Relating Channel- and Actor-based Models of Concurrent Programming

Simon Fowler
University of Edinburgh

3DT and Friends, May 2016

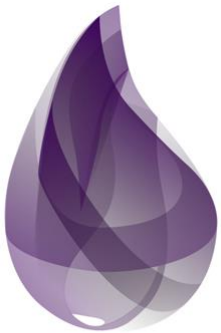
EPSRC Centre for Doctoral Training in Pervasive Parallelism

EPSRC Engineering and Physical Sciences Research Council

THE UNIVERSITY of EDINBURGH informatics

Communication-Centric Programming Languages

Go

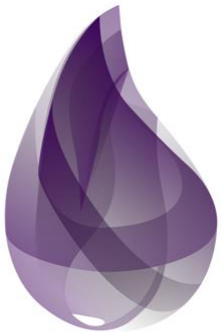


elixir

Communication-Centric Programming Languages

Go

ERLANG



elixir

Channels

Actors

Differences between the Actor Model and Communicating Sequential Processes (CSP)

6

2

When we look at the Actor Model and Communicating Sequential Processes we see that they are both trying to do concurrency based on message passing, yet they are distinct.

(We see implementations of the CSP Model in go-lang's goroutines (and Clojure's core.async) and the Actor Model in Scala's Akka toolkit)

I'm trying to get a simple list of the differences between the Actor Model and CSP. So far I have:

- actors message passing is asynchronous, CSP message passing is synchronous
- actors are composable, CSP is not (necessarily)
- actors always have unbounded non-determinism, CSP may have bounded or unbounded non-determinism
- actors have variable topology whereas CSP has fixed topology
- actors have the principle of locality, CSP does not have locality
- actors are designed around their behaviour, CSP doesn't not necessarily have this

Is this correct? Is there anything I'm missing?

Differences between the Actor Model and Communicating Sequential Processes (CSP)

6

2

Akka Go (programming language) Scala

How are Akka actors different from Go channels?

I don't know anything about actor pattern however I do know goroutines and channels in Go. How are two related to each other?

3 Answers

Roland Kuhn, Akka Tech Lead, Typesafe Inc.
11.8k Views • Upvoted by Ngoc Dao, Creator of Xitrum Scala web framework
Roland is a Most Viewed Writer in Akka.

Disclaimer: I have not used Go, hence my knowledge on that part is limited.

Go Channels model closely the semantics of Communicating sequential processes (Hoare, 1978) while Akka actors implement the Actor model (Hewitt, 1973). Both describe independent processes which communicate via message passing. The main difference is that a message exchange is synchronous in CSP (i.e. a "touch point" of the execution of two processes where they hand over the message) whereas it is completely

Differences between the Actor Model and Communicating Sequential Processes (CSP)

6

2

Akka Go (programming language) Scala

How are Akka actors different from Go channels?

I don't know anything about actor pattern however I do know goroutines and channels in Go. H

3 An

Discl

Go C
(Hoa
descr
differ
execu

Is Scala's actors similar to Go's coroutines?

41

25

If I wanted to port a Go library that uses GoRoutines, would Scala be a good choice because its inbox/akka framework is similar in nature to coroutines?

scala go

share improve this question

edited Jun 7 '14 at 23:17
Benjamin
10.7k • 14 • 87 • 163

asked Mar 24 '14 at 22:01
loyalflow
2,914 • 9 • 51 • 107

Clojure's core.async library is probably a closer fit to go routines than akka. Other than that, your answer is largely subjective to what the developer would be willing to use/learn. – dhable Mar 24 '14 at 23:08

@Dan Not really subjective, I'm looking for a 1:1 feature comparison so porting is a no brainer as oppose to re-writing the library b/c the language differences are so drastic. But you probably have a point... – loyalflow Mar 25 '14 at 20:57

you can look at github.com/rssh/scala-gopher for go-like CSP primitives in scala. – rssh Oct 17 '14 at 22:25

add a comment

Differences between the Actor Model and Communicating Sequential Processes (CSP)

6

2

Akka Go (programming language) Scala

How are Akka actors different from Go channels?

I don't know anything about actor pattern however I do know goroutines and channels in Go. H

3 An

Discl

Go C
(Hoa
descr
differ
execu

Is Scala's actors similar to Go's coroutines?

41

25

If I wanted to port a Go library that uses GoRoutines, would Scala be a good choice because its inbox/akka framework is similar in nature to coroutines?

