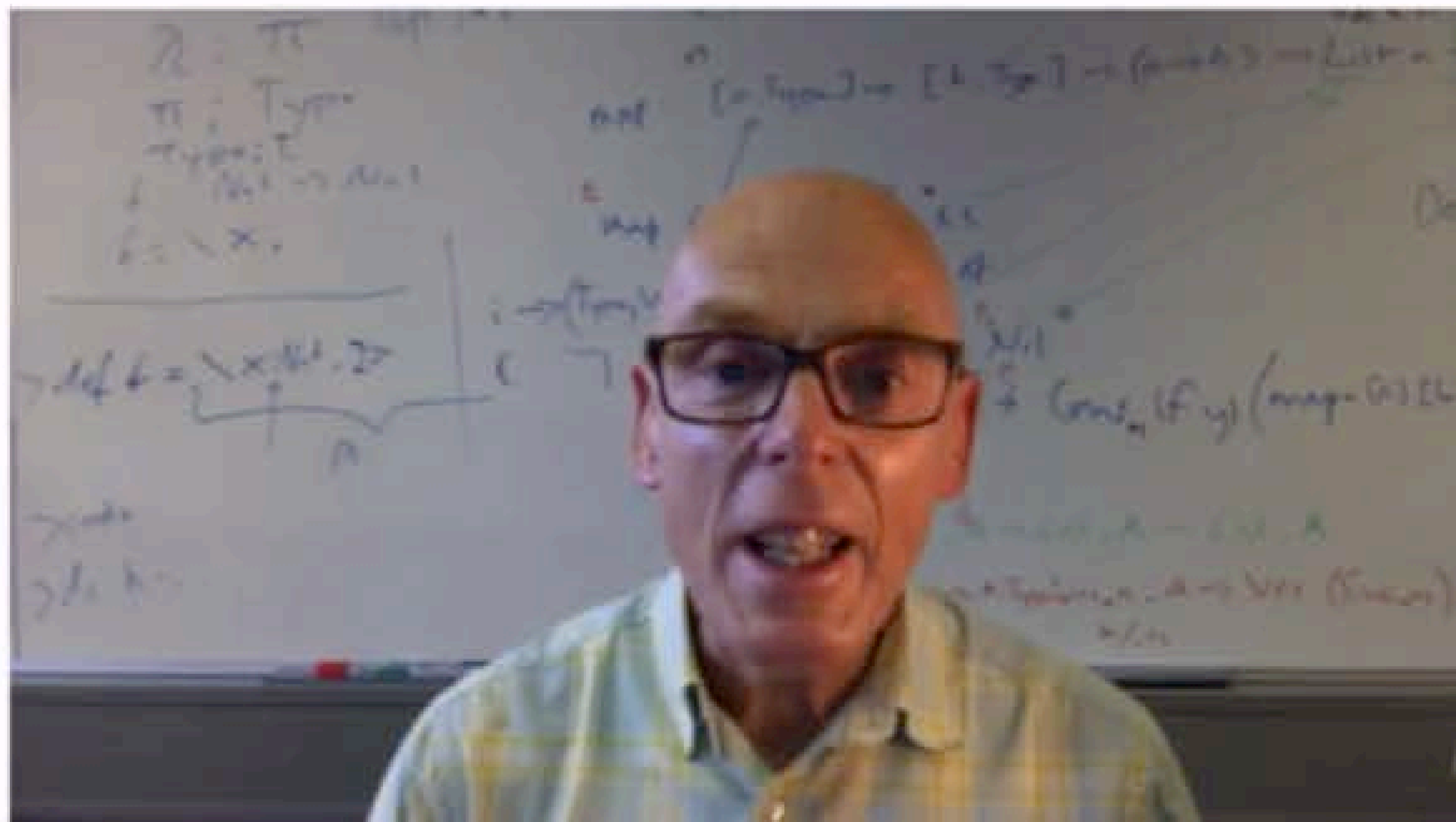
University of Kent

Making code robust

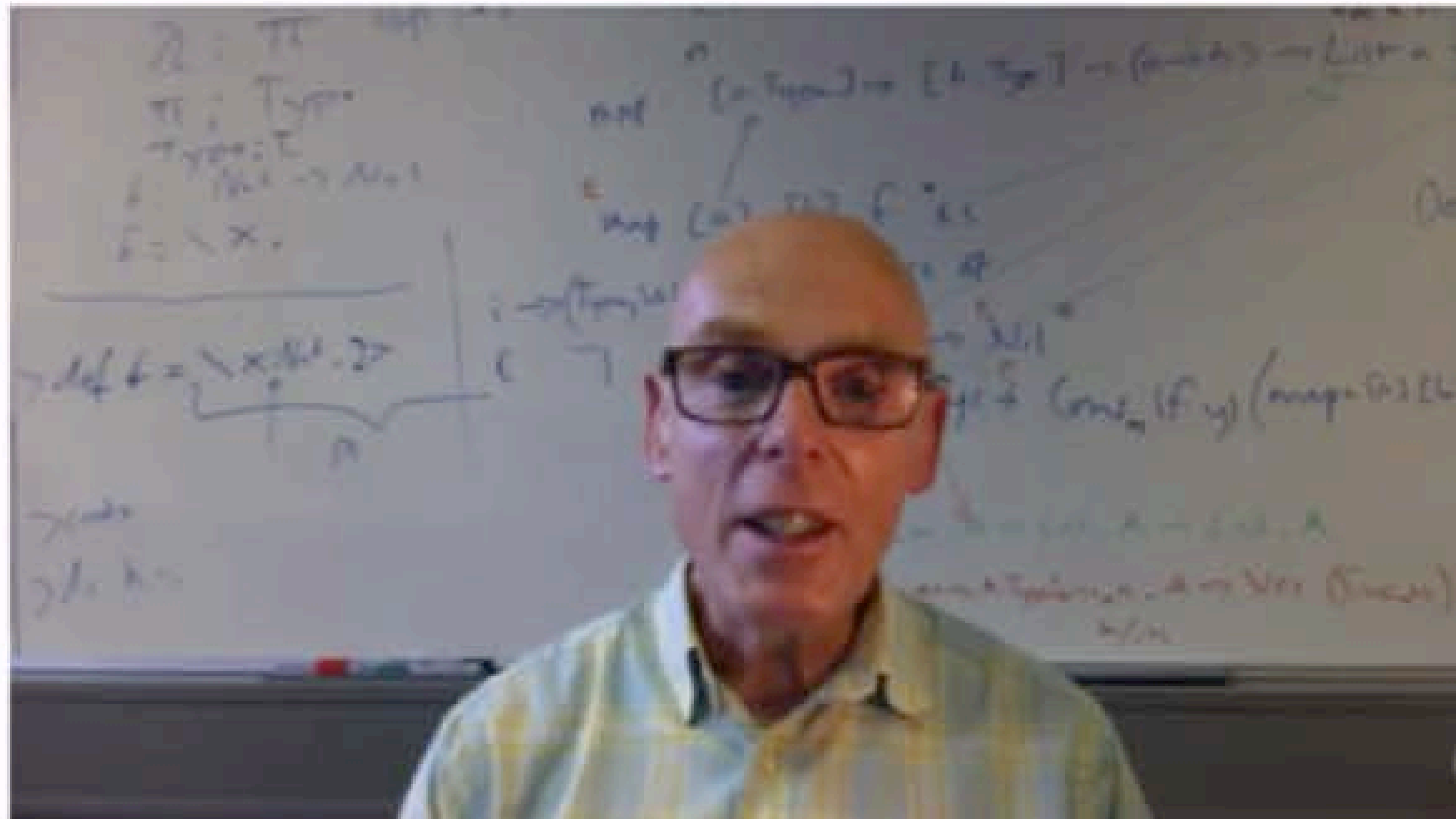


