Introduction to LArSoft

What is LArSoft?

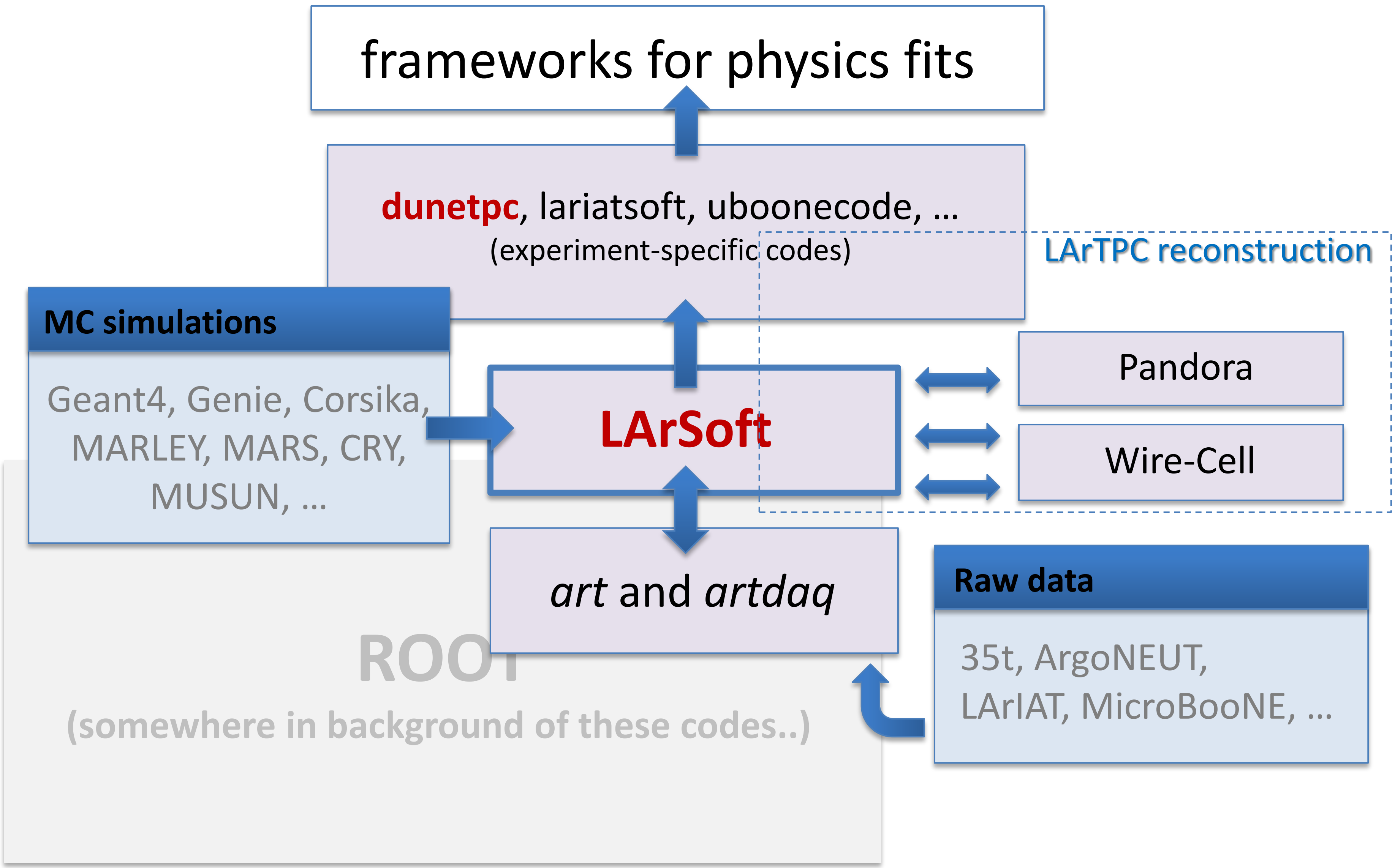
Looking for a definition found this, short and clear:

A toolkit to facilitate simulation, reconstruction and analysis of events from liquid-argon TPC-based detectors.

G.Petrillo

- it is based on art (*event-processing framework* for particle physics experiments)
- it is interfaced/uses
    - external MC physics simulation toolkits
    - other LArTPC reconstruction toolkits
- experiments build on top of LArSoft their specific toolkits
- experiments share their generic tools using LArSoft

Where is LArSoft and DUNE code?



