An End-to-End FSW Development Cycle

for an Interplanetary Spacecraft Mission

Mar Cols Margenet*, Hanspeter Schaub† and Scott Piggott‡

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

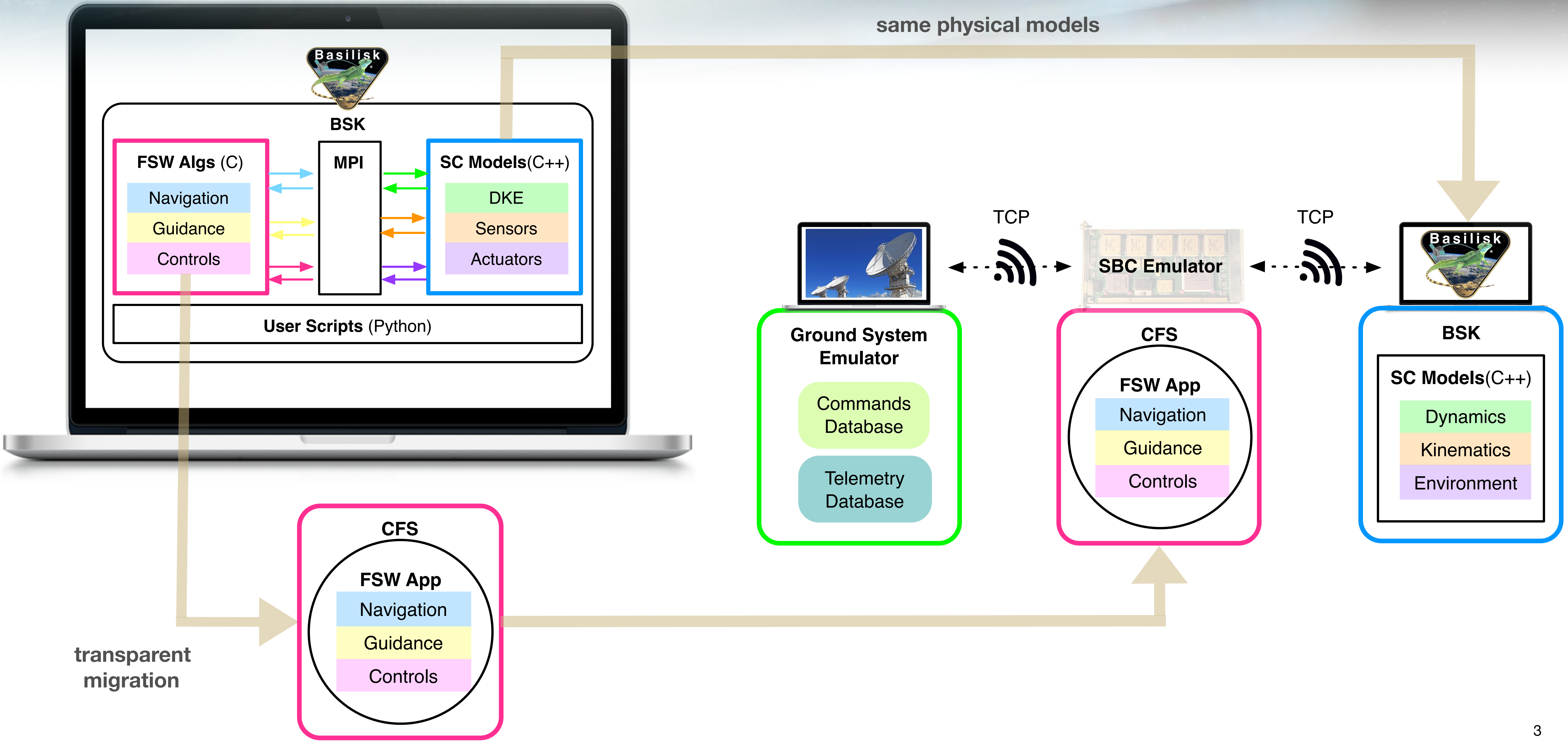
Ann and H. J. Smead Aerospace
Engineering Sciences Department
University of Colorado, **Boulder**

Outline

- **Complete FSW cycle for an interplanetary spacecraft mission**
- **Desktop algorithm design:** the Basilisk software testbed
- **Migration into the Core Flight System**
- **Embedded testing** of CFS-FSW in an emulated flat-sat

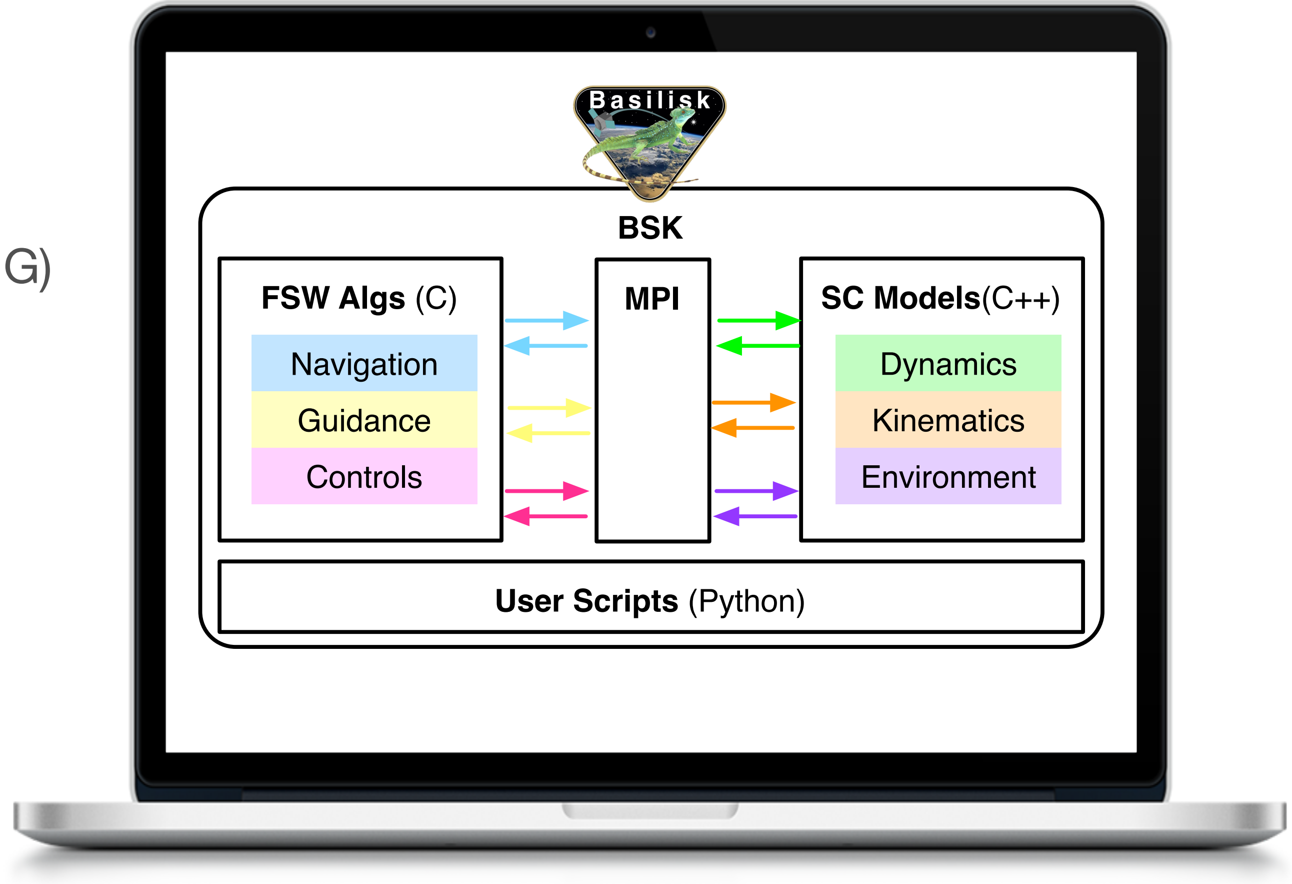
- **Interesting aspects:**
    1. **Automated yet transparent migration into CFS**
    2. **Consistent testing throughout testbeds**

Development Cycle



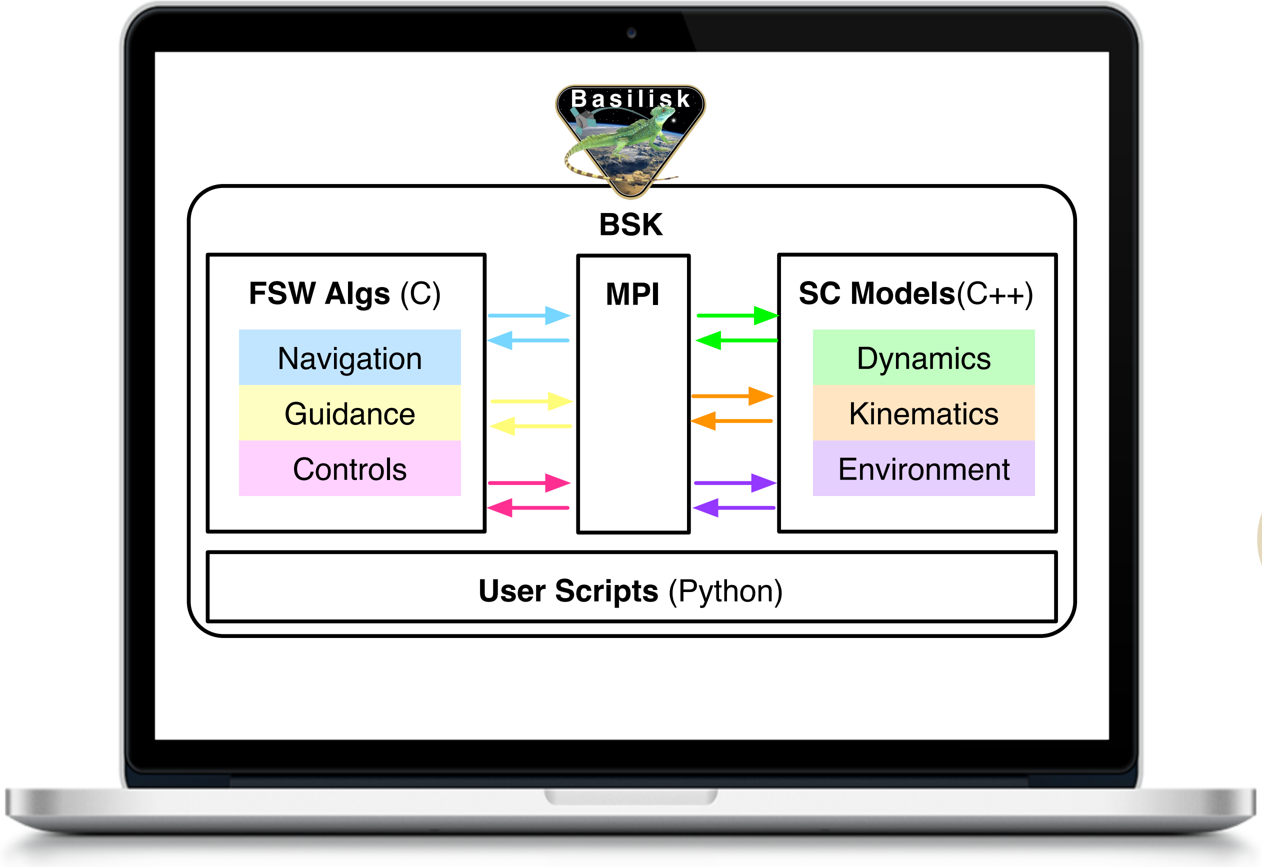
Basilisk Desktop Testbed: Overview

- **Basilisk:** open-source, cross-platform, desktop testbed
  - Designing flight algorithms
  - Testing in closed-loop dynamics
- **Language:** C/C++ source code wrapped in Python (through SWIG)
  - **Python** for: **(1)** setup, **(2)** desktop execution, **(3)** post-proc
- **Nominal configuration**:
  - **Dynamics Process**: spacecraft physical models (C++)
  - **FSW Process**: mission-specific GN&C algorithms (C)
- **Features:** modular arch. & pub-sub message passing

