Introduction to Rholang

Who are we?

Ahsan Fazal
Full stack developer
ahsanfazal.com

Thomas Schoffelen
Full stack developer
thomasschoffelen.com

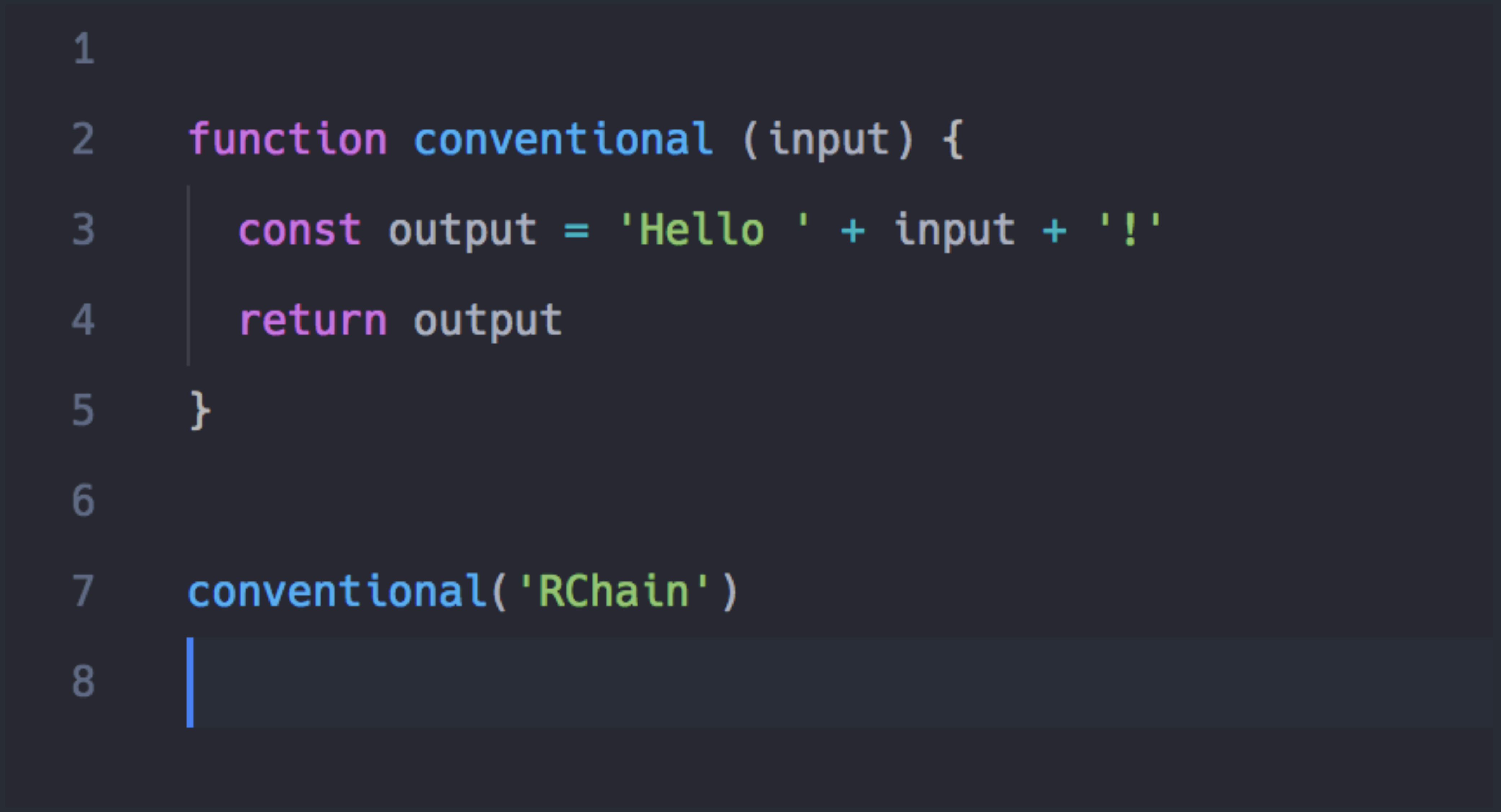
Basics of Rholang Syntax

Conventional programming languages

- define a function
- call that function
- function gets executed

Conventional programming languages

- define a function
- call that function
- function gets executed

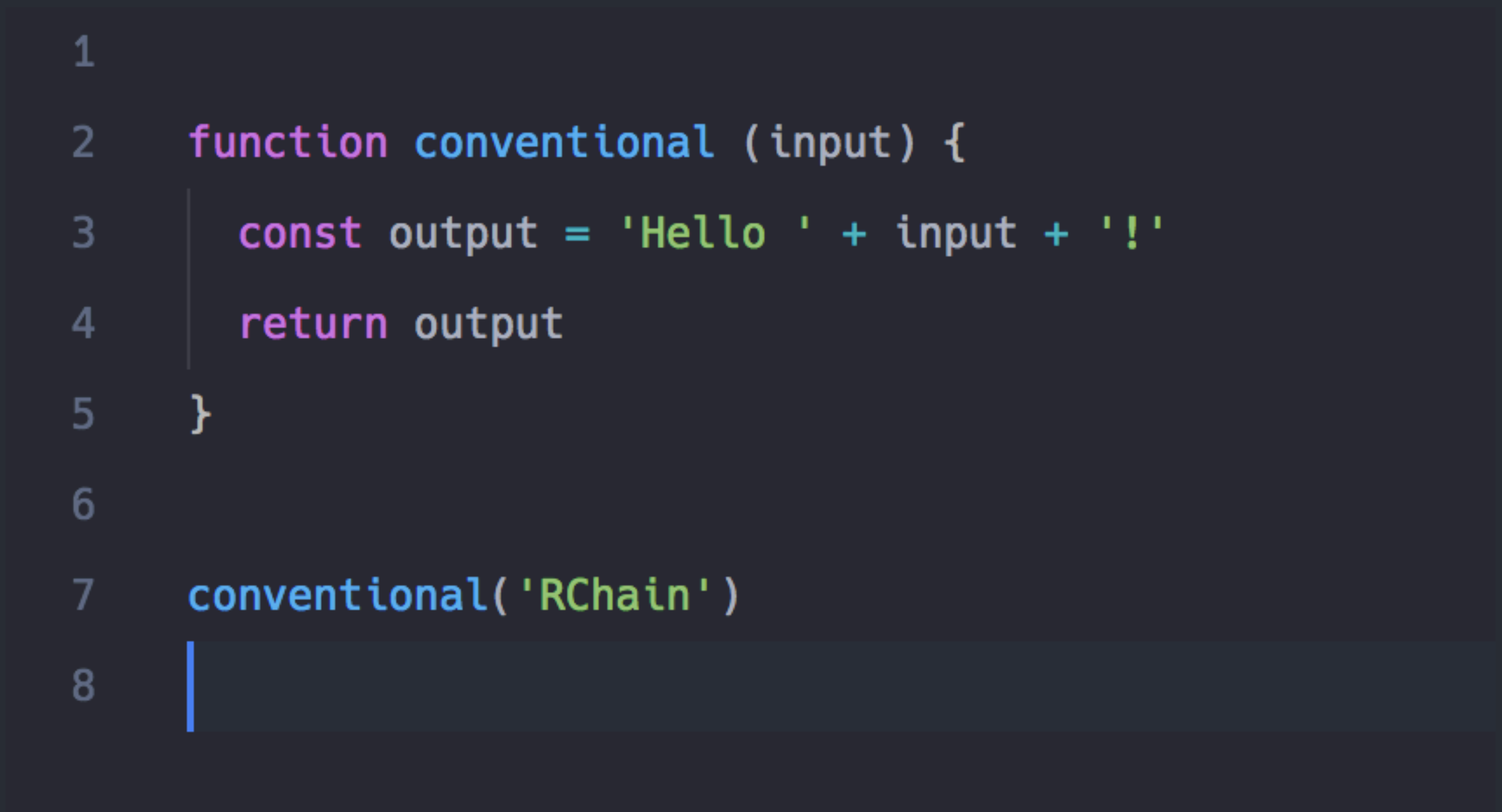


```
function conventional (input) {
  const output = 'Hello ' + input + '!'
  return output
}

conventional('RChain')
```

Conventional programming languages

- define a function
- call that function
- function gets executed



```

function conventional (input) {
  const output = 'Hello ' + input + '!'
  return output
}

conventional('RChain')

```

OUTPUT

Hello RChain!

