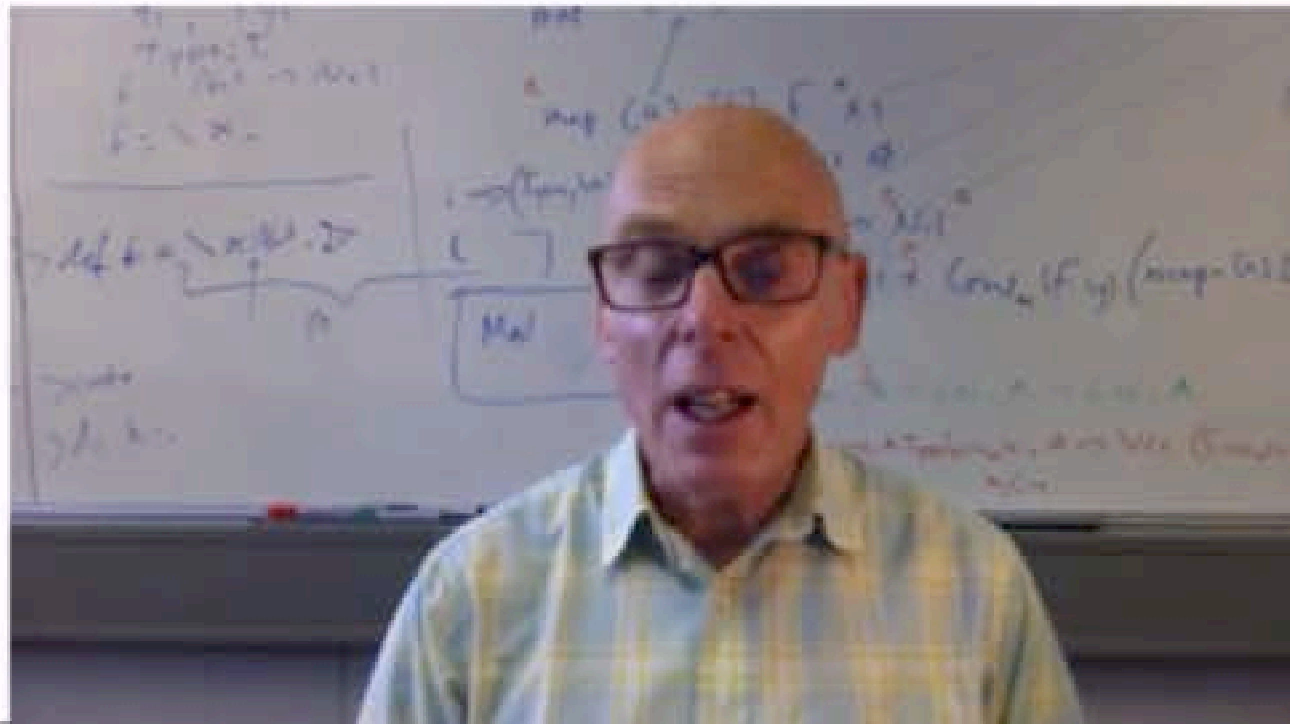
University of Kent

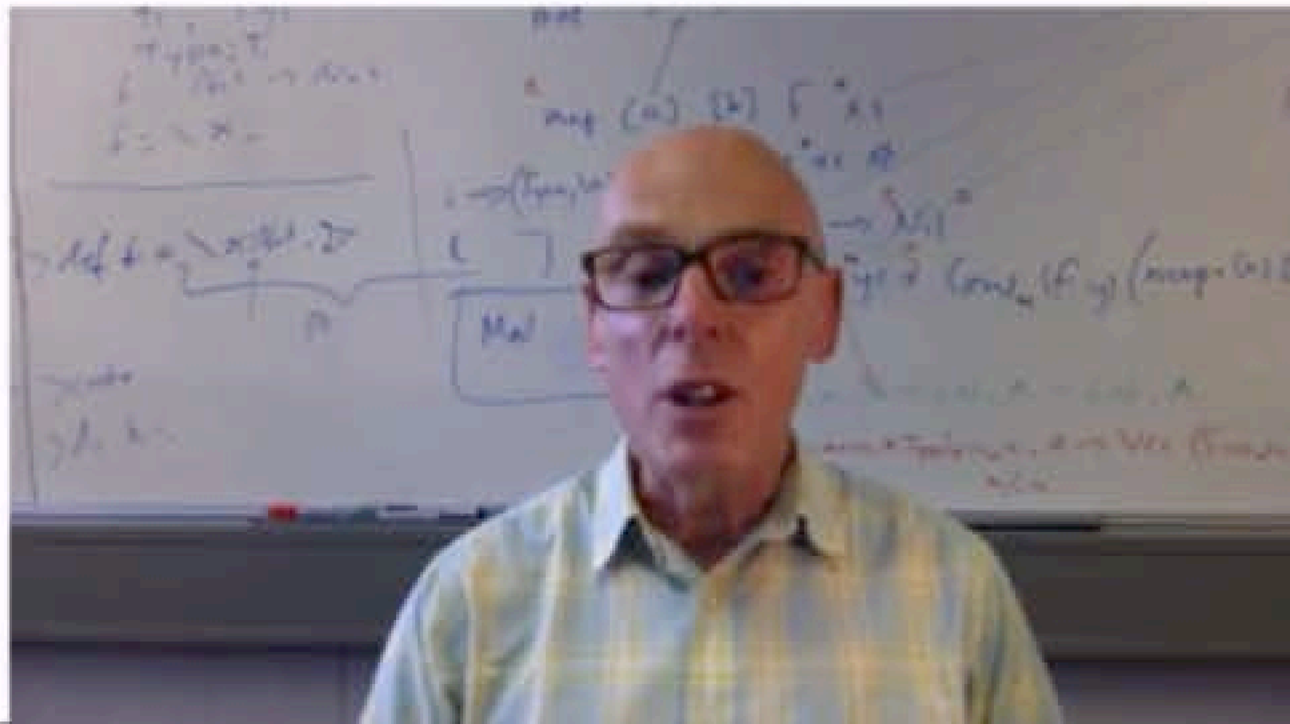
Supervisors



Defensive programming not defensible

Trying to deal with all possible failure modes for a module is doomed to failure …

… just think about Donald Rumsfeld's "unknown unknowns"!

