Proposal of “roobasf” for software framework at Belle II

roobasf development team

R.Itoh (KEK), N.Katayama (KEK), S.H.Lee (Korea), and S.Mineo (Tokyo)

Outline

Definition of a software framework

- What framework DO care:
  * Execution of software components plugged into the framework for each event.
  * Event data I/O defined by a data model.
  * User interface for the execution control
  * Management of input and output data streams
  * Coordination of histograms/N-tuples used by components

- What DOES NOT care:
  * Geometry handling
  * Graphics like event display
  * Constants data base, etc.

  ..... They are supposed to be used from the plugged-in components and the interface through the framework have to be provided, but they are separate issues.

"roobasf" = "Marlin(+LCIO)" in ILC framework

1. Why do we need new software framework for Belle II?

- We are currently using “BASF” framework for all levels of Belle data processing: DAQ, offline reconstruction, MC production, skim to user analyses. It has been quite successful for the use over 10 years with its parallel processing capability.

- However, we think BASF is not good enough to be used in Belle II data processing.

- Main concern is the lack of capability to read and write C++ objects in a data file, even though we are already handling event data structure as C++ objects. Currently they have to be converted into Panther format for the I/O. (Unlike BaBar (or LHC), we did not choose to use Object database when BASF was developed (1996)).

- It's been only several years since Object IO using ROOT had become popular.

Other reasons to have new framework

- The interface for interactive user analysis is obsolete.
    * *Want to discard "HBOOK4"*
    * *Recent HEP community: "ROOT" is the de-facto standard.*

- Complicated parallel processing framework
    * *Different implementations in SMP and network cluster.*

- Lack of distributed input/output file management.
    * *No integrated catalog (name service) management.*
    * *No integrated GRID interface (ad-hoc interface exists, though).*

$\downarrow$

Development of new Belle II software framework!

2. Framework requirements and answers by roobasf

Priority 1: Object I/O and Data model

- The event structure is supposed to be described by a “Data model” written as C++ class definitions.

- Object I/O takes care of reading/writing objects in the “Data model” format.

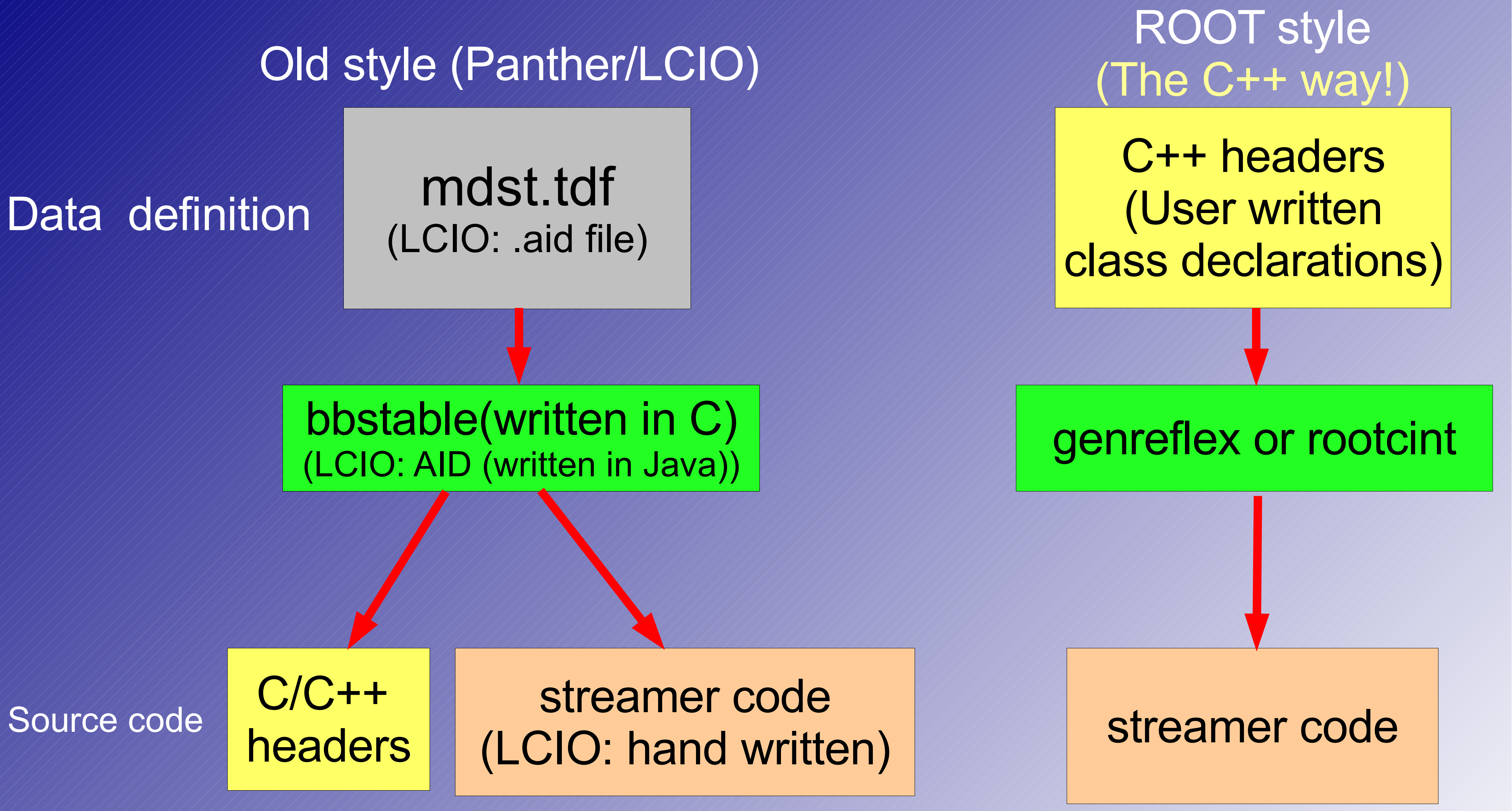
- The Data model will evolve as the software development goes and we will establish our own event Data model to satisfy our software requirements. Object I/O should not restrict the evolution.

- Easiness to add new objects to Data model without writing streamer code.

* ROOT I/O satisfies these requirements and is used as the Object I/O core in roobasf.

* The Data model is completely separated from roobasf.

Definition of structure in Data model and streamer code



Priority 2: Belle compatibility

- To make maximum use of the existing Belle library for the development of Belle II software, we require full compatibility with the Belle library

   <- Belle library is a "treasure box" for Belle II software development.

- The compatibility enables smooth transition from Belle to Belle II.

- Two levels of compatibilities are realized in roobasf.
   a) Module written for BASF can be used on roobasf w/o mods.

   b) Capability to access Panther data by the framework although it is restricted to read-only. We don't want to keep using Panther for Belle II.
- The user interface is fully compatible with BASF.

BASF users can easily move to roobasf
without making any modifications to existing codes.

