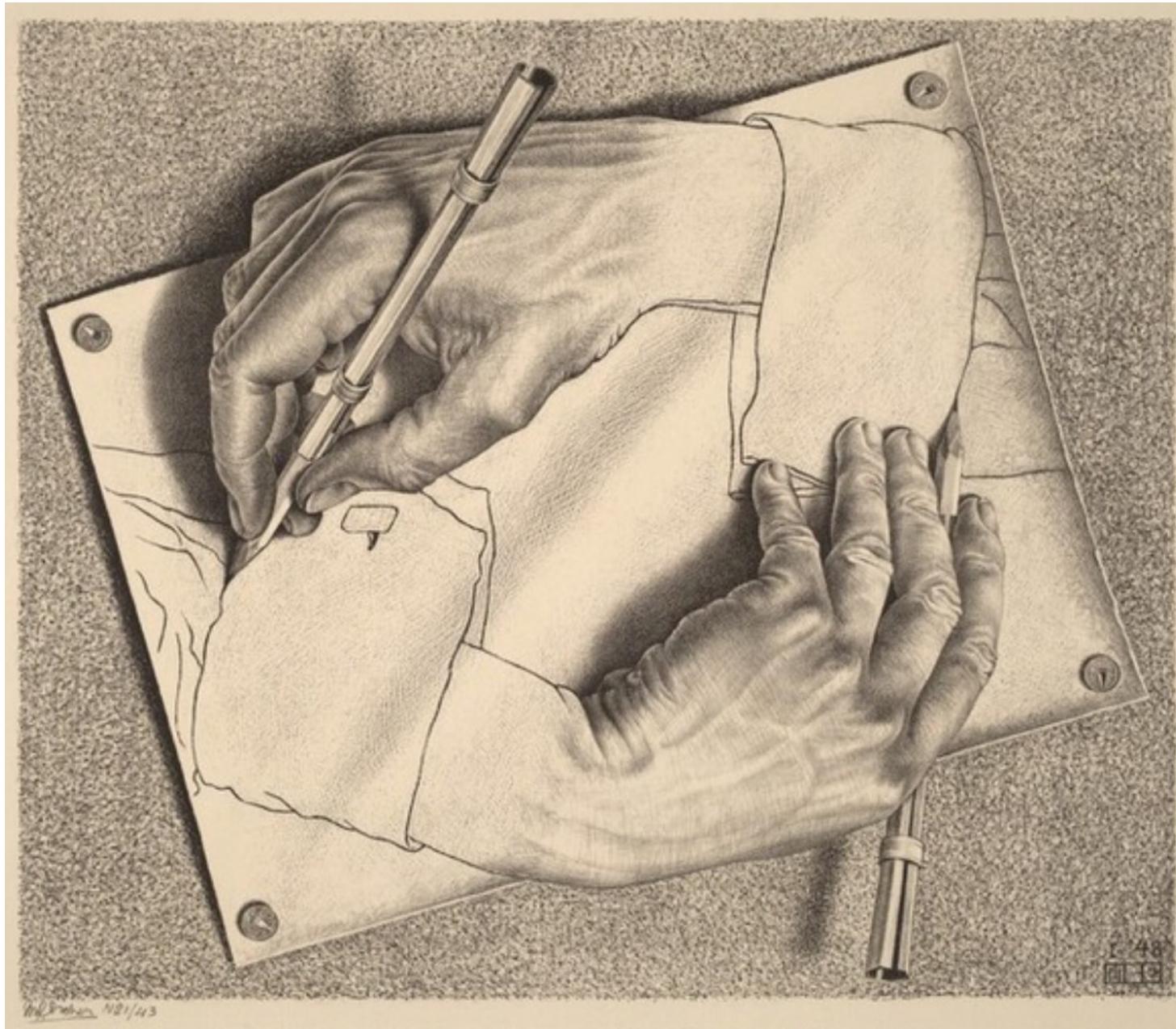


Programming Languages and Techniques (CIS1200)

Bonus Lecture

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 as Java 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(\"Hello, world!\");}}";
```

• • •

```
String p_8120120234231231230 = /* TwitterBot! */
    "class TwitterBot { public static void main(String args[]) {...}}";
```

• • •

```
String p_999932490009023002394008234070234 = /* Minecraft! */
    "class Minecraft { public static void main(String args[]) {...}}";
```

• • •

```
String p_99234992342399999324900023428234073450234534 = /* IntelliJ! */
    "class IntelliJ { public static void main(String args[]) {...}}";
```

Consequence 1: Programs that manipulate programs



Interpreters

- We can create *programs* that manipulate *programs*
- E.g., an *interpreter* is a program that *executes* other programs
 - E.g., interpret ("3 + 4") → 7
- Example 1: JavaScript



JavaScript

IDEs and Compilers

- Example 2:
IDEs like IntelliJ, Eclipse, Codio, VSCode, etc.
 - Manipulate a *representation* of Java programs
 - IntelliJ itself is written in Java
 - So, you could use IntelliJ to edit the code for... IntelliJ?!
- 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 (Java “bytecode”)
 - Can also compile to other representations, e.g., x86 machine code

The screenshot shows the IntelliJ IDEA interface with the Quine.java file open in the editor. The code is a classic quine program. The IntelliJ UI includes a Project tool window on the left, a Run tool window at the bottom, and various status indicators at the bottom right.

