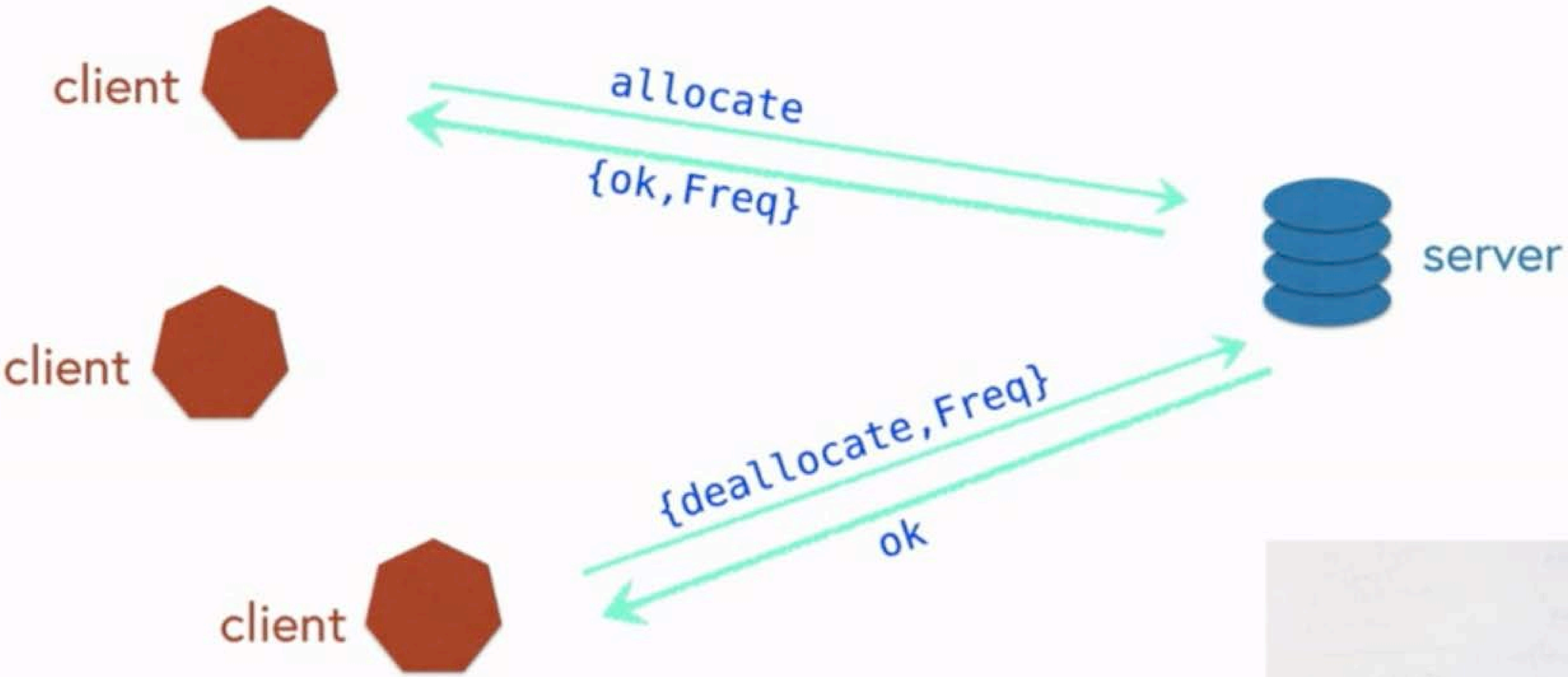
University of Kent

Refactoring the frequency server

A Mobile Frequency Server



Server 1 … explicit send and receive

Messages

request,

process ID of the sender,

service required.