Who is writing LArSoft?

- **major part is added by all experiment collaborators – it is YOU**
  - really!   > 100 developers registered

- **project is coordinated by the Coordination Group**
  - bi-weekly meetings: https://indico.fnal.gov/categoryDisplay.py?categId=405
  - mailing list: larsoft@fnal.gov

  - **rapid development cycle: new release each week**

- several gropups at FNAL are contributing / collaborating
  - LArSoft Architecture group
  - Software and Computing Coordinators Group
  - Steering Group

  - art framework
  - Scientific Computing Division Reconstruction Group
  - Scientific computing simulations
  - …

LArSoft documentation

Remember: > 100 developers registered and potentially developing!

Write code following policies and recommended practices.

- not all is ready: it is OK to propose / ask for recommended way to go
- follow: 1) code examples; 2) existing code with caution
- others will follow your code

LArSoft from overview to all details and links: http://larsoft.org/

art Workbook to understand the base: http://art.fnal.gov/art-workbook/

Simulations steps

particle gun

or

beam simulation

and/or

cosmogenics — Corsika

1

particle list, kinematics

general det.sim. — Geant4

→ use test-beam data to improve models

- intelectual effort on models
- **...and on tools for extracting knowledge from data**

2

energy depositions, light

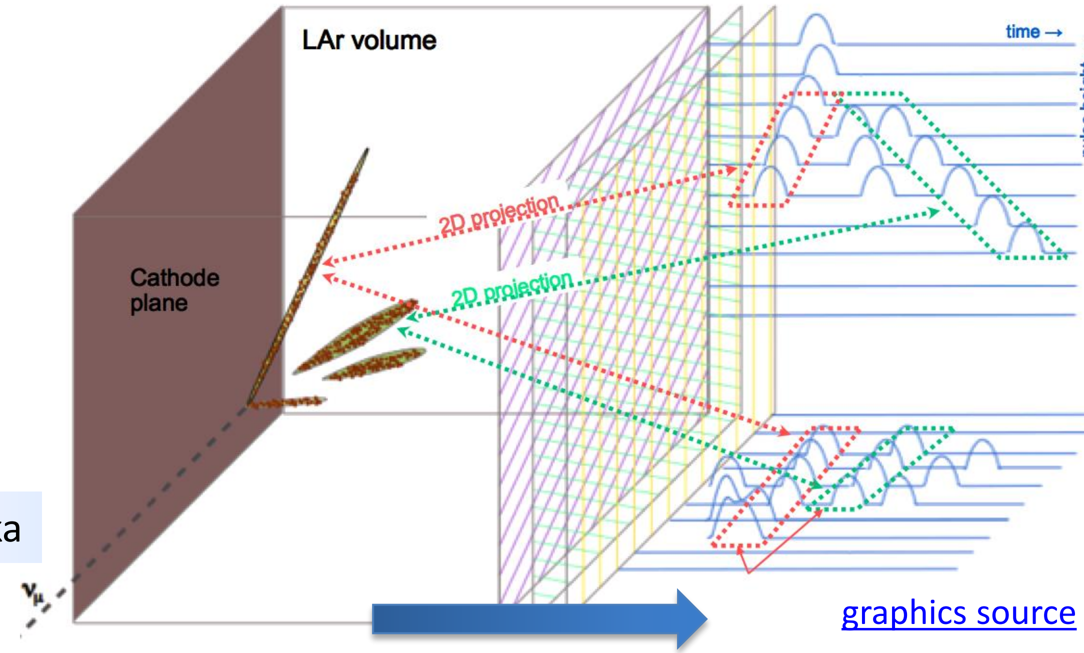
specific det.sim.

