


```
// 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));  
        }  
    }  
});
```

An orange callout box with a dark orange border and a pointed top-left corner, containing the text "Closures result in code specialization".

Closures result in code specialization

Ownership of data retained, reused

An orange callout box with a dark orange border and a pointed top, containing white text.

drain(. .) transfers ownership of datum



RAI capability

An orange callout box with a dark orange border and a pointed top, containing the text "Required to send".

Required to send



Deconstructing 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(data));  
    }  
});
```

It's basically just your code running.

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