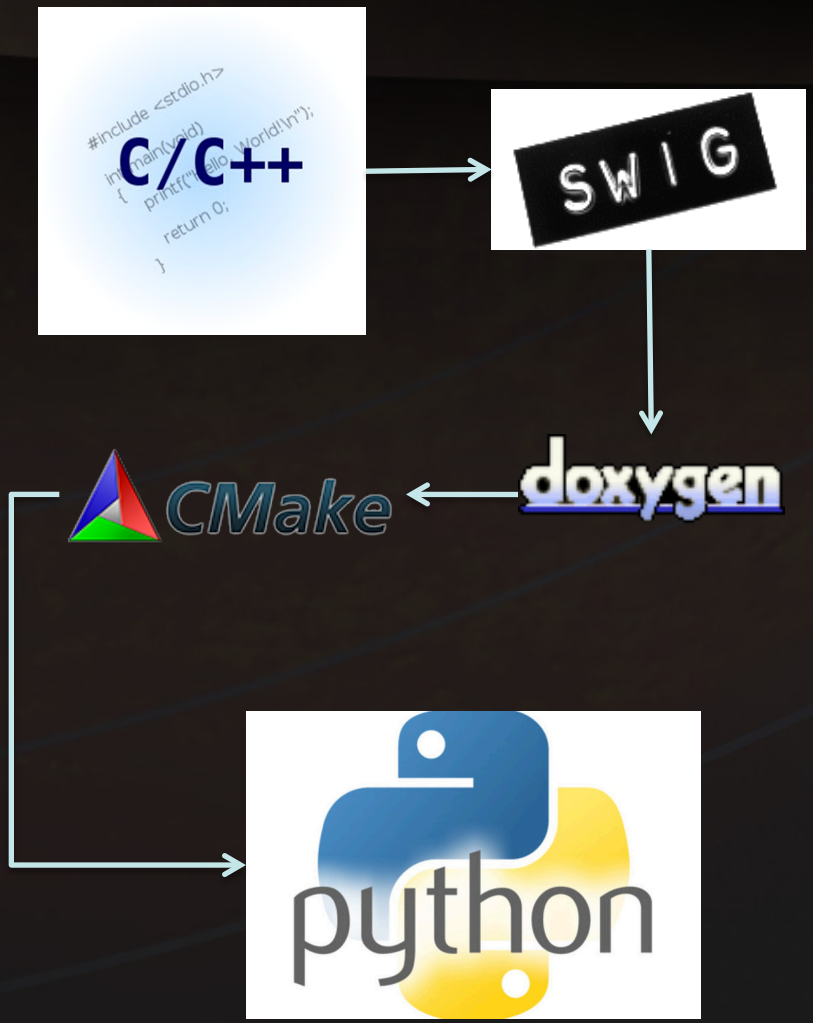
LASP

Laboratory for Atmospheric and Space Physics
University of Colorado **Boulder**

Flight Software Development Through Python

Scott Piggott, Maria Cols Margenet, John Alcorn,
P. Kenneally, and Hanspeter Schaub

