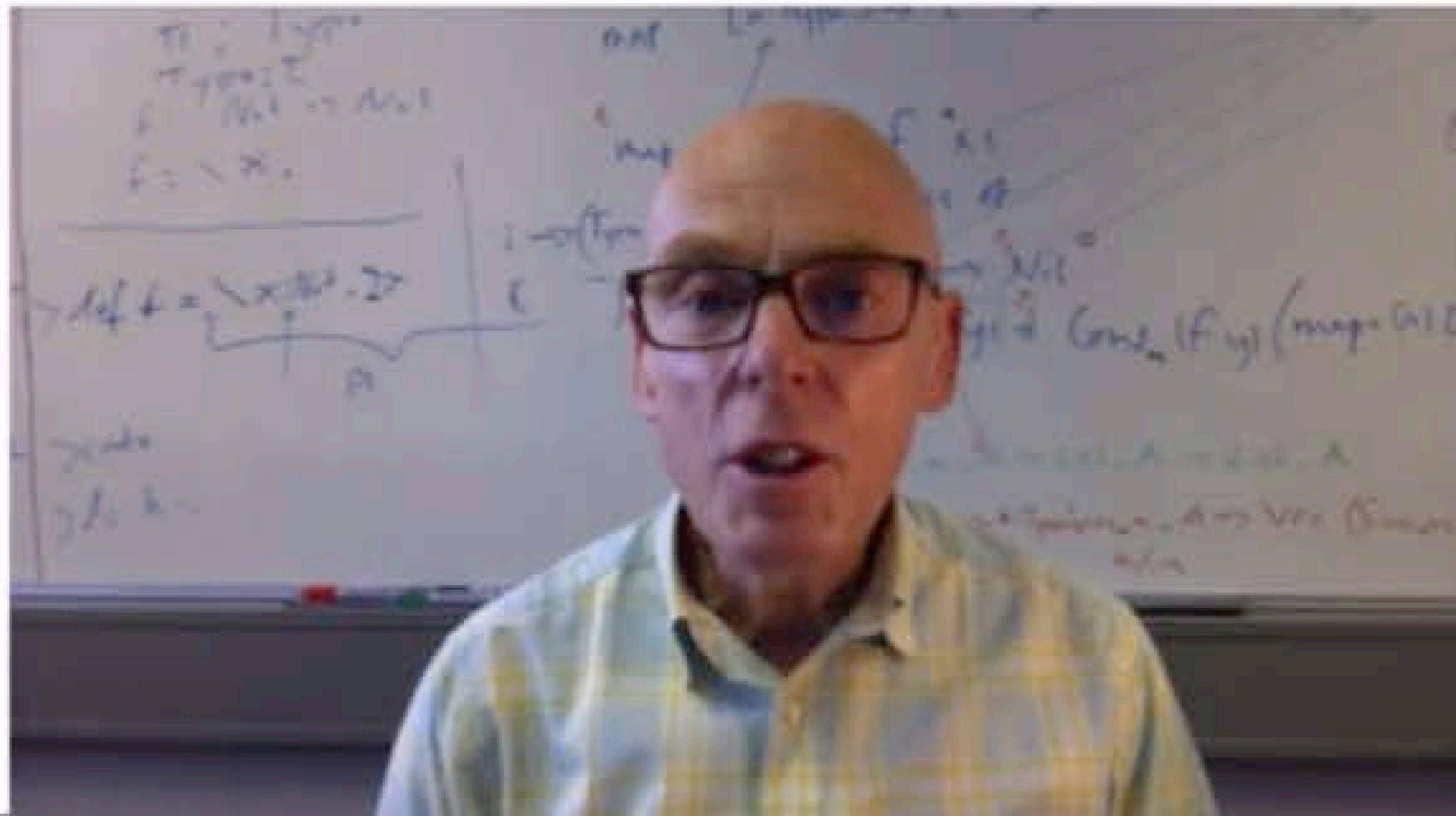
