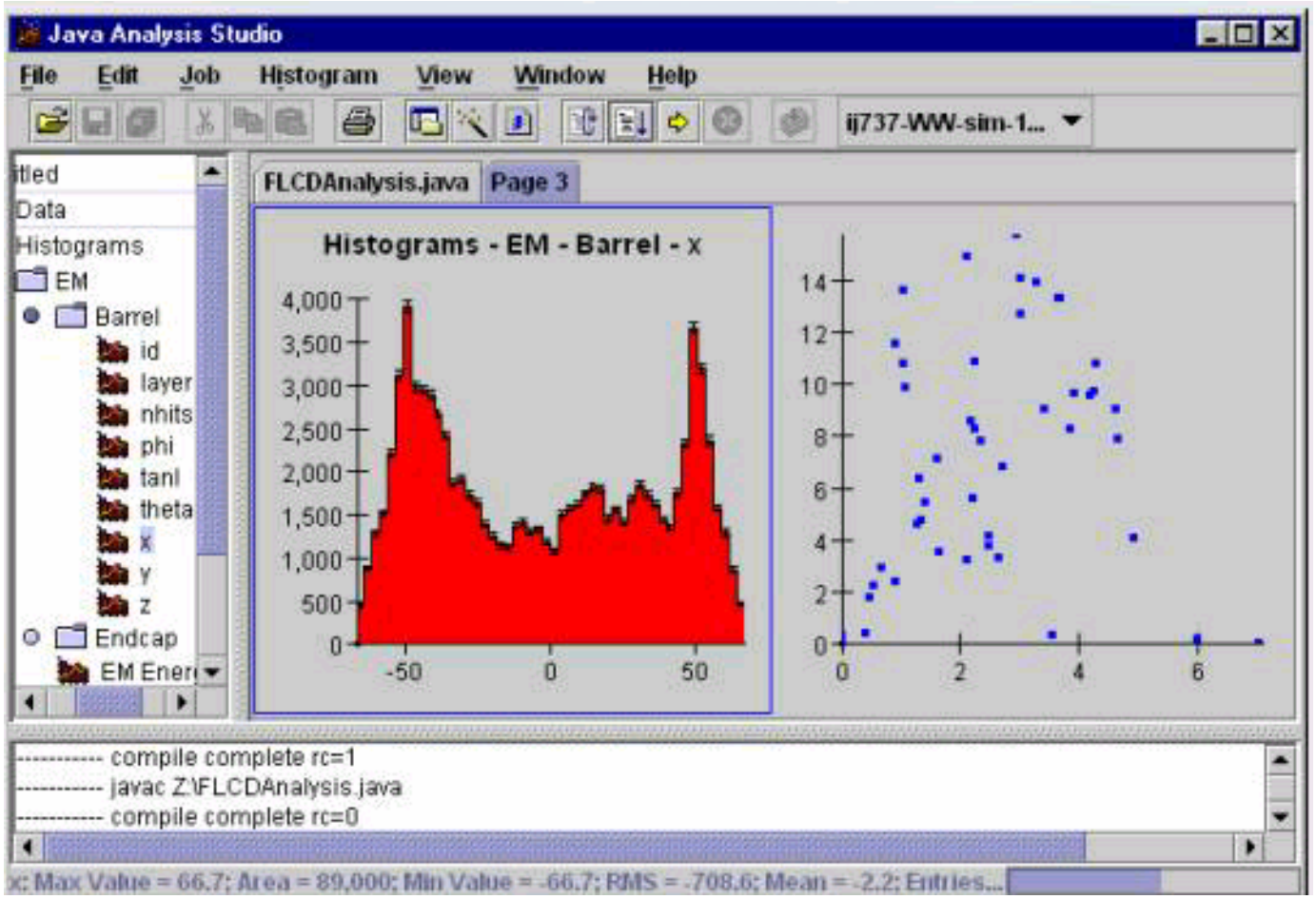
Towards a Nicer User Interface

R.Dubois
D.Flath



The competition : JAS!

