Full Simulation status

Andrea Di Simone
INFN Tor Vergata

Outline

- New developments since Perugia
- Already on SVN:
  - Improved Truth configuration
- To be committed:
  - Centralized management of output files
  - Flexible control of UserActions
  - Particle Follower
  - Simplified event structure
  - ROOT input
  - Staged simulation
- Backup: report on summer bg frame production

Truth configuration

- When truth recording was implemented, not clear yet whether the macro-based UI was going to stay, or if we may migrate to some better mechanism
  - No effort at all in integrating truth configuration with macros.
  - Everything was anyway controlled at runtime by means of separated ascii files
- Now, it is clear that we are not going to drop the macros anytime soon
- One user request for macro-level configuration of truth policies
  - Decided to put some effort on this item
- Presently, all configurability formerly provided by the old ascii files is available at macro level
  - Ascii files were removed from svn. Code to read them is still present, though, to provide some level of b/w compatibility.

Truth, reminder

- One can save the status of any secondary particle at its creation
- In addition, full trajectories (i.e. the “path” the particle follows inside the detector) can be saved as well
- Configuration is specified in policies, controlled by specific macro commands
- Main parameter in a policy is the volume name:
  - The policy will affect only secondaries created in that volume (and its daughter volumes)
    - One can declare multiple policies for each volume
- Policies are designed to allow enough flexibility
  - Example:
    - Save all secondaries from my favorite subdetector
    - Save only photons above a given energy
    - Store trajectory of electrons above a given energy
    - Save all secondaries above threshold in some shielding volume and keep trajectory only for those which exit the original volume

Boundaries, reminder

- The aim is to save a snapshot of particles *exiting* any given volume (a subdetector)
  - Approach similar to the one used for MCTruth
  - Configuration done using policies, but with less parameters
- A set of policies for the main subdetectors is provided as default
- User can add his/her own volume at runtime
- Many uses for this kind of feature
  - Particle flux studies
  - detection/reconstruction efficiency measurement

Truth, boundaries: example

- Example commands taken from MCConfig.mac
  - The first line creates a policy for the volume DCH_container, and calls it DCHpol1.
  - From this point on, a new set of commands becomes available at the prompt, i.e. /truth/policy/DCHpol1/*.
    - In general, any policy will have its own menu, where you can easily configure the different properties
- Boundary configuration is similar

```
/truth/create_policy DCHpol1 DCH_container
/truth/policy/DCHpol1/level 2
/truth/policy/DCHpol1/trackPDG 11
ls /truth/policy/DCHpol1
/truth/policy/DCHpol1/print

/truth/create_boundary SVT_L0_container SVT_L0_container
/truth/boundary/SVT_L0_container/level 2
/truth/boundary/SVT_L0_container/trackPDG 0
```

Output files

- Right now, AnalysisManager takes care only of the hit/truth file
  - if a given part of Bruno needs to write a different ROOT file, it has to manage the file by itself
    - Detector survey histograms
    - Bg frames for fast sim
- Implemented a BrunoFileManager class to centrally manage file creation and insertion of TObjects into a given file
  - Only a very basic set of operations are possible for the time being
    - More to be added in the future, depending on the actual needs
    - Detector surveys and bg frames already migrated to the new class
  - Plan is to improve the file manager by migrating some of the functionality now implemented in the AnalysisManager
    - xrootd support
    - hit/truth recording
  - (Positive) side effect would be to simplify the AnalysisManager itself, which has become in time quite complex

User actions

- BrunoActionSteering is the main class controlling our user actions
- G4 will call the *Action methods of the BrunoActionSteering, which will in turn call all our actions
- New possibility is to suspend an action during event processing
  - No UI for this: it is *meant* to be done only on the C++ side.
- Main advantage is performance gain, in particular when dealing with stepping actions
- See next slides for a concrete use case

