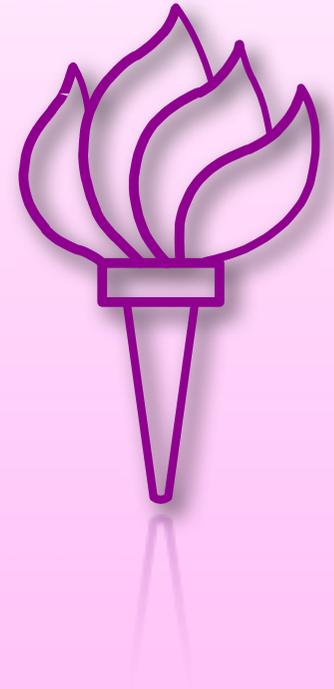


Using SFrame and PROOF

Attila Krasznahorkay Jr.



Overview



- A history of SFrame
- A quick overview of PROOF
- Writing analysis code in the SFrame “framework”
- Running an SFrame analysis



The “SFrame history”

- Around 2006 the CAT physics groups were formed.
- Needed a common framework for ntuple analysis for our group.
 - Accessing input variables
 - Writing histograms/ntuples
 - Providing some sort of configurability for the code
- The CAT SUSY group started working on a project called “SFrame” (as in “SUSY Frame”)
- To save time/effort, many from the CAT Top group joined in using/developing it.

SFrame evolution



- Started out as a thin layer above `TTree::MakeClass()`, with configurability from an XML file
- First major capability: Correct weighting of events
- In the CSC analyses ntuple processing speed became a big issue
 - Nearly everybody was using huge EV ntuples for analysis
 - A “MakeClass analysis” was typically 1-200 Hz in speed
- Streamlined the ntuple reading to achieve 5-10 kHz speeds in the same analyses.

SFrame and PROOF



- Late 2007/early 2008: Port the good ntuple reading speed into the PROOF framework.
- -> Created the SFrame-PROOF code
- Kept the interface of SFrame almost entirely intact -> It was easy to port old code to use PROOF
- Kept the same configurability, weighting of events, etc.
- Recently merged the SFrame-PROOF branch back into the trunk of the code.
- Current code location is [here](#)

SFrame location



The screenshot shows a web browser window with the URL `http://sourceforge.net/projects/sframe/`. The page header includes the SourceForge logo and navigation links like "Find Software", "Develop", "Create Project", "Community", "Site Support", and "About". A search bar is present with the text "enter keyword".

The main content area features the project title "SFrame - A ROOT data analysis framework" by "davidberge, johanneshaller, krasznaa". Below the title are tabs for "Summary", "Files", "Support", and "Develop". The summary text reads: "SFrame is a C++ framework built around the ROOT libraries for analysing (mainly) particle physics data. It gives a very high performance for processing data, by allowing the user to run his/her code on a distributed farm of machines." Below this text are two buttons: "Download Now! Newest release (78.5 KIB)" and "View all files >".

On the right side, there is a "Project Reviews" section with the text "Be the first person to add a review." and a "Would you recommend this project?" section with thumbs up and thumbs down icons. Below that is a "Related Projects" section featuring "NSCL SpecTcl Histogramming system".