User Interface: helper macro & GUI

- Assertion:
  - end users should not have to deal with details of file manipulation, random access to events and some histogram manipulation
- Create helper macro to work in conjunction with 'standard' Event class
  - open, reopen, rewind files
  - process n events from anywhere in the file
  - clear histograms
  - well-defined places to put histogram definitions and event loop code
- Create GUI to give even easier access to helper member functions

MyEvent Helper Macro

R.Dubois
D.Flath

- Started from Root's MakeClass idea
- Goals
  - user only touches a small part of the MyEvent code for histogram definitions and the event loop and analysis
  - much of the code is hidden in the .h file
  - file open, rewind, and random access to events handled by member functions
  - some global histogram manipulation (eg clear)
- Most standard functions held in base class
- MyEvent member functions
  - constructor - opens file
  - destructor cleans up - allows editing of macro and reload
  - Init - open new file
  - Rewind
  - Go(n) - process n events through user analysis
  - StartAtEvent- reset starting point for Go
  - HistDefine - user histograms
  - HistClr - clear all histograms

MyEvent.h

R.Dubois
D.Flath

```
class MyEvent {
  public :
    TFile*      histFile;              // histogram file
    TFile*      f;                     // input file
    TTree       *fTree;                //pointer to the analyzed TTree
    Event*      event;

  MyEvent() {};                        // default ctr
  MyEvent(char* rootFileName); // ctr with root file name
  ~MyEvent();                          // default dtr
  void StartWithEvent(Int_t event);  // start next Go with this event
  void Init(char* rootFileName); // re-init with this root file
  void HClr();                         // Reset() all user histograms
  void AllHistDelete();                // delete all user histograms
  void HistDefine();                   // define user histograms
  void MakeHistList();                 // make list of user histograms
  void Rewind();                       // reset for next Go to start at beginning of file
  void Go(Int_t numEvents=100000); // loop over events

 private:
  Int_t m_StartEvent;                  // starting event
  TObjArray* HistList;                 // list of user histograms
};
```

R.Dubois
D.Flath

Example of Use

```
gROOT->LoadMacro("startmacro.C")                    // load shared libs
gROOT->LoadMacro("MyEvent.C");                      // 'compile' class
MyEvent* m = new MyEvent("MyRootFile.root");        // create MyEvent object

m->Go(500);          // loop over 500 events. Go contains your analysis code
m->Go()                                             // look at remainder of file
...
m->HClr();                                          // clear histograms
m->Init("AnotherRootFile.root");
m->Go(50);
... and so on …
delete m;                                           // prior to reloading macro
gROOT->LoadMacro("MyEvent.C");                      // 'compile' class
… and so on …
```

- It's easy to define macros to hold the gRoot directives.

R.Dubois
D.Flath

User Input: Histogram Definition

```
void MyEvent::HistDefine() {
   // define histograms here

// set up histograms and root file for them here

   histFile = new TFile("Histograms.root","RECREATE");

   TH1F *NLOGS = new TH1F("NLOGS","Num Cal Logs", 100,0,100);
   TH1F *LOGID = new TH1F("LOGID","Cal LogID", 100,0,100);
// end histogram definition _____________________________________

}
```

User Analysis - 1

R.Dubois
D.Flath

```
void MyEvent::Go(Int_t numEvents)
{
// event loop. User analysis goes here. User must refresh pointers to
// histograms.

//     This is the loop skeleton
//       To read only selected branches, Insert statements like:
// fTree->SetBranchStatus("*",0);  // disable all branches
// fTree->SetBranchStatus("branchname",1);  // activate branchname

   printf("\nNumEvents is: %i\n", numEvents);
   if (fTree == 0) return;

   Int_t nentries = fTree->GetEntries();
   printf("\nNum Events in file is: %i\n", nentries);

   Int_t curI;
   Int_t nMax = TMath::Min(numEvents+m_StartEvent,nentries);

   if (m_StartEvent == nentries) {
     printf(" all events in file read\n");
     return;
   }
```