Concurrency

```
eat sandwich | watch tv
```

Channels (sending and receiving)

`channel!()`

```
for(_ <- channel)
for(_ <= channel)
```

```
channel!() |
for(_ <- channel)
for(_ <- channel) |
channel!()
```

Matching

```
new channel in {
  for(@x, @y <- channel) {
     ...
  } |
  channel!(“hello”, “world”)
}
```

```
new channel in {
  for(@x, @y <- channel) {
     ...
  } |
  channel!(“hello”)
}
```

```
new channel in {
  for(@“hello”, @y <- channel) {
     ...
  } |
  channel!(“hey”, “world”)
}
```

```
new channel in {
  for(@“hello”, @y <- channel) {
    ...
  } |
  channel!(“hello”, “world”)
}
```

Named processes

```
contract stdOut {}
```

```
*stdOut
```

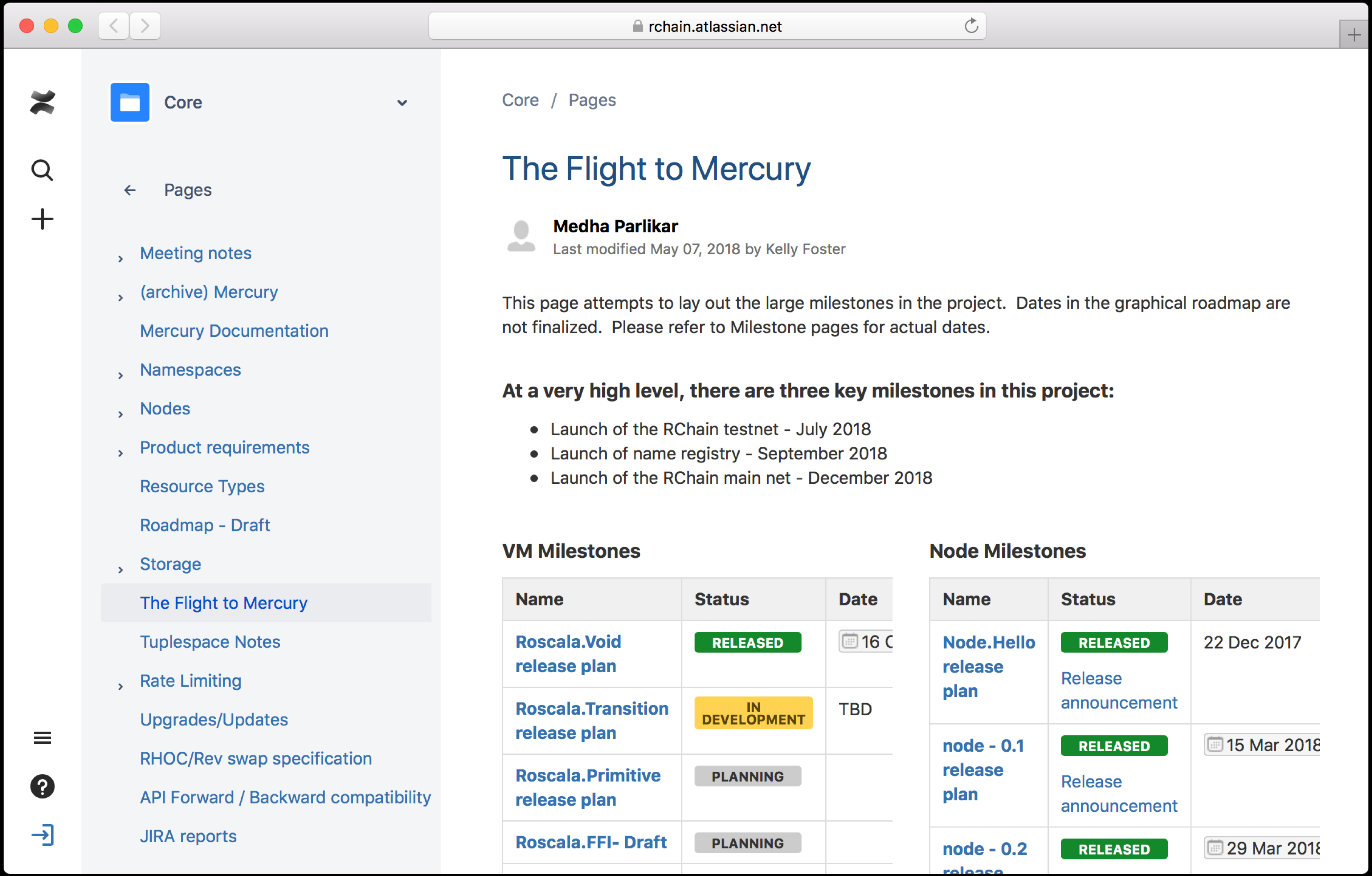
```
channel!(*stdOut)
```

```
@“stdOut”!()
```

Recap

- asynchronous (|)
- send (!) and receive (for loop) on channels
- convert between names and processes
  - @ = process from name
  - * = name from process

Building Smart Contracts Using Rholang



Core / Pages

The Flight to Mercury

Medha Parlikar
Last modified May 07, 2018 by Kelly Foster

This page attempts to lay out the large milestones in the project. Dates in the graphical roadmap are not finalized. Please refer to Milestone pages for actual dates.

At a very high level, there are three key milestones in this project:

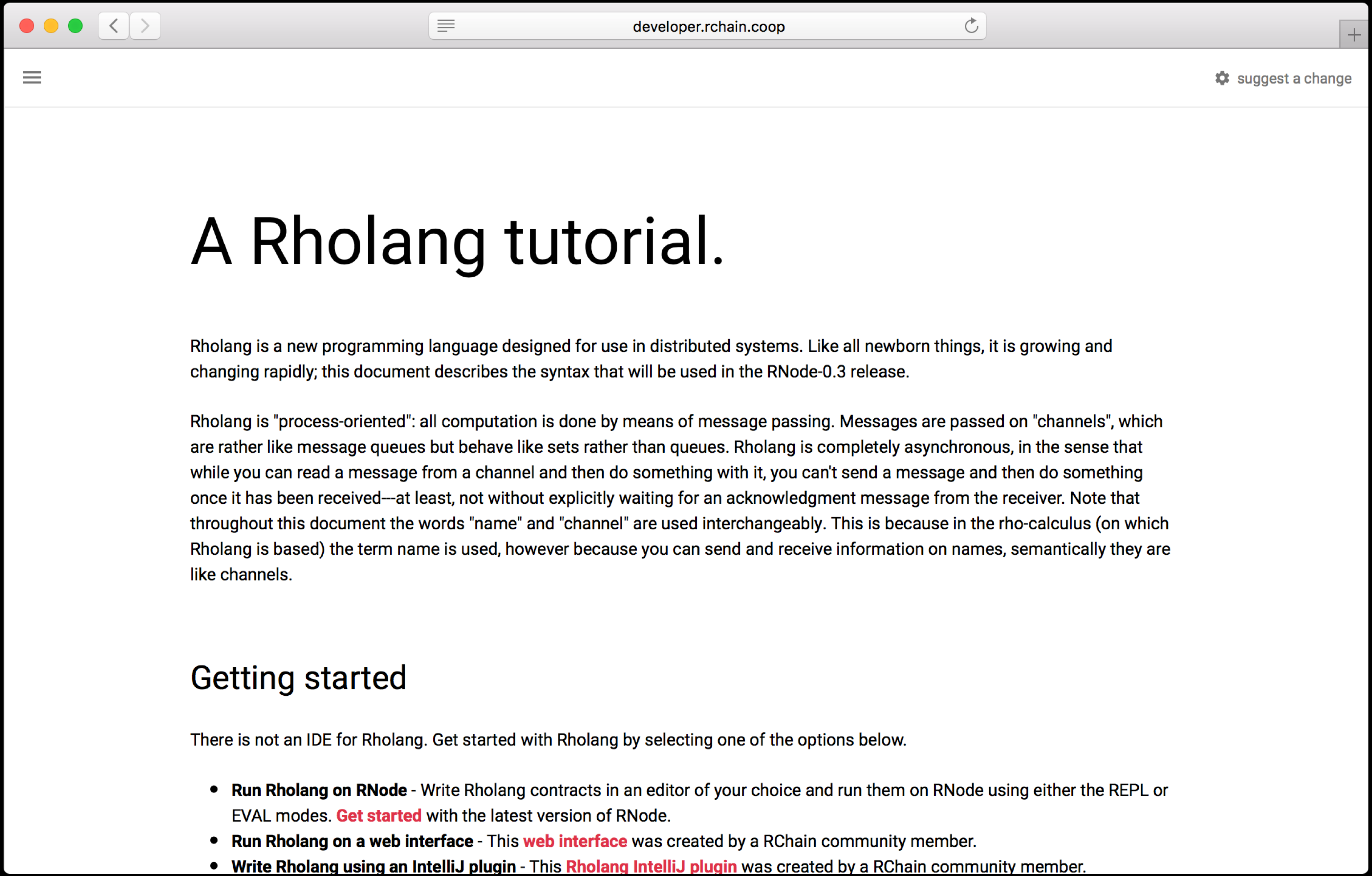
- Launch of the RChain testnet - July 2018
- Launch of name registry - September 2018
- Launch of the RChain main net - December 2018

VM Milestones

Name	Status	Date
Roscala.Void release plan	RELEASED	16 C
Roscala.Transition release plan	IN DEVELOPMENT	TBD
Roscala.Primitive release plan	PLANNING	
Roscala.FFI- Draft	PLANNING	

Node Milestones

Name	Status	Date
Node.Hello release plan	RELEASED Release announcement	22 Dec 2017
node - 0.1 release plan	RELEASED Release announcement	15 Mar 2018
node - 0.2 release	RELEASED	29 Mar 2018



A Rholang tutorial.

