Rust out your C (w/FP Bent)

Carol (Nichols || Goulding)
@carols10cents

photo by flickr user celinet, cropped https://flic.kr/p/8K8m9

Agenda

1 **Caveats**

2 **Background**

3 **Techniques**

4 **Benefits?**

DO NOT

Bad reasons to rewrite your C in Rust

- Cool kids
- I feel like it
- I’m bored
- Job security
- Carol said

Good reasons to rewrite your C in Rust

- Performance
- Safety
- Lower maintenance costs
- Expand # of maintainers
- For fun, not work

I FIGHT FOR THE USERS

Any time you rewrite, code will get better.

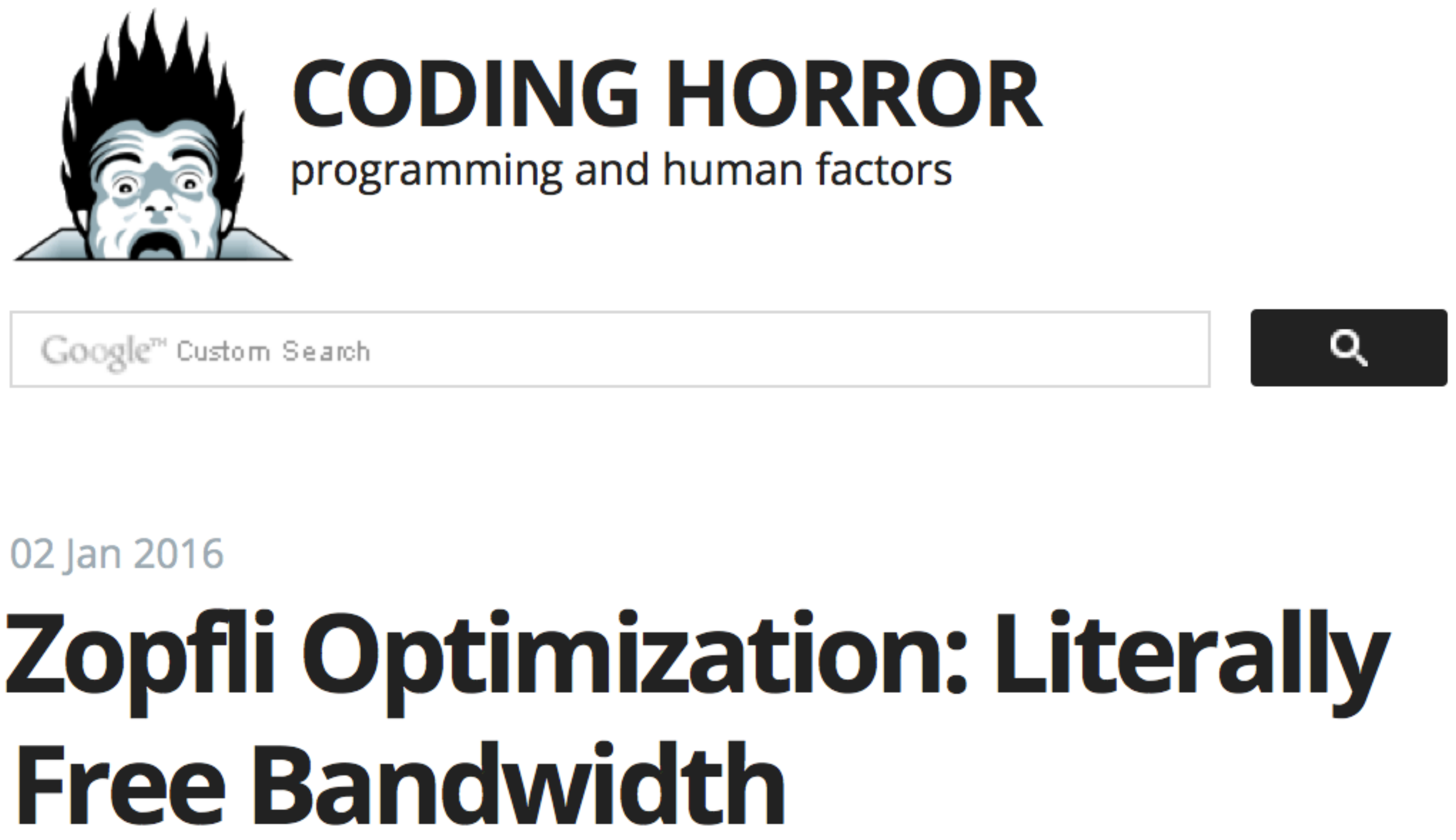
Things I knew before

- Rust
- Legacy code
- Testing

Things I DID NOT know before

- C
- FFI
- zopfli

Background: zopfli



CODING HORROR

programming and human factors

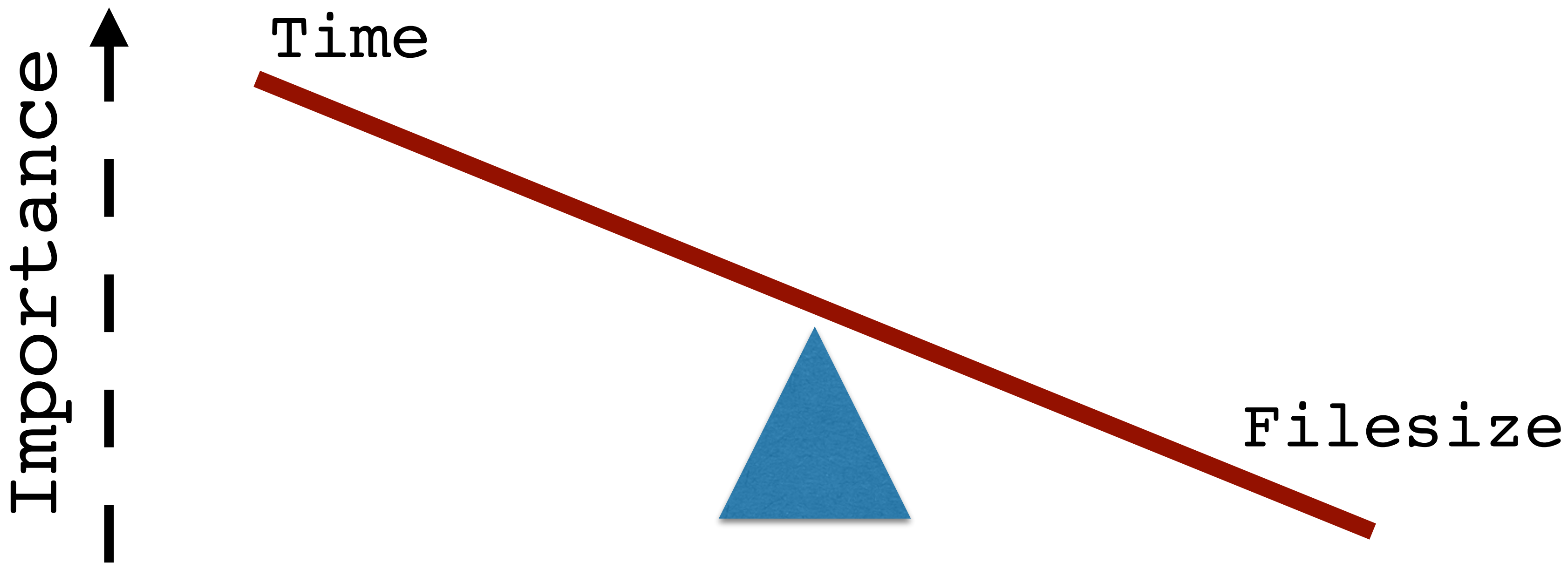
02 Jan 2016

Zopfli Optimization: Literally Free Bandwidth

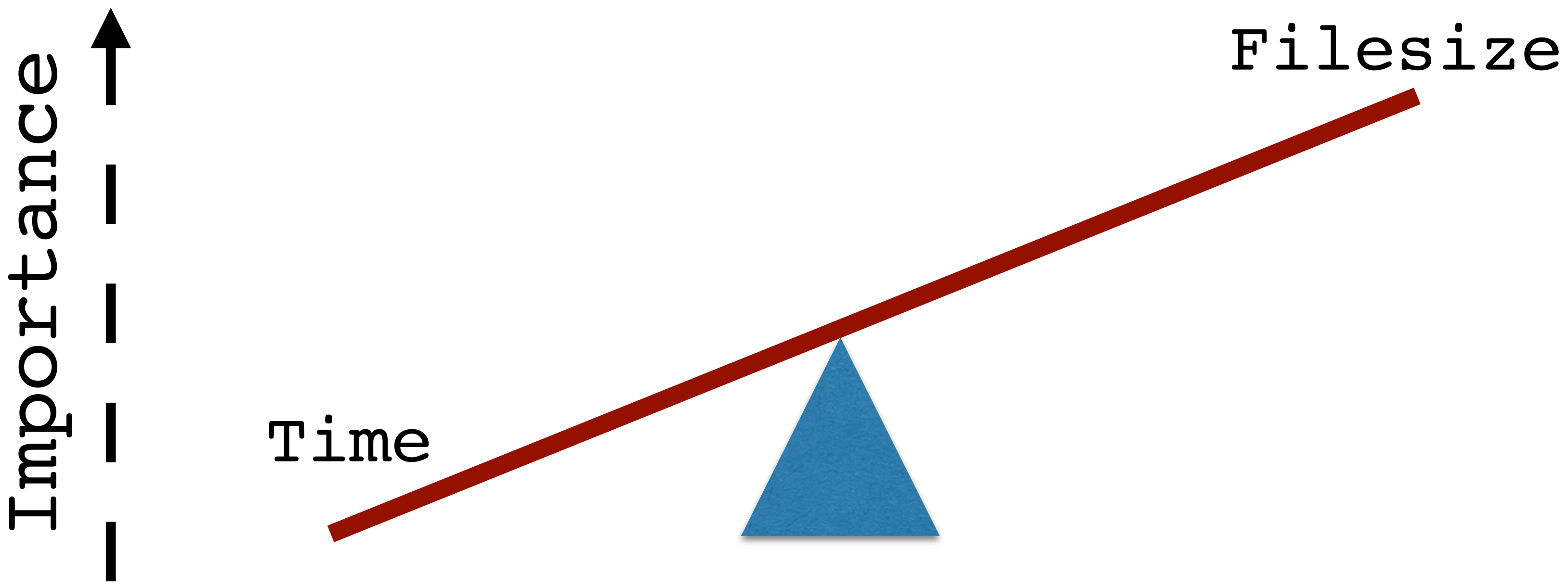
https://blog.codinghorror.com/zopfli-optimization-literally-free-bandwidth/

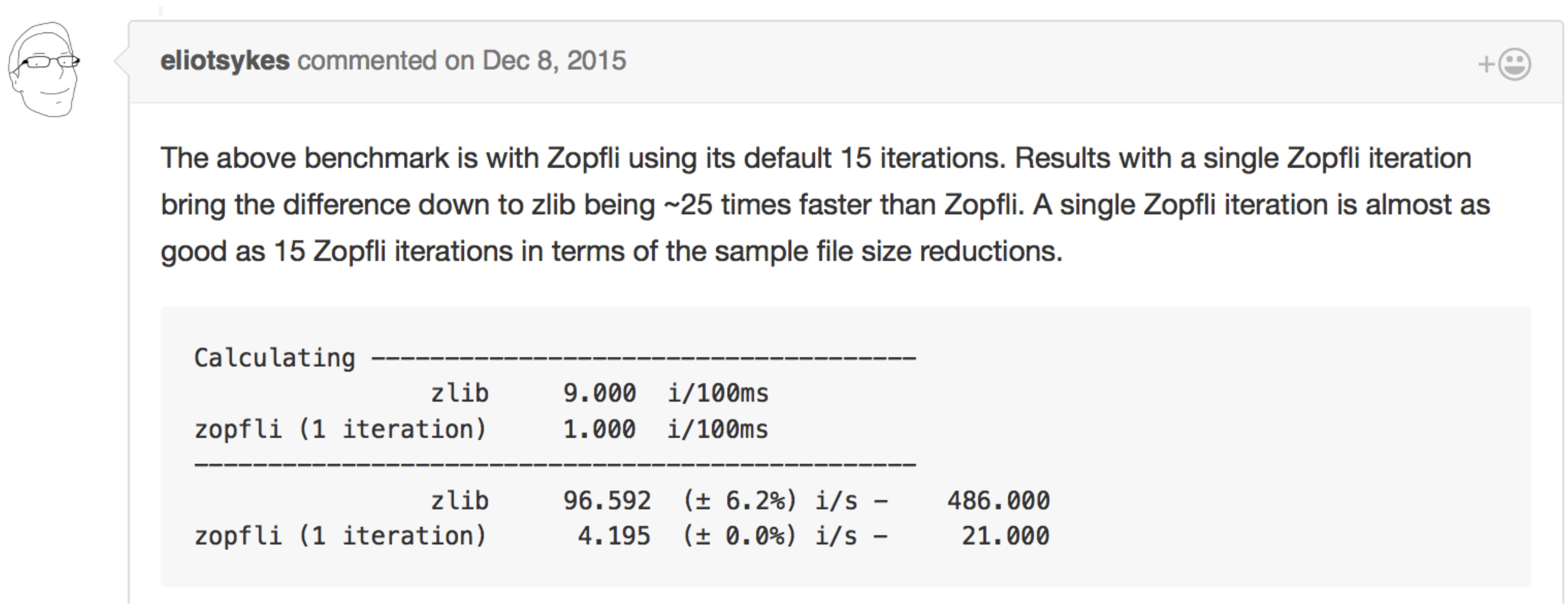
This is a DEFLATE file, I know this!