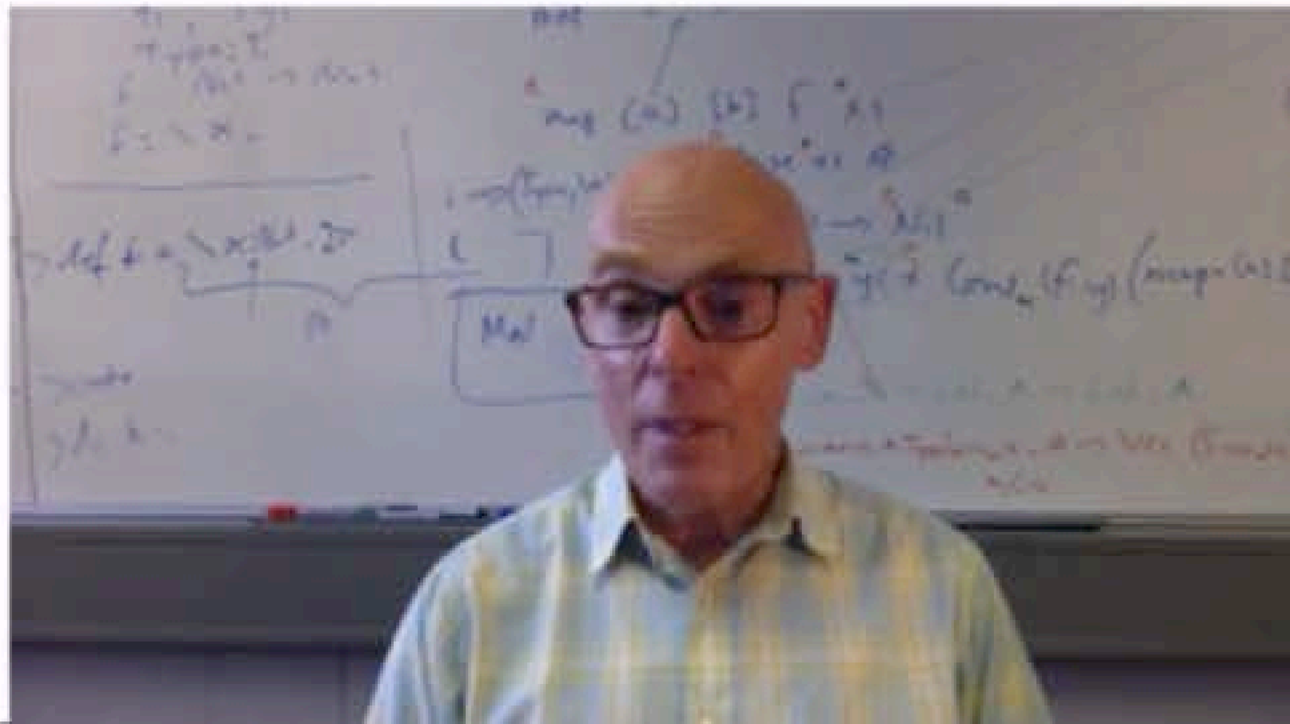


Supervisor and worker

Design **workers** to do a particular job …

… assuming that the rest of the world is behaving appropriately,

… and if not, to fail.



Supervisor and worker

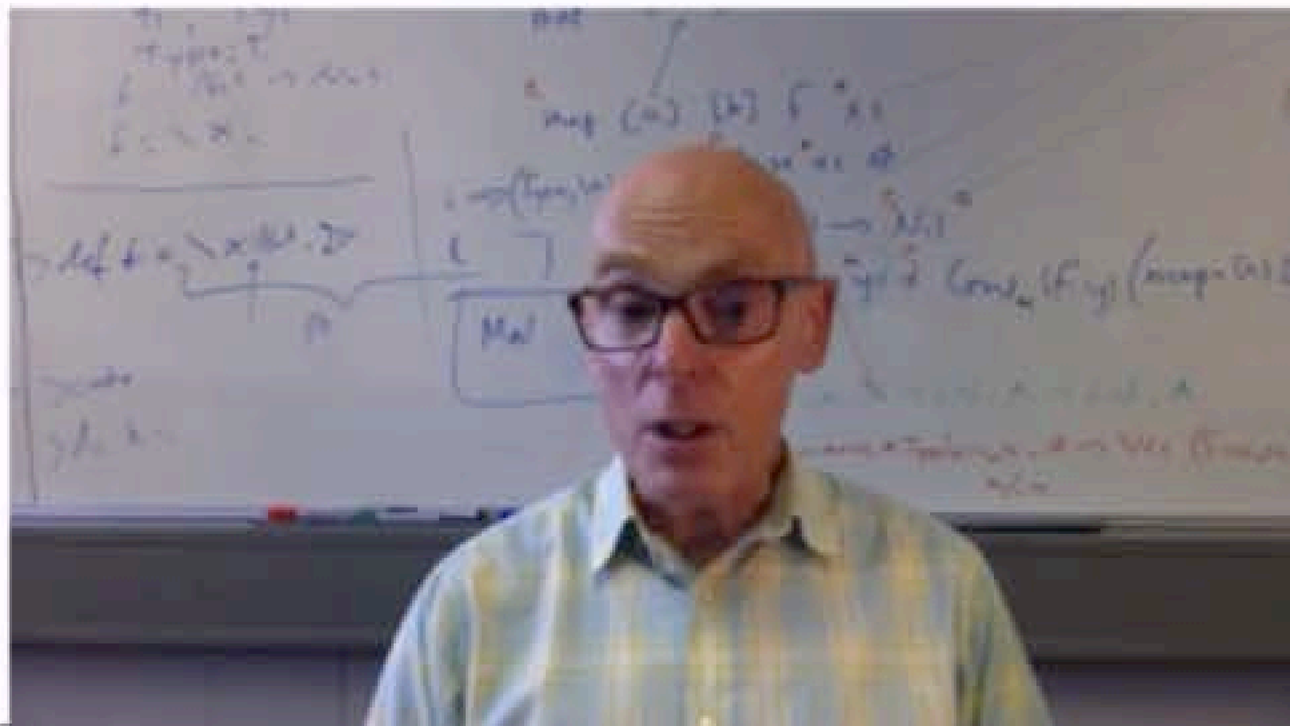
Design **workers** to do a particular job …

