Bender Tutorial

Vanya Belyaev
LAPP/Annecy-le-Vieux & ITEP/Moscow

Outline

- **Bender/Python** overview
- Job configuration
- Data access
- Histograms & N-Tuples
- Algorithms

Bender is not frozen!

Significant improvements in **Bender** semantics are expected (mainly according to the feedback from you)

- Please keep **LoKi** manual with you
  - Especially for **Function/Cuts** tables

$LHCBRELEASES/BENDER/BENDER_v4r1/Doc/LoKiDoc/v3r5/doc/LoKi'*'.(ps,pdf)

- **Doxygen** for **LoKi** is also useful

$LHCBRELEASES/BENDER/BENDER_v4r1/Doc/LoKiDoc/v3r5/doc/html/index.html

Environment (I)

- **Bender v4r1** (based on **DaVinci v12r2**)
- The package **Tutorial/BenderTutor v1r0**
- Only few essential features of **Bender**
- Out of Tutorial scope
  - visualization of histograms
  - visualization of event and detector data
  - CMT-free mode
  - batch jobs

Environment (II)

- get the Tutorial package

```
lbcmt
BenderEnv v4r1
cd $HOME/cmtuser
getpack Tutorial/BenderTutor v1r0
cd Tutorial/BenderTutor/v1r0/cmt
make
source setup.csh ( or . setup.sh
```

To be substituted by Bender + cmt.py

Bender/Python tips

- Python scripts could be executed as "scripts"

```
>    python MyBenderScript.py
>           MyBenderScript.py
```

- Python scripts could be executed from the command prompt (explicit interactivity!)

```
> python
     >>>    import MyBenderScript
```

- Python scripts could be executed with the command prompt (interactivity with "**pawlogon.kumac**")

```
> python –i MyBenderScript.py
```

Common start-up script is possible,
Pere has a lot of nice ideas!

Structure of Gaudi Job

Each "Job" contains 4 essential part

- Configuration of Job environment
  - <ProjectEnv> scripts, CMT

Bender: cmt.py

- Configuration of Job's components
  - Top Level algorithms
  - properties of Algorithms/Services/Tools
  - Input/output

GaudiPython + Bender

- "Analysis Algorithm" coding

Bender

- Job steering

GaudiPython + Bender

2 approaches

Start from pure python prompt

- define everything from Python

Attractive,
but not practical

Make a "smooth" transition from DaVinci/LoKi

- start with existing configuration
- substitute it element by element

Choice for tutorial

Minimal Analysis Job

- **Bender** could be used with "no **Bender**"
- Execute some "**DaVinci**" configuration
- The actual configuration from '*'`.opts` file

- **DaVinci:**

```
DaVinci MyOptionsFile.opts
```

Minimal Bender script

```
from   bendermodule import *
import benderconfig as bender

def configure() :
  bender.config( files =
                [‘MyOptionsFile.opts’])
  return SUCCESS

if __name__ == ‘__main__’ :
  configure()
  g.run(100)
  #g.exit()
```

To be improved

Application and Components Configuration

Job steering

“g” → “gaudi” ? “appMgr”?

../solutions/Minimalistic.py

“Hello, World!” (I)

- The simplest possible “algorithm”
- Follow **LoKi**’s style:
  - inherit the algorithm from useful base class
  - (re)implement the “**analyse**” method

```
class HelloWorld(Algo) :
   def analyse( self ) :
      print ‘Hello, World!’
      return SUCCESS
```

../solutions/HelloWorld.py

"Hello, World!" (II)

- One needs to instantiate the algorithm

```
alg = HelloWorld( 'Hello' )
```

- Add it to the list of 'active' algorithms

```
g.TopAlg  = [ 'Hello' ]
```

Application Configuration

- Execute ☺

```
g.run(10)
```

Part of job steering block

../solutions/HelloWorld.py

Access to the data

- **C++: GaudiAlgorithm/LoKi**

```
const MCParticles* mcps =
   get<MCParticles>('MC/Particles' )
```

