Basilisk: A Flexible, Scalable and Modular Astrodynamics Simulation Framework

Patrick Kenneally[*], Scott Piggott[‡] and Hanspeter Schaub[†]

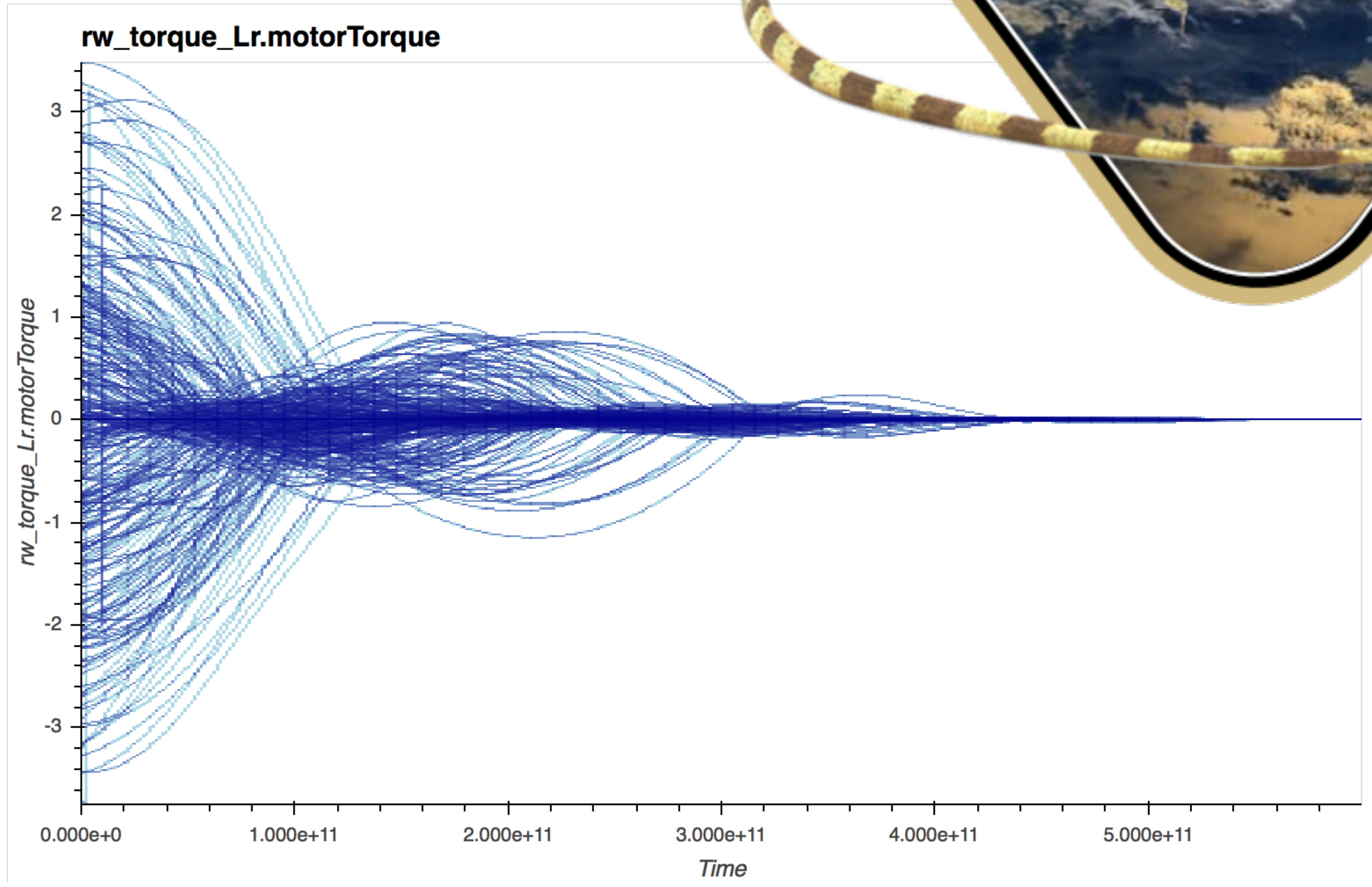
[*]Graduate Researcher, University of Colorado
[†]Professor, Glenn L. Murphy Chair, University of Colorado
[‡]ADCS Integrated Simulation Software Lead, Laboratory for Atmospheric and Space Physics

7th International Conference on Astrodynamics Tools and Techniques 6 - 9 November 2018

Ann and H. J. Smead Aerospace
Engineering Sciences Department
University of Colorado, **Boulder**

Agenda

- Motivation
- Basilisk feature overview
- Basilisk core components
- Basilisk messaging system
- Basilisk dynamics
- Monte carlo simulation
- Examples



Astrodynamics Simulation Tools

AVS Laboratory

- Extensible
- Customizable
- Coupled dynamics
- Hardware and software in-the-loop
- Open source