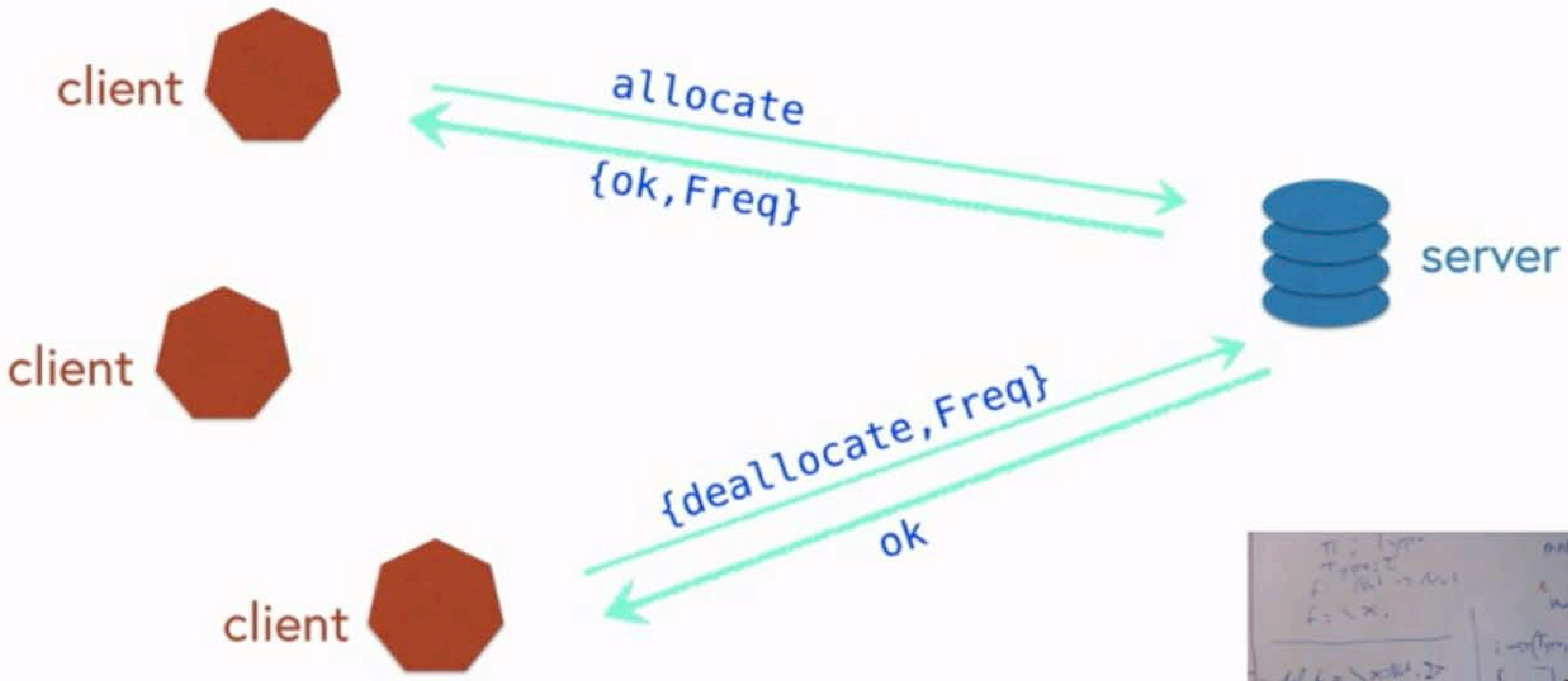
