Framework and Data Flow

Bishoy H. Dongwi
CFNS Edward Bouchet Fellow

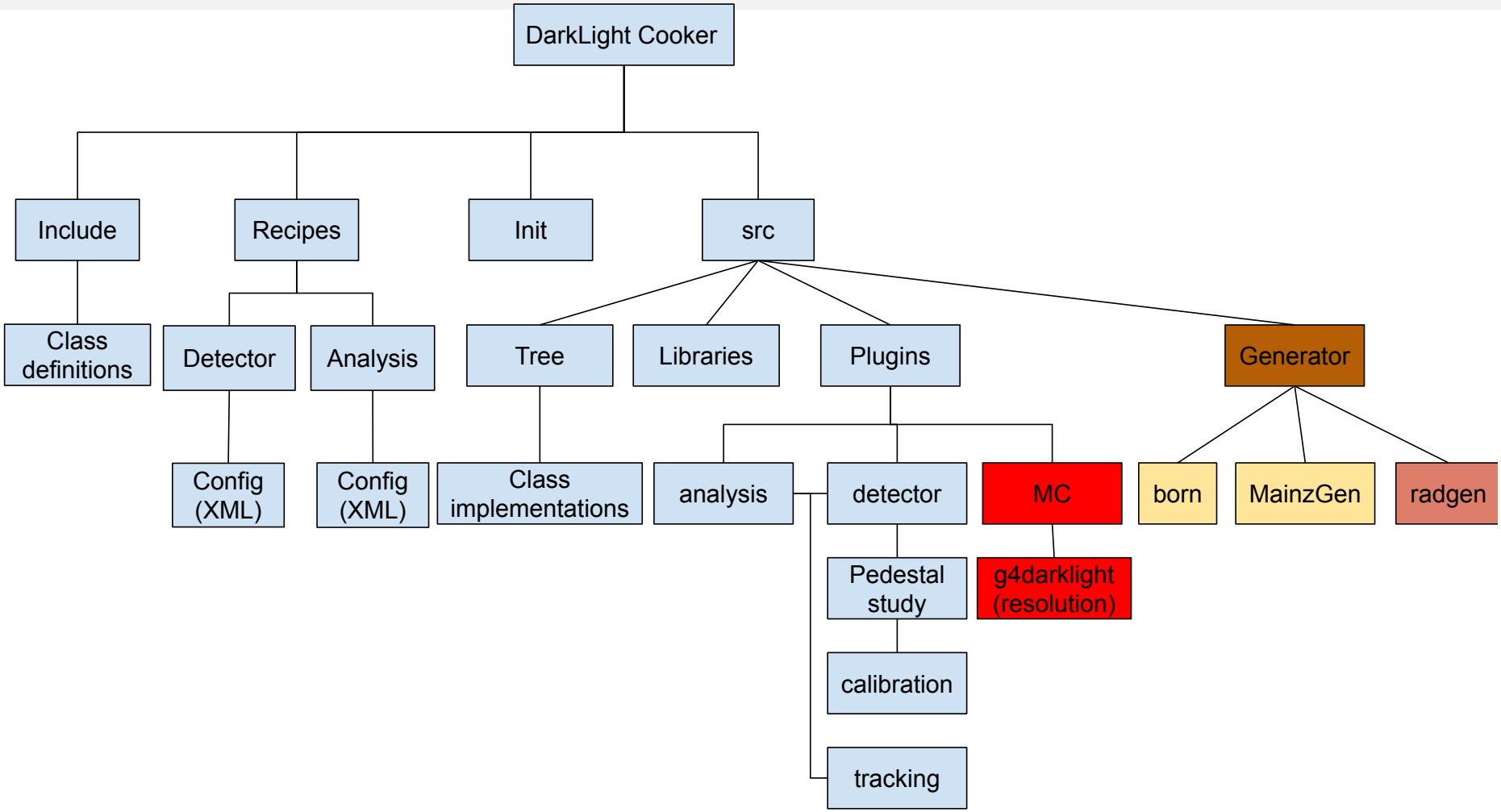
Stony Brook University, Stony Brook NY 11794