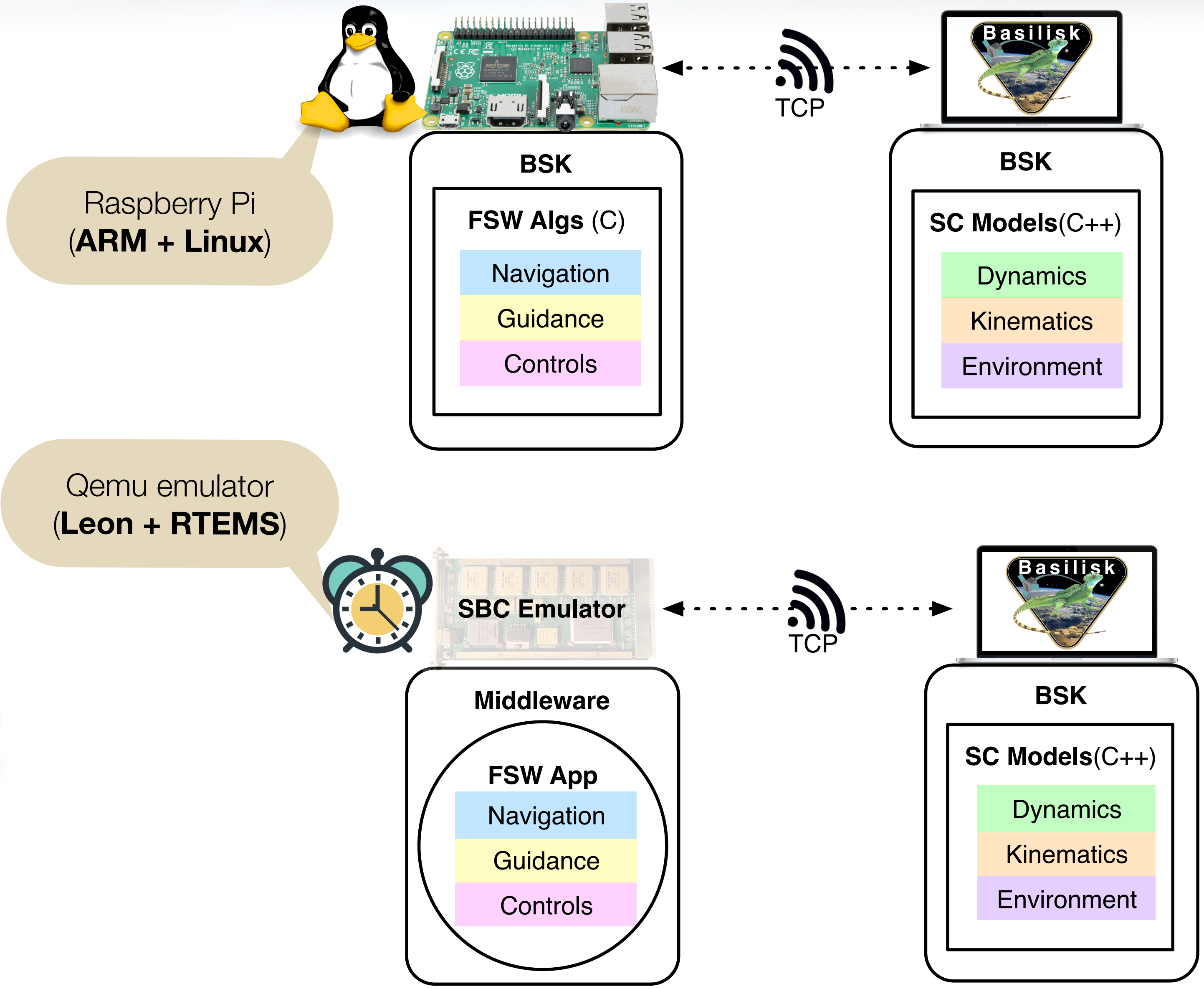


Migration of the Basilisk Flight Application

- **Single processor environment:**

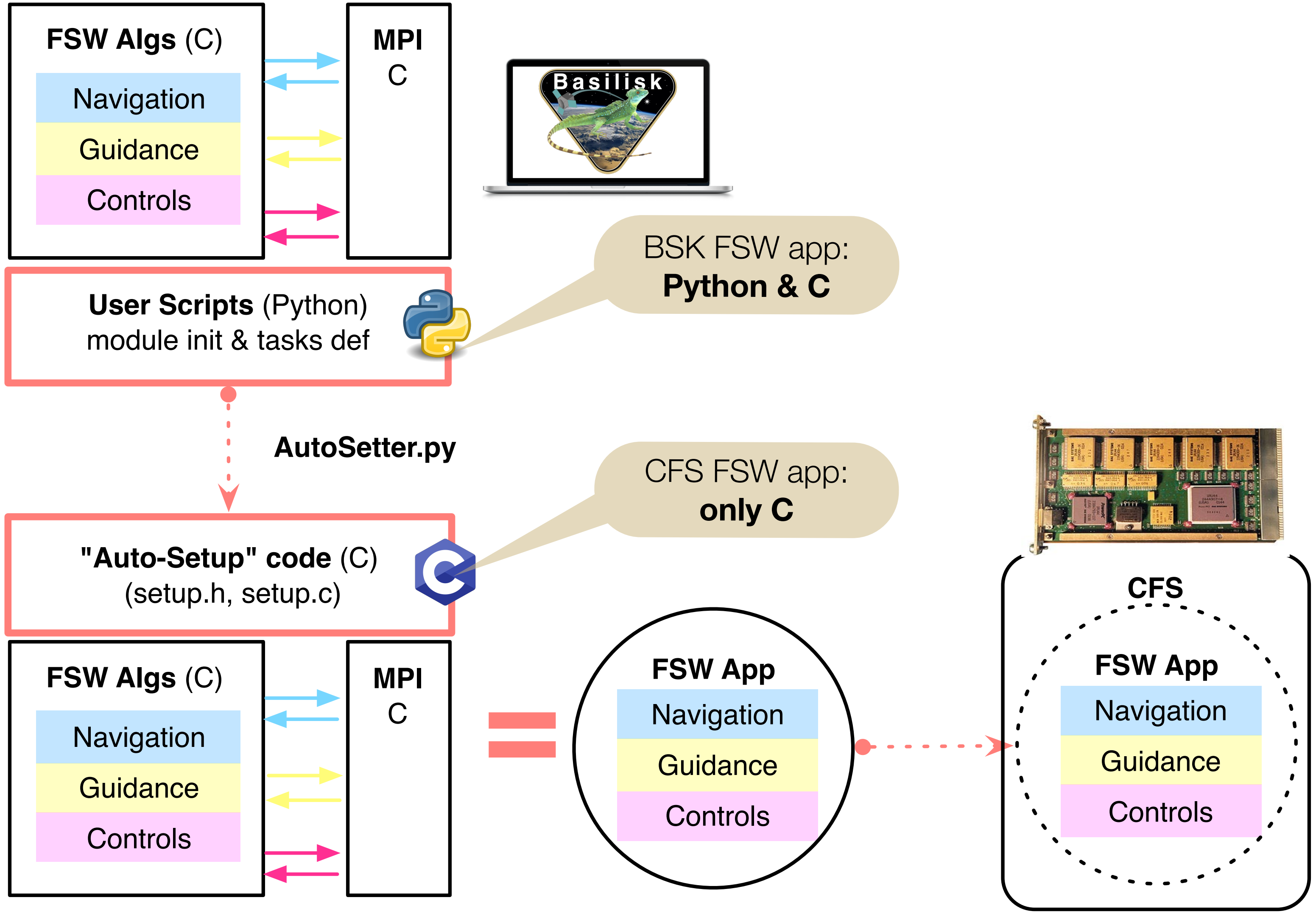


- **Multi processor environment:** realistic testing



FSW Migration: from Basilisk to CFS

- **What does it take to migrate FSW algs?**
- **Recall:** Basilisk leverages Python for FSW
  1. **Setup**
  2. Desktop execution
  3. Post-processing
- **Setup code**: translated from Python to C
  - Modules' variable initialization
  - Tasks definition (modules groups)



Python setup: C module initialization

- **Basilisk C module**: a stand-alone model or self-contained logic.
  - **Config struct**
  - **Generic algorithm calls**: self-init, cross-init, update & reset

```c
// Configuration struct
typedef struct{
    double ISCPntB_B[9]; // Inertia
    double CoM_B[3]; // Center of mass
    char outputMsgName[MAX_LENGHT]
}VehicleConfig;

// Generic algorithms
void SelfInit_vehConfigData(...);
void CrossInit_vehConfigData(...);
void Update_vehConfigData(...);
void Reset_vehConfigData(...);
```

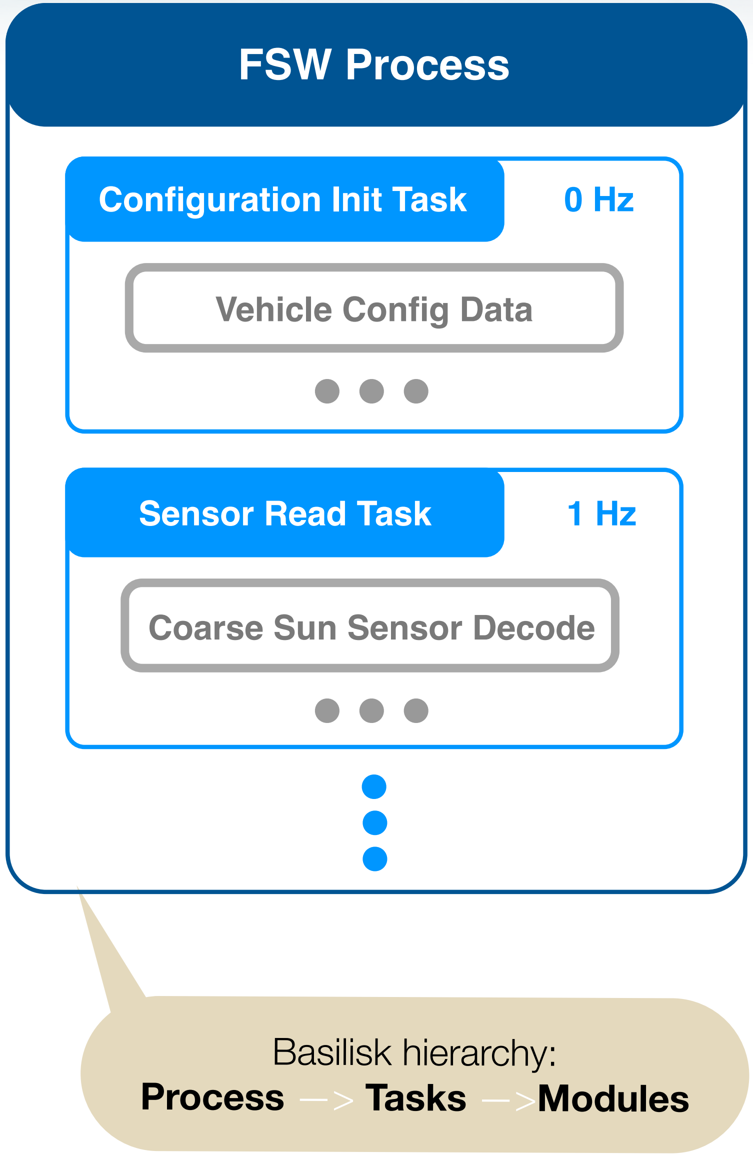
called from Python for desktop execution