Process lifetimes

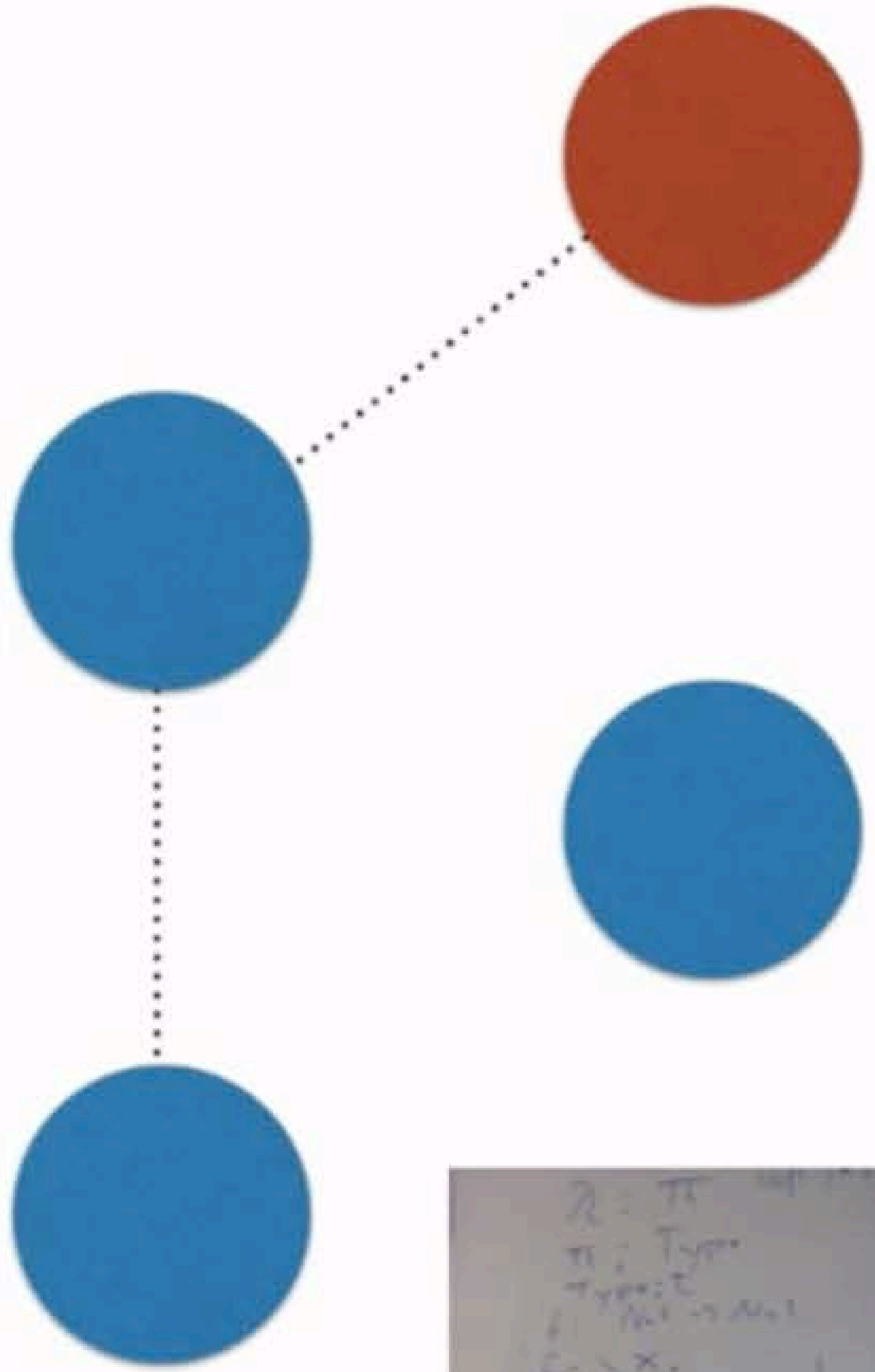
A process can execute indefinitely ...never terminate.

A process can terminate normally.

A process can fail ...or terminate abnormally.