July 10, 2024

Overture

1. Introduction
   - What's New?
2. Running cooker
   - Peeking Under the Hood
3. Sim & Data Flow
   - Framwwork and Data Flow
   - Anatomy of a Plugin

Updates Since Last Time

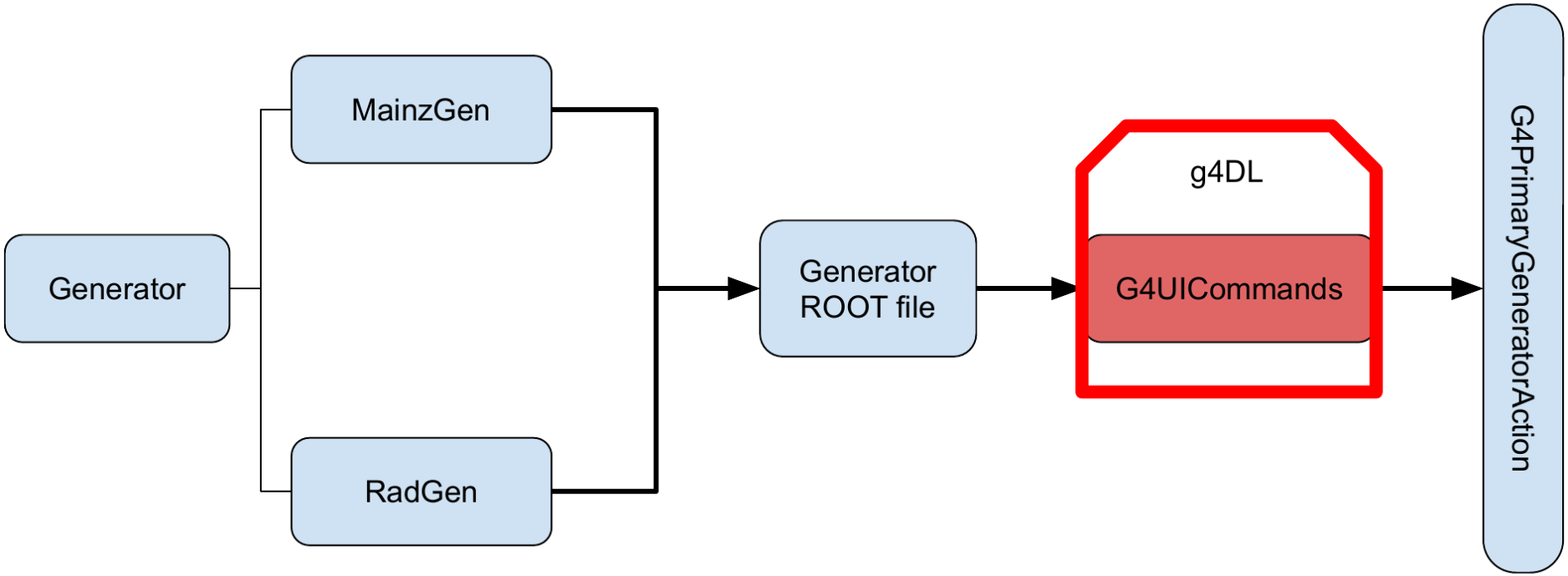


Updates Since Last Time

Updates: Not Exhaustive

- Fully incorporated: `g4darklight`, `MainzGen & RadGen` into `cooker framework`
- Turn on `build options` as follows:
  `cmake ../ -DDo_MC=ON` $\longrightarrow$ `g4DL`
  `cmake ../ -DDo_RadGen=ON` $\longrightarrow$ `RadGen`
  `cmake ../ -DDo_MainzGen=ON` $\longrightarrow$ `MainzGen`
- All sub-components of `cooker` can be developed independently within the framework
- Geometry and material updates: `Trigger hodoscopes, GEMs, APV cards & PMTs`
- `CAD geometry` is up to date(?)
- Digitization is now available for `GEMs` and being developed for `hodoscopes`
- Full suite of `In-house Generator output` can be simulated with `Geant4`
- `GEM` analysis `plugin` is currently under development

