Course 02158

Remote Operations

Hans Henrik Løvengreen

DTU Compute

Remote Operations

Client$_A$:

$\vdots$

$op_1(a_1, a_2, \ldots, a_k)$;

$\vdots$

Client$_B$:

$\vdots$

$op_2(a_1, a_2, \ldots, a_l)$;

$\vdots$

Server:

services

op $op_1(T_1, T_2, \ldots, T_k)$;

op $op_2(T_1, T_2, \ldots, T_l)$;

monitor

var $state : T$;

$op_1(p_1, p_2, \ldots, p_k)$

$\vdots$

$op_2(p_1, p_2, \ldots, p_l)$

$\vdots$

end;

Remote Operations

Client$_A$:

$\vdots$

send $REQ(op_1(a_1, a_2, \ldots, a_k))$

receive *reply*

$\vdots$

Client$_B$:

$\vdots$

send $REQ(op_2(a_1, a_2, \ldots, a_l))$

receive *reply*

$\vdots$

Server:

services

op $op_1(T_1, T_2, \ldots, T_k)$;

op $op_2(T_1, T_2, \ldots, T_l)$;

process *Server*

var *state* : T;

repeat

receive $REQ(op)$ **from** *client*

case *op* :

$op_1()$: ...

$op_2()$: ...

send *reply* **to** *client*

forever;

Modules [Andrews]

Client$_A$:

$\vdots$

$Mod.op_1(a_1, a_2, \ldots, a_k)$;

$\vdots$

Client$_B$:

$\vdots$

$Mod.op_2(a_1, a_2, \ldots, a_l)$;

$\vdots$

Server:

module *Mod*

op $op_1(T_1, T_2, \ldots, T_k)$;

op $op_2(T_1, T_2, \ldots, T_l)$;

body

$\vdots$

end *Mod*;

Modules — Remote Procedures

Client$_A$:

$\vdots$

$Mod.op_1(a_1, a_2, \ldots, a_k);$

$\vdots$

Client$_B$:

$\vdots$

$Mod.op_2(a_1, a_2, \ldots, a_l);$

$\vdots$

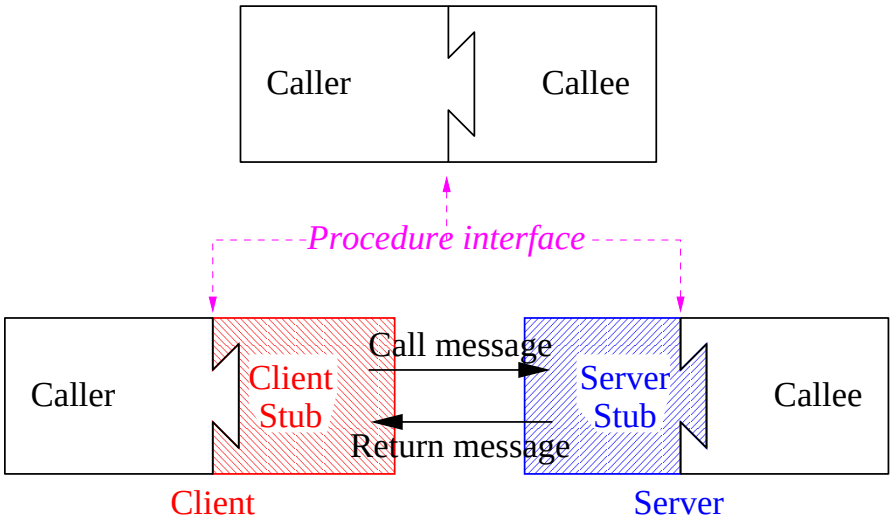
Server:

module *Mod*
  op $op_1(T_1, T_2, \ldots, T_k);$
  op $op_2(T_1, T_2, \ldots, T_l);$
body
  var *state* : $T;$
  proc $op_1(p_1, p_2, \ldots, p_k)$
    $\vdots$
  proc $op_2(p_1, p_2, \ldots, p_l)$
    $\vdots$
end *Mod*;

Bounded Buffer Module — RPC

- **module** *BoundedBuffer*
  op *deposit*(*T*);
  op *fetch*(**var** *T*);
body
  var *buf*[*n*] : *T*; *in*, *out*, *count* : *integer* := 0;
    full : *semaphore* := 0; *empty* : *semaphore* := *n*;
  proc *deposit*(*data* : *T*)
    P(*empty*);
    buf[*in*] := *data*; *in* = *in* + 1 **mod** *n*; *count*++;
    V(*full*);
  proc *fetch*(**var** *data* : *T*)
    P(*full*);
    data := *buf*[*out*]; *out* = *out* + 1 **mod** *n*; *count*−−;
    V(*empty*);
end *BoundedBuffer*;

