Async Rust: past, present, and `Future`

Nick Cameron (@nrc)
The Async Working Group

WÜRZBURG
ZEICHENERKLÄ
MÜNC
Exerzierplatz
Ober
Giesenfeld
LANDSHUT
HEILBRONN

`async` and **`await`** are stable

minimal support in **`std`**

the async WG is working…

safety
performance
ergonomics

safety

performance - precise control

ergonomics

zero-cost abstractions

🪄 aversion to magic

provide the fundamentals

```
async
await
```

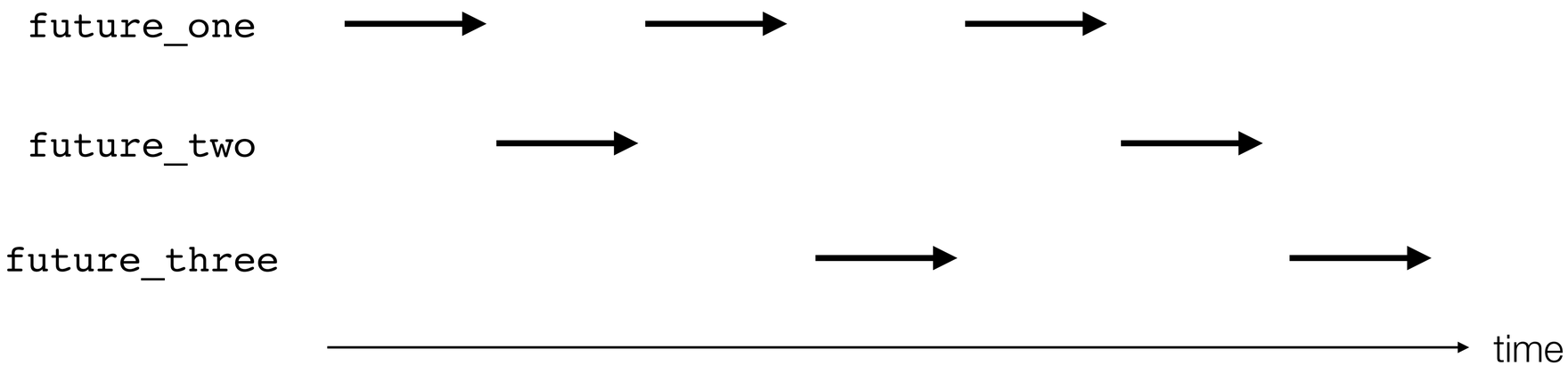
Future

```rust
pub trait Future {
    type Output;

    fn poll(…) -> Poll<Self::Output>;
}

pub enum Poll<T> {
    Ready(T),
    Pending,
}
```

Spannmål
Spannmål
MATVAGN I DALARNA
YHL 581


future_one
future_two
future_three
time

```
async
await
```

```
async fn foo(…) -> String
```

```
async fn foo(…) -> String

fn foo(…) -> impl Future<Output = String>
```

```
trait Foo {
    async fn foo(&self);
}
```

```rust
trait Foo {
    async fn foo(&self);
}

trait Foo {
    fn foo(&self) -> impl Future<()>;
}
```

```
trait Foo {
    async fn [illegible]();
}

trait [illegible]
    fn [illegible](&self) -> impl Future<()>;
}
```

RPITIT

```
trait Foo {
    async fn [illegible]();
}

trait [illegible]
    fn [illegible](&self) -> impl Future<()>;
```

RPITIT

```
trait Foo {
    async fn foo(&self);
}

trait Foo {
    type T: Future<()>;
    fn foo(&self) -> T;
}
```

```
trait Foo {
    async fn foo<U>(&self);
}

trait Foo {
    type T<U>: Future<U>;
    fn foo<U>(&self, …) -> T<U>;
}
```

```
trait Foo {
    async fn [illegible] &self);
}

trait Foo[illegible]
    type [illegible]U>: Future<U>;
    fn foo<U>(&self, …) -> T<U>;
}
```

GATs

```rust
trait Foo {
    async fn foo(&self);
}

&dyn Foo
Box<dyn Foo>
```

```
trait Foo {
    async fn foo(&self);
}

&dyn Foo
Box<dyn Foo>

dyn* Foo
```

```
async
await
```

```
async fn foo(…) -> String

foo().await
```

