Analysis software.

Introduction, design, some new features, and tips

Sam Cunliffe

Bonn-DESY-Göttingen Inclusive Meeting, Hamburg, 13.11.2019

HELMHOLTZ RESEARCH FOR GRAND CHALLENGES

Belle II

basf2

DESY.

My plan for this talk

- The Belle II software.
  - packages, Modules, Paths, DataStore, database.
- A brief (hopefully useful) diversion: **light releases**
- The analysis package.
  - Design.
  - Particles, candidates.
  - New features of interest to inclusive/missing energy analyses.
- Misc. tips and advice.

Kunsthalle Hamburg/Caspar David Friedrich, *Der Wanderer über dem Nebelmeer*

software.belle2.org

questions.belle2.org

Where to go for help/examples

Slightly less flippant

- **Software doc:** software.belle2.org
  - Here is what's new in release-04 (or just go there and search).
- **Q&A:** questions.belle2.org.
- search.belle2.org.
- **Code:** stash.desy.de (the B2 "project" is `basf2`).
- **Tutorials**:
  - Here is the indico category for past starterkit workshops.
  - Interactive tutorials: jupyterhub.belle2.org.
  - `$BELLE2_RELEASE_DIR/analysis/examples/*`

basf2: packages, Modules, Paths, DataStore, and the database

What?

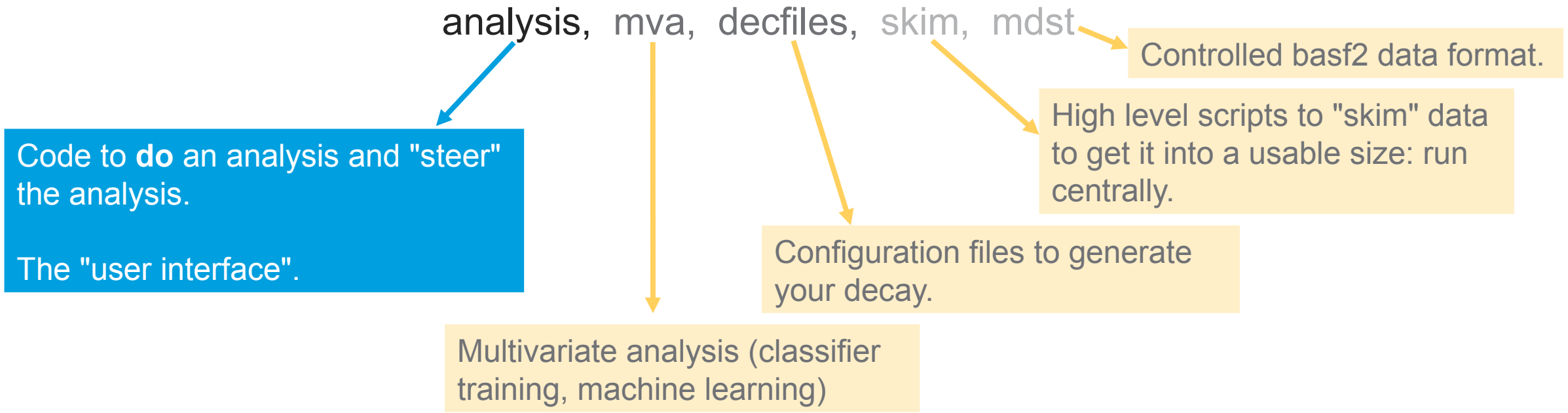
Comput Softw Big Sci (2019) 3: 1.

- `basf2` = Belle II analysis software framework.
  - Why not `b2asf`? I don't know. Nobody consulted me.
- Don't be fooled by the word "analysis".
  - Actually basf2 does everything. You should call it "***The Belle II core software***".
  - Analysis is a subset of basf2.

- It's also a **python module** for steering core functionality.
- It's also the name of the **executable**.
  - … which can be thought of as `ipython3`.

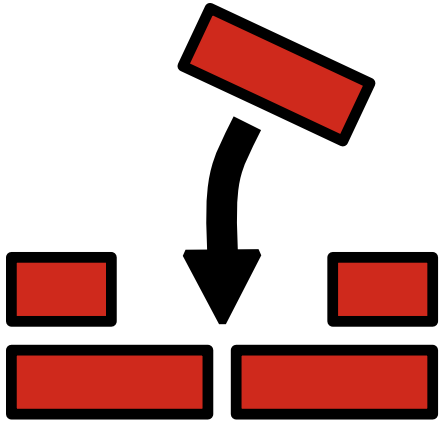
Packages

- Code is organised into directories (== packages).
  - Code for similar tasks lives together.
  - There is one for each subdetector, and one for tracking, simulation, etc.
- As a postdoc/PhD student you might do some development in another package.
  - But as an analyst, you really only care about a few:



Package structure

Some directories are special



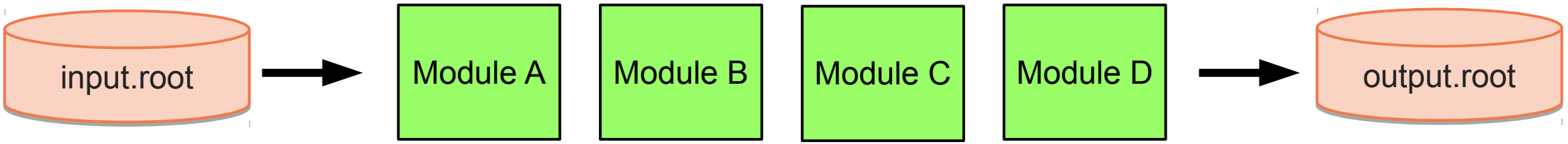
scons.org

- Build system is called `scons`.
  - It's quite cool.
- Some special/important directories within a package.
  - scripts (for python code… not scripts)
  - examples (for example scripts)
  - modules (compiled by the build system)
  - dataobjects (compiled by the build system)
  - dbobjects (database objects; compiled by the build system)
  - tests (executed automatically in pull requests and nightly)
  - tools (executables, added to `$PATH` should be named `b2something-doer`)
  - variables
- All of this will become clear or (I hope) is self-explanatory.

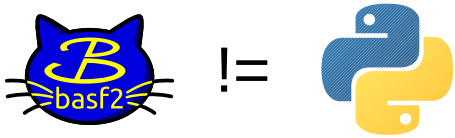
Modules, Paths, the DataStore

What do we need to process the data?

1) A set of classes (modules) that process the data
   → `basf2` **Module**

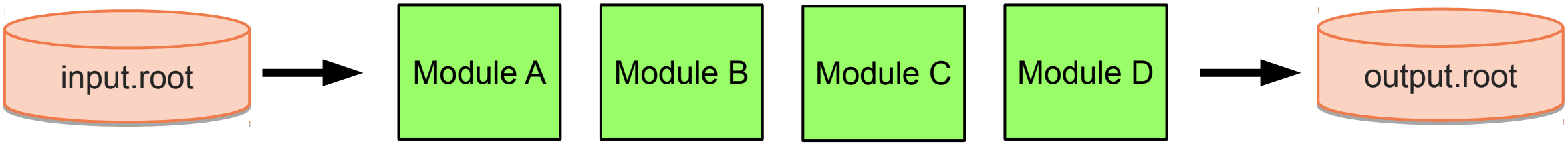


Modules, Paths, the DataStore



What do we need to process the data?

1) A set of classes (modules) that process the data
   → `basf2` **Module**

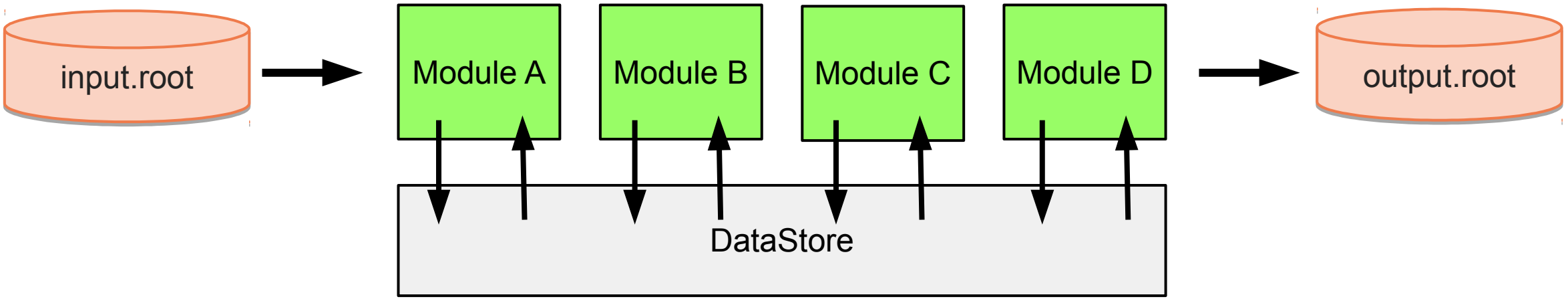


Modules, Paths, the DataStore

What do we need to process the data?

1) A set of classes (modules) that process the data
   → `basf2` **Module**

2) Classes that hold data and allow module to pass thing one to the other
   → `basf2` **dataobject** (lives in the datastore)

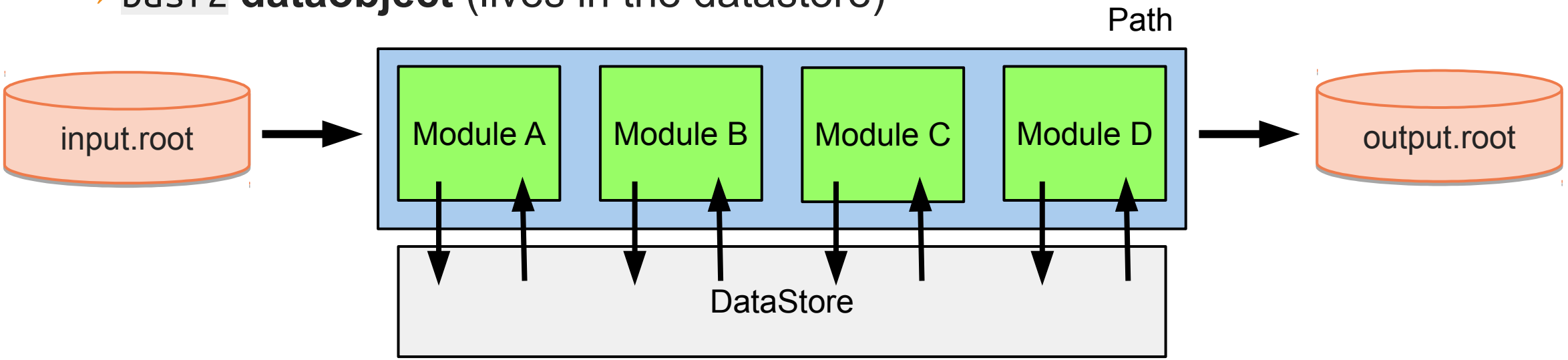


Modules, Paths, the DataStore

What do we need to process the data?

1) A set of classes (modules) that process the data
   → `basf2` **Module**

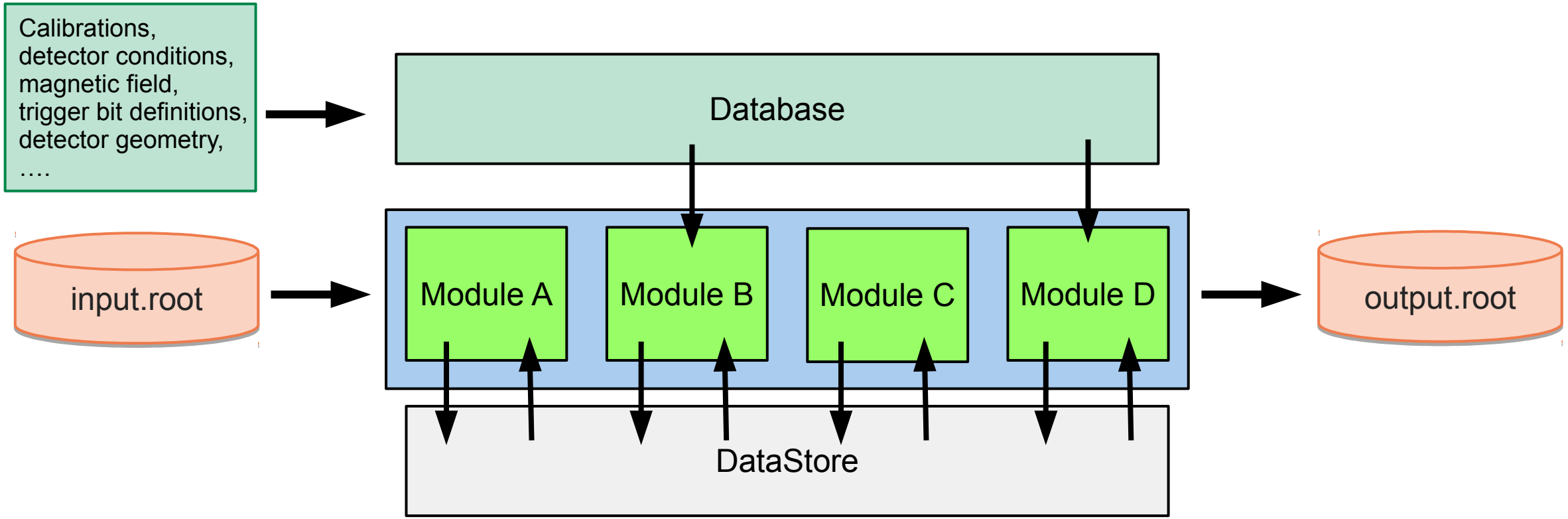
2) Classes that hold data and allow module to pass thing one to the other
   → `basf2` **dataobject** (lives in the datastore)