```
java -Quine.java
...
package org.cis120.bonus;
public class Quine {
    public static void main(String[] args) {
        String[] str = {
            "package org.cis120.bonus;",
            "",
            "public class Quine {",
            "    public static void main(String[] args) {",
            "        String[] str = {",
            "            \"package org.cis120.bonus;\"",
            "        };",
            "        for (int i=0; i<5; i++) { System.out.println(str[i]); }",
            "        for (int i=0; i<11; i++) { System.out.println((char)34 + str[i] + (char)34 + (char)34); }",
            "        for (int i=5; i<11; i++) { System.out.println(str[i]); }",
            "    }",
            "}",
            "}"
        };
    }
}
```

Aside: Grace Hopper

- 1940's: Mathematician working on the Mark-I computer at Harvard
- 1952: Implemented the first "compiler" A-0 (linker/loader) in Philadelphia



Grace Hopper working on the UNIVAC



Later went on to...

- Develop COBOL
- Become an admiral in the U.S. Navy
- Teach at Penn!
(we have an annual Grace Hopper colloquium in her honor and a plaque about the A-0 compiler)

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:
000000001000017d0    subq   $0x8, %rsp
000000001000017d4    leaq    _camlLightbulb__1(%rip), %rbx
000000001000017db    movq   $0x1, %rax
000000001000017e2    callq  _camlWidget__checkbox_1349
000000001000017e7    movq   %rax, (%rsp)
000000001000017eb    movq   0x8(%rax), %rdi
000000001000017ef    subq   $0x20, %r15
000000001000017f3    leaq    _caml_young_limit(%rip), %rax
000000001000017fa    cmpq   (%rax), %r15
000000001000017fd    jb     0x100001840
000000001000017ff    leaq    0x8(%r15), %rbx
00000000100001803    movq   $0xcf7, -0x8(%rbx)
0000000010000180b    leaq    _camlLightbulb__paint_bulb_1256
00000000100001812    movq   %rax, (%rbx)
00000000100001815    movq   $0x3, 0x8(%rbx)
0000000010000181d    movq   %rdi, 0x10(%rbx)
00000000100001821    leaq    _camlLightbulb__6(%rip), %rax
00000000100001828    callq  _camlWidget__canvas_1330
0000000010000182d    movq   (%rsp), %rbx
00000000100001831    movq   (%rbx), %rbx
00000000100001834    movq   (%rax), %rax
00000000100001837    addq   $0x8, %rsp
0000000010000183b    jmp    _camlWidget__hpair_1272
00000000100001840    callq  _caml_call_gc
00000000100001845    jmp    0x1000017ef
00000000100001847    nopw   (%rax,%rax)
```

Example: OCaml byte code

lightbulb.ml

ocamlc

lightbulb.byte

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

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

lightbulb.js

```
branch 24
entry "lightbulb.ml" 1310-1316
Count 0
```

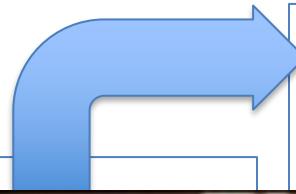
```
global Pervasives!
field 81
```

```
(function(e){"use strict";var
(* A function to paint a bulb *)
let paint_bulb (bal",X=16777215,dK="@[",dl="function",dx=", characters "
let g_new = Gc g, cannot print stack backtrace)\n",c4=246,bG=512,dj="Error"
(if sw du="^",c2="/static/",ax=100,I="0",t=248,c1="Not_found",c3="File
then t.ml",dg="Division_by_zero",dG=">",dd=480,de=-34,df="System error"
else 43,cZ=252,da=200,bE=127,c$="Unix",dE="@{",ae=" ",bF="e",c6="%
not compatible",dC="([/^]*)",an="-",dr="Lwt.%s",bD="name"
,dB=" : file already exists",dm="Assert_failure",R="/",c7="Flags Open_text and Open_binary are not
Gctx.fill_r
in
let (bulb, _) =
Widget.hpair b
in
Array(c);for(var
a=0;a<c;a++)b[a]=d[e+a];return b}function
bS(b,d,a){var
e=String.fromCharCode;if(d==0&&a<=4096&&a==b.length)retu
f=c;for(;0<a;d+=aK,a-=aK)f+=e.apply(null,bT(b,d,Math.min
aN(b){if(e.Uint8Array){var
c=new(e.Uint8Array)(b.l);else
```

js_of_ocaml

Example Compilation: Java to X86

```
class Point {  
    int x;  
    int y;  
    Point(int x, int y) {  
        this.x = x;  
        this.y = y;  
    }  
    void move() {  
        System.out.println("Moving to " + x + ", " + y);  
    }  
}
```



```
.globl __fun__Point.move  
__fun__Point.move:  
    pushl %ebp  
    movl %esp, %ebp
```

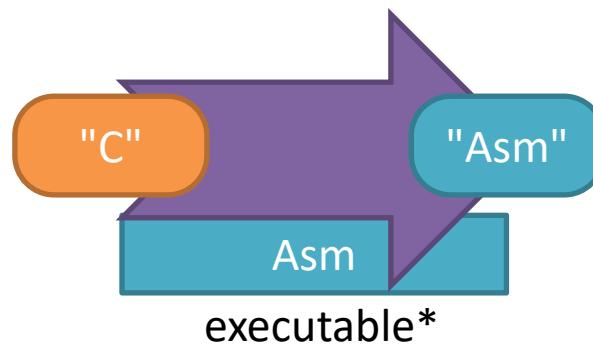


How Could That Work?