MATLAB® & SIMULINK®

STK®

42

DARTS HOME PAGE Spacecraft Dynamics Simulator

NASA GSFC · AFRL GMAT GENERAL MISSION ANALYSIS TOOL

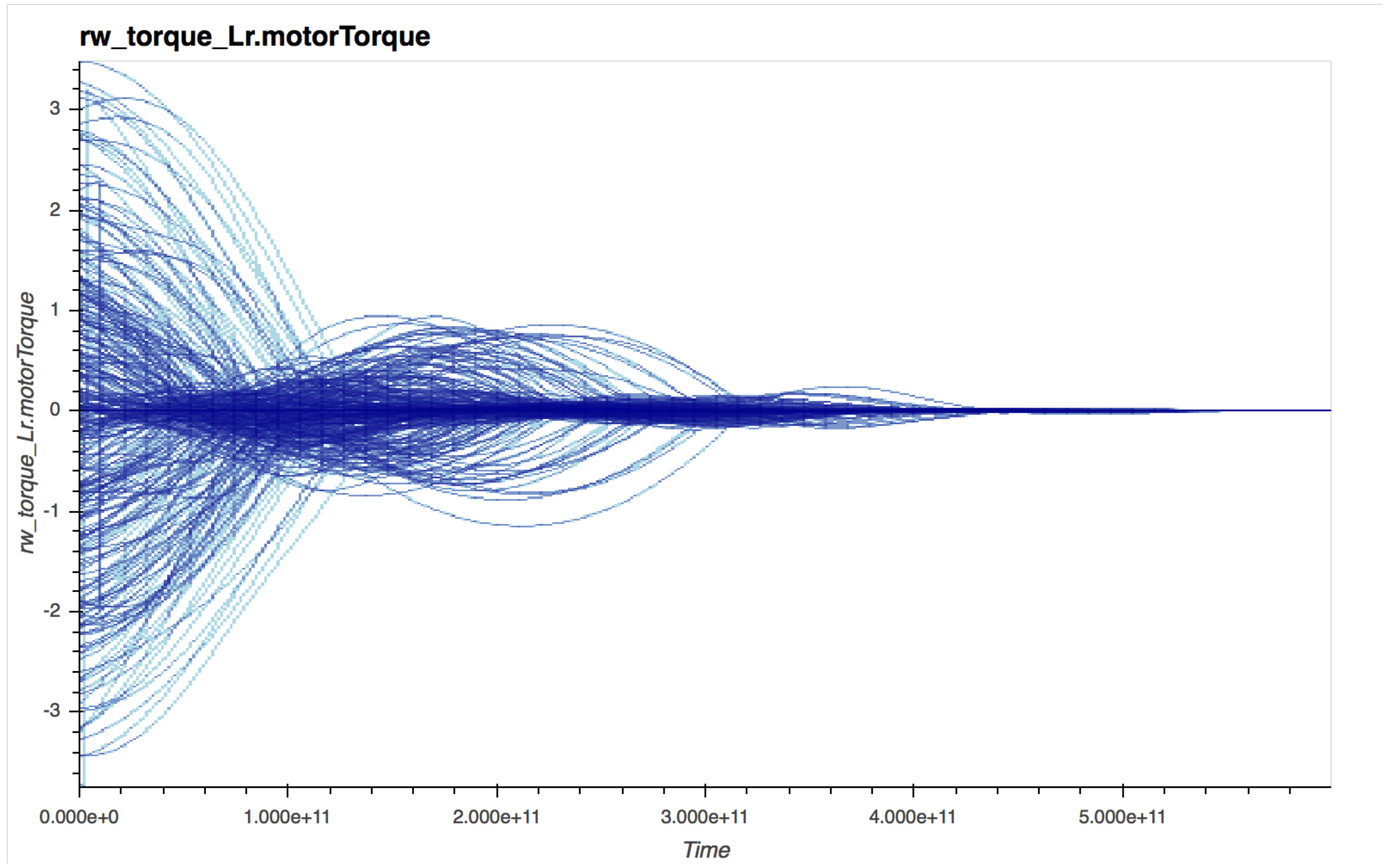
Orekit

FreeFlyer®

trick

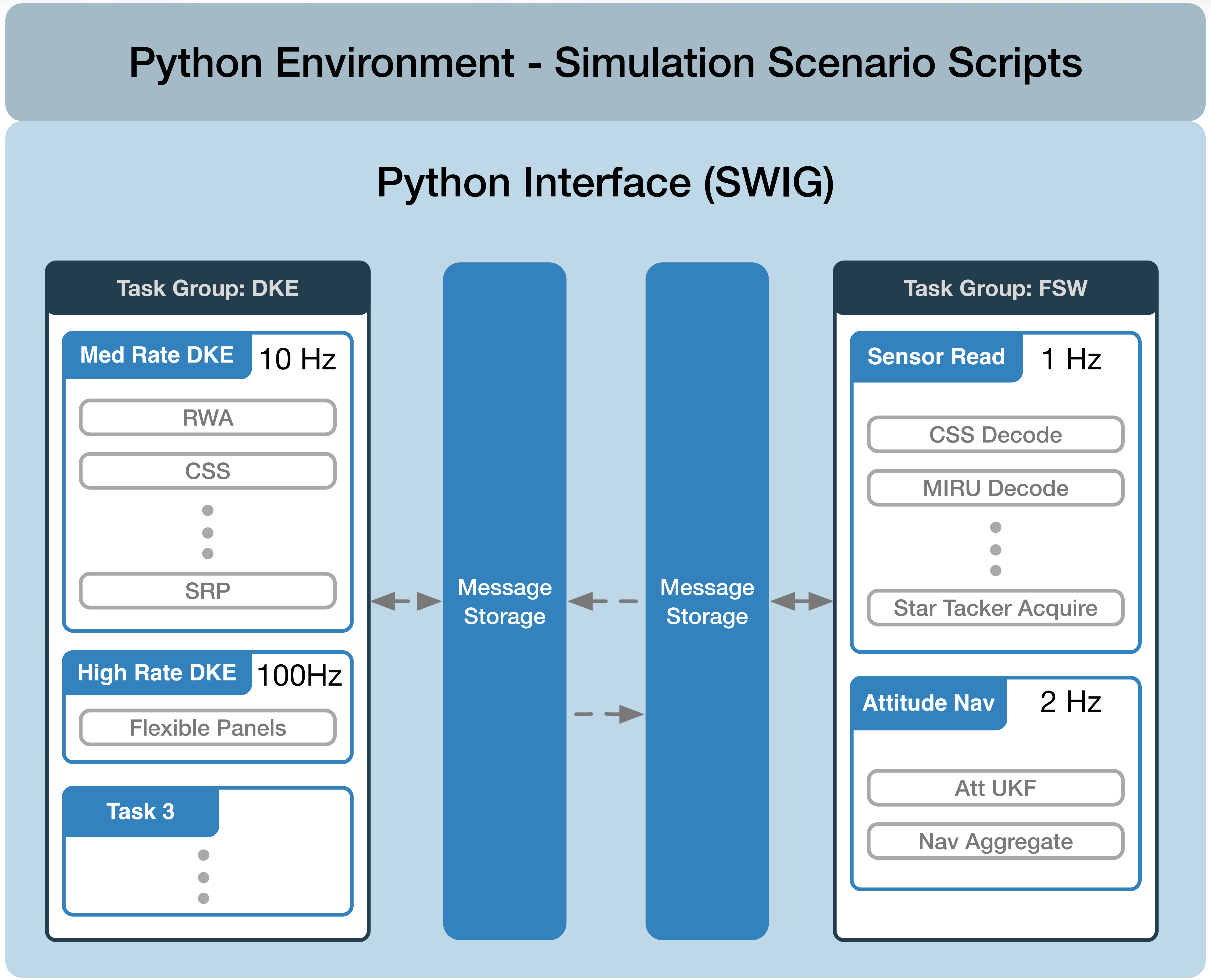
Basilisk Features

- Multi-body dynamics (docking and separation)
- Multiple spacecraft in single simulation
- Multiprocessing Monte Carlo
- Dynamic setting of integration rates
- Modular architecture - extendable across multiple machines and compute platforms
- Speed - simulate **1 year in 1 day** (full spacecraft attitude, orbit, devices)
- Python models (SWIG wrapped C++), analysis with Numpy, PANDAS, matplotlib, DataShader
- Open source



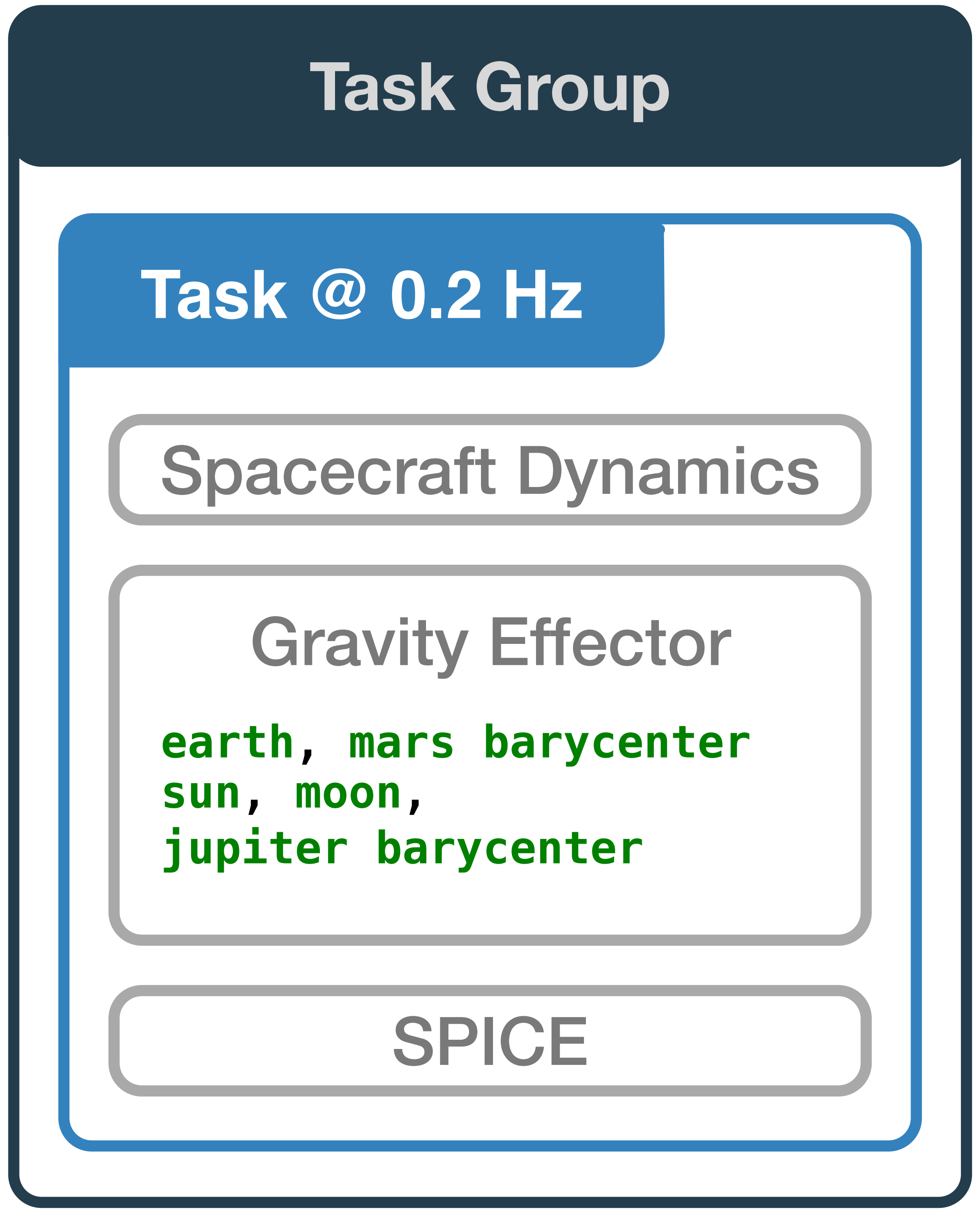
Architecture Overview

- Modules (models) written in C++/C/Fortran/ Python
- SoftWare Interface Generator (SWIG) generated Python interfaces for C++/C/Fortran Modules
- Data exchange between models achieved through a custom Messaging System
- Modules grouped by dynamically set integration rates



Simple Example Simulation Configuration

- Simple replication of Hubble Space Telescope trajectory




```
scSim = SimulationBaseClass.SimBaseClass()
```

```
dynProcess = scSim.CreateNewProcess(simProcessName)
dynProcess.addTask(scSim.CreateNewTask(simTaskName, sec2nanos(5)))
```

```
scObject = spacecraftPlus.SpacecraftPlus()
scSim.AddModelToTask(simTaskName, scObject, None, 1)
```

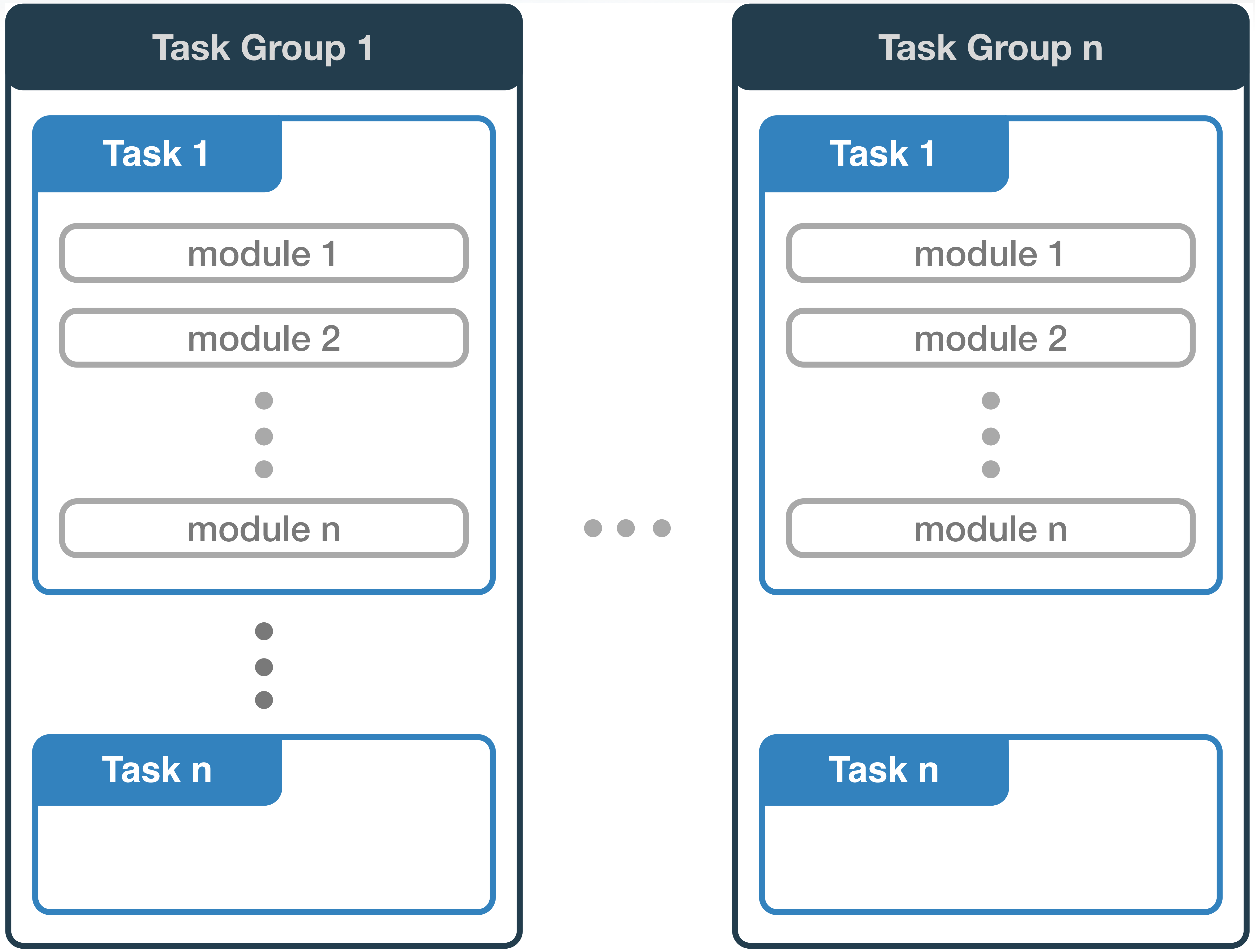
```
gravBodies = gravFactory.createBodies(['earth', 'mars␣barycenter', 'sun', 'moon', 'jupiter␣barycenter'])
scObject.gravField.gravBodies = spacecraftPlus.GravBodyVector(gravFactory.gravBodies.values())
```

```
gravFactory.createSpiceInterface(bskPath+'/supportData/EphemerisData/', timeInitString)
scSim.AddModelToTask(simTaskName, gravFactory.spiceObject, None, -1)
```

```
scSim.InitializeSimulation()
scSim.ConfigureStopTime(simulationTime)
scSim.ExecuteSimulation()
```

Basilisk Core Elements

- **Module**: a stand alone model or self contained logic
  - E.g. Actuator, sensor, dynamics model (SRP, drag, fuel slosh)
  - E.g. Translate a control torque to a RW command voltage
- **Task**: is a container for **Modules**, which has a rate (integration step)
- **Task Group**: a grouping of tasks within which **Modules** exchange messages.



