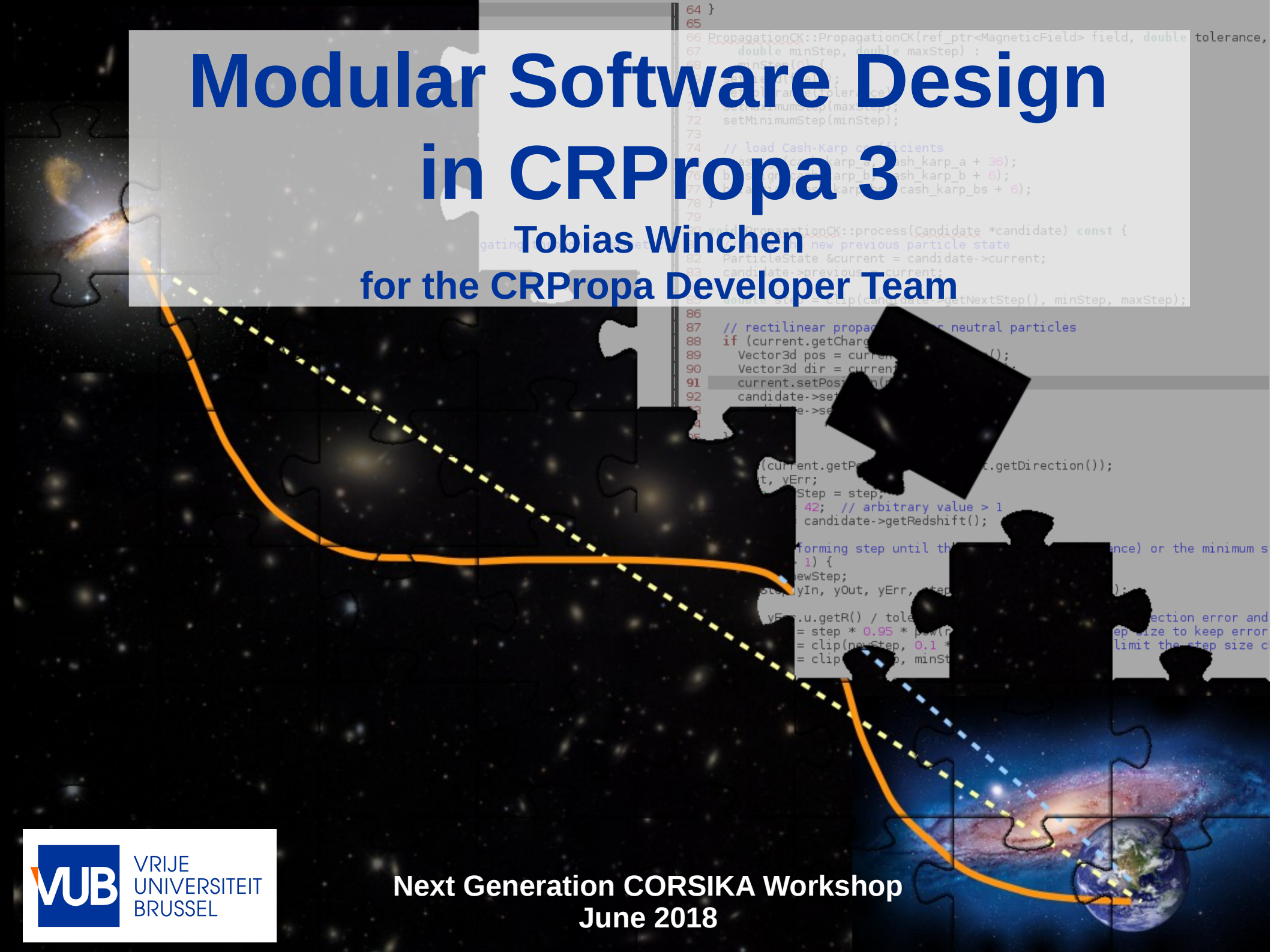
Modular Software Design in CRPropa 3

Tobias Winchen
for the CRPropa Developer Team