… assuming that the rest of the world is behaving appropriately,

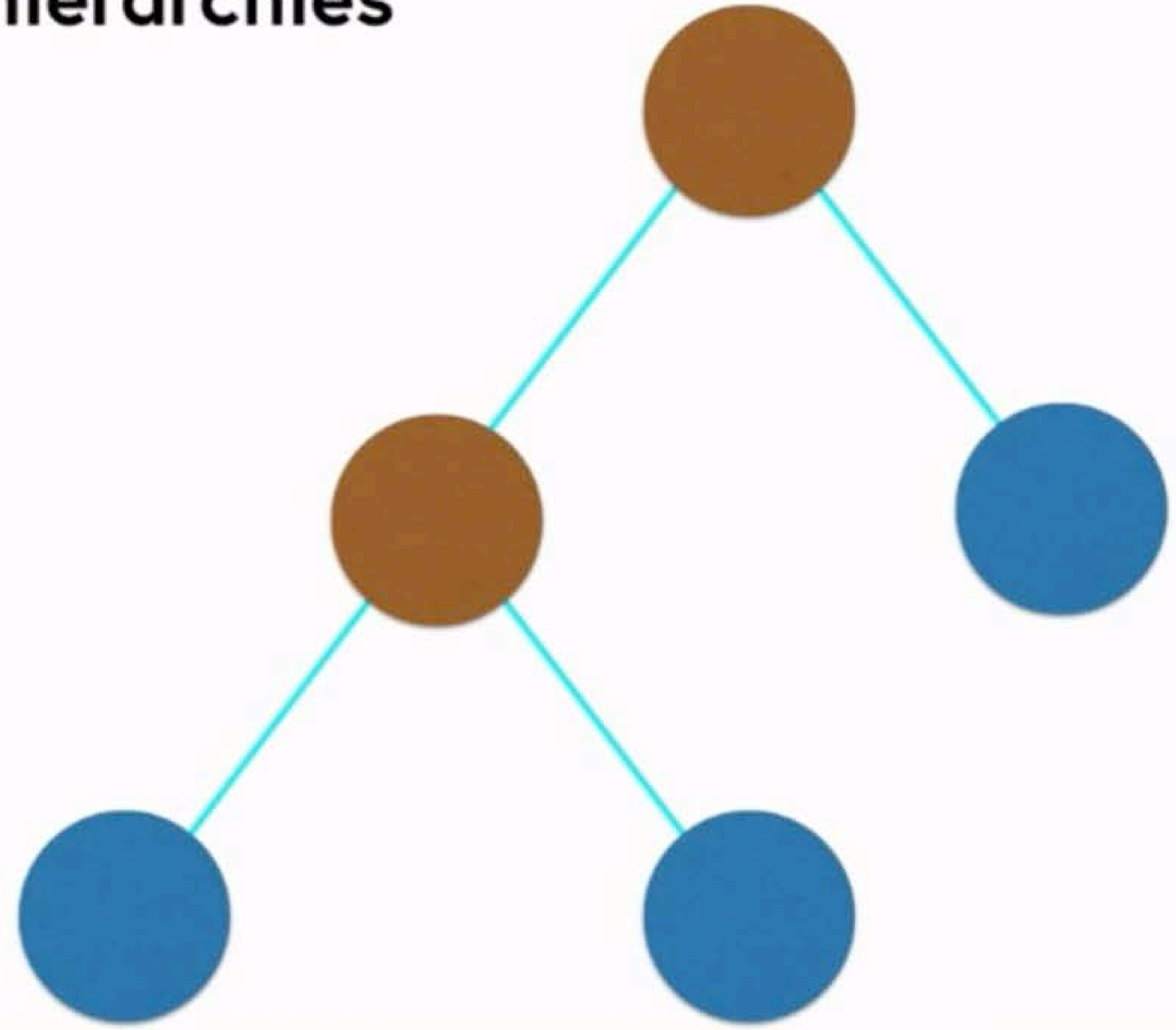
… and if not, to fail.

