

Modular Software Design in CRPropa 3

Tobias Winchen
for the CRPropa Developer Team

```
64 }
65
66 PropagationCK::PropagationCK(ref_ptr<MagneticField> field, double tolerance,
67     double minStep, double maxStep) :
68     minStep(minStep),
69     maxStep(maxStep),
70     field(field), tolerance(tolerance) {}
71
72 setMinimumStep(maxStep);
73 setMinimumStep(minStep);
74
75 // load Cash-Karp coefficients
76 double cash_karp_a[5] = { 25, 17, 9, 7, 5 };
77 double cash_karp_b[5] = { 0, 0, 0, 0, 0 };
78 double cash_karp_bs[5] = { 14, 19, 22, 27, 31 };
79
80 void PropagationCK::process(Candidate *candidate) const {
81     // get new previous particle state
82     ParticleState &current = candidate->current;
83     CandidateState &previous = candidate->previous;
84     double newStep = clip(candidate->getNextStep(), minStep, maxStep);
85
86     // rectilinear propagation for neutral particles
87     if (current.getCharge() == 0) {
88         Vector3d pos = current.getPosition();
89         Vector3d dir = current.getDirection();
90         current.setPosition(pos + dir * newStep);
91         candidate->setPrevious(current);
92         candidate->setStep(newStep);
93     }
94
95     // for charged particles
96     if (current.getCharge() != 0) {
97         Vector3d pos = current.getPosition();
98         Vector3d dir = current.getDirection();
99         double yErr;
100         double newStep = newStep;
101         double step = 42; // arbitrary value > 1
102         candidate->getRedshift();
103
104         // forming step until the error (direction error or the minimum s
105         for (int i = 0; i < 5; i++) {
106             newStep = newStep * step;
107             step = step * 0.95;
108             yErr = yErr + step * 0.95 * pow(10, -10);
109             newStep = clip(newStep, 0.1 * step, step);
110             newStep = clip(newStep, minStep, maxStep);
111         }
112     }
113 }
```

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 Paulo^b, Bergische Universität Wuppertal^e

Toolbox for Simulations of UHECR Propagation

- SimplePropagation
- PropagationCK
- DiffusionSDE

Deflection



- ElectronPairProduction
- PhotoPionProduction
- PhotoDisintegration
- NuclearDecay

Nucleon-
Interacion



- EM(Double/Triple)-
PairProduction
- EMInverseCompton-
Scattering

EM-
Interactions



- Redshift
- SynchrotronRadiation
- AdiabaticCooling

General
Interactions



- MaximumTrajectory-
Length
- MinimumEnergy
- CubicBoundary
- SphericalBoundary
- ...

Boundaries/
Thresholds



- ObserverSmallSphere
- ObserverTracking
- ObserverPoint
- ObserverDetectAll
- ObserverTimeEvolution
- ...

