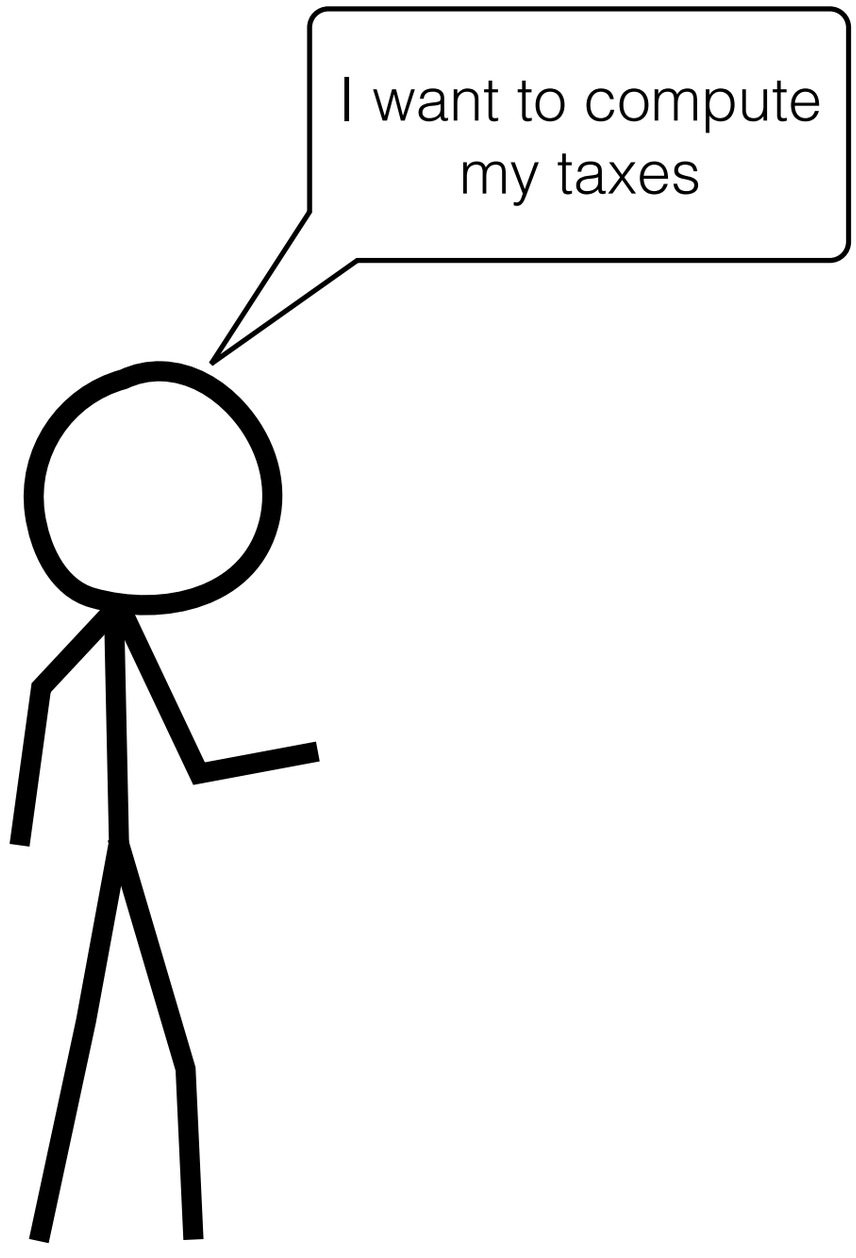
From trash to treasure:

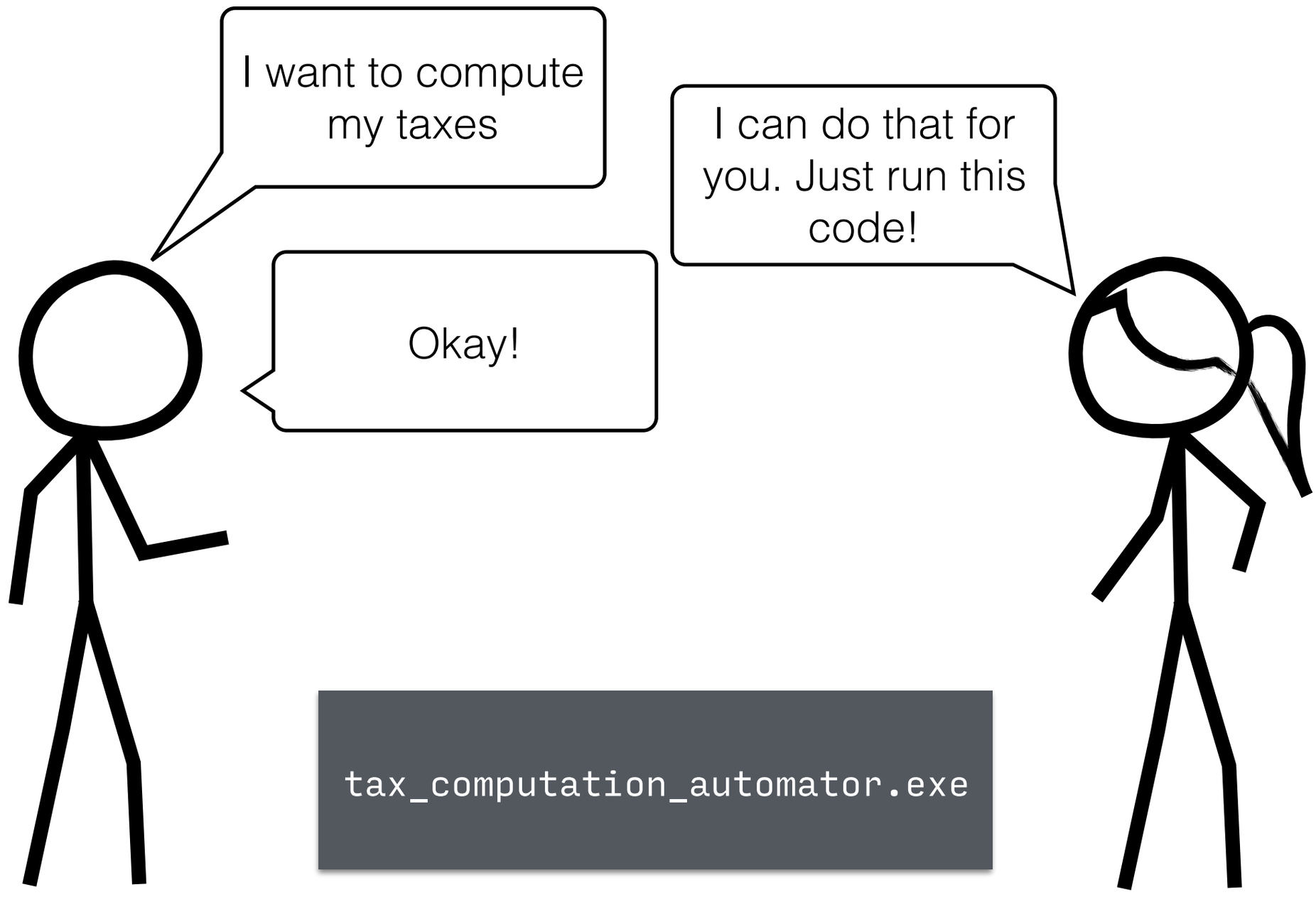
Timing-sensitive garbage collection

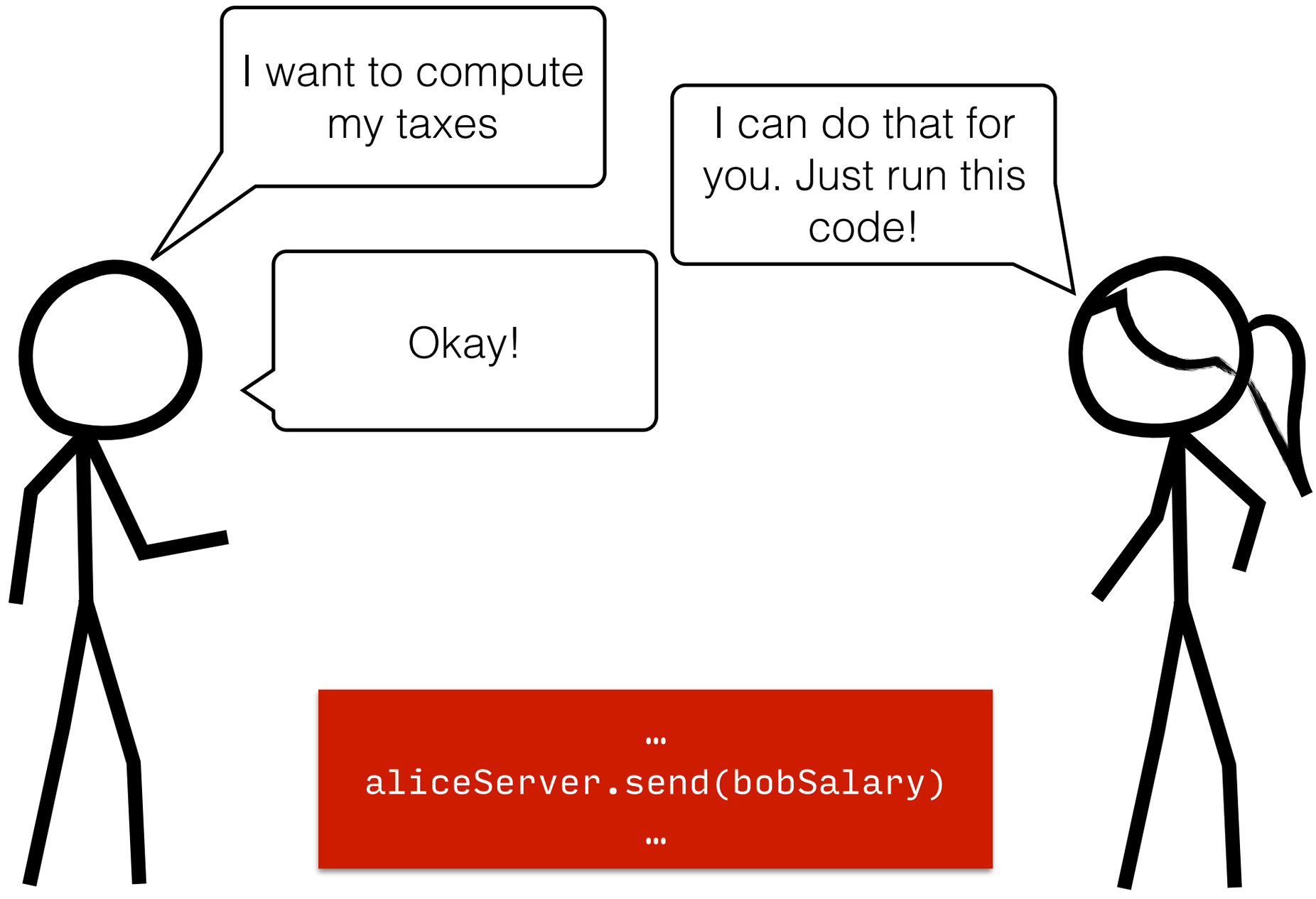
Mathias V. Pedersen & Aslan Askarov

AARHUS UNIVERSITY

S&P 2017


I want to compute
my taxes

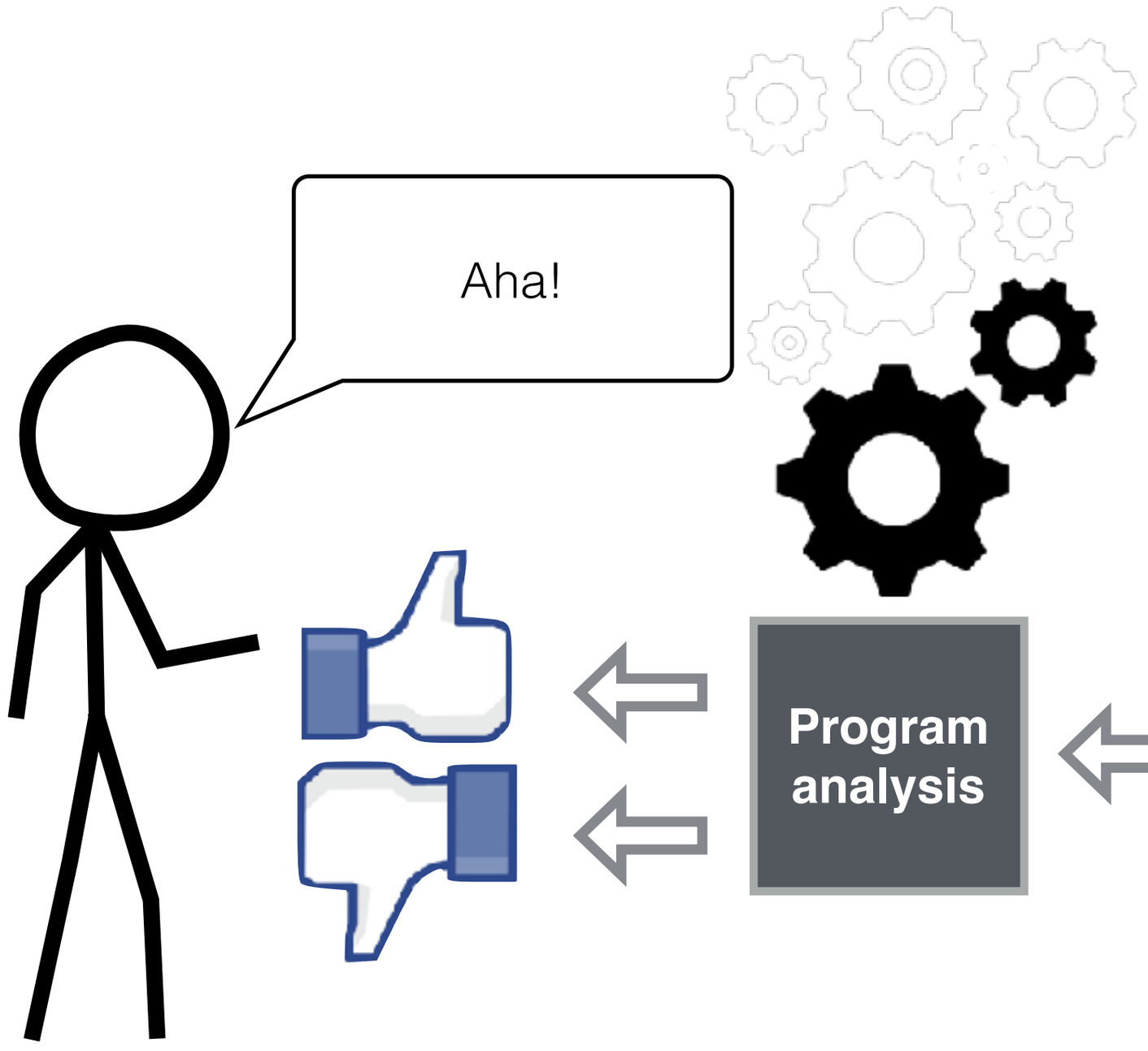

I want to compute my taxes
Okay!
I can do that for you. Just run this code!
tax_computation_automator.exe


I want to compute my taxes
Okay!
I can do that for you. Just run this code!
…
aliceServer.send(bobSalary)
…

