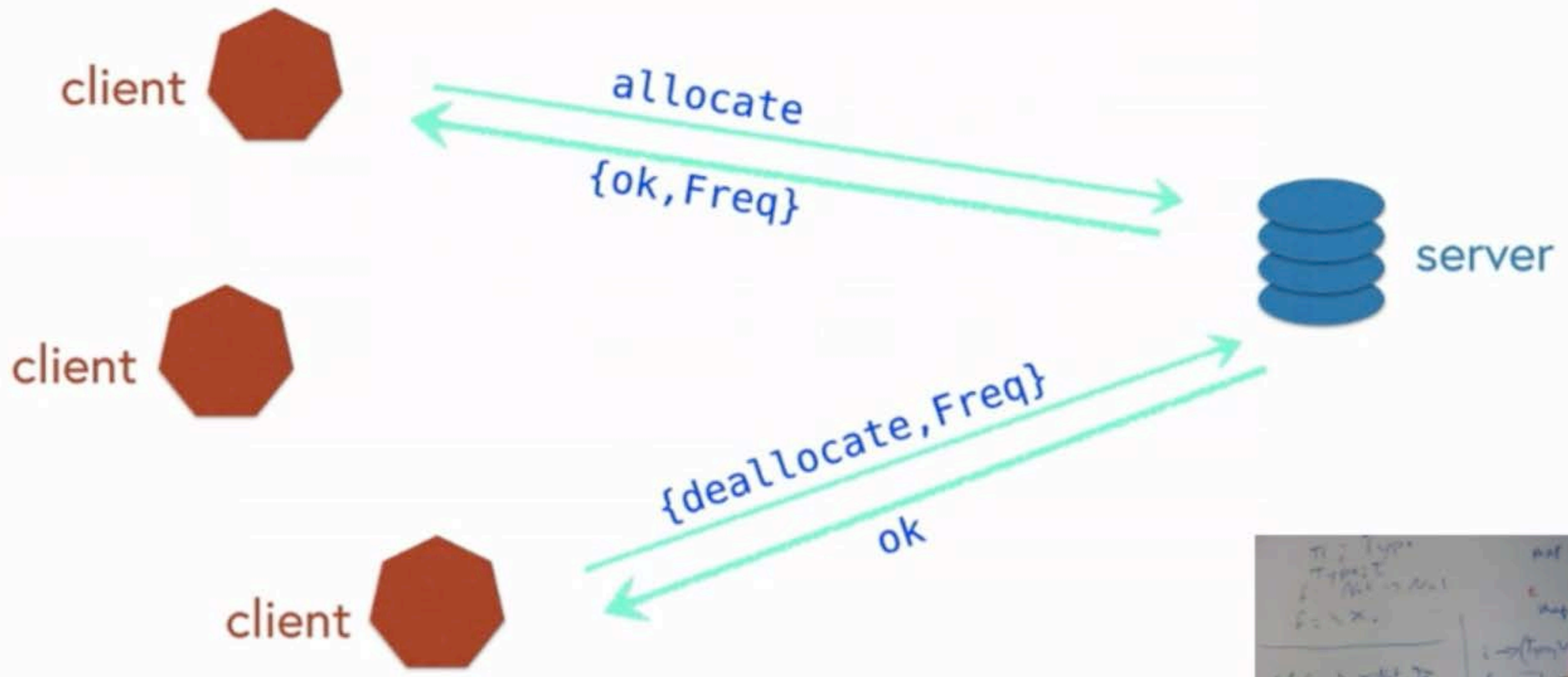
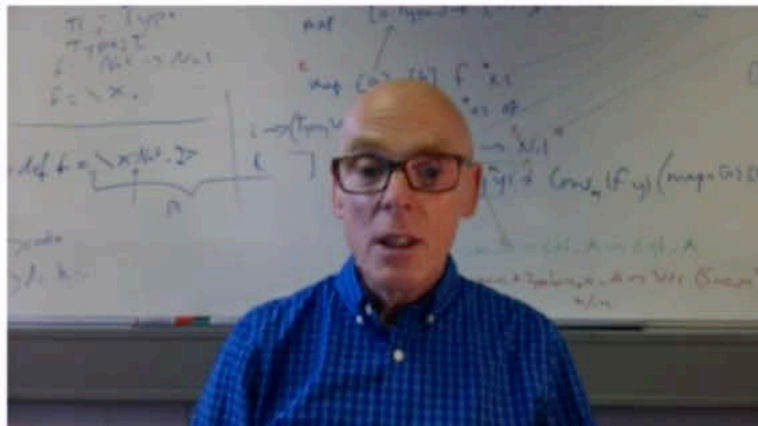

