Final Analysis in ROOT using Parallel Processing: PROOF

James Walder
Lancaster University

Why the need for Parallel Analysis?

- Most final analysis will be performed using a simple ROOT-files,
  - I.e. outside of Athena and pool.root formatted data.
  - For example: D3PDs
- Even now these ‘final’ root files are becoming large to run over.
- May be possible to optimise your current analysis:
  - Reduce the number of Branches that are read each event:
    - Only useful if the branches are unused for all events.
  - Pre-apply a selection – i.e. make an event list:
    - Useful if require many loops over a tight sub-selection of events.
    - Still has a large overhead for the initial selection.
    - Additional code required to implement.
- Modify D3PD maker code to produce minimal required output:
  - More difficult if D3PD provided centrally by performance group

What other parallel processing tools are available?

- The GRID:
  - For AOD, DPD-level analysis the Grid is a large resource of ‘Parallel computing’ power.
    - No need to re-design code yourself, the grid tools exist.
  - Even with D3PDs, event selection, i.e. *skimming* can be performed on the grid.
- Additional tools, e.g. SFrame have been developed as a framework and allow PROOF usage.
  - PROOF is not designed for ‘true’ parallel processing, but is very efficient at:
    - Full parallel processing usually not required for HEP applications.
    - Splitting -> processing -> merging events and their resulting outputs.

Why PROOF?

- Written within ROOT - No additional software required.
- Uses the TSelector class, can be generated in a similar way to MakeClass
  - (if you're familiar to root) using TTree MakeSelector method.

- Producing final plots *should* be possible within a rapid development cycle.
  - Need for parallel processing of many data files
- Most computers (even laptops) are now multi-core processor machines,
  - A pity to ignore wasted cpu cycles.

- PROOF can be run in two configurations:
  - server – client setup,
  - or in the **PROOF-lite** configuration requiring no server setup.
    - Second method simplest for laptop usage, etc.
- Same class (and code) can be used in standard (non-parallel) root code.

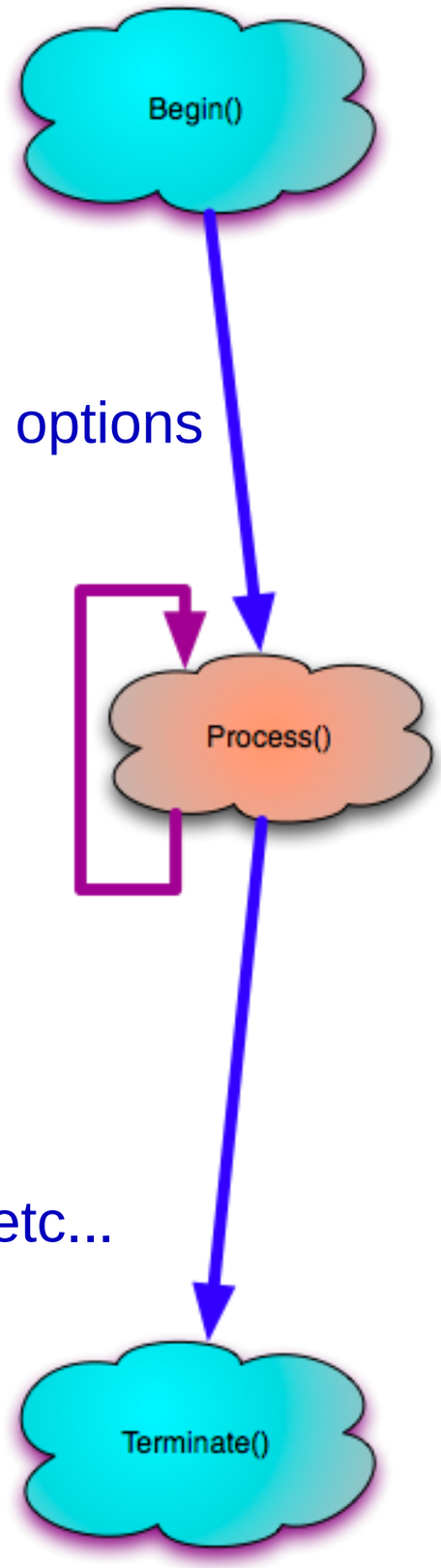
- PROOF used within other experiments,
  - able to be maintained at institutes supporting multiple experiments.

Using PROOF

- First, need to generate a template code which inherits from the TSelector class.
  - Can be created using MakeSelector method:
    - i.e. myTree->MakeSelector("MySelectorClass");
    - Compare with myTree->MakeClass("MyClass");
- This will generate two files:
    - MySelectorClass.h MySelectorClass.cxx
- Modify these files to process your events, Fill histograms, fit curves.
  - (This is the interesting bit)
- Modify the code to work with proof:
  - (This is less interesting, can be mostly copied from other examples)
- To run with proof mode also needs a command script.
  - This copied once, then modified according to needs.

