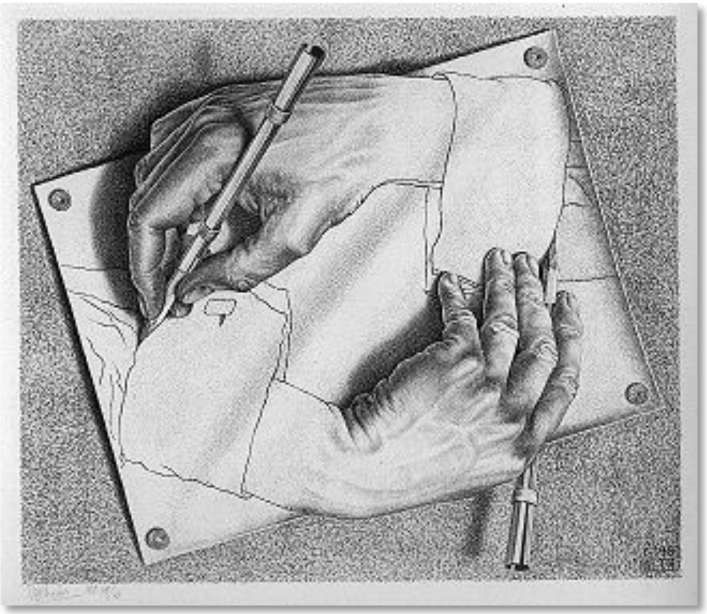
Programming Languages and Techniques (CIS120)

Bonus Lecture

Code *is* Data

Code *is* Data



M.C. Escher, Drawing Hands, 1948

Code *is* Data

- A Java source file is just a sequence of characters.
- We can represent programs with Strings!

```
String p_0 = "class C { public static void main(String args[])
String p_1 = "class C { public static void main(String args[])
String p_2 = "class C { public static void main(String args[])
String p_3 = "class C { public static void main(String args[])
  {System.out.println(\"Hel\");}}";
String p_13 = "class C { public static void main(String args[])
  {System.out.println(\"Hello, world!\");}}";
```

…

```
String p_81201202342312310 = /* PennPals! */
      "class ServerMain { public static void main(String args[]) {…}}";
```

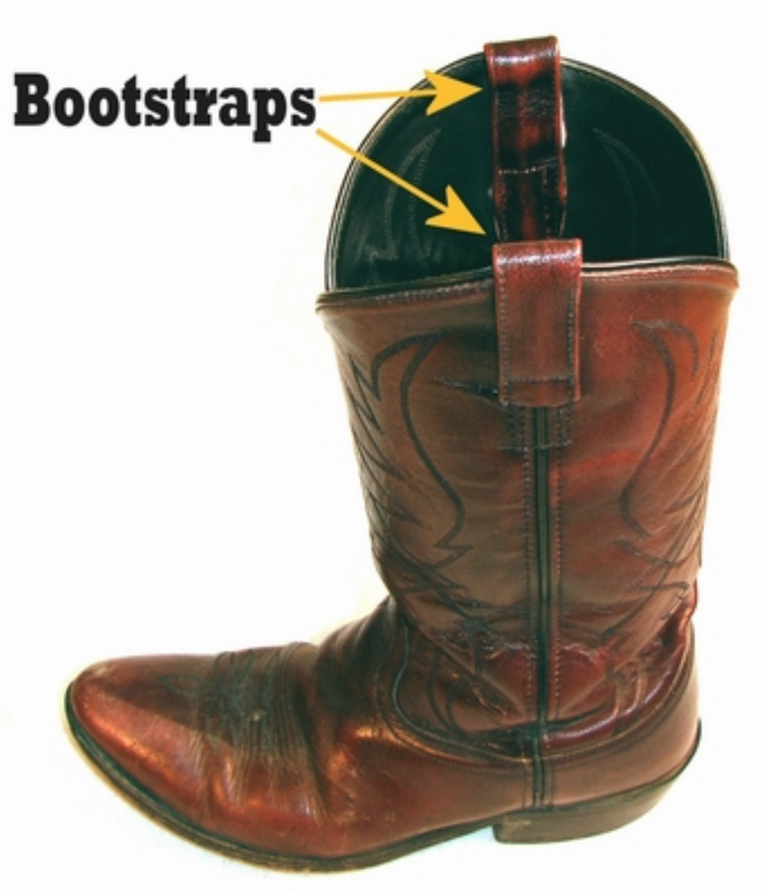
…

```
String p_9999324900090230023940082340702340 = /* Minecraft! */
      "class Minecraft { public static void main(String args[]) {…}}";
```

…

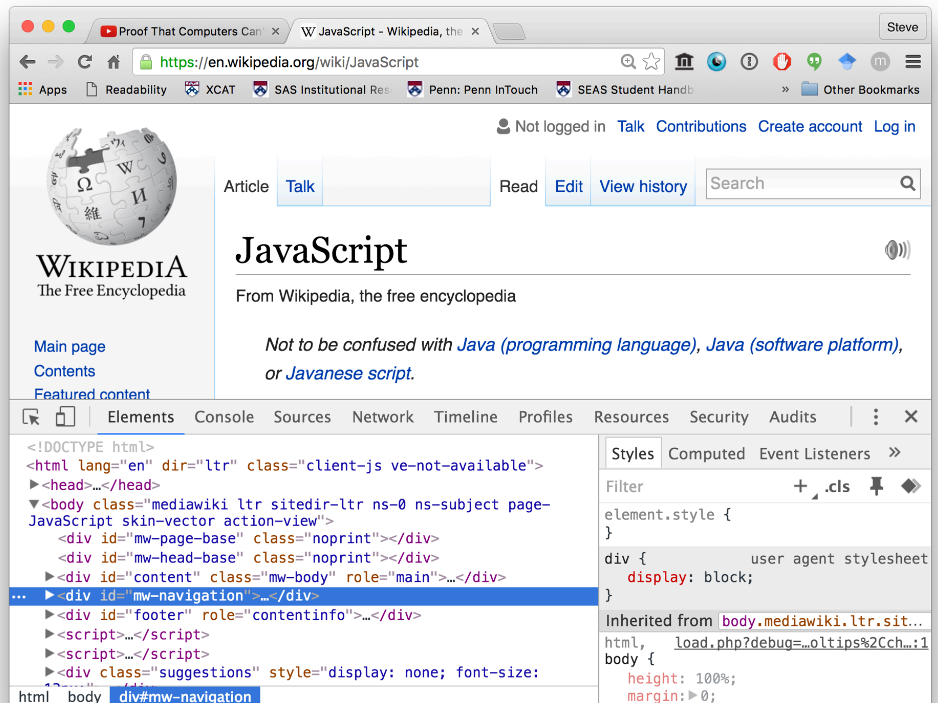
```
String p_9923499234239999324900023428234073450234534 = /* Eclipse! */
      "class Eclipse { public static void main(String args[]) {…}}";
```

