Rust out your C

Carol (Nichols || Goulding)
@carols10cents

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

Agenda

1. **Caveats**
2. **Background**
3. **Techniques**
4. **Tricky Stuff**
5. **Benefits?**

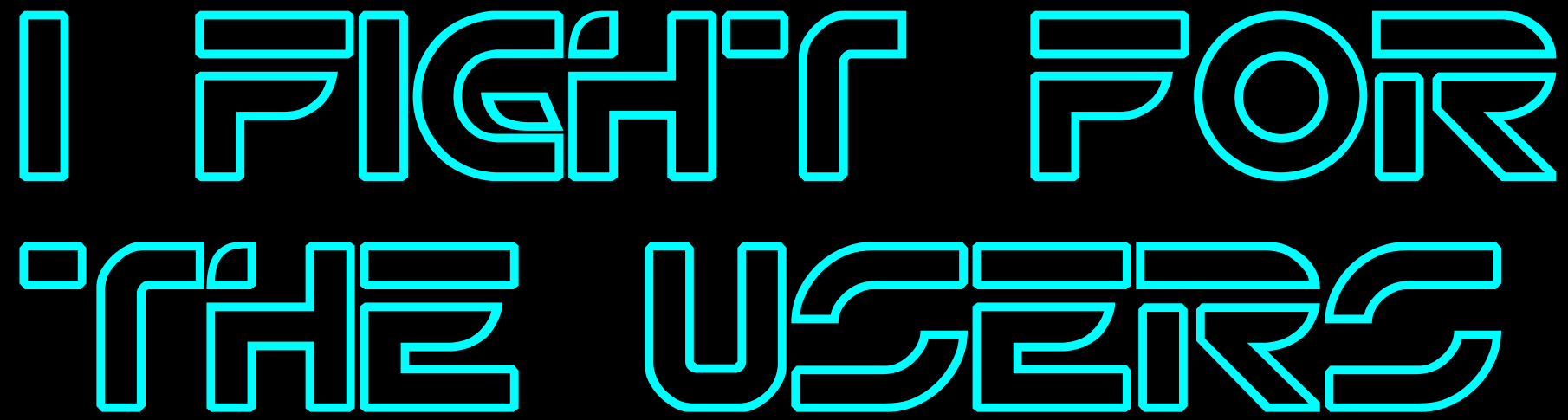
DO NOT

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

Google™ Custom Search

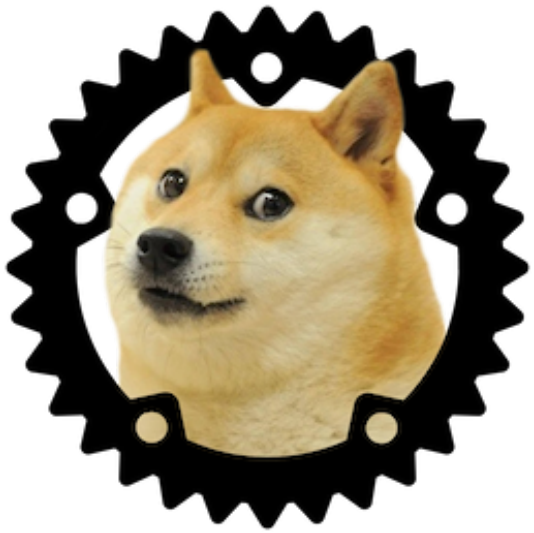
02 Jan 2016

Zopfli Optimization: Literally Free Bandwidth

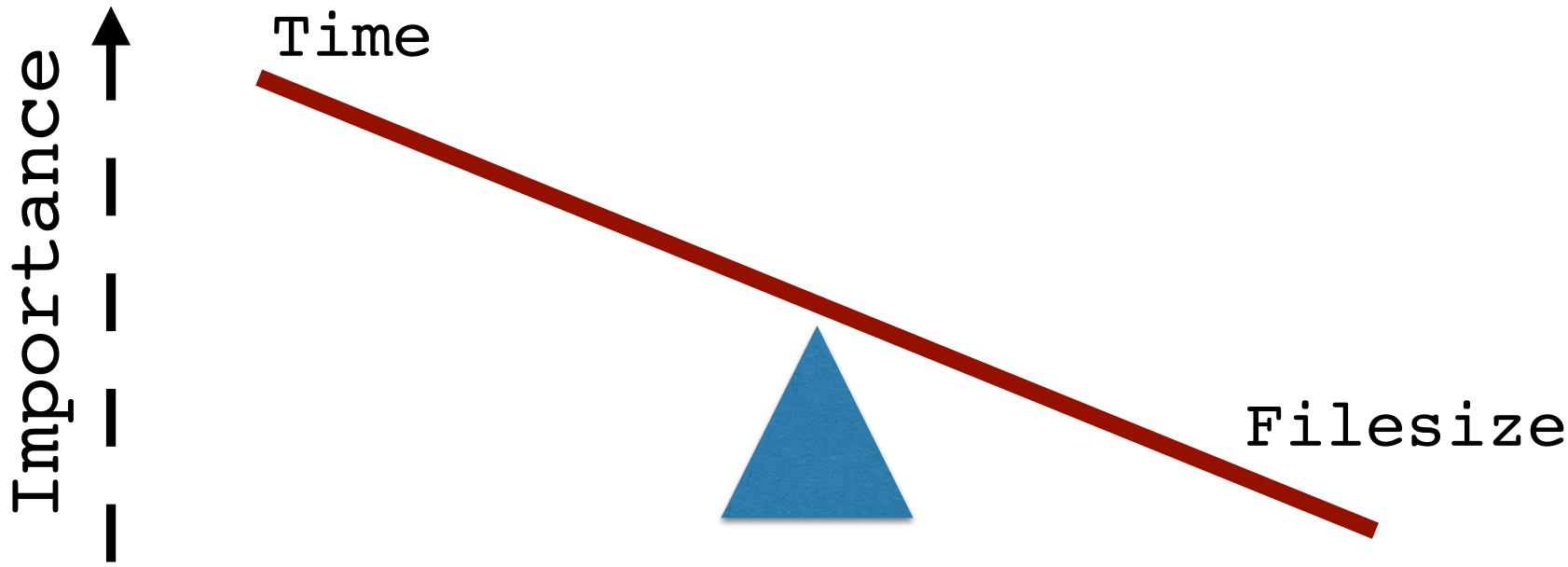
https://blog.codinghorror.com/zopfli-optimization-literally-free-bandwidth/

This is a DEFLATE file, I know this!

