LoKi & Bender
User Friendly Physics Analysis Tools

Vanya BELYAEV  NIKHEF, Amsterdam,
on leave from ITEP/Moscow
• The modern HEP experiment in the coming "LHC-epoch" are large and enormously difficult & complicated
  • Many different & sophisticated techniques for particle/jet detection and identification
    • & simulation
  • Many different reconstruction techniques and methods
  • Many physicists & Software engineers work hard on the software development
    • In parallel ...
  • Huge data samples ($\geq 10^{10}$ /year) are expected
• Complexity of the problem demands the outstanding powerfulness of the software, and (unavoidably) the complexity

• C++ “de-facto” standard of the software for LHC epoch

• No real alternative for reconstruction & simulation task
  • (almost) no generic way to make these tasks “simple”

Should it be the only one option for physics analysis?
• What bad in usage of C++ for physics analysis?
• C++ is not something very simple
• Some initial learning period is required:
  • learning curve is non-trivial, and does take time
  • C++ requires some discipline & training
  • C++ offers many non-trivial ways for solution of many difficult problems
  • Often a great fun for young students…

What about the senior physicists?
The practice shows that as concern the epoche of C++ dominance many experienced senior physicists are cut off from the real physics analysis.

Few exceptions are also obvious:

“Hi, Bill! Would you be so kind to prepare HBOOK-ntuple with the \( <a, b, c, \ldots, x, y, z> \) variables for me around tomorrow lunchtime?”

Is it the only way to reuse their great physics experience?

Are there some other exceptions?
Counterexample?

**KAL by genius Hartwig Albrecht**

- Script-like file
- All technical details are well hidden from end-users
- Transparent physical content of the code
- Looping, histograms, N-tuples, MC truth – at most 1 line!
- Typical analysis program ~ 50-70 lines
- All senior persons, including the spokesman successfully participated in physics analysis

```plaintext
HYPOTH E+ MU+ PI+ 5 K+ PROTON

IDENT PI+ PI+
IDENT K+ K+
IDENT E+ E+
IDENT PROTON PROTON
IDENT MU+ MU+

SELECT K- pi+
  IF P > 2 THEN
    SAVEFITM D0 Dmass 0.040 CHI2 4
  ENDIF
ENDSEL

SELECT D0 pi+
  PLOT MASS L 2.0 H 2.1 NB 100 @
  TEXT ' mass D0 pi+ '
ENDSEL

GO 1000
```
**NIKHEF**

**Is the goal achievable with C++?**

- Majority (but me) is convinced that C++ features (verbosity, static nature etc) do not allow to use it as friendly language for physics analysis.

**GPATTERN package by Thorsten Glebe (HERA-B)**

- Native C++
- Easy, readable and very efficient

---

**TrackPattern PiMinus = pi_minus.with ( pt > 0.1 & p > 1 );**

**TrackPattern PiPlus = pi_plus.with ( pt > 0.1 & p > 1 );**

**TwoProngDecay kShort = K0S.decaysTo ( PiMinus & PiPlus );**

**kShort.with ( vz > 0 );**

**kShort.with ( pt > 0.1 );**
Try to merge the best ideas: LoKi

- **KAL** by Hartwig Albrecht
  - '1-line' semantics
  - Predefined variables
- **GPattern and GCombiner** by Thorsten Glebe
  - Cuts and patterns

- **HepChooser and HepCombiner** from obsolete CLHEP
  - Combinations, loops
- **Loki** by Andrei Alexandresku
  - Functions, name and spirit

```cpp
select ( "K-" , ID == "K-" && CL > 0.01 && P > 5 * GeV ) ;
select ( "PI+" , ID == "pi+" && CL > 0.01 && P > 5 * GeV ) ;

for ( Loop D0 = loop( "K- PI+" , "D0" ) ; D0 ; ++D0 )
{
  if( P( D0 ) > 10 * GeV ) { D0->save( "D0" ) ; }
}

for ( Loop Dstar = loop( "D0 PI+" , "D*+" ) ; Dstar ; ++Dstar )
{
  plot ( "Mass of D0 pi+" , M(Dstar) / GeV , 2.0 , 2.1 , 100 ) ;
}
```
LoKi: major design ideas

