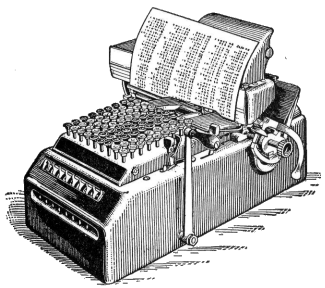
Future Analysis Strategies at Humboldt

End-User Analyses of ATLAS Event Data

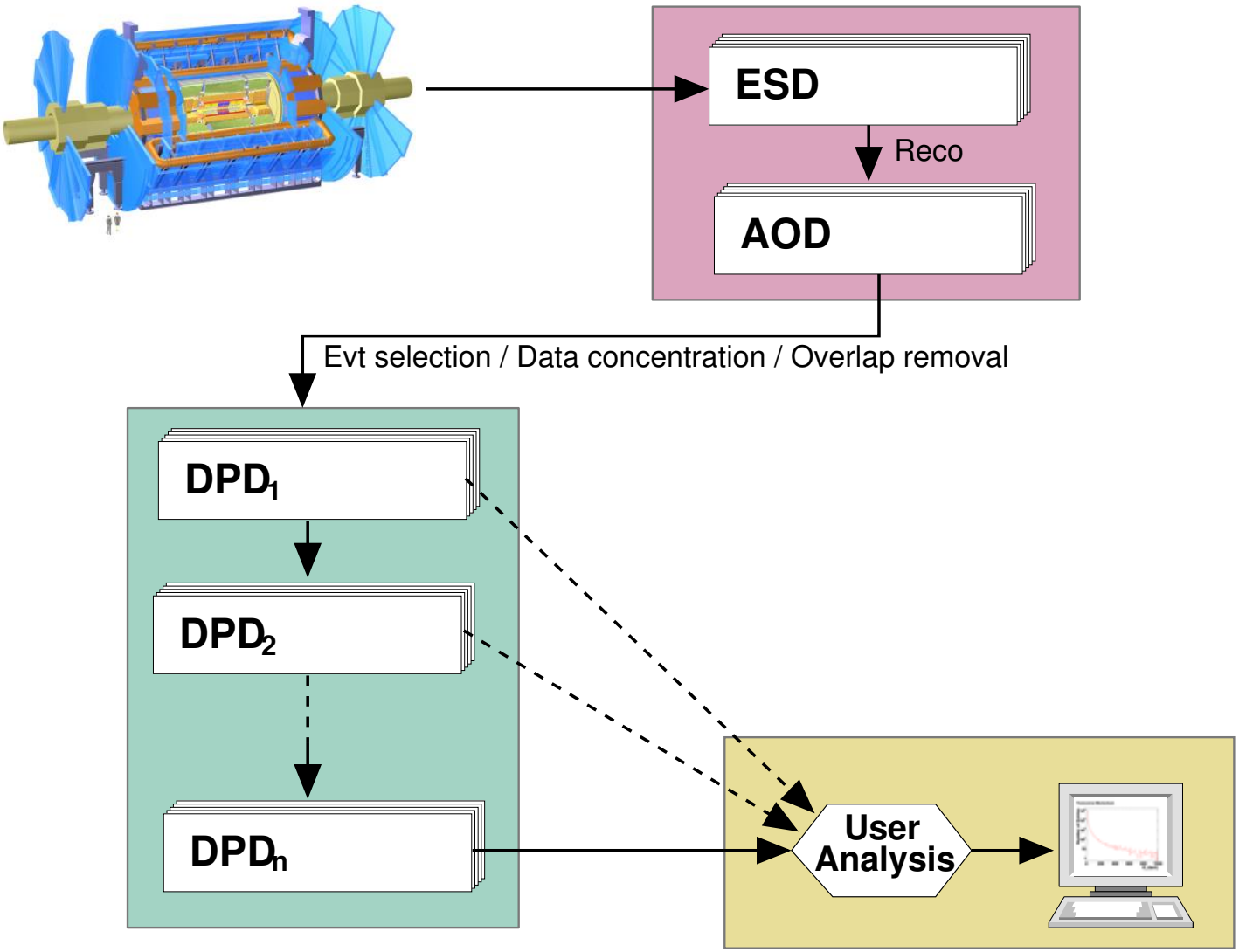
Oliver Maria Kind

Humboldt Universität zu Berlin

Zeuthen Meeting · June 17, 2008



Basic Analysis Chain @ATLAS



DPD & Existing Frameworks

1. **AOD & ATHENA**
   - No RooT interaction possible
   - Very slow
   - Version & tag hell
2. **AOD & AthenaRootAccess**
   - Depends on ATHENA ⇒ slow and version dependent
   - Strong tendency towards PYTHON and PyRooT
3. **EventView (TopView, SusyView, . . .) & SFrame**
   - Structureless data format ⇒ surrender all oo benefits
   - Animate users to write private analysis code
     ⇒ badly tested, bug–prone
   - Changes of variables force re-writing of analysis code
   - Documentation of Ntuple variables always broken
   - Likely to be dropped
4. **Other**
   - Athena Aware Ntuples (AAN) †
   - Structured Athena Ntuples (SAN) †

DPD & Existing Frameworks

1. **AOD & ATHENA**
   - No RooT interaction possible
   - Very slow
   - Version & tag hell
2. **AOD & AthenaRootAccess**
   - Depends on ATHENA $\Rightarrow$ slow and version dependent
   - Strong tendency towards PYTHON and PyRooT
3. **EventView (TopView, SusyView, ...) & SFrame**
   - Structureless data format $\Rightarrow$ surrender all oo benefits
   - Animate users to write private analysis code
     $\Rightarrow$ badly tested, bug–prone
   - Changes of variables force re-writing of analysis code
   - Documentation of Ntuple variables always broken
   - Likely to be dropped
4. **Other**
   - Athena Aware Ntuples (AAN) †
   - Structured Athena Ntuples (SAN) †