VRIJE UNIVERSITEIT BRUSSEL

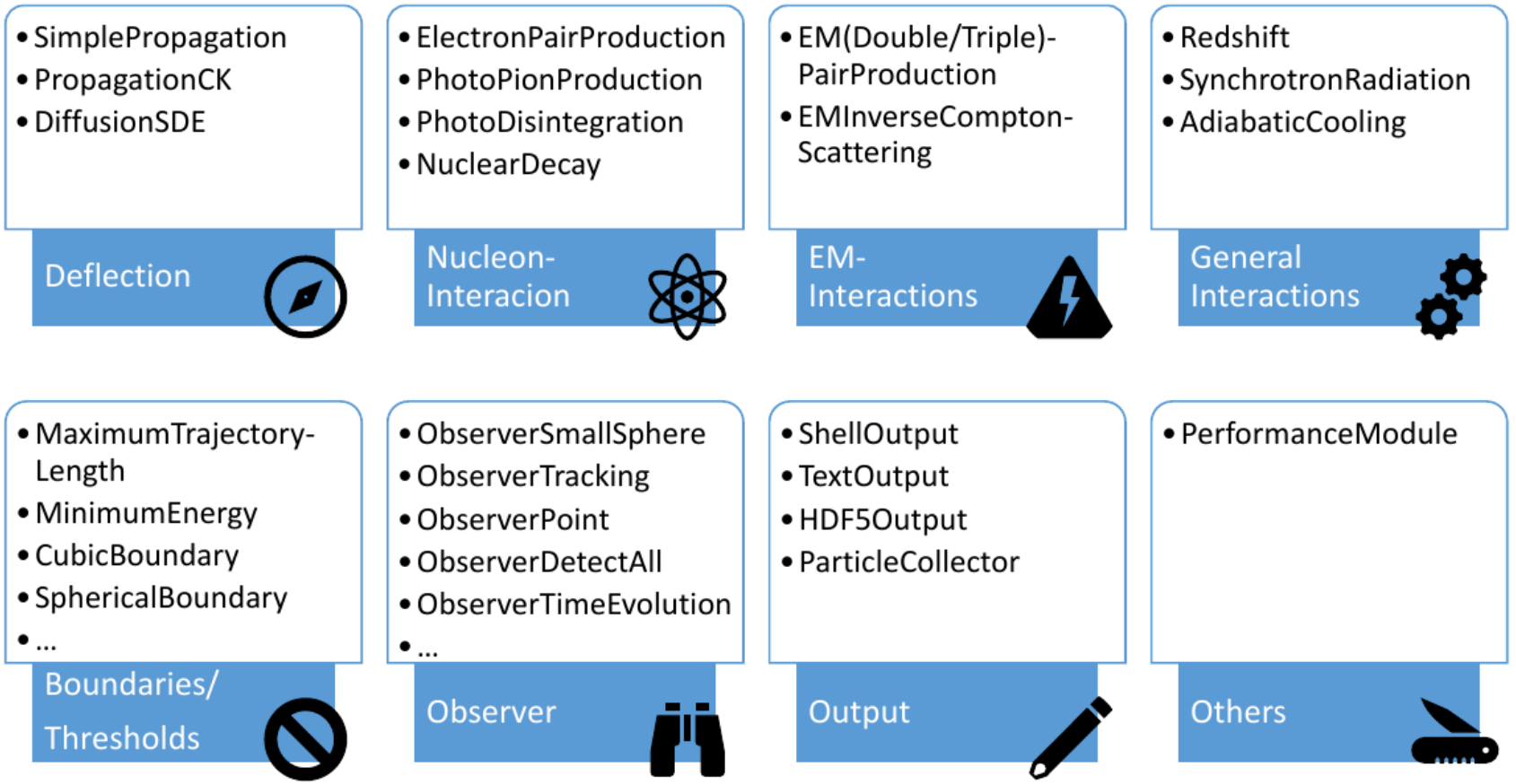
Next Generation CORSIKA Workshop
June 2018