- Compact, easy to read and transparent code
- Hide all technicalities
- Implement all ‘everyday idioms’ as 1-line functions
- Locality:
  - Declare, create and use the objects only ‘locally’
  - 1 analysis = 1 short file
- High CPU performance
  - Reuse of the most modern C++ techniques
  - Paradigm of templated compile time metaprogramming
    - Lets compiler to “write” the code
- Implement everything as reusable components
  - LoKi functions are compatible with Loki, STL, boost, CLHEP
  - LoKi functions are used with cuts, other functions, histograms, tuples, MC truth, etc
- Weak coupling with concrete Event model, tools, etc
- Extendable
Compactness of the code

- Important for readability
- Important for debugging
- Number of simple errors (typos) is proportional to the overall number of lines
- Number of non-trivial errors also grows with the code length
- There many models, but clearly:
  - the first derivative is positive: $E' > 0$
  - the second derivative is also positive: $E'' > 0$
  - Some people claims also the $E^n \geq 0$
"Typical" physics analysis code does not follow the concept of locality

- Declaration & usage of variables often goes in the different places or event different compiling units
- True also for "good old FORTRAN"
- C++ allows to eliminate some part of non-locality
- And (with proper design) it allows to eliminate practically all non-locality:

```c
for( Loop B0 = loop ( "pi+ pi-" ), B0 , ++B ){...
```
Compactness & Locality

Selection criteria

K: IP to all PVs $\chi^2 > 4$

$\phi$: $|M(K^+K^-) - M(\phi)| < 12 \text{ MeV/c}^2; \Gamma = 4.5 \text{ MeV/c}^2, \sigma = 1.1 \text{ MeV/c}^2$

$p(\phi) > 12 \text{ GeV/c}$

$B_s$: $|M(\phi\phi) - M(B_s)| < 24 \text{ MeV/c}^2; \sigma = 12.1 \text{ MeV/c}^2$

Proper time $0.2 < \tau/\tau(B_s) < 4.$

$|tv01| < 0.75$, $tv01 = \cos \theta$

Direction angle $|\psi| < 10 \text{ mrad}$
## Compact code. Code metrics

**COCOMO model:** SLOCCount by David A. Wheeler, 2k+4

<table>
<thead>
<tr>
<th>Selection</th>
<th>SLOC</th>
<th>Person-month</th>
<th>Cost [k$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;...&gt;</td>
<td>2.6 k</td>
<td>6.5</td>
<td>73</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>362</td>
<td>0.8</td>
<td>9</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>1.1 k</td>
<td>2.3</td>
<td>30</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>1.5 k</td>
<td>3.6</td>
<td>40</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>1.4 k</td>
<td>3.4</td>
<td>38</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>3.2 k</td>
<td>8.0</td>
<td>91</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>530</td>
<td>1.2</td>
<td>14</td>
</tr>
<tr>
<td>&lt;...&gt;</td>
<td>1.0 k</td>
<td>2.3</td>
<td>30</td>
</tr>
<tr>
<td><strong>LoKi</strong></td>
<td>128</td>
<td><strong>0.3</strong></td>
<td><strong>2</strong></td>
</tr>
</tbody>
</table>
• select/filter the particles with certain properties
  • Functors: functions and predicates
• Loop over multy-particle combinations, e.g. all $\pi^+\pi^-$ pairs in the event
• Easy creation of virtual particles (on-demand)
  • Optionally apply various kinematical constraints
• Easy histograms & n-tuples (local!)
• “Save" interesting particles/combinations for the subsequent detailed analysis
• MC-truth match
Simple selection of particles (vertices, MC-particles, etc) according to kinematical and/or topological criteria

- Simple selection of particles (vertices, MC-particles, etc) according to kinematical and/or topological criteria

```cpp
select("Kaon", ID=="K-" || ID=="K+");
```

Positive pions with Confidence Level in excess of 1% and pT > 100 MeV/c^2

```cpp
select("Pi+", ID=="pi+" && CL>0.01 && PT>100*MeV);
```

