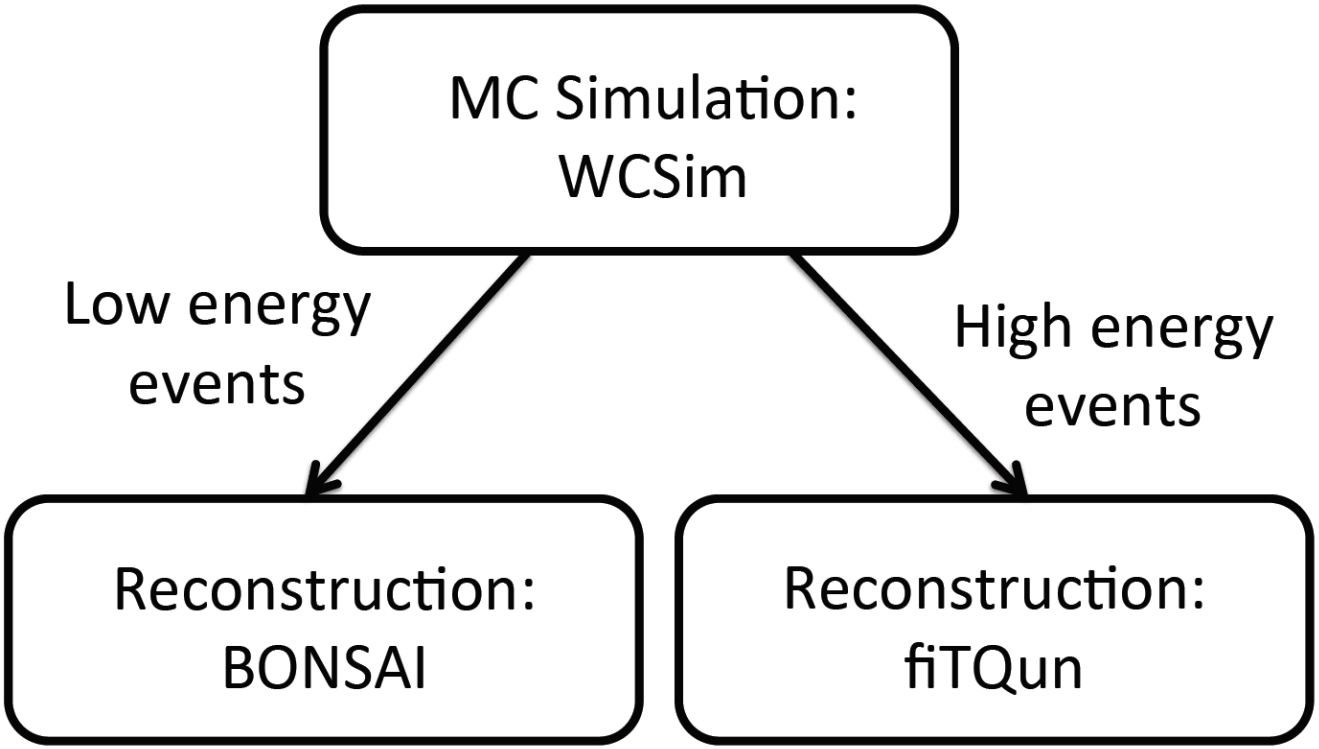
Hyper-K Software

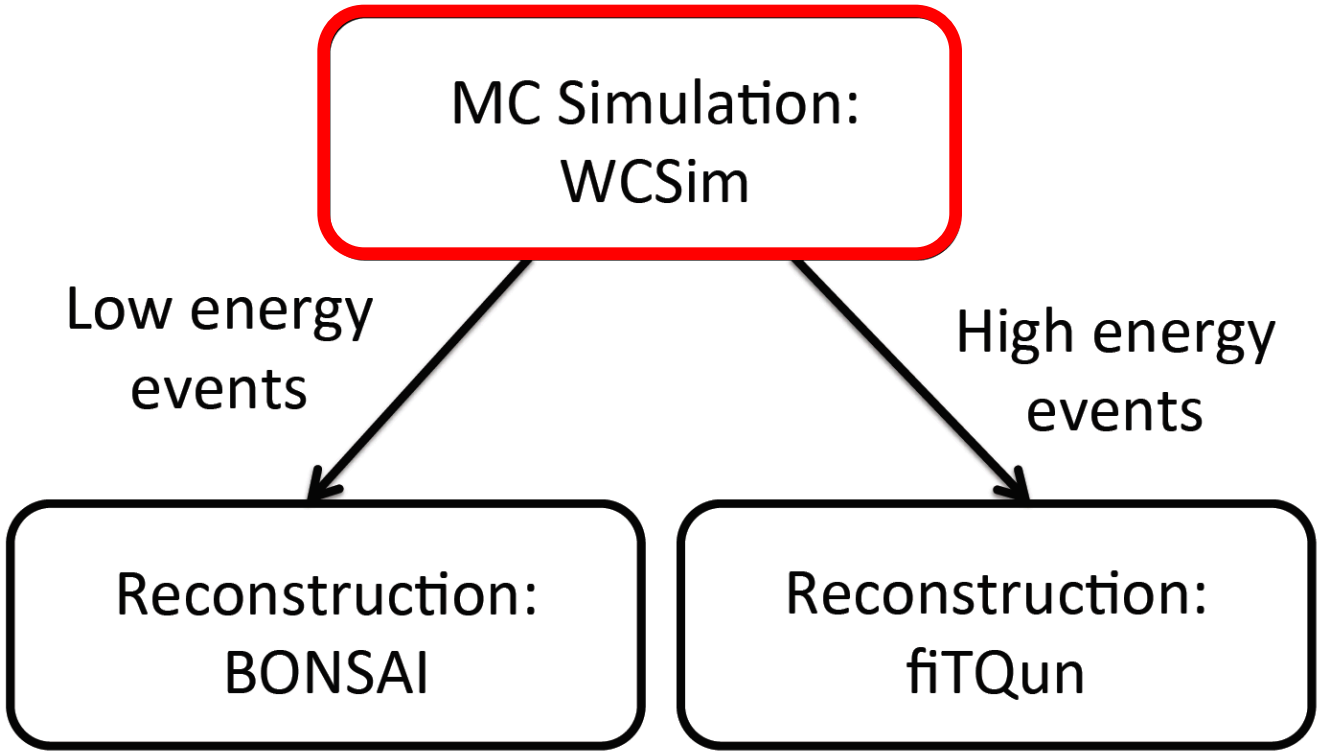
Erin O'Sullivan

Stockholm University

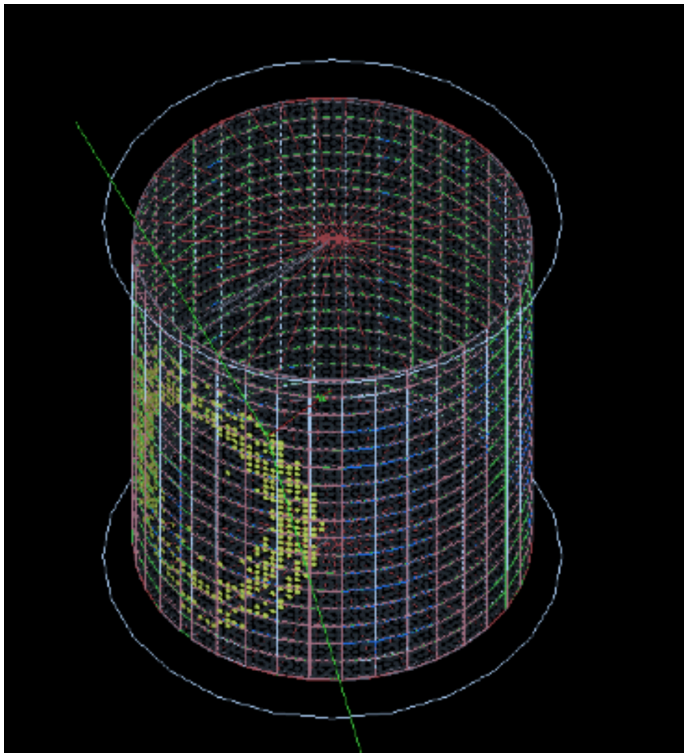
ESSnuSB software workshop

Current software tools


MC Simulation:
WCSim
Low energy
events
High energy
events
Reconstruction:
BONSAI
Reconstruction:
fiTQun


MC Simulation:
WCSim
Low energy
events
High energy
events
Reconstruction:
BONSAI
Reconstruction:
fiTQun

WCSim: A Water Cherenkov Simulation