Simulation Framework CR/Propa

Rafael Alves Batista[a,b], Julia Becker Tjus[c], Andrej Dundovic[a], Martin Erdmann[d], Christopher Heiter[d], Karl-Heinz Kampert[e], Daniel Kuempel[d], Lukas Merten[c], Gero Müller[d], Günter Sigl[a], Arjen van Vliet[a,f], David Walz[d], Tobias Winchen[d,e,g], Marcus Wirtz[d]

RWTH Aachen University[d], Ruhr Universität Bochum[c], Vrije Universiteit Brussels[g], University Hamburg[a], Radboud University Nijmegen[f], University of Sao Paolo[b], Bergische Universität Wuppertal[e]

Toolbox for Simulations of UHECR Propagation



Sketch by Lukas Merten

Download:
http://crpropa.desy.de

Online demo in your Browser:
http://vispa.physik.rwth-aachen.de

Example: Steering / Simulation Setup Code

Example in Python, but C++ would also be possible

```
1 from crpropa import *
2 sim = ModuleList()
3
```

1. Import + define empty simulation

```
4 sim.add( SimplePropagation(1*kpc, 10*Mpc) )
5
6 sim.add( Redshift() )
7 sim.add( PhotoPionProduction(CMB) )
8 sim.add( PhotoPionProduction(IRB) )
9 sim.add( PhotoDisintegration(CMB) )
    … more interactions ...
17 obs = Observer()
18 obs.add( ObserverPoint() )
19 output = TextOutput('events.txt', Output.Event1D)
20 obs.onDetection( output )
21 sim.add( obs )
22
```

2. Add modules

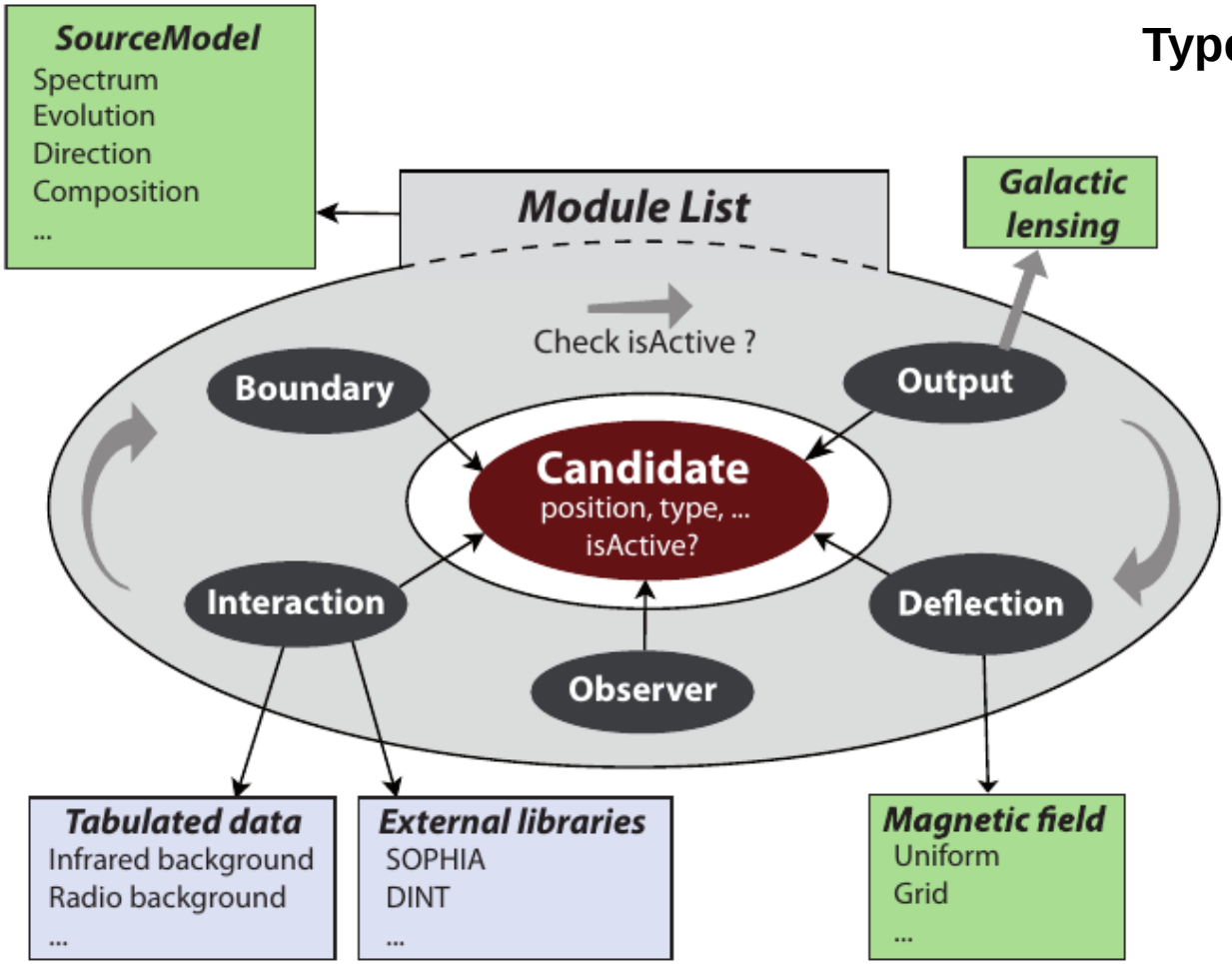
```
23 source = Source()
24 source.add( SourceUniform1D(1 * Mpc, 1000 * Mpc) )
25 source.add( SourceRedshift1D() )
26
27 composition = SourceComposition(1 * EeV, 100 * EeV, -1)
28 composition.add(1,  1,  1)  # H
29 composition.add(4,  2,  1)  # He-4
30 composition.add(14, 7,  1)  # N-14
31 composition.add(56, 26, 1)  # Fe-56
32 source.add( composition )
33
```