Boilerplate

User Analysis - 2

R.Dubois
D.Flath

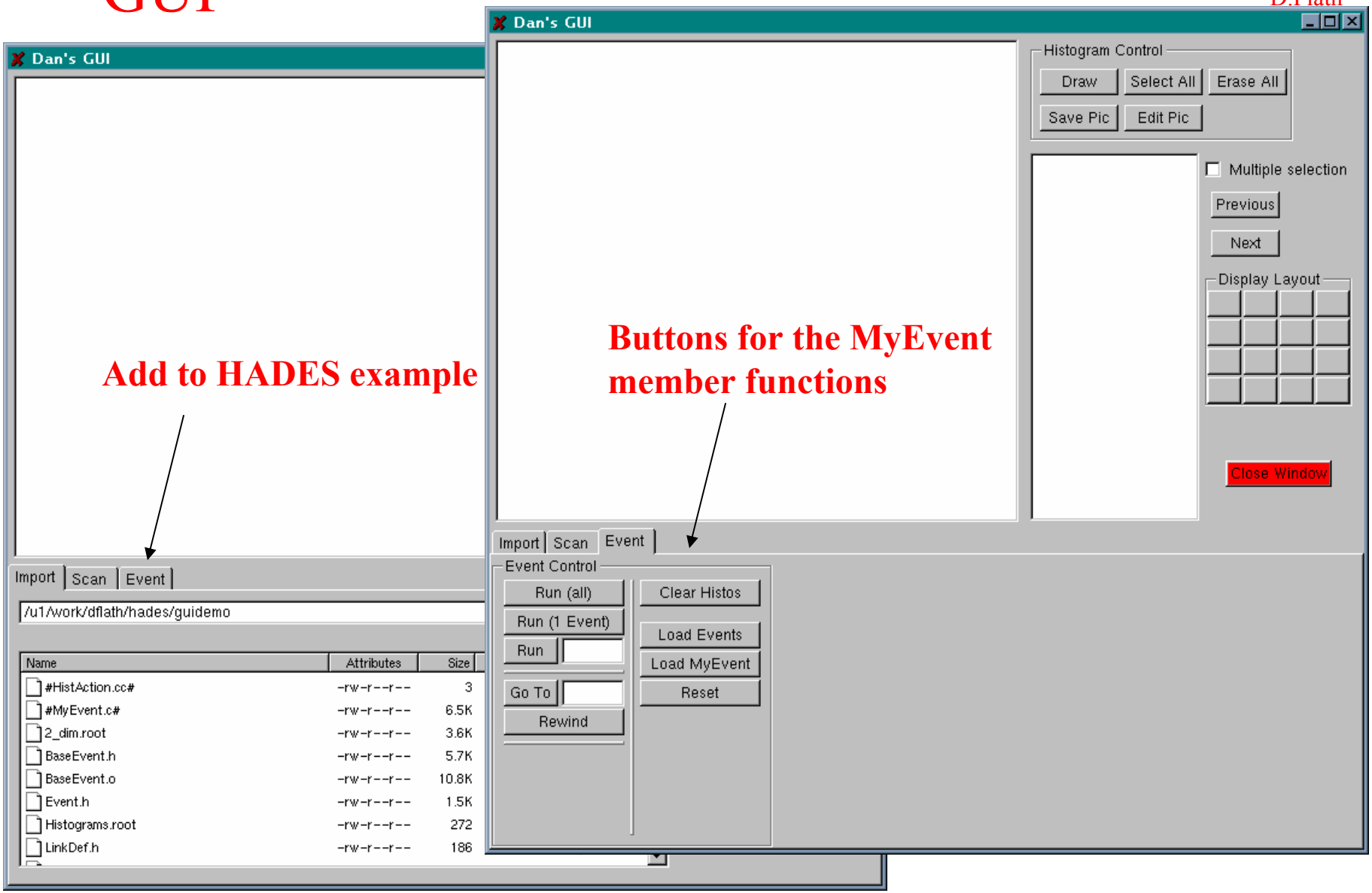
```
// refresh your histogram pointers here _____________________________

   TFile *histFile = (TFile*)gROOT->GetFile("Histograms.root");

   TH1F *NLOGS = (TH1F*)histFile->Get("NLOGS");
   TH1F *LOGID = (TH1F*)histFile->Get("LOGID");

//  end histogram pointer refresh ____________________________________

   Int_t nbytes = 0, nb = 0;
    for (Int_t ievent=m_StartEvent; ievent<nMax; ievent++, curI=ievent) {

                  if (event) event->Clean();
                  nb = fTree->GetEvent(ievent);   nbytes += nb;
                  // start analysis code ____________________________________

                  int nCAL = event->CAL()->GetEntries();
                  NLOGS->Fill(nCAL);
                  for (int ihit=0; ihit < nCAL; ihit++) {

                    CalHit *hit = (CalHit*)event->CAL()->At(ihit);
                    LogID *log = (LogID*)hit->GetLog();
                    LOGID->Fill(log->ID());     }

     }
                  // end analysis code in event loop _____________________

     m_StartEvent = curI;
}
```

Weak point!!

