How to start one analysis on Belle II

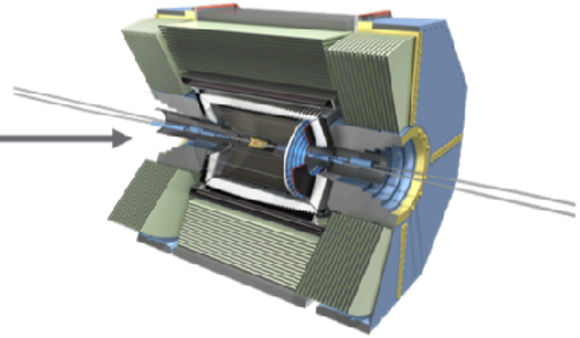
Speaker: Junhao Yin

味物理与CP破坏研讨会

The big picture

Nature

PYTHIA



Published paper

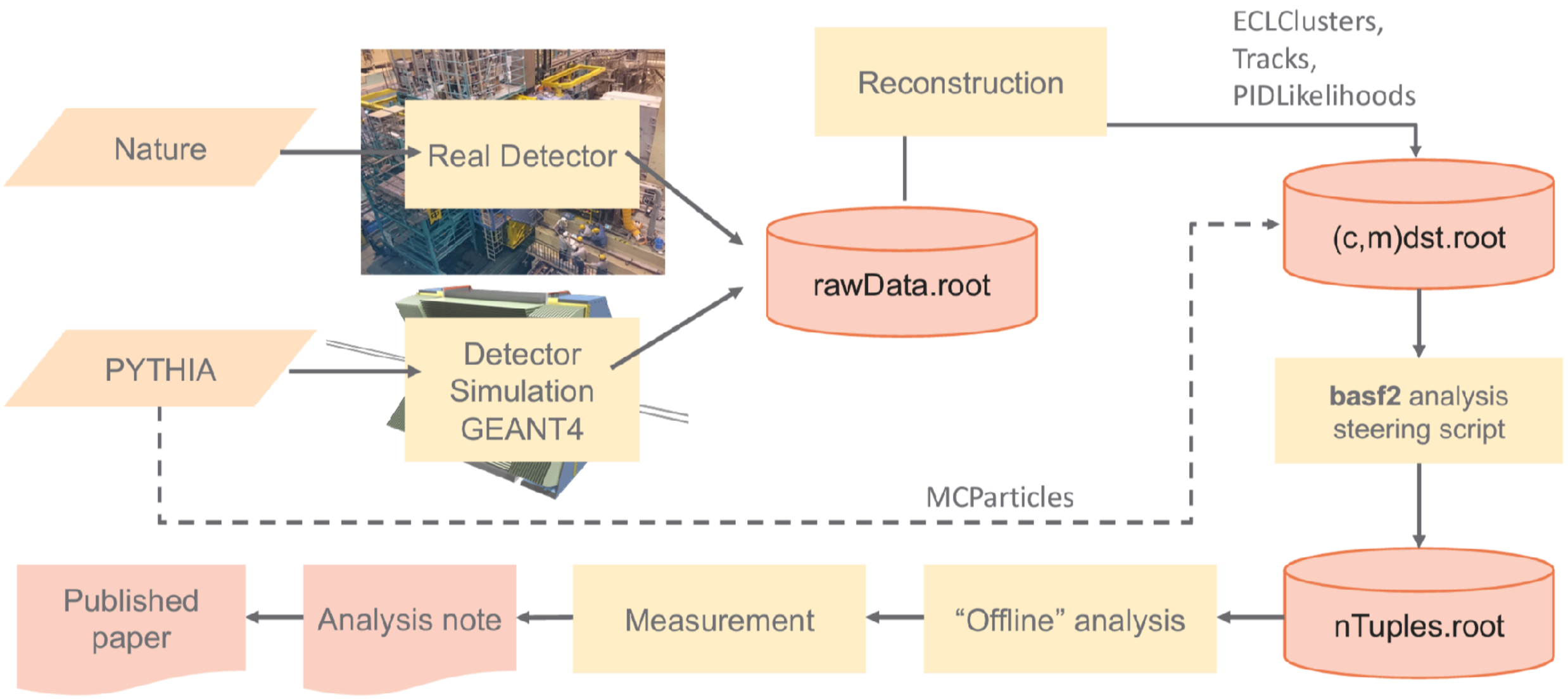
What do you need?

Data storage → root (trees) files
Data processing → C++ code
Scripting → Python

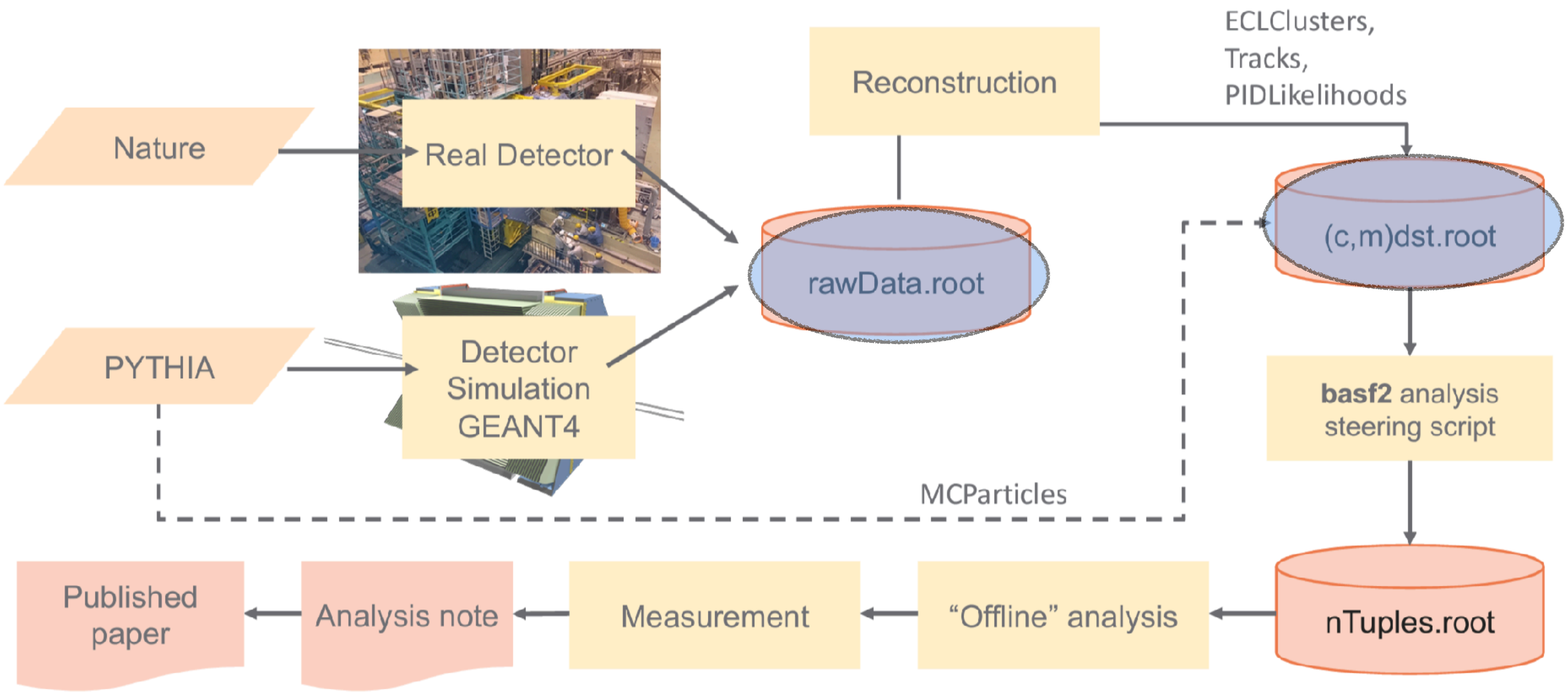

JARGON


WE NEED SOME NEW JARGON, THE PUBLIC ARE STARTING TO UNDERSTAND WHAT WE'RE TALKING ABOUT!

The big picture



The big picture

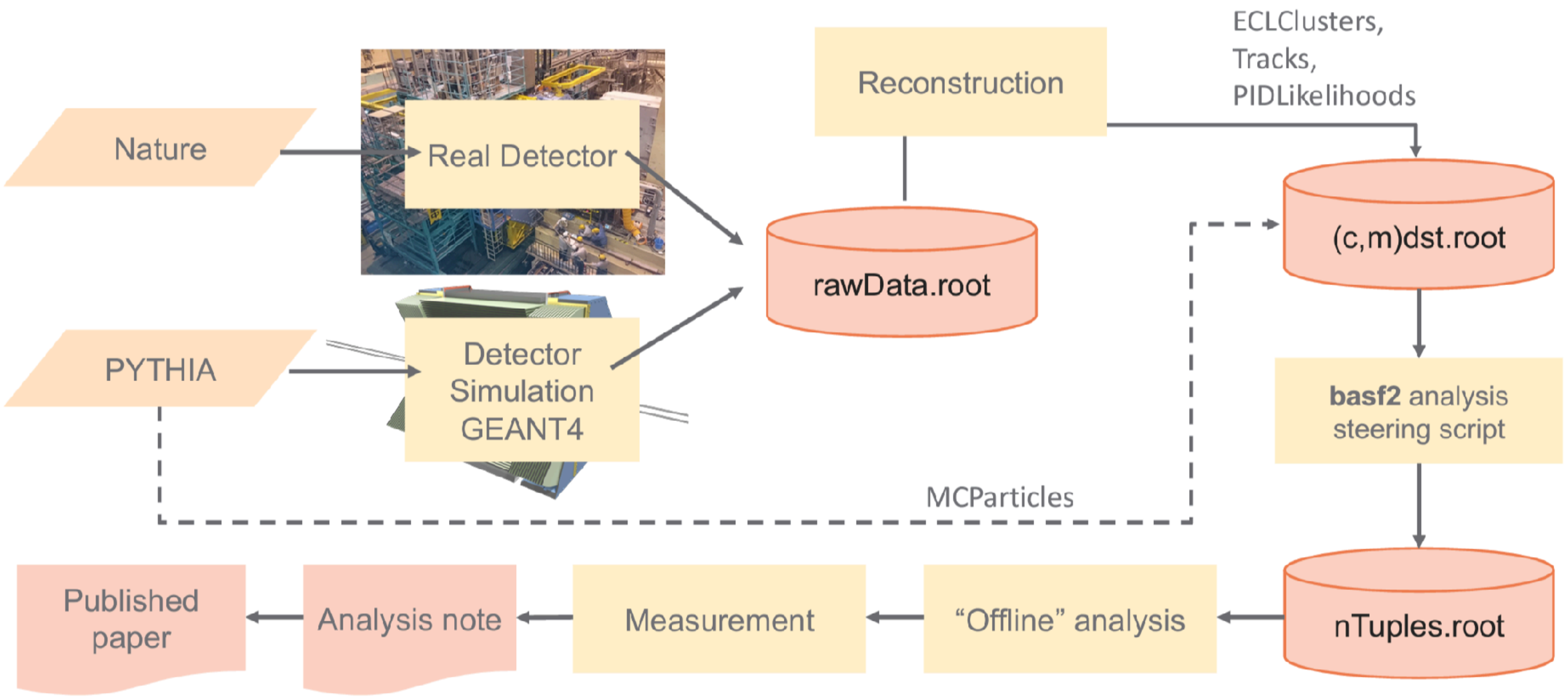


DataStore

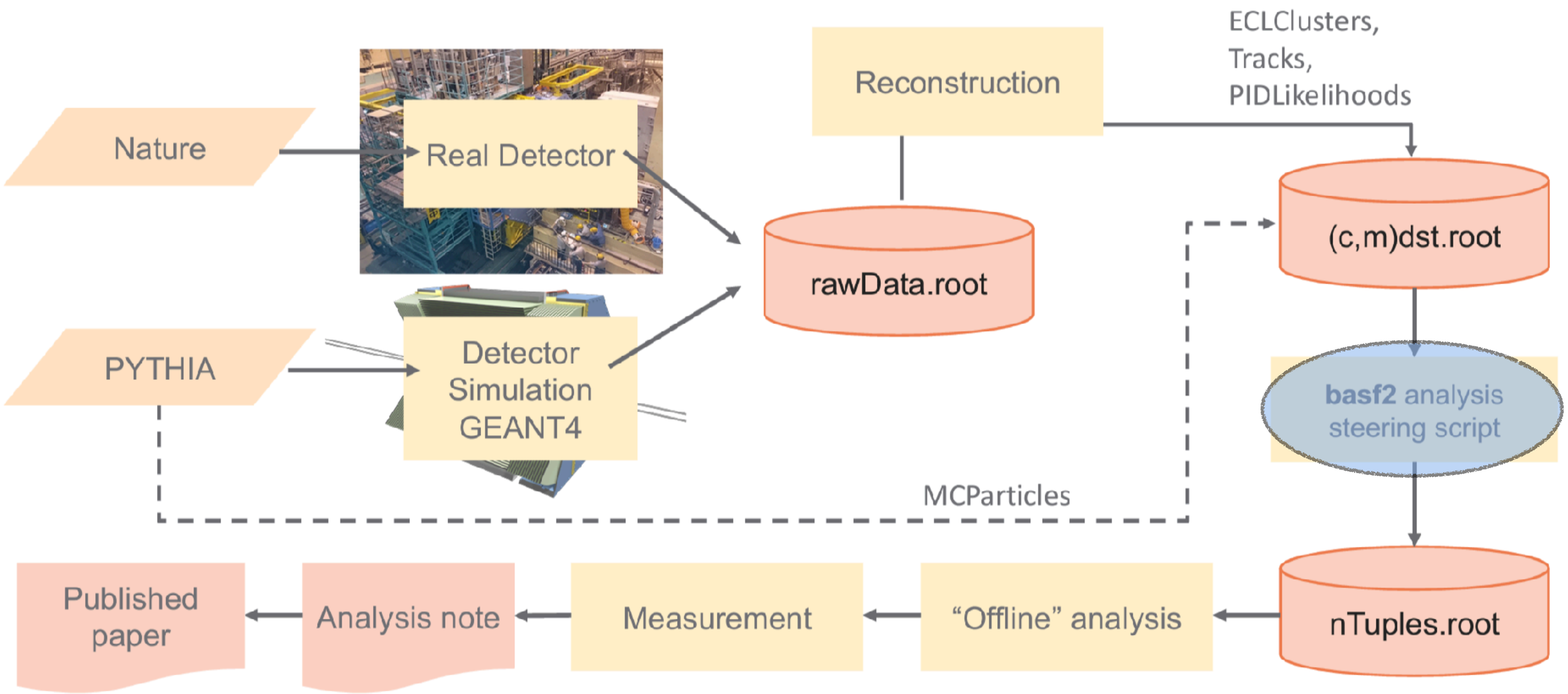
- A dst contains objects which will populate a DataStore.
  - data summary table
  - Basically: a special ROOT file.
- The data for physics analysis are "mdst”
  - mini data summary table.
  - Same structure of a dst, **but with much less information**
  - Input to your analysis package scripts
- The calibration & performance are “cdst”
  - calibration data summary table.
  - mdst + digits
- At the end of your analysis chain you will write out a “normal" root file containing a TTree, TNtuple, or histograms

A relevant question
https://questions.belle2.org/question/219
Objects allowed in an mdst:
https://goo.gl/AB15Ud

The big picture



The big picture



What is “basf2”

belle 2 analysis software framework

basf2 is C++14 **"under the hood"**

- Packages contain C++ **modules** to manipulate data.

- In analysis: we have code to build **particles** from primitive objects (like tracks and calorimeter clusters).
- We also calculate physics quantities, and apply cuts.

Python 3.6 code for **steering**

- Load and configure C++ modules
  - analysis modules and modules from other packages
- Also python does *some* high-level analysis tasks.

- You will write a fair bit of python during the workshop.

Easy to read and use!