3. Define sources

```
34 sim.setShowProgress(True)
35 sim.run(source, 200, True)
36
```

4. Execute modules on output of sources

Overview Object Oriented Design



Types of Objects:

Candidate:

Data storage for state of single cosmic ray particle state (Energy, position, id, ..)

Sources:

Creates new candidates

Modules:

Change state of candidate (Move, interact, output, …)

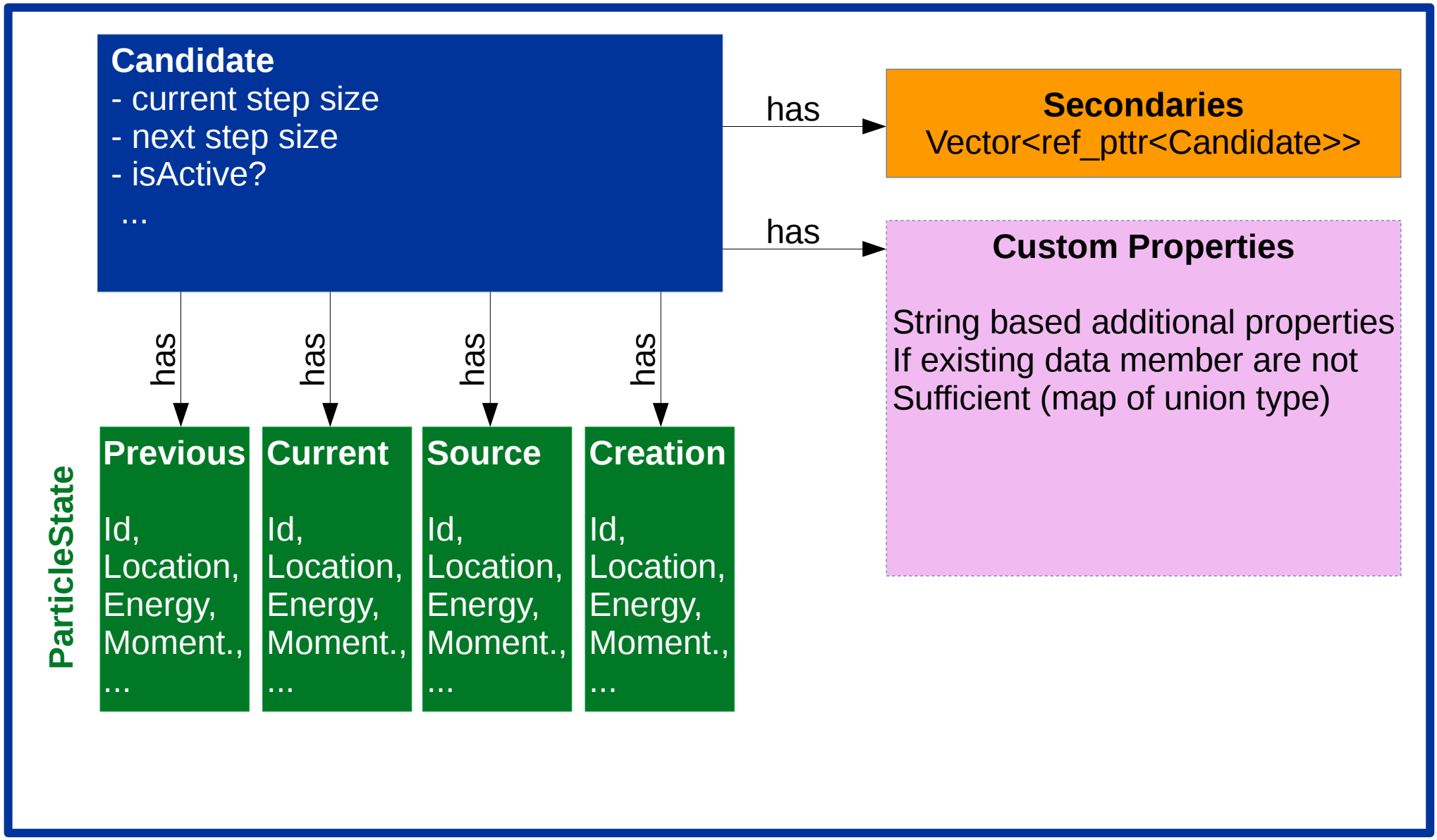
Auxiliary:

Magnetic field, photon field, ...

Sources, Modules and Auxiliary class are independent and stateless to enable parallel processing

New objects can be coded in C++ or Python (Interface generated using SWIG) by the user Without recompiling CRPropa core.

Candidate: Stores Data on Particle



Modules: Modify Candidate

Prototype Propagator

```
1. Save current state:
PreviousState = CurrentState

2. Make step
    - CurrentStep=NextStep
    - Update position
      according to CurrentStep

3.  Set NextStep to maximum

    exit module
```