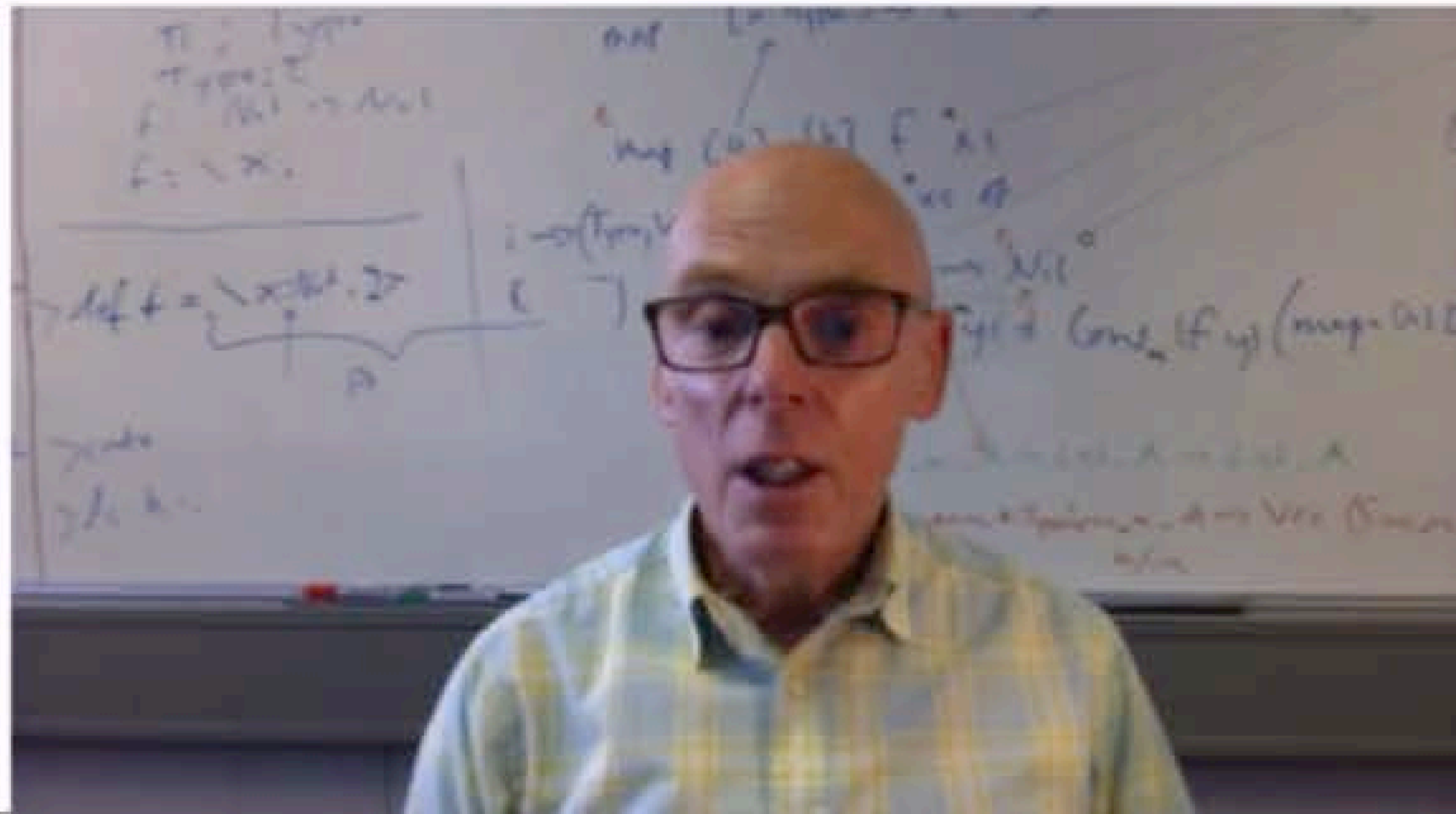
University of Kent

