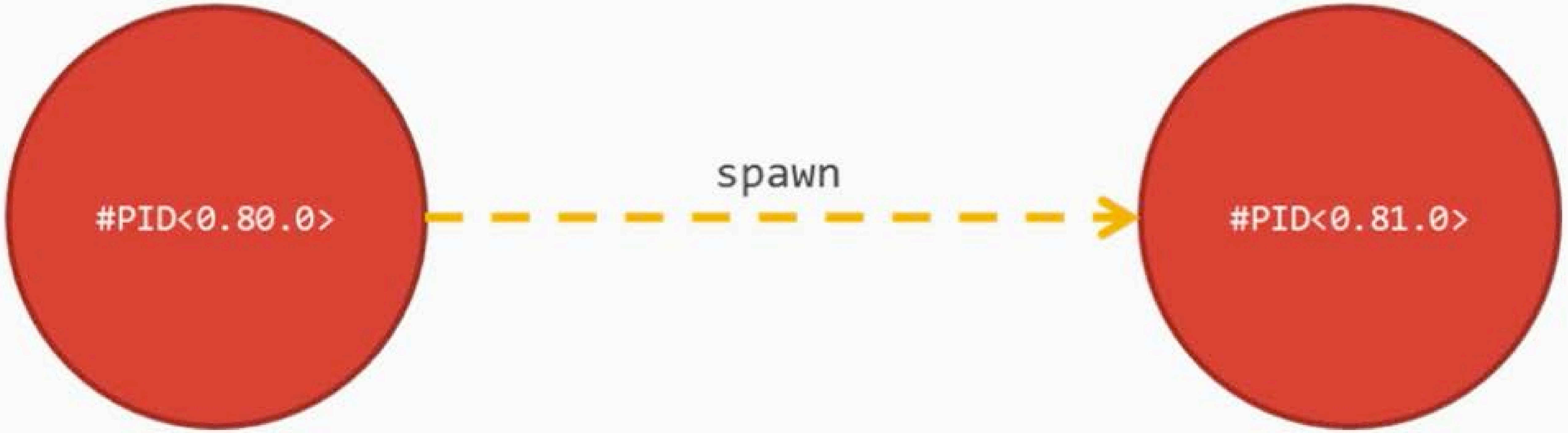
Messaging

Next Video

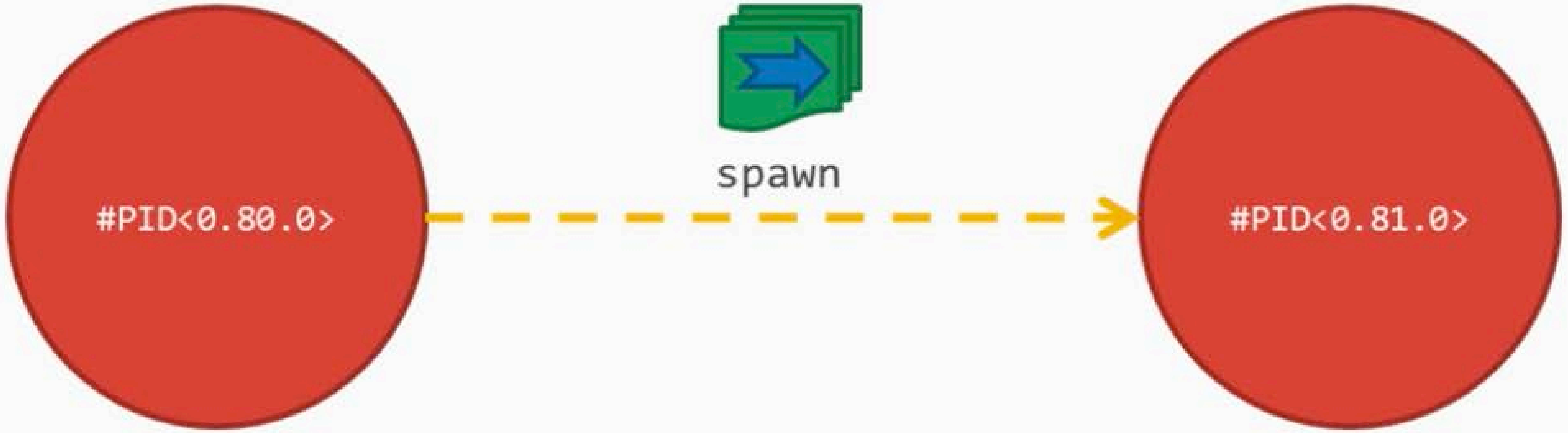
In this Video, we are going to take a look at…