SWIG

- **Python module initialization**

```python
# Instantiate C modules
self.veh_config = VehicleConfig()

def SetVehicleConfigData(self):
    # Initialize vehicle data C module
    self.veh_config.ISCPntB_B = [600.0, 0.0, 0.0, 0.0, 600.0, 0.0, 0.0, 0.0, 600]
    self.veh_config.CoM_B = [0.0, 1.0, 0.0]
    self.veh_config.outputPropsName = "adcs_config_data"
```

Python setup: task groups & rates



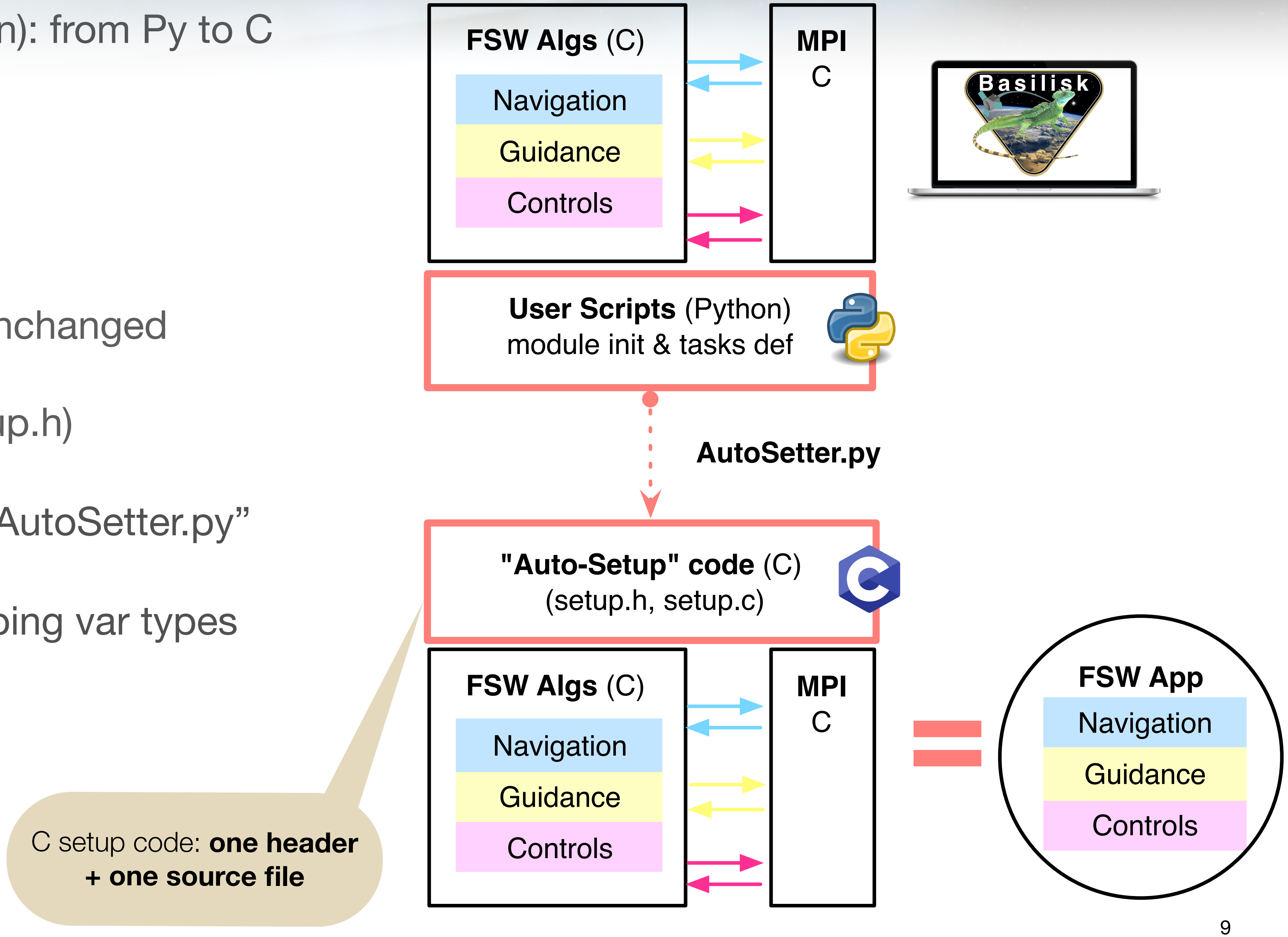
- **Define tasks** at certain rates
- **Modules added into each to task**
- **Example:** init-only task contains vehicle config data module

```python
# Instantiate C modules
self.veh_config = VehicleConfig()

def SetVehicleConfigData(self):
    # Initialize vehicle data C module
    self.veh_config.ISCPntB_B = [600.0, 0.0, 0.0, 0.0, 600.0, 0.0, 0.0, 0.0, 600]
    self.veh_config.CoM_B = [0.0, 1.0, 0.0]
    self.veh_config.outputPropsName = "adcs_config_data"
```

```python
# Create FSW process
self.fswProc = self.CreateNewProcess("FSWProcess", 100)

def populateFSWTasks(self):
    # Create tasks and add modules
    self.fswProc.addTask(self.CreateNewTask("initOnlyTask", int(0)), 200000)
    self.AddModelToTask("initOnlyTask", self.veh_config, 1)
```

Pure-C FSW Application

- **Setup code** (module init & tasks definition): from Py to C
- **Key remark**:
    - Only setup code is translated
    - FSW algs. source code: remains unchanged
- **Pure-C FSW**: FSW Algs + (setup.c + setup.h)
- **Automatic translation of setup code**: “AutoSetter.py”
    - Transparent & open template mapping var types
    - Resulting C setup code: minimal
- **Trick**: Python introspection



Python Introspection: AutoSetter

User Scripts
module init & tasks def

AutoSetter.py

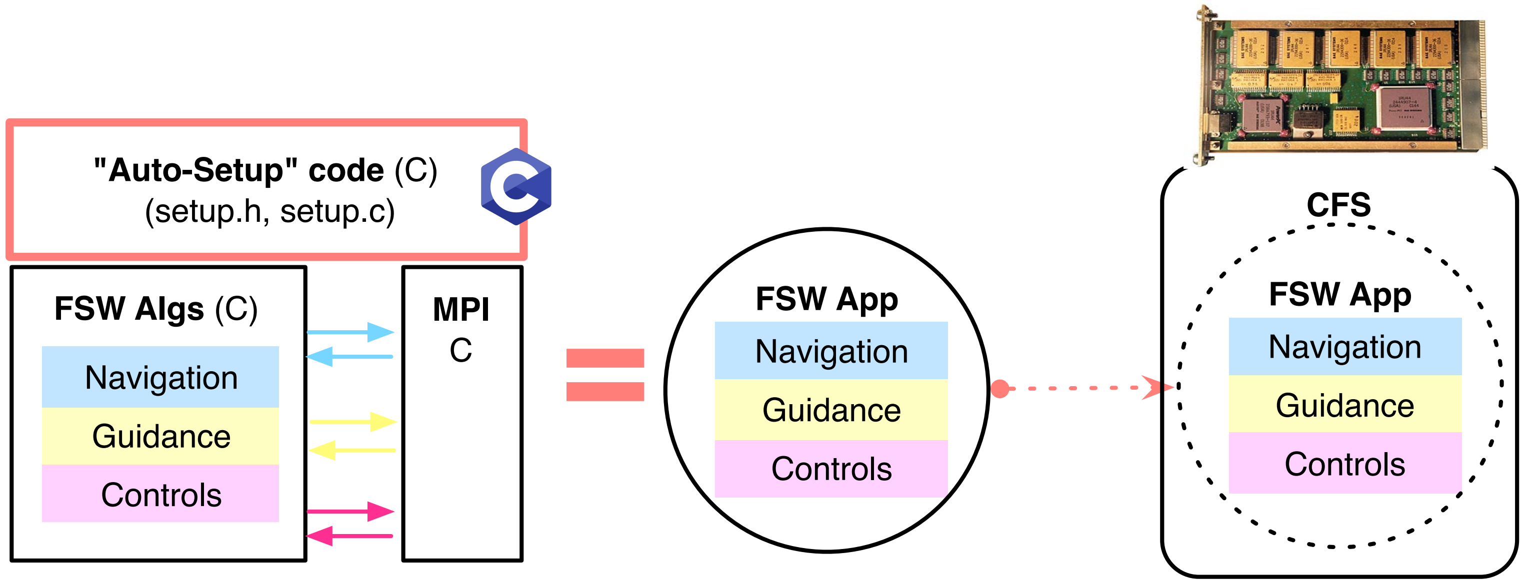
Setup Code
(setup.h, setup.c)

I am a model.
I've these **variables**
and **algorithm** names.

```python
# Instantiate C modules
self.veh_config = VehicleConfig()

def SetVehicleConfigData(self):
    # Initialize vehicle data C module
    self.veh_config.ISCPntB_B = [600.0, 0.0, 0.0, 0.0, 600.0, 0.0, 0.0, 0.0, 600]
    self.veh_config.CoM_B = [0.0, 1.0, 0.0]
    self.veh_config.outputPropsName = "adcs_config_data"
```