DPD & Existing Frameworks

1. **AOD & ATHENA**
   - No RooT interaction possible
   - Very slow
   - Version & tag hell
2. **AOD & AthenaRootAccess**
   - Depends on ATHENA $\Rightarrow$ slow and version dependent
   - Strong tendency towards PYTHON and PyRooT
3. **EventView (TopView, SusyView, . . .) & SFrame**
   - Structureless data format $\Rightarrow$ surrender all oo benefits
   - Animate users to write private analysis code
     $\Rightarrow$ badly tested, bug–prone
   - Changes of variables force re-writing of analysis code
   - Documentation of Ntuple variables always broken
   - Likely to be dropped
4. **Other**
   - Athena Aware Ntuples (AAN) †
   - Structured Athena Ntuples (SAN) †

DPD & Existing Frameworks

1. **AOD & ATHENA**
   - No RooT interaction possible
   - Very slow
   - Version & tag hell
2. **AOD & AthenaRootAccess**
   - Depends on ATHENA $\Rightarrow$ slow and version dependent
   - Strong tendency towards PYTHON and PyRooT
3. **EventView (TopView, SusyView, . . .) & SFrame**
   - Structureless data format $\Rightarrow$ surrender all oo benefits
   - Animate users to write private analysis code
     $\Rightarrow$ badly tested, bug–prone
   - Changes of variables force re-writing of analysis code
   - Documentation of Ntuple variables always broken
   - Likely to be dropped
4. **Other**
   - Athena Aware Ntuples (AAN) †
   - Structured Athena Ntuples (SAN) †

Conclusion

- For an efficient and bug–free analysis, the developmnent of a new analysis framework is inevitable.
- Code of different analyses must be shared within the framework.
  Time is saved and the code is tested in an orthogonal way.
  Gain experience in time.
- Avoid private user code as much as possible.

Conclusion

- For an efficient and bug–free analysis, the developmnent of a new analysis framework is inevitable.
- Code of different analyses must be shared within the framework.
  Time is saved and the code is tested in an orthogonal way.
  Gain experience in time.
- Avoid private user code as much as possible.

Conclusion

- For an efficient and bug–free analysis, the developmnent of a new analysis framework is inevitable.
- Code of different analyses must be shared within the framework.
  Time is saved and the code is tested in an orthogonal way.
  Gain experience in time.
- Avoid private user code as much as possible.

A New Framework — General Design Aspects

- **Data model**
  Use object–oriented approach
  - Coherent structure of complex event data
  - Integrate data structures and corresponding analysis code
  - Benefit from abstraction & encapsulation
- **Language**
  - Choose C++ since reasonably fast
    and widely used in HEP community ▸ Python ?
  - Base upon RooT class libraries
  - Do not use STL ▸ Why ? ▸ Bad example
    Use RooT's container classes & polymorphism instead
- **Project structure**
  - Pre-compiled shared libraries
  - Common CVS repository

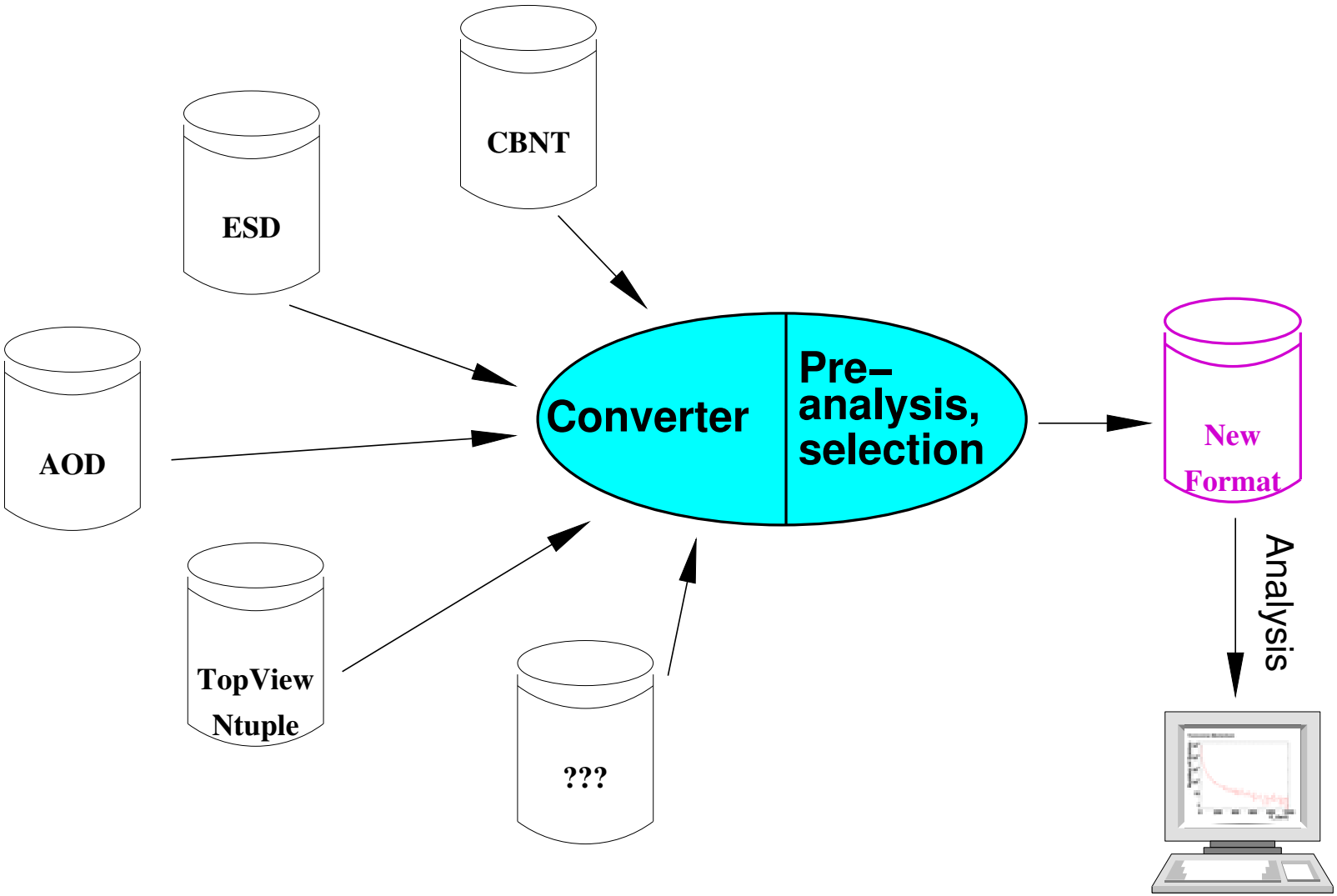
A New Framework — General Design Aspects