- Process message queue
- Sending and receiving messages

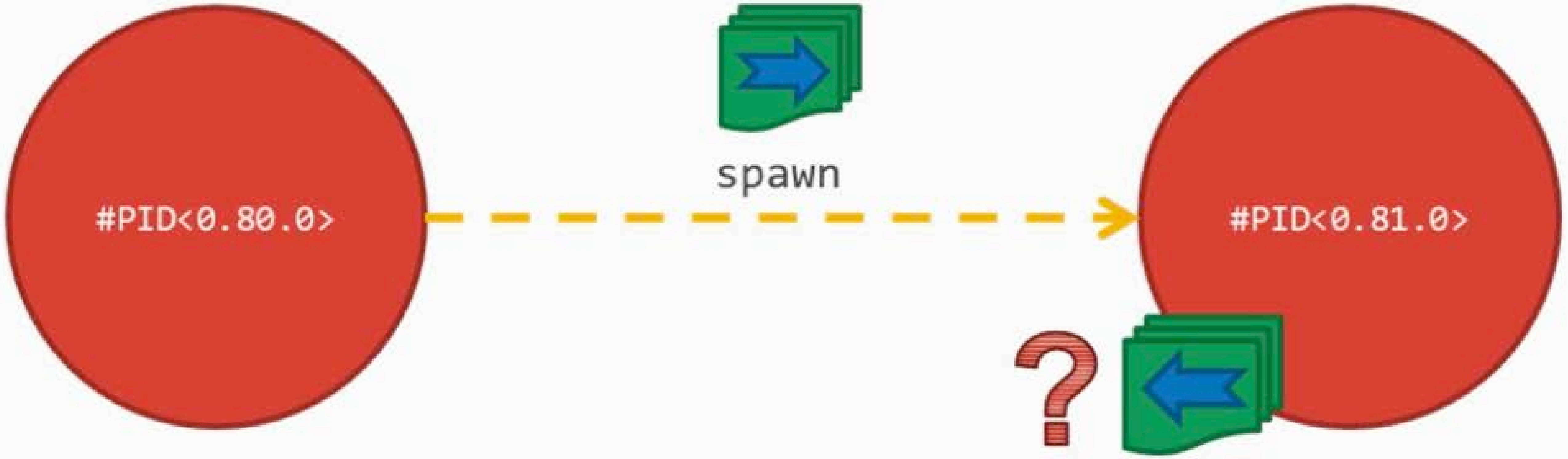
Processes and Messages



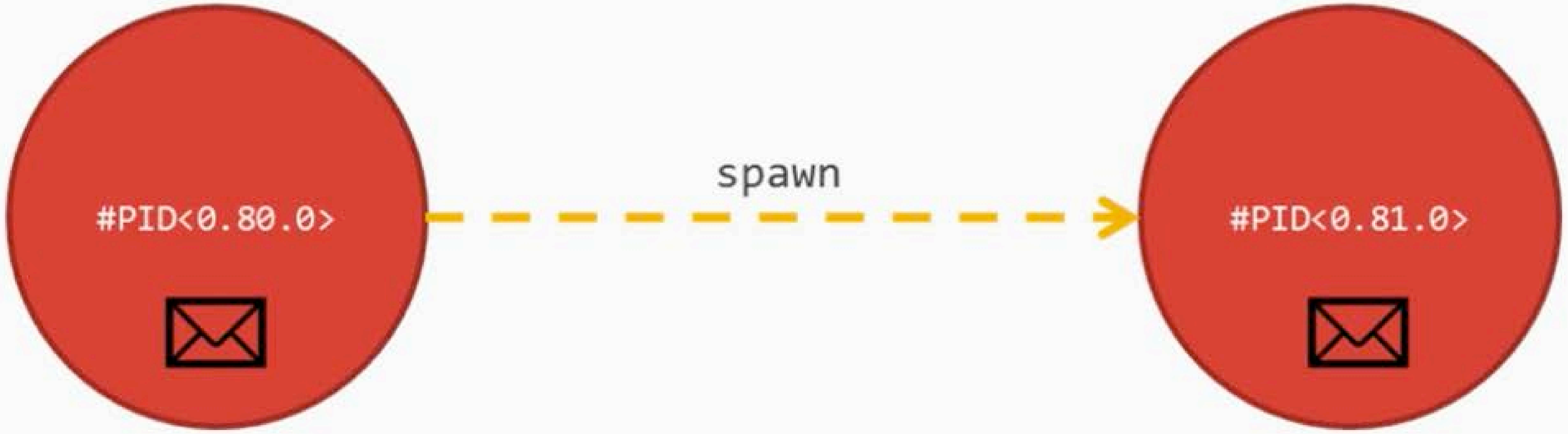
Processes and Messages



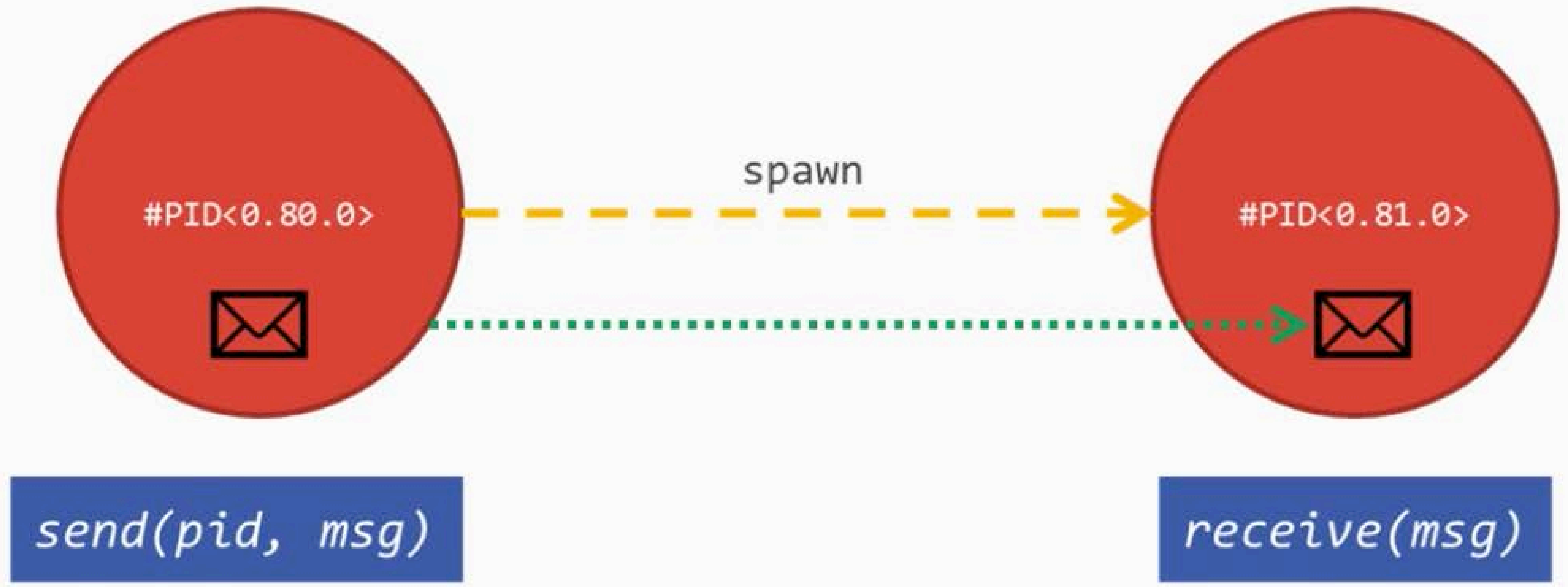
Processes and Messages