First step—Set the environment

After login kekcc:

```
$ source /cvmfs/belle.cern.ch/tools/b2setup release-XX-YY-ZZ / light-XX-YYYY
```

Always use the latest release/light version, for example: release-06-00-14 / light-2212-foldex

If you are not sure about the latest version, use:

```
$ b2setup --help
```

to check the available releases, or use

```
$ b2help-releases
```

to check the recommended release

*release:

>A full package, including everything: analysis, pxd, svd, trg, etc…

*light:

>A light release, only a few packages: analysis, mdst, skim, b2bii, etc…

>Suitable for analysis!

basf2 --info

release version

local release location will show if local work

externals:
/cvmfs/belle.cern.ch/sl7/externals/

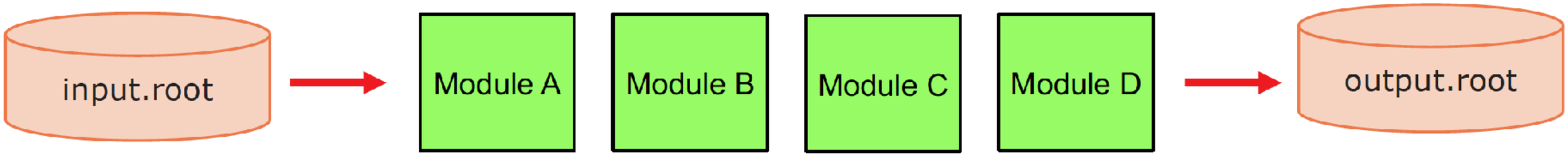
python version

ROOT version

```
                    eeeeeeee
              eeeeeeeeeeeeeeeeee
         eeeeeeee        eeeeeeeeeeeeeee
       eeeeee           eeee         eeeeee
      eeee              eeee           eeee
       eeee            eeee           eeee
        eeeee          eeee          eeee
           eeee       eeeeeeeeeeeeeeee
                      eeeeeeeeeeeee

                    eeeeeeeeeeeee
                    eeeeeeeeeeeeeeeee
      eeee          eeee            eeeeee
    eeee            eeee             eeeeee
  eeee             eeee                eeeee
 eeee              eeee                eeeee
   eeee           eeee             eeeeeee
     eeeeee      eeeeeeeeeeeeeeeeeee
        eeeeeeeeeeeeeeeeeeeeeeee
            eeeeeeeeeeeeeeee

  BBBBBBB            ll ll          2222222
  BB    BB    eeee   ll ll   eeee    22 22
  BB    BB   ee   ee ll ll  ee   ee  22 22
  BBBBBBB   eeeeeeee ll ll eeeeeeee  22 22
  BB    BB  ee       ll ll ee        22 22
  BB     BB  ee   ee ll ll  ee   ee  22 22
  BBBBBBBB    eeeee  ll ll   eeeee  2222222

  basf2 (Belle II Analysis Software Framework)
Copyright(C) 2010-2021 Members of the Belle II Collaboration
 (See "basf2 --license" for more information.)
            Release release-06-00-14
            Version release-06-00-14

-----------------------------------------------
BELLE2_RELEASE:           release-06-00-14
BELLE2_RELEASE_DIR:       /cvmfs/belle.cern.ch/el7/releases/release-06-00-14
BELLE2_LOCAL_DIR:
BELLE2_SUBDIR:            Linux_x86_64/opt
BELLE2_EXTERNALS_VERSION: v01-10-02
BELLE2_ARCH:              Linux_x86_64
Default global tags:      ('release-06-00-07',)
Kernel version:           3.10.0-1160.80.1.el7.x86_64
Python version:           3.8.8
ROOT version:             6.24/00

basf2 module directories:
  /gpfs/home/belle/yinjh/MyBasf2/anal/Yb2rXb/runSig
  /cvmfs/belle.cern.ch/el7/releases/release-06-00-14/modules/Linux_x86_64/opt
```

Data processing

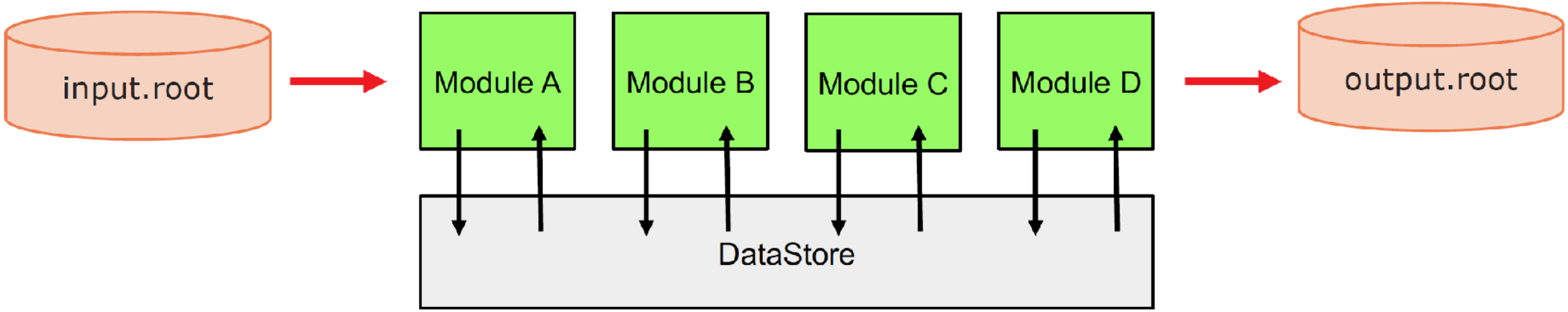
1) A set of classes (modules) that process the data
   → **BASF2 module**



Data processing

1) A set of classes (modules) that process the data
   → **BASF2 module**

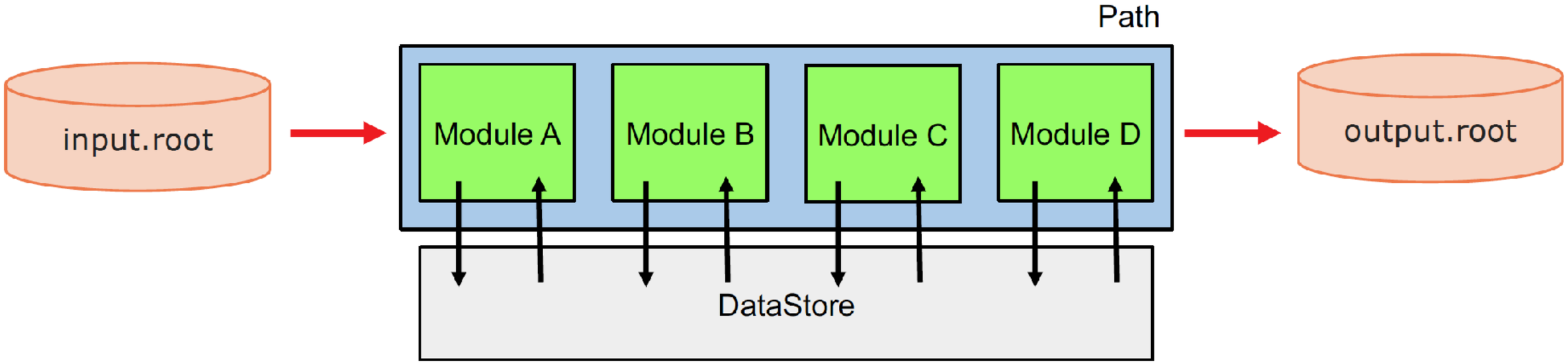
2) A set of classes (dataobjects) that hold the data and allow module to pass thing one to the other
   → **BASF2 dataStore**



Data processing

1) A set of classes (modules) that process the data
   → **BASF2 module**

2) A set of classes (dataobjects) that hold the data and allow module to pass thing one to the other
   → **BASF2 dataStore**



3) An order in which the modules must be executed
   → **BASF2 path**

Looking for more details?

questions https://questions.belle2.org
for anything, not just analysis

documentation https://software.belle2.org
there is fairly good documentation

examples $BELLE2_RELEASE_DIR/<packagename>/examples
for today <packagename> = analysis

the code https://stash.desy.de/projects/B2/repos/basf2/browse
$BELLE2_RELEASE_DIR/

A simple example of MC production

1. Create your own path

```
import basf2 as b2

# Defining one path
my_path = b2.create_path()
```

2. Call a function to load a module and add it to your path.

```
from modularAnalysis import setupEventInfo
setupEventInfo(100, path=my_path)
```

3.Call other modules, i.e. generator, simulation, reconstruction, output

The procedure is same no matter what module you want in path.

- "Register" the module you want
- Set "param"eters to the module
- "Add" the module to you path

4. Process the path

A simple example of MC production

```
import basf2 as b2
from modularAnalysis import setupEventInfo
from mdst import add_mdst_output
from generators import add_inclusive_continuum_generator
from simulation import add_simulation
from reconstruction import add_reconstruction
```

Header

```
# Defining one path
my_path = b2.create_path()
```

create your own path

```
# generation of 100 events according to the specified DECAY table
# e+e- -> ccbar -> D*+ anything
# D*+ -> D0 pi+; D0 -> K- pi+
#
setupEventInfo(100, path=my_path)
```

Produce 100 events

```
add_inclusive_continuum_generator(finalstate="ccbar",
                                  particles=["D*+"],
                                  userdecfile=b2.find_file('analysis/examples/simulations/B2A102-ccbarEventGeneration.dec'),
                                  path=my_path)
```

generate inclusive c-cbar events, each one should contain a $D^{*\pm}$

```
add_simulation(path=my_path)
```

Geant4 simulation. TRG simulation also included.

```
add_reconstruction(path=my_path)

add_mdst_output(mc=True, filename='B2A102-ccbarEventGeneration-kkmc.root', path=my_path)

# Process the events
b2.process(my_path)
```

Process the path

```
# print out the summary
print(b2.statistics)
```

Need release, not light!

FAQ about MC production

Q: Where is the beam energy setting?
A: Automatically set to $\Upsilon(4S)$ with a reasonable beam spread.

Q: What if I want to use custom beam energy?
A: You can use `beamparameters.add_beamparameters()`

Q: Include the beam background?
A: Use the option: "add_simulation(main, bkgfiles=bg)"

Please Note:
It's not recommended to generate the MC sample by yourself.
You cannot use custom MC in your final result.
Reason: 1. It's not trivial to set the correct GT totally by oneself.
2. The signal MC samples may already been produced by other people.
3. MC samples are usually huge. It's waste of storage if all MC are stored on kekcc.

Ask conveners/DP liaisons to check if your MC are produced or not.

You can always ask conveners/DP liaisons to produce the signal MC samples.

DataStore

mdst are basically root trees containing lists of:

→ Track
→ TrackFitResult
→ V0
→ PIDLikelihood
→ ECLCluster
→ KLMCluster
→ KlId
→ TRGSummary
→ SoftwareTriggerResult
→ (MCParticle)
→ ...

The analysis package has modules to convert these
Into more friendly quantities like
→ Particle
→ ParticleList
→ EventShapeContainter
→ TagVertex
→ …

DataStore

- Can I read the mdst with my own, custom made scripts and run the analysis?

DataStore

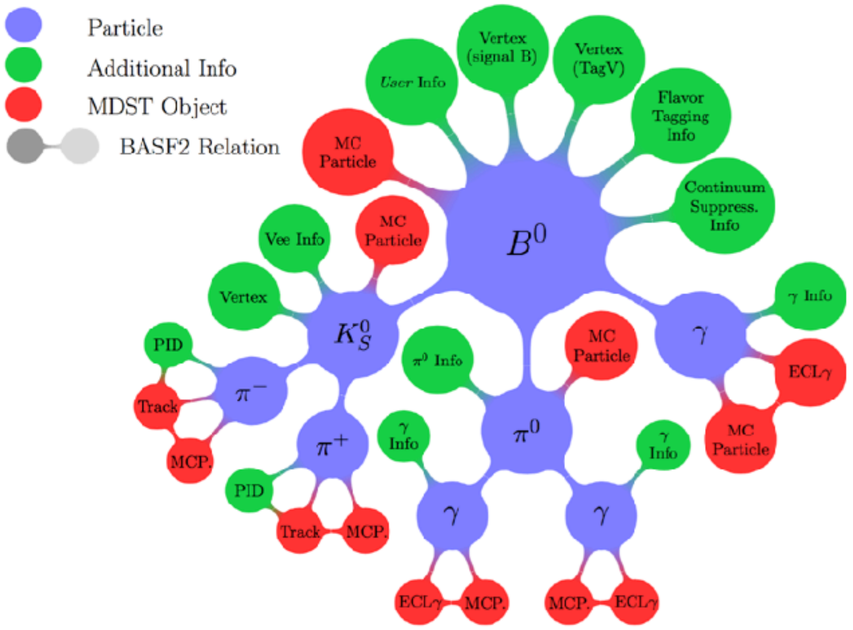
- Can I read the mdst with my own, custom made scripts and run the analysis?

NO!

DataStore

- Can I read the mdst with my own, custom made scripts and run the analysis?

NO!

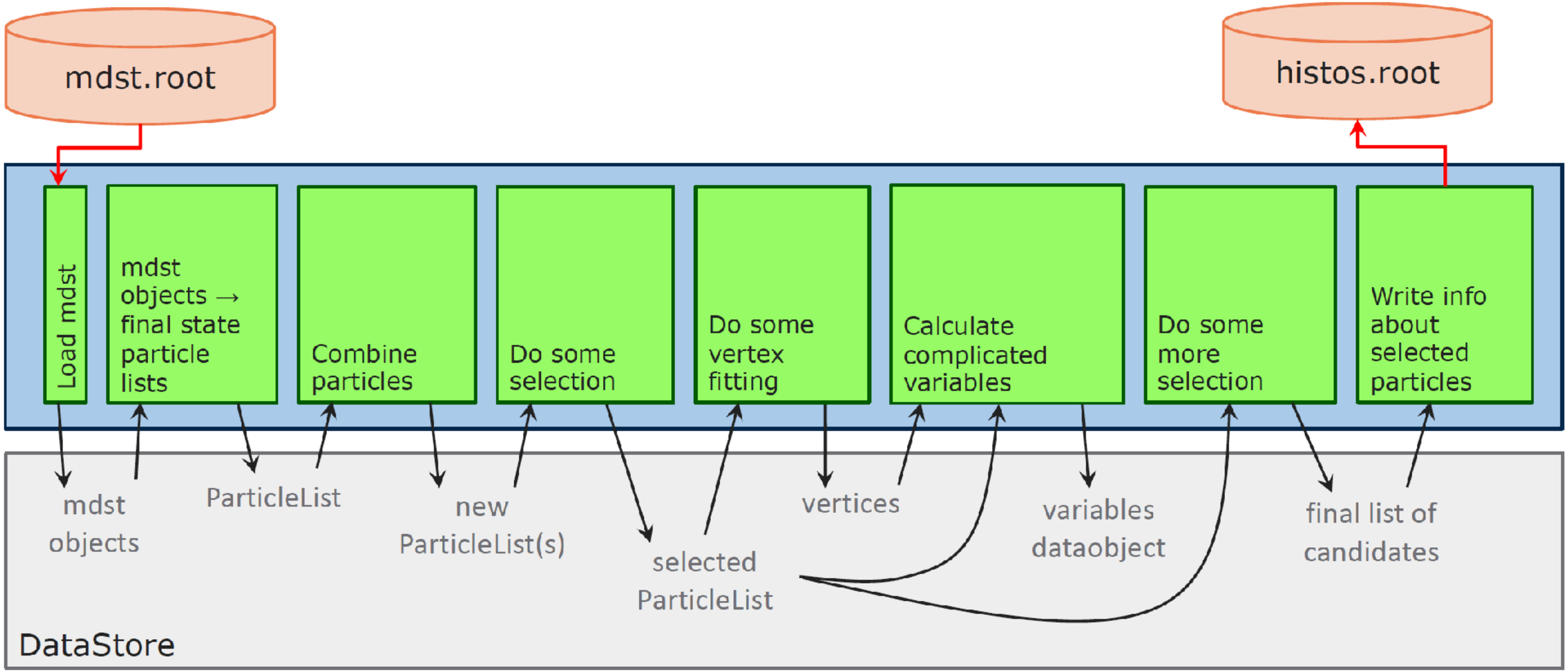


- At each stage we build relations between the dataobjects
  - Like vertex information, ContinuumSuppression → all related to Particles
  - Particles themselves related to primitive mdst objects (clusters, tracks)

- The mdst contains also the relations bewteen the objects stored in it, which are not trivially handeled by a standalone root macro.
- ***Use always basf2-based code.***