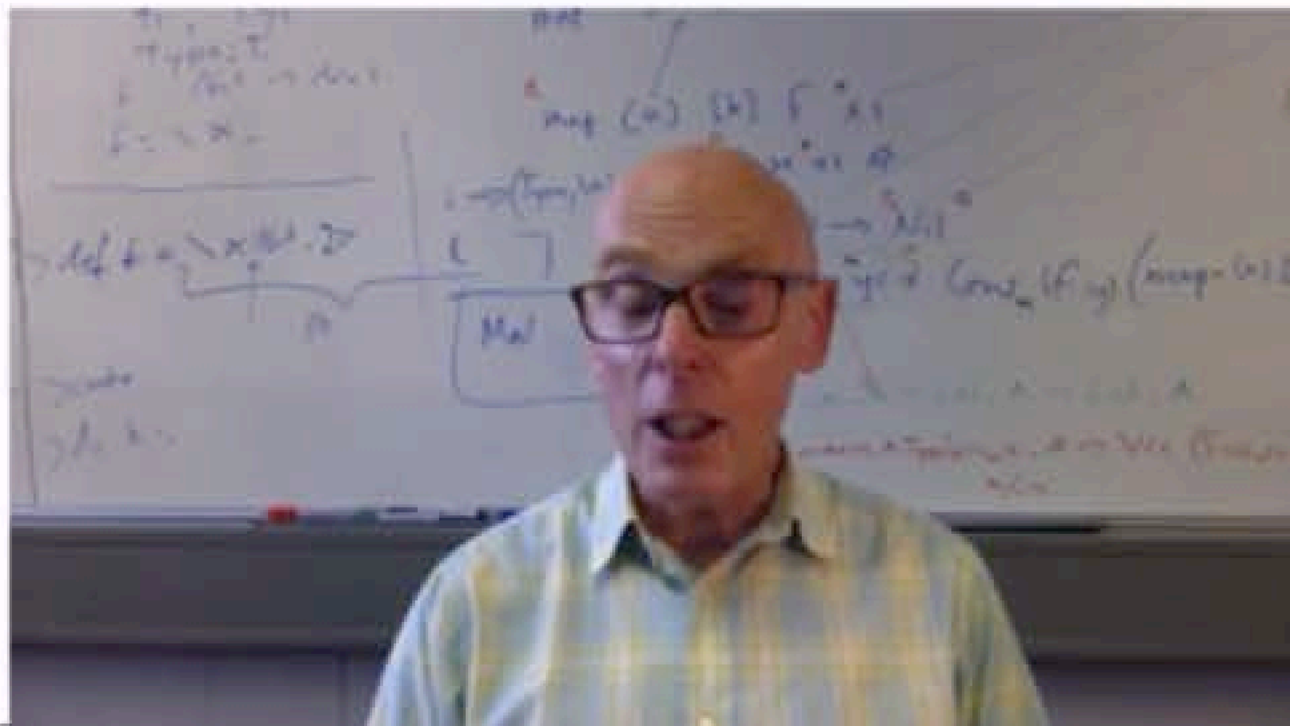
A **supervisor** will do the job of dealing with the failure …

… restarting and taking other appropriate actions.



Supervision hierarchies





How to build a supervisor?

The supervisors spawns and links to the worker using `spawn_link`.