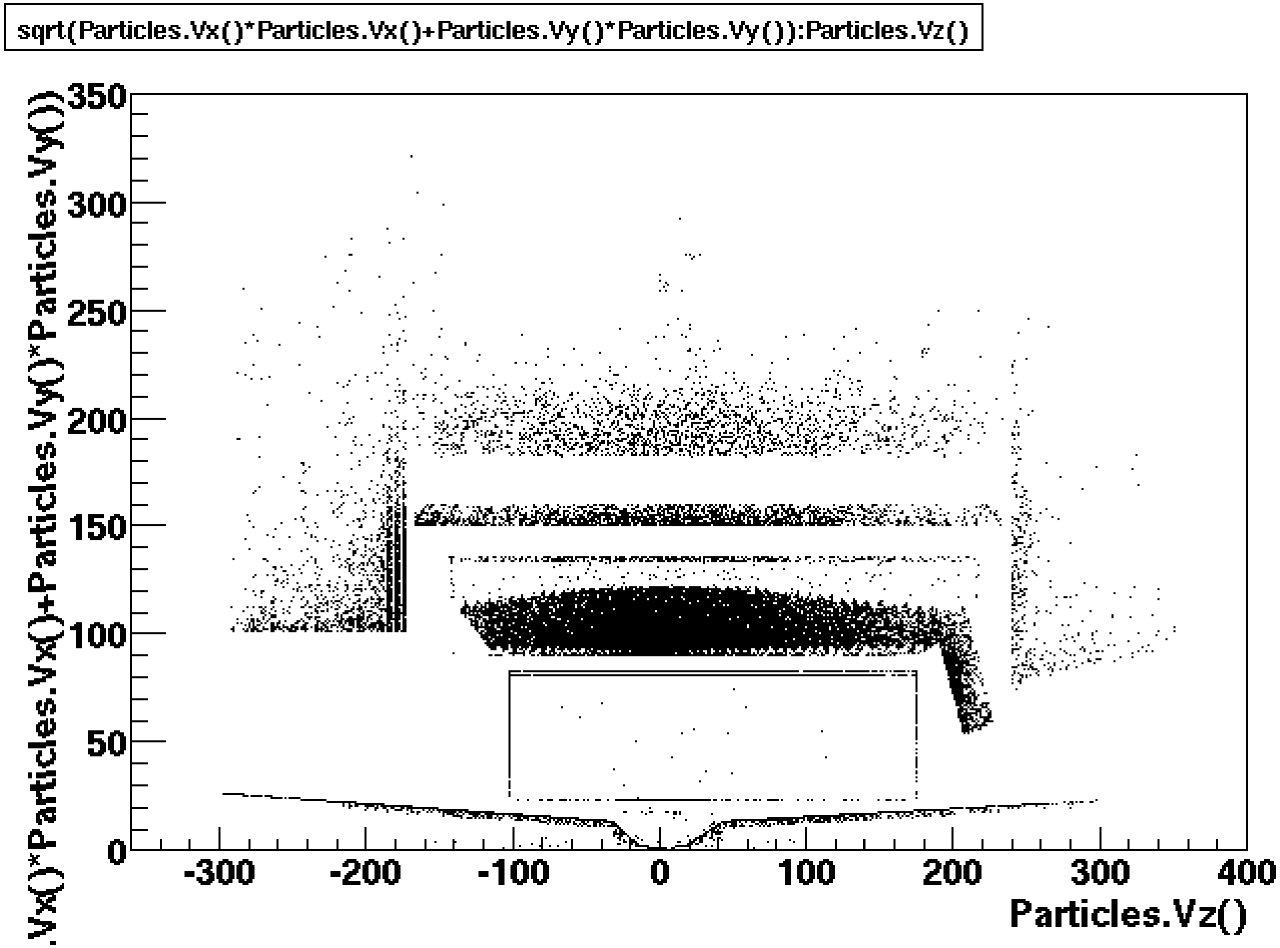
Particle follower

- The idea is to provide means to monitor the evolution of a given type of particle in the detector, adding some truth-level information
- Need a stepping action, which takes care of recording all secondaries produced by the particle of interest
  - In general, it is better to reduce the use of stepping actions as much as possible
- Profit from the fact that G4 always processes one track at a time
  - Register the follower as both a tracking and stepping action
  - @PreUserSteppingAction, check the pdg code of the particle whose stepping is going to happen. Note: this happens once per track.
  - If it is the particle of interest, activate the stepping part of the action
    - @Stepping, record secondaries, energy losses. Note: this happens once per step.
