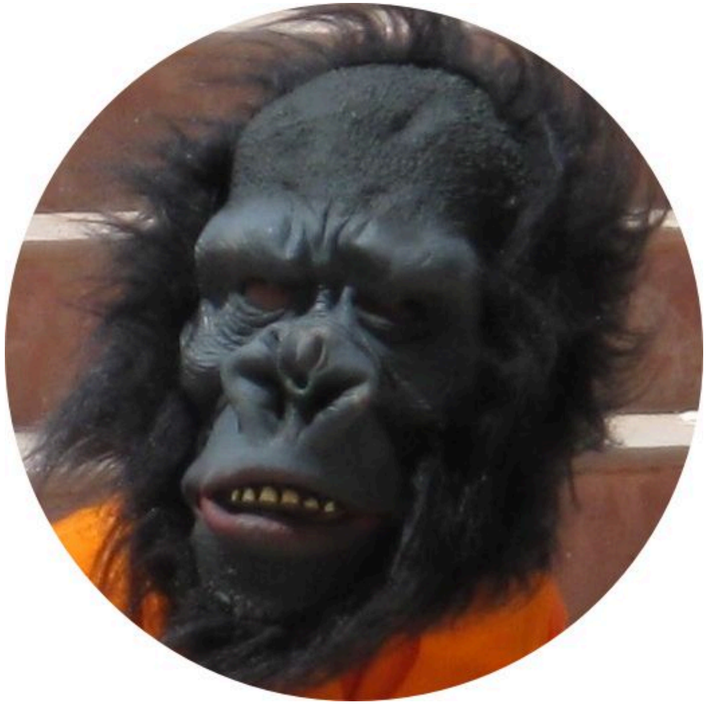
# ETERNAL SUNSHINE OF THE RUSTFMT'ED MIND



@nrc

ncameron.org

ncameron.org

# WHY FORMAT?

# WHY FORMAT?

Developer

# WHY FORMAT?

Developer

Project

# WHY FORMAT?

Developer

Project

Ecosystem

# FORMATTING IS HARD

```
fn foo() {}
```

```
fn foo() {
   ...
}
```

```
fn foo();
```

```
fn foo() -> String {
   ...
}
```

```
fn foo() -> Result<&'_ str, some::Err<String>> {
   ...
}
```

```
pub fn foo() -> String {
    ...
}
```

```
pub(super) fn foo() -> String {
    ...
}
```

```
#[inline]
fn foo() -> String {
   ...
}
```

```
fn foo(x: String) -> String {
   ...
}
```

```
fn foo(
  &self,
  x: String,
  y: String,
  z: String,
) -> String {
  ...
}
```

```
fn foo(
  Foo { mut x, y, .. }: Foo<String>,
) -> String {
  ...
}
```

```
fn foo<T>(x: T) -> String {
   ...
}
```

```
fn foo<T: Into<String>>(x: T) -> String {
   ...
}
```

```
fn foo<
  T: for<'a> Fn(&'a (u8, u16)) -> &'a u8,
>(x: T) -> String {
  ...
}
```

```
fn foo(x: impl Into<String>) -> String {
   ...
}
```

```
fn foo<'a>(x: &'a str) -> String {
   ...
}
```

```
fn foo<const N: usize>(x: [i32; N]) -> String
{
   ...
}
```

```
fn foo<'a, T, U>(x: &'a (T, U)) -> String {
   ...
}
```

```
fn foo<
   'a: 'static,
   T: Into<String>,
   U: Iterator<Item = String>,
>(x: &'a (T, U)) -> String {
   ...
}
```

```
fn foo<'a, T, U>(x: &'a (T, U)) -> String
where
   'a: 'static,
   T: Into<String>,
   U: Iterator<Item = String>,
{
   ...
}
```

```
fn  mod  struct
enum  trait
impl  type  use
const  static
macro_rules!
...
```

```
self.result.push_str(&self.text[start..end]);
```

```
self.result.push_str(&self.text[start..end]);
    ^^      ^^         ^^^   ^^   ^^     ^ ^  ^^^^
```

```
self.result.push_str(&self.text[start..end]);
    ^^      ^^        ^^^   ^^   ^^     ^ ^  ^^^^
```

