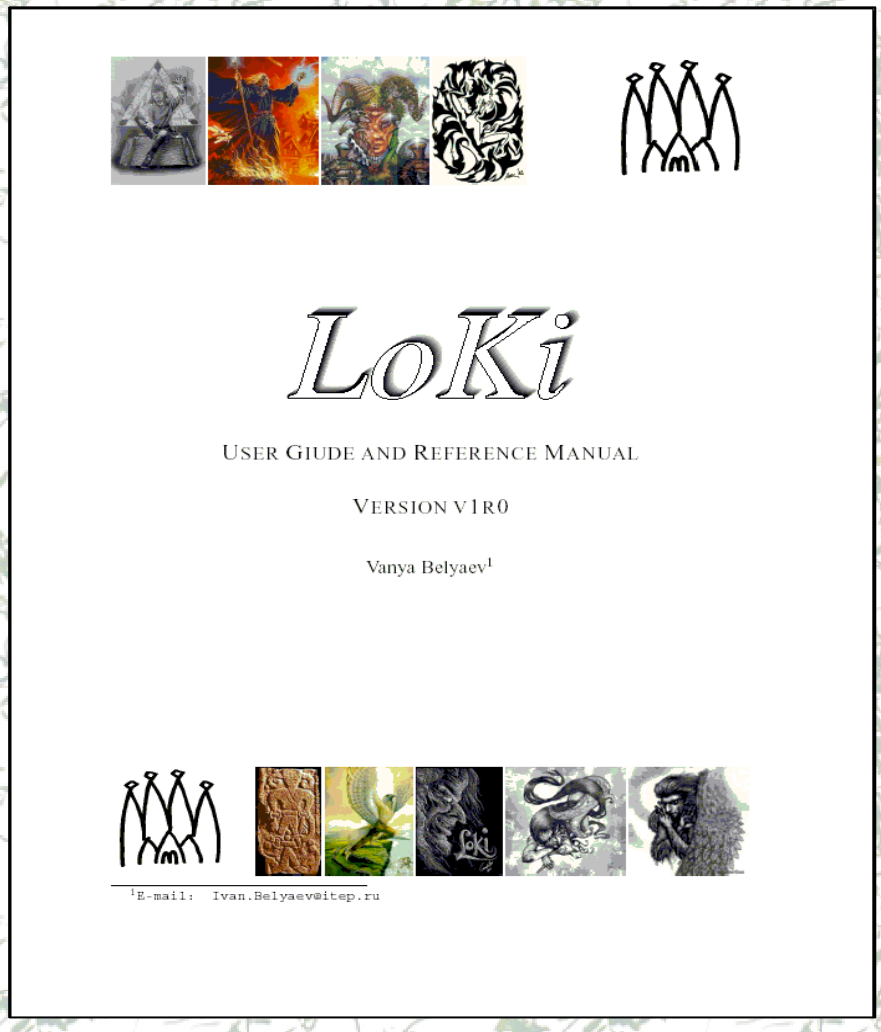
LoKi's Cook-book: Writing analysis algorithms in C++ Tutorial v4r2

Vanya Belyaev (Syracuse)

Outline

- **LoKi**
  - **V4r2 released**
  - **DaVinci v17r5**



LoKi

USER GIUDE AND REFERENCE MANUAL

VERSION V1R0

Vanya Belyaev[1]

[1]E-mail: Ivan.Belyaev@itep.ru

LoKi

"Tool for senior physicists" ?

C++ *Toolkit for user friendly Physics Analysis*

- Available for users from begin of 2003
  - The first analysis has been reported March 2003
    - *Benoit Viaud*: $B^0 \to \phi K_S$
- Used for few **TDR** studies in 2003
- In use for some **DC04** selections/stripping (~¼?)
- In use for private studies,
    - failure to count all users.. ☹
- Mailing list: ***lhcb-loki@cern.ch***
- LoKi pages by Elena Mayatskaya

LoKi

The major design criteria

- Locality
  - Introduce and use objects in *local* scope
  - One file
  - One method
  - One screen
- Compact code
- Safety
  - No need in `new, delete`
- "Standard"
  - Use `STL` idioms & semantics