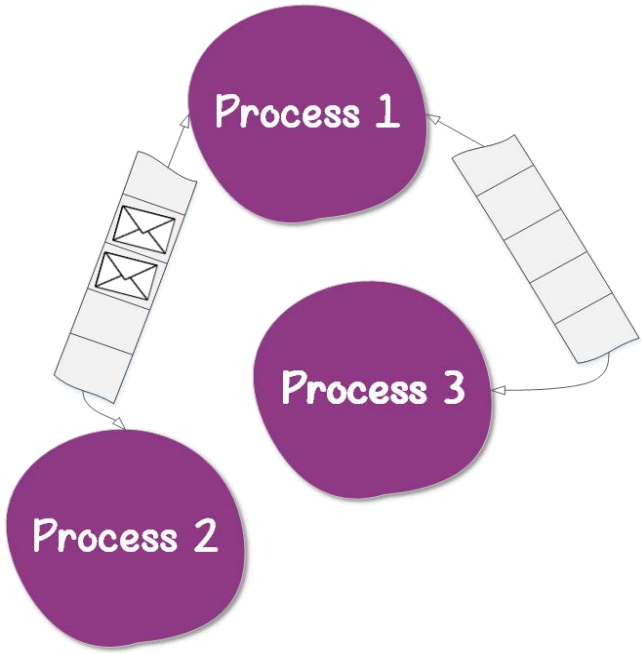
Actors and Hopac

The subject of Actors seems to come up often when discussing Hopac. Both Hopac and actors are models of message passing concurrency, but what are actors and what kind of similarities and differences do actors and Hopac have?

What are actors?

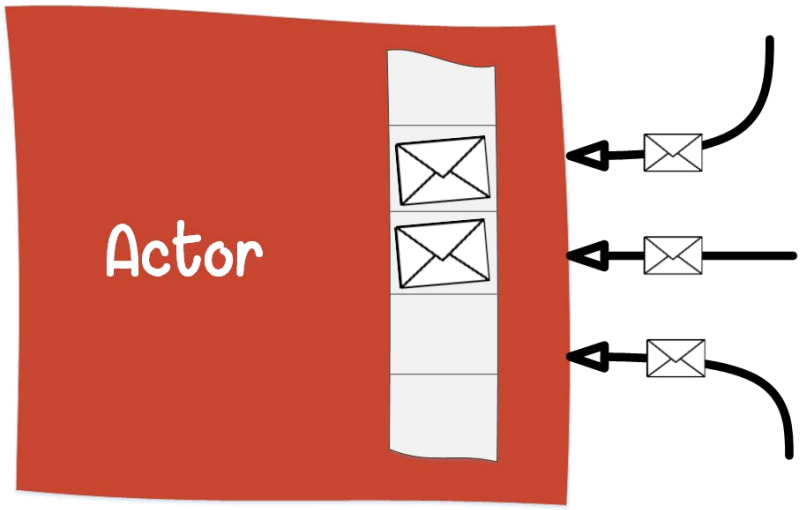
It is somewhat difficult to pin down what exactly actors are, because there are many different incarnations of actors. Fo example, Erlang is often called an actor language and Akka could be called an actor library. While there are similarities between those two, there are also fundamental differences between them and countless other incarnations of ``actors".

One thing that is common to many incarnations of the actor model is that the actor concept within them combines both thread of execution and a message queue or mailbox of some kind. The thread of execution and the mailbox are implic tied together so that, within the thread of an actor, a *receive* statement of some form implicitly takes messages from the mailbox associated with the actor.



Channels

- Multiple lightweight processes
- (Typed) **buffers** between them
- Each process may have access to **multiple channels**
- Processes can both **read** from, and **write** to channels



Actors

- Processes have a single **incoming** message queue, or **mailbox**
- Addressed by **process ID**: can send to process IDs, but only read from **own** mailbox

Communication-Centric Languages: **Actors**

- Processes have a single **incoming** message queue, or **mailbox**
- Addressed by **process ID**: can send to process IDs, but only read from **own** mailbox


```
main() ->
  Pid = spawn(addtwo, add_two,
                [self()]),
  Pid ! 39,
  Pid ! 3,
  receive
    Result ->
      io:format("~p~n", [Result])
  end.

add_two(MainPid) ->
  receive
    Arg1 ->
      receive
        Arg2 ->
          MainPid ! (Arg1 + Arg2)
      end
  end.
```