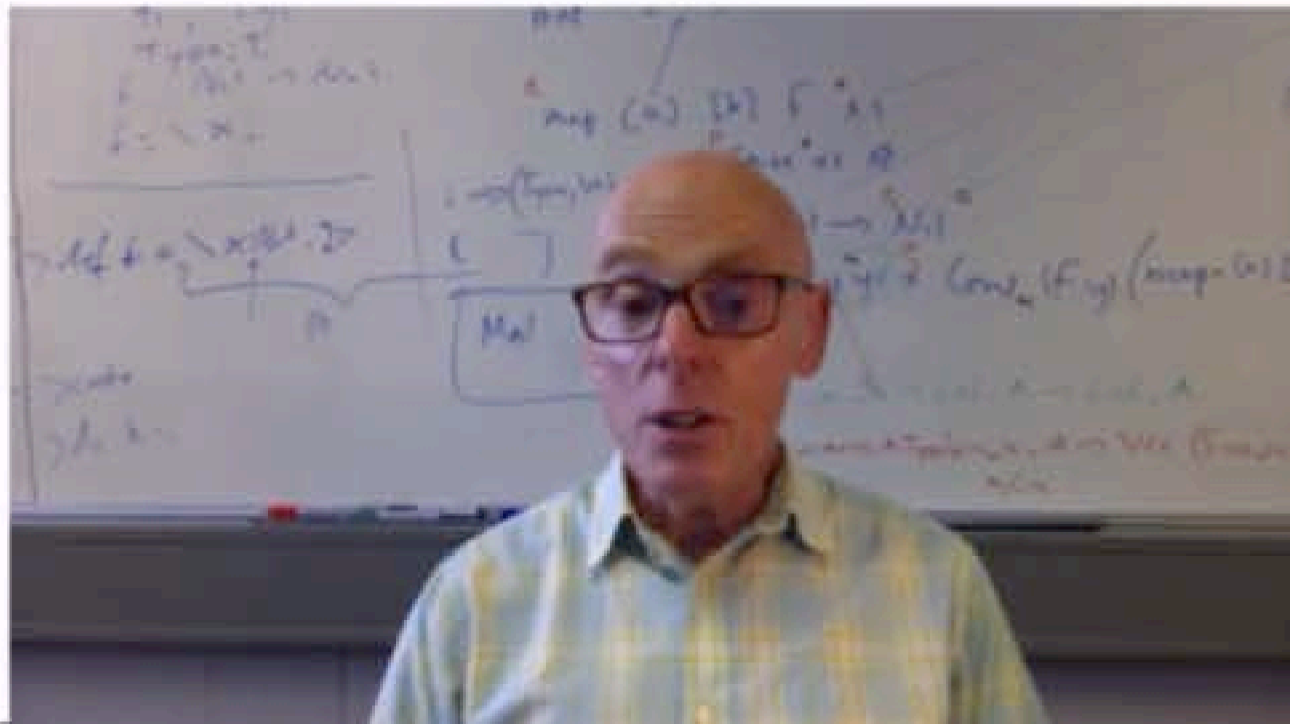
The supervisor monitors process failures …

… and restarts the process / all processes / some processes.

… until some threshold (per second) exceeded.

The supervisor can choose to die itself …

… letting its supervisor handle the problem.