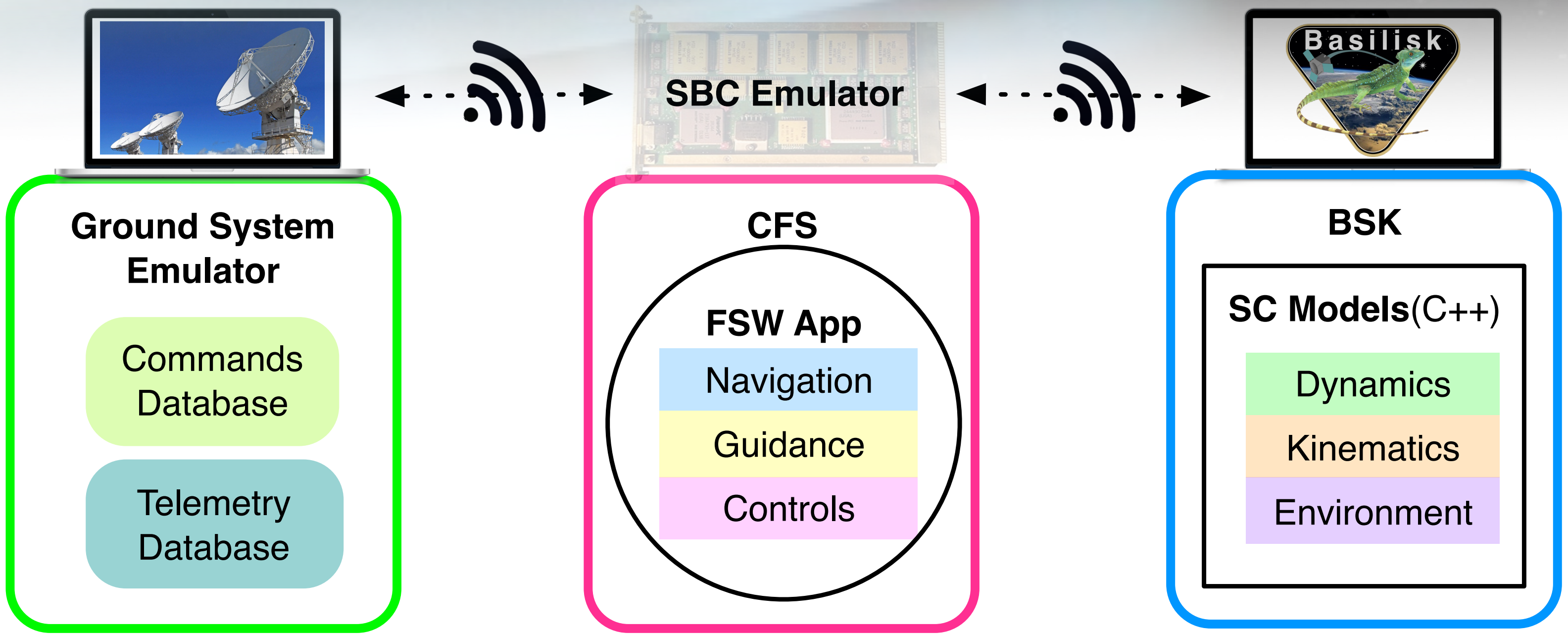
```c
typedef struct{ // Struct with all FSW modules
        VehicleConfig veh_config;
        // [...] More modules below
} AllConfig;
void AllConfig_DataInit(AllConfig *data){ // Modules initialization
        memset(data, 0x0, sizeof(AllConfig));
        // VehicleConfig module init
        data->veh_config.ISCPntB_B[0] = 600.0;
        data->veh_config.ISCPntB_B[4] = 600.0;
        data->veh_config.ISCPntB_B[8] = 600.0;
        strcpy(data->veh_config.outputMsgName, "adcs_config_data");
        // [...] More modules below
}
```

Embedded FSW Dev & Testing

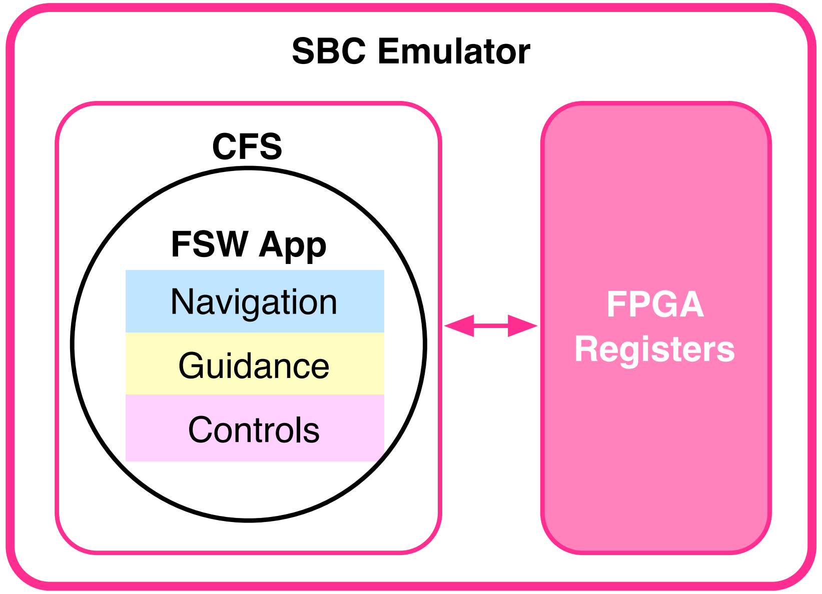


Embedded Testing: CFS

- CFS-FSW testing in **emulated flat-sat**

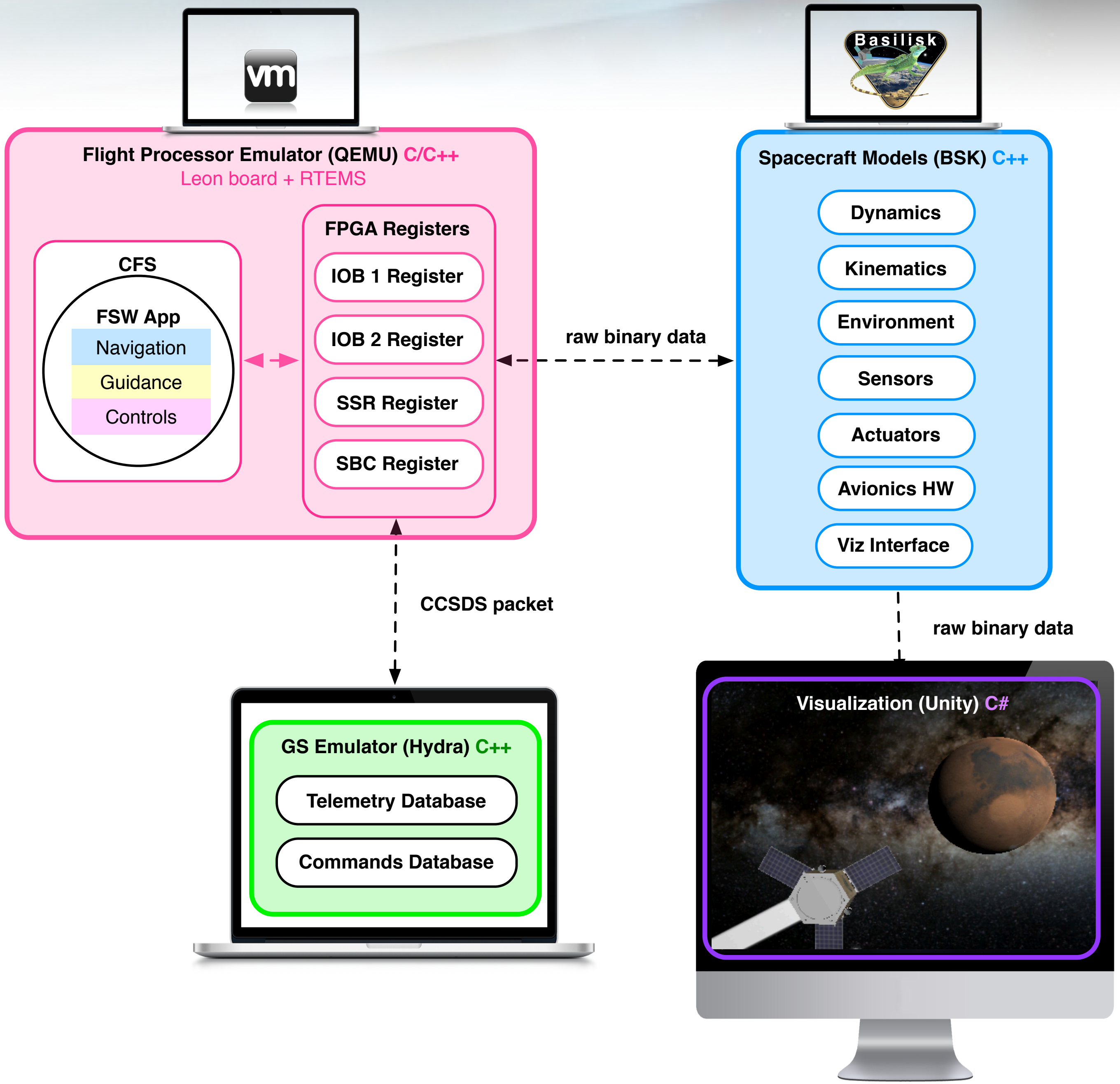


- **Interaction** with embedded FSW **becomes tricky**
- **FPGA registers**: memory map for I/O of bin data

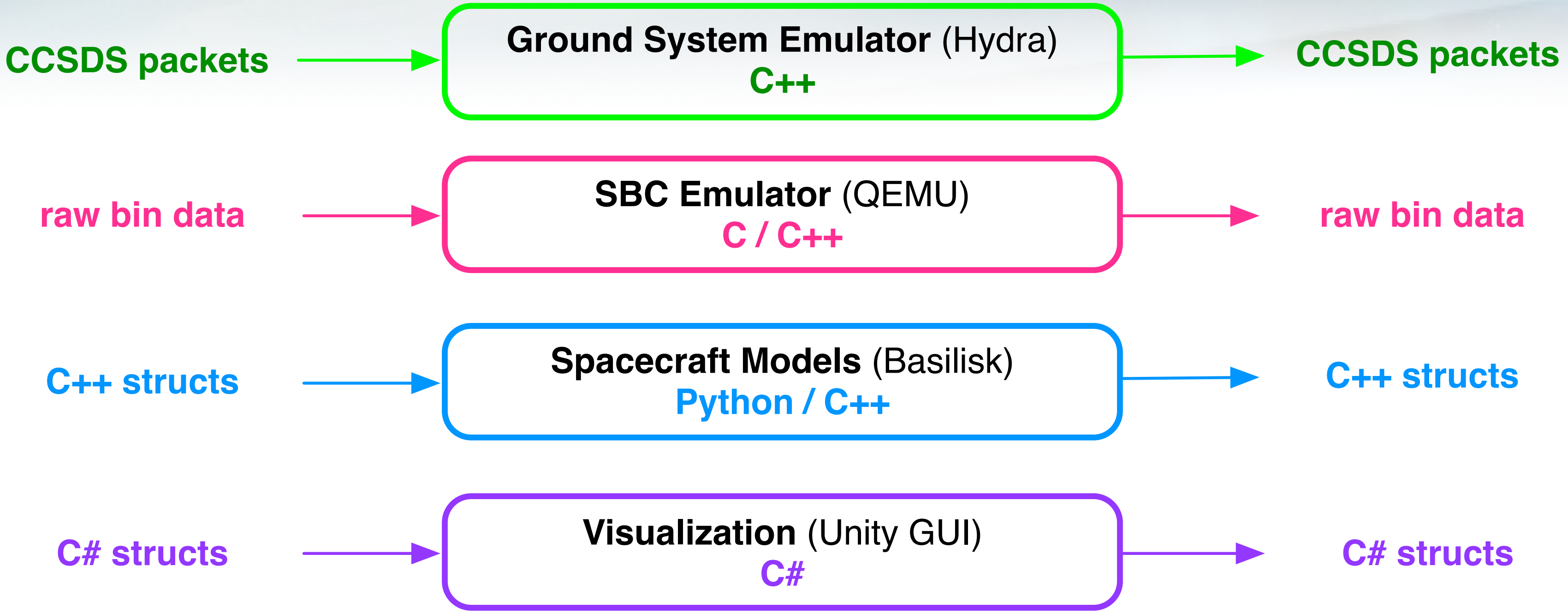


Emulated Flat-Sat Models

- **Flight Processor Emulator**: **QEMU**
  - Virtual Leon3 + RTEMS
  - CFS-FSW app + FPGA registers
- **Spacecraft Models**: **Basilisk**
  - Dynamics, Kinematics, Environment
  - Sensors, Actuators, Avionics HW
- **Ground System Emulator**: **Hydra**
  - Command & Telemetry
- **Visualization**: **Vizard**
  - Unity-based GUI