gzip



zopfli



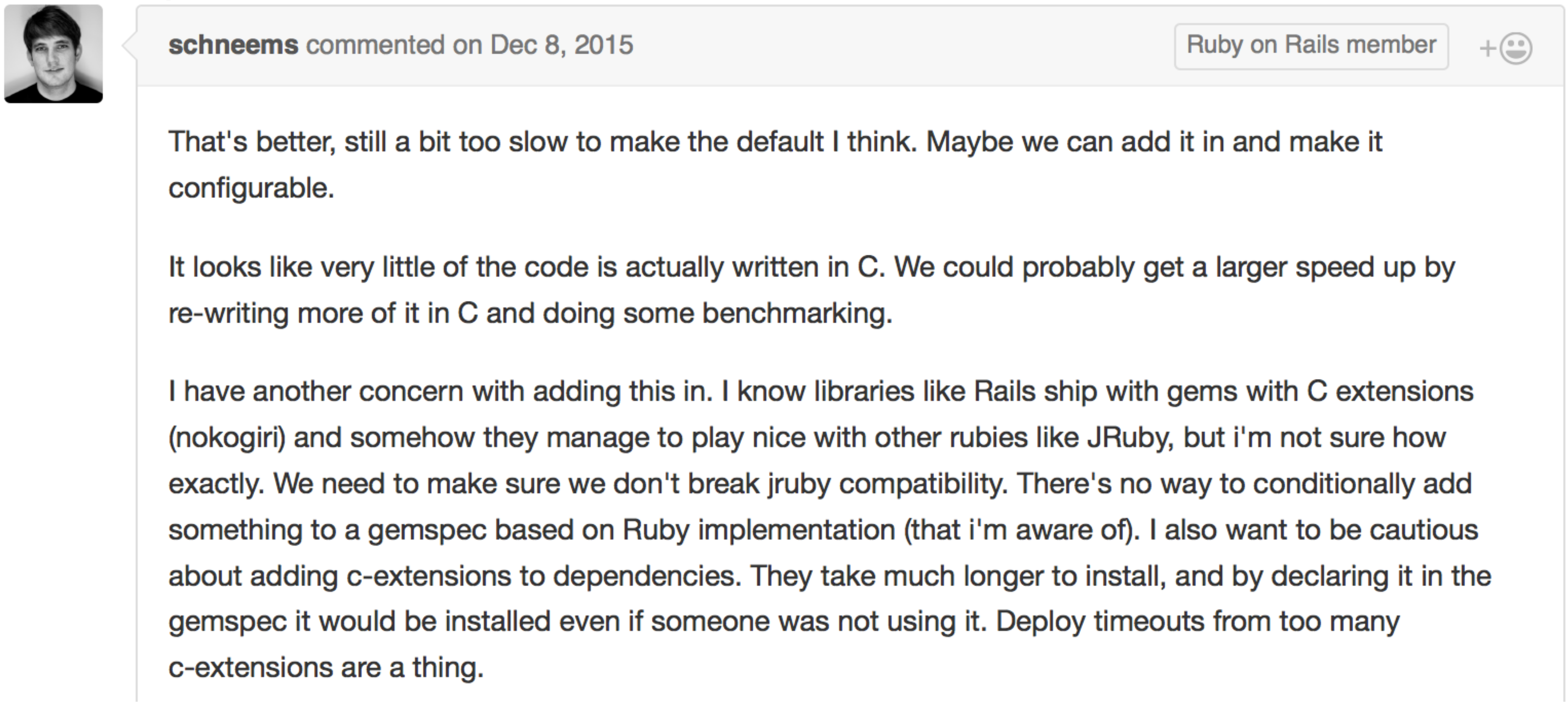


eliotsykes commented on Dec 8, 2015

The above benchmark is with Zopfli using its default 15 iterations. Results with a single Zopfli iteration bring the difference down to zlib being ~25 times faster than Zopfli. A single Zopfli iteration is almost as good as 15 Zopfli iterations in terms of the sample file size reductions.

```
Calculating -------------------------------------
                zlib     9.000  i/100ms
zopfli (1 iteration)     1.000  i/100ms
-------------------------------------------------
                zlib     96.592  (± 6.2%) i/s -    486.000
zopfli (1 iteration)      4.195  (± 0.0%) i/s -     21.000
```

https://github.com/rails/sprockets/pull/193


schneems commented on Dec 8, 2015

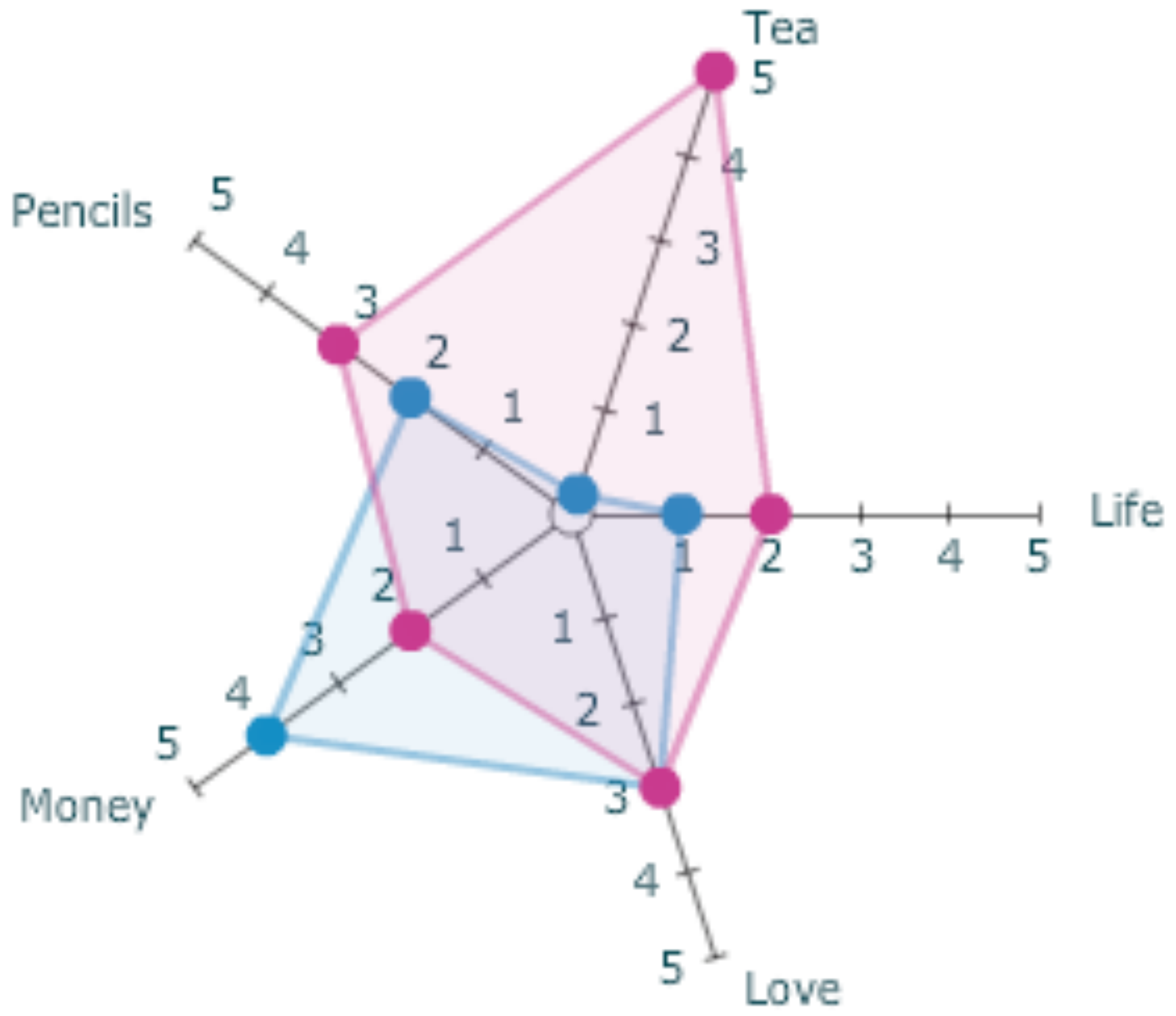
Ruby on Rails member

That's better, still a bit too slow to make the default I think. Maybe we can add it in and make it configurable.

It looks like very little of the code is actually written in C. We could probably get a larger speed up by re-writing more of it in C and doing some benchmarking.