Processes and Messages



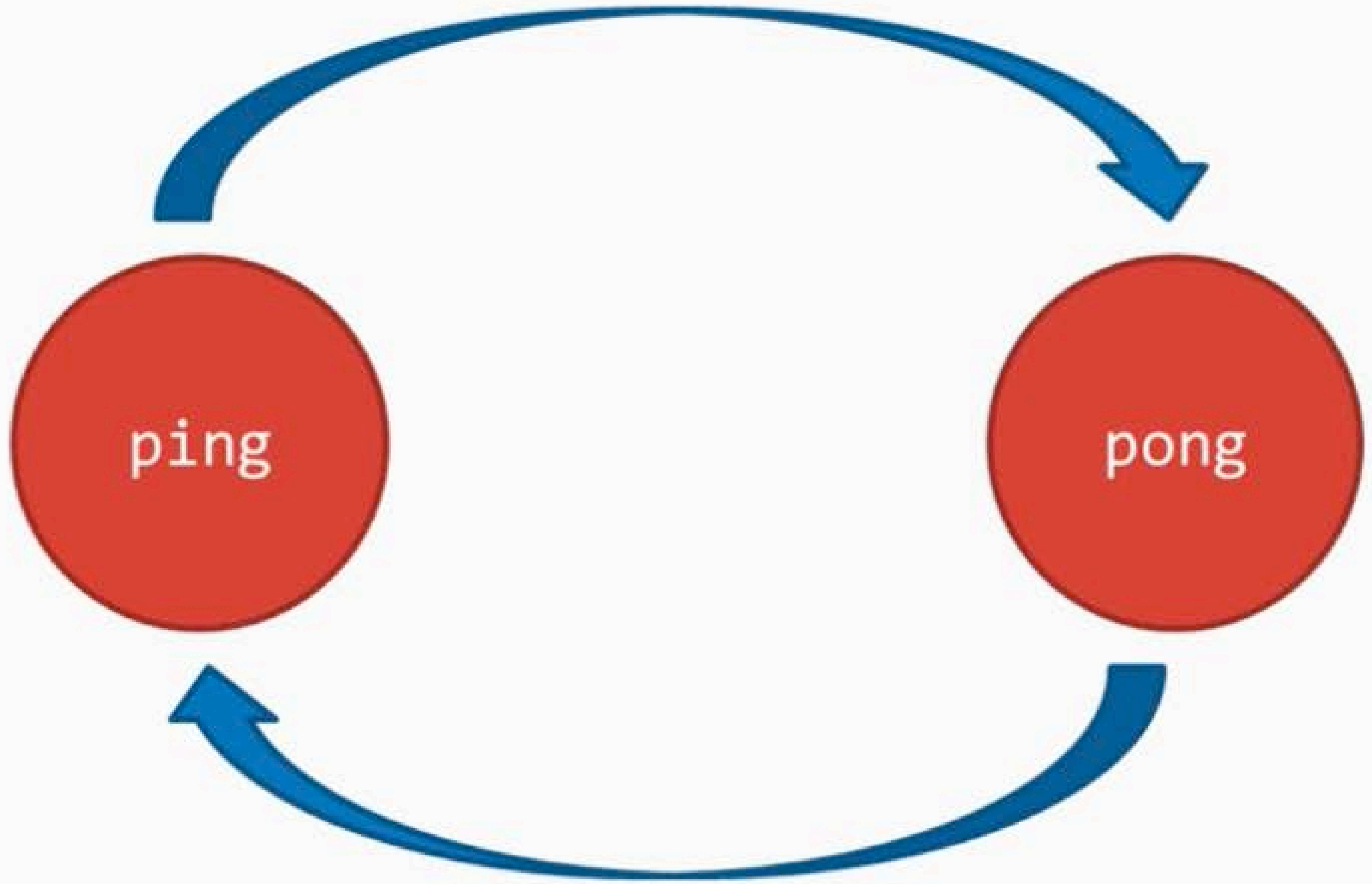
Processes and Messages



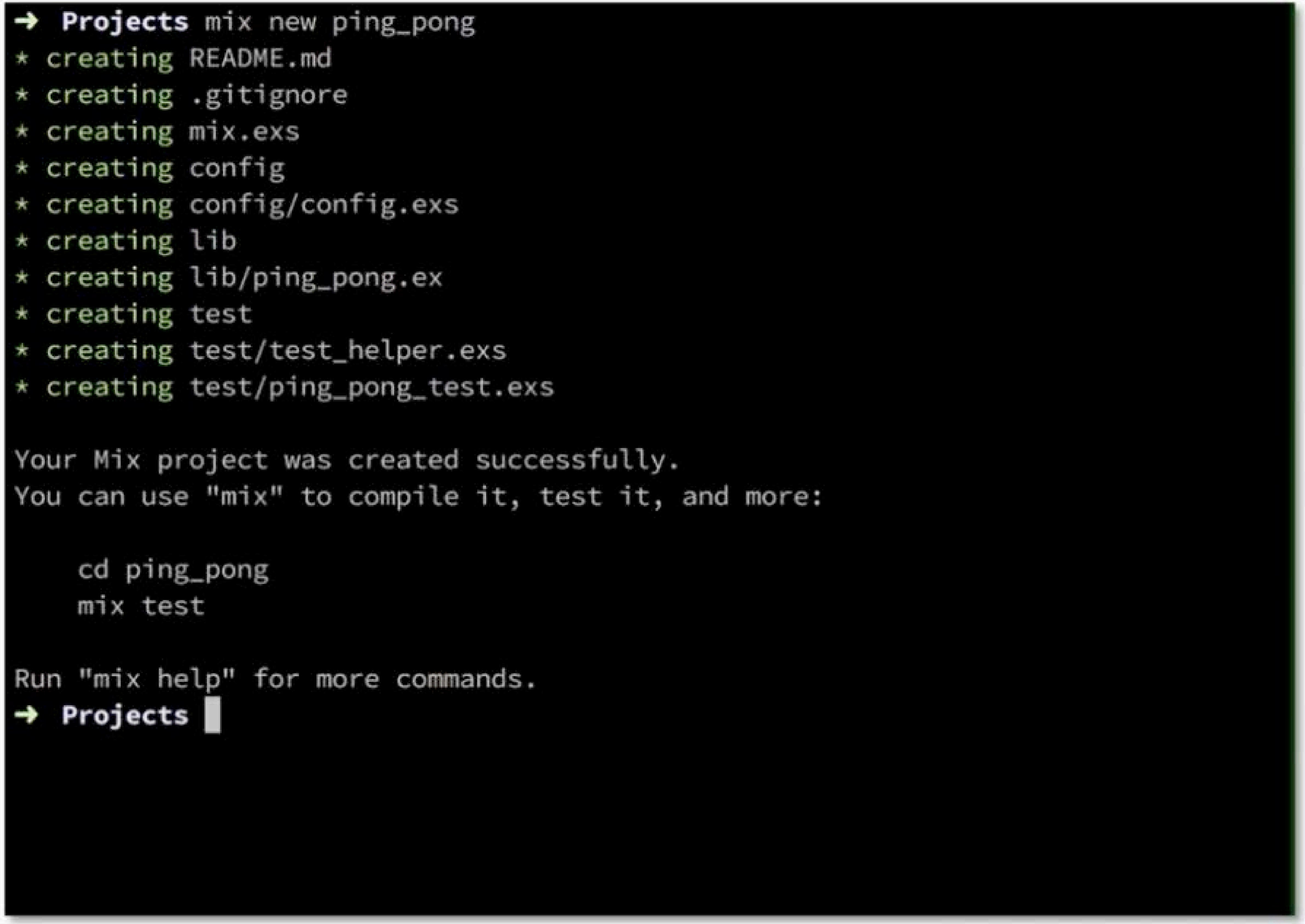
Ping-Pong

Two processes, passing messages back and forth indefinitely.

One is ping, the other is pong.