An open-source, GEANT4-based simulation code for water Cherenkov detectors

https://github.com/WCSim/WCSim

WCSim / WCSim

Unwatch 25 | Star 11 | Fork 92

Code | Issues 49 | Pull requests 18 | Projects 0 | Wiki | Insights | Settings

The WCSim GEANT4 application

Edit

Manage topics

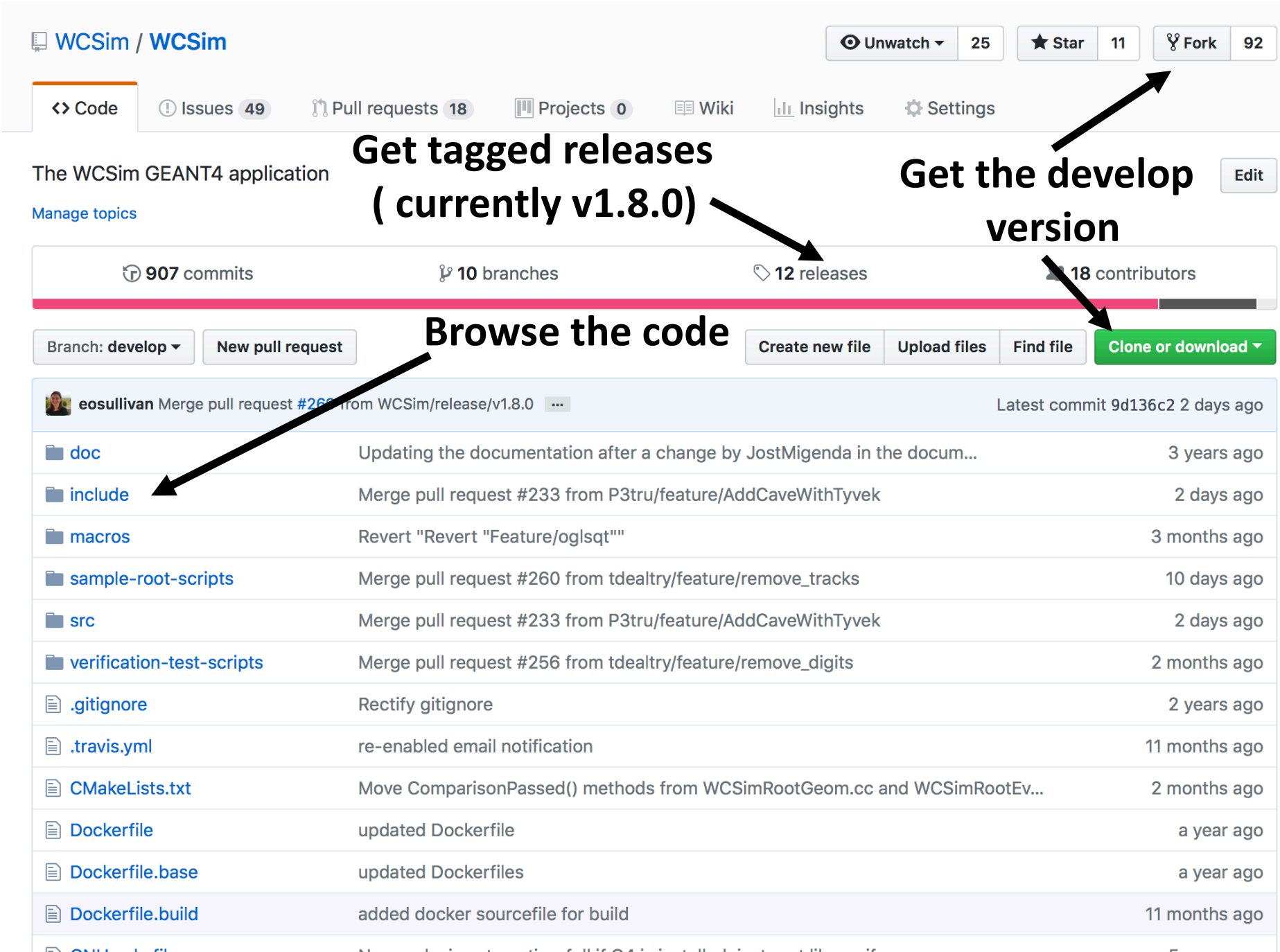
907 commits | 10 branches | 12 releases | 18 contributors

Branch: develop | New pull request | Create new file | Upload files | Find file | Clone or download