- **Python: Bender**

Semantics to be improved

  - Get as 'native' object:

```
mcps = self.get( address = 'MC/Particles')
```

  - Get as `std::vector` or Python's `list`:

```
mcps = self.get( address = 'MC/Particles' ,
                 vector  = TRUE )
mcps = self.get( address = 'MC/Particles' ,
                 list    = TRUE )
```

../solutions/DataAccess.py

Attributes and (python) loops

```
for mcp in mcps :
  print 'ID=' , nameFromPID( mcp.particleID() )
  print 'PX=' , mcp.momentum().px()
  print 'PY=' , mcp.momentum().py()
```

MCParticle

From Dictionaries

- To know the available attributes:

```
help( obj )
help( type( obj ))
```

- ON-LINE help for ALL Python/Bender functions/classes, sometimes it is VERY useful

../solutions/DataAccess.py

Hands-on (I)

- Simple algorithm which gets **`MCVertices`** from the Gaudi Transient Store and prints number of **`MCVertices`** and some information (e.g. **`x/y/z`**-position) for some of them

Hints:

- The **`'*'.opts`** file, which could be used
  - **`$BENDERTUTOROPTS/BenderTutor.opts`**
- The analogous example for **`MCParticles`**:
  - **`../solutions/DataAccess.py`**
- The actual solution is
  - **`../solutions/HandsOn1.py`**

Lets start with physics analysis

- **>95%** of **LoKi**'s idioms are in **Bender**
- The semantic is VERY similar
  - In spite of different languages
  - few 'obvious' exceptions
- In the game:
  - All **Functions/Cuts**
    - a bit more round braces are required
  - All **(v,mc,mcv) select** methods
  - **loops , plots**
  - for **N-Tuples** the functionality is a bit limited
    - A lack of template methods,
    - 'farray' need to be validated

Start from MC-truth (requires no special configurations)

MCselect statement

- Selection of **MCParticles** which satisfy the certain criteria:

LUG, Tab. 13.4, p.84

```
mcmu = self.mcselect( tag = 'mcmu' ,
                      cuts = 'mu+' == MCABSID )
beauty = self.mcselect( tag = 'beauty' , cuts = BEAUTY )
```

Select μ^+ & μ^-

Everything which has b or b̄

- Refine criteria:

```
muFromB = self.mcselect ( tag = 'muFromC',
                          source = mcmu ,
                  cuts = FROMMCTREE( beauty ) )
muPT = self.mcselect( tag = 'withPT' ,
                      source = muFromB ,
                      cuts = ( MCPT > ( 1 * GeV ) ))
```

Everything from "decay" trees (incl. decay-on-flight)

../solutions/MCmuons.py

Change input data

- Get and configure **EventSelector**

```
evtSel = g.evtSel()
evtSel.open( "file")
```

OR

```
evtSel.open( [ "file1", "file2"] )
```

List of input files

- e.g.

```
evtSel.open ( 'LFN:/lhcb/production/DC04/v1/DST/00000543_00000017_5.dst')
```

Hands On (II, II.5)

- Simple algorithm which evaluates the fractions of events which contains of at least B_s or beauty baryons

Hints

- Relevant **MCParticle** functions

  MCID, **MCABSID** , **BEAUTY** , **BARYON**

  LUG, Tab. 13.4, p.84-87

- The most trivial "counter" is

```
if not Bs.empty() : self.Warning( message = 'Bs' )
```

- The analogous algorithm is
  - **../solutions/MCmuons.py**
- The real solution is
  - **../solutions/HandsOn2.py**
  - **../solutions/HandsOn2.5.py**

Find MC-tree (IMCDecayFinder)

Brilliant tool from O.Dormond

- find the MC-decay trees:

```
mc = self.mctruth()
trees = mc.find( decay =
      '[B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc' )
```

