University of Kent

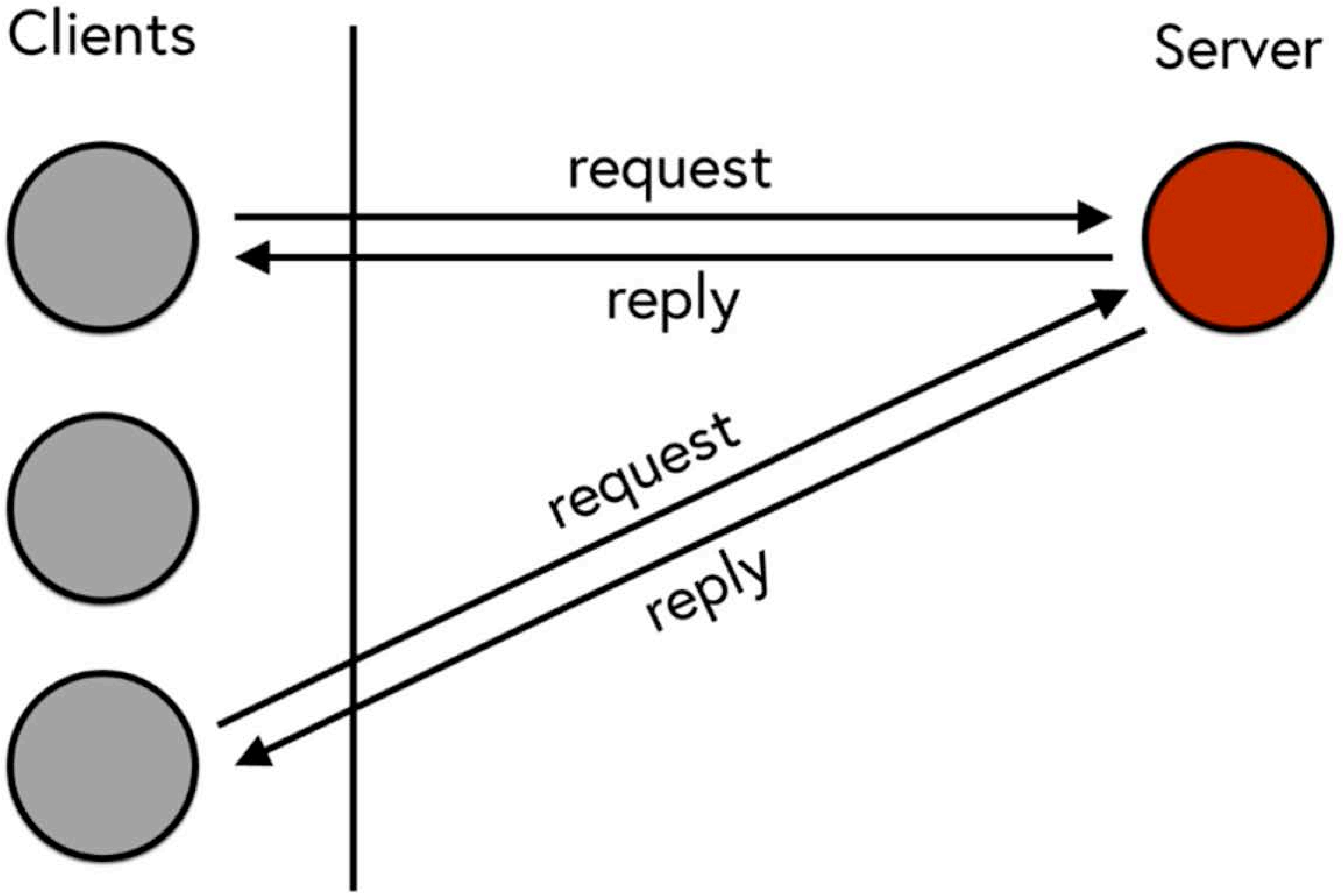
Francesco Cesarini

FOUNDER AND TECHNICAL DIRECTOR, ERLANG SOLUTIONS

University of Kent

What you will learn

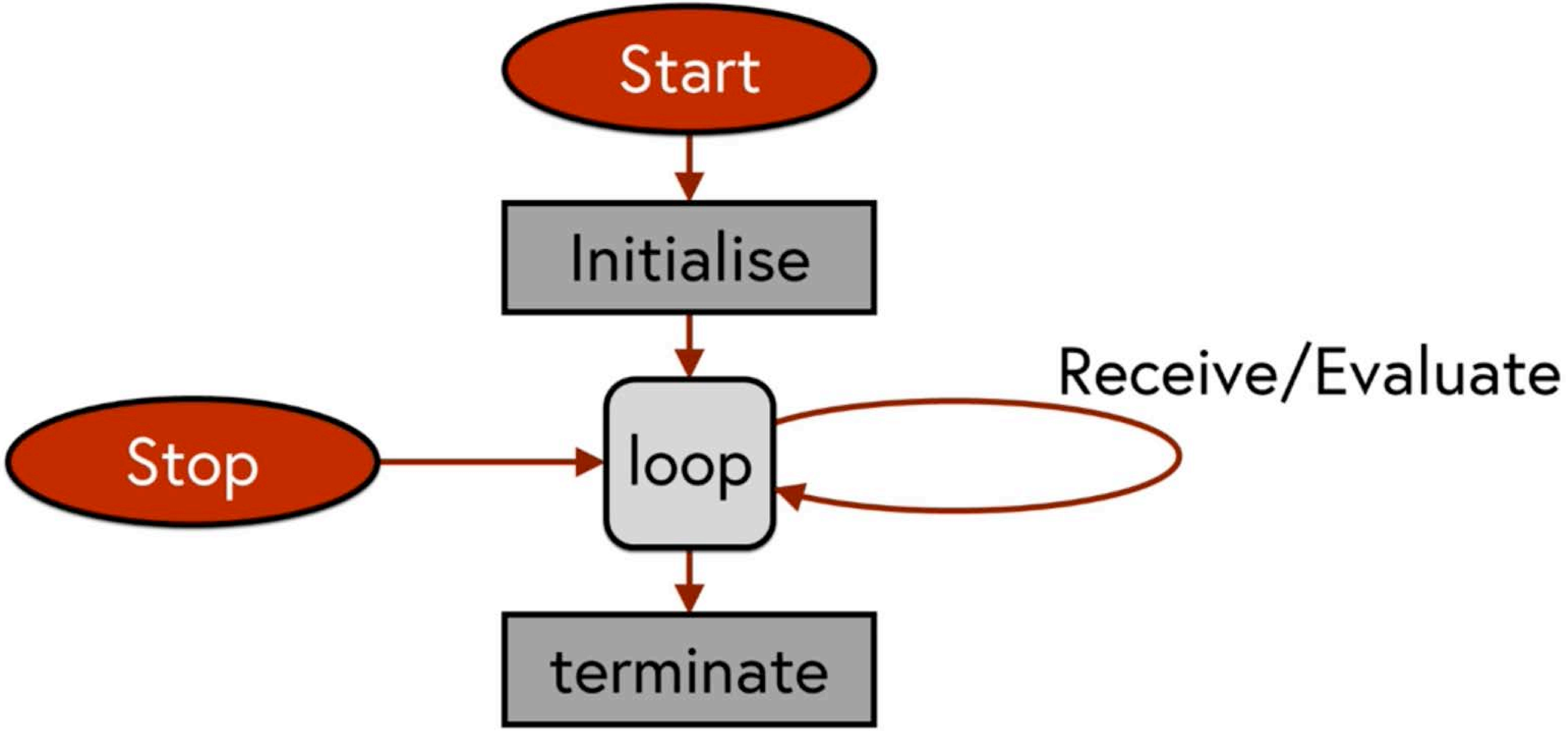
- The road to generics using client-servers
- Writing your own generic server module
- Fault tolerance and supervision trees
- Encapsulating supervision trees in an application and building a release

