Phase 4 Overview

6.1120

Fall 2025

Phase 4: Virtual Machine

```
f = fun(y) {
  x = y + 2;
};
f(1);
```

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure 0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```

Deliverables

- Your VM compiler and interpreter
  - Must conform to language specification from Phase 2
  - Must be able to build your code by running `./build.sh`
- VM compiler must be invoked by `./run.sh compile <input>.mit -o <output>.mitbc` and take MITScript source code as input
- VM interpreter binary should be invoked by `./run.sh vm <input>.mitbc`

Deliverables

- Ten test cases
- Documentation (should also describe how work was divided up)
- Bytecode tests
- End-to-end tests
- Memlimit tests

Due Dates

- Checkpoint #1: October 24 @ 10 pm
  - Autograder score on bytecode tests (public and private) will determine extra credit (up to 5%)
- Checkpoint #2: October 31 @ 10 pm
  - Autograder score on both bytecode and e2e tests will determine extra credit (up to 5%)
- Final due date: November 7 @ 10 pm
  - In addition to bytecode and e2e tests, we will run your submission on memlimit tests

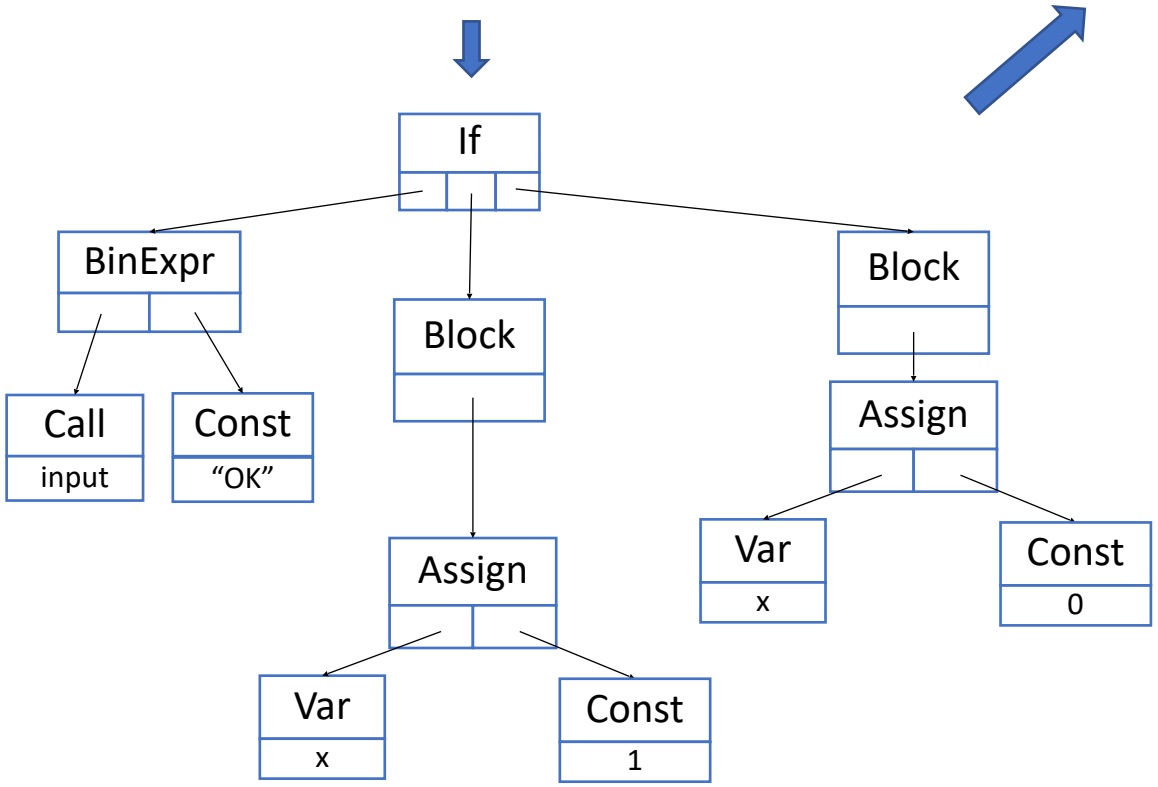
Tips

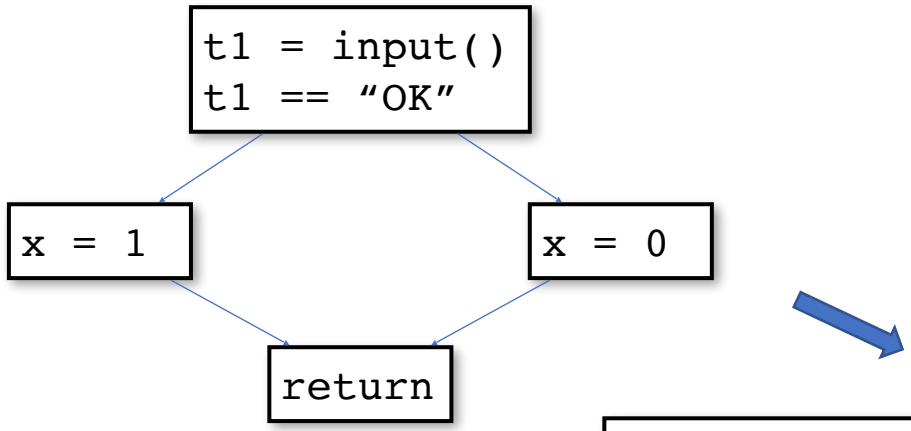
- Don’t optimize prematurely!
  - Always profile and justify optimizations (covered more in R9)
- Be a good friend and teammate
  - Use branches and PRs internally
  - Write good code and comment well
  - Communicate and be accountable

Make sure you start as soon as possible (this phase is very long)!

