Hall A C++ Analyzer

Ole Hansen

Jefferson Lab

Hall C Users Meeting
January 23, 2010

http://hallaweb.jlab.org/root/

Outline

1. Overview
2. Elements
   - Analysis Objects
   - Steering & Control
   - Database
3. Use & Customization

Outline

1. Overview
2. Elements
   - Analysis Objects
   - Steering & Control
   - Database
3. Use & Customization

Outline

1. Overview
2. Elements
   - Analysis Objects
   - Steering & Control
   - Database
3. Use & Customization

Podd – the Hall A C++ Analyzer

- Standard Hall A analysis software since 2003
- Class library on top of ROOT
- Toolbox of analysis modules
- Predefined modules for many generic analysis tasks
  plus for Hall A equipment
- Analysis controlled via ROOT (C++) scripts (& text input files)
- Compiled code: special emphasis on modularity
  - "Everything is a plug-in"
  - User code separate from core code
  - Load external user libraries dynamically at run time
  - Users write no more than the code really needed
  - Core analyzer suitable for fixed installation, like ROOT itself

Features I

- Interactive interface (analyzer classes plus all of ROOT)
- Predefined “analyzer” (initialization, event loop)
- Analysis modules store results in “global variables”
- Tests and output can be defined dynamically (no recompilation)
- Time-dependent database (in flat text files)
- SDK available for rapid development of plug-ins

Features II

- CODA decoder with support for EPICS & scaler data
- Track reconstruction for HRS & BigBite
- Analysis of generic scintillators, Cherenkovs, shower counters
- BPM & raster analysis
- Vertex calculation
- Helicity analysis (prompt & delayed modes)
- Kinematics calculations for common reactions:
  (e, e'), $(e, e'X)$, (γ, X)
- Coincidence time analysis
- Extended target (x_0) corrections
- Energy loss corrections
- Dead-time calculation
- Decoding of generic hardware channels

Analysis Objects

- Any class that produces “results”
- Every analysis object has unique name, e.g. R.s1
- Results stored in “global variables”, prefixed with name:
  e.g.: R.s1.nhits
- `THaAnalysisObject` common base class:
  - Support functions for database access (keys prefixed with name)
  - Support functions for global variable handling
- Actual objects implement various virtual functions
  - `DefineVariables()`
  - `ReadDatabase()`
  - `Init()`
  - etc.