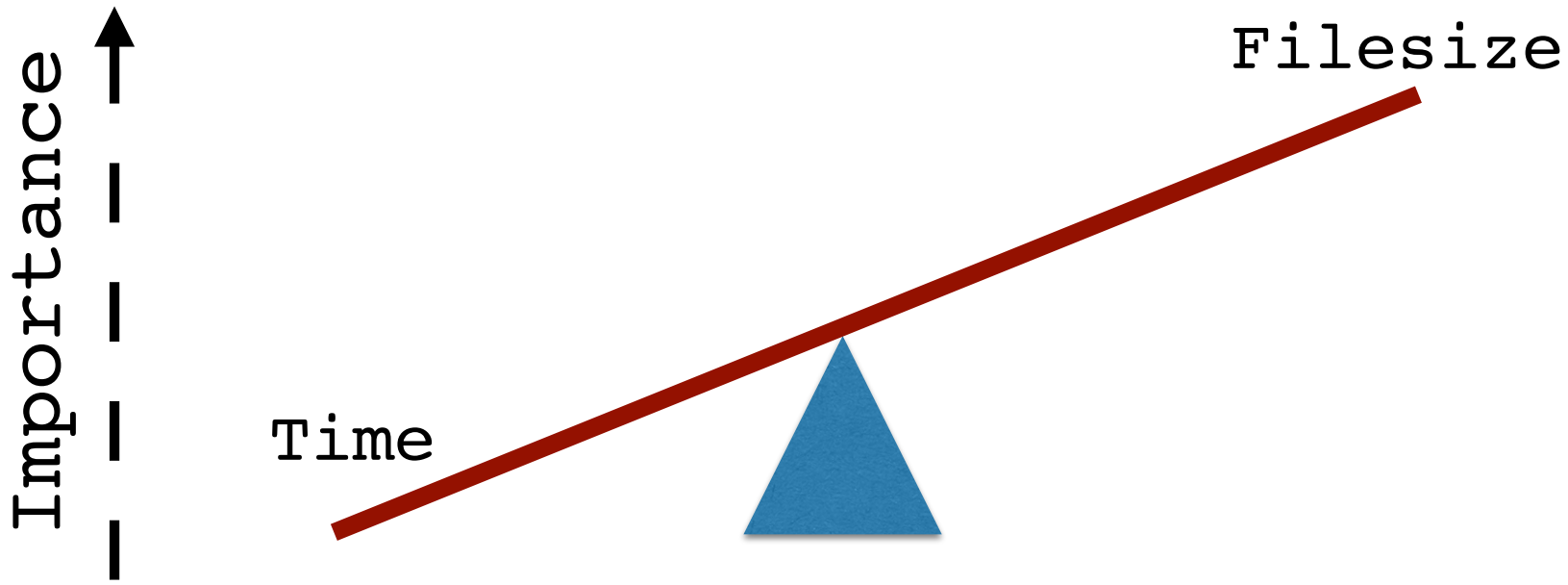


gzip



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

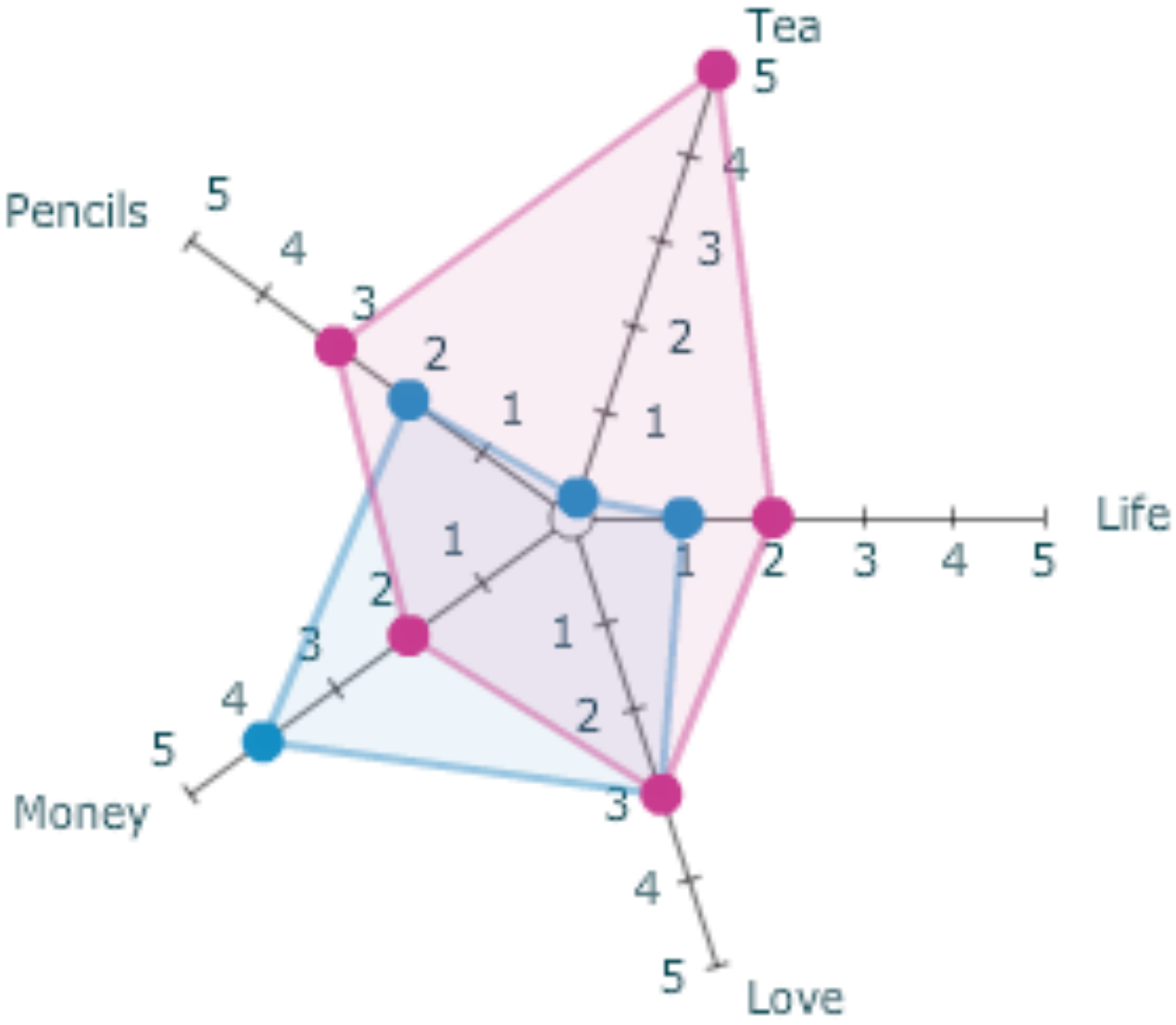
schneems commented on Dec 8, 2015 Ruby on Rails member

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

Makefile changes

```
+ZOPFLI_RUST_RELEASE := target/release/libzopfli.a
 ZOPFLILIB_SRC = src/zopfli/blocksplitter.c src/zopfli/cache.c…
```

Makefile changes

```
 ZOPFLI_RUST_RELEASE := target/release/libzopfli.a
 ZOPFLILIB_SRC = src/zopfli/blocksplitter.c src/zopfli/cache.c…

+.PHONY: target/release/libzopfli.a
+target/release/libzopfli.a:
+	cargo build --verbose --release
```

Makefile changes

```
 ZOPFLI_RUST_RELEASE := target/release/libzopfli.a
 ZOPFLILIB_SRC = src/zopfli/blocksplitter.c src/zopfli/cache.c…

 .PHONY: target/release/libzopfli.a
 target/release/libzopfli.a:
        cargo build --verbose --release

-zopfli:
-       $(CC) $(ZOPFLILIB_SRC) $(ZOPFLIBIN_SRC) $(CFLAGS) -o zopfl
+zopfli: $(ZOPFLI_RUST_RELEASE)
+       $(CC) $(ZOPFLI_RUST_RELEASE) $(ZOPFLILIB_SRC) $
(ZOPFLIBIN_SRC) $(CFLAGS) -o zopfli
```