  - @PostUserSteppingAction, de-activate the stepping part of the action
- This way we avoid the overhead of calling the stepping action for non interesting particles
  - Alternative would be to always execute the stepping part, and check there the pdg code of the particle

Particle follower, example

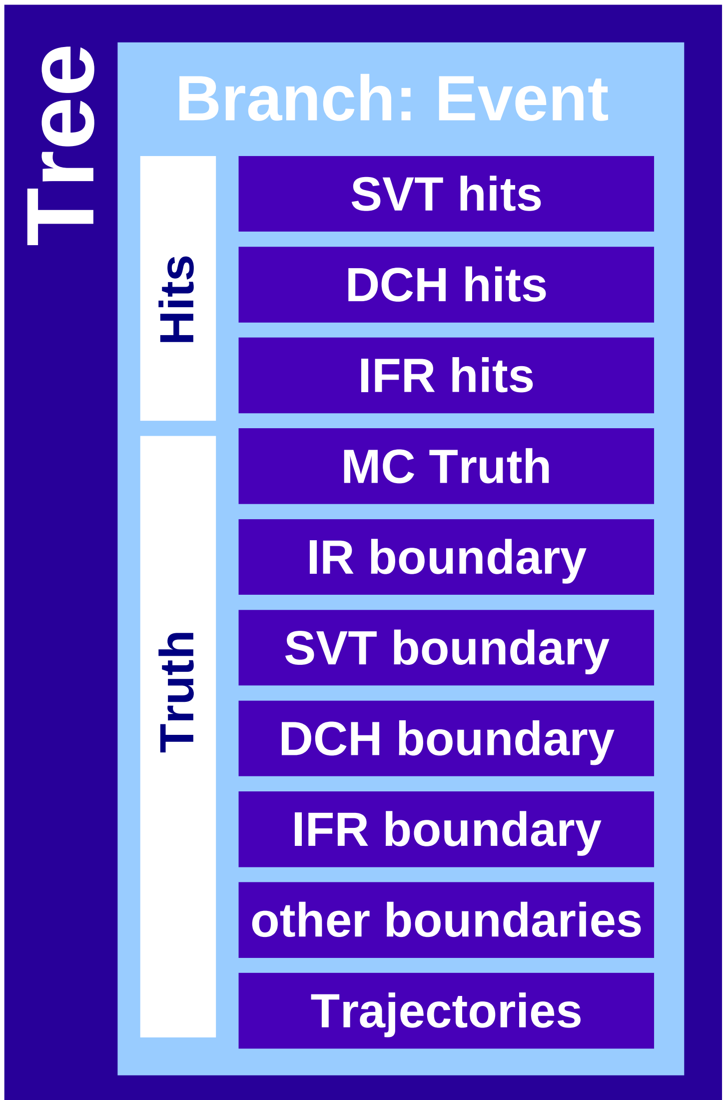
- First application is to improve the handling of neutrons when creating bg-frames for fast simulation
  - fastsim not completely reliable/cpu-efficient when dealing with neutrons
- Idea is to create the bg frame as usual (i.e. saving simulation status at exit of IR), and *in addition* to let G4 simulate neutron propagation
- This is managed by a particle follower:
  - All neutrons are monitored
  - Their secondaries recorded to the same file as the rest of the bg frame
  - Note: in case a secondary neutron is produced, it is followed too