Bootstrap Process

1

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

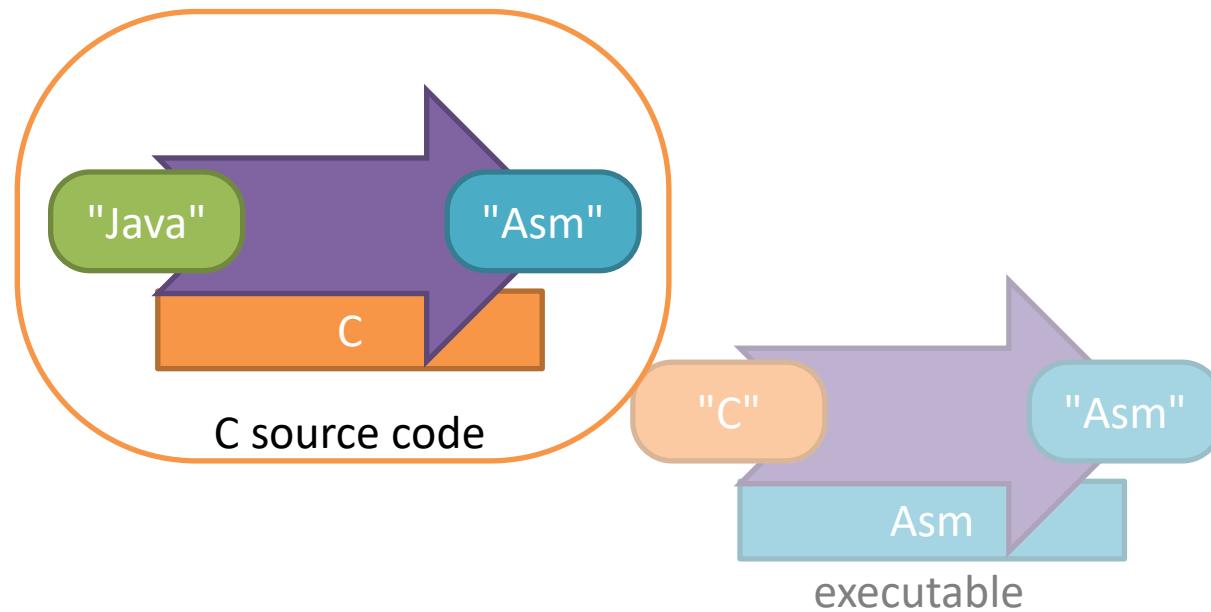


* Actually, you first need to write an “assembler” in machine code by hand. These slides conflate “assembly” with “machine code” to keep the story a bit simpler...

Bootstrap Process

②

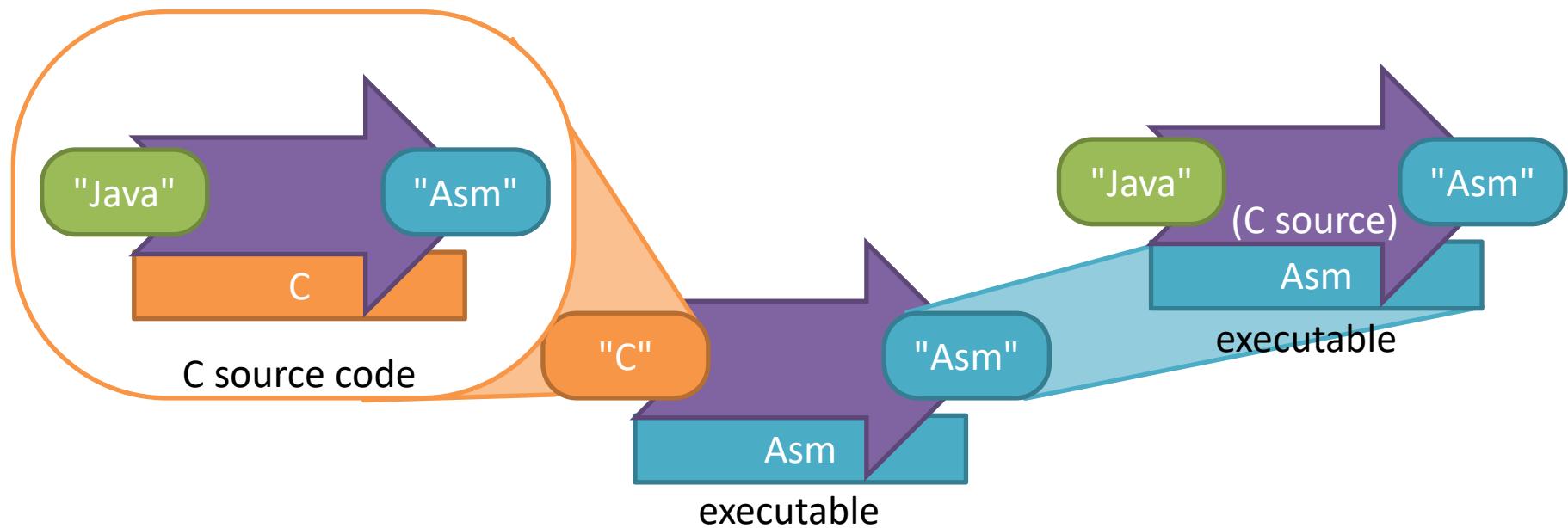
Write, in C, a Java-to-Assembly compiler



