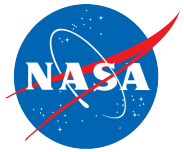


InSAR Scientific Computing Environment

Eric Gurrola, JPL, Giangi Sacco, JPL
Paul A. Rosen, JPL (PI), Howard Zebker, SU

Consulting Collaborators: Mark Simons, Caltech,
Scott Hensley, JPL, David Sandwell, SIO

Students from Stanford: Piyush Shanker and
Cody Wortham

AIST-08-0023
ESTF 2010

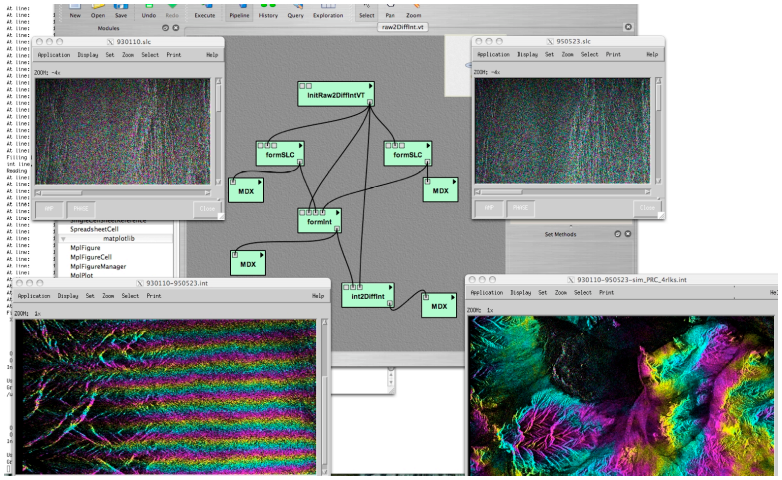
ESTO
Earth Science Technology Office

InSAR Scientific Computing Environment

Objective

- Develop an open-source, modular, extensible InSAR computing environment for the research community to be able to process Level 0 data to Level 3 and support their modeling studies.
- Incorporate state-of-the-art, highly accurate algorithms to automate InSAR processing for non-experts and experts alike
- Document algorithms, formats and interfaces to facilitate community involvement in continuing development beyond the AIST horizon

A computation suite that facilitates interaction with InSAR data and models



Approach

- Develop community-based requirements for InSAR processing methods and generalized data models
- Develop a modular, extensible, object-oriented processing framework
- Develop modules for the ISCE architecture
- Test and document ISCE framework

