Corsika 8

- Input & Steering -

Dominik Baack

Settings vs Compilation

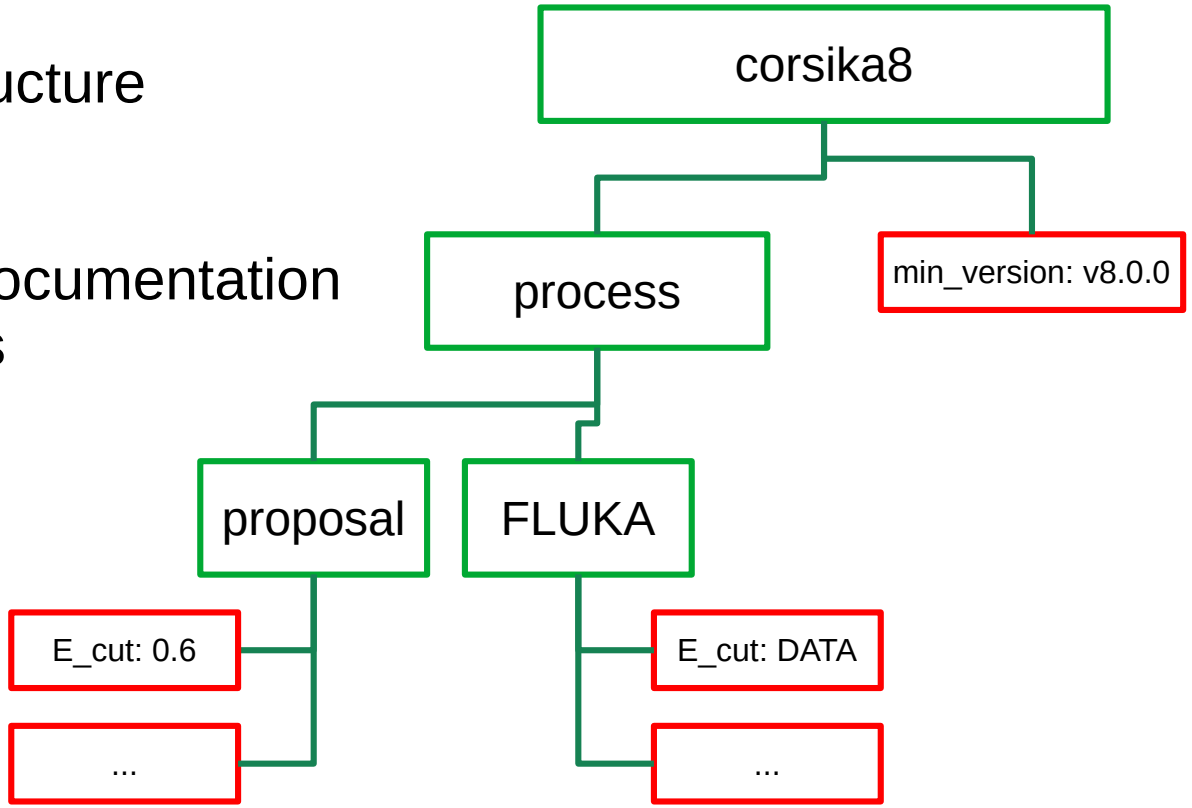
Which fundamental settings should be controllable during runtime without recompilation?

Still open for discussion

- Input File(s) -

At the moment YAML
- Flexible with clear structure
- Human r/w

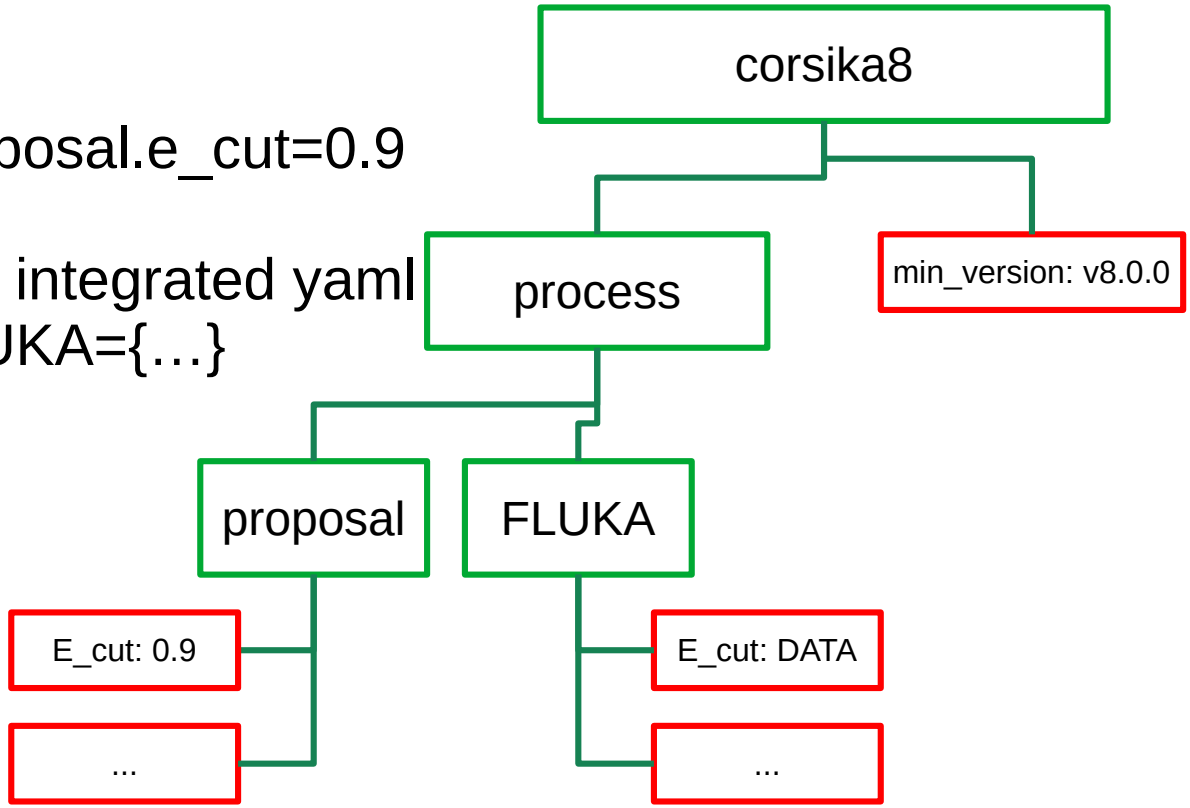
- Requires very clear documentation of all possible settings