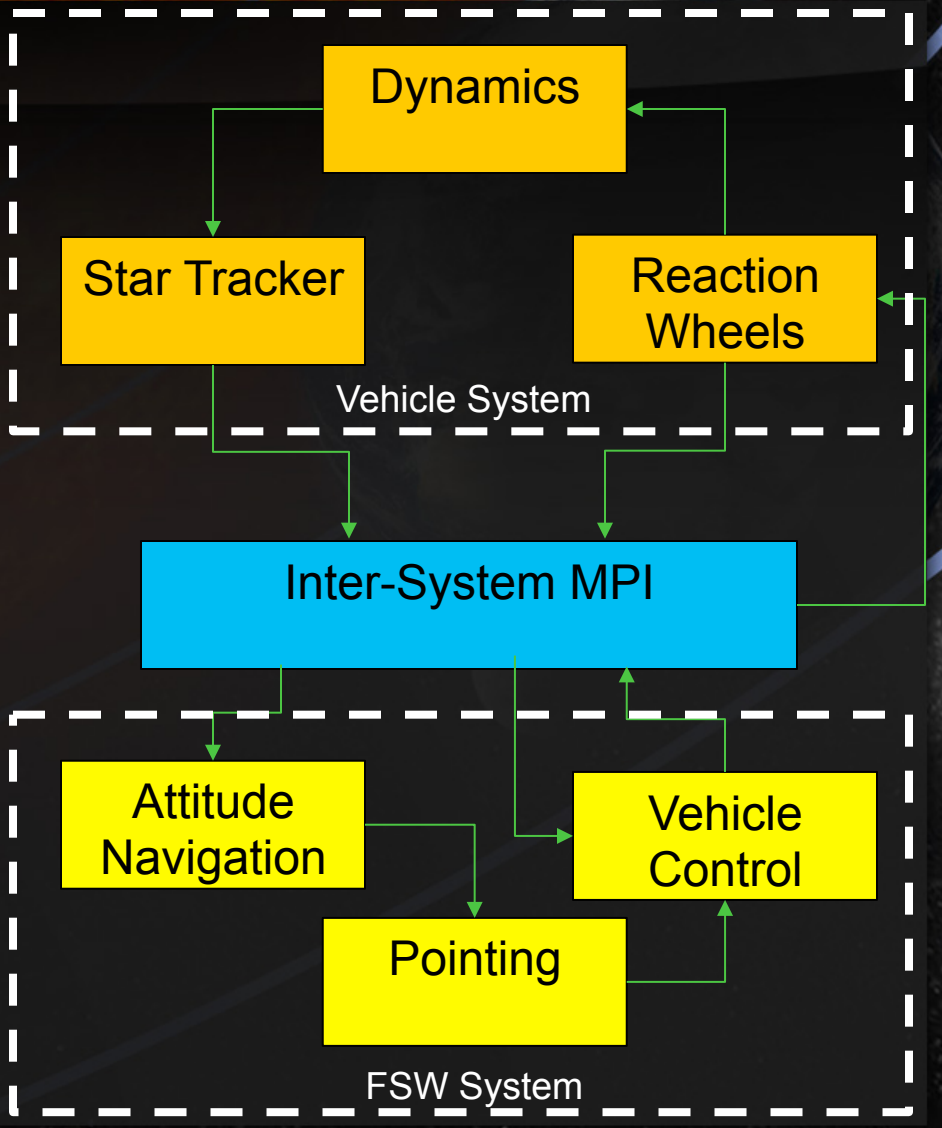
The Basilisk Development Environment



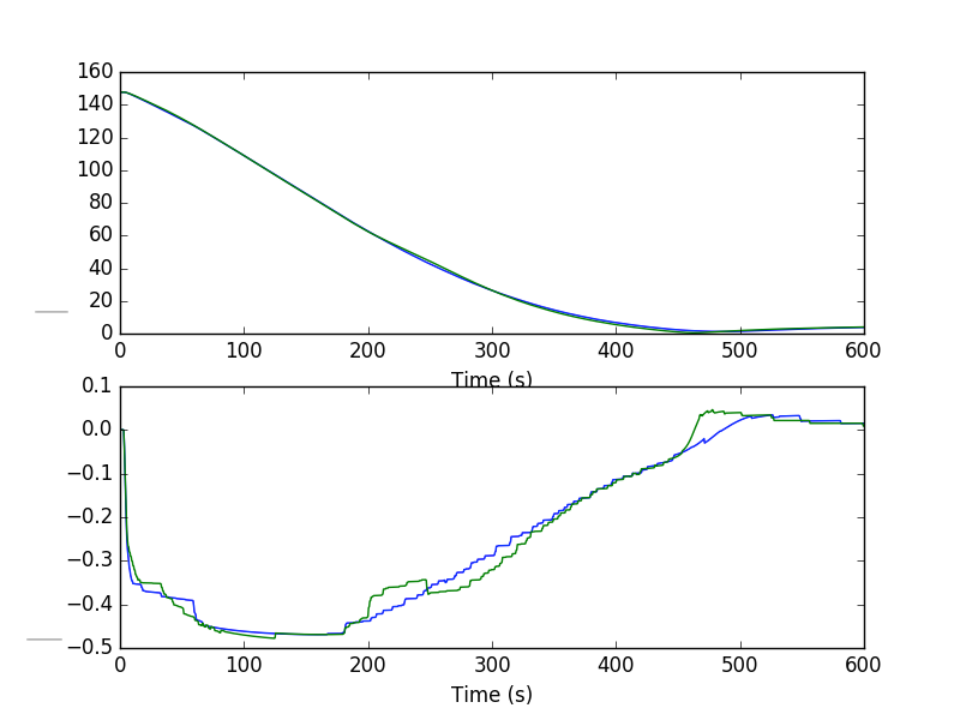
- Collaboration between LASP and the Autonomous Vehicle Systems Laboratory at the University of Colorado
  - Named for the South American Common Basilisk
- Simulation internally runs in C/C++ objects
- SWIG creates Python bindings that allow each module to be treated as Python objects
- Cmake allows the whole package to be built cross-platform
  - Windows, OSX, Linux

Relevant Basilisk Design Features

- Diagram greatly simplified!
- Algorithm interfaces are handled via message passing
  - Pieces work like “Legos”