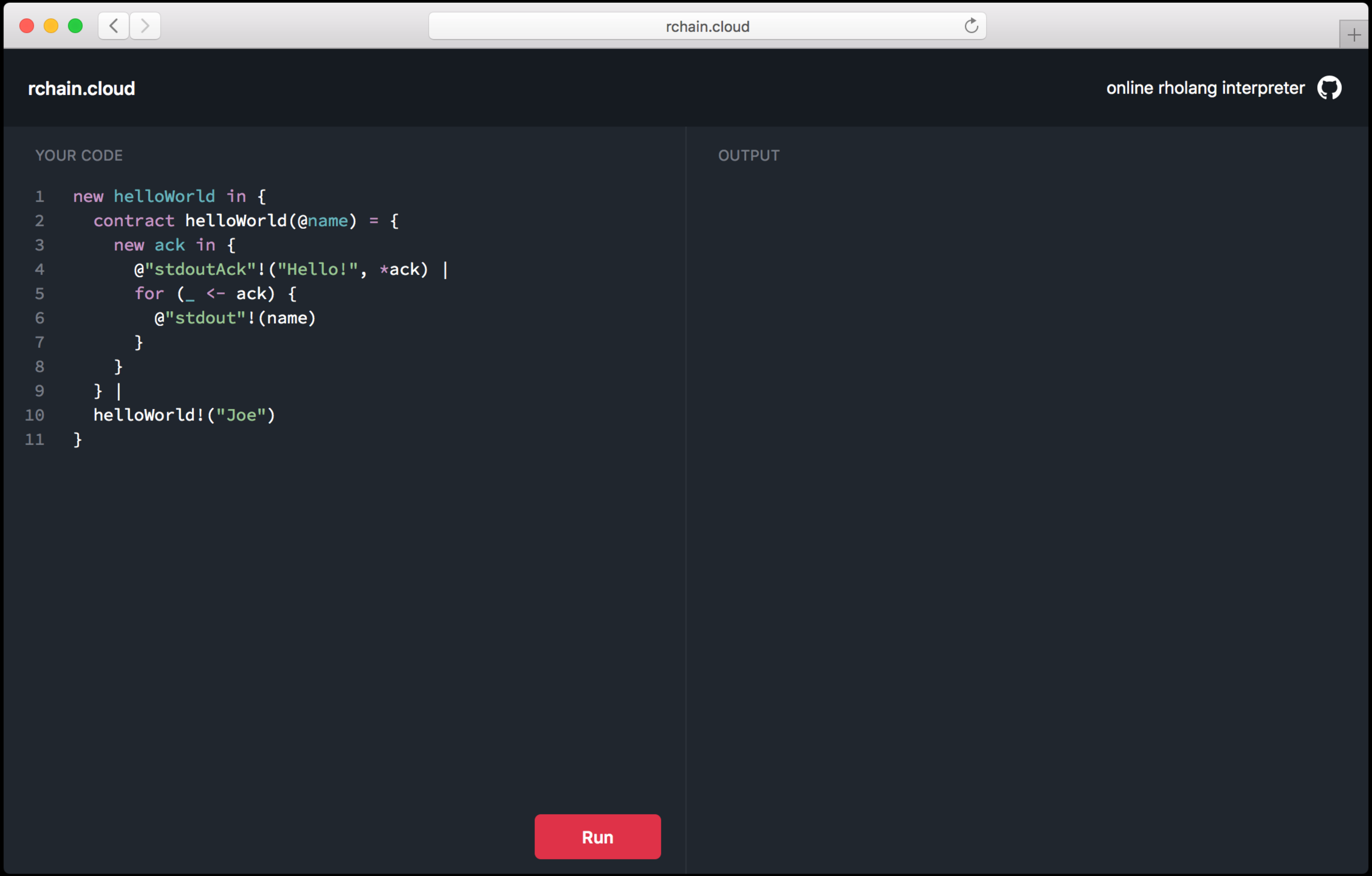
Rholang is a new programming language designed for use in distributed systems. Like all newborn things, it is growing and changing rapidly; this document describes the syntax that will be used in the RNode-0.3 release.

Rholang is "process-oriented": all computation is done by means of message passing. Messages are passed on "channels", which are rather like message queues but behave like sets rather than queues. Rholang is completely asynchronous, in the sense that while you can read a message from a channel and then do something with it, you can't send a message and then do something once it has been received---at least, not without explicitly waiting for an acknowledgment message from the receiver. Note that throughout this document the words "name" and "channel" are used interchangeably. This is because in the rho-calculus (on which Rholang is based) the term name is used, however because you can send and receive information on names, semantically they are like channels.

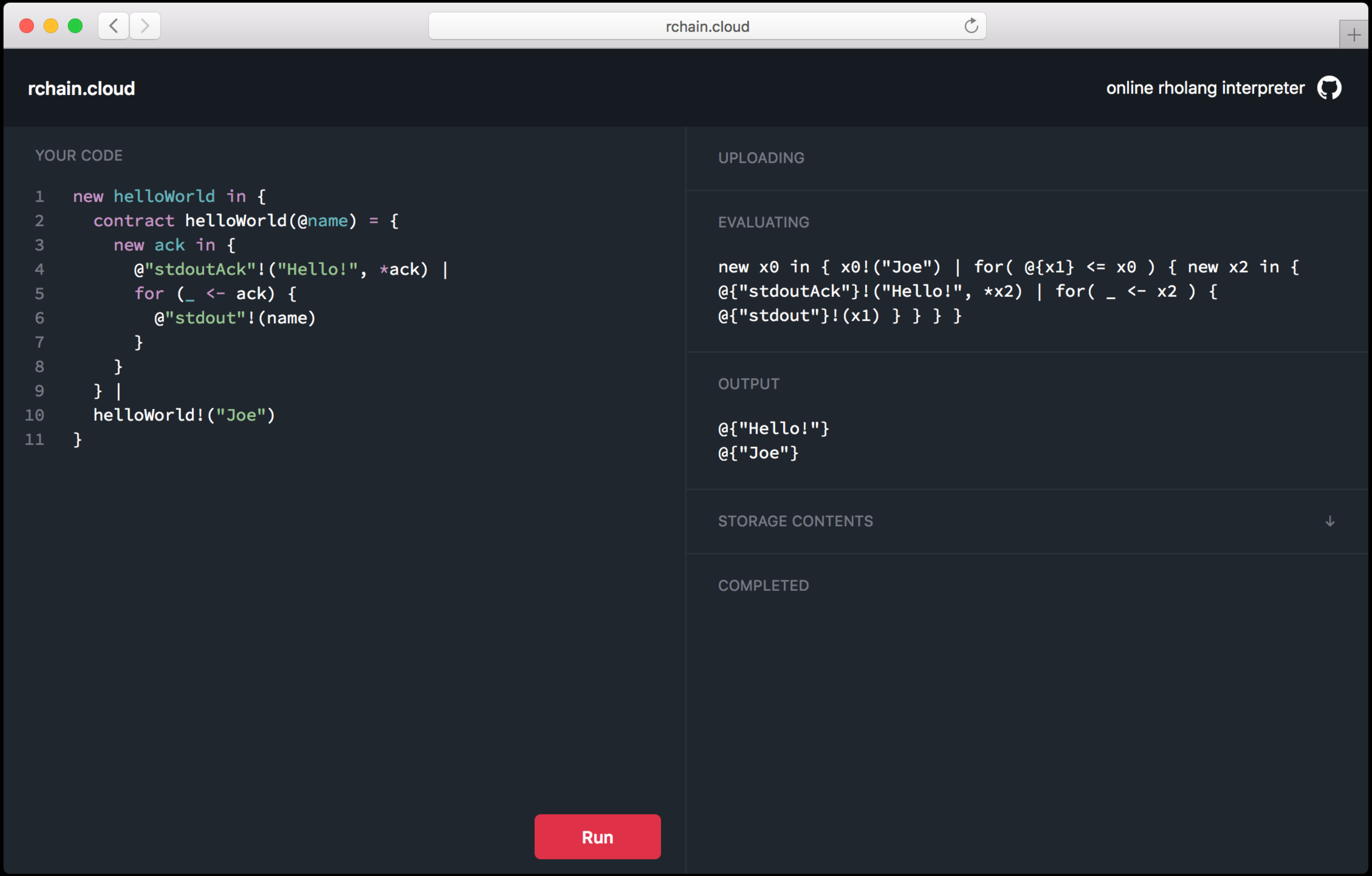
Getting started

There is not an IDE for Rholang. Get started with Rholang by selecting one of the options below.

- **Run Rholang on RNode** - Write Rholang contracts in an editor of your choice and run them on RNode using either the REPL or EVAL modes. **Get started** with the latest version of RNode.
- **Run Rholang on a web interface** - This **web interface** was created by a RChain community member.
- **Write Rholang using an IntelliJ plugin** - This **Rholang IntelliJ plugin** was created by a RChain community member.