Replies

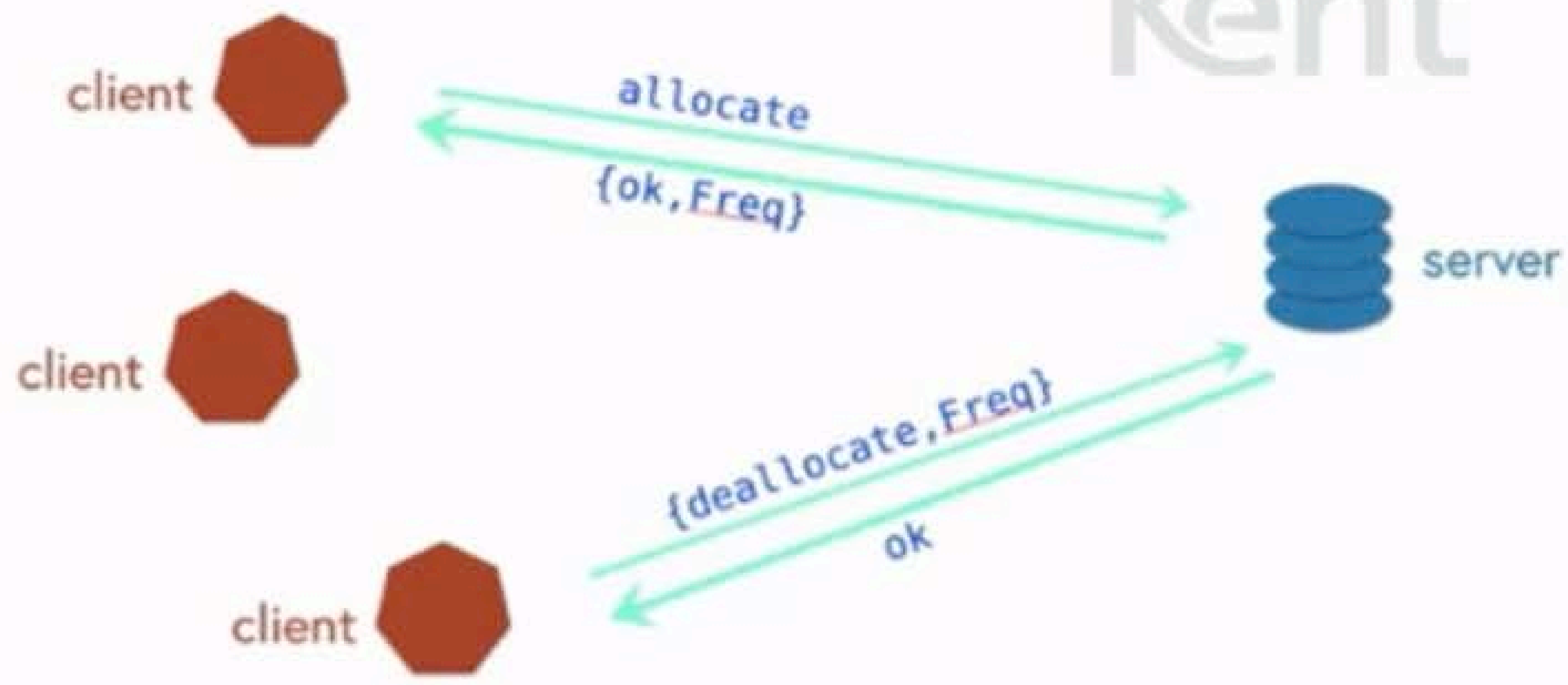
reply,

result (if any).

Loop with updated frequency data.

```
loop(Frequencies) ->
  receive
   {request, Pid, allocate} ->
     {NewFrequencies, Reply} =
           allocate(Frequencies, Pid),
      Pid ! {reply, Reply},
      loop(NewFrequencies);

    {request, Pid , {deallocate, Freq}} ->
      NewFrequencies =
           deallocate(Frequencies, Freq),
      Pid ! {reply, ok},
      loop(NewFrequencies);

    {request, Pid, stop} ->
      Pid ! {reply, stopped}
  end.
```

Client 1: explicit communications



The problem?

The communication protocol is explicit to the client …

… and that prevents re-engineering or upgrading the code in the server.

The solution?