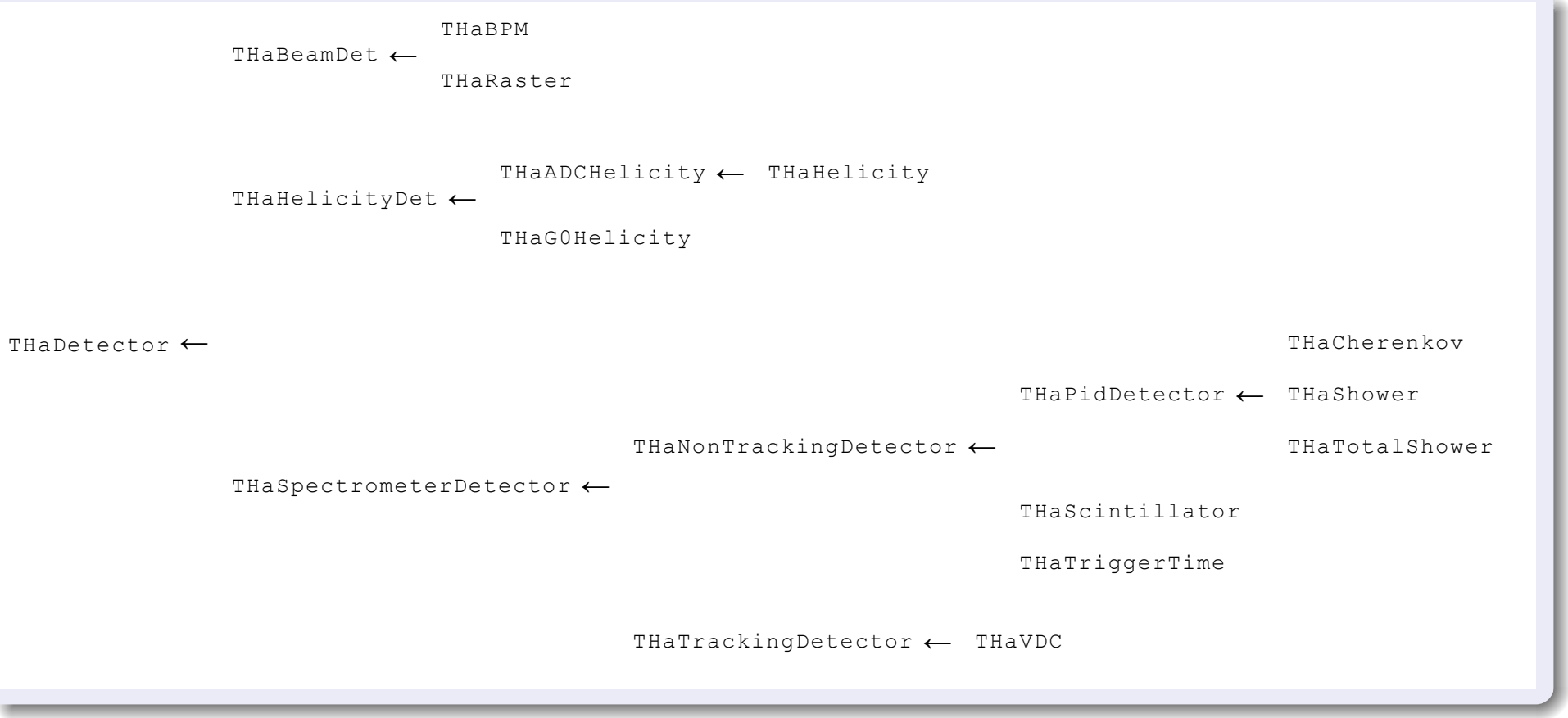
Types of Analysis Objects

- “Detector”
- “Apparatus”
- “Physics Module”

Detector Classes

- Code/data for analyzing a type of detector.
  Examples: Scintillator, Cherenkov, VDC, BPM
- Generic design: One class describes any number of similar detectors, differing only by name and database entries
- Must do `Decode()`
- Embedded in Apparatus or standalone
- Must not assume presence of other detectors

Detector Class Hierarchy



Apparatuses & Spectrometers

- Collection of Detectors
- Combine data from detectors
- Must do `Decode()` and `Reconstruct()`
- "Spectrometer": Apparatus with support for tracks and standard `Reconstruct()` function
- Must not assume presence of other apparatuses