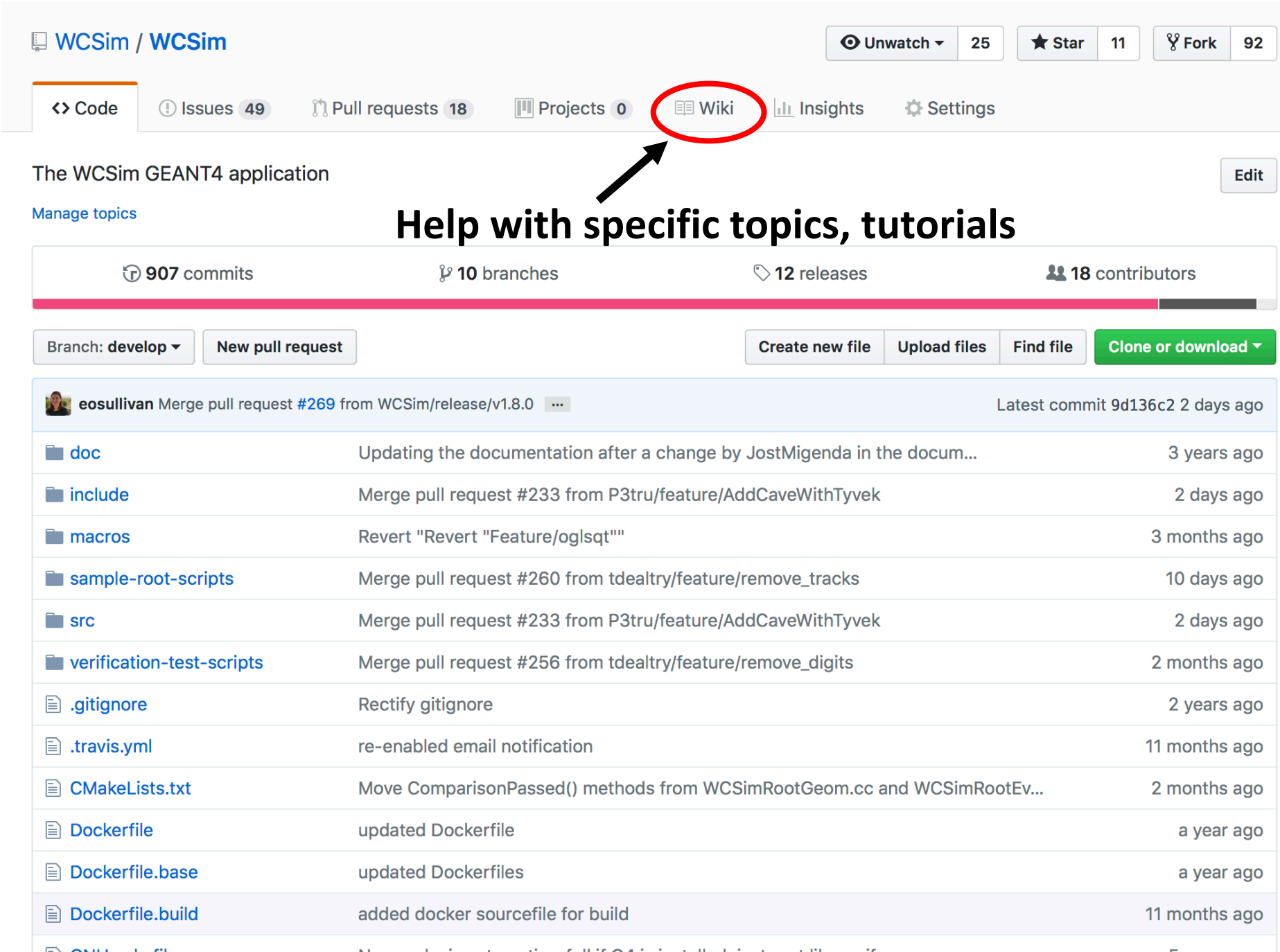
eosullivan Merge pull request #269 from WCSim/release/v1.8.0 ... Latest commit 9d136c2 2 days ago

doc	Updating the documentation after a change by JostMigenda in the docum...	3 years ago
include	Merge pull request #233 from P3tru/feature/AddCaveWithTyvek	2 days ago
macros	Revert "Revert "Feature/oglsqt""	3 months ago
sample-root-scripts	Merge pull request #260 from tdealtry/feature/remove_tracks	10 days ago
src	Merge pull request #233 from P3tru/feature/AddCaveWithTyvek	2 days ago
verification-test-scripts	Merge pull request #256 from tdealtry/feature/remove_digits	2 months ago
.gitignore	Rectify gitignore	2 years ago
.travis.yml	re-enabled email notification	11 months ago
CMakeLists.txt	Move ComparisonPassed() methods from WCSimRootGeom.cc and WCSimRootEv...	2 months ago
Dockerfile	updated Dockerfile	a year ago
Dockerfile.base	updated Dockerfiles	a year ago
Dockerfile.build	added docker sourcefile for build	11 months ago

https://github.com/WCSim/WCSim



https://github.com/WCSim/WCSim



Tutorial: making your own detector

```
void WCSimDetectorConstruction::SetTestGeometry()
{
WCSimPMTObject * PMT = CreatePMTObject("PMT20inch", "glassFaceWCPMT");
WCPMTName = PMT->GetPMTName();
WCPMTExposeHeight = PMT->GetExposeHeight();
WCPMTRadius = PMT->GetRadius();
WCPMTGlassThickness = PMT->GetPMTGlassThickness();
WCIDDiameter = 33.6815*m; //16.900*2*cos(2*pi*rad/75)*m; //inner detector diameter
WCIDHeight = 36.200*m; //"" "" height
WCBarrelPMTOffset = 0.0715*m; //offset from vertical
WCBarrelNumPMTHorizontal = 150;
WCBarrelNRings = 17.;
WCPMTperCellHorizontal= 4;
WCPMTperCellVertical = 3;
WCCapPMTSpacing = 0.707*m; // distance between centers of top and bottom pmts
WCCapEdgeLimit = 16.9*m;
WCBlackSheetThickness = 2.0*cm;
WCAddGd = false;
}
```

Choose the PMT type

Change the tank dimensions

Change the number of phototubes
(can alternatively specify photocoverage)