3) An order in which the modules must be executed
   → `basf2` **Path**

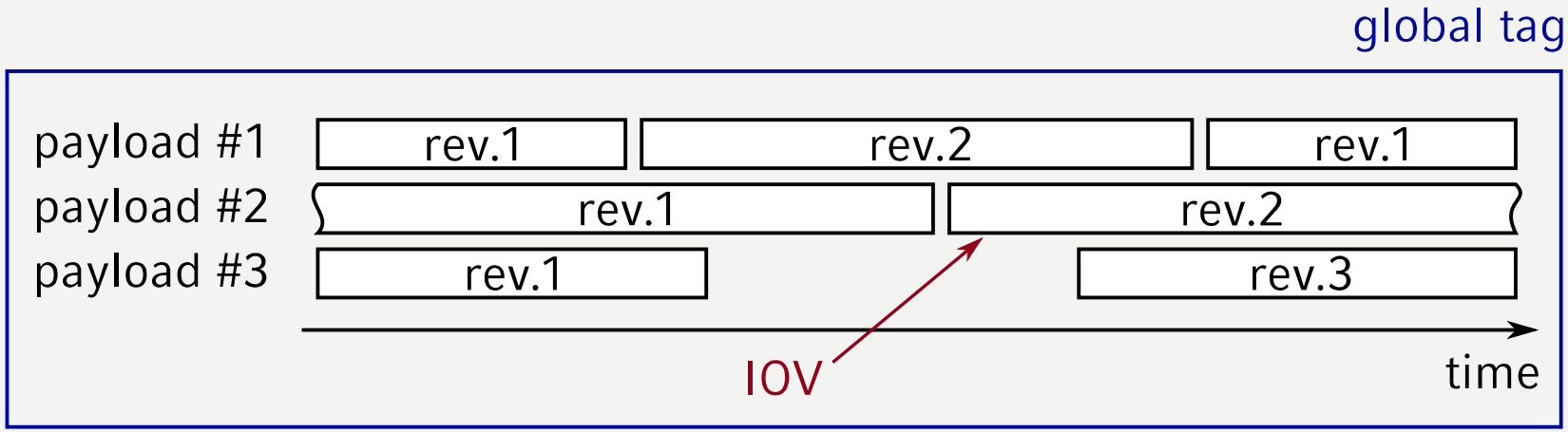
The database

For data-taking / simulation “conditions”



4) The database

What is a Global Tag?



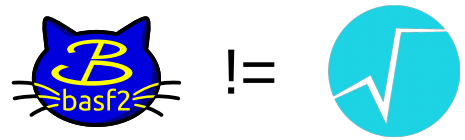
payload — One atom of conditions data (e.g. BeamParameters).
In our case this is just **a file**.
They are identified by name and have different revisions

IOV — Short for "interval of validity", the run interval for which the payload is valid.
Can be a fixed run range (closed) or starting at a given run (open)

global tag — Is an **immutable** set of payloads and their IoVs

Link to Martin's full talk

A word on dataobjects and Belle II file types



This sounds boring but it's quite important

- All `basf2` dataobjects are `Belle2::RelationsObject`s (inherits from `ROOT::TObject`).
  - But you should not try to open files with root.
  - "Kind-of" works. But totally unsupported, absolutely no guarantee of backward (or any) compatibility.
- **dst**: can save any of the dataobjects in the DataStore (with `EStoreFlags::c_WriteOut`)
  - **d**ata **s**ummary **t**able.
- **mdst**: the legacy/official analysis format.
  - **m**ini-dst.
  - Strictly controlled list of dataobjects.
  - Has **M 円** implications in computing budget.
- **cdst**: calibration format.
  - mdst + a few others.
  - Huge, so only produced sparingly and for calibration.
- **udst**: user format. mdst + `Belle2::Particle`s.

```
branches = [
    'Tracks',
    'V0s',
    'TrackFitResults',
    'EventLevelTrackingInfo',
    'PIDLikelihoods',
    'TracksToPIDLikelihoods',
    'ECLClusters',
    'ECLClustersToTracksNamedBremsstrahlung',
    'EventLevelClusteringInfo',
    'TracksToECLClusters',
    'KLMClusters',
    'KlIds',
    'KLMClustersToKlIds',
    'TRGSummary',
    'SoftwareTriggerResult',
]
```

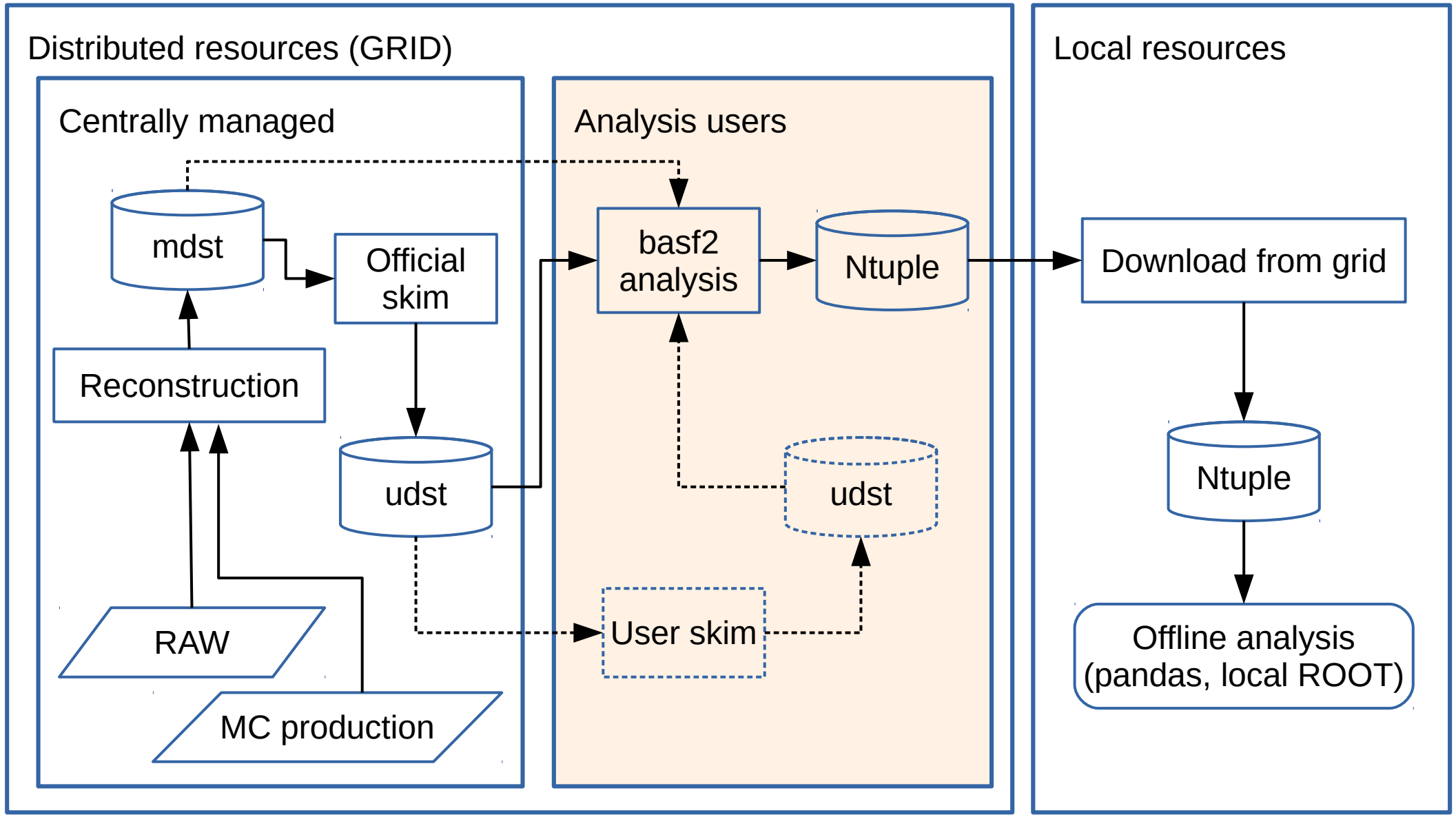
`$BELLE2_RELEASE_DIR/mdst/scripts/mdst.py`

Jargon

A quick resume

- `basf2`: The Belle II software / a **python** module / an executable.
- **Package**: related code lives together.
- **Module**: does a data processing task.
- **Path**: ordered list of modules.
- **DataStore**: place where the **dataobjects** live. Written to and read from by modules.
- **Database**: place where conditions data lives, in the form of **payloads**.
  - Modules can access it.
- **Global tag**: a labeled collection of database data.
  - Historically typed by the user into her scripts.
  - Now not needed*.
- **mdst**: the Belle II data file format for analysis / a package containing the dataobjects.

*) unless you want to run the FEI


Distributed resources (GRID)
Centrally managed
Analysis users
Local resources
mdst
Official skim
basf2 analysis
Ntuple
Download from grid
Reconstruction
udst
udst
Ntuple
RAW
User skim
Offline analysis (pandas, local ROOT)
MC production

Light releases

Are cool

- Subset of packages that compile standalone:
  { **analysis**, mdst, mva, skim,
  framework, geometry }
- Everything that is needed to open mdst
  ⇔ everything needed for analysis.
- Make light releases often → faster development cycle than full `basf2`.
- More flexible. Can be powerful.
  - e.g. FEI with baryonic modes is not available in release-04 but it is in the light release.

`light-YYMM-codename`

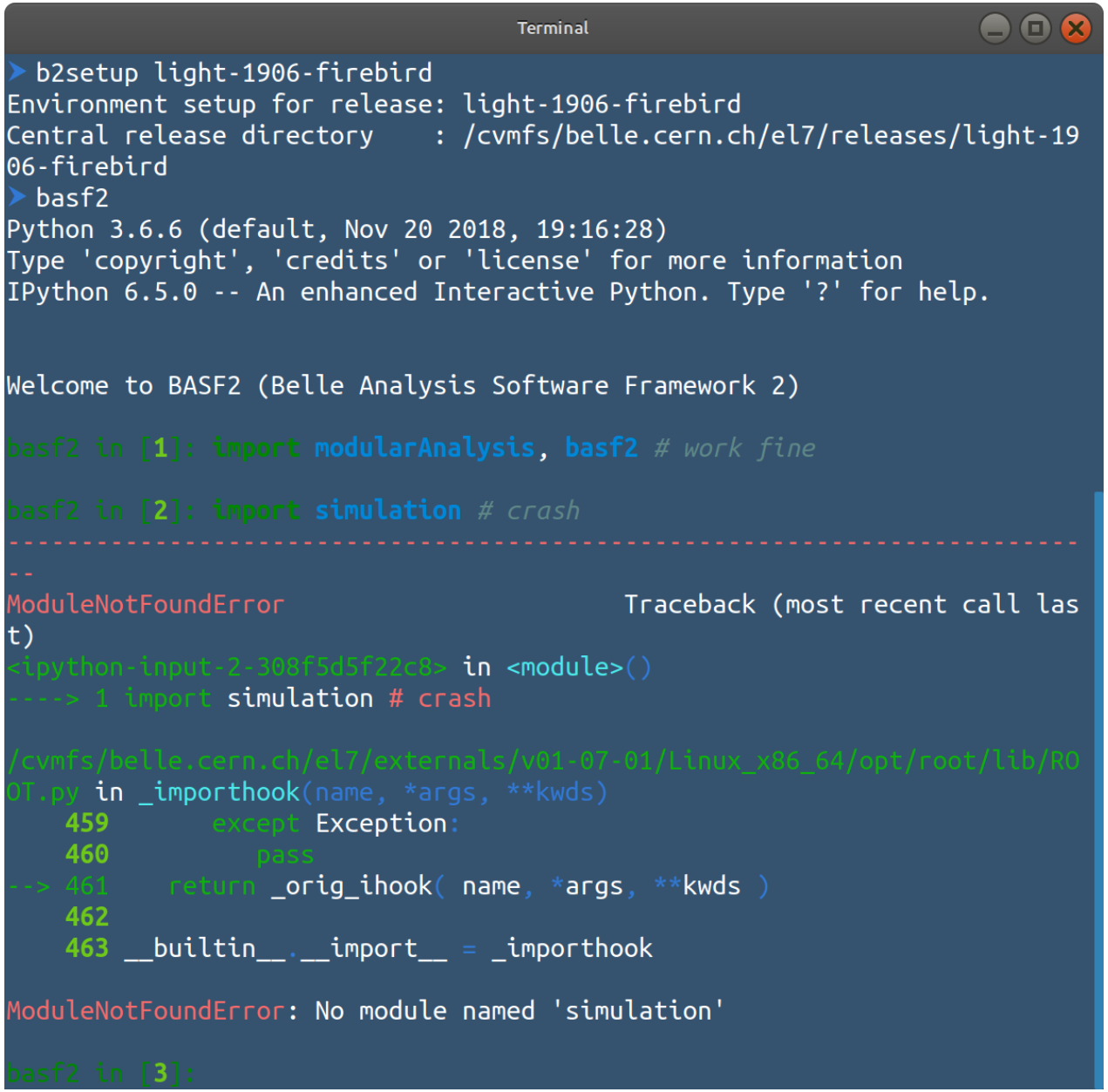
Limitations

You can't generate (you should not be doing that anyway)

- Generation / simulation / reconstruction jobs won't work (by construction).
  → `ModuleNotFoundError`s if you try.
- cdst not supported (by construction)
- Semver major release.
  - Backward compatibility is not guaranteed.

