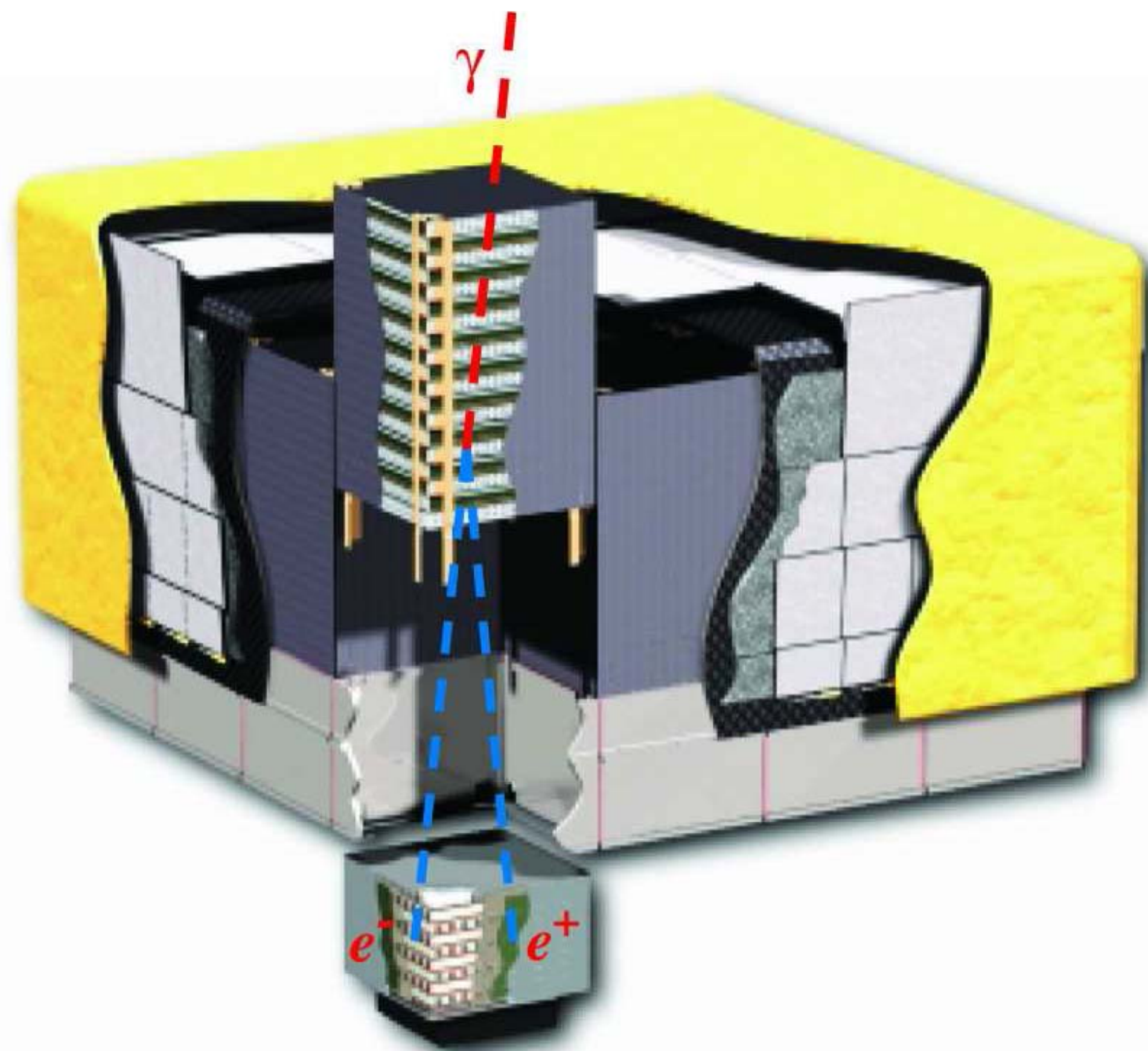


Changing Horses in Midstream: Fermi LAT Offline Computing and SCons



J. R. Bogart (SLAC)

Navid Golpayegani (Wyle IS/NASA GSFC)