I have another concern with adding this in. I know libraries like Rails ship with gems with C extensions (nokogiri) and somehow they manage to play nice with other rubies like JRuby, but i'm not sure how exactly. We need to make sure we don't break jruby compatibility. There's no way to conditionally add something to a gemspec based on Ruby implementation (that i'm aware of). I also want to be cautious about adding c-extensions to dependencies. They take much longer to install, and by declaring it in the gemspec it would be installed even if someone was not using it. Deploy timeouts from too many c-extensions are a thing.

https://github.com/rails/sprockets/pull/193



example radar chart demoing MIT Licensed flex component from
http://www.boost.co.nz/blog/2009/05/flex-radar-chart-component/

Techniques

IF IT AIN'T BAROQUE

DON'T FIX IT

https://commons.wikimedia.org/wiki/File:Johann_Sebastian_Bach.jpg

Golden Master Tests

Remove a function from C

```
size_t CalculateTreeSize(const unsigned* ll_lengths, const unsigned* d_lengths) {
  size_t result = 0;
  int i;

  for(i = 0; i < 8; i++) {
    size_t size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                                     i & 1, i & 2, i & 4);
    if (result == 0 || size < result) result = size;
  }

  return result;
}
```

Remove a function from C

```
extern size_t CalculateTreeSize(const unsigned* ll_lengths, const unsigned*
d_lengths);
```

Add a function to Rust

```
size_t CalculateTreeSize(const unsigned* ll_lengths, const unsigned* d_lengths) {
  size_t result = 0;
  int i;

  for(i = 0; i < 8; i++) {
    size_t size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                                     i & 1, i & 2, i & 4);
    if (result == 0 || size < result) result = size;
  }

  return result;
}
```

Add a function to Rust

```
#[no_mangle]
#[allow(non_snake_case)]
size_t CalculateTreeSize(const unsigned* ll_lengths, const unsigned* d_lengths) {
  size_t result = 0;
  int i;

  for(i = 0; i < 8; i++) {
    size_t size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                                     i & 1, i & 2, i & 4);
    if (result == 0 || size < result) result = size;
  }

  return result;
}
```

Add a function to Rust

```
#[no_mangle]
#[allow(non_snake_case)]
pub extern fn CalculateTreeSize(const unsigned* ll_lengths, const unsigned*
d_lengths) -> size_t {
  size_t result = 0;
  int i;

  for(i = 0; i < 8; i++) {
    size_t size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                             i & 1, i & 2, i & 4);
    if (result == 0 || size < result) result = size;
  }

  return result;
}
```

Add a function to Rust

```
#[no_mangle]
#[allow(non_snake_case)]
pub extern fn CalculateTreeSize(ll_lengths: const unsigned*, d_lengths: const
unsigned*) -> size_t {
  size_t result = 0;
  int i;

  for(i = 0; i < 8; i++) {
    size_t size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                             i & 1, i & 2, i & 4);
    if (result == 0 || size < result) result = size;
  }

  return result;
}
```

Add a function to Rust

```
use libc::{size_t, c_uint};

#[no_mangle]
#[allow(non_snake_case)]
pub extern fn CalculateTreeSize(ll_lengths: *const c_uint, d_lengths: *const
c_uint) -> size_t {
  size_t result = 0;
  int i;

  for(i = 0; i < 8; i++) {
    size_t size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                                     i & 1, i & 2, i & 4);
    if (result == 0 || size < result) result = size;
  }

  return result;
}
```

Add a function to Rust

```
use libc::{size_t, c_uint};

#[no_mangle]
#[allow(non_snake_case)]
pub extern fn CalculateTreeSize(ll_lengths: *const c_uint, d_lengths: *const
c_uint) -> size_t {
  let mut result = 0;
  int i;

  for(i = 0; i < 8; i++) {
    let size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                             i & 1, i & 2, i & 4);
    if (result == 0 || size < result) result = size;
  }

  return result;
}
```

Add a function to Rust

```
use libc::{size_t, c_uint};

#[no_mangle]
#[allow(non_snake_case)]
pub extern fn CalculateTreeSize(ll_lengths: *const c_uint, d_lengths: *const
c_uint) -> size_t {
  let mut result = 0;

  for i in 0..8 {
    let size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                             i & 1, i & 2, i & 4);
    if (result == 0 || size < result) result = size;
  }

  return result;
}
```

Add a function to Rust

```
use libc::{size_t, c_uint};

#[no_mangle]
#[allow(non_snake_case)]
pub extern fn CalculateTreeSize(ll_lengths: *const c_uint, d_lengths: *const
c_uint) -> size_t {
  let mut result = 0;

  for i in 0..8 {
    let size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                             i & 1, i & 2, i & 4);
    if result == 0 || size < result {
      result = size;
    }
  }

  return result;
}
```

Add a function to Rust

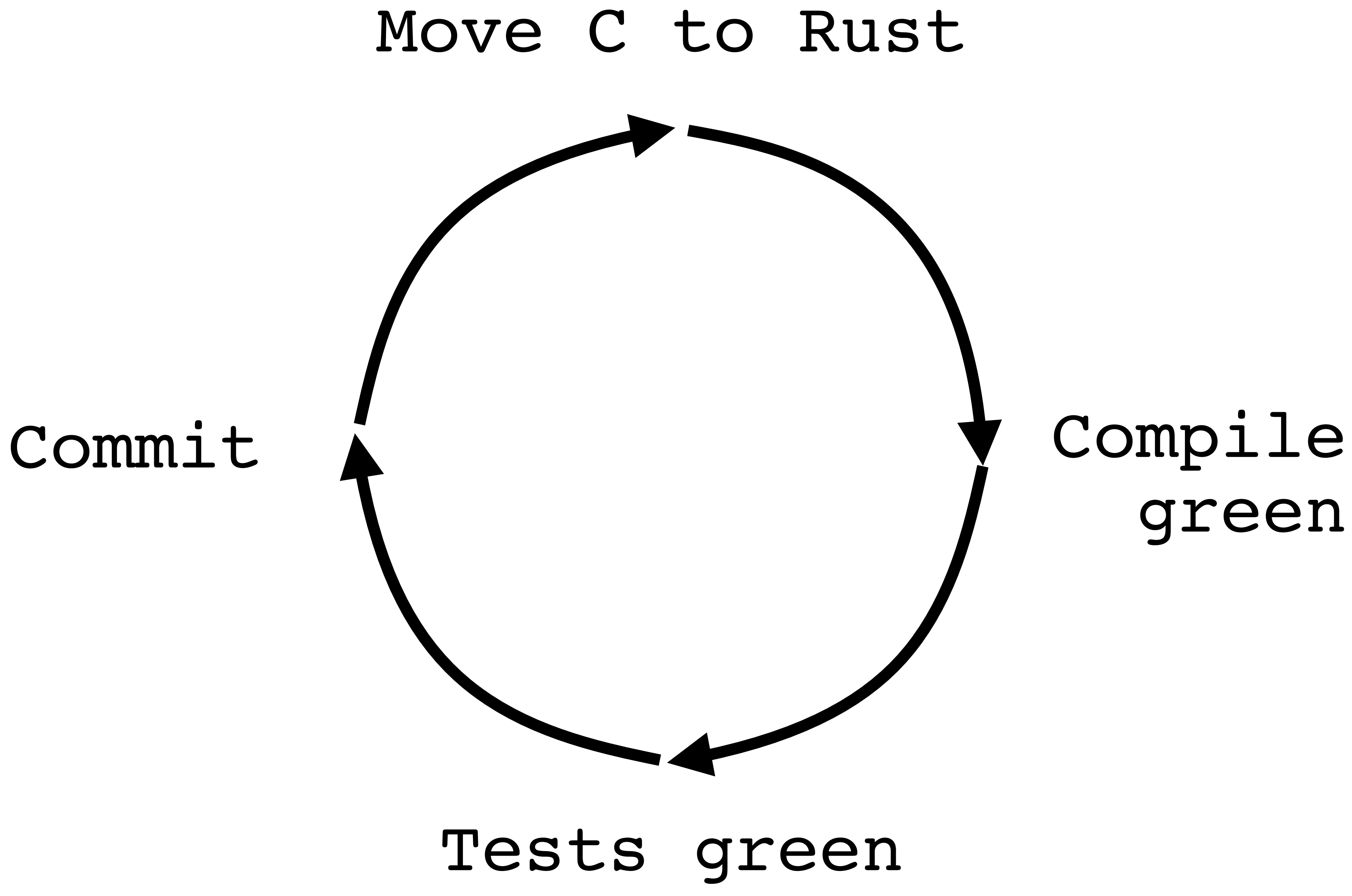
```
use libc::{size_t, c_uint};

#[no_mangle]
#[allow(non_snake_case)]
pub extern fn CalculateTreeSize(ll_lengths: *const c_uint, d_lengths: *const
c_uint) -> size_t {
  let mut result = 0;

  for i in 0..8 {
    let size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                             i & 1, i & 2, i & 4);
    if result == 0 || size < result {
      result = size;
    }
  }

  result
}
```

Incremental


Move C to Rust
Compile green
Tests green
Commit

“Are you done yet?”

