How to make a simple virtual machine

Christian Stigen Larsen — Roxar Software Solutions
Stavanger Software Developers, Aug. 2015

```
while True:
    op = next_instruction()

    if op == foo:
        do_foo()
    elif op == bar:
        do_bar()
    elif
        …
```

```
while True:
    op = next_instruction()

    if op == foo:
        do_foo()
    elif op == bar:
        do_bar()
    elif
        …
```

```
while True:
    op = next_instruction()

    if op == foo:
        do_foo()
    elif op == bar:
        do_bar()
    elif
        …
```

```
while True:
    op = next_instruction()

    if op == foo:
        do_foo()
    elif op == bar:
        do_bar()
    elif
        …
```

```
while True:
    op = next_instruction()

    if op == foo:
        do_foo()
    elif op == bar:
        do_bar()
    elif
        …
```

```
while True:
    op = next_instruction()

    if op == foo:
        do_foo()
    elif op == bar:
        do_bar()
    elif
        …
```

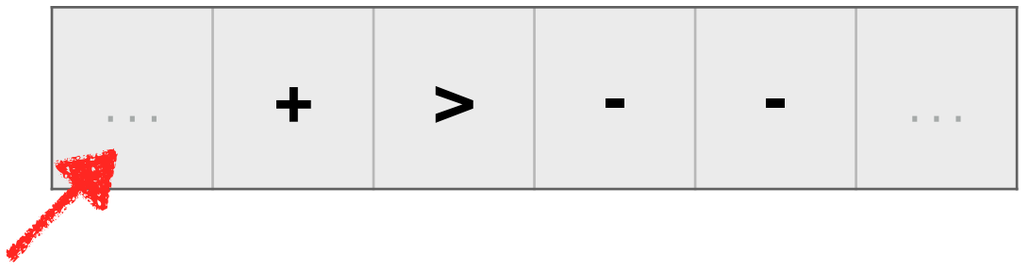
Brainf**k

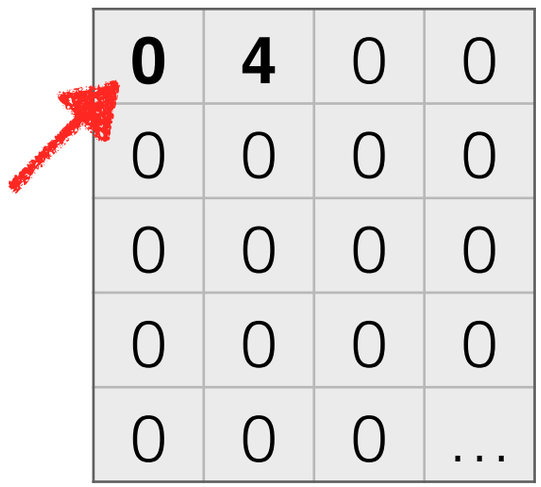
Instructions

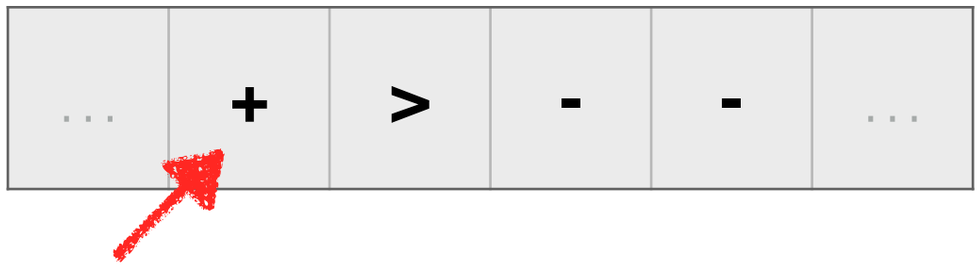
>	Move right
<	Move left
+	Increment
-	Decrement
.	Print
,	Read
[	Start loop
]	End loop

```
++++++++[>++++[>++>+++>+++>+<<<<-]
>+>+>->>+[<]<-]>>.>---.+++++++..++
+.>>.<-.<.++.------.--------.>>+.
>++.
```

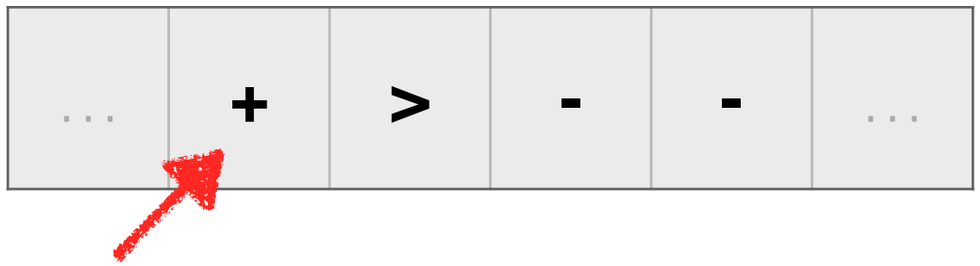
«Hello World!»


...
+
>
-
-
...

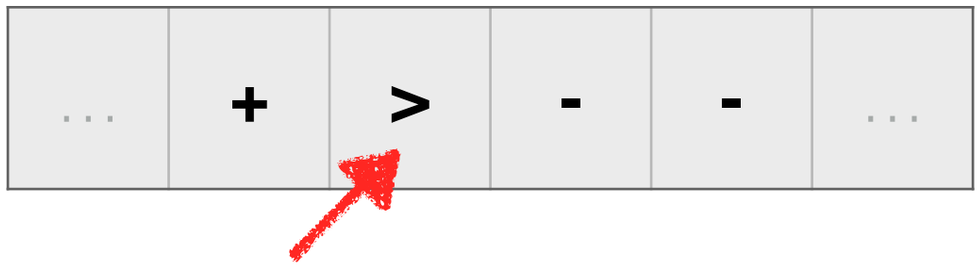

0 4 0 0
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 ...



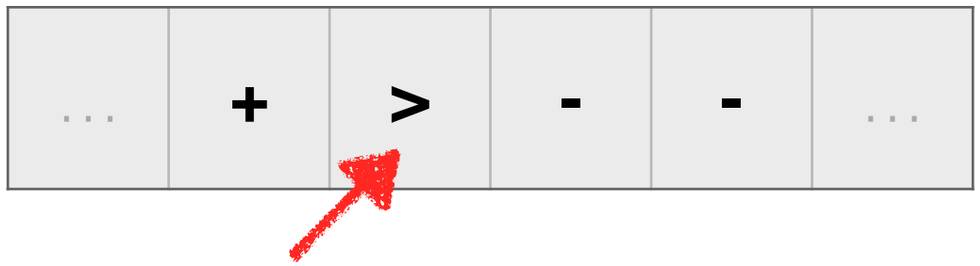
0	**4**	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	...



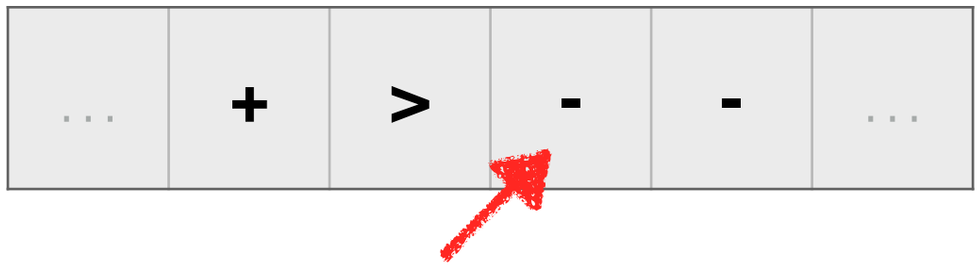
1	4	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	...



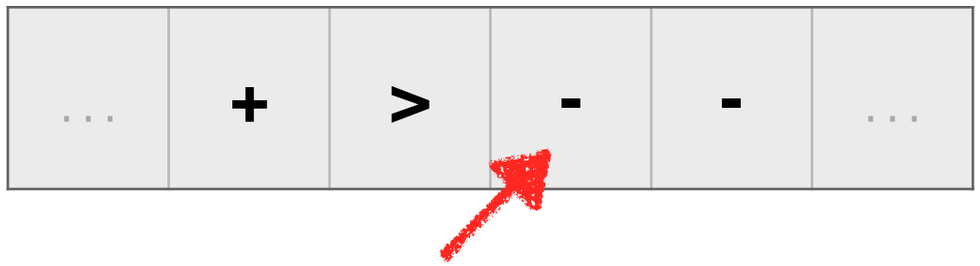
1	**4**	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	...