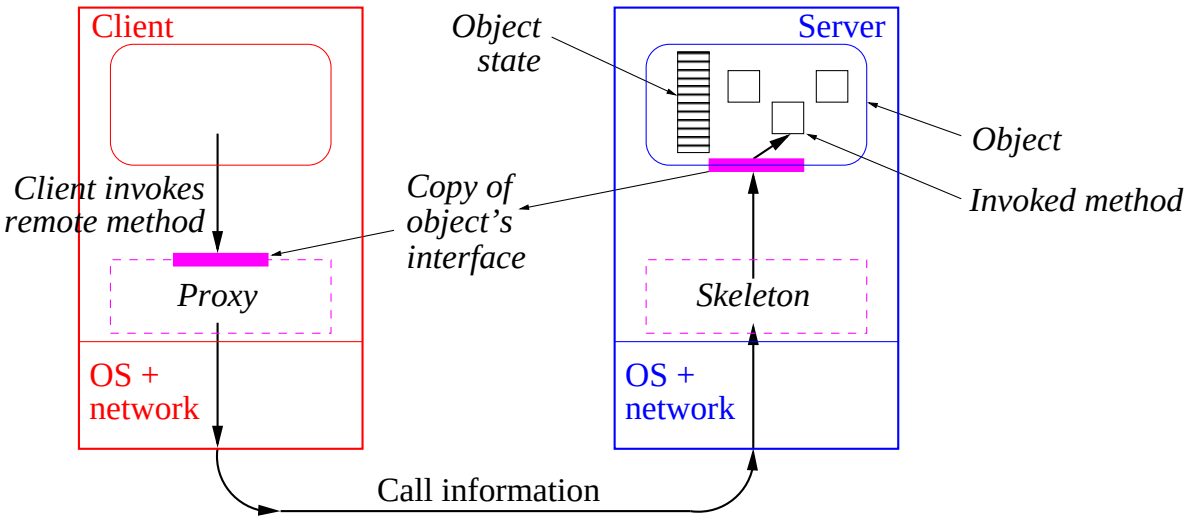
Remote Procedure Call (RPC)



- Call semantics: *Exactly-once*, *at-most-once*, *at-least-once*, *maybe*
- Stubs generated from *Interface Definition Language (IDL)* descriptions

Remote Object Invocation (ROI)

- RPC in an object oriented setting (aka *distributed objects*)



RPC/ROI Technologies

- *Sun RPC* on Unix systems
- *Bindings* in the Android Linux Kernel
- *Remote Method Invocation* in Java
- CORBA
- *Web Services*
- *REST APIs*
- *Microservices*

Beware

- Concurrent calls $\Rightarrow$ **concurrency issues remain** on the server side

Modules — Rendezvous

Client$_A$:

$\vdots$

$Mod.op_1(a_1, a_2, \ldots, a_k);$

$\vdots$

Client$_B$:

$\vdots$

$Mod.op_2(a_1, a_2, \ldots, a_l);$

$\vdots$

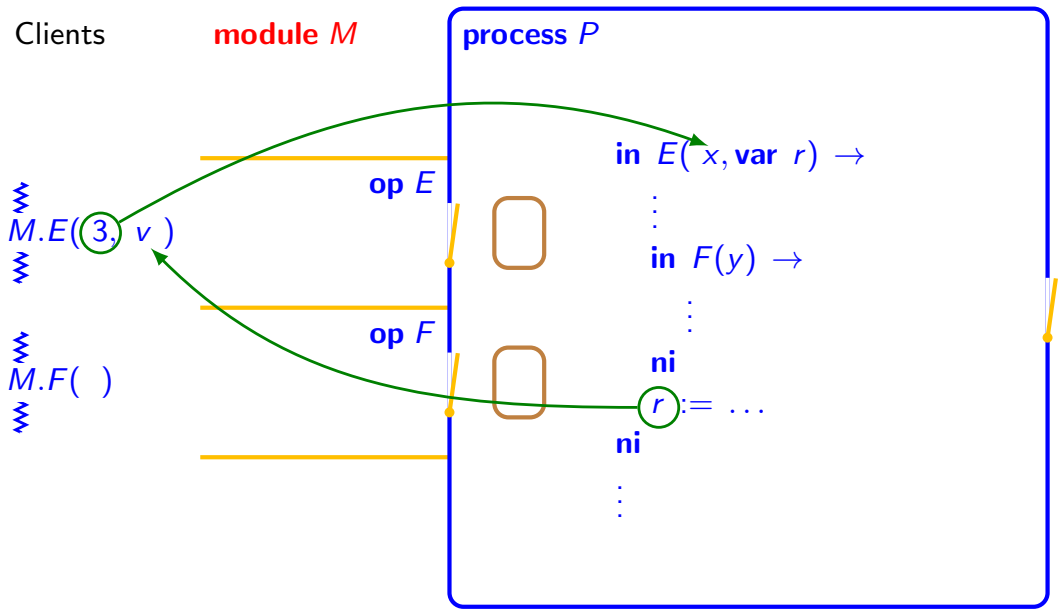
Server:

module *Mod*
  op $op_1(T_1, T_2, \ldots, T_k);$
  op $op_2(T_1, T_2, \ldots, T_l);$
body

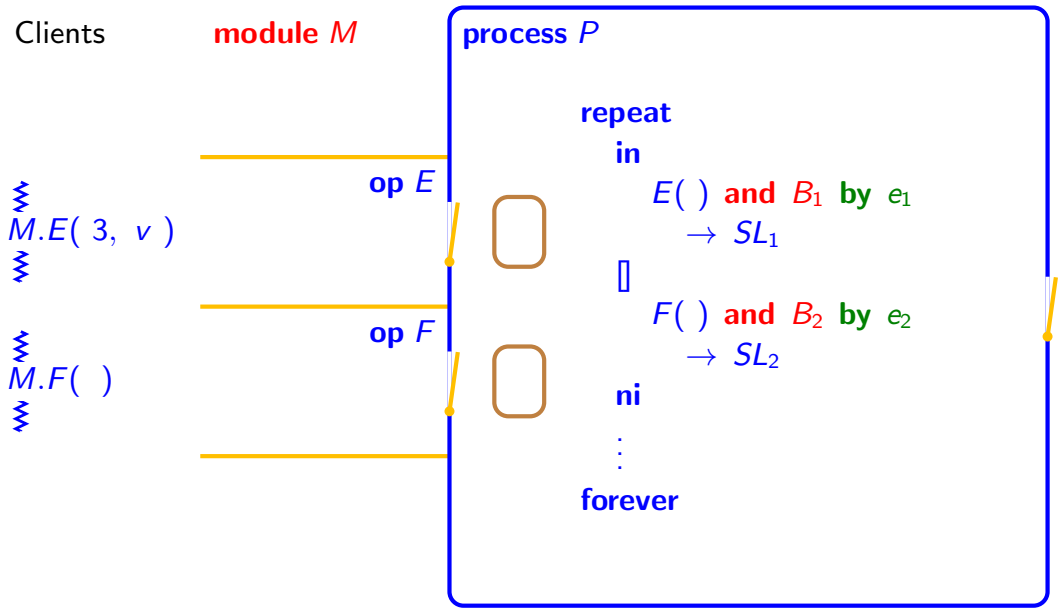
  process *Server*
    var *state* : T;
    repeat
      $\vdots$
      in $op_2(p_1, p_2, \ldots, p_l) \rightarrow$
          SL
      ni;
      $\vdots$
    forever;

end *Mod*;

Rendezvous



Rendezvous — selection



Bounded Buffer Module — Rendezvous

```
module BoundedBuffer
  op deposit(T);
  op fetch(var T);
body
  var buf[n] : T;
      in, out, count : integer := 0;

  process Buffer;
    repeat
      in deposit(item) and count < n →
            buf[in] := item;
            in = in + 1 mod n;  count++;
      [] fetch(var item) and count > 0 →
            item := buf[out];
            out = out + 1 mod n;  count--;
      ni
    forever;

end BoundedBuffer;
```

Time Module

```
module TimeServer
  op get_time() returns integer;
  op delay(int);
  op tick();
body
  var tod : integer := 0;

  process Timer;
    repeat
      in get_time()                          → return tod
      [] delay(waketime) and waketime ≤ tod  → skip
      [] tick()                              → tod := tod + 1
      ni
    forever;

end TimeServer;
```

Shortest-job-next Module

- **module** *SNJ_Allocator*
  op *request*(*time* : *integer*);
  op *release*();
  body
  var *free* : *boolean* := *true*;

  process *SJN*;
  repeat
  in *request*(*time*) **and** *free* **by** *time* → *free* := *false*
  [] *release*() → *free* := *true*
  ni
  forever;

  end *SJNAlloc*;