"META-LANGUAGE"

- The details can be found in "**LoKi** *User Guide & Reference Manual*"
  - ***LHCb-2004-023***
- **DoxyGen** documentation:
  - *Now* available (partly) throught **Phys Doxygen**

LoKi

- To be discusses today:
  - LoKi & DaVinci
  - LoKi basic
  - MC matching
  - Loops & Charge-blind loops
  - Recipies on every day
  - Customization of LoKi

LoKi & DaVinci

- **LoKi** is a toolkit for **DaVinci**
  - Code : **LoKi**
  - Job Configuration & steering: **DaVinci**
- All user code is placed in the body of algorithm, which inherits from **LoKi::Algo** (or **LoKi::AlgoMC**) , which inherits from **DVAlgortithm/GaudiTupleAlg/GaudiHistoAlg/GaudiAlgorithm** chain
  - The actual chain is much more complicated
- Only one mandatory method **analyse()** needs to be redefined
  - majority of mandatory and tedious stuff is hidden by preprocessor **MACROs**

Layers of LoKi

- Multilayered structure
- Low level generic utilities
  - `Range_` , `Selected_` , `Combiner_` , …
  - `STL`-like algorithmic + functional layer
  - Templated, very generic, very efficient
    - (I am *very* proud of them!)
  - Applicable to different and unrelated problems
  - Almost invisible for end-users
- Few hierarchical levels of "specific" utilities
  - Usually only the last layer is visible for end-users
  - `Relations→MCMatchObj→MCMatch→MCTRUTH`
  - `Combiner_→LoopObj→Loop`
  - `(XXX→INTuple→NTuple::Tuple)→Tuples::TupleObj→Tuples::Tuple`

Few differences DC04 ↔DC06

- All event classes in namesapce `LHCb::`

  `Particle → LHCb::Particle`
- All units in namespace Gaudi::Units

  `GeV → Gaudi::Units::GeV`
- Useful (self-explanatory) *typedefs*:

  `LHCb::<Object>::Vector`

  `LHCb::<Object>::ConstVector`

  `LHCb::<Object>::Container`
- `LoKi::Algo` inherits from `DVAlgorithm`
  - *ALL* `DaVinci` lines could be just moved into `LoKi` scope
  - Many `LoKi` lines works just in `DaVinci`

“Hello,World”

```
#include “LoKi/LoKi.h”

LOKI_ALGORIHTM( MyAlg )
{

 info() << “Hello, World” << endreq ;

 return StatusCode::SUCCESS ;
};
```

- Algorithm body,
- implementation of constructor & destructor,
- factories
- `MyAlg::analyse()`

6 lines,

1 functional line

Excersize 0

- Compile & run HelloWorld example

Hints:

- Template is …/templates/TEMPLATE.cpp
- Emacs will not help you ☹
- It is DaVinci *algorithm*: *.opts file is required
- It is Gaudi *component*: *_load.cpp, *_dll.cpp

Solution

../solutions/HelloWorld

From (to?) base classes:

- Generic access to data, tools and services

```
get<TYPE>   (…)
tools<TYPE> (…)
svc<TYPE>   (…)
```

- Printout & error counts:

```
info(), debug() , error() , fatal(), …
Error(…), Warning(…)
```

- Histograms, NTuples and Event Collections

```
plot(…)
nTuple()
evtCol()
```

DaVinci tools

- All DaVinci tools are available through DVAlgorithm base class:

```
IVertexFitter*         vertexFitter          ( const
   std::string& name = "" ) const;
IGeomDispCalculator* geomDispCalculator    ( … ) const ;
IParticleFilter*       particleFilter        ( … ) const ;
IFilterCriterion*      filterCriterion       ( … ) const ;
IParticleCombiner*     particleCombiner      ( … ) const ;
IParticleReFitter*     particleReFitter      ( … ) const ;
```

Basic types

- 4 types of basic "**objects**":

  `Particle, Vertex, MCParticle, MCVertex`

  +HepMC::GenParticle, tags, …

- "**Function**" : functor which gets as argument the pointer to the "**object**" and returns `double`

  ```
  Func, VFunc, MCFunc, MCVFunc      (interface)
  Fun , VFun , MCFun , MCVFun       (assignable)
  ```

- "**Cut/Predicate**": functor, which gets as an argument the pointer to the "**objects**" and returns `bool`

  ```
  Cuts, VCuts, MCCuts, MCVCuts      (interface)
  Cut , VCut , MCCut , MCVCut       (assignable)
  ```

- "**Range**": a lightweight representation (`STL` compliant) of container/sequence of "**objects**"

  `Range, VRange, MCRange, MCVRange`

"Functions"

- LoKi offers about >100 "Functions":
- "Particle Functions", e.g.