Consequence 1: Programs that manipulate programs



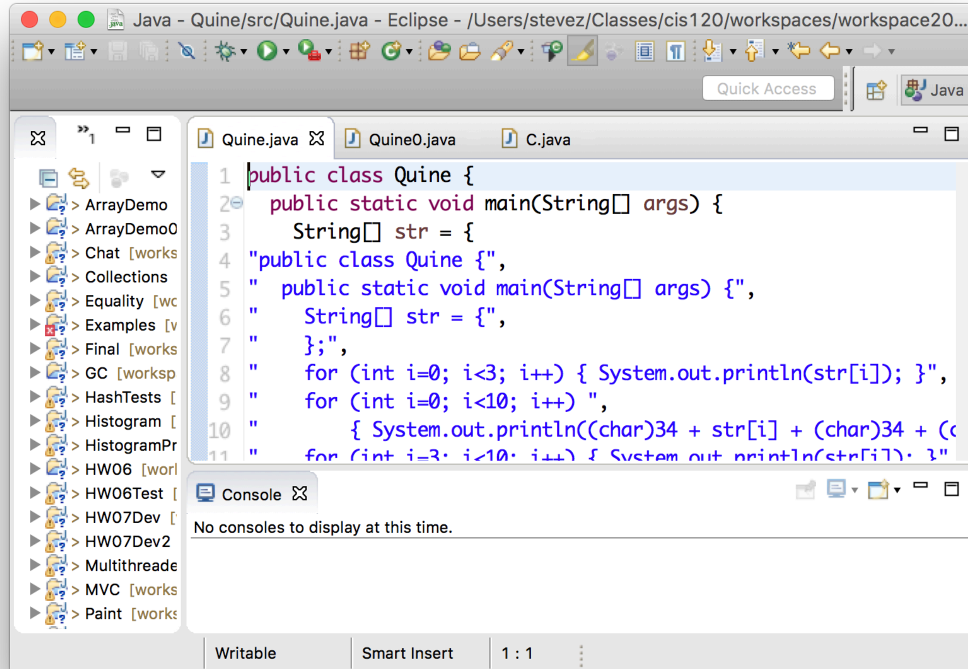
Interpreters

- We can create *programs* that manipulate *programs*
- An *interpreter* is a program that executes other programs
- `interpret ("3 + 4")` ➔ 7
- Example 1: JavaScript



IDEs and Compilers

- Example 2: Eclipse
  - Eclipse manipulates a *representation* of Java programs
  - Eclipse itself is written in Java
  - So you could use Eclipse to edit the code for… Eclipse?!



- Example 3: Compiler
  - The Java compiler takes a String representation of a Java program
  - It outputs a "low-level" representation of the program as a .class file (i.e. Java byte code)
  - Can also compile to other representations, e.g. x86 "machine code"