Observer



- ShellOutput
- TextOutput
- HDF5Output
- ParticleCollector

Output



- PerformanceModule

Others



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.EventID)
20 obs.onDetection( output )
21 sim.add( obs )
22
```

2. Add modules

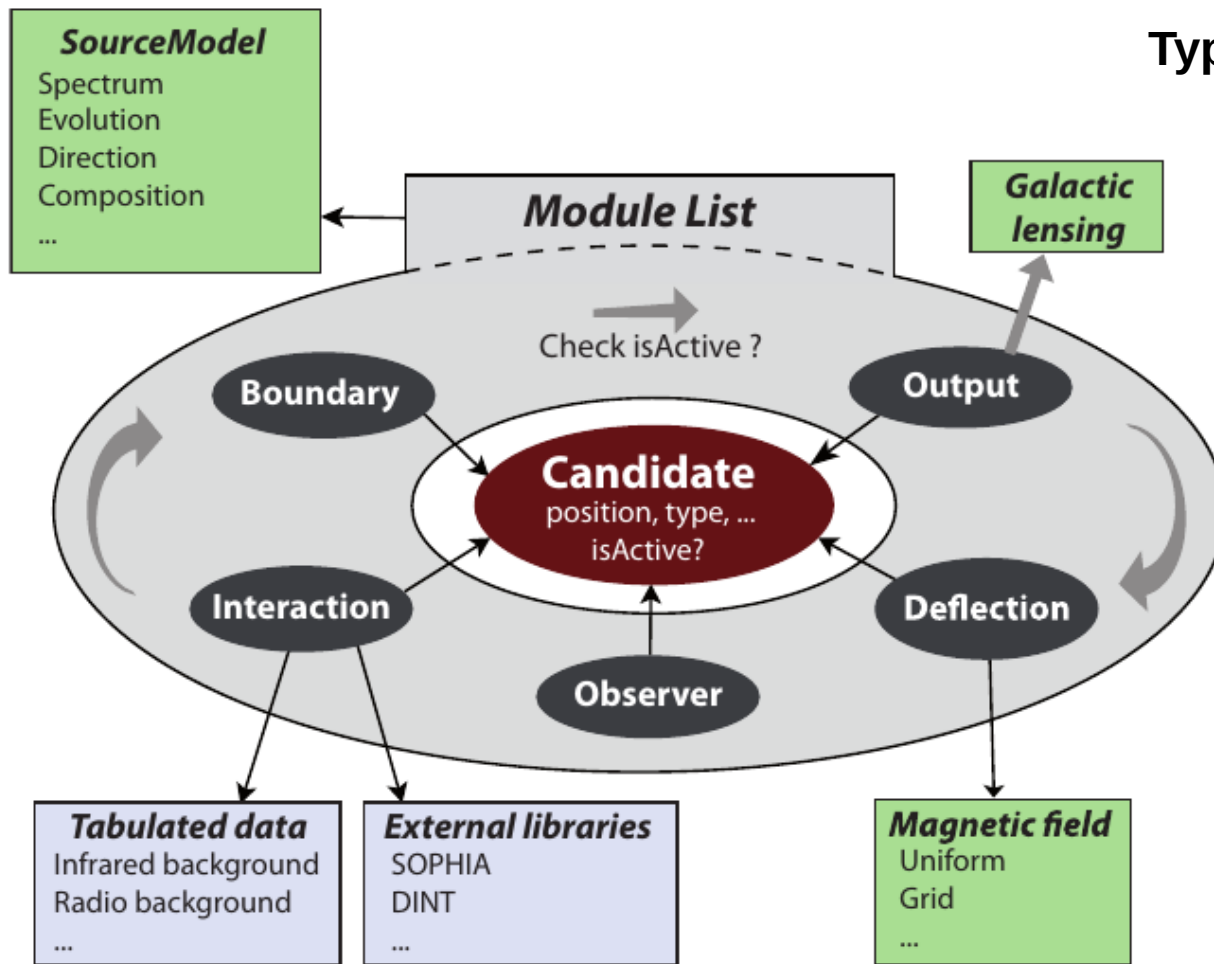
```
23 source = Source()
24 source.add( SourceUniform1D(1 * Mpc, 1000 * Mpc) )
25 source.add( SourceRedshift1D() )
26
27 composition = SourceComposition(1 * EeV, 100 * EeV, -1)
28 composition.add(1, 1, 1) # H
29 composition.add(4, 2, 1) # He-4
30 composition.add(14, 7, 1) # N-14
31 composition.add(56,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