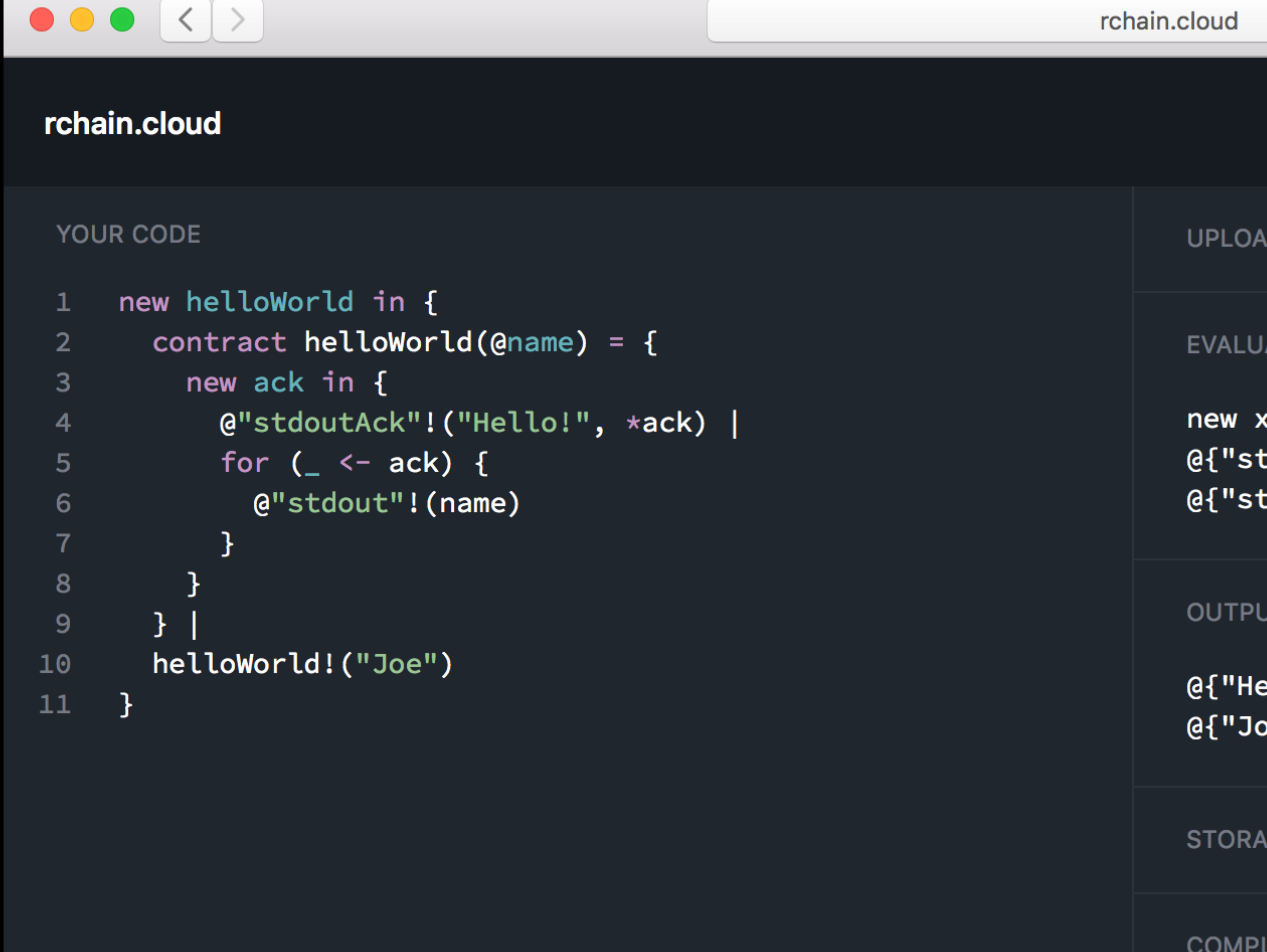
```
new helloWorld in {
  contract helloWorld(@name) = {
    new ack in {
      @"stdoutAck"!("Hello!", *ack) |
      for (_ <- ack) {
        @"stdout"!(name)
      }
    }
  } |
  helloWorld!("Joe")
}
```



```
new helloWorld in {
  contract helloWorld(@name) = {
    new ack in {
      @"stdoutAck"!("Hello!", *ack) |
      for (_ <- ack) {
        @"stdout"!(name)
      }
    }
  } |
  helloWorld!("Joe")
}
```

```
new x0 in { x0!("Joe") | for( @{x1} <= x0 ) { new x2 in {
@{"stdoutAck"}!("Hello!", *x2) | for( _ <- x2 ) {
@{"stdout"}!(x1) } } } }
```

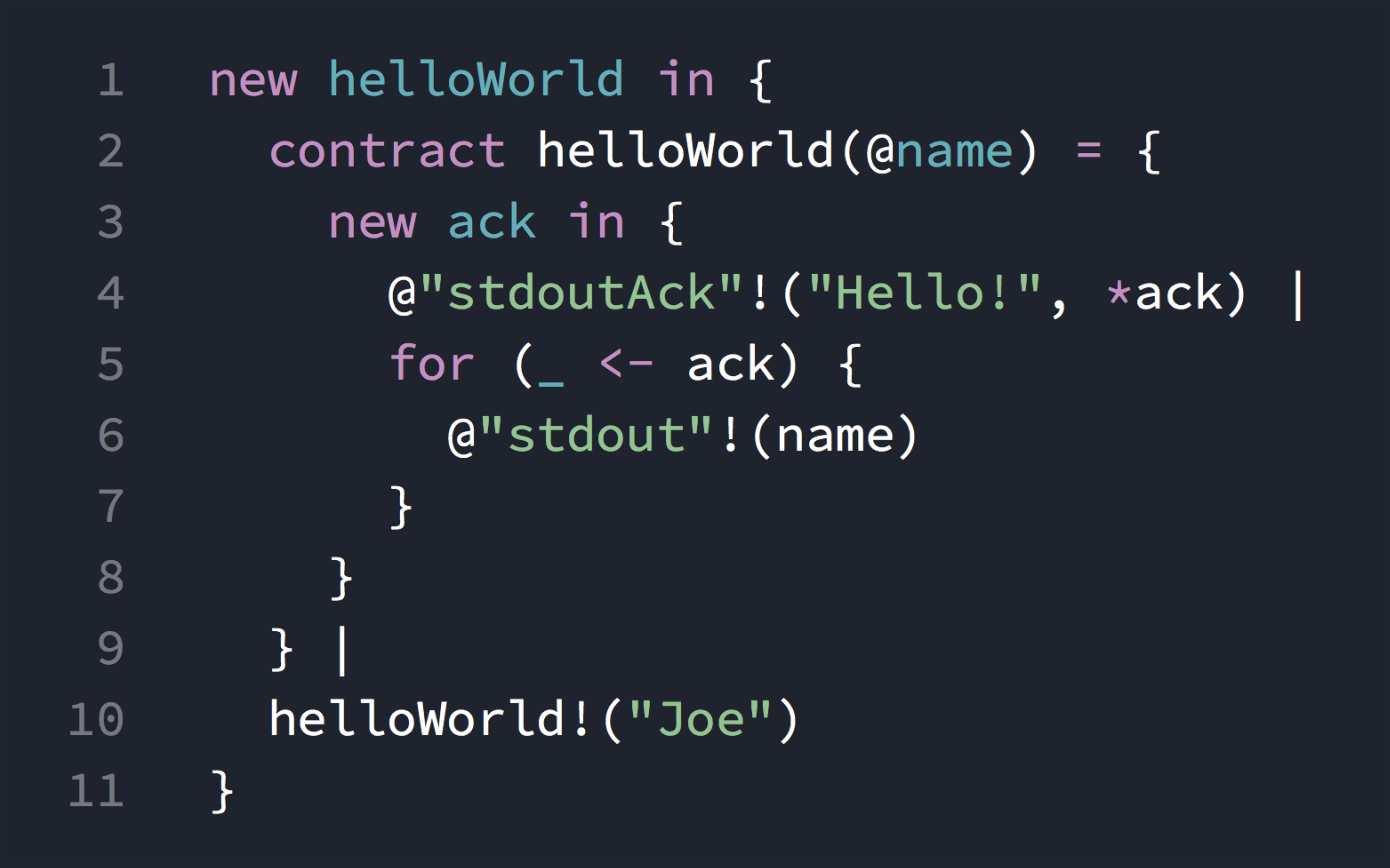
```
@{"Hello!"}
@{"Joe"}
```



```
new helloWorld in {
  contract helloWorld(@name) = {
    new ack in {
      @"stdoutAck"!("Hello!", *ack) |
      for (_ <- ack) {
        @"stdout"!(name)
      }
    }
  } |
  helloWorld!("Joe")
}
```

```
new x
@{"st
@{"st
```

```
@{"He
@{"Jo
```