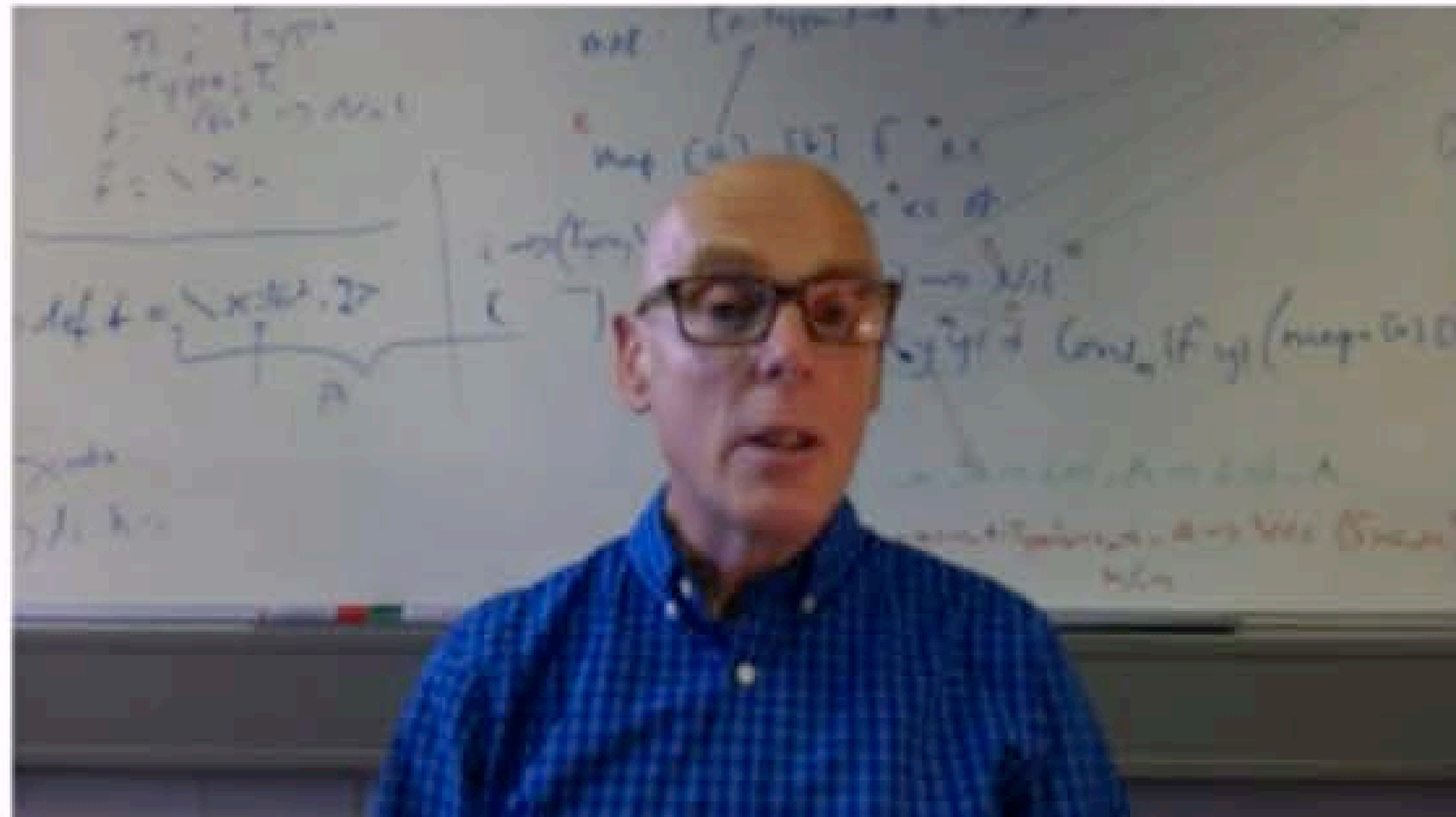
Put the communication behind a functional interface for the client.

```
1> c(frequency).
{ok,frequency}
2> Freq = spawn(frequency, init, []).
<0.44.0>
3> Freq ! {request, self(), allocate}.
{request,<0.40.0>,allocate}
4> receive {reply,Reply} -> Reply end.
{ok,10}
5> …
```

Client version 2: name the server process

Name the server process …

```
start() ->
    register(frequency,
             spawn(frequency, init, [])).
```