University of
Kent

University of Kent

Introducing the frequency server

A Mobile Frequency Server

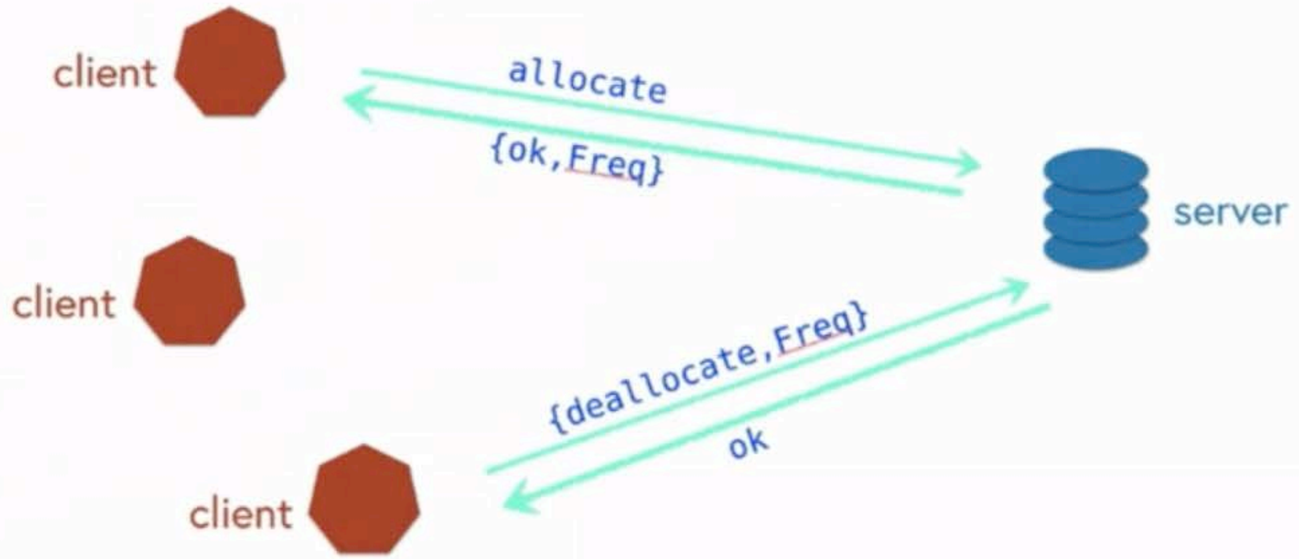




A Mobile Frequency Server

A client can request that a frequency be allocated.

A client can deallocate a frequency.