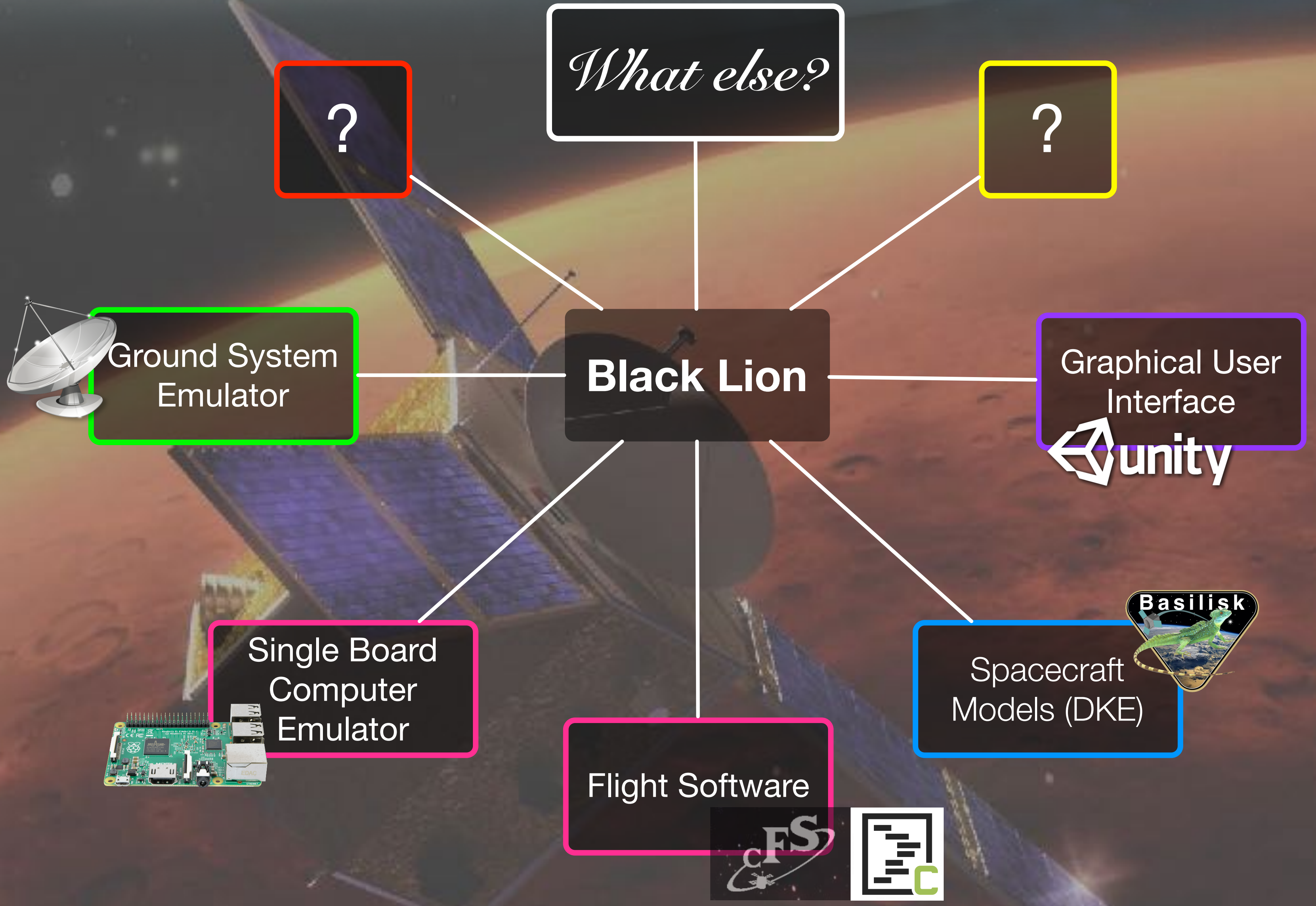
Models heterogeneity



stand-alone apps never designed to work together…

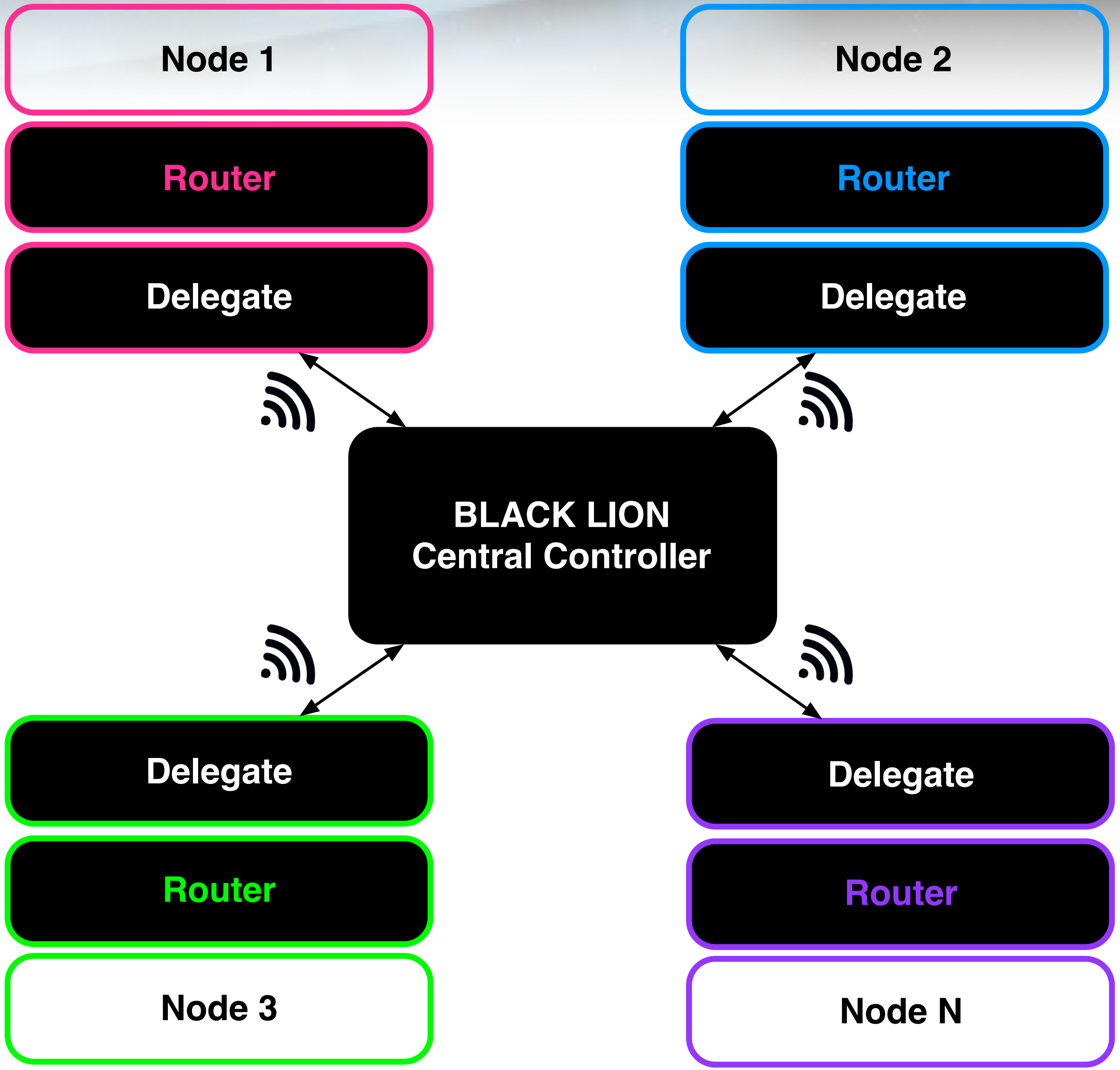
- Written in **different programming languages**
- **Different execution speeds** (asynchronous vs. synch, faster vs. slower than RT)
- **Multi-threaded** vs. **single-threaded** (important for sockets)
- **Different endianness** (specially tricky)

