Making Rust Delightful

Nicholas Cameron
RustCon Asia 2019

@nick_r_cameron
@nrc

https://insights.stackoverflow.com/survey/2019#technology-_-most-loved-dreaded-and-wanted-languages

Practical

1.0

Ergonomics Initiative

https://blog.rust-lang.org/2017/03/02/lang-ergonomics.html

Polish

Ergonomics

Tools

Libraries

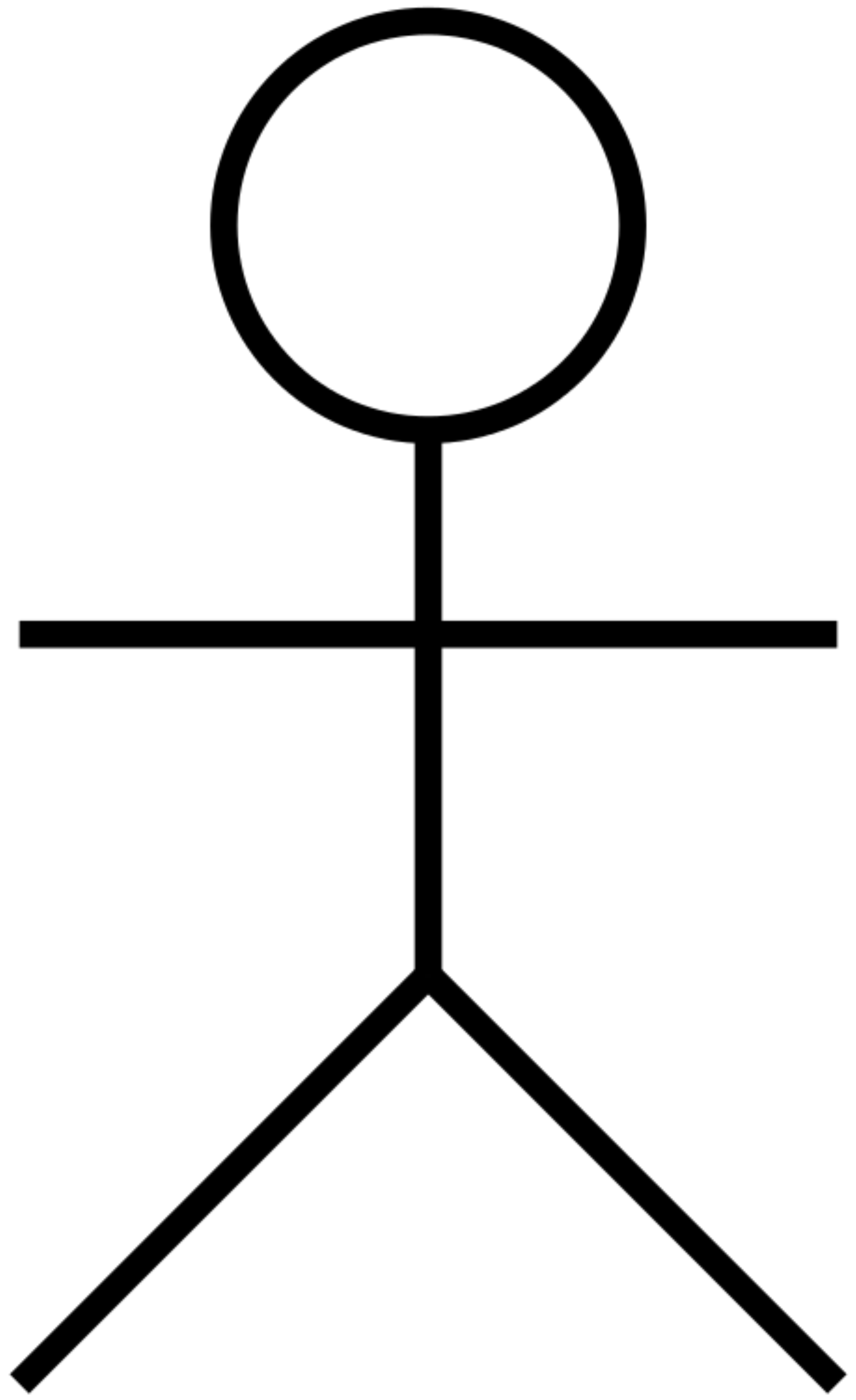
Language

Ergonomics

Ergonomics is difficult!

Context

?



Empathy

Principles and heuristics

Heuristic

The right choice should be the easy choice

Heuristic

Important things should be explicit

Corollary

Avoid boilerplate

What’s important?

