# Session Types in Applied Type System

Hanwen Wu, Boston University

Nov 2015, NEPLS

# Overview

- **Session Types** enforce correct implementation of communication protocols in distributed programming. Global progress is guaranteed.
- **ATS** is a statically typed functional language with DML-style dependent types and linear types.
- **Session types** can be readily implemented **in ATS**.

# Session Types

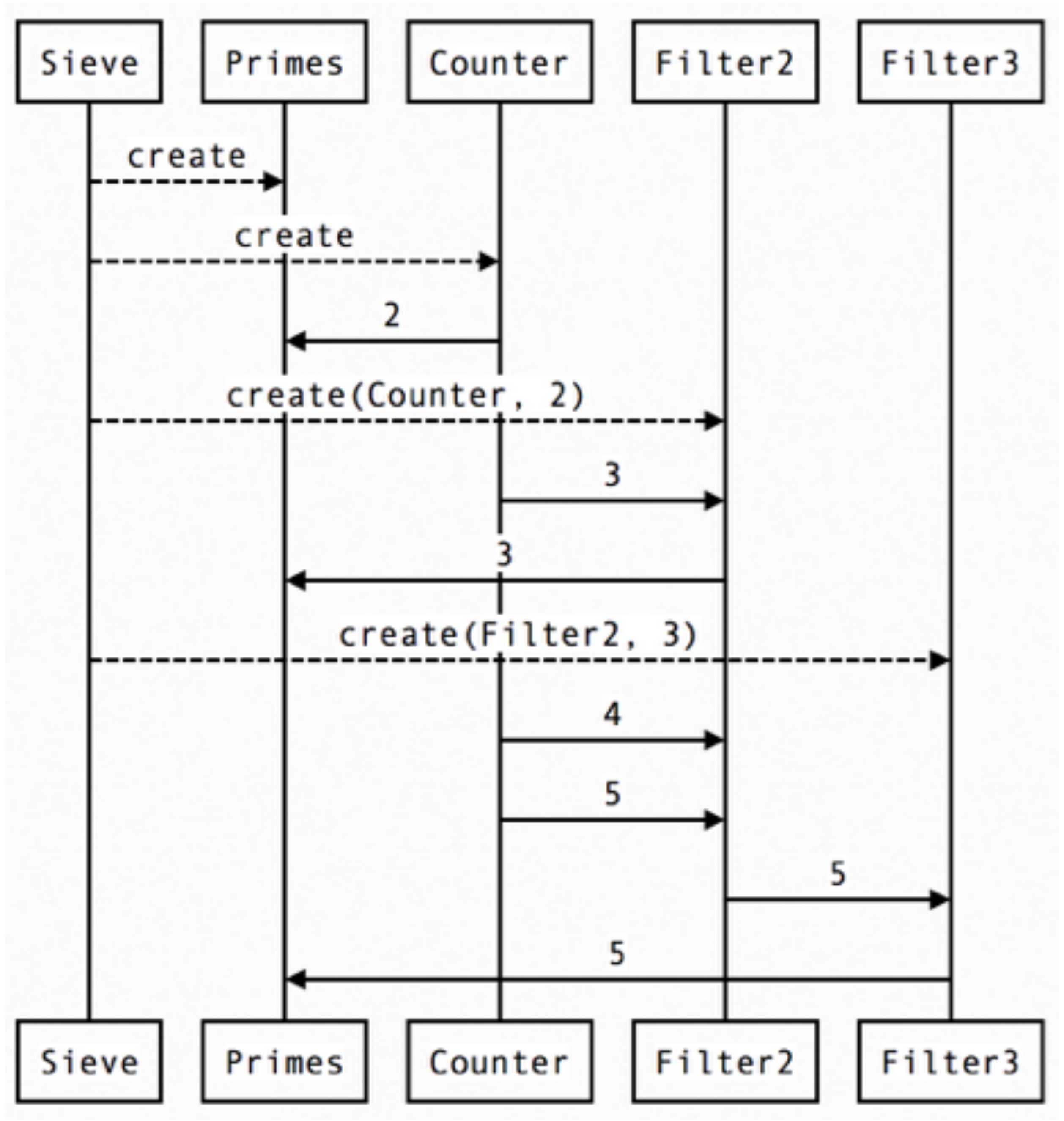
pseudo code:

```
counter_loop(counter, N):
  send(counter, N)
  counter_loop(counter, N+1)

filter_loop(in, out, P):
  N = recv(in)
  if N % P != 0
  then send(out, N)
  else filter_loop(in, out, P)

filter(in, P):
  mod_n = new channel
  spawn filter_loop(in, mod_n, P)
  return mod_n

primes_loop(in, primes):
  N = recv(in)
  send(primes, N)
  filter_n = filter(in, P)
  sieve_loop(filter_n, primes)
```



# Session Types

types for selected functions:

```
fun counter (int): channeg (rpt int)
fun cloop (chanpos (rpt int), int): void

fun filter (channeg (rpt int), int): channeg (rpt int)
fun floop (chanpos (rpt int), channeg (rpt int), int): void

fun primes (): channeg (rpt int)
```

**Note:**

- Positive channels are endpoints hold by the server side.
- Negative channels are endpoints hold by the client side.

## Demo

https://github.com/steinwaywhw/nepls-15-demo
requires ATS/Erlang/Elixir to be installed

## Advantages

- Global progress (deadlock-free) is guaranteed.
- Session protocol is guaranteed to be correctly implemented through type checking.
- Resource leaking is prevented through linear typed channels.
- Extensive support of distributed computing through compiling into Erlang.
- ATS co-programming with Erlang.
- Asynchronous session.
- Session type is part of the language instead of an embedding. Utilizing everything provided by ATS, e.g. dependent type (DML-style), linear type, and proofs.

# Q&A

Thanks!
for more info
https://github.com/steinwaywhw/nepls-15-demo

# Backup

- Session types in ATS supports:
    - dependent types, linear types
    - high-order sessions (mobile sessions)
    - dyadic session for now
- Implementation details
    - Sessions are not symmetric. two endpoints are denoted negative(client) and positive(server) respectively. Though symmetric can be implemented in ATS, too.
    - Global progress is proved based on a formalization of multi-threaded linear lambda calculus extended with channels.
    - A channel is a process in Erlang.