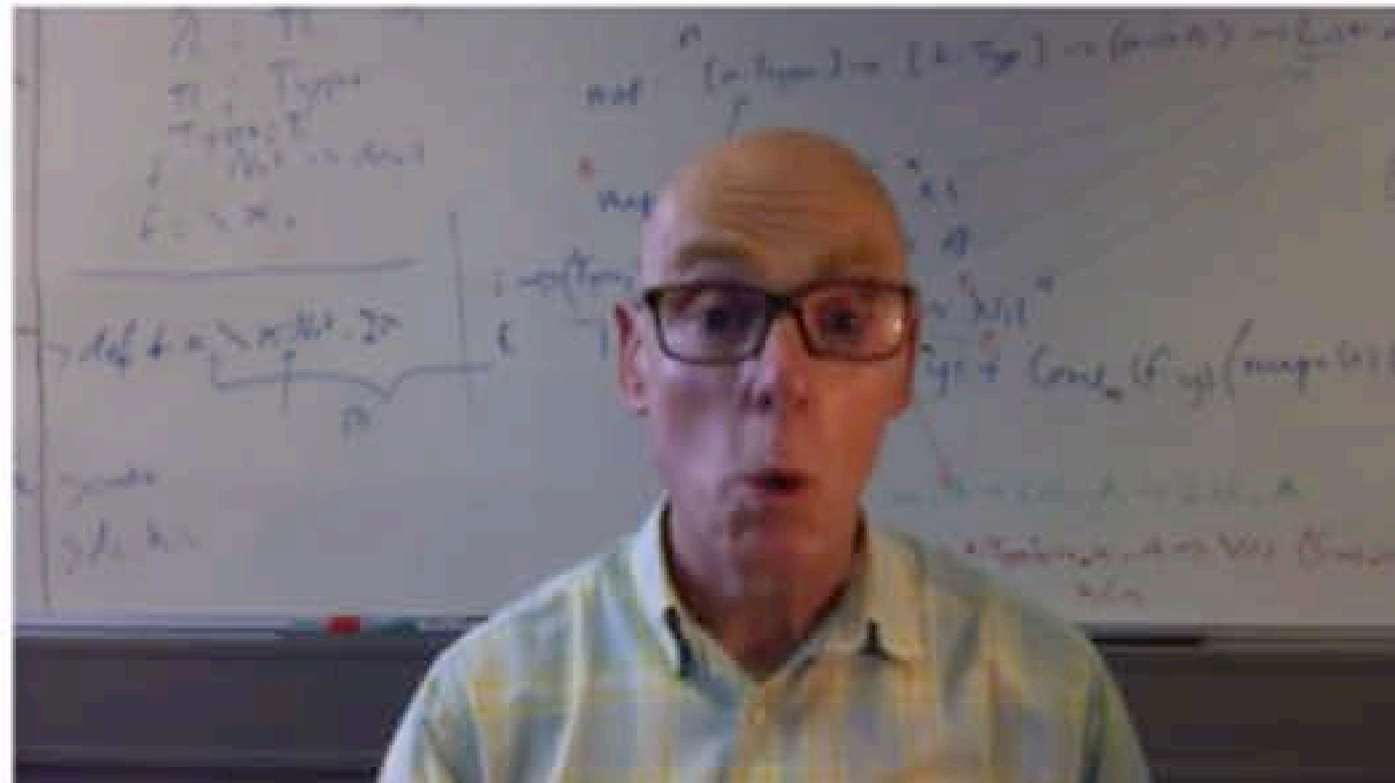
University of Kent

Concurrency and scheduling

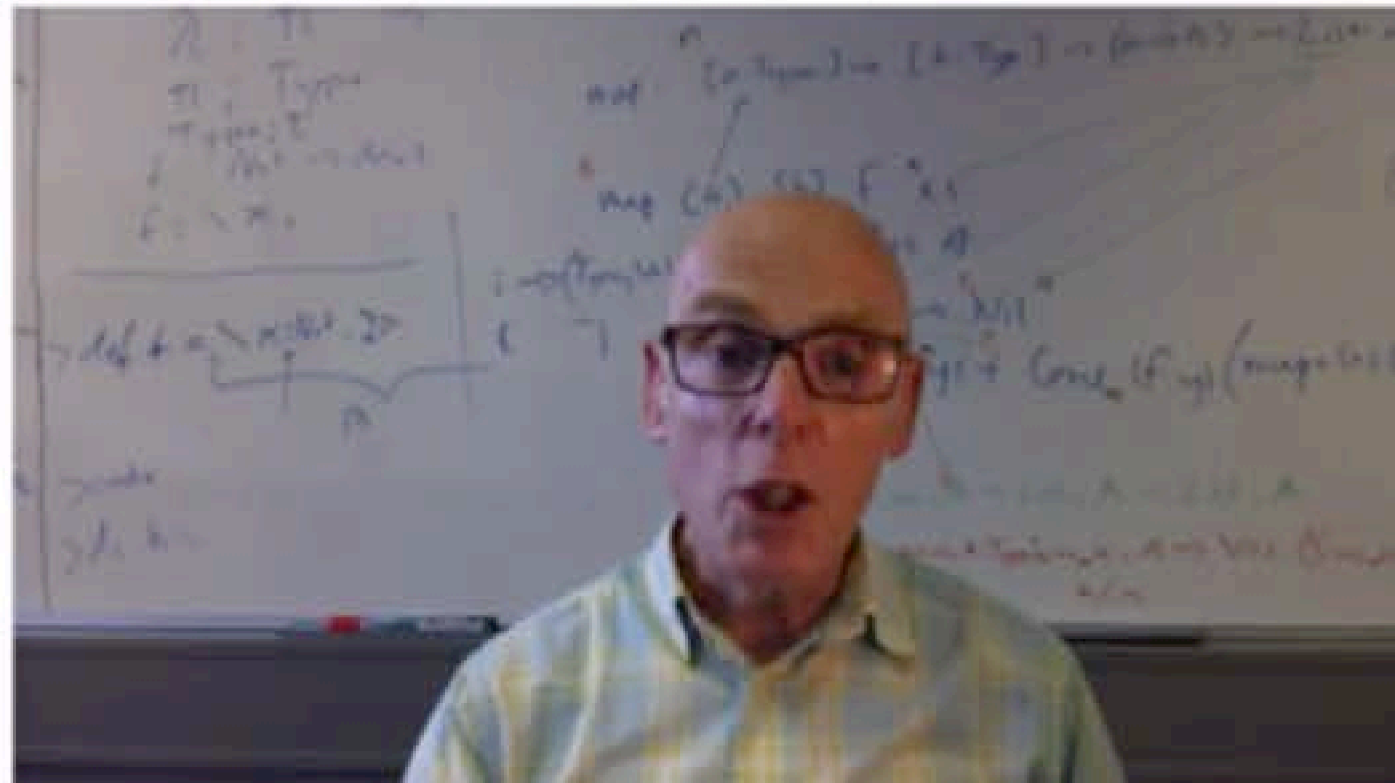


Concurrency *versus* parallelism

Concurrency means the possibility of running independently, and perhaps at the same time.

On a single processing element, concurrent processes timeshare, mediated by a scheduler.

On a multicore processor, there is, potentially, concurrent activity on each core, with each scheduled separately.



Scheduling: running, runnable and waiting.

A process is *waiting* if it is in a `receive` statement and waiting for a message to arrive in the mailbox, and the `after` timer is still counting down to zero.