The Black Lion Communication Architecture



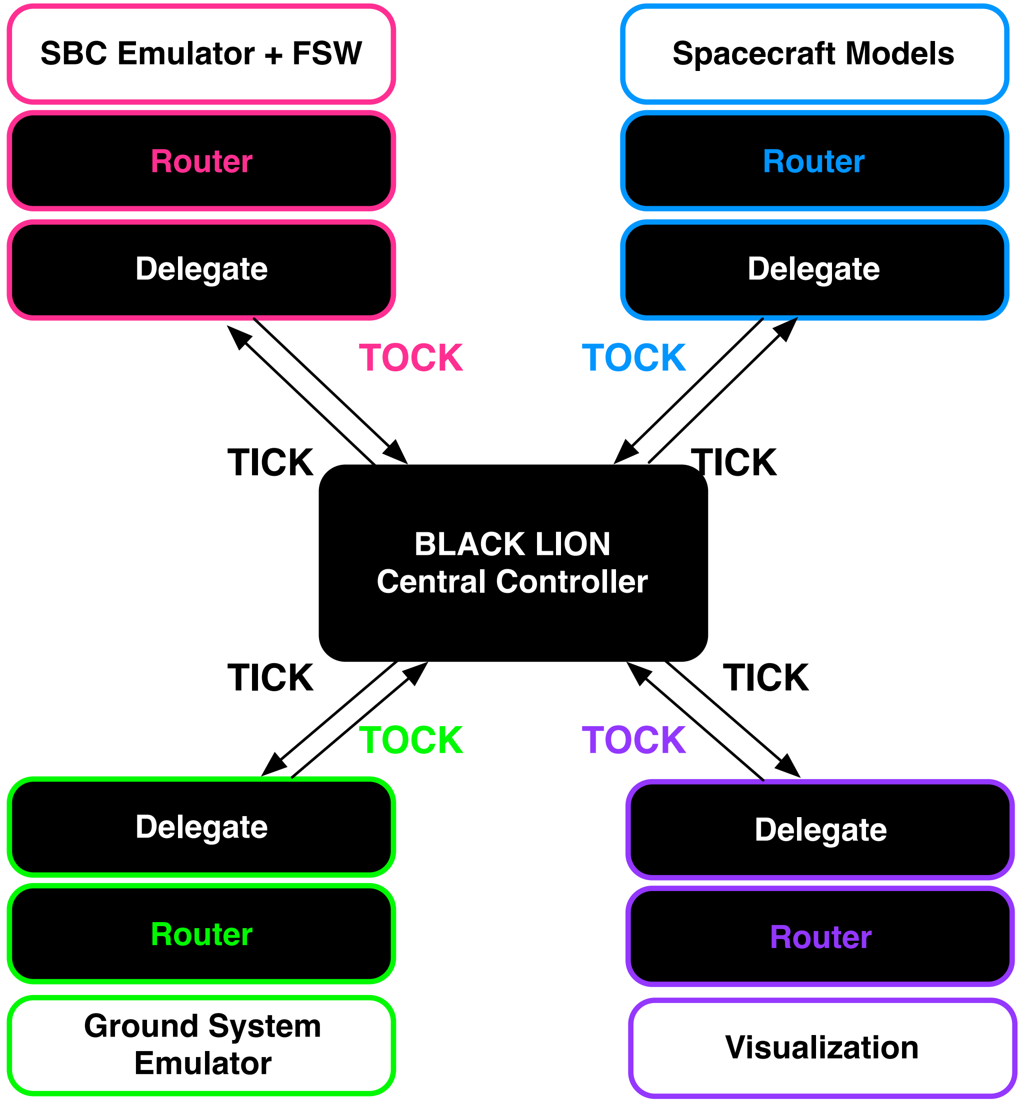
Black Lion: Architecture Overview

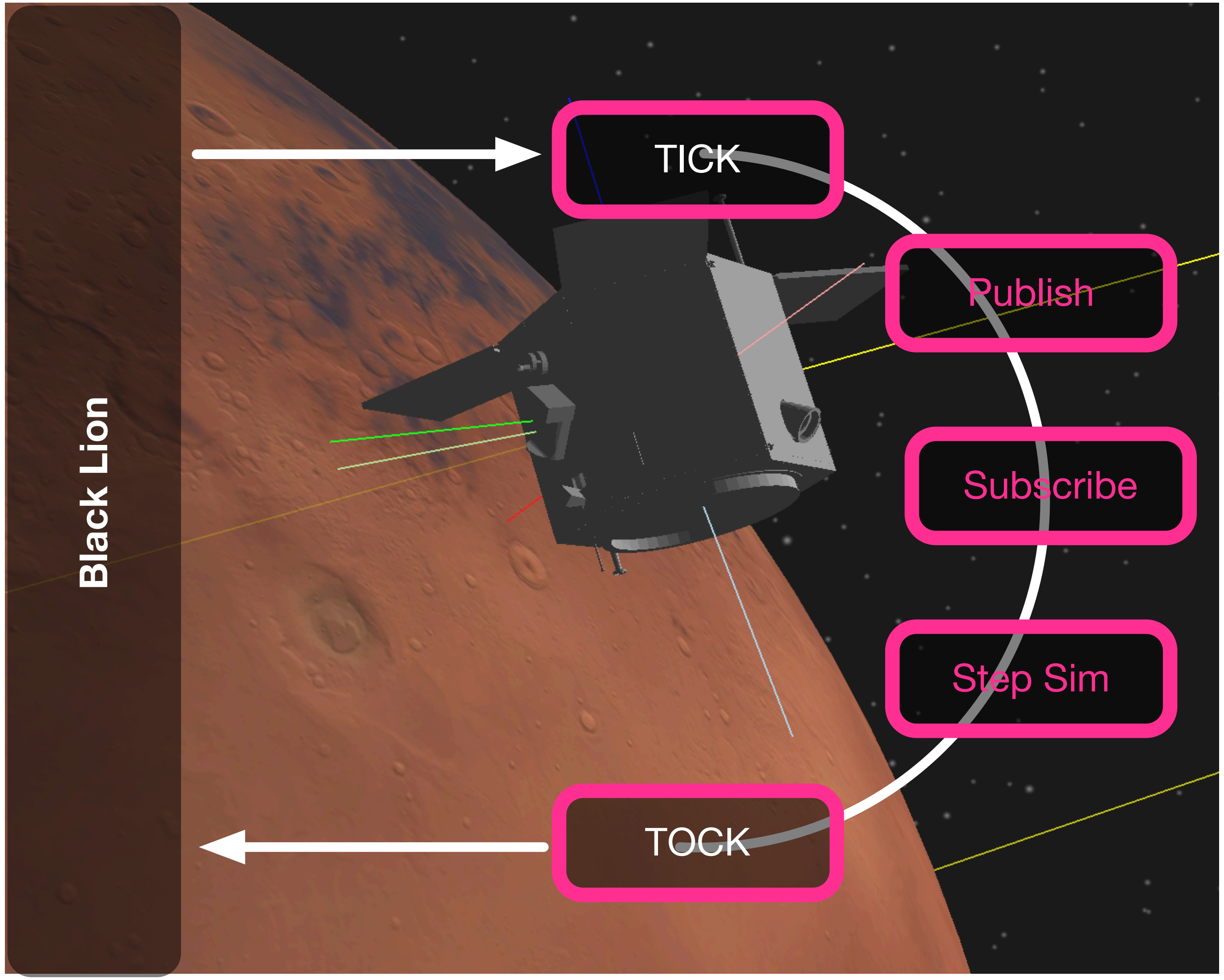
- **Communication Goals**
  1. **Transport** of binary data
  2. **Serialization** of binary data
  3. **Synchronization** of nodes/components
  4. **Dynamicity** in the connections map
- **While** remaining abstracted from each node
  - **Central Controller:** msg broker & synch master
  - **Delegate API:** sockets & connections
  - **Router API:** route data in & out of node



Black Lion: Synchronization

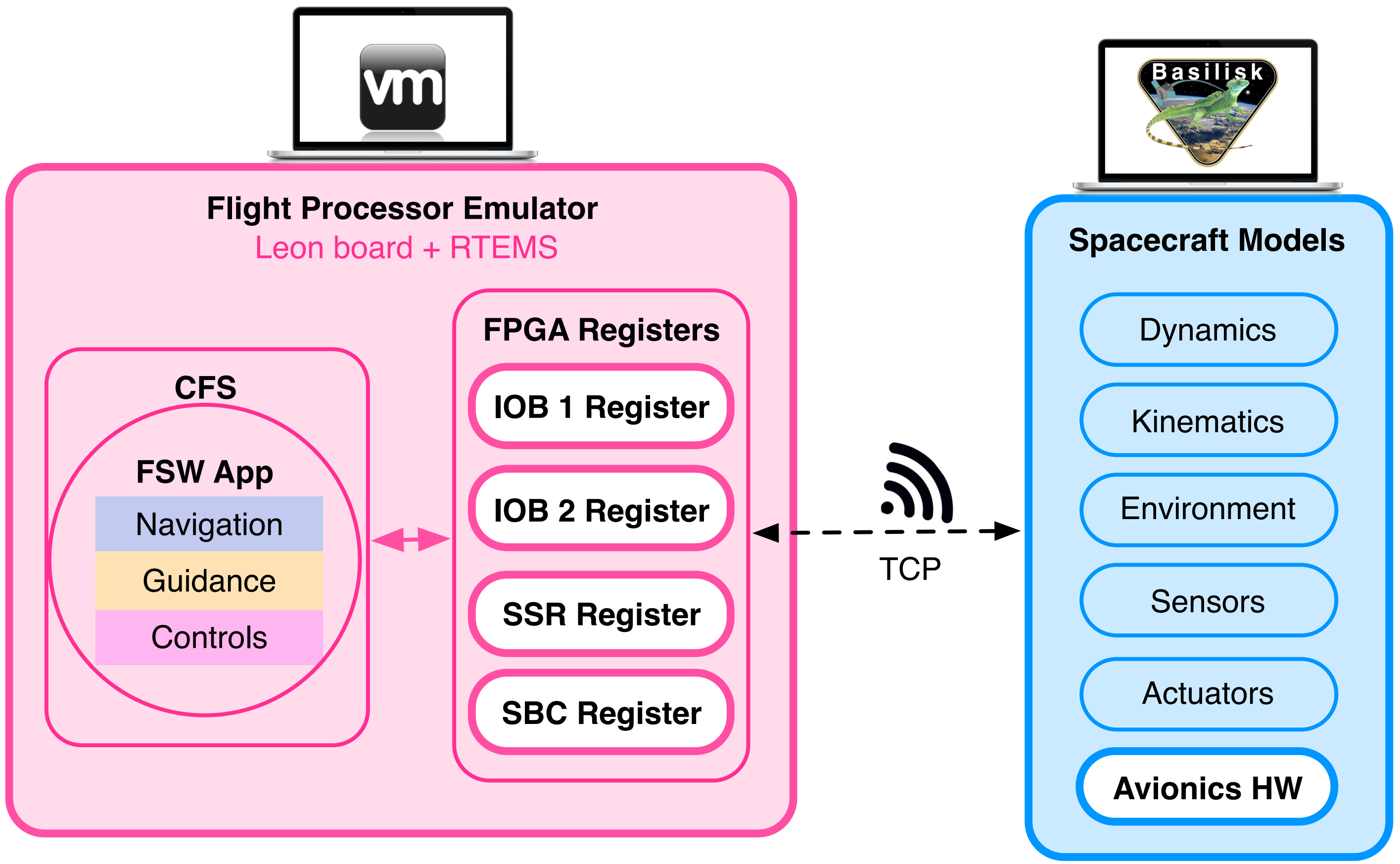
- **"Tick-Tock":** maintain all the nodes in lock-step



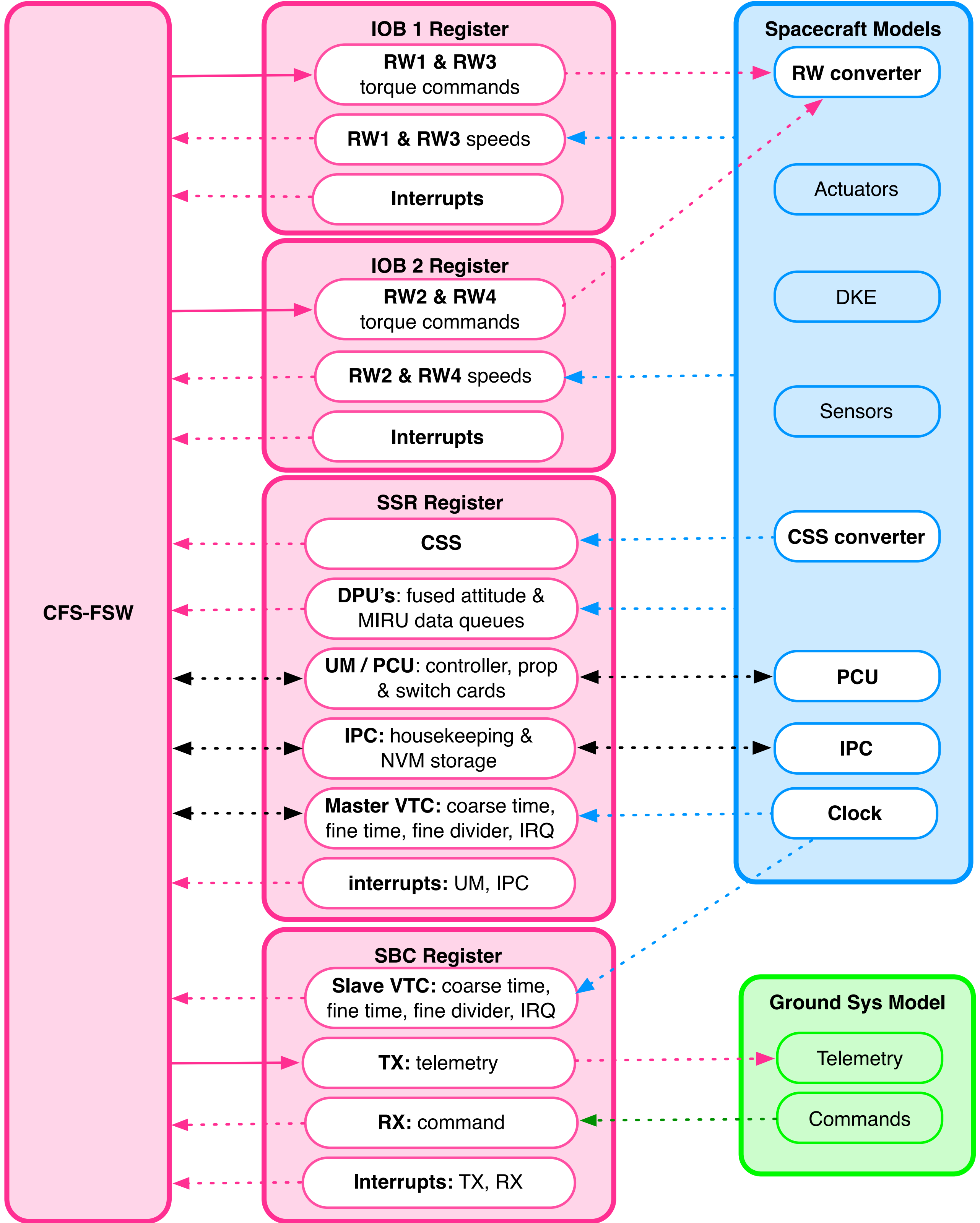


CFS-FSW Interaction: FPGA + Avionics HW

- **FPGA**: 4 different register boards
  - Each register has a memory buffer
  - Shared FSW states are mapped (snorkels)
- **FSW reads/writes in HW-like fashion**
  - Board interrupts are also replicated
- **Avionics HW models**:
  - Leverage complex functionality
  - **PCU**: avionics cards (ctrl, switch, prop)
  - **IPC**: NVM commands & HK packets