Tools

Principle

Put the user first

Principle

Consistency

Principle

Convention over configuration

```
> rustup component add rustfmt
> rustup component add rustfmt
> cargo fmt
> rustup component add rustfmt
> cargo fmt
> cargo fmt -p my-package
```

Principle

Tools can teach the user

```
file.rs:3:9: 3:11 error: mismatched types:
 expected `&usize`,
    found `&u32`
(expected usize,
    found u32) [E0308]
file.rs:3     foo(&x);
                  ^~
error: aborting due to previous error
```

```
error[E0308]: mismatched types
 --> file.rs:3:9
  |
3 |     foo(&x);
  |         ^^ expected usize, found u32
  |
  = note: expected type `&usize`
             found type `&u32`

error: aborting due to previous error
```

Principle

Consistency

Principle

Minimise duplication

`HashMap::find_`

```
fn foo(h: HashMap<String, u8>) {
  h.get(&String::new()); // &String


}
```

```
fn foo(h: HashMap<String, u8>) {
  h.get(&String::new());
  h.get(“”); // &str

}
```

T &T

~~T &T~~

String &str

```
fn foo(h: HashMap<String, u8>) {
  h.get(&String::new());
  h.get(“foo”);
  h.insert(“bar”.to_owned(), 42);
}
```

```
impl<K: Hash, V> HashMap<K, V> {

}
```

```
impl<K: Hash, V> HashMap<K, V> {
  fn get<Q: Hash>(&self, k: &Q) -> Option<&V>
    where K: Borrow<Q>;
}
```

```
impl<K: Hash, V> HashMap<K, V> {
  fn get<Q: Hash>(&self, k: &Q) -> Option<&V>
    where K: Borrow<Q>;
}
```

```
fn foo(h: HashMap<String, u8>) {
  h.get(&String::new());
  h.get(“foo”);
  h.insert(“bar”.to_owned(), 42);
}
```

```
fn foo(h: HashMap<String, u8>) {
  h.get(String::new());
  h.get(“foo”);
  h.insert(“bar”.to_owned(), 42);
}

fn foo(h: HashMap<Vec<u8>, u8>) {
  h.get(vec![]);
  h.get(&[1, 2]);
  h.insert(vec![1, 3], 42);
}
```