Priority 3: DAQ compatibility

- The framework is supposed to be shared with DAQ to ensure the same software development environment, which is already achieved by BASF in Belle.
  (ex. sharing of reconstruction codes between offline and HLT.)

- To use the framework in DAQ, special treatments are required such as the implementation of the network-socket and shared memory interface.

↓

* roobasf is designed/developed considering the sharing with DAQ from the beginning.

* roobasf is a complete “in-house” development of Belle II by two permanent KEK staffs + two Belle II students.

* Current roobasf is only ~6,000 lines of pure C++ code.

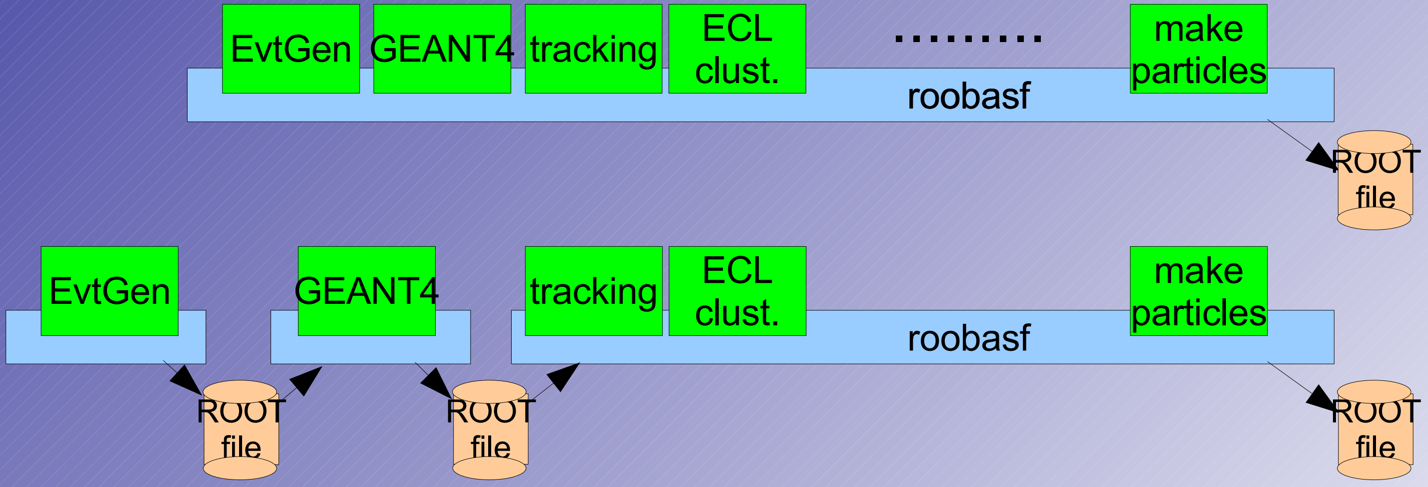
We can make any modifications to framework on request in a timely manner!

Priority 4: Seamless software bus

- By the definition of the software framework, every functional component in Belle II is required to be pluggable to the same framework and work together seamlessly.

- The processing chain can be stopped and restarted at any point by dumping the intermediate objects in a data file.

* roobasf is built to satisfy this requirement as well as BASF by using ROOT I/O and Belle II Event Class.



Priority 5: Parallel Processing

- Current BASF has a built-in parallel processing capability which is quite successful. It is useful to make use of multicore CPUs.

- We require the same parallel processing mechanism to have in the new framework for Object I/O.

- It is expected to be compatible with the use in DAQ.

↓

* roobasf is already capable of parallel processing on multi-core CPUs with ROOT I/O. Implementation is almost completed.

* Parallel processing for network-connected PC cluster is being developed now using MPI (Message Passing Interface).

* They are compatible with the use in DAQ and HLT.

3. Prototype of roobasf: current status of development

- We have released alpha-version prototype of roobasf three weeks ago. It is available on KEKB computer (bwg).

- The evaluation package includes whole Belle library and roobasf is implemented as a part of it.

- roobasf is implemented with an interface to the “Event class” which is capable of managing any kind of objects including but not restricted to Panther data structure.

- Simple examples come with the package.

Prototype implementation : core of roobasf

1. Module driver:
   - Software Bus architecture inherited from BASF
   - Modules for roobasf can be written in the same way as that for BASF.

2. Object I/O with parallel processing capability:
   - Capability to read and write C++ object using ROOT I/O.
   - Performed for a single "event" object (Singleton accessed through Event* Event::Instance().)
   - Object data distribution/collection over shared memory for parallel processing.

3. Scripting:
   - modified BASF's message parser

4. Histogram management with parallel processing capability
   - ROOT histogram/N-tuple management a la BASF
   - Histogram/N-tuple collection from multiple event processes
   - Compatibility interface with existing BelleTuple/BelleHistogram.