C++ type

alias

```
LoKi::Particles::Momentum          P
LoKi::Particles::Identifier        ID
LoKi::Vertices::ImpactParameter    IP
```

- "Vertex Functions"

```
LoKi::Vertices::VertexChi2         VCHI2
```

- "MCParticle Functions"

```
LoKi::MCParticles::ProperLifeTime  MCTIME
```

- "MCVertex Functions"

```
LoKi::McVertices::MCVertexDistance MCVDIST
```

"Metafunctions" (~20)

- Transverse momentum of the first daughter

```
CHILD( PT , 1 )
```

- Δ_{LL}(K-π) for the first daughter of the first doughter

```
CHILD( CHILD( PIDK , 1 )  , 1 )
```

- Minimal Δ_{LL}(K-π) for all daughter kaons in the decay tree:

```
MINTREE( PIDK , "K-" == ABSID )
```

- And a lot of "adapters":

```
VXFUN, MCMOTH, FILTER, …
```

Functions & Cuts

- Operations with functions:

```
Fun fun = P + PT / GeV * sin( 1/ M ) ;
Fun fun = pow(P,Q) + atan2(PX,PY);
```

- Comparisons:

```
Cut cut =  PT > 1.5 * Gaudi::Units::GeV ;
```

- Boolean operations

```
Cut cut = ( PT > 1.5 * Gaudi::Units::GeV )&&( Q < 0 ) ;
```

- Special cases
  - ID,ABSID,MCID,MCABSID,GID,GABSID :

```
Cut cut = "pi+" ==    ID ;
Cut cut = "mu-" == ABSID ;
```

“Own” functions/cuts I

- Class which implements `LoKi::Function<TYPE>` or `LoKi::Predicate<TYPE>` interface :
- `TYPE`

```
const (LHCb::,LHCb::MC,HepMC::Gen)Particle*
const (LHCb::,LHCb::MC,HepMC::Gen)Vertex*
```

- 2 mandatory methods

```
MyType* clone() const ;
result_type operator() ( argument a ) const ;
```

- Optional:

```
std::ostream& fillStream( std::ostream& s ) const {
 return s << “XXX” ; }
```

Every day idioms: simple selections

```
#include "LoKi/LoKi.h"
LOKI_ALGORITHM( MyAlg)
{
  using namespace LoKi          ;
  using namespace LoKi::Cuts
  using namespace LoKi::Types
  Range pions = select( "pi" ,
       "pi+" == ABSID && PT > 0.5 * GeV ) ;
  info() << " found pions:" << pions.size()
         << endreq ;
  return StatusCode::SUCCESS ;
};
```

Select from all loaded/created particles

TAG

Cuts: π^+ and π^- with p_T>500 MeV/c

Simple selections (II)

- Select from other selected range :

```
Range pions = select( "pi" , "pi-" == ABSID ) ;
Range pos   = select( "pi+" , pions , Q > 0 ) ;
```

- Select from **KeyedContainer**:

```
const LHCb::Particle::Container* p =
     get<LHCb::Particles>("Phys/MyChannel/Particles");
Range bs = select( "myBs0" , p ,
                   "B_s0" == ID );
```

- Select from arbitrary *sequence* **seq** :

```
Range k0s = select( "myK0S" ,
     seq.begin() , seq.end() , "KS0" == ID );
```

Easy way to get cuts from *.opts

*.opts :

```
MyLoKiAlg.Cuts = { "ptMin" : 1 * GeV ,
                   "alpha" : 0.99999 } ;
```

*.cpp:

```
Cut ptCutMin = PT > cutValue("ptMin") ;
Cut ptCutMax = PT < cutValue("ptMax" , 5 * GeV ) ;
```

Select tracks with $\min(\chi^2)_{IP}>25$

- Very efficient operation if done **BEFORE** looping, the combinatorics is reduced significantly (and **huge** gain in CPU!)