- System abstraction allows user to sandbox messages
  - Dynamics, FSW, analysis, etc.
- Interface abstraction for multi-process
  - Separate FSW
- Granularity at the Python “brick” level
  - Completely scalable up to full integrated model



Transition to Target FSW

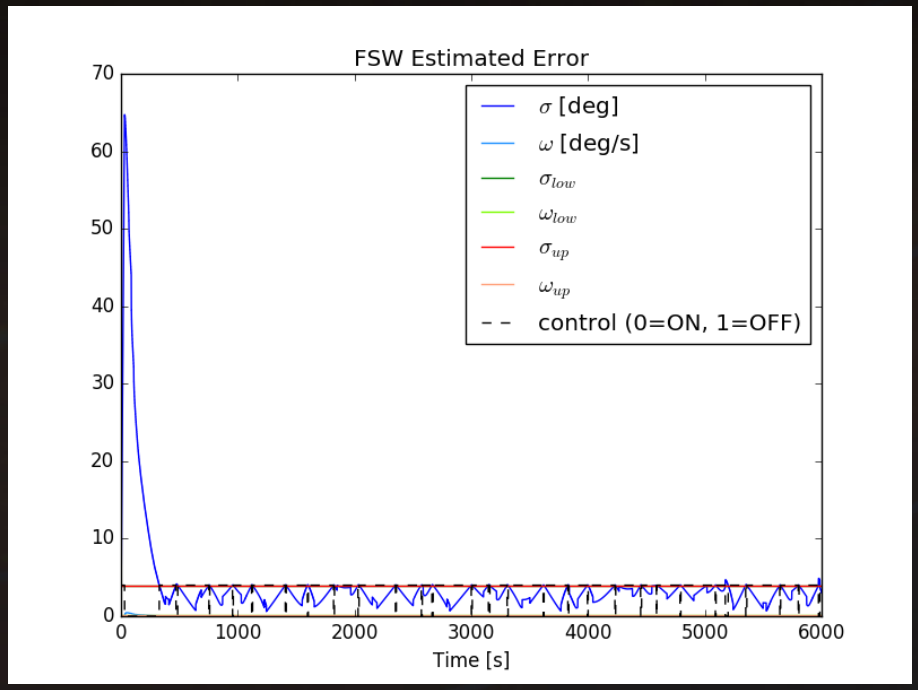


- Overall design philosophy is to capture what was executed
  - Python is very good at introspection
- Configuration parameters are captured post-initialization
  - Python can obtain all parameters from models
- Messaging map is obtained from messaging singleton post-run
- Task/Algorithm prioritization is reflected in code