Cargo.toml changes

```
+[lib]
+crate-type = ["staticlib"]
```

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

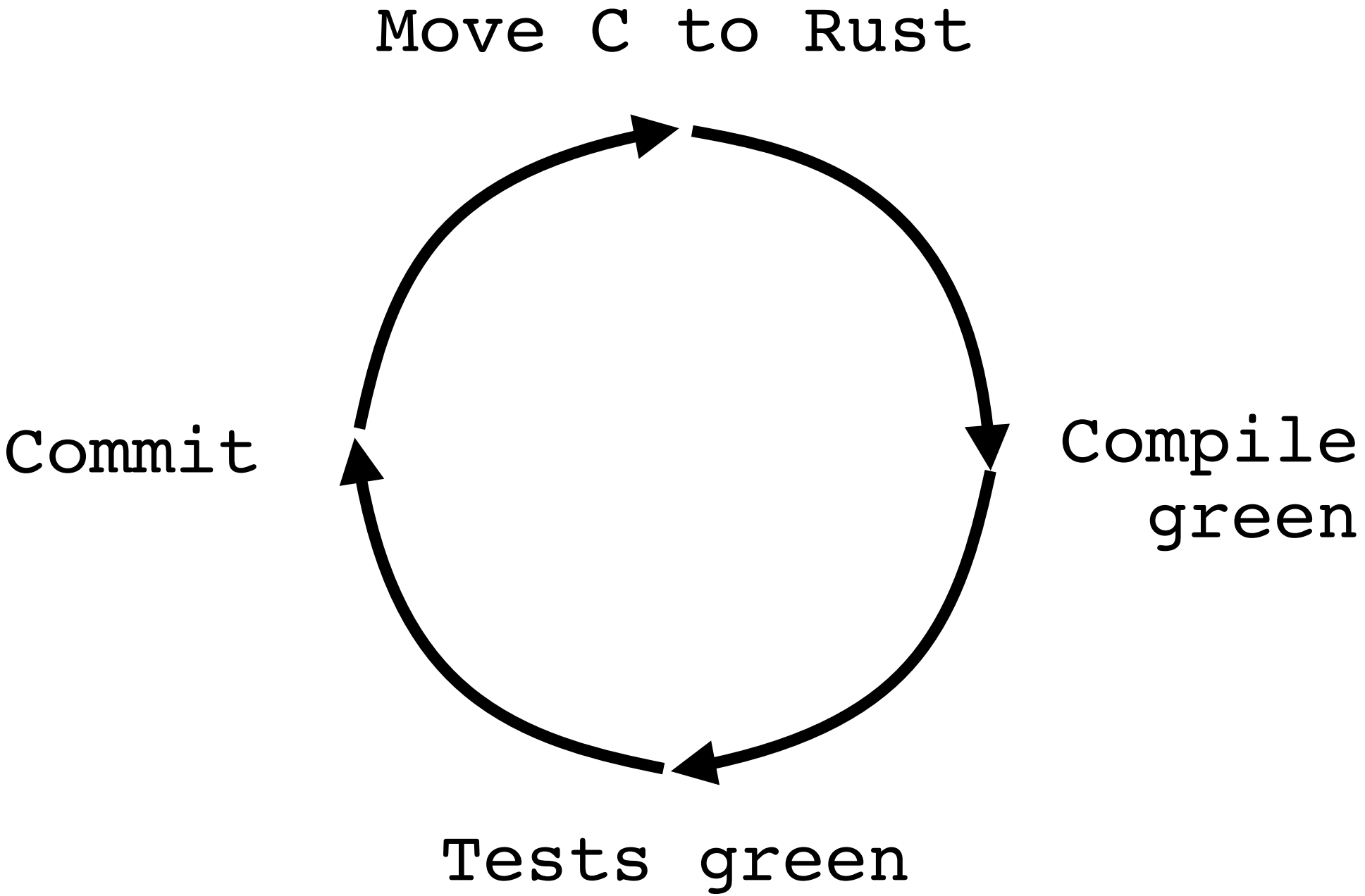
```
use libc::{size_t, c_uint};

#[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.

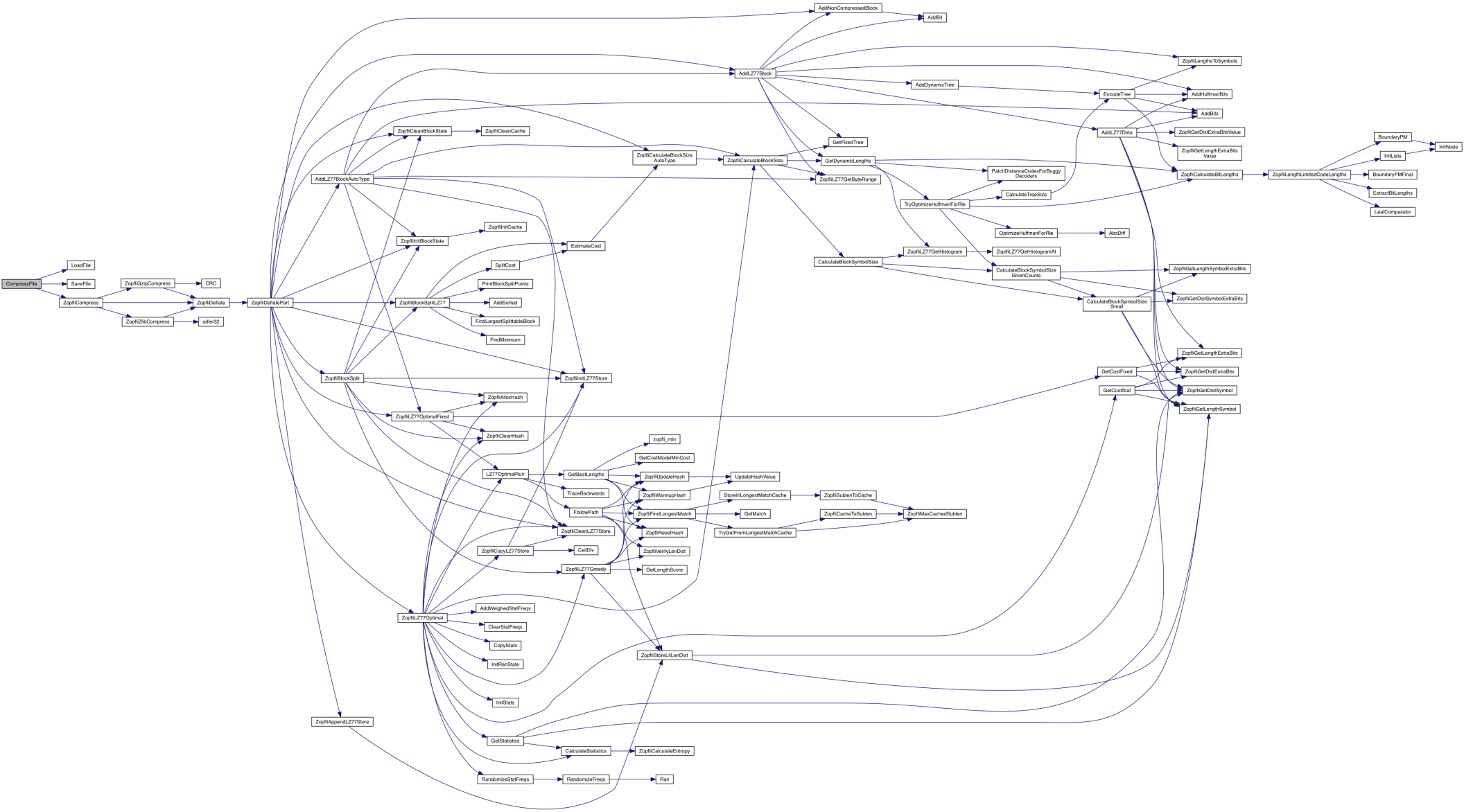
smaller

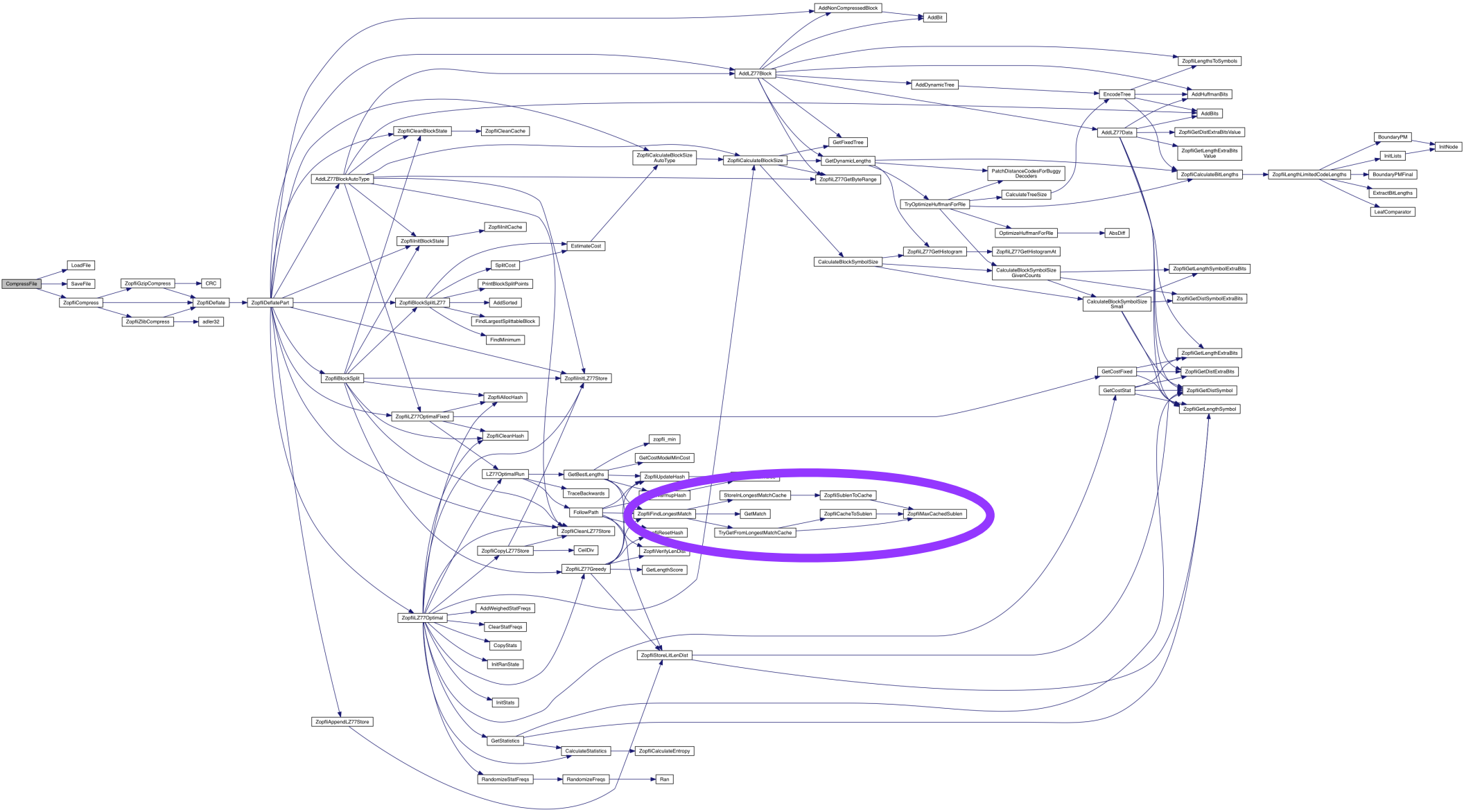
- Extract functions
- Make the C more like Rust first
- Don’t make the Rust idiomatic at all, even when it seems easy

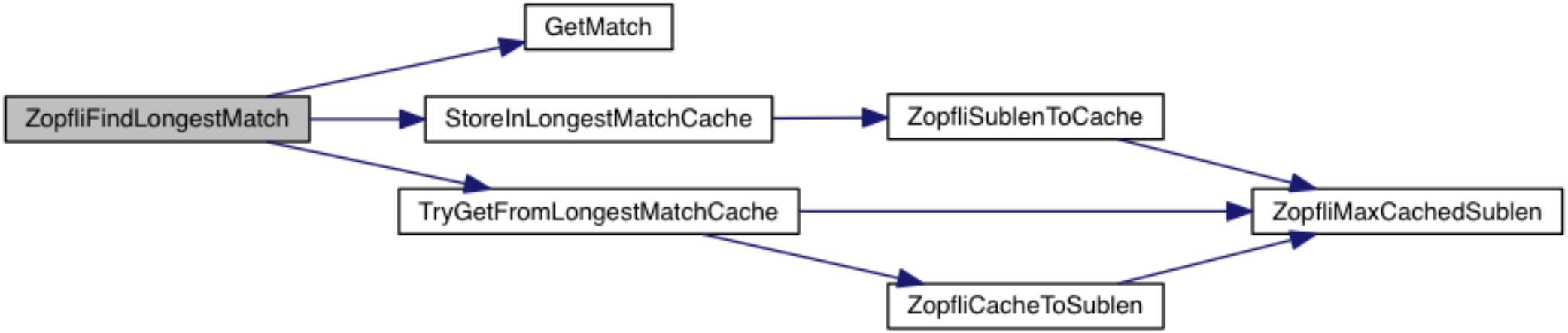
✅ how

what

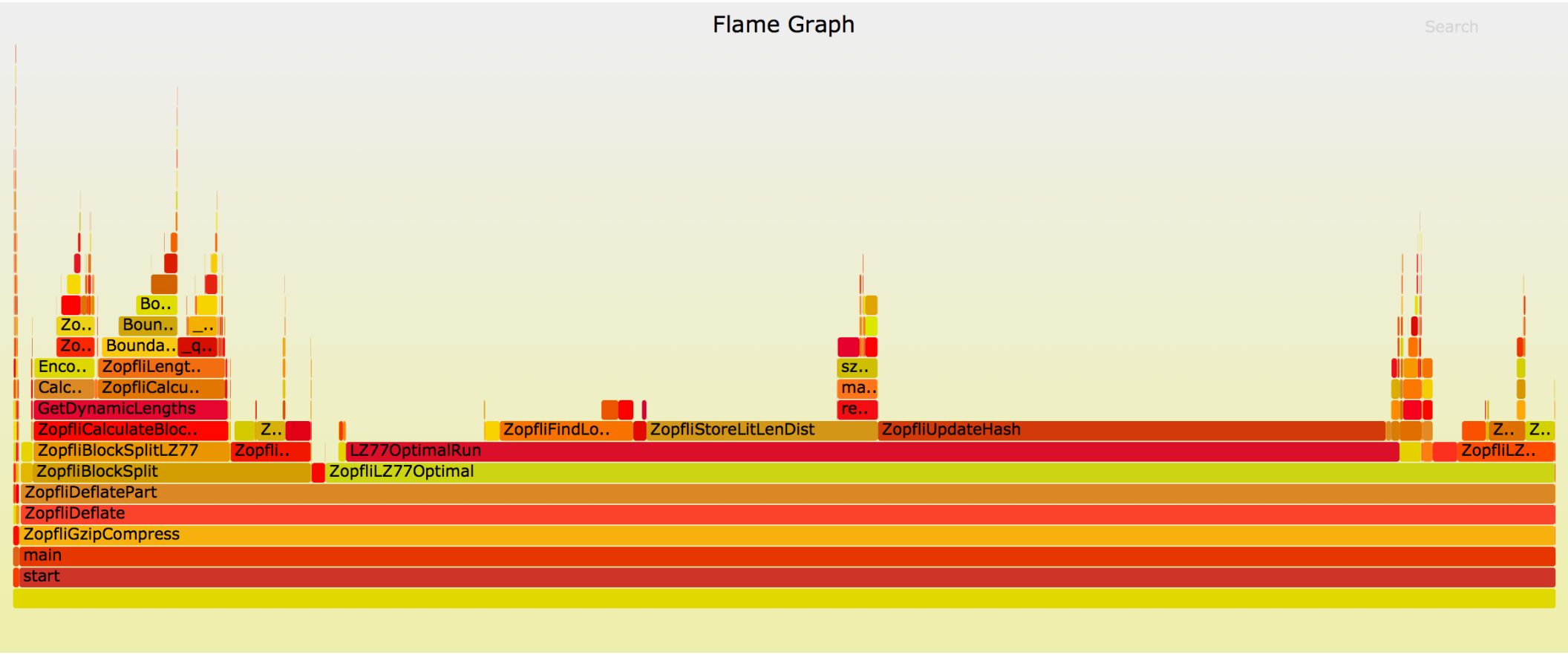
Follow the call graph


CompressFile
LoadFile
SaveFile
ZopfliCompress
ZopfliGzipCompress
ZopfliZlibCompress
CRC
adler32
ZopfliDeflate
ZopfliDeflatePart
AddNonCompressedBlock
AddBit
AddLZ77Block
AddDynamicTree
EncodeTree
ZopfliLengthsToSymbols
AddHuffmanBits
AddBits
AddLZ77Data
ZopfliGetDistExtraBitsValue
ZopfliGetLengthExtraBitsValue
ZopfliCalculateBitLengths
ZopfliLengthLimitedCodeLengths
BoundaryPM
InitLists
InitNode
BoundaryPMFinal
ExtractBitLengths
LeafComparator
ZopfliCleanBlockState
ZopfliCleanCache
AddLZ77BlockAutoType
ZopfliCalculateBlockSizeAutoType
ZopfliCalculateBlockSize
GetFixedTree
GetDynamicLengths
ZopfliLZ77GetByteRange
PatchDistanceCodesForBuggyDecoders
CalculateTreeSize
TryOptimizeHuffmanForRle
OptimizeHuffmanForRle
AbsDiff
ZopfliInitBlockState
ZopfliInitCache
EstimateCost
SplitCost
PrintBlockSplitPoints
AddSorted
FindLargestSplittableBlock
FindMinimum
ZopfiBlockSplitLZ77
CalculateBlockSymbolSize
ZopfiLZ77GetHistogram
ZopfiLZ77GetHistogramAt
CalculateBlockSymbolSizeGivenCounts
CalculateBlockSymbolSizeSmall
ZopfiGetLengthSymbolExtraBits
ZopfiGetDistSymbolExtraBits
ZopfiGetLengthExtraBits
ZopfiGetDistExtraBits
ZopfiGetDistSymbol
ZopfiGetLengthSymbol
GetCostFixed
GetCostStat
ZopfiBlockSplit
ZopfiInitLZ77Store
ZopfiAllocHash
ZopfiLZ77OptimalFixed
ZopfiCleanHash
zopfli_min
GetCostModelMinCost
LZ77OptimalRun
GetBestLengths
ZopfiUpdateHash
UpdateHashValue
TraceBackwards
ZopfiWarmupHash
StoreInLongestMatchCache
ZopfiSublenToCache
FollowPath
ZopfiFindLongestMatch
GetMatch
ZopfiCacheToSublen
ZopfiMaxCachedSublen
TryGetFromLongestMatchCache
ZopfiCleanLZ77Store
ZopfiResetHash
ZopfiCopyLZ77Store
CeilDiv
ZopfiVerifyLenDist
ZopfiLZ77Greedy
GetLengthScore
ZopfiLZ77Optimal
AddWeighedStatFreqs
ClearStatFreqs
CopyStats
InitRanState
InitStats
ZopfiStoreLitLenDist
ZopfiAppendLZ77Store
GetStatistics
CalculateStatistics
ZopfiCalculateEntropy
RandomizeStatFreqs
RandomizeFreqs
Ran

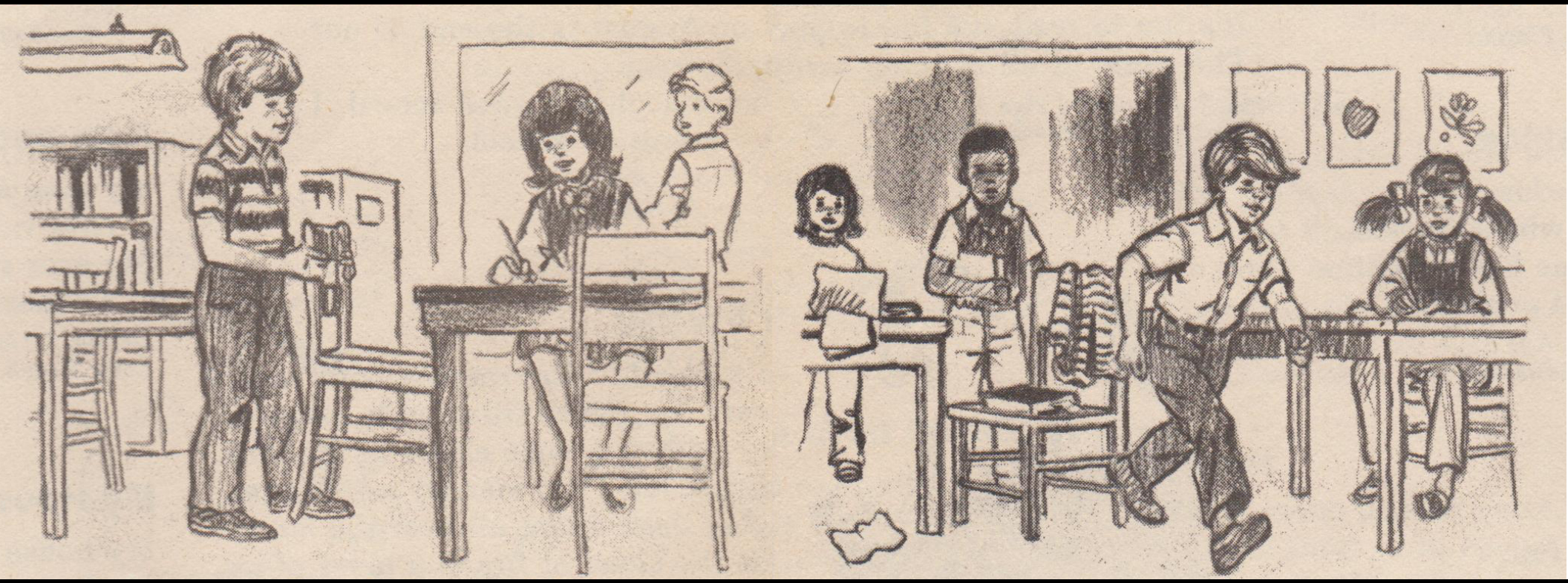

