Lecture 8 In Class 2/5/08

Note Title 2/5/2008

5.3.1 – example protocols

```
fun {NewPortObject2 Proc}
   Sin in
   thread for Msg in Sin do {Proc Msg} end end
   {NewPort Sin}
end
```

To create server

```
proc {ServerProc Msg}
   case Msg
   of calc(X ?Y) then ... statement binding Y end
   end
end
```

record syntax

label of record

```
Server = {NewPortObject2 ServerProc}
```

RPC — RMI

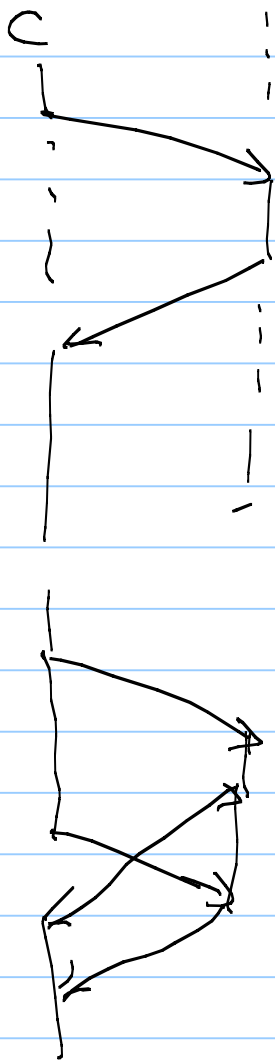
Client

Send requests to port - get replies on a dataflow variable

```
declare X Y
{Send Server calc(3, Y)}
{Wait Y}
```

Asynchronous RPC

```
{Send Server calc(3, Y)}
{Send Server calc(4, Y)}
   ... X ... Y ...
```



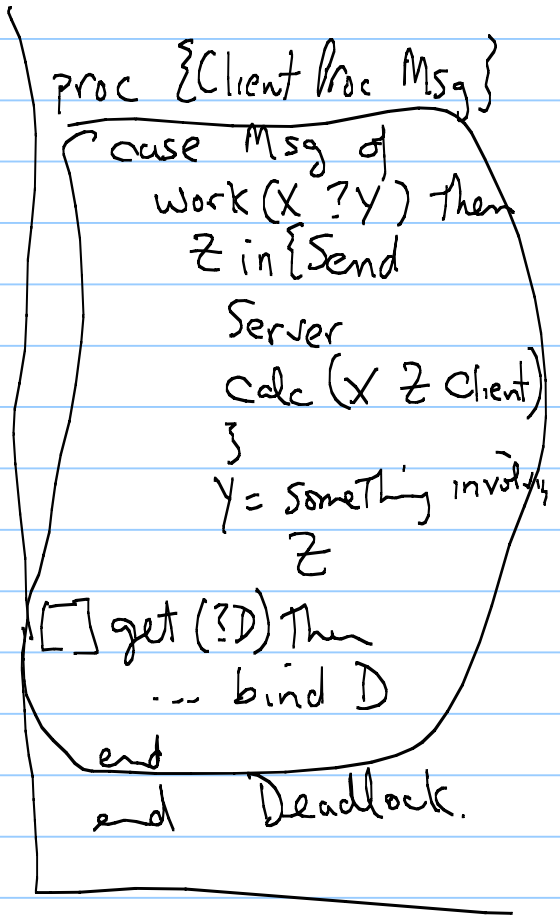
RPC w/ call backs

```
{Server Proc Msg}
   case Msg of
      calc (X ?Y Client) Then D in
         {Send Client get(D)}
         bind Y to something involving X and D
      end
   end
```

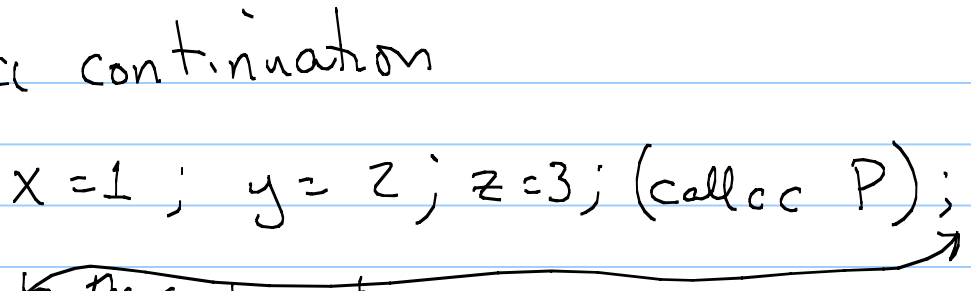
Use another thread

```
case Msg of
   work(X ?Y) Then Z in
      {Send Server calc(X Z Client)}
      thread Y = ... something involving Z end
[] get(?D) ... bind D ...
end
```

— also arrange a new thread to handle the callback.



Use cc continuation



proc {P K} ... {K} ...

5.3.4

5.3.5

Server

```
proc {ServerProc Msg}
   case Msg of
   calc(X ?Y Client Cont) then
      D in
      {Send Client get(D)}
         ... bind Y ...
      {Send Client cont(Cont)}
   end
```

Client

```
proc {ClientProc Msg}
   case Msg of
      work(X ?Y) then Z
         C= proc {$} ... bind Y end
      in
         {Send Server calc(X Z Client C)}
   [] get(D) then bind D ...
   [] cont(C) then {C}
      end
   end
```

5.3.6 — 5.3.8 Error handling, more complex callbacks.
when is deadlock possible, how do you use
threads or continuations to avoid it