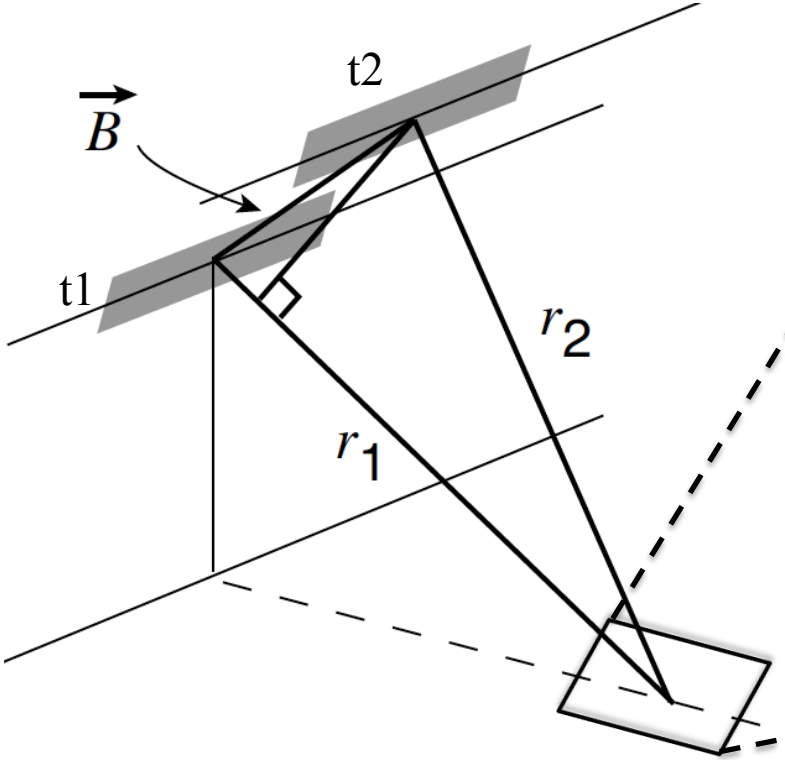
Status after First 15 Months

- Requirements defined
- Architecture designed
- Stanford University Subcontract in place (finally) to provide algorithmic developments for accuracy and additional functionality: time series, persistent scatterers…
- Framework recasting of processing engines more than half finished for ROI_PAC and well under way for Stanford Code after delays in subcontract

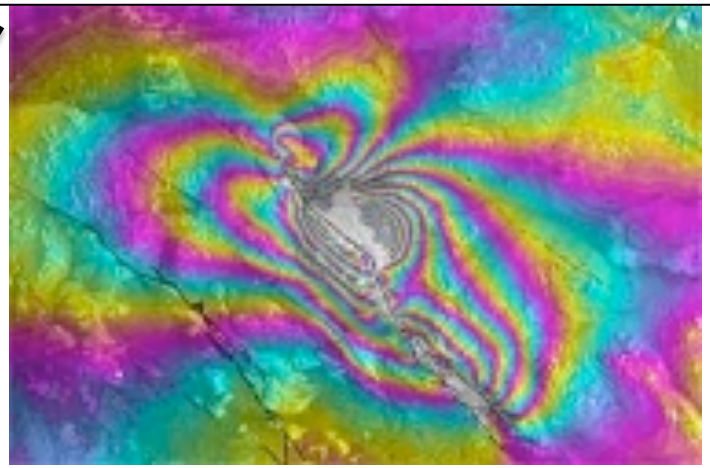
ESTO Earth Science Technology Office

ROI_PAC: Legacy InSAR Processing Software

ROI_PAC: Repeat Orbit Interferometry PACkage --- An Interferometric SAR processing package developed at JPL, Caltech, and Stanford over a decade ago. Widely used through an Open Channel License by a motivated scientist community to process raw radar data from two passes over a scene at different times from a variety of satellites (ERS, JERS, EnviSAT, ALOS), into images, interferograms, and geocoded topography and Earth displacement maps