How To: G4UICommands

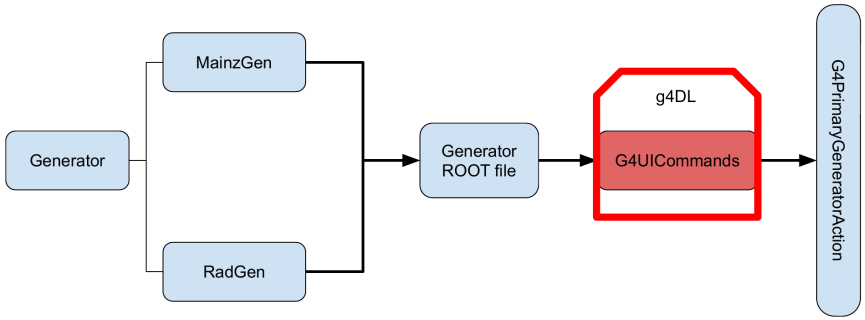


- RadGen: `./bin/darklight_test 1 output.root SampleSetupFile output_cross_section 1000000`
- MainzGen: `bash runMainzGen.sh`
- Preserve `MultiThreading`

How To: G4UICommands

g4DL UI Commands

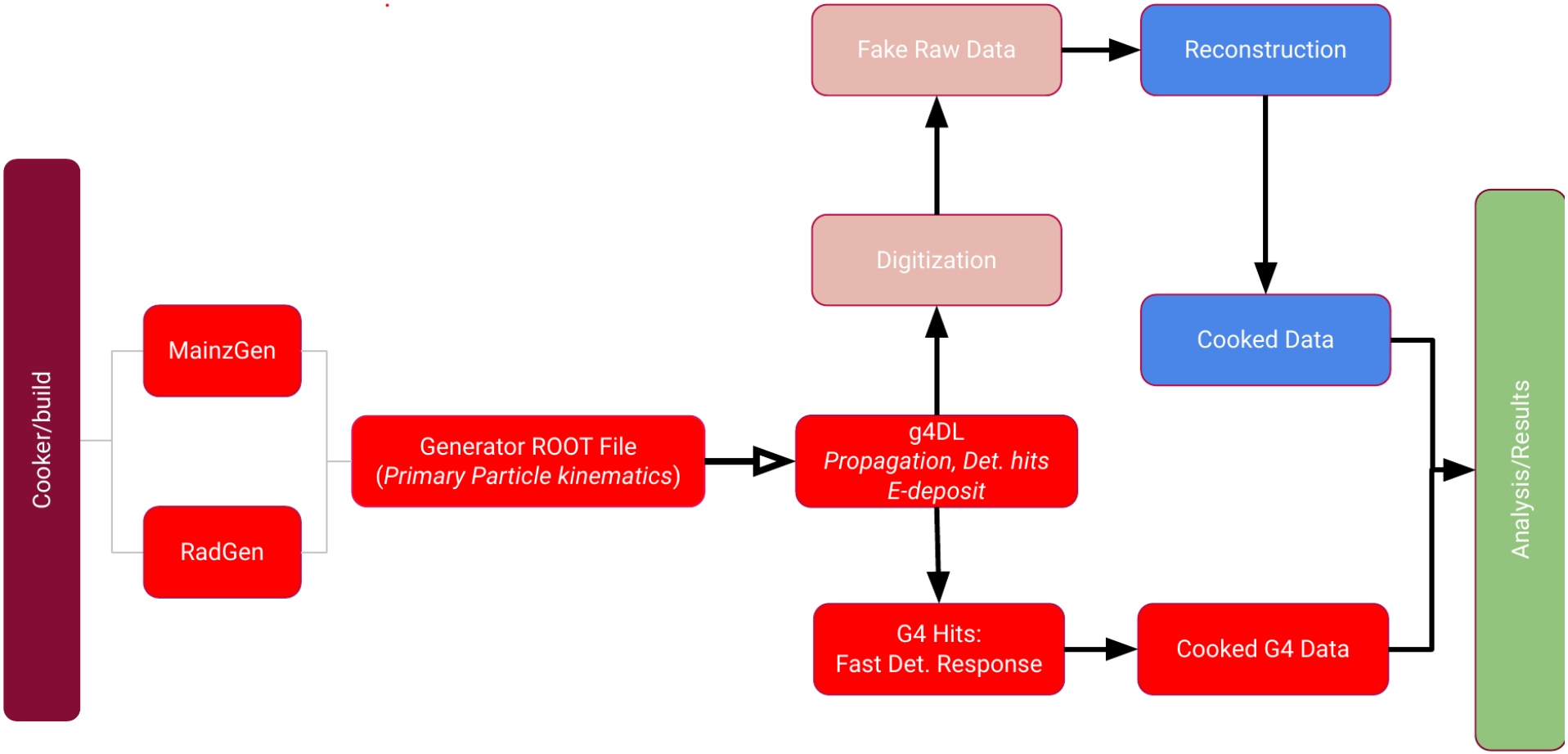
- /gun/mode [<mode>]
- /gun/mode basic
- /gun/mode random: randomized particle gun
- /gun/mode file: read in a data/text file
- /gun/mode hempmc
- /gun/mode bgtest
- /gun/mode validation
- /gun/rootfile <filename>: generator ROOT file