Using TSelector in ROOT

- Simple Structure:
  - Code already created to read the TTree
- Define new histograms and configure according to setup options
- Loop over all events
  - Make selections and fill histograms,
- Perform any necessary fits, etc...
- Finalize and save output,

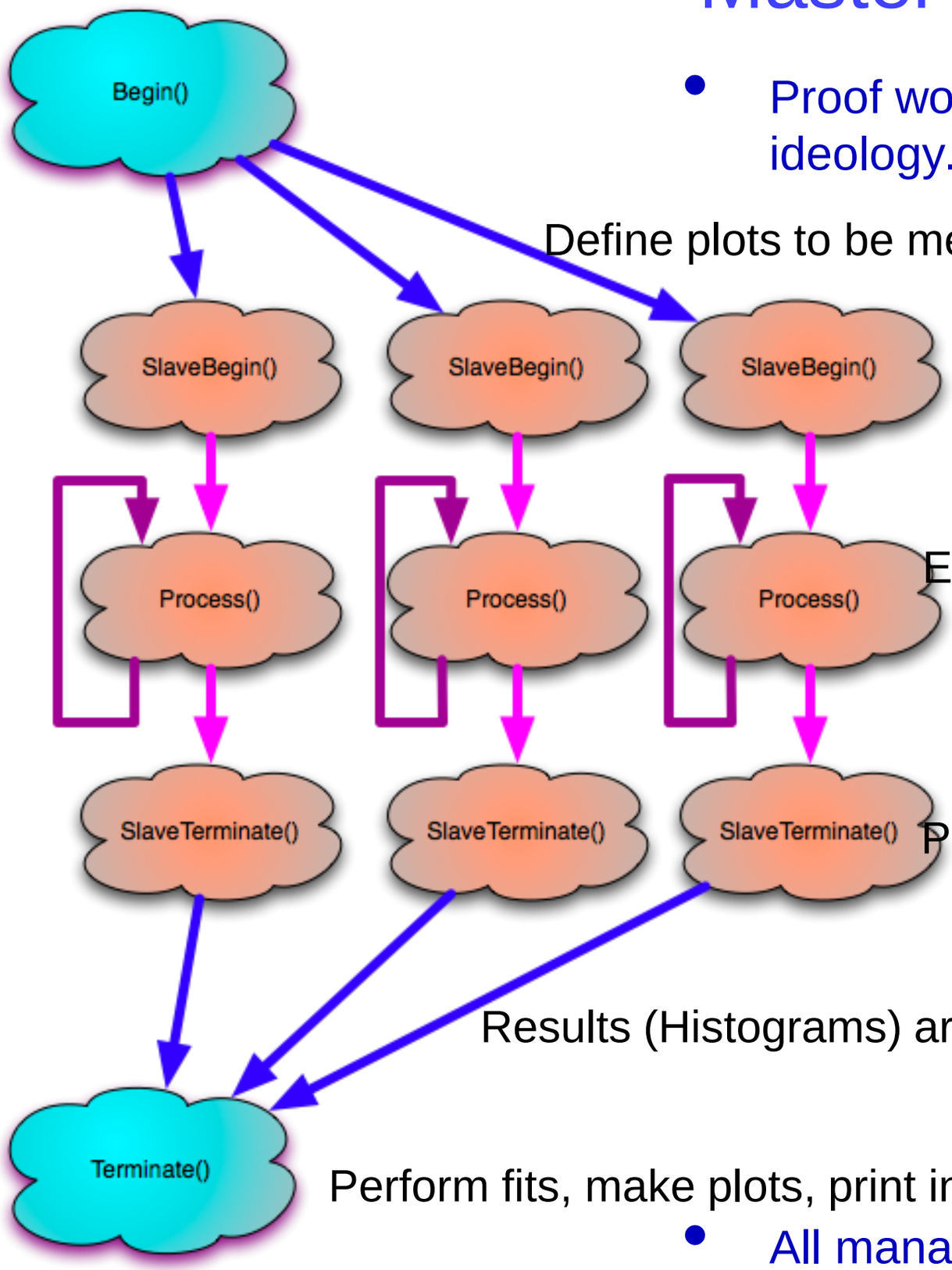


Define histograms, perform other initialisations

Run over all events, Make cuts, Fill Histrograms

Perform fits, make plots, print information

Master – Slaves



- Proof works using One master, many slaves ideology.

Define plots to be merged, and other initialisations

Initialise each slave for processing

Each slave processes a sub-selection of events.
Reads the event, fills histograms etc...

Prepare code for merging, clean up slave clients

Results (Histograms) are merged

Perform fits, make plots, print information

- All managed through the proof code.
- Very little additional code to write yourself

Today's example

- Will see how to create a TSelector-based template file,
  - Run over root files in non-parallel mode.
- Modify the TSelector class to create and fill
  - Histogram output
- Add code to create
  - Slimmed output TTree.
- Run in PROOF parallel mode.
    - Will provide the code that you will need to use to modify the default file.
    - Copy and paste a sample script to 'steer' the processing of the job.
    - Run over some sample data and process the results.
- Caveat:
  - LXPLUS is not the ideal environment for PROOF-Lite analysis
    - If you have ROOT on your laptop, you may want to try the example there.
  - Please shout if you experience any problems - it may not be you !
- With the knowledge from this tutorial, you should be able to perform simple PROOF-based analysis on ROOT-enabled laptops.

Summary

- Parallel processing is already heavily used in the production of data for your analysis.
  - Monte carlo event generation
  - Real data reconstruction
  - Event selection and ntuple production for Athena-level physics analysis
- Tools already exist – GRID - It works, it's powerful, it's managed.
- PROOF may be useful for final analysis – Histogram making, skimming of more complex ntuples.
  - For Analysis requiring simultaneous access to all events, e.g. Maximum likelihood fits, may not be best suited to Proof, although still possible.
- Suggest to write your analysis code to be "PROOF-ready",
  - i.e. using the MakeSelector method to make a TSelector class
  - Can be using in 'normal' non-PROOF mode, or, as time (and data) increases then seamlessly used with PROOF.
- https://twiki.cern.ch/twiki/bin/viewauth/Atlas/RegularComputingProof