From AST to Bytecode

```
if (input() == “OK”) {
    x = 1;
} else {
    x = 0;
}
```






```
function {
  functions = [],
  constants = [
    None, true, false, "OK", 1, 0
  ],
  names = [print, input, intcast,
x],
  instructions = [
    load_global  1
    call         0
    load_const   3
    eq
    not
    if           4
    load_const   4
    store_global 3
    goto         3
    load_const   5
    store_global 3
    load_const   0
    return
  ]
}
```

Common mistakes

```
f = fun(y) {
  x = y + 2;
};
f(1);
```

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func     0
    alloc_closure 0
    store_global  0
    load_global   0
    load_const    2
    call 1
    pop
    load_const 0
    return
  ]}
```

- Improperly indexing into locals, globals, or other arrays
- Directly using a constant instead of indexing
- Incorrect pushing and handling of references

I present to you… instructions.h!

```
enum class Operation {
  // Description: push a constant onto the operand stack
  // Mnemonic:    load_const i
  // Operand 0:   index of constant in enclosing function's list of constants
  // Stack:       S =>  S :: f.constants()[i]
  LoadConst,

  // Description: push a function onto the operand stack
  // Mnemonic:    load_func i
  // Operand 0:   index of function  in enclosing function's list of functions
  // Stack:       S => S :: f.functions()[i]
  LoadFunc,

  // Description: load value of local variable and push onto operand stack
  // Mnemonic:    load_local i
  // Operand 0:   index of local variable to read
  // Stack:       S => S :: value_of(f.local_vars[i])
  LoadLocal,

  // Description: store top of operand stack into local variable
  // Mnemonic:    store_local i
  // Operand 0:   index of local variable to store into
  // Operand 1:   value to store
  // Stack:       S :: operand 1==> S
  StoreLocal,

  // Description: load value of global variable
  // Mnemonic:    load_global i
  // Operand 0:   index of name of global variable in enclosing function's names
  // list Stack:       S =>  S :: global_value_of(f.names[i])
  LoadGlobal,

  // ...
```

MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```

IP	Op Stack	Locals
Stack Frame		

Globals
f

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure 0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

Heap

MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```

IP	Op Stack	Locals
1		
Stack Frame		

Globals
f

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure 0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

Heap

MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```

IP	Op Stack	Locals
2		
Stack Frame	•	

Globals
f

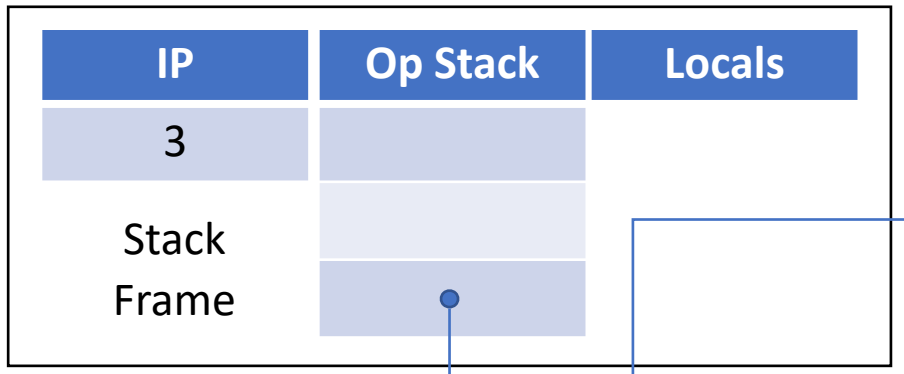
```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure 0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

Heap

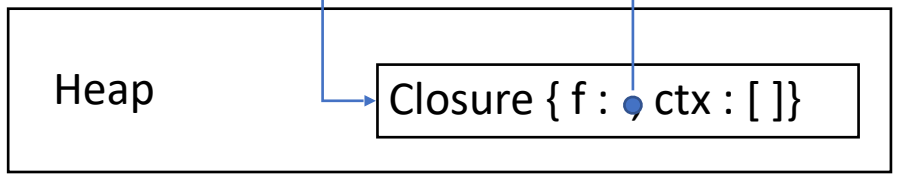
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



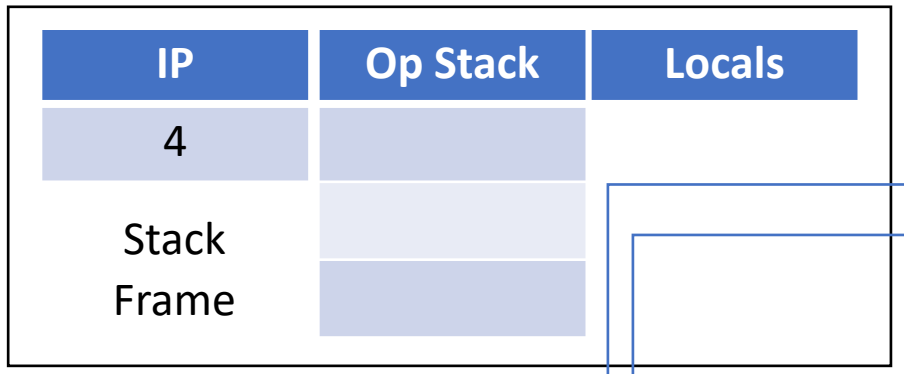

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure  0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```