Modules — hybrid implementations

Client$_A$:

$\vdots$

$Mod.op_1(a_1, a_2, \ldots, a_k)$;

$\vdots$

Client$_B$:

$\vdots$

$Mod.op_2(a_1, a_2, \ldots, a_l)$;

$\vdots$

Server:

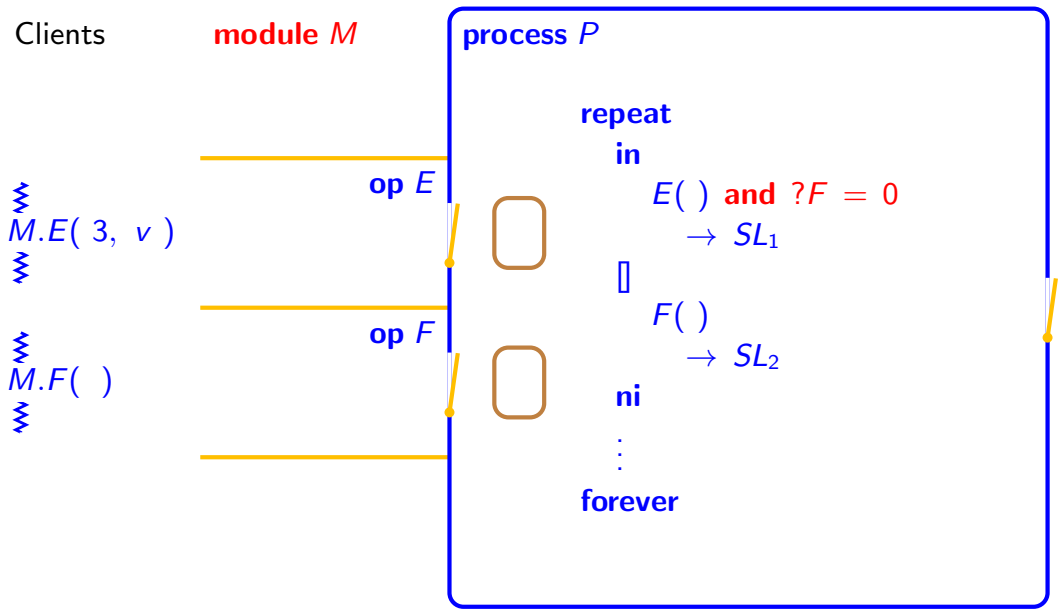
module *Mod*
op $op_1(T_1, T_2, \ldots, T_k)$;
op $op_2(T_1, T_2, \ldots, T_l)$;
body
var *state* : T;

proc $op_1(p_1, p_2, \ldots, p_k)$
$\vdots$

process *Server*
repeat
in $op_2(p_1, p_2, \ldots, p_l)$ →
SL
ni;
$\ldots$
forever;

end *Mod*;

Rendezvous — queue count



Reader/Writer Module

- **module** *ReadersWriters*
  op *read*(**var** *T*);
  op *write*(*T*);
  body
  op *startread*();
  op *endread*();
  var *val* : *T*;
  proc *read*(**var** *r* : *T*)
  ⋮
  process *Writer*
  ⋮
  end *ReadersWriters*;

Reader/Writer Module

- **proc** *read*(**var** r : T)
  startread();
  r := *val*;
  endread()

- **process** *Writer* =
  var *nr* : *integer* := 0;
  repeat
  in *startread*() **and** ?*write* = 0 → *nr* := *nr* + 1
  [] *endread*() → *nr* := *nr* − 1
  [] *write*(v : T) **and** *nr* = 0 → *val* := v
  while ?*startread* > 0 **do**
  in *startread*() → *nr* := *nr* + 1 **ni**
  ni
  forever;

Modules — multiple primitives

Client$_A$:

$\vdots$

call $Mod.op_i(a_1, a_2, \ldots, a_k)$;

$\vdots$

Client$_B$:

$\vdots$

send $Mod.op_j(a_1, a_2, \ldots, a_l)$;

$\vdots$

Server:

module *Mod*
op $op_1(T_1, T_2, \ldots, T_k)$;
op $op_2(T_1, T_2, \ldots, T_l)$;
body
var *state* : T;

proc $op_1(p_1, p_2, \ldots, p_k)$

$\vdots$

process *Server*
repeat
in $op_2(p_1, p_2, \ldots, p_l)$ →
SL
ni;
$\ldots$
forever;

end *Mod*;