Container("Range") of MCParticles

- find MC-decay tree components:

```
phis = mc.find( decay =
' phi(1020) : [B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc' )
```

Container("Range") of MCParticles

- extract 'marked' MC-decay tree components:

```
mus  = mc.find( decay =
     ' [B_s0 -> (J/psi(1S) -> mu+ ^mu-) phi(1020)]cc' )
```

../solutions/MCTrees.py

Add simple histos!

```
for mu in mus :
   self.plot ( title = 'PT of muon from J/psi' ,
               value = MCPT( mu ) / GeV ,
               high  = 10 )
```

MCParticle

The default values : `low = 0, bins = 100, weight = 1`

- Configuration for histograms:

```
g.HistogramPersistency = 'HBOOK'
hsvc = g.service('HistogramPersistencySvc')
hsvc.OutputFile = 'myhistos.hbook'
```

../solutions/MCTrees.py

Add the simple N-Tuple

```
tup   = self.nTuple( title = 'My N-Tuple' )
zOrig = MCVXFUN( MCVZ )
for mu in mus :
    tup.column( name = 'PT', value = MCPT( mu ) / GeV )
    tup.column( name = 'P' , value = MCP( mu ) / GeV )
    tup.column( name = 'Z' , value = zOrig( mu ) / mm)
    tup.write()
```

- Configuration:

```
myAlg = g.algorithm( 'McTree' )
myAlg.NTupleLUN = 'MC'
ntsvc = g.service('NTupleSvc')
ntsvc.Output =
 ["MC DATAFILE='tuples.hbook' TYP='HBOOK' OPT='NEW' "]
```

../solutions/MCTrees.py

Component Properties

- Algorithms

`MyAlg.NTupleLUN = "LUNIT" ;`

```
alg = g.algorithm('MyAlg')
alg.NTupleLUN = 'LUNIT'
```

- Services

`HistogramPersistencySvc.OutputFile = "histo.file";`

```
hsvc = g.service('HistogramPersistencySvc')
hsvc.OutputFile = 'histo.file'
```

- Tools

`MyAlg.PhysDesktop.InputLocations = {"/Event/Phys/Charged"};`

```
tool = g.property('MyAlg.PhysDesktop')
tool.InputLocations = ['/Event/Phys/Charged']
```

- Everything

```
prop = gaudi.iProperty('Holder.Name')
Prop.PropertyName = Value
```

`Holder.Name.PropertyName = Value ;`

Hands On (III)

- The algorithm which gets the kaons from the decay $B_s \to J/\psi\,(\,\phi \to K^+ K^-\,)$, fill histo and N-Tuple

Hints

- One need to define input MC files for this decay
  - see `../solutions/MCTrees.py`
- The similar algorithm
  - `../solutions/MCTrees.py`
- The actual solution
  - `../solutions/HandsOn3.py`

Go from MC to RC data

- At this moment one knows how to:
  - Deal with MC trees, decays, particles
  - Perform simple (`python`) loops
  - Deal with histograms & N-Tuples
    - Some knowledge of 'configuration'
- For RC data one must perform non-trivial algorithm configuration to be able to run
  - Input for RC particles (or ParticleMaker)
  - Dependency on 'other' algorithms (`'PreLoad'`)

Pre-Load Charged Particles (I)

```
g.TopAlg += [ 'LoKiPreLoad/Charged']

desktop = g.property('Charged.PhysDesktop')
desktop.ParticleMakerType = 'CombinedParticleMaker'
```

"Universal" configuration suitable almost for all everything

```
maker = g.property('Charged.PhysDesktop.CombinedPaticleMaker')
maker.ExclusiveSelection = 1>2
maker.Particles =
       [ 'muon' , 'electron' , 'kaon' , 'proton' , 'pion']
```

Very loose cuts, to be refined in the algorithm

```
maker.MuonSelection     = [ "det='MUON' mu-pi='-10.0' "]
maker.ElectronSelection = [ "det='CALO' e-pi='-2.0' "]
maker.KaonSelection = [ "det='RICH' k-pi='-5.0' k-p='-5.0' "]
maker.ProtonSelection   = [ "det='RICH' p-pi='-5.0' "]
maker.PionSelection     = [  "det='RICH pi-k='-5.0' "]
```

Complicated??