Positive muons with \( \chi^2 \text{IP} \) with respect to the primary vertex in excess of 4

```cpp
const Vertex* pv = ... ;
select("MyMu", ID=="mu+" && IPCHI2(point(pv))>4);
```
LoKi: functions and cuts

Large set (>150) of predefined functions

- Simple properties of particles
  - \( P, PT, PX, M, CL, ID, Q, LV01, M12, DMASS, DMCHI2, \ldots \)

- Simple properties of Vertices
  - \( VCHI2, VTYPE, VX, VZ, VDOF, VPRONGS, VTRACKS, \ldots \)

- Topological properties of Particles and Vertices
  - \( IP, IPCHI2, VDCHI2, VDTIME, VDSIGN, DDANG, \ldots \)

- Operations with Functions - other Functions
  - \(+ - \ast / \sin \cos \tan \) \( \text{abs pow min max} \ldots \)

  \[
  \sin(PT) / \cos(\text{PY/PZ}) + \min(\text{abs(PX)}, \text{abs(PY)}) > 100
  \]

Cuts/predicates are formed from functions
LoKi: multiparticle loops

Loops over particle combinations, selects combinations according to kinematical and topological criteria

Loops over particle combinations, selects combinations according to kinematical and topological criteria

**Simple loop over all K- π+ π+ π- combinations**

```c
for( Loop D0 = loop( "K- pi+ pi+ pi-" , "D0") ; D0 ; ++D0 )
{
    if( PT( D0 ) > 1 * GeV && VCHI2( D0 ) < 49 )
    {
        plot("K- pi+ pi+ pi- mass", M(D0)/GeV , 1.5 , 2.0 , 200 );

        Cut dm = abs( DMASS("D0") ) < 30 * MeV ;
        if( dm(D0) ) { D0->save("D0"); }
    }
}
```

**Require pT of combination in excess of 1 GeV/c and \( \chi^2_{VX} < 49 \)**

**Book and fill (1 action!) the histogram**

**Save the combinations with |ΔM|<30 MeV/c^2**
LoKi: Histograms

- Histograms are local & booked on-demand
- No need for pre-booking!
- Include variants for effective implicit loops

```c
for( Loop D0 = loop( "K- pi+" , "D0" ) ; D0 ; ++D0 )
{
    plot( "K- pi+ mass", M(D0)/GeV , 1.7 , 2.0 , 150 );
}
plot( loop( "K- pi+", "D0" ) , "(2)K-pi+ mass" , M12 / GeV , 1.7 , 2.0 , 150 );
plot( select("Kaons", ID == "K-" ) , "PT of kaons ", PT /GeV , 0 , 5 , 100 );
```
LoKi: N-tuples

- N-Tuples are local & booked on-demand: No need for pre-booking of N-Tuple and its items
- Include variants for effective implicit loops
- Both ROOT and HBOOK are supported as persistency: the C++ code is neutral with respect to the N-Tuple actual persistency

```cpp
Tuple tuple = ntuple("My N-Tuple for K- pi+ combinations");
for( Loop D0 = loop("K- pi+", "D0") ; D0 ; ++D0 )
{
    tuple -> column( "M" , M(D0)/GeV);
    tuple -> column( "PT" , PT(D0)/GeV);
    tuple -> fill("PX,PY,PZ", PX(D0)/GeV, PY(D0)/GeV, PZ(D0)/GeV);
    tuple->write() ;
}
```

Book N-tuple
Fill columns one-by-one
Fill few columns at once
Commit N-Tuple row
What else is needed?

