# Corrode

## Automatic C-to-Rust translation

Jamey Sharp

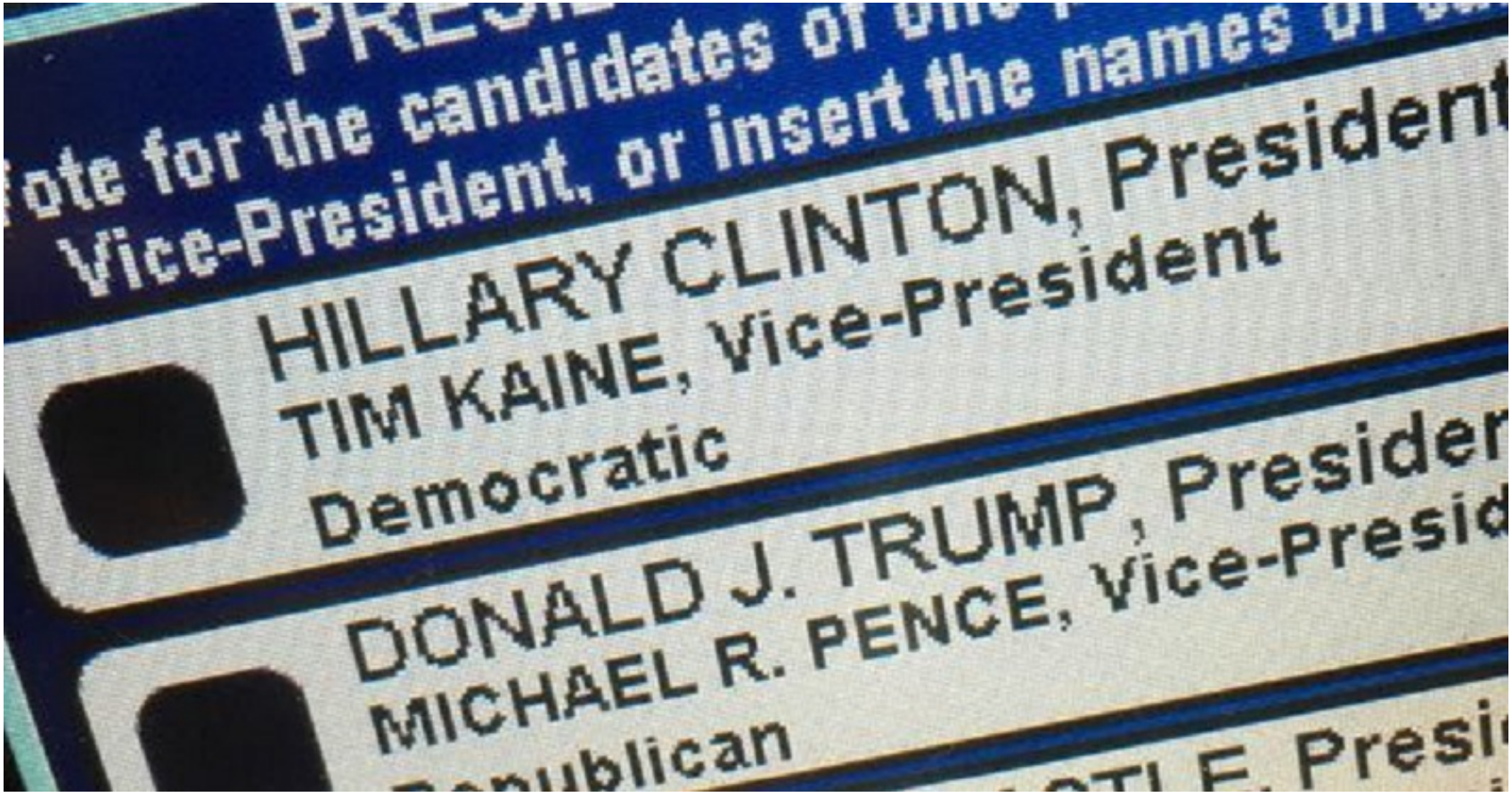
2017-02-04

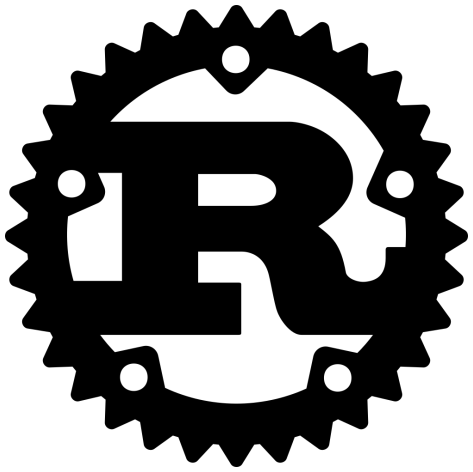
# C to Rust

# cybersecurity


scada_b (22) by flickr/greenmambagreenmamba

# cybersecurity



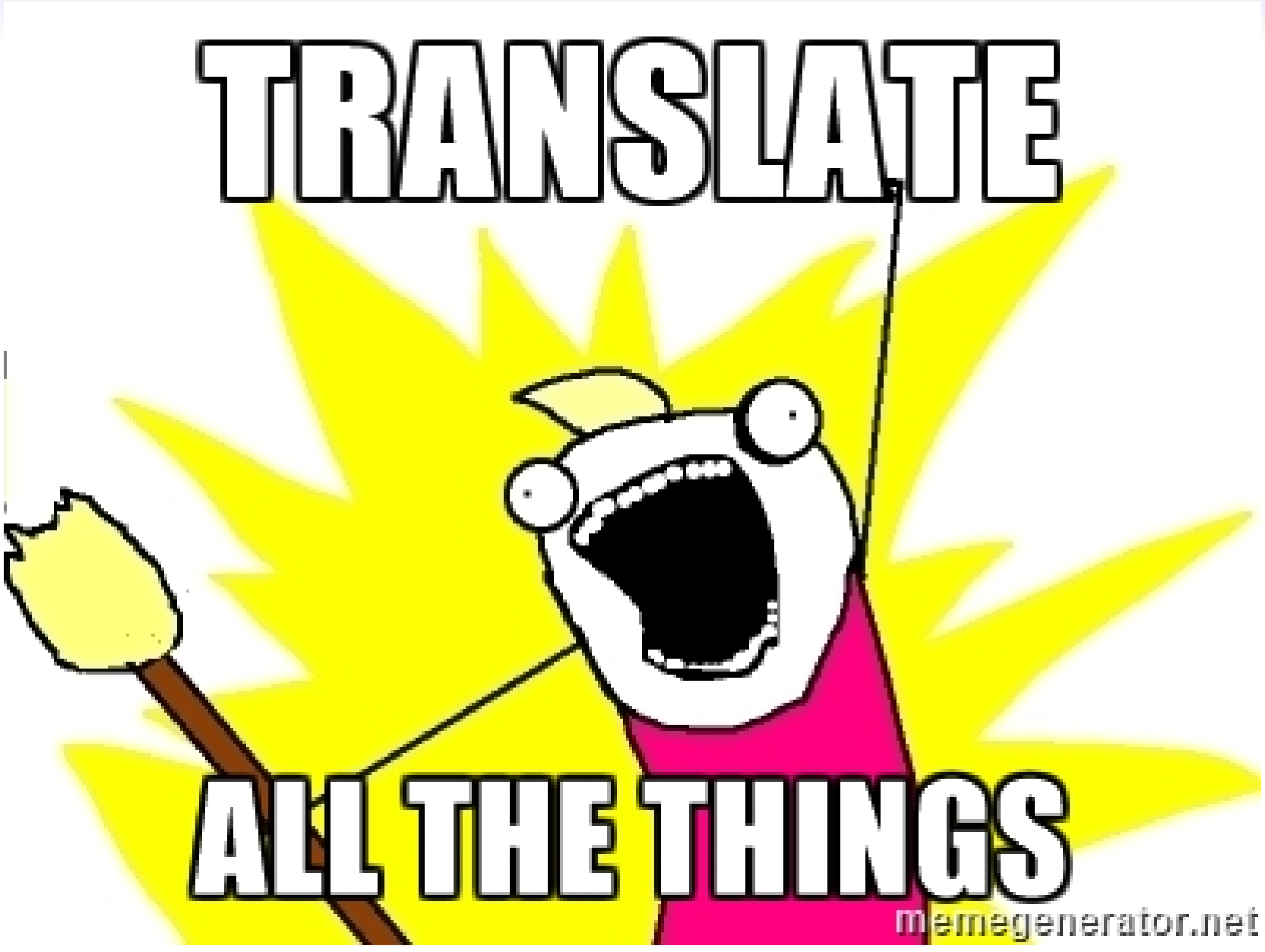


# noteworthy efforts

- Firefox (but that hardly counts!)
- librsvg 2.41.0 release notes (January 3rd):

  > *The big news is that parts of librsvg are now implemented in the Rust programming language, instead of C. The public API remains identical. Rust should provide us with memory safety and nicer built-in abstractions for the code, as well as an easier way to do unit tests.*

- Remacs (Emacs incrementally ported to Rust)
- rusl (musl C library incrementally ported to Rust)
- coreutils re-write


TRANSLATE
ALL THE THINGS
memegenerator.net

# this sounds like a terrible plan

- translate from C to Rust...
- rewrite thousands (or millions) of lines of C, by hand?
- without introducing new bugs?
- tedious at best
- certainly error-prone

# this sounds like a terrible plan

- translate from C to Rust. . .
- rewrite thousands (or millions) of lines of C, by hand?
- without introducing new bugs?
- tedious at best
- certainly error-prone

