

# Parallel design patterns

## ARCHER course

---

Case study: The actor model for ATC



# Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

<https://creativecommons.org/licenses/by-nc-sa/4.0/>

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Acknowledge EPCC as follows: “© EPCC, The University of Edinburgh, [www.epcc.ed.ac.uk](http://www.epcc.ed.ac.uk)”

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

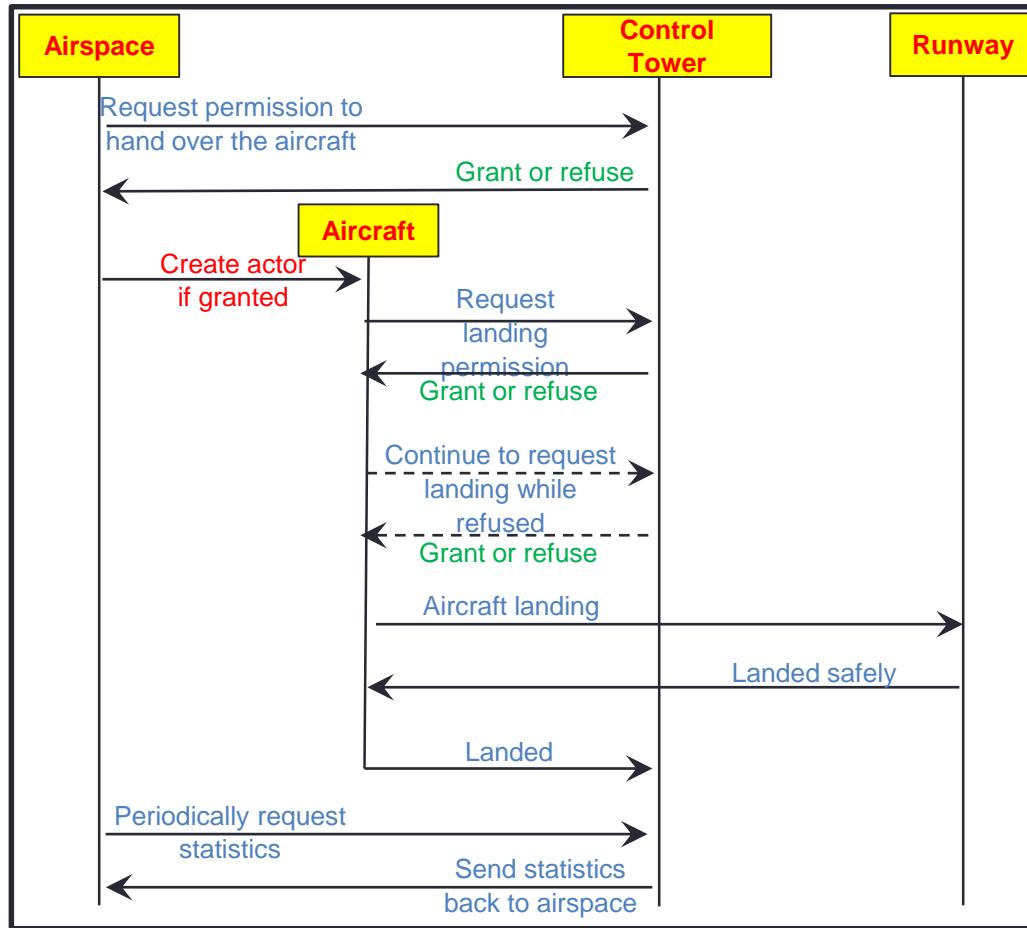
# The problem



# Use the actor pattern to model this via MPI

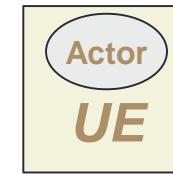
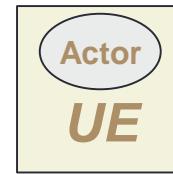
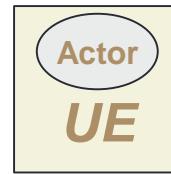
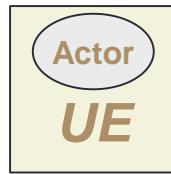
- Four types of actor
  - Air traffic control tower
  - Runway
  - Airspace (the operator introducing/handing aircraft to the ATC tower)
  - Aircraft
- Control tower, runway and airspace are created once at model start up and exist till the end
- Aircraft are much more dynamic, created as the model runs and many actors can be created (and can die)

# Interaction pattern



# Some hints

- A skeleton implementation is included
  - Use it if you want, entirely up to you
  - Worked solutions are also available too
- I strongly suggest one actor per UE as much simpler to do



- I also provide the process pool implementation where workers are actors
- Lots of details in the hand out

# ProcessPool: Important Considerations

- Your code must call `MPI_Init` before any of the following calls
- Every process must call `processPoolInit`
- The master process keeps track of which processes are active, but any process can call `startWorkerProcess` to request that a new worker is created/awoken. An ID is returned by this call which can be used to send a message to the new process.
  - The communications between the master and worker required to make this happen occur “behind the scenes”

# ProcessPool: Important Considerations

- The master process will probably do the job of creating the initial actors in the simulation
- All actors should be implemented using workers

```
while (workerStatus) {  
    int parentId = getCommandData();  
    // insert code here which implements being an actor  
  
    workerStatus=workerSleep();      // This MPI process will  
    sleep, further workers may be run on this process now  
}
```

# ProcessPool: Important Considerations

- All actors **must** call `shouldWorkerStop` at regular intervals to allow the master to terminate the program if required. If `shouldWorkerStop` returns true, then it is **your responsibility** to ensure that the flow of control returns to the line after the call to `workerCode`
- The process pool uses MPI tags 16384 & 16383
  - So avoid using these tags in your code