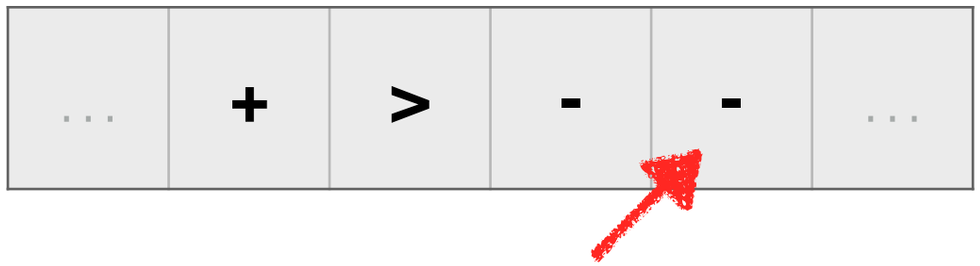
1	**4**	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	...



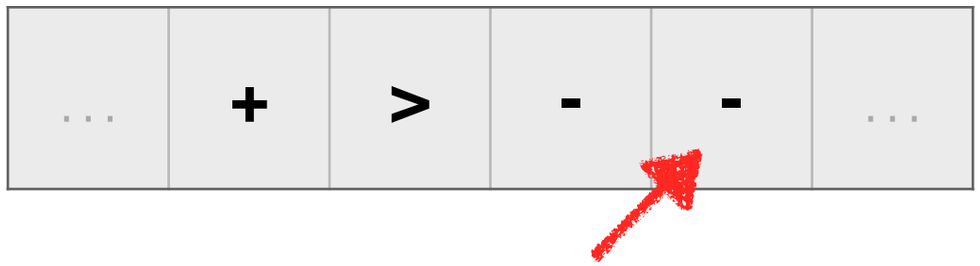
1	**4**	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	...



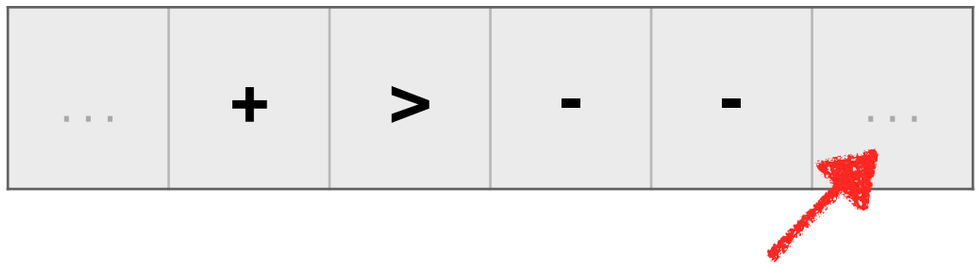
1	3	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	...



1	**3**	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	...



1	2	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	...



1	**2**	0	0
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	...

```
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++.>+
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++.>+++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++..>+
+++++++++.
```

```
++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++.>+
++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++
++++++++++++++++++++.>+++++
++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++
++++++++++..>+
++++++++++.
```

```
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++.>+
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++.>+++++
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++..>+
+++++++++.
```


HE

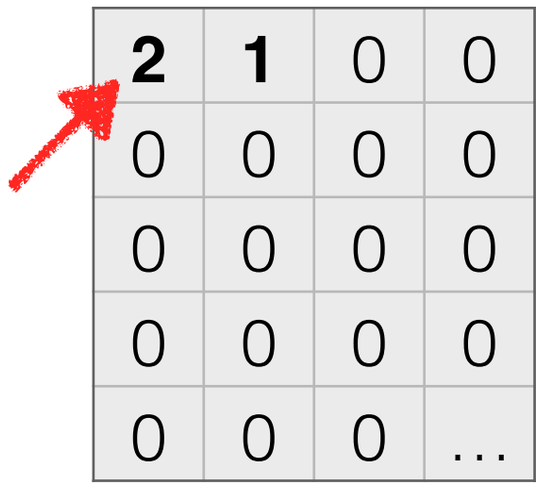
```
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++.>+
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++.>+++++
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++..>+
+++++++++.
```


HELL

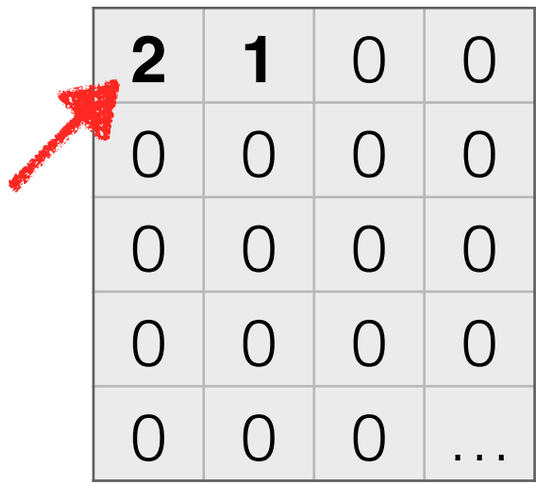
```
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++.>+
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++.>+++++
++++++++++++++++++++++++++++
++++++++++++++++++++++++++++
++++++++++++++++++++++..>+
+++++++++.
```