→ place for many improvements, MC is still to simple

- E field distortions: space charge, det. construction
- signal in induction and collection planes
- electronics noises

3

raw data like from real detector



graphics source

Reconstruction step(s)

raw data, sim. or real detector

↓

deconvolution

↓ ionization = f(time)

hit finding / 3D imaging

↓ 2D hits / 3D cells

4

pattern recognition

trajectory fit

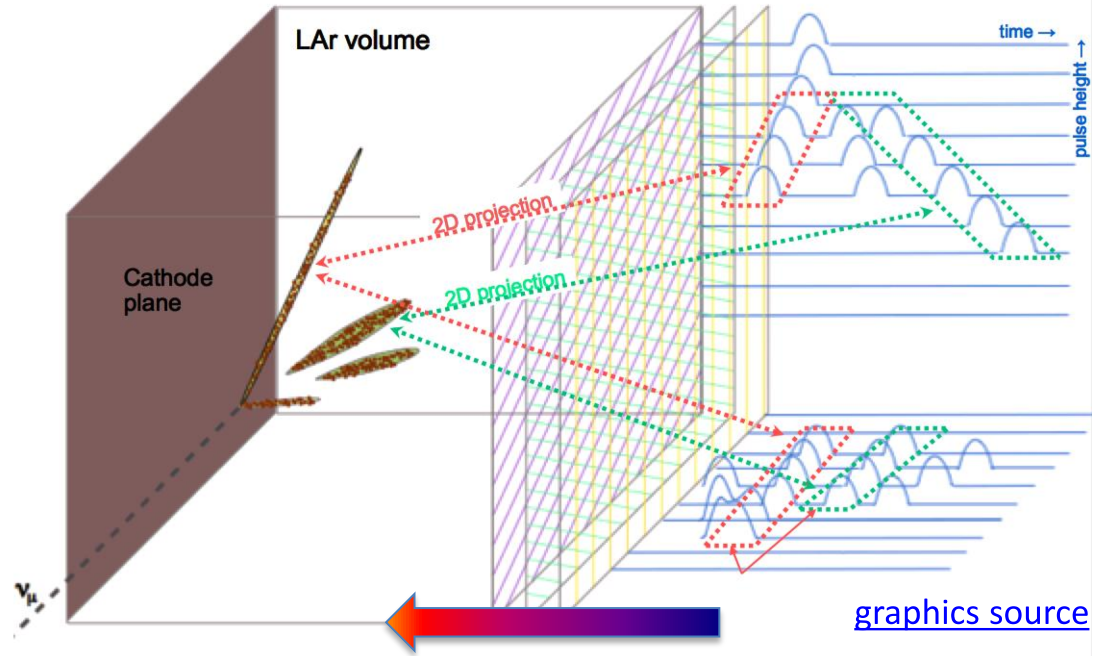
vertex finding

EM showers

hierarchy recognition

↓

full event, ready for PID/kinematics



graphics source

→ various approaches can mix the order in this sequence

→ modular design in art/LArSoft is a good ground:

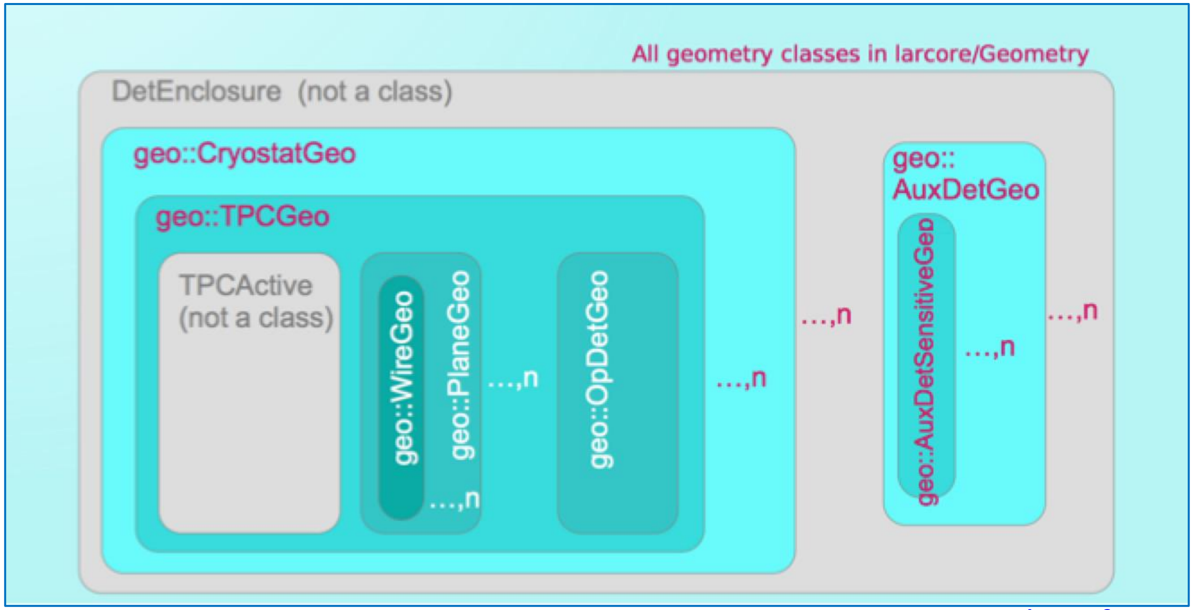
- algorithms imported from other experiments/fields
- Pandora takes hits and returns reach pattern recognition output
- Wire-cell progresively interfaced to use output inside LArSoft