Pre-Load Charged Particles (II)

Could be done a bit easier:

```
g.TopAlg += [ 'LoKiPreLoad/Charged']
import benderPreLoad as preload
```

$BENDERPYTHON/benderPreLoad.py

../solutions/RCSelect.py

```
preload.Charged( Name='Charged' ,
    Kaons = [ "det='RICH' k-pi='-5.0' k-p='-5.0'" ] ,
    Pions = [ "det='RICH' pi-k='-5.0'" ]  )
```

- Alternatively (only hadrons, no $e^{\pm}/\mu^{\pm}$)

```
preload.Hadrons( Name='Charged' ,
    Kaons = [ "det='RICH' k-pi='-5.0' k-p='-5.0'" ] ,
    Pions = [ "det='RICH' pi-k='-5.0'" ]
```

Also for leptons ($e^{\pm}/\mu^{\pm}$)

Algorithm configuration

```
desktop = g.property('MyAlg.PhysDesktop')
desktop.InputLocations = [
  "/Event/Phys/Charged"]
```

- Similar semantic in configuration ('*'.opts) files:

```
MyAlg.PhysDesktop.InputLocations={"/Event/Phys/Charged"} ;
```

../solutions/RCSelect.py

select/loop statements

LUG, Tab. 13.2, p.62-77

```
muons = self.select ( tag = ‘mu’  ,
 cuts = ( ‘mu+’== ABSID ) & ( PT > (1*GeV) ) )
kaons = self.select ( tag = ‘K’  ,
 cuts = ( ‘K+’== ABSID ) & ( PIDK > 0 )   )
```

- Loops:

```
psis=self.loop(formula=‘mu mu’,pid=‘J/psi(1S)’)
phis=self.loop(formula=‘K K’,pid=‘phi(1020’)
```

../solutions/RCSelect.py

Inside the loops (I)

```
dmcut = ADMASS('J/psi(1S)') < ( 50 * MeV )
for psi in psis :
    if not 2.5*GeV< psi.mass(1,2) <3.5*GeV : continue
    if not 0 == SUMQ( psi )          : continue
    if not 0 <= VCHI2( psi ) < 49 : continue
    self.plot ( title = " di-muon invariant mass" ,
                value = M( psi ) / GeV    ,
                low = 2.5 , high = 3.50 )
    if not dmcut( psi ) : continue
    psi.save('psi')

psis = self.selected('psi')
print '# of selected J/psi candidates:', psis.size()
```

$\Sigma q = 0$

$\chi^2_{VX} < 49$

$|\Delta M| < 50\ MeV/c^2$

../solutions/RCSelect.py

Inside the loops (II)

```
dmcut = ADMASS(‘phi(1020’) < ( 12 * MeV )
for phi in phis :
    if not phi.mass(1,2) < 1050*MeV : continue
    if not 0 == SUMQ( phi )         : continue
    if not 0 <= VCHI2( phi ) < 49 : continue
    self.plot ( title = “ di-kaon invariant mass” ,
                value = M( phi ) / GeV    ,
                low = 1.0 , high = 1.050 )
    if not dmcut( phi ) : continue
    phi.save(‘phi’)

phis = self.selected(‘phi’)
print ‘# of selected phi candidates:‘, phis.size()
```

