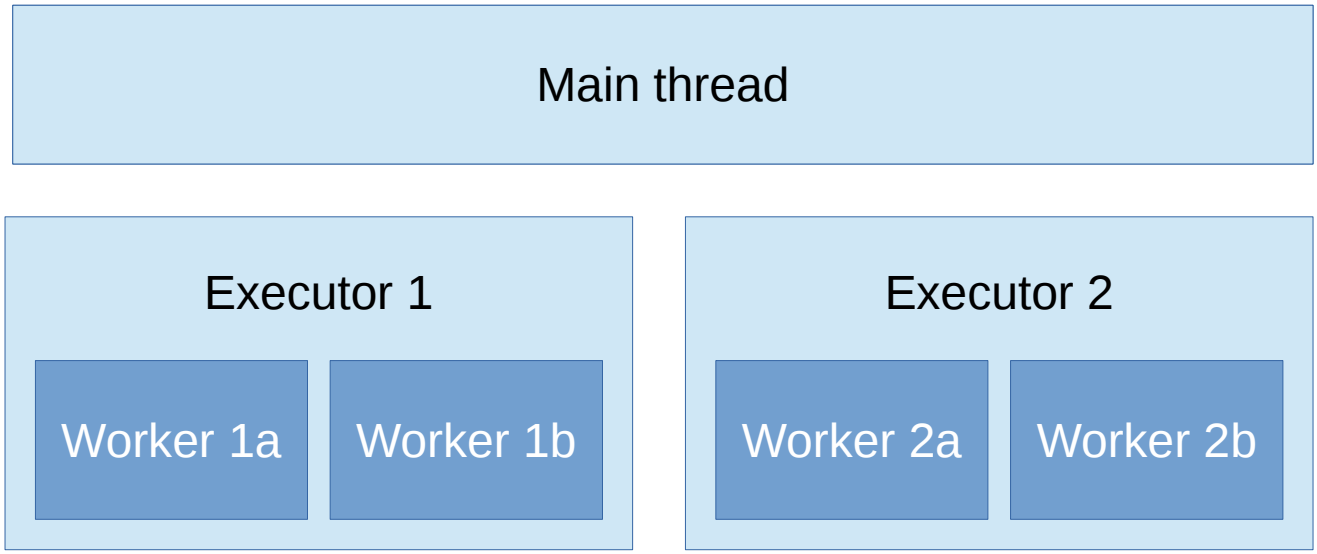
Unhandled exceptions in Phalanstery

A Phalanstery-based application is composed of a number of Ada tasks : one for the main program, one per task executor, and one per worker task within the executors.



All of these tasks can fail if an unhandled exception occurs as a result of an error management oversight, and currently this results in a description of the exception being printed out for debugging purposes, followed by some cleanup actions and the exception being finally re-raised.

But that is not enough : Ada, by design, does not terminate the program if one sub-task of it fails by letting an unhandled exception escape. Which, may be fine most of the time, and even desirable in critical systems. But in the context of Phalanstery's event model, it means that the remaining tasks can easily end up waiting for the failed task through a protected object, creating a deadlock that the Ada runtime will not detect.

For our purposes, it would be fine if the entire program got terminated in the rare event where an unhandled exception exits a task (which generally indicates a coding error), so we would like to override this default Ada behaviour and enforce program termination.

This can be done by aborting the environment task in our exception handlers, which in Ada is guaranteed to abort the rest of the program (or, more exactly, the rest of the partition). That is done using package Task_Identification from the supplementary Annex C of the Ada standard.