A LOT IS UNDER CONSTRUCTION HERE

Geometry

simulation / reconstruction algorithms are (should be…) detector agnostic

- read the geometry information from [geo::Geometry]() service
- detectors have one / many cryostats, one / multiple TPC volumes, 2 / 3 readout planes
  → framework is ready for it
- dual-phase is a new challenge:
  - drift direction is Y (vertical), LArSoft assumed X (horizontal)
    → under investigation, ad hoc detector is rotated by 90° and gas layer is on side
  - no spacing and no drift between readout planes (minor effects)



source: larsoft.org

Services

Classes with a single instance managed by the framework (see link).

- Geometry – as already mentioned

```
const auto & geom = *(art::ServiceHandle< geo::Geometry >());
geom.TPC(itpc, icryo).HasPlane(geo::kU)?
```

- Channel status
  - Channel is noisy, bad, channel counts, …

```
const auto & chStatus = art::ServiceHandle< lariov::ChannelStatusService >()->GetProvider();
chStatus.IsGood(ch)?
```

- Detector properties
  - readout sampling rates and readout window size
  - conversion between readout ticks, times, drift distance
  - dQ/dx → dE/dx
  - electron lifetime

```
const auto & detProp = &*art::ServiceHandle< util::DetectorProperties >();
double x = detProp.ConvertTicksToX(hit->PeakTime(), view, tpc, cryo);
```

- LAr properties
- E-field map
- …

Modules and Algorithms

(see link)

Algorithm class performs simulation / reconstruction / analysis / … task.

- **here is (usually) your intelectual contribution and freedom**
- ideally: independent from the framework
- algorithms use services providers (framework independent interfaces to services), FHiCL configuration, LArSoft data products – this is OK

Module class manages algorithms in order to produce and deliver a result to the framework.

- modules are art framework concept
  - producers, can modify art::Event and add new data products

    → simulation, reconstruction
  - filters, can modify events, return boolean value which may prevent from execution of the following modules in a *path*
  - analyzers, can only read events, use art::TFileService to output results, usually into ROOT's TTree
- simple codes tend to start and stay inside module, be careful when they grow and separate into algorithms

Data products

(see link)

Classes which describe results of algorithms, and can be saved into art::Events.

Once constructed, they are read only.

Describe all stages:

- simulation: simb::MCParticle, sim::SimChannel
- detector outputs: raw::RawDigit, raw::Trigger
- reconstructed objects: recob::Wire, recob::Track
- higher level reconstruction and analysis results: anab::T0, anab::Calorimetry

LArTPC reconstruction / analysis specifics is to use results from all levels at once:

- looking at event identification and single hit charges from given kind of clusters is an every day task
- reconstruction results are plenty of associations between data products: art::Assn
- navigation through event using art::Assn is known to be not easy → but tools are getting better
- we'll have a very simple example today

Configuration

(see very in depth slides)

LArSoft uses FHiCL (Fermilab Hierarchical Configuration Language), it let's you to configure art/LarSoft job:

- algorithms / modules / services parameters
- job itself: input/output/histogram files, sequences (*paths*) of producers and analyzers
- we'll see how it looks like and works in practice today.