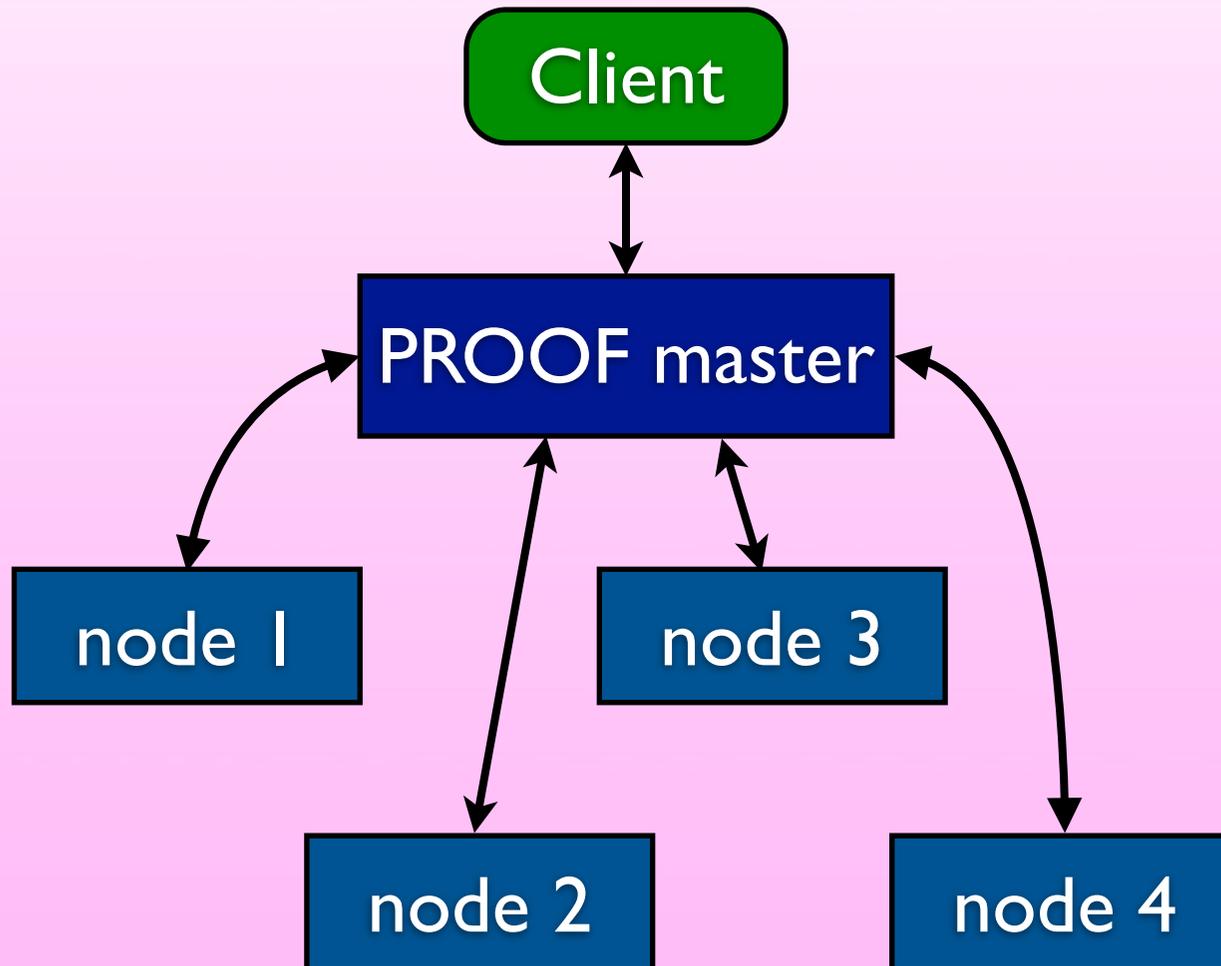
At the bottom of the browser window, a status bar indicates "4 errors occurred in opening the page. For more information, choose Window > Activity."