Heuristic

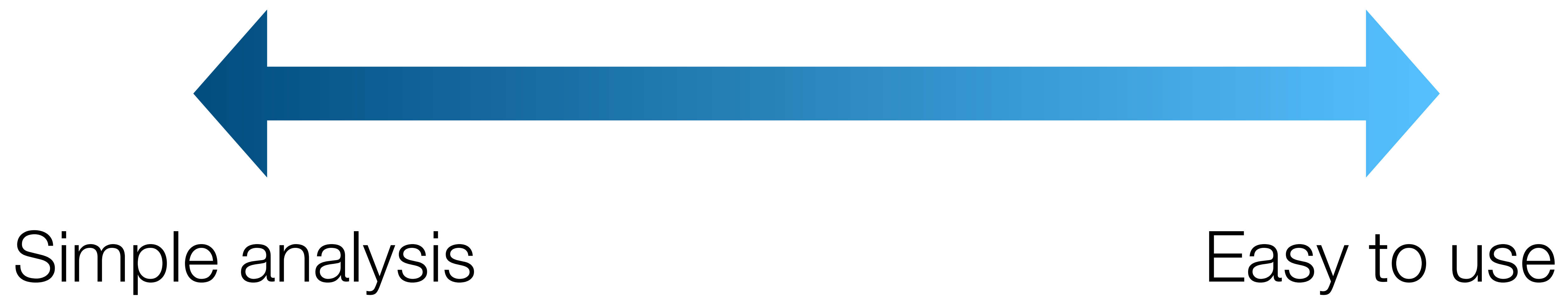
Favour client simplicity

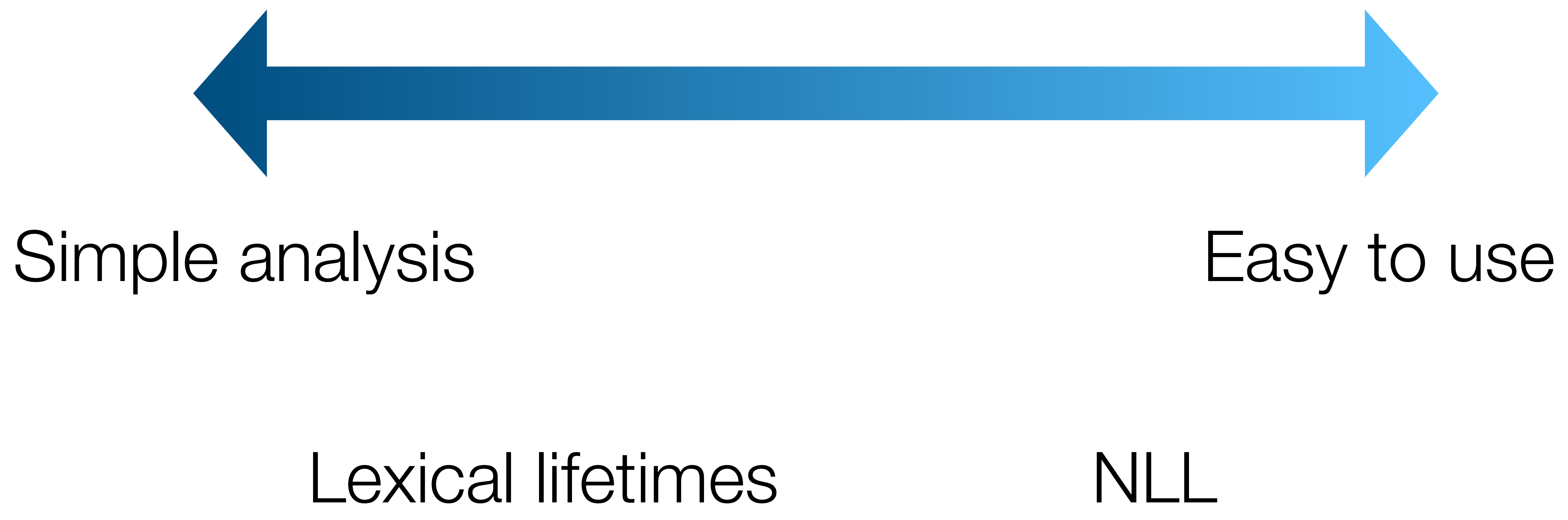
{ }

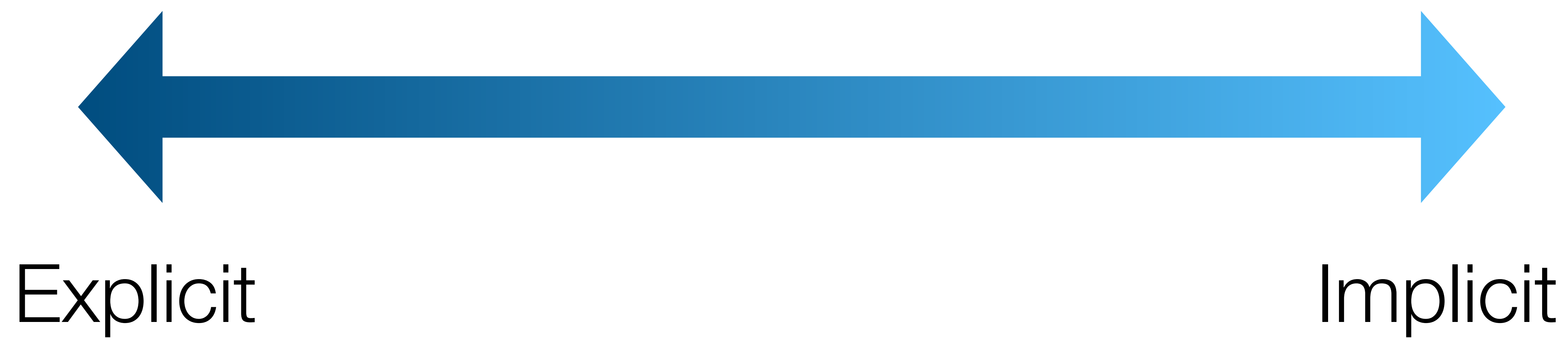
General
Specific

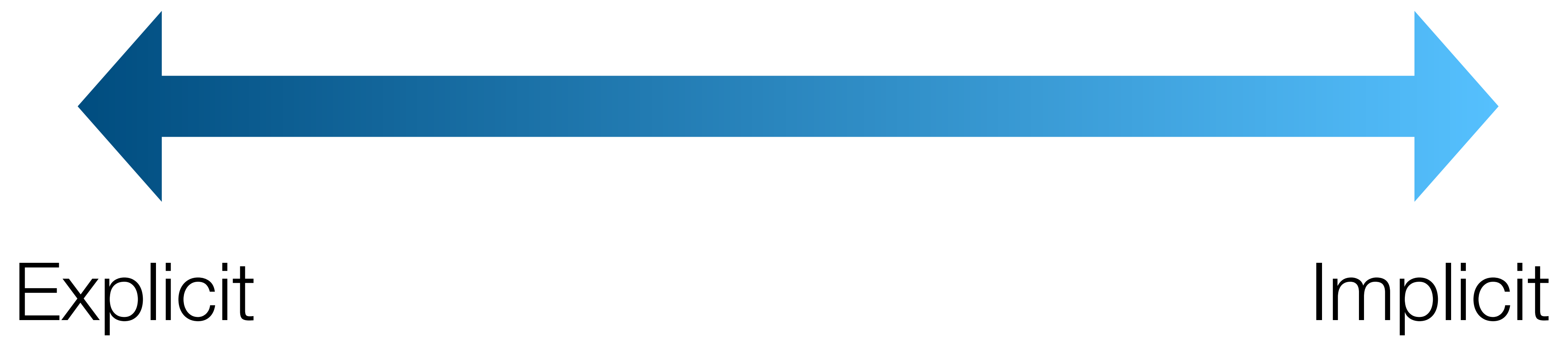
General   Specific

`do`   `?`   `throw`


Simple analysis
Easy to use


Simple analysis
Easy to use
Lexical lifetimes
NLL


Explicit
Implicit



```
move x          &x          x
```

```
fn foo(x: &Option<…>) {
  match x {
    &Some(ref x) => …,
    None => …,
  }
}
```

```
fn foo(x: &Option<…>) {
  match x {
    &Some(ref x) => …,
    None => …,
  }
}
```

```
fn foo(x: &Option<…>) {
  match x {
    Some(x) => …,
    None => …,
  }
}
```

Stability without stagnation

```
fn foo() -> Result<(), IoError> {
  …
  Ok(())
}
```

```
bar(&42);
```

```
await baz();
```

```
    |
    = note: the method `first` exists but the following trait bounds were no
FilteredQuerySource<schema::__diesel_infer_schema::infer_projects::projects:
diesel::expression::predicates::Eq<schema::__diesel_infer_schema::infer_proj
ssion::bound::Bound<diesel::types::Text, &str>>, diesel::expression::predica
_releases::releases::columns::visible, diesel::expression::bound::Bound<dies
ilder::AsQuery`, `diesel::query_source::filter::FilteredQuerySource<schema::
cts::table, diesel::expression::predicates::And<diesel::expression::predicat
projects::projects::columns::name, diesel::expression::bound::Bound<diesel::
dicates::Eq<schema::__diesel_infer_schema::infer_releases::releases::columns