Bootstrap Process

3

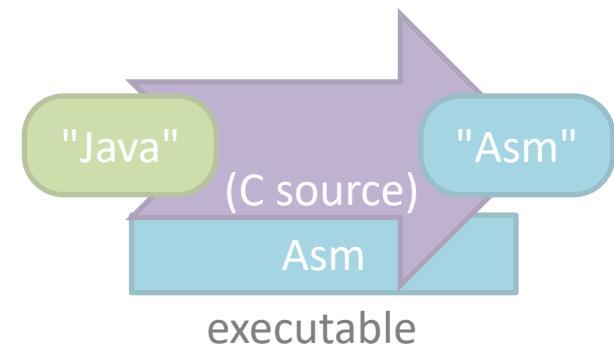
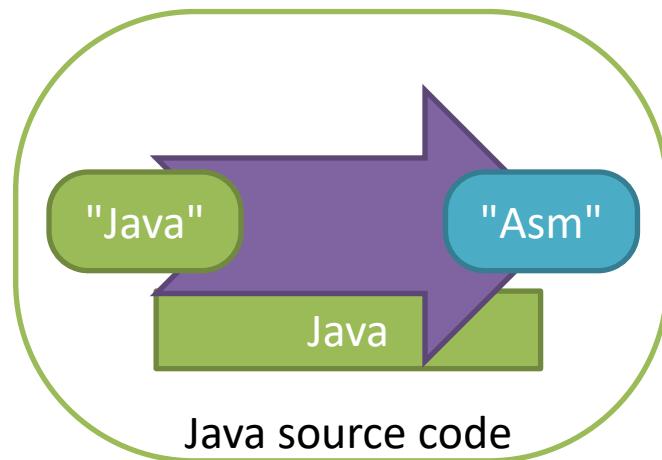
Use the C compiler to compile the Java compiler, yielding a Java compiler executable



Bootstrap Process

4

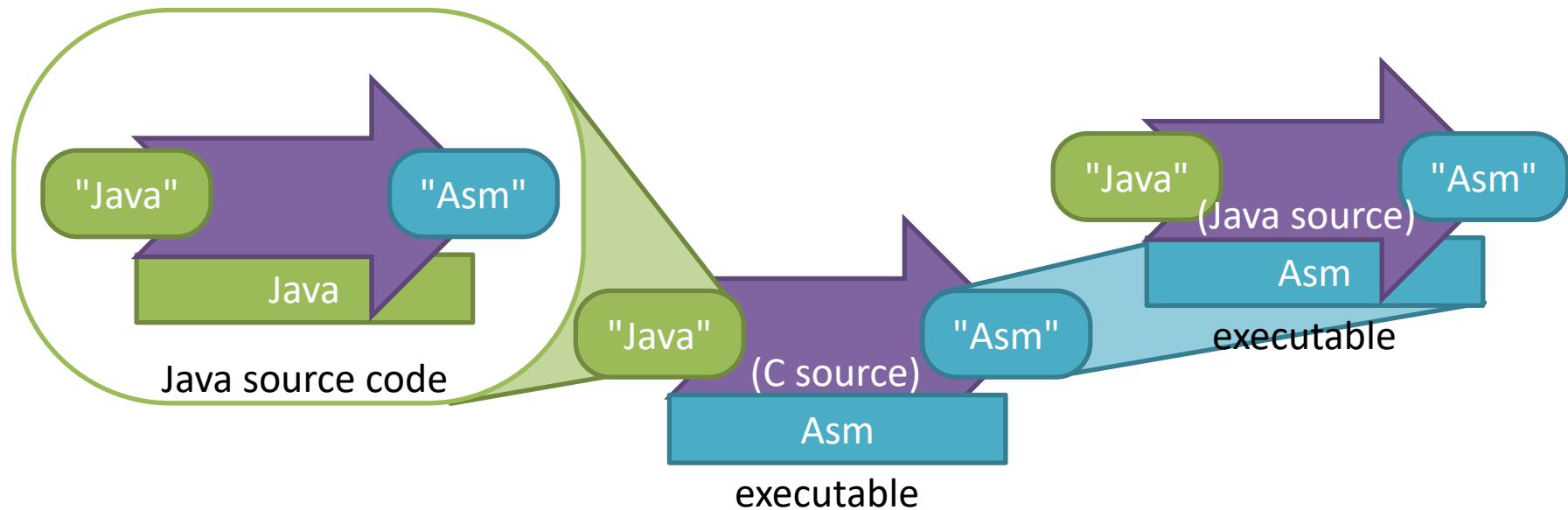
Write a Java compiler in Java



Bootstrap Process

5

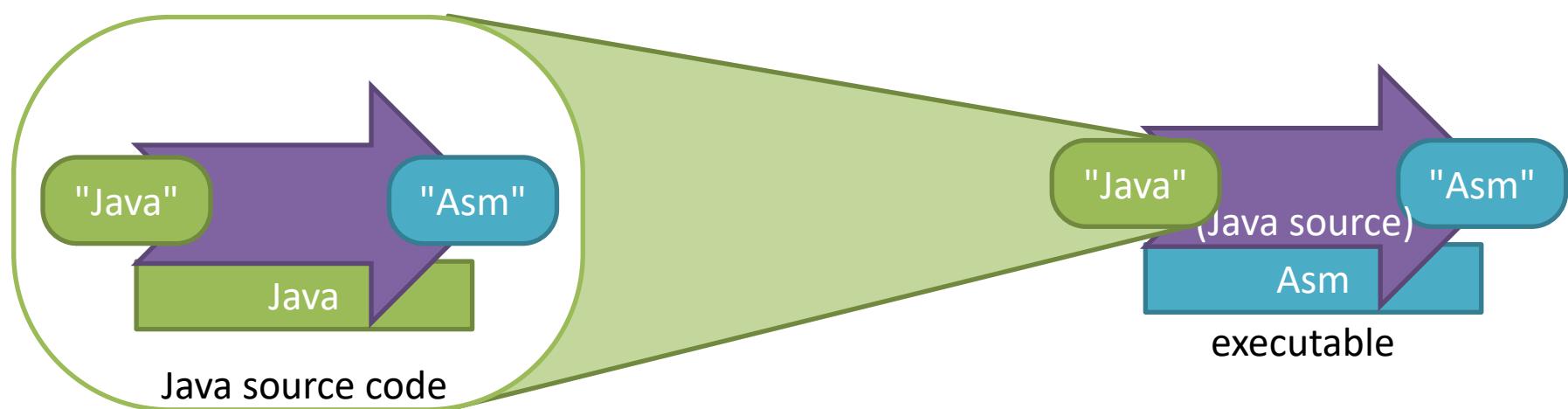
Use the (C source) Java compiler to compile the (Java source) Java compiler