Picture: http://learnyousomeerlang.com/the-hitchhikers-guide-to-concurrency

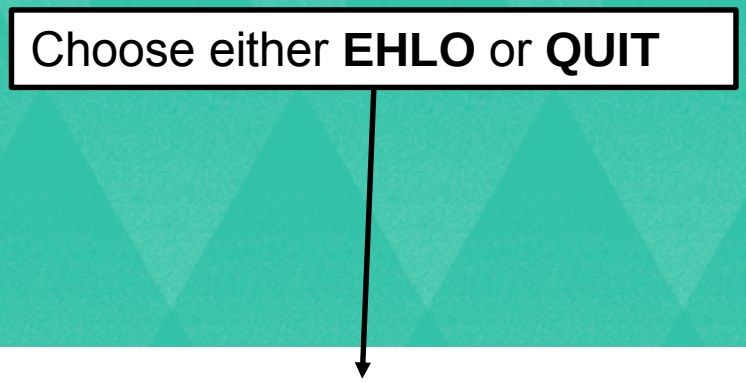
Communication-Centric Languages: Typed Channels

- Lightweight Processes
- (Typed) buffers between them

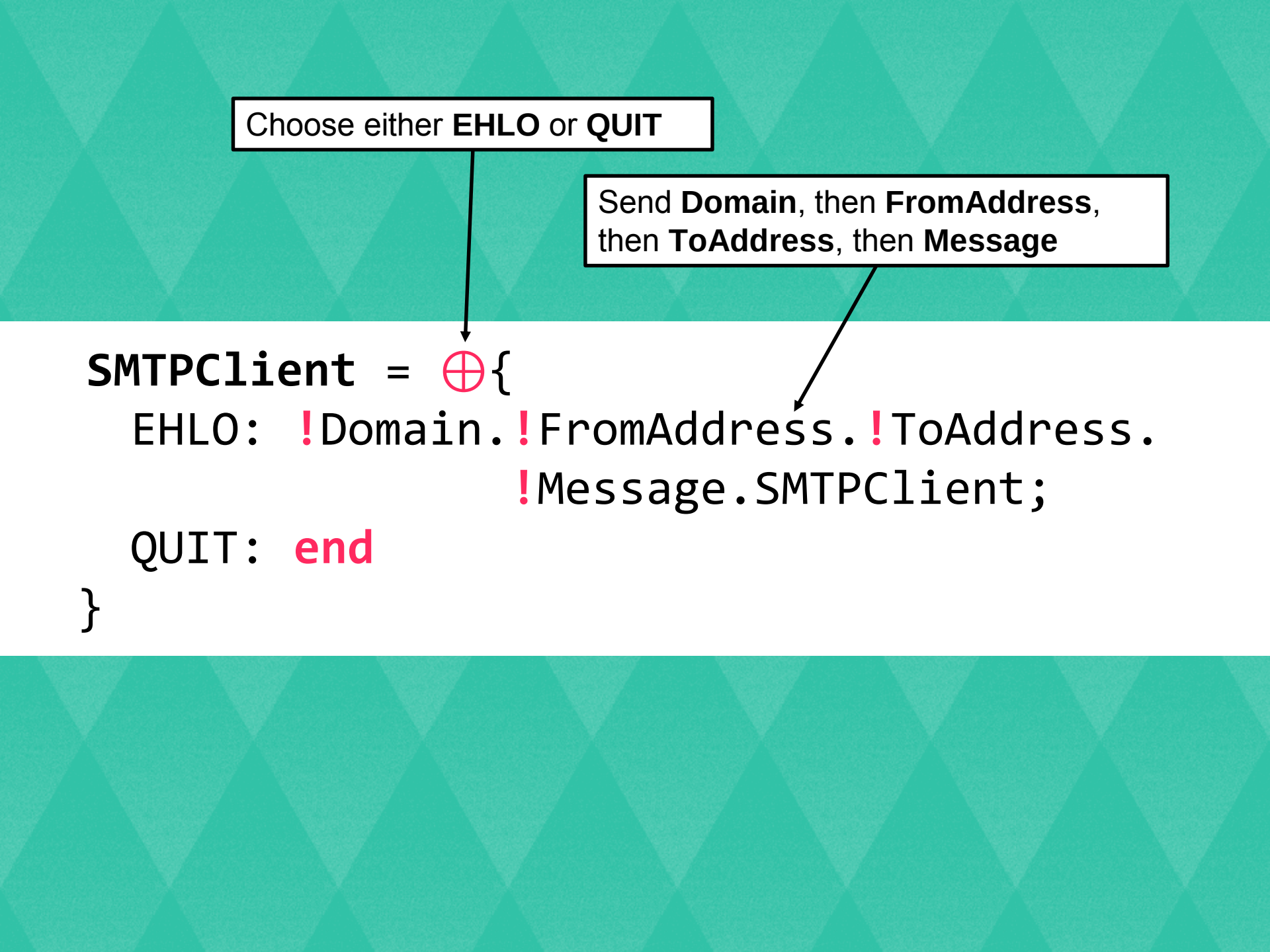
```
func main() {
  ch := make(chan int)
  go addTwo(ch)
  ch <- 39
  ch <- 3
  res := <- ch
  fmt.Println(res)
}

func addTwo(ch chan int) {
  x := <- ch
  y := <- ch
  ch <- x + y
}
```