User analysis

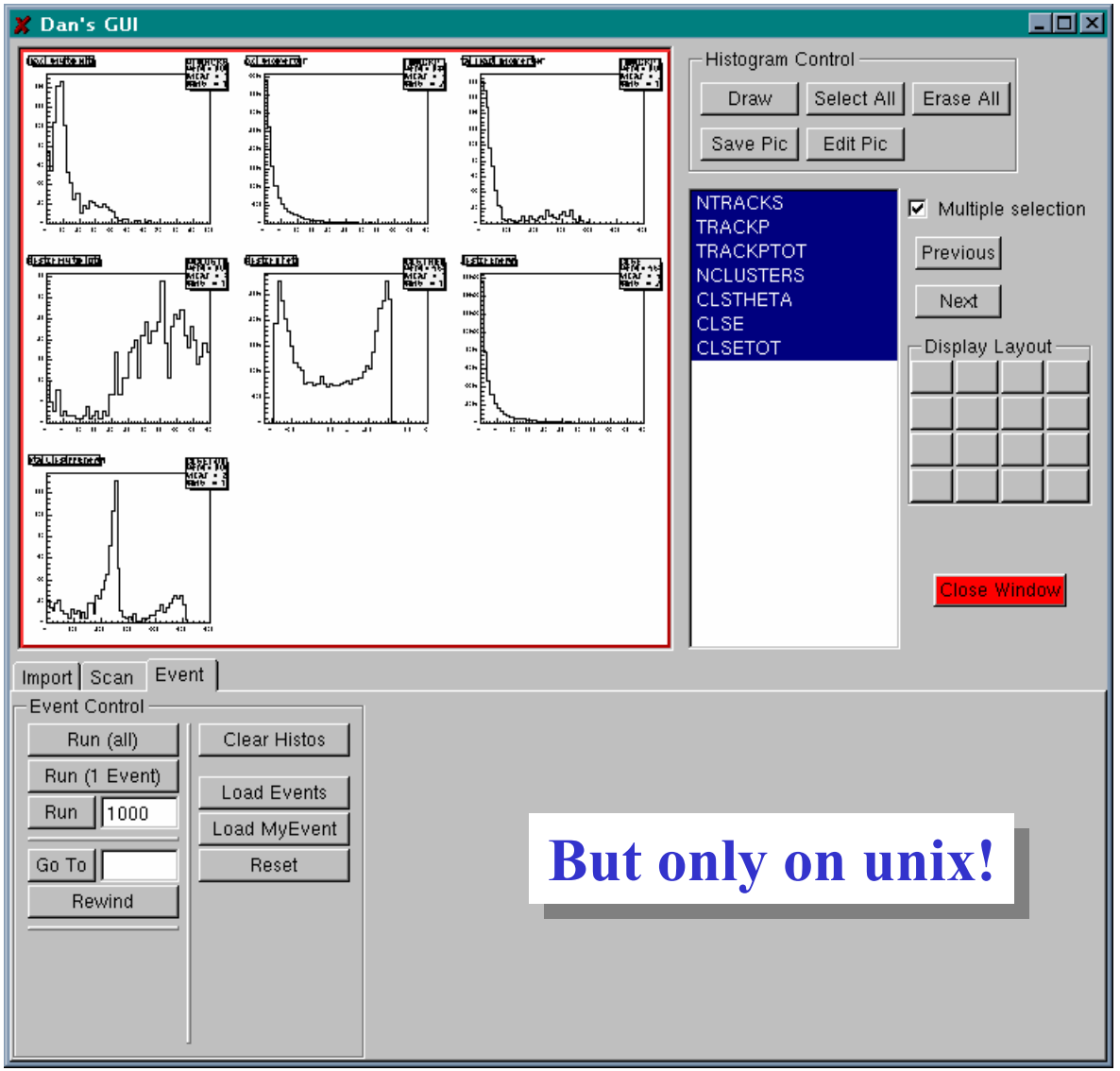
GUI

R.Dubois
D.Flath



GUI - cont'd

R.Dubois
D.Flath



R.Dubois
D.Flath

What's Next? Java GUI?