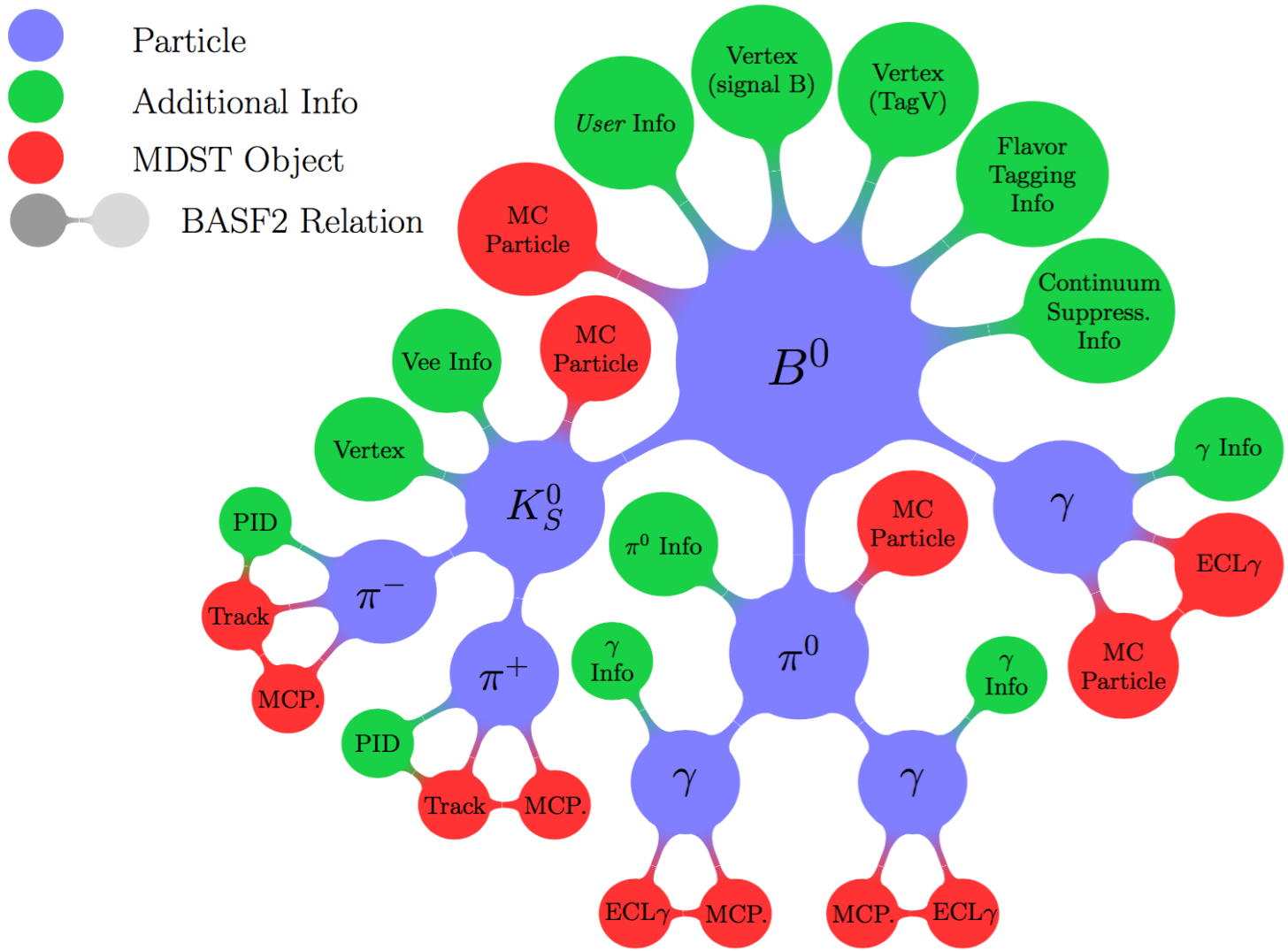
Only supported actions:

- Read a preexisting mdst file.
- Make particles and do analysis things (vertex fits, FEI, TreeFitter).
- Write out user 'tuples



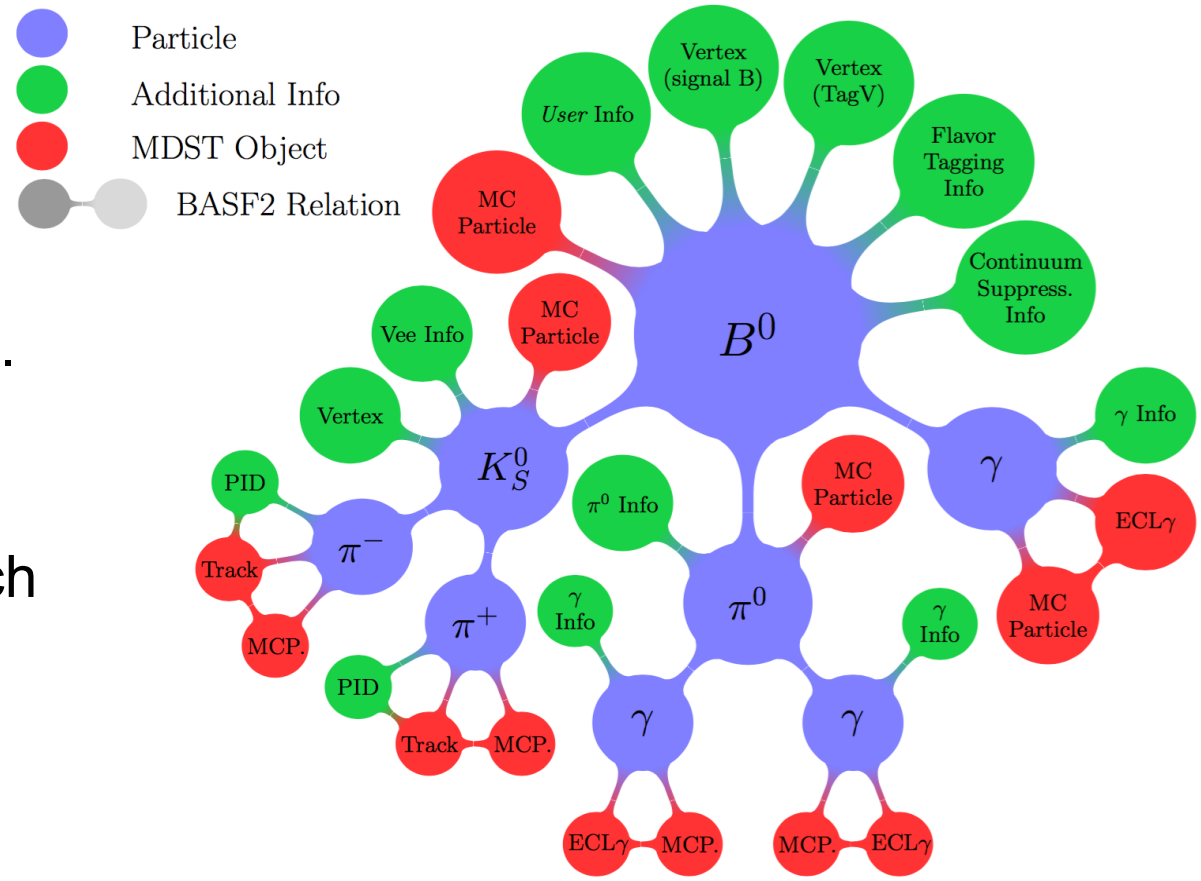
Analysis package: Particles, variables

Particles


Particle
Additional Info
MDST Object
BASF2 Relation
B^0
Vertex (signal B)
Vertex (TagV)
Flavor Tagging Info
Continuum Suppress. Info
User Info
MC Particle
K^0_S
Vee Info
MC Particle
Vertex
π^-
PID
Track
MCP.
π^+
PID
Track
MCP.
π^0
π^0 Info
MC Particle
γ
γ Info
Info
γ
ECLγ
MCP.
γ
Info
γ
ECLγ
MCP.
MCP.
ECLγ

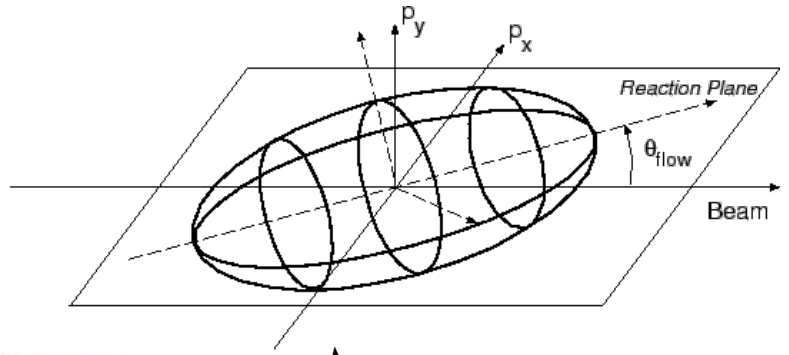
Particles

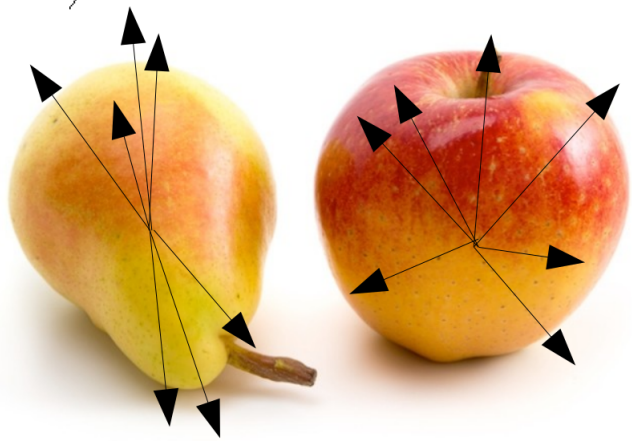
- **Candidate `Belle2::Particle`** created from mdst dataobjects or composite:
  - Track w/ PIDLikelihood → Charged particle.
  - Calorimeter cluster → Neutral particle.
- Core object: `Belle2::ParticleList` of all such candidates in an event.
- Combine `Belle2::ParticleLists` with the `ParticleCombiner` (a module) to create *composite* particle candidates (e.g. $B \to K_S \pi^0 \gamma$).



Analysis tools

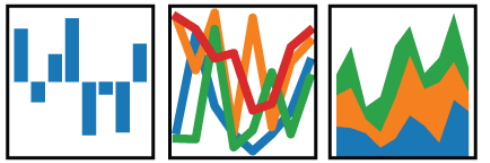
- High-level tools operate on `Belle2::ParticleLists` (once per candidate).
  - Fit vertex (RAVE, Kfit).
  - Kinematic fitting (Kfit, fork of ILC/Marlin).
  - Tag the B flavour.
  - Build the RestOfEvent.
  - Write candidates for offline study.
  - Fit full decay chain.
- Notable exceptions:
  - EventShape.
  - *Full event interpretation*.