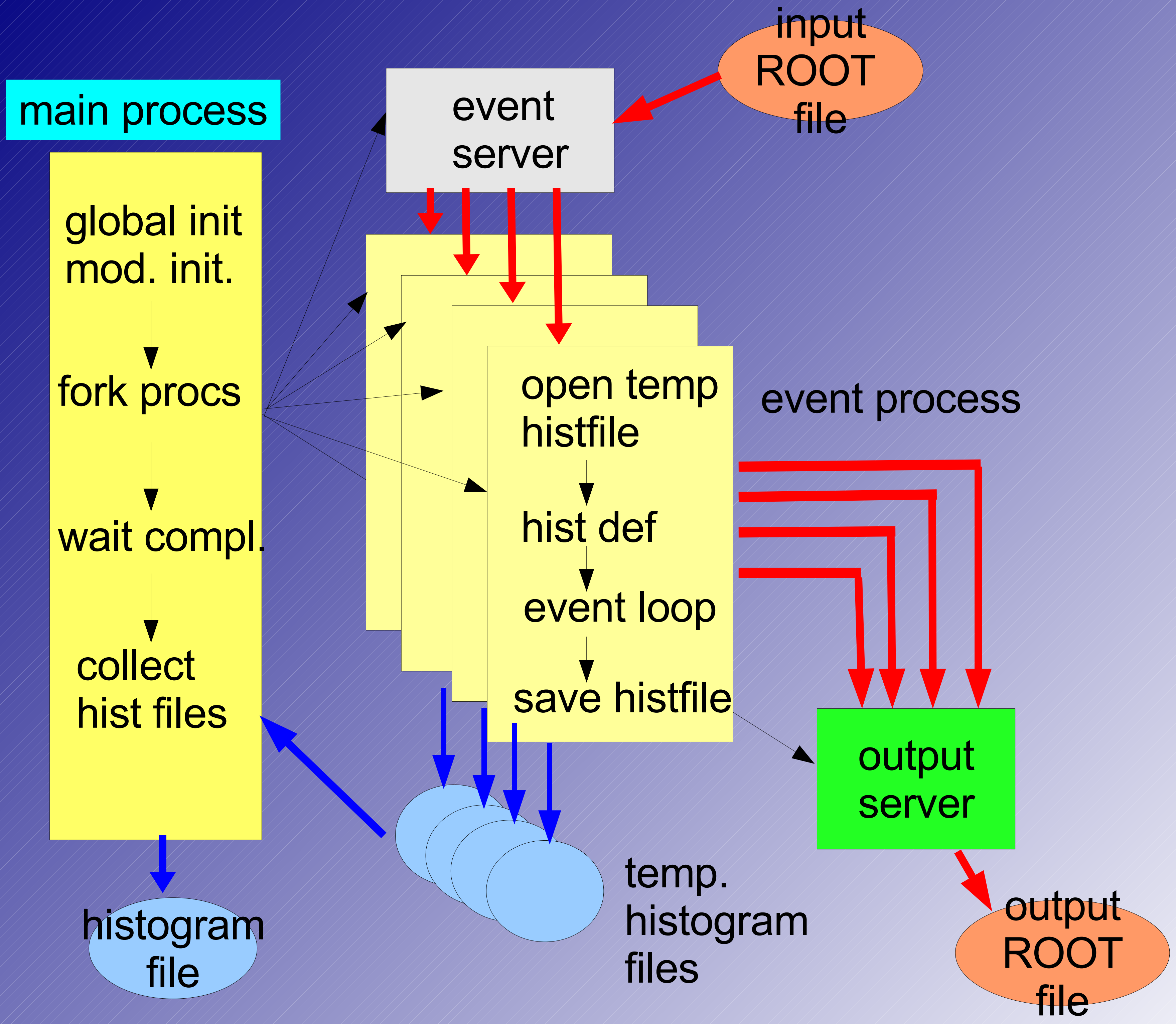
Parallel processing

- “process” based implementation. “thread” is not used at all.
  -> No restrictions in user code for parallel processing.

- Two operation modes:
  a) single process mode (nprocess=0)
    - roobasf is executed as a simple application program running in a single process. -> suitable for debugging

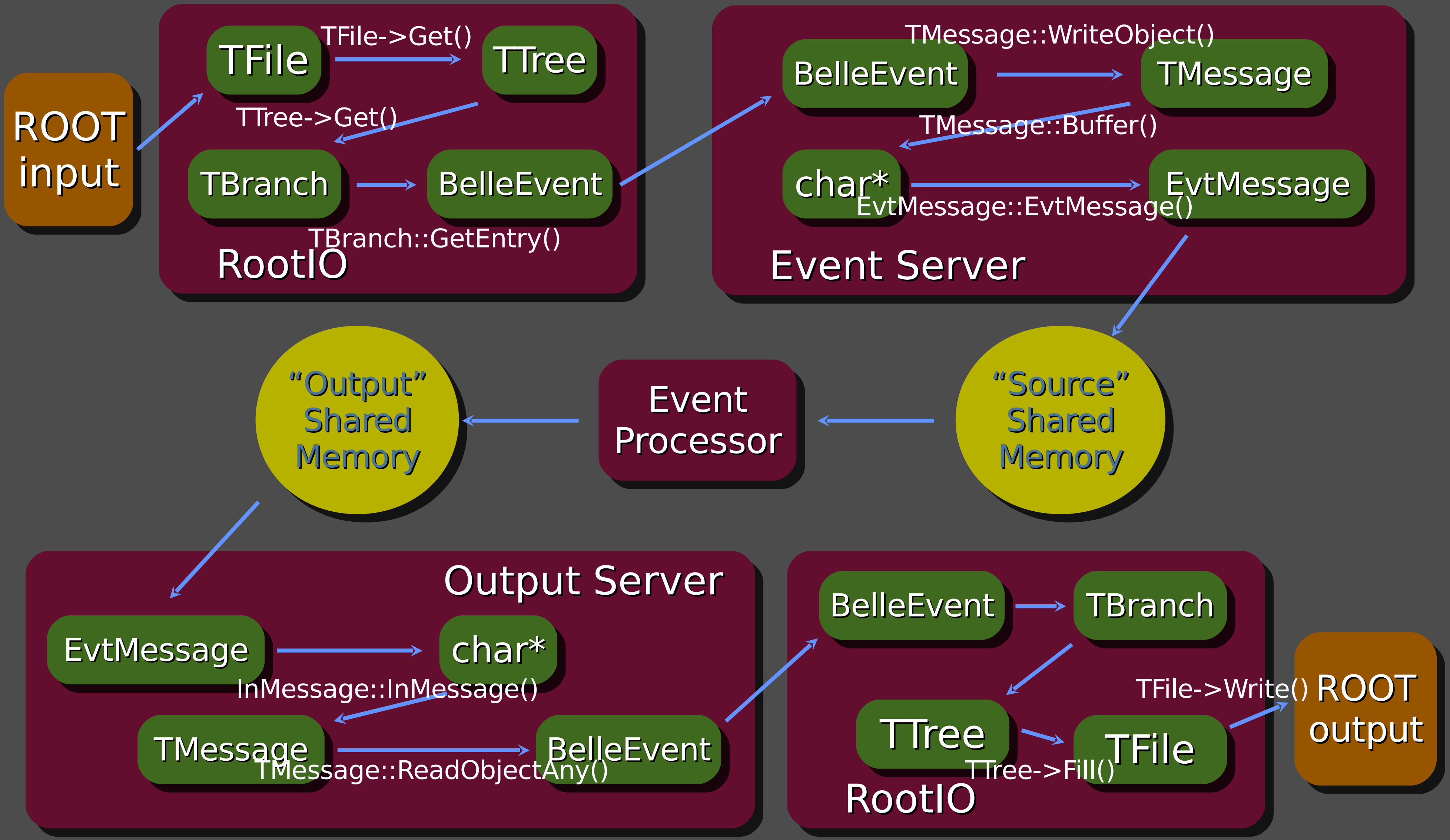
  b) multi-process mode (nprocess>0)
    - roobasf is executed with multiple processes
    - Event objects read from a file are distributed to the processes through a ring buffer.
    - event processing is performed in parallel by the processes.

- Parallel processing is now supported for
  a) ROOT file input and output.
    * Multiple ROOT outputs is not yet supported.
  b) Panther input files with full interface to remote file access (zfserv) and index I/O. No support for output.


main process
global init
mod. init.
fork procs
wait compl.
collect hist files
histogram file
event server
input ROOT file
open temp histfile
hist def
event loop
save histfile
event process
temp. histogram files
output server
output ROOT file

Data Flow Scheme

S.H.Lee



- Current example:
  3 step example based on my simplified $D\bar{D}$ mixing analysis code.

  * *convert*: Conversion of a Panther MDST file to a ROOT file with
    vector::<Particle*> stored in Event object.

  * *recon*: Reconstruction example. Read the ROOT file,
    make a list of reconstructed particles in new
    vector::<Particle*>, and store them in a separate ROOT file.

  * *vtxtime*: Analysis example. Read the ROOT file and
    make histogorams/N-tuples in a separate histogram file.

- The package is distributed as a version of Belle library
  with BELLE_LEVEL=rio-x.y.z. The latest version is 0.5.4.