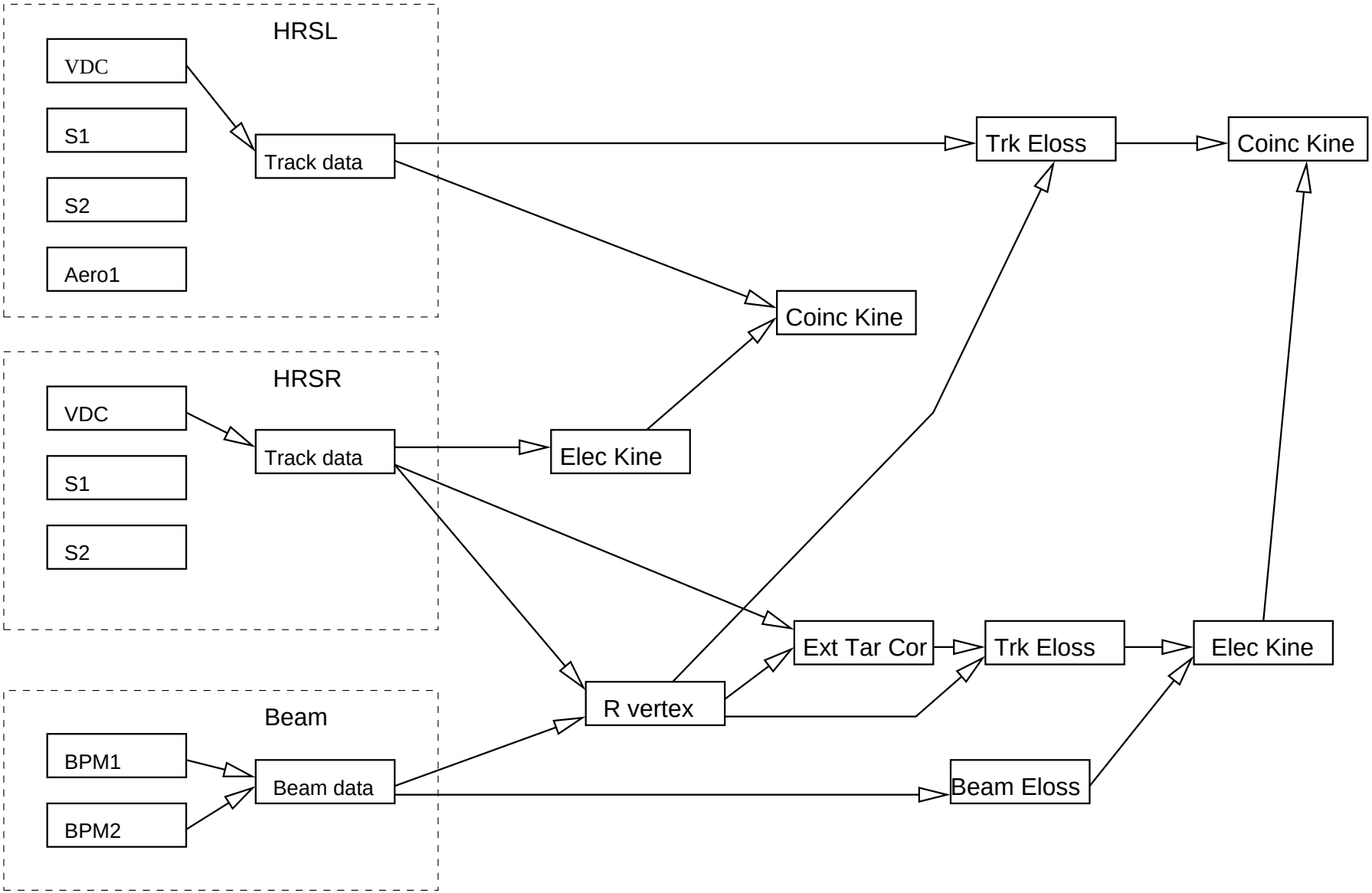
Tracks

- `THaTrack` – reconstructed track with several sets of coordinates, rather spectrometer-centric
- `THaTrackInfo` – generic TRANSPORT track coordinates
- Spectrometers store one or more `THaTrack` objects in a `TClonesArray`
- Multi-track capability inherent in design
- Spectrometers may designate one of their tracks as the "golden track"

Physics Modules

- Combine data from several apparatuses
- Typical applications: kinematics calculations, vertex finding, coincidence time extraction
- Special applications: debugging, event display
- Toolbox design: Modules can be chained, combined, used as needed

Analysis Module Configuration Example



Main Analyzer & Event Loop

- `THaAnalyzer`
- Manages various pieces of information:
  - Analysis objects
  - Output definitions
  - Cut definitions
  - Decoder
  - File names
  - etc.
- Handles initialization of all components
- Implements standard event loop
- May write your own, even as a script

Standard analysis flow is documented here:

`http://hallaweb.jlab.org/root/doc/standard-analyzer.html`

Output Definition Example

```
# -----  Example e12345.odef  -----------------

# Variables to appear in the tree.
variable L.s1.lt[4]
variable L.s1.rt

# The 'block' variables:  All data in Right HRS go to tree.
block R.*

# Formulas can be scalers or vectors.
# Lt4a is a scaler.
formula Lt4a 5.*L.s1.lt[4]

# Cuts can be defined globally and used in histograms.
# Cut C1 is a scaler.  Data is 0 or 1.
cut C1 L.s1.lt[4]>1350

# Histograms can involve formulas, variables, and cuts.
# TH1F, TH1D, TH2F, TH2D supported.
TH1F  rv1n  'L-arm vdc hits on V1'  L.vdc.v1.nhit  10 0 10
```

http://hallaweb.jlab.org/root/doc/output.html

Tests & Cuts Example

```
# -----  Example e12345.cuts  -----------------
Block: RawDecode

evtyp1             g.evtyp==1          // Event type 1 (=HRSR main trigger)
poshel             g.helicity==1
neghel             g.helicity==-1
goodhel            poshel||neghel
RawDecode_master   evtyp1

Block: Decode

NoisyU1            R.vdc.u1.nhit>50
NoisyV1            R.vdc.v1.nhit>50
NoisyU2            R.vdc.u2.nhit>50
NoisyV2            R.vdc.v2.nhit>50
NoisyVDC           NoisyU1||NoisyV1||NoisyU2||NoisyV2
EnoughShowerHits   R.sh.nhit>10
Decode_master      !NoisyVDC
```

http://hallaweb.jlab.org/root/doc/test-guide.html

Test & Cuts Blocks

Blocks of tests evaluated in `THaAnalyzer`:

- RawDecode: After THaEvData::LoadEvent().
- Decode: After Decode().
- CoarseTracking: After CoarseTrack().
- CoarseReconstruct: After CoarseProcess().
- Tracking: After FineTrack().
- Reconstruct: After FineProcess().
- Physics: After Process() of all physics modules, *i.e.* right before writing the output.

Run objects

- CODA files: `THaRun`
- Stores run info (file name, time, number, etc.)
- Stores global run parameters (beam energy, target parameters)

Split Run Example

```
THaRun* r1 = new THaRun( "/data1/e01001_1000.dat.0" );
THaRun* r2 = new THaRun( "/data2/e01001_1000.dat.1" );
analyzer->Process( r1 );
analyzer->Process( r2 );
```

Database

- Currently only flat text files supported
- File names derive from module name:
  `db_R.s1.dat`
- Time-dependence via directory names:
  `DB/20100110/db_R.s1.dat`
- Alternative (finer) time-dependence via time-stamps in file:
  `---[ 2008-02-31 23:59:45 ]`

Example Database File

db_B.mwdc.dat

```
B.mwdc.planeconfig = u1 u1p x1 x1p v1 v1p \
                     u2 x2 v2 \
                     u3 u3p x3 x3p v3 v3p

# "Crate map" for the MWDC. Specifies DAQ module configuration.
# Each rows is:    crate slot_lo slot_hi model# resol   nchan

B.mwdc.cratemap =  3     6       21      1877   500     96  \
                   4     4       11      1877   500     96  \
                   4     17      24      1877   500     96

B.mwdc.search_depth  = 10
B.mwdc.maxslope      = 2.5

B.mwdc.size          = 2.0  0.5   0.0
B.mwdc.x1.size       = 1.4  0.35  0.0
```

Code Example: Database Request

Int_t MWDC::ReadDatabase(const TDatime& date)

```
FILE* file = OpenFile( date );

vector<vector<Int_t> > *cmap = new vector<vector<Int_t> >;
string planeconfig;
Int_t time_cut = 1, pairs_only = 0, mc_data = 0;

DBRequest request[] = {
  { "planeconfig",      &planeconfig,      kString },
  { "cratemap",         cmap,              kIntM,   6 },
  { "timecut",          &time_cut,         kInt,    0, 1 },
  { "pairsonly",        &pairs_only,       kInt,    0, 1 },
  { "MCdata",           &mc_data,          kInt,    0, 1 },
  { 0 }
};

err = LoadDB( file, date, request, fPrefix );
fclose(file);
```

Example Run Database File

db_run.dat

```
-----[ 2008-02-31 23:59:45 ]

ebeam      = 1.2

L.theta = 35.5
R.theta = -101.5
L.pcentral = 0.969301
R.pcentral = 0.473616
```