Code Testing

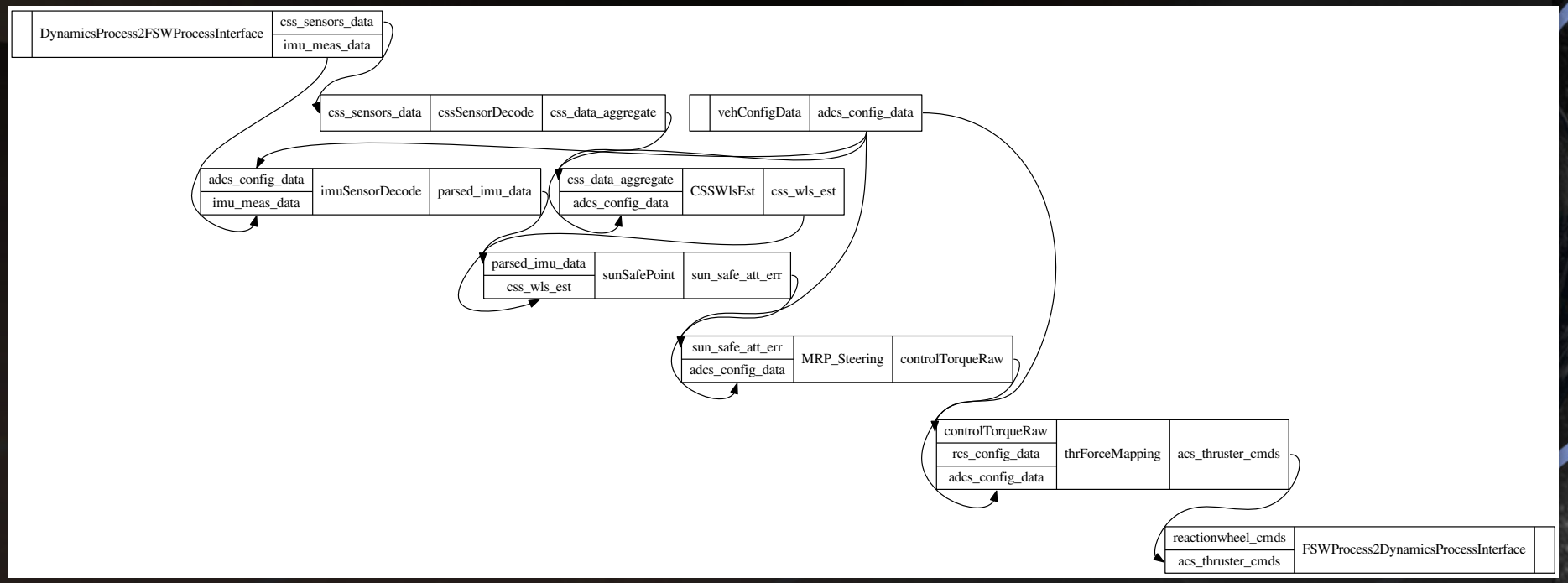
- Unit testing of algorithms performed via Python scripting
  - Significantly easier test setup than with C test-stubs
  - Inline test analysis easily performed with Python/numpy/matplotlib/etc
- Integrated system-level tests can be based around a central test base-class
  - Data display/analysis, remote communication configuration, scenario-macros
  - Post-run products can be generated automatically
- Automated testing can be easily integrated with continuous integration methodologies
  - Basilisk utilizes pytest for system checkout
    - Approximately 300 distinct tests running today
- Complex analysis like code coverage can be performed against the test-suite
- Mission-unique code can be separated from Basilisk and built/run/tested separately
  - Basilisk compatible code can also be integrated directly