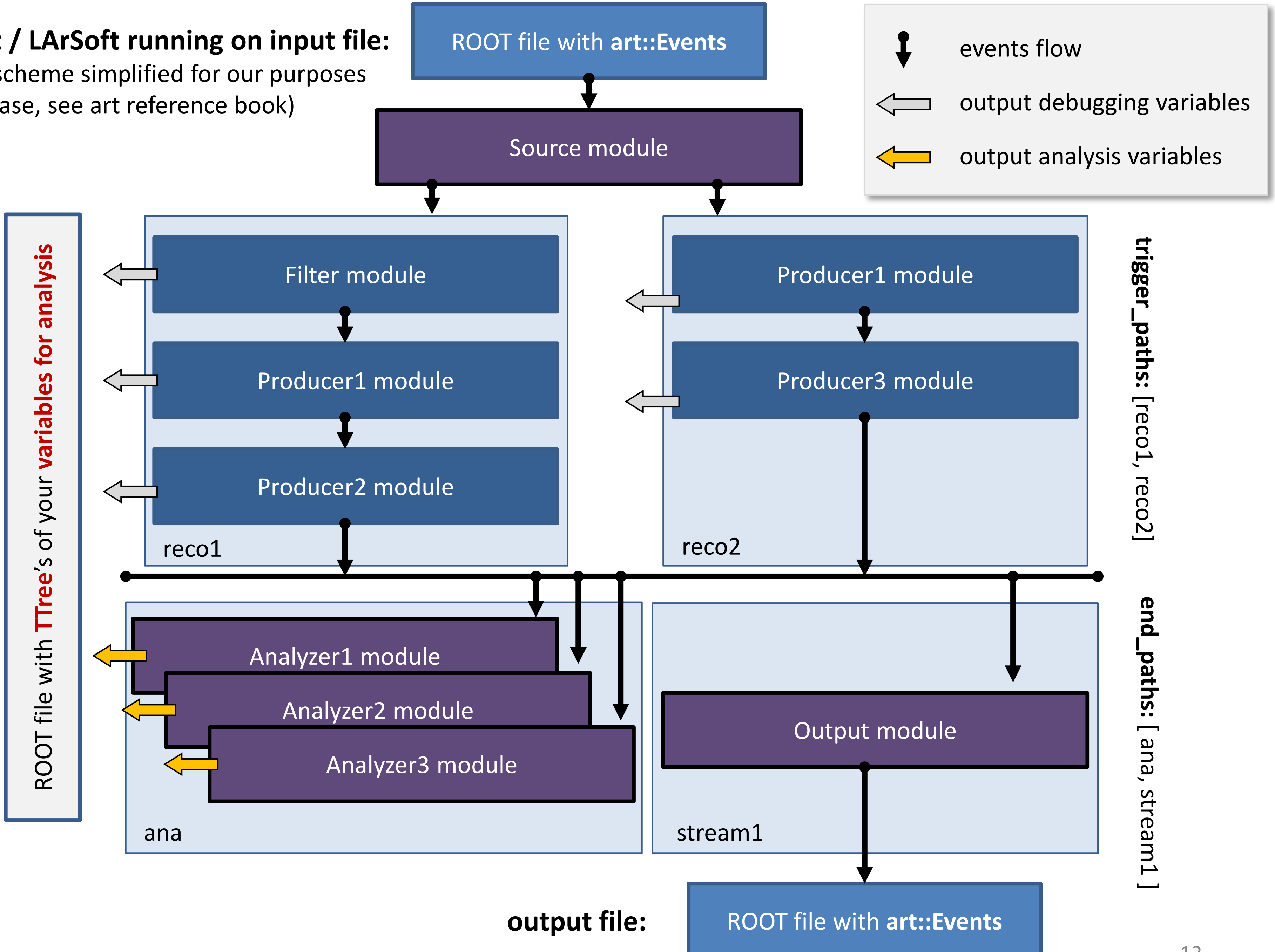
Full LArSoft job configuration is always huge. It is broken into tens of nested tables of parameters, spread over all places in all repositories. The general structure is:

- algorithm and module classes are accompanied with default configuration files
- experiments collect tables with overwritten module parameter values in their .fcl files
- finally standard / your own job file collects services config, gives names (*labels*) to producers and analyzers, puts them in sequences (*trigger_paths*, *end_paths*) and makes last changes to paramer values

- new (~2 year old) feature lets you **validate parameters in code** - use it, typos are easy and dangerous!
- **check the compiled configuration** with:
```
lar  -c config.fcl  input.root  --config-out dump.txt
lar  -c config.fcl  input.root  --config-out dump.txt  --annotate
```

art / LArSoft running on input file:

(a scheme simplified for our purposes
please, see art reference book)



Excersises after coffee