Vertices are selected in a similar way

```
const LHCb::RecVertex::ConstVector& pvs
  = desktop()->primaryVertices() ;
```

The function objects itself

```
Fun mipc2 = MIPCHI2( geo() , pvs ) ;
Range pions = select( "pi" ,
          "pi+" = ABSID && mips > 25) ;
```

Select pions not from primary verstices

Trivial 1-particle loops

- Nothing special: **Range** behaves like **STL**-container

```
Range pions = select( … )   ;
for( Range::iterator ipi = pions.begin() ;
     pions.end() != ipi ; ++ipi )
{
 const LHCb::Particle* p = *ipi ;
   info() << " pion momentum:"
           << P( p ) / Gaudi::Units::GeV << endreq
};
```

Exercise 1

- Select stable particles according to simple criteria

  ABSID, Q, PIDK, PIDmu, P, PT,…
- Sub-select from selected containers using refined criteria from *.opts file
- Count them

Hints:

```
select( … ) , cutValue( … )
```

(Almost) solution:

```
…/solutions/GetData
```

Exercise 2

- Select stable particles according to some simple criteria
- Make simple loop over Range of particles, fill n-tuple using simple functions

  `ABSID, Q, PIDK, PIDmu, P, PT,…`

Hints:

- More configurations for N-tuples is required in `*.opts`
- `nTuple( … ) , column( name , value )`

Solutions:

```
../solutions/SimpleLoop
../solutions/SimpleLoop2
```

Multiparticle loops:

- Loop over selected particle tags:

```
Range mypi = select( "myPi+" , … );
Range myK  = select( "myK-" , … );
for ( Loop D0 = loop( "myK- myPi+" , "D0" ) ;
                                D0 ; ++D0 )
{
  plot( M( D0 )/GeV,"K pi m",1.5,2.0 );
  if  ( VCHI2( D0 ) > 100 ) { continue ; }
  plot( M( D0 )/GeV,"K pi m,chi2",1.5,2.0);
}
```

Loop objects behaves as **Particle**

Loop objects behaves as **Vertex**

Access to daughters:

```
using namespace LoKi::Child
for ( Loop D0 = loop ( "K- pi+" , "D0" ) ; D0 ; ++D0 )
{
   const LHCb::Particle* kaon = D0(1) ;
   const LHCb::Particle* pion = D0(2) ;
   const LHCb::Particle* k1   = child ( D0 , 1 );
   const LHCb::Particle* p1   = child ( D0 , 2 ) ;
}
```

0 is "self"

```
const LHCb::Particle* B   = … ;
const LHcb::Particle* psi = child ( B   , 1 ) ;
const LHCb::Particle* mu  = child ( psi , 1 ) ;
const LHCb::Particle* mu1 = child ( B   , 1 , 1 ) ;
const LHCb::Particle* mu2 = child ( B   , 1 , 2 ) ;
```

Creation

- Different creation strategies: [optional]
- In the loop declaration:

```
Loop D0 = loop( "myK- myPi+" , "D0" , CREATOR )
```

  - nothing – default creation
  - pointer to IParticleCombiner tool
  - nickname or typename of IParticleCombiner tool
    - "", "OffLine" : OfflineVertexFitter
    - "Trigger" : TrgVertexFitter
    - "Kalman","Blind" : BlindVertexFitter
    - ? : MomentumCombiner
- In the loop body:

```
for ( Loop D0 = … ; D0 ; ++D0 )
{
 // optional: Re-create:
 StatusCode sc1 = D0->make( CREATOR )
}
```

(Re)Fits

- Related to the creation strategies: [optional]
- In the loop body:

```
for ( Loop D0 = … ; D0 ; ++D0 )
{
 // optional: Re-Fit
 StatusCode sc2 = D0->reFit( REFIT ) ;
}
```