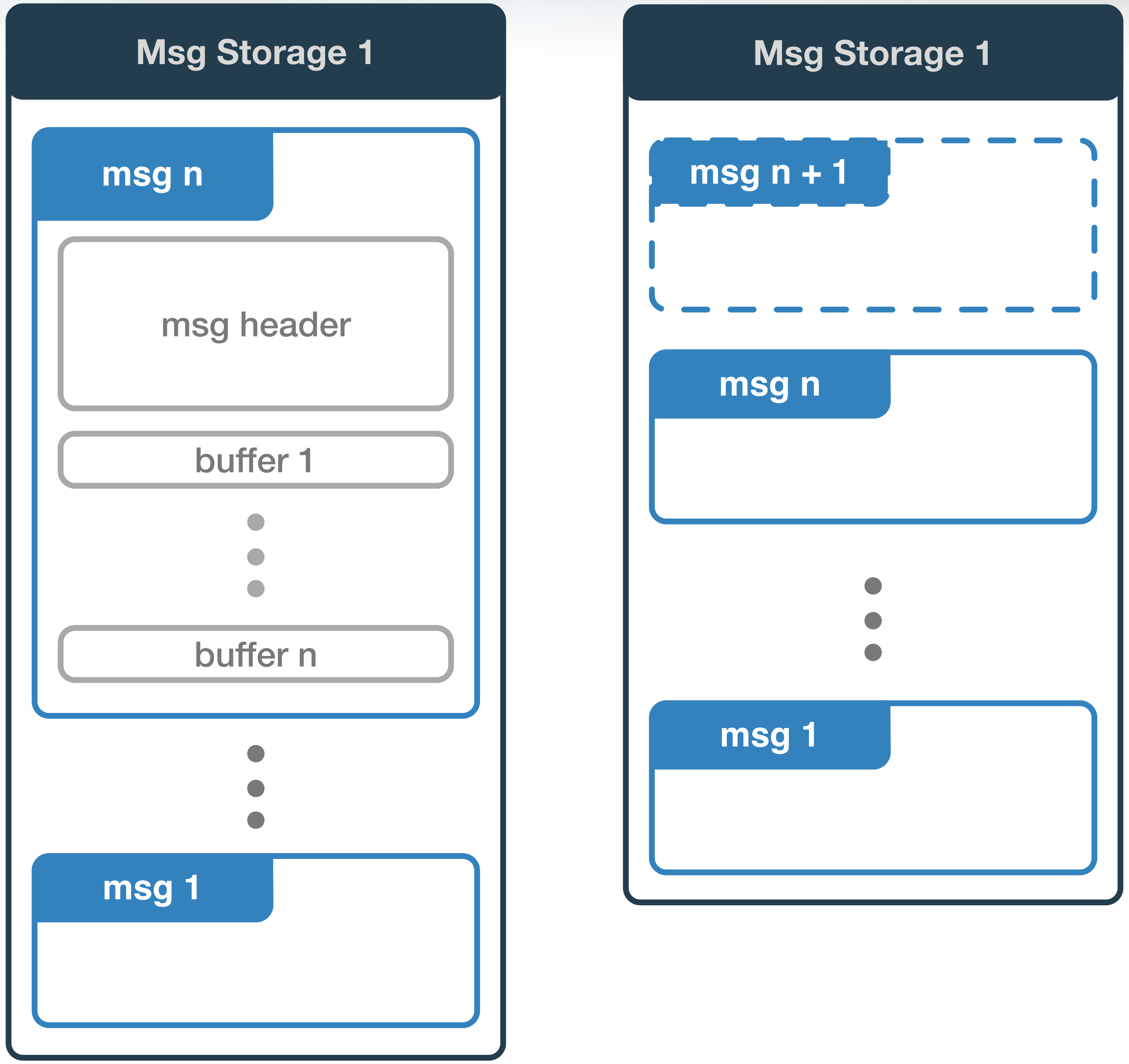
Basilisk Message System

- Messaging creates a common API for Modules to communicate, thus creating Module exchangeability.

- **Message**: a C++ struct

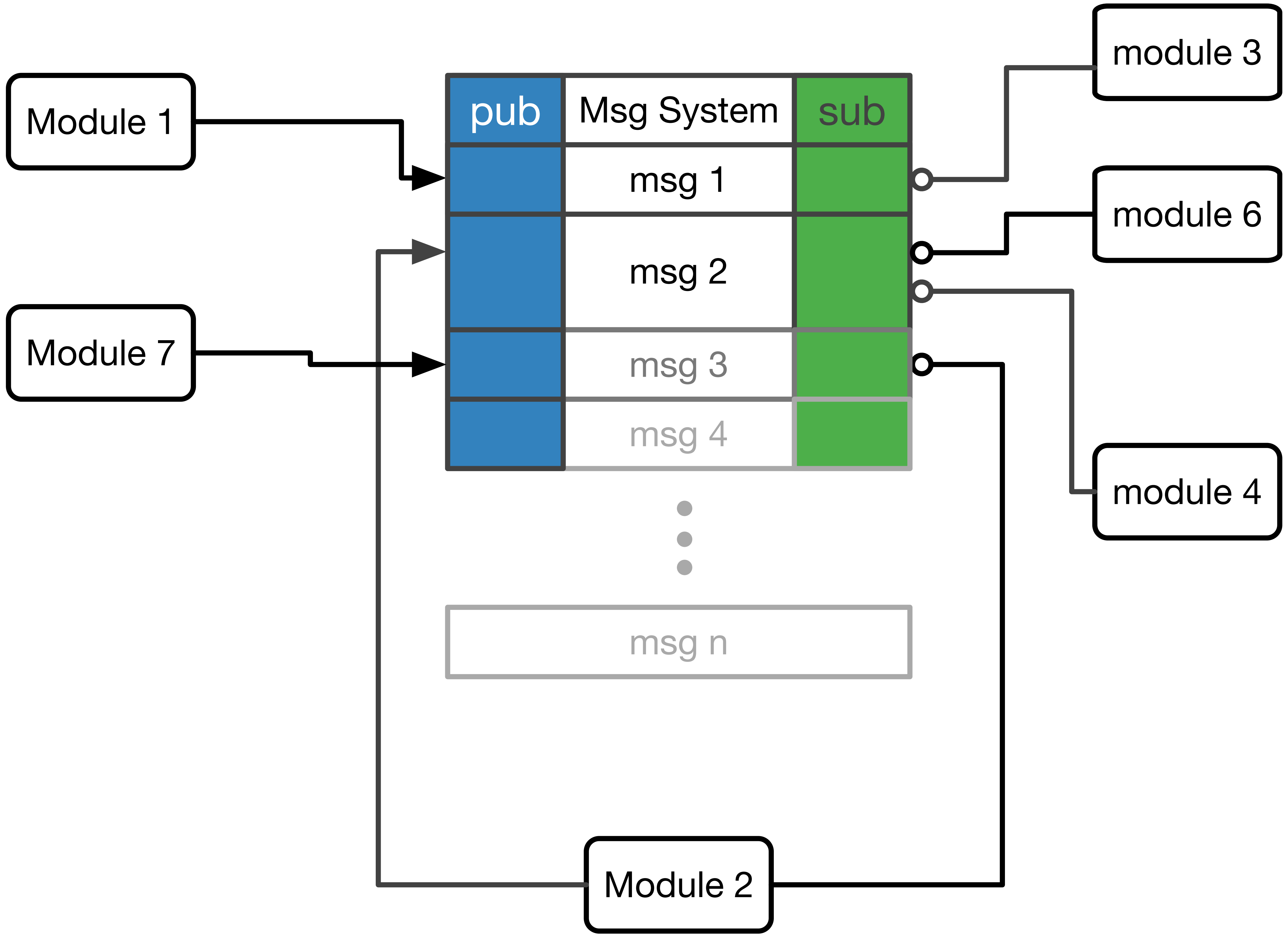
```
typedef struct {
    double maxThrust;
    double thrustFactor;
    double thrustForce = 0;
    double thrustForce_B[3] = {0};
    double thrustTorquePntB_B[3] = {0};
        components
    double thrusterLocation[3] = {0};
    double thrusterDirection[3] = {0};
}THROutputSimMsg;
```

- Each Task Group has an associated message storage container
- Messages are written directly into allocated memory
- Messages are read and written to the messaging system into N buffered message memory entry.
- Messages added to message storage memory block



Basilisk Message System

- A **Pub-Sub** paradigm is implemented to route module input and output messages.
- Message publisher and subscribers are resolved during simulation initialization.