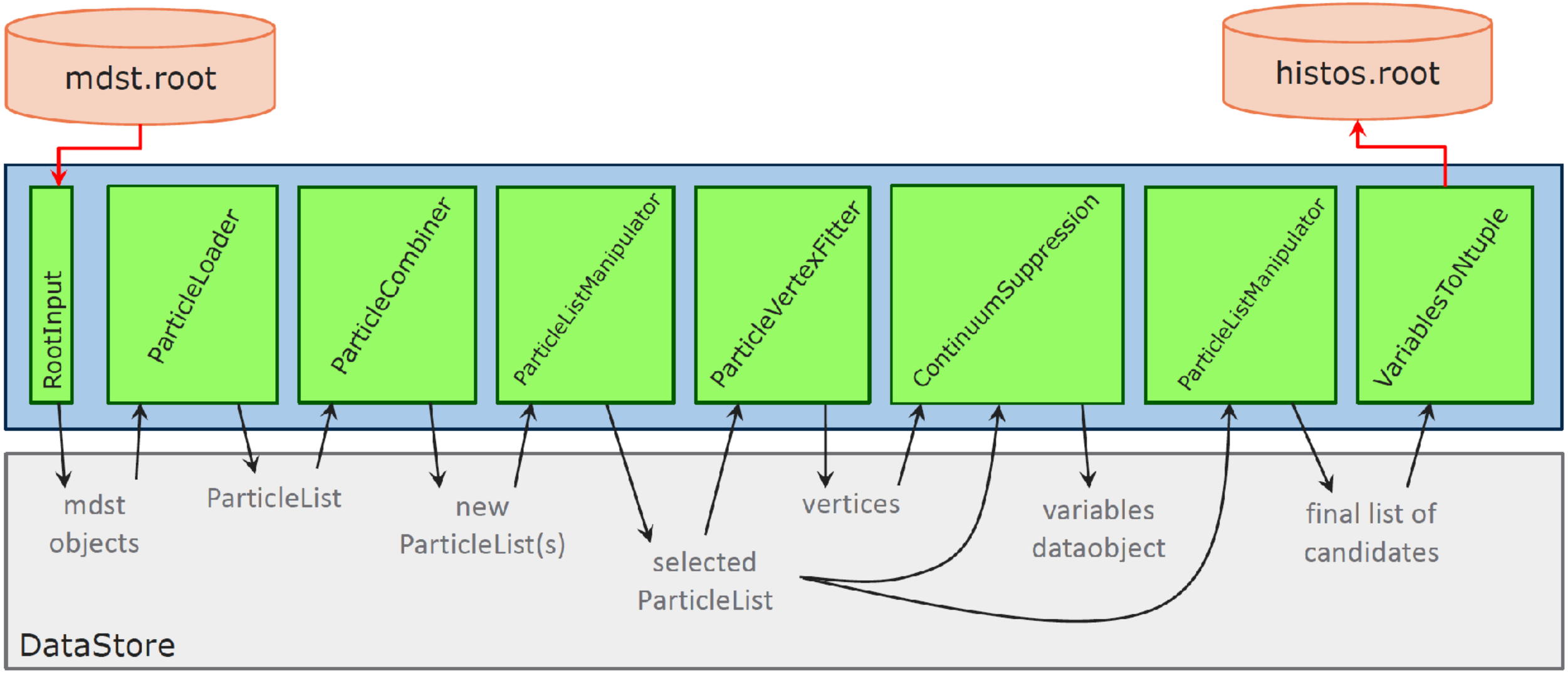
- The relation between analysis object (particles) and the reconstructed objects is not always trivial.
- One particle may have many trackFitResults.
- The ECLClusters are not photons.
- ***Use the modules provided by a detector expert***

A typical path for an analysis job



A typical path for an analysis job

Now with the real names for the modules



Particle-based analysis

- Take particle list.
- Build up decay parents from kids
- Make candidates
- Filter/cut/keep

- In most cases, you will have multiple candidates per event
  - basf2 will restore all candidates
  - We don’t need best candidate: https://arxiv.org/abs/1703.01128

Particlelist

- A common representation of all particle types
  - Charged: e / μ / π / K / p / d [built up from track + hypothesis]
  - γ [built up from ECLClusters + !Track]
  - K^0_L, n [built up from KLMlusters + ECLClusters + !Track]
  - K^0_S, Λ^0, γ [built up from V0 (2 tracks)]
  - Composite particles: π^0 / K^0_S / D / B [built up from combinations]

- Data members of the class are **common to all particle types**: mass, momentum, position, PDG code, …

- A group of all particles and anti-particles that belong together logically.
  - e.g. K^{*0} s (decaying to $K^{\pm}$ and $\pi^{\mp}$ with invariant mass in a certain window)
- Can only store particles of the same PDG code (can be different decay modes).
- **ParticleList** is the **dataobject** on which analysis modules operate.

A simple example of MC production

There are two possible ways to fill a list of stable particles

1. Fill it by yourself

```
trackQuality = 'thetaInCDCAcceptance and nCDCHits>20'
ipCut = 'dr < 0.5 and abs(dz) < 2'
goodTrack = trackQuality + ' and ' + ipCut
ma.fillParticleList('pi+:my', goodTrack + ' and binaryPID(211,321)>0.6', path=my_path)
ma.fillParticleList('K+:my', goodTrack + ' and binaryPID(211,321)<0.4', path=my_path)
```

A simple example of MC production

There are two possible ways to fill a list of stable particles

1. Fill it by yourself

```
trackQuality = 'thetaInCDCAcceptance and nCDCHits>20'
ipCut = 'dr < 0.5 and abs(dz) < 2'
goodTrack = trackQuality + ' and ' + ipCut
ma.fillParticleList('pi+:my', goodTrack + ' and binaryPID(211,321)>0.6', path=my_path)
ma.fillParticleList('K+:my', goodTrack + ' and binaryPID(211,321)<0.4', path=my_path)
```

particle:label

whole name of a particle list

A simple example of MC production

There are two possible ways to fill a list of stable particles

1. Fill it by yourself

```
trackQuality = 'thetaInCDCAcceptance and nCDCHits>20'
ipCut = 'dr < 0.5 and abs(dz) < 2'
goodTrack = trackQuality + ' and ' + ipCut
ma.fillParticleList('pi+:my', goodTrack + ' and binaryPID(211,321)>0.6', path=my_path)
ma.fillParticleList('K+:my', goodTrack + ' and binaryPID(211,321)<0.4', path=my_path)
```

particle:label

whole name of a particle list

2. Use standard tracks

```
import stdCharged as stdc
stdc.stdK(listtype='loose', path=my_path)
stdc.stdPi(listtype='loose', path=my_path)
from stdPi0s import stdPi0s
stdPi0s(listtype = 'eff60_May2020', path=my_path)
```

Reconstruct your decay

```
ma.reconstructDecay(decayString='D0:Kpi -> K-:loose pi+:loose',
                    cut='1.84 < M < 1.89',
                    dmID=1,
                    path=my_path)

ma.reconstructDecay(decayString='D0:Kpipi0 -> K-:loose pi+:loose pi0:eff60_May2020',
                    cut='1.82 < M < 1.91',
                    dmID=2,
                    path=my_path)

copyLists('D0:sig', ['D0:Kpi', 'D0:Kpipi0'], path=my_path)

ma.reconstructDecay(decayString='D*+:sig -> pi+:loose D0:sig',cut='1.8<M<2.2',path=my_path)
```

Reconstruct your decay

```
ma.reconstructDecay(decayString='D0:Kpi -> K-:loose pi+:loose',
                    cut='1.84 < M < 1.89',
                    dmID=1,
                    path=my_path)

ma.reconstructDecay(decayString='D0:Kpipi0 -> K-:loose pi+:loose pi0:eff60_May2020',
                    cut='1.82 < M < 1.91',
                    dmID=2,
                    path=my_path)

copyLists('D0:sig', ['D0:Kpi', 'D0:Kpipi0'], path=my_path)

ma.reconstructDecay(decayString='D*+:sig -> pi+:loose D0:sig',cut='1.8<M<2.2',path=my_path)
```

***dmID is short for “decay mode ID”

You don’t have to call
reconstructDecay(‘decayString='D*-:sig -> pi-:loose anti-D0:sig’,cut='1.8<M<2.2',path=my_path)

Reconstruct your decay

modularAnalysis.reconstructDecay(*decayString, cut, dmID=0, writeOut=False, path=None, candidate_limit=None, ignoreIfTooManyCandidates=True, chargeConjugation=True, allowChargeViolation=False*)
[source]

Creates new Particles by making combinations of existing Particles - it reconstructs unstable particles via their specified decay mode, e.g. in form of a DecayString: `D0 -> K- pi+` or `B+ -> anti-D0 pi+`, ... All possible combinations are created (particles are used only once per candidate) and combinations that pass the specified selection criteria are saved to a newly created (mother) ParticleList. By default the charge conjugated decay is reconstructed as well (meaning that the charge conjugated mother list is created as well) but this can be deactivated.

One can use an `@`-sign to mark a particle as unspecified for inclusive analyses, e.g. in a DecayString: `'@Xsd -> K+ pi-'`.

More details on sphinx

Do some fit

Here I choose TreeFit

```
vertex.treeFit(list_name='D*+:sig',
               conf_level=0,  # 0:keep only fit survivors, -1: keep all candidates;
               ipConstraint=False,
               updateAllDaughters=True,  # update momenta of ALL particles
               massConstraint=['pi0'],  # mass constrain ALL pi0
               path=my_path)
```

Of course there are also many fitter in basf2:

- Use cases
- RAVE
- KFit
- OrcaKinFit
- TagV

Save the variables

The VariableManager

It manages variables

- VariableManager is a place in the analysis package to store variables
  - physics quantities: invariant mass, beam-constrained mass, E, p, p_T, θ, ϕ, highest energy in a cluster
  - counters: event_number, nhits, i_candidate

- Every variable takes at least a `Particle*` as input and returns a `double`
  ~~(even integer counters like event_number where this doesn't make much sense)~~

Save the variables

The VariableManager

It manages variables

- VariableManager is a place in the analysis package to store variables
  - physics quantities: invariant mass, beam-constrained mass, E, p, p_T, θ, ϕ, highest energy in a cluster
  - counters: event_number, nhits, i_candidate

- Every variable takes at least a `Particle*` as input and returns a `double`
  ~~(even integer counters like event_number where this doesn't make much sense)~~

```
ma.reconstructDecay(decayString='D0:Kpipi0 -> K-:loose pi+:loose pi0:eff60_May2020',
                    cut='1.82 < M < 1.91',
                    dmID=2,
                    path=my_path)
```

To get the variable list, a simple command:

```
$ b2help-variables
```

Or you can get the information on software.belle2.org :

Chapter 6.3 Variables

There are more than 100 variables…

Can use `daughter(i-th)` to obtain the variables of the i-th daughter's.

We can also use `formula` to calculate the variables that you need.

Alias

Name in *VariableManager* is too long

For example, the decay mode ID of D^0.

We need to call:

```
'extraInfo(decayModeID)'
```

It would be awful to use this especially in the ***output.root***

But with alias, we can easily replace this with:

```
variables.addAlias('modeID', 'extraInfo(decayModeID)')
```

Variable collections + alias

We can do better!

- In analysis/scripts/variables/collections.py, some variables are put together as a collection:

```
#: Replacement for DeltaEMbc
deltae_mbc = ["Mbc", "deltaE"]

#: Replacement to Kinematics tool
kinematics = ['px', 'py', 'pz', 'pt', 'p', ‘E’]
```

- And we can use `create_aliases` to together with the collections:

```
d0kinematics = vu.create_aliases(vc.kinematics, ‘daughter(0,{variable})','d0')
```

In output.root it would be shown like: d0_px, d0_py, d0_pz, ⋯

- Custom collection is also fine, they are just python list!

```
extra_vars = ['xp','chiProb']
trgs = ['ffy','hie','c4']
for trg in trgs:
    variables.addAlias(f'is_{trg}', f'L1PSNM({trg})' )
    extra_vars.append( f'is_{trg}' )
```

```python
variables.addAlias('modeID', 'extraInfo(decayModeID)')

d0kinematics = vu.create_aliases(vc.kinematics, 'daughter(0,{variable})','d0')
d1kinematics = vu.create_aliases(vc.kinematics, 'daughter(1,{variable})','d1')
d2kinematics = vu.create_aliases(vc.kinematics, 'daughter(2,{variable})','d2')

d0kinematicsMotherCMS = vu.create_aliases(vc.kinematics, 'useRestFrame(daughter(0,{variable}))','MRF_d0')
d1kinematicsMotherCMS = vu.create_aliases(vc.kinematics, 'useRestFrame(daughter(1,{variable}))','MRF_d1')
d2kinematicsMotherCMS = vu.create_aliases(vc.kinematics, 'useRestFrame(daughter(2,{variable}))','MRF_d2')

variables.addAlias('m12','daughterInvM(1,2)')
variables.addAlias('m01','daughterInvM(0,1)')
variables.addAlias('m02','daughterInvM(0,2)')
daughterM = ['m12',
             'm01',
             'm02',
]

extra_vars = ['xp','chiProb']
trgs = ['ffy','hie','c4']
for trg in trgs:
    variables.addAlias(f'is_{trg}', f'L1PSNM({trg})' )
    extra_vars.append( f'is_{trg}' )
```

Particles, variables

Variables for a selected kid particle:

```
import variables.utils as vu
spi_vars = vu.create_aliases_for_selected(list_of_variables= vc.kinematics + vc.mc_truth ,
                                          decay_string='D*+ -> ^pi+ D0 ')
```

Use the 'carat' to specify the particle you are choosing
Order and decay chain should be same as in `reconstrucDecay`

Even for grandkid particle:

```
pi0_vars = vu.create_aliases_for_selected(list_of_variables= vc.inv_mass + vc.kinematics + vc.mc_truth ,
                                          decay_string='D*+ -> pi+ [D0 -> K- pi+ ^pi0] ')
```

Finally!

```
#ADD all variables together:

Dsp_vars =  D0_vars + spi_vars + pi0_vars + vc.inv_mass + vc.kinematics + vc.mc_truth +
extra_vars

#output
ma.variablesToNtuple(decayString='D*+:sig',
                     variables=Dsp_vars,
                     filename=OutputFile,
                     treename=OutputTree,
                     path=my_path)

# Process the events
b2.process(my_path)
```

/home/belle/yinjh/public/sFPCP/tutorial

Directly to histogram?

VariablesToHistogram

```
# ... at the end of your script
from modularAnalysis import variablesToHistogram
variablesToHistogram('K*0:myKst',
                     [('M', 100, 0.7, 1.0),
                      ('p', 100, 0.1, 3.0)], path=mypath)

# process the events
basf2.process(mypath)
```

How to find the module I need?

- All the modules are in `<package>/modules/`.
- In most cases, you don't need to call them directly.
- In `<package>/scripts`, there are well prepared functions to be called. Just like the `stardardCharge`. For example: in `generators/scripts/generators.py`:

```
def add_evtgen_generator(path, finalstate='', signaldecfile=None, coherentMixing=True, parentParticle='Upsilon(4S)'):
    """
    Add EvtGen for mixed and charged BB

    Parameters:
        path (basf2.Path): path where the generator should be added
        finalstate (str): Either "charged" for generation of generic B+/B-, "mixed" for generic B0/anti-B0, or "signal" for
                          generation of user-defined signal mode
        signaldecfile (str): path to decfile which defines the signal decay to be generated
                             (only needed if ``finalstate`` set to "signal")
        coherentMixing: Either True or False. Switches on or off the coherent decay of the B0-B0bar pair.
                        It should always be True, unless you are generating Y(5,6S) -> BBar. In the latter case,
                        setting it False solves the internal limitation of Evtgen that allows to make a
                        coherent decay only starting from the Y(4S).
        parentParticle (str): initial state (used only if it is not Upsilon(4S).
    """
```

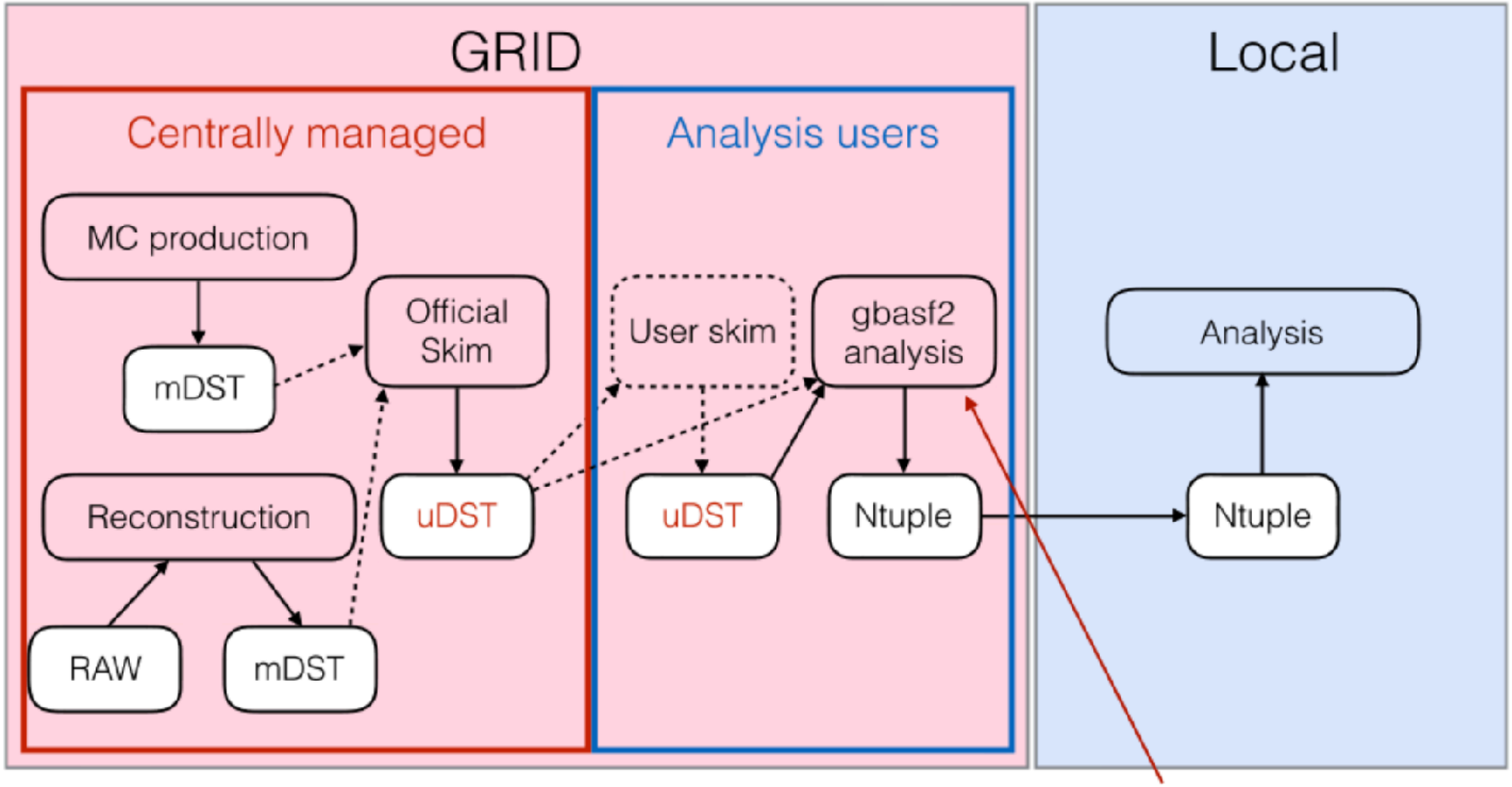
How to find the module I need?