IntoFuture

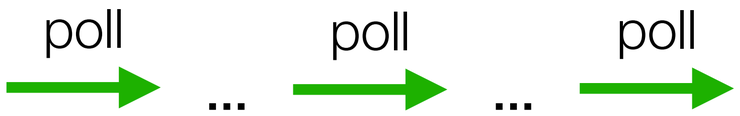
IntoIterator

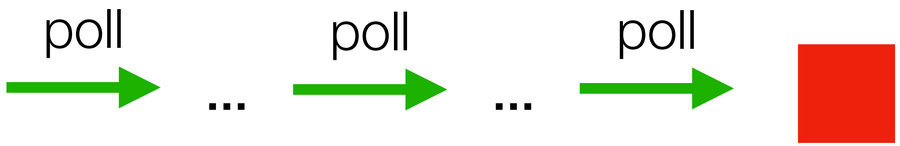
```
for _ in something {
    …
}
```

IntoFuture

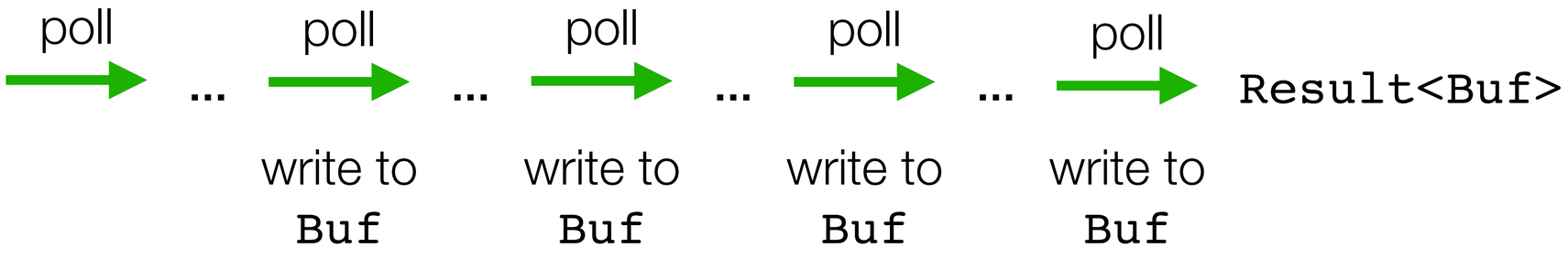
```
builder.property().await
```

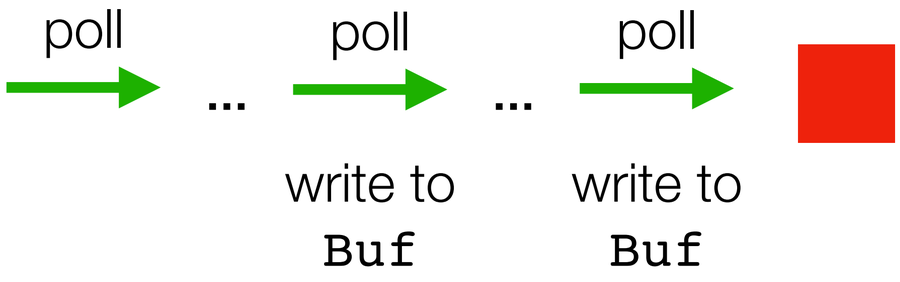
poll
...
poll
...
poll
...
poll
...
poll
Result<…>