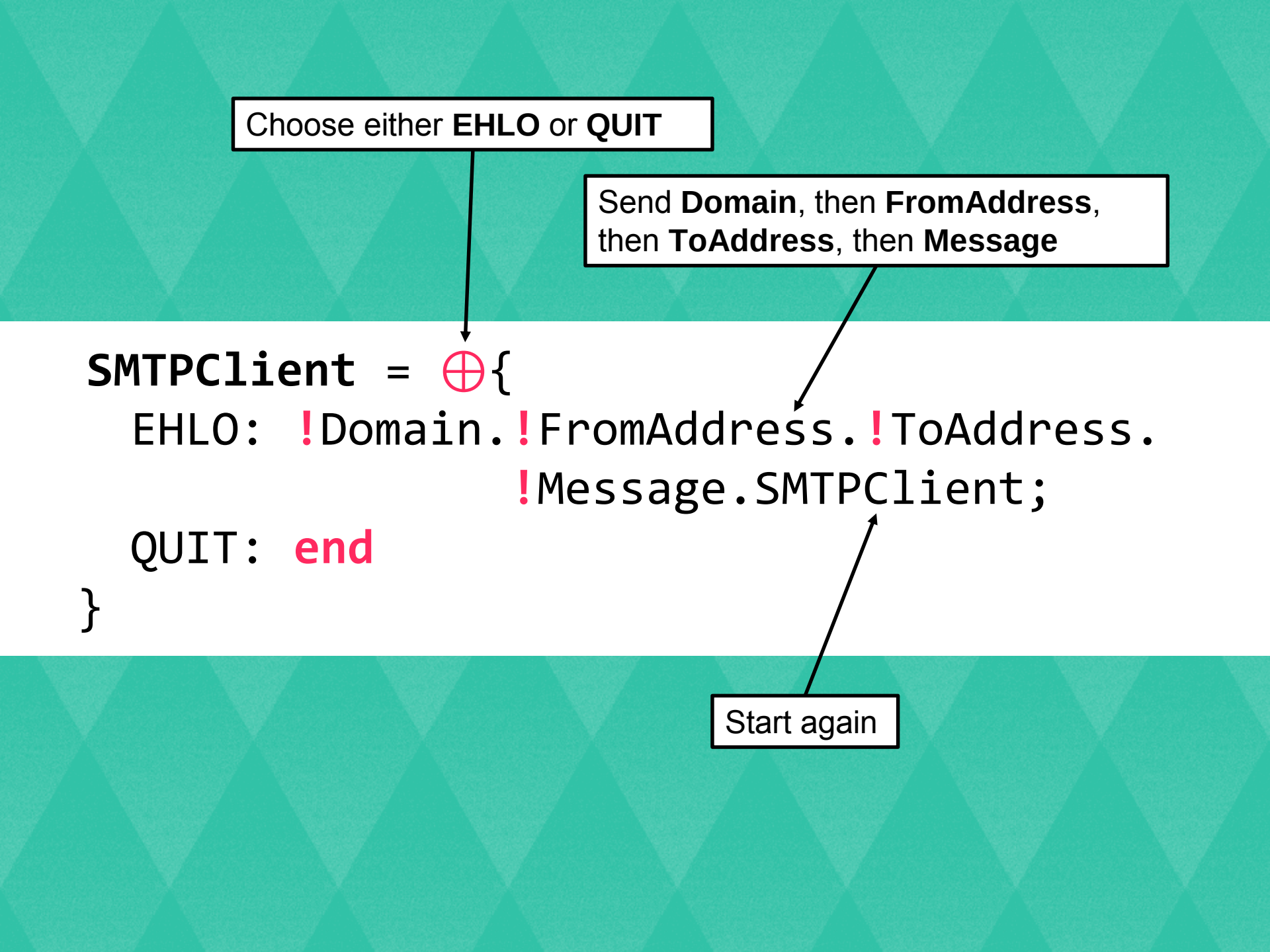
Session types extend typed channels to allow conformance to **protocols** to be checked at compile-time.

