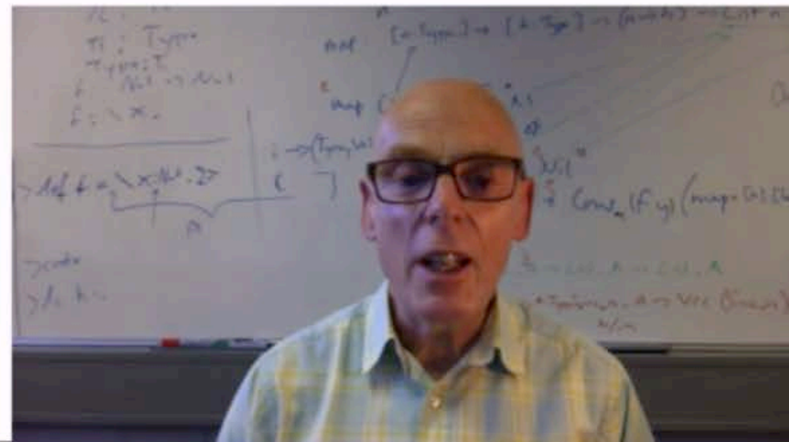

University of
Kent

University of Kent

Links, signals and messages



University of Kent

The Erlang approach

How to deal with an unexpected situation?

University of Kent

Let it fail!

and have another part of the system solve the problem.

University of Kent

Dealing with abnormal termination

When a process terminates abnormally,
it sends a *signal* to all the processes linked to it.

Why not just send a message to them?

The linked process might never deal with the message.

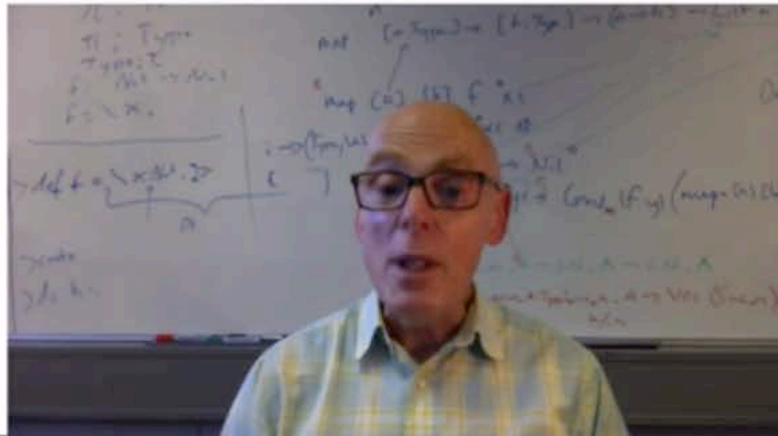
Even if it does, the process has to contain explicit code to deal with this …

University of Kent

Signals ≠ Messages

When a process terminates abnormally, it sends a *signal* to all the processes linked to it, with the reason `killed`.

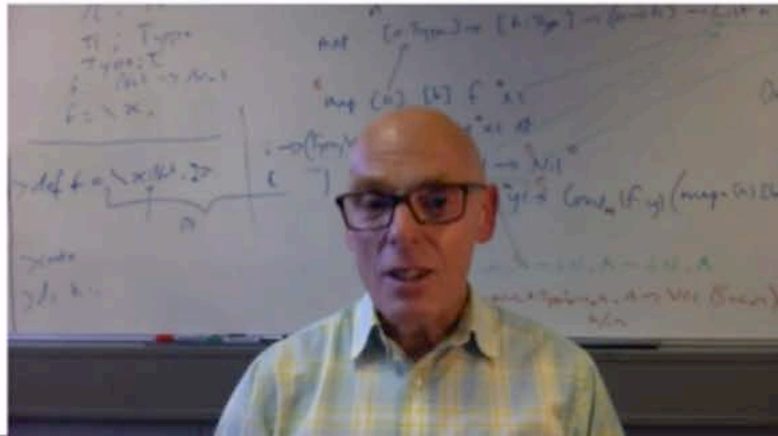
Signals propagate immediately … and the default behaviour on receiving a signal is to terminate (abnormally) yourself.