- V 2.23 now exposes GUI drawing classes for override
- Possible to create a Java GUI framework to solve platform problem
- Classes such as TVirtualX, TVirtualPad, TVirtualHistPainter, etc. need to be subclassed to provide interfaces to Java.
- Then their Java Native Interface counterpart classes must be created.
- Once this is done, people should be able to create a Java GUI that substitutes a "TJavaHistPainter" for the global gHistPainter object, etc. Then have a "TJavaPad" that histograms, etc., get painted to.
- This may not be a good solution
  - lots of work to duplicate **full** Root functionality
  - support nightmare
  - may be OK in short term for just these 3 classes

Java GUI

R.Dubois
D.Flath

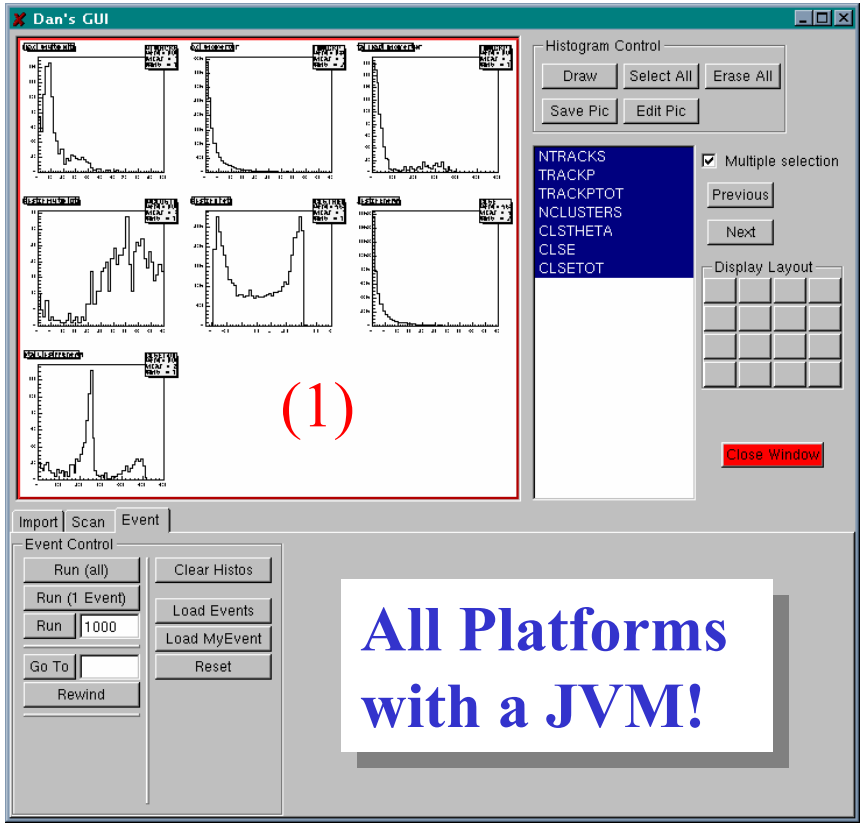
•All GUI widgets are Java (Swing?) components.

•Widget (1) would be an overridden Java canvas which uses JNI to talk to a corresponding ROOT class.

•When GUI events (button clicks, etc.) occur, Java widgets can interact with each other, including the canvas widget.