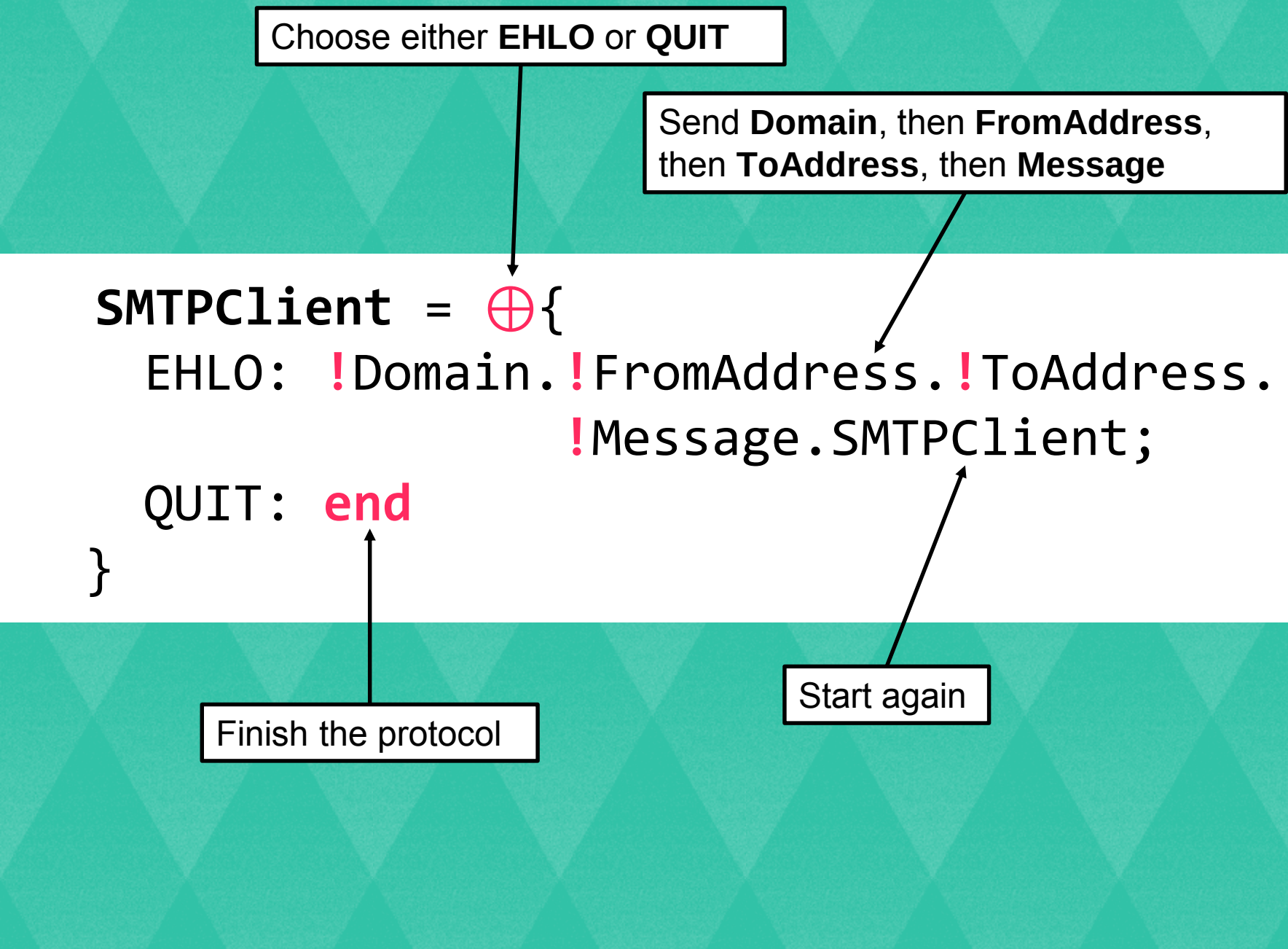
```
SMTPClient = ⊕{
  EHLO: !Domain.!FromAddress.!ToAddress.
                !Message.SMTPClient;
  QUIT: end
}
```