PROOF basics

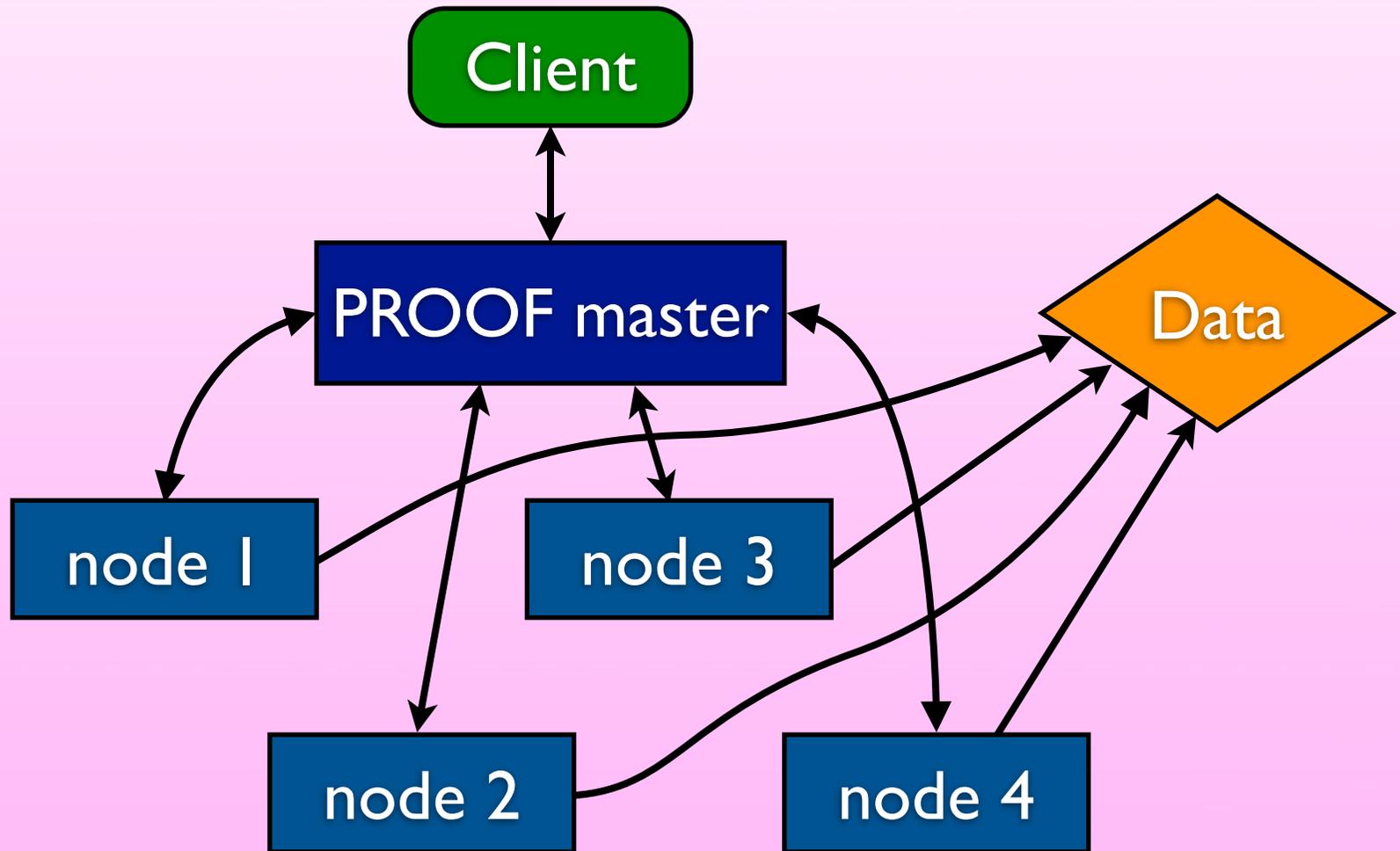


- PROOF is an extension of ROOT for enabling parallel execution of analysis codes on multiple processors
- Very similar in concept as SFrame was -> Develop a class inheriting from a common base class, implementing some virtual functions.
- Instantiates as many of the user's class as many worker nodes are defined, which communicate over the network. -> Possible to create large PROOF clusters
- After all workers are done, the master node collects the output objects (histograms, ntuples, etc.)