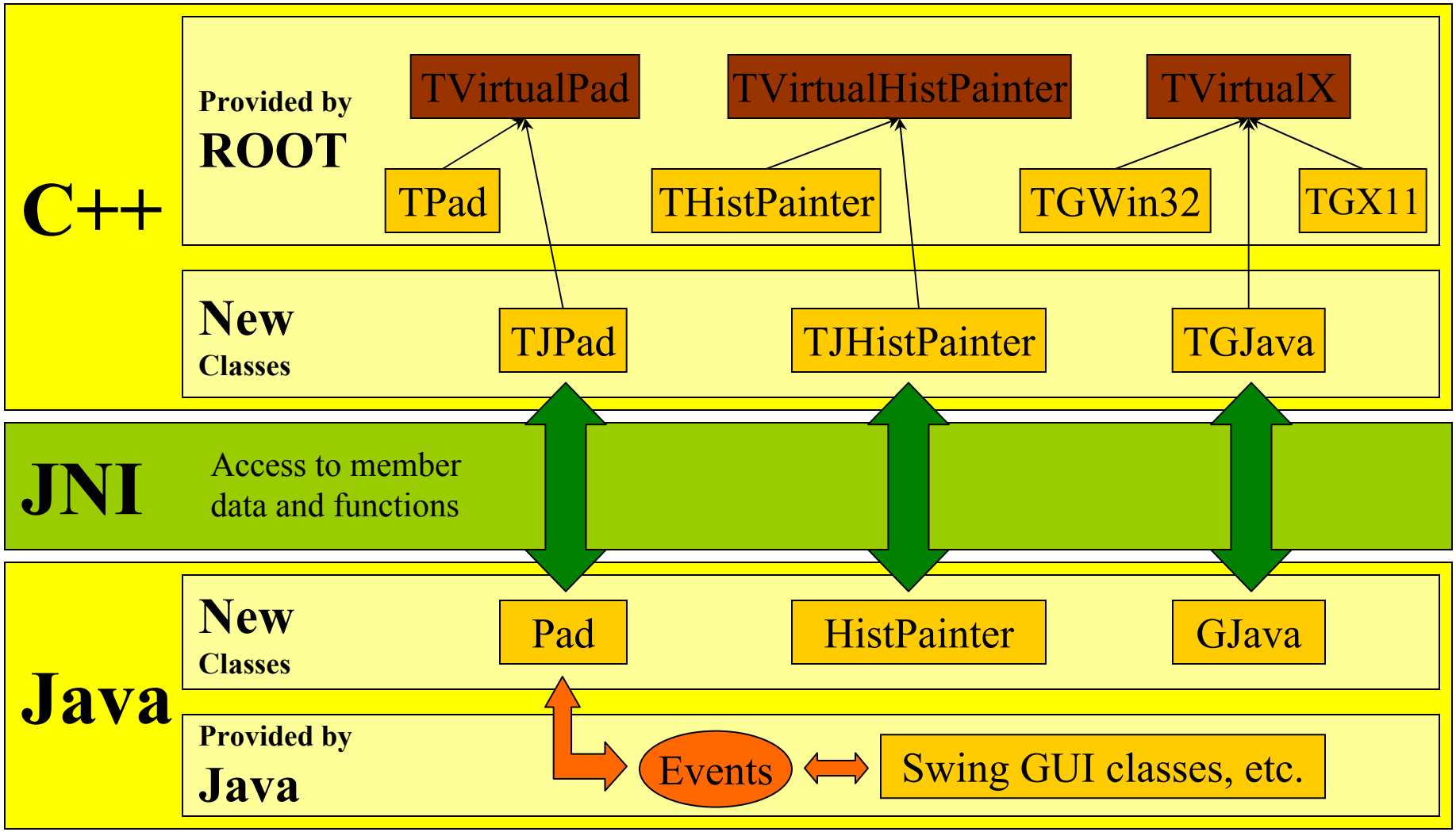
•Other JNI classes could be added to facilitate communication between the Java GUI and ROOT code. For such things as creating, clearing, or updating histograms, or other ROOT objects.

•To reproduce the Hades GUI (without postscript support), three ROOT classes (see next slide) would have to be overridden. Later TVirtualPS, TVirtualTreeViewer, etc., could be overridden as they are needed.



R.Dubois
D.Flath

ROOT - Java interface



Conclusion

- Users need some helper macro to do the mundane work for them
  - we have invented one that suits our needs for now: MyEvent
- GUI makes it even nicer, BUT:
  - Root GUI classes are not maintained equally on NT and Unix
  - investigate a Java interface for the GUI
  - we will talk to the FNAL folks. From initial discussions, it appears more daunting than we thought!