- **Data model**
  Use object–oriented approach
  - Coherent structure of complex event data
  - Integrate data structures and corresponding analysis code
  - Benefit from abstraction & encapsulation
- **Language**
  - Choose C++ since reasonably fast
    and widely used in HEP community ▸ Python ?
  - Base upon RooT class libraries
  - Do not use STL ▸ Why ? ▸ Bad example
    Use RooT's container classes & polymorphism instead
- **Project structure**
  - Pre-compiled shared libraries
  - Common CVS repository

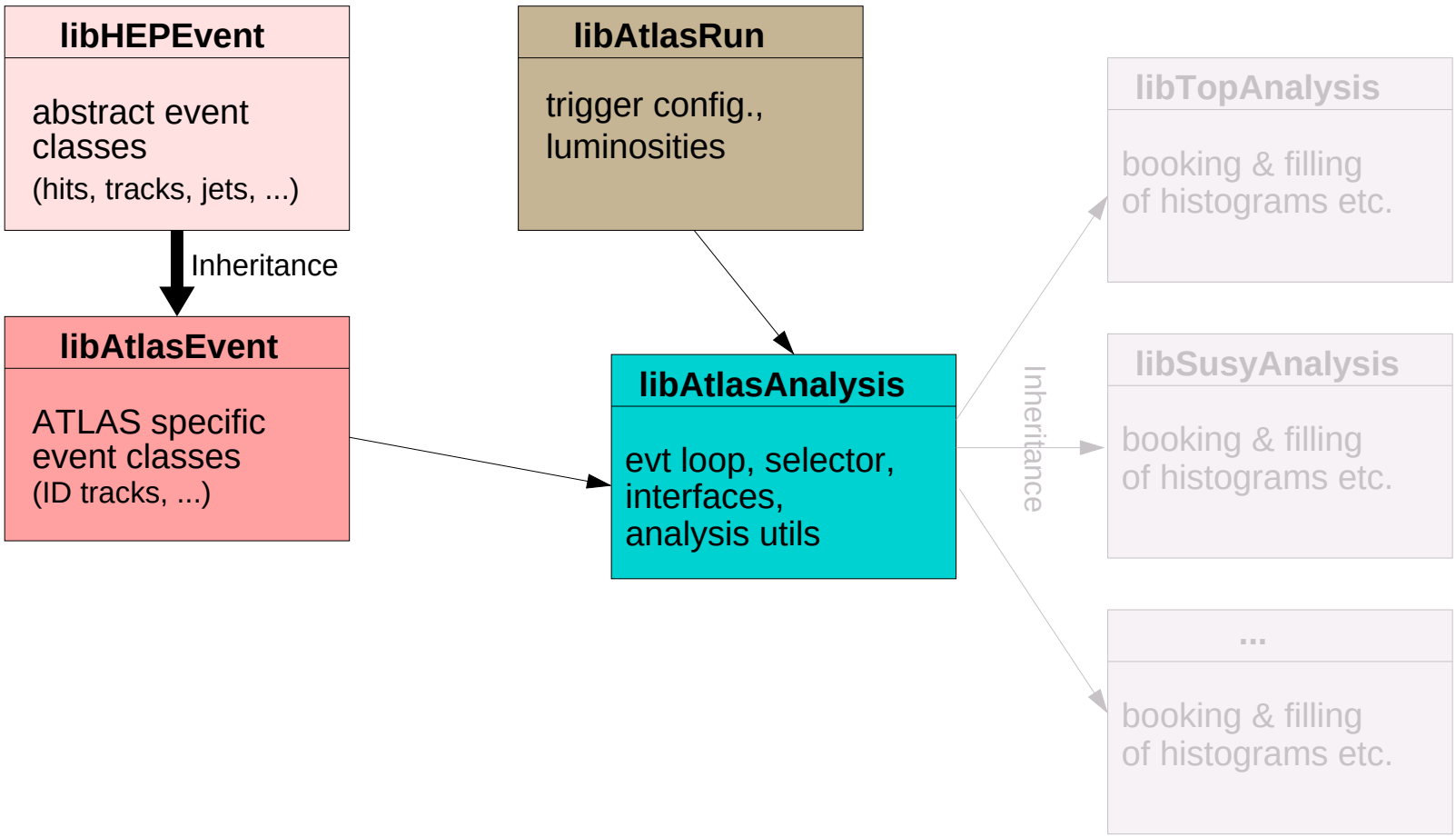
A New Framework — General Design Aspects

- **Data model**
  Use object–oriented approach
  - Coherent structure of complex event data
  - Integrate data structures and corresponding analysis code
  - Benefit from abstraction & encapsulation
- **Language**
  - Choose C++ since reasonably fast
    and widely used in HEP community ▸ Python ?
  - Base upon RooT class libraries
  - Do not use STL ▸ Why ? ▸ Bad example
    Use RooT's container classes & polymorphism instead
- **Project structure**
  - Pre-compiled shared libraries
  - Common CVS repository