CompressFile
LoadFile
SaveFile
ZopfliCompress
ZopfliGzipCompress
ZopfliZlibCompress
CRC
ZopfliDeflate
adler32
ZopfliDeflatePart
AddNonCompressedBlock
AddBit
AddLZ77Block
AddDynamicTree
EncodeTree
ZopfliLengthsToSymbols
AddHuffmanBits
AddBits
AddLZ77Data
ZopfliGetDistExtraBitsValue
ZopfliGetLengthExtraBits Value
ZopfliCalculateBitLengths
ZopfliLengthLimitedCodeLengths
BoundaryPM
InitLists
InitNode
BoundaryPMFinal
ExtractBitLengths
LeafComparator
ZopfliCleanBlockState
ZopfliCleanCache
AddLZ77BlockAutoType
ZopfliCalculateBlockSize AutoType
ZopfliCalculateBlockSize
GetFixedTree
GetDynamicLengths
ZopfliLZ77GetByteRange
PatchDistanceCodesForBuggy Decoders
CalculateTreeSize
TryOptimizeHuffmanForRle
ZopfliInitBlockState
ZopfliInitCache
EstimateCost
SplitCost
PrintBlockSplitPoints
AddSorted
FindLargestSplittableBlock
FindMinimum
ZopfliBlockSplitLZ77
OptimizeHuffmanForRle
AbsDiff
CalculateBlockSymbolSize
ZopfliLZ77GetHistogram
ZopfliLZ77GetHistogramAt
CalculateBlockSymbolSize GivenCounts
CalculateBlockSymbolSize Small
ZopfliGetLengthSymbolExtraBits
ZopfliGetDistSymbolExtraBits
ZopfliGetLengthExtraBits
ZopfliGetDistExtraBits
ZopfliGetDistSymbol
ZopfliGetLengthSymbol
GetCostFixed
GetCostStat
ZopfliBlockSplit
ZopfliInitLZ77Store
ZopfliAllocHash
ZopfliLZ77OptimalFixed
ZopfliCleanHash
zopfli_min
GetCostModelMinCost
LZ77OptimalRun
GetBestLengths
ZopfliUpdateHash
TraceBackwards
FollowPath
ZopfliFindLongestMatch
StoreInLongestMatchCache
ZopfliSublenToCache
GetMatch
ZopfliCacheToSublen
ZopfliMaxCachedSublen
TryGetFromLongestMatchCache
ZopfliCleanLZ77Store
ZopfliResetHash
ZopfliCopyLZ77Store
CeilDiv
ZopfliVerifyLenDist
ZopfiLZ77Greedy
GetLengthScore
ZopfiLZ77Optimal
AddWeighedStatFreqs
ClearStatFreqs
CopyStats
InitRanState
InitStats
ZopfliStoreLitLenDist
ZopfliAppendLZ77Store
GetStatistics
CalculateStatistics
ZopfliCalculateEntropy
RandomizeStatFreqs
RandomizeFreqs
Ran


GetMatch
ZopfliFindLongestMatch
StoreInLongestMatchCache
ZopfliSublenToCache
TryGetFromLongestMatchCache
ZopfliMaxCachedSublen
ZopfliCacheToSublen

Profile


Flame Graph
Search
Bo..
Zo..
Boun..
_..
Zo..
Bounda..
_q..
Enco..
ZopfliLengt..
sz..
Calc..
ZopfliCalcu..
ma..
GetDynamicLengths
re..
ZopfliCalculateBloc..
Z..
ZopfliFindLo..
ZopfliStoreLitLenDist
ZopfliUpdateHash
Z..
Z..
ZopfliBlockSplitLZ77
Zopfli..
LZ77OptimalRun
ZopfliLZ..
ZopfliBlockSplit
ZopfliLZ77Optimal
ZopfliDeflatePart
ZopfliDeflate
ZopfliGzipCompress
main
start

Tricky stuff



Rust upholds your invariants.

C says, “Nuts to your invariants.”