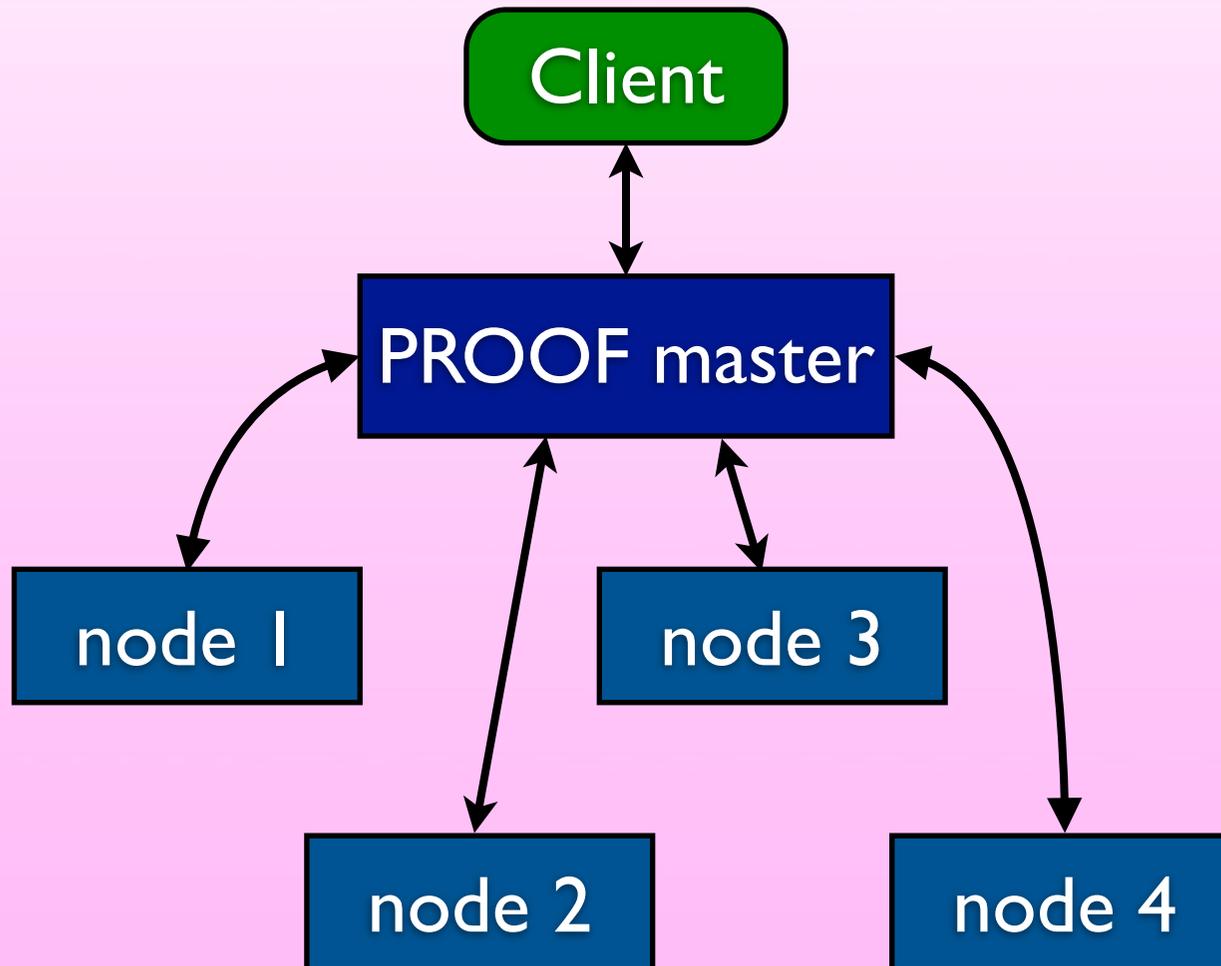
PROOF at work



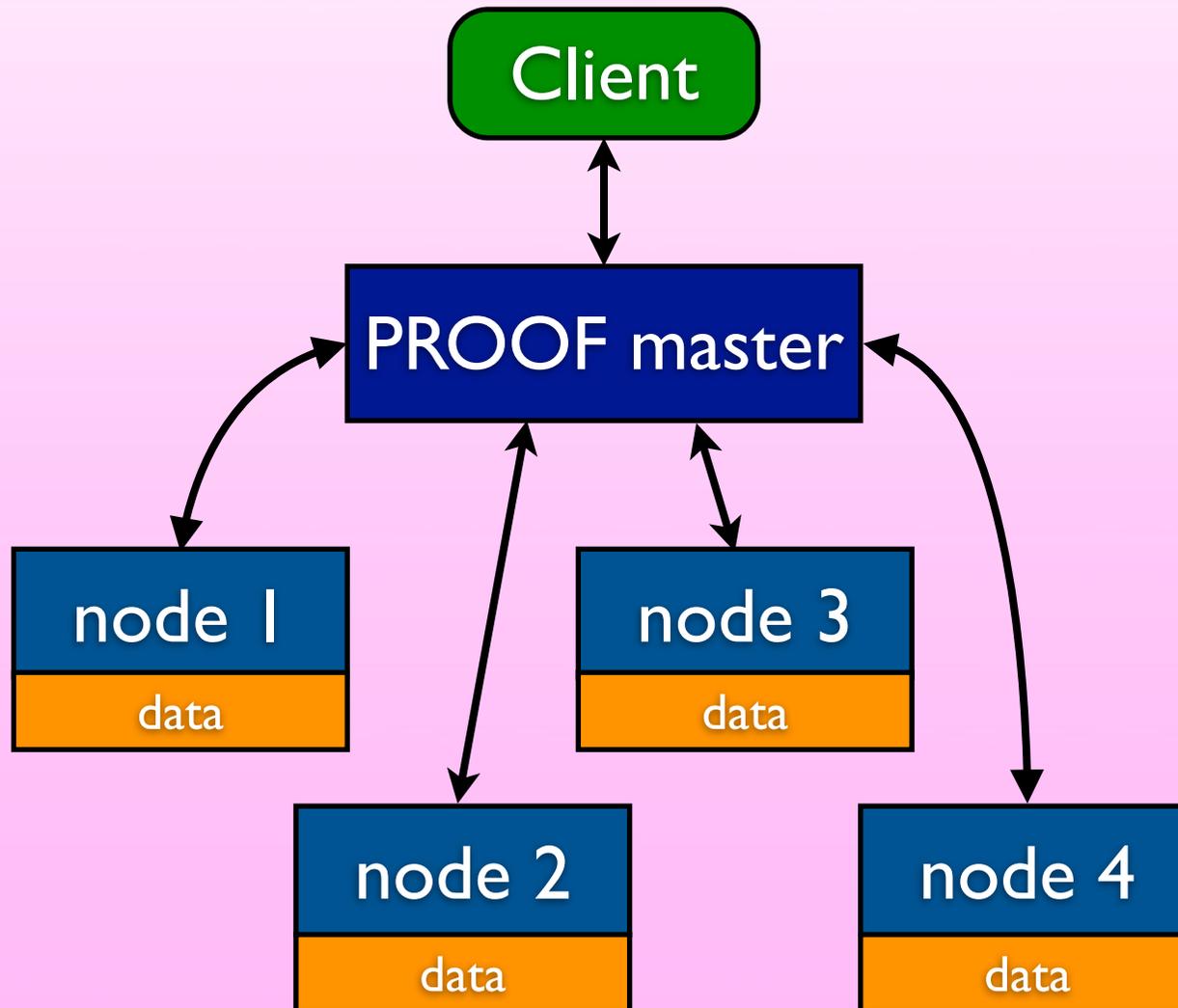
PROOF at work



PROOF at work



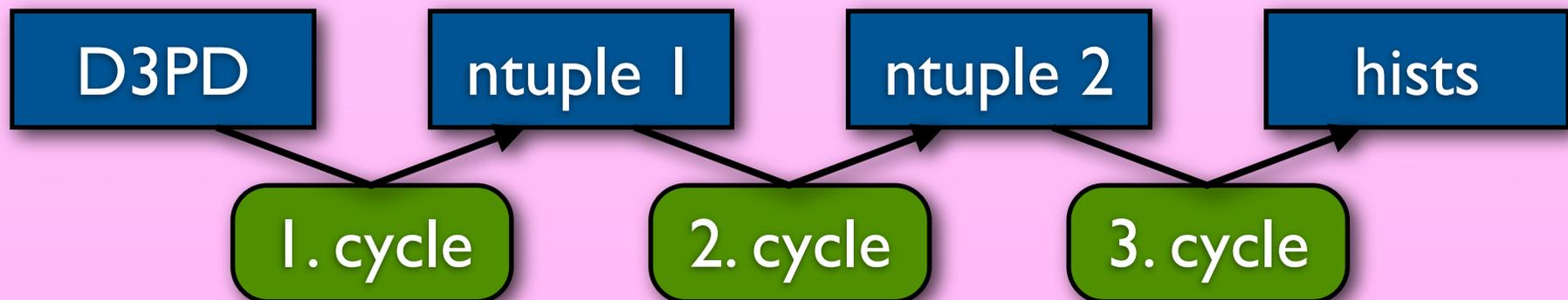
PROOF at work



An SFrame analysis



- Analyses are usually executed in “cycles”. E.g.:
 - 1. cycle: Select the interesting events and write a reduced set of event variables
 - 2. cycle: Calculate some new event-level variables and add them to the events in new output files
 - 3. cycle: Create the final histograms from the previously selected events



An SFrame cycle



```
class FirstCycle : public SCycleBase {

public:
    FirstCycle();
    ~FirstCycle();

    virtual void BeginCycle() throw( SError );
    virtual void EndCycle() throw( SError );
    virtual void BeginInputData( const SInputData& ) throw( SError );
    virtual void EndInputData ( const SInputData& ) throw( SError );
    virtual void BeginMasterInputData( const SInputData& ) throw( SError );
    virtual void EndMasterInputData ( const SInputData& ) throw( SError );
    virtual void BeginInputFile( const SInputData& ) throw( SError );
    virtual void ExecuteEvent( const SInputData&, Double_t weight ) throw( SError );

private:
    ...
};
```

Various common tasks



- Declare a cycle property:

```
DeclareProperty( "D3PDTreeName", m_treeName );
```