Design questions

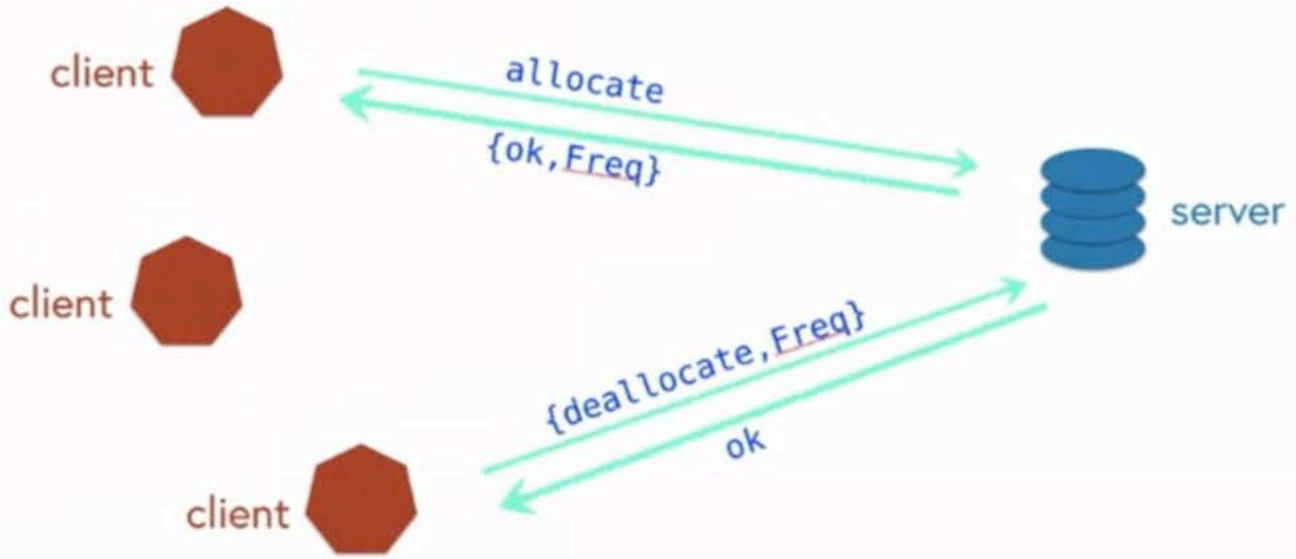
Communication mechanism

Communication protocol

Fault tolerance

Library support: OTP

Scaling



In the server … functional state

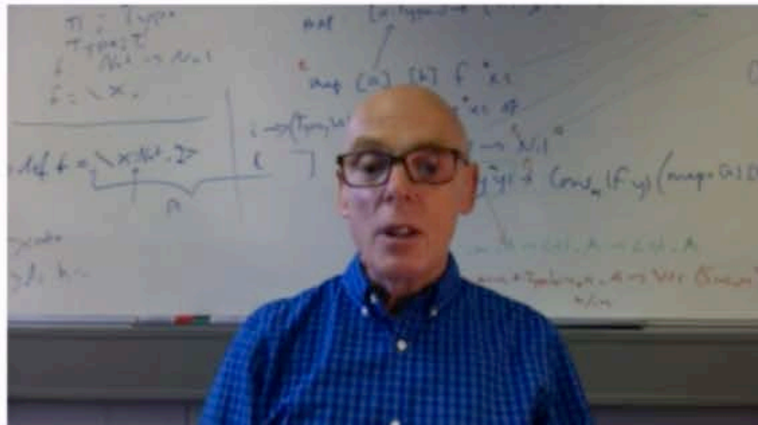
We build a functional model of the state of the server

… a pair of lists `{Free,Allocated}` …

… a function to `allocate` a frequency, if possible, and

… a function to `deallocate` a frequency

… both functions return a new state (and a result).



Frequency Allocation and Deallocation

University of Kent

A pair of lists models free and allocated frequencies.

Return the new pair plus a message with the result.

Deallocation succeeds, assuming the `Freq` was already allocated.

```
allocate({[], Allocated}, _Pid) ->
  {{[], Allocated},
     {error, no_frequency}};

allocate({[Freq|Free], Allocated}, Pid) ->
  {{Free, [{Freq, Pid}|Allocated]},
     {ok, Freq}}.


deallocate({Free, Allocated}, Freq) ->
  NewAllocated
     =lists:keydelete(Freq, 1, Allocated),
  {[Freq|Free],  NewAllocated}.
```

Frequency Allocation and Deallocation

University of Kent

A pair of lists models free and allocated frequencies.

Return the new pair plus a message with the result.