```
static void RandomizeFreqs(RanState*
state, size_t* freqs, int n) {

  int i;
  for (i = 0; i < n; i++) {

    if ((Ran(state) >> 4) % 3 == 0) {
      freqs[i] = freqs[Ran(state) % n];
    }

  }

}
```

```
static void RandomizeFreqs(RanState*
state, size_t* freqs, int n) {

  int i;
  for (i = 0; i < n; i++) {

    if ((Ran(state) >> 4) % 3 == 0) {
      freqs[i] = freqs[Ran(state) % n];
    }

  }

}
```

```
for freq in freqs.iter_mut() {
    if (Ran(state) >> 4) % 3 == 0 {
        let index = (Ran(state) % n as
u32) as usize;
        *freq = freqs[index];
    }
}
```

```
for freq in freqs.iter_mut() {
    if (Ran(state) >> 4) % 3 == 0 {
        let index = (Ran(state) % n as
u32) as usize;
        *freq = freqs[index];
    }
}
```

```
error: cannot use `freqs[..]` because it
was mutably borrowed [E0503]
  *freq = freqs[index];
          ^~~~~~~~~~~~
note: borrow of `*freqs` occurs here
for freq in freqs.iter_mut() {
            ^~~~
```

```
for i in 0..n {
    if (Ran(state) >> 4) % 3 == 0 {
        let index = (Ran(state) % n as
u32) as usize;
        freqs[i] = freqs[index];
    }
}
```

RandomizeFreqs…
shuffle?

Type conversions 😱

```
double GetBestLengths(…, float* costs, …) {

    double newCost = costmodel(in[i]) + costs[j];

    if (newCost < costs[j + 1]) {
        costs[j + 1] = newCost;
    }
    …
}
```

Type conversions 😱

```
double GetBestLengths(…, float* costs, …) {

    double newCost = costmodel(in[i]) + costs[j];

    if (newCost < costs[j + 1]) {
        costs[j + 1] = newCost;
    }
    …
}
```

```
fn GetBestLengths(…, costs: *mut c_float, …) -> c_double {

    let newCost = costmodel(in[i]) + costs[j] as c_double;

    if newCost < costs[j + 1] as c_double {
        costs[j + 1] = newCost as c_float;
    }
    …
}
```

Type conversions 😱

```
double GetBestLengths(…, float* costs, …) {

    double newCost = costmodel(in[i]) + costs[j];

    if (newCost < costs[j + 1]) {
        costs[j + 1] = newCost;
    }
    …
}
```

```
fn GetBestLengths(…, costs: *mut c_float, …) -> c_double {

    let newCost = costmodel(in[i]) + costs[j] as c_double;

    if newCost < costs[j + 1] as c_double {
        costs[j + 1] = newCost as c_float;
    }
    …
}
```

0 means…

- 0
- 0.0
- False
- Null pointer
- End of string
- Sadness
- Frustration

Benefits?

Safety

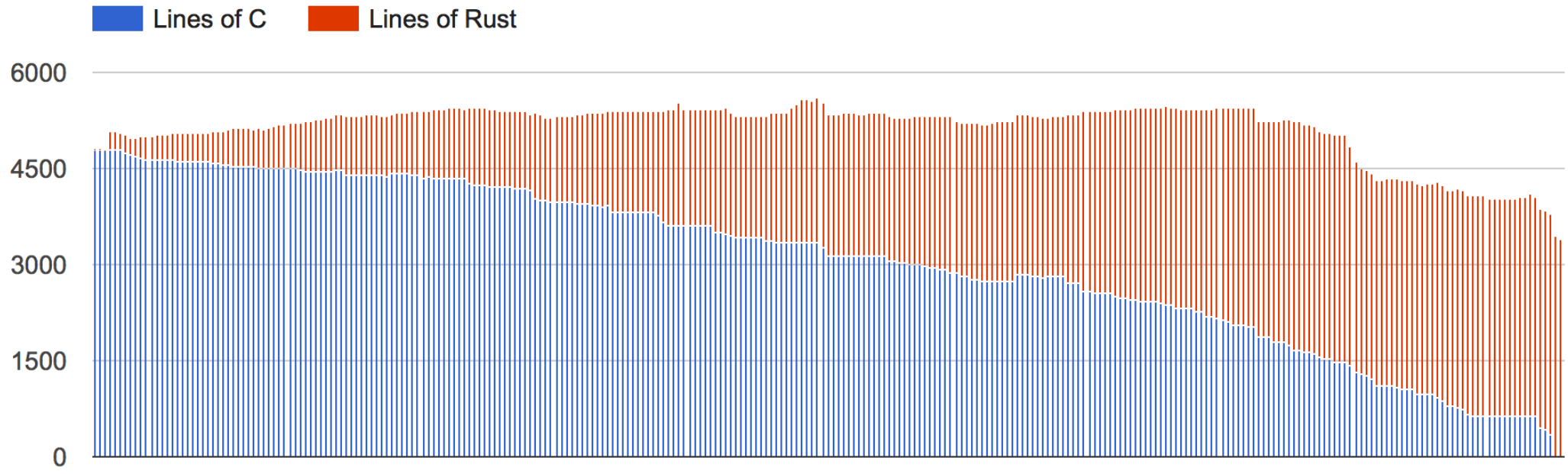
Clarity

Less code