- All the modules are in `<package>/modules/`.
- In most cases, you don’t need to call them directly.
- In `<package>/scripts`, there are well prepared functions to be called. Just like the `stardardCharge`.
- You can also find good instruction on sphinx!

How to find the module I need?

- All the modules are in `<package>/modules/`.
- In most cases, you don't need to call them directly.
- In `<package>/scripts`, there are well prepared functions to be called. Just like the `stardardCharge`.
- You can also find good instruction on sphinx!
- If you failed to figure it by yourself, don't hesitate to ask on questions!
- Or our QQ/Wechat channel!

The Belle II analysis model



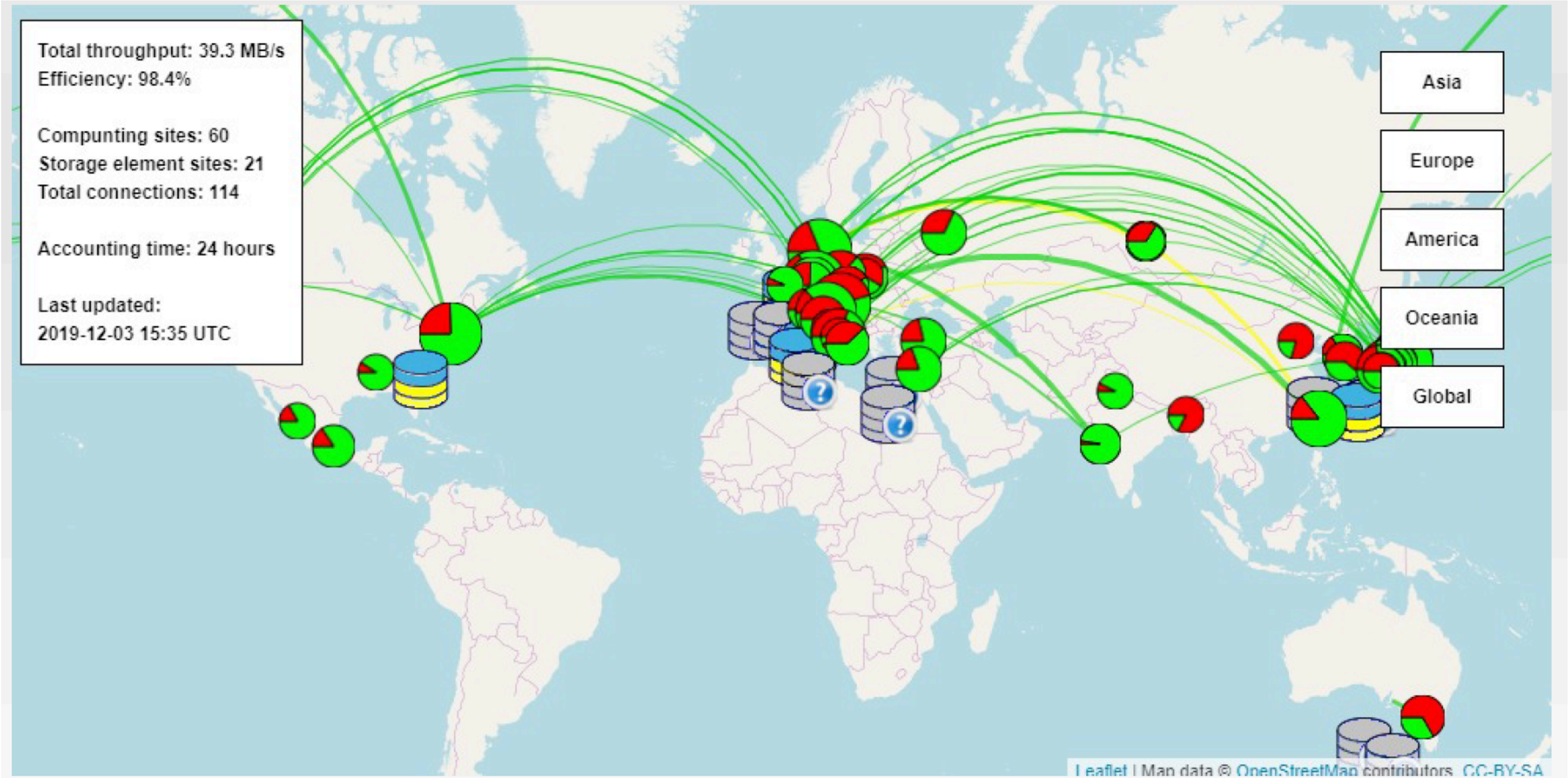
basf2 on the ***Grid***

Grid

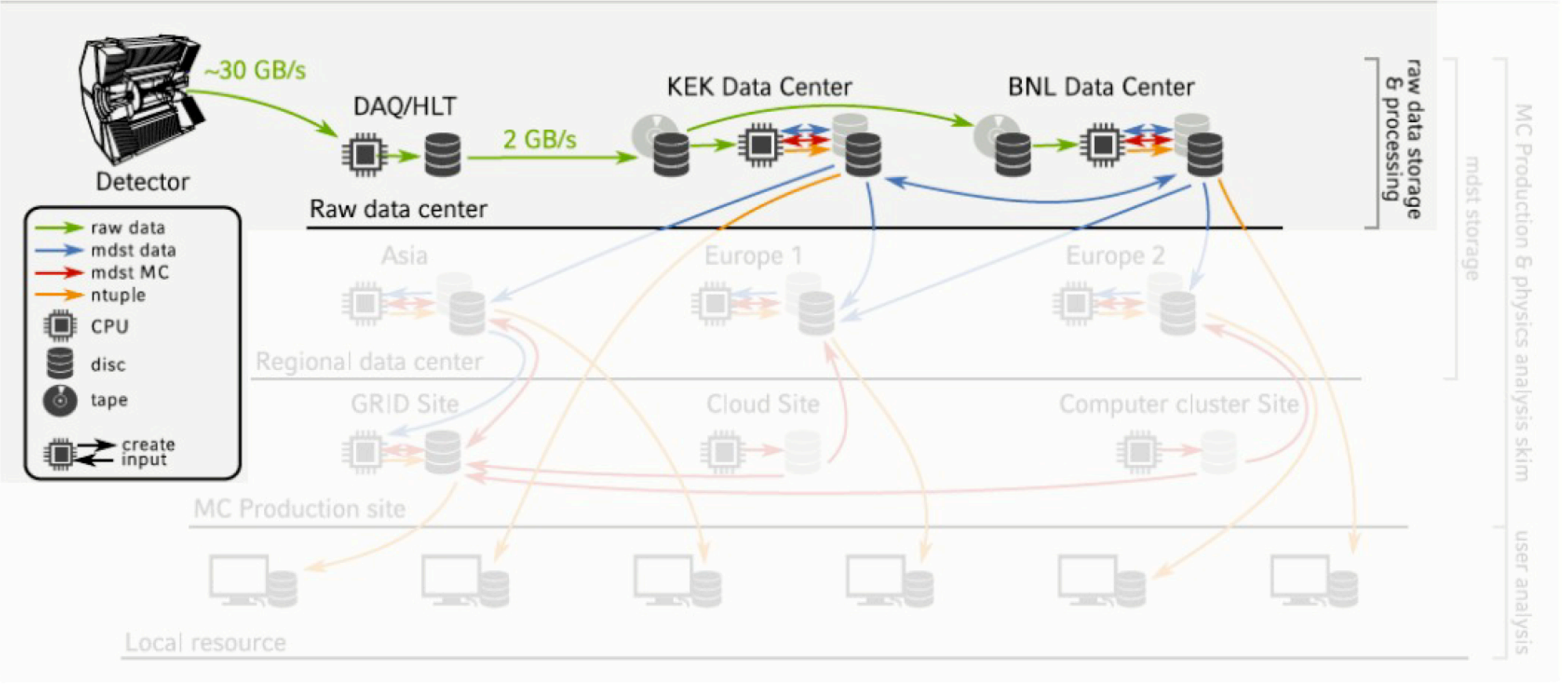
- The grid is a distributed computing system utilised by Belle II, and many other particle physics experiments, to make use of the computing resources of the many universities and institutions worldwide (that are involved inparticle physics research).

- Modern particle physics experiments (will) collect many tens of petabytes of data ...
  - … and even more tens of petabytes of MC.
- Processing all data/MC at a single site, even a national laboratory, is not longer a sustainable model.
- Heavily used by the LHC and other experiments.

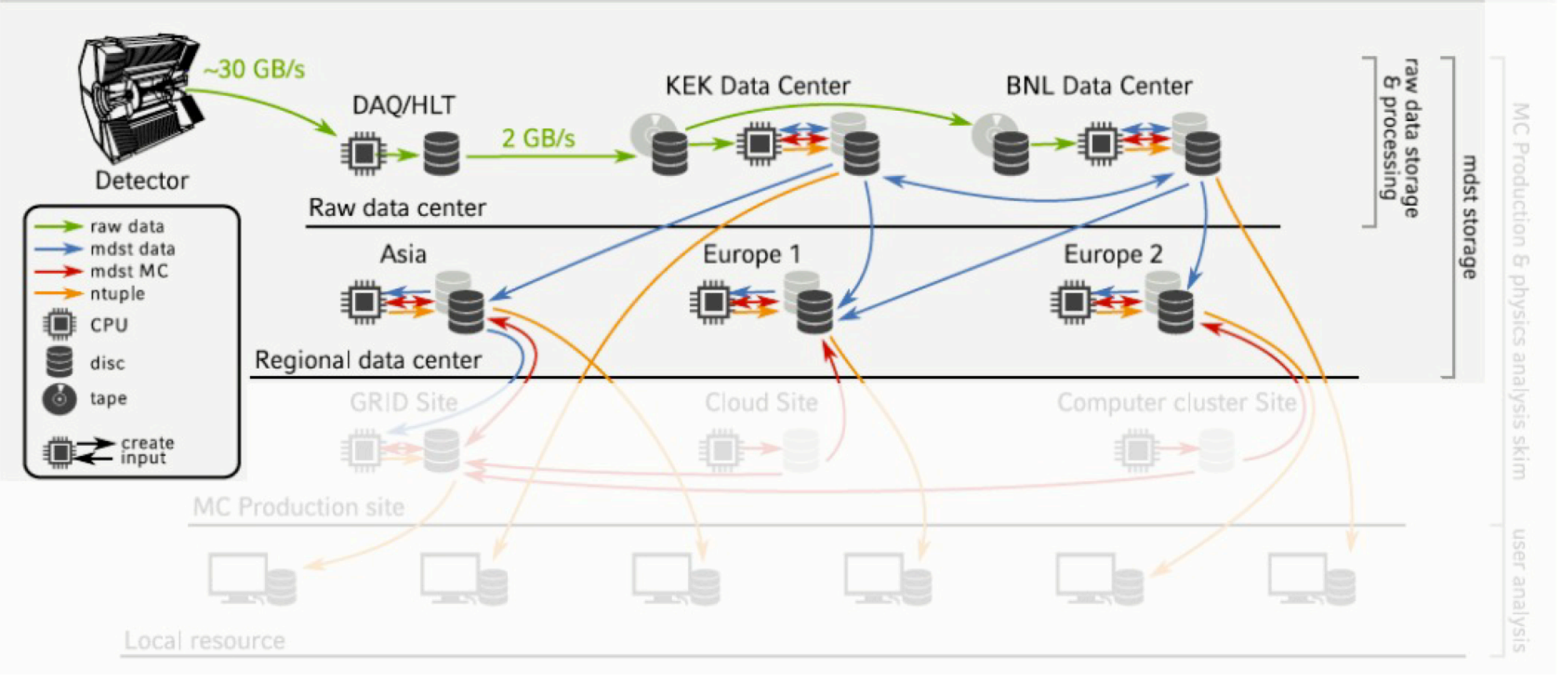
Belle II Grid



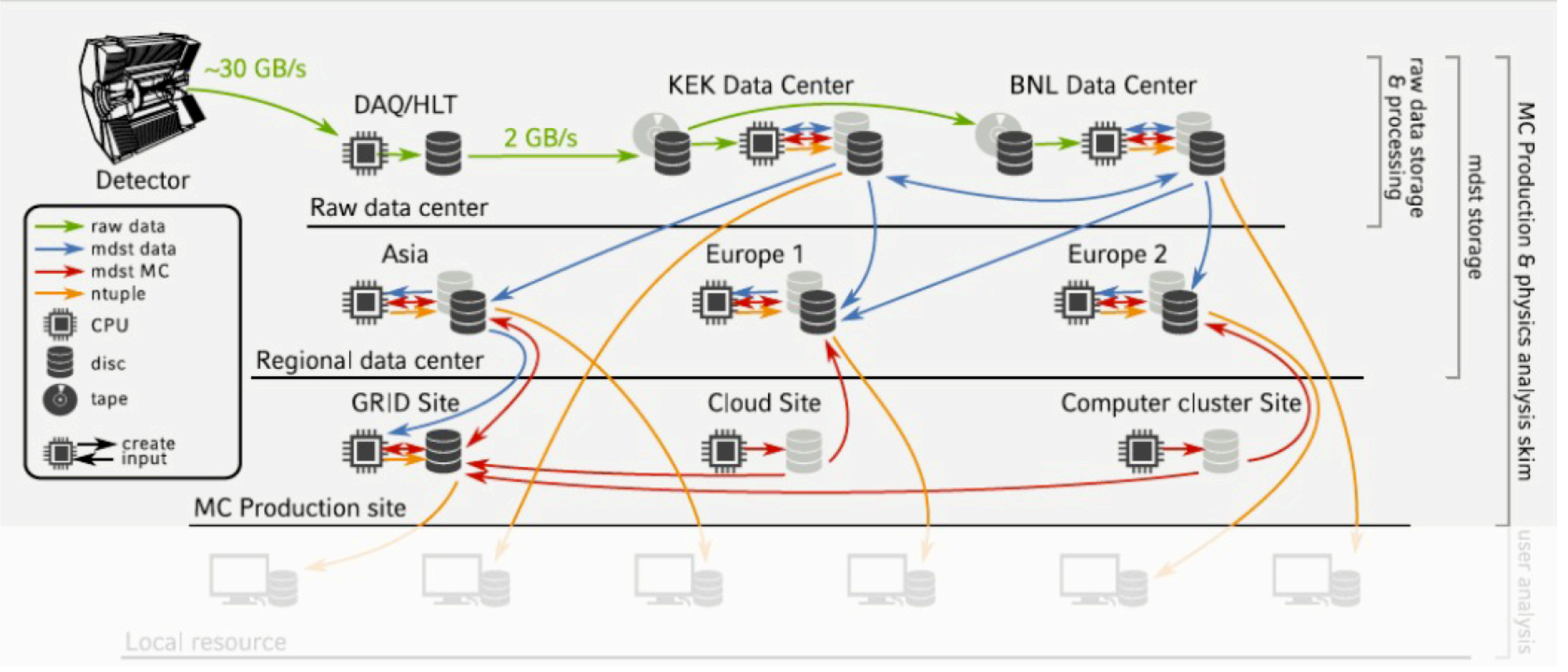
- Raw data storage and reprocessing at KEKCC and BNL - Raw data centers
- mDST storage on GRID storage elements (SE) - Regional data centers
- Skimming and analysis on GRID computing elements (CE) - MC production sites
- nTuple analysis on local resources - Local resources