131,072

```
self.result.push_str(
  &self.text[start..end],
);
```

```
self.
  result.
  push_str(&self.text[start..end]);
```

```
self
   .result
   .push_str(&self.text[start..end]);
```

```
self
  .result
  .push_str(
    &self.text[start..end],
  );
```

```
self
  .result
  .push_str(
    &self.text[
      start..end
    ],
  );
```

```
self.result.push_str(&self
  .text[start..end]);
```

```
fn foo() {
    println!("hello!");
}


fn foo() {
    println!("hello!");
}
```

```
fn foo() {
....println!("hello!");
}
```

```
fn foo() {
 ⟶ println!("hello!");
}
```

# SEMANTICS PRESERVATION

```
foo(a_long_arg, another_long_arg);
```

```
foo(
  a_long_arg,
  another_long_arg
);
```

```
foo(
  a_long_arg,
  another_long_arg,
);
```

```
foo(a_long_arg, another_long_arg,);
```

```
(42,)
```

```
a_macro!(42,)
```

```
use::{b, a, c};
```

```
use::{a, b, c};
```

```
match {
   Some(s) => { s }
   None => { panic!() }
}
```

```
match {
   Some(s) => s,
   None => panic!(),
}
```

```
try!(foo())
```

```
foo()?
```

```
let matrix = [
  0.0,    1.0, 10.1,
  10.0, 100.0,  0.1,
  0.0,    1.0, 10.1,
];
```

```
let matrix = [
  0.0,    1.0, 10.1,
  10.0, 100.0,  0.1,
  0.0,    1.0, 10.1,
];
```

```
let matrix = [
  0.0, 1.0, 10.1,
  10.0, 100.0,
  0.1, 0.0, 1.0,
  10.1,
];
```

# COMMENTS

```
foo(
  // comments can be anywhere
  a_long_arg,
  another_long_arg
);
```

```
fn foo( // here
   &self /* or here */,
   /* or here */x: String,
   y/* or here */: String,
   z:/* or here */ String,
) -> /* or here */ String {
   ...
}
```

```
fn foo( // here
  &self /* or here */,
  /* or here */x: String,
  y/* or here */: String,
  z:/* or here */ String,
) -> /* or here */ String {
  ...
}
```

```
fn foo( // here &self,
  /* or here */x: String,
  y/* or here */: String,
  z:/* or here */ String,
) -> /* or here */ String {
  ...
}
```

```
foo(
   a, // comment about a
   b,
);
```

```
foo(
   a, // comment about a
   b,
);
```

```
foo(
   a,
   // comment about a?
   b,
);
```