Tutorial: making your own photosensor

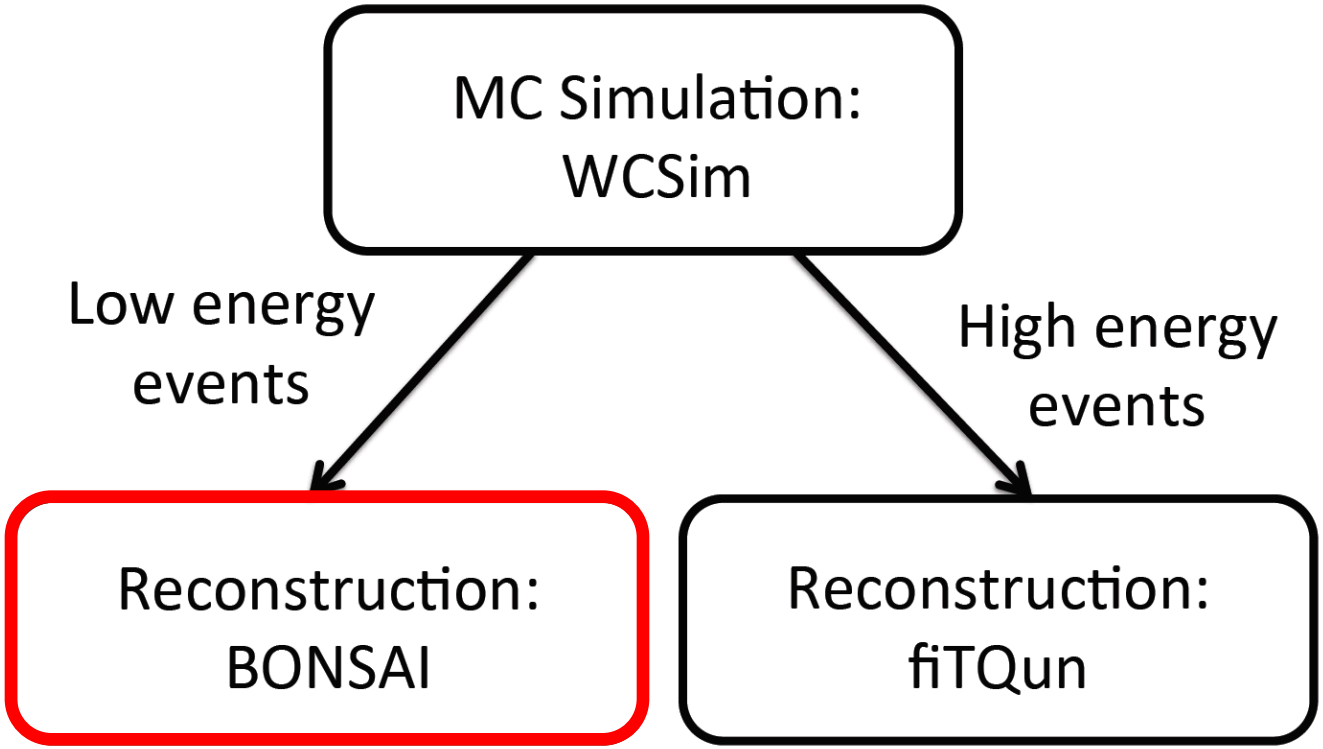
```
PMTTest::~PMTTest(){}
G4String PMTTest::GetPMTName() {G4String PMTName = "20inch"; return PMTName;}
G4double PMTTest::GetExposeHeight() {return .18*m;}
G4double PMTTest::GetRadius() {return .254*m;}
G4double PMTTest::GetPMTGlassThickness() {return 0.4*cm;}
float PMTTest::GettimingResolution(float Q) {
float timingConstant = 10.0;
float timingResolution = 0.33 + sqrt(timingConstant/Q);
// looking at SK's jitter function for 20" tubes
if (timingResolution < 0.58) timingResolution=0.58;
return timingResolution;
}
G4float* PMTTest::Getqpe()
{
static G4float qpe0[501]= {
// 1
0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
0.000000, 0.000000, 0.000000, 0.000000, 0.000000,
```

Specify photosensor geometry

Specify timing and charge properties

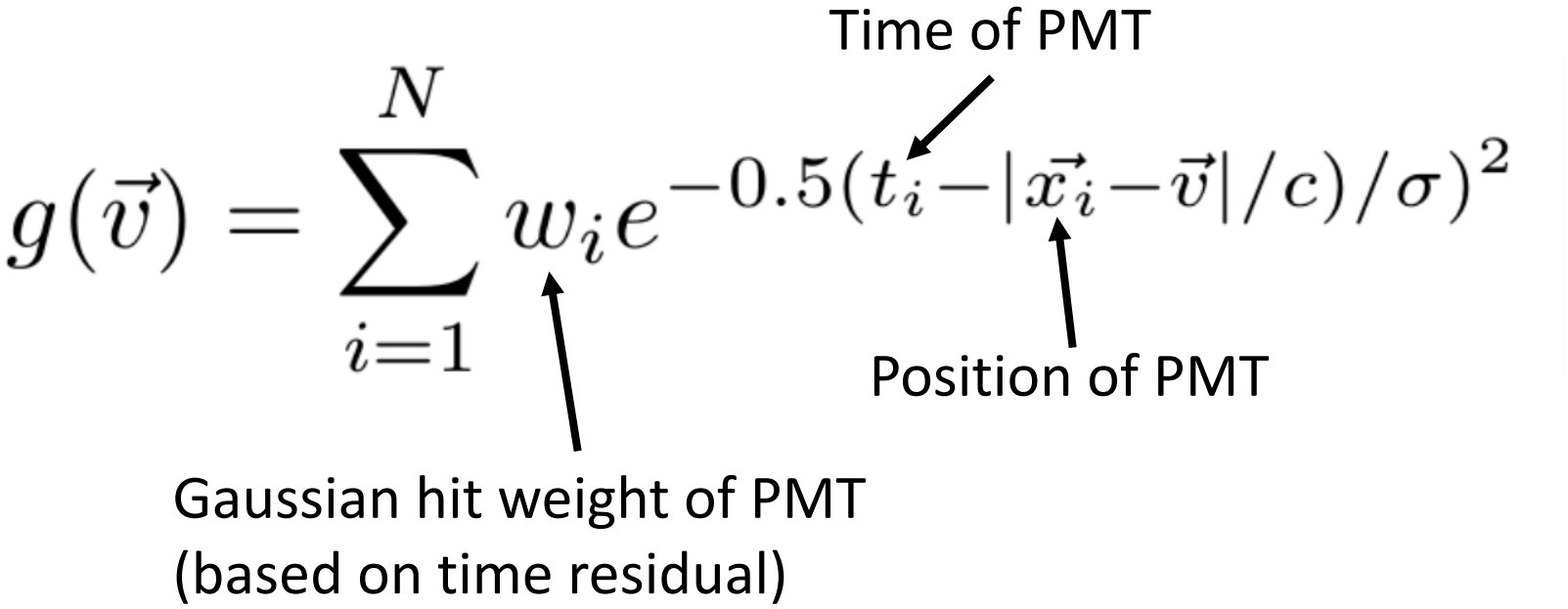
Specify quantum efficiency info

```
G4float* PMTTest::GetQEWavelength(){
static G4float wavelength_value[20] = { 280., 300., 320., 340., 360., 380., 400., 420.,
return wavelength_value;
}
G4float* PMTTest::GetQE(){
static G4float QE[20] = { 0.00, .0139, .0854, .169, .203, .206, .211, .202,.188, .167,
return QE;
}
G4float PMTTest::GetmaxQE(){
const G4float maxQE = 0.211;
return maxQE;
}
```


MC Simulation:
WCSim
Low energy
events
High energy
events
Reconstruction:
BONSAI
Reconstruction:
fiTQun