- Read a variable from a TTree:

```
ConnectVariable( "d3pd", "el_pt", m_el_pt );
```

- Write a variable to the output file:

```
DeclareVariable( m_el_selected, "el_selected" );
```

- Create a new histogram in the output file:

```
Book( TH1D( "ElHist", "Electron histogram", 100,  
           0.0, 100.0 ), "Hists" );
```

Configuring a cycle



```
<JobConfiguration JobName="TestJob" OutputLevel="DEBUG">

  <Library Name="libGenVector" />
  <Library Name="libSFrameUser" />
  <Package Name="SFrameUser.par" />

  <Cycle Name="FirstCycle" TargetLumi="12.3" RunMode="PROOF" ProofServer="lite"
    ProofWorkDir="" OutputDirectory="." PostFix="" >
    <InputData Type="MC" Version="Zee_1" Lumi="0." NEventsMax="2000" NEventsSkip="3000" >

      <In FileName="MyD3PD.root" Lumi="123.4" />

      <InputTree Name="FullRec0" />
      <InputTree Name="CollectionTree" />
      <OutputTree Name="FirstCycleTree" />

    </InputData>
    <UserConfig>
      <Item Name="D3PDTreeName" Value="FullRec0" />
    </UserConfig>
  </Cycle>

</JobConfiguration>
```

Parts of the configuration



- Libraries and “packages”
 - Load a library needed for the analysis cycle:
`<Library Name="libMyLibrary" />`
 - Compile and load a package on PROOF:
`<Package Name="MyLibrary.par" />`
- Define a cycle:
`<Cycle ...>` and `<UserConfig>` nodes
 - Can change the execution mode (PROOF, PROOF-Lite or no PROOF) with changing some items
- Define the data to run on:
`<InputData ...>` nodes, defining all the important properties of the input files

Running a cycle



```
[bash][agh228]:config > sframe_main FirstCycle_config.xml 2>&1 | tee test.log
( INFO ) SCycleController : Initializing
( INFO ) SCycleController : Deleting all analysis cycle algorithms from memory
( INFO ) SCycleController : read xml file: 'FirstCycle_config.xml'
( INFO ) SCycleController : Created cycle 'FirstCycle'
( INFO ) FirstCycle       : Initializing...
( INFO ) FirstCycle       : Reading SInputData: MC
( INFO ) FirstCycle       : Reading SInputData: MC
( INFO ) SCycleConfig     : =====
( INFO ) SCycleConfig     :                               Cycle configuration
( INFO ) SCycleConfig     : - Running mode: PROOF
( INFO ) SCycleConfig     : - PROOF server: lite
( INFO ) SCycleConfig     : - Target luminosity: 1
( INFO ) SCycleConfig     : - Output directory: ./
( INFO ) SCycleConfig     : - Post-fix:
( INFO ) SInputData        : -----
( INFO ) SInputData        : Type                : MC
( INFO ) SInputData        : Version              : Zee_1
( INFO ) SInputData        : Total luminosity    : 209.8pb-1
( INFO ) SInputData        : NEventsMax          : 2000
( INFO ) SInputData        : NEventsSkip         : 3000
( INFO ) SInputData        : Input SFiles        : '/afs/cern.ch/atlas/maxidisk/d181/SFrame/
StacoTaulp3p_dcachepythiazeeSUSYView_1.AAN.root' (file) | '209.8' (lumi)
( INFO ) SInputData        : Input tree          : 'FullRec0'
( INFO ) SInputData        : Input tree          : 'CollectionTree'
( INFO ) SInputData        : Output tree         : 'FirstCycleTree'
( INFO ) SInputData        : -----
( INFO ) SInputData        : -----
( INFO ) SInputData        : Type                : MC
( INFO ) SInputData        : Version              : Zee_2
( INFO ) SInputData        : Total luminosity    : 209.8pb-1
```

Summary



- SFrame provides a nice interface for developing analysis code locally that can be sent to a PROOF cluster with just modifying a configuration file
- The code is used routinely by a large number of groups by now. (Including NYU of course.)
- Possible future development: Smarter handling of input files/datasets
- The framework seems capable enough to handle the ATLAS data just around the corner.

Further material



- Write-up on SFrame: <http://sourceforge.net/apps/mediawiki/sframe/>
- Configuring a small PROOF cluster: http://sourceforge.net/apps/mediawiki/sframe/index.php?title=SFrame-PROOF#Example_setup_of_a_small_PROOF_cluster