diesel::types::Bool, bool>>>> : diesel::query_builder::AsQuery`, `diesel::qu
ma::__diesel_infer_schema::infer_projects::projects::table, diesel::expressi
icates::Eq<schema::__diesel_infer_schema::infer_projects::projects::columns:
el::types::Text, &str>>, diesel::expression::predicates::Eq<schema::__diesel
umns::visible, diesel::expression::bound::Bound<diesel::types::Bool, bool>>>
::query_source::filter::FilteredQuerySource<schema::__diesel_infer_schema::i
ession::predicates::And<diesel::expression::predicates::Eq<schema::__diesel_
mns::name, diesel::expression::bound::Bound<diesel::types::Text, &str>>, die
esel_infer_schema::infer_releases::releases::columns::visible, diesel::expre
l>>>> : diesel::query_builder::AsQuery`, `&diesel::query_source::filter::Fil
ma::infer_projects::projects::table, diesel::expression::predicates::And<die
esel_infer_schema::infer_projects::projects::columns::name, diesel::expressi
, diesel::expression::predicates::Eq<schema::__diesel_infer_schema::infer_re
expression::bound::Bound<diesel::types::Bool, bool>>>> : diesel::query_build
eryFragment<_>`, `&diesel::query_source::filter::FilteredQuerySource<schema:
ects::table, diesel::expression::predicates::And<diesel::expression::predica
```

Summary

Summary

There is no easy takeaway

Summary

There is no easy takeaway

Intention, effort, empathy

Summary

There is no easy takeaway

Intention, effort, empathy

What is important?

Thanks!

Brian Anderson

Niko Matsakis

YOU!