- See Wiki for the usage : http://wiki.kek.jp/display/sbelle/Roobasf

Execution example: “recon”

- parallel processing with nprocess=4

```
!path add_module main Panther2event
path add_module main recon

nprocess set 4
initialize
histogram_root define recon_hist.root
output_root open recon_out.root
process_root ../convert/mc_kpi.root
terminate
```

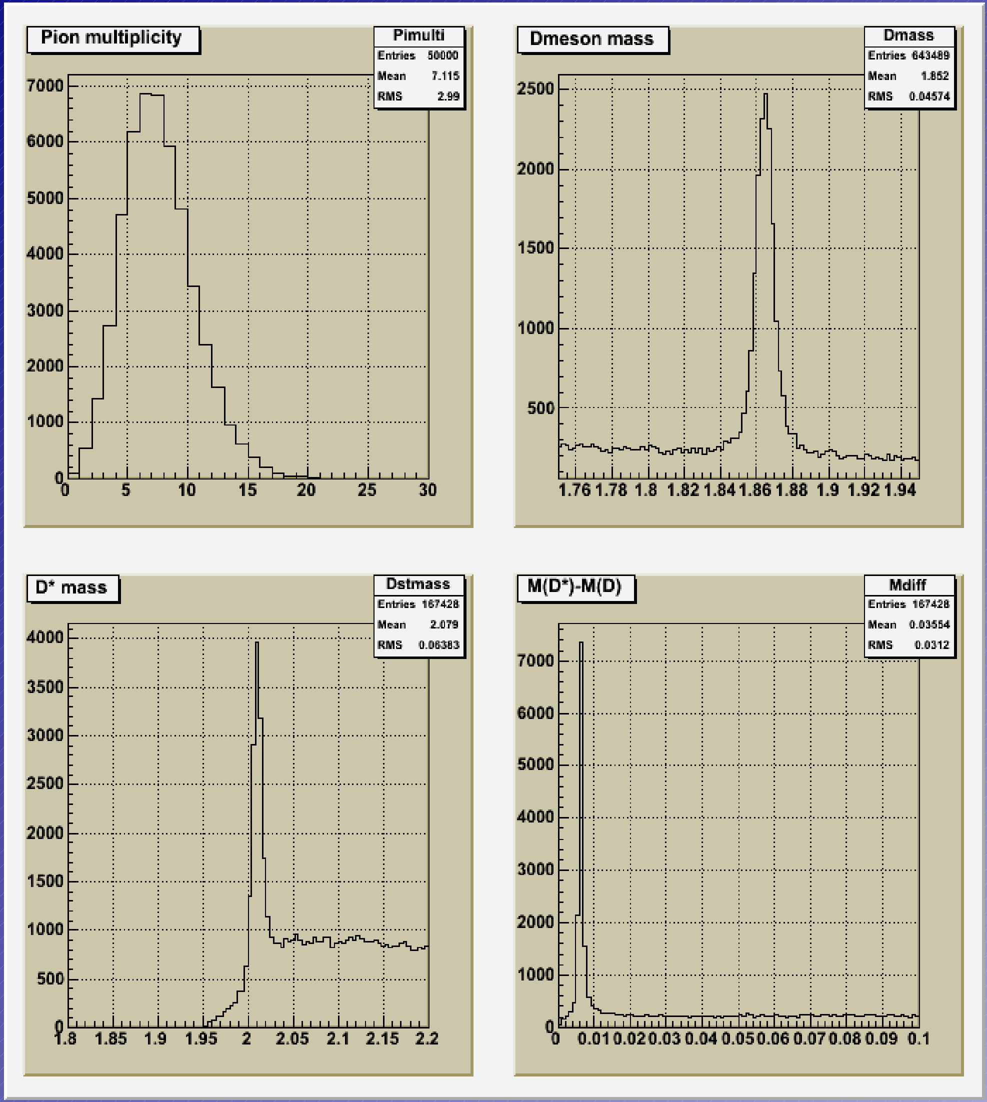
* Almost the same as BASF's script except several new commands:

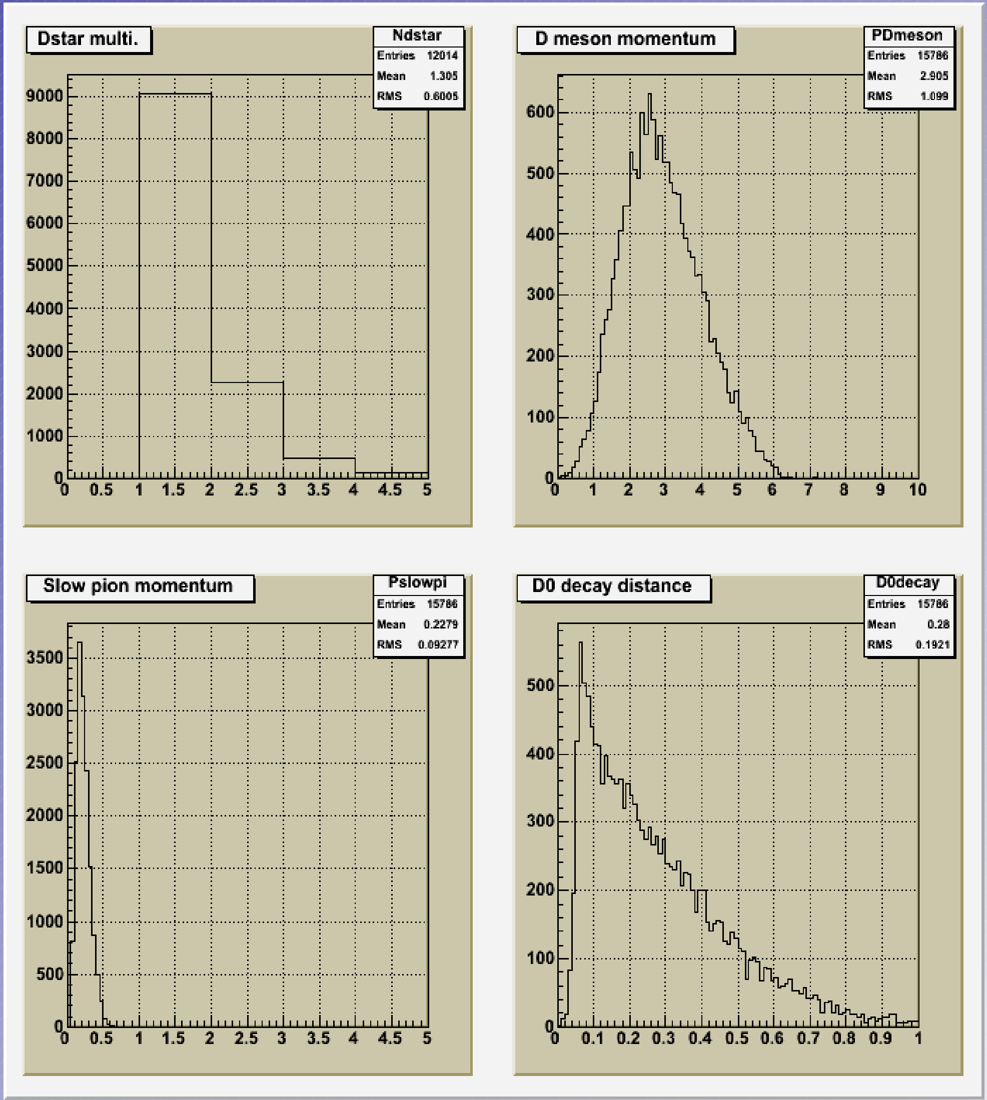
`histogram_root`	: open a ROOT histogram file and call hist_def() functions of modules.
`output_root`	: open a output ROOT file.
`process_root`	: start event loop for a given ROOT input file