When a message is received, or the timer reaches zero, the process becomes *runnable,* and it is placed at the end of the *run queue*.

When a process is scheduled out, the process at the front of the run queue starts *running*.

A running process is *de-scheduled* after 1000 reduction steps (function calls), it is then placed at the end of the run queue.

It is also *de-scheduled* if it enters a `receive` statement with an empty mailbox.

Going deeper …

In fact it's a bit more complicated than was just explained …

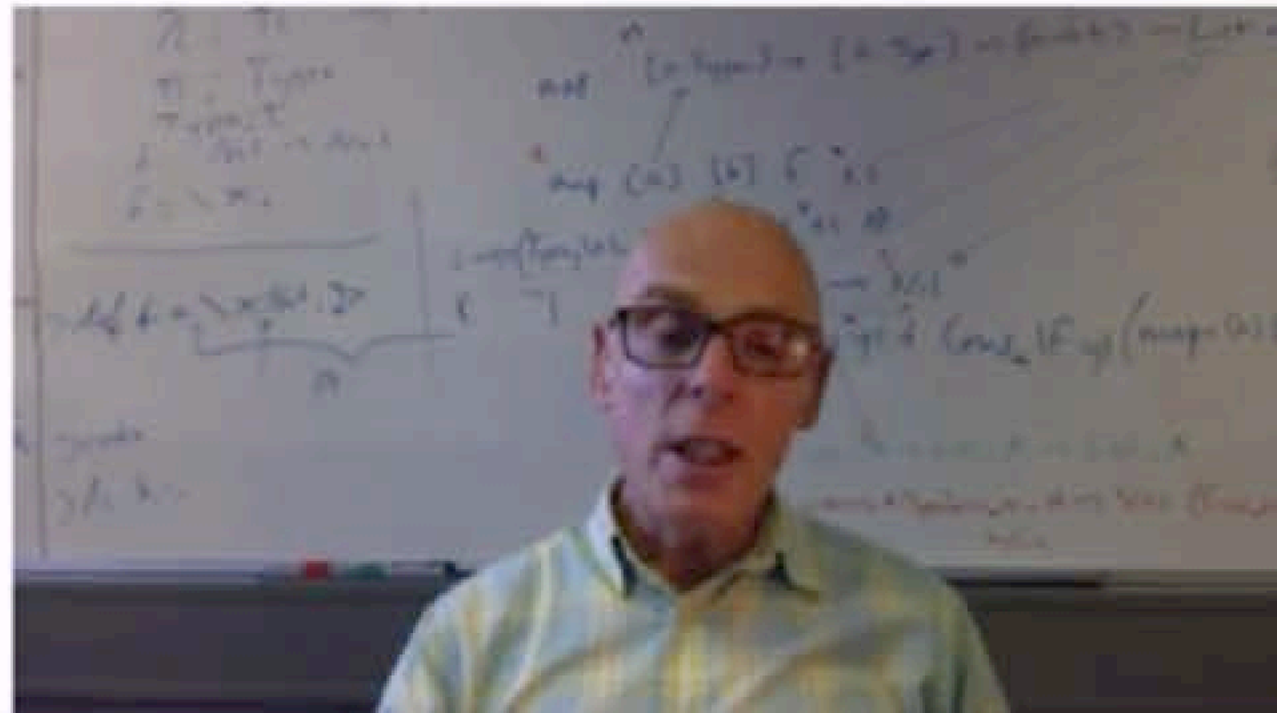
…with multiple priorities provided.