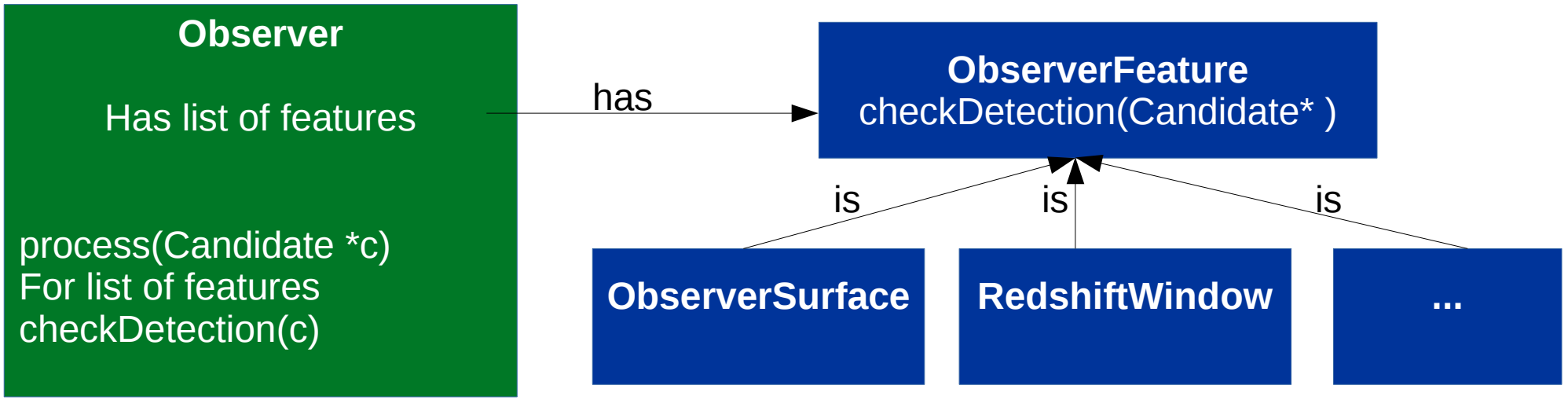
Prototype Interaction

```
1.If not applicable:
    Do nothing

2.Calculate  probability
  to interact in current
  step according to
  current candidate state

3.If no interaction:
    - Limit next step to small
      fraction of m. free path
    - exit module

  If interaction:
    - Modify current particle
    - add secondaries
    - Repeat from 2
```

- Candidate has individual step size
- Every module limits the stepsize to a **small** fraction of its mean free path
  - Order does not matter
  - Modules are independent, no communication required