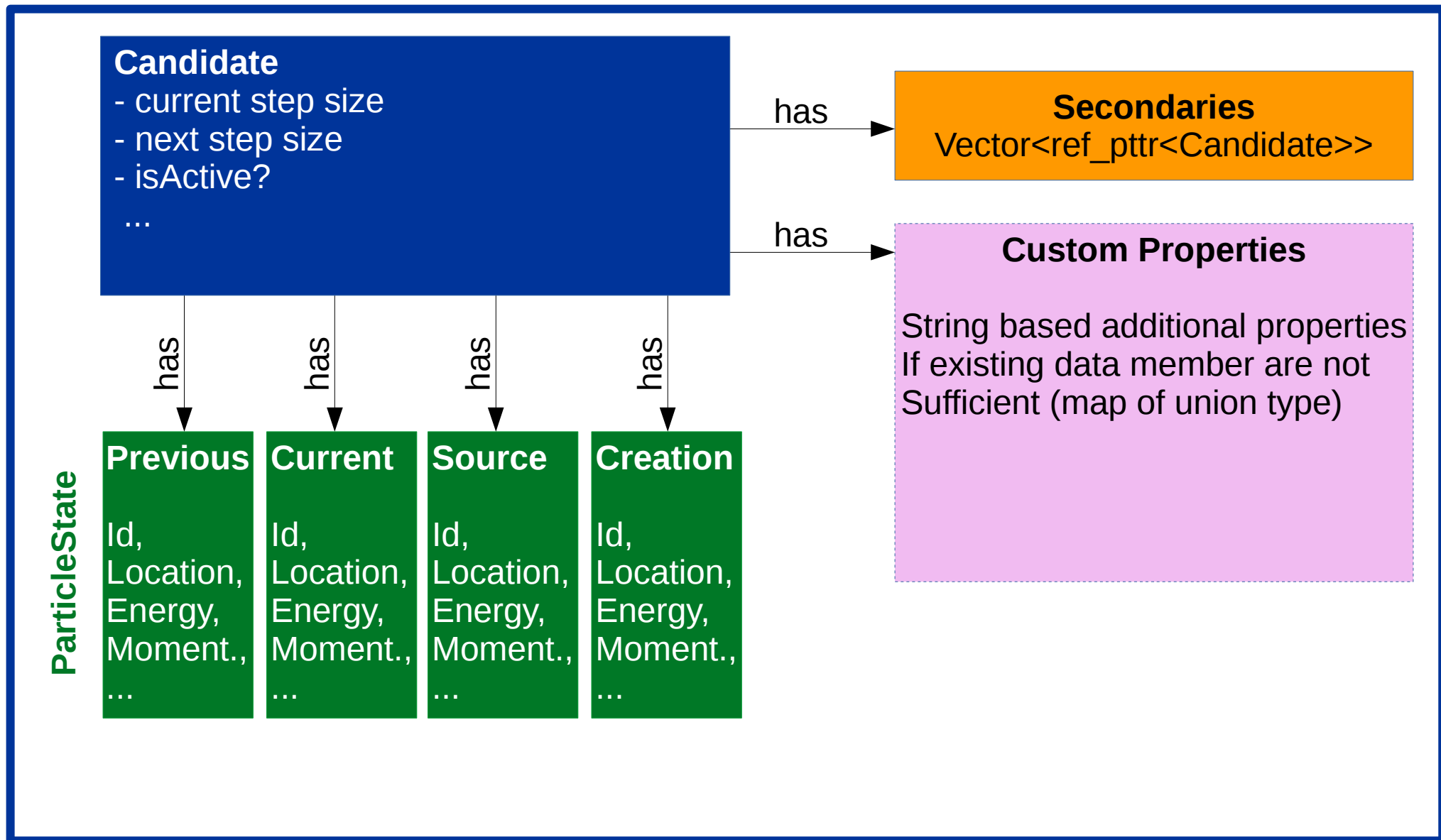
Change state of candidate (Move, interact, output, ...)

Auxiliary:

Magnetic field, photon field, ...

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

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

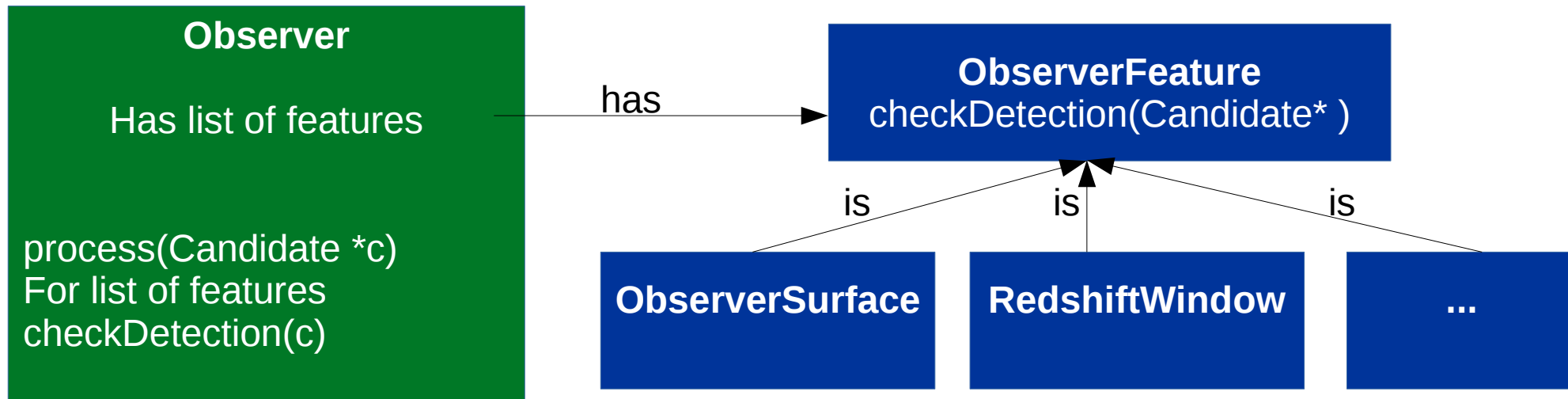
If interaction:

 - 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



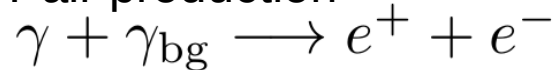
Example: Output particles crossing different planes

```
1
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
```

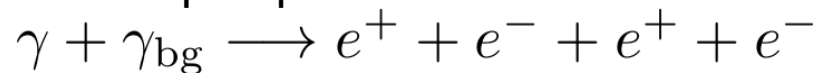
Does this Scale for Many Particles?