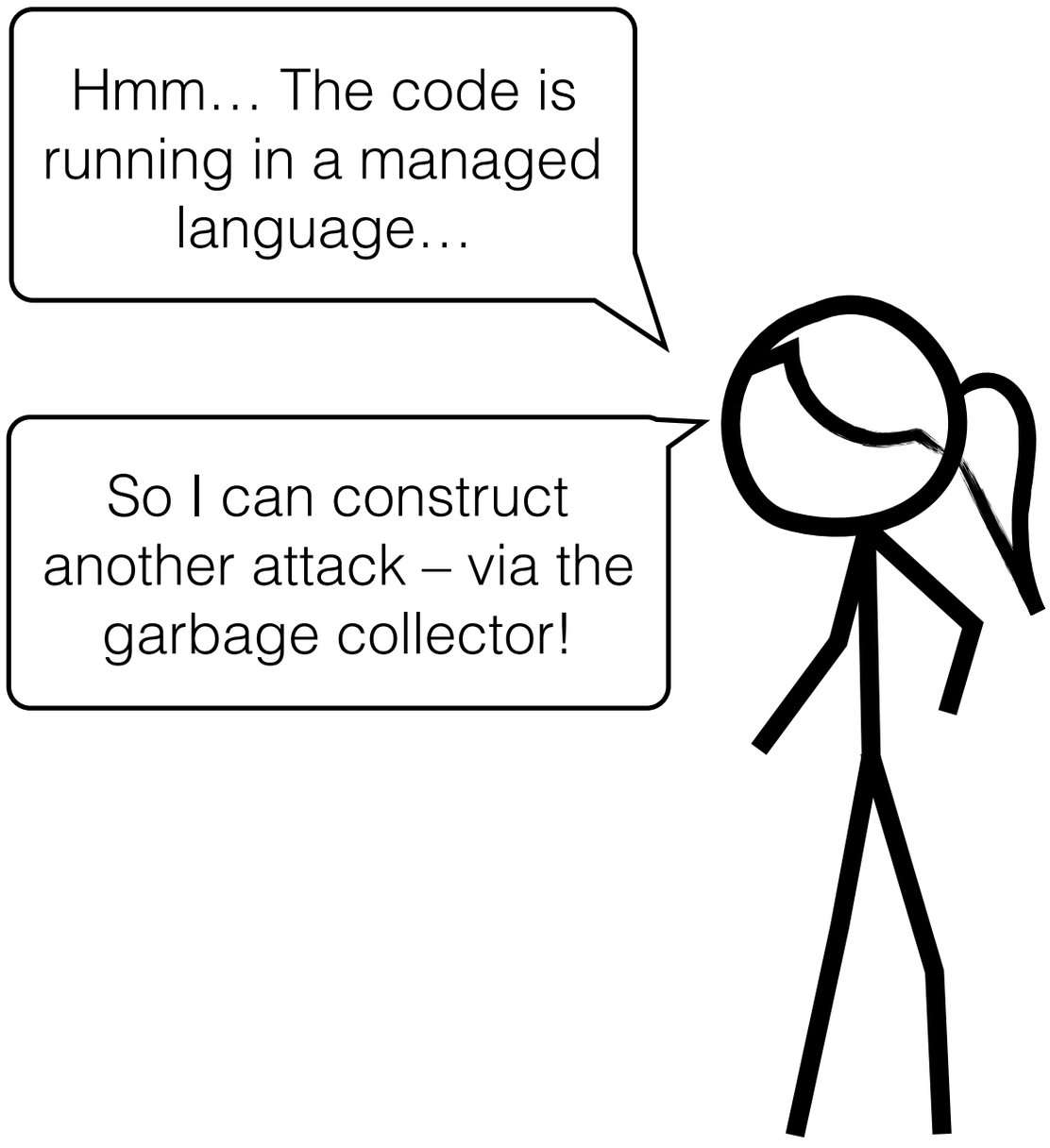
Bob could have been smart


```
…
aliceServer.send(bobSalary)
…
```