Bootstrap Process

6

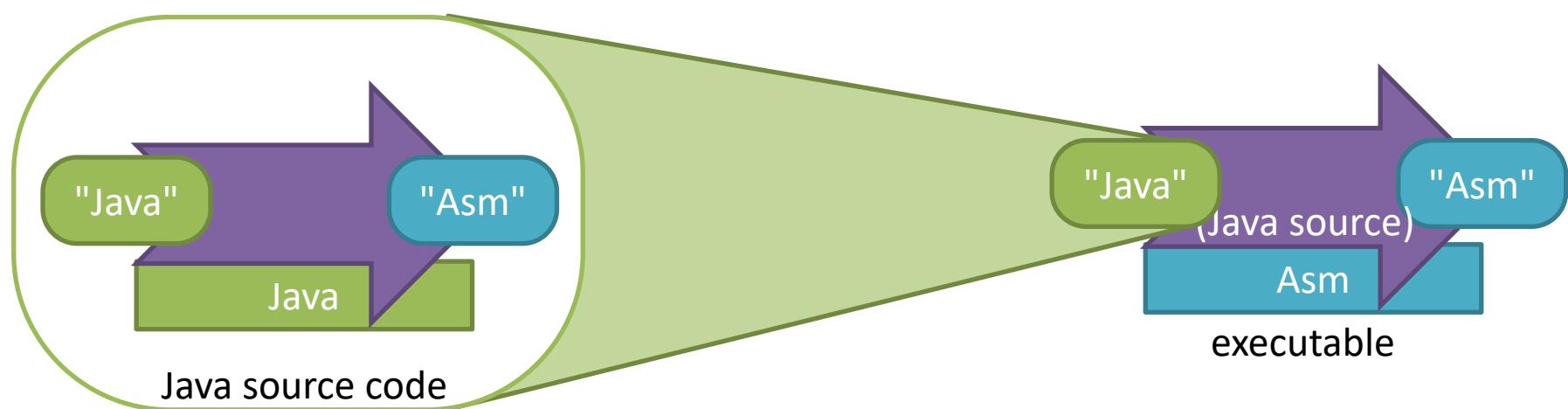
Throw all the earlier stuff away and use the Java compiler executable (derived from Java source) from now on



Bootstrap Process

6

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

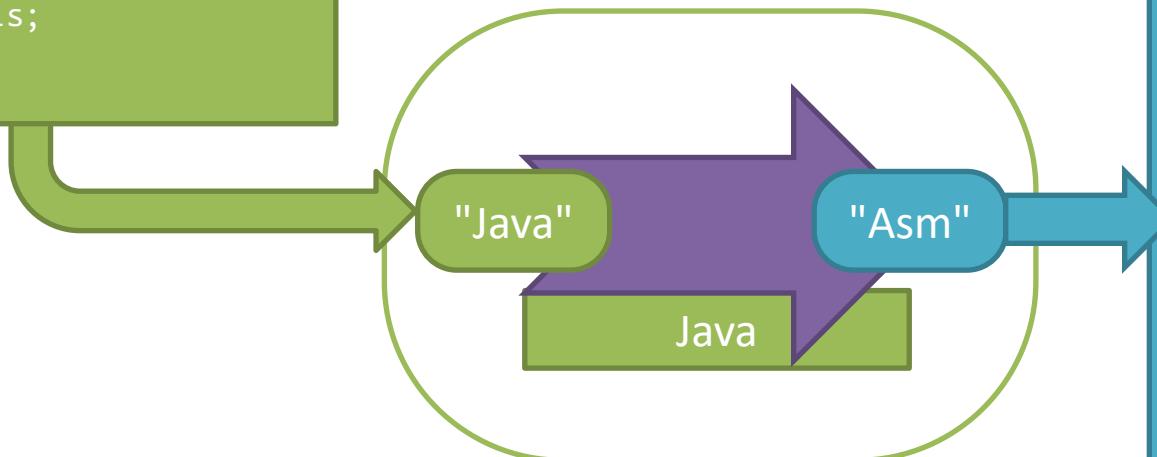


Bootstrap Process

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;
 - E.g., 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

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

Stack

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()

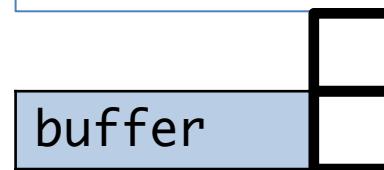
Abstract Stack Machine

Workspace

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

Stack

```
-;
// do some more stuff
```



Allocate space for buffer on the stack.

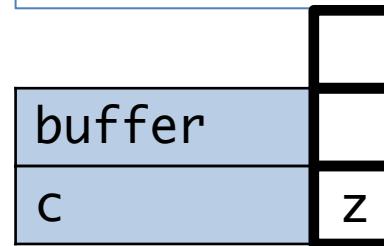
Abstract Stack Machine

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.