These get used for processes internal to the Erlang system …

… in user programming it is **best to stick with a single priority**.

The BEAM also has integrated support for IO …

…so that it "just works" in a concurrent environment.



Multicore BEAM

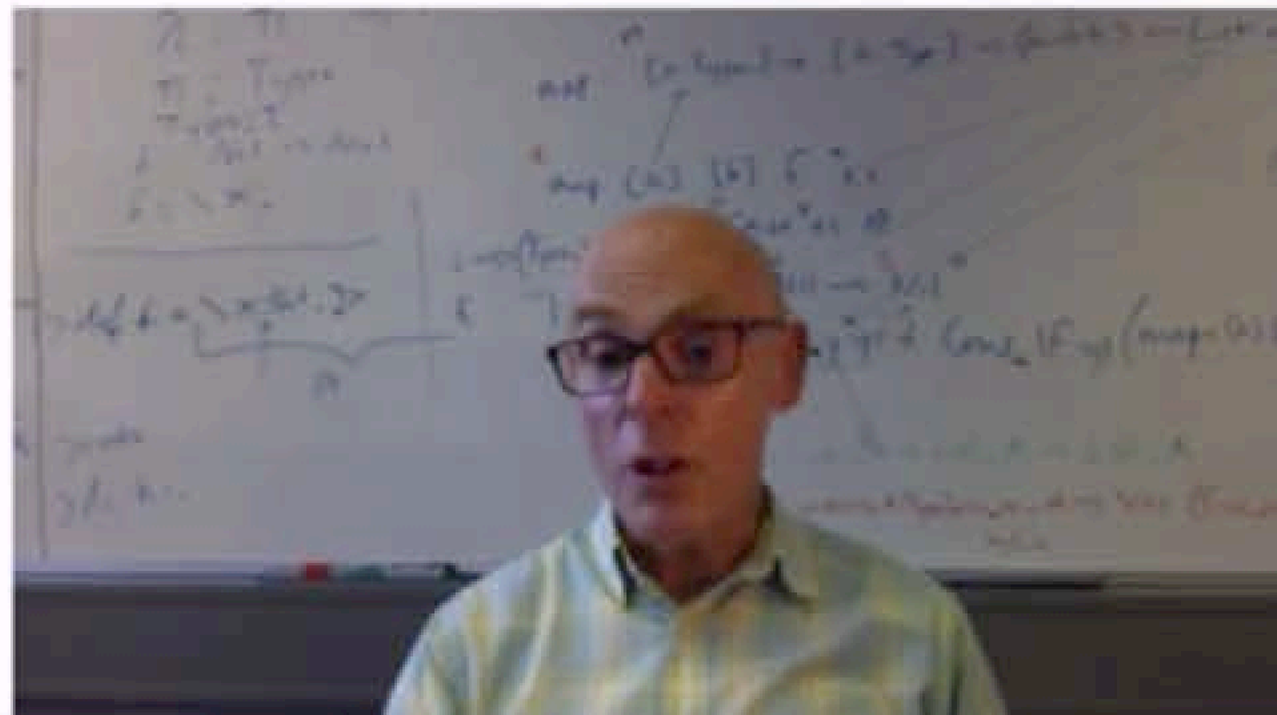
On a multicore platform BEAM will have multiple run queues …

…by default one run-queue per core

…but this is configurable.

Processes are scheduled on the same core that spawned them …

…but they can migrate between queues by a *work-stealing* algorithm



University of Kent