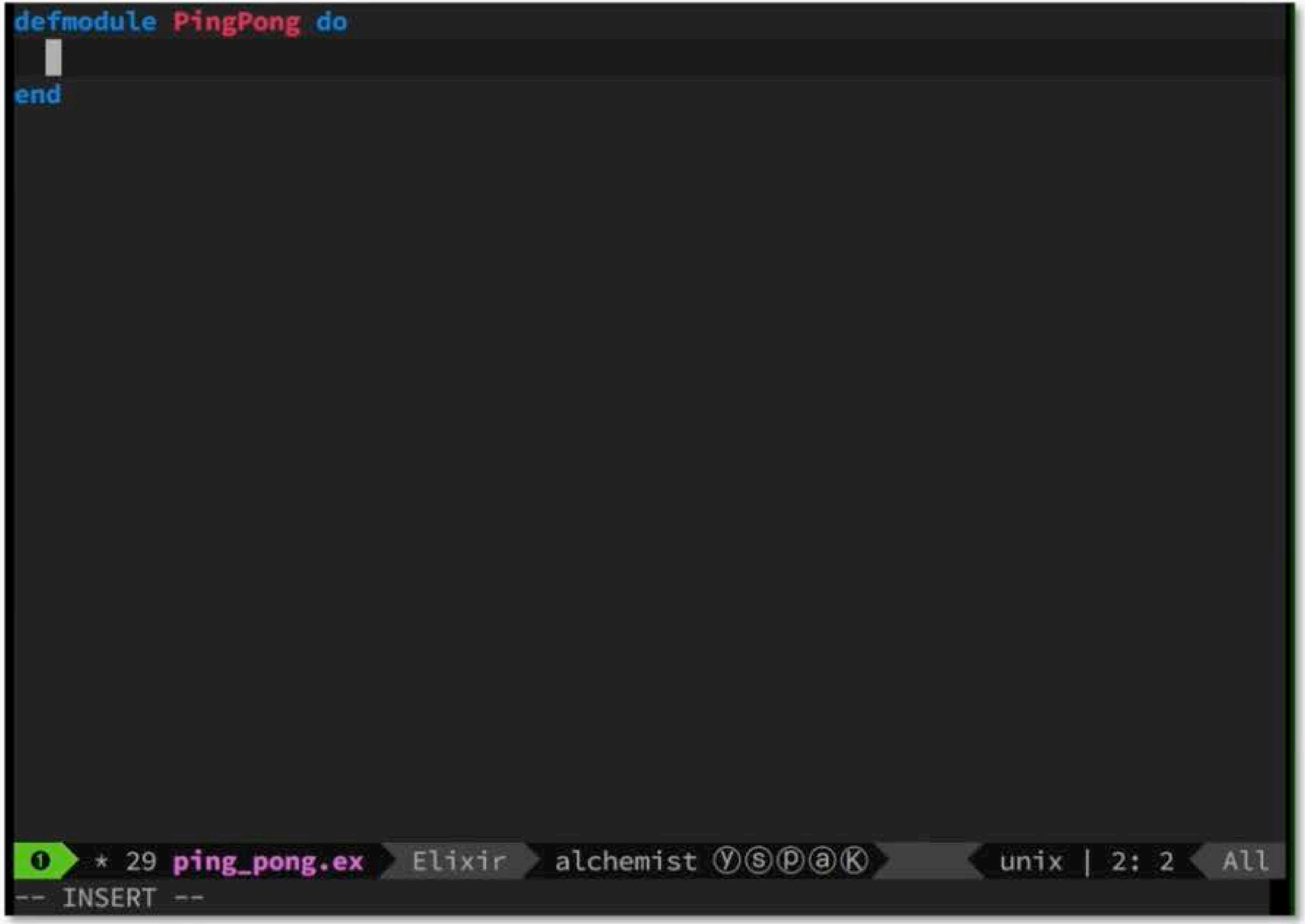


Ping-Pong



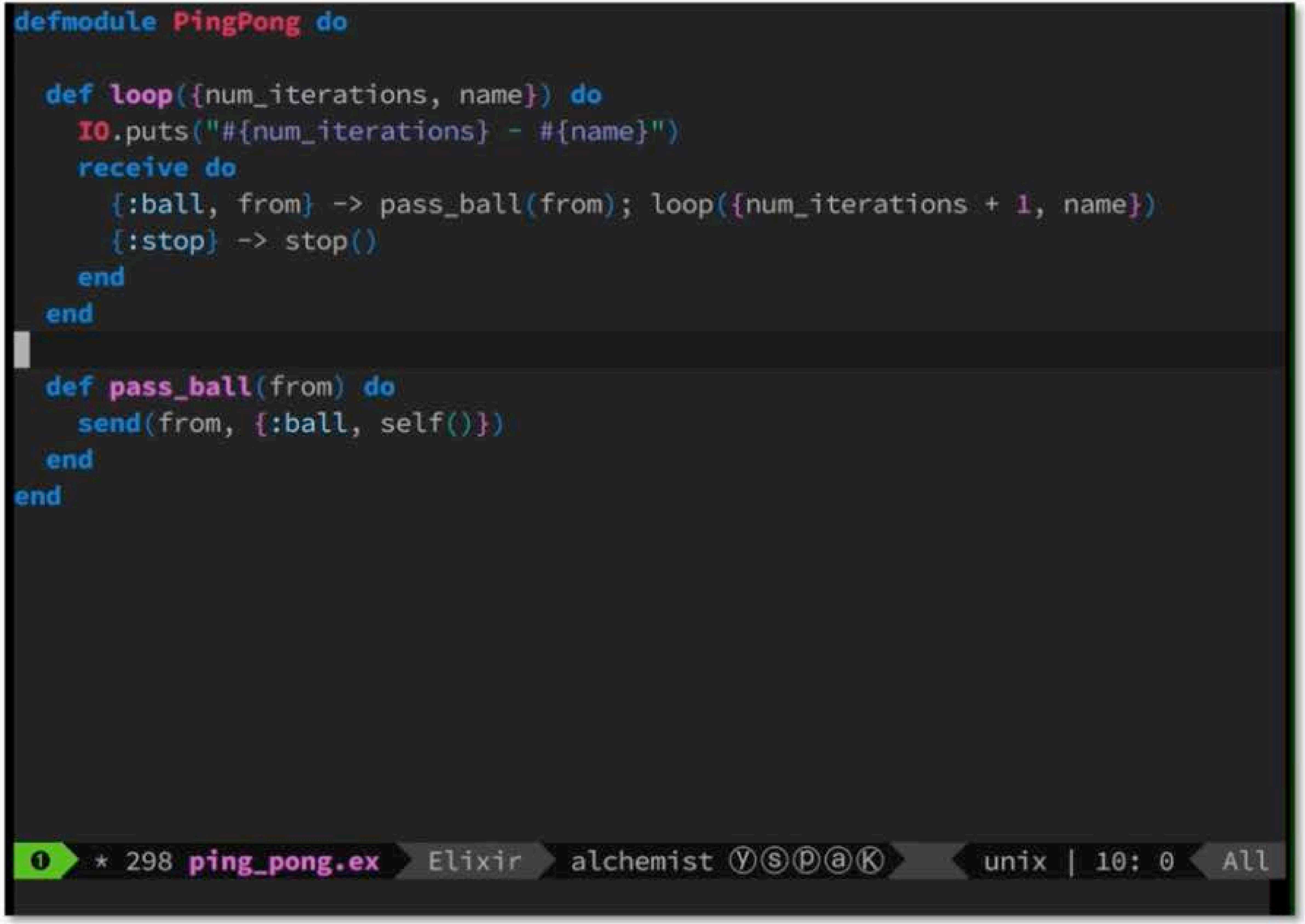
```
➜  Projects mix new ping_pong
* creating README.md
* creating .gitignore
* creating mix.exs
* creating config
* creating config/config.exs
* creating lib
* creating lib/ping_pong.ex
* creating test
* creating test/test_helper.exs
* creating test/ping_pong_test.exs

Your Mix project was created successfully.
You can use "mix" to compile it, test it, and more:

    cd ping_pong
    mix test

Run "mix help" for more commands.
➜  Projects
```

Ping-Pong



```
defmodule PingPong do
  
end
```

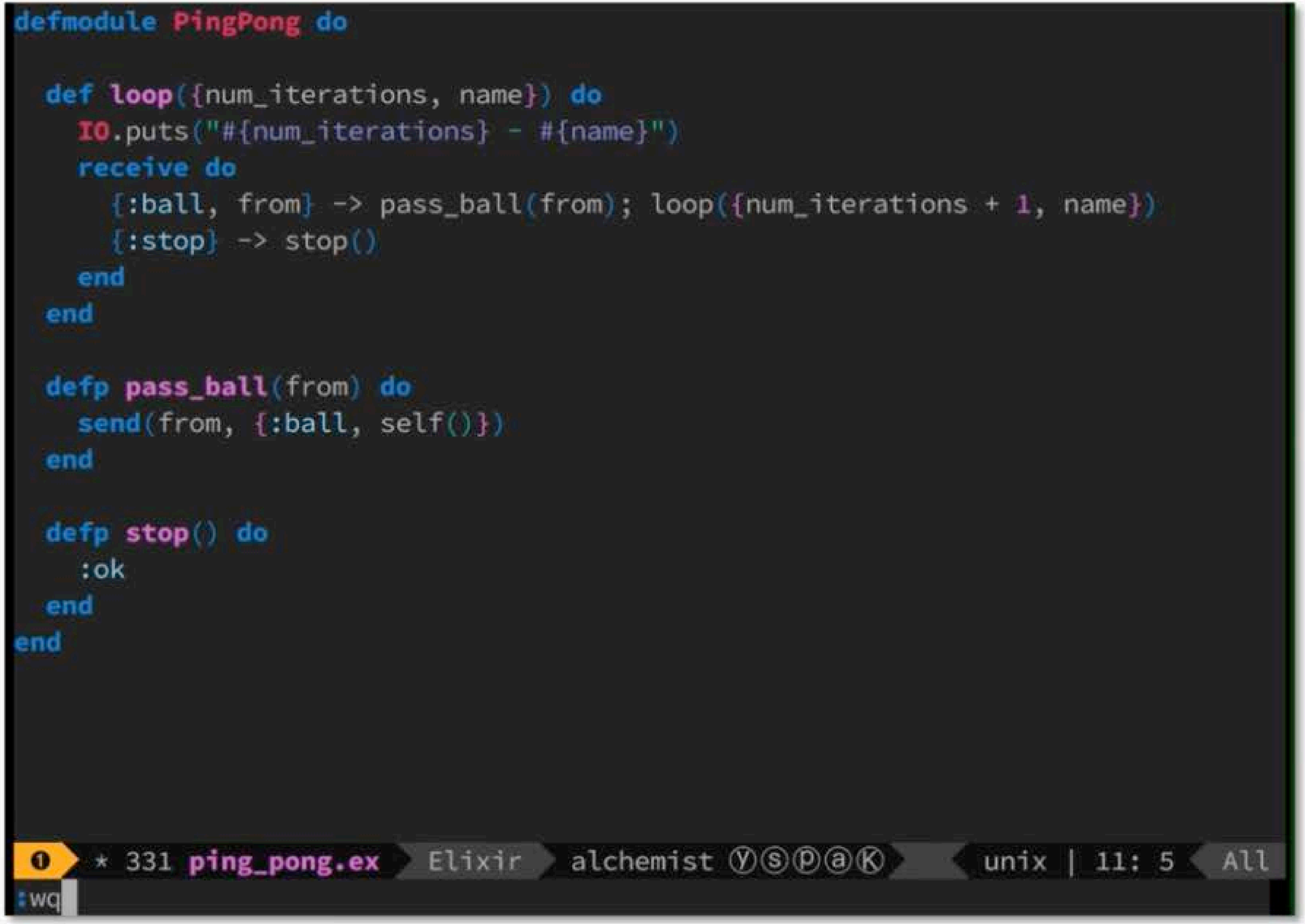
Ping-Pong



```elixir
defmodule PingPong do

  def loop({num_iterations, name}) do
    IO.puts("#{num_iterations} - #{name}")
    receive do
      {:ball, from} -> pass_ball(from); loop({num_iterations + 1, name})
      {:stop} -> stop()
    end
  end

  def pass_ball(from) do
    send(from, {:ball, self()})
  end
end
```

```
0  * 298 ping_pong.ex   Elixir   alchemist ⓨⓢⓟⓐⓚ   unix | 10: 0   All
```