  - nothing – default refitter
  - pointer to IParticleReFitter tool
  - nickname or typename of IParticleReFitter tool
    - "", "OffLine"   : OfflineVertexFitter
    - "Kalman","Blind" : BlindVertexFitte

Save something interesting

```
Cut cut = … ;
for ( Loop D0 = … ; D0 ; ++D0 )
{
 if ( !cut( D0 ) ) { continue ;}
 D0->save( "myD0" ) ;
}
```

TAG

- Extract saved particles:

```
Range d0 = selected( "myD0" )
info() << " D0 saved: "
       <<   d0.size() << endreq;
```

Exercise 3

- Select charged kaons
- Sub-select positive and negatoive
- Make loop over all K^+K^- combination, plot invariant mass under some simple criteria, fill simple N-Tuple
- Save "good" ϕ-candidates
- Count them

Hints:

- Default configurations of creators and refitters are OK
- ϕ name is `phi(1020)`

Solutions:

`../solutions/LoKiLoop`

Using Patterns

- Shortcut for *"loop + cuts + save + selected"*

```
Range phi =
  pattern( "phi" , "K+ K-" ,"phi(1020)",
                ADMASS("phi(1020)") < 10 * MeV ,
                VCHI2 < 25 ) ;
```

- Compact
- Efficient

Get something "working" (1)

```
//Select muons (μ+ and μ-) according to Bs→J/ψφ selection cuts
Range mu = select( "mu" ,                          /* unique tag    */
                   "mu+"  == ABSID &&              /* μ+ and μ-     */
                    PIDmu >  -8    &&              /* ΔLL(μ-π)>-8   */
                    mipc2 > 25     &&              /* χ2IP > 25     */
                    PT    >  300 * MeV ) ;         /* pT>300 MeV/c  */
//Select J/ψ→ μ+μ-
Cut dm = ADMASS("J/psi(1S)") < 50 * MeV ;//ΔM<50MeV/c2
for( Loop Jpsi = loop( "mu mu","J/psi(1S)" );
                        Jpsi ; ++Jpsi )
{
  if ( 0 != SUMQ(Jpsi)    ) { continue ;}/* μ+ & μ-  */
  if ( VCHI2(Jpsi) > 100 ) { continue ;}/* χ2VX<100 */
  if ( dm( Jpsi) ) { Jpsi->save("psi");}/*ΔM<50MeV/c2*/
};
```

$\Sigma q = 0$ and $\chi^2 < 100$

Get something "working" (II)

```
//Select kaons (K+ and K-) according to Bs→J/ψφ selection cuts
Range k = select( "K" ,                        /* unique tag   */
                  "K+"  == ABSID &&            /* K+ and K-    */
                  PIDK  > -2     &&            /* ΔLL(K-π)>-2  */
                  mipc2 >  4     &&            /* χ2IP > 4     */
                  PT    > 500 * MeV ) ;        /* pT>500 MeV/c */
//Select φ→ K+K-
Cut dm = ADMASS("phi(1020)") < 20 * MeV ;//ΔM<20MeV/c2
for( Loop phi = loop( "K K","phi(1020)" ); phi ; ++phi )
{
  if ( 0 != SUMQ(phi)    ) { continue ;}/* K+ & K-  */
  if ( VCHI2(phi) > 100 ) { continue ;}/* χ2VX<100 */
  if ( dm( phi ) ) { phi->save("phi");}/*ΔM<20MeV/c2*/
};
```

Get something "working" (III)

Select Bs according to $B_s \to J/\psi\phi$ selection cuts

```
Cut dm = ADMASS("B_s0") < 50 * MeV;/* ΔM<50MeV/c² */

  // Loop over selected J/ψ and φ
for( Loop Bs = loop( "psi phi","B_s0" ); Bs; ++Bs )
{
  if ( !dm( Bs ) ) { continue ; } ;/* ΔM<50MeV/c² */
  if ( VCHI2(Bs) > 100 ) { continue ;} /* χ²VX<100 */
  if ( mips( Bs ) > 25 ) { continue ;} /* χ²IP<25  */
  Bs->save("Bs");
};
   // Retrieve all saved "Bs"
Range Bs = selected("Bs") ;
if( !Bs.empty() ) { setFilterPassed(true);}
```