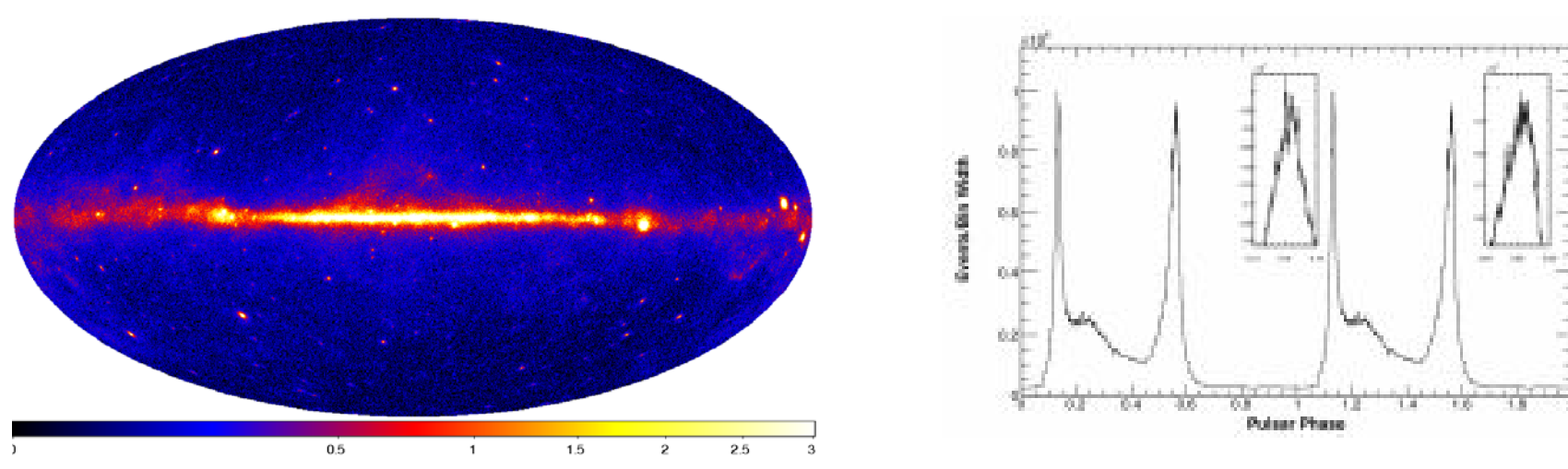
The LAT is essentially a small orbiting HEP detector; the raw data consist of events triggered by interactions with the detector's active elements. The ones of interest after filtering typically correspond to a single photon or cosmic ray particle. Considerable analysis (commonly known as *reconstruction*) is required to identify the best candidate particle and infer trajectory, energy, etc.

Offline Software

The bulk of offline software is organized into two large, partially-overlapping collections: **GlastRelease** and **ScienceTools**. Both are used in automated simulations at SLAC and Lyon CC-IN2P3 and in automated flight data processing at SLAC as well as by individual developers and users, often running on personal laptops and desktops at remote locations.

ScienceTools

Members of the Collaboration use ScienceTools for science analysis of real and simulated data. ScienceTools code is written in a mixture of C++ and python.



Typical outputs: an exposure map and a light curve.

Packages

Both GlastRelease and ScienceTools are organized into *packages*. Build products for a typical package include a library (static or shared), one or more test programs or other applications linked against the library, and program inputs: data files, and configuration files of various kinds. ScienceTools packages also often provide python modules and applications. The package organization facilitates concurrent development by different developers but does not always map well to the dependency hierarchy since an application program in package P may depend on components from more packages than does the library of package P.

Platforms

Much of the initial LAT offline code development was done on Windows. There is still a substantial and critical contribution from Windows developers, particularly for GlastRelease, but the SLAC batch farm machines used for automated processing run Linux. Hence there has always been a requirement to support both operating systems. Our old build system, CMT, would not build our software correctly — or at all — on 64-bit Linux systems, on newer versions of the kernel (anything beyond Redhat Enterprise 4), nor on versions of Visual Studio beyond VS 2003. This has caused consternation among users and developers working on remote systems for some time. More recently, the standard SLAC batch machines were upgraded to Redhat 5, so we can no longer use them to build our software with CMT.

As ScienceTools began to mature sufficiently to be of interest to end-users, it became clear that many preferred to do their analysis on Macs, another platform not supported by the old build system.

Abstract

Several years into GLAST (now Fermi) offline software development it became evident we would need a replacement for our original build system, the Configuration Management Tool (CMT) developed at CERN, in order to support Mac users and to keep pace with newer compilers and operating system versions on our traditional platforms, Linux and Windows. The open source product SCons (**Software Construction Tool**) emerged as the only viable alternative and development began in earnest several months before Fermi's successful launch in June of 2008. Over two years later the conversion is nearing completion.