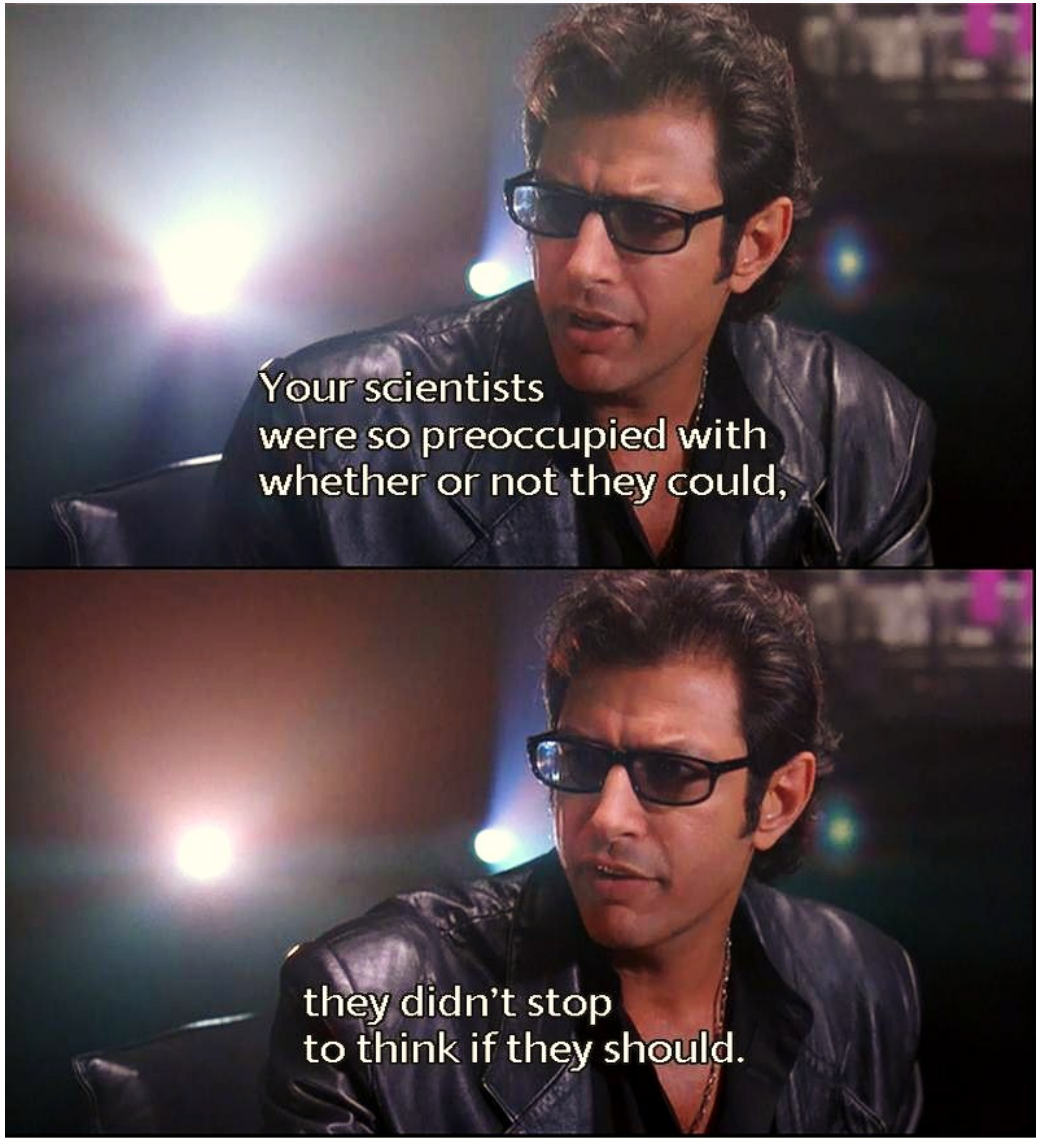
- so don't do it all by hand

# automation: introducing Corrode

- preserve safety and correctness
  - output is *unsafe* Rust
  - *unsafe* means it might crash
  - exactly as safe as the input
  - choose "unsurprising" definitions for UB
- preserve maintainability
  - preserves variable names and code structure
- preserve ABI
  - enables Rust as drop-in replacement for C

# approach

- easier to refactor safe Rust from unsafe Rust than from C!
- incremental manual improvement


Your scientists
were so preoccupied with
whether or not they could,
they didn't stop
to think if they should.

# should we?

- for educational purposes? absolutely!
- otherwise:

1. can we prove the Rust is equivalent to the C?
2. do Rust's advantages help with bugs this project faces?
3. is the project community willing to accept Rust patches?

# case study

# wanted:

- unmaintained open source projects
- written in C
- with security implications
- that are still in use

## wanted:

- unmaintained open source projects
- written in C
- with security implications
- that are still in use

# Concurrent Versions System!

# . . . CVS? . . . why?

- lots of old source code is only available via CVS
  - 6% of Debian users still have CVS installed! [popcon]
- CVS is usually exposed via unauthenticated, unencrypted network protocol
  - remote code execution vulnerabilities $\Rightarrow$ easy exploits
- yet the last release was in 2008
  - who'd want to maintain CVS when "everyone" is using git now?

# . . . CVS? . . . why?

- not a trivial codebase
  - 52k SLOC plus 35k SLOC of libc-portability glue
- relying on many corners of C
  - original C implementation dates to 1989
  - still has some K&R-style functions
- if we can translate this, we can translate most anything

# . . . CVS? . . . why?

- not a trivial codebase
  - 52k SLOC plus 35k SLOC of libc-portability glue
- relying on many corners of C
  - original C implementation dates to 1989
  - still has some K&R-style functions
- if we can translate this, we can translate most anything
- bonus: CVS has an extensive test suite