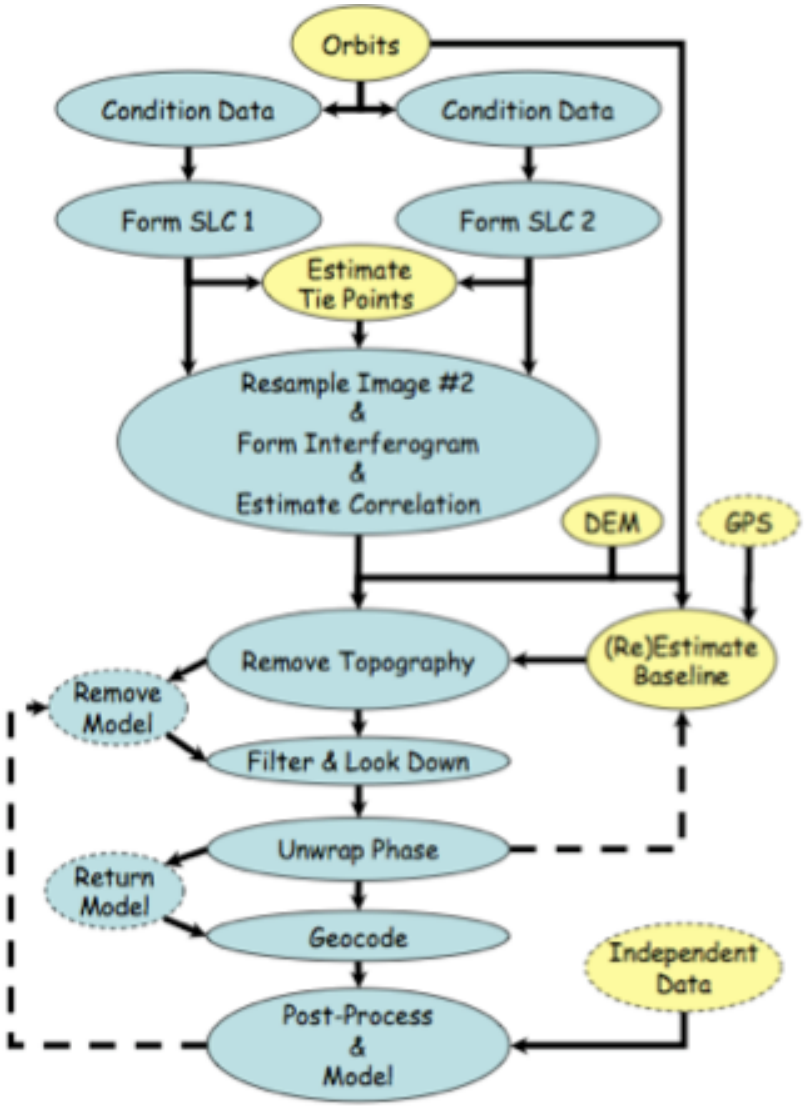


Map of Earth Displacement between t1 and t2



ROI_PAC + DEM

ROI_PAC: InSAR Processing Flow



InSARProc2008

ISCE is designed to be a response to the recommendations coming out of a NASA supported workshop (InSARProc 2008) convened at Stanford University on July 28-31, 2008. The workshop was attended by leading radar processing experts and scientists who use radar processing software from around the world.

The specific goals addressed by the workshop were to:

1. Assess strengths of existing InSAR processing non-commercial packages/methods
2. Define needs and capabilities of next-generation processing systems
3. Set standards and structure for new InSAR processor development

Sponsorship: InSARProc 2008 was endorsed by NASA's DESDynI Steering Committee and sponsored jointly by NASA's Earth Surface and Interior Program and by the Stanford Center for Computational Earth & Environmental Science (CEES).

The next InSARPROC Workshop is being scheduled for late 2010 in Southern California and will unveil an alpha version of ISCE.

InSARProc2008 Recommendations

The high-level guidelines and recommendations coming out of InSARProc2008 were prioritized in two groups:

Highest priority:

- The next-generation software should be accurate in phase and location
- The package should be extensible, modular, and efficient
- The package should be well documented, supported, accessible to all users

Second priority:

- The software should be portable, thus with a small and light footprint
- The new codes should be open source in the sense that they should be available to anyone for inspection, use, modification, and redistribution.
- The code should be thoroughly tested, debugged, pass benchmarks, and verified.
- Results should be readily reproducible and repeatable.
- The package should follow well-defined, standardized products with clear coordinates.

STD_PROC: Improved/Enhanced Processor for ISCE

• STD_PROC is a new InSAR processing package being developed at Stanford under the current AIST.

•STD_PROC overlaps much of the functionality of ROI_PAC but it also extends the functionality to include ***time-series*** analysis methods for analyzing evolution of displacement fields over time from multiple passes over a scene and to include ***persistent scatterer*** methods to allow interferometric processing in the presence of low-correlation.

• STD_PROC borrows from and builds upon the improved processing algorithms developed for SRTM and UAVSAR InSAR processors.

• STD_PROC applies a motion compensation algorithm to produce images in a common geographical coordinate system from the start to facilitate time-series analysis of interferograms formed from multiple pairings of radar images.

• STD_PROC is more efficient and much faster through the use of improved algorithms and the use of OpenMP

Key Drivers of the ISCE Architecture

Key drivers of the ISCE architecture:

1. Preserve the vast expertise and testing currently encoded in Legacy Software including both ROI_PAC and STD_PROC.

2. Make that Legacy Software more lean in terms of the number of auxiliary tasks it needs to do (such as self configuration and I/O configuration).

3. Build modern object oriented structures around and behind the legacy code to manage that code and push rather than pull user configuration onto that code before execution.

4. Implement common functions and services such as I/O through APIs to allow their implementations to change and to allow for user configuration and selection of those functions at run-time.