“proof” that prototype is really working!: Execution log of “recon”

```
bwg04.bnet.kek.jp% more test.log
basf: user interface [roobasf_if] is selected
basf: initialization function [basf_compat] is selected
roobasf> roobasf> path: main
module: recon
roobasf> roobasf> roobasf> roobasf> roobasf> roobasf> roobasf> root_hist_on : np
rocess = 4
Framework: histfile recon_hist.root is registered in RbTupleManager
RootIO: file recon_out.root is opened for write access.
output server forked off.....
roobasf> RingBuffer initialization done
rocHandler: output server forked. pid = 13356
roobasf> RingBuffer initialization done
buftop = 2977d0e0, end = 2d486000
ProcHandler: event server forked. pid = 13357
rootread : tree got.
rootread : Branch event got; m_branch=0dc44ab0
rootread : Entries = 50000
ProcHandler: event process 0 forked. pid = 13358
ProcHandler: event process 1 forked. pid = 13359
ProcHandler: event process 2 forked. pid = 13360
ProcHandler: event process 3 forked. pid = 13361
RbTupleManager: initialized for event process 0 (pid=13358)
RbTupleManager: initialized for event process 2 (pid=13360)
RbTupleManager: initialized for event process 3 (pid=13361)
RbTupleManager: initialized for event process 1 (pid=13359)
baf::process_root: begin_run processing done
baf::process_root: begin_run processing done
baf::process_root: begin_run processing done
baf::process_root: begin_run processing done
event_server: sending EOF...
event_server: sending EOF...
event_server: sending EOF...
event_server: sending EOF...
ROOTIO: closing ROOT file.
rocHandler : event server 13357 completed and removed
ProcHandler : event process 13358 completed and removed
ProcHandler : event process 13360 completed and removed
ProcHandler : event process 13361 completed and removed
ProcHandler : event process 13359 completed and removed
Processing time: 84.490 sec for 11723 events
Processing time: 79.810 sec for 11070 events
Processing time: 95.700 sec for 13163 events
Processing time: 101.390 sec for 14044 events
Output server - EOF received; exit
output server terminated
ROOTIO: closing ROOT file.
roobasf> ProcHandler : output server 13356 completed and removed
RingBuffer: Cleaning up IPC
```

“proof” that prototype is really working! : histograms collected from 4 event processes.





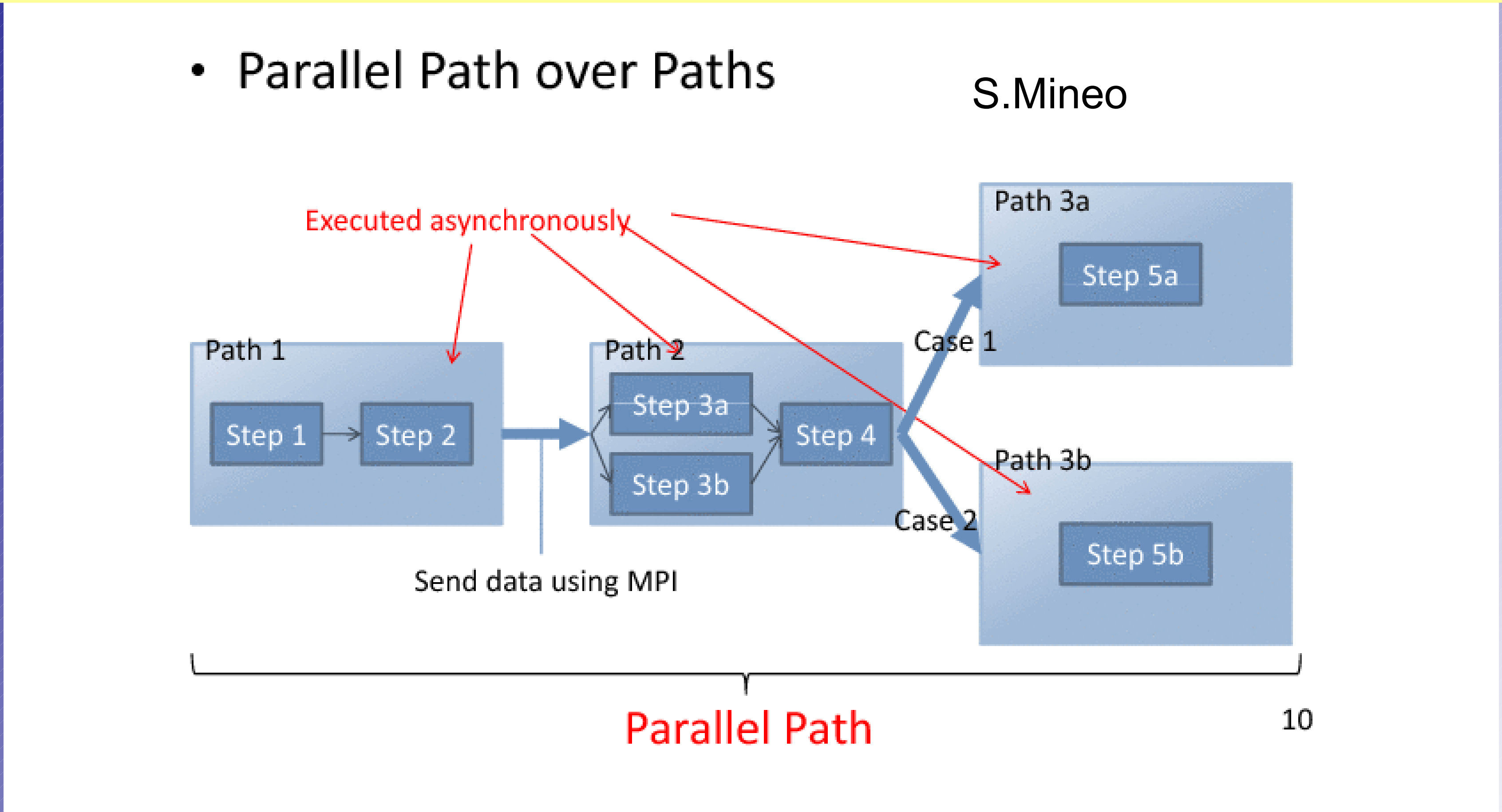
“recon” histograms

“vtxtime” histograms

4. Where do we go? Final goal of roobasf

- In addition to the debug/support of current version, we are planning to implement followings in roobasf.

a) Parallel processing with a network-connected PC cluster with a capability of pipeline parallel.
-> originally required in HSC analysis, but also useful in Belle II HLT.



b) Use of Python as scripting language

- Python is getting very popular in HEP software as programming language.