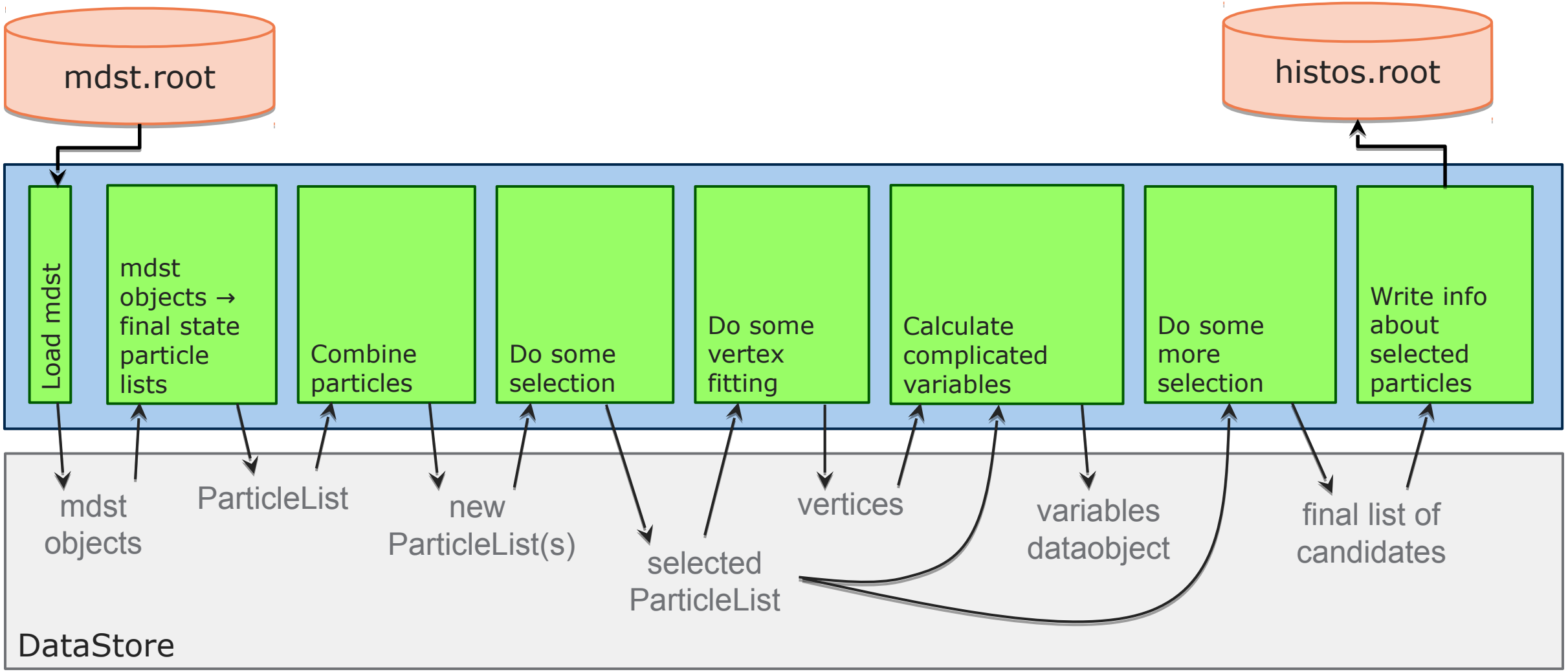
pandas

$$y_i t = \beta' x_{it} + \mu_i + \epsilon_{it}$$



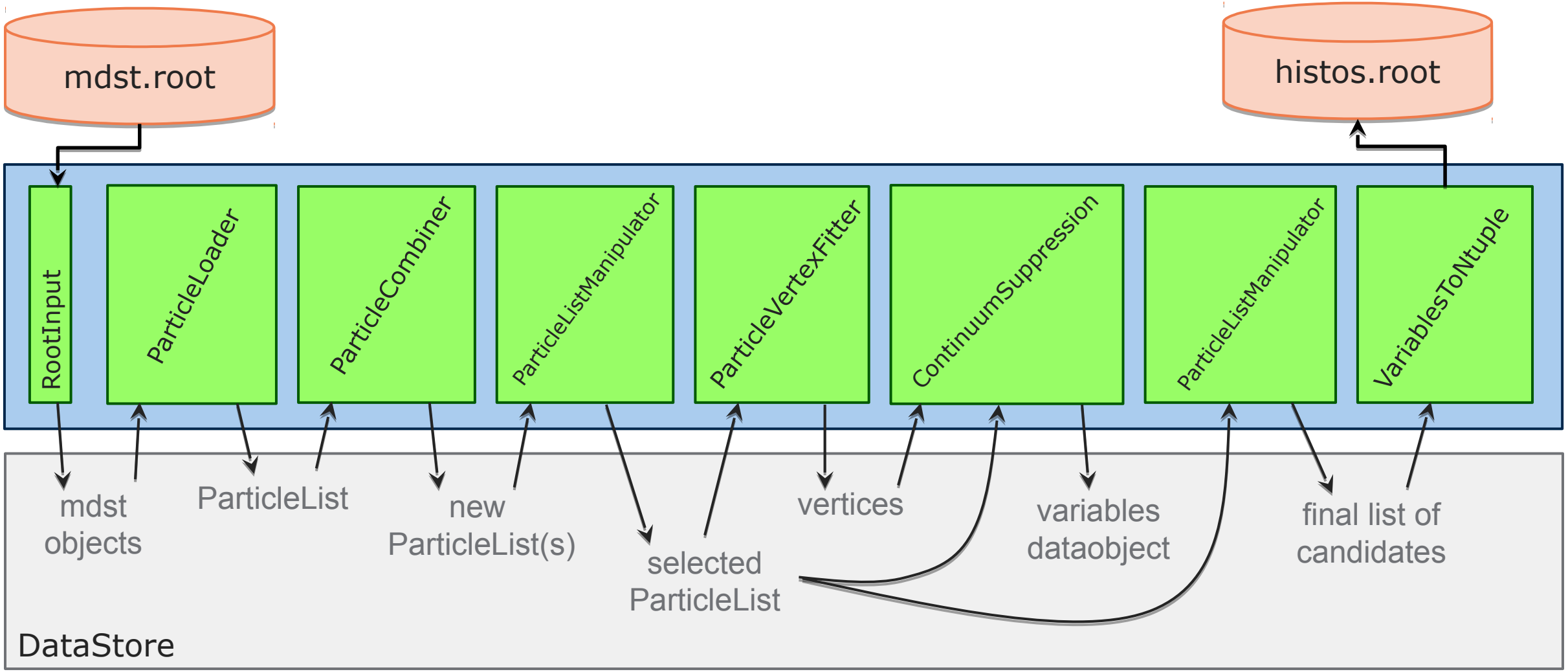
Artist's impression of event shape: U. Tamponi

A typical path for an analysis job



A typical path for an analysis job

Now with the real names for the modules



How to load a module

- The syntax is the same no matter what module you want.
- You “register” it, (optionally) set some parameters, then add it to the path.

```
mod = basf2.register_module('ModuleName')
mod.param('someParameter', value)
mypath.add_module(mod)
```

Can you figure out what this does?

```
pcomb = basf2.register_module('ParticleCombiner')
pcomb.param('decayString', 'K*0:myKst -> K+:good pi-:good')
pcomb.param('cuts', '0.6 < M < 1.0')
path.add_module(pcomb)
```

Can you figure out what this does?

- If you said “combine particles” you’re *close to* correct.
- Actually it adds a module which **will do that** to the path.

```
pcomb = basf2.register_module('ParticleCombiner')
pcomb.param('decayString', 'K*0:myKst -> K+:good pi-:good')
pcomb.param('cuts', '0.6 < M < 1.0')
path.add_module(pcomb)
```

- Actually this is a common misconception.
- We are not in an event loop.
- We are configuring the processing we want to happen to our events.

Shorthand function syntax

reconstructDecay just configures a ParticleCombiner

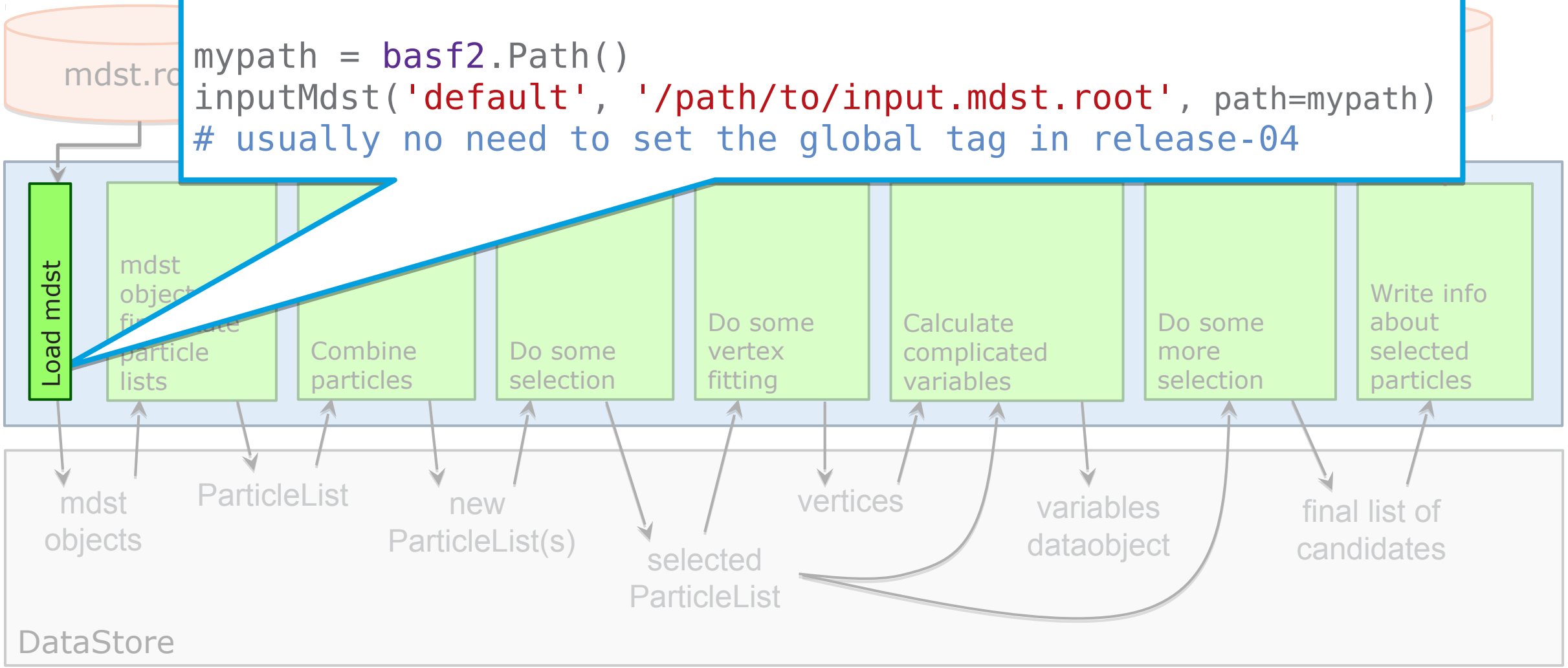
- Module setup can get complex.
- There are convenience functions. But this is a double-edged sword.

```
pcomb = basf2.register_module('ParticleCombiner')
pcomb.param('decayString', 'K*0:myK          > K+:good pi-:good')
pcomb.param('cuts', '0.6 < M < 1.0')
path.add_module(pcomb)
```

```
modularAnalysis.reconstructDecay(
    'K*0:myKst -> K+:good pi-:good', '0.6 < M < 1.0', mypath)
```

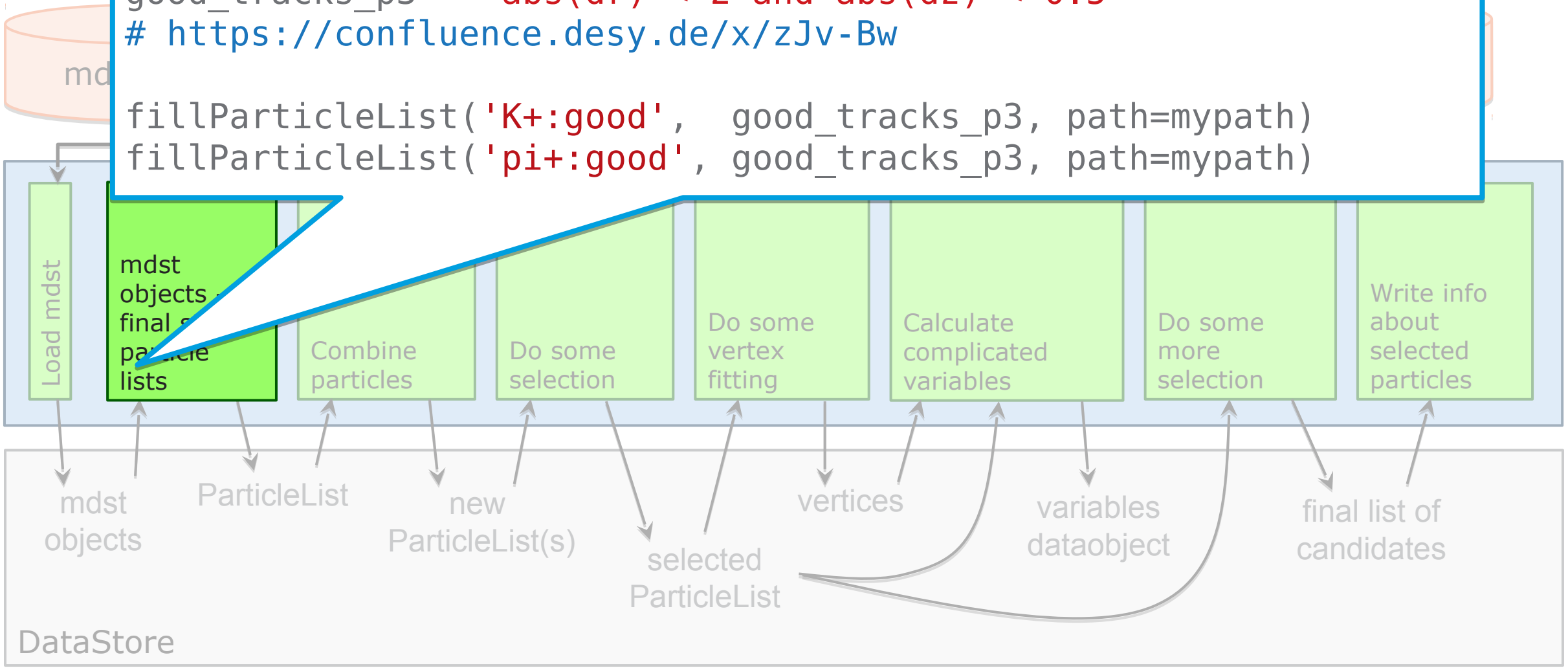
A typical path for an analysis job

```
import basf2
from modularAnalysis import inputMdst

mypath = basf2.Path()
inputMdst('default', '/path/to/input.mdst.root', path=mypath)
# usually no need to set the global tag in release-04
```