Deallocation succeeds, assuming the `Freq` was already allocated.

```
allocate({[], Allocated}, _Pid) ->
  {{[], Allocated},
     {error, no_frequency}};

allocate({[Freq|Free], Allocated}, Pid) ->
  {{Free, [{Freq, Pid}|Allocated]},
     {ok, Freq}}.



deallocate({Free, Allocated}, Freq) ->
  NewAllocated
     =lists:keydelete(Freq, 1, Allocated),
  {[Freq|Free],  NewAllocated}.
```

Frequency Allocation and Deallocation

A pair of lists models free and allocated frequencies.

Return the new pair plus a message with the result.

Deallocation succeeds, assuming the `Freq` was already allocated.

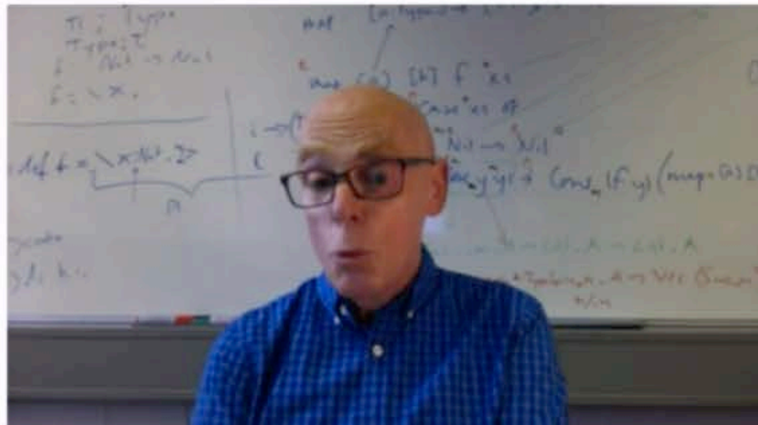
```
allocate({[], Allocated}, _Pid) ->
  {{[], Allocated},
     {error, no_frequency}};

allocate({[Freq|Free], Allocated}, Pid) ->
  {{Free, [{Freq, Pid}|Allocated]},
     {ok, Freq}}.


deallocate({Free, Allocated}, Freq) ->
  NewAllocated
     =lists:keydelete(Freq, 1, Allocated),
  {[Freq|Free],  NewAllocated}.
```

Server version 1 ... explicit send and receive

University of Kent

In our first version we'll use "raw" Erlang communication between client and sever.

Client will need to include their `Pid` in each request so that the server can reply.



Server version 1 … explicit send and receive

University of Kent

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.
```

Server version 1 … explicit send and receive

University of Kent

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.
```

Server version 1 … explicit send and receive

University of Kent

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.
```

Running the server

University of Kent

spawn a process to run init.

init will set up the loop data and call the loop for the first time.

```
spawn(frequency, init, [])).


init() ->
  Frequencies = {get_frequencies(), []},
  loop(Frequencies).

% Hard Coded

get_frequencies() -> [10,11,12,13,14,15].

loop(Frequencies) ->
  receive
   {request, Pid, allocate} -> etc.
```

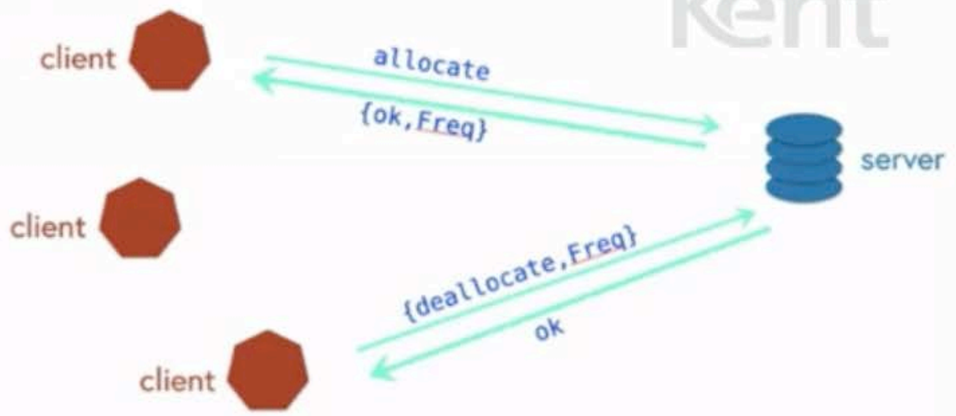
Running the server

spawn a process to run init.

init will set up the loop data and call the loop for the first time.