Neutron follower: results



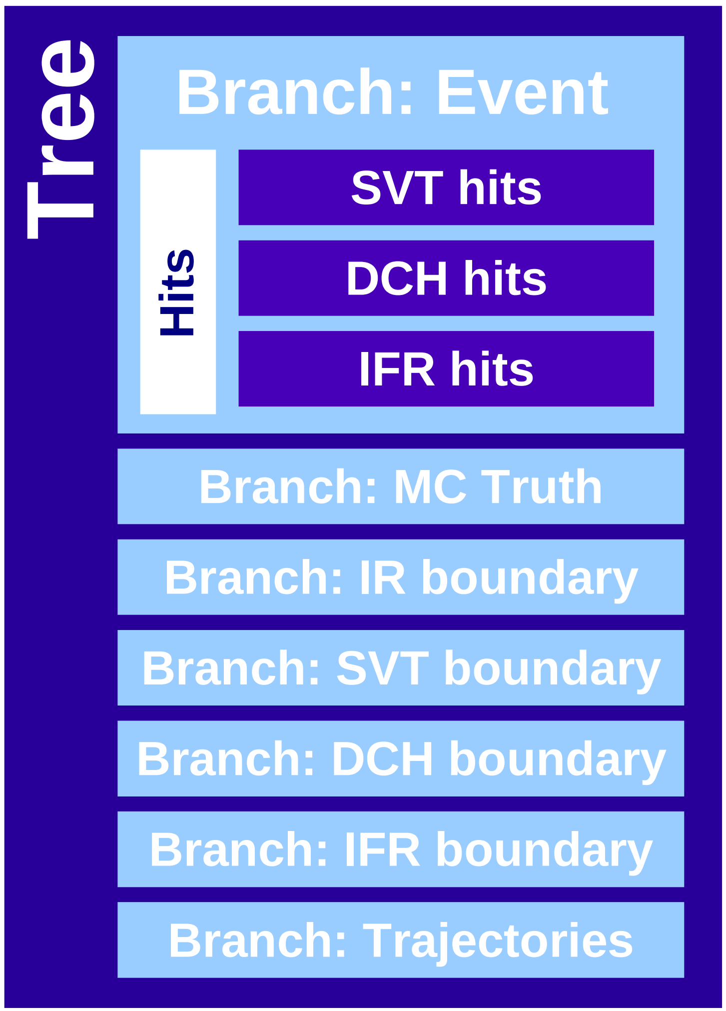
- Neutron interaction vertices, as recorded by the neutron follower
- More details will be given at the bg session

Bruno Event structure



- Simulation results stored in a Tree with one branch. Entries contain instances of a dedicated class, grouping together all hits and MCTruth
- The result of improvements and additions on top of a much simpler original class
- Main disadvantage is lack of flexibility
  - Example: if a user adds his/her own detector boundary, the corresponding fluxes will be saved in the general “other boundaries” collection.
  - In order to have them in a special container (like the default boundaries) he/she will need to
    - Modify the Event class to add the new container.
    - Modify the code responsible for writing of boundary information

New event structure (proposal)



- First simplification could be to separate truth information from detector hits
- Even more, we can save each piece of truth information in separate branches
- This increases modularity
  - E.g: user adding a new detector boundary now does not have to modify any code
  - Just define the new policy in a macro
    - The boundary writer will be clever enough to create the new branch automatically, with a sensible name
- Prototype implemented and working

ROOT input

- Simulation input can be presently one of the following
  - Single particle: run in the same simulation job
  - Beam Strahlung events: run in the same simulation job
  - Ascii file: allows to use results from an external event generator (to be run beforehand as a different process)
- Now external generators can also use a ROOT file for data interchange
- A plain TClonesArray of TParticle, stored as branch in a tree
- BrunoROOTGenerator implemented and tested
- Configurable at runtime via macro file

```
/generator/ROOT/file /path/to/my/file.root
/generator/ROOT/tree NameOfTheTree
/generator/ROOT/branch NameOfTheBranch
```