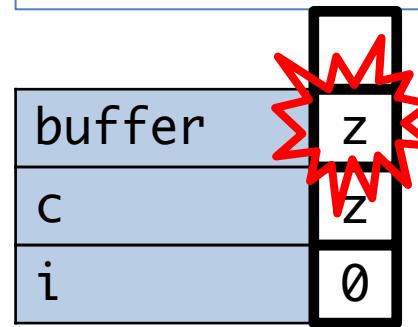
Abstract Stack Machine

Workspace

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

Stack

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



Copy (contents of) c to buffer[0]

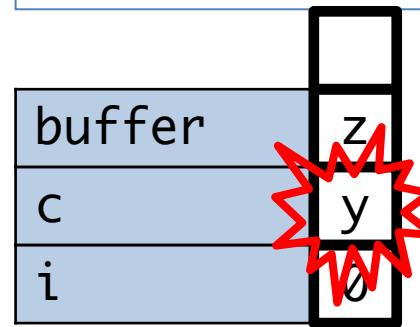
Abstract Stack Machine

Workspace

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

Stack

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



Read next character ... 'y'

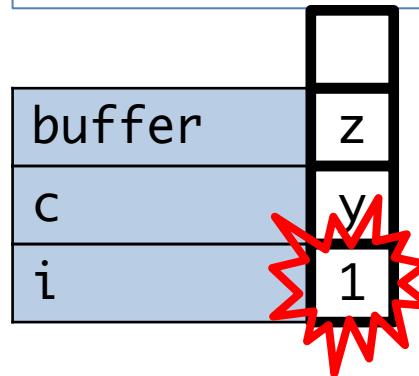
Abstract Stack Machine

Workspace

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

Stack

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



Increment i

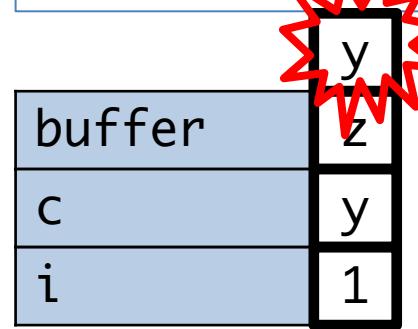
Abstract Stack Machine

Workspace

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

Stack

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



Copy (contents of) c to buffer[1]

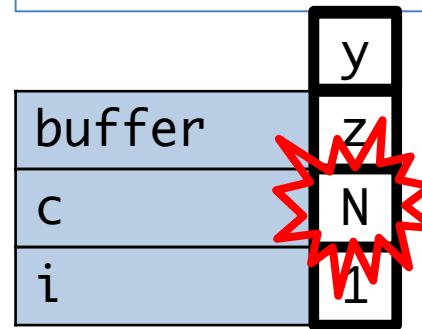
Abstract Stack Machine

Workspace

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

Stack

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



Read next character ... 'N'