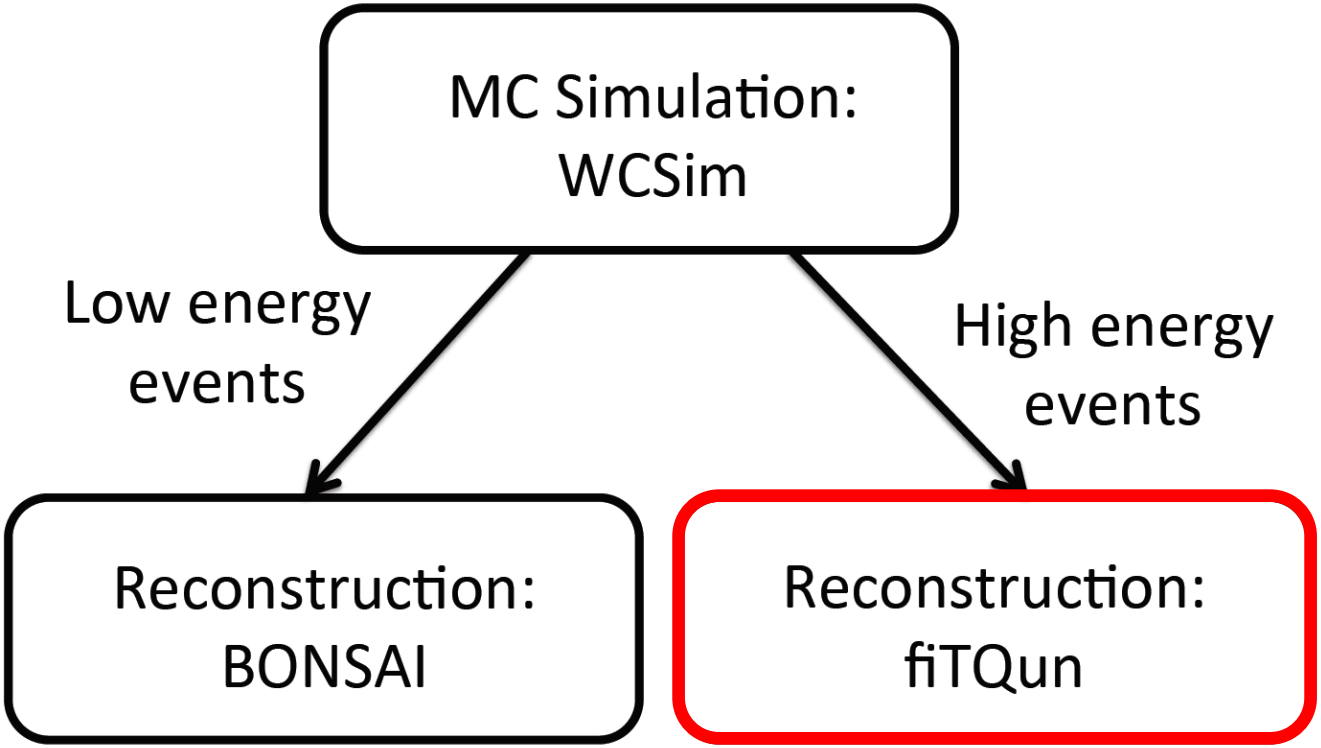
BONSAI: ~ few MeV - ~tens of MeV

Vertex Reconstruction



BONSAI: ~ few MeV - ~tens of MeV

- Directional reconstruction: Uses the fitted vertex and maximizes for direction
- Energy reconstruction: Mostly based on number of hit PMTs


MC Simulation:
WCSim
Low energy
events
High energy
events
Reconstruction:
BONSAI
Reconstruction:
fiTQun

fiTQun: High energy reconstruction

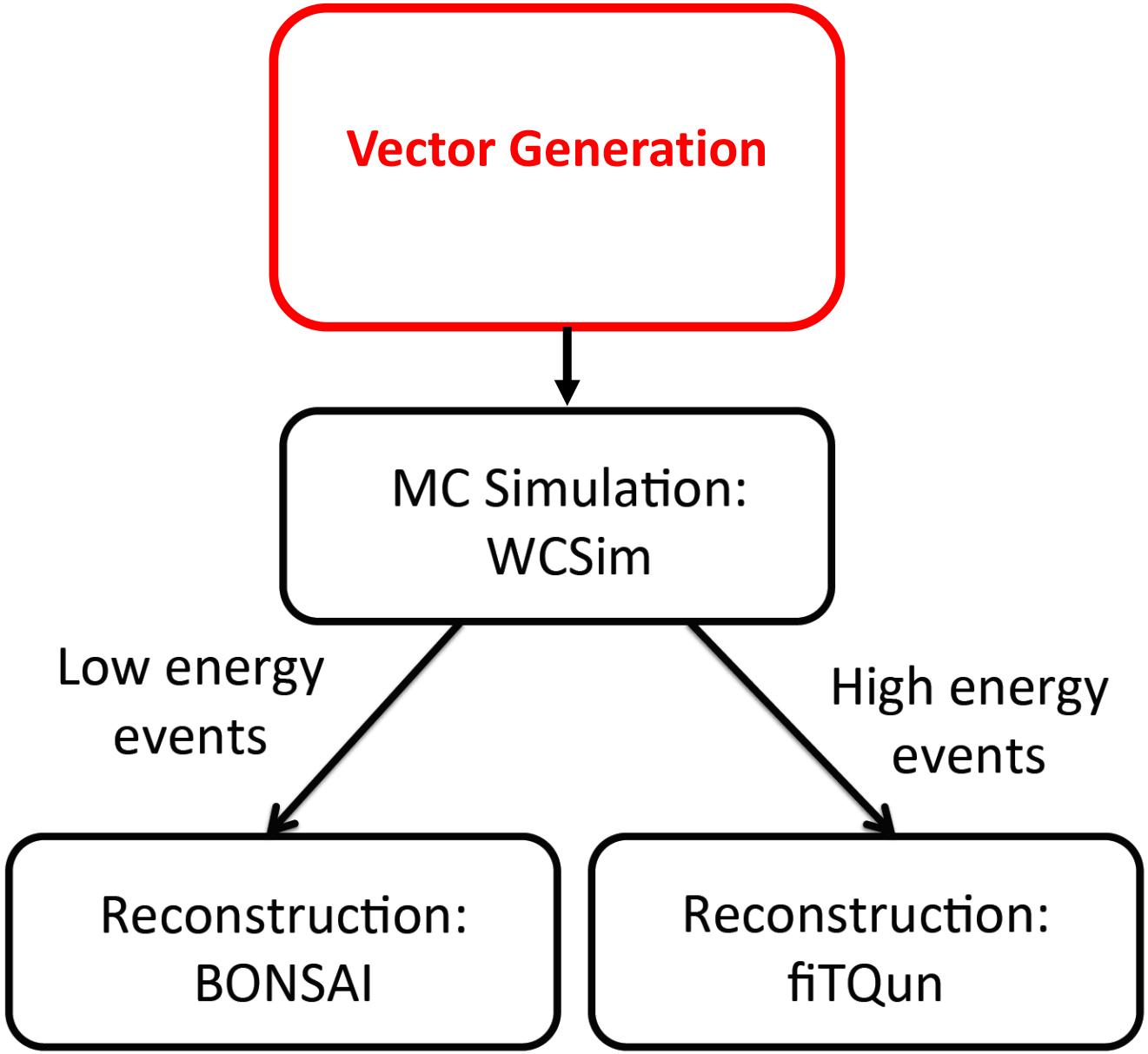
PMT unhit probability PMT hit probability PMT charge pdf

$$L(\mathbf{x}) = \prod_{j}^{\text{unhit}} P_j(\text{unhit}|\mu_j) \prod_{i}^{\text{hit}} \{1 - P_i(\text{unhit}|\mu_i)\} f_q(q_i|\mu_i) f_t(t_i|\mathbf{x})$$

PMT timing pdf

Requires “tuning” with MC for information specific to the photosensors (charge and time response functions, angular response function), as well as scattered and reflected light time PDFs