- Easy “formal” match to MC-truth.
- The simple and naïve questions
  - What MC particle matches to this RC particle?
  - What RC particle matches to this MC particle?
    - Unfortunately they are not formal enough and therefore not so well defined for all cases, require a lot of “if”s and “artificial” cutoffs.
- Reformulate: Does this MC-particle directly or indirectly (through daughters) makes the contribution to the given RC-particle?
  - Formal, well-defined, recursive & applicable for all case
  - One matches a bit more, but one looses nothing
    - Some final filtering is required
- Easy to code & very efficient:
  - just around O(10+10) (recursive) lines
LoKi: MC matching

```
MCRange mcD0s = finder->findDecays("D0 -> K- pi+");

Cut mccut = MCTRUTH( mcTruth() , mcD0s );

for( Loop D0 = loop( "K- pi+" , "D0" ) ; D0 ; ++D0 )
{
    if( mccut( D0 ) )
    {
        plot("mass of true D0->K- pi+", M(D0)/GeV,1.7,2.0,150);
    }
}
```
What else is missing?

• What could be added?
• What can be improved?
  • Locality: one still needs at least one more “unit”: come job-configuration file (e.g. the list of input data files, name of the output file with histograms & n-tuples, output DST, etc...)
  • Compilation time...
• What is missing?
  • Interactivity!

Solution? Go to Python!
Why Python?

- Python is a language with the special emphasize for fast prototyping and development
- Scripting and interactivity combined in a natural way
- Easy integration with the third party software
- Availability of external packages
  - visualization, statistical analysis
    - ROOT, HippoDraw, Panoramix, PyX, GNUplot
  - Event display
    - Panoramix
  - Bookkeeping data base
  - Interface to GRID
    - GAN
  - and many others
Python in LHCb

- **DaVinci**
  - C++ Analysis Application

- **Gaudi**
  - Services, Algorithms & Application Control

- **LoKi**
  - Physics Analysis ToolKit

- **Event Model**

- **Bender**
  - Python Analysis Application

- **GaudiPython**

- **Dictionaries**

- **Externals**: Visualization, Event Display, etc.
The generic package for Gaudi & Python bindings

- Access to major Gaudi Components
  - Services, algorithms and tools
- Application Configuration
  - Algorithm schedule
  - Configuration of all components
  - “Dynamic” reconfiguration is possible

The technique

- LCG dictionaries for C++/Python binding
Bender = LoKi+Python+

- LCG dictionaries for C++/Python binding
  - A bit of raw Boost also
- >95% of LoKi’s C++ functionality is available in Python
  - Non-trivial due to heavy templated nature of LoKi
  - Situation improves with PyLCG evolution
- The mixture of C++ and Python is possible
  - C++ algorithm with Python cuts (“LoKiHybrid”)
- The bulk of actual computations in C++
  - Minimal Python-related penalty
- The conversion between existing Python Bender’s and C++ LoKi’s algorithms is simple in both directions:
  - Semantics is very similar
Physics analysis

- **Functions and cuts**
  
  \[
  \text{cut} = (\text{ID} == \text{"D+"}) \& (P > 5\text{ GeV}) \& (\text{PT} > 2\text{ GeV})
  \]

- **Selection of particles**
  
  \[
  k = \text{self.select( tag = \text{"K+"}, } \text{cuts = ( \text{"K+" == ID } \& (\text{PT} > 0.5 \text{ GeV}) )}
  \]

- **Looping over combinations**
  
  \[
  \text{for phi in self.loop(formula=\text{"K+ K-"}, pid=\text{\text{\text{"phi(1020)\}}}):}
  \]
  
  \[
  m = M(\text{phi}) / \text{GeV}
  \]
  
  \[
  p = P(\text{phi}) / \text{GeV}
  \]

- Saving/retrieve of interesting combinations,
- Vertex/Mass-Vertex/Direction/Lifetime fits
Histos & N-Tuples

**Histograms**

```python
h1 = self.plot ( "phi mass" ,
    M(phi) ,
    1000, 1050 )
```

**N-Tuples**

```python
self = self.nTuple( "Phi NTuple" )
tup.column ( "ID", ID(phi) )
tup.column ( "p" , P (phi) )
tup.column ( "pt" , PT(phi) )
tup.write()
```
Histo visualization

- The histogram visualization can be done through
  - **ROOT**
    - native ROOT through Python prompt
    - `rootPlotter` from PI through AIDA pointer
  - **Panoramix/LaJoconde**
    - Directly through AIDA pointer
  - **HippoDraw**
    - `hippoPlotter` from PI through AIDA pointer
- Few lines “common interface” for trivial plotting exist
- The interactive analysis of Gaudi N-Tuples is possible in Bender with ROOT persistency and ROOT module directly
Everything can be combined