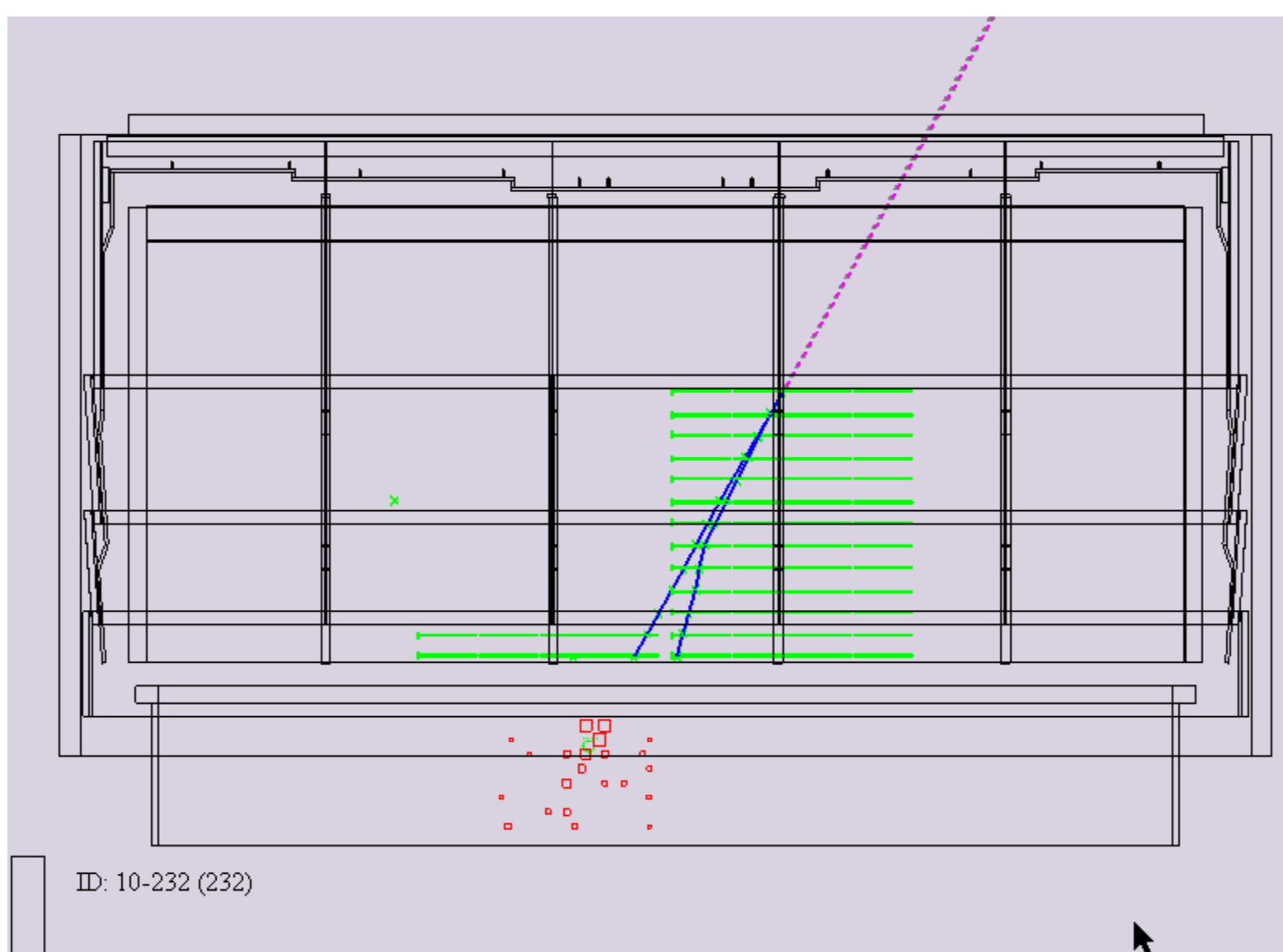
This poster describes the conversion to and our use of SCons, concentrating on the resulting environment for users and developers and how it was achieved. Topics discussed include

- SCons and its interaction with Fermi code organization
- GoGui, a cross-platform gui for Fermi developers
- Issues specific to Windows developer support