Linking processes

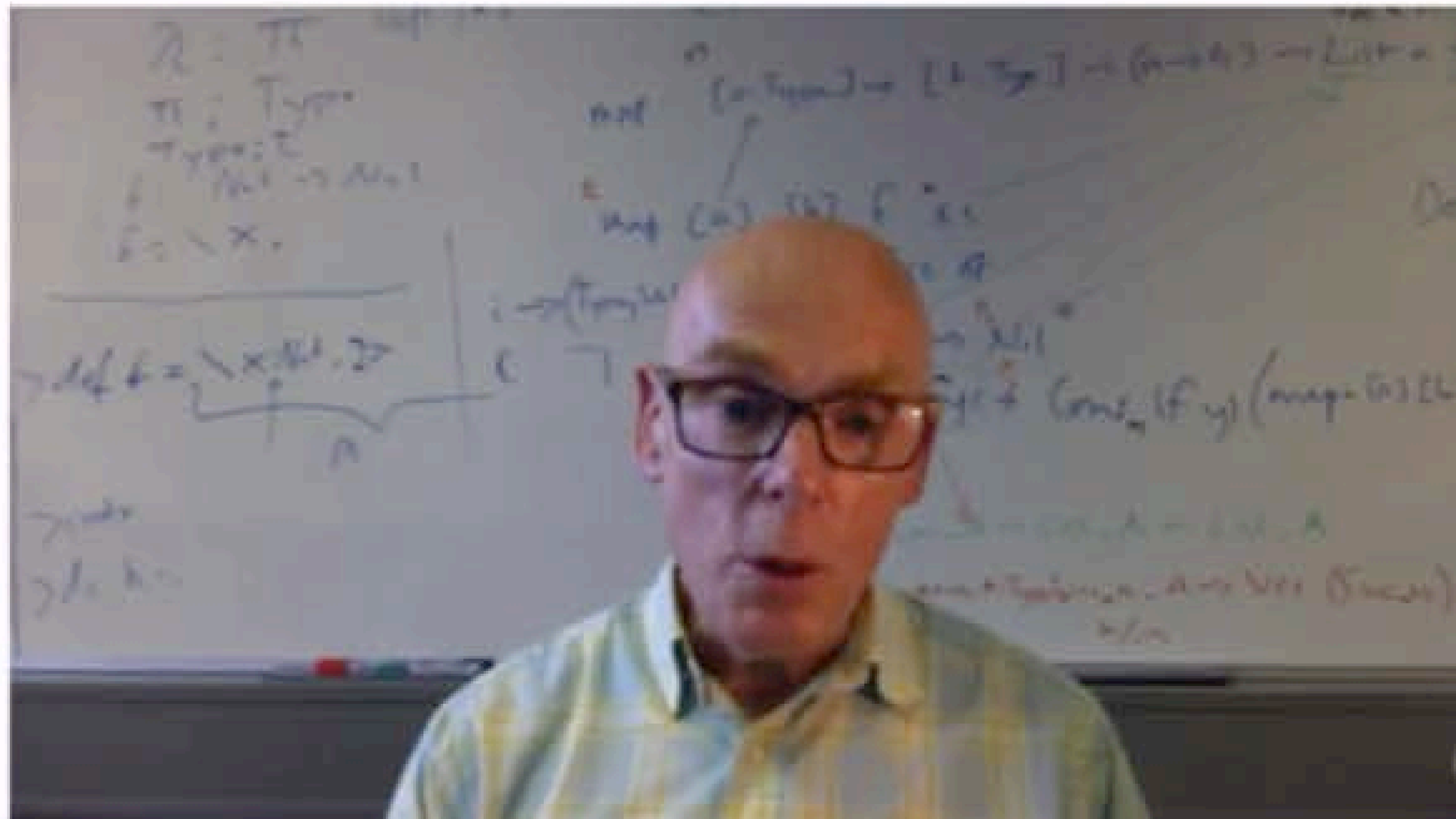


Call `link(Pid)` in one a process to link to …

… the process with process id `Pid`.

If one process fails, linked processes fail too …

… and processes linked to those will also fail.



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

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.
```

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,1,Allocated),
  {[Freq|Free],  NewAllocated}.
```

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.
```

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,1,Allocated),
  {[Freq|Free],  NewAllocated}.
```