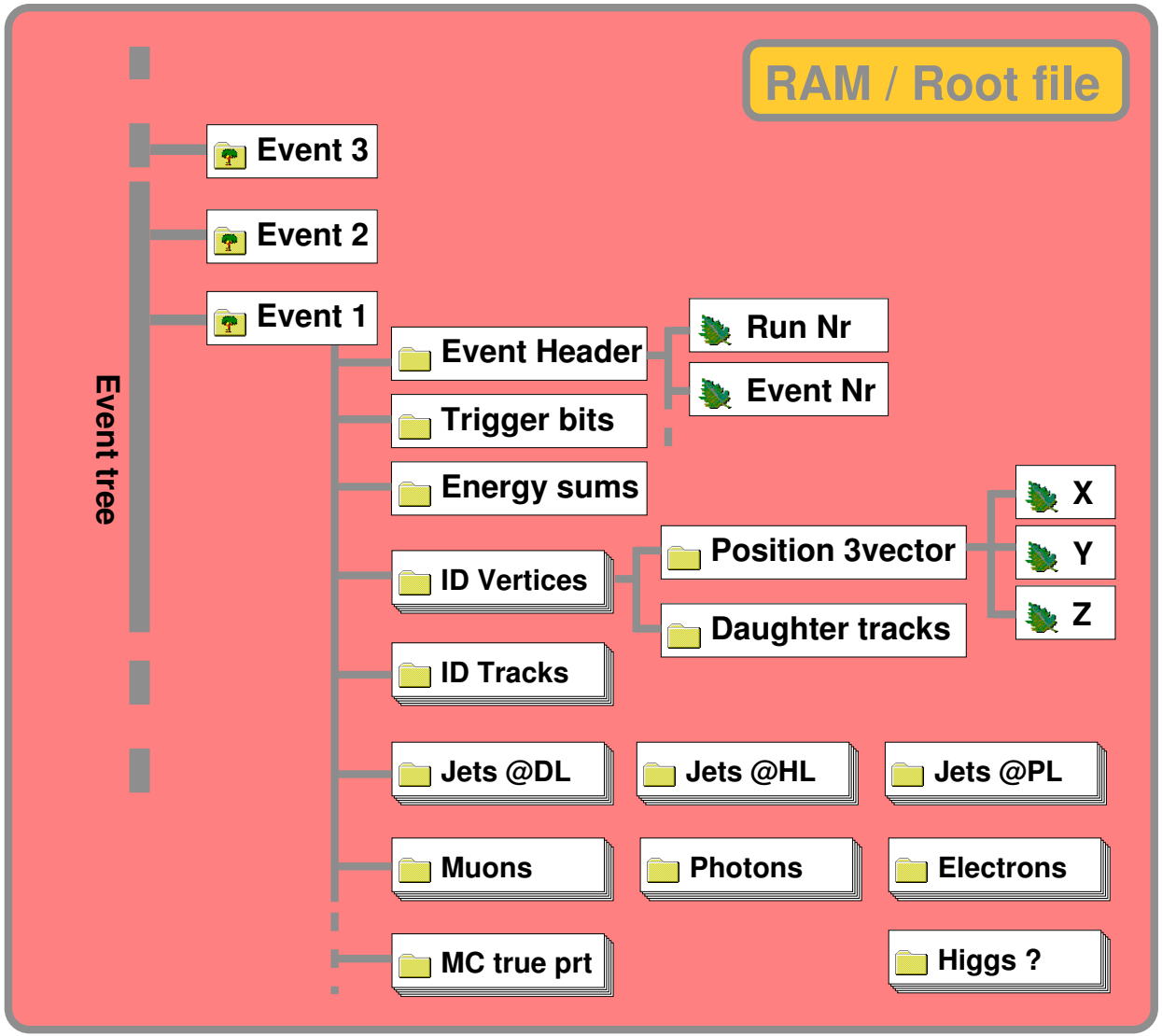
Basic Flowchart



Shared Library Structure



Event Tree & Event Data Format



Example: A Vertex Class

Date members:

```
class HepMCVertex : public TObject {

protected:
  Int_t       fId;          // Id number
  TVector3    fPos;         // Vertex position (cm)
  TRef       *fMother;      // Incoming particle
  TRefArray  *fDaughters;   // Outgoing particles
  ...
}
```