HELL

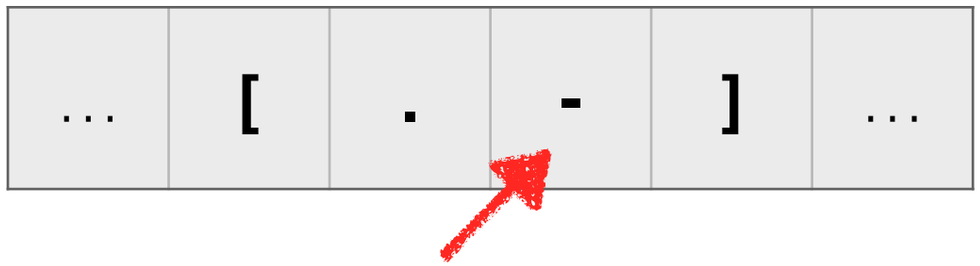
Loops

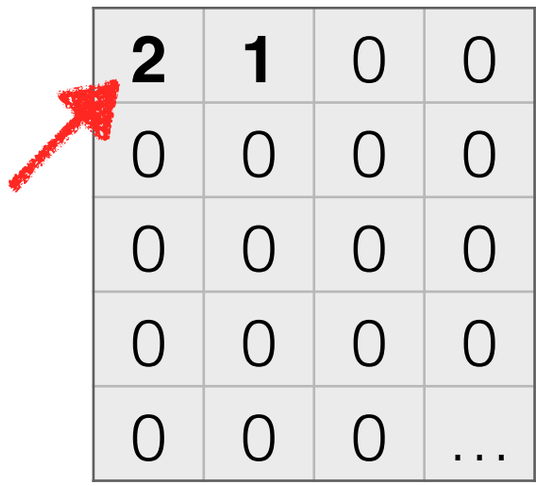


Loops

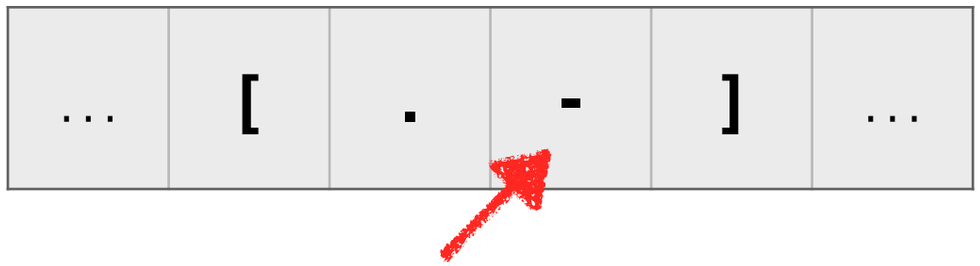


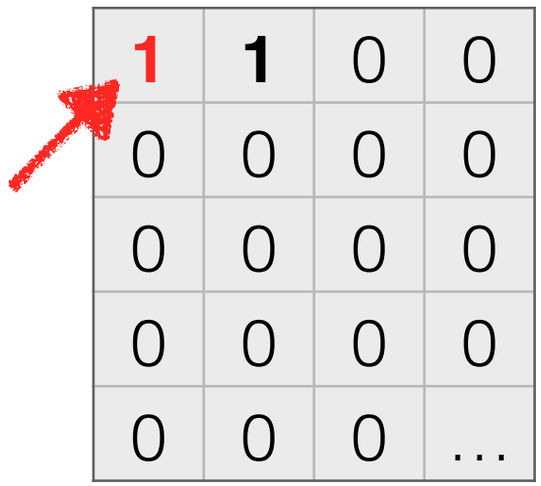
Loops



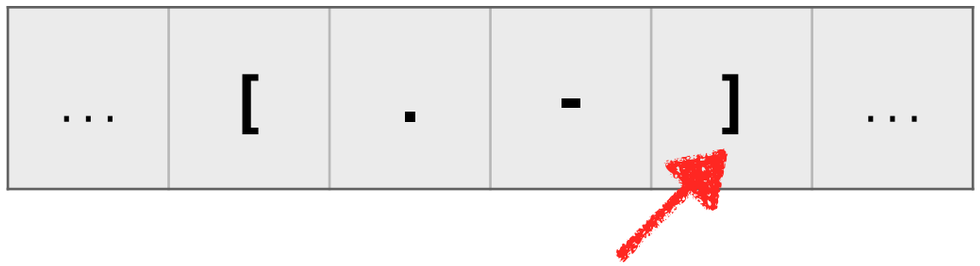


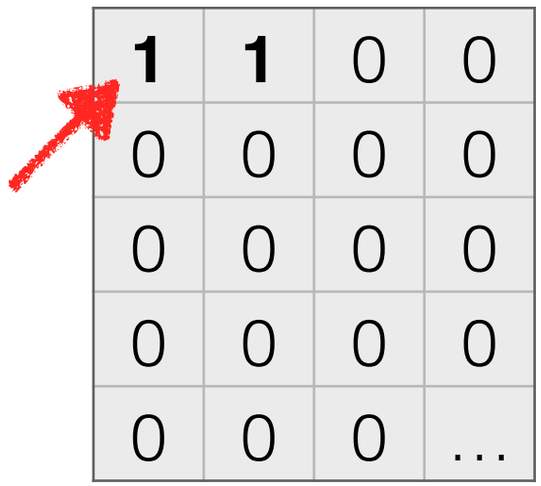
Loops



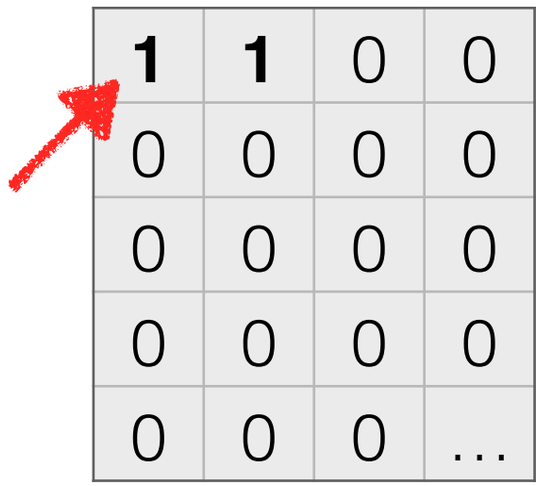


Loops

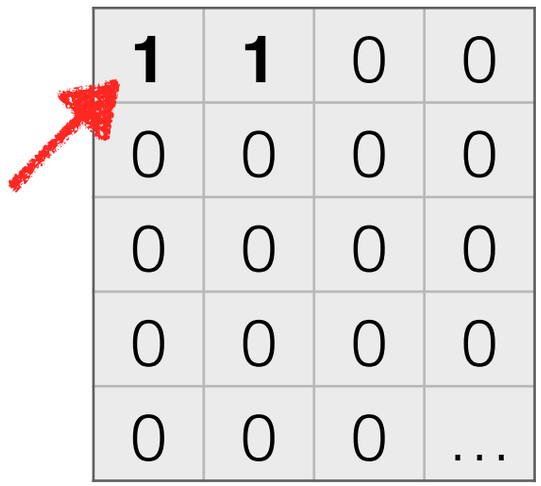




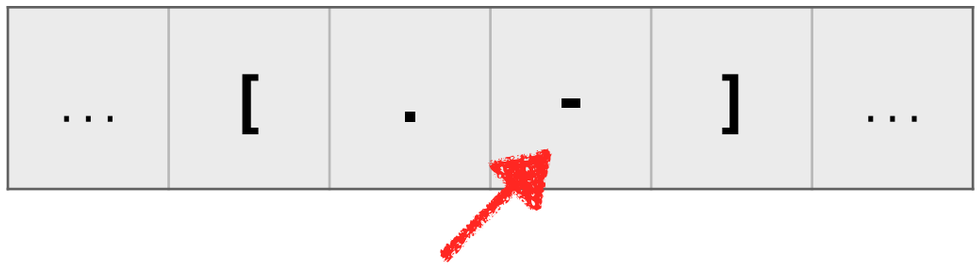
Loops

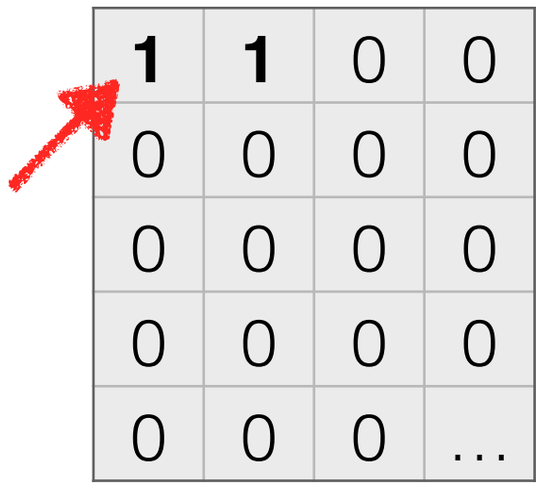


Loops

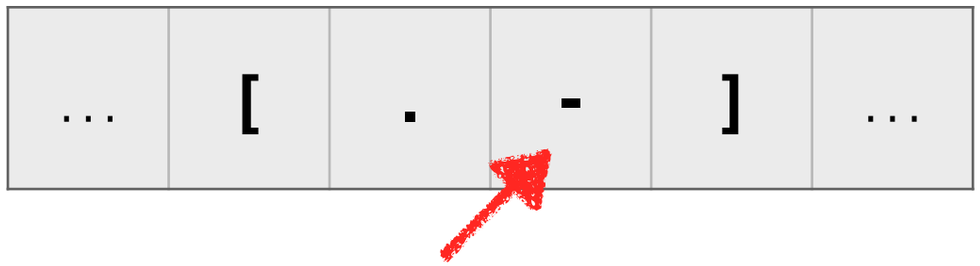


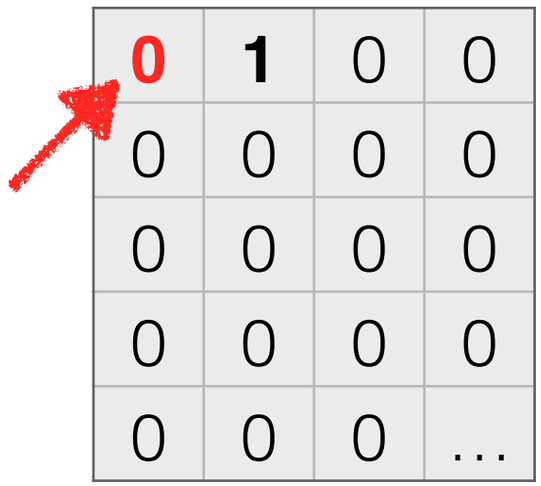
Loops



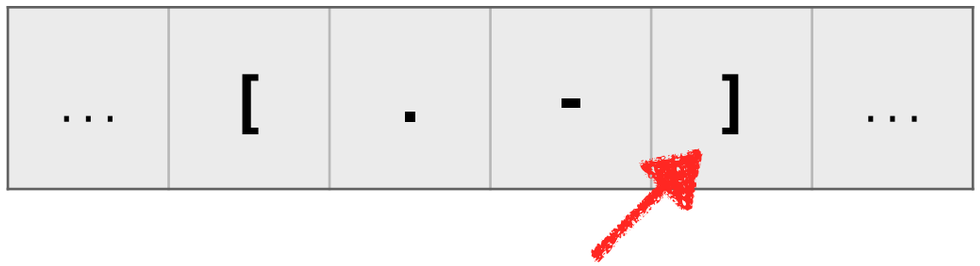


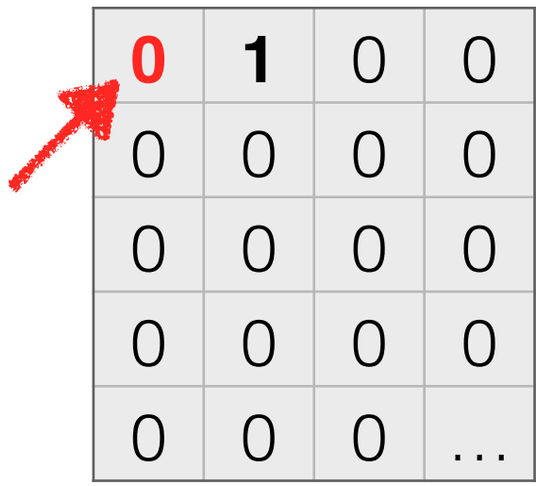
Loops



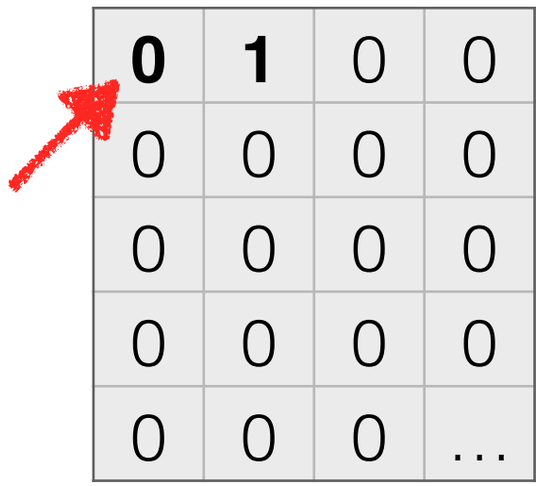


Loops





Loops



Interpreter

```
class Machine:
  def __init__(…):
    self.memory = [0]*memsize
    self.mptr = 0
    self.code = …
    self.cptr = 0
    self.stack = deque()
```

```
class Machine:
  def __init__(…):
    self.memory = [0]*memsize
    self.mptr = 0
    self.code = …
    self.cptr = 0
    self.stack = deque()
```

```
class Machine:
  def __init__(…):
    self.memory = [0]*memsize
    self.mptr = 0
    self.code = …
    self.cptr = 0
    self.stack = deque()
```

```
class Machine:
  def __init__(…):
    self.memory = [0]*memsize
    self.mptr = 0
    self.code = …
    self.cptr = 0
    self.stack = deque()
```

```
def run(self):
  while True:
    instruction = self.next()
    self.dispatch(instruction)
```

```
def run(self):
    while True:
        instruction = self.next()
        self.dispatch(instruction)
```

```
def run(self):
  while True:
    instruction = self.next()
    self.dispatch(instruction)
```

```
def run(self):
  while True:
    instruction = self.next()
    self.dispatch(instruction)
```

```
def next(self):
    instruction = self.code[self.cptr]
    self.cptr += 1
    return instruction
```

```
def next(self):
    instruction = self.code[self.cptr]
    self.cptr += 1
    return instruction
```

```
def next(self):
    instruction = self.code[self.cptr]
    self.cptr += 1
    return instruction
```

```
def dispatch(self, i):
  if i == «>»:
    self.mptr += 1
  elif i == «<»:
    self.mptr -= 1
```

```
def dispatch(self, i):
  if i == «>»:
    self.mptr += 1
  elif i == «<»:
    self.mptr -= 1
```

```
def dispatch(self, i):
  if i == «>»:
    self.mptr += 1
  elif i == «<»:
    self.mptr -= 1
```

```
def dispatch(self, i):
    if i == «>»:
        self.mptr += 1
    elif i == «<»:
        self.mptr -= 1
    elif i == «+»:
        self.memory[self.mptr] += 1
    elif i == «-»:
        self.memory[self.mptr] -= 1
```

```
def dispatch(self, i):
    if i == «>»:
        self.mptr += 1
    elif i == «<»:
        self.mptr -= 1
    elif i == «+»:
        self.memory[self.mptr] += 1
    elif i == «-»:
        self.memory[self.mptr] -= 1
```

```
def dispatch(self, i):
    if i == «>»:
        self.mptr += 1
    elif i == «<»:
        self.mptr -= 1
    elif i == «+»:
        self.memory[self.mptr] += 1
    elif i == «-»:
        self.memory[self.mptr] -= 1
    elif i == «.»:
        value = self.memory[self.mptr]
        sys.stdout.write(value)
    elif i == «,»:
        value = sys.stdint.read(1)
        self.memory[self.mptr] = value
```

```
def dispatch(self, i):
    if i == «>»:
        self.mptr += 1
    elif i == «<»:
        self.mptr -= 1
    elif i == «+»:
        self.memory[self.mptr] += 1
    elif i == «-»:
        self.memory[self.mptr] -= 1
    elif i == «.»:
        value = self.memory[self.mptr]
        sys.stdout.write(value)
    elif i == «,»:
        value = sys.stdint.read(1)
        self.memory[self.mptr] = value
```

```
def dispatch(self, i):
  if i == «>»:
    self.mptr += 1
  elif i == «<»:
    self.mptr -= 1
  elif i == «+»:
    self.memory[self.mptr] += 1
  elif i == «-»:
    self.memory[self.mptr] -= 1
elif i == «.»:
  value = self.memory[self.mptr]
  sys.stdout.write(value)
elif i == «,»:
  value = sys.stdint.read(1)
  self.memory[self.mptr] = value
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
```

[… […] …]

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    count = 1
    while count > 0:
      i = self.next()
      if i == «[»:
        count += 1
      elif i == «]»:
        count -= 1
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
elif i == «]»:
  return_addr = self.stack.pop()
  if self.memory[self.mptr] != 0:
    self.cptr = return_addr
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
elif i == «]»:
  return_addr = self.stack.pop()
  if self.memory[self.mptr] != 0:
    self.cptr = return_addr
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
elif i == «]»:
  return_addr = self.stack.pop()
  if self.memory[self.mptr] != 0:
    self.cptr = return_addr
```

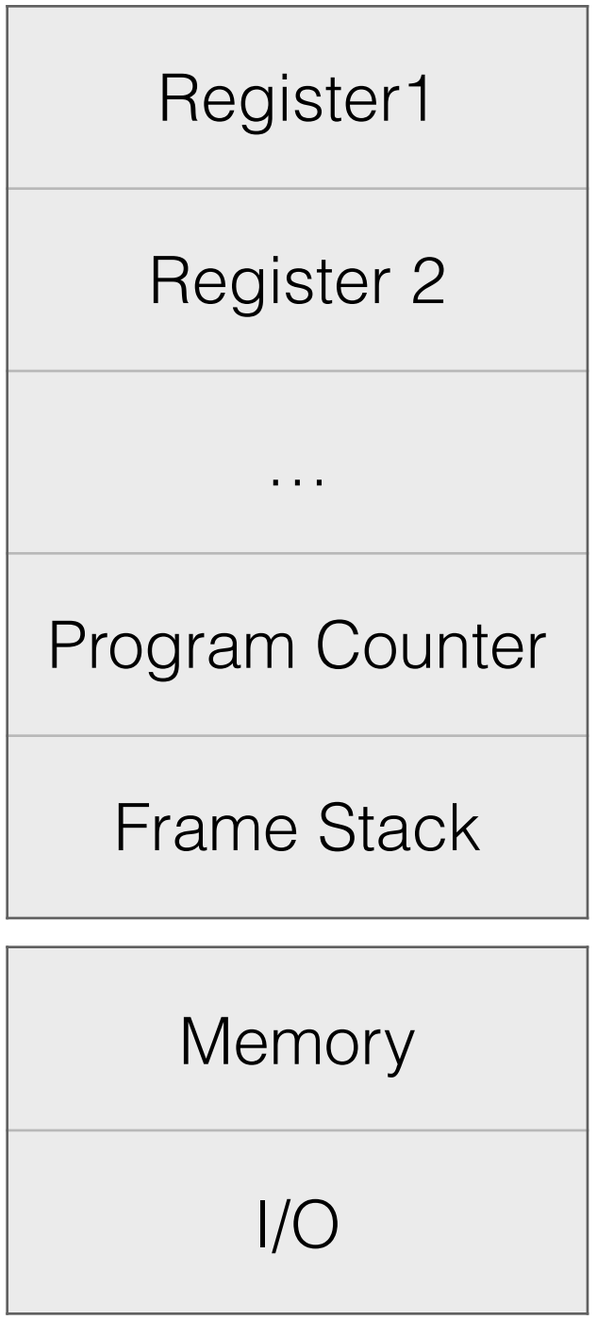
```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
elif i == «]»:
  return_addr = self.stack.pop()
  if self.memory[self.mptr] != 0:
    self.cptr = return_addr
```