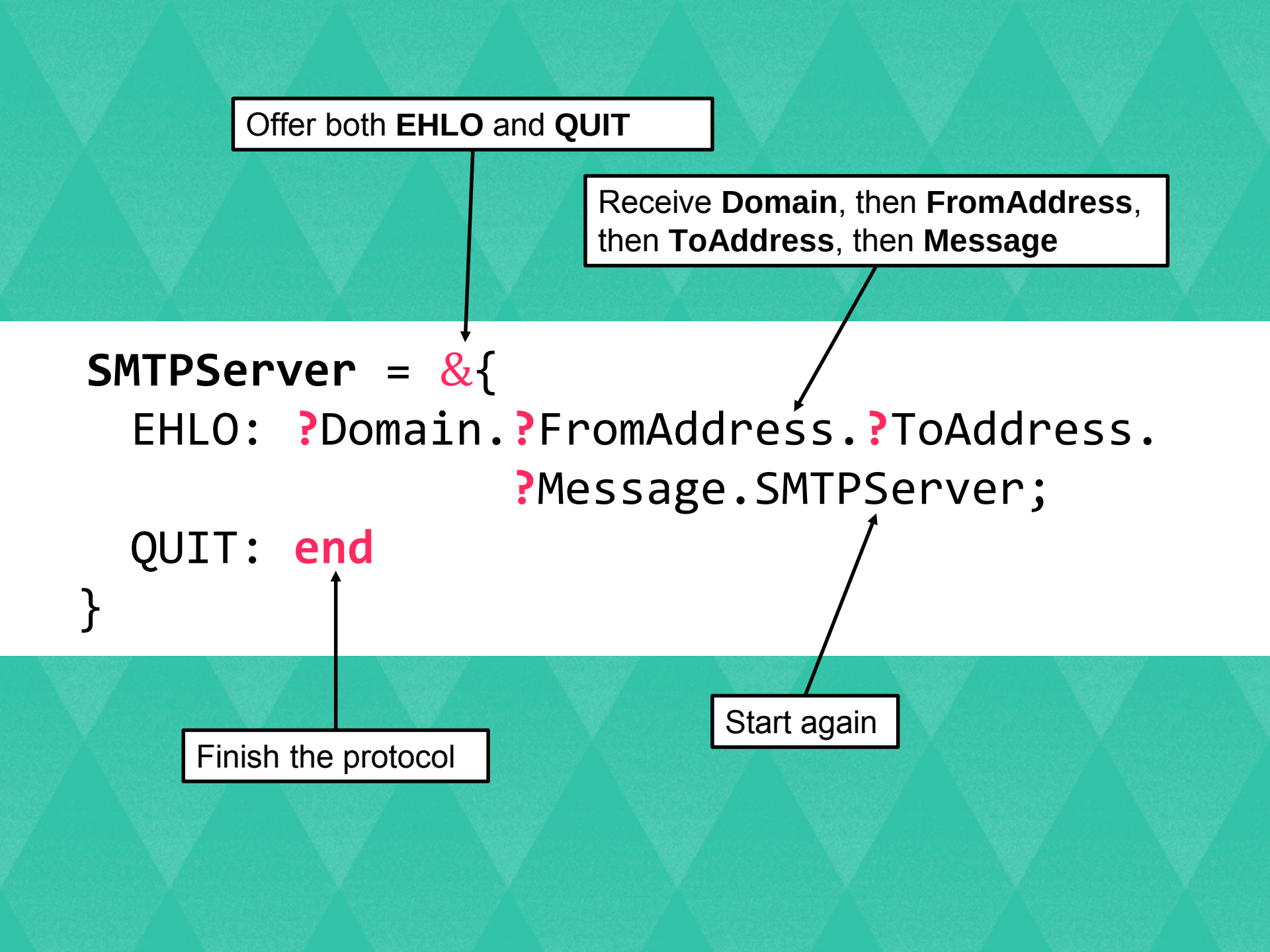



```
SMTPClient = ⊕{
  EHLO: !Domain.!FromAddress.!ToAddress.
                !Message.SMTPClient;
  QUIT: end
}
```


Choose either **EHLO** or **QUIT**
Send **Domain**, then **FromAddress**, then **ToAddress**, then **Message**
SMTPClient = ⊕{
EHLO: !Domain.!FromAddress.!ToAddress.
!Message.SMTPClient;
QUIT: end
}


Choose either **EHLO** or **QUIT**
Send **Domain**, then **FromAddress**, then **ToAddress**, then **Message**
SMTPClient = ⊕{
EHLO: !Domain.!FromAddress.!ToAddress.
!Message.SMTPClient;
QUIT: end
}
Start again


Choose either **EHLO** or **QUIT**
Send **Domain**, then **FromAddress**, then **ToAddress**, then **Message**
SMTPClient = ⊕{
EHLO: !Domain.!FromAddress.!ToAddress.
!Message.SMTPClient;
QUIT: end
}
Finish the protocol
Start again


Offer both **EHLO** and **QUIT**
Receive **Domain**, then **FromAddress**, then **ToAddress**, then **Message**
SMTPServer = &{
EHLO: ?Domain.?FromAddress.?ToAddress.
?Message.SMTPServer;
QUIT: end
}
Finish the protocol
Start again

