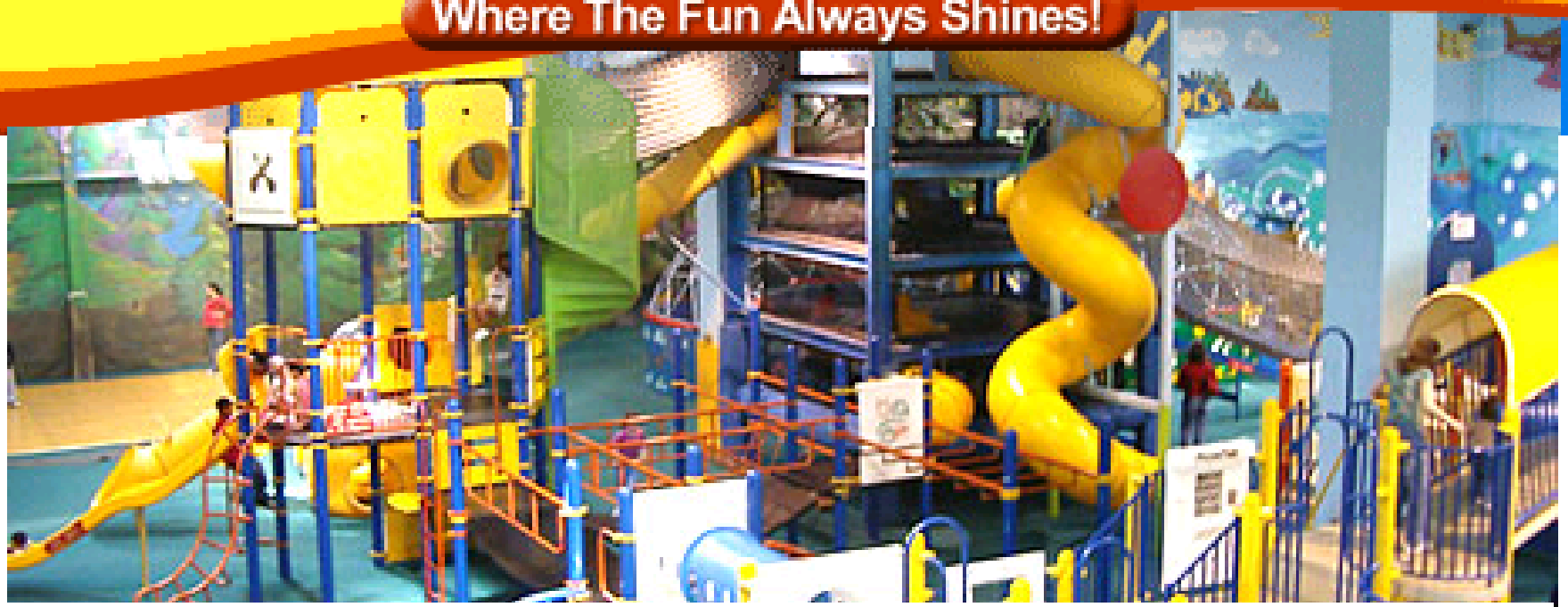
Fun4all