```
elif i == «[»:
  if self.memory[self.mptr] != 0:
    self.stack.append(self.cptr-1)
  elif:
    self.skip_block()
elif i == «]»:
  return_addr = self.stack.pop()
  if self.memory[self.mptr] != 0:
    self.cptr = return_addr
```

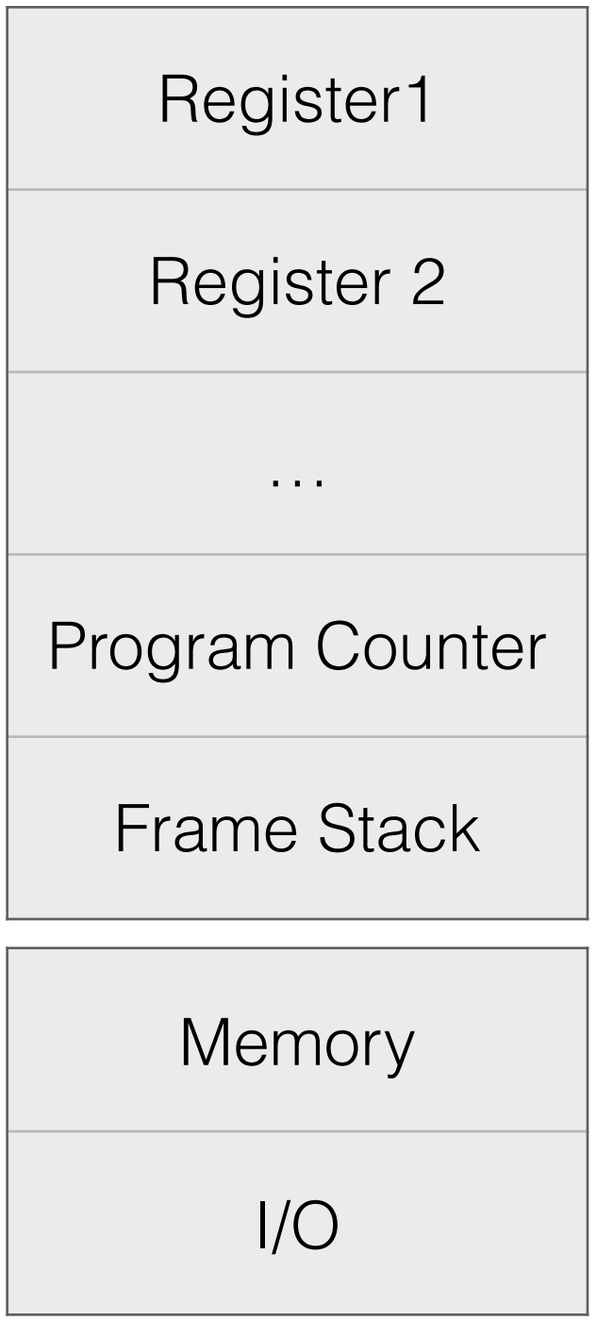
Demo

Stack vs Register Machines

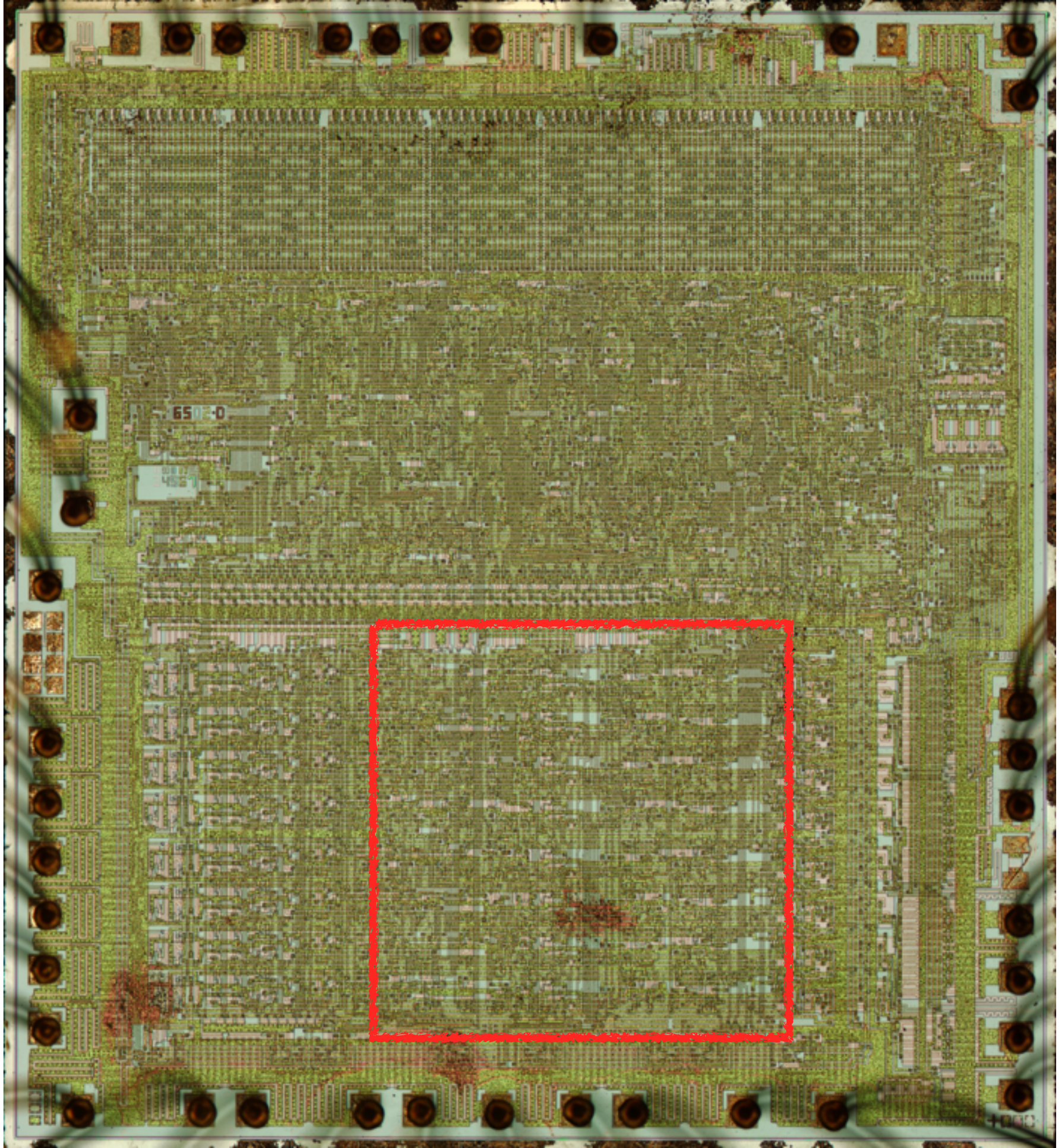
Register Machine

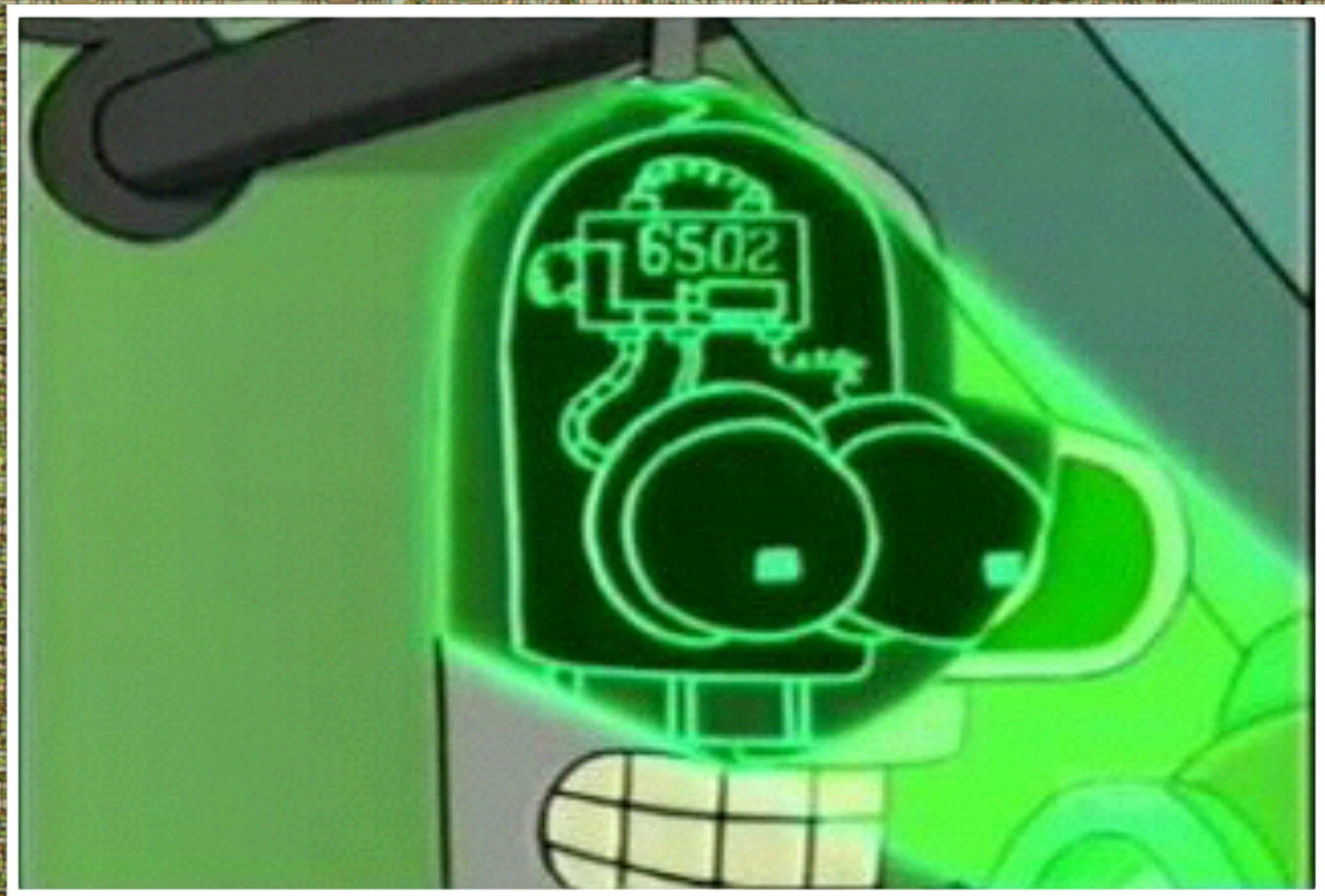


Register Machine

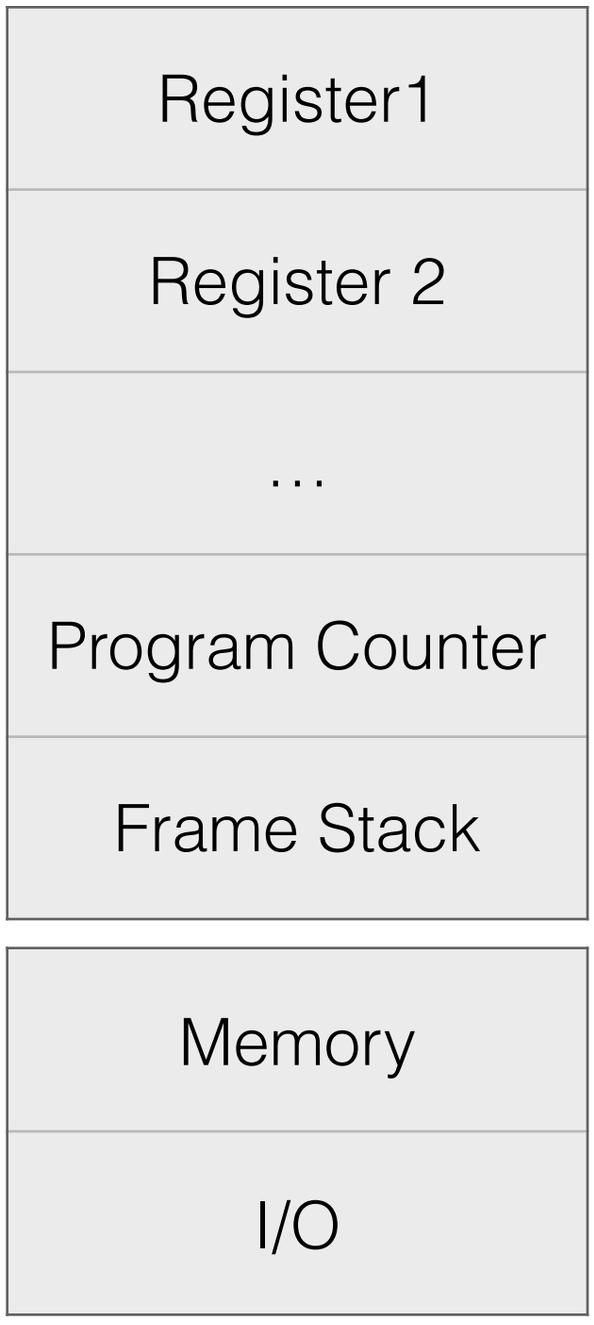