Fully Coupled Dynamics

- StateEffector
  - coupled dynamics
  - states are managed by StateManager
- DynamicEffector
  - uncoupled dynamics
- integrateState( ) called upon the important spacecraftPlus( ) Module

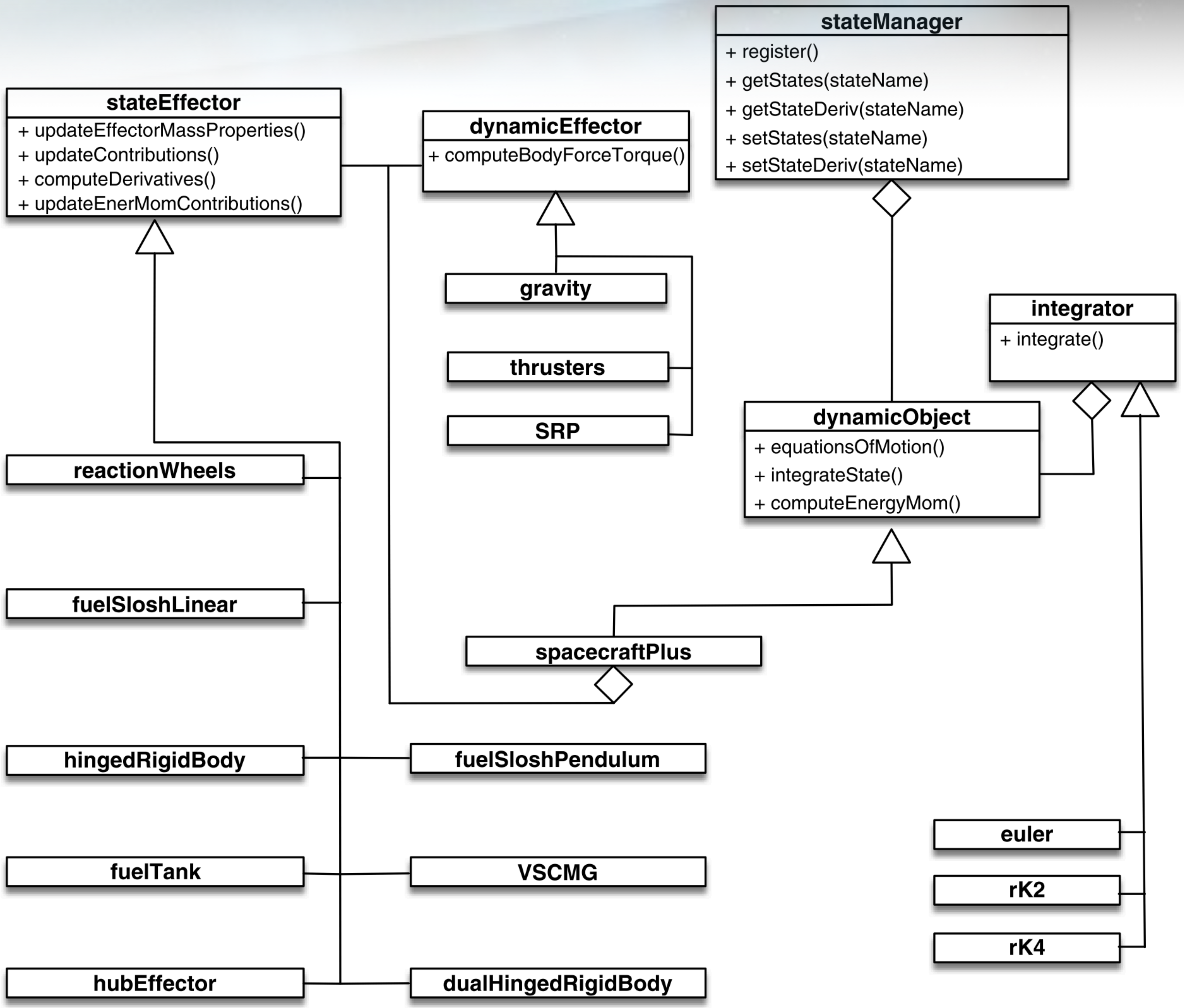
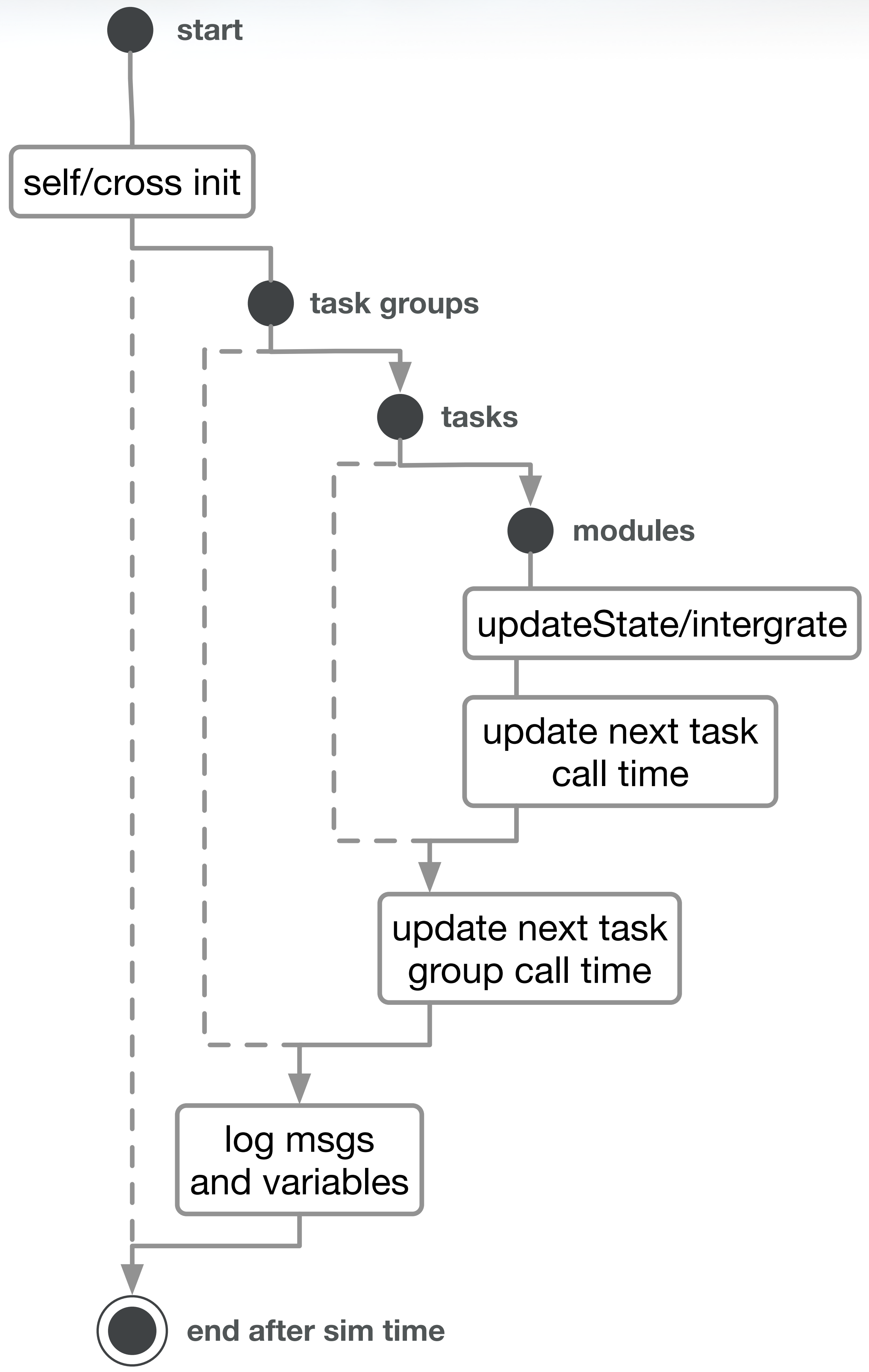


Fig. 4 UML diagram for modular architecture.

Execution Control

- Initialization to set Module defaults and resolve messages
- Loop through all Task Groups
    - Loop through all Tasks
        - Loop through Modules
- Update next call times
- Log messages and variables



Data Logging

- Data from messages logged at request

```
scSim.logThisMessage(scObject.scStateOutMsgName, logRate)
posData = scSim.pullMessageLogData(scObject.scStateOutMsgName +
'.r_BN_N', range(3))
velData = scSim.pullMessageLogData(scObject.scStateOutMsgName +
'.v_BN_N', range(3))
```

