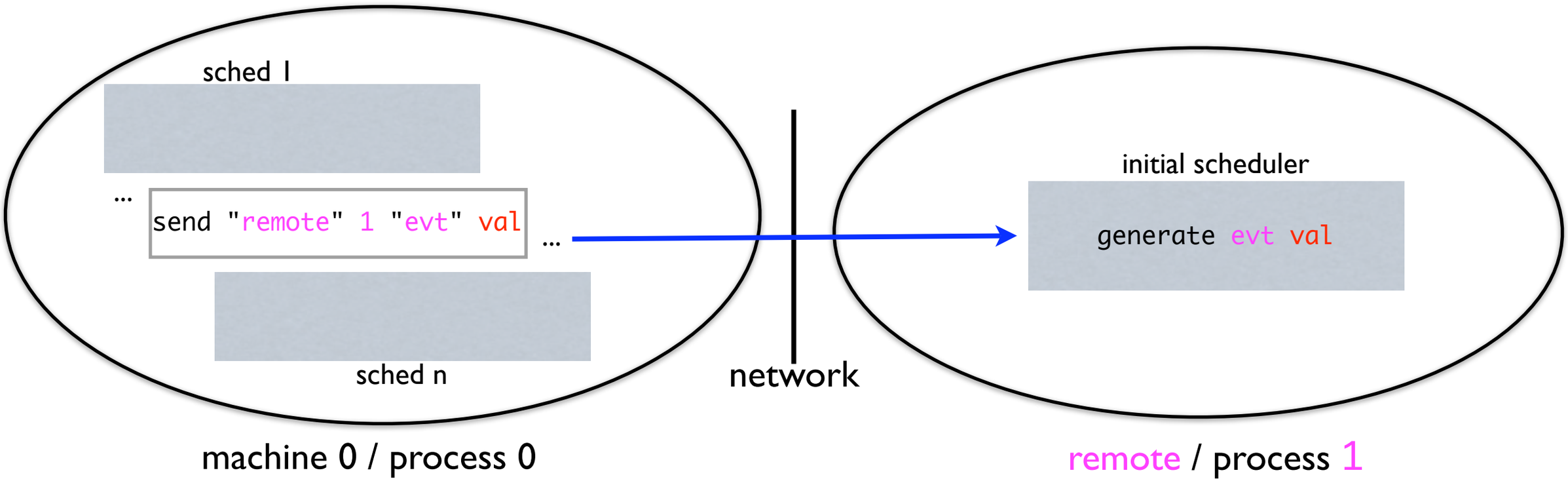
Distributed Programming with FunLoft

Frédéric Boussinot
INRIA Méditerranée
http://www-sop.inria.fr/members/Frederic.Boussinot

March 2009

ANR-08-EMER-010

Multiprocess Model



- Asynchronous execution of the processes
- Execution of send is not instantaneous
- Marshalling/unmarshalling of val
- Possible unknown events (run-time error)
- Possible ill-typed values (run-time error)

Multiprocess Model - 2

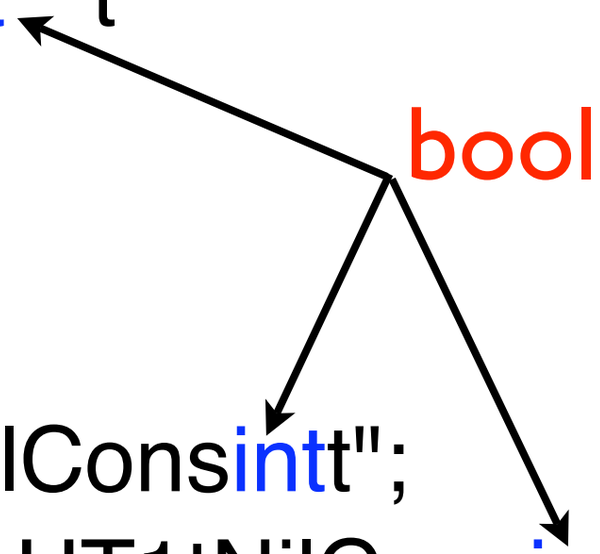
- Send can take time or fail: static analysis to check that it is always called while unlinked
- Elimination of data races: val should not be mutable (or embed a ref or an array; enforced via effects in the type system)
- Val should be defined consistently in both sites: signatures of types are associated to transmitted values
- Implementation using basic RPC (with XDR for marshalling/unmarshalling values)

Type Signatures

- Objective: detect errors due to re-compilation of a process with different type definitions (safety concern). Not for detecting cheating attempts!
- Very primitive solution: signature = definition
- type t = Nil | Cons of int * t
  type t1 = U | T1 of t

  Produces:

  char* t_signature = "tNilConsintt";
  char* t1_signature = "t1UT1tNilConsintt";



Pingpong

- Two symmetric processes exchanging a ball in turn through the network
- The simple solution doesn't work:

```
loop
    begin
        await ball;
        trace ();
        unlink send (target,"ball",())
    end
```

The ball can be lost, because of asynchrony

Pingpong - 2

Solution: launch the ball at the next instant

```
let ball = event

let trace (s,i) =
   begin
      print_string (s);
      print_int (i);
      print_string (" ! ");
      flush ();
   end

let module send_ball (target,num) =
   unlink
      send (target,num,"ball",())
```

```
let module play (msg,target,num) =
   let i = ref 0 in
      begin
         await ball;
         loop
            join
               begin
                  thread send_ball (target,num);
                  await ball;
                  i++;
                  trace (msg,!i);
                  cooperate;
               end
      end
```

```
let module main (argv) =
   let msg = !argv[0] in
   let target = !argv[1] in
   let num = string2int (!argv[2]) in
      thread play (msg,target,num)
```

Pingpong - 3

- No deadlock, no lost of balls
- Correct even with a unique process
- No memory leak

Conclusion

- Efficiency of transfers of signatures ?
- Detection of cheating attempts ?
- Interfacing with SugarCubes, ReactiveML, and HOP ?