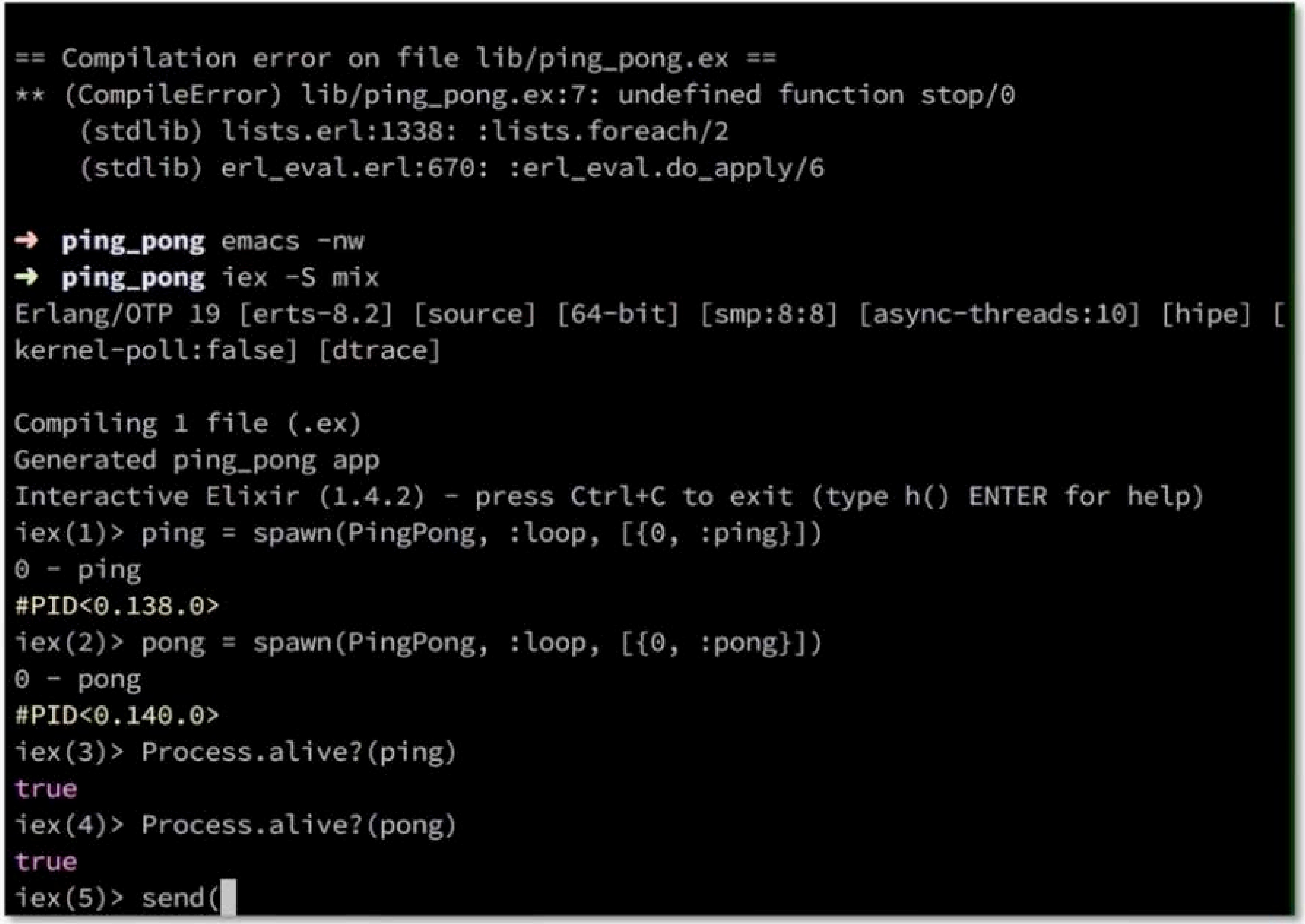
Ping-Pong



```
defmodule PingPong do

  def loop({num_iterations, name}) do
    IO.puts("#{num_iterations} - #{name}")
    receive do
      {:ball, from} -> pass_ball(from); loop({num_iterations + 1, name})
      {:stop} -> stop()
    end
  end

  defp pass_ball(from) do
    send(from, {:ball, self()})
  end

  defp stop() do
    :ok
  end
end
```

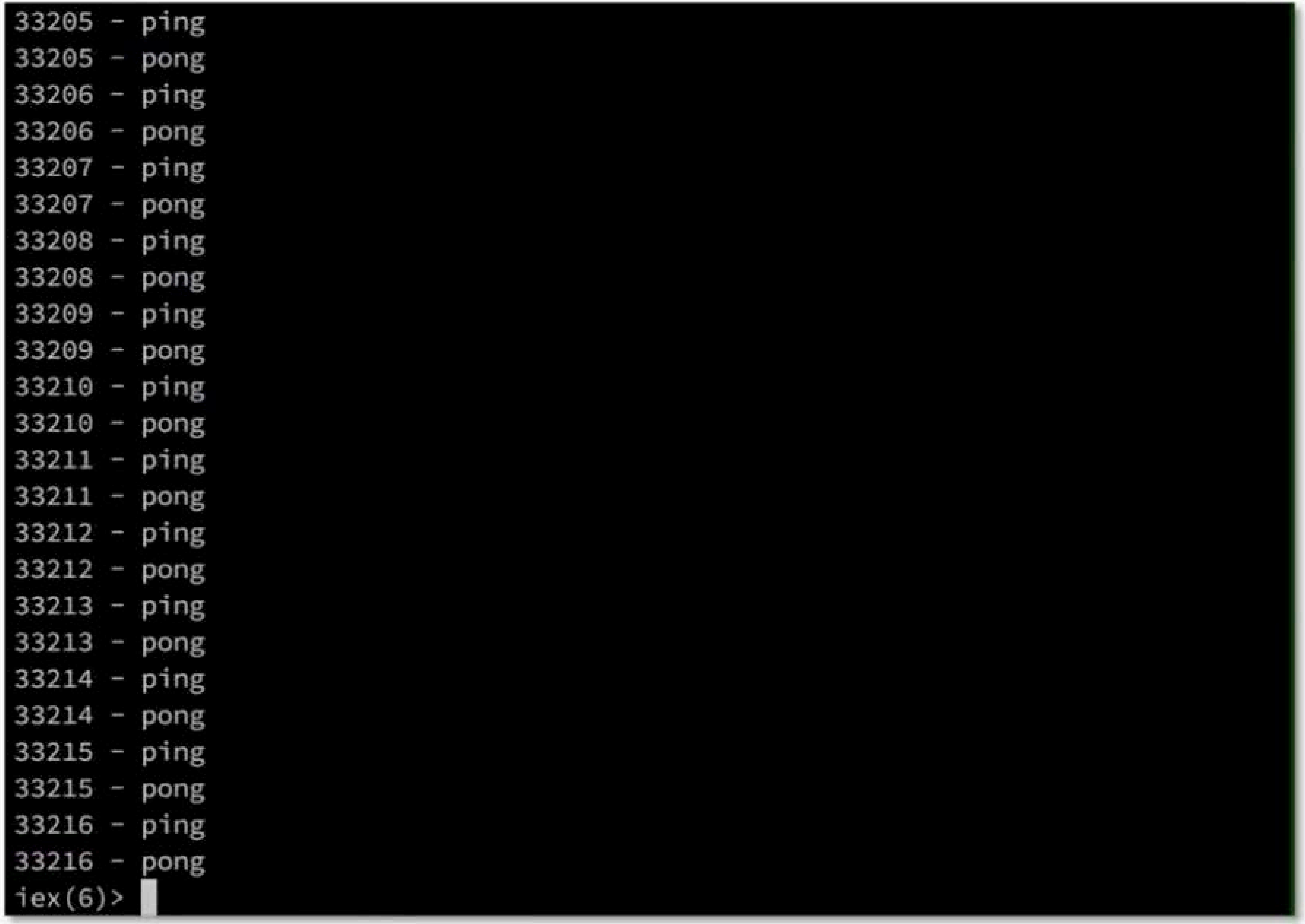
```
❶ * 331 ping_pong.ex  Elixir  alchemist ⓨⓢⓟⓐⓚ   unix | 11: 5  All
:wq
```