Staged simulation

- When combining the ROOTGenerator together with the proposed new event structure, one gets *for free* the possibility to perform a *staged* simulation
- Simulate only up to a given point of the detector, e.g. the calorimeter
- In a second phase (i.e. a different simulation job), use boundary information and ROOTGenerator to resume the simulation job from where it was interrupted, e.g. completing simulation in the IFR
- This may result in huge savings of cpu time, in particular when testing different detector geometries
- Also, allows to quickly react to urgent requests:
  - e.g: SVT needs an urgent production.
  - We can simulate events only up to (excluding) the DCH
    - This is FAST
  - If one day DCH is interested in the same events, can resume simulation from where it was interrupted and add its own piece of code
- Functionality was tested using the bg frames produced this summer
  - The plot showing the results from the neutron follower was actually done using this mechanism
    - Re-use bg frames simulated this summer and start a new simulation from there
    - No need to re-simulate interactions in the IR again

Conclusions

- Several developments since Perugia
- Main focus is to improve usability as much as possible, profiting from the feedback we have from users
- New ROOTGenerator allows easy data interchange with external generators
- Tools for production of bg frames for fastsim may actually be very useful also for fullsim studies
  - Particle follower
- Staged simulation is now possible (pending commit on svn...) and it may prove to be very effective in reducing our CPU time usage
  - needs some sensible UI, and copying of hits from one stage to the following one still to be implemented

Summer production

- Goal was to exercise the full chain of bg frame production and overlay for at least one particular background
- Focus on beamstrahlung
  - Easy to run for Bruno, since it uses an embedded generator
  - Easy to treat in overlay, since has unitary weights
- Also used the first prototype of the production system for job submission
- In total, 5000 fullsim jobs were launched, 250 events each
- Simulation status was saved at the exit of the IR, and simulation killed
  - No particle propagation into subdetectors, no hit production
- Diagnostics reported in the following slides was done with a simple python script