Supervisor and worker

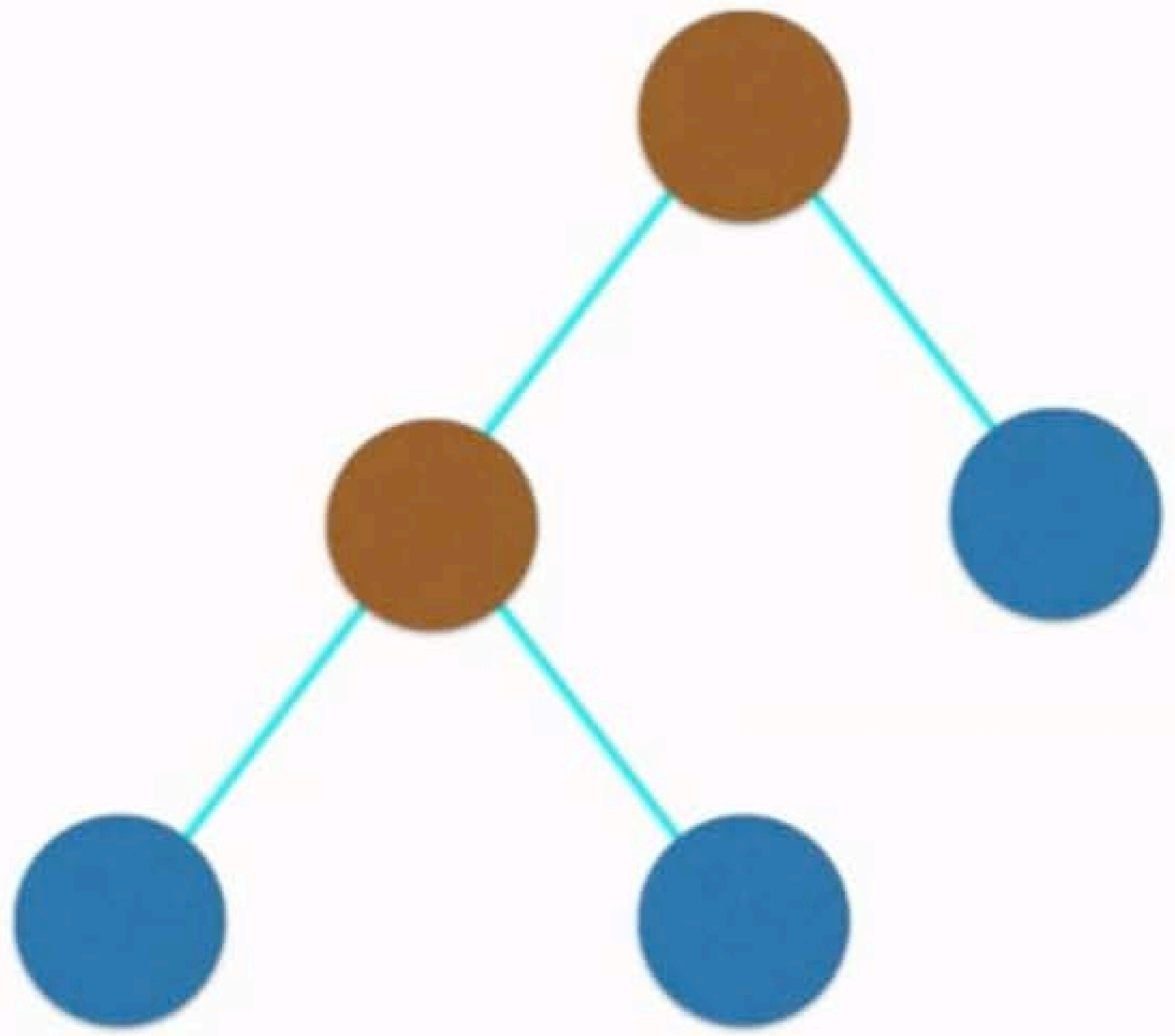
Design **workers** to do a particular job …

… assuming the rest of the world behaving ok,

… and if not, to fail.

A **supervisor** will deal with the failure …

… restarting / taking appropriate action.



Catch exceptions with `try … catch …`

If it `throws` the `div_by_zero` exception, then we return a tuple saying so …

```
try eval (Env,Exp) of
  Res ->
    {ok, Res}
catch
  throw:div_by_zero ->
    {error,div_by_zero}
end
```

Dining Philosophers

http://soarlab.org/people/

Race conditions

No guarantees about ordering (except point to point).