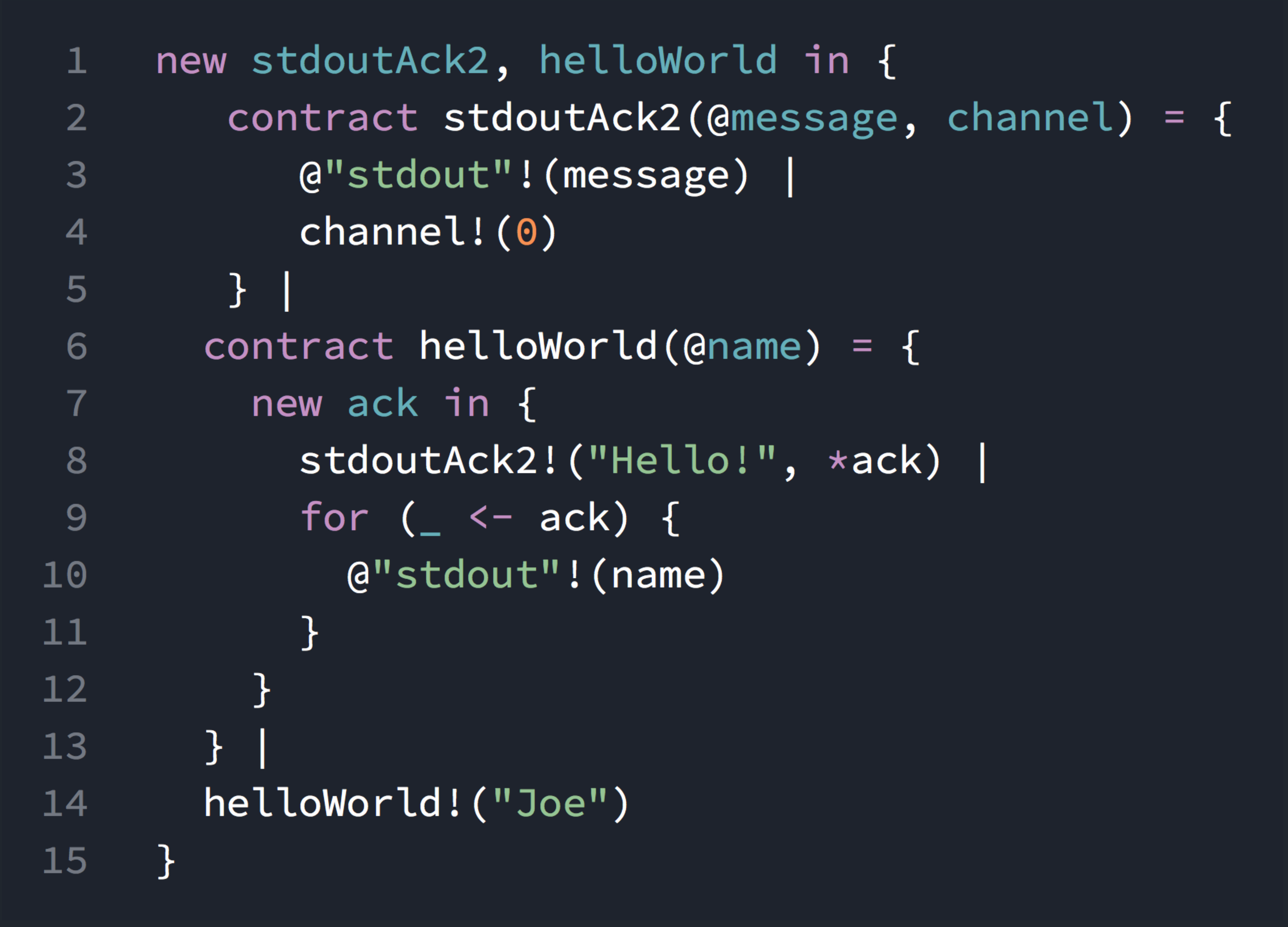


```
new helloWorld in {
  contract helloWorld(@name) = {
    new ack in {
      @"stdoutAck"!("Hello!", *ack) |
      for (_ <- ack) {
        @"stdout"!(name)
      }
    }
  } |
  helloWorld!("Joe")
}
```

```
new helloWorld in {
  contract helloWorld(@name) = {
    new ack in {
      @"stdoutAck"!("Hello!", *ack) |
      for (_ <- ack) {
        @"stdout"!(name)
      }
    }
  } |
  helloWorld!("Joe")
}
```

OUTPUT

```
Hello!
Joe
```



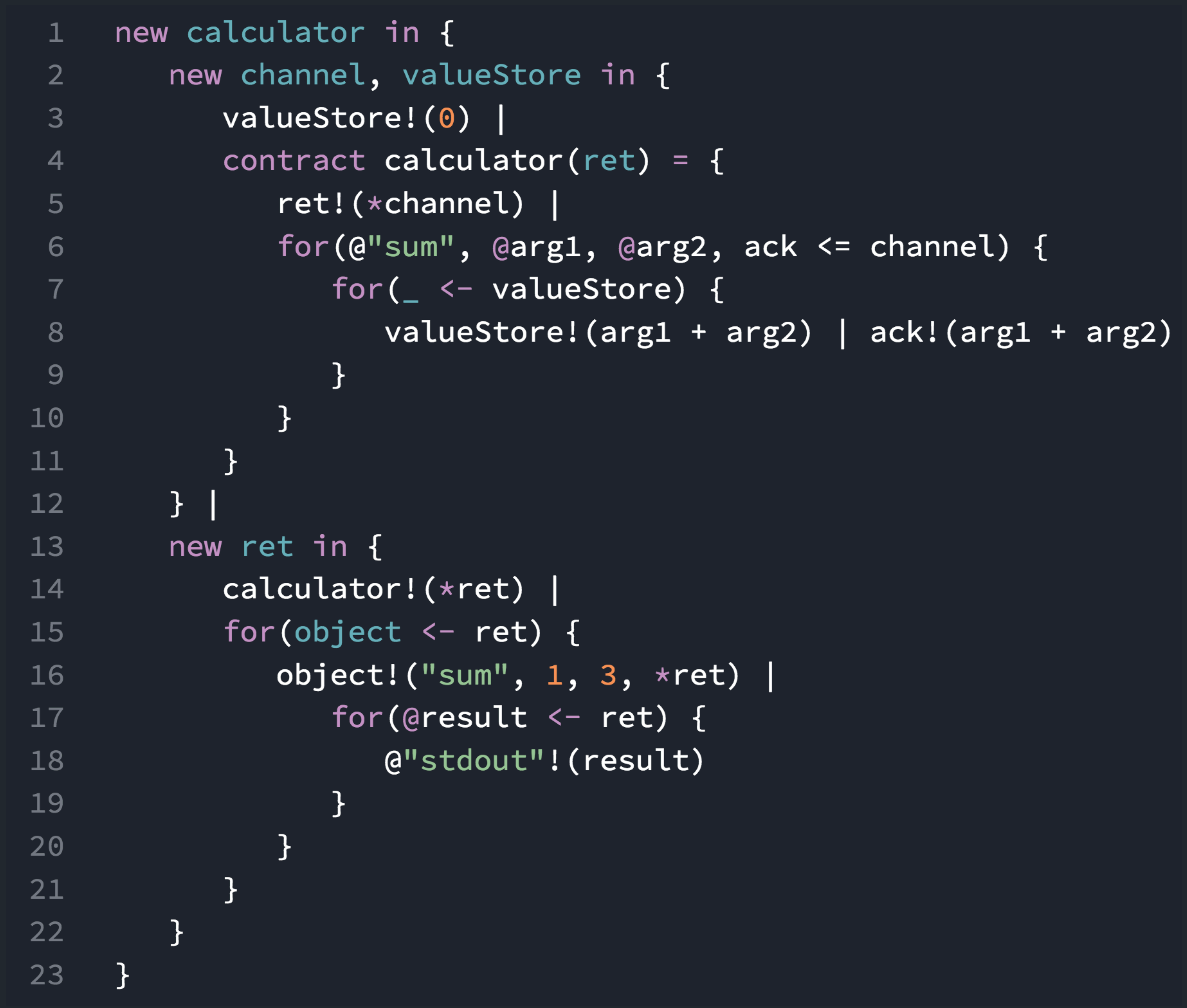
```
new stdoutAck2, helloWorld in {
   contract stdoutAck2(@message, channel) = {
      @"stdout"!(message) |
      channel!(0)
   } |
  contract helloWorld(@name) = {
    new ack in {
      stdoutAck2!("Hello!", *ack) |
      for (_ <- ack) {
        @"stdout"!(name)
      }
    }
  } |
  helloWorld!("Joe")
}
```

```
new stdoutAck2, helloWorld in {
   contract stdoutAck2(@message, channel) = {
      @"stdout"!(message) |
      channel!(0)
   } |
  contract helloWorld(@name) = {
    new ack in {
      stdoutAck2!("Hello!", *ack) |
      for (_ <- ack) {
        @"stdout"!(name)
      }
    }
  } |
  helloWorld!("Joe")
}
```