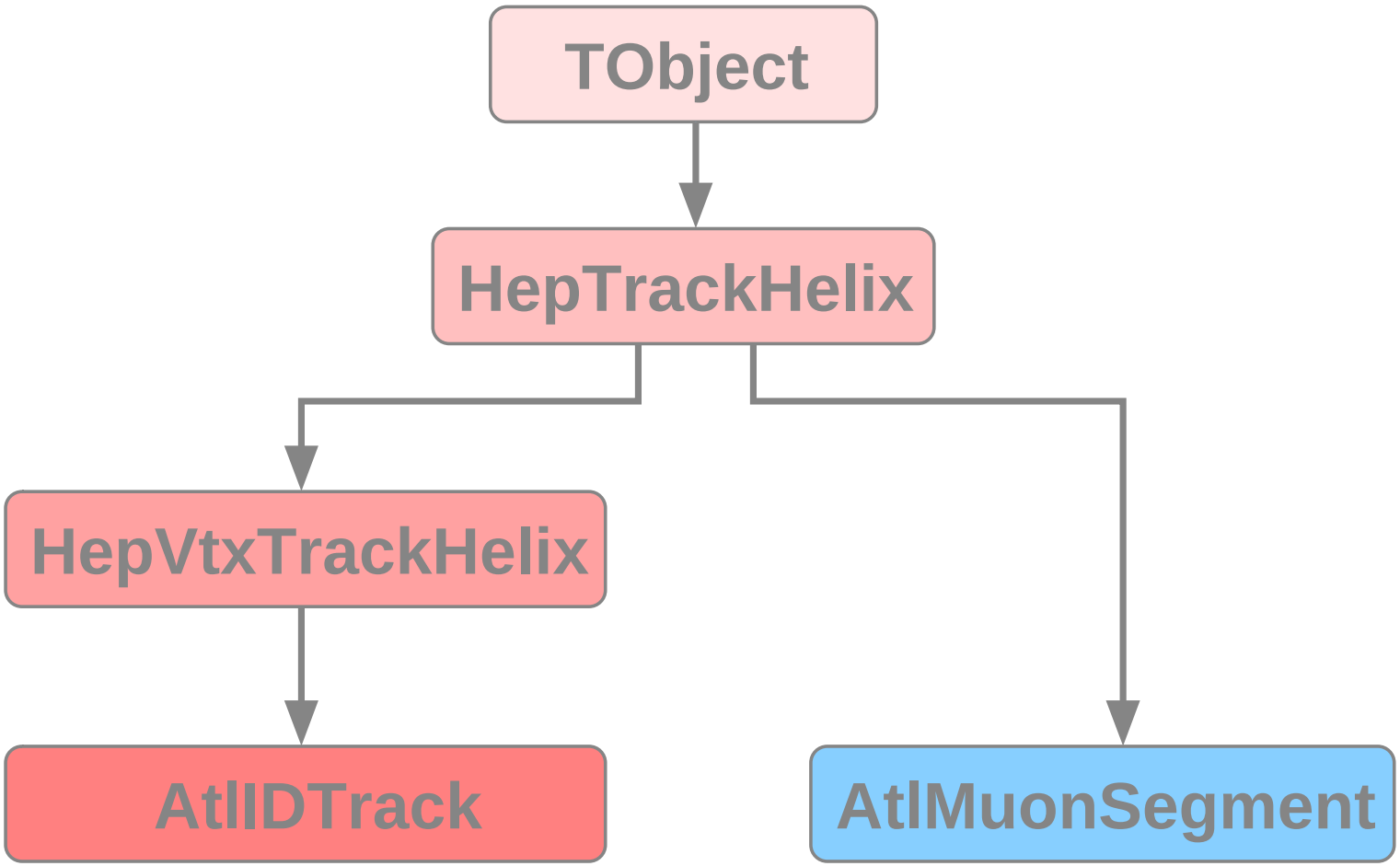
Example: A Vertex Class

Member functions:

```
public:
HepMCVertex();
HepMCVertex(Float_t x, Float_t y, Float_t z);
virtual ~HepMCVertex();
virtual void Clear(Option_t *option = "");
virtual void Print(Option_t *option = "");
void AddDaughter(HepMCParticle *Daughter);

inline Float_t X() { return fPos.X(); }
inline Float_t Y() { return fPos.Y(); }
...
```

Inheritance Example: Track Helices



Miscellaneous

- **Containers**
  Use RooT's TCloneArrays for list of tracks, jets, hits etc. which has much faster I/O due to the use of placement new operator
- **Relationships**
  Use RooT's TRef and TRefArray classes for implementing relationships (e. g. hits↔tracks, tracks↔vertices etc.) rather than index arrays
- **Branch status**
  Ability to switch on/off (whole blocks of) branches easily

```
tree->SetBranchStatus("fTracks*", kFALSE);
```

Miscellaneous

- **Containers**
  Use RooT's TCloneArrays for list of tracks, jets, hits etc. which has much faster I/O due to the use of placement new operator
- **Relationships**
  Use RooT's TRef and TRefArray classes for implementing relationships (e. g. hits↔tracks, tracks↔vertices etc.) rather than index arrays
- **Branch status**
  Ability to switch on/off (whole blocks of) branches easily

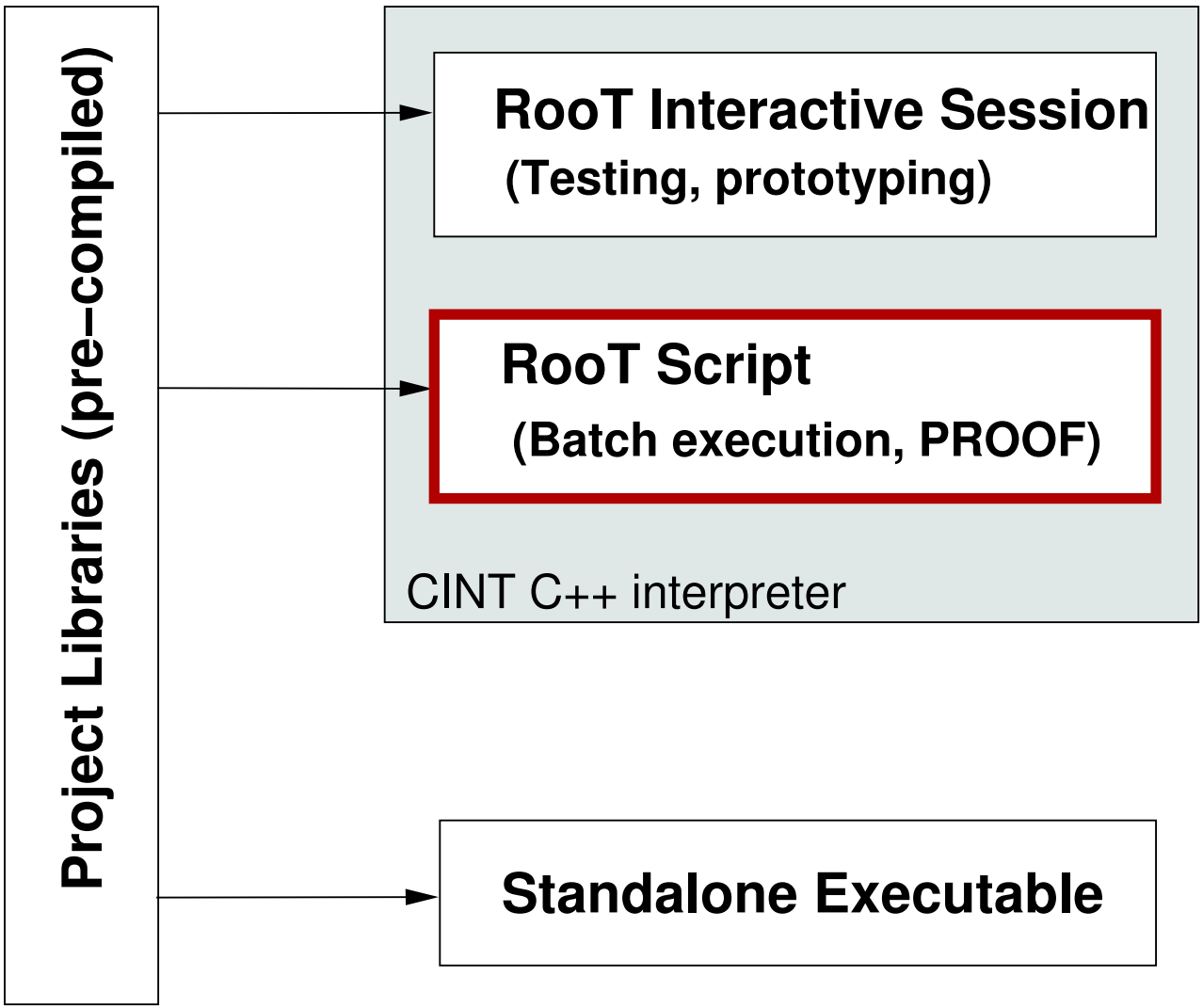
```
tree->SetBranchStatus("fTracks*", kFALSE);
```

Miscellaneous

- **Containers**
  Use RooT's TCloneArrays for list of tracks, jets, hits etc. which has much faster I/O due to the use of placement new operator
- **Relationships**
  Use RooT's TRef and TRefArray classes for implementing relationships (e. g. hits↔tracks, tracks↔vertices etc.) rather than index arrays
- **Branch status**
  Ability to switch on/off (whole blocks of) branches easily

```
tree->SetBranchStatus("fTracks*", kFALSE);
```