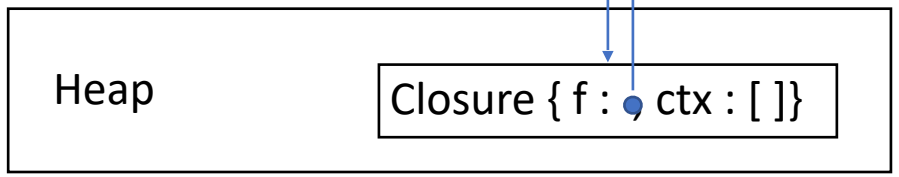
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



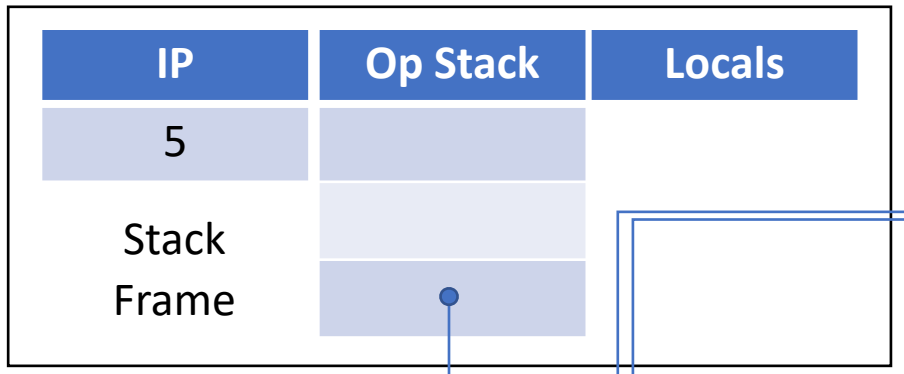

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure 0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```



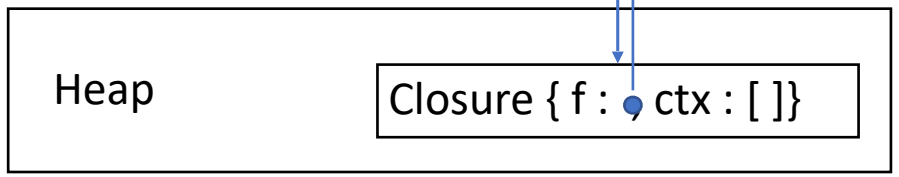
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



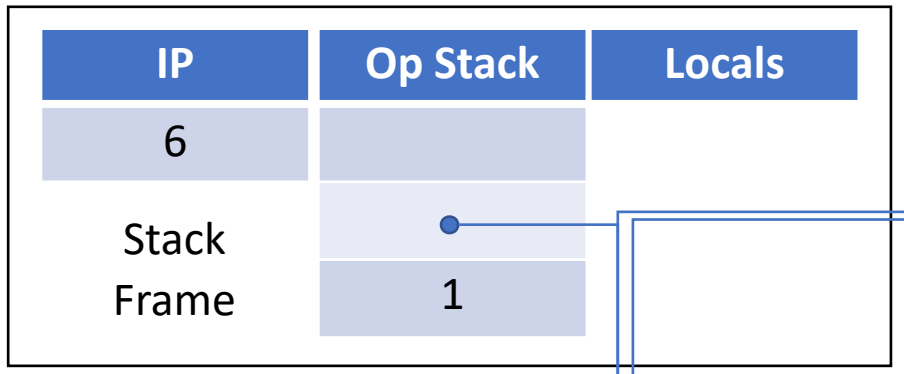

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure 0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```



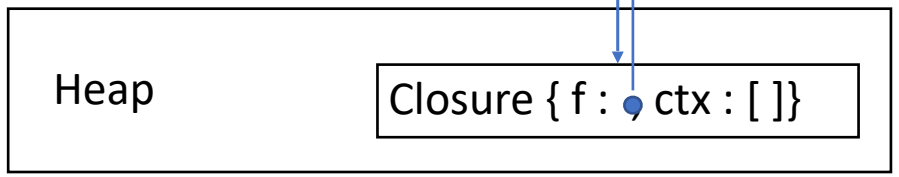
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