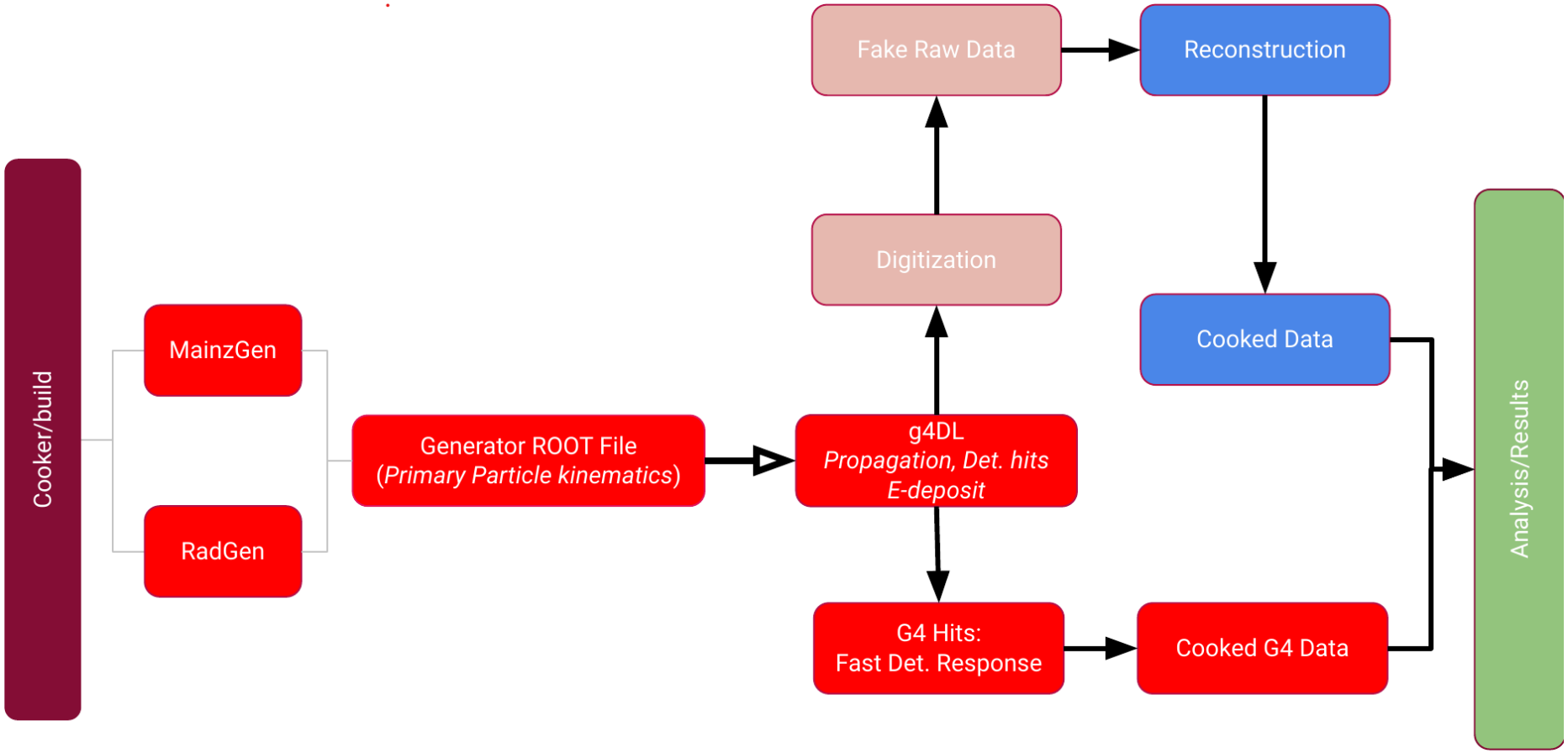
Generator ROOT file

- Using generator ROOT file the order is important
- /gun/rootfile <filename>
- /gun/mode genroot
- /run/beamOn <numberOfEvents>