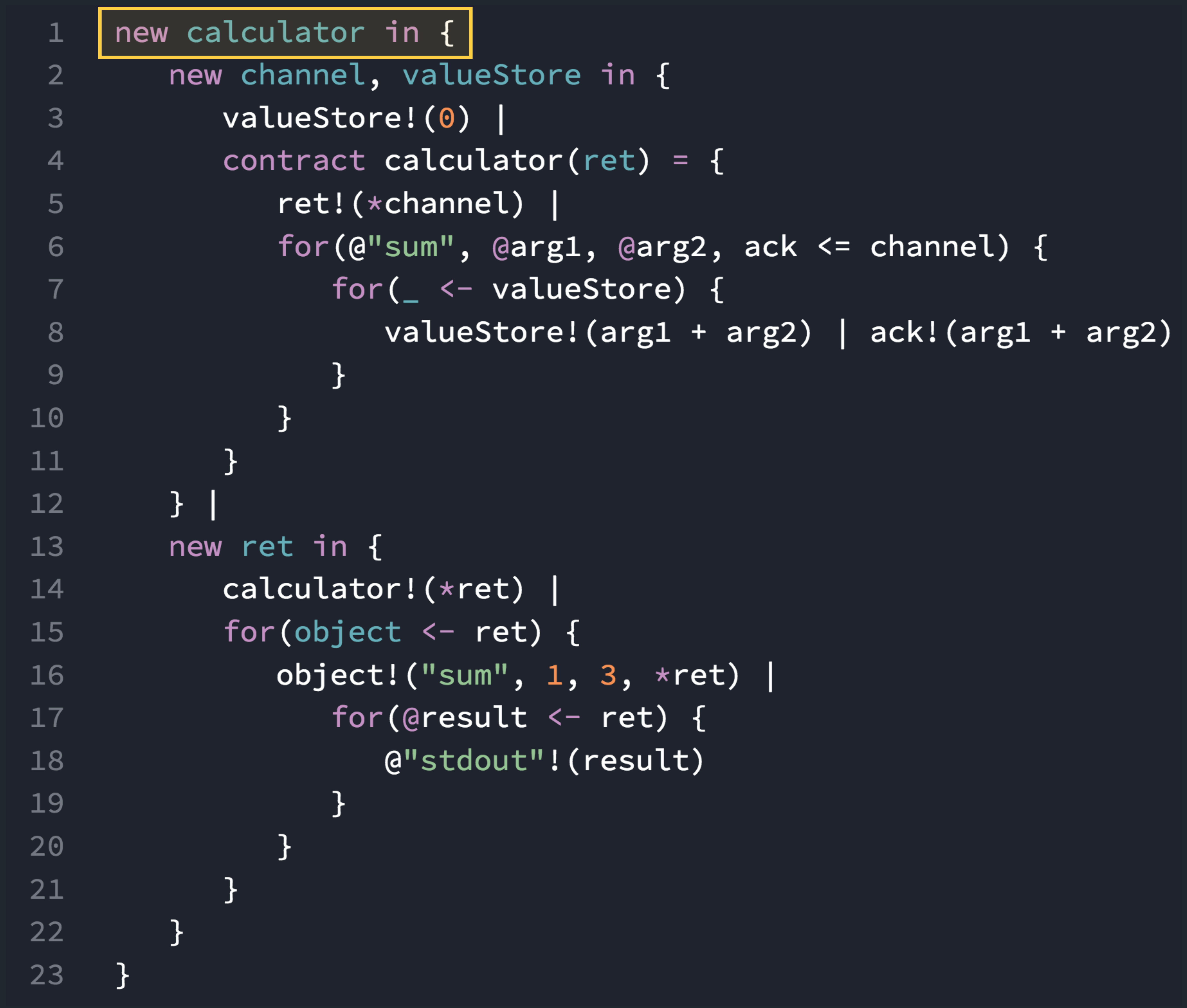
OUTPUT

```
Joe
Hello!
```

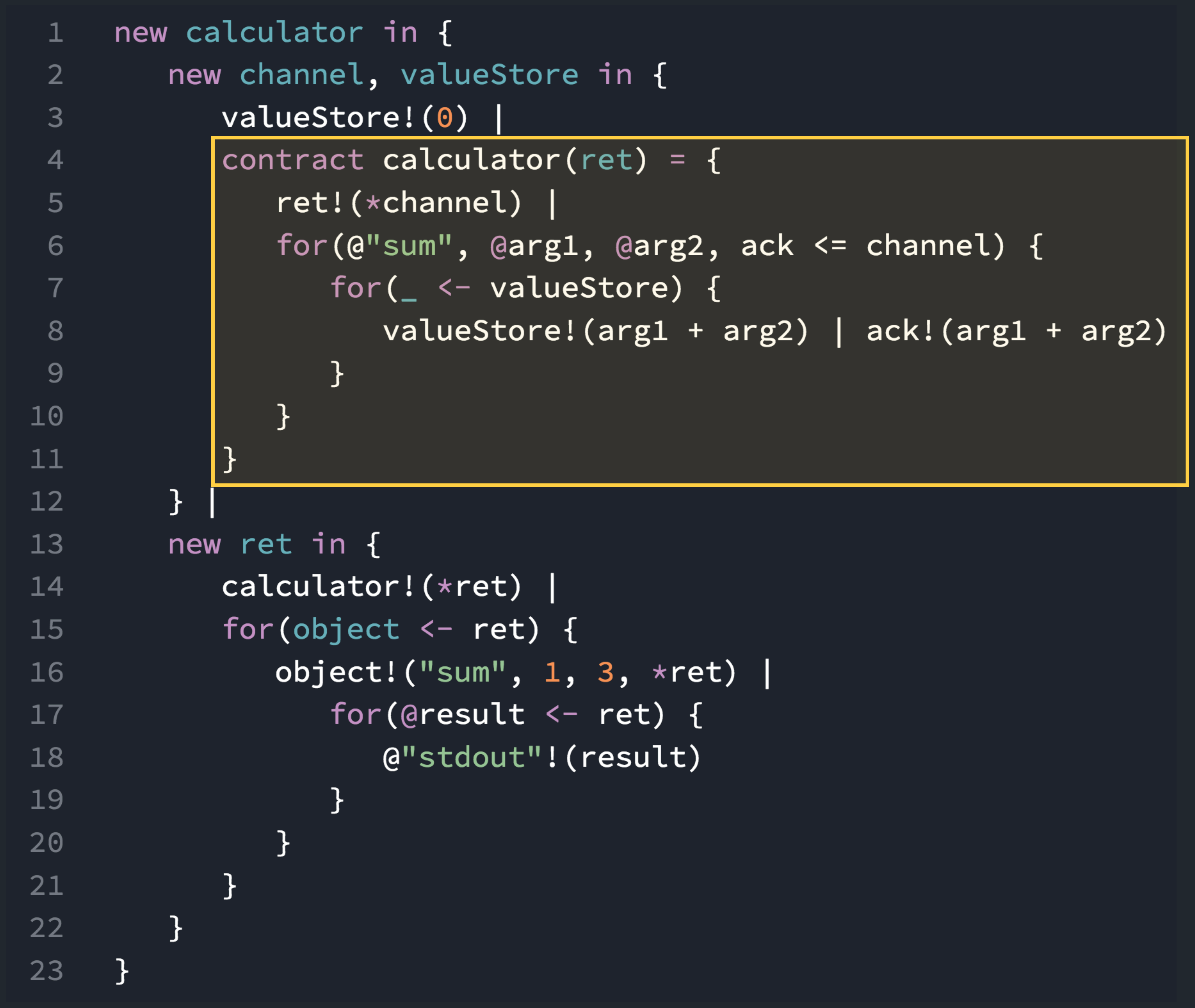
Composability



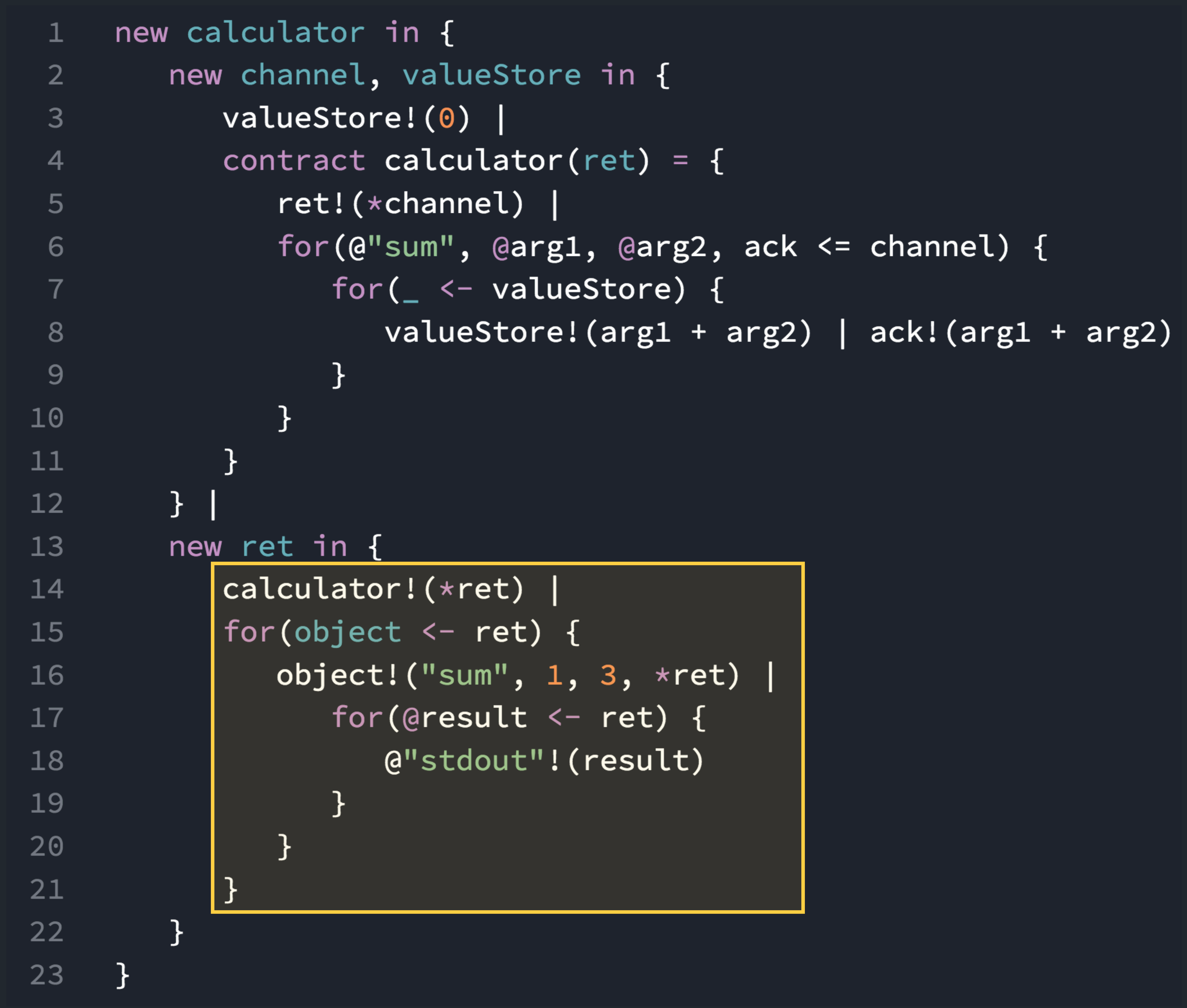
```
new calculator in {
   new channel, valueStore in {
      valueStore!(0) |
      contract calculator(ret) = {
         ret!(*channel) |
         for(@"sum", @arg1, @arg2, ack <= channel) {
            for(_ <- valueStore) {
               valueStore!(arg1 + arg2) | ack!(arg1 + arg2)
            }
         }
      }
   } |
   new ret in {
      calculator!(*ret) |
      for(object <- ret) {
         object!("sum", 1, 3, *ret) |
            for(@result <- ret) {
               @"stdout"!(result)
            }
         }
      }
   }
}
```



```
new calculator in {
   new channel, valueStore in {
      valueStore!(0) |
      contract calculator(ret) = {
         ret!(*channel) |
         for(@"sum", @arg1, @arg2, ack <= channel) {
            for(_ <- valueStore) {
               valueStore!(arg1 + arg2) | ack!(arg1 + arg2)
            }
         }
      }
   } |
   new ret in {
      calculator!(*ret) |
      for(object <- ret) {
         object!("sum", 1, 3, *ret) |
            for(@result <- ret) {
               @"stdout"!(result)
            }
         }
      }
   }
}
```



```
new calculator in {
   new channel, valueStore in {
      valueStore!(0) |
      contract calculator(ret) = {
         ret!(*channel) |
         for(@"sum", @arg1, @arg2, ack <= channel) {
            for(_ <- valueStore) {
               valueStore!(arg1 + arg2) | ack!(arg1 + arg2)
            }
         }
      }
   } |
   new ret in {
      calculator!(*ret) |
      for(object <- ret) {
         object!("sum", 1, 3, *ret) |
            for(@result <- ret) {
               @"stdout"!(result)
            }
         }
      }
   }
}
```