Ways of Running



Access Event Information

- **Mini–analysis (RooT or PRooF)**

```
root [0] new TBrowser();
root [1] tree->Scan("RunNr():EventNr()");
************************************
*    Row   *   RunNr() * EventNr() *
************************************
*        0 *      5200 *     47320 *
*        1 *      5200 *     47322 *
*        2 *      5200 *     47323 *
*        3 *      5200 *     47324 *

root [2] tree->Draw("fVertices.fPos.fZ");
root [3] tree->Draw("fJets.Et()");
root [4] tree->Draw("fJets.Eta()");
```

Access Event Information

- **Print whole event**

```
root [5] evt->Print()
```

```
================================================================================
 Run 5200   Event 47333   LumiBlock   BeamEnergy (GeV)
================================================================================
MC

Hadron Level Jets: #4
------------------------------------------------------------
 Id         Et          M         Theta         Phi          Eta
------------------------------------------------------------
   1     146.159      17.320      16.116     -92.258        1.955
   2     129.644      14.294      59.105     101.654        0.567
   3      61.349       7.546     109.196      -3.123       -0.341
   4      36.583       4.444      52.718      72.312        0.702
------------------------------------------------------------

MC particles: #10
------------------------------------------------------------------------------------
 Id     Type            E           P           Pt         Theta     Phi          Eta
------------------------------------------------------------------------------------
   1          t     670.028     646.771     173.417      15.553   -113.103       1.991
...
```

Access Event Information

- **Peek inside event**

```
root [6] evt->GetMCParticle_ById(2)->Print()
```

```
-------------------------------------------------------------------------------
 Id     Type           E           P           Pt         Theta      Phi          Eta
-------------------------------------------------------------------------------
  2           b     455.168     455.089     127.460      16.265    -92.260        1.946
-------------------------------------------------------------------------------
```

```
root [7] evt->GetMCParticle_ById(2)->DaughterOf()
             ->Print()
```

```
-------------------------------------------------------------------------------
 Id     Type           E           P           Pt         Theta      Phi          Eta
-------------------------------------------------------------------------------
  1           t     670.028     646.771     173.417      15.553   -113.103        1.991
-------------------------------------------------------------------------------
```

Access Event Information

- **A more complicated example**

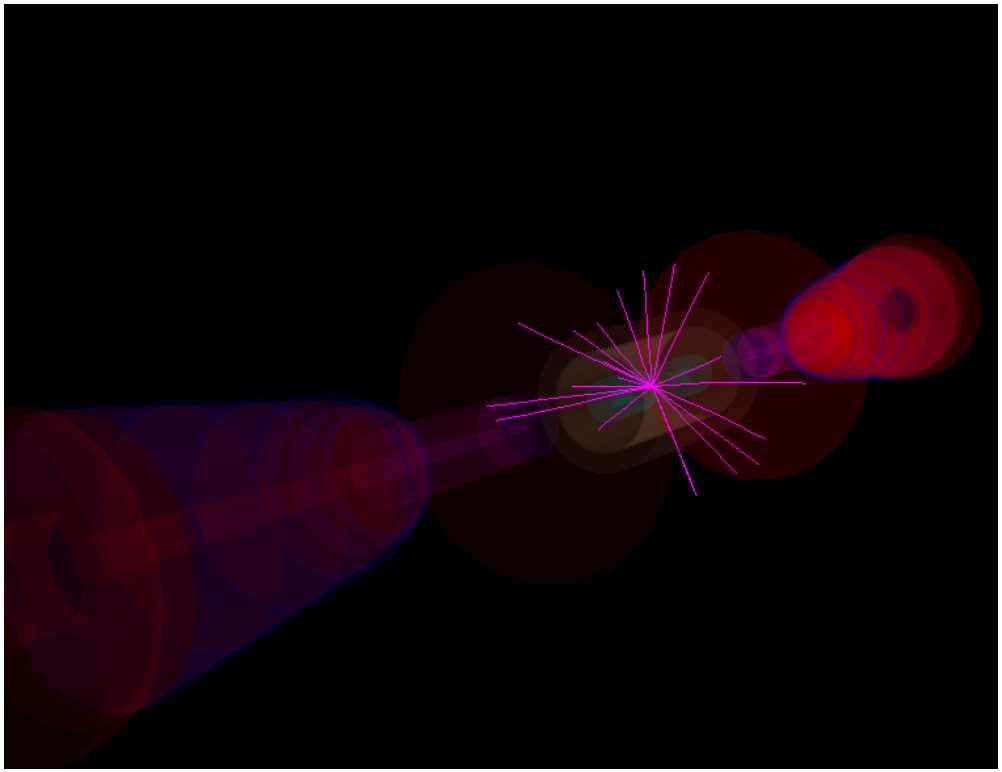
```
root [8] evt->PrintMCGenealogyTree()

t (ID=1, E=670.03GeV, Theta=16deg, Phi=-113deg)
|
+----- W+ (ID=5, E=214.86GeV, Theta=21deg, Phi=-153deg)
|       |
|       +----- nu_e (ID=6, E=103.58GeV, Theta=23deg, Phi=143deg)
|       |
|       +----- e+ (ID=7, E=111.28GeV, Theta=35deg, Phi=-118deg)
|
|
+----- b (ID=2, E=455.17GeV, Theta=16deg, Phi=-92deg)

t_bar (ID=3, E=303.63GeV, Theta=37deg, Phi=95deg)
|
+----- W- (ID=8, E=140.58GeV, Theta=8deg, Phi=39deg)
        |
        +----- s (ID=10, E=52.19GeV, Theta=47deg, Phi=83deg)
        |
        +----- c_bar (ID=9, E=88.39GeV, Theta=18deg, Phi=-69deg)
```

Event Display: RooT's EVE

```
root [9] evt->Draw();
```



- Display events immediatly from the RooT shell
- No (time–consuming) conversion needed anymore
- Neither ATHENA nor ATLANTIS necessary

More Development Tools

- Easy to use Makefiles
- Compile debug versions (also compiler flag)
- Create linker maps, profiling information
- Dump event content or object tables
  $\longrightarrow$ tracking of memory leaks