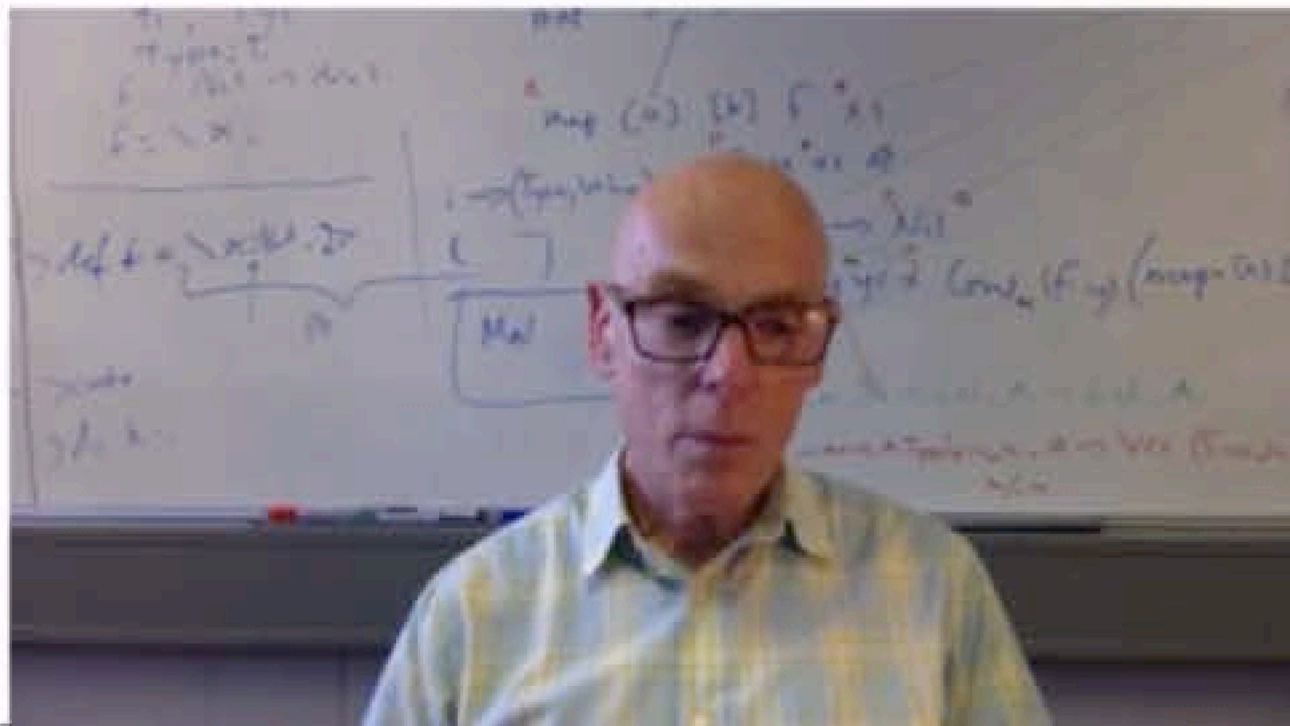


Link or monitor?

It is also possible to monitor processes in Erlang.

A monitor provides a **one way**, **asymmetrical** linkage between processes.