```
#define ZOPFLI_APPEND_DATA(value, data, size) {\
  if (!((*size) & ((*size) - 1))) {\
    /*double alloc size if it's a power of two*/\
    (*data) = (*size) == 0 ? malloc(sizeof(**data))\
                           : realloc((*data), (*size) * 2 *
sizeof(**data));\
  }\
  (*data)[(*size)] = (value);\
  (*size)++;\
}
```

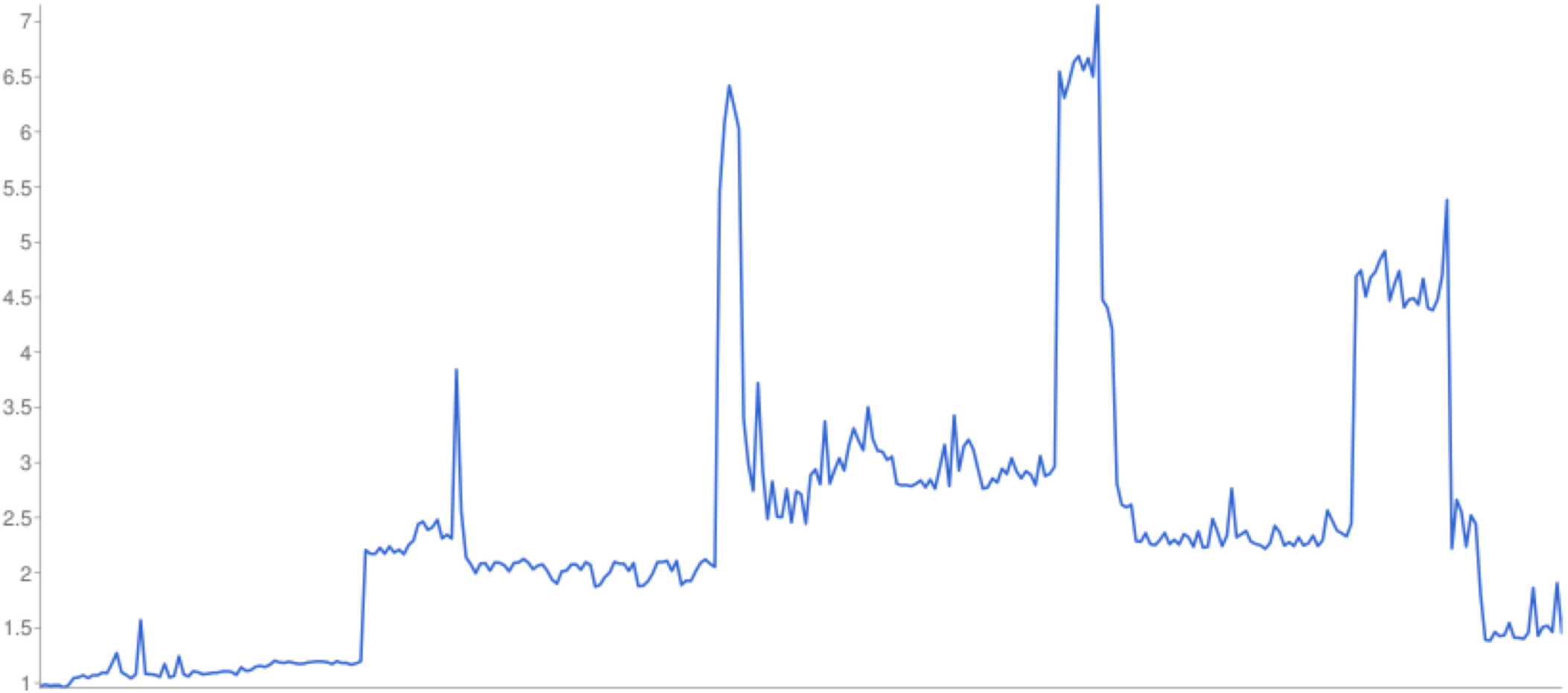
```
#define ZOPFLI_APPEND_DATA(value, data, size) {\
  if (!((*size) & ((*size) - 1))) {\
    /*double alloc size if it's a power of two*/\