Bob could have been smart

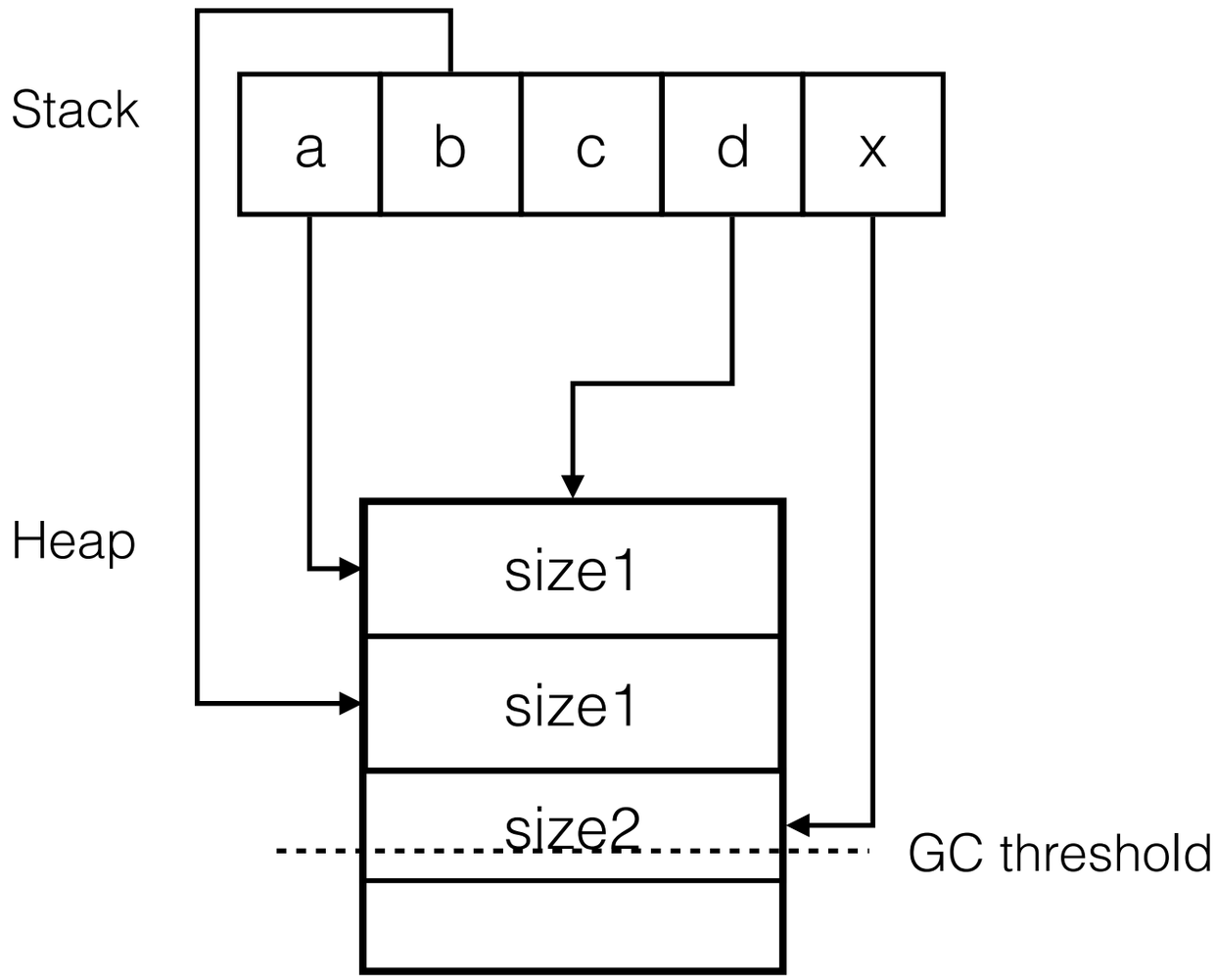


Alice goes to work…



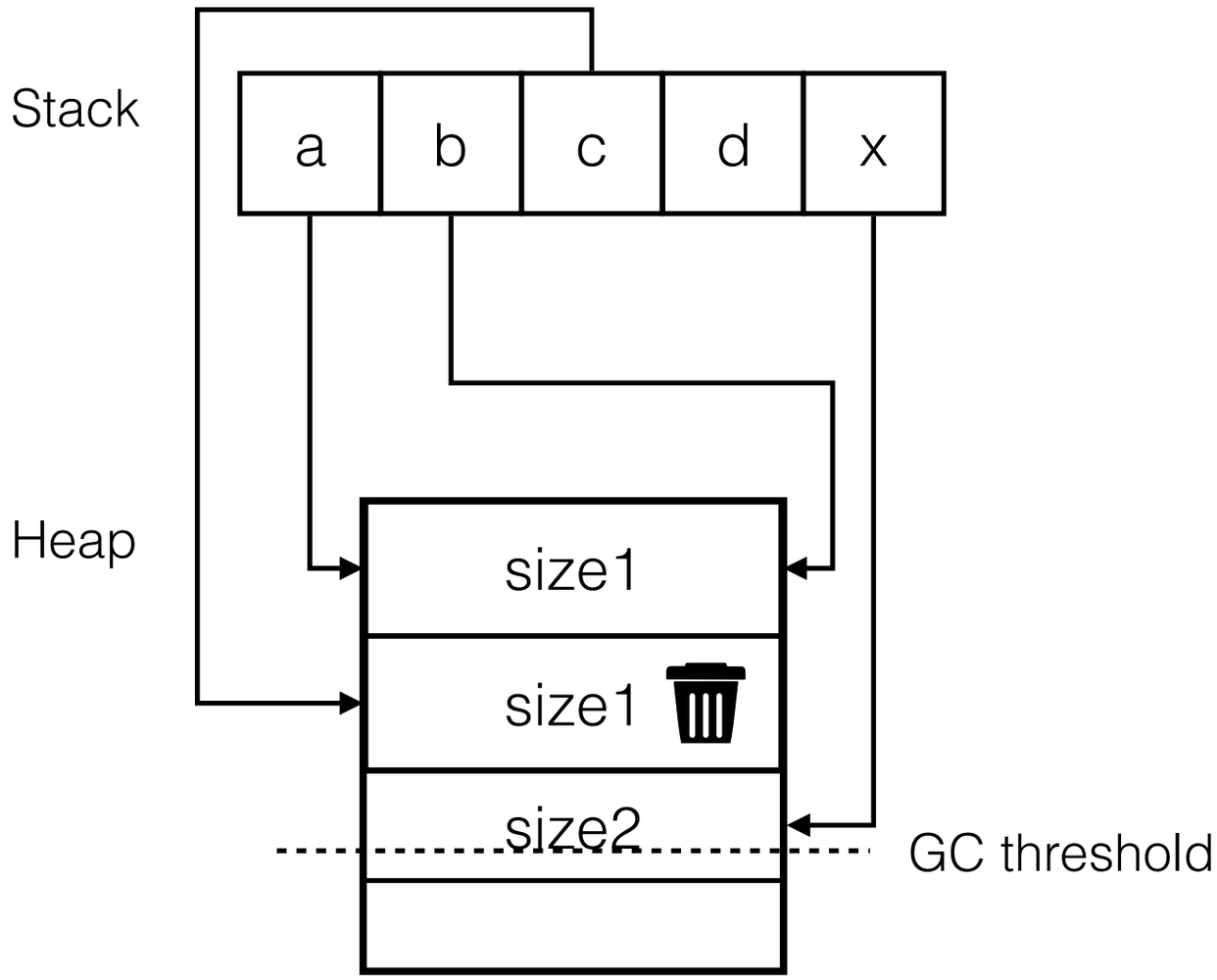
Attack: Leak via GC

```
int[] a = new int[size1];
int[] b = null;
int[] c = null;
int[] d = null;
if (bobSalary > 500000) {
  b = new int[size1];
  d = a;
}
else {
  c = new int[size1];
  b = a;
}
c = null;
long before = System.nanoTime();
int[] x = new int[size2];
long after = System.nanoTime();
System.out.println(after - before);
```