5. Build in polymorphism mechanisms to allow user selections to alter the implementations of major processing steps and common functions. Also allows just-in-time insertion of alternative functions and major components

Some Characteristics of the Legacy Software

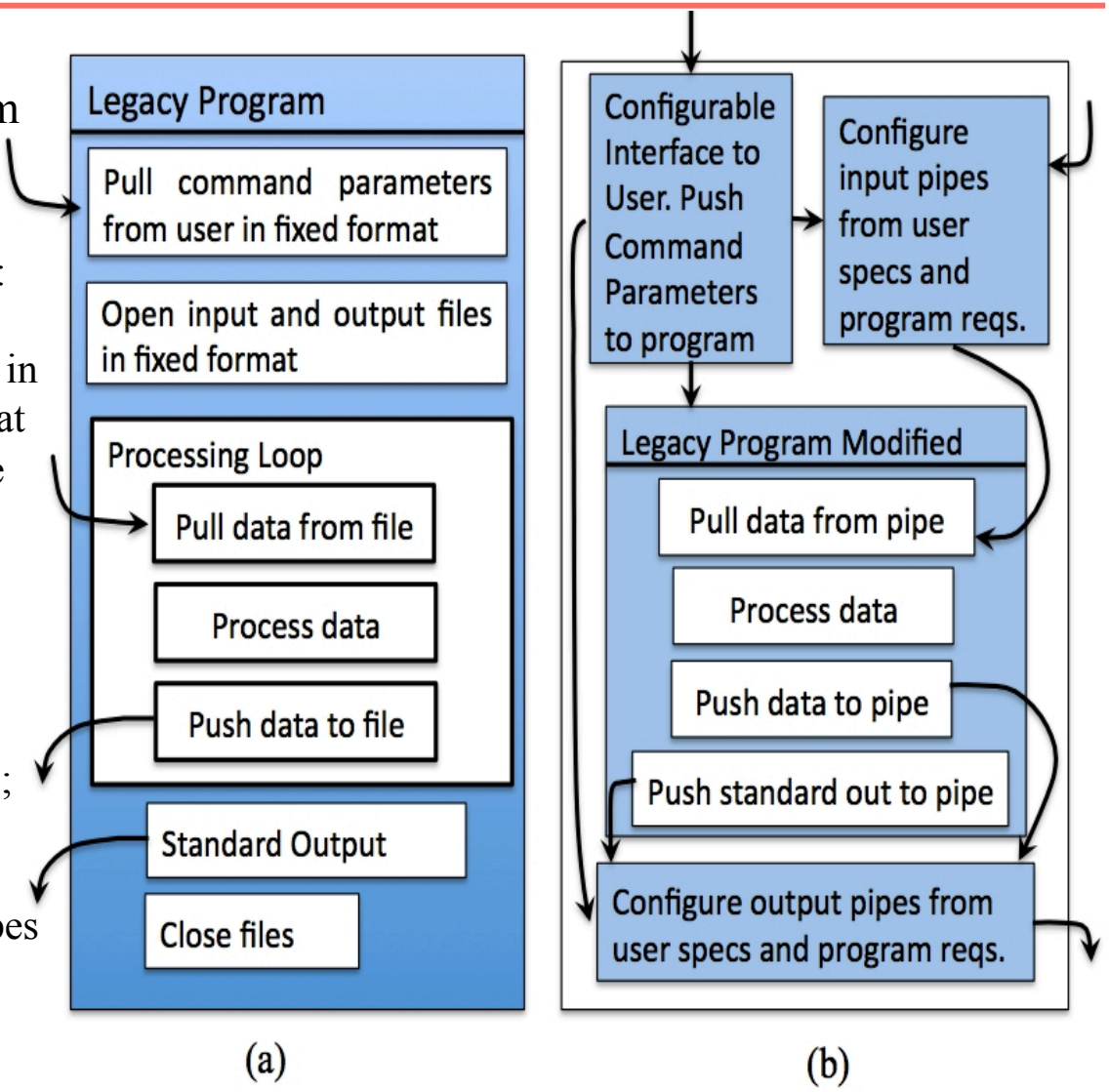
- Large monolithic "main" programs written mostly in Fortran with some C strung together with serial, task-oriented scripts written in Perl (ROI_PAC) or Python (STD_PROC).
- The scripts "exec" the main programs as separate processes one after the other.
- Data and information are passed between the various programs through external binary or text files.
- Often default control parameters and constants are hard-wired deep in the low-levels of the code in multiple locations and can only be changed by modifying the code and then management of the possible multiple instances of use of these parameters becomes difficult.
- Pieces of code are replicated in multiple places both in the low-level processing engines and in the higher-level scripts.
- The low-level programs are given the responsibility of opening data files and getting the data from those files, which binds the low-level programs to the particular external data storage formats. This requires either reformatting of large amounts of data before starting processing or else modification of low-level code for the external storage formats that may be in use at the time.
- Use of given scripts are simple for non-experts to use when they work properly but use of individual stages in the processing (for example when the script fails at a particular stage and debugging of that stage is required) is cumbersome or impossible for the non-expert to do.