Automated HTML Documentation Generation

- RooT provides an easy way for automated hypertext documentation generation á la DOXYGEN
- The system produces complete HTML documents including
  - Class descriptions (made from comments in the source code)
  - Links to member functions, the source code and other classes
  - Class index
  - Inheritance charts
- How it works

```
// Create HTML documentation for all
// classes beginning with 'Hep' or 'Atl'
THtml html;
html.MakeAll(kFALSE, "Hep*");
html.MakeAll(kFALSE, "Atl*");
```

Automated HTML Documentation Generation

- RooT provides an easy way for automated hypertext documentation generation á la DOXYGEN
- The system produces complete HTML documents including
  - Class descriptions
    (made from comments in the source code)
  - Links to member functions, the source code
    and other classes
  - Class index
  - Inheritance charts
- How it works

```
// Create HTML documentation for all
// classes beginning with 'Hep' or 'Atl'
THtml html;
html.MakeAll(kFALSE, "Hep*");
html.MakeAll(kFALSE, "Atl*");
```

Automated HTML Documentation Generation

- RooT provides an easy way for automated hypertext documentation generation á la DOXYGEN
- The system produces complete HTML documents including
  - Class descriptions
    (made from comments in the source code)
  - Links to member functions, the source code
    and other classes
  - Class index
  - Inheritance charts
- How it works

```
// Create HTML documentation for all
// classes beginning with 'Hep' or 'Atl'
THtml html;
html.MakeAll(kFALSE, "Hep*");
html.MakeAll(kFALSE, "Atl*");
```

Example: A Jet Class

Location:	A++ »	.. »	AtlMCJet	
Quick Links:	ROOT	Class Index	Class Hierarchy	
Source:	header file	source file		
Sections:	class description	function members	data members	class charts

class AtlMCJet –
#include "../inc/AtlMCJet.h"
Display options:
☐ Show inherited
☑ Show non-public
[↑ Top] [? Help]

class AtlMCJet: public HepJet

Atlas MC truth jet

The jet is formed by taking all truth particles - with the exception of neutrinos, muons and non interacting particles - within a cone around the reconstructed jet.

For details see the Atlas Wiki.

Author: Oliver Maria Kind <mailto: kind@mail.desy.de>
Update: $Id: AtlMCJet.cxx,v 1.3 2008/04/14 21:08:14 kind Exp $
Copyright: (C) Oliver Maria Kind

Function Members (Methods)

public:

	AtlMCJet ()
	AtlMCJet (const AtlMCJet&)
	AtlMCJet (Int_t Id, Float_t E, Float_t Px, Float_t Py, Float_t Pz)
virtual	**~AtlMCJet** ()
static TClass*	**Class** ()
virtual TClass*	**IsA** () const
AtlMCJet&	**operator=** (const AtlMCJet&)

virtual void **ShowMembers** (TMemberInspector& insp, char* parent)
virtual void **Streamer** (TBuffer& b)
void **StreamerNVirtual** (TBuffer& b)

protected:

class AtlMCJet –
#include "../inc/AtlMCJet.h"
Display options:
☐ Show inherited
☑ Show non-public
[↑ Top] [? Help]

Data Members

public:

protected:

Class Charts

Class Charts

Function documentation

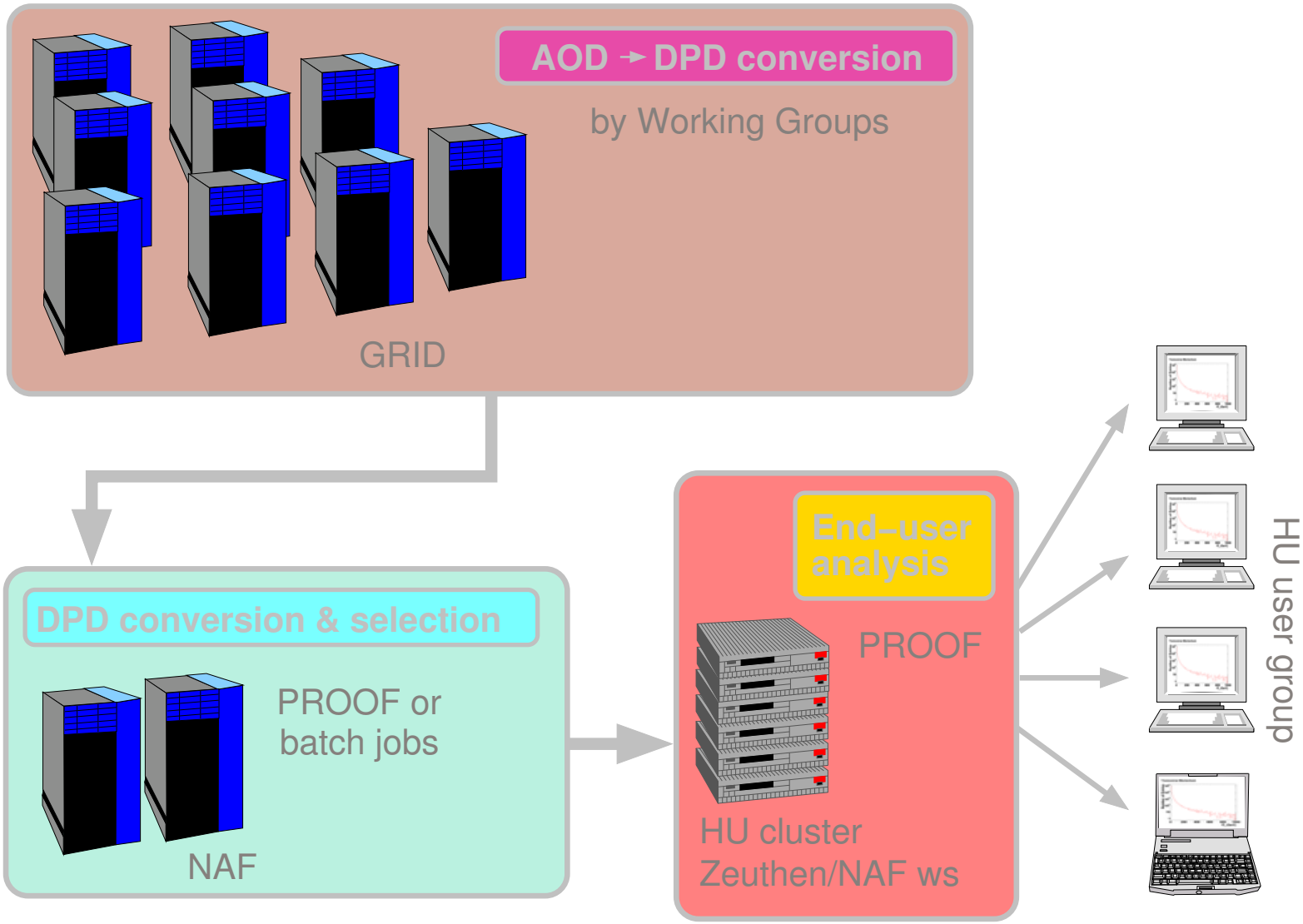
AtlMCJet (const AtlMCJet&)

