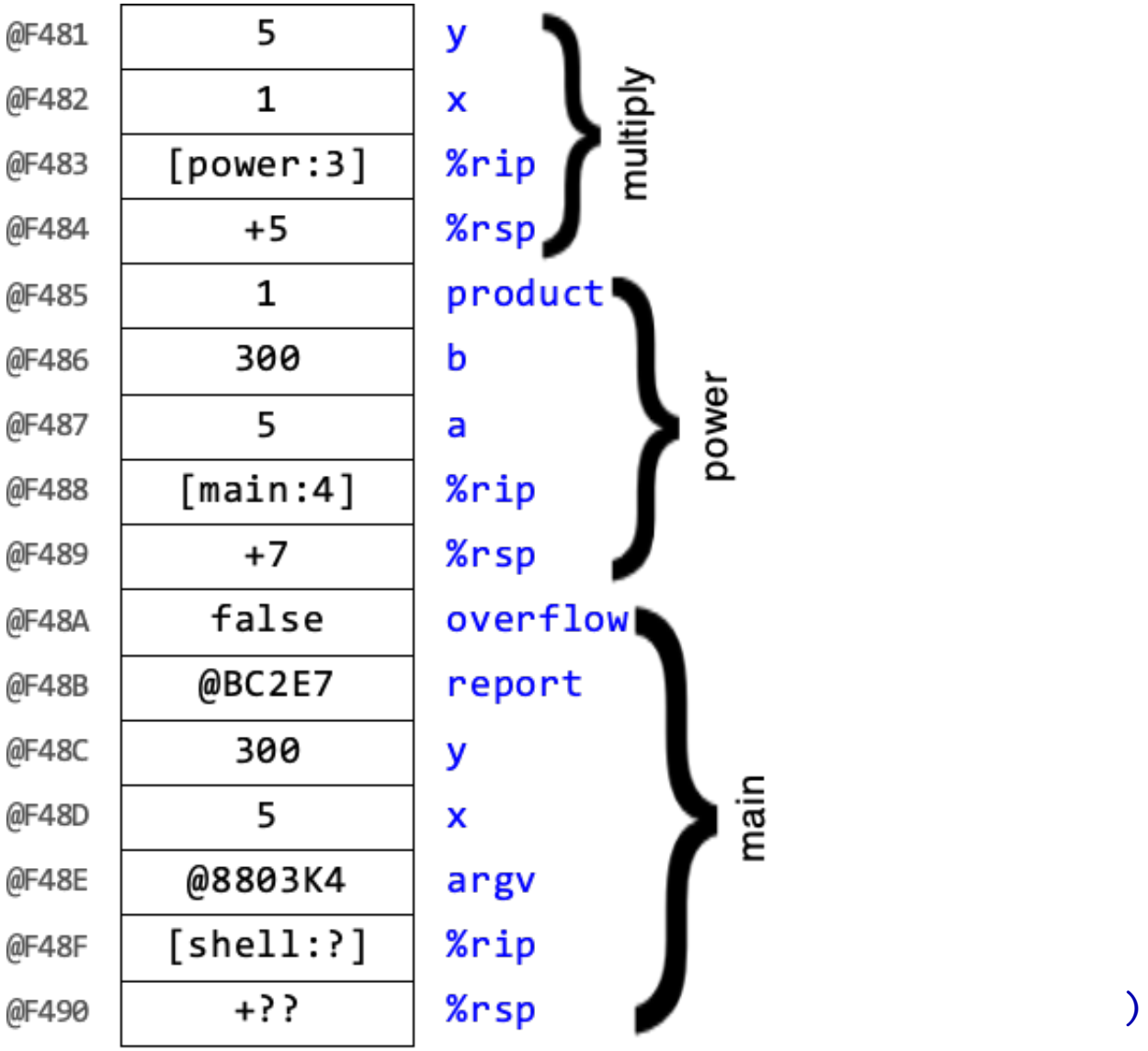
```
  public static void main( String... argv ) {
    System.out.printf("This program verifies whether 5
to the 300th power is bigger than 0.\n");
    int x = 5;
    int y = 300;
    String report = "Yes!";
    boolean overflow;

    if (power(x,y) >= 0) {
      System.out.printf("%s\n",report);
      overflow = false;
    }
    else {
      System.out.printf( "It's not!  Hmmm; overflow?\n"
);
      overflow = true;
    }
    System.exit(overflow ? 1 : 0);  // indicate an
error, to the shell / caller.
  }


  // Return a^b.
  //
  static int power(int a, int b) {
    int product = 1;
    while (b!=0) {
      product = multiply(product, a);
      b--;
    }
    return product;
  }
```

```
  // Return x*y.
  //
  static int multiply(int x, int y) {
    return x*y;
  }

}
""
"The stack, as main first calls power first and it in
turn calls multiply the first time:"
```




```
(list
```


@F485
1
product
@F486
300
b
@F487
5
a
@F488
[main:4]
%rip
@F489
+7
%rsp
power
@F48A
false
overflow
@F48B
@BC2E7
report
@F48C
300
y
@F48D
5
x
@F48E
@8803K4
argv
@F48F
[shell:?]
%rip
@F490
+??
%rsp
main


@F481
5
y
@F482
1
x
@F483
[power:3]
%rip
@F484
+5
%rsp
multiply
@F485
1
product
@F486
300
b
@F487
5
a
@F488
[main:4]
%rip
@F489
+7
%rsp
power
@F48A
false
overflow
@F48B
@BC2E7
report
@F48C
300
y
@F48D
5
x
@F48E
@8803K4
argv
@F48F
[shell:?]
%rip
@F490
+??
%rsp
main

)

>