Attack: Leak via GC

```
int[] a = new int[size1];
int[] b = null;
int[] c = null;
int[] d = null;
if (bobSalary > 500000) {
  b = new int[size1];
  d = a;
}
else {
  c = new int[size1];
  b = a;
}
c = null;
long before = System.nanoTime();
int[] x = new int[size2];
long after = System.nanoTime();
System.out.println(after - before);
```



Demo


```
java Leak1Bit 500000
java Leak1Bit 500001
```

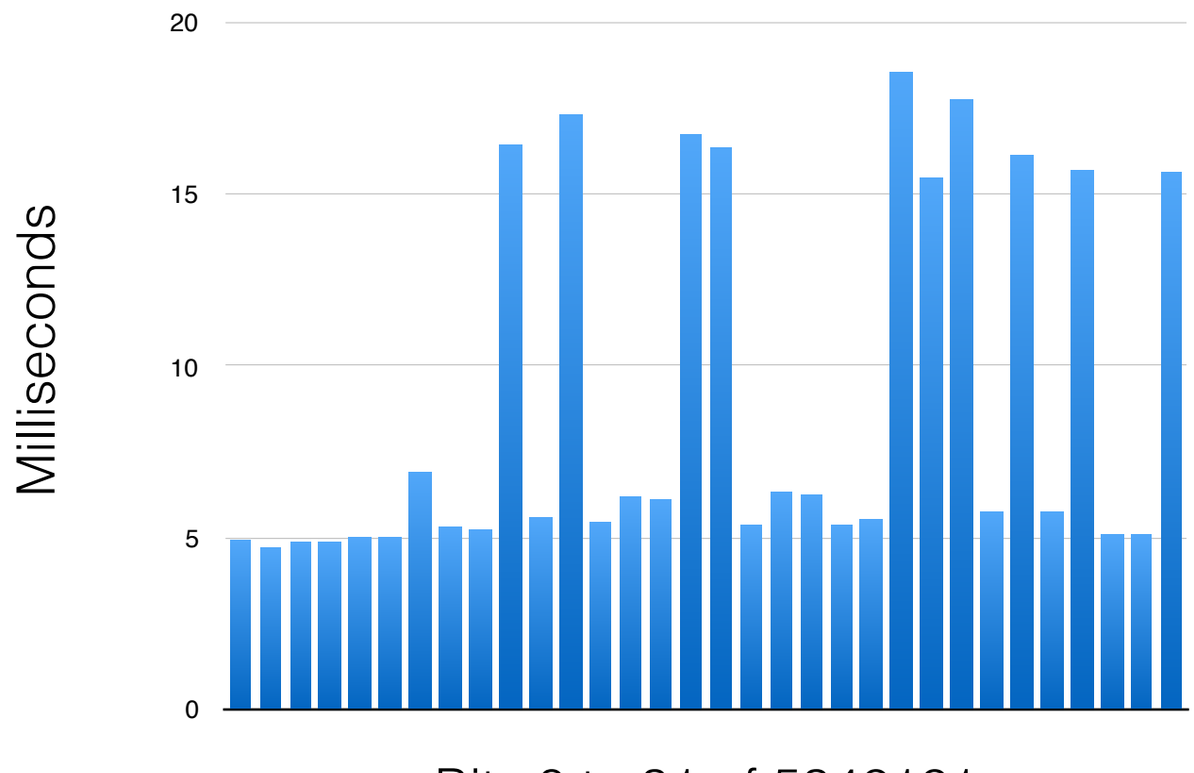
What is “small” and “large”?

Amplification to N bits

```
long[] times = new long[N];

for(int bit = 31; bit >= 0; --bit) {
  for(int i = 0; i < N; ++i) {
    int[][] a = new int[K][size];
    int[][] b;
    int[][] c;
    int[][] d;
    if(((secret >> bit) & 1) > 0) {
      b = new int[K][size];
      d = a;
    }
    else {
      c = new int[K][size];
      b = a;
    }
    long before = System.nanoTime();
    int[] c = new int[size2];
    long after = System.nanoTime();
    if(after - before > threshold) {
      times[i] = after - before;
    }
    else {
      times[i] = 0;
    }
  }

  long sum = 0;
  long gcs = 0;
  for(int i = 0; i < N; ++i) {
    long t = times[i];
    t += times[i];
    if(t != 0) ++gcs;
  }
  if(gcs == 0) {
    ++bit; continue;
  }
  System.out.println(sum / gcs);
}
```