Or everything together: 1st page

```
VRange primaries = vselect( "PVs" ,
  Vertex::Primary == VTYPE ) ;    /* all primary vertices */
Fun mipc2 = MIPCHI2( geo() , primaries ); /* min(χ²IP) */
// muons:
Range mu = select( "mu" ,                /* unique tag   */
                   "mu+"  == ABSID &&    /* μ+ and μ-    */
                    PIDmu >  -8    &&    /* ΔLL(μ-π)>-8  */
                    mipc2 > 25     &&    /* χ²IP > 25    */
                    PT    >  300 * MeV ) ;/* pT>300 MeV/c */
// kaons:
Range k  = select( "K" ,                 /* unique tag   */
                   "K+"  == ABSID &&     /* K+ and K-    */
                   PIDK  >  -2    &&     /* ΔLL(K-π)>-8  */
                   mipc2 >  4     &&     /* χ²IP > 4     */
                   PT    > 500 * MeV ) ; /* pT>500 MeV/c */
```

Or everything together: 2nd page:

```
// Cuts:
Cut dmPsi = ADMASS("J/psi(1S)") < 50*MeV;/* ΔM<50 MeV/c² */
Cut dmPhi = ADMASS("phi(1020)") < 20*MeV;/* ΔM<20 MeV/c² */
Cut dmBs  = ADMASS("B_s0")      < 50*MeV;/* ΔM<50 MeV/c² */
Cut q     = 0 == SUMQ     ;              /* Σq = 0      */
VCut chi2  = VCHI2 < 100 ;               /* χ²vx<50 MeV/c² */
// Loops:
pattern("psi", "mu mu","J/psi(1S)", dmPsi && q , chi2 );
pattern("phi", "K K"  ,"phi(1020" , dmPhi && q , chi2 );
Range Bs =
 pattern("Bs" , "psi phi" , "B_s0" ,
        dmBs && mipc2 < 5  , chi2 );
if( !Bs.empty() ) { setFilterPassed(true);}
```

1+1 page !!!

Exercise 4

- "Reconstruct" J/ψ candidates
- "Reconstruct" ϕ-candidates
- Fill simple N-Tuple(s)
- Save "good" B_s-candidates
- Count them

Hints:

- Default configurations of creators and refitters are OK
- ψ name is `J/psi(1S)`

Solutions:

```
../solutions/PsiPhi
```

MC match

- **LoKi** uses own concept of MC-truth matching, described in details in **LUG**
  - "*Loose*" matching: *none* relations can be lost ☺
  - Some "*extra*" relations could be a bit confusing ☹
  - Technically based on **Relation Tables** from **Kernel/Relations** package
    - Requires:

```
IRelation<LHCb::ProtoParticle,LHCb::MCParticle,double>
IRelation<LHCb::Particle,LHcb::MCParticle>
IRelation<LHCb::Particle,LHCb::MCParticle,double>
IRelation<LHCb::Track,LHCb::MCParticle,double>
```

No way for transitions to `Linkers`

- Natural coupling with **MCDecayFinder** tool and **MCParticle** selections
- Few helper adapter functions

MCMatch

```
MCFinder mc = mcFinder("some name") ;
MCRange mcPsi = mc-> findDecay(
   "B_s0 -> ^J/psi(1S) phi(1020) ");
MCMatch match = mcTruth("some name") ;
Cut truePsi = MCTRUTH( match , mcPsi ) ;
For ( Loop Jpsi = loop("mu mu", … ) ;
  Jpsi ; ++Jpsi)
{
   if( !truePsi( Jpsi) ) { continue ; }
}
```

Evaluates to true, if both muons come from true MC J/psi from this decay chain

MC truth Match

```
Cut truePsi = MCTRUTH( match , mcPsi ) ;
Cut truePhi = MCTRUTH( match , mcPhi ) ;
Cut trueBs  = MCTRUTH( match , mcBs  ) ;
Cut trueMu  = MCTRUTH( match , mcMu  ) ;
Cut trueK   = MCTRUTH( match ,  mcK  ) ;
for( Loop Bs = loop("psi phi", … ); Bs ; ++Bs)
{
tuple -> column("mcbs" ,trueBs  (Bs    ) );
tuple -> column("mcpsi",truePsi (Bs(1)) );
tuple -> column("mcphi",truePhi (Bs(2)) );
tuple -> …
}
```