```
self.memory[0] (kinda)
self.memory[1] (kinda)
self.cptr
self.stack
self.memory
sys.stdin, sys.stdout
```



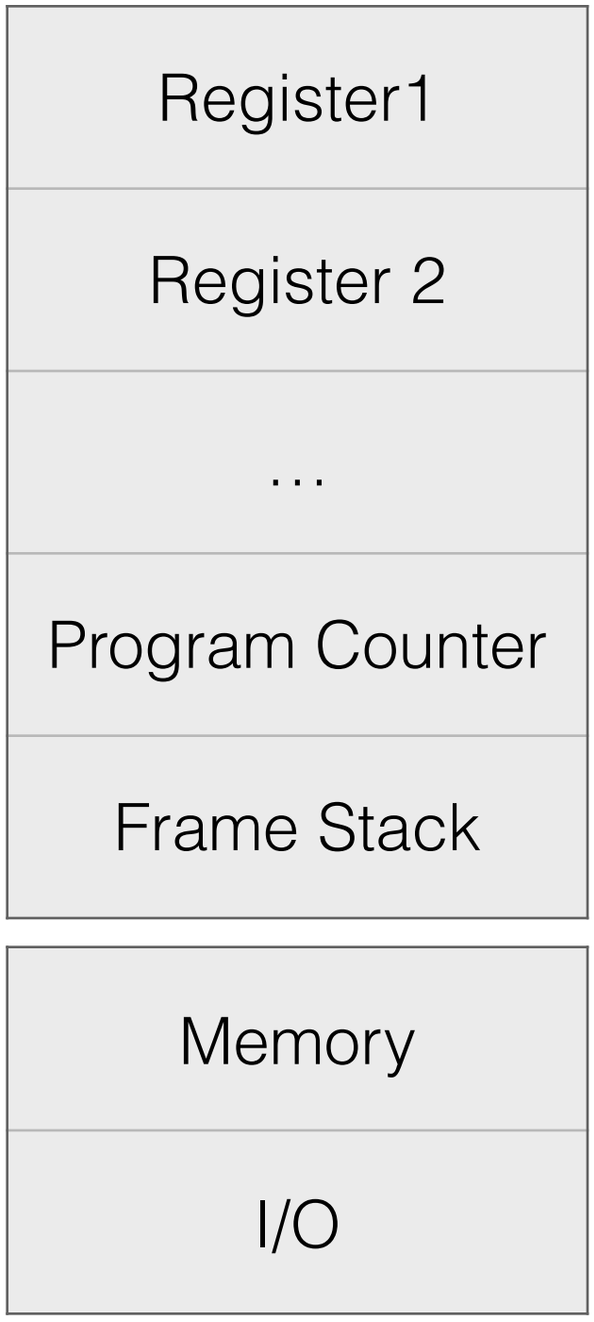

6502

Register Machine

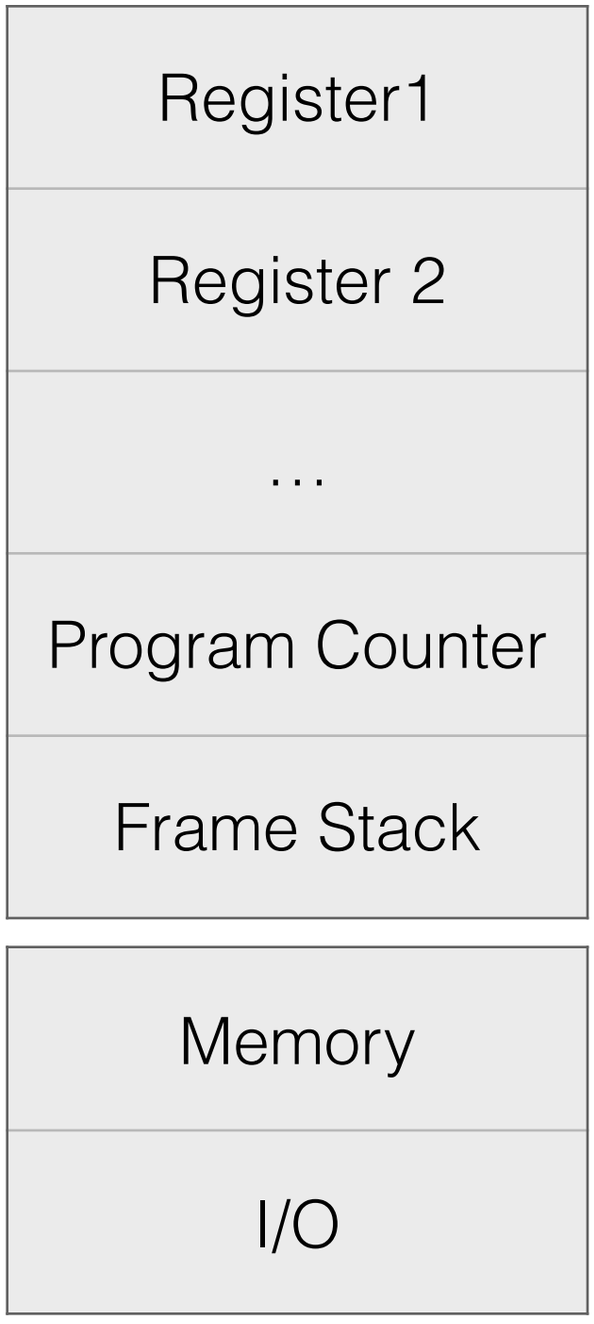