Research questions

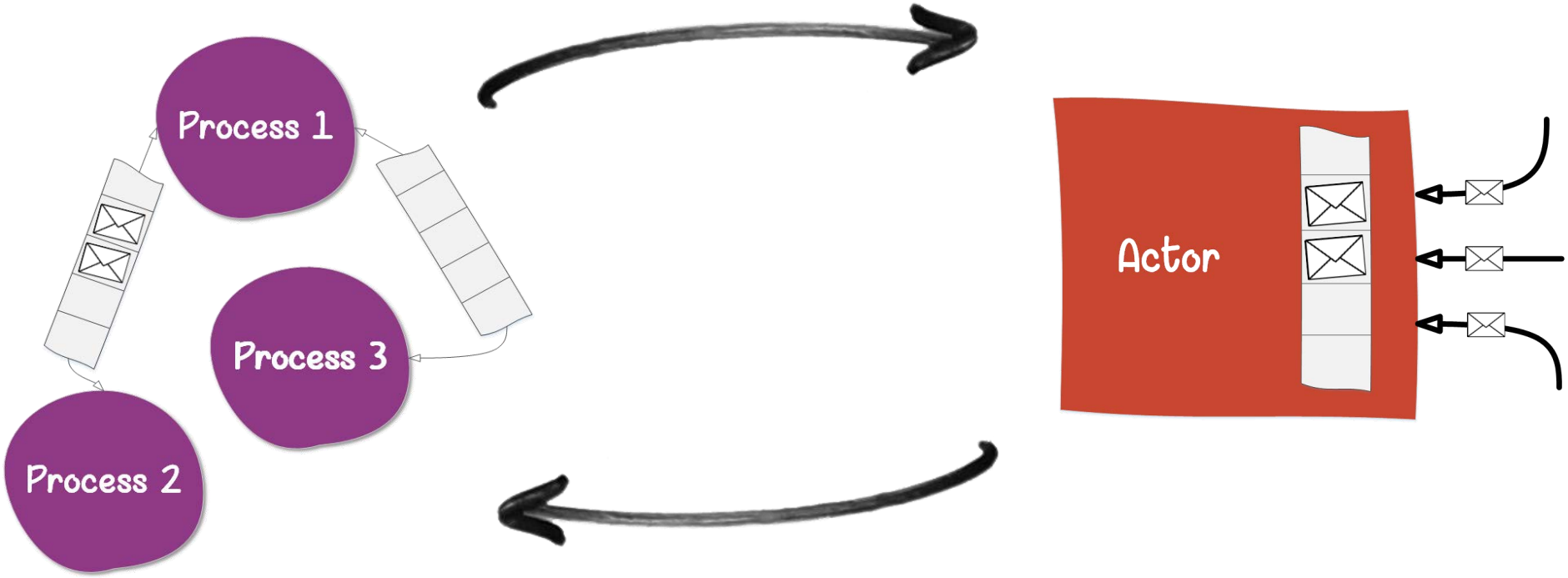
- How do we **model** these communication-centric languages?
- How do we **formally define** translations between them?
  - What **properties** do the translations between them have?
- Can the translations inform the design of **session-typed actor calculi**?

Approach: Concurrent λ-calculi

+

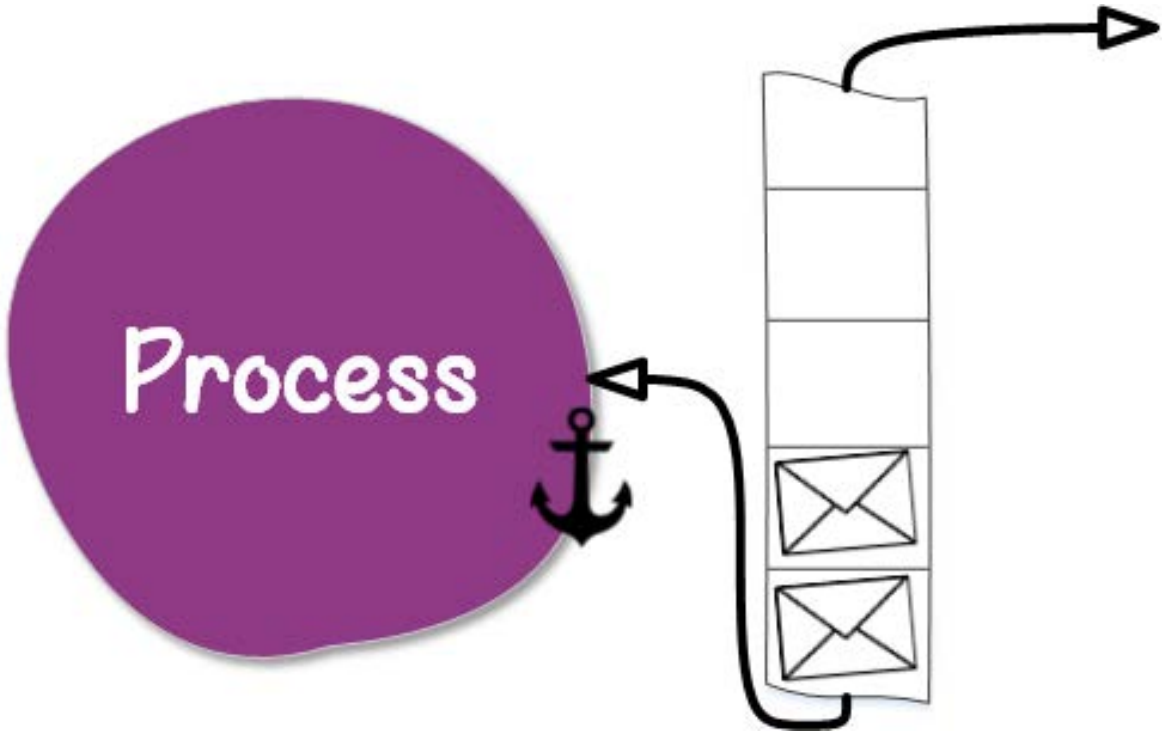
Term language
Term typing rules
Term semantics