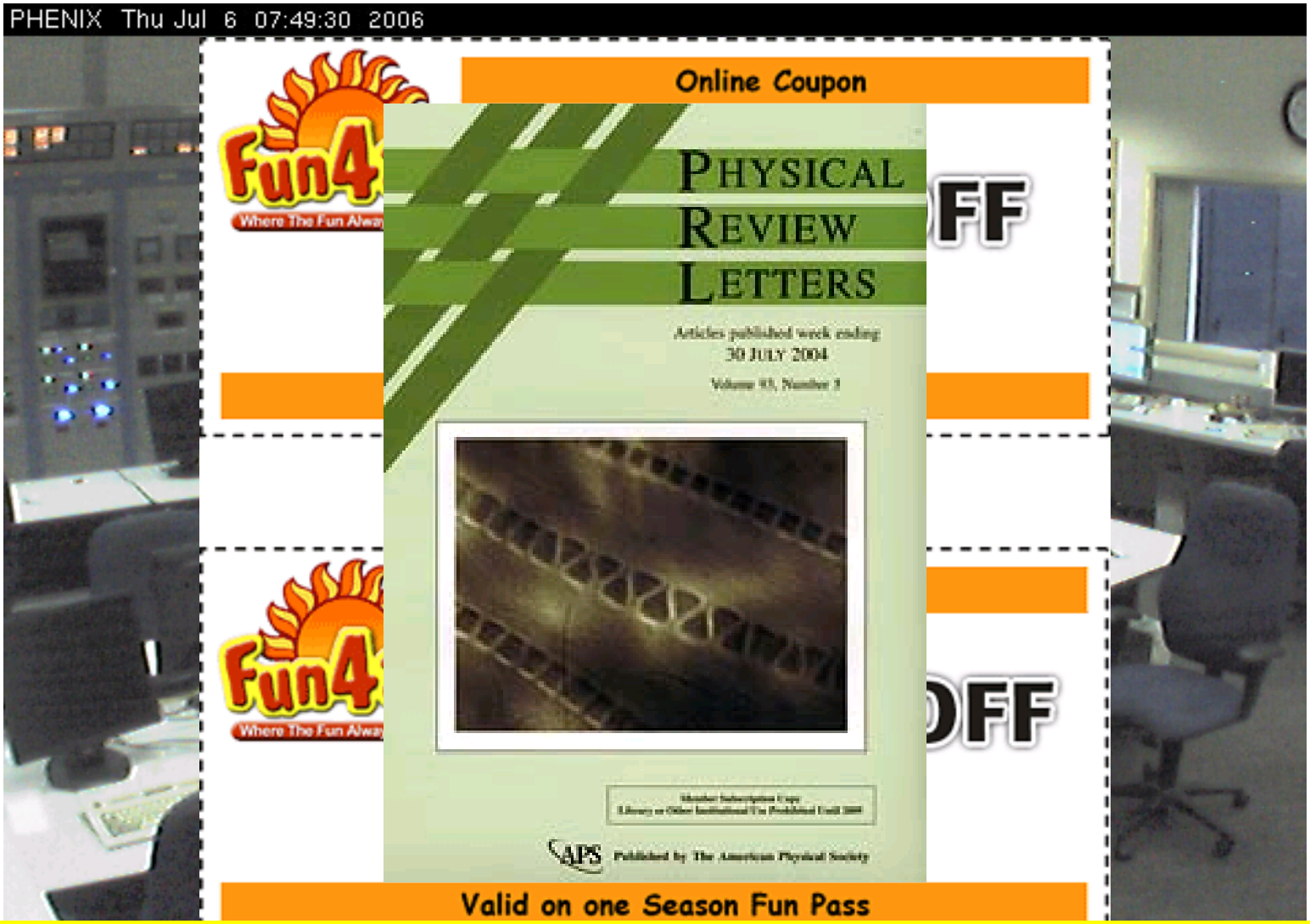
Where The Fun Always Shines!