Hardening the frequency server



A Mobile Frequency Server

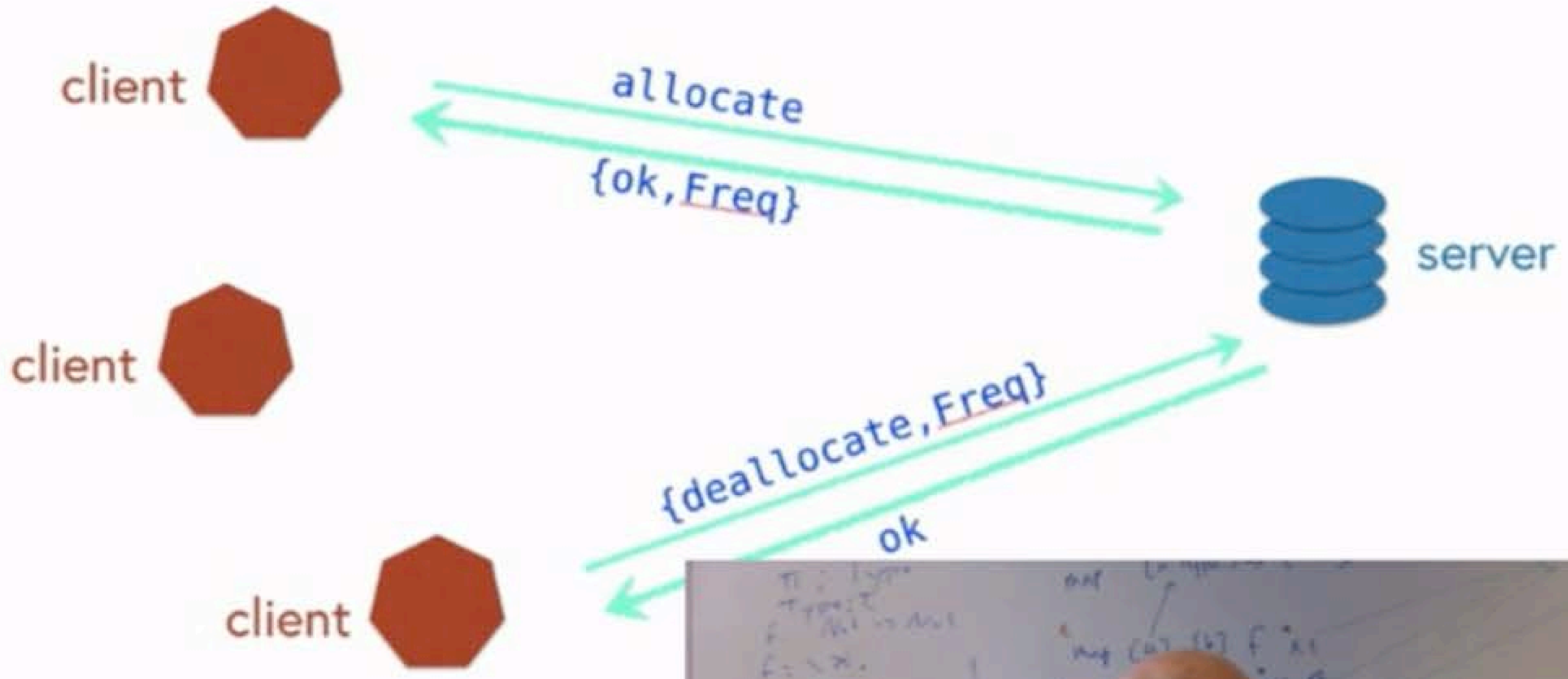


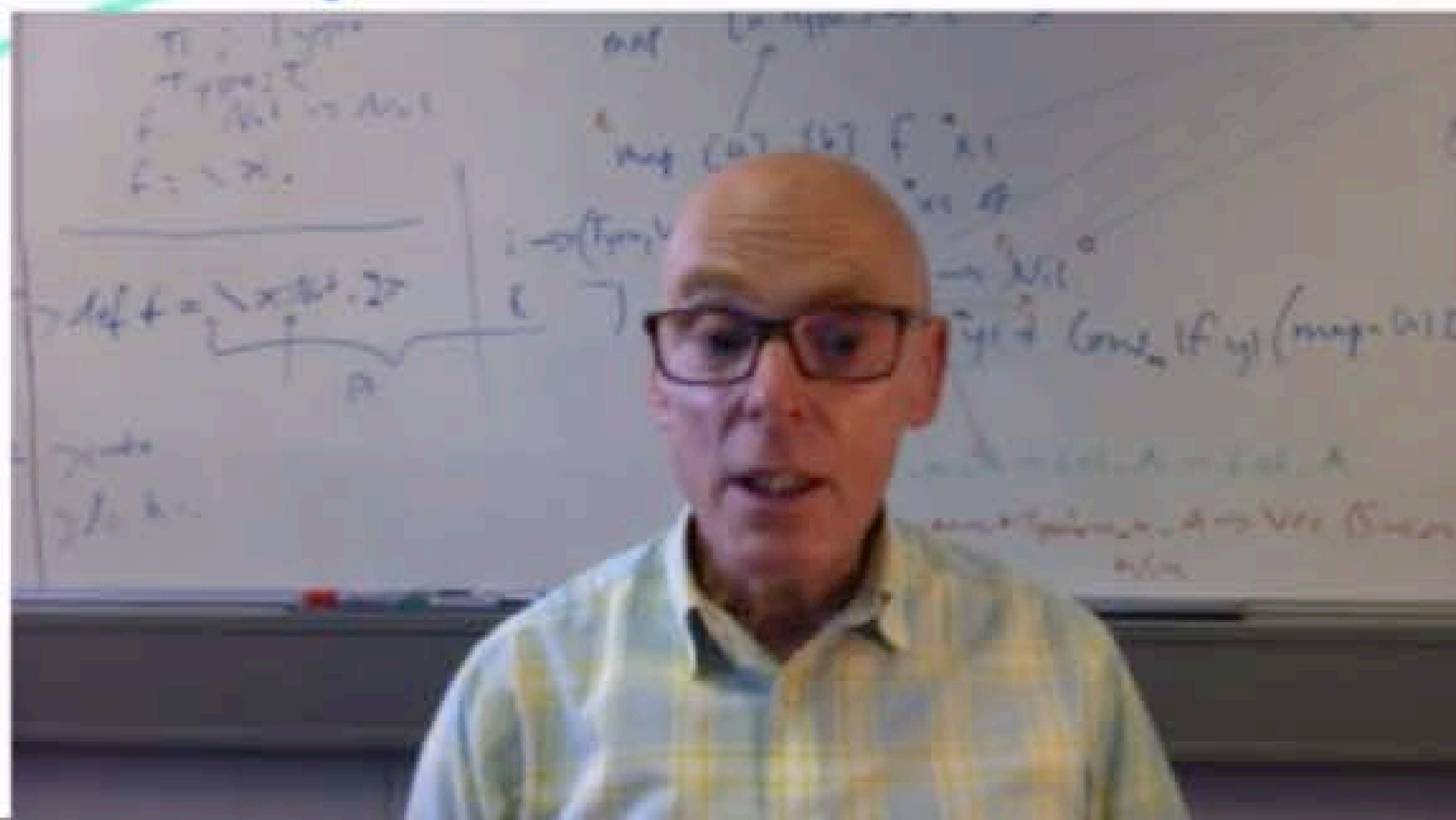


A Mobile Frequency Server

A client can request that a frequency be allocated.

A client can deallocate a frequency.





What happens if a client dies?