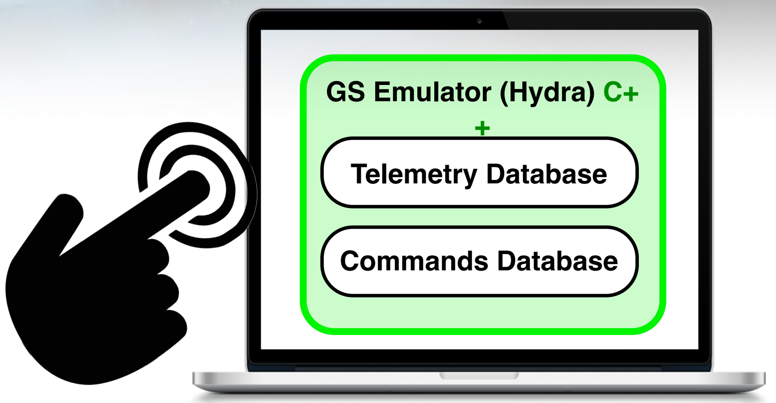


- **Register snorkels**:
  - **Actuators**: RW cmd & speeds
  - **Sensors**: CSS, ST
  - **Avionics**: PCU, IPC
  - **Clocks**
  - **Command** & **telemetry**
- **Heterogeneity challenges**
  - Unidirectional vs. bidirectional
  - Packet &/ descriptor addresses
  - Single word packets vs. queues
  - Fixed-sized vs. variable-sized
  - Add/remove byte headers
  - Endianness handling

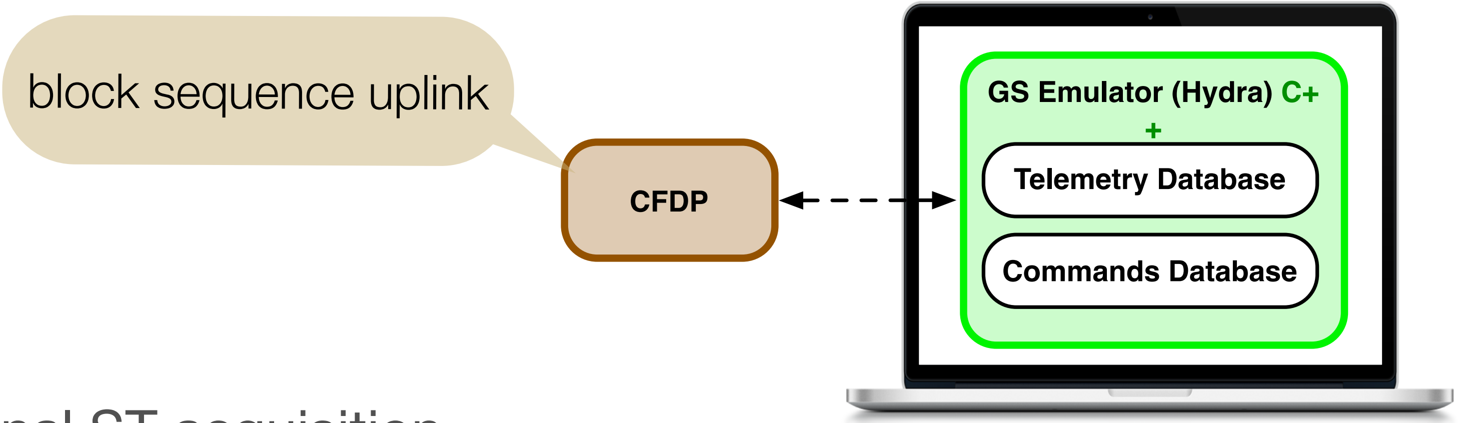


Numerical Closed-Loop Simulations

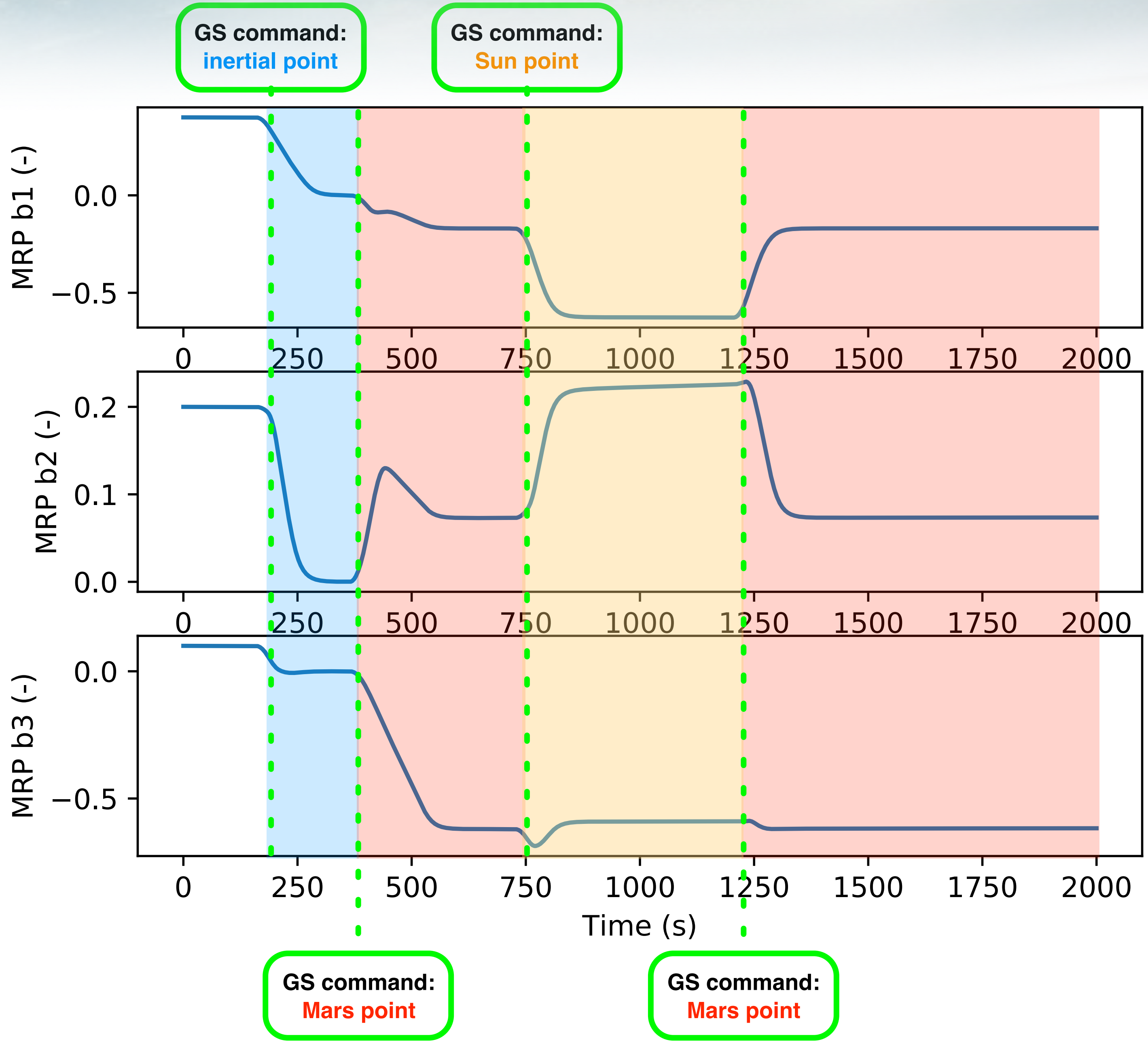
1. **Spacecraft pointing manoeuvres** manually triggered by the user
    - **Monitoring** Command + **Inertial Pointing** Command
    - **Ephemeris Correlation** Command + **Mars Pointing** Command
    - **Sun Pointing** Command
    - **Mars Pointing** Command

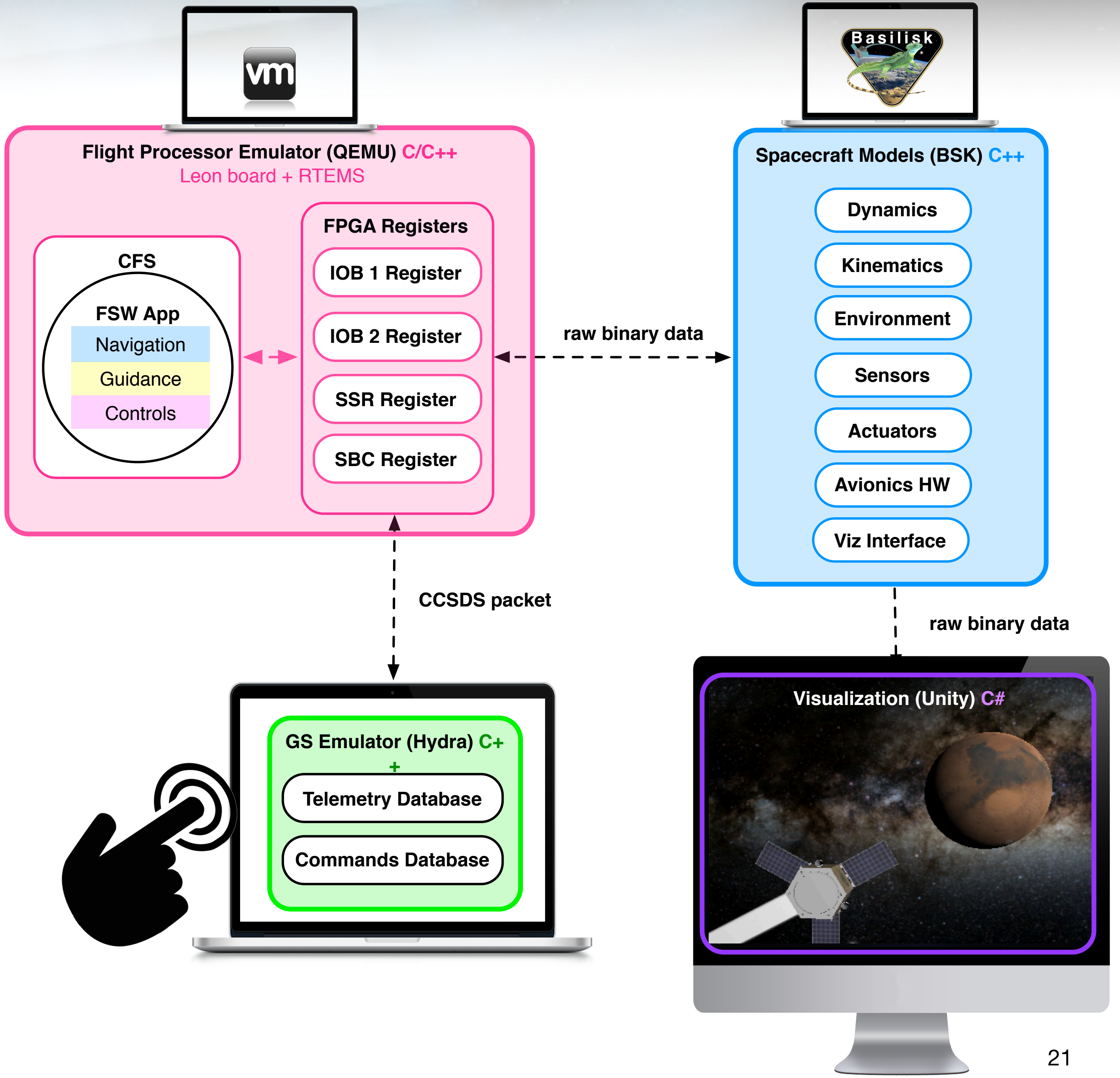


2. **Mars orbit insertion**
    - **Time jam**: bring SC right before MOI
    - **DV burn**: propulsion cards within the PCU
    - **Off-nominal testing**: FSW resets + off-nominal ST acquisition