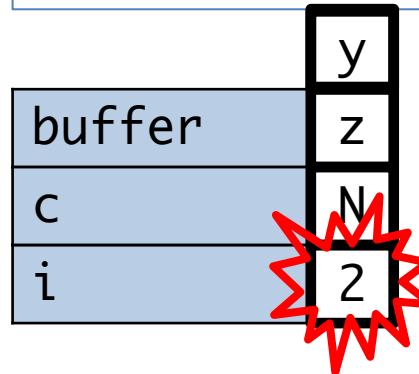
Abstract Stack Machine

Workspace

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

Stack

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



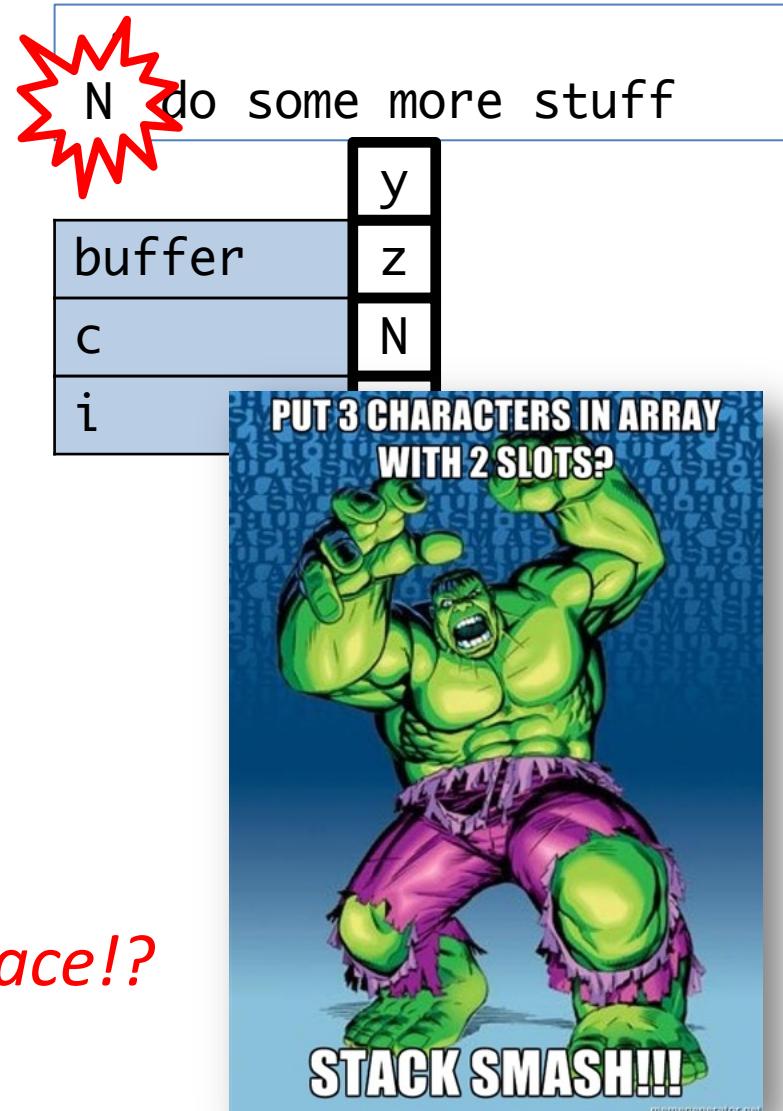
Increment i

Abstract Stack Machine

Workspace

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

Stack



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

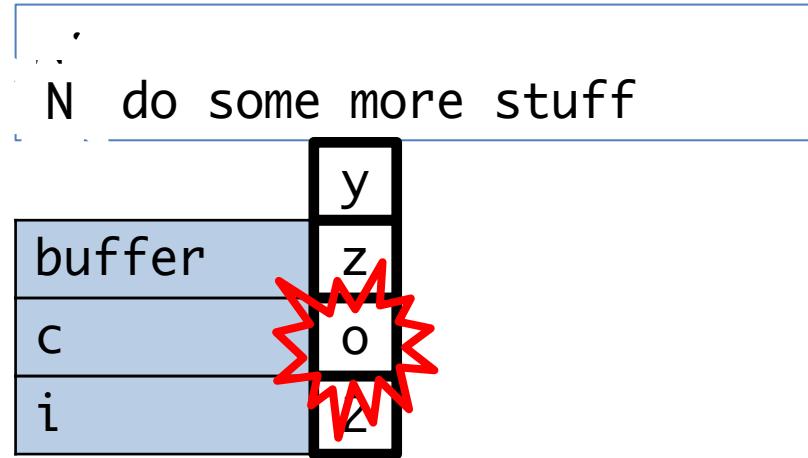
Overwrites the saved workspace!?

Abstract Stack Machine

Workspace

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

Stack



Keep going... read 'o'...

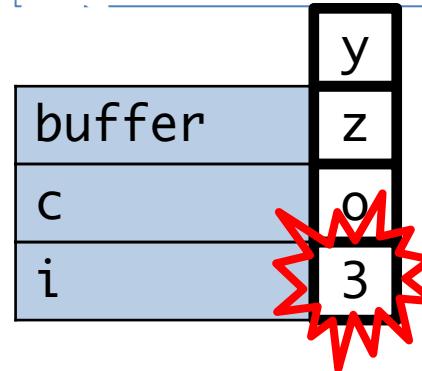
Abstract Stack Machine

Workspace

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

Stack

⋮ do some more stuff



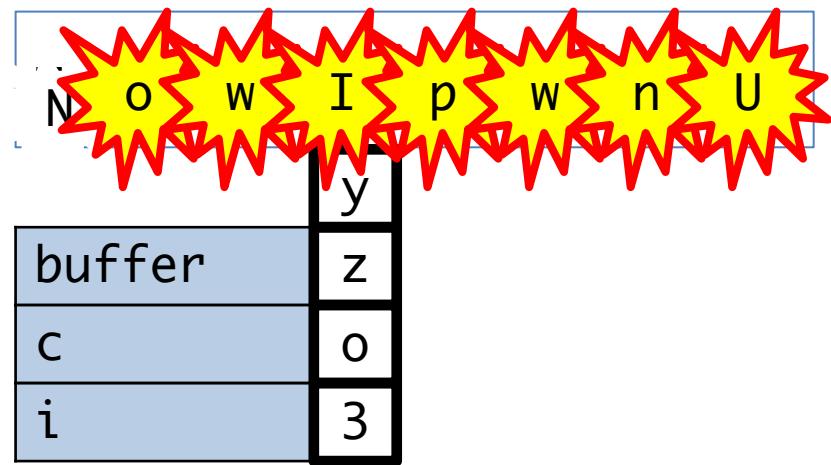
Keep going... read 'o'...increment i...

Abstract Stack Machine

Workspace

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

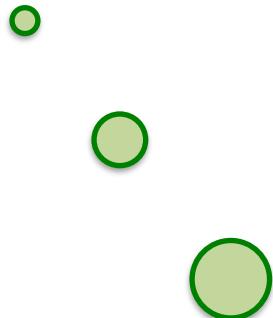
Stack



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

Abstract Stack Machine

Workspace



Later...



Stack

Now I pwn U!!!!

buffer	z
c	0
i	3

Abstract Stack Machine

Workspace

Now I pwn U!!!!

Stack



The stack smashing attack successfully placed *arbitrary code* into the program.

Is this a big deal?



Yep!

Board ▾

Record Request CVE IDs

Record Format JSON are

in 2023.

Search Results

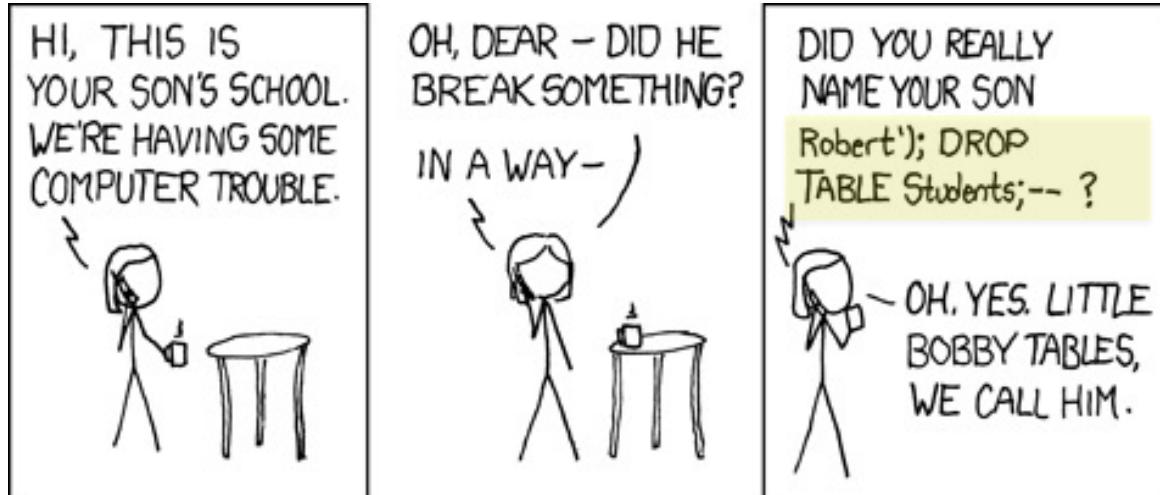
There are 13896 CVE Records that match your search.