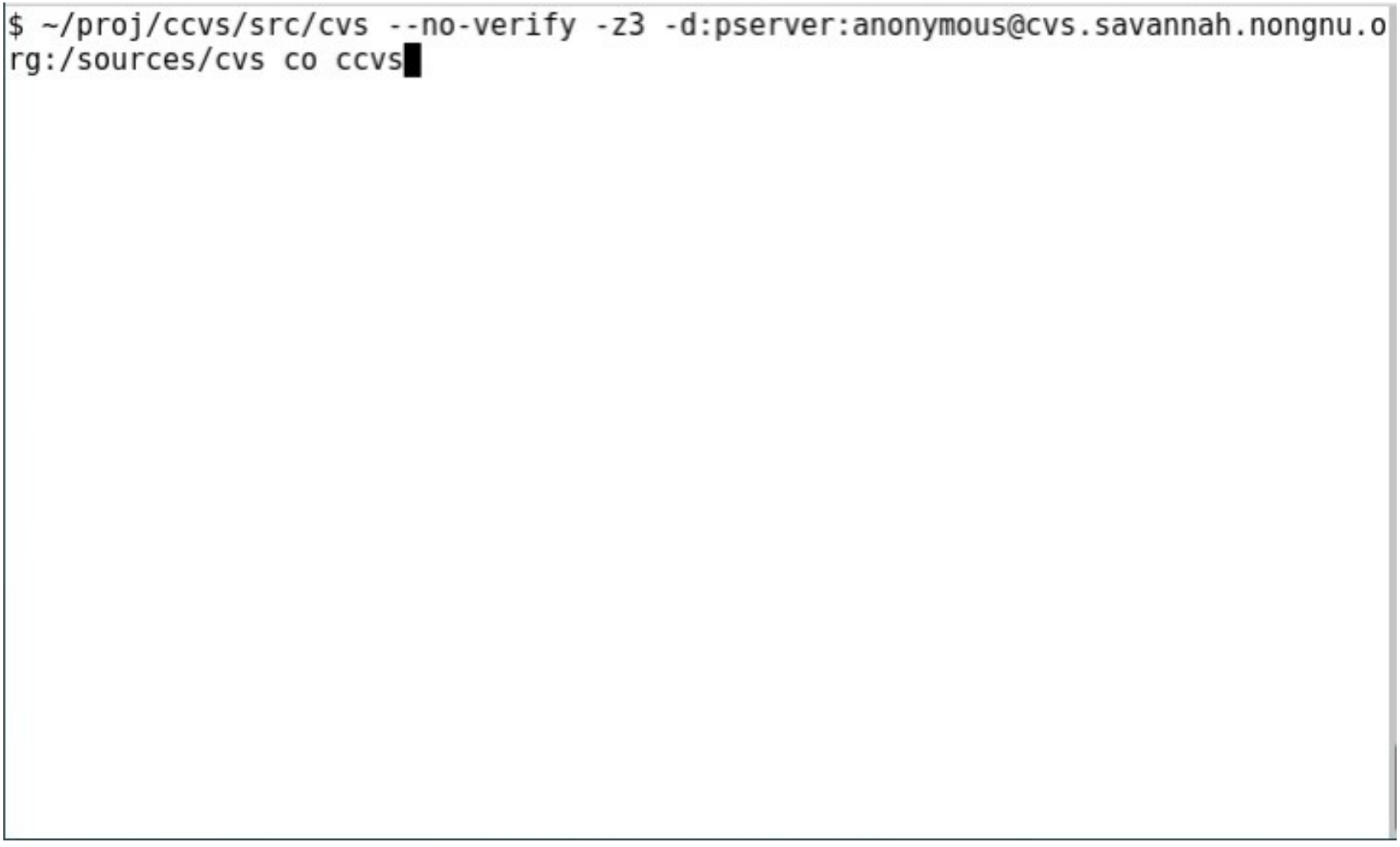
# translation progress

- `https://github.com/jameysharp/cvs-rs`
- 6.4% of `src/*.c` SLOC (3.3k SLOC)
- 10 out of 68 source files

# Corrode is still WIP

- control flow is hard
  - Rust doesn't have C-style `goto` or `switch`
- several corners of C are not yet translated
- some C features don't yet exist in Rust (bit-fields, VLAs)
- currently uses pre-processed source
  - loses comments, expands macros