```
spawn(frequency, init, [])).


init() ->
  Frequencies = {get_frequencies(), []},
  loop(Frequencies).

% Hard Coded

get_frequencies() -> [10,11,12,13,14,15].

loop(Frequencies) ->
  receive
   {request, Pid, allocate} -> etc.
```

Client version 1: everything explicit



Messages

request,
process ID of the sender and
service required.

Replies

reply,
result (if any).

```
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> …
```

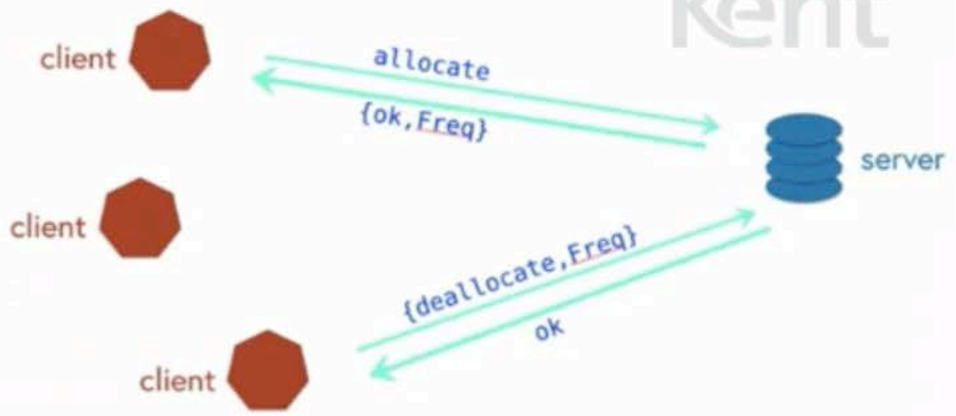
```
init() ->
  Frequencies = {get_frequencies(), []},
  loop(Frequencies).

% Hard Coded
get_frequencies() -> [10,11,12,13,14,15].

%% 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.

%% The Internal Help Functions used to allocate and
%% deallocate frequencies.

allocate({[], Allocated}, _Pid) ->
  {{[], Allocated}, {error, no_frequency}};
```

```
% 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> Freq = spawn(frequency,init,[]).
<0.64.0>
3> Freq ! {request,self(),allocate}.
{request,<0.57.0>,allocate}
4> receive {reply,Msg} -> Msg end.
{ok,10}
5>
```

```
init() ->
  Frequencies = {get_frequencies(), []},
  loop(Frequencies).

% Hard Coded
get_frequencies() -> [10,11,12,13,14,15].

%% 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.

%% The Internal Help Functions used to allocate and
%% deallocate frequencies.

allocate({[], Allocated}, _Pid) ->
  {{[], Allocated}, {error, no_frequency}};
```

```
% 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> Freq = spawn(frequency,init,[]).
<0.64.0>
3> Freq ! {request,self(),allocate}.
{request,<0.57.0>,allocate}
4> receive {reply,Msg} -> Msg end.
{ok,10}
5> Freq ! {request,self(),{deallocate,10}}.
{request,<0.57.0>,{deallocate,10}}
6> f(Msg).
ok
7> receive {reply,Msg} -> Msg end.
ok
8> Freq ! {request,self(),allocate}.
{request,<0.57.0>,allocate}
9> f(Msg).
ok
10> receive {reply,Msg} -> Msg end.
{ok,10}
11>
```

Client version 1: everything explicit



Messages

request,
process ID of the sender and
service required.

Replies

reply,
result (if any).

```
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> …
```


University of
Kent