- Many recent HEP software provides Python interface :
  PyROOT,GaudyPython...

- How about PyRooBasf?

```
path add_module main fix_mdst
path add_condition main <:0:EXIT
process_root_url HadronB:e57.r1..200
terminate
```

roobasf script

```
import PyRooBasf
f = Framework()
fix_mdst = Fix_mdst()
main = Path()
main.add(fix_mdst)
main.add_condition('<:0:exit')
f.add(main)
i = Input(exp=57,
          run=range(1,200),
          type='HadronB')
f.process(i)
```

Python script

You can add your own histogram manipulation code using PyROOT hereafter, for example.

c) Remote ROOT file access over GRID + database management

- We would like to implement the GRID interface in roobasf, but we need to wait for the discussion on the distributed computing.

- Implementation of AMGA interface could be the test case and we will think about it.

- POOL used by LHC experiments might be a candidate for input/output database management

d) Refactoring of the code

- Current roobasf implementation still largely owes existing BASF code and is not optimized. (ex. similar code appears repeatedly and not well modularized.)

↓

Re-coding with better class implementation +
dynamic loading of components.
ex. Recycling the same I/O base class for ROOT and Panther
Dynamic load of the I/O package on request

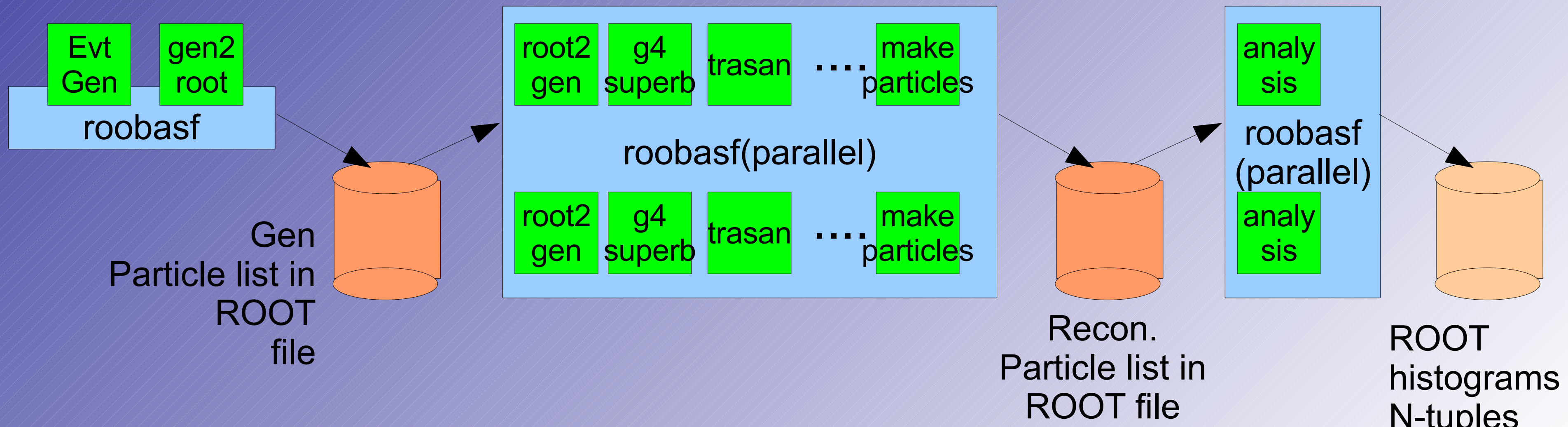
Recent progress and Plan

- Now trying to build an environment to run Belle II MC simulation on roobasf.

- Recycle BASF modules “evtgen”, “g4superb”, “trasan”, “TRAK”, “rececl”, “rec2mdst”, and also “fsim6”

- The simulation modules (g4superb and fsim6) and “rec2mdst” will be modified to output Objects such as Hits, Digis, and Tracks to learn how we go about our Data model.

- It will be released to the collaboration (Belle and Belle II) as an evaluation kit of roobasf by the end of July.



Some comments

a) Belle compatibility

- We totally think the Data model in Belle II should have NO dependence on Panther any more. We don't want to use Panther for Belle II.

- Some of you may worry that having the capability to read Panther data may prevent our transition.

  -> Our answers are:

  1. roobasf cannot write Panther files although they can be read. This forces users to use Belle II Event class to pass data structure between modules.
  2. We will start replacing Panther output of modules with Event class objects from upstream like EvtGen, g4superb, fsim6.... Then users have to modify their code to read objects.

- What we intend is the “smooth transition” from Belle software as described by Katayama-san in comp/soft session. The careful use of roobasf with new Event class makes it possible.

b) Compatibility with Marlin(ILC framework)

- There might already exist some software (tracking, pixel reco...) developed with Marlin(ILC framework) and we don't want to scratch them even when roobasf is chosen.

- It is possible to provide a simple “wrapper” to plug-in Marlin processor into roobasf without any change.
  * At a first look, Marlin processor(=module) interface looks quite similar to that of roobasf.

- The interface to LCIO can easily be provided just like Panther I/O.

- But it should be just a “legacy” implementation like Panther.

c) Geometry and database

- Let me emphasize it again that geometry, database, etc. are completely SEPARATE issues from the framework discussion. We should NOT decide Belle II framework from these viewpoints.

- Geometry is a difficult problem; as you know, Atlas group has as many geometry systems as the number of sub detector groups.