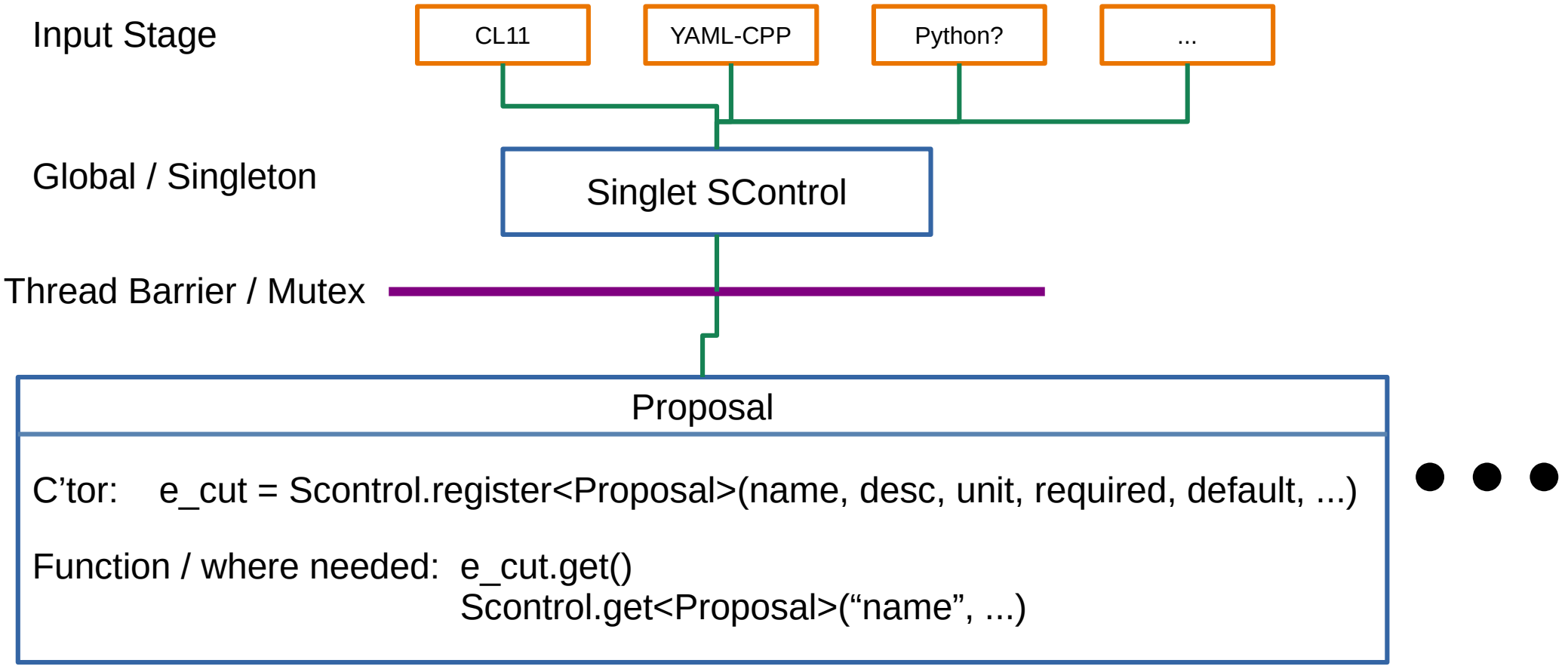
- Command Line -

Precedence over file
- Direct control
  corsika8.process.proposal.e_cut=0.9

- Subtree patching with integrated yaml
  corsika8.process.FLUKA={…}



- Abstraction & Synchronization Layer -



- Abstraction -

Why register?
→ Information's for user -h or –dump_config for example files

Why no callbacks?
→ Cleanup and rebuilding required during parameter change, error prone better to be kept in d’tor

Why templates?
→ Avoid name duplication for input variables
→ Classes needs to inherit from some baseclass to set name=”…”
→ Fallback typeid.name but compiler dependent

Interlude
- OpenMPI Parallelization -

Fixed	vs	Elastic
- Allocation of X Cores on Y Nodes - Calculate predefined task - Finish Jobs after calculations are done		- Allocation of available hardware - Jobs wait active for tasks - Finish Jobs wait for new tasks
- hardly compatible with modern principles		- flexible and follows modern principles
- Easy to implement		- Harder to implement

- Abstraction & Synchronization Layer -

How to handle fixed parallelization?
→ nothing special, provide steering information from the start

How to handle elastic parallelization?
→ Initialize instance as Controller or Responder with minimal steering data
→ Rebuild everything for each new set of control parameters
→ optimization possibilities for later if really required

```
while(sync.keepAlive())
    SControl.get_data(sync)
    corsika8.main()
```