# SUCCESS

# SUCCESS

**Success requires whole-ecosystem adoption**

# TWO STORIES

Technical

Social

# THE STYLE RFC PROCESS

# STYLE RFC PROCESS

Seperate specification from implementation

# STYLE RFC PROCESS

Style RFCs

Style team

Well-defined

# STYLE RFC PROCESS

Open

Consensus-based

Thorough

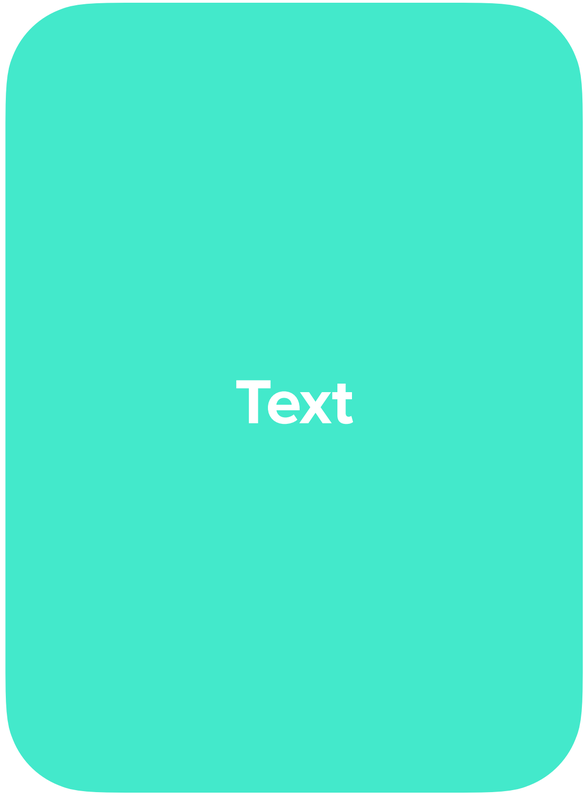
# FORMATTING

# String -> IR -> String

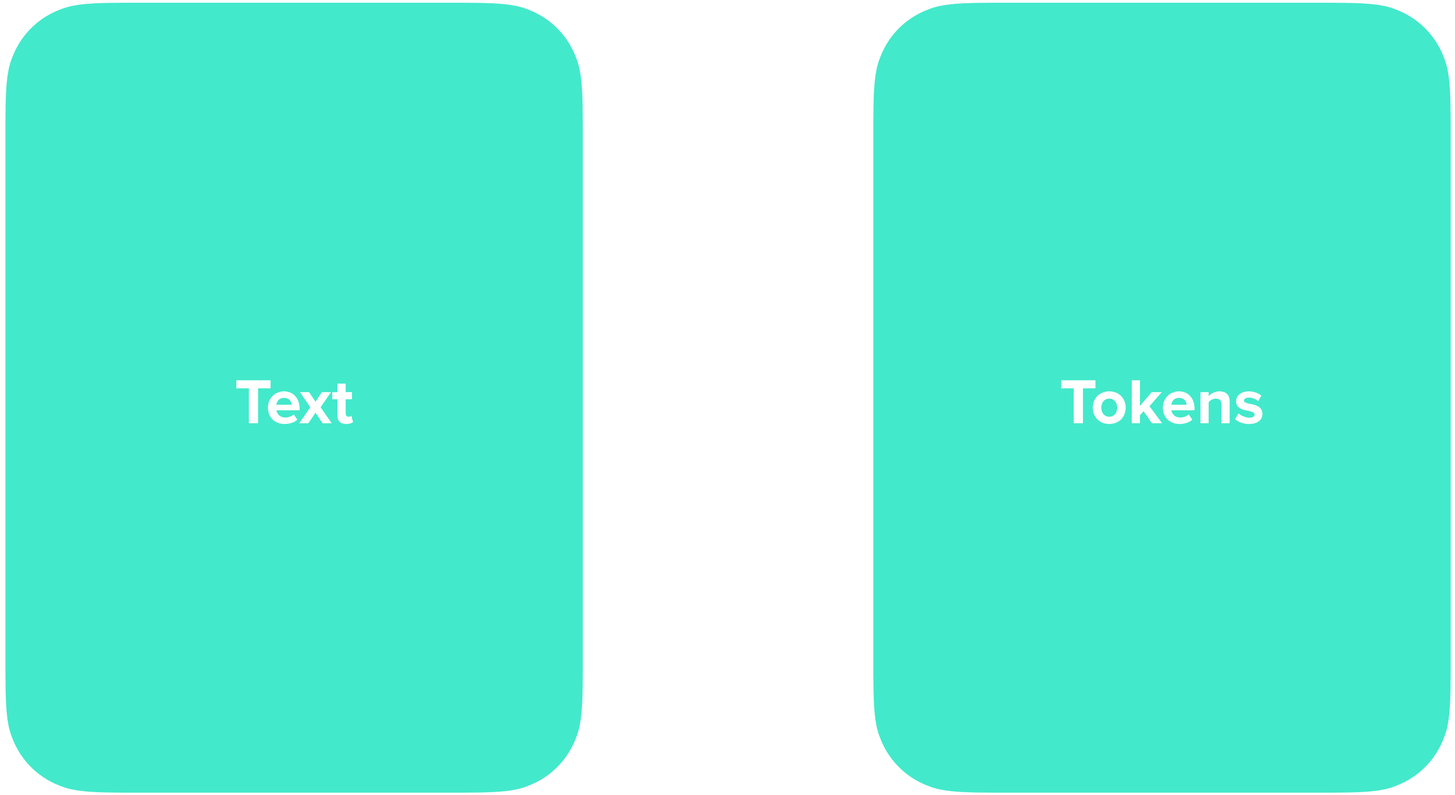
String -> IR -> String

String -> IR -> String

# REPRESENTATION



# REPRESENTATION



# TOKENS

```
foo(a, b);
```

# TOKENS

```
let foo =
  thing1 + thing2;
```

```
let foo = thing1
  + thing2;
```

ncameron.org

# REPRESENTATION

Text

Tokens

AST

ncameron.org

# AST

```
// comments
```

# REPRESENTATION

Semantics

Formatting

# OUTPUT

- Walk the IR
- Build up context
- Follow formatting rules