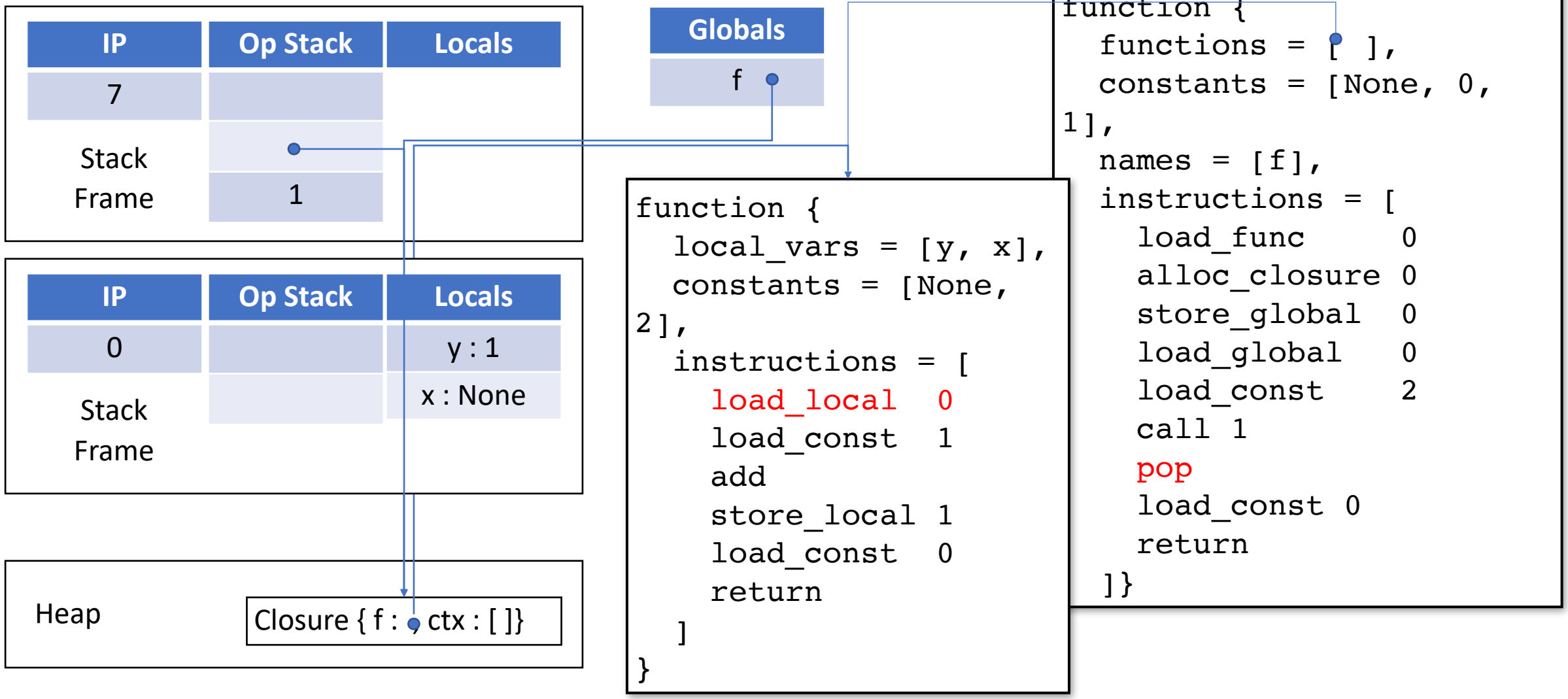

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure  0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```



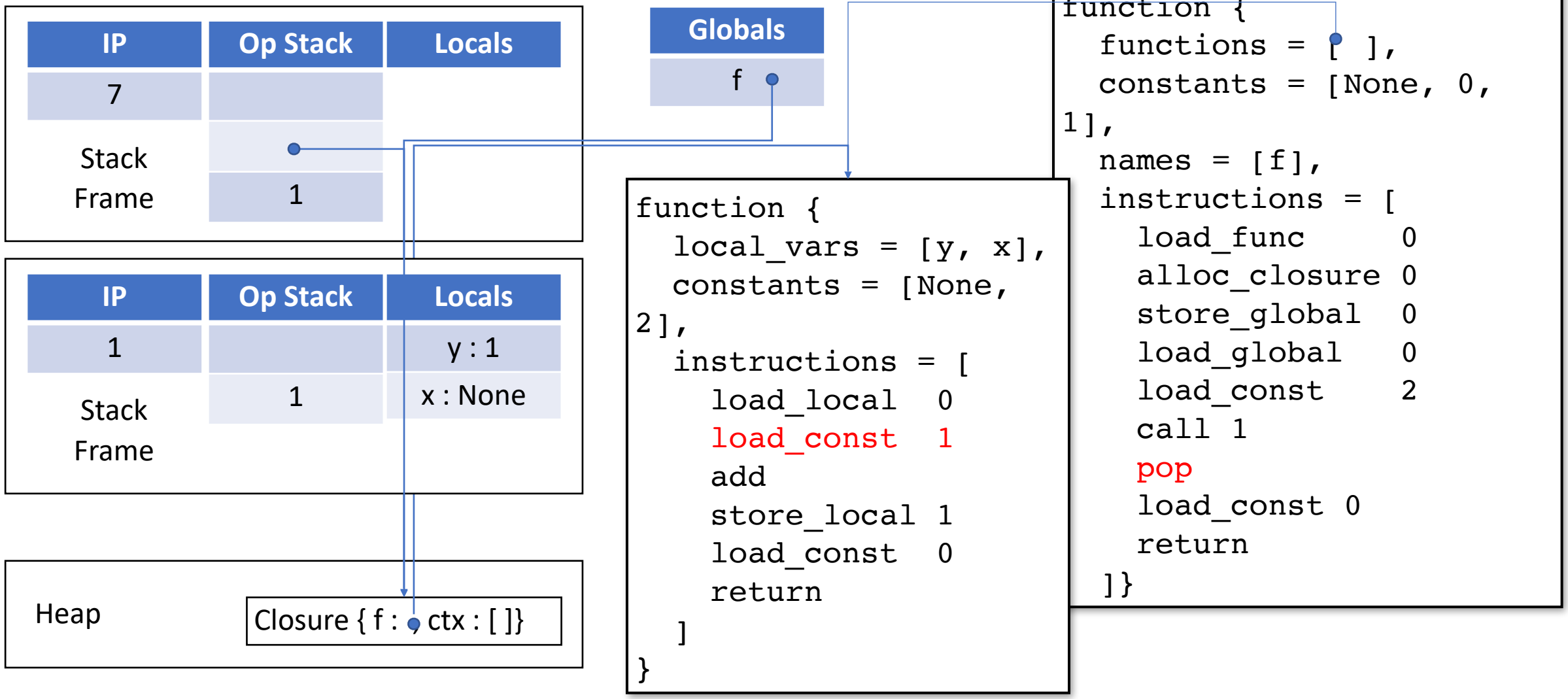
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