HippoDraw

Panoramix

PI/ROOT

ROOT

Bender/Python prompt
The integrated analysis and visualization of statistical and event data is possible.
from Bender.Main import *

class Dstar(Algo):
    def analyse(self):
        self.select('K-', ('K-' == ID) & (PT>1*GeV))
        self.select('pi+', ('pi+' == ID) & (P >3*GeV))
        dmass = ABSDM("D0") < 30 * MeV

        for D0 in self.loop('K- pi+', 'D0'):
            If ( VCHI2(D0) < 4 ) & dmass(D0) : D0.save('D0')

        tup = self.nTuple("D*+ N-Tuple")

        for Dst in self.loop('D0 pi+', 'D*(2010)+'):
            dm = M(Dst)-M1(Dst)
            h1 = self.plot("Delta mass for D*+", dm, 130 * MeV, 170 * MeV)
            tup.column('M', M(Dst) / GeV)
            tup.column('DM', dm / GeV)
            tup.column('p', P(Dst) / GeV)
            tup.column('pt', PT(Dst) / GeV)
            tup.write()

        return SUCCESS
The life is not perfect 😞

- Bender has a lot of nice features. But it also has some clear disadvantage:
  - Some CPU penalty is practically unavoidable
    - but could be minimized with careful design.
  - The weak points are well identified and could be avoided
  - Some external optimizers, like Psyco could be helpful
  - But a bid dangerous to use e.g. in Online/Trigger applications
Solution? “Hybrid”

• When the problem is identified, one has 95% of the solution in the hand. Let’s use “Hybrid”!

• “Hybrid” solution for selection criteria for Online/Trigger application:
  • describe the selection criteria in a friendly way using *Python* strings
  • reuse of LoKi&Bender concepts
  • convert *Python* into *C++* at initialization phase
  • use the constructed *C++* objects in *C++* algorithms/tools
  • No penalty due to *Python*
HltExclusive.Members += { "CombineParticles/HltSelB2DstarMu" };
HltSelB2DstarMu.PhysDesktop.InputLocations = {"Phys/HltSharedDstarWithD02KPi"
HltSelB2DstarMu.DecayDescriptor = "[B0 -> D*(2010)- mu+]cc" ;
HltSelB2DstarMu.DaughtersCuts = { "mu-" : "(PT>400)",
    "D*(2010)+'" : "(MAXTREE('D0'==ABSID,BPVVDCHI2) >6.25 )" } ;
HltSelB2DstarMu.CombinationCut = "(ADAMASS('B0')<2.5*GeV)" ;
HltSelB2DstarMu.MotherCut = "(VFASPF(VCHI2/VDOF)<6.67)" ;
The “hybrid” approach allows the implementation of very efficient and powerful “hybrid” factories:

- Expressions, parameters, units, functions, ....

The approach combines the great CPU performance of C++ with the great flexibility of Python.

The hybrid approach allows to define all cuts in “easy-and-friendly” way.

The preliminary testing with interactivity (GaudiPython or Bender) is possible and recommended.

The same uniform semantics as for LoKi/C++ and Bender/Python.
Summary

• The powerful and simultaneously User friendly physics analysis is reality (in LHCb)
  • LoKi: The powerful C++ Tool kit
  • Bender: The interactive python-based environment for the physics analysis
    • Many physicists in LHCb use them
      • Including the senior physicists
      • including the spokesman
• Also the useful hybrid product as a result: it is not very user friendly, but rather simple, formal, save and efficient.
  • The base for in High Level Trigger
More information

• Savannah portals for LoKi and Bender:
  • http://savannah.cern.ch/Loki
  • http://savannah.cern.ch/Bender
Loki is a god of wit and mischief in Norse mythology.

- Loops & Kinematics

Ostap Suleiman Berta Maria Bender-bei

- The title: “The great schemer”
- Attractive & brilliant cheater

Essential for successful and good physics analysis