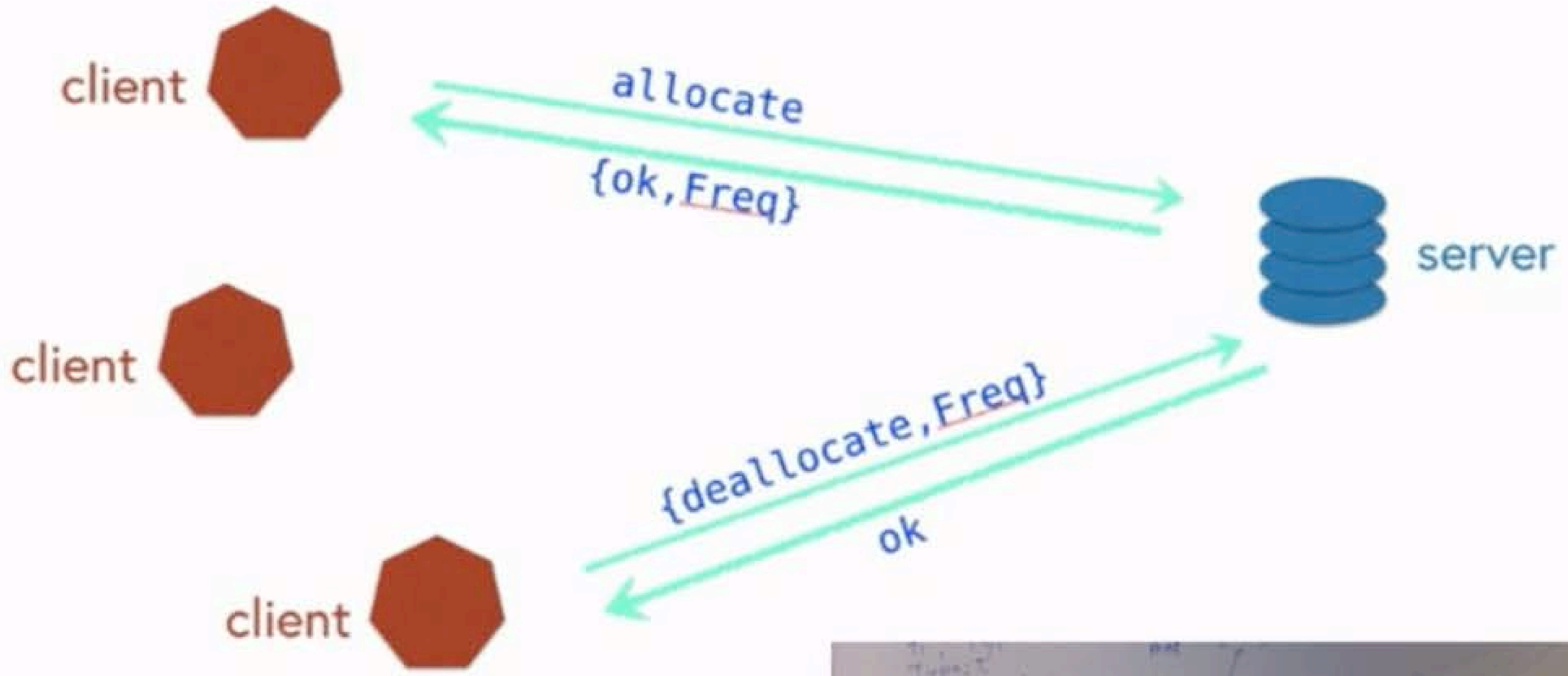
We link supervisor and worker so that a worker will be killed if the supervisor dies … can you think why?