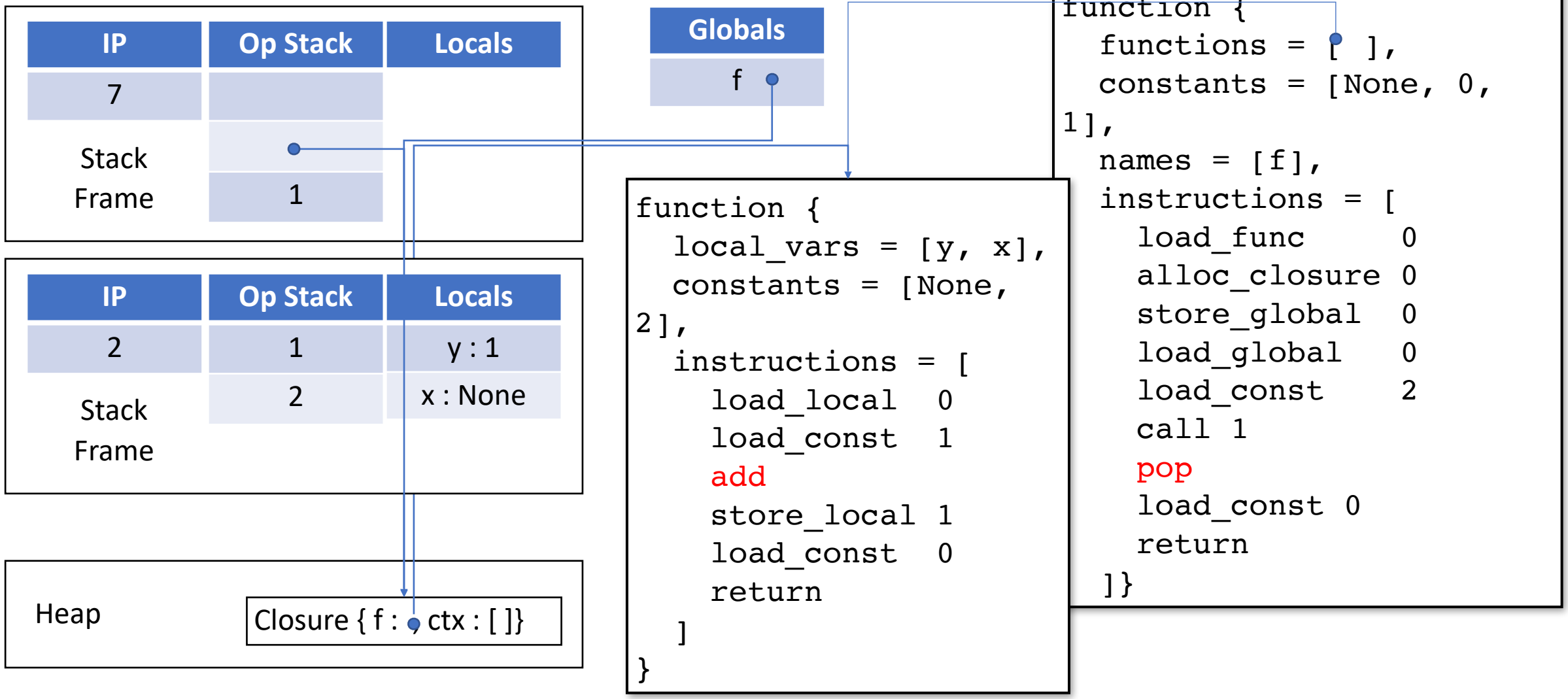
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