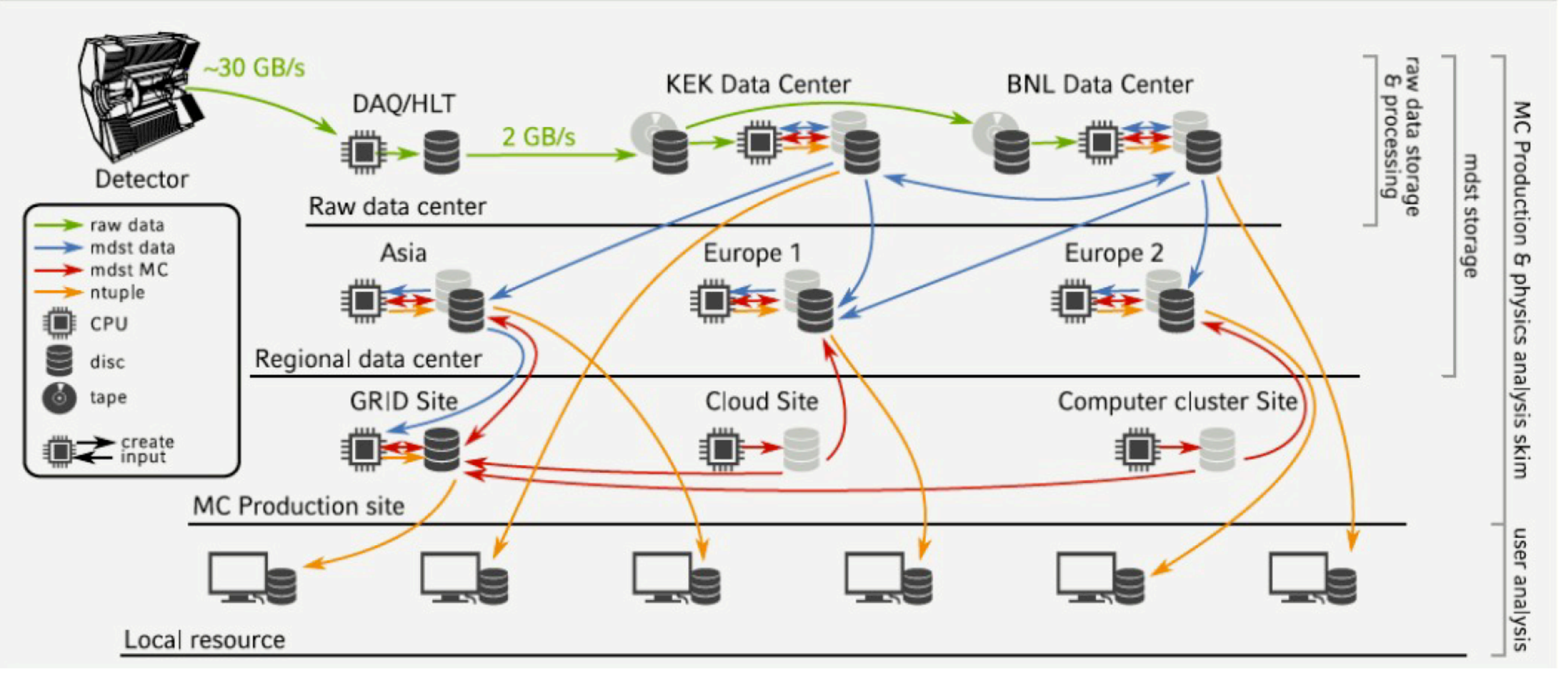
- Raw data storage and reprocessing at KEKCC and BNL - Raw data centers
- mDST storage on GRID storage elements (SE) - Regional data centers
- Skimming and analysis on GRID computing elements (CE) - MC production sites
- nTuple analysis on local resources - Local resources



- Raw data storage and reprocessing at KEKCC and BNL - Raw data centers
- mDST storage on GRID storage elements (SE) - Regional data centers
- Skimming and analysis on GRID computing elements (CE) - MC production sites
- nTuple analysis on local resources - Local resources



- Raw data storage and reprocessing at KEKCC and BNL - Raw data centers
- mDST storage on GRID storage elements (SE) - Regional data centers
- Skimming and analysis on GRID computing elements (CE) - MC production sites
- nTuple analysis on local resources - Local resources



To run an analysis job with gbasf2 in the same way as you would with basf2 in a terminal.

If you run basf2:

```
basf2 myAnalysisScript.py
```

You would like to run:

```
gbasf2 myAnalysisScript.py
```

It is almost this simple!

You need to specify three additional options: a basf2 release, a project name, and the input dataset.

```
gbasf2 myAnalysisScript.py -s basf2-release -p myProject -i
InputDataSet/PathOfInputDataSet/CollectionOfInputDataSet
```

The gbasf2 code is independent of the basf2 release.

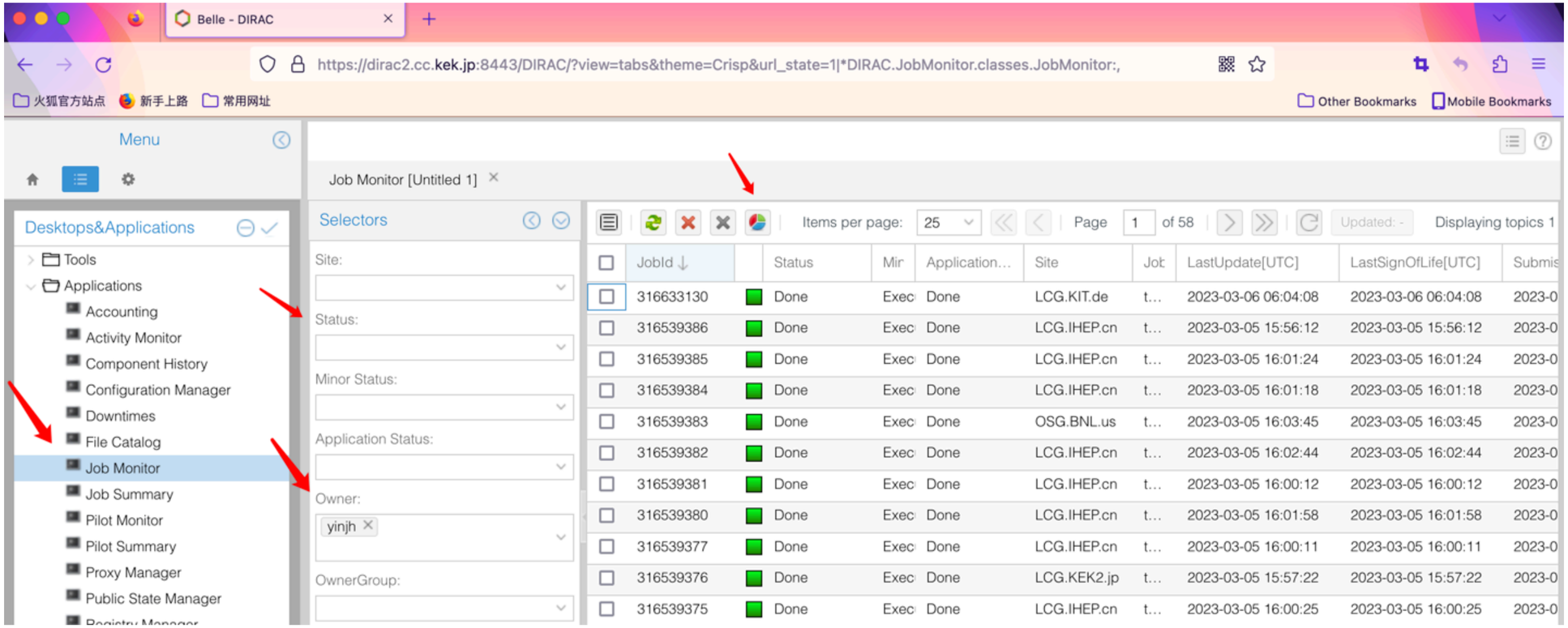
Get more details on the instruction or the confluence page.

Check the job status

After all preparations (keys, certificates, etc…), surf to

https://dirac2.cc.kek.jp:8443/DIRAC/

with the browser where your certificate installed.

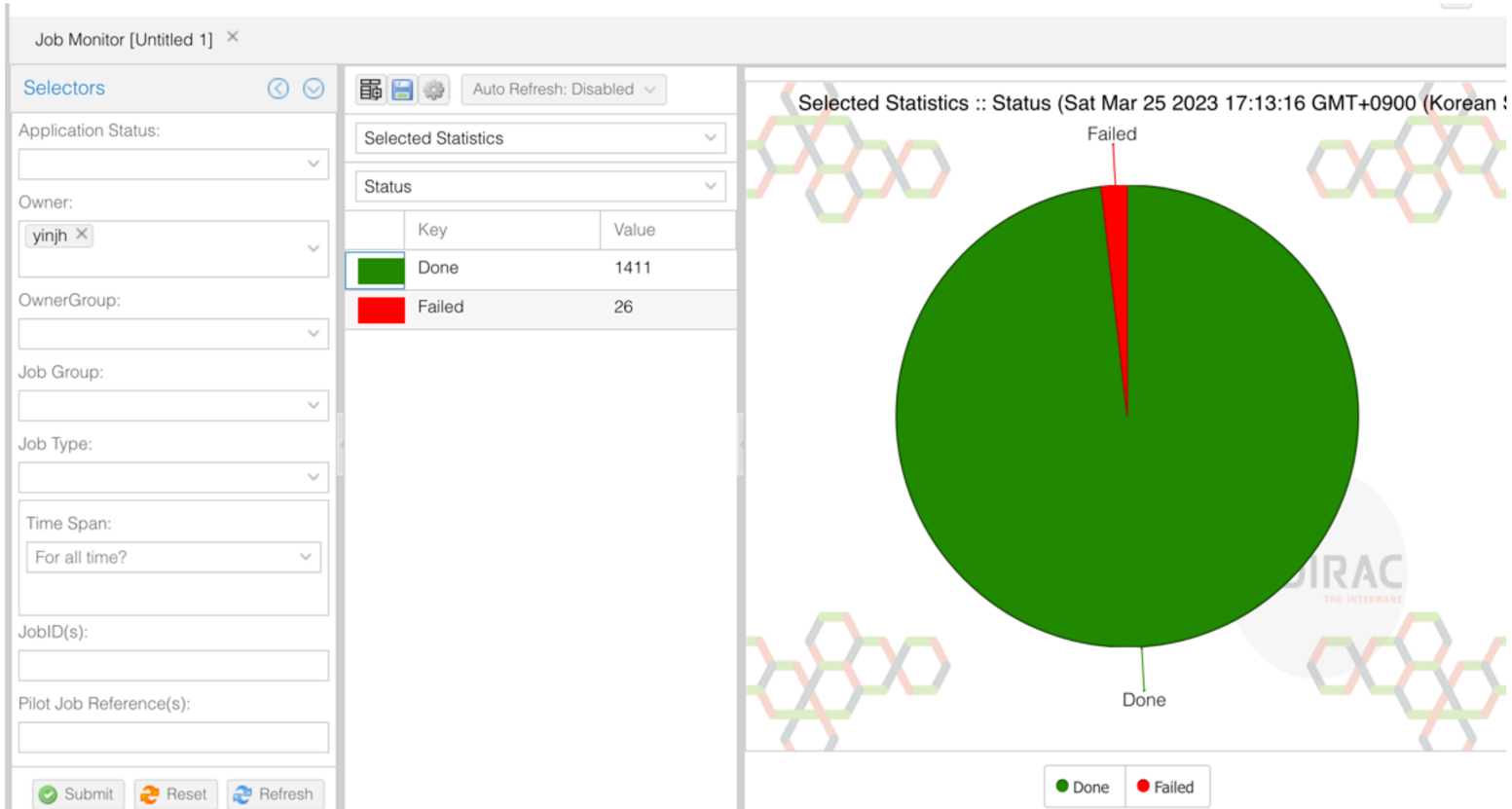


Check the job status

After all preparations (keys, certificates, etc…), surf to

https://dirac2.cc.kek.jp:8443/DIRAC/

with the browser where your certificate installed.



Check the job status

After all preparations (keys, certificates, etc…), surf to

https://dirac2.cc.kek.jp:8443/DIRAC/

with the browser where your certificate installed.

Of course you can also check this on kekcc with a simple command:

```
gb2_job_status -j [job ID] -p [project name]
```

How to find dataset

1. Find the information on confluence,

How to find dataset

1. Find the information on confluence,

MC14rd_c	/belle/MC/release-05-02-14/DB00001457/MC14rd_c	~0.25 per fb of 2021 data (25-35 ?)	35.1	MCProduction	BIIDCO-3648 - MC14rd_a Data Production Campaign IN REVIEW BIIDCO-3723 - MC14rd_c Data Production Campaign IN REVIEW	2021-07-29 - 2022-02-19	14th run dependent MC campaign: corresponding to prompt	Done 2022-07-23
MC14rd_d	/belle/MC/release-05-02-14/DB00001457/MC14rd_d	~23.4 google doc		MCProduction	BIIDCO-3649 - MC14rd_b Data Production Campaign IN PROGRESS BIIDCO-3724 - MC14rd_d Data Production Campaign IN REVIEW	2021-07-29 -	14th run dependent MC campaign: corresponding to proc12	Done 2022-07-23

How to find dataset

1. Find the information on confluence,

Or

2. Use the dataset searcher

How to find dataset

1. Find the information on confluence,

Or

2. Use the dataset searcher

```
gb2_ds_search dataset --data_type Data --campaign bucket9  --general_skim hadron  --data_level mdst
```

If you don’t like the command…

Menu

Desktops&Applications

- Component History
- Configuration Manager
- Downtimes
- File Catalog
- Job Monitor
- Job Summary
- Pilot Monitor
- Pilot Summary
- Proxy Manager
- Public State Manager
- Registry Manager
- Request Monitor
- Resource Summary
- Space Occupancy
- System Administration
- Transformation Monitor
- BelleDIRAC Apps
  - Dataset Searcher
  - B2Monitoring Plot Display
  - B2Monitoring Plot Display User
- Help
- DIRAC

Settings

Job Monitor [Untitled 1] | Dataset Searcher [Untitled 2]

Dataset Searcher

Metadata Searcher | Tree Browser

Data Type: MC / Data

Background level: BGx1 / BGx0 / Other

Background level:		Campaigns:	proc13
Beam Energies:	4S, 4S_scan, 4S_offres	Skim Types:	
Data Levels:		Releases:	
Global Tags:		Experiment Low:	
Experiment High:		Run Low:	
Run High:		MC Event Types:	
General Skim Names:	hadron		

Clear | Search | Help

LPN

/belle/Data/release-06-00-12/DB00002392/proc13/prod00025327/e0007/4S/r03221/hadron/mdst

/belle/Data/release-06-00-12/DB00002392/proc13/prod00025327/e0007/4S/r03231/hadron/mdst

Dataset LFNs Metadata | Dataset Metadata | Download .txt file

Default

skim

Make life easier

The skim package is a collection of high-level analysis scripts that reduce the data set to a manageable size by applying a simple selection. The input to a skim are Belle II File Format files of processed data. The output are so-called User-defined DST Output (udst) files. These files actually contain more information but fewer events.

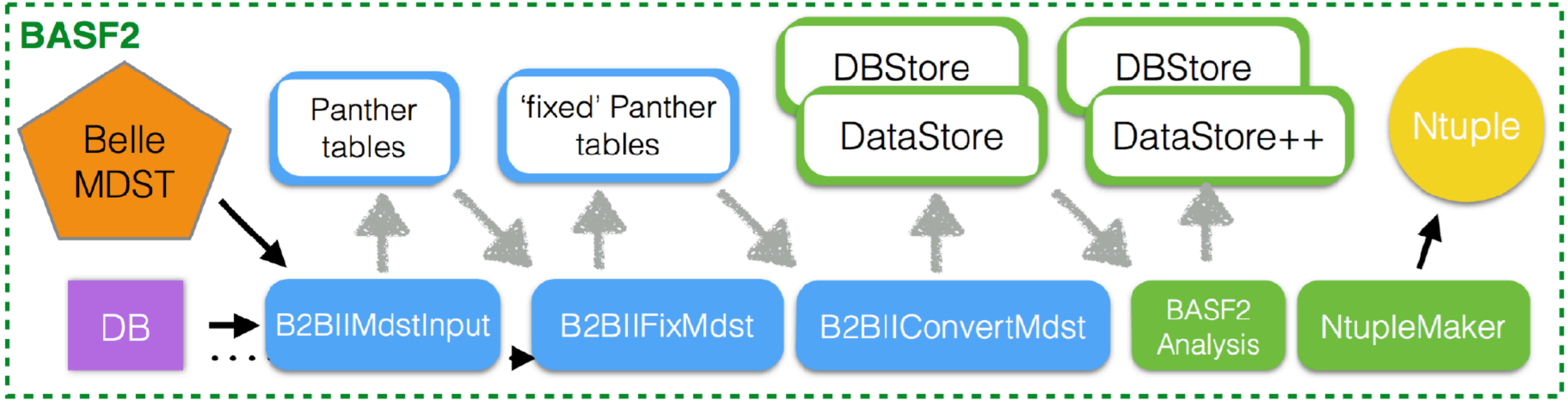
Charm Liaison: @Kaikai He Prep: BIIDP-5742 Requests: BIIDP-6064	CharmAll	DstToDpPi0_DpToHpPi0	17241000	All	Ready	Ready	Ready	Running
		DstToD0Pi_D0ToNeutrals	17240600					
		XToD0_D0ToNeutrals	17230200					
	EWPetal	DstToD0Pi_D0ToVGamma	17241200	All	Ready	Running* *Ready except a few problem runs in exp7 and 10. Resubmitting and they will finish by March 17.	Ready	Running
	CharmHad	DstToD0Pi_D0ToHpJm	17240100	Hadron	Ready	Ready	Ready	Running
		DstToD0Pi_D0ToHpHmPi0	17240300					
		DstToD0Pi_D0ToHpJmKs	17240700					
		LambdacTopHpJm	17230600					
		XToDp_DpToKsHp	17230400					
		XToD0_D0ToHpJm	17230100					
		XToDp_DpToHpHmJp	17230500					

skim ID. Will be used in dataset searcher

B2BII

Belle to Belle II

✦ Read and analyze Belle MDST format in basf2.