Name
CVE-2023-25602
A stack-based buffer overflow in Fortinet FortiWeb versions 6.0.7 and earlier, FortiWeb v6.0.7, allows an attacker to execute unauthorized code or commands via specially crafted command arguments.
CVE-2023-25563
GSS-NTLMSSP is a mechglue plugin for the GSSAPI library that implements NTLM authentication. Prior to 1.2.0, it triggers a denial of service. A 32-bit integer overflow condition can lead to incorrect checks of consistency when accepting a single input buffer of 4GB in length. This could theoretically happen. This vulnerability can be triggered by sending tokens greater than 4GB in length. This can lead to a large, up to 65KB, out-of-bounds read which 1.2.0 contains a patch for the out-of-bounds reads.
CVE-2023-25235
Tenda AC500 V2.0.1.9(1307) is vulnerable to Buffer Overflow in function formOneSsidCfgSet via parameters
CVE-2023-25234
Tenda AC500 V2.0.1.9(1307) is vulnerable to Buffer Overflow in function fromAddressNat via parameters

16,135 buffer overflow attacks listed as of 11/21/23

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; --")"
```



Consequence 3: Undecidability



Undecidability Theorem

Theorem: It is **impossible** to write a method

boolean halts (String prog)

such that, for any valid Java program P , represented as a string p_P ,

halts(p_P)

returns true exactly when the program P halts (and returns 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 an infinite amount of memory and arbitrarily large integers, it is *unknown* whether this program halts for all $n \geq 1$!

Collatz Conjecture (proposed in 1937, still open!)

Consider this Program (let's call it 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 main() {  
        String p_Q = ???; // string representing Q  
        if (HaltDetector.halts(p_Q)) {  
            while (true) {} // infinite loop!  
        }  
    }  
}
```

What happens when we run Q?

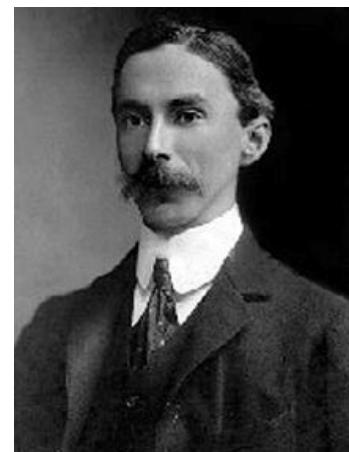
```
public static void Q() {  
    String p_Q = ???; // string representing Q  
    if (HaltDetector.halts(p_Q)) {  
        while (true) {} // infinite loop!  
    }  
}
```

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

if `HaltDetector.halts(p_Q)` \Rightarrow false then Q halts

Contradiction!

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



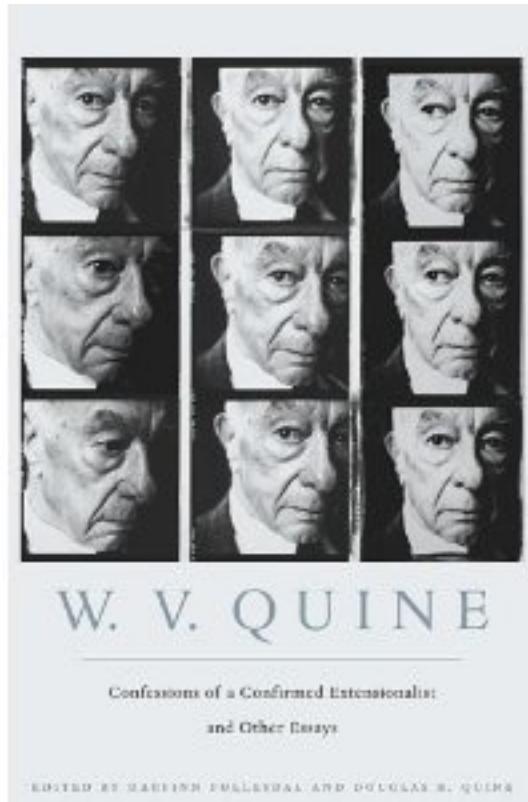
Bertrand Russell, 1901



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

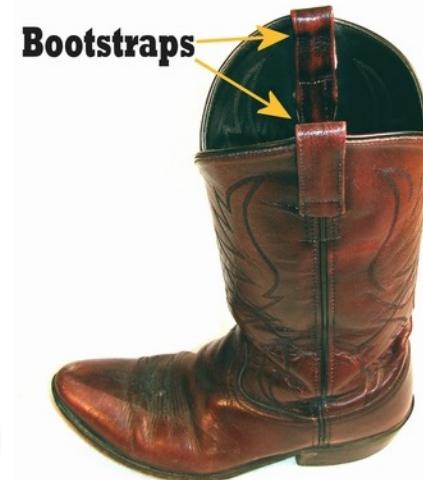


Profound Consequences

- The “halting problem” is *undecidable*
 - *I.e., 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 3410: Compilers and interpreters
- Malware
 - CIS 3310: Intro to Networks and Security
- Undecidability
 - CIS 2620: Automata, Computability and Complexity



Recommended Reading