$\Sigma q = 0$

$\chi^2_{VX} < 49$

$|\Delta M| < 12\ \mathrm{MeV}/c^2$

../solutions/RCSelect.py

Inside the loops (III)

```
dmcut = ADMASS('B_s0' ) < ( 100 * MeV )
bs = self.loop ( formula = 'psi phi' , pid = 'B_s0' )
for B in bs :
    if not 4.5*GeV < B.mass(1,2) < 6.5*GeV : continue
    if not 0 <= VCHI2( B ) < 49 : continue
    self.plot ( title = " J/psi phi invariant mass" ,
                value = M( B ) / GeV    ,
                low = 5.0 , high = 6.0 )
    if not dmcut( B ) : continue
    B.save('Bs')

Bs = self.selected('Bs')
print '# of selected Bs candidates:', Bs.size()
if not Bs.empty() : self.setFilterPassed ( TRUE )
```

../solutions/RCSelect.py

The last step: MC-truth match

- The simplest case: check if RC particle originates from the certain MC-(sub)tree
  - The most frequent case
    - Check for efficiencies
    - Resolution
- The opposite task: what MC particle "corresponds" to RC particle
  - similar (**MCTRUTH** → **RCTRUTH**)
- **NB**: **LoKi** (and **Bender**) uses own concept of MC "loose" matching
  - LUG, chapter 15

MC-truth match

```
mc = self.mctruth(‘MCdecayMatch’)
```

- Select MC-particles

```
mcBs  = mc.find( decay =
  `             [B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc ’ )
mcPhi = mc.find( decay =
  ` phi(1020) : [B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc ’ )
mcPsi = mc.find( decay =
  ` J/psi(1S) : [B_s0 -> (J/psi(1S) -> mu+ mu-) phi(1020)]cc ’ )
```

- Prepare ‘MC-Truth cuts’

```
mcCutBs  = MCTRUTH ( mc , mcBs  )
mcCutPhi = MCTRUTH ( mc , mcPhi )
mcCutPsi = MCTRUTH ( mc , mcPsi )
```

../solutions/RCMCSelect.py

The last step: MC-truth match

```
for psi in psis :
   if not mcCutPsi ( psi ) : continue
   …
for phi in phis :
   if not mcCutPhi ( phi ) : continue
   …
for B in bs :
   if not mcCutBs ( B ) :continue
   …
```

../solutions/RCMCSelect.py

- **Alternatively :**

```
for B in bs :
   psi = B(1)
   phi = B(2)
   …
   tup.column ( name = 'mcpsi' , value = mcCutPsi( psi ) )
   tup.column ( name = 'mcphi' , value = mcCutPhi( phi ) )
   tup.column ( name = 'mc'    , value = mcCutBs ( B ) )
   tup.write()
```

Hands On (IV)

- Simple algorithm which selects kaons, plot di-kaon invariant mass with and without MC-truth flags with different **PIDK** ($= \Delta_{LL}(K\text{-}\pi)$) values (& fill N-Tuple with such information)

Hints

- The relevant functions/cuts
  - `PIDK`, `MCTRUTH`
- The analogous algorithm
  - `../solutions/RCMCSelect.py`
- The actual solution
  - `../solutions/HandsOn4.py`

Histo visualization

- get the histogram

```
hsvc   = g.histSvc()
histo = hsvc['/stat/MyAlg/1']
from bender<XXX> import plotter
```