# demo!


$ ~/proj/ccvs/src/cvs --no-verify -z3 -d:pserver:anonymous@cvs.savannah.nongnu.o
rg:/sources/cvs co ccvs

# test suite passes!

# future work

# non-idiomatic output

*C:*

```
for(unsigned i = 0; i < 10; ++i)
    ...
```

# non-idiomatic output

*C:*

```
for(unsigned i = 0; i < 10; ++i)
    ...
```

*Idiomatic Rust:*

```
for i in 0..10 { ... }
```

# non-idiomatic output

*C:*

```
for(unsigned i = 0; i < 10; ++i)
    ...
```

*Idiomatic Rust:*

```
for i in 0..10 { ... }
```

*Corrode-generated Rust:*

```
let mut i : u32 = 0i32 as (u32);
while i < 10i32 as (u32) {
    ...
    i = i.wrapping_add(1 as (u32));
}
```

# replace raw pointers

- identify safety conditions where safe Rust “borrow” types can replace raw pointer types
- (this is tricky!)

# existing tools

- Rust developers already use style-improving tools:
  - clippy
  - rustfix
- improving these tools benefits all Rust developers, not just Corrode users

# conclusions

# conclusions

- Corrode makes incremental migration from C to Rust feasible
- soon Corrode will support most existing C source!
- but there's plenty left to do to make the resulting Rust better than the original C

# acknowledgements


- Don Marti and the Mozilla Open Innovation Team

Table 1: Contributors:

| | |
|---|---|
| Alec Theriault | Nathan Bergey |
| Bruce Mitchener | Vickenty Fesunov |
| Fabian Zaiser | Getty Ritter |
| Jeff Waugh | Jeremie Jost |
| Nabil Hassein | Amin Bandali |
| Robert Grosse | Sean Jensen-Grey |
| Taylor Cramer | Tommi Komulainen |

# questions?

- `https://github.com/jameysharp/corrode`
- `http://jamey.thesharps.us/search/label/corrode`
- Twitter: @jamey_sharp

# extras

# how do we know Corrode's output is right?

- easy options:
  - Csmith and C-Reduce
  - QuickCheck (randomized property testing) for control-flow analysis
  - translate existing code-bases that have test suites
- hard options:
  - Galois "Software Analysis Workbench": prove LLVM bitcode equivalent
  - CompCert-style formal verification of Corrode

# correct translation of C control flow

```
again:
...
if (pre_stbuf.st_mtime == post_stbuf.st_mtime || !*messagep)
{
    for (;;)
    {
        ...
        if (*line == 'e' || *line == 'E')
            goto again;
        if (*line == '!')
        {
            reuse_log_message = 1;
            break;
        }
        ...
    }
}
```

# array size expressions

```
static struct option long_options[] =
{
    {"execute", 0, NULL, OPT_EXECUTE},
    {"no-execute", 0, NULL, OPT_NOEXECUTE},
    {0, 0, NULL, OPT_NONE}
};
```

# use of uninitialized variables

```
struct stat sb;
if (lstat (file, &sb) < 0)
    ...
```

# use of uninitialized variables

```
struct stat sb;
if (lstat (file, &sb) < 0)
    ...

struct utimbuf t;
memset (&t, 0, sizeof (t));
```

# use of uninitialized variables

```
struct log_data_and_rcs
{
    struct log_data *log_data;
    struct revlist *revlist;
    RCSNode *rcs;
} log_data_and_rcs;

...

log_data_and_rcs.log_data = log_data;
log_data_and_rcs.revlist = revlist;
log_data_and_rcs.rcs = rcsfile;
selrev = walklist (rcsfile->versions, log_count_print,
                   &log_data_and_rcs);
```

# nullable function pointers

C version:

```
if (p->delproc != NULL)
    p->delproc (p);
p->delproc = NULL;
```

# nullable function pointers

C version:

```
if (p->delproc != NULL)
    p->delproc (p);
p->delproc = NULL;
```

Rust version:

```
if let Some(delproc) = (*p).delproc {
    delproc(p);
}
(*p).delproc = None;
```