- Modularity in interactions is paid for by random numbers as need to be generated in every cycle by every module (CRPropa not limited by RNG)

Module Features are Separate Objects

E.g. Observer



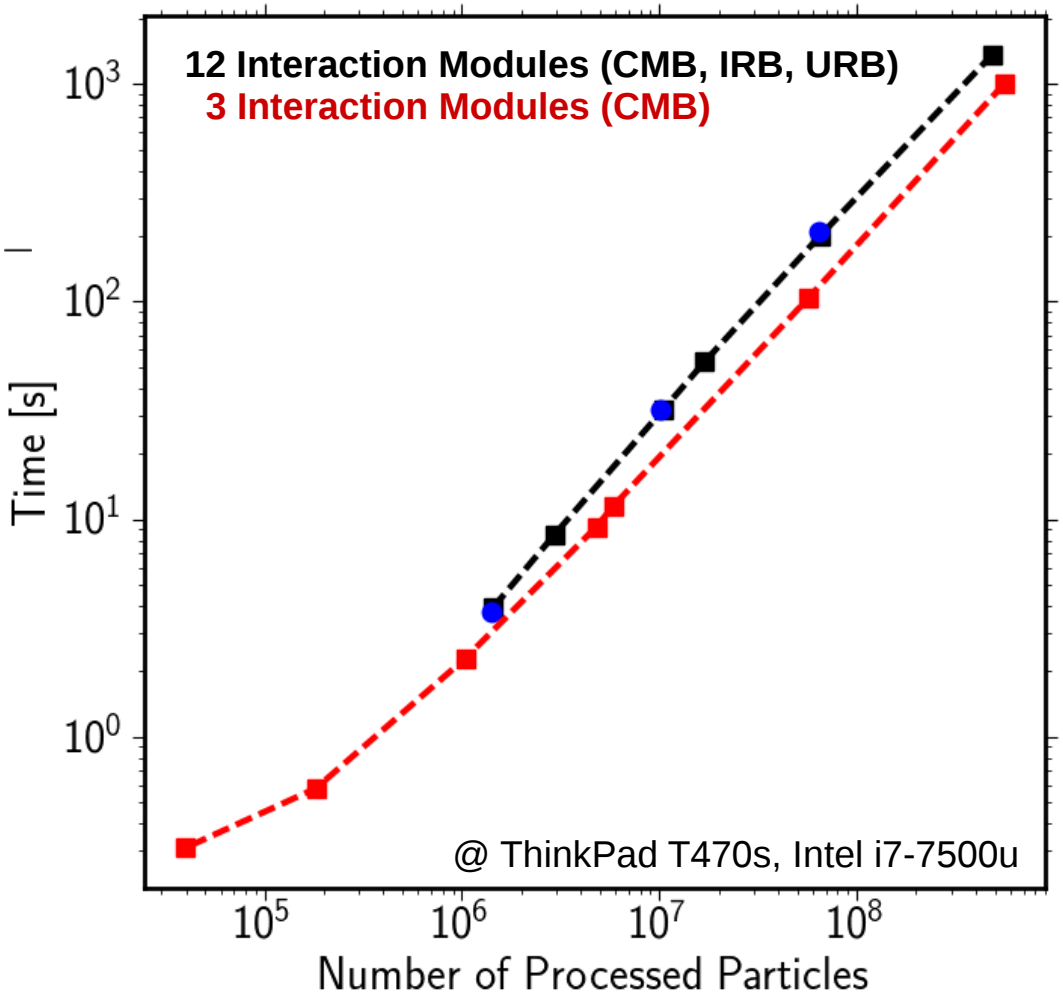
Example: Output particles crossing different planes

```
2 obs = Observer()
3 obs.add( ObserverSurface( Plane( Vector3d(0, 0, 0), Vector3(0,0,1) )))
4 obs.add( ObserverSurface( Plane( Vector3d(0, 0, 100 * meter), Vector3(0,0,1) )))
5 obs.add( ObserverSurface( Plane( Vector3d(0, 0, 200 * meter), Vector3(0,0,1) )))
6 obs.onDetection( TextOutput('output1.txt') )
7
```