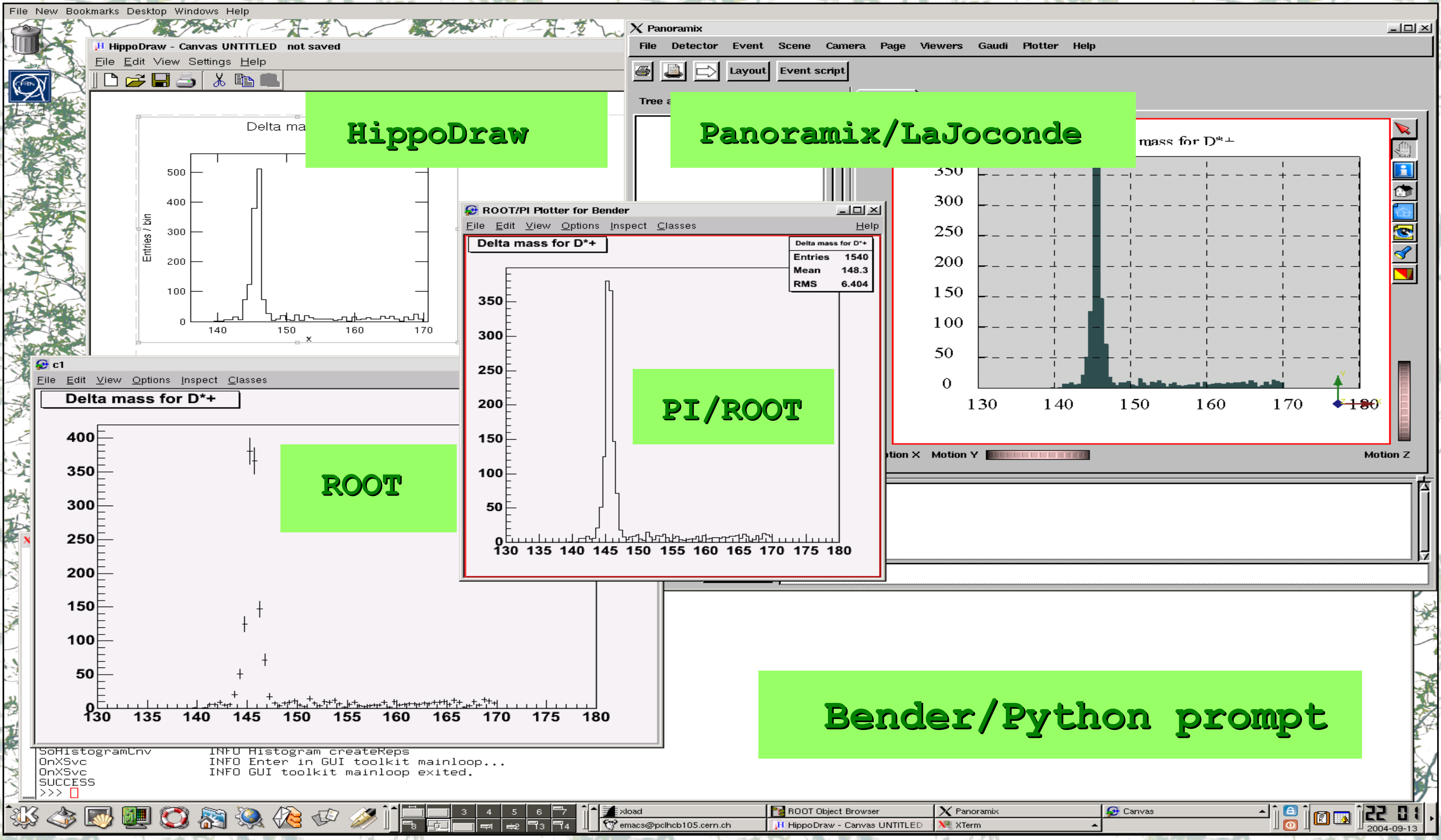
  - <XXX> = ROOT, PiRoot, PiHippo
    - Panoramix/LaJoconde – not for today ☹

```
plotter.plot(histo)
```

- for N-tuples: <XXX> = ROOT

```
g.HistogramPersistency = 'ROOT'
```

Everything can be combined



LoKi

Bender

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

Ostap Suleiman Berta Maria **Bender**-bei

- The cult-hero of books by *I.Il'f & E.Petrov: "The 12 chairs" ,"The golden calf"*
- The title: "The great schemer"
- Attractive & brilliant cheater

Essential for successful and good physics analysis