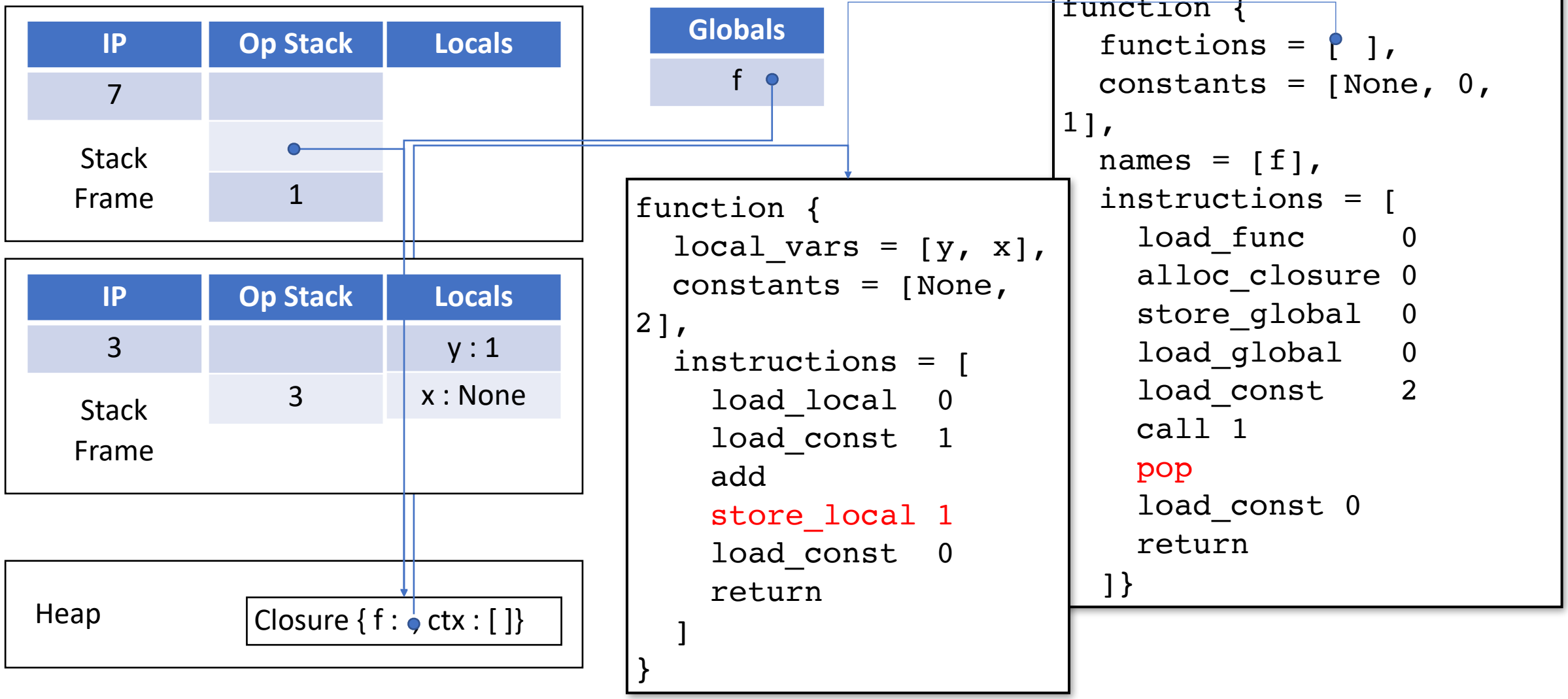
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



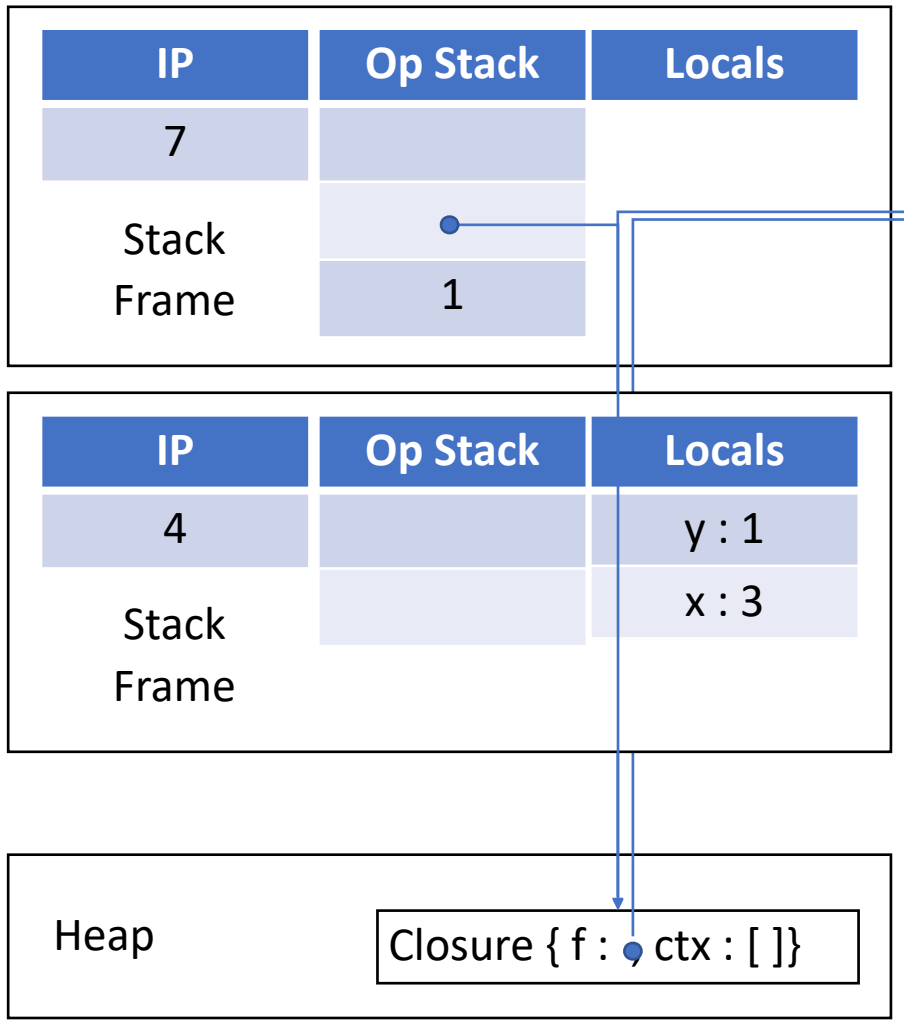
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```