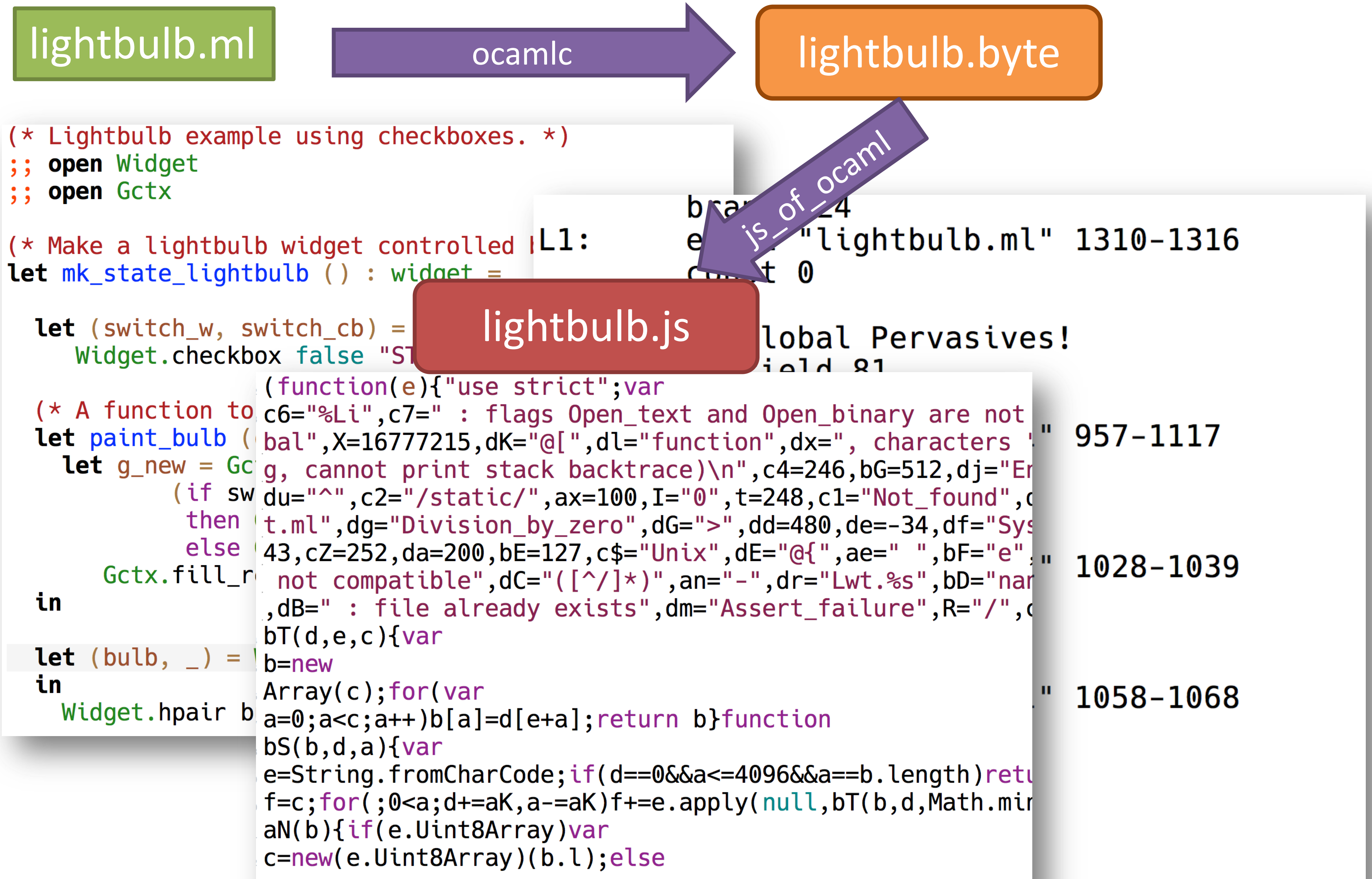
Example: OCaml native code compiler

lightbulb.ml → ocamlopt → lightbulb.native

```
(* Lightbulb example using checkboxes. *)
;; open Widget
;; open Gctx

(* Make a lightbulb widget controlled by
let mk_state_lightbulb () : widget =

  let (switch_w, switch_cb) =
     Widget.checkbox false "STATE LIGHT"

  (* A function to display the bulb  *)
  let paint_bulb (g:gctx) : unit =
    let g_new = Gctx.with_color g
             (if switch_cb.get_value ()
              then Gctx.yellow
              else Gctx.black) in
      Gctx.fill_rect g_new (0, 99) (99,
  in

  let (bulb, _) = Widget.canvas (100,100)
  in
   Widget.hpair bulb switch_w
```

```
_camlLightbulb__mk_state_lightbulb_1253:
00000001000017d0        subq    $0x8, %rsp
00000001000017d4        leaq    _camlLightbulb__1(%rip), %rbx
00000001000017db        movq    $0x1, %rax
00000001000017e2        callq   _camlWidget__checkbox_1349
00000001000017e7        movq    %rax, (%rsp)
00000001000017eb        movq    0x8(%rax), %rdi
00000001000017ef        subq    $0x20, %r15
00000001000017f3        leaq    _caml_young_limit(%rip), %rax
00000001000017fa        cmpq    (%rax), %r15
00000001000017fd        jb      0x100001840
00000001000017ff        leaq    0x8(%r15), %rbx
0000000100001803        movq    $0xcf7, -0x8(%rbx)
000000010000180b        leaq    _camlLightbulb__paint_bulb_1256
0000000100001812        movq    %rax, (%rbx)
0000000100001815        movq    $0x3, 0x8(%rbx)
000000010000181d        movq    %rdi, 0x10(%rbx)
0000000100001821        leaq    _camlLightbulb__6(%rip), %rax
0000000100001828        callq   _camlWidget__canvas_1330
000000010000182d        movq    (%rsp), %rbx
0000000100001831        movq    (%rbx), %rbx
0000000100001834        movq    (%rax), %rax
0000000100001837        addq    $0x8, %rsp
000000010000183b        jmp     _camlWidget__hpair_1272
0000000100001840        callq   _caml_call_gc
0000000100001845        jmp     0x1000017ef
0000000100001847        nopw    (%rax,%rax)
```

Example: OCaml byte code



Example Compilation: Java to X86

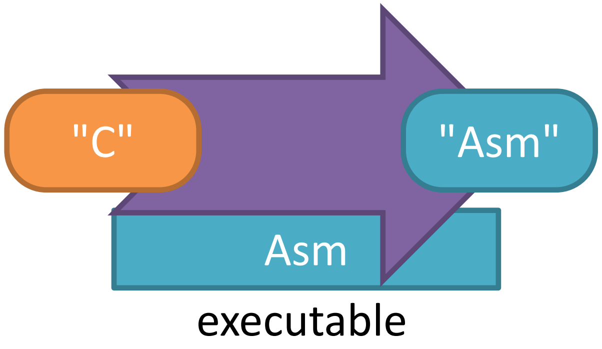
```java
class Point {
  int x;
  int y;
  Point move(int
  int dy) {
    x = x + dx;
    y = y + dy;
    return this;
  )
}
```

```
.globl __fun__Point.move
__fun__Point.move:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
__5:
    movl 8(%ebp), %eax
    movl 4(%eax), %eax
    movl %eax, -4(%ebp)
```

How Could That Work?