- Try not to backtrack

# RUSTFMT

# REQUIREMENTS

Configurability

Flexibility

Stability

# IMPLEMENTATION

rustc's AST

Low abstraction

# IMPLEMENTATION

```
fn foo(&self,
       a: String,
       b: i32) -> u32
{
   ...
}
```

```
fn foo(
   &self,
   a: String,
   b: i32
) -> u32 {
   ...
}
```

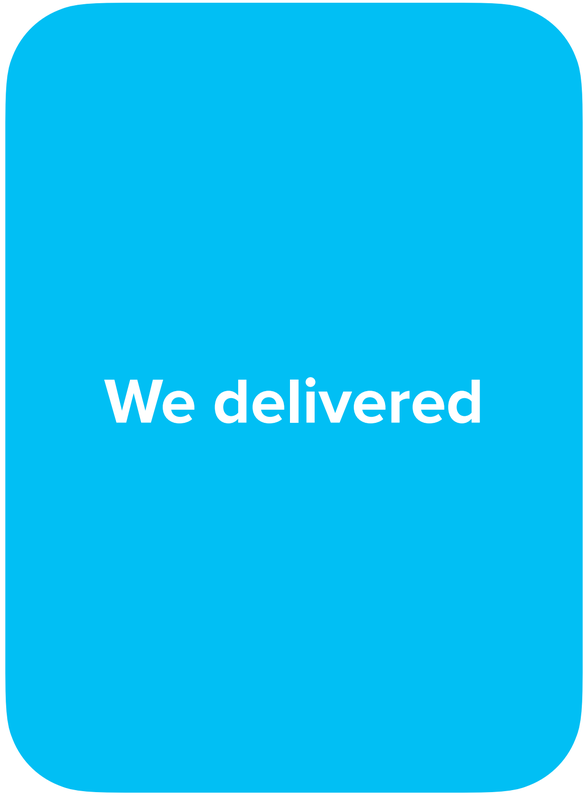
# IMPLEMENTATION

rustc's AST

Low abstraction

# REFLECTION

# RUSTFMT



# RUSTFMT

Output is good

Versatile

# RUSTFMT

Slow

Maintenance

Macros

ncameron.org

# RUSTFMT

Different context

⇒

Different requirements

⇒

Different design

# RUSTFMT

Custom representation

# RUSTFMT

Custom representation

No options

# RUSTFMT

Custom representation

No options

Tolerant

ncameron.org

# STYLE RFC PROCESS

Output is good

Time and energy

# ADOPTION

Widespread

Defaults are common

People seem happy?