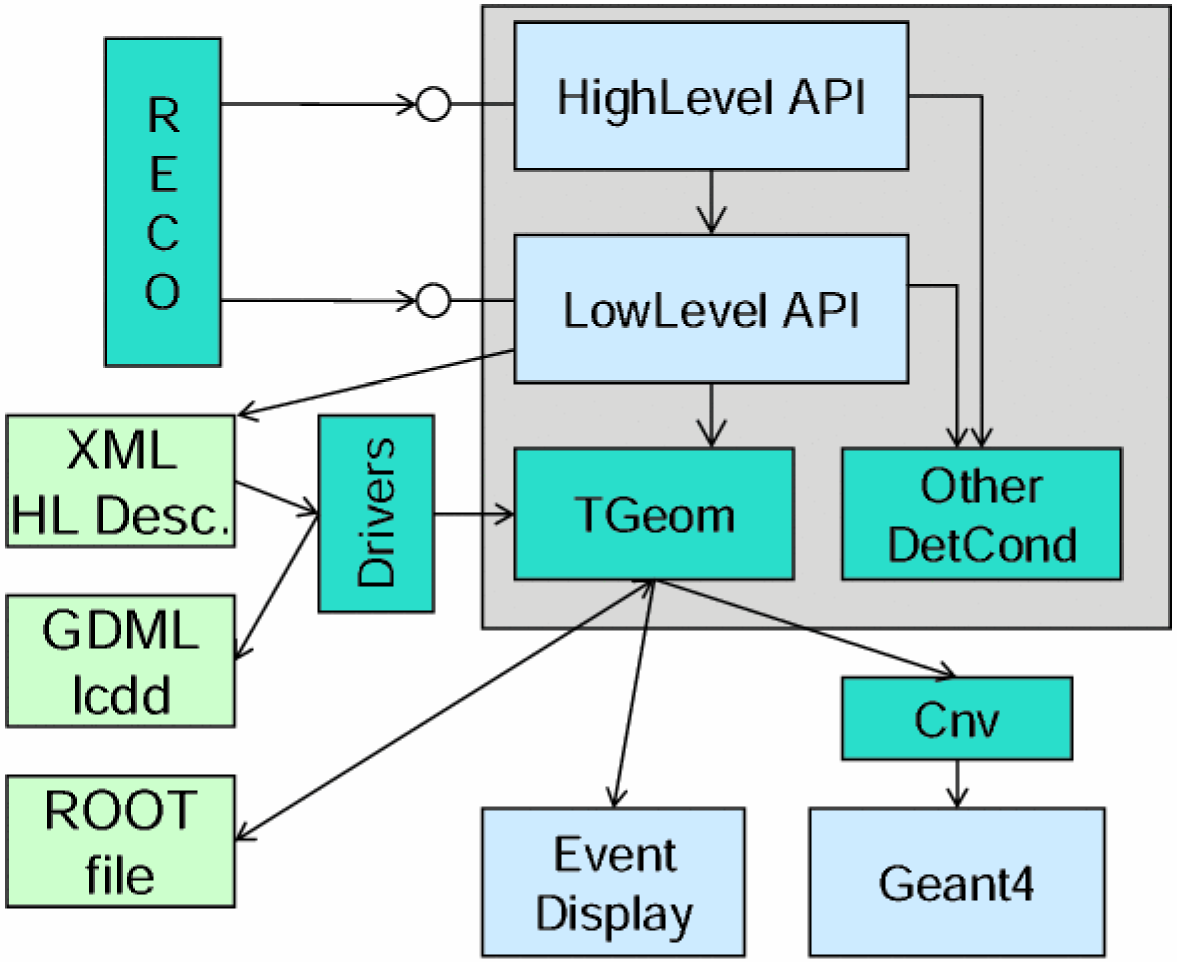
- We should start the geometry working group ASAP.

- Since we will rely on ROOT in Belle II framework, the decision of geometry, database, etc is strongly affected by CERN's approach.

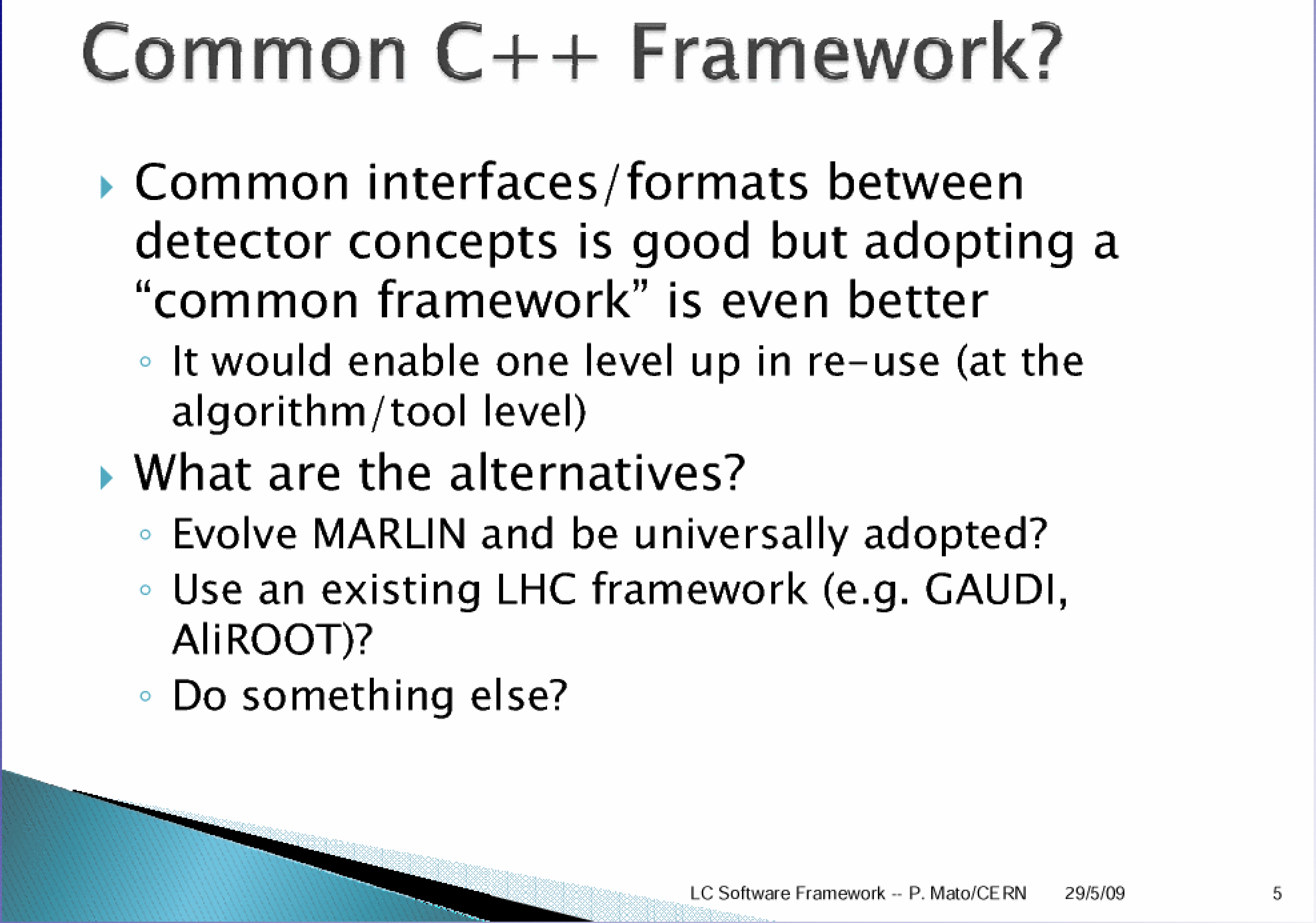
- CLIC people are now joining in ILC/ILD. The current ILC software will likely to be tossed about and may be unstable in coming years. Check slides in ILC Software WS at CERN. http://indico.cern.ch/conferenceDisplay.py?confId=58717

- We should carefully watch CERN's movement and decide our geometry manager, database, etc.