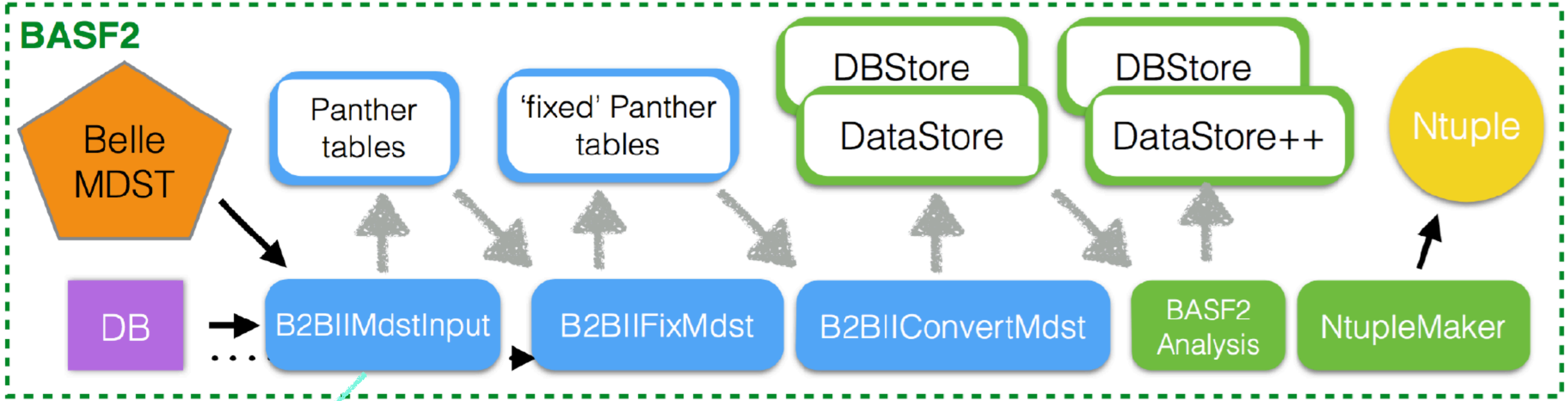


no intermediate file conversion from PANTHER to ROOT

B2BII

Belle to Belle II

✦ Read and analyze Belle MDST format in basf2.



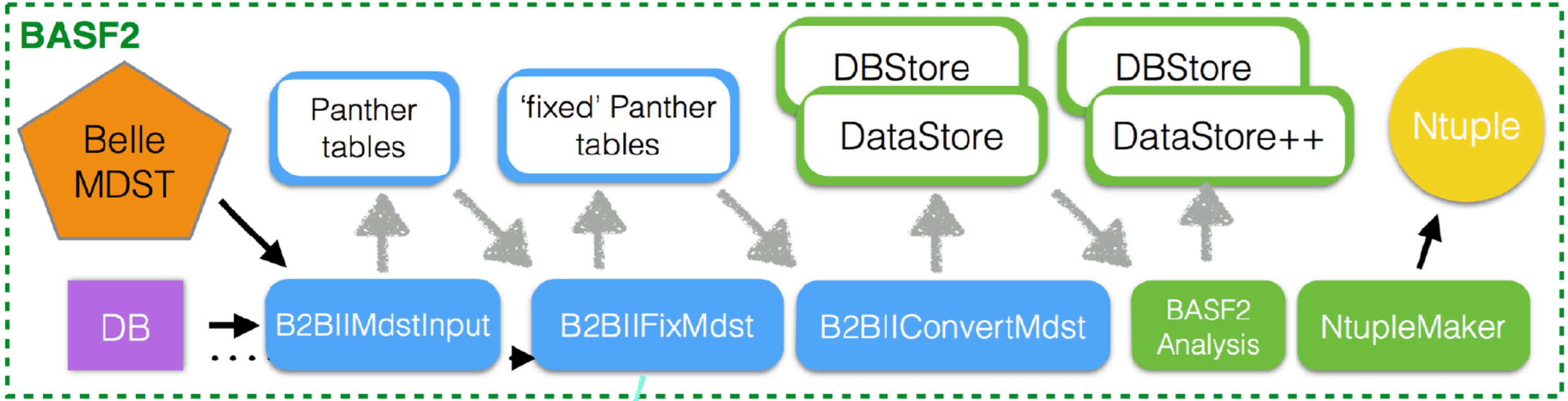
no intermediate file conversion from PANTHER to ROOT

Reads Belle mdst file

B2BII

Belle to Belle II

✦ Read and analyze Belle MDST format in basf2.



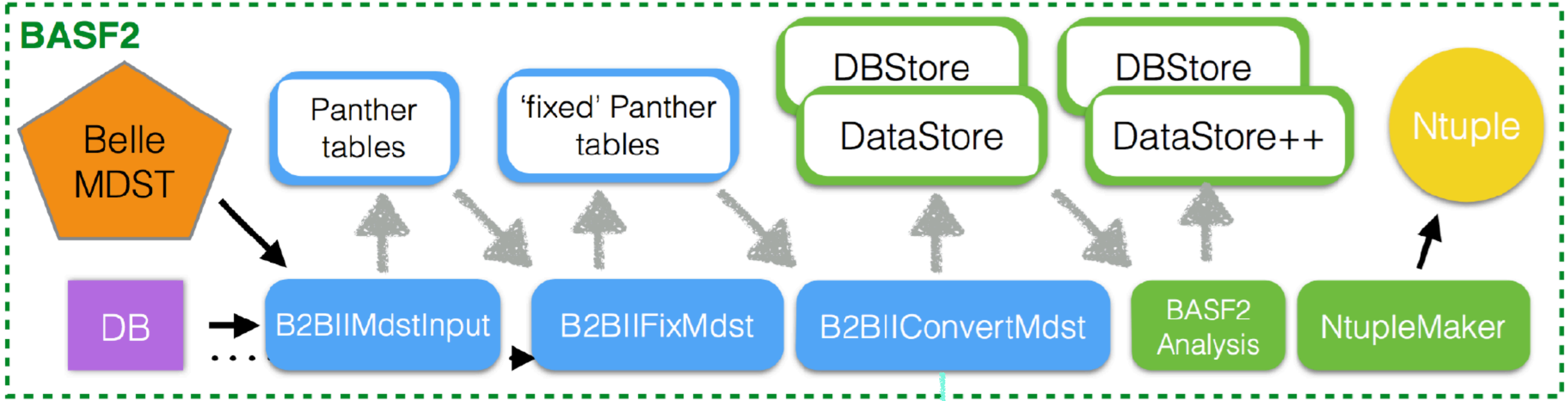
no intermediate file conversion from PANTHER to ROOT

Perform corrections to the Belle mdst data objects ("fix mdst")
Apply HadronB(J) skim by default.

B2BII

Belle to Belle II

✦ Read and analyze Belle MDST format in basf2.



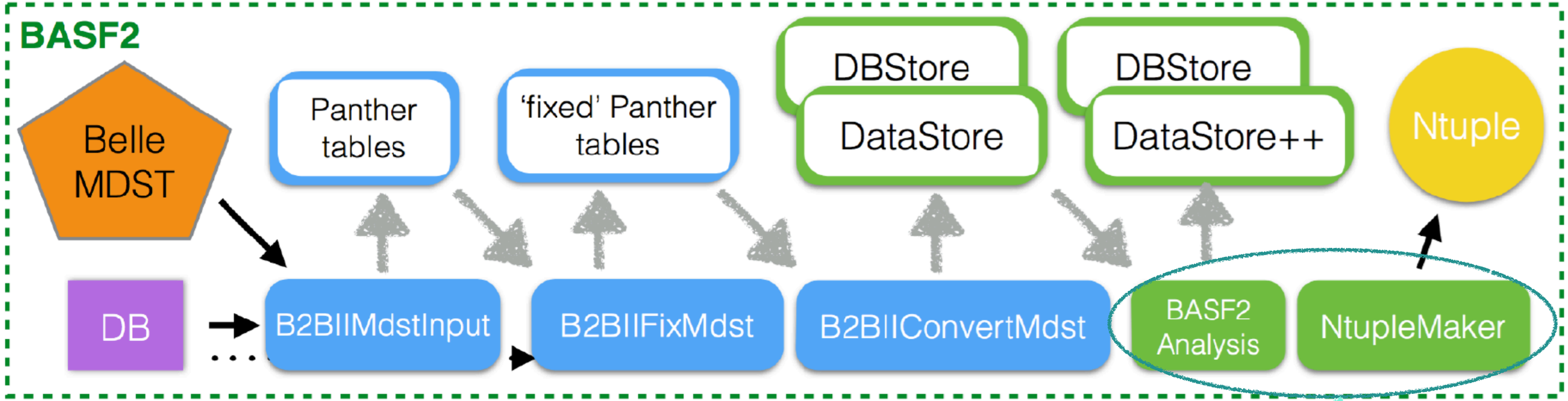
no intermediate file conversion from PANTHER to ROOT

Performs Belle mdst to Belle II mdst conversion

B2BII

Belle to Belle II

✦ Read and analyze Belle MDST format in basf2.



no intermediate file conversion from PANTHER to ROOT

All basf2 modules will work now!

Conversion in B2BII

Belle | Belle II

MC Information
- Gen_hepevt
- MCParticle

Tracking output
- Mdst_charged
- Mdst_trk
- Mdst_trk_fit
- Track
- TrackFitResult

BeamParameters
- BeamEnergy
- IPProfile
- Collision*
- BeamSpot
- BeamParameters

PID information
- atc_pid/eID/muID
- PIDLikelihood

ECL output
- Mdst_ecl
- Mdst_ecl_aux
- Mdst_gamma
- Mdst_pi0
- ECLCluster

V0
- Mdst_vee2
- Mdst_vee2_daughter
- Mdst_vee2_extra
- V0
- TrackFitResult

Klong
- Mdst_klong
- KLMCluster

Particle list for B2BII analysis

basf	basf2	In basf2 analysis script
Mdst_charged	Track	*Create ParticleLists (w/ or w/o cuts) of Final State Particles:* *fillParticleList('K+:mine','atcPIDBelle(3,2)>0.6 and abs(d0)<2 and abs(z0)<4', path=mypath)*
Mdst_gamma	gamma:mdst ECLCluster	*Use already created 'gamma:mdst' ParticleList:* *cutAndCopyList('gamma:mine,'gamma:mdst','E > 0.1', path=mypath)*
Mdst_pi0	pi0:mdst (daughters link to gamma:mdst)	*Use already created 'pi0:mdst' ParticleList:* *reconstructDecay('rho+:myRho -> pi+:all pi0:mdst', '0.6 < M < 1.0', path=mypath)*
Mdst_vee2	K_S0:mdst Lambda0:mdst gamma:v0mdst	*Use already created 'K_S0:mdst', 'Lambda0:mdst', and 'gamma:v0mdst' ParticleLists:* *cutAndCopyList('K_S0:good', 'K_S0:mdst', cut='extraInfo(goodKs)==1', path=mypath)*
Mdst_klong	K_L0:mdst	*Use already created 'K_L0:mdst' ParticleList:* *reconstructDecay('B0:jpsikl-> J/psi:my K_L0:mdst', 'Mbc>5.0', path=mypath)*

How to use B2BII?

- **Just one line in your script:**

```
convertBelleMdstToBelleIIMdst(inputBelleMDSTFile, path=my_path)
```

How to use B2BII?

- **Just one line in your script:**

```
convertBelleMdstToBelleIIMdst(inputBelleMDSTFile, path=my_path)
```

- **Or maybe two...**

```
from b2biiConversion import convertBelleMdstToBelleIIMdst
convertBelleMdstToBelleIIMdst(inputBelleMDSTFile, path=my_path)
```

Generate Belle MC in basf2

In Belle, MC samples are produced in two step:

1. Generate events with generators (phokhara, evtgen, pythia, etc…)

2. Run the gsim (simulation, reconstruction)

By replacing step 1, we can obtain Belle MC in basf2.

The only thing we need to do is to call the module `BelleMCOutput` like this:

```
main.add_module(phokhara)
main.add_module(evtgendecay)
main.add_module("BelleMCOutput", outputFileName=output_mdst)
main.add_module("PrintMCParticles", logLevel=basf2.LogLevel.DEBUG, onlyPrimaries=False)
main.add_module(“Progress")
basf2.process(main)
```

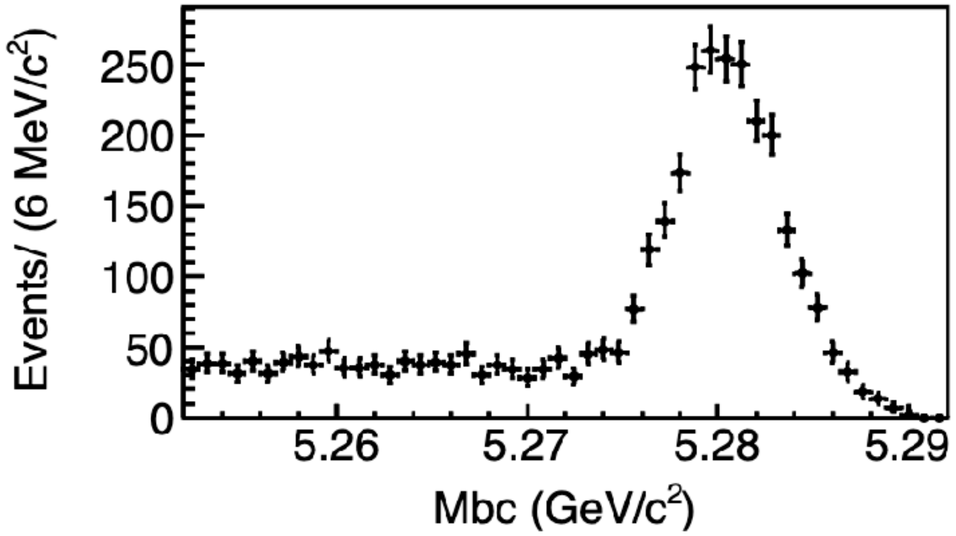
Offline analysis

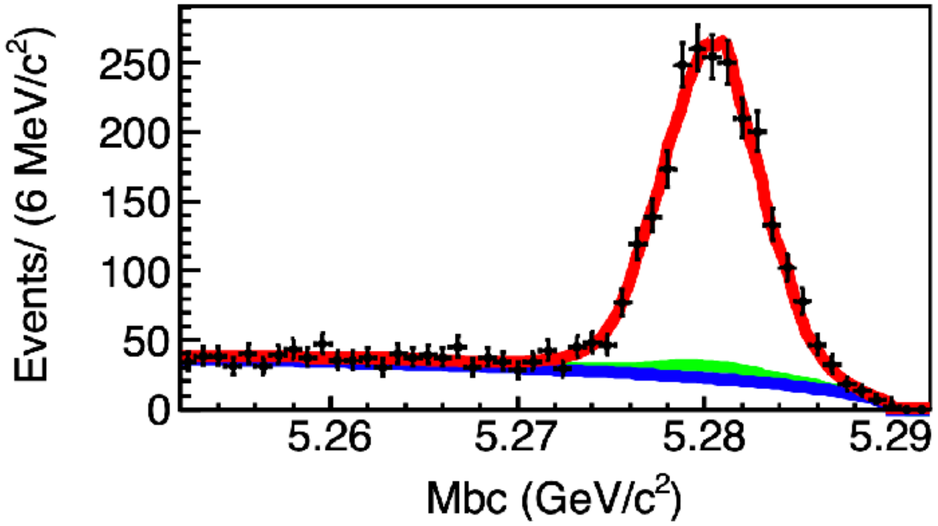
Histograms, graphs, and fit

There are many softwares to handle the dataset, ROOT, pandas, etc…

=> Histograms, graphs. Visualize the variables.

==> Fit. Parameterize variables with theoretical expectation.