Does this Scale for Many Particles?

- EM Cascades of secondary photon and electrons (Heiter et al. 2018)
  - → Pair production
    $\gamma + \gamma_{\mathrm{bg}} \longrightarrow e^+ + e^-$
  - → Double pair production
    $\gamma + \gamma_{\mathrm{bg}} \longrightarrow e^+ + e^- + e^+ + e^-$
  - → Triplet pair production
    $e + \gamma_{\mathrm{bg}} \longrightarrow e + e^+ + e^-$
  - → Inverse Compton scattering
    $e + \gamma_{\mathrm{bg}} \longrightarrow e + \gamma$

  3 background (CMB, IRB, URB) → 12 stochastic interaction modules
- Continuous energy loss due to redshift
- Injected 1000 EeV photons in 4 Mpc distance
- Cascaded to z=0 or variable minimum energy



Implementation scales linearly with number of processed particles

How did we get There?

- Several versions of CRPropa
- Previous modular codes
  - → PAX, PXL, RDAS

Software development philosophy:

- KISS: Keep it simple, not stupid
- YAGNI: You ain't gonna need it
- Refactor often and early
- Dev. Substeps:
  - → Make it work
  - → Make it right
  - → Make it nice
  - → *Make it fast (if needed)*

- Code review (of substantial changes)

  git + pull requests (github)
- Unit Tests + Continuous Integration

  gtest + travis
- Minimize dependencies:
  - → User should not need to compile dependencies
  - → Dependencies should be standard / trivial available on supported platforms (Linux + Mac)
    - » Cmake
    - » Python
    - » Swig
    - » hdf5

    or shipped in static version.
    - » Healpix subset, eigen, kiss, (tinyxml, thread), hepid
  - → Boost (and ROOT) are known to cause problems

Conclusion

VRIJE UNIVERSITEIT BRUSSEL

- CRPropa3 highly modular code design
  - → User friendly (all dependencies except python + swig) shipped + compiles using standardized tool chain (cmake)
  - → Highly modular
  - → Easily extendable (C++/python modules and features without recompiling)
  - → Scales linearly to high particle numbers
  - → Webpage / Code / Issue-tracker / Examples / Documentation / ...

    https://crpropa.desy.de/
    https://github.com/CRPropa/CRPropa3
- Several approaches probably transferable to Next Generation CORSIKA

Contact: tobias.winchen@rwth-aachen.de