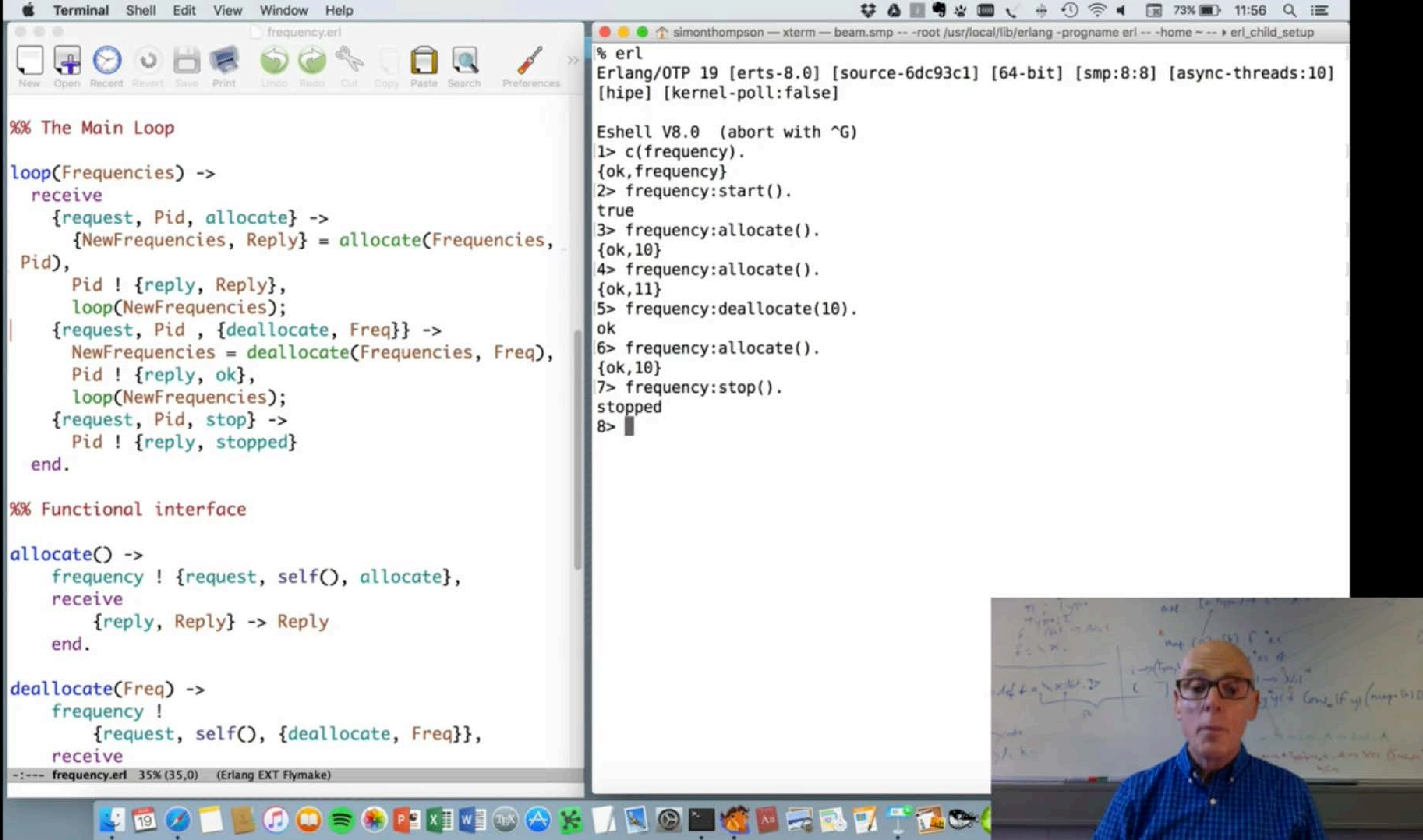
Client version 2: a functional interface

A functional API for the operations hides the process information and message protocol

Each function sends a message and handles the reply.