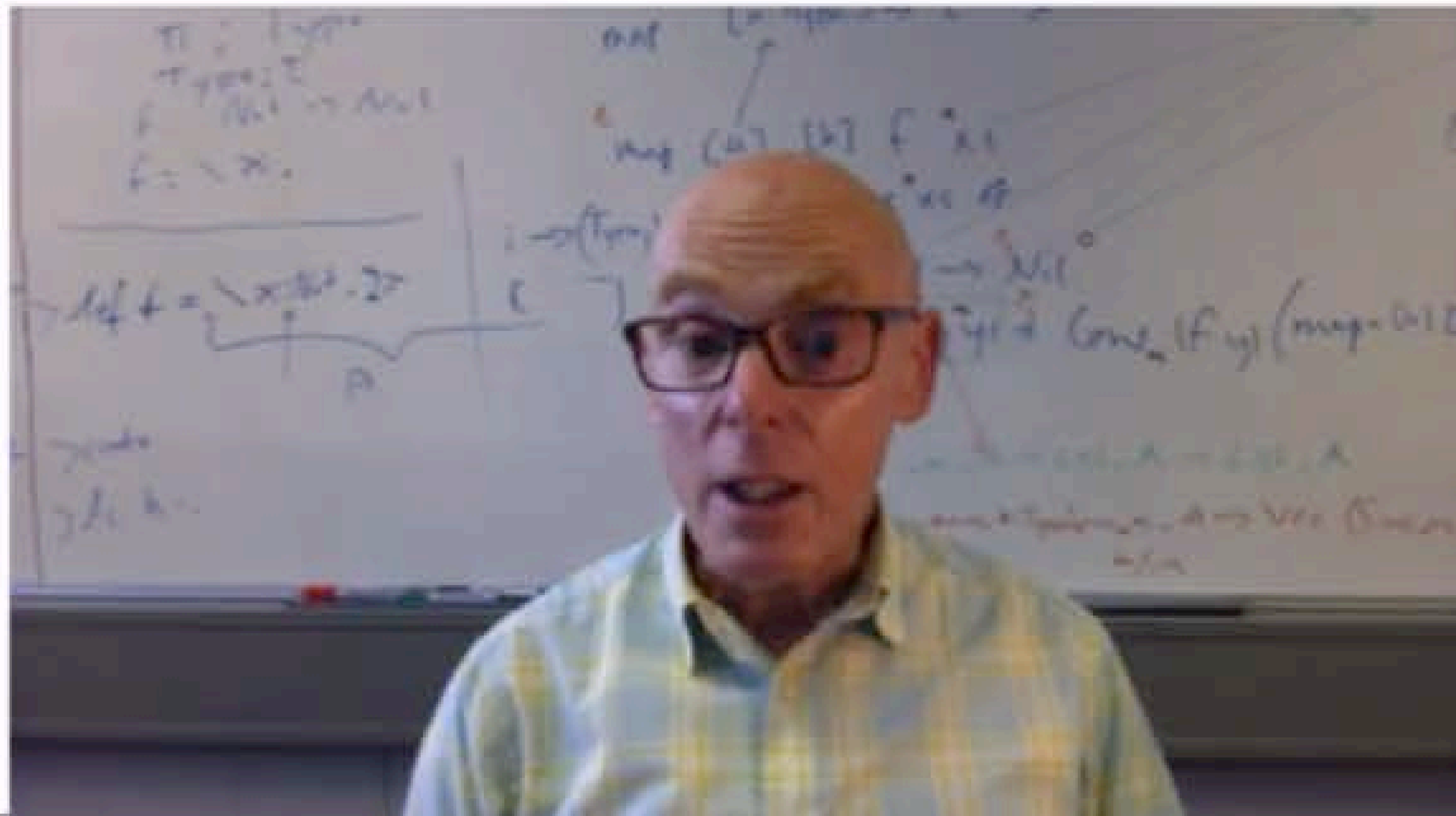
If a client holding a frequency dies …

… then it can't return the frequency, and it is wasted.

Solution?

Link to each client …

… while it has a frequency.



The server must trap exits

```
-module(frequency).
-export([start/0, stop/0, allocate/0, deallocate/1]).
-export([init/0]).

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

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

University of Kent

The server must handle the `{'EXIT', … }` message

```
loop(Frequencies) ->
  receive
    {request, Pid, allocate} ->
      … ;
    {request, Pid , {deallocate, Freq}} ->
      … ;
    {'EXIT', Pid, _Reason} ->
      NewFrequencies = exited(Frequencies, Pid),
      loop(NewFrequencies);
    {request, Pid, stop} ->
      reply(Pid, ok)
  end.
```

Need to define this new aspect of server functionality

Link on allocate / unlink on deallocate

```
allocate({[], Allocated}, _Pid) ->
  {{[], Allocated}, {error, no_frequencies}};
allocate({[Freq|Frequencies], Allocated}, Pid) ->
  link(Pid),
  {{Frequencies,[{Freq,Pid}|Allocated]},{ok,Freq}}.

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

Here's the reason that we store the Pid on allocation.

Handling the exit in the server

```
exited({Free, Allocated}, Pid) ->
  case lists:keysearch(Pid,2,Allocated) of
    {value,{Freq,Pid}} ->
       NewAllocated = lists:keydelete(Freq,1,Allocated),
      {[Freq|Free],NewAllocated};
    false ->
      {Free,Allocated}
  end.
```

Handling the exit in the server

```
exited({Free, Allocated}, Pid) ->
  case lists:keysearch(Pid,2,Allocated) of
    {value,{Freq,Pid}} ->
       NewAllocated = lists:keydelete(Freq,1,Allocated),
      {[Freq|Free],NewAllocated};
    false ->
      {Free,Allocated}
  end.
```

Why check that `{Freq,Pid}` is in `Allocated`?

To avoid the *race condition* that `Freq` has been deallocated in the client and `Pid` terminated before `Freq` can be deallocated in the server.

Links are bidirectional

If the server dies while a client has a frequency, then the client is killed too.

Good?

If the server has died then safest to restart the whole system.

Maintains consistency

Bad?

Shouldn't kill a call just because infrastructure goes down.

Should be able to restart server using knowledge of allocated frequencies.

Links are bidirectional

If the server dies while a client has a frequency, then the client is killed too.

How to avoid?

Get the client to trap exits …

… all the time … or when it has a frequency allocated to it.

Exercise!