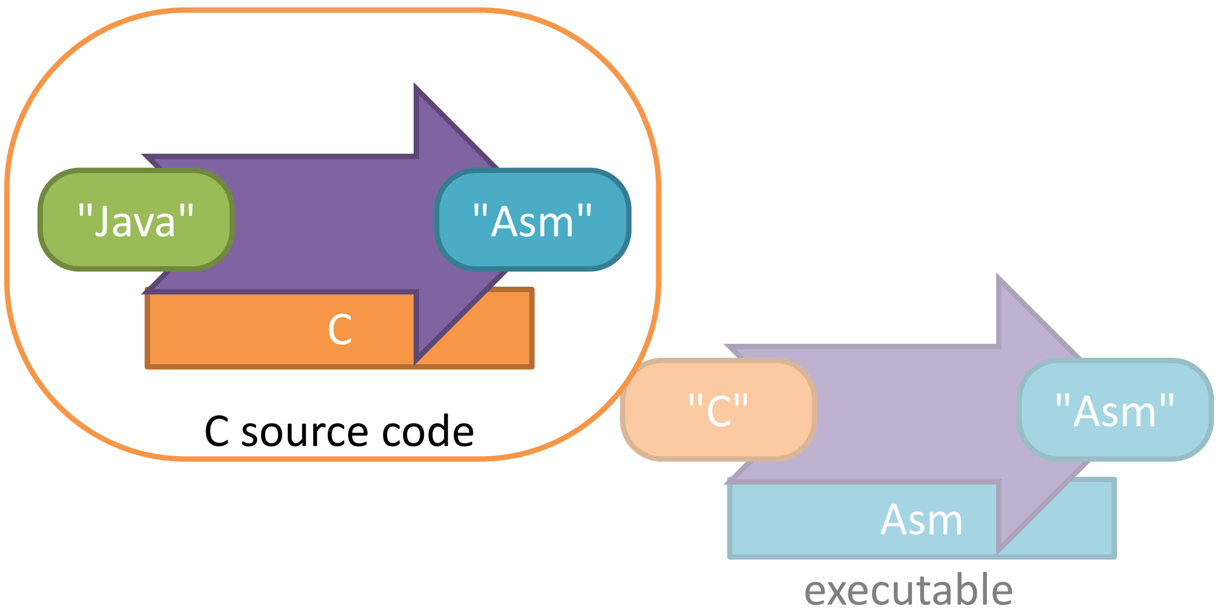
Bootstrap Process

① Implement, by hand in assembly, a compiler for a low-level language like C



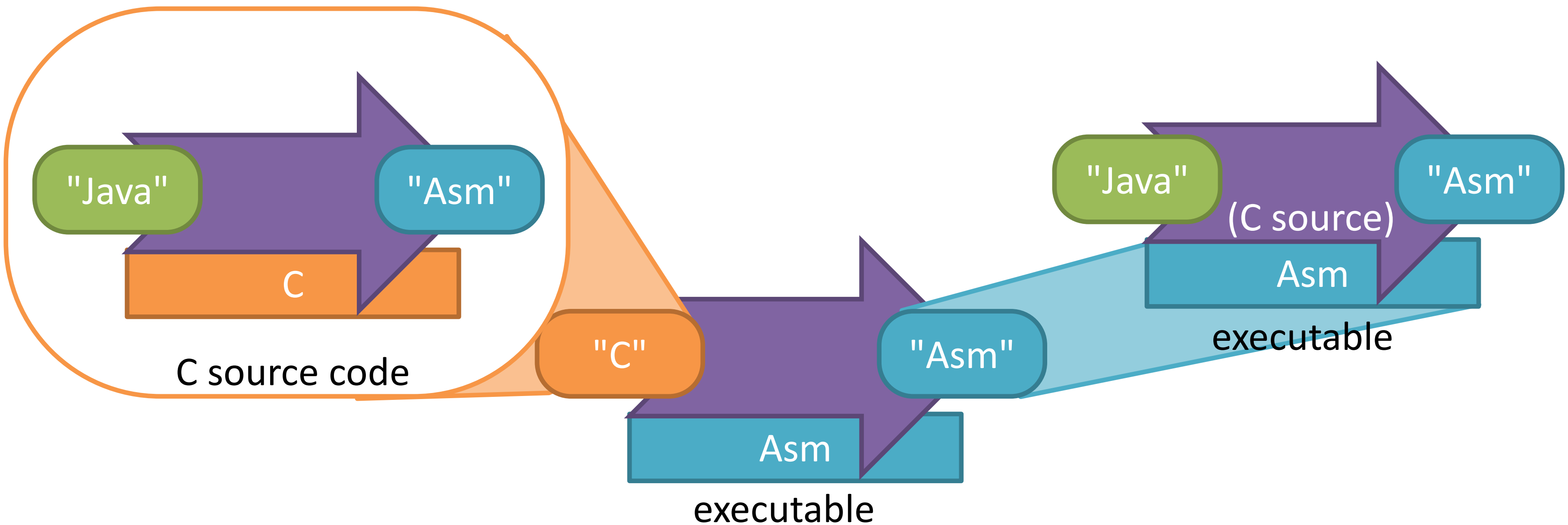
Bootstrap Process

2 Write, in C, a Java-to-Assembly compiler



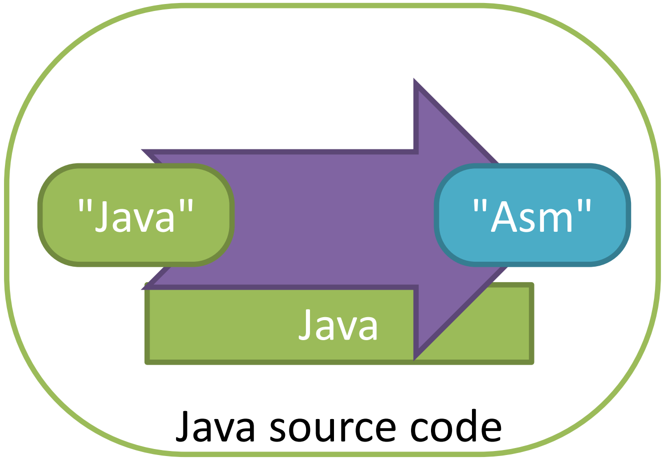
Bootstrap Process

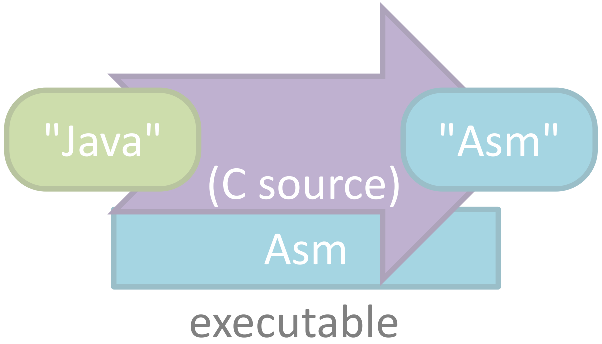
3 Use the C compiler to compile the Java compiler, yielding a Java compiler executable