This “Tutorial” is NOT a C++ introduction

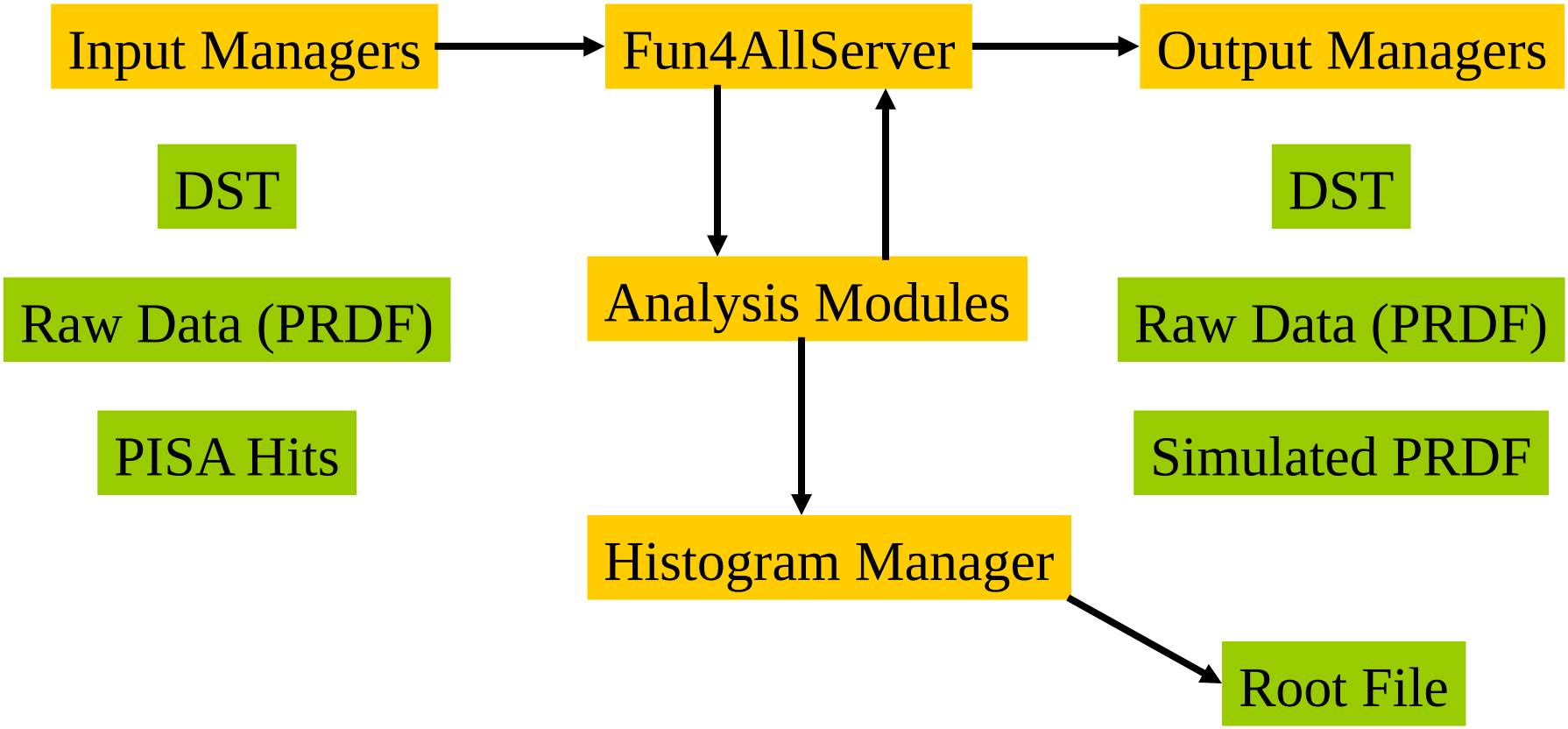
- It should give you an idea how Fun4All works in principle
- Fun4All has been around since Run3 and is constantly evolving (especially between Runs) – this tutorial will only cover the basics
- You don’t need to be a C++ wizard to analyze data
- Have a look at Martins C++ tutorial: http://www.phenix.bnl.gov/~purschke/CppCourse_BNL/
- Speak up now if you don’t know what a base class is, that’s the only thing you need to know here
- Caveat: this is biased towards Central Arm analysis. The other experiment does things differently when it comes to analyzing the data

The basic Flow of a PHENIX Analysis



That does not imply that Fun4all stands between you and PRL, that's the job of the PWG's

Structure of Fun4All



That's all there is to it (as long as you don't look under the hood)

Node Tree

- The Node Tree is at the center of the Phenix software universe (but it’s more or less invisible to you). It’s the way we organize our data.
- We have 3 different Types of Nodes:
- PHCompositeNode: contains other Nodes
- PHDataNode: contains any object
- PHIODataNode: contains objects which can be written out
- PHCompositeNodes and PHIODataNodes can be written to a file and read back
- This file contains root trees, the node structure is reflected by the branch names
- We currently save 2 root trees in each output file, one which contains the eventwise information, one which contains the runwise information
- Input Managers put Data on the node tree, output managers save selected nodes to a file.