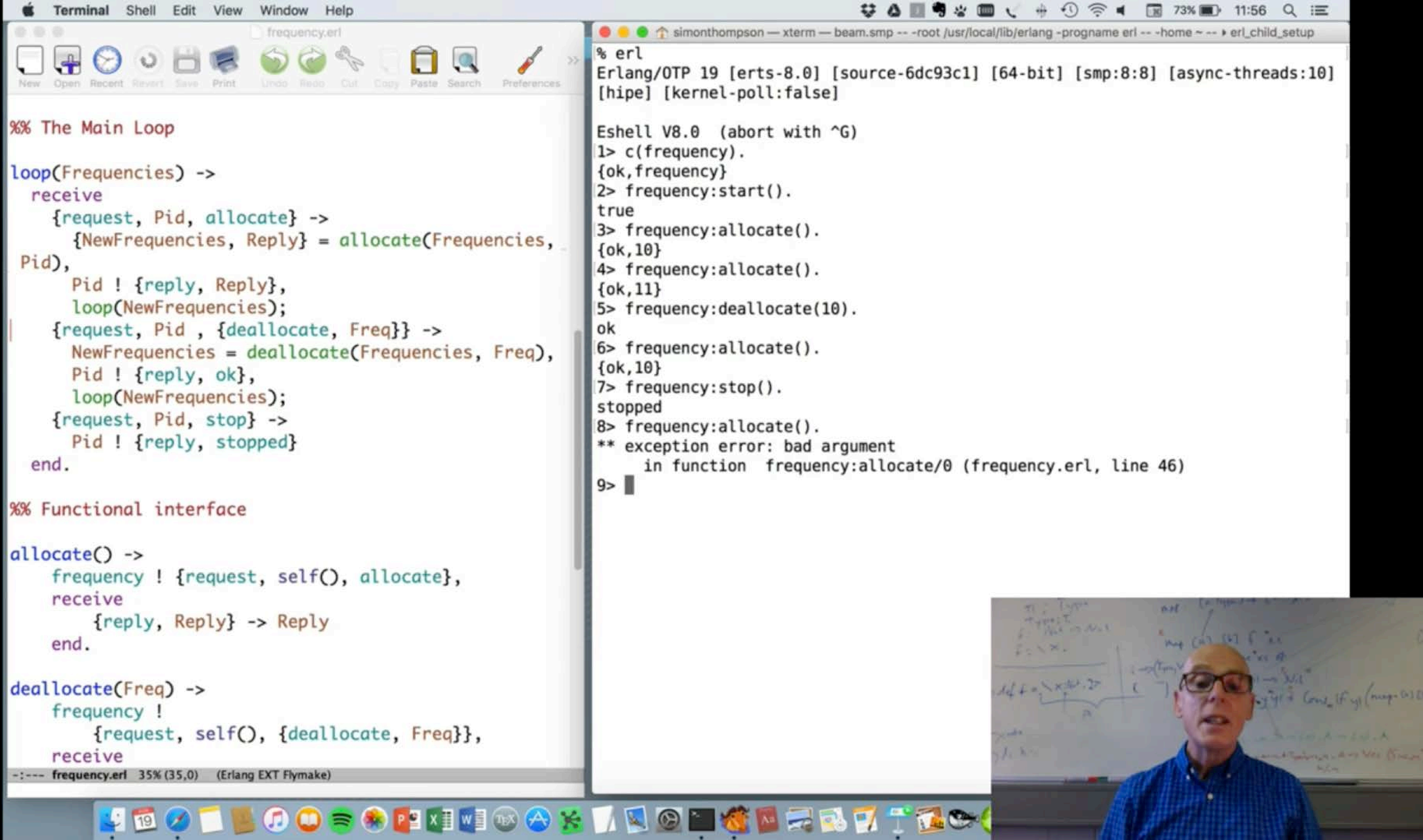
Higher-level API but concurrent aspects are still hand-coded.

```
allocate() ->
  frequency ! {request, self(), allocate},
  receive
    {reply, Reply} -> Reply
  end.

deallocate(Freq) ->
  frequency !
     {request, self(), {deallocate, Freq}},
  receive
    {reply, Reply} -> Reply
  end.

1> frequency:start().
true
2> frequency:allocate().
{ok,10}
```