Ping-Pong



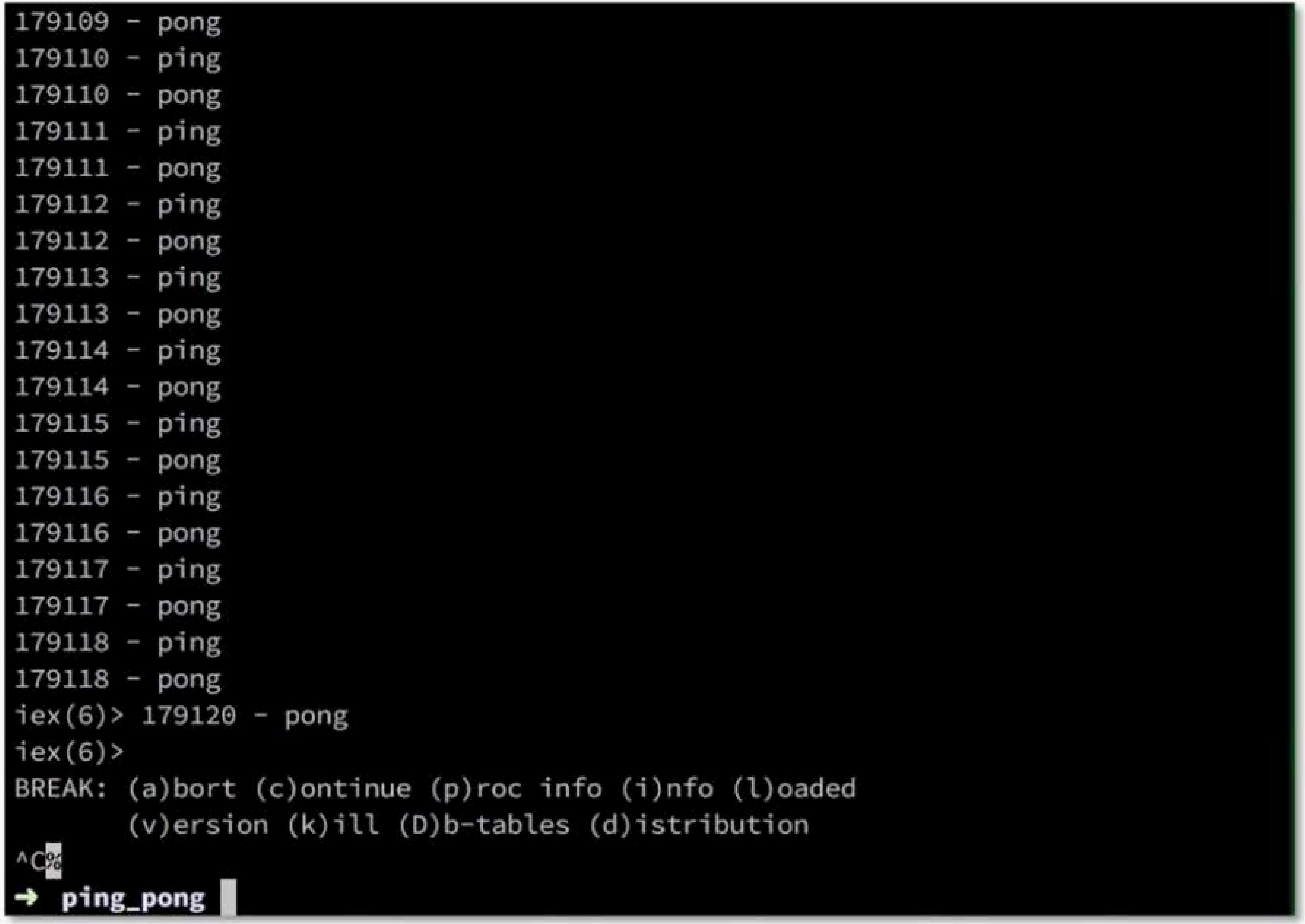
```
== Compilation error on file lib/ping_pong.ex ==
** (CompileError) lib/ping_pong.ex:7: undefined function stop/0
    (stdlib) lists.erl:1338: :lists.foreach/2
    (stdlib) erl_eval.erl:670: :erl_eval.do_apply/6

➜  ping_pong emacs -nw
➜  ping_pong iex -S mix
Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [
kernel-poll:false] [dtrace]

Compiling 1 file (.ex)
Generated ping_pong app
Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> ping = spawn(PingPong, :loop, [{0, :ping}])
0 - ping
#PID<0.138.0>
iex(2)> pong = spawn(PingPong, :loop, [{0, :pong}])
0 - pong
#PID<0.140.0>
iex(3)> Process.alive?(ping)
true
iex(4)> Process.alive?(pong)
true
iex(5)> send(
```

Ping-Pong



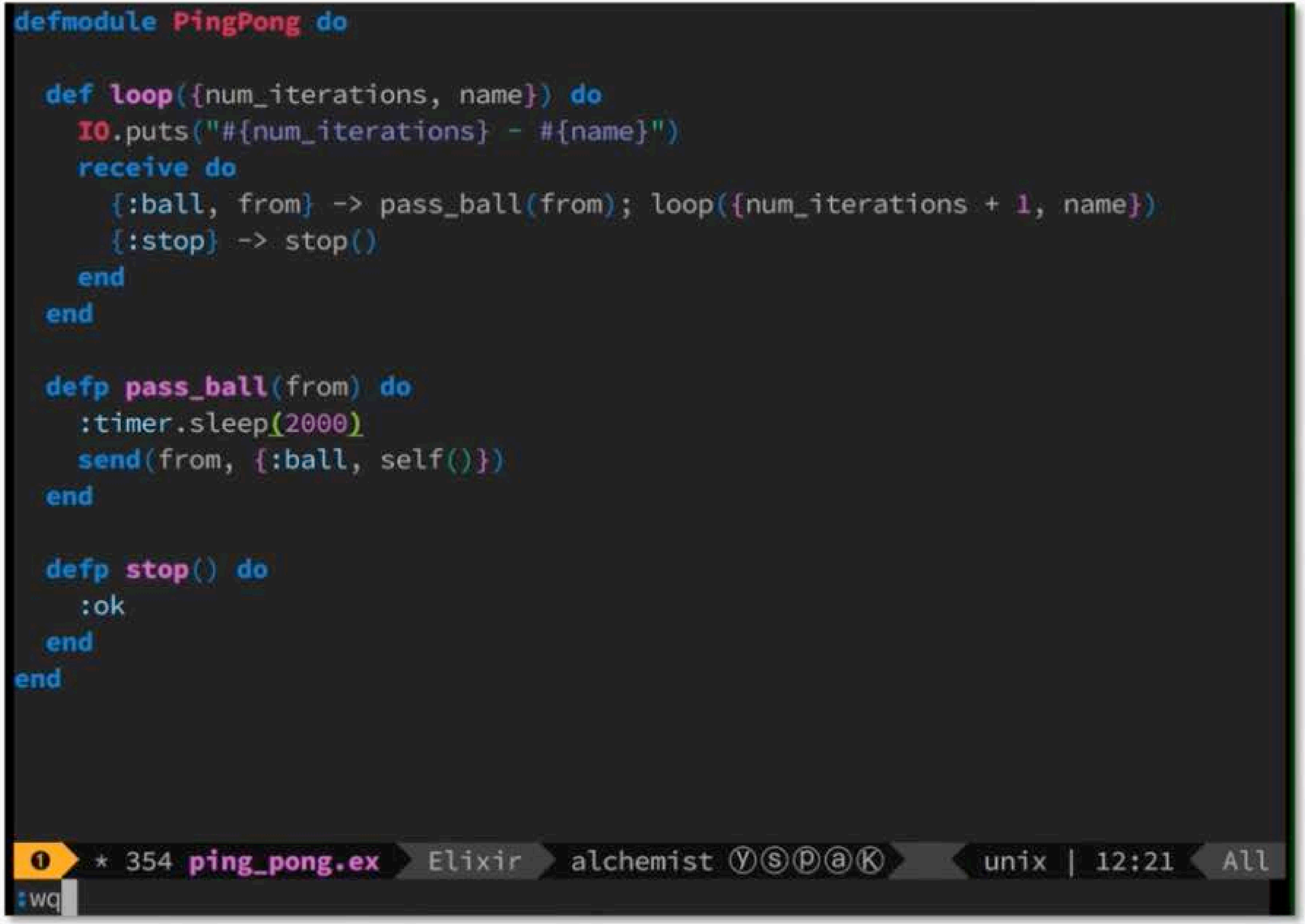
```
33205 - ping
33205 - pong
33206 - ping
33206 - pong
33207 - ping
33207 - pong
33208 - ping
33208 - pong
33209 - ping
33209 - pong
33210 - ping
33210 - pong
33211 - ping
33211 - pong
33212 - ping
33212 - pong
33213 - ping
33213 - pong
33214 - ping
33214 - pong
33215 - ping
33215 - pong
33216 - ping
33216 - pong
iex(6)>
```

Ping-Pong



```
179109 - pong
179110 - ping
179110 - pong
179111 - ping
179111 - pong
179112 - ping
179112 - pong
179113 - ping
179113 - pong
179114 - ping
179114 - pong
179115 - ping
179115 - pong
179116 - ping
179116 - pong
179117 - ping
179117 - pong
179118 - ping
179118 - pong
iex(6)> 179120 - pong
iex(6)>
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
^C%
➜ ping_pong
```