Test Execution Analysis



- Basilisk is designed to facilitate analysis using Python-standard tools
  - numpy/matplotlib/pytest/etc.
- Any C/C++ variable accessible via SWIG/Python can be logged and interrogated
  - Haven't found a type yet that we can't make work!
- All system messages that are exchanged can be logged
  - User-specified rate
- All logs are returned as numpy arrays to simplify analysis

Autocoding a Basilisk Design

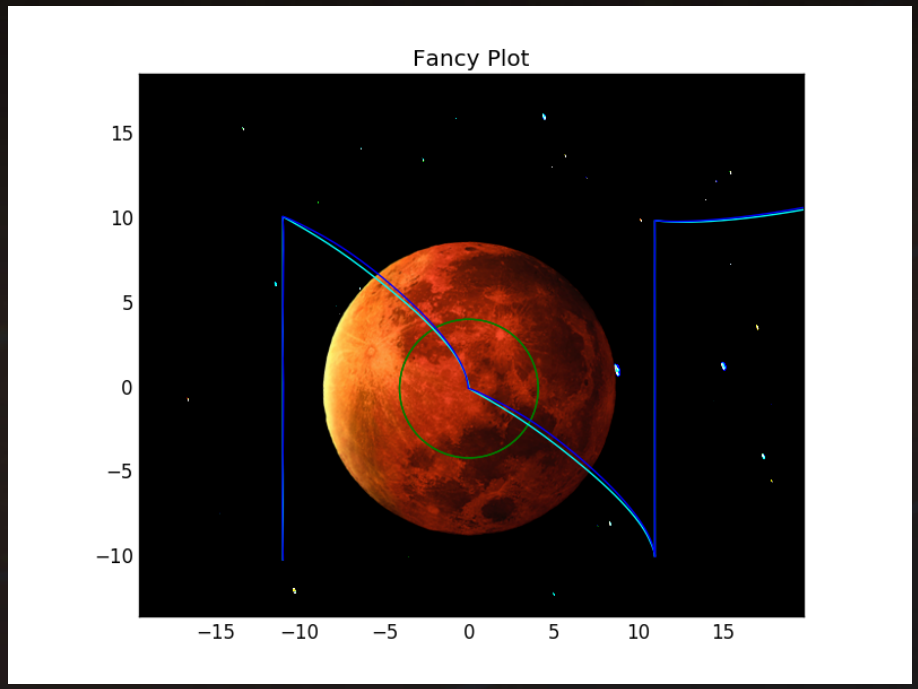


- Each Basilisk task can have boilerplate architecture generated
  - Queries itself to determine how to generate that data
- All model parameters are specified in a single function
  - Complex objects (arrays of structures for example) are totally usable with simple SWIG directives
- Each task requested is then configured with top-level function calls
- Separate rate-groups can be autocoded separately for multi-rate designs
- Analogous to Simulink or Labview autocode