Terminal
Shell
Edit
View
Window
Help
73%
11:56
frequency.erl
New
Open
Recent
Revert
Save
Print
Undo
Redo
Cut
Copy
Paste
Search
Preferences
simonthompson — xterm — beam.smp -- -root /usr/local/lib/erlang -progname erl -- -home ~ -- ▸ erl_child_setup
-:--- frequency.erl 35% (35,0) (Erlang EXT Flymake)

```
%% The Main Loop

loop(Frequencies) ->
  receive
    {request, Pid, allocate} ->
      {NewFrequencies, Reply} = allocate(Frequencies,
 Pid),
      Pid ! {reply, Reply},
      loop(NewFrequencies);
    {request, Pid , {deallocate, Freq}} ->
      NewFrequencies = deallocate(Frequencies, Freq),
      Pid ! {reply, ok},
      loop(NewFrequencies);
    {request, Pid, stop} ->
      Pid ! {reply, stopped}
  end.

%% Functional interface

allocate() ->
    frequency ! {request, self(), allocate},
    receive
        {reply, Reply} -> Reply
    end.

deallocate(Freq) ->
    frequency !
        {request, self(), {deallocate, Freq}},
    receive
```

```
% erl
Erlang/OTP 19 [erts-8.0] [source-6dc93c1] [64-bit] [smp:8:8] [async-threads:10]
[hipe] [kernel-poll:false]

Eshell V8.0  (abort with ^G)
1> c(frequency).
{ok,frequency}
2> frequency:start().
true
3> frequency:allocate().
{ok,10}
4> frequency:allocate().
{ok,11}
5> frequency:deallocate(10).
ok
6> frequency:allocate().
{ok,10}
7> frequency:stop().
stopped
8> 
```


Terminal
Shell
Edit
View
Window
Help
73%
11:56
frequency.erl
New
Open
Recent
Revert
Save
Print
Undo
Redo
Cut
Copy
Paste
Search
Preferences
simonthompson — xterm — beam.smp -- -root /usr/local/lib/erlang -progname erl -- -home ~ -- ▸ erl_child_setup
-:--- frequency.erl 35% (35,0) (Erlang EXT Flymake)

```
%% The Main Loop

loop(Frequencies) ->
  receive
    {request, Pid, allocate} ->
      {NewFrequencies, Reply} = allocate(Frequencies,
 Pid),
      Pid ! {reply, Reply},
      loop(NewFrequencies);
    {request, Pid , {deallocate, Freq}} ->
      NewFrequencies = deallocate(Frequencies, Freq),
      Pid ! {reply, ok},
      loop(NewFrequencies);
    {request, Pid, stop} ->
      Pid ! {reply, stopped}
  end.

%% Functional interface

allocate() ->
    frequency ! {request, self(), allocate},
    receive
        {reply, Reply} -> Reply
    end.

deallocate(Freq) ->
    frequency !
        {request, self(), {deallocate, Freq}},
    receive
```

```
% erl
Erlang/OTP 19 [erts-8.0] [source-6dc93c1] [64-bit] [smp:8:8] [async-threads:10]
[hipe] [kernel-poll:false]

Eshell V8.0  (abort with ^G)
1> c(frequency).
{ok,frequency}
2> frequency:start().
true
3> frequency:allocate().
{ok,10}
4> frequency:allocate().
{ok,11}
5> frequency:deallocate(10).
ok
6> frequency:allocate().
{ok,10}
7> frequency:stop().
stopped
8> frequency:allocate().
** exception error: bad argument
     in function  frequency:allocate/0 (frequency.erl, line 46)
9> 
```

Client version 2: a functional interface

A functional API for the operations hides the process information and message protocol

Each function sends a message and handles the reply.

Higher-level API but concurrent aspects are still hand-coded.

```
allocate() ->
  frequency ! {request, self(), allocate},
  receive
    {reply, Reply} -> Reply
  end.

deallocate(Freq) ->
  frequency !
     {request, self(), {deallocate, Freq}},
  receive
    {reply, Reply} -> Reply
  end.

1> frequency:start().
true
2> frequency:allocate().
{ok,10}
```

University of
Kent