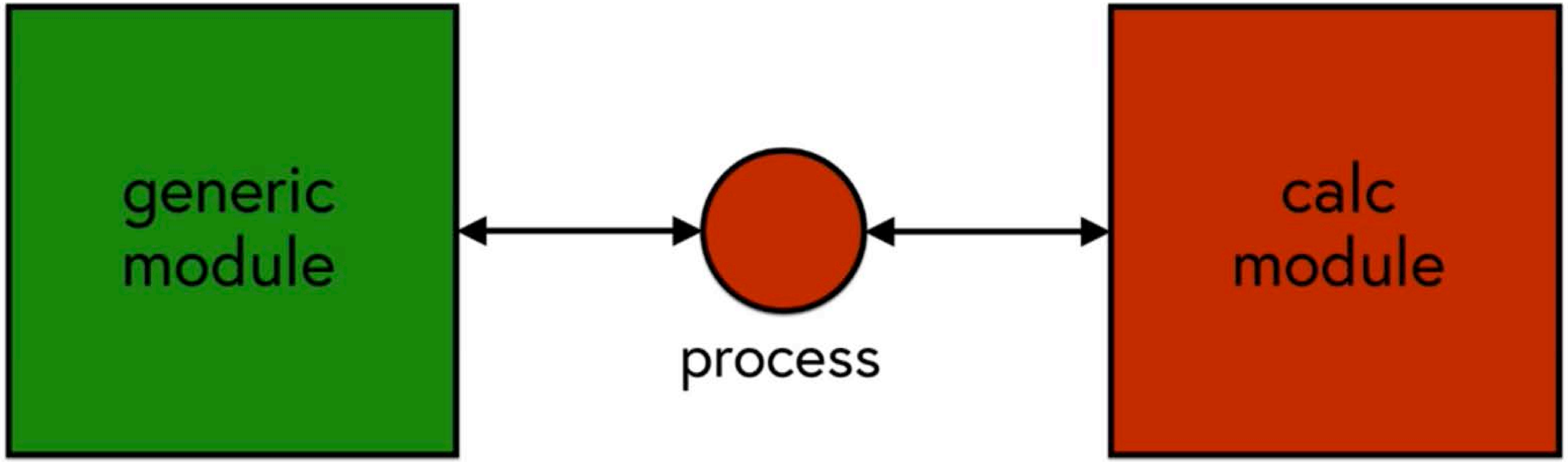

Clients
Server
request
reply
request
reply

```
-module(calc).
-export([start/1, stop/0, eval/1]).
-export([init/1]).

start(Env) ->
   register(calc, spawn(calc, init, [Env])).

stop() ->
   calc ! stop.

eval(Expr) ->
   calc ! {request, self(), {eval, Expr}},
   receive
             {reply, Reply} ->
                Reply
   end.
```

```
init(Env) ->
   io:format(“Starting...~n”),
   loop(Env).

loop(Env) ->
   receive
      {request, From, {eval, Expr}} ->
         From ! {reply, expr:eval(Env, Expr)},
         loop(Env);
      stop ->
         io:format(“Terminating...~n”)
   end.
```

Process Skeletons





- The idea is to split the code in two parts
- The generic part is called the generic behaviour
- The specific part is called the callback module

Generic

- Spawning the server
- Storing the loop data
- Sending requests to the server
- Sending replies to the client
- Receiving server replies
- Stopping the server

Specific

- Initialising the server state
- The loop data
- The client requests
- Handling client requests
- Contents of server reply
- Cleaning up

```erlang
-module(calc).
-export([start/1, stop/0, eval/1]).
-export([init/1]).

start(Env) ->
   register(calc, spawn(calc, init, [Env])).

stop() ->
   calc ! stop.

eval(Expr) ->
   calc ! {request, self(), {eval, Expr}},
   receive
              {reply, Reply} ->
                 Reply
   end.
```

```erlang
init(Env) ->
   io:format("Starting...~n"),
   loop(Env).

loop(Env) ->
   receive
      {request, From, {eval, Expr}} ->
         From ! {reply, expr:eval(Env, Expr)},
         loop(Env);
      stop ->
         io:format("Terminating...~n")
   end.
```

```
-module(calc).                          %% calc
-export([start/1, stop/0, eval/1]).
-export([init/1, handle/2,terminate/1]).

start(Env) ->
    server:start(?MODULE, Env).

init(Env) ->
    io:format("Starting...~n"),
    Env.
```

```
-module(server).                        %% server
-export([start/2, stop/1, request/2]).
-export([init/2]).

start(Name, Args) ->
    register(Name, spawn(?MODULE, init, [Name, Args])).

init(Name, Args) ->
    LoopData = Name:init(Args),
    loop(Name, LoopData).
```

University of Kent