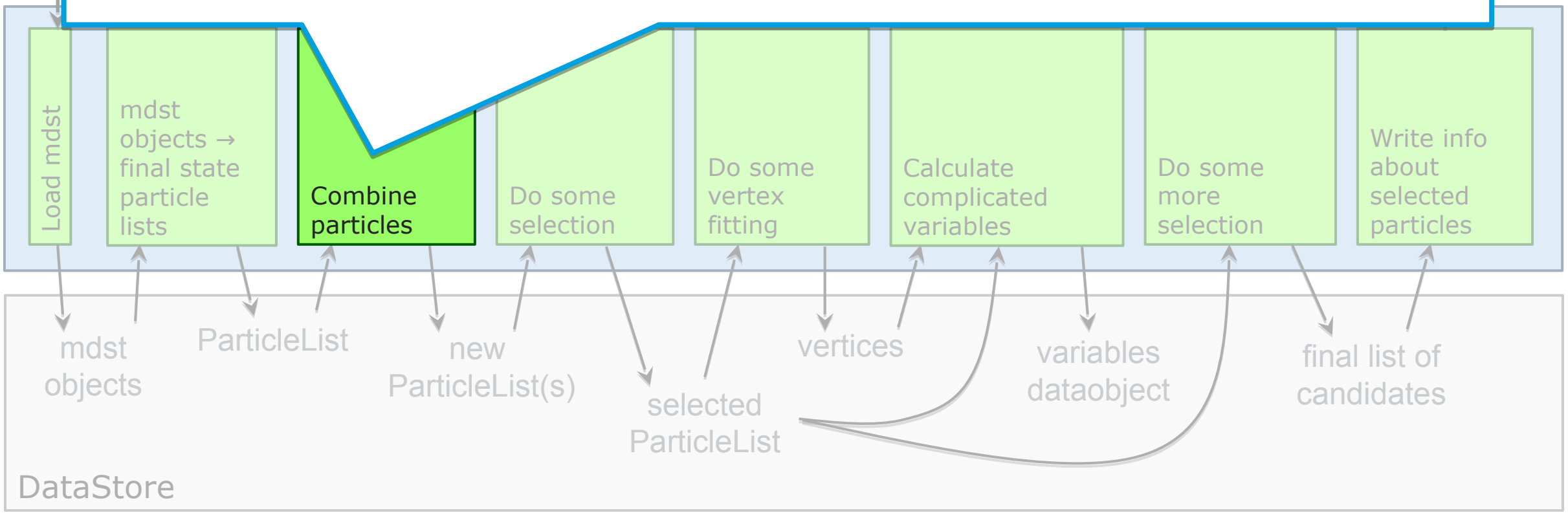
A ty

```
from modularAnalysis import fillParticleList

good_tracks_p3 = 'abs(dr) < 2 and abs(dz) < 0.5'
# https://confluence.desy.de/x/zJv-Bw

fillParticleList('K+:good',  good_tracks_p3, path=mypath)
fillParticleList('pi+:good', good_tracks_p3, path=mypath)
```



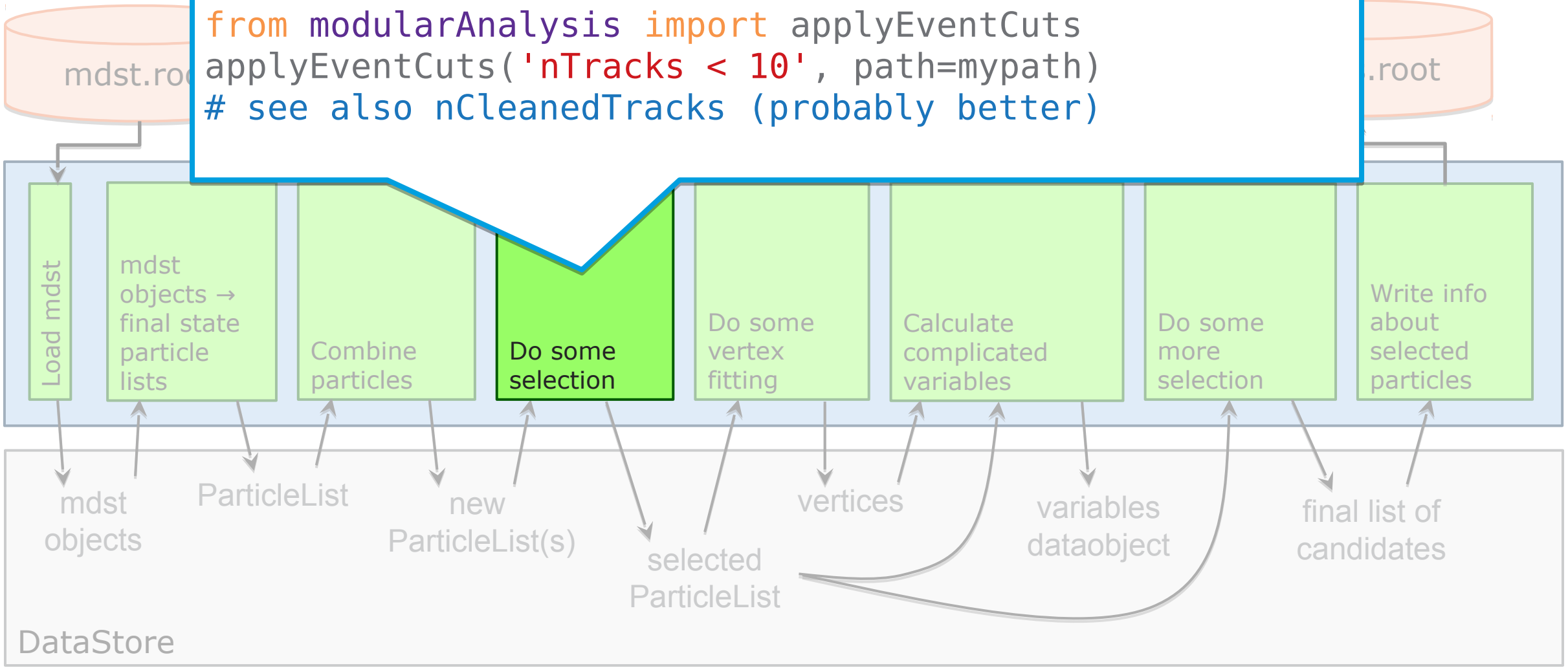
A typical path for an analysis job

```
from modularAnalysis import reconstructDecay
reconstructDecay(
    'K*0:myKst -> K+:good pi-:good', '0.6 < M < 1.0', path=mypath)
```



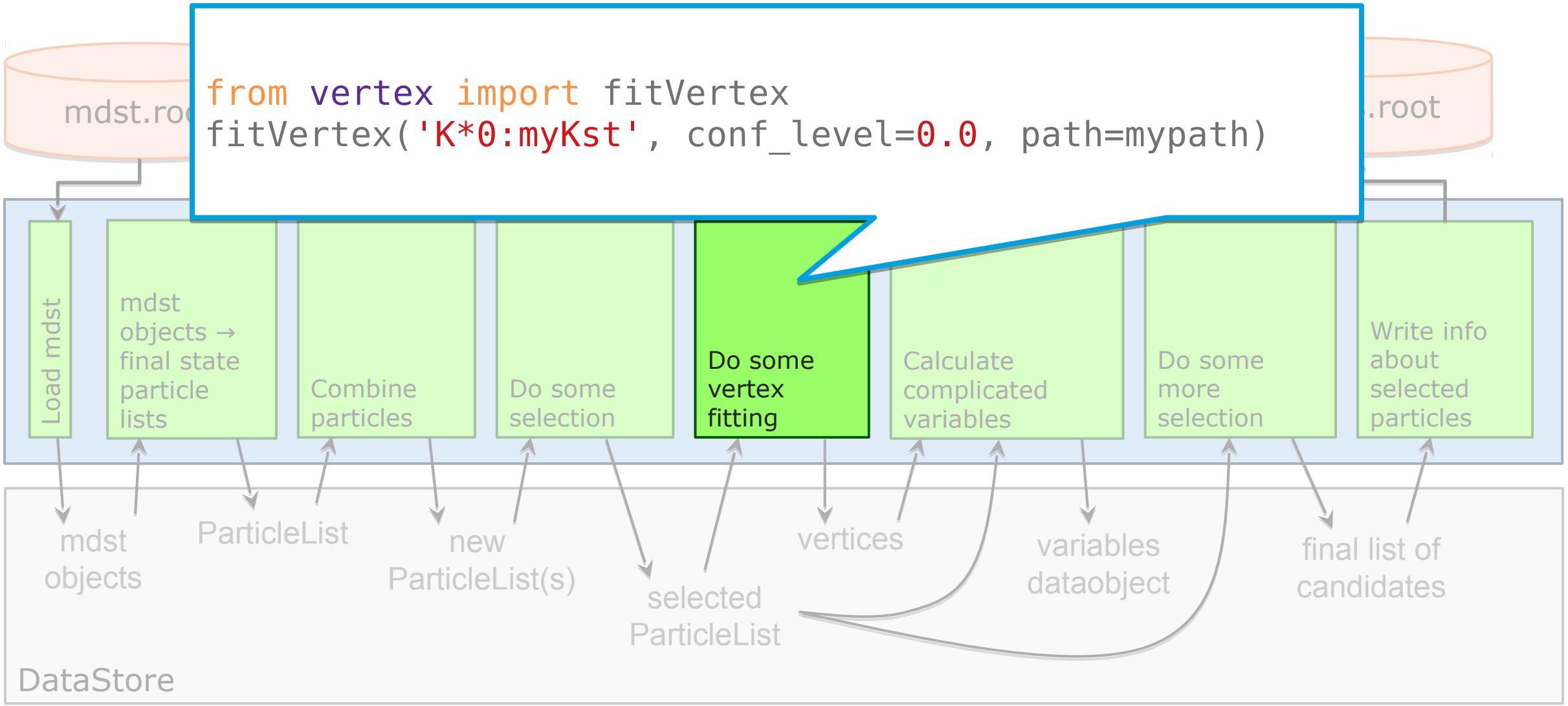
A typical path for an analysis job

```
from modularAnalysis import applyEventCuts
applyEventCuts('nTracks < 10', path=mypath)
# see also nCleanedTracks (probably better)
```



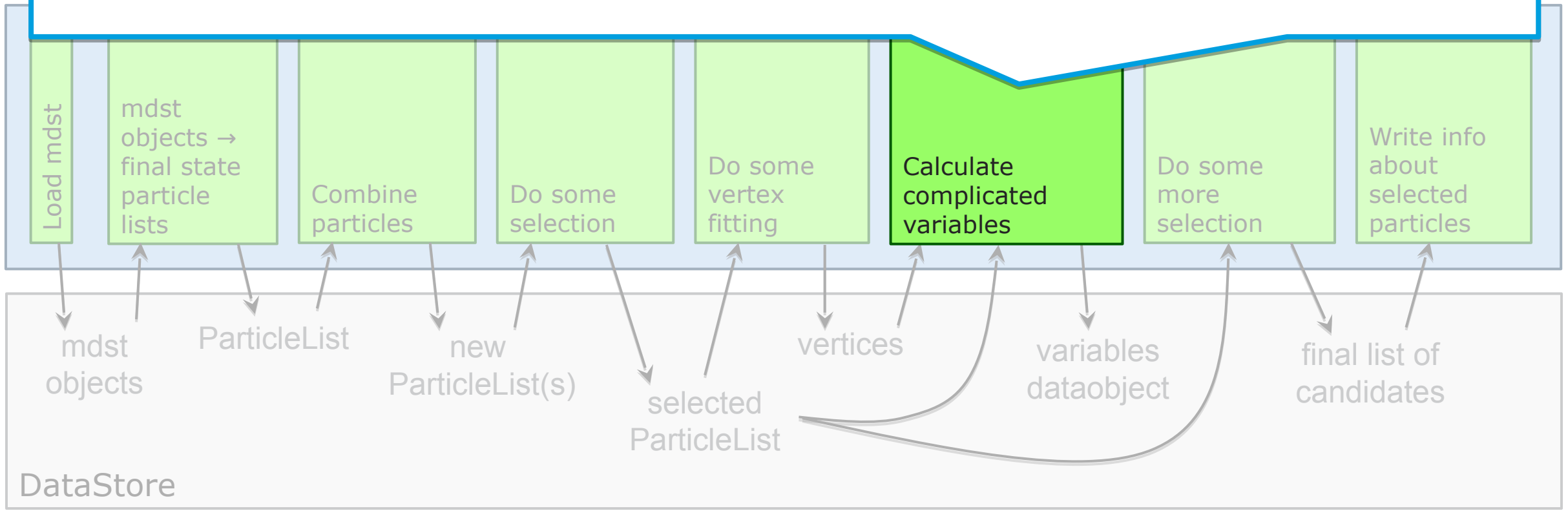
A typical path for an analysis job

```
from vertex import fitVertex
fitVertex('K*0:myKst', conf_level=0.0, path=mypath)
```