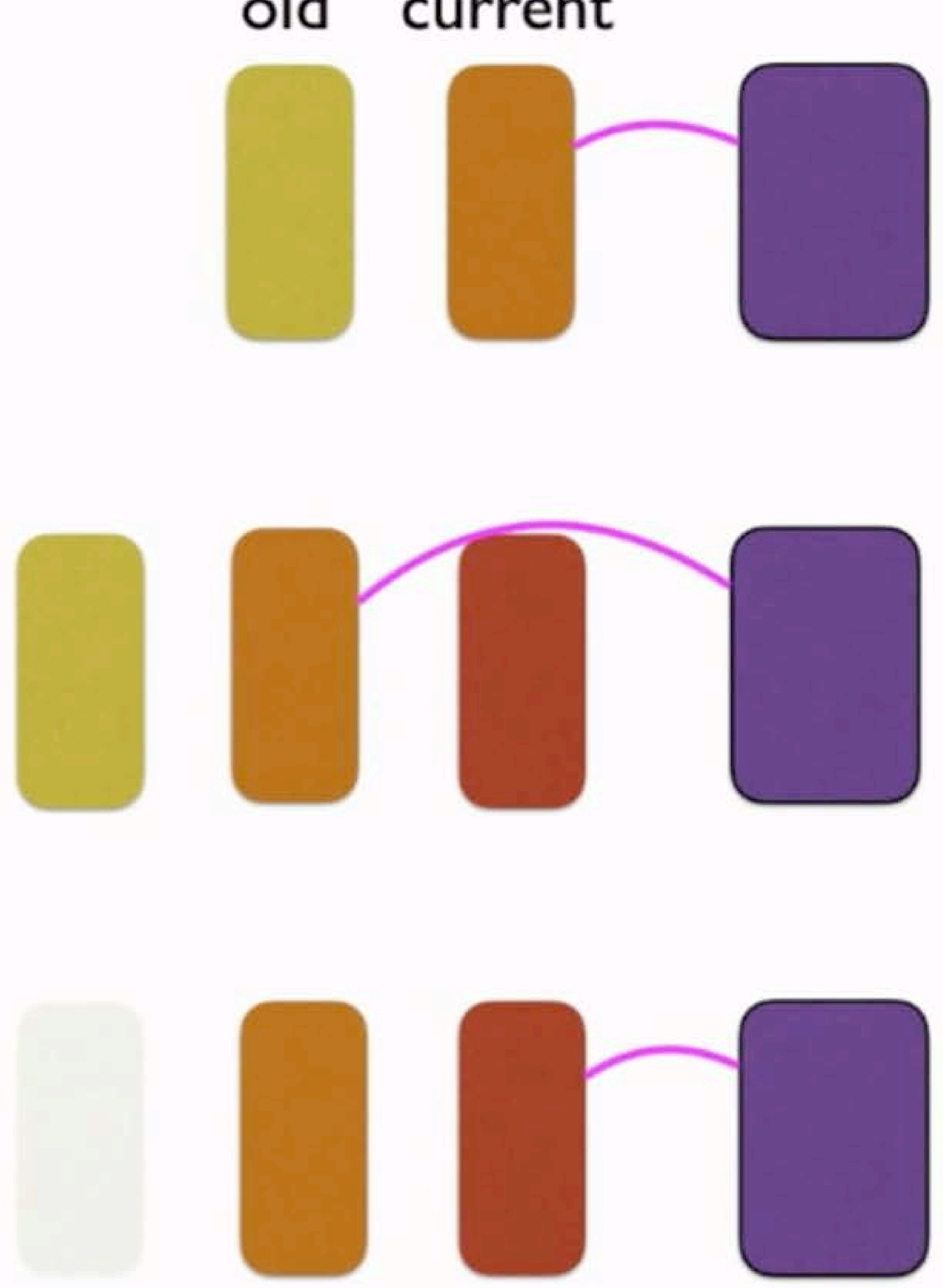
Move from a purely sequential runtime to a concurrent one.

http://www.javacreed.com/wp-content/uploads/2014/11/What-is-Race-Condition-and-How-to-Prevent-It.png

$64,000 question: using the new code

After the new code for `Foo` is loaded, the module using it will use the same code …

…until there's a call to any function in `Foo`, when it switches (for *all of* `Foo`).



University of Kent