Ongoing and future software projects

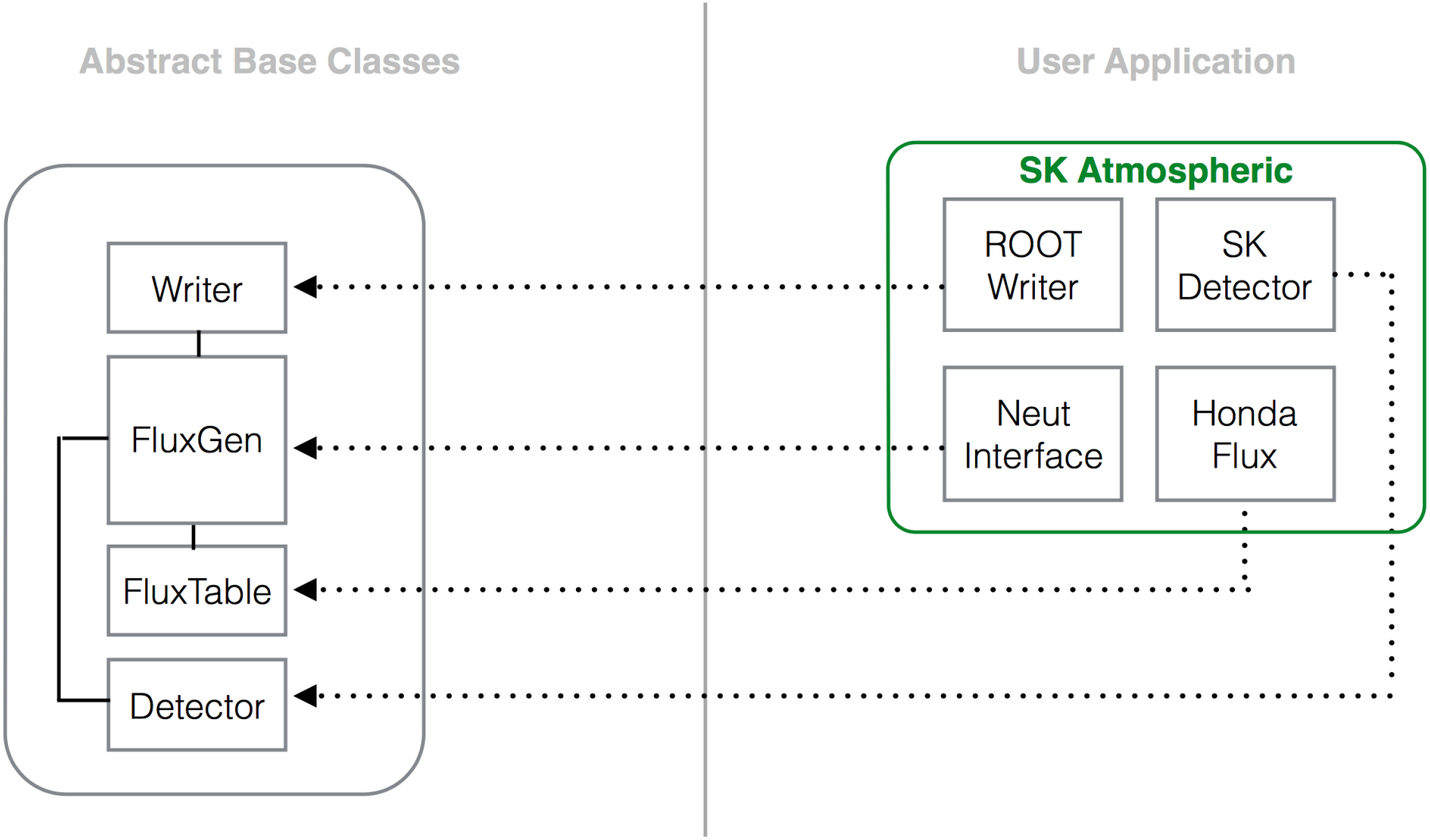

Vector Generation
MC Simulation:
WCSim
Low energy
events
High energy
events
Reconstruction:
BONSAI
Reconstruction:
fiTQun

Vector generation in WCSim

- Can run “particle guns” (ie. electons with a specific energy in a specific direction)
- Tools to make simple vector files (ie. electrons with a specific energy, uniform in some volume). See WCSim/sample-root-scripts/MakeKin.py.
- For more complication situations, need a dedicated vector generator that interfaces with an interaction generator (ie. NEUT, GENIE) and outputs the particle interaction information