RooFit

The RooFit toolkit for data modeling

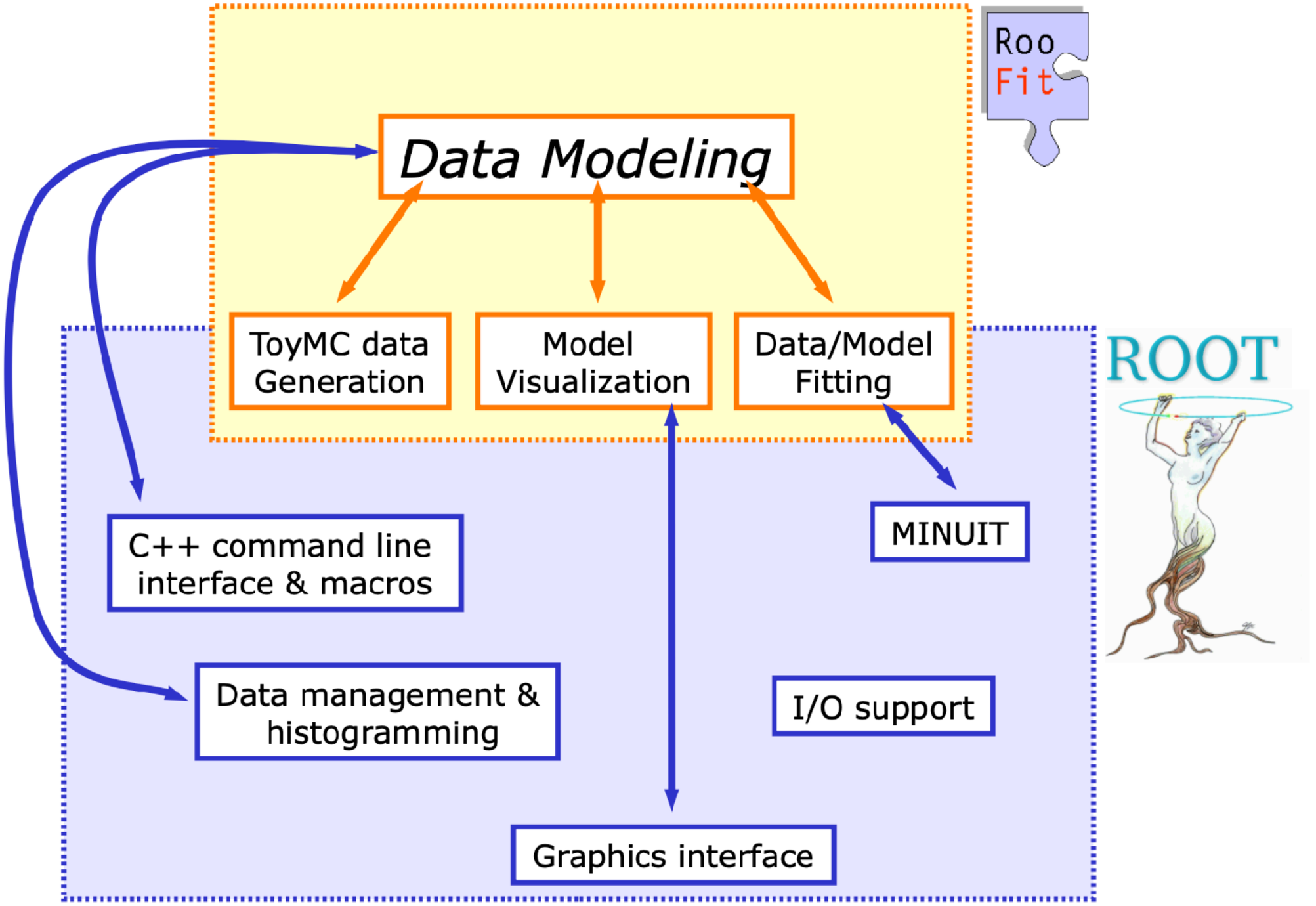
Wouter Verkerke, David Kirkby

RooFit is a library of C++ classes that facilitate data modeling in the ROOT environment. Mathematical concepts such as variables, (probability density) functions and integrals are represented as C++ objects. The package provides a flexible framework for building complex fit models through classes that mimic math operators, and is straightforward to extend. For all constructed models RooFit provides a concise yet powerful interface for fitting (binned and unbinned likelihood, chi^2), plotting and toy Monte Carlo generation as well as sophisticated tools to manage large scale projects. RooFit has matured into an industrial strength tool capable of running the BABAR experiment's most complicated fits and is now available to all users on SourceForge.

Comments: Talk from the 2003 Computing in High Energy and Nuclear Physics (CHEP03), La Jolla, Ca, USA, March 2003, 9 pages, LaTeX, 6 eps figures. PSN MOLT07
Subjects: **Data Analysis, Statistics and Probability (physics.data-an)**
Cite as: arXiv:physics/0306116 **[physics.data-an]**
(or arXiv:physics/0306116v1 **[physics.data-an]** for this version)
https://doi.org/10.48550/arXiv.physics/0306116

Used in many analyses in HEP.

Optimized maximum likelihood fitter + lots of features.


Roo
Fit
Data Modeling
ToyMC data Generation
Model Visualization
Data/Model Fitting
ROOT
C++ command line interface & macros
MINUIT
Data management & histogramming
I/O support
Graphics interface

- Mathematical objects are represented as C++ objects

Mathematical concept		RooFit class
variable	x	`RooRealVar`
function	$f(x)$	`RooAbsReal`
PDF	$f(x)$	`RooAbsPdf`
space point	$x_{\max}$ x	`RooArgSet`
integral	$\int_{x_{\min}} f(x)dx$	`RooRealIntegral`
list of space points		`RooAbsData`
PDF summation	$f_1(x) + f_2(x)$	`RooAddPdf`
PDF product	$f_1(x) \cdot f_2(x)$	`RooProdPdf`

ToyMC for systematic study

ROOT SimpleGenerateToyFit Last Checkpoint: 16 hours ago (autosaved)

Logout Terminal

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Markdown

Example event generation, fitting and ToyMC study

Load the ROOT and fitting modules. Turn on javascipt ROOT for nice interactive plots

In [1]:
```
import ROOT
ROOT.gROOT.ProcessLine(".x $H5CODE/hsfit/LoadFit.C")
%jsroot
```

```
Welcome to JupyROOT 6.16/00
```

Construct a Toy manager for generating initial data set. This would be equivalent to your real data. The argument 1 tells it to only create one data set per bin.

In [2]:
```
toy = ROOT.ToyManager(1)
```

Give an output directory for storing the "data"

In [3]:
```
toy.SetUp().SetOutDir("outSimpleToys/");
toy.SetUp().SetIDBranchName("UID");
```

Declare your fit variable and its range

In [4]:
```
toy.SetUp().LoadVariable("Mmiss[0,10]");
```

Declare your PDF to generate from. Here a Signal is Gaussian with mean 6 (with range 4-7) and width 0.2 (with range 0.0001-3).

LoadSpecies adds this PDF to the total PDF, while the 100 is the typical number of events to generate. Actual number will include Poisson statistics fluctuation.

In [5]:
```
toy.SetUp().FactoryPDF("Gaussian::Signal( Mmiss, Gmean[6,4,7], Gsigma[0.2,0.0001,3] )");
toy.SetUp().LoadSpeciesPDF("Signal",100);
```

The Background BG, is a Chebychev polynomial, which is going to have twice as many (200) events as the signal contribution.

In [6]:
```
toy.SetUp().FactoryPDF("Chebychev::BG(Mmiss,{a0[-0.1,-1,1],a1[0.1,-1,1]})");
toy.SetUp().LoadSpeciesPDF("BG",200);
```

Generate the data!

In [8]:
```
ROOT.Here.Go(toy);
```

Fitting the generated data

A ToyMananger that has been used to generate data can be directly used to create a fitter with the same model. Alternately you can define a completely new fit model here.

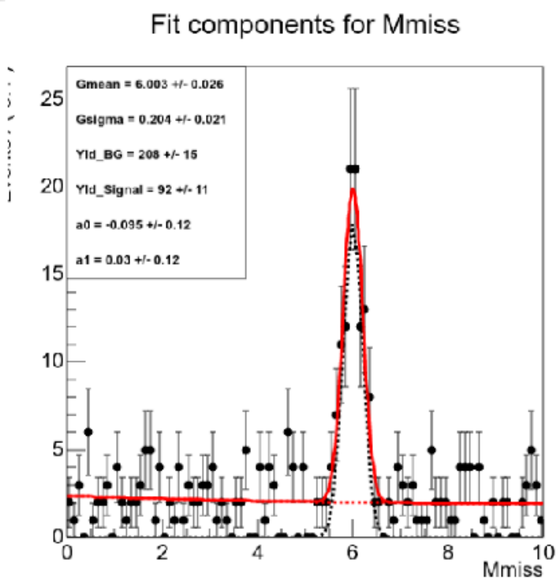
In [9]:
```
fit0=toy.Fitter();
```

```
DataEvents::Load ToyData 1
```

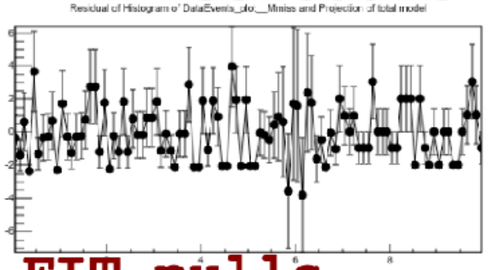
Fit the data on one local CPU.

We should see the minimiser ouput with the final fit plots at the end.

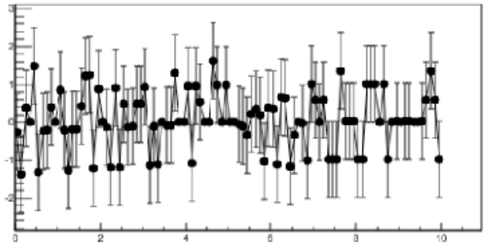
In [10]:
```
ROOT.Here.Go(fit0);
```



FIT residuals



FIT pulls



ToyMC for systematic study

Toy MC study

Now I have successful fit results I want to study the fit for bias etc. I can do this by generating many data sets from my fit results and fitting them to make sure the extracted parameters are consistent each time.

The ToyMC model with the fit results found in the previous cell can be combined into a ToyManager with 1 line of code using the fit model (fit0 here), which also specifies the number of toy datasets (400) to generate. We should then set a new ouput directory and generate the toy datasets.

In [11]:
```
toy2=ROOT.ToyManager.GetFromFit(400,fit0,"ResultsToyBH5Minuit2.root")
toy2.SetUp().SetOutDir("outSimpleToy2");
ROOT.Here.Go(toy2.get());
```

Thre are going to be 400 toy fits so lets run them in parallel with PROOF. The next cell is need to initialise PROOF.

In [12]:
```
from ROOT import TProof
```

Get my fitter from the new ToyManager and run the fits on PROOF with 4 workers.

In [ ]:
```
fit2=toy2.Fitter()
ROOT.Proof.Go(fit2,4)
```

Collect all the fits and create parameter distributions and pulls. This is autimated by the ToyManager Summarise function.

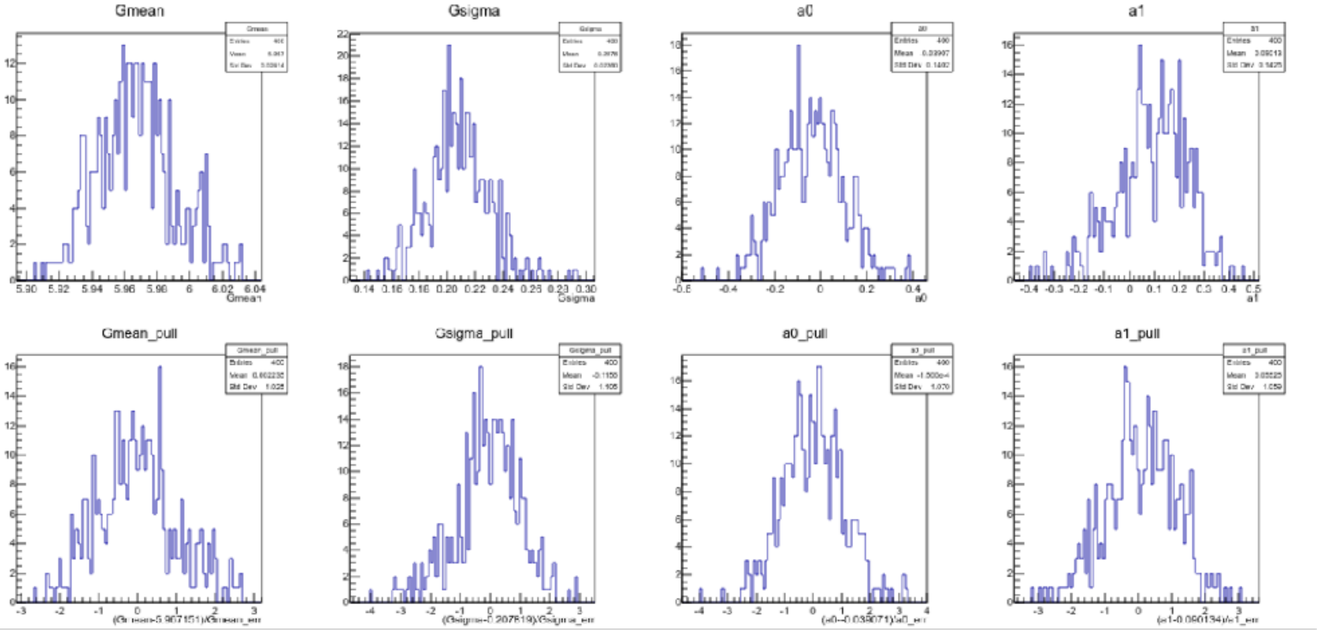
In [14]:
```
toy2.Summarise()
```

```
Summarise ResultTree /work/Dropbox/HaSpect/dev/HASPECT6/tutorials/RooFitExamples/Generators/outSimpleToy2/
 ToyManager::Summarise() Initial Parameters
  1) 0x56492f5c9990 RooRealVar::      Gmean = 5.96906  L(4 - 7)  "Gmean"
  2) 0x56492f5d2120 RooRealVar::     Gsigma = 0.206953  L(0.0001 - 3)  "Gsigma"
  3) 0x56492f5950d0 RooRealVar::         a0 = -0.0433  L(-1 - 1)  "a0"
  4) 0x56492f52e730 RooRealVar::         a1 = 0.0877073  L(-1 - 1)  "a1"
  5) 0x56492f5ca1a0 RooRealVar:: Yld_Signal = 109.655  L(0 - 1e+12)  "Yld_Signal"
  6) 0x56492f5ca9a0 RooRealVar::    Yld_BG = 171.461  L(0 - 1e+12)  "Yld_BG"

Gmean 5.96715 +- 0.0237717 sigma 0.0241361 meanPull 0.00223759 sigmaPull 1.02776
      bias -0.00190544 bias Pull -0.0790702 sigma 1.02767


Gsigma 0.207819 +- 0.0204853 sigma 0.0225982 meanPull -0.115595 sigmaPull 1.1047
      bias 0.00086601 bias Pull -0.0722243 sigma 1.10013


a0 -0.0390713 +- 0.132536 sigma 0.140173 meanPull -0.000150034 sigmaPull 1.06967
      bias 0.00422866 bias Pull 0.0318165 sigma 1.06967


a1 0.0901345 +- 0.132916 sigma 0.142643 meanPull 0.058276 sigmaPull 1.05851
      bias 0.00242745 bias Pull 0.0766501 sigma 1.05951
```

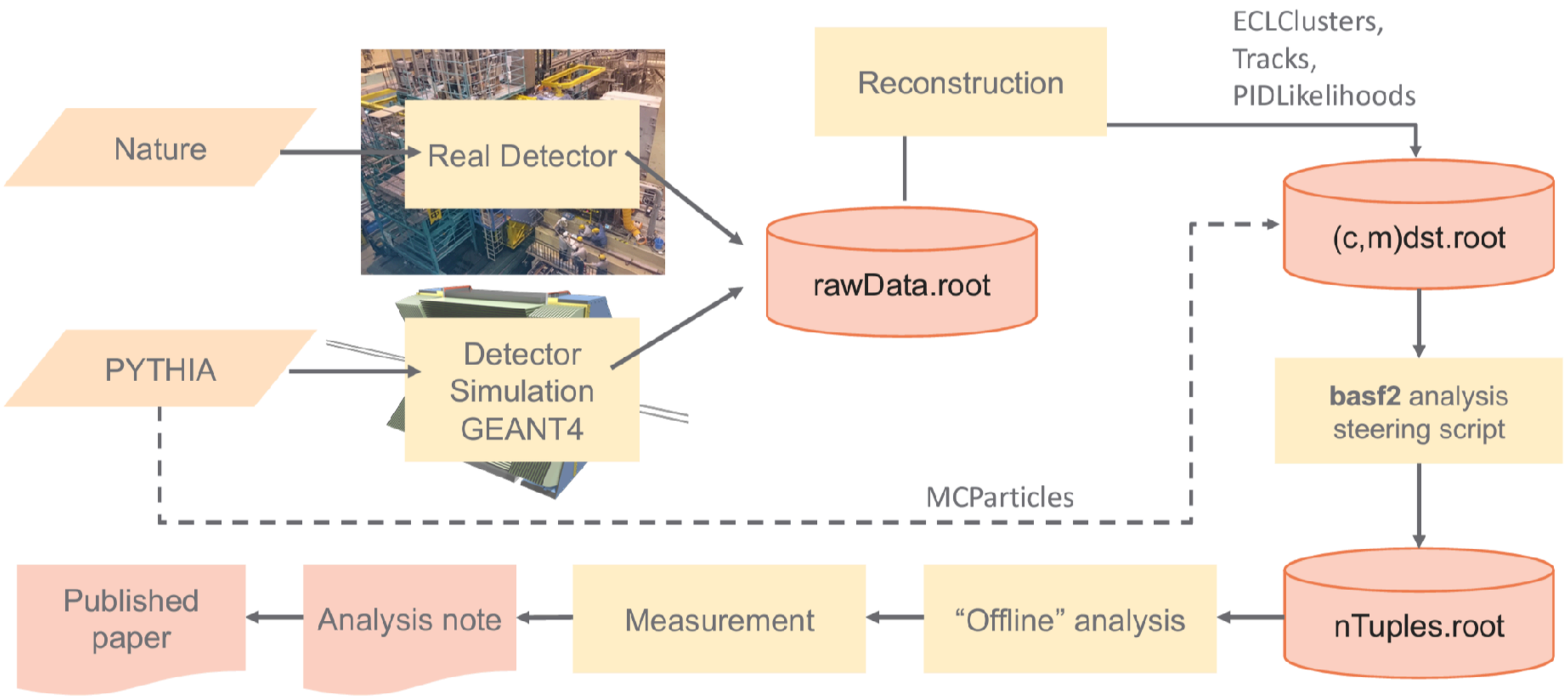
Get new ToyManager from previous fit

Run multicore on PROOF-lite
(or could use ROOT.Farm.Go(fit2))