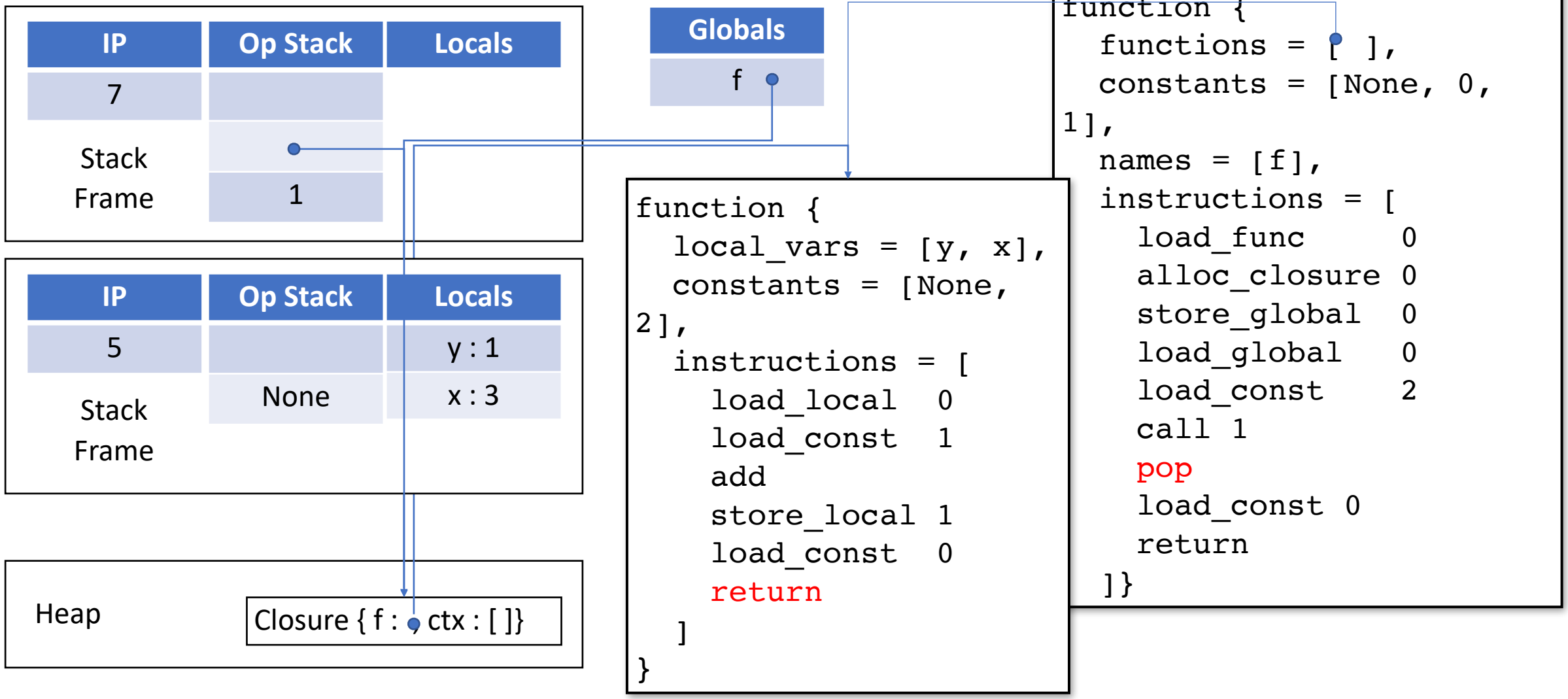



```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local   0
    load_const   1
    add
    store_local 1
    load_const   0
    return
  ]
}
```

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure 0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```

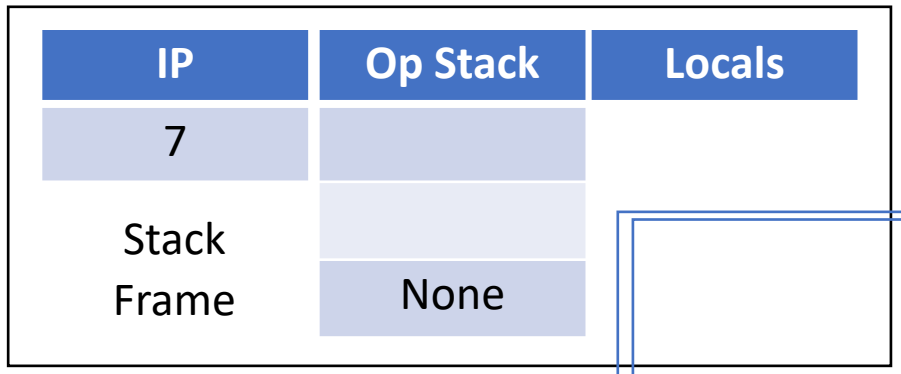
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



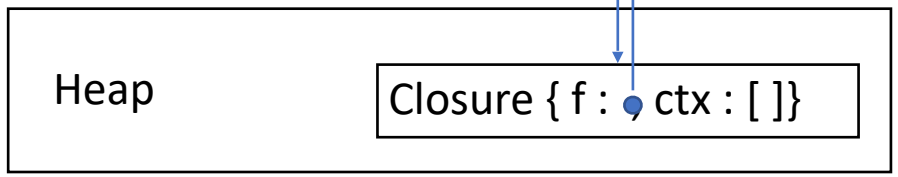
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