poll
...
poll
...
poll


poll
...
poll
...
poll

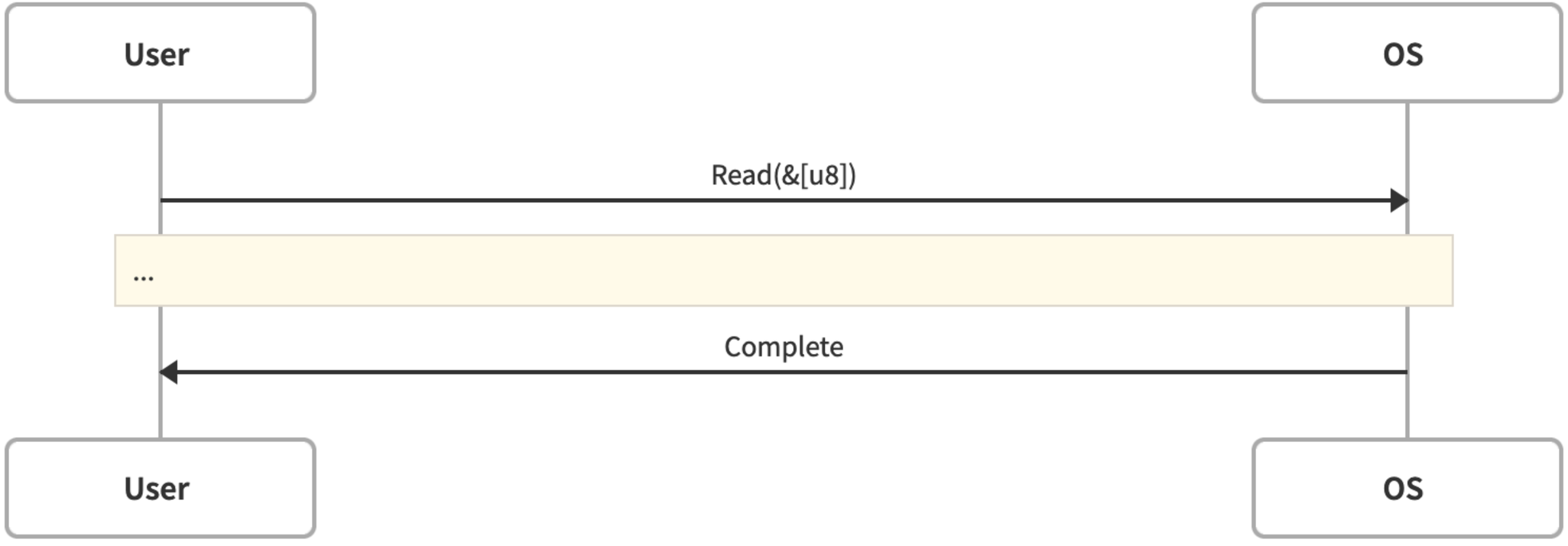
```
select!(
    _ = say_hello() => …,
    _ = say_world() => …,
).await;
```

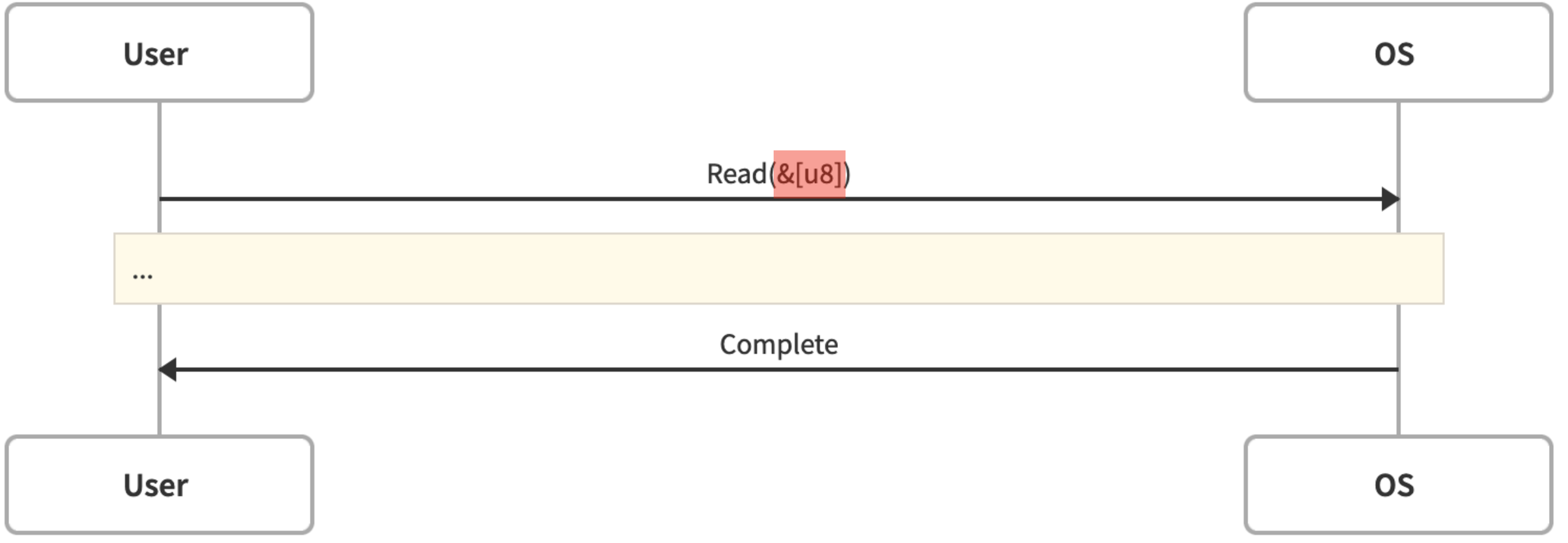

poll
...
poll
write to
Buf
...
poll
write to
Buf
...
poll
write to
Buf
...
poll
write to
Buf
Result<Buf>