Default constructor

AtlMCJet (Int_t Id, Float_t E, Float_t Px, Float_t Py, Float_t Pz)

Default constructor

Analysis Strategy



Project Status

- 3 developers
- Basic structure finished
- Extend event classes
- CBNT & TopView interfaces working (not completed yet)
- Work on ESD &AOD interfaces started

Summary

- Integrated analysis framework for ATLAS
- Object–oriented, heavily based upon RooT libraries
- Used for end–user analyses;
  avoid private analysis code
- Interfaces for all common ATLAS data formats
- Provides many tools for rapid development and testing

Summary

- Integrated analysis framework for ATLAS
- Object–oriented, heavily based upon RooT libraries
- Used for end–user analyses; avoid private analysis code
- Interfaces for all common ATLAS data formats
- Provides many tools for rapid development and testing

Summary

- Integrated analysis framework for ATLAS
- Object–oriented, heavily based upon RooT libraries
- Used for end–user analyses;
  avoid private analysis code
- Interfaces for all common ATLAS data formats
- Provides many tools for rapid development and testing

Summary

- Integrated analysis framework for ATLAS
- Object–oriented, heavily based upon RooT libraries
- Used for end–user analyses;
  avoid private analysis code
- Interfaces for all common ATLAS data formats
- Provides many tools for rapid development and testing

Summary

- Integrated analysis framework for ATLAS
- Object–oriented, heavily based upon RooT libraries
- Used for end–user analyses;
  avoid private analysis code
- Interfaces for all common ATLAS data formats
- Provides many tools for rapid development and testing

Why Do Not Use Python ?

- Against RooT's philosophy of "one language for all"
- Slow compared to compiled code
- Start from scratch again when write C++ code able to compile
  → waste of time, more bugs
- RooT integration is not perfect
- Scatter knowledge

Return

Why Do Not Use STL Data Structures ?

- Code bloat
- Code hard to comprehend for un-experienced developers
- Bug–tracking almost impossible due to additional layer of generated code (in particular true for debugger)
- Not optimised for I/O (c. f. with RooT's TClonesArray class)
  $\Rightarrow$ large decrease in analysis speed

Return

Bad Example

TopView Code

```
// TopView IDs of all particles
vector< int,allocator<int> > *Barcodes = Truth0.Tru_barcode;
vector<int>::iterator Barcodes_Iter1;
vector<int>::iterator Barcodes_Iter2;


// List of parent arrays for all particles
vector< vector<double> > *ParentList = Truth0.Tru_ParentBarcodes;
vector< vector<double> >::iterator ParentList_Iter;
// Parent of a particle (single-entry list)
vector<double> Parent;
vector<double>::iterator Parent_Iter;

// List of daughters arrays for all particles
vector< vector<double> > *DaughtersLists = Truth0.Tru_DecayBarcodes;
vector< vector<double> >::iterator DaughtersLists_Iter;
// Daughter array for a single particle
vector<double> Daughters;
vector<double>::iterator Daughters_Iter;

// Loop over all MC particles and look for their daughters
// and grand-daughters
Int_t id_prt = 0;
HepMCParticle *prt = 0;
HepMCParticle *daughter = 0;
for ( DaughtersLists_Iter = DaughtersLists->begin();
      DaughtersLists_Iter != DaughtersLists->end();
      DaughtersLists_Iter++ ) {
  Daughters = *DaughtersLists_Iter;
  prt = fEvent->GetMCParticle_ById(++id_prt);

  // Loop over all daughters of each particle
  for ( Daughters_Iter = Daughters.begin();
        Daughters_Iter != Daughters.end(); Daughters_Iter++ ) {

    // Check if the daughter particle is contained in the
    // list of TopView's MC particles
    Bool_t exit1 = kFALSE;
    Int_t id_daughter = 0;
    for ( Barcodes_Iter1 = Barcodes->begin();
          Barcodes_Iter1 != Barcodes->end();
          Barcodes_Iter1++ ) {
      id_daughter++;
      if ( *Daughters_Iter == *Barcodes_Iter1 ) {
        daughter = fEvent->GetMCParticle_ById(id_daughter);
        prt->AddDaughter(daughter);
        daughter->SetMother(prt);
        break;
      } else {
        // In case the daughter particle was not found
        // in the lst of particles check if any of the
        // grand-daughters are known by comparing daughter
        // and parent IDs. If the daughter is a parent of
        // some known particle contract the mother-grand-daughter
        // relationship to a mother-daughter relationship and
        // store it this way.
        Barcodes_Iter2 = Barcodes->begin();
        Int_t id_granddaughter = 0;
        for ( ParentList_Iter = ParentList->begin();
              ParentList_Iter != ParentList->end();
              ParentList_Iter++ ) {
          Parent = *ParentList_Iter;
          Parent_Iter = Parent.begin();
          id_granddaughter++;
          if ( *Daughters_Iter == *Parent_Iter ) {
            daughter = fEvent
              ->GetMCParticle_ById(id_granddaughter);
            prt->AddDaughter(daughter);
            daughter->SetMother(prt);
            exit1 = kTRUE;
            break;
          }
          Barcodes_Iter2++;
        }
        if ( exit1 == kTRUE ) break;
      }
    }
  }
}
```

New Framework

```
for ( Int_t i = 0; i < evt->GetN_MCParticles(); i++ ) {
  HepMCParticle *prt = (HepMCParticle*)evt->GetMCParticles()->At(i);
  HepMCParticle *mother = prt->DaughterOf();
}
```


Return