Step 3

Beforehand: When we did this step, the code proposed was using the Producer/Consumer implementation of a finite state machine. This was in contradiction to the model proposed in the step description, of states as functions with direct calls to each others; however after thinking about it we figured that we could see this description as a model which was correctly implemented by the Producer/Consumer approach. The questions that follow have therefore been answered with the Producer/Consumer approach in mind.

```
- Does introducing a convention solve the readability issue?

- How to extend an app with a new feature?
  Does the approach prevent one to perform invasive changes
  in the existing behavior to introduce a new one?

- How to extend the code so that to support new features, e.g.,
  memory-less tasks, state-full tasks, different frequencies?
```

- The concept of finite state machine helps to understand easily the states of the components, but the implementation, with FLAGS in order to run different FSM in parallel, makes the code hard to comprehend.
- In order to add new features, it is easy to consider either the FSM of the new component or the modified FSM including the new feature, to add corresponding flags and modify the consumer functions accordingly. However, due to the states being hardcoded into such functions, those modifications may interfer with previous behavior and are not riskless to implement.
- It is possible and easy to simply add new flags and message producers / consumers and run them in parallel to the other FSMs. This covers the case of memory-less tasks (no FLAGS) and state-full tasks (use all defined FLAGS). The case of different frequencies does not seem obvious to implement, because the algorithm which runs all the FSMs in parallel is a simple `while loop` i.e. there is no real concept of temporality inside our code except for the `delay` between each loop. In order to unify FSMs running on different frequencies, a temporal model might be required.