Bootstrap Process

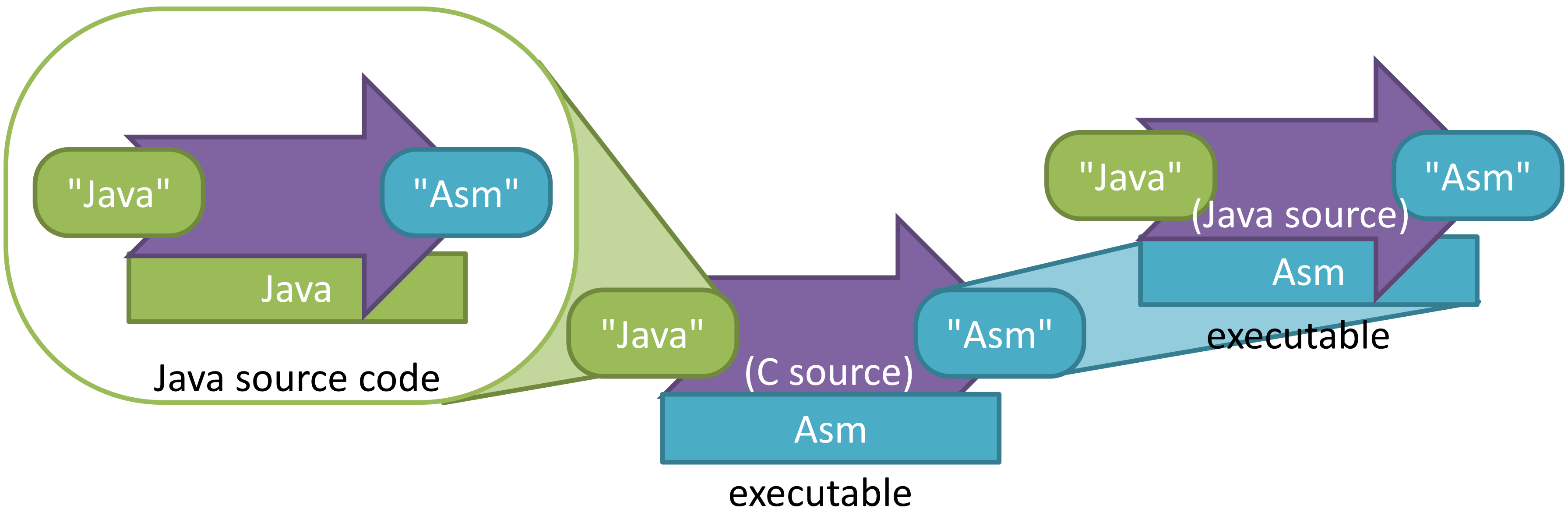
4 Write a Java compiler in Java





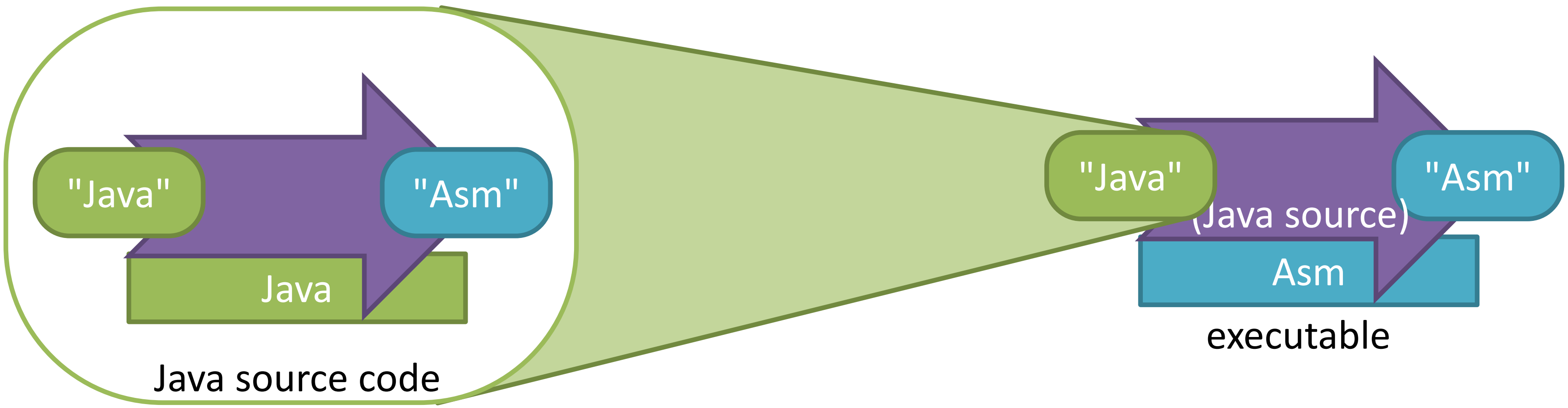
Bootstrap Process

5 Use the (C source) Java compiler to compile the Java compiler (Java source) implementation



Bootstrap Process

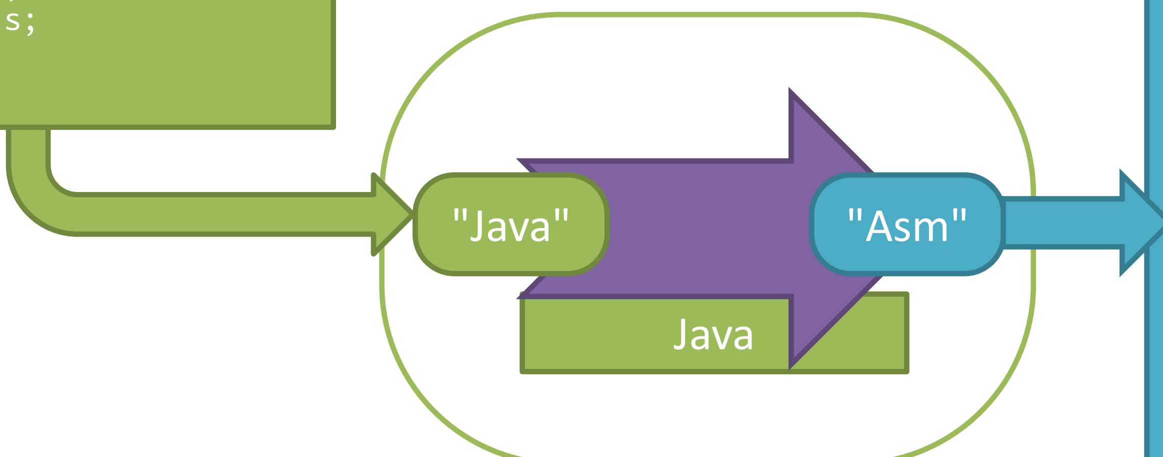
6 Throw all the earlier stuff away and use the Java compiler executable (derived from Java source) from now on



Bootstrap Process

7 Profit! Compile Java programs to Asm!

```
class Point {
  int x;
  int y;
  Point move(int dx, int dy) {
    x = x + dx;
    y = y + dy;
    return this;
  )
}
```




```
.globl __fun__Point.move
__fun__Point.move:
    pushl %ebp
    movl %esp, %ebp
    subl $4, %esp
__5:
    movl 8(%ebp), %eax
    movl 4(%eax), %eax
    movl %eax, -4(%ebp)
    movl 12(%ebp), %ecx
    addl %ecx, -4(%ebp)
    movl -4(%ebp), %ecx
    movl 8(%ebp), %eax
    movl %ecx, 4(%eax)
    movl 8(%ebp), %eax
    movl 0(%eax), %eax
    movl %eax, -4(%ebp)
    movl 16(%ebp), %ecx
    addl %ecx, -4(%ebp)
    movl -4(%ebp), %ecx
    movl 8(%ebp), %eax
```

Consequence 2: Malware

Rene Magritte, The Human Condition, 1933

Consequence 2: Malware

- Why does Java do array bounds checking?
- *Unsafe* language like C and C++ don’t do that checking;
  - They will happily let you write a program that “writes past” the end of an array.