Geometry Proposal



P.Mato(CERN) @ LC software WS (May '09)

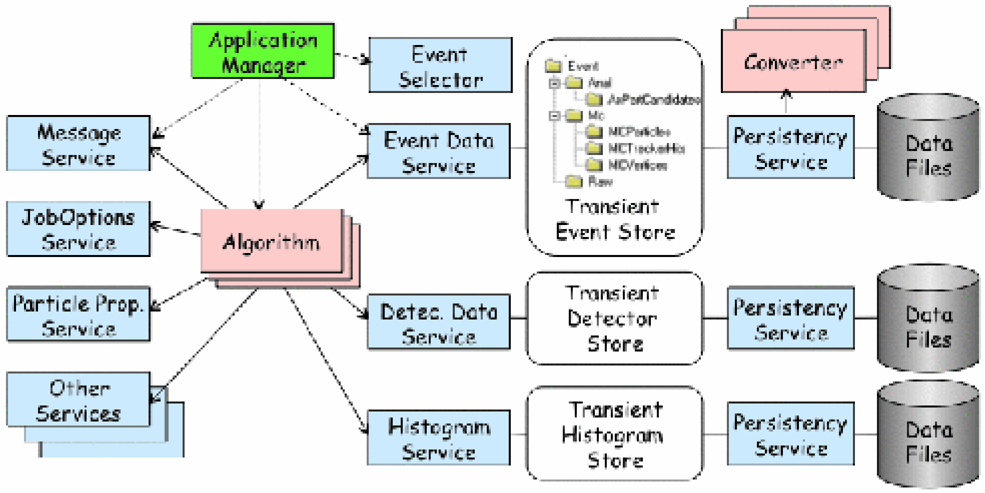


Common C++ Framework?

- Common interfaces/formats between detector concepts is good but adopting a "common framework" is even better
  - It would enable one level up in re-use (at the algorithm/tool level)
- What are the alternatives?
  - Evolve MARLIN and be universally adopted?
  - Use an existing LHC framework (e.g. GAUDI, AliROOT)?
  - Do something else?

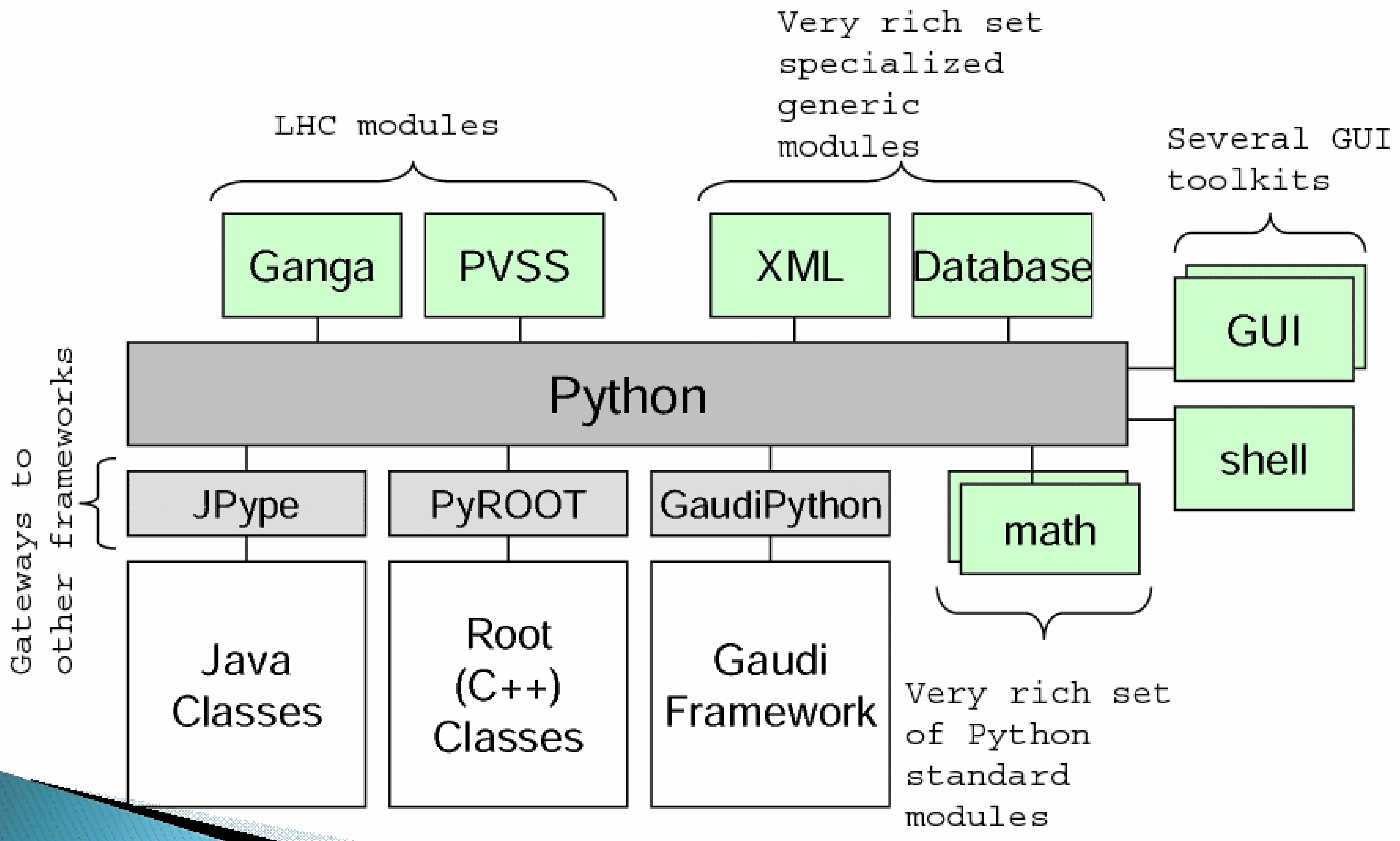
Example: GAUDI Framework

- GAUDI is a mature software framework for event data processing used by several HEP experiments
  - ATLAS, LHCb, HARP, GLAST, Daya Bay, Minerva, BES III,…
- The same framework is used for all applications
  - All applications behave the same way (configuration, logging, control, etc.)
  - Re-use of 'Services' (e.g. Det. description)
  - Re-use of 'Algorithms' (e.g. Recons -> HTL)
- Equivalent to MARLIN



P.Mato(CERN) @ LC software WS (May '09)

What could be done



5. Summary

- “roobasf” was designed and developed to satisfy requirements:
  * ROOT I/O based object I/O separated from Data model
  * Full compatibility with Belle software/data
  * Versatile support by dedicated Belle II staffs+students for DAQ
  * Seamless software pipeline
  * Parallel processing capability

- Alpha version prototype has been released already.
  Preparing evaluation samples (beta release) for the development of GEANT4 simulation and also FSIM based studies.
  -> will be released by the end of this month.
  -> we hope roobasf is used by the detector performance study group.

- If you see any short comings in roobasf, please let us know. We will happy to improve roobasf prototype to make it the framework for Belle II.