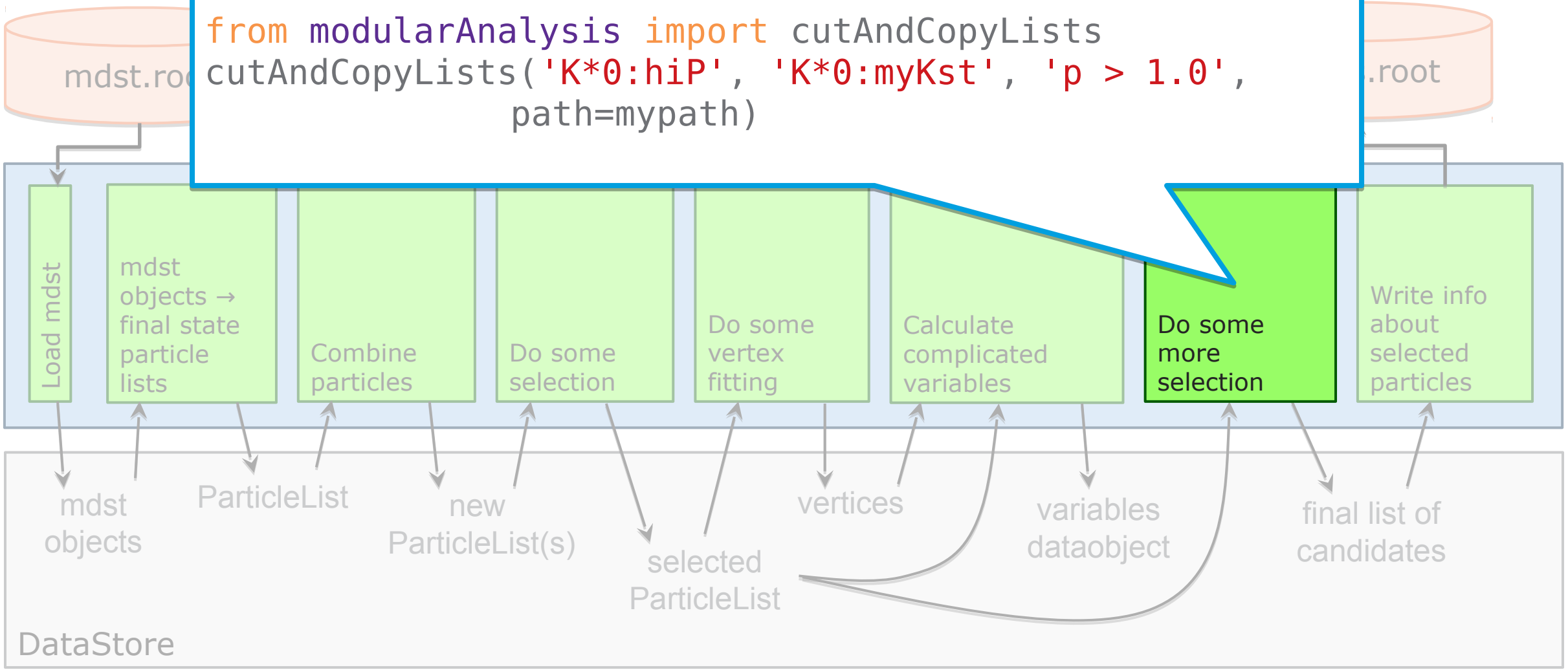
A typical path for an analysis job

```
from modularAnalysis import buildRestOfEvent
buildRestOfEvent('K*0:myKst', path=mypath)
```



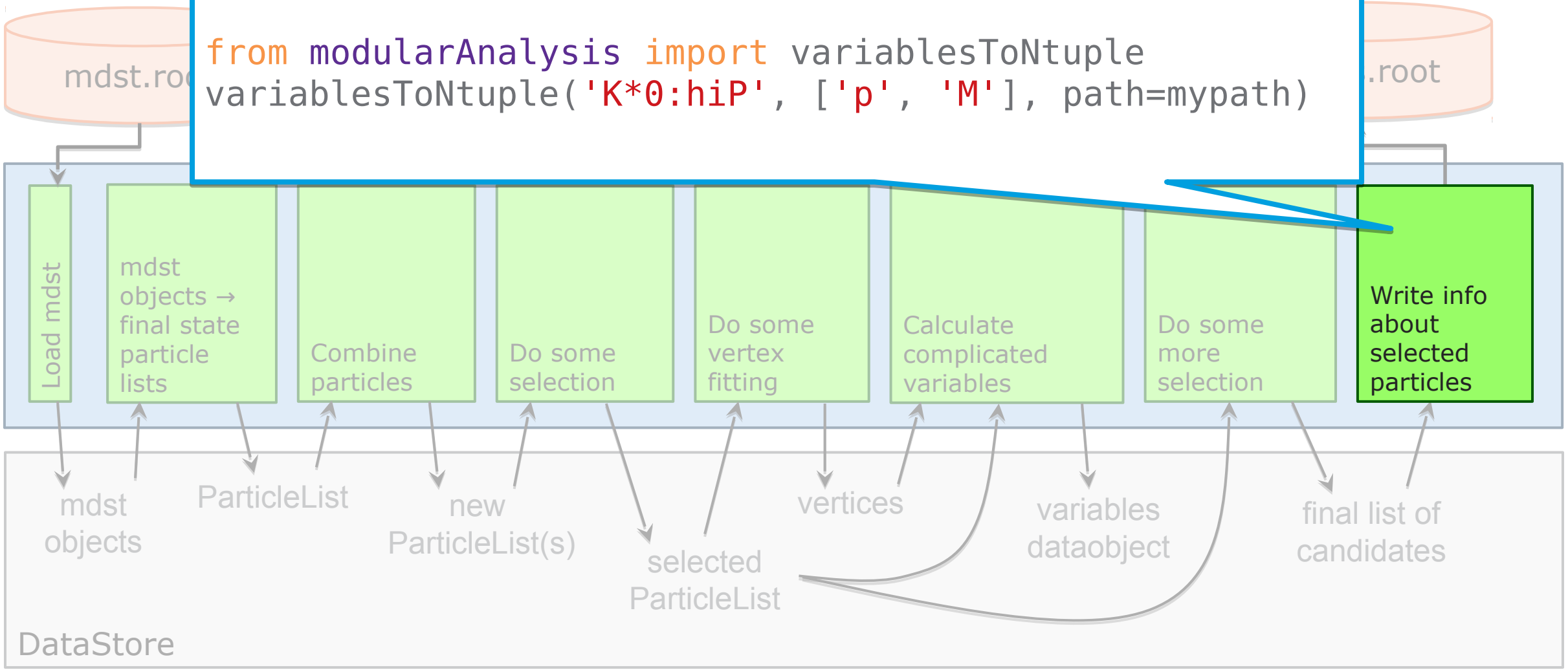
A typical path for an analysis job

```
from modularAnalysis import cutAndCopyLists
cutAndCopyLists('K*0:hiP', 'K*0:myKst', 'p > 1.0',
                path=mypath)
```

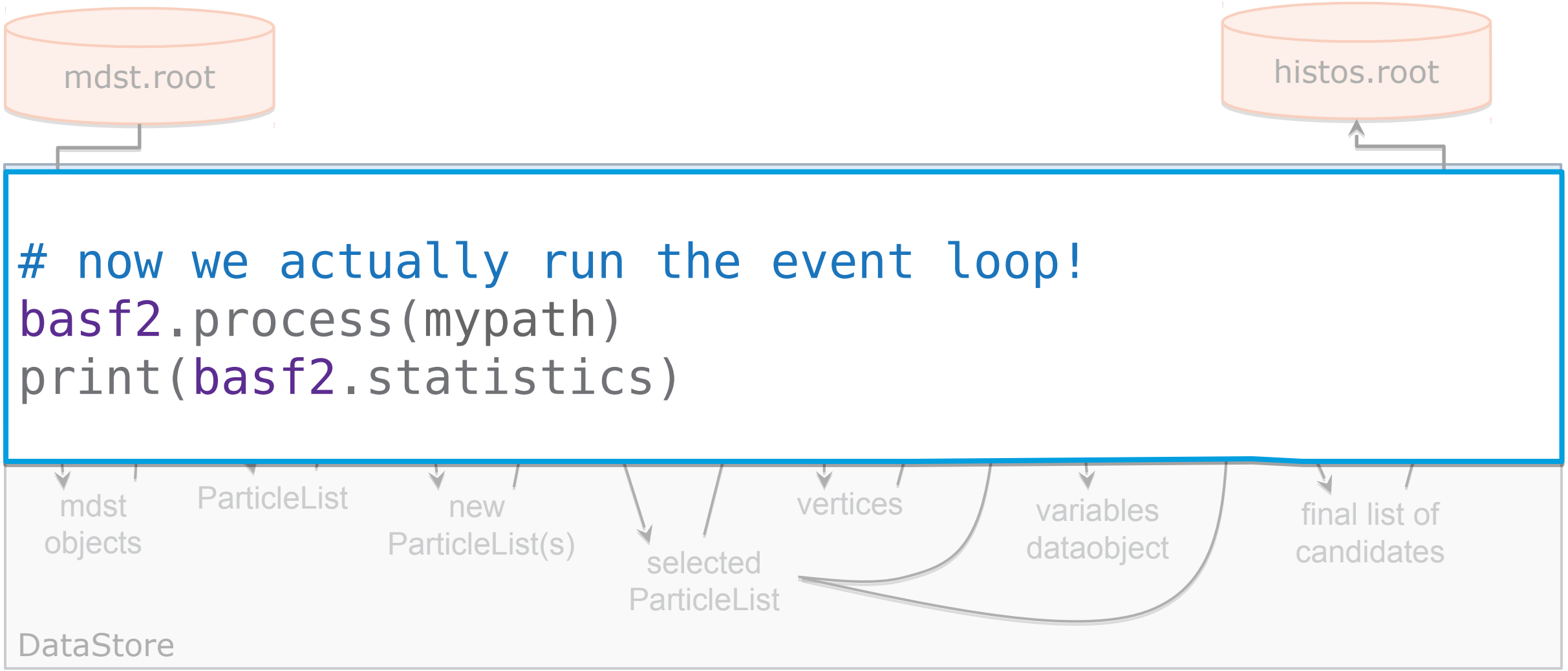


A typical path for an analysis job

```
from modularAnalysis import variablesToNtuple
variablesToNtuple('K*0:hiP', ['p', 'M'], path=mypath)
```



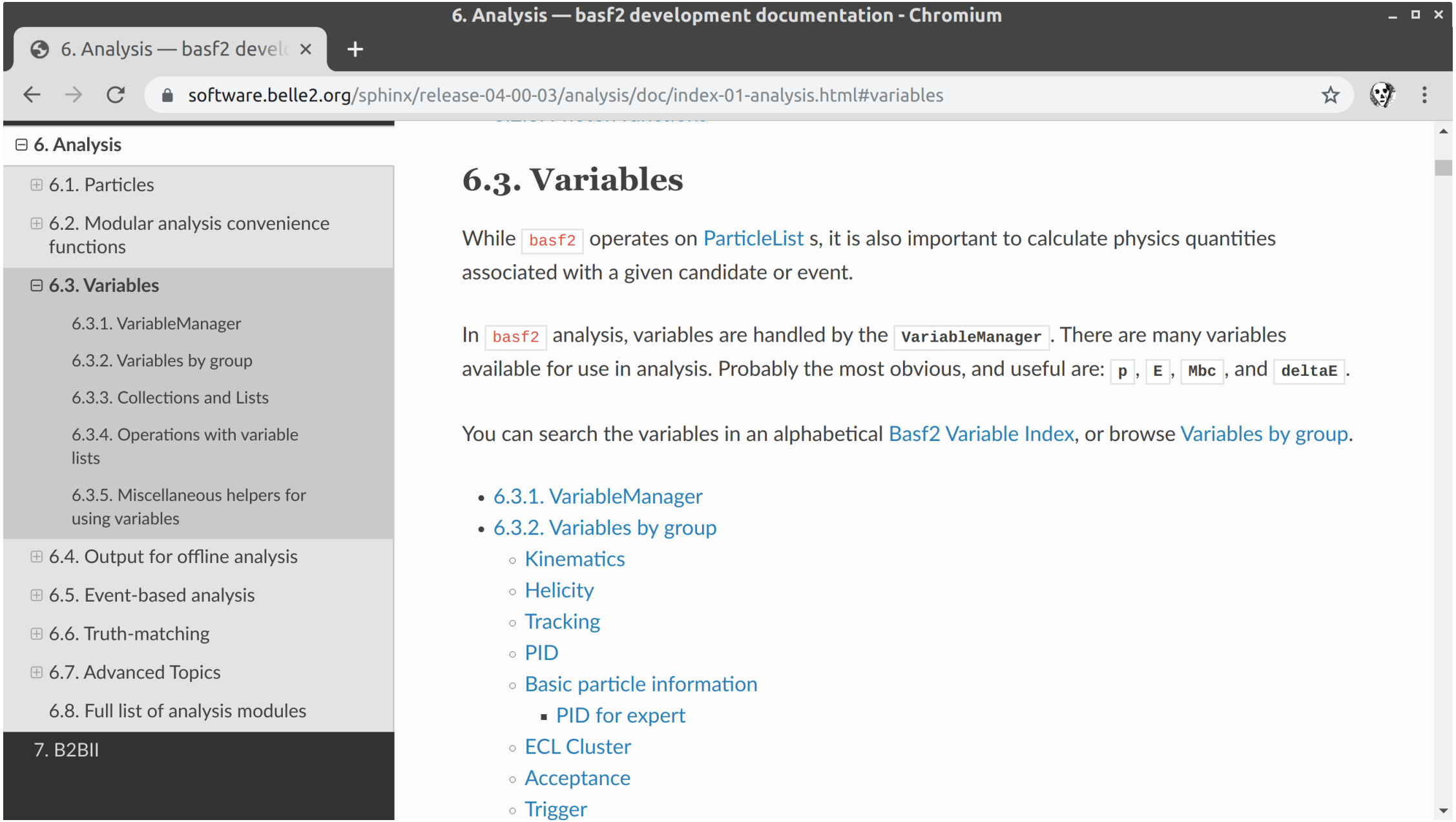
A typical path for an analysis job



The VariableManager

It manages variables

- The place where variables are calculated and evaluated. C++.
- Contains ***a lot*** of things that you might be interested in.
- Easily extensible (but what you want probably already exists).
- Many newcomers first point of entry into the code is tweaking or adding a variable.
  - We are *really* picky about clear documentation.
- Used for selections. And you've already seen it.
  - M, E, p, theta, phi, nTracks, …
  - clusterE, nChargeZeroTracks, nParticlesInList(), totalEnergyOfParticlesInList(), cosAngleBetweenMomentumAndVertexVector, ...