Sim Flow

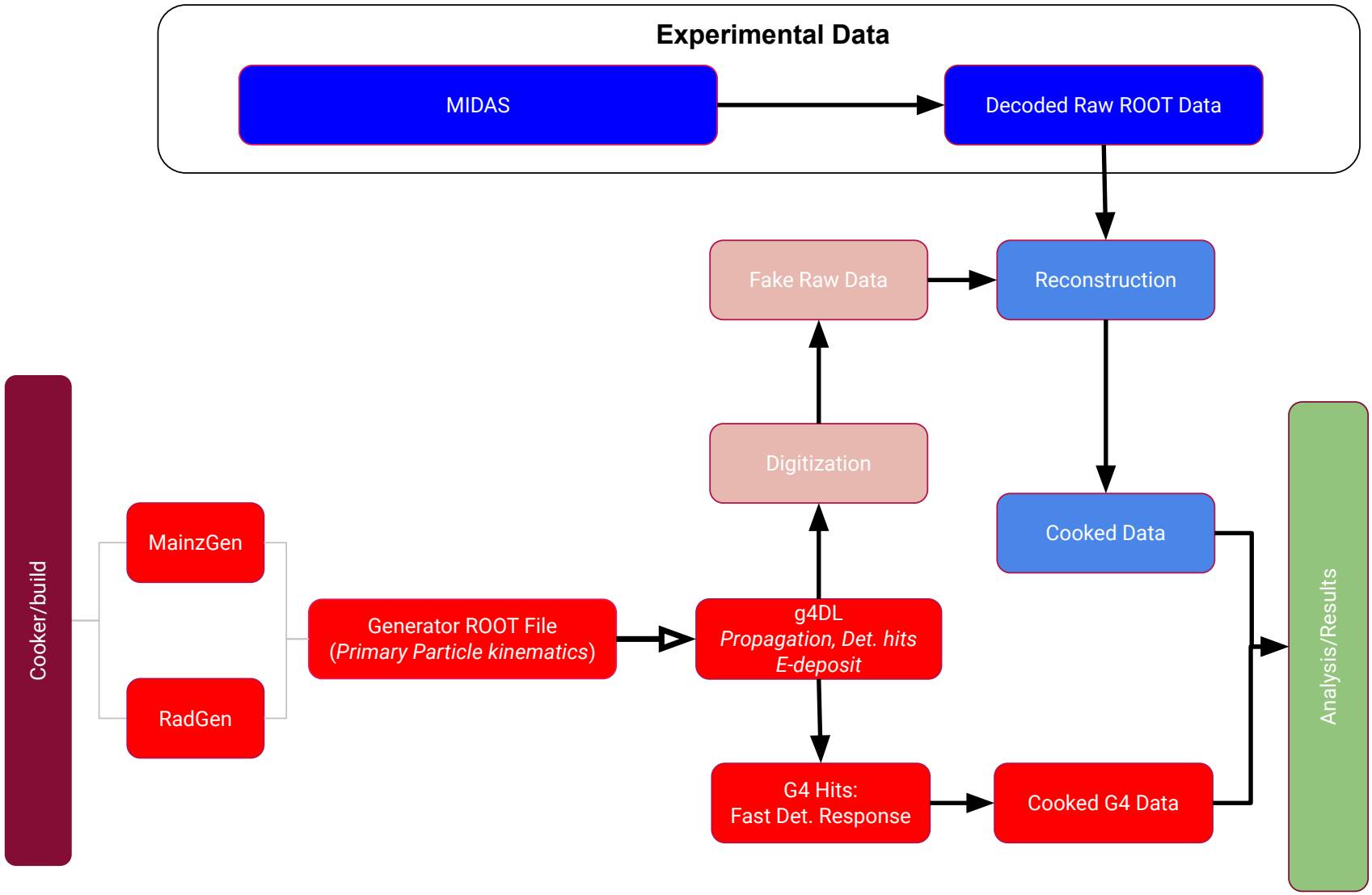

Cooker/build
MainzGen
RadGen
Generator ROOT File
(Primary Particle kinematics)
g4DL
Propagation, Det. hits
E-deposit
Digitization
Fake Raw Data
Reconstruction
Cooked Data
G4 Hits:
Fast Det. Response
Cooked G4 Data
Analysis/Results