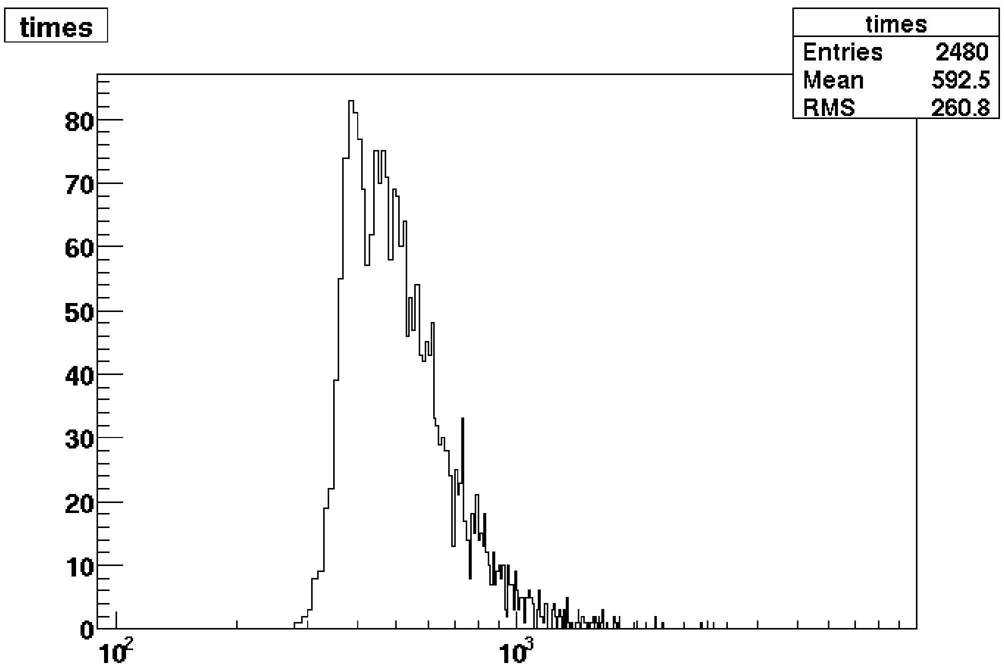
General performance

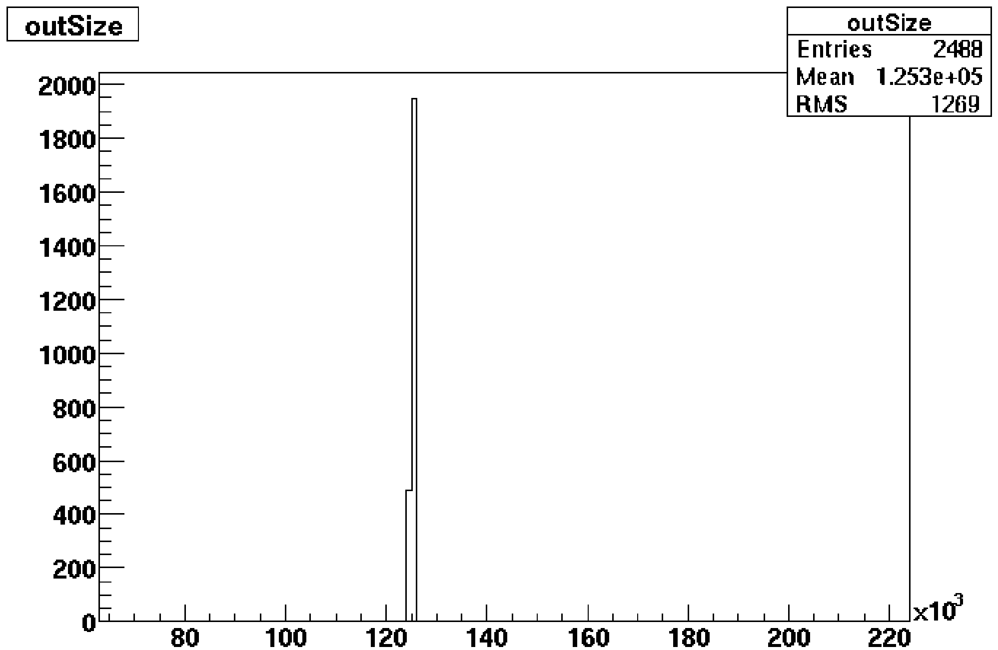
- Log files were available for 4929 jobs
  - no log files means submission problem, not simulation failure
- 4828 log files show the conventional "success" message written to stdout by Bruno
  - this gives a *total* success rate of 4828/4929=97%
- The 101 failures were investigated, and they can be divided in the following categories
  - 45 are related to fs access problems, due to a small bug in the production scripts
  - 11 seem to be due to job killing (either by myself or because of queue time limit, to be checked)
  - 5 are completely mysterious: according to Armando this may be due to general problems of the batch queue
  - 39 jobs failed because of another fs-related problem: my home directory was not accessible anymore
  - 1 job showed a genuine Bruno-related problem
    - the generator produced a huge number of primary particles (N(her)=2000000000).
    - the logs do not show what happened precisely at the job afterwards.
    - for sure it never stopped processing this event. It would be interesting to test the reproducibility of this bug.

General performance (2)

- 2 jobs showed some navigation problems
  - did not lead to job crash, but still it is something most likely related to geometry inconsistencies
    - to be fixed
- Summarizing
  - total success rate (Bruno+production) 97%
  - Bruno-only success rate: ignoring all failures but the one Bruno-related = 4828/4829
  - production-only success rate: ignoring kill problems and the one Bruno failure, normalizing to total submitted jobs = 4828+11+1/5001=96%
- The Bruno-only success rate is of course impressive, but one must consider that we are running on a simplified layout, with only the final focus in, no detector geometry, no hit production, etc.