6. Analysis — basf2 development documentation - Chromium
6. Analysis — basf2 devel
software.belle2.org/sphinx/release-04-00-03/analysis/doc/index-01-analysis.html#variables
6. Analysis
6.1. Particles
6.2. Modular analysis convenience functions
6.3. Variables
6.3.1. VariableManager
6.3.2. Variables by group
6.3.3. Collections and Lists
6.3.4. Operations with variable lists
6.3.5. Miscellaneous helpers for using variables
6.4. Output for offline analysis
6.5. Event-based analysis
6.6. Truth-matching
6.7. Advanced Topics
6.8. Full list of analysis modules
7. B2BII
6.3. Variables
While basf2 operates on ParticleList s, it is also important to calculate physics quantities associated with a given candidate or event.
In basf2 analysis, variables are handled by the VariableManager. There are many variables available for use in analysis. Probably the most obvious, and useful are: p, E, Mbc, and deltaE.
You can search the variables in an alphabetical Basf2 Variable Index, or browse Variables by group.
6.3.1. VariableManager
6.3.2. Variables by group
Kinematics
Helicity
Tracking
PID
Basic particle information
PID for expert
ECL Cluster
Acceptance
Trigger

Aliases

Are awesome and you should use them

- I overhauled and cleaned up the documentation for the VariableManager.
- One thing worth mentioning: aliases.

```
from variables import variables as vm
vm.addAlias('cosPVtx',
            'cosAngleBetweenMomentumAndVertexVector')
```

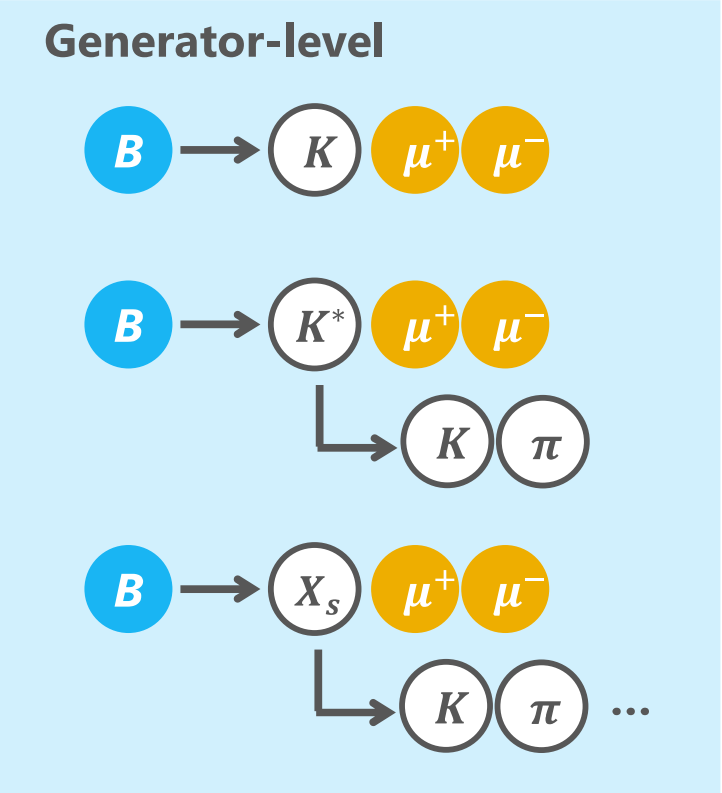
There is no path here!
The VariableManager exists alongside the path

New features of interest to inclusive and missing energy analyses

MC matching for inclusives

Yo Sato, SC; BII-4463

- Helper variables for sum-of-exclusive and *fully* inclusive mc-matching.
- Implemented new grammar: "@X".
- Deleted some unused grammar defined in DecayString.
- Plan to write a BELLE2-NOTE.

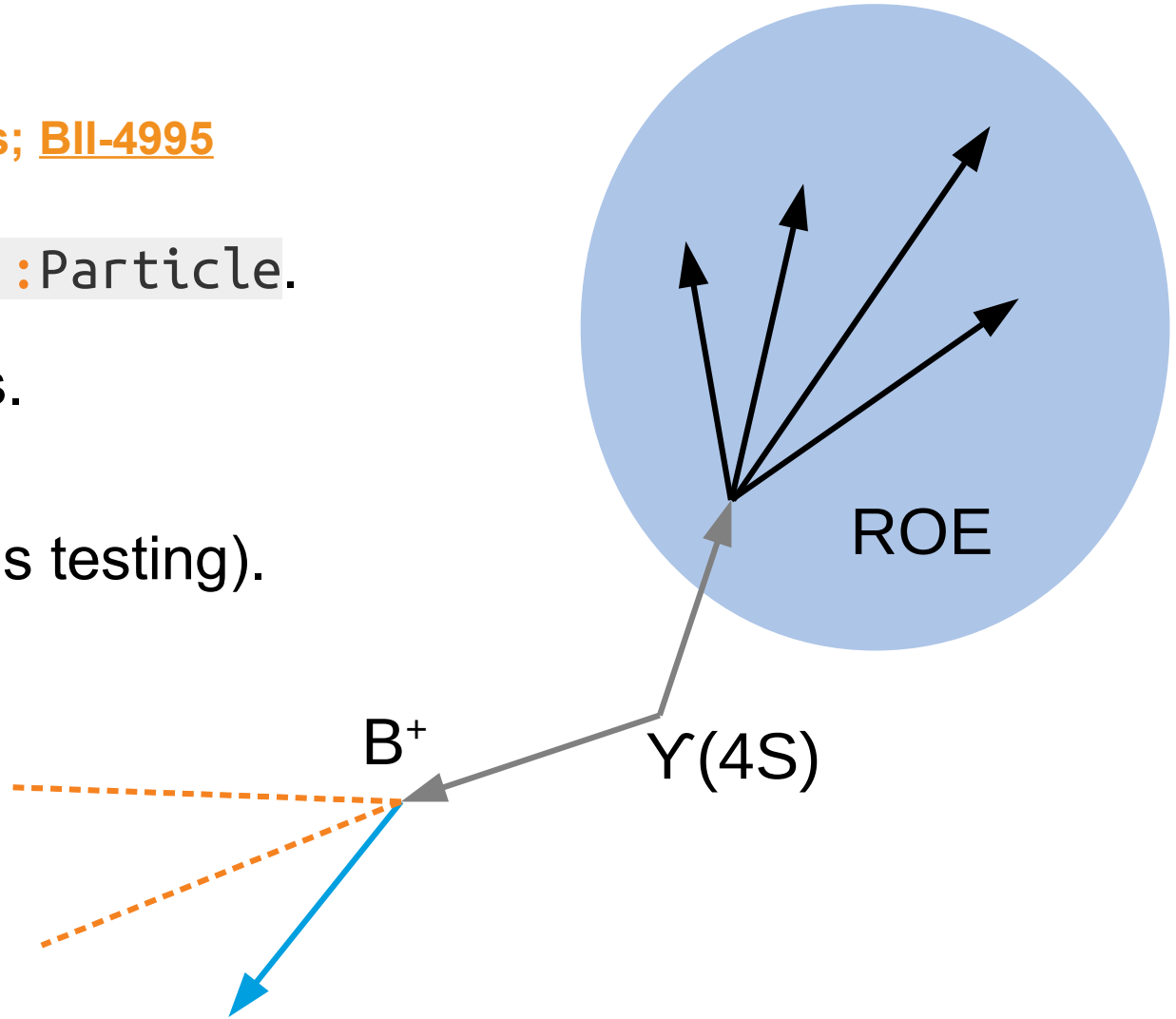



```
from modularAnalysis import reconstructDecay, copyLists
reconstructDecay('@Xsd:0 -> K_S0:all',                  cuts, path=mypath)
reconstructDecay('@Xsd:1 -> K+:loose pi-:loose',        cuts, path=mypath)
reconstructDecay('@Xsd:2 -> K+:loose pi-:loose pi0:all', cuts, path=mypath)

copyLists('Xsd:all', ['Xsd:0', 'Xsd:1', 'Xsd:2'], path=mypath)
reconstructDecay('B0:Xsdmumu -> Xsd:all mu+:loose mu-:loose', cuts, path=mypath)
```

ROEs and superparticles

Sviat Bilokin, Frank Meier, Will Sutcliffe, Peter Lewis; BII-4995

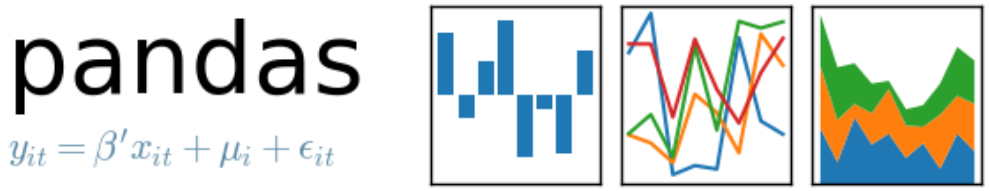
- Actually treat the ROE like a full `Belle2::Particle`.
- Useful in some missing energy analyses.
  - A new AllParticlesCombinerModule.
  - An ROE particle implementation (needs testing).
- Also: provide an official, supported solution for, e.g. vertex-fitting, with neutrinos (or other missing particles).




```
reconstructDecay("vpho:beam -> B+:sig ROE", "", mypath)
```

VariablesToHDF5

Martin Ritter, BII-3808



- “Just” a new module (python, for the moment).
- Output to HDF5 file containing a pytable.
  - Easily read in a `pandas.DataFrame`.
- **Won’t** immediately **work** on the grid (file extension will not be .root).

```
from b2pandas_utils import VariablesToHDF5
v2hdf5 = VariablesToHDF5(
        filename="variables.hdf5", listname="pi+:all",
        variables=list_of_interesting_event_variables)
path.add_module(v2hdf5)
```

Misc. tips

Misc. tips

And common pitfals

- Caution with global event calorimeter information.
  - Clusters are ***not*** unique. The energy depends ***on the hypothesis***.
  - The number of clusters / total energy is background dependent.
  - Make a simple selection and count/sum up “good” photons.
- Global event KLM information.
  - Don’t even.
- Use the ParticleLoader and use `Belle2::Particles` / `Belle2::ParticleLists`.
  - Raw mdst dataobjects are subtle, complicated and the probability of making a mistake → 1.
  - Analysis code shouldn’t really need to interact with them directly.
- Don’t open *dst files in root.
- You should not need to set a global tag in release-04. If you do something is probably wrong.
- Read the convenience functions in modularAnalysis.
- Complain (loudly, via bug reports) about missing documentation.