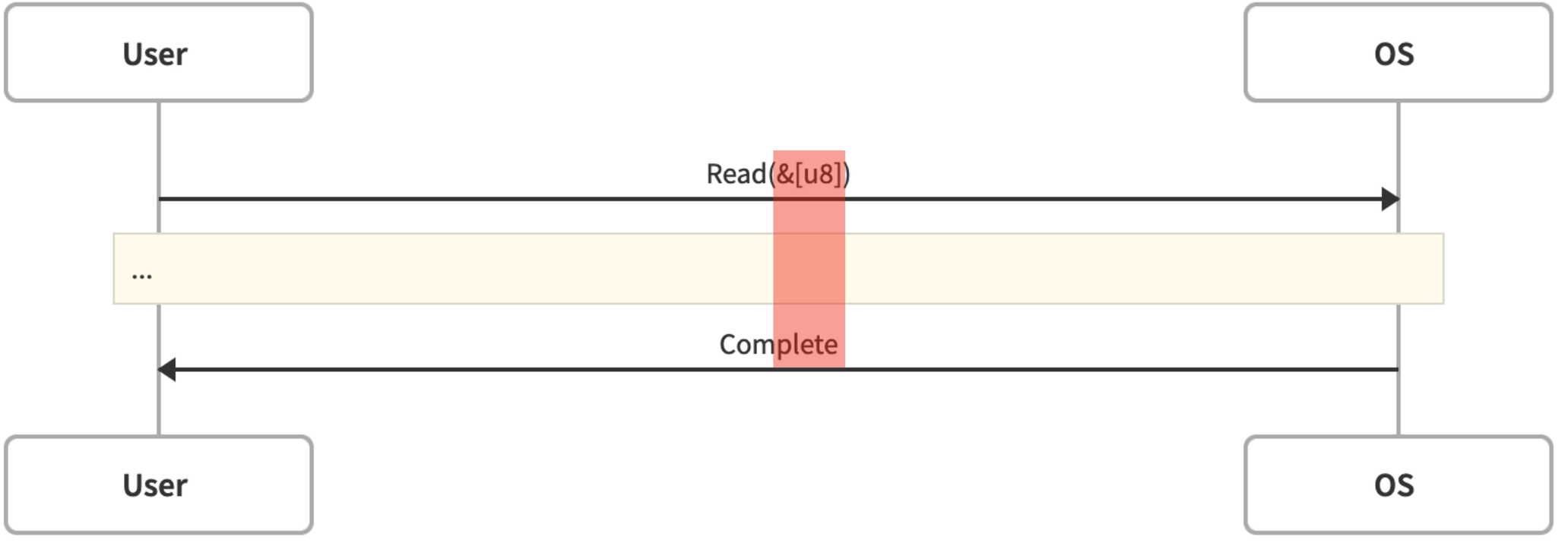

poll
...
poll
...
poll
write to
Buf
write to
Buf

Completion IO

IO_uring
IOCP


User
OS
Read(&[u8])
...
Complete
User
OS


User
OS
Read(&[u8])
...
Complete
User
OS


User
OS
Read(&[u8])
...
Complete
User
OS

```
async fn read_iou(iou: &dyn Read) {
    let mut buf = [0; 1024];
    iou.read(&mut buf).await?;
    …
}
```

```
async fn read_iou(iou: &dyn Read) {
    let mut buf = [0; 1024];
    iou.read(&mut buf).await?;
    …
}

async fn drop_read(…) {
    drop(read_iou(…))
}
```

```
async fn read_iou(iou: &dyn Read) {
    let mut buf = [0; 1024];
    iou.read(&mut buf).await?;
    …
}

async fn multi_read(…) {
    select!(read_iou(…), …)
}
```

cancel the read?

cancel the read?

cancellation is async

async drop?

async drop?

async drop is hard

async drop?

drop is not guaranteed to run

Dear Kate,
Here's to the crazy ones. The misfits.
The troublemakers. The round pegs in the square
holes. The ones who see things differently.
They push the human race forward. And while some
may see them as the crazy ones, we see genius.
Because the people who are crazy enough to think
they can change the world, are the ones who do.
Take care.
John Appleseed