- Result:
  - viruses, worms, “jailbreaking”, Spam, botnets, …

Fundamental issue:

- Code is data

Consider this C Program

```
void m() {
  char[] buffer = new char[2];

  char c = read();
  int i = 0;
  while (c != -1) {
      buffer[i] = c;
      c = read();
      i++;
  }
  process(buffer);
}

void main() {
  m();
  // do some more stuff
}
```

Notes:
- C doesn't check array bounds
- Unlike Java, it stores arrays directly on the stack
- What could possibly go wrong?

Abstract Stack Machine

"Stack Smashing Attack"

Abstract Stack Machine

Workspace

Stack

```
m ();
// do some more stuff
```

Call to main() to start the program…

Abstract Stack Machine

Workspace

```
char[2] buffer;

char c = read();
int i = 0;
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```

Push the saved workspace, run m()

Abstract Stack Machine

Workspace

```
char c = read();
int i = 0;
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```

buffer

Allocate space for buffer on the stack.

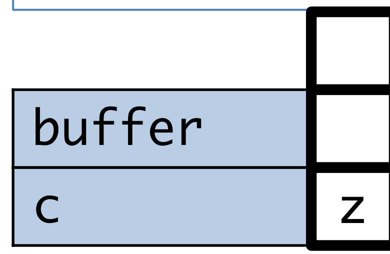
Abstract Stack Machine

Workspace

```
int i = 0;
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```



Allocate space for c.
Read the first user input... 'z'.

Abstract Stack Machine

Workspace

```
while (c != -1) {
   buffer[i] = c;
   c = read();
   i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```

buffer	
c	z
i	0

Allocate space for i.

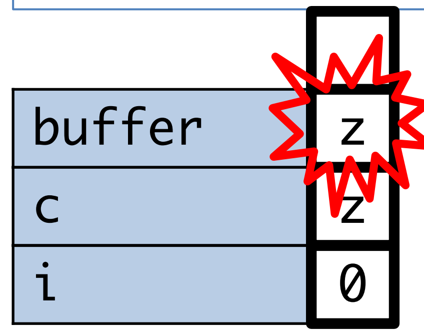
Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```



Copy (contents of) c to buffer[0]

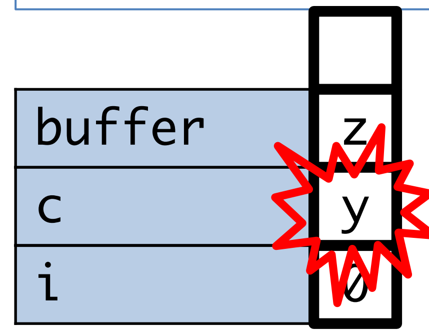
Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```



Read next character … 'y'

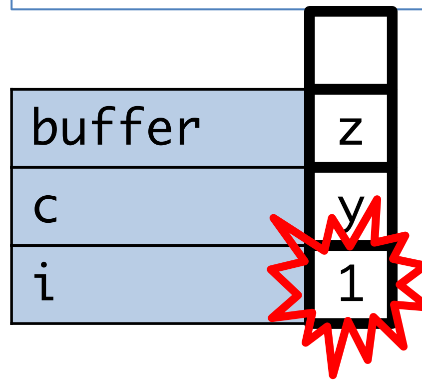
Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```



Increment i

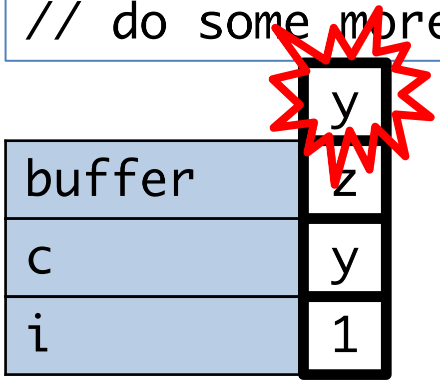
Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```



Copy (contents of) c to buffer[1]

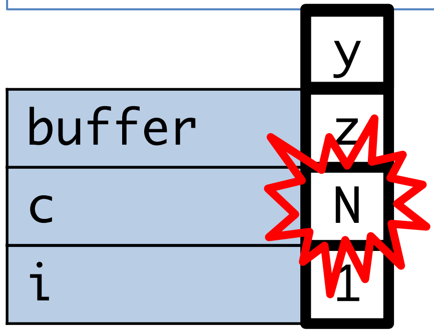
Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```



Read next character ... 'N'

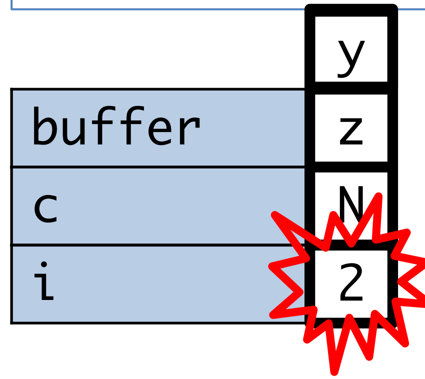
Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

```
_;
// do some more stuff
```



Increment i

Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack

N do some more stuff

	y
buffer	z
c	N
i	

Copy (contents of) c to buffer[2] ?!?

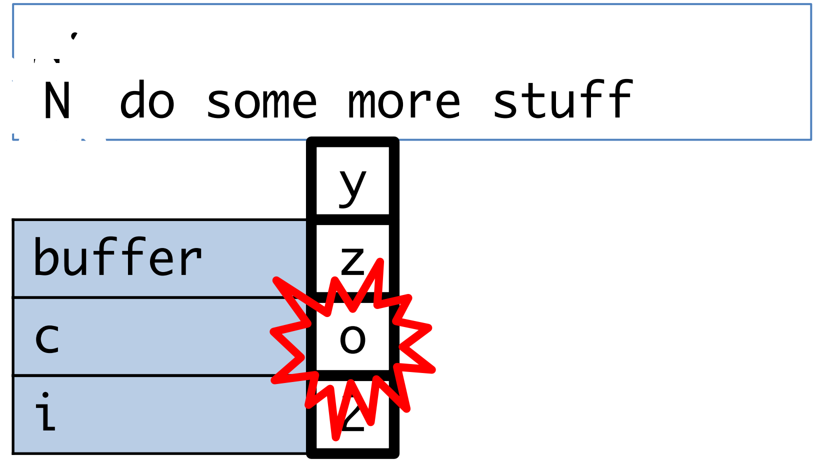
Overwrites the saved workspace!?

Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack



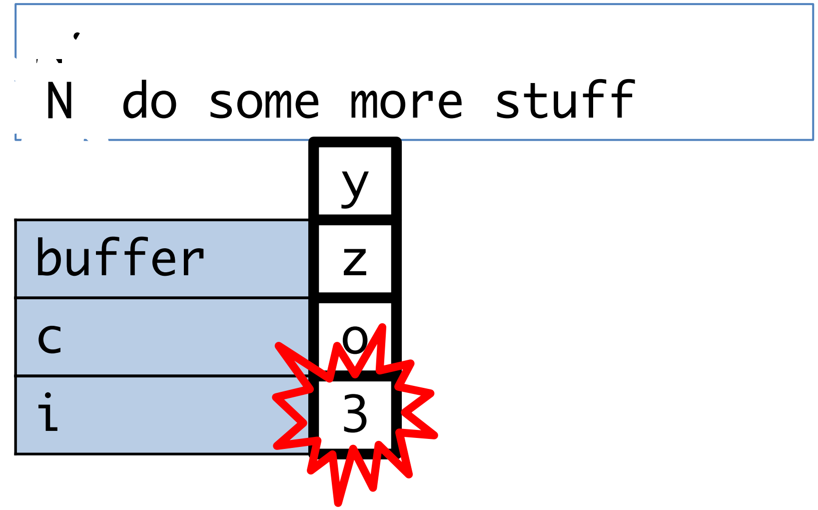
Keep going… read 'o'…

Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack



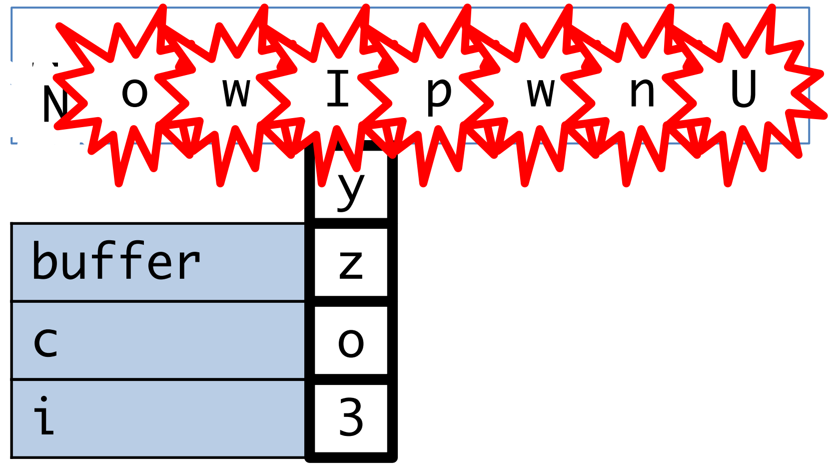
Keep going... read 'o'...increment i...

Abstract Stack Machine

Workspace

```
while (c != -1) {
  buffer[i] = c;
  c = read();
  i++;
}
process(buffer);
```

Stack



Keep going… read 'o'…increment i…write 'o' into saved workspace…

Abstract Stack Machine

Workspace

Stack

```
Now I pwn U!!!!
```

buffer	z
c	o
i	3

Later…

Abstract Stack Machine

Workspace

Stack

```
Now I pwn U!!!!
```

The stack smashing attack successfully wrote *arbitrary* code into the program's workspace…

Is this a big deal?

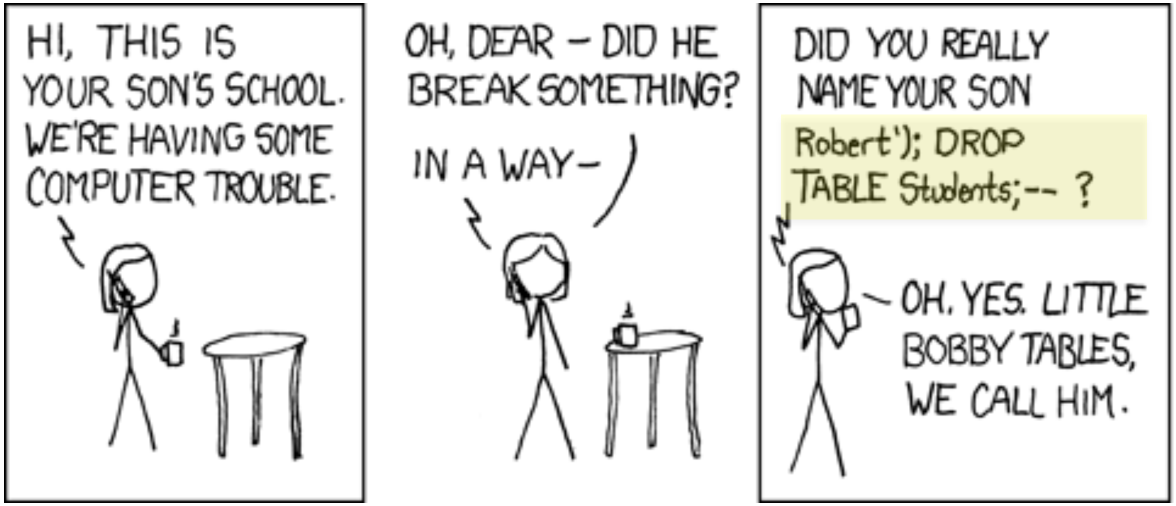
Yep

Other Code Injection Attacks

```
void registerStudent() {
  print("Welcome to student registration.");
  print("Please enter your name:");
  String name = readLine();
  evalSQL("INSERT INTO Students('" + name + "')"  );
}
```

```
"INSERT INTO Students('Robert'); DROP TABLE Students; --')"
```

```
+ "Robert'); DROP TABLE Students; --" + "')"
```



http://xkcd.com/327/

Consequence 3: Undecidability

Undecidability Theorem

<u>Theorem</u>: It is impossible to write a method

```
boolean halts(String prog)
```

such that, for any valid Java program represented as a string `p`,

```
halts(p)
```

returns true exactly when the program halts and false otherwise.