Misc. tips

And common pitfals

- Caution with global event calorimeter information.

```
from variables import variables as vm
from modularAnalysis import fillParticleList

good_photons = 'clusterNHits > 1.5 and E > 0.05'
fillParticleList('gamma:good', good_photons, mypath)
vm.addAlias('smartEECLNeutral',
            'totalEnergyOfParticlesInList(gamma:good)')
```

- You should not need to set a global tag in release-04. If you do something is probably wrong.
- Read the convenience functions in modularAnalysis.
- Complain (loudly, via bug reports) about missing documentation.

software.belle2.org

questions.belle2.org

Kudos

Stuff in these slides profited from ideas and revisions by these people:

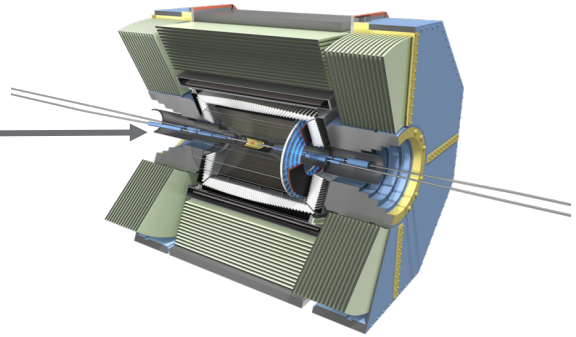
Jake Bennett, **Umberto Tamponi**,
Hannah Wakeling, and **Anže Zupanc**

Appendix

The big picture

Nature

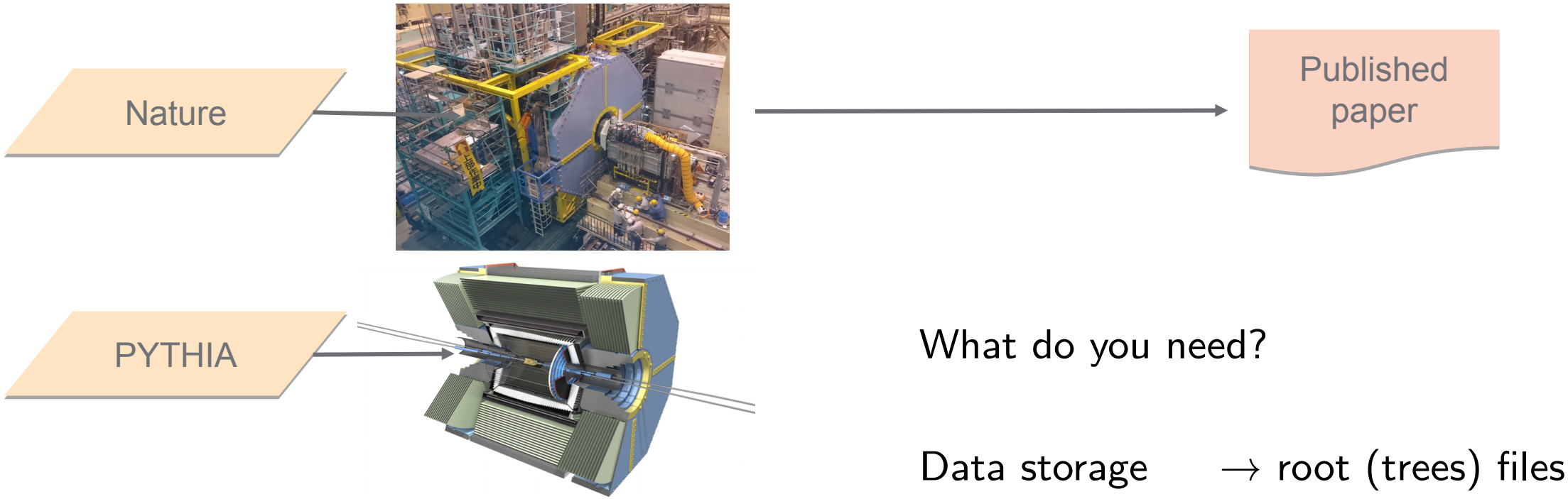
PYTHIA



Published paper

What do you need?

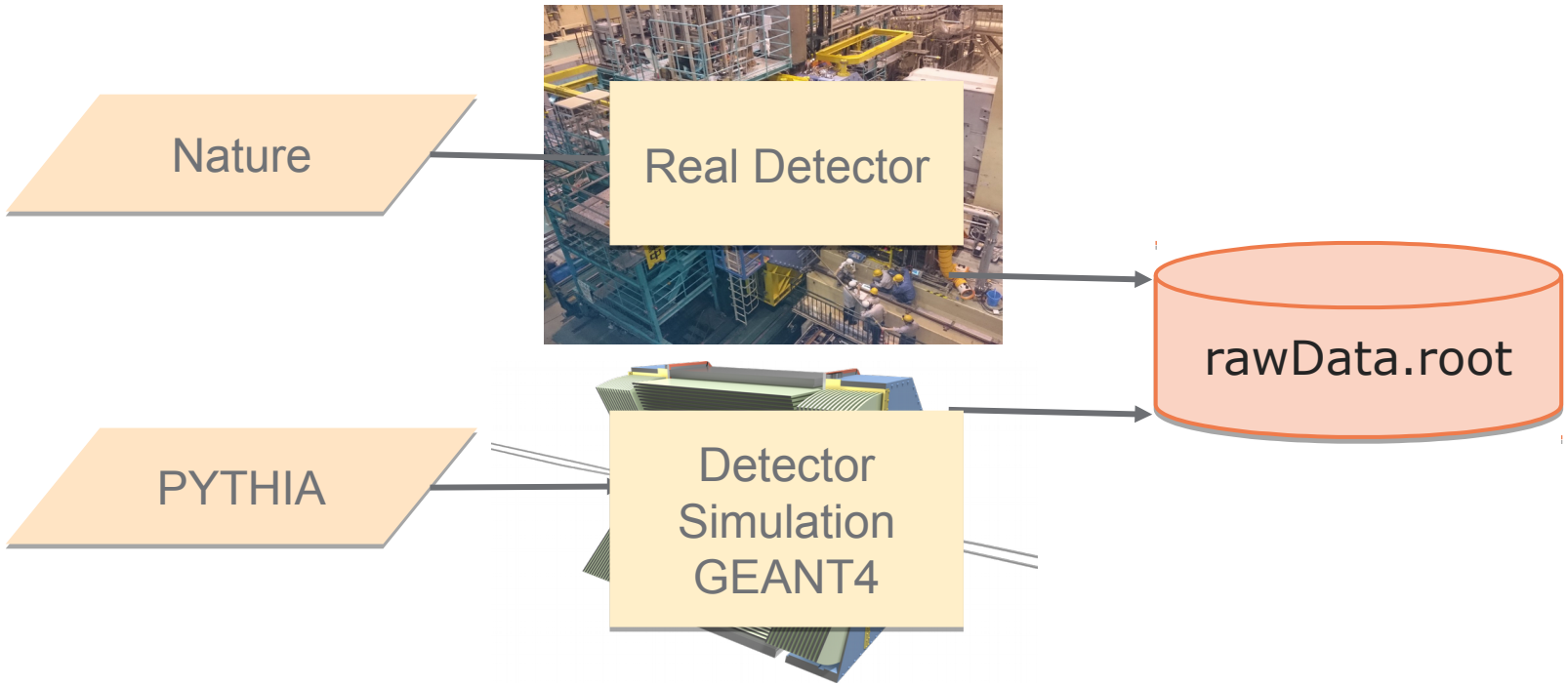
The big picture



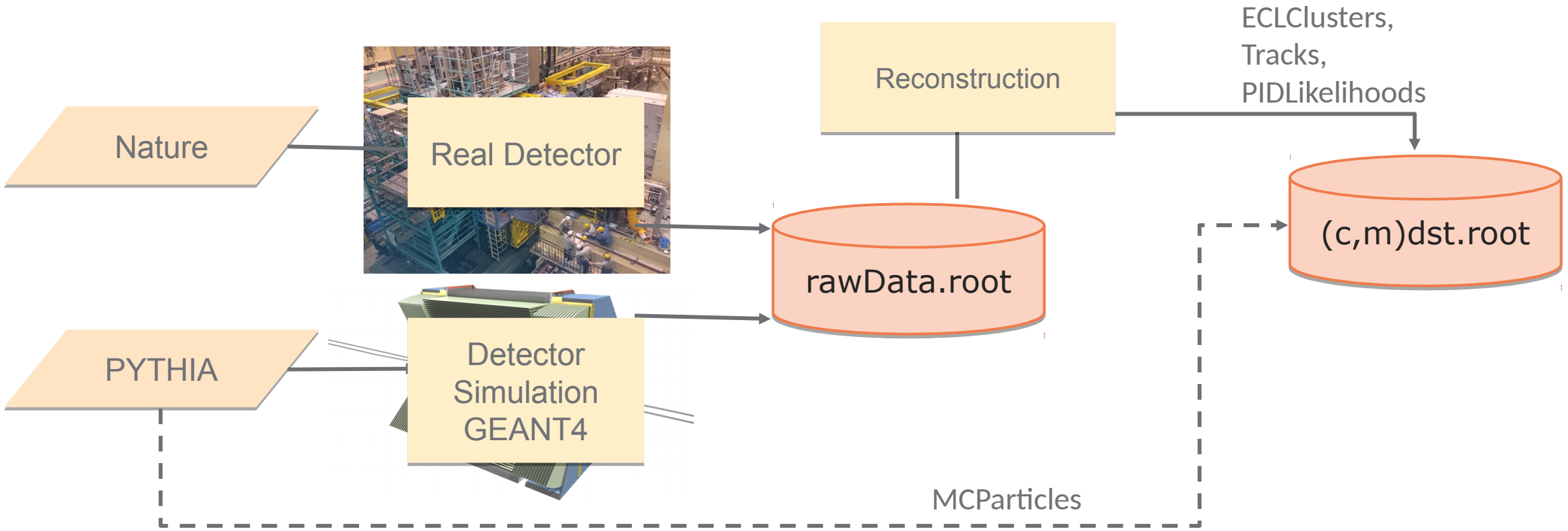
What do you need?

Data storage $\rightarrow$ root (trees) files
Data processing $\rightarrow$ C++ code
Scripting $\rightarrow$ Python

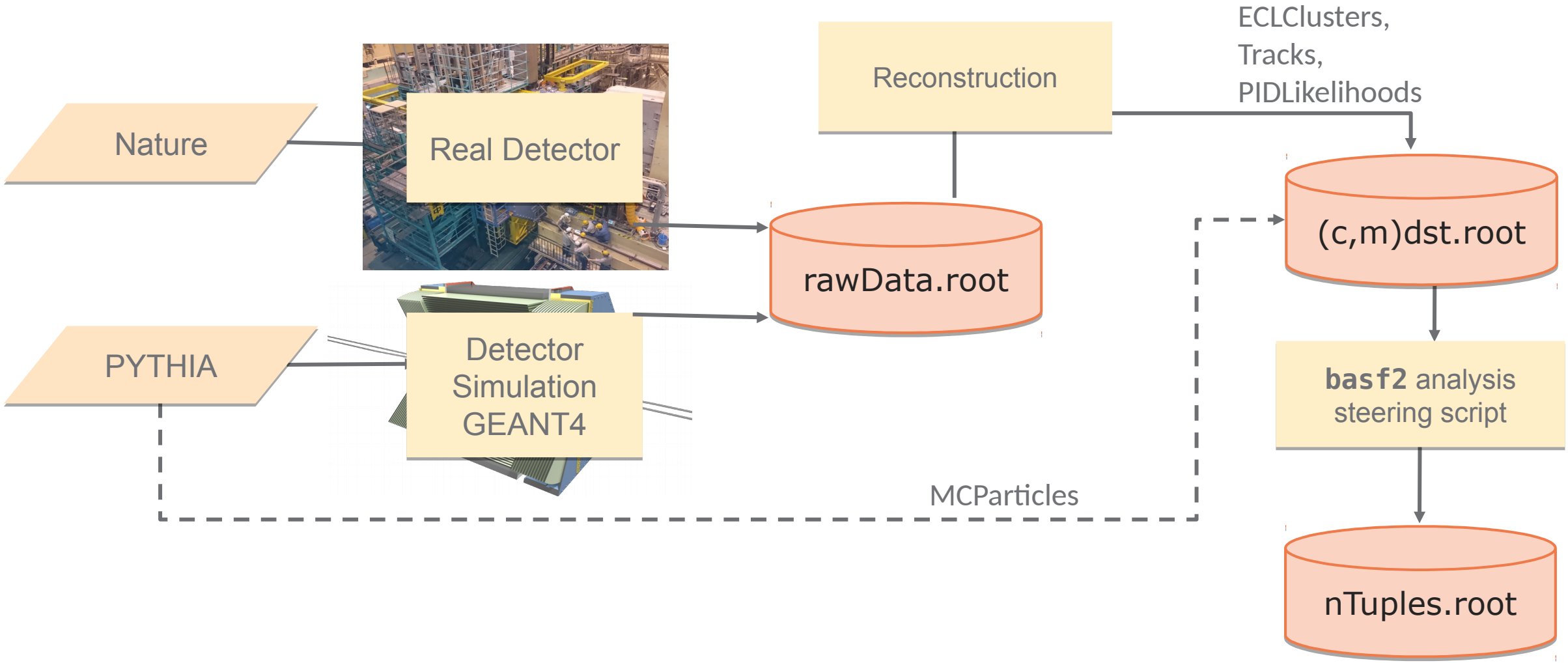
The big picture



The big picture



The big picture



The big picture

Nature

Real Detector

Reconstruction

ECLClusters,
Tracks,
PIDLikelihoods

rawData.root

(c,m)dst.root

PYTHIA

Detector
Simulation
GEANT4

basf2 analysis
steering script

MCParticles

Published
paper

Analysis note

Measurement

“Offline” analysis

nTuples.root

The big picture