Node Tree for real

TOP (PHCompositeNode)/
DCM (PHCompositeNode)/
DST (PHCompositeNode)/
PhHadCglList (PHIODataNode)
EventHeader (PHIODataNode)
Sync (PHIODataNode)
TrigLvl1 (PHIODataNode)
PHGlobal (PHIODataNode)
emcClusterContainer (PHIODataNode)
AccCluster (PHIODataNode)
PHCentralTrack (PHIODataNode)
ReactionPlaneObject (PHIODataNode)
RUN (PHCompositeNode)/
RunHeader (PHIODataNode)
TrigRunLvl1 (PHIODataNode)
TrigRunLvl2 (PHIODataNode)
Flags (PHIODataNode)
DetectorGeometry (PHIODataNode)
PAR (PHCompositeNode)/
PRDF (PHDataNode)

Fun4All prints it out after everything is said and done in the BeginRun(), this is the tree You see when running our analysis train

These Nodes are create by default

These Nodes are “special” – they serve as default for the I/O, the objects under the DST Node are reset after every event to prevent mixing of events. You can select objects under the DST Node for saving, objects under the RUN Node are all saved in the output file

Fun4All can keep multiple node trees and Input/Output Managers can override their default Node where to put the data, but then things get too complicated for this occasion This is only needed for special applications which are not mainstream yet (e.g.embedding).

It’s all about choices - Input

- Fun4AllDstInputManager: Reads DST’s, if reading multiple input files it makes sure all data originates from the same event
- F but d en reading simulated input together with real data for embedding or if you just feel like screwing up)

CAVEAT:
You cannot mix reading of Raw Data and DST/PISA

- Fun4AllPisaInputManager: Reads PISA Hits files
- Raw Data (PRDF) uses still different mechanism

It’s all about choices - Output

- Fun4AllDstOutputManager: Write DST’s
- Fun4AllEventOutputManager: Writes possible)

Caveat:
You can only write Events if the input are Events (PRDF File)

- Fun4AllPrdfOutputManager: Write simulated raw data file.

Your Analysis Module

You need to inherit from the SubsysReco Baseclass (offline/framework/fun4all/SubsysReco.h) which gives the methods which are called by Fun4All. If you don't implement all of them it's perfectly fine (the beauty of base classes)

- In your class you can implement as many additional methods
- As you like, but these are the ones called by Fun4All
  new run is encountered
- Beware of Examples in CVS, all of them are outdated, most of them were already wrong when they were advertised as the ultimate solution for your needs and none of them are kept up to date (but you might get some ideas anyway)
- the Node tree already contains the data from the first event of the new run)
- End(PHCompositeNode *topNode): Last call before we quit

Okay, How do I navigate the Node Tree which is in every argument????

The bad news is that it is up to you to figure out what is inside these objects, there is hardly any documentation and one has to go into the source code to find out what is actually being filled into the variables (e.g. PHCentralTrack and Emc Clusters have different ideas of what “tof” means)

And you don’t even have to know which one of our gazillion versions of PHCentralTrack is used

Blah Blah Blah – Very nice, but how do I run the show?