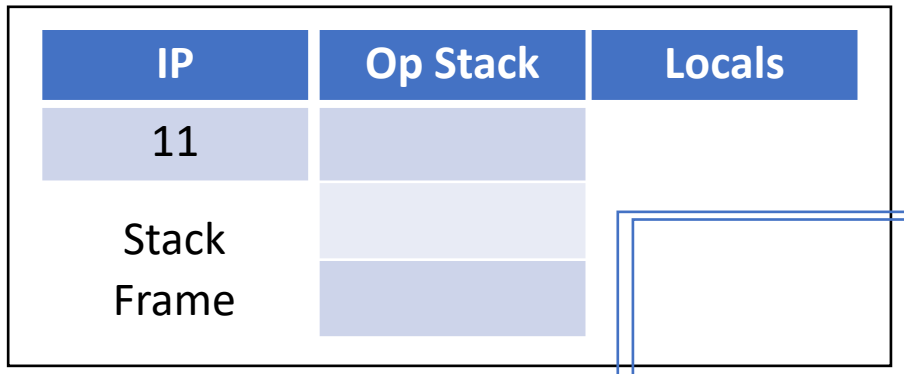

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure 0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```



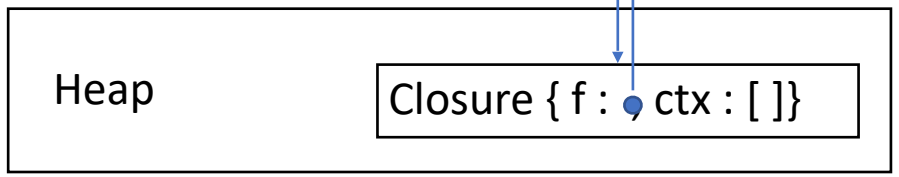
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



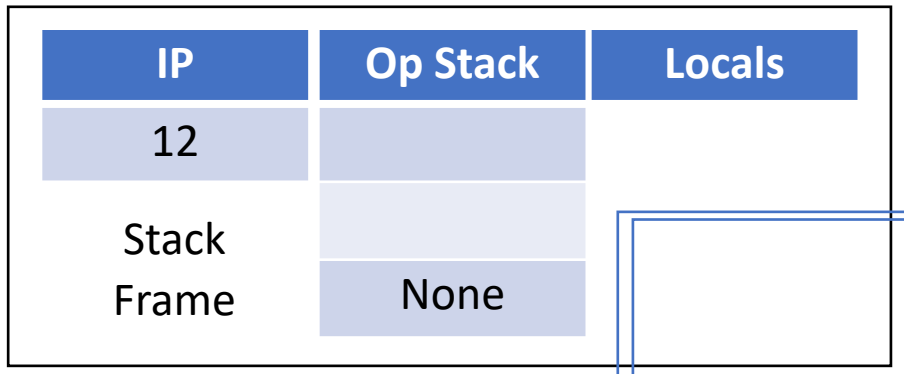

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure  0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```



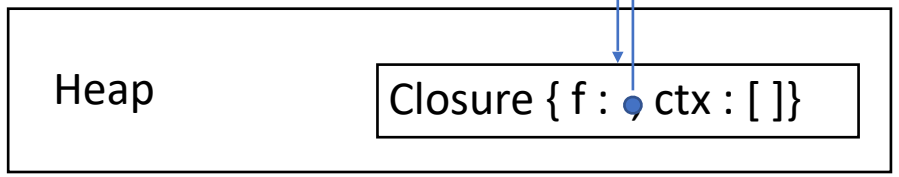
MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```



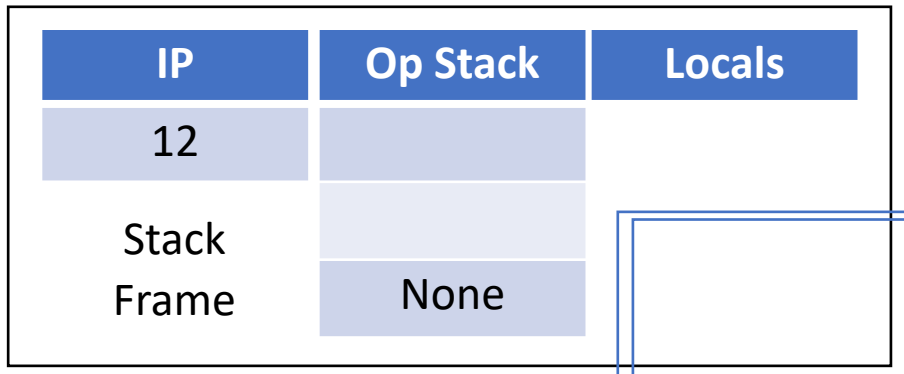

```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure  0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```



MITScript VM

```
f = fun(y) {
  x = y + 2;
};
f(1);
```




```
function {
  local_vars = [y, x],
  constants = [None,
2],
  instructions = [
    load_local  0
    load_const  1
    add
    store_local 1
    load_const  0
    return
  ]
}
```

```
function {
  functions = [ ],
  constants = [None, 0,
1],
  names = [f],
  instructions = [
    load_func      0
    alloc_closure  0
    store_global   0
    load_global    0
    load_const     2
    call 1
    pop
    load_const 0
    return
  ]}
```