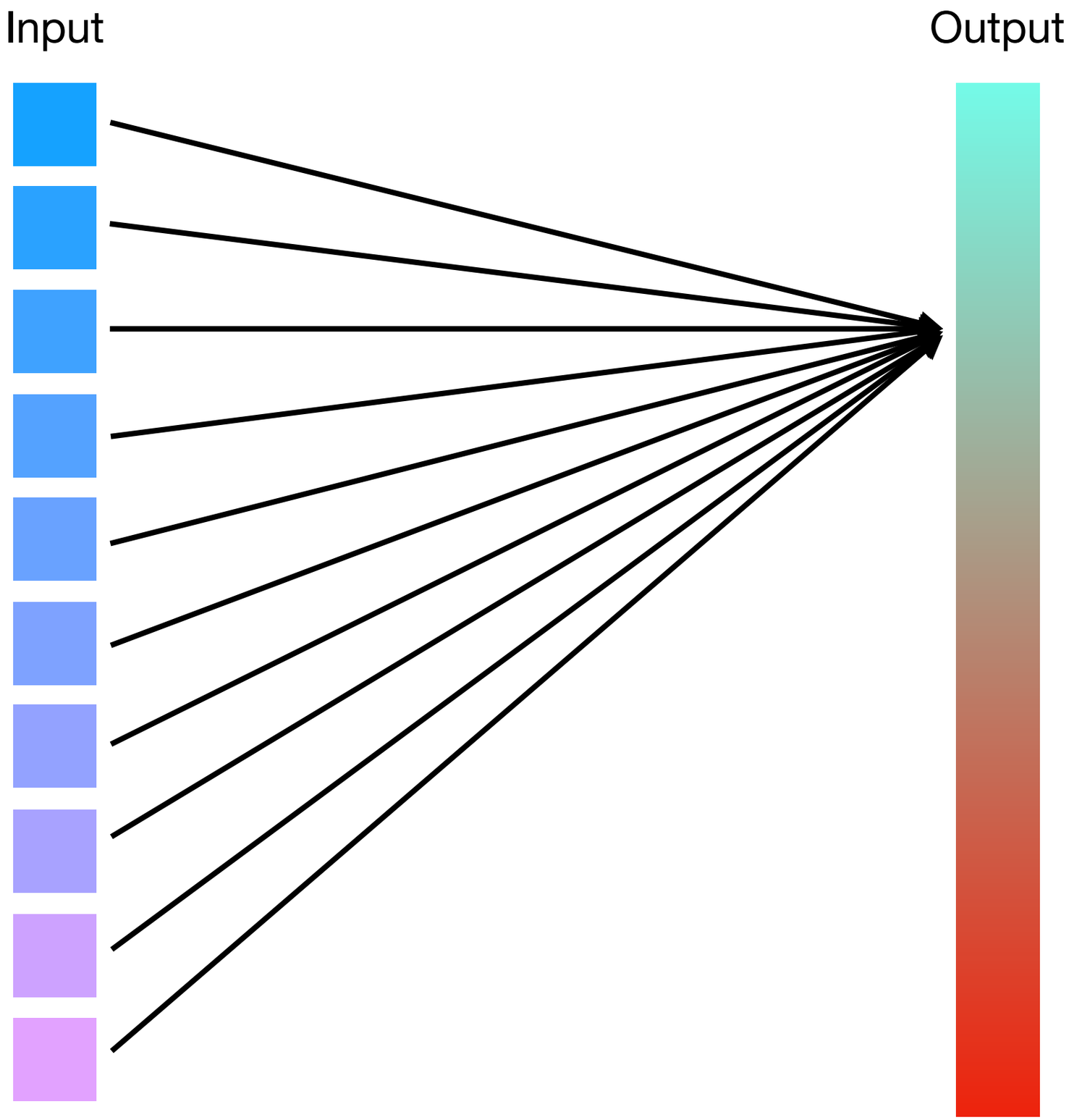


Bits 0 to 31 of 5342121

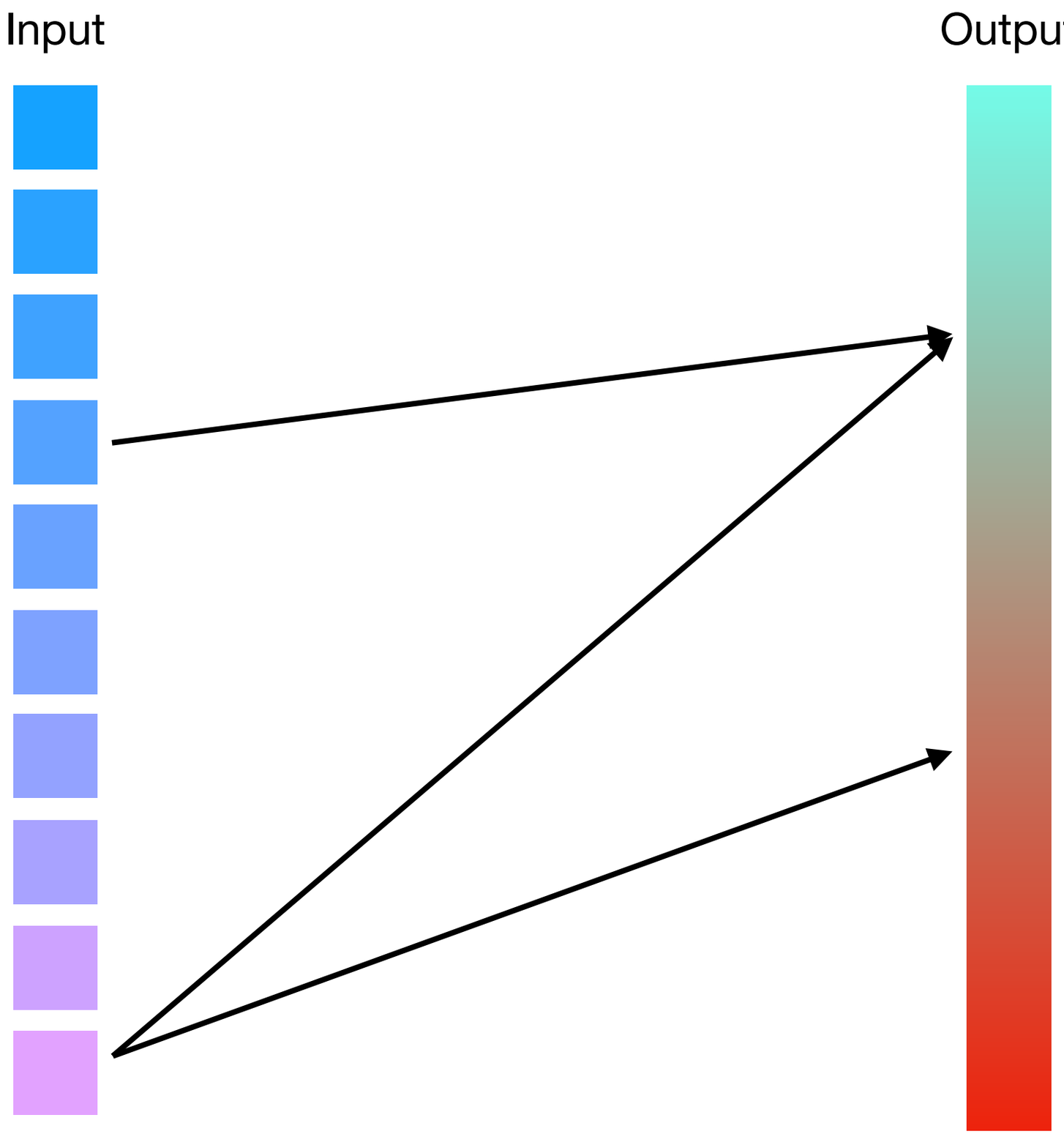
Also works in V8

Rate: 0.98 byte/s

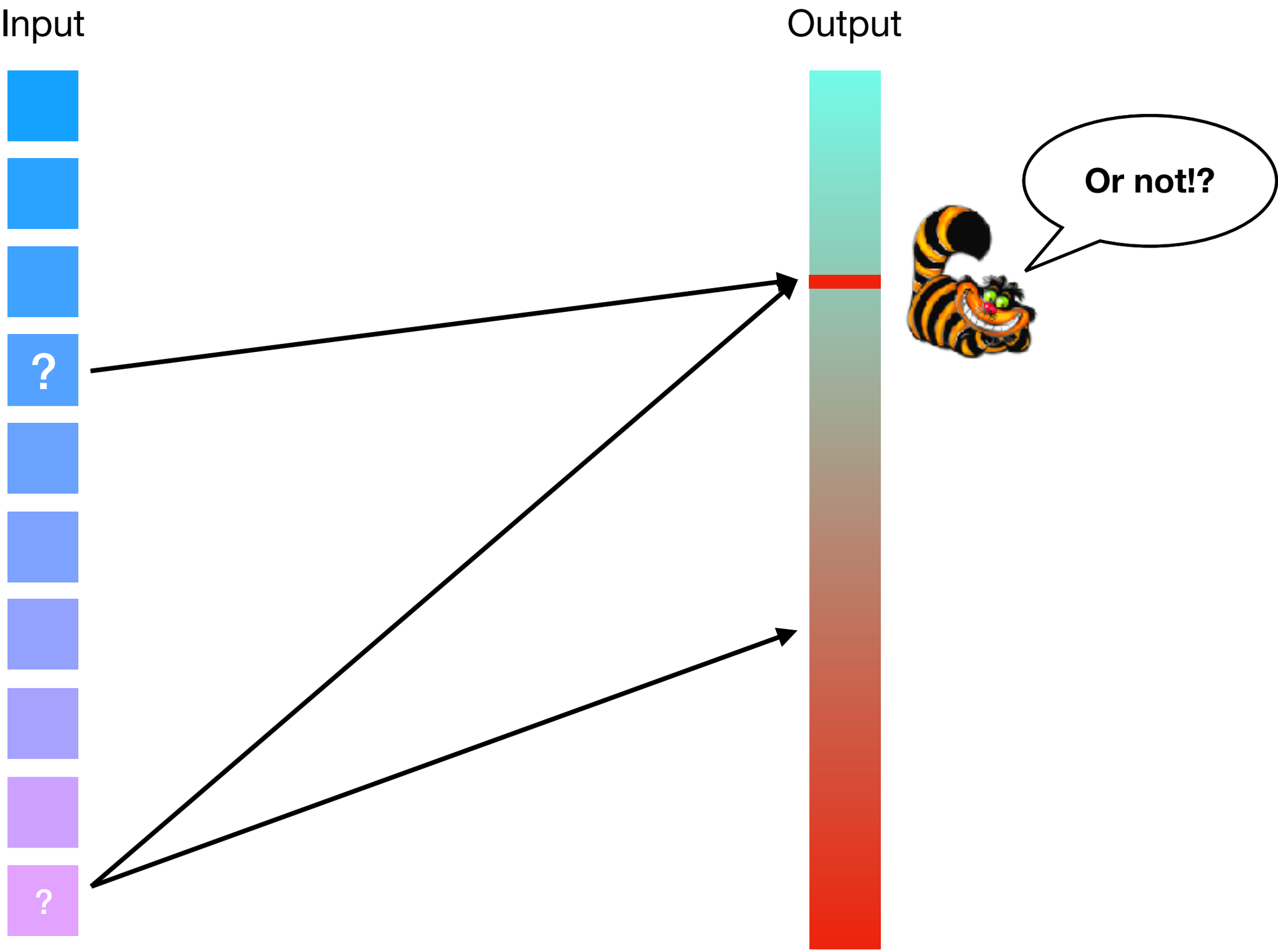
Noninterference



Possibilistic noninterference



Possibilistic noninterference



Main result

Theorem:

Well-typed programs, when run under secure GC, satisfy possibilistic termination-insensitive noninterference.

Formalized in Coq

Conclusion

- Automatic memory management provides an amplifiable timing-channel detectable over a network connection.

- The channel can be closed with standard information flow techniques.

- Meta-theory proved in the Coq proof assistant (35k lines of code)

Questions?