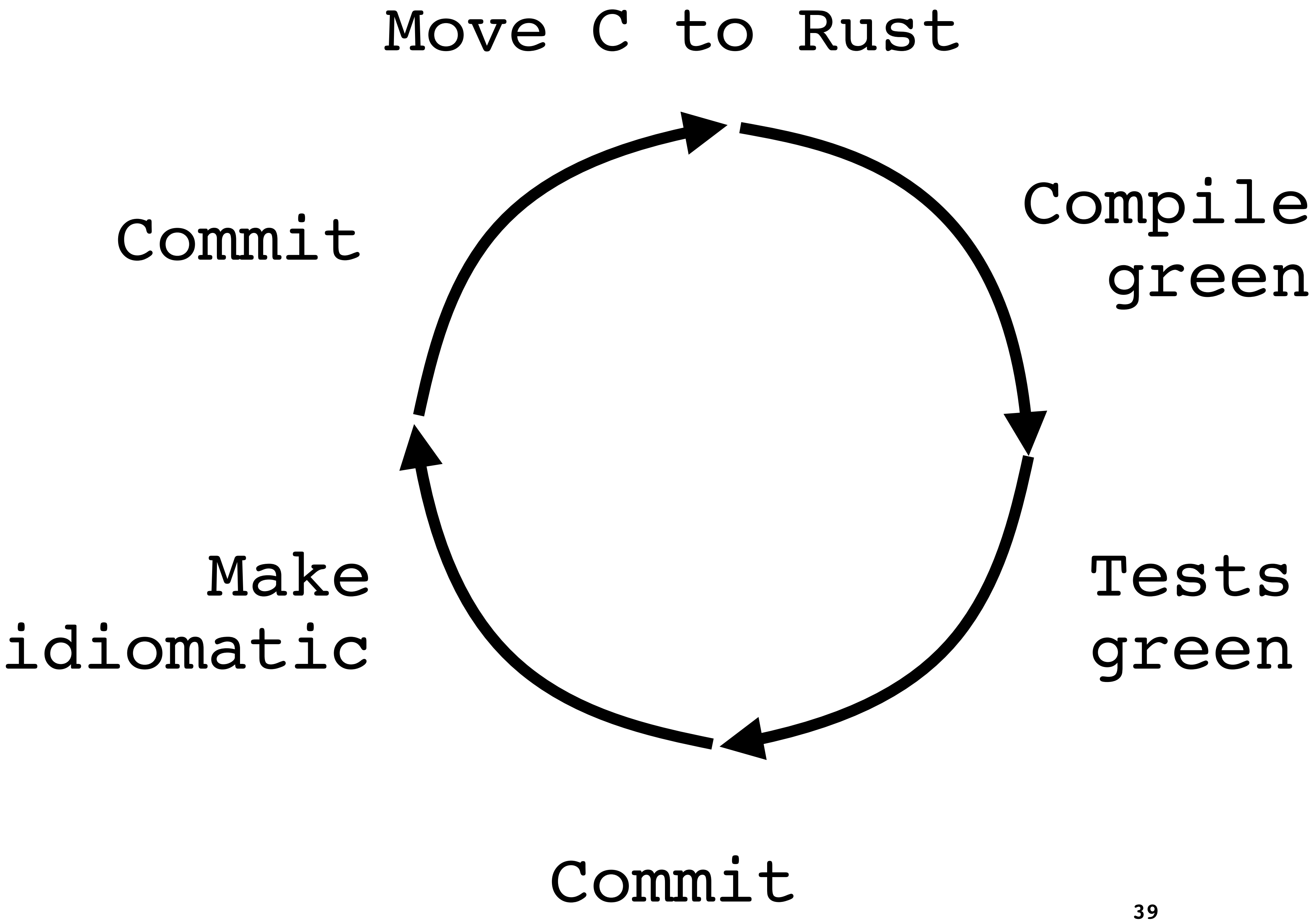
BRING ME
SCHRÖDINGER'S HEAD

What if it doesn’t pass the tests?

git checkout! take a smaller step.

smaller

- Extract functions
- Make the C more like Rust first
- Don’t make the Rust idiomatic at all, even when it seems easy


Move C to Rust
Compile green
Tests green
Commit
Make idiomatic
Commit

Idiomaticize

```
use libc::{size_t, c_uint};

#[no_mangle]
#[allow(non_snake_case)]
pub extern fn CalculateTreeSize(ll_lengths: *const c_uint, d_lengths: *const
c_uint) -> size_t {
  let mut result = 0;

  for i in 0..8 {
    let size = EncodeTreeNoOutput(ll_lengths, d_lengths,
                                  i & 1, i & 2, i & 4);
    if result == 0 || size < result {
      result = size;
    }
  }

  result
}
```

Idiomaticize

```
pub fn calculate_tree_size(ll_lengths: &[u32], d_lengths: &[u32]) -> usize {
  let mut result = 0;

  for i in 0..8 {
    let size = encode_tree_no_output(ll_lengths, d_lengths,
                                     i & 1, i & 2, i & 4);
    if result == 0 || size < result {
      result = size;
    }
  }

  result
}
```

Iterators!

Idiomaticize

```
pub fn calculate_tree_size(ll_lengths: &[u32], d_lengths: &[u32]) -> usize {
    (0..8).map(|i| {
        encode_tree_no_output(ll_lengths, d_lengths,
                              i & 1 > 0, i & 2 > 0, i & 4 > 0)
    }).min().unwrap_or(0)
}
```