Sim Flow

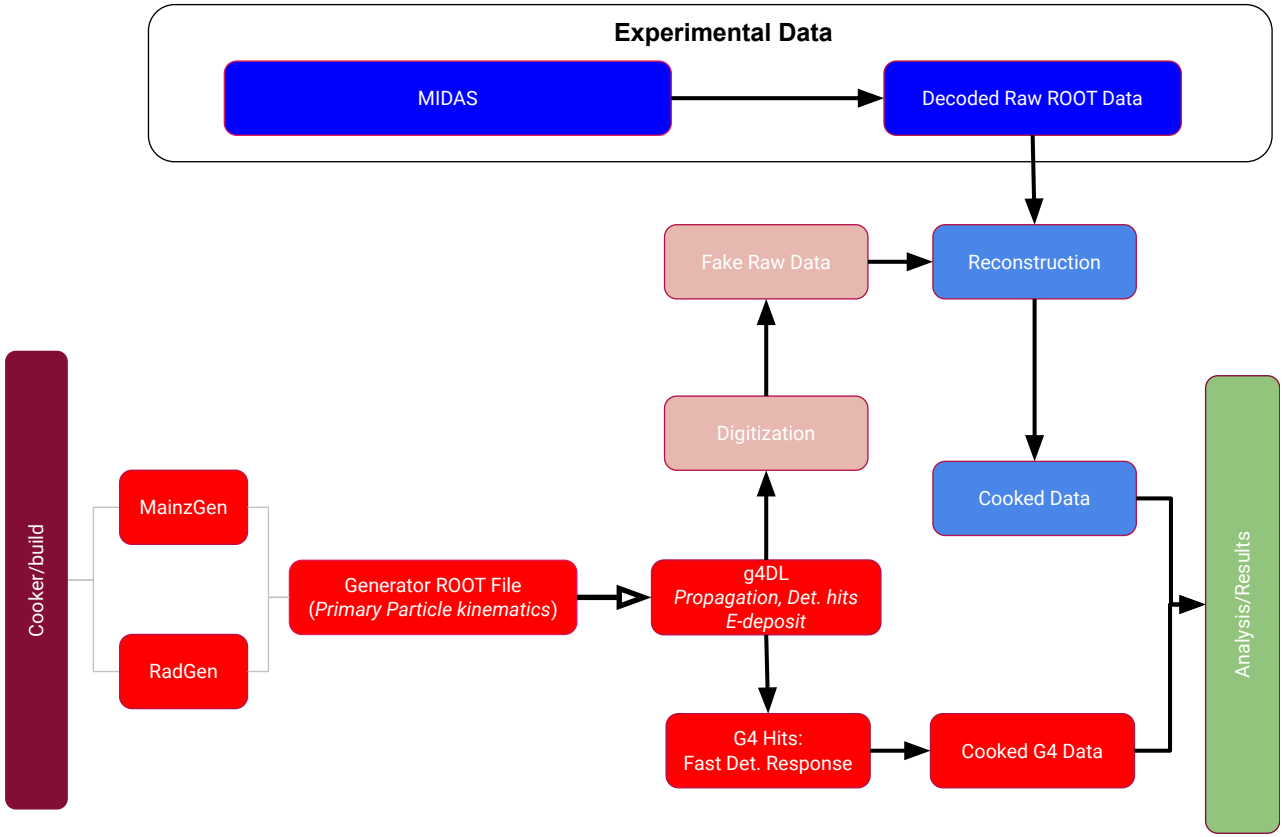


- Laura working *GEMini digi*
- Win working on `Trigger hodoscopes digi`

Sim & Data Flow



Sim & Data Flow



- Daniel working on `Fast Det. Response`
- Bishoy working on `LumiGEM` analysis code

How To: Writing a Plugin

- An example of how to create a `plugin` already exists
  - located: `src/plugins/; ls`
  - `example/`
  - Copy `example dir` to desired location: `detector, analysis` or `subdirectory`
- Change to new directory
- Call `init.sh` with new `plugin name` as follows:
  - `./init.sh <NewPluginName>`
  - Edit top-level `CMakeLists.txt`: `src/plugins/path-to/<NewPluginName>` to register your new directory

How To: Writing a Plugin

Constructor/Destructor

```
Det_GEM::Det_GEM(TTree *in_,TTree *out_,TObject *p_):Plugin(in_,out_,p_){}
Det_GEM:: Det_GEM(){}
```

Startup Call

```
Long_t Det_GEM::startup(){
 //Initialize variables and create new branch on the output TTree
 gemout=new GEM();
 out->Branch("GEM",&gemout,16000,0); //Splitlevel=0
 GEMi=0
 TBranch* bGEM=in->GetBranch("GEM"); //lookup GEM input
 bGEM->SetAddress(&GEMi); //Boolean check to ensure Branch exists
 return 0;
}
```

How To: Writing a Plugin

Event-loop: process routine

```
Long_t Det_GEM::process(){
 gemout->hits.clear();
 .
 .
 .
 GEM0_LocalX.push_back(<someVar>);
 return 0; }
```

Finalize

```
Long_t Det_GEM::finalize(){
 //Called at the end of cooker run process
 //Plot histos and create TCanvas
 return 0; }
```

Factory

```
extern "C"{
 Plugin *factory(TTree *in_,TTree *out_,TObject *p_)
{
 return (Plugin*)new Det_GEM(in_,out_,p_);
}
}
```