    (*data) = (*size) == 0 ? malloc(sizeof(**data))\
                           : realloc((*data), (*size) * 2 *
sizeof(**data));\
  }\
  (*data)[(*size)] = (value);\
  (*size)++;\
}
```

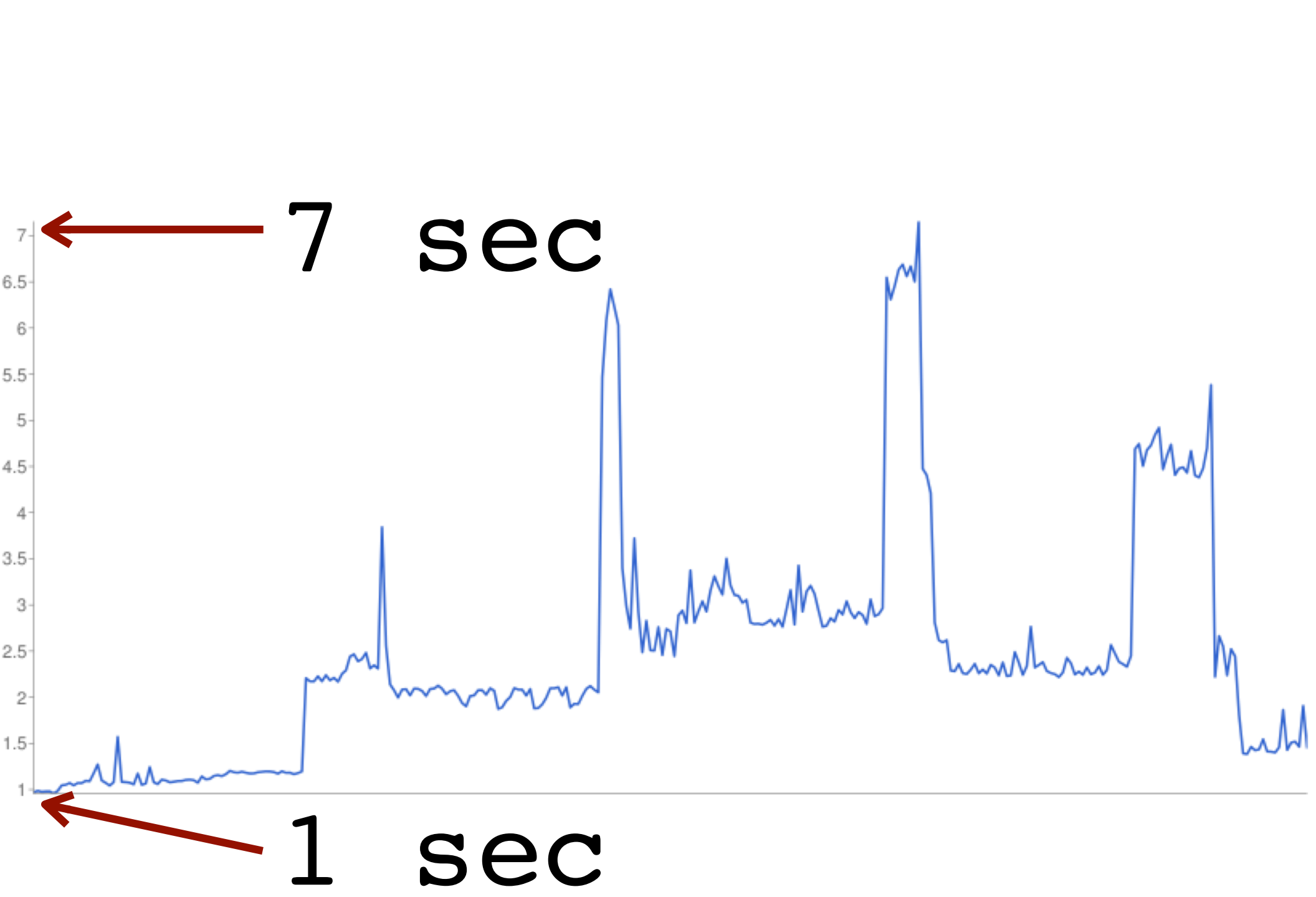
data.push(value)



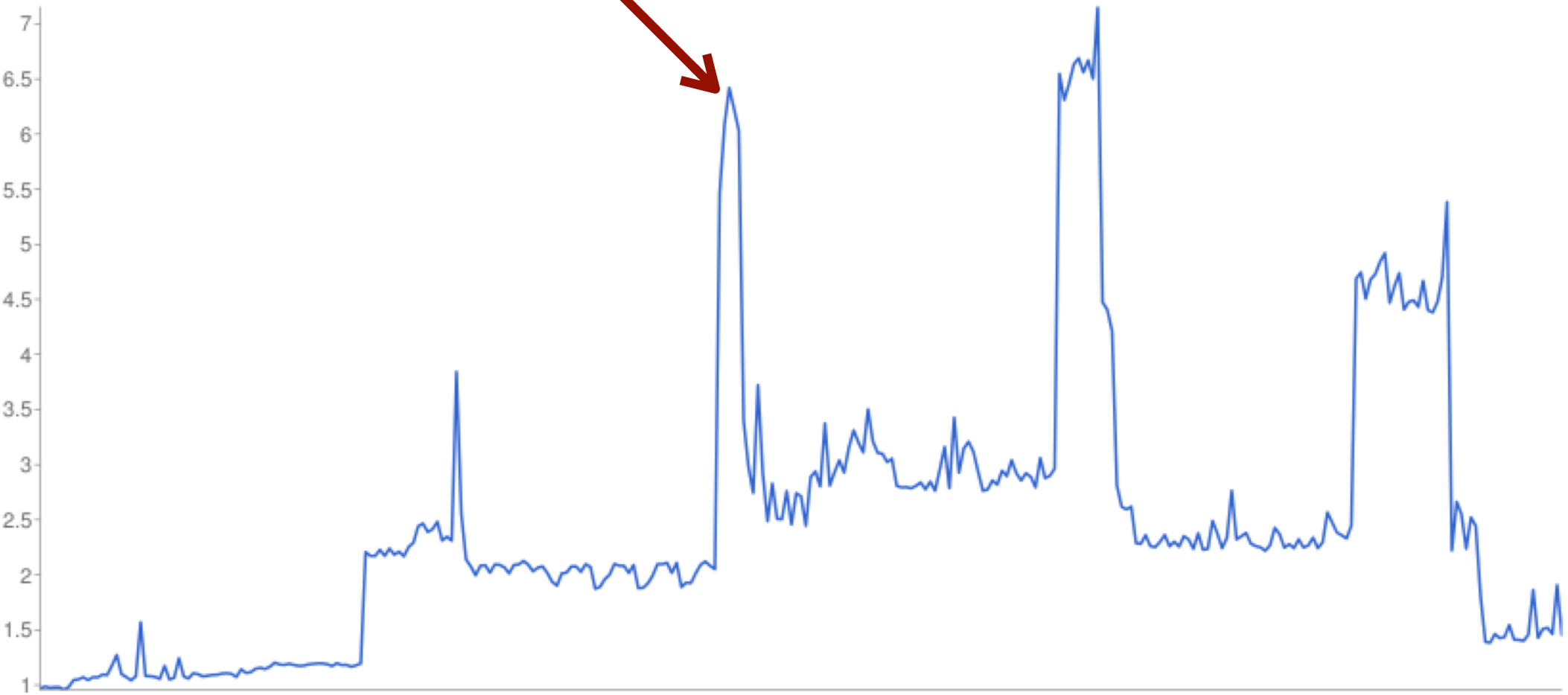
4777 LOC in C ->
3399 LOC in Rust =
71%

Performance

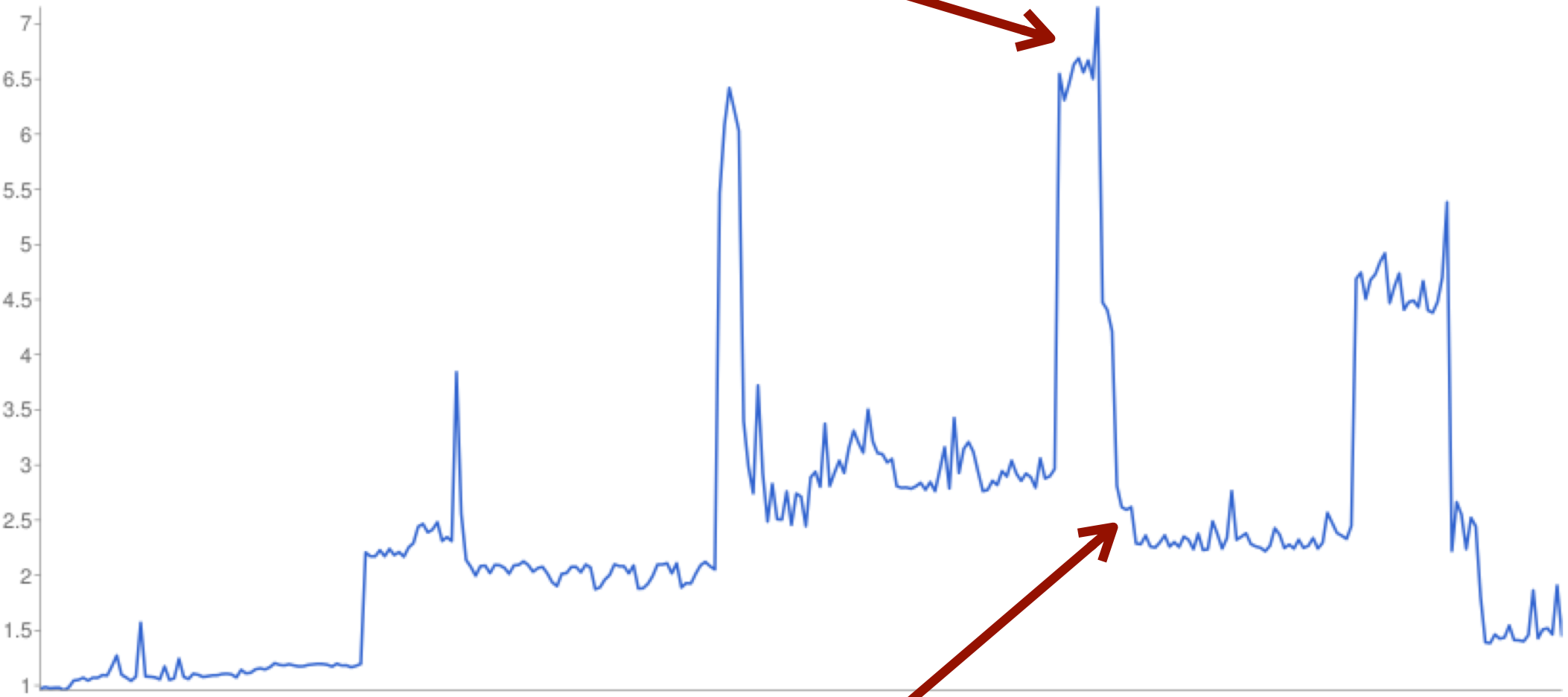

7
6.5
6
5.5
5
4.5
4
3.5
3
2.5
2
1.5
1


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