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

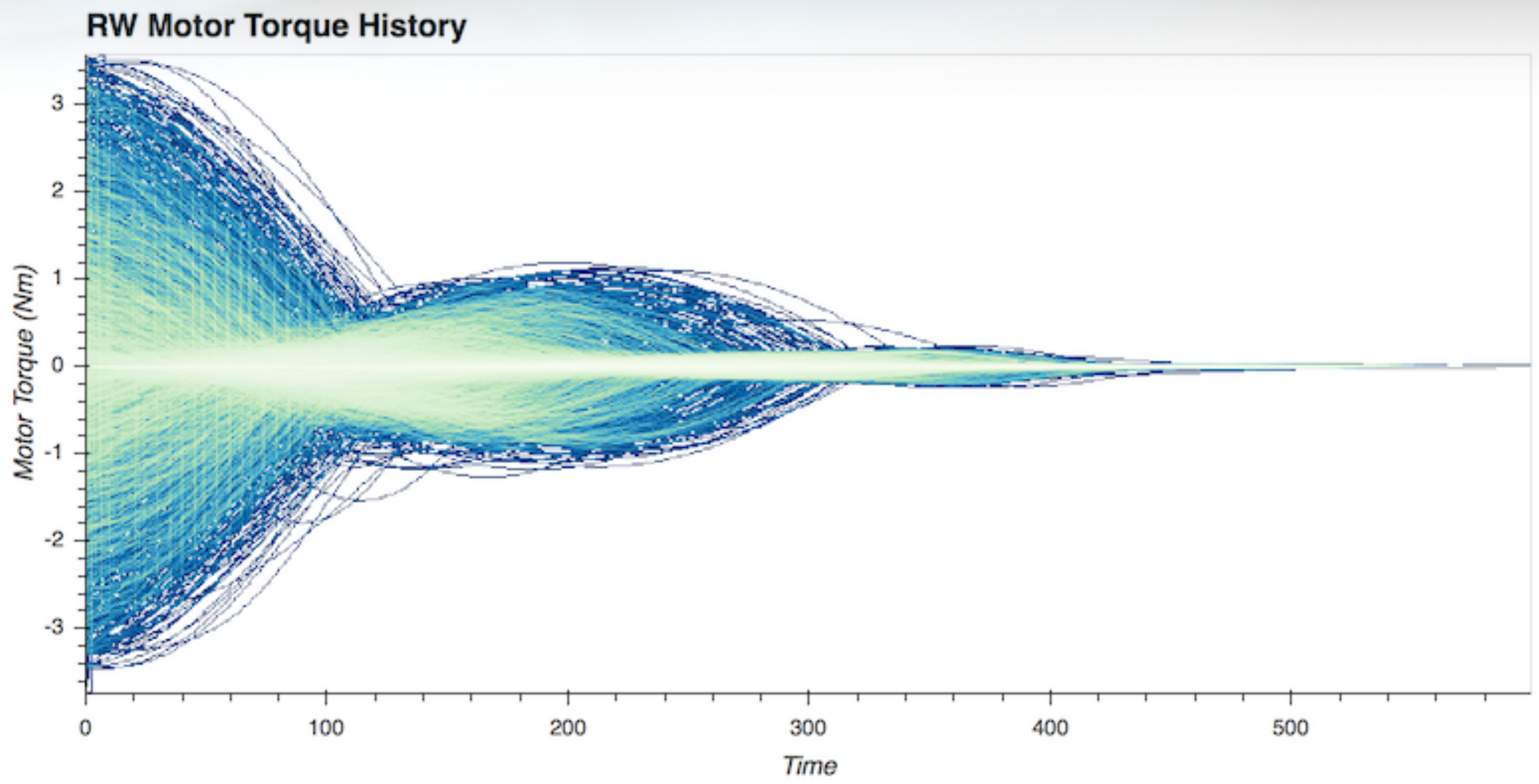
NumPy

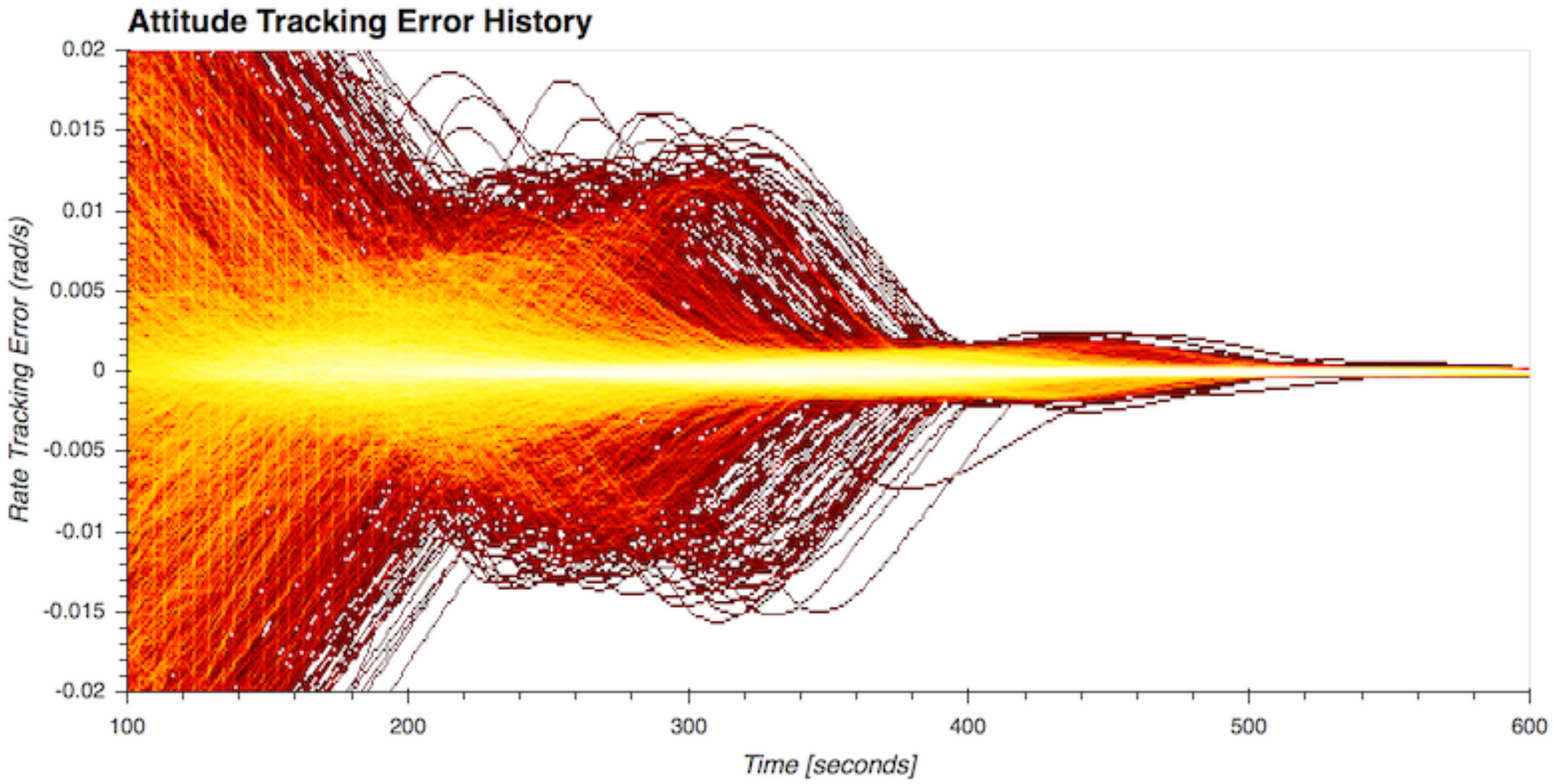
- Data from variables with public scope can be logged

```
scSim.addVariableForLogging(scObject.ModelTag + ".primaryCentralSpacecraft" + ".totOrbEnergy", logRate, 0, 0, 'double')
orbEnergy = scSim.getLogVariableData(scObject.ModelTag + ".primaryCentralSpacecraft" + “.totOrbEnergy")
```

Monte Carlo

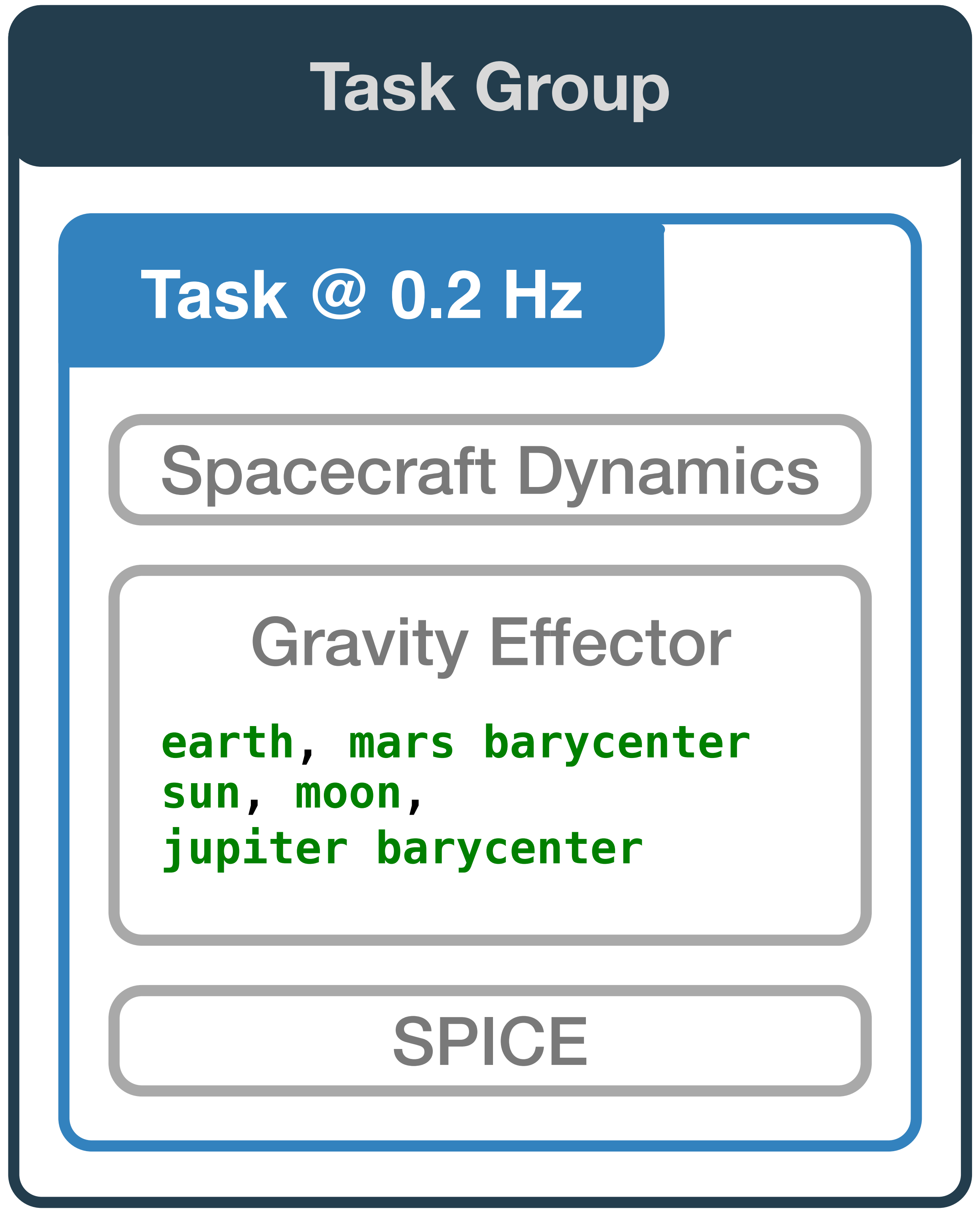
- Multi-processing MC runs
- Dispersions applied to all accessible variable types (scalar, vector, tensor)
- Bit-for-bit repeatable: initial conditions saved as JSON file and can be rerun
- Data analysis and post processing with PANDAS
- Multi-gigabyte data sets plot within second using DataShader plugin to Python's Bokeh plotting module





Example Simulation Configuration

- Simple replication of Hubble Space Telescope trajectory




```
scSim = SimulationBaseClass.SimBaseClass()
```

```
dynProcess = scSim.CreateNewProcess(simProcessName)
dynProcess.addTask(scSim.CreateNewTask(simTaskName, sec2nanos(5)))
```

```
scObject = spacecraftPlus.SpacecraftPlus()
scSim.AddModelToTask(simTaskName, scObject, None, 1)
```

