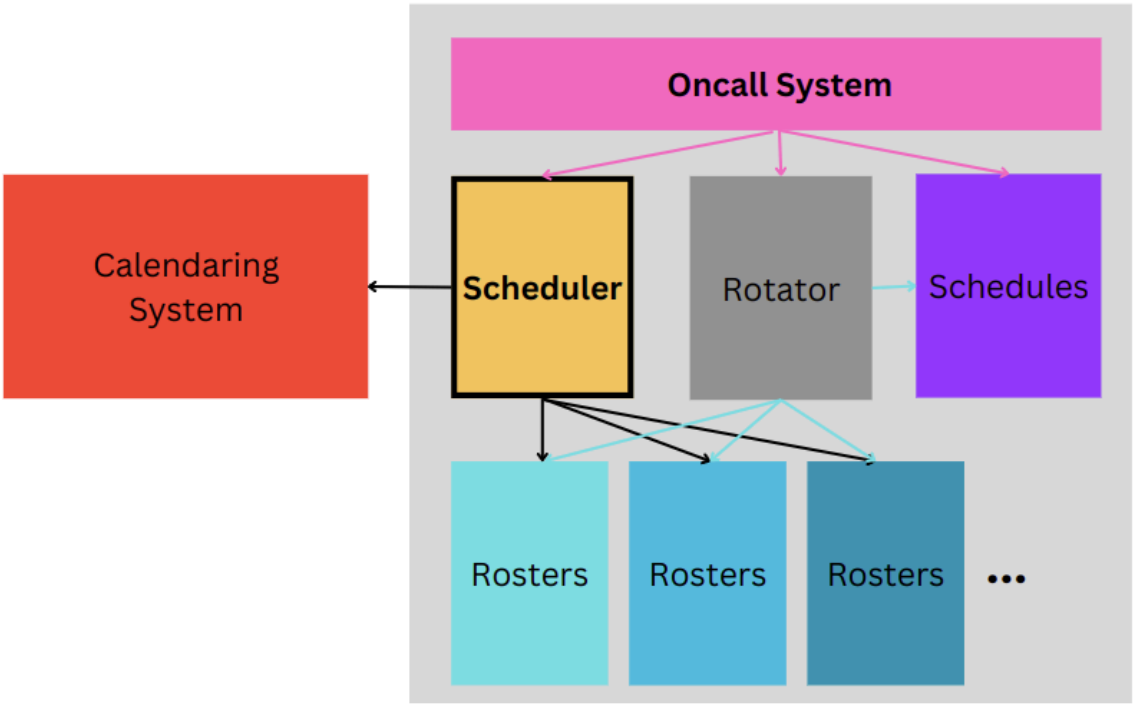
Roster Scheduler: Design

Overview

The core problem we are trying to solve is updating team rosters, i.e. scheduling individuals at a convenient time to do oncall shifts for an oncall schedule. Rotating oncall between teams should be left as an exercise for a larger system.

The scheduler is a small composable element of a larger oncall system, which is responsible for:

- Storing schedules and rosters.
- Rotating between rosters.
- Calling the scheduler when necessary.



The Scheduler interacts only through APIs. By working strictly through APIs the Scheduler can be composed into various oncall systems, calendaring systems, etc… These APIs should probably be supported by a plugin architecture i.e. the core code uses a simple API which actually uses a plugin to talk to the external system. The scheduler could be statically configured to use specific plugins to interact with roster and calendaring systems, or it could be incorporated as part of the request to the scheduler (see below under “APIs” for more detail). Decision TBD.

Design

APIs

The three APIs that need to be created are:

- **Scheduler API**: Call the scheduler with a team roster identifier and ask it to extend or modify the existing roster. The scheduler returns a suggested update to the roster, but does not modify the roster directly.
- **Roster API**: Allow the scheduler to call out and get roster details.
- **Calendar API**: Allow the scheduler to call out and get details of an individual's calendar.

A fourth, useful API which is not strictly a function of the scheduler would be:

- **Accounting API**: Calculate the number of oncall hours worked according to various constraints for reimbursement purposes.

Internal Design

The current prototype is stupid, in a good way. On receiving a request through the Scheduler API:

- It requests roster information (Roster API) and identifies which individuals are oncallers.
- It does calendar requests (Calendar API) and marshals the availability of different people according to shifts.
- It uses a linear constraint solver to generate a large set of different options that meet those constraints.
- It then scores the different options, ranks them and returns the "best" option.

In the first instance the scheduler would treat "preferred not oncall" options as hard constraints (i.e. cannot be scheduled). If no valid options are generated by the constraint solver, these constraints would be removed, and the linear solver would be run again. If still no valid options are available, the scheduler would return an error ("No schedule available/overconstrained/resolve manually.").

The scoring metrics can be altered for each roster and should be included as part of the roster configuration, although in general documentation should encourage users not to mess with the defaults since it would should aim to have a sensible "out of the box" experience.

Build and Test

As part of the build and test process the APIs will be fulfilled by YAML backed files. These are NOT intended to be used in production systems. But they are mostly human readable, easy to generate and parse, and make for easy testing. They YAML files are originally generated in python, which makes it easy to explore the data structures before committing to a file format.