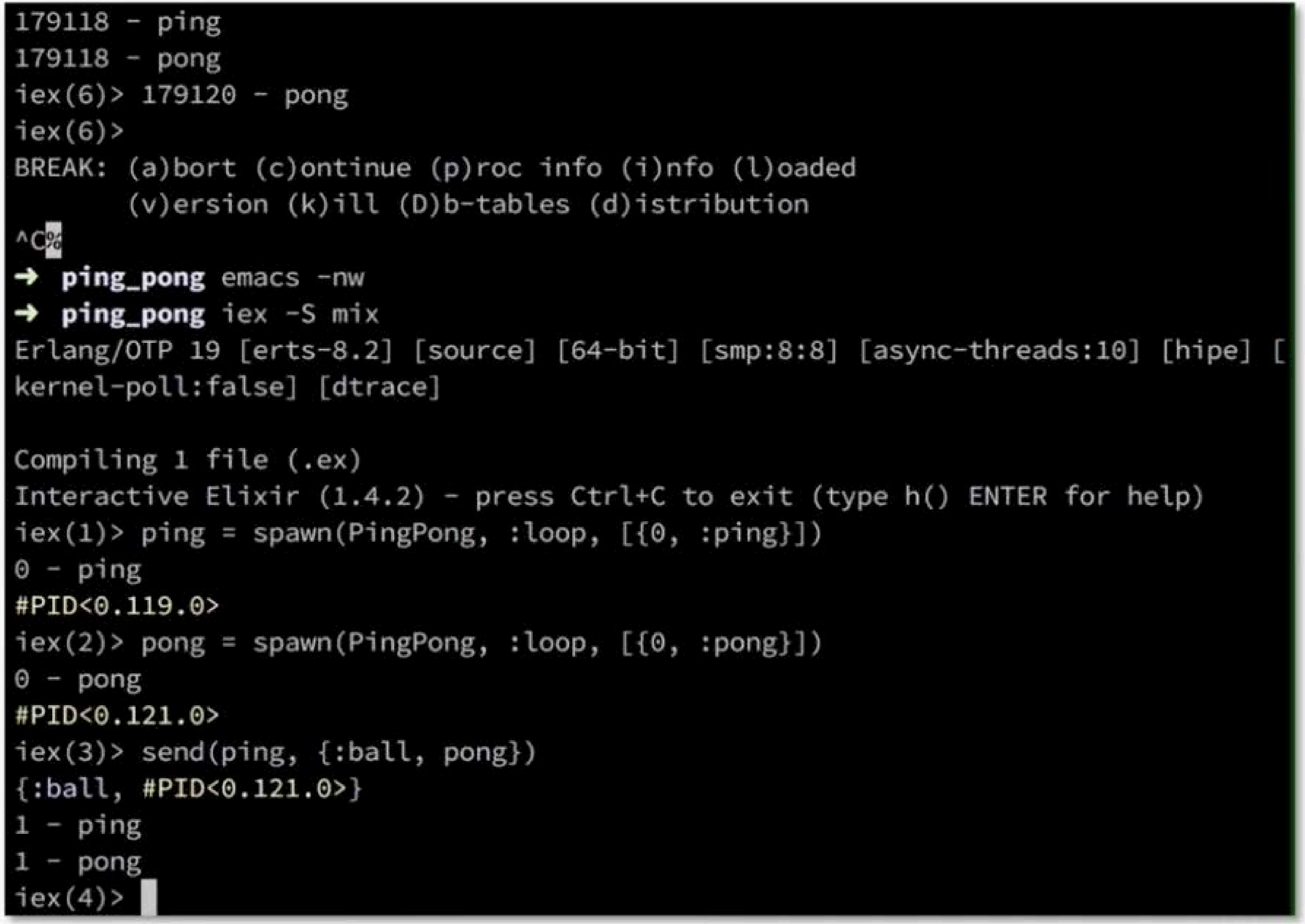
Ping-Pong



```
defmodule PingPong do

  def loop({num_iterations, name}) do
    IO.puts("#{num_iterations} - #{name}")
    receive do
      {:ball, from} -> pass_ball(from); loop({num_iterations + 1, name})
      {:stop} -> stop()
    end
  end

  defp pass_ball(from) do
    :timer.sleep(2000)
    send(from, {:ball, self()})
  end

  defp stop() do
    :ok
  end
end
```

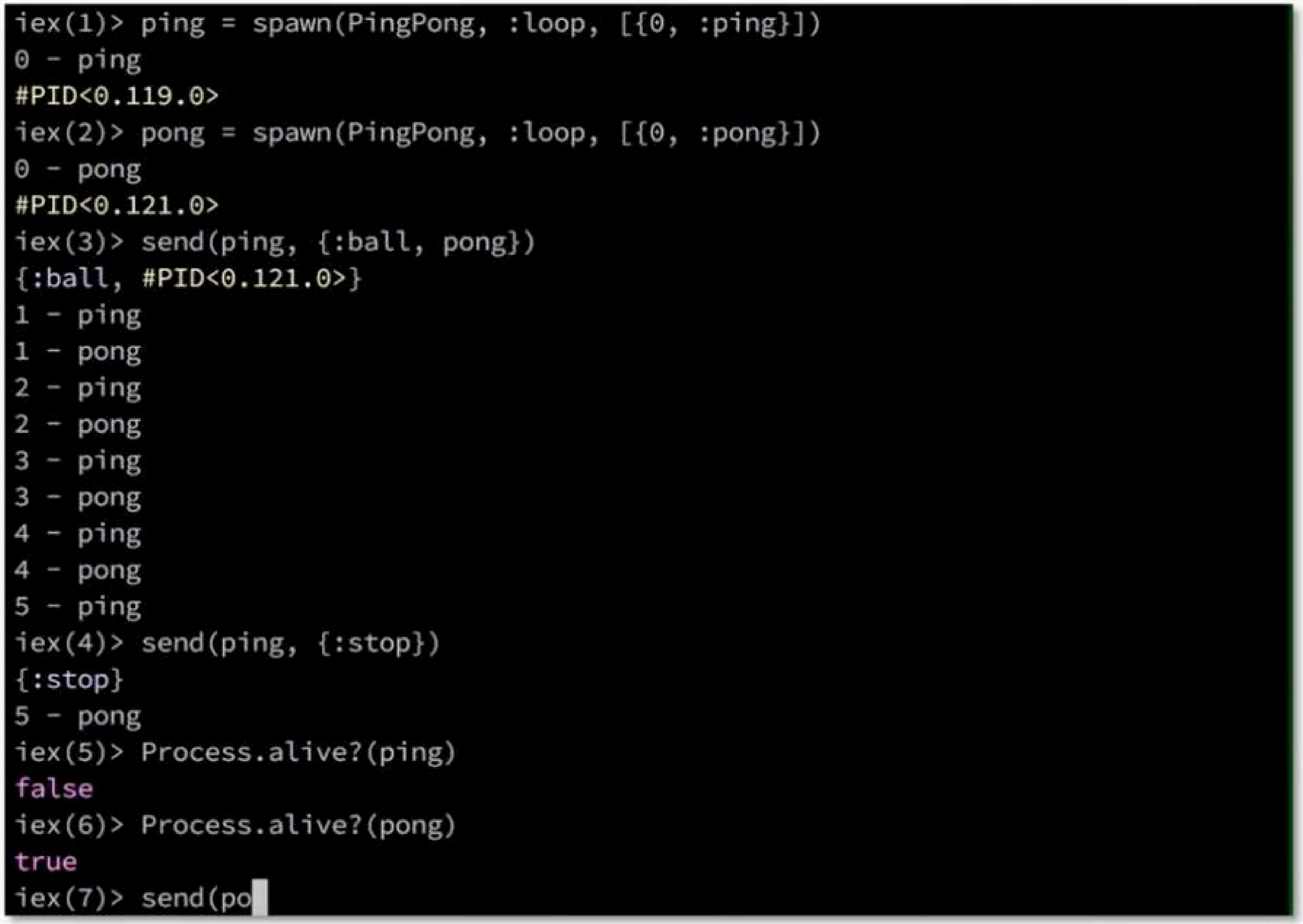
```
❶ * 354 ping_pong.ex   Elixir   alchemist ⓨⓢⓟⓐⓚ   unix | 12:21   All
:wq
```

Ping-Pong



```
179118 - ping
179118 - pong
iex(6)> 179120 - pong
iex(6)>
BREAK: (a)bort (c)ontinue (p)roc info (i)nfo (l)oaded
       (v)ersion (k)ill (D)b-tables (d)istribution
^C%
➜  ping_pong emacs -nw
➜  ping_pong iex -S mix
Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [
kernel-poll:false] [dtrace]

Compiling 1 file (.ex)
Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> ping = spawn(PingPong, :loop, [{0, :ping}])
0 - ping
#PID<0.119.0>
iex(2)> pong = spawn(PingPong, :loop, [{0, :pong}])
0 - pong
#PID<0.121.0>
iex(3)> send(ping, {:ball, pong})
{:ball, #PID<0.121.0>}
1 - ping
1 - pong
iex(4)>
```