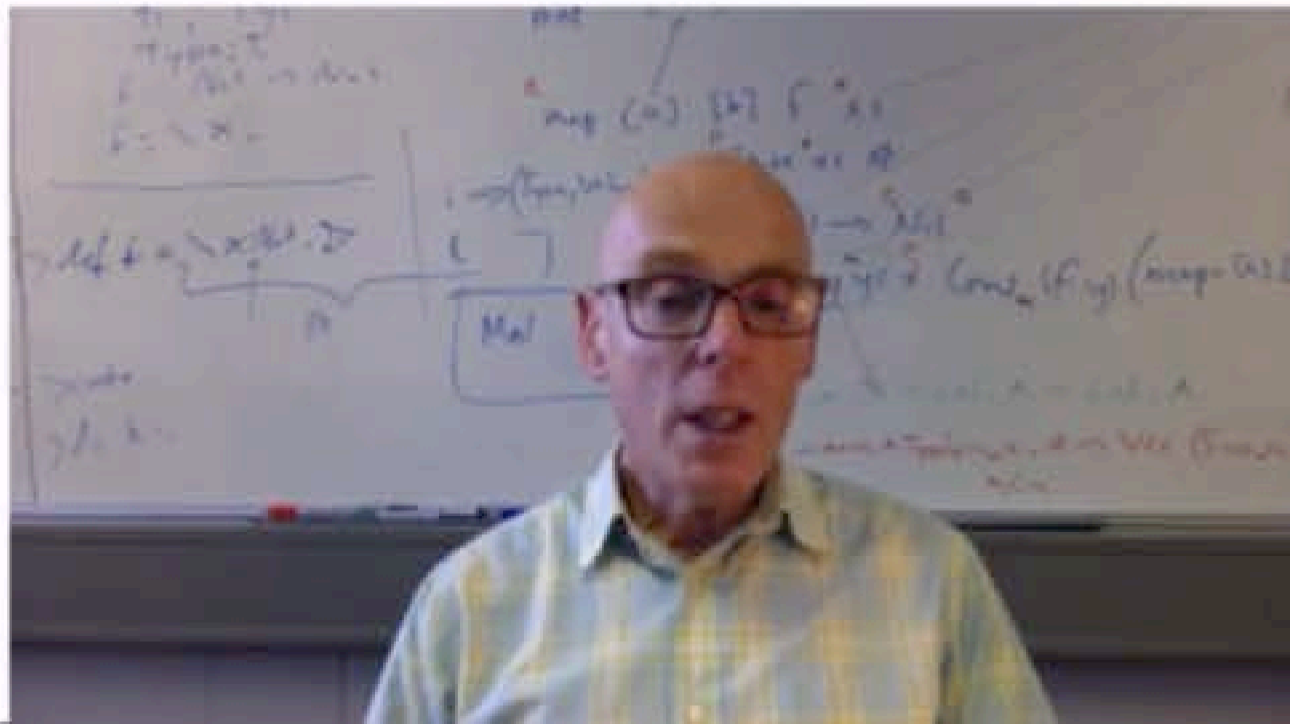


A supervision scenario

Build a frequency multi-server system, with two servers, supervised by a single supervisor.

Try out various restart strategies for the system, using the observer to check which processes have been restarted.





University of Kent