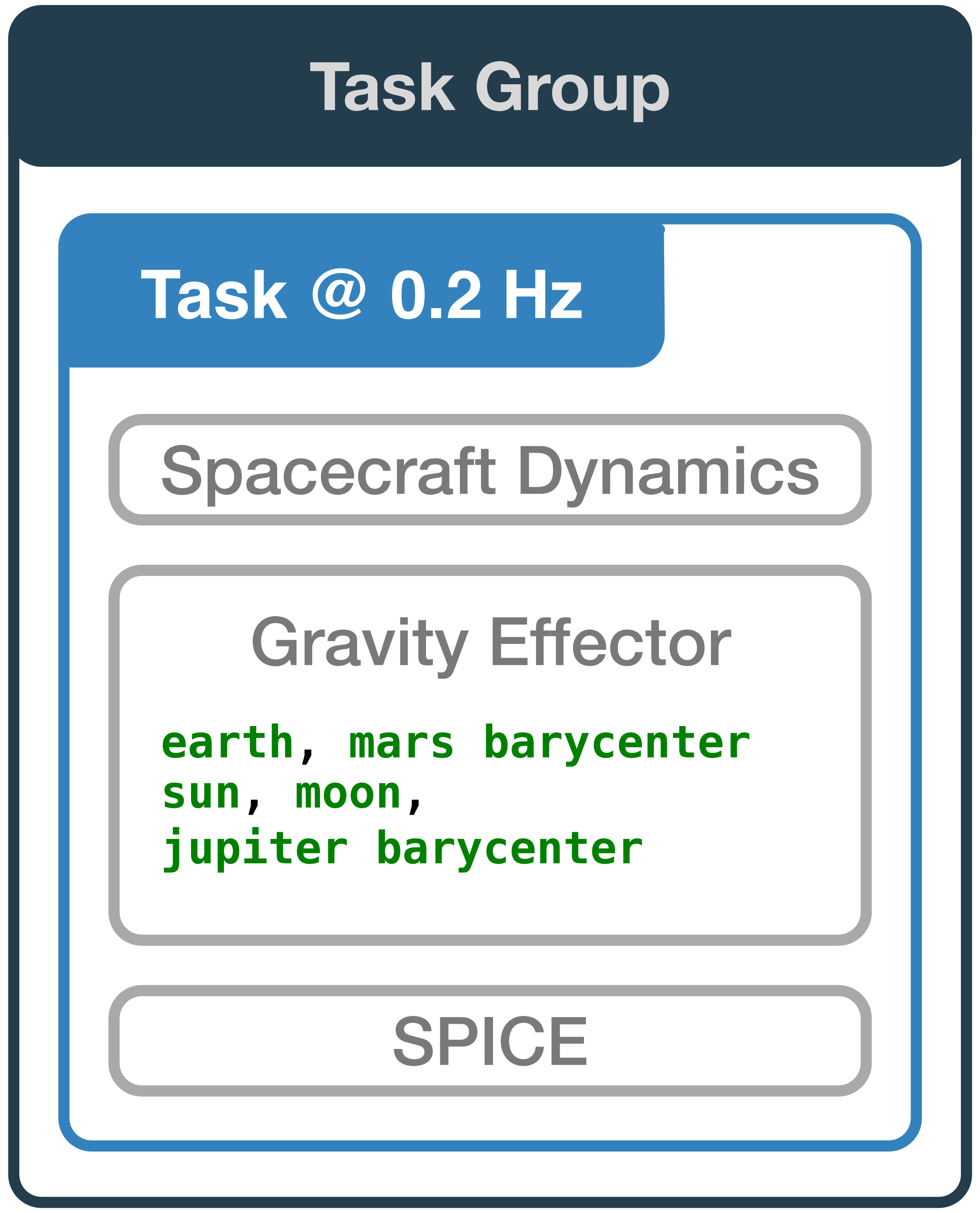
```
gravBodies = gravFactory.createBodies(['earth', 'mars␣barycenter', 'sun', 'moon', 'jupiter␣barycenter'])
scObject.gravField.gravBodies = spacecraftPlus.GravBodyVector(gravFactory.gravBodies.values())
```

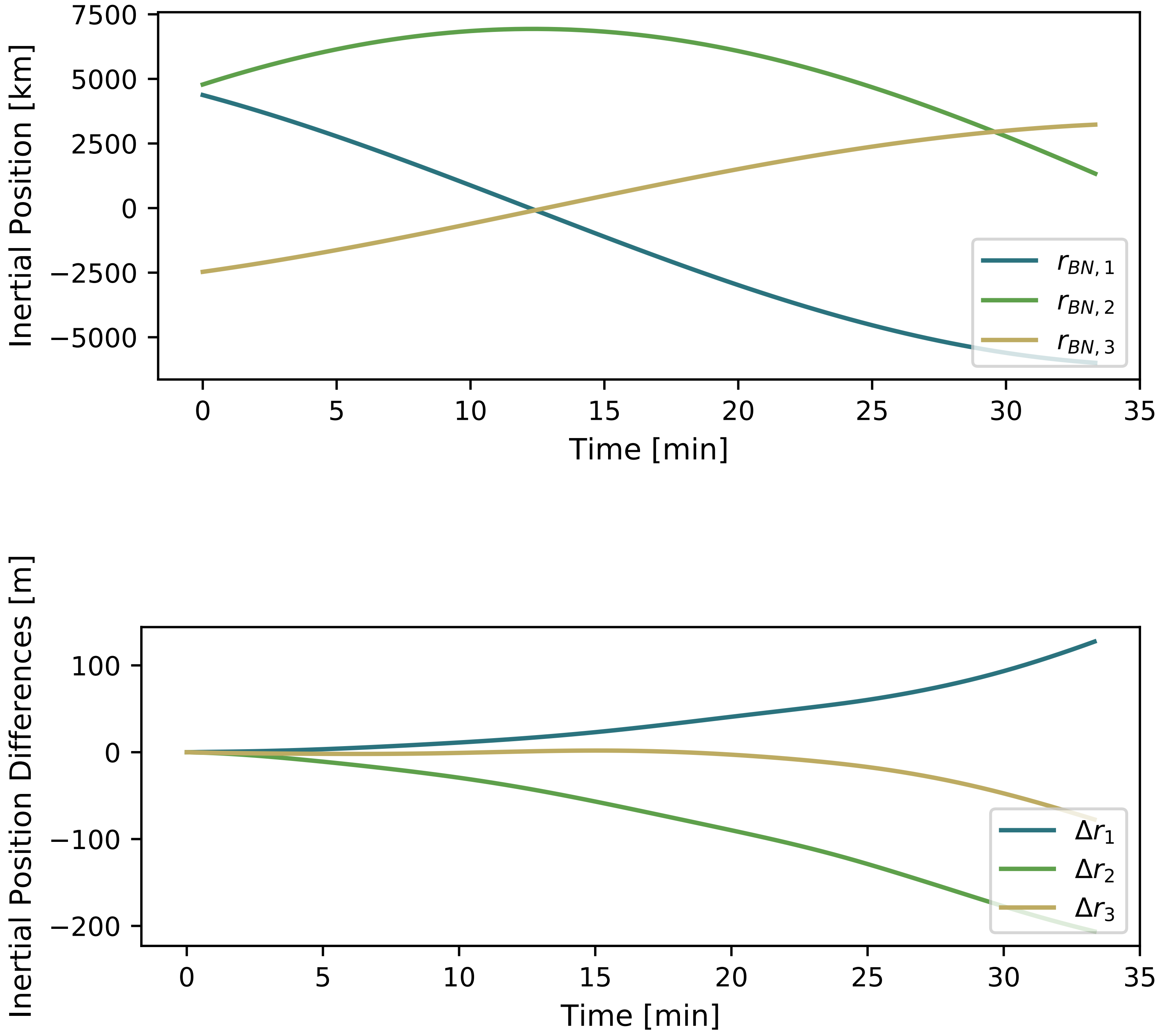
```
gravFactory.createSpiceInterface(bskPath+'/supportData/EphemerisData/', timeInitString)
scSim.AddModelToTask(simTaskName, gravFactory.spiceObject, None, -1)
```

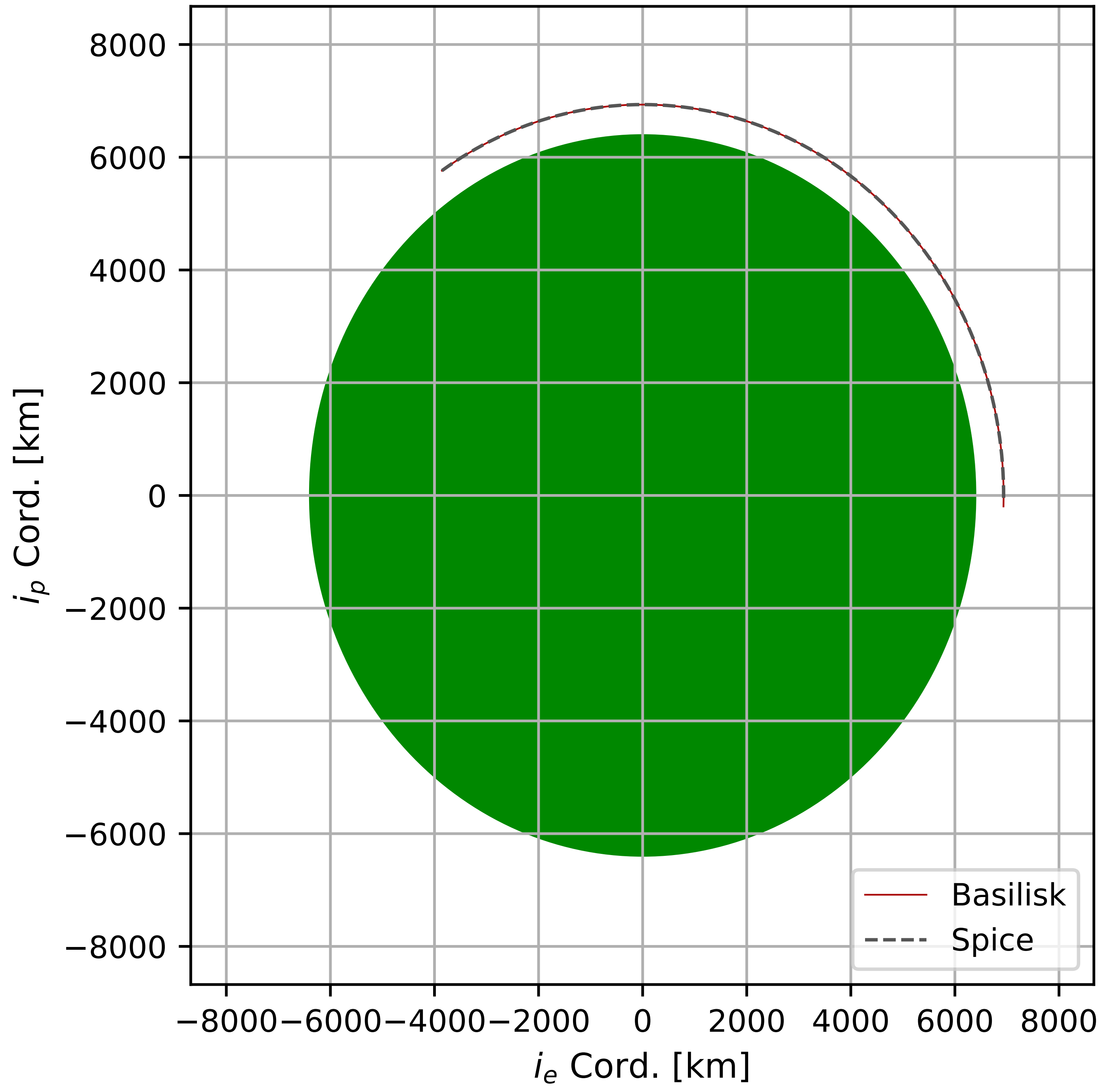
```
scSim.InitializeSimulation()
scSim.ConfigureStopTime(simulationTime)
scSim.ExecuteSimulation()
```