University of Kent

Signals and messages *are* connected

If we're to be able to deal with processes failing … we need to know when a process has failed, without being killed ourselves.



University of Kent

Signals and messages *are* connected

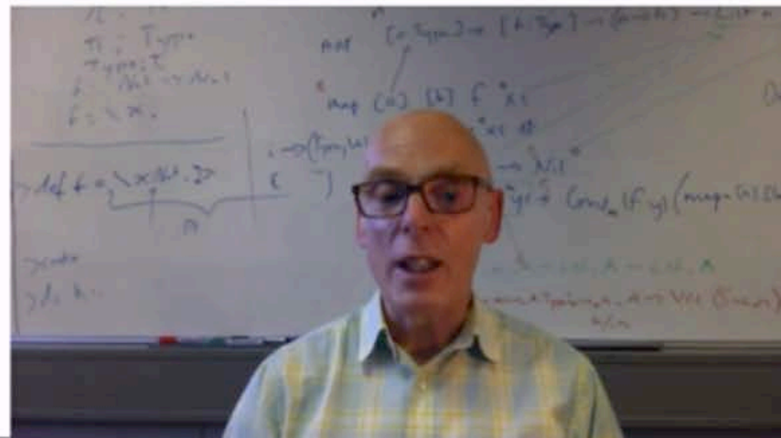
If we're to be able to deal with processes failing … we need to know when a process has failed, without being killed ourselves.

In a process, call

```
process_flag(trap_exit, true)
```

then *exit signals* to that process are converted to *messages:*

```
{'EXIT',FromPid,Reason}
```



University of Kent

Fine-grained control

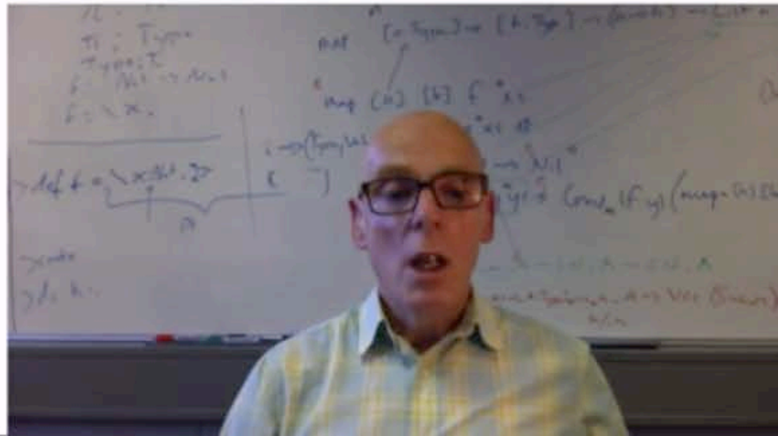
Trapping exits is controlled *per-process* and *dynamically*.

Switched on by calling

```
process_flag(trap_exit, true)
```

and off by

```
process_flag(trap_exit, false)
```



University of Kent

Sending `exit` signals

Exits happen for all sorts of reasons: e.g. division by 0, …

Can be triggered in a process itself by calling `exit(Reason)`.

Can be caused in another process by calling `exit(Pid,Reason)`.

Normal termination has the reason `normal`, any other reason is abnormal.

University of Kent

How a process reacts on receiving an exit

Exit type	Initiated by	Not trapping exits	Trapping exits
Normal	`exit(Pid,normal)`	Nothing	Receives `{'EXIT', Pid, normal}`
Abormal	`exit(Pid,Reason)`	Terminates abnormally	Receives `{'EXIT', Pid, Reason}`
Kill	`exit(Pid,kill)`	Terminates abnormally	Terminates abnormally

University of Kent

What an elegant design!

Signals give a mechanism orthogonal to messages, guaranteed to be able to take down a system.

Trapping exits allow us to handle `exit` signals programatically, as messages, so that we can make more complex recovery code.

But, `exit(Pid,kill)` is guaranteed to override exit trapping, so we can still be sure of being able to kill a process.