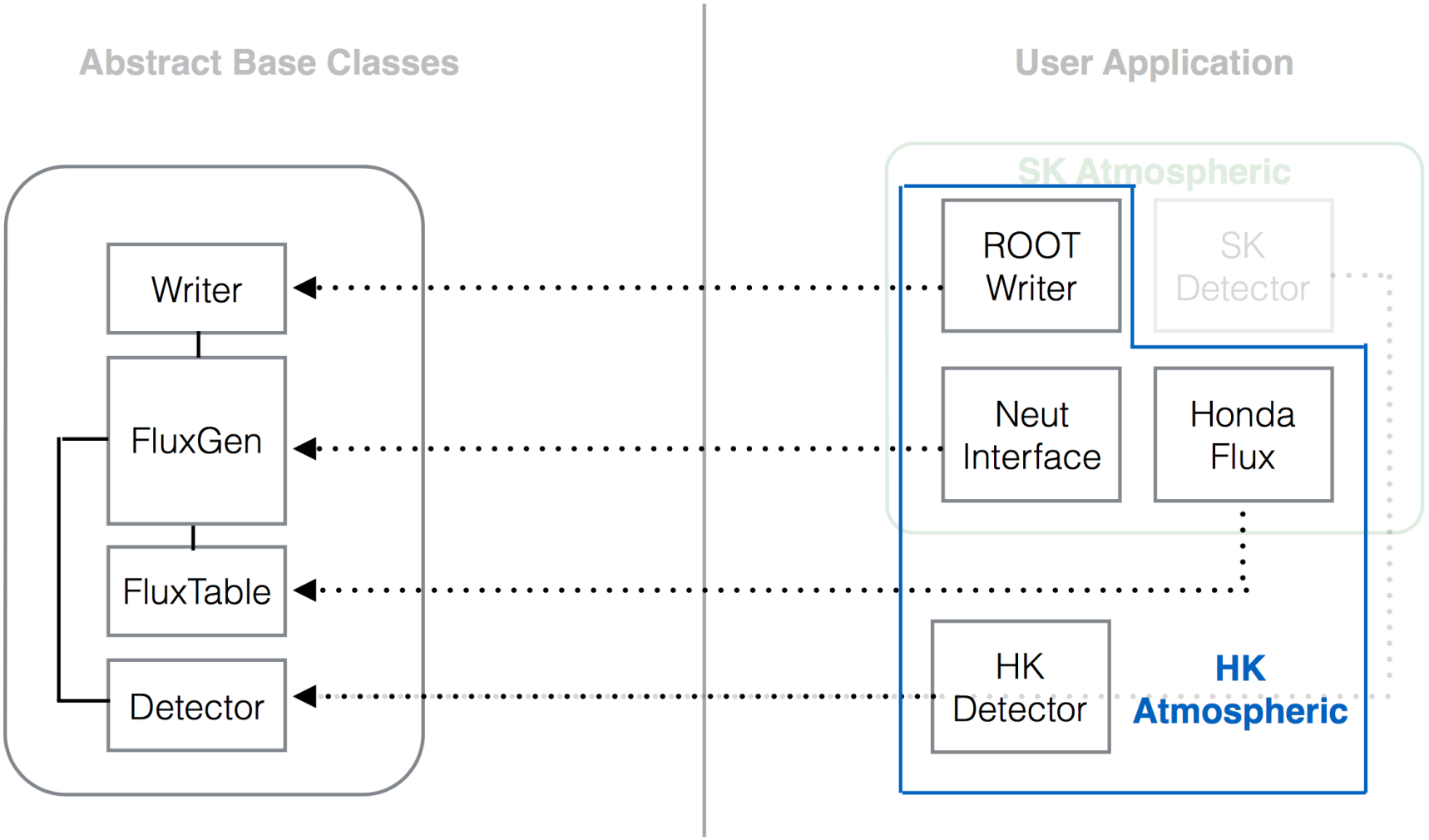
https://github.com/ckachulis/NGen

Vector generation:NGen



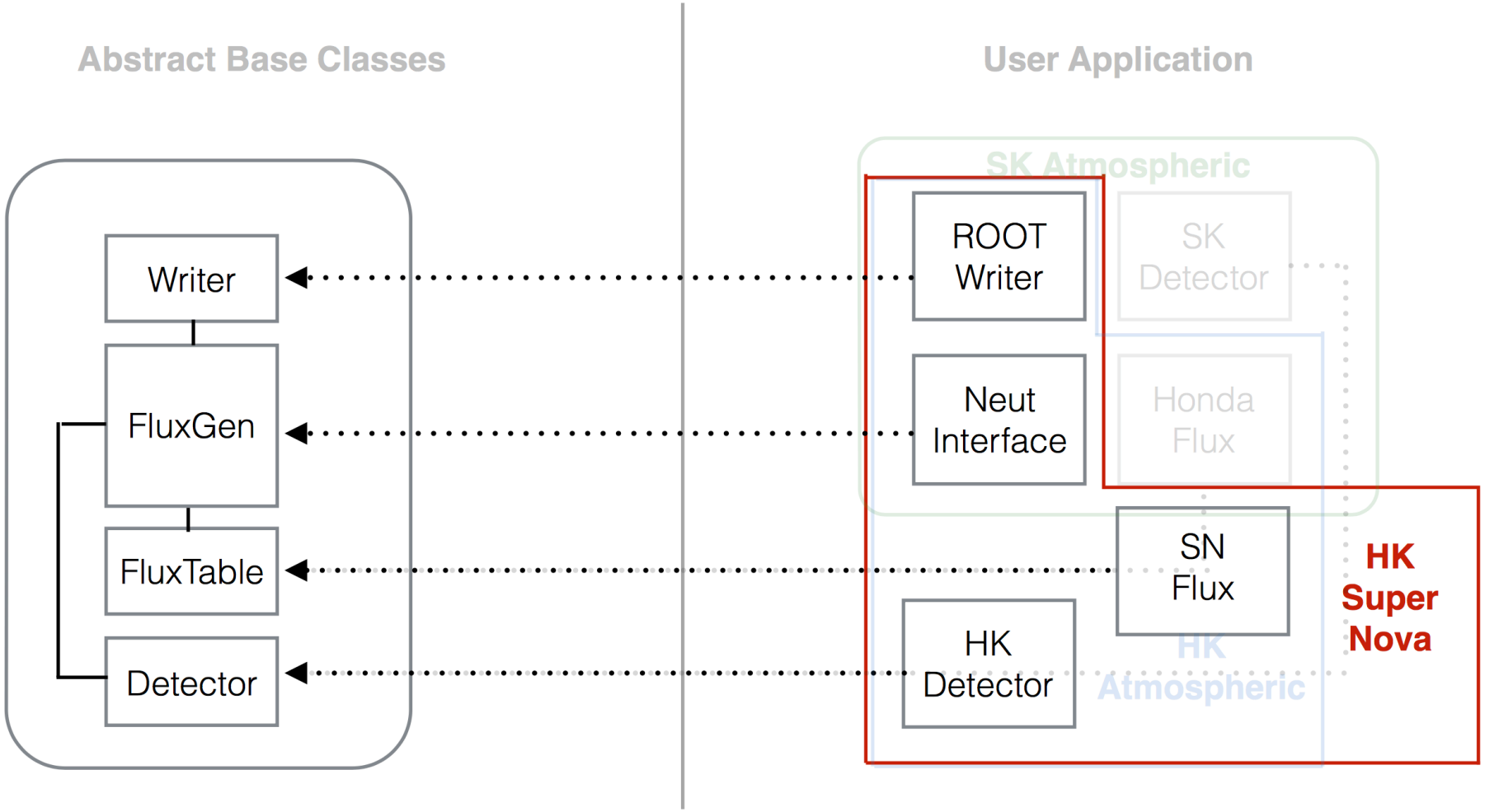
https://github.com/ckachulis/NGen

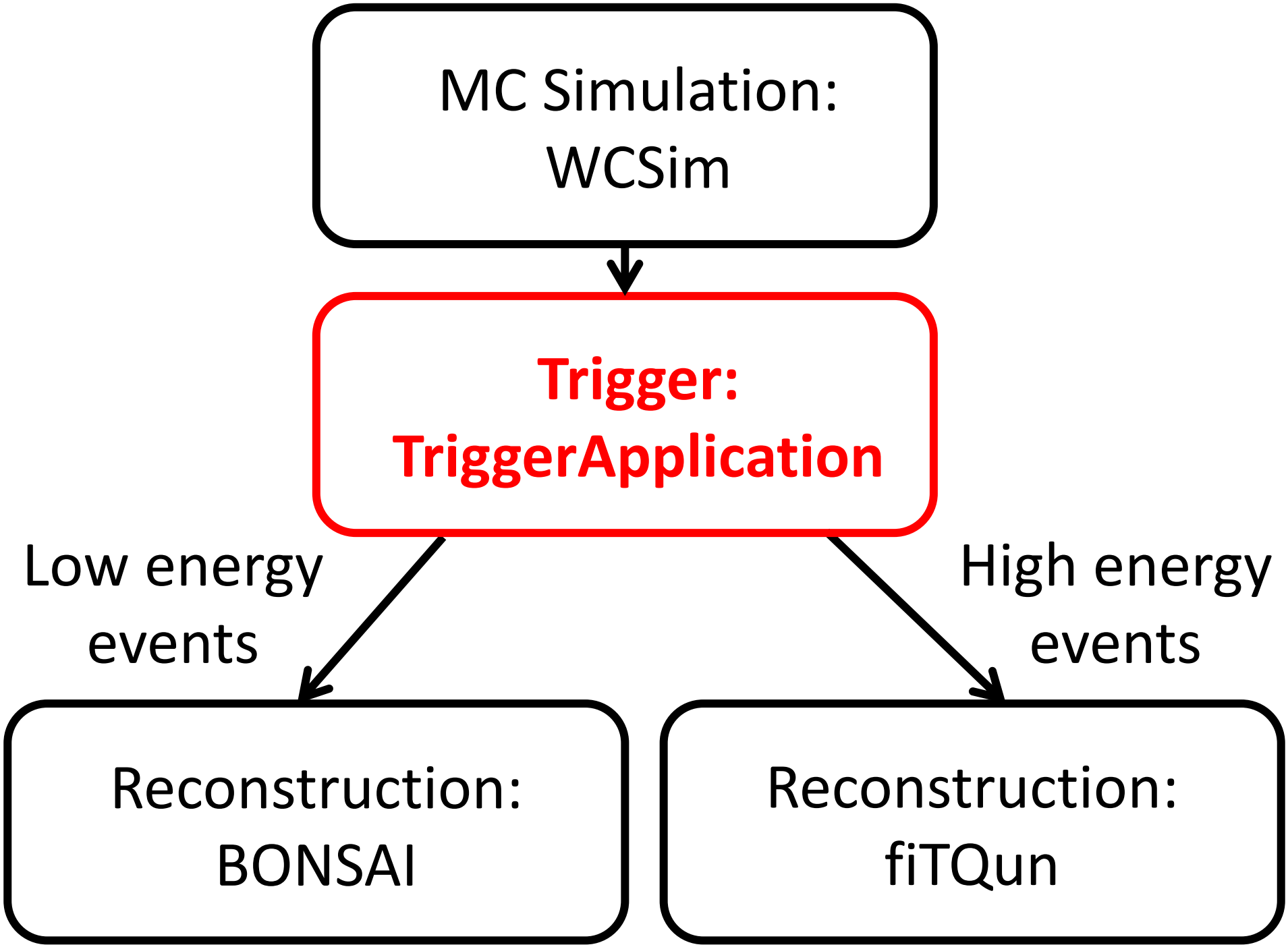
Vector generation:NGen



https://github.com/ckachulis/NGen

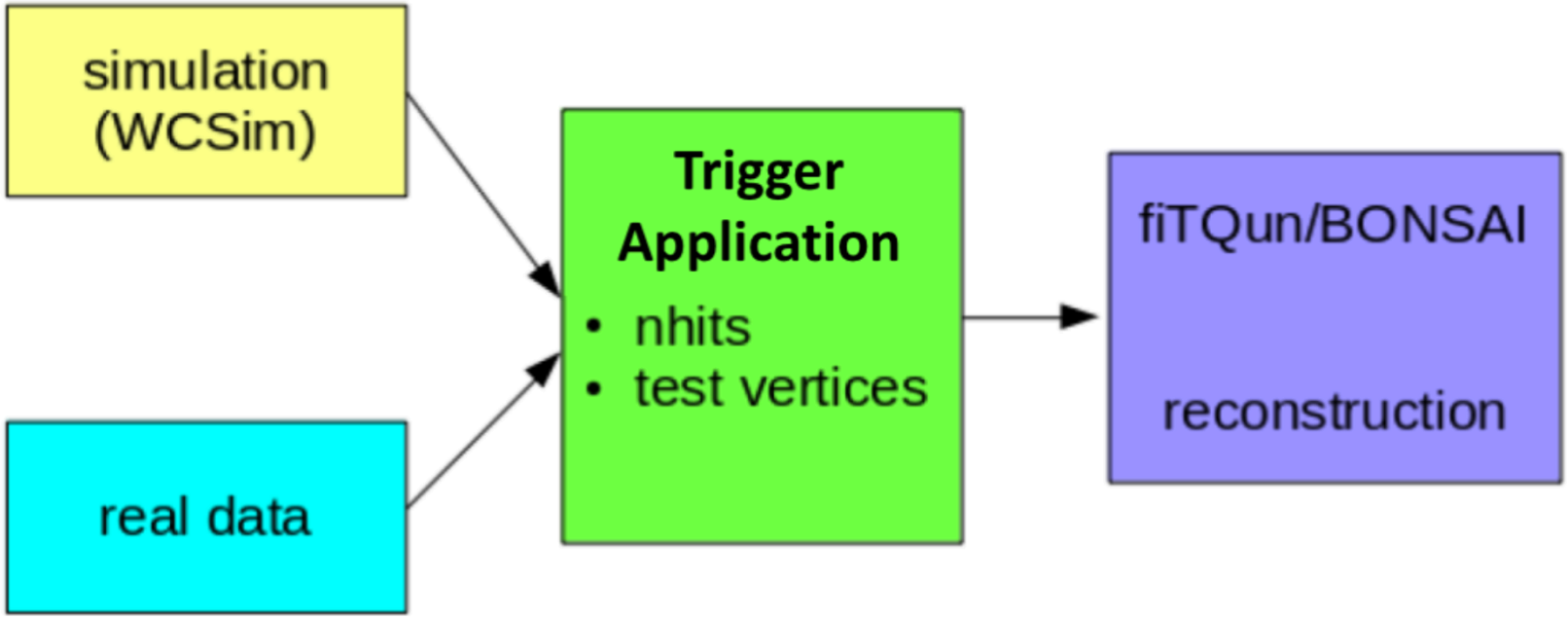
Vector generation:NGen




MC Simulation:
WCSim
Trigger:
TriggerApplication
Low energy
events
High energy
events
Reconstruction:
BONSAI
Reconstruction:
fiTQun

https://github.com/HKDAQ/TriggerApplication

Trigger: TriggerApplication



Same software trigger can be used for both data and Monte Carlo

https://github.com/HKDAQ/TriggerApplication

TriggerApplication

Trigger Application is designed as a modular trigger software to run successive triggers on WCSim data.

its built from ToolDAQ Application[1] which is an open source general DAQ Application template built using the modular ToolDAQ Framework core[2] to give separation between core and implementation code.

Concept

The main executable creates a ToolChain which is an object that holds Tools. Tools are added to the ToolChain and then the ToolChain can be told to Initialise Execute and Finalise each tool in the chain.

The ToolChain also holds a uesr defined DataModel which each tool has access too and can read, update and modify. This is the method by which data is passed between Tools.

User Tools can be generated for use in the tool chain by incuding a Tool header. This can be done manually or by use of the newTool.sh script.

For more information consult the ToolDAQ doc.pdf

https://github.com/ToolDAQ/ToolDAQFramework/blob/master/ToolDAQ%20doc.pdf

---

Tutorial

Note that there are `README.md` files in most folders. Check them out!

Installation with Docker

- WCSim and TriggerApplication now supports containerized installation using Docker.
- Docker allows anyone to install and use code quickly and easily no matter what their OS (Windows, MacOS, linux of any distro) without the headache of dependencies and trying to compile the code.
- Installation instructions can be found on the README file on the WCSim and TriggerApplication Github pages

Summary of tools

- MC Simulation: **WCSim** - https://github.com/WCSim/WCSim
- Reconstruction: **BONSAI** and **fiTQun** (not currently open-source, but I think a copy of fiTQun has been distributed to you?)
- Vector generation: **NGen** - https://github.com/ckachulis/NGen
- External triggering: **TriggerApplication** – https://github.com/HKDAQ/TriggerApplication