RTEMS/CFS/Basilisk

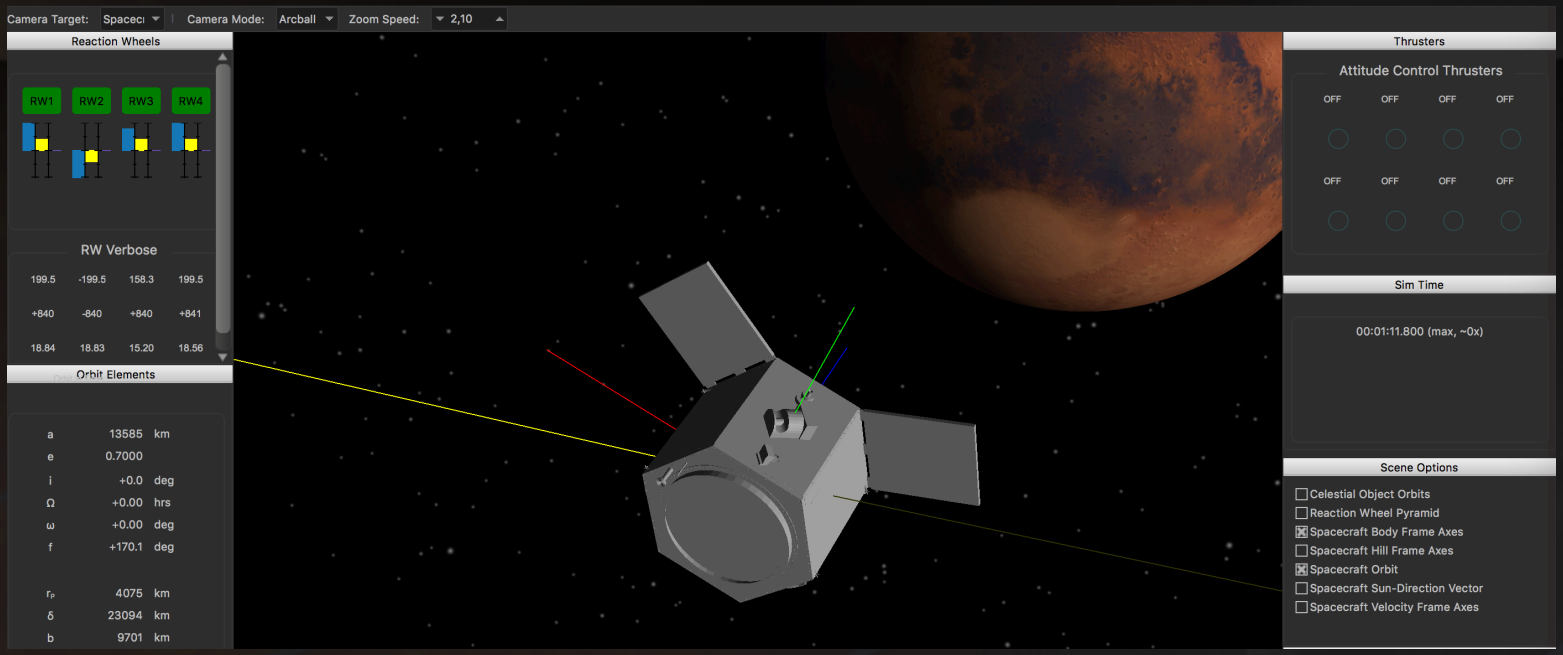
- LASP is working on a project that is using CFS running on the RTEMS operating system
  - Basilisk is being used for the ADCS application development
- All algorithm functionality dispatched from a single application (multi-rate) using an internal blackboard messaging system
  - CFS' messaging is ill-suited to Basilisk's messaging requirements
- Inter-application messages are mirrored out to CFE software bus
- Autocode is easily inserted into the main ADC app and called from CFS

Have you Scanned your Planet Today?



- Cyan line shows a distributed all-Basilisk simulation
- Blue line shows that same simulation running distributed with CFS
- Now we have easy reconfigurability
  - Python is our main development testbed
- Complex vehicle behaviors can be assembled from successive bricks
- All internal code can be examined and analyzed by you

Basilisk Development Direction



- Initial Alpha open-source release available January 15th
  - Invitations readily available!
- Anticipate a full open-source beta release sometime in 2017
- Simulation model verification (unit-level) complete
- Majority of software will be undergoing CDR review in 2017
- Simulation will serve as flatsat driver for flight program
  - Integration starting 2017
- Algorithms should be on-orbit by late 2020

Conclusions

- Basilisk is a free and open-source simulation and algorithm-development testbed
- Entire package is designed from the ground up for the full spacecraft lifecycle on programs of arbitrary complexity
- The entire Basilisk system can be run on most PC operating systems with the addition of some free and open-source products
  - Tested daily on Windows, OSX, and Linux (Mint)
  - Python, Cmake, SWIG, and your favorite compiler
- The topics covered today are not even the coolest parts of the system!
  - Full fidelity vehicle dynamics, vehicle visualization, etc.

Questions?