```
self.memory[0] (kinda)
self.memory[1] (kinda)
self.cptr
self.stack
self.memory
sys.stdin, sys.stdout
```

Register Machine



Register Machine



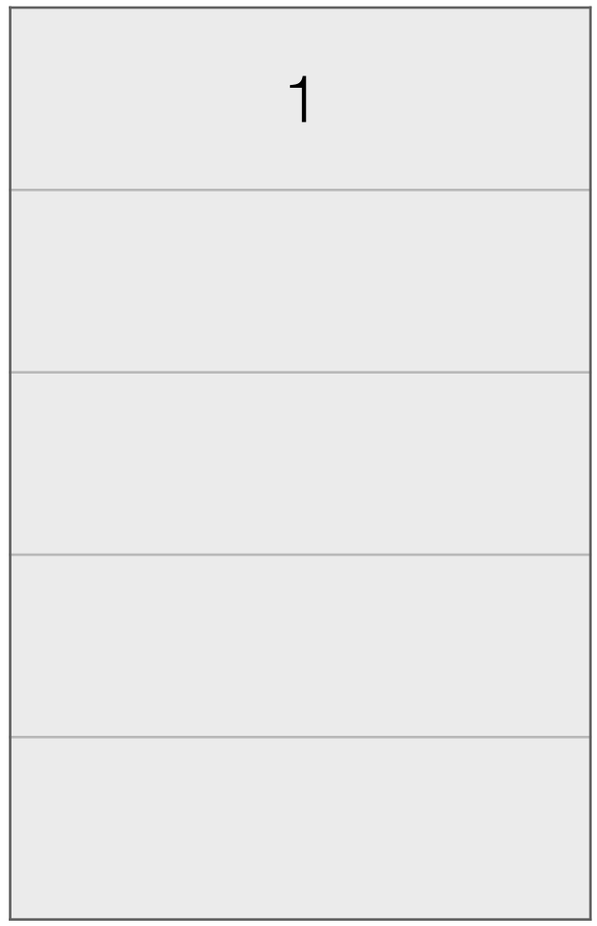
Stack Machine

Some Stack Machines

- JVM
- CLR
- Forth
- CPython
- Lua
- ...

1 2 + 3 *

Data Stack



1 2 + 3 *

Data Stack

2
1

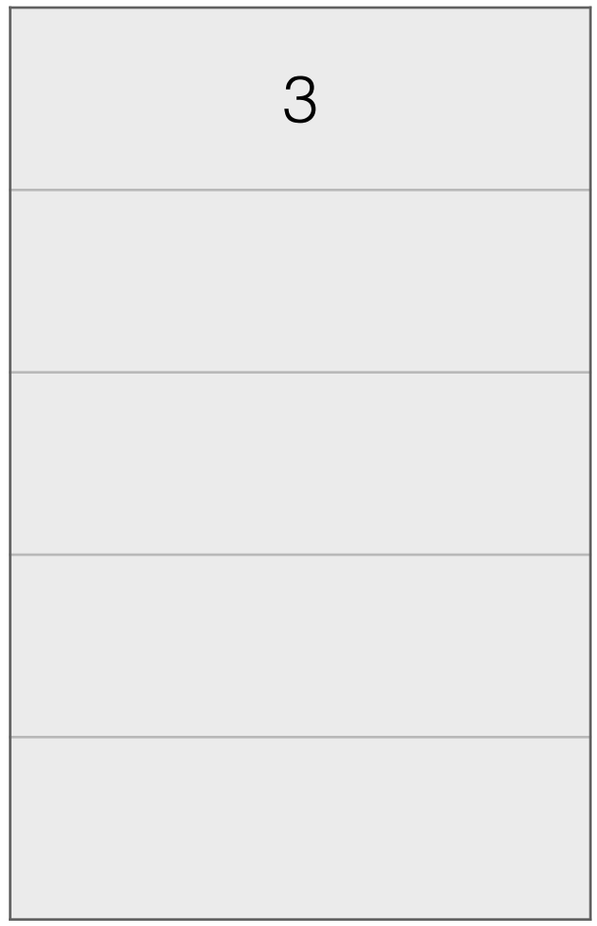
1 2 + 3 *

Data Stack

2
1

1 2 + 3 *

Data Stack



1 2 + 3 *

Data Stack

3
3

1 2 + 3 *

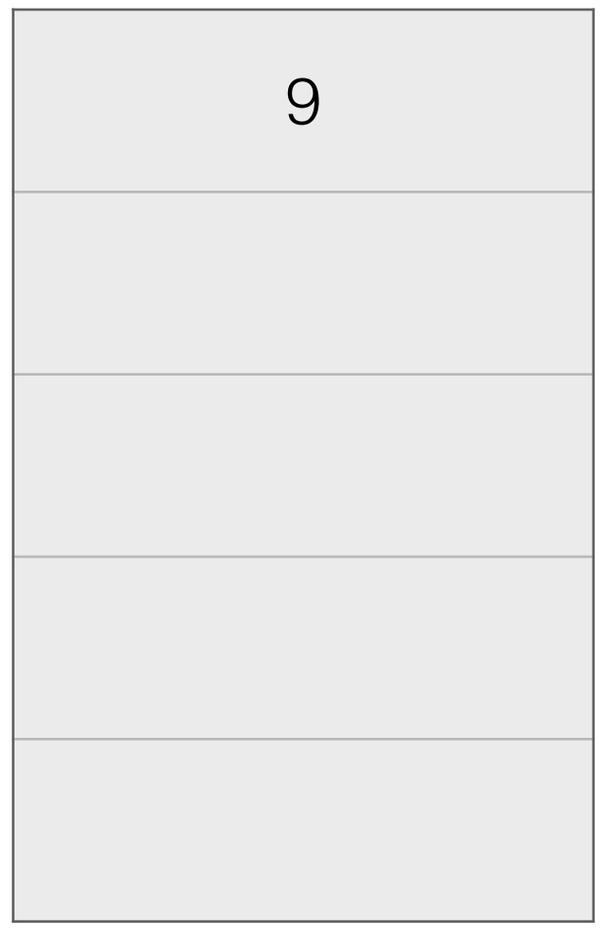
Data Stack

9

1 2 + 3 *

(1+2)*3

Data Stack



Can we make the interpreter faster?

JIT to Python Bytecode

Python Bytecode

```
LOAD_CONST «Hello there»

PRINT_ITEM
PRINT_NEWLINE

LOAD_CONST None
RETURN_VALUE
```

Python Bytecode

```
LOAD_CONST «Hello there»

PRINT_ITEM
PRINT_NEWLINE

LOAD_CONST None
RETURN_VALUE
```

Python Bytecode

```
LOAD_CONST «Hello there»

PRINT_ITEM
PRINT_NEWLINE

LOAD_CONST None
RETURN_VALUE
```

Python Bytecode

```
LOAD_CONST «Hello there»

PRINT_ITEM
PRINT_NEWLINE

LOAD_CONST None
RETURN_VALUE
```

Brainf**k to bytecode

```
def plus():
    LOAD_FAST «ptr»
    LOAD_CONST 1
    INPLACE_ADD
    STORE_FAST «ptr»
```

Loops

```
label «start-of-loop»

LOAD_FAST «memory»
LOAD_FAST «ptr»
BINARY_SUBSCR

LOAD_CONST 0
COMPARE_OP «==»