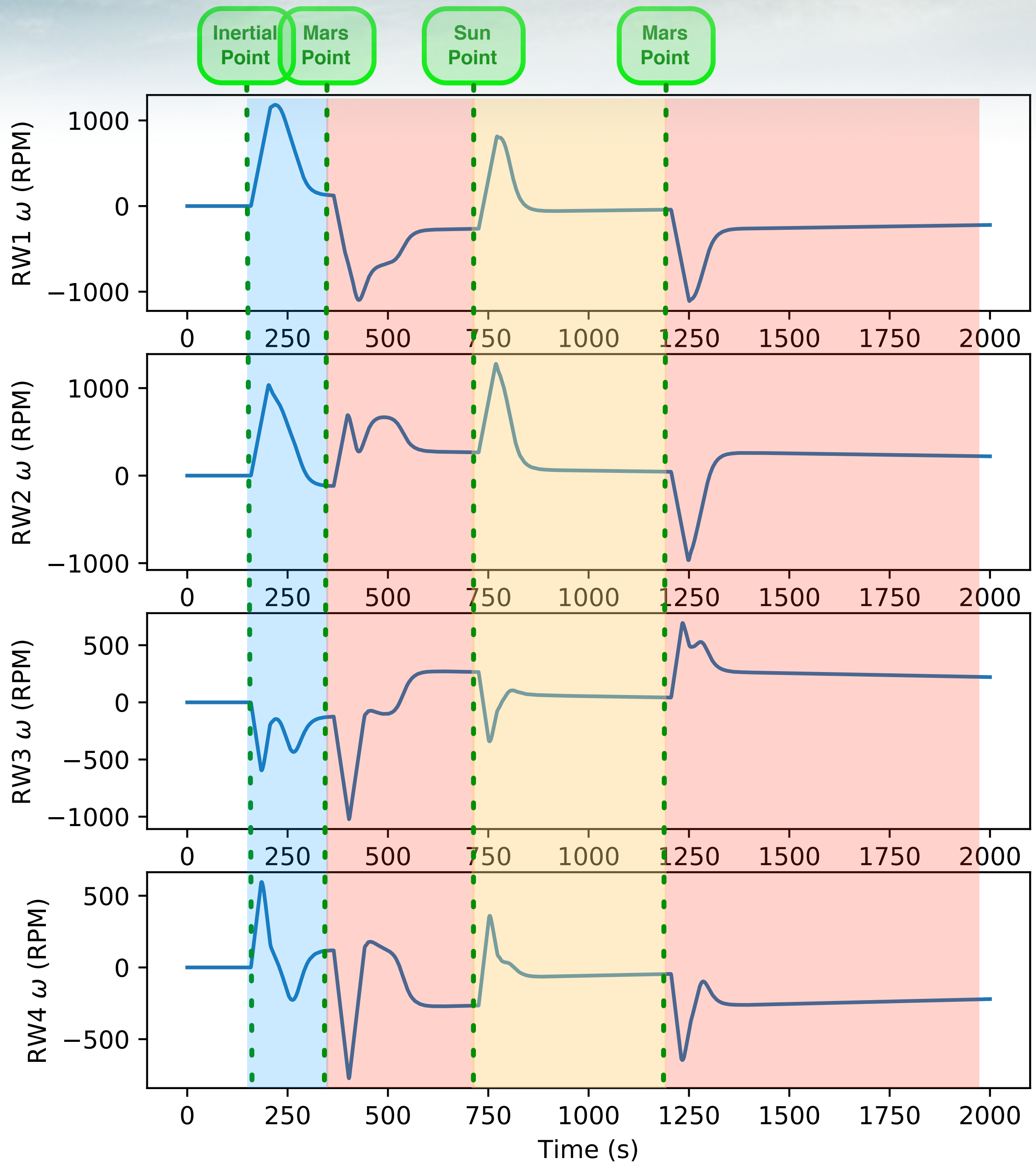


1. Spacecraft Pointing Manoeuvres

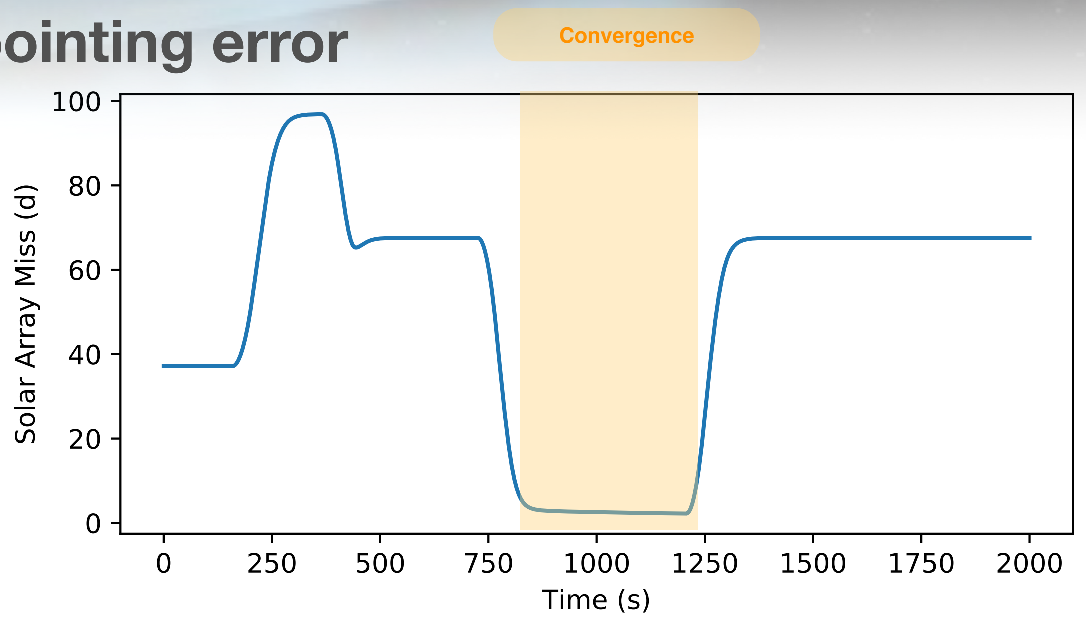




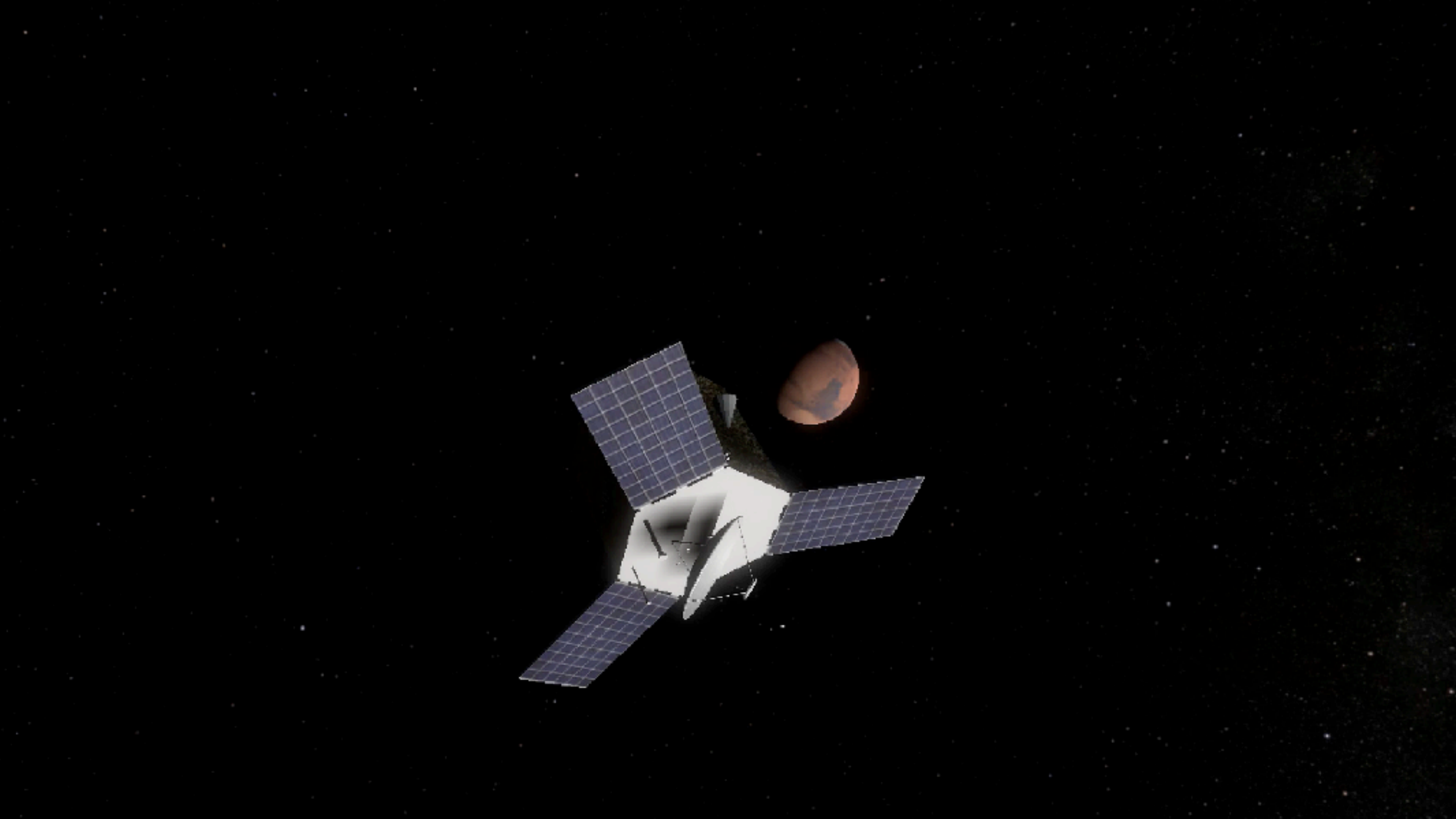
1. Spacecraft Pointing Manoeuvres



- **Sun pointing error**



- **Mars pointing error**



Conclusions

- **Complete FSW cycle for an interplanetary spacecraft mission**
- **Desktop algorithm design**: the Basilisk software testbed
- **Migration into the Core Flight System**
- **Embedded testing** of CFS-FSW in an emulated flat-sat

Thanks for listening!