Example Simulation Configuration

- Simple replication of Hubble Space Telescope trajectory



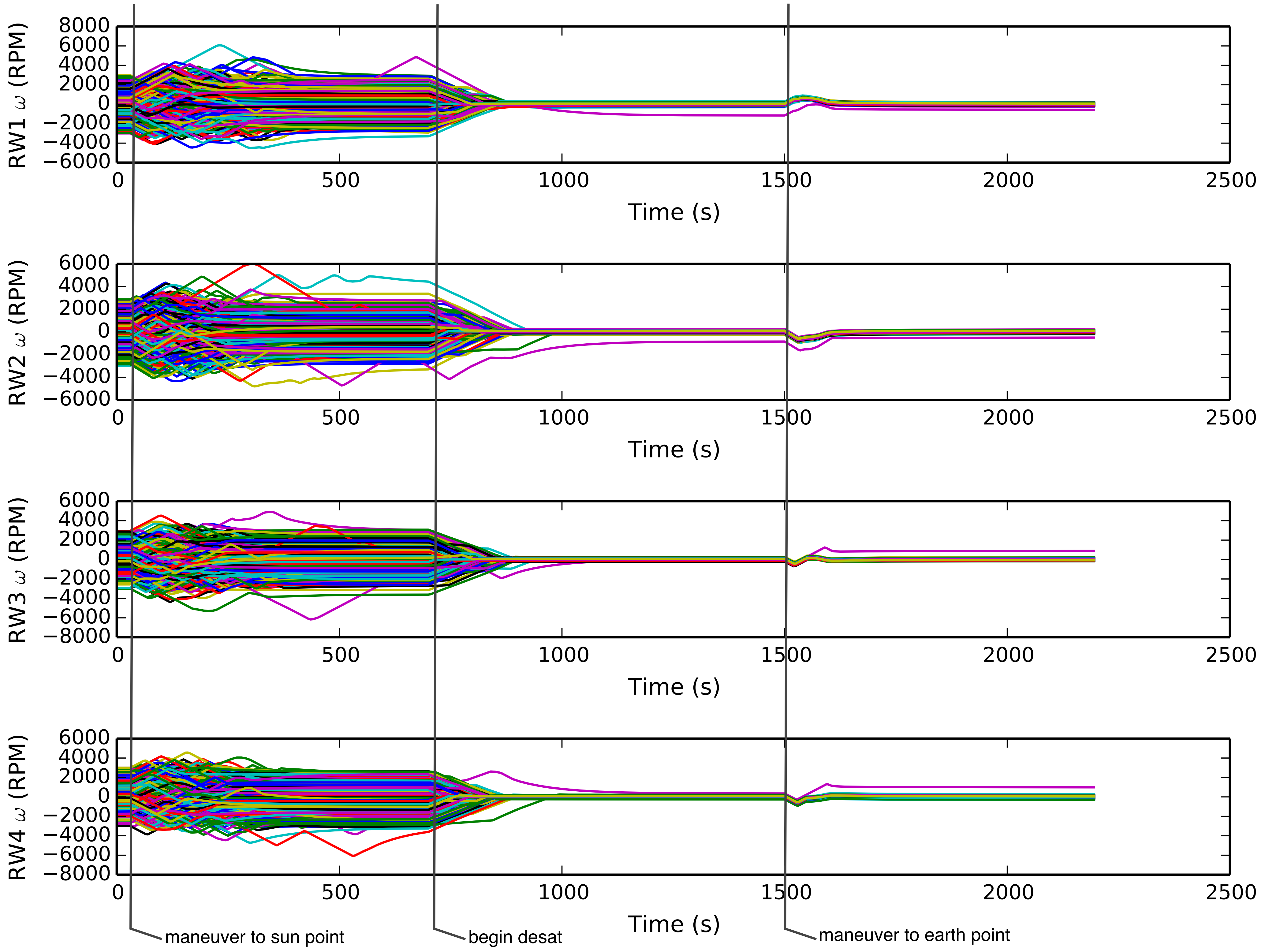




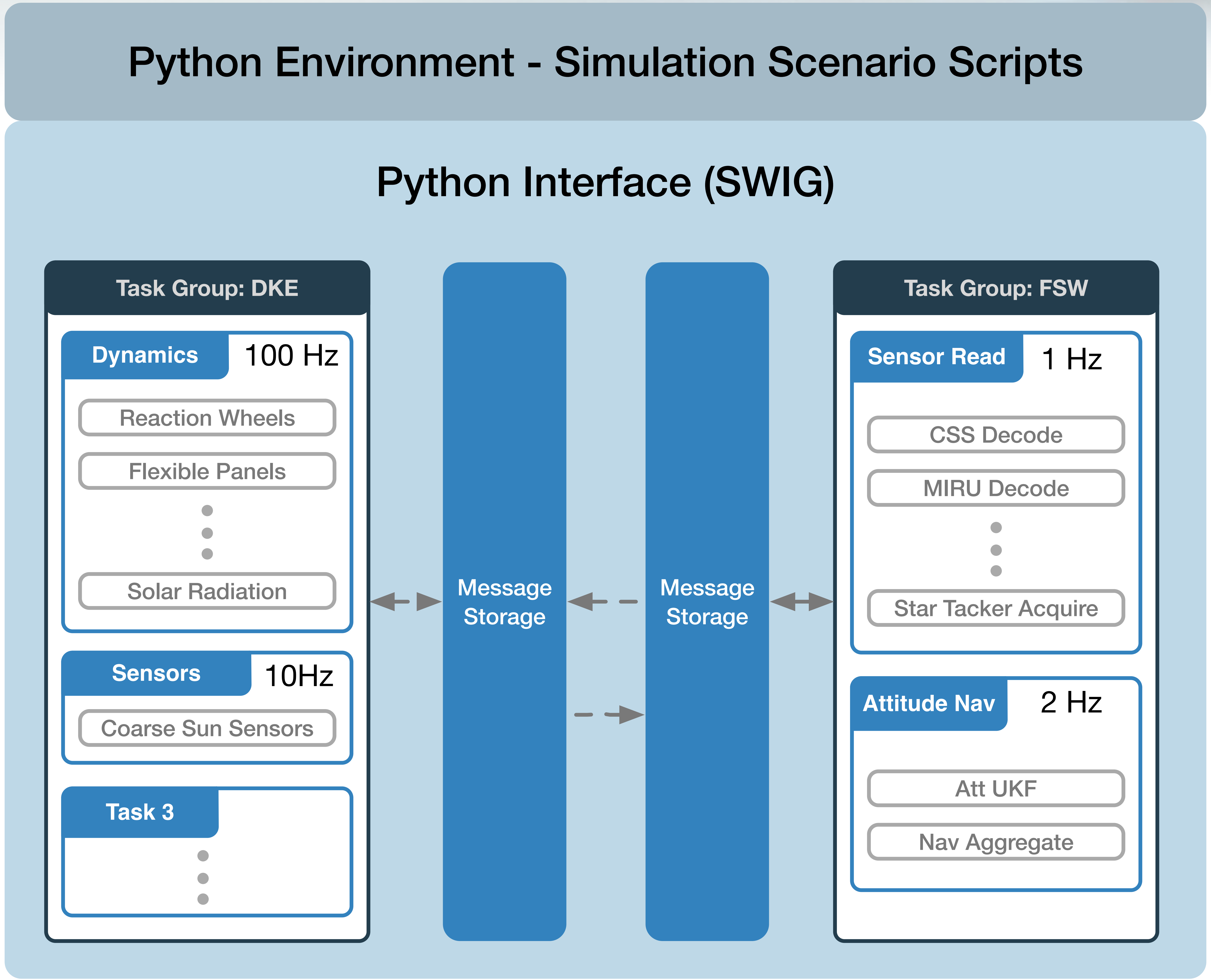
Simulation Control

- Simulation can be controlled according to spacecraft state

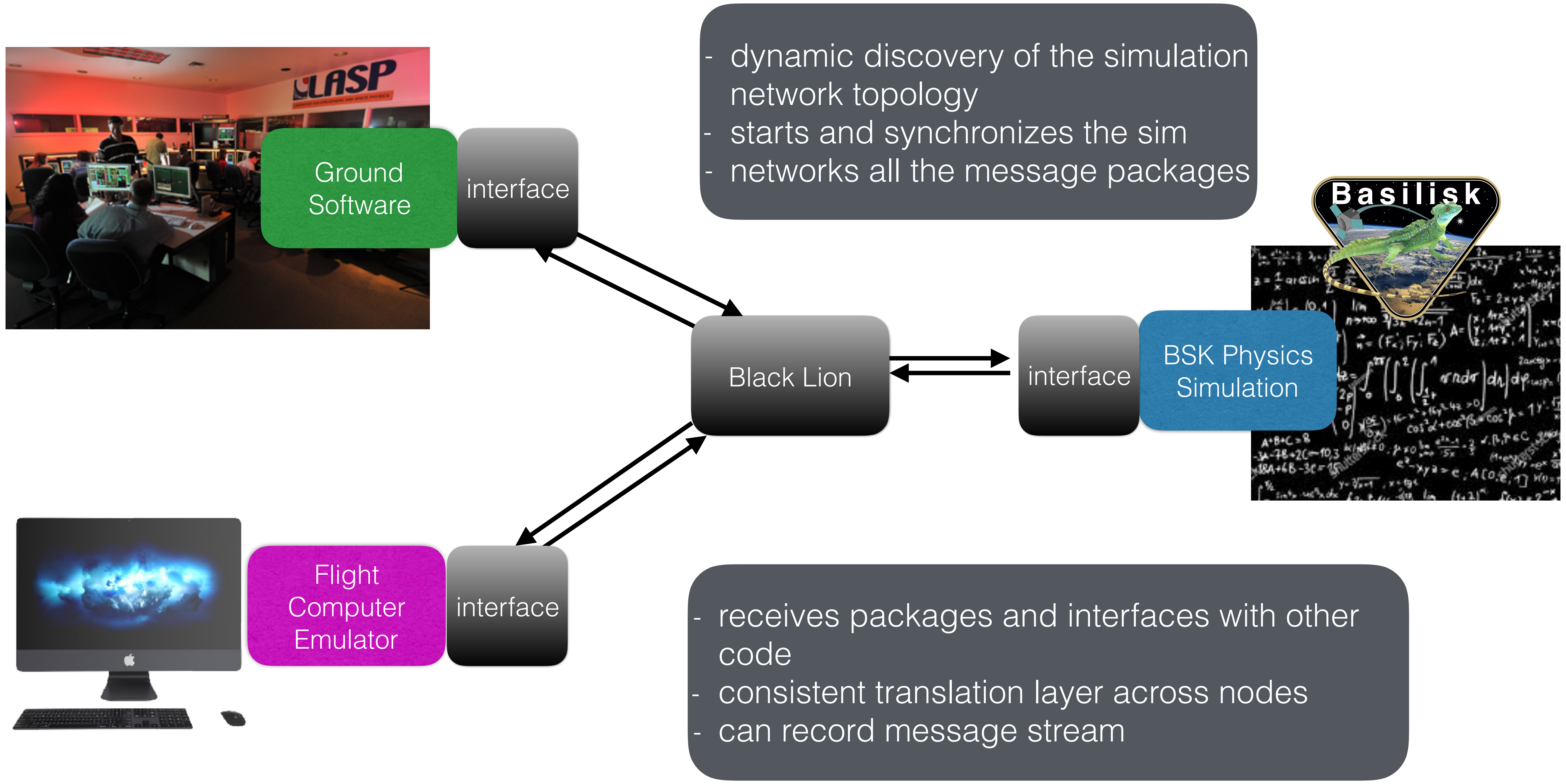
```
scSim.ConfigureStopTime(sec2nanos(20))
scSim.ExecuteSimulation()
# Command the FSW to go into safe mode and advance to ~ periapsis
scSim.modeRequest = ’safeMode’
scSim.ConfigureStopTime(sec2nanos(60))
scSim.ExecuteSimulation()
# Command the FSW to go into Nav only mode
scSim.ConfigureStopTime(sec2nanos(60 * 11 * 1 + 30)))
scSim.modeRequest = ’navOnly’