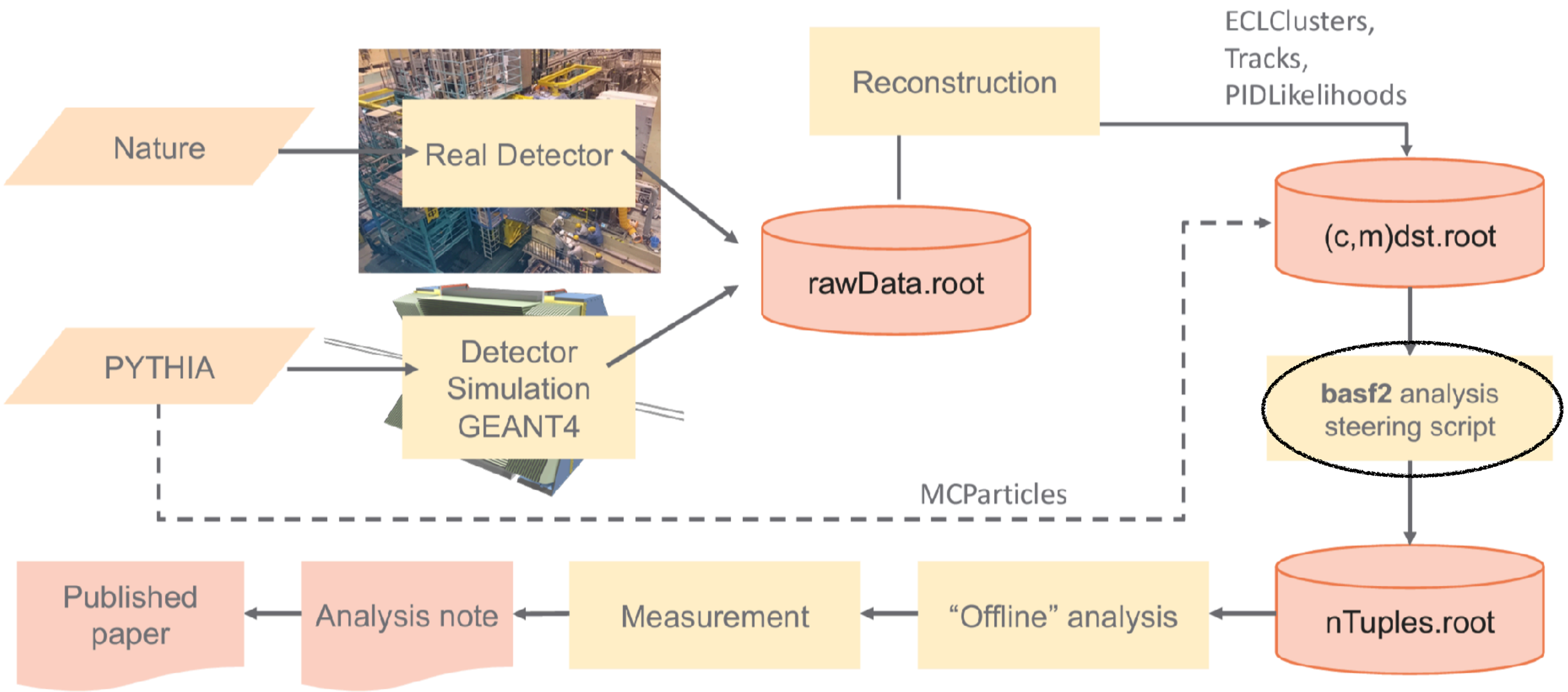
Plot parameter distributions and pulls for the 400 toy fits



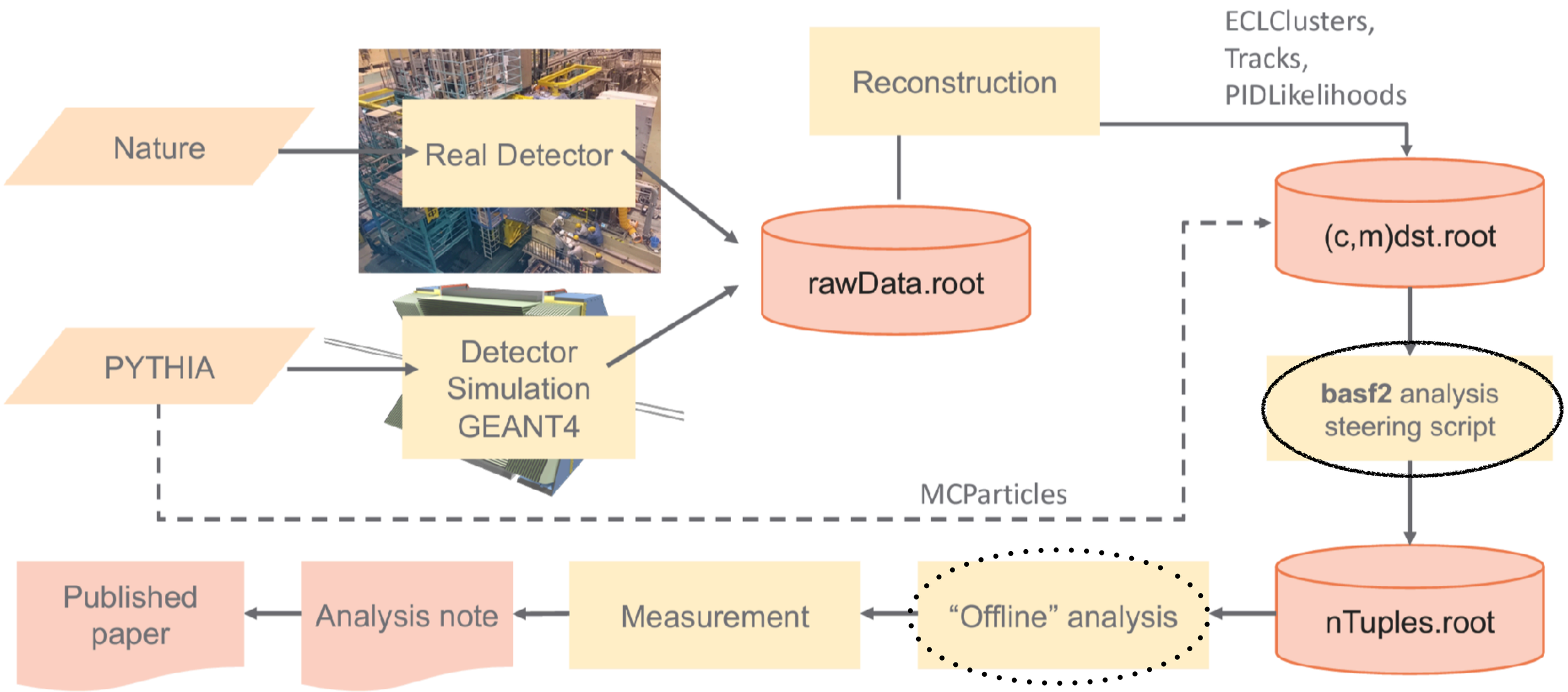
The big picture



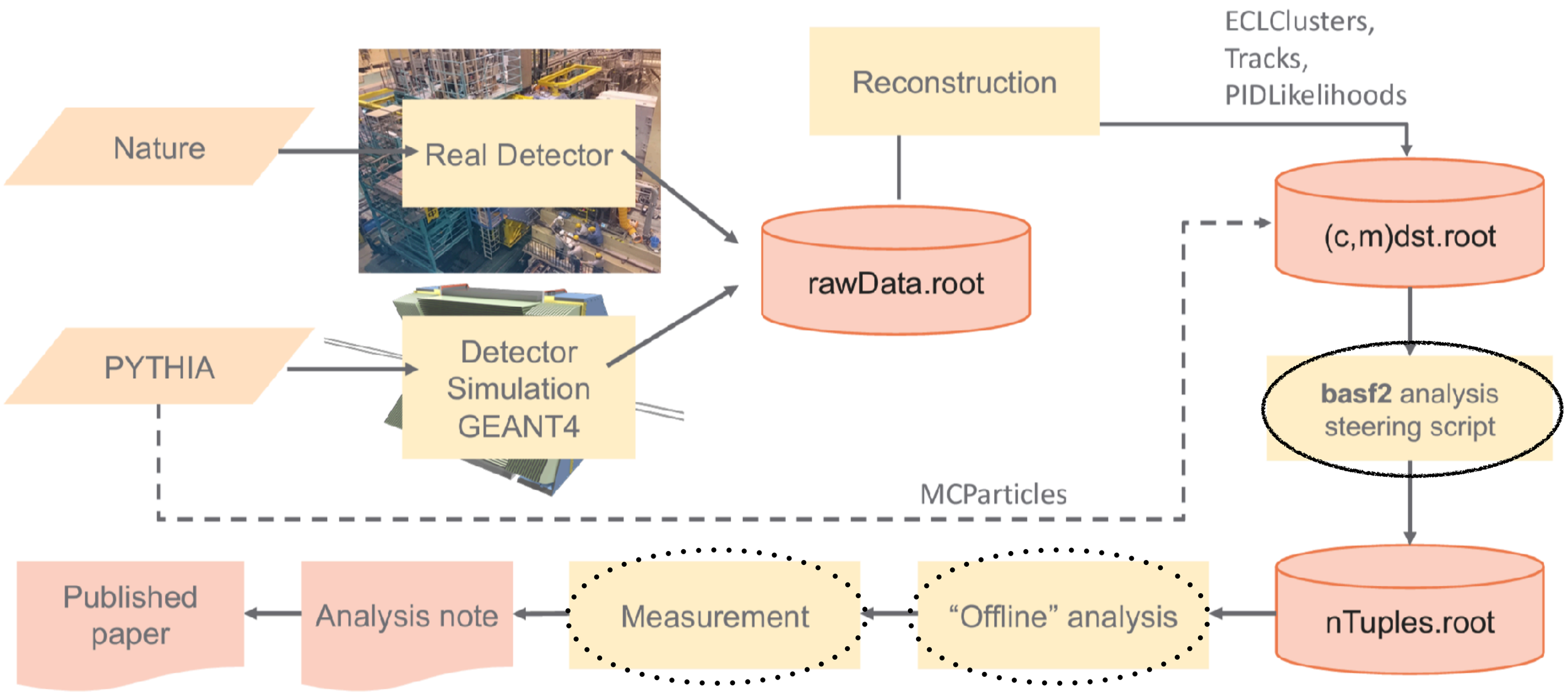
The big picture



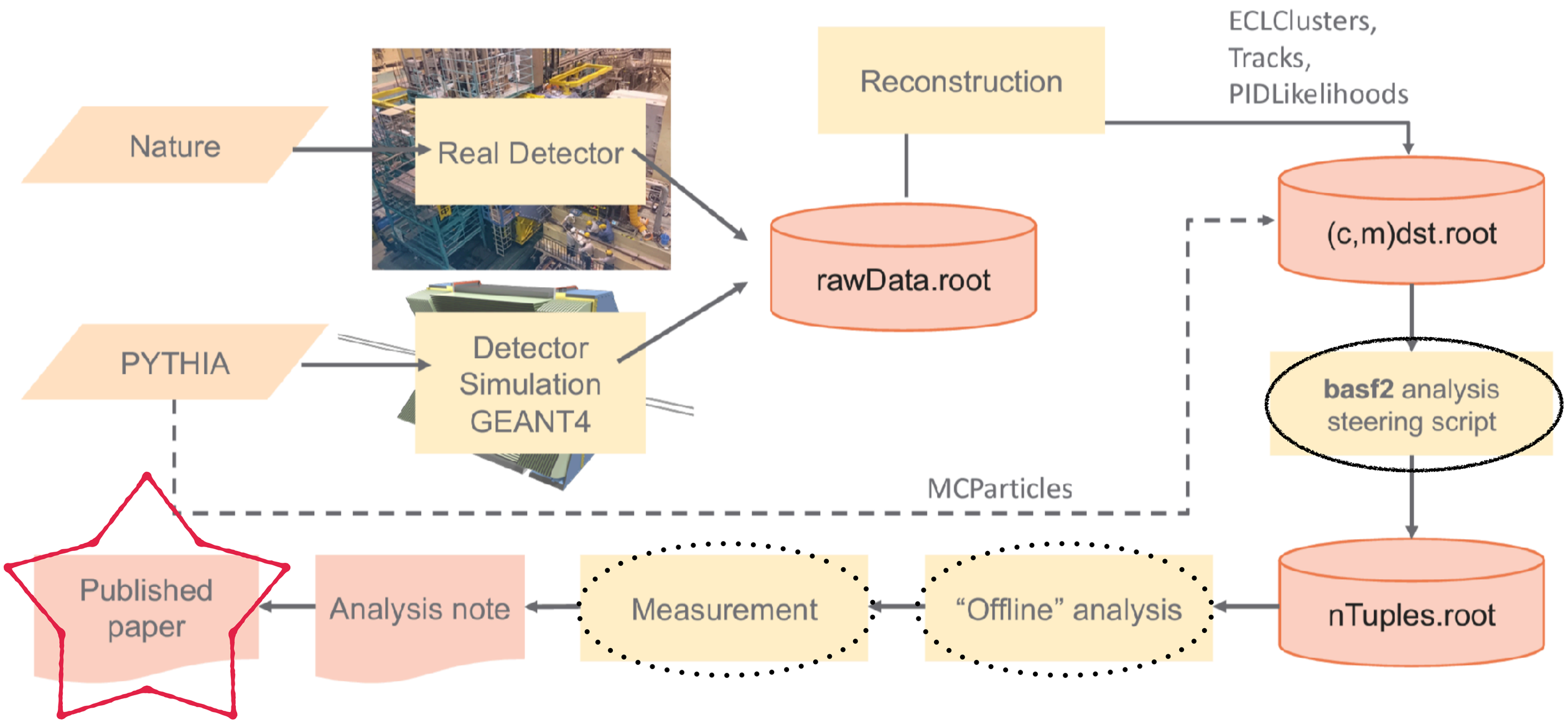
The big picture



The big picture



The big picture



Jargon

Acronym	Stands for	Actually means
`basf`	Belle Analysis Software Framework	The name of the software for the Belle experiment.
`basf2`	Belle 2 Analysis Software Framework (why not b2asf ? … I've no idea)	The name of the software for the Belle 2 experiment. See the pre-B2GM tutorials for more information.
`gbasf2`	Belle 2 Grid Analysis Software Framework	The name of user-side software tools for running basf2 code on the grid.
DIRAC	Distributed Infrastructure with Remote Agent Control	Software to manage jobs and files on the grid http://diracgrid.org/
Belle2DIRAC	Belle II extensions for the DIRAC system	Belle II custom things for the above software.
OSG	Open Science Grid	The worldwide grid of computing resources. Not just for HEP (also astronomy, molecular science…)
(W)LCG	(Worldwide) LHC Computing Grid	The computing grid for CERN experiments.
VO	Virtual Organisation	The name of the group of users who share infrastructure (like a HEP experiment). Confusingly, our VO is called 'belle' (with no '2').
VOMS	Virtual Organisation Membership Service	A system of managing authorisation for certificates with a VO.
LPN	Logical Path Name	Virtual path to a dataset.
FPN	Physical Path Name	The real path to a dataset at a specific site (you should never need to care about this).
MC9	Monte-Carlo (campaign) 9	The ninth campaign for Belle II to generate samples of simulated fake data (MC).

谢谢！