Useful utility `DecayChain`

Prints (MC) decay chains in different formats

- Templated
  - applicable to **Particles**, **MCParticles**, lists, trees, …
  - **std::ostream , MsgStream , '\n' , endreq , …**
  - **(MC)Cut, …**
- Different "formats" are supported
  - Default setting is "reasonable"
- "Intuitive" and recursive

```
DecayChain print ;
dc.print ( WHAT    , /* what to print */
     STREAM , "\n" , /* stream and terminator  */
     ACCEPT ,    /* predicate "to be print"      */
     MARK   ) ; /* predicate "to be colorized" */
// dc.print( Bs , info() , endreq , ALL , MCTRUTH( mc , mcBs )  ) ;
```

Exercise 5

- "Reconstruct" J/ψ candidates
- Fill simple N-Tuple(s) with MC-flags for muons and for J/ψ candidates

Hints:

- The actual base is **`LoKi::AlgoMC`**

```
LOKI_MCALGORITHM( … ) ;
```

- Default configurations of creators and refitters are OK
- ψ name is **`J/psi(1S)`**
- To be efficient:

```
MyAlg.PP2MCs = {"Relations/Rec/ProtoP/charged"} ;
```

Solutions:

```
../solutions/PsiMC
```

Exercise 6 (Homework)

- "almost realistic analysis algorithms"
- "Reconstruct" full $B_s \rightarrow J/\psi\phi$ chain
- Fill simple N-Tuple(s) with all MC-flags

Hints:

- Default configurations of creators and refitters are OK
- ψ name is `J/psi(1S)`
- B_s name is `B_s0`
- To be efficient:

```
MyAlg.PP2MCs = {"Relations/Rec/ProtoP/Charged"} ;
```

Solutions:

```
../solutions/Bs2PsiPhi
```

Exercise 7 (Advanced Homework)

- "almost realistic analysis algorithms"
- "Reconstruct" full $B_s \to D_s K$ chain
- Fill simple N-Tuple(s) with all MC-flags

Hints:

- Both B_s and anti-B_s could decays into this state
  - decay descriptor
- D_s^+ name is `Ds+`
- B_s name is `B_s0`
- To be efficient:

```
MyAlg.PP2MCs = {"Relations/Rec/ProtoP/Charged"} ;
```

Exercise 8 (Homework)

- Run algorithm from Exersises 6&7 on bb-inclusive data,
- Compare the basic distributions for signal and "background"
- P,PT, MIPCHI2 for final-state particles, intermediate particles and selected B-candidates

Hints:

- The nessessary data is available `/software/lhcb/BenderData directory`
- The proper `*.opts` files need to be prepared

Select Primary vertex

MY FAULT WITH v4r2

- Select primary vertex according to some criteria, e.g. the vertex with minimal χ^2_{IP}:

```
VRange pvs = vselected("PVs" ,
                Vertex::Primary == VTYPE ) ;
for ( Loop Bs = loop("psi phi",…); Bs ; ++Bs )
{
  const Vertex* pv =
    SelectPrimaryVertexMin(
    pvs.begin() , pvs.end() ,
    VIPCHI2( Bs , geo() ) ,
    VPSD( Bs, geo() ) > -0.2 * mm) ;
}
```

Sequence of vertices

Selection criterion

Cut: B_s should not be "too" upstream with respect to selected primary vertex

Select Primary vertex:

- Select primary vertex according to some criteria, e.g. the vertex with minimal χ^2_{IP}:

```
VRange pvs = vselected("PVs" ,
              Vertex::Primary == VTYPE ) ;
for ( Loop Bs = loop("psi phi",…) ; Bs ; ++Bs )
{
  VRange::iterator ipv =
    select_min ( pvs.begin() ,
                 pvs.end()   ,
                 VIPCHI2( Bs , geo() ) ,
                 VPSD ( Bs, geo() ) > -0.2 * mm ) ;
   if ( pvs.end() == ipv ) { continue ; }
   const Vertex* pv = *ipv ;
};
```