GlastRelease

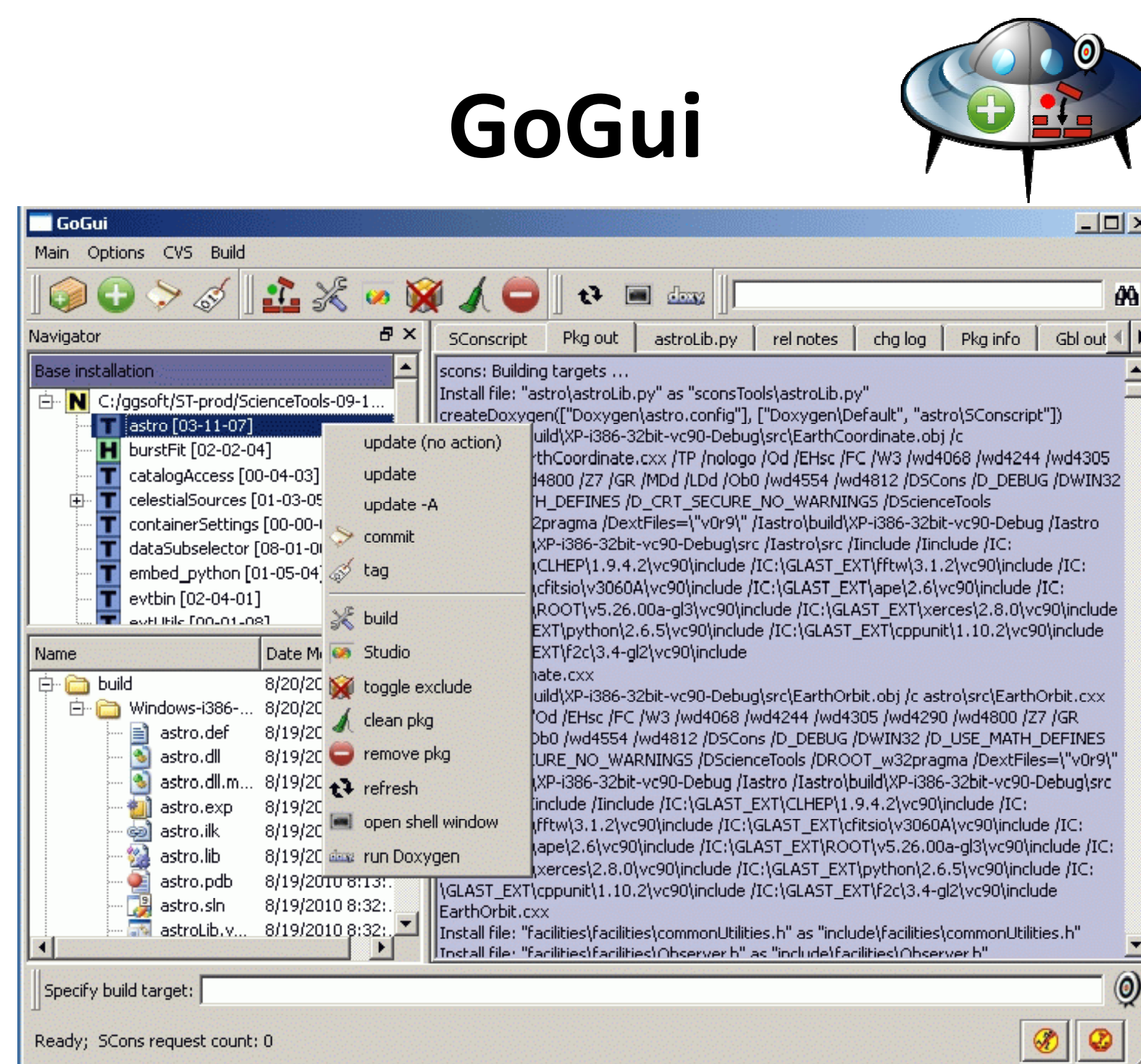
Gleam, the primary build product of GlastRelease, does event reconstruction. Gleam also can simulate events, from astronomical source through digitization in the detector; that is, the input to reconstruction.

Gleam simulations were used years before launch as an aid in determining design parameters for the instrument and developing reconstruction algorithms. It continues to evolve as lessons learned from real data are fed back into simulation and reconstruction. Because of its compute-intensive nature and dependence on external libraries (e.g., Geant for physics simulation) GlastRelease is written almost entirely in C++. The group of active developers is small but crucial to the continued success of the mission.



Simulated Event produced by Gleam: incoming γ converts to a positron and electron. Hits are detected in the tracker; energy is deposited in the calorimeter. Uneven top and side boundaries reproduce actual detector geometry.