- EM Cascades of secondary photon and electrons (Heiter et al. 2018)

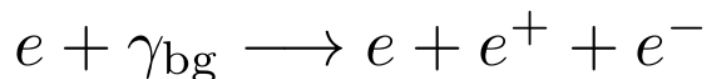
→ Pair production



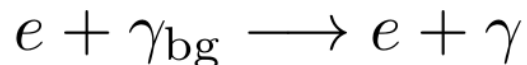
→ Double pair production



→ Triplet pair production

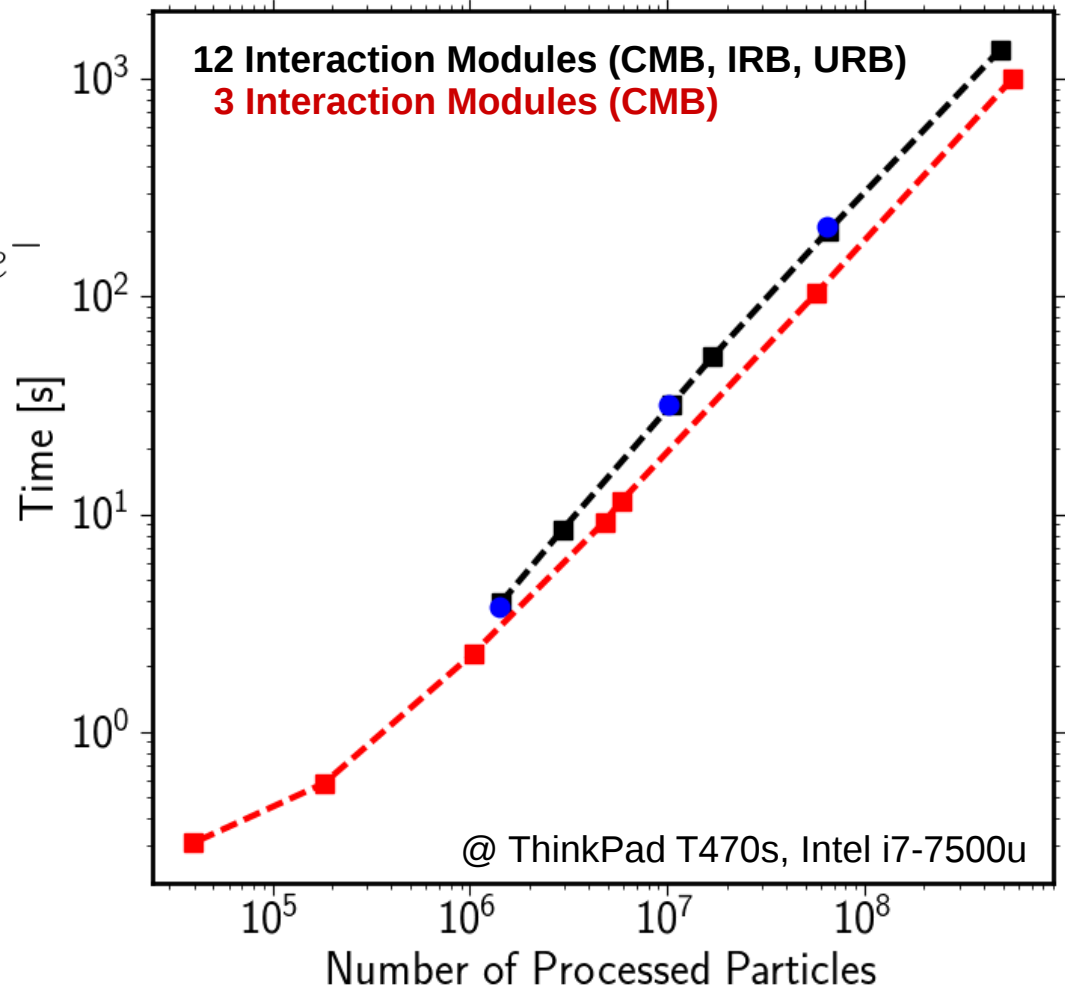


→ Inverse Compton scattering



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

■ 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