More generic &

More efficient

Other features:

- **LoKi** is able to build *jets* (using popular **KtJet** *algorithm*)
- LoKi is able to create Particles from generator infomration: usful to check different decay models with the same code as analysis
- LoKi supports many links inbetween
- RC↔ MC↔ HepMC
- LoKi suppors MC-truth access for reconstructed primary vertices
- And many-many more

Important missing component:

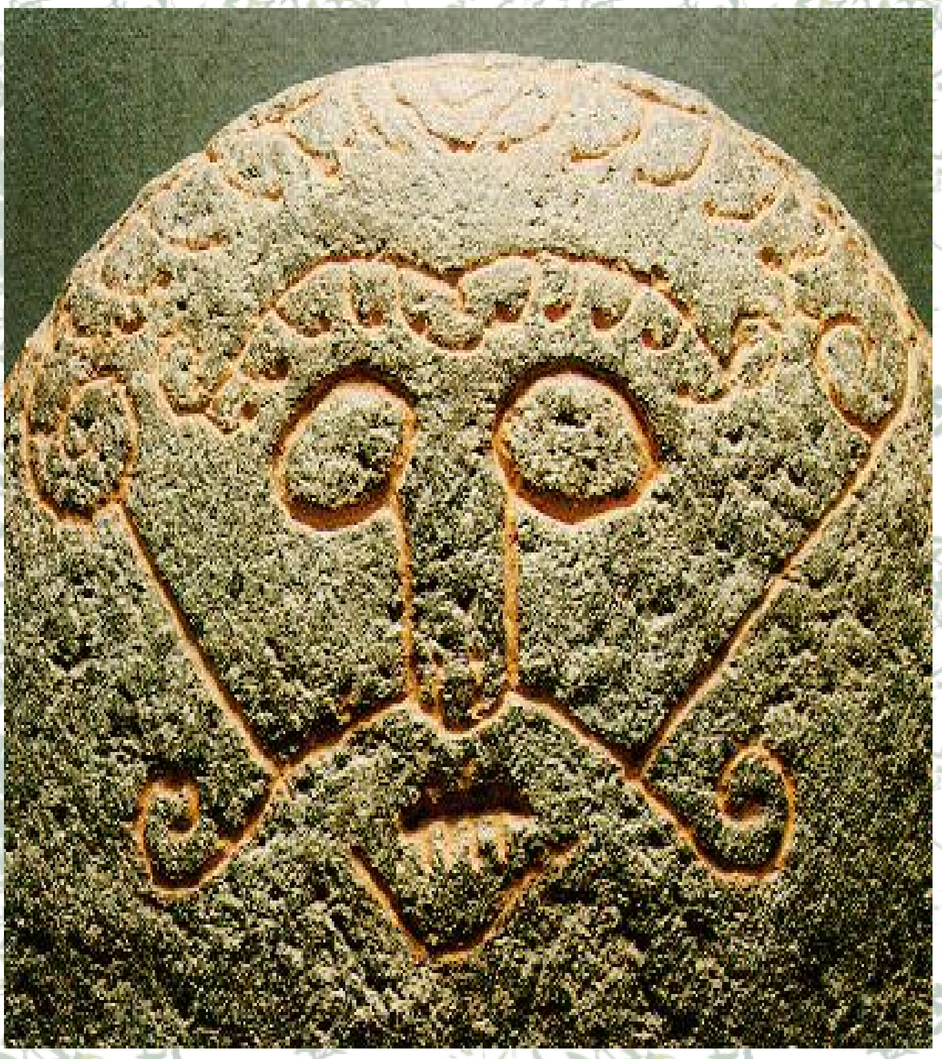
- Need for resurrection of *"easy" tagging*

```
// create tagger object
Tagger tag = tagger("My B-Tagger");
// tag and associate B-candidate
tagger->tag( B ) ;
// extract all tags for given B
TRange tags = tagger->tags( B ) ;
```

LoKi I

- Loki is a god of wit and mischief in Norse mythology
- **Lo**ops & **Ki**nematics

LoKi II



LoKi III