Configurations
Configuration typing rules
Configuration semantics


Process 1
Process 3
Process 2
Actor

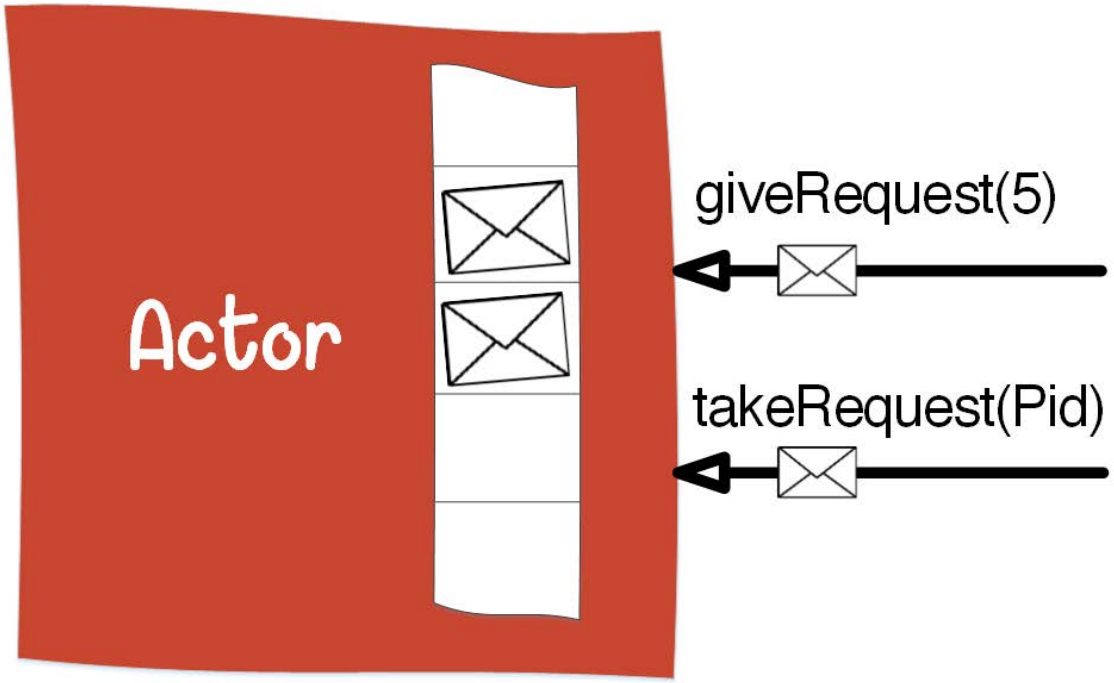
Actors as Channels

Key idea: emulate mailbox with a channel, and “thread it through” the translation.



Channels as Actors

Key idea: write an actor process which provides a channel interface.



So far:

- Calculi for simply-typed channels and actors (λ_{ch} and λ_{act})
- Calculi for session-typed channels and actors (GV and $\lambda_{s\text{-}act}$)
- Translations ($\lambda_{act} \rightarrow \lambda_{ch}$, $\lambda_{ch} \rightarrow \lambda_{act}$, $\lambda_{s\text{-}act} \rightarrow$ GV)
- A lot of proofs!

Still to do:

- $\lambda_{s\text{-}act}$ is not as expressive as GV: what needs to be added?
- Generalising: a taxonomy of session-typed calculi with one localised endpoint