```
// convenient!
stream.map(logic);



// equivalent to the above.
stream.unary_stream(Pipeline, "Map", |input, output| {

    while let Some((time, data)) = input.next() {
        let session = output.session(&time);
        for datum in data.drain(..) {
            session.give(logic(datum));
        }
    }

});
```

Closures result in code specialization

Ownership of data retained, reused


drain(..) transfers ownership of datum

RAII capability


Required to send

Destructuring ensures valid data

Clarity on resource management: Ownership drives collection/reuse.

Memory / control safety at compile time.
Fewer errors, more predictable behavior.

It’s basically just your code running.

```
// convenient!
stream.map(logic);
```

**Clarity on resource management:
Ownership drives collection/reuse.**

```
// equivalent to the above.
stream.unary_stream(Pipeline, "Map", |input, output| {
    while let Some((time, data)) = input.next() {
        let
        for
            session.give(logic(datum));
        }
    }

});
```

It’s basically just your code running.

**Memory / control safety at compile time.
Fewer errors, more predictable behavior.**