yinz ready for lots of code?

```
typedef double CostModelFun(unsigned litlen, unsigned dist, void* context);

/* type: CostModelFun */
static double GetCostFixed(unsigned litlen, unsigned dist, void* unused) {…}

/* type: CostModelFun */
static double GetCostStat(unsigned litlen, unsigned dist, void* context) {…}

static double GetCostModelMinCost(CostModelFun* costmodel, void* costcontext) {
    // . . .
    double c = costmodel(i, 1, costcontext);
    // . . .
}

void ZopfliLZ77OptimalFixed(…) {
    // . . .
    GetCostModelMinCost(GetCostFixed, 0);
    // . . .
}

void ZopfliLZ77Optimal(…) {
    // . . .
    GetCostModelMinCost(GetCostStat, (void*)&stats);
    // . . .
}
```

```
fn get_cost_fixed(litlen: c_uint, dist: c_uint, _unused: c_void) -> c_double {…}

fn get_cost_stat(litlen: c_uint, dist: c_uint, context: *const c_void) -> c_double
{…}



fn get_cost_model_min_cost(
    costmodel: fn(c_uint, c_uint, *const c_void) -> c_double,
    costcontext: *const c_void) -> c_double {
    // . . .
    let c = costmodel(i, 1, costcontext);
    // . . .
}


fn lz77_optimal_fixed(…) {
    // . . .
    get_cost_model_min_cost(get_cost_fixed, ptr::null());
    // . . .
}

fn lz77_optimal(…) {
    // . . .
    let stats_ptr: *const SymbolStats = &stats;
    get_cost_model_min_cost(get_cost_stat, stats_ptr as *const c_void);
    // . . .
}
```

```
fn get_cost_fixed(litlen: c_uint, dist: c_uint, _unused: Option<&SymbolStats>) ->
c_double {…}

fn get_cost_stat(litlen: c_uint, dist: c_uint, context: Option<&SymbolStats>) ->
c_double {…}


fn get_cost_model_min_cost(
    costmodel: fn(c_uint, c_uint, Option<&SymbolStats>) -> c_double,
    costcontext: Option<&SymbolStats>) -> c_double {
    // . . .
    let c = costmodel(i, 1, costcontext);
    // . . .
}


fn lz77_optimal_fixed(…) {
    // . . .
    get_cost_model_min_cost(get_cost_fixed, None);
    // . . .
}

fn lz77_optimal(…) {
    // . . .
    get_cost_model_min_cost(get_cost_stat, Some(&stats));
    // . . .
}
```

```
- fn get_cost_fixed(litlen: c_uint, dist: c_uint, _unused: Option<&SymbolStats>) ->
c_double {…}
+ fn get_cost_fixed(litlen: c_uint, dist: c_uint) -> c_double {…}

- fn get_cost_stat(litlen: c_uint, dist: c_uint, context: Option<&SymbolStats>) ->
c_double {…}
+ fn get_cost_stat(litlen: c_uint, dist: c_uint, context: &SymbolStats) -> c_double
{…}
```

```
 - fn get_cost_model_min_cost(
 -     costmodel: fn(c_uint, c_uint, Option<&SymbolStats>) -> c_double,
 -     costcontext: Option<&SymbolStats>) -> c_double {
 + fn get_cost_model_min_cost<F>(costmodel: F) -> c_double
 +     where F: Fn(c_uint, c_uint) -> c_double
 + {
      // . . .
 -    let c = costmodel(i, 1, costcontext);
 +    let c = costmodel(i, 1);
      // . . .
   }
```

```
fn get_cost_fixed(litlen: c_uint, dist: c_uint) -> c_double {…}

fn get_cost_stat(litlen: c_uint, dist: c_uint, context: &SymbolStats) -> c_double
{…}

fn get_cost_model_min_cost<F>(costmodel: F) -> c_double
    where F: Fn(c_uint, c_uint) -> c_double
{
   // . . .
   let c = costmodel(i, 1);
   // . . .
}

 fn lz77_optimal_fixed(…) {
     // . . .
-    get_cost_model_min_cost(get_cost_fixed, None);
+    get_cost_model_min_cost(get_cost_fixed);
     // . . .
 }

 fn lz77_optimal(…) {
     // . . .
-    get_cost_model_min_cost(get_cost_stat, Some(&stats));
+    get_cost_model_min_cost(|a, b| get_cost_stat(a, b, &stats));
     // . . .
 }
```

Algebraic Data Types!

Closures!

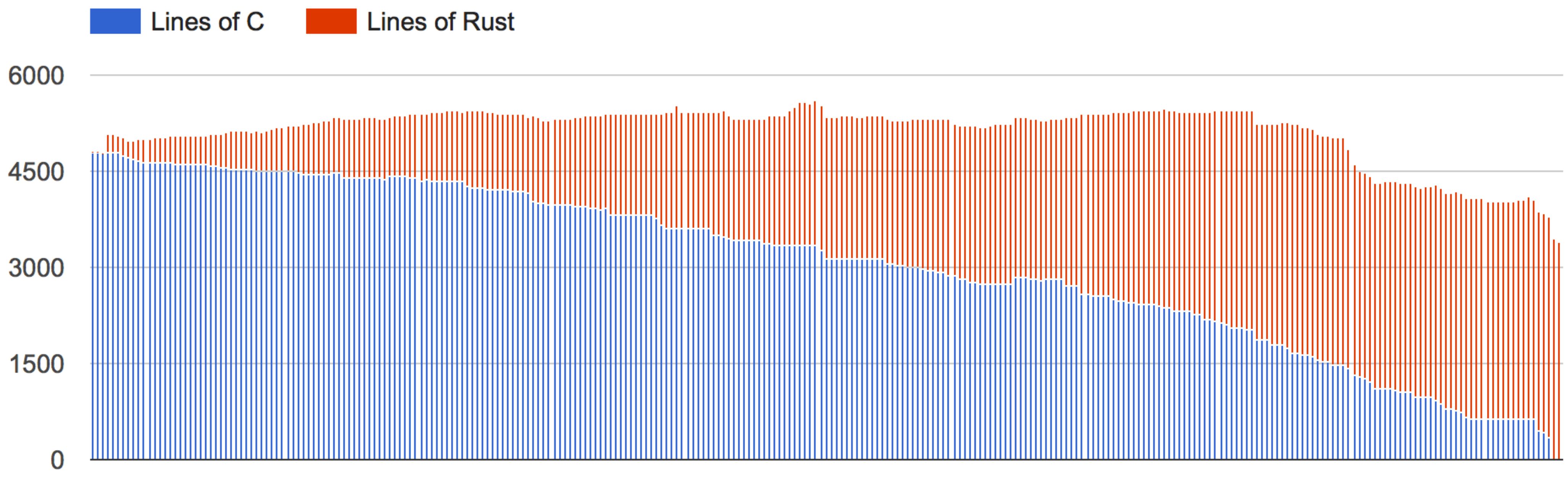
Traits!
(not covered)

Benefits?