GoGui



Issues and Lessons Learned

Conversion of the build system for a mature project is a big deal — no surprise there. In spite of the numerous shortcomings of our old system, we would not have seriously contemplated switching (nor should we have) if not forced by the platform support issue. Particularly sticky were

- **Windows developer support** — We knew it would be difficult, but nevertheless seriously underestimated the time involved. We're still not quite done with GlastRelease support on Windows. Overall, Windows issues have probably added a year to the project. A substantial fraction of that is for developer support.
- **Change to installation strategy** — Installing all files needed at run- or build-time has significant advantages, however it wasn't required as part of the conversion, it added to the work, and it takes some getting used to for developers.
- **Tendency for other upgrades to get dragged in** — New external library versions, new CVS tagging convention and install policy above are examples.
- **Cost of supporting parallel systems** — Most developers and end-users are loathe to switch before they have to. Some minimal communication between systems is necessary but inevitably flaky; it's not worth the time to do a better job.

SCons

SCons supports all platforms of interest to Fermi/LAT Offline. It is written in python as are all local user customization, configuration, etc. This imposes welcome regularity in syntax and behavior and undoubtedly contributes to its impressive extensibility.

Builders and Other Functions

Rules for building a target from its sources are specified in *Builder methods*. SCons comes with several Builders (e.g. for compilation, making libraries, making Java archives, making tar archives, installing files, etc.). SCons also provides several other functions, including **AddOption** (add command-line options) and **Alias** (define new targets in terms of existing ones, similar to phony targets for Make). It is straightforward to extend SCons with custom builders and functions.

Construction Environments

SCons facilitates fine-grained control of dependencies and other aspects of the build process through *construction environments*. One can, for example, independently manipulate compile or link options for each construction environment.

Use and Customization

Standard compiler options and other generic settings are added to an initial environment, baseEnv. Most packages clone baseEnv at least twice — once for an environment to build its library, again for building applications — and customize the clones as appropriate.

Our local extensions include

- Added command-line options (e.g. specify non-default compiler.)
- Added Builders (for Doxygen, dynamic ROOT libraries, ...) and Tools

As a matter of policy (and a switch from previous build system) we install everything needed at compile time or run-time.

Environment set-up

SCons, particularly when combined with our decision to install all files needed at run-time in centralized locations, allowed us to streamline the distribution for remote users (easy to exclude intermediate build products and, for end-user distribution, package source not needed at run time) and simplify set-up, especially for end-users. End-users of an SCons build can establish a process with environment suitable for running all applications by running a single set-up file; they do not need a local installation of SCons.

Windows Project and Solution Files

The largest stumbling block in adapting SCons to make Windows builds has been the creation of adequate project and solution files. SCons native builders produce files which cannot be usefully modified in the Visual Studio environment. Generated files include an encoded version of dependencies and procedures, then invoke SCons to interpret it. The (substantial) advantage of this approach is its robustness: the build proceeds identically whether invoked from Visual Studio or directly from SCons. The disadvantage is that it is impossible to change anything about how the build proceeds from Visual Studio.

The fundamental differences between Windows and Linux make it impossible for a build system to do "the same thing" with both. In our use of CMT, the first step was to generate project and solution files, then use them for the build. Hence automated builds made by the Release Manager (see P064) and builds made interactively by developers were created the same way, but there was a continual maintenance problem in keeping Windows builds similar to Linux builds of the same code base.

SCons does a better job of automating similarity of behavior on Windows and Linux by invoking compilers, linker, etc. directly on both, but at the cost of producing unidiomatic project and solution files. Our approach has been for the Release Manager to produce Windows builds in the standard SCons manner (no VS) and to use additional build targets to generate custom VS project files for our developers. However, capturing all the information of SCons builds and translating, bit by bit, for VS consumption is labor-intensive, unlikely to ever be perfect, and difficult to maintain.

Accompanying Tools

Each of the tools in the large suite developed in support of CMT had to be adapted or entirely rewritten for SCons, including the Release Manager (see P064), Installers (install RM builds remotely) and a new developer gui, GoGui.

The gui, written from scratch in C++ and based on the Qt library, runs on all supported platforms. Among its functions are:

- Access to repository (CVS) for checkout, update, commit, tagging
- Package-centric view but also shows full file hierarchy.
- Build targets
- Run programs
- Debug programs (gdb on Linux; VS on Windows)
- Browse or edit any text file in code base being developed



See other Fermi posters:
 • P064 – Tom Stephens – Release Manager
 • P100 – Richard Dubois – Fermi LAT Computing
 • P101 – Heather Kelly – Maintaining External Code

jr@slac.stanford.edu