Restructuring of a Legacy Program for Flexible Data Flow

Restructuring of a legacy program for data flow:

(a) Typical flow in a legacy program: control parameters and data/ standard I/O pulled from sources in fixed formats. Not configurable at run-time; inflexible; maintenance of code when formats change becomes difficult

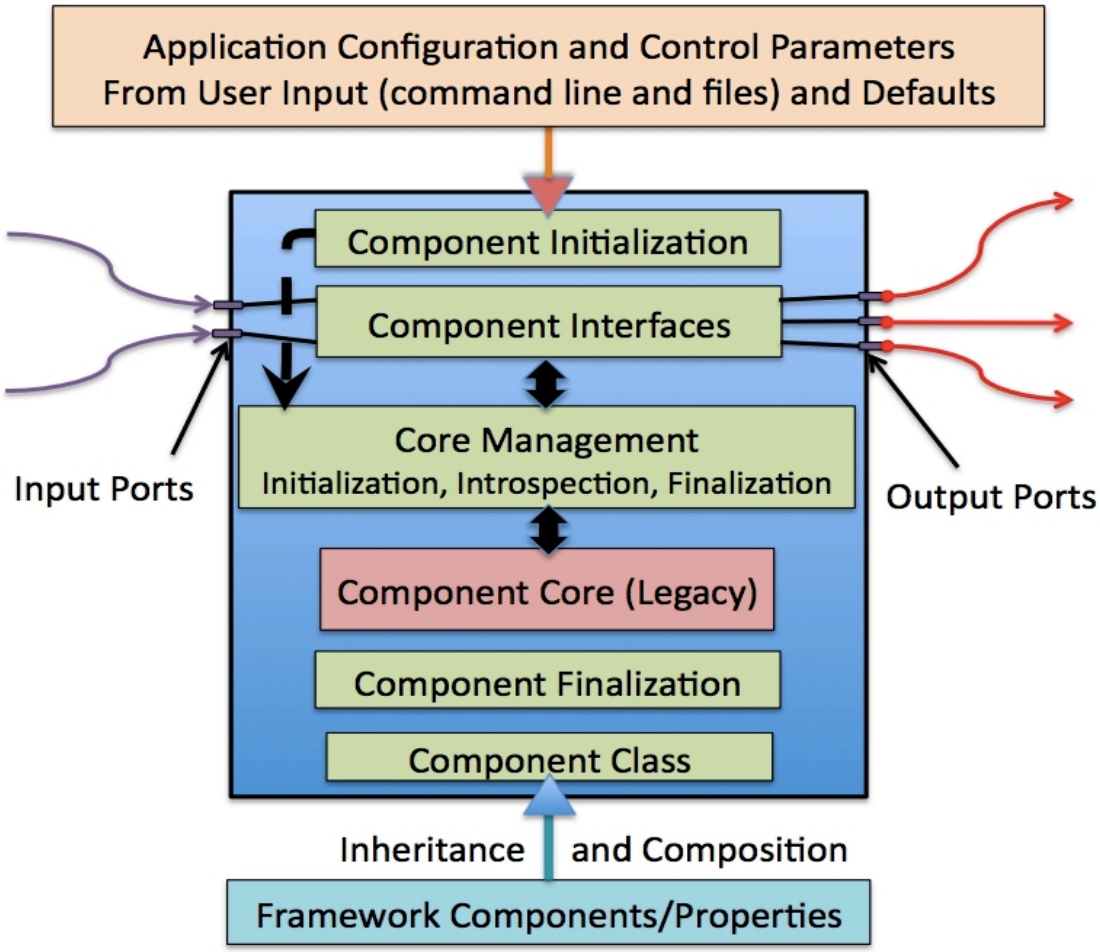
(b) The modified flow in ISCE: external processes manage interface to user and data sources; push control parameters onto the legacy program before execution phase begins; provide opaque pipes for data/standard I/O