Safety

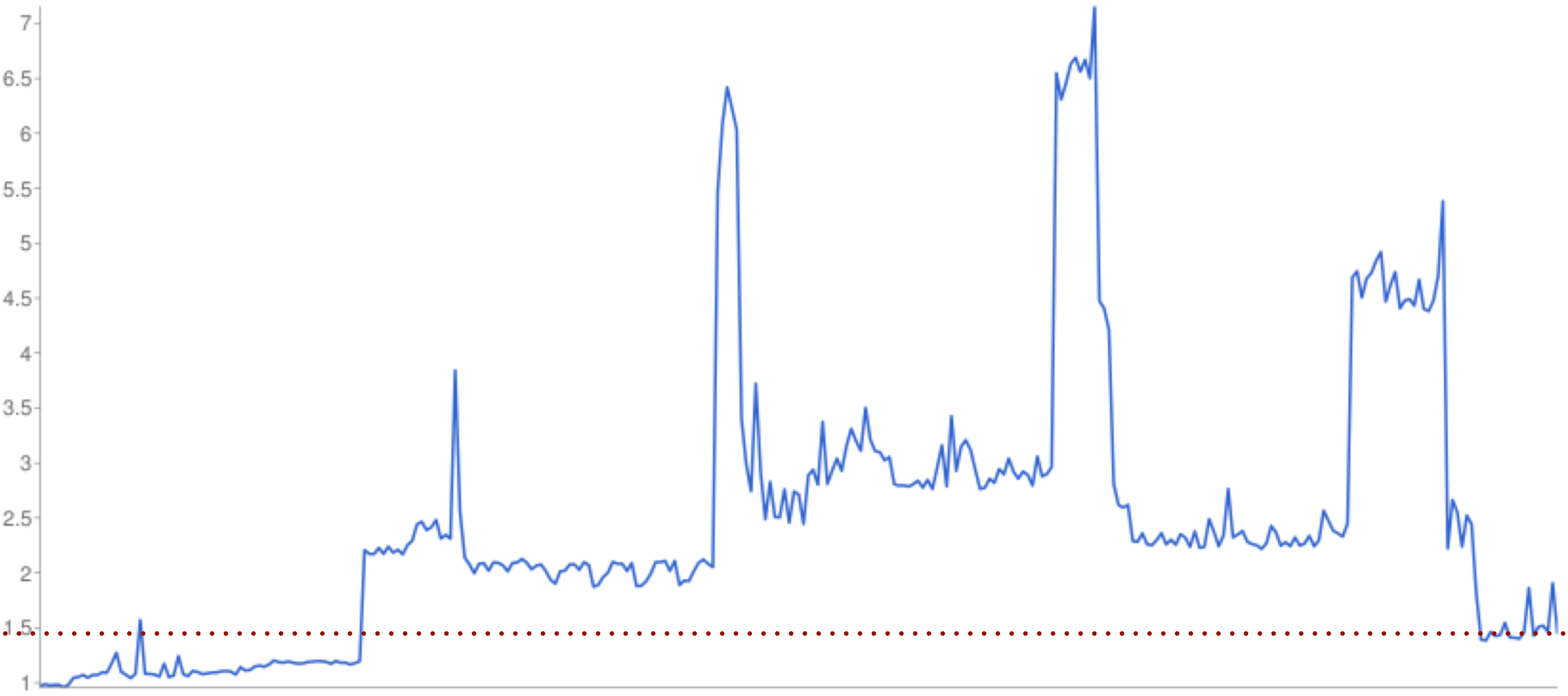
Clarity (more functional)