scSim.ExecuteSimulation()
```



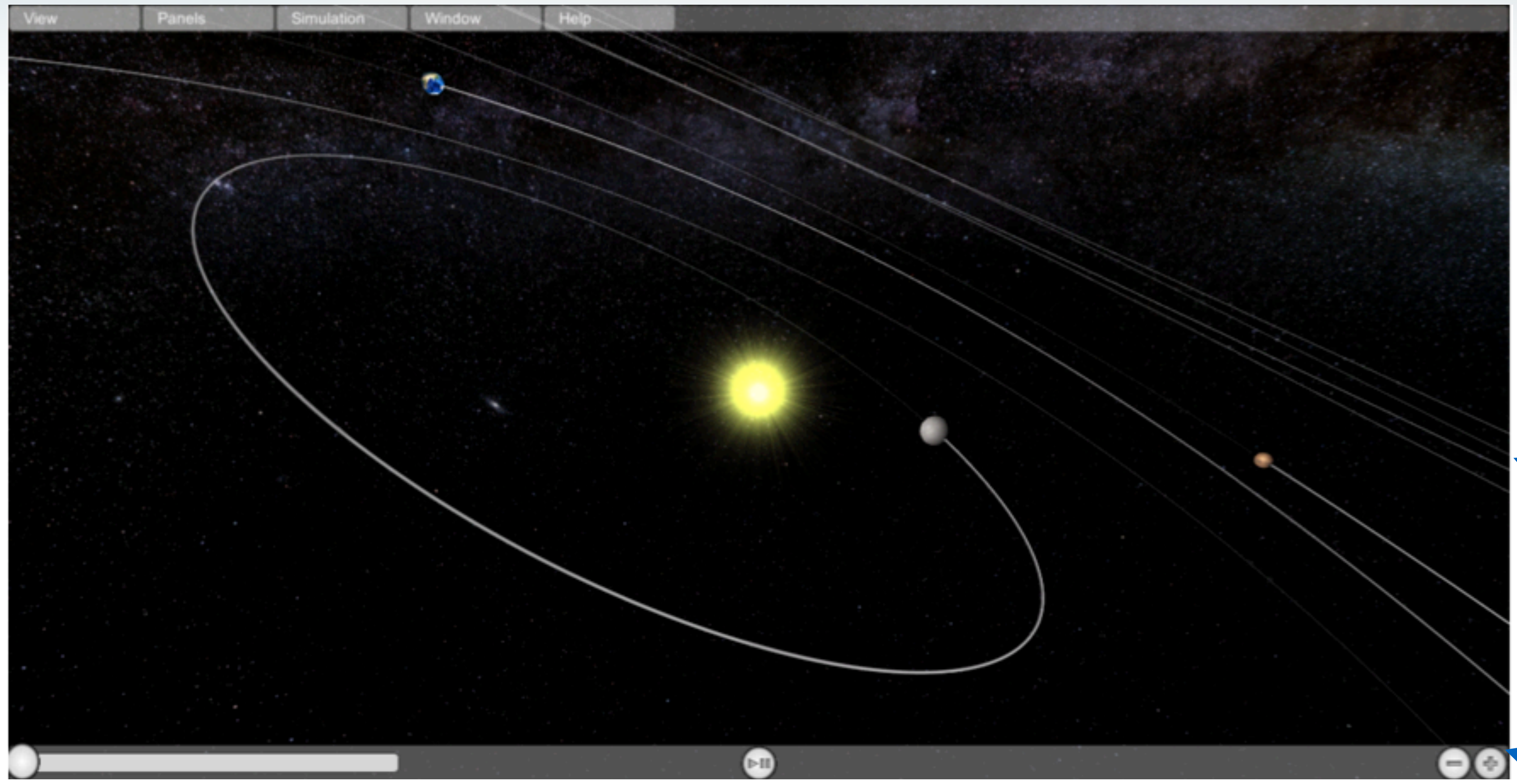
Complex Simulation Configuration



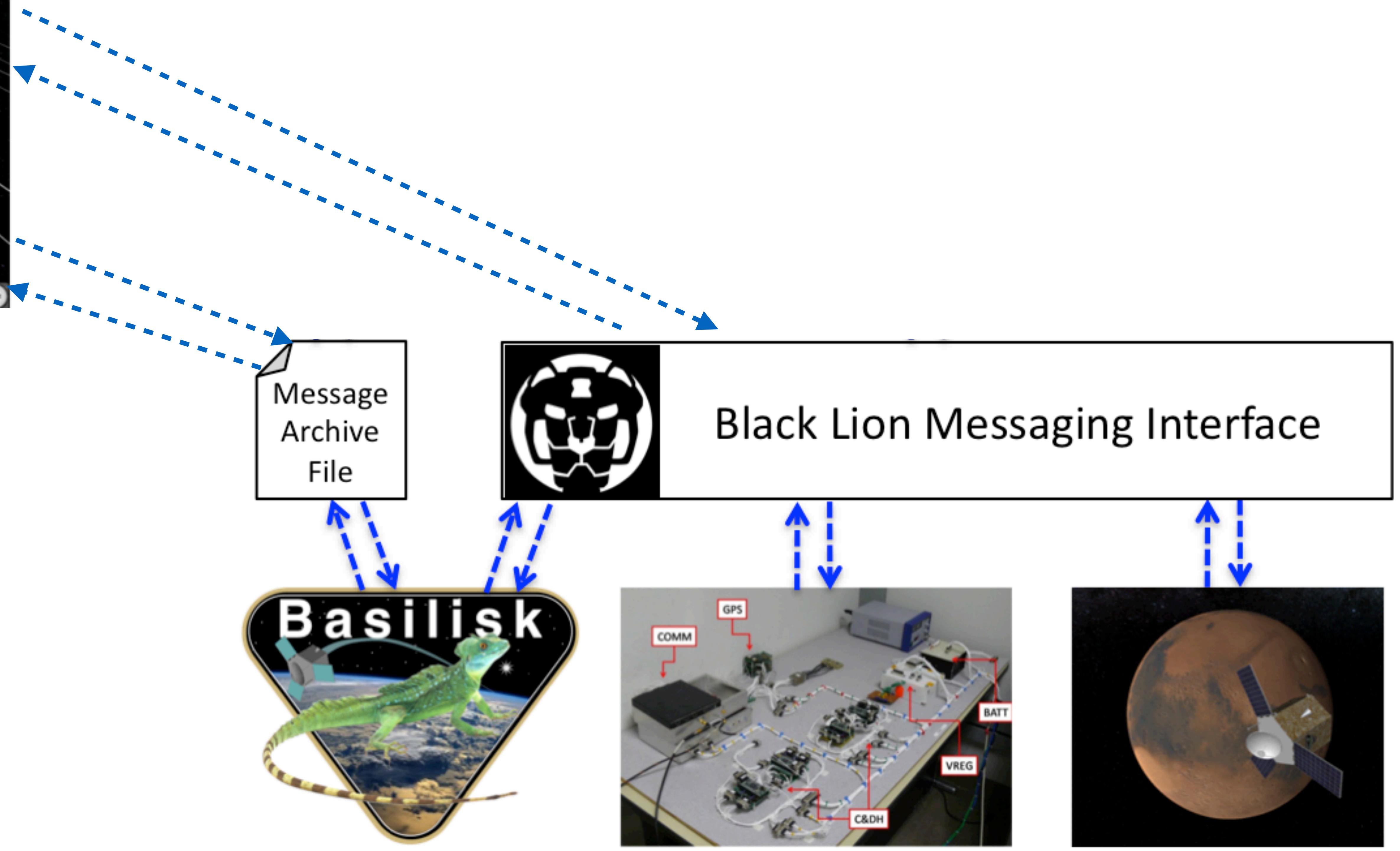
Hardware/Software in-the-loop



Visualization



Unity Visualization



Basilisk Simulations

FlatSat Test Beds

Real-Time Telemetry

Conclusions

- Basilisk's modularity provides for a wide range of spacecraft simulations
- Simulation from early feasibility to complex spacecraft FSW algorithms and dynamics analysis
- Simple simulation configuration and data analysis within the Python environment
- Currently supporting interplanetary and earth orbit missions
- Available via http://hanspeterschaub.info/bskMain.html