cloning/creating objects
instead of modifying

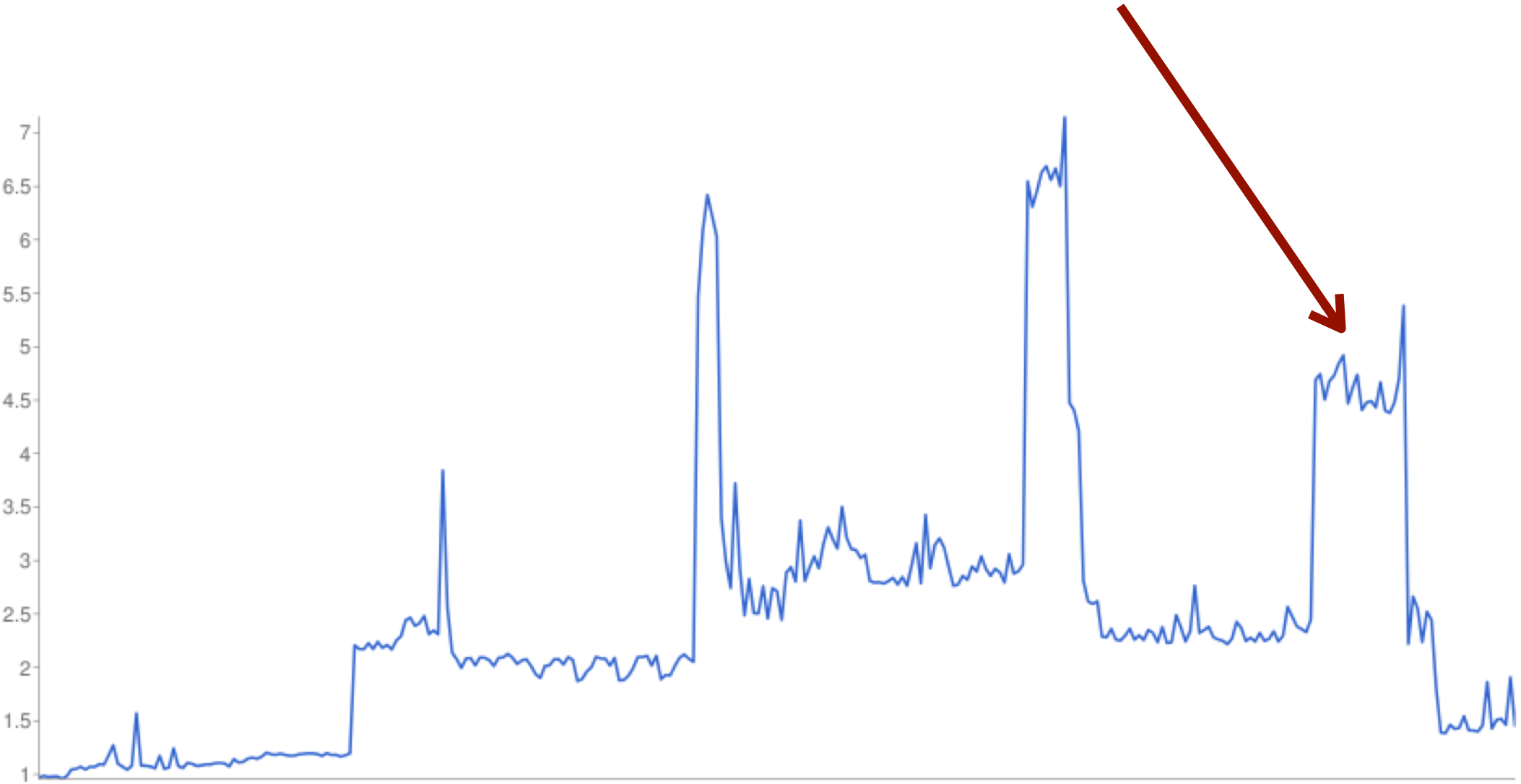


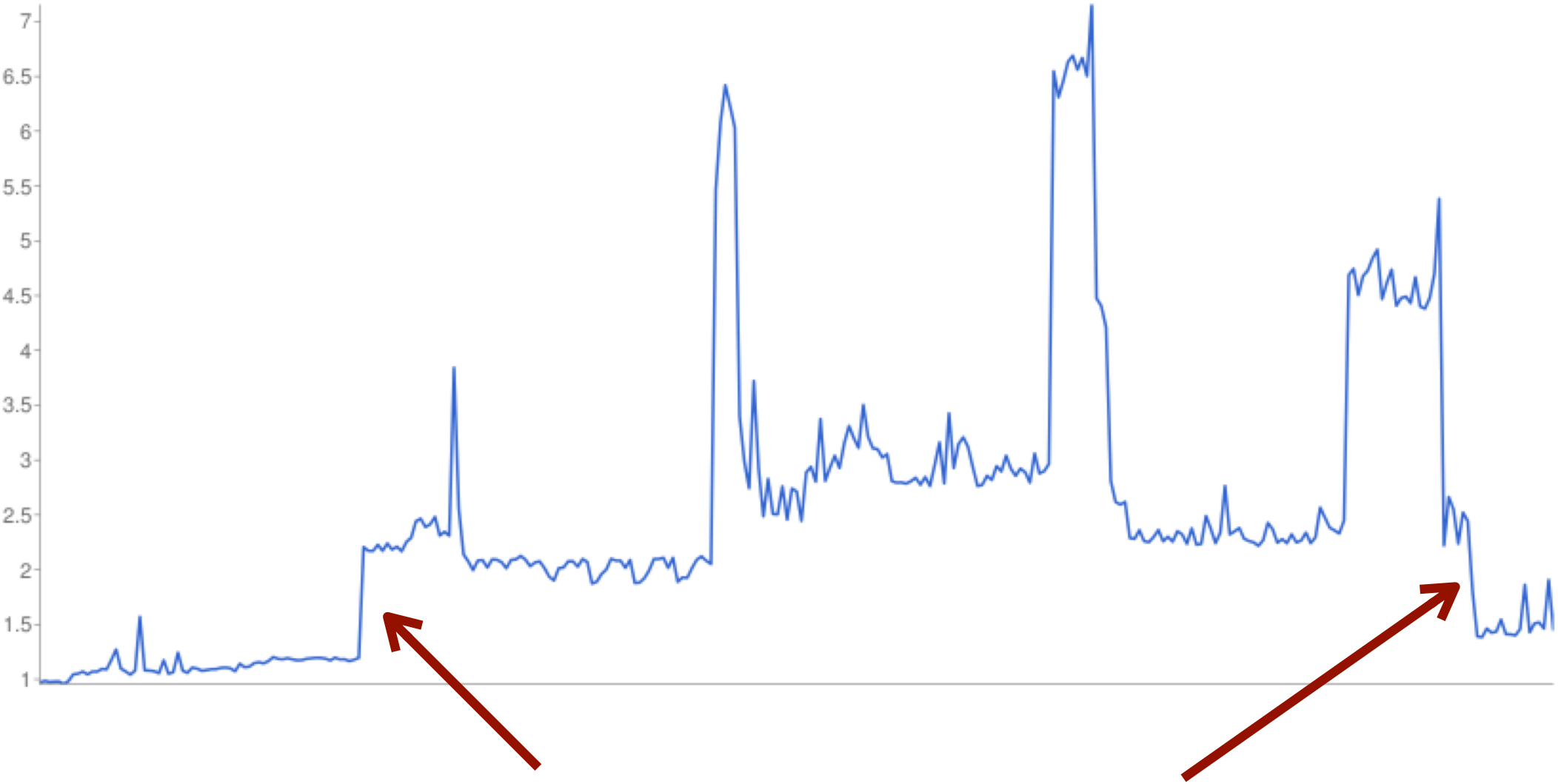
converting lots of ptrs to Vecs

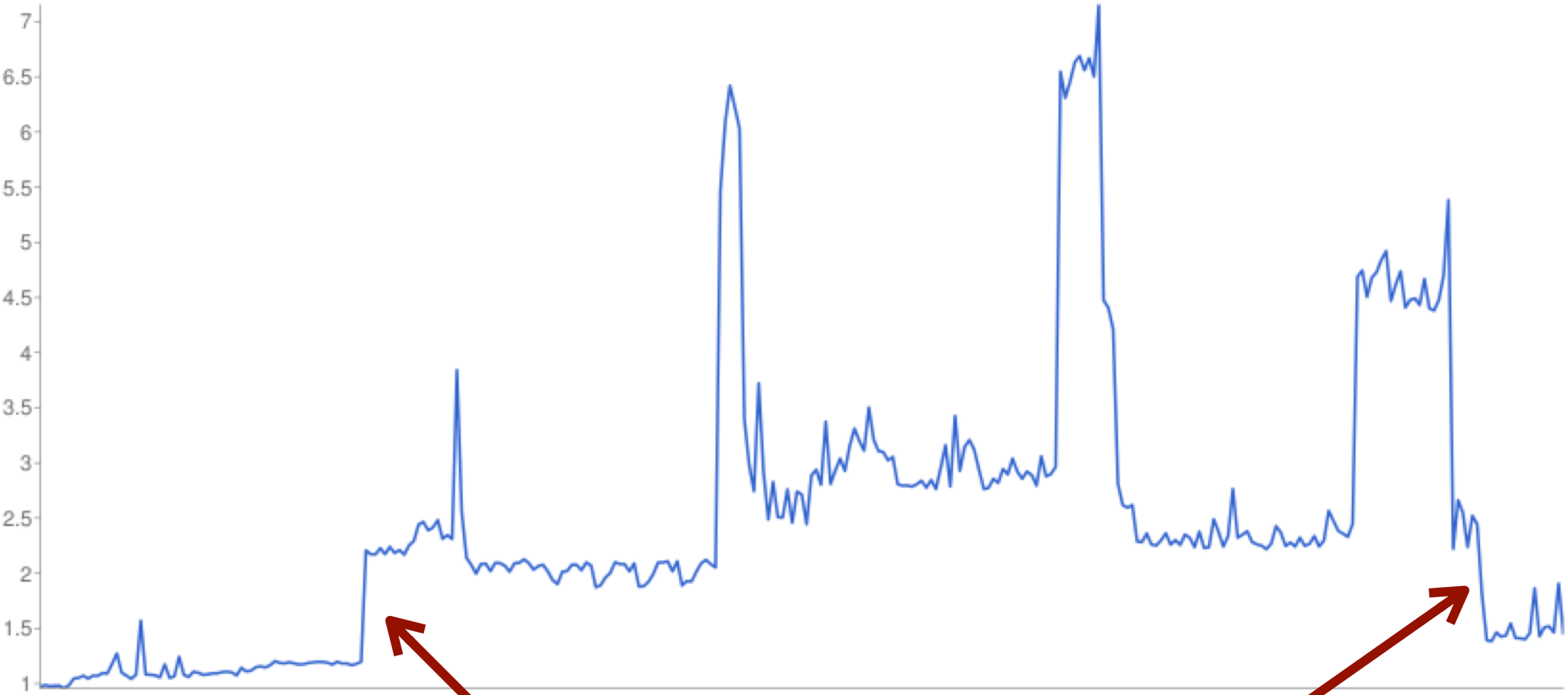


everything converted to Vecs/slices

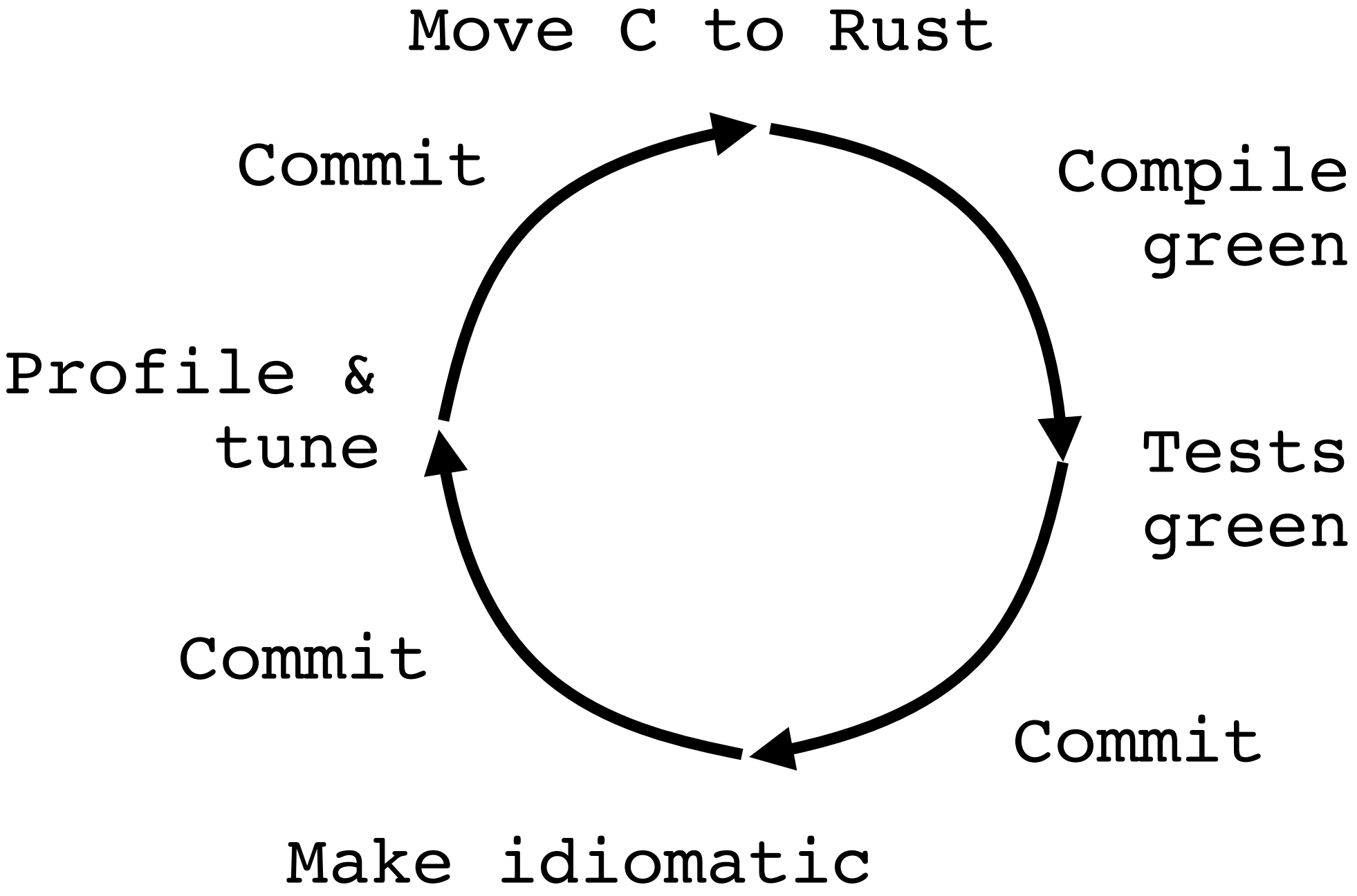
copying lots of doubles




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



"caching" to local vars :(


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.
- 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