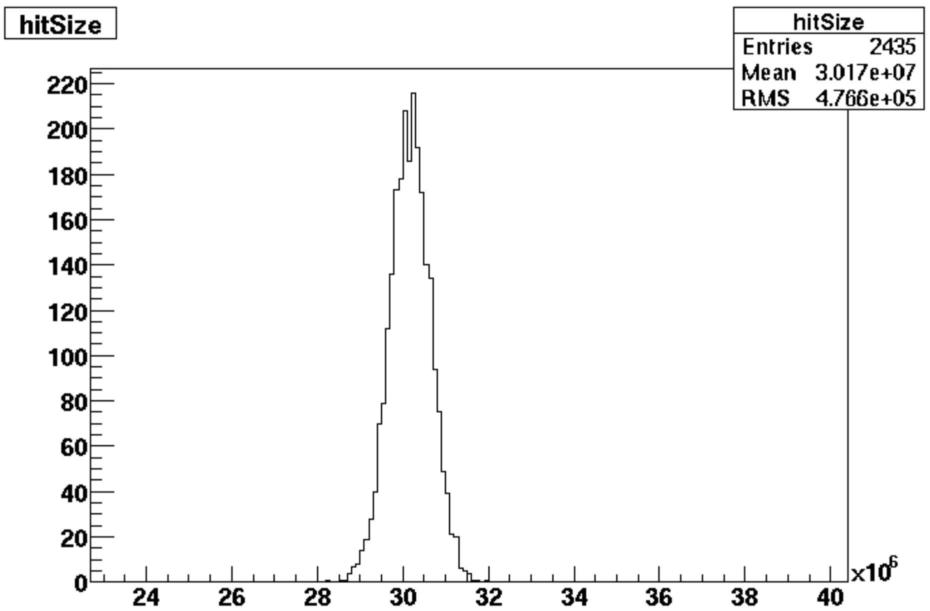
Gallery

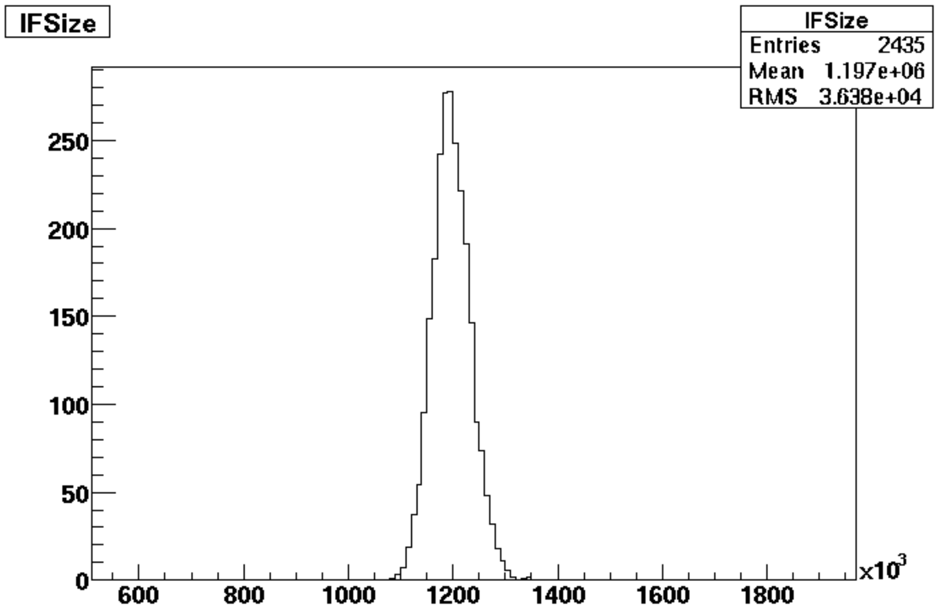




- Left: CPU time distribution for the first batch of 2500 jobs (in minutes)
- Right: size of the stdout log (in bytes)
  - The size of the log files is very useful in diagnosing pathologic jobs, such as the ones with the navigation problems

Gallery





- Left: size of the hit file (in bytes)
  - Reminder: it contains no hits! It's just truth information...
- Right: size of the file with the bg-frame