ncameron.org

# ADOPTION

# ADOPTION

Respected community norms

# ADOPTION

Respected community norms

Built trust

# ADOPTION

Respected community norms

Built trust

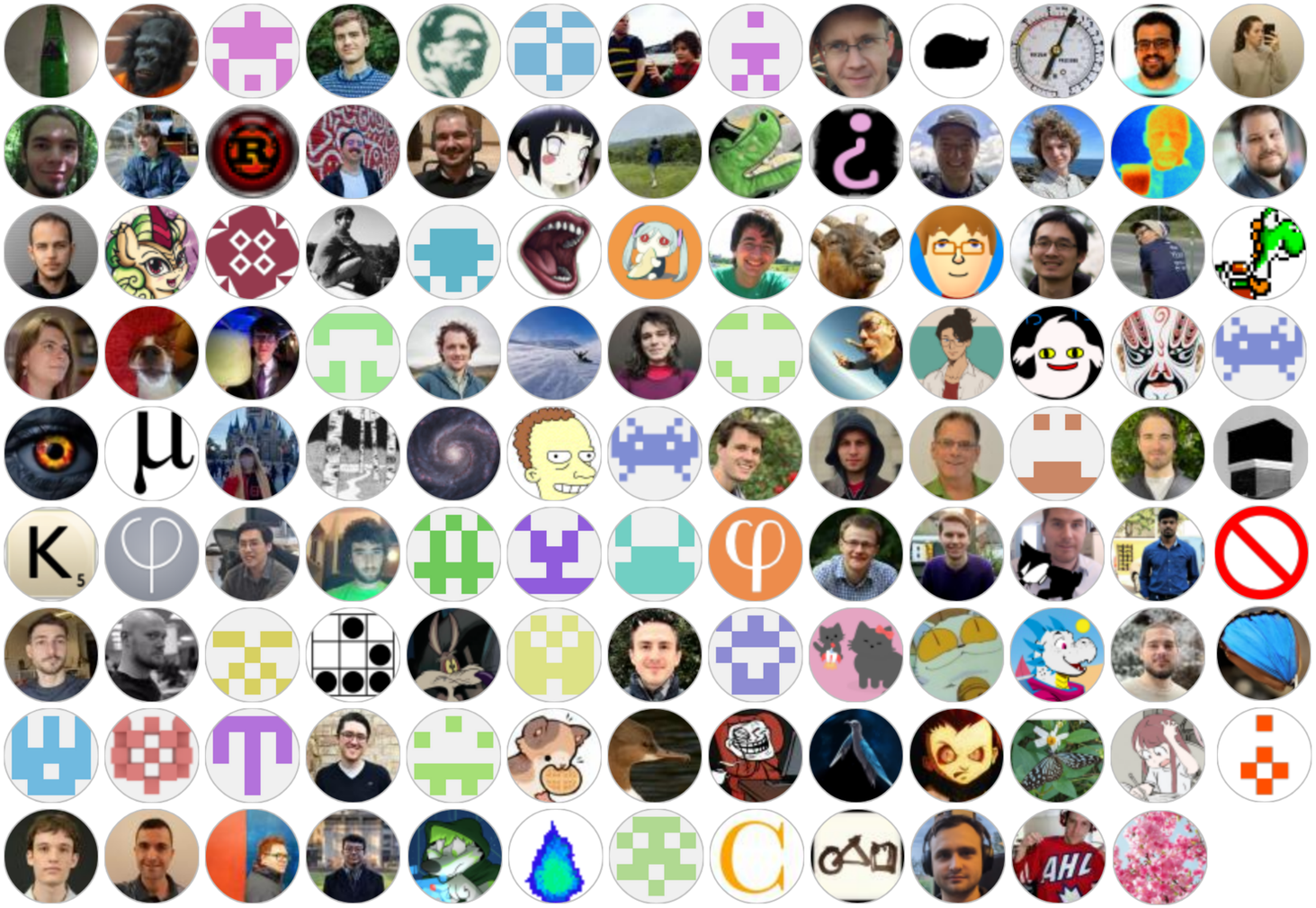
Make it easy

ncameron.org

# THE END

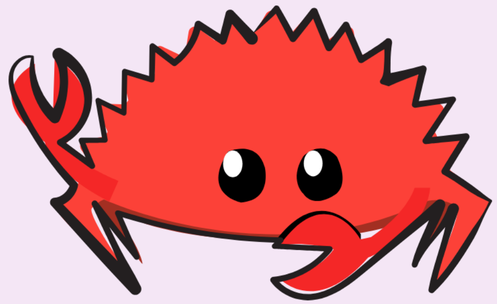
ncameron.org

# THANK YOU!




ncameron.org

RUSTFMT CONF
2024 10—13 Sep
Montreal



ncameron.org