Fun4All provides only building blocks, you need to put them together yourself in your root macro according to your needs

```
void RunMyAnalysis()
{
gSystem->Load("libmynobelprizewinninganalysis.so");
Fun4AllServer *se = Fun4AllServer::instance();
Fun4AllInputManager *in1 = new Fun4AllDstInputManager("DSTIN1");
Fun4AllInputManager *in2 = new Fun4AllDstInputManager("DSTIN2");
se->registerInputManager(in1);
se->registerInputManager(in2);
in1->AddFile("PWG_MinBias_run4AuAu_Central_200GeV_v01_pro66-0000121548-0003.root");
in2->AddFile("CNT_MinBias_run4AuAu_Central_200GeV_v01_pro66-0000121548-0003.root");
SubsysReco *my1 = new FirstNobelPrize();
se->registerSubsystem( my1);
SubsysReco *my2 = new SecondNobelPrize();
```

Use objects spread over 2 DST's – you need 2 input managers

No path to the input file – our file catalog will find it for you and so protects you against us moving files or using multiple Copies to reduce the load on our servers

In which modules are called – nobody protects you from doing it wrong (user knows best approach)

Master Recalibrator

We changed our strategy how to go about reconstructing our data from "start only when all calibrations are final" (which delayed us for a year) to "that's good enough for tracking, let's save enough information to run the final calibration later during analysis" (which now delays us for a year)

But it's user friendly – all you need to do is add this to your macro:

```
Fun4AllServer *se = Fun4AllServer::instance();
MasterRecalibrator *mr = MasterRecalibrator::instance();
se->registerSubsystem(mr);
```

BEFORE you register your analysis

Histogram OutPut

Root's pathetic Tdirectory handling makes it unlikely that you figure out how to create and save histograms on your own reliably (at least that's my experience). Use the Fun4AllHistoManager for this

```
#include "Fun4AllHistoManager.h"
MyAnalysis::MyAnalysis()
{
}
MyAnalysys::Init(PHCompositeNode *topNode)
{
  myhist1 = new TH1F(....); // myhist1 should be declared in MyAnalysis.h
  HistoManager->registerHistogram(myhist1);
}
```

Caveat:

This is relatively new and I am not really using this myself yet. But for the train it seems to work

This will save your histograms when executing the Fun4AllServer::End() in the file "myhistos.root". If you don't set the filename it will construct a name from the name of your analysis module

Create your own TTree

My biased opinion: creating my own root TTrees is a very painful experience and analyzing them is even worse

Bu to
jus

There will be an example how to do this in
/phenix/WWW/p/draft/pinkenbu/MyOwnTTree

It
cl t
by .
You put this object on the node tree and tell an output manager to write it out.

Now you've got your own root TTree which you can either analyze it like you analyzed your old tree or you can feed it back into Fun4All and use another SubsysReco module to analyze it

So you finally admitted to yourself that writing bugfree code is just beyond your current capability

Welcome to the club, you are in good company. More than half of last weeks prospective passengers - including me - just joined

Here are some short tutorials how to run code checkers:

The ever popular and easy to use and understand valgrind (what more than the line number were your code is doing the dirty deed do you want):
http://www.phenix.bnl.gov/WWW/offline/tutorials/valgrind.html

Insure is more cryptic but it does find problems valgrind cannot pick up:
http://www.phenix.bnl.gov/WWW/offline/tutorials/insure.html

And if your code just bombs out on you, don’t use print statements, use gdb instead which will get you to the problem in no time:
http://www.phenix.bnl.gov/WWW/offline/tutorials/debuggingTips.html

Programming hints

- Stay away from Root, use stl instead. Think of it this way – stl is widely used (by virtually everybody coding in C++). Compared to that root has a tiny user community. We have our share of problems, any root version change turned out to be a nightmare so far.
- Be wary about your input – e.g. you do find negative time of flights (particles emitted by the emc hitting the beam pipe?)
- Do **NOT** use pro builds for your analysis. It is not necessary, they are normally old and just patched up to barely run reconstruction. We put a huge effort to keep our output readable across software versions and that's pretty much all you use from the libraries. Also you want the latest and greatest in terms of e.g. recalibrators for your analysis. Only use pro builds if you really need identical tracking (or better identical bugs), otherwise use the ana build – it's tagged, new and stable.

W er

The and
there rns
out t I put
the s