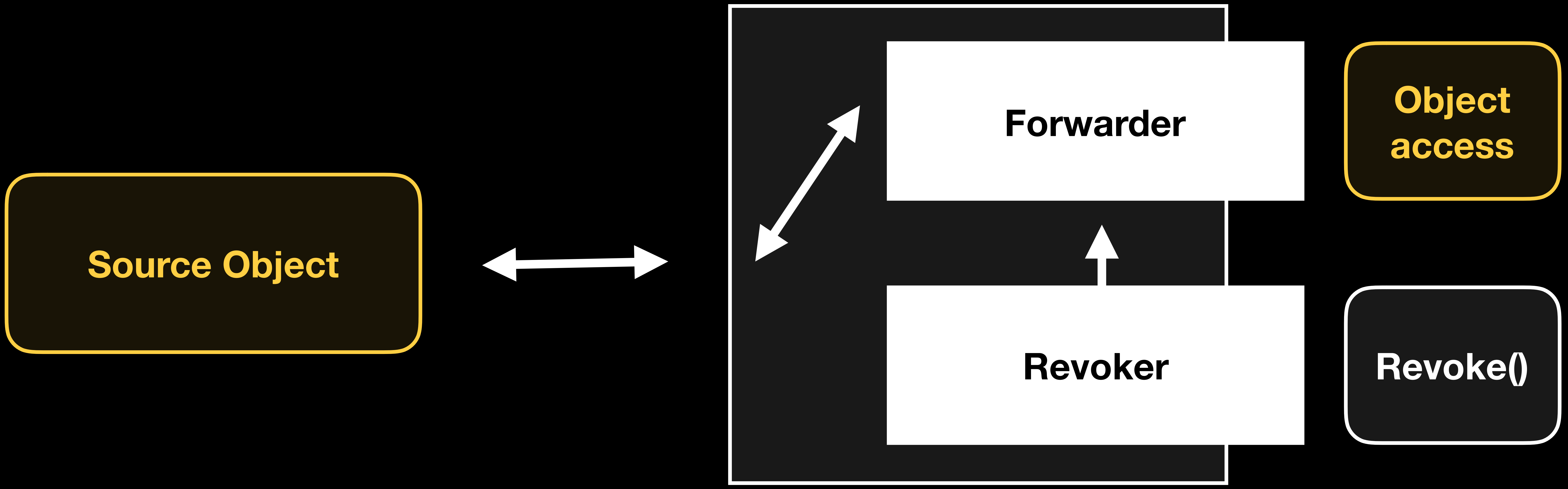


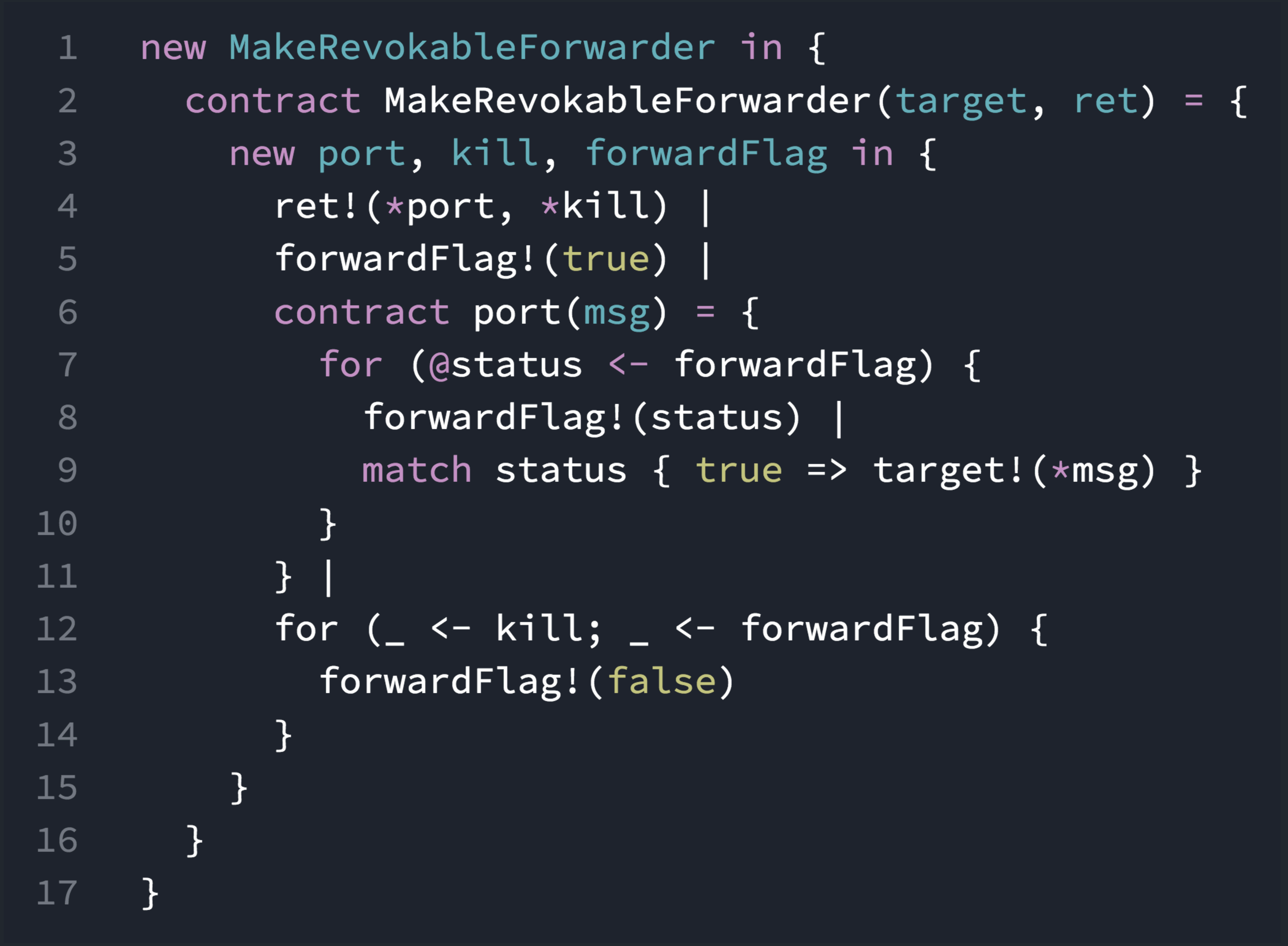
```
new calculator in {
   new channel, valueStore in {
      valueStore!(0) |
      contract calculator(ret) = {
         ret!(*channel) |
         for(@"sum", @arg1, @arg2, ack <= channel) {
            for(_ <- valueStore) {
               valueStore!(arg1 + arg2) | ack!(arg1 + arg2)
            }
         }
      }
   } |
   new ret in {
      calculator!(*ret) |
      for(object <- ret) {
         object!("sum", 1, 3, *ret) |
            for(@result <- ret) {
               @"stdout"!(result)
            }
         }
      }
   }
}
```