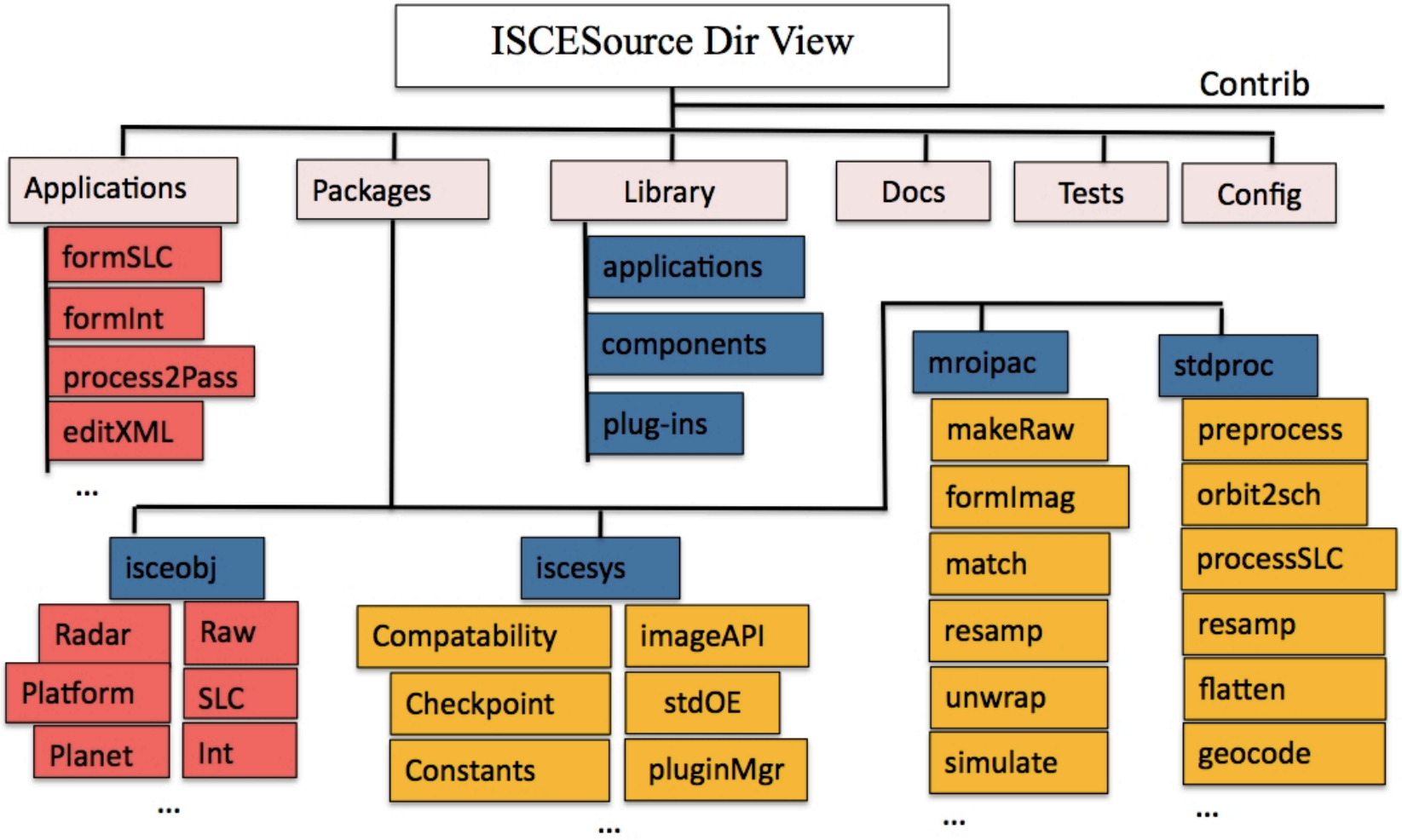
ISCE Component Architecture

Componentization of a legacy program:

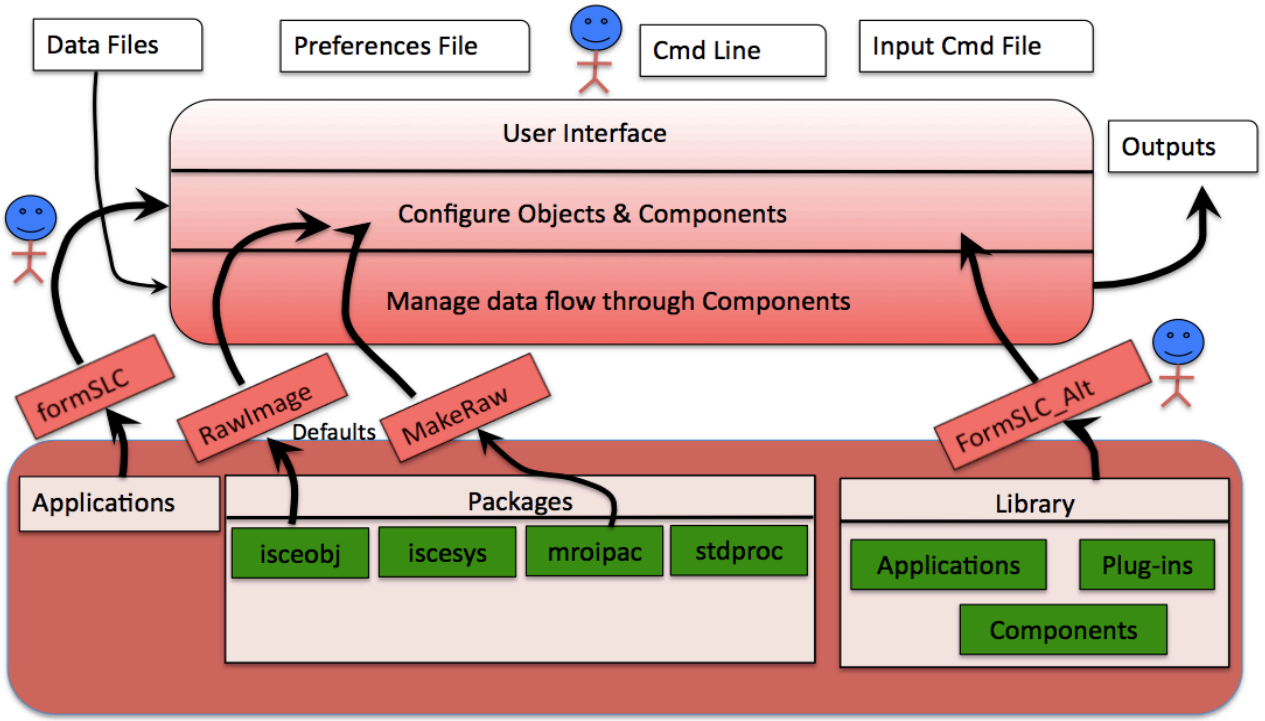
(a) Embed the legacy program at the core of an object-oriented component written in Python.
(b) Provide complete management of the core component through its life-cycle from initialization through proper finalization. Provide previously unavailable introspection capability
(c) Provide input and output ports with well-defined interfaces
(d) Provide well-defined interface for flow of control parameters from the user through controlling applications to the component.
(e) Provide Framework Components and Properties for common object definitions and common tasks.