Ping-Pong

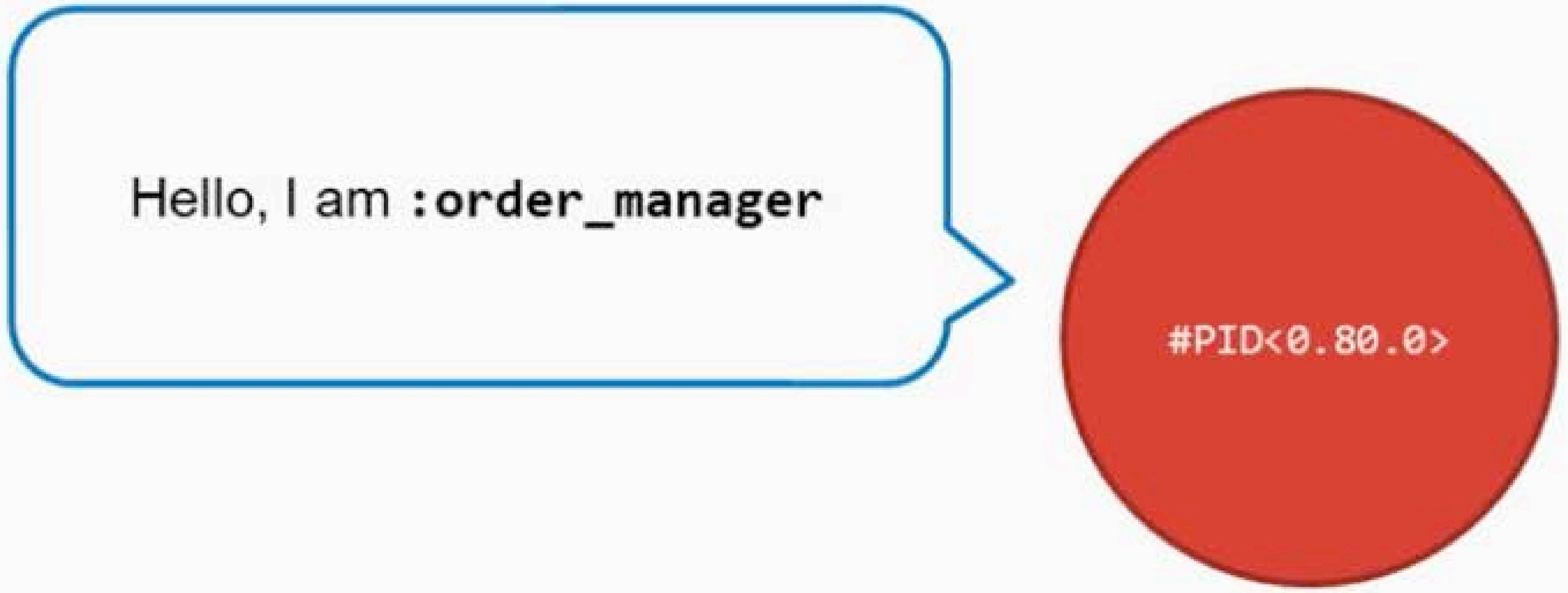


```
iex(1)> ping = spawn(PingPong, :loop, [{0, :ping}])
0 - ping
#PID<0.119.0>
iex(2)> pong = spawn(PingPong, :loop, [{0, :pong}])
0 - pong
#PID<0.121.0>
iex(3)> send(ping, {:ball, pong})
{:ball, #PID<0.121.0>}
1 - ping
1 - pong
2 - ping
2 - pong
3 - ping
3 - pong
4 - ping
4 - pong
5 - ping
iex(4)> send(ping, {:stop})
{:stop}
5 - pong
iex(5)> Process.alive?(ping)
false
iex(6)> Process.alive?(pong)
true
iex(7)> send(po
```

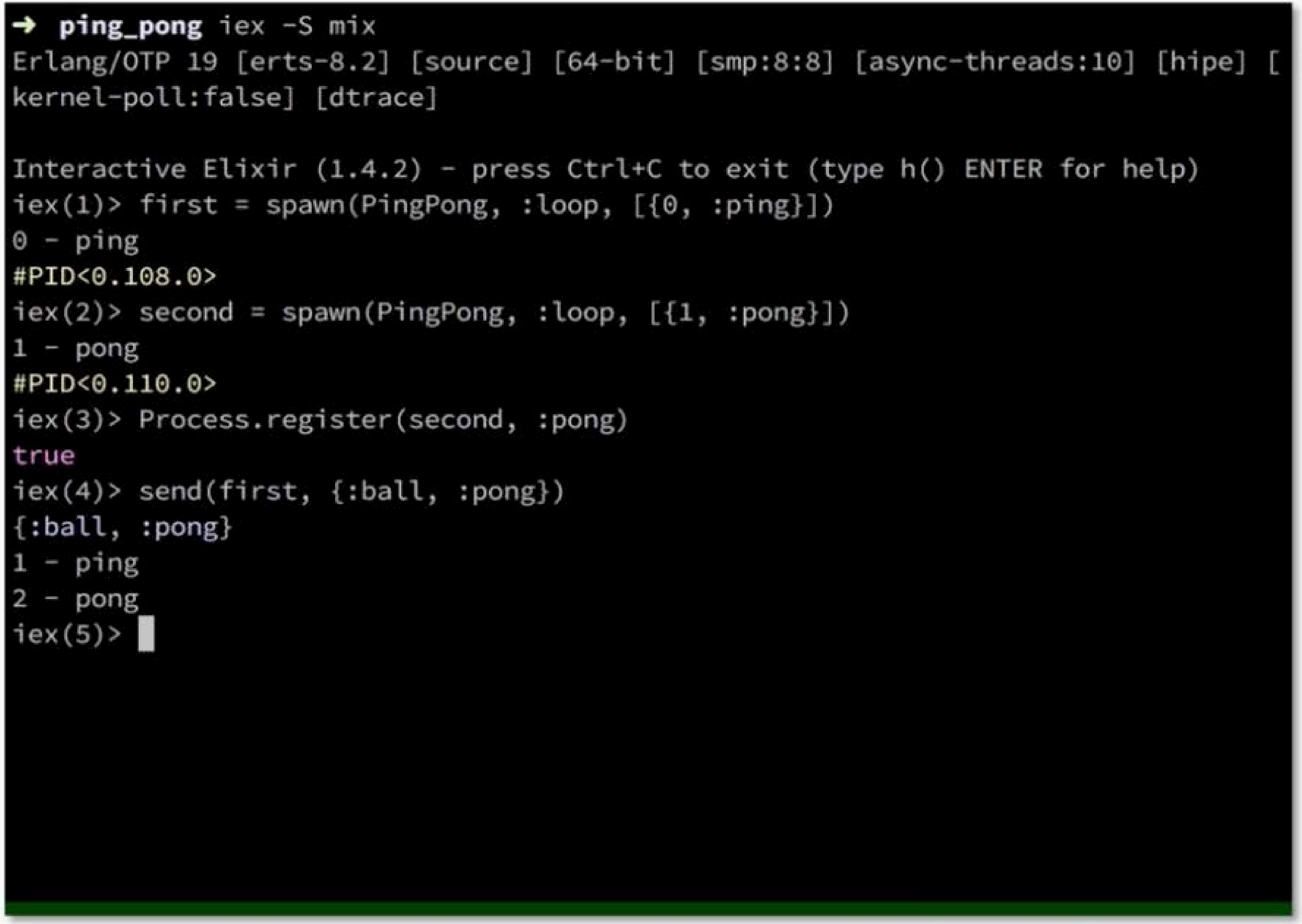
Naming Processes

We can give unique names to processes, which makes it easier to call them without a reference.

`Process.register/2`



Ping-Pong – Naming Processes



```
➜ ping_pong iex -S mix
Erlang/OTP 19 [erts-8.2] [source] [64-bit] [smp:8:8] [async-threads:10] [hipe] [
kernel-poll:false] [dtrace]

Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> first = spawn(PingPong, :loop, [{0, :ping}])
0 - ping
#PID<0.108.0>
iex(2)> second = spawn(PingPong, :loop, [{1, :pong}])
1 - pong
#PID<0.110.0>
iex(3)> Process.register(second, :pong)
true
iex(4)> send(first, {:ball, :pong})
{:ball, :pong}
1 - ping
2 - pong
iex(5)>
```

Summary

- Explored launching and linking
- Discussed messaging