Built-in Security

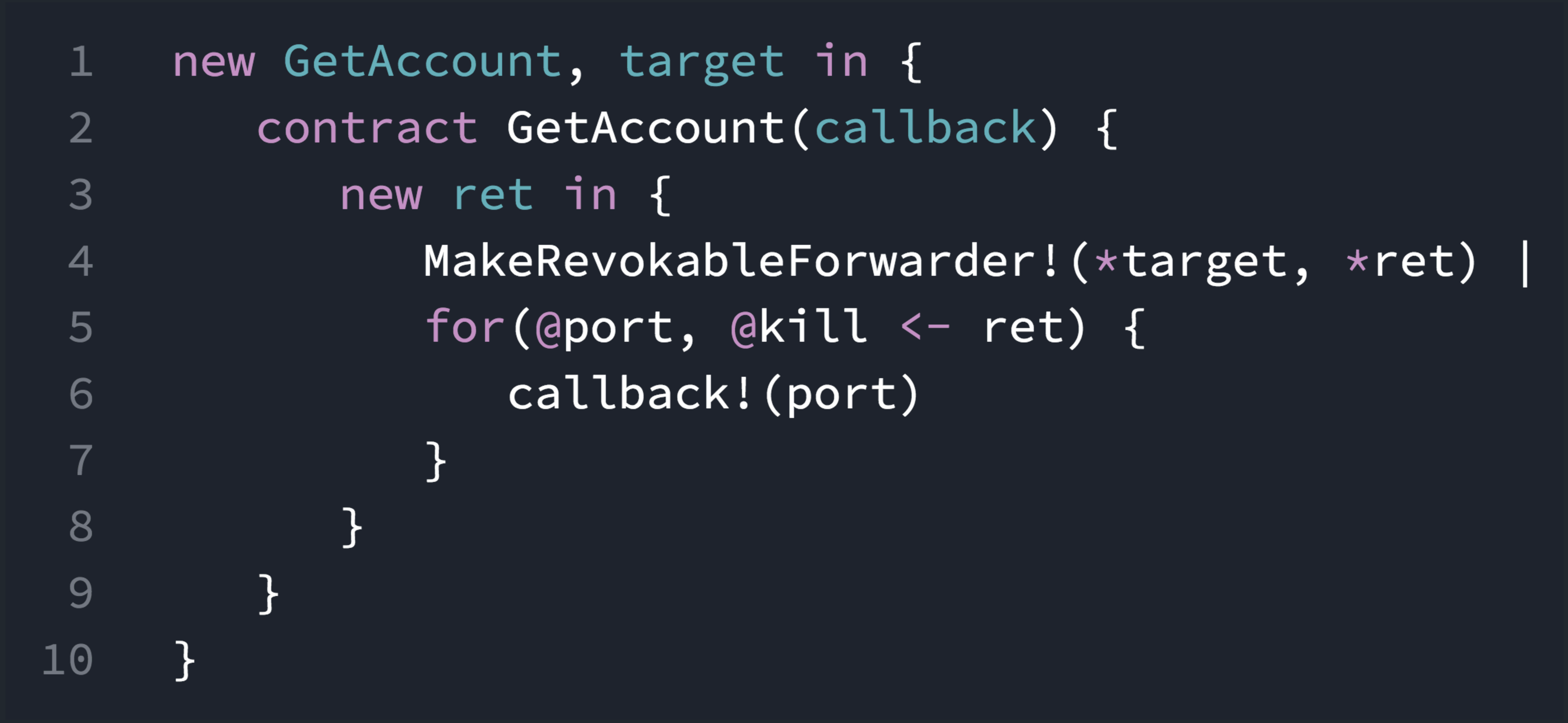
Principle of Least Authority

Revocable Forwarder

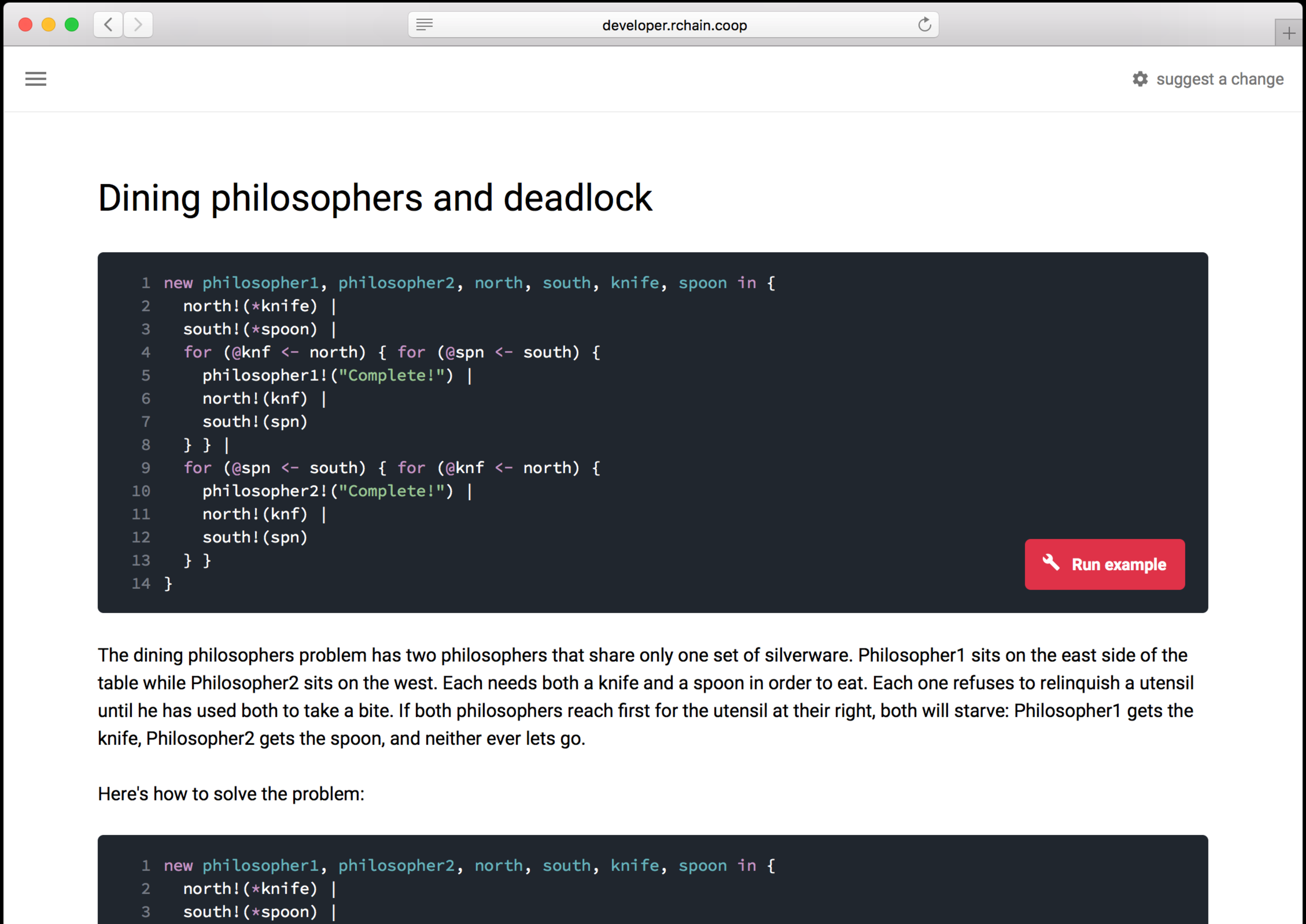




```
new MakeRevokableForwarder in {
  contract MakeRevokableForwarder(target, ret) = {
    new port, kill, forwardFlag in {
      ret!(*port, *kill) |
      forwardFlag!(true) |
      contract port(msg) = {
        for (@status <- forwardFlag) {
          forwardFlag!(status) |
          match status { true => target!(*msg) }
        }
      } |
      for (_ <- kill; _ <- forwardFlag) {
        forwardFlag!(false)
      }
    }
  }
}
```



```
new GetAccount, target in {
    contract GetAccount(callback) {
        new ret in {
            MakeRevokableForwarder!(*target, *ret) |
            for(@port, @kill <- ret) {
                callback!(port)
            }
        }
    }
}
```



Dining philosophers and deadlock

```
new philosopher1, philosopher2, north, south, knife, spoon in {
  north!(*knife) |
  south!(*spoon) |
  for (@knf <- north) { for (@spn <- south) {
    philosopher1!("Complete!") |
    north!(knf) |
    south!(spn)
  } } |
  for (@spn <- south) { for (@knf <- north) {
    philosopher2!("Complete!") |
    north!(knf) |
    south!(spn)
  } }
}
```

The dining philosophers problem has two philosophers that share only one set of silverware. Philosopher1 sits on the east side of the table while Philosopher2 sits on the west. Each needs both a knife and a spoon in order to eat. Each one refuses to relinquish a utensil until he has used both to take a bite. If both philosophers reach first for the utensil at their right, both will starve: Philosopher1 gets the knife, Philosopher2 gets the spoon, and neither ever lets go.

Here's how to solve the problem:

```
new philosopher1, philosopher2, north, south, knife, spoon in {
  north!(*knife) |
  south!(*spoon) |
```

developer.rchain.coop