ISCE Architecture



ISCE Applications



Application architecture. Applications are special components that contain "main" methods that replace the functionality provided by the PERL and Python scripts of ROI_PAC and STD_PROC. They are responsible for gathering user inputs, instantiating components and managing the execution of those components. The blue people indicate points where the user selects input parameters through input files and the command line as well as the components that are instantiated for a particular run.

ISCE Features: Polymorphism

• A key feature of ISCE that is meant to satisfy the requirements for flexibility and extensibility is the runtime polymorphism that we have built into ISCE.

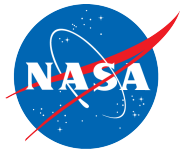
• Runtime Polymorphism is a software mechanism to significantly alter the behavior of the software at runtime through user inputs without requiring the software to be recompiled.

• Through object-oriented principles, *interfaces* and *tasks* can be defined in the software components while deferring the instantiation of the concrete software objects that will implement the *tasks* until run-time when user inputs can be used in helping to decide which objects are appropriate or preferred for the given task for a particular processing run.

•We allow for two types of polymorphism:

(1)Facility polymorphism where major components may be morphed at run-time. Facilities define a task and an interface that are implemented by a component. Registering a Component as a Facility indicates the Component as the default Component to implement the Facility but also alerts the Application to allow the User to specify an alternate Component at run-time to implement the Facility.

(2) P*lug-in* type of polymorphism where lower level, common functions such as implementations of fast Fourier transforms (FFTs) may be selected across the board at run-time.



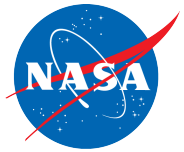
ISCE Features: Provenance

• To satisfy the requirement for well-documented products and for reproducibility of those products ISCE provides mechanisms to preserve the *provenance* of every data file produced by ISCE. Provenance refers to the pedigree of a particular piece of processed data which ISCE preserves in meta-data for data products as well as in a database of ISCE software configuration and usage.

• Versions of applications, components, and other software that were used to produce a data product as well as the configuration parameters used to initialize those applications and components as well as the provenance of the input data and other output data products at the time of creation of the data product of interest are all preserved.

• Provenance will allow an investigator to enter into an exploratory mode of processing his data in which he might try different versions of the software or perhaps iteratively tweak parameters to produce optimal data products while keeping a record of what he has tried at every point in his exploration so that he could reproduce any of the results that he produced at any time during his exploration.

• Provenance also allows users to create a record of what was done to the data that can be shared with the community in the form of publications or scripts, such that processing results are reproducible, which is an important aspect of scientific discovery and refinement.



ISCE Features: APIs

• *Image API* provides a set of library functions that provide the legacy software and new programs developed by users with a reliable and versatile way of performing input and output operations on images. The image API consists of a set of C++ classes that contain an abstraction of a real world image as well as concrete methods to access data from sources (such as, but not limited to, files on disc) and a memory buffer to hold a given portion of an image that can be passed between the C++ and Fortran programs. The C++ classes allow for very general and flexible configuration of the objects instantiated from them without specific regard for the types of images and memory buffer specifications currently in the Fortran programs of ROI_PAC and STD_PROC.

• *Control API* consists of a set of classes, features and methodologies that the ISCE framework utilizes to guarantee an easy, correct, reproducible, extensible and reconfigurable way of passing data among the different computing modules. The control API provides methods for setting, validating, and introspecting the control parameters of a low-level processing engine that was not previously available.

• *StdOE API* consists in a C++ static class used to handle standard Output/Error messages that is run-time configurable so that the destinations for logging messages from the low-level processing engines, for example, can be selected, or can be turned on or off, at run-time.

Conclusion

• An alpha version of ISCE is planned to be unveiled at the 2nd InSARProc Workshop to be held in late 2010 in Southern California. Radar processing experts and scientist users will be invited to discuss developments in InSAR processing since the Stanford 2008 Workshop. ISCE will be demonstrated and the community will try it out during the workshop and provide valuable feedback.

• A beta version of ISCE with full integration of ROI_PAC and STD_PROC software and fleshed-out framework support components to be completed by mid 2011 in time for the testing, bug-fixing, and documenting phase to follow into early 2012.

•Thank you for listening!