BufRead

OwnedRead

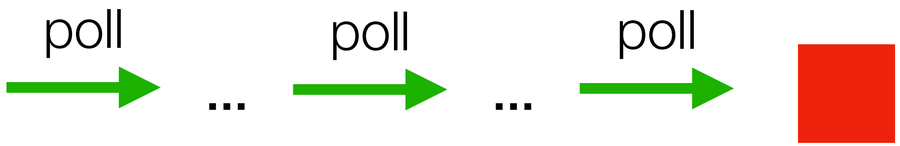
```
pub trait ShinyFuture
```

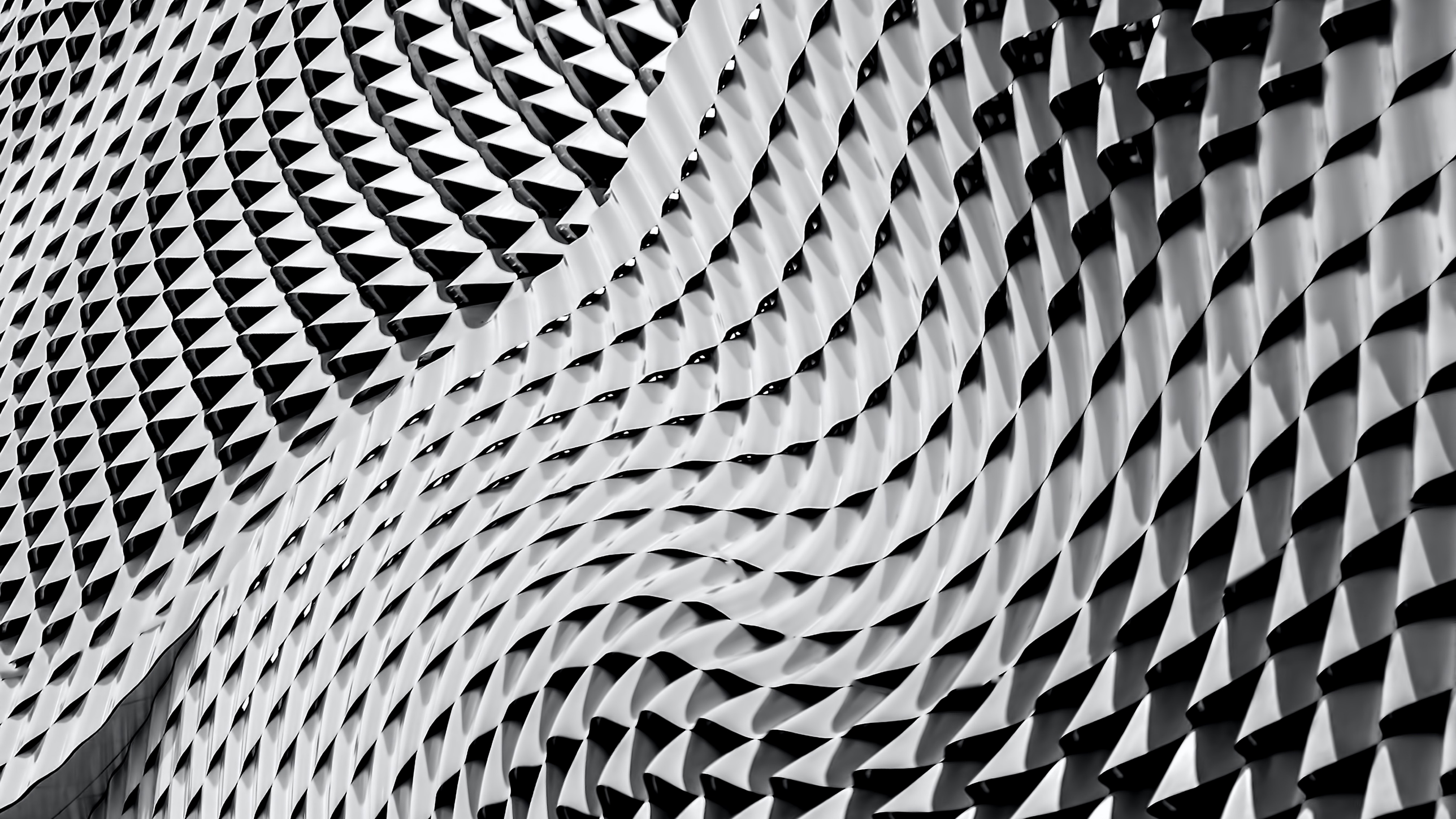
```rust
trait Foo {
    async fn foo(&self);
}
```

```
async |x| { … }
```

```
async fn drop(&mut self) { … }
```

```
async fn main() { … }
```


poll
...
poll
...
poll



help!

https://github.com/rust-lang/wg-async/

https://rust-lang.zulipchat.com/

wg-async

https://github.com/rust-lang/wg-async/

https://rust-lang.zulipchat.com/

wg-async

Nick Cameron

@nick_r_cameron