Alonzo Church, April 1936

Alan Turing, May 1936

Halt Detector

- Suppose we could write such a program:

```
class HaltDetector {
  public static boolean halts(String javaProgram) {
    // ...do some super-clever analysis...
    // return true if javaProgram halts
    // return false if javaProgram does not halt
  }
}
```

- A **correct** implementation of HaltDetector.halts(p) always returns either true or false
  - i.e., it never raises an exception or loops
- `HaltDetector.halts(p)` ⇒ `true` means “p halts”
- `HaltDetector.halts(p)` ⇒ `false` means “p loops forever”

Do these methods halt?

```
boolean m(){ return false; }
```

⇒ YES

```
boolean m(){ while (true) {} }
```

⇒ NO

```
boolean m() {
    if ("abc".length() == 3 ) return true;
    else return m();
}
```

⇒ YES

Does this method halt for *all* n?

```
boolean m (int n) {
  if (n<=1) return true;
  else if ((n%2) == 0) return m (n/2);
  else return m (3*n + 1);
}
⇒ ???
```

Assuming infinite amount of stack space and arbitrarily large integers, it is *unknown* whether this program halts for all n >= 1!

Collatz Conjecture (proposed in 1937, still open)!

What about Q?

```
class HaltDetector {
  public static boolean halts(String javaProgram) {
    // ...do some super-clever analysis...
    // return true if javaProgram halts
    // return false if javaProgram does not
  }
}

class Main {
  public static void Q() {
     String p_Q = ???;   // string representing Q
     if (HaltDetector.halts(p_Q)) {
       while (true) {}  // infinite loop!
     }
  }
}
```

What happens when we run Q?

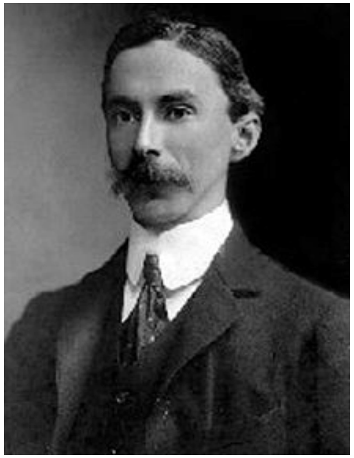
```
public static void Q() {
     String p_Q = ???;    // string representing Q
     if (HaltDetector.halts(p_Q)) {
       while (true) {}  // infinite loop!
     }
  }
```

if `HaltDetector.halts(p_Q)` ⇒ `true`   then Q loops (infinitely)

if `HaltDetector.halts(p_Q)` ⇒ `false`   then Q halts

Contradiction!

- Russell's Paradox (1901)
- Gödel's Incompleteness Theorem (1931)
- Both rely on *self reference*

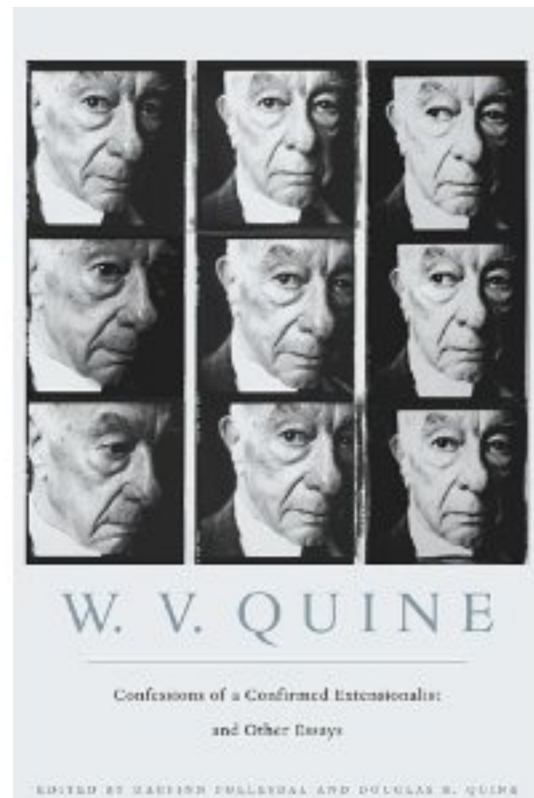


Bertrand Russell, 1901

Kurt Gödel, 1931

Potential Hole in the Proof

- What about the ??? in the program Q?
- It is supposed to be a String representing the program Q itself.
- How can that be possible?
- Answer: code is data!
  - And: there's more than one representation for the same data.
- See Quine.java



Profound Consequences

- The “halting problem” is *undecidable*
  - *There are problems that cannot be solved by a computer program!*
  - See https://www.youtube.com/watch?v=92WHN-pAFCs for a nice video...

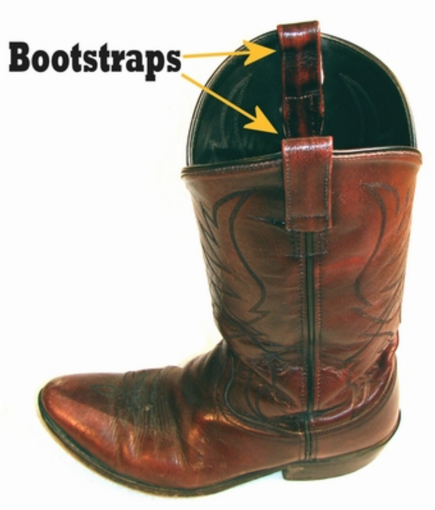
- Rice’s Theorem:
  - Every “interesting” property about computer programs is undecidable!
  - e.g., you can't test whether two functions have "equal behavior"

- You can’t write a perfect virus detector!
  (whether a program is a virus is certainly interesting)
  1. virus detector might go into an infinite loop
  2. it gives you false positives (i.e. says something is a virus when it isn’t)
  3. it gives you false negatives (i.e. it says a program is not a virus when it is)

- Also: You can’t write a perfect autograder!
  (whether a program is correct is certainly interesting)

Recommended Courses

- Programs that manipulate Programs
  - CIS 341: Compilers and interpreters
- Malware
  - CIS 331: Intro to Networks and Security
- Undecidability
  - CIS 262: Automata, Computability and Complexity



Recommended Reading