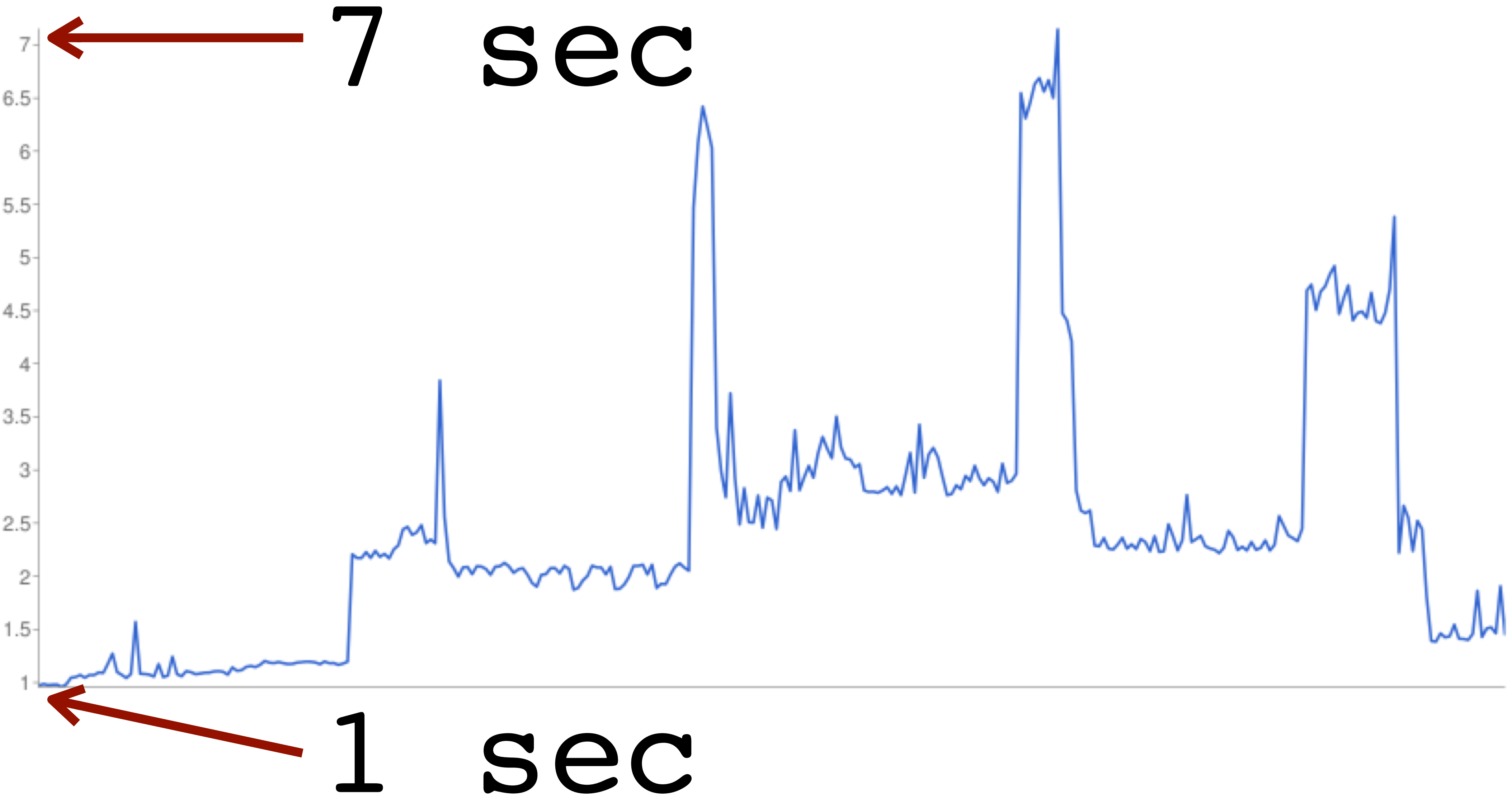
Less code



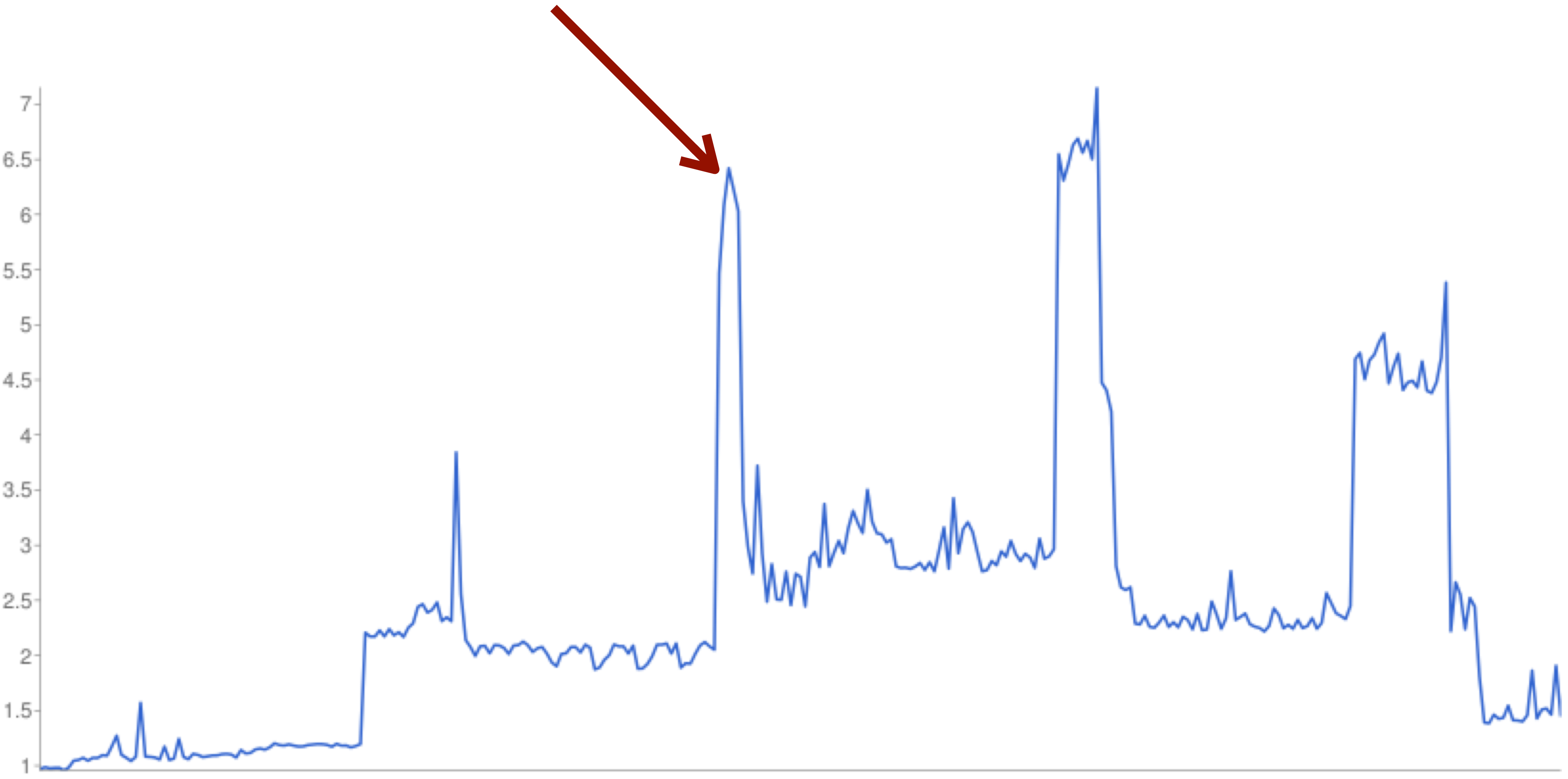
4777 LOC in C -> 3399 LOC in Rust = 71%

Performance


7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1



7 sec
1 sec
7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1


7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1

Bad news :(

```
struct List {
    lookahead1: Node,
    lookahead2: Node,
    next_leaf_index: usize,
}

struct Node {
    weight: usize,
    leaf_counts: Vec<usize>,
}
```

```
current_list.lookahead2 = Node {
    weight: next_leaf.weight,
    leaf_counts: vec![
        current_list
            .lookahead1
            .leaf_counts
            .last()
            .unwrap() + 1
    ],
};
```

```
current_list.lookahead2.weight =
    next_leaf.weight;


current_list.lookahead2.leaf_counts[0] =
    current_list
        .lookahead1
        .leaf_counts
        .last()
        .unwrap() + 1;
```

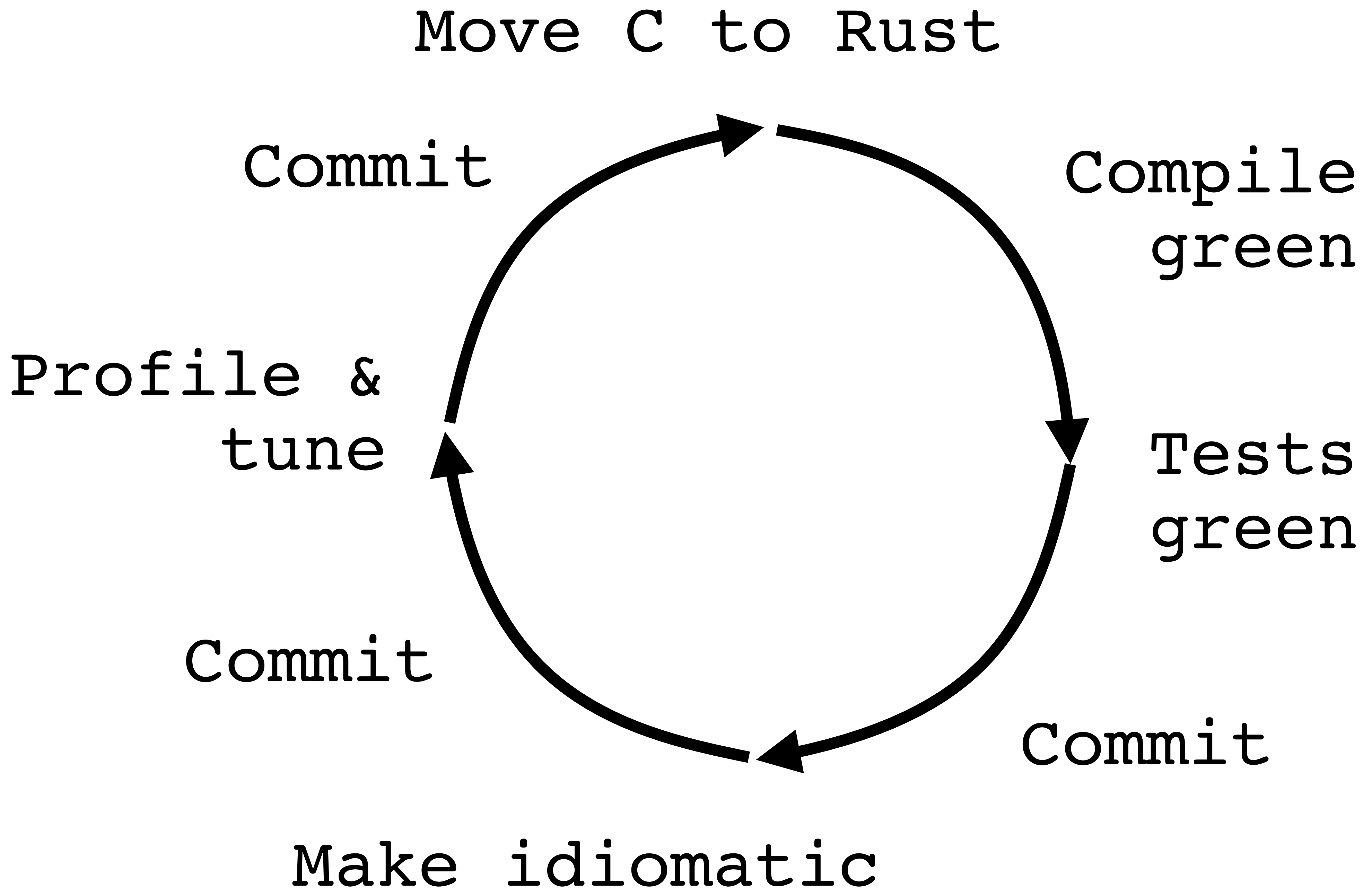
```
fn do_something(list: &mut List)

let mut list = …
do_something(&mut list);
```

vs

```
fn do_something(mut list: List) -> List

let list = …
let list = do_something(list);
```


Move C to Rust
Compile green
Tests green
Commit
Make idiomatic
Commit
Profile & tune
Commit

C-like Rust is slower than idiomatic Rust?

Future work

- Remove all `unsafe`
- Use more iterators
- Stream input/output
- Refactor forever

¯_(ツ)_/¯

my point:

- Incremental rewrites from C to Rust are possible.
- Rust has FP concepts that can improve C code as you rewrite it
- Have reasons for rewriting and measure progress against the reasons.

References (is.gd/c_rust)

- Repo of my code
- These slides
- FFI chapter in The Rust Programming Language book
- Rust FFI Omnibus
- Working Effectively with Legacy Code by Michael Feathers

Thank You

Carol (Nichols || Goulding)
@carols10cents