Example Analysis Script

analyze.C

```
gHaApps->Add( new THaHRS("R", "Right HRS") );
myrun = new THaRun("/data/e12345_6789.dat")
analyzer = new THaAnalyzer;
analyzer->SetOutFile("/work/e12345_6789.root")
analyzer->Process(myrun)
// now inspect results in memory, or quit
```

Requires:

- Database files for HRS detectors
- `output.def` output definition file

Extending the Analyzer

If new modules (detectors, apparatuses, physics modules) are needed:

- Get the SDK:

  http://hallaweb.jlab.org/root/download/podd-SDK-1.2.tar.gz
- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility funrtions provided in core Analyzer (e.g. for database access)
- Create database file(s) for the module(s)

Using the library is now trivial:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH"));
```

Extending the Analyzer

If new modules (detectors, apparatuses, physics modules) are needed:

- Get the SDK:

  http://hallaweb.jlab.org/root/download/podd-SDK-1.2.tar.gz

- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility funrtions provided in core Analyzer (e.g. for database access)
- Create database file(s) for the module(s)

Using the library is now trivial:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH"));
```

Extending the Analyzer

If new modules (detectors, apparatuses, physics modules) are needed:

- Get the SDK:

  http://hallaweb.jlab.org/root/download/podd-SDK-1.2.tar.gz
- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility funrtions provided in core Analyzer (e.g. for database access)
- Create database file(s) for the module(s)

Using the library is now trivial:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH"));
```

Extending the Analyzer

If new modules (detectors, apparatuses, physics modules) are needed:

- Get the SDK:

  http://hallaweb.jlab.org/root/download/podd-SDK-1.2.tar.gz
- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility funrtions provided in core Analyzer (e.g. for database access)
- Create database file(s) for the module(s)

Using the library is now trivial:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH"));
```

Extending the Analyzer

If new modules (detectors, apparatuses, physics modules) are needed:

- Get the SDK:

  http://hallaweb.jlab.org/root/download/podd-SDK-1.2.tar.gz
- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility funrtions provided in core Analyzer (e.g. for database access)
- Create database file(s) for the module(s)

Using the library is now trivial:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH"));
```

Extending the Analyzer

If new modules (detectors, apparatuses, physics modules) are needed:

- Get the SDK:

  http://hallaweb.jlab.org/root/download/podd-SDK-1.2.tar.gz
- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility funrtions provided in core Analyzer (e.g. for database access)
- Create database file(s) for the module(s)

Using the library is now trivial:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH"));
```

Extending the Analyzer

If new modules (detectors, apparatuses, physics modules) are needed:

- Get the SDK:

  http://hallaweb.jlab.org/root/download/podd-SDK-1.2.tar.gz
- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility funrtions provided in core Analyzer (e.g. for database access)
- Create database file(s) for the module(s)

Using the library is now trivial:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH"));
```

Extending the Analyzer

If new modules (detectors, apparatuses, physics modules) are needed:

- Get the SDK:

  http://hallaweb.jlab.org/root/download/podd-SDK-1.2.tar.gz
- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility funrtions provided in core Analyzer (e.g. for database access)
- Create database file(s) for the module(s)

Using the library is now trivial:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH"));
```

Extending the Analyzer

If new modules (detectors, apparatuses, physics modules) are needed:

- Get the SDK:

  http://hallaweb.jlab.org/root/download/podd-SDK-1.2.tar.gz
- Write the code for the module(s) and compile it into a shared library
  - Take advantage of inheritance
  - Use utility funrtions provided in core Analyzer (e.g. for database access)
- Create database file(s) for the module(s)

Using the library is now trivial:

```
gSystem->Load("libKaon.so");
HRSL->AddDetector( new THaRICH("rich", "The Hall A RICH"));
```

Limitations

- ROOT file output often slow
- Multi-threading not easy

Summary

- Podd is mature software, in production use for 6+ years
- Based on successful & widely adopted ROOT framework
- Very flexible design: maintainable & expandable
- Extensive experience & examples available