POP_JUMP_IF_TRUE «end-of-loop»
```

Loops

```
label «start-of-loop»

LOAD_FAST «memory»      memory[ptr]
LOAD_FAST «ptr»
BINARY_SUBSCR

LOAD_CONST 0                  memory[ptr] == 0
COMPARE_OP «==»

POP_JUMP_IF_TRUE «end-of-loop»
if memory[ptr] == 0: goto «end-of-loop»
```

Optimizations

`>>>>`	`ptr += 4`
`<<<<`	`ptr -= 4`
`++++`	`memory[ptr] += 4`
`- - - -`	`memory[ptr] -= 4`

Demo

Can we make it faster?

JIT to Machine Code

GNU Lightning

Register V0	Points to memory cell
Register V1	For arithmetic

```
movi V0, <address of first cell>
```

<	subi V0, 1
>	addi V0, 1

+	ldr V1, V0
	addi V1, 1
	str V0, V1

-	ldr V1, V0
	subi V1, 1
	str V0, V1

<	subi V0, 1
>	addi V0, 1

+	ldr V1, V0
	addi V1, 1
	str V0, V1

-	ldr V1, V0
	subi V1, 1
	str V0, V1

<	subi V0, 1
>	addi V0, 1

+	ldr V1, V0
	addi V1, 1
	str V0, V1

-	ldr V1, V0
	subi V1, 1
	str V0, V1

<	subi V0, 1
>	addi V0, 1

+	ldr V1, V0
	addi V1, 1
	str V0, V1

-	ldr V1, V0
	subi V1, 1
	str V0, V1

[	loop_start:
	  ldr V1, V0
	  beqi V1, 0, loop_end

]	loop_start:
	  ldr V1, V0
	  bnei V1, 0, loop_start

[	loop_start:
	  ldr V1, V0
	  beqi V1, 0, loop_end

]	jmp loop_start

```
++++++++++
>+++[<.>-]
```

```
++++++++++
>+++[<.>-]
```

```
mov     (%rbx),%r13
add     $0xa,%r13
mov     %r13,(%rbx)

add     $0x8,%rbx

mov     (%rbx),%r13
add     $0x3,%r13
mov     %r13,(%rbx)
```

```
++++++++++
>+++[<.>-]
```

```
mov     (%rbx),%r13
add     $0xa,%r13
mov     %r13,(%rbx)

add     $0x8,%rbx

mov     (%rbx),%r13
add     $0x3,%r13
mov     %r13,(%rbx)
```

```
++++++++++
>+++[<.>-]
```

```
mov     (%rbx),%r13
add     $0xa,%r13
mov     %r13,(%rbx)

add     $0x8,%rbx

mov     (%rbx),%r13
add     $0x3,%r13
mov     %r13,(%rbx)
```

```
++++++++++
>+++[<.>-]
```

```
mov     (%rbx),%r13
add     $0xa,%r13
mov     %r13,(%rbx)

add     $0x8,%rbx

mov     (%rbx),%r13
add     $0x3,%r13
mov     %r13,(%rbx)
```

```
++++++++++
>+++[<.>-]
```

```
                mov    (%rbx),%r13
               test    %r13,%r13
               je      0x10006f090
               …
0x10006f048:   sub     $0x8,%rbx
               …
                       mov    (%rbx),%r13
                       test   %r13,%r13
                       jne    0x10006f048
0x10006f090:   …
```

```
++++++++++
>+++[<.>-]
```

```
                    mov     (%rbx),%r13
                   test    %r13,%r13
                   je      0x10006f090
                   …
0x10006f048:  sub     $0x8,%rbx
                   …
                   mov     (%rbx),%r13
                   test    %r13,%r13
                   jne     0x10006f048
0x10006f090:  …
```

```
++++++++++
>+++[<.>-]
```

```
 mov     (%rbx),%r13
test    %r13,%r13
je      0x10006f090
…
0x10006f048:  sub     $0x8,%rbx
…
              mov     (%rbx),%r13
              test    %r13,%r13
              jne     0x10006f048
0x10006f090:  …
```

```
++++++++++
>+++[<.>-]
```

```
 mov     (%rbx),%r13
test    %r13,%r13
je      0x10006f090
```

```
…
0x10006f048:  sub     $0x8,%rbx
…
              mov     (%rbx),%r13
              test    %r13,%r13
              jne     0x10006f048
0x10006f090:  …
```

Demo

Where do we go from here?

Applications

Applications

Berkeley Packet Filter

Applications

ScummVM

Applications

Console Emulators

Applications

Presentation Apps

Applications

Embedded Programming

Applications

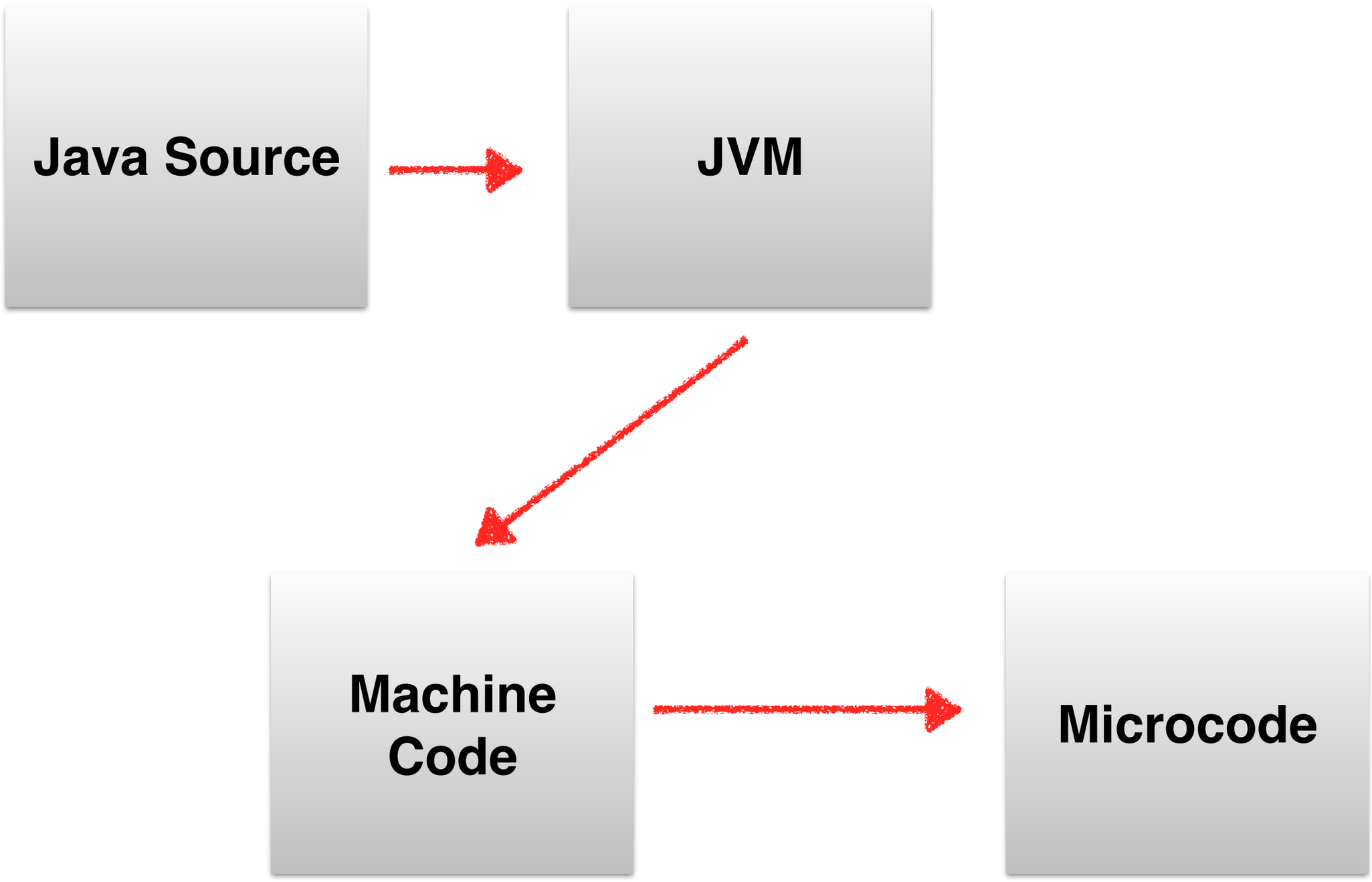
Genetic Programming

Applications

Fast Networking

Turtles all the way down

Turtles all the way down



The end,
at last!